JUMP TRADING SIMULATION INTERFACE

By

Jacob Clifton

David McKiernan

Ryan Schmid

Final Report for ECE 445, Senior Design, Fall 2012

TA: Rajarshi Roy

12 December 2012

Project No. 4

Abstract

Current medical simulations for training students in the medical field involve the use of a complex computer interface to control key parameters. The Jump Trading simulation interface is an ergonomic and easy to use user interface that has been designed for use in these medical simulations. The user interface contains various switches, dials, and buttons that are used to quickly modify the most important parameters of a simulation thus allowing for a more fluid and real-time simulation experience. This report documents the design, construction, verification, and cost of this device.

Contents

1. Introduction

Our project was in collaboration with OSF Healthcare and Jump Trading. OSF Healthcare is currently performing medical simulations intended on training students in the healthcare field, such as medical students, medical residents, and nurses. The simulations are designed to imitate real life events these students will encounter in their careers in the healthcare field, such as responding to victims of heart attacks, car crashes, and much more. As the students are immersed in the simulation, there is an operator behind the scenes who is changing various parameters of the simulation based on the actions the students are taking. The problem is that the computer interface the operator is using is complex and cumbersome, so the user is spending a great deal of time navigating through menus to find and modify the desired parameter. This distracts the operator from being able to pay full attention to student performance and thus degrades the simulation experience.

1.1 Purpose

The purpose of this project was to create a new user interface to use with OSF Healthcare's medical simulations. Our goal was to create a device that could control the most important aspects of the simulation as fast as possible. To achieve this goal, we were to implement an ergonomic and easy to use design. We wanted a device that could fit into one's hand so that one could modify parameters easily. The device needed to be low cost and be able to interface with the Laerdal software. Laerdal software is the software that is currently being used in these medical simulations. Our device is not designed to replace Laerdal software; rather, it is a means of controlling the most important parameters within Laerdal's software to create a better simulation experience.

1.2 Functions

Our device can select and modify twelve of the most important parameters in the simulations using the components on the user interface. The parameters are displayed on an LCD in real-time. The device also includes two buttons for execution of macros and could easily be changed for other functionality. Data is sent through the USB interface where it is interpreted by software on the computer. The software also displays the parameters and their values. Functionality exists to upload new code in order to modify the current functions of the device. New code can be uploaded easily if more functionality were added to the device in the future.

1.3 Blocks

The block diagram for the user interface is shown in Figure 1.1.

