MotoHUD:
Intelligent and Safe Navigational Data Presentation for Motorcyclists

By
Drew Bentz
William Budney

Senior Project

COMPUTER ENGINEERING DEPARTMENT
California Polytechnic State University
San Luis Obsipo
Spring 2013
# Table of Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Acknowledgements</td>
<td>4</td>
</tr>
<tr>
<td>Abstract</td>
<td>5</td>
</tr>
<tr>
<td>I. Introduction</td>
<td>6</td>
</tr>
<tr>
<td>II. Background</td>
<td>7</td>
</tr>
<tr>
<td>III. Requirements</td>
<td>10</td>
</tr>
<tr>
<td>IV. Specifications</td>
<td>11</td>
</tr>
<tr>
<td>V. Design</td>
<td>12</td>
</tr>
<tr>
<td>VI. Testing/Verification</td>
<td>17</td>
</tr>
<tr>
<td>VII. Conclusion</td>
<td>19</td>
</tr>
<tr>
<td>VIII. Bibliography</td>
<td>20</td>
</tr>
<tr>
<td>IX. Appendices</td>
<td>21</td>
</tr>
<tr>
<td>A. Senior Project Analysis</td>
<td>21</td>
</tr>
<tr>
<td>B. Project Timeline</td>
<td>25</td>
</tr>
<tr>
<td>C. Parts List and Costs</td>
<td>26</td>
</tr>
<tr>
<td>D. Hardware/Software and Connections</td>
<td>27</td>
</tr>
<tr>
<td>E. Code</td>
<td>31</td>
</tr>
<tr>
<td>F. User Manual</td>
<td>66</td>
</tr>
<tr>
<td>G. Datasheets</td>
<td>67</td>
</tr>
</tbody>
</table>
## List of Tables and Figures

### Table

<table>
<thead>
<tr>
<th>Table</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>I. Marketing Requirements</td>
<td>10</td>
</tr>
<tr>
<td>II. Engineering Specifications</td>
<td>11</td>
</tr>
<tr>
<td>III. GPS Configurations</td>
<td>17</td>
</tr>
<tr>
<td>IV. Power Consumption</td>
<td>18</td>
</tr>
<tr>
<td>V. Bill of Materials</td>
<td>26</td>
</tr>
</tbody>
</table>

### Figure

<table>
<thead>
<tr>
<th>Figure</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Normal View (Looking Straight)</td>
<td>6</td>
</tr>
<tr>
<td>2. Impaired View (Looking Down)</td>
<td>6</td>
</tr>
<tr>
<td>3. Cereal Bowl Electronics Container</td>
<td>13</td>
</tr>
<tr>
<td>4. Bicycle OLED Screen Mount</td>
<td>14</td>
</tr>
<tr>
<td>5. Speed and Heading Data</td>
<td>15</td>
</tr>
<tr>
<td>6. Directions and Street Data</td>
<td>16</td>
</tr>
<tr>
<td>7. Project Gantt Chart (Timeline)</td>
<td>25</td>
</tr>
<tr>
<td>8. Hardware Overview</td>
<td>26</td>
</tr>
<tr>
<td>9. Wiring Diagram</td>
<td>27</td>
</tr>
<tr>
<td>10. Software Flow Diagram</td>
<td>28</td>
</tr>
<tr>
<td>11. Hardware Schematic</td>
<td>29</td>
</tr>
<tr>
<td>12. User Manual</td>
<td>66</td>
</tr>
</tbody>
</table>
Acknowledgements

We would like to thank our advisor Dr. Bridget Benson for her support with planning, designing, and testing our project. We would also like to thank Jim Budney for inspiring our project and generously paying for our project costs. Parallax was a huge help, they donated various equipment from the OLED screen to the battery pack that powers the system. Lastly, we would like to thank all the professors at Cal Poly whom we’ve had the opportunity to learn from. It is from their great knowledge and guidance that we have been able to succeed as engineers.
Abstract

The system uses a GPS (Global Positioning System) sensor, Bluetooth modem, and OLED (organic light-emitting diode) screen to display navigational information to a motorcycle rider. Currently supported navigational information are speed and heading, with a framework in place for future turn-by-turn navigation. The system is powered by a lithium-ion battery pack and controlled by an Arduino Micro. For turn-by-turn navigation, an Android powered smartphone running our Android App is required. Our system aims to reduce safety hazards from having to tilt or move one’s head when trying to view speed on a motorcycle speedometer, which is usually mounted down by the handlebars.
I. Introduction

With all the safety innovations in automobiles which have caused a drop in the national motor-related deaths, one might think that there are also innovations available to motorcyclists; however, this is not the case. In 2008, motor vehicle crash-related deaths involving cars and light trucks reached an all-time low in the United States. At the same time, however, motorcyclist deaths reached an all-time high, more than doubling between 1999 and 2008. A recent CDC study found that between 2001 and 2008, more than 34,000 motorcyclists were killed and an estimated 1,222,000 persons were treated in a U.S. emergency department (ED) for a non-fatal motorcycle-related injury [1]. Unfortunately, there is a great shortage of data to support our claim that looking down at the instrument cluster while riding a motorcycle is dangerous. Figure 1 shows a motorcyclist’s normal view of the road, while Figure 2 shows the view when looking down at the instrument cluster. It is apparent that looking down to check the speed causes a temporary reduction in road visibility. Our goal was to eliminate this liability by moving the speedometer to the helmet.

![Figure 1: Normal View (Looking Straight)](image1)

![Figure 2: Impaired View (Looking Down)](image2)
II. Background
This section explains the GPS, Bluetooth, Arduino Microcontrollers, and Android technologies. This should serve as a general overview of how they are able to help us in this project.

GPS and NMEA Protocols
We are using the UP501 GPS NMEA Talker, which is a GPS Device that constantly sends out NMEA Sentences. From “GPS Data” we read [2]:

The National Marine Electronics Association (NMEA) has developed a specification that defines the interface between various pieces of marine electronic equipment. The standard permits marine electronics to send information to computers and to other marine equipment ... GPS receiver communication is defined within this specification. Most computer programs that provide real time position information understand and expect data to be in NMEA format. This data includes the complete PVT (position, velocity, time) solution computed by the GPS receiver. The idea of NMEA is to send a line of data called a sentence that is totally self-contained and independent from other sentences. ... All of the standard sentences have a two letter prefix that defines the device that uses that sentence type. (For GPS receivers the prefix is GP) which is followed by a three letter sequence that defines the sentence contents. ... Each sentence begins with a '$' and ends with a carriage return/line feed sequence and can be no longer than 80 characters of visible text (plus the line terminators). The data is contained within this single line with data items separated by commas. The data itself is just ASCII text and may extend over multiple sentences in certain specialized instances but is normally fully contained in one variable length sentence. ... There is a provision for a checksum at the end of each sentence which may or may not be checked by the unit that reads the data. The checksum field consists of a '*' and two hex digits representing an 8 bit exclusive OR of all characters between, but not including, the '$' and '*'.

There are a couple settings that we needed to change to get the UP501 working for our uses. First we hand to change the output mode of the Talker to only output GPRMC sentences, which are the most basic and essential sentences that have speed and heading. With the output decreased from four to one NMEA Sentence, we were able to increase the output rate from 1 Hz to 5 Hz without needing to increase the baud rate.

Bluetooth Communication
How it Works
Bluetooth uses radio frequencies to send all different kinds of data from sound, to strings and numbers to any Bluetooth receiver. A radio frequency has to travel on a certain band. For example, when you are listening to the radio on 93.1, you are traveling on the 93.1MHz wave. For Bluetooth devices to communicate, they must be on the 2.4 GHz band. A lot of devices around you, like your garage opener, already operate on that wave. Bluetooth is very smart in avoiding interference from other devices. One way this is being done is by emitting very low power; this helps lower the signal distance to 10 meters.
Safety

Curt Franklin, an author at How Bluetooth Works wrote in great detail on how the devices connect to one another and are able to ignore surrounding devices:

When Bluetooth-capable devices come within range of one another, an electronic conversation takes place to determine whether they have data to share or whether one needs to control the other. The user doesn't have to press a button or give a command -- the electronic conversation happens automatically. Bluetooth systems create a personal-area network (PAN), or piconet, that may fill a room or may encompass no more distance than that between the cell phone on a belt-clip and the headset on your head. Once a piconet is established, the members randomly hop frequencies in unison so they stay in touch with one another and avoid other piconets that may be operating in the same room (p. 6).

Bluetooth is very smart when it comes to security; not only does it hop from frequency to frequency, it also demands a password. Once the devices have made a connection, they are now able to communicate with one another and transfer data very fast. Demanding some sort of authentication is the safest way to transfer data. A hacker would not have the time to try and run an algorithm to try to find the correct password.

For this project, our Bluetooth connection is protected with a password key that we can change in the code and simply re-program. Initially, the password is set to 1234 to stop random devices being able to connect to motoHUD and send unnecessary data and obstruct the driver. This password only has to be inputted once, and the phone will remember it for future use. Bluetooth is short distance, so only the driver should be able to keep a good connection to the device.

Arduino Microcontrollers

Arduino is an open-source electronics prototyping platform based on flexible, easy-to-use hardware and software. Arduino Microcontrollers are specialized platforms for ATMega Microprocessors. We chose to use the Arduino Micro, which is very small and houses an ATMega 32U4 processor. We also chose to use the Arduino Software IDE to develop the software that runs on the microcontroller. There are a few things of note about Arduino’s Software IDE: first, it allows programmers to use open-source software libraries to abstract the complications of hardware interfacing, and second, it has a pre-defined software flow that cannot be changed by the user. We used the SoftwareSerial library to deal with having to use three UART devices. The Arduino Micro has only one hardware UART port, so using the SoftwareSerial Library allowed us to use regular GPIO pins for UART Communication. The flow of the program is two stepped, first the program calls the function named “setup()” and runs the code in there. Then the program calls the function named “loop()” indefinitely. We had to keep this in mind when designing the software.

Android Apps
Google offers an Eclipse plugin that allows developers to use Eclipse which is one the most popular integrated development environments for Java. Android is all open source, and Google provides an API to interact with their Android operating system.
III. Requirements

When designing the product, we developed a list of marketing requirements, which are high level requirements of the system that do not define implementation. We wanted our system to reliably display speed and heading information without requiring the user to divert their attention from the road.

<table>
<thead>
<tr>
<th>No.</th>
<th>Marketing Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Must display speed and heading at all times.</td>
</tr>
<tr>
<td>2</td>
<td>Must display information in a meaningful and clear fashion.</td>
</tr>
<tr>
<td>3</td>
<td>Must have a long battery life.</td>
</tr>
<tr>
<td>4</td>
<td>Must be able to be recharged.</td>
</tr>
<tr>
<td>5</td>
<td>Must be easy to turn on or off.</td>
</tr>
<tr>
<td>6</td>
<td>Must be inexpensive.</td>
</tr>
<tr>
<td>7</td>
<td>Must be visible in direct sunlight.</td>
</tr>
<tr>
<td>8</td>
<td>Must be able to be mounted onto an existing motorcycle helmet.</td>
</tr>
</tbody>
</table>

*Table I: Marketing Requirements*
## IV. Specifications

The following Engineering Specifications were developed to support the marketing requirements laid forth in Table I.

<table>
<thead>
<tr>
<th>No.</th>
<th>Marketing No.</th>
<th>Engineering Specification</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>Speed and heading are displayed at startup and cannot be disabled. If no valid speed is detected from the GPS, 000 is displayed.</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>Speed is given approximately 50% of the OLED screen real estate. This ensures maximum visibility and leaves room for directional arrows (Turn-by-Turn).</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>Bright colors with high contrast were chosen to display information.</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>Speed is displayed with a 1 MPH resolution and heading is displayed accurate to the eight major directions N, NE, E, SE, S, SW, W, NW, which corresponds to 45 Degree increments.</td>
</tr>
<tr>
<td>5</td>
<td>3</td>
<td>2600 mAH battery provides more than 8 hours continuous use.</td>
</tr>
<tr>
<td>6</td>
<td>4</td>
<td>Lithium Ion 18650 battery is rechargeable up to 1,000 cycles.</td>
</tr>
<tr>
<td>7</td>
<td>5</td>
<td>Power is controlled by an oversized rocker switch, no other controls are needed.</td>
</tr>
<tr>
<td>8</td>
<td>6</td>
<td>Total system cost was less than $300 (see Appendix C).</td>
</tr>
<tr>
<td>9</td>
<td>7</td>
<td>OLED Screen is backlit and bright enough to be read in direct sunlight.</td>
</tr>
<tr>
<td>10</td>
<td>8</td>
<td>The system is self-contained, and only requires a standard full-faced motorcycle helmet.</td>
</tr>
</tbody>
</table>

**Table II: Engineering Specifications**
V. Design
This section covers our hardware and software design choices and problems. It will also explain how the software and hardware come together to complete the project. We will be talking about the flow of our program, how we receive GPS data, and how the Bluetooth android application communicates with the device on the helmet.

Hardware Design

Design Constraints

The hardware was designed with space constraints in mind. We chose small components to keep the size of the system small enough to mount onto a motorcycle helmet. Our next design constraint was the display technology. We originally planned to have some sort of projection technology that would project an image, with focal point at infinity, onto the face shield of the helmet. This plan was quickly abandoned as it requires special materials and expensive technology. Our next idea was to use a transparent OLED screen made by 4D Systems. Because the transparent OLED Screen and the final OLED Screen we decided to use are powered by the same GOLDELOX processor [5][6], they have the same command set and capabilities. This made switching between each screen as simple as plugging one or the other into the system, no code changes were required. The transparent OLED screen turned out to be not bright enough in direct sunlight, so we had to scrap it.

Hardware Overview

The motoHUD system uses a microcontroller and several peripherals including an android smartphone to accomplish our goals (see Figure 9, Appendix D). The main driver of the system is the ATMega 32U4 microprocessor [9] running on the Arduino Micro platform [10]. This microcontroller gives us plenty of GPIO pins for our peripherals, and ample processing power all while taking up very little space. We have the UP501 GPS NMEA Talker [7] connected to the Micro to provide speed and heading. The RN-42 Bluetooth Module [8] provides a serial communication line to an Android smartphone running the MotoHUD app. The uOLED 128-G2 Screen is connected to the Micro and displays the navigational data to the rider [6]. The final component is the battery, we chose the Lithium Ion Boe-Bot Power Pack from Parallax [11] because it provides plenty of power for a sustained run time.

Communication Protocols

Three UART connections are used, one for each peripheral. Each UART connection runs at 9600 baud. All of the UART connections are running in Full-Duplex mode, because every component requires commands for configuration, and every component supplies data to the microcontroller. Each component is wired to the micro; however, only the Bluetooth is connected to the Hardware UART on the micro (see Figure 10, Appendix D). This is because the Bluetooth sends data intermittently and cannot afford to be missed, so it requires the Hardware UART’s built-in buffer to store messages until they can be processed. The other UARTs are connected to regular GPIO pins and are running on the Arduino SoftwareSerial library. Only one SoftwareSerial connection can
receive data at a time, but missing a few GPS points or an LCD ACK is not as critical as missing a navigational command.

**Mounting Hardware**

We mounted the system to the motorcycle helmet in a somewhat unconventional way; we zip-tied a cereal bowl upside down to the top of the helmet, with all of the electronics inside (see Figure 3). We drilled a few holes for charging the battery, connecting the OLED screen, and for connecting the USB cable for reprogramming. The OLED screen is mounted to a bicycle rear-view mirror mount (see Figure 4), which we bought from a local bicycle store. This allows the OLED screen to sit in the lower left corner of the field of view, which makes it visible but not obtrusive.

*Figure 3: Cereal Bowl Electronics Container*
Figure 4: Bicycle OLED Screen Mount

Schematic

A simple schematic was developed (see Figure 11, Appendix D) for making a custom PCB to house the component, but we did not have enough time to make this into a PCB and have it manufactured.

Software Design

This section will describe how the Arduino and Android software work together to control the system, as well as how the GPS and Bluetooth data streams are processed.

Software Flow

The software design was mostly determined by the mechanics of the Arduino IDE described in the Background section of this report. The software operates in two phases, the setup phase, and the running phase as shown in the Software Flow Diagram (see Figure 10, Appendix D). In the setup phase, the setup() function is executed and the UART connections are configured and the various configuration settings are sent to the appropriate devices (see Background section). Once setup is complete, the loop() function is executed. This function will listen on each UART connection for
data, switching to each UART in succession for reading. When a full UART communication string is finished transmitting, the program sets a flag to process the data.

**GPS Data**

Each time the system is powered on, the setup() and loop() functions are executed. When power is lost to the GPS, it must startup in cold start mode. While the GPS is looking for satellites to lock onto, the speed value is displayed as 0. This process can take up to four minutes, which is quite undesirable. In the future, we would like to add a CR2032 Coin Cell Battery to the system, connected to the backup voltage on the GPS to keep it from ever having to enter cold start mode.

If valid data is present to process, the function will use the utility functions parseNMEA() and convert_degrees() to facilitate processing either the GPS data. parseNMEA() will iterate through the NMEA sentence, counting the delimiters until it gets to the speed value. It copies the speed into a buffer, converts it from a string to float, converts it from knots to MPH, converts it back to an integer, converts it back to a string, then places it in the provided buffer. The same is done for the heading value, which comes right after speed. Convert_degrees() takes uses the heading and converts it to one of eight letter approximations of the heading such as NE or S. Once the data is properly parsed and processed, the OLED screen is updated with the new information. Please see Appendix E for full source code disclosure. The speed is displayed at the top of the OLED screen, with the heading displayed right underneath (see Figure 5).

![Figure 5: Speed and Heading Data](image)

**Bluetooth Data**

If valid Bluetooth data is present, which is identified by a 1 byte delimiter, it is processed. The software will look for a message type in the next byte, and depending on the type, will parse the data and display the command. The message types currently supported are Directional Arrow, and Street Name. Each of the message types can be seen in Figure 6 below.
Using Bluetooth for Android is very similar to setting up a server connection using C. We first had to setup a Bluetooth server socket which will wait for a connection to be made. Before a connection can be made, the phone must first use a broadcast to receive a list of all Bluetooth enabled devices that are around the phone. Once the broadcast has been made and the motoHUD device has been chosen, the Bluetooth socket is bound to the paired device socket. This socket is then used to send and receive information from each device. For the microcontroller to understand what kind of data we are sending, each message is sent with a delimiter that specifies if the phone is sending a street name, or a direction.
VI. Testing/Verification
This section will explain our testing procedures and show how we tested each requirement from Section IV.

Component Level Testing
Testing was done incrementally throughout the development process. When a new component was successfully interfaced, quick tests were performed to ensure that the component was working properly. Testing the GPS involved displaying the raw NMEA sentences on a serial window on the PC. Once that was working, and we could tell that our configuration commands were being properly received by the GPS, we were able to write some code to extract out the speed and heading portion of the GPRMC sentence. Displaying this information on a basic 16x2 LCD was the next step, until we were able to properly interface the OLED screen.

Once the Bluetooth was interfaced, we tested it by sending it commands to change its name, which was verified by using a smartphone to scan for available Bluetooth devices. Once we got the phone and Bluetooth paired and communicating, we used a barebones android app as a sort of chat client, where we would send a text string to the Bluetooth, which would simply send back the received string to the phone.

The OLED screen was probably the most frustrating and difficult component to work with. We spent weeks trying to establish a communication with the OLED screen, only to find out that our efforts were in vain. The OLED screens are shipped out in a “Slave” configuration, which allows them to receive UART commands that control the screen; however, our screen was donated by Parallax, and had been taken out of “Slave” mode by one of their engineers. Once we discovered this, we were forced to purchase a 4D Systems Programming Cable, which was required to reconfigure the OLED screen into a “Slave” mode. It was a very joyous moment when we were able to write a line of text across the top of the screen. Testing after properly interfacing was simple; it involved a quick visual check to see if it was displaying the right image.

Integration Testing
Once all the components had been successfully tested separately, we were able to test them as a whole system. Our first tests were predominately driving around town, trying to verify the accuracy and precision of the GPS. Our first challenge was to get one mile per hour precision, which required setting the GPS to a higher output rate, thus giving us more data points between updating the screen. We found that increasing the output rate also improved the accuracy. See Table III for our final GPS results.

<table>
<thead>
<tr>
<th>GPS Fix Rate</th>
<th>GPS Output</th>
<th>GPS Baud Rate</th>
<th>GPS Speed Resolution</th>
<th>GPS Heading Resolution</th>
</tr>
</thead>
<tbody>
<tr>
<td>5 Hz</td>
<td>RMC only</td>
<td>9600</td>
<td>1 MPH</td>
<td>45 Degrees</td>
</tr>
</tbody>
</table>

Table III: GPS Configurations
Specifications 1 – 3 from Table II were verified by visual inspection. 4 was verified by comparing the speed value to a car speedometer, as seen in our online video [12].

The Android App we developed to control the system and provide Turn-by-Turn navigation was the last integration test. We tested the system by sending the system commands from the phone, and visually verifying that the desired results appeared on the screen.

**Battery Life Testing**

The power requirements were tested by running the system off of an Agilent Bench power supply, and recording the current draw of the system with all peripherals enabled. This number was used to calculate the running time of the system, which was determined to be 9.82 hours continuous use.

<table>
<thead>
<tr>
<th>Condition</th>
<th>Input Voltage (V)</th>
<th>Current drawn (A)</th>
<th>Device Power (W)</th>
<th>Life(hours)</th>
</tr>
</thead>
<tbody>
<tr>
<td>No Bluetooth</td>
<td>7</td>
<td>.11</td>
<td>.77</td>
<td>12.5</td>
</tr>
<tr>
<td>Bluetooth on</td>
<td>7</td>
<td>.14</td>
<td>.98</td>
<td>9.82</td>
</tr>
</tbody>
</table>

**Table IV: Power Consumption**

**Usability Testing**

Many hours of testing to determine usability of the system took place at night. It was because of this that we missed one of the great pitfalls of our system – direct sunlight. Once we realized that our screen would be in direct light most of the time, we had to mount it in such a way that would prevent glare or hinder the visibility. As discussed previously, direct sunlight was responsible for eliminating the possibility of using projection technology, or the transparent OLED screen that we bought.

Using the bicycle mount system for the OLED screen turned out to be very good for adjusting the location of the screen to suit the user’s preferences.

Once the project was near completion, we had our advisor test the product on her scooter. She reported that the system worked and was very accurate, but that the heading was too small to be read effectively. We plan on making this larger in the future.
VII. Conclusion

Our system successfully displays speed with one Mile/Hour resolution, and heading with 8 degrees of resolution (see Table III). Our system meets all goals and requirements defined in Section III for a reliable and useable device to display speed and heading on a motorcycle helmet without requiring the rider to look away from the road. The total cost of the system was $225.30 and fit our original budget (see Table IV, Appendix C). A user manual was created as a possible accompanying literature if the product was commercialized (see Appendix F). Future developments for the project are adding a coin cell battery to eliminate the need for cold starting the GPS, adding Turn-by-Turn navigation using Google Maps and our message/control framework, and creating a 3D printed mounting system for attaching the device to the helmet.
VIII. Bibliography


[5] UTOLED Datasheet: Appendix G


[8] RN-42 Datasheet: Appendix G

[9] ATMega 32U4 Datasheet: Appendix G


[12] http://www.youtube.com/watch?v=m75R5Zg8Hy4
IX. Appendices

A. Senior Project Analysis

Analysis of Senior Project Design

Project Title: MotoHUD: Intelligent and Safe Navigational Data Presentation for Motorcyclists

Quarter / Year Submitted: Spring 2013

Student: (Print Name):  Drew Bentz     (Sign) ___________________________

Student: (Print Name):  William Budney     (Sign) ___________________________

Advisor: (Print Name): Bridget Benson  (Initial)   BB  Date:  __________

Summary of Functional Requirements

The system uses a GPS (Global Positioning System) sensor, Bluetooth modem, and OLED (organic light-emitting diode) screen to display navigational information to a motorcycle rider. Currently supported navigational information are speed and heading, with a framework in place for future turn-by-turn navigation. The system is powered by a lithium-ion battery pack and controlled by an Arduino Micro. For turn-by-turn navigation, an Android powered smartphone running our Android App is required. Our system aims to reduce safety hazards from having to tilt or move one’s head when trying to view speed on a motorcycle speedometer, which is usually mounted down by the handlebars.

Primary Constraints

Effectively displaying navigational information on a motorcycle helmet was our biggest constraint. Motorcycles are ridden outdoors, and are in direct sunlight most of the time. This presented a challenge in finding a display technology that would be easy to see in direct sunlight. Several of our designs had to be scrapped because of this problem. Another problem, somewhat related, is that our original plan to have the information be projected onto the face shield could not be realized due to the poor optics associated with the cheap plastic used for making the face shields. Projection technology had to be abandoned. Associated with displaying the information is how we choose to mount the OLED screen onto the helmet. We tried several methods, but only one allowed clear visibility without obstructing the view of the rider.

Obtaining accurate speed measurements was another challenge. We had to find a way to get our GPS to emit only relevant data, and at a much faster pace than how it comes. This was eventually accomplished, but required many hours of testing.
The Arduino development environment proved to be another constraint. The software is full of bugs and has erratic behavior. We also had to rely on software libraries developed by other people, which had their own set of workarounds and bugs.

**Economic**

Our original estimate was approximately $280 and can be broken down as follows:

<table>
<thead>
<tr>
<th>Item</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>GPS</td>
<td>$70</td>
</tr>
<tr>
<td>Bluetooth</td>
<td>$40</td>
</tr>
<tr>
<td>Logic Level Converter</td>
<td>$2</td>
</tr>
<tr>
<td>Arduino Micro</td>
<td>$35</td>
</tr>
<tr>
<td>LCD/Projection</td>
<td>$100</td>
</tr>
<tr>
<td>Battery</td>
<td>$30</td>
</tr>
</tbody>
</table>

Table 1: Original Cost Estimate

The final cost of the project was $225.30 and can be broken down as follows:

<table>
<thead>
<tr>
<th>Qty</th>
<th>Description</th>
<th>Supplier</th>
<th>Part Number</th>
<th>Price($)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Arduino Mico</td>
<td>Radioshack</td>
<td>276-258</td>
<td>29.99</td>
</tr>
<tr>
<td>1</td>
<td>GPS 66 Channel UP-501 GPS Receiver</td>
<td>Sparkfun</td>
<td>GPS - 1002</td>
<td>49.95</td>
</tr>
<tr>
<td>2</td>
<td>Bluetooth SMD Module</td>
<td>Sparkfun</td>
<td>WRL - 10823</td>
<td>39.90</td>
</tr>
<tr>
<td>1</td>
<td>Rechargeable Power Supply</td>
<td>Parallax</td>
<td>28988</td>
<td>44.99</td>
</tr>
<tr>
<td>1</td>
<td>microtivity IM414 Double-sided Prototyping Board</td>
<td>Amazon</td>
<td>B007K7I83C</td>
<td>8.99</td>
</tr>
<tr>
<td>1</td>
<td>uOLED-128-G2</td>
<td>Parallax</td>
<td>28081</td>
<td>49.99</td>
</tr>
<tr>
<td>1</td>
<td>Motorcycle Helmet</td>
<td>Bell Arrow</td>
<td>23426</td>
<td>Provided</td>
</tr>
<tr>
<td>1</td>
<td>Cereal Bowl</td>
<td>Target</td>
<td>N/A</td>
<td>2.00</td>
</tr>
<tr>
<td>1</td>
<td>12V Rocker Switch</td>
<td>Radioshack</td>
<td>275-018</td>
<td>4.49</td>
</tr>
</tbody>
</table>

Total: 225.30

Table 2: Final Cost Breakdown

Additional equipment costs for development include an internet enabled computer that can run the Arduino IDE.

Our original estimated development time was 160 Hours (8 hours per week @ 20 weeks) while our actual development time was closer to 200 Hours total. Please see the Gantt chart below for a more detailed representation of our development timeline:
Environmental

All parts are obtained through retail or wholesale outlets, none of the manufacturing of the project is controlled by us. The parts obtained are RoHS certified. Our system aims to reduce the number of crashes related to diversion of attention to the speedometer mounted to the motorcycle, so it could potentially have a positive impact in that regard.

Manufacturability

Our final design is of a prototype nature, so all of the components can be bought and assembled by anyone in the public, there are not custom manufactured parts. We did design a custom PCB that could house the components we are using and make the whole system have a much smaller profile, but time did not allow us to actually manufacture any custom boards.

Sustainability

The system runs on a rechargeable battery, which is the only consumable part in the system. The user has the ability to change the batteries without hassle after their rechargeable life is depleted. Changing the charging system to something that could be wirelessly charged would be helpful, but that requires special and expensive equipment. Our plan to add turn-by-turn navigation was not
finished, but we did manage to get a robust framework in place to allow future development, because the Bluetooth chip is already powered by the system, adding Turn-by-Turn navigation won’t increase the power requirements of the system.

**Ethical**

The main ethical concern with this project is that in the event of a bug or misuse, the information displayed to the rider may become a distraction, which is the main problem the project is trying to solve. In the event that the rider loses control of the motorcycle because of a distraction caused by our system, the legal and social responsibilities of both parties will have to be considered.

**Health and Safety**

Our device will be using Bluetooth wireless technology at a spot directly adjacent to the brain. Some people may consider this a health risk, but no definitive scientific data exists to support that claim. Other safety concerns were addressed in the Ethical section. Our system can be beneficial to the health of the rider if it allows him/her to avoid traffic collisions. Having custom mounted electronics on a helmet may or may not violate regulations regarding the safety certification that a helmet must pass, depending on state rules and regulations.

**Social and Political**

Assuming the use of our product is allowed, both motorcycle riders and the general motorist population will benefit from keeping motorcycle riders’ attention focused on the road.

**Development**

The Arduino development environment was a new technology to us, and learning to use it has proved useful in other projects as well. We also learned much about the patent application process, even though we decided in the end to not file for IP patents.
### B. Project Timeline

<table>
<thead>
<tr>
<th>Week 1</th>
<th>Week 2</th>
<th>Week 3</th>
<th>Week 4</th>
<th>Week 5</th>
<th>Week 6</th>
<th>Week 7</th>
<th>Week 8</th>
<th>Week 9</th>
<th>Week 10</th>
</tr>
</thead>
<tbody>
<tr>
<td>Research &amp; Order Parts</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Interface GPS</td>
<td>Interface Bluetooth</td>
<td></td>
<td></td>
<td></td>
<td>Write GPS Software</td>
</tr>
<tr>
<td>Week 11</td>
<td>Week 12</td>
<td>Week 13</td>
<td>Week 14</td>
<td>Week 15</td>
<td>Week 16</td>
<td>Week 17</td>
<td>Week 18</td>
<td>Week 19</td>
<td>Week 20</td>
</tr>
<tr>
<td>Write GPS Software (cont.)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Test Software</td>
</tr>
<tr>
<td>Week 21</td>
<td>Week 22</td>
<td>Week 23</td>
<td>Week 24</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Design Docs</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Design Helmet</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Develop Custom PCB</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Key</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Assembly &amp; Wiring</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Test System Comfort &amp; Useablity</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Write Report</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Figure 7**: Project Gantt Chart (Timeline)
C. Parts List and Costs

<table>
<thead>
<tr>
<th>Qty</th>
<th>Description</th>
<th>Supplier</th>
<th>Part Number</th>
<th>Price($)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Arduino Mico</td>
<td>Radioshack</td>
<td>276-258</td>
<td>29.99</td>
</tr>
<tr>
<td>1</td>
<td>GPS 66 Channel UP-501 GPS Receiver</td>
<td>Sparkfun</td>
<td>GPS - 1002</td>
<td>49.95</td>
</tr>
<tr>
<td>2</td>
<td>Bluetooth SMD Module</td>
<td>Sparkfun</td>
<td>WRL - 10823</td>
<td>39.90</td>
</tr>
<tr>
<td>1</td>
<td>Rechargeable Power Supply</td>
<td>Parallax</td>
<td>28988</td>
<td>44.99</td>
</tr>
<tr>
<td>1</td>
<td>microtivity IM414 Double-sided Prototyping Board</td>
<td>Amazon</td>
<td>B007K7183C</td>
<td>8.99</td>
</tr>
<tr>
<td>1</td>
<td>uOLED-128-G2</td>
<td>Parallax</td>
<td>28081</td>
<td>49.99</td>
</tr>
<tr>
<td>1</td>
<td>Motorcycle Helmet</td>
<td>Bell Arrow</td>
<td>23426</td>
<td>Provided</td>
</tr>
<tr>
<td>1</td>
<td>Cereal Bowl</td>
<td>Target</td>
<td>N/A</td>
<td>2.00</td>
</tr>
<tr>
<td>1</td>
<td>12V Rocker Switch</td>
<td>Radioshack</td>
<td>275-018</td>
<td>4.49</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>Total</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Total</td>
<td></td>
<td>225.30</td>
</tr>
</tbody>
</table>

*Table V: Bill of Materials*
D. Hardware/Software and Connections

![Diagram of Hardware Connections]

**Figure 8:** Hardware Overview
Figure 9: Wiring Diagram
Setup()
initGPS
initBluetooth
initLCD
Loop()
Listen(Bluetooth)
Listen(GPS)
updateLCD()
Valid?
YES
NO
Process Data
updateLCD()

Figure 10: Software Flow Diagram
Figure 11: Hardware Schematic
#include <SoftwareSerial.h>

#define rxBTPin 8
#define txBTPin 9

#define BTSerial Serial1
SoftwareSerial lcdSerial(2, 3);
SoftwareSerial gpsSerial(11, 12);

// Set fix rate commands
#define PMTK_SET_NMEA_UPDATE_1HZ "$PMTK220,1000*1F"
#define PMTK_SET_NMEA_UPDATE_5HZ "$PMTK220,200*2C"
#define PMTK_SET_NMEA_UPDATE_10HZ "$PMTK220,100*2F"

// turn on only the second sentence (GPRMC)
#define PMTK_SET_NMEA_OUTPUT_RMCONLY
"$PMTK314,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0*29"

// turn on ALL THE DATA
#define PMTK_SET_NMEA_OUTPUT_ALLDATA
"$PMTK314,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0*28"

// LCD Constants
#define LCD_CLS -41
#define LANDSCAPE 0
#define LANDSCAPE_R 1
#define PORTRAIT 2
#define PORTRAIT_R 3
#define RED 0xF800
#define FORESTGREEN 0x2444
#define DARKBLUE 0x0011
#define DODGERBLUE 0x001F
#define BLACK 0x0000
// ======== Global Data & Variables ======== //
unsigned long spd;
char nmea_sentence[200];
char bt_sentence[100];
char street[255];
char mph[6];
char heading[6];
char state[1];

String nmeaString = "";
String btString = "";

boolean nmeaDone = false;
boolean btDone = false;

// ======== Main Setup Function ======== //
void setup()
{
    // Setup GPS
    GPSSetup();
delay(1000);

    // Setup LCD
    LCDSetup();
delay(1000);

    // Setup Bluetooth
    BtSetup();

    // Allocate memory for string objects
    nmeaString.reserve(200);
    btString.reserve(100);
}

// ======== Bluetooth Setup ======== //
void BtSetup()
{
    // Initialize Hardware Serial
    BTSerial.begin(9600);
}

// ======== GPS Setup ======== //
void GPSSetup()
{
    // 9600 NMEA is the default baud rate
    gpsSerial.begin(9600);

    // Set GPRMC output only
    gpsSerial.println(PMTK_SET_NMEA_OUTPUT_RMCONLY);

    // Set 5 Hz update rate
    gpsSerial.println(PMTK_SET_NMEA_UPDATE_5HZ);

    // Disable GPS tx pin
pinMode(11, LOW);
}

// ======= LCD Setup ======= //
void LCDSetup()
{
    // Setup software serial
    lcdSerial.begin(9600);
    delay(1000);

    // Prepare LCD
    lcd_disable_screensaver();
    lcd_cls();

    lcd_set_screen_mode(LANDSCAPE);
    lcd_set_text_opac(1);
    lcd_set_contrast(15);
    lcd_set_text_bold(0);

    lcd_text_background_color(0);
    lcd_text_foreground_color(DODGERBLUE);
    lcd_set_text_width(6);
    lcd_set_text_height(6);
}

// ======= Main Program Loop ======= //
void loop()
{
    int i, j;
    i = j = 0;

    while(BTSerial.available())
    {
        char inChar = (char)BTSerial.read();
        bt_sentence[j++] = inChar;
        if (inChar == '$')
        {
            bt_sentence[j] = '\0';
            j = 0;
            btDone = true;
        }
    }

    gpsSerial.listen();
    while (gpsSerial.available())
    {
        char inChar = (char)gpsSerial.read();
        nmeaString += inChar;
        if (inChar == '\r')
        {
            nmeaDone = true;
        }
    }
}
if (nmeaDone)
{
    nmeaString.toCharArray(nmea_sentence, 200);
    parseNMEA(nmea_sentence, mph, heading);
    lcd_move_cursor(0, 0);
    lcd_print_str(mph);

    nmeaString = "";
    nmeaDone = false;
}
if (btDone)
{
    lcd_move_cursor(1, 0);
    if (bt_sentence[0] == '@' && bt_sentence[1] != state[0])
    {
        switch (bt_sentence[1])
        {
        case 'L':
            uSD_set_sector_address(0x0000, 0x0200);
            uSD_display_image(5,5);
            lcd_draw_left_arrow();
            state[0] = 'L';
            break;
        case 'R':
            lcd_draw_right_arrow();
            state[0] = 'R';
            break;
        case 'S':
            lcd_draw_straight_arrow();
            state[0] = 'S';
            break;
        case 'U':
            lcd_draw_uturn_arrow();
            state[0] = 'U';
            break;
        default:
            break;
        }
    }
    if (bt_sentence[0] == '%')
    {
        int k = 1;
        while (bt_sentence[k] != '$')
        {
            street[k-1] = bt_sentence[k];
            k++;
        }
        while (k < 10)
        {
            street[k-1] = ' '; 
            k++;
        }
        street[k] = '\0';
    }
LCD_set_text_width(2);
LCD_set_text_height(2);
LCD_move_cursor(7, 0);
LCD_print_str(street);
LCD_set_text_width(6);
LCD_set_text_height(6);

if (bt_sentence[0] == '!')
{
  draw_notification();
}

memset(bt_sentence, 0, 100);
memset(street, 0, 255);
btDone = false;

// ======= Parses NMEA Sentence ======= //
void parseNMEA(char *nmea, char *speed_buf, char *heading_buf)
{
  int i = 0;
  int commas = 0;
  int j;
  float speed_mph, heading;
  int speed_int, heading_int;
  char speed_f[8];
  char heading_f[8];

  while (commas < 7)
  {
    while (nmea[i++] != ',');
    commas++;
  }

  j = 0;
  while (nmea[i] != ',')
  {
    speed_f[j++] = nmea[i++];
  }
  j++;
  speed_f[j] = '\0';
  speed_mph = (float)atof(speed_f);
  speed_mph *= (float)1.15078;

  speed_int = (int) (speed_mph);
  snprintf(speed_buf, 4, "%.3d", speed_int);
  i++;
  j = 0;
  while (nmea[i] != ',')
{ 
    heading_f[j++] = nmea[i++];
}
j++;
heading_f[j] = '\0';

heading_int = (float)atof(heading_f);
snprintf(heading_buf, 4, "%.3d", heading_int);
convert_degrees(heading_int);
}

// ====== Converts from degrees to Heading Indicator ======== //
void convert_degrees(int degree)
{
    lcd_set_text_width(2);
    lcd_set_text_height(2);
    lcd_move_cursor(3, 0);
    if(degree >= 0 && degree <10)
        lcd_print_str("N ");
    else if(degree >= 10 && degree <81)
        lcd_print_str("NE");
    else if(degree >= 81 && degree <100)
        lcd_print_str("E ");
    else if(degree >= 100 && degree <171)
        lcd_print_str("SE");
    else if(degree >= 171 && degree <190)
        lcd_print_str("S ");
    else if(degree >= 190 && degree <261)
        lcd_print_str("SW");
    else if(degree >= 261 && degree <280)
        lcd_print_str("W ");
    else if(degree >= 280 && degree <351)
        lcd_print_str("NW");
    else if(degree >= 351 && degree <361)
        lcd_print_str("N ");
    else
        lcd_print_str(" ");
    lcd_set_text_width(6);
    lcd_set_text_height(6);
}

// ====== Begin LCD Software Library ======== //

// ====== Draws a complex shape ======== //
void draw_notification()
{
    lcd_draw_rectangle_filled(100, 60, 120, 80, RED);
}

// ====== Disables scrolling screensaver ======== //
void lcd_disable_screensaver()
{
    lcdSerial.write((byte)(0x00));
}
lcdSerial.write((byte)(0x0C));
lcdSerial.write((byte)(0x00));
lcdSerial.write((byte)(0x00));
GetAck();
}

// ======= Draws a Rectangle ======== //
void lcd_draw_rectangle(word x1, word y1, word x2, word y2, word color)
{
    lcdSerial.write((byte)(0xFF));
lcdSerial.write((byte)(0xCF));
lcdSerial.write((byte)(x1 >> 8));
lcdSerial.write((byte)(x1));
lcdSerial.write((byte)(y1 >> 8));
lcdSerial.write((byte)(y1));
lcdSerial.write((byte)(x2 >> 8));
lcdSerial.write((byte)(x2));
lcdSerial.write((byte)(y2 >> 8));
lcdSerial.write((byte)(y2));
lcdSerial.write((byte)(color >> 8));
lcdSerial.write((byte)(color));
GetAck();
}

// ======= Draws a Filled Rectangle ======== //
void lcd_draw_rectangle_filled(word x1, word y1, word x2, word y2, word color)
{
    lcdSerial.write((byte)(0xFF));
lcdSerial.write((byte)(0xCE));
lcdSerial.write((byte)(x1 >> 8));
lcdSerial.write((byte)(x1));
lcdSerial.write((byte)(y1 >> 8));
lcdSerial.write((byte)(y1));
lcdSerial.write((byte)(x2 >> 8));
lcdSerial.write((byte)(x2));
lcdSerial.write((byte)(y2 >> 8));
lcdSerial.write((byte)(y2));
lcdSerial.write((byte)(color >> 8));
lcdSerial.write((byte)(color));
GetAck();
}

// ======= Draws a Filled Circle ======== //
void lcd_draw_circle_filled(word x1, word y1, word r, word color)
{
    lcdSerial.write((byte)(0xFF));
lcdSerial.write((byte)(0xCC));
lcdSerial.write((byte)(x1 >> 8));
lcdSerial.write((byte)(x1));
lcdSerial.write((byte)(y1 >> 8));
lcdSerial.write((byte)(y1));
lcdSerial.write((byte)(r >> 8));
lcdSerial.write((byte)(r));
}
lcdSerial.write((byte)(color >> 8));
lcdSerial.write((byte)(color));
GetAck();
}

// ======== Draws a Triangle ======== //
void lcd_draw_triangle(word x1, word y1, word x2, word y2, word x3, word y3, word color)
{
    lcdSerial.write((byte)(0xFF));
lcdSerial.write((byte)(0xC9));
lcdSerial.write((byte)(x1 >> 8));
lcdSerial.write((byte)(x1));
lcdSerial.write((byte)(y1 >> 8));
lcdSerial.write((byte)(y1));
lcdSerial.write((byte)(x2 >> 8));
lcdSerial.write((byte)(x2));
lcdSerial.write((byte)(y2 >> 8));
lcdSerial.write((byte)(y2));
lcdSerial.write((byte)(x3 >> 8));
lcdSerial.write((byte)(x3));
lcdSerial.write((byte)(y3 >> 8));
lcdSerial.write((byte)(y3));
lcdSerial.write((byte)(color >> 8));
lcdSerial.write((byte)(color));
//GetAck();
}

// ======== Draws a Line ======== //
void lcd_draw_line(word x1, word y1, word x2, word y2, word color)
{
    lcdSerial.write((byte)(0xFF));
lcdSerial.write((byte)(0xD2));
lcdSerial.write((byte)(x1 >> 8));
lcdSerial.write((byte)(x1));
lcdSerial.write((byte)(y1 >> 8));
lcdSerial.write((byte)(y1));
lcdSerial.write((byte)(x2 >> 8));
lcdSerial.write((byte)(x2));
lcdSerial.write((byte)(y2 >> 8));
lcdSerial.write((byte)(y2));
lcdSerial.write((byte)(color >> 8));
lcdSerial.write((byte)(color));
GetAck();
}

// ======== Draws a Right Arrow ======== //
void lcd_draw_right_arrow()
{
    lcd_draw_rectangle_filled(30, 45, 110, 110, BLACK);
lcd_draw_rectangle_filled(0, 64, 50, 110, BLACK);
    int i = 0;
lcd_draw_rectangle_filled(56, 70, 72, 110, RED);
lcd_draw_circle_filled(76, 75, 20, RED);
lcd_draw_rectangle_filled(73, 50, 97, 110, BLACK);
lcd_draw_rectangle_filled(73, 55, 85, 75, RED);
for(i = 0; i < 17; i++) {
    lcd_draw_triangle(85, 66-i, 85, 66+i, 85+i, 66, RED);
}
lcd_draw_line(73, 66, 102, 66, RED);

// ======== Draws a Left Arrow ========= //
void lcd_draw_left_arrow()
{
    lcd_draw_rectangle_filled(30, 45, 110, 110, BLACK);
    lcd_draw_rectangle_filled(0, 64, 50, 110, BLACK);
    int i = 0;
    lcd_draw_rectangle_filled(56, 70, 72, 110, RED);
    lcd_draw_circle_filled(52, 75, 20, RED);
    lcd_draw_rectangle_filled(32, 50, 55, 110, BLACK);
    lcd_draw_rectangle_filled(44, 55, 56, 75, RED);
    for(i = 0; i < 17; i++) {
        lcd_draw_triangle(45-i, 66-i, 45+i, 66-i, 45+i, 66, RED);
    }
lcd_draw_line(29, 66, 44, 66, RED);
}

// ======== Draws a Straight Arrow ========= //
void lcd_draw_straight_arrow()
{
    lcd_draw_rectangle_filled(30, 45, 110, 110, BLACK);
    lcd_draw_rectangle_filled(0, 64, 50, 110, BLACK);
    int i = 0;
    lcd_draw_rectangle_filled(56, 70, 72, 110, RED);
    for(i = 0; i < 17; i++) {
        lcd_draw_triangle(64-i, 70, 64+i, 70, 64, 70-i, RED);
    }
lcd_draw_line(64, 55, 64, 70, RED);
}

// ======== Draws a UTurn Arrow ========= //
void lcd_draw_uturn_arrow()
{
    lcd_draw_rectangle_filled(30, 45, 110, 110, BLACK);
    lcd_draw_rectangle_filled(0, 64, 50, 110, BLACK);
    int i = 0;
    lcd_draw_rectangle_filled(56, 70, 72, 110, RED);
    lcd_draw_circle_filled(52, 75, 20, RED);
    lcd_draw_rectangle_filled(32, 50, 55, 110, BLACK);
    lcd_draw_rectangle_filled(32, 70, 46, 75, RED);
    for(i = 0; i < 17; i++) {
        lcd_draw_triangle(39-i, 75, 39+i, 75, 39, 75+i, RED);
    }
}
lcd_draw_line(39, 75, 39, 92, RED);
}

// ==
// == Prints a String ==
// ==
void lcd_print_str(char *str)
{
    int i = 0;
    lcdSerial.write((byte)(0x00));
    lcdSerial.write((byte)(0x06));
    while((str[i] != '\n') && (str[i] != '\0'))
    {
        lcdSerial.write((byte)(str[i++]));
    }
    lcdSerial.write((byte)(0x00));
    GetAck();
}

// ==
// == Sets Text Color ==
// ==
void lcd_text_foreground_color(word color)
{
    lcdSerial.write((byte)(0xFF));
    lcdSerial.write((byte)(0x7F));
    lcdSerial.write((byte)(color >> 8));
    lcdSerial.write((byte)(color));
    GetAck();
}

// ==
// == Sets Text Background Color ==
// ==
void lcd_text_background_color(word color)
{
    lcdSerial.write((byte)(0xFF));
    lcdSerial.write((byte)(0x7E));
    lcdSerial.write((byte)(color >> 8));
    lcdSerial.write((byte)(color));
    GetAck();
}

// ==
// == Moves Screen Cursor ==
// ==
void lcd_move_cursor(word line, word col)
{
    lcdSerial.write((byte)(0xFF));
    lcdSerial.write((byte)(0xE4));
    lcdSerial.write((byte)(line >> 8));
    lcdSerial.write((byte)(line));
    lcdSerial.write((byte)(col >> 8));
    lcdSerial.write((byte)(col));
    GetAck();
}

// ==
// == Prints a Single Character ==
// ==
void lcd_put_char(word chr)
{
}
void lcd_set_text_width(word mult)
void lcd_set_text_xgap(word pxl)
{
    lcdSerial.write((byte)(0xFF));
    lcdSerial.write((byte)(0x7A));
    lcdSerial.write((byte)(pxl >> 8));
    lcdSerial.write((byte)(pxl));
    GetAck();
}

void lcd_set_text_ygap(word pxl)
{
    lcdSerial.write((byte)(0xFF));
    lcdSerial.write((byte)(0x79));
    lcdSerial.write((byte)(pxl >> 8));
    lcdSerial.write((byte)(pxl));
    GetAck();
}

void lcd_set_text_opac(word mode)
{
    lcdSerial.write((byte)(0xFF));
    lcdSerial.write((byte)(0x77));
    lcdSerial.write((byte)(mode >> 8));
    lcdSerial.write((byte)(mode));
    GetAck();
}

void lcd_set_text_bold(word mode)
{
    lcdSerial.write((byte)(0xFF));
    lcdSerial.write((byte)(0x76));
    lcdSerial.write((byte)(mode >> 8));
}
lcdSerial.write((byte)(mode));
GetAck();
}

// ======= Sets Screen Contrast ======== //
void lcd_set_contrast(word cnst)
{
    lcdSerial.write((byte)(0xFF));
    lcdSerial.write((byte)(0x66));
    lcdSerial.write((byte)(cnst >> 8));
    lcdSerial.write((byte)(cnst));
    GetAck();
}

// ======= Initializes MicroSD Card Reader ======= //
void uSD_INIT()
{
    lcdSerial.write((byte)(0xFF));
    lcdSerial.write((byte)(0xB1));
    GetAck();
}

// ======= Sets Byte Address ======== //
void uSD_set_byte_address(word HIword, word LOword)
{
    lcdSerial.write((byte)(0xFF));
    lcdSerial.write((byte)(0xB9));
    lcdSerial.write((byte)(HIword >> 8));
    lcdSerial.write((byte)(HIword));
    lcdSerial.write((byte)(LOword >> 8));
    lcdSerial.write((byte)(LOword));
    GetAck();
}

// ======= Sets Sector Address ======== //
void uSD_set_sector_address(word HIword, word LOword)
{
    lcdSerial.write((byte)(0xFF));
    lcdSerial.write((byte)(0xB8));
    lcdSerial.write((byte)(HIword >> 8));
    lcdSerial.write((byte)(HIword));
    lcdSerial.write((byte)(LOword >> 8));
    lcdSerial.write((byte)(LOword));
    GetAck();
}

// ======= Draws an Image on the Screen ======== //
void uSD_display_image(word x, word y)
{
    lcdSerial.write((byte)(0xFF));
    lcdSerial.write((byte)(0xB3));
    lcdSerial.write((byte)(x >> 8));
    lcdSerial.write((byte)(x));
    lcdSerial.write((byte)(y >> 8));
    lcdSerial.write((byte)(y));
lcdSerial.write((byte)(y));
GetAck();
}

// ======= Gets ACK Message From Screen ======= //
void GetAck(void)
{
    int read;
    unsigned char readx;
    unsigned long sttime;

    lcdSerial.listen();
    sttime = millis();
    read = 0;
    while ((read != 1) && (millis() - sttime < 50))
    {
        if(lcdSerial.available() != 0)
        {
            readx = lcdSerial.read();
            read = 1;
        }
    }
}
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <TextView
        android:id="@+id/title_paired_devices"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="@string/title_paired_devices"
        android:visibility="gone"
        android:background="#666"
        android:textColor="#fff"
        android:paddingLeft="5dp"/>

    <ListView
        android:id="@+id/paired_devices"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:stackFromBottom="true"
        android:layout_weight="1"/>

    <TextView
        android:id="@+id/title_new_devices"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="@string/title_other_devices"
        android:visibility="gone"
        android:background="#666"
        android:textColor="#fff"
        android:paddingLeft="5dp"/>

    <ListView
        android:id="@+id/new_devices"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:stackFromBottom="true"
        android:layout_weight="2"/>

    <Button
        android:id="@+id/button_scan"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="@string/button_scan"/>

</LinearLayout>
// File: main.xml
//
// Authors: Drew Bentz / William Budney
// Advisor: Bridget Benson
// CPE Senior Project
// Cal Poly SLO
// Spring 2013
//
// Function: Layout for the main program
//

<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="#000000"
    android:gravity="center"
    tools:context=".Bluetooth"
>
    <Button
        android:id="@+id/button1"
        style="?android:attr/buttonStyleSmall"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignBaseline="@+id/editText1"
        android:layout_alignTop="@+id/editText1"
        android:layout_alignParentRight="true"
        android:text="Send" />

    <Button
        android:id="@+id/sendAddress"
        style="?android:attr/buttonStyleSmall"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_below="@+id/button1"
        android:layout_alignParentRight="true"
        android:layout_alignTop="@+id/editText2"
        android:text="Send" />

    <ImageButton
        android:id="@+id/imageButton1"
        android:layout_width="wrap_content"
        android:layout_height="207dp"
        android:cropToPadding="false"
        android:src="@drawable/bt_button_nc" />

    <EditText
        android:id="@+id/editText1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginTop="50dp"
        android:layout_marginRight="14dp"
        android:layout_marginLeft="14dp"/>
<ImageButton
    android:id="@+id/right"
    style="?android:attr/buttonStyleSmall"
    android:layout_width="66dp"
    android:layout_height="66dp"
    android:layout_toRightOf="@+id/straight"
    android:layout_alignTop="@+id/straight"
    android:src="@drawable/right" />

<ImageButton
    android:id="@+id/back"
    style="?android:attr/buttonStyleSmall"
    android:layout_width="66dp"
    android:layout_height="66dp"
    android:layout_centerHorizontal="true"
    android:layout_below="@+id/straight"
    android:src="@drawable/uturn" />
</RelativeLayout>
</SlidingDrawer>
</RelativeLayout>
package com.example.bluetooth;
import android.os.Bundle;
import android.os.Handler;
import android.os.Message;
import android.accessibilityservice.AccessibilityService;
import android.accessibilityservice.AccessibilityServiceInfo;
import android.app.Activity;
import android.bluetooth.BluetoothAdapter;
import android.bluetooth.BluetoothDevice;
import android.content.Intent;
import android.util.Log;
import android.view.KeyEvent;
import android.view.Menu;
import android.view.MenuItem;
import android.view.View;
import android.view.accessibility.AccessibilityEvent;
import android.view.inputmethod.EditorInfo;
import android.widget.ArrayAdapter;
import android.widget.Button;
import android.widget.EditText;
import android.widget.ImageView;
import android.widget.ListView;
import android.widget.SlidingDrawer;
import android.widget.SlidingDrawer.OnDrawerCloseListener;
import android.widget.SlidingDrawer.OnDrawerOpenListener;
import android.widget.TextView;
import android.widget.Toast;

public class Bluetooth extends AbstractActivity {
    // Debugging
    private static final String TAG = "BluetoothChat";
    private static final boolean D = true;

    // Intent request codes
    private static final int REQUEST_CONNECT_DEVICE_SECURE = 1;
    private static final int REQUEST_ENABLE_BT = 3;

    // Message types sent from the BluetoothChatService Handler
    public static final int MESSAGE_STATE_CHANGE = 1;
    public static final int MESSAGE_READ = 2;
    public static final int MESSAGE_WRITE = 3;
    public static final int MESSAGE_DEVICE_NAME = 4;
public static final int MESSAGE_TOAST = 5;

// Key names received from the BluetoothChatService Handler
public static final String DEVICE_NAME = "device_name";
public static final String TOAST = "toast";

// Turning commands
public static final String LEFT = "@L$";
public static final String RIGHT = "@R$";
public static final String STRAIGHT = "@S$";
public static final String BACK = "@U$";

// Street command
public static final String START_DELIM_STREET = "%";
public static final String END_DELIM_STREET = "$";

private ListView myConversationView;
private ImageView btButton;
private EditText inputText;
private EditText addressText;
private Button mySendButton;
private Button sendAddress;
private ImageView rightButton;
private ImageView leftButton;
private ImageView straightButton;
private ImageView backButton;
final CharSequence text = "Connecting...";

// Name of the connected device
private String myConnectedDeviceName = null;
// Array adapter for the conversation thread
private ArrayAdapter<String> myConversationArrayAdapter;
// Local Bluetooth adapter
private BluetoothAdapter myBluetoothAdapter = null;
// Member object for the chat services
public BluetoothService myChatService = null;
// String buffer for outgoing messages
private StringBuffer myOutStringBuffer;

SlidingDrawer slidingDrawer;
Button slideButton;

@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);
    btButton = (ImageView) findViewById(R.id.imageButton1);
    inputText = (EditText) findViewById(R.id.editText1);
    btButton.setOnClickListener(new OnClickListener(){
        public void onClick(View v){
            onConnect();
        }
    });
    slideButton = (Button) findViewById(R.id.handle);
    slidingDrawer = (SlidingDrawer) findViewById(R.id.drawer);
slidingDrawer.setOnDrawerOpenListener(new OnDrawerOpenListener() {
    @Override
    public void onDrawerOpened() {
    }
});

slidingDrawer.setOnDrawerCloseListener(new OnDrawerCloseListener() {
    @Override
    public void onDrawerClosed() {
    }
});

// Get local Bluetooth adapter
myBluetoothAdapter = BluetoothAdapter.getDefaultAdapter();

// If the adapter is null, then Bluetooth is not supported
if (myBluetoothAdapter == null) {
    Toast.makeText(this, "Bluetooth is not available",
    Toast.LENGTH_LONG).show();
    finish();
    return;
}

@Override
public void onStart() {
    super.onStart();
    if(D) Log.e(TAG, "++ ON START ++");

    // If BT is not on, request that it be enabled.
    // setupChat() will then be called during onActivityResult
    if (!myBluetoothAdapter.isEnabled()) {
        Intent enableIntent = new Intent(BluetoothAdapter.ACTION_REQUEST_ENABLE);
        startActivityForResult(enableIntent, REQUEST_ENABLE_BT);
    // Otherwise, setup the chat session
    } else {
        if (myChatService == null) setupChat();
    }
}

@Override
public synchronized void onResume() {
    super.onResume();
    if(D) Log.e(TAG, "+ ON RESUME +");

    //Performing this check in onResume() covers the case in which BT
    was
    //not enabled during onStart(), so we were paused to enable it...
    //onResume() will be called when ACTION_REQUEST_ENABLE activity
    return

    }
if (myChatService != null) {
    // Only if the state is STATE_NONE, do we know that we haven't
    // started already
    if (myChatService.getState() == BluetoothService.STATE_NONE) {
        // Start the Bluetooth chat services
        myChatService.start();
    }
}

private void setupChat() {
    Log.d(TAG, "setupChat()");
    // Initialize the array adapter for the conversation thread
    myConversationArrayAdapter = new ArrayAdapter<String>(this,
            R.layout.message);
    // Initialize the compose field with a listener for the return key
    inputText = (EditText) findViewById(R.id.editText1);
    inputText.setOnEditorActionListener(mWriteListener);
    addressText = (EditText) findViewById(R.id.editText2);
    addressText.setOnEditorActionListener(addressListener);
    // Initialize the send button with a listener that for click
    events
    mySendButton = (Button) findViewById(R.id.button1);
    mySendButton.setOnClickListener(new OnClickListener() {
        public void onClick(View v) {
            // Send a message using content of the edit text widget
            TextView view = (TextView) findViewById(R.id.editText1);
            String message = START_DELIM_STREET +
                    (view.getText().toString()) + END_DELIM_STREET;
            sendMessage(message);
        }
    });
    sendAddress = (Button) findViewById(R.id.sendAddress);
    sendAddress.setOnClickListener(new OnClickListener() {
        public void onClick(View v) {
        }
    });
    leftButton = (ImageView) findViewById(R.id.left);
    leftButton.setOnClickListener(new OnClickListener() {
        public void onClick(View v) {
            sendMessage(LEFT);
        }
    });
    rightButton = (ImageView) findViewById(R.id.right);
    rightButton.setOnClickListener(new OnClickListener() {
        public void onClick(View v) {
            sendMessage(RIGHT);
        }
    });
straightButton = (ImageView) findViewById(R.id.straight);
straightButton.setOnClickListener(new OnClickListener() {
    public void onClick(View v) {
        sendMessage(STRAIGHT);
    }
});

backButton = (ImageView) findViewById(R.id.back);
backButton.setOnClickListener(new OnClickListener() {
    public void onClick(View v) {
        sendMessage(BACK);
    }
});

// Initialize the BluetoothChatService to perform bluetooth
collections
myChatService = new BluetoothService(this, mHandler);

// Initialize the buffer for outgoing messages
myOutStringBuffer = new StringBuffer(""");

// The action listener for the EditText widget, to listen for the return
key
private TextView.OnEditorActionListener mWriteListener =
    new TextView.OnEditorActionListener() {
        public boolean onEditorAction(TextView view, int actionId,
            KeyEvent event) {
            // If the action is a key-up event on the return key, send the
            message
            if (actionId == EditorInfo.IME_NULL && event.getAction() ==
                KeyEvent.ACTION_UP) {
                String message = view.getText().toString();
                sendMessage(message);
            }
            if(D) Log.i(TAG, "END onEditorAction");
            return true;
        }
    };

// The action listener for the EditText widget, to listen for the return
key
private TextView.OnEditorActionListener addressListener =
    new TextView.OnEditorActionListener() {
        public boolean onEditorAction(TextView view, int actionId,
            KeyEvent event) {
            // If the action is a key-up event on the return key, send the
            message
            if (actionId == EditorInfo.IME_NULL && event.getAction() ==
                KeyEvent.ACTION_UP) {
                String message = view.getText().toString();
sendMessage(message);

if(D) Log.i(TAG, "END onEditorAction");
return true;

/**
 * Sends a message.
 * @param message  A string of text to send.
 */
private void sendMessage(String message) {
    // Check that we're actually connected before trying anything
    if (myChatService.getState() != BluetoothService.STATE_CONNECTED) {
        Toast.makeText(this, R.string.not_connected,
            Toast.LENGTH_SHORT).show();
        btButton.setImageResource(R.drawable.bt_button_nc);
        return;
    }
    // Check that there's actually something to send
    if (message.length() > 0) {
        // Get the message bytes and tell the BluetoothChatService to write
        byte[] send = message.getBytes();
        myChatService.write(send);

        // Reset out string buffer to zero and clear the edit text field
        myOutStringBuffer.setLength(0);
        inputText.setText(myOutStringBuffer);
    }
}

private final Handler mHandler = new Handler() {
    @Override
    public void handleMessage(Message msg) {
        switch (msg.what) {
        case MESSAGE_STATE_CHANGE:
            if(D) Log.i(TAG, "MESSAGE_STATE_CHANGE: " + msg.arg1);
            switch (msg.arg1) {
            case BluetoothService.STATE_CONNECTED:
                btButton.setImageResource(R.drawable.bt_button_c);
                myConversationArrayAdapter.clear();
                break;
            case BluetoothService.STATE_CONNECTING:
                //setStatus(R.string.title_connecting);
                break;
            case BluetoothService.STATE_LISTEN:
            case BluetoothService.STATE_NONE:
                btButton.setImageResource(R.drawable.bt_button_nc);
                break;
            }
        }
case MESSAGE_WRITE:
    byte[] writeBuf = (byte[]) msg.obj;
    // construct a string from the buffer
    String writeMessage = new String(writeBuf);
    myConversationArrayAdapter.add("Me: " + writeMessage);
    break;

case MESSAGE_READ:
    byte[] readBuf = (byte[]) msg.obj;
    // construct a string from the valid bytes in the buffer
    String readMessage = new String(readBuf, 0, msg.arg1);
    myConversationArrayAdapter.add(myConnectedDeviceName + ": " + readMessage);
    break;

case MESSAGE_DEVICE_NAME:
    // save the connected device's name
    myConnectedDeviceName = msg.getData().getString(DEVICE_NAME);
    Toast.makeText(getApplicationContext(), "Connected to " + myConnectedDeviceName,
    Toast.LENGTH_SHORT).show();
    btButton.setImageResource(R.drawable.bt_button_c);
    break;

case MESSAGE_TOAST:
    Toast.makeText(getApplicationContext(), msg.getData().getString(TOAST),
    Toast.LENGTH_SHORT).show();
    if(msg.getData().getString(TOAST) == "Device connection was lost"){
        btButton.setImageResource(R.drawable.bt_button_nc);
    }
    break;
}
}

public void onActivityResult(int requestCode, int resultCode, Intent data) {
    if(D) Log.d(TAG, "onActivityResult " + resultCode);
    switch (requestCode) {
    case REQUEST_CONNECT_DEVICE_SECURE:
        // When DeviceListActivity returns with a device to connect
        if (resultCode == Activity.RESULT_OK) {
            connectDevice(data, true);
        } else {
            // User did not enable Bluetooth or an error occurred
            Log.d(TAG, "BT not enabled");
        }
        break;
    case REQUEST_ENABLE_BT:
        // When the request to enable Bluetooth returns
        if (resultCode == Activity.RESULT_OK) {
            // Bluetooth is now enabled, so set up a chat session
            setupChat();
        } else {
            // User did not enable Bluetooth or an error occurred
        Log.d(TAG, "BT not enabled");
        }
Toast.makeText(this, R.string.bt_not_enabled_leaving,
Toast.LENGTH_SHORT).show();
finish();
}

private void connectDevice(Intent data, boolean secure) {
    // Get the device MAC address
    String address = data.getExtras()
        .getString(DeviceListActivity.EXTRA_DEVICE_ADDRESS);
    // Get the BluetoothDevice object
    BluetoothDevice device =
        myBluetoothAdapter.getRemoteDevice(address);
    // Attempt to connect to the device
    myChatService.connect(device, secure);
}

public void onConnect() {
    Intent serverIntent = null;
    serverIntent = new Intent(this, DeviceListActivity.class);
    startActivityForResult(serverIntent, REQUEST_CONNECT_DEVICE_SECURE);
}

@Override
public boolean onCreateOptionsMenu(Menu menu) {
    // Inflate the menu; this adds items to the action bar if it is
    // present.
    getMenuInflater().inflate(R.menu.bluetooth, menu);
    return true;
}

@override
public boolean onCreateOptionsMenu(Menu menu) {
    // Inflate the menu; this adds items to the action bar if it is
    // present.
    getMenuInflaterInflater().inflate(R.menu.bluetooth, menu);
    return true;
}
package com.example.bluetooth;
import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;
import java.util.UUID;
import android.bluetooth.BluetoothAdapter;
import android.bluetooth.BluetoothDevice;
import android.bluetooth.BluetoothServerSocket;
import android.bluetooth.BluetoothSocket;
import android.content.Context;
import android.os.Bundle;
import android.os.Handler;
import android.os.Message;
import android.util.Log;
import android.widget.ImageView;

public class BluetoothService {
    private static final String TAG = "BluetoothService";
    // Name for the SDP record when creating server socket
    private static final String NAME_SECURE = "motoHUD";

    // Unique UUID for bluetooth device
    private static final UUID MY_UUID = UUID.fromString("00001101-0000-1000-8000-00805F9B34FB");

    // Members
    private final BluetoothAdapter myAdapter;
    private final Handler myHandler;
    private AcceptThread myAcceptThread;
    private ConnectThread myConnectThread;
    private ConnectedThread myConnectedThread;
    private int myState;

    // Constants for states
    public static final int STATE_NONE = 0; // Doing nothing
    public static final int STATE_LISTEN = 1; // Listening
    public static final int STATE_CONNECTING = 2; // Initiating an outgoing connection
    public static final int STATE_CONNECTED = 3; // Connected
private ImageView btButton;
/
* Constructor. Prepares a new Bluetooth session.
* @param context The UI Activity Context
* @param handler A Handler to send messages back to the UI Activity
*/
public BluetoothService(Context context, Handler handler) {
    myAdapter = BluetoothAdapter.getDefaultAdapter();
    myState = STATE_NONE;
    myHandler = handler;
}

/*
* Set the current state of the chat connection
* @param state An integer defining the current connection state
*/
private synchronized void setState(int state) {
    myState = state;
    // Give the new state to the Handler so the UI Activity can update
    myHandler.obtainMessage(Bluetooth.MESSAGE_STATE_CHANGE, state, -1).sendToTarget();
}

/*
* Return the current connection state.
*/
public synchronized int getState() {
    return myState;
}

/*
* Start the chat service. Specifically start AcceptThread to begin a
* session in listening (server) mode. Called by the Activity
* onResume()            */
public synchronized void start() {
    // Cancel any thread attempting to make a connection
    if (myConnectThread != null) {
        myConnectThread.cancel();
        myConnectThread = null;
    }
    // Cancel any thread currently running a connection
    if (myConnectedThread != null) {
        myConnectedThread.cancel();
        myConnectedThread = null;
    }
    setState(STATE_LISTEN);
    // Start the thread to listen on a BluetoothServerSocket
    if (myAcceptThread == null) {
        myAcceptThread = new AcceptThread(true);
        myAcceptThread.start();
    }
*/
  * Start the ConnectThread to initiate a connection to a remote device.
  * @param device The BluetoothDevice to connect
  * @param secure Socket Security type - Secure (true), Insecure (false)
  */
public synchronized void connect(BluetoothDevice device, boolean secure) {
    // Cancel any thread attempting to make a connection
    if (myState == STATE_CONNECTING) {
        if (myConnectThread != null) {
            myConnectThread.cancel();
            myConnectThread = null;
        }
    }
    // Cancel any thread currently running a connection
    if (myConnectedThread != null) {
        myConnectedThread.cancel();
        myConnectedThread = null;
    }
    // Start the thread to connect with the given device
    myConnectThread = new ConnectThread(device, secure);
    myConnectThread.start();
    setState(STATE_CONNECTING);
}

*/
  * Start the ConnectedThread to begin managing a Bluetooth connection
  * @param socket The BluetoothSocket on which the connection was made
  * @param device The BluetoothDevice that has been connected
  */
public synchronized void connected(BluetoothSocket socket, BluetoothDevice device) {
    // Cancel the thread that completed the connection
    if (myConnectThread != null) {
        myConnectThread.cancel();
        myConnectThread = null;
    }
    // Cancel any thread currently running a connection
    if (myConnectedThread != null) {
        myConnectedThread.cancel();
        myConnectedThread = null;
    }
    // Cancel the accept thread because we only want to connect to one device
    if (myAcceptThread != null) {
        myAcceptThread.cancel();
        myAcceptThread = null;
    }
    // Start the thread to manage the connection and perform transmissions
    myConnectedThread = new ConnectedThread(socket);
myConnectedThread.start();
// Send the name of the connected device back to the UI Activity
Message msg = myHandler.obtainMessage(Bluetooth.MESSAGE_DEVICE_NAME);
Bundle bundle = new Bundle();
bundle.putString(Bluetooth.DEVICE_NAME, device.getName());
msg.setData(bundle);
myHandler.sendMessage(msg);

setState(STATE_CONNECTED);
}

/*
 * Stop all threads
 */
public synchronized void stop() {
// Set members to NULL
if (myConnectThread != null) {
    myConnectThread.cancel();
    myConnectThread = null;
}
if (myConnectedThread != null) {
    myConnectedThread.cancel();
    myConnectedThread = null;
}
if (myAcceptThread != null) {
    myAcceptThread.cancel();
    myAcceptThread = null;
}
setState(STATE_NONE);
}

/*
 * Write to the ConnectedThread in an unsynchronized manner
 * @param out The bytes to write
 * @see ConnectedThread#write(byte[])
 */
public void write(byte[] out) {
    // Create temporary object
    ConnectedThread r;
    // Synchronize a copy of the ConnectedThread
    synchronized (this) {
        // Make sure we are connected
        if (myState != STATE_CONNECTED)
            return;
        r = myConnectedThread;
    }
    // Perform the write unsynchronized
    r.write(out);
}

/*
 * Indicate that the connection attempt failed and notify the UI Activity.
 */
private void connectionFailed() {
// Send a failure message back to the Activity
Message msg = myHandler.obtainMessage(Bluetooth.MESSAGE_TOAST);
Bundle bundle = new Bundle();
bundle.putString(Bluetooth.TOAST, "Unable to connect device");
msg.setData(bundle);
myHandler.sendMessage(msg);

// Start the service over to restart listening mode
BluetoothService.this.start();

private void connectionLost() {
    // Send a failure message back to the Activity
    Message msg = myHandler.obtainMessage(Bluetooth.MESSAGE_TOAST);
    Bundle bundle = new Bundle();
bundle.putString(Bluetooth.TOAST, "Device connection was lost");
msg.setData(bundle);
myHandler.sendMessage(msg);

    // Start the service over to restart listening mode
    BluetoothService.this.start();
}

private class AcceptThread extends Thread {

    private final BluetoothServerSocket myServerSocket;
    //private String mySocketType;

    public AcceptThread(boolean secure) {
        BluetoothServerSocket tmp = null;
        // Create a new listening server socket
        try {
            tmp = myAdapter.listenUsingRfcommWithServiceRecord(NAME_SECURE,
                    MY_UUID);
        } catch (IOException e) {
            Log.e(TAG, "Socket Type: listen() failed", e);
        }
        myServerSocket = tmp;
    }

    public void run() {
        BluetoothSocket socket = null;

        // Listen to the server socket if we're not connected
while (myState != STATE_CONNECTED) {
    try {
        // This is a blocking call and will only return on a
        // successful connection or an exception
        socket = myServerSocket.accept();
    } catch (IOException e) {
        Log.e(TAG, "Socket Type: accept() failed", e);
        break;
    }

    // If a connection was accepted
    if (socket != null) {
        synchronized (BluetoothService.this) {
            switch (myState) {
                case STATE_LISTEN:
                    break;
                case STATE_CONNECTING:
                    // Situation normal. Start the connected
                    connected(socket,
                            socket.getRemoteDevice());
                    break;
                case STATE_NONE:
                    break;
                case STATE_CONNECTED:
                    // Either not ready or already connected.
                    Terminate new socket.
                    try {
                        socket.close();
                    } catch (IOException e) {
                        Log.e(TAG, "Could not close unwanted
                        socket", e);
                    }
                    break;
            }
        }
    }
}

public void cancel() {
    try {
        myServerSocket.close();
    } catch (IOException e) {
        Log.e(TAG, "Socket Type close() of server failed", e);
    }
}

/*
 * This thread runs while attempting to make an outgoing connection
 * with a device. It runs straight through; the connection either
 * succeeds or fails.
 */
private class ConnectThread extends Thread {


private final BluetoothSocket mySocket;
private final BluetoothDevice myDevice;

public ConnectThread(BluetoothDevice device, boolean secure) {
    myDevice = device;
    BluetoothSocket tmp = null;
    try {
        tmp = device.createRfcommSocketToServiceRecord(
                MY_UUID);
    } catch (IOException e) {
        Log.e(TAG, "Socket Type: create() failed", e);
    }
    mySocket = tmp;
}

public void run() {
    // Always cancel discovery because it will slow down a
    connection
    myAdapter.cancelDiscovery();

    // Make a connection to the BluetoothSocket
    try {
        // This is a blocking call and will only return on a
        // successful connection or an exception
        mySocket.connect();
    } catch (IOException e) {
        // Close the socket
        try {
            mySocket.close();
        } catch (IOException e2) {
            Log.e(TAG, "unable to close() socket during connection
        failure", e2);
        }
        connectionFailed();
        return;
    }

    // Reset the ConnectThread because we're done
    synchronized (BluetoothService.this) {
        myConnectThread = null;
    }

    // Start the connected thread
    connected(mySocket, myDevice);
}

public void cancel() {
    try {
        mySocket.close();
    } catch (IOException e) {
        Log.e(TAG, "close() of connect socket failed", e);
    }
}
private class ConnectedThread extends Thread {
    private final BluetoothSocket mySocket;
    private final InputStream myInStream;
    private final OutputStream myOutStream;

    public ConnectedThread(BluetoothSocket socket) {
        Log.d(TAG, "create ConnectedThread");
        mySocket = socket;
        InputStream tmpIn = null;
        OutputStream tmpOut = null;

        try {
            tmpIn = socket.getInputStream();
            tmpOut = socket.getOutputStream();
        } catch (IOException e) {
            Log.e(TAG, "temp sockets not created", e);
        }

        myInStream = tmpIn;
        myOutStream = tmpOut;
    }

    public void run() {
        Log.i(TAG, "BEGIN mConnectedThread");
        byte[] buffer = new byte[1024];
        int bytes;
        // Keep listening to the InputStream while connected
        while (true) {
            try {
                // Read from the InputStream
                bytes = myInStream.read(buffer);
            } catch (IOException e) {
                Log.e(TAG, "disconnected", e);
                connectionLost();
                // Start the service over to restart listening mode
                BluetoothService.this.start();
                break;
            }
        }
    }

    /*
     * Write to the connected OutputStream.
     * @param buffer The bytes to write
     */
    public void write(byte[] buffer) {
        try {
            myOutStream.write(buffer);
        } catch (IOException e) {
            Log.e(TAG, "write failed", e);
        }
    }
}
} catch (IOException e) {
    Log.e(TAG, "Exception during write", e);
}

public void cancel() {
    try {
        mySocket.close();
    } catch (IOException e) {
        Log.e(TAG, "close() of connect socket failed", e);
    }
    
}
Preparing motoHUD

**Turning on the Device**

**STEP 1**
Simply flip the switch. If the red LED on the switch is on, this means the device is running.

**STEP 2**
Allow five seconds for the OLED to initialize and display a speed of 000. Note: It may take up to four minutes for the GPS to receive a fix and display correct data.

**Connecting device to your phone**

**STEP 1**
Locate and press the button on the top of the application. If the button has a red background with a clear Bluetooth image, you are not connected to the device.

**STEP 2**
A list of Bluetooth enabled devices will appear. Click on motoHUD and enter the following password: 1234. The button should now be green with a solid Bluetooth image.

*Figure 12: User Manual*
G. Datasheets
2.0” Transparent OLED Display

*microOLED GOLDELOX Module

µTOLED-20-G2

Document Date: 13th February 2013
Document Revision: 1.1
Overview

The µTOLED-20-G2 display module flaunts a very impressive 2.0” state of the art Passive Matrix Transparent OLED (PMTOLED) technology that is fully transparent and viewable from both the front and rear. This display module is powered with an embedded GOLDELOX graphics processor that delivers ‘stand-alone’ functionality to any project.

The module is an elegant combination of a fully transparent PMTOLED Passive Matrix Screen, along with a modest but comprehensive collection of I/O Features. These include a micro-SD card connector, two general purpose input/output pins (GPIO’s) with Dallas 1-Wire Support, Analog Input and sound generation capability, along with serial communications.

Embedded at the heart of the design is the GOLDELOX processor, which is driven by a highly optimised virtual core engine called EVE (Extensible Virtual Engine).

This display module serves as a perfect solution to be deployed at the forefront of any product design, requiring a brilliance of colour, animation or images on any application. This GOLDELOX driven Intelligent Display Module is a perfect example of where art meets technology.

The µTOLED-20-G2 has a 2.0” PMTOLED Display at the forefront of the design that showcases the power and capabilities of the GOLDELOX processor. Combining a resolution of 128x160 pixels with 65K colours, this display module is perfect for any application requiring a transparent display.

Basic audio generation is possible on the module through the use of RTTTL sounds and beeps. These can be executed easily with a simple command, and are non-blocking which enables user code to be executed while audio is playing.

The micro-SD card slot on the module provides the user with expandable memory space suitable for multimedia file retrieval, including images, animations and movie clips, as well as data logging applications.

This microOLED module can be programmed in its native 4DGL language (similar to C), using the Workshop 4 IDE Software tool suite. It can also be configured very easily as a serial slave device, for use with your favourite host controller. Freedom is at your fingertips with the intelligent µTOLED-20-G2 transparent display module.
# Contents

1. Description .......................................................................................................................... 4

2. Features ................................................................................................................................. 4

3. Pin Configuration and Summary ............................................................................................. 5

4. Hardware Interface - Pins ....................................................................................................... 6
   4.1. Serial Ports - COM0 UART .............................................................................................. 6
   4.2. General Purpose I/O ........................................................................................................ 6
   4.3. System Pins ..................................................................................................................... 7

5. PmmC/Firmware Programming ............................................................................................... 8

6. Module Features ..................................................................................................................... 8
   6.1. Display – 2.0” PMTOLED ............................................................................................... 8
   6.2. GOLDELOX Processor .................................................................................................... 8
   6.3. SD/SDHC Memory Cards ............................................................................................... 9

7. OLED Screen Precautions ...................................................................................................... 9

8. Hardware Tools ..................................................................................................................... 10
   8.1. 4D Programming Cable .................................................................................................. 10

9. 4DGL - Software Language ................................................................................................... 10

10. 4D Systems - Workshop 4 IDE ............................................................................................ 11
    10.1. Workshop 4 – Designer Environment .......................................................................... 11
    10.2. Workshop 4 – ViSi Environment ................................................................................ 11
    10.3. Workshop 4 – Serial Environment ............................................................................. 12

11. Mechanical Details ................................................................................................................ 13

12. Specifications and Ratings .................................................................................................... 14

13. Legal Notice .......................................................................................................................... 16

14. Contact Information .............................................................................................................. 16
1. Description

The μTOLED-20-G2 is an impressive Transparent OLED display module (TOLED) in the 4D Systems microOLED graphics display range. Featuring a 2.0" 128x160 resolution transparent display, it is the ideal size for attractive transparent display applications.

Driving the module and its peripherals is the GOLDELOX processor, a very capable chip which provides impressive graphics power, programmed with 4D Systems Workshop IDE Software. 4D Systems Workshop enables graphic solutions to be constructed rapidly and with ease due to its design being solely for 4D’s graphics controllers.

The μTOLED-20-G2 has a modest but comprehensive range of features suited for an application requiring an amazing transparent display, an analog input, Dallas 1-wire sensor capability, audio generation, or simply digital I/O. This is truly an impressive display solution.

2. Features

- Low-cost 2.0” TOLED display graphics user interface solution.
- 128 x 160 resolution, RGB 65K colour, Passive Matrix TOLED Screen.
- No back lighting with near 360° viewing angle (NOTE: see Section 6.1)
- Easy 10 pin interface to any host device: 3.3Vout, IO2, GND, IO1, RESET, GND, RX, TX, +5V, 5V OUT
- Powered by the 4D-Labs GOLDELOX processor (also available as separate OEM IC)
- 10KB of flash memory for user code storage and 510 bytes of RAM for user variables (255 x 16bit vars).
- Asynchronous hardware serial UART port with auto-baud, TTL interface, with 300 to 600K baud.
- 2 x General Purpose I/O pins.
  - IO1 supports:
    - Digital I/O
    - A/D Converter with 8/10 bit resolution
    - Complex sound generation
    - Dedicated RTTTL tune engine
    - Multi-Switch Joystick, Buttons
    - Dallas 1-Wire
  - IO2 supports:
    - Digital I/O
    - Complex sound generation
    - Dedicated RTTTL tune engine
    - Dallas 1-Wire
- On-board micro-SD memory card adaptor for multimedia storage and data logging purposes. HC memory card support is also available for cards larger than 4GB.
- Built in extensive 4DGL graphics and system library functions.
- Display full colour images, animations, icons and video clips (no audio support for video).
- Supports all available Windows fonts and characters (imported as external fonts).
- 4.0V to 5.5V range operation (single supply), nominal 5.0V.
- PCB Module dimensions: 37.5 x 24.0 x 11.0mm
  - Display dimensions: 37.79 x 68.74 x 1.4mm
- Weight ~ 12g.
- Display Viewing Area: 31.86 x 39.74mm
- RoHS Compliant.
3. Pin Configuration and Summary

<table>
<thead>
<tr>
<th>Pin</th>
<th>Symbol</th>
<th>I/O</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>+5V</td>
<td>P</td>
<td>Main Voltage Supply +ve input pin. Reverse polarity protected. Range is 4.0V to 5.5V, nominal 5.0V.</td>
</tr>
<tr>
<td>2</td>
<td>5V OUT</td>
<td>P</td>
<td>5V OUT provides approximately 4.7 V through a protection diode.</td>
</tr>
<tr>
<td>3</td>
<td>TX</td>
<td>O</td>
<td>Asynchronous Serial Transmit pin. Output data is at TTL voltage levels. Connect this pin to external device Serial Receive (Rx) signal. This pin is tolerant up to 5.0V levels.</td>
</tr>
<tr>
<td>4</td>
<td>IO2</td>
<td>I/O</td>
<td>General purpose IO2 pin. See section 2.2 for more detail.</td>
</tr>
<tr>
<td>5</td>
<td>RX</td>
<td>I</td>
<td>Asynchronous Serial Receive pin. Connect this pin to external device Serial Transmit (Tx) signal. This pin is tolerant up to 5.0V levels.</td>
</tr>
<tr>
<td>6</td>
<td>IO1</td>
<td>I/O</td>
<td>General purpose IO1 pin. See section 2.2 for more detail.</td>
</tr>
<tr>
<td>7</td>
<td>GND</td>
<td>P</td>
<td>Supply Ground.</td>
</tr>
<tr>
<td>8</td>
<td>GND</td>
<td>P</td>
<td>Supply Ground.</td>
</tr>
<tr>
<td>9</td>
<td>RESET</td>
<td>I</td>
<td>Master Reset signal. Internally pulled up to 3.3V via a 4.7K resistor. An active Low pulse greater than 2.0μs will reset the module. If the module needs to be reset externally, only use open collector type circuits. This pin is not driven low by any internal conditions. The host should control this pin via one of its port pins using an open collector/drain arrangement.</td>
</tr>
<tr>
<td>10</td>
<td>3.3V</td>
<td>P</td>
<td>Regulated 3.3 Volts output, maximum available current 50mA to power external circuitry.</td>
</tr>
</tbody>
</table>

I = Input, O = Output, P = Power
4. Hardware Interface - Pins

The μTOLED-20-G2 provides both a hardware and software interface. This section describes in detail the hardware interface pins of the device.

4.1. Serial Ports - COM0 UART

The μTOLED-20-G2 has a dedicated hardware UART that can communicate with external serial devices.

The primary features are:
- Full-Duplex 8 bit data transmission and reception through the TX and RX pins.
- Data format: 8 bits, No Parity, 1 Stop bit.
- Auto Baud feature.
- Baud rates from 300 baud up to 600K baud.

The Serial port is also the primary interface for downloading compiled 4DGL application code as well as future PmmC/Firmware updates for the onboard GOLDELOX processor. Refer to Section 5. PmmC/Firmware Programming for more details.

TX pin 3 (Serial Transmit):
Asynchronous Serial port Transmit pin, TX. The serial output data is at TTL voltage levels. Connect this pin to external serial device Rx signal.

RX pin 5 (Serial Receive):
Asynchronous Serial port Receive pin, RX. Connect this pin to external serial device Transmit Tx signal.

4.2. General Purpose I/O

There are 2 GPIO pins available, IO1 and IO2. Each GPIO has a multitude of high level functions associated with it and these can be selected within 4DGL user application code.

Refer to the separate document titled "GOLDELOX-4DGL-Internal-Functions.pdf" for a complete set of built in 4DGL library functions.

IO1 pin 6 (General Purpose IO1):
General purpose IO1 pin. The following table lists the available GPIO functions and features.

IO2 pin 4 (General Purpose IO2):
General purpose IO2 pin. The following table lists the available GPIO functions and features.

Note: GPIO pins are 5.0V tolerant.
Unused buttons do not need resistors to be connected to the circuit. Table below lists the buttons and corresponding resistor values.

<table>
<thead>
<tr>
<th>Number of Buttons</th>
<th>Button Number</th>
<th>Resistor Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>SW1</td>
<td>22K</td>
</tr>
<tr>
<td>2</td>
<td>SW2</td>
<td>10K</td>
</tr>
<tr>
<td>3</td>
<td>SW3</td>
<td>4.7K</td>
</tr>
<tr>
<td>4</td>
<td>SW4</td>
<td>2.2K</td>
</tr>
<tr>
<td>5</td>
<td>SW5</td>
<td>1.2K</td>
</tr>
</tbody>
</table>

Dallas 1-Wire:
The Dallas 1-Wire protocol is a form of serial communications designed to operate over a single data line plus ground reference. Multiple 1 Wire devices can be attached to the same shared data line to network many devices. One wire device support is available on both the IO1 and the IO2 pins on the μTOLED-20-G2 module.

The following diagram depicts a typical 1-Wire temperature sensor interface.

Sound Output:
The μTOLED-20-G2 module is capable of generating complex sounds and RTTTL tunes from its IO1 and IO2 pins. A simple speaker circuit as shown below can be utilised.

4.3. System Pins

+5V (Module Voltage Input)
Pin 1:
Module supply voltage input pin. This pin must be connected to a regulated supply voltage in the range of 4.0 Volts to 5.5 Volts DC. Nominal operating voltage is 5.0 Volts.

5V Out (~4.7V)
Pin 2:
External circuitry that requires approximately 5V supply can be powered up via this pin. Maximum available current is 50mA.

GND (Module Ground)
Pins 7, 8:
Device ground pins. These pins must be connected to ground.

RESET (Module Master Reset)
Pin 9:
Module Master Reset pin. An active low pulse of greater than 2 micro-seconds will reset the module. Internally pulled up to 3.3V via a 4.7K resistor. Only use open collector type circuits to reset the device if an external reset is required.

3.3Vout (3.3V Regulated Output)
Pin 10:
External circuitry that requires a regulated 3.3V supply can be powered up via this pin. Maximum available current is 50mA.
5. PmmC/Firmware Programming

The GOLDELOX processor on the µTOLED-20-G2 module can be re-programmed with the latest PmmC configuration for updates and future proofing. The chip-level configuration is available as a PmmC (Personality-module-micro-Code) file and the programming must be performed over the serial interface. The chip-resident internal 4DGL functions are part of the GOLDELOX PmmC configuration file so please check regularly for the latest updates and enhancements.

It is recommended that the µTOLED-20-G2 display module be socketed on the application board so that it can be easily removed for PmmC programming.

The PmmC file is programmed into the device with the aid of Workshop 4, the 4D Systems IDE software (See Section 10). To provide a link between the PC and the ICSP interface, a specific 4D Programming Cable is required and is available from 4D Systems.

Using a non-4D programming interface could damage your display, and void your Warranty.

6. Module Features

The µTOLED-20-G2 module is designed to accommodate most applications. Some of the main features of the module are listed below.

6.1. Display – 2.0” PMTOLED

The µTOLED-20-G2 is equipped with a full colour Passive Matrix TOLED screen. Some of the features of the screen are:

- Screen Size: 2.0” diagonal
- 128 x 160 Resolution PMTOLED Display
- Screen Dimensions: 37.79 x 68.74 x 1.4mm.
- Viewing Area: 31.86 x 39.74mm
- Active Area: 31.46 x 39.34mm
- 65K colour
- Brightness: 60 cd/m²
- Viewing Angle: near 360 degrees, See Note.
- No Back lighting

Note: The Displays used are the highest rated ‘Grade A’ Displays, which allow for 0-4 defective pixels. A defective pixel could be solid Black (Dead), White, Red, Green or Blue.

Note: This PMTOLED display is Transparent and is therefore viewable from both the front and rear. When viewing from the rear, the image will be mirrored compared to the front.

6.2. GOLDELOX Processor

The module is designed around the GOLDELOX Graphics Processor from 4D-Labs.

The GOLDELOX is a custom embedded graphics processor designed to interface with many popular OLED and LCD display panels. Powerful graphics, text, image, animation and countless more features are built right inside the chip. It offers a simple plug-n-play interface to many 8bit 80-Series colour LCD and OLED displays.
The chip is designed to work with minimal design effort and all of the data and control signals are provided by the chip to interface directly to the display. Simply choose your display and interface it to the GOLDELOX on your application board. This offers enormous advantage to the designer in development time and cost saving and takes away all of the burden of low level design.

The data sheet for the chip is available from the http://www.4dsystems.com.au website: “GOLDELOX-DS-revx.pdf”

6.3. SD/SDHC Memory Cards

The module supports micro-SD memory cards via the on-board micro-SD connector. The memory card is used for all multimedia file retrieval such as images, animations and movie clips. The memory card can also be used as general purpose storage for data logging applications. Support is available for off the shelf micro-SD and high capacity HC memory cards (4GB and above).

7. OLED Screen Precautions

- Avoid having a White Background. The more pixels that are lit up, the more the display module will consume current. A full white screen will have the highest power consumption.

- Avoid displaying objects or text on White Backgrounds. This will cause a smearing effect which is inherent to all PMOLED displays. Instead try a shaded mixed colour as the background or better still a black background. Ideally have mixed coloured objects/text/icons on a black background.

- Avoid having to display the same image/object on the screen for lengthy periods of time. This will cause a burn-in which is a common problem with all types of display technologies. Blank the screen after a while or dim it very low by adjusting the contrast. Better still; implement a screen saver feature.

- The display can be easily scratched. The soft polarisation film on the glass surface may be damaged if rubbed by hard objects. Handle with care to avoid scratching the display.

- Moisture and water can damage the display. Moisture on the surface of a powered display will cause the electrodes to corrode. Wipe off any moisture gently or let the display dry before usage.

- Dirt from fingerprint oil and fat can easily stain the surface of the display. Gently wipe off any stains with a soft lint-free cloth.

- The performance of the display will degrade under high temperature and humidity. Avoid such conditions when storing.

- Displays are susceptible to mechanical shock and any force exerted on the module may result in deformed zebra strips and cracks.

- Care should be taken when handling the uTOLED-20-G2 display module as the display flex is delicate and could be damaged with rough handling. Undue stress on the display flex can lead to horizontal/vertical black lines.

- Always use the mounting holes on the module’s printed circuit board to mount the display.
8. Hardware Tools

The following hardware tools are required for full control of the μTOLED-20-G2 module.

8.1. 4D Programming Cable

The 4D Programming Cable is an essential hardware tool to program, customise and test the μTOLED-20-G2 module.

The 4D Programming Cable is used to program a new Firmware/PmmC and downloading compiled 4DGL code into the module. It even serves as an interface for communicating serial data to the PC.

The 4D Programming Cable is available from 4D Systems, www.4dsystems.com.au

9. 4DGL - Software Language

The heart of the μTOLED-20-G2 module is the GOLDELOX graphics processor from 4D Labs. The GOLDELOX belongs to a family of processors powered by a highly optimised soft core virtual engine, EVE (Extensible Virtual Engine).

EVE is a proprietary, high performance virtual-machine with an extensive byte-code instruction set optimised to execute compiled 4DGL programs. 4DGL (4D Graphics Language) was specifically developed from ground up for the EVE engine core. It is a high level language which is easy to learn and simple to understand yet powerful enough to tackle many embedded graphics applications.

4DGL is a graphics oriented language allowing rapid application development, and the syntax structure was designed using elements of popular languages such as C, Basic, Pascal and others.

Programmers familiar with these languages will feel right at home with 4DGL. It includes many familiar instructions such as IF..ELSE..ENDIF, WHILE..WEND, REPEAT..UNTIL, GOSUB..ENDSUB, GOTO, PRINT as well as some specialised instructions SERIN, SEROUT, GFX_LINE, GFX_CIRCLE and many more.

For detailed information pertaining to the 4DGL language, please refer to the following documents:
“GOLDELOX-4DGL-Internal-Functions.pdf”

To assist with the development of 4DGL applications, the Workshop 4 IDE combines a full-featured editor, a compiler, a linker and a down-loader into a single PC-based application. It’s all you need to code, test and run your applications.

4DGL is available to be written in two of the three environments offered by the Workshop 4 IDE, Designer and ViSI.
10. 4D Systems - Workshop 4 IDE

Workshop 4 is a comprehensive software IDE that provides an integrated software development platform for all of the 4D family of processors and modules. The IDE combines the Editor, Compiler, Linker and Downloader to develop complete 4DGL application code. All user application code is developed within the Workshop 4 IDE.

The Workshop 4 IDE supports multiple development environments for the user, to cater for different user requirements and skill level.

- **The Designer environment** enables the user to write 4DGL code in its natural form to program the μTOLED-20-G2.
- A visual programming experience, suitably called ViSi, enables drag-and-drop type placement of objects to assist with 4DGL code generation and allows the user to visualise how the display will look while being developed.
- A **Serial** environment is also provided to transform the μTOLED-20-G2 into a slave serial module, allowing the user to control the display from any host microcontroller or device with a serial port.


10.1. Workshop 4 – Designer Environment

Choose the Designer environment to write 4DGL code in its raw form.

The Designer environment provides the user with a simple yet effective programming environment where pure 4DGL code can be written, compiled and downloaded to the μTOLED-20-G2.

10.2. Workshop 4 – ViSi Environment

ViSi was designed to make the creation of graphical displays a more visual experience.

ViSi is a great software tool that allows the user to see the instant results of their desired graphical layout. Additionally, there is a selection of inbuilt dials, gauges and meters that can simply be placed onto the simulated module display. From here each object can have its properties edited, and at the click of a button all relevant 4DGL code associated with that object is produced in the user program. The user can then write 4DGL code around these objects to utilise them in the way they choose.

The Workshop 4 IDE is available from the 4D Systems website. [www.4dsystems.com.au](http://www.4dsystems.com.au)
10.3. Workshop 4 – Serial Environment

The Serial environment in the Workshop 4 IDE provides the user the ability to transform the μTOLED-20-G2 into a slave serial graphics controller.

This enables the user to use their favourite microcontroller or serial device as the Host, without having to learn 4DGL or program in a separate IDE. Once the module is configured and downloaded to from the Serial Environment, simple graphic commands can be sent from the users host microcontroller to display primitives, images, sound or even video.

Refer to the Serial Environment section in the Workshop 4 user manual, for a complete listing of all the supported serial commands “Workshop-4-IDE-User-Manual.pdf”

By default, each module shipped from the 4D Systems factory will come pre-programmed ready for use in the Serial mode.
11. Mechanical Details
12. Specifications and Ratings

**ABSOLUTE MAXIMUM RATINGS**

- Operating ambient temperature: -20°C to +70°C
- Storage temperature: -30°C to +80°C
- Voltage on any digital input pin with respect to GND: -0.3V to 6.0V
- Voltage on SWITCH pin with respect to GND: -0.3V to 6.0V
- Voltage on VCC with respect to GND: -0.3V to 6.0V
- Maximum current out of GND pin: 300mA
- Maximum current into VCC pin: 250mA
- Maximum output current sunk/sourced by any pin: 4mA
- Total power dissipation: 1.0W

**NOTE:** Stresses above those listed here may cause permanent damage to the device. This is a stress rating only and functional operation of the device at those or any other conditions above those indicated in the recommended operation listings of this specification is not implied. Exposure to maximum rating conditions for extended periods may affect device reliability.

**RECOMMENDED OPERATING CONDITIONS**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Conditions</th>
<th>Min</th>
<th>Typ</th>
<th>Max</th>
<th>Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>Supply Voltage (VCC)</td>
<td></td>
<td>4.0</td>
<td>5.0</td>
<td>5.5</td>
<td>V</td>
</tr>
<tr>
<td>Operating Temperature</td>
<td></td>
<td>-30</td>
<td>--</td>
<td>+70</td>
<td>°C</td>
</tr>
<tr>
<td>Input Low Voltage (VIL) RX Pin</td>
<td>GND</td>
<td>--</td>
<td>0.8</td>
<td>V</td>
<td></td>
</tr>
<tr>
<td>Input High Voltage (VIH) RX Pin</td>
<td>2.0, 3.3, 5.0 V</td>
<td>--</td>
<td>--</td>
<td>μs</td>
<td></td>
</tr>
<tr>
<td>Reset Pulse</td>
<td>External Open Collector</td>
<td>2.0</td>
<td>--</td>
<td>--</td>
<td></td>
</tr>
<tr>
<td>Operational Delay</td>
<td>Power-Up or External Reset</td>
<td>1000</td>
<td>--</td>
<td>ms</td>
<td></td>
</tr>
</tbody>
</table>

**GLOBAL CHARACTERISTICS BASED ON OPERATING CONDITIONS**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Conditions</th>
<th>Min</th>
<th>Typ</th>
<th>Max</th>
<th>Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>Supply Current (ICC) VCC = 5.0V</td>
<td></td>
<td>14</td>
<td>40</td>
<td>120</td>
<td>mA</td>
</tr>
<tr>
<td>Low Power Current (ICC) VCC = 5.0V, Contrast = 0</td>
<td></td>
<td>300</td>
<td>500</td>
<td>--</td>
<td>μA</td>
</tr>
<tr>
<td>Output Low Voltage (VOL) TX, IO1, IO2, IOL = 3.4mA</td>
<td></td>
<td>--</td>
<td>--</td>
<td>0.4</td>
<td>V</td>
</tr>
<tr>
<td>Output High Voltage (VOH) TX, IO1, IO2, IOL = -2.0mA</td>
<td></td>
<td>2.4</td>
<td>--</td>
<td>3.3</td>
<td>V</td>
</tr>
<tr>
<td>A/D Converter Resolution IO1 Pin</td>
<td></td>
<td>--</td>
<td>8</td>
<td>10</td>
<td>bits</td>
</tr>
<tr>
<td>Capacitive Loading All pins</td>
<td></td>
<td>--</td>
<td>--</td>
<td>50</td>
<td>pF</td>
</tr>
<tr>
<td>Flash Memory Endurance PmmC/4DGL Programming</td>
<td></td>
<td>--</td>
<td>1000</td>
<td>--</td>
<td>E/W</td>
</tr>
</tbody>
</table>
## OPTICAL CHARACTERISTICS

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Condition</th>
<th>Min</th>
<th>Typ</th>
<th>Max</th>
<th>Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>Luminance (L)</td>
<td>VCC = 5.0V</td>
<td>--</td>
<td>60</td>
<td>--</td>
<td>cd/m²</td>
</tr>
<tr>
<td>Viewing Angle (VA)</td>
<td>VCC = 5.0V</td>
<td>--</td>
<td>--</td>
<td>360</td>
<td>degree</td>
</tr>
<tr>
<td>Operational Lifetime (LT)</td>
<td>100% white pattern. 60 cd/m². End of lifetime is 50% initial intensity.</td>
<td>5000</td>
<td>--</td>
<td>--</td>
<td>hours</td>
</tr>
</tbody>
</table>

**Transmittance**

<table>
<thead>
<tr>
<th>Colour Chromaticity (CIE 1931)</th>
<th>Back Light On, 100% White Pattern</th>
</tr>
</thead>
<tbody>
<tr>
<td>White, Wx</td>
<td>0.270 0.320 0.370</td>
</tr>
<tr>
<td>White, Wy</td>
<td>0.351 0.401 0.451</td>
</tr>
<tr>
<td>Red, Rx</td>
<td>0.608 0.658 0.708</td>
</tr>
<tr>
<td>Red, Ry</td>
<td>0.288 0.338 0.388</td>
</tr>
<tr>
<td>Green, Gx</td>
<td>0.229 0.279 0.329</td>
</tr>
<tr>
<td>Green, Gx</td>
<td>0.594 0.644 0.694</td>
</tr>
<tr>
<td>Blue, Bx</td>
<td>0.119 0.169 0.219</td>
</tr>
<tr>
<td>Blue, By</td>
<td>0.184 0.234 0.284</td>
</tr>
</tbody>
</table>

## ORDERING INFORMATION

**Order Code:** µTOLED-20-G2  
**Package:** 105mm x 65mm x 30mm  
**Packaging:** Module sealed in antistatic foam padded 4D Systems Box
13. Legal Notice

Proprietary Information
The information contained in this document is the property of 4D Systems Pty. Ltd. and may be the subject of patents pending or granted, and must not be copied or disclosed without prior written permission. 4D Systems endeavours to ensure that the information in this document is correct and fairly stated but does not accept liability for any error or omission. The development of 4D Systems products and services is continuous and published information may not be up to date. It is important to check the current position with 4D Systems. 4D Systems reserves the right to modify, update or make changes to Specifications or written material without prior notice at any time.

All trademarks belong to their respective owners and are recognised and acknowledged.

Disclaimer of Warranties & Limitation of Liability
4D Systems makes no warranty, either expressed or implied with respect to any product, and specifically disclaims all other warranties, including, without limitation, warranties for merchantability, non-infringement and fitness for any particular purpose. Information contained in this publication regarding device applications and the like is provided only for your convenience and may be superseded by updates. It is your responsibility to ensure that your application meets with your specifications.

Images and graphics used throughout this document are for illustrative purposes only. All images and graphics used are possible to be displayed on the 4D Systems range of products, however the quality may vary.

In no event shall 4D Systems be liable to the buyer or to any third party for any indirect, incidental, special, consequential, punitive or exemplary damages (including without limitation lost profits, lost savings, or loss of business opportunity) arising out of or relating to any product or service provided or to be provided by 4D Systems, or the use or inability to use the same, even if 4D Systems has been advised of the possibility of such damages.

4D Systems products are not fault tolerant nor designed, manufactured or intended for use or resale as on line control equipment in hazardous environments requiring fail – safe performance, such as in the operation of nuclear facilities, aircraft navigation or communication systems, air traffic control, direct life support machines or weapons systems in which the failure of the product could lead directly to death, personal injury or severe physical or environmental damage ('High Risk Activities'). 4D Systems and its suppliers specifically disclaim any expressed or implied warranty of fitness for High Risk Activities.

Use of 4D Systems’ products and devices in 'High Risk Activities' and in any other application is entirely at the buyer’s risk, and the buyer agrees to defend, indemnify and hold harmless 4D Systems from any and all damages, claims, suits, or expenses resulting from such use. No licenses are conveyed, implicitly or otherwise, under any 4D Systems intellectual property rights.

14. Contact Information

For Technical Support: support@4dsystems.com.au
For Sales Support: sales@4dsystems.com.au
Website: www.4dsystems.com.au

Copyright 4D Systems Pty. Ltd. 2000-2012.
1.5” microOLED GOLDELOX Display
μOLED-128-G2

Document Date: 5th September 2012
Document Revision: 1.0
Overview

The µOLED-128-G2 display module flaunts a 1.5” state of the art Passive Matrix OLED (PMOLED) technology that is both crisp and superior to other display technologies, and powered with an embedded GOLDELOX graphics processor that delivers ‘stand-alone’ functionality to any project.

The module is an elegant combination of a 1.5” PMOLED Passive Matrix Screen, along with a modest but comprehensive collection of I/O Features. These include a micro-SD card connector, two general purpose input/output pins (GPIO’s) with Dallas 1-Wire Support, Analog Input and sound generation capability, along with serial communications.

Embedded at the heart of the design is the GOLDELOX processor, which is driven by a highly optimised virtual core engine called EVE (Extensible Virtual Engine).

This display module serves as a perfect solution to be deployed at the forefront of any product design, requiring a brilliance of colour, animation or images on any application. This GOLDELOX driven Intelligent Display Module is a perfect example of where art meets technology.

The µOLED-128-G2 has a 1.5” PMOLED Display at the forefront of the design that showcases the power and capabilities of the GOLDELOX processor. Combining a resolution of 128x128 pixels with 65K True to Life colours, this display module is perfect for animations, slideshows and other multimedia presentations.

Basic audio generation is possible on the module through the use of RTTTL sounds and beeps. These can be executed easily with a simple command, and are non-blocking which enables user code to be executed while audio is playing.

The micro-SD card slot on the module provides the user with expandable memory space suitable for multimedia file retrieval, including images, animations and movie clips, as well as data logging applications.

This microOLED module can be programmed in its native 4DGL language (similar to C), using the Workshop 4 IDE Software tool suite. It can also be configured very easily as a serial slave device, for use with your favourite host controller. Freedom is at your fingertips with the intelligent µOLED-128-G2 display module.
Contents

1. Description ...................................................................................................................................4

2. Features .......................................................................................................................................4

3. Pin Configuration and Summary ..................................................................................................5

4. Hardware Interface - Pins.............................................................................................................6
   4.1. Serial Ports - COM0 UART ...................................................................................................... 6
   4.2. General Purpose I/O ................................................................................................................. 6
   4.3. System Pins ............................................................................................................................... 7

5. PmmC/Firmware Programming ...................................................................................................8

6. Module Features ..........................................................................................................................8
   6.1. Display – 1.5” PMOLED ........................................................................................................... 8
   6.2. GOLDELOX Processor .............................................................................................................. 8
   6.3. SD/SDHC Memory Cards ......................................................................................................... 9

7. OLED Screen Precautions .............................................................................................................9

8. Hardware Tools ..........................................................................................................................10
   8.1. 4D Programming Cable ............................................................................................................. 10
   8.2. Development Hardware ........................................................................................................... 10

9. 4DGL - Software Language .........................................................................................................10

10. 4D Systems - Workshop 4 IDE ..................................................................................................11
    10.1. Workshop 4 – Designer Environment .................................................................................. 11
    10.2. Workshop 4 – ViSi Environment ......................................................................................... 11
    10.3. Workshop 4 – Serial Environment ..................................................................................... 12

11. Mechanical Details ....................................................................................................................13

12. Schematic Diagram ....................................................................................................................14

13. Specifications and Ratings ........................................................................................................15

14. Legal Notice ..............................................................................................................................17

15. Contact Information ..................................................................................................................17
1. Description

The µOLED-128-G2 is an impressive OLED display module in the 4D Systems microOLED graphics display range. Featuring a 1.5” 128x128 resolution display, it is the ideal size for attractive embedded display applications.

Driving the module and its peripherals is the GOLDELOX processor, a very capable chip which provides impressive graphics power, programmed with 4D Systems Workshop IDE Software. 4D Systems Workshop enables graphic solutions to be constructed rapidly and with ease due to its design being solely for 4D’s graphics controllers.

The µOLED-128-G2 has a modest but comprehensive range of features suited for an application requiring a bright eye catching display, an analog input, Dallas 1-wire sensor capability, audio generation, or simply digital I/O. This is truly an impressive little module.

2. Features

- Low-cost 1.5” OLED display graphics user interface solution.
- 128 x 128 resolution, RGB 65K true to life colours, PMOLED Screen.
- No back lighting with near 180° viewing angle.
- Easy 10 pin interface to any host device: 3.3Vout, IO2, GND, IO1, RESET, GND, RX, TX, +5V, 5V OUT
- Powered by the 4D-Labs GOLDELOX processor (also available as separate OEM IC)
- 10KB of flash memory for user code storage and 510 bytes of RAM for user variables (255 x 16bit vars).
- Asynchronous hardware serial UART port with auto-baud, TTL interface, with 300 to 600K baud.
- 2 x General Purpose I/O pins.
  - IO1 supports:
    - Digital I/O
    - A/D Converter with 8/10 bit resolution
    - Complex sound generation
    - Dedicated RTTTL tune engine
    - Multi-Switch Joystick, Buttons
    - Dallas 1-Wire
  - IO2 supports:
    - Digital I/O
    - Complex sound generation
    - Dedicated RTTTL tune engine
    - Dallas 1-Wire
- On-board micro-SD memory card adaptor for multimedia storage and data logging purposes. HC memory card support is also available for cards larger than 4GB.
- Built in extensive 4DGL graphics and system library functions.
- Display full colour images, animations, icons and video clips (no audio support for video).
- Supports all available Windows fonts and characters (imported as external fonts).
- 4.0V to 5.5V range operation (single supply), nominal 5.0V.
- Module dimensions: 45.5 x 36.0 x 13.7mm (including mounting tabs and header pins).
- Weight ~ 11g.
- Display Viewing Area: 27.0 x 27.0mm
- RoHS Compliant.
## 3. Pin Configuration and Summary

### Interface/Programming Header

<table>
<thead>
<tr>
<th>Pin</th>
<th>Symbol</th>
<th>I/O</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>+5V</td>
<td>P</td>
<td>Main Voltage Supply +ve input pin. Reverse polarity protected. Range is 4.0V to 5.5V, nominal 5.0V.</td>
</tr>
<tr>
<td>2</td>
<td>5V OUT</td>
<td>P</td>
<td>5V OUT provides approximately 4.7 V through a protection diode.</td>
</tr>
<tr>
<td>3</td>
<td>TX</td>
<td>O</td>
<td>Asynchronous Serial Transmit pin. Output data is at TTL voltage levels. Connect this pin to external device Serial Receive (Rx) signal. This pin is tolerant up to 5.0V levels.</td>
</tr>
<tr>
<td>4</td>
<td>IO2</td>
<td>I/O</td>
<td>General purpose IO2 pin. See section 2.2 for more detail.</td>
</tr>
<tr>
<td>5</td>
<td>RX</td>
<td>I</td>
<td>Asynchronous Serial Receive pin. Connect this pin to external device Serial Transmit (Tx) signal. This pin is tolerant up to 5.0V levels.</td>
</tr>
<tr>
<td>6</td>
<td>IO1</td>
<td>I/O</td>
<td>General purpose IO1 pin. See section 2.2 for more detail.</td>
</tr>
<tr>
<td>7</td>
<td>GND</td>
<td>P</td>
<td>Supply Ground.</td>
</tr>
<tr>
<td>8</td>
<td>GND</td>
<td>P</td>
<td>Supply Ground.</td>
</tr>
<tr>
<td>9</td>
<td>RESET</td>
<td>I</td>
<td>Master Reset signal. Internally pulled up to 3.3V via a 4.7K resistor. An active Low pulse greater than 2.0μs will reset the module. If the module needs to be reset externally, only use open collector type circuits. This pin is not driven low by any internal conditions. The host should control this pin via one of its port pins using an open collector/drain arrangement.</td>
</tr>
<tr>
<td>10</td>
<td>3.3V</td>
<td>P</td>
<td>Regulated 3.3 Volts output, maximum available current 50mA to power external circuitry.</td>
</tr>
</tbody>
</table>

I = Input, O = Output, P = Power
4. Hardware Interface - Pins

The μOLED-128-G2 provides both a hardware and software interface. This section describes in detail the hardware interface pins of the device.

4.1. Serial Ports - COM0 UART

The μOLED-128-G2 has a dedicated hardware UART that can communicate with external serial devices.

The primary features are:
• Full-Duplex 8 bit data transmission and reception through the TX and RX pins.
• Data format: 8 bits, No Parity, 1 Stop bit.
• Auto Baud feature.
• Baud rates from 300 baud up to 600K baud.

The Serial port is also the primary interface for downloading compiled 4DGL application code as well as future PmmC/Firmware updates for the on-board GOLDELOX processor. Refer to Section 5. PmmC/Firmware Programming for more details.

TX pin 3 (Serial Transmit):
Asynchronous Serial port Transmit pin, TX. The serial output data is at TTL voltage levels. Connect this pin to external serial device Rx signal.

RX pin 5 (Serial Receive):
Asynchronous Serial port Receive pin, RX. Connect this pin to external serial device Transmit Tx signal.

4.2. General Purpose I/O

There are 2 GPIO pins available, IO1 and IO2. Each GPIO has a multitude of high level functions associated with it and these can be selected within 4DGL user application code.

Refer to the separate document titled "GOLDELOX-4DGL-Internal-Functions.pdf" for a complete set of built in 4DGL library functions.

IO1 pin 6 (General Purpose IO1):
General purpose IO1 pin. The following table lists the available GPIO functions and features.

IO2 pin 4 (General Purpose IO2):
General purpose IO2 pin. The following table lists the available GPIO functions and features.

Note: GPIO pins are 5.0V tolerant.

<table>
<thead>
<tr>
<th>Function</th>
<th>IO1</th>
<th>IO2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Digital Input</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Digital Output</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>A/D Converter 8/10 bits</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Joystick – 5 position multi-switch</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Dallas 1-Wire support</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Sound Generation, RTTTL Tunes</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

Input/Output:
Both IO1 and IO2 pins can be programmed to be Inputs or Outputs. Diagram below shows a LED connected to IO1 (programmed as an output) and a button connected to IO2 (programmed as an input).

Analogue to Digital Converter:
The IO1 pin can be programmed as an A/D input. Option is available to select 8 bit or 10 bit resolution. Diagram below is a circuit of a Light Dependant Resistor (LDR) connected to IO1 to measure and record changes in ambient light.

Joystick - Multi Switch:
Multiple buttons or a multi-switch Joystick can be connected to the IO1 pin on the μOLED-128-G2 module. Up to five buttons or a 5 position multi-switch joystick connects to a junction of a resistor ladder network that forms a voltage divider. The A/D converter of the IO1 pin internally reads the analogue value and decodes it accordingly. This feature is supported by dedicated 4DGL library functions. The following diagrams indicate how to connect up to five individual buttons or a multi switch joystick to the IO1 pin.
Unused buttons do not need resistors to be connected to the circuit. Table below lists the buttons and corresponding resistor values.

<table>
<thead>
<tr>
<th>Number of Buttons</th>
<th>Button Number</th>
<th>Resistor Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>SW1</td>
<td>22K</td>
</tr>
<tr>
<td>2</td>
<td>SW2</td>
<td>10K</td>
</tr>
<tr>
<td>3</td>
<td>SW3</td>
<td>4.7K</td>
</tr>
<tr>
<td>4</td>
<td>SW4</td>
<td>2.2K</td>
</tr>
<tr>
<td>5</td>
<td>SW5</td>
<td>1.2K</td>
</tr>
</tbody>
</table>

Dallas 1-Wire:
The Dallas 1-Wire protocol is a form of serial communications designed to operate over a single data line plus ground reference. Multiple 1 Wire devices can be attached to the same shared data line to network many devices. One wire device support is available on both the IO1 and the IO2 pins on the μOLED-128-G2 module.

The following diagram depicts a typical 1-Wire temperature sensor interface.

Sound Output:
The μOLED-128-G2 module is capable of generating complex sounds and RTTTL tunes from its IO1 and IO2 pins. A simple speaker circuit as shown below can be utilised.

4.3. System Pins

+5V (Module Voltage Input)
Pin 1:
Module supply voltage input pin. This pin must be connected to a regulated supply voltage in the range of 4.0 Volts to 5.5 Volts DC. Nominal operating voltage is 5.0 Volts.

5V Out (~4.7V)
Pin 2:
External circuitry that requires approximately 5V supply can be powered up via this pin. Maximum available current is 50mA.

GND (Module Ground)
Pins 7, 8:
Device ground pins. These pins must be connected to ground.

RESET (Module Master Reset)
Pin 9:
Module Master Reset pin. An active low pulse of greater than 2 micro-seconds will reset the module. Internally pulled up to 3.3V via a 4.7K resistor. Only use open collector type circuits to reset the device if an external reset is required.

3.3Vout (3.3V Regulated Output)
Pin 10:
External circuitry that requires a regulated 3.3V supply can be powered up via this pin. Maximum available current is 50mA.
5. PmmC/Firmware Programming

The GOLDELOX processor on the μOLED-128-G2 module can be re-programmed with the latest PmmC configuration for updates and future proofing. The chip-level configuration is available as a PmmC (Personality-module-micro-Code) file and the programming must be performed over the serial interface. The chip-resident internal 4DGL functions are part of the GOLDELOX PmmC configuration file so please check regularly for the latest updates and enhancements.

It is recommended that the μOLED-128-G2 display module be socketed on the application board so that it can be easily removed for PmmC programming.

The PmmC file is programmed into the device with the aid of Workshop 4, the 4D Systems IDE software (See Section 10). To provide a link between the PC and the ICSP interface, a specific 4D Programming Cable is required and is available from 4D Systems.

Using a non-4D programming interface could damage your display, and void your Warranty.

6. Module Features

The μOLED-128-G2 module is designed to accommodate most applications. Some of the main features of the module are listed below.

6.1. Display – 1.5” PMOLED

The μOLED-128-G2 is equipped with a full colour PMOLED screen. Some of the features of the screen are:

- Screen Size: 1.5”diagonal
- 128 x 128 Resolution PMOLED Display
- Screen Dimensions: 33.5 x 33.5mm.
- Viewing Area: 27 x 27mm
- 65K true to life colours
- Brightness: 100 cd/m2
- Contrast Ratio: 5000:1
- Viewing Angle: > 160 degrees
- No Back lighting

Note: The Displays used are the highest rated ‘Grade A’ Displays, which allow for 0-4 defective pixels. A defective pixel could be solid Black (Dead), White, Red, Green or Blue.

6.2. GOLDELOX Processor

The module is designed around the GOLDELOX Graphics Processor from 4D-Labs.

The GOLDELOX is a custom embedded graphics processor designed to interface with many popular OLED and LCD display panels. Powerful graphics, text, image, animation and countless more features are built right inside the chip. It offers a simple plug-n-play interface to many 8bit 80-Series colour LCD and OLED displays.

The chip is designed to work with minimal design effort and all of the data and control signals are provided by the chip to interface directly to the display. Simply choose your display and interface it.
to the GOLDELOX on your application board. This offers enormous advantage to the designer in development time and cost saving and takes away all of the burden of low level design.

The data sheet for the chip is available from the http://www.4dsystems.com.au website: "GOLDELOX-DS-revx.pdf"

6.3. SD/SDHC Memory Cards

The module supports micro-SD memory cards via the on-board micro-SD connector. The memory card is used for all multimedia file retrieval such as images, animations and movie clips. The memory card can also be used as general purpose storage for data logging applications. Support is available for off the shelf micro-SD and high capacity HC memory cards (4GB and above).

7. OLED Screen Precautions

- Avoid having a White Background. The more pixels that are lit up, the more the display module will consume current. A full white screen will have the highest power consumption.

- Avoid displaying objects or text on White Backgrounds. This will cause a smearing effect which is inherent to all PMOLED displays. Instead try a shaded mixed colour as the background or better still a black background. Ideally have mixed coloured objects/text/icons on a black background.

- Avoid having to display the same image/object on the screen for lengthy periods of time. This will cause a burn-in which is a common problem with all types of display technologies. Blank the screen after a while or dim it very low by adjusting the contrast. Better still; implement a screen saver feature.

- The display can be easily scratched. The soft polarisation film on the glass surface may be damaged if rubbed by hard objects. Handle with care to avoid scratching the display.

- Moisture and water can damage the display. Moisture on the surface of a powered display will cause the electrodes to corrode. Wipe off any moisture gently or let the display dry before usage.

- Dirt from fingerprint oil and fat can easily stain the surface of the display. Gently wipe off any stains with a soft lint-free cloth.

- The performance of the display will degrade under high temperature and humidity. Avoid such conditions when storing.

- Displays are susceptible to mechanical shock and any force exerted on the module may result in deformed zebra strips and cracks.

- Always use the mounting holes on the module’s printed circuit board to mount the display.
8. Hardware Tools

The following hardware tools are required for full control of the μOLED-128-G2 module.

8.1. 4D Programming Cable

The 4D Programming Cable is an essential hardware tool to program, customise and test the μOLED-128-G2 module.

The 4D Programming Cable is used to program a new Firmware/PmmC and downloading compiled 4DGL code into the module. It even serves as an interface for communicating serial data to the PC.

The 4D Programming Cable is available from 4D Systems, www.4dsystems.com.au

8.2. Development Hardware

4D Systems has designed a development board compatible with the μOLED-128-G2, enabling easy prototyping to take place and to experiment with the μOLED-128-G2 display module.

This development board is called the 4DevBoard, and is available from the 4D Systems website store, or from 4D Systems distributors.

9. 4DGL - Software Language

The heart of the μOLED-128-G2 module is the GOLDELOX graphics processor from 4D Labs. The GOLDELOX belongs to a family of processors powered by a highly optimised soft core virtual engine, EVE (Extensible Virtual Engine).

EVE is a proprietary, high performance virtual-machine with an extensive byte-code instruction set optimised to execute compiled 4DGL programs. 4DGL (4D Graphics Language) was specifically developed from ground up for the EVE engine core. It is a high level language which is easy to learn and simple to understand yet powerful enough to tackle many embedded graphics applications.

4DGL is a graphics oriented language allowing rapid application development, and the syntax structure was designed using elements of popular languages such as C, Basic, Pascal and others.

Programmers familiar with these languages will feel right at home with 4DGL. It includes many familiar instructions such as IF..ELSE..ENDIF, WHILE..WEND, REPEAT..UNTIL, GOSUB..ENDSUB, GOTO, PRINT as well as some specialised instructions SERIN, SEROUT, GFX_LINE, GFX_CIRCLE and many more.

For detailed information pertaining to the 4DGL language, please refer to the following documents:
“GOLDELOX-4DGL-Internal-Functions.pdf”

To assist with the development of 4DGL applications, the Workshop 4 IDE combines a full-featured editor, a compiler, a linker and a downloader into a single PC-based application. It’s all you need to code, test and run your applications.

4DGL is available to be written in two of the three environments offered by the Workshop 4 IDE, Designer and ViSi.
10. 4D Systems - Workshop 4 IDE

Workshop 4 is a comprehensive software IDE that provides an integrated software development platform for all of the 4D family of processors and modules. The IDE combines the Editor, Compiler, Linker and Downloader to develop complete 4DGL application code. All user application code is developed within the Workshop 4 IDE.

The Workshop 4 IDE supports multiple development environments for the user, to cater for different user requirements and skill level.

- **The Designer** environment enables the user to write 4DGL code in its natural form to program the μOLED-128-G2.
- A visual programming experience, suitably called **ViSi**, enables drag-and-drop type placement of objects to assist with 4DGL code generation and allows the user to visualise how the display will look while being developed.
- A **Serial** environment is also provided to transform the μOLED-128-G2 into a slave serial module, allowing the user to control the display from any host microcontroller or device with a serial port.

The Workshop 4 IDE is available from the 4D Systems website. [www.4dsystems.com.au](http://www.4dsystems.com.au)


10.1. Workshop 4 – Designer Environment

Choose the Designer environment to write 4DGL code in its raw form.

The Designer environment provides the user with a simple yet effective programming environment where pure 4DGL code can be written, compiled and downloaded to the μOLED-128-G2.

10.2. Workshop 4 – ViSi Environment

ViSi was designed to make the creation of graphical displays a more visual experience.

ViSi is a great software tool that allows the user to see the instant results of their desired graphical layout. Additionally, there is a selection of inbuilt dials, gauges and meters that can simply be placed onto the simulated module display. From here each object can have its properties edited, and at the click of a button all relevant 4DGL code associated with that object is produced in the user program. The user can then write 4DGL code around these objects to utilise them in the way they choose.
10.3. Workshop 4 – Serial Environment

The Serial environment in the Workshop 4 IDE provides the user the ability to transform the μOLED-128-G2 into a slave serial graphics controller.

This enables the user to use their favourite microcontroller or serial device as the Host, without having to learn 4DGL or program in a separate IDE. Once the module is configured and downloaded to from the Serial Environment, simple graphic commands can be sent from the users host microcontroller to display primitives, images, sound or even video.

Refer to the Serial Environment section in the Workshop 4 user manual, for a complete listing of all the supported serial commands “Workshop-4-IDE-User-Manual.pdf”

By default, each module shipped from the 4D Systems factory will come pre-programmed ready for use in the Serial mode.
11. Mechanical Details

1.5" microOLED Display - uOLED-128-G2
12. Schematic Diagram
13. Specifications and Ratings

### ABSOLUTE MAXIMUM RATINGS

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Conditions</th>
<th>Min</th>
<th>Typ</th>
<th>Max</th>
<th>Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>Operating ambient temperature</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Storage temperature</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Voltage on any digital input pin with respect to GND</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Voltage on SWITCH pin with respect to GND</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Voltage on VCC with respect to GND</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Maximum current out of GND pin</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Maximum current into VCC pin</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Maximum output current sunk/sourced by any pin</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Total power dissipation</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**NOTE:** Stresses above those listed here may cause permanent damage to the device. This is a stress rating only and functional operation of the device at those or any other conditions above those indicated in the recommended operation listings of this specification is not implied. Exposure to maximum rating conditions for extended periods may affect device reliability.

