Microcontroller Design: Control Electronics

for

Rantec’s 28VDC Input 3-U Power Supply

By

Leo Hernandez

Senior Project

ELECTRICAL ENGINEERING DEPARTMENT

California Polytechnic State University

San Luis Obispo

December, 2013

TABLE OF CONTENTS

LIST OF TABLES AND FIGURES

Acknowledgements

I would like to thank the following people:

Cal Poly Electrical Engineering Professor, David Braun for providing guidance and support on my senior project. His motivation and technical support allowed me to complete this project on time.

Rantec's Vice President of Engineering, Bryan Kellogg for funding and giving me an opportunity to work on this project and for his support throughout my undergraduate education.

Rantec's Director of Test Engineering and Software Development, Tom Wilkins for proving software support, for supporting my undergraduate education, and for being a mentor throughout my career. His software experience allowed me to develop complex C –Code drivers.

Jeremy Edmonds, for volunteering his time to support me with the printed circuit board design. His experience in PCB design and knowledge of PCB houses allowed me to get a 1-day turn-around on the printed circuit board for this project.

Abstract

The Control Electronics is a real-time data acquisition and control system based on Atmel's AVR microcontroller. The Control Electronics will be embedded into Rantec's 3U power supply to monitor and report power supply data [1]. The power supply data monitored includes power supply output voltages, input voltage, DC bus voltage, and temperature.

The microcontroller continuously monitors DC voltages through seven I^2C compatible analog-to-digital converters via a custom Two-Wire interface. An analog temperature sensor measures power supply data and feeds it as an analog voltage to the microcontroller's built-in analog to digital interface. The microcontroller stores the acquired power supply data into a buffer for easy processing and to maximize its performance.

The Control Electronics reports the power supply data to an external Two-Wire master device via the microcontroller's built-in Two-Wire interface, configured as slave.

This project describes the development and implementation of the Control Electronics data acquisition and control system.

I. Introduction

Rantec is an engineering company that designs and manufactures a variety of ruggedized switching power supplies (DC-DC converters). Over the last four years, Rantec has supplied various power supplies with prognostic capabilities to several of its customers. The Control Electronics project attempts to embed a microcontroller into Rantec's 3U VME power supply to monitor and report power supply data. Rantec recommended Atmel's AVR 8-bit microcontroller [2] to be part of The Control Electronics project instead of a Complex Programmable Logic Device (CPLD) or a Field Programmable Gate Array (FPGA) for data acquisition and system control, due to its flexibility for timing requirements and its low cost. Also, due to Rantec's experience in programming in C-language and in LabVIEW development environment, this project used the CodeVision AVR along with Atmel's Studio C-Compiler to develop the firmware and program the microcontroller. LabVIEW development environment was used to design the Graphical User Interface (GUI) for the user to view and analyze the power supply data.

The Control Electronics features include power supply real-time data acquisition, power supply overvoltage and overtemperature protection. The Control Electronics' fundamental components include one microcontroller, seven I^2C analog-to-digital converters, and one temperature sensor. Rantec selected Atmel's ATmega 164P/364P microcontroller, National Semiconductor's ADC121C021 I^2C-compatible 12-bit analog-to-digital converter, and Texas Instruments' LM94021 analog temperature sensor for the data acquisition system.

This report discusses the electrical, firmware, and software design of the Control Electronics. The electrical design consists selecting and integrating commercial off-the-shelf components (COTS) into a printed circuit. The printed circuit board contains a Joint Test Action Group interface, also known as a JTAG interface. The JTAG interface provides easy access for programming the microcontroller and for on-chip debugging. Section III discusses the electrical design in greater detail.

The firmware design includes developing and implementing C-Code drivers for data gathering and external communication. Specifically, Rantec requested the development of a Two-Wire Interface Master Bit Bang driver for the communication between the microcontroller and seven I^2C analog to digital converters. This driver implements the internal Two-Wire communication protocol by bit banging two microcontroller GPIO pins, configured as TWI_SDA and TWI_SCL signals [5]. This approach was chosen due to the limited microcontroller Two-Wire interface resources. Section IV explains the development of C-Code drivers in detail.

The software design describes the development and testing of a Graphical User Interface (GUI). This software application contains graphical icons and visual indicators to allow the user to interact with the Control Electronics.

II. Requirements and Specifications

Background

Rantec's 3U power supply is a 28VDC input DC-DC Converter with 5 DC output voltages. The DC output voltages are: +3.3V, +5V, +12V, +3.3V_AUX, and -12V_AUX. The 3U power supply has two control signals, /ENABLE and /INHIBIT, and three status signals, /SYSRESET, /FAIL, and INPUT_LOW. The goal of this project is to design and embed a microcontroller-based system into Rantec's 3U power supply.

Electrical Performance

Table I contains the Control Electronics requirements and specifications. These requirements and specifications derive directly from Rantec's document SD35798A.

TABLE I: REQUIREMENTS SPECIFICATIONS FOR THE CONTROL ELECTRONICS

Marketing Requirements	Engineering Specifications	Justification
1	The Control Electronics monitors the output voltages, input voltage, and intermediate bus voltage of the 3U power supply.	This feature provides real time output voltage data.
2	The output voltage readback accuracy is ±1%. The voltage resolution is 100mV.	Voltage readback accuracy of ±1% is typical in the power supply industry.
3	The Control Electronics monitors the temperature of the 3U power supply.	The temperature of the power supply is monitored to prevent a catastrophic failure.
4	The temperature readback accuracy is ±3°C.	The temperature readback accuracy
5	The Control Electronics reports power supply voltage and temperature data to the Master Unit via the I^2C interface bus.	The Master Unit could be a computer or the next assembly such as an aircraft or a vehicle.
6	All output voltages are enabled/ disabled individually by the microcontroller.	The outputs are enabled individually to reduce input inrush current.
3,4,6,7	All outputs are protected against an over temperature condition. Should the chassis temperature of the power supply exceed +85°C (+5°C /-0°C), all of the outputs are turned off by the microcontroller.	The temperature of the power supply is monitored and if the temperature exceeds +85°C, then the outputs are turned off. The overtemperature feature prevents a catastrophic failure.
1,2,6,8	The Control Electronics reports an overvoltage condition to an external I^2C device if the output voltage is between 110-125% of the rated output. The output shall be turned off by the microcontroller if an over voltage condition occurs.	The overvoltage protection feature prevents the output voltage from reaching a dangerous level.
9	The /ENABLE and /INHIBIT control signals determine which output is turned on. See Table II.	These control signals allow the user to turn on or off the power supply output voltages without interrupting the input voltage.

10	The /SYSRESET signal is asserted high 200mS after the 5V Output is within specification following turn-on.	The /SYSRESET status signals allows the user to visually identify a fault with the +5V output voltage. The timing requirement required by the customer.
11	The /FAIL signal is asserted open collector only when all of the outputs are nominally within +/- 10% of their output voltage rating.	The /FAIL indicates that the power supply is operating in a normal condition.
14	The operating temperature range is -55°C to +85°C.	The Control electronics must operate at extreme temperatures.
15	The storage temperature range is -55°C to +125°C.	The Control electronics must survive extreme temperatures.
16	The labor and parts cost not exceed $9,000	The budget is based on the estimated labor and material cost.
14	The Two-Wire Interface Bit Ban driver Is compatible with I^2C devices.	
Marketing Requirements **1.** Output voltage monitor **2.** Output voltage accuracy and resolution **3.** Temperature monitor **4.** Temperature readback accuracy **5.** External communication interface **6.** Output voltages enabled/disabled **7.** Thermal shutdown protection **8.** Over voltage protection **9.** /ENABLE & /INHIBIT control inputs **10.** /SYSRESET status signal		**11.** /FAIL status signal **12.** Extreme operating temperature **13.** Extreme Storage Temperature **14.** Low Cost **15.** Two-Wire Interface Bit Bang Driver

The requirements and specifications table format derives from [3], Chapter 3

Turn-on Sequencing

Upon initial application of the input voltage, all of the outputs are sequenced enabled by the microcontroller. See Figure 1 for the turn-on sequencing.

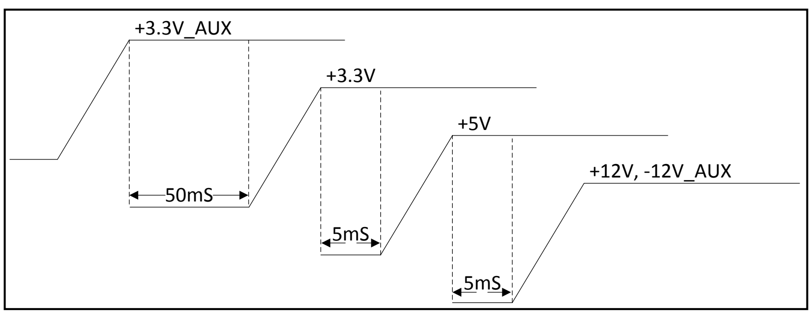


Figure 1: Turn-On Sequence

Table II shows the control inputs that enable the power supply outputs.

TABLE II: OUTPUT VOLTAGE CONTROL

Control Input		Power Supply Outputs	
/ENABLE	**/INHIBIT**	**3.3V_AUX**	**+3.3V, +5V, +12V, -12V_AUX**
High	High	Off	Off
High	Low	Off	Off
Low	High	On	On
Low	Low	On	Off

EMI, Operating temperature, Operating Lifetime, Weight, Dimensions, and Packaging

Electromagnetic Interference (EMI), operating temperature, operating lifetime, weight, dimensions, and packaging is addressed at the power supply level.

III. Electrical Design

Level 0 Block Diagram

Figure 2 shows the Level 0 bock diagram for the Control Electronics. The inputs are power supply temperature, power supply input and output voltages, and control inputs. The outputs are the power supply outputs status signals and the control outputs. The communication interface is the I^2C interface.

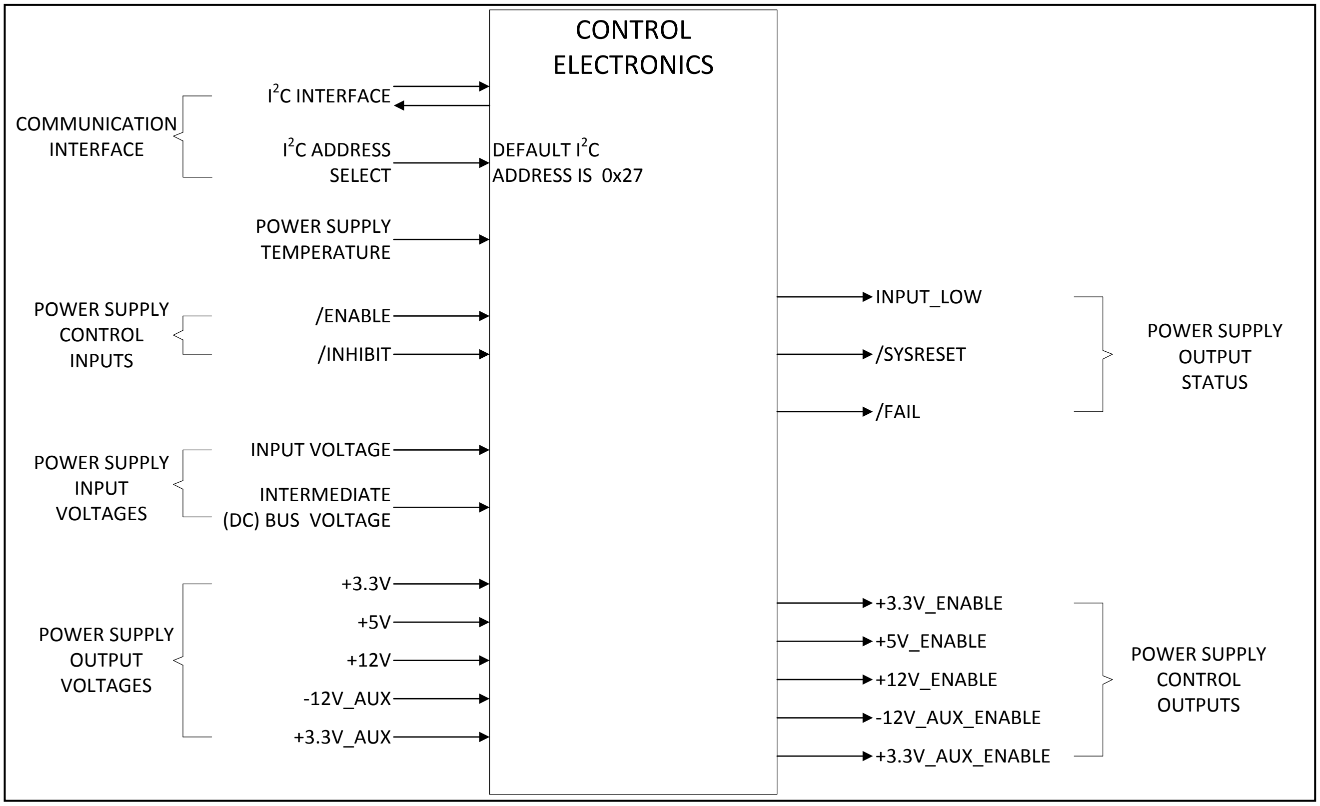


Figure 2: Level 0 Control Electronics Block Diagram

Table III describes the Control Electronics Level 0 Functionality.

TABLE III: LEVEL 0 FUNCTIONALITY

Module	Control Electronics
Inputs	TEMPERATURE - temperature of the unit /ENABLE - Control signal to enable the power supply /INHIBIT - Control signal to enable the power supply INPUT VOLTAGE - Input voltage INTERMEDIATE BUS VOLTAGE - Bus voltage +3.3V - Output voltage / 0 to 20A +5V- Output voltage / 0 to 20A +12V- Output voltage / 0 to 5A -12V_AUX - Output voltage / 0 to 0.5A +3.3V_AUX - Output voltage / 0 to 4A I^2C ADDRESS SELECT - used to set the Control Electronics I^2C address
Outputs	+3.3V_ENABLE - 0 to 5VDC signal +5V_ENABLE - 0 to 5VDC signal +12V_ENABLE - 0 to 5VDC signal +3.3V_AUX_ENABLE - 0 to 5VDC signal -12V_AUX_ENABLE - 0 to 5VDC signal INPUT_LOW - Open-collector TTL signal /SYSRESET - Open-collector TTL signal /FAIL - Open-collector TTL signal
Communication Interface	I^2C compatible Serial Interface.

Inputs

The Control Electronics monitors the power supply temperature. The /ENABLE and the /INHIBIT are Transistor-Transistor Logic (TTL) control inputs that determine which outputs are enabled by the Control *E*lectronics; refer to table II for information on the output voltage control. The power supply input voltage, intermediate bus voltage, +3.3V, +5V, +12V, +3.3V_AUX, and -12V_AUX output voltages go into the Control Electronics for data analysis.

Outputs

The control outputs are +3.3V_ENABLE, =5V_ENABLE, +12_ENABLE, +3.3V_AUX, -12V_AUX. These signals enable and disable the power supply output voltages. The status signals indicate the power supply status. The /INPUT_LOW signal indicates that the power supply input voltage is below the threshold voltage. The /SYSRESET signal indicates that the +5V output voltage is within ± 1% of the nominal voltage. The /FAIL signal indicates that all output voltages are within ±10% of their nominal voltage.

Communication interface

The Control Electronics uses the I^2C interface and operates in slave mode only. The I^2C has a 7-bit hardware address also known as the slave address. The slave address is configured by the I^2C ADDRESS SELECT signals and the default slave address is 0x27.

Level 1 Circuit Diagram

A myriad of microcontrollers, I^2C compatible analog-to-digital converters, and temperature sensors are available for the Control Electronics project. Rantec's Director of Test-Engineering and Software Development selected all the components for this project.

Figure 4 shows the level 1 circuit diagram. The Parts Lists is found in Appendix C and the PB board layout is found in Appendix E. The Control Electronics project is an embedded system that includes one ATmega164P/324P microcontroller, seven I^2C-compatible analog-to-digital converters (ADCs), one analog temperature sensor, JTAG interface, and an LED. The microcontroller performs the following functions:

Voltage Monitor

The Controls Electronics configures the microcontroller's PD0 and PD1 pins for Two-Wire Interface communication. PD0 is the SCL signal and PD1 is the SDA signal. Pull up resistors are necessary for the Two-Wire Interface to operate correctly. The maximum value of the pull-resistors is a function of the bus capacitance and the SDA/SCL maximum rise time. The following formula determines the maximum value for the pull-up resistors [4].

Given parameters [5];

Maximum rise time (from datasheet)	$t_r := 1000 \times 10^{-9} s$	
Input pin capacitance (from datasheet)	$C_{in} := 30pF$	
Number of devices connected to the Control Electronics I2C bus	$N_{dev} := 7$	
Bus capacitance	$C_b := C_{in} \cdot N_{dev}$	$C_b = 210pF$
Maximum pull-up resistor value	$R_{P\_max} := \dfrac{t_r}{0.8473 C_b}$	$R_{P\_max} = 5.62k\Omega$

For safety margin, a 2kΩ pull-up resistor values were chosen for the internal Two-Wire Interface.

The microcontroller uses this internal I^2C interface to communicate with seven I^2C single-channel analog-to-digital converters (ADC121C021CIMM) to retrieve power supply voltage.
The I^2C analog-to-digital converters continuously monitor the power supply input and output voltages. The measured voltage is fed to the microcontroller. The voltage is formatted to a digital value and stored in the TX_Reg_Data array, see tables 7-10 for more information on the TX_Reg_Data array.
The input voltage to the analog-to-digital converters cannot exceed the reference voltage; therefore, resistor dividers are used to scale down the power supply voltage. Figure3 shows the recommended resistor values to scale down the power supply voltage.

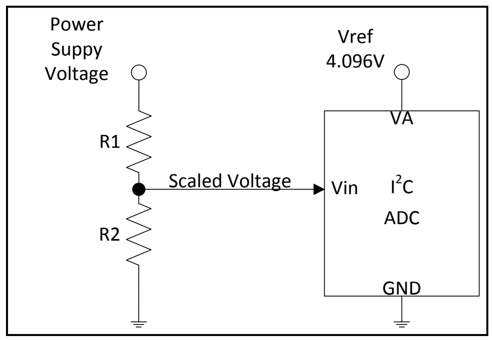


Figure 3: Resistor Divider for ADC Input

TABLE IV: RECOMMENDED RESISTOR VALUES TO SCALE DOWN THE POWER SUPPLY VOLTAGES

DC Voltage	+3.3V	+5V	12V	+3.3_AUX	-12V_AUX	DC Bus	28V
R1 in Ω	1.0E+3	8.91E+03	20.0E+3	1.0E+3	20.0E+3	30.1E+3	150.3E+3
R2 in Ω	10.0E+3	9.8E+3	7.87E+03	10.0E+3	7.87E+03	3.7E+3	14.8E+3

Since all seven I^2C analog-to-digital converters share the same Two-Wire interface bus, each device has a unique I^2C address. Table V describes the analog-to-digital converter I^2C address.

TABLE V: I^2C ANALOG-TO-DIGITAL CONVERTER ADDRESS

I^2C Device	+3.3V	+5V	12V	+3.3_AUX	-12V_AUX	DC Bus	28V
I^2C Hex Address	0x50	0x51	0x52	0x54	0x55	0x56	0x58

Overvoltage Protection

The I^2C analog-to-digital converters contain an Alert pin that is asserted when the power supply output voltage exceeds the programmed threshold [4]. The Alert pins are routed to port B of the microcontroller, see Figure 3. The microcontroller immediately disables the output that has the overvoltage condition. The output is re-enabled by sending the command 0x0A to the controller via the external I^2C bus or by recycling input power.

Control Inputs

The /ENABLE and /INHIBIT are control inputs that are fed PB5 and PB6 of the microcontroller. These signals indicate the microcontroller which power supply outputs to enable during initial power-up; refer to table II to determine which states enable the power supply output voltages.

Control Outputs

The /SYSRESET, /FAIL, and INPU_LOW are output signals from the microcontroller. PA5 pin of the microcontroller indicates the /SYSRESET state. The microcontroller asserts the /SYSRESET when the +5V output is within 1% of the nominal voltage. PA6 pin of the microcontroller indicates the /FAIL state. The microcontroller asserts the /FAIL when all output voltages are within 10% of their nominal voltage. PC6 pin of the microcontroller indicates the /INPUT_LOW state. The microcontroller asserts the /INPUT_LOW signal when the power supply input voltage falls below 12.5VDC.

Temperature Monitor

The analog temperature sensor (LM94021) measures the temperature of the power supply. The temperature sensor output voltage is inversely proportional to the measured power supply temperature [6]. The sensor voltage is fed to the analog-to-digital converter, port 7, of the microcontroller and is converted to a digital value and stored in the Temp_Data array [5].

Overtemperature

The microcontroller disables all outputs if the power supply temperature exceeds 85°C

I^2C Communication Interface

The Control Electronics uses the microcontroller's built-in Two-Wire interface that is compatible with Philips I^2C interface for external serial communication [3]. The Control Electronics monitors and stores power supply data in the TX_Reg_Data array. An external I^2C device can access this data by sending the microcontrollers I^2C address followed by a read command. The microcontroller's I^2C address is set by the I^2C_GA0 and I^2C_GA1 input signals. The default address is 0x27. Table VI describes how to set the microcontroller's I^2C address.

TABLE VI: MICROCONTROLLER'S I2C ADDRESS SELECT

I^2C_GA1	**I^2C_GA0**	**I^2C Hex Address**
LOW	LOW	0x30
LOW	HIGH	0X29
HIGH	LOW	0X28
HIGH	HIGH	0X27

JTAG Interface

The Control Electronic uses the JTAG interface to program the microcontroller and perform on-chip real time debugging.

Status LED

The Control Electronics toggles an LED every 350ms to indicate that the microcontroller is cycling thought the program. The LED staying On or Off all the time indicates that the microcontroller got stuck in a while loop.