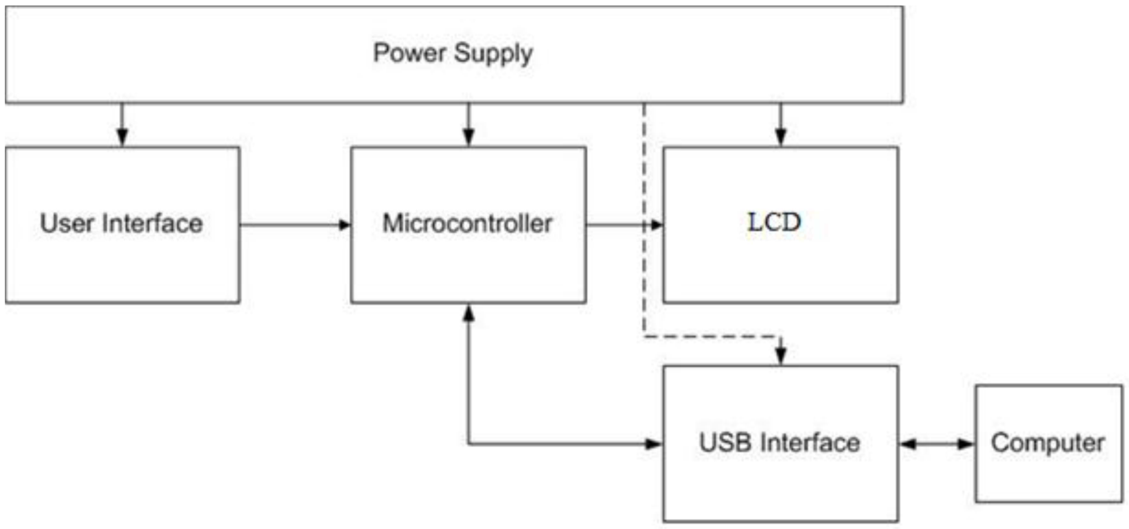

Figure 1.1: Block diagram

1.4 Block Descriptions

1.4.1 User interface

The user interface contains all of the components the user can use to manipulate parameters of the simulation. The design incorporates two six-position switches, a 12-button keypad, a rotary encoder, and two push buttons. Figures 1, 2, and 3 of Appendix B show images of the user interface. The six-position switches are used to select the parameter the operator wishes to modify. One switch is used to select "vitals," such as heart rate, respiratory rate, and oxygen saturation. The other switch is used to select various heart rhythms. Figure 4 of Appendix B (App.B.4) depicts the parameters each switch can select. The push buttons can be used for a variety of different functions. Simulations commonly have pre-programmed "states" that are transitioned through. A button could be used to transition to the next state in a simulation. Buttons can also be used as macros to execute many commands simultaneously. The keypad is used to enter a precise value for a parameter. For example, if the operator wanted to set heart rate to exactly 60BPM. The rotary encoder is used for sweeping parameters up or down. The rotary encoder is very important during simulations because most things do not change instantaneously; they change over time. The rotary encoder allows the user to rapidly or gradually modify a parameter, such as sweeping heart rate from 50BPM to 120BPM. All components of the user interface are connected to the microcontroller. The components of the user interface receive power from both the power supply and the PC through the USB connection.

1.4.2 LCD
The LCD was incorporated into the design so the operator could quickly glance down at the device to distinguish which parameters are being modified and to determine their respective values. The LCD receives the data to display from the microcontroller. It receives power from the power supply. Figure 1.2 depicts the data displayed on the LCD.

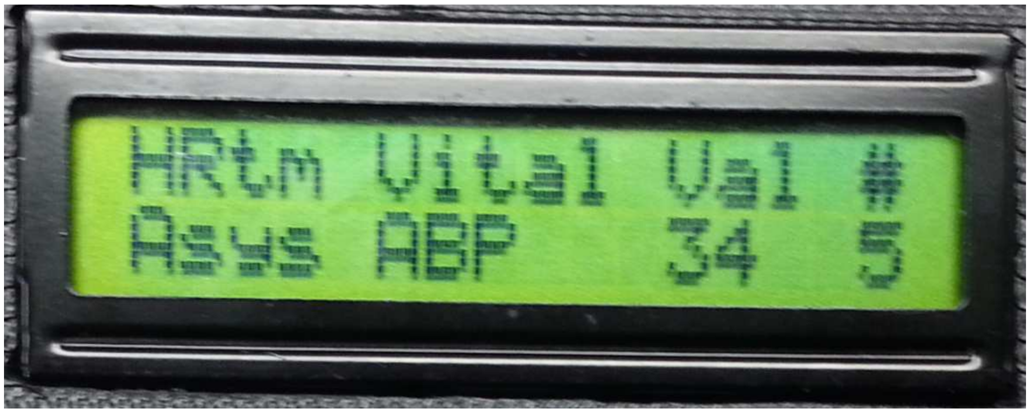

Figure 1.2: The LCD

The two switches are labeled “HRtm” for heart rhythm and “Vital” for vitals. Asystole is the current heart rhythm selected and ABP, or arterial blood pressure is the current vital selected. “Val” is the value ABP has been set to by the rotary encoder. “#” is the number most recently input by the keypad.

1.4.3 Microcontroller
The microcontroller receives power from the power supply and from the PC through the USB interface. The microcontroller receives data from the user interface through analog and digital inputs. The microcontroller is able to interpret the data from the user interface to determine the position each switch is in, values set by keypad or rotary encoder, and if a button is being pressed. The data is then sent serially to the LCD to display the parameters. The data is also sent serially via the USB interface to the computer where it is interpreted by the PC software.

1.4.4 USB Interface
The USB interface serves three purposes. The first purpose is that it serves as a means to upload code to the microcontroller. This gives designers the opportunity down the road to modify the existing code to alter parameters selected by the switches, for example. Additional functionality could also be added. The second purpose of the USB interface is to facilitate communication between the microcontroller and the software running on the computer by transferring data serially across it. Finally, the USB interface allows the PC to provide power to the microcontroller, user interface, and the LCD.

1.4.5 PC software
The initial goal was to be able to interface our device with Laerdal's software so it could be used with their simulations. However, we did not receive the Laerdal SDK until very late in the development process and we determined there was not adequate time to achieve this. Rather, a graphical user interface (GUI) was implemented. The GUI is used as a means to demonstrate that serial communication of data from our device to the PC works. The GUI displays the switch position as well as parameter selected, rotary encoder values, keypad values, and whether a button is being pressed. The idea for using the GUI was outlined in our design review as a contingency plan in the event that we didn't receive the SDK in time to implement it with our user interface.

1.4.6 Power supply
The initial design included a 120VAC to 12V DC wall adapter to power the device. It was later determined that the USB connection from the PC is sufficient to power the device. We left the ability to power the device via 12V wall adapter so that future functionality could be added to the device which would draw more current and consume more power than what can be supplied via USB. The power supply includes a 5V voltage regulator. The voltage regulator is then used to provide power to the microcontroller, LCD, USB interface, and components within the user interface.

