Spring 2016

Automated Syringe Filler

David L. Keister III
University of Akron, dlk60@zips.uakron.edu

Chad W. Borntreger
University of Akron, cwb18@zips.uakron.edu

Ethan M. Oglesbee
University of Akron, emo19@zips.uakron.edu

Zachary M. Stevens
University of Akron, zms4@zips.uakron.edu

Automated Syringe Filler

Design Team D:
Chad Borntreger
David Lee Keister III
Ethan Oglesbee
Zachary Stevens

Faculty Advisor:
Dr. Hariharan

Date: 4/22/2016

Table of Contents

List of Figures

List of Table

Abstract

The automated syringe filling system is a bench-top device designed to remove human error when filling a syringe. The microcontroller powered system will hold multiple medicine types that will fill a single syringe with a user specified amount of one medicine to a precise degree and log this information against a remote database of patient information. Before the system can be accessed, the user's credentials will be checked against the database of authorized users. Once verified a touch screen will be used to enter user input that will be logged to a remote database. The microcontroller will then control the actuators and servo to fill the syringe with the appropriate amount of medicine. The system will automate the process of filling a syringe and remove the element of human error from the syringe filling equation.

1) Problem Statement

1.1) Need

Medical professionals are required to fill syringes on a daily basis to administer medicine. Approximately 400,000 people in the U.S. die each year from preventable medical mistakes, many of which can be attributed to overdose or under dose. Many improper dosing errors are the result of human error while filling a syringe with medicine. If this process could be accurately automated the human error could be greatly reduced, thus potentially preventing a number of the above mentioned medical mistakes. A device that could reduce the number of medical mistakes and log all syringes filled in a database would be highly sought out by such facilities. [CB,DK,EO,ZS]

1.2) Objective

To design a microprocessor powered system that will hold multiple medicine types that will fill a single syringe with a specific amount of one medicine and log this information against a database of patient information. The system will take user input in the form of a medicine type, specific amount, and the patient the treatment is for. Once the input is received it will ready the selected medicine, take the sterile syringe, and fill it to the amount specified by the user. The system will have to perform this operation in a sterile manner and have an extremely small margin of error. Also, the system must be fast enough to not be inconvenient to the user and is designed to be small enough to sit on top of a lab bench or medical table.

The outer housing will be made from a Lexan material and will have a locking front door. Pilot holes will be drilled and small screws will be used to attach side walls and build the housing. In the top of this housing there will be a revolving tray that holds the vials of medicine. The tray only rotates on the horizontal plane and the vertical position is fixed. The vials will be loaded by opening the top of the housing and placing the vials upside down in their respective locations. The revolving tray will be driven by a servo motor. The servo motor will be controlled by the microprocessor.

Located below the rotating tray inside the housing will be the device that holds the syringe. A holding apparatus will hold the syringe while it is loaded in the machine. A micro switch located in the back of the loading area will determine when a syringe has been loaded or unloaded. This apparatus is mounted onto a plate; this plate is mounted onto a linear actuator. Mounted on this plate is the holding apparatus, and a high precision linear actuator that will be used to extract the plunger of the syringe. This actuator allows for the level of precision needed to fill the syringe to a near exact amount. We can calculate the pull distance based off of the dimensions of the syringe.

The entire system will be connected to a remote Apache Derby database of patient information and user credentials that will log all syringes filled against the patient they were for. If a user attempts to fill a syringe they must first select the patient for whom it is to be administered for. If the patient does not exist in the system they must be added before a syringe can be filled. The system will timestamp not only when the syringe was filled, but also when the syringe was removed from the system (via the micro switch). All of this data will be stored in the database. This will ensure a running record of all the filled syringes that left the system and their intended destination, helping guard against malpractice.

The system will remain locked until user credentials are verified, once they are verified the door will unlock. This ensures that only authorized users can access the patient information and the medications. The intention is to have the user input the information needed via a touch screen that is mounted on the system and is interfaced with the microcontroller and the Apache Derby database that would have a basic GUI. [CB,DK,EO,ZS]

1.3) Background

1.3.1) Patent Search

Patent US8671994 is a syringe filling apparatus. The difference between this and the automated syringe filler is that the syringe filling apparatus is filled manually while the automated syringe filler calculates the measurement of fluid to be extracted. The similarities are that the both contain a fluid reservoir, and one filling station. [1]

Patent US5911252 is an automated syringe filling system for radiographic contrast agents and other inject-able substances. This filling system can fill multiple syringes with desired volumes and concentrations. The pumping device engages each syringe and extracts the plunger to withdraw the desired amount of fluid. The mixing valve controls the concentration of the fluid. [2] [CB,DK,EO,ZS]

1.3.2) Article Search

This article refers to a reason that the proposed automated filling process would be a better solution than filling syringes manually. That reason being that syringe filling can be easily contaminated by the filler, therefore an automated approach may be a better, more sterile, solution.

This article details a similar design to the proposed idea and may serve to be a good basis for further research. The design in the article details a method for automated syringe filling involving a rotatable tray of syringes that get filled by a machine and can be controlled by a CPU. [CB,DK,EO,ZS]

1.4) Marketing Requirements

- The target audiences are medical facilities and laboratories, such as veterinarian, doctor, pediatrician, etc.
- Customers are likely to want to purchase this product, because it will eliminate the human error aspect in extracting the medicine.
- There are similar syringe filling devices in factory settings where large numbers of syringes are pre-filled. Our product is designed to fill one syringe at a time.

- The database of patient information coupled with the time logging functionality of the system could help with malpractice lawsuits as the exact time and amount of medicine administered to an individual patient can be logged. This event detection and data storage portion is what will set our system apart from any on the market. [CB,DK,EO,ZS]

2) Design Requirements Specification

2.1) Engineering Requirements

- The system must be able to fill a syringe with a variable amount of liquid with 0.1mL of precision.
    a. The above mentioned data base will be indexed off of the names of the Patients.
    b. The database must be able to be queried and can retrieve and store information on the individual patient level remotely.
- The system must be able to log the following information into a database: Name of Patient, Name of Filler, the amount of medicine distributed, the type of medicine distributed, the time the syringe was filled, and the time the syringe left the system.
- The system must be able to complete steps 1 and 2 within 30 seconds time worst case.
- The system must be able to select between 6 different medicine types.
- The system must have a display to convey information to the user.
- This display must be able to prompt the user for correct input and take in said input.
- The system must be able to operate off of 120 VAC power.
- The system must have a credentials validation step to only allow certified use of the system. [CB,DK,EO,ZS]

2.2) Level Zero Diagram

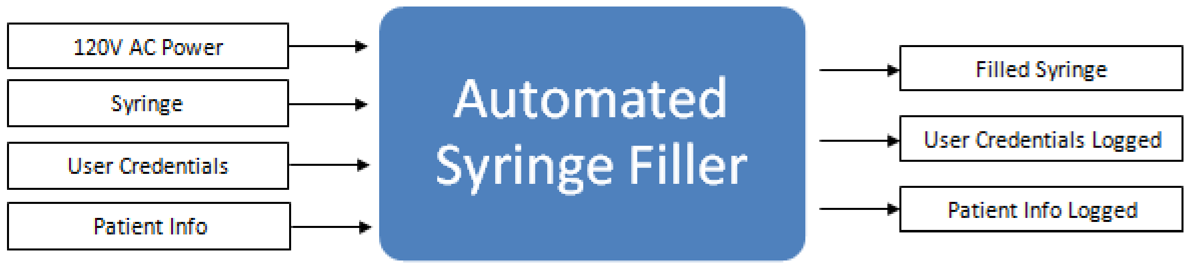


Figure 1: Level Zero System Diagram

Figure 1 shows the level zero system block diagram of the Automated Syringe Filler. With the use of four inputs there is a final result of three outputs. The final results are a filled syringe, as well as the patient and user credentials logged. These inputs and outputs will be broken down later on in the report, along with the specific duty of the Automated Syringe Filler. [ZS]

2.3) Level One Diagram

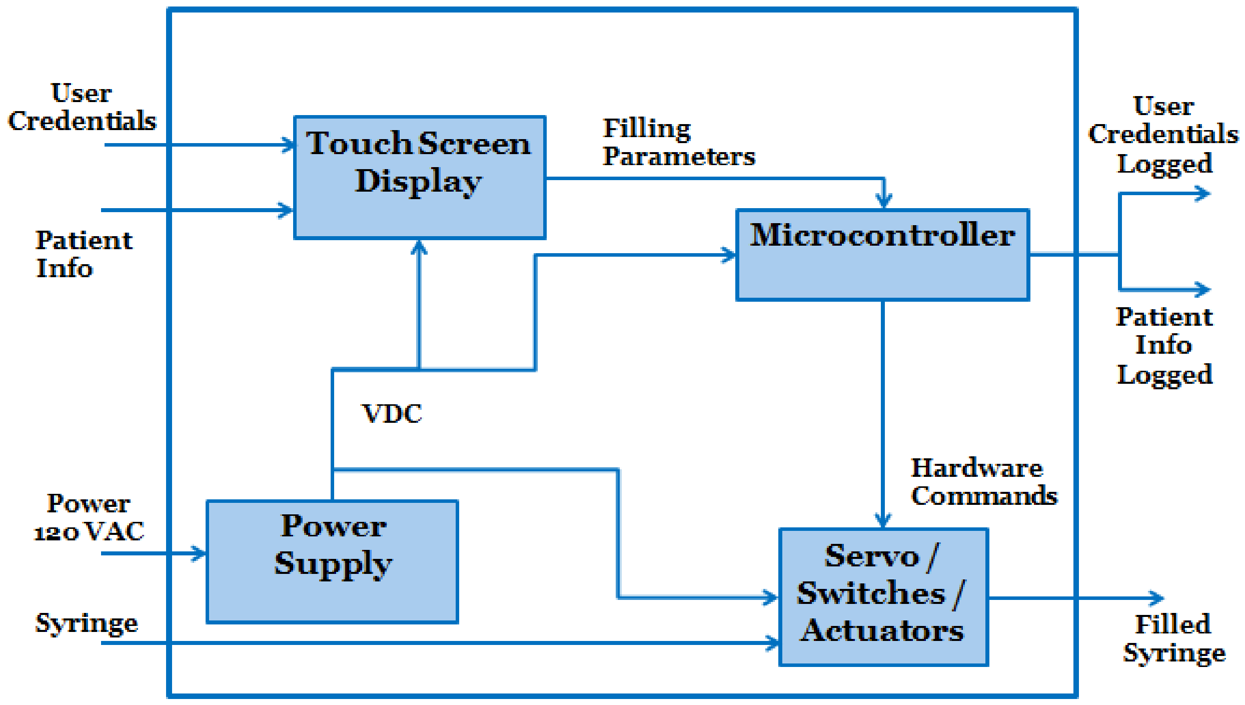


Figure 2: Level One System Diagram

Figure 2 shows the level one system diagram of the Automated Syringe Filler. This diagram shows the same inputs and outputs, with a general break down inside the automated syringe filler. This break down is further explained in the tables below. [ZS]

Module	Power Supply
Input	Wall Outlet: 120 VAC
Outputs	Regulated Voltage: 5 VDC and 12 VDC
Functionality	To take 120 VAC, and convert that AC voltage into DC voltages the system is able to use.

Table 1: Functionality of the Power Supply

Module	Touch Screen Display
Input	Power Supply: 5 VDC Patient Information User Credentials
Outputs	Filling Parameters User Credentials

Functionality	To serve an interface between the user and the system.

Table 2: Functionality of the Touch Screen Display

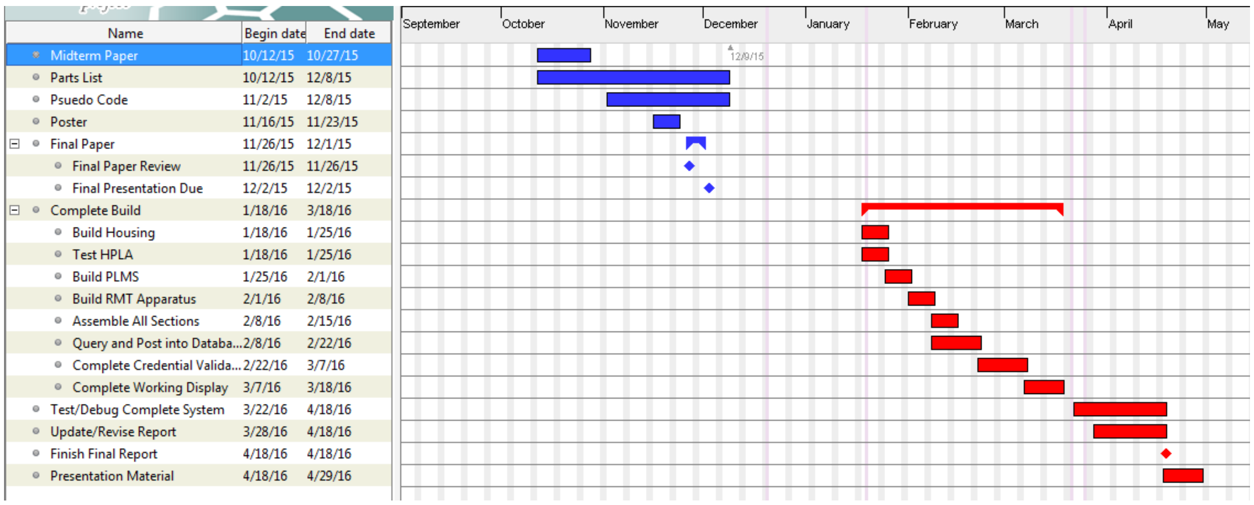
Module	Microcontroller
Input	Power Supply: 5 VDC Patient information Filling Parameters User Credentials Feedback from Servo, Switches and Actuators
Outputs	Positional commands to servo and actuators Lock and unlock commands to locking solenoid Patient info logged User Credentials Logged
Functionality	To translate the information it receives and command the motors to follow the filling parameters, and once the system is finished log all of the information.

Table 3: Functionality of the Microcontroller

Module	Servo, Switches, Actuators
Input	Power Supply: 5 VDC and 12 VDC Syringe Commands from Microcontroller
Outputs	Filled Syringe
Functionality	To take a syringe and fill it with the desired amount of medicine.

Table 4: Functionality of the Motors

2.4) Gantt Chart



Name	Begin date	End date
Midterm Paper	10/12/15	10/27/15
Parts List	10/12/15	12/8/15
Psuedo Code	11/2/15	12/8/15
Poster	11/16/15	11/23/15
Final Paper	11/26/15	12/1/15
Final Paper Review	11/26/15	11/26/15
Final Presentation Due	12/2/15	12/2/15
Complete Build	1/18/16	3/18/16
Build Housing	1/18/16	1/25/16
Test HPLA	1/18/16	1/25/16
Build PLMS	1/25/16	2/1/16
Build RMT Apparatus	2/1/16	2/8/16
Assemble All Sections	2/8/16	2/15/16
Query and Post into Databa...	2/8/16	2/22/16
Complete Credential Valida...	2/22/16	3/7/16
Complete Working Display	3/7/16	3/18/16
Test/Debug Complete System	3/22/16	4/18/16
Update/Revise Report	3/28/16	4/18/16
Finish Final Report	4/18/16	4/18/16
Presentation Material	4/18/16	4/29/16

Figure 3: Gantt Chart

Figure 3 is a Gantt Chart of the expected timeline of the Automated Syringe Filler. The blue color designates the design portion of the project; while the red represents the implementation of the project. All of these are projections and are subject to change during the process of the design project. [ZS]

3) Accepted Technical Design

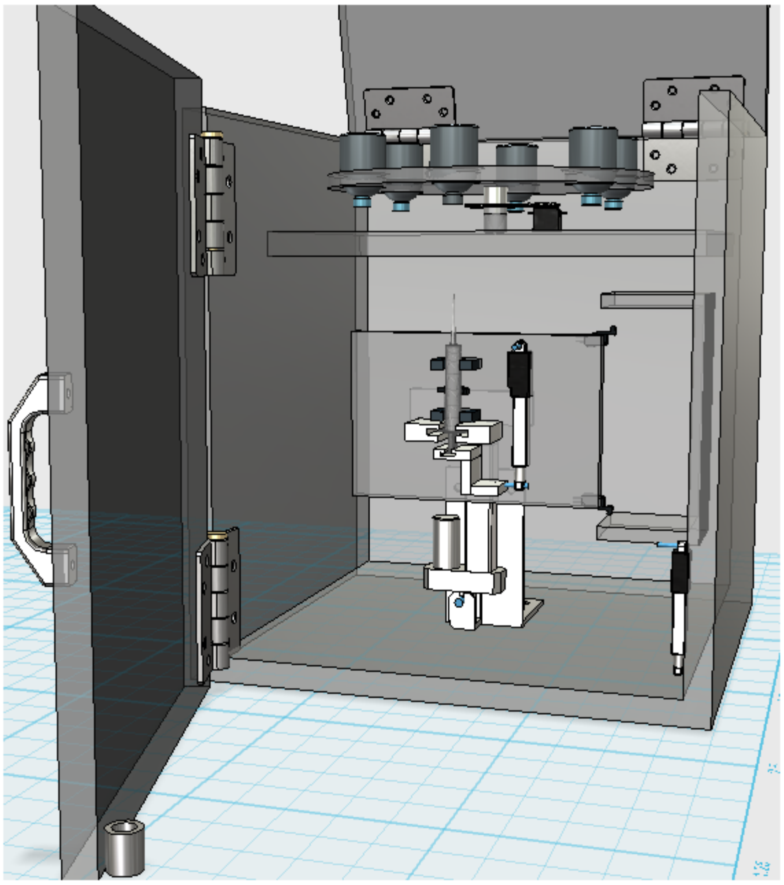


Figure 4: 3D Model Front View

3.1) Primary Linear System

The primary linear motion system consists of a large linear actuator mounted to the center of the base of the housing. The Primary Linear Actuator (PLA) can provide 6 inches of travel that will drive the syringe housing plate upward, thus inserting a preloaded syringe into the selected awaiting vial above. Once the syringe has been filled the PLA will reverse direction and retract, removing the syringe from the vial.

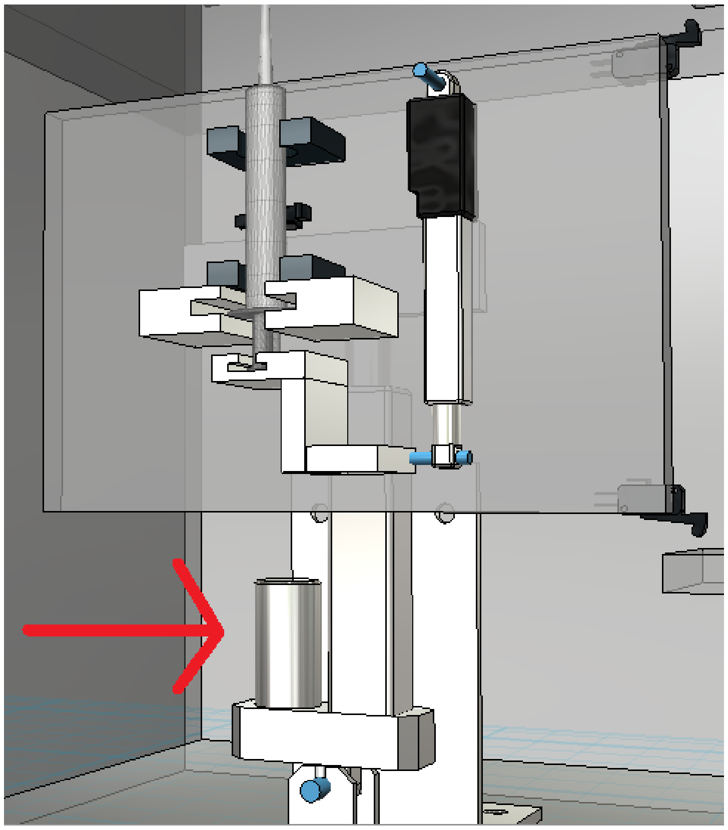


Figure 5: 3D Model PLS

Figure 5 highlights the PLA and how it will be mounted to the base with the supplied hardware. The supplied hardware uses a pin and carriage type mount. The pin and carriage mount eliminates any vertical movement but allows the linear actuator to rotate back and forth in both the clockwise and counter clockwise directions. By using an “L” shape bracket from behind the rotational movement will be eliminated. The “L” shaped bracket is also shown in Figure 5.

The LIN-ACT1-06 made by WindyNation, is the actuator that will be used in this system. Rated at 12 VDC, this actuator is capable of lifting up to 225 pounds which well exceeds the requirement for this application. With a max travel speed of 10mm per second, it will provide the short travel time that is desired. By minimizing the travel time of the PLA the overall cycle time of the syringe filling system can be kept to a minimum.

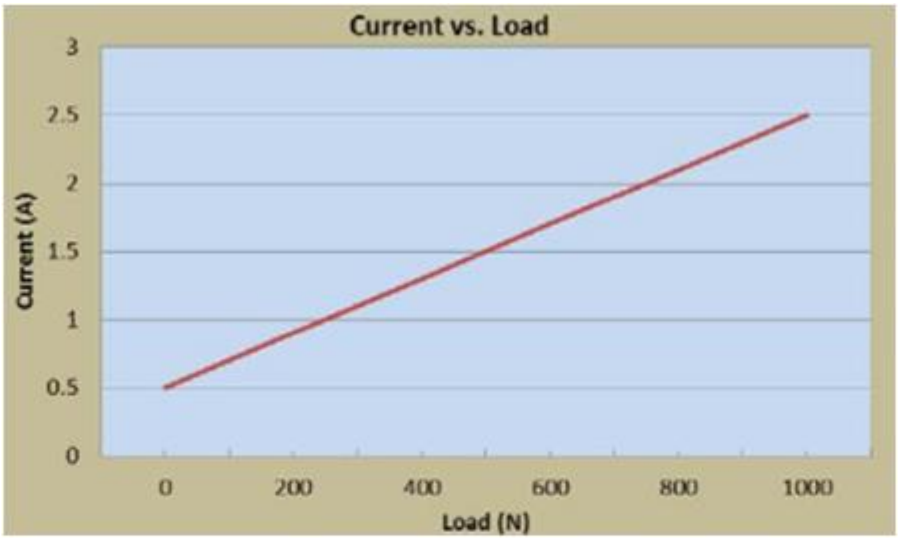


Figure 6: Current vs Load PLA

Error! Reference source not found. shows the current vs. load curve of the PLA. By determining the load, the amount of current that will be drawn can be calculated. The exact weight the PLA will be required to lift depends on multiple variables such as selected materials, weight of these materials, and opposing forces. The maximum estimated weight of the1/2 inch Lexan plate, rubber syringe holding device, high precision linear actuator (HPLA), and the components that sense when a syringe has been loaded is 15 lbs.

Using Newtons second law of motion the force required by the PLA to lift 15 lbs can be found. Using an estimated gravitational acceleration of 9.8 m/s^2:

$$F = M * A \qquad (1)$$

$$F = 6.8kg * 9.8m/s^2$$

$$F = 66.64N$$

From **Error! Reference source not found.** it can be seen that approximately 0.6 Amps will be the maximum current drawn by the PLA.

