Universal-Retro Console Gamepads to USB Adapter with Two-Player Compatibility

by

Miguel Buenrostro

Senior Project

ELECTRICAL ENGINEERING DEPARTMENT

California Polytechnic State University

San Luis Obispo

Advisor: Dr. Bryan Mealy

June 2013

TABLE OF CONTENTS

Section Page

Appendices

Abstract

An Arduino board or microcontroller development board with a microprocessor can be configured to interface with a variety of wired gaming input devices. Through software and hardware design this functionality is accomplished for the use of older gaming system gamepad devices such as the original Nintendo Entertainment System, the Super Nintendo, and Sega Genesis. The Arduino development board offers up to 18 general purpose input output connections which are sufficient for the simultaneous connection of up to 6 gaming devices of the aforementioned gamepads. The microcontroller‘s firmware is changed to achieved plug-and-play compatibility so windows can recognize the device as a compatible gamepad with upwards of 32 available buttons when the device is plugged in through USB. Select buttons and interrupts are used to choose the active gamepads for two-player compatibility; active gamepads can be any combination of NES, SNES and Genesis gamepad controllers. This device is useful in any microcontroller project in which a gamepad would be a beneficial external input.

Introduction

Video gaming has become a popular form of entertainment in modern society. This has been accomplished through numerous gaming systems since the 80's in which newer graphically improved iterations of older systems are released. Earlier consoles with limited graphical capability only require a simple user interface for the input device or gamepad since game-play is limited to a two-dimensional side-scrolling environment in which a D-pad for movement and a few buttons are more than sufficient. Modern day gamers still prefer to play these older games with the same gamepads they are accustomed to when the games were initially release due to the nostalgia factor.

There is currently a market for adapters that allow people to use their older gamepads as USB game controllers for the PC in order to play older games using video game console emulation software. Many of these products only offer a single gamepad adapter capability, which means one must purchase several adapters if one wants to have several players. Using an Arduino microcontroller with an ATmega 328p processor an adapter can be made for the numerous connections offered by the Arduino. Through software design different polling algorithms are configured to have capability of use of multiple gamepads from different systems through a single USB input.

The design of this project features the following three recognizable older system gamepad input devices, the original Nintendo Entertainment System, the Super Nintendo, and Sega Genesis. The firmware of the Arduino AVR Microcontroller is flashed with a Microsoft Human Interface Device (HID) gamepad compliant firmware to allow plug and play compatibility. The design allows a user to use up to two gamepads using any configuration of the three compliant devices at a single time. Using the original AVR Microcontroller firmware one can use the gamepad polling algorithms for the use of multiple gamepads on any future microcontroller project.

Background

There are several products on the market that offer adapters for old console gamepads to USB, but they are limited to one peripheral per one USB and the ports offered are limited to one gamepad from the three different older consoles. My solution is to create a reliable product that offers multiple gamepad inputs for one USB connection.

I programmed a microcontroller with multiple modes of operation in order to accommodate all the different types of input. Selection of the type of mode of operation is determined through two select buttons, the first button for the 1st player gamepad select and second for the 2nd gamepad select. One can also modify the code to have the microcontroller itself determine which gamepads are connected and select a mode of operation accordingly.

The main market appeal for this product is for anyone interested in playing older 2D games with simple controls in which an older gamepad might be desirable for their nostalgic factor. People who were raised in the 80's and 90's will also find this product appealing since they grew up when these systems were out in the market and have a familiarity to these outdated gamepads. An advantage of this project is that the consumer only has to have one product for all their nostalgia gamepad playing needs.

By using a microcontroller with simplified programming techniques this product will allow for greater customization and possible inclusion of other gamepads, which have a similar polling algorithm to the currently configured gamepads.

Requirements and Specifications

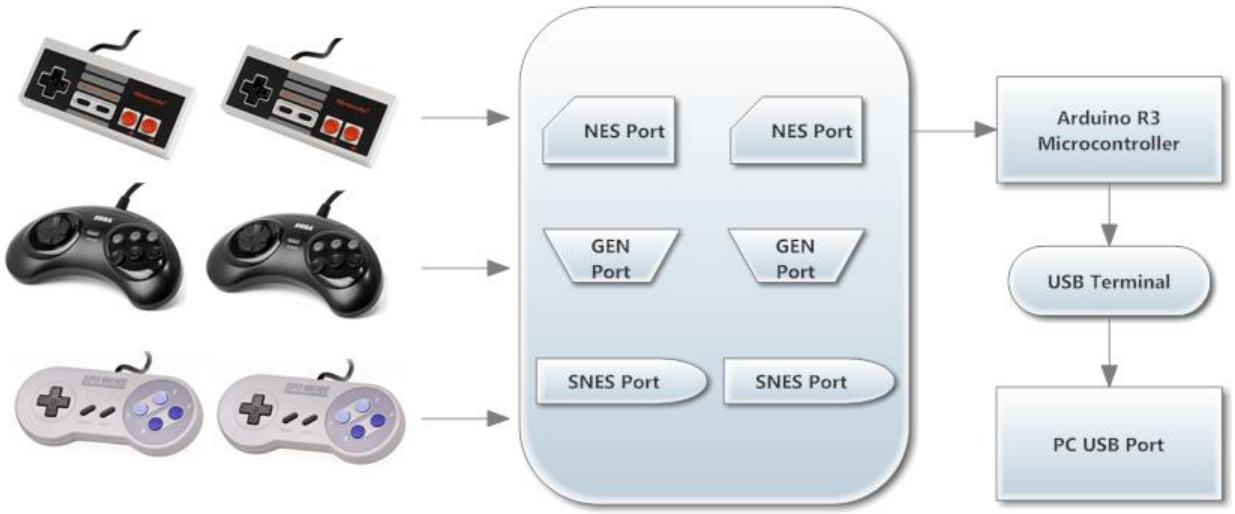

Figure 1: Gamepad interface design.