2 Design Procedure

2.1 User interface

There are a number of areas in the user interface in which there were alternatives in the design. We knew we wanted some type of device that would give us the capability to sweep parameters. The most obvious device was a linear potentiometer. Using a potentiometer was our initial plan. It was not until after the design review when we realized that this was not the best approach. This is because the microcontroller would interpret the position of the potentiometer based on the output voltage. For example, if the potentiometer ranged from 0 to 5V, then an output of 2.5V might correspond to a heart rate of 100BPM. The problem that arises is that if the code had a preset heart rate of 60 BPM and the potentiometer was outputting 2.5V, then the code would immediately change the heart rate to 100BPM which is not desired. Another example would be if one were to set heart rate to 100BPM and then change the switch to select a different parameter, that parameter would automatically be set to whatever value was mapped to the voltage the potentiometer was currently outputting before the user even has a chance to toggle it. These issues were avoided by using a rotary encoder. With the rotary encoder, only the relative motion of the encoder moving forward and backward is detected, not the absolute value of the signal. So, values would not change until the user begins to toggle the rotary encoder. Another benefit of using a rotary encoder as opposed to a potentiometer is that we can fully rotate a rotary encoder which allows a more fluid manipulation of parameters when compared to the potentiometer.

Another design decision we made was with the switches. We knew we wanted to use switches, but we were not sure how many positions the switch should have. There are switches with four, five, six, and even more positions. The decision to use six position switches was made after discussions with OSF Healthcare. They had given us roughly 10 parameters that they considered were very important to have on the device. Choosing six position switches gave a couple of extra positions for more possibilities, but not so many that things became overly complex.

There were two choices of buttons. We could use a toggle button or a momentary button. We chose a momentary push button because we wanted the output voltage to go high briefly to signal a button press but not stay high until the user pushes the button again. This is a much more user friendly approach, and draws less current. The size of the button was also a design factor. We wanted the button to be relatively large so it could be pressed comfortably. If the button was too large it would take up too much space. Also, if the button was too small it would be hard to locate without looking down at the interface.

There were several ways in which the keypad could have been connected to the Arduino. The most straightforward way was to hook each of the 7 output pins to its own digital input pin. Unfortunately, we had less than 7 digital pins available on the Arduino. Another option was to use a shift register to receive all the data from the 7 output pins on the keypad and shift that data into one digital pin. We were going to use that approach until we found a tutorial, reference [2], on using a resistive network and a single analog pin to interpret a key press. That was the easiest approach and it only used one pin, so we went with the resistive network.

2.2 Microcontroller

Choosing the microcontroller was another important decision**.** The first decision was to choose our microcontroller and we debated between the PIC microcontroller and Arduino. The PIC microcontroller was available for free in the senior design lab. This appealed to us from a cost perspective**.** In the end, we chose to go with the Arduino because the group had experience programming with the Arduino. We felt this was the best choice since there would be less of a learning curve so we could get right into programming. Among Arduinos, we chose the Uno because there was a tutorial on how to build our own, which reduced the cost. Furthermore, we determined it had enough analog pins and digital pins to connect all of our devices to it that were necessary.

2.3 LCD

Choosing a display proved to be more challenging than originally anticipated. We were not sure what type of display to choose or what dimensions to choose since we did not know how much data a user may need to gather from the device. We decided to go with a 16x2 LCD. The screen is small, but we wanted our device to be ergonomic and we knew that incorporating a large display would make it difficult to meet this goal. We also did not want the screen to distract the operator from other responsibilities that they need to control such as the camera system or other peripherals. The 16x2 LCD is also relatively inexpensive compared to other displays. Additionally we were able to find a lot of documentation online describing how to connect this LCD with the Arduino.

2.4 GUI

Our design for our GUI was determined using many factors. We decided to use Processing, because of its ease of implementation. We also determined that other GUI's such as QT didn't work as well with the Arduino and were more cumbersome to implement. Initially we wanted to use the GUI created for Laerdal's SimMan manikin, but we didn't receive the SDK early enough in the development process. The Processing language is based on Java and allows a user to compile and run the code instantly which allows the programmer to get instant feedback when working in this environment.

3 Design details

3.1 User interface

3.1.1 Switches

Our design incorporates two KC series six-position rotary switches. This allows for the selection of a total of 12 different parameters by the operator. Each switch contains seven pins: six input pins and one output pin. When the switch is rotated through each position, it creates a connection between output pin and the corresponding input pin, thus the output is whatever signal is present on that input pin. Figure 3.1 shows the switch circuit schematic implemented.

