Jump Trading Medical Simulation Controller

by

Jian Chen

Michal Rys

Final Report for ECE 445, Senior Design, Fall 2013

TA: Justine Fortier

April 30, 2013

Project No. 35

ABSTRACT

The Jump Trading Simulation and Education Center is a high-tech facility used train future medical doctors. The facility's surgery simulators are currently controlled by software running on laptops. In order to make an adjustment to the simulation, the technician must tediously navigate the software's interface with a mouse and keyboard, making on-the-fly adjustments difficult and slow. This report documents the design and construction of a hardware device that simplifies such adjustments by providing a physical interface that that is easily navigable with minimal attention.

Table of Content

I Introduction

1.1 Benefits 1
1.2 Features 1
1.3 Block Diagrams 2

II Design

2.1 Design Procedure 5
2.2 Schematics and Flowcharts 7

III Requirements & Verification

3.1 User Interface 11
3.2 LCD Unit 11
3.3 Micro-controller 12
3.4 Power Supply 12
3.5 Software Interface 12

IV Cost

4.1 Labor 13
4.2 Parts 13

V Conclusion

5.1 Accomplishments 15
5.2 Uncertainties 15
5.3 Ethical Considerations 15
5.4 Future Work 17

VI Device Use Guide

6.1 Connection 18
6.2 Device Use 18

VI References 19

I. INTRODUCTION

1.1 Benefits

- Provides an ergonomic solution to making on-the-fly adjustments of simulation parameters (heart rate, airway respiratory rate, diastolic blood pressure, systolic blood pressure, oxygen saturation)
- Eliminates the use of software GUI as a form of making adjustments to the aforementioned parameters
- Minimizes the amount of attention the technician focuses on physically making an adjustment to a parameter.
- Maximizes the amount of attention the technician focuses on the students and simulation.

1.2 Features

- Large wall-mounted backlit LCD unit for easy glancing at parameter values
- Two adjustment knobs for quick changes to a parameter's target value
- Ability to interface with existing Laerdal software used to control simulations