### RECOMMENDED OPERATING CONDITIONS

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Conditions</th>
<th>Min</th>
<th>Typ</th>
<th>Max</th>
<th>Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>Supply Voltage (VCC)</td>
<td></td>
<td>4.0</td>
<td>5.0</td>
<td>5.5</td>
<td>V</td>
</tr>
<tr>
<td>Operating Temperature</td>
<td></td>
<td>-30</td>
<td>--</td>
<td>+70</td>
<td>°C</td>
</tr>
<tr>
<td>Input Low Voltage (VIL)</td>
<td>RX Pin</td>
<td></td>
<td></td>
<td></td>
<td>V</td>
</tr>
<tr>
<td>Input High Voltage (VIH)</td>
<td>RX Pin</td>
<td>2.0</td>
<td>3.3</td>
<td>5.0</td>
<td>V</td>
</tr>
<tr>
<td>Reset Pulse</td>
<td>External Open Collector</td>
<td>2.0</td>
<td>--</td>
<td>--</td>
<td>µs</td>
</tr>
<tr>
<td>Operational Delay</td>
<td>Power-Up or External Reset</td>
<td>1000</td>
<td>--</td>
<td>--</td>
<td>ms</td>
</tr>
</tbody>
</table>

### GLOBAL CHARACTERISTICS BASED ON OPERATING CONDITIONS

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Conditions</th>
<th>Min</th>
<th>Typ</th>
<th>Max</th>
<th>Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>Supply Current (ICC)</td>
<td>VCC = 5.0V</td>
<td>14</td>
<td>40</td>
<td>120</td>
<td>mA</td>
</tr>
<tr>
<td>Low Power Current (ICC)</td>
<td>VCC = 5.0V, Contrast = 0</td>
<td>300</td>
<td>500</td>
<td>--</td>
<td>µA</td>
</tr>
<tr>
<td>Output Low Voltage (VOL)</td>
<td>TX, IO1, IO2, IOL = 3.4mA</td>
<td></td>
<td></td>
<td>0.4</td>
<td>V</td>
</tr>
<tr>
<td>Output High Voltage (VOH)</td>
<td>TX, IO1, IO2, IOL = -2.0mA</td>
<td>2.4</td>
<td>--</td>
<td>3.3</td>
<td>V</td>
</tr>
<tr>
<td>A/D Converter Resolution</td>
<td>IO1 Pin</td>
<td></td>
<td>8</td>
<td>10</td>
<td>bits</td>
</tr>
<tr>
<td>Capacitive Loading</td>
<td>All pins</td>
<td></td>
<td>--</td>
<td>50</td>
<td>pF</td>
</tr>
<tr>
<td>Flash Memory Endurance</td>
<td>PmmC/4DGL Programming</td>
<td></td>
<td>1000</td>
<td>--</td>
<td>E/W</td>
</tr>
</tbody>
</table>
### CURRENT CONSUMPTION BASED ON DISPLAY USAGE

<table>
<thead>
<tr>
<th>Contrast Setting (Range: 0-15)</th>
<th>Current (mA)</th>
<th>Display Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>High Contrast</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>13.5</td>
<td>All pixels OFF (Black screen)</td>
</tr>
<tr>
<td>15</td>
<td>40.0</td>
<td>Screen has mix text and graphics (Typical usage)</td>
</tr>
<tr>
<td>15</td>
<td>115.0</td>
<td>All pixels ON (White screen)</td>
</tr>
<tr>
<td><strong>Medium Contrast</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>13.5</td>
<td>All pixels OFF (Black screen)</td>
</tr>
<tr>
<td>8</td>
<td>32.0</td>
<td>Screen has mix text and graphics (Typical usage)</td>
</tr>
<tr>
<td>8</td>
<td>110.0</td>
<td>All pixels ON (White screen)</td>
</tr>
<tr>
<td><strong>Low Contrast</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>13.5</td>
<td>All pixels OFF (Black screen)</td>
</tr>
<tr>
<td>0</td>
<td>18.0</td>
<td>Screen has mix text and graphics (Typical usage)</td>
</tr>
<tr>
<td>0</td>
<td>41.0</td>
<td>All pixels ON (White screen)</td>
</tr>
<tr>
<td><strong>Don’t Care</strong></td>
<td>0.3</td>
<td>Screen Power-Down command executed</td>
</tr>
</tbody>
</table>

### OPTICAL CHARACTERISTICS

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Condition</th>
<th>Min</th>
<th>Typ</th>
<th>Max</th>
<th>Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>Luminance (L)</td>
<td>VCC = 5.0V</td>
<td>70</td>
<td>100</td>
<td>--</td>
<td>cd/m²</td>
</tr>
<tr>
<td>Viewing Angle (VA)</td>
<td>VCC = 5.0V</td>
<td>160</td>
<td>--</td>
<td>--</td>
<td>degree</td>
</tr>
<tr>
<td>Contrast Ratio (CR)</td>
<td>VCC = 5.0V</td>
<td>2000:1</td>
<td>5000:1</td>
<td>--</td>
<td>--</td>
</tr>
<tr>
<td>Operational Lifetime (LT)</td>
<td>50% checker board pattern. 90 cd/m². End of lifetime is 50% initial intensity.</td>
<td>10000</td>
<td>15000</td>
<td>--</td>
<td>hours</td>
</tr>
<tr>
<td>Storage Lifetime (ST)</td>
<td>Ta = 25°C, 50% RH</td>
<td>20000</td>
<td>--</td>
<td>--</td>
<td>hours</td>
</tr>
</tbody>
</table>

### ORDERING INFORMATION

**Order Code:** μOLED-128-G2  
**Package:** 65mm x 50mm x 30mm  
**Packaging:** Module sealed in antistatic foam padded 4D Systems Box
14. Legal Notice

Proprietary Information
The information contained in this document is the property of 4D Systems Pty. Ltd. and may be the subject of patents pending or granted, and must not be copied or disclosed without prior written permission. 4D Systems endeavours to ensure that the information in this document is correct and fairly stated but does not accept liability for any error or omission. The development of 4D Systems products and services is continuous and published information may not be up to date. It is important to check the current position with 4D Systems. 4D Systems reserves the right to modify, update or makes changes to Specifications or written material without prior notice at any time.

All trademarks belong to their respective owners and are recognised and acknowledged.

Disclaimer of Warranties & Limitation of Liability
4D Systems makes no warranty, either expressed or implied with respect to any product, and specifically disclaims all other warranties, including, without limitation, warranties for merchantability, non-infringement and fitness for any particular purpose.

Information contained in this publication regarding device applications and the like is provided only for your convenience and may be superseded by updates. It is your responsibility to ensure that your application meets with your specifications.

Images and graphics used throughout this document are for illustrative purposes only. All images and graphics used are possible to be displayed on the 4D Systems range of products, however the quality may vary.

In no event shall 4D Systems be liable to the buyer or to any third party for any indirect, incidental, special, consequential, punitive or exemplary damages (including without limitation lost profits, lost savings, or loss of business opportunity) arising out of or relating to any product or service provided or to be provided by 4D Systems, or the use or inability to use the same, even if 4D Systems has been advised of the possibility of such damages.

4D Systems products are not fault tolerant nor designed, manufactured or intended for use or resale as on line control equipment in hazardous environments requiring fail – safe performance, such as in the operation of nuclear facilities, aircraft navigation or communication systems, air traffic control, direct life support machines or weapons systems in which the failure of the product could lead directly to death, personal injury or severe physical or environmental damage ('High Risk Activities'). 4D Systems and its suppliers specifically disclaim any expressed or implied warranty of fitness for High Risk Activities.

Use of 4D Systems’ products and devices in 'High Risk Activities' and in any other application is entirely at the buyer’s risk, and the buyer agrees to defend, indemnify and hold harmless 4D Systems from any and all damages, claims, suits, or expenses resulting from such use. No licenses are conveyed, implicitly or otherwise, under any 4D Systems intellectual property rights.

15. Contact Information

For Technical Support: support@4dsystems.com.au
For Sales Support: sales@4dsystems.com.au
Website: www.4dsystems.com.au

Copyright 4D Systems Pty. Ltd. 2000-2012.
TECHNICAL DESCRIPTION

Fastrax UP501 GPS Receiver

This document describes the electrical connectivity and main functionality of the Fastrax UP501 hardware.

October 29, 2010

Fastrax Ltd.
TRADEMARKS

Fastrax is a registered trademark of Fastrax Ltd.

All other trademarks are trademarks or registered trademarks of their respective holders.

COPYRIGHT

© 2010 Fastrax Ltd.

DISCLAIMER

This document is compiled and kept up-to-date as conscientiously as possible. Fastrax Ltd. cannot, however, guarantee that the data are free of errors, accurate or complete and, therefore, assumes no liability for loss or damage of any kind incurred directly or indirectly through the use of this document. The information in this document is subject to change without notice and describes only generally the product defined in the introduction of this documentation. Fastrax products are not authorized for use in life-support or safety-critical applications. Use in such applications is done at the sole discretion of the customer. Fastrax will not warrant the use of its devices in such applications.
# CHANGE LOG

<table>
<thead>
<tr>
<th>Rev.</th>
<th>Notes</th>
<th>Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.0</td>
<td>First Release</td>
<td>2010-05-11</td>
</tr>
<tr>
<td>1.1</td>
<td>Added chapter on UP500 vs. UP501, added UP501H, IO voltage specification clarified, tray dimensions added in chapter 7. Table 2 updated. Table 4 updated.</td>
<td>2010-07-01</td>
</tr>
<tr>
<td>1.2</td>
<td>VDD_B mandatory connection clarified in 4.2. UP501D is now standard product, UP501R is custom product, UP501U removed (Table 5).</td>
<td>2010-10-29</td>
</tr>
</tbody>
</table>
CONTENTS

1. GENERAL DESCRIPTION ........................................................................................................ 6
   1.1 Default firmware configuration .................................................................................... 7
   1.2 UP500 vs. UP501 .......................................................................................................... 7

2. SPECIFICATIONS .................................................................................................................. 8
   2.1 General ........................................................................................................................... 8
   2.2 Absolute maximum ratings ............................................................................................ 9

3. OPERATION ............................................................................................................................. 10
   3.1 Operating modes ............................................................................................................ 10
      3.1.1 Tracking/Navigating mode ....................................................................................... 10
      3.1.2 Low Power Tracking/Navigating mode ................................................................. 10
      3.1.3 Backup mode ........................................................................................................ 10

4. CONNECTIVITY ...................................................................................................................... 11
   4.1 Connection assignments ............................................................................................... 11
   4.2 Power supply ................................................................................................................ 11
   4.3 Reset ............................................................................................................................. 12
   4.4 UART ............................................................................................................................ 12
   4.5 PPS ............................................................................................................................... 12
   4.6 Mechanical dimensions and contact numbering ......................................................... 13

5. MODULE OPTIONS ................................................................................................................. 15

6. PCB MOUNTING .................................................................................................................... 16

7. TRAY DIMENSIONS ............................................................................................................... 18
COMPLEMENTARY READING

The following Fastrax reference documents are complementary reading for this document.

<table>
<thead>
<tr>
<th>Ref. #</th>
<th>Document name</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Fastrax UP500 GPS Receiver – Technical Description</td>
</tr>
<tr>
<td>2</td>
<td>NMEA Manual for Fastrax IT500 Series GPS receivers</td>
</tr>
</tbody>
</table>
1. GENERAL DESCRIPTION

The Fastrax UP501 is a GPS receiver module with embedded antenna and tiny form factor 22.0 x 22.0mm x 8mm. The size, mechanics and interface connectivity is an exact match with the Fastrax UP500 [1]. Therefore, customers may replace UP500 with UP501 to gain improved performance with minimal design effort.

The Fastrax UP501 receiver provides very fast TTFF together with market leading weak signal acquisition and tracking sensitivity figures. The Fastrax UP501 module supports enhanced navigation accuracy by utilizing WAAS/EGNOS corrections, which may be enabled via NMEA command [2]. The Fastrax UP501 can also utilize 14-days predicted ephemeris data in AGPS applications.

The Fastrax UP501 module provides complete signal processing from internal antenna to serial data output in NMEA messages. The module requires a power supply VDD and a backup supply VDD_B voltage for non-volatile RTC & RAM blocks. There is a variant of the module available with on-board backup battery, which will eliminate the need for external backup voltage source. PPS signal output is available for accurate timing applications.

The Fastrax UP501 module interfaces to the customer’s application via one serial port, which uses CMOS voltage levels. If RS232 signal levels are required, there is a variant of Fastrax UP501 available with on-board CMOS-to-RS232 level converter.

There is also a Dual-SAW filter version of the UP501 available. This module variant is named as Fastrax UP501D. The Dual-SAW filter is targeted for telematic applications where a radio transmitter is placed close to the GPS receiver. The dual filter design will provide higher attenuation outside of the GPS band and it helps to reduce the risk of EMC issues that are sometimes present when high-gain systems (GPS receiver) that are in strong signal field (radio transmitter).

For complete electrical compatibility with UP500, there is a high-voltage variant UP501H with 5.5V tolerant VDD supply.
1.1 Default firmware configuration

Fastrax UP501 default firmware configuration:

1. Port 0: NMEA 9600 baud
2. NMEA output: GGA, RMC, GSV, GSA (all 1 sec interval)
3. DGPS/SBAS: Disabled (Module supports WAAS/EGNOS)
4. Datum: WGS84

1.2 UP500 vs. UP501

The table below will identify the differences between UP500 and UP501. This information is useful for customers who already have products with UP500 and would like to upgrade to UP501.

<table>
<thead>
<tr>
<th>Item</th>
<th>UP500</th>
<th>UP501</th>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td>VDD Range</td>
<td>+3.0V…+5.5V</td>
<td>+3.0V…+4.2V</td>
<td>UP501H is up to +5.5V</td>
</tr>
<tr>
<td>SBAS default status</td>
<td>Enabled</td>
<td>Disabled</td>
<td>Enable/disable with NMEA command.</td>
</tr>
<tr>
<td>Max. fix rate</td>
<td>5Hz</td>
<td>10Hz</td>
<td>Configurable via NMEA.</td>
</tr>
<tr>
<td>Navigation sensitivity</td>
<td>-158dBm</td>
<td>-165dBm</td>
<td></td>
</tr>
<tr>
<td>Cold start sensitivity</td>
<td>-146dBm</td>
<td>-148dBm</td>
<td></td>
</tr>
<tr>
<td>Power consumption</td>
<td><a href="mailto:90mW@3.0V">90mW@3.0V</a></td>
<td><a href="mailto:75mW@3.0V">75mW@3.0V</a></td>
<td>Typical in navigation.</td>
</tr>
<tr>
<td>AGPS</td>
<td>NA</td>
<td>14-days predicted ephemeris.</td>
<td></td>
</tr>
</tbody>
</table>
2. SPECIFICATIONS

2.1 General

Table 2 General Specifications for UP501.

<table>
<thead>
<tr>
<th>Specification</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>Receiver</td>
<td>GPS L1 C/A-code, SPS</td>
</tr>
<tr>
<td>Channels</td>
<td>66 acquisition and 22 tracking</td>
</tr>
<tr>
<td>Update rate</td>
<td>1 Hz default (fix rate configurable up to 10Hz)</td>
</tr>
<tr>
<td>Acquisition Sensitivity (Cold start)</td>
<td>-148 dBm (1)</td>
</tr>
<tr>
<td>Re-acquisition Sensitivity</td>
<td>-158 dBm (1)</td>
</tr>
<tr>
<td>Navigation Sensitivity</td>
<td>-165 dBm (1)</td>
</tr>
<tr>
<td>Supply voltage, VDD</td>
<td>+3.0 V…+4.2 V (+3.0V…+5.5V for UP501H)</td>
</tr>
<tr>
<td>Back up supply voltage, VDD_B</td>
<td>+2.0 V…+4.2 V (+2.0V…+5.5V for UP501H)</td>
</tr>
<tr>
<td>Power consumption, VDD</td>
<td>75 mW typical @ 3.0 V (2)</td>
</tr>
<tr>
<td></td>
<td>(Typ. <a href="mailto:115mW@3.0V">115mW@3.0V</a> in satellite search phase)</td>
</tr>
<tr>
<td>Power consumption, VDD_B</td>
<td>15 uW typical @ 3.0 V (during battery backup state).</td>
</tr>
<tr>
<td>Operating temperature range</td>
<td>-40 °C…+85 °C (4)</td>
</tr>
<tr>
<td>Serial port protocol</td>
<td>Port 0: NMEA</td>
</tr>
<tr>
<td>Serial data format</td>
<td>8 bits, no parity, 1 stop bit</td>
</tr>
<tr>
<td>Serial data speed (default)</td>
<td>NMEA: 9600 baud</td>
</tr>
<tr>
<td>CMOS I/O signal levels (3)</td>
<td>$V_{IL}$: -0.3V…0.8V, $V_{IH}$: 2.0V…3.6V,</td>
</tr>
<tr>
<td></td>
<td>$V_{OL}$: -0.3V…0.4V, $V_{OH}$: 2.4V…3.2V</td>
</tr>
<tr>
<td>I/O sink/source capability</td>
<td>+/- 2 mA max.</td>
</tr>
<tr>
<td>PPS output</td>
<td>+/- 50 ns (RMS) accuracy</td>
</tr>
</tbody>
</table>

Note (1): measured by conducted measurement from GPS simulator.

Note (2): Navigation with good signals, max. 12 satellites in view.

Note (3): Fastrax UP501R UART signals are RS232 compatible.

Note (4): UP501 backup battery operating range is -10 °C…+60 °C.
# 2.2 Absolute maximum ratings

<table>
<thead>
<tr>
<th>Item</th>
<th>Min</th>
<th>Max</th>
<th>unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Operating temperature</td>
<td>-40</td>
<td>+85</td>
<td>°C</td>
</tr>
<tr>
<td>Storage temperature</td>
<td>-40</td>
<td>+85</td>
<td>°C</td>
</tr>
<tr>
<td>Power dissipation</td>
<td>-</td>
<td>500</td>
<td>mW</td>
</tr>
<tr>
<td>Supply voltage, VDD</td>
<td>-0.3</td>
<td>+4.3 (1)</td>
<td>V</td>
</tr>
<tr>
<td>Supply voltage, VDD_B</td>
<td>-0.3</td>
<td>+4.3 (1)</td>
<td>V</td>
</tr>
<tr>
<td>Input voltage on any input connection</td>
<td>-0.3</td>
<td>+3.6</td>
<td>V</td>
</tr>
<tr>
<td>RF input level</td>
<td>-</td>
<td>+15</td>
<td>dBm</td>
</tr>
</tbody>
</table>

(1) UP501H module variant is 5.5V tolerant on VDD and VDD_B. Contact Fastrax sales for details on availability and Lead Time for UP501H.
3. **OPERATION**

3.1 **Operating modes**

After power up the receiver boots from the internal flash memory for normal operation. Modes of operation:

- Tracking/navigating mode
- Low power tracking/navigating mode
- Backup mode

3.1.1 **Tracking/Navigating mode**

In tracking/navigating mode the Fastrax UP501 receiver module will search for additional satellites and collects almanac data. Once the receiver has collected almanac data (this takes about 12 minutes from Cold Start), it will enter Low Power Tracking mode. The VDD power consumption in table 1 is measured in Low Power Tracking/Navigating mode.

3.1.2 **Low Power Tracking/Navigating mode**

In Low power tracking/navigating mode the receiver continues normal navigation but does not collect further Almanac data. Therefore the current consumption is reduced to level of <75 mW (<85 mW for UP501R with default UART baud rate).

3.1.3 **Backup mode**

When the operating voltage VDD is removed from the Fastrax UP501, the module enters Backup mode. In this mode, the module is keeping time by the RTC oscillator. Also, satellite ephemeris data is stored in battery backup RAM in order to get fast TTFF when VDD is connected again. Any user configuration settings are also valid as long as the backup supply VDD_B is active. When the VDD_B is powered off, the configuration is reset to factory configuration on next power up.
4. CONNECTIVITY

4.1 Connection assignments

The I/O connections are available on the 6-pin, 2.54mm pitch pin-header pads.

Table 4 Connections

<table>
<thead>
<tr>
<th>Contact</th>
<th>Signal name</th>
<th>I/O</th>
<th>Signal description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>RXD</td>
<td>I</td>
<td>UART Port 0 async. input. Internal pull high resistor 75kΩ.</td>
</tr>
<tr>
<td>2</td>
<td>TXD</td>
<td>O</td>
<td>UART Port 0 async. output.</td>
</tr>
<tr>
<td>3</td>
<td>GND</td>
<td>-</td>
<td>Ground</td>
</tr>
<tr>
<td>4</td>
<td>VDD</td>
<td>I</td>
<td>Main power supply</td>
</tr>
<tr>
<td>5</td>
<td>VDD_B</td>
<td>I</td>
<td>Backup supply</td>
</tr>
<tr>
<td>6</td>
<td>PPS</td>
<td>O</td>
<td>Pulse per second output.</td>
</tr>
</tbody>
</table>

Notes:

Figure 1. Pin numbering in the Fastrax UP501 module.

4.2 Power supply

The Fastrax UP501 module (including all variants, except UP501B) requires two separate power supplies: VDD_B for non-volatile back
up block and the VDD for digital parts and I/O. VDD can be switched off when navigation is not needed but if possible keep the backup supply VDD_B active all the time in order to keep the non-volatile RTC & RAM active for fastest possible TTFF. VDD_B has to be connected always when VDD is connected.

Backup supply VDD_B draws typically 5 uA current in back up state. During navigation (while VDD is active) VDD_B current may peak up to 100 uA, while staying at <30 uA average level.

On-board backup battery is available as an assembly option (UP501B). In this case the backup supply (VDD_B) should be left open and the module handles the backup state power supply automatically. The backup battery is charged when VDD supply is connected. A fully charged internal back-up battery is able to keep the UP501B in back-up state for 4hours (typical at room temperature).

Main power supply VDD current varies according to the processor load and satellite acquisition. Typically VDD peak current is up to 40mA during Search mode. In Low Power Tracking mode the average VDD current is typically below 25 mA for the Fastrax UP501 and UP501B modules, and below 28 mA for Fastrax UP501R module with default baud rate of 9600 baud.

### 4.3 Reset

Reset can be initiated by switching off VDD supply for >150 ms.

### 4.4 UART

The device supports UART communication via Port 0 of the GPS IC. With the standard firmware the Port 0 is configured by default to NMEA protocol (9600 baud).

I/O levels at the serial ports are CMOS compatible (see table 1). On-board RS232 level converter is available with the Fastrax UP501R module.

### 4.5 PPS

The pulse-per-second (PPS) output provides an output for timing purposes. There is a 100ms pulse once per second synchronized to UTC second at rising edge when the receiver has a valid 3D position fix available.
4.6 Mechanical dimensions and contact numbering

Module size is 22.0 mm (width) x 22.0 mm (length) x 8 mm (thickness). General tolerance is ±0.3 mm. Detailed mechanical drawing is presented in figure 2. Mechanically the UP501 is fully compatible with the UP500 module.
Figure 2. Mechanics drawing of the Fastrax UP501 module.
5. MODULE OPTIONS

The Fastrax UP501 module is available in three different HW-configurations. This makes it possible for a customer to select optimal solution for different applications depending on required functionality and/or price target.

The following table will summarize the differences in the Fastrax UP501 module options.

<table>
<thead>
<tr>
<th>Variant</th>
<th>Serial Data Interface</th>
<th>On-board Backup Battery</th>
<th>VDD Range</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>UP501</td>
<td>CMOS</td>
<td>NO</td>
<td>3.0V - 4.2V</td>
<td>Standard module, lowest price. CMOS level UART.</td>
</tr>
<tr>
<td>UP501B</td>
<td>CMOS</td>
<td>YES</td>
<td>3.0V - 4.2V</td>
<td>On-board backup battery, simplifies host power management. CMOS level UART.</td>
</tr>
<tr>
<td>UP501D</td>
<td>CMOS</td>
<td>NO</td>
<td>3.0V - 4.2V</td>
<td>Dual SAW filter for enhanced out-of-band interference immunity.</td>
</tr>
<tr>
<td>UP501R</td>
<td>RS232</td>
<td>YES</td>
<td>3.0V - 4.2V</td>
<td>Easy connectivity to RS232 level UART systems. On-board backup battery.</td>
</tr>
<tr>
<td>UP501H</td>
<td>CMOS</td>
<td>NO</td>
<td>3.0V - 5.5V</td>
<td>Extended VDD range. Eliminates need for external GPS voltage regulator in 5V systems.</td>
</tr>
</tbody>
</table>

Fastrax UP501 is available as a standard product with normal lead time. Fastrax UP501B and UP501D are also available as standard products but lead times may be longer than with UP501.

Minimum order quantity (MOQ) is applied on UP501R and UP501H as these modules are not offered as standard products. Please contact Fastrax sales for details on availability.
6. **PCB MOUNTING**

The Fastrax UP501 can be mounted on a customer PCB ("motherboard" in the instructions below) by using standard 2.54mm pitch 1x6 pin header (for example Samtec TLW-106-06-G-S). Two dummy pads are used to solder the module metal shield on the motherboard. Reference pad layout is shown in figure 3.

![Figure 3. Pad layout of mounting side for UP501 module.](image)

There are some rules that needs to be followed in order to maintain good performance for the on-board patch antenna of the UP501:

- Solder the pin header to the module in such way that the pins are as short as possible on the antenna side of the UP501 module (see figure 4).
- Place any active circuitry (processors, memory busses, switching regulators, etc.) on the motherboard as far away as possible from the UP501 module.

- Design a solid VDD source for the UP501 module (VDD supply voltage ripple should be <50 mVp-p).

- If UP501B is used and there is no need for the PPS signal, a 4-pin header (for example Samtec TLW-104-06-G-S) can be used to contact pins #1 through #4. However, the 6-pin header is recommended since it is mechanically more robust. In this case pins #5 and #6 may be left floating on the motherboard.

Figure 4. Side view of the pin header assembly for the UP501 module.
7. **TRAY DIMENSIONS**

UP501 is delivered on trays of 100 pcs. The tray dimensions are presented below.

![Figure 5. UP501 tray dimensions (in mm).](image)
Contact Information

Fastrax Ltd.
Street Address: Valimotie 7, 01510 Vantaa, FINLAND
Tel: +358 (0)424 733 1
Fax: +358 (0)9 8240 9691
http://www.fastraxgps.com

E-mail:
Sales: sales@fastraxgps.com
Support: support@fastraxgps.com
Class 2 Bluetooth® Module

Features

- Fully qualified Bluetooth 2.1/2.0/1.2/1.1 module
- Bluetooth v2.0+EDR support
- Available with on board chip antenna (RN-42) and without antenna (RN-42-N)
- Postage stamp sized form factor, 13.4mm x 25.8 mm x 2mm (RN-42) and 13.4mm x 20 mm x 2 mm (RN-42-N)
- Low power (26µA sleep, 3mA connected, 30mA transmit)
- UART (SPP or HCI) and USB (HCI only) data connection interfaces.
- Sustained SPP data rates - 240Kbps (slave), 300Kbps (master)
- HCI data rates - 1.5Mbps sustained, 3.0Mbps burst in HCI mode
- Embedded Bluetooth stack profiles included (requires no host stack): GAP, SDP, RFCOMM and L2CAP protocols, with SPP and DUN profile support.
- Bluetooth SIG certified
- Castellated SMT pads for easy and reliable PCB mounting
- Certifications: FCC, ICS, CE
- Environmentally friendly, RoHS compliant

Applications

- Cable replacement
- Barcode scanners
- Measurement and monitoring systems
- Industrial sensors and controls
- Medical devices
- Barcode readers
- Computer accessories

Description

The RN42 is a small form factor, low power, highly economic Bluetooth radio for OEM’s adding wireless capability to their products. The RN42 supports multiple interface protocols, is simple to design in and fully certified, making it a complete embedded Bluetooth solution. The RN 42 is functionally compatible with RN 41. With its high performance on chip antenna and support for Bluetooth® Enhanced Data Rate (EDR), the RN42 delivers up to 3 Mbps data rate for distances to 20M. The RN-42 also comes in a package with no antenna (RN-42-N). Useful when the application requires an external antenna, the RN-42-N is shorter in length and has RF pads to route the antenna signal.

Block Diagram
Overview
- Baud rate speeds: 1200bps up to 921Kbps, non-standard baud rates can be programmed.
- Class 2 radio, 60 feet (20meters) distance, 4dBm output transmitter, -80dBm typical receive sensitivity
- Frequency 2402 ~ 2480MHz,
- FHSS/GFSK modulation, 79 channels at 1MHz intervals
- Secure communications, 128 bit encryption
- Error correction for guaranteed packet delivery
- UART local and over-the-air RF configuration
- Auto-discovery/pairing requires no software configuration (instant cable replacement).
- Auto-connect master, IO pin (DTR) and character based trigger modes

Digital I/O Characteristics

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Min</th>
<th>Typ.</th>
<th>Max.</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input logic level LOW</td>
<td>-0.4</td>
<td>-</td>
<td>+0.8</td>
<td>V</td>
</tr>
<tr>
<td>Input logic level HIGH</td>
<td>0.7VDD</td>
<td>-</td>
<td>VDD+0.4</td>
<td>V</td>
</tr>
<tr>
<td>Output logic level LOW</td>
<td>-</td>
<td>-</td>
<td>0.2</td>
<td>V</td>
</tr>
<tr>
<td>Output logic level HIGH</td>
<td>VDD-0.2</td>
<td>-</td>
<td>-</td>
<td>V</td>
</tr>
<tr>
<td>All I/O’s (except reset) default to weakpull down</td>
<td>+0.2</td>
<td>+1.0</td>
<td>+5.0</td>
<td>uA</td>
</tr>
</tbody>
</table>

Environmental Conditions

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Temperature Range (Operating)</td>
<td>-40 °C ~ 85 °C</td>
</tr>
<tr>
<td>Temperature Range (Storage)</td>
<td>-40 °C ~ 85 °C</td>
</tr>
<tr>
<td>Relative Humidity (Operating)</td>
<td>≤90%</td>
</tr>
<tr>
<td>Relative Humidity (Storage)</td>
<td>≤90%</td>
</tr>
</tbody>
</table>

Electrical Characteristics

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Min</th>
<th>Typ.</th>
<th>Max.</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Supply Voltage (DC)</td>
<td>3.0</td>
<td>3.3</td>
<td>3.6</td>
<td>V</td>
</tr>
<tr>
<td>Average power consumption</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Radio ON* (Discovery or Inquiry window time)</td>
<td>40</td>
<td>mA</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Connected Idle (No Sniff)</td>
<td>25</td>
<td>mA</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Connected Idle (Sniff 100 milli secs)</td>
<td>12</td>
<td>mA</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Connected with data transfer</td>
<td>40</td>
<td>45</td>
<td>50</td>
<td>mA</td>
</tr>
<tr>
<td>Deep Sleep Idle mode</td>
<td>26</td>
<td>uA</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

* If in SLAVE mode there are bursts of radio ON time which vary with the windows. Depending on how you set the windows that determines your average current.
## Radio Characteristics

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Freq. (GHz)</th>
<th>Min</th>
<th>Typ</th>
<th>Max</th>
<th>Bluetooth Specification</th>
<th>Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sensitivity @ 0.1%BER</td>
<td>2.402</td>
<td>-</td>
<td>-80</td>
<td>-86</td>
<td>≤ -70</td>
<td>dBm</td>
</tr>
<tr>
<td></td>
<td>2.441</td>
<td>-</td>
<td>-80</td>
<td>-86</td>
<td>dBm</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2.480</td>
<td>-</td>
<td>-80</td>
<td>-86</td>
<td>dBm</td>
<td></td>
</tr>
<tr>
<td>RF Transmit Power</td>
<td>2.402</td>
<td>0</td>
<td>2</td>
<td>4</td>
<td>≤ 4</td>
<td>dBm</td>
</tr>
<tr>
<td></td>
<td>2.441</td>
<td>0</td>
<td>2</td>
<td>4</td>
<td>dBm</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2.480</td>
<td>0</td>
<td>2</td>
<td>4</td>
<td>dBm</td>
<td></td>
</tr>
<tr>
<td>Initial Carrier Frequency Tolerance</td>
<td>2.402</td>
<td>-</td>
<td>5</td>
<td>75</td>
<td>75</td>
<td>kHz</td>
</tr>
<tr>
<td></td>
<td>2.441</td>
<td>-</td>
<td>5</td>
<td>75</td>
<td>kHz</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2.480</td>
<td>-</td>
<td>5</td>
<td>75</td>
<td>kHz</td>
<td></td>
</tr>
<tr>
<td>20dB bandwidth for modulated carrier</td>
<td>-</td>
<td>900</td>
<td>1000</td>
<td>≤ 1000</td>
<td>kHz</td>
<td></td>
</tr>
<tr>
<td>Drift (Five slots packet)</td>
<td>-</td>
<td>15</td>
<td>-</td>
<td>40</td>
<td>kHz</td>
<td></td>
</tr>
<tr>
<td>Drift Rate</td>
<td>-</td>
<td>13</td>
<td>-</td>
<td>20</td>
<td>kHz</td>
<td></td>
</tr>
<tr>
<td>$\Delta f_{1_{avg}}$ Max Modulation</td>
<td>2.402</td>
<td>140</td>
<td>165</td>
<td>175</td>
<td>&gt;140</td>
<td>kHz</td>
</tr>
<tr>
<td></td>
<td>2.441</td>
<td>140</td>
<td>165</td>
<td>175</td>
<td>kHz</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2.480</td>
<td>140</td>
<td>165</td>
<td>175</td>
<td>kHz</td>
<td></td>
</tr>
<tr>
<td>$\Delta f_{2_{avg}}$ Min Modulation</td>
<td>2.402</td>
<td>140</td>
<td>190</td>
<td>-</td>
<td>115</td>
<td>kHz</td>
</tr>
<tr>
<td></td>
<td>2.441</td>
<td>140</td>
<td>190</td>
<td>-</td>
<td>kHz</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2.480</td>
<td>140</td>
<td>190</td>
<td>-</td>
<td>kHz</td>
<td></td>
</tr>
</tbody>
</table>

## Range Characteristics (Approximate range in office environment)

<table>
<thead>
<tr>
<th>Range</th>
<th>RN-42</th>
</tr>
</thead>
<tbody>
<tr>
<td>After One Wall</td>
<td>55 feet</td>
</tr>
<tr>
<td>After Two Walls</td>
<td>60 feet</td>
</tr>
<tr>
<td>After Three Walls</td>
<td>36 feet</td>
</tr>
</tbody>
</table>

The above readings are approximate and may vary depending upon the RF environment. Bluetooth hops in a pseudo-random fashion over the 79 frequencies in the ISM band to adapt to the interference. Data throughput and range vary depending on the RF interference environment.
## Pin Description

<table>
<thead>
<tr>
<th>Pin</th>
<th>Name</th>
<th>Description</th>
<th>Default</th>
<th>Voltage</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>GND</td>
<td>No Connect</td>
<td>0V</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>SPI_MOSI</td>
<td>Programming only</td>
<td>No Connect</td>
<td>3V</td>
</tr>
<tr>
<td>3</td>
<td>PIO6</td>
<td>Set BT master (HIGH=auto-master mode)</td>
<td>Input to RN42 with weak pullup</td>
<td>0V-3.3V</td>
</tr>
<tr>
<td>4</td>
<td>PIO7</td>
<td>Set Baud rate (HIGH = force 9600, LOW = 115K or firmware setting)</td>
<td>Input to RN42 with weak pullup</td>
<td>0V-3.3V</td>
</tr>
<tr>
<td>5</td>
<td>RESET</td>
<td>Active LOW reset</td>
<td>No Connect</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>SPI_CLK</td>
<td>Programming only</td>
<td>No Connect</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>PCM_CLK</td>
<td>PCM interface</td>
<td>No Connect</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>PCM_SYNC</td>
<td>PCM interface</td>
<td>No Connect</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>PCM_IN</td>
<td>PCM interface</td>
<td>No Connect</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>PCM_OUT</td>
<td>PCM interface</td>
<td>No Connect</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>VDD</td>
<td>3.3V regulated power input</td>
<td>No Connect</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>GND</td>
<td></td>
<td>No Connect</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>UART_RX</td>
<td>UART receive input</td>
<td>Input to RN42</td>
<td>0V-3.3V</td>
</tr>
<tr>
<td>14</td>
<td>UART_TX</td>
<td>UART transmit output</td>
<td>High level output from RN42</td>
<td>0V-3.3V</td>
</tr>
<tr>
<td>15</td>
<td>UART_RTS</td>
<td>UART RTS, goes HIGH to disable host transmitter</td>
<td>Low level output from RN42</td>
<td>0V-3.3V</td>
</tr>
<tr>
<td>16</td>
<td>UART_CTS</td>
<td>UART CTS, if set HIGH, disables transmitter</td>
<td>Low level input to RN42</td>
<td>0V-3.3V</td>
</tr>
<tr>
<td>17</td>
<td>USB_D+</td>
<td>USB port</td>
<td>Pull up 1.5K when active</td>
<td>0V-3.3V</td>
</tr>
<tr>
<td>18</td>
<td>USB_D-</td>
<td>USB port</td>
<td>Pull up 1.5K when active</td>
<td>0V-3.3V</td>
</tr>
<tr>
<td>19</td>
<td>PIO2</td>
<td>Status, HIGH when connected, LOW otherwise</td>
<td>Output from RN42</td>
<td>0V-3.3V</td>
</tr>
<tr>
<td>20</td>
<td>PIO3</td>
<td>Auto discovery = HIGH</td>
<td>Input to RN42 with weak pullup</td>
<td>0V-3.3V</td>
</tr>
<tr>
<td>21</td>
<td>PIO5</td>
<td>Status, toggles based on state, LOW on connect</td>
<td>Output from RN42</td>
<td>0V-3.3V</td>
</tr>
<tr>
<td>22</td>
<td>PIO4</td>
<td>Set factory defaults</td>
<td>Input to RN42 with weak pullup</td>
<td>0V-3.3V</td>
</tr>
<tr>
<td>23</td>
<td>SPI_CSBI</td>
<td>Programming only</td>
<td>No Connect</td>
<td></td>
</tr>
<tr>
<td>24</td>
<td>SPI_MISO</td>
<td>Programming only</td>
<td>No Connect</td>
<td></td>
</tr>
<tr>
<td>25</td>
<td>GND</td>
<td>GND for RN42-N</td>
<td>Not Used</td>
<td></td>
</tr>
<tr>
<td>26</td>
<td>RF Pad</td>
<td>RF Pad for RN42-N</td>
<td>Not Used</td>
<td></td>
</tr>
<tr>
<td>27</td>
<td>GND</td>
<td>GND for RN42-N</td>
<td>Not Used</td>
<td></td>
</tr>
<tr>
<td>30</td>
<td>AIO0</td>
<td>Optional analog input</td>
<td>Not Used</td>
<td></td>
</tr>
<tr>
<td>31</td>
<td>AIO8</td>
<td>Status (RF data rx/tx)</td>
<td>Output from RN42</td>
<td>0V-3.3V</td>
</tr>
<tr>
<td>32</td>
<td>PIO9</td>
<td>IO</td>
<td>Input to RN42 with weak pullup</td>
<td>0V-3.3V</td>
</tr>
<tr>
<td>33</td>
<td>PIO10</td>
<td>IO (remote DTR signal)</td>
<td>Input to RN42 with weak pullup</td>
<td>0V-3.3V</td>
</tr>
<tr>
<td>34</td>
<td>PIO11</td>
<td>IO (remote RTS signal)</td>
<td>Input to RN42 with weak pullup</td>
<td>0V-3.3V</td>
</tr>
<tr>
<td>35</td>
<td>AIO1</td>
<td>Optional analog input</td>
<td>Not Used</td>
<td></td>
</tr>
</tbody>
</table>
Typical Application Circuit

Since the RN 41 and RN 42 are functionally compatible, this application diagram applies to RN 41 and RN 42.
RN-42 Module Dimensions

PAD SIZE = 0.8 x 1.30 mm

NOTE: All dimensions are in mm
RN-42-N Module Dimensions

PAD SIZE = 0.8 x 1.30 mm

NOTE: All dimensions are in mm
Design Concerns

1. **Reset circuit.** RN-42 contains a weak pullup to VCC, the polarity of reset on the RN42 is ACTIVE LOW. A power on reset circuit with delay is OPTIONAL on the reset pin of the module. It should only be required if the input power supply has a very slow ramp, or tends to bounce or have instability on power up. Often a microcontroller or embedded CPU IO is available to generate reset once power is stable. If not, there are many low cost power supervisor chips available, such as MCP809, MCP102/121, and Torex XC61F.

2. **Factory reset PIO4.** It is a good idea to connect this pin to a switch, or jumper, or resistor, so it can be accessed. This pin can be used to reset the module to FACTORY DEFAULTS and is often critical in situations where the module has been mis-configured. To set Factory defaults start HIGH, then toggle two times.

3. **Connection status.** PIO5 is available to drive an LED, and blinks at various speeds to indicate status. PIO2 is an output which directly reflects the connection state, it goes HIGH when connected, and LOW otherwise.

4. **HCI mode.** The RN42 module must be loaded with special firmware to run in HCI mode. When in HCI mode the standard SPP/DUN applications are disabled.

5. **Using SPI bus for flash upgrade.** While not required, this bus is very useful for configuring advanced parameters of the Bluetooth modules, and is required for upgrading the firmware on modules. The suggested ref-design shows a 6pin header which can be implemented to gain access to this bus. A minimum-mode version could just use the SPI signals (4pins) and pickup ground and VCC from elsewhere on the design.

6. **Minimizing Radio interference.** When laying out the carrier board for the RN42 module the areas under the antenna and shielding connections should not have surface traces, GND planes, or exposed vias. (See diagram to right) For optimal radio performance the antenna end of RN42 module should protrude 5mm past any metal enclosure.
7. **Antenna Design.** The pattern from the rf_out terminal pad should be designed with 50ohms impedance and traced with straight lines. (see diagram to the right) The rf_out signal line should not run under or near the RN21 module. The GND plane should be on the side of the PCB which the module is mounted. The GND should be reinforced with through-hole connections and other means to stabilize the electric potential.

8. **Soldering Reflow Profile.**
   - Lead-Free Solder Reflow
   - Temp: 230 degree C, 30-40 seconds, Peak 250 degree C maximum.
   - Preheat temp: 165 ± 15 degree C, 90 to 120 seconds.
   - Time: Single Pass, One Time

### Compliance Information

<table>
<thead>
<tr>
<th>Category</th>
<th>Country</th>
<th>Standard</th>
</tr>
</thead>
</table>
| Radio    | USA     | FCC Part 15 Subpart B: 2008 Class B  
          |         | FCC CRF Title 47 Part 15 Subpart C  
          |         | FCC ID: T9J-RN42 |
|          | EUROPE  | ETSI EN 301 489-1 V1.8.1  
          |         | ETSI EN 301 489-17 V2.1.1  
          |         | ETSI EN 300 328 V1.7.1 |
|          | CANADA  | IC RSS-210 low power comm. device  
          |         | Certification Number: 6514A-RN42 |
| EMC      | USA     | FCC CFR47 Part 15 subclass B  
          |         | EN 55022 Class B radiated  
          |         | EN61000-4-2 ESD immunity  
          |         | EN61000-4-3 radiated field  
          |         | EN61000-4-6 RF immunity  
          |         | EN61000-4-8 power magnetic immunity |
| Bluetooth| BQB LISTED | B014867- SPP and DUN profiles |
| Environmental | RoHS | RoHS compliant |

### Ordering Information

<table>
<thead>
<tr>
<th>Part Number</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RN-42</td>
<td>Standard Application firmware (SPP/DUN Master and Slave)</td>
</tr>
<tr>
<td>RN-42-H</td>
<td>HCI firmware (HCI over H4 UART)</td>
</tr>
<tr>
<td>RN-42-U</td>
<td>USB firmware (HCI over USB port, slave device at 12Mbps rate)</td>
</tr>
<tr>
<td>RN-42-N</td>
<td>No Antenna, Standard Application firmware (SPP/DUN Master and Slave)</td>
</tr>
</tbody>
</table>

For other configurations, contact Roving Networks directly.

Copyright © 2011 Roving Networks. All rights reserved.

The Bluetooth trademark and logo are registered trademarks and are owned by the Bluetooth SIG, Inc. All other trademarks are property of their respective owners.

Roving Networks reserves the right to make corrections, modifications, and other changes to its products, documentation and services at any time. Customers should obtain the latest relevant information before placing orders and should verify that such information is current and complete.

Roving Networks assumes no liability for applications assistance or customer product design. Customers are responsible for their products and applications using Roving Networks components. To minimize the risks associated with customer products and applications, customers should provide adequate design and operating safeguards.

Roving Networks products are not authorized for use in safety-critical applications (such as life support) where a failure of the Roving Networks product would reasonably be expected to cause severe personal injury or death, unless officers of the parties have executed an agreement specifically governing such use.
Features

- High Performance, Low Power AVR® 8-Bit Microcontroller
- Advanced RISC Architecture
  - 135 Powerful Instructions – Most Single Clock Cycle Execution
  - 32 x 8 General Purpose Working Registers
  - Fully Static Operation
  - Up to 16 MIPS Throughput at 16 MHz
  - On-Chip 2-cycle Multiplier
- Non-volatile Program and Data Memories
  - 16/32K Bytes of In-System Self-Programmable Flash (ATmega16U4/ATmega32U4)
  - 1.25/2.5K Bytes Internal SRAM (ATmega16U4/ATmega32U4)
  - 512Bytes/1K Bytes Internal EEPROM (ATmega16U4/ATmega32U4)
  - Write/Erase Cycles: 10,000 Flash/100,000 EEPROM
  - Data retention: 20 years at 85°C/ 100 years at 25°C(1)
- Optional Boot Code Section with Independent Lock Bits
  - In-System Programming by On-chip Boot Program
  - True Read-While-Write Operation
  - All supplied parts are preprogrammed with a default USB bootloader
- Programming Lock for Software Security
- JTAG (IEEE std. 1149.1 compliant) Interface
  - Boundary-scan Capabilities According to the JTAG Standard
  - Extensive On-chip Debug Support
  - Programming of Flash, EEPROM, Fuses, and Lock Bits through the JTAG Interface
- USB 2.0 Full-speed/Low Speed Device Module with Interrupt on Transfer Completion
  - Complies fully with Universal Serial Bus Specification Rev 2.0
  - Supports data transfer rates up to 12 Mbit/s and 1.5 Mbit/s
  - Endpoint 0 for Control Transfers: up to 64-bytes
  - 6 Programmable Endpoints with IN or Out Directions and with Bulk, Interrupt or Isochronous Transfers
  - Configurable Endpoints size up to 256 bytes in double bank mode
  - Fully independent 832 bytes USB DPRAM for endpoint memory allocation
  - Suspend/Resume Interrupts
  - CPU Reset possible on USB Bus Reset detection
  - 48 MHz from PLL for Full-speed Bus Operation
  - USB Bus Connection/Disconnection on Microcontroller Request
  - Crystal-less operation for Low speed mode
- Peripheral Features
  - On-chip PLL for USB and High Speed Timer: 32 up to 96 MHz operation
  - One 8-bit Timer/Counter with Separate Prescaler and Compare Mode
  - Two 16-bit Timer/Counter with Separate Prescaler, Compare- and Capture Mode
  - One 10-bit High-Speed Timer/Counter with PLL (64 MHz) and Compare Mode
  - Four 8-bit PWM Channels
  - Four PWM Channels with Programmable Resolution from 2 to 16 Bits
  - Six PWM Channels for High Speed Operation, with Programmable Resolution from 2 to 11 Bits
  - Output Compare Modulator
  - 12-channels, 10-bit ADC (features Differential Channels with Programmable Gain)
  - Programmable Serial USART with Hardware Flow Control
  - Master/Slave SPI Serial Interface

8-bit AVR® Microcontroller with 16/32K Bytes of ISP Flash and USB Controller

ATmega16U4
ATmega32U4

Preliminary
– Byte Oriented 2-wire Serial Interface
– Programmable Watchdog Timer with Separate On-chip Oscillator
– On-chip Analog Comparator
– Interrupt and Wake-up on Pin Change
– On-chip Temperature Sensor

• Special Microcontroller Features
  – Power-on Reset and Programmable Brown-out Detection
  – Internal 8 MHz Calibrated Oscillator
  – Internal clock prescaler & On-the-fly Clock Switching (Int RC / Ext Osc)
  – External and Internal Interrupt Sources
  – Six Sleep Modes: Idle, ADC Noise Reduction, Power-save, Power-down, Standby, and Extended Standby

• I/O and Packages
  – All I/O combine CMOS outputs and LVTTL inputs
  – 26 Programmable I/O Lines
  – 44-lead TQFP Package, 10x10mm
  – 44-lead QFN Package, 7x7mm

• Operating Voltages
  – 2.7 - 5.5V

• Operating temperature
  – Industrial (-40°C to +85°C)

• Maximum Frequency
  – 8 MHz at 2.7V - Industrial range
  – 16 MHz at 4.5V - Industrial range

Note: 1. See “Data Retention” on page 8 for details.
1. Pin Configurations

Figure 1-1. Pinout ATmega16U4/ATmega32U4

2. Overview

The ATmega16U4/ATmega32U4 is a low-power CMOS 8-bit microcontroller based on the AVR enhanced RISC architecture. By executing powerful instructions in a single clock cycle, the ATmega16U4/ATmega32U4 achieves throughputs approaching 1 MIPS per MHz allowing the system designer to optimize power consumption versus processing speed.
The AVR core combines a rich instruction set with 32 general purpose working registers. All the 32 registers are directly connected to the Arithmetic Logic Unit (ALU), allowing two independent registers to be accessed in one single instruction executed in one clock cycle. The resulting architecture is more code efficient while achieving throughputs up to ten times faster than conventional CISC microcontrollers.

The ATmega16U4/ATmega32U4 provides the following features: 16/32K bytes of In-System Programmable Flash with Read-While-Write capabilities, 512Bytes/1K bytes EEPROM, 1.25/2.5K bytes SRAM, 26 general purpose I/O lines (CMOS outputs and LV TTL inputs), 32 general purpose working registers, four flexible Timer/Counters with compare modes and PWM, one more high-speed Timer/Counter with compare modes and PLL adjustable source, one USART (including CTS/RTS flow control signals), a byte oriented 2-wire Serial Interface, a 12-
channels 10-bit ADC with optional differential input stage with programmable gain, an on-chip calibrated temperature sensor, a programmable Watchdog Timer with Internal Oscillator, an SPI serial port, IEEE std. 1149.1 compliant JTAG test interface, also used for accessing the On-chip Debug system and programming and six software selectable power saving modes. The Idle mode stops the CPU while allowing the SRAM, Timer/Counters, SPI port, and interrupt system to continue functioning. The Power-down mode saves the register contents but freezes the Oscillator, disabling all other chip functions until the next interrupt or Hardware Reset. The ADC Noise Reduction mode stops the CPU and all I/O modules except ADC, to minimize switching noise during ADC conversions. In Standby mode, the Crystal/Resonator Oscillator is running while the rest of the device is sleeping. This allows very fast start-up combined with low power consumption.

The device is manufactured using ATMEL’s high-density nonvolatile memory technology. The On-chip ISP Flash allows the program memory to be reprogrammed in-system through an SPI serial interface, by a conventional nonvolatile memory programmer, or by an On-chip Boot program running on the AVR core. The boot program can use any interface to download the application program in the application Flash memory. Software in the Boot Flash section will continue to run while the Application Flash section is updated, providing true Read-While-Write operation. By combining an 8-bit RISC CPU with In-System Self-Programmable Flash on a monolithic chip, the ATMEL ATmega16U4/ATmega32U4 is a powerful microcontroller that provides a highly flexible and cost effective solution to many embedded control applications.

The ATmega16U4/ATmega32U4 AVR is supported with a full suite of program and system development tools including: C compilers, macro assemblers, program debugger/simulators, in-circuit emulators, and evaluation kits.

2.2 Pin Descriptions

2.2.1 VCC

Digital supply voltage.

2.2.2 GND

Ground.

2.2.3 Port B (PB7..PB0)

Port B is an 8-bit bi-directional I/O port with internal pull-up resistors (selected for each bit). The Port B output buffers have symmetrical drive characteristics with both high sink and source capability. As inputs, Port B pins that are externally pulled low will source current if the pull-up resistors are activated. The Port B pins are tri-stated when a reset condition becomes active, even if the clock is not running.

Port B has better driving capabilities than the other ports.

Port B also serves the functions of various special features of the ATmega16U4/ATmega32U4 as listed on page 72.

2.2.4 Port C (PC7,PC6)

Port C is an 8-bit bi-directional I/O port with internal pull-up resistors (selected for each bit). The Port C output buffers have symmetrical drive characteristics with both high sink and source capability. As inputs, Port C pins that are externally pulled low will source current if the pull-up resistors are activated. The Port C pins are tri-stated when a reset condition becomes active, even if the clock is not running.
Only bits 6 and 7 are present on the product pinout.

Port C also serves the functions of special features of the ATmega16U4/ATmega32U4 as listed on page 75.

2.2.5 Port D (PD7..PD0)
Port D is an 8-bit bi-directional I/O port with internal pull-up resistors (selected for each bit). The Port D output buffers have symmetrical drive characteristics with both high sink and source capability. As inputs, Port D pins that are externally pulled low will source current if the pull-up resistors are activated. The Port D pins are tri-stated when a reset condition becomes active, even if the clock is not running.

Port D also serves the functions of various special features of the ATmega16U4/ATmega32U4 as listed on page 77.

2.2.6 Port E (PE6,PE2)
Port E is an 8-bit bi-directional I/O port with internal pull-up resistors (selected for each bit). The Port E output buffers have symmetrical drive characteristics with both high sink and source capability. As inputs, Port E pins that are externally pulled low will source current if the pull-up resistors are activated. The Port E pins are tri-stated when a reset condition becomes active, even if the clock is not running.

Only bits 2 and 6 are present on the product pinout.

Port E also serves the functions of various special features of the ATmega16U4/ATmega32U4 as listed on page 80.

2.2.7 Port F (PF7..PF4, PF1,PF0)
Port F serves as analog inputs to the A/D Converter.

Port F also serves as an 8-bit bi-directional I/O port, if the A/D Converter channels are not used. Port pins can provide internal pull-up resistors (selected for each bit). The Port F output buffers have symmetrical drive characteristics with both high sink and source capability. As inputs, Port F pins that are externally pulled low will source current if the pull-up resistors are activated. The Port F pins are tri-stated when a reset condition becomes active, even if the clock is not running.

Bits 2 and 3 are not present on the product pinout.

Port F also serves the functions of the JTAG interface. If the JTAG interface is enabled, the pull-up resistors on pins PF7(TDI), PF5(TMS), and PF4(TCK) will be activated even if a reset occurs.

2.2.8 D-
USB Full speed / Low Speed Negative Data Upstream Port. Should be connected to the USB D-connector pin with a serial 22 Ohms resistor.

2.2.9 D+
USB Full speed / Low Speed Positive Data Upstream Port. Should be connected to the USB D+ connector pin with a serial 22 Ohms resistor.

2.2.10 UGND
USB Pads Ground.
2.2.11  UVCC
USB Pads Internal Regulator Input supply voltage.

2.2.12  UCAP
USB Pads Internal Regulator Output supply voltage. Should be connected to an external capacitor (1μF).

2.2.13  VBUS
USB VBUS monitor input.

2.2.14  RESET
Reset input. A low level on this pin for longer than the minimum pulse length will generate a reset, even if the clock is not running. The minimum pulse length is given in Table 8-1 on page 50. Shorter pulses are not guaranteed to generate a reset.

2.2.15  XTAL1
Input to the inverting Oscillator amplifier and input to the internal clock operating circuit.

2.2.16  XTAL2
Output from the inverting Oscillator amplifier.

2.2.17  AVCC
AVCC is the supply voltage pin (input) for all the A/D Converter channels. If the ADC is not used, it should be externally connected to VCC. If the ADC is used, it should be connected to VCC through a low-pass filter.

2.2.18  AREF
This is the analog reference pin (input) for the A/D Converter.
3. About

3.1 Disclaimer

Typical values contained in this datasheet are based on simulations and characterization of other AVR microcontrollers manufactured on the same process technology. Min and Max values will be available after the device is characterized.

3.2 Resources

A comprehensive set of development tools, application notes and datasheets are available for download on http://www.atmel.com/avr.

3.3 Code Examples

This documentation contains simple code examples that briefly show how to use various parts of the device. Be aware that not all C compiler vendors include bit definitions in the header files and interrupt handling in C is compiler dependent. Please confirm with the C compiler documentation for more details.

These code examples assume that the part specific header file is included before compilation. For I/O registers located in extended I/O map, "IN", "OUT", "SBIS", "SBIC", "CBI", and "SBI" instructions must be replaced with instructions that allow access to extended I/O. Typically "LDS" and "STS" combined with "SBRS", "SBRC", "SBR", and "CBR".

3.4 Data Retention

Reliability Qualification results show that the projected data retention failure rate is much less than 1 PPM over 20 years at 85°C or 100 years at 25°C.
4. AVR CPU Core

4.1 Introduction

This section discusses the AVR core architecture in general. The main function of the CPU core is to ensure correct program execution. The CPU must therefore be able to access memories, perform calculations, control peripherals, and handle interrupts.

4.2 Architectural Overview

**Figure 4-1.** Block Diagram of the AVR Architecture

In order to maximize performance and parallelism, the AVR uses a Harvard architecture – with separate memories and buses for program and data. Instructions in the program memory are executed with a single level pipelining. While one instruction is being executed, the next instruction is pre-fetched from the program memory. This concept enables instructions to be executed in every clock cycle. The program memory is In-System Reprogrammable Flash memory.
The fast-access Register File contains 32 x 8-bit general purpose working registers with a single clock cycle access time. This allows single-cycle Arithmetic Logic Unit (ALU) operation. In a typical ALU operation, two operands are output from the Register File, the operation is executed, and the result is stored back in the Register File – in one clock cycle.

Six of the 32 registers can be used as three 16-bit indirect address register pointers for Data Space addressing – enabling efficient address calculations. One of these address pointers can also be used as an address pointer for look up tables in Flash program memory. These added function registers are the 16-bit X-, Y-, and Z-register, described later in this section.

The ALU supports arithmetic and logic operations between registers or between a constant and a register. Single register operations can also be executed in the ALU. After an arithmetic operation, the Status Register is updated to reflect information about the result of the operation.

Program flow is provided by conditional and unconditional jump and call instructions, able to directly address the whole address space. Most AVR instructions have a single 16-bit word format. Every program memory address contains a 16- or 32-bit instruction.

Program Flash memory space is divided in two sections, the Boot Program section and the Application Program section. Both sections have dedicated Lock bits for write and read/write protection. The SPM instruction that writes into the Application Flash memory section must reside in the Boot Program section.

During interrupts and subroutine calls, the return address Program Counter (PC) is stored on the Stack. The Stack is effectively allocated in the general data SRAM, and consequently the Stack size is only limited by the total SRAM size and the usage of the SRAM. All user programs must initialize the SP in the Reset routine (before subroutines or interrupts are executed). The Stack Pointer (SP) is read/write accessible in the I/O space. The data SRAM can easily be accessed through the five different addressing modes supported in the AVR architecture.

The memory spaces in the AVR architecture are all linear and regular memory maps.

A flexible interrupt module has its control registers in the I/O space with an additional Global Interrupt Enable bit in the Status Register. All interrupts have a separate Interrupt Vector in the Interrupt Vector table. The interrupts have priority in accordance with their Interrupt Vector position. The lower the Interrupt Vector address, the higher the priority.

The I/O memory space contains 64 addresses for CPU peripheral functions as Control Registers, SPI, and other I/O functions. The I/O Memory can be accessed directly, or as the Data Space locations following those of the Register File, 0x20 - 0x5F. In addition, the ATmega16U4/ATmega32U4 has Extended I/O space from 0x60 - 0xFFFF in SRAM where only the ST/STS/STD and LD/LDS/LDD instructions can be used.

4.3 ALU – Arithmetic Logic Unit

The high-performance AVR ALU operates in direct connection with all the 32 general purpose working registers. Within a single clock cycle, arithmetic operations between general purpose registers or between a register and an immediate are executed. The ALU operations are divided into three main categories – arithmetic, logical, and bit-functions. Some implementations of the architecture also provide a powerful multiplier supporting both signed/unsigned multiplication and fractional format. See the “Instruction Set” section for a detailed description.
4.4 Status Register

The Status Register contains information about the result of the most recently executed arithmetic instruction. This information can be used for altering program flow in order to perform conditional operations. Note that the Status Register is updated after all ALU operations, as specified in the Instruction Set Reference. This will in many cases remove the need for using the dedicated compare instructions, resulting in faster and more compact code.

The Status Register is not automatically stored when entering an interrupt routine and restored when returning from an interrupt. This must be handled by software.

The AVR Status Register – SREG – is defined as:

<table>
<thead>
<tr>
<th>Bit</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read/Write</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td></td>
</tr>
<tr>
<td>Initial Value</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

- **Bit 7 – I**: Global Interrupt Enable
  The Global Interrupt Enable bit must be set for the interrupts to be enabled. The individual interrupt enable control is then performed in separate control registers. If the Global Interrupt Enable Register is cleared, none of the interrupts are enabled independent of the individual interrupt enable settings. The I-bit is cleared by hardware after an interrupt has occurred, and is set by the RETI instruction to enable subsequent interrupts. The I-bit can also be set and cleared by the application with the SEI and CLI instructions, as described in the instruction set reference.

- **Bit 6 – T**: Bit Copy Storage
  The Bit Copy instructions BLD (Bit LoaD) and BST (Bit STore) use the T-bit as source or destination for the operated bit. A bit from a register in the Register File can be copied into T by the BST instruction, and a bit in T can be copied into a bit in a register in the Register File by the BLD instruction.

- **Bit 5 – H**: Half Carry Flag
  The Half Carry Flag H indicates a Half Carry in some arithmetic operations. Half Carry Is useful in BCD arithmetic. See the “Instruction Set Description” for detailed information.

- **Bit 4 – S**: Sign Bit, S = N ⊕ V
  The S-bit is always an exclusive or between the Negative Flag N and the Two’s Complement Overflow Flag V. See the “Instruction Set Description” for detailed information.

- **Bit 3 – V**: Two’s Complement Overflow Flag
  The Two’s Complement Overflow Flag V supports two’s arithmetic complements. See the “Instruction Set Description” for detailed information.

- **Bit 2 – N**: Negative Flag
  The Negative Flag N indicates a negative result in an arithmetic or logic operation. See the “Instruction Set Description” for detailed information.

- **Bit 1 – Z**: Zero Flag
  The Zero Flag Z indicates a zero result in an arithmetic or logic operation. See the “Instruction Set Description” for detailed information.
• **Bit 0 – C: Carry Flag**
The Carry Flag C indicates a carry in an arithmetic or logic operation. See the “Instruction Set Description” for detailed information.

### 4.5 General Purpose Register File

The Register File is optimized for the AVR Enhanced RISC instruction set. In order to achieve the required performance and flexibility, the following input/output schemes are supported by the Register File:

- One 8-bit output operand and one 8-bit result input
- Two 8-bit output operands and one 8-bit result input
- Two 8-bit output operands and one 16-bit result input
- One 16-bit output operand and one 16-bit result input

**Figure 4-2** shows the structure of the 32 general purpose working registers in the CPU.

#### Figure 4-2. AVR CPU General Purpose Working Registers

<table>
<thead>
<tr>
<th>7</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>R0</td>
<td>0x00</td>
</tr>
<tr>
<td>R1</td>
<td>0x01</td>
</tr>
<tr>
<td>R2</td>
<td>0x02</td>
</tr>
<tr>
<td>...</td>
<td>0x0D</td>
</tr>
<tr>
<td>R13</td>
<td>0x0E</td>
</tr>
<tr>
<td>R14</td>
<td>0x0F</td>
</tr>
<tr>
<td>R15</td>
<td>0x10</td>
</tr>
<tr>
<td>R16</td>
<td>0x11</td>
</tr>
<tr>
<td>...</td>
<td>0x1A</td>
</tr>
<tr>
<td>R26</td>
<td>0x1B</td>
</tr>
<tr>
<td>R27</td>
<td>0x1C</td>
</tr>
<tr>
<td>R28</td>
<td>0x1D</td>
</tr>
<tr>
<td>R29</td>
<td>0x1E</td>
</tr>
<tr>
<td>R30</td>
<td>0x1F</td>
</tr>
<tr>
<td>R31</td>
<td>0x20</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>General Purpose Working Registers</th>
</tr>
</thead>
</table>

Most of the instructions operating on the Register File have direct access to all registers, and most of them are single cycle instructions.

As shown in **Figure 4-2**, each register is also assigned a data memory address, mapping them directly into the first 32 locations of the user Data Space. Although not being physically implemented as SRAM locations, this memory organization provides great flexibility in access of the registers, as the X-, Y- and Z-pointer registers can be set to index any register in the file.

#### 4.5.1 The X-register, Y-register, and Z-register

The registers R26..R31 have some added functions to their general purpose usage. These registers are 16-bit address pointers for indirect addressing of the data space. The three indirect address registers X, Y, and Z are defined as described in **Figure 4-3**.
Figure 4-3. The X-, Y-, and Z-registers

<table>
<thead>
<tr>
<th>Bit</th>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>SP15</td>
<td>SP14</td>
<td>SP13</td>
<td>SP12</td>
<td>SP11</td>
<td>SP10</td>
<td>SP9</td>
<td>SP8</td>
<td></td>
</tr>
<tr>
<td>SP7</td>
<td>SP6</td>
<td>SP5</td>
<td>SP4</td>
<td>SP3</td>
<td>SP2</td>
<td>SP1</td>
<td>SP0</td>
<td></td>
</tr>
<tr>
<td>Read/Write</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td></td>
</tr>
<tr>
<td>Initial Value</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

In the different addressing modes these address registers have functions as fixed displacement, automatic increment, and automatic decrement (see the instruction set reference for details).

4.6 Stack Pointer

The Stack is mainly used for storing temporary data, for storing local variables and for storing return addresses after interrupts and subroutine calls. The Stack Pointer Register always points to the top of the Stack. Note that the Stack is implemented as growing from higher memory locations to lower memory locations. This implies that a Stack PUSH command decreases the Stack Pointer.

The Stack Pointer points to the data SRAM Stack area where the Subroutine and Interrupt Stacks are located. This Stack space in the data SRAM must be defined by the program before any subroutine calls are executed or interrupts are enabled. The Stack Pointer must be set to point above 0x0100. The initial value of the stack pointer is the last address of the internal SRAM. The Stack Pointer is decremented by one when data is pushed onto the Stack with the PUSH instruction, and it is decremented by three when the return address is pushed onto the Stack with subroutine call or interrupt. The Stack Pointer is incremented by one when data is popped from the Stack with the POP instruction, and it is incremented by three when data is popped from the Stack with return from subroutine RET or return from interrupt RETI.

The AVR Stack Pointer is implemented as two 8-bit registers in the I/O space. The number of bits actually used is implementation dependent. Note that the data space in some implementations of the AVR architecture is so small that only SPL is needed. In this case, the SPH Register will not be present.
4.6.1 Extended Z-pointer Register for ELPM/SPM - RAMPZ

<table>
<thead>
<tr>
<th>Bit</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAMPZ7</td>
<td>RAMPZ6</td>
<td>RAMPZ5</td>
<td>RAMPZ4</td>
<td>RAMPZ3</td>
<td>RAMPZ2</td>
<td>RAMPZ1</td>
<td>RAMPZ0</td>
<td>RAMPZ</td>
</tr>
<tr>
<td>Read/Write</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
</tr>
<tr>
<td>Initial Value</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

For ELPM/SPM instructions, the Z-pointer is a concatenation of RAMPZ, ZH, and ZL, as shown in Figure 4-4. Note that LPM is not affected by the RAMPZ setting.

**Figure 4-4.** The Z-pointer used by ELPM and SPM

The actual number of bits is implementation dependent. Unused bits in an implementation will always read as zero. For compatibility with future devices, be sure to write these bits to zero.

4.7 Instruction Execution Timing

This section describes the general access timing concepts for instruction execution. The AVR CPU is driven by the CPU clock \( \text{clk}_{\text{CPU}} \), directly generated from the selected clock source for the chip. No internal clock division is used.

**Figure 4-5** shows the parallel instruction fetches and instruction executions enabled by the Harvard architecture and the fast-access Register File concept. This is the basic pipelining concept to obtain up to 1 MIPS per MHz with the corresponding unique results for functions per cost, functions per clocks, and functions per power-unit.

**Figure 4-5.** The Parallel Instruction Fetches and Instruction Executions

**Figure 4-6** shows the internal timing concept for the Register File. In a single clock cycle an ALU operation using two register operands is executed, and the result is stored back to the destination register.
4.8 Reset and Interrupt Handling

The AVR provides several different interrupt sources. These interrupts and the separate Reset Vector each have a separate program vector in the program memory space. All interrupts are assigned individual enable bits which must be written logic one together with the Global Interrupt Enable bit in the Status Register in order to enable the interrupt. Depending on the Program Counter value, interrupts may be automatically disabled when Boot Lock bits BLB02 or BLB12 are programmed. This feature improves software security. See the section “Memory Programming” on page 346 for details.

The lowest addresses in the program memory space are by default defined as the Reset and Interrupt Vectors. The complete list of vectors is shown in “Interrupts” on page 61. The list also determines the priority levels of the different interrupts. The lower the address the higher is the priority level. RESET has the highest priority, and next is INT0 – the External Interrupt Request 0. The Interrupt Vectors can be moved to the start of the Boot Flash section by setting the IVSEL bit in the MCU Control Register (MCUCR). Refer to “Interrupts” on page 61 for more information. The Reset Vector can also be moved to the start of the Boot Flash section by programming the BOOTSTR Fuse, see “Memory Programming” on page 346.

When an interrupt occurs, the Global Interrupt Enable I-bit is cleared and all interrupts are disabled. The user software can write logic one to the I-bit to enable nested interrupts. All enabled interrupts can then interrupt the current interrupt routine. The I-bit is automatically set when a Return from Interrupt instruction – RETI – is executed.

There are basically two types of interrupts. The first type is triggered by an event that sets the Interrupt Flag. For these interrupts, the Program Counter is vectored to the actual Interrupt Vector in order to execute the interrupt handling routine, and hardware clears the corresponding Interrupt Flag. Interrupt Flags can also be cleared by writing a logic one to the flag bit position(s) to be cleared. If an interrupt condition occurs while the corresponding interrupt enable bit is cleared, the Interrupt Flag will be set and remembered until the interrupt is enabled, or the flag is cleared by software. Similarly, if one or more interrupt conditions occur while the Global Interrupt Enable bit is cleared, the corresponding Interrupt Flag(s) will be set and remembered until the Global Interrupt Enable bit is set, and will then be executed by order of priority.

The second type of interrupts will trigger as long as the interrupt condition is present. These interrupts do not necessarily have Interrupt Flags. If the interrupt condition disappears before the interrupt is enabled, the interrupt will not be triggered.

When the AVR exits from an interrupt, it will always return to the main program and execute one more instruction before any pending interrupt is served.
Note that the Status Register is not automatically stored when entering an interrupt routine, nor restored when returning from an interrupt routine. This must be handled by software.

When using the CLI instruction to disable interrupts, the interrupts will be immediately disabled. No interrupt will be executed after the CLI instruction, even if it occurs simultaneously with the CLI instruction. The following example shows how this can be used to avoid interrupts during the timed EEPROM write sequence.

**Assembly Code Example**

```assembly
  in r16, SREG   ; store SREG value
  cli           ; disable interrupts during timed sequence
  sbi EECR, EEMPE ; start EEPROM write
  sbi EECR, EEPE
  out SREG, r16  ; restore SREG value (I-bit)
```

**C Code Example**

```c
  char cSREG;
  cSREG = SREG; /* store SREG value */
  /* disable interrupts during timed sequence */
  __disable_interrupt();
  EECR |= (1<<EEMPE); /* start EEPROM write */
  EECR |= (1<<EEPE);
  SREG = cSREG; /* restore SREG value (I-bit) */
```

When using the SEI instruction to enable interrupts, the instruction following SEI will be executed before any pending interrupts, as shown in this example.
Assembly Code Example

```assembly
sei ; set Global Interrupt Enable
sleep; enter sleep, waiting for interrupt
; note: will enter sleep before any pending
; interrupt(s)
```

C Code Example

```c
__enable_interrupt(); /* set Global Interrupt Enable */
__sleep(); /* enter sleep, waiting for interrupt */
/* note: will enter sleep before any pending interrupt(s) */
```

### 4.8.1 Interrupt Response Time

The interrupt execution response for all the enabled AVR interrupts is five clock cycles minimum. After five clock cycles the program vector address for the actual interrupt handling routine is executed. During these five clock cycle period, the Program Counter is pushed onto the Stack. The vector is normally a jump to the interrupt routine, and this jump takes three clock cycles. If an interrupt occurs during execution of a multi-cycle instruction, this instruction is completed before the interrupt is served. If an interrupt occurs when the MCU is in sleep mode, the interrupt execution response time is increased by five clock cycles. This increase comes in addition to the start-up time from the selected sleep mode.

A return from an interrupt handling routine takes five clock cycles. During these five clock cycles, the Program Counter (three bytes) is popped back from the Stack, the Stack Pointer is incremented by three, and the I-bit in SREG is set.
5. **AVR ATmega16U4/ATmega32U4 Memories**

This section describes the different memories in the ATmega16U4/ATmega32U4. The AVR architecture has two main memory spaces, the Data Memory and the Program Memory space. In addition, the ATmega16U4/ATmega32U4 features an EEPROM Memory for data storage. All three memory spaces are linear and regular.

### Table 5-1. Memory Mapping.

<table>
<thead>
<tr>
<th>Memory</th>
<th>Mnemonic</th>
<th>ATmega32U4</th>
<th>ATmega16U4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Flash</td>
<td>Size</td>
<td>32K bytes</td>
<td>16K bytes</td>
</tr>
<tr>
<td></td>
<td>Start Address</td>
<td>0x0000</td>
<td></td>
</tr>
<tr>
<td></td>
<td>End Address</td>
<td>Flash end</td>
<td>0x7FF(1)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0x3FF(2)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0x1FF(2)</td>
</tr>
<tr>
<td>32 Registers</td>
<td>Size</td>
<td>32 bytes</td>
<td>32 bytes</td>
</tr>
<tr>
<td></td>
<td>Start Address</td>
<td>0x0000</td>
<td>0x0000</td>
</tr>
<tr>
<td></td>
<td>End Address</td>
<td>0x001F</td>
<td>0x001F</td>
</tr>
<tr>
<td>I/O Registers</td>
<td>Size</td>
<td>64 bytes</td>
<td>64 bytes</td>
</tr>
<tr>
<td></td>
<td>Start Address</td>
<td>0x0020</td>
<td>0x0020</td>
</tr>
<tr>
<td></td>
<td>End Address</td>
<td>0x005F</td>
<td>0x005F</td>
</tr>
<tr>
<td>Ext I/O</td>
<td>Size</td>
<td>160 bytes</td>
<td>160 bytes</td>
</tr>
<tr>
<td>Registers</td>
<td>Start Address</td>
<td>0x0060</td>
<td>0x0060</td>
</tr>
<tr>
<td></td>
<td>End Address</td>
<td>0x00FF</td>
<td>0x00FF</td>
</tr>
<tr>
<td>Internal</td>
<td>Size</td>
<td>2.5K bytes</td>
<td>1.25K bytes</td>
</tr>
<tr>
<td>SRAM</td>
<td>Start Address</td>
<td>ISRAM start</td>
<td>0x100</td>
</tr>
<tr>
<td></td>
<td>End Address</td>
<td>ISRAM end</td>
<td>0x0AFF</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0x05FF</td>
</tr>
<tr>
<td>External</td>
<td></td>
<td>Not Present.</td>
<td></td>
</tr>
<tr>
<td>Memory</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EEPROM</td>
<td>Size</td>
<td>1K bytes</td>
<td>512 bytes</td>
</tr>
<tr>
<td></td>
<td>End Address</td>
<td>E2 end</td>
<td>0x03FF</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0x01FF</td>
</tr>
</tbody>
</table>

Notes:  
1. Byte address.  
2. Word (16-bit) address.

### 5.1 In-System Reprogrammable Flash Program Memory

The ATmega16U4/ATmega32U4 contains 16/32K bytes On-chip In-System Reprogrammable Flash memory for program storage. Since all AVR instructions are 16 or 32 bits wide, the Flash is organized as 16K x 16. For software security, the Flash Program memory space is divided into two sections, Boot Program section and Application Program section.

The Flash memory has an endurance of at least 100,000 write/erase cycles. The ATmega16U4/ATmega32U4 Program Counter (PC) is 16 bits wide, thus addressing the 32K program memory locations. The operation of Boot Program section and associated Boot Lock bits for software protection are described in detail in “Memory Programming” on page 346.
“Memory Programming” on page 346 contains a detailed description on Flash data serial downloading using the SPI pins or the JTAG interface.

Constant tables can be allocated within the entire program memory address space (see the LPM – Load Program Memory instruction description and ELPM - Extended Load Program Memory instruction description).

Timing diagrams for instruction fetch and execution are presented in “Instruction Execution Timing” on page 14.

**Figure 5-1.** Program Memory Map

---

5.2 **SRAM Data Memory**

**Figure 5-2** shows how the ATmega16U4/ATmega32U4 SRAM Memory is organized.

The ATmega16U4/ATmega32U4 is a complex microcontroller with more peripheral units than can be supported within the 64 location reserved in the Opcode for the IN and OUT instructions. For the Extended I/O space from $060 - $0FF in SRAM, only the ST/STS/STD and LD/LDS/LDD instructions can be used.

The first 2,816 Data Memory locations address both the Register File, the I/O Memory, Extended I/O Memory, and the internal data SRAM. The first 32 locations address the Register file, the next 64 location the standard I/O Memory, then 160 locations of Extended I/O memory and the next 2,560 locations address the internal data SRAM.

The five different addressing modes for the data memory cover: Direct, Indirect with Displacement, Indirect, Indirect with Pre-decrement, and Indirect with Post-increment. In the Register file, registers R26 to R31 feature the indirect addressing pointer registers.

The direct addressing reaches the entire data space.
The Indirect with Displacement mode reaches 63 address locations from the base address given by the Y- or Z-register.

When using register indirect addressing modes with automatic pre-decrement and post-increment, the address registers X, Y, and Z are decremented or incremented.

The 32 general purpose working registers, 64 I/O registers, and the 1.25/2.5Kbytes of internal data SRAM in the ATmega16U4/ATmega32U4 are all accessible through all these addressing modes. The Register File is described in “General Purpose Register File” on page 12.

**Figure 5-2.** Data Memory Map

<table>
<thead>
<tr>
<th>Data Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>32 Registers</td>
</tr>
<tr>
<td>64 I/O Registers</td>
</tr>
<tr>
<td>160 External I/O Reg.</td>
</tr>
</tbody>
</table>

ISRAM start: $0100

ISRAM end: $05FF / $0AFF

$FFFF

5.2.1 **Data Memory Access Times**

This section describes the general access timing concepts for internal memory access. The internal data SRAM access is performed in two $\text{clk}_{CPU}$ cycles as described in **Figure 5-3**.

**Figure 5-3.** On-chip Data SRAM Access Cycles

The necessary timing considerations for the internal data SRAM access are given in **Figure 5-3**.
5.3 EEPROM Data Memory

The ATmega16U4/ATmega32U4 contains 512Bytes/1K bytes of data EEPROM memory. It is organized as a separate data space, in which single bytes can be read and written. The EEPROM has an endurance of at least 100,000 write/erase cycles. The access between the EEPROM and the CPU is described in the following, specifying the EEPROM Address Registers, the EEPROM Data Register, and the EEPROM Control Register.

For a detailed description of SPI, JTAG and Parallel data downloading to the EEPROM, see page 360, page 365, and page 349 respectively.

5.3.1 EEPROM Read/Write Access

The EEPROM Access Registers are accessible in the I/O space.

The write access time for the EEPROM is given in Table 5-3. A self-timing function, however, lets the user software detect when the next byte can be written. If the user code contains instructions that write the EEPROM, some precautions must be taken. In heavily filtered power supplies, \( V_{CC} \) is likely to rise or fall slowly on power-up/down. This causes the device for some period of time to run at a voltage lower than specified as minimum for the clock frequency used. See “Preventing EEPROM Corruption” on page 25, for details on how to avoid problems in these situations.

In order to prevent unintentional EEPROM writes, a specific write procedure must be followed. Refer to the description of the EEPROM Control Register for details on this.

When the EEPROM is read, the CPU is halted for four clock cycles before the next instruction is executed. When the EEPROM is written, the CPU is halted for two clock cycles before the next instruction is executed.

5.3.2 The EEPROM Address Register – EEARH and EEARL

<table>
<thead>
<tr>
<th>Bit</th>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>EEAR7</td>
<td>EEAR6</td>
<td>EEAR5</td>
<td>EEAR4</td>
<td>EEAR3</td>
<td>EEAR2</td>
<td>EEAR1</td>
<td>EEAR0</td>
</tr>
<tr>
<td>Read/Write</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
</tr>
<tr>
<td>Initial Value</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

- **Bits 15..12 – Res: Reserved Bits**
  These bits are reserved bits in the ATmega16U4/ATmega32U4 and will always read as zero.

- **Bits 11..0 – EEAR8..0: EEPROM Address**
  The EEPROM Address Registers – EEARH and EEARL specify the EEPROM address in the 512Bytes/1K bytes EEPROM space. The EEPROM data bytes are addressed linearly between 0 and E2_END. The initial value of EEAR is undefined. A proper value must be written before the EEPROM may be accessed.

5.3.3 The EEPROM Data Register – EEDR

<table>
<thead>
<tr>
<th>Bit</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>MSB</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Read/Write</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
</tr>
<tr>
<td>Initial Value</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
• Bits 7..0 – EEDR7.0: EEPROM Data
For the EEPROM write operation, the EEDR Register contains the data to be written to the EEPROM in the address given by the EEAR Register. For the EEPROM read operation, the EEDR contains the data read out from the EEPROM at the address given by EEAR.

5.3.4 The EEPROM Control Register – EECR

<table>
<thead>
<tr>
<th>Bit</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read/Write</td>
<td>R</td>
<td>R</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
</tr>
<tr>
<td>Initial Value</td>
<td>0</td>
<td>0</td>
<td>X</td>
<td>X</td>
<td>0</td>
<td>0</td>
<td>X</td>
<td>0</td>
</tr>
</tbody>
</table>

• Bits 7..6 – Res: Reserved Bits
These bits are reserved bits in the ATmega16U4/ATmega32U4 and will always read as zero.

• Bits 5, 4 – EEPM1 and EEPM0: EEPROM Programming Mode Bits
The EEPROM Programming mode bit setting defines which programming action that will be triggered when writing EEPE. It is possible to program data in one atomic operation (erase the old value and program the new value) or to split the Erase and Write operations in two different operations. The Programming times for the different modes are shown in Table 5-2. While EEPE is set, any write to EEPMn will be ignored. During reset, the EEPMn bits will be reset to 0b00 unless the EEPROM is busy programming.

<table>
<thead>
<tr>
<th>Table 5-2. EEPROM Mode Bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>EEPM1</td>
</tr>
<tr>
<td>0</td>
</tr>
<tr>
<td>0</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>1</td>
</tr>
</tbody>
</table>

• Bit 3 – EERIE: EEPROM Ready Interrupt Enable
Writing EERIE to one enables the EEPROM Ready Interrupt if the I bit in SREG is set. Writing EERIE to zero disables the interrupt. The EEPROM Ready interrupt generates a constant interrupt when EEPE is cleared.

• Bit 2 – EEMPE: EEPROM Master Programming Enable
The EEMPE bit determines whether setting EEPE to one causes the EEPROM to be written. When EEMPE is set, setting EEPE within four clock cycles will write data to the EEPROM at the selected address if EEMPE is zero, setting EEPE will have no effect. When EEMPE has been written to one by software, hardware clears the bit to zero after four clock cycles. See the description of the EEPE bit for an EEPROM write procedure.

• Bit 1 – EEPE: EEPROM Programming Enable
The EEPROM Write Enable Signal EEPE is the write strobe to the EEPROM. When address and data are correctly set up, the EEPE bit must be written to one to write the value into the EEPROM. The EEMPE bit must be written to one before a logical one is written to EEPE, other-
wise no EEPROM write takes place. The following procedure should be followed when writing the EEPROM (the order of steps 3 and 4 is not essential):

1. Wait until EEPE becomes zero.
2. Wait until SELFPRTGEN in SPMCSR becomes zero.
3. Write new EEPROM address to EEAR (optional).
4. Write new EEPROM data to EEDR (optional).
5. Write a logical one to the EEMPE bit while writing a zero to EEPE in EECR.
6. Within four clock cycles after setting EEMPE, write a logical one to EEPE.

The EEPROM can not be programmed during a CPU write to the Flash memory. The software must check that the Flash programming is completed before initiating a new EEPROM write. Step 2 is only relevant if the software contains a Boot Loader allowing the CPU to program the Flash. If the Flash is never being updated by the CPU, step 2 can be omitted. See “Memory Programming” on page 346 for details about Boot programming.

**Caution:** An interrupt between step 5 and step 6 will make the write cycle fail, since the EEPROM Master Write Enable will time-out. If an interrupt routine accessing the EEPROM is interrupting another EEPROM access, the EEAR or EEDR Register will be modified, causing the interrupted EEPROM access to fail. It is recommended to have the Global Interrupt Flag cleared during all the steps to avoid these problems.

When the write access time has elapsed, the EEPE bit is cleared by hardware. The user software can poll this bit and wait for a zero before writing the next byte. When EEPE has been set, the CPU is halted for two cycles before the next instruction is executed.

- **Bit 0 – EERE: EEPROM Read Enable**
  The EEPROM Read Enable Signal EERE is the read strobe to the EEPROM. When the correct address is set up in the EEAR Register, the EERE bit must be written to a logic one to trigger the EEPROM read. The EEPROM read access takes one instruction, and the requested data is available immediately. When the EEPROM is read, the CPU is halted for four cycles before the next instruction is executed.

  The user should poll the EEPE bit before starting the read operation. If a write operation is in progress, it is neither possible to read the EEPROM, nor to change the EEAR Register.

  The calibrated Oscillator is used to time the EEPROM accesses. Table 5-3 lists the typical programming time for EEPROM access from the CPU.

**Table 5-3. EEPROM Programming Time**

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Number of Calibrated RC Oscillator Cycles</th>
<th>Typ Programming Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>EEPROM write (from CPU)</td>
<td>26,368</td>
<td>3.3 ms</td>
</tr>
</tbody>
</table>

The following code examples show one assembly and one C function for writing to the EEPROM. The examples assume that interrupts are controlled (e.g. by disabling interrupts globally) so that no interrupts will occur during execution of these functions. The examples also assume that no Flash Boot Loader is present in the software. If such code is present, the EEPROM write function must also wait for any ongoing SPM command to finish.
Assembly Code Example(1)

```
EEPROM_write:
    ; Wait for completion of previous write
    sbic EECR, EPE
    rjmp EEPROM_write
    ; Set up address (r18:r17) in address register
    out EEARH, r18
    out EEARL, r17
    ; Write data (r16) to Data Register
    out EEDR, r16
    ; Write logical one to EEMPE
    sbi EECR, EEMPE
    ; Start eeprom write by setting EPE
    sbi EECR, EPE
    ret
```

C Code Example(1)

```
void EEPROM_write(unsigned int uiAddress, unsigned char ucData)
{
       /* Wait for completion of previous write */
       while(EECR & (1<<EPE))
       {

       /* Set up address and Data Registers */
       EEAR = uiAddress;
       EEDR = ucData;
       /* Write logical one to EEMPE */
       EECR |= (1<<EEMPE);
       /* Start eeprom write by setting EPE */
       EECR |= (1<<EPE);
       }
```

Note:  1. See “Code Examples” on page 8.

The next code examples show assembly and C functions for reading the EEPROM. The examples assume that interrupts are controlled so that no interrupts will occur during execution of these functions.
5.3.5 Preventing EEPROM Corruption

During periods of low $V_{CC}$, the EEPROM data can be corrupted because the supply voltage is too low for the CPU and the EEPROM to operate properly. These issues are the same as for board level systems using EEPROM, and the same design solutions should be applied.

An EEPROM data corruption can be caused by two situations when the voltage is too low. First, a regular write sequence to the EEPROM requires a minimum voltage to operate correctly. Secondly, the CPU itself can execute instructions incorrectly, if the supply voltage is too low.

EEPROM data corruption can easily be avoided by following this design recommendation:

Keep the AVR RESET active (low) during periods of insufficient power supply voltage. This can be done by enabling the internal Brown-out Detector (BOD). If the detection level of the internal BOD does not match the needed detection level, an external low $V_{CC}$ reset Protection circuit can be used. If a reset occurs while a write operation is in progress, the write operation will be completed provided that the power supply voltage is sufficient.
5.4 I/O Memory

The I/O space definition of the ATmega16U4/ATmega32U4 is shown in “Register Summary” on page 408.

All ATmega16U4/ATmega32U4 I/Os and peripherals are placed in the I/O space. All I/O locations may be accessed by the LD/LDS/LDD and ST/STS/STD instructions, transferring data between the 32 general purpose working registers and the I/O space. I/O Registers within the address range 0x00 - 0x1F are directly bit-accessible using the SBI and CBI instructions. In these registers, the value of single bits can be checked by using the SBIS and SBIC instructions. Refer to the instruction set section for more details. When using the I/O specific commands IN and OUT, the I/O addresses 0x00 - 0x3F must be used. When addressing I/O Registers as data space using LD and ST instructions, 0x20 must be added to these addresses. The ATmega16U4/ATmega32U4 is a complex microcontroller with more peripheral units than can be supported within the 64 location reserved in Opcode for the IN and OUT instructions. For the Extended I/O space from 0x60 - 0xFF in SRAM, only the ST/STS/STD and LD/LDS/LDD instructions can be used.

For compatibility with future devices, reserved bits should be written to zero if accessed. Reserved I/O memory addresses should never be written.

Some of the Status Flags are cleared by writing a logical one to them. Note that, unlike most other AVRs, the CBI and SBI instructions will only operate on the specified bit, and can therefore be used on registers containing such Status Flags. The CBI and SBI instructions work with registers 0x00 to 0x1F only.

The I/O and peripherals control registers are explained in later sections.

5.4.1 General Purpose I/O Registers

The ATmega16U4/ATmega32U4 contains three General Purpose I/O Registers. These registers can be used for storing any information, and they are particularly useful for storing global variables and Status Flags. General Purpose I/O Registers within the address range 0x00 - 0x1F are directly bit-accessible using the SBI, CBI, SBIS, and SBIC instructions.

5.4.2 General Purpose I/O Register 2 – GPIOR2

<table>
<thead>
<tr>
<th>Bit</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSB</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LSB</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Read/Write</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
</tr>
<tr>
<td>Initial Value</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

5.4.3 General Purpose I/O Register 1 – GPIOR1

<table>
<thead>
<tr>
<th>Bit</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSB</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LSB</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Read/Write</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
</tr>
<tr>
<td>Initial Value</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

5.4.4 General Purpose I/O Register 0 – GPIOR0

<table>
<thead>
<tr>
<th>Bit</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSB</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LSB</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Read/Write</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
</tr>
<tr>
<td>Initial Value</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
6. System Clock and Clock Options

6.1 Clock Systems and their Distribution

Figure 6-1 presents the principal clock systems in the AVR and their distribution. All of the clocks need not be active at a given time. In order to reduce power consumption, the clocks to modules not being used can be halted by using different sleep modes, as described in “Power Management and Sleep Modes” on page 43. The clock systems are detailed below.

Figure 6-1. Clock Distribution

6.1.1 CPU Clock – clkCPU

The CPU clock is routed to parts of the system concerned with operation of the AVR core. Examples of such modules are the General Purpose Register File, the Status Register and the data memory holding the Stack Pointer. Halting the CPU clock inhibits the core from performing general operations and calculations.

6.1.2 I/O Clock – clkIO

The I/O clock is used by the majority of the I/O modules, like Timer/Counters, SPI, and USART. The I/O clock is also used by the External Interrupt module, but note that some external interrupts are detected by asynchronous logic, allowing such interrupts to be detected even if the I/O clock is halted. Also, TWI address recognition is handled in all sleep modes.

6.1.3 Flash Clock – clkFLASH

The Flash clock controls operation of the Flash interface. The Flash clock is usually active simultaneously with the CPU clock.
6.1.4 ADC Clock – clk\textsubscript{ADC}

The ADC is provided with a dedicated clock domain. This allows halting the CPU and I/O clocks in order to reduce noise generated by digital circuitry. This gives more accurate ADC conversion results.

6.1.5 PLL Prescaler Clock – clk\textsubscript{PllPresc}

The PLL requires a 8 MHz input. A prescaler allows user to use either a 8MHz or a 16MHz source (from a crystal or an external source), using a divider (by 2) if necessary. The output of the prescaler goes into the PLL Input multiplexer, that allows the user to select either the prescaler output of the System Clock Multiplexer, or the Internal 8MHz Calibrated Oscillator.

6.1.6 PLL Output Clock – clk\textsubscript{Pll}

When enabled, the PLL outputs one frequency among numerous choices between 32MHz and 96 MHz. The output frequency is determined by the PLL clock register. The frequency is independent of the power supply voltage. The PLL Output is connected to a postcaler that allows user to generate two different frequencies (clk\textsubscript{USB} and clk\textsubscript{TMR}) from the common PLL signal, each on them resulting of a selected division ratio (/1, /1.5, /2).

6.1.7 High-Speed Timer Clock– clk\textsubscript{TMR}

When enabled, the PLL outputs one frequency among numerous choices between 32MHz and 96 MHz, that goes into the PLL Postcaler. The High Speed Timer frequency input is generated from the PLL Postcaler, that proposes /1, /1.5 and /2 ratios. That can be determined from the PLL clock register. The High Speed Timer maximum frequency input depends on the power supply voltage and reaches its maximum of 64 MHz at 5V.

6.1.8 USB Clock – clk\textsubscript{USB}

The USB hardware module needs for a 48 MHz clock. This clock is generated from the on-chip PLL. The output of the PLL passes through the PLL Postcaler where the frequency can be either divided by 2 or directly connected to the clk\textsubscript{USB} signal.

6.2 Clock Sources

The device has the following clock source options, selectable by Flash Fuse bits as shown below. The clock from the selected source is input to the AVR clock generator, and routed to the appropriate modules.

<table>
<thead>
<tr>
<th>Device Clocking Option</th>
<th>CKSEL[3:0] (or EXCKSEL[3:0])</th>
</tr>
</thead>
<tbody>
<tr>
<td>Low Power Crystal Oscillator</td>
<td>1111 - 1000</td>
</tr>
<tr>
<td>Reserved</td>
<td>0111 - 0110</td>
</tr>
<tr>
<td>Low Frequency Crystal Oscillator</td>
<td>0101 - 0100</td>
</tr>
<tr>
<td>Reserved</td>
<td>0011</td>
</tr>
<tr>
<td>Calibrated Internal RC Oscillator</td>
<td>0010</td>
</tr>
<tr>
<td>External Clock</td>
<td>0000</td>
</tr>
<tr>
<td>Reserved</td>
<td>0001</td>
</tr>
</tbody>
</table>

Note: 1. For all fuses “1” means unprogrammed while “0” means programmed.
6.2.1 Default Clock Source ATmega16U4 and ATmega32U4
The device is shipped with Low Power Crystal Oscillator (8.0MHz-16MHz) enabled and with the fuse CKDIV8 programmed, resulting in 1.0MHz system clock with an 8 MHz crystal. See Table 28-5 on page 348 for an overview of the default Clock Selection Fuse setting.

6.2.2 Default Clock Source ATmega16U4RC and ATmega32U4RC
The device is shipped with Calibrated Internal RC oscillator (8.0MHz) enabled and with the fuse CKDIV8 programmed, resulting in 1.0MHz system clock. See Table 28-5 on page 348 for an overview of the default Clock Selection Fuse setting.

6.2.3 Clock Startup Sequence
Any clock source needs a sufficient V\text{cc} to start oscillating and a minimum number of oscillating cycles before it can be considered stable.

To ensure sufficient V\text{cc}, the device issues an internal reset with a time-out delay (t_{\text{OUT}}) after the device reset is released by all other reset sources. “On-chip Debug System” on page 48 describes the start conditions for the internal reset. The delay (t_{\text{OUT}}) is timed from the Watchdog Oscillator and the number of cycles in the delay is set by the SUTx and CKSELx fuse bits. The selectable delays are shown in Table 6-2. The frequency of the Watchdog Oscillator is voltage dependent as shown in Table 6-2.

<table>
<thead>
<tr>
<th>Typ Time-out (V\text{cc} = 5.0V)</th>
<th>Typ Time-out (V\text{cc} = 3.0V)</th>
<th>Number of Cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 ms</td>
<td>0 ms</td>
<td>0</td>
</tr>
<tr>
<td>4.1 ms</td>
<td>4.3 ms</td>
<td>512</td>
</tr>
<tr>
<td>65 ms</td>
<td>69 ms</td>
<td>8K (6,192)</td>
</tr>
</tbody>
</table>

Main purpose of the delay is to keep the AVR in reset until it is supplied with minimum Vcc. The delay will not monitor the actual voltage and it will be required to select a delay longer than the Vcc rise time. If this is not possible, an internal or external Brown-Out Detection circuit should be used. A BOD circuit will ensure sufficient Vcc before it releases the reset, and the time-out delay can be disabled. Disabling the time-out delay without utilizing a Brown-Out Detection circuit is not recommended.

The oscillator is required to oscillate for a minimum number of cycles before the clock is considered stable. An internal ripple counter monitors the oscillator output clock, and keeps the internal reset active for a given number of clock cycles. The reset is then released and the device will start to execute. The recommended oscillator start-up time is dependent on the clock type, and varies from 6 cycles for an externally applied clock to 32K cycles for a low frequency crystal.

The start-up sequence for the clock includes both the time-out delay and the start-up time when the device starts up from reset. When starting up from Power-save or Power-down mode, Vcc is assumed to be at a sufficient level and only the start-up time is included.

6.3 Low Power Crystal Oscillator
Pins XTAL1 and XTAL2 are input and output, respectively, of an inverting amplifier which can be configured for use as an On-chip Oscillator, as shown in Figure 6-2. Either a quartz crystal or a ceramic resonator may be used.

This Crystal Oscillator is a low power oscillator, with reduced voltage swing on the XTAL2 output. It gives the lowest power consumption, but is not capable of driving other clock inputs.
C1 and C2 should always be equal for both crystals and resonators. The optimal value of the capacitors depends on the crystal or resonator in use, the amount of stray capacitance, and the electromagnetic noise of the environment. Some initial guidelines for choosing capacitors for use with crystals are given in Table 6-3. For ceramic resonators, the capacitor values given by the manufacturer should be used.

**Figure 6-2.** Crystal Oscillator Connections

![Crystal Oscillator Connections Diagram](image)

The Low Power Oscillator can operate in three different modes, each optimized for a specific frequency range. The operating mode is selected by the fuses CKSEL[3..1] as shown in Table 6-3.

**Table 6-3.** Low Power Crystal Oscillator Operating Modes

<table>
<thead>
<tr>
<th>Frequency Range (MHz)</th>
<th>CKSEL3..1</th>
<th>Recommended Range for Capacitors C1 and C2 (pF)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.4 - 0.9</td>
<td>100 (2)</td>
<td>–</td>
</tr>
<tr>
<td>0.9 - 3.0</td>
<td>101</td>
<td>12 - 22</td>
</tr>
<tr>
<td>3.0 - 8.0</td>
<td>110</td>
<td>12 - 22</td>
</tr>
<tr>
<td>8.0 - 16.0</td>
<td>111</td>
<td>12 - 22</td>
</tr>
</tbody>
</table>

Notes: 1. The frequency ranges are preliminary values. Actual values are TBD.
2. This option should not be used with crystals, only with ceramic resonators.
3. If 8 MHz frequency exceeds the specification of the device (depends on VCC), the CKDIV8 Fuse can be programmed in order to divide the internal frequency by 8. It must be ensured that the resulting divided clock meets the frequency specification of the device.

The CKSEL0 Fuse together with the SUT1..0 Fuses select the start-up times as shown in Table 6-4.

**Table 6-4.** Start-up Times for the Low Power Crystal Oscillator Clock Selection

<table>
<thead>
<tr>
<th>Oscillator Source / Power Conditions</th>
<th>Start-up Time from Power-down and Power-save</th>
<th>Additional Delay from Reset (VCC = 5.0V)</th>
<th>CKSEL0</th>
<th>SUT1..0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ceramic resonator, fast rising power</td>
<td>258 CK</td>
<td>14CK + 4.1 ms (1)</td>
<td>0</td>
<td>00</td>
</tr>
<tr>
<td>Ceramic resonator, slowly rising power</td>
<td>258 CK</td>
<td>14CK + 65 ms (1)</td>
<td>0</td>
<td>01</td>
</tr>
<tr>
<td>Ceramic resonator, BOD enabled</td>
<td>1K CK</td>
<td>14CK (2)</td>
<td>0</td>
<td>10</td>
</tr>
<tr>
<td>Ceramic resonator, fast rising power</td>
<td>1K CK</td>
<td>14CK + 4.1 ms (2)</td>
<td>0</td>
<td>11</td>
</tr>
</tbody>
</table>
Table 6-4. Start-up Times for the Low Power Crystal Oscillator Clock Selection (Continued)

<table>
<thead>
<tr>
<th>Oscillator Source / Power Conditions</th>
<th>Start-up Time from Power-down and Power-save</th>
<th>Additional Delay from Reset ($V_{CC} = 5.0V$)</th>
<th>CKSEL0</th>
<th>SUT1..0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ceramic resonator, slowly rising power</td>
<td>1K CK</td>
<td>14CK + 65 ms$^{(2)}$</td>
<td>1</td>
<td>00</td>
</tr>
<tr>
<td>Crystal Oscillator, BOD enabled</td>
<td>16K CK</td>
<td>14CK</td>
<td>1</td>
<td>01</td>
</tr>
<tr>
<td>Crystal Oscillator, fast rising power</td>
<td>16K CK</td>
<td>14CK + 4.1 ms</td>
<td>1</td>
<td>10</td>
</tr>
<tr>
<td>Crystal Oscillator, slowly rising power</td>
<td>16K CK</td>
<td>14CK + 65 ms</td>
<td>1</td>
<td>11</td>
</tr>
</tbody>
</table>

Notes:
1. These options should only be used when not operating close to the maximum frequency of the device, and only if frequency stability at start-up is not important for the application. These options are not suitable for crystals.
2. These options are intended for use with ceramic resonators and will ensure frequency stability at start-up. They can also be used with crystals when not operating close to the maximum frequency of the device, and if frequency stability at start-up is not important for the application.

Table 6-5. Start-up times for the internal calibrated RC Oscillator clock selection

<table>
<thead>
<tr>
<th>Power Conditions</th>
<th>Start-up Time from Power-down and Power-save</th>
<th>Additional Delay from Reset ($V_{CC} = 5.0V$)</th>
<th>SUT1..0</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOD enabled</td>
<td>6 CK</td>
<td>14CK</td>
<td>00</td>
</tr>
<tr>
<td>Fast rising power</td>
<td>6 CK</td>
<td>14CK + 4.1 ms</td>
<td>01</td>
</tr>
<tr>
<td>Slowly rising power</td>
<td>6 CK</td>
<td>14CK + 65 ms$^{(1)}$</td>
<td>10</td>
</tr>
<tr>
<td>Reserved</td>
<td></td>
<td></td>
<td>11</td>
</tr>
</tbody>
</table>

Note: 1. The device is shipped with this option selected.

6.4 Low Frequency Crystal Oscillator

The device can utilize a 32.768 kHz watch crystal as clock source by a dedicated Low Frequency Crystal Oscillator. The crystal should be connected as shown in Figure 6-2. When this Oscillator is selected, start-up times are determined by the SUT Fuses and CKSEL0 as shown in Table 6-6.

Table 6-6. Start-up Times for the Low Frequency Crystal Oscillator Clock Selection

<table>
<thead>
<tr>
<th>Power Conditions</th>
<th>Start-up Time from Power-down and Power-save</th>
<th>Additional Delay from Reset ($V_{CC} = 5.0V$)</th>
<th>CKSEL0</th>
<th>SUT1..0</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOD enabled</td>
<td>1K CK</td>
<td>14CK$^{(1)}$</td>
<td>0</td>
<td>00</td>
</tr>
<tr>
<td>Fast rising power</td>
<td>1K CK</td>
<td>14CK + 4.1 ms$^{(1)}$</td>
<td>0</td>
<td>01</td>
</tr>
<tr>
<td>Slowly rising power</td>
<td>1K CK</td>
<td>14CK + 65 ms$^{(1)}$</td>
<td>0</td>
<td>10</td>
</tr>
<tr>
<td>Reserved</td>
<td></td>
<td></td>
<td>0</td>
<td>11</td>
</tr>
<tr>
<td>BOD enabled</td>
<td>32K CK</td>
<td>14CK$^{(1)}$</td>
<td>1</td>
<td>00</td>
</tr>
<tr>
<td>Fast rising power</td>
<td>32K CK</td>
<td>14CK + 4.1 ms</td>
<td>1</td>
<td>01</td>
</tr>
<tr>
<td>Slowly rising power</td>
<td>32K CK</td>
<td>14CK + 65 ms</td>
<td>1</td>
<td>10</td>
</tr>
<tr>
<td>Reserved</td>
<td></td>
<td></td>
<td>1</td>
<td>11</td>
</tr>
</tbody>
</table>
Note: 1. These options should only be used if frequency stability at start-up is not important for the application.

6.5 Calibrated Internal RC Oscillator

The calibrated internal RC Oscillator by default provides a 8.0 MHz clock. This frequency is nominal value at 3V and 25°C. The device is shipped with the CKDIV8 Fuse programmed. See “System Clock Prescaler” on page 37 for more details. This clock may be selected as the system clock by programming the CKSEL Fuses as shown in Table 6-7. If selected, it will operate with no external components. During reset, hardware loads the calibration byte into the OSCCAL Register and thereby automatically calibrates the RC Oscillator. At 3V and 25°C, this calibration gives a frequency of 8 MHz ± 1%. The oscillator can be calibrated to any frequency in the range 7.3 - 8.1 MHz within ±1% accuracy, by changing the OSCCAL register. When this Oscillator is used as the chip clock, the Watchdog Oscillator will still be used for the Watchdog Timer and for the Reset Time-out. For more information on the pre-programmed calibration value, see the section “Calibration Byte” on page 349.

Table 6-7. Internal Calibrated RC Oscillator Operating Modes(1)(3)

<table>
<thead>
<tr>
<th>Frequency Range(2) (MHz)</th>
<th>CKSEL[3:0]</th>
</tr>
</thead>
<tbody>
<tr>
<td>7.3 - 8.1</td>
<td>0010</td>
</tr>
</tbody>
</table>

Notes:
1. The device is shipped with this option selected.
2. The frequency ranges are preliminary values. Actual values are TBD.
3. If 8 MHz frequency exceeds the specification of the device (depends on VCC), the CKDIV8 Fuse can be programmed in order to divide the internal frequency by 8.

When this Oscillator is selected, start-up times are determined by the SUT Fuses as shown in Table 6-5 on page 31.

Table 6-8. Start-up times for the internal calibrated RC Oscillator clock selection

<table>
<thead>
<tr>
<th>Power Conditions</th>
<th>Start-up Time from Power-down and Power-save</th>
<th>Additional Delay from Reset (VCC = 5.0V)</th>
<th>SUT1..0</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOD enabled</td>
<td>6 CK</td>
<td>14CK</td>
<td>00</td>
</tr>
<tr>
<td>Fast rising power</td>
<td>6 CK</td>
<td>14CK + 4.1 ms</td>
<td>01</td>
</tr>
<tr>
<td>Slowly rising power</td>
<td>6 CK</td>
<td>14CK + 65 ms</td>
<td>10</td>
</tr>
<tr>
<td>reserved</td>
<td></td>
<td></td>
<td>11</td>
</tr>
</tbody>
</table>

6.5.1 Oscillator Calibration Register – OSCCAL

<table>
<thead>
<tr>
<th>Bit</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>CAL7</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
</tr>
<tr>
<td>CAL6</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
</tr>
<tr>
<td>CAL5</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
</tr>
<tr>
<td>CAL4</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
</tr>
<tr>
<td>CAL3</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
</tr>
<tr>
<td>CAL2</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
</tr>
<tr>
<td>CAL1</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
</tr>
<tr>
<td>CAL0</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
</tr>
</tbody>
</table>

OSCCAL

- Bits 7..0 – CAL7..0: Oscillator Calibration Value

The Oscillator Calibration Register is used to trim the Calibrated Internal RC Oscillator to remove process variations from the oscillator frequency. The factory-calibrated value is automatically written to this register during chip reset, giving an oscillator frequency of 8.0 MHz at 25°C. The application software can write this register to change the oscillator frequency. The calibration range is +/- 40% and linear (calibration step ~0.4%). With typical process at 25°C the code should be 127 for 8 MHz. Input value of 0x00 gives the lowest frequency, and 0xFF the highest.
The temperature sensitivity is quite linear but as said previously depends on the process. To determine its slope, the frequency must be measured at two temperatures. The temperature sensor of the ATmega16U4/ATmega32U4 allows such an operation, that is detailed on Section 24.6.1 “Sensor Calibration” on page 300. It is then possible to calibrate the oscillator frequency in function of the temperature measured.

Note that this oscillator is used to time EEPROM and Flash write accesses, and these write times will be affected accordingly. If the EEPROM or Flash are written, do not calibrate to more than 8.8 MHz. Otherwise, the EEPROM or Flash write may fail.

### Oscillator Control Register – RCCTRL

<table>
<thead>
<tr>
<th>Bit</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
<th>Read/Write</th>
<th>Initial Value</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>R</td>
<td>R</td>
<td>R/W</td>
<td>0 0 0 0 0 0 0 0</td>
</tr>
</tbody>
</table>

**Bits 7..1 – Reserved**

Do not set these bits. Bits should be read as ‘0’.

**Bit 0 – RCFREQ: RC Oscillator Frequency Select**

When this bit is cleared (default value), the RC Oscillator output frequency is set to 8 MHz. When the bit is set, the RC output frequency is 1 MHz. Note that the OSCCAL value has the same effect on both 8 MHz and 1 MHz output modes (~0.4% / step).

### External Clock

The device can utilize a external clock source as shown in Figure 6-3. To run the device on an external clock, the CKSEL Fuses must be programmed as shown in Table 6-1.

**Figure 6-3.** External Clock Drive Configuration

![External Clock Drive Configuration](image)

When this clock source is selected, start-up times are determined by the SUT Fuses as shown in Table 6-9.
Table 6-9. Start-up Times for the External Clock Selection

<table>
<thead>
<tr>
<th>Power Conditions</th>
<th>Start-up Time from Power-down and Power-save</th>
<th>Additional Delay from Reset ($V_{cc} = 5.0V$)</th>
<th>SUT1..0</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOD enabled</td>
<td>6 CK</td>
<td>14CK</td>
<td>00</td>
</tr>
<tr>
<td>Fast rising power</td>
<td>6 CK</td>
<td>14CK + 4.1 ms</td>
<td>01</td>
</tr>
<tr>
<td>Slowly rising power</td>
<td>6 CK</td>
<td>14CK + 65 ms</td>
<td>10</td>
</tr>
<tr>
<td>Reserved</td>
<td></td>
<td></td>
<td>11</td>
</tr>
</tbody>
</table>

When applying an external clock, it is required to avoid sudden changes in the applied clock frequency to ensure stable operation of the MCU. A variation in frequency of more than 2% from one clock cycle to the next can lead to unpredictable behavior. If changes of more than 2% is required, ensure that the MCU is kept in Reset during the changes.

Note that the System Clock Prescaler can be used to implement run-time changes of the internal clock frequency while still ensuring stable operation. Refer to “System Clock Prescaler” on page 37 for details.

6.7 Clock Switch

The ATmega16U4/ATmega32U4 product includes a Clock Switch controller, that allows user to switch from one clock source to another one by software, in order to control application power and execution time with more accuracy.

6.7.1 Example of use

The modification may be needed when the device enters in USB Suspend mode. It then switches from External Clock to Calibrated RC Oscillator in order to reduce consumption and wake-up delay. In such a configuration, the External Clock is disabled. The firmware can then use the watchdog timer to be woken-up from power-down in order to check if there is an event on the application. If an event occurs on the application or if the USB controller signals a non-idle state on the USB line (Resume for example), the firmware switches the Clock Multiplexer from the Calibrated RC Oscillator to the External Clock. in order to restart USB operation.

This feature can only be used to switch between Calibrated 8 MHz RC Oscillator, External Clock and Low Power Crystal Oscillator. The Low Frequency Crystal Oscillator must not be used with this feature.
6.8 Clock switch Algorithm

6.8.1 Switch from external clock to RC clock

```c
if (Usb_suspend_detected()) // if (UDINT.SUSPI == 1)
{
    Usb_ack_suspend();      // UDINT.SUSPI = 0;
    Usb_freeze_clock();     // USBCON.FRZCLK = 1;
    Disable_pll();          // PLLCSR.PLL = 0;
    Enable_RC_clock();      // CLKSEL0.RCE = 1;
    while (!RC_clock_ready()); // while (CLKSTA.RCON != 1);
    Select_RC_clock();      // CLKSEL0.CLS = 0;
    Disable_external_clock(); // CLKSEL0.EXE = 0;
}
```
6.8.2 Switch from RC clock to external clock

```c
if (Usb_wake_up_detected())  // if (UDINT.WAKEUPI == 1)
{
    Usb_ack_wake_up();  // UDINT.WAKEUPI = 0;
    Enable_external_clock();  // CKSEL0.EXTE = 1;
    while (!External_clock_ready());  // while (CLKSTA.EXTEN != 1);
    Select_external_clock();  // CLKSEL0.CLKS = 1;
    Enable_pll();  // PLLCSR.PLL = 1;
    Disable_RC_clock();  // CLKSEL0.RE = 0;
    while (!PLL_ready());  // while (PLLCSR.PLL != 1);
    Usb_unfreeze_clock();  // USBCON.FRZCLK = 0;
}
```

6.8.3 CLKSEL0 – Clock Selection Register 0

<table>
<thead>
<tr>
<th>Bit</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
<th>CLKSEL0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read/Write</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>See Bit Description</td>
</tr>
<tr>
<td>Initial Value</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>See Bit Description</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **Bit 7-6 – RCSUT[1:0]: SUT for RC oscillator**
  
  These 2 bits are the SUT value for the RC Oscillator. If the RC oscillator is selected by fuse bits, the SUT fuse are copied into these bits. A firmware change will not have any effect because this additional start-up time is only used after a reset and not after a clock switch.

- **Bit 5-4 – EXSUT[1:0]: SUT for External Clock / Low Power Crystal Oscillator**
  
  These 2 bits are the SUT value for the External Clock / Low Power Crystal Oscillator. If the External Clock / Low Power Crystal Oscillator is selected by fuse bits, the SUT fuses are copied into these bits. The firmware can modify these bits by writing a new value. This value will be used at the next start of the External Clock / Low Power Crystal Oscillator.

- **Bit 3 – RCE: Enable RC Oscillator**
  
  The RCE bit must be written to logic one to enable the RC Oscillator. The RCE bit must be written to logic zero to disable the RC Oscillator.

- **Bit 2 – EXTE: Enable External Clock / Low Power Crystal Oscillator**
  
  The OSCE bit must be written to logic one to enable External Clock / Low Power Crystal Oscillator. The OSCE bit must be written to logic zero to disable the External Clock / Low Power Crystal Oscillator.

- **Bit 0 – CLKS: Clock Selector**
  
  The CLKS bit must be written to logic one to select the External Clock / Low Power Crystal Oscillator as CPU clock. The CLKS bit must be written to logic zero to select the RC Oscillator as CPU clock. After a reset, the CLKS bit is set by hardware if the External Clock / Low Power Crystal Oscillator is selected by the fuse bits configuration. The firmware has to check if the clock is correctly started before selected it.

6.8.4 CLKSEL1 – Clock Selection Register 1

<table>
<thead>
<tr>
<th>Bit</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
<th>CLKSEL1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read/Write</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
</tr>
<tr>
<td>Initial Value</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>
• Bit 7-4 – RCCKSEL[3:0]: CKSEL for RC oscillator
Clock configuration for the RC Oscillator. After a reset, this part of the register is loaded with the 0010b value that corresponds to the RC oscillator. Modifying this value by firmware before switching to RC oscillator is prohibited because the RC clock will not start.

• Bit 3-0 – EXCKSEL[3:0]: CKSEL for External Clock / Low Power Crystal Oscillator
Clock configuration for the External Clock / Low Power Crystal Oscillator. After a reset, if the External Clock / Low Power Crystal Oscillator is selected by fuse bits, this part of the register is loaded with the fuse configuration. Firmware can modify it to change the start-up time after the clock switch.
See “Device Clocking Options Select(1)” on page 28 for EXCKSEL[3:0] configuration. Only Low Power Crystal Oscillator, Calibrated Internal RC Oscillator, and External Clock modes are allowed.

6.8.5 CLKSTA – Clock Status Register

<table>
<thead>
<tr>
<th>Bit</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
<th>RCON</th>
<th>EXTON</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read/Write</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
</tr>
<tr>
<td>Initial Value</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

• Bit 7-2 - Reserved bits
These bits are reserved and will always read as zero.

• Bit 1 – RCON: RC Oscillator On
This bit is set by hardware to one if the RC Oscillator is running.
This bit is set by hardware to zero if the RC Oscillator is stopped.

• Bit 0 – EXTON: External Clock / Low Power Crystal Oscillator On
This bit is set by hardware to one if the External Clock / Low Power Crystal Oscillator is running.
This bit is set by hardware to zero if the External Clock / Low Power Crystal Oscillator is stopped.

6.9 Clock Output Buffer
The device can output the system clock on the CLKO pin. To enable the output, the CKOUT Fuse has to be programmed. This mode is suitable when the chip clock is used to drive other circuits on the system. The clock also will be output during reset, and the normal operation of I/O pin will be overridden when the fuse is programmed. Any clock source, including the internal RC Oscillator, can be selected when the clock is output on CLKO. If the System Clock Prescaler is used, it is the divided system clock that is output.

6.9.1 System Clock Prescaler
The AVR USB has a system clock prescaler, and the system clock can be divided by setting the “CLKPR – Clock Prescaler Register” on page 38. This feature can be used to decrease the system clock frequency and the power consumption when the requirement for processing power is low. This can be used with all clock source options, and it will affect the clock frequency of the CPU and all synchronous peripherals. clkI/O, clkADC, clkCPU, and clkFLASH are divided by a factor as shown in Table 6-10.

When switching between prescaler settings, the System Clock Prescaler ensures that no glitches occurs in the clock system. It also ensures that no intermediate frequency is higher than neither the clock frequency corresponding to the previous setting, nor the clock frequency corresponding to the new setting.
The ripple counter that implements the prescaler runs at the frequency of the undivided clock, which may be faster than the CPU’s clock frequency. Hence, it is not possible to determine the state of the prescaler - even if it were readable, and the exact time it takes to switch from one clock division to the other cannot be exactly predicted. From the time the CLKPS values are written, it takes between T1 + T2 and T1 + 2 * T2 before the new clock frequency is active. In this interval, 2 active clock edges are produced. Here, T1 is the previous clock period, and T2 is the period corresponding to the new prescaler setting.

To avoid unintentional changes of clock frequency, a special write procedure must be followed to change the CLKPS bits:

1. Write the Clock Prescaler Change Enable (CLKPCE) bit to one and all other bits in CLKPR to zero.
2. Within four cycles, write the desired value to CLKPS while writing a zero to CLKPCE. Interrupts must be disabled when changing prescaler setting to make sure the write procedure is not interrupted.

### 6.9.2 CLKPR – Clock Prescaler Register

<table>
<thead>
<tr>
<th>Bit</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
<th>CLKPR</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read/Write</td>
<td>R/W</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td></td>
</tr>
<tr>
<td>Initial Value</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **Bit 7 – CLKPCE: Clock Prescaler Change Enable**

  The CLKPCE bit must be written to logic one to enable change of the CLKPS bits. The CLKPCE bit is only updated when the other bits in CLKPR are simultaneously written to zero. CLKPCE is cleared by hardware four cycles after it is written or when CLKPS bits are written. Rewriting the CLKPCE bit within this time-out period does neither extend the time-out period, nor clear the CLKPCE bit.

- **Bits 3..0 – CLKPS[3..0]: Clock Prescaler Select Bits 3 - 0**

  These bits define the division factor between the selected clock source and the internal system clock. These bits can be written run-time to vary the clock frequency to suit the application requirements. As the divider divides the master clock input to the MCU, the speed of all synchronous peripherals is reduced when a division factor is used. The division factors are given in Table 6-10.

The CKDIV8 Fuse determines the initial value of the CLKPS bits. If CKDIV8 is unprogrammed, the CLKPS bits will be reset to “0000”. If CKDIV8 is programmed, CLKPS bits are reset to “0011”, giving a division factor of 8 at start up. This feature should be used if the selected clock source has a higher frequency than the maximum frequency of the device at the present operating conditions. Note that any value can be written to the CLKPS bits regardless of the CKDIV8 Fuse setting. The Application software must ensure that a sufficient division factor is chosen if
the selected clock source has a higher frequency than the maximum frequency of the device at the present operating conditions. The device is shipped with the CKDIV8 Fuse programmed.

Table 6-10. Clock Prescaler Select

<table>
<thead>
<tr>
<th>CLKPS3</th>
<th>CLKPS2</th>
<th>CLKPS1</th>
<th>CLKPS0</th>
<th>Clock Division Factor</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>8</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>16</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>32</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>64</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>128</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>256</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>Reserved</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>Reserved</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>Reserved</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>Reserved</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>Reserved</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>Reserved</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>Reserved</td>
</tr>
</tbody>
</table>

6.10 PLL

The PLL is used to generate internal high frequency (up to 96MHz) clock for USB interface and/or High Speed Timer module, the PLL input is supplied from an external low-frequency clock (the crystal oscillator or external clock input pin from XTAL1).

6.10.1 Internal PLL

The internal PLL in ATmega16U4/ATmega32U4 generates a clock frequency between 32MHz and 96 MHz from nominally 8MHz input.

The source of the 8MHz PLL input clock is the output of the internal PLL clock prescaler that generates the 8MHz from the clock source multiplexer output (See Section 6.10.2 for PLL interface). The PLL prescaler allows a direct connection (8MHz oscillator) or a divide-by-2 stage for a 16MHz clock input.

The PLL output signal enters the PLL Postcaler stage before being distributed to the USB and High Speed Timer modules. Each of these modules can choose an independent division ratio.
6.10.2 PLL Control and Status Register – PLLCSR

<table>
<thead>
<tr>
<th>Bit</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLLCSR</td>
<td>$29$ (§29)</td>
<td>PINDIV</td>
<td>PLL</td>
<td>PLOCK</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Read/Write</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R/W</td>
<td>R</td>
<td>R</td>
<td>R/W</td>
<td>R</td>
</tr>
<tr>
<td>Initial Value</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

- **Bit 7..5 – Res: Reserved Bits**
  These bits are reserved bits in the ATmega16U4/ATmega32U4 and always read as zero.

- **Bit 4 – PINDIV PLL Input Prescaler (1:1, 1:2)**
  These bits allow to configure the PLL input prescaler to generate the 8MHz input clock for the PLL from either a 8 or 16 MHz input.

  When using a 8 MHz clock source, this bit must be set to 0 before enabling PLL (1:1).

  When using a 16 MHz clock source, this bit must be set to 1 before enabling PLL (1:2).

- **Bit 3..2 – Res: Reserved Bits**
  These bits are reserved bits in the ATmega16U4/ATmega32U4 and always read as zero.

- **Bit 1 – PLLE: PLL Enable**
  When the PLLE is set, the PLL is started. Note that the Calibrated 8 MHz Internal RC oscillator is automatically enabled when the PLLE bit is set and with PINMUX (see PLLFRQ register) is set.

  The PLL must be disabled before entering Power down mode in order to stop Internal RC Oscillator and avoid extra-consumption.

- **Bit 0 – PLOCK: PLL Lock Detector**
  When the PLOCK bit is set, the PLL is locked to the reference clock. After the PLL is enabled, it takes about several ms for the PLL to lock. To clear PLOCK, clear PLLE.

6.10.3 PLL Frequency Control Register – PLLFRQ

<table>
<thead>
<tr>
<th>Bit</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLLFRQ</td>
<td>$32$</td>
<td>PINMUX</td>
<td>PLLUSB</td>
<td>PLLTM1</td>
<td>PLLTM0</td>
<td>PDIV3</td>
<td>PDIV2</td>
<td>PDIV1</td>
</tr>
<tr>
<td>Read/Write</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
</tr>
<tr>
<td>Initial Value</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
• Bit 7 – PINMUX: PLL Input Multiplexer
This bit selects the clock input of the PLL:
  – PINMUX = 0: the PLL input is connected to the PLL Prescaler, that has the Primary System Clock as source
  – PINMUX = 1: the PLL input is directly connected to the Internal Calibrated 8MHz RC Oscillator. This mode allows to work in USB Low Speed mode with no crystal or using a crystal with a value different of 8/16MHz.

• Bit 6 – PLLUSB: PLL Postcaler for USB Peripheral
This bit select the division factor between the PLL output frequency and the USB module input frequency:
  – PLLUSB = 0: no division, direct connection (if PLL Output = 48 MHz)
  – PLLUSB = 1: PLL Output frequency is divided by 2 and sent to USB module (if PLL Output = 96MHz)

• Bit 5..4 – PLLTM1:0: PLL Postcaler for High Speed Timer
These bits codes for the division factor between the PLL Output Frequency and the High Speed Timer input frequency.

Note that the division factor 1.5 will introduce some jitter in the clock, but keeping the error null since the average duty cycle is 50%. See Figure 6-7 for more details.

<table>
<thead>
<tr>
<th>PLLTM1</th>
<th>PLLTM0</th>
<th>PLL Postcaler Factor for High-Speed Timer</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0 (Disconnected)</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1.5</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>

Figure 6-7. PLL Postcaler operation with division factor = 1.5

• Bit 3..0 – PDIV3:0 PLL Lock Frequency
These bits configure the PLL internal VCO clock reference according to the required output frequency value.

<table>
<thead>
<tr>
<th>PDIV3</th>
<th>PDIV2</th>
<th>PDIV1</th>
<th>PDIV0</th>
<th>PLL Output Frequency</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>Not allowed</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>Not allowed</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>Not allowed</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>40 MHz</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>48 MHz</td>
</tr>
<tr>
<td>PDIV3</td>
<td>PDIV2</td>
<td>PDIV1</td>
<td>PDIV0</td>
<td>PLL Output Frequency</td>
</tr>
<tr>
<td>-------</td>
<td>-------</td>
<td>-------</td>
<td>-------</td>
<td>----------------------</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>56 MHz</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>Not allowed</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>72 MHz</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>80 MHz</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>88 MHz</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>96 MHz</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>Not allowed</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>Not allowed</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>Not allowed</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>Not allowed</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>Not allowed</td>
</tr>
</tbody>
</table>

The optimal PLL configuration at 5V is: PLL output frequency = 96 MHz, divided by 1.5 to generate the 64 MHz High Speed Timer clock, and divided by 2 to generate the 48 MHz USB clock.
7. Power Management and Sleep Modes

Sleep modes enable the application to shut down unused modules in the MCU, thereby saving power. The AVR provides various sleep modes allowing the user to tailor the power consumption to the application’s requirements.

To enter any of the five sleep modes, the SE bit in SMCR must be written to logic one and a SLEEP instruction must be executed. The SM2, SM1, and SM0 bits in the SMCR Register select which sleep mode (Idle, ADC Noise Reduction, Power-down, Power-save, or Standby) will be activated by the SLEEP instruction. See Table 7-1 for a summary. If an enabled interrupt occurs while the MCU is in a sleep mode, the MCU wakes up. The MCU is then halted for four cycles in addition to the start-up time, executes the interrupt routine, and resumes execution from the instruction following SLEEP. The contents of the Register File and SRAM are unaltered when the device wakes up from sleep. If a reset occurs during sleep mode, the MCU wakes up and executes from the Reset Vector.

Figure 6-1 on page 27 presents the different clock systems in the ATmega16U4/ATmega32U4, and their distribution. The figure is helpful in selecting an appropriate sleep mode.

7.0.1 Sleep Mode Control Register – SMCR

The Sleep Mode Control Register contains control bits for power management.

<table>
<thead>
<tr>
<th>Bit</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>SM2</td>
<td>SM1</td>
<td>SM0</td>
<td>SE</td>
</tr>
<tr>
<td>Read/Write</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
</tr>
<tr>
<td>Initial Value</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

- • Bits 3, 2, 1 – SM2..0: Sleep Mode Select Bits 2, 1, and 0
  These bits select between the six available sleep modes as shown in Table 7-1.

Table 7-1. Sleep Mode Select

<table>
<thead>
<tr>
<th>SM2</th>
<th>SM1</th>
<th>SM0</th>
<th>Sleep Mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>Idle</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>ADC Noise Reduction</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>Power-down</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>Power-save</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>Reserved</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>Reserved</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>Standby(1)</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>Extended Standby(1)</td>
</tr>
</tbody>
</table>

Note: 1. Standby modes are only recommended for use with external crystals or resonators.

- • Bit 1 – SE: Sleep Enable
  The SE bit must be written to logic one to make the MCU enter the sleep mode when the SLEEP instruction is executed. To avoid the MCU entering the sleep mode unless it is the programmer’s purpose, it is recommended to write the Sleep Enable (SE) bit to one just before the execution of the SLEEP instruction and to clear it immediately after waking up.
7.1 Idle Mode
When the SM2..0 bits are written to 000, the SLEEP instruction makes the MCU enter Idle mode, stopping the CPU but allowing the USB, SPI, USART, Analog Comparator, ADC, 2-wire Serial Interface, Timer/Counters, Watchdog, and the interrupt system to continue operating. This sleep mode basically halts clk_CPU and clk_FLASH, while allowing the other clocks to run.

Idle mode enables the MCU to wake up from external triggered interrupts as well as internal ones like the Timer Overflow and USART Transmit Complete interrupts. If wake-up from the Analog Comparator interrupt is not required, the Analog Comparator can be powered down by setting the ACD bit in the Analog Comparator Control and Status Register – ACSR. This will reduce power consumption in Idle mode. If the ADC is enabled, a conversion starts automatically when this mode is entered.

7.2 ADC Noise Reduction Mode
When the SM2..0 bits are written to 001, the SLEEP instruction makes the MCU enter ADC Noise Reduction mode, stopping the CPU but allowing the ADC, the external interrupts, 2-wire Serial Interface address match and the Watchdog to continue operating (if enabled). This sleep mode basically halts clk/I/O, clkCPU, and clkFLASH, while allowing the other clocks to run (including clkUSB).

This improves the noise environment for the ADC, enabling higher resolution measurements. If the ADC is enabled, a conversion starts automatically when this mode is entered. Apart form the ADC Conversion Complete interrupt, only an External Reset, a Watchdog System Reset, a Watchdog interrupt, a Brown-out Reset, a 2-wire serial interface interrupt, an SPM/EEPROM ready interrupt, an external level interrupt on INT6, an external interrupt on INT3:0 or a pin change interrupt can wake up the MCU from ADC Noise Reduction mode.

7.3 Power-down Mode
When the SM2..0 bits are written to 010, the SLEEP instruction makes the MCU enter Power-down mode. In this mode, the external Oscillator is stopped, while the external interrupts, the 2-wire Serial Interface, and the Watchdog continue operating (if enabled). Only an External Reset, a Watchdog Reset, a Brown-out Reset, 2-wire Serial Interface address match, an external level interrupt on INT6, an external interrupt on INT3:0, a pin change interrupt or an asynchronous USB interrupt sources (VBUSTI, WAKEUPI), can wake up the MCU. This sleep mode basically halts all generated clocks, allowing operation of asynchronous modules only.

Note that if a level triggered interrupt is used for wake-up from Power-down mode, the changed level must be held for some time to wake up the MCU. Refer to “External Interrupts” on page 85 for details.

