**AUTOMATED PET FEEDER**

**ECET 3220 – DIGITAL III**

**PROJECT REPORT**

**PREPARED BY**

Isrrael Mendoza

Raul Rojas

Nyron Daniel

**X\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**X\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**X\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**SUBMITTED TO**

Scott Tippens

Spring 2016

Kennesaw State University

Table of Contents

LIST OF FIGURES/TABLES ........................................................................................................2

ABSTRACT ……………................................................................................................................3

INTRODUCTION...........................................................................................................................5

BODY

Operating Instructions........................................................................................................6

Hardware Description........................................................................................................7

Software Description........................................................................................................10

CONCLUSION.............................................................................................................................22

PROGRAM CODE …………………............................................................................................23

SCHEMATIC...............................................................................................................................86

BLOCK DIAGRAM......................................................................................................................90

SUBSYSTEM INTERFACE DIAGRAM......................................................................................91

WIRING DIAGRAM....................................................................................................................96

BOARD LAYOUT DIAGRAM.......................................................................................................97

PARTS LIST…..........................................................................................................................102

List of Figures/Tables

* Figures
  + Seven Segment Sample Code..................................................................10
  + LCD Sample Code....................................................................................11
  + Hx711 Sample Code.................................................................................15
  + Keypad Sample Code ..............................................................................16
  + Ultrasonic Sample Code...........................................................................17
  + Servo Sample Code .................................................................................18
  + Buttons, LEDs, and Switches Sample Code.............................................20
  + MCP23017 Sample Code.........................................................................21
  + Hall Effect Sample Code...........................................................................21
  + Power Distribution Schematic ..................................................................86
  + Breakout Board Schematic ......................................................................86
  + Keypad Board Schematic ........................................................................87
  + Back Panel Switch Board Schematic........................................................87
  + 7 Segment/LEDs Board Schematic...........................................................87
  + Ranging Module Schematic......................................................................88
  + LCD/MCP23017 Board Schematic...........................................................88
  + Amplifier Board (Hx711) Board Schematic ..............................................88
  + Hall Effect Sensor Board Schematic.........................................................89
  + Servo Schematic ………………………………………………………………89
  + Automated Pet Feeder Block Diagram......................................................90
  + Automated Pet Feeder Wiring Diagram....................................................96
  + Power Distribution board layout................................................................97
  + Breakout Board layout ……......................................................................97
  + Keypad Board layout ...............................................................................98
  + Back Panel Switch Board layout ..............................................................98
  + 7 Segment/LEDs Board layout……...........................................................99
  + Ranging Module Board layout ..................................................................99
  + LCD/MCP23017 Board layout……..........................................................100
  + Amplifier Board (Hx711) Board layout …...............................................100
  + Hall Effect Sensor Board layout…...........................................................101
* Tables
  + Hx711 Channel Selection..........................................................................15
  + Power Distribution Board..........................................................................91
  + Raspberry Pi.............................................................................................92
  + Breakout Interface Board..........................................................................93
  + Keypad Interface Board............................................................................93
  + Back Panel Switch Interface Board..........................................................94
  + 7 Segment/LEDs Interface Board............................................................94
  + Ranging Module Interface Board..............................................................94
  + LCD/MCP23017 Board.............................................................................95
  + Load Cell Amplifier (Hx711) Interface Board ...........................................95
  + Hall Effect Interface Board........................................................................95

Abstract

The Automated Pet Feeder was built entirely around the use and functionality of the Raspberry Pi 2 Model B+. We were challenged to use the C programming language on a Linux operating system to code and interface an LCD display, 12 LED status indicators, a reset and power switch as well as 2 momentary switches and fixed position switches respectively. We were also required to incorporate an Analog to Digital Converter and use Pulse-Width Modulation for a DC motor into the structure, while keeping it at a maximum physical size of forty by forty by forty centimeters respectively.

In an effort to interface these components we needed to use c programming libraries as well as other physical components. Gordon’s Wiring Pi libraries and header files were pivotal in this project. WiringPi.h, wiringPiI2C.h, stdio.h, stdlib.h, lcd.h, mcp23017.h, are all used to aid in the programming of the automated pet feeder. The interface protocols that were used are: I2C, Parallel Communication Interface (PCI), Serial Communication Interface (SCI), and Boolean Logic. In order to increase the number of GPIO pins that were available we incorporated the use of a MCP23017 GPIO expander.

The Automated Pet Feeder has two LED's in the front that indicate the motor status and the food status as well as a four digit seven segment display just above the two LEDs. The motor status LED lets the user know whether or not the motor has jammed while the food status LED will indicate whether food is below twenty percent. The seven segment display will show the countdown to next pet feeding. This will be dependent on the input of the user, who will use the keypad and an LCD display to indicate when the next feeding should be. The feeder also has three interactive buttons on the front which have integrated LED's that will be used to entice the pet to press the buttons. The system has four switches; a reset switch, a power switch, a pet mode switches and a set mode switch. The power switch turns the system on and off as any user would expect and the reset switch resets the entire system. The pet mode switch allows the user to select between a Dog and Cat mode while the set mode switch allows the user to set the pet feeding schedule at his or her leisure. On the inside of the structure are components such as an ultrasonic ranging module, a hall effect sensor, a servo motor, the raspberry pi, a load cell and an amplifier board. Each component has it vital role, for example the ranging module is used to determine the food level while the Hall Effect sensor works with the motor status LED to indicate when the motor is jammed. The amplifier board and load cell work in unison to measure the amount of food in the bowl as the servo motor is used to dispense food. In essence all these mechanisms are controlled and ran by the Raspberry Pi to create the Automated Pet Feeder.

Introduction

According to the 2015-2016 APPA (American Pet Product Association) Pet Owners Survey, 65% of U.S. households own a pet. Of those households the average person works approximately 8.9 hours a day. This creates the problem of unattended, uncared for and unfed pets for many hours. Man’s best friend becomes man’s forgotten friend while its owner is away.

In light of this, we decided to create the solution to not only this problem but for the problem that arises anytime pets could potentially be left alone for an extended period of time. This problem can come into play when the pet’s owner goes on a vacation which the pet cannot attend. The automated pet feeder is ultimate solution. This product will feed pets on a schedule set by the owner and will also include three buttons that encourage pet interaction. The Automated Pet Feeder will allow the user to select between a cat and dog mode while also allowing the user to see the countdown to the next pet feeding. In essence this product will ensure that no pet goes hungry because of its owners’ busy schedule and will also ensure the pets stay active and engaged.

Body

* **Operating Instructions**

1. Turn on the Power Button
2. Once the power is on the user has two choices: i) Leave the Set Mode switch low. Doing so will tell the program to use the previous saved conditions as its operating parameters. So, this allows the user to make the program run by simply turning it on. ii) Toggle the Set Mode High. After the Switch mode is set high, the LCD will display three conditions for which the user is to enter its parameters. The conditions are as follow: First Dispense of the Day, The Number of Dispenses per Day, and the time between each Dispense.
   1. First Dispense of the Day- this condition tells the system when the first dispense should occur.
   2. Number of Dispenses per Day – Next up is the amount of dispenses per day. This condition tells the pie how many times it is to dispense throughout a period of 24 hours.
   3. Time Between Dispense – This function provides the system with the amount of time that needs to pass before it should dispense again.

Note: The user is to use the keypad to input the parameters of each of the conditions. When setting the time of the first dispense, button labeled A will toggle the "A" or "P" in AM and PM. The buttons labeled 0-9 are used to input the time. Once the desired time is selected, the user is to the button labeled "B" to cycle between each condition.

1. Once the user has inputted the desired parameters for each of the conditions, they are to set the Set Mode Switch back low. Once the Set Mode Switch is back low the system will save and begin running with the new conditions.

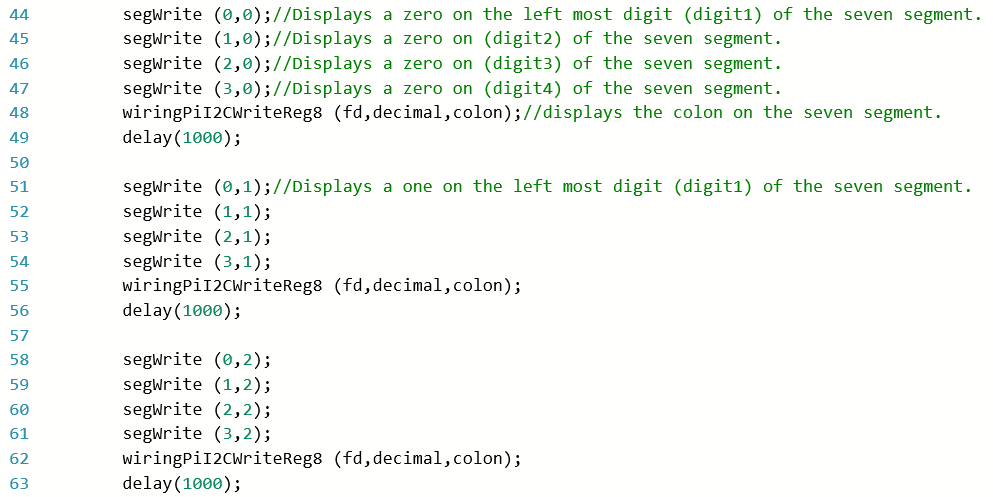
* **Hardware Description**

1. **Raspberry Pi Model B+** *-* The Raspberry Pi 2 B+ consists of 40 GPIO pins but we are limited to the usage of only 26. The GPIO pins that are not being used consist of power, ground, and pre-set pins (ID SC and ID SD). Besides the pre-set pins only set other GPIO pins aside for future purposes. GPIO pins 2 and 3 are the I2C Data and Clock, since we will be using I2C it is crucial to have this pins set aside. The other GPIO pin set aside pin 18. This happens to be the only pin that be used to implement pulse width modulation, which we'll need later for the motor.
2. **4 digit 7 segment (COM-11441)** - This component is a red display that is capable of using TTL, SPI or I2C serial interface. In this case we used the I2C serial interface. The 7 segment can display numbers, most letters and a few characters. It also allows the individual segment control of each digit. It consists of four wires; 1 red power wire connected to 3.3 volts, 1 green ground wire connected to ground, 1 white serial clock (SCL) wire connected to the serial clock GPIO pin of the raspberry pi and 1 blue serial data (SDA) wire connected to the serial data GPIO pin of the raspberry pi. Ensure that the SCL pin and the SDA pin are connected to the I2C SCL pin and SDA pin of the Raspberry Pi respectively.
3. **LCD Display** – This is a 16 x 2 display that allows 32 characters. The text color is white with a blue background. The LCD has 16 pins but only 12 of which are used since we are operating the LCD in 4-bit mode instead of 8-bit mode. Vss, RW, and K are all connected to common ground. Vdd and A are connected to 5V. Vo is connected to a potentiometer for backlight adjusting purposes. RS, RW, and D4-D7 are all connected to the MCP23017.
4. **Load Cell** – Consists of four wires: Red, Black, Green, and White. The Red and black wire go to the “Excitation Voltage”. This is the voltage that is being applied directly to the load cell’s input terminals. Red is the positive excitation (E+) and black is the negative excitation (E-). The other two wire are the Output Wires. Green is the positive output terminal (O+) and white is the negative output terminal (O-). The four wires are connected to the load cell amplifier board (Hx711) being connected to the corresponding labeled pins.
5. **Load Cell Amplifier Board (Hx711)** – the amplifier board has 4 input wires and 4 output wires. The 4 input wires are the excitation wires (red and black) and the output wires (green and white) coming from the Load Cell. The excitation wires are connected to the load cell's input terminal and come from the power supply. The positive lead that come from power is the positive excitation (E+) and the negative lead is the negative excitation (E-). The output wires are the ones that come from the load cell's output terminal and get connected to the Load Cell Amplifier Board (Hx711). The green wire is the positive output signal (O+) and the white wire is the negative output signal (O-).
6. **Keypad** - The keypad used in the project is a common 4x4 keypad. The keypad consists of 4 Rows and 4 Columns. The Rows of the keypad were connected to the MCP23017, while the Columns of the Keypad were connected to directly to the RPi. Also, the Rows have resistors in series for current limitation purposes. The resistor don’t have to be on the Rows they can be vice versa.
7. **Ultrasonic Ranging Module (HC-SR04) -** The ranging module has four pins. It has a Vcc pin which needs to be connected to 5 volts, a ground pin that needs to be connected to ground, a trigger pin that needs to be connected to a GPIO pin set up as an output and an echo pin that needs to be connected to a voltage divider of a 1k ohm resistor in series with a 2.2k ohm resistor that is connected to ground. The wire connection between the two resistors needs to be connected to a GPIO pin that is set up as an input. This voltage divider is set up to protect the pie from any possible high voltages that may return from the echo pin.
8. **Servo Motor** – The motor we choose to use for this project is the MG92B. This motor is a metal-geared servo, which will provide us with sufficient torque just in case the motor were to get stalled. The servo can rotate approximately 180 degrees (90 degrees in each direction). The Servo consists of 3 wires: Vcc, GND, and Signal. The Vcc wire gets connected to a power supply of 5V to 6.6V. The GND wire gets connected to common ground, and the signal pin gets connected to GPIO pin 18.
9. **Buttons, LEDs and Switches** – Varies Buttons, LEDs, and Switches in this project but they were all connected very similarly. They were all connected oriented in such a way so that they could be active low. They were all provided power from the power distribution board, this way the Pie does not have to provide power or ground to any component, thus avoiding any type of heating problems. All of which were also provide a resistor for current limiting purposes.
10. **MCP23017** – Has two 2 bidirectional 8 I/O ports. On port A, we used four pins, specifically GPA4-GPA7. With those 4 pins we connected the rows from the keypad. On Port B we connected the LCD. We ended up needing 6 out of the 8 pins, specifically GPB0-GPB5. The pins that were coming from the LCD were: RS, E, and D4-D7. Besides the I/O ports, we also had to wire the hardware address pins to ground. This gave the device a hardware address of 0. The RESET pin was tied to 3.3V, since we don’t want it to rest at any given time. Power and Ground were tied to Vdd and Vss respectively. And lastly SCL and SDA pins were connected to the pie with pull up resistor in series.
11. **Hall Effect Sensor** – The Hall Effect consists of three Pins: Power, Ground, and Signal. Power is connected to a power supply of 5V. Ground is connected to common ground. Lastly the Signal pin is connected straight to the one of the GPIO pins. However, we did have to tie a pull resistor to the output. Whenever the Hall Effect detects a magnetic field perpendicular to it, it will bring the signal pin low. Essentially it’s an active low switch. The problem we encounter how was that the voltage level on the signal pin was very small when there was not a magnetic field in the presence of the Hall Effect. So, basically the Pie was not able to distinguish whether the Hall Effect was producing a one or zero. So, to resolve this dilemma we had to tie a pull up resistor to the Signal pin. The pull up resistor brings the signal pin to voltage of 3.3V when there is no magnetic field and then down to zero if there was.