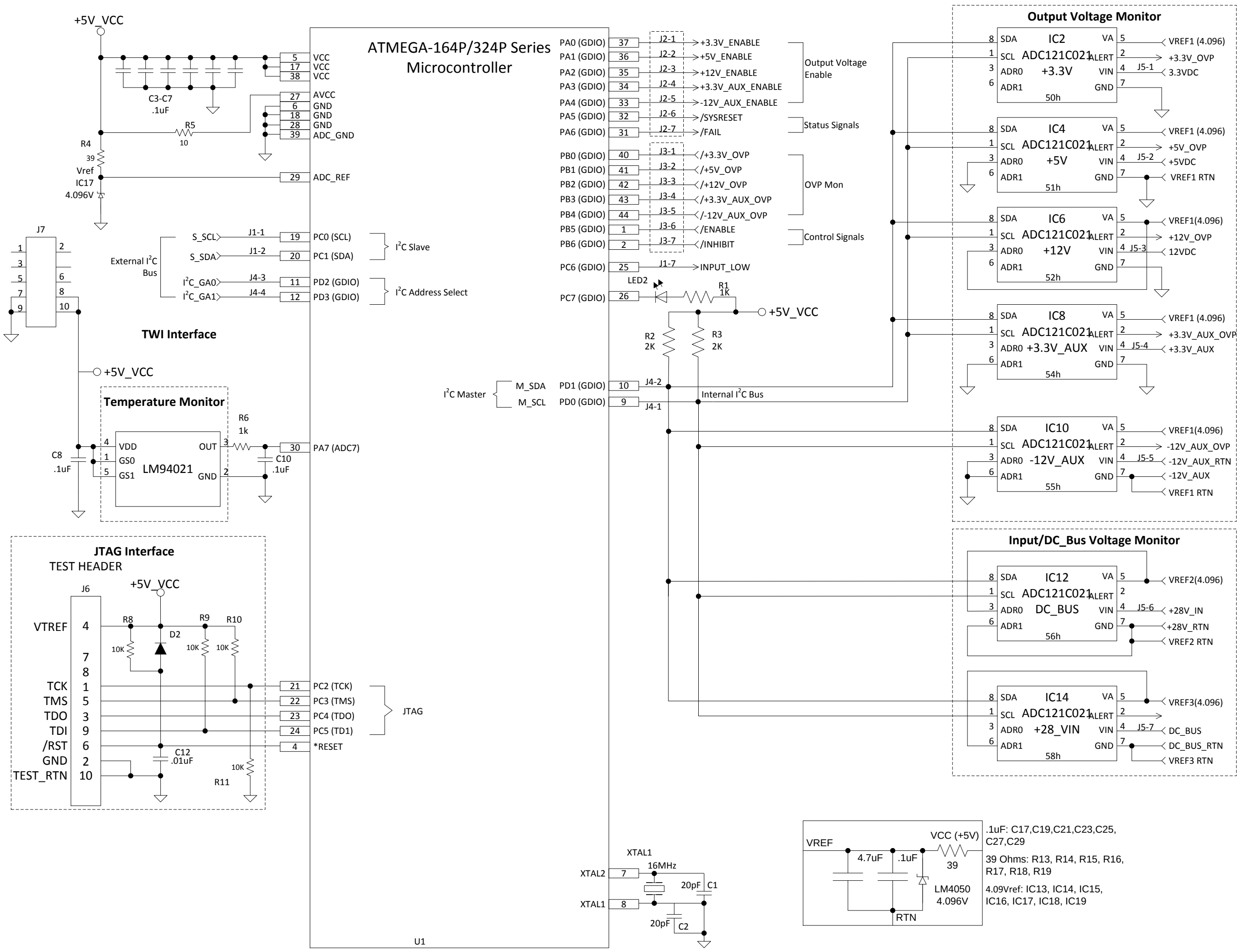


Figure 4: Level 1 Circuit Diagram

IV. Firmware Design

This project used CodeVisionAVR C compiler to develop the firmware, per Rantec's Director of Test Engineering and Software Development recommendation.

Serial communication protocol

The Control Electronics project uses the Phillips I^2C serial communication protocol to communicate with an external I^2C master device, via the microcontroller's built-in I^2C interface. To access power supply data from the Control Electronics, the I^2C master device sends a read command to the ATMega microcontroller and the microcontroller returns up to 22 bytes of data. It's the I^2C master's responsibility to indicate the number of bytes to expect from the microcontroller in one read. For more information on the I^2C communication protocol, refer to Phillips UM10204 I^2C-bus specification and user manual.

Tables VII through X describes the power supply data stored in the microcontroller's transmit registers.

TABLE VII: MICROCONTROLLER'S TRANSMIT REGISTER ASSIGNMENT

Byte Number	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Register Description	VIN		DC_BUS		-12V_AUX		+3.3V_AUX		+12V		+5V		+3.3V	

TABLE VIII: MICROCONTROLLER'S TRANSMIT REGISTER ASSIGNMENT (CONTINUED)

Byte Number	**21**	**20**	**19**	**18**	**17**	**16**	**15**	**14**
Register Description	/AC FAIL	/SYSRESET	/Input_Low	OVP Status	$OutP_{status}$	OVT	UUT Temp	

TABLE IX: MICROCONTROLLER TRANSMIT REGISTER DESCRIPTION

Byte	**Name**	**Description**
1-0	+3.3V	These register contains the power supply +3.3V output voltage. Byte 1 is the MSB and byte 0 is the LSB. Only Bits 11-0 contain voltage data. Bits 15-12 are always set to 0.
3-2	+5V	This register contains the power supply +5V output voltage. Byte 3 is the MSB and byte 2 is the LSB. Only Bits 11-0 contain voltage data. Bits 15-12 are always set to 0.
5-4	+12V	This register contains the power supply +12V output voltage. Byte 5 is the MSB and byte 4 is the LSB. Only Bits 11-0 contain voltage data. Bits 15-12 are always set to 0.
7-6	+3.3V_AUX	This register contains the power supply +3.3V_AUX output voltage. Byte 7 is the MSB and byte 6 is the LSB. Only Bits 11-0 contain voltage data. Bits 15-12 are always set to 0.
9-8	-12V_AUX	This register contains the power supply -12V_AUX output voltage. Byte 9 is the MSB and byte 8 is the LSB. Only Bits 11-0 contain voltage data. Bits 15-12 are always set to 0.
11-10	DC_BUS	This register contains the power supply DC_BUS voltage. Byte 11 is the MSB and byte 10 is the LSB. Only Bits 11-0 contain voltage data. Bits 15-12 are always set to 0.

TABLE X: MICROCONTROLLER TRANSMIT REGISTER DESCRIPTION (CONTINUED)

Byte	Name	Description
13-12	VIN	This register contains the power supply VIN voltage. Byte 13 is the MSB and byte 12 is the LSB. Only Bits 11-0 contain voltage data. Bits 15-12 are always set to 0.
15-14	UUT Temp	This register contains the power supply temperature in °C. Byte 15 is the MSB and byte 14 is the LSB. Only Bits 11-0 contain voltage data. Bits 15-12 are always set to 0.
16	OVT	This is the overtemperature registers. A 0x0F indicates an overtemperature condition. All other values indicate a normal operating temperature.
17	$OutP_{status}$	This register indicates which of the 5 outputs are enable/disabled B0: 0 = +3.3V disabled, 1 = +3.3V enabled. B1: 0 = +5 V disabled, 1 = +5V enabled. B2: 0 = +12V disabled, 1 = +12V enabled. B3: 0 = +3.3V_AUX disabled, 1 = +3.3V_AUX enabled. B4: 0 = -12V_AUX disabled, 1 = -12V_AUX enabled.
18	OVP Status	This register indicates which of the 5 outputs has an overvoltage condition (OVP). B0: 0 = +3.3V not OVP, 1 = +3.3V OVP. B1: 0 = +5 V not OVP, 1 = +5V OVP. B2: 0 = +12V not OVP, 1 = +12V OVP. B3: 0 = +3.3V_AUX not OVP, 1 = +3.3V_AUX OVP. B4: 0 = -12V_AUX not OVP, 1 = -12V_AUX OVP.
19	/INPUT_LOW	This is the /INPUT_LOW register. A 0x0D indicates that the input voltage is below 12.5VDC. All other values indicate that the voltage is above 12.5VDC.
20	/SYSRESET	This is the /SYSRESET register. A 0x0A indicates that the +5V output is within 1% of the nominal voltage during power on. All other values indicate that the voltage is below 1% of the nominal.
21	/FAIL	This is the /FAIL register. A 0x0B indicates that all output voltages are within 10% of their nominal voltage. All other values indicate that all output voltages are not within 10% of their nominal voltage.

System State Diagram

Figure 5 shows the microcontroller's System State Diagram. This diagram describes the behavior of the ATMega microcontroller which includes the main loop and four interrupt routines developed in this project [7]. The interrupt routines include: The Timer0_Interrupt, Timer2_Interrupt, TWI_Interrupt, and OVP_interrupt routine. The Timer0_Interrupt updates power supply input and output voltage data every 317mS. The Timer2_Interrupt maintains the system clock and updates temperature data every 750mS. The TWI Interrupt waits for the START signal from the external Two-Wire Interface master device to send or receive data(9). The OVP_Interrupt triggers when there is a power supply output overvoltage condition.

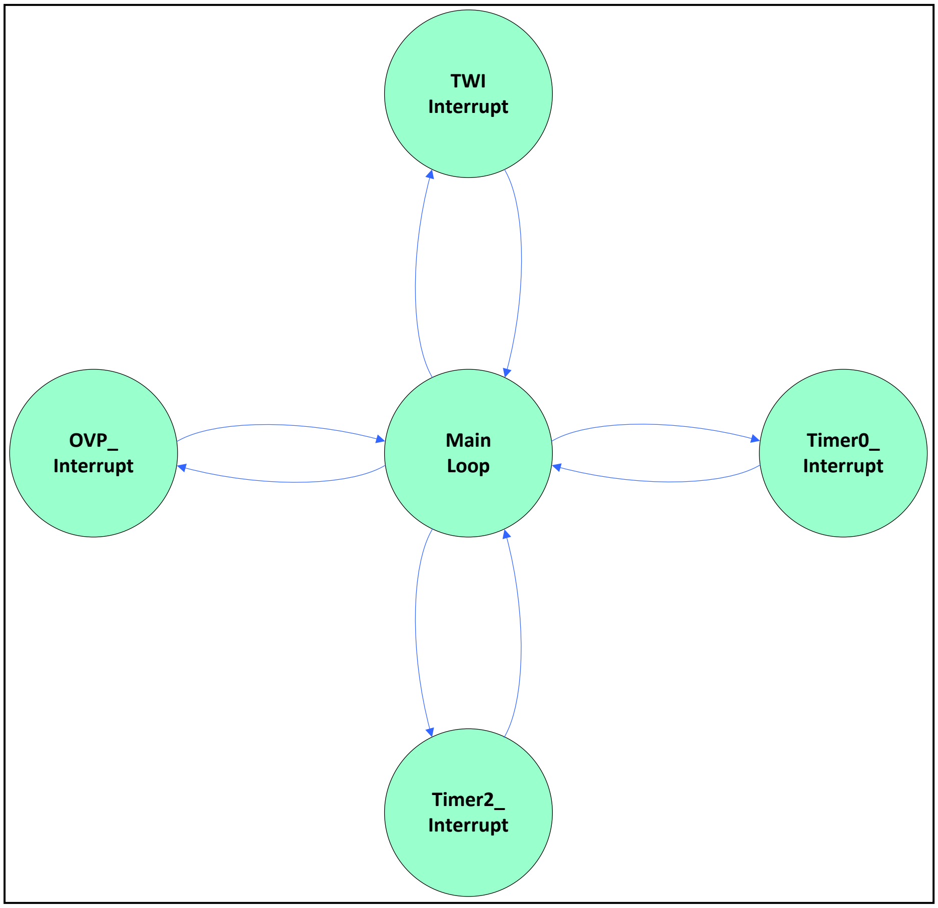


Figure 5: System State Diagram

Main Program

Figure 6 shows the main program flow chart.

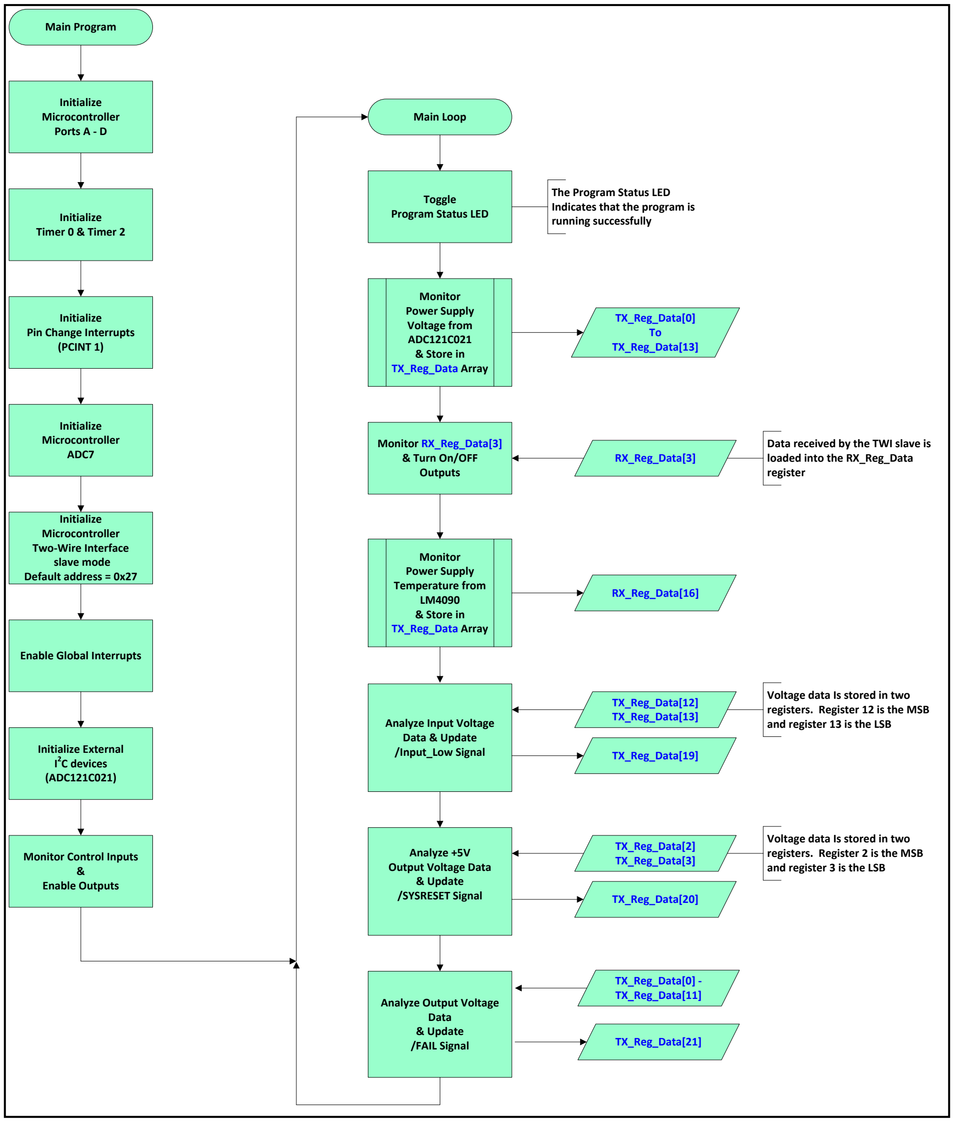


Figure 6: Main Program Flow Chart

The Main program initializes the microcontroller's ports A through D, Timer 0 & Timer 2, the Pin Change interrupts, analog-to-digital converter port 7, and the Two-wire interface. The Two-wire interface 7-bit slave address is set to 0x27. Next, the program enables the global interrupts. After that, the program initializes the I^2C 12-bit analog-to-digital converters (ADC121C021) . Afterwards, the program monitors the /ENABLE and the /INHIBIT control inputs and enables certain output voltages; refer to table II to determine which control input states enable/disable the power supply output voltages.

After the hardware and software initialization, the program enters the main loop. The following steps describe functions that are executed main loop:

First, the program toggles the Program Status LED. The LED blinks every 250mS and provides a visual aid to the user indicating that the program is cycling through the main loop successfully. The LED not blinking that reveals that the program got stuck in a loop and the user must recycle the input power to reset the program.

Second, the program runs the Meas_Volt function to retrieve 14 bytes of voltage data from the seven I^2C 12-bit analog-to-digital converters and stores the data in the TX_Reg_Data array; refer to table VII for more information on the stored voltage data. See figure 6 for more information on the Meas_Volt function.

Third, the program monitors the RX_Reg_Data(3). The 0x0A value in this register enables all power supply output voltages and a value of 0x0C disables all power supply output voltages.

Fourth, the program executes the Meas_Temperature to retrieve 2 bytes of voltage data. The voltage data is compared to a value in look-up table and translated to a temperature. The formatted temperature value is stored in the TX_Reg_Data array; refer to table VIII for more information on the stored temperature data.

Fifth, the program analyzes the input voltage data stored in TX_Reg_Data(12) & TX_Reg_Data(13) and updates the TX_Reg_Data(19) register. If the input voltage is below 12.5V, the program updates the TX_Reg_Data(19) register with the value 0x0D. Otherwise, the program sets the TX_Reg_Data(19) register to 0x00.

Sixth, the program analyzes the +5V output voltage data stored in TX_Reg_Data(2) & TX_Reg_Data(3)and updates the TX_Reg_Data(20) register. If the +5V output voltage is within 4.95V and 5.05V, the program updates the TX_Reg_Data(20) register with the value 0x0A. Otherwise, the program sets the TX_Reg_Data[20] register to 0x00.

Last, the program analyzes the output voltage data stored in TX_Reg_Data(0) through TX_Reg_Data(11) and updates the TX_Reg_Data(21) register. If all output voltages are within 10% of their nominal voltage, the program updates the TX_Reg_Data(21) register with the value 0x0B. Otherwise, the program sets the TX_Reg_Data[21] register to 0x00.

After this step, the program returns to the beginning of the main loop.

C-Code functions

Meas_Volt Function

Figure 7 shows the Meas_Volt function flow chart. This function retrieves voltage from the I^2C analog-to-digital converters. The microcontroller initiates a conversation with the ADC121C021 and waits for the device to acknowledge. If the device acknowledges and the data is valid, the program multipliers the voltage data from the I^2C device by a scaling factor [8]. Figure 6 describes the scaling factors for the power supply output voltages. The formatted voltage data is stored in the TX_Reg_Data [0] through TX_Reg_Data [11] registers.

Index [i]	Signal Name	ADC Address Vout_ADC_Addr[i]	Scale Factor VinScaleFact[i]
0	+3.3V	0x50	0.10
1	+5V	0x51	0.1913846
2	+12V	0x52	0.10
3	+3.3V_AUX	0x54	0.10
4	-12V_AUX	0x55	0.10
5	DC_Bus	0x56	1.1134861
6	Vin	0x58	1.1134861

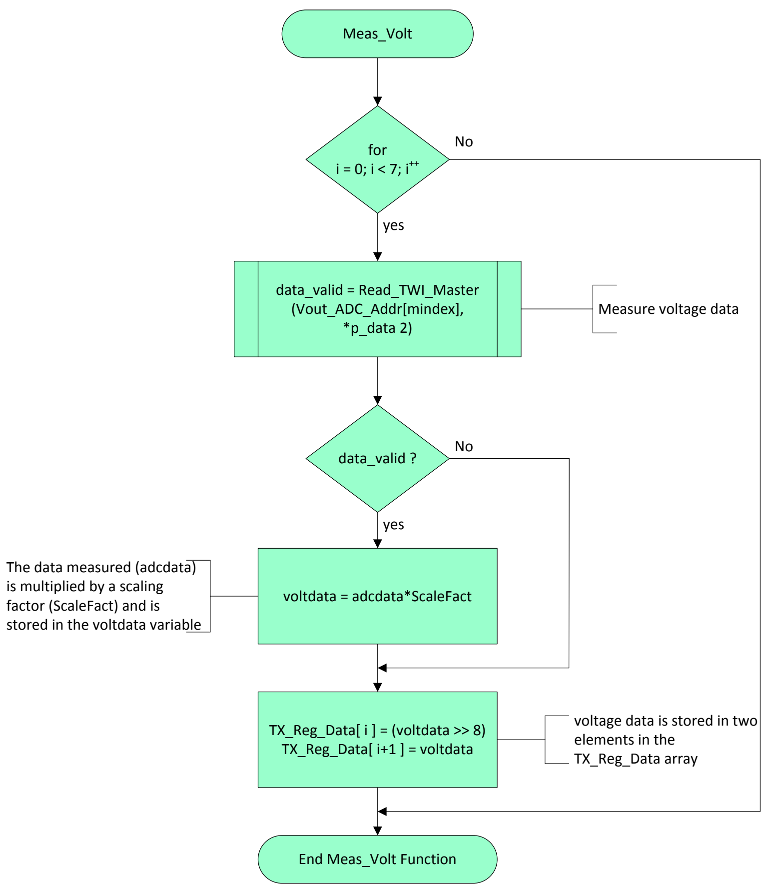


Figure 7: Meas_Volt Function Flow Chart

Meas_Temperature Function

Figure 8 shows the Meas_Temperature function flow chart. The microcontroller starts the conversation with the LM94021 temperature sensor through its analog-to-digital port 7. After the conversion is complete and if there is no timeout, the voltage data is compared to a value in a lookup table and converted to a temperature value. The formatted temperature data is stored in TX_Reg_Data(14) & TX_Reg_Data(15) registers.

