Automated Medicine Dispenser

Medicine Dispenser with Timing Notification

Steven Wells, Erick Ochoa, Riel Fehr

Summary: A secure medicine dispenser, which communicates with a web server attached to pharmacist's database, featuring a Website Interface.

Group Number: 8
Availability: Mondays and Fridays

Abstract

Primary Mission: Save Lives.

Every year, people are hospitalized for incorrect use of their prescribed medication, which is taken for serious afflictions such as mental illness, various types of cancer, chronic pain and others. The current solution of depending on nurses and family members to ensure adherence and accuracy, isn't always plausible. For many people, their illness doesn't allow them to keep track of their own medication. We have developed a system that can potentially reduce the number of incidents every year by using an enclosed, automatic pill dispenser, with a website interface. The dispenser reads information from a pharmacy's web server, for which the "Medi-Code" is the key. The timer for each collection of medication is read by the dispenser and dispersed accordingly, prohibiting any chances of overdose. At this point in time, we have successfully completed the server-to-system communication (ethernet), a 3-port electromechanical system is finished and working, a full web interface is created with multiple features. The website contains timing schedules,

Table of Contents

Functional requirements of project

The goals for the automated pill dispenser are the following:

- To make medication easier by changing pills from a bottle to a cartridge.
- To make prescriptions less prone to human error by allowing the system to know the prescription schedule in an automatic fashion.
- To create a simple, small and readable file format that allows flexibility for dispensing pills are different pills.
- To create a system that is capable of displaying the user's prescription.
- To create an alarm system that is capable of reminding the users in case they have forgotten their pills.
- To create an easy dispensing mechanism that will allow users to access their medication with a single button push.
- To create a dynamic website that is capable of displaying the estimated time for the next pill to be dispensed and the amount of pills left.

Some of the goals were met with different degrees of completion. For example:

- We changed the pill bottles to cartridges. This is designed so that the user doesn't have to open pill bottles. This is useful for users with arthritis that have difficulties opening their medication. However, the design of the final prototype did not take into account the fact that this pill cartridges were to be removable. The final prototype has fixed cartridges which make loading of the pills a bit difficult.
- The original design included an RFID reader. RFID tags were to be placed with the removable RFID cartridges. This allowed the user to only hover the cartridges over the reader to obtain the ID, which would be a key to fetch the prescribed information from a pharmacist's web server. However, implementing the RFID was unsuccessful and this functionality was replaced by creating a HTML form on the user operated web server, embedded on the FPGA. It can be argued that this replacement is as equally error prone to user input as letting the user submit her prescription schedule into the machine because both methods involve the user interacting with the machine via a keyboard. The RFID would of originally removed any human errors by automatically reading the associated ID.
- The simple, and readable file format that we implemented is a text file with well defined fields. This will allows pharmacists to input the prescription in a way that machines would be able to understand.
- A web server was embedded into the FPGA in order to display a website to the users with their prescription's information.
- The LEDs available on the Altera DE2 board were used as an alarm system.

- Dynamic HTML was implemented on two specific routes on the web server. They show the remaining time until the next pill had to taken, name of medication, and amount of pills left.

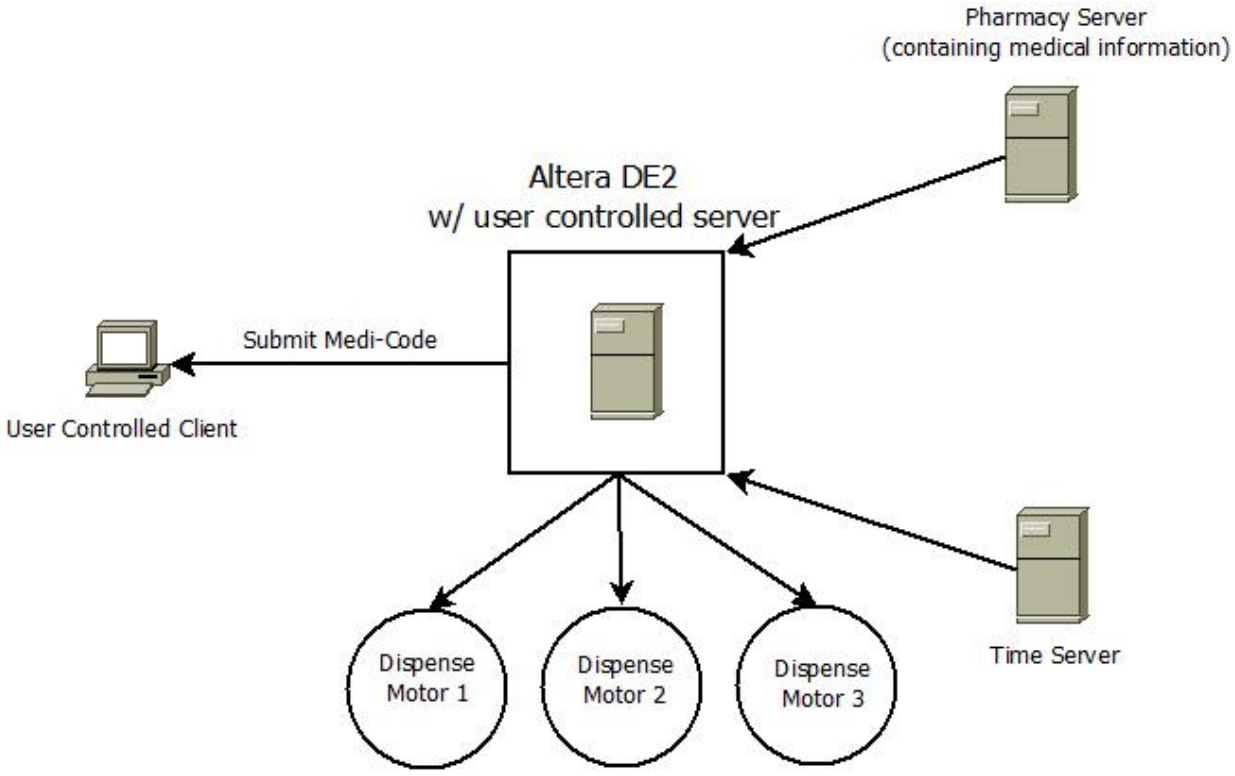
Design & Description of Operation

Peripherals used on DE2

Within the DE2, we chose to only use: ethernet peripheral, GPIO pins, LCD, buttons and the LEDs. We chose to use the ethernet in order to communicate with the pharmacy's server, time server and the client. The GPIO was chosen to operate the 3 servo motors required for dispensing the pills. This was the optimal choice as it easy to implement by simply writing the required VHDL module. The LCD was chosen in order to display current pill statuses and special instructions regarding the medicine being dispensed (ex: take with food). We chose to implement the LEDs to notify the user when the medication is ready to be dispersed and a corresponding button that needs to be pressed in order for the medication to be given out.

Overall Design of the System

The overall designed system can be seen here:



Our system is initiated when the user submits a "Medi-Code" from a web browser that accesses the user controlled server on the FPGA. Then the user's server (acting as a client) calls the

server located at the Pharmacy in order to obtain the required medical information corresponding with the submitted “Medi-Code”. The medical information is then stored onto the server located on the FPGA where it calls a 3rd party server in order to determine the local time, since there is no internal clock on the DE2 itself. This time is needed used to determine when the user needs their next dose. Once the local time is equal to the required dosage time, 8 green LEDs are illuminated to indicate to the user that the system is ready to distribute the necessary medication. When the user presses the “dispense” button, the DE2 sends the appropriate signals to the correct servo motors in order to push out the needed medication. When the pills have been distributed, 16 red LEDs may be on to indicate that the recent dosage requires special instructions. These instructions are then displayed on the LCD, notifying the user. The system then enters an idle state where it continues to check the time server every 30s while cross referencing with the next dosage time or the user enters another “Medi-Code” to virtually load any of the other cartridge positions.