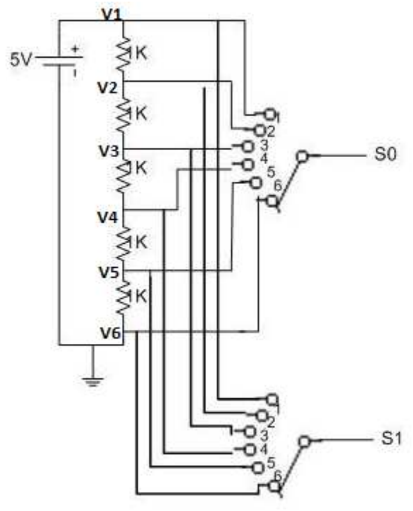

Figure 3.1: Switch circuit schematic

The circuit contains five resistors placed in series, connected to 5V from the voltage regulator. The resistors are all 1kΩ. The resistors are all the same value so that an equal voltage drop can be obtained at each node in the circuit. From observation of the circuit, one can see the top node of the circuit, V1, is connected to 5V and the bottom node V6 is connected to 0V (ground). The calculations of the other node voltages using the voltage divider rule are shown below.

$$V_2 = 5V * \frac{4K\Omega}{5K\Omega} = 4V$$
$$V_3 = 5V * \frac{3K\Omega}{5K\Omega} = 3V$$
$$V_4 = 5V * \frac{2K\Omega}{5K\Omega} = 2V$$
$$V_5 = 5V * \frac{1K\Omega}{5K\Omega} = 1V$$

With node voltages of approximately 5V, 4V, 3V, 2V, 1V and 0V, six unique positions of the switch can be identified by connecting each node of the circuit to a different input switch position. The switch output is connected to the microcontroller which will interpret the switch position based on the voltage it sees.

3.1.2 Keypad

A 12 button keypad is used for precise data entry. The keypad contains the numbers zero through nine as well as a pound and star key. The pound key acts as an enter key to confirm the inputted value. The star key acts as a backspace. The keypad has seven output pins and thus would normally require seven pins analog pins on the microcontroller to function. We did not have seven pins available to dedicate to the keypad so we implemented a circuit that reduces the amount of analog pins needed from seven to one. When a key is pressed on the keypad, two pins on the keypad are shorted together. Table App.B.1 shows the pins shorted for each key pressed [1]. Figure 3.2 shows the keypad schematic [2]. The circuit is essentially a resistor network. When no key is pressed, all pins of the keypad are open circuited and no current will flow. When a key is pressed, two pins of the keypad are shorted together. For example, if the "0" key is pressed, pins 1 and 4 are shorted together. Current can then flow through the 180Ω resistor, through pin 4, out pin 1, and then through a 1kΩ resistor to ground. The circuit is connected to the microcontroller from the node labeled "Analog Input."

Essentially, depending on the key pressed, current flows through a different series of resistors in the network, which creates a different output voltage seen by the microcontroller. The microcontroller can then map the voltage to what key is pressed. The small 1nF capacitor is included to prevent interference induced in the wires from varying voltages detected in the circuit [2].

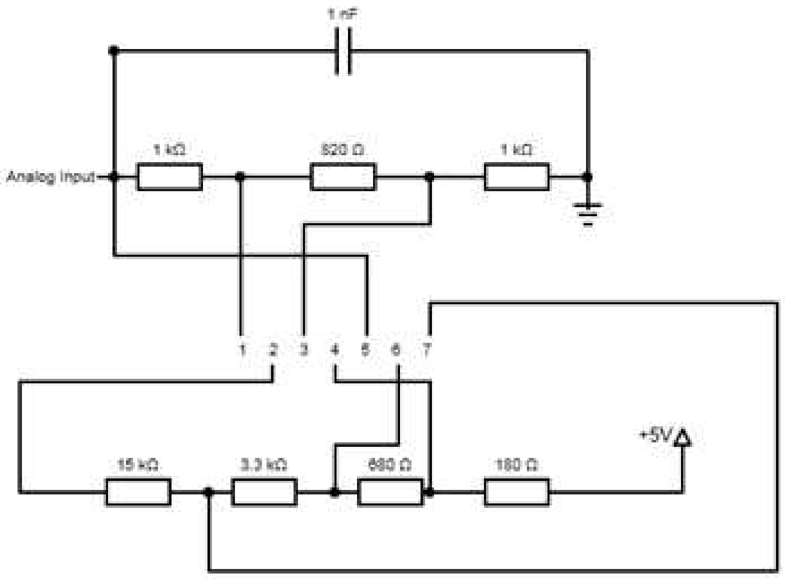

Figure 3.2: Keypad schematic

3.1.3 Buttons

Two buttons are connected to the Atmega328 via two digital pins. Figure 3.3 shows the schematic for the buttons [3]. One pin of each button is connected to 5V from the regulator.