The PLA will be controlled via a combination of three components: a microprocessor, a multi-stage relay board, and a pair custom limit switches. Utilization of two digital output pins from the microprocessor will trigger a series of relays on the relay board to control power to the linear actuator. Section 3.6 discusses the design of the relay board in depth along with a wiring diagram of the board. If the control signal to the relays is lost, they will fail to an “off” state, causing the PLA to stop abruptly and not drift to a stop. To achieve the desired length of extension and retraction a custom set of limit switches will be implemented. [CB]

3.1.1) PLA Limit Switch Design

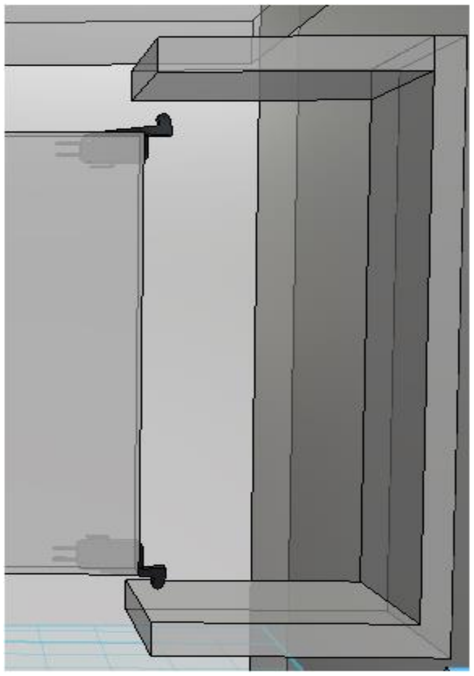


Figure 7: PLA Limit Switch Design

Since the PLA does not have a built in feedback signal, a custom set of limit switches will be used to indicate when the upper and lower limits have been reached. As shown in Figure **7** a ridged "C" shape bracket will be mounted to the side wall of the housing providing an upper limit shelf and a lower limit shelf. The limit switches will be attached to the right hand side of the syringe housing plate. As the PLA extends and retracts the syringe housing plate with attached limit switches will move up and down with it.

As the PLA is extended the syringe will move upward and the needle will pierce the rubber mouth of the vial, at this point the upper limit switch will contact the upper limit shelf sending a signal back to the microcontroller to stop the PLA and hold its position. The same holds true for when the PLA is retracting and reaches the lower limit shelf. Figure 8 shows how each limit switch will be wired to achieve the desired performance. [CB]

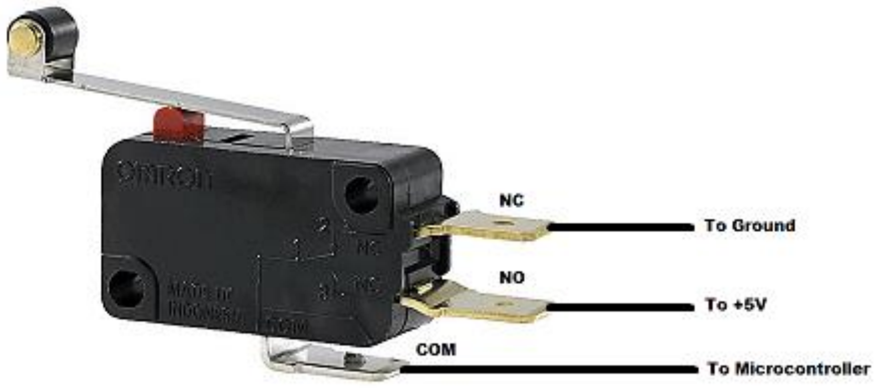


Figure 8: PLA Limit Switch Wiring

3.2) Syringe Filling System

To detect when a syringe has been inserted into the machine, a micro switch will be used. The micro switch will be mounted to the back of the syringe loading location. Until the microcontroller receives feedback from the micro switch, the filling process cannot be started.

In order to accurately fill the syringe a High Precision Linear Actuator (HPLA) will be used to extract the plunger. The HPLA will attach to the plunger via a “Z” shaped bracket. This was done so that the syringe would be mounted higher than the HPLA, in order to avoid obstructions between the syringe housing plate and the equipment above it. The HPLA is equipped with an 18,000 Ohm potentiometer, providing analog position feedback for closed loop control. The Firgelli L16-P is the HPLA that will be used. The L16-P is rated for 12V and will be controlled via a Firgelli Linear Actuator Controller (LAC).

By pairing this actuator with a LAC, control of more than just position is feasible. Characteristics such as speed, sensitivity, and stroke limits are all adjustable through the LAC interface software. The LAC can be configured as a standalone controller or it can be used as an interface board between a microcontroller and the HPLA. The optimal solution is to use it as an interface board and handle the control of the actuator within the microcontroller.

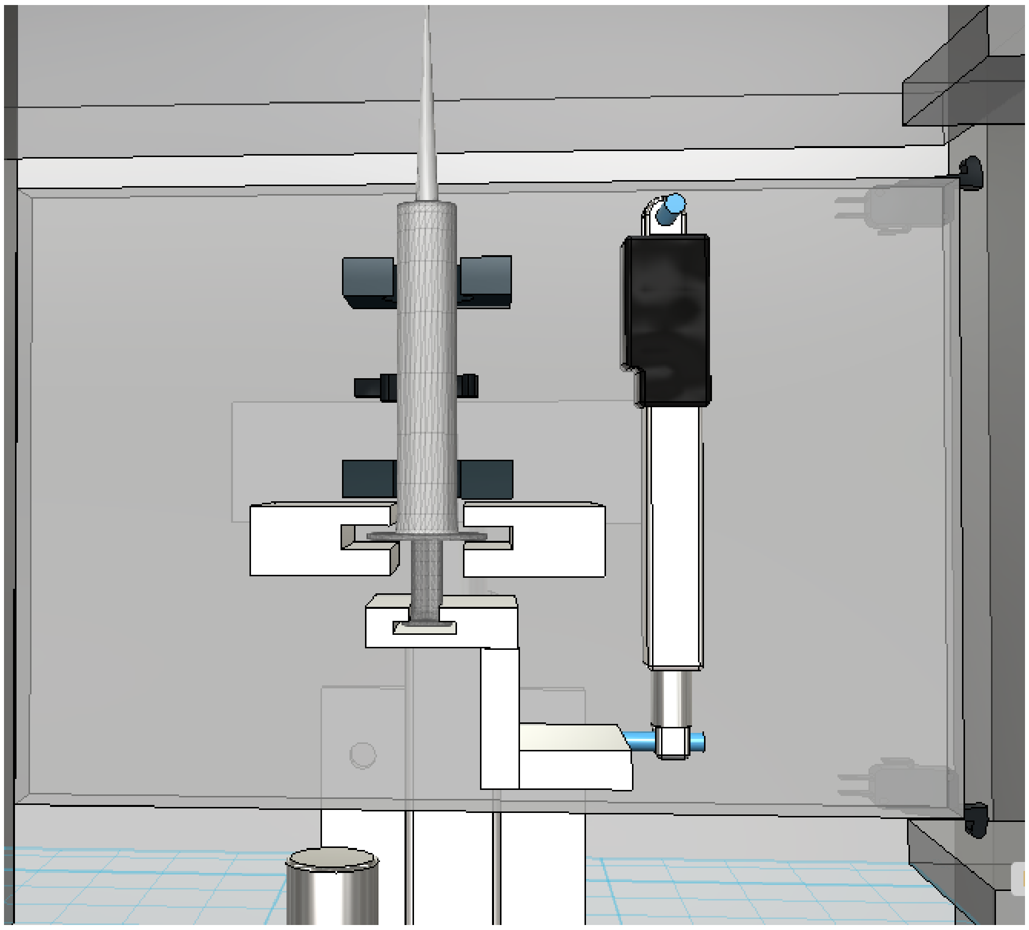


Figure 9: HPLA Close Up

The L16-P HPLA is available with three different gearing options to allow for either a fast acting actuator or a slower actuator capable of greater forces. The median option with a gearing ratio of 63:1 is best suited for this application. The 63:1 gearing provides a max lifting force of 100 Newtons and a max back drive force of 46 Newtons. The estimated force to extract the plunger of the syringe is between 10 and 20 Newtons. From the current vs. load curve shown in **Error! Reference source not found.** (Green Line for gearing of 63:1) the current draw will be between .1 and .2 Amps.

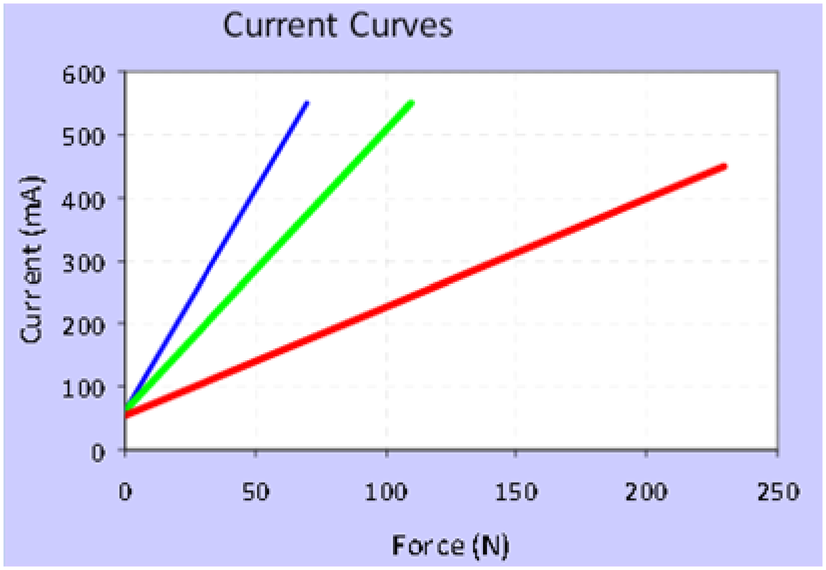


Figure 10: Current vs. Load HPLA

From Figure 11below it can be seen that two black rubber semi-circle forms will be utilized to hold the syringe to the syringe housing plate. One will be located near the top of the syringe and another near the bottom. As the syringe is loaded the user will receive physical feedback that the syringe has been “snapped” into place both from the top form and the bottom form. These rubber forms will help hold the syringe in place and eliminate any movement while filling is in progress. [CB]

3.3) Revolving Medicine Tray

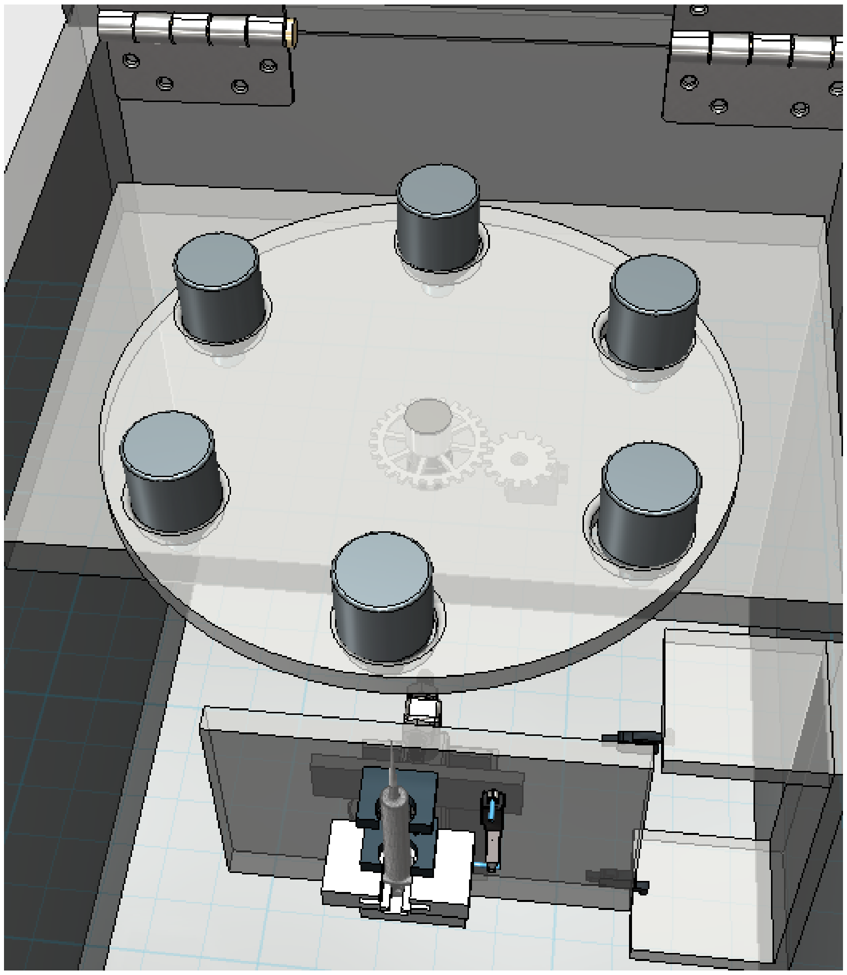


Figure 11: 3D Model Top View

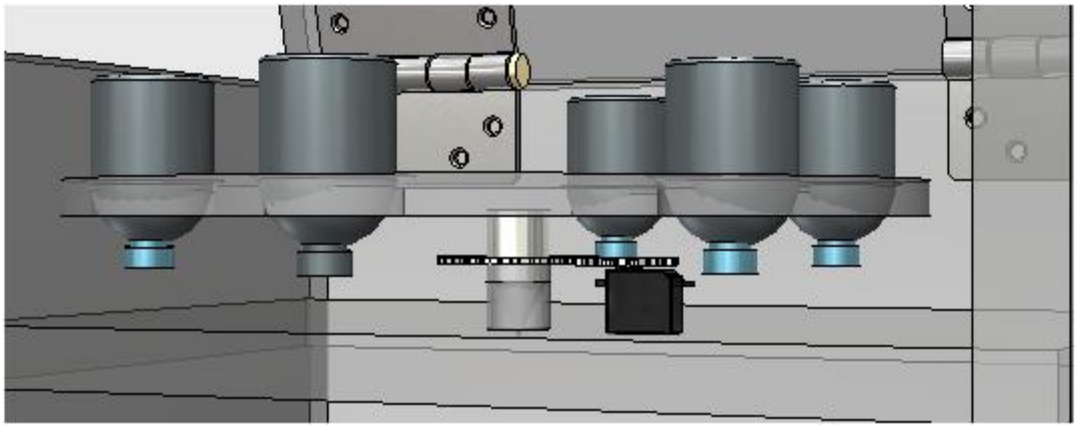


Figure 12: 3D Model RMT Close Up

The Rotating Medicine Tray (RMT) will be controlled by a digital servo motor which will be connected to the shaft holding the tray via two gears. The top of the shaft will connect to the tray, and the bottom of the shaft will rest on the tray base which is shown in Figure 12. The bottom of the shaft will be milled to a point; this point will rest in a circular divot that will be drilled into the surface of the tray base. This will provide a single point of rotation for the shaft while minimizing the frictional resistance. To keep the RMT from tilting side to side a sleeve that fits around the RMT shaft will be fixed to the tray base. The inside diameter of the sleeve will match the outside diameter of the shaft. The tray base will extend outward 9inches from the back wall, leaving 4 inches of space to expose one medicine vial to the syringe located below.

The RMT will contain six 10mL medicine vials filled with 6 different medicines. Each vial will have a slot, 60 degrees apart, in the circular tray. The slots will have a hole big enough to only fit the neck of the vial. Above the slot will be a washer that will be beveled to fit the dimensions of the vials body. Each vial will then be held down by a rubber strap, the rubber strap will be attached by two bolts on each side of the vial slot. The use of the rubber strap will make for easy insertion and removal, as well as great support for holding the vials down while being penetrated and revolved.

The servo motor that was chosen is a digital servo rather than an analog servo due to the precision of the dead band. Calculations were done to determine the precision of each motor. The radius of the tray will be a 3 inches to allow space for all 6 vials to fit.

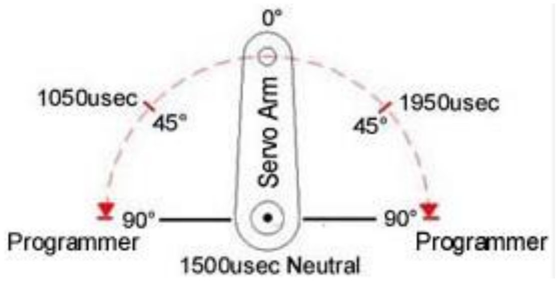


Figure 13: Servo Motor Rotation Diagram

From the times shown in Figure 13, the servo motors take a 1950µs pulse width to move to a 45° position. In order for the system to work properly, the servo will need to be able to position itself within a 3mm window. This is the diameter of the rubber mouth of the vial that the syringe will puncture. The ratio that was used is

$$\frac{45°}{1950\mu s - 1500\mu s} = \frac{1°}{10\mu s} = \frac{a}{n}; \qquad (2)$$

where *a* is the angle of movement and *n* is the pulse width it takes to move that angle. For each of the servo motors there is a dead band that the servos can't operate within. This is the shortest pulse width the servo can output in order to get any movement. The analog servo has a dead band of 8µs. With the use of equation (2), the angle of the dead band is able to be calculated. For the analog servo the value of *a* is 0.8°. This value is then put into equation (3) to give us a value of x, which is the linear approximation the arc length, in millimeters, of the angle *a*. All of the same calculations were done for the digital servo.

$$\tan(a) = \frac{x}{r} \qquad (3)$$

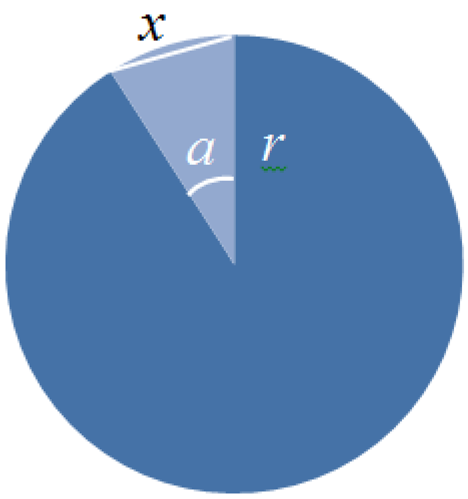


Figure 14: Linear Approximation Diagram

Servo Motor	For r = 3in
Analog	a = 0.8° , n = 8μs , x = 1.064mm
Digital	a = 0.2° , n = 2μs , x = 0.266mm

Table 5: Servo Motor Dead Band Width Distance

The servo that was chosen only has a range of motion of 90° as shown in Figure 13. For the application of the system a rotation of 300° is needed to reach all 6 vials on the RMT. To reach this required angle a 4:1 gear ratio is needed. The gear attached to the servo spline will have 96 teeth with a 48° pitch and the gear attached to the shaft will have 24 teeth with a 48° pitch. This in turn will increase the servo dead band width distance by a factor. These calculations are shown in Table 6.

Servo Motor	For r = 3in
Analog	a = 0.8° , n = 8μs , x =4.256mm
Digital	a = 0.2° , n = 2μs , x = 1.064mm

Table 6: Servo Motors after Gearing

3.4) Syringe Filling System Housing

The syringe filling system will be enclosed in a cubic housing. The housing will be 24 inches tall by 12 inches wide by 12 inches deep. The material of the housing will consist of two different materials. The first material is ¼ inch Lexan polycarbonate clear sheets, and the second material will be plywood sheets. The Lexan will be used for the front door and the right wall;

this will allow the user to view the system while it is operating. The plywood will be used for all of the remaining structural walls. In an ideal situation the entire structure would be made of Lexan, but due to cost the Lexan will only be used for major viewing areas. If the design team can get Lexan donated, the entire structure will be made from Lexan.

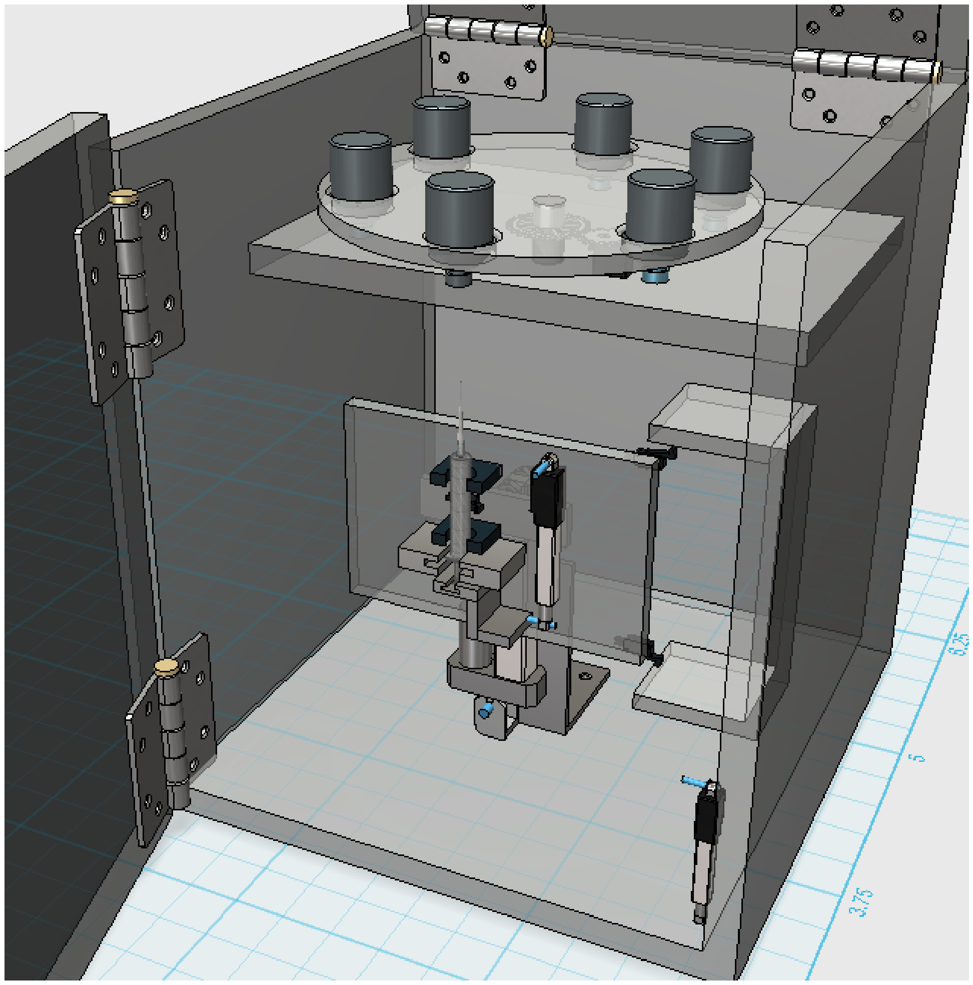


Figure 15: 3D Model Corner View

The housing will be split into different sections; such as the base, front door, top door, and side walls. The base is where the PLA will be mounted. As mentioned earlier, the PLA will be mounted using an “L” bracket for support. The PLA will hold the HPLA which will be attached to the syringe housing plate, a ½ inch Lexan. A model showing the Lexan plate is shown in Figure 15.

The front door will allow the operator access to insert and remove the syringe. The front door will have a handle for ease of use, as well as a locking mechanism. The locking mechanism that is being used is a DSOL-0844-12, which is a 12V DC open frame pull solenoid made by Delta Electronics. The lock will be toggled by the relay board. When the solenoid is powered off or de-energized, this lock is extended, locking the system. Therefore when the solenoid is powered on or energized, the lock is retracted, unlocking the system; which in turn anytime the system is unplugged or not in use the system will be locked. The top door was created so the operator would have an ease of access to the revolving medicine tray. The top door will not have a lock, but instead will only have the ability to open if the front door is opened. The top door will have a notch beveled out of the plywood on the side portion of the top door; this is more easily shown in Figure 16 below.

