**ECE 323 PROJECT SPECIFICATION**

**Rattanai Sawaspanich**

**ECE 322 PROJECT SPECIFICATION**

**Revision History**

|  |  |  |  |
| --- | --- | --- | --- |
| Date | Time spent | Task | Contributor |
| Voltage Amplifier | | |  |
| 01/11/15 | **2 hr** | **- Understanding the concept of Voltage Amplifier**  **- Initialized a voltage amplifier design** | **Rattanai Sawaspanich** |
| 01/14/15 | **4 hr** | **- Designed the voltage amplifier in SPICE**  **- Adjusted the resistor to prevent upper/bottom clipping of signal** | **Rattanai Sawaspanich** |
| 01/15/15 | **3 hr** | **- Revise the resistors value to match the gain** | **Rattanai Sawaspanich** |
| Voltage Buffer & Power Amplifier | | |  |
| 01/16/15 | **3 hr** | **- Initialized a voltage buffer design**  **- Initialized a power amplifier design** | **Rattanai Sawaspanich** |
| 01/21/15 | **3 hr** | **- Revised voltage amplifier to prevent clipped output from the buffer**  **- Revised voltage buffer resistance** | **Rattanai Sawaspanich** |
| 01/22/15 | **3 hr** | **- Revised resistance values for power amplifier**  **- Breadboarded the entire circuit** | **Rattanai Sawaspanich** |
| 02/11/15 | **5 hr** | **- Revise the entire circuit (Gain was too low)**  **- Revised the entire circuit to lower a harmonic distortion**  **- Finalized the design** | **Rattanai**  **Sawaspanich** |
| 02/12/15  - 02/11/15 | **13 hr** | **- Prototyped the entire circuit**  **- Tested connection**  **- Fixed loose connection/change BJT**  **- Document the circuit** | **Rattanai Sawaspanich** |
| Project Improvement (Volume Control Circuit w/ LED Display) | | |  |
| 02/13/15 | **2 hr** | **- Purposed a project**  **- Initialized the project idea** | **Rattanai Sawaspanich** |
| 02/15/15 | **4 hr** | **- Find Wunderboard - Downloaded VMWare - Installed / Configured Virtual Machine Environment** | **Rattanai Sawaspanich** |
| 02/16/15 | **1.5 hr** | **- Restored VM image from CS162 - Recovered Wunderboard Develop Kit** | **Rattanai Sawaspanich** |
| Date | Time spent | **Task** | **Contributor** |
| Project Improvement (Volume Control Circuit w/ LED Display) {Continue} | | | |
| 02/17/15 | **3 hr** | **- Reconfigured Wunderboard boot loader**  **- Edited Wunderboard header file**  **- Reconfigured Wunderboard LED** | **Rattanai Sawaspanich** |
| 02/18/15 | **1.5 hr** | **- Search for Digital Potentiometer**  **- Ordered Digital Potentiometer**  **- Researched about SPI** | **Rattanai Sawaspanich** |
| 02/20/15 | **6 hr** | **- Tested if Wonderboard is working**  **- Initialized SPI software implementation**  **- Debugging SPI transmission (still did not work)** | **Rattanai Sawaspanich** |
| 02/27/15 | **5 hr** | **- Revised ports initialization**  **- Edited LED lighting**  **- Edited SPI code (Wunderboard now is able to send data)** | **Rattanai Sawaspanich** |
| 02/28/15 | **5 hr** | **- Re-initialized DDR and PORT setup**  **- Connected Wunderboard with Digital Potentiometer for the first time (Did not work, of course)** | **Rattanai Sawaspnaich** |
| 03/03/15 | **4 hr** | **- Re-initialized DDR and PORT setup**  **- Create PORTC, pin 7 to be SS output (flash: active high)**  **- Wunderboard and Digital Potentiometer was able to communicate**  **- Wunderboard LED had a glitching/ghosting issue** | **Rattanai Sawaspanich** |
| 03/04/15 | **5 hr** | **- Designed the Wunderboard LED**  **- Implemented a software for Wunderboard LED display**  **- Implemented button interface** | **Rattanai Sawaspanich** |
| 03/05/15 | **3 hr** | **- Create logarithmic scale resistance (Linear gain)**  **- Related button press with LED display, and circuit gain** | **Rattanai Sawaspanich** |

**Cost Analysis**

**Voltage Amplifier (per channel)**

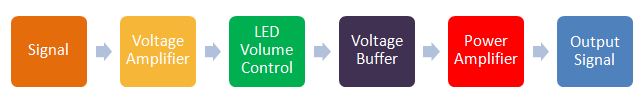
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Qty | Parts | Supplier | Part No. | Price (per unit) |
| 12 | **Resistors** | **Jameco** | **35991** | **0.01** |
| 4 | **2N4401 Transistor** | **Mouser** | **512-2N4401BU** | **0.23** |
| 1 | **2N4403 Transistor** | **Mouser** | **512-2N4403BU** | **0.23** |
| 5 | **470 uF Capacitor** | **IEEE** |  | **0.32** |
| 1 | **10K Potentiometer** | **IEEE** |  | **1** |
| 1 | **Protoboard** | **TekBot** |  | **5** |
|  |  |  | **Total** | **12.59** |