There are many factors in the development of this project since each gamepad has different interface specifications. Through proper planning and organization, I was able to integrate 2 NES, 2 SNES and 2 Genesis gamepads through an Atmega328p microcontroller and interface the system with a PC computer through USB. The number of gamepad ports exceeds that of Arduino connections, because of this multiple GPIO's are shared, for shared data ports diodes are necessary to guarantee both closed and open circuit connections. External interrupts are used to select the type of gamepad currently in use such as NES, SNES or Genesis in order to simultaneously integrate the different gamepads.

Design

Nintendo Entertainment System - Gamepad Interface

Process for retrieving the button states of an NES gamepad is as follows. The state of button A is retrieved when the latch signal of the microcontroller is maintained high and the button is pressed, when this occurs the data signal is grounded. Once the state of button A is retrieved the latch signal is grounded to low in order to retrieve the remaining gamepad button states. Grounding the latch signal initializes a cycling of the button states after the rising edge of the clock signal. The data input signal cycles through the different buttons states in the following order B, Start, Select, Up, Down, Left, Right. To retrieve accurate button polling information there is 1µs for every cycle of the clock signal.

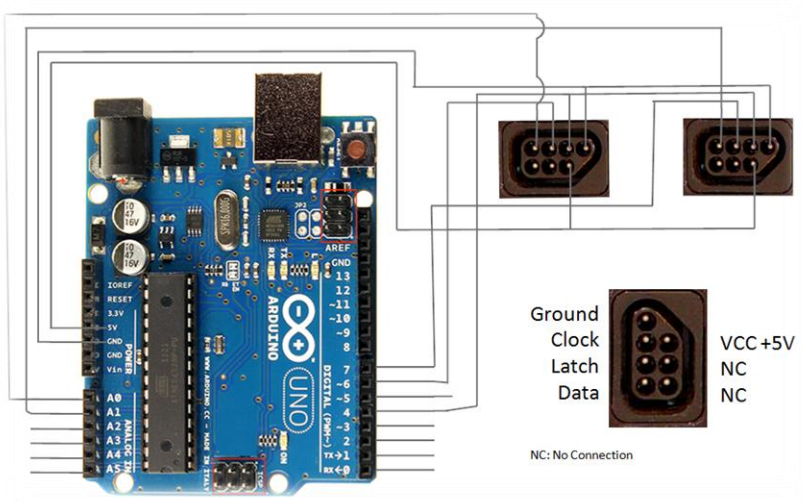

Figure 2: NES gamepad ports connection diagram.

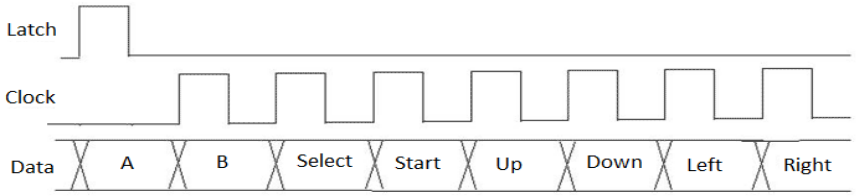

Figure 3: The NES gamepad timing diagram.

Sega Genesis System - Gamepad Interface

The Sega Genesis Gamepad functions slightly different from that of the NES. A Genesis gamepad uses a standard 9-pin connector to interface with the console. The nine pins consist of six data lines, a select input, a 5 Volt input, and a ground connection. There are two versions of the Genesis Gamepad a standard three-button gamepad and a six button gamepad with a maximum of 11 buttons states to be retrieved. Different pulses are sent to the gamepad through the select line. During the first two pulses the Genesis six button gamepad responds the same to the three-button gamepad, however after the third pulse the controller will output new button information of the additional three buttons through the data lines. A pulse consists of changing the voltage from the select line from logic low to immediately logic high.

Pin	Select : Low	Select: High	Select: pulse*3
1	D-Pad UP	D-Pad UP	Button Z
2	D-Pad Down	D-Pad Down	Button Y
3	Logic low	D-Pad Left	Button X
4	Logic low	D-Pad Right	Logic low
6	Button A	Button B	Logic low
9	Button Start	Button C	Logic low

Pin	Other pins
5	Power +5V
7	Select signal
8	Ground

Table 1: Genesis button states with select mode.