1.3 Block Diagrams

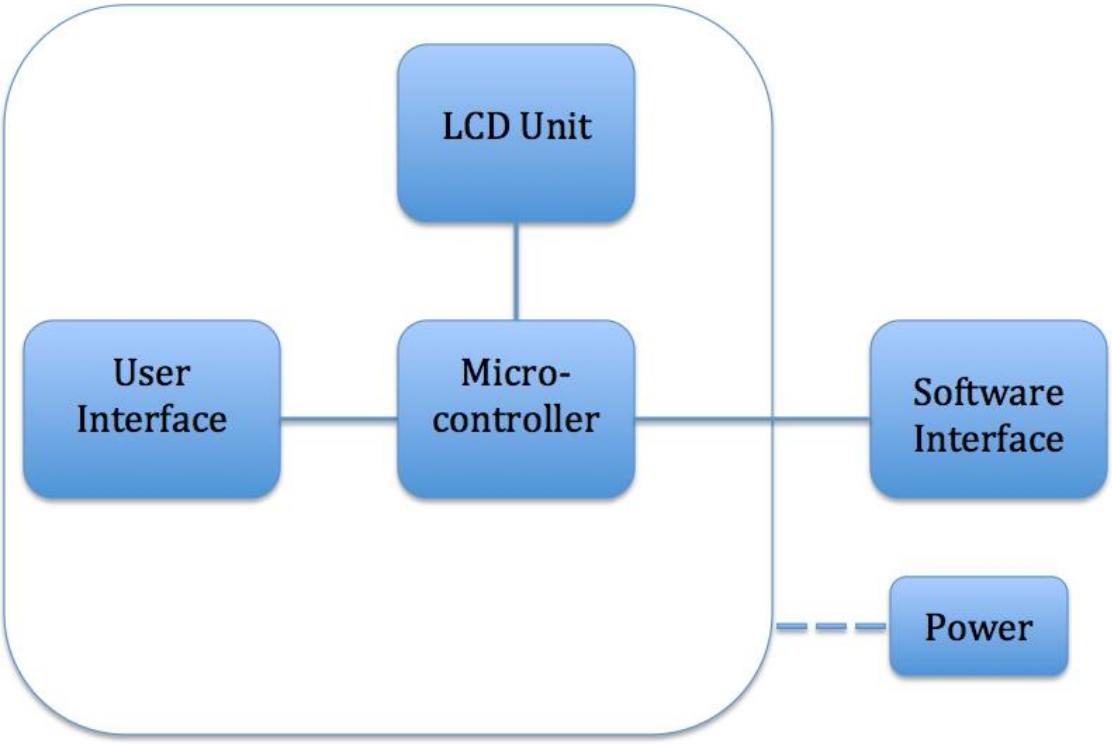

Figure 1 Top Level Block Diagram

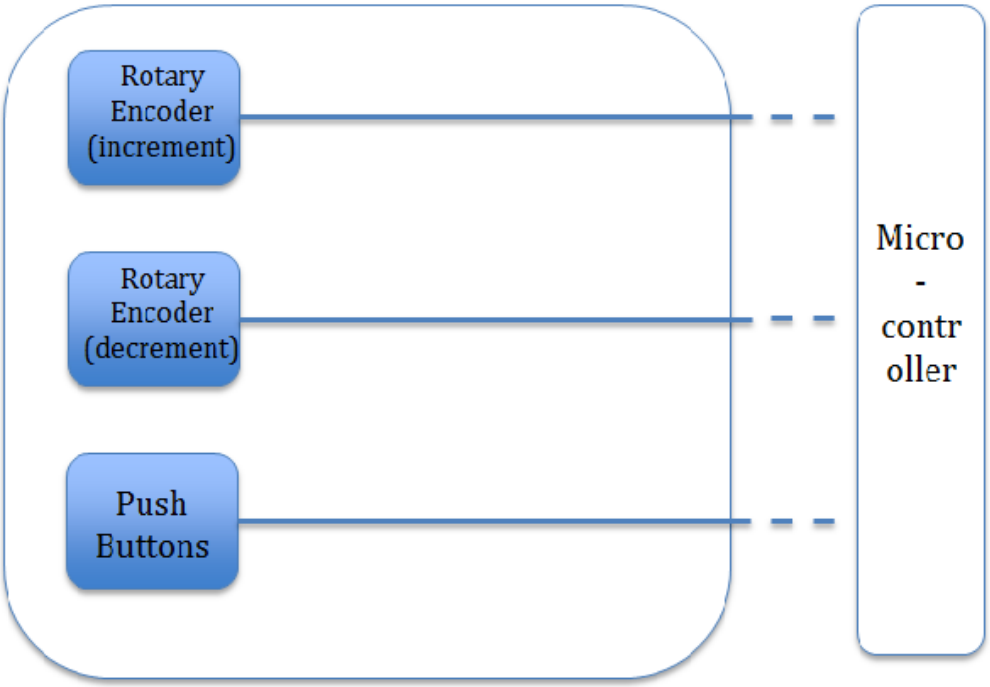

Figure 2 User Interface

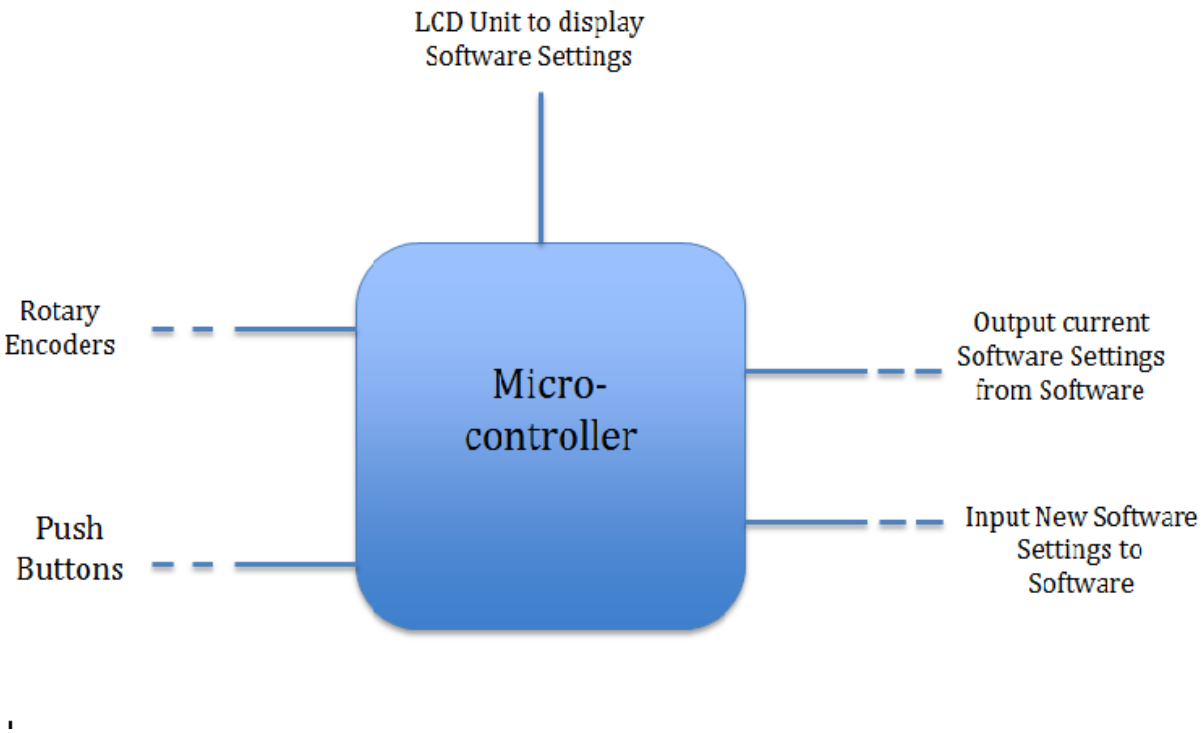

Figure 3 Micro-controller

1.3.1 User Interface:

The User Interface consists of all hardware associated with selecting, updating, and initiating the submittal of a SimMan parameter.

1.3.2 Microcontroller:

This is the central hub of the device. The Microcontroller block processes input from the user, sends data over a serial connection, and parses and displays data received from the Software Interface.

1.3.3 LCD Unit:

The LCD Unit displays all current values of changeable simulation parameters. Additionally, while the device is in the parameter value input state, the current input value is displayed.

1.3.4 Software Interface:

The software interface has two main functions. First, upon receiving a request from the microcontroller, the Software Interface actually performs the parameter changes via the Laerdal SimMan software development kit. Second, upon detecting software side parameter changes, the Software Interface sends the freshest value of the parameter to the Microcontroller to display.

1.3.5 Power Unit:

The device is powered by 12V DC supplied by a wall-connected cable.

II. DESIGN

2.1 Design Procedure

2.1.1 User Interface:

Push buttons are used for parameter selection. Every adjustable parameter has a dedicated corresponding button on the device. The device has 5 such buttons, one each for heart rate, airway respiratory rate (awrr), diastolic blood pressure, systolic blood pressure, and oxygen saturation. After selecting a parameter button, the user then uses two rotary encoders to adjust the value of the parameter. One rotary encoder is used for incrementing, the other for decrementing the value.
Value submission is controlled by a "Submit" button on the device. If the controller is in a value adjustment state, then this button will cause the microcontroller to begin pushing the values to the Laerdal software.
Another pushbutton is used to allow the simulation to be paused.
All User Interface components feed directly into digital input pins on the microcontroller.
There are alternatives for the choice of parameter selection mechanism. A rotary selector could serve the same purpose as our button setup. The final design did not adapt this due to us believing that the buttons allow for a more intuitive user experience. Because the user would be using this device in a simulation environment, his/her attention would be focusing on the observation room, user could easily lose track of the position of rotary selector, and thus lose sense to which parameter is being selected.