**LED Volume control**

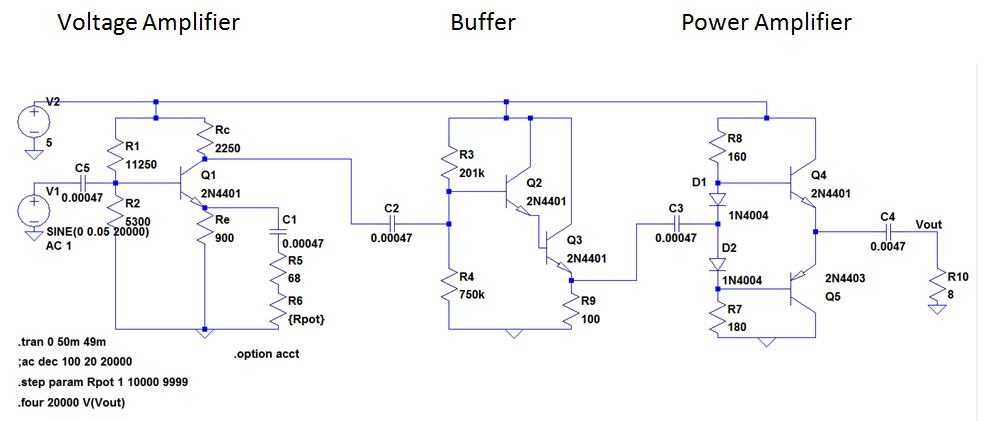
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Qty | Parts | Supplier | Part No. | Price (per unit) |
| 1 | **Wonderboard** | **TekBot** |  | **Free  ($60 New)** |
| 1 | **Digital Potentiometer** | **Newark** | **69K7607** | **1.34** |
|  |  |  | **Total** | **1.34** |

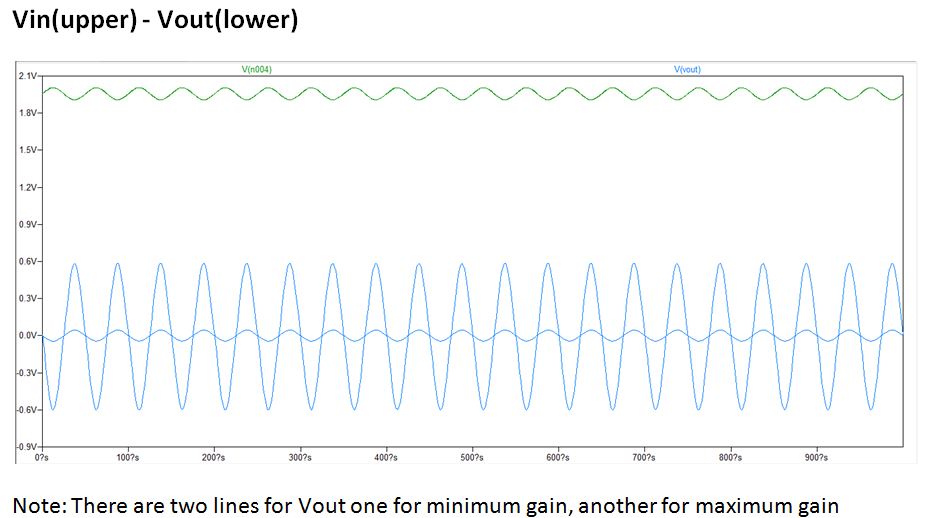
**HARDWARE: Design**

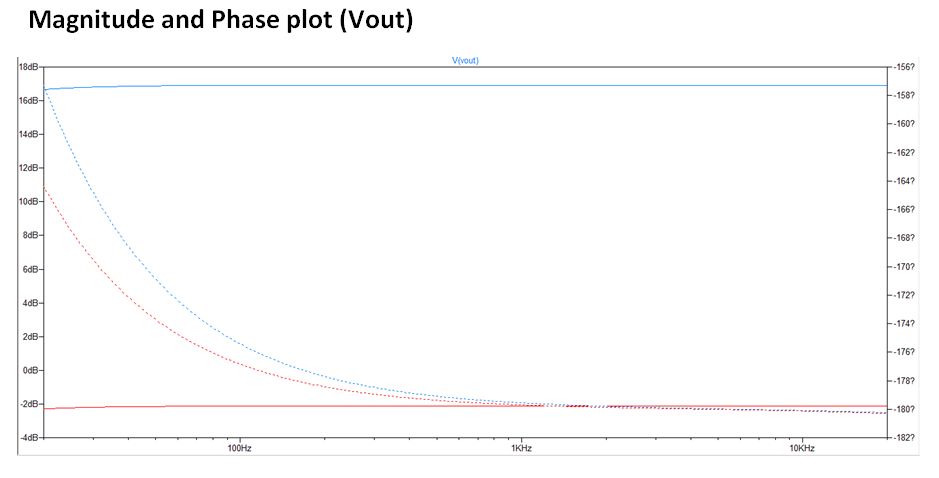
**Block Diagram**

****

|  |  |  |
| --- | --- | --- |
| Name | Input | Output |
| Voltage Amplifier | Voltage: 0 - 2 V Freq: 20 - 20k Hz Current: 1 mA | Voltage: 2.7 V ± 1V  Freq: 20 - 20k Hz  Current: 1 mA Resistance: 1k Ohm |
| LED Volume Control | Voltage: 3.3 V MOSI: 16 bits SCK: 30M Hz | Resistance:  0.99 - 10k Ohm Gain: 2 - 20 |
| Voltage Buffer | Voltage: 2.7 V ± 1V  Freq: 20 - 20k Hz  Current: 1 mA | Voltage: 2.5 V ± 1V  Freq: 20 - 20k Hz  Current: 1 mA Resistance: 100 Ohm |
| Power Amplifier | Voltage: 2.5 V ± 1V  Freq: 20 - 20k Hz  Current: 1 mA | Voltage: 2.5 V ± 1V  Freq: 20 - 20k Hz  Current: 0 - 100 mA |

**HARDWARE: Schematic**





(Please refer to previous lab reports for further mathematical analysis)