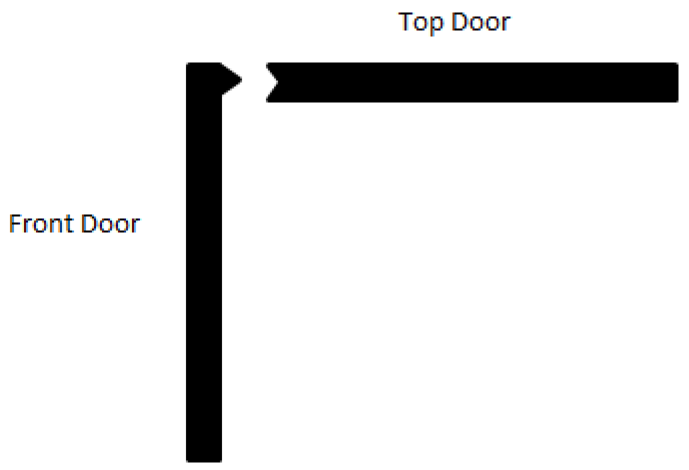


Figure 16: Top Door Locking Diagram

The notch will allow the front door to fit in the top door like a puzzle piece would fit together; which will lock the top door any time the front door is closed. The side walls are there to ensure the entire system is enclosed and when the doors are locked nothing inside is able to be accessed. All of the housing parameters are shown in Figure 8. [ZS]

3.5) Level Two Power Supply Design

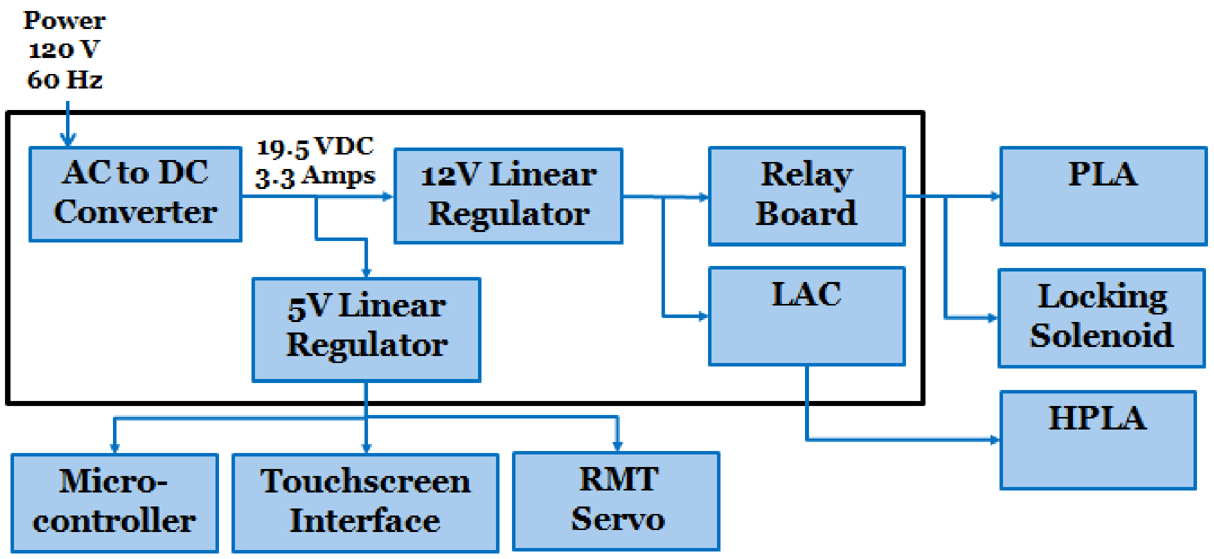


Figure 17: Level 2 Power Supply Diagram

Powering the Automated Syringe Filler through a 120 VAC, 60 Hz wall outlet yields the most practical solution based on the application. A laptop charging cord will convert 120 VAC to a usable DC Voltage. The laptop charging cord is rated at 19.7 Volts and 3.3 Amps, providing approximately 65 Watts of power. The electronic components of the Automated Syringe Filler operate off of 5 VDC and 12 VDC shown in Figure 17. [CB]

3.5.1) Linear Regulators:

The 12V linear regulator is made by Micrel and is capable of accepting input voltages of up to 26V. It is rated to output 12V at 3A. Depending on which devices are active downstream of the linear regulator will determine the current draw from that regulator. These downstream devices will draw less current than the 3A maximum output of the linear regulator. The linear regulator has three pins, Vin, Vout, and ground. Vin will be connected to output of the AC to DC converter. Vout will supply power to the relay board and the linear actuator controller.

Texas Instruments makes the 5V linear regulator shown in Figure (13). It is capable of accepting up to 25V input and outputting 5V at 3.2A. The 5V branch of Figure (13) supplies the microcontroller, touch screen, and RMT servo. The servo will draw substantially more current than the microcontroller and the touch screen. If all three devices are active, they will still not exceed the current rating of the linear actuator.

3.6) Relay Board Design

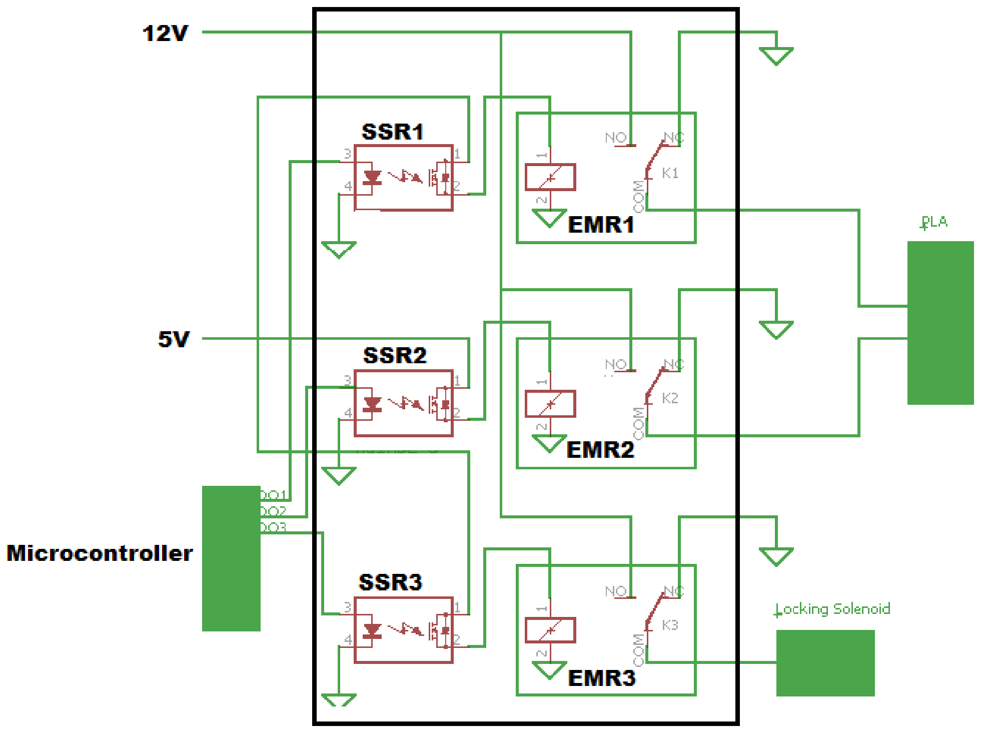


Figure 18: Relay Board Schematic

The Relay board shown in Figure (14) will act as a bridge between the microcontroller and the higher powered devices such as the PLA and the locking solenoid. By utilizing a multi-stage relay design the microcontroller will be capable of toggling the high powered devices on and off. The first stage of the relay board consists of three solid state relays (SSR), labeled as SSR1, SSR2 and SSR3 in Figure (14). The second stage of the relay board is comprised of three electromagnetic relays, labeled as EMR1, EMR2 and EMR3. The EMRs require more power to switch on and off than an SSR. The microcontroller can't output enough power to toggle the EMR's, hence the multi-stage relay design is needed.

The microcontroller is capable of outputting 3.3V at 10mA which is enough to turn on the LED within the SSR. A photovoltaic unit also inside the SSR housing senses the emitted light and turns on the MOSFET, thus completing the circuit that will energize the second stage of relays. The 5V circuit completed by the SSR will energize the coil of the EMR switching it on. By controlling the EMRs, the PLA and locking solenoid are also being controlled.

The multi-stage relay design was not the only option considered while designing the relay board. The other design that was considered was similar but instead of using a solid state relay to trigger the EMR's a transistor would be used. The downside to this option is that it requires a more complex design to complete the same task. To switch the transistor on and off an op-amp would be required between the microcontroller and the transistor to amplify the signal to be larger than the threshold voltage of the transistor. Another consideration is that EMR's are electrically noisy devices so some isolation options would need to be considered. [CB]

3.6.1) Relay Simulation

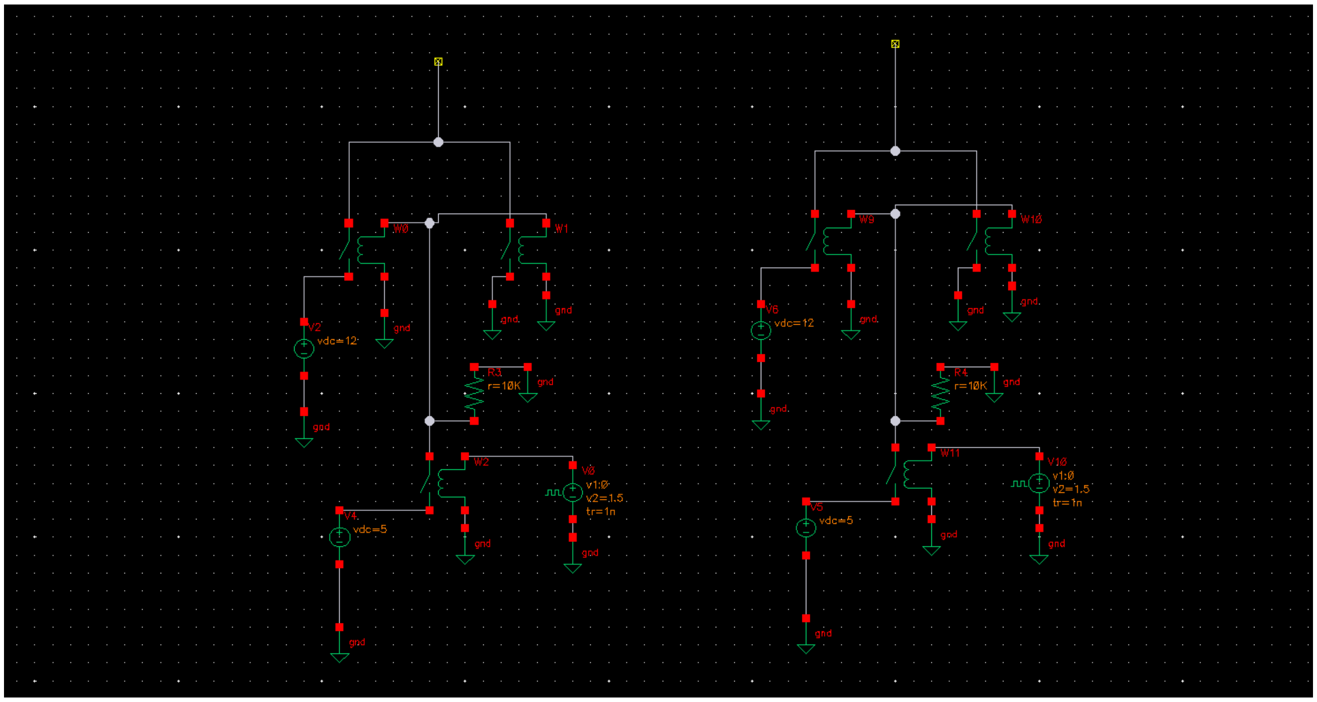


Figure 19: Relay Design using Cadence

Figure 19 shows the design of the relay board using Cadence. Cadence was used to simulate the relay circuit in order to prove that the circuit that was created will work. The design in Figure 19 is excluding SSR3 and EMR3; this is because the design, and simulation, of the relays controlling the PLA will prove the concept is functional. The simulation design shows two separate 5 VDC sources. This was done for simulation purposes only; the 5 VDC is the same 5 VDC source for both sets of relays. The same goes for the 12 VDC sources shown in Figure **19**. The two Vpulse signals were created to simulate each possible situation the microcontroller could produce for the PLA. The first relays, starting at the bottom of Figure **19**, are SSR1 and SSR2. These relays will take Vpulse signal and toggle on or off if the relay receives a 1.5 VDC signal or not. The SSR1 and SSR2 will output the 5 VDC to toggle the EMR1 and EMR2 relays. Figure 20, below, shows the results of the relay design simulation.

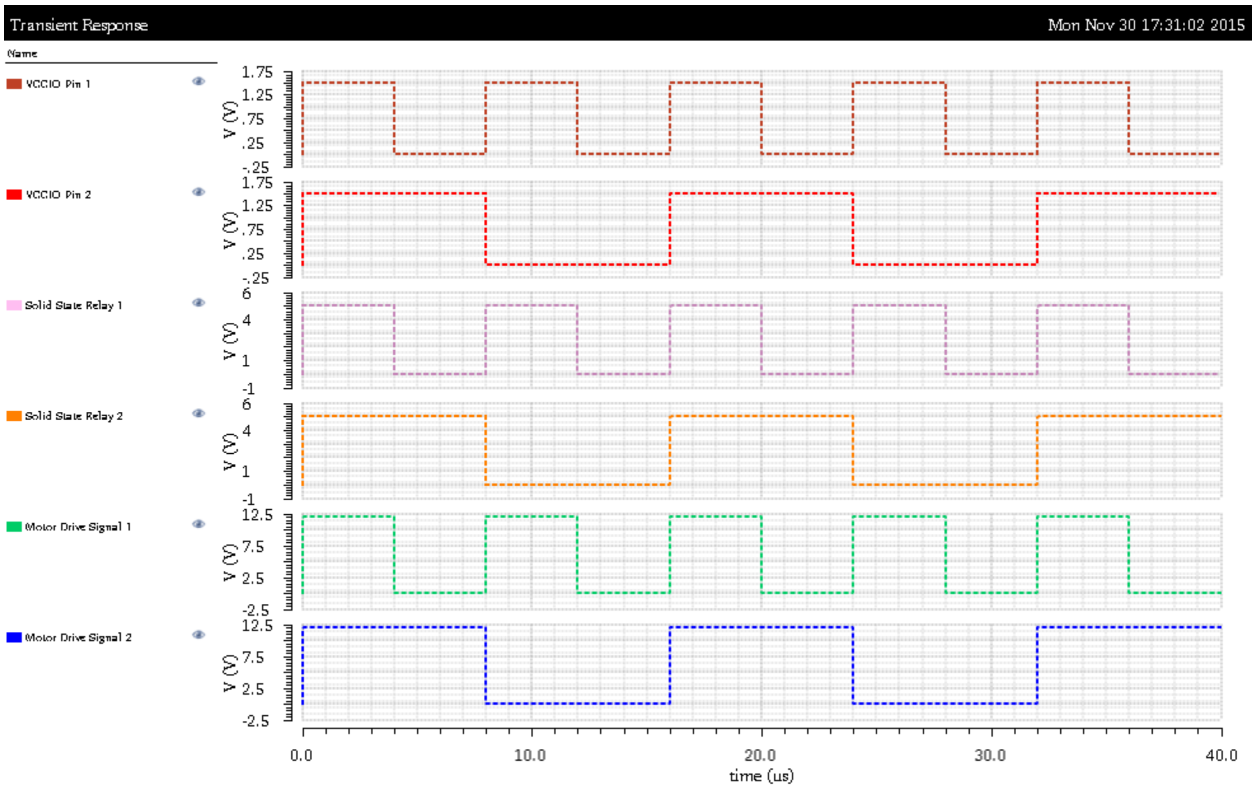


Figure 20: Relay Simulation using Cadence

The brown and red waveforms in Figure 20 represent the two digital outputs of the microcontroller. For control purposes of the PLA only one digital output of the microcontroller will be active at a time. Therefore the first 4us of the waveforms can be ignored as this is an invalid state. Further examining the waveforms between 4us and 8us it can be seen that pin 1 is low while pin 2 is high. The purple and yellow waveforms are the output of the solid state relays, they correspond with the outputs of the microcontroller proving that the relay has been switched. The last two waveforms (green and blue) represent the leads that will be connected to the motor, and it can be seen that there is a (- +) polarity on the motor. This will result in an extension of the PLA.

Looking at the waveforms between 8us and 12us, pin 1 is high while pin 2 is low. This yields a (+ -) polarity on the motor. This polarity will cause the PLA to retract. Examining the waveform from 12us to 16us will provide information on the third and final possible configuration where pin 1 and pin 2 are both low. With both solid state relays turned off there is a (- -) polarity applied to the leads of the motor, resulting in no power being supplied to the motor. Between 16us and 40us the waveforms repeat and the previously stated information still holds true. An interlock will be implemented in the programming to prevent digital output pins 1 and 2 from both being active at the same time, ensuring that the motor will not receive a (+ +) polarity resulting in a damaged or burnt out PLA. [CB, DK, EO, ZS]

3.7) Level Two Servo/Switches/Actuators Design

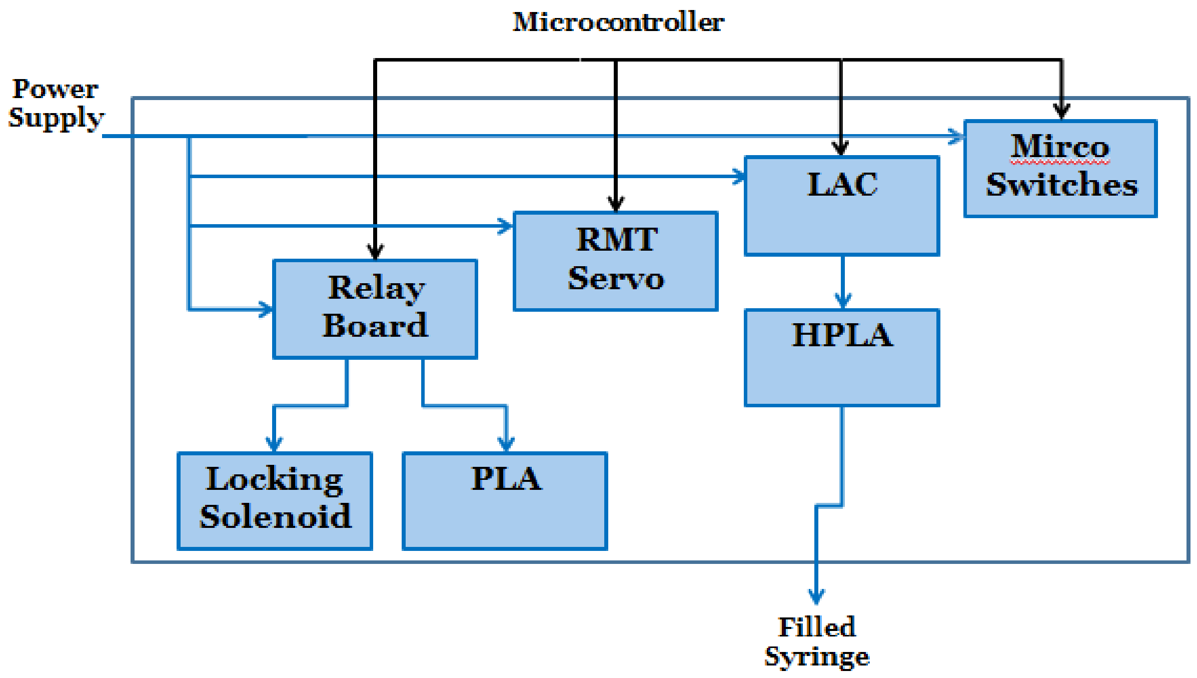


Figure 21: Level 2 Servos/Actuators/Sensors Design

The design of the servo motor, actuators, and sensors is shown in Figure 16. The power supply comes in to the relay board, HPLA, and micro switch. The locking solenoid and PLA will be powered through the relay board. The relay board will also provide the signal, from the microcontroller, to the locking solenoid and PLA. The other signals will go directly to the RMT servo, HPLA, and proximity sensor. The end result will be the filled syringe, which will come from the HPLA. [ZS]

3.8) User Credentials and Patient Information Database

The User Credentials and Patient Information Database (UCPID) will be a remotely hosted database consisting of two tables; a User Credential Table (UCT) and a Patient Information Table (PIT).

The database will be implemented using Apache Derby in Java. This database software was chosen because it allows for concurrent remote writes and queries to the database without having to worry about race conditions. It also allows for relational tables and robust querying which may be needed if the records are to be checked.

The database was chosen to be hosted remotely for several reasons. The primary reason is that if the database was hosted on board the machine then each database in a network would have to be updated individually. Using the remote database, the UCT only has to be updated on the remote database for those changes to propagate to each machine on a network. This will prevent users who have been revoked of access from accessing machines that may have not been updated yet. The remote database also removes the restrictions on size that come with hosting a database on the machine. The remote database method allows for a more robust database software that allows for relational tables and more robust queries. All important information is also stored remotely, which can help to mitigate security issues as the remote database can be stored in a secure area.

The UCT will be used as the table to query against to validate that appropriate users are accessing the machine and that the machine is to be properly unlocked. The table will store the identification numbers of authorized users. The UCT will not be remotely editable from the Automated Syringe Filler to prevent an unauthorized user from putting their information into the table.

The PIT will be used to log data from each time the machine is accessed. The table will allow for the record keeping of access to the machine. The table will include information about which medicine is being withdrawn, the amount, the user that is doing the withdrawal, the unique identification number of the machine accessed, and the intended recipient of the medicine. The PIT will be able to be queried against, and written to, but will not allow for records to be remotely deleted to prevent tampering with records. [DK]

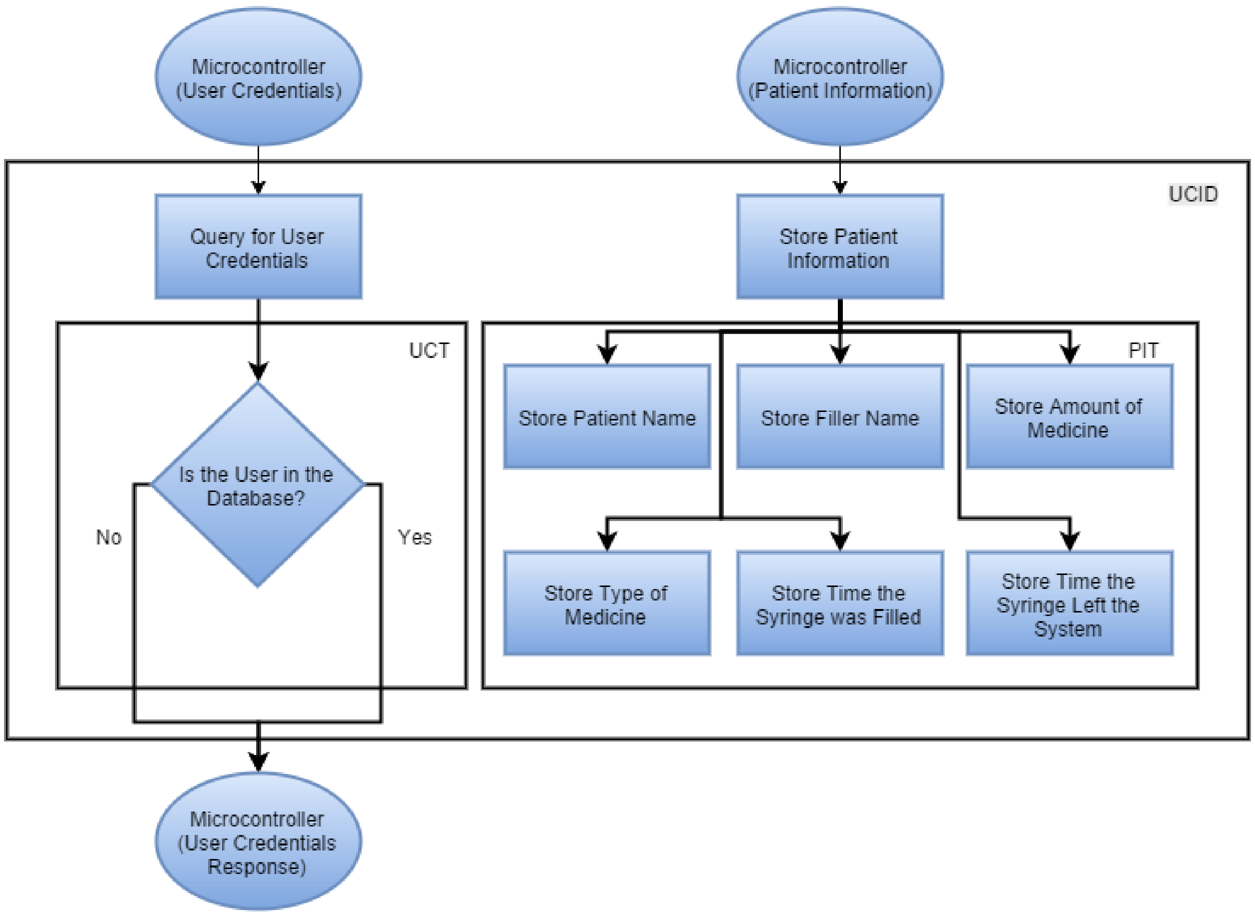

Microcontroller (User Credentials)
Microcontroller (Patient Information)
UCID
Query for User Credentials
Store Patient Information
UCT
PIT
Is the User in the Database?
No
Yes
Store Patient Name
Store Filler Name
Store Amount of Medicine
Store Type of Medicine
Store Time the Syringe was Filled
Store Time the Syringe Left the System
Microcontroller (User Credentials Response)

Figure 22: User Credentials and Patient Information Database Diagram

3.9) Microcontroller

The Automated Syringe Filling system will utilize a TI Hercules launchpad LAUNCHXL2-RM46 board as its microcontroller. The board contains a 32-bit ARM Cortex-R4F CPU that operates at 220 MHz and has 192 KB ECC RAM and 1.25 MB ECC Flash as well as timing peripherals for motor control support. It will be programmed in Embedded C using the Code Composer Studio (CCS) integrated development editor and will control all of the functionality of the system. The board runs off of a 5v supply voltage and can output 3.3v via it's VCCIO pins.