Design of the Dispenser

The designed dispensing mechanism can be seen here:

The mechanism is designed to be encased in plexiglass in order to prevent any unauthorized access to the motors or pills.

Powered by a HiTec HS-635HB servo, the motor rotates in both the clockwise and counterclockwise direction, thereby dispensing a pill each time. The pills are being represented by PEZ candy along with their corresponding, inverted, PEZ dispenser which represents a pill cartridge. When the pills are loaded, the force of the spring inside the PEZ cartridge pushes the pills down into an ejection compartment which allows only one pill to reside in at a time. This allows for a single pill to be dispensed on each iteration of the motor.

The servo motors are being operated by a VHDL module that is a pulse width modulator (PWM), see Appendix B. The PWM alternates from 2 states (High and Low), each corresponding to extreme CW and CCW positions. By knowing the input clock’s frequency (50MHz), we were able to choose a specific output pulse width by counting a specific number of clock edges and setting the output high until the count reduces to zero. We sent these pulses at a period of 20μs, as specified by the HiTec servo motor data sheet. This was accomplished by using the same technique as the pulse width portion, except we reset the pulse width count every time the period count deprecates to zero.

Bill of Materials

Component Description	Quantity	Unit Price
Altera/Terasic DE2 development board	1	$517.72
PEZ dispensers	3	$3.00
Router**	1	$159.99
HiTec HS-635HB Servo Motors	3	$27.99
40-pin ribbon cable	1	$10
Ethernet Cables	2	$8
RFID tags**	4	$3.99
RFID reader w. breakout board**	2+2	29.95 + 0.95
'L' brackets	2	0.50
Particle Board for support	2 sq-ft	$10
¼" cabinet screws	42	$5.00 for 50 pack
Perf board	1	$7.50
Total	-----	$670.19

** => not included in total, as well as a 5V, 1A power supply.

Sources of Reusable Design

Name	Documentation	Source Size	Compiled Size	RAM
DM9000A Driver	Found on 2013W appnotes	20.9 kB		
Interniche TCP/IP Stack	Vendor's website		12 kB	
UCOS II RTOS	Vendor's website		6 kB - 20 kB	+500 bytes

The DM9000A Driver is needed because the driver that is generated by the BSP script does not provide the functions needed by Interniche TCP/IP Stack to initialize and control the device. The driver generated by the BSP allows us to control the hardware, but without the abstraction of sockets.

Interniche TCP / IP Stack allows us to use socket libraries. These are useful because we would otherwise need to create a finite state machine that keeps track of the communication exchange through ACKing packets.

UCOS II RTOS will allow us to create threads and synchronize them in a proper way.

Datasheet

I/O Signals

Device	***Signal (pin#)***	***Description***	***MAX Voltage***	***MAX Current***	***Connected to:***
Servo Motors (HiTec HS-635HB)	Ground	Zero volts supply pin and the common ground.	n/a	n/a	GPIO1(30)
	VCC Supply	Voltage required to operated the motor	+6.0V	2500 mA	Voltage supply
	Data	The motor is controlled with a square wave with a pulse width of 1.1 ms - 1.9 ms	5V pk-pk		GPIO1[32, 34,36]

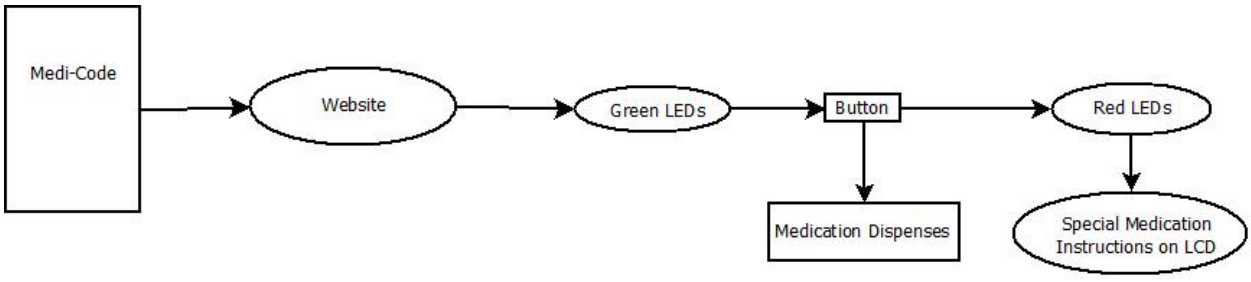
Device	***Signal (pin#)***	***Description***	***Connected to:***
DE2	GPIO1 (32)	This pin will act as the output data controlling Motor1.	Data Pin of Motor1
	GPIO1 (34)	This pin will act as the output data controlling Motor2.	Data Pin of Motor2
	GPIO1 (36)	This pin will act as the output data controlling Motor3.	Data Pin of Motor3

	GPIO1(30)	Ground	all gnd of external devices
	Ethernet	Used to connect to multiple servers and clients	Router

Performance

The performance of the system is perfectly acceptable when the board is programmed through JTAG every time, i.e., hooked up to the computer and run from eclipse. In this case, all times are negligible besides the webpage refresh rate and the motors dispensing. It takes approximately 1 second for 2 pills to be dispensed, and 1 second per webpage selection, these are the bottlenecks of user perceived performance. (Power consumption analysis of the system is located in the sustainability section later in the report). We had huge performance issues when we moved all software to the flash memory, as mentioned in testing. When running from flash memory, all components work correctly, just not in a reasonable timely manner. The startup time (defined as being the first time we see the LCD idle display of the current pill counts), takes approximately 2.5 minutes. After this, we enter a medi-code in the website and it takes anywhere from 3-4 minutes for the alarm to turn on, where it should normally take no more than 10 seconds. After the alarm is on and the button is pressed, even the motors work slower, but the system finishes the request not much slower than it would with JTAG from this point. Lastly, since the board is hosting the website, it becomes so slow that it is unusable leaving no choice but to run the system from JTAG.

User-perspective Block Diagram



Background Reading

Our project idea sprouted from a “statistical report [showing] that almost 55 percent of elderly people in US fail to adhere to their daily medication routine [and] 26 percent of [those] errors [were] severe”[1] within a year. Using our project would eliminate a major problem associated with elderly patients suffering from physical and/or mental disabilities.

With the idea of creating an automatic pill dispensing machine, we contacted a professor who works within the Pharmacy department at the University of Alberta to find how patients generally interact with their medication on a daily basis. One of the major features that we were informed to implement was flexibility, as all patients live different lives. Therefore, we designed a user friendly website that allows the ability to choose what time specific events are during the day (ex: Lunch is between 12pm - 2pm). We were also informed that implementing pharmacy-loaded medication cartridges is possible, since pharmacists already load third party packages such as blister packs. Another small, but important feature that we added because of the professor’s input, was the ability to have the machine dispense the pills only on the user’s queue (ex: on a button press). Instead of our previous design, where the machine gave the required dose regardless if the user was around.