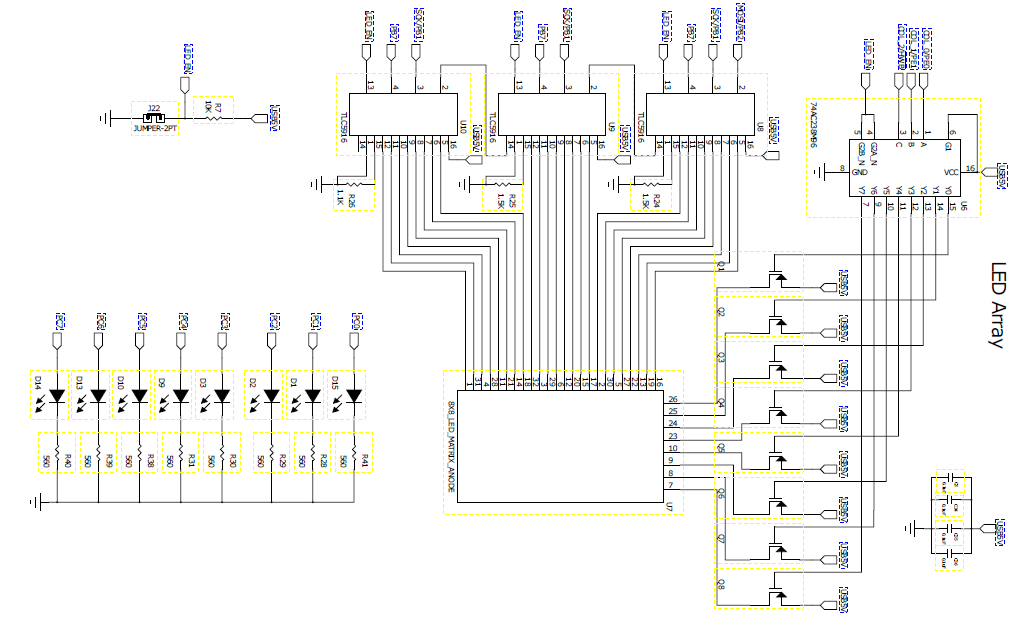
**Project Improvement: Volume Controller with an LED Display**

**Project Deliverable**My project improvement deliverable is a digital volume controller which displays a number representing how loud the speaker is in scale from 0 to 9. There are three parts to the circuit: Wunderboard, LED array, and a 10k Digital Potentiometer. Wunderboard, LED array, and a digital potentiometer are intercommunicated using SPI at 30MHz frequency. In the SPI set up, Wunderboard is a Master while the other two devices are slaves.

**Background Concept**

**Wunderboard**Wunderboard is an embedded device which is distributed and used in EECS162. The board uses atmel AT90USB128x/64x featuring an LED Display (8x8 RGB), USB Ports (USB Programming and USB UART), and Audio Output. In the improvement project, I use only a small number of features that AT90USB can do. Particularly, 8x8 LED display, SPI serial communication, programmable microcontroller.

Programming   
Wunderboard uses C as a programming language. To program Wunderboard is like to program any other embedded devices, the programmer needs to read the microcontroller datasheet to figure out how does the hardware relates to the software. Most of the time, a function or a variable you need will be there.

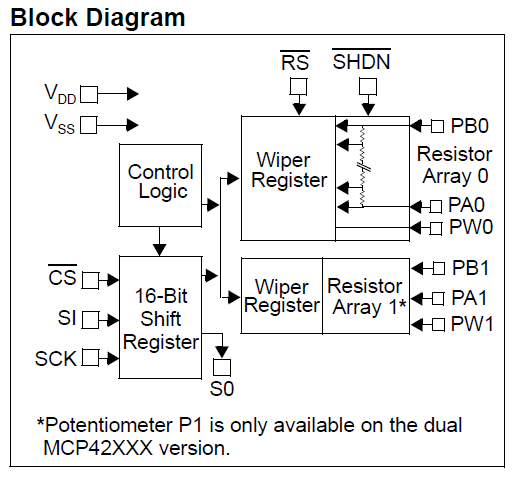
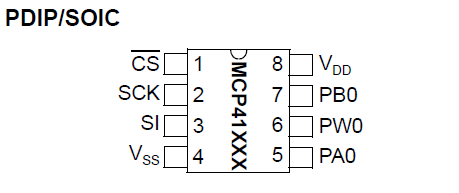
LED Display  
Wunderboard has a built in feature of an 8x8 RGB LED, communicates via wired SPI. There are four registors controlling the LED array: one for red LED, one for blue LED, one for Green, and another one to control the location of the LED and its color. The RGB LED array is hard wired to PortB on Wunderboad. Given that PortB is also a hard wired SPI communication port, the program has to make sure that it sends the right data, at the right time to PortB. The program has to send the LED display signal when the LED display is on and send the command signal when the digital potentiometer is on. Both through PortB.  
(Please refer to Wunderboard schematic in Appendix A for further detail)

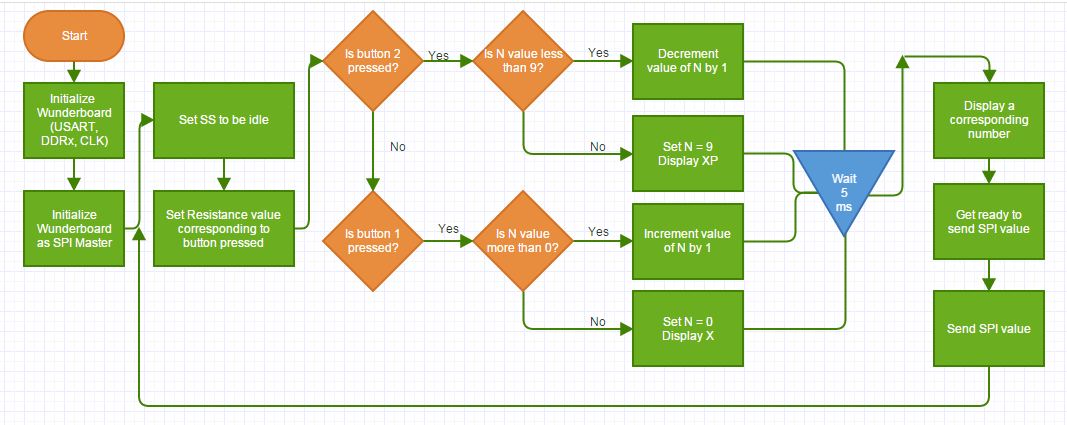
**SPI**Serial Peripheral Interface (SPI) is a synchronous serial data protocol used by microcontrollers for communicating with one or more peripheral devices quickly over short distance. It can also be used for communication between two microcontroller.  
(Source: http://arduino.cc/en/Reference/SPI )

The main four common lines for SPI are  
**MISO (Master In Slave Out)** - A line for a slave to send a data to master  
**MOSI (Master Out Slave In) -** A line for the master to send a data to a slave  
**SCK (Serial Clock) -** A standard clock pulse that each device agree upon.   
 (Usually SCK is generated by master)   
**SS/CS (Slave Select OR Chip Select)** - A line for master to enable a slave device.   
 When SS is high, a slave ignore the master.  
 When the SS is low, a slave commute with master.