* **Software Description**

The Automated Pet Feeder was built around the use of the Raspberry Pi Model B+ with the Linuxoperating system. The entire system was programmed using the C computer programming language with additional libraries for the Raspberry Pi such as WiringPi, stdlib, WiringPiI2C and stdio. These libraries made it easier to interface vital components of the pet feeder without having to do extra coding.

1. **Raspberry Pi** – The Raspberry Pi is the single board computer (SBC) of the system. We use the Raspbian version of Linux in a DOS format to code the entire system. The sole programming Language used is the C language. The GPIO pins of the pi are coded to be high or low and as inputs and/or outputs depending on the way we wanted to use them. Integrated components of the pi such as I2C are programmed, enabled and used to interface multiple components.
2. **Main Code** – The main code of the project was created as a state machine. While the program is idle with no user interaction, the code scans inputs, updates data on lcd and 7 segments, and continuously checks the dispense schedule and compares it to the current time. The next possible state is the Set Mode state in which the user inputs the desired dispense schedule. This state is actually comprised of 3 sub-states or “steps”. The steps include first dispense set, times per day set, and time between set. After all settings have been adjusted and the user is satisfied then the program returns to the idle state. The final state of the program is the dispense state. This state is a temporary state in which the dispense process begins. The dispense process consists of generating random numbers which correspond to a random order in which the front pedals must be pressed to dispense the food. If either the pet successfully presses the order of the buttons correctly or 3 minutes pass, then the food dispenses and the program again returns to the idle state.
3. **4 digit 7 segment Display** - The seven display was interfaced using the I2C bus on the Raspberry Pi. When powered up the display uses the serial clock (SCL) and serial data (SDA) to transmit data. The synchronous protocol of I2C allows the clock to transmit with data. The first step in interfacing the 4 digit 7 segment is enabling I2C on the pi. You do this by adding files to the modules folder. You type in "sudo nano /etc/modules" in the console to enter the modules folder and in that folder you add "i2c –bcm2708" and "i2c –dev" to the kernel. Next you need to load I2C by typing "gpio load i2c" into the console. Ensure that your pi is updated and upgraded. After this, you will have to find the device number on the I2C bus. In order to find this the command "sudo i2cdetect –y 1" will need to be used and should return a number such as 71 on a grid which is the address of that peripheral on the bus. Each device has a unique way of communicating on the I2c bus. In terms of the seven segment to communicate a command such as "sudo i2cset –y 1 0x71 0x7d 0x06" needs to used. By doing typing 0x71 you are first selecting what device you are communicating to on the bus, then 0x7d selects what digit you would like to write to, and finally 0x06 selects the number you would like to write to that digit. There are three header files that need to be included when coding and also a line of code that is needed to set up the I2C GPIO pins. They are “#include <stdio.h>", "#include <wiringPiI2C.h", "#include <wiringPi.h>" and "wiringPiSetupGpio();" respectively. For further clarification on interfacing the seven segment please refer to the code section of this report.

Figure 1 – Seven Segment Sample Code

1. **LCD Display** – The LCD Display uses a Parallel load to communicate between devices. Under normal circumstances the LCD would be operated in 8 Bit Mode (Pins D0-D7) but since were limited to the number of GPIO pins, we were forced to work with 4 Bit Mode (pins D4-D7). Using the WiringPi library, WiringPiDev, we were able to simplify our work. But in order for you to use the WiringPiDev library, you must include the header file <lcd.h> as seen on line 3 in figure 2. The header file contains of the function declarations and without it you won't get far. After you include the corresponding header file, the next essential step is to initialize the LCD. If you try to call any function using the library without having initialized the LCD first the program will not know what you are talking about and freak out. So in order to prevent you simply initialize the LCD before you call upon any functions that are in the WiringPiDev. The LCD initialization function looks like: int lcdInit (rows, cols, bits, rs, strb, d0, d1, d2, d3, d4, d5, d6, d7). Rows and Cols are the rows and columns on the display. Bits is the bit mode you are using, either 4-bit or 8-bit mode. The rs and represent the pin numbers of the displays RS pin and Strobe (E) pin. The parameters d0 – d7 are the pin numbers of the 8 data pins on the LCD. The return value is the "handle" to be used for all subsequent calls to the LCD library when dealing with the LCD, or –1 to indicate a fault.

Note: when using 4-bit mode, only the first 4 are used if you are using 4-bit mode

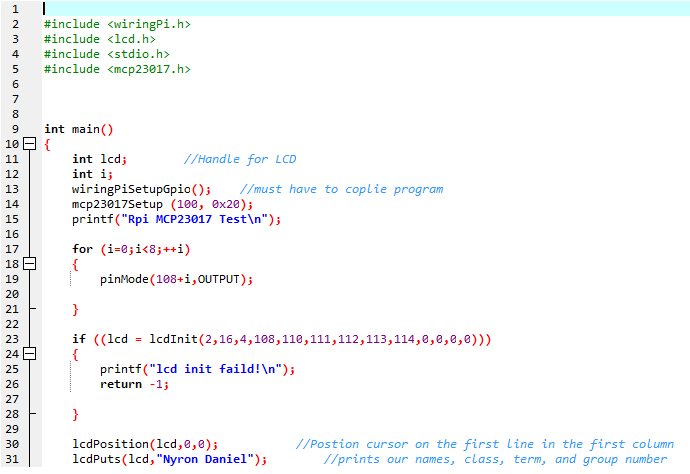
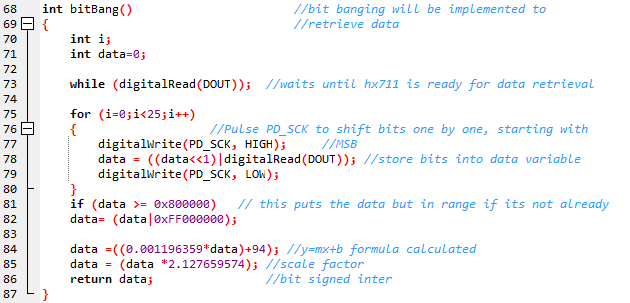


Figure 2 – LCD Sample Code

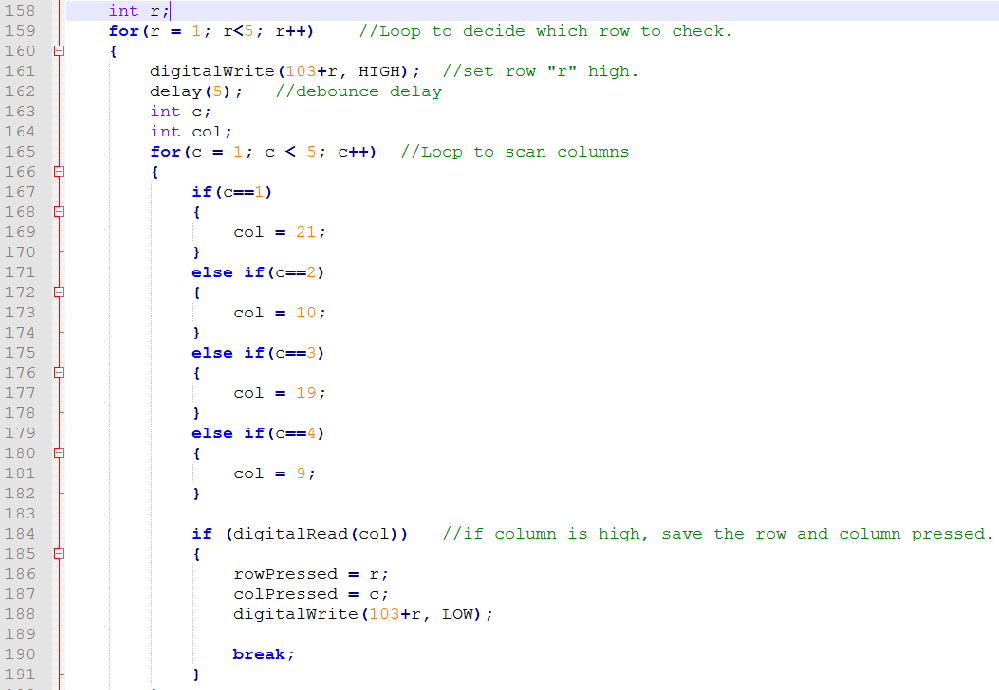
1. **Load Cell Amplifier Board (Hx711)** - the Hx711 is a chip especially designed to interact with Load Cells. The Hx711 uses a Serial Protocol to communicate with other devices. Specifically, it uses a method called Bit Banging to send the data out. The Hx711 has 8 pins, four of which go the to the load cell, 2 of which are power and ground, and the last two are called PD\_SCK and DOUT. When talking to the hx711 for any reason the only pins used are the PD\_SCK and DOUT. Depending on what you want it to do you simply pulse a pin while keeping the other low or high, or you toggle a pin for a certain amount of time. The Chip will automatically toggle DOUT depending on whether it is ready or not for you to retrieve data from it, all you have to do is simply read it. If DOUT is low then you can began retrieving data from it, if not then you must wait till the chip brings it high (line 73). Once DOUT is high we can begin retrieving data, using the Bit Bang Method. Line 75 – 80 are used for the bit banging. As seen in figure 3, the process happens in a For Loop with 25 iterations. Two Reason for the For Loop: i) the hx711 has two input channels to choose from, Channel A and Channel B. Channel A is a programmable channel that can be programmed to have a gain of either 128 or 64. Channel B is a fixed gain channel with a gain of 32. The way you select what channel you want and at what gain (Channel A), depends on how many times you pulse the PD\_SCK pin, refer to the Table 1. So, depending on what gain you want the number of iterations of the For Loop will change respectively. ii) Lines 77-79 is where the actual Bit Banging Method is implemented. First PD\_SCK is set high. After PD\_SCK is set high we implement a digital read function to read the bit and at the same we Logically OR the bit with an integer variable that has been shifted left, equivalent to a times 2 in binary. This allows us to stores the bit that was read with each iteration and store it to a variable. After the bit is stored in the variable PD\_SCK is brought back down and the next iteration begins. After the Bit Banging Process is complete, depending on the value of that was generated if it is below 800000h (MIN) or above 7FFFFFh (MAX) the output will be saturated. So, lines 81 and 82 are used to prevent that. What comes next in lines 84 and 85 is using the data generated from the bit bang method and plugging it into a y=mx+b equation to convert the data into the corresponding grams measurement. The slope, m, was calculated by placing objects with known masses, then using the formula (y2-y1)/(x2-x1) to calculate the corresponding slope, the units for the slope are grams per millivolt. The y intercept, B, was calculating by finding where the scale read zero. Reading the load with nothing on it would return a negative value so, we simply kept adding weight to see at what amount of mass the scale read zero. Once the zero mark was found we had figured out our y-intercept, the units for the y-intercept are grams. The input, x, is the data that is retrieved from the bit bang method. The units for the input are millivolts. The output, y, is the weight that is placed on the load cell in grams. Finally the last part, line 85, we simply scale the output, since we noticed that all of our readings were off by a certain percentage every time.

Table 1 – Hx711 Channel Selection

|  |  |  |
| --- | --- | --- |
| PD\_SCK Pulses | Input Channel | Gain |
| 25 | A | 128 |
| 26 | B | 32 |
| 27 | A | 64 |

Figure 3 – Hx711 Sample Code

1. **Keypad** - The Keypad is essentially made up of 16 buttons, which will notify the pie when a button was pressed. The way the RPi "scans" the keypad to see if a button was indeed presses is by, setting a row high at a time and checking the columns. This process repeats itself 4 times since there are 4 rows on the keypad. Lines 159 through 191 shows the process of how the program checks to see if a button was pressed. A For Loop is use to test the four rows on the Keypad. The iteration of the For Loop correlates to the row on the keypad. So, if the For loop is on its third iteration, then that correlates to the 3 row on the keypad. For each iteration we set the corresponding row high (line 161) and check to see if any of the columns were pressed. If a column key was pressed the program key will then go into the IF statement (line 184) and both the row and column that correspond to the button pressed.