We followed up by contacting two more professors that work in the Computer Science and Nursing Departments at the University of Alberta and are currently working in tandem designing a Smart Condo that is trying to implement a Smart Pill Dispenser. They helped us pick out specific features to add/remove in our design at the time. For example, they instructed us to remove a “pill taken” status that would attempt to remind or notify 3rd parties when a pill hasn’t been removed from the bowl. It was later removed because it is a waste of resources to enforce pill adherence. They also advised us to have the apparatus secure (ex: have the machine encased in a secure material) to eliminate unauthorized access to the medication.

We decided to design our project with the intention of having universal and interchangeable pill cartridges in order to eliminate any possibility of the user having contact of the medication prior to their scheduled dosage. We designed our system to have universal pill cart

The use of universal pill cartridges is going to be simulated with multiple PEZ dispensers and PEZ candy acting as different pills. We are going to modify these dispensers by removing the head/lever system allowing for the servo motor, which is pushing the pill out of the cartridge, to be implemented.

Our project will require a mechanical aspect in order to eject the PEZ pills. This is going to be implemented using multiple servo motors controlled by the DE2 board. Each motor will be assigned to a specific PEZ dispenser (acting as pill cartridges) in order to push the pill out to the user’s tray. The motors being used are the Hitec HS-635HB servos. These motors allow for a clockwise and counterclockwise rotation (total of 80°) controlled by a pulse width modulator ranging from 1100us to 1900us. The motors also require a 4.8V to 6.0V power supply, which has a stall current ranging from 2.0A to 2.5A, respectively. We have chosen to power the motors with an external power source, because having a direct connection to the GPIO would cause too

[1] Jyothi K. Vinjumur. 2010. Web based medicine intake tracking application. Page 1

much strain on the DE2. These motors provide a minimum stall torque of 5.5 kg.cm (with a 4.8V voltage input), which, as demonstrated by basic tests (pushing heavy objects on a table), will be more than sufficient for our project.

We are using the Altera IP Ethernet and GPIO modules. The GPIO module will allow us to communicate with the RFID reader providing it with the required 3.3V as well as the needed data pins in order to send the read information from the corresponding tags. The ethernet module will be used to access the internet to obtain information from the pharmacist's web server.

Software Design

Definitions

1. **New Cartridge Thread:** One of four main threads. It will deal with getting the value associated with the key (the RFID data) and storing it on a text file.
2. **Cartridges data structure**: Shared structure between threads. It will store information about the state of the pill cartridges including: dosage, frequency, times of day, special instructions, name of medicines, and quantity of pills remaining. It needs a semaphore to be accessed. The file format is specified on the following section.
3. **Record**: A record is the granular information unit of pill cartridges. It will contain all of the information fields regarding a specific pill cartridge including: name of pill in cartridge, quantity left, dosage, limit, frequency and several others. See below for all the fields.
4. **Field**: A record is composed of fields. These fields contain one unit of information useful for determining the whole state of the record. For example one field is name field. It contains the name of the drug. Another field is the "allow in morning field". See below for specification of fields.
5. **Alert Thread**: One of the four main threads. Alert thread will periodically check the cartridge data structure. The Alert Thread will query first, if the user should be alerted, and second if the user has taken her pill. If both of these conditions are met, an alert will be issued.
6. **Motor Thread**: One of the four main threads. Motor thread pend on a button press. The user will be able to press a button and then the thread will check the cartridge data structure. The thread will first query to find out whether or not the user is allowed to get pills at that moment. Then query to find out whether or not a pill has already been dispensed for that time frame. Only if the user is allowed to get their pills and they have not been dispensed more than the allowed limit, can the user get their pills according to the dosage specified on the cartridge data structure.
7. **Server Thread:** One of the four main threads. It will support HTTP 1.0 GET and POST requests. The GET request will serve an html file containing a form for updating the user's time preferences. The POST request will update the user's time preferences. The POST will also allow us to add different prescriptions via the medicode.
8. **HTML File:** The file that will be served by the server thread.

Specification for the pharmacy file

The record will be specified as follows: rows are separated by newline characters. columns are separated by tab characters. This file will be parsed and stored locally as a structure.

	Col 1	Col 2	Col 3	Col 4	Col 5	Col 6	Col 7	Col 8
Row 1 <Name of drug>	<string>							
Row 2 <Time frame>	MORN	NOON	ANOON	EVENING	NIGHT	MNIGHT	LNIGHT	WHEN
Row 3 <Dosage>	<int>	<int>	<int>	<int>	<int>	<int>	<int>	<int>
Row 4 <Dispensed>	<int>	<int>	<int>	<int>	<int>	<int>	<int>	<int>
Row 5 <Limit>	<int>	<int>	<int>	<int>	<int>	<int>	<int>	<int>
Row 6 <Rate>	<int>	<int>	<int>	<int>	<int>	<int>	<int>	<int>
Row 7 <Rate pos>	<int>	<int>	<int>	<int>	<int>	<int>	<int>	<int>
Row 8 <Quantity in cartridge>	<int>							
Row 9 <Instructions>	<String>							

The **time frame** row specifies the information, which is relevant at the moment. It's values are short names for common times of the day. The real time period, which corresponds to these day times, can be changed within the source code or by the user via a web interface. For example, we can change the morning time period from its default range (8:00 am to 10:00am) to a more friendly one for someone who is not an early bird (9:00 am to 11:00am).

The **dosage** row specifies how many pills should be dispensed per time frame. For example, take one pill or take two pills. The pills will be dispensed based on the time frames specified in row 2.

The **dispensed** row specifies how many doses have been dispensed. The reason why this is not just a Boolean value is because sometimes doctors specify a maximum limit, for example painkillers ("Take no more than three to ease the pain.") This row will be updated every day at 4 a.m. to reset to zero. This is so because every day we need to start dispensing again.

The **limit** row specifies how many doses the user is allowed to take within a time period.

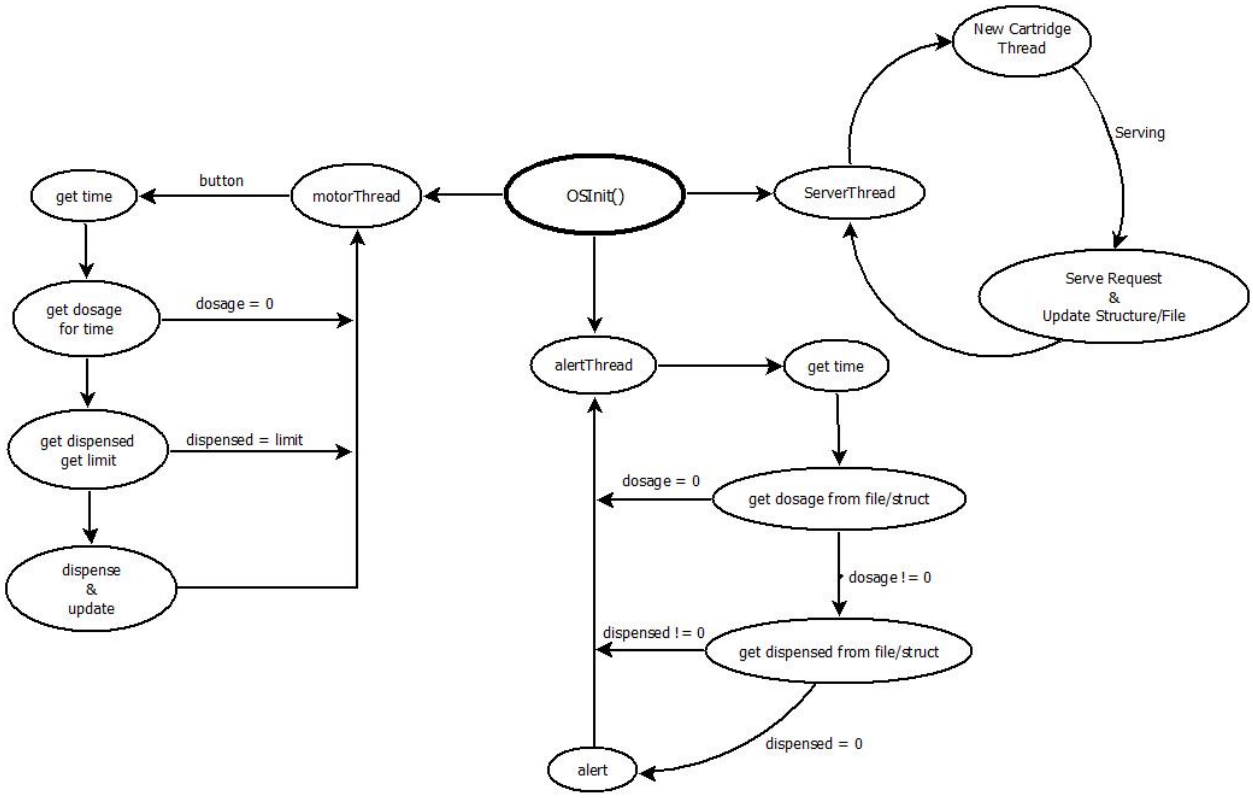
The **rate** row specifies how often the dosages are to be dispensed. For example: a value of 0 means that the dosage should be dispensed daily. If it is one, then it means every other day. If it is two then skip two days before dispensing the next dosage.