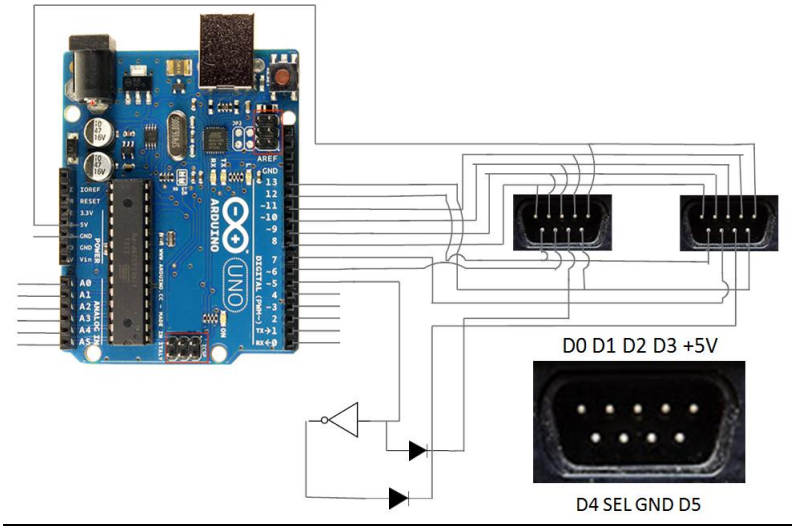

Figure 4: Genesis gamepad ports connection diagram.

To connect two Genesis gamepads to a single Arduino board, the data lines for the two Genesis Gamepads are connected to the same Arduino GPIOs through the use of two diodes and an inverter. When polling the data lines out of one gamepad we must ensure gamepad presses out of the second will not interfere with the grounding of the lines from the first. A supply of 5 Volts is sent to the ground input of the gamepad that is not in use and a supply of 0 Volts or virtual ground to the one in use. An inverter will switch between these two states for both genesis gamepads out of a single Arduino port.

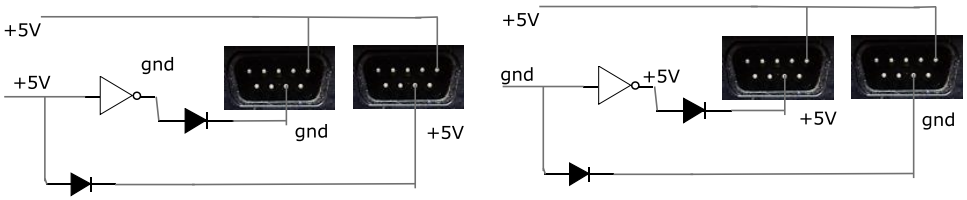

Figure 5: Genesis gamepad port select diagram.

In this implementation, digital port 5 alternates between logic low and logic high in order to enable either gamepad 1 when output is logic high or enable gamepad 2 when logic is low. The diodes

give infinite resistance when 5 Volts are sent through which increases gamepad compatibility since different gamepads have different resistances.

Super Nintendo System – Gamepad Interface

The Super Nintendo gamepad works the same way as the NES gamepad. Process for retrieving the button states of an SNES gamepad is as follows. The select line is initially set to logic high and the data port is grounded when pressing button B. The latch signal then goes to logic low to retrieve the button states after every rising-edge of the clock signal in the following order Y, Select, Start, Up, Down, Left, Right, A, X, Left bumper and Right bumper.

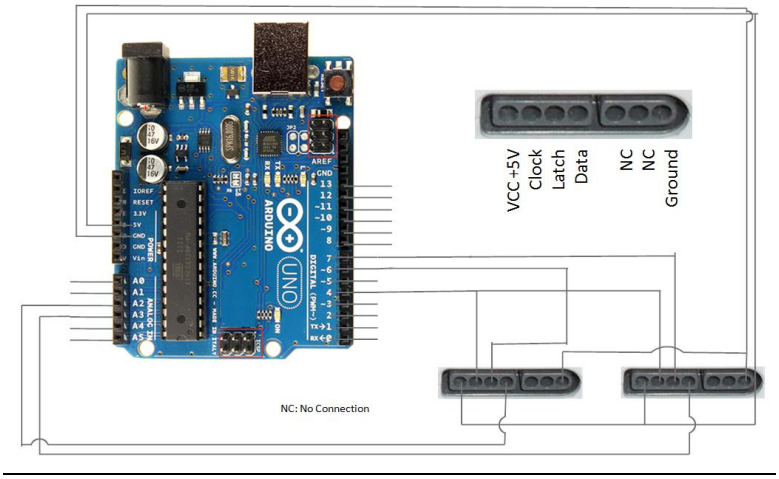

Figure 6: Super Nintendo gamepad ports connection diagram.

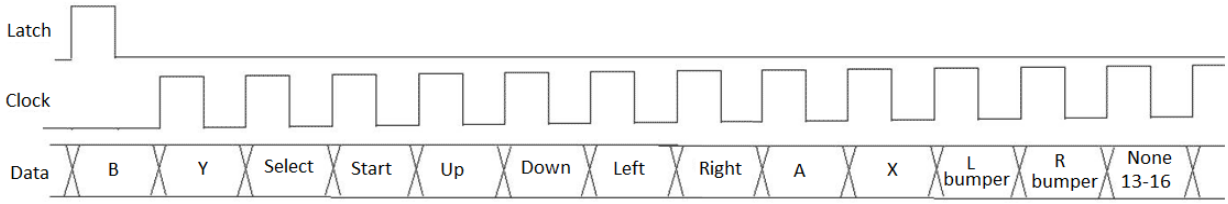

Figure 7: SNES gamepad timing diagram.