This board was chosen as it contains all of the functionality the system needs in a single package with a low cost factor of around twenty dollars. Not only does the board contain the features listed above but it has a few more specific features that make the RM46 ideal for our purposes. The board has 10/100 Ethernet MAC which will allow the system to communicate with the remote database, and has two SPI modules which will allow communication with the display unit. The microcontroller also has a PWM unit which will allow for control of the HLPA and servo motor. Also, it has on board emulation which will allow for debugging software on the board without any addition tools. This feature could prove beneficial for this project as it could help streamline development to maintain the schedule.

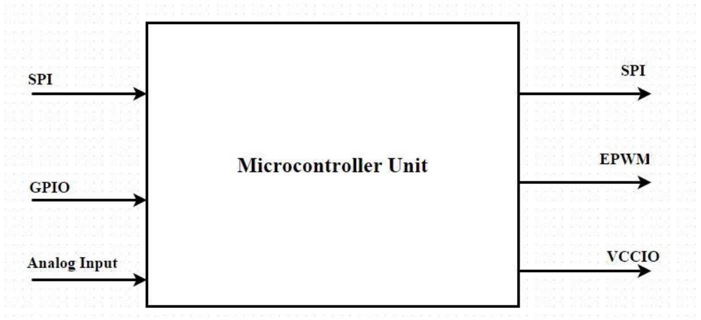


Figure 23: Level 1 Microcontroller Design

The Microcontroller must be able to handle 3 different input types; an SPI interface, GPIO (General Purpose Input Output) and Analog Input. Alternatively there will also be three output types; a SPI Interface, EPWM (pulse width modulated signals) and output supply voltages of 3.3 volts provided by the board's VCCIO pins. The SPI Interface will be used to send data and commands to the LCD Display as well as monitor feedback from the display. The GPIO inputs will consist of three pins that monitor feedback from the upper contact, the lower contact, and the syringe sensor. A single input pin from the Analog to Digital Conversion Unit will be used to monitor the analog feedback from the LAC. There will be two EPWM signals sent out from the microcontroller; one will be a 3.3v 1kHZ square wave sent to the LAC that will be used to set the position of the HLA, and the second PWM signal will be used to set the position of the RMT servomotor via varying the pulse width of the signal. Lastly three VCCIO pins will be utilized, two of the three will toggle the relays that drive the PLA, and the third will toggle the relay that energizes the locking solenoid. A pinout diagram for the microcontroller can be seen below in Figure 18. [EO]

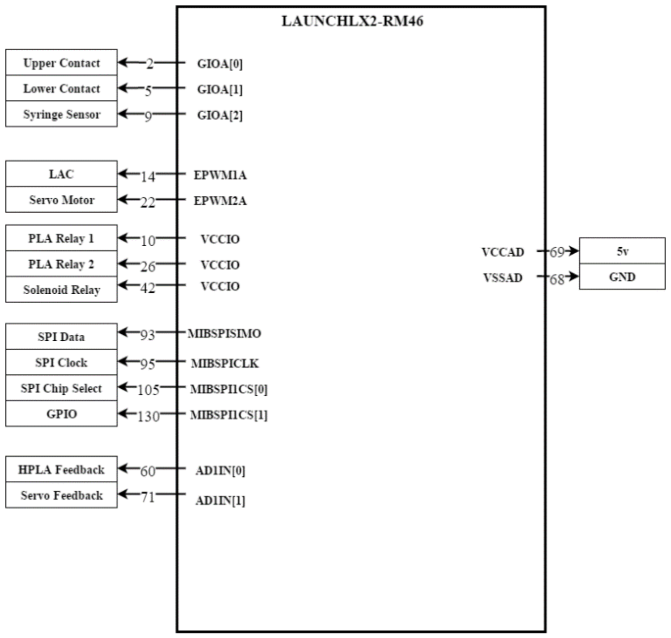


Figure 24: Microcontroller Pinout Diagram

3.9.1) Microcontroller Software

Figure 25 below shows a level 2 depiction of the software flow for the microcontroller. When idle, the system will constantly loop with the LCD displaying a pin pad to the user awaiting a user ID number to be entered. Once an ID number has been entered the system will construct a query comparing the value against the user table found in the remote database. If the user ID is not found that it will be thrown out and the cycle awaiting a new user ID will begin again.

Once a valid user ID has been entered a 3.3v voltage will be sent across pin 42 toggling the door solenoid’s relay, unlocking the door. When the door is unlocked the general IO pin 9 will be monitored, waiting for an empty syringe to be placed into the system by the user. If a syringe does not enter the system before a designated timeout period ends, then the voltage to the door solenoid relay will be cut, locking the door and the user credentials loop will start over again. If a syringe is entered the door is also locked and the user is prompted for the filling parameters via the LCD touchscreen that will be explored in more detail further in this document. The user will be prompted to enter in the following information:

- Patient Name / ID
- Type of medicine to be administered
- Amount of medicine to be administered

The next step once the filling parameters have been received is the RMT needs to be put into the correct position based off of the chosen medicine type. The servo that drives the RMT takes in a PWM signal that maps rotational position to signals with varying pulse widths. As mentioned in the RMT section of this document a signal with a 1500us pulse width is the neutral position. As the pulse width is varied within the range of 1050us to 1950us we can (because of the gearing done with the RMT) achieve 360° of motion. All 6 of the positions of the RMT will be stored in a Look up table with the appropriate pulse widths so finding the correct position is as easy as pulling the correct pulse width out of the table. This pulse width will be used to generate the needed 3.3V square wave which will be sent to the servo across pin 22.

The next step once the RMT is in place is to drive the needle into the medicine vial via moving the PLA upward. To do this 3.3v signals will be sent across VCCIO pins 10 and 26, toggling the PLA's relays and giving it a positive polarity. This will cause to PLA to start extending at which point pin 2(a general IO pin) will be monitored. Pin 2 is connected to the upper contact that will inform the microcontroller as to when the PLA is in place at which point the signal to the relays will be cut, stalling the PLA at its current position.

Now to compute the pull distance that is needed to fill the syringe to the user's desired amount. This will be done using the following equation:

$$H = \frac{V}{(\pi * r^2)} \qquad (4)$$

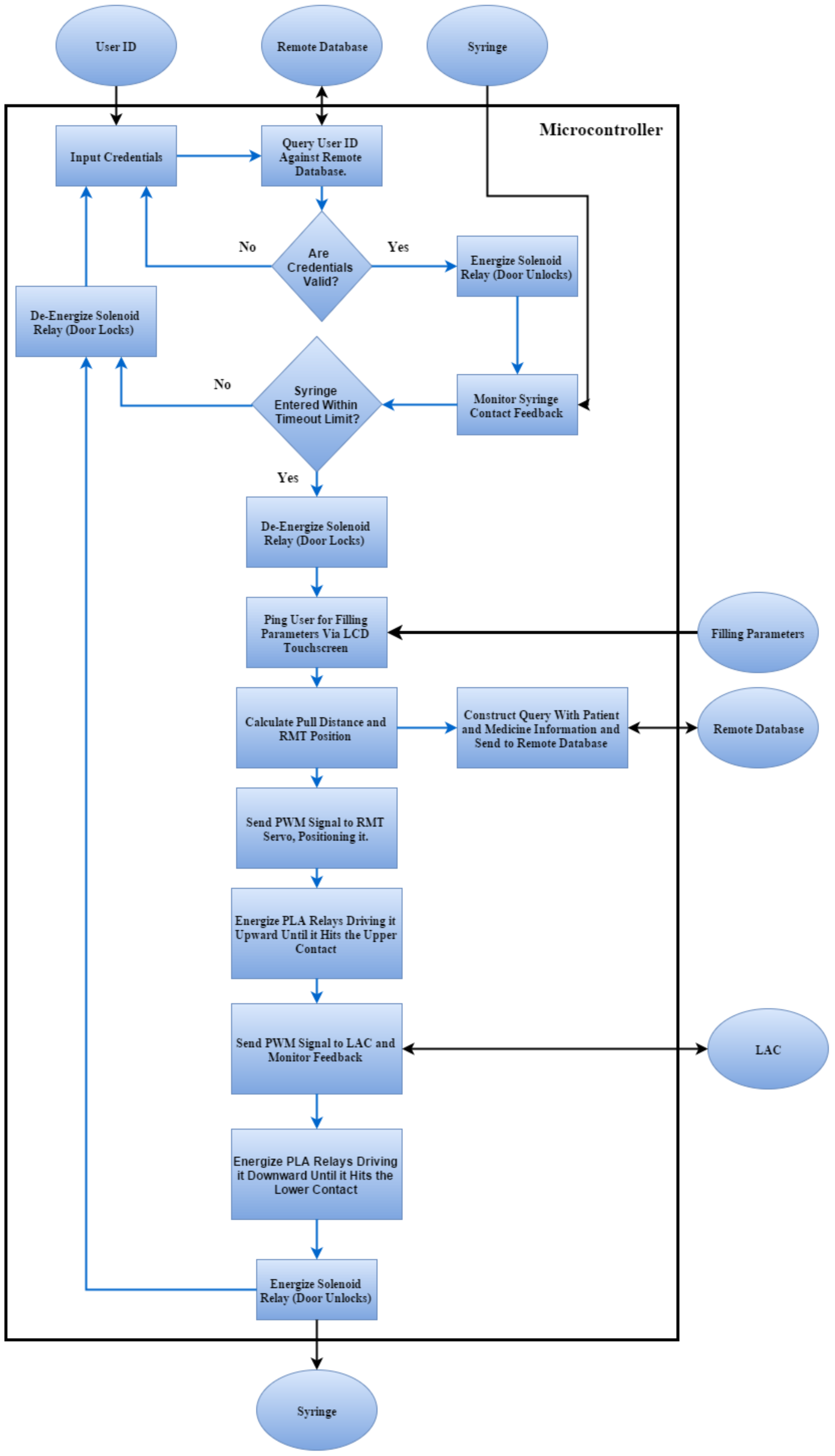
H = pull distance
V = Inputted volume
r = radius of the syringe

Pull distance (H) is the exact distance that the plunger of the syringe must travel in order for the desired volume to be achieved.

Because the system only allows one type of 3mL syringes r can be considered a constant 4.365mm. Then using the inputted volume this equation will yield the exact distance the syringe plunger needs to move in order to achieve this volume. Once H is found, it needs to be converted to a percentage of the total stroke length of the HPLA, which is 100mm. Dividing H by 100 will therefore yield the percentage of the total stroke length that the HPLA needs to move. This conversion is necessary because the LAC uses the duty cycle of a PWM signal as positional input for the HPLA. When this calculation is complete a PWM signal with the appropriate duty cycle is sent to the LAC across pin 14. The LAC then handles moving the HPLA to the correct distance, this process is discussed in detail in the next section. Once this signal has been sent to the LAC the microcontroller will begin monitoring the feedback from the actuator on pin 60. This will be analog feedback received so it will be run through microcontroller's analog to digital converter. The microcontroller is merely monitoring this feedback to know when the LAC has finished moving the HPLA so it will check the change of rate of the feedback. Once the feedback holds at a constant value for a designated period of time the it will mean the HPLA has finished moving and it is safe to move on to the next stage.

At this point the syringe is filled to the correct amount and needs to be outputted back to the user. The VCCIO pins 10 and 26 will be energized again to toggle the PLA's relays but this time giving the PLA the opposite polarity, driving it downward. One the PLA is in motion the microcontroller will monitor pin 5 (the general IO pin hooked up to the lower contact) in order to know when to stop moving the PLA. Once the PLA is in its resting position the door solenoid’s relay will be toggled unlocking the door and the syringe contact switch will be monitored again to know when the syringe has left the system. Once this happens the microcontroller will take a timestamp and send it, along with the other filling parameters, and send them in a post query to the remote database, therefore saving when the syringe left the system and the filling parameters for that syringe under the patient’s name.

At this point the door solenoid relay will be toggled one last time locking the door, the software will then post a pin-pad on the display and reenter it’s idle loop, waiting for a user ID. [EO]


User ID
Remote Database
Syringe
Microcontroller
Input Credentials
Query User ID Against Remote Database.
No
Are Credentials Valid?
Yes
Energize Solenoid Relay (Door Unlocks)
De-Energize Solenoid Relay (Door Locks)
No
Syringe Entered Within Timeout Limit?
Monitor Syringe Contact Feedback
Yes
De-Energize Solenoid Relay (Door Locks)
Ping User for Filling Parameters Via LCD Touchscreen
Filling Parameters
Calculate Pull Distance and RMT Position
Construct Query With Patient and Medicine Information and Send to Remote Database
Remote Database
Send PWM Signal to RMT Servo, Positioning it.
Energize PLA Relays Driving it Upward Until it Hits the Upper Contact
Send PWM Signal to LAC and Monitor Feedback
LAC
Energize PLA Relays Driving it Downward Until it Hits the Lower Contact
Energize Solenoid Relay (Door Unlocks)
Syringe
Figure 25: Level 2 Microcontroller Software Design

3.9.2) Linear Actuator Controller (LAC) Software

The Linear Actuator Controller (LAC) comes will a control program by default but also has a full API that allows for custom control programs to be written for the LAC. This API can be used with a few different languages including Visual C++ and Labview. For the purpose of this project a custom control program will be written as it allows for more direct control over the behavior of the actuator, for example the API supports setting a maximum pull distance. This will be set to the distance it takes to pull 3 mLs out of the type of the syringe as that is the largest marked volume on the type of syringe being used. This control software is extremely important to the system as a whole as this is where the engineering requirement of being able to fill a syringe with 0.1mL of accuracy.

The LAC has a precision coefficient from 0-1023 that can be set which changes the amount of potential error the actuator will have. The magnitude of error the LAC will have depending upon this coefficient can be found using the equation:

$$\frac{Value}{1024} * Stroke = |error| \qquad (5)$$

Value is the precision coefficient to be set and *Stroke* is the length of the actuator at full extension. The HPLA (High Precision Linear Actuator) being used for this project has a stroke length of 100mm. The LAC's default control software automatically uses a precision coefficient of 4 so using these two values the level of accuracy of the HPLA can be calculated.

$$\frac{4}{1024} * 100mm = \pm 0.390625mm$$

Now this range of accuracy in millimeters needs to be converted to milliliters which can be done by manipulating the equation used earlier to calculate the pull distance.

$$V = \pi * r^2 * h \qquad (6)$$

$$V = \pi * 4.365^2 * 0.390625$$

$$V = \pm 23.381mmm^3$$

$$V = \pm 0.023381mL$$

This shows that using the default precision settings for the LAC will be more than ample to meet the engineering requirement. The API does give the functionality to change the coefficient from 4 to anything else 0-1023 however we are likely to leave it as even though we could reduce it and get even more precise. The reason being that if you reduce the coefficient too much you run the risk of the actuator falling into an endless loop of pushing to just before and just after the desired position as the level of accuracy is smaller space between two actuator positions.

The operation mode we are choosing for the LAC uses the duty cycle percentage for a 3.3v 1kHZ square wave and relates it to the percentage of extension the actuator needs to be moved to 0% being fully retracted and 100% being fully extended. This means that the microcontroller can drive the LAC with a single EPMW pin.

The flow of the control software of the LAC can be seen below in Figure 26. Once the PWM signal is received by the LAC it will find the duty cycle of the signal. Next this percentage is decoded by the LAC into an actuator position. Once the desired position is found the LAC will compare this against the set Limit (a full 3mL syringe which means a 47.157mm extension). If it is greater than the limit then the desired position this thrown out and the actuator is moved to the limit position. This is done by the LAC sending a signal to the actuator to move and monitoring potentiometer feedback, constantly taking the difference from the feedback position and the desired position until the actuator is extended the proper amount. Once this is done the LAC will idle waiting for the next PWM signal to be sent from the microcontroller. [EO]

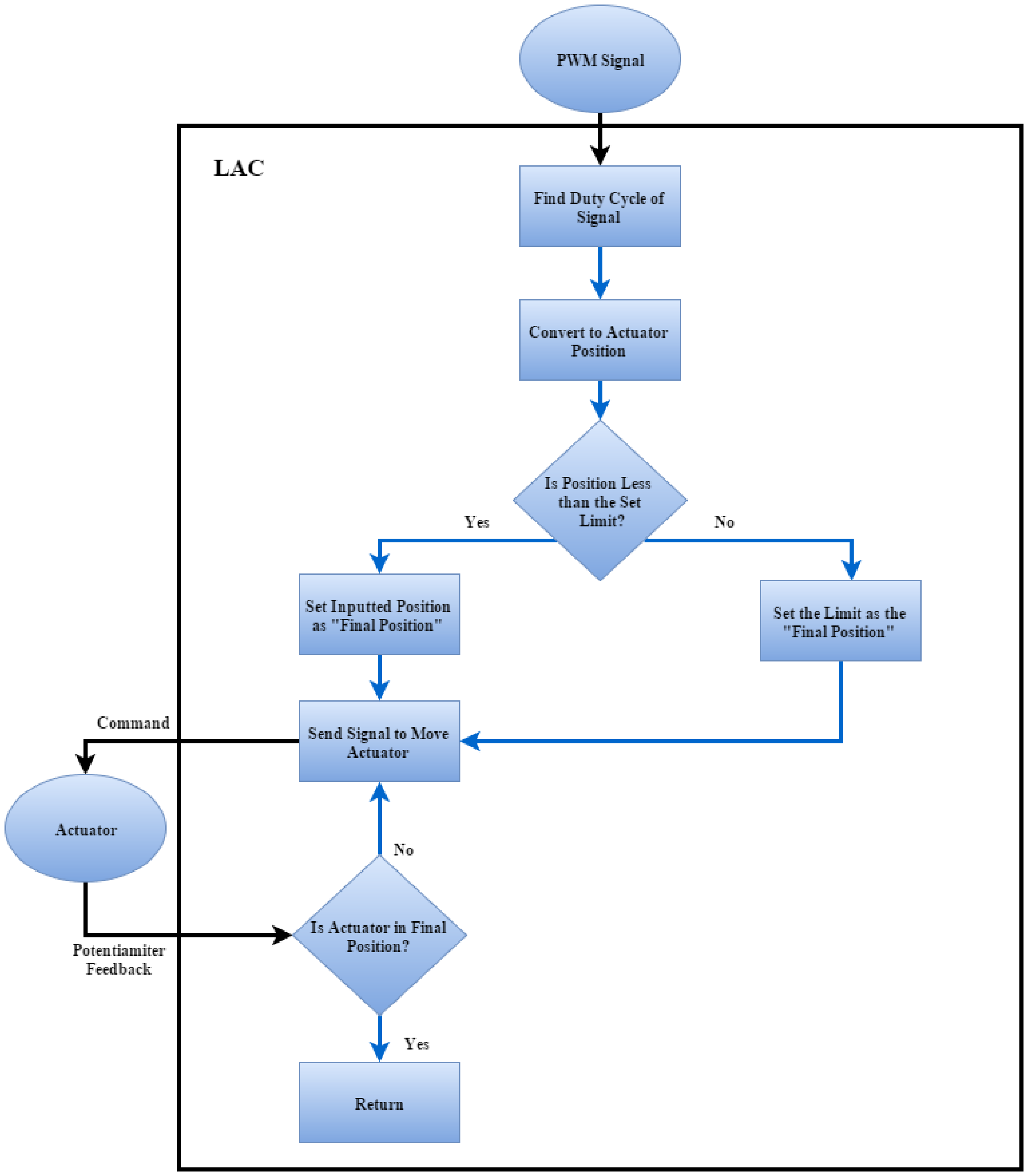

PWM Signal
LAC
Find Duty Cycle of Signal
Convert to Actuator Position
Is Position Less than the Set Limit?
Yes
No
Set Inputted Position as "Final Position"
Set the Limit as the "Final Position"
Command
Send Signal to Move Actuator
Actuator
No
Is Actuator in Final Position?
Potentiamiter Feedback
Yes
Return

Figure 26: LAC Software Flowchart

3.10) Display

The display that will be used is the Kentec 3.5" Resistive LCD Touchscreen that is offered as a boosterpack to the Hercules board. This LCD touch screen will display a keyboard and allow for the user to input the patient and medicine information. Once the information is entered it will then be sent to the database and used to operate the machine.

The touchscreen will interface with the board through the use of the board's SPI features. Since this specific touchscreen is sold as a boosterpack to the Hercules board, there will be minimal issues in implementation.