The **rate position** row specifies what day in the current rate the machine is. This value needs to be updated daily by an increment. When the rate position is zero, that means that in this day, the pill will be dispensed (or has been dispensed). After position zero, the rate position will acquire the value of the rate field immediately above it. It will decrement each day. The entire row will update every day at 4 a.m. to reflect a day change.

The **quantity in cartridge** row specifies how many pills remain in the cartridge. This value will be updated every time there is a dosage dispensed. The value will decrement by the value in dosage at the specified time frame.

The **instruction** row specifies special instructions such as "before meals" or "after meals". The special instructions will be displayed on an LCD.

Finite State Machine Diagram



Description of FSM Diagram

The system will have four main threads. It will have, what will be known as **New Cartridge Thread,** the **Server Thread**, the **Alert Thread**, and the **Motor Thread.**

The **New Cartridge Thread** will be initialized by a low **Server Thread**. When **Server Thread** detects an HTTP POST into the uri that handles new cartridges, then it will initialize the **New Cartridge Thread.** This will result in the following actions.

- Perform an HTTP/1.1 GET request with the pharmacy webserver.
- Listen on the socket to obtain the body of the HTTP response.
- Query **Cartridges data structure**
- Add the information to the data structure.

The **Alert Thread** will be initialized before OSInit() function is called. Its main purpose is to query the cartridges structure. Specifically, for each record in the cartridge structure, it will access the dosage field associated with the current time. Current time is obtained through a web server that serves time.

Once the dosage field has been checked, we check the dispensed field. If dispensed field is 0, then we alert the user.

The **Motor Thread** will be initialized before OSInit() function is called. Its main purpose is to wait for a button press and then start executing its task. Its task will be to query the cartridge structure and access each record's dosage field. If the dosage field is within the current time, it will compare the dispensed field with the limit field. If the dispensed field is smaller than the limit field, then it dispenses pills to the user. It then updates the cartridge structure to reflect changes. Specifically it will increment the dispensed field by one.

The **Server Thread** will be initialized before OSInit() function is called. Its main purpose is to serve the **HTML file** which contains a form for updating the time preference of the user. When the user clicks the submit button it will send an HTTP POST request. The Server will parse the response and update the values that were submitted. This function does blocks when listening to the socket. The user is expected to use her browser of choice to display the **HTML file** correctly.

Status

Dynamic HTML

- Paths on URI were selected in order to determine if they need dynamic HTML.
- If dynamic HTML is needed, then we read the file template associated with this resource.
- Then we load the entire file onto a string.
- The template has placeholders for variables which we need to change. The placeholders are a string in the form of two opening curly braces, the unique token which will identify which value has to be printed there, and two closing curly braces. (I.e. "{{ token }}")
- Then the location beginning and end location of the first token are found. This is used to replace the token with the actual value that we need.
- We repeat the previous step until all tokens have been replaced and no more are found.
- Then we serve this string as if we were serving a file.

Test Plan / Results

Ethernet

In order to test if Ethernet interface was working, we pinged the device. The device replied back. A file was also hosted on the DE2 board using the Web Server Demo, and we were able to access it from the computers in the lab.

Software

The **web client** first communicated with Google, and received Google's landing page. The html document was printed to stdout.

The **parser** was fed the **example.txt** , text file with an example data as specified by the format NODB.txt, via stdin. We are expecting the pharmacist's web server to supply this file. The parser was able to correctly assign values from this plain text file into a C struct capable of holding this information as specified by the format.

I was able to set a web server that serves just **example.txt**, and the client was able to download the example.txt and feed it to the parser.

RFID System

The expected operation of the RFID system will first, need to test the Tag in Range interrupt to alert the system to be ready for data collection. Second, the board needs to receive the RFID number, and lastly the checksum needs to be checked according to the associated convention for the RFID tags.

The tag ID's look similar to this: 0C000621A58E, where the last byte (8E) is the checksum.

```
0C      =       00001100
00      =       00000000
06      =       00000110
21      =       00100001
A5      =       10100101
              ----------
CHECKSUM =  10001110 (8E)  → Therefore the data was received without corruption.
```

The RFID system was not fully completed for use in the final system. We were able to get some data sent by the tags using a different uart configurations on the board, but the data received did not hold any consistency or follow the correct format above. We were getting random bytes that have no correlation to the above protocols, using the university program uart and the general uart. We also designed our own uart and implemented it on the board, but it again, gave the same results. This was very interesting that we received all of the same results for different tests, but this confirmed there was nothing wrong with the uart cores themselves. I then resorted to using signaltap, a tool given by quartus to analyse individual internal signals for debugging. Yet again I found that the uart was outputting correct values to the FPGA, and the signals were being corrupted in the FPGA somewhere. In the end, we were unable to fix this problem in time so we resorted to a user interface on the system website for entering the "Medi-Code" that would otherwise be input by the RFID tag.

Servo Motors

The written VHDL code (see Appendix B) that is going to be implemented to control our motors was tested with ModelSim. With the clock set at 50 MHz, the enable/data signal was set high a 2 times with the intention of moving the motor from -40° to +40° and returning to the -40°

position when the second data signal goes high. As seen in the provided waveform, in Appendix C, the results were as expected with the coe_motor signal being set to 1 for a required width of 1900 µs (+40°) and 1100 µs (-40°).

To test the motors, they must rotate to their extremes in both the clockwise and counterclockwise directions, on command. This is going to be accomplished by sending the motors all combinations of 3-bit data and observing the motor's reactions.