Figure 4 – Keypad Sample Code

1. **Ultrasonic Ranging Module (HC-SR04)** - The ranging module's purpose in this project is to determine the food level of the reservoir. The main things needed to interface this component through software are two GPIO pins, one set up as an input and the other set up as an output. In order to begin communication with the ranging module you first need to set GPIO 4 (Trigger pin - TRIGPIN) high for 20 microseconds. This is done by using a digital write and a delay for example "digitalWrite (TRIGPIN, HIGH);" followed by "delayMicroseconds (20);” This code ultimately initializes the ranging module and makes the HC-SR04 send out an ultrasonic signal. After this signal is send out, the trigger pin is set low and the echo pin will then be used to measure how long it takes to get a signal back. The echo pin will be read while waiting for the signal to return. This starts the count. When the echo pin goes high the count stops and the time difference is converted into distance measured in centimeters. Below is an example of this code.

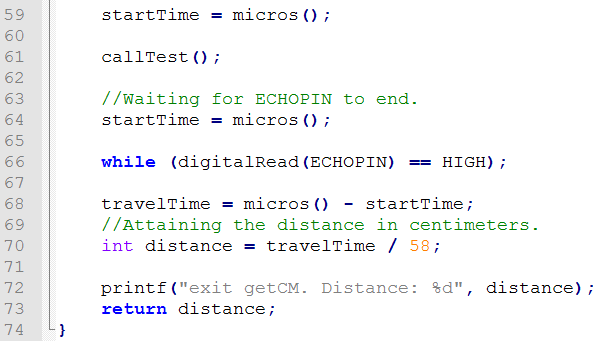


Figure 5 – Ultrasonic Sample Code

The difference in time is divided by 58 in order to change the speed of sound into centimeters. When interfacing the ranging module you have the option of defining the GPIO pins for example "#define TRIGPIN 4". This component also needs WiringPi libraries such as "#include <stdio.h>","#include <wiringPi.h>" and "#include <stdlib.h>" in order to function properly.

1. **Servo Motor** - The Mg92B 180° high torque micro servo operates within a 1 to 2 ms pulse width modulation signal at a frequency of 50Hz (T = 20 ms). To operate the servo, software modulation was used to create the appropriate signal. To incorporate the motor services to the final design of the project, two major functions had to be created. The first function provided the actual modulation for a certain amount of iterations and the other provided a sophisticated method of dispense which could detect whether the motor was stalled and would attempt to unjam on its own.

The first function, called motorMove(), is the core of all the services provided by the servo motor. Using a 10 iteration FOR loop, the GPIO pin for the motor signal was set HIGH for a time value equal to the integer input of the function. To calculate the appropriate LOW time of the modulated signal, the following formula was used: LOW time = 20 ms – High Time. This function can be observed below.

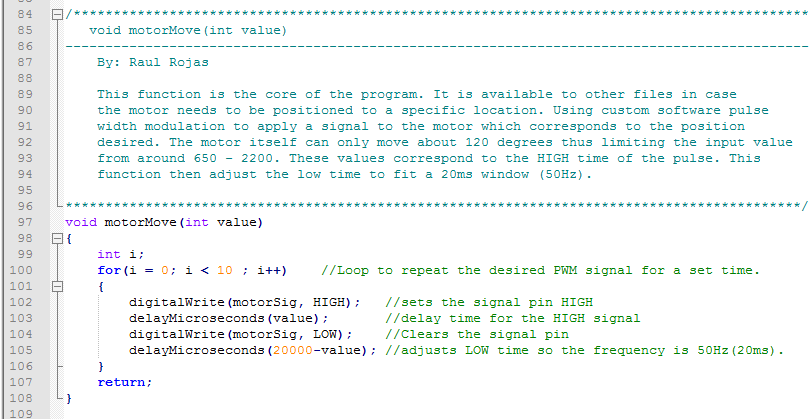
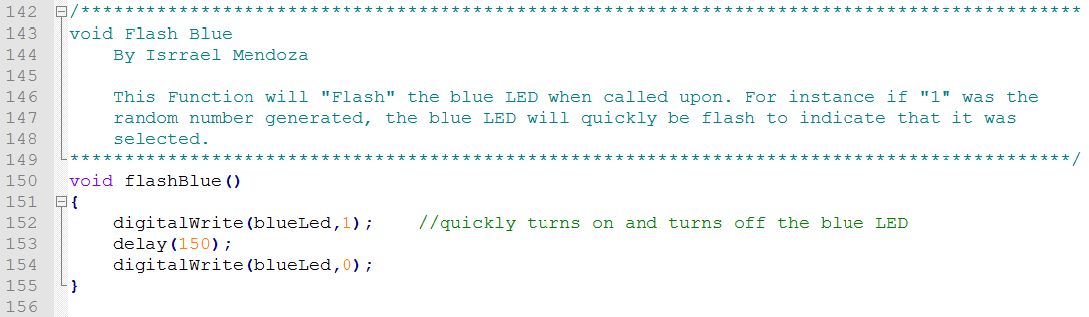


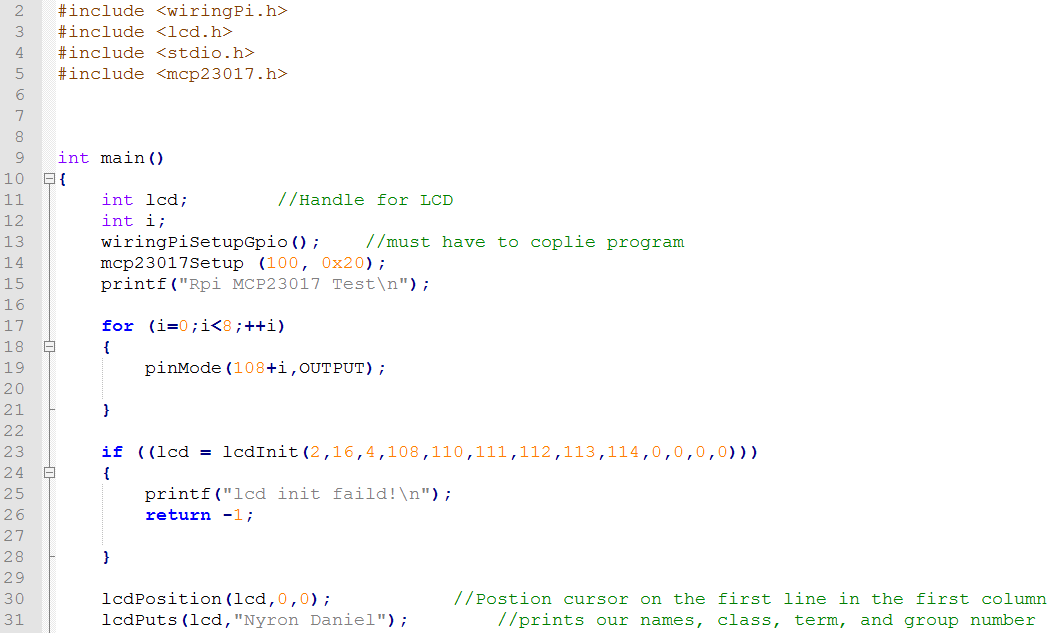
Figure 6: motorMove() Function.

The second function, called motorDispense (), provides a sophisticated method of dispense where it works in conjunction with the Hall Effect sensor to determine the position of the motor and whether the motor is unable to move. To accomplish this, a magnet was attached to the shaft of the dispense mechanism, which is rotated by the motor, and the Hall Effect sensor is positioned in the 90 degree position. So essentially when the motor rotates the "middle" or 90 degree position, the Hall Effect sensor should switch on. Moreover, to determine whether the motor is actually moving, the motor is moved in only 90 degree increments thus allowing us to determine where the magnet/motor is positioned. To determine where the motor is going to move next after a 90 degree turn, it must know where it was previously located, whether that is in the 180 or 0 degree position. To allow correct dispensing of food, a total of 180 degrees needed to be accomplished in two 90 degree maneuvers. As a result a switch structure was used to determine the next position of the motor. To observe this process in more detail, refer the source code in the appendix in the file finalMotorDispense.c.

1. **Buttons, LEDs and Switches** - All of these peripherals do not require any type of protocol, since we are not communication to another device. The reason why we use these peripherals is to provide both the user and the RPi with status information. Depending on what job the peripheral is assigned we will either: read the pin it's connected to or write to the pin it's connected to. Typically we read Buttons and Switches, since they are inputs. While, we typically write to LEDs, since they are used to indicate the status of something. If we refer to line 152 of the LED simple code below we can see how easily it is to write to a pin. Blue Led inside the parentheses indicates the variable name that was assigned to the GPIO pin to that specific LED. The "1" or "0" that follows the variable name indicates whether you want to the pin to be High or Low. To read a pin is the same as a digitalWrite but instead we use a digitalRead and only need to indicate the GPIO pin number we want to read or by indicating its variable name.

Figure 7 – Buttons, LEDs, and Switches Sample Code

1. **MCP23017** - The MCP23017 uses I2C as its interface protocol to communicate with the RPi. Since it uses I2C, this means that it only needs two pins accomplish its data transfers. The two pins are: SDA (serial Data) and SCL (Serial Clock). For this project we only wrote to the MCP23017. We did not read from it because each time we tried to read a pin, it didn’t seem to like it so we work our way around it. Using the WiringPiDev library simplified the process of writing to it using I2C.But before any communication is done you must remember to in the header file <mcp23017.h>. To initialize the MCP23017 the following syntax must be used, mcp23017Setup (int pinBase, int i2cAddress). Pin Base can be any number you wish above 64 and the i2c Address is the device in the I2C bus, which typically happens to be 0x20. You can call mcp23017Setup as many time as needed for each MCP23017 you have in the system, just make sure you use a different pin bas and I2C bus address. After you have initialized the MCP23017 writing to a device is as simple as referring the pin as if it were a GPIO pin. Not to mention you can also use all of the WiringPi commands with any of the MCP23013 pins, after they have been initialized. If we look at line 23 we are setting up the initializing the LCD. As mentioned before, the LCD asks you to input the pin number of that it’s connected to that way it knows what pin to write to. As we can see the numbers in the one hundred range are the pins we are using with the MCP23017 setup. We can simply refer to them as if they were regular GPIO pins. Another example, if you wanted to set pin 108 high, you can simply as follows: digitalWrite (108, high).

Figure 8 - MCP23017 Sample Code

1. **Hall Effect** - The Hall Effect does not require a Protocol since it does talk to any other peripheral, it is essentially just a switch. The output signal of the Hall Effect is held high (3.3V) if there is no magnetic field perpendicular to it. If there is a magnetic field, the Hall Effect will bring the output low (0 V). So, all we need to do with it is read the status of the output pin. The Hall Effect sample code shows a function that was created to read the status of the Hall Effect. The parameter of the digitalRead is the variable that was assigned to the GPIO pin number for the Hall Effect.

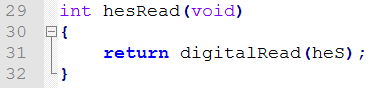


Figure 9 – Hall Effect Sample Code

**Conclusion**

In conclusion, this project was educational, challenging and extremely time consuming all at the same time. It was fun working together and learning about each other while achieving separate and collective goals. Some milestones required us to have patience with one another as well as teach each other, essentially making us grow as individuals. This project was educational not only in the fact that we learnt from each other but also in the countless hours of research we did both individually and collectively on various components of the Automated Pet Feeder. Besides all the collaborative work the challenges started from day one.

The very first hurdle we came across was simply trying to find specific times where and when we could meet up (school / life balance). Our first task was to take a picture together which was simple but the later milestones required us to meet more frequently and to also be available for extended periods of time. This would mean that group members who had other obligations outside of school such as work, mandatory activities and important family matters would have to find some way to balance those times out with this group project time. We solved this problem by first dividing up certain parts of the milestones into three different tasks and then allowing each team member to work on their respective potion at their leisure. We set deadlines before the actual hard deadline and checked in on how we were doing individually through skype. We also used Microsoft One Drive to collaborate on documents we needed to complete jointly.

The second major challenge was coding the system. We all had different coding experience and some of us were more talented in that area than others. In order to simplify the coding process, the most skilled team member in that field taught us the basics and allowed us to build on that knowledge. There were many pitfalls when it came to coding. We got a few hundred thousand errors and had to fight through them by using complex research techniques.