**Digital Potentiometer**

The digital potentiometer that is used in this project improvement is MCP41010 manufactured by Microchip. MCP41010 is a 256 position, digital potentiometer range from 0 to 10kOhms. The digital potentiometer is a single channel device and is manufactured in an 8-pin PDIP/SOIC package. The wiper position of the digital potentiometer varies linearly and is controlled via SPI interface (mode 0,0 or 1,1). During the operation, the potentiometer consume less than 1 µA. A pot has an extra featured called "shutdown mode." The potentiometer disconnects Terminal A from the resistor stack and simultaneously connects the wiper to Terminal B. When the chip is powered up, the wiper is set to the mid-scale position results in resistance of 5kOhms. The RS (reset) pin implemnts a hardware reset and also returns the wiper to mid-scale. MCP41010 operates from a single 2.7 - 5.5V supply.



**Software: Flow Chart**

**Function Interfaces**

|  |  |  |  |
| --- | --- | --- | --- |
| Function Name | Input | Output | Description |
| CPU\_PRESCALE(int) | Prescale | - | Sync Wunderboard clock with the CPU |
| USART\_init(int) | BaudRate | - | Initialize USART BaudRate |
| USART\_send\_string(char\*) | String | - | Send an input string to other connected dev |
| setup\_ADC(int, int) | Prescale,  FreeRunning | - | Initialize accelerometer |
| Get\_Byte\_USART() | - | Transferred character | Receive a character (at a time) from USART communication |
| spi\_init\_master() | - | - | Initialize Wunderboard as an SPI Master |
| spi\_send\_ss(int) | Pin | {1,0} | Send a Chip Selected / Slave Select signal to a given pin in PortC (Pin0:7) |
| spi\_trans\_master(char\*) | Data | SPI\_data | Send two consecutive 8-bits information in a row, then call spi\_send\_ss() to flash the slave device |
| init\_portB\_led() | - | - | Initialize Wunderboard PortB to be ready to communicate with LED array |
| disp\_gN() | - | - | Display a green number N on an LED display |
| disp\_xP() | - | - | Display a sad face on an LED display |
| disp\_x() | - | - | Display a red X on an LED display |
| led\_off() | - | - | Disable SS on the LED display |
| led\_blue(int, int) | Row, Col | - | Display a single blue LED at location specified by Row and Col (0 <= Row <8 , 0<= Col < 8) |
| led\_red(int, int) | Row, Col | - | Display a red LED at a specified location |
| led\_green(int, int) | Row, Col | - | Display a green LED at a specified location |

**Appendix A: Wunderboard Schematic**

**Appendix B:   
Useful section of MCP41XXX Datasheet (Digital Potentiometer)  
 1. Characteristics Page 1-2   
 2. Detailed Serial Interface Timing Page 6   
 3. Performance Curves Page 7   
 4. Pin Descriptions Page 12   
 5. Mode of Operation Page 14   
 6. SPI Format Page 18**

**Appendix C: C Source Code**

/\*\*

\* **@file** wunder.c

\* **@author** Dan Albert, Marhshal Horn (Wunderboard Loader)

\* **@author** Rattanai Sawaspanich (SPI setup, and Digital Potentiometer ctrl)

\* **@date** Created 12/15/2010

\* **@date** Last updated 1/26/2015

\* **@version** 1.0

\*

\* **@section** LICENSE

\*

\* This program is free software; you can redistribute it and/or modify

\* it under the terms of the GNU General Public License as published by

\* the Free Software Foundation; either version 2 of the License, or

\* (at your option) any later version.

\*

\* This program is distributed in the hope that it will be useful,

\* but WITHOUT ANY WARRANTY; without even the implied warranty of

\* MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the

\* GNU General Public License for more details.

\*

\* You should have received a copy of the GNU General Public License

\* along with this program; if not, write to the Free Software

\* Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston,

\* MA 02110-1301, USA.

\*

\* **@section** DESCRIPTION

\*

\* This program will test the various components of the Wunderboard when used

\* with the corresponding host test script.

\*

\*/

#include <util/delay.h>

#include <avr/interrupt.h>

#include <avr/io.h>

#include <avr/pgmspace.h>

#include <stdint.h>

#include <stdlib.h>

#include <string.h>

#include "adc.h"

#include "diskio.h"

#include "types.h"

#include "usart.h"

#include "leds.h"

// 9600 baud

#define BAUD\_RATE 51

#define CPU\_PRESCALE(n) (CLKPR = 0x80, CLKPR = (n))

#define SS PB0

#define SCK PB1

#define MOSI PB2

void initialize**(** void **)**

**{**

CPU\_PRESCALE**(**0**);**

USART\_init**(**BAUD\_RATE**);**

USART\_transmit**(**'\f'**);** // Send form feed to clear the terminal.