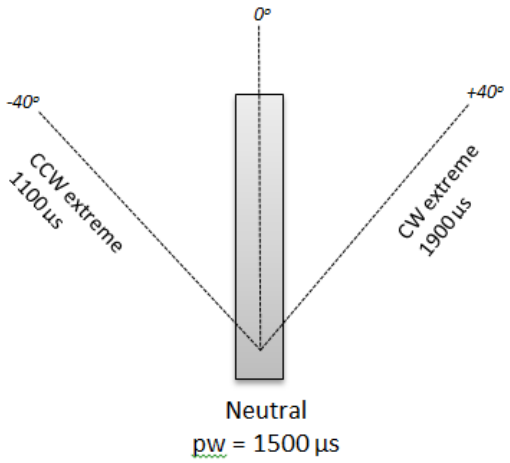


Figure 1: Possible servo motor positions

3-bit Input	*Initial Position*	*Expected Motor Operation*	*Expected Final Position*
000	-40° or +40°	No motor movement	-40° or +40°, respectively
001	-40°	Right motor (Motor 3) will turn it's maximum (80°) rotation in the CW direction.	+40°
001	+40°	Right motor (Motor 3) will turn it's maximum (80°) rotation in the CCW direction.	-40°
010	-40°	Middle motor (Motor 2) will turn it's maximum (80°) rotation in the CW direction.	+40°
010	+40°	Middle motor (Motor 2) will turn it's maximum (80°) rotation in the CCW direction.	-40°

011	-40°	Middle and right motor (Motors 2 and 3) will turn it's maximum (80°) rotation in the CW direction.	+40°
011	+40°	Middle and right motor (Motors 2 and 3) will turn it's maximum (80°) rotation in the CCW direction.	-40°
100	-40°	Left motor (Motor 1) will turn it's maximum (80°) rotation in the CW direction.	+40°
100	+40°	Left motor (Motor 1) will turn it's maximum (80°) rotation in the CCW direction.	-40°
101	-40°	Left and right motor (Motors 1 and 3) will turn it's maximum (80°) rotation in the CW direction.	+40°
101	+40°	Left and right motor (Motors 1 and 3) will turn it's maximum (80°) rotation in the CCW direction.	-40°
110	-40°	Left and middle motor (Motors 1 and 2) will turn it's maximum (80°) rotation in the CW direction.	+40°
110	+40°	Left and middle motor (Motors 1 and 2) will turn it's maximum (80°) rotation in the CCW direction.	-40°
111	-40°	All motors will turn it's maximum (80°) rotation in the CW direction.	+40°
111	+40°	All motors will turn it's maximum (80°) rotation in the CCW direction.	-40°

Integration Testing

When our mechanical dispensing system was completed, with the motors installed, we tested the workings of the entire project. This included testing the alarm LEDs, dispense button, LCD and the motors all working as one unit. We created a text file representing the file gathered from the pharmacy server, titled "medicode_a", we then entered this into the website on the "Enter MediCode" tab. Once the code is entered, the file is parsed and all scheduling information

is uploaded to the web page for reference, but also to the board for alarm timing. Once all information is saved, it is then analyzed constantly so when the next pills are to be taken, the green “alarm” LEDs light up to signal the user to press the dispense button. Once the button is pressed the required pills are dispensed and the “alarm” LEDs turn off. Once the button is pressed, the system checks for any special instructions contained in the information file pertaining to the drugs being dispensed. If so, the system turns the 18 red LEDs on and displays the special instruction onto the LCD. Once the screen has been cleared (approx 15s), the LCD then continues to display the current pill count of each cartridge.

The above description is the ideal case of what should happen in any given time, we had to still test sections of the system even after integration was implemented. We first tested the motors, LEDs and button together to ensure that when the green LEDs are on, the button is pressed, and the motors all work quickly and correctly. We had to test this for all different cases where some have multiple pills and some have none, which we concluded with our system working correctly in this area.

We then needed to test all the sections of the website which included a schedule page, a current contents page, “Medi-Code” insertion page, and time definition customization page. We tested each of these pages’ interaction with the pharmacy’s file served individually. The schedule page displayed the medication name, dosage time, and time until next pill, all corresponding to the correct cartridge slot number. Next, the current contents of the pills had to display the correct pills before and after dispensing. When this was confirmed, we had already been testing the medi-code page, since that is how the webpage knows what files to grab. Lastly, we needed to change the definition of some time sections, for example, changing the meaning of morning from 6:00-8:00 to 8:00-10:00. This had to be tested using the already working alarm component, if a morning pill alarms at the new time section, then it works correctly. After determining all of these parts worked together, we concluded our system was working to our original plans, besides the implementation of the RFID device. (Refer to Appendix: F)

Safety and Environmental Impact

Our project has no major safety concerns towards the developers or the users; in fact, one main focus of this project is to reduce safety concerns with medication adherence.

The maximum output and input voltage for the DE2 board, GPIO pins as well as the RFID reader is 3.3V. The maximum voltage required for the servomotors is 5.5V.
There are no hazardous substances contained in our project

Sustainability

These are considering maximum values to show maximum possible power consumptions. Although true ratings will not be much lower.

Power Consumption = [Current * Voltage] * time

Motors:

Active Power = (500mA) x (6.0V) = 3 W * 3600s = 10.8 kWh
Stall Power = (2500mA) x (6.0V) = 15 W * 3600s = 54 kWh

DE2:

Running the helloworld.c using the niosII_e(efficient) processor
Measured Power = (0.446A) x (9.0V) = 4.014 W * 3600s = 14.4 kWh

The **duty cycles** of the board are programmed to be 50%, which greatly exceeds the duty cycles of the motor, therefore that will be the main duty cycle considered. Realistically speaking, the motors do not have a duty cycle, since they are turned on at only specific times, and the active time of the motors are relatively negligible.

A rough estimate of the average power consumption of the system, approximating the motors to be in stall mode for the majority of the time, over 24 hours would be:

Power = (54 kWh + 14.4 kWh) * 24 hours = 6480 kWh/day

What area of fixed solar cells in Edmonton could power your project continuously?
Given a modern solar cell produces about 9 [W] / square-foot providing output for about 5 hours per day, equating to 6.4 kWh/sq-in. This means a cell of size 82 sq-ft could power it daily. This number seems quite large, which leads us to believe the system is quite inefficient.

Torque Calculations

Servo power is stated as ounce-inches (oz.in.). This is the maximum amount of power the given servo will apply with a one inch arm. For example, if a servo has a power rating of 16 oz.in, the maximum amount of power it will be able to apply with a 1” arm will be 1 lbs.force.

An HS-635HB has 83 oz.in of power at 6.0V
83 / 16 = 5.19 lbs.force
5.19 pounds of force with a 1” arm

If you were to put a 2” arm on the HS-635HB:
5.19 / 2 = 2.59 lbs.force
2.59 pounds of force with a 2” arm

References

[1] Jyothi K. Vinjumur, Eric Becker, Shahina Ferdous, Georgios Galatas, and Fillia Makedon. 2010. Web based medicine intake tracking application. In Proceedings of the 3rd International Conference on PErvasive Technologies Related to Assistive Environments (PETRA '10), Fillia Makedon, Ilias Maglogiannis, and Sarantos Kapidakis (Eds.). ACM, New York, NY, USA, , Article 37 , 8 pages.