Program Code

1 /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
 2 finalMain.c   
 3 =============================================================================================  
 4 Group Name: Automatic Pet Feeder  
 5 Group Members: Raul Rojas,Isrrael Mendoza,Nyron Daniel  
 6   
 7 The objective of this code is to provide the main funcitonality for the automated pet   
 8 feeder. The program begins by polling inputs and updated live weight and time data   
 9 on the LCD. The user must input a dispense schedule for their pet by activating the   
 10 setMode Switch. Once the Schedule has been set, the program will calculate all possible   
 11 dispense times and will compare those values with the current time. When its time to   
 12 dispense, the program will beggin a button pattern which the pet must mimic to dispense   
 13 food. If the button pattern is not accomplished within 3 minutes, the food will dispense   
 14 anyway.   
 15   
 16 Linked Files: finalSetups.c, finalScanDelay.c, ButtonPattern.c, scanKeypad.c,   
 17 lcd4BitMode.c, sevenSegmentnested.c, finalFileIO.c, finalScheduler.c, finalTime.c,   
 18 finalMotorDispense.c, hx711.c, rangingMod.c   
 19 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
 20   
 21 #include <stdio.h>  
 22 #include <stdlib.h>  
 23 #include <wiringPi.h>  
 24 #include <wiringPiI2C.h>  
 25 #include <lcd.h>  
 26 #include <mcp23017.h>  
 27 #include "finalMain.h"  
 28   
 29 extern int lcd; // handle used for displaying content to the lcd.   
 30 extern int fd; // Handle used for Displaying content to the 4 digit 7 segment display.  
 31 extern int button;  
 32 extern int petModeFlag; //stores the current state of the Pet Mode Switch.  
 33 extern int setModeFlag; //stores the current state of the setMode Swithc.  
 34 extern int petModeStorage;   
 35 extern int dispenseFlag; // Flag indicates when a dispense routine needs to be serviced  
 36   
 37 int timeCount = 0;  
 38   
 39 int main(void)  
 40 {  
 41 printf("Entered Main. \n");  
 42   
 43 setups(); //Function Which performs all setup commands for all peripherals.  
 44   
 45 while(digitalRead(powerSw) == 0) //wait until the power switch is moved to "ON"  
 46 {  
 47 printf("waiting for Power Switch. \n");  
 48 delay(20);  
 49 }  
 50   
 51 while(digitalRead(powerSw)) //When PowerSw low, exit program and execute powerdown seq.  
 52 {  
 53 bootSequence(); // boot sequence tests all displays and LED's  
 54 defaultDisplay(); // function which displays constant words on the LCD and 7 seg.  
 55 while(digitalRead(resetSw) == 0 && digitalRead(powerSw)) //runs while pwr/resetSw Set  
 56 {  
 57 //Main code routines begin here  
 58 //===========================================================================================  
 59 timeCount++; //variable counts time iterations to update the current time/Weight  
 60 smartDelay(20); //Function used to provide both a delay and Scan buttons pressed.  
 61   
 62 if(setModeFlag) //if the setMode Switch is set, go into set mode.   
 63 {  
 64 setMode();  
 65 }  
 66   
 67 dispenseCheck(); //function checks if current time = a scheduled dispense time  
 68   
 69 if (dispenseFlag == 1) //if the above is true, begin dispensing  
 70 {  
 71   
 72 int modeAmount;  
 73 if(petModeFlag) //Checks PetMode switch and sets a dispense limit accordingly  
 74 {  
 75 modeAmount = 120; //more food for Dog mode  
 76 }  
 77 else  
 78 {  
 79 modeAmount = 60; //Less food for Cat Mode  
 80 }  
 81 while(1)  
 82 {  
 83   
 84 motorDispense(); //perform dispense  
 85   
 86 //When the dispense limit is reached, triple check to avoid false posatives  
 87 if(updateWeight() > modeAmount)   
 88 {  
 89 if(updateWeight() > modeAmount)  
 90 {  
 91 if(updateWeight() > modeAmount)  
 92 {  
 93 break; //stop dispensing when the above is satisfied.   
 94 }  
 95 }  
 96 }  
 97   
 98 }  
 99   
100 //vibMove();  
101 //delay(2000);  
102 dispenseFlag = 0; // reset dispense flag.   
103 }  
104 if(timeCount == 15) //update time and weight on lcd every 15\*20ms.  
105 {  
106 updateLcdTime();  
107 foodCheck(); //check the food level   
108 updateWeight();  
109 timeCount = 0; //reset the count;  
110 updateTimer(); //updates the countdown timer on 7 segment displays.  
111 }  
112   
113 //===========================================================================================  
114 }  
115   
116 // wait until reset switch is released.   
117 while(digitalRead(resetSw) == 1 && digitalRead(powerSw))   
118 {  
119 printf("reset switch pressed. \n");  
120 delay(20);  
121 }  
122 lcdClear(lcd); // clear the lcd on exit of program.  
123 printf("reset switch released \n");  
124 }  
125 pwDownSequence(); //perform a power down sequence.   
126 return 0;  
127 }  
128

129 /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
130 void bootSequence(void)  
131 ---------------------------------------------------------------------------------------------  
132   
133 The objective of this function is to perform a boot test for all displays and LED's on   
134 the automated pet feeder, this is to demonstrate the user whether all the indicators   
135 are working properly.  
136 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
137 void bootSequence(void)  
138 {  
139 //Set all LED's High to for testing.   
140 digitalWrite(motorStatLed, LOW);   
141 digitalWrite(foodStatLed, LOW);  
142 digitalWrite(redLed, LOW);  
143 digitalWrite(blueLed, LOW);  
144 digitalWrite(greenLed, LOW);  
145   
146 int i;  
147   
148 for(i=0; i<4; i++)  
149 {  
150 segWrite(i, 0); //Write all zeros to 4 digit 7 seg. for testing.   
151 }  
152   
153 lcdTest(); // display the LCD test, show names, class, group, etc..   
154   
155 //Turn off all LED's and clear all Displays.   
156 digitalWrite(motorStatLed, HIGH);  
157 digitalWrite(foodStatLed, HIGH);  
158 digitalWrite(redLed, HIGH);  
159 digitalWrite(blueLed, HIGH);  
160 digitalWrite(greenLed, HIGH);  
161   
162 segClear();  
163 defaultDisplay();  
164 return;  
165 }  
166   
167 /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
168 void defaultDisplay(void)  
169 ---------------------------------------------------------------------------------------------  
170 This function displays all constants on both the 7 segment and lcd displays.   
171 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
172 void defaultDisplay(void)  
173 {  
174 lcdPosition(lcd, 0, 1);  
175 lcdPrintf(lcd, "Weight(g):");  
176 wiringPiI2CWriteReg8 (fd,decimal,colon);  
177 }  
178   
179 /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
180 void pwDownSequence(void)  
181 ---------------------------------------------------------------------------------------------  
182 This Function ensures all indicators and displays are off before terminating the program.  
183 it also executes a console command to properly shutdown the raspberry Pi.   
184 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
185   
186 void pwDownSequence(void)  
187 {  
188 //Ensure all LED's and displays have been clear and set off.   
189 digitalWrite(motorStatLed, HIGH);  
190 digitalWrite(foodStatLed, HIGH);  
191 digitalWrite(redLed, HIGH);  
192 digitalWrite(blueLed, HIGH);  
193 digitalWrite(greenLed, HIGH);  
194   
195 lcdClear(lcd);  
196   
197 // perform a system shutdown to properly turn off the raspberry pi.   
198 system("sudo shutdown -h now");  
199 return;  
200 }  
201   
202

1 /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
 2 Final demo main Header File (finalMain.h)   
 3 By: Isrrael Mendoza, Raul Rojas, and Nyron Daniels   
 4   
 5 This file will contain all of the function prototypes and constant definitions that will   
 6 be utilized in the main code.   
 7 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
 8 //MISC  
 9 void delay();  
 10 void smartDelay(int ms);  
 11 void bootSequence(void);  
 12 void pwDownSequence(void);  
 13 void setups(void);  
 14 void defaultDisplay(void);  
 15   
 16 //foodPattern setups  
 17 void foodPattern(void);  
 18   
 19 void flashRed();  
 20 void flashBlue();  
 21 void flashGreen();  
 22 void chooseLed();  
 23 void blinkRedLed();  
 24 void blinkBlueLed();  
 25 void blinkGreenLed();  
 26 void waitForFirstPedal();  
 27 void waitForSecondPedal();  
 28 void waitForThirdPedal();  
 29   
 30 //finalScheduler (setMode) prototypes  
 31 void timeSet(void);  
 32 void timesPerDay(void);  
 33 void timeBetweenDisp (void);  
 34 void setMode(void);  
 35 int numberPressed(int buttonPressed);  
 36   
 37 //finalTime prototypes.  
 38 int getTimeH(void);  
 39 int getTimeHMilit(void);  
 40 int getTimeM(void);  
 41 int getTimeS(void);  
 42 void dispenseCheck(void);  
 43 void updateTimer(void);  
 44 //finalFileIO prototypes  
 45 void saveHour(void);  
 46 void saveMin(void);  
 47 void getHour(void);  
 48 void getMin(void);  
 49 void saveAP(void);  
 50 void getAP(void);  
 51 void savePerDay(void);  
 52 void getPerDay(void);  
 53 void saveTimeBetween(void);  
 54 void getTimeBetween(void);  
 55 void updateLcdTime(void);  
 56

57 //Servo Motor Headers  
 58 #define motorSig 18  
 59 #define motorStatLed 14  
 60 #define hes 13  
 61   
 62 void motorMove(int value);  
 63 void motorDispense(void);  
 64 void motorUnstall(void);  
 65 void vibMove(void);  
 66   
 67 //ADC HEADERS  
 68 int getMass();  
 69 int tare();  
 70 int updateWeight(void);  
 71 int avgWeight(void);  
 72   
 73 #define PD\_SCK 7  
 74 #define DOUT 12  
 75   
 76 //RANGING MODULE HEADERS  
 77 int getCM();  
 78 void foodCheck(void);  
 79   
 80   
 81 #define TRIGPIN 4  
 82 #define ECHOPIN 6  
 83   
 84 //LCD HEADERS  
 85 int lcdTest(void);  
 86 #define LCD\_RS 108 //defining constants  
 87 #define LCD\_E 109  
 88 #define LCD\_D4 110  
 89 #define LCD\_D5 111  
 90 #define LCD\_D6 112  
 91 #define LCD\_D7 113  
 92   
 93 //SEVEN SEGMENT HEADERS  
 94 void segWrite(int Digit,int Data);  
 95 void segClear(void);  
 96   
97 #define zero 0x3F  
 98 #define one 0x06  
 99 #define two 0x5B  
100 #define three 0x4F  
101 #define four 0x66  
102 #define five 0x6D  
103 #define six 0x7D  
104 #define seven 0x07  
105 #define eight 0x7F  
106 #define nine 0x67  
107 #define A 0x77  
108 #define B 0x7c  
109 #define C 0x39  
110 #define D 0x5E  
111 #define E 0x79  
112 #define F 0x71  
113 #define decimal 0x77  
114 #define colon 0x10  
115 #define digit1 0x7B  
116 #define digit2 0x7C  
117 #define digit3 0x7D  
118 #define digit4 0x7E  
119   
120 //HALL EFFECT SENSOR HEADERS  
121 int hesRead(void);  
122 #define hes 13  
123

//Keypad Setups  
125 int scanKeypadM(void);  
126 int scanLoop(void);  
127 int scanKeypad(void);  
128   
129 #define r1 104  
130 #define r2 105  
131 #define r3 106  
132 #define r4 107  
133 #define c1 21  
134 #define c2 10  
135 #define c3 19  
136 #define c4 9  
137   
138 //General IO Gpio pin constants  
139 #define motorStatLed 14   
140 #define foodStatLed 15  
141 #define powerSw 24  
142 #define petModeSw 5  
143 #define setModeSw 25  
144 #define firstPedal 17  
145 #define secondPedal 27  
146 #define thirdPedal 22  
147 #define redLed 16  
148 #define blueLed 26  
149 #define greenLed 20  
150 #define resetSw 23  
151   
152

1 /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
 2 setups.c  
 3 ---------------------------------------------------------------------------------------------  
 4 by: Raul Rojas  
 5   
 6 Function: void setups(void)  
 7 This funciton contains all necessary lines of code which initialize wiringPi libraries  
 8 as well as any other necessry GPIO set ups for peripherals. Setups are orginized by  
 9 peripheral.   
 10 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
 11   
12 #include <stdio.h>  
 13 #include <wiringPi.h>  
 14 #include <wiringPiI2C.h>  
 15 #include <stdlib.h>  
 16 #include <lcd.h>  
 17 #include <mcp23017.h>  
 18 #include "finalMain.h"  
 19   
 20 int lcd; //declaring LCD handle variable.   
 21 int fd;  
 22   
 23 void setups(void)  
 24 {  
 25   
 26 wiringPiSetupGpio();  
 27   
 28 //General IO setups  
 29 pinMode(motorStatLed, OUTPUT);  
 30 pinMode(foodStatLed, OUTPUT);  
 31 pinMode(redLed, OUTPUT);  
 32 pinMode(blueLed, OUTPUT);  
 33 pinMode(greenLed, OUTPUT);  
 34