Though the touchscreen is small, it will allow for plenty of space to enter in the patient and medical information if one field is editable at a time. The user will be prompted to enter in one field at a time until all fields are entered. The bottom portion of the screen will display the keyboard and the top portion will display the information prompt and the information being entered. Since the screen is resistive, the user’s finger or a small stylus can be used to enter in the information. Once all fields are entered, the keyboard will drop away showing all the entered information. From there, the user will be prompted to either confirm or edit their input. If confirmed the information will then be used by the microcontroller to log to the database and continue on the syringe filling process. [DK]

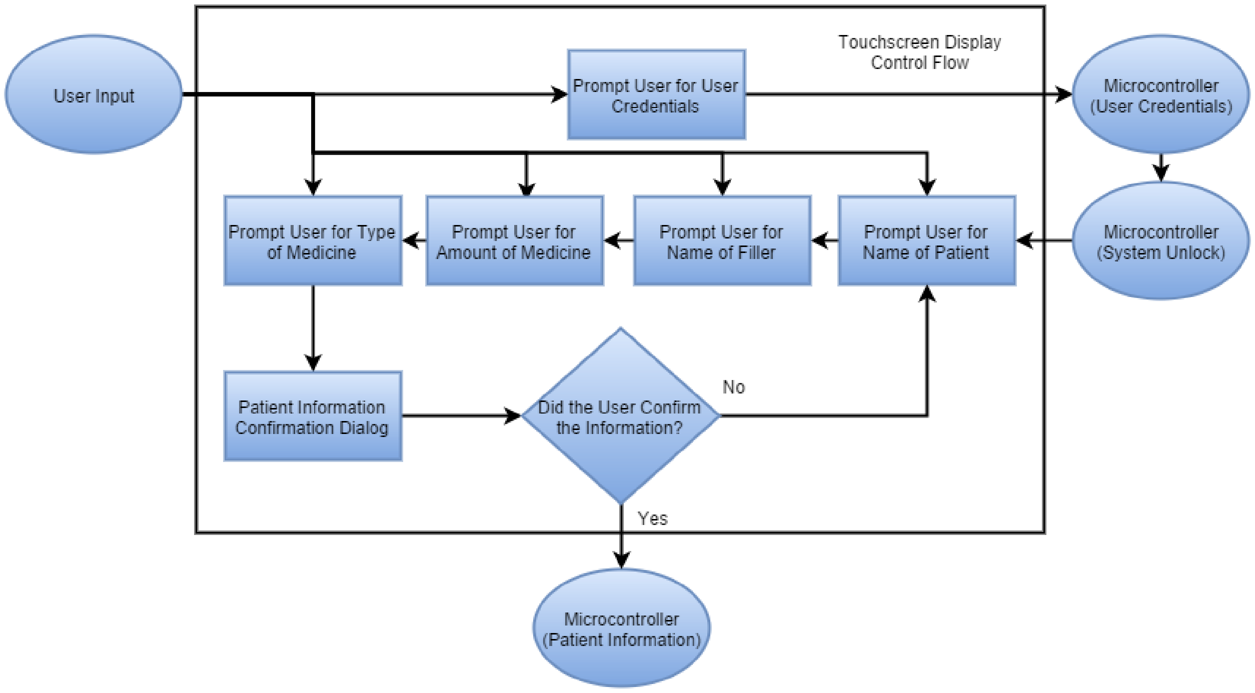

User Input
Touchscreen Display Control Flow
Prompt User for User Credentials
Microcontroller (User Credentials)
Microcontroller (System Unlock)
Prompt User for Type of Medicine
Prompt User for Amount of Medicine
Prompt User for Name of Filler
Prompt User for Name of Patient
Patient Information Confirmation Dialog
Did the User Confirm the Information?
No
Yes
Microcontroller (Patient Information)

Figure 27: Touch Screen Diagram

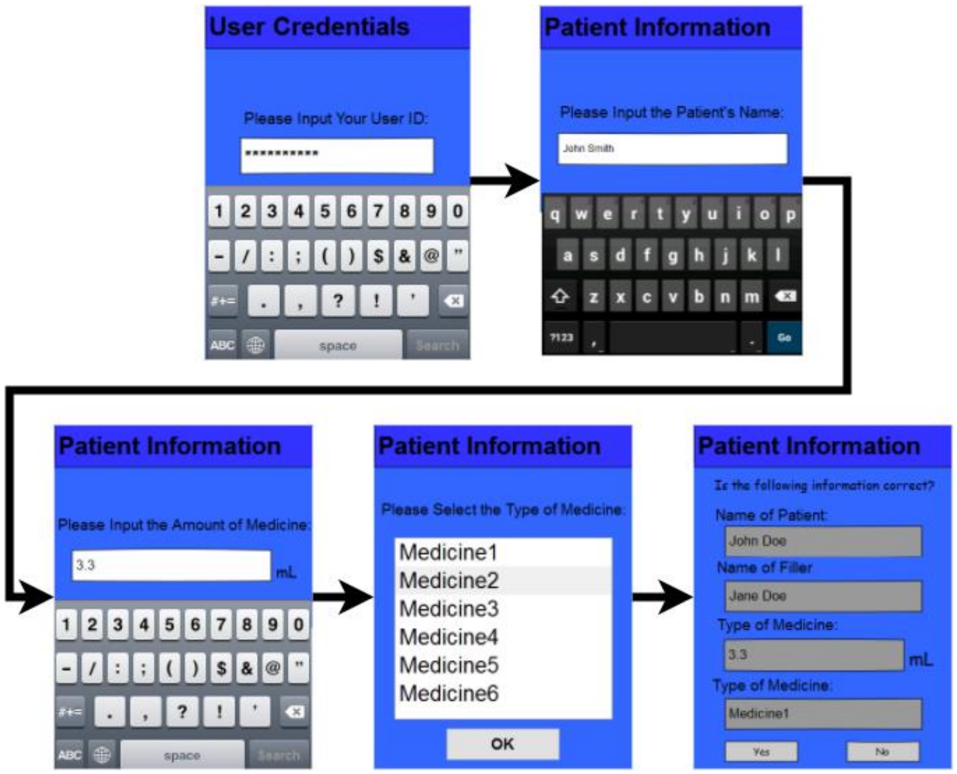


Figure 28: GUI Flow Diagram

3.11) System Diagram

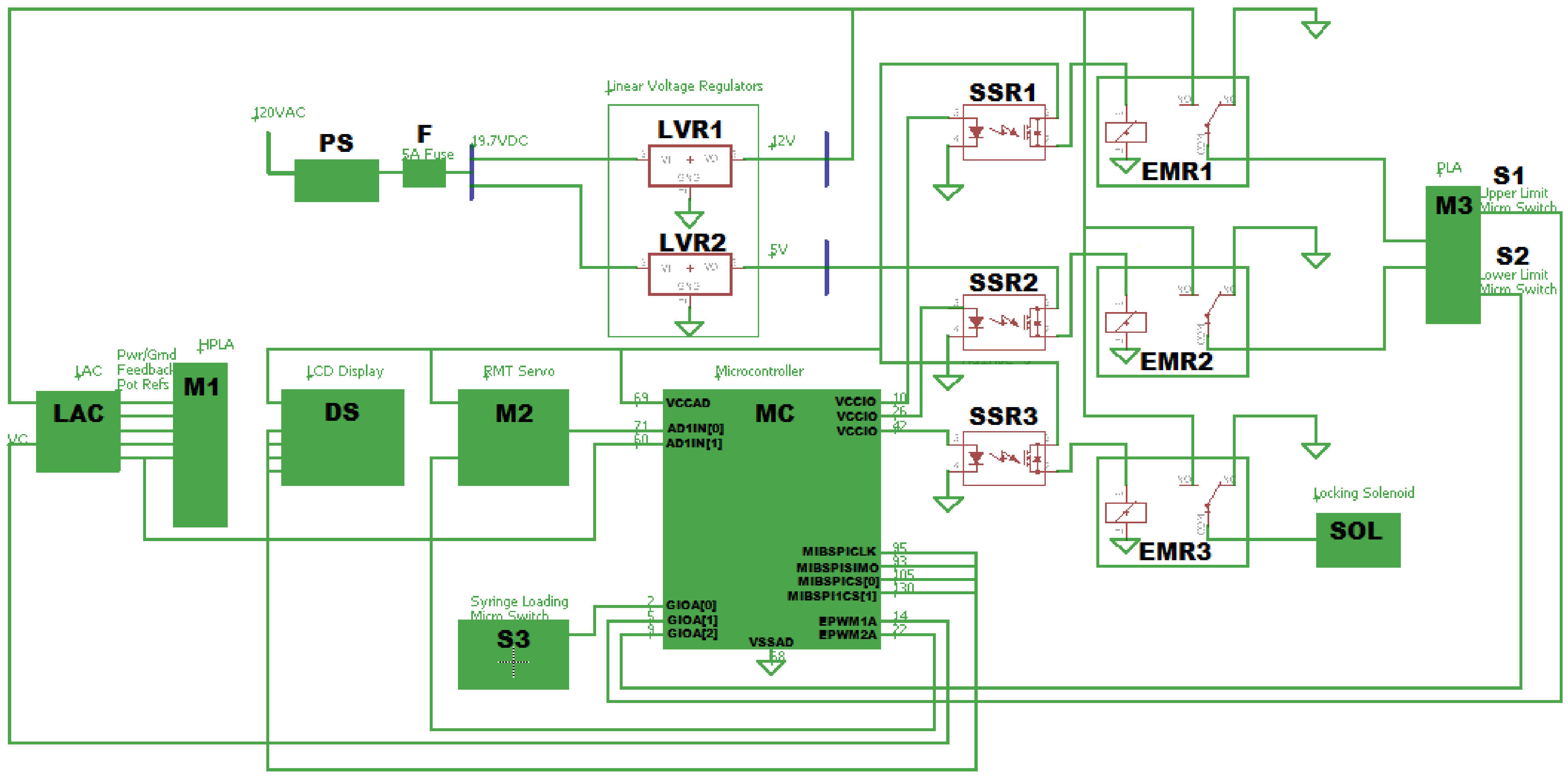


Figure 29: System Overview Schematic

4) Parts List

Part Type	Part Description	Ref	Qty	Price per unit	Total cost	Suggested Vendor	Vendor Part Number	Website	Mfg Part Number	Datasheet
	4in firgelli high precision actuator with potentiometer feedback	M1	1	$80	$80	Firgelli	L16-P	http://store.firgelli.com/category_s/1823.htm	L16-P	http://www.firgelli.com/pdf/LAC_Advanced_Configuration.pdf
	Linear actuator controller for the 4in firgelli actuator	LAC	1	$40	$40	Firgelli	LAC	http://store.firgelli.com/LAC_Board_p/lac.htm	LAC	http://www.firgelli.com/Uploads/LAC_Datasheet.pdf
Actuators	6in windynation linear actuator, no built in positional feedback	M3	1	$55	$55	Firgelli	LIN-ACT1BR-06	http://www.amazon.com/gp/product/B00P4SR2WW?keywords=6in%20linear%20actuator&qid=1444849546&ref_=sr_1_2&sr=8-2	LIN-ACT1-06	http://www.windynation.com/cm/Actuator%20Manual_R3.pdf
	HiTEC digital Servo 90deg rotation	M2	1	$25	$25	Servo City	HS-5495HB	https://www.servocity.com/html/hs-5495bh_servo.html	35495S	http://hitecrcd.com/products/servos/sport-servos/digital-sport-servos/hs-5495bh-hv-digital-karbonite-gear-sport-servo/product
	48P, 24T, Servo Gear		1	$2	$2	Servo City	SPBD48-24-24	https://www.servocity.com/html/48_pitch_plain_bore_gears.html		
Servo	48P, 96T, Servo Gear		1	$3	$3	Servo City	SPBD48-24-96	https://www.servocity.com/html/48_pitch_plain_bore_gears.html		
	SPST Solid State Relay to toggle EMR's	SSR1,SSR2,SSR3	3	$1	$3	DigiKey	TLP222AF-ND	http://www.digikey.com/product-detail/en/TLP222AF/TLP222AF-ND/871243	TLP222AF	http://www.semicon.toshiba.co.jp/info/docget.jsp?type=datasheet&lang=en&pid=TLP222A
	SPDT EM Relay for main actuator control	EMR1,EMR2	2	$6	$12	DigiKey	PB282-ND	http://www.digikey.com/product-detail/en/V23026A1001B201/PB282-ND/254499	V23026A1001B201	http://www.te.com/commerce/DocumentDelivery/DDEController?Action=srchrtrv&DocNm=108-98009&DocType=SS&DocLang=EN
Relays	SPDT EM Relay for Locking Solenoid	EMR3	1	$4	$4	DigiKey	PB1249-ND	http://www.digikey.com/product-detail/en/1462042-7/PB1249-ND/2126947	1462042-7	http://www.te.com/commerce/DocumentDelivery/DDEController?Action=srchrtrv&DocNm=108-98025&DocType=SS&DocLang=EN

Part Type	Part Description		Qty	Price per unit	Total cost	Suggested Vendor	Vendor Part Number	Website	Mfg Part Number	Datasheet
Solenoid	Pull Solenoid 12V, 2A for door lock	SOL	1	$14	$14	DigiKey	1144-1301-ND	http://www.digikey.com/product-detail/en/DSOL-0844-12/1144-1301-ND/5213992	DSOL-0844-12	http://www.deltaww.com/filecenter/Products/download/04/0409/DSOL-0844.pdf
Micro Switch	SPST Normally Open Micro Switch	S1,S2,S3	3	$2	$6	DigiKey	MS0850506F020P1C-ND	http://www.digikey.com/product-detail/en/MS0850506F020P1C/MS0850506F020P1C-ND/1628082	MS0850506F020P1C	https://www.e-switch.com/system/asset/product_line/data_sheet/124/MS.pdf
Voltage Regulators	12V Linear Voltage Regulator (Micrel)	LVR1	1	$4	$4	DigiKey	576-1118-ND	http://www.digikey.com/product-detail/en/MIC29300-12WT/576-1118-ND/771587	MIC29300-12WT	http://www.micrel.com/_PDF/mic29150.pdf
	5V Linear Voltage Regulator (TI)	LVR2	1	$2	$2	DigiKey	296-35391-1-ND	http://www.digikey.com/product-detail/en/LM1085ISX-5.0%2FNOPB/296-35391-1-ND/3739095	LM1085ISX-5.0/NOPB	http://www.ti.com/lit/ds/symlink/lm1085.pdf
Micro Controller	TI Launchpad Hercules	MC	1	$20	$20	Texas Instruments	LAUNCHXL2-RM46	http://www.ti.com/ww/en/launchpad/launchpads-hercules-launchxl2-rm46.html#tabs	LAUNCHXL2-RM46	http://www.ti.com/tool/LAUNCHXL2-RM46
Touchscreen Interface	3.5in LCD touchscreen Display	DS	1	$25	$25	Texas Instruments	BOOSTXL-K350QVG-S1	http://www.ti.com/tool/boostxl-k350qvg-s1	BOOSTXL-K350QVG-S1	http://www.ti.com/tool/boostxl-k350qvg-s1
Housing Material	Plywood, .5inx4ftx8ft		1	$19	$19	Home Depot	166081	http://www.homedepot.com/p/Unbranded-19-32-in-x-4-ft-x-8-ft-Rtd-Sheathing-Syp-166081/100004472		
	LEXAN , Sheet Poly Clear 0.236x24x24 In		1	$64	$64	Metal Fastners and Steel Trading Comp	GRA0112001058	http://www.metsorj.com/ProductDetails.asp?ProductCode=E3D812		
	LEXAN , Sheet Poly Clear 0.5x12x12 In		1	$22	$22	United States Plastic Comp	43042	http://www.usplastic.com/catalog/item.aspx?itemid=28080&catid=704		
Power Supply	INSTEN 1042783, laptop charging cord	PS	1	$19		Newegg	N82E16834980059	http://www.newegg.com/Product/Product.aspx?Item=N82E16834980059	1042783	
Fuse	Bel Fuse Inc, 5Amp	F	1	$1		DigiKey	507-1253-ND	http://www.digikey.com/product-detail/en/5ST%205-R/507-1253-ND/1009025	5ST 5-R	http://belfuse.com/pdfs/5ST.pdf

5) Design Team Information

- Chad Borntreger, Electrical Engineering, Hardware Lead
- David Lee Keister III, Computer Engineering, Software Lead
- Ethan Oglesbee, Computer Engineering, Team Lead
- Zachary Stevens, Electrical Engineering, Archivist

6) Conclusions and Recommendations

Engineering Requirements	Implementation that satisfies requirement
1) Fill a syringe with 0.1mL of precision	The HPLA has the desired level of accuracy to allow $0.023381mL$ of precision.
2) User Credentials and Patient Information Logged	Apache Derby Database will store the logged information once the syringe is filled and leaves the system.
3) System must complete requirements 1 and 2 in 30 seconds.	Considering the travel speed of the servo and actuators, along with the data processing time; the system will finish within 30 seconds.
4) Selection between 6 different medicines	The range of motion of the RMT allows for 6 vials to be accessed.
5) System must have a touch screen interface	A 3.5" Resistive LCD Touchscreen will act as the interface for the system.
6) System must be locked to unauthorized personnel	A pull solenoid will allow the system to unlock when the system is accessed by an authorized user.

Many areas of this design can be done several different ways. If the Automated Syringe Filler were to go into production, some of the parts may change in order to meet specific customizations of the vials and syringes being used. The syringe holding device would need to be bigger or smaller, and the RMT would need to be sized to fit a different vial size. All of the features chosen for this project were researched and calculated to be implemented specifically into this system.

7) References

[1] Gary J. Pond, Dennis Cotic, “Syringe Filler Apparatus”, U.S. Patent US 8671994 B2, March 18, 2014

[2] Douglas Cassel, “Automated syringe filling system for radiographic contrast agents and other injectable substances”, U.S. Patent US 5911252 A, June 15, 1999

[3] Automated syringe filler and loading apparatus, S. R. Strangis. 01)). *US Patent: 8807177* Available: http://ezproxy.uakron.edu:2048/login?url=http://search.proquest.com/docview/1559724651?accountid=14471, 1401.

[4] 'Sterile Filling', *Optima-packaging-group.de*, 2015. [Online]. Available: http://www.optima-packaging-group.de/opg/pharma/en/productfields/technology.php5?tID=1. [Accessed: 06- Dec-2015].

8) Appendices

See section 4) Parts List for datasheet links.

9) Finalization

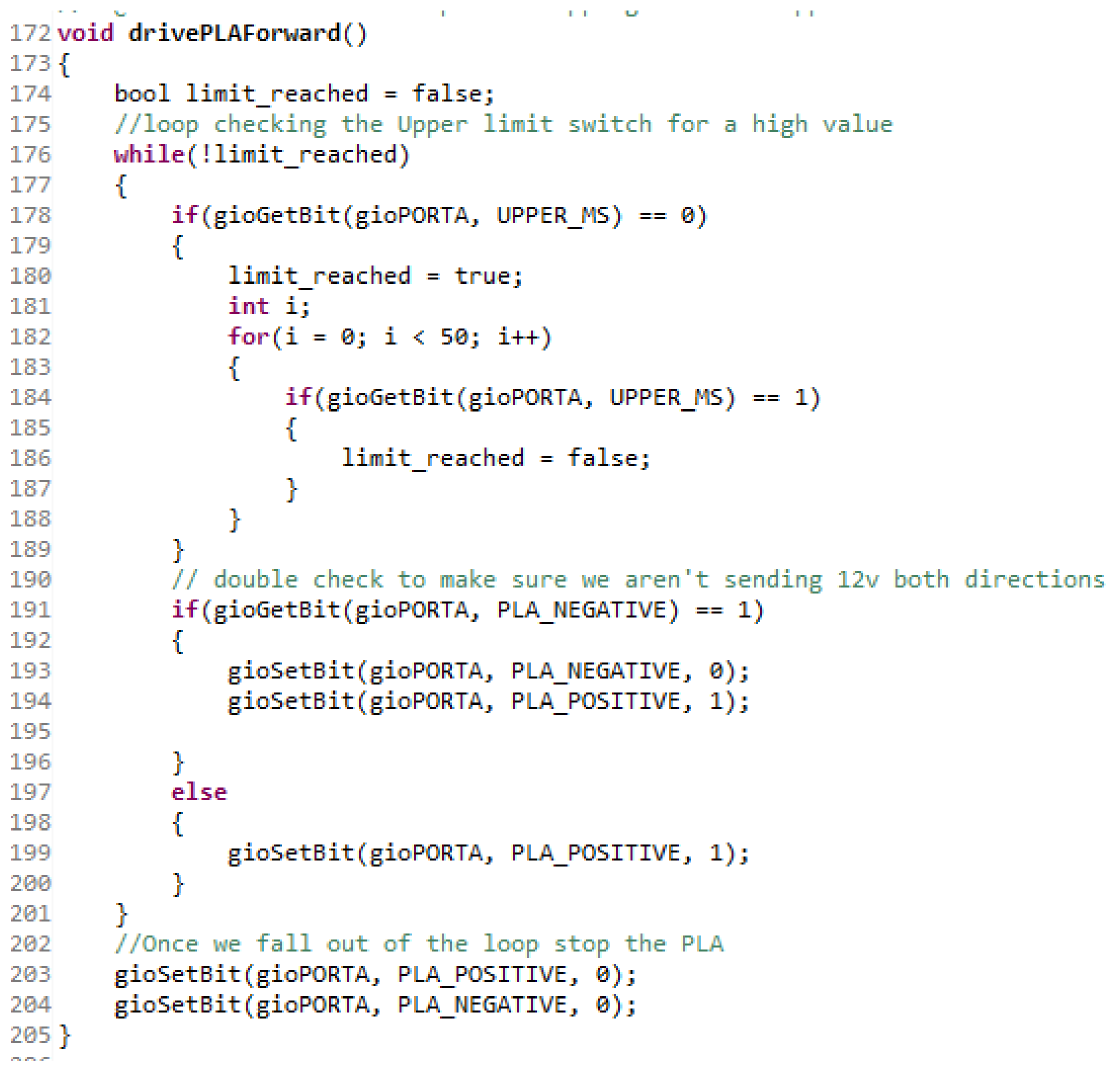
9.1) Microcontroller

For implementing the microcontroller control code a combination of Code Composer Studio IDE(CCS) and the Halcogen tool made by TI was used. Halcogen allows the user to select the specific drivers to be used (GIO, SPI, HET, etc) and select basic configuration settings such as which GIO Pins will be set to output and other such settings. Once these are set, Halcogen will generate all of the include files needed to drive these modules of the microcontroller. These files were then imported into CCS where they are used in conjunction with user written code to run the microcontroller. [EO]

9.1.1) PLA

For implementing the PLA control code GIO pins were used to send 3.3V out to the solid state relays instead of VCCIO pins. Four total GIO pins were used to drive the PLA, 2 set to input, two set to output. These pins were #defined at the beginning of the main.c file to make the code more readable and were named PLA_NEGATIVE, PLA_POSITIVE, UPPER_MS, and LOWER_MS. Two functions were written called drivePLAForward() and drivePLABackward() that handle the movement of the PLA. [EO]

These functions consist of a while loop that checks the input of the UPPER_MS pin or LOWER_MS pin. These pins are wired to either the upper or lower limit micro switch so as long as this input reads as “0” the PLA has not reached its movement limit. The interior of the loop is what drives the PLA upward or downward which is achieved by setting one of the two output pins, PLA_POSITVE or PLA_NEGATIVE, to “1” and the other to “0”. This will send the 3.3V out to the relay board, then powering the PLA with 12V in the appropriate direction. IF checks were placed in the loop to ensure both PLA_POSITIVE and PLA_NEGATIVE are never set to “1” at the same time as this would cause 12V to be sent to both terminals of the PLA. Once the input pin being monitored in the loop reads as “1” the function enters a redundancy check. The limit switches output is check fifty more times to ensure that the limit is, in fact, reached at which point the loop is broken out of and both PLA_POSITIVE and PLA_NEGATIVE are set to “0”, halting the PLA. This redundancy check was added because during testing of the micro switches it was found that if the output is only checked once there were a number of false positives being thrown due to noise. Once this check was put into place the switches operated perfectly. Below is a screen shot of the drivePLAForward() function. [EO]



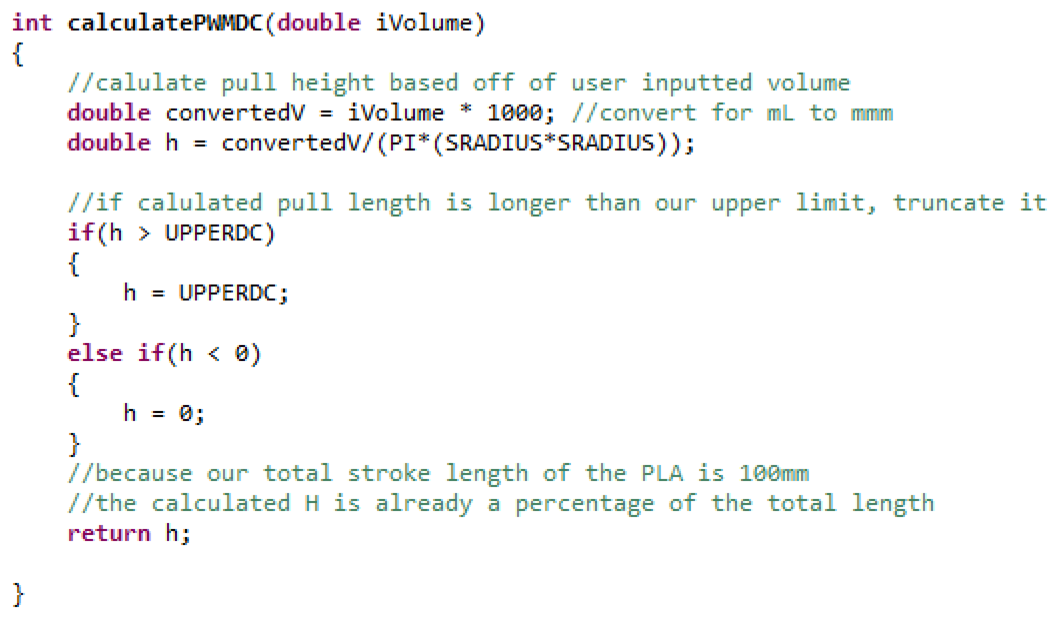
```
void drivePLAForward()
{
    bool limit_reached = false;
    //loop checking the Upper limit switch for a high value
    while(!limit_reached)
    {
        if(gioGetBit(gioPORTA, UPPER_MS) == 0)
        {
            limit_reached = true;
            int i;
            for(i = 0; i < 50; i++)
            {
                if(gioGetBit(gioPORTA, UPPER_MS) == 1)
                {
                    limit_reached = false;
                }
            }
        }
        // double check to make sure we aren't sending 12v both directions
        if(gioGetBit(gioPORTA, PLA_NEGATIVE) == 1)
        {
            gioSetBit(gioPORTA, PLA_NEGATIVE, 0);
            gioSetBit(gioPORTA, PLA_POSITIVE, 1);

        }
        else
        {
            gioSetBit(gioPORTA, PLA_POSITIVE, 1);
        }
    }
    //Once we fall out of the loop stop the PLA
    gioSetBit(gioPORTA, PLA_POSITIVE, 0);
    gioSetBit(gioPORTA, PLA_NEGATIVE, 0);
}
```

Figure 30: drivePLAForward() Function Code

9.1.2) HPLA

In order to drive the HPLA a 1kHz, 3.3V peak to peak, pulse width modulated signal needed to be sent to the LAC. A different module of the microcontroller was used to send this signal than originally planned. As stated above in section 3.9.1 the EPWM module was going to be used to generate our signal, however the High Efficiency Timer (HET) module was used instead. The reason for this switch was simply because it turned out to be easier from a programming perspective to generate the signal we wanted. The LAC then interprets the signal and moves the HPLA to the correct stroke length based of the duty cycle of the input signal.