[2] Solar Estimate.org. Accessed: 2014, March 4. Solar & Wind Energy Calculations: The (very) Basics. Available:
http://www.solar-estimate.org/?page=solar-calculations

[3] Dispensing device for tablets, by E Haas. (Nov 12, 1969). Accessed 2014, Feb 9. Available:
http://blog.pauldrapeau.com/wp-content/uploads/2011/08/patents.png

[4] Altera 10-Gbps Ethernet MAC MegaCore Function user guide. Updated February 2014. Accessed: 2014, February 18. Available:
http://www.altera.com/literature/ug/10Gbps_MAC.pdf

[5] Thing Magic, A division of Trimble. “Why use RFID”. Accessed: 2014, March 15. Available:
http://www.thingmagic.com/why-use-rfid

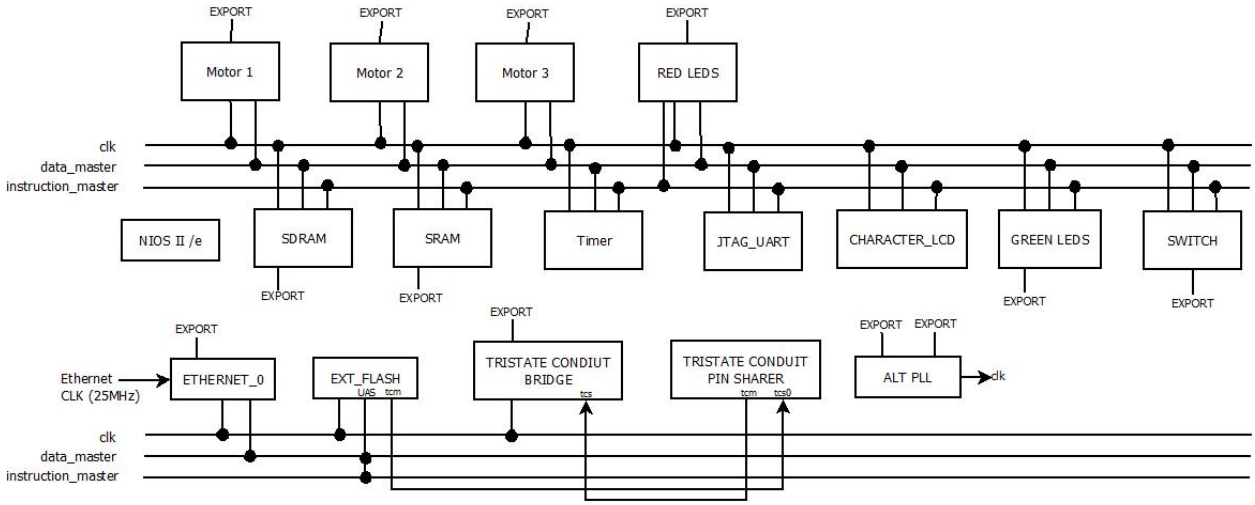
[6] “A Simplified VHDL UART”, Accessed: 2014, March 25. Available:
http://esd.cs.ucr.edu/labs/uart/uart.html

[7] Free BSD. “Serial and UART tutorial, by Frank Durda. November 12, 2013. Accessed: 2014 March 21. Available:
https://www.freebsd.org/doc/en/articles/serial-uart/

[8] Way 2 Tutorial. “Dynamic HTML”, Accessed: 2014, March 30. Available:
http://way2tutorial.com/html/dynamic_html_references.php

Appendix

Appendix A: Hardware

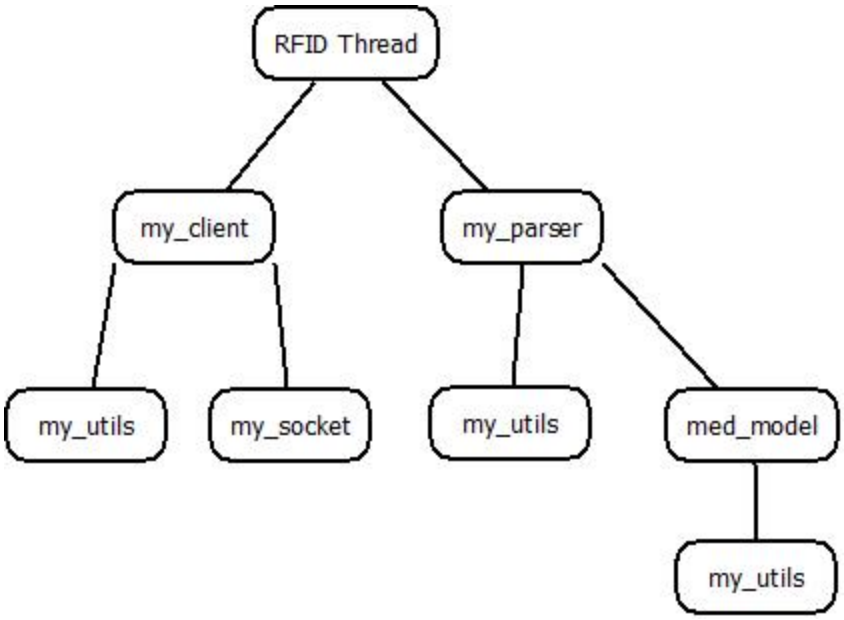


Appendix B: Software

Most of our software is on a github repository. Here is a link to the github repo.
https://github.com/efferifick/ece-492

*Please note that due to historical reasons, the name RFID thread is the name of the New Cartridge Thread.

Hierarchical diagram of C code.



This basically means that modules include other modules and use part of their code.

The C and H files of named my_utils, contain utility functions and macro declarations that are necessary for most, if not all, other files. For example, it declares a macro named MAX_STR that declares the unit size of strings which we handle.

med_model file contains the data type for the medical records.

my_socket file contains socket functions with proper error handling.

my_parser file contains parsing functions.

rfid_thread file calls functions from my_client and my_parser.

All of the code currently passes the tests.

There is also a vhdl file for pulse width modulation, which talks to the servo motors. It is included here.

VHDL Code for Motors

```
--############################################################
--
--ECE 492 - Automated Pill Dispenser Project - Group 8
--Written by: Steven Wells
--
--Date: March 3, 2014
--Description: Move a Hitec HS-635HB servo motor to/from it's
-- furthest counter-clockwise state to it's furthest clockwise
-- state by altering the pulse width to/from 2.3ms and 0.6ms.
--
--Notes:
-- - This code is written to work with a 50MHz clock
--############################################################

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity gpio_servo_pwm is port(

        clk : in std_logic;
        reset  : in std_logic;
        avs_s0_write_n: in std_logic;
        avs_s0_writedata: in std_logic_vector(7 downto 0);
        coe_motor : out std_logic
);
end entity gpio_servo_pwm;
```

```
architecture Behavioral of gpio_servo_pwm is

	type state is (low, high);

	signal current_state:state := high; -- Set as default

begin

	p1: process(clk, reset, avs_s0_write_n)
		variable period: integer := 1000000; -- Set for 10^6 falling edges (clock speed set at 50MHz) = 20ms period interval
		variable count: integer := 115000; -- Set as default in order to move the motor to a position we know (CW extreme @ 2.3ms PW).
		begin
			if reset = '1' then
				count := 0;

			elsif falling_edge(clk) then
				-- Every rising edge, count down the period and count.
				if (avs_s0_write_n = '1') then
					if (period > 0) then
						period := period - 1;

					-- Decrement count while setting the output HIGH
						if (count > 0) then
							count := count - 1;
							coe_motor <= '1';
						elsif (count = 0) then
							coe_motor <= '0';
						end if;

					-- When the period count becomes 0, reset the period and reset the count in order to reciprocate the previous pulses
					-- over and over.
					elsif (period = 0) then
						period := 1000000;-- Resetting the period to 20ms
						case current_state is
							when low =>
								count := 30000; -- Resetting count, depending on what the current state is set to
							when high =>
								count := 115000; -- Resetting count, depending on what the current state is set to
						end case;
					end if;
				elsif (avs_s0_write_n = '0') then
					case current_state is
					when low =>
						current_state <= high;
						count := 115000;
					when high =>
						current_state <= low;
						count := 30000;
					end case;
				end if;
			end if;
```

```
        end process;
end behavioral;
```

Appendix C: Simulation waveform for VHDL motor code.