Construction

Connecting gamepad ports to a single board

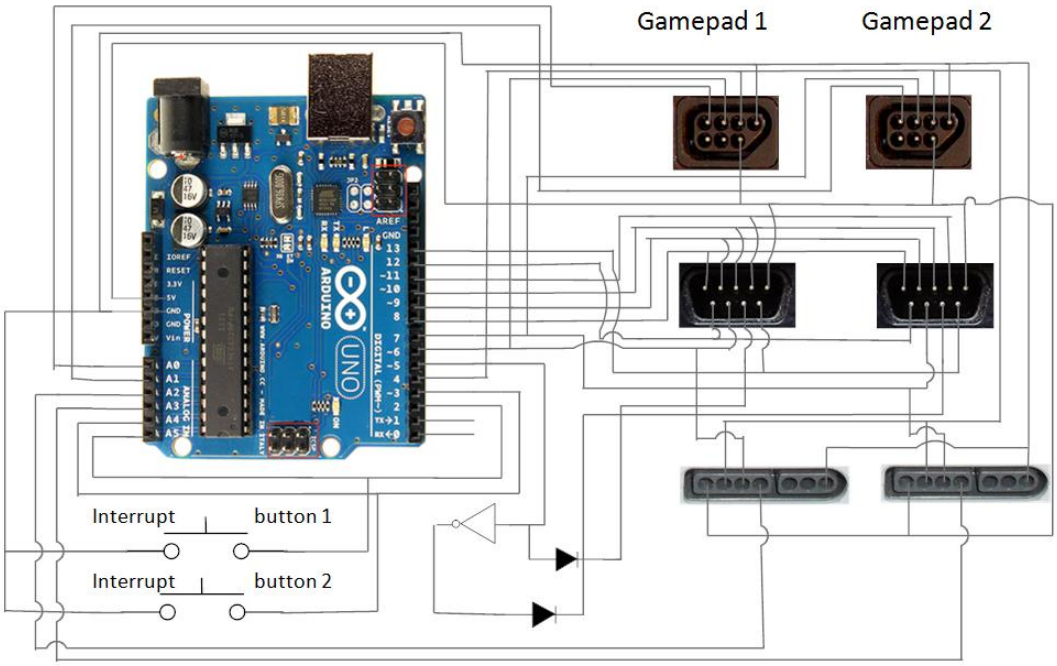

Figure 8: Final Design connection diagram.

Since many ports share similar signals such as select, latch and clock, multiple connections to the game ports are shared. Digital port 6 is responsible for sending latch and select signals to the first player NES, SNES and Genesis connection while digital port 7 is responsible for the second. The clock signal is generated by switching the states between logic low and high. While not producing a 50% duty

cycle signal it has the benefit of reducing the time between switching states as well as only being used when necessary. Digital port 4 is used to output the clock signal to both NES and SNES gamepads, the Sega Genesis gamepad does not require a clock signal for operation.

There are two device external buttons, which are in charge of initializing two separate interrupts for the device. When the buttons are pressed, they initialize an interrupt, which will change the current active gamepad. Two interrupts are implemented in this design, the first for the first gamepad the second interrupt for the second gamepad.

ATMEGA 328-P to USB interface

The Arduino 3 development board uses an Atmega 328P processor to execute the programmable instructions; it also has an ATmega16U2 microcontroller which converts signals coming from the computer to the first serial port. By re-flashing the firmware with a Microsoft Human Interface Device (HID) compliant firmware one can make the Arduino board be recognized as a variety of devices such as MIDI device, keyboard, mouse, or in our case a joystick. One must first set the development board to Device Firmware Upgrade (DFU) mode, process for the setup of this mode differs for the type of microcontroller being used. Process for an Arduino R3 is simple, all one has to do is ground the pin shown on figure 9. The computer will then look for the appropriate DFU drivers, which are supplied on the Arduino manufacturer website. Once device enters into DFU mode, device manager will display the device depending on the type of USB microcontroller, for the Arduino R3 it should display as ATmega16U2.

Figure 9: Arduino board with pin to ground for DFU mode.

There are a variety of software programmers one can use to re-flash the firmware, a simple one to use which uses a Graphical User Interface (GUI) is FLIP from Atmel.

Once the type of USB microcontroller is selected from devices a firmware is then selected to re-flash the device. The USB cable is then un-connected and re-connected to the computer, if a joystick firmware was selected the device will behave as a joystick.

Joystick (HID) firmware

There are several skill sets and knowledge which I was lacking, primarily USB operation and USB driver design. After discussing this proposal with my peers they foresaw several complications with obtaining a working driver for the computer and USB. Driver software can be very difficult and time consuming to develop. As a solution, I changed the microcontroller firmware to an open source driver for a joystick with up to 32-button capability.