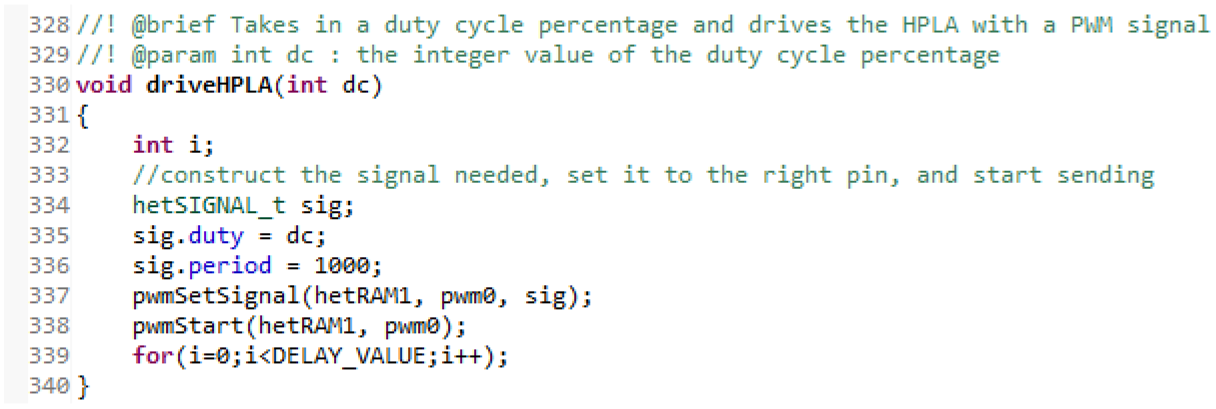
To calculate the proper duty cycle for the signal a function called calculatePWMDC(double iVolume) was written. This function, based off of the equations discussed in section 3.9.1 takes in the inputted volume from the user, converts it from mL to mmm and then computes the duty cycle need, the function is shown below.



```
int calculatePWMDC(double iVolume)
{
    //calulate pull height based off of user inputted volume
    double convertedV = iVolume * 1000; //convert for mL to mmm
    double h = convertedV/(PI*(SRADIUS*SRADIUS));

    //if calulated pull length is longer than our upper limit, truncate it
    if(h > UPPERDC)
    {
        h = UPPERDC;
    }
    else if(h < 0)
    {
        h = 0;
    }
    //because our total stroke length of the PLA is 100mm
    //the calculated H is already a percentage of the total length
    return h;

}
```

Figure 31: calculatePWMDC(doulbe iVolume) Code

An upper limit, UPPERDC, has been set to 50 will only allow duty cycles less than or equal to the maximum pull volume, 3mL, of the chosen syringe type. Once the percentage of the duty cycle has been calculated all that is left to do is send a PWM signal with a matching duty cycle to the LAC. To achieve this end a function driveHPLA(int dc), shown below, was written that takes in the integer value of the duty cycle, constructs the appropriate signal, and starts sending the signal to the LAC. This function does not stop sending the signal however, as the HPLA requires a constant input in order to hold its location which was an unforeseen issue. This was compensated for by merely continuous sending the signal until the micro switch that monitors the syringe location indicates that the syringe has left the system. [EO]



```
//! @brief Takes in a duty cycle percentage and drives the HPLA with a PWM signal
//! @param int dc : the integer value of the duty cycle percentage
void driveHPLA(int dc)
{
    int i;
    //construct the signal needed, set it to the right pin, and start sending
    hetSIGNAL_t sig;
    sig.duty = dc;
    sig.period = 1000;
    pwmSetSignal(hetRAM1, pwm0, sig);
    pwmStart(hetRAM1, pwm0);
    for(i=0;i<DELAY_VALUE;i++);
}
```

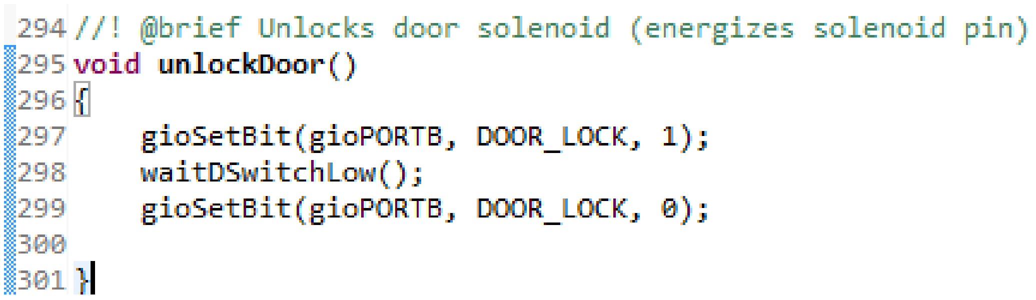
Figure 32: driveHPLA(int dc) Code

Once the LAC receives this signal it uses its internal control software to drive the HPLA to the desired length using the potentiometer feedback of the HPLA. Originally the microcontroller was going to monitor this feedback as well as a way to know when to stop sending the PWM signal. The thinking was that once the HPLA finds a resting position, the PWM signal would not be needed anymore. As mentioned above, this was an incorrect assumption and therefore the microcontroller no longer needs to monitor this feedback as the syringe monitoring micro switch is the flag to stop the PWM signal. The LAC has a precision constant that can be set, from 1 to 1024 and determines how accurate the final resting position of the HPLA will be. By default this is set to 4 which ensures approximately 99.63% accuracy.

When the LAC was first wired up and sent the correct PWM signal it was unable to find a resting position, constantly over and under shooting in an endless loop. At first the only way found to remedy this issue was to lower the precision setting of the LAC down to about 80% accuracy. While this did allow the HPLA to find a final resting position is was not accurate enough to meet the requirements set for this project. After more experimentation and speaking with the manufacturer of the HPLA and LAC we discovered that by putting a 10k ohm resistor in series with the PWM signal before it reached the HPLA cleaned up the signal enough that the LAC was able to find a final resting position for the HPLA on the default setting of 99.63% accuracy. [EO]

9.1.3) Door Solenoid

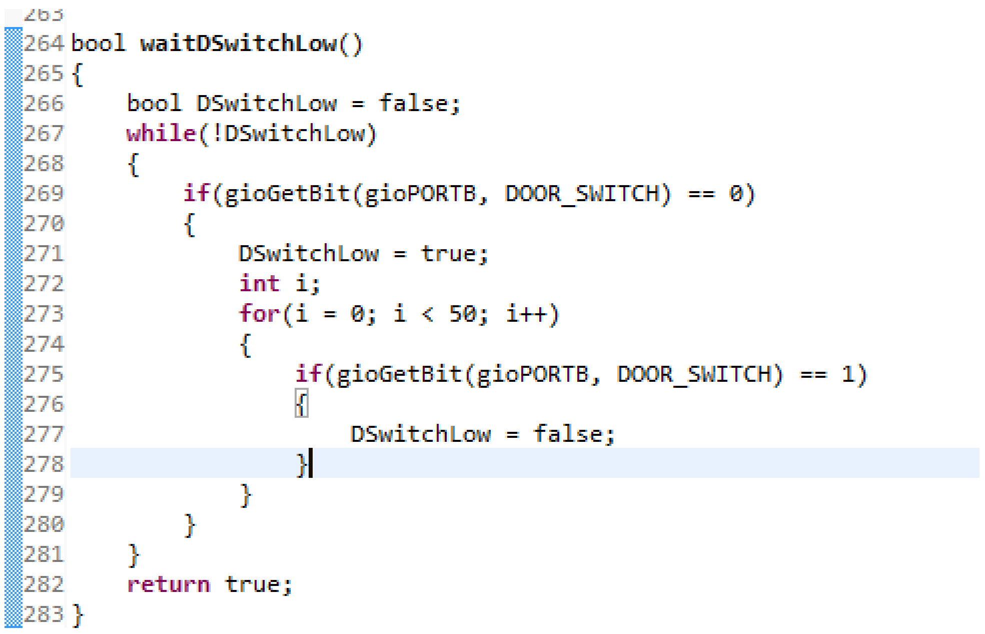
Controlling the door solenoid was a simple process. Instead of the originally planned VCCIO pin, a GIO pin was chosen, set to output, and #defined at the beginning of the main.c file as DOOR_LOCK. Two functions were written lockDoor() and unlockDoor(), shown below, that simply toggle this output pin from between “0” and “1”. Setting this pin to “1” will send 3.3V out from the DOOR_LOCK pin which till trigger our relay board to energize the door solenoid therefore unlocking the door.



```c
//! @brief Unlocks door solenoid (energizes solenoid pin)
void unlockDoor()
{
    gioSetBit(gioPORTB, DOOR_LOCK, 1);
    waitDSwitchLow();
    gioSetBit(gioPORTB, DOOR_LOCK, 0);

}
```

Figure 33: unlockDoor() Code

Two other functions were written to monitor the door micro switch called waitDSwitchHigh() and waitDSwitchLow(), shown below. These two functions contained a loop that would not return until the door switch was either open or closed. This allowed the microcontroller to know whether the door was physically open or closed. This was important because the door solenoid could not be kept energized for long periods of time due to temperature issues. With the help of these two functions the door solenoid was only energized for a few seconds at a time as the door was about to open or about to close. [EO]



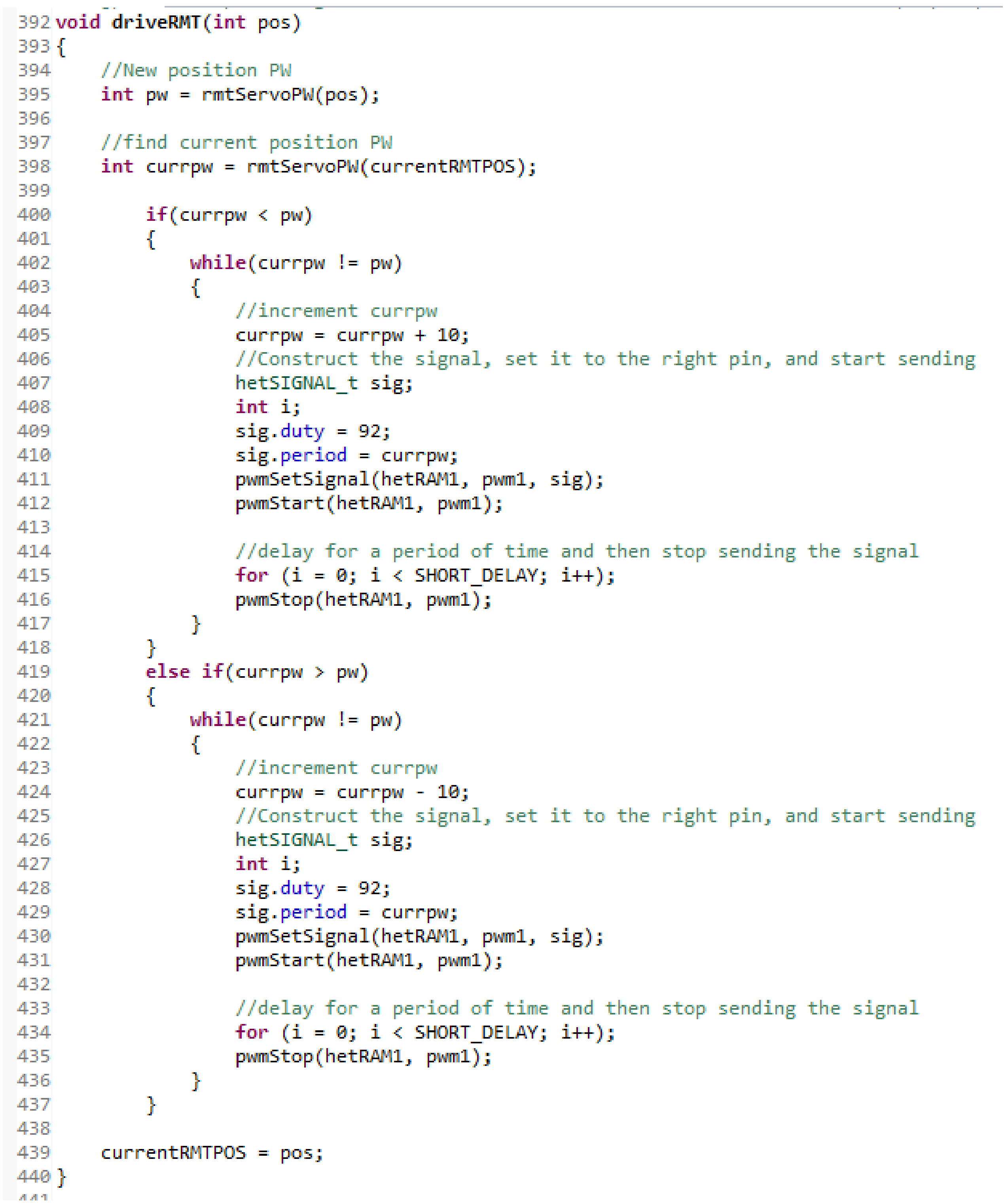
```c
bool waitDSwitchLow()
{
    bool DSwitchLow = false;
    while(!DSwitchLow)
    {
        if(gioGetBit(gioPORTB, DOOR_SWITCH) == 0)
        {
            DSwitchLow = true;
            int i;
            for(i = 0; i < 50; i++)
            {
                if(gioGetBit(gioPORTB, DOOR_SWITCH) == 1)
                {
                    DSwitchLow = false;
                }
            }
        }
    }
    return true;
}
```

Figure 34: waitDSwitchHigh() Code

9.1.4) RMT

The servo that drives the RMT required a 3.3V peak to peak PWM signal to operate. The pulse width of this signal determined which position the servo would move to with 1500us being the neutral position, as discussed in section 3.3 of this report. A 4-1 gearing was used to allow for 6 positions across 360 degrees for the RMT and because of this the different pulse widths for each position were able to be calculated exactly. These positions will not change within the system so they can be considered constant values and were therefore #defined at the beginning of main.c as RMT0 – RMT5 for all six positions. A function was written called rmtServoPW(int pos) that takes in the integer position the RMT needs to be moved to and returns the appropriate pulse width. This is achieved by a simple switch statement that equates the integer position to the correct #defined constant.

This conversion from positon to pulse width occurs within the function driveRMT(int pos) which drives the RMT to the correct position. A global integer variable was defined in the system named currentRMTPOS that tracks the current position of the RMT. After every fill cycle the RMT is reset to its 0 position which allows the system to assume that at startup the current position is 0. The driveRMT(int pos) function moves the RMT to the desired position in 10us intervals by slowly incrementing the currpw variable that represents the current pulse width. Once currpw makes the desired pulse width, currentRMTPOS is updated with the new position and the function returns. The pwm signals are constructed the High Efficiency Timer module of the microcontroller and sends this signal. The length of the signal pulse is determined by a short manual delay placed into the function. The servo is moved in such small steps because during testing the servo would skip gears if moved directly to the desired position in a signal motion. To allay this, the stepping method was implemented which was slower but overall made the system more robust. [EO]



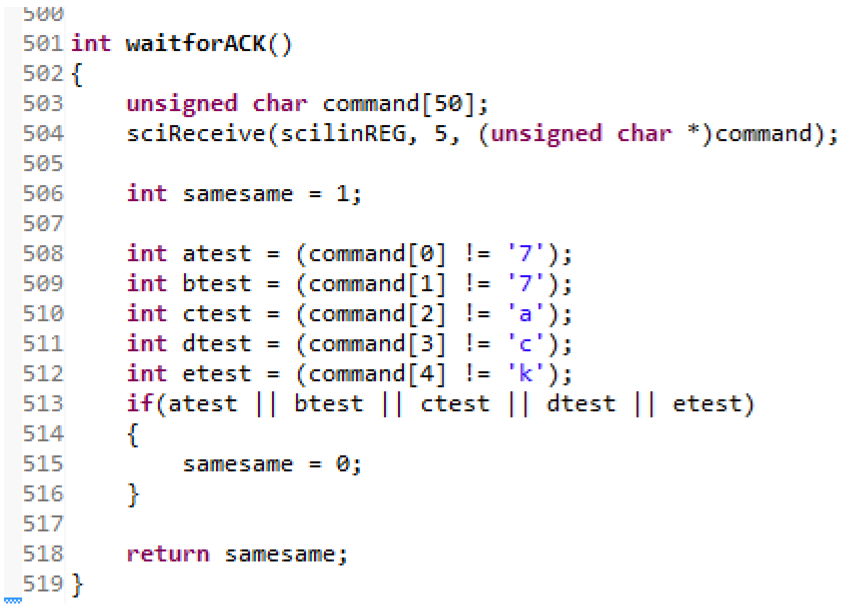
```
void driveRMT(int pos)
{
    //New position PW
    int pw = rmtServoPW(pos);

    //find current position PW
    int currpw = rmtServoPW(currentRMTPOS);

        if(currpw < pw)
        {
            while(currpw != pw)
            {
                //increment currpw
                currpw = currpw + 10;
                //Construct the signal, set it to the right pin, and start sending
                hetSIGNAL_t sig;
                int i;
                sig.duty = 92;
                sig.period = currpw;
                pwmSetSignal(hetRAM1, pwm1, sig);
                pwmStart(hetRAM1, pwm1);

                //delay for a period of time and then stop sending the signal
                for (i = 0; i < SHORT_DELAY; i++);
                pwmStop(hetRAM1, pwm1);
            }
        }
        else if(currpw > pw)
        {
            while(currpw != pw)
            {
                //increment currpw
                currpw = currpw - 10;
                //Construct the signal, set it to the right pin, and start sending
                hetSIGNAL_t sig;
                int i;
                sig.duty = 92;
                sig.period = currpw;
                pwmSetSignal(hetRAM1, pwm1, sig);
                pwmStart(hetRAM1, pwm1);

                //delay for a period of time and then stop sending the signal
                for (i = 0; i < SHORT_DELAY; i++);
                pwmStop(hetRAM1, pwm1);
            }
        }

    currentRMTPOS = pos;
}
```

Figure 35: driveRMT(int pos) Code

9.1.5) User Input

User input was originally going to be given to the microcontroller via a SPI controlled touchscreen. During implementation compromises had to be made and the touchscreen was shelfed and replaced with a java emulated GUI's that were controlled by the same java function that controls the remote database. The reasoning behind this switch and the details of the java application are discussed in section 9.11. The microcontroller talked with this application via an SCI connection that was achieved across a USB cable. A basic packet command send and receive system was created so the two sides could effective communicate with one another which is detailed in section 9.11.