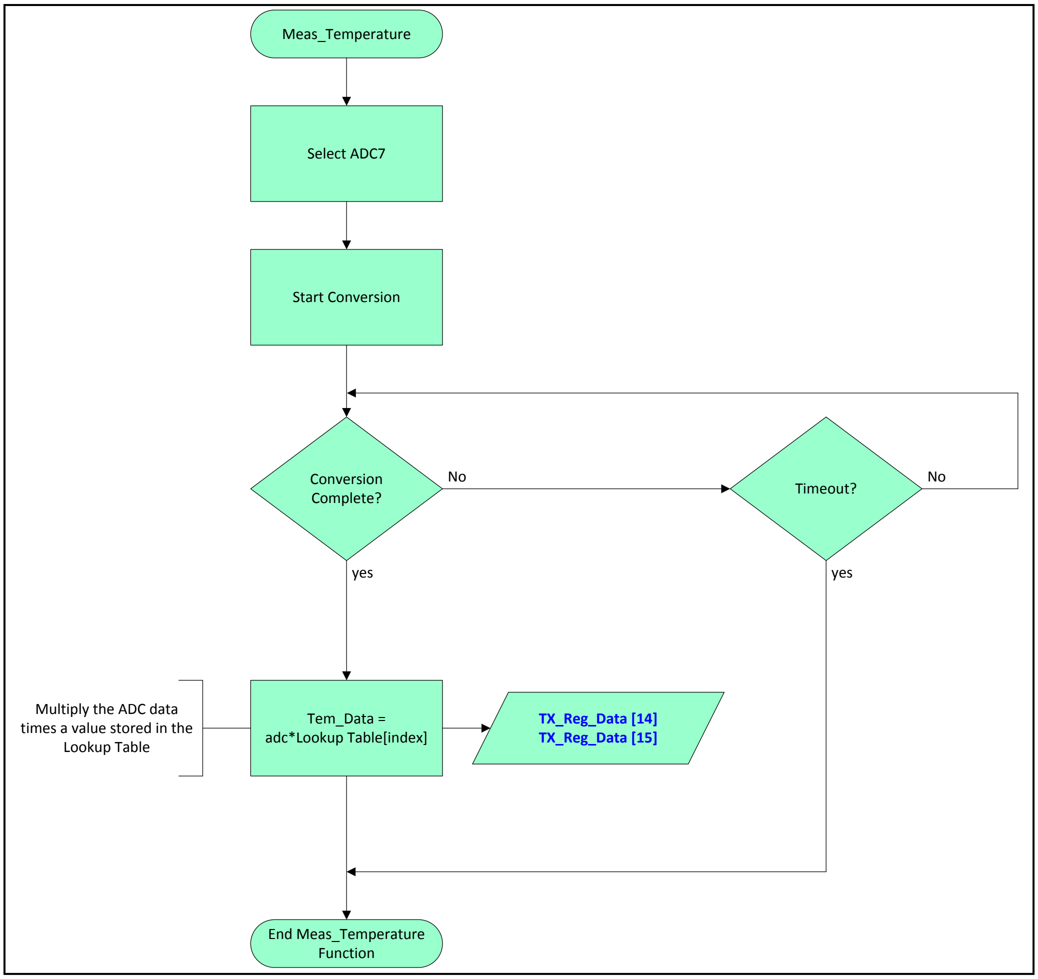


Figure 8: Meas_Temperature Function Flow Chart

TWI_Master Function

Figure 9 shows the TWI_Master function flow chart. This function implements the Two-wire interface communication protocol by the bit banging technique and is compatible with the I^2C protocol. The bit banging method was chosen due to its low cost implementation that utilizes software instead of hardware to handle the timing and synchronization requirements for serial communication [10], [11], [12].

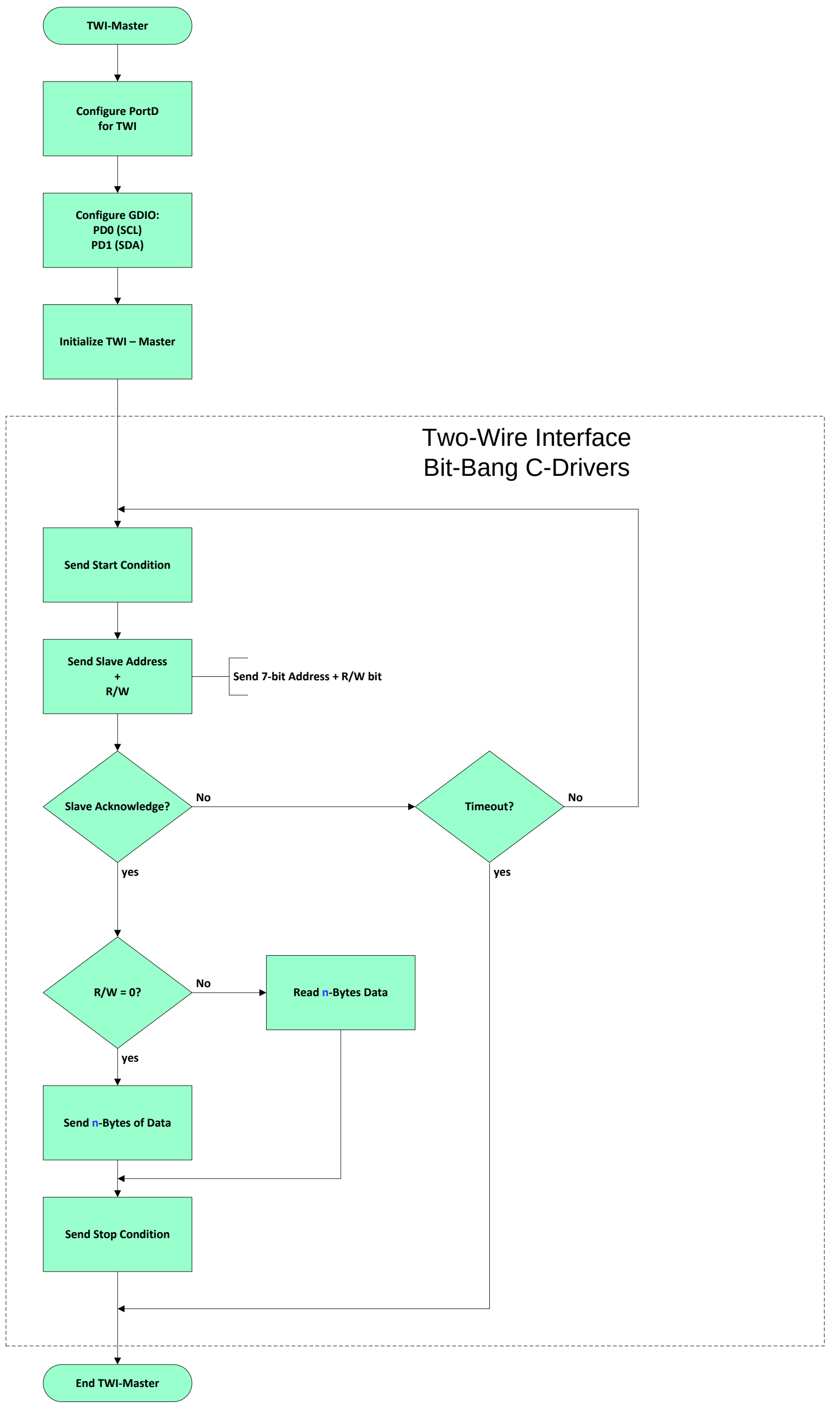


Figure 9: Two-Wire Master Function Flow Chart

The function configures the microcontroller's GPIO pins, PD0 and PD1, for Two-Wire serial communication, which is compatible with the I^2C interface. PD0 is the SCL signal and PD1 is the SDA. The master device (the microcontroller) sends the start condition followed the slave 7-bit address and the Read or Write command to the I^2C device. The microcontroller waits for the slave device to acknowledge. If the slave device acknowledges, then the microcontroller sends or receives n-bytes of data. If the slave device does not acknowledge, a timeout occurs and the program exits this function. The TWI_Master function supports up to 100 KHz clock frequency. Almost any two microcontroller GPIO pins can be configured for Two-Wire interface communication.
For more information on the I^2C communication protocol, refer to Phillips UM10204 I^2C-bus specification and user manual.

V. Graphical User Interface

A Graphical user interface is required to retrieve the Control Electronics data. National Instruments LabVIEW development environment was chosen to develop the Graphical User Interface due to its flexibility to integrate hardware and software. Figure 10 shows the Control Electronics Graphical User Interface.

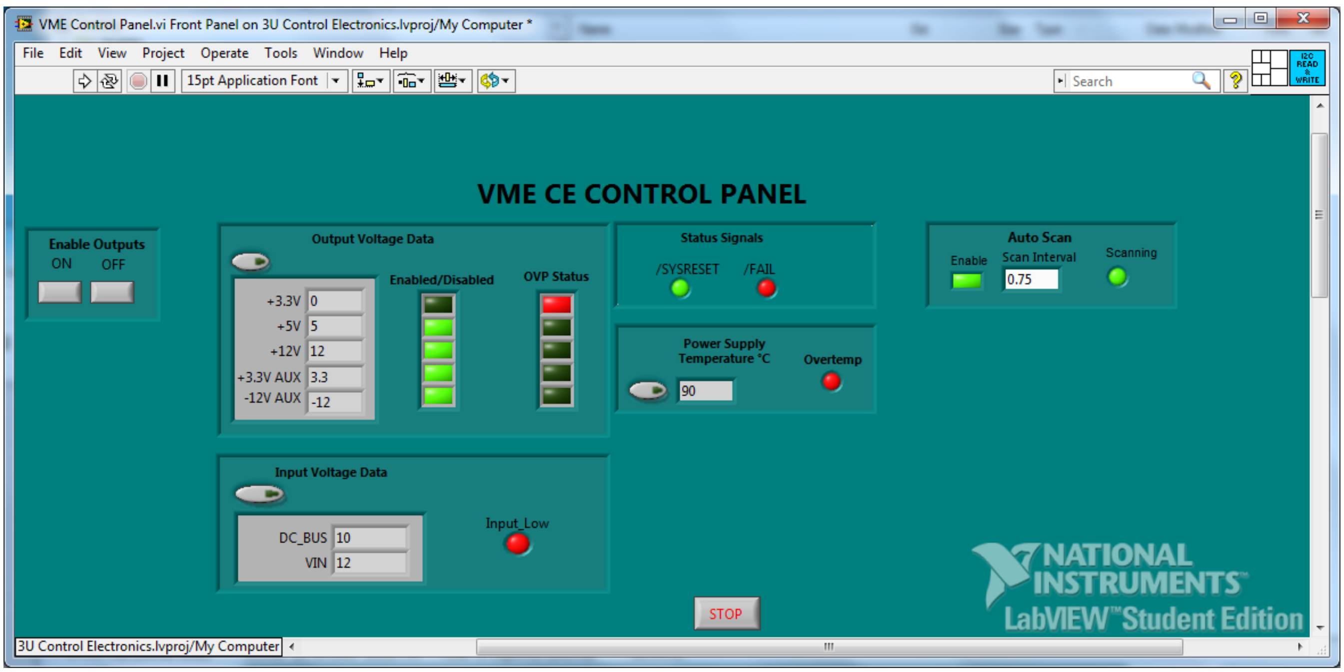


Figure10: Control Electronics Front Panel

Graphical User Interface Features

Auto Scan
The user clicks the enable control to automatically retrieve data from the microcontroller. The default scan interval is 750 milli-seconds and can be updated by the user.

Enable Outputs
The user commands the Control Electronics to enable and disable the power supply outputs by clicking the ON or OFF buttons.

Output Voltage Data
Power supply output voltage data is displayed as well as the enable/disable and overvoltage protection status.

Input Voltage Data
Power supply input voltage is displayed as well as the INPUT_LOW status.

Status Signals
The /SYSRESET signal indicates when the power supply +5V output is within ±1% of its nominal voltage. The /FAIL signals indicates when all power supply outputs are within ±10% of their nominal voltage.

Power Supply Temperature
The power supply temperature is displayed as well as the overtemperature status.

Figure 11 shows the Control Electronics Block Diagram. This diagram shows the Graphical User Interface source code. The code is executed from left to right. First, the variables are initialized. Then the program enters the main loop. In the main loop, the program waits for user input. When the user selects the auto scan feature, the program requests data from the Control Electronics at a user defined scanned interval.

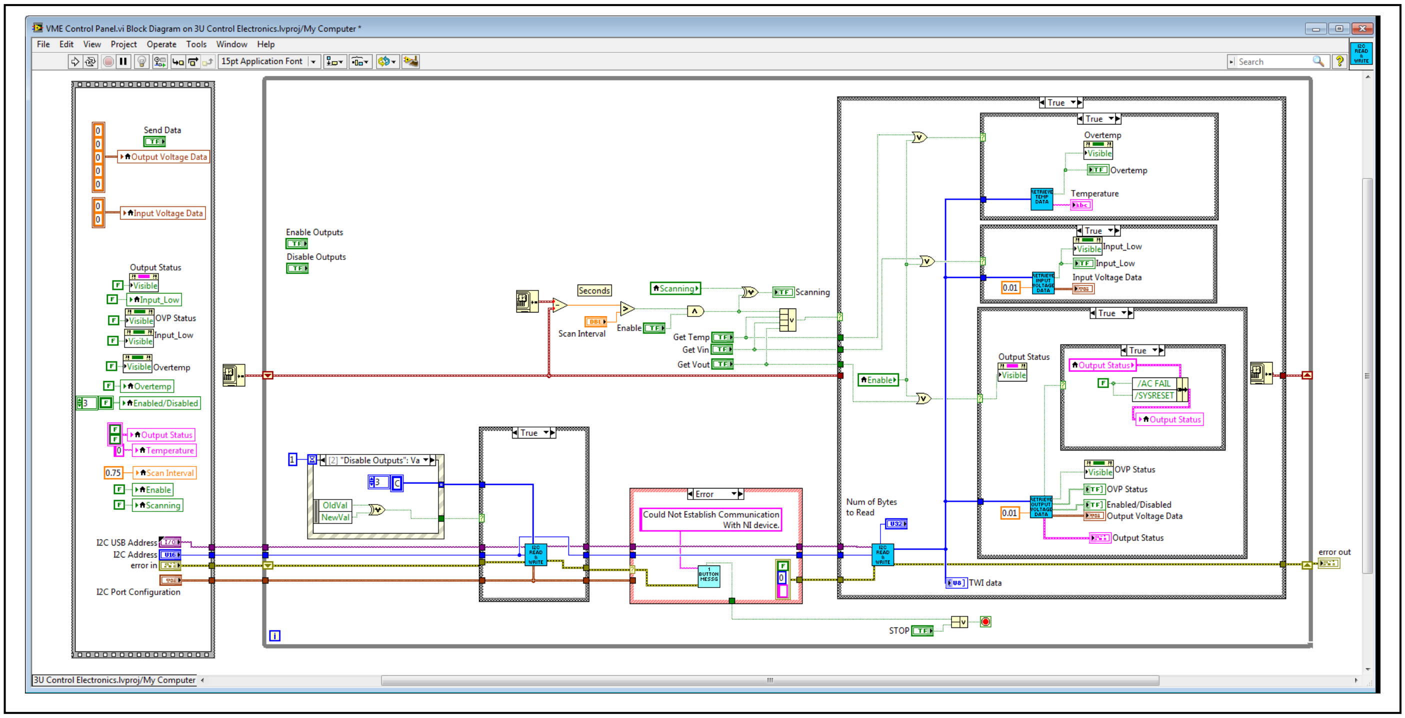


Figure11: Control Electronics Block Diagram

VI. Test Plans

The following was used to complete this senior project.

Software

- LabVIEW 2010-2012 student version development environment
- Atmel Studio 5.1 & 6.1 C-Compiler
- CodeVision AVR V3.05, V3.06, and V3.07 C-Compiler
- MS Office Visio student version

Hardware

- Atmel STK600 development board
- JTAG ICE II debugger
- NI-8451 USB-I^2C interface

VII. Development and Construction

Printed Circuit Board construction

Figure 12 shows seven ADC121C021 I^2C analog-to-digital converters and three I^2C isolators soldered to a prototype. A 10-pin header was soldered to clip test leads and allow easy debugging.
Figure 13 shows the LM94021 temperature sensor soldered to PC board. These two boards were used to develop and the initial firmware.

Prototype I

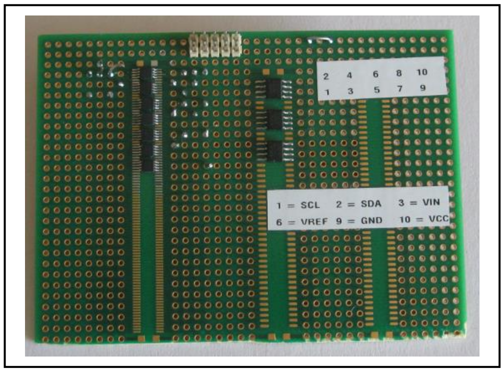


Figure 12: Board with 7 ADCs and I2C Isolators

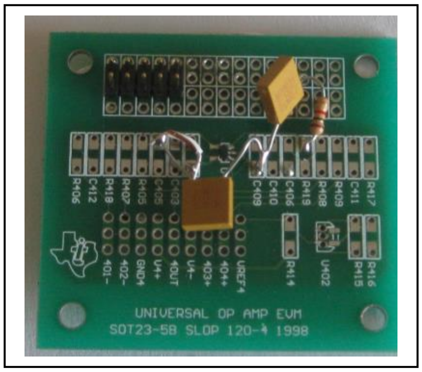


Figure 13: Board with Temperature Sensor

Prototype II

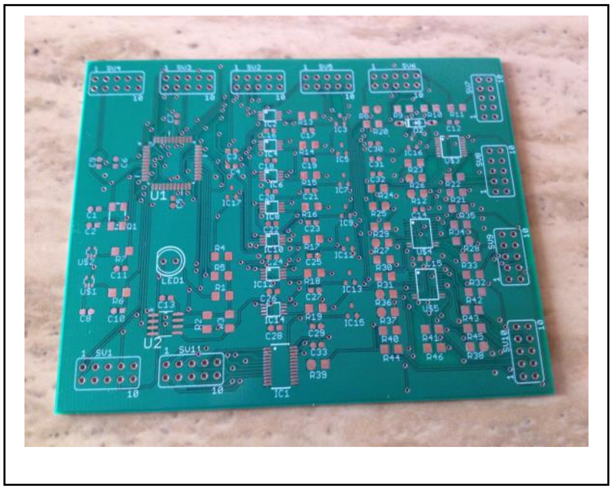


Figure 14: Control Electronics Bare Board

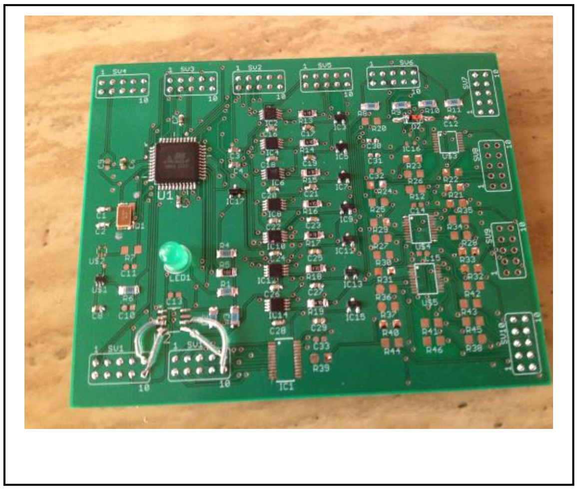


Figure 15: Partially Built board

VIII. Integration and Test Results

Output Voltage Monitor and Resolution

Figure 16 shows the Internal Two-Wire interface communication. The microcontroller uses the Two-Wire interface bit bang driver to monitor the power supply input and output voltages every 12.8mS. Figure 17 shows the measured output voltages. Figure 18 shows the measured DC_Bus and Input voltage. The measured voltage resolution is 10 milli-volts and is well within the required 100 milli-volts.

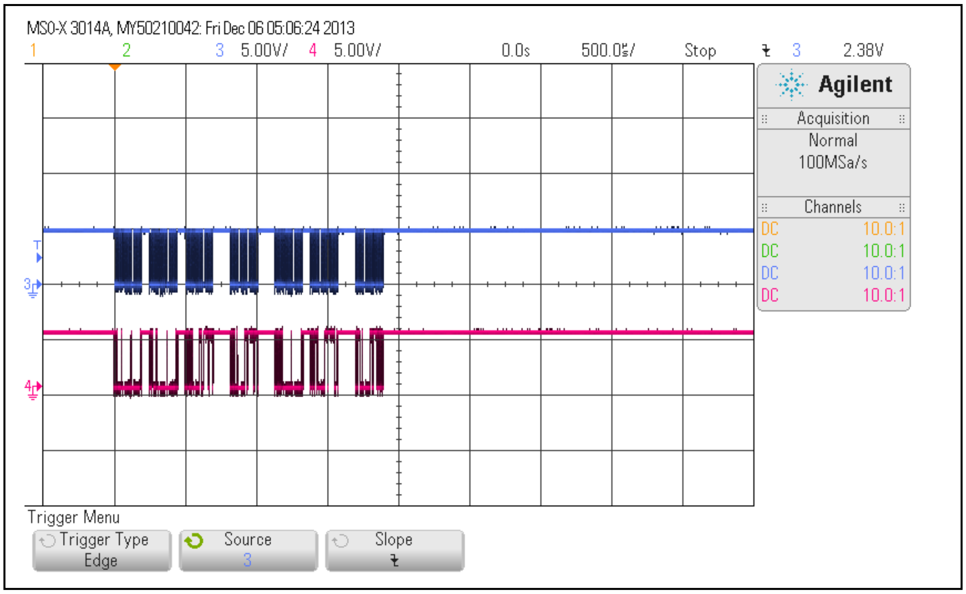


Figure 16: Internal Two-Wire Interface Communication

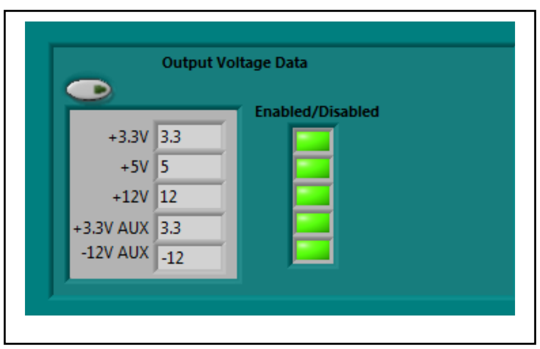


Figure 17: Measured Output Voltages

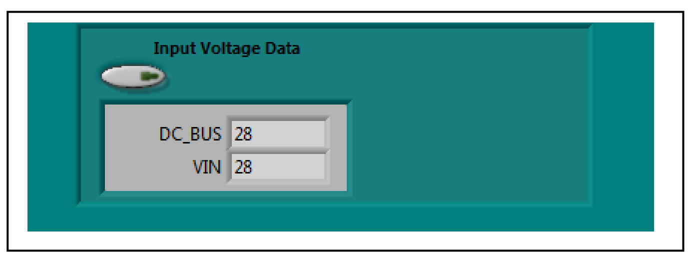


Figure 18: Measured DC_Bus & Input Voltage

Overvoltage Protection

Figure 19 shows the overvoltage condition. The Control Electronics reports an overvoltage condition if any of the power supply output voltages exceed 125% of the rated output. The microcontroller disables the power supply output that causes an overvoltage condition.