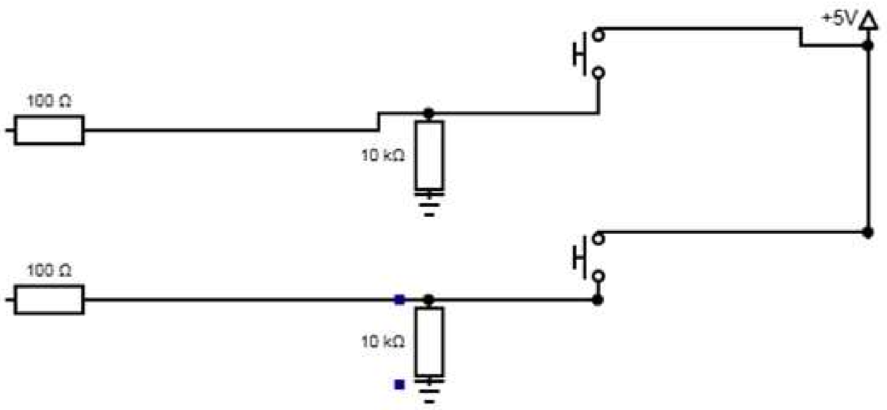

Figure 3.3: Button schematic

The second pin is connected to the digital pin of the microcontroller. A 10kΩ resistor functions as a pull down resistor. We also incorporated a 100Ω resistor to limit the current flowing into the microcontroller. When the button is pressed, the circuit is completed, and current can flow. The Arduino will see approximately 5V at its input. When the button is not pressed, no current flows and the Arduino will see ground.

3.1.4 Rotary encoder

Our design uses one rotary encoder to sweep up and down parameters. The rotary encoder has a full 360 degree range of rotation, as opposed to the limited range provided by a potentiometer. The rotary encoder outputs Gray code, which is interpreted in the microcontroller code to determine the direction of rotation. This particular rotary encoder outputs 2-bit Gray code, 00, 01, 11, or 10. The rotary encoder has three pins: the middle is connected to ground and the other two are connected to two separate analog input pins. We were able to locate preexisting code for interfacing the rotary encoder with the Arduino [4]. The code works by initializing a counter variable to 0 and then reading the analog input pins to determine if the rotary encoder is being rotated. The code determines rotation by comparing the previous state of the rotary encoder and the current state. Since Gray code is well defined, we know what states correspond to what direction of rotation. If the previous state was 00 and the new state is 01, than we should increase the value. If the previous state was 10 and the new state is 11, then we should decrease the value.

3.2 Microcontroller

We used the ATMEGA 328 microcontroller with Arduino bootloader to process the data from the rotary encoder, switches, keypad, and buttons. We decided to go with a "Do it yourself" Arduino instead of buying a preassembled board [5]. It was cheaper and gave us more flexibility with the size of our final PCB. Figure App.B.5 shows the Arduino schematic. In total, out project used 5/6 analog input pins and 10/14 digital pins. The rotary encoder is connected to analog pins 0 and 1, the 6-way switches are connected to analog pins 2 and 3, and the keypad is connected to analog pin 4. Digital pins 0 and 1 (RX/TX) are used to transmit and receive data from the USB/Serial interface. The two buttons are connected to digital pins 9 and 10. The remaining digital pins (12, 11, 7, 6, 5, and 4) are connected to the LCD. Appendix E shows the functioning microcontroller code. The serial port operates at 115200 Baud, the fastest possible speed. The LCD we bought was compatible with the well-developed LiquidCrystal library, which makes writing to the LCD very easy. The code starts by initializing and assigning all the input pins to variables and prints the default menu on the LCD screen. The main loop of the code first reads the button pins to determine if a button has been pressed, then checks the position of the 6-way switch, then checks to see if a key has been pressed, and finally reads the encoder. If an event occurs (i.e. a button has been pressed) the code writes the appropriate data to the LCD and serial port. The code runs fast enough to register all events without error. We were having an issue where the keypad was outputting erroneous voltages for several of the keys and the microcontroller would misinterpret the pressed key. We fixed this by implementing a moving average within the code, so that the error caused by the wrong voltage value was negated.

3.3 Graphical user interface (GUI)

Figure App.F.1 shows the final design of the GUI. Figure App.F.2 shows the final code. Our design needed to have a corresponding switch or textbox for each button, switch, and rotary encoder. We also wanted to have a way for the user to see what position each 6 way switch value corresponds to so we added text boxes under each switch so that we know what "vital" and "heart rhythm" is going to be manipulated. Another factor that we needed to consider was the speed of transmission of data (baud). We initially started at 9600 baud, but eventually we were able to get our system to function at 115,200 baud. The code was simple to create with the templates provided in the Processing library and using the ControlP5 Library [6]. ControlP5 allowed us to easily create controllers for our GUI which included sliders, buttons, knobs, and text fields. The input is sent from the microcontroller as serial data across the USB interface to the PC where the processing code interprets the serial data and updates values within the GUI accordingly. In the code we attached a leading character to each button, switch, and rotary encoder so that we could distinguish which serial data corresponds to which controller on the GUI. We also had an end line character after each piece of data so that we knew how much data needed to be sent to each controller. Once processing received the data based on the leading character value we would take off the leading character and strip away the end line character and we would be left with the data that needed to be updated in the GUI.