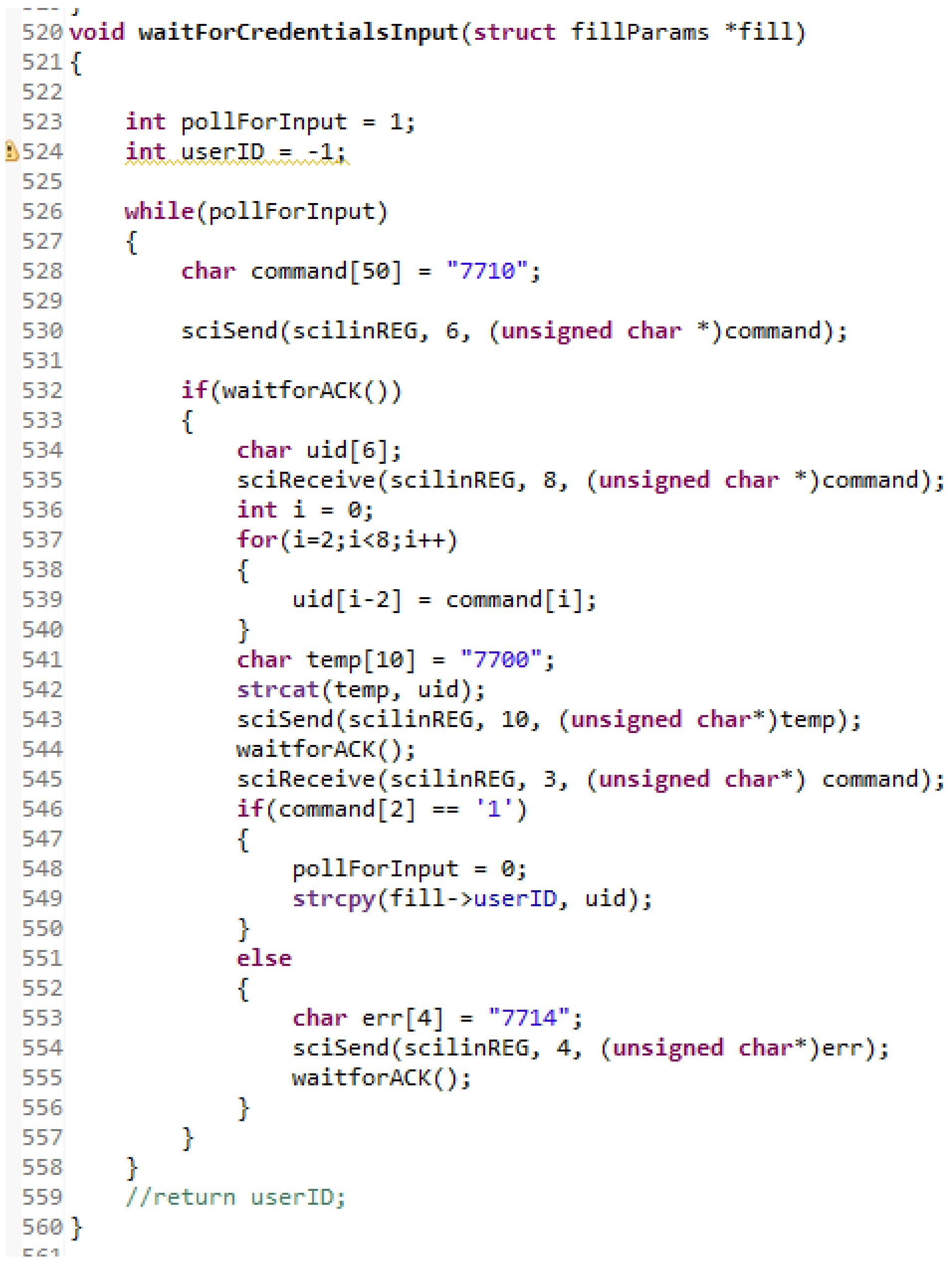
In order to ensure that commands sent form the microcontroller to the java application were received correctly an ACK or acknowledgment system had to be created. Because of the nature of the SCI communication used a simple character array "ack" was sent as the acknowledgment by the java application. The microcontroller needed to wait for an ACK after every command sent to the java application before continuing on so a function int waitforACK() was written that would loop while waiting for the ACK to be received. Once a messaged is received via the SCI connection it is validated to make sure it is the ACK message and returns a 1 if it is and a 0 otherwise.



```
int waitforACK()
{
    unsigned char command[50];
    sciReceive(scilinREG, 5, (unsigned char *)command);

    int samesame = 1;

    int atest = (command[0] != '7');
    int btest = (command[1] != '7');
    int ctest = (command[2] != 'a');
    int dtest = (command[3] != 'c');
    int etest = (command[4] != 'k');
    if(atest || btest || ctest || dtest || etest)
    {
        samesame = 0;
    }

    return samesame;
}
```

Figure 36: waitForACK() Code

Once the system could send commands and receive ACKs a function was written to ping the user for user credentials called waitForUserCredentials(struct fillParams *fill). This function, seen below in Figure 37 contains a while loop that endless polls the java application waiting for a valid userID. The loop constructs a command for the java application asking it to ping the user for a userID. Once this command was constructed and sent the function waits for an ACK to be returned from the java app, once this happened the function then receives a packet contained the userID. Once the userID is obtained, a second command is constructed, with the userID included, and sent to the remote database, asking it to check the validity of the ID. Once a response from the database is received, ‘1’ being valid, ‘0’ being invalid, an IF check occurs. If the IF check is passed the function stored the userID and returns out, as it was valid. If the IF check fails, an error code is sent to the java application and the function goes back to the top of the loop, restarting the process. [EO]



```
void waitForCredentialsInput(struct fillParams *fill)
{

    int pollForInput = 1;
    int userID = -1;

    while(pollForInput)
    {
        char command[50] = "7710";

        sciSend(scilinREG, 6, (unsigned char *)command);

        if(waitforACK())
        {
            char uid[6];
            sciReceive(scilinREG, 8, (unsigned char *)command);
            int i = 0;
            for(i=2;i<8;i++)
            {
                uid[i-2] = command[i];
            }
            char temp[10] = "7700";
            strcat(temp, uid);
            sciSend(scilinREG, 10, (unsigned char*)temp);
            waitforACK();
            sciReceive(scilinREG, 3, (unsigned char*) command);
            if(command[2] == '1')
            {
                pollForInput = 0;
                strcpy(fill->userID, uid);
            }
            else
            {
                char err[4] = "7714";
                sciSend(scilinREG, 4, (unsigned char*)err);
                waitforACK();
            }
        }
    }
    //return userID;
}
```

Figure 37: waitForCredentialsInput() Code

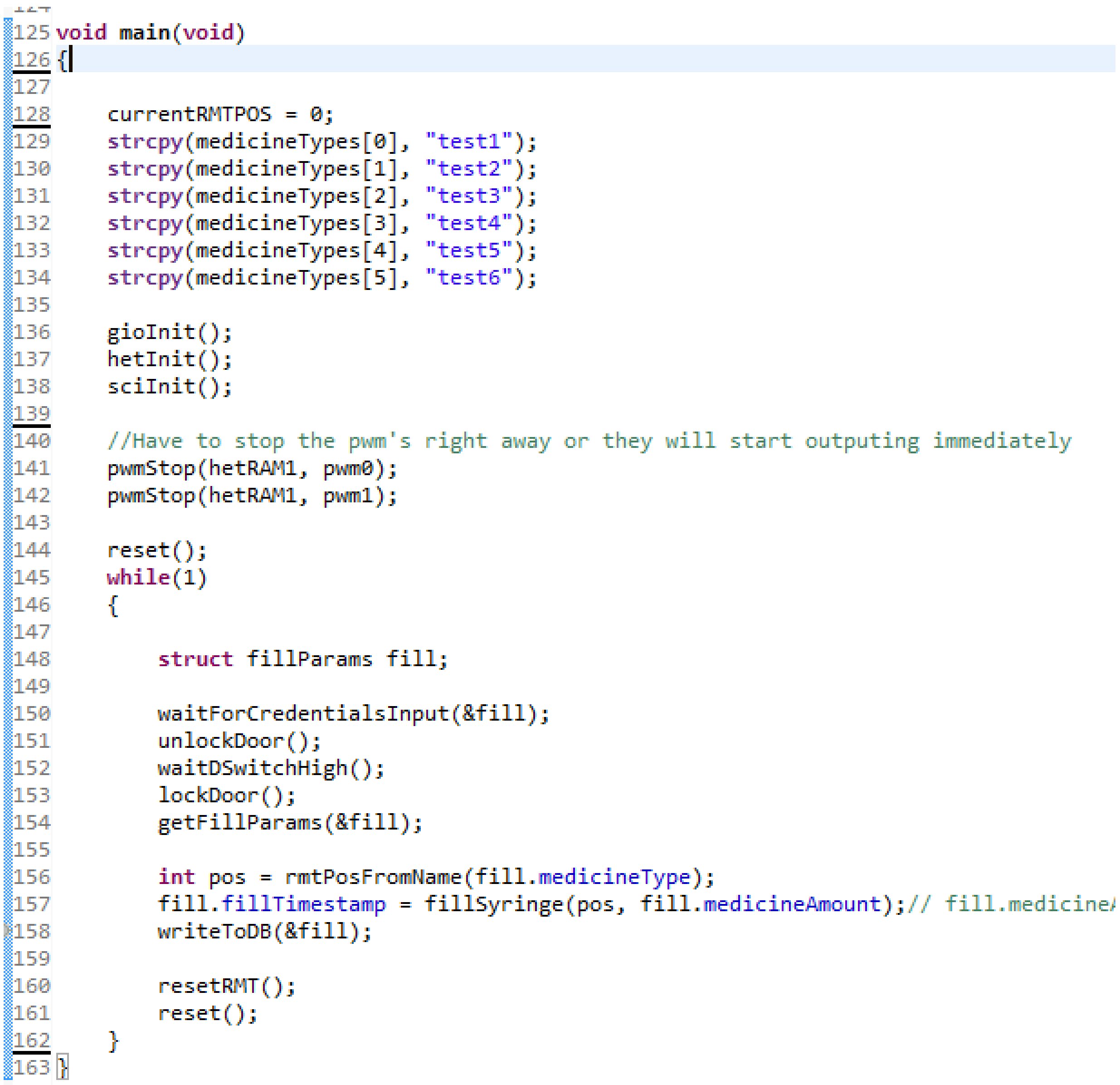
The rest of the needed user input is obtained from a single function called getFillParams(struct fillParams *fill). This function uses three similar while loops to get the following information from the user; patientID, medicine type, and medicine amount. Only one of the loops will be explained here as they are extremely similar in operation but all three can be seen in Figure 38. Much like in waitForCredentialsInput() the function starts by constructing a command for the java app to ping the user for a patientID and then sends the command via the SCI connections. Once sent the function waits for an ACK and upon receiving one, receives a second packet containing the patient ID. Again, just like in waitForCredentialsInput() the system quires the database for validity of the patientID and if valid stores it in the fill structure. If the ID had been invalid the loop would start again. This same paradigm is followed to get the medicine type, and medicine amount. For the sake of space on the microcontroller most input validation for these two fields was handled on the java application side. [EO]

```
void getFillParams(struct fillParams *fill)
{
        //Loop for getting the patient ID
        int patientID;
        int pollForInput = 1;
        while(pollForInput)
        {
                char command[50] = "7711";

                sciSend(scilinREG, 4, (unsigned char *)command);

                if(waitforACK())
                {
                        char pid[6];
                        sciReceive(scilinREG, 8, (unsigned char *)command);
                        int i = 0;
                        for(i=2;i<8;i++)
                        {
                            pid[i-2] = command[i];
                        }

                        char temp[10] = "7701";
                        strcat(temp, pid);
                        sciSend(scilinREG, 10, (unsigned char*)temp);
                        waitforACK();
                        sciReceive(scilinREG, 3, (unsigned char*) command);
                        if(command[2] == '1')
                        {
                                pollForInput = 0;
                                strcpy(fill->patientID, pid);
                        }
                        else
                        {
                                char err[4] = "7714";
                                sciSend(scilinREG, 4, (unsigned char*)err);
                                waitforACK();
                        }
                }
        }

        //Loop for getting the medicine type
        pollForInput = 1;
        while(pollForInput)
        {
                char command[50] = "7712";

                sciSend(scilinREG, 4, (unsigned char *)command);

                if(waitforACK())
                {
                        char medType[5];
                        sciReceive(scilinREG, 7, (unsigned char*)command);
                        int i;
                        for( i = 2; i<7; i++)
                        {
                                fill->medicineType[i-2]/*medType[i-2]*/ = command[i];
                        }
                        pollForInput = 0;
                }
else
                {
                        char err[4] = "7714";
                        sciSend(scilinREG, 4, (unsigned char*)err);
                        waitforACK();
                }
        }

        //Loop for getting the medicine amount
        pollForInput = 1;
        while(pollForInput)
        {
                char command[50] = "7713";

                sciSend(scilinREG, 4, (unsigned char *)command);

                if(waitforACK())
                {
                        char medAmount[4];
                        sciReceive(scilinREG, 6, (unsigned char *)command);
                        int i;
                        for(i = 2; i < 6; i++)
                        {
                                medAmount[i-2] = command[i];
                        }
                        double medicineA = strtod(medAmount, NULL);

                        if(medicineA < 3)
                        {
                                fill->medicineAmount = medicineA;
                        }
                        else
                        {
                                fill->medicineAmount = 3;
                        }
                        pollForInput = 0;

                }
        }
                                        }
```

Figure 38: getFillParams() Code

9.1.6) Final Control Loop

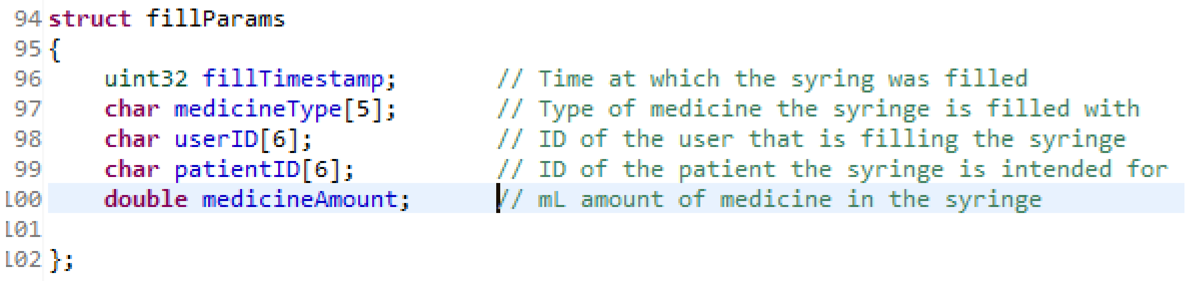
All of the pieces of code discussed before in this section were used in conjunction with each other to create a final control loop that was placed in the void main(void) function within the main.c file. This is the function that runs when the system is first turns on and will run in an endless loop until the system is powered down, shown below.



```c
void main(void)
{

    currentRMTPOS = 0;
    strcpy(medicineTypes[0], "test1");
    strcpy(medicineTypes[1], "test2");
    strcpy(medicineTypes[2], "test3");
    strcpy(medicineTypes[3], "test4");
    strcpy(medicineTypes[4], "test5");
    strcpy(medicineTypes[5], "test6");

    gioInit();
    hetInit();
    sciInit();

    //Have to stop the pwm's right away or they will start outputing immediately
    pwmStop(hetRAM1, pwm0);
    pwmStop(hetRAM1, pwm1);

    reset();
    while(1)
    {

        struct fillParams fill;

        waitForCredentialsInput(&fill);
        unlockDoor();
        waitDSwitchHigh();
        lockDoor();
        getFillParams(&fill);

        int pos = rmtPosFromName(fill.medicineType);
        fill.fillTimestamp = fillSyringe(pos, fill.medicineAmount);// fill.medicine
        writeToDB(&fill);

        resetRMT();
        reset();
    }
}
```

Figure 39: main() Code

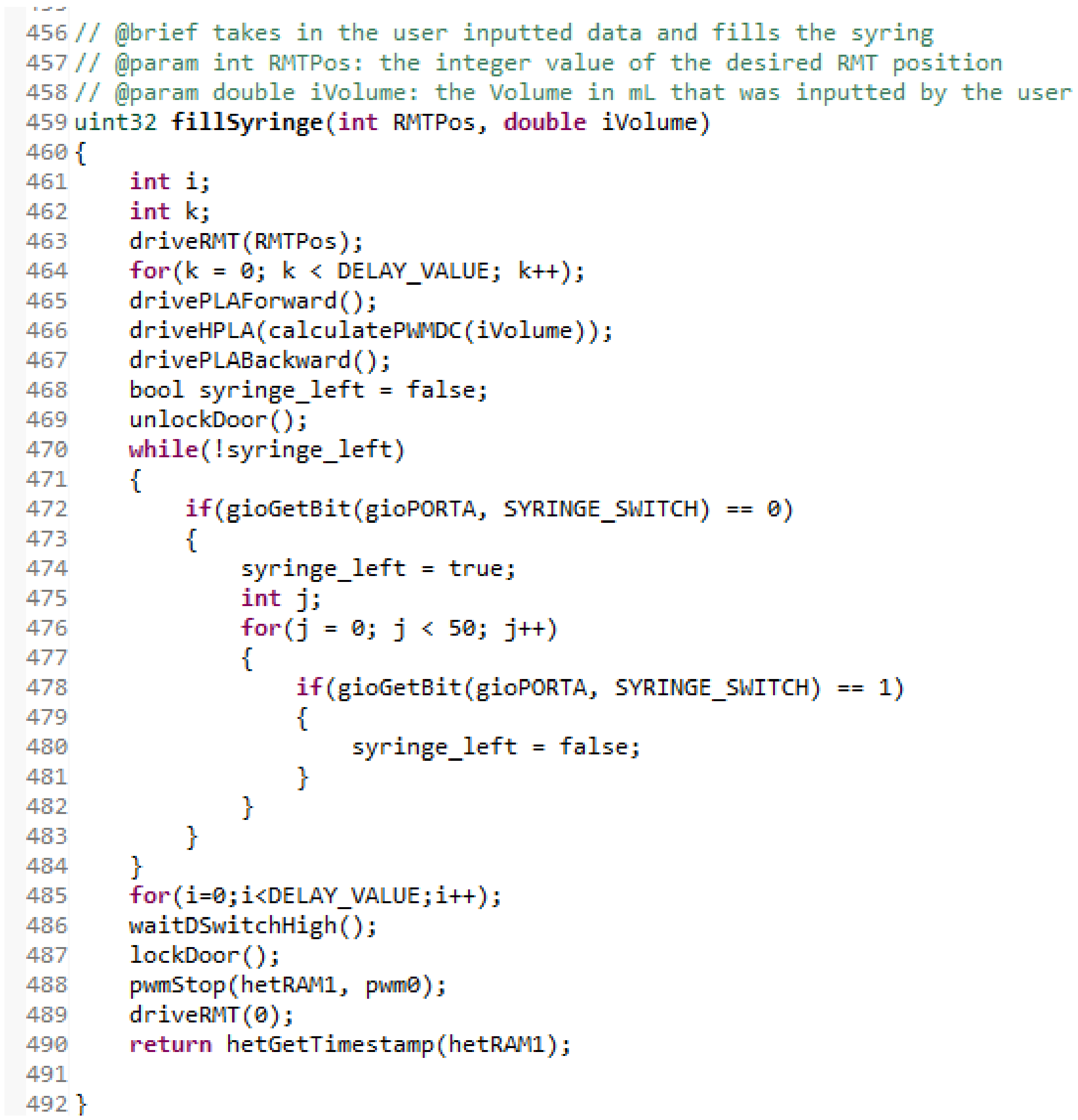
The first section of the main() is the initialization process for the system and it occurs before the while(1) loop. First all constants used by the system are set which includes the integer currentRMTPOS, discussed in section 9.1.4, is set to 0 since the RMT starts in its resting position. Next the array of medicine types is populated. Originally these names would be populated via user input when loading in the medicine vials but because of the timeline of this project they are simply declared at this location in the code. These characters arrays are checked against user input in the filling process to determine which positon the RMT needs to be moved to. Then the different modules of the microcontroller used need to be initialized which are general input/output (GIO), Serial Communication Interface (SCI) and the High Efficiency Timer (HET). The last bit of initialization is stopping the pwm signals used to drive the HPLA and the RMT. Because of the nature of the HET initialization function these pwm signals would immediately start outputting if not stopped at this time. [EO]



```
struct fillParams
{
    uint32 fillTimestamp;       // Time at which the syring was filled
    char medicineType[5];       // Type of medicine the syringe is filled with
    char userID[6];             // ID of the user that is filling the syringe
    char patientID[6];          // ID of the patient the syringe is intended for
    double medicineAmount;      // mL amount of medicine in the syringe

};
```

Figure 40: fillParams Stucture

Once the system is initialized it moves into the main control loop which is an endless loop and starts at the while(1) command in Figure 39. A structure named “fill” is declared that will hold all of the parameters needed for the system to fill the syringe, a breakdown of this structure can be seen in Figure 40 above. This struct is then passed into the waitForCredentialsInput() function. Once that function has received a verified userID it will return and the system will unlock the door, allowing the user to input a syringe into the system. Once the door is shut the system locks the door and the user is then pinged to enter in the reset of the filling parameters in the following order; patientID, medicine type, and medicine amount. These values are all stored in the same fill structure as before. Then the system uses the rmtPosFromName() function to convert from the character array name of the medicine type to one of the six positions of the rmt (discussed in section 9.1.4). This position integer as well as the medicine amount is passed to a function called fillSyring(int RMTpos, double iVolume) that contains the control code to fill the syringe, shown below in Figure 41. [EO]

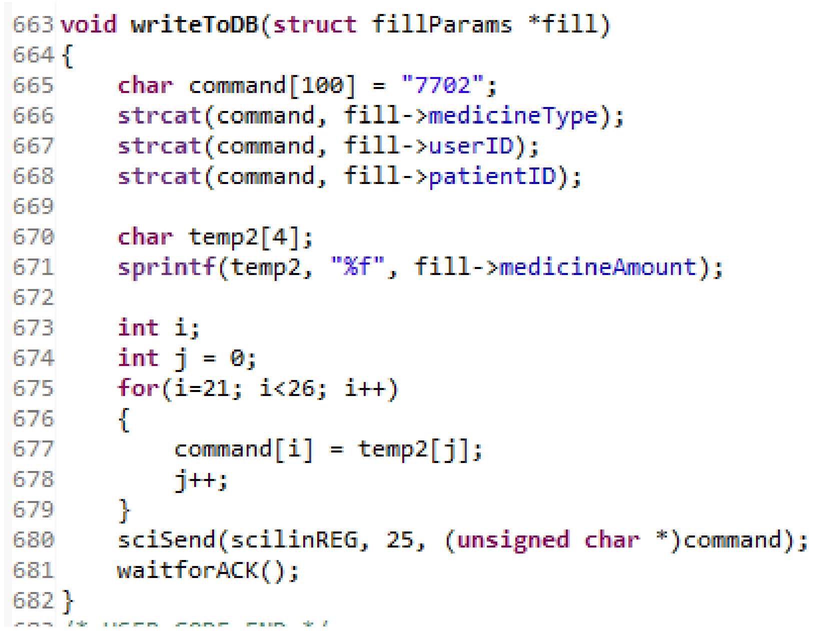


```
// @brief takes in the user inputted data and fills the syring
// @param int RMTPos: the integer value of the desired RMT position
// @param double iVolume: the Volume in mL that was inputted by the user
uint32 fillSyringe(int RMTPos, double iVolume)
{
    int i;
    int k;
    driveRMT(RMTPos);
    for(k = 0; k < DELAY_VALUE; k++);
    drivePLAForward();
    driveHPLA(calculatePWMDC(iVolume));
    drivePLABackward();
    bool syringe_left = false;
    unlockDoor();
    while(!syringe_left)
    {
        if(gioGetBit(gioPORTA, SYRINGE_SWITCH) == 0)
        {
            syringe_left = true;
            int j;
            for(j = 0; j < 50; j++)
            {
                if(gioGetBit(gioPORTA, SYRINGE_SWITCH) == 1)
                {
                    syringe_left = false;
                }
            }
        }
    }
    for(i=0;i<DELAY_VALUE;i++);
    waitDSwitchHigh();
    lockDoor();
    pwmStop(hetRAM1, pwm0);
    driveRMT(0);
    return hetGetTimestamp(hetRAM1);

}
```

Figure 41: fillSyringe Code

The fillSyringe function first takes the rmt position integer and uses it to drive the RMT in place via the driveRMT(int pos) function. Once in place the PLA is driven upward and into the medicine vial. Once the upper limit of the PLA is reached the inputted volume is then used to drive the HPLA downward, filling the syringe to the specified amount. Once this was done, the PLA was driven back down to its lower limit at which point the door was unlocked and system idled until the syringe is removed and the door is again closed. The door is then locked again and both the HPLA and the RMT are reset to their resting positions.