Appendix D: Quick Start Manual

The Automated Pill Dispenser is intended for two parties: end users and pharmacies. In order to set up, there are several things that have to exist a priori. Here is a list of things that have to exist:

1. The pharmacy has to have a web server.
2. The user must know the IP address of the pharmacy.
3. User must have access to pill cartridges.

In order for an end user to set up for the first time the user needs an Ethernet cable and a router with Internet access. The user needs to follow these steps:

4. Connect the Altera DE2 to the router via the Ethernet cable.
5. Connect the Altera DE2 to the PCB via the ribbon cable. Make sure to connect pins as shown on the layout.
6. Connect the motors to the power source.
7. Connect the Altera DE2 to the power source.

8. Power on the Altera DE2.

After Power on, the Altera DE2 needs a couple of minutes to negotiate an IP address with your router. The system should now be fully operational and its settings should be empty. I.e. no medicines are on the schedule inside the DE2.

In order to schedule a prescription, follow these steps:

9. Obtain a pill cartridge from the pharmacy.
10. From any device that has access to your router's local area network, visit the embedded web server by typing its IP address onto a browser.
11. Click on "Insert Medicode" link.
12. Type the medicode associated with your pill cartridge and click the submit button.

After step twelve, all the information regarding your prescription is available on the website and available to the system.

13. Optionally, the user can change her time preferences, view the schedule, and visit other areas of the website.
14. When alarmed, the user should press the button to dispense the medications.

Please note that the project is not on flash memory, therefore if one wishes to run it, it first needs to be put onto RAM via Eclipse for NIOS II.

Appendix E: Future work

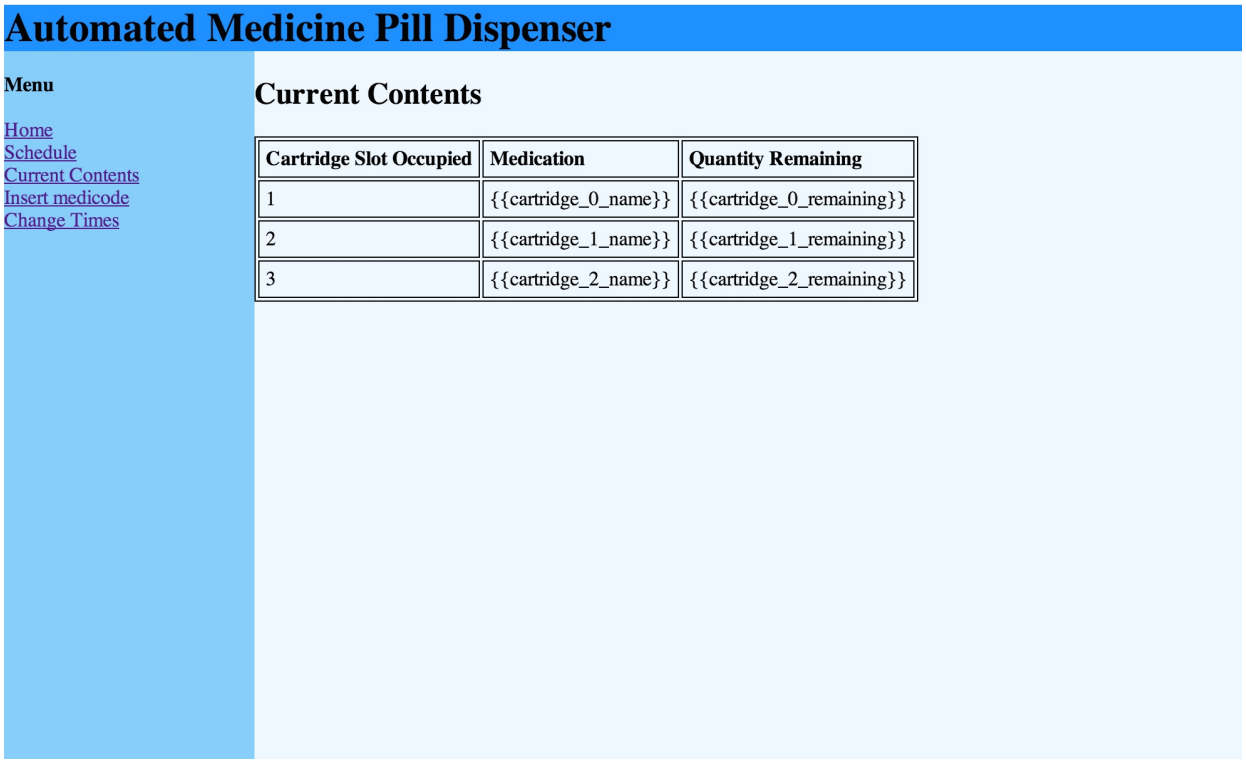
As of April 8, 2014, there are some improvements in mind for the Automated Pill Dispenser. Some of these would require an overall refactoring of the current code. Other improvements, would be easy to implement, but were not implemented due to time constraints.
Here is a list of wanted future work by the current developers:

- The system has no writable filesystem. The system should have a writable filesystem. This is an issue because if power stops being supplied to the board, all of the information about the current state of the pills would be lost. If we had a writable filesystem, backing up this information would be as easy as writing to a file whenever we perform a change in the data structures that hold the cartridge information. Another reason why this would be a nice to have is the web server serves files. However, when we are doing dynamic html, we are handling special cases and instead of serving a file, we are sending a string. It would be nice to parse a template, then write down the HTML that we want into a file and then serve that file instead of handling a strings and doing a lot of memory allocation and deallocation to handle memory. Since dynamic HTML was considered an extra feature from the basic set of features, not a lot of thought was given to the file system until the very end.
- Motors have an “on” position and they toggle between states. This is also an issue because if power is lost, and the system is given power back again, motors will go to the first state. This means that if motors were on the second state, they will move to the first state after power on and will dispense a pill. A great add-on would be DC motors which have a decoder so that the position can be read and then we just jump into the next one whenever the system powers on.
- Writing a web server in C is fun, but it is kind of hard. There are ton of web development frameworks written in Python. It would have been far easier to use Django or Flask to create the website. However, since there is no Python interpreter for the Nios II processor, the web server had to be written exclusively in C, C++. Maybe, writing this on a Raspberry Pi or finding a Python to C compiler would have been nice.
- Using RFID technology is a nice feature as it removes the need to enter codes into the webpage and helps with certain patients with arthritis or other motor control conditions.
- Getting the cartridges to be removable would have also been nice.
- We could parse JSON in C in a friendly way. There are a couple of JSON parsing libraries written for C in C but they come nowhere close to how JSON is parsed in Python. If Python would have been available, the file format that was created for the prescription could have been just a JSON file. This allows for extensibility, code reusability, and would benefit from the fact that it is already an industry standard that is more popular than XML.
- Making the website’s design more user friendly and better looking. Maybe include AJAX so that the values update automatically without to reload the website.

- Implementing a user functionality. Right now the project lacks understanding of multiple users. This means that the pill schedule is set and is assumed for only a single user. However, multiple users could benefit from having this system. User identification can be done via swipe cards, fingerprint recognition, or some other type of key value pair. I personally prefer fingerprint because there is no need to carry additional form of user authentication.
- Using a database. With a database, I would feel safer making and logging changes to the data structure. However, porting BerkeleyDB into the DE2 was something we did not have time for. It would also be easier to extend how many cartridges are being handled by the system.
- Having a bigger LCD for messages. We are using the LCD included in the Altera DE2 to include special instructions when taking the medication. However, we are restricted to 16 characters per line. Special instructions when taking the medication could potentially be longer than this. Therefore, having a bigger display for showing more characters could really improve the look and feel of this system.