3.4 Power supply

There are currently two methods to power the device. The first method is via a 120VAC to 12VDC wall adapter. The wall adapter is capable of providing 1A of current to our device. The wall adapter connects to our device via the PRT-00119 2.1mm barrel power jack. The high voltage pin of the power jack then connects to the input pin of 5V regulator located on the PCB. This regulator is then used to power components throughout the device such as the microcontroller and circuits for the keypad and switches.

The second method is to use the power output of the USB, which outputs 5V and 500mA. When we plug the USB into the PC, it puts 5V directly onto the 5V bus of the PCB. Since the current draw of our entire circuit is less than 500 mA, we are able to use this power. If our circuit were to draw more than 500 mA (due to added functionality, new components, etc.) then the USB would not be capable of powering the board, and the wall jack would need to be used instead. For that reason, we have left the wall jack in the design.

3.5 PCB

The PCB was designed using Eagle software. All circuit components required for the user interface, microcontroller, and USB interface were integrated onto the PCB. Figure App.D.1 shows a layout of all components on the PCB. Figure App.D.2 shows our PCB with all components soldered to it.

4 Design Verification

All design verifications and procedures can be found in Table App.A.1. This section covers verification procedure and results.

4.1 User interface

Verification of the user interface is accomplished by verifying each component of the user interface which includes the following: switches, keypad, push buttons, and rotary encoder.

4.1.1 Switches

The proper functionality of the switches was verified by connecting an oscilloscope to the output pin of the switch. We then rotated the switch through all six positions. Figure App.C.1 shows the oscilloscope output. The data shows that as the switch is rotated through each position, there are six discrete output voltages of approximately 0V, 1V, 2V, 3V, 4V, and 5V. This functions exactly as we wanted it to. One might notice that the voltage drops to 0V very briefly between each switch position. This occurs because the switch briefly is open circuited as it transfers between switch positions which gives a momentary 0V output. We determined that this would not be a problem in our design because it happens so quickly. The user is not able to notice that the switch selects the position corresponding to 0V for a brief moment. This issue could easily be fixed by taking multiple samples of the switch voltage and averaging them together and then using that signal to interpret switch location. This solves the problem that would arise if the program were to use the sample right at the 0V transition.

4.1.2 Keypad

To verify the keypad, we had the microcontroller send the interpreted key press to the GUI, which would then display the key that was pressed. To determine the accuracy of our design, we pressed each key 25 times and compared the key press with the key displayed. Table App.C.1 shows the results. The key accuracy column is how many times the key press matched the key displayed. From the data it can be seen that numbers zero through nine were reasonably accurate. However, the zero, pound, and star keys were displaying the wrong key press very often. To see why this was happening, we used the serial output capabilities of the Arduino to display the voltages it was seeing from the keypad. We observed that the voltages for those three keys were very close together so the Arduino was having trouble mapping the voltage to the key pressed. To fix this issue, we incorporated a moving average function within the Arduino code. The moving average function took 1000 samples of the key pressed and averaged them together to get a more accurate voltage read and thus a more accurate signal. After incorporating this function, the keypad accuracy dramatically increased. Table App.C.2 shows the results after incorporating the averaging function. The keypad still has occasional trouble with the zero, pound, and star key in particular, but the backspace functionality of the star key was incorporated partially for this reason.

4.1.3 Push buttons

To verify proper operation of the push buttons, we connected the output pin to the oscilloscope. The button was pressed and released several times. The oscilloscope data is shown in Figure App.C.2. The pushbuttons function just as desired. When the button is pressed the output is approximately 5V. When the button is released the output is approximately 0V.

4.1.4 Rotary encoder

To verify the rotary encoder, code was written in Processing that would plot the rotary encoder value as a function of time. Figure App.C.3 shows the plot. The rotary encoder was swept through its entire range of values (0 to 255) in the forward direction and then swept from 255 back down to 0 in the reverse direction. The plot shows smooth sweeping functionality of the rotary encoder as desired.

4.2 USB interface

To verify the USB interface we sent serial data from the microcontroller to the PC. The processing terminal printed out the serial data that we sent over. We compared the printed data to the sent data to confirm accuracy. Figure App.C.4 shows the printed data from the USB interface data. The printed data matches the data that was sent.

4.3 Microcontroller

To verify the microcontroller, we wrote code to interpret the various inputs and displayed the data to the LCD. We changed the inputs and watched the corresponding values change on the LCD. Figure App.C.7 shows the LCD verifying the proper operation of the microcontroller.

4.4 Power supply

To verify the power supply we measured the voltage from the wall power adapter as well as from the voltage regulator. The voltages were measured using a multimeter. The voltage from the power adapter was 12.43V. Figure App.C.5 shows the multimeter display. The voltage from the regulator output was measured to be 5.12V. Figure App.C.6 shows the multimeter display.