After the syringe has been filled and leaves the system the only step left is for the system to post the filling information to the remote database via the writeToDB function shown in Figure 42, that simple constructs a command with all of the fill parameters and tells the database to post the information. The database application takes timestamps the post instead of the microcontroller sending the timestamp with the filling parameters. This was changed because the microcontroller is not hooked up to any form of internet and therefore the internal clock has no reference to the time of day. By having the remote database capture the timestamp it ensures and accurate timestamp that is related to the time of day as the remote database is run off of an internet connected machine with a reliable system clock. Once the post is complete the loop returns to the top and starts polling for user credentials again, ready for another fill action. [EO]



```
void writeToDB(struct fillParams *fill)
{
    char command[100] = "7702";
    strcat(command, fill->medicineType);
    strcat(command, fill->userID);
    strcat(command, fill->patientID);

    char temp2[4];
    sprintf(temp2, "%f", fill->medicineAmount);

    int i;
    int j = 0;
    for(i=21; i<26; i++)
    {
        command[i] = temp2[j];
        j++;
    }
    sciSend(scilinREG, 25, (unsigned char *)command);
    waitforACK();
}
```

Figure 42: writeToDB() Code

The final form of the system achieved all of the engineering requirements in terms of microcontroller related functionality. The system successfully; took remote user input, filled a syringe well within 0.1mL of precision in under 30 seconds, and posted all of the relevant data to a remote database.[EO]

9.2) Relay Board

The multistage relay board design has stayed the same, but the components that were used to implement the design have changed. The first stage of three solid state relays are the same as in the original design. The relays used to implement the second stage have changed. EMR1 and EMR2 have been replaced by the same relay used as EMR3. EMR3 was rated to carry a current of 2 amps for the door locking solenoid, where EMR1 and EMR2 had a lower current rating. It was concluded via testing that these two relays were not sufficient to produce reliable, consistent results when controlling the PLA.

While using these insufficient relays the actuator would occasionally perform as expected but would not provide consistent results. If the relay would stay energized for more than a second or so driving the actuator outward, it would than hang-up and not de-energize. This resulted is a 12 volts being applied to both terminals of the PLA motor. With 12 volts at both terminals, the PLA would not be able to retract. Testing was performed using EMR3 as the driving relay for the PLA and the results were consistent and reliable. At this point, the best solution was to switch out the insufficient relays for relays with a higher current rating. [CB]

9.21) 2V Regulator

A 2V regulator was added to the circuit board in order to run microswitches. This was also done as to not send too much voltage to the micro controller. A LM317T from RadioShack was used. This required an RC circuit to yield an expected voltage. The regulator is rated for 1.2-37 VDC. With only needing a 2V output, the equation Vo = 1.25 (1+R2/R1) was used, which gave an output of 1.84 volts. This was acceptable since the input of the micro controller needs between a 1.2-3.3 VDC to work properly. [ZS]

9.3) Power Distribution

The original power supply design utilized a laptop charging cord as the AC to DC converter. Linear voltage regulators would then step down the voltage for power to the various electronic components. Upon implementation it became apparent that this power supply would not work. It was concluded that depending on the type of load applied to the power brick the operation became inconsistent. For example, the door locking solenoid is the device that requires the most power to operate, the power supply had no issues while running this device. If a different type of inductive load such as a motor was applied to the supply, it would fail.

The power supply was designed to charge the battery of a laptop and was not equipped to handle the noise that the motor injected back onto the line. Avenues were explored as to whether placing a capacitor on the line to help stabilize and provide a path to ground for high frequency noise would help. After discussion with the design team and Greg Lewis, it was concluded that it would be best to pursue other options for a power supply. A Mean Well SP-480-15 was used to achieve the proper voltages and currents. This power supply is rated to convert 100-240 VAC to 15 VDC and 32 Amps. The amperage of this device far exceed the needs for our system, but it was donated by the University of Akron to help achieve the needs of the system. After testing with the new power supply, with and without a load, the system works as expected. [CB, ZS]

9.4) Limit Switches

The original setup that was planned for the PLA limit switches, was to add a bracket on the side wall for the microswitches to make contact with. After having the structure build and most of the components in place, we came to the realization that the original design would be too bulky and maybe not fit. The new design utilizes the shelf as the upper limit; this allows the microswitch to be mounted on the side of the syringe plate. An initial concern was that the microswitch would not trigger the PLA to stop quick enough as to not cause any damage. After testing everything, the response of the microswitch was almost instantaneous. For the lower limit we designed a bracket to fit on the motor of the PLA. By doing this we were able to eliminate the side bracket. [ZS]

9.5) Locking Solenoid

The locking solenoid was one of the hardest components to get in working condition for the system. The first problem to arise was the electromagnet would be de-energized and the pin would stick in the solenoid, this was due to the small magnetic force that was still there. We experimented with using a second magnet to get the pin to drop out, but there was no consistency. The solution to the problem was an extended spring that was attached to the pin and solenoid. When the solenoid was energized, the spring will compress; and after being de-energized, the spring will extend to its original form and pull the pin out. This solution took time to figure out, but after being re-engineered the locking solenoid worked perfectly. [ZS]

9.6) LAC Modification

The HPLA is one of the main parts of the system. The LAC is what controls the HPLA. When the HPLA first arrived, we immediately began to test it. The HPLA is rated to be 99.6% accurate, with the normal settings. With these settings the HPLA would attempt to move to the correct length, and when it got there it would jitter and never find the exact length. After contacting Firgelli with the issue, they got back to us and advised us to put a 10kΩ resistor in series with the PWM signal. After retesting the HPLA, the correct lengths were being achieved every time. [ZS]

9.7) Housing

The 12x12x24 inch housing is made out of ¼ inch polycarbonate lexan, held together with a series of "L" shaped brackets and number 6 machine screws. The wooden base is made from two 12x12x.5 inch pieces of plywood. The front door and top lid utilize hinges that are mounted on the inside to improve the security of the housing. Mounting the hinges on the inside created a minor problem. It allowed for a small gap between the door and sidewall along with lid and back wall. Getting rid of this gap would allow the project to be more aesthetically pleasing along with improving its functionality. As a solution to this problem the team constructed a series of seam covers from a pleather material. Cutting a rectangle out of the pleather, folding it in half, inside out and sewing the two long ends together results in a cylinder. Turning this cylinder right side out, and puncturing holes for the hinge screws provided the best solution for the problem. The left hand side of Figure 43 shows the solution that the team developed. [CB]

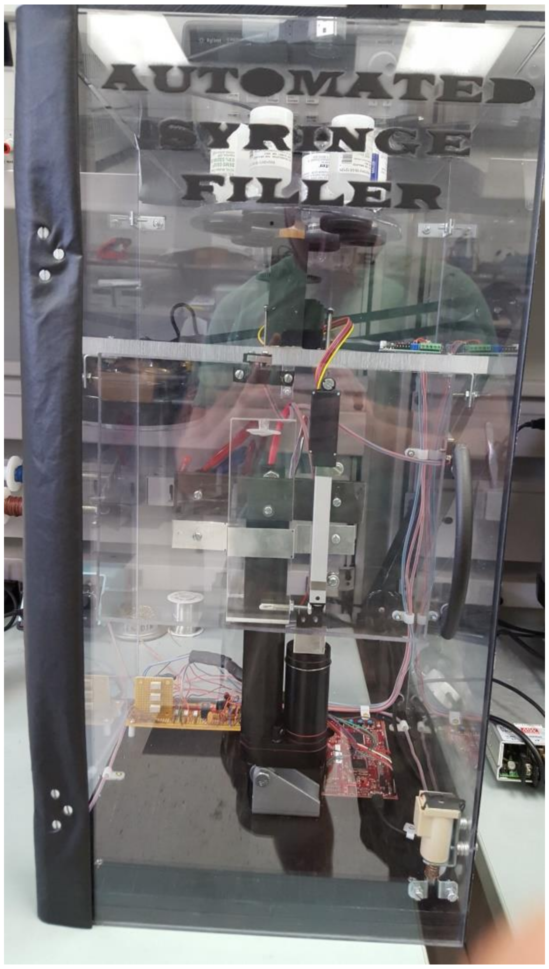


Figure 43: Front View

9.8) Syringe Holding Apparatus

The original design called for a Z-Shaped bracket that would attach from the plunger of the syringe to ram of the HPLA. During the implementation phase it became apparent that this would not be necessary. A simple rectangular piece of ½ lexan was sufficient to transfer the linear motion of the actuator to the plunger of the syringe. Figure 44 shows a close up view of the final syringe holding apparatus. [CB]

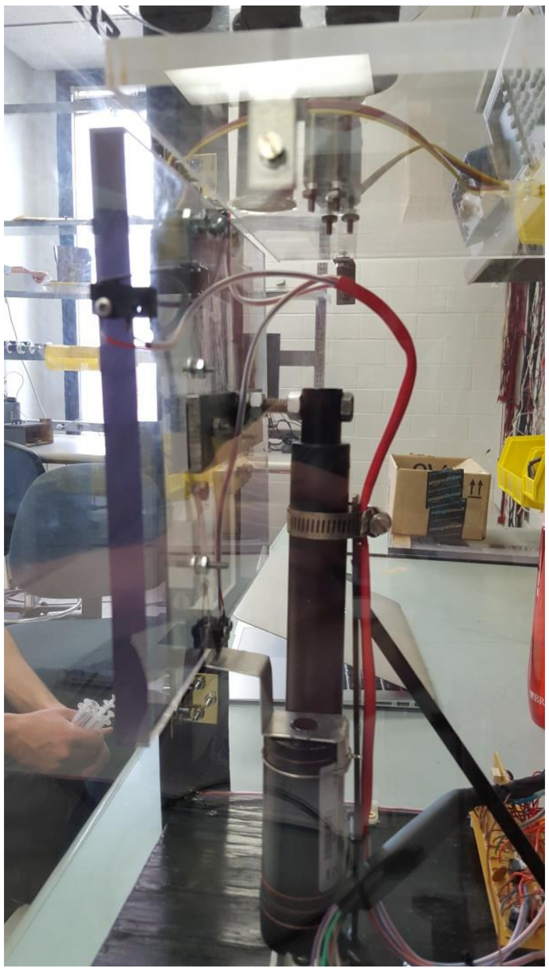


Figure 44: HPLA Front View

9.9) RMT Vial Holding Apparatus

The original design called for an elastic band fastened on each side of the vial and then stretched over the top of the vial, holding it in place. The concept behind this idea would have been sufficient but was not idea. The idea of using rubber grommets with an inside diameter equal to the outside diameter of the head of the vial to hold the vials was proposed. After further evaluation and discussion a suitable grommet was found. To implement these grommets a Dremel was used to create holes in the lexan that the grommets would fit into. The frictional force of the vials being pressed into the grommets was more than adequate to keep the vials secure. Figure 45 shows how this was implemented. [CB]

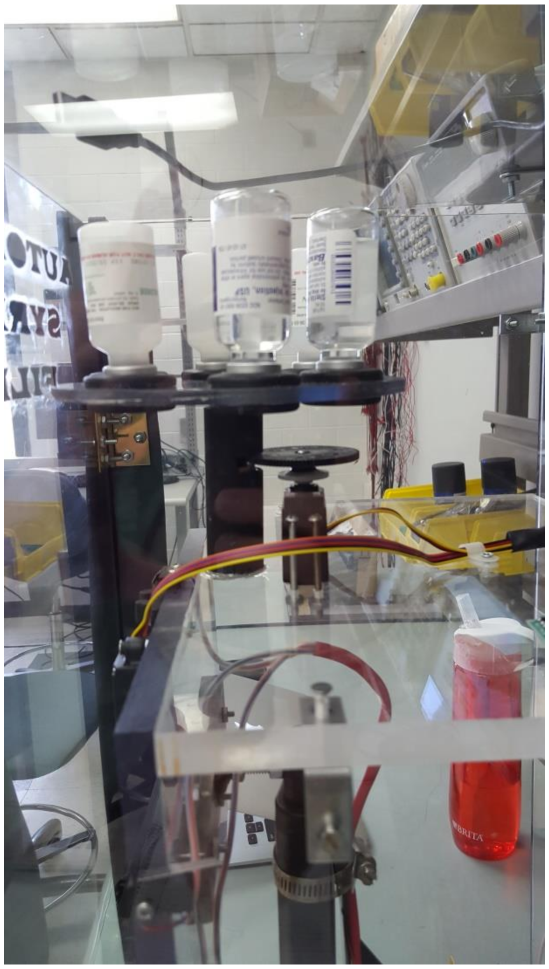


Figure 45: RMT Side View

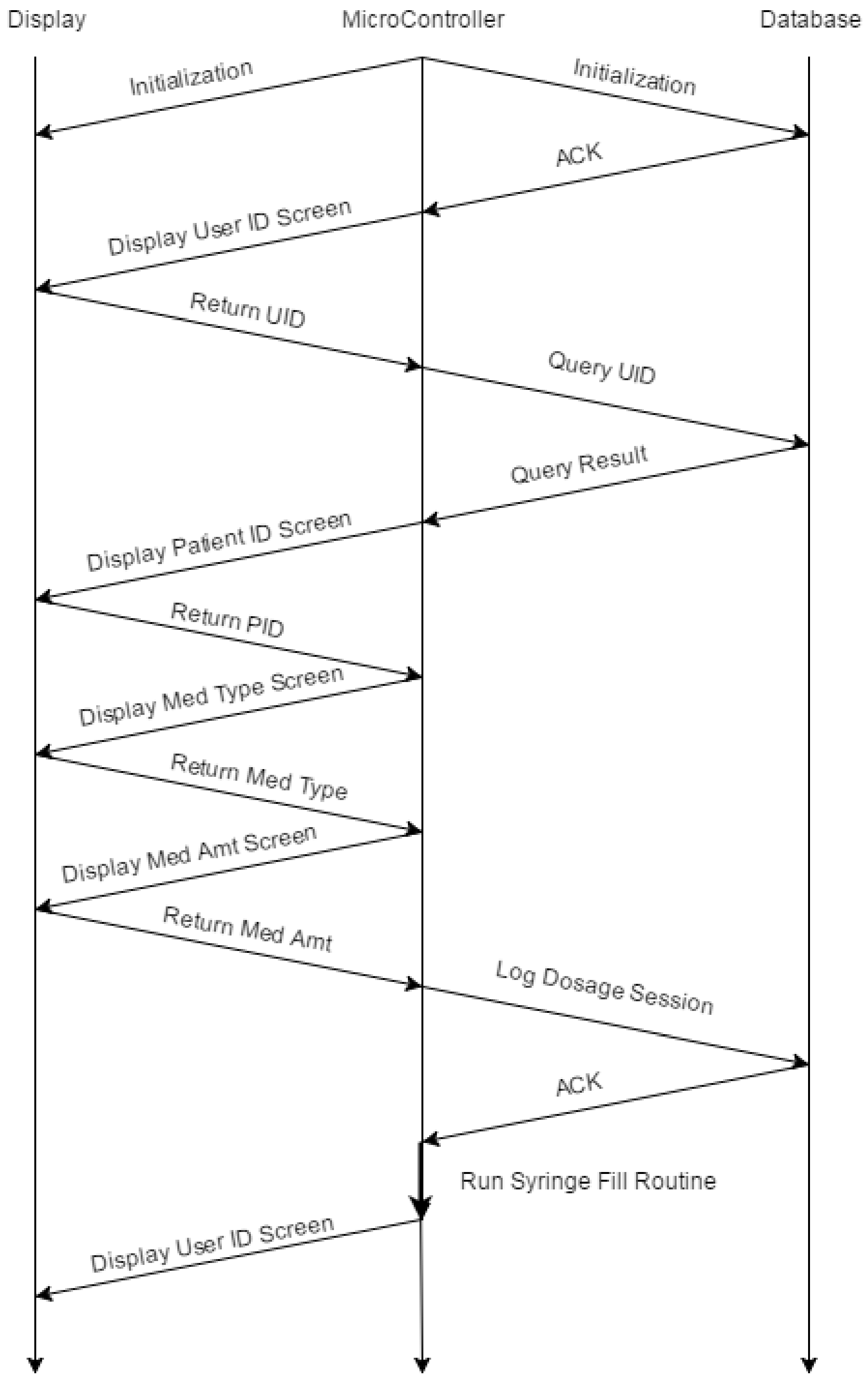
9.10) RMT Servo

The main issue that arose during the implementation phase of the design project was the deadband of the Servo used to control the RMT. It was stated that this digital servo had a 2us deadband, while testing out the alignment of the needle to the aperture of the vial it became apparent that this deadband was much larger. Some of this deadband can be attributed to the “slop” in the gearing between the servo and the tray but the majority of it belongs to the servo itself. A deadband of approximately 7us was experienced. Due to the budgetary constraints a higher quality servo was no available. [CB]

9.11) Database

The final database implementation changed from the original implementation slightly. The final database implemented 3 different types of tables: a patient information table, a user credentials table, and a new type of table called a dosage information table. The dosage information table was created so that the static information logged in the patient information table would not have to be logged multiple times. This would result in less data to be logged and a more scalable database. The new database is structured in such a way that the user credentials table is the same, the patient information table now only contains the patient ID an patient name fields, and the new dosage information table contains all the dosage information for a single patient. Therefore for every patient entry in the patient information table, there is now a dosage table for that patient tracking fills logged to that patients ID. The fields in the table are the User ID of the user filling the syringe, the Dosage ID that is a unique ID assigned to this dosage, the fill time where the time that the fill occurs is logged, the medicine type, and the medicine amount. This table allows for easier navigation and more scalability of the database. [DK]

The microcontroller was able to query the database to make sure patient and user IDs existed. This allowed for the verification of our user. Unfortunately there was trouble implementing the Ethernet as the Hercules RM46x board had made the claim that it was "Ethernet Ready" when in reality communicating via Ethernet required interfacing with an Ethernet transceiver. The transceiver acted as the PHY layer that the microcontroller communicated with. Due to the team's lack of experience with such low level Ethernet protocol, the Ethernet communication was scrapped rather quickly after initial attempts at getting it working failed. The backup option of using the serial communication port that the team was sure worked was used and a communication flow was established. The communication flow is as shown below. [DK]



[DK]

9.11.1) Database Cost Reductions

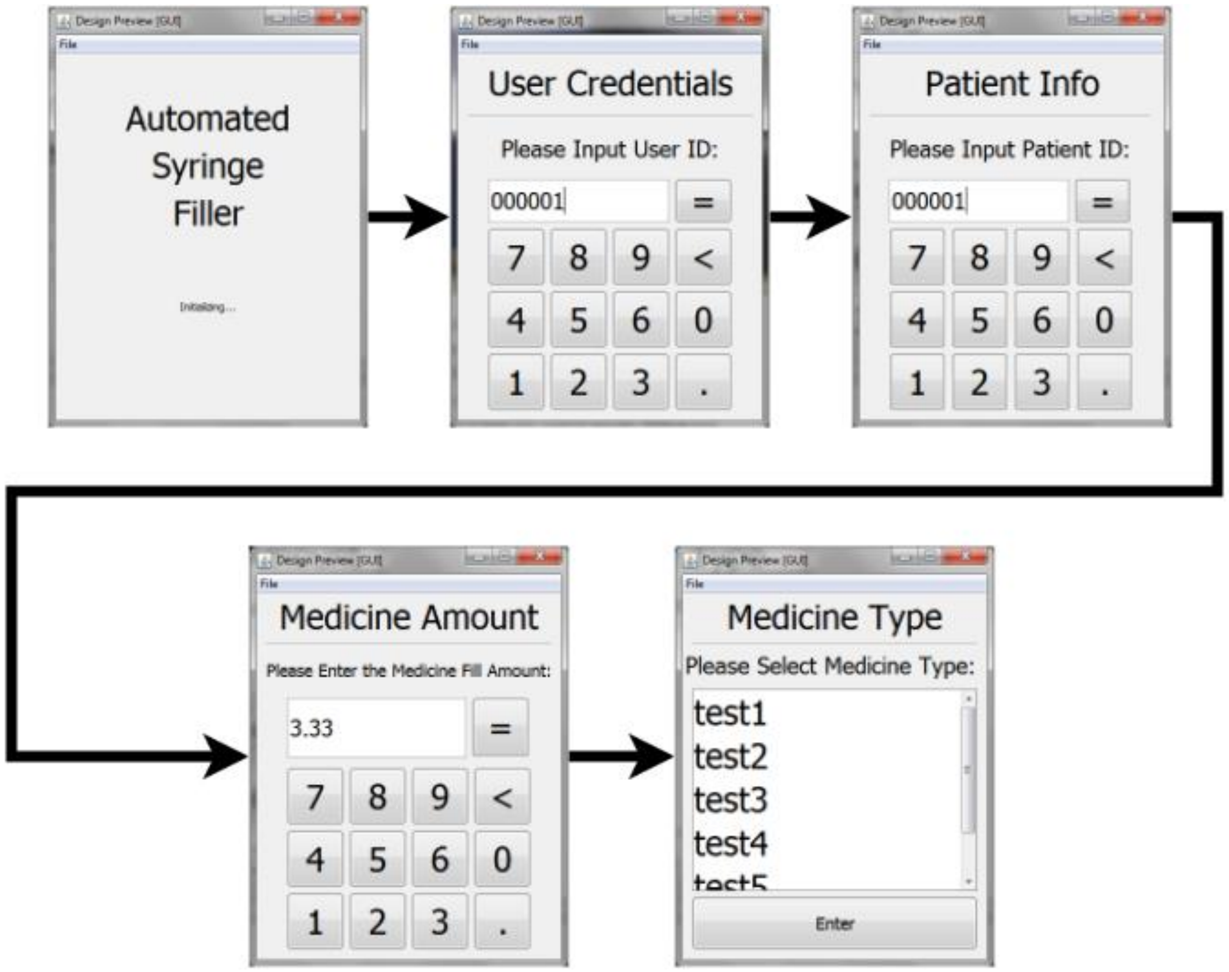
There is not much that could be done to reduce the cost of the database as the derby database software used is free and open source. A potential cost mitigation strategy would be to use cheaper dedicated servers than the desktop computer the team demoed on to host the database. [DK]

9.12) Display

The final display implementation was significantly different than the original plan entailed. The original plan detailed a touchscreen display that had several screens where the user would input the fill information. The final display was created to be a mock touchscreen display that was run on a computer. The reason for this change was an inability to get the touchscreen bought working with the microcontroller properly. The original touchscreen was to be communicated to be SPI. Unfortunately the driver supplied with the touchscreen did not seem to work properly, so the driver for the touchscreen was rewritten to work with the RM46x board. The display still did not seem to work properly beyond turning on. In a final effort to get the display to work the SPI pins were hooked up to an oscilloscope where the output was tracked. It was determined that the touchscreen was not working as expected and the team made the call to move on to other options as to not waste too much time on a display that may not work. [DK]

The second option, which was the option finally implemented in the final design, was to create a java display to mock what the touchscreen would look like and how it would function. The display was created using the Netbeans GUI builder and shared the serial connection to the computer that the database used. The microcontroller sent commands to the java application that signals the application to flip pages on the GUI. The entire control flow is talked about more in section 9.12. [DK]

The GUI flow was similar to the planned implementation with the exception of the removal of the confirmation screen and the addition of an initialization screen before any commands have been sent to the screen. The User Credentials screen has a keypad with the ability to accept mouse clicks to simulate user input. The Patient Credentials screen has nearly the exact same appearance and functionality as the User Credentials screen. Both screens only accept 6 digit numbers as input because that is the way user and patient IDs are formatted. The Medicine Type screen has a selection box where a medicine can be clicked on and selected in order to indicate the wanted medicine. The Medicine Amount screen has a field where the amount of medicine can be entered using a keypad and the user can only enter the format in as a number with the format x.xx. The Confirmation screen in the original design did not make it into the final implementation due to time constraints. The final screens look like the diagram in the figure below. [DK]



[DK]

9.12.1) Display Cost Reductions

To reduce the cost of the display a cheaper display could be designed. The display originally cost around $20 and did not interface well with the RM46x board that was used. The display could also be built manually as during the design process the team saw what appeared to be a cost effective method of building a resistive touch screen display with separate parts which wound up cheaper than the screen originally purchased. [DK]