2.1.2 Microcontroller:

The device uses an Arduino Due microcontroller. The Arduino Due runs an Atmel SAM3X8E ARM Cortex-M3 processor and features 54 digital and 12 analog input/output (i/o) pins and an 84 MHz clock speed. All digital circuitry associated with the microcontroller will run at 3.3V. The microcontroller receives data from two sources: parameter values from the User Interface over digital and analog i/o lines, and Laerdal software status over the Universal Serial Bus (USB) mounted on the board from the software interface. After processing the input, the microcontroller outputs values to the LCD Unit over digital i/o pins, and send commands to the Software Interface over the USB.

2.1.3 LCD Unit:

The LCD Unit consist of 3 separate 4 lines*20 characters LCD's. Two LCD's are dedicated to displaying the values of all parameters currently set in the Laerdal software. This allows the technician controlling the simulation to easily view all current values simultaneously. If the device is in a parameter-setting state, then the third LCD displays a live value of the parameter being set by the technician through the user interface. The LCD Unit receives all display data from the Microcontroller.

2.1.4 Software Interface:

The Software Interface facilitates all communication between the device and the Laerdal software. Upon launch, the Software Interface performs a port scan on all serial ports available, and attempts a handshaking protocol. Only the port on which our device is present will cause the handshaking procedure to run to completion, thus establishing where the software interface can find the device. The Software interface invokes various tools available through the Laerdal SDK to perform two main tasks: updating parameters as inputted by the user and sending update packets to the microcontroller whenever a parameter is changed within the Laerdal software itself. This two-way communication allows the device to display the freshest possible values on the device, and to actually perform the update of a parameter in the system. The communication interface is implemented as a C# windows application using a USB connection to communicate with the microcontroller.

2.1.5 Power Unit

Power is supplied by a 120 V plug into the wall. The adapter converts 120V AC into 12V DC and feeds into the Arduino microcontroller. 3.3V pins are used to power up LCD Unit. The total current draw is:

Each connection to the 3.3-Volt pin draws 50mA current. Up to 17 pins could be drawing power from the 3.3V pins (7 for buttons, 6 for 3 LCDs, and 2 for the 2 rotary encoders).

Thus:

I(3.3-Volt pin) = 17*50mA = 850mA;

All calculation is based on theoretical maximum current draw, in practice, the current draw is less than 850mA.

2.2 Schematics and Flowcharts

2.2.1 Schematic for overall system

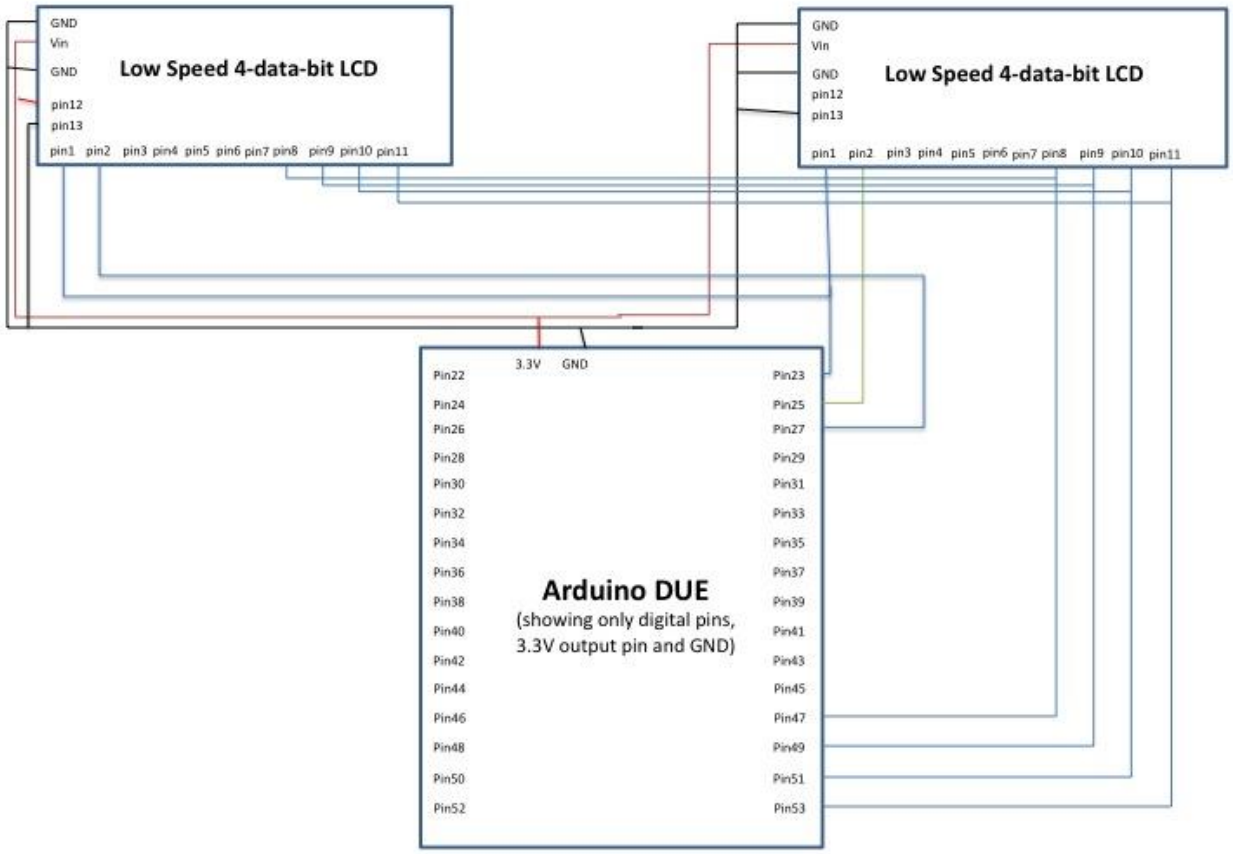

Figure 4: Schematic for low speed 4-data-bid LCD using shared data lines and separate Enable Lines