When waking up from Power-down mode, there is a delay from the wake-up condition occurs until the wake-up becomes effective. This allows the clock to restart and become stable after having been stopped. The wake-up period is defined by the same CKSEL Fuses that define the Reset Time-out period, as described in “Clock Sources” on page 28.

7.4 Power-save Mode
When the SM2..0 bits are written to 011, the SLEEP instruction makes the MCU enter Power-save mode. For compatibility reasons with AT90USB64/128 this mode is still present but since Timer 2 Asynchronous operation is not present here, this mode is identical to Power-down.
7.5 Standby Mode

When the SM2..0 bits are 110 and an external crystal/resonator clock option is selected, the SLEEP instruction makes the MCU enter Standby mode. This mode is identical to Power-down with the exception that the Oscillator is kept running. From Standby mode, the device wakes up in six clock cycles.

7.6 Extended Standby Mode

When the SM2..0 bits are 111 and an external crystal/resonator clock option is selected, the SLEEP instruction makes the MCU enter Extended Standby mode. For compatibility reasons with AT90USB64/128 this mode is still present but since Timer 2 Asynchronous operation is not present here, this mode is identical to Standby-mode.

Table 7-2. Active Clock Domains and Wake-up Sources in the Different Sleep Modes.

<table>
<thead>
<tr>
<th>Sleep Mode</th>
<th>Active Clock Domains</th>
<th>Oscillators</th>
<th>Wake-up Sources</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>clkCPU</td>
<td>clkINT3x</td>
<td>clkBO</td>
</tr>
<tr>
<td>Idle</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>ADCNRM</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Power-down</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Power-save</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Standby(1)</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Extended Standby</td>
<td>X</td>
<td>X</td>
<td>(2)</td>
</tr>
</tbody>
</table>

Notes:
1. Only recommended with external crystal or resonator selected as clock source.
2. For INT6, only level interrupt.
3. Asynchronous USB interrupts are VBUSSTI and WAKEUPI.

7.7 Power Reduction Register

The Power Reduction Register, PRR, provides a method to stop the clock to individual peripherals to reduce power consumption. The current state of the peripheral is frozen and the I/O registers can not be read or written. Resources used by the peripheral when stopping the clock will remain occupied, hence the peripheral should in most cases be disabled before stopping the clock. Waking up a module, which is done by clearing the bit in PRR, puts the module in the same state as before shutdown.

Module shutdown can be used in Idle mode and Active mode to significantly reduce the overall power consumption. In all other sleep modes, the clock is already stopped.
7.7.1  Power Reduction Register 0 - PRR0

<table>
<thead>
<tr>
<th>Bit</th>
<th>PRTWI</th>
<th>PRTIM2</th>
<th>PRTIM0</th>
<th>-</th>
<th>PRTIM1</th>
<th>PRSPI</th>
<th>-</th>
<th>PRADC</th>
<th>PRR0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read/Write</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R</td>
<td>R/W</td>
<td>R/W</td>
<td>R</td>
<td>R/W</td>
<td>R/W</td>
</tr>
<tr>
<td>Initial Value</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

- **Bit 7 - PRTWI: Power Reduction TWI**
  Writing a logic one to this bit shuts down the TWI by stopping the clock to the module. When waking up the TWI again, the TWI should be re-initialized to ensure proper operation.

- **Bit 6 - Res: Reserved bit**
  This bit is reserved and will always read as zero.

- **Bit 5 - PRTIM0: Power Reduction Timer/Counter0**
  Writing a logic one to this bit shuts down the Timer/Counter0 module. When the Timer/Counter0 is enabled, operation will continue like before the shutdown.

- **Bit 4 - Res: Reserved bit**
  This bit is reserved and will always read as zero.

- **Bit 3 - PRTIM1: Power Reduction Timer/Counter1**
  Writing a logic one to this bit shuts down the Timer/Counter1 module. When the Timer/Counter1 is enabled, operation will continue like before the shutdown.

- **Bit 2 - PRSPI: Power Reduction Serial Peripheral Interface**
  Writing a logic one to this bit shuts down the Serial Peripheral Interface by stopping the clock to the module. When waking up the SPI again, the SPI should be re-initialized to ensure proper operation.

- **Bit 1 - Res: Reserved bit**
  These bits are reserved and will always read as zero.

- **Bit 0 - PRADC: Power Reduction ADC**
  Writing a logic one to this bit shuts down the ADC. The ADC must be disabled before shut down. The analog comparator cannot use the ADC input MUX when the ADC is shut down.

7.7.2  Power Reduction Register 1 - PRR1

<table>
<thead>
<tr>
<th>Bit</th>
<th>PRUSB</th>
<th>-</th>
<th>-</th>
<th>PRTIM4</th>
<th>PRTIM3</th>
<th>-</th>
<th>-</th>
<th>PRUSART1</th>
<th>PRR1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read/Write</td>
<td>R/W</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R/W</td>
<td>R</td>
<td>R</td>
<td>R/W</td>
<td>R/W</td>
</tr>
<tr>
<td>Initial Value</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

- **Bit 7 - PRUSB: Power Reduction USB**
  Writing a logic one to this bit shuts down the USB by stopping the clock to the module. When waking up the USB again, the USB should be re-initialized to ensure proper operation.

- **Bit 6..5 - Res: Reserved bits**
  These bits are reserved and will always read as zero.
• **Bit 4 - PRTIM4: Power Reduction Timer/Counter4**
  Writing a logic one to this bit shuts down the Timer/Counter4 module. When the Timer/Counter4 is enabled, operation will continue like before the shutdown.

• **Bit 3 - PRTIM3: Power Reduction Timer/Counter3**
  Writing a logic one to this bit shuts down the Timer/Counter3 module. When the Timer/Counter3 is enabled, operation will continue like before the shutdown.

• **Bit 2.1 - Res: Reserved bits**
  These bits are reserved and will always read as zero.

• **Bit 0 - PRUSART1: Power Reduction USART1**
  Writing a logic one to this bit shuts down the USART1 by stopping the clock to the module. When waking up the USART1 again, the USART1 should be re-initialized to ensure proper operation.

### 7.8 Minimizing Power Consumption

There are several issues to consider when trying to minimize the power consumption in an AVR controlled system. In general, sleep modes should be used as much as possible, and the sleep mode should be selected so that as few as possible of the device’s functions are operating. All functions not needed should be disabled. In particular, the following modules may need special consideration when trying to achieve the lowest possible power consumption.

#### 7.8.1 Analog to Digital Converter

If enabled, the ADC will be enabled in all sleep modes. To save power, the ADC should be disabled before entering any sleep mode. When the ADC is turned off and on again, the next conversion will be an extended conversion. Refer to “Analog to Digital Converter - ADC” on page 292 for details on ADC operation.

#### 7.8.2 Analog Comparator

When entering Idle mode, the Analog Comparator should be disabled if not used. When entering ADC Noise Reduction mode, the Analog Comparator should be disabled. In other sleep modes, the Analog Comparator is automatically disabled. However, if the Analog Comparator is set up to use the Internal Voltage Reference as input, the Analog Comparator should be disabled in all sleep modes. Otherwise, the Internal Voltage Reference will be enabled, independent of sleep mode. Refer to “Analog Comparator” on page 289 for details on how to configure the Analog Comparator.

#### 7.8.3 Brown-out Detector

If the Brown-out Detector is not needed by the application, this module should be turned off. If the Brown-out Detector is enabled by the BODLEVEL Fuses, it will be enabled in all sleep modes, and hence, always consume power. In the deeper sleep modes, this will contribute significantly to the total current consumption. Refer to “Brown-out Detection” on page 52 for details on how to configure the Brown-out Detector.

#### 7.8.4 Internal Voltage Reference

The Internal Voltage Reference will be enabled when needed by the Brown-out Detection, the Analog Comparator or the ADC. If these modules are disabled as described in the sections above, the internal voltage reference will be disabled and it will not be consuming power. When
turned on again, the user must allow the reference to start up before the output is used. If the reference is kept on in sleep mode, the output can be used immediately. Refer to “Internal Voltage Reference” on page 54 for details on the start-up time.

7.8.5 Watchdog Timer

If the Watchdog Timer is not needed in the application, the module should be turned off. If the Watchdog Timer is enabled, it will be enabled in all sleep modes, and hence, always consume power. In the deeper sleep modes, this will contribute significantly to the total current consumption. Refer to “Interrupts” on page 61 for details on how to configure the Watchdog Timer.

7.8.6 Port Pins

When entering a sleep mode, all port pins should be configured to use minimum power. The most important is then to ensure that no pins drive resistive loads. In sleep modes where both the I/O clock (clkI/O) and the ADC clock (clkADC) are stopped, the input buffers of the device will be disabled. This ensures that no power is consumed by the input logic when not needed. In some cases, the input logic is needed for detecting wake-up conditions, and it will then be enabled. Refer to the section “Digital Input Enable and Sleep Modes” on page 69 for details on which pins are enabled. If the input buffer is enabled and the input signal is left floating or have an analog signal level close to VCC/2, the input buffer will use excessive power.

For analog input pins, the digital input buffer should be disabled at all times. An analog signal level close to VCC/2 on an input pin can cause significant current even in active mode. Digital input buffers can be disabled by writing to the Digital Input Disable Registers (IDR1 and IDR0). Refer to “Digital Input Disable Register 1 – IDR1” on page 291 and “Digital Input Disable Register 1 – IDR1” on page 291 for details.

7.8.7 On-chip Debug System

If the On-chip debug system is enabled by the OCDEN Fuse and the chip enters sleep mode, the main clock source is enabled, and hence, always consumes power. In the deeper sleep modes, this will contribute significantly to the total current consumption.

There are three alternative ways to disable the OCD system:

- Disable the OCDEN Fuse.
- Disable the JTAGEN Fuse.
- Write one to the JTD bit in MCUCR.
8. System Control and Reset

8.0.1 Resetting the AVR

During reset, all I/O Registers are set to their initial values, and the program starts execution from the Reset Vector. The instruction placed at the Reset Vector must be a JMP – Absolute Jump – instruction to the reset handling routine. If the program never enables an interrupt source, the Interrupt Vectors are not used, and regular program code can be placed at these locations. This is also the case if the Reset Vector is in the Application section while the Interrupt Vectors are in the Boot section or vice versa. The circuit diagram in Figure 8-1 shows the reset logic. Table 8-1 defines the electrical parameters of the reset circuitry.

The I/O ports of the AVR are immediately reset to their initial state when a reset source goes active. This does not require any clock source to be running.

After all reset sources have gone inactive, a delay counter is invoked, stretching the internal reset. This allows the power to reach a stable level before normal operation starts. The time-out period of the delay counter is defined by the user through the SUT and CKSEL Fuses. The different selections for the delay period are presented in “Clock Sources” on page 28.

8.0.2 Reset Sources

The ATmega16U4/ATmega32U4 has five sources of reset:

- Power-on Reset. The MCU is reset when the supply voltage is below the Power-on Reset threshold \( V_{\text{POT}} \).
- External Reset. The MCU is reset when a low level is present on the \text{RESET} pin for longer than the minimum pulse length.
- Watchdog Reset. The MCU is reset when the Watchdog Timer period expires and the Watchdog is enabled.
- Brown-out Reset. The MCU is reset when the supply voltage \( V_{\text{CC}} \) is below the Brown-out Reset threshold \( V_{\text{BOT}} \) and the Brown-out Detector is enabled.
- JTAG AVR Reset. The MCU is reset as long as there is a logic one in the Reset Register, one of the scan chains of the JTAG system. Refer to the section “IEEE 1149.1 (JTAG) Boundary-scan” on page 320 for details.
- USB End of Reset. The MCU is reset (excluding the USB controller that remains enabled and attached) on the detection of a USB End of Reset condition on the bus, if this feature is enabled by the user.
Table 8-1. Reset Characteristics

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter</th>
<th>Condition</th>
<th>Min</th>
<th>Typ</th>
<th>Max</th>
<th>Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>( V_{\text{POT}} )</td>
<td>Power-on Reset Threshold Voltage (rising)</td>
<td></td>
<td>1.4</td>
<td>2.3</td>
<td>V</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Power-on Reset Threshold Voltage (falling)</td>
<td></td>
<td>1.3</td>
<td>2.3</td>
<td>V</td>
<td></td>
</tr>
<tr>
<td>( V_{\text{POR}} )</td>
<td>( V_{\text{CC}} ) Start Voltage to ensure internal Power-on Reset signal</td>
<td></td>
<td>-0.1</td>
<td></td>
<td>+0.1</td>
<td>V</td>
</tr>
<tr>
<td>( V_{\text{CCRR}} )</td>
<td>VCC Rise Rate to ensure internal Power-on Reset signal</td>
<td></td>
<td>0.3</td>
<td></td>
<td></td>
<td>V/ms</td>
</tr>
<tr>
<td>( V_{\text{RST}} )</td>
<td>RESET Pin Threshold Voltage</td>
<td></td>
<td>0.2</td>
<td>( V_{\text{CC}} )</td>
<td>0.85</td>
<td>V</td>
</tr>
<tr>
<td>( t_{\text{RST}} )</td>
<td>Minimum pulse width on RESET Pin</td>
<td></td>
<td>5V, 25°C</td>
<td></td>
<td>400</td>
<td>ns</td>
</tr>
</tbody>
</table>

Notes: 1. The Power-on Reset will not work unless the supply voltage has been below \( V_{\text{POT}} \) (falling)

8.0.3 Power-on Reset

A Power-on Reset (POR) pulse is generated by an On-chip detection circuit. The detection level is defined in Table 8-1. The POR is activated whenever \( V_{\text{CC}} \) is below the detection level. The POR circuit can be used to trigger the start-up Reset, as well as to detect a failure in supply voltage.

A Power-on Reset (POR) circuit ensures that the device is reset from Power-on. Reaching the Power-on Reset threshold voltage invokes the delay counter, which determines how long the
device is kept in RESET after $V_{CC}$ rise. The RESET signal is activated again, without any delay, when $V_{CC}$ decreases below the detection level.

**Figure 8-2.** MCU Start-up, RESET Tied to $V_{CC}$

![Diagram](image)

**Figure 8-3.** MCU Start-up, RESET Extended Externally

![Diagram](image)

### 8.0.4 External Reset

An External Reset is generated by a low level on the RESET pin. Reset pulses longer than the minimum pulse width (see Table 8-1) will generate a reset, even if the clock is not running. Shorter pulses are not guaranteed to generate a reset. When the applied signal reaches the Reset Threshold Voltage – $V_{RST}$ – on its positive edge, the delay counter starts the MCU after the Time-out period – $t_{OUT}$ – has expired.

**Figure 8-4.** External Reset During Operation

![Diagram](image)
8.0.5 Brown-out Detection

ATmega16U4/ATmega32U4 has an On-chip Brown-out Detection (BOD) circuit for monitoring the \(V_{CC}\) level during operation by comparing it to a fixed trigger level. The trigger level for the BOD can be selected by the BODELEVEL Fuses. The trigger level has a hysteresis to ensure spike free Brown-out Detection. The hysteresis on the detection level should be interpreted as \(V_{BOT+} = V_{BOT} + V_{HYST}/2\) and \(V_{BOT-} = V_{BOT} - V_{HYST}/2\).

### Table 8-2. BODELEVEL Fuse Coding

<table>
<thead>
<tr>
<th>BODELEVEL 2..0 Fuses</th>
<th>Min (V_{BOT})</th>
<th>Typ (V_{BOT})</th>
<th>Max (V_{BOT})</th>
<th>Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>111</td>
<td></td>
<td></td>
<td></td>
<td>BOD Disabled</td>
</tr>
<tr>
<td>110</td>
<td>1.8</td>
<td>2.0</td>
<td>2.2</td>
<td>V</td>
</tr>
<tr>
<td>101</td>
<td>2.0</td>
<td>2.2</td>
<td>2.4</td>
<td>V</td>
</tr>
<tr>
<td>100</td>
<td>2.2</td>
<td>2.4</td>
<td>2.6</td>
<td>V</td>
</tr>
<tr>
<td>011</td>
<td>2.4</td>
<td>2.6</td>
<td>2.8</td>
<td>V</td>
</tr>
<tr>
<td>010</td>
<td>3.2</td>
<td>3.4</td>
<td>3.6</td>
<td>V</td>
</tr>
<tr>
<td>001</td>
<td>3.3</td>
<td>3.5</td>
<td>3.7</td>
<td>V</td>
</tr>
<tr>
<td>000</td>
<td>4.0</td>
<td>4.3</td>
<td>4.5</td>
<td>V</td>
</tr>
</tbody>
</table>

### Table 8-3. Brown-out Characteristics

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter</th>
<th>Min</th>
<th>Typ</th>
<th>Max</th>
<th>Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>(V_{HYST})</td>
<td>Brown-out Detector Hysteresis</td>
<td>50</td>
<td></td>
<td></td>
<td>mV</td>
</tr>
<tr>
<td>(t_{BOD})</td>
<td>Min Pulse Width on Brown-out Reset</td>
<td></td>
<td></td>
<td></td>
<td>ns</td>
</tr>
</tbody>
</table>

When the BOD is enabled, and \(V_{CC}\) decreases to a value below the trigger level (\(V_{BOT}\), in Figure 8-5), the Brown-out Reset is immediately activated. When \(V_{CC}\) increases above the trigger level (\(V_{BOT+}\), in Figure 8-5), the delay counter starts the MCU after the Time-out period \(t_{OUT}\) has expired.

The BOD circuit will only detect a drop in \(V_{CC}\) if the voltage stays below the trigger level for longer than \(t_{BOD}\) given in Table 8-1.

### Figure 8-5. Brown-out Reset During Operation

![Brown-out Reset During Operation](image-url)
8.0.6 Watchdog Reset

When the Watchdog times out, it will generate a short reset pulse of one CK cycle duration. On the falling edge of this pulse, the delay timer starts counting the Time-out period \( t_{\text{OUT}} \). Refer to page 55 for details on operation of the Watchdog Timer.

**Figure 8-6.** Watchdog Reset During Operation

8.0.7 USB Reset

When the USB controller is enabled and configured with the USB Reset CPU feature enabled and if a valid USB Reset signalling is detected on the bus, the CPU core is reset but the USB controller remains enabled and attached. This feature may be used to enhance device reliability.

**Figure 8-7.** USB Reset During Operation

8.0.8 MCU Status Register – MCUSR

The MCU Status Register provides information on which reset source caused an MCU reset.

<table>
<thead>
<tr>
<th>Bit</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
<th>MCUSR</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read/Write</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>MCUSR</td>
</tr>
<tr>
<td>Initial Value</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **Bit 7..6 - Reserved**
  These bits are reserved and should be read as 0. Do not set these bits.
• **Bit 5 – USBRF: USB Reset Flag**
  This bit is set if a reset is being caused by a logic one in the JTAG Reset Register selected by the JTAG instruction AVR_RESET. This bit is reset by a Power-on Reset, or by writing a logic zero to the flag.

• **Bit 4 – JTRF: JTAG Reset Flag**
  This bit is set if a reset is being caused by a logic one in the JTAG Reset Register selected by the JTAG instruction AVR_RESET. This bit is reset by a Power-on Reset, or by writing a logic zero to the flag.

• **Bit 3 – WDRF: Watchdog Reset Flag**
  This bit is set if a Watchdog Reset occurs. The bit is reset by a Power-on Reset, or by writing a logic zero to the flag.

• **Bit 2 – BORF: Brown-out Reset Flag**
  This bit is set if a Brown-out Reset occurs. The bit is reset by a Power-on Reset, or by writing a logic zero to the flag.

• **Bit 1 – EXTRF: External Reset Flag**
  This bit is set if an External Reset occurs. The bit is reset by a Power-on Reset, or by writing a logic zero to the flag.

• **Bit 0 – PORF: Power-on Reset Flag**
  This bit is set if a Power-on Reset occurs. The bit is reset only by writing a logic zero to the flag.

To make use of the Reset Flags to identify a reset condition, the user should read and then Reset the MCUSR as early as possible in the program. If the register is cleared before another reset occurs, the source of the reset can be found by examining the Reset Flags.

### 8.1 Internal Voltage Reference

ATmega16U4/ATmega32U4 features an internal bandgap reference. This reference is used for Brown-out Detection, and it can be used as an input to the Analog Comparator or the ADC.

#### 8.1.1 Voltage Reference Enable Signals and Start-up Time

The voltage reference has a start-up time that may influence the way it should be used. The start-up time is given in Table 8-4. To save power, the reference is not always turned on. The reference is on during the following situations:

1. When the BOD is enabled (by programming the BODLEVEL [2..0] Fuse).
2. When the bandgap reference is connected to the Analog Comparator (by setting the ACBG bit in ACSR).
3. When the ADC is enabled.

Thus, when the BOD is not enabled, after setting the ACBG bit or enabling the ADC, the user must always allow the reference to start up before the output from the Analog Comparator or
ADC is used. To reduce power consumption in Power-down mode, the user can avoid the three conditions above to ensure that the reference is turned off before entering Power-down mode.

Table 8-4. Internal Voltage Reference Characteristics(1)

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter</th>
<th>Condition</th>
<th>Min</th>
<th>Typ</th>
<th>Max</th>
<th>Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>VBG</td>
<td>Bandgap reference voltage</td>
<td>TBD</td>
<td>TBD</td>
<td>1.1</td>
<td>TBD</td>
<td>V</td>
</tr>
<tr>
<td>IBG</td>
<td>Bandgap reference start-up time</td>
<td>TBD</td>
<td>40</td>
<td>70</td>
<td>μs</td>
<td></td>
</tr>
<tr>
<td>IBG</td>
<td>Bandgap reference current</td>
<td>TBD</td>
<td>10</td>
<td>TBD</td>
<td>μA</td>
<td></td>
</tr>
</tbody>
</table>

Note: 1. Values are guidelines only. Actual values are TBD.

8.2 Watchdog Timer

ATmega16U4/ATmega32U4 has an Enhanced Watchdog Timer (WDT). The main features are:

- Clocked from separate On-chip Oscillator
- 3 Operating modes
  - Interrupt
  - System Reset
  - Interrupt and System Reset
- Selectable Time-out period from 16ms to 8s
- Possible Hardware fuse Watchdog always on (WDTON) for fail-safe mode

Figure 8-8. Watchdog Timer

The Watchdog Timer (WDT) is a timer counting cycles of a separate on-chip 128 kHz oscillator. The WDT gives an interrupt or a system reset when the counter reaches a given time-out value. In normal operation mode, it is required that the system uses the WDR - Watchdog Timer Reset instruction to restart the counter before the time-out value is reached. If the system doesn't restart the counter, an interrupt or system reset will be issued.

In Interrupt mode, the WDT gives an interrupt when the timer expires. This interrupt can be used to wake the device from sleep-modes, and also as a general system timer. One example is to limit the maximum time allowed for certain operations, giving an interrupt when the operation has run longer than expected. In System Reset mode, the WDT gives a reset when the timer
expires. This is typically used to prevent system hang-up in case of runaway code. The third mode, Interrupt and System Reset mode, combines the other two modes by first giving an interrupt and then switch to System Reset mode. This mode will for instance allow a safe shutdown by saving critical parameters before a system reset.

The Watchdog always on (WDTON) fuse, if programmed, will force the Watchdog Timer to System Reset mode. With the fuse programmed the System Reset mode bit (WDE) and Interrupt mode bit (WDIE) are locked to 1 and 0 respectively. To further ensure program security, alterations to the Watchdog set-up must follow timed sequences. The sequence for clearing WDE and changing time-out configuration is as follows:

1. In the same operation, write a logic one to the Watchdog change enable bit (WDCE) and WDE. A logic one must be written to WDE regardless of the previous value of the WDE bit.
2. Within the next four clock cycles, write the WDE and Watchdog prescaler bits (WDP) as desired, but with the WDCE bit cleared. This must be done in one operation.

The following code example shows one assembly and one C function for turning off the Watchdog Timer. The example assumes that interrupts are controlled (e.g. by disabling interrupts globally) so that no interrupts will occur during the execution of these functions.
Assembly Code Example(1)

```
WDT_off:
    ; Turn off global interrupt
    cli
    ; Reset Watchdog Timer
    wdr
    ; Clear WDRF in MCUSR
    in  r16, MCUSR
    andi r16, (0xff & (0<<WDRF))
    out MCUSR, r16
    ; Write logical one to WDCE and WDE
    ; Keep old prescaler setting to prevent unintentional time-out
    in  r16, WDTCSR
    ori r16, (1<<WDCE) | (1<<WDE)
    out WDTCSR, r16
    ; Turn off WDT
    ldi r16, (0<<WDE)
    out WDTCSR, r16
    ; Turn on global interrupt
    sei
    ret
```

C Code Example(1)

```
void WDT_off(void)
{
    __disable_interrupt();
    __no_integral();
    /* Clear WDRF in MCUSR */
    MCUSR &= ~(1<<WDRF);
    /* Write logical one to WDCE and WDE */
    /* Keep old prescaler setting to prevent unintentional time-out */
    WDTCSR |= (1<<WDCE) | (1<<WDE);
    /* Turn off WDT */
    WDTCSR = 0x00;
    __enable_interrupt();
}
```

Note: 1. The example code assumes that the part specific header file is included.

Note: If the Watchdog is accidentally enabled, for example by a runaway pointer or brown-out condition, the device will be reset and the Watchdog Timer will stay enabled. If the code is not set up to handle the Watchdog, this might lead to an eternal loop of time-out resets. To avoid this situation, the application software should always clear the Watchdog System Reset Flag (WDRF) and the WDE control bit in the initialization routine, even if the Watchdog is not in use.
The following code example shows one assembly and one C function for changing the time-out value of the Watchdog Timer.

**Assembly Code Example**(1)

```assembly
WDT_Prescaler_Change:
    ; Turn off global interrupt
    cli
    ; Reset Watchdog Timer
    wdr
    ; Start timed sequence
    in  r16, WDTCSR
    ori r16, (1<<WDE) | (1<<WDP)
    out WDTCSR, r16
    ; -- Got four cycles to set the new values from here -
    ; Set new prescaler(time-out) value = 64K cycles (~0.5 s)
    ldi r16, (1<<WDE) | (1<<WDP2) | (1<<WDP0)
    out WDTCSR, r16
    ; -- Finished setting new values, used 2 cycles -
    ; Turn on global interrupt
    sei
    ret
```

**C Code Example**(1)

```c
void WDT_Prescaler_Change(void)
{
    __disable_interrupt();
    __watchdog_reset();
    /* Start timed sequence */
    WDTCSR |= (1<<WDE) | (1<<WDE);
    /* Set new prescaler(time-out) value = 64K cycles (~0.5 s) */
    WDTCSR = (1<<WDE) | (1<<WDP2) | (1<<WDP0);
    __enable_interrupt();
}
```

Note: 1. The example code assumes that the part specific header file is included.

Note: The Watchdog Timer should be reset before any change of the WDP bits, since a change in the WDP bits can result in a time-out when switching to a shorter time-out period.

### 8.2.1 Watchdog Timer Control Register - WDTCSR

<table>
<thead>
<tr>
<th>Bit</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
<th>Read/Write</th>
<th>Initial Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>WDTCSR</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>0 0 0 0 X 0 0 0</td>
<td>0 0 0 0 0 0 0 0</td>
</tr>
</tbody>
</table>

- **Bit 7 - WDIF: Watchdog Interrupt Flag**
  
  This bit is set when a time-out occurs in the Watchdog Timer and the Watchdog Timer is configured for interrupt. WDIF is cleared by hardware when executing the corresponding interrupt
handling vector. Alternatively, WDIF is cleared by writing a logic one to the flag. When the I-bit in SREG and WDIE are set, the Watchdog Time-out Interrupt is executed.

- **Bit 6 - WDIE: Watchdog Interrupt Enable**
  When this bit is written to one and the I-bit in the Status Register is set, the Watchdog Interrupt is enabled. If WDE is cleared in combination with this setting, the Watchdog Timer is in Interrupt Mode, and the corresponding interrupt is executed if time-out in the Watchdog Timer occurs.

  If WDE is set, the Watchdog Timer is in Interrupt and System Reset Mode. The first time-out in the Watchdog Timer will set WDIF. Executing the corresponding interrupt vector will clear WDIE and WDIF automatically by hardware (the Watchdog goes to System Reset Mode). This is useful for keeping the Watchdog Timer security while using the interrupt. To stay in Interrupt and System Reset Mode, WDIE must be set after each interrupt. This should however not be done within the interrupt service routine itself, as this might compromise the safety-function of the Watchdog System Reset mode. If the interrupt is not executed before the next time-out, a System Reset will be applied.

  **Table 8-5. Watchdog Timer Configuration**

<table>
<thead>
<tr>
<th>WDTON</th>
<th>WDE</th>
<th>WDIE</th>
<th>Mode</th>
<th>Action on Time-out</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>Stopped</td>
<td>None</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>Interrupt Mode</td>
<td>Interrupt</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>System Reset Mode</td>
<td>Reset</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>Interrupt and System</td>
<td>Interrupt, then go to</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Reset Mode</td>
<td>System Reset Mode</td>
</tr>
<tr>
<td>1</td>
<td>x</td>
<td>x</td>
<td>System Reset Mode</td>
<td>Reset</td>
</tr>
</tbody>
</table>

- **Bit 4 - WDCE: Watchdog Change Enable**
  This bit is used in timed sequences for changing WDE and prescaler bits. To clear the WDE bit, and/or change the prescaler bits, WDCE must be set.

  Once written to one, hardware will clear WDCE after four clock cycles.

- **Bit 3 - WDE: Watchdog System Reset Enable**
  WDE is overridden by WDRF in MCUSR. This means that WDE is always set when WDRF is set. To clear WDE, WDRF must be cleared first. This feature ensures multiple resets during conditions causing failure, and a safe start-up after the failure.

- **Bit 5, 2..0 - WDP3..0: Watchdog Timer Prescaler 3, 2, 1 and 0**
  The WDP3..0 bits determine the Watchdog Timer prescaling when the Watchdog Timer is running. The different prescaling values and their corresponding time-out periods are shown in *Table 8-6 on page 60.*
### Table 8-6. Watchdog Timer Prescale Select

<table>
<thead>
<tr>
<th>WDP3</th>
<th>WDP2</th>
<th>WDP1</th>
<th>WDP0</th>
<th>Number of WDT Oscillator Cycles</th>
<th>Typical Time-out at $V_{CC} = 5.0\text{V}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2K (2048) cycles</td>
<td>16 ms</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>4K (4096) cycles</td>
<td>32 ms</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>8K (8192) cycles</td>
<td>64 ms</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>16K (16384) cycles</td>
<td>0.125 s</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>32K (32768) cycles</td>
<td>0.25 s</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>64K (65536) cycles</td>
<td>0.5 s</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>128K (131072) cycles</td>
<td>1.0 s</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>256K (262144) cycles</td>
<td>2.0 s</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>512K (524288) cycles</td>
<td>4.0 s</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1024K (1048576) cycles</td>
<td>8.0 s</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
<td>Reserved</td>
</tr>
</tbody>
</table>
9. Interrupts

This section describes the specifics of the interrupt handling as performed in ATmega16U4/ATmega32U4. For a general explanation of the AVR interrupt handling, refer to “Reset and Interrupt Handling” on page 15.

9.1 Interrupt Vectors in ATmega16U4/ATmega32U4

<table>
<thead>
<tr>
<th>Vector No.</th>
<th>Program Address (^{(1)})</th>
<th>Source</th>
<th>Interrupt Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$0000 (^{(1)})</td>
<td>RESET</td>
<td>External Pin, Power-on Reset, Brown-out Reset, Watchdog Reset, and JTAG AVR Reset</td>
</tr>
<tr>
<td>2</td>
<td>$0002</td>
<td>INT0</td>
<td>External Interrupt Request 0</td>
</tr>
<tr>
<td>3</td>
<td>$0004</td>
<td>INT1</td>
<td>External Interrupt Request 1</td>
</tr>
<tr>
<td>4</td>
<td>$0006</td>
<td>INT2</td>
<td>External Interrupt Request 2</td>
</tr>
<tr>
<td>5</td>
<td>$0008</td>
<td>INT3</td>
<td>External Interrupt Request 3</td>
</tr>
<tr>
<td>6</td>
<td>$000A</td>
<td>Reserved</td>
<td>Reserved</td>
</tr>
<tr>
<td>7</td>
<td>$000C</td>
<td>Reserved</td>
<td>Reserved</td>
</tr>
<tr>
<td>8</td>
<td>$000E</td>
<td>INT6</td>
<td>External Interrupt Request 6</td>
</tr>
<tr>
<td>9</td>
<td>$0010</td>
<td>Reserved</td>
<td>Reserved</td>
</tr>
<tr>
<td>10</td>
<td>$0012</td>
<td>PCINT0</td>
<td>Pin Change Interrupt Request 0</td>
</tr>
<tr>
<td>11</td>
<td>$0014</td>
<td>USB General</td>
<td>USB General Interrupt request</td>
</tr>
<tr>
<td>12</td>
<td>$0016</td>
<td>USB Endpoint</td>
<td>USB Endpoint Interrupt request</td>
</tr>
<tr>
<td>13</td>
<td>$0018</td>
<td>WDT</td>
<td>Watchdog Time-out Interrupt</td>
</tr>
<tr>
<td>14</td>
<td>$001A</td>
<td>Reserved</td>
<td>Reserved</td>
</tr>
<tr>
<td>15</td>
<td>$001C</td>
<td>Reserved</td>
<td>Reserved</td>
</tr>
<tr>
<td>16</td>
<td>$001E</td>
<td>Reserved</td>
<td>Reserved</td>
</tr>
<tr>
<td>17</td>
<td>$0020</td>
<td>TIMER1 CAPT</td>
<td>Timer/Counter1 Capture Event</td>
</tr>
<tr>
<td>18</td>
<td>$0022</td>
<td>TIMER1 COMPA</td>
<td>Timer/Counter1 Compare Match A</td>
</tr>
<tr>
<td>19</td>
<td>$0024</td>
<td>TIMER1 COMPB</td>
<td>Timer/Counter1 Compare Match B</td>
</tr>
<tr>
<td>20</td>
<td>$0026</td>
<td>TIMER1 COMPC</td>
<td>Timer/Counter1 Compare Match C</td>
</tr>
<tr>
<td>21</td>
<td>$0028</td>
<td>TIMER1 OVF</td>
<td>Timer/Counter1 Overflow</td>
</tr>
<tr>
<td>22</td>
<td>$002A</td>
<td>TIMER0 COMPA</td>
<td>Timer/Counter0 Compare Match A</td>
</tr>
<tr>
<td>23</td>
<td>$002C</td>
<td>TIMER0 COMPB</td>
<td>Timer/Counter0 Compare match B</td>
</tr>
<tr>
<td>24</td>
<td>$002E</td>
<td>TIMER0 OVF</td>
<td>Timer/Counter0 Overflow</td>
</tr>
<tr>
<td>25</td>
<td>$0030</td>
<td>SPI (STC)</td>
<td>SPI Serial Transfer Complete</td>
</tr>
<tr>
<td>26</td>
<td>$0032</td>
<td>USART1 RX</td>
<td>USART1 Rx Complete</td>
</tr>
<tr>
<td>27</td>
<td>$0034</td>
<td>USART1 UDRE</td>
<td>USART1 Data Register Empty</td>
</tr>
<tr>
<td>28</td>
<td>$0036</td>
<td>USART1TX</td>
<td>USART1 Tx Complete</td>
</tr>
<tr>
<td>29</td>
<td>$0038</td>
<td>ANALOG COMP</td>
<td>Analog Comparator</td>
</tr>
</tbody>
</table>
Table 9-1. Reset and Interrupt Vectors (Continued)

<table>
<thead>
<tr>
<th>Vector No.</th>
<th>Program Address(3)</th>
<th>Source</th>
<th>Interrupt Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>30</td>
<td>$003A</td>
<td>ADC</td>
<td>ADC Conversion Complete</td>
</tr>
<tr>
<td>31</td>
<td>$003C</td>
<td>EE READY</td>
<td>EEPROM Ready</td>
</tr>
<tr>
<td>32</td>
<td>$003E</td>
<td>TIMER3 CAPT</td>
<td>Timer/Counter3 Capture Event</td>
</tr>
<tr>
<td>33</td>
<td>$0040</td>
<td>TIMER3 COMPA</td>
<td>Timer/Counter3 Compare Match A</td>
</tr>
<tr>
<td>34</td>
<td>$0042</td>
<td>TIMER3 COMPB</td>
<td>Timer/Counter3 Compare Match B</td>
</tr>
<tr>
<td>35</td>
<td>$0044</td>
<td>TIMER3 COMPC</td>
<td>Timer/Counter3 Compare Match C</td>
</tr>
<tr>
<td>36</td>
<td>$0046</td>
<td>TIMER3 OVF</td>
<td>Timer/Counter3 Overflow</td>
</tr>
<tr>
<td>37</td>
<td>$0048</td>
<td>TWI</td>
<td>2-wire Serial Interface</td>
</tr>
<tr>
<td>38</td>
<td>$004A</td>
<td>SPM READY</td>
<td>Store Program Memory Ready</td>
</tr>
<tr>
<td>39</td>
<td>$004C</td>
<td>TIMER4 COMPA</td>
<td>Timer/Counter4 Compare Match A</td>
</tr>
<tr>
<td>40</td>
<td>$004E</td>
<td>TIMER4 COMPB</td>
<td>Timer/Counter4 Compare Match B</td>
</tr>
<tr>
<td>41</td>
<td>$0050</td>
<td>TIMER4 COMPD</td>
<td>Timer/Counter4 Compare Match D</td>
</tr>
<tr>
<td>42</td>
<td>$0052</td>
<td>TIMER4 OVF</td>
<td>Timer/Counter4 Overflow</td>
</tr>
<tr>
<td>43</td>
<td>$0054</td>
<td>TIMER4 FPF</td>
<td>Timer/Counter4 Fault Protection Interrupt</td>
</tr>
</tbody>
</table>

Notes:  
1. When the BOOTRST Fuse is programmed, the device will jump to the Boot Loader address at reset, see “Memory Programming” on page 346.
2. When the IVSEL bit in MCUCR is set, Interrupt Vectors will be moved to the start of the Boot Flash Section. The address of each Interrupt Vector will then be the address in this table added to the start address of the Boot Flash Section.

Table 9-2 shows reset and Interrupt Vectors placement for the various combinations of BOOTRST and IVSEL settings. If the program never enables an interrupt source, the Interrupt Vectors are not used, and regular program code can be placed at these locations. This is also the case if the Reset Vector is in the Application section while the Interrupt Vectors are in the Boot section or vice versa.

Table 9-2. Reset and Interrupt Vectors Placement(1)

<table>
<thead>
<tr>
<th>BOOTRST</th>
<th>IVSEL</th>
<th>Reset Address</th>
<th>Interrupt Vectors Start Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0x0000</td>
<td>0x0002</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0x0000</td>
<td>Boot Reset Address + 0x0002</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>Boot Reset Address</td>
<td>0x0002</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>Boot Reset Address</td>
<td>Boot Reset Address + 0x0002</td>
</tr>
</tbody>
</table>

Note:  
1. The Boot Reset Address is shown in Table 27-8 on page 344. For the BOOTRST Fuse “1” means unprogrammed while “0” means programmed.

9.1.1 Moving Interrupts Between Application and Boot Space

The General Interrupt Control Register controls the placement of the Interrupt Vector table.
9.1.2 MCU Control Register – MCUCR

<table>
<thead>
<tr>
<th>Bit</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
<th>MCUCR</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>R/W</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Read/Write</td>
<td>R/W</td>
<td>R</td>
<td>R</td>
<td>R/W</td>
<td>R</td>
<td>R</td>
<td>R/W</td>
<td>R/W</td>
<td></td>
</tr>
<tr>
<td>Initial Value</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

- **Bit 1 – IVSEL: Interrupt Vector Select**
  When the IVSEL bit is cleared (zero), the Interrupt Vectors are placed at the start of the Flash memory. When this bit is set (one), the Interrupt Vectors are moved to the beginning of the Boot Loader section of the Flash. The actual address of the start of the Boot Flash Section is determined by the BOOTSZ Fuses. Refer to the section “Memory Programming” on page 346 for details. To avoid unintentional changes of Interrupt Vector tables, a special write procedure must be followed to change the IVSEL bit:
  
  a. Write the Interrupt Vector Change Enable (IVCE) bit to one.
  b. Within four cycles, write the desired value to IVSEL while writing a zero to IVCE.

Interruption will automatically be disabled while this sequence is executed. Interrupts are disabled in the cycle IVCE is set, and they remain disabled until after the instruction following the write to IVSEL. If IVSEL is not written, interrupts remain disabled for four cycles. The I-bit in the Status Register is unaffected by the automatic disabling.

Note: If Interrupt Vectors are placed in the Boot Loader section and Boot Lock bit BLB02 is programmed, interrupts are disabled while executing from the Application section. If Interrupt Vectors are placed in the Application section and Boot Lock bit BLB12 is programmed, interrupts are disabled while executing from the Boot Loader section. Refer to the section “Memory Programming” on page 346 for details on Boot Lock bits.

- **Bit 0 – IVCE: Interrupt Vector Change Enable**
The IVCE bit must be written to logic one to enable change of the IVSEL bit. IVCE is cleared by hardware four cycles after it is written or when IVSEL is written. Setting the IVCE bit will disable interrupts, as explained in the IVSEL description above. See Code Example below.
Assembly Code Example

Move_interrupts:
    ; Enable change of Interrupt Vectors
    ldi r16, (1<<IVCE)
    out MCUCR, r16
    ; Move interrupts to Boot Flash section
    ldi r16, (1<<IVSEL)
    out MCUCR, r16
    ret

C Code Example

void Move_interrupts(void)
{
    /* Enable change of Interrupt Vectors */
    MCUCR = (1<<IVCE);
    /* Move interrupts to Boot Flash section */
    MCUCR = (1<<IVSEL);
}
10. I/O-Ports

10.1 Introduction

All AVR ports have true Read-Modify-Write functionality when used as general digital I/O ports. This means that the direction of one port pin can be changed without unintentionally changing the direction of any other pin with the SBI and CBI instructions. The same applies when changing drive value (if configured as output) or enabling/disabling of pull-up resistors (if configured as input). Each output buffer has symmetrical drive characteristics with both high sink and source capability. The pin driver is strong enough to drive LED displays directly. All port pins have individually selectable pull-up resistors with a supply-voltage invariant resistance. All I/O pins have protection diodes to both Vcc and Ground as indicated in Figure 10-1. Refer to “Electrical Characteristics” on page 378 for a complete list of parameters.

Figure 10-1. I/O Pin Equivalent Schematic

All registers and bit references in this section are written in general form. A lower case "x" represents the numbering letter for the port, and a lower case "n" represents the bit number. However, when using the register or bit defines in a program, the precise form must be used. For example, PORTB3 for bit no. 3 in Port B, here documented generally as PORTxn. The physical I/O Registers and bit locations are listed in “Register Description for I/O-Ports” on page 82.

Three I/O memory address locations are allocated for each port, one each for the Data Register – PORTx, Data Direction Register – DDRx, and the Port Input Pins – PINx. The Port Input Pins I/O location is read only, while the Data Register and the Data Direction Register are read/write. However, writing a logic one to a bit in the PINx Register, will result in a toggle in the corresponding bit in the Data Register. In addition, the Pull-up Disable – PUD bit in MCUCR disables the pull-up function for all pins in all ports when set.

Using the I/O port as General Digital I/O is described in “Ports as General Digital I/O” on page 66. Most port pins are multiplexed with alternate functions for the peripheral features on the device. How each alternate function interferes with the port pin is described in “Alternate Port Functions” on page 70. Refer to the individual module sections for a full description of the alternate functions.
Note that enabling the alternate function of some of the port pins does not affect the use of the other pins in the port as general digital I/O.

10.2 Ports as General Digital I/O

The ports are bi-directional I/O ports with optional internal pull-ups. Figure 10-2 shows a functional description of one I/O-port pin, here generically called Px.

Figure 10-2. General Digital I/O

Note: WRx, WPx, WDx, RRx, RPx, and RDx are common to all pins within the same port. clk/DO, SLEEP, and PUD are common to all ports.

10.2.1 Configuring the Pin

Each port pin consists of three register bits: DDxn, PORTxn, and PINxn. As shown in “Register Description for I/O-Ports” on page 82, the DDxn bits are accessed at the DDRx I/O address, the PORTxn bits at the PORTx I/O address, and the PINxn bits at the PINx I/O address.

The DDxn bit in the DDRx Register selects the direction of this pin. If DDxn is written logic one, Px is configured as an output pin. If DDxn is written logic zero, Px is configured as an input pin.

If PORTxn is written logic one when the pin is configured as an input pin, the pull-up resistor is activated. To switch the pull-up resistor off, PORTxn has to be written logic zero or the pin has to be configured as an output pin. The port pins are tri-stated when reset condition becomes active, even if no clocks are running.
If PORTx is written logic one when the pin is configured as an output pin, the port pin is driven high (one). If PORTx is written logic zero when the pin is configured as an output pin, the port pin is driven low (zero).

10.2.2 Toggling the Pin

Writing a logic one to PINx toggles the value of PORTx, independent on the value of DDRx. Note that the SBI instruction can be used to toggle one single bit in a port.

10.2.3 Switching Between Input and Output

When switching between tri-state ([DDx, PORTx] = 0b00) and output high ([DDx, PORTx] = 0b11), an intermediate state with either pull-up enabled ([DDx, PORTx] = 0b01) or output low ([DDx, PORTx] = 0b10) occurs. Normally, the pull-up enabled state is fully acceptable, as a high-impedance environment will not notice the difference between a strong high driver and a pull-up. If this is not the case, the PUD bit in the MCUCR Register can be set to disable all pull-ups in all ports.

Switching between input with pull-up and output low generates the same problem. The user must use either the tri-state ([DDx, PORTx] = 0b00) or the output high state ([DDx, PORTx] = 0b11) as an intermediate step.

Table 10-1 summarizes the control signals for the pin value.

<table>
<thead>
<tr>
<th>DDx</th>
<th>PORTx</th>
<th>PUD (in MCUCR)</th>
<th>I/O</th>
<th>Pull-up</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>X</td>
<td>Input</td>
<td>No</td>
<td>Tri-state (Hi-Z)</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>Input</td>
<td>Yes</td>
<td>Px will source current if ext. pulled low.</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>Input</td>
<td>No</td>
<td>Tri-state (Hi-Z)</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>X</td>
<td>Output</td>
<td>No</td>
<td>Output Low (Sink)</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>X</td>
<td>Output</td>
<td>No</td>
<td>Output High (Source)</td>
</tr>
</tbody>
</table>

10.2.4 Reading the Pin Value

Independent of the setting of Data Direction bit DDx, the port pin can be read through the PINx Register bit. As shown in Figure 10-2, the PINx Register bit and the preceding latch constitute a synchronizer. This is needed to avoid metastability if the physical pin changes value near the edge of the internal clock, but it also introduces a delay. Figure 10-3 shows a timing diagram of the synchronization when reading an externally applied pin value. The maximum and minimum propagation delays are denoted $t_{pd,max}$ and $t_{pd,min}$ respectively.
Consider the clock period starting shortly after the first falling edge of the system clock. The latch is closed when the clock is low, and goes transparent when the clock is high, as indicated by the shaded region of the “SYNC LATCH” signal. The signal value is latched when the system clock goes low. It is clocked into the PINxn Register at the succeeding positive clock edge. As indicated by the two arrows tpd max and tpd min, a single signal transition on the pin will be delayed between ½ and 1½ system clock period depending upon the time of assertion.

When reading back a software assigned pin value, a nop instruction must be inserted as indicated in Figure 10-4. The out instruction sets the “SYNC LATCH” signal at the positive edge of the clock. In this case, the delay tpd through the synchronizer is 1 system clock period.

The following code example shows how to set port B pins 0 and 1 high, 2 and 3 low, and define the port pins from 4 to 7 as input with pull-ups assigned to port pins 6 and 7. The resulting pin values are read back again, but as previously discussed, a nop instruction is included to be able to read back the value recently assigned to some of the pins.
Assembly Code Example

```assembly
... ; Define pull-ups and set outputs high
; Define directions for port pins
ldi r16, (1<<PB7) | (1<<PB6) | (1<<PB1) | (1<<PB0)
ldi r17, (1<<DDB3) | (1<<DDB2) | (1<<DDB1) | (1<<DDB0)
out PORTB, r16
out DDRB, r17
; Insert nop for synchronization
nop
; Read port pins
in r16, PINB
...
```

C Code Example

```c
unsigned char i;
...
/* Define pull-ups and set outputs high */
/* Define directions for port pins */
PORTB = (1<<PB7) | (1<<PB6) | (1<<PB1) | (1<<PB0);
DDRB = (1<<DDB3) | (1<<DDB2) | (1<<DDB1) | (1<<DDB0);
/* Insert nop for synchronization*/
__no_operation();
/* Read port pins */
i = PINB;
...
```

Note: 1. For the assembly program, two temporary registers are used to minimize the time from pull-ups are set on pins 0, 1, 6, and 7, until the direction bits are correctly set, defining bit 2 and 3 as low and redefining bits 0 and 1 as strong high drivers.

10.2.5 Digital Input Enable and Sleep Modes

As shown in Figure 10-2, the digital input signal can be clamped to ground at the input of the Schmidt-trigger. The signal denoted SLEEP in the figure, is set by the MCU Sleep Controller in Power-down mode, Power-save mode, and Standby mode to avoid high power consumption if some input signals are left floating, or have an analog signal level close to \( V_{CC}/2 \).

SLEEP is overridden for port pins enabled as external interrupt pins. If the external interrupt request is not enabled, SLEEP is active also for these pins. SLEEP is also overridden by various other alternate functions as described in “Alternate Port Functions” on page 70.

If a logic high level (“one”) is present on an asynchronous external interrupt pin configured as “Interrupt on Rising Edge, Falling Edge, or Any Logic Change on Pin” while the external interrupt is not enabled, the corresponding External Interrupt Flag will be set when resuming from the above mentioned Sleep mode, as the clamping in these sleep mode produces the requested logic change.
10.2.6 Unconnected Pins

If some pins are unused, it is recommended to ensure that these pins have a defined level. Even though most of the digital inputs are disabled in the deep sleep modes as described above, floating inputs should be avoided to reduce current consumption in all other modes where the digital inputs are enabled (Reset, Active mode and Idle mode).

The simplest method to ensure a defined level of an unused pin, is to enable the internal pull-up. In this case, the pull-up will be disabled during reset. If low power consumption during reset is important, it is recommended to use an external pull-up or pull-down. Connecting unused pins directly to VCC or GND is not recommended, since this may cause excessive currents if the pin is accidentally configured as an output.

10.3 Alternate Port Functions

Most port pins have alternate functions in addition to being general digital I/Os. Figure 10-5 shows how the port pin control signals from the simplified Figure 10-2 can be overridden by alternate functions. The overriding signals may not be present in all port pins, but the figure serves as a generic description applicable to all port pins in the AVR microcontroller family.

Figure 10-5. Alternate Port Functions(1)
Note: 1. WRx, WPx, WDx, RRx, RPx, and RDx are common to all pins within the same port. clk<>/SLEEP, and PUD are common to all ports. All other signals are unique for each pin.

Table 10-2 summarizes the function of the overriding signals. The pin and port indexes from Figure 10-5 are not shown in the succeeding tables. The overriding signals are generated internally in the modules having the alternate function.

Table 10-2. Generic Description of Overriding Signals for Alternate Functions

<table>
<thead>
<tr>
<th>Signal Name</th>
<th>Full Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PUOE</td>
<td>Pull-up Override Enable</td>
<td>If this signal is set, the pull-up enable is controlled by the PUOV signal. If this signal is cleared, the pull-up is enabled when (DDxn, PORTxn, PUD) = 0b010.</td>
</tr>
<tr>
<td>PUOV</td>
<td>Pull-up Override Value</td>
<td>If PUOE is set, the pull-up is enabled/disabled when PUOV is set/cleared, regardless of the setting of the DDxn, PORTxn, and PUD Register bits.</td>
</tr>
<tr>
<td>DDOE</td>
<td>Data Direction Override Enable</td>
<td>If this signal is set, the Output Driver Enable is controlled by the DDOV signal. If this signal is cleared, the Output driver is enabled by the DDxn Register bit.</td>
</tr>
<tr>
<td>DDOV</td>
<td>Data Direction Override Value</td>
<td>If DDOE is set, the Output Driver is enabled/disabled when DDOV is set/cleared, regardless of the setting of the DDxn Register bit.</td>
</tr>
<tr>
<td>PVOE</td>
<td>Port Value Override Enable</td>
<td>If this signal is set and the Output Driver is enabled, the port value is controlled by the PVOV signal. If PVOE is cleared, and the Output Driver is enabled, the port Value is controlled by the PORTxn Register bit.</td>
</tr>
<tr>
<td>PVOV</td>
<td>Port Value Override Value</td>
<td>If PVOE is set, the port value is set to PVOV, regardless of the setting of the PORTxn Register bit.</td>
</tr>
<tr>
<td>PTOE</td>
<td>Port Toggle Override Enable</td>
<td>If PTOE is set, the PORTxn Register bit is inverted.</td>
</tr>
<tr>
<td>DIEOE</td>
<td>Digital Input Enable Override Enable</td>
<td>If this bit is set, the Digital Input Enable is controlled by the DIEOV signal. If this signal is cleared, the Digital Input Enable is determined by MCU state (Normal mode, sleep mode).</td>
</tr>
<tr>
<td>DIEOV</td>
<td>Digital Input Enable Override Value</td>
<td>If DIEOE is set, the Digital Input is enabled/disabled when DIEOV is set/cleared, regardless of the MCU state (Normal mode, sleep mode).</td>
</tr>
<tr>
<td>DI</td>
<td>Digital Input</td>
<td>This is the Digital Input to alternate functions. In the figure, the signal is connected to the output of the schmitt trigger but before the synchronizer. Unless the Digital Input is used as a clock source, the module with the alternate function will use its own synchronizer.</td>
</tr>
<tr>
<td>AIO</td>
<td>Analog Input/Output</td>
<td>This is the Analog Input/output to/from alternate functions. The signal is connected directly to the pad, and can be used bi-directionally.</td>
</tr>
</tbody>
</table>

The following subsections shortly describe the alternate functions for each port, and relate the overriding signals to the alternate function. Refer to the alternate function description for further details.
10.3.1 MCU Control Register – MCUCR

<table>
<thead>
<tr>
<th>Bit</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>JTD</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
</tr>
<tr>
<td>PUD</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>IVSEL</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>IVCE</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
</tbody>
</table>

- **Bit 4 – PUD: Pull-up Disable**
  When this bit is written to one, the pull-ups in the I/O ports are disabled even if the DDxn and PORTxn Registers are configured to enable the pull-ups ((DDxn, PORTxn) = 0b01). See “Configuring the Pin” on page 66 for more details about this feature.

10.3.2 Alternate Functions of Port B

The Port B pins with alternate functions are shown in Table 10-3.

**Table 10-3. Port B Pins Alternate Functions**

<table>
<thead>
<tr>
<th>Port Pin</th>
<th>Alternate Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>PB7</td>
<td>OC0A/OC1C/PCINT7/RTS (Output Compare and PWM Output A for Timer/Counter0, Output Compare and PWM Output C for Timer/Counter1 or Pin Change Interrupt 7 or UART flow control RTS signal)</td>
</tr>
<tr>
<td>PB6</td>
<td>OC1B/PCINT6/OC.4B/ADC13 (Output Compare and PWM Output B for Timer/Counter1 or Pin Change Interrupt 6 or Timer 4 Output Compare B / PWM output or Analog to Digital Converter channel 13)</td>
</tr>
<tr>
<td>PB5</td>
<td>OC1A/PCINT5/OC.4B/ADC12 (Output Compare and PWM Output A for Timer/Counter1 or Pin Change Interrupt 5 or Timer 4 Complementary Output Compare B / PWM output or Analog to Digital Converter channel 12)</td>
</tr>
<tr>
<td>PB4</td>
<td>PCINT4/ADC11 (Pin Change Interrupt 4 or Analog to Digital Converter channel 11)</td>
</tr>
<tr>
<td>PB3</td>
<td>PDO/MISO/PCINT3 (Programming Data Output or SPI Bus Master Input/Slave Output or Pin Change Interrupt 3)</td>
</tr>
<tr>
<td>PB2</td>
<td>PDI/MOSI/PCINT2 (Programming Data Input or SPI Bus Master Output/Slave Input or Pin Change Interrupt 2)</td>
</tr>
<tr>
<td>PB1</td>
<td>SCK/PCINT1 (SPI Bus Serial Clock or Pin Change Interrupt 1)</td>
</tr>
<tr>
<td>PB0</td>
<td>SS/PCINT0 (SPI Slave Select input or Pin Change Interrupt 0)</td>
</tr>
</tbody>
</table>

The alternate pin configuration is as follows:

- **OC0A/OC1C/PCINT7/RTS, Bit 7**
  OC0A, Output Compare Match A output: The PB7 pin can serve as an external output for the Timer/Counter0 Output Compare. The pin has to be configured as an output (DDB7 set “one”) to serve this function. The OC0A pin is also the output pin for the PWM mode timer function.

  OC1C, Output Compare Match C output: The PB7 pin can serve as an external output for the Timer/Counter1 Output Compare C. The pin has to be configured as an output (DDB7 set “one”) to serve this function. The OC1C pin is also the output pin for the PWM mode timer function.

  PCINT7, Pin Change Interrupt source 7: The PB7 pin can serve as an external interrupt source.

- **RTS:** RTS flow control signal used by enhanced UART.
• **OC1B/PCINT6/OC.4B/ADC12, Bit 6**
  OC1B, Output Compare Match B output: The PB6 pin can serve as an external output for the Timer/Counter1 Output Compare B. The pin has to be configured as an output (DDB6 set “one”) to serve this function. The OC1B pin is also the output pin for the PWM mode timer function.
  PCINT6, Pin Change Interrupt source 6: The PB7 pin can serve as an external interrupt source.
  OC.4B: Timer 4 Output Compare B. This pin can be used to generate a high-speed PWM signal from Timer 4 module. The pin has to be configured as an output (DDB6 set “one”) to serve this function.

• **OC1A/PCINT5/OC.4B/ADC12, Bit 5**
  OC1A, Output Compare Match A output: The PB5 pin can serve as an external output for the Timer/Counter1 Output Compare A. The pin has to be configured as an output (DDB5 set (one)) to serve this function. The OC1A pin is also the output pin for the PWM mode timer function.
  PCINT5, Pin Change Interrupt source 5: The PB7 pin can serve as an external interrupt source.
  OC.4B: Timer 4 Output Compare B. This pin can be used to generate a high-speed PWM signal from Timer 4 module, complementary to OC.4B (PB5) signal. The pin has to be configured as an output (DDB5 set (one)) to serve this function.
  ADC12: Analog to Digital Converter, channel 12.

• **PCINT4/ADC11, Bit 4**
  PCINT4, Pin Change Interrupt source 4: The PB7 pin can serve as an external interrupt source.
  ADC11, Analog to Digital Converter channel 11.

• **PDO/MISO/PCINT3 – Port B, Bit 3**
  PDO, SPI Serial Programming Data Output. During Serial Program Downloading, this pin is used as data output line for the ATmega16U4/ATmega32U4.
  MISO: Master Data input, Slave Data output pin for SPI channel. When the SPI is enabled as a master, this pin is configured as an input regardless of the setting of DDB3. When the SPI is enabled as a slave, the data direction of this pin is controlled by DDB3. When the pin is forced to be an input, the pull-up can still be controlled by the PORTB3 bit.
  PCINT3, Pin Change Interrupt source 3: The PB7 pin can serve as an external interrupt source.

• **PDI/MOSI/PCINT2 – Port B, Bit 2**
  PDI, SPI Serial Programming Data Input. During Serial Program Downloading, this pin is used as data input line for the ATmega16U4/ATmega32U4.
  MOSI: SPI Master Data output, Slave Data input for SPI channel. When the SPI is enabled as a slave, this pin is configured as an input regardless of the setting of DDB2. When the SPI is enabled as a master, the data direction of this pin is controlled by DDB2. When the pin is forced to be an input, the pull-up can still be controlled by the PORTB2 bit.
  PCINT2, Pin Change Interrupt source 2: The PB7 pin can serve as an external interrupt source.
• **SCK/PCINT1 – Port B, Bit 1**
  SCK: Master Clock output, Slave Clock input pin for SPI channel. When the SPI is enabled as a slave, this pin is configured as an input regardless of the setting of DDB1. When the SPI0 is enabled as a master, the data direction of this pin is controlled by DDB1. When the pin is forced to be an input, the pull-up can still be controlled by the PORTB1 bit.

  PCINT1, Pin Change Interrupt source 1: The PB7 pin can serve as an external interrupt source.

• **SS/PCINT0 – Port B, Bit 0**
  SS: Slave Port Select input. When the SPI is enabled as a slave, this pin is configured as an input regardless of the setting of DDB0. As a slave, the SPI is activated when this pin is driven low. When the SPI is enabled as a master, the data direction of this pin is controlled by DDB0. When the pin is forced to be an input, the pull-up can still be controlled by the PORTB0 bit.

Table 10-4 and Table 10-5 relate the alternate functions of Port B to the overriding signals shown in Figure 10-5 on page 70. SPI MSTR INPUT and SPI SLAVE OUTPUT constitute the MISO signal, while MOSI is divided into SPI MSTR OUTPUT and SPI SLAVE INPUT.

PCINT0, Pin Change Interrupt source 0: The PB7 pin can serve as an external interrupt source.

**Table 10-4. Overriding Signals for Alternate Functions in PB7/PB4**

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>PUOE</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>PUOV</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>DDOE</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>DDOV</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>PVOE</td>
<td>OC0/OC1C ENABLE</td>
<td>OC1B ENABLE</td>
<td>OC1A ENABLE</td>
<td>0</td>
</tr>
<tr>
<td>PVOV</td>
<td>OC0/OC1C</td>
<td>OC1B</td>
<td>OC1A</td>
<td>0</td>
</tr>
<tr>
<td>DIEOE</td>
<td>PCINT7 • PCIE0</td>
<td>PCINT6 • PCIE0</td>
<td>PCINT5 • PCIE0</td>
<td>PCINT4 • PCIE0</td>
</tr>
<tr>
<td>DIEOV</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>DI</td>
<td>PCINT7 INPUT</td>
<td>PCINT6 INPUT</td>
<td>PCINT5 INPUT</td>
<td>PCINT4 INPUT</td>
</tr>
<tr>
<td>AIO</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
</tbody>
</table>
Table 10-5. Overriding Signals for Alternate Functions in PB3-PB0

<table>
<thead>
<tr>
<th>Signal Name</th>
<th>PB3/PD0/PCINT3/ MISO</th>
<th>PB2/PD1/PCINT2/ MOSI</th>
<th>PB1/PCINT1/ SCK</th>
<th>PB0/PCINT0/ SS</th>
</tr>
</thead>
<tbody>
<tr>
<td>PUOE</td>
<td>SPE • MSTR</td>
<td>SPE • MSTR</td>
<td>SPE • MSTR</td>
<td>SPE • MSTR</td>
</tr>
<tr>
<td>PUOV</td>
<td>PORTB3 • PUD</td>
<td>PORTB2 • PUD</td>
<td>PORTB1 • PUD</td>
<td>PORTB0 • PUD</td>
</tr>
<tr>
<td>DDOE</td>
<td>SPE • MSTR</td>
<td>SPE • MSTR</td>
<td>SPE • MSTR</td>
<td>SPE • MSTR</td>
</tr>
<tr>
<td>DDOV</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>PVOE</td>
<td>SPE • MSTR</td>
<td>SPE • MSTR</td>
<td>SPE • MSTR</td>
<td>0</td>
</tr>
<tr>
<td>PVOV</td>
<td>SPI SLAVE OUTPUT</td>
<td>SPI MSTR OUTPUT</td>
<td>SCK OUTPUT</td>
<td>0</td>
</tr>
<tr>
<td>DIEOE</td>
<td>PCINT3 • PCIE0</td>
<td>PCINT2 • PCIE0</td>
<td>PCINT1 • PCIE0</td>
<td>PCINT0 • PCIE0</td>
</tr>
<tr>
<td>DIEOV</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>DI</td>
<td>SPI MSTR INPUT</td>
<td>SPI SLAVE INPUT</td>
<td>SCK INPUT</td>
<td>SPI SS</td>
</tr>
<tr>
<td></td>
<td>PCINT3 INPUT</td>
<td>PCINT2 INPUT</td>
<td>PCINT1 INPUT</td>
<td>PCINT0 INPUT</td>
</tr>
<tr>
<td>AIO</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
</tbody>
</table>

10.3.3 Alternate Functions of Port C

The Port C alternate function is as follows:

Table 10-6. Port C Pins Alternate Functions

<table>
<thead>
<tr>
<th>Port Pin</th>
<th>Alternate Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>PC7</td>
<td>ICP3/CLKO/OC4A (Input Capture Timer 3 or CLK0 (Divided System Clock) or Output Compare and direct PWM output A for Timer 4)</td>
</tr>
<tr>
<td>PC6</td>
<td>OC.3A/OC4A (Output Compare and PWM output A for Timer/Counter3 or Output Compare and complementary PWM output A for Timer 4)</td>
</tr>
<tr>
<td>PC5</td>
<td>Not present on pin-out.</td>
</tr>
<tr>
<td>PC4</td>
<td></td>
</tr>
<tr>
<td>PC3</td>
<td></td>
</tr>
<tr>
<td>PC2</td>
<td></td>
</tr>
<tr>
<td>PC1</td>
<td></td>
</tr>
<tr>
<td>PC0</td>
<td></td>
</tr>
</tbody>
</table>

- ICP3/CLKO/OC4A – Port C, Bit 7
  ICP3: If Timer 3 is correctly configured, this pin can serve as Input Capture feature.
  CLKO: When the corresponding fuse is enabled, this pin outputs the internal microcontroller working frequency. If the clock prescaler is used, this will affect this output frequency.
  OC.4A: Timer 4 Output Compare A. This pin can be used to generate a high-speed PWM signal from Timer 4 module. The pin has to be configured as an output (DDC7 set "one") to serve this function.
• **OC.3A/OC.4A** – Port C, Bit 6
  
  OC.3A: Timer 3 Output Compare A. This pin can be used to generate a PWM signal from Timer 3 module.

  OC.4A: Timer 4 Output Compare A. This pin can be used to generate a high-speed PWM signal from Timer 4 module, complementary to OC.4A (PC7) signal. The pin has to be configured as an output (DDC6 set “one”) to serve this function.

  Table 10-7 relate the alternate functions of Port C to the overriding signals shown in Figure 10-5 on page 70.

  **Table 10-7.** Overriding Signals for Alternate Functions in PC7.PC6

<table>
<thead>
<tr>
<th>Signal Name</th>
<th>PC7/ICP3/CLKO/OC.4A</th>
<th>PC6/OC.3A/OC.4A</th>
</tr>
</thead>
<tbody>
<tr>
<td>PUOE</td>
<td>SRE • (XMM&lt;1)</td>
<td>SRE • (XMM&lt;2)</td>
</tr>
<tr>
<td>PUOV</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>DDOE</td>
<td>SRE • (XMM&lt;1)</td>
<td>SRE • (XMM&lt;2)</td>
</tr>
<tr>
<td>DDOV</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>PVOE</td>
<td>SRE • (XMM&lt;1)</td>
<td>SRE • (XMM&lt;2)</td>
</tr>
<tr>
<td>PVOV</td>
<td>A15</td>
<td>if (SRE,XMM&lt;2) then A14 else OC3A</td>
</tr>
<tr>
<td>DIEOE</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>DIEOV</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>DI</td>
<td>ICP3 input</td>
<td>–</td>
</tr>
<tr>
<td>AIO</td>
<td>–</td>
<td>–</td>
</tr>
</tbody>
</table>
10.3.4 Alternate Functions of Port D

The Port D pins with alternate functions are shown in Table 10-8.

<table>
<thead>
<tr>
<th>Port Pin</th>
<th>Alternate Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>PD7</td>
<td>T0/OC.4D/ADC10 (Timer/Counter0 clock Input or Timer 4 Output Compare D / PWM output or Analog to Digital Converter channel 10)</td>
</tr>
<tr>
<td>PD6</td>
<td>T1/OC.4D/ADC9 (Timer/Counter1 clock Input or Timer 4 Output Complementary Compare D / PWM output or Analog to Digital Converter channel 9)</td>
</tr>
<tr>
<td>PD5</td>
<td>XCK1/CTS (USART1 External Clock Input/Output or UART flow control CTS signal)</td>
</tr>
<tr>
<td>PD4</td>
<td>ICP1/ADC8 (Timer/Counter1 Input Capture Trigger or Analog to Digital Converter channel 8)</td>
</tr>
<tr>
<td>PD3</td>
<td>INT3/TXD1 (External Interrupt3 Input or USART1 Transmit Pin)</td>
</tr>
<tr>
<td>PD2</td>
<td>INT2/RXD1 (External Interrupt2 Input or USART1 Receive Pin)</td>
</tr>
<tr>
<td>PD1</td>
<td>INT1/SDA (External Interrupt1 Input or TWI Serial Data)</td>
</tr>
<tr>
<td>PD0</td>
<td>INT0/SCL/OC0B (External Interrupt0 Input or TWI Serial Clock or Output Compare for Timer/Counter0)</td>
</tr>
</tbody>
</table>

The alternate pin configuration is as follows:

- **T0/OC.4D/ADC10 – Port D, Bit 7**
  
  T0, Timer/Counter0 counter source.

  OC.4D: Timer 4 Output Compare D. This pin can be used to generate a high-speed PWM signal from Timer 4 module. The pin has to be configured as an output (DDD7 set “one”) to serve this function.

  ADC10: Analog to Digital Converter, Channel 10.

- **T1/OC.4D/ADC9 – Port D, Bit 6**
  
  T1, Timer/Counter1 counter source.

  OC.4D: Timer 4 Output Compare D. This pin can be used to generate a high-speed PWM signal from Timer 4 module, complementary to OC.4D (PD7) signal. The pin has to be configured as an output (DDD6 set “one”) to serve this function.

  ADC9: Analog to Digital Converter, Channel 9.

- **XCK1/CTS – Port D, Bit 5**
  
  XCK1, USART1 External clock. The Data Direction Register (DDD5) controls whether the clock is output (DDD5 set) or input (DDD5 cleared). The XCK1 pin is active only when the USART1 operates in Synchronous mode.

  CTS: Clear-To-Send flow control signal used by enhanced UART module.

- **ICP1/ADC8 – Port D, Bit 4**
  
  ICP1 – Input Capture Pin 1: The PD4 pin can act as an input capture pin for Timer/Counter1.

  ADC8: Analog to Digital Converter, Channel 8.
• **INT3/TXD1 – Port D, Bit 3**

INT3, External Interrupt source 3: The PD3 pin can serve as an external interrupt source to the MCU.

TXD1, Transmit Data (Data output pin for the USART1). When the USART1 Transmitter is enabled, this pin is configured as an output regardless of the value of DDD3.

• **INT2/RXD1 – Port D, Bit 2**

INT2, External Interrupt source 2. The PD2 pin can serve as an External Interrupt source to the MCU.

RXD1, Receive Data (Data input pin for the USART1). When the USART1 receiver is enabled this pin is configured as an input regardless of the value of DDD2. When the USART forces this pin to be an input, the pull-up can still be controlled by the PORTD2 bit.

• **INT1/SDA – Port D, Bit 1**

INT1, External Interrupt source 1. The PD1 pin can serve as an external interrupt source to the MCU.

SDA, 2-wire Serial Interface Data: When the TWEN bit in TWCR is set (one) to enable the 2-wire Serial Interface, pin PD1 is disconnected from the port and becomes the Serial Data I/O pin for the 2-wire Serial Interface. In this mode, there is a spike filter on the pin to suppress spikes shorter than 50 ns on the input signal, and the pin is driven by an open drain driver with slew-rate limitation.

• **INT0/SCL/OC0B – Port D, Bit 0**

INT0, External Interrupt source 0. The PD0 pin can serve as an external interrupt source to the MCU.

SCL, 2-wire Serial Interface Clock: When the TWEN bit in TWCR is set (one) to enable the 2-wire Serial Interface, pin PD0 is disconnected from the port and becomes the Serial Clock I/O pin for the 2-wire Serial Interface. In this mode, there is a spike filter on the pin to suppress spikes shorter than 50 ns on the input signal, and the pin is driven by an open drain driver with slew-rate limitation.

OC.0B: Timer 0 Output Compare B. This pin can be used to generate a PWM signal from the Timer 0 module.

Table 10-9 and Table 10-10 relates the alternate functions of Port D to the overriding signals shown in Figure 10-5 on page 70.
Table 10-9. Overriding Signals for Alternate Functions PD7..PD4

<table>
<thead>
<tr>
<th>Signal Name</th>
<th>PD7/T0/OC4D/ADC10</th>
<th>PD6/T1/OC4D/ADC9</th>
<th>PD5/XCK1/CTS</th>
<th>PD4/ICP1/ADC8</th>
</tr>
</thead>
<tbody>
<tr>
<td>PUOE</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>PUOV</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>DDOE</td>
<td>0</td>
<td>0</td>
<td>XCK1 OUTPUT ENABLE</td>
<td>0</td>
</tr>
<tr>
<td>DDOV</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>PVOE</td>
<td>0</td>
<td>0</td>
<td>XCK1 OUTPUT ENABLE</td>
<td>0</td>
</tr>
<tr>
<td>PVOV</td>
<td>0</td>
<td>0</td>
<td>XCK1 OUTPUT</td>
<td>0</td>
</tr>
<tr>
<td>DIEO</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>DIEV</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>DI</td>
<td>T0 INPUT</td>
<td>T1 INPUT</td>
<td>XCK1 INPUT</td>
<td>ICP1 INPUT</td>
</tr>
<tr>
<td>AIO</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
</tbody>
</table>

Table 10-10. Overriding Signals for Alternate Functions in PD3.PD0\(^1\)

<table>
<thead>
<tr>
<th>Signal Name</th>
<th>PD3/INT3/TXD1</th>
<th>PD2/INT2/RXD1</th>
<th>PD1/INT1/SDA</th>
<th>PD0/INTO/SCL/OC0B</th>
</tr>
</thead>
<tbody>
<tr>
<td>PUOE</td>
<td>TXEN1</td>
<td>RXEN1</td>
<td>TWEN</td>
<td>TWEN</td>
</tr>
<tr>
<td>PUOV</td>
<td>0</td>
<td>PORTD2 • PUD</td>
<td>PORTD1 • PUD</td>
<td>PORTD0 • PUD</td>
</tr>
<tr>
<td>DDOE</td>
<td>TXEN1</td>
<td>RXEN1</td>
<td>TWEN</td>
<td>TWEN</td>
</tr>
<tr>
<td>DDOV</td>
<td>1</td>
<td>0</td>
<td>SDA_OUT</td>
<td>SCL_OUT</td>
</tr>
<tr>
<td>PVOE</td>
<td>TXEN1</td>
<td>0</td>
<td>TWEN ENABLE</td>
<td>TWEN</td>
</tr>
<tr>
<td>PVOV</td>
<td>TXD1</td>
<td>0</td>
<td>0</td>
<td>OC0B</td>
</tr>
<tr>
<td>DIEO</td>
<td>INT3 ENABLE</td>
<td>INT2 ENABLE</td>
<td>INT1 ENABLE</td>
<td>INT0 ENABLE</td>
</tr>
<tr>
<td>DIEV</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>DI</td>
<td>INT3 INPUT</td>
<td>INT2 INPUT/RXD1</td>
<td>INT1 INPUT</td>
<td>INT0 INPUT</td>
</tr>
<tr>
<td>AIO</td>
<td>–</td>
<td>–</td>
<td>SDA INPUT</td>
<td>SCL INPUT</td>
</tr>
</tbody>
</table>

Note: 1. When enabled, the 2-wire Serial Interface enables Slew-Rate controls on the output pins PD0 and PD1. This is not shown in this table. In addition, spike filters are connected between the AIO outputs shown in the port figure and the digital logic of the TWI module.
10.3.5 Alternate Functions of Port E

The Port E pins with alternate functions are shown in Table 10-11.

<table>
<thead>
<tr>
<th>Port Pin</th>
<th>Alternate Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>PE7</td>
<td>Not present on pin-out.</td>
</tr>
<tr>
<td>PE6</td>
<td>INT6/AIN0 (External Interrupt 6 Input or Analog Comparator Positive Input)</td>
</tr>
<tr>
<td>PE5</td>
<td></td>
</tr>
<tr>
<td>PE4</td>
<td>Not present on pin-out.</td>
</tr>
<tr>
<td>PE3</td>
<td></td>
</tr>
<tr>
<td>PE2</td>
<td>HWB (Hardware bootloader activation)</td>
</tr>
<tr>
<td>PE1</td>
<td>Not present on pin-out.</td>
</tr>
<tr>
<td>PE0</td>
<td></td>
</tr>
</tbody>
</table>

- **INT6/AIN0 – Port E, Bit 6**
  INT6, External Interrupt source 6: The PE6 pin can serve as an external interrupt source.
  AIN0 – Analog Comparator Negative input. This pin is directly connected to the negative input of the Analog Comparator.

- **HWB – Port E, Bit 2**
  HWB allows to execute the bootloader section after reset when tied to ground during external reset pulse. The HWB mode of this pin is active only when the HWBE fuse is enable. During normal operation (excluded Reset), this pin acts as a general purpose I/O.

10.3.6 Alternate Functions of Port F

The Port F has an alternate function as analog input for the ADC as shown in Table 10-13. If some Port F pins are configured as outputs, it is essential that these do not switch when a conversion is in progress. This might corrupt the result of the conversion. If the JTAG interface is
enabled, the pull-up resistors on pins PF7(TDI), PF5(TMS), and PF4(TCK) will be activated even if a Reset occurs.

**Table 10-13. Port F Pins Alternate Functions**

<table>
<thead>
<tr>
<th>Port Pin</th>
<th>Alternate Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>PF7</td>
<td>ADC7/TDI (ADC input channel 7 or JTAG Test Data Input)</td>
</tr>
<tr>
<td>PF6</td>
<td>ADC6/TDO (ADC input channel 6 or JTAG Test Data Output)</td>
</tr>
<tr>
<td>PF5</td>
<td>ADC5/TMS (ADC input channel 5 or JTAG Test Mode Select)</td>
</tr>
<tr>
<td>PF4</td>
<td>ADC4/TCK (ADC input channel 4 or JTAG Test Clock)</td>
</tr>
<tr>
<td>PF3</td>
<td>Not present on pin-out.</td>
</tr>
<tr>
<td>PF2</td>
<td>ADC1 (ADC input channel 1)</td>
</tr>
<tr>
<td>PF1</td>
<td>ADC0 (ADC input channel 0)</td>
</tr>
</tbody>
</table>

- **TDI, ADC7 – Port F, Bit 7**
  ADC7, Analog to Digital Converter, Channel 7.

TDI, JTAG Test Data In: Serial input data to be shifted in to the Instruction Register or Data Register (scan chains). When the JTAG interface is enabled, this pin can not be used as an I/O pin.

- **TDO, ADC6 – Port F, Bit 6**
  ADC6, Analog to Digital Converter, Channel 6.

TDO, JTAG Test Data Out: Serial output data from Instruction Register or Data Register. When the JTAG interface is enabled, this pin can not be used as an I/O pin.

The TDO pin is tri-stated unless TAP states that shift out data are entered.

- **TMS, ADC5 – Port F, Bit 5**
  ADC5, Analog to Digital Converter, Channel 5.

TMS, JTAG Test Mode Select: This pin is used for navigating through the TAP-controller state machine. When the JTAG interface is enabled, this pin can not be used as an I/O pin.

- **TCK, ADC4 – Port F, Bit 4**
  ADC4, Analog to Digital Converter, Channel 4.

TCK, JTAG Test Clock: JTAG operation is synchronous to TCK. When the JTAG interface is enabled, this pin can not be used as an I/O pin.

- **ADC3 – ADC0 – Port F, Bit 1..0**
  Analog to Digital Converter, Channel 1.0
Table 10-14. Overriding Signals for Alternate Functions in PF7..PF4

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>PUOE</td>
<td>JTAGEN</td>
<td>JTAGEN</td>
<td>JTAGEN</td>
<td>JTAGEN</td>
</tr>
<tr>
<td>PUOV</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>DDOE</td>
<td>JTAGEN</td>
<td>JTAGEN</td>
<td>JTAGEN</td>
<td>JTAGEN</td>
</tr>
<tr>
<td>DDOV</td>
<td>0</td>
<td>SHIFT_IR</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>PVOE</td>
<td>0</td>
<td>JTAGEN</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>PVOV</td>
<td>0</td>
<td>TDO</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>DIEOE</td>
<td>JTAGEN</td>
<td>JTAGEN</td>
<td>JTAGEN</td>
<td>JTAGEN</td>
</tr>
<tr>
<td>DIEOV</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>DI</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>AIO</td>
<td>TDI/ADC7 INPUT</td>
<td>ADC6 INPUT</td>
<td>TMS/ADC5 INPUT</td>
<td>TCK/ADC4 INPUT</td>
</tr>
</tbody>
</table>

Table 10-15. Overriding Signals for Alternate Functions in PF1..PF0

<table>
<thead>
<tr>
<th>Signal Name</th>
<th>PF1/ADC1</th>
<th>PF0/ADC0</th>
</tr>
</thead>
<tbody>
<tr>
<td>PUOE</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>PUOV</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>DDOE</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>DDOV</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>PVOE</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>PVOV</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>DIEOE</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>DIEOV</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>DI</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>AIO</td>
<td>ADC1 INPUT</td>
<td>ADC0 INPUT</td>
</tr>
</tbody>
</table>

10.4 Register Description for I/O-Ports

10.4.1 Port B Data Register – PORTB

<table>
<thead>
<tr>
<th>Bit</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
<th>POR</th>
<th>R/W</th>
<th>R/W</th>
<th>R/W</th>
<th>R/W</th>
<th>R/W</th>
<th>R/W</th>
<th>R/W</th>
<th>R/W</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initial Value</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
### 10.4.2 Port B Data Direction Register – DDRB

<table>
<thead>
<tr>
<th>Bit</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>DDR7</td>
<td>DDR6</td>
<td>DDR5</td>
<td>DDR4</td>
<td>DDR3</td>
<td>DDR2</td>
<td>DDR1</td>
<td>DDR0</td>
<td>DDRB</td>
</tr>
<tr>
<td>Read/Write</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
</tr>
<tr>
<td>Initial Value</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

### 10.4.3 Port B Input Pins Address – PINB

<table>
<thead>
<tr>
<th>Bit</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>PINB7</td>
<td>PINB6</td>
<td>PINB5</td>
<td>PINB4</td>
<td>PINB3</td>
<td>PINB2</td>
<td>PINB1</td>
<td>PINB0</td>
<td>PINB</td>
</tr>
<tr>
<td>Read/Write</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
</tr>
<tr>
<td>Initial Value</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
</tr>
</tbody>
</table>

### 10.4.4 Port C Data Register – PORTC

<table>
<thead>
<tr>
<th>Bit</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>PORTC7</td>
<td>PORTC6</td>
<td>PORTC5</td>
<td>PORTC4</td>
<td>PORTC3</td>
<td>PORTC2</td>
<td>PORTC1</td>
<td>PORTC0</td>
<td>PORTC</td>
</tr>
<tr>
<td>Read/Write</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
</tr>
<tr>
<td>Initial Value</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

### 10.4.5 Port C Data Direction Register – DDRC

<table>
<thead>
<tr>
<th>Bit</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>DDC7</td>
<td>DDC6</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>DDRC</td>
</tr>
<tr>
<td>Read/Write</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
</tr>
<tr>
<td>Initial Value</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

### 10.4.6 Port C Input Pins Address – PINC

<table>
<thead>
<tr>
<th>Bit</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>PINC7</td>
<td>PINC6</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>PINC</td>
</tr>
<tr>
<td>Read/Write</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
</tr>
<tr>
<td>Initial Value</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
</tr>
</tbody>
</table>

### 10.4.7 Port D Data Register – PORTD

<table>
<thead>
<tr>
<th>Bit</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>PORTD7</td>
<td>PORTD6</td>
<td>PORTD5</td>
<td>PORTD4</td>
<td>PORTD3</td>
<td>PORTD2</td>
<td>PORTD1</td>
<td>PORTD0</td>
<td>PORTD</td>
</tr>
<tr>
<td>Read/Write</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
</tr>
<tr>
<td>Initial Value</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

### 10.4.8 Port D Data Direction Register – DDRD

<table>
<thead>
<tr>
<th>Bit</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>DDD7</td>
<td>DDD6</td>
<td>DDD5</td>
<td>DDD4</td>
<td>DDD3</td>
<td>DDD2</td>
<td>DDD1</td>
<td>DDD0</td>
<td>DDRD</td>
</tr>
<tr>
<td>Read/Write</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
</tr>
<tr>
<td>Initial Value</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

### 10.4.9 Port D Input Pins Address – PIND

<table>
<thead>
<tr>
<th>Bit</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>PIND7</td>
<td>PIND6</td>
<td>PIND5</td>
<td>PIND4</td>
<td>PIND3</td>
<td>PIND2</td>
<td>PIND1</td>
<td>PIND0</td>
<td>PIND</td>
</tr>
<tr>
<td>Read/Write</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
</tr>
<tr>
<td>Initial Value</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
</tr>
</tbody>
</table>
### 10.4.10 Port E Data Register – PORTE

<table>
<thead>
<tr>
<th>Bit</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
<th>PORTE</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>-</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Read/Write</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td></td>
</tr>
<tr>
<td>Initial Value</td>
<td>0 0 0 0 0 0 0 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### 10.4.11 Port E Data Direction Register – DDRE

<table>
<thead>
<tr>
<th>Bit</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
<th>DDRE</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>-</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Read/Write</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td></td>
</tr>
<tr>
<td>Initial Value</td>
<td>0 0 0 0 0 0 0 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### 10.4.12 Port E Input Pins Address – PINE

<table>
<thead>
<tr>
<th>Bit</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
<th>PINE</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>-</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Read/Write</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td></td>
</tr>
<tr>
<td>Initial Value</td>
<td>N/A N/A N/A N/A N/A N/A N/A N/A</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### 10.4.13 Port F Data Register – PORTF

<table>
<thead>
<tr>
<th>Bit</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
<th>PORTF</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>PORTF 7</td>
<td>PORTF 6</td>
<td>PORTF 5</td>
<td>PORTF 4</td>
<td>-</td>
<td>-</td>
<td>PORTF 1</td>
<td>PORTF 0</td>
<td></td>
</tr>
<tr>
<td>Read/Write</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td></td>
</tr>
<tr>
<td>Initial Value</td>
<td>0 0 0 0 0 0 0 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### 10.4.14 Port F Data Direction Register – DDRF

<table>
<thead>
<tr>
<th>Bit</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
<th>DDRF</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>DDF7</td>
<td>DDF6</td>
<td>DDF5</td>
<td>DDF4</td>
<td>-</td>
<td>-</td>
<td>DDF1</td>
<td>DDF0</td>
<td></td>
</tr>
<tr>
<td>Read/Write</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td></td>
</tr>
<tr>
<td>Initial Value</td>
<td>0 0 0 0 0 0 0 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### 10.4.15 Port F Input Pins Address – PINF

<table>
<thead>
<tr>
<th>Bit</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
<th>PINF</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>PINF7</td>
<td>PINF6</td>
<td>PINF5</td>
<td>PINF4</td>
<td>-</td>
<td>-</td>
<td>PINF1</td>
<td>PINF0</td>
<td></td>
</tr>
<tr>
<td>Read/Write</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td></td>
</tr>
<tr>
<td>Initial Value</td>
<td>N/A N/A N/A N/A N/A N/A N/A N/A</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
11. External Interrupts

The External Interrupts are triggered by the INT6, INT3:0 pin or any of the PCINT7..0 pins. Observe that, if enabled, the interrupts will trigger even if the INT[6;3:0] or PCINT7..0 pins are configured as outputs. This feature provides a way of generating a software interrupt.

The Pin change interrupt PC10 will trigger if any enabled PCINT7:0 pin toggles. PCMSK0 Register control which pins contribute to the pin change interrupts. Pin change interrupts on PCINT7 ..0 are detected asynchronously. This implies that these interrupts can be used for waking the part also from sleep modes other than Idle mode.

The External Interrupts can be triggered by a falling or rising edge or a low level. This is set up as indicated in the specification for the External Interrupt Control Registers – EICRA (INT3:0) and EICRB (INT6). When the external interrupt is enabled and is configured as level triggered, the interrupt will trigger as long as the pin is held low. Note that recognition of falling or rising edge interrupts on INT6 requires the presence of an I/O clock, described in “System Clock and Clock Options” on page 27. Low level interrupts and the edge interrupt on INT3:0 are detected asynchronously. This implies that these interrupts can be used for waking the part also from sleep modes other than Idle mode. The I/O clock is halted in all sleep modes except Idle mode.

Note that if a level triggered interrupt is used for wake-up from Power-down, the required level must be held long enough for the MCU to complete the wake-up to trigger the level interrupt. If the level disappears before the end of the Start-up Time, the MCU will still wake up, but no interrupt will be generated. The start-up time is defined by the SUT and CKSEL Fuses as described in “System Clock and Clock Options” on page 27.

11.0.1 External Interrupt Control Register A – EICRA

The External Interrupt Control Register A contains control bits for interrupt sense control.

<table>
<thead>
<tr>
<th>Bit</th>
<th>ISC31</th>
<th>ISC30</th>
<th>ISC21</th>
<th>ISC20</th>
<th>ISC11</th>
<th>ISC10</th>
<th>ISC01</th>
<th>ISC00</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read/Write</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
</tr>
<tr>
<td>Initial Value</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

- **Bits 7..0 – ISC31, ISC30 – ISC00, ISC00: External Interrupt 3 - 0 Sense Control Bits**

The External Interrupts 3 - 0 are activated by the external pins INT3:0 if the SREG I-flag and the corresponding interrupt mask in the EIMSK is set. The level and edges on the external pins that activate the interrupts are defined in Table 11-1. Edges on INT3..INT0 are registered asynchronously. Pulses on INT3:0 pins wider than the minimum pulse width given in Table 11-2 will generate an interrupt. Shorter pulses are not guaranteed to generate an interrupt. If low level interrupt is selected, the low level must be held until the completion of the currently executing instruction to generate an interrupt. If enabled, a level triggered interrupt will generate an interrupt request as long as the pin is held low. When changing the ISCn bit, an interrupt can occur. Therefore, it is recommended to first disable INTn by clearing its Interrupt Enable bit in the EIMSK Register. Then, the ISCn bit can be changed. Finally, the INTn interrupt flag should be cleared by writing a logical one to its Interrupt Flag bit (INTF) in the EIFR Register before the interrupt is re-enabled.
Table 11-1. Interrupt Sense Control

<table>
<thead>
<tr>
<th>ISCn1</th>
<th>ISCn0</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>The low level of INTn generates an interrupt request.</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>Any edge of INTn generates asynchronously an interrupt request.</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>The falling edge of INTn generates asynchronously an interrupt request.</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>The rising edge of INTn generates asynchronously an interrupt request.</td>
</tr>
</tbody>
</table>

Note: 1. n = 3, 2, 1or 0.
When changing the ISCn1/ISCn0 bits, the interrupt must be disabled by clearing its Interrupt Enable bit in the EIMSK Register. Otherwise an interrupt can occur when the bits are changed.

Table 11-2. Asynchronous External Interrupt Characteristics

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter</th>
<th>Condition</th>
<th>Min</th>
<th>Typ</th>
<th>Max</th>
<th>Units</th>
</tr>
</thead>
</table>
| tINT   | Minimum pulse width for asynchronous external interrupt | 50 | ns |}

11.0.2 External Interrupt Control Register B – EICRB

<table>
<thead>
<tr>
<th>Bit</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>-</td>
<td>-</td>
<td>ISC61</td>
<td>ISC60</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Read/Write</th>
<th>R/W</th>
<th>R/W</th>
<th>R/W</th>
<th>R/W</th>
<th>R/W</th>
<th>R/W</th>
<th>R/W</th>
<th>R/W</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initial Value</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

- Bit 7..6 – Res: Reserved Bits
  These bits are reserved bits in the ATmega16U4/ATmega32U4 and always read as zero.

- Bits 5, 4 – ISC61, ISC60: External Interrupt 6 Sense Control Bits
  The External Interrupt 6 is activated by the external pin INT6 if the SREG I-flag and the corresponding interrupt mask in the EIMSK is set. The level and edges on the external pin that activate the interrupt are defined in Table 11-3. The value on the INT6 pin are sampled before detecting edges. If edge or toggle interrupt is selected, pulses that last longer than one clock period will generate an interrupt. Shorter pulses are not guaranteed to generate an interrupt. Observe that CPU clock frequency can be lower than the XTAL frequency if the XTAL divider is enabled. If low level interrupt is selected, the low level must be held until the completion of the currently executing instruction to generate an interrupt. If enabled, a level triggered interrupt will generate an interrupt request as long as the pin is held low.

Table 11-3. Interrupt Sense Control

<table>
<thead>
<tr>
<th>ISC61</th>
<th>ISC60</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>The low level of INT6 generates an interrupt request.</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>Any logical change on INT6 generates an interrupt request.</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>The falling edge between two samples of INT6 generates an interrupt request.</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>The rising edge between two samples of INT6 generates an interrupt request.</td>
</tr>
</tbody>
</table>

Note: 1. When changing the ISC61/ISC60 bits, the interrupt must be disabled by clearing its Interrupt Enable bit in the EIMSK Register. Otherwise an interrupt can occur when the bits are changed.
• **Bit 3.0 – Res: Reserved Bits**
These bits are reserved bits in the ATmega16U4/ATmega32U4 and always read as zero.

### 11.0.3 External Interrupt Mask Register – EIMSK

<table>
<thead>
<tr>
<th>Bit</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read/Write</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
</tr>
<tr>
<td>Initial Value</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

• **Bits 7..0 – INT6, INT3 – INT0: External Interrupt Request 6, 3 - 0 Enable**
When an INT[6;3;0] bit is written to one and the I-bit in the Status Register (SREG) is set (one), the corresponding external pin interrupt is enabled. The Interrupt Sense Control bits in the External Interrupt Control Registers – EICRA and EICRB – defines whether the external interrupt is activated on rising or falling edge or level sensed. Activity on any of these pins will trigger an interrupt request even if the pin is enabled as an output. This provides a way of generating a software interrupt.

### 11.0.4 External Interrupt Flag Register – EIFR

<table>
<thead>
<tr>
<th>Bit</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read/Write</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
</tr>
<tr>
<td>Initial Value</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

• **Bits 7..0 – INTF6, INTF3 - INTF0: External Interrupt Flags 6, 3 - 0**
When an edge or logic change on the INT[6;3;0] pin triggers an interrupt request, INTF7:0 becomes set (one). If the I-bit in SREG and the corresponding interrupt enable bit, INT[6;3;0] in EIMSK, are set (one), the MCU will jump to the interrupt vector. The flag is cleared when the interrupt routine is executed. Alternatively, the flag can be cleared by writing a logical one to it. These flags are always cleared when INT[6;3;0] are configured as level interrupt. Note that when entering sleep mode with the INT3:0 interrupts disabled, the input buffers on these pins will be disabled. This may cause a logic change in internal signals which will set the INTF3:0 flags. See "Digital Input Enable and Sleep Modes" on page 69 for more information.

### 11.0.5 Pin Change Interrupt Control Register - PCICR

<table>
<thead>
<tr>
<th>Bit</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read/Write</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R/W</td>
</tr>
<tr>
<td>Initial Value</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

• **Bit 0 – PCIE0: Pin Change Interrupt Enable 0**
When the PCIE0 bit is set (one) and the I-bit in the Status Register (SREG) is set (one), pin change interrupt 0 is enabled. Any change on any enabled PCINT7..0 pin will cause an interrupt. The corresponding interrupt of Pin Change Interrupt Request is executed from the PC10 Interrupt Vector. PCINT7..0 pins are enabled individually by the PCMSK0 Register.

### 11.0.6 Pin Change Interrupt Flag Register – PCIFR

<table>
<thead>
<tr>
<th>Bit</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read/Write</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R/W</td>
</tr>
<tr>
<td>Initial Value</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
• **Bit 0 – PCIF0: Pin Change Interrupt Flag 0**
When a logic change on any PCINT7..0 pin triggers an interrupt request, PCIF0 becomes set (one). If the I-bit in SREG and the PCIE0 bit in EIMSK are set (one), the MCU will jump to the corresponding Interrupt Vector. The flag is cleared when the interrupt routine is executed. Alternatively, the flag can be cleared by writing a logical one to it.

### 11.0.7 Pin Change Mask Register 0 – PCMSK0

<table>
<thead>
<tr>
<th>Bit</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
<th>PCMSK0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read/Write</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td></td>
</tr>
<tr>
<td>Initial Value</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

• **Bit 7..0 – PCINT7..0: Pin Change Enable Mask 7..0**
Each PCINT7..0 bit selects whether pin change interrupt is enabled on the corresponding I/O pin. If PCINT7..0 is set and the PCIE0 bit in PCICR is set, pin change interrupt is enabled on the corresponding I/O pin. If PCINT7..0 is cleared, pin change interrupt on the corresponding I/O pin is disabled.
12. Timer/Counter0, Timer/Counter1, and Timer/Counter3 Prescalers

Timer/Counter0, 1, and 3 share the same prescaler module, but the Timer/Counters can have different prescaler settings. The description below applies to all Timer/Counters. Tn is used as a general name, n = 0, 1 or 3.

12.1 Internal Clock Source

The Timer/Counter can be clocked directly by the system clock (by setting the CSn2:0 = 1). This provides the fastest operation, with a maximum Timer/Counter clock frequency equal to system clock frequency (f_{CLK, IO}). Alternatively, one of four taps from the prescaler can be used as a clock source. The prescaled clock has a frequency of either f_{CLK, IO}/8, f_{CLK, IO}/64, f_{CLK, IO}/256, or f_{CLK, IO}/1024.

12.2 Prescaler Reset

The prescaler is free running, i.e., operates independently of the Clock Select logic of the Timer/Counter, and it is shared by the Timer/Counter Tn. Since the prescaler is not affected by the Timer/Counter’s clock select, the state of the prescaler will have implications for situations where a prescaled clock is used. One example of prescaling artifacts occurs when the timer is enabled and clocked by the prescaler (6 > CSn2:0 > 1). The number of system clock cycles from when the timer is enabled to the first count occurs can be from 1 to N+1 system clock cycles, where N equals the prescaler divisor (8, 64, 256, or 1024).

It is possible to use the prescaler reset for synchronizing the Timer/Counter to program execution. However, care must be taken if the other Timer/Counter that shares the same prescaler also uses prescaling. A prescaler reset will affect the prescaler period for all Timer/Counters it is connected to.

12.3 External Clock Source

An external clock source applied to the Tn pin can be used as Timer/Counter clock (clk_{Tn}). The Tn pin is sampled once every system clock cycle by the pin synchronization logic. The synchronized (sampled) signal is then passed through the edge detector. Figure 12-1 shows a functional equivalent block diagram of the Tn synchronization and edge detector logic. The registers are clocked at the positive edge of the internal system clock (clk_{io}). The latch is transparent in the high period of the internal system clock.

The edge detector generates one clk_{Tn} pulse for each positive (CSn2:0 = 7) or negative (CSn2:0 = 6) edge it detects.

Figure 12-1. Tn/T0 Pin Sampling

The synchronization and edge detector logic introduces a delay of 2.5 to 3.5 system clock cycles from an edge has been applied to the Tn pin to the counter is updated.

Enabling and disabling of the clock input must be done when Tn has been stable for at least one system clock cycle, otherwise it is a risk that a false Timer/Counter clock pulse is generated.
Each half period of the external clock applied must be longer than one system clock cycle to ensure correct sampling. The external clock must be guaranteed to have less than half the system clock frequency \( f_{\text{ExtClk}} < \frac{f_{\text{clk,I/O}}}{2} \) given a 50/50% duty cycle. Since the edge detector uses sampling, the maximum frequency of an external clock it can detect is half the sampling frequency (Nyquist sampling theorem). However, due to variation of the system clock frequency and duty cycle caused by Oscillator source (crystal, resonator, and capacitors) tolerances, it is recommended that maximum frequency of an external clock source is less than \( \frac{f_{\text{clk,I/O}}}{2.5} \).

An external clock source cannot be prescaled.

**Figure 12-2.** Prescaler for synchronous Timer/Counters

Note: T3 input is not available on the ATmega16U4/ATmega32U4 products. “Tn” only refers to either T0 or T1 inputs.

### 12.4 General Timer/Counter Control Register – GTCCR

<table>
<thead>
<tr>
<th>Bit</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
<th>GTCCR</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read/Write</td>
<td>R/W</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R/W</td>
<td>R/W</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Initial Value</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

- **Bit 7 – TSM: Timer/Counter Synchronization Mode**

  Writing the TSM bit to one activates the Timer/Counter Synchronization mode. In this mode, the value that is written to the PSRASY and PSRSYNC bits is kept, hence keeping the corresponding prescaler reset signals asserted. This ensures that the corresponding Timer/Counters are halted and can be configured to the same value without the risk of one of them advancing during configuration. When the TSM bit is written to zero, the PSRASY and PSRSYNC bits are cleared by hardware, and the Timer/Counters start counting simultaneously.

- **Bit 0 – PSRSYNC: Prescaler Reset for Synchronous Timer/Counters**

  When this bit is one, Timer/Counter0 and Timer/Counter1 and Timer/Counter3 prescaler will be Reset. This bit is normally cleared immediately by hardware, except if the TSM bit is set. Note that Timer/Counter0, Timer/Counter1 and Timer/Counter3 share the same prescaler and a reset of this prescaler will affect all timers.
13. 8-bit Timer/Counter0 with PWM

Timer/Counter0 is a general purpose 8-bit Timer/Counter module, with two independent Output Compare Units, and with PWM support. It allows accurate program execution timing (event management) and wave generation. The main features are:

- Two Independent Output Compare Units
- Double Buffered Output Compare Registers
- Clear Timer on Compare Match (Auto Reload)
- Glitch Free, Phase Correct Pulse Width Modulator (PWM)
- Variable PWM Period
- Frequency Generator
- Three Independent Interrupt Sources (TOV0, OCF0A, and OCF0B)

13.1 Overview

A simplified block diagram of the 8-bit Timer/Counter is shown in Figure 13-1. For the actual placement of I/O pins, refer to “Pinout ATmega16U4/ATmega32U4” on page 3. CPU accessible I/O Registers, including I/O bits and I/O pins, are shown in bold. The device-specific I/O Register and bit locations are listed in the “8-bit Timer/Counter Register Description” on page 102.

Figure 13-1. 8-bit Timer/Counter Block Diagram

13.1.1 Registers

The Timer/Counter (TCNT0) and Output Compare Registers (OCRA0A and OCR0B) are 8-bit registers. Interrupt request (abbreviated to Int Req. in the figure) signals are all visible in the Timer Interrupt Flag Register (TIFR0). All interrupts are individually masked with the Timer Interrupt Mask Register (TIMSK0). TIFR0 and TIMSK0 are not shown in the figure.

The Timer/Counter can be clocked internally, via the prescaler, or by an external clock source on the T0 pin. The Clock Select logic block controls which clock source and edge the Timer/Counter uses to increment (or decrement) its value. The Timer/Counter is inactive when no clock source is selected. The output from the Clock Select logic is referred to as the timer clock (clkT0).
The double buffered Output Compare Registers (OCR0A and OCR0B) are compared with the Timer/Counter value at all times. The result of the compare can be used by the Waveform Generator to generate a PWM or variable frequency output on the Output Compare pins (OC0A and OC0B). See “Output Compare Unit” on page 93, for details. The Compare Match event will also set the Compare Flag (OCF0A or OCF0B) which can be used to generate an Output Compare interrupt request.

### 13.1.2 Definitions

Many register and bit references in this section are written in general form. A lower case “n” replaces the Timer/Counter number, in this case 0. A lower case “x” replaces the Output Compare Unit, in this case Compare Unit A or Compare Unit B. However, when using the register or bit defines in a program, the precise form must be used, i.e., TCNT0 for accessing Timer/Counter0 counter value and so on.

The definitions in the table below are also used extensively throughout the document.

| BOTTOM | The counter reaches the BOTTOM when it becomes 0x00. |
| MAX    | The counter reaches its MAXimum when it becomes 0xFF (decimal 255). |
| TOP    | The counter reaches the TOP when it becomes equal to the highest value in the count sequence. The TOP value can be assigned to be the fixed value 0xFF (MAX) or the value stored in the OCR0A Register. The assignment is dependent on the mode of operation. |

### 13.2 Timer/Counter Clock Sources

The Timer/Counter can be clocked by an internal or an external clock source. The clock source is selected by the Clock Select logic which is controlled by the Clock Select (CS02:0) bits located in the Timer/Counter Control Register (TCCR0B). For details on clock sources and prescaler, see “Timer/Counter0, Timer/Counter1, and Timer/Counter3 Prescalers” on page 89.

### 13.3 Counter Unit

The main part of the 8-bit Timer/Counter is the programmable bi-directional counter unit. Figure 13-2 shows a block diagram of the counter and its surroundings.

**Figure 13-2. Counter Unit Block Diagram**

Signal description (internal signals):

- **count**: Increment or decrement TCNT0 by 1.
- **direction**: Select between increment and decrement.
clear Clear TCNT0 (set all bits to zero).
	
clk_{Tn} Timer/Counter clock, referred to as clk_{T0} in the following.

top Signalize that TCNT0 has reached maximum value.

top Signalize that TCNT0 has reached minimum value (zero).

Depending of the mode of operation used, the counter is cleared, incremented, or decremented at each timer clock (clk_{Tn}). clk_{T0} can be generated from an external or internal clock source, selected by the Clock Select bits (CS02:0). When no clock source is selected (CS02:0 = 0) the timer is stopped. However, the TCNT0 value can be accessed by the CPU, regardless of whether clk_{T0} is present or not. A CPU write overrides (has priority over) all counter clear or count operations.

The counting sequence is determined by the setting of the WGM01 and WGM00 bits located in the Timer/Counter Control Register (TCCR0A) and the WGM02 bit located in the Timer/Counter Control Register B (TCCR0B). There are close connections between how the counter behaves (counts) and how waveforms are generated on the Output Compare outputs OC0A and OC0B. For more details about advanced counting sequences and waveform generation, see “Modes of Operation” on page 96.

The Timer/Counter Overflow Flag (TOV0) is set according to the mode of operation selected by the WGM02:0 bits. TOV0 can be used for generating a CPU interrupt.

13.4 Output Compare Unit

The 8-bit comparator continuously compares TCNT0 with the Output Compare Registers (OCR0A and OCR0B). Whenever TCNT0 equals OCR0A or OCR0B, the comparator signals a match. A match will set the Output Compare Flag (OCF0A or OCF0B) at the next timer clock cycle. If the corresponding interrupt is enabled, the Output Compare Flag generates an Output Compare interrupt. The Output Compare Flag is automatically cleared when the interrupt is executed. Alternatively, the flag can be cleared by software by writing a logical one to its I/O bit location. The Waveform Generator uses the match signal to generate an output according to operating mode set by the WGM02:0 bits and Compare Output mode (COM0x1:0) bits. The max and bottom signals are used by the Waveform Generator for handling the special cases of the extreme values in some modes of operation (“Modes of Operation” on page 96).

Figure 13-3 shows a block diagram of the Output Compare unit.
The OCR0x Registers are double buffered when using any of the Pulse Width Modulation (PWM) modes. For the normal and Clear Timer on Compare (CTC) modes of operation, the double buffering is disabled. The double buffering synchronizes the update of the OCR0x Compare Registers to either top or bottom of the counting sequence. The synchronization prevents the occurrence of odd-length, non-symmetrical PWM pulses, thereby making the output glitch-free.

The OCR0x Register access may seem complex, but this is not the case. When the double buffering is enabled, the CPU has access to the OCR0x Buffer Register, and if double buffering is disabled the CPU will access the OCR0x directly.

### 13.4.1 Force Output Compare

In non-PWM waveform generation modes, the match output of the comparator can be forced by writing a one to the Force Output Compare (FOC0x) bit. Forcing Compare Match will not set the OCF0x Flag or reload/clear the timer, but the OC0x pin will be updated as if a real Compare Match had occurred (the COM0x1:0 bits settings define whether the OC0x pin is set, cleared or toggled).

### 13.4.2 Compare Match Blocking by TCNT0 Write

All CPU write operations to the TCNT0 Register will block any Compare Match that occur in the next timer clock cycle, even when the timer is stopped. This feature allows OCR0x to be initialized to the same value as TCNT0 without triggering an interrupt when the Timer/Counter clock is enabled.

### 13.4.3 Using the Output Compare Unit

Since writing TCNT0 in any mode of operation will block all Compare Matches for one timer clock cycle, there are risks involved when changing TCNT0 when using the Output Compare Unit, independently of whether the Timer/Counter is running or not. If the value written to TCNT0 equals the OCR0x value, the Compare Match will be missed, resulting in incorrect waveform
generation. Similarly, do not write the TCNT0 value equal to BOTTOM when the counter is down-counting.

The setup of the OC0x should be performed before setting the Data Direction Register for the port pin to output. The easiest way of setting the OC0x value is to use the Force Output Compare (FOC0x) strobe bits in Normal mode. The OC0x Registers keep their values even when changing between Waveform Generation modes.

Be aware that the COM0x1:0 bits are not double buffered together with the compare value. Changing the COM0x1:0 bits will take effect immediately.

13.5 Compare Match Output Unit

The Compare Output mode (COM0x1:0) bits have two functions. The Waveform Generator uses the COM0x1:0 bits for defining the Output Compare (OC0x) state at the next Compare Match. Also, the COM0x1:0 bits control the OC0x pin output source. Figure 13-4 shows a simplified schematic of the logic affected by the COM0x1:0 bit setting. The I/O Registers, I/O bits, and I/O pins in the figure are shown in bold. Only the parts of the general I/O Port Control Registers (DDR and PORT) that are affected by the COM0x1:0 bits are shown. When referring to the OC0x state, the reference is for the internal OC0x Register, not the OC0x pin. If a system reset occurs, the OC0x Register is reset to “0”.

Figure 13-4. Compare Match Output Unit, Schematic

The general I/O port function is overridden by the Output Compare (OC0x) from the Waveform Generator if either of the COM0x1:0 bits are set. However, the OC0x pin direction (input or output) is still controlled by the Data Direction Register (DDR) for the port pin. The Data Direction Register bit for the OC0x pin (DDR_OC0x) must be set as output before the OC0x value is visible on the pin. The port override function is independent of the Waveform Generation mode.

The design of the Output Compare pin logic allows initialization of the OC0x state before the output is enabled. Note that some COM0x1:0 bit settings are reserved for certain modes of operation. See “8-bit Timer/Counter Register Description” on page 102.
13.5.1 Compare Output Mode and Waveform Generation
The Waveform Generator uses the COM0x1:0 bits differently in Normal, CTC, and PWM modes. For all modes, setting the COM0x1:0 = 0 tells the Waveform Generator that no action on the OC0x Register is to be performed on the next Compare Match. For compare output actions in the non-PWM modes refer to Table 13-2 on page 102. For fast PWM mode, refer to Table 13-3 on page 102, and for phase correct PWM refer to Table 13-4 on page 103.

A change of the COM0x1:0 bits state will have effect at the first Compare Match after the bits are written. For non-PWM modes, the action can be forced to have immediate effect by using the FOC0x strobe bits.

13.6 Modes of Operation
The mode of operation, i.e., the behavior of the Timer/Counter and the Output Compare pins, is defined by the combination of the Waveform Generation mode (WGM02:0) and Compare Output mode (COM0x1:0) bits. The Compare Output mode bits do not affect the counting sequence, while the Waveform Generation mode bits do. The COM0x1:0 bits control whether the PWM output generated should be inverted or not (inverted or non-inverted PWM). For non-PWM modes the COM0x1:0 bits control whether the output should be set, cleared, or toggled at a Compare Match (See “Compare Match Output Unit” on page 95).

For detailed timing information see “Timer/Counter Timing Diagrams” on page 100.

13.6.1 Normal Mode
The simplest mode of operation is the Normal mode (WGM02:0 = 0). In this mode the counting direction is always up (incrementing), and no counter clear is performed. The counter simply overruns when it passes its maximum 8-bit value (TOP = 0xFF) and then restarts from the bottom (0x00). In normal operation the Timer/Counter Overflow Flag (TOV0) will be set in the same timer clock cycle as the TCNT0 becomes zero. The TOV0 Flag in this case behaves like a ninth bit, except that it is only set, not cleared. However, combined with the timer overflow interrupt that automatically clears the TOV0 Flag, the timer resolution can be increased by software. There are no special cases to consider in the Normal mode, a new counter value can be written anytime.

The Output Compare Unit can be used to generate interrupts at some given time. Using the Output Compare to generate waveforms in Normal mode is not recommended, since this will occupy too much of the CPU time.

13.6.2 Clear Timer on Compare Match (CTC) Mode
In Clear Timer on Compare or CTC mode (WGM02:0 = 2), the OCR0A Register is used to manipulate the counter resolution. In CTC mode the counter is cleared to zero when the counter value (TCNT0) matches the OCR0A. The OCR0A defines the top value for the counter, hence also its resolution. This mode allows greater control of the Compare Match output frequency. It also simplifies the operation of counting external events.

The timing diagram for the CTC mode is shown in Figure 13-5. The counter value (TCNT0) increases until a Compare Match occurs between TCNT0 and OCR0A, and then counter (TCNT0) is cleared.
An interrupt can be generated each time the counter value reaches the TOP value by using the OCF0A Flag. If the interrupt is enabled, the interrupt handler routine can be used for updating the TOP value. However, changing TOP to a value close to BOTTOM when the counter is running with none or a low prescaler value must be done with care since the CTC mode does not have the double buffering feature. If the new value written to OCR0A is lower than the current value of TCNT0, the counter will miss the Compare Match. The counter will then have to count to its maximum value (0xFF) and wrap around starting at 0x00 before the Compare Match can occur.

For generating a waveform output in CTC mode, the OC0A output can be set to toggle its logical level on each Compare Match by setting the Compare Output mode bits to toggle mode (COM0A1:0 = 1). The OC0A value will not be visible on the port pin unless the data direction for the pin is set to output. The waveform generated will have a maximum frequency of \( f_{OC0} = \frac{f_{clk \ I/O}}{2 \cdot N \cdot (1 + OCRnx)} \)

The \( N \) variable represents the prescaler factor (1, 8, 64, 256, or 1024).

As for the Normal mode of operation, the TOV0 Flag is set in the same timer clock cycle that the counter counts from MAX to 0x00.

### 13.6.3 Fast PWM Mode

The fast Pulse Width Modulation or fast PWM mode (WGM02:0 = 3 or 7) provides a high frequency PWM waveform generation option. The fast PWM differs from the other PWM option by its single-slope operation. The counter counts from BOTTOM to TOP then restarts from BOTTOM. TOP is defined as 0xFF when WGM2:0 = 3, and OCR0A when WGM2:0 = 7. In non-inverting Compare Output mode, the Output Compare (OC0x) is cleared on the Compare Match between TCNT0 and OCR0x, and set at BOTTOM. In inverting Compare Output mode, the output is set on Compare Match and cleared at BOTTOM. Due to the single-slope operation, the operating frequency of the fast PWM mode can be twice as high as the phase correct PWM mode that use dual-slope operation. This high frequency makes the fast PWM mode well suited for power regulation, rectification, and DAC applications. High frequency allows physically small sized external components (coils, capacitors), and therefore reduces total system cost.

In fast PWM mode, the counter is incremented until the counter value matches the TOP value. The counter is then cleared at the following timer clock cycle. The timing diagram for the fast
PWM mode is shown in Figure 13-6. The TCNT0 value is in the timing diagram shown as a histogram for illustrating the single-slope operation. The diagram includes non-inverted and inverted PWM outputs. The small horizontal line marks on the TCNT0 slopes represent Compare Matches between OCR0x and TCNT0.

**Figure 13-6. Fast PWM Mode, Timing Diagram**

The Timer/Counter Overflow Flag (TOV0) is set each time the counter reaches TOP. If the interrupt is enabled, the interrupt handler routine can be used for updating the compare value.

In fast PWM mode, the compare unit allows generation of PWM waveforms on the OC0x pins. Setting the COM0x1:0 bits to two will produce a non-inverted PWM and an inverted PWM output can be generated by setting the COM0x1:0:0 bits to one allows the OC0A pin to toggle on Compare Matches if the WGM02 bit is set. This option is not available for the OC0B pin (See Table 13-3 on page 102). The actual OC0x value will only be visible on the port pin if the data direction for the port pin is set as output. The PWM waveform is generated by setting (or clearing) the OC0x Register at the Compare Match between OCR0x and TCNT0, and clearing (or setting) the OC0x Register at the timer clock cycle the counter is cleared (changes from TOP to BOTTOM).

The PWM frequency for the output can be calculated by the following equation:

\[ f_{OC_n\text{PWM}} = \frac{f_{\text{clk\_I/O}}}{N \cdot 256} \]

The \( N \) variable represents the prescaler factor (1, 8, 64, 256, or 1024).

The extreme values for the OCR0A Register represents special cases when generating a PWM waveform output in the fast PWM mode. If the OCR0A is set equal to BOTTOM, the output will be a narrow spike for each MAX+1 timer clock cycle. Setting the OCR0A equal to MAX will result in a constantly high or low output (depending on the polarity of the output set by the COM0A1:0 bits.)

A frequency (with 50% duty cycle) waveform output in fast PWM mode can be achieved by setting OC0x to toggle its logical level on each Compare Match (COM0x1:0 = 1). The waveform generated will have a maximum frequency of \( f_{OC0} = f_{\text{clk\_I/O}}/2 \) when OCR0A is set to zero. This
feature is similar to the OC0A toggle in CTC mode, except the double buffer feature of the Output Compare unit is enabled in the fast PWM mode.

13.6.4 Phase Correct PWM Mode

The phase correct PWM mode (WGM02:0 = 1 or 5) provides a high resolution phase correct PWM waveform generation option. The phase correct PWM mode is based on a dual-slope operation. The counter counts repeatedly from BOTTOM to TOP and then from TOP to BOTTOM. TOP is defined as 0xFF when WGM2:0 = 1, and OCR0A when WGM2:0 = 5. In non-inverting Compare Output mode, the Output Compare (OC0x) is cleared on the Compare Match between TCNT0 and OCR0x while up counting, and set on the Compare Match while down-counting. In inverting Output Compare mode, the operation is inverted. The dual-slope operation has lower maximum operation frequency than single slope operation. However, due to the symmetric feature of the dual-slope PWM modes, these modes are preferred for motor control applications.

In phase correct PWM mode the counter is incremented until the counter value matches TOP. When the counter reaches TOP, it changes the count direction. The TCNT0 value will be equal to TOP for one timer clock cycle. The timing diagram for the phase correct PWM mode is shown on Figure 13-7. The TCNT0 value is in the timing diagram shown as a histogram for illustrating the dual-slope operation. The diagram includes non-inverted and inverted PWM outputs. The small horizontal line marks on the TCNT0 slopes represent Compare Matches between OCR0x and TCNT0.

**Figure 13-7. Phase Correct PWM Mode, Timing Diagram**

The Timer/Counter Overflow Flag (TOV0) is set each time the counter reaches BOTTOM. The Interrupt Flag can be used to generate an interrupt each time the counter reaches the BOTTOM value.

In phase correct PWM mode, the compare unit allows generation of PWM waveforms on the OC0x pins. Setting the COM0x1:0 bits to two will produce a non-inverted PWM. An inverted PWM output can be generated by setting the COM0x1:0 to three: Setting the COM0A0 bits to
one allows the OC0A pin to toggle on Compare Matches if the WGM02 bit is set. This option is not available for the OC0B pin (See Table 13-4 on page 103). The actual OC0x value will only be visible on the port pin if the data direction for the port pin is set as output. The PWM waveform is generated by clearing (or setting) the OC0x Register at the Compare Match between OCR0x and TCNT0 when the counter increments, and setting (or clearing) the OC0x Register at Compare Match between OCR0x and TCNT0 when the counter decrements. The PWM frequency for the output when using phase correct PWM can be calculated by the following equation:

\[ f_{OCxPCPWM} = \frac{f_{clk_{I/O}}}{N \cdot 510} \]

The N variable represents the prescaler factor (1, 8, 64, 256, or 1024).

The extreme values for the OCR0A Register represent special cases when generating a PWM waveform output in the phase correct PWM mode. If the OCR0A is set equal to BOTTOM, the output will be continuously low and if set equal to MAX the output will be continuously high for non-inverted PWM mode. For inverted PWM the output will have the opposite logic values.

At the very start of period 2 in Figure 13-7 OCnx has a transition from high to low even though there is no Compare Match. The point of this transition is to guarantee symmetry around BOTTOM. There are two cases that give a transition without Compare Match.

- OCR0A changes its value from MAX, like in Figure 13-7. When the OCR0A value is MAX the OCn pin value is the same as the result of a down-counting Compare Match. To ensure symmetry around BOTTOM the OCn value at MAX must correspond to the result of an up-counting Compare Match.
- The timer starts counting from a value higher than the one in OCR0A, and for that reason misses the Compare Match and hence the OCn change that would have happened on the way up.

### 13.7 Timer/Counter Timing Diagrams

The Timer/Counter is a synchronous design and the timer clock (clk\(_{I/O}\)) is therefore shown as a clock enable signal in the following figures. The figures include information on when Interrupt Flags are set. Figure 13-8 contains timing data for basic Timer/Counter operation. The figure shows the count sequence close to the MAX value in all modes other than phase correct PWM mode.

**Figure 13-8.** Timer/Counter Timing Diagram, no Prescaling

![Timer/Counter Timing Diagram, no Prescaling](image)

*Figure 13-9 shows the same timing data, but with the prescaler enabled.*
Figure 13-9. Timer/Counter Timing Diagram, with Prescaler ($f_{\text{clk,L}}/8$)

Figure 13-10 shows the setting of OCF0B in all modes and OCF0A in all modes except CTC mode and PWM mode, where OCR0A is TOP.

Figure 13-10. Timer/Counter Timing Diagram, Setting of OCF0x, with Prescaler ($f_{\text{clk,L}}/8$)

Figure 13-11 shows the setting of OCF0A and the clearing of TCNT0 in CTC mode and fast PWM mode where OCR0A is TOP.

Figure 13-11. Timer/Counter Timing Diagram, Clear Timer on Compare Match mode, with Prescaler ($f_{\text{clk,L}}/8$)
13.8 8-bit Timer/Counter Register Description

13.8.1 Timer/Counter Control Register A – TCCR0A

<table>
<thead>
<tr>
<th>Bit</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
<th>TCCR0A</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read/Write</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R</td>
<td>R</td>
<td>R/W</td>
<td>R/W</td>
<td></td>
</tr>
<tr>
<td>Initial Value</td>
<td>0 0 0 0 0 0 0 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **Bits 7:6 – COM0A1:0: Compare Match Output A Mode**
  These bits control the Output Compare pin (OC0A) behavior. If one or both of the COM0A1:0 bits are set, the OC0A output overrides the normal port functionality of the I/O pin it is connected to. However, note that the Data Direction Register (DDR) bit corresponding to the OC0A pin must be set in order to enable the output driver.

  When OC0A is connected to the pin, the function of the COM0A1:0 bits depends on the WGM02:0 bit setting. **Table 13-2** shows the COM0A1:0 bit functionality when the WGM02:0 bits are set to a normal or CTC mode (non-PWM).

<table>
<thead>
<tr>
<th>COM0A1</th>
<th>COM0A0</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>Normal port operation, OC0A disconnected.</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>Toggle OC0A on Compare Match</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>Clear OC0A on Compare Match</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>Set OC0A on Compare Match</td>
</tr>
</tbody>
</table>

**Table 13-3** shows the COM0A1:0 bit functionality when the WGM01:0 bits are set to fast PWM mode.

<table>
<thead>
<tr>
<th>COM0A1</th>
<th>COM0A0</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>Normal port operation, OC0A disconnected.</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>WGM02 = 0: Normal Port Operation, OC0A Disconnected. WGM02 = 1: Toggle OC0A on Compare Match.</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>Clear OC0A on Compare Match, set OC0A at TOP</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>Set OC0A on Compare Match, clear OC0A at TOP</td>
</tr>
</tbody>
</table>

Note: 1. A special case occurs when OCR0A equals TOP and COM0A1 is set. In this case, the Compare Match is ignored, but the set or clear is done at TOP. See “Fast PWM Mode” on page 97 for more details.
Table 13-4 shows the COM0A1:0 bit functionality when the WGM02:0 bits are set to phase correct PWM mode.

<table>
<thead>
<tr>
<th>COM0A1</th>
<th>COM0A0</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>Normal port operation, OC0A disconnected.</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>WGM02 = 0: Normal Port Operation, OC0A Disconnected.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>WGM02 = 1: Toggle OC0A on Compare Match.</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>Clear OC0A on Compare Match when up-counting. Set OC0A on Compare Match when down-counting.</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>Set OC0A on Compare Match when up-counting. Clear OC0A on Compare Match when down-counting.</td>
</tr>
</tbody>
</table>

Note: 1. A special case occurs when OCR0A equals TOP and COM0A1 is set. In this case, the Compare Match is ignored, but the set or clear is done at TOP. See “Phase Correct PWM Mode” on page 99 for more details.

- **Bits 5:4 – COM0B1:0: Compare Match Output B Mode**
  These bits control the Output Compare pin (OC0B) behavior. If one or both of the COM0B1:0 bits are set, the OC0B output overrides the normal port functionality of the I/O pin it is connected to. However, note that the Data Direction Register (DDR) bit corresponding to the OC0B pin must be set in order to enable the output driver.

When OC0B is connected to the pin, the function of the COM0B1:0 bits depends on the WGM02:0 bit setting. Table 13-2 shows the COM0A1:0 bit functionality when the WGM02:0 bits are set to a normal or CTC mode (non-PWM).

<table>
<thead>
<tr>
<th>COM01</th>
<th>COM00</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>Normal port operation, OC0B disconnected.</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>Toggle OC0B on Compare Match</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>Clear OC0B on Compare Match</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>Set OC0B on Compare Match</td>
</tr>
</tbody>
</table>

Table 13-3 shows the COM0B1:0 bit functionality when the WGM02:0 bits are set to fast PWM mode.

<table>
<thead>
<tr>
<th>COM01</th>
<th>COM00</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>Normal port operation, OC0B disconnected.</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>Reserved</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>Clear OC0B on Compare Match, set OC0B at TOP</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>Set OC0B on Compare Match, clear OC0B at TOP</td>
</tr>
</tbody>
</table>

Note: 1. A special case occurs when OCR0B equals TOP and COM0B1 is set. In this case, the Compare Match is ignored, but the set or clear is done at TOP. See “Fast PWM Mode” on page 97 for more details.
Table 13-4 shows the COM0B1:0 bit functionality when the WGM02:0 bits are set to phase correct PWM mode.

### Table 13-7. Compare Output Mode, Phase Correct PWM Mode (1)

<table>
<thead>
<tr>
<th>COM0A1</th>
<th>COM0A0</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>Normal port operation, OC0B disconnected.</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>Reserved</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>Clear OC0B on Compare Match when up-counting. Set OC0B on Compare Match when down-counting.</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>Set OC0B on Compare Match when up-counting. Clear OC0B on Compare Match when down-counting.</td>
</tr>
</tbody>
</table>

Note: 1. A special case occurs when OCR0B equals TOP and COM0B1 is set. In this case, the Compare Match is ignored, but the set or clear is done at TOP. See “Phase Correct PWM Mode” on page 99 for more details.

- **Bits 3, 2 – Res: Reserved Bits**
  These bits are reserved bits in the ATmega16U4/ATmega32U4 and will always read as zero.

- **Bits 1:0 – WGM01:0: Waveform Generation Mode**
  Combined with the WGM02 bit found in the TCCR0B Register, these bits control the counting sequence of the counter, the source for maximum (TOP) counter value, and what type of waveform generation to be used, see Table 13-8. Modes of operation supported by the Timer/Counter unit are: Normal mode (counter), Clear Timer on Compare Match (CTC) mode, and two types of Pulse Width Modulation (PWM) modes (see “Modes of Operation” on page 96).

### Table 13-8. Waveform Generation Mode Bit Description

<table>
<thead>
<tr>
<th>Mode</th>
<th>WGM2</th>
<th>WGM1</th>
<th>WGM0</th>
<th>Timer/Counter Mode of Operation</th>
<th>TOP</th>
<th>Update of OCRx at</th>
<th>TOV Flag Set on (1)(2)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>Normal</td>
<td>0xFF</td>
<td>Immediate</td>
<td>MAX</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>PWM, Phase Correct</td>
<td>0xFF</td>
<td>TOP</td>
<td>BOTTOM</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>CTC</td>
<td>OCRA</td>
<td>Immediate</td>
<td>MAX</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>Fast PWM</td>
<td>0xFF</td>
<td>TOP</td>
<td>MAX</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>Reserved</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>PWM, Phase Correct</td>
<td>OCRA</td>
<td>TOP</td>
<td>BOTTOM</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>Reserved</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>Fast PWM</td>
<td>OCRA</td>
<td>TOP</td>
<td>TOP</td>
</tr>
</tbody>
</table>

Notes: 1. MAX = 0xFF
   2. BOTTOM = 0x00
13.8.2 Timer/Counter Control Register B – TCCR0B

<table>
<thead>
<tr>
<th>Bit</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read/Write</td>
<td>W</td>
<td>W</td>
<td>R</td>
<td>R</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
</tr>
<tr>
<td>Initial Value</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

- **Bit 7 – FOC0A: Force Output Compare A**
  The FOC0A bit is only active when the WGM bits specify a non-PWM mode.

  However, for ensuring compatibility with future devices, this bit must be set to zero when TCCR0B is written when operating in PWM mode. When writing a logical one to the FOC0A bit, an immediate Compare Match is forced on the Waveform Generation unit. The OC0A output is changed according to its COM0A1:0 bits setting. Note that the FOC0A bit is implemented as a strobe. Therefore it is the value present in the COM0A1:0 bits that determines the effect of the forced compare.