USART\_send\_string**(**"WunderBoard initializing...\r\n"**);**

USART\_send\_string**(**"\tSetting ADC prescaler and disabling free running mode...\r\n"**);**

setup\_ADC**(**ADC\_PRESCALER\_32**,** FALSE**);**

USART\_send\_string**(**"\tEnabling ADC...\r\n"**);**

ADC\_enable**();**

USART\_send\_string**(**"\tSetting ADC reference to Vcc...\r\n"**);**

ADC\_set\_reference**(**ADC\_REF\_VCC**);**

// Configure IO //

USART\_send\_string**(**"\tConfiguring IO...\r\n"**);**

//DDRx corresponds to PORTx/PINx, dependng on direction of data flow -- PORT for output, PIN for input

DDRA **=** 0x00**;** // Buttons and switches

DDRB **=** 0xE7**;** // Red enable, green enable and audio out

DDRC **=** 0xff**;** // Discrete LEDs

DDRE **=** 0x47**;** // LED Column

DDRF **=** 0x00**;** // Accelerometer

// Disable pullups and set outputs low //

PORTA **=** 0x00**;**

PORTB **=** 0x01**;**

PORTC **=** 0x81**;**

PORTE **=** 0x00**;**

PORTF **=** 0x00**;**

//Set OC1A to toggle

TCCR1A **=** 0b01000000**;**

// Clk/64 and CTC mode

TCCR1B **=** 0b00001011**;**

OCR1A **=** 24**;**

USART\_send\_string**(**"\tSetting SPI\r\n"**);**

//Set the SPI bus appropriately to use the LED array

SPCR **=** **(**1**<<**SPE**)|(**1**<<**MSTR**)|(**1**<<**SPR0**);**

sei**();**

**}**

/\*\*\*\*\*\*\*\*\*\*\*\*\*\* MAIN \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

unsigned char Get\_Byte\_USART**()**

**{**

**if(**UCSR1A **&** **(**1**<<**RXC1**))**

**return** **(**UDR1**);**

**else**

**return** **(**unsigned char**)** 0**;**

**}**

ISR**(**USART1\_RX\_vect**)**

**{**

//Thy shall do nothing

**}**

ISR**(**BADISR\_vect**){**

//DO NOTHING

**}**

unsigned char initialize\_TIMER0**()**

**{**

//clean up a bit

TCCR0A **&=** **~**0b11**;**

TCCR0B **&=** **~**0b111**;**

/\* Set the CTC mode \*/

TCCR0A **|=** 0b00000010**;**//(1<<WGMO1);

/\* Set the Clock Frequency \*/

TCCR0B **|=** 0b00000101**;**//(101<<CS00);

/\* Set initial count value \*/

OCR0A **=** 0**;**

**return** 0**;**

**}**

// This function checks if TIMER0 has elapsed.

unsigned char check\_TIMER0**()**

**{**

**if(**TIFR0 **&** **(**1**<<**OCF0A**)** **!=** 0**){**

TIFR0 **=** **(**1**<<**OCF0A**);**

**return** 1**;**

**}else{**

**return** 0**;**

**}**

**}**

///////////////// SPI //////////////////

/\* Function: spi\_init\_master

\* Initialize Wuderboard as a Master in SPI

\* SPI Output: PortB

\* Pin0 -- Do nothing

\* Pin1 -- SCK

\* Pin2 -- MOSI (square)

\* Pin3 -- small detail MOSI

\* Pin4 -- Do nothing (MOSI really really small)

\* Pin5 -- Wunderboard clk

\* Pin6 -- Do nothing

\* Pin7 -- Do nothing

\*/

void spi\_init\_master**(){**

/\*Look at these bits to set the SPI config

\* SPDR -- SPI Data Register

\*/

//Setting the PORTA to be the output (PIN1,2,3,5)

//NOTE: Write an (int) 1 to a DDRx sets it on the output mode

// DDRB |= (3<<1)|(1<<5);

DDRB **|=** **(**1**<<**PB1**)** **|** **(**1**<<**PB2**)** **|** **(**1**<<**5**)** **|** **(**1**<<**7**);**

//Look at the datasheet.(P.174, AT90USB64/128)

/\*Setting SPCR

\* SPE -- SPI\_en -- (1:Enable, 0:Nope)

\* DORD -- Data order -- (1:LSB first, 0:MSB first)

\* MSTR -- Master/Slave? -- (1:Master, 0:Slave)

\* CPOL -- Clock polarity -- (1:Rising, 0:Falling comparing to clk)

\* CPHA -- Clock phase -- (Sampling at 0:Leading 1:Trailing)

\* SPR1/0 -- Clock scaling

\*/

//Setting SPCR s.t.

//SPCR = Enable SPI | LSB leads| Master Mode|

//Rising edge sck| lead edge sampling | (Freq = clk/4)

//NOTE: The actual Freq = clk/2 (b/c doubling the trans rate)

SPCR **=** 1**<<**SPE **|** 0**<<**DORD **|** 1**<<**MSTR **|**

0**<<**CPOL **|** 0**<<**CPHA **|** **(**0**<<**SPR1 **|** 0**<<**SPR0**);**

//Enable the double the transmittion rate in SPSR (write 1 to it)

SPSR **=** **(**1**<<**SPI2X**);**

**return** **;**

**}**

void spi\_send\_ss**(**int bit\_num**){**

//Toggle SPI PortC to flash the digital pot

PORTC **|=** **(**1**<<**bit\_num**);**

PORTC **&=** **~(**1**<<**bit\_num**);**

**return;**

**}**

void spi\_trans\_master**(**char **\***data**){**

**while(!(**SPSR **&** **(**1**<<**SPIF**)));**

//Write a message to SPDR, ready to be sent

SPDR **=** data**[**0**];**

//Wait until the SPSR is empty, then send the data

//Check SPIF bit in SPSR

**while(!(**SPSR **&** **(**1**<<**SPIF**)));**

SPDR **=** data**[**1**];**

// while(!(SPSR & (1<<SPIF)));

spi\_send\_ss**(**7**);**

**}**

////////////// SPI ENDS HERE /////////////

///////////// LED CONTROL /////////////

void init\_portB\_led**(){**

DDRB **=** 0xE7**;** // Red enable, green enable and audio out

DDRC **=** 0x00**;** //Disable SS

PORTB **=** 0x01**;** //Setting PORTB pin 1 to be an output

**}**

///////////////// LED ///////////////////

//Displaying '0'