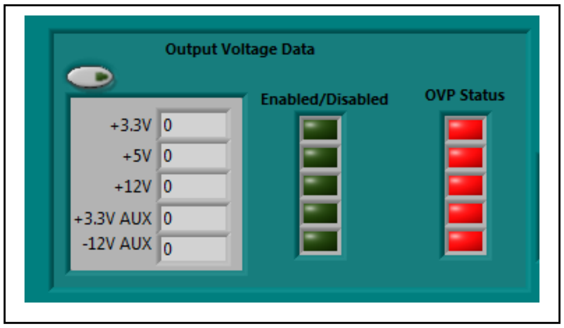


Figure 19: Overvoltage Condition

Temperature Monitor and Over Temperature

The Control Electronics monitors temperature. The temperature readback accuracy will be verified and the system level. Figure 20 shows the measured temperature. Figure 21 shows the Control Electronics reporting an over temperature condition. The control Electronics disables all of the power supply output voltages if the power supply temperature exceeds +85 ^{O}C. The operating and storage temperature will be verified at the system level.

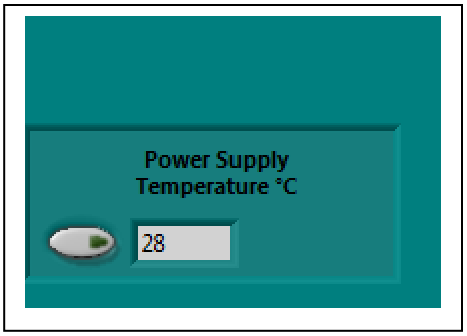


Figure 20: Measured Temperature

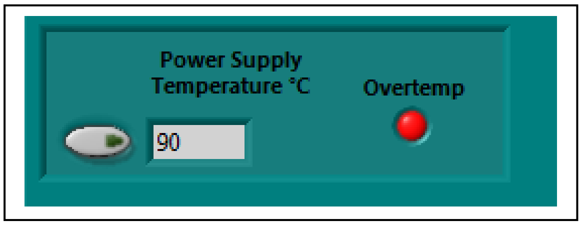


Figure 21: Over Temperature Condition

/ENABLE and /INHIBIT

Table XI describes the /ENABLE and /INHIBIT test results. The Control Electronics allows the user to control the logic states of the /ENABLE and /INHIBIT signals to turn ON or OFF the power supply output voltages.

TABLE XI: /ENABLE AND INHIBIT TEST RESULTS

Control Input		Power Supply Outputs		
/ENABLE	**/INHIBIT**	**3.3V_AUX**	**+3.3V, +5V, +12V, -12V_AUX**	**Test Results**
High	High	Off	Off	Verified
High	Low	Off	Off	Verified
Low	High	On	On	Verified
Low	Low	On	Off	Verified

/SYSRESET and /FAIL Status Signals

Figure 22 shows the +5V and /SYSRESET waveforms. The Control Electronics asserts the/SYSRESET signal 200mS after the +5V output is within ± 1% of the nominal voltage. The Control Electronics asserts the /FAIL signal when all of the power supply output voltages are within ±10% of their output voltage rating. Figure 23 shows the Control Electronics asserting the /SYSRESET and /FAIL signals.

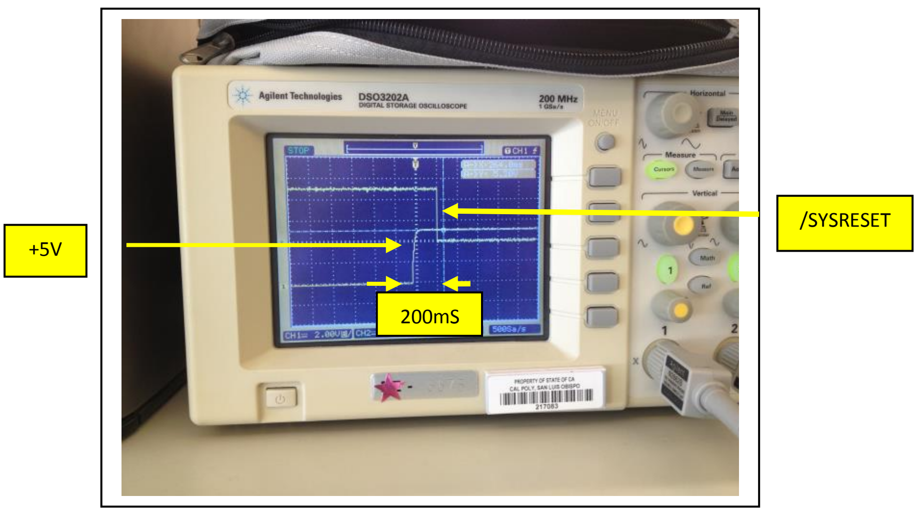


Figure 22: +5V and /SYSRESET Waveforms

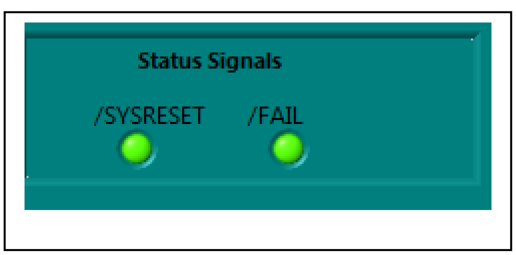


Figure 23: /SYSRESET and /FAIL Status Signals

Turn-On/Enable Sequence

Figure 24 shows the Enable turn-on sequence. The active low signals enable the power supply outputs and will be calibrated at the system level to satisfy the turn-on specification from figure 1.

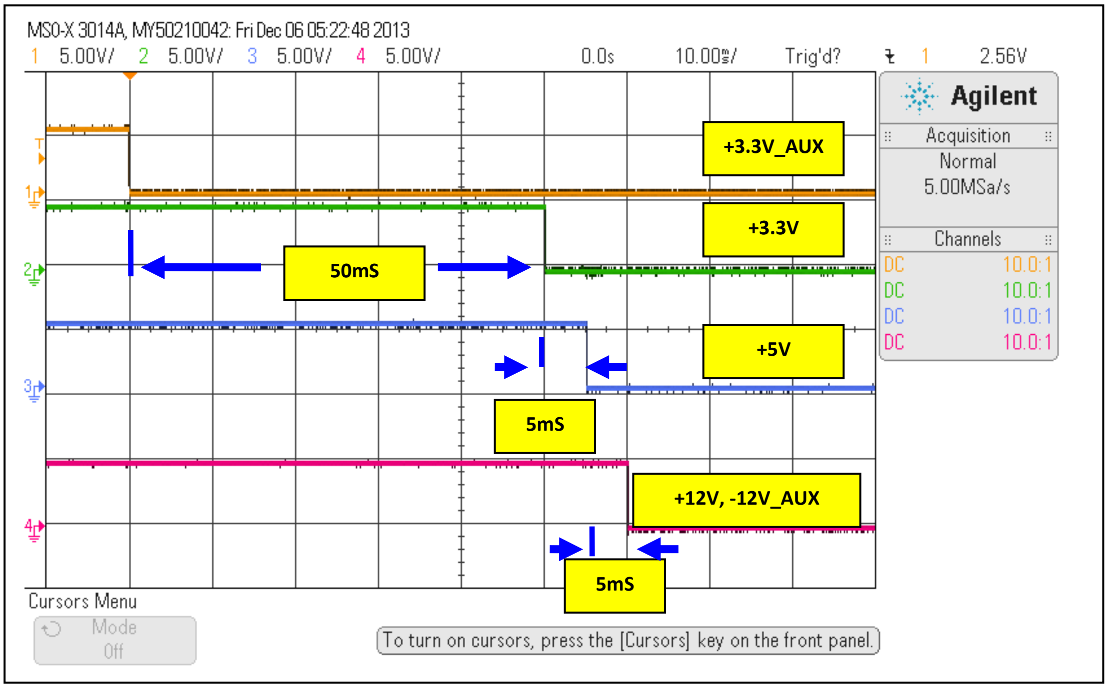


Figure24: Enable Turn-On Sequence

External Master Two-Wire Interface

A National Instruments Ni-8451 I^{2}C/SPI device emulates a master TWI that communicates with the Control Electronics.

The drivers were developed using the LabVIEW Student Edition development environment. Figure 25 shows the LabVIEW Front Panel and Block Diagram I^{2}C device driver.

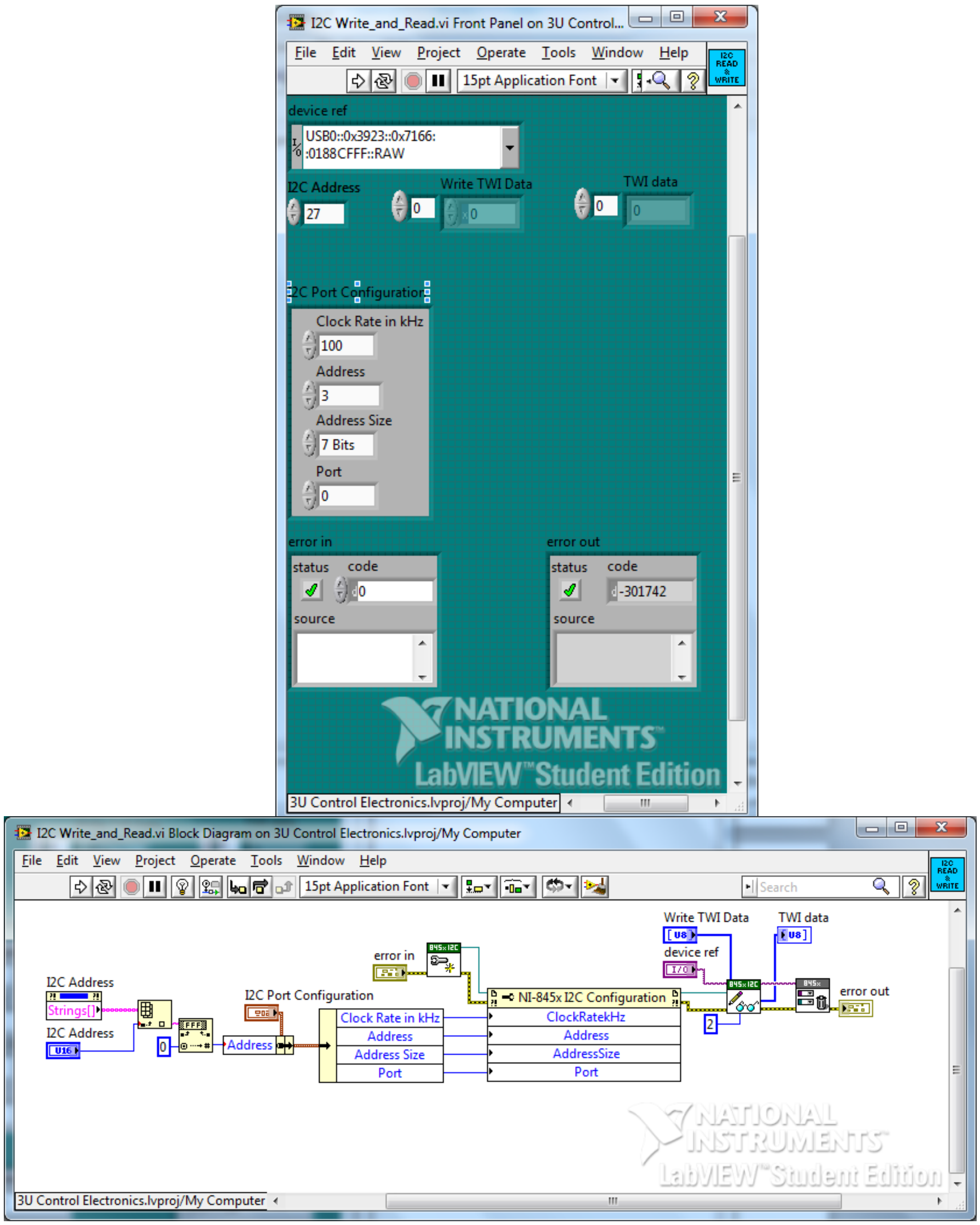


Figure 25: LabVIEW I^{2}C Device Driver

Figure 26 shows the Two-Wire Interface Waveforms (SCL and SDA). The NI-8451 I^2C/SPI device generates the SCL and SDA waveforms.

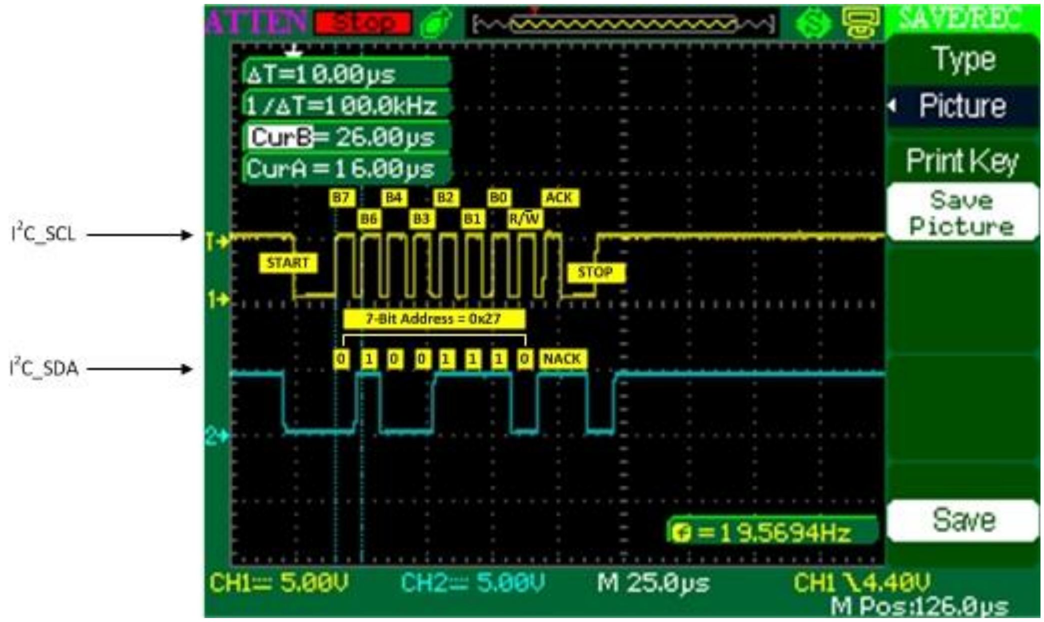


Figure 26: NI-8451 Two-Wire Interface Waveforms

This test verifies that the LabVIEW I^2C drivers command the NI-8451 device to generate a 100 kHz Clock (SCL) and send the 7-bit address with R/W. The MSB is sent first via the SDA line. The ninth cycle is where the Two-Wire interface device pulls SDA low to acknowledge that the data was received successfully. Figure 26 shows that the SDA signal is high during the ninth cycle, since the Two-Wire interface device is not connected to the Two-Wire Interface bus; therefore, a Not Acknowledge is expected.

Master and Slave Two-Wire Interface Integration

Figure 27 shows a picture of the initial Two-wire master/Slave setup. The photograph demonstrates the hardware, firmware, and software integration. The computer communicates with the microcontroller via the NI-USB to I^2C device. The scope captures the SDA and SCL waveforms.

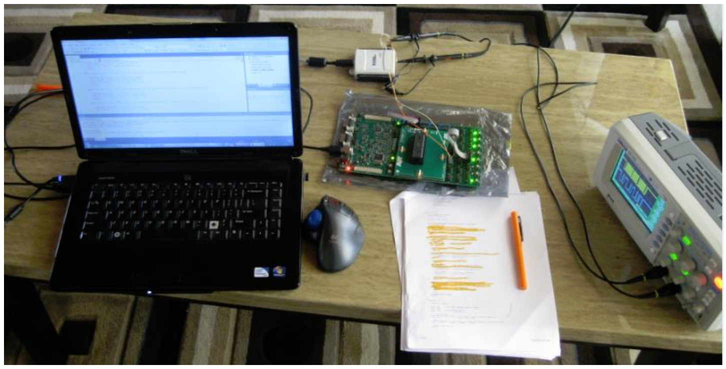


Figure 27: TWI Master-Slave Device Testing

Figure 28 shows the TWI Waveforms – Master device writing data to the microcontroller. The picture demonstrates that the microcontroller acknowledges that the data was successfully received on the ninth cycle.

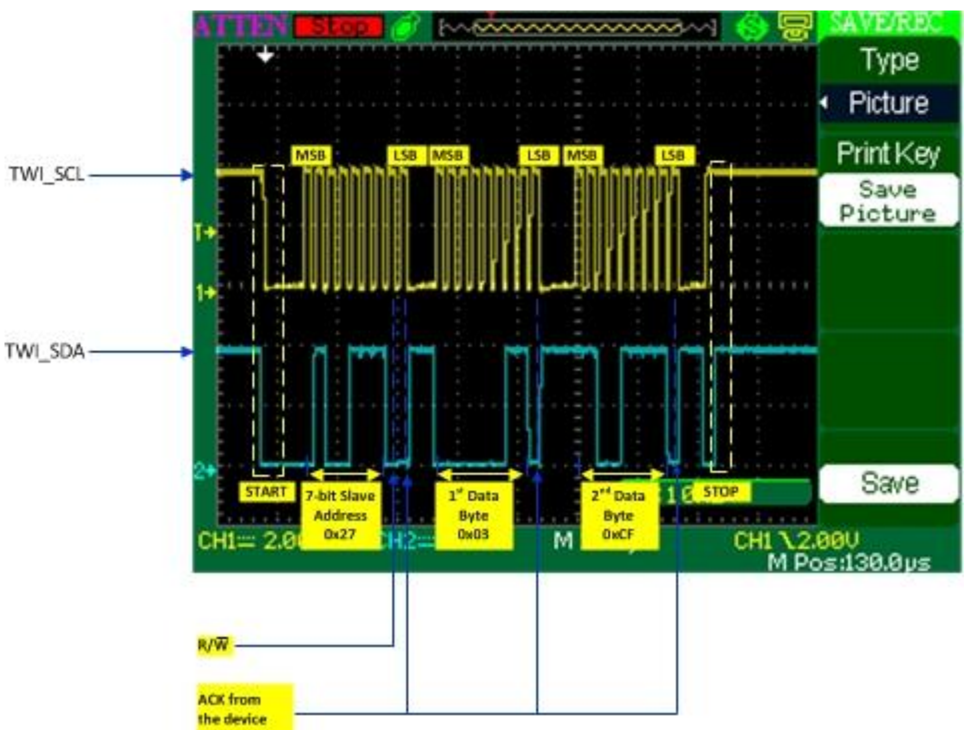


Figure 28: TWI Waveforms – Master Device Writing Data to the Microcontroller

Figure 29 shows the TWI Waveforms – Master device reading data from the microcontroller. Per the I^2C protocol, the TWI master does not acknowledge after the last data byte is received.

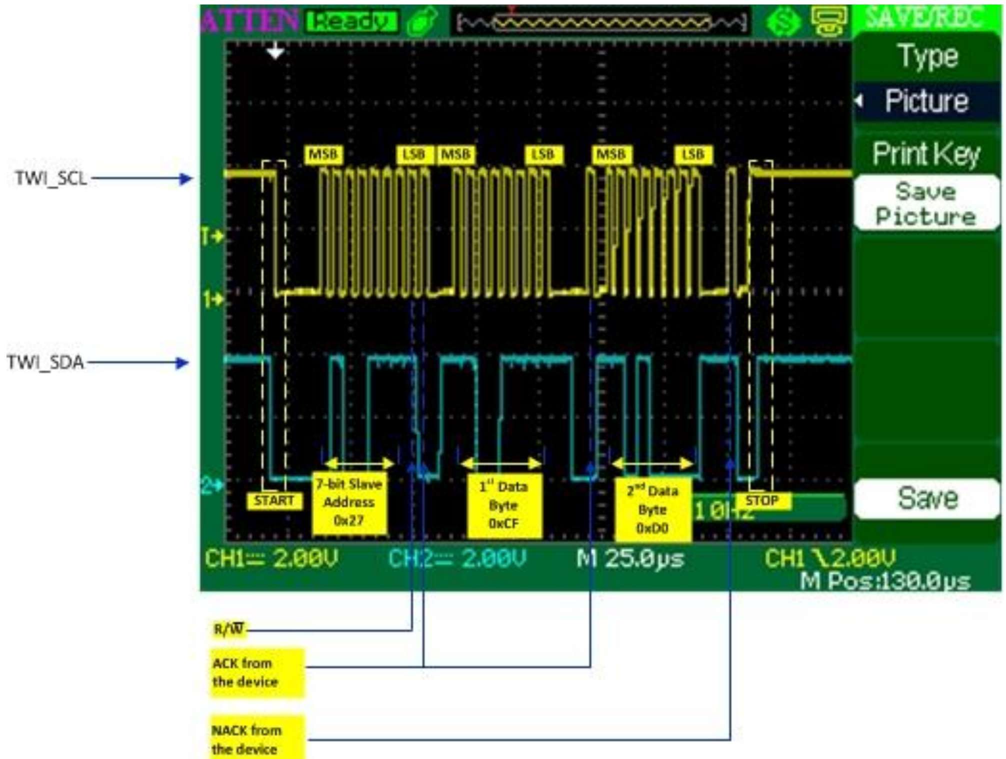


Figure 29: TWI Waveforms – Master Device Reading Data from the Microcontroller

Figure 30 demonstrates the final hardware, firmware, and software integration. Figure 31 shows a photograph of the senior project demonstration at the Cal Poly Senior Project Expo.

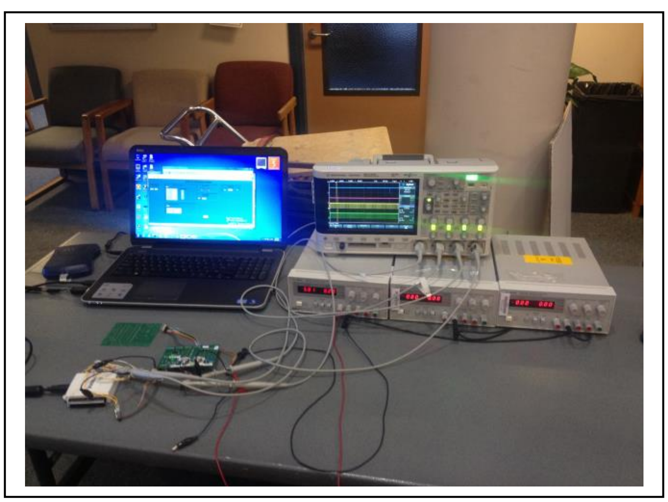


Figure 30: Final Hardware, Firmware, and Software Integration

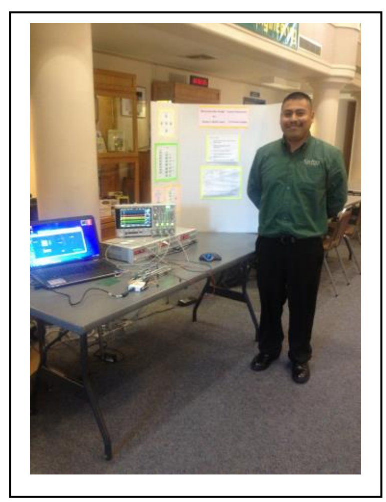


Figure 31: Senior Project Expo

IX. Problems Encountered

1. TWI Master clock

While implementing the master TWI protocol communication via the bit banging method, it was discovered that due to hardware limitations, the TWI CLOCK (SCL) of 400kz could not be achieved. The TWI clock was decreased to 100 kHz.