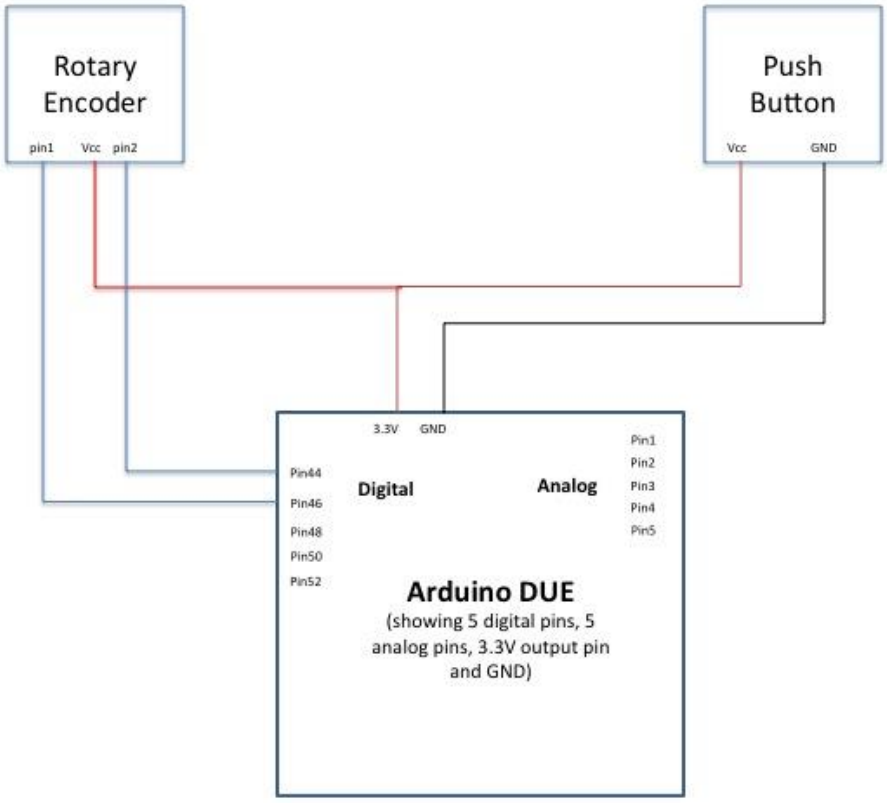

Figure 5: Schematic for Rotary Encoder and Push Button (7 buttons connected through internal pull-up resistors, 2 rotary encoders are connected with similar wiring)