  A FOC0A strobe will not generate any interrupt, nor will it clear the timer in CTC mode using OCR0A as TOP.

  The FOC0A bit is always read as zero.

- **Bit 6 – FOC0B: Force Output Compare B**
  The FOC0B bit is only active when the WGM bits specify a non-PWM mode.

  However, for ensuring compatibility with future devices, this bit must be set to zero when TCCR0B is written when operating in PWM mode. When writing a logical one to the FOC0B bit, an immediate Compare Match is forced on the Waveform Generation unit. The OC0B output is changed according to its COM0B1:0 bits setting. Note that the FOC0B bit is implemented as a strobe. Therefore it is the value present in the COM0B1:0 bits that determines the effect of the forced compare.

  A FOC0B strobe will not generate any interrupt, nor will it clear the timer in CTC mode using OCR0B as TOP.

  The FOC0B bit is always read as zero.

- **Bits 5:4 – Res: Reserved Bits**
  These bits are reserved bits and will always read as zero.

- **Bit 3 – WGM02: Waveform Generation Mode**
  See the description in the “Timer/Counter Control Register A – TCCR0A” on page 102.

- **Bits 2:0 – CS02:0: Clock Select**
  The three Clock Select bits select the clock source to be used by the Timer/Counter.

**Table 13-9. Clock Select Bit Description**

<table>
<thead>
<tr>
<th>CS02</th>
<th>CS01</th>
<th>CS00</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>No clock source (Timer/Counter stopped)</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>clk_{I/O}/(No prescaling)</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>clk_{I/O}/8 (From prescaler)</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>clk_{I/O}/64 (From prescaler)</td>
</tr>
</tbody>
</table>
Table 13-9. Clock Select Bit Description (Continued)

<table>
<thead>
<tr>
<th>CS02</th>
<th>CS01</th>
<th>CS00</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>clk_in/256 (From prescaler)</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>clk_in/1024 (From prescaler)</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>External clock source on T0 pin. Clock on falling edge.</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>External clock source on T0 pin. Clock on rising edge.</td>
</tr>
</tbody>
</table>

If external pin modes are used for the Timer/Counter0, transitions on the T0 pin will clock the counter even if the pin is configured as an output. This feature allows software control of the counting.

13.8.3 Timer/Counter Register – TCNT0

The Timer/Counter Register gives direct access, both for read and write operations, to the Timer/Counter unit 8-bit counter. Writing to the TCNT0 Register blocks (removes) the Compare Match on the following timer clock. Modifying the counter (TCNT0) while the counter is running, introduces a risk of missing a Compare Match between TCNT0 and the OCR0x Registers.

13.8.4 Output Compare Register A – OCR0A

The Output Compare Register A contains an 8-bit value that is continuously compared with the counter value (TCNT0). A match can be used to generate an Output Compare interrupt, or to generate a waveform output on the OC0A pin.

13.8.5 Output Compare Register B – OCR0B

The Output Compare Register B contains an 8-bit value that is continuously compared with the counter value (TCNT0). A match can be used to generate an Output Compare interrupt, or to generate a waveform output on the OC0B pin.

13.8.6 Timer/Counter Interrupt Mask Register – TIMSK0

- Bits 7..3, 0 – Res: Reserved Bits
  These bits are reserved bits and will always read as zero.
• **Bit 2 – OCIE0B: Timer/Counter Output Compare Match B Interrupt Enable**
  When the OCIE0B bit is written to one, and the I-bit in the Status Register is set, the Timer/Counter Compare Match B interrupt is enabled. The corresponding interrupt is executed if a Compare Match in Timer/Counter occurs, i.e., when the OCF0B bit is set in the Timer/Counter Interrupt Flag Register – TIFR0.

• **Bit 1 – OCIE0A: Timer/Counter0 Output Compare Match A Interrupt Enable**
  When the OCIE0A bit is written to one, and the I-bit in the Status Register is set, the Timer/Counter0 Compare Match A interrupt is enabled. The corresponding interrupt is executed if a Compare Match in Timer/Counter0 occurs, i.e., when the OCF0A bit is set in the Timer/Counter 0 Interrupt Flag Register – TIFR0.

• **Bit 0 – TOIE0: Timer/Counter0 Overflow Interrupt Enable**
  When the TOIE0 bit is written to one, and the I-bit in the Status Register is set, the Timer/Counter0 Overflow interrupt is enabled. The corresponding interrupt is executed if an overflow in Timer/Counter0 occurs, i.e., when the TOV0 bit is set in the Timer/Counter 0 Interrupt Flag Register – TIFR0.

### 13.8.7 Timer/Counter 0 Interrupt Flag Register – TIFR0

<table>
<thead>
<tr>
<th>Bit</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
<th>TIFR0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read/Write</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Initial Value</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

• **Bits 7..3, 0 – Res: Reserved Bits**
  These bits are reserved bits in the ATmega16U4/ATmega32U4 and will always read as zero.

• **Bit 2 – OCF0B: Timer/Counter 0 Output Compare B Match Flag**
  The OCF0B bit is set when a Compare Match occurs between the Timer/Counter and the data in OCR0B – Output Compare Register0 B. OCF0B is cleared by hardware when executing the corresponding interrupt handling vector. Alternatively, OCF0B is cleared by writing a logic one to the flag. When the I-bit in SREG, OCIE0B (Timer/Counter Compare B Match Interrupt Enable), and OCF0B are set, the Timer/Counter Compare Match Interrupt is executed.

• **Bit 1 – OCF0A: Timer/Counter 0 Output Compare A Match Flag**
  The OCF0A bit is set when a Compare Match occurs between the Timer/Counter0 and the data in OCR0A – Output Compare Register0. OCF0A is cleared by hardware when executing the corresponding interrupt handling vector. Alternatively, OCF0A is cleared by writing a logic one to the flag. When the I-bit in SREG, OCIE0A (Timer/Counter0 Compare Match Interrupt Enable), and OCF0A are set, the Timer/Counter0 Compare Match Interrupt is executed.

• **Bit 0 – TOV0: Timer/Counter0 Overflow Flag**
  The bit TOV0 is set when an overflow occurs in Timer/Counter0. TOV0 is cleared by hardware when executing the corresponding interrupt handling vector. Alternatively, TOV0 is cleared by writing a logic one to the flag. When the SREG I-bit, TOIE0 (Timer/Counter0 Overflow Interrupt Enable), and TOV0 are set, the Timer/Counter0 Overflow interrupt is executed.

The setting of this flag is dependent of the WGM02:0 bit setting. Refer to Table 13-8, “Waveform Generation Mode Bit Description” on page 104.
14. 16-bit Timers/Counters (Timer/Counter1 and Timer/Counter3)

The 16-bit Timer/Counter unit allows accurate program execution timing (event management), wave generation, and signal timing measurement. The main features are:

- True 16-bit Design (i.e., Allows 16-bit PWM)
- Three independent Output Compare Units
- Double Buffered Output Compare Registers
- One Input Capture Unit
- Input Capture Noise Canceler
- Clear Timer on Compare Match (Auto Reload)
- Glitch-free, Phase Correct Pulse Width Modulator (PWM)
- Variable PWM Period
- Frequency Generator
- External Event Counter
- Ten independent interrupt sources (TOV1, OCF1A, OCF1B, OCF1C, ICF1, TOV3, OCF3A, OCF3B, OCF3C and ICF3)

14.1 Overview

Most register and bit references in this section are written in general form. A lower case “n” replaces the Timer/Counter number, and a) lower case “x” replaces the Output Compare unit channel. However, when using the register or bit defines in a program, the precise form must be used, i.e., TCNT1 for accessing Timer/Counter1 counter value and so on.

A simplified block diagram of the 16-bit Timer/Counter is shown in Figure 14-1. For the actual placement of I/O pins, see “Pinout ATmega16U4/ATmega32U4” on page 3. CPU accessible I/O Registers, including I/O bits and I/O pins, are shown in bold. The device-specific I/O Register and bit locations are listed in the “16-bit Timers/Counters (Timer/Counter1 and Timer/Counter3)” on page 108.

The Power Reduction Timer/Counter1 bit, PRTIM1, in “Power Reduction Register 0 - PRR0” on page 46 must be written to zero to enable Timer/Counter1 module.

The Power Reduction Timer/Counter3 bit, PRTIM3, in “Power Reduction Register 1 - PRR1” on page 46 must be written to zero to enable Timer/Counter3 module.
Figure 14-1. 16-bit Timer/Counter Block Diagram

Note: 1. Refer to “Pinout ATmega16U4/ATmega32U4” on page 3, Table 10-3 on page 72, and Table 10-6 on page 75 for Timer/Counter1 and 3 and 3 pin placement and description.
2. Tn only refers to T1 since T3 input is not available on the ATmega16U4/ATmega32U4 product.

14.1.1 Registers

The Timer/Counter (TCNTn), Output Compare Registers (OCRnA/B/C), and Input Capture Register (ICRn) are all 16-bit registers. Special procedures must be followed when accessing the 16-bit registers. These procedures are described in the section “Accessing 16-bit Registers” on page 110. The Timer/Counter Control Registers (TCCRNnA/B/C) are 8-bit registers and have no CPU access restrictions. Interrupt requests (shorten as Int.Req.) signals are all visible in the Timer Interrupt Flag Register (TIFRn). All interrupts are individually masked with the Timer Interrupt Mask Register (TIMSKn). TIFRn and TIMSKn are not shown in the figure since these registers are shared by other timer units.

The Timer/Counter can be clocked internally, via the prescaler, or by an external clock source on the Tn pin. The Clock Select logic block controls which clock source and edge the Timer/Counter uses to increment (or decrement) its value. The Timer/Counter is inactive when no clock source is selected. The output from the clock select logic is referred to as the timer clock (clkTn).
The double buffered Output Compare Registers (OCRnA/B/C) are compared with the Timer/Counter value at all time. The result of the compare can be used by the Waveform Generator to generate a PWM or variable frequency output on the Output Compare pin (OCnA/B/C). See “Output Compare Units” on page 117. The compare match event will also set the Compare Match Flag (OCFnA/B/C) which can be used to generate an Output Compare interrupt request.

The Input Capture Register can capture the Timer/Counter value at a given external (edge triggered) event on either the Input Capture pin (ICPn) or on the Analog Comparator pins (See “Analog Comparator” on page 289.) The Input Capture unit includes a digital filtering unit (Noise Canceler) for reducing the chance of capturing noise spikes.

The TOP value, or maximum Timer/Counter value, can in some modes of operation be defined by either the OCRnA Register, the ICRn Register, or by a set of fixed values. When using OCRnA as TOP value in a PWM mode, the OCRnA Register can not be used for generating a PWM output. However, the TOP value will in this case be double buffered allowing the TOP value to be changed in run time. If a fixed TOP value is required, the ICRn Register can be used as an alternative, freeing the OCRnA to be used as PWM output.

### 14.1.2 Definitions

The following definitions are used extensively throughout the document:

| BOTTOM | The counter reaches the BOTTOM when it becomes 0x0000. |
| MAX    | The counter reaches its MAXimum when it becomes 0xFFFF (decimal 65535). |
| TOP    | The counter reaches the TOP when it becomes equal to the highest value in the count sequence. The TOP value can be assigned to be one of the fixed values: 0x00FF, 0x01FF, or 0x03FF, or to the value stored in the OCRnA or ICRn Register. The assignment is dependent of the mode of operation. |

### 14.2 Accessing 16-bit Registers

The TCNTn, OCRnA/B/C, and ICRn are 16-bit registers that can be accessed by the AVR CPU via the 8-bit data bus. The 16-bit register must be byte accessed using two read or write operations. Each 16-bit timer has a single 8-bit register for temporary storing of the high byte of the 16-bit access. The same Temporary Register is shared between all 16-bit registers within each 16-bit timer. Accessing the low byte triggers the 16-bit read or write operation. When the low byte of a 16-bit register is written by the CPU, the high byte stored in the Temporary Register, and the low byte written are both copied into the 16-bit register in the same clock cycle. When the low byte of a 16-bit register is read by the CPU, the high byte of the 16-bit register is copied into the Temporary Register in the same clock cycle as the low byte is read.

Not all 16-bit accesses use the Temporary Register for the high byte. Reading the OCRnA/B/C 16-bit registers does not involve using the Temporary Register.

To do a 16-bit write, the high byte must be written before the low byte. For a 16-bit read, the low byte must be read before the high byte.

The following code examples show how to access the 16-bit timer registers assuming that no interrupts updates the temporary register. The same principle can be used directly for accessing the OCRnA/B/C and ICRn Registers. Note that when using “C”, the compiler handles the 16-bit access.
Assembly Code Examples

```assembly
...;
; Set TCNTn to 0x01FF
ldi r17, 0x01
ldi r16, 0xFF
out TCNTnH, r17
out TCNTnL, r16
; Read TCNTn into r17:r16
in r16, TCNTnL
in r17, TCNTnH
...
```

C Code Examples

```c
unsigned int i;
...
/* Set TCNTn to 0x01FF */
TCNTn = 0x1FF;
/* Read TCNTn into i */
i = TCNTn;
...
```

Note: 1. See “Code Examples” on page 8.

The assembly code example returns the TCNTn value in the r17:r16 register pair.

It is important to notice that accessing 16-bit registers are atomic operations. If an interrupt occurs between the two instructions accessing the 16-bit register, and the interrupt code updates the temporary register by accessing the same or any other of the 16-bit Timer Registers, then the result of the access outside the interrupt will be corrupted. Therefore, when both the main code and the interrupt code update the temporary register, the main code must disable the interrupts during the 16-bit access.

The following code examples show how to do an atomic read of the TCNTn Register contents. Reading any of the OCRnA/B/C or ICRn Registers can be done by using the same principle.
Assembly Code Example

TIM16_ReadTCNTn:
   ; Save global interrupt flag
   in r18,SREG
   ; Disable interrupts
   cli
   ; Read TCNTn into r17:r16
   in r16,TCNTnL
   in r17,TCNTnH
   ; Restore global interrupt flag
   out SREG,r18
   ret

C Code Example

unsigned int TIM16_ReadTCNTn( void )
{
   unsigned char sreg;
   unsigned int i;
   /* Save global interrupt flag */
   sreg = SREG;
   /* Disable interrupts */
   __disable_interrupt();
   /* Read TCNTn into i */
   i = TCNTn;
   /* Restore global interrupt flag */
   SREG = sreg;
   return i;
}

Note: 1. See “Code Examples” on page 8.
The assembly code example returns the TCNTn value in the r17:r16 register pair.
The following code examples show how to do an atomic write of the TCNTn Register contents. Writing any of the OCRnA/B/C or ICRn Registers can be done by using the same principle.

### Assembly Code Example

```
TIM16_WriteTCNTn:
    ; Save global interrupt flag
    in r18, SREG
    ; Disable interrupts
    cli
    ; Set TCNTn to r17:r16
    out TCNTnH, r17
    out TCNTnL, r16
    ; Restore global interrupt flag
    out SREG, r18
    ret
```

### C Code Example

```
void TIM16_WriteTCNTn( unsigned int i )
{
    unsigned char sreg;
    unsigned int i;
    /* Save global interrupt flag */
    sreg = SREG;
    /* Disable interrupts */
    __disable_interrupt();
    /* Set TCNTn to i */
    TCNTn = i;
    /* Restore global interrupt flag */
    SREG = sreg;
}
```

**Note:**

The assembly code example requires that the r17:r16 register pair contains the value to be written to TCNTn.

### 14.2.1 Reusing the Temporary High Byte Register

If writing to more than one 16-bit register where the high byte is the same for all registers written, then the high byte only needs to be written once. However, note that the same rule of atomic operation described previously also applies in this case.

### 14.3 Timer/Counter Clock Sources

The Timer/Counter can be clocked by an internal or an external clock source. The clock source is selected by the Clock Select logic which is controlled by the Clock Select (CSn2:0) bits located in the Timer/Counter control Register B (TCCRB). For details on clock sources and prescaler, see “Timer/Counter0, Timer/Counter1, and Timer/Counter3 Prescalers” on page 89.
14.4 Counter Unit

The main part of the 16-bit Timer/Counter is the programmable 16-bit bi-directional counter unit. **Figure 14-2** shows a block diagram of the counter and its surroundings.

**Figure 14-2.** Counter Unit Block Diagram

![Counter Unit Block Diagram](image)

Signal description (internal signals):

- **Count**: Increment or decrement TCNTn by 1.
- **Direction**: Select between increment and decrement.
- **Clear**: Clear TCNTn (set all bits to zero).
- **clk$_{Tn}$**: Timer/Counter clock.
- **TOP**: Signalize that TCNTn has reached maximum value.
- **BOTTOM**: Signalize that TCNTn has reached minimum value (zero).

The 16-bit counter is mapped into two 8-bit I/O memory locations: **Counter High** (TCNTnH) containing the upper eight bits of the counter, and **Counter Low** (TCNTnL) containing the lower eight bits. The TCNTnH Register can only be indirectly accessed by the CPU. When the CPU does an access to the TCNTnH I/O location, the CPU accesses the high byte temporary register (TEMP). The temporary register is updated with the TCNTnH value when the TCNTnL is read, and TCNTnH is updated with the temporary register value when TCNTnL is written. This allows the CPU to read or write the entire 16-bit counter value within one clock cycle via the 8-bit data bus. It is important to notice that there are special cases of writing to the TCNTn Register when the counter is counting that will give unpredictable results. The special cases are described in the sections where they are of importance.

Depending on the mode of operation used, the counter is cleared, incremented, or decremented at each timer clock (clk$_{Tn}$). The clk$_{Tn}$ can be generated from an external or internal clock source, selected by the **Clock Select** bits (CSn2:0). When no clock source is selected (CSn2:0 = 0) the timer is stopped. However, the TCNTn value can be accessed by the CPU, independent of whether clk$_{Tn}$ is present or not. A CPU write overrides (has priority over) all counter clear or count operations.

The counting sequence is determined by the setting of the **Waveform Generation mode** bits (WGMn3:0) located in the **Timer/Counter Control Registers A and B** (TCCRnA and TCCRnB). There are close connections between how the counter behaves (counts) and how waveforms are generated on the Output Compare outputs OCnx. For more details about advanced counting sequences and waveform generation, see “Modes of Operation” on page 120.
The Timer/Counter Overflow Flag (TOVn) is set according to the mode of operation selected by the WGMn3:0 bits. TOVn can be used for generating a CPU interrupt.

### 14.5 Input Capture Unit

The Timer/Counter incorporates an input capture unit that can capture external events and give them a time-stamp indicating time of occurrence. The external signal indicating an event, or multiple events, can be applied via the ICPn pin or alternatively, for the Timer/Counter1 only, via the Analog Comparator unit. The time-stamps can then be used to calculate frequency, duty-cycle, and other features of the signal applied. Alternatively the time-stamps can be used for creating a log of the events.

The Input Capture unit is illustrated by the block diagram shown in Figure 14-3. The elements of the block diagram that are not directly a part of the input capture unit are gray shaded. The small "n" in register and bit names indicates the Timer/Counter number.

**Figure 14-3.** Input Capture Unit Block Diagram

Note: The Analog Comparator Output (ACO) can only trigger the Timer/Counter1 ICP – not Timer/Counter3, 4 or 5.

When a change of the logic level (an event) occurs on the Input Capture Pin (ICPn), alternatively on the analog Comparator output (ACO), and this change confirms to the setting of the edge detector, a capture will be triggered. When a capture is triggered, the 16-bit value of the counter (TCNTn) is written to the Input Capture Register (ICRn). The Input Capture Flag (ICFn) is set at the same system clock as the TCNTn value is copied into ICRn Register. If enabled (TICIEn = 1), the input capture flag generates an input capture interrupt. The ICFn flag is automatically cleared when the interrupt is executed. Alternatively the ICFn flag can be cleared by software by writing a logical one to its I/O bit location.
Reading the 16-bit value in the Input Capture Register (ICRn) is done by first reading the low byte (ICRnL) and then the high byte (ICRnH). When the low byte is read the high byte is copied into the high byte Temporary Register (TEMP). When the CPU reads the ICRnH I/O location it will access the TEMP Register.

The ICRn Register can only be written when using a Waveform Generation mode that utilizes the ICRn Register for defining the counter’s TOP value. In these cases the Waveform Generation mode (WGMn3:0) bits must be set before the TOP value can be written to the ICRn Register. When writing the ICRn Register the high byte must be written to the ICRnH I/O location before the low byte is written to ICRnL.

For more information on how to access the 16-bit registers refer to “Accessing 16-bit Registers” on page 110.

14.5.1 Input Capture Trigger Source

The main trigger source for the input capture unit is the Input Capture Pin (ICPn). Timer/Counter1 can alternatively use the analog comparator output as trigger source for the input capture unit. The Analog Comparator is selected as trigger source by setting the analog Comparator Input Capture (ACIC) bit in the Analog Comparator Control and Status Register (ACSR). Be aware that changing trigger source can trigger a capture. The input capture flag must therefore be cleared after the change.

Both the Input Capture Pin (ICPn) and the Analog Comparator output (ACO) inputs are sampled using the same technique as for the Tn pin (Figure 12-1 on page 89). The edge detector is also identical. However, when the noise canceler is enabled, additional logic is inserted before the edge detector, which increases the delay by four system clock cycles. Note that the input of the noise canceler and edge detector is always enabled unless the Timer/Counter is set in a Waveform Generation mode that uses ICRn to define TOP.

An input capture can be triggered by software by controlling the port of the ICPn pin.

14.5.2 Noise Canceler

The noise canceler improves noise immunity by using a simple digital filtering scheme. The noise canceler input is monitored over four samples, and all four must be equal for changing the output that in turn is used by the edge detector.

The noise canceler is enabled by setting the Input Capture Noise Canceler (ICNCn) bit in Timer/Counter Control Register B (TCCRBn). When enabled the noise canceler introduces additional four system clock cycles of delay from a change applied to the input, to the update of the ICRn Register. The noise canceler uses the system clock and is therefore not affected by the prescaler.

14.5.3 Using the Input Capture Unit

The main challenge when using the Input Capture unit is to assign enough processor capacity for handling the incoming events. The time between two events is critical. If the processor has not read the captured value in the ICRn Register before the next event occurs, the ICRn will be overwritten with a new value. In this case the result of the capture will be incorrect.

When using the Input Capture interrupt, the ICRn Register should be read as early in the interrupt handler routine as possible. Even though the Input Capture interrupt has relatively high priority, the maximum interrupt response time is dependent on the maximum number of clock cycles it takes to handle any of the other interrupt requests.
Using the Input Capture unit in any mode of operation when the TOP value (resolution) is actively changed during operation, is not recommended.

Measurement of an external signal’s duty cycle requires that the trigger edge is changed after each capture. Changing the edge sensing must be done as early as possible after the ICRn Register has been read. After a change of the edge, the Input Capture Flag (ICFn) must be cleared by software (writing a logical one to the I/O bit location). For measuring frequency only, the clearing of the ICFn Flag is not required (if an interrupt handler is used).

### 14.6 Output Compare Units

The 16-bit comparator continuously compares TCNTn with the Output Compare Register (OCRnx). If TCNT equals OCRnx the comparator signals a match. A match will set the Output Compare Flag (OCFx) at the next timer clock cycle. If enabled (OCIEnx = 1), the Output Compare Flag generates an Output Compare interrupt. The OCFnx Flag is automatically cleared when the interrupt is executed. Alternatively the OCFnx Flag can be cleared by software by writing a logical one to its I/O bit location. The Waveform Generator uses the match signal to generate an output according to operating mode set by the Waveform Generation mode (WGMn3:0) bits and Compare Output mode (COMnx1:0) bits. The TOP and BOTTOM signals are used by the Waveform Generator for handling the special cases of the extreme values in some modes of operation (See “Modes of Operation” on page 120.)

A special feature of Output Compare unit A allows it to define the Timer/Counter TOP value (i.e., counter resolution). In addition to the counter resolution, the TOP value defines the period time for waveforms generated by the Waveform Generator.

**Figure 14-4** shows a block diagram of the Output Compare unit. The small “n” in the register and bit names indicates the device number (n = n for Timer/Counter n), and the “x” indicates Output Compare unit (A/B/C). The elements of the block diagram that are not directly a part of the Output Compare unit are gray shaded.

**Figure 14-4.** Output Compare Unit, Block Diagram
The OCRnx Register is double buffered when using any of the twelve *Pulse Width Modulation* (PWM) modes. For the Normal and *Clear Timer on Compare* (CTC) modes of operation, the double buffering is disabled. The double buffering synchronizes the update of the OCRnx Compare Register to either TOP or BOTTOM of the counting sequence. The synchronization prevents the occurrence of odd-length, non-symmetrical PWM pulses, thereby making the output glitch-free.

The OCRnx Register access may seem complex, but this is not case. When the double buffering is enabled, the CPU has access to the OCRnx Buffer Register, and if double buffering is disabled the CPU will access the OCRnx directly. The content of the OCR1x (Buffer or Compare) Register is only changed by a write operation (the Timer/Counter does not update this register automatically as the TCNT1 and ICR1 Register). Therefore OCR1x is not read via the high byte temporary register (TEMP). However, it is a good practice to read the low byte first as when accessing other 16-bit registers. Writing the OCRnx Registers must be done via the TEMP Register since the compare of all 16 bits is done continuously. The high byte (OCRnxH) has to be written first. When the high byte I/O location is written by the CPU, the TEMP Register will be updated by the value written. Then when the low byte (OCRnxL) is written to the lower eight bits, the high byte will be copied into the upper 8-bits of either the OCRnx buffer or OCRnx Compare Register in the same system clock cycle.

For more information of how to access the 16-bit registers refer to “Accessing 16-bit Registers” on page 110.

### 14.6.1 Force Output Compare

In non-PWM Waveform Generation modes, the match output of the comparator can be forced by writing a one to the *Force Output Compare* (FOCnx) bit. Forcing compare match will not set the OCFnx Flag or reload/clear the timer, but the OCnx pin will be updated as if a real compare match had occurred (the COMn1:0 bits settings define whether the OCnx pin is set, cleared or toggled).

### 14.6.2 Compare Match Blocking by TCNTn Write

All CPU writes to the TCNTn Register will block any compare match that occurs in the next timer clock cycle, even when the timer is stopped. This feature allows OCRnx to be initialized to the same value as TCNTn without triggering an interrupt when the Timer/Counter clock is enabled.

### 14.6.3 Using the Output Compare Unit

Since writing TCNTn in any mode of operation will block all compare matches for one timer clock cycle, there are risks involved when changing TCNTn when using any of the Output Compare channels, independent of whether the Timer/Counter is running or not. If the value written to TCNTn equals the OCRnx value, the compare match will be missed, resulting in incorrect waveform generation. Do not write the TCNTn equal to TOP in PWM modes with variable TOP values. The compare match for the TOP will be ignored and the counter will continue to 0xFFFF. Similarly, do not write the TCNTn value equal to BOTTOM when the counter is downcounting.

The setup of the OCnx should be performed before setting the Data Direction Register for the port pin to output. The easiest way of setting the OCnx value is to use the Force Output Compare (FOCnx) strobe bits in Normal mode. The OCnx Register keeps its value even when changing between Waveform Generation modes.

Be aware that the COMnx1:0 bits are not double buffered together with the compare value. Changing the COMnx1:0 bits will take effect immediately.
14.7 Compare Match Output Unit

The Compare Output mode (COMnx1:0) bits have two functions. The Waveform Generator uses the COMnx1:0 bits for defining the Output Compare (OCnx) state at the next compare match. Secondly the COMnx1:0 bits control the OCnx pin output source. Figure 14-5 shows a simplified schematic of the logic affected by the COMnx1:0 bit setting. The I/O Registers, I/O bits, and I/O pins in the figure are shown in bold. Only the parts of the general I/O Port Control Registers (DDR and PORT) that are affected by the COMnx1:0 bits are shown. When referring to the OCnx state, the reference is for the internal OCnx Register, not the OCnx pin. If a system reset occur, the OCnx Register is reset to “0”.

Figure 14-5. Compare Match Output Unit, Schematic

The general I/O port function is overridden by the Output Compare (OCnx) from the Waveform Generator if either of the COMnx1:0 bits are set. However, the OCnx pin direction (input or output) is still controlled by the Data Direction Register (DDR) for the port pin. The Data Direction Register bit for the OCnx pin (DDR_OCnx) must be set as output before the OCnx value is visible on the pin. The port override function is generally independent of the Waveform Generation mode, but there are some exceptions. Refer to Table 14-2, Table 14-3 and Table 14-4 for details.

The design of the Output Compare pin logic allows initialization of the OCnx state before the output is enabled. Note that some COMnx1:0 bit settings are reserved for certain modes of operation. See “16-bit Timers/Counters (Timer/Counter1 and Timer/Counter3)” on page 108.

The COMnx1:0 bits have no effect on the Input Capture unit.

14.7.1 Compare Output Mode and Waveform Generation

The Waveform Generator uses the COMnx1:0 bits differently in normal, CTC, and PWM modes. For all modes, setting the COMnx1:0 = 0 tells the Waveform Generator that no action on the OCnx Register is to be performed on the next compare match. For compare output actions in the
non-PWM modes refer to Table 14-2 on page 130. For fast PWM mode refer to Table 14-3 on page 130, and for phase correct and phase and frequency correct PWM refer to Table 14-4 on page 131.

A change of the COMnx1:0 bits state will have effect at the first compare match after the bits are written. For non-PWM modes, the action can be forced to have immediate effect by using the FOCnx strobe bits.

14.8 Modes of Operation

The mode of operation, i.e., the behavior of the Timer/Counter and the Output Compare pins, is defined by the combination of the Waveform Generation mode (WGMn3:0) and Compare Output mode (COMnx1:0) bits. The Compare Output mode bits do not affect the counting sequence, while the Waveform Generation mode bits do. The COMnx1:0 bits control whether the PWM output generated should be inverted or not (inverted or non-inverted PWM). For non-PWM modes the COMnx1:0 bits control whether the output should be set, cleared or toggle at a compare match (See “Compare Match Output Unit” on page 119.)

For detailed timing information refer to “Timer/Counter Timing Diagrams” on page 127.

14.8.1 Normal Mode

The simplest mode of operation is the Normal mode (WGMn3:0 = 0). In this mode the counting direction is always up (incrementing), and no counter clear is performed. The counter simply overruns when it passes its maximum 16-bit value (MAX = 0xFFFF) and then restarts from the BOTTOM (0x0000). In normal operation the Timer/Counter Overflow Flag (TOVn) will be set in the same timer clock cycle as the TCNTn becomes zero. The TOVn Flag in this case behaves like a 17th bit, except that it is only set, not cleared. However, combined with the timer overflow interrupt that automatically clears the TOVn Flag, the timer resolution can be increased by software. There are no special cases to consider in the Normal mode, a new counter value can be written anytime.

The Input Capture unit is easy to use in Normal mode. However, observe that the maximum interval between the external events must not exceed the resolution of the counter. If the interval between events are too long, the timer overflow interrupt or the prescaler must be used to extend the resolution for the capture unit.

The Output Compare units can be used to generate interrupts at some given time. Using the Output Compare to generate waveforms in Normal mode is not recommended, since this will occupy too much of the CPU time.

14.8.2 Clear Timer on Compare Match (CTC) Mode

In Clear Timer on Compare or CTC mode (WGMn3:0 = 4 or 12), the OCRnA or ICRn Register are used to manipulate the counter resolution. In CTC mode the counter is cleared to zero when the counter value (TCNTn) matches either the OCRnA (WGMn3:0 = 4) or the ICRn (WGMn3:0 = 12). The OCRnA or ICRn define the top value for the counter, hence also its resolution. This mode allows greater control of the compare match output frequency. It also simplifies the operation of counting external events.

The timing diagram for the CTC mode is shown in Figure 14-6. The counter value (TCNTn) increases until a compare match occurs with either OCRnA or ICRn, and then counter (TCNTn) is cleared.
Figure 14-6. CTC Mode, Timing Diagram

An interrupt can be generated at each time the counter value reaches the TOP value by either using the OCFnA or ICFn Flag according to the register used to define the TOP value. If the interrupt is enabled, the interrupt handler routine can be used for updating the TOP value. However, changing the TOP to a value close to BOTTOM when the counter is running with none or a low prescaler value must be done with care since the CTC mode does not have the double buffering feature. If the new value written to OCRnA or ICRn is lower than the current value of TCNTn, the counter will miss the compare match. The counter will then have to count to its maximum value (0xFFFF) and wrap around starting at 0x0000 before the compare match can occur. In many cases this feature is not desirable. An alternative will then be to use the fast PWM mode using OCRnA for defining TOP (WGMn3:0 = 15) since the OCRnA then will be double buffered.

For generating a waveform output in CTC mode, the OCNa output can be set to toggle its logical level on each compare match by setting the Compare Output mode bits to toggle mode (COMnA1:0 = 1). The OCNa value will not be visible on the port pin unless the data direction for the pin is set to output (DDR_OCnA = 1). The waveform generated will have a maximum frequency of \( f_{OCnA} = f_{clk_{1/O}} / 2 \) when OCRnA is set to zero (0x0000). The waveform frequency is defined by the following equation:

\[
 f_{OCnA} = \frac{f_{clk_{1/O}}}{2 \cdot N \cdot (1 + OCRnA)}
\]

The \( N \) variable represents the prescaler factor (1, 8, 64, 256, or 1024).

As for the Normal mode of operation, the TOVn Flag is set in the same timer clock cycle that the counter counts from MAX to 0x0000.

14.8.3 Fast PWM Mode

The fast Pulse Width Modulation or fast PWM mode (WGMn3:0 = 5, 6, 7, 14, or 15) provides a high frequency PWM waveform generation option. The fast PWM differs from the other PWM options by its single-slope operation. The counter counts from BOTTOM to TOP then restarts from BOTTOM. In non-inverting Compare Output mode, the Output Compare (OCnx) is set on the compare match between TCNTn and OCRnx, and cleared at TOP. In inverting Compare Output mode output is cleared on compare match and set at TOP. Due to the single-slope operation, the operating frequency of the fast PWM mode can be twice as high as the phase correct and phase and frequency correct PWM modes that use dual-slope operation. This high frequency makes the fast PWM mode well suited for power regulation, rectification, and DAC applications. High frequency allows physically small sized external components (coils, capacitors), hence reduces total system cost.
The PWM resolution for fast PWM can be fixed to 8-, 9-, or 10-bit, or defined by either ICRn or OCRnA. The minimum resolution allowed is 2-bit (ICRn or OCRnA set to 0x0000), and the maximum resolution is 16-bit (ICRn or OCRnA set to MAX). The PWM resolution in bits can be calculated by using the following equation:

\[ R_{FPWM} = \log_2 \left( \frac{TOP + 1}{2} \right) \]

In fast PWM mode the counter is incremented until the counter value matches either one of the fixed values 0x00FF, 0x01FF, or 0x03FF (WGMn3:0 = 5, 6, or 7), the value in ICRn (WGMn3:0 = 14), or the value in OCRnA (WGMn3:0 = 15). The counter is then cleared at the following timer clock cycle. The timing diagram for the fast PWM mode is shown in Figure 14-7. The figure shows fast PWM mode when OCRnA or ICRn is used to define TOP. The TCNTn value is in the timing diagram shown as a histogram for illustrating the single-slope operation. The diagram includes non-inverted and inverted PWM outputs. The small horizontal line marks on the TCNTn slopes represent compare matches between OCRn and TCNTn. The OCnx Interrupt Flag will be set when a compare match occurs.

**Figure 14-7. Fast PWM Mode, Timing Diagram**

The Timer/Counter Overflow Flag (TOVn) is set each time the counter reaches TOP. In addition the OCRnA or ICFn Flag is set at the same timer clock cycle as TOVn is set when either OCRnA or ICRn is used for defining the TOP value. If one of the interrupts are enabled, the interrupt handler routine can be used for updating the TOP and compare values.

When changing the TOP value the program must ensure that the new TOP value is higher or equal to the value of all of the Compare Registers. If the TOP value is lower than any of the Compare Registers, a compare match will never occur between the TCNTn and the OCRnx. Note that when using fixed TOP values the unused bits are masked to zero when any of the OCRnx Registers are written.

The procedure for updating ICRn differs from updating OCRnA when used for defining the TOP value. The ICRn Register is not double buffered. This means that if ICRn is changed to a low value when the counter is running with none or a low prescaler value, there is a risk that the new ICRn value written is lower than the current value of TCNTn. The result will then be that the counter will miss the compare match at the TOP value. The counter will then have to count to the MAX value (0xFFFF) and wrap around starting at 0x0000 before the compare match can occur. The OCRnA Register however, is double buffered. This feature allows the OCRnA I/O location
to be written anytime. When the OCRnA I/O location is written the value written will be put into the OCRnA Buffer Register. The OCRnA Compare Register will then be updated with the value in the Buffer Register at the next timer clock cycle the TCNTn matches TOP. The update is done at the same timer clock cycle as the TCNTn is cleared and the TOVn Flag is set.

Using the ICRn Register for defining TOP works well when using fixed TOP values. By using ICRn, the OCRnA Register is free to be used for generating a PWM output on OCnA. However, if the base PWM frequency is actively changed (by changing the TOP value), using the OCRnA as TOP is clearly a better choice due to its double buffer feature.

In fast PWM mode, the compare units allow generation of PWM waveforms on the OCnx pins. Setting the COMnx1:0 bits to two will produce a non-inverted PWM and an inverted PWM output can be generated by setting the COMnx1:0 to three (see Table on page 130). The actual OCnx value will only be visible on the port pin if the data direction for the port pin is set as output (DDR_OCnx). The PWM waveform is generated by setting (or clearing) the OCnx Register at the compare match between OCRnx and TCNTn, and clearing (or setting) the OCnx Register at the timer clock cycle the counter is cleared (changes from TOP to BOTTOM).

The PWM frequency for the output can be calculated by the following equation:

\[ f_{OCnXPWM} = \frac{f_{clk\ I/O}}{N \cdot (1 + \text{TOP})} \]

The N variable represents the prescaler divider (1, 8, 64, 256, or 1024).

The extreme values for the OCRnx Register represents special cases when generating a PWM waveform output in the fast PWM mode. If the OCRnx is set equal to BOTTOM (0x0000) the output will be a narrow spike for each TOP+1 timer clock cycle. Setting the OCRnx equal to TOP will result in a constant high or low output (depending on the polarity of the output set by the COMnx1:0 bits.)

A frequency (with 50% duty cycle) waveform output in fast PWM mode can be achieved by setting OCnA to toggle its logical level on each compare match (COMnA1:0 = 1). This applies only if OCR1A is used to define the TOP value (WGM13:0 = 15). The waveform generated will have a maximum frequency of \( f_{OCnA} = f_{clk\ I/O}/2 \) when OCRnA is set to zero (0x0000). This feature is similar to the OCnA toggle in CTC mode, except the double buffer feature of the Output Compare unit is enabled in the fast PWM mode.

14.8.4 Phase Correct PWM Mode

The phase correct Pulse Width Modulation or phase correct PWM mode (WGMn3:0 = 1, 2, 3, 10, or 11) provides a high resolution phase correct PWM waveform generation option. The phase correct PWM mode is, like the phase and frequency correct PWM mode, based on a dual-slope operation. The counter counts repeatedly from BOTTOM (0x0000) to TOP and then from TOP to BOTTOM. In non-inverting Compare Output mode, the Output Compare (OCnx) is cleared on the compare match between TCNTn and OCRnx while upcounting, and set on the compare match while downcounting. In inverting Output Compare mode, the operation is inverted. The dual-slope operation has lower maximum operation frequency than single slope operation. However, due to the symmetric feature of the dual-slope PWM modes, these modes are preferred for motor control applications.

The PWM resolution for the phase correct PWM mode can be fixed to 8-, 9-, or 10-bit, or defined by either ICRn or OCRnA. The minimum resolution allowed is 2-bit (ICRn or OCRnA set to
0x0003), and the maximum resolution is 16-bit (ICRn or OCRnA set to MAX). The PWM resolution in bits can be calculated by using the following equation:

\[ R_{PCPWM} = \frac{\log(TOP + 1)}{\log(2)} \]

In phase correct PWM mode the counter is incremented until the counter value matches either one of the fixed values 0x00FF, 0x01FF, or 0x03FF (WGMn3:0 = 1, 2, or 3), the value in ICRn (WGMn3:0 = 10), or the value in OCRnA (WGMn3:0 = 11). The counter has then reached the TOP and changes the count direction. The TCNTn value will be equal to TOP for one timer clock cycle. The timing diagram for the phase correct PWM mode is shown on Figure 14-8. The figure shows phase correct PWM mode when OCRnA or ICRn is used to define TOP. The TCNTn value is in the timing diagram shown as a histogram for illustrating the dual-slope operation. The diagram includes non-inverted and inverted PWM outputs. The small horizontal line marks on the TCNTn slopes represent compare matches between OCRnx and TCNTn. The OCRnx Interrupt Flag will be set when a compare match occurs.

**Figure 14-8. Phase Correct PWM Mode, Timing Diagram**

The Timer/Counter Overflow Flag (TOVn) is set each time the counter reaches BOTTOM. When either OCRnA or ICRn is used for defining the TOP value, the OCRnA or ICFn Flag is set accordingly at the same timer clock cycle as the OCRnx Registers are updated with the double buffer value (at TOP). The Interrupt Flags can be used to generate an interrupt each time the counter reaches the TOP or BOTTOM value.

When changing the TOP value the program must ensure that the new TOP value is higher or equal to the value of all of the Compare Registers. If the TOP value is lower than any of the Compare Registers, a compare match will never occur between the TCNTn and the OCRnx. Note that when using fixed TOP values, the unused bits are masked to zero when any of the OCRnx Registers are written. As the third period shown in Figure 14-8 illustrates, changing the TOP actively while the Timer/Counter is running in the phase correct mode can result in an unsymmetrical output. The reason for this can be found in the time of update of the OCRnx Reg-
ister. Since the OCRnx update occurs at TOP, the PWM period starts and ends at TOP. This implies that the length of the falling slope is determined by the previous TOP value, while the length of the rising slope is determined by the new TOP value. When these two values differ the two slopes of the period will differ in length. The difference in length gives the unsymmetrical result on the output.

It is recommended to use the phase and frequency correct mode instead of the phase correct mode when changing the TOP value while the Timer/Counter is running. When using a static TOP value there are practically no differences between the two modes of operation.

In phase correct PWM mode, the compare units allow generation of PWM waveforms on the OCnx pins. Setting the COMnx1:0 bits to two will produce a non-inverted PWM and an inverted PWM output can be generated by setting the COMnx1:0 to three (See Table 14-4 on page 131). The actual OCnx value will only be visible on the port pin if the data direction for the port pin is set as output (DDR_OCnx). The PWM waveform is generated by setting (or clearing) the OCnx Register at the compare match between OCRnx and TCNTn when the counter increments, and clearing (or setting) the OCnx Register at compare match between OCRnx and TCNTn when the counter decrements. The PWM frequency for the output when using phase correct PWM can be calculated by the following equation:

\[ f_{OCnxPWM} = \frac{f_{clk\cdot V/O}}{2 \cdot N \cdot TOP} \]

The N variable represents the prescaler divider (1, 8, 64, 256, or 1024).

The extreme values for the OCRnx Register represent special cases when generating a PWM waveform output in the phase correct PWM mode. If the OCRnx is set equal to BOTTOM the output will be continuously low and if set equal to TOP the output will be continuously high for non-inverted PWM mode. For inverted PWM the output will have the opposite logic values. If OCR1A is used to define the TOP value (WGM13:0 = 11) and COM1A1:0 = 1, the OC1A output will toggle with a 50% duty cycle.

### 14.8.5 Phase and Frequency Correct PWM Mode

The phase and frequency correct Pulse Width Modulation, or phase and frequency correct PWM mode (WGMn3:0 = 8 or 9) provides a high resolution phase and frequency correct PWM waveform generation option. The phase and frequency correct PWM mode is, like the phase correct PWM mode, based on a dual-slope operation. The counter counts repeatedly from BOTTOM (0x0000) to TOP and then from TOP to BOTTOM. In non-inverting Compare Output mode, the Output Compare (OCnx) is cleared on the compare match between TCNTn and OCRnx while upcounting, and set on the compare match while downcounting. In inverting Compare Output mode, the operation is inverted. The dual-slope operation gives a lower maximum operation frequency compared to the single-slope operation. However, due to the symmetric feature of the dual-slope PWM modes, these modes are preferred for motor control applications.

The main difference between the phase correct, and the phase and frequency correct PWM mode is the time the OCRnx Register is updated by the OCRnx Buffer Register, (see Figure 14-8 and Figure 14-9).

The PWM resolution for the phase and frequency correct PWM mode can be defined by either ICRn or OCRnA. The minimum resolution allowed is 2-bit (ICRn or OCRnA set to 0x0003), and
the maximum resolution is 16-bit (ICRn or OCRnA set to MAX). The PWM resolution in bits can be calculated using the following equation:

\[ R_{PFCPWM} = \frac{\log(TOP + 1)}{\log(2)} \]

In phase and frequency correct PWM mode the counter is incremented until the counter value matches either the value in ICRn (WGMn3:0 = 8), or the value in OCRnA (WGMn3:0 = 9). The counter has then reached the TOP and changes the count direction. The TCNTn value will be equal to TOP for one timer clock cycle. The timing diagram for the phase correct and frequency correct PWM mode is shown on Figure 14-9. The figure shows phase and frequency correct PWM mode when OCRnA or ICRn is used to define TOP. The TCNTn value is in the timing diagram shown as a histogram for illustrating the dual-slope operation. The diagram includes non-inverted and inverted PWM outputs. The small horizontal line marks on the TCNTn slopes represent compare matches between OCRnx and TCNTn. The OCnx Interrupt Flag will be set when a compare match occurs.

Figure 14-9. Phase and Frequency Correct PWM Mode, Timing Diagram

The Timer/Counter Overflow Flag (TOVn) is set at the same timer clock cycle as the OCRnx Registers are updated with the double buffer value (at BOTTOM). When either OCRnA or ICRn is used for defining the TOP value, the OCnA or ICFn Flag set when TCNTn has reached TOP. The Interrupt Flags can then be used to generate an interrupt each time the counter reaches the TOP or BOTTOM value.

When changing the TOP value the program must ensure that the new TOP value is higher or equal to the value of all of the Compare Registers. If the TOP value is lower than any of the Compare Registers, a compare match will never occur between the TCNTn and the OCRnx.

As Figure 14-9 shows the output generated is, in contrast to the phase correct mode, symmetrical in all periods. Since the OCRnx Registers are updated at BOTTOM, the length of the rising and the falling slopes will always be equal. This gives symmetrical output pulses and is therefore frequency correct.
Using the ICRn Register for defining TOP works well when using fixed TOP values. By using ICRn, the OCRnA Register is free to be used for generating a PWM output on OCnA. However, if the base PWM frequency is actively changed by changing the TOP value, using the OCRnA as TOP is clearly a better choice due to its double buffer feature.

In phase and frequency correct PWM mode, the compare units allow generation of PWM waveforms on the OCnx pins. Setting the COMnx1:0 bits to two will produce a non-inverted PWM and an inverted PWM output can be generated by setting the COMnx1:0 to three (See Table 14-4 on page 131). The actual OCnx value will only be visible on the port pin if the data direction for the port pin is set as output (DDR,OCnx). The PWM waveform is generated by setting (or clearing) the OCnx Register at the compare match between OCRnx and TCNTn when the counter increments, and clearing (or setting) the OCnx Register at compare match between OCRnx and TCNTn when the counter decrements. The PWM frequency for the output when using phase and frequency correct PWM can be calculated by the following equation:

\[ f_{OCnxPWM} = \frac{f_{clk\ VO}}{2 \cdot N \cdot TOP} \]

The N variable represents the prescaler divider (1, 8, 64, 256, or 1024).

The extreme values for the OCRnx Register represents special cases when generating a PWM waveform output in the phase correct PWM mode. If the OCRnx is set equal to BOTTOM the output will be continuously low and if set equal to TOP the output will be set to high for non-inverted PWM mode. For inverted PWM the output will have the opposite logic values. If OCR1A is used to define the TOP value (WGM13:0 = 9) and COM1A1:0 = 1, the OC1A output will toggle with a 50% duty cycle.

### 14.9 Timer/Counter Timing Diagrams

The Timer/Counter is a synchronous design and the timer clock (clk\_TCn) is therefore shown as a clock enable signal in the following figures. The figures include information on when Interrupt Flags are set, and when the OCRnx Register is updated with the OCRnx buffer value (only for modes utilizing double buffering). Figure 14-10 shows a timing diagram for the setting of OCFnx.

**Figure 14-10.** Timer/Counter Timing Diagram, Setting of OCFnx, no Prescaling

```
clkn/0
clkn/1 (clk\_TCn/1)
```

<table>
<thead>
<tr>
<th>TCNTn</th>
<th>OCRnx - 1</th>
<th>OCRnx</th>
<th>OCRnx + 1</th>
<th>OCRnx + 2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Figure 14-11** shows the same timing data, but with the prescaler enabled.
**Figure 14-11.** Timer/Counter Timing Diagram, Setting of OCFnx, with Prescaler ($f_{clk_{IO}}/8$)

**Figure 14-12** shows the count sequence close to TOP in various modes. When using phase and frequency correct PWM mode the OCRnx Register is updated at BOTTOM. The timing diagrams will be the same, but TOP should be replaced by BOTTOM, TOP-1 by BOTTOM+1 and so on. The same renaming applies for modes that set the TOVn Flag at BOTTOM.

**Figure 14-12.** Timer/Counter Timing Diagram, no Prescaling

**Figure 14-13** shows the same timing data, but with the prescaler enabled.
14.10 16-bit Timer/Counter Register Description

14.10.1 Timer/Counter1 Control Register A – TCCR1A

<table>
<thead>
<tr>
<th>Bit</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read/Write</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
</tr>
<tr>
<td>Initial Value</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

14.10.2 Timer/Counter3 Control Register A – TCCR3A

<table>
<thead>
<tr>
<th>Bit</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read/Write</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
</tr>
<tr>
<td>Initial Value</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

- Bit 7:6 – COMnA1:0: Compare Output Mode for Channel A
- Bit 5:4 – COMnB1:0: Compare Output Mode for Channel B
- Bit 3:2 – COMnC1:0: Compare Output Mode for Channel C

The COMnA1:0, COMnB1:0, and COMnC1:0 control the output compare pins (OCnA, OCnB, and OCnC respectively) behavior. If one or both of the COMnA1:0 bits are written to one, the OCnA output overrides the normal port functionality of the I/O pin it is connected to. If one or both of the COMnB1:0 bits are written to one, the OCnB output overrides the normal port functionality of the I/O pin it is connected to. If one or both of the COMnC1:0 bits are written to one, the OCnC output overrides the normal port functionality of the I/O pin it is connected to. However, note that the Data Direction Register (DDR) bit corresponding to the OCnA, OCnB or OCnC pin must be set in order to enable the output driver.
When the OCnA, OCnB or OCnC is connected to the pin, the function of the COMn:0 bits is dependent of the WGMn:0:3 bits setting. **Table 14-2** shows the COMn:0 bit functionality when the WGMn:0:3 bits are set to a normal or a CTC mode (non-PWM).

**Table 14-2.** Compare Output Mode, non-PWM

<table>
<thead>
<tr>
<th>COMnA1/COMnB1/COMnC1</th>
<th>COMnA0/COMnB0/COMnC0</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>Normal port operation, OCnA/OCnB/OCnC disconnected.</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>Toggle OCnA/OCnB/OCnC on compare match.</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>Clear OCnA/OCnB/OCnC on compare match (set output to low level).</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>Set OCnA/OCnB/OCnC on compare match (set output to high level).</td>
</tr>
</tbody>
</table>

**Table 14-3** shows the COMn:0 bit functionality when the WGMn:0:3 bits are set to the fast PWM mode.

**Table 14-3.** Compare Output Mode, Fast PWM

<table>
<thead>
<tr>
<th>COMnA1/COMnB1/COMnC0</th>
<th>COMnA0/COMnB0/COMnC0</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>Normal port operation, OCnA/OCnB/OCnC disconnected.</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>WGM13:0 = 14 or 15: Toggle OC1A on Compare Match, OC1B and OC1C disconnected (normal port operation). For all other WGM1 settings, normal port operation, OC1A/OC1B/OC1C disconnected.</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>Clear OCnA/OCnB/OCnC on compare match, set OCnA/OCnB/OCnC at TOP</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>Set OCnA/OCnB/OCnC on compare match, clear OCnA/OCnB/OCnC at TOP</td>
</tr>
</tbody>
</table>

**Note:** A special case occurs when OCRnA/OCRnB/OCRnC equals TOP and COMnA1/COMnB1/COMnC1 is set. In this case the compare match is ignored, but the set or clear is done at TOP. See “Fast PWM Mode” on page 97 for more details.

**Table 14-4** shows the COMn:0 bit functionality when the WGMn:0:3 bits are set to the phase correct and frequency correct PWM mode.
### Table 14-4. Compare Output Mode, Phase Correct and Phase and Frequency Correct PWM

<table>
<thead>
<tr>
<th>COMnA1/COMnB/COMnC1</th>
<th>COMnA0/COMnB0/COMnC0</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>Normal port operation, OCnA/OCnB/OCnC disconnected.</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>WGM13:0 = 8, 9 10 or 11: Toggle OC1A on Compare Match, OC1B and OC1C disconnected (normal port operation). For all other WGM1 settings, normal port operation, OC1A/OC1B/OC1C disconnected.</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>Clear OCnA/OCnB/OCnC on compare match when up-counting. Set OCnA/OCnB/OCnC on compare match when downcounting.</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>Set OCnA/OCnB/OCnC on compare match when up-counting. Clear OCnA/OCnB/OCnC on compare match when downcounting.</td>
</tr>
</tbody>
</table>

Note: A special case occurs when OCRnA/OCRnB/OCRnC equals TOP and COMnA1/COMnB1/COMnC1 is set. See “Phase Correct PWM Mode” on page 99, for more details.

- **Bit 1:0 – WGMn1:0: Waveform Generation Mode**
  Combined with the WGMn3:2 bits found in the TCCRnB Register, these bits control the counting sequence of the counter, the source for maximum (TOP) counter value, and what type of waveform generation to be used, see Table 14-5. Modes of operation supported by the Timer/Counter unit are: Normal mode (counter), Clear Timer on Compare match (CTC) mode, and three types of Pulse Width Modulation (PWM) modes. (See “Modes of Operation” on page 96.).

### Table 14-5. Waveform Generation Mode Bit Description (1)

<table>
<thead>
<tr>
<th>Mode</th>
<th>WGMn3</th>
<th>WGMn2 (CTCn)</th>
<th>WGMn1 (PWMn1)</th>
<th>WGMn0 (PWMn0)</th>
<th>Timer/Counter Mode of Operation</th>
<th>TOP</th>
<th>Update of OCRnx at</th>
<th>TOVnFlag Set on</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>Normal</td>
<td>0xFF</td>
<td>Immediate MAX</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>PWM, Phase Correct, 8-bit</td>
<td>0x00F</td>
<td>TOP BOTTOM</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>PWM, Phase Correct, 9-bit</td>
<td>0x01F</td>
<td>TOP BOTTOM</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>PWM, Phase Correct, 10-bit</td>
<td>0x03F</td>
<td>TOP BOTTOM</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>CTC</td>
<td>OCRnA</td>
<td>Immediate MAX</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>Fast PWM, 8-bit</td>
<td>0x00F</td>
<td>TOP</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>Fast PWM, 9-bit</td>
<td>0x01F</td>
<td>TOP</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>Fast PWM, 10-bit</td>
<td>0x03F</td>
<td>TOP</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>PWM, Phase and Frequency Correct</td>
<td>OCRnA</td>
<td>BOTTOM</td>
<td>BOTTOM</td>
</tr>
<tr>
<td>9</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>PWM, Phase and Frequency Correct</td>
<td>OCRnA</td>
<td>BOTTOM</td>
<td>BOTTOM</td>
</tr>
<tr>
<td>10</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>PWM, Phase Correct</td>
<td>OCRnA</td>
<td>TOP</td>
<td>BOTTOM</td>
</tr>
<tr>
<td>11</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>PWM, Phase Correct</td>
<td>OCRnA</td>
<td>TOP</td>
<td>BOTTOM</td>
</tr>
<tr>
<td>12</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>CTC</td>
<td>OCRnA</td>
<td>Immediate MAX</td>
<td></td>
</tr>
</tbody>
</table>
Table 14-5. Waveform Generation Mode Bit Description (Continued)\(^{(1)}\)

<table>
<thead>
<tr>
<th>Mode</th>
<th>WGMn3</th>
<th>WGMn2 (CTCn)</th>
<th>WGMn1 (PWMn1)</th>
<th>WGMn0 (PWMn0)</th>
<th>Timer/Counter Mode of Operation</th>
<th>TOP</th>
<th>Update of OCRnx at</th>
<th>TOVn Flag Set on</th>
</tr>
</thead>
<tbody>
<tr>
<td>13</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>(Reserved)</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>14</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>Fast PWM</td>
<td>ICRn</td>
<td>TOP</td>
<td>TOP</td>
</tr>
<tr>
<td>15</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>Fast PWM</td>
<td>OCRnA</td>
<td>TOP</td>
<td>TOP</td>
</tr>
</tbody>
</table>

Note: 1. The CTCn and PWMn1:0 bit definition names are obsolete. Use the WGMn2:0 definitions. However, the functionality and location of these bits are compatible with previous versions of the timer.

14.10.3 Timer/Counter1 Control Register B – TCCR1B

<table>
<thead>
<tr>
<th>Bit</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
<th>TCCR1B</th>
</tr>
</thead>
<tbody>
<tr>
<td>R/W</td>
<td>R/W</td>
<td>R</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td></td>
</tr>
</tbody>
</table>

Initial Value: 0 0 0 0 0 0 0 0

- **Bit 7 – ICNC1**: Input Capture Noise Canceler
  Setting this bit (to one) activates the Input Capture Noise Canceler. When the Noise Canceler is activated, the input from the Input Capture Pin (ICPn) is filtered. The filter function requires four successive equal valued samples of the ICPn pin for changing its output. The input capture is therefore delayed by four Oscillator cycles when the noise canceler is enabled.

- **Bit 6 – ICES1**: Input Capture Edge Select
  This bit selects which edge on the Input Capture Pin (ICPn) that is used to trigger a capture event. When the ICES1 bit is written to zero, a falling (negative) edge is used as trigger, and when the ICES1 bit is written to one, a rising (positive) edge will trigger the capture.

  When a capture is triggered according to the ICESn setting, the counter value is copied into the Input Capture Register (ICRn). The event will also set the Input Capture Flag (ICFn), and this can be used to cause an Input Capture Interrupt, if this interrupt is enabled.

  When the ICRn is used as TOP value (see description of the WGMn3:0 bits located in the TCCRnA and the TCCRnB Register), the ICPn is disconnected and consequently the input capture function is disabled.

- **Bit 5 – Reserved Bit**
  This bit is reserved for future use. For ensuring compatibility with future devices, this bit must be written to zero when TCCRnB is written.

- **Bit 4:3 – WGMn3:2**: Waveform Generation Mode
  See TCCRnA Register description.

- **Bit 2:0 – CSn2:0**: Clock Select
  The three clock select bits select the clock source to be used by the Timer/Counter, see Figure 13-8 and Figure 13-9.
### Table 14-6. Clock Select Bit Description

<table>
<thead>
<tr>
<th>CSn2</th>
<th>CSn1</th>
<th>CSn0</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>No clock source. (Timer/Counter stopped)</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>clk( \nu )/1 (No prescaling)</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>clk( \nu )/8 (From prescaler)</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>clk( \nu )/64 (From prescaler)</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>clk( \nu )/256 (From prescaler)</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>clk( \nu )/1024 (From prescaler)</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>External clock source on Tn pin. Clock on falling edge</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>External clock source on Tn pin. Clock on rising edge</td>
</tr>
</tbody>
</table>

If external pin modes are used for the Timer/Countern, transitions on the Tn pin will clock the counter even if the pin is configured as an output. This feature allows software control of the counting.

### 14.10.5 Timer/Counter1 Control Register C – TCCR1C

<table>
<thead>
<tr>
<th>Bit</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
<th>TCCR1C</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>W</td>
<td>W</td>
<td>W</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td></td>
</tr>
</tbody>
</table>

**Initial Value**

0 0 0 0 0 0 0 0 0

### 14.10.6 Timer/Counter3 Control Register C – TCCR3C

<table>
<thead>
<tr>
<th>Bit</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
<th>TCCR3C</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>W</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td></td>
</tr>
</tbody>
</table>

**Initial Value**

0 0 0 0 0 0 0 0 0

- **Bit 7 – FOCnA: Force Output Compare for Channel A**

  The FOCnA/FOCnB/FOCnC bits are only active when the WGMn3:0 bits specifies a non-PWM mode. When writing a logical one to the FOCnA/FOCnB/FOCnC bit, an immediate compare match is forced on the waveform generation unit. The OCnA/OCnB/OCnC output is changed according to its COMn1:0 bits setting. Note that the FOCnA/FOCnB/FOCnC bits are implemented as strobos. Therefore it is the value present in the COMn1:0 bits that determine the effect of the forced compare.

  A FOCnA/FOCnB/FOCnC strobe will not generate any interrupt nor will it clear the timer in Clear Timer on Compare Match (CTC) mode using OCRnA as TOP.

  The FOCnA/FOCnB/FOCnB bits are always read as zero.

- **Bit 4:0 – Reserved Bits**

  These bits are reserved for future use. For ensuring compatibility with future devices, these bits must be written to zero when TCCRnC is written.
### 14.10.7 Timer/Counter1 – TCNT1H and TCNT1L

<table>
<thead>
<tr>
<th>Bit</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>TCNT1H</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
</tr>
<tr>
<td>TCNT1L</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
</tr>
</tbody>
</table>

**Read/Write**: R/W

**Initial Value**: 0 0 0 0 0 0 0 0

The two Timer/Counter I/O locations (TCNTnH and TCNTnL, combined TCNTn) give direct access, both for read and for write operations, to the Timer/Counter unit 16-bit counter. To ensure that both the high and low bytes are read and written simultaneously when the CPU accesses these registers, the access is performed using an 8-bit temporary High Byte Register (TEMP). This temporary register is shared by all the other 16-bit registers. See “Accessing 16-bit Registers” on page 110.

Modifying the counter (TCNTn) while the counter is running introduces a risk of missing a compare match between TCNTn and one of the OCRnx Registers.

Writing to the TCNTn Register blocks (removes) the compare match on the following timer clock for all compare units.

### 14.10.8 Timer/Counter3 – TCNT3H and TCNT3L

<table>
<thead>
<tr>
<th>Bit</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>TCNT3H</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
</tr>
<tr>
<td>TCNT3L</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
</tr>
</tbody>
</table>

**Read/Write**: R/W

**Initial Value**: 0 0 0 0 0 0 0 0

### 14.10.9 Output Compare Register 1 A – OCR1AH and OCR1AL

<table>
<thead>
<tr>
<th>Bit</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>OCR1AH</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
</tr>
<tr>
<td>OCR1AL</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
</tr>
</tbody>
</table>

**Read/Write**: R/W

**Initial Value**: 0 0 0 0 0 0 0 0

### 14.10.10 Output Compare Register 1 B – OCR1BH and OCR1BL

<table>
<thead>
<tr>
<th>Bit</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>OCR1BH</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
</tr>
<tr>
<td>OCR1BL</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
</tr>
</tbody>
</table>

**Read/Write**: R/W

**Initial Value**: 0 0 0 0 0 0 0 0

### 14.10.11 Output Compare Register 1 C – OCR1CH and OCR1CL

<table>
<thead>
<tr>
<th>Bit</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>OCR1CH</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
</tr>
<tr>
<td>OCR1CL</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
</tr>
</tbody>
</table>

**Read/Write**: R/W

**Initial Value**: 0 0 0 0 0 0 0 0
14.10.12 Output Compare Register 3 A – OCR3AH and OCR3AL

<table>
<thead>
<tr>
<th>Bit</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
</table>

| Read/Write | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |
| Initial Value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

14.10.13 Output Compare Register 3 B – OCR3BH and OCR3BL

<table>
<thead>
<tr>
<th>Bit</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>OCR3B[15:8]</td>
<td>OCR3B[7:0]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

| Read/Write | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |
| Initial Value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

14.10.14 Output Compare Register 3 C – OCR3CH and OCR3CL

<table>
<thead>
<tr>
<th>Bit</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>OCR3C[15:8]</td>
<td>OCR3C[7:0]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

| Read/Write | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |
| Initial Value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

The Output Compare Registers contain a 16-bit value that is continuously compared with the counter value (TCNTn). A match can be used to generate an Output Compare interrupt, or to generate a waveform output on the OCnx pin.

The Output Compare Registers are 16-bit in size. To ensure that both the high and low bytes are written simultaneously when the CPU writes to these registers, the access is performed using an 8-bit temporary High Byte Register (TEMP). This temporary register is shared by all the other 16-bit registers. See “Accessing 16-bit Registers” on page 110.

14.10.15 Input Capture Register 1 – ICR1H and ICR1L

<table>
<thead>
<tr>
<th>Bit</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>ICR1[15:8]</td>
<td>ICR1[7:0]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

| Read/Write | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |
| Initial Value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

14.10.16 Input Capture Register 3 – ICR3H and ICR3L

<table>
<thead>
<tr>
<th>Bit</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>ICR3[15:8]</td>
<td>ICR3[7:0]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

| Read/Write | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |
| Initial Value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

The Input Capture is updated with the counter (TCNTn) value each time an event occurs on the ICPn pin (or optionally on the Analog Comparator output for Timer/Counter1). The Input Capture can be used for defining the counter TOP value.

The Input Capture Register is 16-bit in size. To ensure that both the high and low bytes are read simultaneously when the CPU accesses these registers, the access is performed using an 8-bit temporary High Byte Register (TEMP). This temporary register is shared by all the other 16-bit registers. See “Accessing 16-bit Registers” on page 110.
14.10.17 Timer/Counter1 Interrupt Mask Register – TIMSK1

<table>
<thead>
<tr>
<th>Bit</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read/Write</td>
<td>R</td>
<td>R</td>
<td>R/W</td>
<td>R</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
</tr>
<tr>
<td>Initial Value</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

14.10.18 Timer/Counter3 Interrupt Mask Register – TIMSK3

<table>
<thead>
<tr>
<th>Bit</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read/Write</td>
<td>R</td>
<td>R</td>
<td>R/W</td>
<td>R</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
</tr>
<tr>
<td>Initial Value</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

- **Bit 5 – IClEn**: Timer/Countern, Input Capture Interrupt Enable
  When this bit is written to one, and the I-flag in the Status Register is set (interrupts globally enabled), the Timer/Countern Input Capture interrupt is enabled. The corresponding Interrupt Vector (See “Interrupts” on page 61.) is executed when the ICFn Flag, located in TIFRn, is set.

- **Bit 3 – OCIEnC**: Timer/Countern, Output Compare C Match Interrupt Enable
  When this bit is written to one, and the I-flag in the Status Register is set (interrupts globally enabled), the Timer/Countern Output Compare C Match interrupt is enabled. The corresponding Interrupt Vector (See “Interrupts” on page 61.) is executed when the OCFnC Flag, located in TIFRn, is set.

- **Bit 2 – OCIEnB**: Timer/Countern, Output Compare B Match Interrupt Enable
  When this bit is written to one, and the I-flag in the Status Register is set (interrupts globally enabled), the Timer/Countern Output Compare B Match interrupt is enabled. The corresponding Interrupt Vector (See “Interrupts” on page 61.) is executed when the OCFnB Flag, located in TIFRn, is set.

- **Bit 1 – OCIEnA**: Timer/Countern, Output Compare A Match Interrupt Enable
  When this bit is written to one, and the I-flag in the Status Register is set (interrupts globally enabled), the Timer/Countern Output Compare A Match interrupt is enabled. The corresponding Interrupt Vector (See “Interrupts” on page 61.) is executed when the OCFnA Flag, located in TIFRn, is set.

- **Bit 0 – TOIEn**: Timer/Countern, Overflow Interrupt Enable
  When this bit is written to one, and the I-flag in the Status Register is set (interrupts globally enabled), the Timer/Countern Overflow interrupt is enabled. The corresponding Interrupt Vector (See “Interrupts” on page 61.) is executed when the TOVn Flag, located in TIFRn, is set.

14.10.19 Timer/Counter1 Interrupt Flag Register – TIFR1

<table>
<thead>
<tr>
<th>Bit</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read/Write</td>
<td>R</td>
<td>R</td>
<td>R/W</td>
<td>R</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
</tr>
<tr>
<td>Initial Value</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
14.10.20 Timer/Counter3 Interrupt Flag Register – TIFR3

<table>
<thead>
<tr>
<th>Bit</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
<th>TIFR3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read/Write</td>
<td>-</td>
<td>-</td>
<td>R/W</td>
<td>R</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td></td>
</tr>
<tr>
<td>Initial Value</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

- **Bit 5 – ICFn: Timer/Countern, Input Capture Flag**
  This flag is set when a capture event occurs on the ICPn pin. When the Input Capture Register (ICRn) is set by the WGMn3:0 to be used as the TOP value, the ICFn Flag is set when the counter reaches the TOP value.

  ICFn is automatically cleared when the Input Capture Interrupt Vector is executed. Alternatively, ICFn can be cleared by writing a logic one to its bit location.

- **Bit 3 – OCFnC: Timer/Countern, Output Compare C Match Flag**
  This flag is set in the timer clock cycle after the counter (TCNTn) value matches the Output Compare Register C (OCRnC).

  Note that a Forced Output Compare (FOCnC) strobe will not set the OCFnC Flag.

  OCFnC is automatically cleared when the Output Compare Match C Interrupt Vector is executed. Alternatively, OCFnC can be cleared by writing a logic one to its bit location.

- **Bit 2 – OCFnB: Timer/Countern1, Output Compare B Match Flag**
  This flag is set in the timer clock cycle after the counter (TCNTn) value matches the Output Compare Register B (OCRnB).

  Note that a Forced Output Compare (FOCnB) strobe will not set the OCFnB Flag.

  OCFnB is automatically cleared when the Output Compare Match B Interrupt Vector is executed. Alternatively, OCFnB can be cleared by writing a logic one to its bit location.

- **Bit 1 – OCF1A: Timer/Countern1, Output Compare A Match Flag**
  This flag is set in the timer clock cycle after the counter (TCNTn value matches the Output Compare Register A (OCRnA).

  Note that a Forced Output Compare (FOCnA) strobe will not set the OCFnA Flag.

  OCFnA is automatically cleared when the Output Compare Match A Interrupt Vector is executed. Alternatively, OCFnA can be cleared by writing a logic one to its bit location.

- **Bit 0 – TOVn: Timer/Countern, Overflow Flag**
  The setting of this flag is dependent of the WGMn3:0 bits setting. In Normal and CTC modes, the TOVn Flag is set when the timer overflows. Refer to Table 14-5 on page 131 for the TOVn Flag behavior when using another WGMn3:0 bit setting.

  TOVn is automatically cleared when the Timer/Countern Overflow Interrupt Vector is executed. Alternatively, TOVn can be cleared by writing a logic one to its bit location.
15. 10-bit High Speed Timer/Counter4

15.1 Features

- Up to 10-Bit Accuracy
- Three Independent Output Compare Units
- Clear Timer on Compare Match (Auto Reload)
- Glitch Free, Phase and Frequency Correct Pulse Width Modulator (PWM)
- Enhanced PWM mode: one optional additional accuracy bit without effect on output frequency
- Variable PWM Period
- Independent Dead Time Generators for each PWM channels
- Synchronous update of PWM registers
- Five Independent Interrupt Sources (TOV4, OCF4A, OCF4B, OCF4D, FPF4)
- High Speed Asynchronous and Synchronous Clocking Modes
- Separate Prescaler Unit

15.2 Overview

Timer/Counter4 is a general purpose high speed Timer/Counter module, with three independent Output Compare Units, and with enhanced PWM support.

The Timer/Counter4 features a high resolution and a high accuracy usage with the lower prescaling opportunities. It can also support three accurate and high speed Pulse Width Modulators using clock speeds up to 64 MHz. In PWM mode Timer/Counter4 and the output compare registers serve as triple stand-alone PWMs with non-overlapping, non-inverted and inverted outputs. The enhanced PWM mode allows to get one more accuracy bit while keeping the frequency identical to normal mode (a PWM 8 bits accuracy in enhanced mode outputs the same frequency that a PWM 7 bits accuracy in normal mode). Similarly, the high prescaling opportunities make this unit useful for lower speed functions or exact timing functions with infrequent actions. A lock feature allows user to update the PWM registers and

A simplified block diagram of the Timer/Counter4 is shown in Figure 15-1. For actual placement of the I/O pins, refer to “Pinout ATmega16U4/ATmega32U4” on page 3. The device-specific I/O register and bit locations are listed in the “Register Description” on page 162.
15.2.1 Speed

The maximum speed of the Timer/Counter4 is 64 MHz. However, if a supply voltage below 4 volts is used, it is recommended to decrease the input frequency, because the Timer/Counter4 is not running fast enough on low voltage levels.

15.2.2 Accuracy

The Timer/Counter4 is a 10-bit Timer/Counter module that can alternatively be used as an 8-bit Timer/Counter. The Timer/Counter4 registers are basically 8-bit registers, but on top of that there is a 2-bit High Byte Register (TC4H) that can be used as a common temporary buffer to access the two MSBs of the 10-bit Timer/Counter4 registers by the AVR CPU via the 8-bit data bus, if the 10-bit accuracy is used. Whereas, if the two MSBs of the 10-bit registers are written to zero the Timer/Counter4 is working as an 8-bit Timer/Counter. When reading the low byte of any 8-bit register the two MSBs are written to the TC4H register, and when writing the low byte of any 8-bit register the two MSBs are written from the TC4H register. Special procedures must be followed when accessing the 10-bit Timer/Counter4 values via the 8-bit data bus. These procedures are described in the section “Accessing 10-Bit Registers” on page 159.

The Enhanced PWM mode allows to add a resolution bit to each Compare register A/B/D, while the output frequency remains identical to a Normal PWM mode. That means that the TC4H register contains one more bit that will be the MSB in a 11-bits enhanced PWM operation. See the section “Enhanced Compare/PWM mode” on page 148 for details about this feature and how to use it.
15.2.3 Registers

The Timer/Counter (TCNT4) and Output Compare Registers (OCR4A, OCR4B, OCR4C and OCR4D) are 8-bit registers that are used as a data source to be compared with the TCNT4 contents. The OCR4A, OCR4B and OCR4D registers determine the action on the OC4A, OC4B and OC4D pins and they can also generate the compare match interrupts. The OCR4C holds the Timer/Counter TOP value, i.e. the clear on compare match value. The Timer/Counter4 High Byte Register (TC4H) is a 2-bit register that is used as a common temporary buffer to access the MSB bits of the Timer/Counter4 registers, if the 10-bit accuracy is used.

Interrupt request (overflow TOV4, compare matches OCF4A, OCF4B, OCF4D and fault protection FPF4) signals are visible in the Timer Interrupt Flag Register (TIFR4) and Timer/Counter4 Control Register D (TCCR4D). The interrupts are individually masked with the Timer Interrupt Mask Register (TIMSK4) and the FPIE4 bit in the Timer/Counter4 Control Register D (TCCR4D).

Control signals are found in the Timer/Counter Control Registers TCCR4A, TCCR4B, TCCR4C, TCCR4D and TCCR4E.

15.2.4 Synchronization

In asynchronous clocking mode the Timer/Counter4 and the prescaler allow running the CPU from any clock source while the prescaler is operating on the fast peripheral clock (PCK) having frequency up to 64 MHz. This is possible because there is a synchronization boundary between the CPU clock domain and the fast peripheral clock domain. Figure 15-2 shows Timer/Counter 4 synchronization register block diagram and describes synchronization delays in between registers. Note that all clock gating details are not shown in the figure.

The Timer/Counter4 register values go through the internal synchronization registers, which cause the input synchronization delay, before affecting the counter operation. The registers TCCR4A, TCCR4B, TCCR4C, TCCR4D, OCR4A, OCR4B, OCR4C and OCR4D can be read back right after writing the register. The read back values are delayed for the Timer/Counter4 (TCNT4) register, Timer/Counter4 High Byte Register (TC4H) and flags (OCF4A, OCF4B, OCF4D and TOV4), because of the input and output synchronization.

The system clock frequency must be lower than half of the PCK frequency, because the synchronization mechanism of the asynchronous Timer/Counter4 needs at least two edges of the PCK when the system clock is high. If the frequency of the system clock is too high, it is a risk that data or control values are lost.
15.2.5 Definitions

Many register and bit references in this section are written in general form. A lower case “n” replaces the Timer/Counter number, in this case 0. A lower case “x” replaces the Output Compare Unit, in this case Compare Unit A, B, C or D. However, when using the register or bit defines in a program, the precise form must be used, i.e., TCNT4 for accessing Timer/Counter4 counter value and so on. The definitions in Table 15-1 are used extensively throughout the document.

<table>
<thead>
<tr>
<th>BOTTOM</th>
<th>MAX</th>
<th>TOP</th>
</tr>
</thead>
<tbody>
<tr>
<td>The counter reaches the BOTTOM when it becomes 0.</td>
<td>The counter reaches its MAXimum value when it becomes 0x3FF (decimal 1023).</td>
<td>The counter reaches the TOP value (stored in the OCR1C) when it becomes equal to the highest value in the count sequence. The TOP has a value 0x0FF as default after reset.</td>
</tr>
</tbody>
</table>
15.3 Counter Unit

The main part of the Timer/Counter4 is the programmable bi-directional counter unit. Figure 15-3 shows a block diagram of the counter and its surroundings.

**Figure 15-3. Counter Unit Block Diagram**

[Diagram of counter unit block diagram]

Signal description (internal signals):

- **count**: TCNT4 increment or decrement enable.
- **direction**: Select between increment and decrement.
- **clear**: Clear TCNT4 (set all bits to zero).
- **clk<sub>T4</sub>**: Timer/Counter clock, referred to as clk<sub>T4</sub> in the following.
- **top**: Signalize that TCNT4 has reached maximum value.
- **bottom**: Signalize that TCNT4 has reached minimum value (zero).

Depending of the mode of operation used, the counter is cleared, incremented, or decremented at each timer clock (clk<sub>T4</sub>). The timer clock is generated from a synchronous system clock or an asynchronous PLL clock using the Clock Select bits (CS4<3:0>) and the PLL Postscaler for High Speed Timer bits (PLLTM1:0). When no clock source is selected (CS4<3:0> = 0) the timer is stopped. However, the TCNT4 value can be accessed by the CPU, regardless of whether clk<sub>T1</sub> is present or not. A CPU write overrides (has priority over) all counter clear or count operations.

The counting sequence of the Timer/Counter4 is determined by the setting of the WGM10 and PWM4x bits located in the Timer/Counter4 Control Registers (TCCR4A, TCCR4C and TCCR4D). For more details about advanced counting sequences and waveform generation, see “Modes of Operation” on page 149. The Timer/Counter Overflow Flag (TOV4) is set according to the mode of operation selected by the PWM4x and WGM40 bits. The Overflow Flag can be used for generating a CPU interrupt.

15.3.1 Counter Initialization for Asynchronous Mode

To change Timer/Counter4 to the asynchronous mode follow the procedure below:

1. Enable PLL.
2. Wait 100μs for PLL to stabilize.
3. Poll the PLOCK bit until it is set.
4. Configure the PLLTM1:0 bits in the PLLFRQ register to enable the asynchronous mode (different from 0:0 value).
15.4 Output Compare Unit

The comparator continuously compares TCNT4 with the Output Compare Registers (OCR4A, OCR4B, OCR4C and OCR4D). Whenever TCNT4 equals to the Output Compare Register, the comparator signals a match. A match will set the Output Compare Flag (OCF4A, OCF4B or OCF4D) at the next timer clock cycle. If the corresponding interrupt is enabled, the Output Compare Flag generates an Output Compare interrupt. The Output Compare Flag is automatically cleared when the interrupt is executed. Alternatively, the flag can be cleared by software by writing a logical one to its I/O bit location. The Waveform Generator uses the match signal to generate an output according to operating mode set by the PWM4x, WGM40 and Compare Output mode (COM4x1:0) bits. The top and bottom signals are used by the Waveform Generator for handling the special cases of the extreme values in some modes of operation (See “Modes of Operation” on page 149.). Figure 15-4 shows a block diagram of the Output Compare unit.

Figure 15-4. Output Compare Unit, Block Diagram

The OCR4x Registers are double buffered when using any of the Pulse Width Modulation (PWM) modes. For the normal mode of operation, the double buffering is disabled. The double buffering synchronizes the update of the OCR4x Compare Registers to either top or bottom of the counting sequence. The synchronization prevents the occurrence of odd-length, non-symmetrical PWM pulses, thereby making the output glitch-free. See Figure 15-5 for an example. During the time between the write and the update operation, a read from OCR4A, OCR4B, OCR4C or OCR4D will read the contents of the temporary location. This means that the most recently written value always will read out of OCR4A, OCR4B, OCR4C or OCR4D.
15.4.1 Force Output Compare
In non-PWM waveform generation modes, the match output of the comparator can be forced by writing a one to the Force Output Compare (FOC4x) bit. Forcing Compare Match will not set the OCF4x Flag or reload/clear the timer, but the Waveform Output (OCW4x) will be updated as if a real Compare Match had occurred (the COM4x1:0 bits settings define whether the Waveform Output (OCW4x) is set, cleared or toggled).

15.4.2 Compare Match Blocking by TCNT4 Write
All CPU write operations to the TCNT4 Register will block any Compare Match that occur in the next timer clock cycle, even when the timer is stopped. This feature allows OCR4x to be initialized to the same value as TCNT4 without triggering an interrupt when the Timer/Counter clock is enabled.

15.4.3 Using the Output Compare Unit
Since writing TCNT4 in any mode of operation will block all Compare Matches for one timer clock cycle, there are risks involved when changing TCNT4 when using the Output Compare Unit, independently of whether the Timer/Counter is running or not. If the value written to TCNT4 equals the OCR4x value, the Compare Match will be missed, resulting in incorrect waveform generation. Similarly, do not write the TCNT4 value equal to BOTTOM when the counter is down-counting.

The setup of the Waveform Output (OCW4x) should be performed before setting the Data Direction Register for the port pin to output. The easiest way of setting the OCW4x value is to use the Force Output Compare (FOC4x) strobe bits in Normal mode. The OC4x keeps its value even when changing between Waveform Generation modes.

Be aware that the COM4x1:0 bits are not double buffered together with the compare value. Changing the COM4x1:0 bits will take effect immediately.
15.5 Dead Time Generator

The Dead Time Generator is provided for the Timer/Counter4 PWM output pairs to allow driving external power control switches safely. The Dead Time Generator is a separate block that can be used to insert dead times (non-overlapping times) for the Timer/Counter4 complementary output pairs OC4x and OC4x when the PWM mode is enabled and the COM4x1:0 bits are set to “01”. The sharing of tasks is as follows: the Waveform Generator generates the Waveform Output (OCW4x) and the Dead Time Generator generates the non-overlapping PWM output pair from the Waveform Output. Three Dead Time Generators are provided, one for each PWM output. The non-overlap time is adjustable and the PWM output and it’s complementary output are adjusted separately, and independently for both PWM outputs.

**Figure 15-6.** Output Compare Unit, Block Diagram

The Dead Time Generation is based on the 4-bit down counters that count the dead time, as shown in Figure 15-7. There is a dedicated prescaler in front of the Dead Time Generator that can divide the Timer/Counter4 clock (PCK or CK) by 1, 2, 4 or 8. This provides for large range of dead times that can be generated. The prescaler is controlled by two control bits DTPS41..40. The block has also a rising and falling edge detector that is used to start the dead time counting period. Depending on the edge, one of the transitions on the rising edges, OC4x or OC4x is delayed until the counter has counted to zero. The comparator is used to compare the counter with zero and stop the dead time insertion when zero has been reached. The counter is loaded with a 4-bit DT4H or DT4L value from DT4 I/O register, depending on the edge of the Waveform Output (OCW4x) when the dead time insertion is started. The Output Compare Output are delayed by one timer clock cycle at minimum from the Waveform Output when the Dead Time is adjusted to zero. The outputs OC4x and OC4x are inverted, if the PWM Inversion Mode bit PWM4X is set. This will also cause both outputs to be high during the dead time.

**Figure 15-7.** Dead Time Generator
The length of the counting period is user adjustable by selecting the dead time prescaler setting by using the DTPS41:40 control bits, and selecting then the dead time value in I/O register DT4. The DT4 register consists of two 4-bit fields, DT4H and DT4L that control the dead time periods of the PWM output and its' complementary output separately in terms of the number of prescaled dead time generator clock cycles. Thus the rising edge of OC4x and OC4x can have different dead time periods as the \( t_{\text{non-overlap}} \) / rising edge is adjusted by the 4-bit DT4H value and the \( t_{\text{non-overlap}} \) / falling edge is adjusted by the 4-bit DT4L value.

**Figure 15-8.** The Complementary Output Pair, COM4x1:0 = 1

\[ \text{OCWnx, OCnx, OCnx, (COMnx = 1)} \]

\( t_{\text{non-overlap}} / \text{rising edge} \quad t_{\text{non-overlap}} / \text{falling edge} \)

### 15.6 Compare Match Output Unit

The Compare Output Mode (COM4x1:0) bits have two functions. The Waveform Generator uses the COM4x1:0 bits for defining the inverted or non-inverted Waveform Output (OCW4x) at the next Compare Match. Also, the COM4x1:0 bits control the OC4x and OC4x pin output source. **Figure 15-9** shows a simplified schematic of the logic affected by the COM4x1:0 bit setting. The I/O Registers, I/O bits, and I/O pins in the figure are shown in bold. Only the parts of the general I/O Port Control Registers (DDR and PORT) that are affected by the COM4x1:0 bits are shown.

In Normal Mode (non-PWM) the Dead Time Generator is disabled and it is working like a synchronizer: the Output Compare (OC4x) is delayed from the Waveform Output (OCW4x) by one timer clock cycle. Whereas in Fast PWM Mode and in Phase and Frequency Correct PWM Mode when the COM4x1:0 bits are set to “01” both the non-inverted and the inverted Output Compare output are generated, and an user programmable Dead Time delay is inserted for these complementary output pairs (OC4x and OC4x). The functionality in PWM modes is similar to Normal mode when any other COM4x1:0 bit setup is used. When referring to the OC4x state, the reference is for the Output Compare output (OC4x) from the Dead Time Generator, not the OC4x pin. If a system reset occur, the OC4x is reset to “0”.

The general I/O port function is overridden by the Output Compare (OC4x / OC4x) from the Dead Time Generator if either of the COM4x1:0 bits are set. However, the OC4x pin direction (input or output) is still controlled by the Data Direction Register (DDR) for the port pin. The Data Direction Register bit for the OC4x and OC4x pins (DDR_OC4x and DDR_OC4x) must be set as output before the OC4x and OC4x values are visible on the pin. The port override function is independent of the Output Compare mode.

The design of the Output Compare Pin Configuration logic allows initialization of the OC4x state before the output is enabled. Note that some COM4x1:0 bit settings are reserved for certain modes of operation. For Output Compare Pin Configurations refer to Table 15-2 on page 151, Table 15-3 on page 152, Table 15-4 on page 154, and Table 15-5 on page 155.
15.6.1 Compare Output Mode and Waveform Generation

The Waveform Generator uses the COM4x1:0 bits differently in Normal mode and PWM modes. For all modes, setting the COM4x1:0 = 0 tells the Waveform Generator that no action on the OCW4x Output is to be performed on the next Compare Match. For compare output actions in the non-PWM modes refer to Table 15-6 on page 162. For fast PWM mode, refer to Table 15-7 on page 162, and for the Phase and Frequency Correct PWM refer to Table 15-8 on page 163. A change of the COM4x1:0 bits state will have effect at the first Compare Match after the bits are written. For non-PWM modes, the action can be forced to have immediate effect by using the FOC4x strobe bits.
15.6.2 Enhanced Compare/PWM mode

When the bit ENHC4 of TCCR4E register is set, the Enhanced Compare/PWM mode is enabled. This mode allows user to add an accuracy bit to Output Compare Register OCR4A, OCR4B and OCR4D. Like explained previously, a compare condition appears when one of the three Output Compare Registers (OCR4A/B/D) matches the value of TCNT4 (10-bits resolution). In basic PWM Mode, the corresponding enabled output toggles on the Compare Match. The Enhanced Compare/PWM mode introduces a bit that determines on which internal clock edge the Compare Match condition is actually signalled. That means that the corresponding outputs will toggle on the standard clock edge (like in Normal mode) if the LSB of OCR4A/B/D is ‘0’, or on the opposite (next) edge if the LSB is ‘1’.

User will notice that between Normal and Enhanced PWM modes, the output frequency will be identical, while the PWM resolution will be better in second case.

Writing to the Output Compare registers OCR4A/B/D or reading them will be identical in both modes. In Enhanced mode, user must just consider that the TC4H register can be up to 3-bits wide (and have the same behavior than during 2-bits operation). That will concern OCR4A, OCR4B and OCR4D registers accesses only. Indeed, the OCR4C register must not include the additional accuracy bit, and remains in the resolution that determines the output signal period.

**Figure 15-10.** How register access works in Enhanced mode

That figure shows that the true OCR4A/B/D value corresponds to the value loaded by the user shifted on the right in order to transfer the least significant bit directly to the Waveform generation module.

The maximum available resolution is 11-bits, but any other resolution can be specified. For example, a 8-bits resolution will allow to obtain the same frequency than a Normal PWM mode with 7-bits resolution.

**Example:**

- PLL Postcaler output = 64 MHz, No Prescaler on Timer/Counter4.
- Setting OCR4C = 0x7F determines a full 7-bits theoretical resolution, and so a 500kHz output frequency.
15.7 Synchronous update

To avoid unsynchronous and incoherent values in a cycle, if a synchronous update of one of several values is necessary, all values can be updated at the same time at the end of the PWM cycle by the Timer controller. The new set of values is calculated by software and the effective update can be initiated by software.

Figure 15-11. Lock feature and Synchronous update

In normal operation, each write to a Compare register is effective at the end of the current cycle. But some cases require that two or more Compare registers are updated synchronously, and that may not be always possible, mostly at high speed PWM frequencies. That may result in some PWM periods with incoherent values.

When using the Lock feature (TLOCK4=1), the values written to the Compare registers are not effective and temporarily buffered. When releasing the TLOCK4 bit, the update is initiated and the new whole set of values will be loaded at the end of the current PWM cycle.

See Section 15.12.5 "TCCR4E – Timer/Counter4 Control Register E" on page 169.

15.8 Modes of Operation

The mode of operation, i.e., the behavior of the Timer/Counter and the Output Compare pins, is defined by the combination of the Waveform Generation mode (bits PWM4x and WGM40) and Compare Output mode (COM4x1:0) bits. The Compare Output mode bits do not affect the counting sequence, while the Waveform Generation mode bits do. The COM4x1:0 bits control whether the PWM output generated should be inverted, non-inverted or complementary. For non-PWM modes the COM4x1:0 bits control whether the output should be set, cleared, or toggled at a Compare Match.

15.8.1 Normal Mode

The simplest mode of operation is the Normal mode (PWM4x = 0), the counter counts from BOTTOM to TOP (defined as OCR4C) then restarts from BOTTOM. The OCR4C defines the TOP value for the counter, hence also its resolution, and allows control of the Compare Match output frequency. In toggle Compare Output Mode the Waveform Output (OCW4x) is toggled at Compare Match between TCNT4 and OCR4x. In non-inverting Compare Output Mode the
Waveform Output is cleared on the Compare Match. In inverting Compare Output Mode the Waveform Output is set on Compare Match.

The timing diagram for the Normal mode is shown in Figure 15-12. The counter value (TCNT4) that is shown as a histogram in the timing diagram is incremented until the counter value matches the TOP value. The counter is then cleared at the following clock cycle. The diagram includes the Waveform Output (OCW4x) in toggle Compare Mode. The small horizontal line marks on the TCNT4 slopes represent Compare Matches between OCR4x and TCNT4.

Figure 15-12. Normal Mode, Timing Diagram

The Timer/Counter Overflow Flag (TOV4) is set in the same clock cycle as the TCNT4 becomes zero. The TOV4 Flag in this case behaves like a 11th bit, except that it is only set, not cleared. However, combined with the timer overflow interrupt, that automatically clears the TOV4 Flag, the timer resolution can be increased by software. There are no special cases to consider in the Normal mode, a new counter value can be written anytime.

The Output Compare Unit can be used to generate interrupts at some given time. Using the Output Compare to generate waveforms in Normal mode is not recommended, since this will occupy too much of the CPU time. For generating a waveform, the OCW4x output can be set to toggle its logical level on each Compare Match by setting the Compare Output mode bits to toggle mode (COM4x1:0 = 1). The OC4x value will not be visible on the port pin unless the data direction for the pin is set to output. The waveform generated will have a maximum frequency of \( f_{OC4x} = \frac{f_{\text{clk}4}}{4} \) when OCR4C is set to zero. The waveform frequency is defined by the following equation:

\[
f_{OC4x} = \frac{f_{\text{clk}4}}{2 \cdot (1 + OCR4C)}
\]

Resolution shows how many bit is required to express the value in the OCR4C register. It is calculated by following equation:

\[
\text{Resolution}_{\text{PWM}} = \log_2(OCR4C + 1).
\]
The Output Compare Pin configurations in Normal Mode are described in Table 15-2.

<table>
<thead>
<tr>
<th>COM4x1</th>
<th>COM4x0</th>
<th>OC4x Pin</th>
<th>OC4x Pin</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>Disconnected</td>
<td>Disconnected</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>Disconnected</td>
<td>OC4x</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>Disconnected</td>
<td>OC4x</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>Disconnected</td>
<td>OC4x</td>
</tr>
</tbody>
</table>

15.8.2 Fast PWM Mode

The fast Pulse Width Modulation or fast PWM mode (PWM4x = 1 and WGM40 = 0) provides a high frequency PWM waveform generation option. The fast PWM differs from the other PWM option by its single-slope operation. The counter counts from BOTTOM to TOP (defined as OCR4C) then restarts from BOTTOM. In non-inverting Compare Output mode the Waveform Output (OCW4x) is cleared on the Compare Match between TCNT4 and OCR4x and set at BOTTOM. In inverting Compare Output mode, the Waveform Output is set on Compare Match and cleared at BOTTOM. In complementary Compare Output mode the Waveform Output is cleared on the Compare Match and set at BOTTOM.

Due to the single-slope operation, the operating frequency of the fast PWM mode can be twice as high as the Phase and Frequency Correct PWM mode that use dual-slope operation. This high frequency makes the fast PWM mode well suited for power regulation, rectification, and DAC applications. High frequency allows physically small sized external components (coils, capacitors), and therefore reduces total system cost.

The timing diagram for the fast PWM mode is shown in Figure 15-13. The counter is incremented until the counter value matches the TOP value. The counter is then cleared at the following timer clock cycle. The TCNT4 value is in the timing diagram shown as a histogram for illustrating the single-slope operation. The diagram includes the Waveform Output in non-inverted and inverted Compare Output modes. The small horizontal line marks on the TCNT4 slopes represent Compare Matches between OCR4x and TCNT4.

**Figure 15-13.** Fast PWM Mode, Timing Diagram
The Timer/Counter Overflow Flag (TOV4) is set each time the counter reaches TOP. If the interrupt is enabled, the interrupt handler routine can be used for updating the compare value. In fast PWM mode, the compare unit allows generation of PWM waveforms on the OC4x pins. Setting the COM4x1:0 bits to two will produce a non-inverted PWM and setting the COM4x1:0 to three will produce an inverted PWM output. Setting the COM4x1:0 bits to one will enable complementary Compare Output mode and produce both the non-inverted (OC4x) and inverted output (OC4x). The actual value will only be visible on the port pin if the data direction for the port pin is set as output. The PWM waveform is generated by setting (or clearing) the Waveform Output (OCW4x) at the Compare Match between OCR4x and TCNT4, and clearing (or setting) the Waveform Output at the timer clock cycle the counter is cleared (changes from TOP to BOTTOM).

The PWM frequency for the output can be calculated by the following equation:

\[ f_{OCxPWM} = \frac{f_{clk4}}{N} \]

The \( N \) variable represents the number of steps in single-slope operation. The value of \( N \) equals either to the TOP value.

The extreme values for the OCR4x Register represents special cases when generating a PWM waveform output in the fast PWM mode. If the OCR4x is set equal to BOTTOM, the output will be a narrow spike for each MAX+1 timer clock cycle. Setting the OCR4x equal to MAX will result in a constantly high or low output (depending on the polarity of the output set by the COM4x1:0 bits.)

A frequency (with 50% duty cycle) waveform output in fast PWM mode can be achieved by setting the Waveform Output (OCW4x) to toggle its logical level on each Compare Match (COM4x1:0 = 1). The waveform generated will have a maximum frequency of \( f_{OC4} = f_{clk4}/4 \) when OCR4C is set to three.

The general I/O port function is overridden by the Output Compare value (OC4x / OC4x) from the Dead Time Generator, if either of the COM4x1:0 bits are set and the Data Direction Register bits for the OC4X and OC4X pins are set as an output. If the COM4x1:0 bits are cleared, the actual value from the port register will be visible on the port pin. The Output Compare Pin configurations are described in Table 15-3.

### Table 15-3. Output Compare Pin Configurations in Fast PWM Mode

<table>
<thead>
<tr>
<th>COM4x1</th>
<th>COM4x0</th>
<th>OC4x Pin</th>
<th>OC4x Pin</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>Disconnected</td>
<td>Disconnected</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>OC4x</td>
<td>OC4x</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>Disconnected</td>
<td>OC4x</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>Disconnected</td>
<td>OC4x</td>
</tr>
</tbody>
</table>

#### 15.8.3 Phase and Frequency Correct PWM Mode

The Phase and Frequency Correct PWM Mode (PWM4x = 1 and WGM40 = 1) provides a high resolution Phase and Frequency Correct PWM waveform generation option. The Phase and Frequency Correct PWM mode is based on a dual-slope operation. The counter counts repeatedly from BOTTOM to TOP (defined as OCR4C) and then from TOP to BOTTOM. In non-inverting Compare Output Mode the Waveform Output (OCW4x) is cleared on the Compare
Match between TCNT4 and OCR4x while upcounting, and set on the Compare Match while down-counting. In inverting Output Compare mode, the operation is inverted. In complementary Compare Output Mode, the Waveform Output is cleared on the Compare Match and set at BOTTOM. The dual-slope operation has lower maximum operation frequency than single slope operation. However, due to the symmetric feature of the dual-slope PWM modes, these modes are preferred for motor control applications.

The timing diagram for the Phase and Frequency Correct PWM mode is shown on Figure 15-14 in which the TCNT4 value is shown as a histogram for illustrating the dual-slope operation. The counter is incremented until the counter value matches TOP. When the counter reaches TOP, it changes the count direction. The TCNT4 value will be equal to TOP for one timer clock cycle. The diagram includes the Waveform Output (OCW4x) in non-inverted and inverted Compare Output Mode. The small horizontal line marks on the TCNT4 slopes represent Compare Matches between OCR4x and TCNT4.

Figure 15-14. Phase and Frequency Correct PWM Mode, Timing Diagram

The Timer/Counter Overflow Flag (TOV4) is set each time the counter reaches BOTTOM. The Interrupt Flag can be used to generate an interrupt each time the counter reaches the BOTTOM value.

In the Phase and Frequency Correct PWM mode, the compare unit allows generation of PWM waveforms on the OC4x pins. Setting the COM4x1:0 bits to two will produce a non-inverted PWM and setting the COM4x1:0 to three will produce an inverted PWM output. Setting the COM4A1:0 bits to one will enable complementary Compare Output mode and produce both the non-inverted (OC4x) and inverted output (OC4x). The actual values will only be visible on the port pin if the data direction for the port pin is set as output. The PWM waveform is generated by clearing (or setting) the Waveform Output (OCW4x) at the Compare Match between OCR4x and TCNT4 when the counter increments, and setting (or clearing) the Waveform Output at Compare Match when the counter decrements. The PWM frequency for the output when using the Phase and Frequency Correct PWM can be calculated by the following equation:

\[ f_{OCnxPCPWM} = \frac{f_{\text{clk}4x}}{N} \]
The \( N \) variable represents the number of steps in dual-slope operation. The value of \( N \) equals to the TOP value.

The extreme values for the OCR4C Register represent special cases when generating a PWM waveform output in the Phase and Frequency Correct PWM mode. If the OCR4C is set equal to BOTTOM, the output will be continuously low and if set equal to MAX the output will be continuously high for non-inverted PWM mode. For inverted PWM the output will have the opposite logic values.

The general I/O port function is overridden by the Output Compare value (OC4x / OC4x̅) from the Dead Time Generator, if either of the COM4x1:0 bits are set and the Data Direction Register bits for the OC4X and OC4X̅ pins are set as an output. If the COM4x1:0 bits are cleared, the actual value from the port register will be visible on the port pin. The configurations of the Output Compare Pins are described in Table 15-4.

<table>
<thead>
<tr>
<th>COM4x1</th>
<th>COM4x0</th>
<th>OC4x Pin</th>
<th>OC4x Pin</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>Disconnected</td>
<td>Disconnected</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>OC4x</td>
<td>OC4x</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>Disconnected</td>
<td>OC4x</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>Disconnected</td>
<td>OC4x</td>
</tr>
</tbody>
</table>

### 15.8.4 PWM6 Mode

The PWM6 Mode (PWM4A = 1, WGM41 = 1 and WGM40 = x) provide PWM waveform generation option e.g. for controlling Brushless DC (BLDC) motors. In the PWM6 Mode the OCR4A Register controls all six Output Compare waveforms as the same Waveform Output (OCW4A) from the Waveform Generator is used for generating all waveforms. The PWM6 Mode also provides an Output Compare Override Enable Register (OC4OE) that can be used with an instant response for disabling or enabling the Output Compare pins. If the Output Compare Override Enable bit is cleared, the actual value from the port register will be visible on the port pin.

The PWM6 Mode provides two counter operation modes, a single-slope operation and a dual-slope operation. If the single-slope operation is selected (the WGM40 bit is set to 0), the counter counts from BOTTOM to TOP (defined as OCR4C) then restart from BOTTOM like in Fast PWM Mode. The PWM waveform is generated by setting (or clearing) the Waveform Output (OCW4A) at the Compare Match between OCR4A and TCNT4, and clearing (or setting) the Waveform Output at the timer clock cycle the counter is cleared (changes from TOP to BOTTOM). The Timer/Counter Overflow Flag (TOV4) is set each time the counter reaches the TOP and, if the interrupt is enabled, the interrupt handler routine can be used for updating the compare value.

Whereas, if the dual-slope operation is selected (the WGM40 bit is set to 1), the counter counts repeatedly from BOTTOM to TOP (defined as OCR4C) and then from TOP to BOTTOM like in Phase and Frequency Correct PWM Mode. The PWM waveform is generated by setting (or clearing) the Waveform Output (OCW4A) at the Compare Match between OCR4A and TCNT4 when the counter increments, and clearing (or setting) the Waveform Output at the Compare Match between OCR4A and TCNT4 when the counter decrements. The Timer/Counter Overflow Flag (TOV4) is set each time the counter reaches the BOTTOM and, if the interrupt is enabled, the interrupt handler routine can be used for updating the compare value.

The timing diagram for the PWM6 Mode in single-slope operation (WGM41 = 0) when the COM4A1:0 bits are set to “10” is shown in Figure 15-15. The counter is incremented until the
counter value matches the TOP value. The counter is then cleared at the following timer clock cycle. The TCNT4 value is in the timing diagram shown as a histogram for illustrating the single-slope operation. The timing diagram includes Output Compare pins OC4A and OC4A, and the corresponding Output Compare Override Enable bits (OC4OE1..OC4OE0).

**Figure 15-15. PWM6 Mode, Single-slope Operation, Timing Diagram**

The general I/O port function is overridden by the Output Compare value (OC4x / OC4x) from the Dead Time Generator if either of the COM4x1:0 bits are set. The Output Compare pins can also be overridden by the Output Compare Override Enable bits OC4OE5..OC4OE0. If an Override Enable bit is cleared, the actual value from the port register will be visible on the port pin and, if the Override Enable bit is set, the Output Compare pin is allowed to be connected on the port pin. The Output Compare Pin configurations are described in Table 15-5.

**Table 15-5. Output Compare Pin configurations in PWM6 Mode**

<table>
<thead>
<tr>
<th>COM4A1</th>
<th>COM4A0</th>
<th>OC4A Pin (PC6)</th>
<th>OC4A Pin (PC7)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>Disconnected</td>
<td>Disconnected</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>OC4A • OC4OE0</td>
<td>OC4A • OC4OE1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>OC4A • OC4OE0</td>
<td>OC4A • OC4OE1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>OC4A • OC4OE0</td>
<td>OC4A • OC4OE1</td>
</tr>
<tr>
<td>COM4B1</td>
<td>COM4B0</td>
<td>OC4B Pin (PB5)</td>
<td>OC4B Pin (PB6)</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>Disconnected</td>
<td>Disconnected</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>OC4A • OC4OE2</td>
<td>OC4A • OC4OE3</td>
</tr>
</tbody>
</table>
Table 15-5. Output Compare Pin configurations in PWM6 Mode (Continued)

<table>
<thead>
<tr>
<th>COM4A1</th>
<th>COM4A0</th>
<th>OC4A Pin (PC6)</th>
<th>OC4A Pin (PC7)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>OC4A • OC4OE2</td>
<td>OC4A • OC4OE3</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>OC4A • OC4OE2</td>
<td>OC4A • OC4OE3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>COM4D1</th>
<th>COM4D0</th>
<th>OC4D Pin (PD6)</th>
<th>OC4D Pin (PD7)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>Disconnected</td>
<td>Disconnected</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>OC4A • OC4OE4</td>
<td>OC4A • OC4OE5</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>OC4A • OC4OE4</td>
<td>OC4A • OC4OE5</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>OC4A • OC4OE4</td>
<td>OC4A • OC4OE5</td>
</tr>
</tbody>
</table>

15.9 Timer/Counter Timing Diagrams

The Timer/Counter is a synchronous design and the timer clock (clkT4) is therefore shown as a clock enable signal in the following figures. The figures include information on when Interrupt Flags are set.

Figure 15-16 contains timing data for basic Timer/Counter operation. The figure shows the count sequence close to the MAX value in all modes other than Phase and Frequency Correct PWM Mode. Figure 15-17 shows the same timing data, but with the prescaler enabled, in all modes other than Phase and Frequency Correct PWM Mode. Figure 15-18 shows the setting of OCF4A, OCF4B and OCF4D in all modes, and Figure 15-19 shows the setting of TOV4 in Phase and Frequency Correct PWM Mode.

**Figure 15-16.** Timer/Counter Timing Diagram, no Prescaling

![Timer/Counter Timing Diagram, no Prescaling](image)

**Figure 15-17.** Timer/Counter Timing Diagram, with Prescaler (f_{clkT4}/8)

![Timer/Counter Timing Diagram, with Prescaler](image)
15.10 Fault Protection Unit

The Timer/Counter4 incorporates a Fault Protection unit that can disable the PWM output pins, if an external event is triggered. The external signal indicating an event can be applied via the external interrupt INT0 pin or alternatively, via the analog-comparator unit. The Fault Protection unit is illustrated by the block diagram shown in Figure 15-20. The elements of the block diagram that are not directly a part of the Fault Protection unit are gray shaded.

When the Fault Protection mode is enabled by the Fault Protection Enable (FPEN4) bit and a change of the logic level (an event) occurs on the external interrupt pin (INT0), alternatively on the Analog Comparator output (ACO), and this change confirms to the setting of the edge detector, a Fault Protection mode will be triggered. When a Fault Protection is triggered, the COM4x bits are cleared, Output Comparators are disconnected from the PWM output pins and the PORTB register bits are connected on the PWM output pins. The Fault Protection Enable (FPEN4) is automatically cleared at the same system clock as the COM4nx bits are cleared. If the Fault Protection Interrupt Enable bit (FPIE4) is set, a Fault Protection interrupt is generated and the FPEN4 bit is cleared. Alternatively the FPEN4 bit can be polled by software to figure out when the Timer/Counter has entered to Fault Protection mode.
15.10.1 Fault Protection Trigger Source
The main trigger source for the Fault Protection unit is the external interrupt pin (INT0). Alternatively, the Analog Comparator output can be used as trigger source for the Fault Protection unit. The Analog Comparator is selected as trigger source by setting the Fault Protection Analog Comparator (FPAC4) bit in the Timer/Counter4 Control Register (TCCR4D). Be aware that changing trigger source can trigger a Fault Protection mode. Therefore it is recommended to clear the FPF4 flag after changing trigger source, setting edge detector or enabling the Fault Protection.

Both the external interrupt pin (INT0) and the Analog Comparator output (ACO) inputs are sampled using the same technique as for the T0 pin (Figure 12-1 on page 89). The edge detector is also identical. However, when the noise canceler is enabled, additional logic is inserted before the edge detector, which increases the delay by four system clock cycles. An Input Capture can also be triggered by software by controlling the port of the INT0 pin.

15.10.2 Noise Canceler
The noise canceler improves noise immunity by using a simple digital filtering scheme. The noise canceler input is monitored over four samples, and all four must be equal for changing the output that in turn is used by the edge detector.

The noise canceler is enabled by setting the Fault Protection Noise Canceler (FPNC4) bit in Timer/Counter4 Control Register D (TCCR4D). When enabled the noise canceler introduces additional four system clock cycles of delay from a change applied to the input. The noise canceler uses the system clock and is therefore not affected by the prescaler.
15.11 Accessing 10-Bit Registers

If 10-bit values are written to the TCNTn and OCRnA/B/C/D registers, the 10-bit registers can be byte accessed by the AVR CPU via the 8-bit data bus using two read or write operations. The 10-bit registers have a common 2-bit Timer/Counter4 High Byte Register (TC4H) that is used for temporary storing of the two MSBs of the 10-bit access. The same TC4H register is shared between all 10-bit registers. Accessing the low byte triggers the 10-bit read or write operation. When the low byte of a 10-bit register is written by the CPU, the high byte stored in the TC4H register, and the low byte written are both copied into the 10-bit register in the same clock cycle. When the low byte of a 10-bit register is read by the CPU, the high byte of the 10-bit register is copied into the TC4H register in the same clock cycle as the low byte is read.

To do a 10-bit write, the high byte must be written to the TC4H register before the low byte is written. For a 10-bit read, the low byte must be read before the high byte.

The following code examples show how to access the 10-bit timer registers assuming that no interrupts updates the TC4H register. The same principle can be used directly for accessing the OCRnA/B/C/D registers.

<table>
<thead>
<tr>
<th>Assembly Code Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
</tr>
<tr>
<td>; Set TCNTn to 0x01FF</td>
</tr>
<tr>
<td>ldi r17,0x01</td>
</tr>
<tr>
<td>ldi r16,0xFF</td>
</tr>
<tr>
<td>out TCnH,r17</td>
</tr>
<tr>
<td>out TCNTn,r16</td>
</tr>
<tr>
<td>; Read TCNTn into r17:r16</td>
</tr>
<tr>
<td>in r16,TCnH</td>
</tr>
<tr>
<td>in r17,TCnH</td>
</tr>
<tr>
<td>...</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>C Code Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>unsigned int i;</td>
</tr>
<tr>
<td>...</td>
</tr>
<tr>
<td>/* Set TCNTn to 0x01FF */</td>
</tr>
<tr>
<td>TCnH = 0x01;</td>
</tr>
<tr>
<td>TCNTn = 0xFF;</td>
</tr>
<tr>
<td>/* Read TCNTn into i */</td>
</tr>
<tr>
<td>i = TCNTn;</td>
</tr>
<tr>
<td>i</td>
</tr>
<tr>
<td>...</td>
</tr>
</tbody>
</table>

Note: 1. The example code assumes that the part specific header file is included. For I/O registers located in extended I/O map, “IN”, “OUT”, “SBIS”, “SBIC”, “CBI”, and “SBI” instructions must be replaced with instructions that allow access to extended I/O. Typically “LDS” and “STS” combined with “SBRS”, “SBRC”, “SBR”, and “CBR”.

The assembly code example returns the TCNTn value in the r17:r16 register pair.
It is important to notice that accessing 10-bit registers are atomic operations. If an interrupt occurs between the two instructions accessing the 10-bit register, and the interrupt code updates the TC4H register by accessing the same or any other of the 10-bit timer registers, then the result of the access outside the interrupt will be corrupted. Therefore, when both the main code and the interrupt code update the TC4H register, the main code must disable the interrupts during the 16-bit access.

The following code examples show how to do an atomic read of the TCNTn register contents. Reading any of the OCRnA/B/C/D registers can be done by using the same principle.

<table>
<thead>
<tr>
<th>Assembly Code Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>TIM1_ReadTCNTn:</td>
</tr>
<tr>
<td>; Save global interrupt flag</td>
</tr>
<tr>
<td>in r18, SREG</td>
</tr>
<tr>
<td>; Disable interrupts</td>
</tr>
<tr>
<td>cli</td>
</tr>
<tr>
<td>; Read TCNTn into r17:r16</td>
</tr>
<tr>
<td>in r16, TCNTn</td>
</tr>
<tr>
<td>in r17, TCNH</td>
</tr>
<tr>
<td>; Restore global interrupt flag</td>
</tr>
<tr>
<td>out SREG, r18</td>
</tr>
<tr>
<td>ret</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>C Code Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>unsigned int TIM1_ReadTCNTn( void )</td>
</tr>
<tr>
<td>{</td>
</tr>
<tr>
<td>unsigned char sreg;</td>
</tr>
<tr>
<td>unsigned int i;</td>
</tr>
<tr>
<td>/* Save global interrupt flag */</td>
</tr>
<tr>
<td>sreg = SREG;</td>
</tr>
<tr>
<td>/* Disable interrupts */</td>
</tr>
<tr>
<td>_CLI();</td>
</tr>
<tr>
<td>/* Read TCNTn into i */</td>
</tr>
<tr>
<td>i = TCNTn;</td>
</tr>
<tr>
<td>i</td>
</tr>
<tr>
<td>/* Restore global interrupt flag */</td>
</tr>
<tr>
<td>SREG = sreg;</td>
</tr>
<tr>
<td>return i;</td>
</tr>
<tr>
<td>}</td>
</tr>
</tbody>
</table>

Note: 1. The example code assumes that the part specific header file is included.

For I/O registers located in extended I/O map, "IN", "OUT", "SBIS", "SBIC", "CBI", and "SBI" instructions must be replaced with instructions that allow access to extended I/O. Typically "LDS" and "STS" combined with "SBRS", "SBRC", "SBR", and "CBR".

The assembly code example returns the TCNTn value in the r17:r16 register pair.
The following code examples show how to do an atomic write of the TCNTn register contents. Writing any of the OCRnA/B/C/D registers can be done by using the same principle.

<table>
<thead>
<tr>
<th>Assembly Code Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>TIMI_WriteTCNTn:</td>
</tr>
<tr>
<td>; Save global interrupt flag</td>
</tr>
<tr>
<td>in r18,SREG</td>
</tr>
<tr>
<td>; Disable interrupts</td>
</tr>
<tr>
<td>cli</td>
</tr>
<tr>
<td>; Set TCNTn to r17:r16</td>
</tr>
<tr>
<td>out TCnH,r17</td>
</tr>
<tr>
<td>out TCnTn,r16</td>
</tr>
<tr>
<td>; Restore global interrupt flag</td>
</tr>
<tr>
<td>out SREG,r18</td>
</tr>
<tr>
<td>ret</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>C Code Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>void TIMI_WriteTCNTn( unsigned int i )</td>
</tr>
<tr>
<td>{</td>
</tr>
<tr>
<td>unsigned char sreg;</td>
</tr>
<tr>
<td>unsigned int i;</td>
</tr>
<tr>
<td>/* Save global interrupt flag */</td>
</tr>
<tr>
<td>sreg = SREG;</td>
</tr>
<tr>
<td>/* Disable interrupts */</td>
</tr>
<tr>
<td>_CLI();</td>
</tr>
<tr>
<td>/* Set TCNTn to i */</td>
</tr>
<tr>
<td>TCnH = (i &gt;&gt; 8);</td>
</tr>
<tr>
<td>TCnTn = (unsigned char)i;</td>
</tr>
<tr>
<td>/* Restore global interrupt flag */</td>
</tr>
<tr>
<td>SREG = sreg;</td>
</tr>
<tr>
<td>}</td>
</tr>
</tbody>
</table>

Note: 1. The example code assumes that the part specific header file is included. For I/O registers located in extended I/O map, “IN”, “OUT”, “SBIS”, “SBIC”, “CBI”, and “SBI” instructions must be replaced with instructions that allow access to extended I/O. Typically “LDS” and “STS” combined with “SBRS”, “SBRC”, “SBR”, and “CBR”.

The assembly code example requires that the r17:r16 register pair contains the value to be written to TCNTn.

15.11.1 Reusing the temporary high byte register

If writing to more than one 10-bit register where the high byte is the same for all registers written, then the high byte only needs to be written once. However, note that the same rule of atomic operation described previously also applies in this case.
15.12 Register Description

15.12.1 TCCR4A – Timer/Counter4 Control Register A

<table>
<thead>
<tr>
<th>Bit</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
<th>TCCR4A</th>
</tr>
</thead>
<tbody>
<tr>
<td>COM4A1</td>
<td>R/W</td>
<td>COM4A0</td>
<td>R/W</td>
<td>COM4B1</td>
<td>R/W</td>
<td>COM4B0</td>
<td>W</td>
<td>FOC4A</td>
<td>W</td>
</tr>
<tr>
<td>FOC4B</td>
<td>W</td>
<td>PWM4A</td>
<td>R/W</td>
<td>PWM4B</td>
<td>R/W</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **Bits 7,6 - COM4A1, COM4A0: Comparator A Output Mode, Bits 1 and 0**
  These bits control the behavior of the Waveform Output (OCW4A) and the connection of the Output Compare pin (OC4A). If one or both of the COM4A1:0 bits are set, the OC4A output overrides the normal port functionality of the I/O pin it is connected to. The complementary OC4B output is connected only in PWM modes when the COM4A1:0 bits are set to “01”. Note that the Data Direction Register (DDR) bit corresponding to the OC4A and OC4A pins must be set in order to enable the output driver.

The function of the COM4A1:0 bits depends on the PWM4A, WGM40 and WGM41 bit settings. **Table 15-6** shows the COM4A1:0 bit functionality when the PWM4A bit is set to Normal Mode (non-PWM).

**Table 15-6.** Compare Output Mode, Normal Mode (non-PWM)

<table>
<thead>
<tr>
<th>COM4A1..0</th>
<th>OCW4A Behavior</th>
<th>OC4A Pin</th>
<th>OC4A Pin</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>Normal port operation.</td>
<td>Disconnected</td>
<td>Disconnected</td>
</tr>
<tr>
<td>01</td>
<td>Toggle on Compare Match.</td>
<td>Connected</td>
<td>Disconnected</td>
</tr>
<tr>
<td>10</td>
<td>Clear on Compare Match.</td>
<td>Connected</td>
<td>Disconnected</td>
</tr>
<tr>
<td>11</td>
<td>Set on Compare Match.</td>
<td>Connected</td>
<td>Disconnected</td>
</tr>
</tbody>
</table>

**Table 15-7** shows the COM4A1:0 bit functionality when the PWM4A, WGM40 and WGM41 bits are set to fast PWM mode.

**Table 15-7.** Compare Output Mode, Fast PWM Mode

<table>
<thead>
<tr>
<th>COM4A1..0</th>
<th>OCW4A Behavior</th>
<th>OC4A</th>
<th>OC4A</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>Normal port operation.</td>
<td>Disconnected</td>
<td>Disconnected</td>
</tr>
<tr>
<td>01</td>
<td>Cleared on Compare Match. Set when TCNT4 = 0x000.</td>
<td>Connected</td>
<td>Connected</td>
</tr>
<tr>
<td>10</td>
<td>Cleared on Compare Match. Set when TCNT4 = 0x000.</td>
<td>Connected</td>
<td>Disconnected</td>
</tr>
<tr>
<td>11</td>
<td>Set on Compare Match. Cleared when TCNT4 = 0x000.</td>
<td>Connected</td>
<td>Disconnected</td>
</tr>
</tbody>
</table>
Table 15-8 shows the COM4A1:0 bit functionality when the PWM4A, WGM40 and WGM41 bits are set to Phase and Frequency Correct PWM Mode.

<table>
<thead>
<tr>
<th>COM1A1..0</th>
<th>OCW1A Behavior</th>
<th>OC4A Pin</th>
<th>OC4A Pin</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>Normal port operation.</td>
<td>Disconnected</td>
<td>Disconnected</td>
</tr>
<tr>
<td>01</td>
<td>Cleared on Compare Match when up-counting. Set on Compare Match when down-counting.</td>
<td>Connected</td>
<td>Connected</td>
</tr>
<tr>
<td>10</td>
<td>Cleared on Compare Match when up-counting. Set on Compare Match when down-counting.</td>
<td>Connected</td>
<td>Disconnected</td>
</tr>
<tr>
<td>11</td>
<td>Set on Compare Match when up-counting. Cleared on Compare Match when down-counting.</td>
<td>Connected</td>
<td>Disconnected</td>
</tr>
</tbody>
</table>

Table 15-9 shows the COM4A1:0 bit functionality when the PWM4A, WGM40 and WGM41 bits are set to single-slope PWM6 Mode. In the PWM6 Mode, the same Waveform Output (OCW4A) is used for generating all waveforms and the Output Compare values OC4A and OC4A are connected on OC4x and OC4x pins as described below.

<table>
<thead>
<tr>
<th>COM4A1..0</th>
<th>OCW4A Behavior</th>
<th>OC4x Pin</th>
<th>OC4x Pin</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>Normal port operation.</td>
<td>Disconnected</td>
<td>Disconnected</td>
</tr>
<tr>
<td>01</td>
<td>Cleared on Compare Match. Set when TCNT4 = 0x000.</td>
<td>OC4A</td>
<td>OC4A</td>
</tr>
<tr>
<td>10</td>
<td>Cleared on Compare Match. Set when TCNT4 = 0x000.</td>
<td>OC4A</td>
<td>OC4A</td>
</tr>
<tr>
<td>11</td>
<td>Set on Compare Match. Cleared when TCNT4 = 0x000.</td>
<td>OC4A</td>
<td>OC4A</td>
</tr>
</tbody>
</table>

Table 15-10 shows the COM4A1:0 bit functionality when the PWM4A, WGM40 and WGM41 bits are set to dual-slope PWM6 Mode.

<table>
<thead>
<tr>
<th>COM4A1..0</th>
<th>OCW4A Behavior</th>
<th>OC4x Pin</th>
<th>OC4x Pin</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>Normal port operation.</td>
<td>Disconnected</td>
<td>Disconnected</td>
</tr>
<tr>
<td>01</td>
<td>Cleared on Compare Match when up-counting. Set on Compare Match when down-counting.</td>
<td>OC4A</td>
<td>OC4A</td>
</tr>
<tr>
<td>10</td>
<td>Cleared on Compare Match when up-counting. Set on Compare Match when down-counting.</td>
<td>OC4A</td>
<td>OC4A</td>
</tr>
<tr>
<td>11</td>
<td>Set on Compare Match when up-counting. Cleared on Compare Match when down-counting.</td>
<td>OC4A</td>
<td>OC4A</td>
</tr>
</tbody>
</table>

- **Bits 5,4 - COM4B1, COM4B0: Comparator B Output Mode, Bits 1 and 0**

These bits control the behavior of the Waveform Output (OCW4B) and the connection of the Output Compare pin (OC4B). If one or both of the COM4B1:0 bits are set, the OC4B output overrides the normal port functionality of the I/O pin it is connected to. The complementary OC4B output is connected only in PWM modes when the COM4B1:0 bits are set to “01”. Note
that the Data Direction Register (DDR) bit corresponding to the OC4B pin must be set in order to enable the output driver.

The function of the COM4B1:0 bits depends on the PWM4B and WGM40 bit settings. **Table 15-11** shows the COM4B1:0 bit functionality when the PWM4B bit is set to Normal Mode (non-PWM).

**Table 15-11.** Compare Output Mode, Normal Mode (non-PWM)

<table>
<thead>
<tr>
<th>COM4B1..0</th>
<th>OCW4B Behavior</th>
<th>OC4B Pin</th>
<th>OC4B Pin</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>Normal port operation.</td>
<td>Disconnect</td>
<td>Disconnect</td>
</tr>
<tr>
<td>01</td>
<td>Toggle on Compare Match.</td>
<td>Connect</td>
<td>Disconnect</td>
</tr>
<tr>
<td>10</td>
<td>Clear on Compare Match.</td>
<td>Connect</td>
<td>Disconnect</td>
</tr>
<tr>
<td>11</td>
<td>Set on Compare Match.</td>
<td>Connect</td>
<td>Disconnect</td>
</tr>
</tbody>
</table>

**Table 15-12** shows the COM4B1:0 bit functionality when the PWM4B and WGM40 bits are set to Fast PWM Mode.

**Table 15-12.** Compare Output Mode, Fast PWM Mode

<table>
<thead>
<tr>
<th>COM4B1..0</th>
<th>OCW4B Behavior</th>
<th>OC4B Pin</th>
<th>OC4B Pin</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>Normal port operation.</td>
<td>Disconnect</td>
<td>Disconnect</td>
</tr>
<tr>
<td>01</td>
<td>Cleared on Compare Match.</td>
<td>Connect</td>
<td>Connect</td>
</tr>
<tr>
<td></td>
<td>Set when TCNT4 = 0x000.</td>
<td>Connect</td>
<td>Connect</td>
</tr>
<tr>
<td>10</td>
<td>Cleared on Compare Match.</td>
<td>Connect</td>
<td>Disconnect</td>
</tr>
<tr>
<td></td>
<td>Set when TCNT4 = 0x000.</td>
<td>Connect</td>
<td>Disconnect</td>
</tr>
<tr>
<td>11</td>
<td>Set on Compare Match.</td>
<td>Connect</td>
<td>Disconnect</td>
</tr>
<tr>
<td></td>
<td>Cleared when TCNT4 = 0x000.</td>
<td>Connect</td>
<td>Disconnect</td>
</tr>
</tbody>
</table>

**Table 15-13** shows the COM4B1:0 bit functionality when the PWM4B and WGM40 bits are set to Phase and Frequency Correct PWM Mode.

**Table 15-13.** Compare Output Mode, Phase and Frequency Correct PWM Mode

<table>
<thead>
<tr>
<th>COM4B1..0</th>
<th>OCW4B Behavior</th>
<th>OC4B Pin</th>
<th>OC4B Pin</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>Normal port operation.</td>
<td>Disconnect</td>
<td>Disconnect</td>
</tr>
<tr>
<td>01</td>
<td>Cleared on Compare Match when up-counting.</td>
<td>Connect</td>
<td>Connect</td>
</tr>
<tr>
<td></td>
<td>Set on Compare Match when down-counting.</td>
<td>Connect</td>
<td>Connect</td>
</tr>
<tr>
<td>10</td>
<td>Cleared on Compare Match when up-counting.</td>
<td>Connect</td>
<td>Disconnect</td>
</tr>
<tr>
<td></td>
<td>Set on Compare Match when down-counting.</td>
<td>Connect</td>
<td>Disconnect</td>
</tr>
<tr>
<td>11</td>
<td>Set on Compare Match when up-counting.</td>
<td>Connect</td>
<td>Disconnect</td>
</tr>
<tr>
<td></td>
<td>Cleared on Compare Match when down-counting.</td>
<td>Connect</td>
<td>Disconnect</td>
</tr>
</tbody>
</table>

**Bit 3 - FOC4A: Force Output Compare Match 4A**
The FOC4A bit is only active when the PWM4A bit specify a non-PWM mode.

Writing a logical one to this bit forces a change in the Waveform Output (OCW4A) and the Output Compare pin (OC4A) according to the values already set in COM4A1 and COM4A0. If COM4A1 and COM4A0 written in the same cycle as FOC4A, the new settings will be used. The Force Output Compare bit can be used to change the output pin value regardless of the timer
value. The automatic action programmed in COM4A1 and COM4A0 takes place as if a compare match had occurred, but no interrupt is generated. The FOC4A bit is always read as zero.

- **Bit 2 - FOC4B: Force Output Compare Match 4B**
The FOC4B bit is only active when the PWM4B bit specify a non-PWM mode.

Writing a logical one to this bit forces a change in the Waveform Output (OCW4B) and the Output Compare pin (OC4B) according to the values already set in COM4B1 and COM4B0. If COM4B1 and COM4B0 written in the same cycle as FOC4B, the new settings will be used. The Force Output Compare bit can be used to change the output pin value regardless of the timer value. The automatic action programmed in COM4B1 and COM4B0 takes place as if a compare match had occurred, but no interrupt is generated.

The FOC4B bit is always read as zero.

- **Bit 1 - PWM4A: Pulse Width Modulator A Enable**
When set (one) this bit enables PWM mode based on comparator OCR4A

- **Bit 0 - PWM4B: Pulse Width Modulator B Enable**
When set (one) this bit enables PWM mode based on comparator OCR4B.

### 15.12.2 TCCR4B – Timer/Counter4 Control Register B

<table>
<thead>
<tr>
<th>Bit</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read/Write</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
</tr>
<tr>
<td>Initial value</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

- **Bit 7 - PWM4X: PWM Inversion Mode**
When this bit is set (one), the PWM Inversion Mode is selected and the Dead Time Generator outputs, OC4x and OC4x are inverted.

- **Bit 6 - PSR4: Prescaler Reset Timer/Counter4**
When this bit is set (one), the Timer/Counter4 prescaler (TCNT4 is unaffected) will be reset. The bit will be cleared by hardware after the operation is performed. Writing a zero to this bit will have no effect. This bit will always read as zero.

- **Bits 5,4 - DTPS41, DTPS40: Dead Time Prescaler Bits**
The Timer/Counter4 Control Register B is a 8-bit read/write register.