35 digitalWrite(motorStatLed, HIGH);  
 36 digitalWrite(foodStatLed, HIGH);  
 37 digitalWrite(redLed, HIGH);  
 38 digitalWrite(blueLed, HIGH);  
 39 digitalWrite(greenLed, HIGH);  
 40   
 41 pinMode(resetSw, INPUT);  
 42 pinMode(powerSw, INPUT);  
 43 pinMode(petModeSw, INPUT);  
 44 pinMode(setModeSw, INPUT);  
 45 pinMode(firstPedal, INPUT);  
 46 pinMode(secondPedal, INPUT);  
 47 pinMode(thirdPedal, INPUT);  
 48   
 49 //Motor Setups  
 50 pinMode(motorSig, OUTPUT);  
 51 pinMode(motorStatLed, OUTPUT);  
 52 pinMode(hes, INPUT);  
 53   
 54 //SEVEN SEGMENT SETUP  
 55 //int fd;  
 56 if((fd=wiringPiI2CSetup(0x71)))//Intializing seven segment on I2C bus and handle   
 57 {}  
 58   
 59 //LCD SETUP  
 60 int i;  
 61 //wiringPiSetupGpio(); //must have to coplie program  
 62 mcp23017Setup (100, 0x20); //Gpio expander initialization.  
 63 //printf("Rpi MCP23017 Test\n");  
 64   
 65 for (i=0;i<6;++i) //loop to set pin modes for LCD. (on GPIO expander pins)  
 66 {  
 67 pinMode(108+i,OUTPUT);  
 68   
 69 }  
 70   
 71 //LCD initialization and handle set up  
 72   
 73 lcd = lcdInit(2,16,4,LCD\_RS, LCD\_E, LCD\_D4, LCD\_D5, LCD\_D6, LCD\_D7,0,0,0,0);  
 74 //{  
 75 //printf("lcd init failed!\n");  
 76 //return -1;  
 77 //}  
 78   
 79 //KEYPAD SETUP  
 80 for (i = 0; i < 4; i++) //setting up pin modes.  
 81 {  
 82 pinMode (104+i, OUTPUT); //all rows = outputs, all columns = inputs.  
 83 }  
 84 pinMode(c1, INPUT); //Column pin assignments   
 85 pinMode(c2, INPUT);  
 86 pinMode(c3, INPUT);  
 87 pinMode(c4, INPUT);  
 88   
 89 //ADC SETUP HX711  
 90 pinMode (DOUT,INPUT); //setting up pin modes  
 91 pinMode (PD\_SCK, OUTPUT);  
 92 digitalWrite (PD\_SCK, LOW); //setting hx711 in normal mode  
 93   
 94 //RANGIN MODULE SETUPS  
 95 //wiringPiSetupGpio();  
 96 pinMode(TRIGPIN, OUTPUT);  
 97 pinMode(ECHOPIN, INPUT);  
 98   
 99 //TRIGPIN must start low in order to get a clear signal.  
100 digitalWrite(TRIGPIN, LOW);  
101 // delay(30);  
102   
103 //HES SETUP  
104 pinMode(hes, INPUT);  
105   
106   
107 }

1 #include <stdio.h>  
 2 #include <stdlib.h>  
 3 #include <wiringPi.h>  
 4 #include <wiringPiI2C.h>  
 5 #include <lcd.h>  
 6 #include <mcp23017.h>  
 7 #include "finalMain.h"  
 8   
 9 int button;  
 10 int petModeFlag;  
 11 int setModeFlag;  
 12   
 13 int firstPedalFlag;  
 14 int secondPedalFlag;  
 15 int thirdPedalFlag;  
 16 int dispenseFlag;  
 17   
 18 /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
 19 void smartDelay(int ms)  
 20 ---------------------------------------------------------------------------------------------  
 21 The purpose of this funciton is to provide a substitute to the delay() function. The   
 22 smartDelay() will delay for the desired time with an accuracy of +-10ms while also   
 23 polling to determine the state of all inputs on the automated pet feeder. The   
 24 smartDelay() does not do anything in itself as it uses global variables/Flags to allow   
 25 the calling function to service the requests and to avoid generating a large call stack.  
 26 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
 27 void smartDelay(int ms)  
 28 {  
 29 if (ms < 20) //making sure that the while loop executes at least once.  
 30 {  
 31 ms = 20;  
 32 }  
 33   
 34 int i; //calculating the amount of times to preform the while loop.  
 35   
 36 //int stat = 0;  
 37 int timeCheck = 0; //used for testing how much time has been delayed in total.  
 38   
 39 for(i = ms/20; i >= 0; i--)  
 40 {  
 41 //Priority 1: Power and reset Switch/Button.  
 42 if(digitalRead(powerSw) == 0 || digitalRead(resetSw) == 1 )  
 43 {  
 44 break;  
 45 }  
 46   
 47 //Priority 2: Set mode & Pet mode Switches.  
 48 if(digitalRead(setModeSw))  
 49 {  
 50 setModeFlag = 1;  
 51 }  
 52 else  
 53 {  
 54 setModeFlag = 0;  
 55 }  
 56

57 if(digitalRead(petModeSw))  
 58 {  
 59 petModeFlag = 1;  
 60 }  
 61 else  
 62 {  
 63 petModeFlag = 0;  
 64 }  
 65   
 66 //Priority 3: keypad scan/delay  
 67 while((button = scanKeypad()) > 0)  
 68 {  
 69 if (button == 16)  
 70 {  
 71 dispenseFlag = 1;  
 72 }  
 73   
 74 if(button > 12)  
 75 {  
 76 //printf("button pressed: %d \n", button);  
 77 }  
 78 else if(button > 8)  
 79 {  
 80 delay(5);  
 81 }  
 82 else if(button > 4)  
 83 {  
 84 delay(10);  
 85 }  
 86 else if(button > 0)  
 87 {  
 88 delay(15);  
 89 }  
 90 timeCheck += 20;  
 91   
 92 if(timeCheck >= ms || i <= 0)  
 93 {  
 94 i = 0;  
 95 //printf("breaking.. \n");  
 96 break;  
 97 }  
 98 else  
 99 {  
100 //printf("smartDelay loop iteration: %d \n", i);  
101 //printf("Time delayed: %d \n", timeCheck);  
102 i--;  
103 }  
104 }  
105 //printf("smartDelay loop iteration: %d \n", i);  
106 //printf("Time delayed: %d \n", timeCheck);  
107   
108   
109 }  
110   
111 //priority 4: front pedals  
112 if (digitalRead(firstPedal))  
113 {  
114 while(digitalRead(firstPedal))  
115 {  
116 firstPedalFlag = 1;  
117 }  
118 printf("firstPedalFlag: %d\n", firstPedalFlag);  
119 //return;  
120 }  
121 else   
122 {  
123 firstPedalFlag = 0;   
124 }  
125   
126 if (digitalRead(secondPedal))  
127 {  
128 while(digitalRead(secondPedal))  
129 {  
130 secondPedalFlag = 1;  
131 }  
132 //return;  
133 }  
134 else   
135 {  
136 secondPedalFlag = 0;   
137 }  
138   
139 if (digitalRead(thirdPedal))  
140 {  
141 while(digitalRead(thirdPedal))  
142 {  
143 thirdPedalFlag = 1;  
144 }  
145 //return;  
146 }  
147 else   
148 {  
149 thirdPedalFlag = 0;   
150 }  
151 //printf("secondPedalFlag: %d\n", secondPedalFlag);  
152 //printf("thirdPedalFlag: %d\n", thirdPedalFlag);  
153 return;  
154 }

1 #include <stdio.h>  
 2 #include <stdlib.h>  
 3 #include <time.h>   
 4 #include "finalMain.h"  
 5 #include <wiringPi.h>  
 6   
 7 extern int dispenseFlag;  
 8   
 9 extern int firstPedalFlag;  
 10 extern int secondPedalFlag;  
 11 extern int thirdPedalFlag;  
 12   
 13 int coolCount;   
 14 int i;   
 15 int n;  
 16 int number;  
 17 int firstNumber=-1;  
 18 int secondNumber=-1;  
 19 int thirdNumber=-1;  
 20 int startTime;  
 21 int timer;  
 22   
 23 time\_t t;  
 24 /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
 25 void foodPattern(void)  
 26 ---------------------------------------------------------------------------------------------  
 27 This funciton begins the dispense process. It generates 3 random numbers that will   
 28 correspond to each pedal and associated LED and the pet must press the pedals in order  
 29 to dispense food. If the pet is unsuccessful to do this in 3 minutes, then the food will   
 30 dispense anyway. The pattern will first quickly blink in order (no 2 random numbers will   
 31 be the same.) and then will blink the first pedal in a slower pace and wait for it   
 32 be pressed.   
 33 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
 34 void foodPattern(void)  
 35 {  
 36   
 37 digitalWrite (redLed, HIGH);  
 38 digitalWrite (blueLed, HIGH);  
 39 digitalWrite (greenLed, HIGH);  
 40   
 41   
 42   
 43 n=3; //chooses the amount of buttons that must be pressed before food is dispensed  
 44   
 45 srand((unsigned) time(&t)); //intializes random number generator  
 46   
 47 for (i=0; i < n; i++ ) //3 random numbers between 0 and 2 will be generated  
 48 {   
 49 if (i>0)  
 50 {   
 51 while (number == firstNumber || number == secondNumber || number == thirdNumber)  
 52 {  
 53 number = rand()%3; //store random number in variable   
 54 }   
 55 }  
 56 else   
 57 {  
 58 number = rand()%3;  
 59 }   
 60 printf("Numer: %d\n", number);  
 61 if (i==0)   
 62 {  
 63 firstNumber = number;   
 64 chooseLed(); //stores the random number generated to a variable out side  
 65 } //the loops that generates the number. This way, the random  
 66 else if (i==1) // number is saved before it is overwritten by the next  
 67 {  
 68 secondNumber = number;  
 69 chooseLed();  
 70 }   
 71 else if (i==2)  
 72 {  
 73 thirdNumber = number;  
 74 chooseLed();  
 75 }  
 76   
 77 }  
 78   
 79 printf("First Number\n");  
 80 switch (firstNumber)  
 81 {  
 82 case 0:  
 83 waitForFirstPedal(); //depending on the first random that was generated   
 84 break; // the program will select the corresponding case  
 85 case 1:   
 86 waitForSecondPedal();  
 87 break;  
 88 case 2:  
 89 waitForThirdPedal();  
 90 break;  
 91 }  
 92   
 93 printf("secondNumber\n");   
 94 switch (secondNumber)  
 95 {  
 96 case 0:  
 97 waitForFirstPedal();  
 98 break; //the same proccess happens to 2 more times, since  
 99 case 1: //there is 3 different random number generated, which  
100 waitForSecondPedal(); //correlates to three dog pedals that must be pressed   
101 break; //before food is despensed  
102 case 2:  
103 waitForThirdPedal();  
104 break;  
105 }  
106   
107 printf("ThirdNumber\n");   
108 switch (thirdNumber)  
109 {  
110 case 0:  
111 waitForFirstPedal();  
112 break;  
113 case 1:  
114 waitForSecondPedal(); //once the third case is reached, the program will set the   
115 break; //dispense flag high wether the last dog pedal was pressed   
116 case 2: //or if the 1 minute timeout was reached  
117 waitForThirdPedal();  
118 break;   
119 }  
120   
121 printf("3 Minute Timeout\n");  
122 printf("Despense FOOD!\n"); //returns to main  
123 return;  
124 }   
125   
126   
127 /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
128 void Flash Red   
129 By Isrrael Mendoza  
130   
131 This Function will "Flash" the red LED when called upon. For instance if "0" was the   
132 random number generated, the red LED will quickly be flash to indicate that it was  
133 selected.  
134 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
135 void flashRed()  
136 {  
137 digitalWrite(redLed,0); //quickly turns on and turns off the red LED  
138 delay(250);  
139 digitalWrite(redLed,1);  
140   
141 }  
142   
143 /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
144 void Flash Blue   
145 By Isrrael Mendoza  
146   
147 This Function will "Flash" the blue LED when called upon. For instance if "1" was the   
148 random number generated, the blue LED will quickly be flash to indicate that it was  
149 selected.  
150 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
151 void flashBlue()  
152 {  
153 digitalWrite(blueLed,0); //quickly turns on and turns off the blue LED  
154 delay(250);  
155 digitalWrite(blueLed,1);  
156 }  
157   
158 /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
159 void Flash Green   
160 By Isrrael Mendoza  
161   
162 This Function will "Flash" the green LED when called upon. For instance if "2" was the   
163 random number generated, the green LED will quickly be flash to indicate that it was  
164 selected.  
165 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
166 void flashGreen()  
167 {  
168 digitalWrite(greenLed,0); //quickly turns on and turns off the green LED  
169 delay(250);  
170 digitalWrite(greenLed,1);  
171 }  
172