2. Master TWI device

Originally, the Aardvark device was selected as the master device to communicate with the ATMega164P microcontroller. However, the LabVIEW drivers could not be installed on a Dell laptop with Windows 7 64-bit operating system. After extensive literature search, it was discovered that the device manufacturer, Total Phase, only supports 32-bit operating systems.
An alternative choice had to be made. The following options were available:

A. Use existing TWI code and operate the ATmega as a master and slave TWI device
B. Borrow a USB to I^2C/SPI device from the customer
C. Ask the customer to purchase a USB to I^2C/SPI device (spend approximately $423).

The customer resolved the problem by purchasing a USB-8451 I^2C/SPI device from National Instruments.

3. Microcontroller External Interrupts

When implementing the Pin Change Interrupts on Port B of the ATMega164P microcontroller, some unknown event continuously triggered the PCINT1 routine. Extensive debugging showed that Port Bit1 (PCINT9) continuously triggered the PCINT1 routine. It turns out that when the CKOUT fuse is programmed, PBB1's alternate function is to output the system clock, regardless of PORTB settings. Somehow, the CKOUT fuse had been inadvertently programmed. The problem was fixed by disabling the CKOUT fuse.

4. TWI Bit Banging Source Code

There bit banging application note contains errors and due to limited knowledge on C-programming, it took over 8 hours to realize that it had errors.

5. TWI Slave Source Code

There was not much information available on the Two-Wire Interface source code. Many online application notes were pieced together to develop the Two-Wire Interface for this project.

6. JTAG ICE II Debugger

When performing on-chip debugging for the first time, the JTAG ICE II debugger did not function correctly. The Atmel Studio development environment failed to recognize the debugger device ID. The help feature provided by Atmel provided little information in setting up the JTAG ICE II debugger. The AVR freaks website had many threads regarding the debugger but none of offered a solution to the problem.

7. PC Board Construction

The temperature reported by the microcontroller fluctuated from 25°C to 50°C. This was due to the missing filter capacitor (C10) on the output of the temperature sensor. Figure 32 shows the schematic and photograph of the missing capacitor.

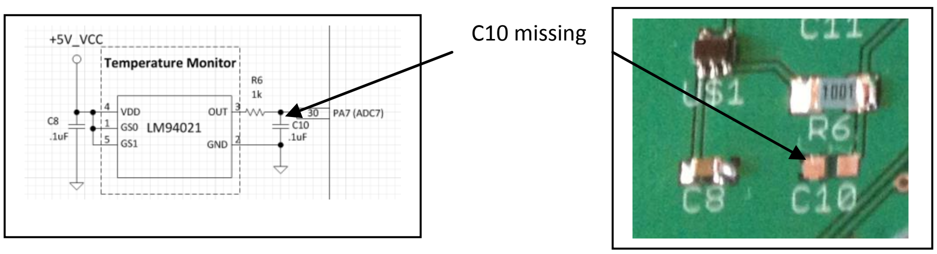


Figure32: Schematic and Photograph of the Missing Capacitor

8. /SYSRESET Testing

The /SYSRESET timing requirement did not work the first time. This was due to missing code to assert this signal. Code was added to assert the /SYSRESET signal when the +5V is within ± 1%. The initial test revealed that the /SYSRESET signal stayed logic high all the time. This was due to not initializing and resetting signal when required.

After updating the source code to initialize and reset the signal when required, the /SYSRESET signal was asserted 400mS after the trigger source instead of 200mS. This was due to the program's slow sample rate of 350mS that monitor the +5V output voltage. To fix this problem, the Nyquist-Shannon sampling theorem was implemented and the sample rate was changed to 12.8mS [].

The delay between the +5V being within ± 1% was updated to 122mS and the /SYSRESET assertion time met the requirement. Figure 22, shows the /SYSRESET signal meeting the assertion time

9. /FAIL Testing

The /FAIL status signal did not function during the initial testing. This was due to missing code to assert the signal. Code was added to assert the /FAIL signal when all of the power supply outputs are within ±10 of their rated output. Figure 26 shows the Control Electronics asserting the /FAIL signal.

10. Turn-On Sequence

The turn-On sequence from figure 1 cannot be verified at this level. The turn-on sequence will be verified at the system level, when the Control Electronics is embedded into Rantec's power supply.

X. Conclusion

Though the project has not been embedded into Rantec's 3U power supply, the Control Electronics met most of its standalone specifications and performance. All C-Code drivers have gone through extensive testing and proven to be reliable. There was one drawback with the Two-Wire Interface master bit-bang driver. This driver did not meet the Two-Wire clock speed requirement of 400 kHz. It was discovered that the hardware limits the clock speed to 100 kHz. For future projects, the clock speed can be increased by replacing the 16MHz crystal oscillator with a higher frequency oscillator.

The Two-Wire Interface slave driver clock speed is determined by the master device and based on the microcontroller's datasheet; it's capable of operating up to 400 kHz. This operating speed was not verified, since the master device (National Instruments NI-8451 USB to I2C adapter) has a maximum clock speed of 250 kHz; nevertheless, it's possible for this diver to operate at 400 kHz.

The printed circuit board designed for this project had no electrical flaws or manufacturing defects. The only thing that could be improved is the silkscreen. For example, U$1 could be replaced with U1.

This project successfully integrated hardware and software for the Control Electronics design. The Two-Wire interface master and slave drivers are compatible with Philip's I^2C serial communication protocol, and the printed circuit board can be used for future projects.

XI. References

[1] Rantec Power Systems Inc, "SD35798: Requirements Specification for a 28V Input Conduction-Cooled VITA 62/3U/4HP Power Supply," Los Osos, 2012.

[2] Atmel Corporation, "8-bit Atmel Microcontroller with 16K/32K/64K Bytes In-System Programmable Flash," 1 July 2010. [Online]. Available: http://www.atmel.com/products/microcontrollers/avr/default.aspx?tab=documents&Asset_Type=010+Datasheet. [Accessed 1 February 2013].

[3] R. Ford and C. Coulston, Design for Electrical and Computer Engineers, McGraw-Hill, 2007, p. 37

[4] Philips Semiconductors, "UM10204 I2C-bus specification and user manual," 9 October 2012. [Online]. Available: www.nxp.com/documents/user_manual/UM10204.pdf. [Accessed 1 February 2013]

[5] Texas Instruments Incorporated, "ADC121C021/ADC121C021Q/ADC121C027 I2C-Compatible, 12-Bit Analog-to-Digital," 2013. [Online]. Available: http://www.ti.com/lit/ds/symlink/adc121c021.pdf. [Accessed 7 January 2013]

[6] Texas Instruments Incorporated, "LM94021/LM94021Q Multi-Gain Analog Temperature Sensor," 2013. [Online]. Available: http://www.ti.com/lit/ds/symlink/lm94021.pdf. [Accessed 27 February 2013].

[7] B. W. Kernighan and D. M. Ritchie, The C Programming Language, second ed., New Jersey: Prentice Hall, 1989.

[8] Atmel Corporation, "Atmel AVR127: Understanding ADC parameters," 2011. [Online]. Available: www.atmel.com/images/doc8456.pdf. [Accessed 3 February 2013].

[9] Atmel Corporation, "AVR: Using the TWI Module as I^2C Slave," 2009. [Online]. Available: www.atmel.com/images/doc2565.pdf. [Accessed 3 February 2013].

[10] Atmel Corporation, "Atmel AVR156: TWI Master Bit Bang Driver," June 2012. [Online]. Available: http://www.atmel.com/Images/doc42010.pdf. [Accessed 3 February 2013].

[11] D. Johnson, "Implementing serial bus interfaces with general purpose digital instrumentation," in AUTOTESTCON, 2009 IEEE, Irvine, 2009.

[12] G. L. Dybsetter and J. C. Hahin, "Two-wire interface having embedded per frame reliability information". USA Patent 7,287,208, 23 October 2007.

[13] M. Nahvi, Nahvi on Signals and Systems (Lecture Notes for EE228 and EE328), San Luis Obispo: El Corral Publications, 2010.

Appendix A. Senior Project Analysis

Project Title: Microcontroller Design: Control Electronics for Rantec's 28VDC Input 3-U Power Supply

Student's Name: Leo Hernandez
Student's Signature:
Advisor's Name: Dr. Braun
Advisor's Initials:
Date: December 6, 2012

- ## Summary of Functional Requirements

  The Control Electronics monitors the output voltages, input voltage, intermediate bus voltage, and the temperature of the 3U Power Supply. The control electronics reports this data via the I^2C Interface Bus to the outside world. The Control Electronics greatly reduces troubleshooting time. Refer to Table I for more information on the requirements specifications for the Control Electronics.

- ## Primary Constraints

  - Limited time to build and test the project. The customer may need the Firmware by June of 2013 to meet their Qualification schedule.
  - Limited knowledge of C-programming.
  - Lack of resources to design the printed circuit board and limited knowledge of printed circuit board houses that can build a reliable prototype at a low cost.

- ## Economic

  After the project ends, the design will be embedded into Rantec's power supply for Design and Verification Testing. After the power supply meets the Design and Verification parameters, the power supply goes into a Qualification stage at the customers facility and then into full production.

  If manufactured on a commercial basis

  - The Non Re-occurring cost is $8,224.
  - Approximately 50 units sold per year.
  - $400 to build and test each device. A minimum of 20 pieces are required to cover the manufacturing equipment setup cost.
  - $1000 per unit.
  - The estimated profit is $25,000 per year.

- ## Environmental

  The Control Electronics contains semiconductor devices. These semiconductor devices are manufactured on wafers containing natural elements such as silicon, germanium, gallium or arsenic. Chemicals involved in the manufacturing process of semiconductors, such as arsenic and cadmium are carcinogens, and pose a severe health risk if not handled properly. Great care should be taken so that these toxic components don't pollute our environment.

- ## Manufacturability

  The Control Electronics design could be built on a single printed circuit board. Printed circuit boards provide noise immunity and greatly reduce the manufacturing cost when the design is produced in

mass quantities. Should the design go into full production, the product will be manufactured per IPC-JSTD-001 Soldering Standard.

- **Sustainability**

  The printed circuit board for this project is highly maintainable and requires low maintenance. Components chosen for this project such as resistors, capacitors, and the microprocessor come in standard packages and therefor are easily replaceable. Dust can accumulate over electronics after a long period of time. The dust can be wiped off with a small brush using isopropyl alcohol. The printed circuit board is recyclable; after the project's life cycle, the user should take printed circuit board to a recycling center where valuable items such as gold and copper elements can be re-used.

- **Ethical**

  The Control Electronics contains semiconductor devices. These semiconductor devices are manufactured on wafers containing natural elements such as silicon, germanium, gallium or arsenic. These devices are manufactured in semiconductor fabrication plants located in countries where rules and regulations are not as strict as in the United States. The primary reason that semiconductor parts are built in third world countries is the extremely low labor cost.

- **Health and Safety**

  This product operates at relatively low voltages (3.3-28 VDC), posing no major safety hazard to the user. The user however, shall be careful not to touch any electronic component while the electrical power is applied. Once fully assembled, the Control Electronics shall have a protective cover to reduce electrical hazards. The assemblers assembling the Control Electronics project shall avoid breathing the solder fumes. An air ionizer fan can be used blow the solder fumes away from the assembler.

- **Social and political**

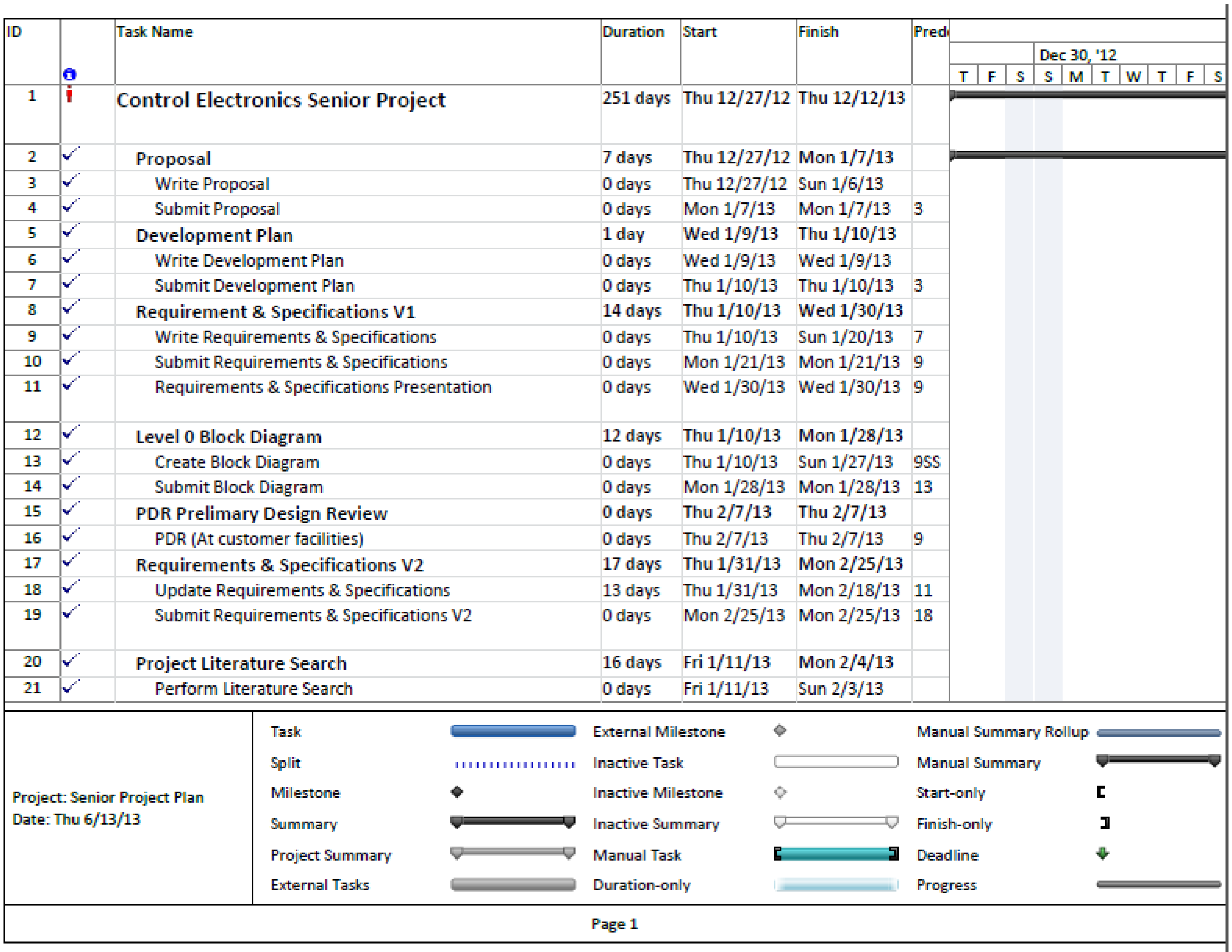
  The Control Electronics is integrated into Rantec's 3U DC-DC converter. This project directly impacts Rantec Power Systems Inc. by expanding their custom DC-DC converter product line and could potentially attract new customers.
  This U.S. made product provides job opportunities to people living in the San Luis Obispo County.

- **Development**

  Since this project integrated hardware, firmware and software, several skills were learned in this project. As an electrical engineering student, it was very challenging programming in C, specially, developing firmware in two different development environments, Atmel Studio and CodeVision AVR. Atmel studio is free software; however, the customer recommended using CodeVision AVR, a $220 C-compiler. CodeVision AVR is great C-compiler, however, it has a different user interface and it was not possible to simply copy the code from the Atmel Studio development environment.
  A requisite for the Control Electronics project was to develop a Two-Wire Interface master bit bang driver; this demanded over 30 hours of extensive reading of the I^2C serial communication protocol. Another requisite required the development of the Two-Wire Interface slave driver; it took over 20 hours of research to develop this driver, since there is not much information available online.
  At the completion of this project, the skills learned include, microcontroller design, C-programming, on-board chip debugging, printed circuit board layout techniques, and in-depth knowledge of the I^2C serial communication protocol.

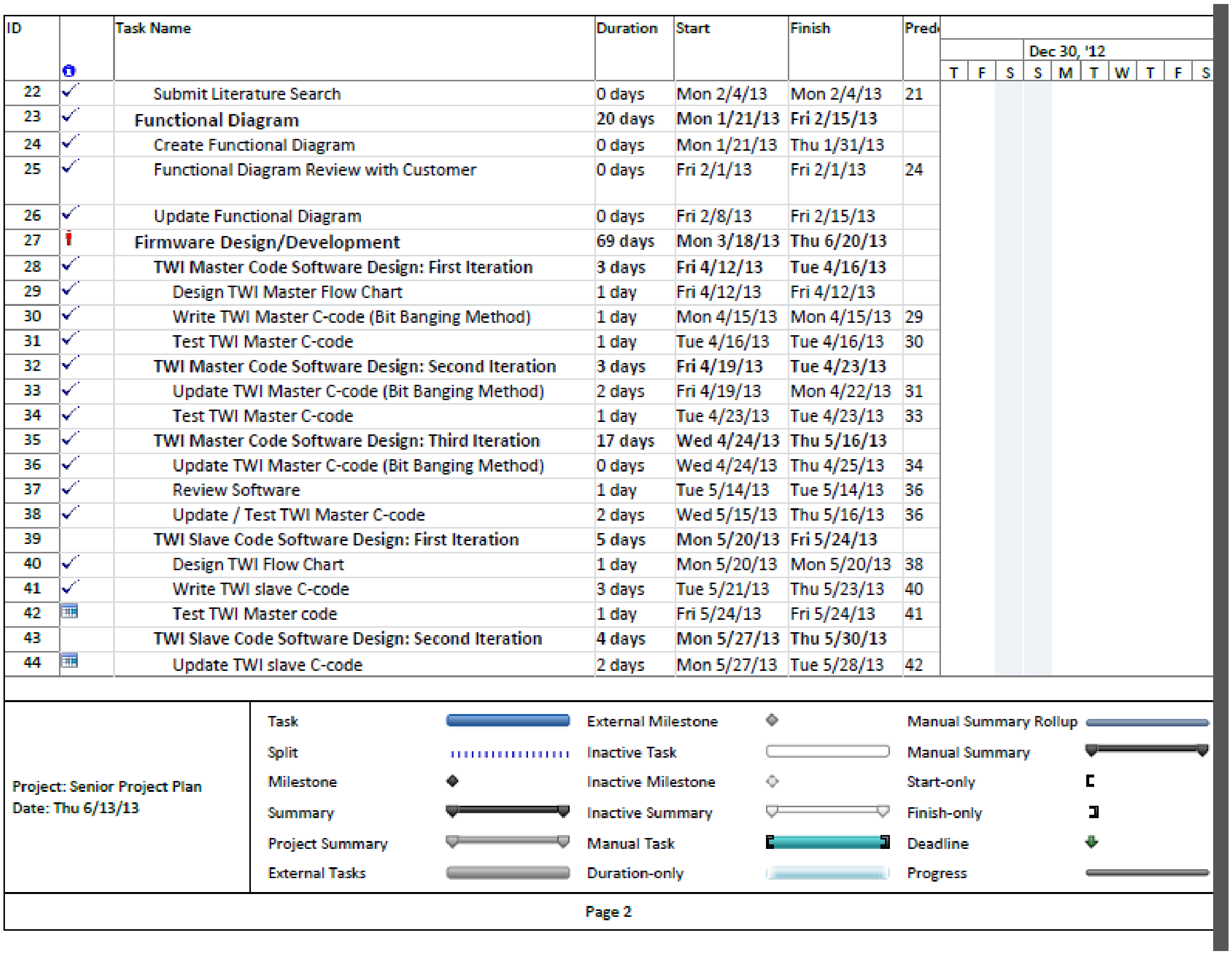
Appendix B. Project Planning

The Gantt chart shows the original senior project schedule.