The dedicated Dead Time prescaler in front of the Dead Time Generator can divide the Timer/Counter4 clock (PCK or CK) by 1, 2, 4 or 8 providing a large range of dead times that can be generated. The Dead Time prescaler is controlled by two bits DTPS41 and DTPS40 from the Dead Time Prescaler register. These bits define the division factor of the Dead Time prescaler. The division factors are given in Table 15-14.
Table 15-14.  Division factors of the Dead Time prescaler

<table>
<thead>
<tr>
<th>DTPS41</th>
<th>DTPS40</th>
<th>Prescaler divides the T/C4 clock by</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1x (no division)</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>2x</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>4x</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>8x</td>
</tr>
</tbody>
</table>

• Bits 3 .. 0 - CS43, CS42, CS41, CS40: Clock Select Bits 3, 2, 1, and 0
The Clock Select bits 3, 2, 1, and 0 define the prescaling source of Timer/Counter4.

Table 15-15.  Timer/Counter4 Prescaler Select

<table>
<thead>
<tr>
<th>CS43</th>
<th>CS42</th>
<th>CS41</th>
<th>CS40</th>
<th>Asynchronous Clocking Mode</th>
<th>Synchronous Clocking Mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>T/C4 stopped</td>
<td>T/C4 stopped</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>PCK</td>
<td>CK</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>PCK/2</td>
<td>CK/2</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>PCK/4</td>
<td>CK/4</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>PCK/8</td>
<td>CK/8</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>PCK/16</td>
<td>CK/16</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>PCK/32</td>
<td>CK/32</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>PCK/64</td>
<td>CK/64</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>PCK/128</td>
<td>CK/128</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>PCK/256</td>
<td>CK/256</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>PCK/512</td>
<td>CK/512</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>PCK/1024</td>
<td>CK/1024</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>PCK/2048</td>
<td>CK/2048</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>PCK/4096</td>
<td>CK/4096</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>PCK/8192</td>
<td>CK/8192</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>PCK/16384</td>
<td>CK/16384</td>
</tr>
</tbody>
</table>

The Stop condition provides a Timer Enable/Disable function.

15.12.3  TCCR4C – Timer/Counter4 Control Register C

<table>
<thead>
<tr>
<th>Bit</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read/Write</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
</tr>
<tr>
<td>Initial value</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

• Bits 7,6 - COM4A1S, COM4A0S: Comparator A Output Mode, Bits 1 and 0
These bits are the shadow bits of the COM4A1 and COM4A0 bits that are described in the section “TCCR4A – Timer/Counter4 Control Register A” on page 162.
• **Bits 5,4 - COM4B1S, COM4B0S: Comparator B Output Mode, Bits 1 and 0**
These bits are the shadow bits of the COM4A1 and COM4A0 bits that are described in the section “TCCR4A – Timer/Counter4 Control Register A” on page 162.

• **Bits 3,2 - COM4D1, COM4D0: Comparator D Output Mode, Bits 1 and 0**
These bits control the behavior of the Waveform Output (OCW4D) and the connection of the Output Compare pin (OC4D). If one or both of the COM4D1:0 bits are set, the OC4D output overrides the normal port functionality of the I/O pin it is connected to. The complementary OC4D output is connected only in PWM modes when the COM4D1:0 bits are set to “01”. Note that the Data Direction Register (DDR) bit corresponding to the OC4D pin must be set in order to enable the output driver.

The function of the COM4D1:0 bits depends on the PWM4D and WGM40 bit settings. **Table 15-16** shows the COM4D1:0 bit functionality when the PWM4D bit is set to a Normal Mode (non-PWM).

**Table 15-16. Compare Output Mode, Normal Mode (non-PWM)**

<table>
<thead>
<tr>
<th>COM4D1..0</th>
<th>OCW4D Behavior</th>
<th>OC4D Pin</th>
<th>OC4D Pin</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>Normal port operation.</td>
<td>Disconnected</td>
<td>Disconnected</td>
</tr>
<tr>
<td>01</td>
<td>Toggle on Compare Match.</td>
<td>Connected</td>
<td>Disconnected</td>
</tr>
<tr>
<td>10</td>
<td>Clear on Compare Match.</td>
<td>Connected</td>
<td>Disconnected</td>
</tr>
<tr>
<td>11</td>
<td>Set on Compare Match.</td>
<td>Connected</td>
<td>Disconnected</td>
</tr>
</tbody>
</table>

**Table 15-17** shows the COM4D1:0 bit functionality when the PWM4D and WGM40 bits are set to Fast PWM Mode.

**Table 15-17. Compare Output Mode, Fast PWM Mode**

<table>
<thead>
<tr>
<th>COM4D1..0</th>
<th>OCW4D Behavior</th>
<th>OC4D Pin</th>
<th>OC4D Pin</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>Normal port operation.</td>
<td>Disconnected</td>
<td>Disconnected</td>
</tr>
<tr>
<td>01</td>
<td>Cleared on Compare Match. Set when TCNT4 = 0x000.</td>
<td>Connected</td>
<td>Connected</td>
</tr>
<tr>
<td>10</td>
<td>Cleared on Compare Match. Set when TCNT4 = 0x000.</td>
<td>Connected</td>
<td>Disconnected</td>
</tr>
<tr>
<td>11</td>
<td>Set on Compare Match. Clear when TCNT4 = 0x000.</td>
<td>Connected</td>
<td>Disconnected</td>
</tr>
</tbody>
</table>

**Table 15-18 on page 168** shows the COM4D1:0 bit functionality when the PWM4D and WGM40 bits are set to Phase and Frequency Correct PWM Mode.
Table 15-18. Compare Output Mode, Phase and Frequency Correct PWM Mode

<table>
<thead>
<tr>
<th>COM4D1..0</th>
<th>OCW4D Behavior</th>
<th>OC4D Pin</th>
<th>OC4D Pin</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>Normal port operation.</td>
<td>Disconnected</td>
<td>Disconnected</td>
</tr>
<tr>
<td>01</td>
<td>Cleared on Compare Match when up-counting. Set on Compare Match when down-counting.</td>
<td>Connected</td>
<td>Connected</td>
</tr>
<tr>
<td>10</td>
<td>Cleared on Compare Match when up-counting. Set on Compare Match when down-counting.</td>
<td>Connected</td>
<td>Disconnected</td>
</tr>
<tr>
<td>11</td>
<td>Set on Compare Match when up-counting. Cleared on Compare Match when down-counting.</td>
<td>Connected</td>
<td>Disconnected</td>
</tr>
</tbody>
</table>

- **Bit 1 - FOC4D: Force Output Compare Match 4D**
  The FOC4D bit is only active when the PWM4D bit specify a non-PWM mode.
  Writing a logical one to this bit forces a change in the Waveform Output (OCW4D) and the Output Compare pin (OC4D) according to the values already set in COM4D1 and COM4D0. If COM4D1 and COM4D0 written in the same cycle as FOC4D, the new settings will be used. The Force Output Compare bit can be used to change the output pin value regardless of the timer value. The automatic action programmed in COM4D1 and COM4D0 takes place as if a compare match had occurred, but no interrupt is generated. The FOC4D bit is always read as zero.

- **Bit 0 - PWM4D: Pulse Width Modulator D Enable**
  When set (one) this bit enables PWM mode based on comparator OCR4D.

### 15.12.4 TCCR4D – Timer/Counter4 Control Register D

<table>
<thead>
<tr>
<th>Bit</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read/Write</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
</tr>
<tr>
<td>Initial value</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

- **Bit 7 - FPIE4: Fault Protection Interrupt Enable**
  Setting this bit (to one) enables the Fault Protection Interrupt.

- **Bit 6 – FPEN4: Fault Protection Mode Enable**
  Setting this bit (to one) activates the Fault Protection Mode.

- **Bit 5 – FPNC4: Fault Protection Noise Canceler**
  Setting this bit activates the Fault Protection Noise Canceler. When the noise canceler is activated, the input from the Fault Protection Pin (INT0) is filtered. The filter function requires four successive equal valued samples of the INTO pin for changing its output. The Fault Protection is therefore delayed by four Oscillator cycles when the noise canceler is enabled.

- **Bit 4 – FPES4: Fault Protection Edge Select**
  This bit selects which edge on the Fault Protection Pin (INT0) is used to trigger a fault event. When the FPES4 bit is written to zero, a falling (negative) edge is used as trigger, and when the FPES4 bit is written to one, a rising (positive) edge will trigger the fault.
• Bit 3 - FPAC4: Fault Protection Analog Comparator Enable
When written logic one, this bit enables the Fault Protection function in Timer/Counter4 to be triggered by the Analog Comparator. The comparator output is in this case directly connected to the Fault Protection front-end logic, making the comparator utilize the noise canceler and edge select features of the Timer/Counter4 Fault Protection interrupt. When written logic zero, no connection between the Analog Comparator and the Fault Protection function exists. To make the comparator trigger the Timer/Counter4 Fault Protection interrupt, the FPIE4 bit in the Timer/Counter4 Control Register D (TCCR4D) must be set.

• Bit 2- FPF4: Fault Protection Interrupt Flag
When the FPIE4 bit is set (one), the Fault Protection Interrupt is enabled. Activity on the pin will cause an interrupt request even, if the Fault Protection pin is configured as an output. The corresponding interrupt of Fault Protection Interrupt Request is executed from the Fault Protection Interrupt Vector. The bit FPF4 is cleared by hardware when executing the corresponding interrupt handling vector. Alternatively, FPF4 is cleared after a synchronization clock cycle by writing a logical one to the flag. When the SREG I-bit, FPIE4 and FPF4 are set, the Fault Interrupt is executed.

• Bits 1:0 - WGM41, WGM40: Waveform Generation Mode Bits
This bit associated with the PWM4x bits control the counting sequence of the counter, the source for type of waveform generation to be used, see Table 15-19. Modes of operation supported by the Timer/Counter4 are: Normal mode (counter), Fast PWM Mode, Phase and Frequency Correct PWM and PWM6 Modes.

Table 15-19. Waveform Generation Mode Bit Description

<table>
<thead>
<tr>
<th>PWM4x</th>
<th>WGM41..40</th>
<th>Timer/Counter Mode of Operation</th>
<th>TOP Update of OCR4x at</th>
<th>TOV4 Flag Set on</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>xx</td>
<td>Normal</td>
<td>OCR4C Immediate</td>
<td>TOP</td>
</tr>
<tr>
<td>1</td>
<td>00</td>
<td>Fast PWM</td>
<td>OCR4C TOP</td>
<td>TOP</td>
</tr>
<tr>
<td>1</td>
<td>01</td>
<td>Phase and Frequency Correct PWM</td>
<td>OCR4C BOTTOM</td>
<td>BOTTOM</td>
</tr>
<tr>
<td>1</td>
<td>10</td>
<td>PWM6 / Single-slope</td>
<td>OCR4C TOP</td>
<td>TOP</td>
</tr>
<tr>
<td>1</td>
<td>11</td>
<td>PWM6 / Dual-slope</td>
<td>OCR4C BOTTOM</td>
<td>BOTTOM</td>
</tr>
</tbody>
</table>

15.12.5 TCCR4E – Timer/Counter4 Control Register E

<table>
<thead>
<tr>
<th>Bit</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>TLOCK4</td>
<td>ENHC4</td>
<td>OC4OE5</td>
<td>OC4OEU</td>
<td>OC4OE3</td>
<td>OC4OE2</td>
<td>OC4OE1</td>
<td>OC4OE0</td>
</tr>
<tr>
<td>Read/Write</td>
<td>R</td>
<td>R</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
</tr>
<tr>
<td>Initial value</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

• Bit 7 - TLOCK4: Register Update Lock
This bit controls the Compare registers update. When this bit is set, writing to the Compare registers will not affect the output, however the values are stored and will be updated to the Compare registers when the TLOCK4 bit will be cleared.

Refer to Section 15.7 "Synchronous update" on page 149 for more details.
• **Bit 6- ENHC4: Enhanced Compare/PWM Mode**  
  When this bit is set, the Waveform Generation Module works in enhanced mode: the compare registers OCR4A/B/D can welcome one more accuracy bit, while the LSB determines on which clock edge the Compare condition is signalled and the output pin level is updated.

• **Bits 5:0 – OC4OE5:OC4OE0: Output Compare Override Enable Bits**  
  These bits are the Output Compare Override Enable bits that are used to connect or disconnect the Output Compare Pins in PWM6 Modes with an instant response on the corresponding Output Compare Pins. The actual value from the port register will be visible on the port pin, when the Output Compare Override Enable Bit is cleared. Table 15-20 shows the Output Compare Override Enable Bits and their corresponding Output Compare pins.

  **Table 15-20.** Output Compare Override Enable Bits vs. Output Compare Pins

<table>
<thead>
<tr>
<th>OC4OE0</th>
<th>OC4OE1</th>
<th>OC4OE2</th>
<th>OC4OE3</th>
<th>OC4OE4</th>
<th>OC4OE5</th>
</tr>
</thead>
<tbody>
<tr>
<td>OC4A (PC6)</td>
<td>OC4A (PC7)</td>
<td>OC4B (PB5)</td>
<td>OC4B (PB6)</td>
<td>OC4D (PD6)</td>
<td>OC4D (PD7)</td>
</tr>
</tbody>
</table>

15.12.6 **TCNT4 – Timer/Counter4**

<table>
<thead>
<tr>
<th>Bit</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read/Write</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
</tr>
<tr>
<td>Initial value</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

This 8-bit register contains the value of Timer/Counter4.

The Timer/Counter4 is realized as a 10-bit up/down counter with read and write access. Due to synchronization of the CPU, Timer/Counter4 data written into Timer/Counter4 is delayed by one and half CPU clock cycles in synchronous mode and at most one CPU clock cycles for asynchronous mode. When a 10-bit accuracy is preferred, special procedures must be followed for accessing the 10-bit TCNT4 register via the 8-bit AVR data bus. These procedures are described in section “Accessing 10-Bit Registers” on page 159. Alternatively the Timer/Counter4 can be used as an 8-bit Timer/Counter. Note that the Timer/Counter4 always starts counting up after writing the TCNT4 register.

15.12.7 **TC4H – Timer/Counter4 High Byte**

<table>
<thead>
<tr>
<th>Bit</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read/Write</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
</tr>
<tr>
<td>Initial value</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

The temporary Timer/Counter4 register is an 2-bit read/write register.

• **Bits 7:3- Res: Reserved Bits**  
  These bits are reserved bits in the ATmega16U4/ATmega32U4 and always reads as zero.

• **Bits 2- TC410: Additional MSB bits for 11-bit accesses in Enhanced PWM mode**  
  If 10-bit accuracy is used, the Timer/Counter4 High Byte Register (TC4H) is used for temporary storing the MSB bits (TC49, TC48) of the 10-bit accesses. The same TC4H register is shared between all 10-bit registers within the Timer/Counter4. Note that special procedures must be followed when accessing the 10-bit TCNT4 register via the 8-bit AVR data bus. These procedures are described in section “Accessing 10-Bit Registers” on page 159.
• Bits 1:0 - TC49, TC48: Two MSB bits of the 10-bit accesses
If 10-bit accuracy is used, the Timer/Counter4 High Byte Register (TC4H) is used for temporary storing the MSB bits (TC49, TC48) of the 10-bit accesses. The same TC4H register is shared between all 10-bit registers within the Timer/Counter4. Note that special procedures must be followed when accessing the 10-bit TCNT4 register via the 8-bit AVR data bus. These procedures are described in section “Accessing 10-Bit Registers” on page 159.

15.12.8 OCR4A – Timer/Counter4 Output Compare Register A

<table>
<thead>
<tr>
<th>Bit</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
<th>Read/Write</th>
<th>Initial value</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSB</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>0 0 0 0 0 0 0 0</td>
</tr>
<tr>
<td>LSB</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>0 0 0 0 0 0 0 0</td>
</tr>
</tbody>
</table>

The output compare register A is an 8-bit read/write register.

The Timer/Counter Output Compare Register A contains data to be continuously compared with Timer/Counter4. Actions on compare matches are specified in TCCR4A. A compare match does only occur if Timer/Counter4 counts to the OCR4A value. A software write that sets TCNT4 and OCR4A to the same value does not generate a compare match.

A compare match will set the compare interrupt flag OCF4A after a synchronization delay following the compare event.

Note that, if 10-bit accuracy is used special procedures must be followed when accessing the internal 10-bit Output Compare Registers via the 8-bit AVR data bus. These procedures are described in section “Accessing 10-Bit Registers” on page 159.

15.12.9 OCR4B – Timer/Counter4 Output Compare Register B

<table>
<thead>
<tr>
<th>Bit</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
<th>Read/Write</th>
<th>Initial value</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSB</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>0 0 0 0 0 0 0 0</td>
</tr>
<tr>
<td>LSB</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>0 0 0 0 0 0 0 0</td>
</tr>
</tbody>
</table>

The output compare register B is an 8-bit read/write register.

The Timer/Counter Output Compare Register B contains data to be continuously compared with Timer/Counter4. Actions on compare matches are specified in TCCR4B. A compare match does only occur if Timer/Counter4 counts to the OCR4B value. A software write that sets TCNT4 and OCR4B to the same value does not generate a compare match.

A compare match will set the compare interrupt flag OCF4B after a synchronization delay following the compare event.

Note that, if 10-bit accuracy is used special procedures must be followed when accessing the internal 10-bit Output Compare Registers via the 8-bit AVR data bus. These procedures are described in section “Accessing 10-Bit Registers” on page 159.

15.12.10 OCR4C – Timer/Counter4 Output Compare Register C

<table>
<thead>
<tr>
<th>Bit</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
<th>Read/Write</th>
<th>Initial value</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSB</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>1 1 1 1 1 1 1 1</td>
</tr>
<tr>
<td>LSB</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>1 1 1 1 1 1 1 1</td>
</tr>
</tbody>
</table>
The output compare register C is an 8-bit read/write register.

The Timer/Counter Output Compare Register C contains data to be continuously compared with Timer/Counter4, and a compare match will clear TCNT4. This register has the same function in Normal mode and PWM modes.

Note that, if a smaller value than three is written to the Output Compare Register C, the value is automatically replaced by three as it is a minimum value allowed to be written to this register.

Note that, if 10-bit accuracy is used special procedures must be followed when accessing the internal 10-bit Output Compare Registers via the 8-bit AVR data bus. These procedures are described in section "Accessing 10-Bit Registers" on page 159.

15.12.11 OCR4D – Timer/Counter4 Output Compare Register D

<table>
<thead>
<tr>
<th>Bit</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read/Write</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
</tr>
</tbody>
</table>

The output compare register D is an 8-bit read/write register.

The Timer/Counter Output Compare Register D contains data to be continuously compared with Timer/Counter4. Actions on compare matches are specified in TCCR4A. A compare match does only occur if Timer/Counter4 counts to the OCR4D value. A software write that sets TCNT4 and OCR4D to the same value does not generate a compare match.

A compare match will set the compare interrupt flag OCF4D after a synchronization delay following the compare event.

Note that, if 10-bit accuracy is used special procedures must be followed when accessing the internal 10-bit Output Compare Registers via the 8-bit AVR data bus. These procedures are described in section "Accessing 10-Bit Registers" on page 159.

15.12.12 TIMSK4 – Timer/Counter4 Interrupt Mask Register

<table>
<thead>
<tr>
<th>Bit</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read/Write</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
</tr>
</tbody>
</table>

- **Bit 7 - OCIE4D: Timer/Counter4 Output Compare Interrupt Enable**
When the OCIE4D bit is set (one) and the I-bit in the Status Register is set (one), the Timer/Counter4 Compare Match D interrupt is enabled. The corresponding interrupt at vector $010$ is executed if a compare match D occurs. The Compare Flag in Timer/Counter4 is set (one) in the Timer/Counter Interrupt Flag Register.

- **Bit 6 - OCIE4A: Timer/Counter4 Output Compare Interrupt Enable**
When the OCIE4A bit is set (one) and the I-bit in the Status Register is set (one), the Timer/Counter4 Compare Match A interrupt is enabled. The corresponding interrupt at vector $003$ is executed if a compare match A occurs. The Compare Flag in Timer/Counter4 is set (one) in the Timer/Counter Interrupt Flag Register.
• **Bit 5 - OCIE4B: Timer/Counter4 Output Compare Interrupt Enable**

When the OCIE4B bit is set (one) and the I-bit in the Status Register is set (one), the Timer/Counter4 Compare Match B interrupt is enabled. The corresponding interrupt at vector $009$ is executed if a compare match B occurs. The Compare Flag in Timer/Counter4 is set (one) in the Timer/Counter Interrupt Flag Register.

• **Bit 2 - TOIE4: Timer/Counter4 Overflow Interrupt Enable**

When the TOIE4 bit is set (one) and the I-bit in the Status Register is set (one), the Timer/Counter4 Overflow interrupt is enabled. The corresponding interrupt (at vector $004$) is executed if an overflow in Timer/Counter4 occurs. The Overflow Flag (Timer4) is set (one) in the Timer/Counter Interrupt Flag Register - TIFR4.

### 15.12.13 TIFR4 – Timer/Counter4 Interrupt Flag Register

<table>
<thead>
<tr>
<th>Bit</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>OCF4D</td>
<td>OCF4A</td>
<td>OCF4B</td>
<td>TOV4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Read/Write</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td></td>
</tr>
<tr>
<td>Initial value</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

• **Bit 7 - OCF4D: Output Compare Flag 4D**

The OCF4D bit is set (one) when compare match occurs between Timer/Counter4 and the data value in OCR4D - Output Compare Register 4D. OCF4D is cleared by hardware when executing the corresponding interrupt handling vector. Alternatively, OCF4D is cleared, after synchronization clock cycle, by writing a logic one to the flag. When the I-bit in SREG, OCIE4D, and OCF4D are set (one), the Timer/Counter4 D compare match interrupt is executed.

• **Bit 6 - OCF4A: Output Compare Flag 4A**

The OCF4A bit is set (one) when compare match occurs between Timer/Counter4 and the data value in OCR4A - Output Compare Register 4A. OCF4A is cleared by hardware when executing the corresponding interrupt handling vector. Alternatively, OCF4A is cleared, after synchronization clock cycle, by writing a logic one to the flag. When the I-bit in SREG, OCIE4A, and OCF4A are set (one), the Timer/Counter4 A compare match interrupt is executed.

• **Bit 5 - OCF4B: Output Compare Flag 4B**

The OCF4B bit is set (one) when compare match occurs between Timer/Counter4 and the data value in OCR4B - Output Compare Register 4B. OCF4B is cleared by hardware when executing the corresponding interrupt handling vector. Alternatively, OCF4B is cleared, after synchronization clock cycle, by writing a logic one to the flag. When the I-bit in SREG, OCIE4B, and OCF4B are set (one), the Timer/Counter4 B compare match interrupt is executed.

• **Bit 2 - TOV4: Timer/Counter4 Overflow Flag**

In Normal Mode and Fast PWM Mode the TOV4 bit is set (one) each time the counter reaches TOP at the same clock cycle when the counter is reset to BOTTOM. In Phase and Frequency Correct PWM Mode the TOV4 bit is set (one) each time the counter reaches BOTTOM at the same clock cycle when zero is clocked to the counter.

The bit TOV4 is cleared by hardware when executing the corresponding interrupt handling vector. Alternatively, TOV4 is cleared, after synchronization clock cycle, by writing a logical one to the flag. When the SREG I-bit, and TOIE4 (Timer/Counter4 Overflow Interrupt Enable), and TOV4 are set (one), the Timer/Counter4 Overflow interrupt is executed.
15.12.14 DT4 – Timer/Counter4 Dead Time Value

<table>
<thead>
<tr>
<th>Bit</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>DT4H3</td>
<td>DT4H2</td>
<td>DT4H1</td>
<td>DT4H0</td>
<td>DT4L3</td>
<td>DT4L2</td>
<td>DT4L1</td>
<td>DT4L0</td>
</tr>
<tr>
<td>Read/Write</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
</tr>
<tr>
<td>Initial value</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

The dead time value register is an 8-bit read/write register.

The dead time delay of all Timer/Counter4 channels are adjusted by the dead time value register, DT4. The register consists of two fields, DT4H3..0 and DT4L3..0, one for each complementary output. Therefore a different dead time delay can be adjusted for the rising edge of OC4x and the rising edge of OC4x.

- **Bits 7:4- DT4H3:DT4H0: Dead Time Value for OC4x Output**
  The dead time value for the OC1x output. The dead time delay is set as a number of the prescaled timer/counter clocks. The minimum dead time is zero and the maximum dead time is the prescaled time/counter clock period multiplied by 15.

- **Bits 3:0- DT4L3:DT4L0: Dead Time Value for OC4x Output**
  The dead time value for the OC4x output. The dead time delay is set as a number of the prescaled timer/counter clocks. The minimum dead time is zero and the maximum dead time is the prescaled time/counter clock period multiplied by 15.
16. Output Compare Modulator (OCM1C0A)

16.1 Overview

The Output Compare Modulator (OCM) allows generation of waveforms modulated with a carrier frequency. The modulator uses the outputs from the Output Compare Unit C of the 16-bit Timer/Counter1 and the Output Compare Unit of the 8-bit Timer/Counter0. For more details about these Timer/Counters see “Timer/Counter0, Timer/Counter1, and Timer/Counter3 Prescalers” on page 89.

Figure 16-1. Output Compare Modulator, Block Diagram

When the modulator is enabled, the two output compare channels are modulated together as shown in the block diagram (Figure 16-1).

16.2 Description

The Output Compare unit 1C and Output Compare unit 2 shares the PB7 port pin for output. The outputs of the Output Compare units (OC1C and OC0A) overrides the normal PORTB7 Register when one of them is enabled (i.e., when COMnx1:0 is not equal to zero). When both OC1C and OC0A are enabled at the same time, the modulator is automatically enabled.

The functional equivalent schematic of the modulator is shown on Figure 16-2. The schematic includes part of the Timer/Counter units and the port B pin 7 output driver circuit.

Figure 16-2. Output Compare Modulator, Schematic
When the modulator is enabled the type of modulation (logical AND or OR) can be selected by the PORTB7 Register. Note that the DDRB7 controls the direction of the port independent of the COMnx1:0 bit setting.

16.2.1 Timing Example

Figure 16-3 illustrates the modulator in action. In this example the Timer/Counter1 is set to operate in fast PWM mode (non-inverted) and Timer/Counter0 uses CTC waveform mode with toggle Compare Output mode (COMnx1:0 = 1).

Figure 16-3. Output Compare Modulator, Timing Diagram

In this example, Timer/Counter0 provides the carrier, while the modulating signal is generated by the Output Compare unit C of the Timer/Counter1.

The resolution of the PWM signal (OC1C) is reduced by the modulation. The reduction factor is equal to the number of system clock cycles of one period of the carrier (OC0A). In this example the resolution is reduced by a factor of two. The reason for the reduction is illustrated in Figure 16-3 at the second and third period of the PB7 output when PORTB7 equals zero. The period 2 high time is one cycle longer than the period 3 high time, but the result on the PB7 output is equal in both periods.
17. Serial Peripheral Interface – SPI

The Serial Peripheral Interface (SPI) allows high-speed synchronous data transfer between the ATmega16U4/ATmega32U4 and peripheral devices or between several AVR devices. The ATmega16U4/ATmega32U4 SPI includes the following features:

- Full-duplex, Three-wire Synchronous Data Transfer
- Master or Slave Operation
- LSB First or MSB First Data Transfer
- Seven Programmable Bit Rates
- End of Transmission Interrupt Flag
- Write Collision Flag Protection
- Wake-up from Idle Mode
- Double Speed (CK/2) Master SPI Mode

USART can also be used in Master SPI mode, see “USART in SPI Mode” on page 214.

The Power Reduction SPI bit, PRSPI, in “Power Reduction Register 0 - PRR0” on page 46 on page 50 must be written to zero to enable SPI module.

Figure 17-1. SPI Block Diagram

---

Note: 1. Refer to “Pinout ATmega16U4/ATmega32U4” on page 3, and Table 10-3 on page 72 for SPI pin placement.

The interconnection between Master and Slave CPUs with SPI is shown in Figure 17-2. The system consists of two shift Registers, and a Master clock generator. The SPI Master initiates the
communication cycle when pulling low the Slave Select SS pin of the desired Slave. Master and Slave prepare the data to be sent in their respective shift Registers, and the Master generates the required clock pulses on the SCK line to interchange data. Data is always shifted from Master to Slave on the Master Out – Slave In, MOSI, line, and from Slave to Master on the Master In – Slave Out, MISO, line. After each data packet, the Master will synchronize the Slave by pulling high the Slave Select, SS, line.

When configured as a Master, the SPI interface has no automatic control of the SS line. This must be handled by user software before communication can start. When this is done, writing a byte to the SPI Data Register starts the SPI clock generator, and the hardware shifts the eight bits into the Slave. After shifting one byte, the SPI clock generator stops, setting the end of Transmission Flag (SPIF). If the SPI Interrupt Enable bit (SPIE) in the SPCR Register is set, an interrupt is requested. The Master may continue to shift the next byte by writing it into SPDR, or signal the end of packet by pulling high the Slave Select, SS line. The last incoming byte will be kept in the Buffer Register for later use.

When configured as a Slave, the SPI interface will remain sleeping with MISO tri-stated as long as the SS pin is driven high. In this state, software may update the contents of the SPI Data Register, SPDR, but the data will not be shifted out by incoming clock pulses on the SCK pin until the SS pin is driven low. As one byte has been completely shifted, the end of Transmission Flag, SPIF is set. If the SPI Interrupt Enable bit, SPIE, in the SPCR Register is set, an interrupt is requested. The Slave may continue to place new data to be sent into SPDR before reading the incoming data. The last incoming byte will be kept in the Buffer Register for later use.

**Figure 17-2. SPI Master-slave Interconnection**

The system is single buffered in the transmit direction and double buffered in the receive direction. This means that bytes to be transmitted cannot be written to the SPI Data Register before the entire shift cycle is completed. When receiving data, however, a received character must be read from the SPI Data Register before the next character has been completely shifted in. Otherwise, the first byte is lost.

In SPI Slave mode, the control logic will sample the incoming signal of the SCK pin. To ensure correct sampling of the clock signal, the frequency of the SPI clock should never exceed fosc/4.
When the SPI is enabled, the data direction of the MOSI, MISO, SCK, and S5 pins is overridden according to Table 17-1. For more details on automatic port overrides, refer to “Alternate Port Functions” on page 70.

Table 17-1. SPI Pin Overrides

<table>
<thead>
<tr>
<th>Pin</th>
<th>Direction, Master SPI</th>
<th>Direction, Slave SPI</th>
</tr>
</thead>
<tbody>
<tr>
<td>MOSI</td>
<td>User Defined</td>
<td>Input</td>
</tr>
<tr>
<td>MISO</td>
<td>Input</td>
<td>User Defined</td>
</tr>
<tr>
<td>SCK</td>
<td>User Defined</td>
<td>Input</td>
</tr>
<tr>
<td>S5</td>
<td>User Defined</td>
<td>Input</td>
</tr>
</tbody>
</table>

Note: 1. See “Alternate Functions of Port B” on page 72 for a detailed description of how to define the direction of the user defined SPI pins.

The following code examples show how to initialize the SPI as a Master and how to perform a simple transmission. DDR_SPI in the examples must be replaced by the actual Data Direction Register controlling the SPI pins. DD_MOSI, DD_MISO and DD_SCK must be replaced by the actual data direction bits for these pins. E.g. if MOSI is placed on pin PB5, replace DD_MOSI with DDB5 and DDR_SPI with DDRB.
Assembly Code Example

SPI_MasterInit:
; Set MOSI and SCK output, all others input
ldi r17,(1<<DD_MOSI)|(1<<DD_SCK)
out DDR_SPI,r17
; Enable SPI, Master, set clock rate fck/16
ldi r17,(1<<SPE)|(1<<MSTR)|(1<<SPR0)
out SPCR,r17
ret

SPI_MasterTransmit:
; Start transmission of data (r16)
out SPDR,r16
Wait_Transmit:
; Wait for transmission complete
sbis SPSR,SPIF
rjmp Wait_Transmit
ret

C Code Example

void SPI_MasterInit(void)
{
    /* Set MOSI and SCK output, all others input */
    DDR_SPI = (1<<DD_MOSI)|(1<<DD_SCK);
    /* Enable SPI, Master, set clock rate fck/16 */
    SPCR = (1<<SPE)|(1<<MSTR)|(1<<SPR0);
}

void SPI_MasterTransmit(char cData)
{
    /* Start transmission */
    SPDR = cData;
    /* Wait for transmission complete */
    while(!(SPSR & (1<<SPIF)))
        ;
}

Note: 1. See “Code Examples” on page 8.
The following code examples show how to initialize the SPI as a Slave and how to perform a simple reception.

**Assembly Code Example**

```
Assembly Code Example

SPI_SlaveInit:
    ; Set MISO output, all others input
    ldi  r17,(1<<DD_MISO)
    out  DDR_SPI,r17
    ; Enable SPI
    ldi  r17,(1<<SPE)
    out  SPCR,r17
    ret

SPI_SlaveReceive:
    ; Wait for reception complete
    sbis SPSR,SPIF
    rjmp SPI_SlaveReceive
    ; Read received data and return
    in  r16,SPDR
    ret
```

**C Code Example**

```
C Code Example

void SPI_SlaveInit(void)
{
    /* Set MISO output, all others input */
    DDR_SPI = (1<<DD_MISO);
    /* Enable SPI */
    SPCR = (1<<SPE);
}

char SPI_SlaveReceive(void)
{
    /* Wait for reception complete */
    while(!(SPSR & (1<<SIF)))
    ;
    /* Return Data Register */
    return SPDR;
}
```

Note: 1. See “Code Examples” on page 8.

17.1 SS Pin Functionality

17.1.1 Slave Mode

When the SPI is configured as a Slave, the Slave Select (SS) pin is always input. When SS is held low, the SPI is activated, and MISO becomes an output if configured so by the user. All
other pins are inputs. When S5 is driven high, all pins are inputs, and the SPI is passive, which means that it will not receive incoming data. Note that the SPI logic will be reset once the S5 pin is driven high.

The S5 pin is useful for packet/byte synchronization to keep the slave bit counter synchronous with the master clock generator. When the S5 pin is driven high, the SPI slave will immediately reset the send and receive logic, and drop any partially received data in the Shift Register.

17.1.2 Master Mode

When the SPI is configured as a Master (MSTR in SPCR is set), the user can determine the direction of the S5 pin.

If S5 is configured as an output, the pin is a general output pin which does not affect the SPI system. Typically, the pin will be driving the S5 pin of the SPI Slave.

If S5 is configured as an input, it must be held high to ensure Master SPI operation. If the S5 pin is driven low by peripheral circuitry when the SPI is configured as a Master with the S5 pin defined as an input, the SPI system interprets this as another master selecting the SPI as a slave and starting to send data to it. To avoid bus contention, the SPI system takes the following actions:

1. The MSTR bit in SPCR is cleared and the SPI system becomes a Slave. As a result of the SPI becoming a Slave, the MOSI and SCK pins become inputs.
2. The SPIF Flag in SPSR is set, and if the SPI interrupt is enabled, and the I-bit in SREG is set, the interrupt routine will be executed.

Thus, when interrupt-driven SPI transmission is used in Master mode, and there exists a possibility that S5 is driven low, the interrupt should always check that the MSTR bit is still set. If the MSTR bit has been cleared by a slave select, it must be set by the user to re-enable SPI Master mode.

17.1.3 SPI Control Register – SPCR

<table>
<thead>
<tr>
<th>Bit</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>SPI</td>
<td>R/W</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SPE</td>
<td>R/W</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DORD</td>
<td>R/W</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MSTR</td>
<td>R/W</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CPOL</td>
<td>R/W</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CPHA</td>
<td>R/W</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SPR1</td>
<td>R/W</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SPR0</td>
<td>R/W</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SPCR</td>
<td>R/W</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **Bit 7 – SPIE: SPI Interrupt Enable**
  This bit causes the SPI interrupt to be executed if SPIF bit in the SPSR Register is set and the if the Global Interrupt Enable bit in SREG is set.

- **Bit 6 – SPE: SPI Enable**
  When the SPE bit is written to one, the SPI is enabled. This bit must be set to enable any SPI operations.

- **Bit 5 – DORD: Data Order**
  When the DORD bit is written to one, the LSB of the data word is transmitted first.
  When the DORD bit is written to zero, the MSB of the data word is transmitted first.

- **Bit 4 – MSTR: Master/Slave Select**
  This bit selects Master SPI mode when written to one, and Slave SPI mode when written logic zero. If S5 is configured as an input and is driven low while MSTR is set, MSTR will be cleared,
and SPIF in SPSR will become set. The user will then have to set MSTR to re-enable SPI Master mode.

- **Bit 3 – CPOL: Clock Polarity**
  When this bit is written to one, SCK is high when idle. When CPOL is written to zero, SCK is low when idle. Refer to Figure 17-3 and Figure 17-4 for an example. The CPOL functionality is summarized below:

<table>
<thead>
<tr>
<th>CPOL</th>
<th>Leading Edge</th>
<th>Trailing Edge</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Rising</td>
<td>Falling</td>
</tr>
<tr>
<td>1</td>
<td>Falling</td>
<td>Rising</td>
</tr>
</tbody>
</table>

- **Bit 2 – CPHA: Clock Phase**
  The settings of the Clock Phase bit (CPHA) determine if data is sampled on the leading (first) or trailing (last) edge of SCK. Refer to Figure 17-3 and Figure 17-4 for an example. The CPOL functionality is summarized below:

<table>
<thead>
<tr>
<th>CPHA</th>
<th>Leading Edge</th>
<th>Trailing Edge</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Sample</td>
<td>Setup</td>
</tr>
<tr>
<td>1</td>
<td>Setup</td>
<td>Sample</td>
</tr>
</tbody>
</table>

- **Bits 1, 0 – SPR1, SPR0: SPI Clock Rate Select 1 and 0**
  These two bits control the SCK rate of the device configured as a Master. SPR1 and SPR0 have no effect on the Slave. The relationship between SCK and the Oscillator Clock frequency $f_{osc}$ is shown in the following table:

<table>
<thead>
<tr>
<th>SPI2X</th>
<th>SPR1</th>
<th>SPR0</th>
<th>SCK Frequency</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>$f_{osc}/4$</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>$f_{osc}/16$</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>$f_{osc}/64$</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>$f_{osc}/128$</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>$f_{osc}/2$</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>$f_{osc}/8$</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>$f_{osc}/32$</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>$f_{osc}/64$</td>
</tr>
</tbody>
</table>

### 17.1.4 SPI Status Register – SPSR

<table>
<thead>
<tr>
<th>Bit</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
<th>SPI2X</th>
<th>SPSR</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read/Write</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R/W</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Initial Value</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

---

**Table 17-2.** CPOL Functionality

**Table 17-3.** CPHA Functionality

**Table 17-4.** Relationship Between SCK and the Oscillator Frequency

---

---
- **Bit 7 – SPIF: SPI Interrupt Flag**
  When a serial transfer is complete, the SPIF Flag is set. An interrupt is generated if SPIE in SPCR is set and global interrupts are enabled. If SS is an input and is driven low when the SPI is in Master mode, this will also set the SPIF Flag. SPIF is cleared by hardware when executing the corresponding interrupt handling vector. Alternatively, the SPIF bit is cleared by first reading the SPI Status Register with SPIF set, then accessing the SPI Data Register (SPDR).

- **Bit 6 – WCOL: Write COLLision Flag**
  The WCOL bit is set if the SPI Data Register (SPDR) is written during a data transfer. The WCOL bit (and the SPIF bit) are cleared by first reading the SPI Status Register with WCOL set, and then accessing the SPI Data Register.

- **Bit 5..1 – Res: Reserved Bits**
  These bits are reserved bits in the ATmega16U4/ATmega32U4 and will always read as zero.

- **Bit 0 – SPI2X: Double SPI Speed Bit**
  When this bit is written logic one the SPI speed (SCK Frequency) will be doubled when the SPI is in Master mode (see Table 17-4). This means that the minimum SCK period will be two CPU clock periods. When the SPI is configured as Slave, the SPI is only guaranteed to work at f_{osc}/4 or lower.

  The SPI interface on the ATmega16U4/ATmega32U4 is also used for program memory and EEPROM downloading or uploading. See page 360 for serial programming and verification.

### 17.1.5 SPI Data Register – SPDR

<table>
<thead>
<tr>
<th>Bit</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td></td>
</tr>
<tr>
<td>Initial Value</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>Undefined</td>
</tr>
</tbody>
</table>

The SPI Data Register is a read/write register used for data transfer between the Register File and the SPI Shift Register. Writing to the register initiates data transmission. Reading the register causes the Shift Register Receive buffer to be read.

### 17.2 Data Modes

There are four combinations of SCK phase and polarity with respect to serial data, which are determined by control bits CPHA and CPOL. The SPI data transfer formats are shown in Figure 17-3 and Figure 17-4. Data bits are shifted out and latched in on opposite edges of the SCK signal, ensuring sufficient time for data signals to stabilize. This is clearly seen by summarizing Table 17-2 and Table 17-3, as done below:
Table 17-5.  CPOL Functionality

<table>
<thead>
<tr>
<th>CPOL = 0, CPHA = 0</th>
<th>Leading Edge</th>
<th>Trailing Edge</th>
<th>SPI Mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sample (Rising)</td>
<td>Setup (Falling)</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>CPOL = 0, CPHA = 1</td>
<td>Setup (Rising)</td>
<td>Sample (Falling)</td>
<td>1</td>
</tr>
<tr>
<td>CPOL = 1, CPHA = 0</td>
<td>Sample (Falling)</td>
<td>Setup (Rising)</td>
<td>2</td>
</tr>
<tr>
<td>CPOL = 1, CPHA = 1</td>
<td>Setup (Falling)</td>
<td>Sample (Rising)</td>
<td>3</td>
</tr>
</tbody>
</table>

Figure 17-3.  SPI Transfer Format with CPHA = 0

Figure 17-4.  SPI Transfer Format with CPHA = 1
18. USART

The Universal Synchronous and Asynchronous serial Receiver and Transmitter (USART) is a highly flexible serial communication device. The main features are:

- Full Duplex Operation (Independent Serial Receive and Transmit Registers)
- Asynchronous or Synchronous Operation
- Flow control CTS/RTS signals hardware management
- Master or Slave Clocked Synchronous Operation
- High Resolution Baud Rate Generator
- Supports Serial Frames with 5, 6, 7, 8, or 9 Data Bits and 1 or 2 Stop Bits
- Odd or Even Parity Generation and Parity Check Supported by Hardware
- Data OverRun Detection
- Framing Error Detection
- Noise Filtering Includes False Start Bit Detection and Digital Low Pass Filter
- Three Separate Interrupts on TX Complete, TX Data Register Empty and RX Complete
- Multi-processor Communication Mode
- Double Speed Asynchronous Communication Mode

18.1 Overview

A simplified block diagram of the USART Transmitter is shown in Figure 18-1 on page 187. CPU accessible I/O Registers and I/O pins are shown in bold.
18.2 Clock Generation

The Clock Generation logic generates the base clock for the Transmitter and Receiver. The USARTn supports four modes of clock operation: Normal asynchronous, Double Speed asynchronous, Master synchronous and Slave synchronous mode. The UMSELn bit in USART Control and Status Register C (UCSRC) selects between asynchronous and synchronous operation. Double Speed (asynchronous mode only) is controlled by the U2Xn found in the

---

**Figure 18-1.** USART Block Diagram

The dashed boxes in the block diagram separate the three main parts of the USART (listed from the top): Clock Generator, Transmitter and Receiver. Control Registers are shared by all units. The Clock Generation logic consists of synchronization logic for external clock input used by synchronous slave operation, and the baud rate generator. The XCKn (Transfer Clock) pin is only used by synchronous transfer mode. The Transmitter consists of a single write buffer, a serial Shift Register, Parity Generator and Control logic for handling different serial frame formats. The write buffer allows a continuous transfer of data without any delay between frames. The Receiver is the most complex part of the USART module due to its clock and data recovery units. The recovery units are used for asynchronous data reception. In addition to the recovery units, the Receiver includes a Parity Checker, Control logic, a Shift Register and a two level receive buffer (UDRn). The Receiver supports the same frame formats as the Transmitter, and can detect Frame Error, Data OverRun and Parity Errors.

---

Note: 1. See "Pinout ATmega16U4/ATmega32U4" on page 3, Table 10-8 on page 77 and for USART pin placement.
UCSRnA Register. When using synchronous mode (UMSELn = 1), the Data Direction Register for the XCKn pin (DDR_XCKn) controls whether the clock source is internal (Master mode) or external (Slave mode). The XCKn pin is only active when using synchronous mode.

Figure 18-2 shows a block diagram of the clock generation logic.

### Figure 18-2. Clock Generation Logic, Block Diagram

Signal description:

- **txclk**: Transmitter clock (Internal Signal).
- **rxclk**: Receiver base clock (Internal Signal).
- **xcki**: Input from XCK pin (internal Signal). Used for synchronous slave operation.
- **xcko**: Clock output to XCK pin (Internal Signal). Used for synchronous master operation.
- **fosc**: XTAL pin frequency (System Clock).

### 18.2.1 Internal Clock Generation – The Baud Rate Generator

Internal clock generation is used for the asynchronous and the synchronous master modes of operation. The description in this section refers to Figure 18-2.

The USART Baud Rate Register (UBRRn) and the down-counter connected to it function as a programmable prescaler or baud rate generator. The down-counter, running at system clock (f_{osc}), is loaded with the UBRRn value each time the counter has counted down to zero or when the UBRRLn Register is written. A clock is generated each time the counter reaches zero. This clock is the baud rate generator clock output (= f_{osc}/(UBRRn+1)). The Transmitter divides the baud rate generator clock output by 2, 8 or 16 depending on mode. The baud rate generator output is used directly by the Receiver's clock and data recovery units. However, the recovery units use a state machine that uses 2, 8 or 16 states depending on mode set by the state of the UMSELn, U2Xn and DDR_XCKn bits.
Table 18-1 contains equations for calculating the baud rate (in bits per second) and for calculating the UBRRn value for each mode of operation using an internally generated clock source.

<table>
<thead>
<tr>
<th>Operating Mode</th>
<th>Equation for Calculating Baud Rate&lt;sup&gt;(1)&lt;/sup&gt;</th>
<th>Equation for Calculating UBRR Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Asynchronous Normal mode (U2Xn = 0)</td>
<td>( BAUD = \frac{f_{osc}}{16(UBRRn + 1)} )</td>
<td>( UBRRn = \frac{f_{osc}}{16BAUD} - 1 )</td>
</tr>
<tr>
<td>Asynchronous Double Speed mode (U2Xn = 1)</td>
<td>( BAUD = \frac{f_{osc}}{8(UBRRn + 1)} )</td>
<td>( UBRRn = \frac{f_{osc}}{8BAUD} - 1 )</td>
</tr>
<tr>
<td>Synchronous Master mode</td>
<td>( BAUD = \frac{f_{osc}}{2(UBRRn + 1)} )</td>
<td>( UBRRn = \frac{f_{osc}}{2BAUD} - 1 )</td>
</tr>
</tbody>
</table>

Note: 1. The baud rate is defined to be the transfer rate in bit per second (bps)

- **BAUD**: Baud rate (in bits per second, bps)
- **f_{osc}**: System Oscillator clock frequency
- **UBRRn**: Contents of the UBRRHn and UBRRLn Registers, (0-4095)

Some examples of UBRRn values for some system clock frequencies are found in Table 18-9 on page 210.

18.2.2 Double Speed Operation (U2Xn)

The transfer rate can be doubled by setting the U2Xn bit in UCSRN.A. Setting this bit only has effect for the asynchronous operation. Set this bit to zero when using synchronous operation.

Setting this bit will reduce the divisor of the baud rate divider from 16 to 8, effectively doubling the transfer rate for asynchronous communication. Note however that the Receiver will in this case only use half the number of samples (reduced from 16 to 8) for data sampling and clock recovery, and therefore a more accurate baud rate setting and system clock are required when this mode is used. For the Transmitter, there are no downsides.
18.2.3 External Clock

External clocking is used by the synchronous slave modes of operation. The description in this section refers to Figure 18-2 for details.

External clock input from the XCKn pin is sampled by a synchronization register to minimize the chance of meta-stability. The output from the synchronization register must then pass through an edge detector before it can be used by the Transmitter and Receiver. This process introduces a two CPU clock period delay and therefore the maximum external XCKn clock frequency is limited by the following equation:

\[ f_{\text{XCK}} < \frac{f_{\text{osc}}}{4} \]

Note that \( f_{\text{osc}} \) depends on the stability of the system clock source. It is therefore recommended to add some margin to avoid possible loss of data due to frequency variations.

18.2.4 Synchronous Clock Operation

When synchronous mode is used (UMSELn = 1), the XCKn pin will be used as either clock input (Slave) or clock output (Master). The dependency between the clock edges and data sampling or data change is the same. The basic principle is that data input (on RxDn) is sampled at the opposite XCKn clock edge of the edge the data output (TxDn) is changed.

Figure 18-3. Synchronous Mode XCKn Timing.

![Synchronous Mode XCKn Timing](image)

The UCPOLn bit UCRSC selects which XCKn clock edge is used for data sampling and which is used for data change. As Figure 18-3 shows, when UCPOLn is zero the data will be changed at rising XCKn edge and sampled at falling XCKn edge. If UCPOLn is set, the data will be changed at falling XCKn edge and sampled at rising XCKn edge.

18.3 Frame Formats

A serial frame is defined to be one character of data bits with synchronization bits (start and stop bits), and optionally a parity bit for error checking. The USART accepts all 30 combinations of the following as valid frame formats:

- 1 start bit
- 5, 6, 7, 8, or 9 data bits
- no, even or odd parity bit
- 1 or 2 stop bits
A frame starts with the start bit followed by the least significant data bit. Then the next data bits, up to a total of nine, are succeeding, ending with the most significant bit. If enabled, the parity bit is inserted after the data bits, before the stop bits. When a complete frame is transmitted, it can be directly followed by a new frame, or the communication line can be set to an idle (high) state. Figure 18-4 illustrates the possible combinations of the frame formats. Bits inside brackets are optional.

**Figure 18-4. Frame Formats**

<table>
<thead>
<tr>
<th>(IDLE)</th>
<th>St</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>[P]</th>
<th>Sp1</th>
<th>[Sp2]</th>
<th>(St/IDLE)</th>
</tr>
</thead>
</table>

- **St** Start bit, always low.
- **(n)** Data bits (0 to 8).
- **P** Parity bit. Can be odd or even.
- **Sp** Stop bit, always high.
- **IDLE** No transfers on the communication line (RxDn or TxDn). An IDLE line must be high.

The frame format used by the USART is set by the UCSZn2:0, UPMn1:0 and USBSn bits in UCSRnB and UCSRnC. The Receiver and Transmitter use the same setting. Note that changing the setting of any of these bits will corrupt all ongoing communication for both the Receiver and Transmitter.

The USART Character SiZe (UCSZn2:0) bits select the number of data bits in the frame. The USART Parity mode (UPMn1:0) bits enable and set the type of parity bit. The selection between one or two stop bits is done by the USART Stop Bit Select (USBSn) bit. The Receiver ignores the second stop bit. An FE (Frame Error) will therefore only be detected in the cases where the first stop bit is zero.

### 18.3.1 Parity Bit Calculation

The parity bit is calculated by doing an exclusive-or of all the data bits. If odd parity is used, the result of the exclusive or is inverted. The relation between the parity bit and data bits is as follows:

\[
P_{\text{even}} = d_{n-1} \oplus \ldots \oplus d_3 \oplus d_2 \oplus d_1 \oplus d_0 \oplus 0
\]

\[
P_{\text{odd}} = d_{n-1} \oplus \ldots \oplus d_3 \oplus d_2 \oplus d_1 \oplus d_0 \oplus 1
\]

- **P_{\text{even}}** Parity bit using even parity
- **P_{\text{odd}}** Parity bit using odd parity
- **d_n** Data bit n of the character

If used, the parity bit is located between the last data bit and first stop bit of a serial frame.
18.4 USART Initialization

The USART has to be initialized before any communication can take place. The initialization process normally consists of setting the baud rate, setting frame format and enabling the Transmitter or the Receiver depending on the usage. For interrupt driven USART operation, the Global Interrupt Flag should be cleared (and interrupts globally disabled) when doing the initialization.

Before doing a re-initialization with changed baud rate or frame format, be sure that there are no ongoing transmissions during the period the registers are changed. The TXCn Flag can be used to check that the Transmitter has completed all transfers, and the RXC Flag can be used to check that there are no unread data in the receive buffer. Note that the TXCn Flag must be cleared before each transmission (before UDRn is written) if it is used for this purpose.

The following simple USART initialization code examples show one assembly and one C function that are equal in functionality. The examples assume asynchronous operation using polling (no interrupts enabled) and a fixed frame format. The baud rate is given as a function parameter. For the assembly code, the baud rate parameter is assumed to be stored in the r17:r16 Registers.

Assembly Code Example

```
USART_Init:
    ; Set baud rate
    out UBRRHn, r17
    out UBRLn, r16
    ; Enable receiver and transmitter
    ldi r16, (1<<RXEn) | (1<<TXEn)
    out USRnB, r16
    ; Set frame format: 8data, 2stop bit
    ldi r16, (1<<USBsn) | (3<<UCSzn0)
    out USRnC, r16
    ret
```

C Code Example

```
void USART_Init( unsigned int baud )
{
    /* Set baud rate */
    UBRRHn = (unsigned char)(baud>>8);
    UBRLn = (unsigned char)baud;
    /* Enable receiver and transmitter */
    USRnB = (1<<RXEn) | (1<<TXEn);
    /* Set frame format: 8data, 2stop bit */
    USRnC = (1<<USBsn) | (3<<UCSzn0);
}
```

Note: 1. See “Code Examples” on page 8.

More advanced initialization routines can be made that include frame format as parameters, disable interrupts and so on. However, many applications use a fixed setting of the baud and control registers, and for these types of applications the initialization code can be placed directly in the main routine, or be combined with initialization code for other I/O modules.
18.5 Data Transmission – The USART Transmitter

The USART Transmitter is enabled by setting the Transmit Enable (TXEN) bit in the UCSRnB Register. When the Transmitter is enabled, the normal port operation of the TxDn pin is overridden by the USART and given the function as the Transmitter’s serial output. The baud rate, mode of operation and frame format must be set up once before doing any transmissions. If synchronous operation is used, the clock on the XCKn pin will be overridden and used as transmission clock.

18.5.1 Sending Frames with 5 to 8 Data Bit

A data transmission is initiated by loading the transmit buffer with the data to be transmitted. The CPU can load the transmit buffer by writing to the UDRn I/O location. The buffered data in the transmit buffer will be moved to the Shift Register when the Shift Register is ready to send a new frame. The Shift Register is loaded with new data if it is in idle state (no ongoing transmission) or immediately after the last stop bit of the previous frame is transmitted. When the Shift Register is loaded with new data, it will transfer one complete frame at the rate given by the Baud Register, U2Xn bit or by XCKn depending on mode of operation.

The following code examples show a simple USART transmit function based on polling of the Data Register Empty (UDREn) Flag. When using frames with less than eight bits, the most significant bits written to the UDRn are ignored. The USART has to be initialized before the function can be used. For the assembly code, the data to be sent is assumed to be stored in Register R16.

Assembly Code Example

```assembly
USART_Transmit:
    ; Wait for empty transmit buffer
    sbis UCSRA,UDREn
    rjmp USART_Transmit
    ; Put data (rl6) into buffer, sends the data
    out UDRn,rl6
    ret
```

C Code Example

```c
void USART_Transmit( unsigned char data )
{
    /* Wait for empty transmit buffer */
    while ( !( UCSRA & (1<<UDREn)) )
    ;
    /* Put data into buffer, sends the data */
    UDRn = data;
}
```

Note: 1. See “Code Examples” on page 8.

The function simply waits for the transmit buffer to be empty by checking the UDREn Flag, before loading it with new data to be transmitted. If the Data Register Empty interrupt is utilized, the interrupt routine writes the data into the buffer.
18.5.2 Sending Frames with 9 Data Bit

If 9-bit characters are used (UCSZn = 7), the ninth bit must be written to the TXB8 bit in UCSRnB before the low byte of the character is written to UDRn. The following code examples show a transmit function that handles 9-bit characters. For the assembly code, the data to be sent is assumed to be stored in registers R17:R16.

### Assembly Code Example

```assembly
ASSEMBLY_CODE

USART_Transmit:
    ; Wait for empty transmit buffer
    sbis UCSRnA,UDREn
    rjmp USART_Transmit
    ; Copy 9th bit from r17 to TXB8
    cbi UCSRnB,TXB8
    sbrc r17,0
    sbi UCSRnB,TXB8
    ; Put LSB data (r16) into buffer, sends the data
    out UDRn,r16
    ret
```

### C Code Example

```c
C_CODE

void USART_Transmit( unsigned int data )
{
    /* Wait for empty transmit buffer */
    while( !( UCSRnA & (1<<UDREn)) )
    {
        /* Copy 9th bit to TXB8 */
        UCSRnB &= ~(1<<TXB8);
        if ( data & 0x0100 )
            UCSRnB |= (1<<TXB8);
        /* Put data into buffer, sends the data */
        UDRn = data;
    }
}
```

Notes:

1. These transmit functions are written to be general functions. They can be optimized if the contents of the UCSRnB is static. For example, only the TXB8 bit of the UCSRnB Register is used after initialization.


The ninth bit can be used for indicating an address frame when using multi processor communication mode or for other protocol handling as for example synchronization.

18.5.3 Transmitter Flags and Interrupts

The USART Transmitter has two flags that indicate its state: USART Data Register Empty (UDREn) and Transmit Complete (TXCn). Both flags can be used for generating interrupts.

The Data Register Empty (UDREn) Flag indicates whether the transmit buffer is ready to receive new data. This bit is set when the transmit buffer is empty, and cleared when the transmit buffer contains data to be transmitted that has not yet been moved into the Shift Register. For compatibility with future devices, always write this bit to zero when writing the UCSRnA Register.
When the Data Register Empty Interrupt Enable (UDRIEn) bit in UCSRnB is written to one, the USART Data Register Empty Interrupt will be executed as long as UDREn is set (provided that global interrupts are enabled). UDREn is cleared by writing UDn. When interrupt-driven data transmission is used, the Data Register Empty interrupt routine must either write new data to UDn in order to clear UDREn or disable the Data Register Empty interrupt, otherwise a new interrupt will occur once the interrupt routine terminates.

The Transmit Complete (TXCn) Flag bit is set one when the entire frame in the Transmit Shift Register has been shifted out and there are no new data currently present in the transmit buffer. The TXCn Flag bit is automatically cleared when a transmit complete interrupt is executed, or it can be cleared by writing a one to its bit location. The TXCn Flag is useful in half-duplex communication interfaces (like the RS-485 standard), where a transmitting application must enter receive mode and free the communication bus immediately after completing the transmission.

When the Transmit Complete Interrupt Enable (TXCIEn) bit in UCSRnB is set, the USART Transmit Complete Interrupt will be executed when the TXCn Flag becomes set (provided that global interrupts are enabled). When the transmit complete interrupt is used, the interrupt handling routine does not have to clear the TXCn Flag, this is done automatically when the interrupt is executed.

18.5.4 Parity Generator

The Parity Generator calculates the parity bit for the serial frame data. When parity bit is enabled (UPMn1 = 1), the transmitter control logic inserts the parity bit between the last data bit and the first stop bit of the frame that is sent.

18.5.5 Disabling the Transmitter

The disabling of the Transmitter (setting the TXEN to zero) will not become effective until ongoing and pending transmissions are completed, i.e., when the Transmit Shift Register and Transmit Buffer Register do not contain data to be transmitted. When disabled, the Transmitter will no longer override the TxDn pin.

18.6 Data Reception – The USART Receiver

The USART Receiver is enabled by writing the Receive Enable (RXEn) bit in the UCSRnB Register to one. When the Receiver is enabled, the normal pin operation of the RxDn pin is overridden by the USART and given the function as the Receiver’s serial input. The baud rate, mode of operation and frame format must be set up once before any serial reception can be done. If synchronous operation is used, the clock on the XCKn pin will be used as transfer clock.

18.6.1 Receiving Frames with 5 to 8 Data Bits

The Receiver starts data reception when it detects a valid start bit. Each bit that follows the start bit will be sampled at the baud rate or XCKn clock, and shifted into the Receive Shift Register until the first stop bit of a frame is received. A second stop bit will be ignored by the Receiver. When the first stop bit is received, i.e., a complete serial frame is present in the Receive Shift Register, the contents of the Shift Register will be moved into the receive buffer. The receive buffer can then be read by reading the UDn I/O location.

The following code example shows a simple USART receive function based on polling of the Receive Complete (RXCn) Flag. When using frames with less than eight bits the most significant
bits of the data read from the UDRn will be masked to zero. The USART has to be initialized before the function can be used.

### Assembly Code Example

```assembly
UART_Receive:
  ; Wait for data to be received
  sbis UCSRN0, RXCn
  rjmp UART_Receive
  ; Get and return received data from buffer
  in r16, UDRn
  ret
```

### C Code Example

```c
unsigned char UART_Receive( void )
{
  /* Wait for data to be received */
  while ( !(UCSR0A & (1<<RXCn)) )
    ;
  /* Get and return received data from buffer */
  return UDRn;
}
```

Note: 1. See “Code Examples” on page 8.

The function simply waits for data to be present in the receive buffer by checking the RXCn Flag, before reading the buffer and returning the value.

### 18.6.2 Receiving Frames with 9 Data Bits

If 9-bit characters are used (UCSZn=7) the ninth bit must be read from the RXB8n bit in UCSRN before reading the low bits from the UDRn. This rule applies to the FEn, DORn and UPEn Status Flags as well. Read status from UCSRN0, then data from UDRn. Reading the UDRn I/O location will change the state of the receive buffer FIFO and consequently the TXB8n, FEn, DORn and UPEn bits, which all are stored in the FIFO, will change.

The following code example shows a simple USART receive function that handles both nine bit characters and the status bits.
Assembly Code Example

```assembly
UART_Receive:
    ; Wait for data to be received
    sbis UCSRNn, RXCn
    rjmp UART_Receive
    ; Get status and 9th bit, then data from buffer
    in r18, UCSRNn
    in r17, UCSRNn
    in r16, UDn
    ; If error, return -1
    andi r18, (1<<FEn) | (1<<DORn) | (1<<UEn)
    breq UART_ReceiveNoError
    ldi r17, HIGH(-1)
    ldi r16, LOW(-1)
UART_ReceiveNoError:
    ; Filter the 9th bit, then return
    lsr r17
    andi r17, 0x01
    ret
```

C Code Example

```c
#include <avr/io.h>

unsigned int UART_Receive( void )
{
    unsigned char status, resh, resl;
    /* Wait for data to be received */
    while ( !(UCSRnA & (1<<RXCn)) )
    {
        /* Get status and 9th bit, then data */
        /* from buffer */
        status = UCSRNn;
        resh = UCSRNn;
        resl = UDn;
        /* If error, return -1 */
        if ( status & (1<<FEn) | (1<<DORn) | (1<<UEn) )
            return -1;
        /* Filter the 9th bit, then return */
        resh = (resh >> 1) & 0x01;
        return ((resh << 8) | resl);
    }
}
```

Note: 1. See “Code Examples” on page 8.

The receive function example reads all the I/O Registers into the Register File before any computation is done. This gives an optimal receive buffer utilization since the buffer location read will be free to accept new data as early as possible.
18.6.3 Receive Compete Flag and Interrupt

The USART Receiver has one flag that indicates the Receiver state.

The Receive Complete (RXCn) Flag indicates if there are unread data present in the receive buffer. This flag is one when unread data exist in the receive buffer, and zero when the receive buffer is empty (i.e., does not contain any unread data). If the Receiver is disabled (RXEnn = 0), the receive buffer will be flushed and consequently the RXCn bit will become zero.

When the Receive Complete Interrupt Enable (RXCIn) in UCSRnB is set, the USART Receive Complete interrupt will be executed as long as the RXCn Flag is set (provided that global interrupts are enabled). When interrupt-driven data reception is used, the receive complete routine must read the received data from UDRn in order to clear the RXCn Flag, otherwise a new interrupt will occur once the interrupt routine terminates.

18.6.4 Receiver Error Flags

The USART Receiver has three Error Flags: Frame Error (FEn), Data OverRun (DORn) and Parity Error (UPEn). All can be accessed by reading UCSRnA. Common for the Error Flags is that they are located in the receive buffer together with the frame for which they indicate the error status. Due to the buffering of the Error Flags, the UCSRnA must be read before the receive buffer (UDRn), since reading the UDRn I/O location changes the buffer read location. Another equality for the Error Flags is that they can not be altered by software doing a write to the flag location. However, all flags must be set to zero when the UCSRnA is written for upward compatibility of future USART implementations. None of the Error Flags can generate interrupts.

The Frame Error (FEn) Flag indicates the state of the first stop bit of the next readable frame stored in the receive buffer. The FEn Flag is zero when the stop bit was correctly read (as one), and the FEn Flag will be one when the stop bit was incorrect (zero). This flag can be used for detecting out-of-sync conditions, detecting break conditions and protocol handling. The FEn Flag is not affected by the setting of the USBSn bit in UCSRnC since the Receiver ignores all, except for the first, stop bits. For compatibility with future devices, always set this bit to zero when writing to UCSRnA.

The Data OverRun (DORn) Flag indicates data loss due to a receiver buffer full condition. A Data OverRun occurs when the receive buffer is full (two characters), it is a new character waiting in the Receive Shift Register, and a new start bit is detected. If the DORn Flag is set there was one or more serial frame lost between the frame last read from UDRn, and the next frame read from UDRn. For compatibility with future devices, always write this bit to zero when writing to UCSRnA. The DORn Flag is cleared when the frame received was successfully moved from the Shift Register to the receive buffer.

The Parity Error (UPEn) Flag indicates that the next frame in the receive buffer had a Parity Error when received. If Parity Check is not enabled the UPEn bit will always be read zero. For compatibility with future devices, always set this bit to zero when writing to UCSRnA. For more details see "Parity Bit Calculation" on page 191 and "Parity Checker" on page 198.

18.6.5 Parity Checker

The Parity Checker is active when the high USART Parity mode (UPMn1) bit is set. Type of Parity Check to be performed (odd or even) is selected by the UPMn0 bit. When enabled, the Parity Checker calculates the parity of the data bits in incoming frames and compares the result with the parity bit from the serial frame. The result of the check is stored in the receive buffer together with the received data and stop bits. The Parity Error (UPEn) Flag can then be read by software to check if the frame had a Parity Error.
The UPEn bit is set if the next character that can be read from the receive buffer had a Parity Error when received and the Parity Checking was enabled at that point (UPMn1 = 1). This bit is valid until the receive buffer (UDRn) is read.

18.6.6 Disabling the Receiver
In contrast to the Transmitter, disabling of the Receiver will be immediate. Data from ongoing receptions will therefore be lost. When disabled (i.e., the RXENn is set to zero) the Receiver will no longer override the normal function of the RxDn port pin. The Receiver buffer FIFO will be flushed when the Receiver is disabled. Remaining data in the buffer will be lost.

18.6.7 Flushing the Receive Buffer
The receiver buffer FIFO will be flushed when the Receiver is disabled, i.e., the buffer will be emptied of its contents. Unread data will be lost. If the buffer has to be flushed during normal operation, due to for instance an error condition, read the UDRn I/O location until the RXCn Flag is cleared. The following code example shows how to flush the receive buffer.

<table>
<thead>
<tr>
<th>Assembly Code Example(1)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>USART_Flush:</strong></td>
</tr>
<tr>
<td><strong>sbis</strong> UCSRnA, RXCn</td>
</tr>
<tr>
<td><strong>ret</strong></td>
</tr>
<tr>
<td><strong>in</strong> r16, UDRn</td>
</tr>
<tr>
<td><strong>rjmp</strong> USART_Flush</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>C Code Example(1)</th>
</tr>
</thead>
<tbody>
<tr>
<td>void USART_Flush( void )</td>
</tr>
<tr>
<td>{</td>
</tr>
<tr>
<td>unsigned char dummy;</td>
</tr>
<tr>
<td>while ( UCSRnA &amp; (1&lt;&lt;RXCn) ) dummy = UDRn;</td>
</tr>
<tr>
<td>}</td>
</tr>
</tbody>
</table>

Note: 1. See “Code Examples” on page 8.

18.7 Asynchronous Data Reception
The USART includes a clock recovery and a data recovery unit for handling asynchronous data reception. The clock recovery logic is used for synchronizing the internally generated baud rate clock to the incoming asynchronous serial frames at the RxDn pin. The data recovery logic samples and low pass filters each incoming bit, thereby improving the noise immunity of the Receiver. The asynchronous reception operational range depends on the accuracy of the internal baud rate clock, the rate of the incoming frames, and the frame size in number of bits.

18.7.1 Asynchronous Clock Recovery
The clock recovery logic synchronizes internal clock to the incoming serial frames. Figure 18-5 illustrates the sampling process of the start bit of an incoming frame. The sample rate is 16 times the baud rate for Normal mode, and eight times the baud rate for Double Speed mode. The horizontal arrows illustrate the synchronization variation due to the sampling process. Note the larger time variation when using the Double Speed mode (U2Xn = 1) of operation. Samples denoted zero are samples done when the RxDn line is idle (i.e., no communication activity).
When the clock recovery logic detects a high (idle) to low (start) transition on the RxDn line, the start bit detection sequence is initiated. Let sample 1 denote the first zero-sample as shown in the figure. The clock recovery logic then uses samples 8, 9, and 10 for Normal mode, and samples 4, 5, and 6 for Double Speed mode (indicated with sample numbers inside boxes on the figure), to decide if a valid start bit is received. If two or more of these three samples have logical high levels (the majority wins), the start bit is rejected as a noise spike and the Receiver starts looking for the next high to low-transition. If however, a valid start bit is detected, the clock recovery logic is synchronized and the data recovery can begin. The synchronization process is repeated for each start bit.

18.7.2 Asynchronous Data Recovery

When the receiver clock is synchronized to the start bit, the data recovery can begin. The data recovery unit uses a state machine that has 16 states for each bit in Normal mode and eight states for each bit in Double Speed mode. Figure 18-6 shows the sampling of the data bits and the parity bit. Each of the samples is given a number that is equal to the state of the recovery unit.

The decision of the logic level of the received bit is taken by doing a majority voting of the logic value to the three samples in the center of the received bit. The center samples are emphasized on the figure by having the sample number inside boxes. The majority voting process is done as follows: If two or all three samples have high levels, the received bit is registered to be a logic 1. If two or all three samples have low levels, the received bit is registered to be a logic 0. This majority voting process acts as a low pass filter for the incoming signal on the RxDn pin. The recovery process is then repeated until a complete frame is received. Including the first stop bit. Note that the Receiver only uses the first stop bit of a frame.

Figure 18-7 shows the sampling of the stop bit and the earliest possible beginning of the start bit of the next frame.
The same majority voting is done to the stop bit as done for the other bits in the frame. If the stop bit is registered to have a logic 0 value, the Frame Error (FE) Flag will be set.

A new high to low transition indicating the start bit of a new frame can come right after the last of the bits used for majority voting. For Normal Speed mode, the first low level sample can be at point marked (A) in Figure 18-7. For Double Speed mode the first low level must be delayed to (B). (C) marks a stop bit of full length. The early start bit detection influences the operational range of the Receiver.

18.7.3  Asynchronous Operational Range

The operational range of the Receiver is dependent on the mismatch between the received bit rate and the internally generated baud rate. If the Transmitter is sending frames at too fast or too slow bit rates, or the internally generated baud rate of the Receiver does not have a similar (see Table 18-2) base frequency, the Receiver will not be able to synchronize the frames to the start bit.

The following equations can be used to calculate the ratio of the incoming data rate and internal receiver baud rate.

\[
R_{\text{slow}} = \frac{(D + 1)S}{S - 1 + D \cdot S + S_F} \\
R_{\text{fast}} = \frac{(D + 2)S}{(D + 1)S + S_M}
\]

D  Sum of character size and parity size (D = 5 to 10 bit)
S  Samples per bit. S = 16 for Normal Speed mode and S = 8 for Double Speed mode.
S_F First sample number used for majority voting. S_F = 8 for normal speed and S_F = 4 for Double Speed mode.
S_M Middle sample number used for majority voting. S_M = 9 for normal speed and S_M = 5 for Double Speed mode.

\(R_{\text{slow}}\) is the ratio of the slowest incoming data rate that can be accepted in relation to the receiver baud rate. \(R_{\text{fast}}\) is the ratio of the fastest incoming data rate that can be accepted in relation to the receiver baud rate.

Table 18-2 and Table 18-3 list the maximum receiver baud rate error that can be tolerated. Note that Normal Speed mode has higher toleration of baud rate variations.
Table 18-2. Recommended Maximum Receiver Baud Rate Error for Normal Speed Mode (U2Xn = 0)

<table>
<thead>
<tr>
<th>D # (Data+Parity Bit)</th>
<th>R_{slow} (%)</th>
<th>R_{fast} (%)</th>
<th>Max Total Error (%)</th>
<th>Recommended Max Receiver Error (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>93.20</td>
<td>106.67</td>
<td>+6.67/-6.8</td>
<td>± 3.0</td>
</tr>
<tr>
<td>6</td>
<td>94.12</td>
<td>105.79</td>
<td>+5.79/-5.88</td>
<td>± 2.5</td>
</tr>
<tr>
<td>7</td>
<td>94.81</td>
<td>105.11</td>
<td>+5.11/-5.19</td>
<td>± 2.0</td>
</tr>
<tr>
<td>8</td>
<td>95.36</td>
<td>104.58</td>
<td>+4.58/-4.54</td>
<td>± 2.0</td>
</tr>
<tr>
<td>9</td>
<td>95.81</td>
<td>104.14</td>
<td>+4.14/-4.19</td>
<td>± 1.5</td>
</tr>
<tr>
<td>10</td>
<td>96.17</td>
<td>103.78</td>
<td>+3.78/-3.83</td>
<td>± 1.5</td>
</tr>
</tbody>
</table>

Table 18-3. Recommended Maximum Receiver Baud Rate Error for Double Speed Mode (U2Xn = 1)

<table>
<thead>
<tr>
<th>D # (Data+Parity Bit)</th>
<th>R_{slow} (%)</th>
<th>R_{fast} (%)</th>
<th>Max Total Error (%)</th>
<th>Recommended Max Receiver Error (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>94.12</td>
<td>105.66</td>
<td>+5.66/-5.88</td>
<td>± 2.5</td>
</tr>
<tr>
<td>6</td>
<td>94.92</td>
<td>104.92</td>
<td>+4.92/-5.08</td>
<td>± 2.0</td>
</tr>
<tr>
<td>7</td>
<td>95.52</td>
<td>104.35</td>
<td>+4.35/-4.48</td>
<td>± 1.5</td>
</tr>
<tr>
<td>8</td>
<td>96.00</td>
<td>103.90</td>
<td>+3.90/-4.00</td>
<td>± 1.5</td>
</tr>
<tr>
<td>9</td>
<td>96.39</td>
<td>103.53</td>
<td>+3.53/-3.61</td>
<td>± 1.5</td>
</tr>
<tr>
<td>10</td>
<td>96.70</td>
<td>103.23</td>
<td>+3.23/-3.30</td>
<td>± 1.0</td>
</tr>
</tbody>
</table>

The recommendations of the maximum receiver baud rate error was made under the assumption that the Receiver and Transmitter equally divides the maximum total error.

There are two possible sources for the receivers baud rate error. The Receiver’s system clock (XTAL) will always have some minor instability over the supply voltage range and the temperature range. When using a crystal to generate the system clock, this is rarely a problem, but for a resonator the system clock may differ more than 2% depending of the resonators tolerance. The second source for the error is more controllable. The baud rate generator can not always do an exact division of the system frequency to get the baud rate wanted. In this case an UBRR value that gives an acceptable low error can be used if possible.

18.8 Multi-processor Communication Mode

Setting the Multi-processor Communication mode (MPCMn) bit in UCSRNnA enables a filtering function of incoming frames received by the USART Receiver. Frames that do not contain address information will be ignored and not put into the receive buffer. This effectively reduces the number of incoming frames that has to be handled by the CPU, in a system with multiple MCUs that communicate via the same serial bus. The Transmitter is unaffected by the MPCMn setting, but has to be used differently when it is a part of a system utilizing the Multi-processor Communication mode.

If the Receiver is set up to receive frames that contain 5 to 8 data bits, then the first stop bit indicates if the frame contains data or address information. If the Receiver is set up for frames with
nine data bits, then the ninth bit (RXB8n) is used for identifying address and data frames. When the frame type bit (the first stop or the ninth bit) is one, the frame contains an address. When the frame type bit is zero the frame is a data frame.

The Multi-processor Communication mode enables several slave MCUs to receive data from a master MCU. This is done by first decoding an address frame to find out which MCU has been addressed. If a particular slave MCU has been addressed, it will receive the following data frames as normal, while the other slave MCUs will ignore the received frames until another address frame is received.

18.8.1 Using MPC Mn

For an MCU to act as a master MCU, it can use a 9-bit character frame format (UCSZn = 7). The ninth bit (TXB8n) must be set when an address frame (TXB8n = 1) or cleared when a data frame (TXB = 0) is being transmitted. The slave MCUs must in this case be set to use a 9-bit character frame format.

The following procedure should be used to exchange data in Multi-processor Communication mode:

1. All Slave MCUs are in Multi-processor Communication mode (MPC Mn in UCSRnA is set).
2. The Master MCU sends an address frame, and all slaves receive and read this frame. In the Slave MCUs, the RXCn Flag in UCSRnA will be set as normal.
3. Each Slave MCU reads the UDRn Register and determines if it has been selected. If so, it clears the MPC Mn bit in UCSRnA, otherwise it waits for the next address byte and keeps the MPC Mn setting.
4. The addressed MCU will receive all data frames until a new address frame is received. The other Slave MCUs, which still have the MPC Mn bit set, will ignore the data frames.
5. When the last data frame is received by the addressed MCU, the addressed MCU sets the MPC Mn bit and waits for a new address frame from master. The process then repeats from 2.

Using any of the 5- to 8-bit character frame formats is possible, but impractical since the Receiver must change between using n and n+1 character frame formats. This makes full-duplex operation difficult since the Transmitter and Receiver uses the same character size setting. If 5- to 8-bit character frames are used, the Transmitter must be set to use two stop bit (USBSn = 1) since the first stop bit is used for indicating the frame type.

Do not use Read-Modify-Write instructions (SBI and CBI) to set or clear the MPC Mn bit. The MPC Mn bit shares the same I/O location as the TXCn Flag and this might accidentally be cleared when using SBI or CBI instructions.

18.9 Hardware Flow Control

The hardware flow control can be enabled by software.

CTS: (Clear to Send)
RTS: (Request to Send)
18.9.1 Receiver Flow Control

The reception flow can be controlled by hardware using the RTS pin. The aim of the flow control is to inform the external transmitter when the internal receive Fifo is full. Thus the transmitter can stop sending characters. RTS usage and so associated flow control is enabled using RTSEN bit in UCSRnD.

Figure 18-8. Reception Flow Control Waveform Example

<table>
<thead>
<tr>
<th>FIFO Index</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>CPU Read</th>
</tr>
</thead>
<tbody>
<tr>
<td>RXD</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RTS</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Figure 18-9. RTS behavior**

 RTS will rise at 2/3 of the last received stop bit if the receive fifo is full.

To ensure reliable transmissions, even after a RTS rise, an extra-data can still be received and stored in the Receive Shift Register.

18.9.2 Transmission Flow Control

The transmission flow can be controlled by hardware using the CTS pin controlled by the external receiver. The aim of the flow control is to stop transmission when the receiver is full of data (CTS = 1). CTS usage and so associated flow control is enabled using CTSEN bit in UCSRnD.

The CTS pin is sampled at each CPU write and at the middle of the last stop bit that is currently being sent.

**Figure 18-10. CTS behavior**

Write from CPU

<table>
<thead>
<tr>
<th>TXD</th>
<th>Start</th>
<th>Byte0</th>
<th>Stop</th>
<th>Start</th>
<th>Byte1</th>
<th>Stop</th>
<th>Start</th>
<th>Byte2</th>
</tr>
</thead>
<tbody>
<tr>
<td>CTS</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
18.10 USART Register Description

18.10.1 USART I/O Data Register n– UDRn

<table>
<thead>
<tr>
<th>Bit</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read/Write</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
</tr>
<tr>
<td>Initial Value</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

The USART Transmit Data Buffer Register and USART Receive Data Buffer Registers share the same I/O address referred to as USART Data Register or UDRn. The Transmit Data Buffer Register (TXB) will be the destination for data written to the UDRn Register location. Reading the UDRn Register location will return the contents of the Receive Data Buffer Register (RXB).

For 5-, 6-, or 7-bit characters the upper unused bits will be ignored by the Transmitter and set to zero by the Receiver.

The transmit buffer can only be written when the UDREN Flag in the UCSRnA Register is set. Data written to UDRn when the UDREN Flag is not set, will be ignored by the USART Transmitter. When data is written to the transmit buffer, and the Transmitter is enabled, the Transmitter will load the data into the Transmit Shift Register when the Shift Register is empty. Then the data will be serially transmitted on the TXDn pin.

The receive buffer consists of a two level FIFO. The FIFO will change its state whenever the receive buffer is accessed. Due to this behavior of the receive buffer, do not use Read-Modify-Write instructions (SBI and CBI) on this location. Be careful when using bit test instructions (SBIC and SBIS), since these also will change the state of the FIFO.

18.10.2 USART Control and Status Register A – UCSRnA

<table>
<thead>
<tr>
<th>Bit</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read/Write</td>
<td>RXCn</td>
<td>TXCn</td>
<td>UDREN</td>
<td>FEn</td>
<td>DORn</td>
<td>UPEn</td>
<td>U2Xn</td>
<td>MPCn</td>
</tr>
<tr>
<td>Initial Value</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

- **Bit 7 – RXCn: USART Receive Complete**
  This flag bit is set when there are unread data in the receive buffer and cleared when the receive buffer is empty (i.e., does not contain any unread data). If the Receiver is disabled, the receive buffer will be flushed and consequently the RXCn bit will become zero. The RXCn Flag can be used to generate a Receive Complete interrupt (see description of the RXCIEn bit).

- **Bit 6 – TXCn: USART Transmit Complete**
  This flag bit is set when the entire frame in the Transmit Shift Register has been shifted out and there are no new data currently present in the transmit buffer (UDRn). The TXCn Flag bit is automatically cleared when a transmit complete interrupt is executed, or it can be cleared by writing a one to its bit location. The TXCn Flag can generate a Transmit Complete interrupt (see description of the TXCIEn bit).

- **Bit 5 – UDREN: USART Data Register Empty**
  The UDREN Flag indicates if the transmit buffer (UDRn) is ready to receive new data. If UDREN is one, the buffer is empty, and therefore ready to be written. The UDREN Flag can generate a Data Register Empty interrupt (see description of the UDRIEn bit). UDREN is set after a reset to indicate that the Transmitter is ready.
• Bit 4 – FEn: Frame Error
This bit is set if the next character in the receive buffer had a Frame Error when received. I.e., when the first stop bit of the next character in the receive buffer is zero. This bit is valid until the receive buffer (UDRn) is read. The FEn bit is zero when the stop bit of received data is one. Always set this bit to zero when writing to UCSRnA.

• Bit 3 – DORn: Data OverRun
This bit is set if a Data OverRun condition is detected. A Data OverRun occurs when the receive buffer is full (two characters), it is a new character waiting in the Receive Shift Register, and a new start bit is detected. This bit is valid until the receive buffer (UDRn) is read. Always set this bit to zero when writing to UCSRnA.

• Bit 2 – UPEn: USART Parity Error
This bit is set if the next character in the receive buffer had a Parity Error when received and the Parity Checking was enabled at that point (UPMn1 = 1). This bit is valid until the receive buffer (UDRn) is read. Always set this bit to zero when writing to UCSRnA.

• Bit 1 – U2Xn: Double the USART Transmission Speed
This bit only has effect for the asynchronous operation. Write this bit to zero when using synchronous operation.
Writing this bit to one will reduce the divisor of the baud rate divider from 16 to 8 effectively doubling the transfer rate for asynchronous communication.

• Bit 0 – MPCMn: Multi-processor Communication Mode
This bit enables the Multi-processor Communication mode. When the MPCMn bit is written to one, all the incoming frames received by the USART Receiver that do not contain address information will be ignored. The Transmitter is unaffected by the MPCMn setting. For more detailed information see “Multi-processor Communication Mode” on page 202.

18.10.3 USART Control and Status Register n B – UCSRnB

<table>
<thead>
<tr>
<th>Bit</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
<th>UCSRnB</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>RXCIn</td>
<td>TXCIn</td>
<td>UDRIn</td>
<td>RXEn</td>
<td>TXEn</td>
<td>UCSZn2</td>
<td>RXB8n</td>
<td>TXB8n</td>
<td></td>
</tr>
<tr>
<td>Read/Write</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R</td>
<td>R/W</td>
<td></td>
</tr>
<tr>
<td>Initial Value</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

• Bit 7 – RXCIn: RX Complete Interrupt Enable n
Writing this bit to one enables interrupt on the RXCn Flag. A USART Receive Complete interrupt will be generated only if the RXCIn bit is written to one, the Global Interrupt Flag in SREG is written to one and the RXCn bit in UCSRnA is set.

• Bit 6 – TXCIn: TX Complete Interrupt Enable n
Writing this bit to one enables interrupt on the TXCn Flag. A USART Transmit Complete interrupt will be generated only if the TXCIn bit is written to one, the Global Interrupt Flag in SREG is written to one and the TXCn bit in UCSRnA is set.

• Bit 5 – UDRIEn: USART Data Register Empty Interrupt Enable n
Writing this bit to one enables interrupt on the UDREn Flag. A Data Register Empty interrupt will be generated only if the UDRIEn bit is written to one, the Global Interrupt Flag in SREG is written to one and the UDREn bit in UCSRnA is set.
• **Bit 4 – RXENn: Receiver Enable n**
  Writing this bit to one enables the USART Receiver. The Receiver will override normal port operation for the RxDn pin when enabled. Disabling the Receiver will flush the receive buffer invalidating the FEn, DORn, and UPEn Flags.

• **Bit 3 – TXENn: Transmitter Enable n**
  Writing this bit to one enables the USART Transmitter. The Transmitter will override normal port operation for the TxDn pin when enabled. The disabling of the Transmitter (writing TXENn to zero) will not become effective until ongoing and pending transmissions are completed, i.e., when the Transmit Shift Register and Transmit Buffer Register do not contain data to be transmitted. When disabled, the Transmitter will no longer override the TxDn port.

• **Bit 2 – UCSZn2: Character Size n**
  The UCSZn2 bits combined with the UCSZn1:0 bit in UCSRNc sets the number of data bits (Character SiZe) in a frame the Receiver and Transmitter use.

• **Bit 1 – RXB8n: Receive Data Bit 8 n**
  RXB8n is the ninth data bit of the received character when operating with serial frames with nine data bits. Must be read before reading the low bits from UDRn.

• **Bit 0 – TXB8n: Transmit Data Bit 8 n**
  TXB8n is the ninth data bit in the character to be transmitted when operating with serial frames with nine data bits. Must be written before writing the low bits to UDRn.

### 18.10.4 USART Control and Status Register n C – UCSRNc

<table>
<thead>
<tr>
<th>Bit</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
<th>UCSTRnC</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>UMSELn1</td>
<td>UMSELn0</td>
<td>UPMn1</td>
<td>UPMn0</td>
<td>USBn</td>
<td>UCSZn1</td>
<td>UCSZn0</td>
<td>UCPOLn</td>
<td></td>
</tr>
<tr>
<td>Read/Write</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td></td>
</tr>
<tr>
<td>Initial Value</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

• **Bits 7:6 – UMSELn1:0 USART Mode Select**
  These bits select the mode of operation of the USARTn as shown in Table 18-4.

<table>
<thead>
<tr>
<th>UMSELn1</th>
<th>UMSELn0</th>
<th>Mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>Asynchronous USART</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>Synchronous USART</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>(Reserved)</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>Master SPI (MSPIM)(^{(1)})</td>
</tr>
</tbody>
</table>

Note: 1. See "USART in SPI Mode" on page 214 for full description of the Master SPI Mode (MSPIM) operation

• **Bits 5:4 – UPMn1:0: Parity Mode**
  These bits enable and set type of parity generation and check. If enabled, the Transmitter will automatically generate and send the parity of the transmitted data bits within each frame. The
Receiver will generate a parity value for the incoming data and compare it to the UPMn setting. If a mismatch is detected, the UPEn Flag in UCSRnA will be set.

Table 18-5.  UPMn Bit Settings

<table>
<thead>
<tr>
<th>UPMn1</th>
<th>UPMn0</th>
<th>Parity Mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>Disabled</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>Reserved</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>Enabled, Even Parity</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>Enabled, Odd Parity</td>
</tr>
</tbody>
</table>

• Bit 3 – USBSn: Stop Bit Select
This bit selects the number of stop bits to be inserted by the Transmitter. The Receiver ignores this setting.

Table 18-6.  USBS Bit Settings

<table>
<thead>
<tr>
<th>USBSn</th>
<th>Stop Bit(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1-bit</td>
</tr>
<tr>
<td>1</td>
<td>2-bit</td>
</tr>
</tbody>
</table>

• Bit 2:1 – UCSZn1:0: Character Size
The UCSZn1:0 bits combined with the UCSZn2 bit in UCSRnB sets the number of data bits (Character Size) in a frame the Receiver and Transmitter use.

Table 18-7.  UCSZn Bit Settings

<table>
<thead>
<tr>
<th>UCSZn2</th>
<th>UCSZn1</th>
<th>UCSZn0</th>
<th>Character Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>5-bit</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>6-bit</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>7-bit</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>8-bit</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>Reserved</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>Reserved</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>Reserved</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>9-bit</td>
</tr>
</tbody>
</table>

• Bit 0 – UCPOLn: Clock Polarity
This bit is used for synchronous mode only. Write this bit to zero when asynchronous mode is used. The UCPOLn bit sets the relationship between data output change and data input sample, and the synchronous clock (XCKn).

Table 18-8.  UCPOLn Bit Settings

<table>
<thead>
<tr>
<th>UCPOLn</th>
<th>Transmitted Data Changed (Output of TxDn Pin)</th>
<th>Received Data Sampled (Input on RxDn Pin)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Rising XCKn Edge</td>
<td>Falling XCKn Edge</td>
</tr>
<tr>
<td>1</td>
<td>Falling XCKn Edge</td>
<td>Rising XCKn Edge</td>
</tr>
</tbody>
</table>
18.10.5 USART Control and Status Register n D– UCSRnD

<table>
<thead>
<tr>
<th>Bit</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
<th>UCSRnD</th>
</tr>
</thead>
<tbody>
<tr>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td></td>
</tr>
</tbody>
</table>

Initial Value: 0 0 0 0 0 0 0 0

- **Bits 7:2 – Reserved bits**
  These bits are reserved and will be read as ‘0’. Do not set these bits.

- **Bits 1 – CTSEN: UART CTS Signal Enable**
  Set this bit by firmware to enable the transmission flow control signal (CTS). Transmission will be enabled only if CTS input = 0. Clear this bit to disable the transmission flow control signal. Transmission will occur without hardware condition. Data Direction Register bit must be correctly clear to enable the pin as an input.

- **Bits 0 – RTSEN: UART RTS Signal Enable**
  Set this bit by firmware to enable the reception flow control signal (RTS). In this case the RTS line will automatically rise when the FIFO is full. Clear this bit to disable the reception flow control signal. Data Direction Register bit must be correctly set to enable the pin as an output.

18.10.6 USART Baud Rate Registers – UBRRLn and UBRRHn

<table>
<thead>
<tr>
<th>Bit</th>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>UBRRHn</th>
<th>UBRRLn</th>
</tr>
</thead>
<tbody>
<tr>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Initial Value: 0 0 0 0 0 0 0 0

- **Bit 15:12 – Reserved Bits**
  These bits are reserved for future use. For compatibility with future devices, these bit must be written to zero when UBRRH is written.

- **Bit 11:0 – UBRR11:0: USART Baud Rate Register**
  This is a 12-bit register which contains the USART baud rate. The UBRRH contains the four most significant bits, and the UBRRRL contains the eight least significant bits of the USART baud rate. Ongoing transmissions by the Transmitter and Receiver will be corrupted if the baud rate is changed. Writing UBRRRL will trigger an immediate update of the baud rate prescaler.

18.11 Examples of Baud Rate Setting

For standard crystal and resonator frequencies, the most commonly used baud rates for asynchronous operation can be generated by using the UBRR settings in Table 18-9 to Table 18-12. UBRR values which yield an actual baud rate differing less than 0.5% from the target baud rate, are bold in the table. Higher error ratings are acceptable, but the Receiver will have less noise
resistance when the error ratings are high, especially for large serial frames (see “Asynchronous Operational Range” on page 201). The error values are calculated using the following equation:

$$\text{Error[\%]} = \left( \frac{\text{BaudRate}_{\text{Closet Match}}}{\text{BaudRate}} - 1 \right) \times 100\%$$

### Table 18-9. Examples of UBRRn Settings for Commonly Used Oscillator Frequencies

<table>
<thead>
<tr>
<th>Baud Rate (bps)</th>
<th>$f_{osc} = 1.0000$ MHz</th>
<th>$f_{osc} = 1.8432$ MHz</th>
<th>$f_{osc} = 2.0000$ MHz</th>
</tr>
</thead>
<tbody>
<tr>
<td>U2Xn = 0</td>
<td>U2Xn = 1</td>
<td>U2Xn = 0</td>
<td>U2Xn = 1</td>
</tr>
<tr>
<td>UBRR</td>
<td>Error</td>
<td>UBRR</td>
<td>Error</td>
</tr>
<tr>
<td>2400</td>
<td>25</td>
<td>0.2%</td>
<td>51</td>
</tr>
<tr>
<td>4800</td>
<td>12</td>
<td>0.2%</td>
<td>25</td>
</tr>
<tr>
<td>9600</td>
<td>6</td>
<td>-7.0%</td>
<td>12</td>
</tr>
<tr>
<td>14.4k</td>
<td>3</td>
<td>8.5%</td>
<td>8</td>
</tr>
<tr>
<td>19.2k</td>
<td>2</td>
<td>8.5%</td>
<td>6</td>
</tr>
<tr>
<td>28.8k</td>
<td>1</td>
<td>8.5%</td>
<td>3</td>
</tr>
<tr>
<td>38.4k</td>
<td>1</td>
<td>-18.6%</td>
<td>2</td>
</tr>
<tr>
<td>57.6k</td>
<td>0</td>
<td>8.5%</td>
<td>1</td>
</tr>
<tr>
<td>76.8k</td>
<td>-</td>
<td>-</td>
<td>1</td>
</tr>
<tr>
<td>115.2k</td>
<td>-</td>
<td>-</td>
<td>0</td>
</tr>
<tr>
<td>230.4k</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>250k</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Max. (1)</td>
<td>62.5 kbps</td>
<td>125 kbps</td>
<td>115.2 kbps</td>
</tr>
</tbody>
</table>

1. UBRR = 0, Error = 0.0%
Table 18-10. Examples of UBRRn Settings for Commonly Used Oscillator Frequencies (Continued)

<table>
<thead>
<tr>
<th>Baud Rate (bps)</th>
<th>$f_{\text{osc}} = 3.6864$ MHz</th>
<th>$f_{\text{osc}} = 4.0000$ MHz</th>
<th>$f_{\text{osc}} = 7.3728$ MHz</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>U2Xn = 0</td>
<td>U2Xn = 1</td>
<td>U2Xn = 0</td>
</tr>
<tr>
<td></td>
<td>UBRR</td>
<td>Error</td>
<td>UBRR</td>
</tr>
<tr>
<td>2400</td>
<td>95</td>
<td>0.0%</td>
<td>191</td>
</tr>
<tr>
<td>4800</td>
<td>47</td>
<td>0.0%</td>
<td>95</td>
</tr>
<tr>
<td>9600</td>
<td>23</td>
<td>0.0%</td>
<td>47</td>
</tr>
<tr>
<td>14.4k</td>
<td>15</td>
<td>0.0%</td>
<td>31</td>
</tr>
<tr>
<td>19.2k</td>
<td>11</td>
<td>0.0%</td>
<td>23</td>
</tr>
<tr>
<td>28.8k</td>
<td>7</td>
<td>0.0%</td>
<td>15</td>
</tr>
<tr>
<td>38.4k</td>
<td>5</td>
<td>0.0%</td>
<td>11</td>
</tr>
<tr>
<td>57.6k</td>
<td>3</td>
<td>0.0%</td>
<td>7</td>
</tr>
<tr>
<td>76.8k</td>
<td>2</td>
<td>0.0%</td>
<td>5</td>
</tr>
<tr>
<td>115.2k</td>
<td>1</td>
<td>0.0%</td>
<td>3</td>
</tr>
<tr>
<td>230.4k</td>
<td>0</td>
<td>0.0%</td>
<td>1</td>
</tr>
<tr>
<td>250k</td>
<td>0</td>
<td>-7.8%</td>
<td>1</td>
</tr>
<tr>
<td>0.5M</td>
<td>–</td>
<td>–</td>
<td>0</td>
</tr>
<tr>
<td>1M</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
</tbody>
</table>

Max. (1) 230.4 kbps 460.8 kbps 250 kbps 0.5 Mbps 460.8 kbps 921.6 kbps

1. UBRR = 0, Error = 0.0%
Table 18-11. Examples of UBRRn Settings for Commonly Used Oscillator Frequencies (Continued)

<table>
<thead>
<tr>
<th>Baud Rate (bps)</th>
<th>$f_{osc} = 8.0000$ MHz</th>
<th>$f_{osc} = 11.0592$ MHz</th>
<th>$f_{osc} = 14.7456$ MHz</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>U2Xn = 0</td>
<td>U2Xn = 1</td>
<td>U2Xn = 0</td>
</tr>
<tr>
<td>UBRR</td>
<td>Error</td>
<td>UBRR</td>
<td>Error</td>
</tr>
<tr>
<td>2400</td>
<td>207 0.2%</td>
<td>416 -0.1%</td>
<td>287 0.0%</td>
</tr>
<tr>
<td>4800</td>
<td>103 0.2%</td>
<td>207 0.2%</td>
<td>143 0.0%</td>
</tr>
<tr>
<td>9600</td>
<td>51 0.2%</td>
<td>103 0.2%</td>
<td>71 0.0%</td>
</tr>
<tr>
<td>14.4k</td>
<td>34 -0.8%</td>
<td>68 0.6%</td>
<td>47 0.0%</td>
</tr>
<tr>
<td>19.2k</td>
<td>25 0.2%</td>
<td>51 0.2%</td>
<td>35 0.0%</td>
</tr>
<tr>
<td>28.8k</td>
<td>16 2.1%</td>
<td>34 -0.8%</td>
<td>23 0.0%</td>
</tr>
<tr>
<td>38.4k</td>
<td>12 0.2%</td>
<td>25 0.2%</td>
<td>17 0.0%</td>
</tr>
<tr>
<td>57.6k</td>
<td>8 -3.5%</td>
<td>16 2.1%</td>
<td>11 0.0%</td>
</tr>
<tr>
<td>76.8k</td>
<td>6 -7.0%</td>
<td>12 0.2%</td>
<td>8 0.0%</td>
</tr>
<tr>
<td>115.2k</td>
<td>3 8.5%</td>
<td>8 -3.5%</td>
<td>5 0.0%</td>
</tr>
<tr>
<td>230.4k</td>
<td>1 8.5%</td>
<td>3 8.5%</td>
<td>2 0.0%</td>
</tr>
<tr>
<td>250k</td>
<td>1 0.0%</td>
<td>3 0.0%</td>
<td>2 -7.8%</td>
</tr>
<tr>
<td>0.5M</td>
<td>0 0.0%</td>
<td>1 0.0%</td>
<td>2 -7.8%</td>
</tr>
<tr>
<td>1M</td>
<td>– – 0.0%</td>
<td>– – 0.0%</td>
<td>– – 0.0%</td>
</tr>
</tbody>
</table>

1. UBRR = 0, Error = 0.0%

Max. (1) 0.5 Mbps 1 Mbps 691.2 kbps 1.3824 Mbps 921.6 kbps 1.8432 Mbps
Table 18-12. Examples of UBRRn Settings for Commonly Used Oscillator Frequencies (Continued)

<table>
<thead>
<tr>
<th>Baud Rate (bps)</th>
<th>f&lt;sub&gt;osc&lt;/sub&gt; = 16.0000 MHz</th>
<th>f&lt;sub&gt;osc&lt;/sub&gt; = 18.4320 MHz</th>
<th>f&lt;sub&gt;osc&lt;/sub&gt; = 20.0000 MHz</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>UBRR</td>
<td>Error</td>
<td>UBRR</td>
</tr>
<tr>
<td>2400</td>
<td>416</td>
<td>-0.1%</td>
<td>832</td>
</tr>
<tr>
<td>4800</td>
<td>207</td>
<td>0.2%</td>
<td>416</td>
</tr>
<tr>
<td>9600</td>
<td>103</td>
<td>0.2%</td>
<td>207</td>
</tr>
<tr>
<td>14.4k</td>
<td>68</td>
<td>0.6%</td>
<td>138</td>
</tr>
<tr>
<td>19.2k</td>
<td>51</td>
<td>0.2%</td>
<td>103</td>
</tr>
<tr>
<td>28.8k</td>
<td>34</td>
<td>-0.8%</td>
<td>68</td>
</tr>
<tr>
<td>38.4k</td>
<td>25</td>
<td>0.2%</td>
<td>51</td>
</tr>
<tr>
<td>57.6k</td>
<td>16</td>
<td>2.1%</td>
<td>34</td>
</tr>
<tr>
<td>76.8k</td>
<td>12</td>
<td>0.2%</td>
<td>25</td>
</tr>
<tr>
<td>115.2k</td>
<td>8</td>
<td>-3.5%</td>
<td>16</td>
</tr>
<tr>
<td>230.4k</td>
<td>3</td>
<td>8.5%</td>
<td>8</td>
</tr>
<tr>
<td>250k</td>
<td>3</td>
<td>0.0%</td>
<td>7</td>
</tr>
<tr>
<td>0.5M</td>
<td>1</td>
<td>0.0%</td>
<td>3</td>
</tr>
<tr>
<td>1M</td>
<td>0</td>
<td>0.0%</td>
<td>1</td>
</tr>
<tr>
<td>Max. (1)</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

1. UBRR = 0, Error = 0.0%

1 Mbps | 2 Mbps | 1.152 Mbps | 2.304 Mbps | 1.25 Mbps | 2.5 Mbps
19. USART in SPI Mode

The Universal Synchronous and Asynchronous serial Receiver and Transmitter (USART) can be set to a master SPI compliant mode of operation. The Master SPI Mode (MSPIM) has the following features:

- Full Duplex, Three-wire Synchronous Data Transfer
- Master Operation
- Supports all four SPI Modes of Operation (Mode 0, 1, 2, and 3)
- LSB First or MSB First Data Transfer (Configurable Data Order)
- Queued Operation (Double Buffered)
- High Resolution Baud Rate Generator
- High Speed Operation (FXCKmax = fCK/2)
- Flexible Interrupt Generation

19.1 Overview

Setting both UMSELn1:0 bits to one enables the USART in MSPIM logic. In this mode of operation the SPI master control logic takes direct control over the USART resources. These resources include the transmitter and receiver shift register and buffers, and the baud rate generator. The parity generator and checker, the data and clock recovery logic, and the RX and TX control logic is disabled. The USART RX and TX control logic is replaced by a common SPI transfer control logic. However, the pin control logic and interrupt generation logic is identical in both modes of operation.

The I/O register locations are the same in both modes. However, some of the functionality of the control registers changes when using MSPIM.

19.2 Clock Generation

The Clock Generation logic generates the base clock for the Transmitter and Receiver. For USART MSPIM mode of operation only internal clock generation (i.e. master operation) is supported. The Data Direction Register for the XCKn pin (DDR_XCKn) must therefore be set to one (i.e. as output) for the USART in MSPIM to operate correctly. Preferably the DDR_XCKn should be set up before the USART in MSPIM is enabled (i.e. TXENn and RXENn bit set to one).

The internal clock generation used in MSPIM mode is identical to the USART synchronous master mode. The baud rate or UBRRn setting can therefore be calculated using the same equations, see Table 19-1:

Table 19-1. Equations for Calculating Baud Rate Register Setting

<table>
<thead>
<tr>
<th>Operating Mode</th>
<th>Equation for Calculating Baud Rate(1)</th>
<th>Equation for Calculating UBRRn Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Synchronous Master mode</td>
<td>$BAUD = \frac{f_{OSC}}{2(UBRRn + 1)}$</td>
<td>$UBRRn = \frac{f_{OSC}}{2BAUD} - 1$</td>
</tr>
</tbody>
</table>
Note: 1. The baud rate is defined to be the transfer rate in bit per second (bps)

<table>
<thead>
<tr>
<th>BAUD</th>
<th>Baud rate (in bits per second, bps)</th>
</tr>
</thead>
<tbody>
<tr>
<td>f_{osc}</td>
<td>System Oscillator clock frequency</td>
</tr>
<tr>
<td>UBRRn</td>
<td>Contents of the UBRRnH and UBRRnL Registers, (0-4095)</td>
</tr>
</tbody>
</table>

19.3 SPI Data Modes and Timing

There are four combinations of XCKn (SCK) phase and polarity with respect to serial data, which are determined by control bits UCPHAn and UCPOLn. The data transfer timing diagrams are shown in Figure 19-1. Data bits are shifted out and latched in on opposite edges of the XCKn signal, ensuring sufficient time for data signals to stabilize. The UCPOLn and UCPHAn functionality is summarized in Table 19-2. Note that changing the setting of any of these bits will corrupt all ongoing communication for both the Receiver and Transmitter.

Table 19-2. UCPOLn and UCPHAn Functionality

<table>
<thead>
<tr>
<th>UCPOLn</th>
<th>UCPHAn</th>
<th>SPI Mode</th>
<th>Leading Edge</th>
<th>Trailing Edge</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>Sample (Rising)</td>
<td>Setup (Falling)</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>Setup (Rising)</td>
<td>Sample (Falling)</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>2</td>
<td>Sample (Falling)</td>
<td>Setup (Rising)</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>3</td>
<td>Setup (Falling)</td>
<td>Sample (Rising)</td>
</tr>
</tbody>
</table>

Figure 19-1. UCPHAn and UCPOLn data transfer timing diagrams.

19.4 Frame Formats

A serial frame for the MSPIM is defined to be one character of 8 data bits. The USART in MSPIM mode has two valid frame formats:

- 8-bit data with MSB first
- 8-bit data with LSB first

A frame starts with the least or most significant data bit. Then the next data bits, up to a total of eight, are succeeding, ending with the most or least significant bit accordingly. When a complete frame is transmitted, a new frame can directly follow it, or the communication line can be set to an idle (high) state.
The UDORDn bit in UCSRnC sets the frame format used by the USART in MSPIM mode. The Receiver and Transmitter use the same setting. Note that changing the setting of any of these bits will corrupt all ongoing communication for both the Receiver and Transmitter.

16-bit data transfer can be achieved by writing two data bytes to UDRn. A UART transmit complete interrupt will then signal that the 16-bit value has been shifted out.

19.4.1 USART MSPIM Initialization

The USART in MSPIM mode has to be initialized before any communication can take place. The initialization process normally consists of setting the baud rate, setting master mode of operation (by setting DDR_XCKn to one), setting frame format and enabling the Transmitter and the Receiver. Only the transmitter can operate independently. For interrupt driven USART operation, the Global Interrupt Flag should be cleared (and thus interrupts globally disabled) when doing the initialization.

Note: To ensure immediate initialization of the XCKn output the baud-rate register (UBRRn) must be zero at the time the transmitter is enabled. Contrary to the normal mode USART operation the UBRRn must then be written to the desired value after the transmitter is enabled, but before the first transmission is started. Setting UBRRn to zero before enabling the transmitter is not necessary if the initialization is done immediately after a reset since UBRRn is reset to zero.

Before doing a re-initialization with changed baud rate, data mode, or frame format, be sure that there is no ongoing transmissions during the period the registers are changed. The TXCn Flag can be used to check that the Transmitter has completed all transfers, and the RXCn Flag can be used to check that there are no unread data in the receive buffer. Note that the TXCn Flag must be cleared before each transmission (before UDRn is written) if it is used for this purpose.

The following simple USART initialization code examples show one assembly and one C function that are equal in functionality. The examples assume polling (no interrupts enabled). The baud rate is given as a function parameter. For the assembly code, the baud rate parameter is assumed to be stored in the r17:r16 registers.
Assembly Code Example

```assembly
UART_Init:
    clr r18
    out UBRRnH, r18
    out UBRRnL, r18
    ; Setting the XCKn port pin as output, enables master mode.
    sbi XCKn_DDR, XCKn
    ; Set MSPI mode of operation and SPI data mode 0.
    ldi r18, (1<<UMSELn1)| (1<<UMSELn0)| (0<<UCPHAn)| (0<<UCPOLn)
    out UCSRnC, r18
    ; Enable receiver and transmitter.
    ldi r18, (1<<RXEnn)| (1<<TXEnn)
    out UCSRnB, r18
    ; Set baud rate. 
    ; IMPORTANT: The Baud Rate must be set after the transmitter is enabled!
    out UBRRnH, r17
    out UBRRnL, r18
    ret
```

C Code Example

```c
void UART_Init( unsigned int baud )
{
    UBRRn = 0;
    /* Setting the XCKn port pin as output, enables master mode. */
    XCKn_DDR |= (1<<XCKn);
    /* Set MSPI mode of operation and SPI data mode 0. */
    UCSRnC = (1<<UMSELn1)| (1<<UMSELn0)| (0<<UCPHAn)| (0<<UCPOLn);
    /* Enable receiver and transmitter. */
    UCSRnB = (1<<RXEnn)| (1<<TXEnn);
    /* Set baud rate. */
    /* IMPORTANT: The Baud Rate must be set after the transmitter is enabled */
    UBRRn = baud;
}
```

Note: 1. See “Code Examples” on page 8.

### 19.5 Data Transfer

Using the UART in MSPI mode requires the Transmitter to be enabled, i.e. the TXEnn bit in the UCSRnB register is set to one. When the Transmitter is enabled, the normal port operation of the TxDn pin is overridden and given the function as the Transmitter’s serial output. Enabling the receiver is optional and is done by setting the RXEnn bit in the UCSRnB register to one. When the receiver is enabled, the normal pin operation of the RxDn pin is overridden and given the function as the Receiver’s serial input. The XCKn will in both cases be used as the transfer clock.
After initialization the USART is ready for doing data transfers. A data transfer is initiated by writing to the UDn I/O location. This is the case for both sending and receiving data since the transmitter controls the transfer clock. The data written to UDn is moved from the transmit buffer to the shift register when the shift register is ready to send a new frame.

Note: To keep the input buffer in sync with the number of data bytes transmitted, the UDn register must be read once for each byte transmitted. The input buffer operation is identical to normal USART mode, i.e. if an overflow occurs the character last received will be lost, not the first data in the buffer. This means that if four bytes are transferred, byte 1 first, then byte 2, 3, and 4, and the UDn is not read before all transfers are completed, then byte 3 to be received will be lost, and not byte 1.

The following code examples show a simple USART in MSPI mode transfer function based on polling the Data Register Empty (UDREn) Flag and the Receive Complete (RXCn) Flag. The USART has to be initialized before the function can be used. For the assembly code, the data to be sent is assumed to be stored in Register R16 and the data received will be available in the same register (R16) after the function returns.

The function simply waits for the transmit buffer to be empty by checking the UDREn Flag, before loading it with new data to be transmitted. The function then waits for data to be present in the receive buffer by checking the RXCn Flag, before reading the buffer and returning the value.

---

### Assembly Code Example

```
USART_MSPI_Transfer:
    ; Wait for empty transmit buffer
    sbis UCSnA, UDREn
    rjmp USART_MSPI_Transfer

    ; Put data (rl6) into buffer, sends the data
    out UDn, rl6
    ; Wait for data to be received

USART_MSPI_Wait_RXCn:
    sbis UCSnA, RXCn
    rjmp USART_MSPI_Wait_RXCn

    ; Get and return received data from buffer
    in rl6, UDn
    ret
```

### C Code Example

```
unsigned char USART_Receive( void )
{
    /* Wait for empty transmit buffer */
    while ( !(UCSRnA & (1<<UDREn)) );
    /* Put data into buffer, sends the data */
    UDn = data;
    /* Wait for data to be received */
    while ( !(UCSRnA & (1<<RXCn)) );
    /* Get and return received data from buffer */
    return UDn;
}
```
19.5.1 Transmitter and Receiver Flags and Interrupts
The RXCn, TXCn, and UDREN flags and corresponding interrupts in USART in MSPIM mode are identical in function to the normal USART operation. However, the receiver error status flags (FE, DOR, and PE) are not in use and is always read as zero.

19.5.2 Disabling the Transmitter or Receiver
The disabling of the transmitter or receiver in USART in MSPIM mode is identical in function to the normal USART operation.

19.6 USART MSPIM Register Description
The following section describes the registers used for SPI operation using the USART.

19.6.1 USART MSPIM I/O Data Register - UDRn
The function and bit description of the USART data register (UDRn) in MSPI mode is identical to normal USART operation. See “USART I/O Data Register n– UDRn” on page 205.

19.6.2 USART MSPIM Control and Status Register n A - UCSRnA

<table>
<thead>
<tr>
<th>Bit</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>RXCn</td>
<td>TXCn</td>
<td>UDREN</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>UCSRnA</td>
</tr>
<tr>
<td>Read/Write</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
</tr>
<tr>
<td>Initial Value</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

- **Bit 7 - RXCn**: USART Receive Complete
  This flag bit is set when there are unread data in the receive buffer and cleared when the receive buffer is empty (i.e., does not contain any unread data). If the Receiver is disabled, the receive buffer will be flushed and consequently the RXCn bit will become zero. The RXCn Flag can be used to generate a Receive Complete interrupt (see description of the RXClEn bit).

- **Bit 6 - TXCn**: USART Transmit Complete
  This flag bit is set when the entire frame in the Transmit Shift Register has been shifted out and there are no new data currently present in the transmit buffer (UDRn). The TXCn Flag bit is automatically cleared when a transmit complete interrupt is executed, or it can be cleared by writing a one to its bit location. The TXCn Flag can generate a Transmit Complete interrupt (see description of the TXClEn bit).

- **Bit 5 - UDREN**: USART Data Register Empty
  The UDREN Flag indicates if the transmit buffer (UDRn) is ready to receive new data. If UDREN is one, the buffer is empty, and therefore ready to be written. The UDREN Flag can generate a Data Register Empty interrupt (see description of the UDRIE bit). UDREN is set after a reset to indicate that the Transmitter is ready.

- **Bit 4:0 - Reserved Bits in MSPI mode
  When in MSPI mode, these bits are reserved for future use. For compatibility with future devices, these bits must be written to zero when UCSRnA is written.
19.6.3  USART MSPIM Control and Status Register n B - UCSRnB

<table>
<thead>
<tr>
<th>Bit</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
<th>Read/Write</th>
<th>Initial Value</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>R/W</td>
<td>0 0 0 0 0 0 1 1 0</td>
</tr>
<tr>
<td>RXCEn</td>
<td>TXCEn</td>
<td>UDRIE</td>
<td>RXEn</td>
<td>TXEn</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>UCSRnB</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **Bit 7 - RXCEn: RX Complete Interrupt Enable**
  Writing this bit to one enables interrupt on the RXCn Flag. A USART Receive Complete interrupt will be generated only if the RXCEn bit is written to one, the Global Interrupt Flag in SREG is written to one and the RXCn bit in UCSRnA is set.

- **Bit 6 - TXCEn: TX Complete Interrupt Enable**
  Writing this bit to one enables interrupt on the TXCn Flag. A USART Transmit Complete interrupt will be generated only if the TXCEn bit is written to one, the Global Interrupt Flag in SREG is written to one and the TXCn bit in UCSRnA is set.

- **Bit 5 - UDRIE: USART Data Register Empty Interrupt Enable**
  Writing this bit to one enables interrupt on the UDREn Flag. A Data Register Empty interrupt will be generated only if the UDRIE bit is written to one, the Global Interrupt Flag in SREG is written to one and the UDREn bit in UCSRnA is set.

- **Bit 4 - RXEn: Receiver Enable**
  Writing this bit to one enables the USART Receiver in MSPIM mode. The Receiver will override normal port operation for the RxDn pin when enabled. Disabling the Receiver will flush the receive buffer. Only enabling the receiver in MSPI mode (i.e. setting RXEn=1 and TXEn=0) has no meaning since it is the transmitter that controls the transfer clock and since only master mode is supported.

- **Bit 3 - TXEn: Transmitter Enable**
  Writing this bit to one enables the USART Transmitter. The Transmitter will override normal port operation for the TxDn pin when enabled. The disabling of the Transmitter (writing TXEn to zero) will not become effective until ongoing and pending transmissions are completed, i.e., when the Transmit Shift Register and Transmit Buffer Register do not contain data to be transmitted. When disabled, the Transmitter will no longer override the TxDn port.

- **Bit 2:0 - Reserved Bits in MSPI mode**
  When in MSPI mode, these bits are reserved for future use. For compatibility with future devices, these bits must be written to zero when UCSRnB is written.

19.6.4  USART MSPIM Control and Status Register n C - UCSRnC

<table>
<thead>
<tr>
<th>Bit</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
<th>Read/Write</th>
<th>Initial Value</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>R/W</td>
<td>0 0 0 0 0 0 1 1 0</td>
</tr>
<tr>
<td>UMSELn1</td>
<td>UMSELn0</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>UDORDn</td>
<td>UCPHAn</td>
<td>UCPOLn</td>
<td>UCSRnC</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

ATmega16/32U4

7766F–AVR–11/10
• Bit 7:6 - UMSELn1:0: USART Mode Select
These bits select the mode of operation of the USART as shown in Table 19-3. See “USART Control and Status Register n C – UCSRnC” on page 207 for full description of the normal USART operation. The MSPIM is enabled when both UMSELn bits are set to one. The UDORDn, UCPHAn, and UCPOLn can be set in the same write operation where the MSPIM is enabled.

Table 19-3. UMSELn Bits Settings

<table>
<thead>
<tr>
<th>UMSELn1</th>
<th>UMSELn0</th>
<th>Mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>Asynchronous USART</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>Synchronous USART</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>(Reserved)</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>Master SPI (MSPIM)</td>
</tr>
</tbody>
</table>

• Bit 5:3 - Reserved Bits in MSPI mode
When in MSPI mode, these bits are reserved for future use. For compatibility with future devices, these bits must be written to zero when UCSRnC is written.

• Bit 2 - UDORDn: Data Order
When set to one the LSB of the data word is transmitted first. When set to zero the MSB of the data word is transmitted first. Refer to the Frame Formats section page 4 for details.

• Bit 1 - UCPHAn: Clock Phase
The UCPHAn bit setting determine if data is sampled on the leasing edge (first) or tailing (last) edge of XCKn. Refer to the SPI Data Modes and Timing section page 4 for details.

• Bit 0 - UCPOLn: Clock Polarity
The UCPOLn bit sets the polarity of the XCKn clock. The combination of the UCPOLn and UCPHAn bit settings determine the timing of the data transfer. Refer to the SPI Data Modes and Timing section page 4 for details.

19.6.5 USART MSPIM Baud Rate Registers - UBRRnL and UBRRnH
The function and bit description of the baud rate registers in MSPI mode is identical to normal USART operation. See “USART Baud Rate Registers – UBRRLn and UBRRHn” on page 209.

19.7 AVR USART MSPIM vs. AVR SPI
The USART in MSPIM mode is fully compatible with the AVR SPI regarding:
• Master mode timing diagram.
• The UCPOLn bit functionality is identical to the SPI CPOL bit.
• The UCPHAn bit functionality is identical to the SPI CPHA bit.
• The UDORDn bit functionality is identical to the SPI DORD bit.
However, since the USART in MSPIM mode reuses the USART resources, the use of the USART in MSPIM mode is somewhat different compared to the SPI. In addition to differences of the control register bits, and that only master operation is supported by the USART in MSPIM mode, the following features differ between the two modules:
• The USART in MSPIM mode includes (double) buffering of the transmitter. The SPI has no buffer.
• The USART in MSPIM mode receiver includes an additional buffer level.
• The SPI WCOL (Write Collision) bit is not included in USART in MSPIM mode.
• The SPI double speed mode (SPI2X) bit is not included. However, the same effect is achieved by setting UBRRn accordingly.
• Interrupt timing is not compatible.
• Pin control differs due to the master only operation of the USART in MSPIM mode.

A comparison of the USART in MSPIM mode and the SPI pins is shown in Table 19-4 on page 222.

Table 19-4. Comparison of USART in MSPIM mode and SPI pins.

<table>
<thead>
<tr>
<th>USART_MSPIM</th>
<th>SPI</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>TxDn</td>
<td>MOSI</td>
<td>Master Out only</td>
</tr>
<tr>
<td>RxDn</td>
<td>MISO</td>
<td>Master In only</td>
</tr>
<tr>
<td>XCKn</td>
<td>SCK</td>
<td>(Functionally identical)</td>
</tr>
<tr>
<td>(N/A)</td>
<td>S/S</td>
<td>Not supported by USART in MSPIM</td>
</tr>
</tbody>
</table>
20. 2-wire Serial Interface

20.1 Features

- Simple Yet Powerful and Flexible Communication Interface, only two Bus Lines Needed
- Both Master and Slave Operation Supported
- Device can Operate as Transmitter or Receiver
- 7-bit Address Space Allows up to 128 Different Slave Addresses
- Multi-master Arbitration Support
- Up to 400 kHz Data Transfer Speed
- Slew-rate Limited Output Drivers
- Noise Suppression Circuitry Rejects Spikes on Bus Lines
- Fully Programmable Slave Address with General Call Support
- Address Recognition Causes Wake-up When AVR is in Sleep Mode

20.2 2-wire Serial Interface Bus Definition

The 2-wire Serial Interface (TWI) is ideally suited for typical microcontroller applications. The TWI protocol allows the systems designer to interconnect up to 128 different devices using only two bi-directional bus lines, one for clock (SCL) and one for data (SDA). The only external hardware needed to implement the bus is a single pull-up resistor for each of the TWI bus lines. All devices connected to the bus have individual addresses, and mechanisms for resolving bus contention are inherent in the TWI protocol.

Figure 20-1. TWI Bus Interconnection

\[\text{Figure 20-1. TWI Bus Interconnection}\]

20.2.1 TWI Terminology

The following definitions are frequently encountered in this section.

Table 20-1. TWI Terminology

<table>
<thead>
<tr>
<th>Term</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Master</td>
<td>The device that initiates and terminates a transmission. The Master also generates the SCL clock.</td>
</tr>
<tr>
<td>Slave</td>
<td>The device addressed by a Master.</td>
</tr>
<tr>
<td>Transmitter</td>
<td>The device placing data on the bus.</td>
</tr>
<tr>
<td>Receiver</td>
<td>The device reading data from the bus.</td>
</tr>
</tbody>
</table>
The Power Reduction TWI bit, PRTWI bit in “Power Reduction Register 0 - PRR0” on page 46 must be written to zero to enable the 2-wire Serial Interface.

20.2.2 Electrical Interconnection

As depicted in Figure 20-1, both bus lines are connected to the positive supply voltage through pull-up resistors. The bus drivers of all TWI-compliant devices are open-drain or open-collector. This implements a wired-AND function which is essential to the operation of the interface. A low level on a TWI bus line is generated when one or more TWI devices output a zero. A high level is output when all TWI devices trim-state their outputs, allowing the pull-up resistors to pull the line high. Note that all AVR devices connected to the TWI bus must be powered in order to allow any bus operation.

The number of devices that can be connected to the bus is only limited by the bus capacitance limit of 400 pF and the 7-bit slave address space. A detailed specification of the electrical characteristics of the TWI is given in “SPI Timing Characteristics” on page 383. Two different sets of specifications are presented there, one relevant for bus speeds below 100 kHz, and one valid for bus speeds up to 400 kHz.

20.3 Data Transfer and Frame Format

20.3.1 Transferring Bits

Each data bit transferred on the TWI bus is accompanied by a pulse on the clock line. The level of the data line must be stable when the clock line is high. The only exception to this rule is for generating start and stop conditions.

Figure 20-2. Data Validity

20.3.2 START and STOP Conditions

The Master initiates and terminates a data transmission. The transmission is initiated when the Master issues a START condition on the bus, and it is terminated when the Master issues a STOP condition. Between a START and a STOP condition, the bus is considered busy, and no other master should try to seize control of the bus. A special case occurs when a new START condition is issued between a START and STOP condition. This is referred to as a REPEATED START condition, and is used when the Master wishes to initiate a new transfer without relinquishing control of the bus. After a REPEATED START, the bus is considered busy until the next STOP. This is identical to the START behavior, and therefore START is used to describe both START and REPEATED START for the remainder of this datasheet, unless otherwise noted. As
depicted below, START and STOP conditions are signalled by changing the level of the SDA line when the SCL line is high.

**Figure 20-3.** START, REPEATED START and STOP conditions

20.3.3 Address Packet Format

All address packets transmitted on the TWI bus are 9 bits long, consisting of 7 address bits, one READ/WRITE control bit and an acknowledge bit. If the READ/WRITE bit is set, a read operation is to be performed, otherwise a write operation should be performed. When a Slave recognizes that it is being addressed, it should acknowledge by pulling SDA low in the ninth SCL (ACK) cycle. If the addressed Slave is busy, or for some other reason can not service the Master’s request, the SDA line should be left high in the ACK clock cycle. The Master can then transmit a STOP condition, or a REPEATED START condition to initiate a new transmission. An address packet consisting of a slave address and a READ or a WRITE bit is called SLA+R or SLA+W, respectively.

The MSB of the address byte is transmitted first. Slave addresses can freely be allocated by the designer, but the address 0000 000 is reserved for a general call.

When a general call is issued, all slaves should respond by pulling the SDA line low in the ACK cycle. A general call is used when a Master wishes to transmit the same message to several slaves in the system. When the general call address followed by a Write bit is transmitted on the bus, all slaves set up to acknowledge the general call will pull the SDA line low in the ack cycle. The following data packets will then be received by all the slaves that acknowledged the general call. Note that transmitting the general call address followed by a Read bit is meaningless, as this would cause contention if several slaves started transmitting different data.

All addresses of the format 1111 xxx should be reserved for future purposes.

**Figure 20-4.** Address Packet Format
20.3.4 Data Packet Format

All data packets transmitted on the TWI bus are nine bits long, consisting of one data byte and
an acknowledge bit. During a data transfer, the Master generates the clock and the START and
STOP conditions, while the Receiver is responsible for acknowledging the reception. An
Acknowledgment (ACK) is signalled by the Receiver pulling the SDA line low during the ninth SCL
cycle. If the Receiver leaves the SDA line high, a NACK is signalled. When the Receiver has
received the last byte, or for some reason cannot receive any more bytes, it should inform the
Transmitter by sending a NACK after the final byte. The MSB of the data byte is transmitted first.

Figure 20-5. Data Packet Format

20.3.5 Combining Address and Data Packets into a Transmission

A transmission basically consists of a START condition, a SLA+R/W, one or more data packets
and a STOP condition. An empty message, consisting of a START followed by a STOP condi-
tion, is illegal. Note that the Wired-ANDing of the SCL line can be used to implement
handshaking between the Master and the Slave. The Slave can extend the SCL low period by
pulling the SCL line low. This is useful if the clock speed set up by the Master is too fast for the
Slave, or the Slave needs extra time for processing between the data transmissions. The Slave
extending the SCL low period will not affect the SCL high period, which is determined by the
Master. As a consequence, the Slave can reduce the TWI data transfer speed by prolonging the
SCL duty cycle.

Figure 20-6 shows a typical data transmission. Note that several data bytes can be transmitted
between the SLA+R/W and the STOP condition, depending on the software protocol imple-
mented by the application software.

Figure 20-6. Typical Data Transmission
20.4 Multi-master Bus Systems, Arbitration and Synchronization

The TWI protocol allows bus systems with several masters. Special concerns have been taken in order to ensure that transmissions will proceed as normal, even if two or more masters initiate a transmission at the same time. Two problems arise in multi-master systems:

- An algorithm must be implemented allowing only one of the masters to complete the transmission. All other masters should cease transmission when they discover that they have lost the selection process. This selection process is called arbitration. When a contending master discovers that it has lost the arbitration process, it should immediately switch to Slave mode to check whether it is being addressed by the winning master. The fact that multiple masters have started transmission at the same time should not be detectable to the slaves, i.e. the data being transferred on the bus must not be corrupted.

- Different masters may use different SCL frequencies. A scheme must be devised to synchronize the serial clocks from all masters, in order to let the transmission proceed in a lockstep fashion. This will facilitate the arbitration process.

The wired-ANDing of the bus lines is used to solve both these problems. The serial clocks from all masters will be wired-ANDed, yielding a combined clock with a high period equal to the one from the Master with the shortest high period. The low period of the combined clock is equal to the low period of the Master with the longest low period. Note that all masters listen to the SCL line, effectively starting to count their SCL high and low time-out periods when the combined SCL line goes high or low, respectively.

![Figure 20-7. SCL Synchronization Between Multiple Masters]

Arbitration is carried out by all masters continuously monitoring the SDA line after outputting data. If the value read from the SDA line does not match the value the Master had output, it has lost the arbitration. Note that a Master can only lose arbitration when it outputs a high SDA value while another Master outputs a low value. The losing Master should immediately go to Slave mode, checking if it is being addressed by the winning Master. The SDA line should be left high, but losing masters are allowed to generate a clock signal until the end of the current data or address packet. Arbitration will continue until only one Master remains, and this may take many
bits. If several masters are trying to address the same Slave, arbitration will continue into the data packet.