5. Costs

5.1 Parts

A list of parts purchased for the project is summarized in Table 4.1. The total cost of parts was approximately $145.93.

Table 4.1: Part costs

Nomenclature	Part Number	Vendor	Unit Price	Quantity	Total Price
User Interface					
Rotary Switch 6 Pos	KC26A30.001NLS	Digikey	$5.73	2	$11.46
Linear 1K Potentiometer	296UD102B1N	Digikey	$1.60	1	$1.60
Push Button	PR141C1900	Digikey	$1.43	4	$5.72
Keypad -12	COM-08653	Digikey	$3.95	1	$3.95
Rotary Encoder – Illuminated	COM-10982	Sparkfun	$3.95	2	$7.90
Chicken Head Knob	COM-10000	Sparkfun	$0.95	2	$1.90
Resistor – 100 Ohm	P100BATB-ND	Digikey	$0.09	4	$0.36
Resistor – 10k Ohms	CF14JT10K0TR-ND	Digikey	$0.08	3	$0.24
Resistor – 1K Ohms	CF18JT1K00CT-ND	Digikey	$0.09	5	$0.45
Microcontroller					
7805 Voltage regulator	COM-00107	Sparkfun	$1.25	1	$1.25
Green-LED	COM-09592	Sparkfun	$0.35	5	$1.75
Red-LED	COM-09590	Sparkfun	$0.35	5	$1.75
Resistor – 220 Ohms	CF14JT220RTR-ND	Digikey	$0.08	3	$0.24
Resistor – 10k Ohms	CF14JT10K0TR-ND	Digikey	$0.08	3	$0.24
Capacitor – 10uF	P10425TB-ND	Digikey	$0.08	2	$0.16
16 MHz Clock Crystal	COM-00536	Sparkfun	$0.95	1	$0.95
Small Momentary Tact Switch	COM-00097	Sparkfun	$0.35	1	$0.35
Row Male Header Pins	PRT-00553	Sparkfun	$1.95	1	$1.95
ATMEGA 328	DEV-10524	Sparkfun	$5.50	1	$5.50
FTDI Basic Breakout (board)	DEV-09716	Sparkfun	$14.95	1	$14.95

Capacitor – 22pF	P10426TB-ND	Sparkfun	$0.08	2	$0.08
Arduino Uno	DEV-1102	Sparkfun	$29.95	1	$29.95
LCD Display					
LCD Screen	KS0108B	Digikey	$19.95	1	$19.95
Linear 1k Potentiometer	296UD102B1N	Digikey	$1.60	1	$1.60
USB Interface					
USB Connector	292303-1	Digikey	$1.44	1	$1.44
USB 2.0 A Male to A Male	AE9930-ND	Digikey	$3.19	1	$3.19
Power Supply					
Wall Adapter Power supply 12 Volt DC 1-Amp	N/A	Amazon	$2.05	1	$2.05
3-D Housing Device	N/A	Fab Lab	$25	1	$25

5.2 Labor

The total costs for labor are summarized in Table 4.2. Labor costs were calculated using the ideal but realistic salary for each team member. Each team member worked on the project approximately 15 hours per week for a total of 12 weeks in the semester.

Table 4.2: Labor costs

Member	$/hour	# of weeks	Hours/week	Total hours	Subtotal	(x2.5)
Jacob Clifton	$35	12	15	180	$6,300	$15,750
Ryan Schmid	$35	12	15	180	$6,300	$15,750
David McKiernan	$35	12	15	180	$6,300	$15,750
				Total	$47,250	

5.3 Total cost

Total costs are shown in Table 4.3. The total cost is the sum of parts and labor costs.

Table 4.3: Total cost

Section	**Total**
Labor	$47,250.00
Parts	$145.93
Grand Total	$47,395.93

6. Conclusion

6.1 Accomplishments

Our project had many successes and we accomplished a great deal over the entire development cycle this semester. The 3D housing we created for our device was a huge success: it fits comfortably into the hand of the user and the housing device is also the most complex project to date produced from the 3D printer at the Champaign-Urbana Community Fab Lab.

Our device is ergonomic and has been laid out well. The buttons are easily accessible by the users thumb and the rotary encoder can be manipulated easily with one finger. The switches and keypad can also be accessed quickly. We were also able to attain the maximum communication rate of 115200 baud between the microcontroller and PC, allowing for manipulation of data as quickly as possible which corresponds to real time manipulation of data. We feel our design will dramatically improve the simulation experience to better train the students.

Finally, we are most proud of the fact that every design goal (other than implementation of the Laerdal software) and verification we had stated in our design review at the beginning of the semester was accomplished. We originally created a contingency plan for the Laerdal software, because we knew this was something that was out of our control based on when we received the software or if we received it at all.