Appendix F: HTML Web Pages

[1] Current Contents



Automated Medicine Pill Dispenser

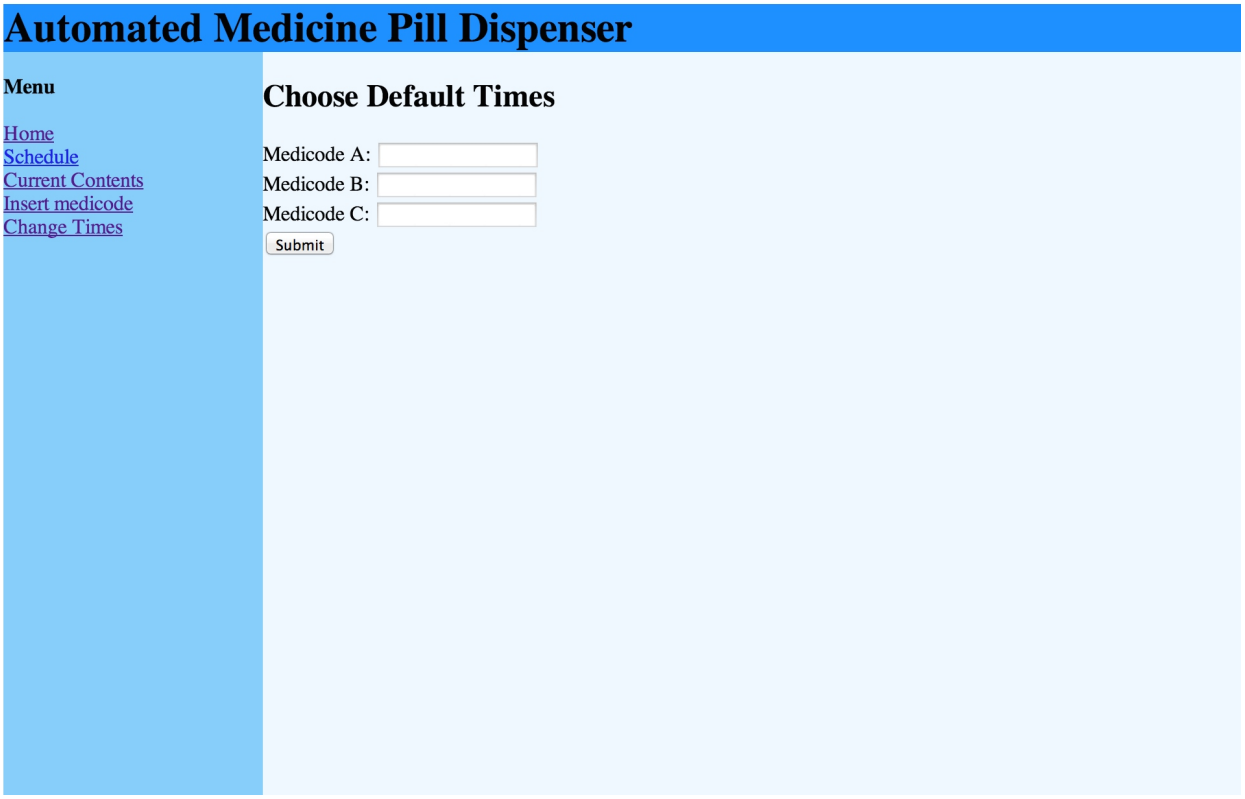
Menu

Home
Schedule
Current Contents
Insert medicode
Change Times

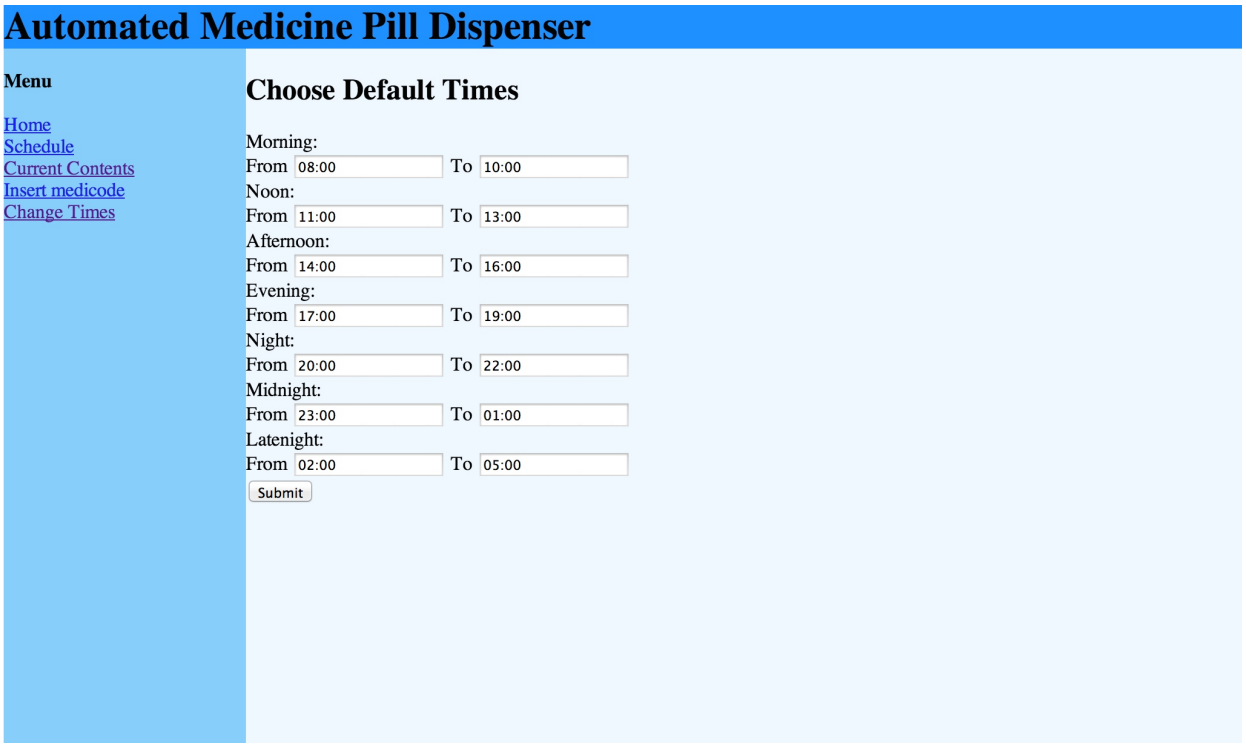
Current Contents

Cartridge Slot Occupied	Medication	Quantity Remaining
1	{{cartridge_0_name}}	{{cartridge_0_remaining}}
2	{{cartridge_1_name}}	{{cartridge_1_remaining}}
3	{{cartridge_2_name}}	{{cartridge_2_remaining}}

[2] Insert Medicode



[3] Change Times



[4] Schedule

Automated Medicine Pill Dispenser

Menu

Home
Schedule
Current Contents
Insert medicode
Change Times

Current Timers

Cartridge Slot Occupied	Medication	Dosage Time	Time Until Next Dispense
1	Medication 1	1/day	04:00 hrs
2	Medication 2	2/day	04:00 hrs
3	Medication 3	1/day at lunch	00:10 minutes

Set Default Times

Appendix G: Full-System Photo