2.2.2 Flow Charts for System Software

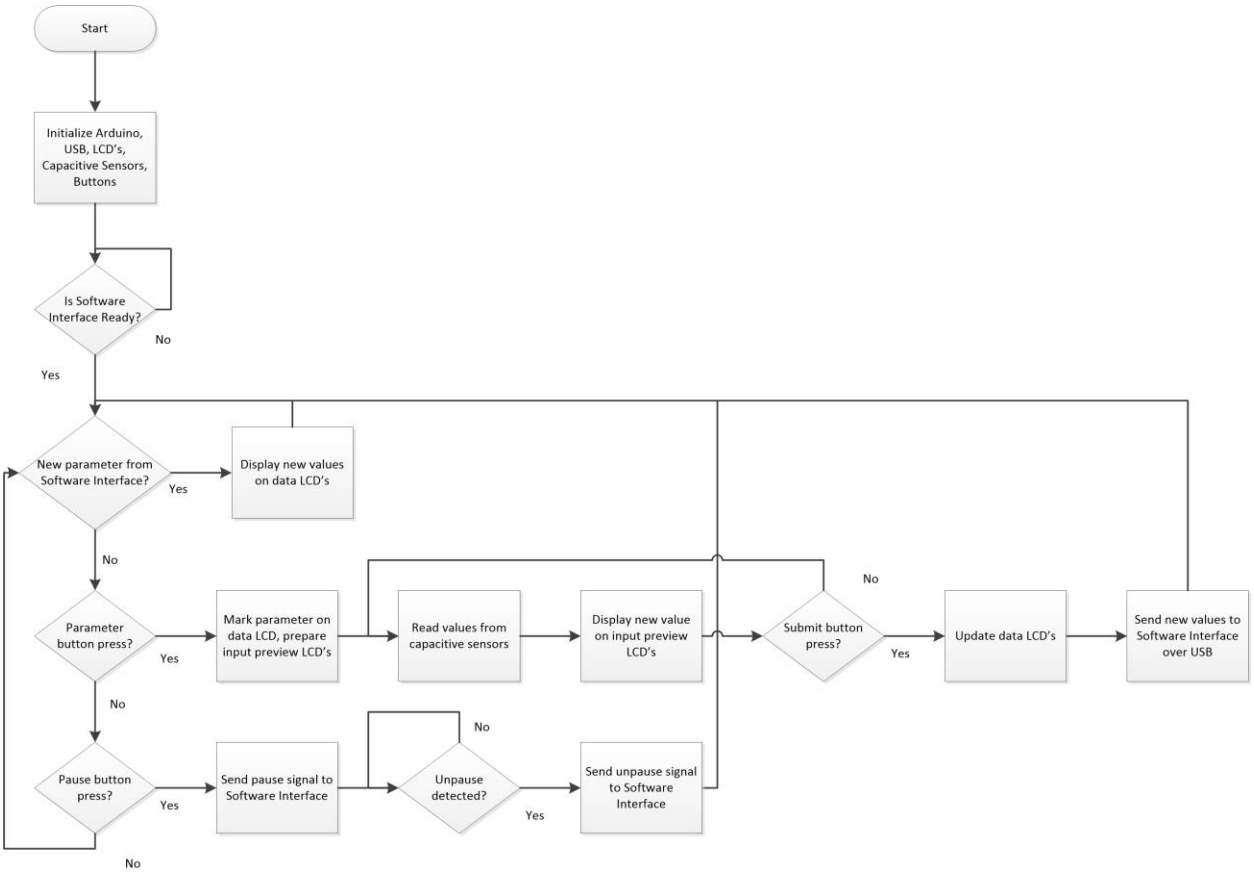

Figure 6: Flow chart for microcontroller software

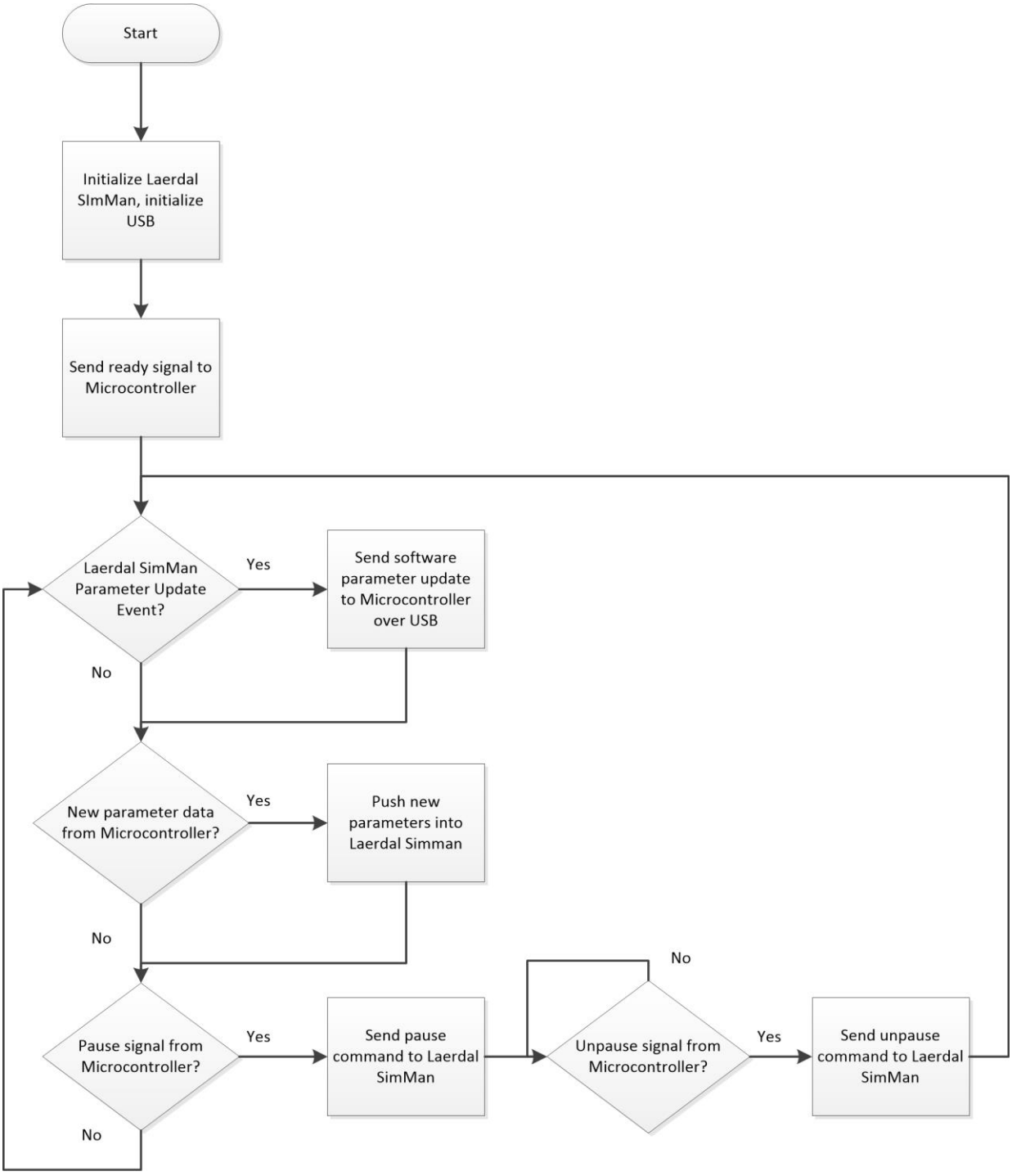

Figure 7: Flow chart for Software Interface

III REQUIREMENTS & VERIFICATION

For the verification procedure table, please refer to the table appended at the end of the report.

3.1 User Interface

Each button connects to a 3-Volt input pin on the Arduino using internal pull-up resistors and ground. A multimeter was used to perform tests on the buttons. When the button is released in natural state, the voltage read 3.13V; when the button is pressed, the voltage read 0V. This was the expected result

When testing our capacitive touch pads for functionality, we ran into problems. Our setup consisted of a test program running on the Arduino Due that sent the coordinates reported by the touch sensor over a serial connection. Unfortunately, the Arduino would always incorrectly send (0,0) coordinates. After successfully using an older-generation Arduino with the same program, we delved into the library code supplied by the part manufacturer to see where the problem may be. Eventually we narrowed the problem down to library calls related to I2C communication. Despite official documentation saying otherwise, we found that the I2C communication library for the Arduino Due is not yet fully functional. Our findings were confirmed by numerous posts on the internet.