**Figure 20-8.** Arbitration Between Two Masters

Note that arbitration is not allowed between:

- A REPEATED START condition and a data bit.
- A STOP condition and a data bit.
- A REPEATED START and a STOP condition.

It is the user software’s responsibility to ensure that these illegal arbitration conditions never occur. This implies that in multi-master systems, all data transfers must use the same composition of SLA+R/W and data packets. In other words: All transmissions must contain the same number of data packets, otherwise the result of the arbitration is undefined.

**20.5 Overview of the TWI Module**

The TWI module is comprised of several submodules, as shown in **Figure 20-9**. All registers drawn in a thick line are accessible through the AVR data bus.
20.5.1 SCL and SDA Pins

These pins interface the AVR TWI with the rest of the MCU system. The output drivers contain a slew-rate limiter in order to conform to the TWI specification. The input stages contain a spike suppression unit removing spikes shorter than 50 ns. Note that the internal pull-ups in the AVR pads can be enabled by setting the PORT bits corresponding to the SCL and SDA pins, as explained in the I/O Port section. The internal pull-ups can in some systems eliminate the need for external ones.

20.5.2 Bit Rate Generator Unit

This unit controls the period of SCL when operating in a Master mode. The SCL period is controlled by settings in the TWI Bit Rate Register (TWBR) and the Prescaler bits in the TWI Status Register (TWSR). Slave operation does not depend on Bit Rate or Prescaler settings, but the CPU clock frequency in the Slave must be at least 16 times higher than the SCL frequency. Note that slaves may prolong the SCL low period, thereby reducing the average TWI bus clock period. The SCL frequency is generated according to the following equation:
SCL frequency = \frac{\text{CPU Clock frequency}}{16 + 2(\text{TWBR}) \cdot 4^{\text{TWPS}}}

- TWBR = Value of the TWI Bit Rate Register.
- TWPS = Value of the prescaler bits in the TWI Status Register.

Note: TWBR should be 10 or higher if the TWI operates in Master mode. If TWBR is lower than 10, the Master may produce an incorrect output on SDA and SCL for the remainder of the byte. The problem occurs when operating the TWI in Master mode, sending Start + SLA + R/W to a Slave (a Slave does not need to be connected to the bus for the condition to happen).

20.5.3 Bus Interface Unit

This unit contains the Data and Address Shift Register (TWDR), a START/STOP Controller and Arbitration detection hardware. The TWDR contains the address or data bytes to be transmitted, or the address or data bytes received. In addition to the 8-bit TWDR, the Bus Interface Unit also contains a register containing the (N)ACK bit to be transmitted or received. This (N)ACK Register is not directly accessible by the application software. However, when receiving, it can be set or cleared by manipulating the TWI Control Register (TWCR). When in Transmitter mode, the value of the received (N)ACK bit can be determined by the value in the TWIS.

The START/STOP Controller is responsible for generation and detection of START, REPEATED START, and STOP conditions. The START/STOP controller is able to detect START and STOP conditions even when the AVR MCU is in one of the sleep modes, enabling the MCU to wake up if addressed by a Master.

If the TWI has initiated a transmission as Master, the Arbitration Detection hardware continuously monitors the transmission trying to determine if arbitration is in process. If the TWI has lost an arbitration, the Control Unit is informed. Correct action can then be taken and appropriate status codes generated.

20.5.4 Address Match Unit

The Address Match unit checks if received address bytes match the seven-bit address in the TWI Address Register (TWAR). If the TWI General Call Recognition Enable (TWGCE) bit in the TWAR is written to one, all incoming address bits will also be compared against the General Call address. Upon an address match, the Control Unit is informed, allowing correct action to be taken. The TWI may or may not acknowledge its address, depending on settings in the TWCR. The Address Match unit is able to compare addresses even when the AVR MCU is in sleep mode, enabling the MCU to wake up if addressed by a Master. If another interrupt (e.g., INT0) occurs during TWI Power-down address match and wakes up the CPU, the TWI aborts operation and return to it’s idle state. If this cause any problems, ensure that TWI Address Match is the only enabled interrupt when entering Power-down.

20.5.5 Control Unit

The Control unit monitors the TWI bus and generates responses corresponding to settings in the TWI Control Register (TWCR). When an event requiring the attention of the application occurs on the TWI bus, the TWI Interrupt Flag (TWINT) is asserted. In the next clock cycle, the TWI Status Register (TWSR) is updated with a status code identifying the event. The TWSR only contains relevant status information when the TWI Interrupt Flag is asserted. At all other times, the TWSR contains a special status code indicating that no relevant status information is available. As long as the TWINT Flag is set, the SCL line is held low. This allows the application software to complete its tasks before allowing the TWI transmission to continue.
The TWINT Flag is set in the following situations:

- After the TWI has transmitted a START/REPEATED START condition.
- After the TWI has transmitted SLA+R/W.
- After the TWI has transmitted an address byte.
- After the TWI has lost arbitration.
- After the TWI has been addressed by own slave address or general call.
- After the TWI has received a data byte.
- After a STOP or REPEATED START has been received while still addressed as a Slave.
- When a bus error has occurred due to an illegal START or STOP condition.

### 20.6 TWI Register Description

#### 20.6.1 TWI Bit Rate Register – TWBR

<table>
<thead>
<tr>
<th>Bit</th>
<th>TWBR7</th>
<th>TWBR6</th>
<th>TWBR5</th>
<th>TWBR4</th>
<th>TWBR3</th>
<th>TWBR2</th>
<th>TWBR1</th>
<th>TWBR0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read/Write</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
</tr>
</tbody>
</table>

Initial Value: 0 0 0 0 0 0 0 0

- **Bits 7..0 – TWI Bit Rate Register**
  
  TWBR selects the division factor for the bit rate generator. The bit rate generator is a frequency divider which generates the SCL clock frequency in the Master modes. See “Bit Rate Generator Unit” on page 229 for calculating bit rates.

#### 20.6.2 TWI Control Register – TWCR

<table>
<thead>
<tr>
<th>Bit</th>
<th>TWINT</th>
<th>TWEA</th>
<th>TWSTA</th>
<th>TWSTO</th>
<th>TWWC</th>
<th>TWEN</th>
<th>–</th>
<th>TWIE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read/Write</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R</td>
<td>R/W</td>
<td>R</td>
<td>R/W</td>
</tr>
</tbody>
</table>

Initial Value: 0 0 0 0 0 0 0 0

The TWCR is used to control the operation of the TWI. It is used to enable the TWI, to initiate a Master access by applying a START condition to the bus, to generate a Receiver acknowledge, to generate a stop condition, and to control halting of the bus while the data to be written to the bus are written to the TWDR. It also indicates a write collision if data is attempted written to TWDR while the register is inaccessible.

- **Bit 7 – TWINT: TWI Interrupt Flag**

  This bit is set by hardware when the TWI has finished its current job and expects application software response. If the I-bit in SREG and TWIE in TWCR are set, the MCU will jump to the TWI Interrupt Vector. While the TWINT Flag is set, the SCL low period is stretched. The TWINT Flag must be cleared by software by writing a logic one to it. Note that this flag is not automatically cleared by hardware when executing the interrupt routine. Also note that clearing this flag starts the operation of the TWI, so all accesses to the TWI Address Register (TWAR), TWI Status Register (TWSR), and TWI Data Register (TWDR) must be complete before clearing this flag.

- **Bit 6 – TWEA: TWI Enable Acknowledge Bit**

  The TWEA bit controls the generation of the acknowledge pulse. If the TWEA bit is written to one, the ACK pulse is generated on the TWI bus if the following conditions are met:
1. The device’s own slave address has been received.
2. A general call has been received, while the TWGCE bit in the TWAR is set.
3. A data byte has been received in Master Receiver or Slave Receiver mode.

By writing the TWEA bit to zero, the device can be virtually disconnected from the 2-wire Serial Bus temporarily. Address recognition can then be resumed by writing the TWEA bit to one again.

- **Bit 5 – TWSTA: TWI START Condition Bit**
The application writes the TWSTA bit to one when it desires to become a Master on the 2-wire Serial Bus. The TWI hardware checks if the bus is available, and generates a START condition on the bus if it is free. However, if the bus is not free, the TWI waits until a STOP condition is detected, and then generates a new START condition to claim the bus Master status. TWSTA must be cleared by software when the START condition has been transmitted.

- **Bit 4 – TWSTO: TWI STOP Condition Bit**
Writing the TWSTO bit to one in Master mode will generate a STOP condition on the 2-wire Serial Bus. When the STOP condition is executed on the bus, the TWSTO bit is cleared automatically. In Slave mode, setting the TWSTO bit can be used to recover from an error condition. This will not generate a STOP condition, but the TWI returns to a well-defined unaddressed Slave mode and releases the SCL and SDA lines to a high impedance state.

- **Bit 3 – TWWC: TWI Write Collision Flag**
The TWWC bit is set when attempting to write to the TWI Data Register – TWDR when TWINT is low. This flag is cleared by writing the TWDR Register when TWINT is high.

- **Bit 2 – TWEN: TWI Enable Bit**
The TWEN bit enables TWI operation and activates the TWI interface. When TWEN is written to one, the TWI takes control over the I/O pins connected to the SCL and SDA pins, enabling the slew-rate limiters and spike filters. If this bit is written to zero, the TWI is switched off and all TWI transmissions are terminated, regardless of any ongoing operation.

- **Bit 1 – Res: Reserved Bit**
This bit is a reserved bit and will always read as zero.

- **Bit 0 – TWIE: TWI Interrupt Enable**
When this bit is written to one, and the I-bit in SREG is set, the TWI interrupt request will be activated for as long as the TWINT Flag is high.

### 20.6.3 TWI Status Register – TWSR

<table>
<thead>
<tr>
<th>Bit</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>TWSR</td>
<td>TW7</td>
<td>TW6</td>
<td>TW5</td>
<td>TW4</td>
<td>TW3</td>
<td>–</td>
<td>TWPS1</td>
<td>TWPS0</td>
</tr>
<tr>
<td>Read/Write</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R/W</td>
<td>R/W</td>
<td></td>
</tr>
<tr>
<td>Initial Value</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

- **Bits 7..3 – TWS: TWI Status**
These 5 bits reflect the status of the TWI logic and the 2-wire Serial Bus. The different status codes are described later in this section. Note that the value read from TWSR contains both the 5-bit status value and the 2-bit prescaler value. The application designer should mask the pres-
caler bits to zero when checking the Status bits. This makes status checking independent of prescaler setting. This approach is used in this datasheet, unless otherwise noted.

- **Bit 2 – Res: Reserved Bit**
  This bit is reserved and will always read as zero.

- **Bits 1..0 – TWPS: TWI Prescaler Bits**
  These bits can be read and written, and control the bit rate prescaler.

<table>
<thead>
<tr>
<th>TWPS1</th>
<th>TWPS0</th>
<th>Prescaler Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>16</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>64</td>
</tr>
</tbody>
</table>

To calculate bit rates, see “Bit Rate Generator Unit” on page 229. The value of TWPS1..0 is used in the equation.

### 20.6.4 TWI Data Register – TWDR

<table>
<thead>
<tr>
<th>Bit</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
<th>TWDR</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read/Write</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td></td>
</tr>
<tr>
<td>Initial Value</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

In Transmit mode, TWDR contains the next byte to be transmitted. In Receive mode, the TWDR contains the last byte received. It is writable while the TWI is not in the process of shifting a byte. This occurs when the TWI Interrupt Flag (TWINT) is set by hardware. Note that the Data Register cannot be initialized by the user before the first interrupt occurs. The data in TWDR remains stable as long as TWINT is set. While data is shifted out, data on the bus is simultaneously shifted in. TWDR always contains the last byte present on the bus, except after a wake up from a sleep mode by the TWI interrupt. In this case, the contents of TWDR is undefined. In the case of a lost bus arbitration, no data is lost in the transition from Master to Slave. Handling of the ACK bit is controlled automatically by the TWI logic, the CPU cannot access the ACK bit directly.

- **Bits 7..0 – TWD: TWI Data Register**
  These eight bits constitute the next data byte to be transmitted, or the latest data byte received on the 2-wire Serial Bus.

### 20.6.5 TWI (Slave) Address Register – TWAR

<table>
<thead>
<tr>
<th>Bit</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
<th>TWAR</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read/Write</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td></td>
</tr>
<tr>
<td>Initial Value</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

The TWAR should be loaded with the 7-bit Slave address (in the seven most significant bits of TWAR) to which the TWI will respond when programmed as a Slave Transmitter or Receiver, and not needed in the Master modes. In multi master systems, TWAR must be set in masters which can be addressed as Slaves by other Masters.
The LSB of TWAR is used to enable recognition of the general call address (0x00). There is an associated address comparator that looks for the slave address (or general call address if enabled) in the received serial address. If a match is found, an interrupt request is generated.

- **Bits 7..1 – TWA: TWI (Slave) Address Register**
  These seven bits constitute the slave address of the TWI unit.

- **Bit 0 – TWGCE: TWI General Call Recognition Enable Bit**
  If set, this bit enables the recognition of a General Call given over the 2-wire Serial Bus.

### 20.6.6 TWI (Slave) Address Mask Register – TWAMR

<table>
<thead>
<tr>
<th>Bit</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
<th>TWAMR</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read/Write</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
</tr>
<tr>
<td>Initial Value</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

- **Bits 7..1 – TWAM: TWI Address Mask**
  The TWAMR can be loaded with a 7-bit Slave Address mask. Each of the bits in TWAMR can mask (disable) the corresponding address bit in the TWI Address Register (TWAR). If the mask bit is set to one then the address match logic ignores the compare between the incoming address bit and the corresponding bit in TWAR. Figure 20-10 shows the address match logic in detail.

![Figure 20-10. TWI Address Match Logic, Block Diagram](image)

- **Bit 0 – Res: Reserved Bit**
  This bit is reserved and will always read as zero.

### 20.7 Using the TWI

The AVR TWI is byte-oriented and interrupt based. Interrupts are issued after all bus events, like reception of a byte or transmission of a START condition. Because the TWI is interrupt-based, the application software is free to carry on other operations during a TWI byte transfer. Note that the TWI Interrupt Enable (TWIE) bit in TWCR together with the Global Interrupt Enable bit in SREG allow the application to decide whether or not assertion of the TWINT Flag should generate an interrupt request. If the TWIE bit is cleared, the application must poll the TWINT Flag in order to detect actions on the TWI bus.

When the TWINT Flag is asserted, the TWI has finished an operation and awaits application response. In this case, the TWI Status Register (TWSR) contains a value indicating the current

---

**Note:** The extracted text contains a diagram that is not rendered here. It represents the TWI Address Match Logic, showing the flow of signals and logic gates involved in the address match process. The diagram is crucial for understanding how the TWI addresses are compared and matched during communication.
The state of the TWI bus. The application software can then decide how the TWI should behave in the next TWI bus cycle by manipulating the TWCR and TWDR Registers.

**Figure 20-11** is a simple example of how the application can interface to the TWI hardware. In this example, a Master wishes to transmit a single data byte to a Slave. This description is quite abstract, a more detailed explanation follows later in this section. A simple code example implementing the desired behavior is also presented.

**Figure 20-11. Interfacing the Application to the TWI in a Typical Transmission**

1. The first step in a TWI transmission is to transmit a START condition. This is done by writing a specific value into TWCR, instructing the TWI hardware to transmit a START condition. Which value to write is described later on. However, it is important that the TWINT bit is set in the value written. Writing a one to TWINT clears the flag. The TWI will not start any operation as long as the TWINT bit in TWCR is set. Immediately after the application has cleared TWINT, the TWI will initiate transmission of the START condition.

2. When the START condition has been transmitted, the TWINT Flag in TWCR is set, and TWSR is updated with a status code indicating that the START condition has successfully been sent.

3. The application software should now examine the value of TWSR, to make sure that the START condition was successfully transmitted. If TWSR indicates otherwise, the application software might take some special action, like calling an error routine. Assuming that the status code is as expected, the application must load SLA+W into TWDR. Remember that TWDR is used both for address and data. After TWDR has been loaded with the desired SLA+W, a specific value must be written to TWCR, instructing the TWI hardware to transmit the SLA+W present in TWDR. Which value to write is described later on. However, it is important that the TWINT bit is set in the value written. Writing a one to TWINT clears the flag. The TWI will not start any operation as long as the TWINT bit in TWCR is set. Immediately after the application has cleared TWINT, the TWI will initiate transmission of the address packet.

4. When the address packet has been transmitted, the TWINT Flag in TWCR is set, and TWSR is updated with a status code indicating that the address packet has successfully been sent. The status code will also reflect whether a Slave acknowledged the packet or not.
5. The application software should now examine the value of TWSR, to make sure that the address packet was successfully transmitted, and that the value of the ACK bit was as expected. If TWSR indicates otherwise, the application software might take some special action, like calling an error routine. Assuming that the status code is as expected, the application must load a data packet into TWDR. Subsequently, a specific value must be written to TWCR, instructing the TWI hardware to transmit the data packet present in TWDR. Which value to write is described later on. However, it is important that the TWINT bit is set in the value written. Writing a one to TWINT clears the flag. The TWI will not start any operation as long as the TWINT bit in TWCR is set. Immediately after the application has cleared TWINT, the TWI will initiate transmission of the data packet.

6. When the data packet has been transmitted, the TWINT Flag in TWCR is set, and TWSR is updated with a status code indicating that the data packet has successfully been sent. The status code will also reflect whether a Slave acknowledged the packet or not.

7. The application software should now examine the value of TWSR, to make sure that the data packet was successfully transmitted, and that the value of the ACK bit was as expected. If TWSR indicates otherwise, the application software might take some special action, like calling an error routine. Assuming that the status code is as expected, the application must write a specific value to TWCR, instructing the TWI hardware to transmit a STOP condition. Which value to write is described later on. However, it is important that the TWINT bit is set in the value written. Writing a one to TWINT clears the flag. The TWI will not start any operation as long as the TWINT bit in TWCR is set. Immediately after the application has cleared TWINT, the TWI will initiate transmission of the STOP condition. Note that TWINT is NOT set after a STOP condition has been sent.

Even though this example is simple, it shows the principles involved in all TWI transmissions. These can be summarized as follows:

- When the TWI has finished an operation and expects application response, the TWINT Flag is set. The SCL line is pulled low until TWINT is cleared.
- When the TWINT Flag is set, the user must update all TWI Registers with the value relevant for the next TWI bus cycle. As an example, TWDR must be loaded with the value to be transmitted in the next bus cycle.
- After all TWI Register updates and other pending application software tasks have been completed, TWCR is written. When writing TWCR, the TWINT bit should be set. Writing a one to TWINT clears the flag. The TWI will then commence executing whatever operation was specified by the TWCR setting.

In the following an assembly and C implementation of the example is given. Note that the code below assumes that several definitions have been made, for example by using include-files.
<table>
<thead>
<tr>
<th>Assembly Code Example</th>
<th>C Example</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 ldi r16, (1&lt;&lt;TWINT)</td>
<td>(TWCR = (1&lt;&lt;TWINT))</td>
<td>Send START condition</td>
</tr>
<tr>
<td>(1&lt;&lt;TWSTA) (1&lt;&lt;TWEN)</td>
<td>(1&lt;&lt;TWEN)</td>
<td></td>
</tr>
<tr>
<td>out TWCR, r16</td>
<td></td>
<td></td>
</tr>
<tr>
<td>wait1: in r16,TWCR</td>
<td>while (! (TWCR &amp; (1&lt;&lt;TWINT)))</td>
<td>Wait for TWINT Flag set. This indicates that the START condition has been transmitted</td>
</tr>
<tr>
<td>sbrs r16,TWINT</td>
<td>;</td>
<td></td>
</tr>
<tr>
<td>rjmp wait1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>in r16,TWSR</td>
<td>if ((TWSR &amp; 0xF8) != START)</td>
<td>Check value of TWI Status Register. Mask prescaler bits. If status different from START go to ERROR</td>
</tr>
<tr>
<td>andi r16, 0xF8</td>
<td>ERROR();</td>
<td></td>
</tr>
<tr>
<td>cpi r16, START</td>
<td></td>
<td></td>
</tr>
<tr>
<td>brne ERROR</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3 ldi r16, SLA_W</td>
<td>TWDR = SLA_W;</td>
<td>Load SLA_W into TWDR Register. Clear TWINT bit in TWCR to start transmission of address</td>
</tr>
<tr>
<td>out TWDR, r16</td>
<td>TWCR = (1&lt;&lt;TWINT)</td>
<td>(1&lt;&lt;TWEN);</td>
</tr>
<tr>
<td>ldi r16, (1&lt;&lt;TWINT)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(1&lt;&lt;TWEN)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>out TWCR, r16</td>
<td></td>
<td></td>
</tr>
<tr>
<td>wait2: in r16,TWCR</td>
<td>while (! (TWCR &amp; (1&lt;&lt;TWINT)))</td>
<td>Wait for TWINT Flag set. This indicates that the SLA+W has been transmitted, and ACK/NACK has been received.</td>
</tr>
<tr>
<td>sbrs r16,TWINT</td>
<td>;</td>
<td></td>
</tr>
<tr>
<td>rjmp wait2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>in r16,TWSR</td>
<td>if ((TWSR &amp; 0xF8) != MT_SLA_ACK)</td>
<td>Check value of TWI Status Register. Mask prescaler bits. If status different from MT_SLA_ACK go to ERROR</td>
</tr>
<tr>
<td>andi r16, 0xF8</td>
<td>ERROR();</td>
<td></td>
</tr>
<tr>
<td>cpi r16, MT_SLA_ACK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>brne ERROR</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5 ldi r16, DATA</td>
<td>TWDR = DATA;</td>
<td>Load DATA into TWDR Register. Clear TWINT bit in TWCR to start transmission of data</td>
</tr>
<tr>
<td>out TWDR, r16</td>
<td>TWCR = (1&lt;&lt;TWINT)</td>
<td>(1&lt;&lt;TWEN);</td>
</tr>
<tr>
<td>ldi r16, (1&lt;&lt;TWINT)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(1&lt;&lt;TWEN)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>out TWCR, r16</td>
<td></td>
<td></td>
</tr>
<tr>
<td>wait3: in r16,TWCR</td>
<td>while (! (TWCR &amp; (1&lt;&lt;TWINT)))</td>
<td>Wait for TWINT Flag set. This indicates that the DATA has been transmitted, and ACK/NACK has been received.</td>
</tr>
<tr>
<td>sbrs r16,TWINT</td>
<td>;</td>
<td></td>
</tr>
<tr>
<td>rjmp wait3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>in r16,TWSR</td>
<td>if ((TWSR &amp; 0xF8) != MT_DATA_ACK)</td>
<td>Check value of TWI Status Register. Mask prescaler bits. If status different from MT_DATA_ACK go to ERROR</td>
</tr>
<tr>
<td>andi r16, 0xF8</td>
<td>ERROR();</td>
<td></td>
</tr>
<tr>
<td>cpi r16, MT_DATA_ACK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>brne ERROR</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7 ldi r16, (1&lt;&lt;TWINT)</td>
<td>TWCR = (1&lt;&lt;TWINT)</td>
<td>(1&lt;&lt;TWEN)</td>
</tr>
</tbody>
</table>
20.8 Transmission Modes

The TWI can operate in one of four major modes. These are named Master Transmitter (MT), Master Receiver (MR), Slave Transmitter (ST) and Slave Receiver (SR). Several of these modes can be used in the same application. As an example, the TWI can use MT mode to write data into a TWI EEPROM, MR mode to read the data back from the EEPROM. If other masters are present in the system, some of these might transmit data to the TWI, and then SR mode would be used. It is the application software that decides which modes are legal.

The following sections describe each of these modes. Possible status codes are described along with figures detailing data transmission in each of the modes. These figures contain the following abbreviations:

S: START condition
Rs: REPEATED START condition
R: Read bit (high level at SDA)
W: Write bit (low level at SDA)
A: Acknowledge bit (low level at SDA)
Ā: Not acknowledge bit (high level at SDA)
Data: 8-bit data byte
P: STOP condition
SLA: Slave Address

In Figure 20-13 to Figure 20-19, circles are used to indicate that the TWINT Flag is set. The numbers in the circles show the status code held in TWSR, with the prescaler bits masked to zero. At these points, actions must be taken by the application to continue or complete the TWI transfer. The TWI transfer is suspended until the TWINT Flag is cleared by software.

When the TWINT Flag is set, the status code in TWSR is used to determine the appropriate software action. For each status code, the required software action and details of the following serial transfer are given in Table 20-3 to Table 20-6. Note that the prescaler bits are masked to zero in these tables.

20.8.1 Master Transmitter Mode

In the Master Transmitter mode, a number of data bytes are transmitted to a Slave Receiver (see Figure 20-12). In order to enter a Master mode, a START condition must be transmitted. The format of the following address packet determines whether Master Transmitter or Master Receiver mode is to be entered. If SLA+W is transmitted, MT mode is entered, if SLA+R is transmitted, MR mode is entered. All the status codes mentioned in this section assume that the prescaler bits are zero or are masked to zero.
A START condition is sent by writing the following value to TWCR:

<table>
<thead>
<tr>
<th>TWCR value</th>
<th>TWINT</th>
<th>TWEA</th>
<th>TWSTA</th>
<th>TWSTO</th>
<th>TWWC</th>
<th>TWEN</th>
<th>–</th>
<th>TWIE</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
<td>X</td>
<td>1</td>
<td>0</td>
<td>X</td>
<td>1</td>
<td>0</td>
<td>X</td>
</tr>
</tbody>
</table>

TWEN must be set to enable the 2-wire Serial Interface, TWSTA must be written to one to transmit a START condition and TWINT must be written to one to clear the TWINT Flag. The TWI will then test the 2-wire Serial Bus and generate a START condition as soon as the bus becomes free. After a START condition has been transmitted, the TWINT Flag is set by hardware, and the status code in TWSR will be 0x08 (see Table 20-3). In order to enter MT mode, SLA+W must be transmitted. This is done by writing SLA+W to TWDR. Thereafter the TWINT bit should be cleared (by writing it to one) to continue the transfer. This is accomplished by writing the following value to TWCR:

<table>
<thead>
<tr>
<th>TWCR value</th>
<th>TWINT</th>
<th>TWEA</th>
<th>TWSTA</th>
<th>TWSTO</th>
<th>TWWC</th>
<th>TWEN</th>
<th>–</th>
<th>TWIE</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
<td>X</td>
<td>0</td>
<td>0</td>
<td>X</td>
<td>1</td>
<td>0</td>
<td>X</td>
</tr>
</tbody>
</table>

When SLA+W have been transmitted and an acknowledgement bit has been received, TWINT is set again and a number of status codes in TWSR are possible. Possible status codes in Master mode are 0x18, 0x20, or 0x38. The appropriate action to be taken for each of these status codes is detailed in Table 20-3.

When SLA+W has been successfully transmitted, a data packet should be transmitted. This is done by writing the data byte to TWDR. TWDR must only be written when TWINT is high. If not, the access will be discarded, and the Write Collision bit (TWWC) will be set in the TWCR Register. After updating TWDR, the TWINT bit should be cleared (by writing it to one) to continue the transfer. This is accomplished by writing the following value to TWCR:

<table>
<thead>
<tr>
<th>TWCR value</th>
<th>TWINT</th>
<th>TWEA</th>
<th>TWSTA</th>
<th>TWSTO</th>
<th>TWWC</th>
<th>TWEN</th>
<th>–</th>
<th>TWIE</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
<td>X</td>
<td>0</td>
<td>0</td>
<td>X</td>
<td>1</td>
<td>0</td>
<td>X</td>
</tr>
</tbody>
</table>

This scheme is repeated until the last byte has been sent and the transfer is ended by generating a STOP condition or a repeated START condition. A STOP condition is generated by writing the following value to TWCR:

<table>
<thead>
<tr>
<th>TWCR value</th>
<th>TWINT</th>
<th>TWEA</th>
<th>TWSTA</th>
<th>TWSTO</th>
<th>TWWC</th>
<th>TWEN</th>
<th>–</th>
<th>TWIE</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
<td>X</td>
<td>0</td>
<td>1</td>
<td>X</td>
<td>1</td>
<td>0</td>
<td>X</td>
</tr>
</tbody>
</table>

A REPEATED START condition is generated by writing the following value to TWCR:

<table>
<thead>
<tr>
<th>TWCR value</th>
<th>TWINT</th>
<th>TWEA</th>
<th>TWSTA</th>
<th>TWSTO</th>
<th>TWWC</th>
<th>TWEN</th>
<th>–</th>
<th>TWIE</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
<td>X</td>
<td>1</td>
<td>0</td>
<td>X</td>
<td>1</td>
<td>0</td>
<td>X</td>
</tr>
</tbody>
</table>
After a repeated START condition (state 0x10) the 2-wire Serial Interface can access the same Slave again, or a new Slave without transmitting a STOP condition. Repeated START enables the Master to switch between Slaves, Master Transmitter mode and Master Receiver mode without losing control of the bus.

### Table 20-3. Status codes for Master Transmitter Mode

<table>
<thead>
<tr>
<th>Status Code (TWSR)</th>
<th>Status of the 2-wire Serial Bus and 2-wire Serial Interface Hardware</th>
<th>Application Software Response</th>
<th>To/from TWDR STA</th>
<th>To TWCR STO TWIN TWE</th>
<th>Next Action Taken by TWI Hardware</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x08</td>
<td>A START condition has been transmitted</td>
<td>Load SLA+W</td>
<td>0</td>
<td>0</td>
<td>X</td>
</tr>
<tr>
<td>0x10</td>
<td>A repeated START condition has been transmitted</td>
<td>Load SLA+W or Load SLA+R</td>
<td>0</td>
<td>0</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x18</td>
<td>SLA+W has been transmitted; ACK has been received</td>
<td>Load data byte or</td>
<td>0</td>
<td>0</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td></td>
<td>No TWDR action or</td>
<td>1</td>
<td>0</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td></td>
<td>No TWDR action or</td>
<td>0</td>
<td>1</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td></td>
<td>No TWDR action</td>
<td>1</td>
<td>1</td>
<td>X</td>
</tr>
<tr>
<td>0x20</td>
<td>SLA+W has been transmitted; NOT ACK has been received</td>
<td>Load data byte or</td>
<td>0</td>
<td>0</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td></td>
<td>No TWDR action or</td>
<td>1</td>
<td>0</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td></td>
<td>No TWDR action or</td>
<td>0</td>
<td>1</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td></td>
<td>No TWDR action</td>
<td>1</td>
<td>1</td>
<td>X</td>
</tr>
<tr>
<td>0x28</td>
<td>Data byte has been transmitted; ACK has been received</td>
<td>Load data byte or</td>
<td>0</td>
<td>0</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td></td>
<td>No TWDR action or</td>
<td>1</td>
<td>0</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td></td>
<td>No TWDR action or</td>
<td>0</td>
<td>1</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td></td>
<td>No TWDR action</td>
<td>1</td>
<td>1</td>
<td>X</td>
</tr>
<tr>
<td>0x30</td>
<td>Data byte has been transmitted; NOT ACK has been received</td>
<td>Load data byte or</td>
<td>0</td>
<td>0</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td></td>
<td>No TWDR action or</td>
<td>1</td>
<td>0</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td></td>
<td>No TWDR action or</td>
<td>0</td>
<td>1</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td></td>
<td>No TWDR action</td>
<td>1</td>
<td>1</td>
<td>X</td>
</tr>
<tr>
<td>0x38</td>
<td>Arbitration lost in SLA+W or data bytes</td>
<td>No TWDR action or</td>
<td>0</td>
<td>0</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td></td>
<td>No TWDR action</td>
<td>1</td>
<td>0</td>
<td>X</td>
</tr>
</tbody>
</table>
20.8.2 Master Receiver Mode

In the Master Receiver mode, a number of data bytes are received from a Slave Transmitter (Slave see Figure 20-14). In order to enter a Master mode, a START condition must be transmitted. The format of the following address packet determines whether Master Transmitter or Master Receiver mode is to be entered. If SLA+W is transmitted, MT mode is entered, if SLA+R is transmitted, MR mode is entered. All the status codes mentioned in this section assume that the prescaler bits are zero or are masked to zero.
Figure 20-14. Data Transfer in Master Receiver Mode

A START condition is sent by writing the following value to TWCR:

<table>
<thead>
<tr>
<th>TWCR value</th>
<th>TWINT</th>
<th>TWEA</th>
<th>TWSTA</th>
<th>TWSTO</th>
<th>TWWC</th>
<th>TWEN</th>
<th>–</th>
<th>TWIE</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
<td>X</td>
<td>1</td>
<td>0</td>
<td>X</td>
<td>1</td>
<td>0</td>
<td>X</td>
</tr>
</tbody>
</table>

TWEN must be written to one to enable the 2-wire Serial Interface, TWSTA must be written to one to transmit a START condition and TWINT must be set to clear the TWINT Flag. The TWI will then test the 2-wire Serial Bus and generate a START condition as soon as the bus becomes free. After a START condition has been transmitted, the TWINT Flag is set by hardware, and the status code in TWSR will be 0x08 (See Table 20-3). In order to enter MR mode, SLA+R must be transmitted. This is done by writing SLA+R to TWDR. Thereafter the TWINT bit should be cleared (by writing it to one) to continue the transfer. This is accomplished by writing the following value to TWCR:

<table>
<thead>
<tr>
<th>TWCR value</th>
<th>TWINT</th>
<th>TWEA</th>
<th>TWSTA</th>
<th>TWSTO</th>
<th>TWWC</th>
<th>TWEN</th>
<th>–</th>
<th>TWIE</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
<td>X</td>
<td>0</td>
<td>0</td>
<td>X</td>
<td>1</td>
<td>0</td>
<td>X</td>
</tr>
</tbody>
</table>

When SLA+R have been transmitted and an acknowledgement bit has been received, TWINT is set again and a number of status codes in TWSR are possible. Possible status codes in Master mode are 0x38, 0x40, or 0x48. The appropriate action to be taken for each of these status codes is detailed in Table 20-4. Received data can be read from the TWDR Register when the TWINT Flag is set high by hardware. This scheme is repeated until the last byte has been received. After the last byte has been received, the MR should inform the ST by sending a NACK after the last received data byte. The transfer is ended by generating a STOP condition or a repeated START condition. A STOP condition is generated by writing the following value to TWCR:

<table>
<thead>
<tr>
<th>TWCR value</th>
<th>TWINT</th>
<th>TWEA</th>
<th>TWSTA</th>
<th>TWSTO</th>
<th>TWWC</th>
<th>TWEN</th>
<th>–</th>
<th>TWIE</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
<td>X</td>
<td>0</td>
<td>1</td>
<td>X</td>
<td>1</td>
<td>0</td>
<td>X</td>
</tr>
</tbody>
</table>

A REPEATED START condition is generated by writing the following value to TWCR:

<table>
<thead>
<tr>
<th>TWCR value</th>
<th>TWINT</th>
<th>TWEA</th>
<th>TWSTA</th>
<th>TWSTO</th>
<th>TWWC</th>
<th>TWEN</th>
<th>–</th>
<th>TWIE</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
<td>X</td>
<td>1</td>
<td>0</td>
<td>X</td>
<td>1</td>
<td>0</td>
<td>X</td>
</tr>
</tbody>
</table>

After a repeated START condition (state 0x10) the 2-wire Serial Interface can access the same Slave again, or a new Slave without transmitting a STOP condition. Repeated START enables
the Master to switch between Slaves, Master Transmitter mode and Master Receiver mode without losing control over the bus.

Table 20-4. Status codes for Master Receiver Mode

<table>
<thead>
<tr>
<th>Status Code (TWSR) PreScaler Bits are 0</th>
<th>Status of the 2-wire Serial Bus and 2-wire Serial Interface Hardware</th>
<th>Application Software Response</th>
<th>To/from TWDR</th>
<th>To TWCR</th>
<th>Next Action Taken by TWI Hardware</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x08 A START condition has been transmitted</td>
<td>Load SLA+R</td>
<td>STA 0</td>
<td>STO 0</td>
<td>TWIN 1</td>
<td>TWE X</td>
</tr>
<tr>
<td>0x10 A repeated START condition has been transmitted</td>
<td>Load SLA+R or Load SLA+W</td>
<td>STA 0</td>
<td>STO 0</td>
<td>TWIN 1</td>
<td>TWE X</td>
</tr>
<tr>
<td>0x38 Arbitration lost in SLA+R or NOT ACK bit</td>
<td>No TWDR action or No TWDR action</td>
<td>STA 0</td>
<td>STO 0</td>
<td>TWIN 1</td>
<td>TWE X</td>
</tr>
<tr>
<td>0x40 SLA+R has been transmitted; ACK has been received</td>
<td>No TWDR action or No TWDR action</td>
<td>STA 0</td>
<td>STO 0</td>
<td>TWIN 1</td>
<td>TWE 0</td>
</tr>
<tr>
<td>0x48 SLA+R has been transmitted; NOT ACK has been received</td>
<td>No TWDR action or No TWDR action or No TWDR action</td>
<td>STA 1</td>
<td>STO 0</td>
<td>TWIN 1</td>
<td>TWE X</td>
</tr>
<tr>
<td>0x50 Data byte has been received; ACK has been returned</td>
<td>Read data byte or Read data byte</td>
<td>STA 0</td>
<td>STO 0</td>
<td>TWIN 1</td>
<td>TWE 0</td>
</tr>
<tr>
<td>0x58 Data byte has been received; NOT ACK has been returned</td>
<td>Read data byte or Read data byte or Read data byte</td>
<td>STA 1</td>
<td>STO 1</td>
<td>TWIN 1</td>
<td>TWE X</td>
</tr>
</tbody>
</table>
20.8.3 Slave Receiver Mode

In the Slave Receiver mode, a number of data bytes are received from a Master Transmitter (see Figure 20-16). All the status codes mentioned in this section assume that the prescaler bits are zero or are masked to zero.

Figure 20-16. Data transfer in Slave Receiver mode

To initiate the Slave Receiver mode, TWAR and TWCR must be initialized as follows:
The upper 7 bits are the address to which the 2-wire Serial Interface will respond when addressed by a Master. If the LSB is set, the TWI will respond to the general call address (0x00), otherwise it will ignore the general call address.

<table>
<thead>
<tr>
<th>TWCR value</th>
<th>TWINT</th>
<th>TWEA</th>
<th>TWSTA</th>
<th>TWSTO</th>
<th>TWWC</th>
<th>TWEN</th>
<th>–</th>
<th>TWIE</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>X</td>
</tr>
</tbody>
</table>

TWEN must be written to one to enable the TWI. The TWEA bit must be written to one to enable the acknowledgement of the device’s own slave address or the general call address. TWSTA and TWSTO must be written to zero.

When TWAR and TWCR have been initialized, the TWI waits until it is addressed by its own slave address (or the general call address if enabled) followed by the data direction bit. If the direction bit is “0” (write), the TWI will operate in SR mode, otherwise ST mode is entered. After its own slave address and the write bit have been received, the TWINT Flag is set and a valid status code can be read from TWSR. The status code is used to determine the appropriate software action. The appropriate action to be taken for each status code is detailed in Table 20-5. The Slave Receiver mode may also be entered if arbitration is lost while the TWI is in the Master mode (see states 0x68 and 0x78).

If the TWEA bit is reset during a transfer, the TWI will return a “Not Acknowledge” (“1”) to SDA after the next received data byte. This can be used to indicate that the Slave is not able to receive any more bytes. While TWEA is zero, the TWI does not acknowledge its own slave address. However, the 2-wire Serial Bus is still monitored and address recognition may resume at any time by setting TWEA. This implies that the TWEA bit may be used to temporarily isolate the TWI from the 2-wire Serial Bus.

In all sleep modes other than Idle mode, the clock system to the TWI is turned off. If the TWEA bit is set, the interface can still acknowledge its own slave address or the general call address by using the 2-wire Serial Bus clock as a clock source. The part will then wake up from sleep and the TWI will hold the SCL clock low during the wake up and until the TWINT Flag is cleared (by writing it to one). Further data reception will be carried out as normal, with the AVR clocks running as normal. Observe that if the AVR is set up with a long start-up time, the SCL line may be held low for a long time, blocking other data transmissions.

Note that the 2-wire Serial Interface Data Register – TWDR does not reflect the last byte present on the bus when waking up from these Sleep modes.
<table>
<thead>
<tr>
<th>Status Code (TWSR)</th>
<th>Status of the 2-wire Serial Bus and 2-wire Serial Interface Hardware</th>
<th>Application Software Response</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>No TWDR action to/from TWDR</td>
<td>STA</td>
</tr>
<tr>
<td>0x60</td>
<td>Own SLA+W has been received; ACK has been returned</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>No TWDR action</td>
<td>X</td>
</tr>
<tr>
<td>0x68</td>
<td>Arbitration lost in SLA+R/W as Master; own SLA+W has been received; ACK has been returned</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>No TWDR action</td>
<td>X</td>
</tr>
<tr>
<td>0x70</td>
<td>General call address has been received; ACK has been returned</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>No TWDR action</td>
<td>X</td>
</tr>
<tr>
<td>0x78</td>
<td>Arbitration lost in SLA+R/W as Master; General call address x has been received; ACK has been returned</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>No TWDR action</td>
<td>X</td>
</tr>
<tr>
<td>0x80</td>
<td>Previously addressed with own SLA+W; data has been received; ACK has been returned</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>0x88</td>
<td>Previously addressed with own SLA+W; data has been received; NOT ACK has been returned</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>Read data byte or</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Read data byte or</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>0x90</td>
<td>Previously addressed with general call; data has been received; ACK has been returned</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>0x98</td>
<td>Previously addressed with general call; data has been received; NOT ACK has been returned</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>Read data byte or</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Read data byte or</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>0xA0</td>
<td>A STOP condition or repeated START condition has been received while still addressed as Slave</td>
<td>No action</td>
</tr>
<tr>
<td></td>
<td></td>
<td>No action</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
</tr>
</tbody>
</table>
20.8.4 Slave Transmitter Mode

In the Slave Transmitter mode, a number of data bytes are transmitted to a Master Receiver (see Figure 20-18). All the status codes mentioned in this section assume that the prescaler bits are zero or are masked to zero.

Figure 20-18. Data Transfer in Slave Transmitter Mode
To initiate the Slave Transmitter mode, TWAR and TWCR must be initialized as follows:

<table>
<thead>
<tr>
<th>TWAR value</th>
<th>TWA6</th>
<th>TWA5</th>
<th>TWA4</th>
<th>TWA3</th>
<th>TWA2</th>
<th>TWA1</th>
<th>TWA0</th>
<th>TWGCE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Device’s Own Slave Address</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The upper seven bits are the address to which the 2-wire Serial Interface will respond when addressed by a Master. If the LSB is set, the TWI will respond to the general call address (0x00), otherwise it will ignore the general call address.

<table>
<thead>
<tr>
<th>TWCR value</th>
<th>TWINT</th>
<th>TWEA</th>
<th>TWSTA</th>
<th>TWSTO</th>
<th>TWWC</th>
<th>TWEN</th>
<th>–</th>
<th>TWIE</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>–</td>
<td>X</td>
</tr>
</tbody>
</table>

TWEN must be written to one to enable the TWI. The TWEA bit must be written to one to enable the acknowledgement of the device’s own slave address or the general call address. TWSTA and TWSTO must be written to zero.

When TWAR and TWCR have been initialized, the TWI waits until it is addressed by its own slave address (or the general call address if enabled) followed by the data direction bit. If the direction bit is “1” (read), the TWI will operate in ST mode, otherwise SR mode is entered. After its own slave address and the write bit have been received, the TWINT Flag is set and a valid status code can be read from TWSR. The status code is used to determine the appropriate software action. The appropriate action to be taken for each status code is detailed in Table 20-6. The Slave Transmitter mode may also be entered if arbitration is lost while the TWI is in the Master mode (see state 0xB0).

If the TWEA bit is written to zero during a transfer, the TWI will transmit the last byte of the transfer. State 0xC0 or state 0xC8 will be entered, depending on whether the Master Receiver transmits a NACK or ACK after the final byte. The TWI is switched to the not addressed Slave mode, and will ignore the Master if it continues the transfer. Thus the Master Receiver receives all “1” as serial data. State 0xC8 is entered if the Master demands additional data bytes (by transmitting ACK), even though the Slave has transmitted the last byte (TWEA zero and expecting NACK from the Master).

While TWEA is zero, the TWI does not respond to its own slave address. However, the 2-wire Serial Bus is still monitored and address recognition may resume at any time by setting TWEA. This implies that the TWEA bit may be used to temporarily isolate the TWI from the 2-wire Serial Bus.

In all sleep modes other than Idle mode, the clock system to the TWI is turned off. If the TWEA bit is set, the interface can still acknowledge its own slave address or the general call address by using the 2-wire Serial Bus clock as a clock source. The part will then wake up from sleep and the TWI will hold the SCL clock will low during the wake up and until the TWINT Flag is cleared (by writing it to one). Further data transmission will be carried out as normal, with the AVR clocks running as normal. Observe that if the AVR is set up with a long start-up time, the SCL line may be held low for a long time, blocking other data transmissions.

Note that the 2-wire Serial Interface Data Register – TWDR does not reflect the last byte present on the bus when waking up from these sleep modes.
## Table 20-6. Status Codes for Slave Transmitter Mode

<table>
<thead>
<tr>
<th>Status Code (TWSR) Prescaler Bits are 0</th>
<th>Status of the 2-wire Serial Bus and 2-wire Serial Interface Hardware</th>
<th>Application Software Response</th>
<th>To/from TWDR</th>
<th>To TWCR</th>
<th>Next Action Taken by TWI Hardware</th>
</tr>
</thead>
<tbody>
<tr>
<td>0xA8 Own SLA+R has been received; ACK has been returned</td>
<td>Load data byte</td>
<td>X</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Load data byte</td>
<td>X</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0xB0 Arbitration lost in SLA+R/W as Master; own SLA+R has been received; ACK has been returned</td>
<td>Load data byte</td>
<td>X</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Load data byte</td>
<td>X</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0xBB Data byte in TWDR has been transmitted; ACK has been received</td>
<td>Load data byte</td>
<td>X</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Load data byte</td>
<td>X</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0xC0 Data byte in TWDR has been transmitted; NOT ACK has been received</td>
<td>No TWDR action or</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>No TWDR action or</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>No TWDR action or</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>No TWDR action</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0xC8 Last data byte in TWDR has been transmitted (TWEA = ‘0’); ACK has been received</td>
<td>No TWDR action or</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>No TWDR action or</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>No TWDR action or</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>No TWDR action</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
Figure 20.19. Formats and States in the Slave Transmitter Mode

20.8.5 Miscellaneous States

There are two status codes that do not correspond to a defined TWI state, see Table 20.7.

Status 0xF8 indicates that no relevant information is available because the TWINT Flag is not set. This occurs between other states, and when the TWI is not involved in a serial transfer.

Status 0x00 indicates that a bus error has occurred during a 2-wire Serial Bus transfer. A bus error occurs when a START or STOP condition occurs at an illegal position in the format frame. Examples of such illegal positions are during the serial transfer of an address byte, a data byte, or an acknowledge bit. When a bus error occurs, TWINT is set. To recover from a bus error, the TWSTO Flag must set and TWINT must be cleared by writing a logic one to it. This causes the TWI to enter the not addressed Slave mode and to clear the TWSTO Flag (no other bits in TWCR are affected). The SDA and SCL lines are released, and no STOP condition is transmitted.

Table 20.7. Miscellaneous States

<table>
<thead>
<tr>
<th>Status Code (TWSR) Prescaler Bits are 0</th>
<th>Status of the 2-wire Serial Bus and 2-wire Serial Interface Hardware</th>
<th>Application Software Response</th>
<th>To/from TWDR</th>
<th>To TWCR</th>
<th>Next Action Taken by TWI Hardware</th>
</tr>
</thead>
<tbody>
<tr>
<td>0xF8</td>
<td>No relevant state information available; TWINT = &quot;0&quot;</td>
<td>No TWDR action</td>
<td>No TDWR action</td>
<td>STA</td>
<td>No TWCR action</td>
</tr>
<tr>
<td>0x00</td>
<td>Bus error due to an illegal START or STOP condition</td>
<td>No TWDR action</td>
<td>No TDWR action</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

20.8.6 Combining Several TWI Modes

In some cases, several TWI modes must be combined in order to complete the desired action. Consider for example reading data from a serial EEPROM. Typically, such a transfer involves the following steps:

1. The transfer must be initiated.
2. The EEPROM must be instructed what location should be read.
3. The reading must be performed.
4. The transfer must be finished.
Note that data is transmitted both from Master to Slave and vice versa. The Master must instruct the Slave what location it wants to read, requiring the use of the MT mode. Subsequently, data must be read from the Slave, implying the use of the MR mode. Thus, the transfer direction must be changed. The Master must keep control of the bus during all these steps, and the steps should be carried out as an atomical operation. If this principle is violated in a multi master system, another Master can alter the data pointer in the EEPROM between steps 2 and 3, and the Master will read the wrong data location. Such a change in transfer direction is accomplished by transmitting a REPEATED START between the transmission of the address byte and reception of the data. After a REPEATED START, the Master keeps ownership of the bus. The following figure shows the flow in this transfer.

**Figure 20-20.** Combining Several TWI Modes to Access a Serial EEPROM

<table>
<thead>
<tr>
<th>S</th>
<th>SLA+W</th>
<th>A</th>
<th>ADDRESS</th>
<th>A</th>
<th>Rs</th>
<th>SLA+R</th>
<th>A</th>
<th>DATA</th>
<th>A</th>
<th>P</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>Transmitted from master to slave</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Transmitted from slave to master</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### 20.9 Multi-master Systems and Arbitration

If multiple masters are connected to the same bus, transmissions may be initiated simultaneously by one or more of them. The TWI standard ensures that such situations are handled in such a way that one of the masters will be allowed to proceed with the transfer, and that no data will be lost in the process. An example of an arbitration situation is depicted below, where two masters are trying to transmit data to a Slave Receiver.

**Figure 20-21.** An Arbitration Example

Several different scenarios may arise during arbitration, as described below:

- Two or more masters are performing identical communication with the same Slave. In this case, neither the Slave nor any of the masters will know about the bus contention.
- Two or more masters are accessing the same Slave with different data or direction bit. In this case, arbitration will occur, either in the READ/WRITE bit or in the data bits. The masters trying to output a one on SDA while another Master outputs a zero will lose the arbitration. Losing masters will switch to not addressed Slave mode or wait until the bus is free and transmit a new START condition, depending on application software action.
- Two or more masters are accessing different slaves. In this case, arbitration will occur in the SLA bits. Masters trying to output a one on SDA while another Master outputs a zero will lose the arbitration. Masters losing arbitration in SLA will switch to Slave mode to check if they are being addressed by the winning Master. If addressed, they will switch to SR or ST mode, depending on the value of the READ/WRITE bit. If they are not being addressed, they will switch to not addressed Slave mode or wait until the bus is free and transmit a new START condition, depending on application software action.

This is summarized in Figure 20-22. Possible status values are given in circles.

Figure 20-22. Possible Status Codes Caused by Arbitration
21. USB controller

21.1 Features

- Supports full-speed and low-speed Device role
- Complies with USB Specification v2.0
- Supports ping-pong mode (dual bank)
- 832 bytes of DPRAM:
  - 1 endpoint 64 bytes max (default control endpoint)
  - 1 endpoints of 256 bytes max, (one or two banks)
  - 5 endpoints of 64 bytes max, (one or two banks)
- Crystal-less operation for low-speed mode

21.2 Block Diagram

The USB controller provides the hardware to interface a USB link to a data flow stored in a double port memory (DPRAM).

The USB controller requires a 48 MHz ±0.25% reference clock (for Full-Speed operation), which is the output of an internal PLL. The on-chip PLL generates the internal high frequency (48 MHz) clock for USB interface. The PLL clock input can be configured to use external low-power crystal oscillator, external source clock or internal RC (see Section “Crystal-less operation”, page 256).

The 48MHz clock is used to generate a 12 MHz Full-speed (or 1.5 MHz Low-Speed) bit clock from the received USB differential data and to transmit data according to full or low speed USB device tolerance. Clock recovery is done by a Digital Phase Locked Loop (DPLL) block, which is compliant with the jitter specification of the USB bus.

To comply with the USB Electrical specification, USB buffers (D+ or D-) should be powered within the 3.0 to 3.6V range. As ATmega16U4/ATmega32U4 can be powered up to 5.5V, an internal regulator provides the USB buffers power supply.

Figure 21-1. USB controller Block Diagram overview
21.3 Typical Application Implementation

Depending on the target application power supply, the ATmega16U4/ATmega32U4 requires different hardware typical implementations.

**Figure 21-2.** Operating modes versus frequency and power-supply

21.3.1 Bus Powered device

**Figure 21-3.** Typical Bus powered application with 5V I/O
21.3.2 Self Powered device

Figure 21-4. Typical Bus powered application with 3V I/O

Figure 21-5. Typical Self powered application with 3.4V to 5.5V I/O
21.4 Crystal-less operation

To reduce external components count and BOM cost, the USB module can be configured to operate in low-speed mode with internal RC oscillator as input source clock for the PLL. The internal RC oscillator is factory calibrated to satisfy the USB low speed frequency accuracy within the 0°C and -40°C temperature range.

For USB full-speed operation only external crystal oscillator or external source clock can be used.

21.5 Design guidelines

- Serial resistors on USB Data lines must have 22 Ohms value (+/- 5%).
- Traces from the input USB receptable (or from the cable connection in the case of a tethered device) to the USB microcontroller pads should be as short as possible, and follow differential traces routing rules (same length, as near as possible, avoid via accumulation).
- Voltage transient / ESD suppressors may also be used to prevent USB pads to be damaged by external disturbances.
- $U_{cap}$ capacitor should be 1μF (+/- 10%) for correct operation.
- A 10μF capacitor is highly recommended on VBUS line
21.6 General Operating Modes

21.6.1 Introduction

The USB controller is disabled and reset after an hardware reset generated by:

- Power on reset
- External reset
- Watchdog reset
- Brown out reset
- JTAG reset

But another available and optional CPU reset source is:

- USB End Of Reset

In this case, the USB controller is reset, but not disabled (so that the device remains attached).

21.6.2 Power-on and reset

The next diagram explains the USB controller main states on power-on:

**Figure 21-7. USB controller states after reset**

![USB controller state diagram]

USB Controller state after an hardware reset is ‘Reset’. In this state:

- USBE is not set
- the USB controller clock is stopped in order to minimize the power consumption (FRZCLK=1),
- the USB controller is disabled,
- the USB pad is in the suspend mode,
- the Device USB controller internal state is reset.

After setting USBE, the USB Controller enters the Device state. The controller is 'Idle'.

The USB Controller can at any time be stopped by clearing USBE. In fact, clearing USBE acts as an hardware reset.

21.6.3 Interrupts

Two interrupts vectors are assigned to USB interface.
Almost all these interrupts are time-relative events that will be detected only if the USB clock is enabled (FRZCLK bit set), except for:

- **VBUS** plug-in detection (insert, remove)
- **WAKEUP** interrupt that will trigger each time a state change is detected on the data lines.

This asynchronous interrupts allow to wake-up a device that is in power-down mode, generally after that the USB has entered the Suspend state.
Each endpoint has 8 interrupts sources associated with flags, and each source can be enabled or not to trigger the corresponding endpoint interrupt. If, for an endpoint, at least one of the sources is enabled to trigger interrupt, the corresponding event(s) will make the program branch to the USB Endpoint Interrupt vector. The user may determine the source (endpoint) of the interrupt by reading the UEINT register, and then handle the event detected by polling the different flags.

21.7 Power modes

21.7.1 Idle mode

In this mode, the CPU core is halted (CPU clock stopped). The Idle mode is taken wether the USB controller is running or not. The CPU “wakes up” on any USB interrupts.

21.7.2 Power down

In this mode, the oscillator is stopped and halts all the clocks (CPU and peripherals). The USB controller “wakes up” when:

- the WAKEUPI interrupt is triggered
- the VBUStI interrupt is triggered
21.7.3 Freeze clock

The firmware has the ability to reduce the power consumption by setting the FRZCLK bit, which freeze the clock of USB controller. When FRZCLK is set, it is still possible to access to the following registers:

- USBCON, USBSTA, USBINT
- UDCON (detach, ...)
- UDINT
- UDIEN

Moreover, when FRZCLK is set, only the following interrupts may be triggered:

- WAKEUPI
- VBUSITI

21.8 Speed Control

The speed selection (Full Speed or Low Speed) depends on the D+/D- pull-up. The LSM bit in UDCON register allows to select an internal pull up on D- (Low Speed mode) or D+ (Full Speed mode) data lines.

**Figure 21-11. Device mode Speed Selection**

21.9 Memory management

The controller only supports the following memory allocation management.

The reservation of a Pipe or an Endpoint can only be made in the increasing order (Pipe/Endpoint 0 to the last Pipe/Endpoint). The firmware shall thus configure them in the same order.

The reservation of a Pipe or an Endpoint “k" is done when its ALLOC bit is set. Then, the hardware allocates the memory and inserts it between the Pipe/Endpoints “k-1” and “k+1”. The “k+1” Pipe/Endpoint memory “slides” up and its data is lost. Note that the “k+2" and upper Pipe/Endpoint memory does not slide.

Clearing a Pipe enable (PEN) or an Endpoint enable (EPEN) does not clear either its ALLOC bit, or its configuration (EP/PSIZE/PSIZE, EPBK/PBK). To free its memory, the firmware should clear ALLOC. Then, the “k+1” Pipe/Endpoint memory automatically “slides” down. Note that the “k+2" and upper Pipe/Endpoint memory does not slide.
The following figure illustrates the allocation and reorganization of the USB memory in a typical example:

**Table 21.1. Allocation and reorganization USB memory flow**

<table>
<thead>
<tr>
<th>EPEN=1</th>
<th>ALLOC=1</th>
<th>EPEN=0</th>
<th>ALLOC=0</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>EPEND</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>EPEND</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>EPEND</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>EPEND</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- First, Endpoint 0 to Endpoint 5 are configured, in the growing order. The memory of each is reserved in the DPRAM.
- Then, the Endpoint 3 is disabled (EPEN=0), but its memory reservation is internally kept by the controller.
- Its ALLOC bit is cleared: the Endpoint 4 “slides” down, but the Endpoint 5 does not “slide”.
- Finally, if the firmware chooses to reconfigure the Endpoint 3, with a bigger size. The controller reserved the memory after the Endpoint 2 memory and automatically “slide” the Endpoint 4. The Endpoint 5 does not move and a memory conflict appear, in that both Endpoint 4 and 5 use a common area. The data of those endpoints are potentially lost.

Note that:

- the data of Endpoint 0 are never lost whatever the activation or deactivation of the higher Endpoint. Its data is lost if it is deactivated.
- Deactivate and reactivate the same Endpoint with the same parameters does not lead to a “slide” of the higher endpoints. For those endpoints, the data are preserved.
- CFGOK is set by hardware even in the case where there is a “conflict” in the memory allocation.

### 21.10 PAD suspend

The next figures illustrates the pad behaviour:

- In the “idle” mode, the pad is put in low power consumption mode.
- In the “active” mode, the pad is working.
The SUSPI flag indicated that a suspend state has been detected on the USB bus. This flag automatically put the USB pad in Idle. The detection of a non-idle event sets the WAKEUPI flag and wakes-up the USB pad.

Moreover, the pad can also be put in the “idle” mode if the DETACH bit is set. It come back in the active mode when the DETACH bit is cleared.

21.11 Plug-in detection

The USB connection is detected by the VBUS pad, thanks to the following architecture:
Figure 21-13. Plug-in Detection Input Block Diagram

The control logic of the VBUS pad outputs a signal regarding the VBUS voltage level:

- The “Session_valid” signal is active high when the voltage on the VBUS pad is higher or equal to 1.4V. If lower than 1.4V, the signal is not active.
- The VBUS status bit is set when “Session_valid” signal is active (VBUS > 1.4V).
- The VBUS flag is set each time the VBUS state changes.
- The USB peripheral cannot attach to the bus while VBUS bit is not set.

21.12 Registers description

21.12.1 USB general registers

<table>
<thead>
<tr>
<th>Bit</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read/Write</td>
<td>R/W</td>
<td>R/W</td>
<td>R</td>
<td>R/W</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R/W</td>
</tr>
<tr>
<td>Initial Value</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

- **7-1 – Reserved**
  These bits are reserved. Do not modify these bits.

- **0 – UVREGE: USB pad regulator Enable**
  Set to enable the USB pad regulator. Clear to disable the USB pad regulator.

<table>
<thead>
<tr>
<th>Bit</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read/Write</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R</td>
<td>R</td>
<td>R/W</td>
<td>R/W</td>
</tr>
<tr>
<td>Initial Value</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

- **7 – USBE: USB macro Enable Bit**
  Set to enable the USB controller. Clear to disable and reset the USB controller, to disable the USB transceiver and to disable the USB controller clock inputs.

- **6 – Reserved**
  The value read from these bits is always 0. Do not set these bits.
• **5 – FRZCLK: Freeze USB Clock Bit**
  Set to disable the clock inputs (the "Resume Detection" is still active). This reduces the power consumption. Clear to enable the clock inputs.

• **4 – OTGPADE: VBUS Pad Enable**
  Set to enable the VBUS pad. Clear to disable the VBUS pad.
  Note that this bit can be set/cleared even if USBE=0. That allows the VBUS detection even if the USB macro is disable.

• **3-1 – Reserved**
  The value read from these bits is always 0. Do not set these bits.

• **0 – VBSTE: VBUS Transition Interrupt Enable Bit**
  Set this bit to enable the VBUS Transition interrupt generation.
  Clear this bit to disable the VBUS Transition interrupt generation.

<table>
<thead>
<tr>
<th>Bit</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>USBSTA</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>ID</td>
<td>VBUS</td>
</tr>
<tr>
<td>Read/Write</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
</tr>
<tr>
<td>Initial Value</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

• **7-2 - Reserved**
  The value read from these bits is always 0. Do not set these bits.

• **1 - ID: ID status**
  This bit is always read as “1”, it has been conserved for compatibility with AT90USB64/128 (in which it indicates the value of the OTG ID pin).

• **0 – VBUS: VBus Flag**
  The value read from this bit indicates the state of the VBUS pin. This bit can be used in device mode to monitor the USB bus connection state of the application. See Section 21.11, page 262 for more details.

<table>
<thead>
<tr>
<th>Bit</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>USBINT</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>VBUSTI</td>
<td></td>
</tr>
<tr>
<td>Read/Write</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R/W</td>
<td>R/W</td>
</tr>
<tr>
<td>Initial Value</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

7-1 - Reserved
The value read from these bits is always 0. Do not set these bits.

• **0 – VBUSTI: IVBUS Transition Interrupt Flag**
  Set by hardware when a transition (high to low, low to high) has been detected on the VBUS pad.
  Shall be cleared by software.
21.13 USB Software Operating modes

Depending on the USB operating mode, the software should perform some of the following operations:

**Power On the USB interface**
- Power-On USB pads regulator
- Configure PLL interface
- Enable PLL
- Check PLL lock
- Enable USB interface
- Configure USB interface (USB speed, Endpoints configuration...)
- Wait for USB VBUS information connection
- Attach USB device

**Power Off the USB interface**
- Detach USB interface
- Disable USB interface
- Disable PLL
- Disable USB pad regulator

**Suspending the USB interface**
- Clear Suspend Bit
- Freeze USB clock
- Disable PLL
- Be sure to have interrupts enabled to exit sleep mode
- Make the MCU enter sleep mode

**Resuming the USB interface**
- Enable PLL
- Wait PLL lock
- Unfreeze USB clock
- Clear Resume information
22. USB Device Operating modes

22.1 Introduction

The USB device controller supports full speed and low speed data transfers. In addition to the default control endpoint, it provides six other endpoints, which can be configured in control, bulk, interrupt or isochronous modes:

- Endpoint 0: programmable size FIFO up to 64 bytes, default control endpoint
- Endpoints 1 programmable size FIFO up to 256 bytes in ping-pong mode.
- Endpoints 2 to 6: programmable size FIFO up to 64 bytes in ping-pong mode.

The controller starts in the “idle” mode. In this mode, the pad consumption is reduced to the minimum.

22.2 Power-on and reset

The next diagram explains the USB device controller main states on power-on:

Figure 22-1. USB device controller states after reset

![USB device controller states](image)

The reset state of the Device controller is:

- the macro clock is stopped in order to minimize the power consumption (FRZCLK set),
- the USB device controller internal state is reset (all the registers are reset to their default value. Note that DETACH is set.)
- the endpoint banks are reset
- the D+ or D- pull up are not activated (mode Detach)

The D+ or D- pull-up will be activated as soon as the DETACH bit is cleared and VBUS is present.

The macro is in the ‘Idle’ state after reset with a minimum power consumption and does not need to have the PLL activated to enter this state.

The USB device controller can at any time be reset by clearing USBE (disable USB interface).

22.3 Endpoint reset

An endpoint can be reset at any time by setting in the UERST register the bit corresponding to the endpoint (EPRSTx). This resets:

- the internal state machine on that endpoint,
- the Rx and Tx banks are cleared and their internal pointers are restored,
• the UEINTX, UESTA0X and UESTA1X are restored to their reset value.
The data toggle field remains unchanged.
The other registers remain unchanged.
The endpoint configuration remains active and the endpoint is still enabled.
The endpoint reset may be associated with a clear of the data toggle command (RSTDT bit) as
an answer to the CLEAR_FEATURE USB command.

22.4 USB reset
When an USB reset is detected on the USB line (SE0 state with a minimum duration of 2.5μs),
the next operations are performed by the controller:
• all the endpoints are disabled
• the default control endpoint remains configured (see Section 22.3, page 266 for more
details).
If the CPU hardware reset function is activated (RSTCPU bit set in UDCON register), a reset is
generated to the CPU core without disabling the USB controller (that follows the same behavior
than after a standard USB End of Reset, and remains attached). That feature may be used to
enhance device reliability.

22.5 Endpoint selection
Prior to any operation performed by the CPU, the endpoint must first be selected. This is done
by setting the EPNUM2:0 bits (UENUM register) with the endpoint number which will be man-
aged by the CPU.
The CPU can then access to the various endpoint registers and data.

22.6 Endpoint activation
The endpoint is maintained under reset as long as the EPEN bit is not set.
The following flow must be respected in order to activate an endpoint:
Figure 22-2. Endpoint activation flow:

As long as the endpoint is not correctly configured (CFGOK cleared), the hardware does not acknowledge the packets sent by the host.

CFGOK is will not be sent if the Endpoint size parameter is bigger than the DPRAM size.
A clear of EPEN acts as an endpoint reset (see Section 22.3, page 266 for more details). It also performs the next operation:

- The configuration of the endpoint is kept (EPSIZE, EPBK, ALLOC kept)
- It resets the data toggle field.
- The DPRAM memory associated to the endpoint is still reserved.

See Section 21.9, page 260 for more details about the memory allocation/reorganization.

22.7 Address Setup

The USB device address is set up according to the USB protocol:

- the USB device, after power-up, responds at address 0
- the host sends a SETUP command (SET_ADDRESS(addr)),
- the firmware handles this request, and records that address in UADD, but keep ADDEN cleared,
- the USB device firmware sends an IN command of 0 bytes (IN 0 Zero Length Packet),
- then, the firmware can enable the USB device address by setting ADDEN. The only accepted address by the controller is the one stored in UADD.

ADDEN and UADD shall not be written at the same time.

UADD contains the default address 00h after a power-up or USB reset.
ADDEN is cleared by hardware:

- after a power-up reset,
- when an USB reset is received,
- or when the macro is disabled (USBE cleared)

When this bit is cleared, the default device address 00h is used.

### 22.8 Suspend, Wake-up and Resume

After a period of 3 ms during which the USB line was inactive, the controller switches to the full-speed mode and triggers (if enabled) the SUSPI (suspend) interrupt. The firmware may then set the FRZCLK bit.

The CPU can also, depending on software architecture, enter in the idle mode to lower again the power consumption.

There are two ways to recover from the “Suspend” mode:

- First one is to clear the FRZCLK bit. This is possible if the CPU is not in the Idle mode.
- Second way, if the CPU is “idle”, is to enable the WAKEUPI interrupt (WAKEUPE set). Then, as soon as an non-idle signal is seen by the controller, the WAKEUPI interrupt is triggered.
  The firmware shall then clear the FRZCLK bit to restart the transfer.

There are no relationship between the SUSPI interrupt and the WAKEUPI interrupt: the WAKEUPI interrupt is triggered as soon as there are non-idle patterns on the data lines. Thus, the WAKEUPI interrupt can occurs even if the controller is not in the “suspend” mode.

When the WAKEUPI interrupt is triggered, if the SUSPI interrupt bit was already set, it is cleared by hardware.

When the SUSPI interrupt is triggered, if the WAKEUPI interrupt bit was already set, it is cleared by hardware.

### 22.9 Detach

The reset value of the DETACH bit is 1.

It is possible to re.enumerate a device, simply by setting and clearing the DETACH bit (but firmware must take in account a debouncing delay of some milliseconds).

- Setting DETACH will disconnect the pull-up on the D+ or D- pad (depending on full or low speed mode selected). Then, clearing DETACH will connect the pull-up on the D+ or D- pad.

**Figure 22-3.** Detach a device in Full-speed:
22.10 Remote Wake-up

The “Remote Wake-up” (or “upstream resume”) feature is the only operation allowed to be sent by the device on its own initiative. Anyway, to do that, the device should first have received a DEVICE_REMOTE_WAKEUP request from the host.

- First, the USB controller must have detected the “suspend” state of the line: the remote wake-up can only be sent when a SUSPI flag is set.
- The firmware has then the ability to set RMWKUP to send the “upstream resume” stream. This will automatically be done by the controller after 5ms of inactivity on the USB line.
- When the controller starts to send the “upstream resume”, the UPRSMI interrupt is triggered (if enabled).
- RMWKUP is cleared by hardware at the end of the “upstream resume”.
- If the controller detects a good “End Of Resume” signal from the host, an EORSMI interrupt is triggered (if enabled).

22.11 STALL request

For each endpoint, the STALL management is performed using 2 bits:

- STALLRQ (enable stall request)
- STALLRQC (disable stall request)
- STALLEDI (stall sent interrupt)

To send a STALL handshake at the next request, the STALLRQ request bit has to be set. All following requests will be handshake'd with a STALL until the STALLRQC bit is set.

Setting STALLRQC automatically clears the STALLRQ bit. The STALLRQC bit is also immediately cleared by hardware after being set by software. Thus, the firmware will never read this bit as set.

Each time the STALL handshake is sent, the STALLEDI flag is set by the USB controller and the EPINTx interrupt will be triggered (if enabled).

The incoming packets will be discarded (RXOUTI and RWAL will not be set).

The host will then send a command to reset the STALL: the firmware just has to set the STALLRQC bit and to reset the endpoint.

22.11.1 Special consideration for Control Endpoints

A SETUP request is always ACK’ed.

If a STALL request is set for a Control Endpoint and if a SETUP request occurs, the SETUP request has to be ACK’ed and the STALLRQ request and STALLEDI sent flags are automatically reset (RXSETUPI set, TXIN cleared, STALLED cleared, TXINI cleared...).

This management simplifies the enumeration process management. If a command is not supported or contains an error, the firmware set the STALL request flag and can return to the main task, waiting for the next SETUP request.

This function is compliant with the Chapter 8 test that may send extra status for a GET_DESCRIPTOR. The firmware sets the STALL request just after receiving the status. All extra status will be automatically STALL’ed until the next SETUP request.
22.11.2 STALL handshake and Retry mechanism

The Retry mechanism has priority over the STALL handshake. A STALL handshake is sent if the STALLRQ request bit is set and if there is no retry required.

22.12 CONTROL endpoint management

A SETUP request is always ACK’ed. When a new setup packet is received, the RXSTPI interrupt is triggered (if enabled). The RXOUTI interrupt is not triggered.

The FIFOCON and RWAL fields are irrelevant with CONTROL endpoints. The firmware shall thus never use them on that endpoints. When read, their value is always 0.

CONTROL endpoints are managed by the following bits:

- RXSTPI is set when a new SETUP is received. It shall be cleared by firmware to acknowledge the packet and to clear the endpoint bank.
- RXOUTI is set when a new OUT data is received. It shall be cleared by firmware to acknowledge the packet and to clear the endpoint bank.
- TXINI is set when the bank is ready to accept a new IN packet. It shall be cleared by firmware to send the packet and to clear the endpoint bank.

22.12.1 Control Write

The next figure shows a control write transaction. During the status stage, the controller will not necessary send a NAK at the first IN token:

- If the firmware knows the exact number of descriptor bytes that must be read, it can then anticipate on the status stage and send a ZLP for the next IN token,
- or it can read the bytes and poll NAKINI, which tells that all the bytes have been sent by the host, and the transaction is now in the status stage.
22.12.2 Control Read

The next figure shows a control read transaction. The USB controller has to manage the simultaneous write requests from the CPU and the USB host:

A NAK handshake is always generated at the first status stage command.

When the controller detect the status stage, all the data written by the CPU are erased, and clearing TXINI has no effects.

The firmware checks if the transmission is complete or if the reception is complete.

The OUT retry is always ack’ed. This reception:
- set the RXOUTI flag (received OUT data)
- set the TXINI flag (data sent, ready to accept new data)

software algorithm:

```
set transmit ready
wait (transmit complete OR Receive complete)
if receive complete, clear flag and return
if transmit complete, continue
```

Once the OUT status stage has been received, the USB controller waits for a SETUP request. The SETUP request have priority over any other request and has to be ACK’ed. This means that any other flag should be cleared and the fifo reset when a SETUP is received.

WARNING: the byte counter is reset when the OUT Zero Length Packet is received. The firmware has to take care of this.

22.13 OUT endpoint management

OUT packets are sent by the host. All the data can be read by the CPU, which acknowledges or not the bank when it is empty.

22.13.1 Overview

The Endpoint must be configured first.

Each time the current bank is full, the RXOUTI and the FIFOCON bits are set. This triggers an interrupt if the RXOUTE bit is set. The firmware can acknowledge the USB interrupt by clearing the RXOUTI bit. The Firmware read the data and clear the FIFOCON bit in order to free the current bank. If the OUT Endpoint is composed of multiple banks, clearing the FIFOCON bit will
switch to the next bank. The RXOUTI and FIFOCON bits are then updated by hardware in accordance with the status of the new bank.

RXOUTI shall always be cleared before clearing FIFOCON.

The RWAL bit always reflects the state of the current bank. This bit is set if the firmware can read data from the bank, and cleared by hardware when the bank is empty.

**Example with 1 OUT data bank**

**Example with 2 OUT data banks**

### 22.13.2 Detailed description

#### 22.13.2.1

The data are read by the CPU, following the next flow:

- When the bank is filled by the host, an endpoint interrupt (EPINTx) is triggered, if enabled (RXOUTE set) and RXOUTI is set. The CPU can also poll RXOUTI or FIFOCON, depending on the software architecture,
- The CPU acknowledges the interrupt by clearing RXOUTI,
- The CPU can read the number of byte (N) in the current bank (N=BYCT),
- The CPU can read the data from the current bank ("N" read of UEDATX),
- The CPU can free the bank by clearing FIFOCON when all the data is read, that is:
  - after "N" read of UEDATX,
  - as soon as RWAL is cleared by hardware.

If the endpoint uses 2 banks, the second one can be filled by the HOST while the current one is being read by the CPU. Then, when the CPU clear FIFOCON, the next bank may be already ready and RXOUTI is set immediately.
22.14 IN endpoint management

IN packets are sent by the USB device controller, upon an IN request from the host. All the data can be written by the CPU, which acknowledge or not the bank when it is full. Overview

The Endpoint must be configured first.

The TXINI bit is set by hardware when the current bank becomes free. This triggers an interrupt if the TXINE bit is set. The FIFOCON bit is set at the same time. The CPU writes into the FIFO and clears the FIFOCON bit to allow the USB controller to send the data. If the IN Endpoint is composed of multiple banks, this also switches to the next data bank. The TXINI and FIFOCON bits are automatically updated by hardware regarding the status of the next bank.

TXINI shall always be cleared before clearing FIFOCON.

The RWAL bit always reflects the state of the current bank. This bit is set if the firmware can write data to the bank, and cleared by hardware when the bank is full.

Example with 1 IN data bank

Example with 2 IN data banks

22.14.1 Detailed description

The data are written by the CPU, following the next flow:

- When the bank is empty, an endpoint interrupt (EPINTx) is triggered, if enabled (TXINE set) and TXINI is set. The CPU can also poll TXINI or FIFOCON, depending the software architecture choice,
- The CPU acknowledges the interrupt by clearing TXINI,
- The CPU can write the data into the current bank (write in UEDATx),
- The CPU can free the bank by clearing FIFOCON when all the data are written, that is:
• after “N” write into UEDATX
• as soon as RWAL is cleared by hardware.

If the endpoint uses 2 banks, the second one can be read by the HOST while the current is being written by the CPU. Then, when the CPU clears FIFOCON, the next bank may be already ready (free) and TXIN is set immediately.

22.14.1.1 Abort

An “abort” stage can be produced by the host in some situations:

• In a control transaction: ZLP data OUT received during a IN stage,
• In an isochronous IN transaction: ZLP data OUT received on the OUT endpoint during a IN stage on the IN endpoint
• ...

The KILLBK bit is used to kill the last “written” bank. The best way to manage this abort is to perform the following operations:

Table 22-1. Abort flow

<table>
<thead>
<tr>
<th>Endpoint Abort</th>
<th>Clear UEIENX, TXINE</th>
</tr>
</thead>
<tbody>
<tr>
<td>NBUSYBK = 0</td>
<td>Yes</td>
</tr>
<tr>
<td>Endpoint reset</td>
<td>KILLBK = 1</td>
</tr>
<tr>
<td>Yes</td>
<td>KILLBK = 1</td>
</tr>
<tr>
<td>No</td>
<td></td>
</tr>
<tr>
<td>No</td>
<td></td>
</tr>
</tbody>
</table>

Abort done

| Disable the TXIN interrupt. |
| Aborting is based on the fact that no banks are busy, meaning that nothing has to be sent. |
| Kill the last written bank. |
| Wait for the end of the procedure. |

22.15 Isochronous mode

22.15.1 Underflow

An underflow can occur during IN stage if the host attempts to read a bank which is empty. In this situation, the UNDERFI interrupt is triggered.

An underflow can also occur during OUT stage if the host sends a packet while the banks are already full. Typically, the CPU is not fast enough. The packet is lost.

It is not possible to have underflow error during OUT stage, in the CPU side, since the CPU should read only if the bank is ready to give data (RXOUT = 1 or RWAL = 1)
22.15.2 CRC Error

A CRC error can occur during OUT stage if the USB controller detects a bad received packet. In this situation, the STALLEDI interrupt is triggered. This does not prevent the RXOUTI interrupt from being triggered.

22.16 Overflow

In Control, Isochronous, Bulk or Interrupt Endpoint, an overflow can occur during OUT stage, if the host attempts to write in a bank that is too small for the packet. In this situation, the OVERFI interrupt is triggered (if enabled). The packet is acknowledged and the RXOUTI interrupt is also triggered (if enabled). The bank is filled with the first bytes of the packet.

It is not possible to have overflow error during IN stage, in the CPU side, since the CPU should write only if the bank is ready to access data (TXIN=1 or RWAL=1).

22.17 Interrupts

The next figure shows all the interrupts sources:

Figure 22-4. USB Device Controller Interrupt System

There are 2 kind of interrupts: processing (i.e. their generation are part of the normal processing) and exception (errors).

Processing interrupts are generated when:

• VBUS plug-in detection (insert, remove)(VBUSTI)
• Upstream resume(UPRSMI)
• End of resume(EORSMI)
• Wake up(WAKEUPI)
• End of reset (Speed Initialization)(EORSTI)
• Start of frame(SOFI, if FNCERR=0)
• Suspend detected after 3 ms of inactivity(SUSPI)

Exception Interrupts are generated when:

• CRC error in frame number of SOFI(SOFI, FNCERR=1)
Processing interrupts are generated when:

- Ready to accept IN data (EPINTx, TXIN=1)
- Received OUT data (EPINTx, RXOUTI=1)
- Received SETUP (EPINTx, RXSTPI=1)

Exception Interrupts are generated when:

- Stalled packet (EPINTx, STALLEDI=1)
- CRC error on OUT in isochronous mode (EPINTx, STALLEDI=1)
- Overflow in isochronous mode (EPINTx, OVERFI=1)
- Underflow in isochronous mode (EPINTx, UNDERFI=1)
- NAK IN sent (EPINTx, NAKINI=1)
- NAK OUT sent (EPINTx, NAKOUTI=1)

### 22.18 Registers

#### 22.18.1 USB device general registers

<table>
<thead>
<tr>
<th>Bit</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
<th>UDCON</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read/Write</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td></td>
</tr>
<tr>
<td>Initial Value</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>
• **7-4 - Reserved**
The value read from these bits is always 0. Do not set these bits.

• **3 - RSTCPU - USB Reset CPU bit**
Set this bit to 1 by firmware in order to reset the CPU on the detection of a USB End of Reset signal (without disabling the USB controller and Attached state). This bit is reset when the USB controller is disabled, but is not affected by the CPU reset generated after a USB End of Reset (remains enabled).

• **2 - LSM - USB Device Low Speed Mode Selection**
When configured USB is configured in device mode, this bit allows to select the USB the USB Low Speed or Full Speed Mod.
Clear to select full speed mode (D+ internal pull-up will be activate with the ATTACH bit will be set).
Set to select low speed mode (D- internal pull-up will be activate with the ATTACH bit will be set). This bit has no effect when the USB interface is configured in HOST mode.

• **1 - RMWKUP - Remote Wake-up Bit**
Set to send an “upstream-resume” to the host for a remote wake-up (the SUSPI bit must be set).
Cleared by hardware when signalling finished. Clearing by software has no effect.
See Section 22.10, page 270 for more details.

• **0 - DETACH - Detach Bit**
Set to physically detach de device (disconnect internal pull-up on D+ or D-).
Clear to reconnect the device. See Section 22.9, page 269 for more details.

```
<table>
<thead>
<tr>
<th>Bit</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read/Write</td>
<td>-</td>
<td>UPRSMI</td>
<td>EORSMI</td>
<td>WAKEUP</td>
<td>EORSTI</td>
<td>SOFI</td>
<td>-</td>
<td>SUSPI</td>
</tr>
<tr>
<td>Initial Value</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
```

• **7 - Reserved**
The value read from this bits is always 0. Do not set this bit.

• **6 - UPRSMI - Upstream Resume Interrupt Flag**
Set by hardware when the USB controller is sending a resume signal called “Upstream Resume”. This triggers an USB interrupt if UPRSME is set.
Shall be cleared by software (USB clocks must be enabled before). Setting by software has no effect.

• **5 - EORSMI - End Of Resume Interrupt Flag**
Set by hardware when the USB controller detects a good “End Of Resume” signal initiated by the host. This triggers an USB interrupt if EORSME is set.
Shall be cleared by software. Setting by software has no effect.
• 4 - WAKEUPI - Wake-up CPU Interrupt Flag
Set by hardware when the USB controller is re-activated by a filtered non-idle signal from the
lines (not by an upstream resume). This triggers an interrupt if WAKEUPE is set.
Shall be cleared by software (USB clock inputs must be enabled before). Setting by software
has no effect.
See Section 22.8, page 269 for more details.

• 3 - EORSTI - End Of Reset Interrupt Flag
Set by hardware when an “End Of Reset” has been detected by the USB controller. This triggers
an USB interrupt if EORSTE is set.
Shall be cleared by software. Setting by software has no effect.

• 2 - SOFI - Start Of Frame Interrupt Flag
Set by hardware when an USB “Start Of Frame” PID (SOF) has been detected (every 1 ms).
This triggers an USB interrupt if SOFE is set.

• 1 - Reserved
The value read from this bits is always 0. Do not set this bit

• 0 - SUSPI - Suspend Interrupt Flag
Set by hardware when an USB “Suspend” ‘idle bus for 3 frame periods: a J state for 3 ms) is
detected. This triggers an USB interrupt if SUSPE is set.
Shall be cleared by software. Setting by software has no effect.
See Section 22.8, page 269 for more details.
The interrupt bits are set even if their corresponding ‘Enable’ bits is not set.

<table>
<thead>
<tr>
<th>Bit</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
<th>UDIEN</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read/Write</td>
<td>-</td>
<td>UPRSME</td>
<td>EORSME</td>
<td>WAKEUPI</td>
<td>EORSTE</td>
<td>SOFE</td>
<td>-</td>
<td>SUSPE</td>
<td></td>
</tr>
<tr>
<td>Initial Value</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

• 7 - Reserved
The value read from this bits is always 0. Do not set this bit.

• 6 - UPRSME - Upstream Resume Interrupt Enable Bit
Set to enable the UPRSMI interrupt.
Clear to disable the UPRSMI interrupt.

• 5 - EORSME - End Of Resume Interrupt Enable Bit
Set to enable the EORSMI interrupt.
Clear to disable the EORSMI interrupt.

• 4 - WAKEUPE - Wake-up CPU Interrupt Enable Bit
Set to enable the WAKEUPI interrupt.
Clear to disable the WAKEUPI interrupt.
• **3 - EORSTE - End Of Reset Interrupt Enable Bit**
Set to enable the EORSTI interrupt. This bit is set after a reset.
Clear to disable the EORSTI interrupt.

• **2 - SOFE - Start Of Frame Interrupt Enable Bit**
Set to enable the SOFI interrupt.
Clear to disable the SOFI interrupt.

• **1 - Reserved**
The value read from this bits is always 0. Do not set this bit.

• **0 - SUSPE - Suspend Interrupt Enable Bit**
Set to enable the SUSPI interrupt.
Clear to disable the SUSPI interrupt.

<table>
<thead>
<tr>
<th>Bit</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read/Write</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>W</td>
</tr>
<tr>
<td>Initial Value</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

• **7 - ADDEN - Address Enable Bit**
Set to activate the UADD (USB address).
Cleared by hardware. Clearing by software has no effect.
See Section 22.7, page 268 for more details.

<table>
<thead>
<tr>
<th>Bit</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read/Write</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
</tr>
<tr>
<td>Initial Value</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

• **7-3 - Reserved**
The value read from these bits is always 0. Do not set these bits.

• **2-0 - FNUM10:8 - Frame Number Upper Value**
Set by hardware. These bits are the 3 MSB of the 11-bits Frame Number information. They are provided in the last received SOF packet. FNUM is updated if a corrupted SOF is received.
• **Frame Number Lower Value**
Set by hardware. These bits are the 8 LSB of the 11-bits Frame Number information.

<table>
<thead>
<tr>
<th>Bit</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read/Write</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
</tr>
<tr>
<td>Initial Value</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

• **7-5 - Reserved**
The value read from these bits is always 0. Do not set these bits.

• **4 - FNCERR - Frame Number CRC Error Flag**
Set by hardware when a corrupted Frame Number in start of frame packet is received. This bit and the SOFI interrupt are updated at the same time.

• **3-0 - Reserved**
The value read from these bits is always 0. Do not set these bits.

### 22.18.2 USB device endpoint registers

<table>
<thead>
<tr>
<th>Bit</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read/Write</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
</tr>
<tr>
<td>Initial Value</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

• **7-3 - Reserved**
The value read from these bits is always 0. Do not set these bits.

• **2-0 - EPNUM2:0 Endpoint Number Bits**
Load by software to select the number of the endpoint which shall be accessed by the CPU. See Section 22.5, page 267 for more details.

EPNUM = 111b is forbidden.

<table>
<thead>
<tr>
<th>Bit</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read/Write</td>
<td>R</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
</tr>
<tr>
<td>Initial Value</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

• **7 - Reserved**
The value read from these bits is always 0. Do not set these bits.

• **6-0 - EPRST6:0 - Endpoint FIFO Reset Bits**
Set to reset the selected endpoint FIFO prior to any other operation, upon hardware reset or when an USB bus reset has been received. See Section 22.3, page 266 for more information.
Then, clear by software to complete the reset operation and start using the endpoint.

<table>
<thead>
<tr>
<th>Bit</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
<th>UEOCNX</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read/Write</td>
<td>R</td>
<td>R</td>
<td>W</td>
<td>W</td>
<td>W</td>
<td>R</td>
<td>R</td>
<td>R/W</td>
<td></td>
</tr>
<tr>
<td>Initial Value</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

- **7-6 - Reserved**
The value read from these bits is always 0. Do not set these bits.

- **5 - STALLRQ - STALL Request Handshake Bit**
Set to request a STALL answer to the host for the next handshake.

  Cleared by hardware when a new SETUP is received. Clearing by software has no effect.

  See Section 22.11, page 270 for more details.

- **4 - STALLRQC - STALL Request Clear Handshake Bit**
Set to disable the STALL handshake mechanism.

  Cleared by hardware immediately after the set. Clearing by software has no effect.

  See Section 22.11, page 270 for more details.

- **RSTDT - Reset Data Toggle Bit**
Set to automatically clear the data toggle sequence:

  For OUT endpoint: the next received packet will have the data toggle 0.

  For IN endpoint: the next packet to be sent will have the data toggle 0.

  Cleared by hardware instantaneously. The firmware does not have to wait that the bit is cleared.

  Clearing by software has no effect.

- **2 - Reserved**
The value read from these bits is always 0. Do not set these bits.

- **1 - Reserved**
The value read from these bits is always 0. Do not set these bits.

- **0 - EPEN - Endpoint Enable Bit**
Set to enable the endpoint according to the device configuration. Endpoint 0 shall always be enabled after a hardware or USB reset and participate in the device configuration.

  Clear this bit to disable the endpoint. See Section 22.6, page 267 for more details.

<table>
<thead>
<tr>
<th>Bit</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
<th>UECFG00X</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read/Write</td>
<td>R/W</td>
<td>R/W</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R/W</td>
<td></td>
</tr>
<tr>
<td>Initial Value</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

- **7-6 - ETYPE1:0 - Endpoint Type Bits**
Set this bit according to the endpoint configuration:
00b: Control 10b: Bulk
01b: Isochronous 11b: Interrupt

- **5-1 - Reserved**
The value read from these bits is always 0. Do not set these bits.

- **0 - EDIR - Endpoint Direction Bit**
Set to configure an IN direction for bulk, interrupt or isochronous endpoints.
Clear to configure an OUT direction for bulk, interrupt, isochronous or control endpoints.

<table>
<thead>
<tr>
<th>Bit</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read/Write</td>
<td>R</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R</td>
</tr>
<tr>
<td>Initial Value</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

- **7 - Reserved**
The value read from these bits is always 0. Do not set these bits.

- **6-4 - EPSIZE2:0 - Endpoint Size Bits**
Set this bit according to the endpoint size:

  - 000b: 8 bytes
  - 100b: 128 bytes
  - 001b: 16 bytes
  - 101b: 256 bytes
  - 010b: 32 bytes
  - 110b: 512 bytes
  - 011b: 64 bytes

- **3-2 - EPBK1:0 - Endpoint Bank Bits**
Set this field according to the endpoint size:

  - 00b: One bank
  - 01b: Double bank
  - 1xb: Reserved. Do not use this configuration.

- **1 - ALLOC - Endpoint Allocation Bit**
Set this bit to allocate the endpoint memory.
Clear to free the endpoint memory.
See Section 22.6, page 267 for more details.

- **0 - Reserved**
The value read from these bits is always 0. Do not set these bits.
• 7 - CFGOK - Configuration Status Flag
Set by hardware when the endpoint X size parameter (EPSIZE) and the bank parametrization (EPBK) are correct compared to the max FIFO capacity and the max number of allowed bank. This bit is updated when the bit ALLOC is set.
If this bit is cleared, the user should reprogram the UECFG1X register with correct EPSIZE and EPBK values.

• 6 - OVERFI - Overflow Error Interrupt Flag
Set by hardware when an overflow error occurs in an isochronous endpoint. An interrupt (EPINTx) is triggered (if enabled).
See Section 22.15, page 275 for more details.
Shall be cleared by software. Setting by software has no effect.

• 5 - UNDERFI - Flow Error Interrupt Flag
Set by hardware when an underflow error occurs in an isochronous endpoint. An interrupt (EPINTx) is triggered (if enabled).
See Section 22.15, page 275 for more details.
Shall be cleared by software. Setting by software has no effect.

• 4 - Reserved
The value read from these bits is always 0. Do not set these bits.

• 3-2 - DTSEQ1:0 - Data Toggle Sequencing Flag
Set by hardware to indicate the PID data of the current bank:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>00b</td>
<td>Data0</td>
</tr>
<tr>
<td>01b</td>
<td>Data1</td>
</tr>
<tr>
<td>1xb</td>
<td>Reserved.</td>
</tr>
</tbody>
</table>

For OUT transfer, this value indicates the last data toggle received on the current bank.
For IN transfer, it indicates the Toggle that will be used for the next packet to be sent. This is not relative to the current bank.

• 1-0 - NBUSYBK1:0 - Busy Bank Flag
Set by hardware to indicate the number of busy bank.
For IN endpoint, it indicates the number of busy bank(s), filled by the user, ready for IN transfer.
For OUT endpoint, it indicates the number of busy bank(s) filled by OUT transaction from the host.

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>00b</td>
<td>All banks are free</td>
</tr>
<tr>
<td>01b</td>
<td>1 busy bank</td>
</tr>
<tr>
<td>10b</td>
<td>2 busy banks</td>
</tr>
<tr>
<td>11b</td>
<td>Reserved.</td>
</tr>
</tbody>
</table>
• **7-3 - Reserved**
The value read from these bits is always 0. Do not set these bits.

• **2 - CTRLDIR - Control Direction (Flag, and bit for debug purpose)**
Set by hardware after a SETUP packet, and gives the direction of the following packet:
- 1 for IN endpoint
- 0 for OUT endpoint.
Can not be set or cleared by software.

• **1-0 - CURRBK1:0 - Current Bank (all endpoints except Control endpoint) Flag**
Set by hardware to indicate the number of the current bank:
00b  Bank0
01b  Bank1
1xb  Reserved.
Can not be set or cleared by software.

• **7 - FIFOCON - FIFO Control Bit**
For OUT and SETUP Endpoint:
Set by hardware when a new OUT message is stored in the current bank, at the same time than RXOUT or RXSTP.
Clear to free the current bank and to switch to the following bank. Setting by software has no effect.

For IN Endpoint:
Set by hardware when the current bank is free, at the same time than TXIN.
Clear to send the FIFO data and to switch the bank. Setting by software has no effect.

• **6 - NAKINI - NAK IN Received Interrupt Flag**
Set by hardware when a NAK handshake has been sent in response of a IN request from the host. This triggers an USB interrupt if NAKINE is sent.
Shall be cleared by software. Setting by software has no effect.
5 - RWAL - Read/Write Allowed Flag
Set by hardware to signal:
- for an IN endpoint: the current bank is not full i.e. the firmware can push data into the FIFO,
- for an OUT endpoint: the current bank is not empty, i.e. the firmware can read data from the FIFO.
The bit is never set if STALLRQ is set, or in case of error.
Cleared by hardware otherwise.
This bit shall not be used for the control endpoint.

4 - NAKOUTI - NAK OUT Received Interrupt Flag
Set by hardware when a NAK handshake has been sent in response of a OUT/PING request from the host. This triggers an USB interrupt if NAKOUTE is sent.
Shall be cleared by software. Setting by software has no effect.

3 - RXSTPI - Received SETUP Interrupt Flag
Set by hardware to signal that the current bank contains a new valid SETUP packet. An interrupt (EPINTx) is triggered (if enabled).
Shall be cleared by software to handshake the interrupt. Setting by software has no effect.
This bit is inactive (cleared) if the endpoint is an IN endpoint.

2 - RXOUTI / KILLBK - Received OUT Data Interrupt Flag
Set by hardware to signal that the current bank contains a new packet. An interrupt (EPINTx) is triggered (if enabled).
Shall be cleared by software to handshake the interrupt. Setting by software has no effect.

Kill Bank IN Bit
Set this bit to kill the last written bank.
Cleared by hardware when the bank is killed. Clearing by software has no effect.
See page 275 for more details on the Abort.

1 - STALLEDI - STALLEDI Interrupt Flag
Set by hardware to signal that a STALL handshake has been sent, or that a CRC error has been detected in a OUT isochronous endpoint.
Shall be cleared by software. Setting by software has no effect.

0 - TXINI - Transmitter Ready Interrupt Flag
Set by hardware to signal that the current bank is free and can be filled. An interrupt (EPINTx) is triggered (if enabled).
Shall be cleared by software to handshake the interrupt. Setting by software has no effect.
This bit is inactive (cleared) if the endpoint is an OUT endpoint.
• 7 - FLERRE - Flow Error Interrupt Enable Flag  
Set to enable an endpoint interrupt (EPINTx) when OVERFI or UNDERFI are sent.  
Clear to disable an endpoint interrupt (EPINTx) when OVERFI or UNDERFI are sent.

• 6 - NAKINE - NAK IN Interrupt Enable Bit  
Set to enable an endpoint interrupt (EPINTx) when NAKINI is set.  
Clear to disable an endpoint interrupt (EPINTx) when NAKINI is set.

• 5 - Reserved  
The value read from these bits is always 0. Do not set these bits.

• 4 - NAKOUTE - NAK OUT Interrupt Enable Bit  
Set to enable an endpoint interrupt (EPINTx) when NAKOUTI is set.  
Clear to disable an endpoint interrupt (EPINTx) when NAKOUTI is set.

• 3 - RXSTPE - Received SETUP Interrupt Enable Flag  
Set to enable an endpoint interrupt (EPINTx) when RXSTPI is sent.  
Clear to disable an endpoint interrupt (EPINTx) when RXSTPI is sent.

• 2 - RXOUTE - Received OUT Data Interrupt Enable Flag  
Set to enable an endpoint interrupt (EPINTx) when RXOUTI is sent.  
Clear to disable an endpoint interrupt (EPINTx) when RXOUTI is sent.

• 1 - STALLEDE - Stalled Interrupt Enable Flag  
Set to enable an endpoint interrupt (EPINTx) when STALLEDI is sent.  
Clear to disable an endpoint interrupt (EPINTx) when STALLEDI is sent.

• 0 - TXINE - Transmitter Ready Interrupt Enable Flag  
Set to enable an endpoint interrupt (EPINTx) when TXINI is sent.  
Clear to disable an endpoint interrupt (EPINTx) when TXINI is sent.

• 7-0 - DAT7:0 -Data Bits  
Set by the software to read/write a byte from/to the endpoint FIFO selected by EPNUM.
• 7-3 - Reserved
The value read from these bits is always 0. Do not set these bits.

• 2-0 - BYCT10:8 - Byte count (high) Bits
Set by hardware. This field is the MSB of the byte count of the FIFO endpoint. The LSB part is provided by the UEBCLX register.

• 7-0 - BYCT7:0 - Byte Count (low) Bits
Set by the hardware. BYCT10:0 is:
- (for IN endpoint) increased after each writing into the endpoint and decremented after each byte sent,
- (for OUT endpoint) increased after each byte sent by the host, and decremented after each byte read by the software.

• 7 - Reserved
The value read from these bits is always 0. Do not set these bits.

• 6-0 - EPINT6:0 - Endpoint Interrupts Bits
Set by hardware when an interrupt is triggered by the UEINTX register and if the corresponding endpoint interrupt enable bit is set.
Cleared by hardware when the interrupt source is served.
23. Analog Comparator

The Analog Comparator compares the input values on the positive pin AIN+ and negative pin AIN-. When the voltage on the positive pin AIN+ is higher than the voltage on the negative pin AIN-, the Analog Comparator output, ACO, is set. The comparator's output can be set to trigger the Timer/Counter1 Input Capture function. In addition, the comparator can trigger a separate interrupt, exclusive to the Analog Comparator. The user can select Interrupt triggering on comparator output rise, fall or toggle. A block diagram of the comparator and its surrounding logic is shown in Figure 23-1. AIN+ can be connected either to the AIN0 (PE6) pin, or to the internal Bandgap reference. AIN- can only be connected to the ADC multiplexer.

The Power Reduction ADC bit, PRADC, in “Power Reduction Register 0 - PRR0” on page 46 must be disabled by writing a logical zero to be able to use the ADC input MUX.

Figure 23-1. Analog Comparator Block Diagram(2)

Notes:
1. See Table 23-2 on page 291.
2. Refer to “Pinout ATmega16U4/ATmega32U4” on page 3 and Table 10-3 on page 72 for Analog Comparator pin placement.

23.0.1 ADC Control and Status Register B – ADCSRB

<table>
<thead>
<tr>
<th>Bit</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
<th>ADCCSRB</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read/Write</td>
<td>R</td>
<td>R/W</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>ADCSRB</td>
</tr>
<tr>
<td>Initial Value</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>ADCSRB</td>
</tr>
</tbody>
</table>

- **Bit 6 – ACME: Analog Comparator Multiplexer Enable**

  When this bit is written logic one and the ADC is switched off (ADEN in ADCSRA is zero), the ADC multiplexer is connected to the negative input to the Analog Comparator. When this bit is written logic zero, the Bandgap reference is connected to the negative input of the Analog Comparator (See “Internal Voltage Reference” on page 54.). For a detailed description of this bit, see “Analog Comparator Multiplexed Input” on page 291.

23.0.2 Analog Comparator Control and Status Register – ACSR

<table>
<thead>
<tr>
<th>Bit</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
<th>ACSR</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read/Write</td>
<td>R/W</td>
<td>R/W</td>
<td>R</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>ACSR</td>
</tr>
<tr>
<td>Initial Value</td>
<td>0</td>
<td>0</td>
<td>N/A</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>ACSR</td>
</tr>
</tbody>
</table>

---

ATmega16/32U4

7766F–AVR–11/10
• Bit 7 – ACD: Analog Comparator Disable
When this bit is written logic one, the power to the Analog Comparator is switched off. This bit can be set at any time to turn off the Analog Comparator. This will reduce power consumption in Active and Idle mode. When changing the ACD bit, the Analog Comparator Interrupt must be disabled by clearing the ACIE bit in ACSR. Otherwise an interrupt can occur when the bit is changed.

• Bit 6 – ACBG: Analog Comparator Bandgap Select
When this bit is set, a fixed bandgap reference voltage replaces the positive input to the Analog Comparator. When this bit is cleared, AI0 is applied to the positive input of the Analog Comparator. See “Internal Voltage Reference” on page 54.

• Bit 5 – ACO: Analog Comparator Output
The output of the Analog Comparator is synchronized and then directly connected to ACO. The synchronization introduces a delay of 1 - 2 clock cycles.

• Bit 4 – ACI: Analog Comparator Interrupt Flag
This bit is set by hardware when a comparator output event triggers the interrupt mode defined by ACIS1 and ACIS0. The Analog Comparator interrupt routine is executed if the ACIE bit is set and the I-bit in SREG is set. ACI is cleared by hardware when executing the corresponding interrupt handling vector. Alternatively, ACI is cleared by writing a logic one to the flag.

• Bit 3 – ACIE: Analog Comparator Interrupt Enable
When the ACIE bit is written logic one and the I-bit in the Status Register is set, the Analog Comparator interrupt is activated. When written logic zero, the interrupt is disabled.

• Bit 2 – ACIC: Analog Comparator Input Capture Enable
When written logic one, this bit enables the input capture function in Timer/Counter1 to be triggered by the Analog Comparator. The comparator output is in this case directly connected to the input capture front-end logic, making the comparator utilize the noise canceler and edge select features of the Timer/Counter1 Input Capture interrupt. When written logic zero, no connection between the Analog Comparator and the input capture function exists. To make the comparator trigger the Timer/Counter1 Input Capture interrupt, the ICIE1 bit in the Timer Interrupt Mask Register (TIMSK1) must be set.

• Bits 1, 0 – ACIS1, ACIS0: Analog Comparator Interrupt Mode Select
These bits determine which comparator events that trigger the Analog Comparator interrupt. The different settings are shown in Table 23-1.

<table>
<thead>
<tr>
<th>ACIS1</th>
<th>ACIS0</th>
<th>Interrupt Mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>Comparator Interrupt on Output Toggle.</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>Reserved</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>Comparator Interrupt on Falling Output Edge.</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>Comparator Interrupt on Rising Output Edge.</td>
</tr>
</tbody>
</table>

When changing the ACIS1/ACIS0 bits, the Analog Comparator Interrupt must be disabled by clearing its Interrupt Enable bit in the ACSR Register. Otherwise an interrupt can occur when the bits are changed.
23.1 Analog Comparator Multiplexed Input

It is possible to select any of the ADC13..0 pins to replace the negative input to the Analog Comparator. The ADC multiplexer is used to select this input, and consequently, the ADC must be switched off to utilize this feature. If the Analog Comparator Multiplexer Enable bit (ACME in ADCSRB) is set and the ADC is switched off (ADEN in ADCSRA is zero), and MUX2..0 in ADMUX select the input pin to replace the negative input to the Analog Comparator, as shown in Table 23-2. If ACME is cleared or ADEN is set, the Bandgap reference is applied to the negative input to the Analog Comparator.

<table>
<thead>
<tr>
<th>ACME</th>
<th>ADEN</th>
<th>MUX2..0</th>
<th>Analog Comparator Negative Input</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>x</td>
<td>xxx</td>
<td>Bandgap Ref.</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>xxx</td>
<td>Bandgap Ref.</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>000</td>
<td>ADC0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>001</td>
<td>ADC1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>010</td>
<td>N/A</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>011</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>100</td>
<td>ADC4</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>101</td>
<td>ADC5</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>110</td>
<td>ADC6</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>111</td>
<td>ADC7</td>
</tr>
</tbody>
</table>

23.1.1 Digital Input Disable Register 1 – DIDR1

<table>
<thead>
<tr>
<th>Bit</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
<th><strong>AIN0D</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Read/Write</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R/W</td>
<td></td>
</tr>
<tr>
<td>Initial Value</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

- **Bit 0 – AIN0D: AIN0 Digital Input Disable**
  When this bit is written logic one, the digital input buffer on the AIN0 pin is disabled. The corresponding PIN Register bit will always read as zero when this bit is set. When an analog signal is applied to the AIN0 pin and the digital input from this pin is not needed, this bit should be written logic one to reduce power consumption in the digital input buffer.
24. Analog to Digital Converter - ADC

24.1 Features

- 10/8-bit Resolution
- 0.5 LSB Integral Non-linearity
- ± 2 LSB Absolute Accuracy
- 65 - 260 µs Conversion Time
- Up to 15 kSPS at Maximum Resolution
- Twelve Multiplexed Single-Ended Input Channels
- One Differential amplifier providing gain of 1x - 10x - 40x - 200x
- Temperature sensor
- Optional Left Adjustment for ADC Result Readout
- 0 - V\text{CC} \ ADC Input Voltage Range
- Selectable 2.56 V ADC Reference Voltage
- Free Running or Single Conversion Mode
- ADC Start Conversion by Auto Triggering on Interrupt Sources
- Interrupt on ADC Conversion Complete
- Sleep Mode Noise Canceler

The ATmega16U4/ATmega32U4 features a 10-bit successive approximation ADC. The ADC is connected to an 12-channel Analog Multiplexer which allows six single-ended voltage inputs constructed from several pins of Port B, D and F. The single-ended voltage inputs refer to 0V (GND).

The device also supports 32 differential voltage input combinations, thanks to a differential amplifier equipped with a programmable gain stage, providing amplification steps of 0 dB (1x), 10 dB (10x), 16dB (40x) or 23dB (200x) on the differential input voltage before the A/D conversion. Two differential analog input channels share a common negative terminal (ADC0/ADC1), while any other ADC input can be selected as the positive input terminal. If 1x, 10x or 40x gain is used, 8-bit resolution can be expected. If 200x gain is used, 7-bit resolution can be expected.

The ADC contains a Sample and Hold circuit which ensures that the input voltage to the ADC is held at a constant level during conversion. A block diagram of the ADC is shown in Figure 24-1.

The ADC has a separate analog supply voltage pin, AV\text{CC}. AV\text{CC} must not differ more than ± 0.3V from V\text{CC}. See the paragraph “ADC Noise Canceler” on page 301 on how to connect this pin.

Internal reference voltages of nominally 2.56V or AV\text{CC} are provided On-chip. The voltage reference may be externally decoupled at the AREF pin by a capacitor for better noise performance.
Figure 24-1. Analog to Digital Converter Block Schematic
24.2 Operation

The ADC converts an analog input voltage to a 10-bit digital value through successive approximation. The minimum value represents GND and the maximum value represents the voltage on the AREF pin minus 1 LSB. Optionally, AVcc or an internal 2.56V reference voltage may be connected to the AREF pin by writing to the REFSn bits in the ADMUX Register. The internal voltage reference may thus be decoupled by an external capacitor at the AREF pin to improve noise immunity.

The analog input channel and differential gain are selected by writing to the MUX bits in ADMUX. Any of the ADC input pins, as well as GND and a fixed bandgap voltage reference, can be selected as single ended inputs to the ADC. A selection of ADC input pins can be selected as positive and negative inputs to the differential amplifier.

The ADC is enabled by setting the ADC Enable bit, ADEN in ADCSRA. Voltage reference and input channel selections will not go into effect until ADEN is set. The ADC does not consume power when ADEN is cleared, so it is recommended to switch off the ADC before entering power saving sleep modes.

The ADC generates a 10-bit result which is presented in the ADC Data Registers, ADCH and ADCL. By default, the result is presented right adjusted, but can optionally be presented left adjusted by setting the ADLAR bit in ADMUX.

If the result is left adjusted and no more than 8-bit precision is required, it is sufficient to read ADCH. Otherwise, ADCL must be read first, then ADCH, to ensure that the content of the Data Registers belongs to the same conversion. Once ADCL is read, ADC access to Data Registers is blocked. This means that if ADCL has been read, and a conversion completes before ADCH is read, neither register is updated and the result from the conversion is lost. When ADCH is read, ADC access to the ADCH and ADCL Registers is re-enabled.

The ADC has its own interrupt which can be triggered when a conversion completes. The ADC access to the Data Registers is prohibited between reading of ADCH and ADCL, the interrupt will trigger even if the result is lost.

24.3 Starting a Conversion

A single conversion is started by writing a logical one to the ADC Start Conversion bit, ADSC. This bit stays high as long as the conversion is in progress and will be cleared by hardware when the conversion is completed. If a different data channel is selected while a conversion is in progress, the ADC will finish the current conversion before performing the channel change.

Alternatively, a conversion can be triggered automatically by various sources. Auto Triggering is enabled by setting the ADC Auto Trigger Enable bit, ADATE in ADCSRA. The trigger source is selected by setting the ADC Trigger Select bits, ADTS in ADCSRB (See description of the ADTS bits for a list of the trigger sources). When a positive edge occurs on the selected trigger signal, the ADC prescaler is reset and a conversion is started. This provides a method of starting conversions at fixed intervals. If the trigger signal is still set when the conversion completes, a new conversion will not be started. If another positive edge occurs on the trigger signal during conversion, the edge will be ignored. Note that an interrupt flag will be set even if the specific interrupt is disabled or the Global Interrupt Enable bit in SREG is cleared. A conversion can thus be triggered without causing an interrupt. However, the interrupt flag must be cleared in order to trigger a new conversion at the next interrupt event.
Using the ADC Interrupt Flag as a trigger source makes the ADC start a new conversion as soon as the ongoing conversion has finished. The ADC then operates in Free Running mode, constantly sampling and updating the ADC Data Register. The first conversion must be started by writing a logical one to the ADSC bit in ADCSRA. In this mode the ADC will perform successive conversions independently of whether the ADC Interrupt Flag, ADIF is cleared or not.

If Auto Triggering is enabled, single conversions can be started by writing ADSC in ADCSRA to one. ADSC can also be used to determine if a conversion is in progress. The ADSC bit will be read as one during a conversion, independently of how the conversion was started.

### 24.4 Prescaling and Conversion Timing

By default, the successive approximation circuitry requires an input clock frequency between 50 kHz and 200 kHz to get maximum resolution. If a lower resolution than 10 bits is needed, the input clock frequency to the ADC can be higher than 200 kHz to get a higher sample rate. Alternatively, setting the ADHSM bit in ADCSRB allows an increased ADC clock frequency at the expense of higher power consumption.

The ADC module contains a prescaler, which generates an acceptable ADC clock frequency from any CPU frequency above 100 kHz. The prescaling is set by the ADPS bits in ADCSRA. The prescaler starts counting from the moment the ADC is switched on by setting the ADEN bit
in ADCSRA. The prescaler keeps running for as long as the ADEN bit is set, and is continuously reset when ADEN is low.

When initiating a single ended conversion by setting the ADSC bit in ADCSRA, the conversion starts at the following rising edge of the ADC clock cycle. See “Differential Channels” on page 297 for details on differential conversion timing.

A normal conversion takes 13 ADC clock cycles. The first conversion after the ADC is switched on (ADEN in ADCSRA is set) takes 25 ADC clock cycles in order to initialize the analog circuitry.

The actual sample-and-hold takes place 1.5 ADC clock cycles after the start of a normal conversion and 13.5 ADC clock cycles after the start of an first conversion. When a conversion is complete, the result is written to the ADC Data Registers, and ADIF is set. In Single Conversion mode, ADSC is cleared simultaneously. The software may then set ADSC again, and a new conversion will be initiated on the first rising ADC clock edge.

When Auto Triggering is used, the prescaler is reset when the trigger event occurs. This assures a fixed delay from the trigger event to the start of conversion. In this mode, the sample-and-hold takes place two ADC clock cycles after the rising edge on the trigger source signal. Three additional CPU clock cycles are used for synchronization logic.

In Free Running mode, a new conversion will be started immediately after the conversion completes, while ADSC remains high. For a summary of conversion times, see Table 24-1.

Figure 24-4. ADC Timing Diagram, First Conversion (Single Conversion Mode)

Figure 24-5. ADC Timing Diagram, Single Conversion
24.4.1 Differential Channels

When using differential channels, certain aspects of the conversion need to be taken into consideration.

Differential conversions are synchronized to the internal clock $\text{CK}_{\text{ADC2}}$ equal to half the ADC clock frequency. This synchronization is done automatically by the ADC interface in such a way that the sample-and-hold occurs at a specific phase of $\text{CK}_{\text{ADC2}}$. A conversion initiated by the user (i.e., all single conversions, and the first free running conversion) when $\text{CK}_{\text{ADC2}}$ is low will take the same amount of time as a single ended conversion (13 ADC clock cycles from the next prescaled clock cycle). A conversion initiated by the user when $\text{CK}_{\text{ADC2}}$ is high will take 14 ADC cycles.
clock cycles due to the synchronization mechanism. In Free Running mode, a new conversion is initiated immediately after the previous conversion completes, and since \( \text{CK}_{\text{ADC2}} \) is high at this time, all automatically started (i.e., all but the first) Free Running conversions will take 14 ADC clock cycles.

If differential channels are used and conversions are started by Auto Triggering, the ADC must be switched off between conversions. When Auto Triggering is used, the ADC prescaler is reset before the conversion is started. Since the stage is dependent of a stable ADC clock prior to the conversion, this conversion will not be valid. By disabling and then re-enabling the ADC between each conversion (writing \( \text{ADEN} \) in \( \text{ADCSRA} \) to “0” then to “1”), only extended conversions are performed. The result from the extended conversions will be valid. See “Prescaling and Conversion Timing” on page 295 for timing details.

The gain stage is optimized for a bandwidth of 4 kHz at all gain settings. Higher frequencies may be subjected to non-linear amplification. An external low-pass filter should be used if the input signal contains higher frequency components than the gain stage bandwidth. Note that the ADC clock frequency is independent of the gain stage bandwidth limitation. E.g. the ADC clock period may be 6 μs, allowing a channel to be sampled at 12 kSPS, regardless of the bandwidth of this channel.

24.5 Changing Channel or Reference Selection

The MUXn and REFS1:0 bits in the ADMUX Register are single buffered through a temporary register to which the CPU has random access. This ensures that the channels and reference selection only takes place at a safe point during the conversion. The channel and reference selection is continuously updated until a conversion is started. Once the conversion starts, the channel and reference selection is locked to ensure a sufficient sampling time for the ADC. Continuous updating resumes in the last ADC clock cycle before the conversion completes (ADIF in \( \text{ADCSRA} \) is set). Note that the conversion starts on the following rising ADC clock edge after ADSC is written. The user is thus advised not to write new channel or reference selection values to ADMUX until one ADC clock cycle after ADSC is written.

If Auto Triggering is used, the exact time of the triggering event can be indeterministic. Special care must be taken when updating the ADMUX Register, in order to control which conversion will be affected by the new settings.

If both ADATE and ADEN is written to one, an interrupt event can occur at any time. If the ADMUX Register is changed in this period, the user cannot tell if the next conversion is based on the old or the new settings. ADMUX can be safely updated in the following ways:

a. When ADATE or ADEN is cleared.
b. During conversion, minimum one ADC clock cycle after the trigger event.
c. After a conversion, before the interrupt flag used as trigger source is cleared.

When updating ADMUX in one of these conditions, the new settings will affect the next ADC conversion.

Special care should be taken when changing differential channels. Once a differential channel has been selected, the stage may take as much as 125 μs to stabilize to the new value. Thus conversions should not be started within the first 125 μs after selecting a new differential channel. Alternatively, conversion results obtained within this period should be discarded.

The same settling time should be observed for the first differential conversion after changing ADC reference (by changing the REFS1:0 bits in ADMUX).
The settling time and gain stage bandwidth is independent of the ADHSM bit setting.

24.5.1 ADC Input Channels

When changing channel selections, the user should observe the following guidelines to ensure that the correct channel is selected:

- In Single Conversion mode, always select the channel before starting the conversion. The channel selection may be changed one ADC clock cycle after writing one to ADSC. However, the simplest method is to wait for the conversion to complete before changing the channel selection.

- In Free Running mode, always select the channel before starting the first conversion. The channel selection may be changed one ADC clock cycle after writing one to ADSC. However, the simplest method is to wait for the first conversion to complete, and then change the channel selection. Since the next conversion has already started automatically, the next result will reflect the previous channel selection. Subsequent conversions will reflect the new channel selection.

When switching to a differential gain channel, the first conversion result may have a poor accuracy due to the required settling time for the automatic offset cancellation circuitry. The user should preferably disregard the first conversion result.

24.5.2 ADC Voltage Reference

The reference voltage for the ADC (V_{REF}) indicates the conversion range for the ADC. Single ended channels that exceed V_{REF} will result in codes close to 0x3FF. V_{REF} can be selected as either AV_{CC}, internal 2.56V reference, or external AREF pin.

AV_{CC} is connected to the ADC through a passive switch. The internal 2.56V reference is generated from the internal bandgap reference (V_{BG}) through an internal amplifier. In either case, the external AREF pin is directly connected to the ADC, and the reference voltage can be made more immune to noise by connecting a capacitor between the AREF pin and ground. V_{REF} can also be measured at the AREF pin with a high impedance voltmeter. Note that V_{REF} is a high impudent source, and only a capacitive load should be connected in a system.

If the user has a fixed voltage source connected to the AREF pin, the user may not use the other reference voltage options in the application, as they will be shorted to the external voltage. If no external voltage is applied to the AREF pin, the user may switch between AV_{CC} and 2.56V as reference selection. The first ADC conversion result after switching reference voltage source may be inaccurate, and the user is advised to discard this result.

If differential channels are used, the selected reference should not be closer to AV_{CC} than indicated in Table 29-5 on page 384.

24.6 Temperature Sensor

The ATmega16U4/ATmega32U4 includes an on-chip temperature sensor, whose value can be read through the A/D Converter.

The temperature measurement is based on an on-chip temperature sensor that is coupled to a single ended ADC input. MUX[5..0] bits in ADMUX register enables the temperature sensor. The internal 2.56V voltage reference must also be selected for the ADC voltage reference source in the temperature sensor measurement. When the temperature sensor is enabled, the ADC converter can be used in single conversion mode to measure the voltage over the temperature sensor.
The temperature sensor and its internal driver are enabled when ADMUX value selects the temperature sensor as ADC input. The propagation delay of this driver is approximatively $2\mu S$. Therefore two successive conversions are required. The correct temperature measurement will be the second one.

One can also reduce this timing to one conversion by setting the ADMUX during the previous conversion. Indeed the ADMUX can be programmed to select the temperature sensor just after the beginning of the previous conversion start event and then the driver will be enabled $2 \mu S$ before sampling and hold phase of temperature sensor measurement.

24.6.1 Sensor Calibration

The sensor initial tolerance is large (+/-10°C), but its characteristic is linear. Thus, if the application requires accuracy, the firmware must include a calibration stage to use the sensor for direct temperature measurement.

Another application of this sensor may concern the Internal Calibrated RC Oscillator, whose the frequency can be adjusted by the user through the OSCCAL register (see Section 6.5.1 "Oscillator Calibration Register – OSCCAL" on page 32). During the production, a calibration is done at two temperatures (+25°C and +85°C, with a tolerance of +/-10°C). At each temperature, the temperature sensor value $T_i$ is measured and stored in EEPROM memory, and the OSCCAL calibration value $O_i$ (i.e. the value that should be set in OSCCAL register at this temperature to have an accurate 8MHz output) is stored in another memory zone.

Thanks to these four values and the linear characteristics of the temperature sensor and Internal RC Oscillator, firmware can easily recalibrate the RC Oscillator on-the-go in function of the temperature sensor measure (an application note describes the operation):

Figure 24-8. Linear Characterization of OSCCAL in function of $T^\circ$ measurement from ADC

Notes:
1. The temperature sensor calibration values cannot be used to do accurate temperature measurements since the calibration temperature during production is not accurate (+/- 10°C)
2. Be aware that if EESAVE fuse is left unprogrammed, any chip erase operation will clear the temperature sensor calibration values contained in EEPROM memory.
3. Accuracy results after a software recalibration of OSCCAL in function of $T^\circ$ will be given when device will be fully characterized.
24.7 ADC Noise Canceler

The ADC features a noise canceler that enables conversion during sleep mode to reduce noise induced from the CPU core and other I/O peripherals. The noise canceler can be used with ADC Noise Reduction and Idle mode. To make use of this feature, the following procedure should be used:

a. Make sure that the ADC is enabled and is not busy converting. Single Conversion mode must be selected and the ADC conversion complete interrupt must be enabled.

b. Enter ADC Noise Reduction mode (or Idle mode). The ADC will start a conversion once the CPU has been halted.

c. If no other interrupts occur before the ADC conversion completes, the ADC interrupt will wake up the CPU and execute the ADC Conversion Complete interrupt routine. If another interrupt wakes up the CPU before the ADC conversion is complete, that interrupt will be executed, and an ADC Conversion Complete interrupt request will be generated when the ADC conversion completes. The CPU will remain in active mode until a new sleep command is executed.

Note that the ADC will not be automatically turned off when entering other sleep modes than Idle mode and ADC Noise Reduction mode. The user is advised to write zero to ADEN before entering such sleep modes to avoid excessive power consumption.

If the ADC is enabled in such sleep modes and the user wants to perform differential conversions, the user is advised to switch the ADC off and on after waking up from sleep to prompt an extended conversion to get a valid result.

24.7.1 Analog Input Circuitry

The analog input circuitry for single ended channels is illustrated in Figure 24-9. An analog source applied to ADCn is subjected to the pin capacitance and input leakage of that pin, regardless of whether that channel is selected as input for the ADC. When the channel is selected, the source must drive the S/H capacitor through the series resistance (combined resistance in the input path).

The ADC is optimized for analog signals with an output impedance of approximately 10 kΩ or less. If such a source is used, the sampling time will be negligible. If a source with higher impedance is used, the sampling time will depend on how long time the source needs to charge the S/H capacitor, with can vary widely. The user is recommended to only use low impedance sources with slowly varying signals, since this minimizes the required charge transfer to the S/H capacitor.

If differential gain channels are used, the input circuitry looks somewhat different, although source impedances of a few hundred kΩ or less is recommended.

Signal components higher than the Nyquist frequency (f_{ADC}/2) should not be present for either kind of channels, to avoid distortion from unpredictable signal convolution. The user is advised to remove high frequency components with a low-pass filter before applying the signals as inputs to the ADC.
24.7.2 Analog Noise Canceling Techniques

Digital circuitry inside and outside the device generates EMI which might affect the accuracy of analog measurements. If conversion accuracy is critical, the noise level can be reduced by applying the following techniques:

a. Keep analog signal paths as short as possible. Make sure analog tracks run over the analog ground plane, and keep them well away from high-speed switching digital tracks.

b. The $V_{CC}$ pin on the device should be connected to the digital $V_{CC}$ supply voltage via an LC network as shown in Figure 24-10.

c. Use the ADC noise canceler function to reduce induced noise from the CPU.

d. If any ADC port pins are used as digital outputs, it is essential that these do not switch while a conversion is in progress.

Figure 24-10. ADC Power Connections

Note: The same circuitry should be used for AVCC filtering on the ADC8-ADC13 side.
24.7.3 Offset Compensation Schemes
The gain stage has a built-in offset cancellation circuitry that nulls the offset of differential measurements as much as possible. The remaining offset in the analog path can be measured directly by selecting the same channel for both differential inputs. This offset residue can be then subtracted in software from the measurement results. Using this kind of software based offset correction, offset on any channel can be reduced below one LSB.

24.7.4 ADC Accuracy Definitions
An n-bit single-ended ADC converts a voltage linearly between GND and $V_{REF}$ in $2^n$ steps (LSBs). The lowest code is read as 0, and the highest code is read as $2^n-1$.

Several parameters describe the deviation from the ideal behavior:

- Offset: The deviation of the first transition (0x000 to 0x001) compared to the ideal transition (at 0.5 LSB). Ideal value: 0 LSB.

**Figure 24-11. Offset Error**

- Gain Error: After adjusting for offset, the Gain Error is found as the deviation of the last transition (0x3FE to 0x3FF) compared to the ideal transition (at 1.5 LSB below maximum). Ideal value: 0 LSB
Figure 24-12. Gain Error

- Integral Non-linearity (INL): After adjusting for offset and gain error, the INL is the maximum deviation of an actual transition compared to an ideal transition for any code. Ideal value: 0 LSB.

Figure 24-13. Integral Non-linearity (INL)

- Differential Non-linearity (DNL): The maximum deviation of the actual code width (the interval between two adjacent transitions) from the ideal code width (1 LSB). Ideal value: 0 LSB.
24.8 ADC Conversion Result

After the conversion is complete (ADIF is high), the conversion result can be found in the ADC Result Registers (ADCL, ADCH).

For single ended conversion, the result is:

\[
ADC = \frac{V_{IN} \cdot 1023}{V_{REF}}
\]

where \( V_{IN} \) is the voltage on the selected input pin and \( V_{REF} \) the selected voltage reference (see Table 24-3 on page 307 and Table 24-4 on page 308). 0x000 represents analog ground, and 0x3FF represents the selected reference voltage minus one LSB.

If differential channels are used, the result is:

\[
ADC = \frac{(V_{POS} - V_{NEG}) \cdot GAIN \cdot 512}{V_{REF}}
\]

where \( V_{POS} \) is the voltage on the positive input pin, \( V_{NEG} \) the voltage on the negative input pin, GAIN the selected gain factor and \( V_{REF} \) the selected voltage reference. The result is presented in two’s complement form, from 0x200 (-512d) through 0x1FF (+511d). Note that if the user wants to perform a quick polarity check of the result, it is sufficient to read the MSB of the result (ADC9 in ADCH). If the bit is one, the result is negative, and if this bit is zero, the result is positive. Figure 24-15 shows the decoding of the differential input range.

Table 82 shows the resulting output codes if the differential input channel pair (ADCn - ADCm) is selected with a reference voltage of \( V_{REF} \).
Figure 24-15. Differential Measurement Range

Table 24-2. Correlation Between Input Voltage and Output Codes

<table>
<thead>
<tr>
<th>$V_{ADCn}$</th>
<th>Read code</th>
<th>Corresponding decimal value</th>
</tr>
</thead>
<tbody>
<tr>
<td>$V_{ADCm} + V_{REF}/GAIN$</td>
<td>0x1FF</td>
<td>511</td>
</tr>
<tr>
<td>$V_{ADCm} + 0.999 V_{REF}/GAIN$</td>
<td>0x1FF</td>
<td>511</td>
</tr>
<tr>
<td>$V_{ADCm} + 0.998 V_{REF}/GAIN$</td>
<td>0x1FE</td>
<td>510</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>$V_{ADCm} + 0.001 V_{REF}/GAIN$</td>
<td>0x001</td>
<td>1</td>
</tr>
<tr>
<td>$V_{ADCm}$</td>
<td>0x000</td>
<td>0</td>
</tr>
<tr>
<td>$V_{ADCm} - 0.001 V_{REF}/GAIN$</td>
<td>0x3FF</td>
<td>-1</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>$V_{ADCm} - 0.999 V_{REF}/GAIN$</td>
<td>0x201</td>
<td>-511</td>
</tr>
<tr>
<td>$V_{ADCm} - V_{REF}/GAIN$</td>
<td>0x200</td>
<td>-512</td>
</tr>
</tbody>
</table>

Example 1:
- $ADMUX = 0xE9$, $MUX5 = 0$ (ADC1 - ADC0, 10x gain, 2.56V reference, left adjusted result)
- Voltage on ADC1 is 300 mV, voltage on ADC0 is 500 mV.
- $ADCR = 512 \times 10 \times (300 - 500) / 2560 = -400 = 0x270$
– ADCL will thus read 0x00, and ADCH will read 0x9C.
  Writing zero to ADLAR right adjusts the result: ADCL = 0x70, ADCH = 0x02.

Example 2:
– ADMUX = 0xF0, MUX5 = 0 (ADC0 - ADC1, 1x gain, 2.56V reference, left adjusted result)
– Voltage on ADC0 is 300 mV, voltage on ADC1 is 500 mV.
– ADCR = 512 * 1 * (300 - 500) / 2560 = -41 = 0x029.
– ADCL will thus read 0x40, and ADCH will read 0x0A.
  Writing zero to ADLAR right adjusts the result: ADCL = 0x00, ADCH = 0x29.

24.9 ADC Register Description

24.9.1 ADC Multiplexer Selection Register – ADMUX

<table>
<thead>
<tr>
<th>Bit</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
<th>ADMUX</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read/Write</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td></td>
</tr>
<tr>
<td>Initial Value</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

• Bit 7:6 – REFS1:0: Reference Selection Bits
  These bits select the voltage reference for the ADC, as shown in Table 24-3. If these bits are changed during a conversion, the change will not go in effect until this conversion is complete (ADIF in ADCSRA is set). The internal voltage reference options may not be used if an external reference voltage is being applied to the AREF pin.