Software for the joystick firmware is under the following copyright

Copyright 2010 Dean Camera (dean [at] four walled cubicle [dot] com) Permission to use, copy, modify, distribute, and sell this software and its documentation for any purpose is hereby granted without fee,

provided that the above copyright notice appear in all copies and that both that the copyright notice and this permission notice and warranty disclaimer appear in supporting documentation, and that the name of the author not be used in advertising or publicity pertaining to distribution of the software without specific, written prior permission. The author disclaim all warranties with regard to this software, including all implied warranties of merchantability and fitness. In no event shall the author be liable for any special, indirect or consequential damages or any damages whatsoever resulting from loss of use, data or profits, whether in an action of contract, negligence or other tortuous action, arising out of or in connection with the use or performance of this software.

Interrupts for gamepad selector

The joystick firmware has a maximum of 32-buttons recognized for the driver. This limits the amount of gamepads that can be played at a single time, for the design of this device a maximum of 2 players are allowed at a single time in which one can use two NES, SNES or Genesis gamepads or a combination of the two. Interrupts are useful for this problem since it alerts the processor that the user wants to change the current gamepad in use. Two interrupts are used in this device; the first interrupt is in charge of changing the first player's gamepad while the second interrupt is responsible for changing the second gamepad. A delay is necessary to prevent de-bouncing when an interrupt button is pressed.

Testing

Initial testing and implementation of this project consists of supplying logic high voltage to the multiple gamepads, an oscilloscope is then connected to the data lines to observe their behavior when certain buttons are pressed. Different logic states for the select and latch lines changes the output of the data lines for all the gamepads. Software is then written which retrieves the button states of all the gamepads and stores the information to integer arrays. An initial problem consists of inappropriate timing delays causing false reads when retrieving certain button states.

Having to change the USB firmware on the device to joystick and then back to Arduino Uno's default firmware for program uploading, and then back to joystick is not practical so I added a few methods into the code in order to test that the polling functions for the multiple gamepads were working properly. A Universal Synchronous Asynchronous Receiver Transmitter (USART) function is used for this purpose. Data is sent to the PC through USB using the asynchronous method of serial communication. After each loop through the main program cycle, serial data is sent back to the PC using a baud rate of 9600 bits/second.

Once all the methods are verified to work properly the data information of the button states is sent to the Joystick driver and further testing is then done using PC games to ensure there is low lag between gamepad button presses and a game's reaction time.

Results

System response is functional for game-play of PC games. External buttons initialize interrupts which changes the current active gamepad. There is still a slight de-bouncing problem in which a single press might account for multiple presses. Given a microcontroller board with additional general purpose inputs and outputs one can add LED's which could indicate the current active gamepad. While uncommon, issues of false reads have occurred when using the open source joystick firmware and playing PC games. This problem does not appear to occur when using the device's original firmware and testing the regular button states. A unique firmware for the arduino board which is Joystick (HID) compliant was not developed in this project, given more time a firmware can be developed with a higher reliability than the currently used open source firmware. The current design is reliable for a microcontroller project in which one requires multiple gamepads for external inputs. Future projects can modify the supplied code and use upwards of 6 gamepads at a single time for control purposes with a high reliability.

Conclusion

This project consists of a lot of patience and trial and error. Multiple hours were spent debugging problems through the design process. It is especially difficult trying to keep track of all the different connections of the design. After developing working code, the code is altered in multiple ways in order to try to increase the speed for which the data of the button states are retrieved. Given more time one could potentially develop a circuit board to minimize the amount of wire connections, the circuit board could then act as a shield for the Arduino microcontroller. This device is useful in any microcontroller project in which a gamepad would be a beneficial external input.

References

1. Drew Hall, (January 19, 2012) *Converting an NES controller into a USB Controller*[online] <http://www.zero-soft.com/HW/USB_NES/index.php?page=2 >

2. User 'Ant.b' (October 01, 2010) *Tutorial – How to change firmware on 8u2*[online] <http://Arduino.cc/forum/index.php/topic,111.0.html >

3. Charles Rosenberg (September 9, 1996) *Sega Six Button Controller Hardware Info* [online] < http://www.cs.cmu.edu/~chuck/infopg/segasix.txt >

4. Jim Christy(1999) *SNES / Super Famicom Joystick Data*[online] <http://www.gamesx.com/controldata/snesdat.htm >

5. Dean Camera(2013) *Arduino-big-joystick* [online] <https://github.com/harlequin-tech/Arduino-usb/tree/master/firmwares/Arduino-big-joystick >

6. Atmel Flip Software for Firmware Microcontroller uploading <http://www.atmel.com/tools/FLIP.aspx >

Appendix

Order of design process

1. NES gamepad interface (working with a PC game)
2. 2-NES gamepad interface (working with a PC 2-player game)
3. Super Nintendo gamepad interface (working with a PC game)
4. Simultaneous interface of both NES gamepad and SNES gamepad with a switch to select gamepad input.
5. 2 - Super Nintendo gamepads interface (working with a PC 2-player game)
6. Sega Genesis gamepad interface (working with a PC game)
7. 2- Sega Genesis gamepad interface (working with a PC 2-player game)
8. Simultaneous interface of NES, SNES and Genesis gamepads with a switch to select gamepad input.
9. Complete simultaneous interface between all gamepads and 2 player compatibility.