Because we no longer were able to use the capacitive touch sensors, we decided to implement parameter value input functionality with two rotary encoders. To test the encoders, an Arduino test program was written to increment or decrement and display a numerical value. Due to a nuance of the encoders' design, we were not able to get consistent performance for counter clockwise rotation detection. To compensate for this, we re-assigned one of the encoders from selecting Adjustment Time, to decrementing a parameter value. We verified encoder functionality by using an oscilloscope. Both encoder output pins were monitored and were correctly changing between 0V and 3V.

3.2 LCD Unit

Each of the three LCDs were wired with a 4 data pin connection and tested. First we ran a test program on all 3 LCDs, and correct values were displayed. The refreshing rates were fast enough to catch and display all the real time

changing values. Additionally, a multimeter was used to test each pin's voltage. Pin 2, 15 were 3.2 volts, pin 1, 3, 5, 6, 16 were 0 volt, and data pin 14, 13, 12, 11 are within range of 0 to 3 volts. All the values were within our expected ranges.

3.3 Micro-controller

The Arduino was connected with all 7 buttons and 2 rotary encoders. A test program was written to check if buttons were being properly debounced, and to ensure that when a button is pressed and held the system only reacts a single time. A test program and serial listener were used to test serial communication. First, we verified that data packets were being properly constructed and sent over the serial connection. When testing serial reads, we found that the Arduino Due was leaking book-keeping bytes into the data delivered to our program. Online searches showed that this is a known issue with the Arduino Due firmware. To get around this problem, we created custom headers for all communication between our Microcontroller and the Software interface. When reading from the serial port, our Microcontroller would only accept packets with our headers.
Each of the pins are also measured voltages with the multimeter, the voltages are reading within 3.3 volts. They are in the expected safe range.

3.4 Power Supply

The power cord is plugged into the wall, and connected to the Arduino. The Arduino is fully functioning.

3.5 Software Interface

The Software Interface was tested in three phases. First, the interface was modified to not accept any serial input, and attempt to submit and retrieve parameter values from Laerdal SimMan. Next, we tested if our parameter-update detection events were being properly triggered by creating a window with a text box that displayed a timestamp every time a parameter change was detected. We fed manual parameter changes into the SimMan Virtual Manikin, and confirmed that the Software Interface was properly reacting. Finally, we re-enabled serial communication and used the on-screen textbox to display all data received from the Arduino. We verified that all data was being received as expected.

IV. COST

4.1 LABOR:

Member	$/hour	# of weeks	Hours/week	Total hours	Subtotal	(x2.5)
Jian	50	12	15	180	9000	22,500
Michal	50	12	15	180	9000	22,500
					Total: $ 45,000	

4.2 PARTS:

Name	**Cost Each**	**Quantity Needed**	**Total Cost**
Arduino Due	48.45	2	96.90
Capacitive Touch Kit For Arduino	18.50	2	88.68 (incl ship)
small push buttons	n/a	8	5.95
illuminated latching pushbutton switch	3.95	1	3.95
Large button	3.95	7	27.65
3.3 V backlit lcd (20x4)	21.95	4	87.80
Arduino Power Supply	6.95	4	27.80
2m micro usb cable	8.60	2	17.20

potentiometer (for controlling contrast on lcd)	1.25	4	5.00
Logic Level Converter	1.95	2	3.90
Enclosure A	24	1	24
Enclosure B	16	1	16

- **GRAND TOTAL = LABOR + PARTS = $45353.15**

V. Conclusion

5.1 Accomplishment

The project achieved all initial requirements proposed by Jump Trading. With the push buttons and rotary encoders, users could easily adjust parameters without looking at the controller. The LCD unit helps the user to have a glimpse at the value and parameter they are changing.

5.2 Uncertainties

After constructing the device, we noticed that our LCD Unit would heat up during operation. The enclosure for LCD Unit is housing 3 LCDs and the microcontroller. All these components are generating considerable heat, and the enclosure is sealed with only three holes for cable connections. We do not know if this heating is dangerous to the microcontroller, but we have not run into any issues so far.

5.3 Ethical Considerations

Our project was to create a user input device for the medical simulation system used by Jump Trading to help the simulator smoothly transition between the process of observing (when the simulator is looking out horizontally) and parameters-adjusting (when the user is looking down to the device holding in one's hands). Several IEEE Code of Ethics were addressed:

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;	The Jump Trading Simulation and Education Center offers medical training programs by simulating real-time medical scenarios to training medical students to better prepare them for real-life practice in their future. Our device makes it easier for the simulator to change parameters during the process of simulation. The device allows technicians to free their eyes from looking at the monitor while changing parameters. Thus they could better focus on the on-going

	simulation. Our project serves for the safety, health, and public welfare.
2. To avoid real or perceived conflicts of interest whenever possible, and to disclose them to affected parties when they do exist;	We kept a communication channel with Jump Trading through emails, and phone calls. Our project design is built based on our natural understanding. If a conflict did occur we disclosed it as soon as possible. We also worked on open source components and software, trying to avoid any potential patent infringement by other third parties.
3. To be honest and realistic in stating claims or estimates based on available data;	We tried to make a detailed verification plan. The real verification process followed this plan.
4. To reject bribery in all its forms;	Bribery did not occur, but in the case it happened, we would have rejected any kind of bribery and reported the incident.
5. To improve the understanding of technology; it's appropriate application, and potential consequences;	By preliminary research, we learned to use different technologies to approach the problem. By comparing the cons and pros we chose the best approach given the circumstances.
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;	We carefully evaluated all our initial ideas. Because we needed to deliver a functioning product, we often made conservative choices in design to maximize the chances having a functioning device.
7. To seek, accept, and offer honest criticism of technical work, to acknowledge and correct errors, and to credit properly the contributions of	We were open to all critiques and criticisms. We also acknowledged and corrected errors.

others;	
8. To treat fairly all persons regardless of such factors as race, religion, gender, disability, age, or national origin;	We made every effort to keep the equality of our working environment. No racial, religious, gender or any kind of bias was tolerated in the team.
9. To avoid injuring others, their property, reputation, or employment by false or malicious action;	We strictly followed all the lab safety codes and regulations. We also kept a high self-discipline and respect for each other, and conducted our behavior lawfully.
10. To assist colleagues and co-workers in their professional development and to support them in following this code of ethics.	We were committed to every team member in helping with each other's academic and professional development.