ID		Task Name	Duration	Start	Finish	Pred
1		Control Electronics Senior Project	251 days	Thu 12/27/12	Thu 12/12/13	
2	✓	Proposal	7 days	Thu 12/27/12	Mon 1/7/13	
3	✓	Write Proposal	0 days	Thu 12/27/12	Sun 1/6/13	
4	✓	Submit Proposal	0 days	Mon 1/7/13	Mon 1/7/13	3
5	✓	Development Plan	1 day	Wed 1/9/13	Thu 1/10/13	
6	✓	Write Development Plan	0 days	Wed 1/9/13	Wed 1/9/13	
7	✓	Submit Development Plan	0 days	Thu 1/10/13	Thu 1/10/13	3
8	✓	Requirement & Specifications V1	14 days	Thu 1/10/13	Wed 1/30/13	
9	✓	Write Requirements & Specifications	0 days	Thu 1/10/13	Sun 1/20/13	7
10	✓	Submit Requirements & Specifications	0 days	Mon 1/21/13	Mon 1/21/13	9
11	✓	Requirements & Specifications Presentation	0 days	Wed 1/30/13	Wed 1/30/13	9
12	✓	Level 0 Block Diagram	12 days	Thu 1/10/13	Mon 1/28/13	
13	✓	Create Block Diagram	0 days	Thu 1/10/13	Sun 1/27/13	9SS
14	✓	Submit Block Diagram	0 days	Mon 1/28/13	Mon 1/28/13	13
15	✓	PDR Prelimary Design Review	0 days	Thu 2/7/13	Thu 2/7/13	
16	✓	PDR (At customer facilities)	0 days	Thu 2/7/13	Thu 2/7/13	9
17	✓	Requirements & Specifications V2	17 days	Thu 1/31/13	Mon 2/25/13	
18	✓	Update Requirements & Specifications	13 days	Thu 1/31/13	Mon 2/18/13	11
19	✓	Submit Requirements & Specifications V2	0 days	Mon 2/25/13	Mon 2/25/13	18
20	✓	Project Literature Search	16 days	Fri 1/11/13	Mon 2/4/13	
21	✓	Perform Literature Search	0 days	Fri 1/11/13	Sun 2/3/13	

Figure 33: Original Gantt Chart for the Senior Project—Pane 1



ID	Task Name	Duration	Start	Finish	Predecessors
22	Submit Literature Search	0 days	Mon 2/4/13	Mon 2/4/13	21
23	**Functional Diagram**	**20 days**	**Mon 1/21/13**	**Fri 2/15/13**	
24	Create Functional Diagram	0 days	Mon 1/21/13	Thu 1/31/13	
25	Functional Diagram Review with Customer	0 days	Fri 2/1/13	Fri 2/1/13	24
26	Update Functional Diagram	0 days	Fri 2/8/13	Fri 2/15/13	
27	**Firmware Design/Development**	**69 days**	**Mon 3/18/13**	**Thu 6/20/13**	
28	**TWI Master Code Software Design: First Iteration**	**3 days**	**Fri 4/12/13**	**Tue 4/16/13**	
29	Design TWI Master Flow Chart	1 day	Fri 4/12/13	Fri 4/12/13	
30	Write TWI Master C-code (Bit Banging Method)	1 day	Mon 4/15/13	Mon 4/15/13	29
31	Test TWI Master C-code	1 day	Tue 4/16/13	Tue 4/16/13	30
32	**TWI Master Code Software Design: Second Iteration**	**3 days**	**Fri 4/19/13**	**Tue 4/23/13**	
33	Update TWI Master C-code (Bit Banging Method)	2 days	Fri 4/19/13	Mon 4/22/13	31
34	Test TWI Master C-code	1 day	Tue 4/23/13	Tue 4/23/13	33
35	**TWI Master Code Software Design: Third Iteration**	**17 days**	**Wed 4/24/13**	**Thu 5/16/13**	
36	Update TWI Master C-code (Bit Banging Method)	0 days	Wed 4/24/13	Thu 4/25/13	34
37	Review Software	1 day	Tue 5/14/13	Tue 5/14/13	36
38	Update / Test TWI Master C-code	2 days	Wed 5/15/13	Thu 5/16/13	36
39	**TWI Slave Code Software Design: First Iteration**	**5 days**	**Mon 5/20/13**	**Fri 5/24/13**	
40	Design TWI Flow Chart	1 day	Mon 5/20/13	Mon 5/20/13	38
41	Write TWI slave C-code	3 days	Tue 5/21/13	Thu 5/23/13	40
42	Test TWI Master code	1 day	Fri 5/24/13	Fri 5/24/13	41
43	**TWI Slave Code Software Design: Second Iteration**	**4 days**	**Mon 5/27/13**	**Thu 5/30/13**	
44	Update TWI slave C-code	2 days	Mon 5/27/13	Tue 5/28/13	42

Figure 34: Gantt Chart for the Senior Project Pane—2

ID		Task Name	Duration	Start	Finish	Pred
45		Test TWI Master code	2 days	Wed 5/29/13	Thu 5/30/13	44
46		**TWI Slave Code Software Design: Third Iteration**	**5 days**	**Fri 5/31/13**	**Thu 6/6/13**	
47		Update TWI slave C-code	2 days	Fri 5/31/13	Mon 6/3/13	45
48		Review Software	1 day	Tue 6/4/13	Tue 6/4/13	47
49		Update / Test TWI Master code	3 days	Tue 6/4/13	Thu 6/6/13	47
50		**Software Requirements**	**27 days**	**Wed 5/15/13**	**Thu 6/20/13**	
51		Create SRS	10 days	Wed 5/15/13	Thu 6/20/13	49
52		SRS Review	0 days	Thu 6/20/13	Thu 6/20/13	51
53		**Interim Report**	**58 days**	**Mon 3/18/13**	**Wed 6/5/13**	
54		Create Interim Report	54 days	Mon 3/18/13	Thu 5/30/13	
55		Submit interim Report	4 days	Fri 5/31/13	Wed 6/5/13	54
56		**Electrical Design**	**53.2 days**	**Fri 4/12/13**	**Wed 6/26/13**	
57		Electrical Design	6 days	Fri 4/12/13	Mon 6/24/13	52
58		Electrical Design Review	2 days	Mon 6/24/13	Wed 6/26/13	57
59		Update Electrical Design	2 days	Mon 6/24/13	Wed 6/26/13	58FF
60		**Build Prototype Circuit**	**20 days**	**Fri 4/5/13**	**Thu 5/2/13**	
61		Order Parts	6 days	Fri 4/5/13	Fri 4/12/13	
62		Build Prototype Circuit	10 days	Thu 4/18/13	Wed 5/1/13	61
63		Inspect Prototype Circuit	1 day	Thu 5/2/13	Thu 5/2/13	62
64		**Integration**	**26 days**	**Thu 5/30/13**	**Thu 7/4/13**	
65		Test/ Integrate Hardware & software	20 days	Thu 5/30/13	Wed 6/26/13	
66		Validation	6 days	Thu 6/27/13	Thu 7/4/13	65
67		**Project Report**	**57 days**	**Wed 9/25/13**	**Thu 12/12/13**	
68		Write Project Report	41 days	Wed 9/25/13	Wed 11/20/13	

Dec 30, '12
T F S S M T W T F S

Project: Senior Project Plan
Date: Thu 6/13/13

Task	External Milestone	Manual Summary Rollup
Split	Inactive Task	Manual Summary
Milestone	Inactive Milestone	Start-only
Summary	Inactive Summary	Finish-only
Project Summary	Manual Task	Deadline
External Tasks	Duration-only	Progress

Page 3

Figure 35: Gantt Chart for the Senior Project—Pane 3

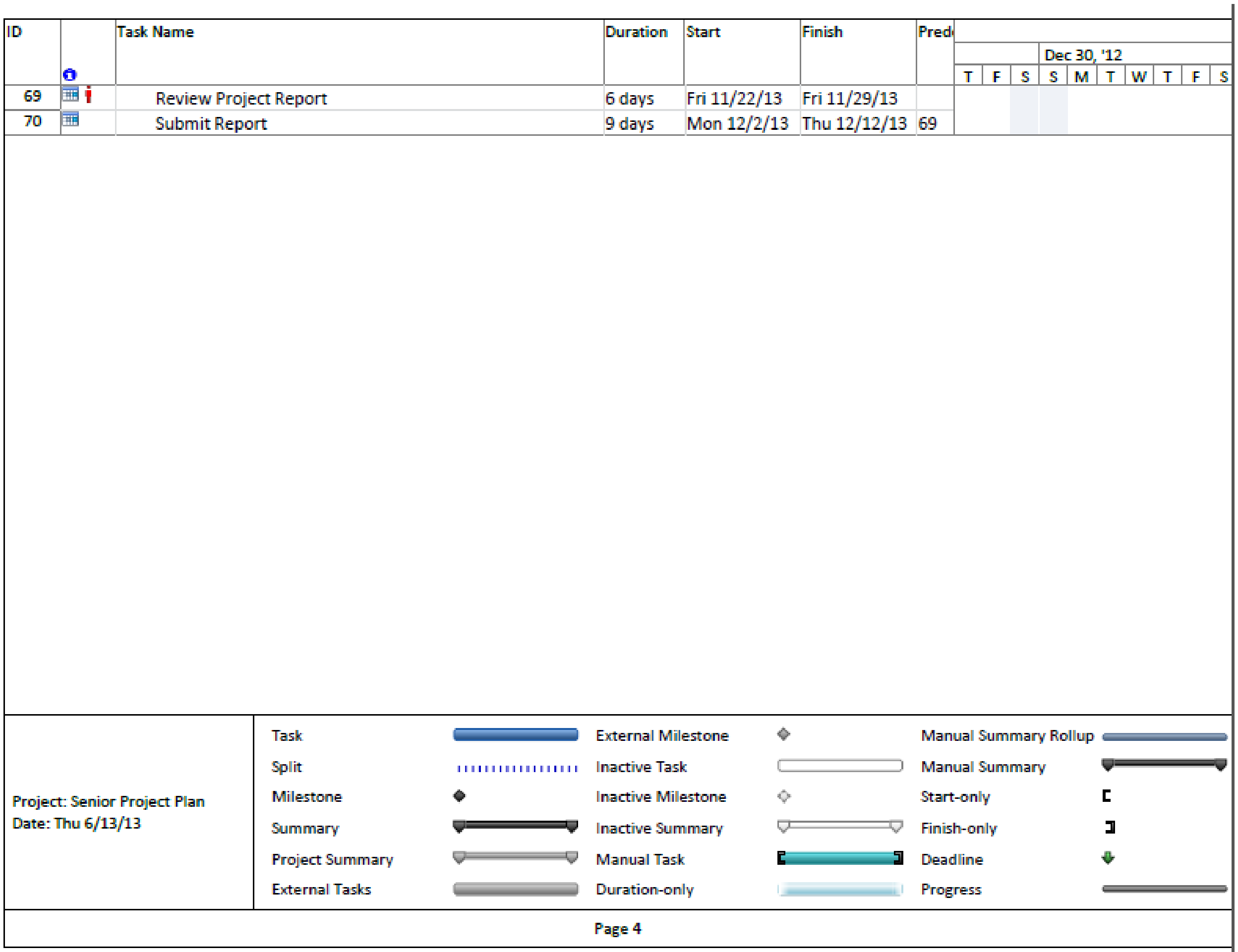


Figure 36: Gantt Chart for the Senior Project—Pane 4

The Gantt chart shows the actual senior project schedule.

ID	ℹ	Task Name	Duration	Start	Finish	Pred
1	✓	**Control Electronics Senior Project**	**254 days**	**Thu 12/27/12**	**Sun 12/15/13**	
2	✓	**Proposal**	**7 days**	**Thu 12/27/12**	**Mon 1/7/13**	
3	✓	Write Proposal	0 days	Thu 12/27/12	Sun 1/6/13	
4	✓	Submit Proposal	0 days	Mon 1/7/13	Mon 1/7/13	3
5	✓	**Development Plan**	**1 day**	**Wed 1/9/13**	**Thu 1/10/13**	
6	✓	Write Development Plan	0 days	Wed 1/9/13	Wed 1/9/13	
7	✓	Submit Development Plan	0 days	Thu 1/10/13	Thu 1/10/13	3
8	✓	**Requirement & Specifications V1**	**14 days**	**Thu 1/10/13**	**Wed 1/30/13**	
9	✓	Write Requirements & Specifications	0 days	Thu 1/10/13	Sun 1/20/13	7
10	✓	Submit Requirements & Specifications	0 days	Mon 1/21/13	Mon 1/21/13	9
11	✓	Requirements & Specifications Presentation	0 days	Wed 1/30/13	Wed 1/30/13	9
12	✓	**Level 0 Block Diagram**	**12 days**	**Thu 1/10/13**	**Mon 1/28/13**	
13	✓	Create Block Diagram	0 days	Thu 1/10/13	Sun 1/27/13	9SS
14	✓	Submit Block Diagram	0 days	Mon 1/28/13	Mon 1/28/13	13
15	✓	**PDR Prelimary Design Review**	**0 days**	**Thu 2/7/13**	**Thu 2/7/13**	
16	✓	PDR (At customer facilities)	0 days	Thu 2/7/13	Thu 2/7/13	9
17	✓	**Requirements & Specifications V2**	**17 days**	**Thu 1/31/13**	**Mon 2/25/13**	
18	✓	Update Requirements & Specifications	13 days	Thu 1/31/13	Mon 2/18/13	11
19	✓	Submit Requirements & Specifications V2	0 days	Mon 2/25/13	Mon 2/25/13	18
20	✓	**Project Literature Search**	**16 days**	**Fri 1/11/13**	**Mon 2/4/13**	
21	✓	Perform Literature Search	0 days	Fri 1/11/13	Sun 2/3/13	
22	✓	Submit Literature Search	0 days	Mon 2/4/13	Mon 2/4/13	21
23	✓	**Functional Diagram**	**20 days**	**Mon 1/21/13**	**Fri 2/15/13**	
24	✓	Create Functional Diagram	0 days	Mon 1/21/13	Thu 1/31/13	
25	✓	Functional Diagram Review with Customer	0 days	Fri 2/1/13	Fri 2/1/13	24
26	✓	Update Functional Diagram	0 days	Fri 2/8/13	Fri 2/15/13	

Apr 21,
W T F S S M

Project: Senior Project Plan
Date: Sat 12/14/13

Task	External Milestone	Manual Summary Rollup
Split	Inactive Task	Manual Summary
Milestone	Inactive Milestone	Start-only
Summary	Inactive Summary	Finish-only
Project Summary	Manual Task	Deadline
External Tasks	Duration-only	Progress

Page 1

Figure 37: Actual Gantt Chart for the Senior Project—Pane 1

ID	Task Name	Duration	Start	Finish	Pred
27	**Firmware Design/Development**	**129 days**	**Mon 3/18/13**	**Thu 9/12/13**	
28	**TWI Master Code Software Design: First Iteration**	**3 days**	**Fri 4/12/13**	**Tue 4/16/13**	
29	Design TWI Master Flow Chart	1 day	Fri 4/12/13	Fri 4/12/13	
30	Write TWI Master C-code (Bit Banging Method)	1 day	Mon 4/15/13	Mon 4/15/13	29
31	Test TWI Master C-code	1 day	Tue 4/16/13	Tue 4/16/13	30
32	**TWI Master Code Software Design: Second Iteration**	**3 days**	**Fri 4/19/13**	**Tue 4/23/13**	
33	Update TWI Master C-code (Bit Banging Method)	2 days	Fri 4/19/13	Mon 4/22/13	31
34	Test TWI Master C-code	1 day	Tue 4/23/13	Tue 4/23/13	33
35	**TWI Master Code Software Design: Third Iteration**	**17 days**	**Wed 4/24/13**	**Thu 5/16/13**	
36	Update TWI Master C-code (Bit Banging Method)	0 days	Wed 4/24/13	Wed 4/24/13	34
37	Review Software	1 day	Tue 5/14/13	Tue 5/14/13	36
38	Update / Test TWI Master C-code	2 days	Wed 5/15/13	Thu 5/16/13	36
39	**TWI Slave Code Software Design: First Iteration**	**5 days**	**Mon 5/20/13**	**Fri 5/24/13**	
40	Design TWI Flow Chart	1 day	Mon 5/20/13	Mon 5/20/13	38
41	Write TWI slave C-code	3 days	Tue 5/21/13	Thu 5/23/13	40
42	Test TWI Slave code	1 day	Fri 5/24/13	Fri 5/24/13	41
43	**TWI Slave Code Software Design: Second Iteration**	**4 days**	**Mon 5/27/13**	**Thu 5/30/13**	
44	Update TWI slave C-code	2 days	Mon 5/27/13	Tue 5/28/13	42
45	Test TWI Slave code	2 days	Wed 5/29/13	Thu 5/30/13	44
46	**TWI Slave Code Software Design: Third Iteration**	**10 days**	**Fri 8/30/13**	**Thu 9/12/13**	
47	Update TWI slave C-code	7 days	Fri 8/30/13	Mon 9/9/13	45
48	Review Software	1 day	Tue 9/10/13	Tue 9/10/13	47
49	Update / Test TWI Master code	3 days	Tue 9/10/13	Thu 9/12/13	47
50	**Interim Report**	**58 days**	**Mon 3/18/13**	**Wed 6/5/13**	
51	Create Interim Report	54 days	Mon 3/18/13	Thu 5/30/13	
52	Submit interim Report	4 days	Fri 5/31/13	Wed 6/5/13	51
53	**Electrical Design**	**54 days**	**Fri 4/12/13**	**Wed 6/26/13**	

Apr 21,
W T F S S M

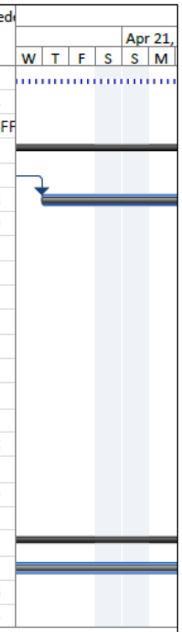
Project: Senior Project Plan
Date: Sat 12/14/13

Task | Split | Milestone | Summary | Project Summary | External Tasks
External Milestone | Inactive Task | Inactive Milestone | Inactive Summary | Manual Task | Duration-only
Manual Summary Rollup | Manual Summary | Start-only | Finish-only | Deadline | Progress

Page 2

Figure 38: Actual Gantt Chart for the Senior Project—Pane 2

ID		Task Name	Duration	Start	Finish	Pred
54	✓	Electrical Design	4.8 days	Fri 4/12/13	Mon 6/24/13	
55	✓	Electrical Design Review	1 day	Tue 6/25/13	Tue 6/25/13	54
56	✓	Update Electrical Design	2 days	Tue 6/25/13	Wed 6/26/13	55FF
57	✓	**Build Prototype Circuit: First Iteration**	**20 days**	**Fri 4/5/13**	**Thu 5/2/13**	
58	✓	Order Parts	6 days	Fri 4/5/13	Fri 4/12/13	
59	✓	Build Prototype Circuit	10 days	Thu 4/18/13	Wed 5/1/13	58
60	✓	Inspect Prototype Circuit	1 day	Thu 5/2/13	Thu 5/2/13	59
61	✓	**Integration: First Iteration**	**12 days**	**Thu 8/29/13**	**Fri 9/13/13**	
62	✓	Test/ Integrate Hardware & software	6 days	Thu 8/29/13	Thu 9/5/13	
63	✓	Update Software	6 days	Fri 9/6/13	Fri 9/13/13	62
64	✓	**Build Prototype Circuit: Second Iteration**	**22 days**	**Fri 10/4/13**	**Mon 11/4/13**	
65	✓	Order Parts	20 days	Fri 10/4/13	Thu 10/31/13	
66	✓	Build Prototype Circuit	3 days	Thu 10/31/13	Mon 11/4/13	
67	✓	**Integration: Third Iteration**	**10 days**	**Mon 11/4/13**	**Fri 11/15/13**	
68	✓	Test/ Integrate Hardware & software	9 days	Mon 11/4/13	Thu 11/14/13	
69	✓	Update Software	1 day	Fri 11/15/13	Fri 11/15/13	68
70	✓	**Integration: Third Iteration**	**15 days**	**Sun 11/17/13**	**Fri 12/6/13**	
71	✓	Test/ Integrate Hardware & software	7 days	Sun 11/17/13	Tue 11/26/13	69
72	✓	Final Demostration	8 days	Wed 11/27/13	Fri 12/6/13	
73	✓	**Project Report**	**229 days**	**Thu 1/31/13**	**Sun 12/15/13**	
74	✓	Write Project Report	226 days	Thu 1/31/13	Thu 12/12/13	
75	✓	Review Project Report	1 day	Sun 12/15/13	Sun 12/15/13	76
76	✓	Submit Report	1 day	Sat 12/14/13	Sat 12/14/13	74



Project: Senior Project Plan
Date: Sat 12/14/13

Task	External Milestone	Manual Summary Rollup
Split	Inactive Task	Manual Summary
Milestone	Inactive Milestone	Start-only
Summary	Inactive Summary	Finish-only
Project Summary	Manual Task	Deadline
External Tasks	Duration-only	Progress

Page 3

Figure 39: Actual Gantt Chart for the Senior Project—Pane 3

Appendix C. Parts List and Cost Estimate

Table XII shows the Control Electronics Parts List.

TABLE XII: PARTS LIST

Description	QTY	Reference Designator	Value	Package
Capacitor	1	C12	.01uF	0603
Capacitor	2	C1, C2	20pF	0603
Capacitor	22	C3, C4, C5, C6, C7, C8, C9, C10, C11, C13, C14, C15, C17, C19, C21, C23, C25, C27, C29, C30, C32, C33, C34	.1uF	0603
Capacitor	8	C16,C18,C20,C22,C24, C26,C28, C31	4.7uF	0603
Resistor	9	R4, R13, R14, R15, R16, R17, R18, R17	40.2	1206
Resistor	1	R20	200	1206
Resistor	1	R40	95.3	0603
Resistor	1	R44	1.87k	1206
Resistor	1	R34	13k	1206
Resistor	1	R41	9.76k	1206
Resistor	1	R45	5.23k	1206
Resistor	4	R1, R8, R9, R10, R11	10k	1206
Resistor		R2, R3	2k	1206
Resistor		R22	28k	1206
Resistor	18	R6, R7, R21, R23, R24, R29, R31, R32, R33, R35, R40, R41, R42, R43, R44, R45, R46	1K	1206
				1206
Resistor		R5	10	1206
AVR Microcontroller	1	U1	ATMega324P	TQPF
Temp Sensor	2	U$1, U$2	LM94021	SOT23
1-channel ADC	7	IC2, IC4, IC6, IC8, IC10, IC12, IC14	ADC121C021CIMM	8VSSOP
4.096V Ref Zener	8	IC3, IC5, IC7, IC9, IC11, IC13, IC15, IC17	LM4050	SOT23
10P Double Row Header	11	SV1-SV11	10 Pins	Thru hole
16Mhz Crystal Oscillator	1	Xtal	ECS-160-20-30B-DU	SMT
Resistor	9	R25, R26, R27, R28, R36, R37	3.74k	0805
Diode	1	D2	1N4148	Thru hole
LED	1	LED2	Green LED	Thru hole

Tables XIII trough XV describe the material and labor cost for this project.