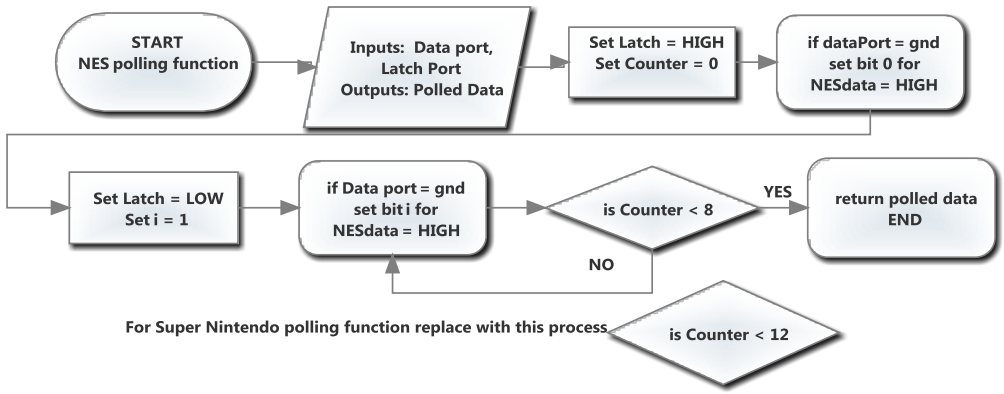

Figure 10: NES polling function software diagram

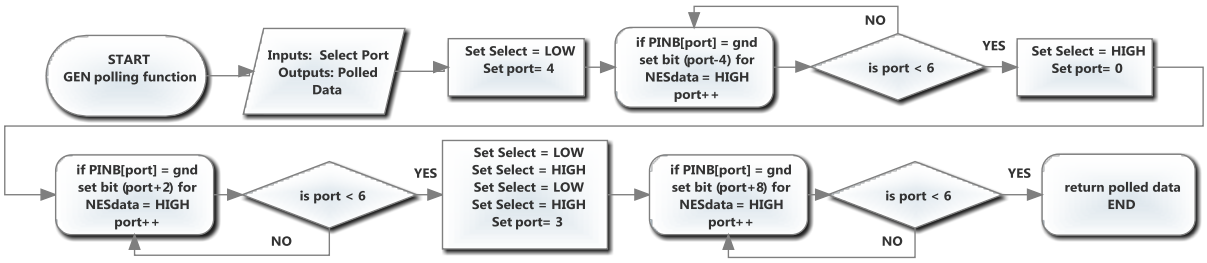

Figure 11: GEN polling function software diagram

Source Code

```
/*
  * Universal-Retro Console Gamepads to USB Adapter
  * with Two-Player Compatibility
*/

/* Author: Miguel Buenrostro
 * joyReport code provided by: Darran Hunt
 * Released into the public domain.
 */

#define F_CPU 16000000               // define internal CLK speed

#define CLOCK   4                    // define CLOCK output

#include <avr/io.h>                  //registers locations and some other things
#include <util/delay.h>              //software delay functions
#include <avr/interrupt.h>

#define BAUD_PRESCALE 103
#define def_delay 6
#undef DEBUG

#define NUM_BUTTONS    40
#define NUM_AXES       8             // 8 axes, X, Y, Z, etc

typedef struct joyReport_t {
    int16_t axis[NUM_AXES];
    uint8_t button[(NUM_BUTTONS+7)/8]; // 8 buttons per byte
} joyReport_t;

joyReport_t joyReport;

void mainUSART(int value);
int getKbit(int k, int n);
int getNES(int LATCH,int DTA);
int getSNES(int LATCH,int DTA);
int getGEN(int LATCH);
int ArraySize(int Array[]);
void printUSART(int value1, int value2, int value3, int value4, int value5, int value6);
void usart_init(uint16_t baudin, uint32_t clk_speedin);
void usart_send( uint8_t data );
int setPAD(int inp_butt,int data,int inp_num);
uint8_t  usart_recv(void);
uint8_t  usart_istheredata(void);
////JOYSTICK CONTROLLER CODE
void setup(void);
void loop(void);
void setButton(joyReport_t *joy, uint8_t button);
void clearButton(joyReport_t *joy, uint8_t button);
void sendJoyReport(joyReport_t *report);

uint8_t button=0;      // current button
bool press = true;     // turn buttons on?
int myNESdata1 = 0;
int myNESdata2 = 0;
int myGENdata1 = 0;
int myGENdata2 = 0;
int mySNESdata1 = 0;
int mySNESdata2 = 0;
int tmpPress   = 0;
int  k         = 0;
volatile int mode1 = 0;
volatile int mode2 = 0;
int press1     = 0;
int press2     = 0;


void setup()
```

```
{
        UCSR0B = 0;                             //Sets TX and RX low//
        usart_init(9600, 16000000 );
        //Serial.begin(115200);

        DDRB   &= ~(1<<5);                      //Set Genesis D5 input
        DDRB   &= ~(1<<4);                      //Set Genesis D4 input
        DDRB   &= ~(1<<3);                      //Set Genesis D3 input
        DDRB   &= ~(1<<2);                      //Set Genesis D2 input
        DDRB   &= ~(1<<1);                      //Set Genesis D1 input
        DDRB   &= ~(1<<0);                      //Set Genesis D0 input

        DDRD   |=  (1<<7);                      //Set for Gamepad2 Select
        DDRD   |=  (1<<6);                      //Set for Gamepad1 Select
        DDRD   |=  (1<<5);                      //Set for Genesis Controller Select
        DDRD   |=  (1<<4);                      //Set Clock for output
        DDRD   &= ~(1<<3);                      //Set Gamepad NES2 input
        DDRD   &= ~(1<<2);                      //Set Gamepad NES1 input

        DDRC   |=  (1<<5);                      //Set for 5V output
        DDRC   |=  (1<<4);                      //Set for 5V output
        DDRC   &= ~(1<<3);                      //Set Analog1  input
        DDRC   &= ~(1<<2);                      //Set Analog0  input
        DDRC   &= ~(1<<1);                      //Set Analog1  input
        DDRC   &= ~(1<<0);                      //Set Analog0  input

        PORTC   |= (1<<5);                      //Set high for 5V
        PORTC   |= (1<<4);                      //Set high for 5V
       delay(2000);

        for (uint8_t ind=0; ind<8; ind++) {
                      joyReport.axis[ind] = ind*1000;
        }
        for (uint8_t ind=0; ind<sizeof(joyReport.button); ind++) {
            joyReport.button[ind] = 0;
        }

        attachInterrupt(0, intLoop1, FALLING);