173 /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
174 Blink Red LED  
175 By Isrrael Mendoza  
176   
177 will simply "blink" the Red LED when call upon. Similar to the "Flash" Red function   
178 but the difference between them is that this function turn on and turn off the LED at  
179 slower rate while it scans the corresponding pedal.If the pedal was pressed it'll   
180 return from the function  
181 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
182 void blinkRedLed()  
183 {  
184 printf("Blink Red LED\n");  
185 int i;   
186 digitalWrite (redLed, 1);  
187 for (i=0;i<25;i++) //Loop that will consisting blink the LED and simutanlously  
188 { //check the firstPedalFlag to see if it was pressed  
189 if (firstPedalFlag)  
190 { //this loop blinks the LED Every 0.5 seconds   
191 break;  
192 }  
193 smartDelay (20); //smart delay is used to check the status of the pedal  
194 if (firstPedalFlag)  
195 {  
196 break;  
197 }  
198 }  
199   
200 if (firstPedalFlag)  
201 {  
202 return;  
203 }  
204   
205 digitalWrite (redLed,0);  
206 for (i=0;i<25;i++)  
207 {  
208 if(firstPedalFlag)  
209 {  
210 break;  
211 }  
212 smartDelay (20);  
213 if(firstPedalFlag)  
214 {  
215 break;  
216 }  
217 }  
218   
219 digitalWrite (redLed,1);  
220   
221 }   
222   
223 /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
224 Blink Blue LED  
225 By Isrrael Mendoza  
226   
227 will simply blink the Blue LED when call upon. Similar to the "Flash" Blue function   
228 but the difference between them is that this function turn on and turn off the LED at  
229 slower rate while it scans the corresponding pedal.If the pedal was pressed it'll   
230 return from the function  
231 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
232 void blinkBlueLed()  
233 {  
234 printf("Blink Blue LED\n");  
235 int i;  
236 digitalWrite (blueLed, 1);  
237 for (i=0;i<25;i++) //Loop that will consisting blink the LED and simutanlously  
238 { //check the firstPedalFlag to see if it was pressed  
239 if (secondPedalFlag)  
240 {  
241 break; //this loop blinks the LED Every 0.5 seconds  
242 }  
243 smartDelay (20); //smart delay is used to check the status of the pedal  
244 if (secondPedalFlag)  
245 {  
246 break;  
247 }  
248 }  
249   
250 if (firstPedalFlag)  
251 {  
252 return;  
253 }  
254   
255 digitalWrite (blueLed,0);  
256 for (i=0;i<25;i++)  
257 {  
258 if(secondPedalFlag)  
259 {  
260 break;  
261 }  
262 smartDelay (20);  
263 if(secondPedalFlag)  
264 {  
265 break;  
266 }  
267 }  
268   
269 digitalWrite (blueLed,1);  
270   
271 }   
272   
273 /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
274 Blink Green LED  
275 By Isrrael Mendoza  
276   
277 will simply blink the Green LED when call upon. Similar to the "Flash" Blue function   
278 but the difference between them is that this function turn on and turn off the LED at  
279 slower rate while it scans the corresponding pedal.If the pedal was pressed it'll   
280 return from the function  
281 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
282 void blinkGreenLed()  
283 {  
284 printf("Blink Green LED\n");  
285 int i;  
286 digitalWrite (greenLed, 1);  
287 for (i=0;i<25;i++) //Loop that will consisting blink the LED and simutanlously  
288 { //check the firstPedalFlag to see if it was pressed  
289 if (thirdPedalFlag)  
290 {  
291 break; //this loop blinks the LED Every 0.5 seconds  
292 }  
293 smartDelay (20); //smart delay is used to check the status of the pedal  
294 if (thirdPedalFlag)  
295 {  
296 break;  
297 }  
298 }  
299   
300 if (firstPedalFlag)  
301 {  
302 return;  
303 }  
304   
305 digitalWrite (greenLed,0);   
306 for (i=0;i<25;i++)  
307 {

308 if (thirdPedalFlag)  
309 {  
310 break;  
311 }  
312 smartDelay (20);  
313 if (thirdPedalFlag)  
314 {  
315 break;  
316 }  
317 }  
318 digitalWrite (greenLed,1);  
319 }   
320   
321 /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
322 Choose LED   
323 By Isrrael Mendoza  
324   
325 This function will choosing the corresponding LED to blink according to what number was   
326 That was generated. If a zero was generated the corresponding LED is the RED LED. If a  
327 one was generated the corresponding LED is the Blue LED. And lastly, if a Two was  
328 generated the corresponding LED is the Green  
329 random number   
330 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
331 void chooseLed()  
332 {  
333 switch (number)  
334 {  
335 case 0:  
336 flashRed();  
337 break;  
338 case 1:  
339 flashBlue(); //depending which number was generated, the corresponding  
340 break; //LED will be flashed, thus indicating the pattern  
341 case 2:  
342 flashGreen();  
343 break;  
344 }  
345 }   
346   
347 /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
348 Wait For First Pedal  
349 by Isrrael Mendoza  
350   
351 This function waits for the First Dog Pedal (Red LED) to be pressed. If it does not get   
352 pressed within 1 miutes the program will timeout and automatically go onto the next LED   
353 that needs to be pressed before dispense  
354 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
355   
356 void waitForFirstPedal()  
357 {  
358 int t;  
359 coolCount ++;  
360 for (t=0;t<29;t++)  
361 {  
362 printf("%d\n",t);   
363 if (firstPedalFlag) //loop waits for a minute, while consistently  
364 { //checking the first pedal status and blinking  
365 break; //the corresponding LED  
366 }  
367 blinkRedLed();  
368 if (firstPedalFlag)   
369 {  
370 break;  
371 }  
372 smartDelay(20);  
373 if (firstPedalFlag)   
374 {  
375 break;  
376 }  
377   
378 }  
379 if (firstPedalFlag) //if first pedal was pressed then and if it  
380 { //happens to be the third case then it will   
381 printf("First Pedal Pressed\n"); //set the dispense flag high, otherwise it   
382 if (coolCount == 3) // the program will just set pedalFlag low  
383 {  
384 dispenseFlag =1;  
385 printf("First Pedal, High, Dispense Flag Set\n");  
386 }   
387 t = 0;  
388 firstPedalFlag=0;  
389 }   
390 else   
391 {  
392 printf("1 Mintue Timeout Reached!\n"); //if first pedal was not pressed the 1  
393 if (coolCount == 3) //minute time out was reached. If it   
394 { //happens to be the third case then it  
395 dispenseFlag = 1; //will set the dispense flag high   
396 printf("First Pedal, Timeout, Dispense Flag Set\n");   
397 }   
398 t = 0;  
399 }  
400 }   
401   
402 /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
403 Wait For Second Pedal  
404 by Isrrael Mendoza  
405   
406 This function waits for the Second Dog Pedal (Blue LED) to be pressed. If it does not get   
407 pressed within 1 miutes the program will timeout and automatically go onto the next LED   
408 that needs to be pressed before dispense  
409 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
410 void waitForSecondPedal()  
411 {  
412 coolCount ++;  
413 int t;  
414 for (t=0;t<29;t++)  
415 {  
416 printf("%d\n",t);  
417 if (secondPedalFlag) //loop waits for a minute, while consistently  
418 { //checking the first pedal status and blinking  
419 break; //the corresponding LED  
420 }  
421 blinkBlueLed();  
422 if (secondPedalFlag)   
423 {  
424 break;  
425 }  
426 smartDelay(20);  
427 if (secondPedalFlag)   
428 {  
429 break;  
430 }  
431 }  
432 if (secondPedalFlag) //if first pedal was pressed then and if it  
433 { //happens to be the third case then it will   
434 printf("Second Pedal Pressed\n"); //set the dispense flag high, otherwise it  
435 if (coolCount == 3) // the program will just set pedalFlag low  
436 {  
437 dispenseFlag=1;  
438 printf("Second Pedal, High, Dispense Flag Set\n");  
439 }   
440 t = 0;  
441 secondPedalFlag=0;  
442 }   
443 else   
444 {  
445 printf("1 Mintue Timeout Reached!\n"); //if first pedal was not pressed the 1  
446 if (coolCount == 3) //minute time out was reached. If it   
447 { //happens to be the third case then it  
448 dispenseFlag=1; //will set the dispense flag high   
449 printf("Second Pedal, Timeout, Dispense Flag Set\n");  
450 }   
451 t = 0;  
452 }  
453 }   
454   
455 /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
456 Wait For Second Pedal  
457 by Isrrael Mendoza  
458   
459 This function waits for the Second Dog Pedal (Blue LED) to be pressed. If it does not get   
460 pressed within 1 miutes the program will timeout and automatically go onto the next LED   
461 that needs to be pressed before dispense  
462 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
463   
464 void waitForThirdPedal()  
465 {  
466 coolCount ++;  
467 int t;  
468 for (t=0;t<29;t++)  
469 {  
470 printf("%d\n",t);  
471 if (thirdPedalFlag) //loop waits for a minute, while consistently  
472 { //checking the first pedal status and blinking  
473 break; //the corresponding LED  
474 }  
475 blinkGreenLed();  
476 if (thirdPedalFlag)   
477 {  
478 break;  
479 }  
480 smartDelay(20);  
481 if (thirdPedalFlag)   
482 {  
483 break;  
484 }  
485 }   
486 if (thirdPedalFlag) //if first pedal was pressed then and if it  
487 { //happens to be the third case then it will  
488 printf("Third Pedal Pressed\n"); //set the dispense flag high, otherwise it  
489 if (coolCount == 3) // the program will just set pedalFlag low  
490 {  
491 dispenseFlag= 1;  
492 printf("Third Pedal, High, Dispense Flag Set\n");  
493 }   
494 t = 0;  
495 thirdPedalFlag=0;  
496 }   
497 else   
498 {  
499 printf("1 Mintue Timeout Reached!\n"); //if first pedal was not pressed the 1  
500 if (coolCount == 3) //minute time out was reached. If it   
501 { //happens to be the third case then it  
502 dispenseFlag=1; //will set the dispense flag high   
503 printf("Third Pedal, Timeout, Dispense Flag Set\n");  
504 }   
505 t = 0;  
506 }  
507 }

1 /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
 2 scanKeypad.c  
 3 ---------------------------------------------------------------------------------------------  
 4 Objective: This program scans the keypad for any button pressed.  
 5 Things of note:  
 6 GPIO assignments:  
 7 ROWS:  
 8 r1 = 104  
 9 r2 = 105  
 10 r3 = 106  
 11 r4 = 107  
 12   
 13 Columns:  
 14 c1 = 21  
 15 c2 = 10  
 16 c3 = 19  
 17 c4 = 9  
 18   
 19 integer returned from this function corresponds to the number as shown below  
 20 within the button. If no buttn is pressed the function returns 0.  
 21   
 22 +----+----+----+----+  
 23 | 1 | 2 | 3 | 4 | row1  
 24 +----+----+----+----+  
 25 | 5 | 6 | 7 | 8 | row2  
 26 +----+----|----+----+  
 27 | 9 | 10 | 11 | 12 | row3  
 28 +----+----+----+----+  
 29 | 13 | 14 | 15 | 16 | row4  
 30 +----+----+----+----+  
 31 col1 col2 col3 col4  
 32   
 33 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
 34 #include <stdio.h>  
 35 #include <wiringPi.h>  
 36 #include <mcp23017.h>  
 37 #include "finalMain.h"  
 38   
 39 /\*  
 40 int r1 = 104; //setting up constant variable to represent gpio pin numbers.  
 41 int r2 = 105;  
 42 int r3 = 106;  
 43 int r4 = 107;  
 44   
 45 int c1 = 21;  
 46 int c2 = 10;  
 47 int c3 = 19;  
 48 int c4 = 9;  
 49 \*/  
 50   
 51 int rowPressed =0; //set these as global so that the main and the scan loop can access/modify  
 52 int colPressed =0;  
 53   
 54 //extern int lcd;  
 55   
 56 //void delay();  
 57 //int scanKeypadM(void);  
 58 //int scanLoop(void);  
 59 //int scanKeypad(void);  
 60 //void segWrite(int Digit,int Data);  
 61 //void segClear(void);  
 62   
 63   
 64 int scanKeypad(void)  
 65 {  
 66 //wiringPiSetupGpio(); //required wiringPi library set up function  
 67 //mcp23017Setup(100, 0x20); //required mcp23017 extender set up function.  
 68 /\*  
 69 int i;  
 70 for (i = 0; i < 4; i++) //setting up pin modes.  
 71 {  
 72 //delay(20);  
 73 //if(i<4)  
 74 //{  
 75 pinMode (104+i, OUTPUT); //all rows = outputs, all columns = inputs.  
 76 //}  
 77 //else  
 78 //{  
 79 // pinMode (100+i, INPUT);  
 80 //}  
 81 }  
 82 pinMode(9, INPUT);  
 83 pinMode(10, INPUT);  
 84 pinMode(19, INPUT);  
 85 pinMode(21, INPUT);  
 86 \*/  
 87 //printf("program started. \n");  
 88 //while(1)  
 89 //{  
 90 //delay(20);  
 91 //printf("entered while 1 loop\n");  
 92 int button = scanKeypadM();  
 93 if(button != 0)  
 94 {  
 95 //printf("you Pressed: %d \n", button);  
 96 colPressed =0;  
 97 rowPressed = 0;  
 98 }  
 99 else  
100 {  
101 //printf("nothing pressed\n");  
102 }  
103 // delay(100);  
104 //}  
105 return button;  
106 }  
107   
108 /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
109 int scanKeypad(void)  
110   
111 This function scans the keypad via the gpio extender mcp 23017.  
112 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
113 int scanKeypadM(void)  
114 {  
115 //printf("entered scanKeypad\n");  
116 //delay(100);  
117   
118 /\*  
119 wiringPiSetupGpio(); //required wiringPi library set up function  
120 //mcp23017Setup(100, 0x20); //required mcp23017 extender set up function.  
121   
122 int i;  
123 for (i = 0; i < 8; i++) //setting up pin modes.  
124 {  
125 if(i<4)  
126 {  
127 pinMode (19+i, OUTPUT); //all rows = outputs, all columns = inputs.  
128 }  
129 else  
130 {   
131 pinMode (19+i, INPUT);  
132 }  
133 }  
134 \*/  
135 if (scanLoop()) //if a row and column is press, return the button pressed to caller  
136 {  
137 int buttonPressed = colPressed + ((rowPressed-1)\*4);  
138   
139 return buttonPressed;  
140 }  
141 else  
142 {  
143 return 0;  
144 }  
145   
146 }  
147   
148   
149 /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
150 int scanLoop(void)  
151   
152 This function performs a nested loop to scan both rows and columns. If a button is   
153 pressed, it will store the column and row that was pressed in the global variables   
154 rowPressed, and colPressed.  
155 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
156 int scanLoop(void)  
157 {  
158 // printf("entered scanLoop\n");  
159 // delay(100);  
160   
161 int r;  
162 //int row;  
163 for(r = 1; r<5; r++) //Loop to decide which row to check.  
164 {  
165   
166 //delay(20);  
167 //printf(" row check: %d \n",r);  
168 digitalWrite(103+r, HIGH); //set row "r" high.   
169 delay(5); //debounce delay  
170 int c;  
171 int col;  
172 for(c = 1; c < 5; c++) //Loop to scan columns  
173 {  
174 if(c==1)  
175 {  
176 col = 21;  
177 }  
178 else if(c==2)  
179 {  
180 col = 10;  
181 }  
182 else if(c==3)  
183 {  
184 col = 19;  
185 }  
186 else if(c==4)  
187 {  
188 col = 9;  
189 }  
190   
191 //delay(20);  
192 // printf("col check: %d \n",c);  
193 if (digitalRead(col)) //if column is high, save the row and column pressed.  
194 {  
195 //delay(20);  
196 rowPressed = r;  
197 colPressed = c;  
198 // printf("rowPressed: %d \n",r);  
199 // printf("colPressed: %d \n",c);  
200 //delay(20);  
201 digitalWrite(103+r, LOW);  
202 //delay(20);  
203 break;  
204 }  
205 //else  
206 //{}  
207 }  
208 if (colPressed > 0)  
209 {  
210 //delay(20);  
211 digitalWrite(103+r, LOW);  
212 //delay(20);  
213 return rowPressed;  
214 }  
215 //delay(20);  
216 digitalWrite(103+r, LOW);  
217 //delay(20);  
218 }  
219 return 0;  
220 }  
221   
222   
223 //int newScan()  
224