void disp\_g0**(){**

init\_portB\_led**();**

led\_green**(**1**,**3**);**

led\_off**();**

led\_green**(**1**,**4**);**

led\_off**();**

led\_green**(**1**,**5**);**

led\_off**();**

led\_green**(**2**,**3**);**

led\_off**();**

led\_green**(**3**,**3**);**

led\_off**();**

led\_green**(**4**,**3**);**

led\_off**();**

led\_green**(**2**,**5**);**

led\_off**();**

led\_green**(**3**,**5**);**

led\_off**();**

led\_green**(**4**,**5**);**

led\_off**();**

led\_green**(**5**,**3**);**

led\_off**();**

led\_green**(**5**,**4**);**

led\_off**();**

led\_green**(**5**,**5**);**

led\_off**();**

//ReInite the PortC to SPI mode

DDRC **=** 0xFF**;**

**}**

//Displaying '1'

void disp\_g1**(){**

init\_portB\_led**();**

**for(**int i **=** 1**;** i**<**6**;++**i**){**

led\_green**(**i**,**4**);**

led\_off**();**

**}**

//ReInite the PortC to SPI mode

DDRC **=** 0xFF**;**

**}**

//Displaying '2'

void disp\_g2**(){**

init\_portB\_led**();**

led\_green**(**1**,**3**);**

led\_off**();**

led\_green**(**1**,**4**);**

led\_off**();**

led\_green**(**1**,**5**);**

led\_off**();**

led\_green**(**2**,**5**);**

led\_off**();**

led\_green**(**3**,**5**);**

led\_off**();**

led\_green**(**3**,**4**);**

led\_off**();**

led\_green**(**3**,**3**);**

led\_off**();**

led\_green**(**4**,**3**);**

led\_off**();**

led\_green**(**5**,**3**);**

led\_off**();**

led\_green**(**5**,**4**);**

led\_off**();**

led\_green**(**5**,**5**);**

led\_off**();**

//ReInite the PortC to SPI mode

DDRC **=** 0xFF**;**

**}**

//Displaying '3'

void disp\_g3**(){**

init\_portB\_led**();**

led\_green**(**1**,**3**);**

led\_off**();**

led\_green**(**1**,**4**);**

led\_off**();**

led\_green**(**1**,**5**);**

led\_off**();**

led\_green**(**2**,**5**);**

led\_off**();**

led\_green**(**3**,**5**);**

led\_off**();**

led\_green**(**3**,**4**);**

led\_off**();**

led\_green**(**3**,**3**);**

led\_off**();**

led\_green**(**4**,**5**);**

led\_off**();**

led\_green**(**5**,**3**);**

led\_off**();**

led\_green**(**5**,**4**);**

led\_off**();**

led\_green**(**5**,**5**);**

led\_off**();**

//ReInite the PortC to SPI mode

DDRC **=** 0xFF**;**

**}**

//Displaying '4'

void disp\_g4**(){**

init\_portB\_led**();**

led\_green**(**1**,**3**);**

led\_off**();**

led\_green**(**1**,**5**);**

led\_off**();**

led\_green**(**2**,**3**);**

led\_off**();**

led\_green**(**2**,**5**);**

led\_off**();**

led\_green**(**3**,**5**);**

led\_off**();**

led\_green**(**3**,**4**);**

led\_off**();**

led\_green**(**3**,**3**);**

led\_off**();**

led\_green**(**4**,**5**);**

led\_off**();**

led\_green**(**5**,**5**);**

led\_off**();**

//ReInite the PortC to SPI mode

DDRC **=** 0xFF**;**

**}**

//Displaying '5'

void disp\_g5**(){**

init\_portB\_led**();**

led\_green**(**1**,**3**);**

led\_off**();**

led\_green**(**1**,**4**);**

led\_off**();**

led\_green**(**1**,**5**);**

led\_off**();**

led\_green**(**2**,**3**);**

led\_off**();**

led\_green**(**3**,**5**);**

led\_off**();**

led\_green**(**3**,**4**);**

led\_off**();**

led\_green**(**3**,**3**);**

led\_off**();**

led\_green**(**4**,**5**);**

led\_off**();**

led\_green**(**5**,**3**);**

led\_off**();**

led\_green**(**5**,**4**);**

led\_off**();**

led\_green**(**5**,**5**);**

led\_off**();**

//ReInite the PortC to SPI mode

DDRC **=** 0xFF**;**

**}**

//Displaying '6'

void disp\_g6**(){**

init\_portB\_led**();**

led\_green**(**1**,**3**);**

led\_off**();**

led\_green**(**1**,**4**);**

led\_off**();**

led\_green**(**1**,**5**);**

led\_off**();**

led\_green**(**2**,**3**);**

led\_off**();**

led\_green**(**3**,**5**);**

led\_off**();**

led\_green**(**3**,**4**);**

led\_off**();**

led\_green**(**3**,**3**);**

led\_off**();**

led\_green**(**4**,**3**);**

led\_off**();**

led\_green**(**4**,**5**);**

led\_off**();**

led\_green**(**5**,**3**);**

led\_off**();**

led\_green**(**5**,**4**);**

led\_off**();**

led\_green**(**5**,**5**);**

led\_off**();**

//ReInite the PortC to SPI mode

DDRC **=** 0xFF**;**

**}**

//Displaying '7'

void disp\_g7**(){**

init\_portB\_led**();**

**for(**int i **=** 3**;** i**<**6**;** **++**i**){**

led\_green**(**1**,**i**);**

led\_off**();**

**}**

**for(**int i **=** 1**;** i**<**6**;** **++**i**){**

led\_green**(**i**,**5**);**

led\_off**();**

**}**

//ReInite the PortC to SPI mode

DDRC **=** 0xFF**;**

**}**

//Displaying '8'

void disp\_g8**(){**

init\_portB\_led**();**

led\_green**(**1**,**3**);**

led\_off**();**

led\_green**(**1**,**4**);**

led\_off**();**

led\_green**(**1**,**5**);**

led\_off**();**

led\_green**(**2**,**3**);**

led\_off**();**

led\_green**(**2**,**5**);**

led\_off**();**

led\_green**(**3**,**5**);**

led\_off**();**

led\_green**(**3**,**4**);**

led\_off**();**

led\_green**(**3**,**3**);**

led\_off**();**

led\_green**(**4**,**3**);**

led\_off**();**

led\_green**(**4**,**5**);**

led\_off**();**

led\_green**(**5**,**3**);**

led\_off**();**

led\_green**(**5**,**4**);**

led\_off**();**

led\_green**(**5**,**5**);**

led\_off**();**

//ReInite the PortC to SPI mode

DDRC **=** 0xFF**;**

**}**

//Displaying '9'