        attachInterrupt(1, intLoop2, FALLING);
        //detachInterrupt(0)
}
/* Turn each button on in sequence 1 - 40, then off 1 - 40
 * add values to each axis each loop
 */
void loop()
{
        mainLoop();
        padMode(mode1, 1);
        padMode(mode2, 2);
        button = 0;
        printUSART(myNESdata1,myNESdata2,myGENdata1,myGENdata2,mySNESdata1,mySNESdata2);
        //sendJoyReport(&joyReport);
        press1=0;
        press2=0;
        _delay_us(500);
}

void intLoop1()
{
  if(press1 == 0)
  {
    press1=1;
    if (mode1<2) {
    //_delay_ms(140);
    mode1++;
    }else{
    mode1 = 0;
    }
  }
}
```

```
void intLoop2()
{
    if(press2==0)
    {
      press2=1;
    if (mode2<2) {
    //_delay_ms(140);
    mode2++;
    }else{
    mode2 = 0;
    }
    }
}

void padMode(int mode, int padNUM)
{
    if(padNUM ==1)
    {
      if(mode==0)
      {
        button = setPAD(button,myNESdata1,8);
      }else if(mode==1)
      {
        button = setPAD(button,myGENdata1,11);
      }else if(mode==2)
      {
        button = setPAD(button,mySNESdata1,12);
      }
      button = 16;
    }
    if(padNUM ==2)
    {
      if(mode==0)
      {
        button = setPAD(button,myNESdata2,8);
      }else if(mode==1)
      {
        button = setPAD(button,myGENdata2,11);
      }else if(mode==2)
      {
        button = setPAD(button,mySNESdata2,12);
      }
    }
}

void mainLoop()
{
        myNESdata1 = getNES(6,0);

        PORTD |= (1<<5);              //Set GENESIS select high for Gamepad1
        PORTD |= (1<<7);
        _delay_ms(3);
        myGENdata1 = getGEN(6);

        mySNESdata1 = getSNES(6,3);

        myNESdata2 = getNES(7,1);

        PORTD &= ~(1<<5);             //Set GENESIS select low
        PORTD |= (1<<6);
        _delay_ms(2);
        myGENdata2 = getGEN(7);

        mySNESdata2 = getSNES(7,2);
}

int setPAD(int inp_butt,int data,int inp_num)
{
  int out_butt = inp_butt;
  int p;
  for(p=0;p<inp_num;p++)
```

```
  {
      tmpPress = getKbit(p,data);
      if (tmpPress) {
         setButton(&joyReport, out_butt);
      } else {
         clearButton(&joyReport, out_butt);
      }
      out_butt++;
  }
  return out_butt;
}

////////////////Gamepad Controller Code//////////////////////////


int getNES(int LATCH,int DTA)
{
      int NESdata = 0;
      int i = 0;
      if(!(PINC & (1<<DTA)))
        NESdata |= (1<<0);
      PORTD &= ~(1<<LATCH); //Set latch low
      for(i=1; i<8; i++)
      {
        PORTD |= (1<<CLOCK); //Set clock high
        _delay_us(1); //Necessary delay
        if(!(PINC & (1<<DTA)))
          NESdata |= (1 << i);
        PORTD &= ~(1<<CLOCK); //Set clock low
      }
      PORTD |= (1<<LATCH); //Set latch high
      return NESdata;

}

int getSNES(int LATCH,int DTA)
{
      int SNESdata = 0;
      int i = 0;
      if(!(PINC & (1<<DTA)))
        SNESdata |= (1<<0);
      PORTD &= ~(1<<LATCH);           //Set latch low
      for(i=1; i<12; i++)
      {
        PORTD |= (1<<CLOCK);          //Set clock high
        _delay_us(1);                 //Necessary delay
        if(!(PINC & (1<<DTA)))
          SNESdata |= (1 << i);
        PORTD &= ~(1<<CLOCK);         //Set clock low
      }
      PORTD |= (1<<LATCH);            //Set latch high
      return SNESdata;

}

int getGEN(int LATCH)
{
    int GENdata = 0;
    int i = 0;

    PORTD &= ~(1<<LATCH);             //Set latch low
    _delay_us(1);
    for(i=4; i<6; i++)//i=4
    {
        if(!(PINB & (1<<i)))
          GENdata |= (1 << (i-4));
    }

    PORTD |= (1<<LATCH);              //Set latch high
    for(i=0; i<6; i++)
    {
```

```
        if(!(PINB & (1<<i)))
          GENdata |= (1 << (i+2));
    }
    PORTD &= ~(1<<LATCH);             //Set latch low
    PORTD |= (1<<LATCH);              //Set latch high

    PORTD &= ~(1<<LATCH);             //Set latch low
    PORTD |= (1<<LATCH);              //Set latch high