1 /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
 2 \* lcd4BitMode.c  
 3 \* Group 4: Automatic Pet Feeder  
 4 \* Group Members: Raul Rojas, Nyron Daniel & Isrreal Mendoza  
 5 \*   
 6 \* This program demonstrates the ability to control and minipulate the LCD display of our  
 7 \* Automatic Pet Feeder. When executed, this program will show the names of our group   
 8 \* members, as well as the class name, term and our group number. The lcd will then   
 9 \* display a set of random characters and will then be cleared.  
10 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
11 #include <wiringPi.h>  
12 #include <lcd.h>  
13 #include <stdio.h>  
14 #include <mcp23017.h>  
15   
16 #define LCD\_RS 108  
17 #define LCD\_E 109  
18 #define LCD\_D4 110  
19 #define LCD\_D5 111  
20 #define LCD\_D6 112  
21 #define LCD\_D7 113  
22 extern int lcd; //Handle for LCD  
23   
24 int lcdTest(void);  
25   
int lcdTest(void)  
27 {  
47   
48 lcdPosition(lcd,0,0); //Postion cursor on the first line in the first column  
49 lcdPuts(lcd,"Nyron Daniel"); //prints our names, class, term, and group number  
50 delay(1000);  
51 lcdPosition(lcd,0,1);  
52 lcdPuts(lcd,"Raul Rojas");  
53 delay(1000);  
54 lcdPosition(lcd,0,0);  
55 lcdPuts(lcd,"Isrrael Mendoza");  
56 delay(1000);  
57 lcdPosition(lcd,0,0);  
58 lcdPuts(lcd,"Digital 3-Group4");  
59 lcdPosition(lcd,0,1);  
60 lcdPuts(lcd,"Spring 2016");  
61 delay(2000);  
62 lcdClear(lcd); //clears LCD  
63   
64 80 return 0;  
81 }  
82

1 /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
 2 \* sevenSegmentnested.c  
 3 \*--------------------------------------------------------------------------------------------  
 4 \* By: Nyron Daniel  
 5 \* Group 4: Automatic Pet Feeder  
 6 \* Group Members: Raul Rojas, Nyron Daniel & Isrreal Mendoza  
 7 \*   
 8 \* This program demonstrates the ability to control and minipulate the 4 digit 7   
 9 \* segment of our Automatic Pet Feeder. When executed this program will begin to   
 10 \* count from 0 to 9 on each digit of the display and will then be cleared. It will  
 11 \* continue to cycle through the count after each clear until the program is terminated.  
 12 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
 13   
 14 #include <stdio.h>  
 15 #include <wiringPiI2C.h>  
 16 #include <wiringPi.h>  
 17 #include "finalMain.h"  
 18 /\*  
 19 //declaring function prototypes   
 20 void delay();   
 21 void segWrite(int Digit,int Data);  
 22 void segClear(void);  
 23 void sevenSegmentTest (void);  
 24 \*/  
 25 extern int fd; //including variables from other files   
 26   
 27 /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
 28 void sevenSegmentTest (void)  
 29   
 30 This function tests the functionality of the 4 digit 7 segment display. once this   
 31 function is called it stays in an infinate while loop.   
 32 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
 33 void sevenSegmentTest (void)  
 34 {  
 35 /\*   
 36 wiringPiSetupGpio();  
 37 if((fd=wiringPiI2CSetup(0x71)))//Intializing the seven segment on the I2C bus.  
 38 {  
 39   
 40 }  
 41 \*/  
 42 //while(1)  
 43 //{  
 44 segWrite (0,0);//Displays a zero on the left most digit (digit1) of the seven segment.  
 45 segWrite (1,0);//Displays a zero on (digit2) of the seven segment.  
 46 segWrite (2,0);//Displays a zero on (digit3) of the seven segment.  
 47 segWrite (3,0);//Displays a zero on (digit4) of the seven segment.  
 48 wiringPiI2CWriteReg8 (fd,decimal,colon);//displays the colon on the seven segment.   
 49 delay(1000);  
 50   
 51 segWrite (0,1);//Displays a one on the left most digit (digit1) of the seven segment.  
 52 segWrite (1,1);  
 53 segWrite (2,1);  
 54 segWrite (3,1);  
 55 wiringPiI2CWriteReg8 (fd,decimal,colon);   
 56 delay(1000);  
 57   
 58 segWrite (0,2);  
 59 segWrite (1,2);  
 60 segWrite (2,2);  
 61 segWrite (3,2);  
 62 wiringPiI2CWriteReg8 (fd,decimal,colon);   
 63 delay(1000);  
 64   
 65 segWrite (0,3);  
 66 segWrite (1,3);  
 67 segWrite (2,3);  
 68 segWrite (3,3);  
 69 wiringPiI2CWriteReg8 (fd,decimal,colon);   
 70 delay(1000);  
 71   
 72   
 73 segWrite (0,4);  
 74 segWrite (1,4);  
 75 segWrite (2,4);  
 76 segWrite (3,4);  
 77 wiringPiI2CWriteReg8 (fd,decimal,colon);   
 78 delay(1000);  
 79   
 80 segWrite (0,5);  
 81 segWrite (1,5);  
 82 segWrite (2,5);  
 83 segWrite (3,5);  
 84 wiringPiI2CWriteReg8 (fd,decimal,colon);   
 85 delay(1000);  
 86   
87 segWrite (0,6);  
 88 segWrite (1,6);  
 89 segWrite (2,6);  
 90 segWrite (3,6);  
 91 wiringPiI2CWriteReg8 (fd,decimal,colon);   
 92 delay(1000);  
 93   
 94 segWrite (0,7);  
 95 segWrite (1,7);  
 96 segWrite (2,7);  
 97 segWrite (3,7);  
 98 wiringPiI2CWriteReg8 (fd,decimal,colon);   
 99 delay(1000);  
100   
101 segWrite (0,8);  
102 segWrite (1,8);  
103 segWrite (2,8);  
104 segWrite (3,8);  
105 wiringPiI2CWriteReg8 (fd,decimal,colon);   
106 delay(1000);  
107   
108 segWrite (0,9);  
109 segWrite (1,9);  
110 segWrite (2,9);  
111 segWrite (3,9);  
112 wiringPiI2CWriteReg8 (fd,decimal,colon);   
113 delay(1000);  
114 segClear ();//Call a function that clears the seven segment.  
115 //}   
116 //return 0;  
117 }  
118   
119 /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
120 \*(segWrite) Write to the segment Function  
121 \* By: Nyron Daniel  
122 \*  
123 \* This function checks for the key pressed on the keypad and then displays the pressed digit  
124 \* on the four digit seven segment.  
125 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
126   
127 void segWrite(int Digit,int Data)  
128 {  
129 switch (Digit)//Checks to see what digit is selected   
130 {  
131 case 0: //In this case the left most digit (digit1) was selected.  
132 switch (Data) //Checks to see what number will be displayed.   
133 { case 0:  
134 wiringPiI2CWriteReg8 (fd,digit1,zero);//Writes a zero to digit 1.  
135 printf("Digit 1 - Number 0 was selected\n");  
136 break;  
137 case 1:  
138 wiringPiI2CWriteReg8 (fd,digit1,one);//Writes a one to digit 1.  
139 printf("Digit 1 - Number 1 was selected\n");  
140 break;  
141 case 2:  
142 wiringPiI2CWriteReg8 (fd,digit1,two);//Writes a two to digit 1.  
143 printf("Digit 1 - Number 2 was selected\n");  
144 break;  
145 case 3:  
146 wiringPiI2CWriteReg8 (fd,digit1,three);//Writes a three to digit 1.  
147 printf("Digit 1 - Number 3 was selected\n");  
148 break;  
149 case 4:  
150 wiringPiI2CWriteReg8 (fd,digit1,four);//Writes a four to digit 1.  
151 printf("Digit 1 - Number 4 was selected\n");  
152 break;  
153 case 5:  
154 wiringPiI2CWriteReg8 (fd,digit1,five);//Writes a five to digit 1.  
155 printf("Digit 1 - Number 5 was selected\n");  
156 break;  
157 case 6:  
158 wiringPiI2CWriteReg8 (fd,digit1,six);//Writes a six to digit 1.  
159 printf("Digit 1 - Number 6 was selected\n");  
160 break;  
161 case 7:  
162 wiringPiI2CWriteReg8 (fd,digit1,seven);//Writes a seven to digit 1.  
163 printf("Digit 1 - Number 7 was selected\n");  
164 break;  
165 case 8:  
166 wiringPiI2CWriteReg8 (fd,digit1,eight);//Writes an eight to digit 1.  
167 printf("Digit 1 - Number 8 was selected\n");  
168 break;  
169 case 9:  
170 wiringPiI2CWriteReg8 (fd,digit1,nine);//Writes a nine to digit 1.  
171 printf("Digit 1 - Number 9 was selected\n");  
172 break;  
173 default:  
174 printf("Invalid number was selected\n");  
175   
176 }  
177   
 break;  
179 case 1: // In this case the second digit from the left was selected.  
180 switch (Data) //Checks to see what number needs to be displayed.   
181 { case 0:  
182 wiringPiI2CWriteReg8 (fd,digit2,zero);//Writes a zero to digit 2.  
183 printf("Digit 2 - Number 0 was selected\n");  
184 break;  
185 case 1:  
186 wiringPiI2CWriteReg8 (fd,digit2,one);  
187 printf("Digit 2 - Number 1 was selected\n");  
188 break;  
189 case 2:  
190 wiringPiI2CWriteReg8 (fd,digit2,two);  
191 printf("Digit 2 - Number 2 was selected\n");  
192 break;  
193 case 3:  
194 wiringPiI2CWriteReg8 (fd,digit2,three);  
195 printf("Digit 2 - Number 3 was selected\n");  
196 break;  
197 case 4:  
198 wiringPiI2CWriteReg8 (fd,digit2,four);  
199 printf("Digit 2 - Number 4 was selected\n");  
200 break;  
201 case 5:  
202 wiringPiI2CWriteReg8 (fd,digit2,five);  
203 printf("Digit 2 - Number 5 was selected\n");  
204 break;  
205 case 6:  
206 wiringPiI2CWriteReg8 (fd,digit2,six);  
207 printf("Digit 2 - Number 6 was selected\n");  
208 break;  
209 case 7:  
210 wiringPiI2CWriteReg8 (fd,digit2,seven);  
211 printf("Digit 2 - Number 7 was selected\n");  
212 break;  
213 case 8:  
214 wiringPiI2CWriteReg8 (fd,digit2,eight);  
215 printf("Digit 2 - Number 8 was selected\n");  
216 break;  
217 case 9:  
218 wiringPiI2CWriteReg8 (fd,digit2,nine);  
219 printf("Digit 2 - Number 9 was selected\n");  
220 break;  
221 default:  
222 printf("Invalid number was selected\n");  
223   
224 }  
225   
226 break;  
227 case 2://In this case the third from the left digit is selected.  
228 switch (Data)   
229 { case 0:  
230 wiringPiI2CWriteReg8 (fd,digit3,zero);  
231 printf("Digit 3 - Number 0 was selected\n");  
232 break;  
233 case 1:  
234 wiringPiI2CWriteReg8 (fd,digit3,one);  
235 printf("Digit 3 - Number 1 was selected\n");  
236 break;  
237 case 2:  
238 wiringPiI2CWriteReg8 (fd,digit3,two);  
239 printf("Digit 3 - Number 2 was selected\n");  
240 break;  
241 case 3:  
242 wiringPiI2CWriteReg8 (fd,digit3,three);  
243 printf("Digit 3 - Number 3 was selected\n");  
244 break;  
245 case 4:  
246 wiringPiI2CWriteReg8 (fd,digit3,four);  
247 printf("Digit 3 - Number 4 was selected\n");  
248 break;  
249 case 5:  
250 wiringPiI2CWriteReg8 (fd,digit3,five);  
251 printf("Digit 3 - Number 5 was selected\n");  
252 break;  
253 case 6:  
254 wiringPiI2CWriteReg8 (fd,digit3,six);  
255 printf("Digit 3 - Number 6 was selected\n");  
256 break;  
257 case 7:  
258 wiringPiI2CWriteReg8 (fd,digit3,seven);  
259 printf("Digit 3 - Number 7 was selected\n");  
260 break;  
261 case 8:  
262 wiringPiI2CWriteReg8 (fd,digit3,eight);  
263 printf("Digit 3 - Number 8 was selected\n");  
264 break;  
265 case 9:  
266 wiringPiI2CWriteReg8 (fd,digit3,nine);  
267 printf("Digit 3 - Number 9 was selected\n");  
268 break;  
269 default:  
270 printf("Invalid number was selected\n");  
271 }   
272 break;  
273 case 3: //In this case the fourth digit is selected.  
274 switch (Data)   
275 { case 0:  
276 wiringPiI2CWriteReg8 (fd,digit4,zero);  
277 printf("Digit 4 - Number 0 was selected\n");  
278 break;  
279 case 1:  
280 wiringPiI2CWriteReg8 (fd,digit4,one);  
281 printf("Digit 4 - Number 1 was selected\n");  
282 break;  
283 case 2:  
284 wiringPiI2CWriteReg8 (fd,digit4,two);  
285 printf("Digit 4 - Number 2 was selected\n");  
286 break;  
287 case 3:  
288 wiringPiI2CWriteReg8 (fd,digit4,three);  
289 printf("Digit 4 - Number 3 was selected\n");  
290 break;  
291 case 4:  
292 wiringPiI2CWriteReg8 (fd,digit4,four);  
293 printf("Digit 4 - Number 4 was selected\n");  
294 break;  
295 case 5:  
296 wiringPiI2CWriteReg8 (fd,digit4,five);  
297 printf("Digit 4 - Number 5 was selected\n");  
298 break;  
299 case 6:  
300 wiringPiI2CWriteReg8 (fd,digit4,six);  
301 printf("Digit 4 - Number 6 was selected\n");  
302 break;  
303 case 7:  
304 wiringPiI2CWriteReg8 (fd,digit4,seven);  
305 printf("Digit 4 - Number 7 was selected\n");  
306 break;  
307 case 8:  
308 wiringPiI2CWriteReg8 (fd,digit4,eight);  
309 printf("Digit 4 - Number 8 was selected\n");  
310 break;  
311 case 9:  
312 wiringPiI2CWriteReg8 (fd,digit4,nine);  
313 printf("Digit 4 - Number 9 was selected\n");  
314 break;  
315 default:  
316 printf("Invalid number was selected\n");  
317   
318 }  
319   
320 }  
321 }  
322 /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
323 \*(segClear) Clears the segment Function  
324 \* By: Nyron Daniel  
325 \*  
326 \* This function clears the four digit seven segment and sets the cursor at the left most  
327 \* digit (Digit1)  
328 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
329   
330 void segClear (void)   
331 {  
332 wiringPiI2CWrite (fd,0x76); //Clears the seven segment.  
333 printf("Cleared Screen!\n");  
334 //delay(500);  
335   
336 }  
337 /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
338