5.4 Future Work

Future expansion might include:

1. Adding more functionality by adding more push buttons
2. Explore other input methods, including capacitive touch surface again to improve ergonomics.
3. Consider using wireless communication between the LCD unit and hand hold controller unit to increase the mobility and durability of the communication.

VI. Device Use Guide

6.1 Connection

1. Connect the usb cable between the device and computer running SimMan.
2. Connect the power supply to the device.
3. Launch Interface.exe
4. Select the Manikin being used and click "Connect"
5. The device is ready to use

6.2 Device Use

The LCD's display the values of the parameters currently live in the manikin. Changes can be made either through the device or through SimMan.

To pause the simulation, press the "pause" button. To unpause, press the "pause" button again.

To load a new parameter:

1. Press the corresponding parameter button.
2. Use the knobs to adjust the value on the bottom screen to the desired level. The right knob is used to increase the value, and the left to decrease the value.
3. Press the submit button.

VII. References

1. "Arduino - Analog Input." Arduino - AnalogInput. N.p., n.d. Web. 30 Sept. 2012. <http://arduino.cc/en/Tutorial/AnalogInput>.

2. "Arduino - ArduinoBoardUno." Arduino - ArduinoBoardUno. N.p., n.d. Web. 30 Sept. 2012. <http://arduino.cc/en/Main/ArduinoBoardUno>.

3. "Arduino." Build Your Own. N.p., n.d. Web. 30 Sept. 2012. <http://www.instructables.com/id/Build-Your-Own-Arduino/>.

4. "Arduino Character LCD Tutorial." Arduino Character LCD Tutorial. N.p., n.d. Web. 30 Sept. 2012. Sept. 2012. <http://www.hacktronics.com/Tutorials/arduino-character-lcd-tutorial.html>.

5. "Setting up an Arduino on a Breadboard." Physical Computing at ITP. N.p., n.d. Web. 30 <http://itp.nyu.edu/physcomp/Tutorials/ArduinoBreadboard>.

Appendix

VERIFICATION PROCEDURE

Performance Requirements	Testing/ Verification
User Interface 1. Each of the 7 buttons should function properly as on/off buttons. They are all edge-triggered. 2. Capacitive Touch Surface for Value should function properly. 3. Capacitive Touch Surface for Time should function properly. 4. LED should be lighted when	1. Using a breadboard, feeding it with 3V voltage, connecting a button using a Multimeter to measure the voltage across the resister. 2. Connect Capacitive Touch Surface for Value to a 12pin-to-6pin converter, feed the VCC pin with 5 V DC voltage, connect GND pin to the Ground, use a Multimeter to test the voltage of

PAUSE button is pressed, and should be turned off when the PAUSE button is un-pressed. 5. The output voltage of Capacitive Touch Surface for Value's digital pins should be regulated to 3.3 volts. (The new Arduino digital pins only take 3.3 Volts input) 6. The output voltage of Capacitive Touch Surface for Time's digital pins should be regulated to 3.3 volts. (The new Arduino digital pins only take 3.3 Volts input)	the rest 4 pins when touch different grid on the Touch sensing part of the device. 3. Connect Capacitive Touch Surface for Time to a 12pin-to-6pin converter, feed the VCC pin with 5 V DC voltage, connect GND pin to the Ground, use a Multimeter to test the voltage of the rest 4 pins when touch different grid on the Touch sensing part of the device. 4. Connect the LED button in series with button on the breadboard, connect the breadboard with 3.3V DC voltage and ground accordingly. When button is pressed, LED should light up, when button is un-pressed, the light should turn off. 5. Following the same procedures of step2, but extend the output to pass it through a voltage divider, regulate output from 5V to 3.00V±.33V. When the corresponding pin is high, the voltage reading from the Multimeter should be 3.00±.33 Volts. 6. Following the same procedures of step3, but extend the output to pass it through a voltage divider, regulate output from 5V to 3.00±.33V. When the corresponding pin is high, the voltage reading from the Multimeter should be 3 Volts.
<u>LCD Unit</u> 1. When turned on, LCD_names1 must stably runs on 3.3 ±	1. Power the LCD, use a Multimeter to measure the power pin of the LCD, the reading should be 3.30 ± 0.33