TABLE XIII: LABOR HOURS

	Estimated	Actual
Proposal	2	4
Development Plan	2	2
Requirements	24	24
Presentation	1	1
Specifications	18	18
PDR	5	5
Flow Charts	32	40
Block Diagram	20	25
CDR	10	10
Firmware Development	80	123
Build Prototype circuit	24	24
Test Prototypes	40	40
Hardware & Software integration	40	60
Maintain	10	2
Total Labor Hours	**308**	**378**

TABLE XIII: MATERIAL COST

Material/Software	Estimated Cost	Actual Cost
Development Kit	$300.00	$300.00
Software Development Environment	$300.00	$300.00
Aardvark I2C/SPI Host Adapter	$275.00	$0.00
NI-8451 I^2C/SPI device	$0.00	$473.00
Microprocessor	$2.00	$2.00
ADCs	$5.00	$6.00
Crystal	$2.00	$0.00
SM Resistors	$10.00	$5.00
SM Capacitors	$5.00	$5.00
Prototype Board	$100.00	$516.00
Total Material Cost	**$1000.00**	**$1559.00**

TABLE XIV: LABOR & MATERIAL COST

		Estimated	Actual
Labor Cost	Estimated: 308 hours @ $25/hour Actual: 378 hours @ $25/hour	$7,700.00	$9450.00
Material Cost	Material Cost	$992.00	$875.00
	Total Cost	**$8,224.00**	**$11,009.00**

Appendix E. Source code

Header File

```
/***************************************************************************
VME.h
Two Wire Interface Header
Created by Leo H. for Rantec's 3U power supply
4/19/2012
***************************************************************************/

/*********************************Global Variables*****************************/

// Global timers
unsigned int T0_Outp_Enable;  // LED timer
unsigned int ADC_Timer2;  // ADC Timer
unsigned int Temp_Timer1;  // Temp Timer
unsigned int Timer2;  // ADC Timer

//Global TWI variables
unsigned int TWI_Status;  // Reset TWI hardware
unsigned int TWI_uC_Address;

//unsigned int Reg_Data;
char Reg_Data[22];
char TX_Reg_Data[22];
char PortA_Status;
char OVP_Status;  // Port B
char SysReset;
char FAIL;


unsigned int OVP_Limit[7];
unsigned int OVP_Hysteresis[7];
char Volt_ADC_Addr[7];
float Volt_Scl_Fact[7];



/********************** Measurement Data Variables *************/

int Temp_Data;  //Temperature measurement data
```

```
/**********Temperature Sensor ADC Lookup Table********/
// Vref = 4.096V
unsigned int TEMP_TABLE [244] = {
     832, 829, 827, 824, 822, 819, 817, 814, 811, 808, 805, 803, 800, 797, 793, 790,
     787, 784, 780, 777, 774, 771, 761, 764, 761, 758, 754, 751, 748, 745, 741, 738,
     735, 731, 728, 725, 722, 718, 715, 712, 708, 705, 702, 698, 695, 692, 689, 685,
     682, 679, 675, 672, 669, 665, 662, 658, 655, 652, 648, 645, 642, 638, 635, 632,
     628, 625, 622, 618, 615, 612, 608, 605, 602, 598, 595, 591, 588, 585, 581, 578,
     575, 571, 568, 565, 561, 558, 554, 551, 548, 544, 441, 537, 534, 531, 527, 524,
     520, 517, 514, 510, 507, 503, 500, 496, 493, 490, 486, 483, 479, 476, 472, 469,
     465, 462, 458, 455, 451, 448, 444, 441, 437, 434, 430, 427, 423, 420, 416, 413,
     409, 406, 402, 399, 395, 392, 388, 385, 381, 378, 374, 371, 367, 364, 360, 357,
     353, 350, 346, 343, 339, 336, 332, 329, 325, 322, 318, 314, 311, 307, 304, 300,
     297, 293, 290, 286, 283, 279, 275, 272, 268, 265, 261, 258, 254, 250, 247, 243,
     240, 236, 232, 229, 225, 222, 218, 215, 211, 207, 204, 200, 197, 193, 189, 186,
     182, 178, 175, 171, 168, 164, 160, 157, 153, 149, 146, 142, 138, 135, 132, 128,
     124, 121, 117, 114, 110, 107, 103, 99,  96,  92,  89,  85,  81,  78,  74,  71,
     67,  63,  60,  56,  53,  49,  45,  42,  38,  35,  31,  28,  24,  20,  17,  13,
     10,  6,   2,   0 };

/********************Function Prototypes*********/
/*
Retrieve Temperature Data
*/
void Retrieve_Temp_Data(void);

/*
Sequence enable outputs
*/
void Seq_Enable(void);

/*
Enable outputs
*/
void Output_Enable (char outno, char cmd);


/********************TWI Master Function Prototypes***/
/*
Retrieve ADC Voltage Data
*/
char Retrieve_ADC_Volt_Data(char num);

/*
Measure ADC voltage data
*/
char Meas_Volt(char mindex, char *p_volt);
/*
Measure unit temperature
*/
int Meas_Temperature(void);

/*
```

```
Send TWI START, address, & read/write
*/
char Send_TWI_Start_Addr_RW(char slv_addr, char rw);

/*
**** Send TWI Master Write ******
*/
char Write_TWI_Master(char slv_addr, char *p_data, char dlength);


/*
**** Send TWI Master Read ******
*/
char Read_TWI_Master( char slv_addr, char *p_data, char dlength);


/*
**** Initialize TWI ADcs ******
*/
void Initialize_TWI_ADCs(void);


/************TWI Bit Banging Function Prototypes*******/

/*
**** Send I2C byte ******
*/
int Send_Byte(char byte);

/*
**** ReceiveI2C byte ******
*/
char Receive_Byte(char lastbyte);

/*
**** Send I2C start signal ******
*/
void Send_Start_bit(void);

/*
**** Write I2C SCL bit ******
*/
void Write_SCL(unsigned char state);
/*
**** Write I2C SDA bit ******
*/
void Write_SDA(unsigned char state);

/*
**** Send I2C stop signal ******
*/
void Send_Stop_bit(void);
```