    for(i=0; i<3; i++)//i<3
    {
        if(!(PINB & (1<<i)))
          GENdata |= (1 << (i+8));
    }

   // _delay_ms(20);
    return GENdata;
}


////////////////JOYSTICK CONTROLLER CODE/////////////////////////////

// Send an HID report to the USB interface
void sendJoyReport(struct joyReport_t *report)
{
#ifndef DEBUG
    Serial.write((uint8_t *)report, sizeof(joyReport_t));
#else
    // dump human readable output for debugging
    for (uint8_t ind=0; ind<NUM_AXES; ind++) {
    }
    Serial.println();
    for (uint8_t ind=0; ind<NUM_BUTTONS/8; ind++) {
    }
    Serial.println();
#endif
}

// turn a button on
void setButton(joyReport_t *joy, uint8_t button)
{
    uint8_t index = button/8;
    uint8_t bit = button - 8*index;
    joy->button[index] |= 1 << bit;
}

// turn a button off
void clearButton(joyReport_t *joy, uint8_t button)
{
    uint8_t index = button/8;
    uint8_t bit = button - 8*index;
    joy->button[index] &= ~(1 << bit);
}

////////////////DONT MESS WITH/////////////////////////////

int numToASC(int num)
{
       return num+48;
}

int getKbit(int k, int n)
{
       return ((n & ( 1 << k )) >> k) ;
}

void printUSART(int value1, int value2, int value3, int value4, int value5, int value6)
{
       int j = 0;
       int n = 8;
       int tmpValue=0;
```

```
        usart_send(10);                         //New Line
        usart_send(13);                         //Enter Start of line

        tmpValue = numToASC(mode1);
               usart_send(tmpValue);
               usart_send(32);                  //Space Character

        tmpValue = numToASC(mode2);
         usart_send(tmpValue);
               usart_send(32);          //Space Character

        for (j=0; j<8; j++)
        {
            tmpValue = getKbit(j,value1);
            tmpValue = numToASC(tmpValue);
            usart_send(tmpValue);
        }

         usart_send(32);                //Space Character

               for (j=0; j<8; j++)
        {
            tmpValue = getKbit(j,value2);
            tmpValue = numToASC(tmpValue);
            usart_send(tmpValue);
        }

               usart_send(32);          //Space Character

               for (j=0; j<11; j++)
        {
            tmpValue = getKbit(j,value3);
            tmpValue = numToASC(tmpValue);
            usart_send(tmpValue);
        }

        usart_send(32);                 //Space Character

         for (j=0; j<11; j++)
        {
            tmpValue = getKbit(j,value4);
            tmpValue = numToASC(tmpValue);
            usart_send(tmpValue);
        }

         usart_send(32);                //Space Character

         for (j=0; j<12; j++)
        {
            tmpValue = getKbit(j,value5);
            tmpValue = numToASC(tmpValue);
            usart_send(tmpValue);
        }

         usart_send(32);                //Space Character

         for (j=0; j<12; j++)
        {
            tmpValue = getKbit(j,value6);
            tmpValue = numToASC(tmpValue);
            usart_send(tmpValue);
        }

}

int ArraySize(int Array[])
{
  return (sizeof(Array)/sizeof(int));

}
```

```
void usart_init(uint16_t baudin, uint32_t clk_speedin)
{
        uint32_t ubrr = (clk_speedin/16UL)/baudin-1;
        UBRR0H = (unsigned char)(ubrr>>8);
        UBRR0L = (unsigned char)ubrr;
        /* Enable ssreceiver and transmitter */
        UCSR0B = (1<<RXEN0)|(1<<TXEN0);
        /* Set frame format: 8data, 1stop bit */
        UCSR0C = (0<<USBS0)|(3<<UCSZ00);
        UCSR0A &= ~(1<<U2X0);
}

void usart_send( uint8_t data )
{
        /* Wait for empty transmit buffer */
        while ( !( UCSR0A & (1<<UDRE0)) );
        /* Put data into buffer, sends the data */
        UDR0 = data;
}

uint8_t  usart_recv(void)
{
        while ( !(UCSR0A & (1<<RXC0)) )
        return UDR0;
}

uint8_t  usart_istheredata(void)
{
         return (UCSR0A & (1<<RXC0));
}
```