10% volts. 2. When turned on, LCD_name2 must stably runs on 3.3 ± 10% volts. 3. When turned on, LCD_values must stably runs on 3.3 ± 10% volts. 4. When turned on, LCD_time must stably runs on 3.3 ± 10% volts. 5. LCD 's display delay from the input submition must be less than 0.3 second.	Volts. 2. Power the LCD, use a Multimeter to measure the power pin of the LCD, the reading should be 3.30 ± 0.33 Volts. 3. Power the LCD, use a Multimeter to measure the power pin of the LCD, the reading should be 3.30 ± 0.33 Volts. 4. Power the LCD, use a Multimeter to measure the power pin of the LCD, the reading should be 3.3 ± 0.33 Volts. 5. Connect the LCD to Arduino, connect Arduino to User Interface, use a oscilloscope to measure the voltages of LCD input pin and User Interface output pin, read the time difference of two spikes, this difference should be less then 0.3 seconds.
Micro-controller Inputs: 1. Each of the 7 buttons should map to a specific parameter we are going to change. 2. Capacitive Touch Surface for Value should behave according to the design, with moving up mapping to 1 unit increase value of the chosen parameter; moving down mapping to 1 unit decrease value of the chosen parameter; moving right mapping to 0.1 unit increase	Inputs: 1. Connect the buttons through wires to the pins on the Arduino board. Using a Arduino test code on the computer to read out the value when different buttons were pressed: pin1 (Heart Rate): correspond variable value is assigned to 1; pin2 (Respiratory Rate/ awRR): correspond variable value is assigned to 1; pin3 Oxygen Saturation: correspond variable value is assigned to 1; pin4 Blood Pressure (Diastolic):

value of the chosen parameter; moving left mapping to 0.1 unit decrease of the chosen parameter. 3. Capacitive Touch Surface for Time should behave according to the design, with moving up mapping to 1 unit increase time of the chosen parameter; moving down mapping to 1 unit decrease time of the chosen parameter; moving right mapping to 0.1 unit increase time of the chosen parameter; moving left mapping to 0.1 unit decrease time of the chosen parameter. 4. Arduino should correctly present the input value from SimMan software through USB cable connection. Outputs: 1. Arduino should output a 6-pin output to the LCD_names1 to display the parameter names. 2. Arduino should output a 6-pin output to the LCD_names2 to display the parameter names. 3. Arduino should output a 6-pin output to the LCD_values to display the parameter values. 4. Arduino should output a 6-pin output to the LCD_time to display the parameter evolution time. 5. 3V3 VCC pin outputs 3.00V ± .33V 6. 5 VCC pin outputs 5.0V ± .5V	correspond variable value is assigned to 1; pin5 Blood Pressure (Systolic): correspond variable value is assigned to 1; pin6 Pause: correspond variable value is assigned to 1; pin7 Submit/ Save: correspond variable value is assigned to 1; 2. Connect the Capacitive Surface for Value through wires to the pins on the Arduino board. Using a Arduino test code on the computer to read out the value when different gestures were performed: Sweeping up-to-down by one sensing grid: value decrease by 1 unit. Sweeping down-to-up by one sensing grid: value increase by 1 unit. Sweeping left-to-right by one sensing grid: value increase by 0.1 units. Sweeping right-to-left by one sensing grid: value decrease by 0.1 units. 3. Connect the Capacitive Surface for Time through wires to the pins on the Arduino board. Using a Arduino test code on the computer to read out the value when different gestures were performed: Sweeping up-to-down by one sensing grid: value decrease by 1 unit. Sweeping down-to-up by one sensing grid: value increase by 1 unit. Sweeping left-to-right by one sensing grid: value increase by

	0.1 units. Sweeping right-to-left by one sensing grid: value decrease by 0.1 units. 4. Arduino is connected to the computer (running SimMan) through USB cable, a Arduino test code is going to test and display the value received from the SimMan software on the test computer. When change heart rate from 80 to 90, the reading pulling out from the SimMan is also going to update from 80 to 90. Outputs: The five pins for the data line of the 6-pin digital lines are shared by the 4 LCDs, while the 6th control line would be used to select between different LCDs to change display. 1. When one of the 'Heart Rate' 'Respiratory Rate/ awRR' 'Blood Pressure (Diastolic)' 'Blood Pressure (Systolic)' is being selected, the LCD_names1 is going to be selected, and the corresponding parameter is going to be marked with a little '*' beside it. 2. When 'Oxygen Saturation' is being selected, the LCD_names2 is going to be selected, and the corresponding parameter is going to be marked with a '*' beside it. (Three more parameters could be implemented for future expansion). 3. When one of the parameters was chosen, the LCD_values should

	be selected, and start to display the selected value. 4. When one of the parameters was chosen, the LCD_time should be selected, and start to display the selected value's evolution time. 5. When measured with a multimeter, the voltage drop between VCC 3V3 and ground is within the voltage tolerance. The voltage should remain within the tolerance both with no external devices hooked up to the microcontroller, and under full load. 6. When measured with a multimeter, the voltage drop between VCC 5V and ground is within the voltage tolerance. The voltage should remain within the tolerance both with no external devices hooked up to the microcontroller, and under full load.
Power Supply 1. Power supply should be regulated between 7-12 Volts, 1000±100 mA. 2. Power from wall should be 120 ± 12V AC	1. Plug the power cord into the wall, measure the output voltage use a Multimeter. It should read 9.0±0.9 Volts; measure the current using a Multimeter. It should read 1000±100mA. 2. Measure the voltage with a multimeter and check if it is within the acceptable bounds.
Software Interface 1. Parameter updates from the microcontroller should be received through the usb connection.	1. Test code will be used to display the values received from the microcontroller, without sending them to the Laerdal SimMan software. These values will be checked

2. Parameter updates from the microntroller should be properly parsed and communicated to the Laerdal SimMan software. 3. Pause signals from the microcontroller should be received through the usb connection. 4. Pause signals from the microntroller should be properly parsed and communicated to the Laerdal SimMan software.	against the input into the microcontroller to confirm that they are the same. 2. Test code will be used to pass parameters to the Laerdal SimMan software without input from the Microcontroller. These values will be checked both ends to verify that they are the same after the code executes. 3. Test code will be used to display when a pause signal is received from the microcontroller, without sending it to the Laerdal SimMan software. One pause signal sent by the microcontroller should correspond to one pause signal received by the software interface. 4. Test code will be used to send pause signals to the Laerdal SimMan software without input from the Microcontroller.