Main Program

```
/*****************************************************
Project : Control Electronics
Version : 1.0
Date    : 4/19/2013
Author  : Leo Hernandez
Company : Rantec Power Systems Inc. & Cal Poly State University, San Luis Obispo
Comments: This is a microcontroller designs project for Rantec Power Systems Inc.


Chip type               : ATmega324P
Program type            : Application
AVR Core Clock frequency: 16.000000 MHz
Memory model            : Small
External RAM size       : 0
Data Stack size         : 512
*****************************************************/

#include <mega324.h>
#include <delay.h>
#include <stdlib.h>
#include <math.h>
#include <iobits.h>
#include "VME.h"


#define TW_SR_SLA_ACK  0x60 // SLA+W has been received; ACK has been returned
#define TW_SR_DATA_ACK 0x80 // Previously addressed with own SLA+W; data has been
received; ACK has been returned
#define TW_SR_ARB_LOST_SLA_ACK 0x68 // Arbitration lost in SLA+R/W as Master; own SLA+W
has been received; ACK has been returned
#define TW_SR_STOP 0xA0 // STOP condition or repeated START condition has been received
while still addressed as Slave
#define TW_SR_DATA_NACK 0x88 // Previously addressed with own SLA+W; data has been
received; NOT ACK has been returned

#define TW_ST_SLA_ACK 0xA8 // Own SLA+R has been received; ACK has been returned
#define TW_ST_DATA_ACK 0xB8 // Data byte in TWDR has been transmitted; ACK has been
received
#define TW_ST_LAST_DATA 0xC8 // Last data byte in TWDR has been transmitted (TWEA = “0”);
ACK has been received
#define TW_ST_DATA_NACK 0xC0 // Data byte in TWDR has been transmitted; NOT ACK has been
received
#define TW_BUS_ERROR 0x00 // Bus error due to an illegal START or STOP condition


/*********************************************************************************
ISR TIMER0_OVF_vect
*********************************************************************************/
// Overflows every 15.872uS
interrupt [TIM0_OVF] void timer0_ovf_isr(void)
{
T0_Outp_Enable++;
Temp_Timer1++;
} //


/*********************************************************************************
ISR TIMER2_OVF_vect
```

```
********************************************************************************/
// Overflows every 128uS
interrupt [TIM2_OVF] void timer2_ovf_isr(void)
{
ADC_Timer2++;
Timer2++;
}


/*******************************************************************************
ISR TWI_vect
********************************************************************************/
interrupt [TWI] void i2c_slave_isr(void)

{

    static unsigned char t_index = 0;


    TWI_Status = (TWSR & 0xF8); // Mask TWPS1, & TWPS0

    switch(TWI_Status)
    {

        /********************************Slave Receiver********************************/
        case TW_SR_SLA_ACK: // 0x60: SLA+W has been received; ACK has been returned
            t_index = 0;
            break;

        case TW_SR_DATA_ACK: // 0x80: Previously addressed with own SLA+W; data has been
        received; ACK has been returned
            Reg_Data[t_index] = TWDR;
            t_index++;
            break;

        case TW_SR_ARB_LOST_SLA_ACK: // 0x68: Arbitration lost in SLA+R/W as Master; own
        SLA+W has been received; ACK has been returned
            break;

        case TW_SR_STOP: // 0xA0: A STOP condition or repeated START condition has been
        received while still addressed as Slave
            break;


      /*********************************Error Recovery********************************/
        case TW_SR_DATA_NACK: // 0x88: Previously addressed with own SLA+W; data has been
        received; NOT ACK has been returned
            break;


      /*******************************SlaveTransmitter********************************/

        case TW_ST_SLA_ACK: // 0xA8: Own SLA+R has been received; ACK has been returned
            t_index = 0;
            TWDR = TX_Reg_Data[t_index]; // Load data in the TWI register
            t_index++;
            break;

        case TW_ST_DATA_ACK: // 0xB8: Data byte in TWDR has been transmitted; ACK has
        been received
```

```
            TWDR = TX_Reg_Data[t_index]; // Load data in the TWI register
            t_index++;
            break;

        case TW_ST_LAST_DATA: // 0xC8: Last data byte in TWDR has been transmitted (TWEA
        = "0"); ACK has been received
            break;

        case TW_ST_DATA_NACK: // 0xC0: Data byte in TWDR has been transmitted; NOT ACK
        has been received
            break;

    /******************************TWI Slave Bus Error*********************************/
        case TW_BUS_ERROR: // 0x00: Bus error due to an illegal START or STOP condition
            break;

        default:
            TWCR |= (1<<TWINT); // Clear TWI interrupt flag

    }
        //Nbytes = bytes;
        TWCR |= (1<<TWINT); // Clear TWI interrupt flag

}



/*********************************************************************************
ISR(PCINT1_vect)
*********************************************************************************/
/*
    PortB     B4            B3             B2       B1      B0
    outno    4            3             2         1        0
            -12V_AUX    +3.3V_AUX    +12V    +5V    +3.3V

*/
// Pin change 8-15 interrupt service routine
interrupt [PC_INT1] void pin_change_isr1(void)
{
// PORTA |= (1<<PORTA6) // /FAIL indicates an output voltage is outside the +-10%
tolerance
    char i, portb_data;
    char ovdata = 0x00;

    portb_data = PINB; // Get port data
    ovdata = ~portb_data & 0x1F; // 1 = OVP condition


    if (ovdata)
    {
       OVP_Status = ovdata;

        for (i = 0; i < 5; i++)
        {
            if ( ovdata & (1<<i) )
            {
                Output_Enable(i,2); // overvoltage condition occurred - shutdown output
            }
            if ( i == 1 )
            {
```

```
                SysReset = 0x00; //  Reset SYsreset Signal
                TX_Reg_Data[20] = SysReset; // Reset /Sysreset
            }
        }

        FAIL = 0x00;
        TX_Reg_Data[21] = FAIL;
    }
    PCIFR = 0X02; // Clear interrupt flag
    //PCIFR = (1<<PCIF1); // Clear interrupt flag


}

/*
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
====================================================================================
                                Main - Start Program
====================================================================================
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
*/
void main(void)
{

    int i;
    char portd_data;
    char vdc_enable = 0x03; // initialize /ENABLE & /INHIBIT to logic high
    char outno;


    T0_Outp_Enable = 0x00;
    ADC_Timer2 = 0x00;
    Temp_Timer1 = 0x00;
    Timer2 = 0x00;
    PortA_Status  = 0x00;
    OVP_Status = 0x00;
    SysReset = 0x00;
    FAIL = 0x00;


// Crystal Oscillator division factor: 1
#pragma optsize-
CLKPR=(1<<CLKPCE);
CLKPR=(0<<CLKPCE) | (0<<CLKPS3) | (0<<CLKPS2) | (0<<CLKPS1) | (0<<CLKPS0);
#ifdef _OPTIMIZE_SIZE_
#pragma optsize+
#endif


/***********************************************************************************
Port A Initialization
***********************************************************************************/
/*
PA4= /-12V_AUX_ENABLE, PA3= /+3.3V_AUX_ENABLE, PA2= /+12V_ENABLE, PA1= /+5V_ENABLE, PA0=
/+3.3V_ENABLE,
PA5 = /SYSRESET
PA6 = /FAIL
PA7 ADC7 input
*/
    DDRA = 0x7F; // PA0-PA7 set as output pins
```

```
    PORTA = 0x7F; // All outputs disabled


/*********************************************************************************
Port B Initialization
*********************************************************************************/
/*
PB4= /-12V_AUX_OVP, PB3= /+3.3V_AUX_OVP, PB2= /+12V_OVP, PB1= /+5V_OVP, PB0= /+3.3V_OVP,
PB5 = /ENABLE
PB6 = /INHIBIT
PB7 Not used (configured as input)
*/

    DDRB = 0x00; // Configure as input
    PORTB = 0xFF;  // Enable pull-up resistors


/*********************************************************************************
Port C Initialization
*********************************************************************************/
/*
JATAG port
*/

    DDRC = 0xC0;
    PORTC = 0xC0; // (JTAG port) Configured as input


/*********************************************************************************
Port D Initialization
*********************************************************************************/
    DDRD = 0x03; // PD0 & PD1 set as output pins (SCL & SDA)
    PORTD = 0x00; // All outputs disabled
    PORTD |= (1<<PORTD3) | (1<<PORTD2); // Enable pull-up resistors


/*********************************************************************************
Timer/Counter 0 initialization
*********************************************************************************/
    TCCR0A = 0x00;
    TCNT0 = 0;            // Initialize counter
    OCR0A = 0x00;
    OCR0B = 0x00;
    TIFR0 |= (1<<TOV0);    // clear overflow flag

    /*For 16MHz clock:
    B2-0 settings are:
        Setting          Frequency   Period      Timeout (256 count)
        0 = no clock
        1 = ClkIO         16MHz         62nS        15.872uS
        2 = ClkIO/8      2MHz        500nS       128uS
        3 = ClkIO/32     500kHz      2.00uS      512uS
        4 = ClkIO/64     250kHz      4.00uS      1.024mS
        5 = ClkIO/128    125kHz      8.00uS      2.048mS
        6 = ClkIO/256    62.5kHz     16.00uS     4.096mS
        7 = ClkIO/1024   15.625kHz   64.00uS     16.384mS*/
    TCCR0B = 0x01;        //set CLK/1 = 62ns
```

```
/*******************************************************************************
Timer/Counter 2 initialization
*******************************************************************************/
/*
    Clock source: System Clock
    Mode: Normal top=FFh
    OC2A output: Disconnected
    OC2B output: Disconnected
    */
    ASSR = 0x00;        // Set asynchronous settings to 0.
    TCCR2A = 0x00;
    TCNT2 = 0x00;
    OCR2A = 0x00;
    OCR2B = 0x00;
    /*

    TCCR2B B2-0 (CS02-CS00) select the timer clock
    For 16MHz clock:
    B2-0 settings are:
        Setting          Frequency   Period      Timeout (256 count)
        0 = no clock
        1 = ClkIO         16MHz         0.062uS     15.872uS
        2 = ClkIO/8      2MHz        .500uS      128uS
        3 = ClkIO/32     500kHz      2.00uS      512uS
        4 = ClkIO/64     250kHz      4.00uS      1.024mS
        5 = ClkIO/128    125kHz      8.00uS      2.048mS
        6 = ClkIO/256    62.5kHz     16.00uS     4.096mS
        7 = ClkIO/1024   15.625kHz   64.00uS     16.384mS
    */
    TCCR2B = 0x02; //set clock/8 128uS timeout



/*******************************************************************************
External Interrupt Initialization
*******************************************************************************/
    EICRA = 0x00; // The low level of INTn generates an interrupt request

    PCICR = (1<<PCIE1); // Enable PCINT1 interrupts
    PCIFR = (1<<PCIF1); // Clear PCINT1 flags
    PCMSK1 = (1<<PCINT12) | (1<<PCINT11) | (1<<PCINT10) | (1<<PCINT9) | (1<<PCINT8); //
(PCINT) 9 Ensure not to select CLKOUT in the fuse bytes!

    TIMSK0 = 0x01;    // Timer0 overflow interrupt enabled
    TIMSK1 = 0x00;    // Timer1 overflow interrupt disabled
    TIMSK2 = 0x01;    // Timer2 overflow interrupt enabled


/*******************************************************************************
ADC Initialization
*******************************************************************************/
    DIDR0 = 0x7F;
    ADMUX = 0x07;
       //ADMUX = 0x47;   // To debug
    ADCSRA = 0x87; // ADC enabled, frequency = 125kH
```

```
/**********************************************************************************
USART0 Initialization
**********************************************************************************/
    UCSR0B = 0x00; // USART0 disabled


/**********************************************************************************
USART1 Initialization
**********************************************************************************/
    UCSR1B = 0x00; // USART1 disabled


/**********************************************************************************
SPI Bus Initialization
**********************************************************************************/
    SPCR = 0x00; // SPI disabled


/**********************************************************************************
Analog Comparator Initialization
**********************************************************************************/
    ACSR = 0x80;
    ADCSRB = 0x00;
    DIDR1 = 0x00;


/**********************************************************************************
Two Wire Interface Bus Slave Initialization
**********************************************************************************/

    portd_data = (PIND & 0x0C);  // Monitor PD3 & PD2 to determine TWI slave address
    //i2c_add = ~portd_data & 0xC0;

    switch(portd_data)
    {
        case 0x00:
            TWI_uC_Address = 0x30;
            break;

        case 0x04:
            TWI_uC_Address = 0x29;
            break;

        case 0x08:
            TWI_uC_Address = 0x28;
            break;

        case 0x0C:
            TWI_uC_Address = 0x27;
            break;

        default:
            TWI_uC_Address = 0x02;

    }

    TWAR = (TWI_uC_Address << 1 | 0); // Shift microcontroller address to the left,
ignore general call address
    TWDR = 0xFF;  // Release SDA pin
```

```
    TWCR = (1<<TWINT) | (1<<TWEN) | (1<<TWEA) | (1<<TWIE);  // Clear TWINT, enable TWI
hardware, enable ACK own address, enable interrupt


/*********************************************************************************
OVP_Limit, OVP_Hysteresis, & voltage scaling factor
*********************************************************************************/

    OVP_Limit[0] = 0xE7B; // OVP = 3.707V (+3.3V)
    OVP_Limit[1] = 0xCC1; // +OVP = 6.25V (+5V)
    OVP_Limit[2] = 0xE7B; // OVP = 3.707V (+12V)
    OVP_Limit[3] = 0xE7B; // OVP = 3.707V (+3.3V_AUX)
    OVP_Limit[4] = 0xE7B; // OVP = -3.707V (-12V_AUX)
    OVP_Limit[5] = 0xF45; // OVP = 3.707 (DC_Bus)
    OVP_Limit[6] = 0xF45; // OVP = 3.707V (V28Vin)

    OVP_Hysteresis[0] = 0x38D; // +3.3V
    OVP_Hysteresis[1] = 0x28D; // +5V
    OVP_Hysteresis[2] = 0x289; // +12V
    OVP_Hysteresis[3] = 0x38D; // +3.3V_AUX
    OVP_Hysteresis[4] = 0x289; // -12V_AUX
    OVP_Hysteresis[5] = 0x38D; // (DC_Bus)
    OVP_Hysteresis[6] = 0x28D; // (V28Vin)

    Volt_ADC_Addr[0] = 0x50; // +3.3V
    Volt_ADC_Addr[1] = 0x51; // +5V
    Volt_ADC_Addr[2] = 0x52; // +12V
    Volt_ADC_Addr[3] = 0x54; // +3.3V_AUX
    Volt_ADC_Addr[4] = 0x55; // -12V_AUX
    Volt_ADC_Addr[5] = 0x56; // (DC_Bus)
    Volt_ADC_Addr[6] = 0x58; // (V28Vin)

    Volt_Scl_Fact[0] = 0.10;     // +3.3V
    Volt_Scl_Fact[1] = 0.1913846; // +5V
    Volt_Scl_Fact[2] = 0.10; // +12V
    Volt_Scl_Fact[3] = 0.10;     //+3.3V_AUX
    Volt_Scl_Fact[4] = 0.10; // -12V_AUX
    Volt_Scl_Fact[5] = 0.064; // (DC_Bus)
    Volt_Scl_Fact[6] = 1.1134861; // (V28Vin)


// Global enable interrupts
#asm("sei")


/****************** Initializing Program for the user ****************************/

    Initialize_TWI_ADCs();

    for ( i = 0; i < 27; i++ )
    {
        PORTC ^= (1<<PORTC7);
        delay_ms(50);
    }


/****************************** Control Inputs ************************************/

    vdc_enable = (PINB & 0x60);  // Monitor PB5 & PB6
```

```
    if (vdc_enable == 0x00)
    {
        Output_Enable(3,1); // Enable 3.3V_AUX
        OVP_Status = 0x00;

        for (i = 0; i < 3; i++)
        {
            Output_Enable(i,2); // Disable outputs: +3.3VDC, +5VDC, +12VDC
        }

        Output_Enable(4,2); // Disable -12V_AUX
    }

    else if(vdc_enable == 0x40) // Enable all outputs
    {
        Seq_Enable();
        OVP_Status = 0x00;

    }

    else
    {
        for ( i = 0; i < 5; i++ )
        {
            Output_Enable(i,2); // Disable all outputs
        }
    }


/*****************************************************************************/
/*                                Main Loop                                  */
/*****************************************************************************/

while (1)
      {
        TX_Reg_Data[17] = PortA_Status; // Output Enable status
        delay_us(1); // This delay is used to debug only one IF Statement inside the
while loop
        TX_Reg_Data[18] = OVP_Status; // Output Enable status

        delay_us(1); // This delay is used to debug only one IF Statement inside the
while loop

      if ( Timer2 >= 1953 ) // Get ADC data every 250mS   // WAS 1953
      {
         PORTC ^= (1<<PORTC7);
         Timer2 = 0x00;
      }



/************************* Get ADC voltage data **********************************/

        if ( ADC_Timer2 >= 100 ) // Get ADC data every 250mS
        {
            outno = 0x00;

            for ( i = 0; i < 7; i++ )
            {
```

```
                Retrieve_ADC_Volt_Data(outno); // Get voltage data from TWI device,
returns true is data is valid
                outno++;
            }

            ADC_Timer2 = 0; // reset Start_Timer variable
        } // End TWI monitor


/************************** Turn On/OFF outputs ***********************************/

        if (Reg_Data[3] == 0x0A)
        {
            OVP_Status = 0x00; // Clear OVP condition
            Seq_Enable(); // Enable Outputs
            delay_ms(250);  // To debug
            Reg_Data[3] = 0x00; // Update register data to detect OV condition
        }

        if(Reg_Data[3] == 0x0C)
        {

            for (i = 0; i < 5; i++)
            {
                OVP_Status = 0x00; // Clear OVP condition
                Output_Enable(i,2);  // Disable outputs
                delay_ms(200);  // To debug
            }
                Reg_Data[3] = 0x00; // Update to detect other conditions
        }


/************************** Get temperature data **********************************/

        if ( Temp_Timer1 > 20000) // every 317.4mS
        {
            Retrieve_Temp_Data();
        }

        if (Temp_Data > 850) // 85 degrees C
        {
        TX_Reg_Data[16] = 0x0F; // Indicate an overtemp condition
            for  (i = 0; i < 5; i++ )
            {
                //Overtemp occurred - shutdown outputs
                Output_Enable(i,2);  // Disable outputs
                delay_ms(100);  // To debug
            }
        }

        if (Temp_Data < 840) // 84 degrees C
        {
            TX_Reg_Data[16] = 0x00; // Indicate an normal temp condition
        }

/******************************** Input Low  ************************************/
        if ( TX_Reg_Data[12] <= 0x04 ) //  MSB
        {
            if ( TX_Reg_Data[13] < 0xE2 ) // If input is < 12.5 then indicate an UV
condition
```

```
            {
                TX_Reg_Data[19] = 0x0D;
                for (i = 0; i < 5; i++)
                {
                    OVP_Status = 0x00; // Clear OVP condition
                    Output_Enable(i,2);  // Disable outputs
                }

               SysReset = 0x00;
               TX_Reg_Data[20] = SysReset; // Assert /SYSRESET
               FAIL = 0x00;
               TX_Reg_Data[21] = FAIL;

            }
        }
        else
        {
        TX_Reg_Data[19] = 0x00; // Indicate an normal VIN condition
        }


/********************************* /SYSRESET **************************************/

// if the +5V is within regulation, the /SYSRESET signal is asserted
        if ( TX_Reg_Data[2] >= 0x01 ) //  MSB
        {
            if ( (TX_Reg_Data[3] > 0xEF ) && ( SysReset == 0 ) ) // If input is > 4.95
            then assert /Sysreset
            {   SysReset = 0x0A;
                delay_ms(200);
                TX_Reg_Data[20] = SysReset; // Assert /SYSRESET
            }

            if ( TX_Reg_Data[3] < 0xEA)
            {
               SysReset = 0x00;
               TX_Reg_Data[20] = SysReset; // Assert /SYSRESET
            }
        }

        if ( TX_Reg_Data[2] < 0x01 ) //  MSB
        {
            SysReset = 0x00;
            TX_Reg_Data[20] = SysReset; // Assert /SYSRESET
        }


/*********************************** /FAIL *************************************/


        if ( ( TX_Reg_Data[1] > 0x29 ) && ( TX_Reg_Data[0] >= 0x01 ) ) //  +3.3V MSB
            if ( ( TX_Reg_Data[3] > 0xC2 ) && ( TX_Reg_Data[2] >= 0x01 ) ) //  +5V MSB
                if ( ( TX_Reg_Data[5] > 0x29 )  && ( TX_Reg_Data[4] >= 0x01 ) ) //  +12V
                MSB
                    if ( ( TX_Reg_Data[7] > 0x29 ) && ( TX_Reg_Data[6] >= 0x01 ) ) //
                    +3.3V_AUX MSB
                        if ( ( TX_Reg_Data[9] > 0x29 ) && ( TX_Reg_Data[8] >= 0x01 ) ) //
                        -12V_AUX MSB
                        {
                           FAIL = 0x0B;
```

```
                        TX_Reg_Data[21] = FAIL;
                    }
        if ( ( TX_Reg_Data[0] < 0x01 ) || ( TX_Reg_Data[2] < 0x01 ) || ( TX_Reg_Data[4] <
           0x01 ) || ( TX_Reg_Data[6] < 0x01 ) || ( TX_Reg_Data[8] < 0x01 ) )
        {
            FAIL = 0x00;
            TX_Reg_Data[21] = FAIL;
        }

    } // End while loop

} // End main program


/*
==================================================================================
                                End Main Program
==================================================================================
*/
```

```
/*****************************************************************************
Retrieve_ADC_Volt_Data
*****************************************************************************/
/*
    Retrieve_ADC_Volt_Data
    num = ADC number
    0        1      2      3           4                5          6
    +3.3V    +5V    +12V   +3.3V_AUX   -12V_AUX       DC_Bus      Vin
    cmd = 0x01, 0x02,  0x03,  0x04,       0x04      0x05        0x06
*/


char Retrieve_ADC_Volt_Data(char num)
{
    char vdata[16];
    char data_valid = 0, i;
    char reg_ind;

    reg_ind =  num*2;

    if (num < 0x07)
    {

        //It's a valid command
        data_valid = Meas_Volt(num, vdata);
        if (data_valid)
        {

            for ( i = 0; i < 2; i++ )
            {
                TX_Reg_Data[reg_ind + i] = vdata[i];
            }

        }
    }

    return data_valid;

}  // End Process_Status_Cmd


/*****************************************************************************
Initialize_TWI_ADCs
*****************************************************************************/
/*
    Initialize all the TWI Analog-to-Digital Converters for output voltage measurements

    Configure single channel ADCs
    0        1     2       3          4        5           6
    +3.3V  +5V  +12V  +3.3V_AUX  -12V_AUX      DC_Bus      Vin
*/


void Initialize_TWI_ADCs(void)
{
    int i;
    char regdata[3];

    for (i=0; i < 7; i++)
    {
```

```
        // Write OVP voltage to High Lim reg 0x04
        regdata[0] = 0x04; // Reg 0x04
        regdata[1] = (char)((OVP_Limit[i] >> 8) & 0x0F); // Shift bits to right, 8 
places.  Load MSB bytes
        regdata[2] = (char)(OVP_Limit[i] & 0xFF); // Load LSB bytes
        Write_TWI_Master(Volt_ADC_Addr[i],regdata,3); // Write to High Lim reg

        // Write OVP Hysteresis to Alert Hyst reg 0x05
        regdata[0] = 0x05; // Reg 0x05
        regdata[1] = (char)((OVP_Hysteresis[i] >> 8) & 0x0F); // Shift bits to right, 8 
places.  Load MSB bytes
        regdata[2] = (char)(OVP_Hysteresis[i] & 0xFF);   // Load LSB bytes
        Write_TWI_Master(Volt_ADC_Addr[i],regdata,3); // Write to Alert Hyst reg

        // Set configuration register data
        // Write to config reg 0x02
        regdata[0] = 0x02; // Reg 0x02
        regdata[1] = 0x24; // Configure auto-conversion 32uS, Alert pin enable, active 
low
        Write_TWI_Master(Volt_ADC_Addr[i],regdata,2); // Write to config reg

        // Reset over range bit in the Alert Status Register
        regdata[0] = 0x01;
        regdata[1] = 0x02;
        Write_TWI_Master(Volt_ADC_Addr[i],regdata,2);

        // Write to conv result reg 0x00
        regdata[0] = 0x00; // Reg 0x00
        Write_TWI_Master(Volt_ADC_Addr[i],regdata,1);
    }

} // Initialize_TWI_ADCs


/*******************************************************************************
Meas_Volt
*******************************************************************************/
/*
    Initialize all the TWI Analog-to-Digital Converters for output voltage measurements
    Measure voltage    0        1     2       3          4       5          6
    mindex = meas index
                    +3.3V    +5V  +12V  +3.3V_AUX  -12V_AUX        DC_Bus      Vin
    p_Volt = two byte voltage data
    p_mvalue = measured voltage
    Returns data valid
*/

char Meas_Volt(char mindex, char *p_volt)
{
    // char nbytes = 0;
    char data_valid = 0, p_data[2];
    unsigned int adcdata;
    //int voltdata;
    int voltdata;
    float mvalue, fpart, ipart;

    data_valid = Read_TWI_Master(Volt_ADC_Addr[mindex], p_data, 2);

    if (data_valid)
```

```c
    {

        // Convert 2 byte adc data to integer
        adcdata = ((unsigned int)(p_data[0] & 0xF)) << 8; //may need to write 0x0F
        adcdata |= (unsigned int)p_data[1];

        // Scale voltage data
        mvalue = (float) adcdata * Volt_Scl_Fact[mindex];

        // Return 2 bytes
        fpart = modf(mvalue, &ipart);
        voltdata = (int) ipart;
        if (fpart >= 0.5) voltdata ++;  // Round up
        p_volt[0] = (char)(voltdata >> 8);
        p_volt[1] = (char)voltdata;

    }

    return data_valid;

} // Meas_Volt


/*********************************************************************************
Write_TWI_Master
*********************************************************************************/
/*
    Write to TWI device via the master TWI interface
    slv_addr = TWI slave address
    *p_data = data byte array
    dlength = data length
    returns 1 if data is successfully transmitted to the TWI device
*/


 char Write_TWI_Master(char slv_addr, char *p_data, char dlength)
{
    char data_valid = 0, i=0;

    /*Send Start and Address+Write to TWI device*/
    data_valid = Send_TWI_Start_Addr_RW(slv_addr, 0);
    /* If TWI device acknowledges address, then send data*/

    if (data_valid)
    {
        do
        {
            data_valid = Send_Byte(p_data[i]);  //Send data to TWI device and device
returns ACK or NACk
        } while ((++i < dlength) && data_valid);  //loop if not last byte and ACK
received

    }
    Send_Stop_bit();   //Send STOP condition to TWI

    return data_valid;  //Return true if data is successfully transmitted

} // End Write_TWI_Master
```

```
/*****************************************************************************
Read_TWI_Master
*****************************************************************************/
/*
    Read from TWI device via the master TWI interface
    slv_addr = TWI slave address
    *p_data = receive data byte array
    dlength = data length
    Returns 1 if data is successfully received from the TWI device
*/


char Read_TWI_Master( char slv_addr, char *p_data, char dlength)
{
    char data_valid = 0, i = 0;

    /*Send Start and Address+Write to TWI device*/

    data_valid = Send_TWI_Start_Addr_RW(slv_addr, 1);

    if( data_valid )
    {
        // Receive data bytes
        do
        {
            if (i == (dlength - 1)) // Check to see if last byte
            {
                 p_data[i] = Receive_Byte(1);//send NACK before STOP condition
            }
            else
            {
                 p_data[i] = Receive_Byte(0);//Receive byte and send ACK
            }

        } while ( ++i < dlength ); //loop if not last byte and no error occurred

    }

    Send_Stop_bit();  //Terminate communication
    return data_valid;

} // End Read_TWI_Master


/*****************************************************************************
Send_TWI_Start_Addr_RW
*****************************************************************************/
/*
    Send Start condition and slave address + Read/Write* to TWI device
    slv_addr = TWI slave address
    rw = 1=Read, 0=write
    returns 1 if TWI ACK, returns 0 if TWI NACK
*/


char Send_TWI_Start_Addr_RW(char slv_addr, char rw)
{
    char ack = 0;
    char retry = 1;
    //char retry = 0;
```

```
    do
    {

        Send_Start_bit();  //Send START condition
        ack = Send_Byte(slv_addr << 1 |rw);  // Send slave address + read/write, returns
true if TWI device ACK


    } while ( !ack && (retry-- == 1) );  //Send repeated start if not acknowledge

    return ack;

}  // End Send_TWI_Start_Addr_RW



/*********************************************************************************
Sequence_Enable
*********************************************************************************/
void Seq_Enable(void)
{

                Output_Enable(3,1); // Enable +3.3V_Aux
                delay_ms(41);
                Output_Enable(0,1); // Enable +3.3V
                delay_ms(4);
                Output_Enable(1,1); // Enable +5V
                delay_ms(4);
                Output_Enable(2,1); // Enable +12V
                Output_Enable(4,1); // Enable +12V_AUX

 } // End Seq_Enable


/*********************************************************************************
Output_Enable
*********************************************************************************/
/*
    Enable DC outputs
    PortA     B4            B3            B2      B1      B0
    outno     4             3             2       1       0
          -12V_AUX     +3.3V_AUX     +12V    +5V    +3.3V
    cmd: 2 = disable, 1 = enable output
*/

void Output_Enable(char outno, char cmd)
{

    if (cmd == 0x01)
    {
        PORTA &= ~(1<<outno); // Enable Output
        PortA_Status |= (1<<outno);
    }

    else if( cmd == 0x2)
    {
        PORTA |= (1<<outno); // Disable Output
        PortA_Status &= ~(1<<outno);
    }
```

```c
} // Output_Enable


/**************************************************************************
Retrieve_Temp_Data
**************************************************************************/
void Retrieve_Temp_Data(void)
{
    char t_index = 0;
    Temp_Data =  Meas_Temperature();

    //Two Temperature bytes
    TX_Reg_Data[t_index + 14] = (char)(Temp_Data >> 8); // High byte
    TX_Reg_Data[t_index + 15] = (char)(Temp_Data & 0xFF); // Low byte
}


/**************************************************************************
Meas_Temperature
**************************************************************************/
/*

*/

int Meas_Temperature()
{
    unsigned int adcdata;
    char i, hibyte, lowbyte;
    float tval, tval1, tval2, fpart, ipart;
    int tempval;

    ADMUX = 0x07;
    ADCSRA |= (1<<ADSC); // Start conversation
    while ( (ADCSRA & (1<<ADIF)) == 0 ); //  Wait for conversation to complete
    lowbyte = ADCL; // Retrieve low byte
    hibyte = ADCH; // Retrieve high byte
    adcdata = ((unsigned int)hibyte) << 8; // Add high byte
    adcdata|= (unsigned int)(lowbyte); // Add low byte
    ADCSRA |= (1<<ADIF); // Clear interrupt flag



    // Get Temperature
    i=0;
    while (adcdata < TEMP_TABLE[i++]);

    i--;
    tval = (float)i -55; // (add -55 degC offset)


    if ( (adcdata > TEMP_TABLE[i]) && (i > 0) )
    {
        // Value is between table values - interpolate value
        tval1 = (float)(TEMP_TABLE[i]);
        tval2 = (float)(TEMP_TABLE[i-1]);
        tval -= (((float)adcdata - tval1) / (tval2 - tval1));
        tval *= 10;
        fpart = modf(tval, &ipart);
        tempval = (int) ipart;
```

```
        //Round
        if (fpart >= .5)
        tempval++; // Round up

    }
    else
    {
        tval *= 10; //
        tempval = (int) tval;
    }
    return tempval;  //Integer value with LSB = 0.1C
}



/*
/XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX  I2C Bit Bang Drivers XXXXXXXXXXXXXXXXXXXXXXXXXX/
*/


/*********************************************************************************
Send_Byte
*********************************************************************************/
/*
    Sends one byte of data to the TWI slave device, MSB first.
    Returns a 1 if the slave device Acknowledges and a 0 if the slave device doesn't
Acknowledge.
    This function does not update the TWI_error variable, since it returns an error
*/

int Send_Byte(char byte)
{
    char i;
    char ack = 0;
    int timeout = 0;


    for(i = 0; i < 8; i++)
    {
        Write_SDA(byte & 0x80); // Send bit, MSB first
        byte = (byte << 1); // shift bits to the left
        Write_SCL(1);
        Write_SCL(0);
    }

    /*The master generates the clock pulse for ACKs, wait for ACK from slave device */

    // Release SDA pin, wait for ACKs
    Write_SDA(1);

    Write_SCL(1);
    ack = PIND.1;
    while ( ack == 1 && ++timeout < 10 )
    {
        ack = PIND.1;
    }

    Write_SCL(0);

    return~(ack);
```

```
} // End Send_Byte


/*****************************************************************************************
Receive_Byte
*****************************************************************************************/
/*
    Receive one byte of data from the TWI slave device, MSB first.
    lastbyte = 1, signals the TWI slave device that the read is over
*/

char Receive_Byte(char lastbyte)
{
    int i;
    char SDA_bit;
    char result = 0;

      Write_SDA(1);
    for(i = 7; i >= 0; i--)
    {
        // Receive each bit at a time, MSB first
        Write_SCL(1);
        SDA_bit = PIND.1; // read SDA pin
        result |= SDA_bit << (i); //Shift left
        Write_SCL(0);
    }

    // Send ACK
    /* // If last byte, then send Master sends NACK.  Else, Master sends ACK.
      (Immediately preceding a STOP condition, this bit must be a NACK) */
    Write_SDA(lastbyte);
    Write_SCL(1);
    Write_SCL(0);
    Write_SDA(1); // Set SDA to input, release SDA pin

    return(result);

} // End Receive_Byte


/*****************************************************************************************
Send_Start_bit
*****************************************************************************************/
/*
    Send Start Bit
    A start condition occurs when SDA transitions from high to Low while the SCL is high.
*/

void Send_Start_bit(void)
{

    Write_SDA(1); // Set SDA high
    Write_SCL(1); // To debug on STK600
    // Set_SCL_High(); // if timeout after 35uS occurs, then return TWI error, else
return no error
    Write_SDA(0);
    Write_SCL(0);

} // End Send_Start_bit
```

```
/******************************************************************************
Write_SDA
******************************************************************************/
/*
    Configure SDA pin as input or output
    0 = input with high Z
    1 = output
*/
void Write_SDA( unsigned char state)
{
    if (state)
    {
        DDRD.1 = 0; // configure PDB1 as high-z input
    }
    else
    {
        DDRD.1 = 1; // configure PORTDB1 as output,
        PORTD.1 = 0; // Set SDA pin low
    }

} // End Write_SDA



/******************************************************************************
Write_SCL
******************************************************************************/
/*
    Configure SCL pin as input or output
    if state = 1, then configure SCL pin as input & high Z
    if state = 0, then configure SCL pin as output
*/

void Write_SCL(unsigned char state)
{
    if (state)
    {
        DDRD.0 = 0; // Configure PORTDB0 as input & high Z
    }
    else
    {
        DDRD.0 = 1;    // Configure PORTDB0 as output,
        PORTD.0 = 0; // Set SCL pin low
    }
} // End Write_SCL



/******************************************************************************
Send_Stop_bit
******************************************************************************/
/*
    A stop condition occurs when SDA transitions low to high while the SCL is high.
    Assumes SCL is low at the beginning
*/

void Send_Stop_bit(void)
```

```
{
    Write_SDA(0); // Set SDA low
    Write_SCL(1); // Set SCL high
    #asm
        nop
        nop
        nop
        nop
    #endasm

    // Release SDA
       Write_SDA(1); // Set SDA high

} // End Send_Stop_bit
```

Appendix E. PC Board Layout

Figure 40 shows the Control Electronics PC board layout.

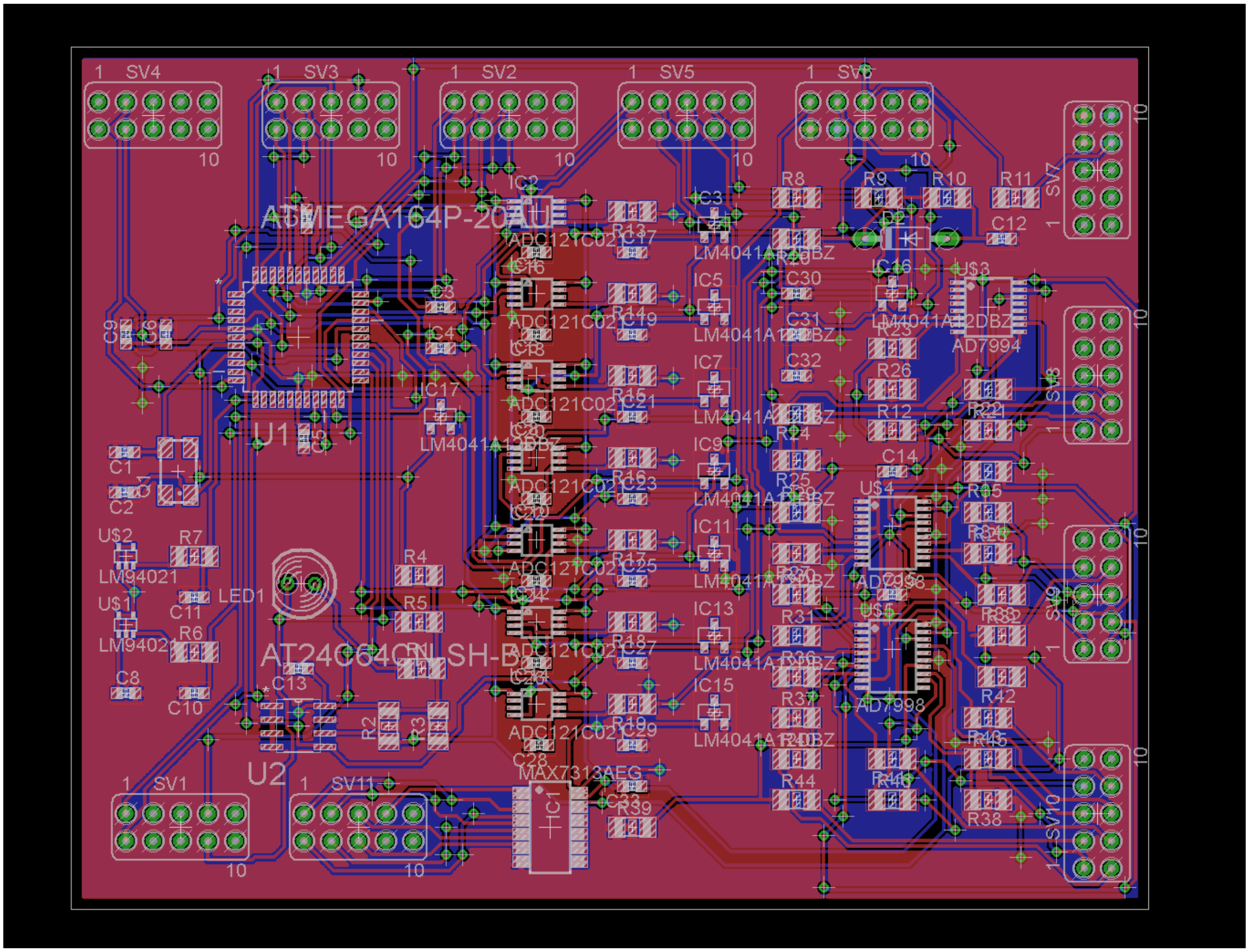


Figure 40: Control Electronics PC board layout