6.2 Uncertainties

Although our device works well, there are a few uncertainties we have. When we were assembling our PCB and components into the housing device, we noticed that many of the wires were placed under a decent amount of stress in order to fit various components such as the switches to their desired locations. We are worried that over time some wires may become disconnected due to these stresses. In fact, during testing we noticed that sometimes the LCD would flicker and stop displaying if we moved the device. This is likely due to a wire becoming loose from the solder joint.

Another uncertainty we have is with the reliability of some of the mechanical parts. One of our switches seems to have broken after our project was completely assembled. The switch was most likely broken by trying to rotate it past its intended range of motion. The switch turns much more loosely than the other. Additionally, when used, the GUI for the switch is flickering between switch positions, which also points to an issue with the switch.

6.3 Ethical considerations

Our device will be used in conjunction with other peripherals that all are designed to aid the learning process in the medical field. Furthermore, we are also working with OSF Healthcare and Jump Trading. As such, we will need to abide by the IEEE Code of Ethics. The ethical considerations most relevant to us are given in Table 5.1.

Table 5.1: Ethical considerations

1. to accept responsibility in making decisions consistent with the safety, health, and welfare of the public, and to disclose promptly factors that might endanger the public or the environment;	It is our responsibility to ensure that our device is designed in a manner that will function correctly and safely with the welfare of the public in mind. We must keep in mind that our device will be used to train future doctors and that it must perform exactly as its intended purpose described so that the operator may do the best job possible to facilitate the continued learning of the students using in the simulations
2. to avoid real or perceived conflicts of interest whenever possible, and to disclose them to affected parties when they do exist;	We will be working with Jump Trading and it is in our mutual interest to ensure that a line of communication will be open to create an environment where both parties are aware of what the other is doing so there is no misunderstandings or miscommunication which could cause conflicts of interest and if a conflict does arise we will disclose them as soon as possible
3. to be honest and realistic in stating claims or estimates based on available data;	Communication with both our TA's and Jump Trading is important and we must strive to be as honest and forthright as possible in our

	plans and expectations for our project
6. to maintain and improve our technical competence and to undertake technological tasks for others only if qualified by training or experience, or after full disclosure of pertinent limitations;	The purpose of our project at its root is to improve our technical knowledge and to work together as a team in order to design and implement a product from inception all the way to a functioning prototype. Throughout the semester we will strive to further our understanding of the technologies that we are using and to discover both their potential and limitations
7. to seek, accept, and offer honest criticism of technical work, to acknowledge and correct errors, and to credit properly the contributions of others;	We will ensure that we credit anyone who has had a contribution in helping create our design. It is our job to credit and cite anything that we gained from the work of others

6.4 Future work

We have achieved all of our design goals, but there is still much that can be done. As stated in section 5.2, there are some reliability issues. We would like to replace the switches with switches that are more reliable to be able to handle all of the strains that will be placed on them. We would also like to disassemble the device and check all of the solder joints for quality.

The PCB needs a few modifications to the Eagle file, because one of traces didn't go to a digital pin, and the USB had two traces that were reversed so we had to break the traces and connect wires in order to fix the problem.

Although we felt the device was ergonomic, we still feel it is a bit too large. Now that we know everything fits well into the housing, we could downsize the housing a bit. We feel this will make it more comfortable to use, especially for people with smaller hands.

We would like to improve the labeling on our device. Currently, the switch positions aren't labeled, so the only way to know what parameter is selected is to look at the LCD or the GUI. We didn't want to label the positions with permanent labels such as stickers because we liked that the user has the ability to reprogram the switches or other devices to do the functionality the user wants. We would like to implement some type of electronic labeling so that the labeling is programmable.

Finally, the most important goal we would like to accomplish is to be able to connect our device with Laerdal’s software. This was an original goal of the project but we were not able to meet it since we didn’t get the SDK until late in the design process. Achieving this would mean our project could actually be used in simulations. This please OSF Healthcare and Jump Trading.

References

[1] *COM-08653,* datasheet.2011. Available at: http://dlnmh9ip6v2uc.cloudfront.net/datasheets/Components/General/SparkfunCOM-08653_Datasheet.pdf

[2] Arduino 3 Wire Matrix Keypad, web page. Available at: http://www.instructables.com/id/Arduino-3-wire-Matrix-Keypad/. Accessed September 2012.

[3] Button, web page. Available at: http://arduino.cc/en/Tutorial/Button. Accessed September 2012

[4] Reading Rotary Encoder On Arduino, web page. Available at: http://www.circuitsathome.com/mcu/programming/reading-rotary-encoder-on-arduino. Accessed September 2012.

[5] Build Your Own Arduino, web page. Available at: http://www.instructables.com/id/Build-Your-Own-Arduino/. Accessed September 2012.

[6] ControlP5, web page. Available at: http://www.sojamo.de/libraries/controlP5/. Accessed November 2012.