<table>
<thead>
<tr>
<th>REFS1</th>
<th>REFS0</th>
<th>Voltage Reference Selection</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>AREF, Internal Vref turned off</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>AVcc with external capacitor on AREF pin</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>Reserved</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>Internal 2.56V Voltage Reference with external capacitor on AREF pin</td>
</tr>
</tbody>
</table>

• Bit 5 – ADLAR: ADC Left Adjust Result
  The ADLAR bit affects the presentation of the ADC conversion result in the ADC Data Register. Write one to ADLAR to left adjust the result. Otherwise, the result is right adjusted. Changing the ADLAR bit will affect the ADC Data Register immediately, regardless of any ongoing conversions. For a complete description of this bit, see “The ADC Data Register – ADCL and ADCH” on page 311.

• Bits 4:0 – MUX4:0: Analog Channel Selection Bits
  The value of these bits selects which combination of analog inputs are connected to the ADC. These bits also select the gain for the differential channels. See Table 24-4 for details. If these bits are changed during a conversion, the change will not go in effect until this conversion is complete (ADIF in ADCSRA is set).
### Table 24-4. Input Channel and Gain Selections

<table>
<thead>
<tr>
<th>MUX5..0&lt;sup&gt;(1)&lt;/sup&gt;</th>
<th>Single Ended Input</th>
<th>Positive Differential Input</th>
<th>Negative Differential Input</th>
<th>Gain</th>
</tr>
</thead>
<tbody>
<tr>
<td>000000</td>
<td>ADC0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>000001</td>
<td>ADC1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>00010</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td></td>
</tr>
<tr>
<td>000011</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>000100</td>
<td>ADC4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>000101</td>
<td>ADC5</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>000110</td>
<td>ADC6</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>000111</td>
<td>ADC7</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>00100</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td></td>
</tr>
<tr>
<td>001001</td>
<td>ADC1</td>
<td>ADC0</td>
<td>10x</td>
<td></td>
</tr>
<tr>
<td>001010</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td></td>
</tr>
<tr>
<td>001011</td>
<td>ADC1</td>
<td>ADC0</td>
<td>200x</td>
<td></td>
</tr>
<tr>
<td>001100</td>
<td>N/A</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>001101</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>001110</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>001111</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>010000</td>
<td>ADC0</td>
<td>ADC1</td>
<td>1x</td>
<td></td>
</tr>
</tbody>
</table>
Table 24-4.  Input Channel and Gain Selections (Continued)

<table>
<thead>
<tr>
<th>MUX5..0&lt;sup&gt;(1)&lt;/sup&gt;</th>
<th>Single Ended Input</th>
<th>Positive Differential Input</th>
<th>Negative Differential Input</th>
<th>Gain</th>
</tr>
</thead>
<tbody>
<tr>
<td>010000</td>
<td>N/A</td>
<td>ADC4</td>
<td>ADC1</td>
<td>1x</td>
</tr>
<tr>
<td>010010</td>
<td></td>
<td>ADC5</td>
<td>ADC1</td>
<td>1x</td>
</tr>
<tr>
<td>010011</td>
<td>N/A</td>
<td>ADC6</td>
<td>ADC1</td>
<td>1x</td>
</tr>
<tr>
<td>010100</td>
<td></td>
<td>ADC7</td>
<td>ADC1</td>
<td>1x</td>
</tr>
<tr>
<td>010101</td>
<td></td>
<td>N/A</td>
<td>1.1V (V&lt;sub&gt;Band Gap&lt;/sub&gt;)</td>
<td></td>
</tr>
<tr>
<td>011000</td>
<td></td>
<td>ADC8</td>
<td>ADC0</td>
<td>40x</td>
</tr>
<tr>
<td>011001</td>
<td></td>
<td>ADC9</td>
<td>ADC0</td>
<td>40x</td>
</tr>
<tr>
<td>011010</td>
<td></td>
<td>ADC10</td>
<td>ADC0</td>
<td>40x</td>
</tr>
<tr>
<td>011011</td>
<td></td>
<td>ADC11</td>
<td>ADC0</td>
<td>40x</td>
</tr>
<tr>
<td>011100</td>
<td></td>
<td>ADC12</td>
<td>ADC0</td>
<td>40x</td>
</tr>
<tr>
<td>011101</td>
<td></td>
<td>ADC13</td>
<td>ADC0</td>
<td>40x</td>
</tr>
<tr>
<td>011110</td>
<td></td>
<td>N/A</td>
<td>ADC0</td>
<td>40x</td>
</tr>
<tr>
<td>011111</td>
<td></td>
<td>Temperature Sensor</td>
<td>ADC0</td>
<td>40x</td>
</tr>
</tbody>
</table>

1. MUX5..0<sup>(1)</sup> is a bit field used to select input channels and gains for the ADC.
Table 24-4. Input Channel and Gain Selections (Continued)

<table>
<thead>
<tr>
<th>MUX5..0(1)</th>
<th>Single Ended Input</th>
<th>Positive Differential Input</th>
<th>Negative Differential Input</th>
<th>Gain</th>
</tr>
</thead>
<tbody>
<tr>
<td>110100</td>
<td></td>
<td>ADC4</td>
<td>ADC1</td>
<td>40x</td>
</tr>
<tr>
<td>110101</td>
<td></td>
<td>ADC5</td>
<td>ADC1</td>
<td>40x</td>
</tr>
<tr>
<td>110110</td>
<td></td>
<td>ADC6</td>
<td>ADC1</td>
<td>40x</td>
</tr>
<tr>
<td>110111</td>
<td></td>
<td>ADC7</td>
<td>ADC1</td>
<td>40x</td>
</tr>
<tr>
<td>111000</td>
<td>N/A</td>
<td>ADC4</td>
<td>ADC0</td>
<td>200x</td>
</tr>
<tr>
<td>111001</td>
<td></td>
<td>ADC5</td>
<td>ADC0</td>
<td>200x</td>
</tr>
<tr>
<td>111010</td>
<td></td>
<td>ADC6</td>
<td>ADC0</td>
<td>200x</td>
</tr>
<tr>
<td>111011</td>
<td></td>
<td>ADC7</td>
<td>ADC0</td>
<td>200x</td>
</tr>
<tr>
<td>111100</td>
<td></td>
<td>ADC4</td>
<td>ADC1</td>
<td>200x</td>
</tr>
<tr>
<td>111101</td>
<td></td>
<td>ADC5</td>
<td>ADC1</td>
<td>200x</td>
</tr>
<tr>
<td>111110</td>
<td></td>
<td>ADC6</td>
<td>ADC1</td>
<td>200x</td>
</tr>
<tr>
<td>111111</td>
<td></td>
<td>ADC7</td>
<td>ADC1</td>
<td>200x</td>
</tr>
</tbody>
</table>

Note: 1. MUX5 bit make part of ADCSRB register

24.9.2 ADC Control and Status Register A – ADCSRA

<table>
<thead>
<tr>
<th>Bit</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
<th>ADCSRA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read/Write</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td></td>
</tr>
<tr>
<td>Initial Value</td>
<td>0 0 0 0 0 0 0 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **Bit 7 – ADEN: ADC Enable**
  Writing this bit to one enables the ADC. By writing it to zero, the ADC is turned off. Turning the ADC off while a conversion is in progress, will terminate this conversion.

- **Bit 6 – ADSC: ADC Start Conversion**
  In Single Conversion mode, write this bit to one to start each conversion. In Free Running mode, write this bit to one to start the first conversion. The first conversion after ADSC has been written after the ADC has been enabled, or if ADSC is written at the same time as the ADC is enabled, will take 25 ADC clock cycles instead of the normal 13. This first conversion performs initialization of the ADC.
  ADCS will read as one as long as a conversion is in progress. When the conversion is complete, it returns to zero. Writing zero to this bit has no effect.

- **Bit 5 – ADATE: ADC Auto Trigger Enable**
  When this bit is written to one, Auto Triggering of the ADC is enabled. The ADC will start a conversion on a positive edge of the selected trigger signal. The trigger source is selected by setting the ADC Trigger Select bits, ADTS in ADCSRB.

- **Bit 4 – ADIF: ADC Interrupt Flag**
  This bit is set when an ADC conversion completes and the Data Registers are updated. The ADC Conversion Complete Interrupt is executed if the ADIE bit and the I-bit in SREG are set. ADIF is cleared by hardware when executing the corresponding interrupt handling vector. Alternatively, ADIF is cleared by writing a logical one to the flag. Beware that if doing a Read-Modify-
Write on ADCSRA, a pending interrupt can be disabled. This also applies if the SBI and CBI instructions are used.

- **Bit 3 – ADIE: ADC Interrupt Enable**
  When this bit is written to one and the I-bit in SREG is set, the ADC Conversion Complete Interrupt is activated.

- **Bits 2:0 – ADPS2:0: ADC Prescaler Select Bits**
  These bits determine the division factor between the XTAL frequency and the input clock to the ADC.

### Table 24-5. ADC Prescaler Selections

<table>
<thead>
<tr>
<th>ADPS2</th>
<th>ADPS1</th>
<th>ADPS0</th>
<th>Division Factor</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>8</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>16</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>32</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>64</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>128</td>
</tr>
</tbody>
</table>

#### 24.9.3 The ADC Data Register – ADCL and ADCH

##### 24.9.3.1 ADLAR = 0

<table>
<thead>
<tr>
<th>Bit</th>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ADCL</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ADCH</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Bit</td>
<td>7</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Read/Write</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
</tr>
<tr>
<td>Initial Value</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

##### 24.9.3.2 ADLAR = 1

<table>
<thead>
<tr>
<th>Bit</th>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ADCL</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ADCH</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Bit</td>
<td>7</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Read/Write</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
</tr>
<tr>
<td>Initial Value</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

When an ADC conversion is complete, the result is found in these two registers. If differential channels are used, the result is presented in two’s complement form.

When ADCL is read, the ADC Data Register is not updated until ADCH is read. Consequently, if the result is left adjusted and no more than 8-bit precision (7 bit + sign bit for differential input
channels) is required, it is sufficient to read ADCH. Otherwise, ADCL must be read first, then ADCH.

The ADLAR bit in ADMUX, and the MUXn bits in ADMUX affect the way the result is read from the registers. If ADLAR is set, the result is left adjusted. If ADLAR is cleared (default), the result is right adjusted.

- **ADC9:0: ADC Conversion Result**
  These bits represent the result from the conversion, as detailed in “ADC Conversion Result” on page 305.

### 24.9.4 ADC Control and Status Register B – ADCSRB

<table>
<thead>
<tr>
<th>Bit</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
<th>ADCSRB</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read/Write</td>
<td>R/W</td>
<td>R/W</td>
<td>R</td>
<td>R</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td></td>
</tr>
<tr>
<td>Initial Value</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

- **Bit 7 – ADHSM: ADC High Speed Mode**
  Writing this bit to one enables the ADC High Speed mode. This mode enables higher conversion rate at the expense of higher power consumption.

- **Bit 5 – MUX5: Analog Channel Additional Selection Bits**
  This bit make part of MUX5:0 bits of ADRCSRB and ADMUX register, that select the combination of analog inputs connected to the ADC (including differential amplifier configuration).

- **Bit 3:0 – ADTS3:0: ADC Auto Trigger Source**
  If ADATE in ADCSRA is written to one, the value of these bits selects which source will trigger an ADC conversion. If ADATE is cleared, the ADTS3:0 settings will have no effect. A conversion will be triggered by the rising edge of the selected interrupt flag. Note that switching from a trigger source that is cleared to a trigger source that is set, will generate a positive edge on the trigger signal. If ADEN in ADCSRA is set, this will start a conversion. Switching to Free Running mode (ADTS[3:0]=0) will not cause a trigger event, even if the ADC Interrupt Flag is set.

#### Table 24-6. ADC Auto Trigger Source Selections

<table>
<thead>
<tr>
<th>ADTS3</th>
<th>ADTS2</th>
<th>ADTS1</th>
<th>ADTS0</th>
<th>Trigger Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>Free Running mode</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>Analog Comparator</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>External Interrupt Request 0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>Timer/Counter0 Compare Match</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>Timer/Counter0 Overflow</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>Timer/Counter1 Compare Match B</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>Timer/Counter1 Overflow</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>Timer/Counter1 Capture Event</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>Timer/Counter4 Overflow</td>
</tr>
</tbody>
</table>
Table 24-6. ADC Auto Trigger Source Selections (Continued)

<table>
<thead>
<tr>
<th>ADTS3</th>
<th>ADTS2</th>
<th>ADTS1</th>
<th>ADTS0</th>
<th>Trigger Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>Timer/Counter4 Compare Match A</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>Timer/Counter4 Compare Match B</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>Timer/Counter4 Compare Match D</td>
</tr>
</tbody>
</table>

24.9.5 Digital Input Disable Register 0 – DIDR0

<table>
<thead>
<tr>
<th>Bit</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
<th>DIDR0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read/Write</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td></td>
</tr>
<tr>
<td>Initial Value</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

- Bit 7:4, 1:0 – ADC7D..4D - ADC1D..0D: ADC7:4 - ADC1:0 Digital Input Disable
  When this bit is written logic one, the digital input buffer on the corresponding ADC pin is disabled. The corresponding PIN Register bit will always read as zero when this bit is set. When an analog signal is applied to the ADC7..4 / ADC1..0 pin and the digital input from this pin is not needed, this bit should be written logic one to reduce power consumption in the digital input buffer.

24.9.6 Digital Input Disable Register 2 – DIDR2

<table>
<thead>
<tr>
<th>Bit</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
<th>DIDR2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read/Write</td>
<td>-</td>
<td>-</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td></td>
</tr>
<tr>
<td>Initial Value</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

- Bit 5:0 – ADC13D..ADC8D: ADC13:8 Digital Input Disable
  When this bit is written logic one, the digital input buffer on the corresponding ADC pin is disabled. The corresponding PIN Register bit will always read as zero when this bit is set. When an analog signal is applied to the ADC13..8 pin and the digital input from this pin is not needed, this bit should be written logic one to reduce power consumption in the digital input buffer.
25. JTAG Interface and On-chip Debug System

25.0.1 Features

- JTAG (IEEE std. 1149.1 Compliant) Interface
- Boundary-scan Capabilities According to the IEEE std. 1149.1 (JTAG) Standard
- Debugger Access to:
  - All Internal Peripheral Units
  - Internal and External RAM
  - The Internal Register File
  - Program Counter
  - EEPROM and Flash Memories
- Extensive On-chip Debug Support for Break Conditions, Including
  - AVR Break Instruction
  - Break on Change of Program Memory Flow
  - Single Step Break
  - Program Memory Break Points on Single Address or Address Range
  - Data Memory Break Points on Single Address or Address Range
- Programming of Flash, EEPROM, Fuses, and Lock Bits through the JTAG Interface
- On-chip Debugging Supported by AVR Studio®

25.1 Overview

The AVR IEEE std. 1149.1 compliant JTAG interface can be used for

- Testing PCBs by using the JTAG Boundary-scan capability
- Programming the non-volatile memories, Fuses and Lock bits
- On-chip debugging

A brief description is given in the following sections. Detailed descriptions for Programming via the JTAG interface, and using the Boundary-scan Chain can be found in the sections “Programming via the JTAG Interface” on page 365 and “IEEE 1149.1 (JTAG) Boundary-scan” on page 320, respectively. The On-chip Debug support is considered being private JTAG instructions, and distributed within ATMEG and to selected third party vendors only.

Figure 25-1 shows a block diagram of the JTAG interface and the On-chip Debug system. The TAP Controller is a state machine controlled by the TCK and TMS signals. The TAP Controller selects either the JTAG Instruction Register or one of several Data Registers as the scan chain (Shift Register) between the TDI – input and TDO – output. The Instruction Register holds JTAG instructions controlling the behavior of a Data Register.

The ID-Register, Bypass Register, and the Boundary-scan Chain are the Data Registers used for board-level testing. The JTAG Programming Interface (actually consisting of several physical and virtual Data Registers) is used for serial programming via the JTAG interface. The Internal Scan Chain and Break Point Scan Chain are used for On-chip debugging only.

25.2 Test Access Port – TAP

The JTAG interface is accessed through four of the AVR’s pins. In JTAG terminology, these pins constitute the Test Access Port – TAP. These pins are:

- TMS: Test mode select. This pin is used for navigating through the TAP-controller state machine.
- TCK: Test Clock. JTAG operation is synchronous to TCK.
- TDI: Test Data In. Serial input data to be shifted in to the Instruction Register or Data Register (Scan Chains).
- TDO: Test Data Out. Serial output data from Instruction Register or Data Register.
The IEEE std. 1149.1 also specifies an optional TAP signal; TRST – Test ReSeT – which is not provided.

When the JTAGEN Fuse is unprogrammed, these four TAP pins are normal port pins, and the
TAP controller is in reset. When programmed, the input TAP signals are internally pulled high
and the JTAG is enabled for Boundary-scan and programming. The device is shipped with this
fuse programmed.

For the On-chip Debug system, in addition to the JTAG interface pins, the RESET pin is moni-
tored by the debugger to be able to detect external reset sources. The debugger can also pull
the RESET pin low to reset the whole system, assuming only open collectors on the reset line
are used in the application.

**Figure 25-1.** Block Diagram
Figure 25-2. TAP Controller State Diagram

25.3 TAP Controller

The TAP controller is a 16-state finite state machine that controls the operation of the Boundary-scan circuitry, JTAG programming circuitry, or On-chip Debug system. The state transitions depicted in Figure 25-2 depend on the signal present on TMS (shown adjacent to each state transition) at the time of the rising edge at TCK. The initial state after a Power-on Reset is Test-Logic-Reset.

As a definition in this document, the LSB is shifted in and out first for all Shift Registers.

Assuming Run-Test/Idle is the present state, a typical scenario for using the JTAG interface is:

- At the TMS input, apply the sequence 1, 1, 0, 0 at the rising edges of TCK to enter the Shift Instruction Register – Shift-IR state. While in this state, shift the four bits of the JTAG instructions into the JTAG Instruction Register from the TDI input at the rising edge of TCK. The TMS input must be held low during input of the 3 LSBs in order to remain in the Shift-IR state. The MSB of the instruction is shifted in when this state is left by setting TMS high.

While the instruction is shifted in from the TDI pin, the captured IR-state 0x01 is shifted out on the TDO pin. The JTAG Instruction selects a particular Data Register as path between TDI and TDO and controls the circuitry surrounding the selected Data Register.
• Apply the TMS sequence 1, 1, 0 to re-enter the Run-Test/Idle state. The instruction is latched onto the parallel output from the Shift Register path in the Update-IR state. The Exit-IR, Pause-IR, and Exit2-IR states are only used for navigating the state machine.

• At the TMS input, apply the sequence 1, 0, 0 at the rising edges of TCK to enter the Shift Data Register – Shift-DR state. While in this state, upload the selected Data Register (selected by the present JTAG instruction in the JTAG Instruction Register) from the TDI input at the rising edge of TCK. In order to remain in the Shift-DR state, the TMS input must be held low during input of all bits except the MSB. The MSB of the data is shifted in when this state is left by setting TMS high. While the Data Register is shifted in from the TDI pin, the parallel inputs to the Data Register captured in the Capture-DR state is shifted out on the TDO pin.

• Apply the TMS sequence 1, 1, 0 to re-enter the Run-Test/Idle state. If the selected Data Register has a latched parallel-output, the latching takes place in the Update-DR state. The Exit-DR, Pause-DR, and Exit2-DR states are only used for navigating the state machine.

As shown in the state diagram, the Run-Test/Idle state need not be entered between selecting JTAG instruction and using Data Registers, and some JTAG instructions may select certain functions to be performed in the Run-Test/Idle, making it unsuitable as an Idle state.

Note: Independent of the initial state of the TAP Controller, the Test-Logic-Reset state can always be entered by holding TMS high for five TCK clock periods.

For detailed information on the JTAG specification, refer to the literature listed in “Bibliography” on page 319.

25.4 Using the Boundary-scan Chain

A complete description of the Boundary-scan capabilities are given in the section “IEEE 1149.1 (JTAG) Boundary-scan” on page 320.

25.5 Using the On-chip Debug System

As shown in Figure 25-1, the hardware support for On-chip Debugging consists mainly of

• A scan chain on the interface between the internal AVR CPU and the internal peripheral units.

• Break Point unit.

• Communication interface between the CPU and JTAG system.

All read or modify/write operations needed for implementing the Debugger are done by applying AVR instructions via the internal AVR CPU Scan Chain. The CPU sends the result to an I/O memory mapped location which is part of the communication interface between the CPU and the JTAG system.

The Break Point Unit implements Break on Change of Program Flow, Single Step Break, two Program Memory Break Points, and two combined Break Points. Together, the four Break Points can be configured as either:

• 4 single Program Memory Break Points.

• 3 Single Program Memory Break Point + 1 single Data Memory Break Point.

• 2 single Program Memory Break Points + 2 single Data Memory Break Points.

• 2 single Program Memory Break Points + 1 Program Memory Break Point with mask (“range Break Point”).
• 2 single Program Memory Break Points + 1 Data Memory Break Point with mask ("range Break Point").

A debugger, like the AVR Studio, may however use one or more of these resources for its internal purpose, leaving less flexibility to the end-user.

A list of the On-chip Debug specific JTAG instructions is given in “On-chip Debug Specific JTAG Instructions” on page 318.

The JTAGEN Fuse must be programmed to enable the JTAG Test Access Port. In addition, the OCDEN Fuse must be programmed and no Lock bits must be set for the On-chip debug system to work. As a security feature, the On-chip debug system is disabled when either of the LB1 or LB2 Lock bits are set. Otherwise, the On-chip debug system would have provided a back-door into a secured device.

The AVR Studio enables the user to fully control execution of programs on an AVR device with On-chip Debug capability, AVR In-Circuit Emulator, or the built-in AVR Instruction Set Simulator. AVR Studio® supports source level execution of Assembly programs assembled with ATMEL Corporation’s AVR Assembler and C programs compiled with third party vendors’ compilers.

AVR Studio runs under Microsoft® Windows® 95/98/2000 and Microsoft Windows NT®.

For a full description of the AVR Studio, please refer to the AVR Studio User Guide. Only highlights are presented in this document.

All necessary execution commands are available in AVR Studio, both on source level and on disassembly level. The user can execute the program, single step through the code either by tracing into or stepping over functions, step out of functions, place the cursor on a statement and execute until the statement is reached, stop the execution, and reset the execution target. In addition, the user can have an unlimited number of code Break Points (using the BREAK instruction) and up to two data memory Break Points, alternatively combined as a mask (range) Break Point.

25.6 On-chip Debug Specific JTAG Instructions

The On-chip debug support is considered being private JTAG instructions, and distributed within ATMEL and to selected third party vendors only. Instruction opcodes are listed for reference.

25.6.1 PRIVATE0; 0x8

Private JTAG instruction for accessing On-chip debug system.

25.6.2 PRIVATE1; 0x9

Private JTAG instruction for accessing On-chip debug system.

25.6.3 PRIVATE2; 0xA

Private JTAG instruction for accessing On-chip debug system.

25.6.4 PRIVATE3; 0xB

Private JTAG instruction for accessing On-chip debug system.
25.7 On-chip Debug Related Register in I/O Memory

25.7.1 On-chip Debug Register – OCDR

<table>
<thead>
<tr>
<th>Bit</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read/Write</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
</tr>
<tr>
<td>Initial Value</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

The OCDR Register provides a communication channel from the running program in the microcontroller to the debugger. The CPU can transfer a byte to the debugger by writing to this location. At the same time, an internal flag; I/O Debug Register Dirty – IDRD – is set to indicate to the debugger that the register has been written. When the CPU reads the OCDR Register the 7 LSB will be from the OCDR Register, while the MSB is the IDRD bit. The debugger clears the IDRD bit when it has read the information.

In some AVR devices, this register is shared with a standard I/O location. In this case, the OCDR Register can only be accessed if the OCDEN Fuse is programmed, and the debugger enables access to the OCDR Register. In all other cases, the standard I/O location is accessed.

Refer to the debugger documentation for further information on how to use this register.

25.8 Using the JTAG Programming Capabilities

Programming of AVR parts via JTAG is performed via the 4-pin JTAG port, TCK, TMS, TDI, and TDO. These are the only pins that need to be controlled/observed to perform JTAG programming (in addition to power pins). It is not required to apply 12V externally. The JTAGEN Fuse must be programmed and the JTD bit in the MCUCR Register must be cleared to enable the JTAG Test Access Port.

The JTAG programming capability supports:

- Flash programming and verifying.
- EEPROM programming and verifying.
- Fuse programming and verifying.
- Lock bit programming and verifying.

The Lock bit security is exactly as in parallel programming mode. If the Lock bits LB1 or LB2 are programmed, the OCDEN Fuse cannot be programmed unless first doing a chip erase. This is a security feature that ensures no back-door exists for reading out the content of a secured device.

The details on programming through the JTAG interface and programming specific JTAG instructions are given in the section “Programming via the JTAG Interface” on page 365.

25.9 Bibliography

For more information about general Boundary-scan, the following literature can be consulted:

26. IEEE 1149.1 (JTAG) Boundary-scan

26.1 Features

- JTAG (IEEE std. 1149.1 compliant) Interface
- Boundary-scan Capabilities According to the JTAG Standard
- Full Scan of all Port Functions as well as Analog Circuitry having Off-chip Connections
- Supports the Optional IDCODE Instruction
- Additional Public AVR_RESET Instruction to Reset the AVR

26.2 System Overview

The Boundary-scan chain has the capability of driving and observing the logic levels on the digital I/O pins, as well as the boundary between digital and analog logic for analog circuitry having off-chip connections. At system level, all ICs having JTAG capabilities are connected serially by the TDI/TDO signals to form a long Shift Register. An external controller sets up the devices to drive values at their output pins, and observe the input values received from other devices. The controller compares the received data with the expected result. In this way, Boundary-scan provides a mechanism for testing interconnections and integrity of components on Printed Circuits Boards by using the four TAP signals only.

The four IEEE 1149.1 defined mandatory JTAG instructions IDCODE, BYPASS, SAMPLE/PRELOAD, and EXTEST, as well as the AVR specific public JTAG instruction AVR_RESET can be used for testing the Printed Circuit Board. Initial scanning of the Data Register path will show the ID-Code of the device, since IDCODE is the default JTAG instruction. It may be desirable to have the AVR device in reset during test mode. If not reset, inputs to the device may be determined by the scan operations, and the internal software may be in an undetermined state when exiting the test mode. Entering reset, the outputs of any port pin will instantly enter the high impedance state, making the HIGHZ instruction redundant. If needed, the BYPASS instruction can be issued to make the shortest possible scan chain through the device. The device can be set in the reset state either by pulling the external RESET pin low, or issuing the AVR_RESET instruction with appropriate setting of the Reset Data Register.

The EXTEST instruction is used for sampling external pins and loading output pins with data. The data from the output latch will be driven out on the pins as soon as the EXTEST instruction is loaded into the JTAG IR-Register. Therefore, the SAMPLE/PRELOAD should also be used for setting initial values to the scan ring, to avoid damaging the board when issuing the EXTEST instruction for the first time. SAMPLE/PRELOAD can also be used for taking a snapshot of the external pins during normal operation of the part.

The JTAGEN Fuse must be programmed and the JTD bit in the I/O Register MCUCR must be cleared to enable the JTAG Test Access Port.

When using the JTAG interface for Boundary-scan, using a JTAG TCK clock frequency higher than the internal chip frequency is possible. The chip clock is not required to run.

26.3 Data Registers

The Data Registers relevant for Boundary-scan operations are:

- Bypass Register
- Device Identification Register
- Reset Register
- Boundary-scan Chain
26.3.1 Bypass Register

The Bypass Register consists of a single Shift Register stage. When the Bypass Register is selected as path between TDI and TDO, the register is reset to 0 when leaving the Capture-DR controller state. The Bypass Register can be used to shorten the scan chain on a system when the other devices are to be tested.

26.3.2 Device Identification Register

Figure 26-1 shows the structure of the Device Identification Register.

Figure 26-1. The Format of the Device Identification Register

<table>
<thead>
<tr>
<th>Bit</th>
<th>MSB</th>
<th>LSB</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>28</td>
<td>27</td>
</tr>
<tr>
<td>12</td>
<td>11</td>
<td>10</td>
</tr>
</tbody>
</table>

Device ID

- **Version**: 4 bits
- **Part Number**: 16 bits
- **Manufacturer ID**: 11 bits

26.3.2.1 Version

Version is a 4-bit number identifying the revision of the component. The JTAG version number follows the revision of the device. Revision A is 0x0, revision B is 0x1 and so on.

26.3.2.2 Part Number

The part number is a 16-bit code identifying the component. The JTAG Part Number for ATmega16U4/ATmega32U4 is listed in Table 26-1.

<table>
<thead>
<tr>
<th>Part Number</th>
<th>JTAG Part Number (Hex)</th>
</tr>
</thead>
<tbody>
<tr>
<td>AVR USB</td>
<td>0x9782</td>
</tr>
</tbody>
</table>

26.3.2.3 Manufacturer ID

The Manufacturer ID is a 11-bit code identifying the manufacturer. The JTAG manufacturer ID for ATMEG is listed in Table 26-2.

<table>
<thead>
<tr>
<th>Manufacturer</th>
<th>JTAG Manufacturer ID (Hex)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ATMEG</td>
<td>0x01F</td>
</tr>
</tbody>
</table>

26.3.3 Reset Register

The Reset Register is a test Data Register used to reset the part. Since the AVR tri-states Port Pins when reset, the Reset Register can also replace the function of the unimplemented optional JTAG instruction HIGHZ.

A high value in the Reset Register corresponds to pulling the external Reset low. The part is reset as long as there is a high value present in the Reset Register. Depending on the fuse settings for the clock options, the part will remain reset for a reset time-out period (refer to “Clock Sources” on page 28) after releasing the Reset Register. The output from this Data Register is not latched, so the reset will take place immediately, as shown in Figure 26-2.
26.3.4 Boundary-scan Chain

The Boundary-scan Chain has the capability of driving and observing the logic levels on the digital I/O pins, as well as the boundary between digital and analog logic for analog circuitry having off-chip connections.

See “Boundary-scan Chain” on page 324 for a complete description.

26.4 Boundary-scan Specific JTAG Instructions

The Instruction Register is 4-bit wide, supporting up to 16 instructions. Listed below are the JTAG instructions useful for Boundary-scan operation. Note that the optional HIGHZ instruction is not implemented, but all outputs with tri-state capability can be set in high-impedance state by using the AVR_RESET instruction, since the initial state for all port pins is tri-state.

As a definition in this datasheet, the LSB is shifted in and out first for all Shift Registers.

The OPCODE for each instruction is shown behind the instruction name in hex format. The text describes which Data Register is selected as path between TDI and TDO for each instruction.

26.4.1 EXTEST; 0x0

Mandatory JTAG instruction for selecting the Boundary-scan Chain as Data Register for testing circuitry external to the AVR package. For port-pins, Pull-up Disable, Output Control, Output Data, and Input Data are all accessible in the scan chain. For Analog circuits having off-chip connections, the interface between the analog and the digital logic is in the scan chain. The contents of the latched outputs of the Boundary-scan chain is driven out as soon as the JTAG IR-Register is loaded with the EXTEST instruction.

The active states are:

- Capture-DR: Data on the external pins are sampled into the Boundary-scan Chain.
- Shift-DR: The Internal Scan Chain is shifted by the TCK input.
- Update-DR: Data from the scan chain is applied to output pins.

26.4.2 IDCODE; 0x1

Optional JTAG instruction selecting the 32 bit ID-Register as Data Register. The ID-Register consists of a version number, a device number and the manufacturer code chosen by JEDEC. This is the default instruction after power-up.
The active states are:
• Capture-DR: Data in the IDCODE Register is sampled into the Boundary-scan Chain.
• Shift-DR: The IDCODE scan chain is shifted by the TCK input.

26.4.3 SAMPLE_PRELOAD; 0x2
Mandatory JTAG instruction for pre-loading the output latches and taking a snap-shot of the input/output pins without affecting the system operation. However, the output latches are not connected to the pins. The Boundary-scan Chain is selected as Data Register.

The active states are:
• Capture-DR: Data on the external pins are sampled into the Boundary-scan Chain.
• Shift-DR: The Boundary-scan Chain is shifted by the TCK input.
• Update-DR: Data from the Boundary-scan chain is applied to the output latches. However, the output latches are not connected to the pins.

26.4.4 AVR_RESET; 0xC
The AVR specific public JTAG instruction for forcing the AVR device into the Reset mode or releasing the JTAG reset source. The TAP controller is not reset by this instruction. The one bit Reset Register is selected as Data Register. Note that the reset will be active as long as there is a logic “one” in the Reset Chain. The output from this chain is not latched.

The active states are:
• Shift-DR: The Reset Register is shifted by the TCK input.

26.4.5 BYPASS; 0xF
Mandatory JTAG instruction selecting the Bypass Register for Data Register.

The active states are:
• Capture-DR: Loads a logic “0” into the Bypass Register.
• Shift-DR: The Bypass Register cell between TDI and TDO is shifted.

26.5 Boundary-scan Related Register in I/O Memory

26.5.1 MCU Control Register – MCUCR
The MCU Control Register contains control bits for general MCU functions.

<table>
<thead>
<tr>
<th>Bit</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>JTD</td>
<td>R/W</td>
<td>–</td>
<td>–</td>
<td>PUD</td>
<td>–</td>
<td>–</td>
<td>IVSEL</td>
<td>IVCE</td>
</tr>
<tr>
<td>Read/Write</td>
<td>R/W</td>
<td>R</td>
<td>R</td>
<td>R/W</td>
<td>R</td>
<td>R</td>
<td>R/W</td>
<td>R/W</td>
</tr>
<tr>
<td>Initial Value</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

• Bits 7 – JTD: JTAG Interface Disable
When this bit is zero, the JTAG interface is enabled if the JTAGEN Fuse is programmed. If this bit is one, the JTAG interface is disabled. In order to avoid unintentional disabling or enabling of the JTAG interface, a timed sequence must be followed when changing this bit: The application software must write this bit to the desired value twice within four cycles to change its value. Note that this bit must not be altered when using the On-chip Debug system.
26.5.2 MCU Status Register – MCUSR

The MCU Status Register provides information on which reset source caused an MCU reset.

<table>
<thead>
<tr>
<th>Bit</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
<th>MCUSR</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read/Write</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td></td>
</tr>
<tr>
<td>Initial Value</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>See Bit Description</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **Bit 4 – JTRF: JTAG Reset Flag**
  
  This bit is set if a reset is being caused by a logic one in the JTAG Reset Register selected by the JTAG instruction AVR_RESET. This bit is reset by a Power-on Reset, or by writing a logic zero to the flag.

26.6 Boundary-scan Chain

The Boundary-scan chain has the capability of driving and observing the logic levels on the digital I/O pins, as well as the boundary between digital and analog logic for analog circuitry having off-chip connection.

26.6.1 Scanning the Digital Port Pins

**Figure 26-3** shows the Boundary-scan Cell for a bi-directional port pin. The pull-up function is disabled during Boundary-scan when the JTAG IC contains EXTEST or SAMPLE_PRELOAD. The cell consists of a bi-directional pin cell that combines the three signals Output Control - OCxn, Output Data - ODxn, and Input Data - IDxn, into only a two-stage Shift Register. The port and pin indexes are not used in the following description.

The Boundary-scan logic is not included in the figures in the datasheet. **Figure 26-4** shows a simple digital port pin as described in the section "I/O-Ports" on page 65. The Boundary-scan details from **Figure 26-3** replaces the dashed box in **Figure 26-4**.

When no alternate port function is present, the Input Data - ID - corresponds to the PINxn Register value (but ID has no synchronizer), Output Data corresponds to the PORT Register, Output Control corresponds to the Data Direction - DD Register, and the Pull-up Enable - PUExn - corresponds to logic expression PUDcdot DDxn - PORTxn.

Digital alternate port functions are connected outside the dotted box in **Figure 26-4** to make the scan chain read the actual pin value. For analog function, there is a direct connection from the external pin to the analog circuit. There is no scan chain on the interface between the digital and the analog circuitry, but some digital control signal to analog circuitry are turned off to avoid driving contention on the pads.

When JTAG IR contains EXTEST or SAMPLE_PRELOAD the clock is not sent out on the port pins even if the CKOUT fuse is programmed. Even though the clock is output when the JTAG IR contains SAMPLE_PRELOAD, the clock is not sampled by the boundary scan.
Figure 26-3. Boundary-scan Cell for Bi-directional Port Pin with Pull-up Function.
26.6.2 Scanning the RESET Pin

The RESET pin accepts 5V active low logic for standard reset operation, and 12V active high logic for High Voltage Parallel programming. An observe-only cell as shown in Figure 26-5 is inserted for the 5V reset signal.

Figure 26-4. General Port Pin Schematic Diagram

Figure 26-5. Observe-only Cell
26.7 ATmega16U4/ATmega32U4 Boundary-scan Order

Table 26-3 shows the Scan order between TDI and TDO when the Boundary-scan chain is selected as data path. Bit 0 is the LSB; the first bit scanned in, and the first bit scanned out. The scan order follows the pin-out order as far as possible. Exceptions from the rules are the Scan chains for the analog circuits, which constitute the most significant bits of the scan chain regardless of which physical pin they are connected to. In Figure 26-3, PXn. Data corresponds to FF0, FXn. Control corresponds to FF1, PXn. Bit 4, 5, 6 and 7 of Port F is not in the scan chain, since these pins constitute the TAP pins when the JTAG is enabled. The USB pads are not included in the boundary-scan.

<table>
<thead>
<tr>
<th>Bit Number</th>
<th>Signal Name</th>
<th>Module</th>
</tr>
</thead>
<tbody>
<tr>
<td>88</td>
<td>PE6.Data</td>
<td>Port E</td>
</tr>
<tr>
<td>87</td>
<td>PE6.Control</td>
<td></td>
</tr>
<tr>
<td>86</td>
<td>Reserved</td>
<td></td>
</tr>
<tr>
<td>85</td>
<td>Reserved</td>
<td></td>
</tr>
<tr>
<td>84</td>
<td>Reserved</td>
<td></td>
</tr>
<tr>
<td>83</td>
<td>Reserved</td>
<td></td>
</tr>
<tr>
<td>82</td>
<td>PB0.Data</td>
<td></td>
</tr>
<tr>
<td>81</td>
<td>PB0.Control</td>
<td></td>
</tr>
<tr>
<td>80</td>
<td>PB1.Data</td>
<td></td>
</tr>
<tr>
<td>79</td>
<td>PB1.Control</td>
<td></td>
</tr>
<tr>
<td>78</td>
<td>PB2.Data</td>
<td></td>
</tr>
<tr>
<td>77</td>
<td>PB2.Control</td>
<td></td>
</tr>
<tr>
<td>76</td>
<td>PB3.Data</td>
<td>Port B</td>
</tr>
<tr>
<td>75</td>
<td>PB3.Control</td>
<td></td>
</tr>
<tr>
<td>74</td>
<td>PB4.Data</td>
<td></td>
</tr>
<tr>
<td>73</td>
<td>PB4.Control</td>
<td></td>
</tr>
<tr>
<td>72</td>
<td>PB5.Data</td>
<td></td>
</tr>
<tr>
<td>71</td>
<td>PB5.Control</td>
<td></td>
</tr>
<tr>
<td>70</td>
<td>PB6.Data</td>
<td></td>
</tr>
<tr>
<td>69</td>
<td>PB6.Control</td>
<td></td>
</tr>
<tr>
<td>68</td>
<td>PB7.Data</td>
<td></td>
</tr>
<tr>
<td>67</td>
<td>PB7.Control</td>
<td></td>
</tr>
<tr>
<td>66</td>
<td>Reserved</td>
<td>PORTE</td>
</tr>
<tr>
<td>65</td>
<td>Reserved</td>
<td></td>
</tr>
<tr>
<td>64</td>
<td>Reserved</td>
<td></td>
</tr>
<tr>
<td>63</td>
<td>Reserved</td>
<td></td>
</tr>
<tr>
<td>62</td>
<td>RSTT</td>
<td>Reset Logic (Observe Only)</td>
</tr>
<tr>
<td>Bit Number</td>
<td>Signal Name</td>
<td>Module</td>
</tr>
<tr>
<td>------------</td>
<td>---------------</td>
<td>--------</td>
</tr>
<tr>
<td>61</td>
<td>PD0.Data</td>
<td></td>
</tr>
<tr>
<td>60</td>
<td>PD0.Control</td>
<td></td>
</tr>
<tr>
<td>59</td>
<td>PD1.Data</td>
<td></td>
</tr>
<tr>
<td>58</td>
<td>PD1.Control</td>
<td></td>
</tr>
<tr>
<td>57</td>
<td>PD2.Data</td>
<td></td>
</tr>
<tr>
<td>56</td>
<td>PD2.Control</td>
<td></td>
</tr>
<tr>
<td>55</td>
<td>PD3.Data</td>
<td>Port D</td>
</tr>
<tr>
<td>54</td>
<td>PD3.Control</td>
<td></td>
</tr>
<tr>
<td>53</td>
<td>PD4.Data</td>
<td></td>
</tr>
<tr>
<td>52</td>
<td>PD4.Control</td>
<td></td>
</tr>
<tr>
<td>51</td>
<td>PD5.Data</td>
<td></td>
</tr>
<tr>
<td>50</td>
<td>PD5.Control</td>
<td></td>
</tr>
<tr>
<td>49</td>
<td>PD6.Data</td>
<td></td>
</tr>
<tr>
<td>48</td>
<td>PD6.Control</td>
<td></td>
</tr>
<tr>
<td>47</td>
<td>PD7.Data</td>
<td></td>
</tr>
<tr>
<td>46</td>
<td>PD7.Control</td>
<td></td>
</tr>
<tr>
<td>45</td>
<td>Reserved</td>
<td>Port E</td>
</tr>
<tr>
<td>44</td>
<td>Reserved</td>
<td></td>
</tr>
<tr>
<td>43</td>
<td>Reserved</td>
<td></td>
</tr>
<tr>
<td>42</td>
<td>Reserved</td>
<td></td>
</tr>
<tr>
<td>41</td>
<td>Reserved</td>
<td></td>
</tr>
<tr>
<td>40</td>
<td>Reserved</td>
<td></td>
</tr>
<tr>
<td>39</td>
<td>Reserved</td>
<td></td>
</tr>
<tr>
<td>38</td>
<td>Reserved</td>
<td></td>
</tr>
<tr>
<td>37</td>
<td>Reserved</td>
<td></td>
</tr>
<tr>
<td>36</td>
<td>Reserved</td>
<td></td>
</tr>
<tr>
<td>35</td>
<td>Reserved</td>
<td></td>
</tr>
<tr>
<td>34</td>
<td>Reserved</td>
<td></td>
</tr>
<tr>
<td>33</td>
<td>Reserved</td>
<td></td>
</tr>
<tr>
<td>32</td>
<td>Reserved</td>
<td></td>
</tr>
<tr>
<td>31</td>
<td>Reserved</td>
<td></td>
</tr>
<tr>
<td>30</td>
<td>Reserved</td>
<td></td>
</tr>
<tr>
<td>29</td>
<td>Reserved</td>
<td></td>
</tr>
<tr>
<td>28</td>
<td>Reserved</td>
<td></td>
</tr>
<tr>
<td>27</td>
<td>Reserved</td>
<td></td>
</tr>
<tr>
<td>26</td>
<td>Reserved</td>
<td></td>
</tr>
</tbody>
</table>
### 26.8 Boundary-scan Description Language Files

Boundary-scan Description Language (BSDL) files describe Boundary-scan capable devices in a standard format used by automated test-generation software. The order and function of bits in the Boundary-scan Data Register are included in this description. BSDL files are available for ATmega16U4/ATmega32U4.

<table>
<thead>
<tr>
<th>Bit Number</th>
<th>Signal Name</th>
<th>Module</th>
</tr>
</thead>
<tbody>
<tr>
<td>25</td>
<td>PE2.Data</td>
<td>Port E</td>
</tr>
<tr>
<td>24</td>
<td>PE2.Control</td>
<td></td>
</tr>
<tr>
<td>23</td>
<td>Reserved</td>
<td></td>
</tr>
<tr>
<td>22</td>
<td>Reserved</td>
<td></td>
</tr>
<tr>
<td>21</td>
<td>Reserved</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>Reserved</td>
<td></td>
</tr>
<tr>
<td>19</td>
<td>Reserved</td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>Reserved</td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>Reserved</td>
<td>Reserved</td>
</tr>
<tr>
<td>16</td>
<td>Reserved</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>Reserved</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>Reserved</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>Reserved</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>Reserved</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>Reserved</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>Reserved</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>Reserved</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>Reserved</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>Reserved</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>Reserved</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>Reserved</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Reserved</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>PF1.Data</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>PF1.Control</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>PF0.Data</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>PF0.Control</td>
<td></td>
</tr>
</tbody>
</table>
27. Boot Loader Support – Read-While-Write Self-Programming

The Boot Loader Support provides a real Read-While-Write Self-Programming mechanism for downloading and uploading program code by the MCU itself. This feature allows flexible application software updates controlled by the MCU using a Flash-resident Boot Loader program. The Boot Loader program can use any available data interface and associated protocol to read code and write (program) that code into the Flash memory, or read the code from the program memory. The program code within the Boot Loader section has the capability to write into the entire Flash, including the Boot Loader memory. The Boot Loader can thus even modify itself, and it can also erase itself from the code if the feature is not needed anymore. The size of the Boot Loader memory is configurable with fuses and the Boot Loader has two separate sets of Boot Lock bits which can be set independently. This gives the user a unique flexibility to select different levels of protection. General information on SPM and ELPM is provided in See “AVR CPU Core” on page 9.

27.1 Boot Loader Features

- Read-While-Write Self-Programming
- Flexible Boot Memory Size
- High Security (Separate Boot Lock Bits for a Flexible Protection)
- Separate Fuse to Select Reset Vector
- Optimized Page(1) Size
- Code Efficient Algorithm
- Efficient Read-Modify-Write Support

Note: 1. A page is a section in the Flash consisting of several bytes (see Table 28-11 on page 351) used during programming. The page organization does not affect normal operation.

27.2 Application and Boot Loader Flash Sections

The Flash memory is organized in two main sections, the Application section and the Boot Loader section (see Figure 27-2). The size of the different sections is configured by the BOOTSZ Fuses as shown in Table 27-8 on page 344 and Figure 27-2. These two sections can have different level of protection since they have different sets of Lock bits.

27.2.1 Application Section

The Application section is the section of the Flash that is used for storing the application code. The protection level for the Application section can be selected by the application Boot Lock bits (Boot Lock bits 0), see Table 27-2 on page 334. The Application section can never store any Boot Loader code since the SPM instruction is disabled when executed from the Application section.

27.2.2 BLS – Boot Loader Section

While the Application section is used for storing the application code, the The Boot Loader software must be located in the BLS since the SPM instruction can initiate a programming when executing from the BLS only. The SPM instruction can access the entire Flash, including the BLS itself. The protection level for the Boot Loader section can be selected by the Boot Loader Lock bits (Boot Lock bits 1), see Table 27-3 on page 334.

27.3 Read-While-Write and No Read-While-Write Flash Sections

Whether the CPU supports Read-While-Write or if the CPU is halted during a Boot Loader software update is dependent on which address that is being programmed. In addition to the two
sections that are configurable by the BOOTSZ Fuses as described above, the Flash is also divided into two fixed sections, the Read-While-Write (RWW) section and the No Read-While-Write (NRWW) section. The limit between the RWW- and NRWW sections is given in Table 27-1 and Figure 27-1 on page 332. The main difference between the two sections is:

- When erasing or writing a page located inside the RWW section, the NRWW section can be read during the operation.
- When erasing or writing a page located inside the NRWW section, the CPU is halted during the entire operation.

Note that the user software can never read any code that is located inside the RWW section during a Boot Loader software operation. The syntax “Read-While-Write section” refers to which section that is being programmed (erased or written), not which section that actually is being read during a Boot Loader software update.

27.3.1 RWW – Read-While-Write Section

If a Boot Loader software update is programming a page inside the RWW section, it is possible to read code from the Flash, but only code that is located in the NRWW section. During an ongoing programming, the software must ensure that the RWW section never is being read. If the user software is trying to read code that is located inside the RWW section (i.e., by load program memory, call, or jump instructions or an interrupt) during programming, the software might end up in an unknown state. To avoid this, the interrupts should either be disabled or moved to the Boot Loader section. The Boot Loader section is always located in the NRWW section. The RWW Section Busy bit (RWWSB) in the Store Program Memory Control and Status Register (SPMCSR) will be read as logical one as long as the RWW section is blocked for reading. After a programming is completed, the RWWSB must be cleared by software before reading code located in the RWW section. See “Store Program Memory Control and Status Register – SPMCSR” on page 336. for details on how to clear RWWSB.

27.3.2 NRWW – No Read-While-Write Section

The code located in the NRWW section can be read when the Boot Loader software is updating a page in the RWW section. When the Boot Loader code updates the NRWW section, the CPU is halted during the entire Page Erase or Page Write operation.

<table>
<thead>
<tr>
<th>Table 27-1. Read-While-Write Features</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Which Section does the Z-pointer Address During the Programming?</strong></td>
</tr>
<tr>
<td>RWW Section</td>
</tr>
<tr>
<td>NRWW Section</td>
</tr>
</tbody>
</table>
Figure 27-1. Read-While-Write vs. No Read-While-Write

- Read-While-Write (RWW) Section
- No Read-While-Write (NRWW) Section
- Z-pointer Addresses RWW Section
- Code Located in NRWW Section Can be Read During the Operation
- Z-pointer Addresses NRWW Section
- CPU is Halted During the Operation
Figure 27-2. Memory Sections

Note: 1. The parameters in the figure above are given in Table 27-8 on page 344.

27.4 Boot Loader Lock Bits

If no Boot Loader capability is needed, the entire Flash is available for application code. The Boot Loader has two separate sets of Boot Lock bits which can be set independently. This gives the user a unique flexibility to select different levels of protection.

The user can select:

- To protect the entire Flash from a software update by the MCU.
- To protect only the Boot Loader Flash section from a software update by the MCU.
- To protect only the Application Flash section from a software update by the MCU.
- Allow software update in the entire Flash.

See Table 27-2 and Table 27-3 for further details. The Boot Lock bits can be set by software and in Serial or in Parallel Programming mode. They can only be cleared by a Chip Erase command only. The general Write Lock (Lock Bit mode 2) does not control the programming of the Flash memory by SPM instruction. Similarly, the general Read/Write Lock (Lock Bit mode 1) does not control reading nor writing by (E)LPM/SPM, if it is attempted.
Table 27-2. Boot Lock Bit0 Protection Modes (Application Section)(1)

<table>
<thead>
<tr>
<th>BLB0 Mode</th>
<th>BLB02</th>
<th>BLB01</th>
<th>Protection</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>No restrictions for SPM or (E)LPM accessing the Application section.</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>0</td>
<td>SPM is not allowed to write to the Application section.</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>0</td>
<td>SPM is not allowed to write to the Application section, and (E)LPM executing from the Boot Loader section is not allowed to read from the Application section. If Interrupt Vectors are placed in the Boot Loader section, interrupts are disabled while executing from the Application section.</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>1</td>
<td>(E)LPM executing from the Boot Loader section is not allowed to read from the Application section. If Interrupt Vectors are placed in the Boot Loader section, interrupts are disabled while executing from the Application section.</td>
</tr>
</tbody>
</table>

Note: 1. “1” means unprogrammed, “0” means programmed

Table 27-3. Boot Lock Bit1 Protection Modes (Boot Loader Section)(1)

<table>
<thead>
<tr>
<th>BLB1 Mode</th>
<th>BLB12</th>
<th>BLB11</th>
<th>Protection</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>No restrictions for SPM or (E)LPM accessing the Boot Loader section.</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>0</td>
<td>SPM is not allowed to write to the Boot Loader section.</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>0</td>
<td>SPM is not allowed to write to the Boot Loader section, and (E)LPM executing from the Application section is not allowed to read from the Boot Loader section. If Interrupt Vectors are placed in the Application section, interrupts are disabled while executing from the Boot Loader section.</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>1</td>
<td>(E)LPM executing from the Application section is not allowed to read from the Boot Loader section. If Interrupt Vectors are placed in the Application section, interrupts are disabled while executing from the Boot Loader section.</td>
</tr>
</tbody>
</table>

Note: 1. “1” means unprogrammed, “0” means programmed

27.5 Entering the Boot Loader Program

The bootloader can be executed with three different conditions:

27.5.1 Regular application conditions.

A jump or call from the application program. This may be initiated by a trigger such as a command received via USART, SPI or USB.

27.5.2 Boot Reset Fuse

The Boot Reset Fuse (BOOTRST) can be programmed so that the Reset Vector is pointing to the Boot Flash start address after a reset. In this case, the Boot Loader is started after a reset. After the application code is loaded, the program can start executing the application code. Note that the fuses cannot be changed by the MCU itself. This means that once the Boot Reset Fuse...
is programmed, the Reset Vector will always point to the Boot Loader Reset and the fuse can only be changed through the serial or parallel programming interface.

**Table 27-4. Boot Reset Fuse**

<table>
<thead>
<tr>
<th>BOOTRST</th>
<th>Reset Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Reset Vector = Application Reset (address 0x0000)</td>
</tr>
<tr>
<td>0</td>
<td>Reset Vector = Boot Loader Reset (see Table 27-8 on page 344)</td>
</tr>
</tbody>
</table>

Note: 1. “1” means unprogrammed, “0” means programmed

### 27.5.3 External Hardware conditions

The Hardware Boot Enable Fuse (HWBE) can be programmed (See Table 27-5) so that upon special hardware conditions under reset, the bootloader execution is forced after reset.

**Table 27-5. Hardware Boot Enable Fuse**

<table>
<thead>
<tr>
<th>HWBE</th>
<th>Reset Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>ALE/HWB pin cannot be used to force Boot Loader execution after reset</td>
</tr>
<tr>
<td>0</td>
<td>ALE/HWB pin is used during reset to force bootloader execution after reset</td>
</tr>
</tbody>
</table>

Note: 1. “1” means unprogrammed, “0” means programmed

When the HWBE fuse is enable the ALE/HWB pin is configured as input during reset and sampled during reset rising edge. When ALE/HWB pin is ‘0’ during reset rising edge, the reset vector will be set as the Boot Loader Reset address and the Boot Loader will be executed (See Figures 27-3).

**Figure 27-3. Boot Process Description**
27.5.4 Store Program Memory Control and Status Register – SPMCSR

The Store Program Memory Control and Status Register contains the control bits needed to control the Boot Loader operations.

<table>
<thead>
<tr>
<th>Bit</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
<th>SPMCSR</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read/Write</td>
<td>R/W</td>
<td>R</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
</tr>
<tr>
<td>Initial Value</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

- **Bit 7 – SPMIE: SPM Interrupt Enable**
  When the SPMIE bit is written to one, and the I-bit in the Status Register is set (one), the SPM ready interrupt will be enabled. The SPM ready Interrupt will be executed as long as the SPMEN bit in the SPMCSR Register is cleared.

- **Bit 6 – RWWSB: Read-While-Write Section Busy**
  When a Self-Programming (Page Erase or Page Write) operation to the RWW section is initiated, the RWWSB bit will be set (one) by hardware. When the RWWSB bit is set, the RWW section cannot be accessed. The RWWSB bit will be cleared if the RWWSRE bit is written to one after a Self-Programming operation is completed. Alternatively the RWWSB bit will automatically be cleared if a page load operation is initiated.

- **Bit 5 – SIGRD: Signature Row Read**
  If this bit is written to one at the same time as SPMEN, the next LPM instruction within three clock cycles will read a byte from the signature row into the destination register. see “Reading the Signature Row from Software” on page 341 for details. An SPM instruction within four cycles after SIGRD and SPMEN are set will have no effect. This operation is reserved for future use and should not be used.

- **Bit 4 – RWWSRE: Read-While-Write Section Read Enable**
  When programming (Page Erase or Page Write) to the RWW section, the RWW section is blocked for reading (the RWWSB will be set by hardware). To re-enable the RWW section, the user software must wait until the programming is completed (SPMEN will be cleared). Then, if the RWWSRE bit is written to one at the same time as SPMEN, the next SPM instruction within four clock cycles re-enables the RWW section. The RWW section cannot be re-enabled while the Flash is busy with a Page Erase or a Page Write (SPMEN is set). If the RWWSRE bit is written while the Flash is being loaded, the Flash load operation will abort and the data loaded will be lost.

- **Bit 3 – BLBSET: Boot Lock Bit Set**
  If this bit is written to one at the same time as SPMEN, the next SPM instruction within four clock cycles sets Boot Lock bits, according to the data in R0. The data in R1 and the address in the Z-pointer are ignored. The BLBSET bit will automatically be cleared upon completion of the Lock bit set, or if no SPM instruction is executed within four clock cycles.

An (E)LPM instruction within three cycles after BLBSET and SPMEN are set in the SPMCSR Register, will read either the Lock bits or the Fuse bits (depending on Z0 in the Z-pointer) into the destination register. See “Reading the Fuse and Lock Bits from Software” on page 340 for details.
• **Bit 2 – PGWRT: Page Write**
If this bit is written to one at the same time as SPMEN, the next SPM instruction within four clock cycles executes Page Write, with the data stored in the temporary buffer. The page address is taken from the high part of the Z-pointer. The data in R1 and R0 are ignored. The PGWRT bit will auto-clear upon completion of a Page Write, or if no SPM instruction is executed within four clock cycles. The CPU is halted during the entire Page Write operation if the NRWW section is addressed.

• **Bit 1 – PGERS: Page Erase**
If this bit is written to one at the same time as SPMEN, the next SPM instruction within four clock cycles executes Page Erase. The page address is taken from the high part of the Z-pointer. The data in R1 and R0 are ignored. The PGERS bit will auto-clear upon completion of a Page Erase, or if no SPM instruction is executed within four clock cycles. The CPU is halted during the entire Page Write operation if the NRWW section is addressed.

• **Bit 0 – SPMEN: Store Program Memory Enable**
This bit enables the SPM instruction for the next four clock cycles. If written to one together with either RWWSRE, BLBSET, PGWRT" or PGERS, the following SPM instruction will have a special meaning, see description above. If only SPMEN is written, the following SPM instruction will store the value in R1:R0 in the temporary page buffer addressed by the Z-pointer. The LSB of the Z-pointer is ignored. The SPMEN bit will auto-clear upon completion of an SPM instruction, or if no SPM instruction is executed within four clock cycles. During Page Erase and Page Write, the SPMEN bit remains high until the operation is completed.

Writing any other combination than “10001”, “01001”, “00101”, “00011” or “00001” in the lower five bits will have no effect.

Note: Only one SPM instruction should be active at any time.

### 27.6 Addressing the Flash During Self-Programming

The Z-pointer is used to address the SPM commands. The Z pointer consists of the Z-registers ZL and ZH in the register file, and RAMPZ in the I/O space. The number of bits actually used is implementation dependent. Note that the RAMPZ register is only implemented when the program space is larger than 64K bytes.

<table>
<thead>
<tr>
<th>Bit</th>
<th>23</th>
<th>22</th>
<th>21</th>
<th>20</th>
<th>19</th>
<th>18</th>
<th>17</th>
<th>16</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZH (R31)</td>
<td>Z15</td>
<td>Z14</td>
<td>Z13</td>
<td>Z12</td>
<td>Z11</td>
<td>Z10</td>
<td>Z9</td>
<td>Z8</td>
</tr>
<tr>
<td>ZL (R30)</td>
<td>Z7</td>
<td>Z6</td>
<td>Z5</td>
<td>Z4</td>
<td>Z3</td>
<td>Z2</td>
<td>Z1</td>
<td>Z0</td>
</tr>
</tbody>
</table>

Since the Flash is organized in pages (see Table 28-11 on page 351), the Program Counter can be treated as having two different sections. One section, consisting of the least significant bits, is addressing the words within a page, while the most significant bits are addressing the pages. This is shown in Figure 27-4. Note that the Page Erase and Page Write operations are addressed independently. Therefore it is of major importance that the Boot Loader software addresses the same page in both the Page Erase and Page Write operation. Once a programming operation is initiated, the address is latched and the Z-pointer can be used for other operations.

The (E)LPM instruction use the Z-pointer to store the address. Since this instruction addresses the Flash byte-by-byte, also bit Z0 of the Z-pointer is used.
27.7 Self-Programming the Flash

The program memory is updated in a page by page fashion. Before programming a page with the data stored in the temporary page buffer, the page must be erased. The temporary page buffer is filled one word at a time using SPM and the buffer can be filled either before the Page Erase command or between a Page Erase and a Page Write operation:

Alternative 1, fill the buffer before a Page Erase

- Fill temporary page buffer
- Perform a Page Erase
- Perform a Page Write

Alternative 2, fill the buffer after Page Erase

- Perform a Page Erase
- Fill temporary page buffer
- Perform a Page Write

If only a part of the page needs to be changed, the rest of the page must be stored (for example in the temporary page buffer) before the erase, and then be rewritten. When using alternative 1, the Boot Loader provides an effective Read-Modify-Write feature which allows the user software to first read the page, do the necessary changes, and then write back the modified data. If alternative 2 is used, it is not possible to read the old data while loading since the page is already erased. The temporary page buffer can be accessed in a random sequence. It is essential that the page address used in both the Page Erase and Page Write operation is addressing the same
page. See “Simple Assembly Code Example for a Boot Loader” on page 342 for an assembly code example.

27.7.1 Performing Page Erase by SPM
To execute Page Erase, set up the address in the Z-pointer, write “X0000011” to SPMCSR and execute SPM within four clock cycles after writing SPMCSR. The data in R1 and R0 is ignored. The page address must be written to PCPAGE in the Z-register. Other bits in the Z-pointer will be ignored during this operation.

- Page Erase to the RWW section: The NRWW section can be read during the Page Erase.
- Page Erase to the NRWW section: The CPU is halted during the operation.

27.7.2 Filling the Temporary Buffer (Page Loading)
To write an instruction word, set up the address in the Z-pointer and data in R1:R0, write “00000001” to SPMCSR and execute SPM within four clock cycles after writing SPMCSR. The content of PCWORD in the Z-register is used to address the data in the temporary buffer. The temporary buffer will auto-erase after a Page Write operation or by writing the RWWSRE bit in SPMCSR. It is also erased after a system reset. Note that it is not possible to write more than one time to each address without erasing the temporary buffer.

If the EEPROM is written in the middle of an SPM Page Load operation, all data loaded will be lost.

27.7.3 Performing a Page Write
To execute Page Write, set up the address in the Z-pointer, write “X0000101” to SPMCSR and execute SPM within four clock cycles after writing SPMCSR. The data in R1 and R0 is ignored. The page address must be written to PCPAGE. Other bits in the Z-pointer must be written to zero during this operation.

- Page Write to the RWW section: The NRWW section can be read during the Page Write.
- Page Write to the NRWW section: The CPU is halted during the operation.

27.7.4 Using the SPM Interrupt
If the SPM interrupt is enabled, the SPM interrupt will generate a constant interrupt when the SPMEN bit in SPMCSR is cleared. This means that the interrupt can be used instead of polling the SPMCSR Register in software. When using the SPM interrupt, the Interrupt Vectors should be moved to the BLS section to avoid that an interrupt is accessing the RWW section when it is blocked for reading. How to move the interrupts is described in “Interrupts” on page 61.

27.7.5 Consideration While Updating BLS
Special care must be taken if the user allows the Boot Loader section to be updated by leaving Boot Lock bit11 unprogrammed. An accidental write to the Boot Loader itself can corrupt the entire Boot Loader, and further software updates might be impossible. If it is not necessary to change the Boot Loader software itself, it is recommended to program the Boot Lock bit11 to protect the Boot Loader software from any internal software changes.

27.7.6 Prevent Reading the RWW Section During Self-Programming
During Self-Programming (either Page Erase or Page Write), the RWW section is always blocked for reading. The user software itself must prevent that this section is addressed during the self programming operation. The RWWSB in the SPMCSR will be set as long as the RWW section is busy. During Self-Programming the Interrupt Vector table should be moved to the BLS
as described in “Interrupts” on page 61, or the interrupts must be disabled. Before addressing
the RWW section after the programming is completed, the user software must clear the
RWWSB by writing the RWWSRE. See “Simple Assembly Code Example for a Boot Loader” on
page 342 for an example.

27.7.7 Setting the Boot Loader Lock Bits by SPM
To set the Boot Loader Lock bits, write the desired data to R0, write “X0001001” to SPMCSR
and execute SPM within four clock cycles after writing SPMCSR. The only accessible Lock bits
are the Boot Lock bits that may prevent the Application and Boot Loader section from any soft-
ware update by the MCU.

<table>
<thead>
<tr>
<th>Bit</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>R0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

See Table 27-2 and Table 27-3 for how the different settings of the Boot Loader bits affect the
Flash access.

If bits 5..2 in R0 are cleared (zero), the corresponding Boot Lock bit will be programmed if an
SPM instruction is executed within four cycles after BLBSET and SPMEN are set in SPMCSR.
The Z-pointer is don’t care during this operation, but for future compatibility it is recommended to
load the Z-pointer with 0x0001 (same as used for reading the IOcell bits). For future compatibility it
is also recommended to set bits 7, 6, 1, and 0 in R0 to “1” when writing the Lock bits. When pro-
gramming the Lock bits the entire Flash can be read during the operation.

27.7.8 EEPROM Write Prevents Writing to SPMCSR
Note that an EEPROM write operation will block all software programming to Flash. Reading the
Fuses and Lock bits from software will also be prevented during the EEPROM write operation. It
is recommended that the user checks the status bit (EEPE) in the EECR Register and verifies
that the bit is cleared before writing to the SPMCSR Register.

27.7.9 Reading the Fuse and Lock Bits from Software
It is possible to read both the Fuse and Lock bits from software. To read the Lock bits, load the
Z-pointer with 0x0001 and set the BLBSET and SPMEN bits in SPMCSR. When an (E)LPM
instruction is executed within three CPU cycles after the BLBSET and SPMEN bits are set in
SPMCSR, the value of the Lock bits will be loaded in the destination register. The BLBSET and
SPMEN bits will auto-clear upon completion of reading the Lock bits or if no (E)LPM instruction
is executed within three CPU cycles or no SPM instruction is executed within four CPU cycles.
When BLBSET and SPMEN are cleared, (E)LPM will work as described in the Instruction set
Manual.

<table>
<thead>
<tr>
<th>Bit</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rd</td>
<td>-</td>
<td>-</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

The algorithm for reading the Fuse Low byte is similar to the one described above for reading
the Lock bits. To read the Fuse Low byte, load the Z-pointer with 0x0000 and set the BLBSET
and SPMEN bits in SPMCSR. When an (E)LPM instruction is executed within three cycles after
the BLBSET and SPMEN bits are set in the SPMCSR, the value of the Fuse Low byte (FLB) will
be loaded in the destination register as shown below. Refer to Table 28-5 on page 348 for a
detailed description and mapping of the Fuse Low byte.

<table>
<thead>
<tr>
<th>Bit</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rd</td>
<td>FLB7</td>
<td>FLB6</td>
<td>FLB5</td>
<td>FLB4</td>
<td>FLB3</td>
<td>FLB2</td>
<td>FLB1</td>
<td>FLB0</td>
</tr>
</tbody>
</table>
Similarly, when reading the Fuse High byte, load 0x0003 in the Z-pointer. When an (E)LPM instruction is executed within three cycles after the BLBSET and SPMEN bits are set in the SPMCSR, the value of the Fuse High byte (FHB) will be loaded in the destination register as shown below. Refer to Table 28-4 on page 348 for detailed description and mapping of the Fuse High byte.

<table>
<thead>
<tr>
<th>Bit</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rd</td>
<td>FHB7</td>
<td>FHB6</td>
<td>FHB5</td>
<td>FHB4</td>
<td>FHB3</td>
<td>FHB2</td>
<td>FHB1</td>
<td>FHB0</td>
</tr>
</tbody>
</table>

When reading the Extended Fuse byte, load 0x0002 in the Z-pointer. When an (E)LPM instruction is executed within three cycles after the BLBSET and SPMEN bits are set in the SPMCSR, the value of the Extended Fuse byte (EFB) will be loaded in the destination register as shown below. Refer to Table 28-3 on page 347 for detailed description and mapping of the Extended Fuse byte.

<table>
<thead>
<tr>
<th>Bit</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rd</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>EFB2</td>
<td>EFB1</td>
<td>EFB0</td>
</tr>
</tbody>
</table>

Fuse and Lock bits that are programmed, will be read as zero. Fuse and Lock bits that are unprogrammed, will be read as one.

### 27.7.10 Reading the Signature Row from Software

To read the Signature Row from software, load the Z-pointer with the signature byte address given in Table 27-6 on page 341 and set the SIGRD and SPMEN bits in SPMCSR. When an LPM instruction is executed within three CPU cycles after the SIGRD and SPMEN bits are set in SPMCSR, the signature byte value will be loaded in the destination register. The SIGRD and SPMEN bits will auto-clear upon completion of reading the Signature Row Lock bits or if no LPM instruction is executed within three CPU cycles. When SIGRD and SPMEN are cleared, LPM will work as described in the Instruction set Manual.

#### Table 27-6. Signature Row Addressing

<table>
<thead>
<tr>
<th>Signature Byte</th>
<th>Z-Pointer Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>Device Signature Byte 1</td>
<td>0x0000</td>
</tr>
<tr>
<td>Device Signature Byte 2</td>
<td>0x0002</td>
</tr>
<tr>
<td>Device Signature Byte 3</td>
<td>0x0004</td>
</tr>
<tr>
<td>RC Oscillator Calibration Byte</td>
<td>0x0001</td>
</tr>
</tbody>
</table>

Note: All other addresses are reserved for future use.

### 27.7.11 Preventing Flash Corruption

During periods of low \( V_{CC} \), the Flash program can be corrupted because the supply voltage is too low for the CPU and the Flash to operate properly. These issues are the same as for board level systems using the Flash, and the same design solutions should be applied.

A Flash program corruption can be caused by two situations when the voltage is too low. First, a regular write sequence to the Flash requires a minimum voltage to operate correctly. Secondly, the CPU itself can execute instructions incorrectly, if the supply voltage for executing instructions is too low.

Flash corruption can easily be avoided by following these design recommendations (one is sufficient):
1. If there is no need for a Boot Loader update in the system, program the Boot Loader Lock bits to prevent any Boot Loader software updates.

2. Keep the AVR RESET active (low) during periods of insufficient power supply voltage. This can be done by enabling the internal Brown-out Detector (BOD) if the operating voltage matches the detection level. If not, an external low $V_{CC}$ reset protection circuit can be used. If a reset occurs while a write operation is in progress, the write operation will be completed provided that the power supply voltage is sufficient.

3. Keep the AVR core in Power-down sleep mode during periods of low $V_{CC}$. This will prevent the CPU from attempting to decode and execute instructions, effectively protecting the SPMCSR Register and thus the Flash from unintentional writes.

### 27.7.12 Programming Time for Flash when Using SPM

The calibrated RC Oscillator is used to time Flash accesses. Table 27-7 shows the typical programming time for Flash accesses from the CPU.

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Min Programming Time</th>
<th>Max Programming Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Flash write (Page Erase, Page Write, and write Lock bits by SPM)</td>
<td>3.7 ms</td>
<td>4.5 ms</td>
</tr>
</tbody>
</table>

### 27.7.13 Simple Assembly Code Example for a Boot Loader

```asm
;--the routine writes one page of data from RAM to Flash
; the first data location in RAM is pointed to by the Y pointer
; the first data location in Flash is pointed to by the Z-pointer
;--error handling is not included
;--the routine must be placed inside the Boot space
; (at least the Do_spm sub routine). Only code inside NRW section can
; be read during Self-Programming (Page Erase and Page Write).
;--registers used: r0, r1, templ (r16), temp2 (r17), looplo (r24),
;   loophi (r25), spmcrcval (r20)
; storing and restoring of registers is not included in the routine
; register usage can be optimized at the expense of code size
;--It is assumed that either the interrupt table is moved to the Boot
; loader section or that the interrupts are disabled.
.equ PAGESIZEB = PAGESIZE*2 ;PAGESIZEB is page size in BYTES, not words
.org SMALLBOOTSTART
Write_page:
; Page Erase
ldi spmcrcval, (1<<PGERS) | (1<<SPMEN)
call Do_spm

; re-enable the RW section
ldi spmcrcval, (1<<RWWSRE) | (1<<SPMEN)
call Do_spm

; transfer data from RAM to Flash page buffer
ldi looplo, low(PAGESIZEB) ;init loop variable
ldi loophi, high(PAGESIZEB) ;not required for PAGESIZEB<=256
Wrloop:
    ld r0, Y+
    ld r1, Y+
    ldi spmcrcval, (1<<SPMEN)
call Do_spm
    adiw ZH:2L, 2
    sbiw loophi:looplo, 2 ;use subi for PAGESIZEB<=256
```

---

ATmega16/32U4

ATMEL®

7766F–AVR–11/10

342
brne Wrloop

; execute Page Write
subi ZL, low(PAGESIZEB) ;restore pointer
sbc ZH, high(PAGESIZEB) ;not required for PAGESIZEB<=256
ldi spmcrval, (1<<PGWRT) | (1<<SPMEN)
call Do_spm

; re-enable the RW section
ldi spmcrval, (1<<RWNSRE) | (1<<SPMEN)
call Do_spm

; read back and check, optional
ldi looplo, low(PAGESIZEB) ;init loop variable
ldi loophi, high(PAGESIZEB) ;not required for PAGESIZEB<=256
subi YL, low(PAGESIZEB) ;restore pointer
sbc YH, high(PAGESIZEB)
Rdloop:
elpm r0, Z+
ld r1, Y+
cpse r0, r1
jmp Error
sbiw loophi:looplo, 1 ;use subi for PAGESIZEB<=256
brne Rdloop

; return to RW section
; verify that RW section is safe to read
Return:
in temp1, SPMCSR
shrs temp1, RWWSB ; If RWWSB is set, the RW section is not ready yet
ret
; re-enable the RW section
ldi spmcrval, (1<<RWNSRE) | (1<<SPMEN)
call Do_spm
rjmp Return

Do_spm:
; check for previous SPM complete
Wait_spm:
in temp1, SPMCSR
sbr temp1, SPMEN
rjmp Wait_spm
; input: spmcrval determines SPM action
; disable interrupts if enabled, store status
in temp2, SREG
cli
; check that no EEPROM write access is present
Wait_eb:
sbic EBCR, EEPE
rjmp Wait_eb
; SPM timed sequence
out SPMCSR, spmcrval
spm
; restore SREG (to enable interrupts if originally enabled)
out SREG, temp2
ret
27.7.14 ATmega16U4/ATmega32U4 Boot Loader Parameters

In Table 27-8 through Table 27-10, the parameters used in the description of the Self-Programming are given.

Table 27-8. Boot Size Configuration (Word Addresses)\(^{(1)}\)

<table>
<thead>
<tr>
<th>Device</th>
<th>BOOTSZ1</th>
<th>BOOTSZ0</th>
<th>Boot Size</th>
<th>Application Flash Section</th>
<th>Boot Loader Flash Section</th>
<th>End Application Section</th>
<th>Boot Reset Address (Start Boot Loader Section)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ATmega32U4</td>
<td>1 1</td>
<td></td>
<td>256 words</td>
<td>4</td>
<td>0x0000 - 0x3EFF</td>
<td>0x3F00 - 0x3FF</td>
<td>0x3EFF</td>
</tr>
<tr>
<td></td>
<td>1 0</td>
<td></td>
<td>512 words</td>
<td>8</td>
<td>0x0000 - 0x3DFF</td>
<td>0x3E00 - 0x3FF</td>
<td>0x3DFF</td>
</tr>
<tr>
<td></td>
<td>0 1</td>
<td></td>
<td>1024 words</td>
<td>16</td>
<td>0x0000 - 0x3BFF</td>
<td>0x3C00 - 0x3FF</td>
<td>0x3BFF</td>
</tr>
<tr>
<td></td>
<td>0 0</td>
<td></td>
<td>2048 words</td>
<td>32</td>
<td>0x0000 - 0x37FF</td>
<td>0x3800 - 0x3FF</td>
<td>0x37FF</td>
</tr>
<tr>
<td>ATmega16U4</td>
<td>1 1</td>
<td></td>
<td>256 words</td>
<td>4</td>
<td>0x0000 - 0x1EFF</td>
<td>0x1F00 - 0x1FF</td>
<td>0x1EFF</td>
</tr>
<tr>
<td></td>
<td>1 0</td>
<td></td>
<td>512 words</td>
<td>8</td>
<td>0x0000 - 0x1DFF</td>
<td>0x1E00 - 0x1FF</td>
<td>0x1DFF</td>
</tr>
<tr>
<td></td>
<td>0 1</td>
<td></td>
<td>1024 words</td>
<td>16</td>
<td>0x0000 - 0x1BFF</td>
<td>0x1C00 - 0x1FF</td>
<td>0x1BFF</td>
</tr>
<tr>
<td></td>
<td>0 0</td>
<td></td>
<td>2048 words</td>
<td>32</td>
<td>0x0000 - 0x17FF</td>
<td>0x1800 - 0x1FF</td>
<td>0x1800</td>
</tr>
</tbody>
</table>

Note: 1. The different BOOTSZ Fuse configurations are shown in Figure 27-2

Table 27-9. Read-While-Write Limit (Word Addresses)\(^{(1)}\)

<table>
<thead>
<tr>
<th>Device</th>
<th>Section</th>
<th>Pages</th>
<th>Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>ATmega32U4</td>
<td>Read-While-Write section (RWW)</td>
<td>224</td>
<td>0x0000 - 0x37FF</td>
</tr>
<tr>
<td></td>
<td>No Read-While-Write section (NRWW)</td>
<td>32</td>
<td>0x3800 - 0x3FF</td>
</tr>
<tr>
<td>ATmega16U4</td>
<td>Read-While-Write section (RWW)</td>
<td>97</td>
<td>0x0000 - 0x17FF</td>
</tr>
<tr>
<td></td>
<td>No Read-While-Write section (NRWW)</td>
<td>32</td>
<td>0x1800 - 0x1FF</td>
</tr>
</tbody>
</table>

Note: 1. For details about these two section, see “NRWW – No Read-While-Write Section” on page 331 and “RWW – Read-While-Write Section” on page 331.
Table 27-10. Explanation of different variables used in Figure 27-4 and the mapping to the Z-pointer

<table>
<thead>
<tr>
<th>Variable</th>
<th>Corresponding Z-value&lt;sup&gt;(3)&lt;/sup&gt;</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PCMSB</td>
<td>13</td>
<td>Most significant bit in the Program Counter. (The Program Counter is 14 bits PC[13:0])</td>
</tr>
<tr>
<td>PAGEMSB</td>
<td>6</td>
<td>Most significant bit which is used to address the words within one page (64 words in a page requires six bits PC [5:0]).</td>
</tr>
<tr>
<td>ZPCMSB</td>
<td>Z14</td>
<td>Bit in Z-pointer that is mapped to PCMSB. Because Z0 is not used, the ZPCMSB equals PCMSB + 1.</td>
</tr>
<tr>
<td>ZPAGEMSB</td>
<td>Z7</td>
<td>Bit in Z-pointer that is mapped to PCMSB. Because Z0 is not used, the ZPAGEMSB equals PAGEMSB + 1.</td>
</tr>
<tr>
<td>PCPAGE</td>
<td>PC[13:6]</td>
<td>Program Counter page address: Page select, for Page Erase and Page Write</td>
</tr>
<tr>
<td>PCWORD</td>
<td>PC[5:0]</td>
<td>Program Counter word address: Word select, for filling temporary buffer (must be zero during Page Write operation)</td>
</tr>
</tbody>
</table>

Note: 1. Z0: should be zero for all SPM commands, byte select for the (E)LPM instruction.

Note: See “Addressing the Flash During Self-Programming” on page 337 for details about the use of Z-pointer during Self-Programming.
28. Memory Programming

28.1 Program And Data Memory Lock Bits

The ATmega16U4/ATmega32U4 provides six Lock bits which can be left unprogrammed (“1”) or can be programmed (“0”) to obtain the additional features listed in Table 28-2. The Lock bits can only be erased to “1” with the Chip Erase command.

Table 28-1. Lock Bit Byte\(^{(1)}\)

<table>
<thead>
<tr>
<th>Lock Bit Byte</th>
<th>Bit No</th>
<th>Description</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>7</td>
<td>–</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>–</td>
<td>1</td>
</tr>
<tr>
<td>BLB12</td>
<td>5</td>
<td>Boot Lock bit</td>
<td>1</td>
</tr>
<tr>
<td>BLB11</td>
<td>4</td>
<td>Boot Lock bit</td>
<td>0</td>
</tr>
<tr>
<td>BLB02</td>
<td>3</td>
<td>Boot Lock bit</td>
<td>1</td>
</tr>
<tr>
<td>BLB01</td>
<td>2</td>
<td>Boot Lock bit</td>
<td>1</td>
</tr>
<tr>
<td>LB2</td>
<td>1</td>
<td>Lock bit</td>
<td>0</td>
</tr>
<tr>
<td>LB1</td>
<td>0</td>
<td>Lock bit</td>
<td>1</td>
</tr>
</tbody>
</table>

Note: 1. “1”: unprogrammed, “0”: programmed

Table 28-2. Lock Bit Protection Modes\(^{(1)(2)}\)

<table>
<thead>
<tr>
<th>Memory Lock Bits</th>
<th>Protection Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>LB Mode</td>
<td>LB2</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>BLB0 Mode</td>
<td>BLB02</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
</tr>
</tbody>
</table>
Table 28-2.  Lock Bit Protection Modes\textsuperscript{(1)(2)} (Continued)

<table>
<thead>
<tr>
<th>Memory Lock Bits</th>
<th>Protection Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>BLB1 Mode</td>
<td>BLB12</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
</tr>
</tbody>
</table>

Notes: 1. Program the Fuse bits and Boot Lock bits before programming the LB1 and LB2. 2. “1” means unprogrammed, “0” means programmed

28.2 Fuse Bits

The ATmega16U4/ATmega32U4 has three bytes. Table 28-3 - Table 28-5 describe briefly the functionality of all the fuses and how they are mapped into the Fuse bytes. Note that the fuses are read as logical zero, “0”, if they are programmed.

Table 28-3.  Extended Fuse Byte\textsuperscript{(2)}

<table>
<thead>
<tr>
<th>Fuse Low Byte</th>
<th>Bit No</th>
<th>Description</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>–</td>
<td>7</td>
<td>–</td>
<td>1</td>
</tr>
<tr>
<td>–</td>
<td>6</td>
<td>–</td>
<td>1</td>
</tr>
<tr>
<td>–</td>
<td>5</td>
<td>–</td>
<td>1</td>
</tr>
<tr>
<td>–</td>
<td>4</td>
<td>–</td>
<td>1</td>
</tr>
<tr>
<td>HWBE</td>
<td>3</td>
<td>Hardware Boot Enable</td>
<td>0 (programmed) 1 (unprogrammed)</td>
</tr>
<tr>
<td>BODLEVEL2\textsuperscript{(1)}</td>
<td>2</td>
<td>Brown-out Detector trigger level</td>
<td>0 (programmed)</td>
</tr>
<tr>
<td>BODLEVEL1\textsuperscript{(1)}</td>
<td>1</td>
<td>Brown-out Detector trigger level</td>
<td>1 (unprogrammed)</td>
</tr>
<tr>
<td>BODLEVEL0\textsuperscript{(1)}</td>
<td>0</td>
<td>Brown-out Detector trigger level</td>
<td>1 (unprogrammed)</td>
</tr>
</tbody>
</table>

Notes: 1. See Table 8-2 on page 52 for BODLEVEL Fuse decoding. 2. “1” means unprogrammed, “0” means programmed
Table 28-4. Fuse High Byte

<table>
<thead>
<tr>
<th>Fuse High Byte</th>
<th>Bit No</th>
<th>Description</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>OCDEN(4)</td>
<td>7</td>
<td>Enable OCD</td>
<td>1 (unprogrammed, OCD disabled)</td>
</tr>
<tr>
<td>JTAGEN</td>
<td>6</td>
<td>Enable JTAG</td>
<td>0 (programmed, JTAG enabled)</td>
</tr>
<tr>
<td>SPIEN(1)</td>
<td>5</td>
<td>Enable Serial Program and Data Downloading</td>
<td>0 (programmed, SPI prog. enabled)</td>
</tr>
<tr>
<td>WDTON(3)</td>
<td>4</td>
<td>Watchdog Timer always on</td>
<td>1 (unprogrammed)</td>
</tr>
<tr>
<td>EESAVE</td>
<td>3</td>
<td>EEPROM memory is preserved through the Chip Erase</td>
<td>1 (unprogrammed, EEPROM preserved)</td>
</tr>
<tr>
<td>BOOTSZ1</td>
<td>2</td>
<td>Select Boot Size (see Table 28-7 for details)</td>
<td>0 (programmed)(2)</td>
</tr>
<tr>
<td>BOOTSZ0</td>
<td>1</td>
<td>Select Boot Size (see Table 28-7 for details)</td>
<td>0 (programmed)(2)</td>
</tr>
<tr>
<td>BOOTRST</td>
<td>0</td>
<td>Select Bootloader Address as Reset Vector</td>
<td>1 (unprogrammed, Reset vector @0x0000)</td>
</tr>
</tbody>
</table>

Note:  1. The SPIEN Fuse is not accessible in serial programming mode.
   2. The default value of BOOTSZ1..0 results in maximum Boot Size. See Table 28-8 on page 344 for details.
   3. See "Watchdog Timer" on page 55 for details.
   4. Never ship a product with the OCDEN Fuse programmed regardless of the setting of Lock bits and JTAGEN Fuse. A programmed OCDEN Fuse enables some parts of the clock system to be running in all sleep modes. This may increase the power consumption.

Table 28-5. Fuse Low Byte

<table>
<thead>
<tr>
<th>Fuse Low Byte</th>
<th>Bit Nr</th>
<th>Description</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>ATmega16U4/32U4</td>
</tr>
<tr>
<td>CKDIV8(3)</td>
<td>7</td>
<td>Divide clock by 8</td>
<td>0 (programmed)</td>
</tr>
<tr>
<td>CKOUT(2)</td>
<td>6</td>
<td>Clock output</td>
<td>1 (unprogrammed)</td>
</tr>
<tr>
<td>SUT1</td>
<td>5</td>
<td>Select start-up time</td>
<td>0 (programmed)</td>
</tr>
<tr>
<td>SUT0</td>
<td>4</td>
<td>Select start-up time</td>
<td>1 (unprogrammed)</td>
</tr>
<tr>
<td>CKSEL3</td>
<td>3</td>
<td>Select Clock source</td>
<td>1 (unprogrammed)(1)</td>
</tr>
<tr>
<td>CKSEL2</td>
<td>2</td>
<td>Select Clock source</td>
<td>1 (unprogrammed)(1)</td>
</tr>
<tr>
<td>CKSEL1</td>
<td>1</td>
<td>Select Clock source</td>
<td>1 (unprogrammed)(1)</td>
</tr>
<tr>
<td>CKSEL0</td>
<td>0</td>
<td>Select Clock source</td>
<td>0 (programmed)(1)</td>
</tr>
</tbody>
</table>

Note:  1. The default setting of CKSEL3..0 results in Low Power Crystal Oscillator for ATmega16U4 and ATmega32U4, and Internal RC oscillator for ATmega16U4RC and ATmega32U4RC. See Table 6-1 on page 28 for details.
   2. The CKOUT Fuse allow the system clock to be output on PORTC7. See "Clock Output Buffer" on page 37 for details.
   3. See "System Clock Prescaler" on page 37 for details.

The status of the Fuse bits is not affected by Chip Erase. Note that the Fuse bits are locked if Lock bit1 (LB1) is programmed. Program the Fuse bits before programming the Lock bits.
28.2.1 Latching of Fuses

The fuse values are latched when the device enters programming mode and changes of the fuse values will have no effect until the part leaves Programming mode. This does not apply to the EESAVE Fuse which will take effect once it is programmed. The fuses are also latched on Power-up in Normal mode.

28.3 Signature Bytes

All ATMEG microcontrollers have a three-byte signature code which identifies the device. This code can be read in both serial and parallel mode, also when the device is locked. The three bytes reside in a separate address space.

ATmega16U4 Signature Bytes:
1. 0x000: 0x1E (indicates manufactured by ATMEL).
2. 0x001: 0x94 (indicates 16KB Flash memory).
3. 0x002: 0x88 (indicates ATmega16U4 device).

ATmega32U4 Signature Bytes:
1. 0x000: 0x1E (indicates manufactured by ATMEL).
2. 0x001: 0x95 (indicates 32KB Flash memory).
3. 0x002: 0x87 (indicates ATmega32U4 device).

28.4 Calibration Byte

The ATmega16U4/ATmega32U4 has a byte calibration value for the internal RC Oscillator. This byte resides in the high byte of address 0x000 in the signature address space. During reset, this byte is automatically written into the OSCCAL Register to ensure correct frequency of the calibrated RC Oscillator.

28.5 Parallel Programming Parameters, Pin Mapping, and Commands

This section describes how to parallel program and verify Flash Program memory, EEPROM Data memory, Memory Lock bits, and Fuse bits in the ATmega16U4/ATmega32U4. Pulses are assumed to be at least 250 ns unless otherwise noted.

28.5.1 Signal Names

In this section, some pins of the ATmega16U4/ATmega32U4 are referenced by signal names describing their functionality during parallel programming, see Figure 28-1 and Table 28-6. Pins not described in the following table are referenced by pin names.

The XA1/XA0 pins determine the action executed when the XTAL1 pin is given a positive pulse. The bit coding is shown in Table 28-9.

When pulsing WR or OE, the command loaded determines the action executed. The different commands are shown in Table 28-10.
Figure 28-1. Parallel Programming(1)

Note: 1. Unused Pins should be left floating.

Table 28-6. Pin Name Mapping

<table>
<thead>
<tr>
<th>Signal Name in Programming Mode</th>
<th>Pin Name</th>
<th>I/O</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>RDY/BSY</td>
<td>PD1</td>
<td>O</td>
<td>0: Device is busy programming, 1: Device is ready for new command.</td>
</tr>
<tr>
<td>OE</td>
<td>PD2</td>
<td>I</td>
<td>Output Enable (Active low).</td>
</tr>
<tr>
<td>WR</td>
<td>PD3</td>
<td>I</td>
<td>Write Pulse (Active low).</td>
</tr>
<tr>
<td>BS1</td>
<td>PD4</td>
<td>I</td>
<td>Byte Select 1.</td>
</tr>
<tr>
<td>XA0</td>
<td>PD5</td>
<td>I</td>
<td>XTAL Action Bit 0</td>
</tr>
<tr>
<td>XA1</td>
<td>PD6</td>
<td>I</td>
<td>XTAL Action Bit 1</td>
</tr>
<tr>
<td>PAGEL</td>
<td>PD7</td>
<td>I</td>
<td>Program Memory and EEPROM data Page Load.</td>
</tr>
<tr>
<td>BS2</td>
<td>PE6</td>
<td>I</td>
<td>Byte Select 2.</td>
</tr>
<tr>
<td>DATA</td>
<td>PB7-0</td>
<td>I/O</td>
<td>Bi-directional Data bus (Output when OE is low).</td>
</tr>
</tbody>
</table>

Table 28-7. BS2 and BS1 Encoding

<table>
<thead>
<tr>
<th>BS2</th>
<th>BS1</th>
<th>Flash / EEPROM Address</th>
<th>Flash Data Loading / Reading</th>
<th>Fuse Programming</th>
<th>Reading Fuse and Lock Bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>Low Byte</td>
<td>Low Byte</td>
<td>Low Byte</td>
<td>Fuse Low Byte</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>High Byte</td>
<td>High Byte</td>
<td>High Byte</td>
<td>Lock bits</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>Extended High Byte</td>
<td>Reserved</td>
<td>Extended Byte</td>
<td>Extended Fuse Byte</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>Reserved</td>
<td>Reserved</td>
<td>Reserved</td>
<td>Fuse High Byte</td>
</tr>
</tbody>
</table>
Table 28-8. Pin Values Used to Enter Programming Mode

<table>
<thead>
<tr>
<th>Pin</th>
<th>Symbol</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>PAGEL</td>
<td>Prog_enable[3]</td>
<td>0</td>
</tr>
<tr>
<td>XA1</td>
<td>Prog_enable[2]</td>
<td>0</td>
</tr>
<tr>
<td>XA0</td>
<td>Prog_enable[1]</td>
<td>0</td>
</tr>
<tr>
<td>BS1</td>
<td>Prog_enable[0]</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 28-9. XA1 and XA0 Encoding

<table>
<thead>
<tr>
<th>XA1</th>
<th>XA0</th>
<th>Action when XTAL1 is Pulsed</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>Load Flash or EEPROM Address (High or low address byte determined by BS2 and BS1).</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>Load Data (High or Low data byte for Flash determined by BS1).</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>Load Command</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>No Action, Idle</td>
</tr>
</tbody>
</table>

Table 28-10. Command Byte Bit Encoding

<table>
<thead>
<tr>
<th>Command Byte</th>
<th>Command Executed</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000 0000</td>
<td>Chip Erase</td>
</tr>
<tr>
<td>0100 0000</td>
<td>Write Fuse bits</td>
</tr>
<tr>
<td>0010 0000</td>
<td>Write Lock bits</td>
</tr>
<tr>
<td>0001 0000</td>
<td>Write Flash</td>
</tr>
<tr>
<td>0001 0001</td>
<td>Write EEPROM</td>
</tr>
<tr>
<td>0000 1000</td>
<td>Read Signature Bytes and Calibration byte</td>
</tr>
<tr>
<td>0000 0100</td>
<td>Read Fuse and Lock bits</td>
</tr>
<tr>
<td>0000 0010</td>
<td>Read Flash</td>
</tr>
<tr>
<td>0000 0011</td>
<td>Read EEPROM</td>
</tr>
</tbody>
</table>

Table 28-11. No. of Words in a Page and No. of Pages in the Flash

<table>
<thead>
<tr>
<th>Flash Size</th>
<th>Page Size</th>
<th>PCWORD</th>
<th>No. of Pages</th>
<th>PCPAGE</th>
<th>PCMSB</th>
</tr>
</thead>
</table>
Table 28-12. No. of Words in a Page and No. of Pages in the EEPROM

<table>
<thead>
<tr>
<th>EEPROM Size</th>
<th>Page Size</th>
<th>PCWORD</th>
<th>No. of Pages</th>
<th>PCPAGE</th>
<th>EEAMSB</th>
</tr>
</thead>
</table>

28.6 Parallel Programming

28.6.1 Enter Programming Mode

The following algorithm puts the device in parallel programming mode:

1. Apply 4.5 - 5.5V between VCC and GND.
2. Set RESET to “0” and toggle XTAL1 at least six times.
3. Set the Prog_enable pins listed in Table 28-8 on page 351 to “0000” and wait at least 100 ns.
4. Apply 11.5 - 12.5V to RESET. Any activity on Prog_enable pins within 100 ns after +12V has been applied to RESET, will cause the device to fail entering programming mode.
5. Wait at least 50 µs before sending a new command.

28.6.2 Considerations for Efficient Programming

The loaded command and address are retained in the device during programming. For efficient programming, the following should be considered:

- The command needs only be loaded once when writing or reading multiple memory locations.
- Skip writing the data value 0xFF, that is the contents of the entire EEPROM (unless the EESAVE Fuse is programmed) and Flash after a Chip Erase.
- Address high byte needs only be loaded before programming or reading a new 256 word window in Flash or 256 byte EEPROM. This consideration also applies to Signature bytes reading.

28.6.3 Chip Erase

The Chip Erase will erase the Flash and EEPROM\(^{(1)}\) memories plus Lock bits. The Lock bits are not reset until the program memory has been completely erased. The Fuse bits are not changed. A Chip Erase must be performed before the Flash and/or EEPROM are reprogrammed.

Note: 1. The EEPROM memory is preserved during Chip Erase if the EESAVE Fuse is programmed.

Load Command “Chip Erase”

1. Set XA1, XA0 to “10”. This enables command loading.
2. Set BS1 to “0”.
3. Set DATA to “1000 0000”. This is the command for Chip Erase.
4. Give XTAL1 a positive pulse. This loads the command.
5. Give WR a negative pulse. This starts the Chip Erase. RDY/BSY goes low.
6. Wait until RDY/BSY goes high before loading a new command.
28.6.4 Programming the Flash

The Flash is organized in pages, see Table 28-11 on page 351. When programming the Flash, the program data is latched into a page buffer. This allows one page of program data to be programmed simultaneously. The following procedure describes how to program the entire Flash memory:

A. Load Command “Write Flash”
   1. Set XA1, XA0 to “10”. This enables command loading.
   2. Set BS1 to “0”.
   3. Set DATA to “0001 0000”. This is the command for Write Flash.
   4. Give XTAL1 a positive pulse. This loads the command.

B. Load Address Low byte (Address bits 7..0)
   1. Set XA1, XA0 to “00”. This enables address loading.
   2. Set BS2, BS1 to “00”. This selects the address low byte.
   3. Set DATA = Address low byte (0x00 - 0xFF).
   4. Give XTAL1 a positive pulse. This loads the address low byte.

C. Load Data Low Byte
   1. Set XA1, XA0 to “01”. This enables data loading.
   2. Set DATA = Data low byte (0x00 - 0xFF).
   3. Give XTAL1 a positive pulse. This loads the data byte.

D. Load Data High Byte
   1. Set BS1 to “1”. This selects high data byte.
   2. Set XA1, XA0 to “01”. This enables data loading.
   3. Set DATA = Data high byte (0x00 - 0xFF).
   4. Give XTAL1 a positive pulse. This loads the data byte.

E. Latch Data
   1. Set BS1 to “1”. This selects high data byte.
   2. Give PAGEL a positive pulse. This latches the data bytes. (See Figure 28-3 for signal waveforms)

F. Repeat B through E until the entire buffer is filled or until all data within the page is loaded.

While the lower bits in the address are mapped to words within the page, the higher bits address the pages within the FLASH. This is illustrated in Figure 28-2 on page 354. Note that if less than eight bits are required to address words in the page (pagesize < 256), the most significant bit(s) in the address low byte are used to address the page when performing a Page Write.

G. Load Address High byte (Address bits15..8)
   1. Set XA1, XA0 to “00”. This enables address loading.
   2. Set BS2, BS1 to “01”. This selects the address high byte.
   3. Set DATA = Address high byte (0x00 - 0xFF).
   4. Give XTAL1 a positive pulse. This loads the address high byte.

H. Load Address Extended High byte (Address bits 23..16)
   1. Set XA1, XA0 to “00”. This enables address loading.
   2. Set BS2, BS1 to “10”. This selects the address extended high byte.
3. Set DATA = Address extended high byte (0x00 - 0xFF).
4. Give XTAL1 a positive pulse. This loads the address high byte.

I. Program Page

1. Set BS2, BS1 to “00”
2. Give WR a negative pulse. This starts programming of the entire page of data. RDY/BSY goes low.
3. Wait until RDY/BSY goes high (See Figure 28-3 for signal waveforms).

J. Repeat B through I until the entire Flash is programmed or until all data has been programmed.

K. End Page Programming

1. 1. Set XA1, XA0 to “10”. This enables command loading.
2. Set DATA to “0000 0000”. This is the command for No Operation.
3. Give XTAL1 a positive pulse. This loads the command, and the internal write signals are reset.

Figure 28-2. Addressing the Flash Which is Organized in Pages

Note: 1. PCPAGE and PCWORD are listed in Table 28-11 on page 351.
Figure 28-3. Programming the Flash Waveforms\(^{(1)}\)

Note: 1. “XX” is don’t care. The letters refer to the programming description above.

28.6.5 Programming the EEPROM

The EEPROM is organized in pages, see Table 28-12 on page 352. When programming the EEPROM, the program data is latched into a page buffer. This allows one page of data to be programmed simultaneously. The programming algorithm for the EEPROM data memory is as follows (refer to “Programming the Flash” on page 353 for details on Command, Address and Data loading):

1. A: Load Command “0001 0001”.
2. G: Load Address High Byte (0x00 - 0xFF).
3. B: Load Address Low Byte (0x00 - 0xFF).
4. C: Load Data (0x00 - 0xFF).
5. E: Latch data (give PAGEL a positive pulse).
K: Repeat 3 through 5 until the entire buffer is filled.

L: Program EEPROM page

1. Set BS2, BS1 to “00”.
2. Give WR a negative pulse. This starts programming of the EEPROM page. RDY/BSY goes low.
3. Wait until RDY/BSY goes high before programming the next page (See Figure 28-4 for signal waveforms).
Figure 28-4. Programming the EEPROM Waveforms

28.6.6 Reading the Flash

The algorithm for reading the Flash memory is as follows (refer to “Programming the Flash” on page 353 for details on Command and Address loading):

1. A: Load Command “0000 0010”.
2. H: Load Address Extended Byte (0x00-0xFF).
3. G: Load Address High Byte (0x00 - 0xFF).
4. B: Load Address Low Byte (0x00 - 0xFF).
5. Set OE to “0”, and BS1 to “0”. The Flash word low byte can now be read at DATA.
6. Set BS to “1”. The Flash word high byte can now be read at DATA.
7. Set OE to “1”.

28.6.7 Reading the EEPROM

The algorithm for reading the EEPROM memory is as follows (refer to “Programming the Flash” on page 353 for details on Command and Address loading):

1. A: Load Command “0000 0011”.
2. G: Load Address High Byte (0x00 - 0xFF).
3. B: Load Address Low Byte (0x00 - 0xFF).
4. Set OE to “0”, and BS1 to “0”. The EEPROM Data byte can now be read at DATA.
5. Set OE to “1”.

28.6.8 Programming the Fuse Low Bits

The algorithm for programming the Fuse Low bits is as follows (refer to “Programming the Flash” on page 353 for details on Command and Data loading):

1. A: Load Command “0100 0000”.
2. C: Load Data Low Byte. Bit n = “0” programs and bit n = “1” erases the Fuse bit.
3. Give WR a negative pulse and wait for RDY/BSY to go high.
28.6.9 Programming the Fuse High Bits

The algorithm for programming the Fuse High bits is as follows (refer to “Programming the Flash” on page 353 for details on Command and Data loading):

1. A: Load Command “0100 0000”.
2. C: Load Data Low Byte. Bit n = “0” programs and bit n = “1” erases the Fuse bit.
3. Set BS2, BS1 to “01”. This selects high data byte.
4. Give WR a negative pulse and wait for RDY/BSY to go high.
5. Set BS2, BS1 to “00”. This selects low data byte.

28.6.10 Programming the Extended Fuse Bits

The algorithm for programming the Extended Fuse bits is as follows (refer to “Programming the Flash” on page 353 for details on Command and Data loading):

1. 1. A: Load Command “0100 0000”.
2. 2. C: Load Data Low Byte. Bit n = “0” programs and bit n = “1” erases the Fuse bit.
3. 3. Set BS2, BS1 to “10”. This selects extended data byte.
4. 4. Give WR a negative pulse and wait for RDY/BSY to go high.
5. 5. Set BS2, BS1 to “00”. This selects low data byte.

Figure 28-5. Programming the FUSES Waveforms

28.6.11 Programming the Lock Bits

The algorithm for programming the Lock bits is as follows (refer to “Programming the Flash” on page 353 for details on Command and Data loading):

1. A: Load Command “0010 0000”.
2. C: Load Data Low Byte. Bit n = “0” programs the Lock bit. If LB mode 3 is programmed (LB1 and LB2 is programmed), it is not possible to program the Boot Lock bits by any External Programming mode.
3. Give WR a negative pulse and wait for RDY/BSY to go high.

The Lock bits can only be cleared by executing Chip Erase.
28.6.12 Reading the Fuse and Lock Bits

The algorithm for reading the Fuse and Lock bits is as follows (refer to “Programming the Flash” on page 353 for details on Command loading):

1. A: Load Command “0000 0100”.
2. Set $\overline{OE}$ to “0”, and BS2, BS1 to “00”. The status of the Fuse Low bits can now be read at DATA (“0” means programmed).
3. Set $\overline{OE}$ to “0”, and BS2, BS1 to “11”. The status of the Fuse High bits can now be read at DATA (“0” means programmed).
4. Set $\overline{OE}$ to “0”, and BS2, BS1 to “10”. The status of the Extended Fuse bits can now be read at DATA (“0” means programmed).
5. Set $\overline{OE}$ to “0”, and BS2, BS1 to “01”. The status of the Lock bits can now be read at DATA (“0” means programmed).
6. Set $\overline{OE}$ to “1”.

Figure 28-6. Mapping Between BS1, BS2 and the Fuse and Lock Bits During Read

28.6.13 Reading the Signature Bytes

The algorithm for reading the Signature bytes is as follows (refer to “Programming the Flash” on page 353 for details on Command and Address loading):

1. A: Load Command “0000 1000”.
2. B: Load Address Low Byte (0x00 - 0x02).
3. Set $\overline{OE}$ to “0”, and BS to “0”. The selected Signature byte can now be read at DATA.
4. Set $\overline{OE}$ to “1”.

28.6.14 Reading the Calibration Byte

The algorithm for reading the Calibration byte is as follows (refer to “Programming the Flash” on page 353 for details on Command and Address loading):

1. A: Load Command “0000 1000”.
2. B: Load Address Low Byte, 0x00.
3. Set $\overline{OE}$ to “0”, and BS1 to “1”. The Calibration byte can now be read at DATA.
4. Set $\overline{OE}$ to “1”.
28.6.15  Parallel Programming Characteristics

Figure 28-7. Parallel Programming Timing, Including some General Timing Requirements

Figure 28-8. Parallel Programming Timing, Loading Sequence with Timing Requirements

Note: 1. The timing requirements shown in Figure 28-7 (i.e., tDVXH, tXHXL, and tXLDX) also apply to loading operation.

Figure 28-9. Parallel Programming Timing, Reading Sequence (within the Same Page) with Timing Requirements

359
Note: 1. The timing requirements shown in Figure 28-7 (i.e., \( t_{DVXH} \), \( t_{NHXL} \), and \( t_{HLX} \)) also apply to reading operation.

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter</th>
<th>Min</th>
<th>Typ</th>
<th>Max</th>
<th>Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>( V_{PP} )</td>
<td>Programming Enable Voltage</td>
<td>11.5</td>
<td>12.5</td>
<td></td>
<td>V</td>
</tr>
<tr>
<td>( I_{PP} )</td>
<td>Programming Enable Current</td>
<td></td>
<td></td>
<td>250</td>
<td>( \mu A )</td>
</tr>
<tr>
<td>( t_{DVXH} )</td>
<td>Data and Control Valid before XTAL1 High</td>
<td>67</td>
<td></td>
<td></td>
<td>ns</td>
</tr>
<tr>
<td>( t_{NHXH} )</td>
<td>XTAL1 Low to XTAL1 High</td>
<td>200</td>
<td></td>
<td></td>
<td>ns</td>
</tr>
<tr>
<td>( t_{NHXL} )</td>
<td>XTAL1 Pulse Width High</td>
<td>150</td>
<td></td>
<td></td>
<td>ns</td>
</tr>
<tr>
<td>( t_{HLX} )</td>
<td>Data and Control Hold after XTAL1 Low</td>
<td>67</td>
<td></td>
<td></td>
<td>ns</td>
</tr>
<tr>
<td>( t_{HLWL} )</td>
<td>XTAL1 Low to ( WR ) Low</td>
<td>0</td>
<td></td>
<td></td>
<td>ns</td>
</tr>
<tr>
<td>( t_{NHPL} )</td>
<td>XTAL1 Low to PAGEL high</td>
<td>0</td>
<td></td>
<td></td>
<td>ns</td>
</tr>
<tr>
<td>( t_{PLXH} )</td>
<td>PAGEL low to XTAL1 high</td>
<td>150</td>
<td></td>
<td></td>
<td>ns</td>
</tr>
<tr>
<td>( t_{BVPH} )</td>
<td>BS1 Valid before PAGEL High</td>
<td>67</td>
<td></td>
<td></td>
<td>ns</td>
</tr>
<tr>
<td>( t_{PHPL} )</td>
<td>PAGEL Pulse Width High</td>
<td>150</td>
<td></td>
<td></td>
<td>ns</td>
</tr>
<tr>
<td>( t_{PLPL} )</td>
<td>BS1 Hold after PAGEL Low</td>
<td>67</td>
<td></td>
<td></td>
<td>ns</td>
</tr>
<tr>
<td>( t_{WLBX} )</td>
<td>BS2/1 Hold after ( WR ) Low</td>
<td>67</td>
<td></td>
<td></td>
<td>ns</td>
</tr>
<tr>
<td>( t_{PLWL} )</td>
<td>PAGEL Low to ( WR ) Low</td>
<td>67</td>
<td></td>
<td></td>
<td>ns</td>
</tr>
<tr>
<td>( t_{BVWL} )</td>
<td>BS2/1 Valid to ( WR ) Low</td>
<td>67</td>
<td></td>
<td></td>
<td>ns</td>
</tr>
<tr>
<td>( t_{VLWH} )</td>
<td>( WR ) Pulse Width Low</td>
<td>150</td>
<td></td>
<td></td>
<td>ns</td>
</tr>
<tr>
<td>( t_{VLRL} )</td>
<td>( WR ) Low to RDY/BSY Low</td>
<td>0</td>
<td></td>
<td>1</td>
<td>( \mu S )</td>
</tr>
<tr>
<td>( t_{VLRH} )</td>
<td>( WR ) Low to RDY/BSY High(1)</td>
<td>3.7</td>
<td></td>
<td>4.5</td>
<td>ms</td>
</tr>
<tr>
<td>( t_{VLRLH} )</td>
<td>( WR ) Low to RDY/BSY High for Chip Erase(2)</td>
<td>7.5</td>
<td></td>
<td>9</td>
<td>ms</td>
</tr>
<tr>
<td>( t_{XOLD} )</td>
<td>XTAL1 Low to ( OE ) Low</td>
<td>0</td>
<td></td>
<td></td>
<td>ns</td>
</tr>
<tr>
<td>( t_{BVDV} )</td>
<td>BS1 Valid to DATA valid</td>
<td>0</td>
<td></td>
<td>250</td>
<td>ns</td>
</tr>
<tr>
<td>( t_{OLED} )</td>
<td>( OE ) Low to DATA Valid</td>
<td>250</td>
<td></td>
<td></td>
<td>ns</td>
</tr>
<tr>
<td>( t_{OHDD} )</td>
<td>( OE ) High to DATA Tri-stated</td>
<td>250</td>
<td></td>
<td></td>
<td>ns</td>
</tr>
</tbody>
</table>

Notes: 1. \( t_{VLRH} \) is valid for the Write Flash, Write EEPROM, Write Fuse bits and Write Lock bits commands.
2. \( t_{VLRLH, CE} \) is valid for the Chip Erase command.

### 28.7 Serial Downloading

Both the Flash and EEPROM memory arrays can be programmed using a serial programming bus while \( \text{RESET} \) is pulled to GND. The serial programming interface consists of pins SCK, PDI (input) and PDO (output). After \( \text{RESET} \) is set low, the Programming Enable instruction needs to be executed first before program/erase operations can be executed. NOTE, in Table 28-14 on page 361, the pin mapping for serial programming is listed. Not all packages use the SPI pins dedicated for the internal Serial Peripheral Interface - SPI.
28.8 Serial Programming Pin Mapping

Table 28-14. Pin Mapping Serial Programming

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Pins (TQFP-64)</th>
<th>I/O</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PDI</td>
<td>PB2</td>
<td>I</td>
<td>Serial Data in</td>
</tr>
<tr>
<td>PDO</td>
<td>PB3</td>
<td>O</td>
<td>Serial Data out</td>
</tr>
<tr>
<td>SCK</td>
<td>PB1</td>
<td>I</td>
<td>Serial Clock</td>
</tr>
</tbody>
</table>

Figure 28-10. Serial Programming and Verify(1)

Notes: 1. If the device is clocked by the internal Oscillator, it is no need to connect a clock source to the XTL1 pin.
2. $V_{CC} - 0.3V < AVCC < V_{CC} + 0.3V$, however, AVCC should always be within 1.8 - 5.5V

When programming the EEPROM, an auto-erase cycle is built into the self-timed programming operation (in the Serial mode ONLY) and there is no need to first execute the Chip Erase instruction. The Chip Erase operation turns the content of every memory location in both the Program and EEPROM arrays into 0xFF.

Depending on CKSEL Fuses, a valid clock must be present. The minimum low and high periods for the serial clock (SCK) input are defined as follows:

Low: > 2 CPU clock cycles for $f_{ck} < 12$ MHz, 3 CPU clock cycles for $f_{ck} \geq 12$ MHz

High: > 2 CPU clock cycles for $f_{ck} < 12$ MHz, 3 CPU clock cycles for $f_{ck} \geq 12$ MHz

28.8.1 Serial Programming Algorithm

When writing serial data to the ATmega16U4/ATmega32U4, data is clocked on the rising edge of SCK.

When reading data from the ATmega16U4/ATmega32U4, data is clocked on the falling edge of SCK. See Figure 28-11 for timing details.

To program and verify the ATmega16U4/ATmega32U4 in the serial programming mode, the following sequence is recommended (See four byte instruction formats in Table 28-16):
1. Power-up sequence:
   Apply power between V_{CC} and GND while RESET and SCK are set to “0”. In some systems, the programmer can not guarantee that SCK is held low during power-up. In this case, RESET must be given a positive pulse of at least two CPU clock cycles duration after SCK has been set to “0”.

2. Wait for at least 20 ms and enable serial programming by sending the Programming Enable serial instruction to pin PDI.

3. The serial programming instructions will not work if the communication is out of synchronization. When in sync. the second byte (0x53), will echo back when issuing the third byte of the Programming Enable instruction. Whether the echo is correct or not, all four bytes of the instruction must be transmitted. If the 0x53 did not echo back, give RESET a positive pulse and issue a new Programming Enable command.

4. The Flash is programmed one page at a time. The memory page is loaded one byte at a time by supplying the 7 LSB of the address and data together with the Load Program Memory Page instruction. To ensure correct loading of the page, the data low byte must be loaded before data high byte is applied for a given address. The Program Memory Page is stored by loading the Write Program Memory Page instruction with the address lines 15..8. Before issuing this command, make sure the instruction Load Extended Address Byte has been used to define the MSB of the address. The extended address byte is stored until the command is re-issued, i.e., the command needs only be issued for the first page, and when crossing the 64KWord boundary. If polling (RDY/BSY) is not used, the user must wait at least t_{WD_FLASH} before issuing the next page. (See Table 28-15.) Accessing the serial programming interface before the Flash write operation completes can result in incorrect programming.

5. The EEPROM array is programmed one byte at a time by supplying the address and data together with the appropriate Write instruction. An EEPROM memory location is first automatically erased before new data is written. If polling is not used, the user must wait at least t_{WD_EEPROM} before issuing the next byte. (See Table 28-15.) In a chip erased device, no 0xFFs in the data file(s) need to be programmed.

6. Any memory location can be verified by using the Read instruction which returns the content at the selected address at serial output PDO. When reading the Flash memory, use the instruction Load Extended Address Byte to define the upper address byte, which is not included in the Read Program Memory instruction. The extended address byte is stored until the command is re-issued, i.e., the command needs only be issued for the first page, and when crossing the 64KWord boundary.

7. At the end of the programming session, RESET can be set high to commence normal operation.

8. Power-off sequence (if needed):
   Set RESET to “1”.
   Turn V_{CC} power off.

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Minimum Wait Delay</th>
</tr>
</thead>
<tbody>
<tr>
<td>t_{WD_FLASH}</td>
<td>4.5 ms</td>
</tr>
<tr>
<td>t_{WD_EEPROM}</td>
<td>9.0 ms</td>
</tr>
<tr>
<td>t_{WD_ERASE}</td>
<td>9.0 ms</td>
</tr>
</tbody>
</table>
Figure 28-11. Serial Programming Waveforms
### Table 28-16. Serial Programming Instruction Set

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Instruction Format</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Programming Enable</td>
<td>1010 1100 0101 0011 xxxx xxxx xxxx xxxx</td>
<td>Enable Serial Programming after RESET goes low.</td>
</tr>
<tr>
<td>Chip Erase</td>
<td>1010 1100 100x xxxx xxxx xxxx xxxx xxxx</td>
<td>Chip Erase EEPROM and Flash.</td>
</tr>
<tr>
<td>Load Extended Address Byte</td>
<td>0100 1101 0000 0000 cccc cccc cccc xxxx xxxx</td>
<td>Defines Extended Address Byte for Read Program Memory and Write Program Memory Page.</td>
</tr>
<tr>
<td>Read Program Memory</td>
<td>0010 H000 aaaa aaaa bbbb bbbb 0000 0000</td>
<td>Read H (high or low) data o from Program memory at word address c.a.b.</td>
</tr>
<tr>
<td>Load Program Memory Page</td>
<td>0100 H000 xxxx xxxx xbbb bbbb iiii iiii</td>
<td>Write H (high or low) data i to Program Memory page at word address b. Data low byte must be loaded before Data high byte is applied within the same address.</td>
</tr>
<tr>
<td>Write Program Memory Page</td>
<td>0100 1100 aaaa aaaa bbxx xxxx xxxx xxxx xxxx</td>
<td>Write Program Memory Page at address c.a.b.</td>
</tr>
<tr>
<td>Read EEPROM Memory</td>
<td>1010 0000 0000 aaaa bbbb bbbb 0000 0000</td>
<td>Read data o from EEPROM memory at address a.b.</td>
</tr>
<tr>
<td>Write EEPROM Memory</td>
<td>1100 0000 0000 aaaa bbbb bbbb iiii iiii</td>
<td>Write data i to EEPROM memory at address a.b.</td>
</tr>
<tr>
<td>Load EEPROM Memory Page (page access)</td>
<td>1100 0001 0000 0000 0000 00bb iiii iiii</td>
<td>Load data i to EEPROM memory page buffer. After data is loaded, program EEPROM page.</td>
</tr>
<tr>
<td>Write EEPROM Memory Page (page access)</td>
<td>1100 0010 0000 aaaa bbbb bb00 xxxx xxxx</td>
<td>Write EEPROM page at address a.b.</td>
</tr>
<tr>
<td>Read Lock bits</td>
<td>0101 1000 0000 0000 xxxx xxxx xxxx oo 0000</td>
<td>Read Lock bits. &quot;0&quot; = programmed, &quot;1&quot; = unprogrammed. See Table 28-1 on page 346 for details.</td>
</tr>
<tr>
<td>Write Lock bits</td>
<td>1010 1100 111x xxxx xxxx xxxx xxxx 1iii iiii</td>
<td>Write Lock bits. Set bits = &quot;0&quot; to program Lock bits. See Table 28-1 on page 346 for details.</td>
</tr>
<tr>
<td>Read Signature Byte</td>
<td>0011 0000 000x xxxx xxxx xx bb 0000 0000</td>
<td>Read Signature Byte o at address b.</td>
</tr>
<tr>
<td>Write Fuse bits</td>
<td>1010 1100 1010 0000 xxxx xxxx iiii iiii</td>
<td>Set bits = &quot;0&quot; to program, &quot;1&quot; to unprogram.</td>
</tr>
<tr>
<td>Write Fuse High bits</td>
<td>1010 1100 1010 1000 xxxx xxxx iiii iiii</td>
<td>Set bits = &quot;0&quot; to program, &quot;1&quot; to unprogram.</td>
</tr>
<tr>
<td>Write Extended Fuse Bits</td>
<td>1010 1100 1010 0100 xxxx xxxx iiii iiii</td>
<td>Set bits = &quot;0&quot; to program, &quot;1&quot; to unprogram. See Table 28-3 on page 347 for details.</td>
</tr>
<tr>
<td>Read Fuse bits</td>
<td>0101 0000 0000 0000 xxxx xxxx 0000 0000</td>
<td>Read Fuse bits. &quot;0&quot; = programmed, &quot;1&quot; = unprogrammed.</td>
</tr>
<tr>
<td>Read Fuse High bits</td>
<td>0101 1000 0000 1000 xxxx xxxx 0000 0000</td>
<td>Read Fuse High bits. &quot;0&quot; = pro- grammed, &quot;1&quot; = unprogrammed.</td>
</tr>
</tbody>
</table>
Table 28-16. Serial Programming Instruction Set (Continued)

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Instruction Format</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read Extended Fuse Bits</td>
<td>0101 0000 0000 1000 xxxxxxx</td>
<td>Read Extended Fuse bits. “0” = programmed, “1” = unprogrammed. See Table 28-3 on page 347 for details.</td>
</tr>
<tr>
<td>Read Calibration Byte</td>
<td>0011 1000 0000 xxxxxxx</td>
<td>Read Calibration Byte</td>
</tr>
<tr>
<td>Poll RDY/BSY</td>
<td>1111 0000 0000 xxxxxxx</td>
<td>If o = “1”, a programming operation is still busy. Wait until this bit returns to “0” before applying another command.</td>
</tr>
</tbody>
</table>

Note: a = address high bits, b = address low bits, c = address extended bits, H = 0 - Low byte, 1 - High Byte, o = data out, i = data in,
   x = don’t care

28.8.2 Serial Programming Characteristics

For characteristics of the Serial Programming module see “SPI Timing Characteristics” on page 383.

28.9 Programming via the JTAG Interface

Programming through the JTAG interface requires control of the four JTAG specific pins: TCK, TMS, TDI, and TDO. Control of the reset and clock pins is not required.

To be able to use the JTAG interface, the JTAGEN Fuse must be programmed. The device is default shipped with the fuse programmed. In addition, the JTD bit in MCUCSR must be cleared. Alternatively, if the JTD bit is set, the external reset can be forced low. Then, the JTD bit will be cleared after two chip clocks, and the JTAG pins are available for programming. This provides a means of using the JTAG pins as normal port pins in Running mode while still allowing In-System Programming via the JTAG interface. Note that this technique can not be used when using the JTAG pins for Boundary-scan or On-chip Debug. In these cases the JTAG pins must be dedicated for this purpose.

During programming the clock frequency of the TCK Input must be less than the maximum frequency of the chip. The System Clock Prescaler can not be used to divide the TCK Clock Input into a sufficiently low frequency.

As a definition in this datasheet, the LSB is shifted in and out first of all Shift Registers.

28.9.1 Programming Specific JTAG Instructions

The Instruction Register is 4-bit wide, supporting up to 16 instructions. The JTAG instructions useful for programming are listed below.

The OPCODE for each instruction is shown behind the instruction name in hex format. The text describes which Data Register is selected as path between TDI and TDO for each instruction.

The Run-Test/Idle state of the TAP controller is used to generate internal clocks. It can also be used as an idle state between JTAG sequences. The state machine sequence for changing the instruction word is shown in Figure 28-12.
28.9.2  **AVR_RESET (0xC)**

The AVR specific public JTAG instruction for setting the AVR device in the Reset mode or taking the device out from the Reset mode. The TAP controller is not reset by this instruction. The one bit Reset Register is selected as Data Register. Note that the reset will be active as long as there is a logic “one” in the Reset Chain. The output from this chain is not latched.

The active states are:
- Shift-DR: The Reset Register is shifted by the TCK input.

28.9.3  **PROG_ENABLE (0x4)**

The AVR specific public JTAG instruction for enabling programming via the JTAG port. The 16-bit Programming Enable Register is selected as Data Register. The active states are the following:
- Shift-DR: The programming enable signature is shifted into the Data Register.
- Update-DR: The programming enable signature is compared to the correct value, and Programming mode is entered if the signature is valid.
28.9.4 PROG_COMMANDS (0x5)
The AVR specific public JTAG instruction for entering programming commands via the JTAG port. The 15-bit Programming Command Register is selected as Data Register. The active states are the following:

- **Capture-DR:** The result of the previous command is loaded into the Data Register.
- **Shift-DR:** The Data Register is shifted by the TCK input, shifting out the result of the previous command and shifting in the new command.
- **Update-DR:** The programming command is applied to the Flash inputs
- **Run-Test/Idle:** One clock cycle is generated, executing the applied command

28.9.5 PROG_PAGELOAD (0x6)
The AVR specific public JTAG instruction to directly load the Flash data page via the JTAG port. An 8-bit Flash Data Byte Register is selected as the Data Register. This is physically the 8 LSBs of the Programming Command Register. The active states are the following:

- **Shift-DR:** The Flash Data Byte Register is shifted by the TCK input.
- **Update-DR:** The content of the Flash Data Byte Register is copied into a temporary register. A write sequence is initiated that within 11 TCK cycles loads the content of the temporary register into the Flash page buffer. The AVR automatically alternates between writing the low and the high byte for each new Update-DR state, starting with the low byte for the first Update-DR encountered after entering the PROG_PAGELOAD command. The Program Counter is pre-incremented before writing the low byte, except for the first written byte. This ensures that the first data is written to the address set up by PROG_COMMANDS, and loading the last location in the page buffer does not make the program counter increment into the next page.

28.9.6 PROG_PAGEREAD (0x7)
The AVR specific public JTAG instruction to directly capture the Flash content via the JTAG port. An 8-bit Flash Data Byte Register is selected as the Data Register. This is physically the 8 LSBs of the Programming Command Register. The active states are the following:

- **Capture-DR:** The content of the selected Flash byte is captured into the Flash Data Byte Register. The AVR automatically alternates between reading the low and the high byte for each new Capture-DR state, starting with the low byte for the first Capture-DR encountered after entering the PROG_PAGEREAD command. The Program Counter is post-incremented after reading each high byte, including the first read byte. This ensures that the first data is captured from the first address set up by PROG_COMMANDS, and reading the last location in the page makes the program counter increment into the next page.
- **Shift-DR:** The Flash Data Byte Register is shifted by the TCK input.

28.9.7 Data Registers
The Data Registers are selected by the JTAG instruction registers described in section “Programming Specific JTAG Instructions” on page 365. The Data Registers relevant for programming operations are:

- **Reset Register**
- **Programming Enable Register**
- **Programming Command Register**
- **Flash Data Byte Register**
28.9.8 Reset Register

The Reset Register is a Test Data Register used to reset the part during programming. It is required to reset the part before entering Programming mode.

A high value in the Reset Register corresponds to pulling the external reset low. The part is reset as long as there is a high value present in the Reset Register. Depending on the Fuse settings for the clock options, the part will remain reset for a Reset Time-out period (refer to “Clock Sources” on page 28) after releasing the Reset Register. The output from this Data Register is not latched, so the reset will take place immediately, as shown in Figure 8-1 on page 50.

28.9.9 Programming Enable Register

The Programming Enable Register is a 16-bit register. The contents of this register is compared to the programming enable signature, binary code 0b1010_0011_0111_0000. When the contents of the register is equal to the programming enable signature, programming via the JTAG port is enabled. The register is reset to 0 on Power-on Reset, and should always be reset when leaving Programming mode.