void disp\_g9**(){**

init\_portB\_led**();**

led\_green**(**1**,**3**);**

led\_off**();**

led\_green**(**1**,**4**);**

led\_off**();**

led\_green**(**1**,**5**);**

led\_off**();**

led\_green**(**2**,**3**);**

led\_off**();**

led\_green**(**2**,**5**);**

led\_off**();**

led\_green**(**3**,**5**);**

led\_off**();**

led\_green**(**3**,**4**);**

led\_off**();**

led\_green**(**3**,**3**);**

led\_off**();**

led\_green**(**4**,**5**);**

led\_off**();**

led\_green**(**5**,**5**);**

led\_off**();**

//ReInite the PortC to SPI mode

DDRC **=** 0xFF**;**

**}**

//Displaying 'XP'

void disp\_xP**(){**

init\_portB\_led**();**

led\_red**(**2**,**1**);**

led\_off**();**

led\_red**(**2**,**2**);**

led\_off**();**

led\_red**(**2**,**5**);**

led\_off**();**

led\_red**(**2**,**6**);**

led\_off**();**

led\_red**(**3**,**1**);**

led\_off**();**

led\_red**(**3**,**2**);**

led\_off**();**

led\_red**(**3**,**5**);**

led\_off**();**

led\_red**(**3**,**6**);**

led\_off**();**

**for(**int i **=** 2**;** i**<**6**;++**i**){**

led\_red**(**5**,**i**);**

led\_off**();**

**}**

led\_red**(**6**,**1**);**

led\_off**();**

led\_red**(**6**,**6**);**

led\_off**();**

led\_blue**(**4**,**6**);**

led\_off**();**

//ReInite the PortC to SPI mode

DDRC **=** 0xFF**;**

**}**

//Displaying 'X'

void disp\_x**(){**

init\_portB\_led**();**

**for(**int i **=** 0**;** i**<**8**;++**i**){**

led\_red**(**i**,**i**);**

led\_red**(**7**-**i**,**7**-**i**);**

led\_off**();**

**}**

//ReInite the PortC to SPI mode

DDRC **=** 0xFF**;**

**}**

///////////// The Main ////////////////

int main**(**int argc**,** char **\*\***argv**)**

**{**

initialize**();**

spi\_init\_master**();**

clear\_array**();**

// INFO info = {0b0001, 0b0001, 0b11000010 };

/\*Defined variables:

\* SS --> PB0

\* SCK --> PB1

\* MOSI --> PB2

\*/

srand**(**17**);**

char misc **=** 0b00010001**;**

char r\_val **=** 125**;**

char arr**[**2**]** **=** **{** misc**,** r\_val**};**

int N **=**5**;**

//R\_spi change the resistor value to behave in log scale

char r\_spi**[**10**]** **=** **{**255**,**254**,**253**,**251**,**248**,**245**,**238**,**211**,**155**,**0**};**

**while(**1**){**

//Use buttons to adjust the resistor

//Change PORTC to be active low

//(Idle = low; done = high)

//Starting PORTC Pin7 as a high (For active high)

// PORTC = 0x80;

//Second try, PORTC Pin7 start as a low (Active Low)

PORTC **=** 0x00**;**

r\_val **=** r\_spi**[**9**-**N**];**

arr**[**1**]** **=** r\_val**;**

//ACTIVE LOW!! PRESSING UP mean goin down in POT

**if(**PINA **&** 0b00000100 **){**

**if(**N **<** 9**)**

**++**N**;**

**else{**

N **=** 9**;**

disp\_xP**();**

**}**

PORTC **|=** **(**1**<<**0**);**

\_delay\_ms**(**5**);**

**}**

**else** **if(**PINA **&** 0b00000010**){**

**if(**N **>** 0**)**

**--**N**;**

**else{**

N **=** 0**;**

disp\_x**();**

**}**

PORTC **|=** **(**1**<<**0**);**

\_delay\_ms**(**5**);**

**}**

**if(**N**==**0**)**

disp\_g0**();**

**else** **if** **(**N**==**1**)**

disp\_g1**();**

**else** **if** **(**N**==**2**)**

disp\_g2**();**

**else** **if** **(**N**==**3**)**

disp\_g3**();**

**else** **if** **(**N**==**4**)**

disp\_g4**();**

**else** **if** **(**N**==**5**)**

disp\_g5**();**

**else** **if** **(**N**==**6**)**

disp\_g6**();**

**else** **if** **(**N**==**7**)**

disp\_g7**();**

**else** **if** **(**N**==**8**)**

disp\_g8**();**

**else** **if** **(**N**==**9**)**

disp\_g9**();**

**else** **if** **(**N **<** 0**)**

disp\_x**();**

**else**

disp\_xP**();**

//Reinitiateion PORTB to be on the SPI transmittion mode

//DDRB |= (1<<PB1) | (1<<PB2) | (1<<5) | (1<<7);

DDRB **|=** **(**1**<<**PB1**)** **|** **(**1**<<**PB2**)** **|** **(**1**<<**5**)** **|** **(**0**<<**7**);**

//Reinite PORTC to be the SPI out

DDRC **=** 0xFF**;**

//Send the signal

//The first part, 8bits, is the command code

//The second part, 8bits, is the resistor value

spi\_trans\_master**(**arr**);**

**}**

**}**

#include "leds.h"

#ifndef LED\_LATCH

#define LED\_LATCH (1 << 7)

#endif

/\*

Strobing the LED latch:

The LED latch is bit 7 of PORTB. Strobing it means that you are turning the bit on, then

turning the bit off.

\*/

void set\_array\_red**(**unsigned char rows**)**

**{**

SPDR **=** rows**;**

**while(!(**SPSR **&** **(**1 **<<** SPIF**)));**

SPDR **=** 0x00**;**

**while(!(**SPSR **&** **(**1 **<<** SPIF**)));**

SPDR **=** 0x00**;**

**while(!(**SPSR **&** **(**1 **<<** SPIF**)));**

PORTB **|=** LED\_LATCH**;**

PORTB **&=** **~**LED\_LATCH**;**

**}**

void set\_array\_green**(**unsigned char rows**)**

**{**

//write to SPDR

//check SPSR register, see datasheet for bit

//Strobe LED Latch

SPDR **=** 0x00**;**

**while(!(**SPSR **&** **(**1 **<<** SPIF**)));**

SPDR **=** rows**;**

**while(!(**SPSR **&** **(**1 **<<** SPIF**)));**

SPDR **=** 0x00**;**

**while(!(**SPSR **&** **(**1 **<<** SPIF**)));**

PORTB **|=** LED\_LATCH**;**

PORTB **&=** **~**LED\_LATCH**;**

**return;**

**}**

void set\_array\_blue**(**unsigned char rows**)**

**{**

SPDR **=** 0x00**;**

**while(!(**SPSR **&** **(**1 **<<** SPIF**)));**

SPDR **=** 0x00**;**

**while(!(**SPSR **&** **(**1 **<<** SPIF**)));**

SPDR **=** rows**;**

**while(!(**SPSR **&** **(**1 **<<** SPIF**)));**

PORTB **|=** LED\_LATCH**;**

PORTB **&=** **~**LED\_LATCH**;**

**}**

void set\_array\_yellow**(**unsigned char rows**)**

**{**

SPDR **=** rows**;**

**while(!(**SPSR **&** **(**1 **<<** SPIF**)));**

SPDR **=** rows**;**

**while(!(**SPSR **&** **(**1 **<<** SPIF**)));**

SPDR **=** 0x00**;**

**while(!(**SPSR **&** **(**1 **<<** SPIF**)));**

PORTB **|=** LED\_LATCH**;**

PORTB **&=** **~**LED\_LATCH**;**

**}**

void set\_array\_purple**(**unsigned char rows**)**

**{**

SPDR **=** rows**;**

**while(!(**SPSR **&** **(**1 **<<** SPIF**)));**

SPDR **=** 0x00**;**

**while(!(**SPSR **&** **(**1 **<<** SPIF**)));**

SPDR **=** rows**;**

**while(!(**SPSR **&** **(**1 **<<** SPIF**)));**

PORTB **|=** LED\_LATCH**;**

PORTB **&=** **~**LED\_LATCH**;**

**}**

void set\_array\_gb**(**unsigned char rows**)**

**{**

SPDR **=** 0x00**;**

**while(!(**SPSR **&** **(**1 **<<** SPIF**)));**

SPDR **=** rows**;**

**while(!(**SPSR **&** **(**1 **<<** SPIF**)));**

SPDR **=** rows**;**

**while(!(**SPSR **&** **(**1 **<<** SPIF**)));**

PORTB **|=** LED\_LATCH**;**

PORTB **&=** **~**LED\_LATCH**;**

**}**

void clear\_array**()**

**{**

update\_row**(** 0**,** 0**,** 0**);**

**}**

void update\_row**(** uint8\_t red**,** uint8\_t green**,** uint8\_t blue **)**

**{**

SPDR **=** red**;**

**while(!(**SPSR **&** **(**1 **<<** SPIF**)));**

SPDR **=** green**;**

**while(!(**SPSR **&** **(**1 **<<** SPIF**)));**

SPDR **=** blue**;**

**while(!(**SPSR **&** **(**1 **<<** SPIF**)));**

PORTB **|=** LED\_LATCH**;**

PORTB **&=** **~**LED\_LATCH**;**

**}**

void led\_red**(** uint8\_t x**,** uint8\_t y **)**

**{**

PORTE **=** **((**7 **-** **(**y **&** 0x07**))** **|** **(**PORTE **&** 0xF8**));**

set\_array\_red**((**1 **<<** 7**)** **>>** **(**x **&** 0x07**));**

/\* Let's index this from the top left, like a normal screen.

\* However, the LED array is indexed in hardware from the bottom right,

\* with the bit position controlling y, and PORTE controlling x.

\*/

//to reverse the row, you can use (7 - (row & 0x07)) | (PORTE & 0xF8)

**}**

void led\_green**(** uint8\_t x**,** uint8\_t y **)**

**{**

PORTE **=** **((**7 **-** **(**y **&** 0x07**))** **|** **(**PORTE **&** 0xF8**));**

set\_array\_green**((**1 **<<** 7**)** **>>** **(**x **&** 0x07**));**

**}**

void led\_blue**(** uint8\_t x**,** uint8\_t y **)**

**{**

PORTE **=** **((**7 **-** **(**y **&** 0x07**))** **|** **(**PORTE **&** 0xF8**));**

set\_array\_blue**((**1 **<<** 7**)** **>>** **(**x **&** 0x07**));**

**}**

void led\_yellow**(** uint8\_t x**,** uint8\_t y **)**

**{**

PORTE **=** **((**7 **-** **(**y **&** 0x07**))** **|** **(**PORTE **&** 0xF8**));**

set\_array\_yellow**((**1 **<<** 7**)** **>>** **(**x **&** 0x07**));**

**}**

void led\_purple**(** uint8\_t x**,** uint8\_t y **)**

**{**

PORTE **=** **((**7 **-** **(**y **&** 0x07**))** **|** **(**PORTE **&** 0xF8**));**

set\_array\_purple**((**1 **<<** 7**)** **>>** **(**x **&** 0x07**));}**

void led\_gb**(** uint8\_t x**,** uint8\_t y **)**

**{**

PORTE **=** **((**7 **-** **(**y **&** 0x07**))** **|** **(**PORTE **&** 0xF8**));**

set\_array\_gb**((**1 **<<** 7**)** **>>** **(**x **&** 0x07**));**

**}**

void led\_off**(** void **)**

**{**

update\_row**(** 0**,** 0**,** 0**);**

**}**