Figure 28-13. Programming Enable Register

28.9.10 Programming Command Register

The Programming Command Register is a 15-bit register. This register is used to serially shift in programming commands, and to serially shift out the result of the previous command, if any. The JTAG Programming Instruction Set is shown in Table 28-17. The state sequence when shifting in the programming commands is illustrated in Figure 28-15.
**Figure 28-14.** Programming Command Register
Table 28-17. JTAG Programming Instruction
Set:  a = address high bits, b = address low bits, c = address extended bits, H = 0 - Low byte, 1 - High Byte, o = data out, i = data in, x = don't care

<table>
<thead>
<tr>
<th>Instruction</th>
<th>TDI Sequence</th>
<th>TDO Sequence</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>1a. Chip Erase</td>
<td>0100011_10000000</td>
<td>xxxxxxx_xxxxxxxxx</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0110001_10000000</td>
<td>xxxxxxx_xxxxxxxxx</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0110011_10000000</td>
<td>xxxxxxx_xxxxxxxxx</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0110011_10000000</td>
<td>xxxxxxx_xxxxxxxxx</td>
<td></td>
</tr>
<tr>
<td>1b. Poll for Chip Erase Complete</td>
<td>0110011_10000000</td>
<td>xxxxxxxo_xxxxxxxxx</td>
<td>(2)</td>
</tr>
<tr>
<td>2a. Enter Flash Write</td>
<td>0100011_00010000</td>
<td>xxxxxxx_xxxxxxxxx</td>
<td></td>
</tr>
<tr>
<td>2b. Load Address Extended High Byte</td>
<td>0001011_cccccccc</td>
<td>xxxxxxx_xxxxxxxxx</td>
<td>(10)</td>
</tr>
<tr>
<td>2c. Load Address High Byte</td>
<td>0001011_aaaaaaaaa</td>
<td>xxxxxxx_xxxxxxxxx</td>
<td></td>
</tr>
<tr>
<td>2d. Load Address Low Byte</td>
<td>0000011_bbbbbbbb</td>
<td>xxxxxxx_xxxxxxxxx</td>
<td></td>
</tr>
<tr>
<td>2e. Load Data Low Byte</td>
<td>0010011_iiiiiiii</td>
<td>xxxxxxx_xxxxxxxxx</td>
<td></td>
</tr>
<tr>
<td>2f. Load Data High Byte</td>
<td>0010111_iiiiiiii</td>
<td>xxxxxxx_xxxxxxxxx</td>
<td></td>
</tr>
<tr>
<td>2g. Latch Data</td>
<td>0110111_00000000</td>
<td>xxxxxxx_xxxxxxxxx</td>
<td>(1)</td>
</tr>
<tr>
<td></td>
<td>1110111_00000000</td>
<td>xxxxxxx_xxxxxxxxx</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0110111_00000000</td>
<td>xxxxxxx_xxxxxxxxx</td>
<td></td>
</tr>
<tr>
<td>2h. Write Flash Page</td>
<td>0110111_00000000</td>
<td>xxxxxxx_xxxxxxxxx</td>
<td>(1)</td>
</tr>
<tr>
<td></td>
<td>0110111_00000000</td>
<td>xxxxxxx_xxxxxxxxx</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0110111_00000000</td>
<td>xxxxxxx_xxxxxxxxx</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0110111_00000000</td>
<td>xxxxxxx_xxxxxxxxx</td>
<td></td>
</tr>
<tr>
<td>2i. Poll for Page Write Complete</td>
<td>0110111_00000000</td>
<td>xxxxxxxo_xxxxxxxxx</td>
<td>(2)</td>
</tr>
<tr>
<td>3a. Enter Flash Read</td>
<td>0100011_00000010</td>
<td>xxxxxxx_xxxxxxxxx</td>
<td></td>
</tr>
<tr>
<td>3b. Load Address Extended High Byte</td>
<td>0001011_cccccccc</td>
<td>xxxxxxx_xxxxxxxxx</td>
<td>(10)</td>
</tr>
<tr>
<td>3c. Load Address High Byte</td>
<td>0000111_aaaaaaaaa</td>
<td>xxxxxxx_xxxxxxxxx</td>
<td></td>
</tr>
<tr>
<td>3d. Load Address Low Byte</td>
<td>0000011_bbbbbbbb</td>
<td>xxxxxxx_xxxxxxxxx</td>
<td></td>
</tr>
<tr>
<td>3e. Read Data Low and High Byte</td>
<td>0110010_00000000</td>
<td>xxxxxxx_xxxxxxxxx</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0110110_00000000</td>
<td>xxxxxxx-o00000000</td>
<td>Low byte</td>
</tr>
<tr>
<td></td>
<td>0110111_00000000</td>
<td>xxxxxxx-o00000000</td>
<td>High byte</td>
</tr>
<tr>
<td>4a. Enter EEPROM Write</td>
<td>0100011_00010001</td>
<td>xxxxxxx_xxxxxxxxx</td>
<td></td>
</tr>
<tr>
<td>4b. Load Address High Byte</td>
<td>0001011_aaaaaaaaa</td>
<td>xxxxxxx_xxxxxxxxx</td>
<td>(10)</td>
</tr>
<tr>
<td>4c. Load Address Low Byte</td>
<td>0000011_bbbbbbbb</td>
<td>xxxxxxx_xxxxxxxxx</td>
<td></td>
</tr>
<tr>
<td>4d. Load Data Byte</td>
<td>0010011_iiiiiiii</td>
<td>xxxxxxx_xxxxxxxxx</td>
<td></td>
</tr>
<tr>
<td>4e. Latch Data</td>
<td>0110111_00000000</td>
<td>xxxxxxx_xxxxxxxxx</td>
<td>(1)</td>
</tr>
<tr>
<td></td>
<td>1110111_00000000</td>
<td>xxxxxxx_xxxxxxxxx</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0110111_00000000</td>
<td>xxxxxxx_xxxxxxxxx</td>
<td></td>
</tr>
<tr>
<td>4f. Write EEPROM Page</td>
<td>0110011_00000000</td>
<td>xxxxxxx_xxxxxxxxx</td>
<td>(1)</td>
</tr>
<tr>
<td></td>
<td>0110001_00000000</td>
<td>xxxxxxx_xxxxxxxxx</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0110011_00000000</td>
<td>xxxxxxx_xxxxxxxxx</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0110011_00000000</td>
<td>xxxxxxx_xxxxxxxxx</td>
<td></td>
</tr>
</tbody>
</table>
Table 28-17. JTAG Programming Instruction (Continued)
Set (Continued) a = address high bits, b = address low bits, c = address extended bits, H = 0 - Low byte, 1 - High Byte,
o = data out, i = data in, x = don't care

<table>
<thead>
<tr>
<th>Instruction</th>
<th>TDI Sequence</th>
<th>TDO Sequence</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>4g. Poll for Page Write Complete</td>
<td>0100011_00000000</td>
<td>xxxxxxxo, xxxxxxxx</td>
<td>(2)</td>
</tr>
<tr>
<td>5a. Enter EEPROM Read</td>
<td>0100011_00000011</td>
<td>xxxxxxx_xxxxxxx</td>
<td></td>
</tr>
<tr>
<td>5b. Load Address High Byte</td>
<td>0000111_aaaaaaa</td>
<td>xxxxxxx_xxxxxxx</td>
<td>(10)</td>
</tr>
<tr>
<td>5c. Load Address Low Byte</td>
<td>0000011_____bbb</td>
<td>xxxxxxx_xxxxxxx</td>
<td></td>
</tr>
<tr>
<td>5d. Read Data Byte</td>
<td>0110011_bbbbbbb</td>
<td>xxxxxxx_xxxxxxx</td>
<td></td>
</tr>
<tr>
<td>6a. Enter Fuse Write</td>
<td>0100011_01000000</td>
<td>xxxxxxx_xxxxxxx</td>
<td></td>
</tr>
<tr>
<td>6b. Load Data Low Byte(a)</td>
<td>0010011_____iii</td>
<td>xxxxxxx_xxxxxxx</td>
<td>(3)</td>
</tr>
<tr>
<td>6c. Write Fuse Extended Byte</td>
<td>0111011_00000000</td>
<td>xxxxxxx_xxxxxxx</td>
<td>(1)</td>
</tr>
<tr>
<td></td>
<td>0111001_00000000</td>
<td>xxxxxxx_xxxxxxx</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0111011_00000000</td>
<td>xxxxxxx_xxxxxxx</td>
<td></td>
</tr>
<tr>
<td>6d. Poll for Fuse Write Complete</td>
<td>0110111_00000000</td>
<td>xxxxxxxo, xxxxxxxx</td>
<td>(2)</td>
</tr>
<tr>
<td>6e. Load Data Low Byte(b)</td>
<td>0010011_____iii</td>
<td>xxxxxxx_xxxxxxx</td>
<td>(3)</td>
</tr>
<tr>
<td>6f. Write Fuse High Byte</td>
<td>0110111_00000000</td>
<td>xxxxxxx_xxxxxxx</td>
<td>(1)</td>
</tr>
<tr>
<td></td>
<td>0110101_00000000</td>
<td>xxxxxxx_xxxxxxx</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0110111_00000000</td>
<td>xxxxxxx_xxxxxxx</td>
<td></td>
</tr>
<tr>
<td>6g. Poll for Fuse Write Complete</td>
<td>0110111_00000000</td>
<td>xxxxxxxo, xxxxxxxx</td>
<td>(2)</td>
</tr>
<tr>
<td>6h. Load Data Low Byte(b)</td>
<td>0010011_____iii</td>
<td>xxxxxxx_xxxxxxx</td>
<td>(3)</td>
</tr>
<tr>
<td>6i. Write Fuse Low Byte</td>
<td>0110011_00000000</td>
<td>xxxxxxx_xxxxxxx</td>
<td>(1)</td>
</tr>
<tr>
<td></td>
<td>0110001_00000000</td>
<td>xxxxxxx_xxxxxxx</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0110011_00000000</td>
<td>xxxxxxx_xxxxxxx</td>
<td></td>
</tr>
<tr>
<td>6j. Poll for Fuse Write Complete</td>
<td>0110011_00000000</td>
<td>xxxxxxxo, xxxxxxxx</td>
<td>(2)</td>
</tr>
<tr>
<td>7a. Enter Lock Bit Write</td>
<td>0100011_00100000</td>
<td>xxxxxxx_xxxxxxx</td>
<td></td>
</tr>
<tr>
<td>7b. Load Data Byte(c)</td>
<td>0010011_1111111</td>
<td>xxxxxxx_xxxxxxx</td>
<td>(4)</td>
</tr>
<tr>
<td>7c. Write Lock Bits</td>
<td>0110011_00000000</td>
<td>xxxxxxx_xxxxxxx</td>
<td>(1)</td>
</tr>
<tr>
<td></td>
<td>0110001_00000000</td>
<td>xxxxxxx_xxxxxxx</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0110011_00000000</td>
<td>xxxxxxx_xxxxxxx</td>
<td></td>
</tr>
<tr>
<td>7d. Poll for Lock Bit Write complete</td>
<td>0110011_00000000</td>
<td>xxxxxxxo, xxxxxxxx</td>
<td>(2)</td>
</tr>
<tr>
<td>8a. Enter Fuse/Lock Bit Read</td>
<td>0100011_00000100</td>
<td>xxxxxxx_xxxxxxx</td>
<td></td>
</tr>
<tr>
<td>8b. Read Extended Fuse Byte(c)</td>
<td>0111010_00000000</td>
<td>xxxxxxx_xxxxxxx</td>
<td>(6)</td>
</tr>
<tr>
<td></td>
<td>0111011_00000000</td>
<td>xxxxxxx_xxxxxxx</td>
<td></td>
</tr>
<tr>
<td>8c. Read Fuse High Byte(d)</td>
<td>0111110_00000000</td>
<td>xxxxxxx_xxxxxxx</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0111111_00000000</td>
<td>xxxxxxx_xxxxxxx</td>
<td>(7)</td>
</tr>
</tbody>
</table>
Table 28-17. JTAG Programming Instruction (Continued)

Set (Continued) \(a\) = address high bits, \(b\) = address low bits, \(c\) = address extended bits, \(H\) = 0 - Low byte, 1 - High Byte, \(o\) = data out, \(i\) = data in, \(x\) = don't care

<table>
<thead>
<tr>
<th>Instruction</th>
<th>TDI Sequence</th>
<th>TDO Sequence</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>8d. Read Fuse Low Byte(^{(8)})</td>
<td>0110010_00000000</td>
<td>xxxxxxx_xxxxxxx</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0110011_00000000</td>
<td>xxxxxxx_00000000</td>
<td></td>
</tr>
<tr>
<td>8e. Read Lock Bits(^{(9)})</td>
<td>0110110_00000000</td>
<td>xxxxxxx_xxxxxxx</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0110111_00000000</td>
<td>xxxxxxx_xxxxxxx</td>
<td></td>
</tr>
<tr>
<td>8f. Read Fuses and Lock Bits</td>
<td>0111010_00000000</td>
<td>xxxxxxx_xxxxxxx</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0111110_00000000</td>
<td>xxxxxxx_xxxxxxx</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0110010_00000000</td>
<td>xxxxxxx_xxxxxxx</td>
<td>(5) Fuse Ext. byte</td>
</tr>
<tr>
<td></td>
<td>0110110_00000000</td>
<td>xxxxxxx_xxxxxxx</td>
<td>Fuse High byte</td>
</tr>
<tr>
<td></td>
<td>0110111_00000000</td>
<td>xxxxxxx_xxxxxxx</td>
<td>Fuse Low byte</td>
</tr>
<tr>
<td></td>
<td>0110112_00000000</td>
<td>xxxxxxx_xxxxxxx</td>
<td>Lock bits</td>
</tr>
<tr>
<td>9a. Enter Signature Byte Read</td>
<td>0100011_00001000</td>
<td>xxxxxxx_xxxxxxx</td>
<td></td>
</tr>
<tr>
<td>9b. Load Address Byte</td>
<td>0000011_bbbbbbb</td>
<td>xxxxxxx_xxxxxxx</td>
<td></td>
</tr>
<tr>
<td>9c. Read Signature Byte</td>
<td>0110010_00000000</td>
<td>xxxxxxx_xxxxxxx</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0110011_00000000</td>
<td>xxxxxxx_xxxxxxx</td>
<td></td>
</tr>
<tr>
<td>10a. Enter Calibration Byte Read</td>
<td>0100011_00001000</td>
<td>xxxxxxx_xxxxxxx</td>
<td></td>
</tr>
<tr>
<td>10b. Load Address Byte</td>
<td>0000011_bbbbbbb</td>
<td>xxxxxxx_xxxxxxx</td>
<td></td>
</tr>
<tr>
<td>10c. Read Calibration Byte</td>
<td>0110110_00000000</td>
<td>xxxxxxx_xxxxxxx</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0110111_00000000</td>
<td>xxxxxxx_xxxxxxx</td>
<td></td>
</tr>
<tr>
<td>11a. Load No Operation Command</td>
<td>0100011_00000000</td>
<td>xxxxxxx_xxxxxxx</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0110011_00000000</td>
<td>xxxxxxx_xxxxxxx</td>
<td></td>
</tr>
</tbody>
</table>

Notes:
1. This command sequence is not required if the seven MSB are correctly set by the previous command sequence (which is normally the case).
2. Repeat until \(o\) = “1”.
3. Set bits to “0” to program the corresponding Fuse, “1” to unprogram the Fuse.
4. Set bits to “0” to program the corresponding Lock bit, “1” to leave the Lock bit unchanged.
5. “0” = programmed, “1” = unprogrammed.
6. The bit mapping for Fuses Extended byte is listed in Table 28-3 on page 347
7. The bit mapping for Fuses High byte is listed in Table 28-4 on page 348
8. The bit mapping for Fuses Low byte is listed in Table 28-5 on page 348
9. The bit mapping for Lock bytes bit is listed in Table 28-1 on page 346
10. Address bits exceeding PCMSB and EEAMSB (Table 28-11 and Table 28-12) are don’t care
11. All TDI and TDO sequences are represented by binary digits (0b...).
28.9.11 Flash Data Byte Register

The Flash Data Byte Register provides an efficient way to load the entire Flash page buffer before executing Page Write, or to read out/verify the content of the Flash. A state machine sets up the control signals to the Flash and senses the strobe signals from the Flash, thus only the data words need to be shifted in/out.

The Flash Data Byte Register actually consists of the 8-bit scan chain and a 8-bit temporary register. During page load, the Update-DR state copies the content of the scan chain over to the temporary register and initiates a write sequence that within 11 TCK cycles loads the content of the temporary register into the Flash page buffer. The AVR automatically alternates between writing the low and the high byte for each new Update-DR state, starting with the low byte for the first Update-DR encountered after entering the PROG_PAGELOAD command. The Program Counter is pre-incremented before writing the low byte, except for the first written byte. This ensures that the first data is written to the address set up by PROG_COMMANDS, and loading the last location in the page buffer does not make the Program Counter increment into the next page.

During Page Read, the content of the selected Flash byte is captured into the Flash Data Byte Register during the Capture-DR state. The AVR automatically alternates between reading the low and the high byte for each new Capture-DR state, starting with the low byte for the first Cap-
ture-DR encountered after entering the PROG_PAGEREAD command. The Program Counter is post-incremented after reading each high byte, including the first read byte. This ensures that the first data is captured from the first address set up by PROG_COMMANDS, and reading the last location in the page makes the program counter increment into the next page.

Figure 28-16. Flash Data Byte Register

The state machine controlling the Flash Data Byte Register is clocked by TCK. During normal operation in which eight bits are shifted for each Flash byte, the clock cycles needed to navigate through the TAP controller automatically feeds the state machine for the Flash Data Byte Register with sufficient number of clock pulses to complete its operation transparently for the user. However, if too few bits are shifted between each Update-DR state during page load, the TAP controller should stay in the Run-Test/Idle state for some TCK cycles to ensure that there are at least 11 TCK cycles between each Update-DR state.

28.9.12 Programming Algorithm

All references below of type “1a”, “1b”, and so on, refer to Table 28-17.

28.9.13 Entering Programming Mode

1. Enter JTAG instruction AVR_RESET and shift 1 in the Reset Register.
2. Enter instruction PROG_ENABLE and shift 0b1010_0011_0111_0000 in the Programming Enable Register.

28.9.14 Leaving Programming Mode

1. Enter JTAG instruction PROG_COMMANDS.
2. Disable all programming instructions by using no operation instruction 11a.
3. Enter instruction PROG_ENABLE and shift 0b0000_0000_0000_0000 in the programming Enable Register.
4. Enter JTAG instruction AVR_RESET and shift 0 in the Reset Register.
28.9.15 Performing Chip Erase

1. Enter JTAG instruction PROG_COMMANDS.
2. Start Chip Erase using programming instruction 1a.
3. Poll for Chip Erase complete using programming instruction 1b, or wait for \( t_{\text{WLRT_CE}} \) (refer to Table 28-13 on page 360).

28.9.16 Programming the Flash

Before programming the Flash a Chip Erase must be performed, see “Performing Chip Erase” on page 375.

1. Enter JTAG instruction PROG_COMMANDS.
2. Enable Flash write using programming instruction 2a.
3. Load address Extended High byte using programming instruction 2b.
4. Load address High byte using programming instruction 2c.
5. Load address Low byte using programming instruction 2d.
6. Load data using programming instructions 2e, 2f and 2g.
7. Repeat steps 5 and 6 for all instruction words in the page.
8. Write the page using programming instruction 2h.
9. Poll for Flash write complete using programming instruction 2i, or wait for \( t_{\text{WLRT}} \) (refer to Table 28-13 on page 360).
10. Repeat steps 3 to 9 until all data have been programmed.

A more efficient data transfer can be achieved using the PROG_PAGELOAD instruction:

1. Enter JTAG instruction PROG_COMMANDS.
2. Enable Flash write using programming instruction 2a.
3. Load the page address using programming instructions 2b, 2c and 2d. PCWORD (refer to Table 28-11 on page 351) is used to address within one page and must be written as 0.
4. Enter JTAG instruction PROG_PAGELOAD.
5. Load the entire page by shifting in all instruction words in the page byte-by-byte, starting with the LSB of the first instruction in the page and ending with the MSB of the last instruction in the page. Use Update-DR to copy the contents of the Flash Data Byte Register into the Flash page location and to auto-increment the Program Counter before each new word.
6. Enter JTAG instruction PROG_COMMANDS.
7. Write the page using programming instruction 2h.
8. Poll for Flash write complete using programming instruction 2i, or wait for \( t_{\text{WLRT}} \) (refer to Table 28-13 on page 360).
9. Repeat steps 3 to 8 until all data have been programmed.

28.9.17 Reading the Flash

1. Enter JTAG instruction PROG_COMMANDS.
2. Enable Flash read using programming instruction 3a.
3. Load address using programming instructions 3b, 3c and 3d.
4. Read data using programming instruction 3e.
5. Repeat steps 3 and 4 until all data have been read.

A more efficient data transfer can be achieved using the PROG_PAGEREAD instruction:
28.9.18 Programming the EEPROM

Before programming the EEPROM a Chip Erase must be performed, see “Performing Chip Erase” on page 375.

1. Enter JTAG instruction PROG_COMMANDS.
2. Enable EEPROM write using programming instruction 4a.
3. Load address High byte using programming instruction 4b.
4. Load address Low byte using programming instruction 4c.
5. Load data using programming instructions 4d and 4e.
6. Repeat steps 4 and 5 for all data bytes in the page.
7. Write the data using programming instruction 4f.
8. Poll for EEPROM write complete using programming instruction 4g, or wait for $t_{\text{WLRH}}$ (refer to Table 28-13 on page 360).
9. Repeat steps 3 to 8 until all data have been programmed.

Note that the PROG_PAGELOAD instruction can not be used when programming the EEPROM.

28.9.19 Reading the EEPROM

1. Enter JTAG instruction PROG_COMMANDS.
2. Enable EEPROM read using programming instruction 5a.
3. Load address using programming instructions 5b and 5c.
4. Read data using programming instruction 5d.
5. Repeat steps 3 and 4 until all data have been read.

Note that the PROG_PAGEREAD instruction can not be used when reading the EEPROM.

28.9.20 Programming the Fuses

1. Enter JTAG instruction PROG_COMMANDS.
2. Enable Fuse write using programming instruction 6a.
3. Load data high byte using programming instructions 6b. A bit value of “0” will program the corresponding fuse, a “1” will unprogram the fuse.
4. Write Fuse High byte using programming instruction 6c.
5. Poll for Fuse write complete using programming instruction 6d, or wait for $t_{\text{WLRH}}$ (refer to Table 28-13 on page 360).
6. Load data low byte using programming instructions 6e. A “0” will program the fuse, a “1” will unprogram the fuse.
7. Write Fuse low byte using programming instruction 6f.
8. Poll for Fuse write complete using programming instruction 6g, or wait for \( t_{\text{WLRH}} \) (refer to Table 28-13 on page 360).

28.9.21 Programming the Lock Bits
1. Enter JTAG instruction PROG_COMMANDS.
2. Enable Lock bit write using programming instruction 7a.
3. Load data using programming instructions 7b. A bit value of “0” will program the corresponding lock bit, a “1” will leave the lock bit unchanged.
4. Write Lock bits using programming instruction 7c.
5. Poll for Lock bit write complete using programming instruction 7d, or wait for \( t_{\text{WLRH}} \) (refer to Table 28-13 on page 360).

28.9.22 Reading the Fuses and Lock Bits
1. Enter JTAG instruction PROG_COMMANDS.
2. Enable Fuse/Lock bit read using programming instruction 8a.
3. To read all Fuses and Lock bits, use programming instruction 8e.
   To only read Fuse High byte, use programming instruction 8b.
   To only read Fuse Low byte, use programming instruction 8c.
   To only read Lock bits, use programming instruction 8d.

28.9.23 Reading the Signature Bytes
1. Enter JTAG instruction PROG_COMMANDS.
2. Enable Signature byte read using programming instruction 9a.
3. Load address 0x00 using programming instruction 9b.
4. Read first signature byte using programming instruction 9c.
5. Repeat steps 3 and 4 with address 0x01 and address 0x02 to read the second and third signature bytes, respectively.

28.9.24 Reading the Calibration Byte
1. Enter JTAG instruction PROG_COMMANDS.
2. Enable Calibration byte read using programming instruction 10a.
3. Load address 0x00 using programming instruction 10b.
4. Read the calibration byte using programming instruction 10c.
29. Electrical Characteristics

29.1 Absolute Maximum Ratings*

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Condition</th>
<th>Min.</th>
<th>Typ.</th>
<th>Max.</th>
<th>Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>$V_{IL}$</td>
<td>$V_{CC} = 2.7V - 5.5V$</td>
<td>-0.5</td>
<td></td>
<td>$0.2V_{CC}$</td>
<td>V</td>
</tr>
<tr>
<td>$V_{IL1}$</td>
<td>$V_{CC} = 2.7V - 5.5V$</td>
<td>-0.5</td>
<td></td>
<td>$0.1V_{CC}$</td>
<td>V</td>
</tr>
<tr>
<td>$V_{IL2}$</td>
<td>$V_{CC} = 2.7V - 5.5V$</td>
<td>-0.5</td>
<td></td>
<td>$0.1V_{CC}$</td>
<td>V</td>
</tr>
<tr>
<td>$V_{IH}$</td>
<td>$V_{CC} = 2.7V - 5.5V$</td>
<td>$0.2V_{CC} + 0.9V$</td>
<td></td>
<td>$V_{CC} + 0.5$</td>
<td>V</td>
</tr>
<tr>
<td>$V_{IH1}$</td>
<td>$V_{CC} = 2.7V - 5.5V$</td>
<td>$0.7V_{CC}$</td>
<td></td>
<td>$V_{CC} + 0.5$</td>
<td>V</td>
</tr>
<tr>
<td>$V_{IH2}$</td>
<td>$V_{CC} = 2.7V - 5.5V$</td>
<td>$0.9V_{CC}$</td>
<td></td>
<td>$V_{CC} + 0.5$</td>
<td>V</td>
</tr>
<tr>
<td>$V_{OL}$</td>
<td>$I_{OL} = 10mA$, $V_{CC} = 5V$</td>
<td></td>
<td>$0.7$</td>
<td></td>
<td>V</td>
</tr>
<tr>
<td>$V_{OH}$</td>
<td>$I_{OH} = -10mA$, $V_{CC} = 5V$</td>
<td></td>
<td>$2.3$</td>
<td></td>
<td>V</td>
</tr>
<tr>
<td>$I_{IL}$</td>
<td>$V_{CC} = 5.5V$, pin low (absolute value)</td>
<td></td>
<td></td>
<td>$1$</td>
<td>$\mu$A</td>
</tr>
<tr>
<td>$I_{IH}$</td>
<td>$V_{CC} = 5.5V$, pin high (absolute value)</td>
<td></td>
<td></td>
<td>$1$</td>
<td>$\mu$A</td>
</tr>
<tr>
<td>$R_{RST}$</td>
<td></td>
<td></td>
<td></td>
<td>30</td>
<td>$k\Omega$</td>
</tr>
<tr>
<td>$R_{PU}$</td>
<td></td>
<td></td>
<td></td>
<td>20</td>
<td>$k\Omega$</td>
</tr>
</tbody>
</table>

*NOTICE: Stresses beyond those listed under “Absolute Maximum Ratings” may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or other conditions beyond those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.

29.2 DC Characteristics

$T_A = -40^\circ C$ to $85^\circ C$, $V_{CC} = 2.7V$ to $5.5V$ (unless otherwise noted)
$T_A = -40^\circ C$ to $85^\circ C$, $V_{CC} = 2.7V$ to $5.5V$ (unless otherwise noted)  (Continued)

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter</th>
<th>Condition</th>
<th>Min.</th>
<th>Typ.</th>
<th>Max.</th>
<th>Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>$I_{CC}$</td>
<td>Power Supply Current$^6$</td>
<td>Active 4MHz, $V_{CC} = 3V$ (ATmega16U4/ATmega32U4)</td>
<td></td>
<td>5</td>
<td>mA</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Active 8MHz, $V_{CC} = 5V$ (ATmega16U4/ATmega32U4)</td>
<td></td>
<td>10</td>
<td>15</td>
<td>mA</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Idle 4MHz, $V_{CC} = 3V$ (ATmega16U4/ATmega32U4)</td>
<td></td>
<td>2</td>
<td>mA</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Idle 8MHz, $V_{CC} = 5V$ (ATmega16U4/ATmega32U4)</td>
<td></td>
<td>6</td>
<td>mA</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Power-down mode</td>
<td>WDT enabled, $V_{CC} = 3V$, Regulator Disabled</td>
<td></td>
<td>&lt;10</td>
<td>12</td>
<td>$\mu A$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>WDT disabled, $V_{CC} = 3V$, Regulator Disabled</td>
<td></td>
<td>1</td>
<td>5</td>
<td>$\mu A$</td>
</tr>
<tr>
<td>$V_{ACIO}$</td>
<td>Analog Comparator Input Offset Voltage</td>
<td>$V_{CC} = 5V$</td>
<td></td>
<td>&lt;10</td>
<td>40</td>
<td>mV</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$V_I = V_{CC}/2$</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$I_{ACLK}$</td>
<td>Analog Comparator Input Leakage Current</td>
<td>$V_{CC} = 5V$</td>
<td></td>
<td>-50</td>
<td>50</td>
<td>nA</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$V_I = V_{CC}/2$</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$I_{ACID}$</td>
<td>Analog Comparator Propagation Delay</td>
<td>$V_{CC} = 2.7V$</td>
<td></td>
<td>750</td>
<td></td>
<td>ns</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$V_{CC} = 4.0V$</td>
<td></td>
<td>500</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$R_{USB}$</td>
<td>USB Series resistor (external)</td>
<td></td>
<td></td>
<td>22±5%</td>
<td></td>
<td>$\Omega$</td>
</tr>
<tr>
<td>$V_{reg}$</td>
<td>Regulator Output Voltage</td>
<td>$C_{UCA} = 1\mu F\pm20%$, $U_{VCC} \geq 4.0V$, $I_{L} \leq 80mA$</td>
<td></td>
<td>3.0</td>
<td>3.3</td>
<td>3.6</td>
</tr>
<tr>
<td></td>
<td></td>
<td>, or $U_{VCC} \geq 3.4V$, $I_{L} \leq 55mA$</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Note:  
1. "Max" means the highest value where the pin is guaranteed to be read as low  
2. "Min" means the lowest value where the pin is guaranteed to be read as high  
3. Although each I/O port can sink more than the test conditions (20mA at $V_{CC} = 5V$, 10mA at $V_{CC} = 3V$) under steady state conditions (non-transparent), the following must be observed: ATmega16U4/ATmega32U4: 
   1.) The sum of all IOL, for ports A0-A7, G2, C4-C7 should not exceed 100 mA. 
   2.) The sum of all IOL, for ports C0-C3, G0-G1, D0-D7 should not exceed 100 mA. 
   3.) The sum of all IOL, for ports G3-G5, B0-B7, E0-E7 should not exceed 100 mA. 
   4.) The sum of all IOL, for ports F0-F7 should not exceed 100 mA. 
If IOL exceeds the test condition, VOL may exceed the related specification. Pins are not guaranteed to sink current greater than the listed test condition. 

4. Although each I/O port can source more than the test conditions (20mA at $V_{CC} = 5V$, 10mA at $V_{CC} = 3V$) under steady state conditions (non-transparent), the following must be observed: ATmega16U4/ATmega32U4: 
   1) The sum of all IOH, for ports A0-A7, G2, C4-C7 should not exceed 100 mA. 
   2) The sum of all IOH, for ports C0-C3, G0-G1, D0-D7 should not exceed 100 mA. 
   3) The sum of all IOH, for ports G3-G5, B0-B7, E0-E7 should not exceed 100 mA. 
   4) The sum of all IOH, for ports F0-F7 should not exceed 100 mA. 
5. All DC Characteristics contained in this datasheet are based on simulation and characterization of other AVR microcontrollers manufactured in the same process technology. These values are preliminary values representing design targets, and will be updated after characterization of actual silicon 
6. Values with "Power Reduction Register 1 - PRR1" disabled (0x00).
7. Maximum regulator output current should be reduced by the USB buffer current required when USB is active (about 25mA). The remaining regulator output current can be used for the external application.

8. As specified on the USB Electrical chapter, the D+/D- pads can withstand voltages down to -1V applied through a 39 Ohms resistor

29.3 External Clock Drive Waveforms

**Figure 29-1.** External Clock Drive Waveforms

29.4 External Clock Drive

**Table 29-1.** External Clock Drive

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter</th>
<th>$V_{CC}=1.8-5.5V$</th>
<th>$V_{CC}=2.7-5.5V$</th>
<th>$V_{CC}=4.5-5.5V$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Min.</td>
<td>Max.</td>
<td>Min.</td>
</tr>
<tr>
<td>$1/\tau_{CLCL}$</td>
<td>Oscillator Frequency</td>
<td>0</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>$\tau_{CLCL}$</td>
<td>Clock Period</td>
<td>500</td>
<td>125</td>
<td>62.5</td>
</tr>
<tr>
<td>$t_{CHCX}$</td>
<td>High Time</td>
<td>200</td>
<td>50</td>
<td>25</td>
</tr>
<tr>
<td>$t_{CLCX}$</td>
<td>Low Time</td>
<td>200</td>
<td>50</td>
<td>25</td>
</tr>
<tr>
<td>$t_{CLCH}$</td>
<td>Rise Time</td>
<td>2.0</td>
<td>1.6</td>
<td>0.5</td>
</tr>
<tr>
<td>$t_{CHCL}$</td>
<td>Fall Time</td>
<td>2.0</td>
<td>1.6</td>
<td>0.5</td>
</tr>
<tr>
<td>$\Delta t_{CLCL}$</td>
<td>Change in period from one clock cycle to the next</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
</tbody>
</table>

Note: All DC Characteristics contained in this datasheet are based on simulation and characterization of other AVR microcontrollers manufactured in the same process technology. These values are preliminary values representing design targets, and will be updated after characterization of actual silicon.

29.5 Maximum speed vs. $V_{CC}$

Maximum frequency is depending on $V_{CC}$. As shown in Figure 29-2, the Maximum Frequency vs. $V_{CC}$ curve is linear between $2.7V < V_{CC} < 5.5V$. 

---

380
29.6 2-wire Serial Interface Characteristics

Table 29-2 describes the requirements for devices connected to the 2-wire Serial Bus. The ATmega16U4/ATmega32U4 2-wire Serial Interface meets or exceeds these requirements under the noted conditions.

Timing symbols refer to Figure 29-3.

Table 29-2. 2-wire Serial Bus Requirements

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter</th>
<th>Condition</th>
<th>Min</th>
<th>Max</th>
<th>Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>VIL</td>
<td>Input Low-voltage</td>
<td>-0.5</td>
<td>0.3 V_{CC}</td>
<td>V</td>
<td></td>
</tr>
<tr>
<td>VIH</td>
<td>Input High-voltage</td>
<td>0.7 V_{CC}</td>
<td>V_{CC} + 0.5</td>
<td>V</td>
<td></td>
</tr>
<tr>
<td>V_{phys}^{(1)}</td>
<td>Hysteresis of Schmitt Trigger Inputs</td>
<td>0.05 V_{CC}^{(2)}</td>
<td>–</td>
<td>V</td>
<td></td>
</tr>
<tr>
<td>VOL^{(1)}</td>
<td>Output Low-voltage</td>
<td>3 mA sink current</td>
<td>0</td>
<td>0.4</td>
<td>V</td>
</tr>
<tr>
<td>t_{R}^{(1)}</td>
<td>Rise Time for both SDA and SCL</td>
<td>20 + 0.1 C_{b}^{(3)(2)}</td>
<td>300</td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>t_{of}^{(1)}</td>
<td>Output Fall Time from V_{ILmin} to V_{ILmax}</td>
<td>10 pF &lt; C_{b} &lt; 400 pF^{(3)}</td>
<td>20 + 0.1 C_{b}^{(3)(2)}</td>
<td>250</td>
<td>ns</td>
</tr>
<tr>
<td>t_{isp}^{(1)}</td>
<td>Spikes Suppressed by Input Filter</td>
<td>0</td>
<td>50^{(2)}</td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>I_{i}</td>
<td>Input Current each I/O Pin</td>
<td>0.1V_{CC} &lt; I_{i} &lt; 0.9V_{CC}</td>
<td>-10</td>
<td>10</td>
<td>μA</td>
</tr>
<tr>
<td>C_{i}^{(1)}</td>
<td>Capacitance for each I/O Pin</td>
<td>–</td>
<td>10</td>
<td>pF</td>
<td></td>
</tr>
<tr>
<td>f_{SCL}</td>
<td>SCL Clock Frequency</td>
<td>f_{CK}^{(4)} &gt; max(16f_{SCL}, 250kHz)^{(5)}</td>
<td>0</td>
<td>400</td>
<td>kHz</td>
</tr>
<tr>
<td>Rp</td>
<td>Value of Pull-up resistor</td>
<td>f_{SCL} ≤ 100 kHz</td>
<td>V_{CC} - 0.4V/3mA</td>
<td>1000ns/C_{b}</td>
<td>Ω</td>
</tr>
<tr>
<td></td>
<td></td>
<td>f_{SCL} &gt; 100 kHz</td>
<td>V_{CC} - 0.4V/3mA</td>
<td>300ns/C_{b}</td>
<td>Ω</td>
</tr>
<tr>
<td>t_{HD,STA}</td>
<td>Hold Time (repeated) START Condition</td>
<td>f_{SCL} ≤ 100 kHz</td>
<td>4.0</td>
<td>–</td>
<td>μs</td>
</tr>
<tr>
<td></td>
<td></td>
<td>f_{SCL} &gt; 100 kHz</td>
<td>0.6</td>
<td>–</td>
<td>μs</td>
</tr>
<tr>
<td>t_{LOW}</td>
<td>Low Period of the SCL Clock</td>
<td>f_{SCL} ≤ 100 kHz^{(6)}</td>
<td>4.7</td>
<td>–</td>
<td>μs</td>
</tr>
<tr>
<td></td>
<td></td>
<td>f_{SCL} &gt; 100 kHz^{(7)}</td>
<td>1.3</td>
<td>–</td>
<td>μs</td>
</tr>
</tbody>
</table>
Table 29-2. 2-wire Serial Bus Requirements (Continued)

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter</th>
<th>Condition</th>
<th>Min</th>
<th>Max</th>
<th>Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>t(_{\text{HIGH}})</td>
<td>High period of the SCL clock</td>
<td>f(_{\text{SCL}}) (\leq) 100 kHz</td>
<td>4.0</td>
<td>–</td>
<td>(\mu)s</td>
</tr>
<tr>
<td></td>
<td></td>
<td>f(_{\text{SCL}}) &gt; 100 kHz</td>
<td>0.6</td>
<td>–</td>
<td>(\mu)s</td>
</tr>
<tr>
<td>t(_{\text{SU,STA}})</td>
<td>Set-up time for a repeated START condition</td>
<td>f(_{\text{SCL}}) (\leq) 100 kHz</td>
<td>4.7</td>
<td>–</td>
<td>(\mu)s</td>
</tr>
<tr>
<td></td>
<td></td>
<td>f(_{\text{SCL}}) &gt; 100 kHz</td>
<td>0.6</td>
<td>–</td>
<td>(\mu)s</td>
</tr>
<tr>
<td>t(_{\text{HD,DAT}})</td>
<td>Data hold time</td>
<td>f(_{\text{SCL}}) (\leq) 100 kHz</td>
<td>0</td>
<td>3.45</td>
<td>(\mu)s</td>
</tr>
<tr>
<td></td>
<td></td>
<td>f(_{\text{SCL}}) &gt; 100 kHz</td>
<td>0</td>
<td>0.9</td>
<td>(\mu)s</td>
</tr>
<tr>
<td>t(_{\text{SU,DAT}})</td>
<td>Data setup time</td>
<td>f(_{\text{SCL}}) (\leq) 100 kHz</td>
<td>250</td>
<td>–</td>
<td>ns</td>
</tr>
<tr>
<td></td>
<td></td>
<td>f(_{\text{SCL}}) &gt; 100 kHz</td>
<td>100</td>
<td>–</td>
<td>ns</td>
</tr>
<tr>
<td>t(_{\text{SU,STO}})</td>
<td>Setup time for STOP condition</td>
<td>f(_{\text{SCL}}) (\leq) 100 kHz</td>
<td>4.0</td>
<td>–</td>
<td>(\mu)s</td>
</tr>
<tr>
<td></td>
<td></td>
<td>f(_{\text{SCL}}) &gt; 100 kHz</td>
<td>0.6</td>
<td>–</td>
<td>(\mu)s</td>
</tr>
<tr>
<td>t(_{\text{BUF}})</td>
<td>Bus free time between a STOP and START condition</td>
<td>f(_{\text{SCL}}) (\leq) 100 kHz</td>
<td>4.7</td>
<td>–</td>
<td>(\mu)s</td>
</tr>
<tr>
<td></td>
<td></td>
<td>f(_{\text{SCL}}) &gt; 100 kHz</td>
<td>1.3</td>
<td>–</td>
<td>(\mu)s</td>
</tr>
</tbody>
</table>

Notes:
1. In ATmega16U4/ATmega32U4, this parameter is characterized and not 100% tested.
2. Required only for f\(_{\text{SCL}}\) > 100 kHz.
3. C\(_{\text{D}}\) = capacitance of one bus line in pF.
4. f\(_{\text{CK}}\) = CPU clock frequency
5. This requirement applies to all ATmega16U4/ATmega32U4 2-wire Serial Interface operation. Other devices connected to the 2-wire Serial Bus need only obey the general f\(_{\text{SCL}}\) requirement.
6. The actual low period generated by the ATmega16U4/ATmega32U4 2-wire Serial Interface is (1/f\(_{\text{SCL}}\) - 2/f\(_{\text{CK}}\)), thus f\(_{\text{CK}}\) must be greater than 6 MHz for the low time requirement to be strictly met at f\(_{\text{SCL}}\) = 100 kHz.
7. The actual low period generated by the ATmega16U4/ATmega32U4 2-wire Serial Interface is (1/f\(_{\text{SCL}}\) - 2/f\(_{\text{CK}}\)), thus the low time requirement will not be strictly met for f\(_{\text{SCL}}\) > 308 kHz when f\(_{\text{CK}}\) = 8 MHz. Still, ATmega16U4/ATmega32U4 devices connected to the bus may communicate at full speed (400 kHz) with other ATmega16U4/ATmega32U4 devices, as well as any other device with a proper t\(_{\text{LOW}}\) acceptance margin.

Figure 29-3. 2-wire Serial Bus Timing

![2-wire Serial Bus Timing Diagram](image-url)
29.7 SPI Timing Characteristics

See Figure 29-4 and Figure 29-5 for details.

Table 29-3. SPI Timing Parameters

<table>
<thead>
<tr>
<th>Description</th>
<th>Mode</th>
<th>Min</th>
<th>Typ</th>
<th>Max</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>SCK period</td>
<td>Master</td>
<td>See Table 17-4</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>SCK high/low</td>
<td>Master</td>
<td>50% duty cycle</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Rise/Fall time</td>
<td>Master</td>
<td>TBD</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Setup</td>
<td>Master</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>Hold</td>
<td>Master</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>Out to SCK</td>
<td>Master</td>
<td>0.5 • t\text{ck}</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>SCK to out</td>
<td>Master</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>SCK to out high</td>
<td>Master</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>SSS low to out</td>
<td>Slave</td>
<td>15</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>SCK period</td>
<td>Slave</td>
<td>4 • t\text{ck}</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>SCK high/low(^{(1)})</td>
<td>Slave</td>
<td>2 • t\text{ck}</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>Rise/Fall time</td>
<td>Slave</td>
<td>TBD</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>Setup</td>
<td>Slave</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>Hold</td>
<td>Slave</td>
<td>t\text{ck}</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>SCK to out</td>
<td>Slave</td>
<td>15</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>SCK to SSS high</td>
<td>Slave</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>SSS high to tri-state</td>
<td>Slave</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>SSS low to SCK</td>
<td>Slave</td>
<td>20</td>
<td></td>
</tr>
</tbody>
</table>

Note: 1. In SPI Programming mode the minimum SCK high/low period is:
   - 2 t\text{CLK} for f\text{CK} < 12 MHz
   - 3 t\text{CLK} for f\text{CK} > 12 MHz

Figure 29-4. SPI Interface Timing Requirements (Master Mode)
29.8 Hardware Boot Entrance Timing Characteristics

Figure 29-6. Hardware Boot Timing Requirements

Table 29-4. Hardware Boot Timings

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter</th>
<th>Min</th>
<th>Max</th>
</tr>
</thead>
<tbody>
<tr>
<td>tSHRH</td>
<td>HWB low Setup before Reset High</td>
<td>0</td>
<td>StartUpTime(SUT) + Time Out Delay(TOUT)</td>
</tr>
<tr>
<td>tHHRH</td>
<td>HWB low Hold after Reset High</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 29-5. ADC Characteristics

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter</th>
<th>Condition</th>
<th>Min</th>
<th>Typ</th>
<th>Max</th>
<th>Units</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Resolution</td>
<td>Single Ended Conversion</td>
<td>10</td>
<td></td>
<td></td>
<td>Bits</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Differential conversion, gain = 1x/10x/40x</td>
<td>8</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Differential conversion, gain = 200x</td>
<td>8</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>TUE</td>
<td>Absolute accuracy</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>$V_{REF} = 4V, \ V_{CC} = 4V, \ ADC \ clock = 200 \ kHz$</td>
<td>2.0</td>
<td>3.0</td>
<td></td>
<td>LSB</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Gain = 1x/10x/40x, $V_{REF} = 4V, \ V_{CC} = 5V, \ ADC \ clock = 200 \ kHz$</td>
<td>2.0</td>
<td>3.0</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Gain = 200x, $V_{REF} = 4V, \ V_{CC} = 5V, \ ADC \ clock = 200 \ kHz$</td>
<td>2.0</td>
<td></td>
<td>4.0</td>
<td></td>
</tr>
<tr>
<td>Symbol</td>
<td>Parameter</td>
<td>Condition</td>
<td>Min</td>
<td>Typ</td>
<td>Max</td>
<td>Units</td>
</tr>
<tr>
<td>--------</td>
<td>----------------------------</td>
<td>---------------------------------------------------------------------------</td>
<td>------</td>
<td>------</td>
<td>------</td>
<td>-------</td>
</tr>
<tr>
<td>INL</td>
<td>Integral Non-Linearity</td>
<td>$V_{\text{REF}} = 4V, \ V_{\text{CC}} = 4V, \ \text{ADC clock} = 200 \ kHz$</td>
<td>0.5</td>
<td>1.5</td>
<td></td>
<td>LSB</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Gain = 1x/10x/40x, $V_{\text{REF}} = 4V, \ V_{\text{CC}} = 5V, \ \text{ADC clock} = 200 \ kHz$</td>
<td>0.3</td>
<td>1.5</td>
<td></td>
<td>LSB</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Gain = 200x, $V_{\text{REF}} = 4V, \ V_{\text{CC}} = 5V, \ \text{ADC clock} = 200 \ kHz$</td>
<td>0.5</td>
<td>1.5</td>
<td></td>
<td>LSB</td>
</tr>
<tr>
<td>DNL</td>
<td>Differential Non-Linearity</td>
<td>$V_{\text{REF}} = 4V, \ V_{\text{CC}} = 4V, \ \text{ADC clock} = 200 \ kHz$</td>
<td>0.4</td>
<td>0.7</td>
<td></td>
<td>LSB</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Gain = 1x/10x/40x, $V_{\text{REF}} = 4V, \ V_{\text{CC}} = 5V, \ \text{ADC clock} = 200 \ kHz$</td>
<td>0.3</td>
<td>1.0</td>
<td></td>
<td>LSB</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Gain = 200x, $V_{\text{REF}} = 4V, \ V_{\text{CC}} = 5V, \ \text{ADC clock} = 200 \ kHz$</td>
<td>0.6</td>
<td>1.0</td>
<td></td>
<td>LSB</td>
</tr>
<tr>
<td>Gain Error</td>
<td></td>
<td>$V_{\text{REF}} = 4V, \ V_{\text{CC}} = 4V, \ \text{ADC clock} = 200 \ kHz$</td>
<td>-2.5</td>
<td>-1.0</td>
<td>2.5</td>
<td>LSB</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Gain = 1x/10x/40x, $V_{\text{REF}} = 4V, \ V_{\text{CC}} = 5V, \ \text{ADC clock} = 200 \ kHz$</td>
<td>0.0</td>
<td>-1.5</td>
<td>-2.5</td>
<td>LSB</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Gain = 200x, $V_{\text{REF}} = 4V, \ V_{\text{CC}} = 5V, \ \text{ADC clock} = 200 \ kHz$</td>
<td>0.0</td>
<td>-1.8</td>
<td>-3.0</td>
<td>LSB</td>
</tr>
<tr>
<td>Offset Error</td>
<td></td>
<td>$V_{\text{REF}} = 4V, \ V_{\text{CC}} = 4V, \ \text{ADC clock} = 200 \ kHz$</td>
<td>-2.5</td>
<td>1.5</td>
<td>2.5</td>
<td>LSB</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$V_{\text{REF}} = 4V, \ V_{\text{CC}} = 5V, \ \text{ADC clock} = 200 \ kHz, \text{Differential mode}$</td>
<td>-2.0</td>
<td>0.0</td>
<td>2.0</td>
<td>LSB</td>
</tr>
<tr>
<td>$V_{\text{REF}}$</td>
<td>Reference Voltage</td>
<td>Single Ended Conversion</td>
<td>2.56</td>
<td></td>
<td></td>
<td>AVCC</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Differential Conversion</td>
<td>2.56</td>
<td></td>
<td></td>
<td>AVCC - 0.5</td>
</tr>
<tr>
<td>AVCC</td>
<td>Analog Supply Voltage</td>
<td>$V_{\text{CC}} = 0.3$</td>
<td></td>
<td></td>
<td></td>
<td>V</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$V_{\text{CC}} + 0.3$</td>
<td></td>
<td></td>
<td></td>
<td>V</td>
</tr>
<tr>
<td>$V_{\text{IN}}$</td>
<td>Input Voltage</td>
<td>Single ended channels</td>
<td>GND</td>
<td></td>
<td></td>
<td>V</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Differential Conversion</td>
<td>0</td>
<td></td>
<td></td>
<td>AVCC</td>
</tr>
<tr>
<td>Input Bandwidth</td>
<td></td>
<td>Single Ended Channels</td>
<td>38.5</td>
<td></td>
<td></td>
<td>kHz</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Differential Channels</td>
<td>4</td>
<td></td>
<td></td>
<td>kHz</td>
</tr>
<tr>
<td>$V_{\text{INT}}$</td>
<td>Internal Voltage Reference</td>
<td>2.56V</td>
<td>2.4</td>
<td>2.56</td>
<td>2.8</td>
<td>V</td>
</tr>
<tr>
<td>$R_{\text{REF}}$</td>
<td>Reference Input Resistance</td>
<td></td>
<td>32</td>
<td></td>
<td></td>
<td>kΩ</td>
</tr>
<tr>
<td>$R_{\text{AIN}}$</td>
<td>Analog Input Resistance</td>
<td></td>
<td>100</td>
<td></td>
<td></td>
<td>MΩ</td>
</tr>
</tbody>
</table>
30. Typical Characteristics

The following charts show typical behavior. These figures are not tested during manufacturing. All current consumption measurements are performed with all I/O pins configured as inputs and with internal pull-ups enabled. A sine wave generator with rail-to-rail output is used as clock source.

All Active- and Idle current consumption measurements are done with all bits in the PRR registers set and thus, the corresponding I/O modules are turned off. Also the Analog Comparator is disabled during these measurements. See “Power Reduction Register” on page 45 for details.

The power consumption in Power-down mode is independent of clock selection.

The current consumption is a function of several factors such as: operating voltage, operating frequency, loading of I/O pins, switching rate of I/O pins, code executed and ambient temperature. The dominating factors are operating voltage and frequency.

The current drawn from capacitive loaded pins may be estimated (for one pin) as $C_L \times V_{cc} \times f$
where $C_L$ = load capacitance, $V_{cc}$ = operating voltage and $f$ = average switching frequency of I/O pin.

The parts are characterized at frequencies higher than test limits. Parts are not guaranteed to function properly at frequencies higher than the ordering code indicates.

The difference between current consumption in Power-down mode with Watchdog Timer enabled and Power-down mode with Watchdog Timer disabled represents the differential current drawn by the Watchdog Timer.

30.1 Active Supply Current

Figure 30-1. Active Supply Current vs. Low Frequency (1MHz) and T= 25°C
Figure 30-2.  Active Supply Current vs. Low Frequency (1MHz) and T= 85°C

Figure 30-3.  Active Supply Current vs. Frequency (1-16 MHz) and T= -40°C
Figure 30-4. Active Supply Current vs. Frequency (1-16 MHz) and T = 25°C

Figure 30-5. Active Supply Current vs. Frequency (1-16 MHz) and T = 85°C
30.2 Idle Supply Current

Figure 30-6. Idle Supply Current vs. Low Frequency (1 MHz) and T = 25°C

Figure 30-7. Idle Supply Current vs. Low Frequency (1 MHz) and T = 85°C
Figure 30-8. Idle Supply Current vs. Frequency (1-16 MHz) T = 25°C

Figure 30-9. Idle Supply Current vs. Frequency (1-16 MHz) T = 85°C
30.3 Power-down Supply Current

**Figure 30-10.** Power-Down Supply Current vs. \( V_{CC} \) (WDT Disabled)

**Figure 30-11.** Power-Down Supply Current vs. \( V_{CC} \) (WDT Enabled)
30.4 Power-save Supply Current

Figure 30-13. Power-Save Supply Current vs. $V_{CC}$ (WDT Disabled)
30.5 Pin Pull-Up

**Figure 30-14.** I/O Pin Pull-up Resistor Current vs. Input Voltage ($V_{CC} = 2.7$ V)

**Figure 30-15.** I/O Pin Pull-up Resistor Current vs. Input Voltage ($V_{CC} = 5$ V)
30.6 Pin Driver Strength

Figure 30-16. Reset Pull-up Resistor Current vs. Reset Pin Voltage (V_{CC}=5V)

Figure 30-17. I/O Pin Output Voltage vs. Sink Current (V_{CC} = 3 V)
Figure 30-18. I/O Pin Output Voltage vs. Sink Current (V\textsubscript{CC} = 5 V)

![Graph showing I/O pin output voltage vs. sink current for different temperatures (85 °C, 25 °C, -40 °C).]

Figure 30-19. I/O Pin Output Voltage vs. Source Current (V\textsubscript{CC} = 3 V)

![Graph showing I/O pin output voltage vs. source current for different temperatures (85 °C, 25 °C, -40 °C).]
Figure 30-20. I/O Pin Output Voltage vs. Source Current ($V_{CC} = 5$ V)

Figure 30-21. USB DP LO Pull-Up Resistor Current vs. USB Pin Voltage
30.7 Pin Threshold and Hysteresis

**Figure 30-22.** I/O Pin Input Threshold Voltage vs. $V_{CC}$ ($V_{IH}$ IO Pin read as ‘1’)

**Figure 30-23.** I/O Pin Input Threshold Voltage vs. $V_{CC}$ ($V_{IL}$ IO Pin read as ‘0’)

Figure 30-24. USB Pin Input Threshold Voltage vs. $V_{CC}$ ($V_{HH}$, IO Pin read as ‘1’)

Figure 30-25. USB Pin Input Threshold Voltage vs. $V_{CC}$ ($V_{IL}$, I/O Pin read as ‘0’)

**Figure 30-26.** Vbus Pin Input Threshold Voltage vs. $V_{CC}$ ($V_{th}$, IO Pin read as ‘1’)

**Figure 30-27.** Vbus Pin Input Threshold Voltage vs. $V_{CC}$ ($V_{il}$, I/O Pin read as ‘0’)}
30.8 BOD Threshold

**Figure 30-28.** BOD Thresholds vs. Temperature (BODLEVEL is 2.6 V)

**Figure 30-29.** BOD Thresholds vs. Temperature (BODLEVEL is 3.5 V)
Figure 30-30. BOD Thresholds vs. Temperature (BODLEVEL is 4.3 V)

Figure 30-31. Bandgap Voltage vs. Vcc
30.9 Internal Oscillator Speed

Figure 30-32. Bandgap Voltage vs. Temperature

Figure 30-33. Watchdog Oscillator Frequency vs. Temperature
Figure 30-34. Watchdog Oscillator Frequency vs. $V_{CC}$

![Graph showing Watchdog Oscillator Frequency vs. $V_{CC}$ with curves for different temperatures (-40 °C, 25 °C, 85 °C).]

Figure 30-35. Calibrated 8 MHz RC Oscillator Frequency vs. Oscal Value

![Graph showing Calibrated 8 MHz RC Oscillator Frequency vs. Oscal Value with curves for different temperatures (-40 °C, 25 °C, 85 °C).]
Figure 30-36. Calibrated 8 MHz RC Oscillator Frequency vs. Temperature

Figure 30-37. Calibrated 8 MHz RC Oscillator Frequency vs. Operating Voltage
30.10 Current Consumption of Peripheral Units

Figure 30-38. OSCCAL VALUE STEP SIZE IN % (Base frequency = 0.0 MHz)

Figure 30-39. USB Regulator Level vs. $V_{CC}$
**Figure 30-40.** USB Regulator Level with load 75 Ω vs. VCC

**Figure 30-41.** ADC Internal Vref vs. Vcc
30.11 Current Consumption in Reset and Reset Pulsewidth

Figure 30-42. Internal Reference Voltage vs. Sink Current

Figure 30-43. Reset Supply Current vs. Frequency (1 - 20 MHz)
# 31. Register Summary

<table>
<thead>
<tr>
<th>Address</th>
<th>Name</th>
<th>Bit 7</th>
<th>Bit 6</th>
<th>Bit 5</th>
<th>Bit 4</th>
<th>Bit 3</th>
<th>Bit 2</th>
<th>Bit 1</th>
<th>Bit 0</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>(0xFF)</td>
<td>Reserved</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>(0xFE)</td>
<td>Reserved</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>(0xFD)</td>
<td>Reserved</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>(0xFC)</td>
<td>Reserved</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>(0xFB)</td>
<td>Reserved</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>(0xFA)</td>
<td>Reserved</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>(0xF9)</td>
<td>Reserved</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>(0xF8)</td>
<td>Reserved</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>(0xF7)</td>
<td>Reserved</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>(0xF6)</td>
<td>Reserved</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>(0xF5)</td>
<td>Reserved</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>(0xF4)</td>
<td>UENT</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>EPINT6:0</td>
</tr>
<tr>
<td>(0xF3)</td>
<td>UESCHX</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>BYCT10:8</td>
</tr>
<tr>
<td>(0xF2)</td>
<td>UESCLX</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>BYCT7:0</td>
</tr>
<tr>
<td>(0xF1)</td>
<td>UEDATX</td>
<td>DAT7:0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(0xF0)</td>
<td>UEFIEN</td>
<td>FLMRX</td>
<td>NAKINE</td>
<td></td>
<td>NAKOUTE</td>
<td>RXSTPE</td>
<td>RXOUTE</td>
<td>STALDE</td>
<td>TXINE</td>
<td></td>
</tr>
<tr>
<td>(0xEF)</td>
<td>UESTA1X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>CURRBRK1:0</td>
</tr>
<tr>
<td>(0xE5)</td>
<td>UESTA0X</td>
<td>CFGOK</td>
<td>OVER1</td>
<td>UNDER1</td>
<td></td>
<td>DTST0:1:0</td>
<td></td>
<td></td>
<td></td>
<td>NBUSYBK1:0</td>
</tr>
<tr>
<td>(0xED)</td>
<td>UECFG1X</td>
<td>EPSZ2:0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>ALLOC</td>
</tr>
<tr>
<td>(0xEC)</td>
<td>UECFG0X</td>
<td>EPTYPE1:0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(0xEB)</td>
<td>UECONX</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>(0xEA)</td>
<td>UERST</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>EPSTM:0</td>
</tr>
<tr>
<td>(0xE9)</td>
<td>UENUM</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>EPNMS2:0</td>
</tr>
<tr>
<td>(0xE8)</td>
<td>UEINTX</td>
<td>FIFCON</td>
<td>NAKIN</td>
<td>RWAL</td>
<td>NAKOUT1</td>
<td>RXSTPI</td>
<td>RXOUT1</td>
<td>STALDE</td>
<td>TXINI</td>
<td></td>
</tr>
<tr>
<td>(0xE7)</td>
<td>Reserved</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>(0xE6)</td>
<td>UDMFN</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>FNCERR</td>
</tr>
<tr>
<td>(0xE5)</td>
<td>UDNUM</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>FNUM10:8</td>
</tr>
<tr>
<td>(0xE4)</td>
<td>UDNUML</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>FNUM7:0</td>
</tr>
<tr>
<td>(0xE3)</td>
<td>UDADDR</td>
<td>ADDREN</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>UADD6:0</td>
</tr>
<tr>
<td>(0xE2)</td>
<td>UDREN</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>(0xE1)</td>
<td>UDINT</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>(0xE0)</td>
<td>UDCON</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>RSTCPU</td>
</tr>
<tr>
<td>(0xDF)</td>
<td>Reserved</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>(0xD8)</td>
<td>USBTST</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>VBSTI</td>
</tr>
<tr>
<td>(0xD7)</td>
<td>UHWCN</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>UVEGE</td>
</tr>
<tr>
<td>(0xD6)</td>
<td>Reserved</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>(0xD5)</td>
<td>Reserved</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>(0xD4)</td>
<td>DT4</td>
<td>DT4H3</td>
<td>DT4H2</td>
<td>DT4H1</td>
<td>DT4H0</td>
<td>DT4L3</td>
<td>DT4L2</td>
<td>DT4L1</td>
<td>DT4L0</td>
<td></td>
</tr>
<tr>
<td>(0xD3)</td>
<td>Reserved</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>(0xD2)</td>
<td>OCR4D</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>Timer/Counte</td>
</tr>
<tr>
<td>(0xD1)</td>
<td>OCR4C</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>Timer/Counte</td>
</tr>
<tr>
<td>(0xD0)</td>
<td>OCR4B</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>Timer/Counte</td>
</tr>
<tr>
<td>(0xAF)</td>
<td>OCR4A</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>Timer/Counte</td>
</tr>
<tr>
<td>(0xAE)</td>
<td>OCR4</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>Timer/Counte</td>
</tr>
<tr>
<td>(0xAD)</td>
<td>OCR4</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>Timer/Counte</td>
</tr>
<tr>
<td>(0xAC)</td>
<td>OCR4</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>Timer/Counte</td>
</tr>
<tr>
<td>(0xAB)</td>
<td>OCR4</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>Timer/Counte</td>
</tr>
<tr>
<td>(0xA9)</td>
<td>OCR4</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>Timer/Counte</td>
</tr>
<tr>
<td>(0xA8)</td>
<td>OCR4</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>Timer/Counte</td>
</tr>
<tr>
<td>(0xA7)</td>
<td>OCR4</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>Timer/Counte</td>
</tr>
<tr>
<td>(0xA6)</td>
<td>OCR4</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>Timer/Counte</td>
</tr>
<tr>
<td>(0xA5)</td>
<td>OCR4</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>Timer/Counte</td>
</tr>
<tr>
<td>(0xA4)</td>
<td>OCR4</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>Timer/Counte</td>
</tr>
<tr>
<td>(0xA3)</td>
<td>OCR4</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>Timer/Counte</td>
</tr>
<tr>
<td>(0xA2)</td>
<td>OCR4</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>Timer/Counte</td>
</tr>
<tr>
<td>(0xA1)</td>
<td>OCR4</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>Timer/Counte</td>
</tr>
<tr>
<td>(0xA0)</td>
<td>OCR4</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>Timer/Counte</td>
</tr>
<tr>
<td>(0x9F)</td>
<td>OCR4</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>Timer/Counte</td>
</tr>
<tr>
<td>(0x9E)</td>
<td>OCR4</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>Timer/Counte</td>
</tr>
<tr>
<td>(0x9D)</td>
<td>OCR4</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>Timer/Counte</td>
</tr>
<tr>
<td>(0x9C)</td>
<td>OCR4</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>Timer/Counte</td>
</tr>
<tr>
<td>(0x9B)</td>
<td>OCR4</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>Timer/Counte</td>
</tr>
<tr>
<td>(0x9A)</td>
<td>OCR4</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>Timer/Counte</td>
</tr>
<tr>
<td>(0x99)</td>
<td>OCR4</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>Timer/Counte</td>
</tr>
<tr>
<td>(0x98)</td>
<td>OCR4</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>Timer/Counte</td>
</tr>
</tbody>
</table>

*NOTE: All registers are 8-bit wide.*
<table>
<thead>
<tr>
<th>Address</th>
<th>Name</th>
<th>Bit 7</th>
<th>Bit 6</th>
<th>Bit 5</th>
<th>Bit 4</th>
<th>Bit 3</th>
<th>Bit 2</th>
<th>Bit 1</th>
<th>Bit 0</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>(0x8E)</td>
<td>TCNT4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(0x8D)</td>
<td>TWAMR</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(0x8C)</td>
<td>TWCR</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(0x8B)</td>
<td>TWDR</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(0x8A)</td>
<td>TWAR</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(0x89)</td>
<td>TWSP</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(0x88)</td>
<td>TWBR</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(0x87)</td>
<td>Reserved</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(0x86)</td>
<td>Reserved</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(0x85)</td>
<td>Reserved</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(0x84)</td>
<td>Reserved</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(0x83)</td>
<td>Reserved</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(0x82)</td>
<td>Reserved</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(0x81)</td>
<td>Reserved</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(0x80)</td>
<td>Reserved</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(0xFF)</td>
<td>Reserved</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(0xEF)</td>
<td>Reserved</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(0xEF)</td>
<td>Reserved</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(0x9D)</td>
<td>OCR2CH</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(0x9C)</td>
<td>OCR2CL</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(0x9B)</td>
<td>OCR2BH</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(0x9A)</td>
<td>OCR2BL</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(0x99)</td>
<td>OCR2AH</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(0x98)</td>
<td>OCR2AL</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(0x97)</td>
<td>ICBR</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(0x96)</td>
<td>ICRL</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(0x95)</td>
<td>TONT3H</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(0x94)</td>
<td>TCNT3L</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(0x93)</td>
<td>Reserved</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(0x92)</td>
<td>TCR3IC</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(0x91)</td>
<td>TCR3IB</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(0x90)</td>
<td>TCR3IA</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(0x8F)</td>
<td>Reserved</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(0x8E)</td>
<td>Reserved</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(0x8D)</td>
<td>OCR1CH</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(0x8C)</td>
<td>OCR1CL</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(0x8B)</td>
<td>OCR1BH</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(0x8A)</td>
<td>OCR1BL</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(0x89)</td>
<td>OCR1AH</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(0x88)</td>
<td>OCR1AL</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(0x87)</td>
<td>ICR1H</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(0x86)</td>
<td>ICR1L</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(0x85)</td>
<td>TONT1H</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(0x84)</td>
<td>TCNT1L</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(0x83)</td>
<td>Reserved</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(0x82)</td>
<td>TCR1IC</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(0x81)</td>
<td>TCR1IB</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(0x80)</td>
<td>TCR1IA</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(0x7F)</td>
<td>DIDR1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(0x7E)</td>
<td>DIDR0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(0x7D)</td>
<td>DIDR2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Address</td>
<td>Name</td>
<td>Bit 7</td>
<td>Bit 6</td>
<td>Bit 5</td>
<td>Bit 4</td>
<td>Bit 3</td>
<td>Bit 2</td>
<td>Bit 1</td>
<td>Bit 0</td>
<td>Page</td>
</tr>
<tr>
<td>-----------</td>
<td>---------</td>
<td>-------</td>
<td>-------</td>
<td>-------</td>
<td>-------</td>
<td>-------</td>
<td>-------</td>
<td>-------</td>
<td>-------</td>
<td>------</td>
</tr>
<tr>
<td>0x1B (0x3B)</td>
<td>PCIFR</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>PCIF0</td>
</tr>
<tr>
<td>0x1A (0x3A)</td>
<td>Reserved</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>0x19 (0x39)</td>
<td>TIFR4</td>
<td>OCF4D</td>
<td>OCF4A</td>
<td>OCF4B</td>
<td>-</td>
<td>-</td>
<td>TOV4</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>0x18 (0x38)</td>
<td>TIFR3</td>
<td>-</td>
<td>-</td>
<td>ICF3</td>
<td>OCF3C</td>
<td>OCF3B</td>
<td>OCF3A</td>
<td>TOV3</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>0x17 (0x37)</td>
<td>Reserved</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>0x16 (0x36)</td>
<td>TIFR1</td>
<td>-</td>
<td>-</td>
<td>ICF1</td>
<td>OCF1C</td>
<td>OCF1B</td>
<td>OCF1A</td>
<td>TOV1</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>0x15 (0x35)</td>
<td>TIFR0</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>OCF0B</td>
<td>OCF0A</td>
<td>TOV0</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>0x14 (0x34)</td>
<td>Reserved</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>0x13 (0x33)</td>
<td>Reserved</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>0x12 (0x32)</td>
<td>Reserved</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>0x11 (0x31)</td>
<td>PORTF</td>
<td>PORTF</td>
<td>PORTF</td>
<td>PORTF</td>
<td>PORTF</td>
<td>PORTF</td>
<td>PORTF</td>
<td>PORTF</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>0x10 (0x30)</td>
<td>DDRF</td>
<td>DDF7</td>
<td>DDF6</td>
<td>DDF5</td>
<td>DDF4</td>
<td>-</td>
<td>-</td>
<td>DDF1</td>
<td>DDF0</td>
<td>-</td>
</tr>
<tr>
<td>0x0F (0x2F)</td>
<td>PINF</td>
<td>PINF</td>
<td>PINF</td>
<td>PINF</td>
<td>PINF</td>
<td>-</td>
<td>-</td>
<td>PINF</td>
<td>PINF</td>
<td>-</td>
</tr>
<tr>
<td>0x0E (0x2E)</td>
<td>PORTE</td>
<td>PORTE</td>
<td>PORTE</td>
<td>PORTE</td>
<td>PORTE</td>
<td>-</td>
<td>-</td>
<td>PORTE</td>
<td>PORTE</td>
<td>-</td>
</tr>
<tr>
<td>0x0D (0x2D)</td>
<td>DDRE</td>
<td>DDRE</td>
<td>DDRE</td>
<td>DDRE</td>
<td>DDRE</td>
<td>-</td>
<td>-</td>
<td>DDRE</td>
<td>DDRE</td>
<td>-</td>
</tr>
<tr>
<td>0x0C (0x2C)</td>
<td>PINE</td>
<td>PINE</td>
<td>PINE</td>
<td>PINE</td>
<td>PINE</td>
<td>-</td>
<td>-</td>
<td>PINE</td>
<td>PINE</td>
<td>-</td>
</tr>
<tr>
<td>0x0B (0x2B)</td>
<td>PORTD</td>
<td>PORTD</td>
<td>PORTD</td>
<td>PORTD</td>
<td>PORTD</td>
<td>PORTD</td>
<td>PORTD</td>
<td>PORTD</td>
<td>PORTD</td>
<td>PORTD</td>
</tr>
<tr>
<td>0x0A (0x2A)</td>
<td>DDRD</td>
<td>DDRD</td>
<td>DDRD</td>
<td>DDRD</td>
<td>DDRD</td>
<td>DDRD</td>
<td>DDRD</td>
<td>DDRD</td>
<td>DDRD</td>
<td>DDRD</td>
</tr>
<tr>
<td>0x09 (0x29)</td>
<td>PIND</td>
<td>PIND</td>
<td>PIND</td>
<td>PIND</td>
<td>PIND</td>
<td>PIND</td>
<td>PIND</td>
<td>PIND</td>
<td>PIND</td>
<td>PIND</td>
</tr>
<tr>
<td>0x08 (0x28)</td>
<td>PORTC</td>
<td>PORTC</td>
<td>PORTC</td>
<td>PORTC</td>
<td>PORTC</td>
<td>PORTC</td>
<td>PORTC</td>
<td>PORTC</td>
<td>PORTC</td>
<td>PORTC</td>
</tr>
<tr>
<td>0x07 (0x27)</td>
<td>DDRC</td>
<td>DDRC</td>
<td>DDRC</td>
<td>DDRC</td>
<td>DDRC</td>
<td>DDRC</td>
<td>DDRC</td>
<td>DDRC</td>
<td>DDRC</td>
<td>DDRC</td>
</tr>
<tr>
<td>0x06 (0x26)</td>
<td>PINC</td>
<td>PINC</td>
<td>PINC</td>
<td>PINC</td>
<td>PINC</td>
<td>PINC</td>
<td>PINC</td>
<td>PINC</td>
<td>PINC</td>
<td>PINC</td>
</tr>
<tr>
<td>0x05 (0x25)</td>
<td>PORTB</td>
<td>PORTB</td>
<td>PORTB</td>
<td>PORTB</td>
<td>PORTB</td>
<td>PORTB</td>
<td>PORTB</td>
<td>PORTB</td>
<td>PORTB</td>
<td>PORTB</td>
</tr>
<tr>
<td>0x04 (0x24)</td>
<td>DDRB</td>
<td>DDRB</td>
<td>DDRB</td>
<td>DDRB</td>
<td>DDRB</td>
<td>DDRB</td>
<td>DDRB</td>
<td>DDRB</td>
<td>DDRB</td>
<td>DDRB</td>
</tr>
<tr>
<td>0x03 (0x23)</td>
<td>PINB</td>
<td>PINB</td>
<td>PINB</td>
<td>PINB</td>
<td>PINB</td>
<td>PINB</td>
<td>PINB</td>
<td>PINB</td>
<td>PINB</td>
<td>PINB</td>
</tr>
<tr>
<td>0x02 (0x22)</td>
<td>Reserved</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>0x01 (0x21)</td>
<td>Reserved</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>0x00 (0x20)</td>
<td>Reserved</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

Note:
1. For compatibility with future devices, reserved bits should be written to zero if accessed. Reserved I/O memory addresses should never be written.
2. I/O registers within the address range $00 - $1F are directly bit-accessible using the SBI and CBI instructions. In these registers, the value of single bits can be checked by using the SBIS and SBIC instructions.
3. Some of the status flags are cleared by writing a logical one to them. Note that the CBI and SBI instructions will operate on all bits in the I/O register, writing a one back into any flag read as set, thus clearing the flag. The CBI and SBI instructions work with registers 0x00 to 0x1F only.
4. When using the I/O specific commands IN and OUT, the I/O addresses $00 - $3F must be used. When addressing I/O registers as data space using LD and ST instructions, $20 must be added to these addresses. The ATmega16U4/ATmega32U4 is a complex microcontroller with more peripheral units than can be supported within the 64 location reserved in Opcode for the IN and OUT instructions. For the Extended I/O space from $60 - $1FF in SRAM, only the ST/STS/STD and LD/LDS/LDD instructions can be used.
## 32. Instruction Set Summary

<table>
<thead>
<tr>
<th>Mnemonics</th>
<th>Operands</th>
<th>Description</th>
<th>Operation</th>
<th>Flags</th>
<th>#Clocks</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADD</td>
<td>Rd, Rr</td>
<td>Add two Registers</td>
<td>Rd ← Rd + Rr</td>
<td>Z,N,V,H</td>
<td>1</td>
</tr>
<tr>
<td>ADC</td>
<td>Rd, Rr</td>
<td>Add with Carry two Registers</td>
<td>Rd ← Rd + C</td>
<td>Z,N,V,H</td>
<td>1</td>
</tr>
<tr>
<td>ADIW</td>
<td>Rd, K</td>
<td>Add Immediate to Word</td>
<td>Rdh:Rdl ← Rd + K</td>
<td>Z,N,V,S</td>
<td>2</td>
</tr>
<tr>
<td>SUB</td>
<td>Rd, Rr</td>
<td>Subtract two Registers</td>
<td>Rd ← Rd - Rr</td>
<td>Z,N,V,H</td>
<td>1</td>
</tr>
<tr>
<td>SUBI</td>
<td>Rd, K</td>
<td>Subtract Constant from Register</td>
<td>Rd ← Rd - K</td>
<td>Z,N,V,H</td>
<td>1</td>
</tr>
<tr>
<td>SBC</td>
<td>Rd, Rr</td>
<td>Subtract with Carry two Registers</td>
<td>Rd ← Rd - Rr - C</td>
<td>Z,N,V,H</td>
<td>1</td>
</tr>
<tr>
<td>SBCI</td>
<td>Rd, K</td>
<td>Subtract with Carry Constant from Register</td>
<td>Rd ← Rd - K - C</td>
<td>Z,N,V,H</td>
<td>1</td>
</tr>
<tr>
<td>SBW</td>
<td>Rd, K</td>
<td>Subtract Immediate from Word</td>
<td>Rdh:Rdl ← Rdh:Rdl - K</td>
<td>Z,N,V,S</td>
<td>2</td>
</tr>
<tr>
<td>AND</td>
<td>Rd, Rr</td>
<td>Logical AND Registers</td>
<td>Rd ← Rd • Rr</td>
<td>Z,N,V</td>
<td>1</td>
</tr>
<tr>
<td>ANDI</td>
<td>Rd, K</td>
<td>Logical AND Register and Constant</td>
<td>Rd ← Rd • K</td>
<td>Z,N,V</td>
<td>1</td>
</tr>
<tr>
<td>OR</td>
<td>Rd, Rr</td>
<td>Logical OR Registers</td>
<td>Rd ← Rd v Rr</td>
<td>Z,N,V</td>
<td>1</td>
</tr>
<tr>
<td>ORI</td>
<td>Rd, K</td>
<td>Logical OR Register and Constant</td>
<td>Rd ← Rd v K</td>
<td>Z,N,V</td>
<td>1</td>
</tr>
<tr>
<td>EOR</td>
<td>Rd, Rr</td>
<td>Exclusive OR Registers</td>
<td>Rd ← Rd ⊕ Rr</td>
<td>Z,N,V</td>
<td>1</td>
</tr>
<tr>
<td>COM</td>
<td>Rd</td>
<td>One's Complement</td>
<td>Rd ← 0xFF - Rd</td>
<td>Z,N,V</td>
<td>1</td>
</tr>
<tr>
<td>NEG</td>
<td>Rd</td>
<td>Two's Complement</td>
<td>Rd ← 0x00 - Rd</td>
<td>Z,N,V,H</td>
<td>1</td>
</tr>
<tr>
<td>SBR</td>
<td>Rd, k</td>
<td>Set Bit(s) in Register</td>
<td>Rd ← Rd’ k</td>
<td>Z,N,V</td>
<td>1</td>
</tr>
<tr>
<td>CBR</td>
<td>Rd, K</td>
<td>Clear Bit(s) in Register</td>
<td>Rd ← Rd ⊕ (0xFF - K)</td>
<td>Z,N,V</td>
<td>1</td>
</tr>
<tr>
<td>INC</td>
<td>Rd</td>
<td>Increment</td>
<td>Rd ← Rd + 1</td>
<td>Z,N,V</td>
<td>1</td>
</tr>
<tr>
<td>DEC</td>
<td>Rd</td>
<td>Decrement</td>
<td>Rd ← Rd - 1</td>
<td>Z,N,V</td>
<td>1</td>
</tr>
<tr>
<td>TST</td>
<td>Rd</td>
<td>Test for Zero or Minus</td>
<td>Rd ← Rd = Rd</td>
<td>Z,N,V</td>
<td>1</td>
</tr>
<tr>
<td>CLR</td>
<td>Rd</td>
<td>Clear Register</td>
<td>Rd ← Rd</td>
<td>Z,N,V</td>
<td>1</td>
</tr>
<tr>
<td>SER</td>
<td>Rd</td>
<td>Set Register</td>
<td>Rd ← 0xFF</td>
<td>None</td>
<td>1</td>
</tr>
<tr>
<td>MUL</td>
<td>Rd, Rr</td>
<td>Multiply Unsigned</td>
<td>R1.R0 ← Rd • Rr</td>
<td>Z,C</td>
<td>2</td>
</tr>
<tr>
<td>MULS</td>
<td>Rd, Rr</td>
<td>Multiply Signed</td>
<td>R1.R0 ← Rd • Rr</td>
<td>Z,C</td>
<td>2</td>
</tr>
<tr>
<td>FMUL</td>
<td>Rd, Rr</td>
<td>Fractional Multiply Unsigned</td>
<td>R1.R0 ← (Rd • Rr) • Rr</td>
<td>Z,C</td>
<td>2</td>
</tr>
<tr>
<td>FMULS</td>
<td>Rd, Rr</td>
<td>Fractional Multiply Signed</td>
<td>R1.R0 ← (Rd • Rr) • Rr</td>
<td>Z,C</td>
<td>2</td>
</tr>
</tbody>
</table>

### BRANCH INSTRUCTIONS

<table>
<thead>
<tr>
<th>Mnemonics</th>
<th>Operands</th>
<th>Description</th>
<th>Operation</th>
<th>Flags</th>
<th>#Clocks</th>
</tr>
</thead>
<tbody>
<tr>
<td>JMP</td>
<td>k</td>
<td>Relative Jump</td>
<td>PC ← PC + k + 1</td>
<td>None</td>
<td>2</td>
</tr>
<tr>
<td>LJMP</td>
<td>k</td>
<td>Indirect Jump to (Z)</td>
<td>PC ← (Z)</td>
<td>None</td>
<td>2</td>
</tr>
<tr>
<td>ELJMP</td>
<td>k</td>
<td>Extended Indirect Jump to (Z)</td>
<td>PC ← (EINZ)</td>
<td>None</td>
<td>2</td>
</tr>
<tr>
<td>JMP</td>
<td>k</td>
<td>Direct Jump</td>
<td>PC ← k</td>
<td>None</td>
<td>3</td>
</tr>
<tr>
<td>RCALL</td>
<td>k</td>
<td>Relative Subroutine Call</td>
<td>PC ← PC + k + 1</td>
<td>None</td>
<td>4</td>
</tr>
<tr>
<td>ICALL</td>
<td>k</td>
<td>Indirect Call to (Z)</td>
<td>PC ← Z</td>
<td>None</td>
<td>4</td>
</tr>
<tr>
<td>EI CALL</td>
<td>k</td>
<td>Extended Indirect Call to (Z)</td>
<td>PC ← (EINZ)</td>
<td>None</td>
<td>4</td>
</tr>
<tr>
<td>CALL</td>
<td>k</td>
<td>Direct Subroutine Call</td>
<td>PC ← k</td>
<td>None</td>
<td>5</td>
</tr>
<tr>
<td>RET</td>
<td></td>
<td>Subroutine Return</td>
<td>PC ← STACK</td>
<td>None</td>
<td>5</td>
</tr>
<tr>
<td>RETI</td>
<td></td>
<td>Interrupt Return</td>
<td>PC ← STACK</td>
<td>None</td>
<td>5</td>
</tr>
</tbody>
</table>

### Compare

<table>
<thead>
<tr>
<th>Mnemonics</th>
<th>Operands</th>
<th>Description</th>
<th>Operation</th>
<th>Flags</th>
<th>#Clocks</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPSE</td>
<td>Rd, Rr</td>
<td>Compare, Skip if Equal</td>
<td>if (Rd = Rr) PC ← PC + 2 or 3</td>
<td>None</td>
<td>1/2/3</td>
</tr>
<tr>
<td>CP</td>
<td>Rd, Rr</td>
<td>Compare</td>
<td>Rd ← Rr</td>
<td>Z,N,V,C,H</td>
<td>1</td>
</tr>
<tr>
<td>CPC</td>
<td>Rd, Rr</td>
<td>Compare with Carry</td>
<td>Rd ← Rr • C</td>
<td>Z,N,V,C,H</td>
<td>1</td>
</tr>
<tr>
<td>CPI</td>
<td>Rd, K</td>
<td>Compare Register with Immediate</td>
<td>Rd ← Rd’</td>
<td>Z,N,V,C,H</td>
<td>1</td>
</tr>
<tr>
<td>SBRB</td>
<td>R, b</td>
<td>Skip if Bit in Register Cleared</td>
<td>if (Rb = 0) PC ← PC + 2 or 3</td>
<td>None</td>
<td>1/2/3</td>
</tr>
<tr>
<td>SBRS</td>
<td>R, b</td>
<td>Skip if Bit in Register is Set</td>
<td>if (Rb = 1) PC ← PC + 2 or 3</td>
<td>None</td>
<td>1/2/3</td>
</tr>
<tr>
<td>SBIC</td>
<td>P, b</td>
<td>Skip if Bit in I/O Register Cleared</td>
<td>if (Pb = 0) PC ← PC + 2 or 3</td>
<td>None</td>
<td>1/2/3</td>
</tr>
<tr>
<td>SBIS</td>
<td>P, b</td>
<td>Skip if Bit in I/O Register is Set</td>
<td>if (Pb = 1) PC ← PC + 2 or 3</td>
<td>None</td>
<td>1/2/3</td>
</tr>
<tr>
<td>BRBS</td>
<td>s, k</td>
<td>Branch if Status Flag Set</td>
<td>if (SREG(s) = 1) PC ← PC + k + 1</td>
<td>None</td>
<td>1/2</td>
</tr>
<tr>
<td>BRBC</td>
<td>s, k</td>
<td>Branch if Status Flag Cleared</td>
<td>if (SREG(s) = 0) PC ← PC + k + 1</td>
<td>None</td>
<td>1/2</td>
</tr>
<tr>
<td>BREQ</td>
<td>k</td>
<td>Branch if Equal</td>
<td>if (Z = 1) PC ← PC + k + 1</td>
<td>None</td>
<td>1/2</td>
</tr>
<tr>
<td>BRNE</td>
<td>k</td>
<td>Branch if Not Equal</td>
<td>if (Z = 0) PC ← PC + k + 1</td>
<td>None</td>
<td>1/2</td>
</tr>
<tr>
<td>BRES</td>
<td>k</td>
<td>Branch if Carry Cleared</td>
<td>if (C = 1) PC ← PC + k + 1</td>
<td>None</td>
<td>1/2</td>
</tr>
<tr>
<td>BRZC</td>
<td>k</td>
<td>Branch if Carry Cleared</td>
<td>if (C = 0) PC ← PC + k + 1</td>
<td>None</td>
<td>1/2</td>
</tr>
<tr>
<td>BRSH</td>
<td>k</td>
<td>Branch if Same or Higher</td>
<td>if (C = 0) PC ← PC + k + 1</td>
<td>None</td>
<td>1/2</td>
</tr>
<tr>
<td>BRLO</td>
<td>k</td>
<td>Branch if Lower</td>
<td>if (C = 1) PC ← PC + k + 1</td>
<td>None</td>
<td>1/2</td>
</tr>
<tr>
<td>BRMI</td>
<td>k</td>
<td>Branch if Minus</td>
<td>if (N = 1) PC ← PC + k + 1</td>
<td>None</td>
<td>1/2</td>
</tr>
<tr>
<td>BRPL</td>
<td>k</td>
<td>Branch if Plus</td>
<td>if (N = 0) PC ← PC + k + 1</td>
<td>None</td>
<td>1/2</td>
</tr>
<tr>
<td>BRGE</td>
<td>k</td>
<td>Branch if Greater or Equal, Signed</td>
<td>if (N = V = 0) PC ← PC + k + 1</td>
<td>None</td>
<td>1/2</td>
</tr>
<tr>
<td>BHLT</td>
<td>k</td>
<td>Branch if Less Than Zero, Signed</td>
<td>if (N = V = 1) PC ← PC + k + 1</td>
<td>None</td>
<td>1/2</td>
</tr>
<tr>
<td>BRHS</td>
<td>k</td>
<td>Branch if Half Carry Flag Set</td>
<td>if (H = 1) PC ← PC + k + 1</td>
<td>None</td>
<td>1/2</td>
</tr>
<tr>
<td>BRHCO</td>
<td>k</td>
<td>Branch if Half Carry Flag Cleared</td>
<td>if (H = 0) PC ← PC + k + 1</td>
<td>None</td>
<td>1/2</td>
</tr>
<tr>
<td>BRTS</td>
<td>k</td>
<td>Branch if T Flag Set</td>
<td>if (T = 1) PC ← PC + k</td>
<td>None</td>
<td>1/2</td>
</tr>
<tr>
<td>BRTC</td>
<td>k</td>
<td>Branch if T Flag Cleared</td>
<td>if (T = 0) PC ← PC + k</td>
<td>None</td>
<td>1/2</td>
</tr>
<tr>
<td>BRVS</td>
<td>k</td>
<td>Branch if Overflow Flag is Set</td>
<td>if (V = 1) PC ← PC + k + 1</td>
<td>None</td>
<td>1/2</td>
</tr>
<tr>
<td>Mnemonics</td>
<td>Operands</td>
<td>Description</td>
<td>Operation</td>
<td>Flags</td>
<td>#Clocks</td>
</tr>
<tr>
<td>-----------</td>
<td>----------</td>
<td>-------------</td>
<td>-----------</td>
<td>-------</td>
<td>--------</td>
</tr>
<tr>
<td>BRVC</td>
<td>k</td>
<td>Branch if Overflow Flag is Cleared</td>
<td>if ( V = 0 ) then ( PC ), ( PC + k + 1 )</td>
<td>None</td>
<td>1/2</td>
</tr>
<tr>
<td>BRIE</td>
<td>k</td>
<td>Branch if Interrupt Enabled</td>
<td>if ( I = 1 ) then ( PC ), ( PC + k + 1 )</td>
<td>None</td>
<td>1/2</td>
</tr>
<tr>
<td>BRID</td>
<td>k</td>
<td>Branch if Interrupt Disabled</td>
<td>if ( I = 0 ) then ( PC ), ( PC + k + 1 )</td>
<td>None</td>
<td>1/2</td>
</tr>
</tbody>
</table>

| BIT AND BIT-TEST INSTRUCTIONS | | | | | |
|-------------------------------|------------------|--------------|------|------|
| SBI                            | P + b             | Set Bit in I/O Register | iOP(I,b) + 1 | None | 2     |
| CB1                            | P + b             | Clear Bit in I/O Register | iOP(I,b) - 0 | None | 2     |
| LSL                            | Rd                | Logical Shift Left | Rd(n+1) . . . Rd(0) . . . 0 | Z,C,N,V | 1     |
| LSR                            | Rd                | Logical Shift Right | Rd(n) . . . Rd(n-1) . . . Rd(7) . . . 0 | Z,C,N,V | 1     |
| ROL                            | Rd                | Rotate Left Through Carry | Rd(0), C, Rd(n-1), Rd(n), . . . Rd(7) | Z,C,N,V | 1     |
| ROR                            | Rd                | Rotate Right Through Carry | Rd(7), C, Rd(n-1), Rd(n), Rd(0) | Z,C,N,V | 1     |
| ASR                            | Rd                | Arithmetic Shift Right | Rd(n) - Rd(n-1), n=0 . . . 6 | Z,C,N,V | 1     |
| SWAP                           | Rd                | Swap Nibbles | Rd(3.0), Rd(7.4), Rd(7.4), Rd(3.0) | None | 1     |
| BSET                           | s                 | Flag Set | SREG(s) . . . 1 | SREG(s) | 1     |
| BCLR                           | s                 | Flag Clear | SREG(s) . . . 0 | SREG(s) | 1     |
| BST                            | Rd, b             | Bit Store from Register to T | T . . . Rd(b) | T | 1     |
| BLD                            | Rd, b             | Bit Load from T to Register | Rd(b) . . . T | None | 1     |
| SEC                            |                    | Set Carry | C = 1 | C | 1     |
| CLC                            |                    | Clear Carry | C = 0 | C | 1     |
| SEN                            |                    | Set Negative Flag | N = 1 | N | 1     |
| CLN                            |                    | Clear Negative Flag | N = 0 | N | 1     |
| SEZ                            |                    | Set Zero Flag | Z = 1 | Z | 1     |
| CLZ                            |                    | Clear Zero Flag | Z = 0 | Z | 1     |
| SFI                            |                    | Global Interrupt Enable | I = 1 | I | 1     |
| CLI                            |                    | Global Interrupt Disable | I = 0 | I | 1     |
| SES                            |                    | Set Signed Test Flag | S = 1 | S | 1     |
| CLS                            |                    | Clear Signed Test Flag | S = 0 | S | 1     |
| SEV                            |                    | Set Two's Complement Overflow | V = 1 | V | 1     |
| CLV                            |                    | Clear Two's Complement Overflow | V = 0 | V | 1     |
| SET                            |                    | Set T in SREG | T = 1 | T | 1     |
| CLT                            |                    | Clear T in SREG | T = 0 | T | 1     |
| SEH                            |                    | Set Half Carry Flag in SREG | H = 1 | H | 1     |
| CLH                            |                    | Clear Half Carry Flag in SREG | H = 0 | H | 1     |

<p>| DATA TRANSFER INSTRUCTIONS | | | | | |
|----------------------------|------------------|--------------|------|------|
| MOV                            | Rd, Rs           | Move Between Registers | Rd . . . Rs | None | 1     |
| MOVW                           | Rd, Rs           | Copy Register Word | Rd+1.Rd . . . R+1.Rs | None | 1     |
| LDI                            | Rd, K            | Load Immediate | Rd . . . K | None | 1     |
| LD                             | Rd, X            | Load Indirect | Rd . . . (X) | None | 2     |
| LD                             | Rd, X +       | Load Indirect and Post-Inc. | Rd . . . (X). X . . . X + 1 | None | 2     |
| LD                             | Rd. . . X       | Load Indirect and Pre-Dec. | X . . . X . . . Rd . . . (X) | None | 2     |
| LD                             | Rd, Y           | Load Indirect | Rd . . . (Y) | None | 2     |
| LD                             | Rd. . . Y       | Load Indirect and Post-Inc. | Rd . . . (Y). Y . . . Y + 1 | None | 2     |
| LDD                            | Rd, Y+q         | Load Indirect with Displacement | Rd . . . (Y+q) | None | 2     |
| LD                             | Rd, Z           | Load Indirect | Rd . . . (Z) | None | 2     |
| LD                             | Rd. . . Z       | Load Indirect and Post-Inc. | Rd . . . (Z). Z . . . Z + 1 | None | 2     |
| LDD                            | Rd. . . Z+q     | Load Indirect with Displacement | Rd . . . (Z+q) | None | 2     |
| LDS                            | Rd, k           | Load Direct from SRAM | Rd . . . (k) | None | 2     |
| ST                             | X, Rs           | Store Indirect | (X) . . . Rs | None | 2     |
| ST                             | X, Rs           | Store Indirect and Post-Inc. | (X). X . . . X + 1 | None | 2     |
| ST                             | X, Rs           | Store Indirect and Pre-Dec. | X . . . X . . . (X) | None | 2     |
| ST                             | Y, Rs           | Store Indirect | (Y) . . . Rs | None | 2     |
| ST                             | Y, Rs           | Store Indirect and Post-Inc. | (Y). Y . . . Y . . . Y + 1 | None | 2     |
| ST                             | Y, Rs           | Store Indirect and Pre-Dec. | Y . . . Y . . . (Y). Y | None | 2     |
| STD                            | Y+q, Rs         | Store Indirect with Displacement | (Y+q) . . . Rs | None | 2     |
| ST                             | Z, Rs           | Store Indirect | (Z) . . . Rs | None | 2     |
| ST                             | Z+q, Rs         | Store Indirect and Post-Inc. | (Z) . . . Z . . . Z + 1 | None | 2     |
| ST                             | Z+q, Rs         | Store Indirect and Pre-Dec. | Z . . . Z . . . (Z) . . . (Z) | None | 2     |
| STD                            | Z+q, Rs         | Store Indirect with Displacement | (Z+q) . . . Rs | None | 2     |
| STS                            | k, Rs           | Store Direct to SRAM | (k) . . . Rs | None | 2     |
| LPM                            |                    | Load Program Memory | R0 . . . (Z) | None | 3     |
| LPM                            | Rd, Z           | Load Program Memory | Rd . . . (Z) | None | 3     |
| LPM                            | Rd, Z+          | Load Program Memory and Post-Inc | Rd . . . (Z). Z . . . Z + 1 | None | 3     |
| LPM                            | Rd, Z           | Extended Load Program Memory | R0 . . . (RAMPZ.Z) | None | 3     |
| LPM                            | Rd, Z           | Extended Load Program Memory | Rd . . . (Z) | None | 3     |
| LPM                            | Rd, Z           | Extended Load Program Memory | Rd . . . (RAMPZ.Z). RAMPZ.Z . . . RAMPZ.Z+1 | None | 3     |</p>
<table>
<thead>
<tr>
<th>Mnemonics</th>
<th>Operands</th>
<th>Description</th>
<th>Operation</th>
<th>Flags</th>
<th>#Clocks</th>
</tr>
</thead>
<tbody>
<tr>
<td>SPM</td>
<td>Rd, P</td>
<td>Store Program Memory</td>
<td>(Z), R1:R0</td>
<td>None</td>
<td>-</td>
</tr>
<tr>
<td>IN</td>
<td>P, Pr</td>
<td>In Port</td>
<td>Rd . P</td>
<td>None</td>
<td>1</td>
</tr>
<tr>
<td>OUT</td>
<td>P, Pr</td>
<td>Out Port</td>
<td>P . Rr</td>
<td>None</td>
<td>1</td>
</tr>
<tr>
<td>PUSH</td>
<td>Rr</td>
<td>Push Register on Stack</td>
<td>STACK . Rr</td>
<td>None</td>
<td>2</td>
</tr>
<tr>
<td>POP</td>
<td>Rd</td>
<td>Pop Register from Stack</td>
<td>Rd . STACK</td>
<td>None</td>
<td>2</td>
</tr>
</tbody>
</table>

**MCU CONTROL INSTRUCTIONS**

<table>
<thead>
<tr>
<th>Mnemonics</th>
<th>Operands</th>
<th>Description</th>
<th>Operation</th>
<th>Flags</th>
<th>#Clocks</th>
</tr>
</thead>
<tbody>
<tr>
<td>NOP</td>
<td></td>
<td>No Operation</td>
<td></td>
<td>None</td>
<td>1</td>
</tr>
<tr>
<td>SLEEP</td>
<td></td>
<td>Sleep</td>
<td>(see specific description for Sleep function)</td>
<td>None</td>
<td>1</td>
</tr>
<tr>
<td>WDR</td>
<td></td>
<td>Watchdog Reset</td>
<td>(see specific description for WDR/timer)</td>
<td>None</td>
<td>1</td>
</tr>
<tr>
<td>BREAK</td>
<td></td>
<td>Break</td>
<td>For On-chip Debug Only</td>
<td>None</td>
<td>N/A</td>
</tr>
</tbody>
</table>
## 33. Ordering Information

### 33.1 ATmega16U4

<table>
<thead>
<tr>
<th>Speed (MHz)</th>
<th>Power Supply</th>
<th>Ordering Code</th>
<th>Default Oscillator</th>
<th>Package</th>
<th>Operation Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>16</td>
<td>2.7 - 5.5V</td>
<td>ATmega16U4-AU</td>
<td>External XTAL</td>
<td>44ML</td>
<td>Industrial (-40° to +85°C)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ATmega16U4RC-AU</td>
<td>Internal Calib. RC</td>
<td>44PW</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>ATmega16U4-MU</td>
<td>External XTAL</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>ATmega16U4RC-MU</td>
<td>Internal Calib. RC</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Package Type

<table>
<thead>
<tr>
<th>Package Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>44ML</td>
<td>ML, 44 - Lead, 10 x 10 mm Body Size, 1.0 mm Body Thickness 0.8 mm Lead Pitch, Thin Profile Plastic Quad Flat Package (TQFP)</td>
</tr>
<tr>
<td>44PW</td>
<td>PW, 44 - Lead 7.0 x 7.0 mm Body, 0.50 mm Pitch Quad Flat No Lead Package (QFN)</td>
</tr>
</tbody>
</table>
## 33.2 ATmega32U4

<table>
<thead>
<tr>
<th>Speed (MHz)</th>
<th>Power Supply</th>
<th>Ordering Code</th>
<th>Default Oscillator</th>
<th>Package</th>
<th>Operation Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>16</td>
<td>2.7 - 5.5 V</td>
<td>ATmega32U4-AU</td>
<td>External XTAL</td>
<td>44ML</td>
<td>Industrial (-40°C to +85°C)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ATmega32U4RC-AU</td>
<td>Internal Calib. RC</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>ATmega32U4-MU</td>
<td>External XTAL</td>
<td>44PW</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>ATmega32U4RC-MU</td>
<td>Internal Calib. RC</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Package Type

<table>
<thead>
<tr>
<th>Package</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>44ML</td>
<td>ML, 44 - Lead, 10 x 10 mm Body Size, 1.0 mm Body Thickness 0.8 mm Lead Pitch, Thin Profile Plastic Quad Flat Package (TQFP)</td>
</tr>
<tr>
<td>44PW</td>
<td>PW, 44 - Lead 7.0 x 7.0 mm Body, 0.50 mm Pitch Quad Flat No Lead Package (QFN)</td>
</tr>
</tbody>
</table>
34. Packaging Information

34.1 TQFP44

<table>
<thead>
<tr>
<th>SYMBOL</th>
<th>Min</th>
<th>Max</th>
<th>NOTES</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>----</td>
<td>1.20</td>
<td></td>
</tr>
<tr>
<td>A1</td>
<td>0.95</td>
<td>1.05</td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>0.09</td>
<td>0.20</td>
<td></td>
</tr>
<tr>
<td>D</td>
<td>12.00</td>
<td>BSC</td>
<td></td>
</tr>
<tr>
<td>D1</td>
<td>10.00</td>
<td>BSC</td>
<td></td>
</tr>
<tr>
<td>E</td>
<td>12.00</td>
<td>BSC</td>
<td></td>
</tr>
<tr>
<td>E1</td>
<td>10.00</td>
<td>BSC</td>
<td></td>
</tr>
<tr>
<td>J</td>
<td>0.05</td>
<td>0.15</td>
<td></td>
</tr>
<tr>
<td>L</td>
<td>0.45</td>
<td>0.75</td>
<td></td>
</tr>
<tr>
<td>e</td>
<td>0.80</td>
<td>BSC</td>
<td></td>
</tr>
<tr>
<td>f</td>
<td>0.30</td>
<td>0.45</td>
<td></td>
</tr>
</tbody>
</table>

**COMMON DIMENSIONS IN MM**

07/27/07

**TITLE**
ML, 44 - Lead, 10x10 mm Body Size, 1.0 mm Body Thickness
0.8 mm Lead Pitch, Thin Profile Plastic Quad Flat Package (TQFP)

**ATMEL Nantes S.A.**
La Chanterie - BP 70662
44506 Nantes Cedex 3 - France

**DRAWING No.**
ML

**REV.**
G
34.2 QFN44

**TOP VIEW**

**SIDE VIEW**

**DRAWINGS NOTScaled**

**COMMON DIMENSIONS IN MM**

<table>
<thead>
<tr>
<th>SYMBOL</th>
<th>MIN</th>
<th>NOM</th>
<th>MAX</th>
<th>NOTES</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0.80</td>
<td>0.85</td>
<td>0.90</td>
<td></td>
</tr>
<tr>
<td>J</td>
<td>0.00</td>
<td></td>
<td>0.05</td>
<td></td>
</tr>
<tr>
<td>D/E</td>
<td>7.00</td>
<td></td>
<td>BCC</td>
<td></td>
</tr>
<tr>
<td>D2/E2</td>
<td>5.10</td>
<td>5.20</td>
<td>5.30</td>
<td></td>
</tr>
<tr>
<td>N</td>
<td>44</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>e</td>
<td>0.50</td>
<td></td>
<td>BCC</td>
<td></td>
</tr>
<tr>
<td>L</td>
<td>0.45</td>
<td>0.55</td>
<td>0.65</td>
<td></td>
</tr>
<tr>
<td>b</td>
<td>0.18</td>
<td>0.23</td>
<td>0.30</td>
<td></td>
</tr>
</tbody>
</table>

**BOTTOM VIEW**

Compliant JEDEC Standard MO-220 variation WKKD-1

01/09/2009

**TITLE**

PW, 44 - Lead 7.0x7.0 mm Body, 0.60 mm Pitch
Quad Flat No Lead Package (QFN)

**DRAWING No.**

PW

**REV.**

0
35. Errata

The revision letter in this section refers to the revision of the ATmega16U4/ATmega32U4 device.

35.1 ATmega16U4/ATmega32U4 Rev E

- Spike on TWI pins when TWI is enabled
- High current consumption in sleep mode
- MSB of OCR4A/B/D is write only in 11-bits enhanced PWM mode

1. Spike on TWI pins when TWI is enabled
   100 ns negative spike occurs on SDA and SCL pins when TWI is enabled.

   **Problem Fix/work around**
   Enable ATmega16U4/ATmega32U4 TWI before the other nodes of the TWI network.

2. High current consumption in sleep mode
   If a pending interrupt cannot wake the part up from the selected mode, the current consumption will increase during sleep when executing the SLEEP instruction directly after a SEI instruction.

   **Problem Fix/work around**
   Before entering sleep, interrupts not used to wake up the part from the sleep mode should be disabled.

3. MSB of OCR4A/B/D is write only in 11-bits enhanced PWM mode
   In the 11-bits enhanced PWM mode the MSB of OCR4A/B/D is write only. A read of OCR4A/B/D will always return zero in the MSB position.

   **Problem Fix/work around**
   None.

35.2 ATmega16U4/ATmega32U4 Rev D

- Spike on TWI pins when TWI is enabled
- High current consumption in sleep mode
- Timer 4 11-bits enhanced PWM mode

1. Spike on TWI pins when TWI is enabled
   100 ns negative spike occurs on SDA and SCL pins when TWI is enabled.

   **Problem Fix/work around**
   Enable ATmega16U4/ATmega32U4 TWI before the other nodes of the TWI network.

2. High current consumption in sleep mode
   If a pending interrupt cannot wake the part up from the selected mode, the current consumption will increase during sleep when executing the SLEEP instruction directly after a SEI instruction.

   **Problem Fix/work around**
   Before entering sleep, interrupts not used to wake up the part from the sleep mode should be disabled.
3. Timer 4 11-bits enhanced PWM mode
   Timer 4 11-bits enhanced mode is not functional.
   **Problem Fix/work around**
   None.

### 35.3 ATmega16U4/ATmega32U4 Rev C
Not sampled

### 35.4 ATmega16U4/ATmega32U4 Rev B
- Spike on TWI pins when TWI is enabled
  100 ns negative spike occurs on SDA and SCL pins when TWI is enabled.
  **Problem Fix/work around**
  Enable ATmega16U4/ATmega32U4 TWI before the other nodes of the TWI network.

- High current consumption in sleep mode
  If a pending interrupt cannot wake the part up from the selected mode, the current consumption will increase during sleep when executing the SLEEP instruction directly after a SEI instruction.
  **Problem Fix/work around**
  Before entering sleep, interrupts not used to wake up the part from the sleep mode should be disabled.

- Incorrect execution of VBUSTI interrupt
  The CPU may incorrectly execute the interrupt vector related to the VBUSTI interrupt flag.
  **Problem fix/work around**
  Do not enable this interrupt. Firmware must process this USB event by polling VBUSTI.

- Timer 4 11-bits enhanced PWM mode
  Timer 4 11-bits enhanced mode is not functional.
  **Problem Fix/work around**
  None.
35.5 ATmega16U4/ATmega32U4 Rev A

- Spike on TWI pins when TWI is enabled
- High current consumption in sleep mode
- Increased power consumption in power-down mode
- Internal RC oscillator start up may fail
- Internal RC oscillator calibration
- Incorrect execution of VBUSTI interrupt
- Timer 4 enhanced mode issue

1. Spike on TWI pins when TWI is enabled
   100 ns negative spike occurs on SDA and SCL pins when TWI is enabled.
   
   **Problem Fix/work around**
   Enable ATmega16U4/ATmega32U4 TWI before the other nodes of the TWI network.

2. High current consumption in sleep mode
   If a pending interrupt cannot wake the part up from the selected mode, the current consump-
   tion will increase during sleep when executing the SLEEP instruction directly after a SEI
   instruction.
   
   **Problem Fix/work around**
   Before entering sleep, interrupts not used to wake up the part from the sleep mode should
   be disabled.

3. Increased power consumption in power-down mode
   The typical power consumption is increased by about 30 μA in power-down mode.
   
   **Problem Fix/work around**
   None.

4. Internal RC oscillator start up may fail
   When the part is configured to start on internal RC oscillator, the oscillator may not start
   properly after power-on.
   
   **Problem Fix/work around**
   Do not configure the part to start on internal RC oscillator.

5. Internal RC oscillator calibration
   8 MHz frequency can be impossible to reach with internal RC even when using maximal
   OSCAL value.
   
   **Problem Fix/work around**
   None.

6. Incorrect execution of VBUSTI interrupt
   The CPU may incorrectly execute the interrupt vector related to the VBUSTI interrupt flag.
   
   **Problem fix/work around**
   Do not enable this interrupt. Firmware must process this USB event by polling VBUSTI.
7. Timer 4 11-bits enhanced PWM mode
Timer 4 11-bits enhanced mode is not functional.

Problem Fix/work around
None.
36. Datasheet Revision History for ATmega16U4/ATmega32U4

Please note that the referring page numbers in this section are referred to this document. The referring revision in this section are referring to the document revision.

36.1 Rev. 7766F – 11/10

1. Replaced the “QFN44” on page 418 by an updated drawing.
2. Updated “ADC Control and Status Register B – ADCSRB” on page 289. Defined the ADCSRB register as in “ADC Control and Status Register B – ADCSRB” on page 312.
3. Updated the last page according to Atmel new Brand Style Guide.

36.2 Rev. 7766E – 04/10

1. Updated “Features” on page 1.
2. Updated “Features” on page 253.
3. Updated Figure 21-9 on page 258.
4. Updated Section 21.8 on page 260.
5. Updated “Features” on page 292.
7. Updated “Program And Data Memory Lock Bits” on page 346.
8. Updated Table 28-5 on page 348.
10. Updated Figure 29-2 on page 381.
11. Added “Typical Characteristics” on page 386.
13. Updated “Errata” on page 419.

36.3 Rev. 7766D – 01/09

1. Updated Memory section in “Features” on page 1.
2. Added section “Resources” on page 8.
3. Added section “Data Retention” on page 8.
4. Updated “Ordering Information” on page 415.

36.4 Rev. 7766C – 11/08

1. Updated Memory section in “Features” on page 1.
36.5 Rev. 7766B – 11/08

1. Added ATmega16U4 device.
2. Created errata section and added ATmega16U4.
3. Updated High Speed Timer, asynchronous description Section 15. on page 139

36.6 Rev. 7766A – 07/08

1. Initial revision
# Table of Contents

1. **Pin Configurations** ................................................................. 3
2. **Overview** ................................................................. 3
   2.1 Block Diagram .............................................................. 4
   2.2 Pin Descriptions .......................................................... 5
3. **About** ................................................................. 8
   3.1 Disclaimer ............................................................... 8
   3.2 Resources ................................................................. 8
   3.3 Code Examples .......................................................... 8
   3.4 Data Retention .......................................................... 8
4. **AVR CPU Core** ........................................................... 9
   4.1 Introduction ............................................................... 9
   4.2 Architectural Overview .................................................. 9
   4.3 ALU – Arithmetic Logic Unit ........................................... 10
   4.4 Status Register .......................................................... 11
   4.5 General Purpose Register File ....................................... 12
   4.6 Stack Pointer ............................................................. 13
   4.7 Instruction Execution Timing .......................................... 14
   4.8 Reset and Interrupt Handling ........................................ 15
5. **AVR ATmega16U4/ATmega32U4 Memories** .............................. 18
   5.1 In-System Reprogrammable Flash Program Memory ............... 18
   5.2 SRAM Data Memory ...................................................... 19
   5.3 EEPROM Data Memory ................................................ 21
   5.4 I/O Memory ............................................................... 26
6. **System Clock and Clock Options** ........................................ 27
   6.1 Clock Systems and their Distribution ................................ 27
   6.2 Clock Sources ........................................................... 28
   6.3 Low Power Crystal Oscillator ......................................... 29
   6.4 Low Frequency Crystal Oscillator .................................... 31
   6.5 Calibrated Internal RC Oscillator .................................... 32
   6.6 External Clock ............................................................ 33
   6.7 Clock Switch .............................................................. 34
   6.8 Clock switch Algorithm ............................................... 35
   6.9 Clock Output Buffer .................................................... 37
7 **Power Management and Sleep Modes** ............................................. 43
   7.1 Idle Mode .................................................................................. 44
   7.2 ADC Noise Reduction Mode .................................................... 44
   7.3 Power-down Mode ..................................................................... 44
   7.4 Power-save Mode ..................................................................... 44
   7.5 Standby Mode .......................................................................... 45
   7.6 Extended Standby Mode ........................................................... 45
   7.7 Power Reduction Register ......................................................... 45
   7.8 Minimizing Power Consumption .............................................. 47

8 **System Control and Reset** ............................................................. 49
   8.1 Internal Voltage Reference ....................................................... 54
   8.2 Watchdog Timer ........................................................................ 55

9 **Interrupts** .................................................................................. 61
   9.1 Interrupt Vectors in ATmega16U4/ATmega32U4 .......................... 61

10 **I/O-Ports** .................................................................................... 65
   10.1 Introduction ............................................................................ 65
   10.2 Ports as General Digital I/O ..................................................... 66
   10.3 Alternate Port Functions .......................................................... 70
   10.4 Register Description for I/O-Ports ............................................ 82

11 **External Interrupts** ..................................................................... 85

12 **Timer/Counter0, Timer/Counter1, and Timer/Counter3 Prescalers ... 89**
   12.1 Internal Clock Source ............................................................... 89
   12.2 Prescaler Reset ....................................................................... 89
   12.3 External Clock Source ............................................................. 89
   12.4 General Timer/Counter Control Register – GTCCR .................. 90

13 **8-bit Timer/Counter0 with PWM** ............................................... 91
   13.1 Overview ................................................................................ 91
   13.2 Timer/Counter Clock Sources .................................................. 92
   13.3 Counter Unit .......................................................................... 92
   13.4 Output Compare Unit .............................................................. 93
   13.5 Compare Match Output Unit .................................................... 95
   13.6 Modes of Operation ............................................................... 96
   13.7 Timer/Counter Timing Diagrams ............................................. 100
13.88-bit Timer/Counter Register Description ...................................... 102

14 16-bit Timers/Counters (Timer/Counter1 and Timer/Counter3) ...... 108
14.1 Overview .................................................................................. 108
14.2 Accessing 16-bit Registers .......................................................... 110
14.3 Timer/Counter Clock Sources ...................................................... 113
14.4 Counter Unit .............................................................................. 114
14.5 Input Capture Unit ...................................................................... 115
14.6 Output Compare Units ................................................................. 117
14.7 Compare Match Output Unit ......................................................... 119
14.8 Modes of Operation .................................................................... 120
14.9 Timer/Counter Timing Diagrams .................................................. 127
14.10 16-bit Timer/Counter Register Description .................................. 129

15 10-bit High Speed Timer/Counter 4 ............................................... 138
15.1 Features .................................................................................... 138
15.2 Overview ................................................................................... 138
15.3 Counter Unit .............................................................................. 142
15.4 Output Compare Unit .................................................................. 143
15.5 Dead Time Generator .................................................................. 145
15.6 Compare Match Output Unit ......................................................... 146
15.7 Synchronous Update .................................................................... 149
15.8 Modes of Operation ..................................................................... 149
15.9 Timer/Counter Timing Diagrams .................................................. 156
15.10 Fault Protection Unit .................................................................. 157
15.11 Accessing 10-Bit Registers .......................................................... 159
15.12 Register Description .................................................................. 162

16 Output Compare Modulator (OCM1C0A) ...................................... 175
16.1 Overview .................................................................................... 175
16.2 Description ................................................................................ 175

17 Serial Peripheral Interface – SPI .................................................... 177
17.1 SS Pin Functionality ..................................................................... 181
17.2 Data Modes ................................................................................ 184

18 USART .......................................................................................... 186
18.1 Overview .................................................................................... 186
18.2 Clock Generation ......................................................................... 187
18.3 Frame Formats .................................................................................. 190
18.4 USART Initialization ........................................................................ 192
18.5 Data Transmission – The USART Transmitter .................................... 193
18.6 Data Reception – The USART Receiver .............................................. 195
18.7 Asynchronous Data Reception .......................................................... 199
18.8 Multi-processor Communication Mode .............................................. 202
18.9 Hardware Flow Control ..................................................................... 203
18.10 USART Register Description .......................................................... 205
18.11 Examples of Baud Rate Setting ....................................................... 209

19 USART in SPI Mode ........................................................................ 214
  19.1 Overview ...................................................................................... 214
  19.2 Clock Generation .......................................................................... 214
  19.3 SPI Data Modes and Timing .......................................................... 215
  19.4 Frame Formats ............................................................................. 215
  19.5 Data Transfer ................................................................................ 217
  19.6 USART MSPIM Register Description ........................................... 219
  19.7 AVR USART MSPIM vs. AVR SPI ................................................ 221

20 2-wire Serial Interface ..................................................................... 223
  20.1 Features ....................................................................................... 223
  20.22-wire Serial Interface Bus Definition ............................................. 223
  20.3 Data Transfer and Frame Format .................................................. 224
  20.4 Multi-master Bus Systems, Arbitration and Synchronization ............ 227
  20.5 Overview of the TWI Module ....................................................... 228
  20.6 TWI Register Description ............................................................ 231
  20.7 Using the TWI ............................................................................. 234
  20.8 Transmission Modes .................................................................... 238
  20.9 Multi-master Systems and Arbitration ......................................... 251

21 USB controller ............................................................................... 253
  21.1 Features ...................................................................................... 253
  21.2 Block Diagram ............................................................................ 253
  21.3 Typical Application Implementation .............................................. 254
  21.4 Crystal-less operation .................................................................. 256
  21.5 Design guidelines ....................................................................... 256
  21.6 General Operating Modes ............................................................ 257
  21.7 Power modes ............................................................................. 259
22 USB Device Operating modes .............................................. 266
   22.1 Introduction ....................................................... 266
   22.2 Power-on and reset ............................................... 266
   22.3 Endpoint reset .................................................... 266
   22.4 USB reset ........................................................... 267
   22.5 Endpoint selection ................................................ 267
   22.6 Endpoint activation .............................................. 267
   22.7 Address Setup ..................................................... 268
   22.8 Suspend, Wake-up and Resume .................................. 269
   22.9 Detach ............................................................... 269
   22.10 Remote Wake-up .................................................. 270
   22.11 STALL request ................................................... 270
   22.12 CONTROL endpoint management .............................. 271
   22.13 OUT endpoint management ..................................... 272
   22.14 IN endpoint management ....................................... 274
   22.15 Isochronous mode ............................................... 275
   22.16 Overflow .......................................................... 276
   22.17 Interrupts .......................................................... 276
   22.18 Registers ........................................................... 277

23 Analog Comparator ...................................................... 289
   23.1 Analog Comparator Multiplexed Input .......................... 291

24 Analog to Digital Converter - ADC .................................. 292
   24.1 Features ............................................................. 292
   24.2 Operation ............................................................ 294
   24.3 Starting a Conversion .............................................. 294
   24.4 Prescaling and Conversion Timing ............................... 295
   24.5 Changing Channel or Reference Selection ..................... 298
   24.6 Temperature Sensor ............................................... 299
   24.7 ADC Noise Canceler .............................................. 301
24.8 ADC Conversion Result ................................................................. 305
24.9 ADC Register Description .......................................................... 307

25  JTAG Interface and On-chip Debug System ................................. 314
   25.1 Overview ................................................................................. 314
   25.2 Test Access Port – TAP .......................................................... 314
   25.3 TAP Controller ..................................................................... 316
   25.4 Using the Boundary-scan Chain .............................................. 317
   25.5 Using the On-chip Debug System ............................................ 317
   25.6 On-chip Debug Specific JTAG Instructions ............................ 318
   25.7 On-chip Debug Related Register in I/O Memory ...................... 319
   25.8 Using the JTAG Programming Capabilities ............................. 319
   25.9 Bibliography .......................................................................... 319

26  IEEE 1149.1 (JTAG) Boundary-scan ........................................... 320
   26.1 Features ................................................................................ 320
   26.2 System Overview ................................................................. 320
   26.3 Data Registers ...................................................................... 320
   26.4 Boundary-scan Specific JTAG Instructions ............................ 322
   26.5 Boundary-scan Related Register in I/O Memory ...................... 323
   26.6 Boundary-scan Chain ............................................................. 324
   26.7 ATmega16U4/ATmega32U4 Boundary-scan Order .................... 327
   26.8 Boundary-scan Description Language Files .............................. 329

27  Boot Loader Support – Read-While-Write Self-Programming .......... 330
   27.1 Boot Loader Features ............................................................ 330
   27.2 Application and Boot Loader Flash Sections ........................... 330
   27.3 Read-While-Write and No Read-While-Write Flash Sections ...... 330
   27.4 Boot Loader Lock Bits ............................................................ 333
   27.5 Entering the Boot Loader Program ......................................... 334
   27.6 Addressing the Flash During Self-Programming ....................... 337
   27.7 Self-Programming the Flash .................................................. 338

28  Memory Programming ................................................................. 346
   28.1 Program And Data Memory Lock Bits .................................... 346
   28.2 Fuse Bits .............................................................................. 347
   28.3 Signature Bytes .................................................................... 349
   28.4 Calibration Byte .................................................................... 349
   28.5 Parallel Programming Parameters, Pin Mapping, and Commands .... 349
35.1 ATmega16U4/ATmega32U4 Rev E ................................................................. 419
35.2 ATmega16U4/ATmega32U4 Rev D ................................................................. 419
35.3 ATmega16U4/ATmega32U4 Rev C ................................................................. 420
35.4 ATmega16U4/ATmega32U4 Rev B ................................................................. 420
35.5 ATmega16U4/ATmega32U4 Rev A ................................................................. 421

36  Datasheet Revision History for ATmega16U4/ATmega32U4 .......... 423
    36.1 Rev. 7766F – 11/10 ................................................................. 423
    36.2 Rev. 7766E – 04/10 ................................................................. 423
    36.3 Rev. 7766D – 01/09 ................................................................. 423
    36.4 Rev. 7766C – 11/08 ................................................................. 424
    36.5 Rev. 7766B – 11/08 ................................................................. 424
    36.6 Rev. 7766A – 07/08 ................................................................. 424
Reference Designs ARE PROVIDED “AS IS” AND “WITH ALL FAULTS. Arduino DISCLAIMS ALL OTHER WARRANTIES, EXPRESS OR IMPLIED, REGARDING PRODUCTS, INCLUDING BUT NOT LIMITED TO, ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Arduino may make changes to specifications and product descriptions at any time, without notice. The Customer must not rely on the absence or characteristics of any features or instructions marked “reserved” or “undetermined.” Arduino reserves these for future definition and shall have no responsibility whatsoever for conflicts or incompatibilities arising from future changes to them. The product information on the Web Site or Materials is subject to change without notice. Do not finalize a design with this information. ARDUINO is a registered trademark.
Li-ion Boe-Bot Power Pack-Charger (#28988)

The Li-ion Boe-Bot Power Pack-Charger is an integrated battery, power supply, and charging system designed specifically for the Boe-Bot robot. It is designed to replace the 4-cell AA battery pack that comes standard on the Boe-Bot.

The Power Pack-Charger is powered by two 18650 Li-ion cells, providing approximately 7.4 VDC @ 2000 to 2600 mAh, depending on which cells you use. Parallax's #28987 cells are rated at 2600 mAh, and have built-in protections against over-discharging or excessive current conditions. This charger/power pack will work with most protected and un-protected Li-ion 18650 cells.

Features

• Fits into the Boe-Bot's chassis, replacing the 4-cell AA battery pack
• Holds two rechargeable/replaceable 3.7 volt Li-ion 18650 cells
• Provides up to 6 hours of continuous motorized operation (depending on cell capacity)
• Rechargeable Li-ion cells don't need to be removed from your Boe-Bot for recharging—charging circuitry is built into the board.
• Charge with a 7.5 VDC, 2.1 mm center-positive barrel jack wall transformer (Parallax part #750-00009, or equivalent).
• Automatic charge/discharge switching circuitry and on-board output fuse protection
• Multiple LED indicators provide charge readiness information for each individual cell; status key for the LED indicators is printed on the board.
• Aggressive holders retain cells in any board orientation and in moderate shock environments, such as mobile robotic applications.
• Dedicated circuitry continuously monitors the charging process to ensure safety, efficiency, and to maximize the number of charge/discharge cycles of each cell.

Key Specifications

• Charging Power Requirements:  +5–12 VDC @ 1 A (min.);  2.1 mm center positive barrel jack supply (#750-00009 works well)
• Power Output: Unregulated nominal 7.4 VDC @ 1800–2600 mAh (depending on cells used)
• Dimensions:  2.65 x 3.05 x 1.00 in (7.6 x 10.2 x 2.54 cm)
• Charging Time:  1 to 6 hours, depending upon the discharge level and capacity of the cells used

Application Ideas

• Additional power for Boe-Bot accessories
• Extended run-time for your Boe-Bot
Packing List

- (1) Li-ion Boe-Bot Power-Pack / Charger – 2 Cell PCB, 2.63 in x 3.0 in (6.7 cm x 7.6 cm)
- (1) Li-ion Battery Cable (#802-00020)
- (2) 2-amp fuses; 1 pre-installed, 1 spare (#452-00065)
- (3) Hex Aluminum F/M standoffs (#713-00024)
- (3) #4-40 x ¼” pan head screws (#700-00028)
- (3) #4-40 hex nuts (#700-00003)

Additional Items Required

- (2) Li-ion 18650-size cells (#28987 or similar)
- +5 to +12 VDC power supply, center positive, + 2.1 mm barrel jack (#750-00009 or equiv.)
- Safety glasses
- Small Phillips screwdriver
- Needle-nose pliers
- Multi-meter (VOM)

Assembly Instructions

**CAUTION:** Do not install the Li-ion cells until you reach the appropriate step.

**Step 1.** Set the jumper on your Boe-Bot’s Board of Education to power the servo headers from Vdd. Setting the jumper so that the servos are powered from Vin will power the servo with the 7.4V input, which is above the 6.0V specification for the servos. Although extended testing in Parallax has not produced any failures at voltages less than 8 volts, this testing did not sample all prior servo manufacturing lots, so the potential for failure or damage at 7.4V has not been determined.

![Image of jumper settings](image-url)
**Step 2.** Remove the AA batteries from your Boe-Bot.

**Step 3.** Remove the two drive wheels and the rear caster ball from your Boe-Bot.

**Step 4.** Turn the chassis over so that it is oriented as shown below. Remove the (4) cell AA battery holder by loosening the two flat-head screws and nuts that attach it to the chassis (Figure 1).

![Figure 1](image-url)
**Step 5.** Loosen, but do not remove, the (2) #4-40 screws that are closest to the “bottom” of the Boe-Bot, as shown in Figure 2.

![Boe-Bot, upside-down, wheels removed](image)

**Step 6.** Remove the (2) nuts and (2) screws that are closest to the top of the chassis, as shown in Figure 2.

![Boe-Bot, upside-down, side view](image)

*Figure 2*
**Step 7.** Insert the (3) F/M standoffs into the slots, as shown in Figure 3. The standoffs should be on the “inside” of the Boe-Bot chassis. Use (3) #4-40 nuts to loosely attach them to the chassis. Do not tighten yet.

**Step 8.** Plug the power cable into the Power Pack output jack and thread the cable through the hole to the top side of the chassis.
Step 9. Carefully insert the Li-ion Power-Pack at a slight angle, into the chassis, as shown in the photo. Insert the board so as to not crimp the servo cables. Use (3) #4-40 x ¼" long pan head screws to attach the Li-ion Power Pack to the standoffs. Tighten the (3) nuts that hold the standoffs to the chassis, as shown in Figure 4.
**Step 10.** Re-attach the Boe-Bot drive wheels and its Ball Caster.

**Step 11.** Carefully remove the cells from their packaging. Note the positive cell polarity. Various brands of cells are marked differently. The positive terminal may be indicated by a ringed indentation near one end of the cell, or the packaging may be printed with “+” and/or “-” designators (as shown in Figure 5).

![Figure 5](image.png)

Place the bottom (negative) end of a cell against the springs in Cell Holder “A”, and then gently slide it down and in at a slight angle into the holder until it snaps into place. Repeat for Cell Holder “B.” Note: Cells without internal PCB’s will snap into the holders easily.

Higher quality cells (those that have internal PCB protection such as Parallax #28987), are slightly longer. They will fit into the holder, but they are a very tight fit. The holder’s ends will flex a bit as you insert the cell. Sight along the sides of the cells to make sure that they are completely seated into the cell holders.

If you need to remove the cells (i.e. when the cells wear out), you can use a small, flat, piece plastic (such as a comb or similar) or a small pair of needle-nose pliers with tape or heat-shrink tubing attached to the ends (to prevent shorting) to gently pry them out.

Lithium cells come pre-charged, so treat them carefully—they already contain a significant amount of energy. Handle with care and do not short the terminals!

**Step 12.** Connect a center-positive 2.1 mm wall transformer (such as Parallax #750-00009) to the Charging Power Input jack as noted in Figure 4. The power supply should have a voltage output between +5 VDC and +12 VDC. The amount of current necessary to charge the cells is controlled automatically by the circuitry; however your wall transformer should be rated at 1 amp or more to minimize charging time. If the charging current supply is too low, the charging circuits will not operate. Also, a lower current will not harm the cells; they’ll just take longer to charge.

**Step 13.** Upon application of charging power, the green LED (visible through the slot on the right rear) will turn on. After a few moments, the blue status indicator LEDs should come on. Solid blue means that the cells are being charged.
Circuit Description and Operation

**CAUTION:** The PCB near the cell holders may get hot! However, when the Li-ion Power Pack is properly installed in the under-belly of the Boe-Bot, any potentially “hot” areas on the PCB are safely protected.

With cells installed in their respective holders, and with no external connections to the input/output jacks, the circuitry is inactive and there is no current flow (other than some very, very small leakage current through the inactive charging circuits).

Upon application of +5 to +12 VDC to the Charging Power Input jack (J1), the following happens:

a) The Charging Power Input (Green LED) is activated.
b) Each cell is electrically disconnected and isolated from the other.
c) The Cell Power Output jacks are disconnected from the cells, and disabled.
d) The dual charging circuits begin a qualification mode to determine each cell’s characteristics.
e) After checking the cells, LED status indicators are activated. If required, each cell begins charging their respective cells.

When Charging Power is removed, it results in the following:

a) Cell charging circuits are disabled and Status indicator LEDs are disabled.
b) The cells are electrically connected into a series configuration.
c) Cell Power Output jack is connected to the cells, resulting in a 7.4 VDC (max.) output.

Jack/Plug/Indicators Functional Descriptions

**Charge Power Input:** 2.1 mm barrel jack, center positive. +5 to +12 VDC input. Do not reverse the input voltage. Charging time is dependent on the amperage available from the power supply you choose, as well as the capacity of the cells you choose.

**Cell Power Output:** Polarized, right angle, female, bottom pin positive.

This Power Output jack is the output from the cells.

When there is no Charge Power Input (i.e. when the wall charger is disconnected from the PCB), the two 3.7 volt cells are electrically connected together in a series configuration, and the resulting power (3.7 VDC x 2 cells = 7.4 VDC) is available at this jack.

Upon application of Charge Power (such as from a wall transformer), Cell Power Output is disconnected from the on-board cells, and this jack is disabled.

**Green LED:** Charge Power Indicator—whenever charging power is applied to the Charging Power Input barrel jack, the board is receiving power.

**Cell A Blue Status:** This LED indicates the charging status of Cell A.
- Solid = The cell is charging.
- Blinking = The cell has been fully charged.
- Off = The cell was already fully charged and no charging process was needed, or there is no cell in the holder.
**Cell A Red Status:** This LED indicates a fault condition in Cell A.
- **Solid** = There is/was a fault in the cell, or there was a glitch during the charging process. Remove and then re-apply power to the board, to see if the condition persists.
- **Blinking** = The temperature of the cell is outside the safe charging zone. The safe charging zone is typically set for between 32 and 113 °F (0 to 45 °C).
- **Off** = There is no fault condition detected with Cell A, or there is no cell in the holder.

**Cell B Blue Status:** This LED indicates the charging status of Cell B.
- **Solid** = The cell is charging.
- **Blinking** = The cell has been fully charged.
- **Off** = The cell was already fully charged and no charging process was needed, or there is no cell in the holder.

**Cell B Red Status:** This LED indicates a fault condition in Cell B.
- **Solid** = There is/was a fault in the cell, or there was a glitch during the charging process. Remove and then re-apply power to the board, to see if the condition persists.
- **Blinking** = The temperature of the cell is outside the safe charging zone. The safe charging zone is typically set for between 32 and 113 °F (0 to 45 °C).
- **Off** = There is no fault condition detected with Cell B, or there is no cell in the holder.

**Application Ideas**

Your Boe-Bot is now packed with a lot of energy. It can now go several **miles** on a single charge! You can also operate any number of new devices, such as additional servos, cameras, sensors, etc., with no fear of running out of power anytime soon.

Used properly, your Li-ion Power-Pack can replace several thousand “AA” alkaline batteries! This is both good for our environment as well as your pocketbook.