1 #include <stdio.h>  
 2 #include <wiringPi.h>  
 3 #include "finalMain.h"  
 4   
 5 extern int petModeFlag;  
 6 extern int setModeFlag;  
 7 int petModeStorage = 0;  
 8   
 9 int hour;  
 10 int min;  
 11   
 12 extern int hourTenths;  
 13 extern int hourOnes;  
 14 extern int minTenths;  
 15 extern int minOnes;  
 16 extern char ap;  
 17 extern int perDay;  
 18 extern int timeBetweenHTenths;  
 19 extern int timeBetweenHOnes;  
 20 extern int timeBetweenMTenths;  
 21 extern int timeBetweenMOnes;  
 22   
 23 FILE \*hourTenthsFile;  
 24 FILE \*hourOnesFile;  
 25 FILE \*minTenthsFile;  
 26 FILE \*minOnesFile;  
 27 FILE \*apFile;  
 28 FILE \*perDayFile;  
 29 FILE \*timeBetweenHTenthsFile;  
 30 FILE \*timeBetweenHOnesFile;  
 31 FILE \*timeBetweenMTenthsFile;  
 32 FILE \*timeBetweenMOnesFile;  
 33   
 34   
 35 /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
 36 void saveHour(void)  
 37 ---------------------------------------------------------------------------------------------  
 38 This function saves the Hour portion of the time setting which was inputed by the user.   
 39 Each digit of the Hour is saved in seperate files to provide an easy method of data   
 40 retrieval.   
 41 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
 42 void saveHour(void)  
 43 {  
 44 //hour = (hourTenths \* 10) + hourOnes;  
 45 //printf("hour going in file: %d \n", hour);  
 46 //min = (minuteTenths \* 10) + minuteOnes;  
 47   
 48   
 49 //FILE \*hourTenthsFile;   
 50 //Open the Hour Tenths file.  
 51 hourTenthsFile = fopen("/home/pi/DigitalThree/final/userSettings/hourTenthsFile.txt", "w+");  
 52 printf("updating hourTenthsFile \n");  
 53 fprintf(hourTenthsFile, "%d", hourTenths); //write into file the Tenths digit of the Hour  
 54 fclose(hourTenthsFile); //close the file.  
 55 printf("finished writing hourThenthsFile\n");  
 56   
 57 delay(10);  
 58   
 59 //perform the same operation as above for the "one's" digit of the Hour.  
 60 hourOnesFile = fopen("/home/pi/DigitalThree/final/userSettings/hourOnesFile.txt", "w+");  
 61 printf("updating hourOnesFile \n");  
 62 fprintf(hourOnesFile, "%d", hourOnes);  
 63 fclose(hourOnesFile);  
 64 printf("finished writing hourOnesFile \n");  
 65 return;   
 66 }  
 67

68 /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
 69 void saveMin(void)  
 70 ---------------------------------------------------------------------------------------------  
 71 This Function saves the minutes that the user inputed in the set mode in the same   
 72 manner as the saveHour() function.   
 73 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
 74   
 75 void saveMin(void)  
 76 {  
 77 minTenthsFile = fopen("/home/pi/DigitalThree/final/userSettings/minTenthsFile.txt", "w+");  
 78 printf("updating minTenthsFile \n");  
 79 fprintf(minTenthsFile, "%d", minTenths);  
 80 fclose(minTenthsFile);  
 81 printf("finished writing hourThenthsFile\n");  
 82   
 83 delay(10);  
 84 minOnesFile = fopen("/home/pi/DigitalThree/final/userSettings/minOnesFile.txt", "w+");  
 85 printf("updating minOnesFile \n");  
 86 fprintf(minOnesFile, "%d", minOnes);  
 87 fclose(minOnesFile);  
 88 printf("finished writing minOnesFile \n");  
 89 return;   
 90 }  
 91   
 92 /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
 93 void getHour(void)  
 94 ---------------------------------------------------------------------------------------------  
 95 This function retrieves the Hour that was saved by the user and stores it in the Hour   
 96 global variables.   
 97 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
 98   
 99 void getHour(void)  
100 {  
101 //open the hour tenths file reading purposes only.   
102 hourTenthsFile = fopen("/home/pi/DigitalThree/final/userSettings/hourTenthsFile.txt", "r");  
103 //printf("Reading hourTenthsFile \n");  
104 hourTenths = fgetc(hourTenthsFile); //retrieve the first character of the tenths file.  
105 fclose(hourTenthsFile); //close the file.  
106 //character retrieved from file is in ASCii, must offset to get actual number.   
107 hourTenths = hourTenths - 48;   
108 //printf("hour from file in buffer is: %d \n", hourTenths);  
109   
110 //perform the same steps as above to retrieve the Ones digit of the Hour.   
111 hourOnesFile = fopen("/home/pi/DigitalThree/final/userSettings/hourOnesFile.txt", "r");  
112 //printf("Reading hourOnesFile \n");  
113 hourOnes = fgetc(hourOnesFile);  
114 fclose(hourOnesFile);  
115 hourOnes = hourOnes - 48;  
116 //printf("hour from file in buffer is: %d \n", hourOnes);  
117   
118 return;   
119 }  
120   
121 /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
122 void getMin(void)  
123 ---------------------------------------------------------------------------------------------  
124 This function retieves the Minutes saved by the user in the same manner as getHour()   
125 funciton.   
126 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
127 void getMin(void)  
128 {  
129 minTenthsFile = fopen("/home/pi/DigitalThree/final/userSettings/minTenthsFile.txt", "r");  
130 //printf("Reading minTenthsFile \n");  
131 minTenths = fgetc(minTenthsFile);  
132 fclose(minTenthsFile);  
133 minTenths = minTenths - 48;  
134 //printf("hour from file in buffer is: %d \n", minTenths);  
135   
136 minOnesFile = fopen("/home/pi/DigitalThree/final/userSettings/minOnesFile.txt", "r");  
137 //printf("Reading minOnesFile \n");  
138 minOnes = fgetc(minOnesFile);  
139 fclose(minOnesFile);  
140 minOnes = minOnes - 48;  
141 //printf("hour from file in buffer is: %d \n", minOnes);  
142   
143 return;   
144 }  
145   
146 /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
147 void saveAP(void)  
148 ---------------------------------------------------------------------------------------------  
149 This Function saves whether the user inputed an AM or PM time for the first dispense of   
150 the day.   
151 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
152 void saveAP(void)  
153 {  
154 //open the AM/PM file for writing purposes and clear the previous entry.  
155 apFile = fopen("/home/pi/DigitalThree/final/userSettings/apFile.txt", "w+");  
156 printf("updating apFile \n");  
157 fprintf(apFile, "%c", ap); //write character into the file.  
158 fclose(apFile); //close the file  
159 printf("finished writing apFile \n");  
160 return;  
161 }  
162   
163 /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
164 void getAP(void)  
165 ---------------------------------------------------------------------------------------------  
166 This Function retrieves the settings for AM or PM which the user inputted for the   
167 first dispense of the day.   
168 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
169 void getAP(void)  
170 {  
171 // open document for reading purposes only.   
172 apFile = fopen("/home/pi/DigitalThree/final/userSettings/apFile.txt", "r");  
173 printf("Reading apFile \n");   
174 ap = fgetc(apFile); // retrieve character and store it into the global variable   
175 fclose(apFile); // close file.   
176 printf("Character from apFile is: %c \n", ap);  
177 }  
178   
179 /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
180 void savePerDay(void)  
181 ---------------------------------------------------------------------------------------------  
182 This Function saves the amount of dispenses per day that the user has set. This function  
183 is similar to the saveHour() function however, it saves into another file.   
184 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
185 void savePerDay(void)  
186 {  
187 perDayFile = fopen("/home/pi/DigitalThree/final/userSettings/perDayFile.txt", "w+");  
188 printf("updating perDayFile \n");  
189 fprintf(perDayFile, "%d", perDay);  
190 fclose(perDayFile);  
191 printf("finished writing perDayFile.\n");  
192 return;  
193 }  
194   
/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
196 void getPerDay(void)  
197 ---------------------------------------------------------------------------------------------  
198 This Function retrieves the dispenses per day value that was saved by the user.   
199 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
200   
201 void getPerDay(void)  
202 {  
203 perDayFile = fopen("/home/pi/DigitalThree/final/userSettings/perDayFile.txt", "r");  
204 printf("Reading perDayFile \n");  
205 perDay = fgetc(perDayFile);  
206 fclose(perDayFile);  
207 perDay = perDay - 48;  
208 printf("perDay value from file is: %d \n", perDay);  
209 return;  
210 }  
211   
212 /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
213 void saveTimeBetween(void)  
214 ---------------------------------------------------------------------------------------------  
215 This Funciton saves the time between each dispense setting that the user specified in   
216 the set mode.   
217 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
218 void saveTimeBetween(void)  
219 {  
220 timeBetweenHTenthsFile = fopen("/home/pi/DigitalThree/final/userSettings/timeBetweenHTenthsFile.txt", "w+");  
221 //printf("updating timeBetweenHTenthsFile \n");  
222 fprintf(timeBetweenHTenthsFile, "%d", timeBetweenHTenths);  
223 fclose(timeBetweenHTenthsFile);  
224 //printf("finished writing timeBetweenHTenthsFile.\n");  
225   
226 timeBetweenHOnesFile = fopen("/home/pi/DigitalThree/final/userSettings/timeBetweenHOnesFile.txt", "w+");  
227 //printf("updating timeBetweenHOnesFile \n");  
228 fprintf(timeBetweenHOnesFile, "%d", timeBetweenHOnes);  
229 fclose(timeBetweenHOnesFile);  
230 //printf("finished writing timeBetweenHOnesFile.\n");  
231   
232 timeBetweenMTenthsFile = fopen("/home/pi/DigitalThree/final/userSettings/timeBetweenMTenthsFile.txt", "w+");  
233 //printf("updating timeBetweenMTenthsFile \n");  
234 fprintf(timeBetweenMTenthsFile, "%d", timeBetweenMTenths);  
235 fclose(timeBetweenMTenthsFile);  
236 //printf("finished writing timeBetweenMTenthsFile.\n");  
237   
238 timeBetweenMOnesFile = fopen("/home/pi/DigitalThree/final/userSettings/timeBetweenMOnesFile.txt", "w+");  
239 //printf("updating timeBetweenMOnesFile \n");  
240 fprintf(timeBetweenMOnesFile, "%d", timeBetweenMOnes);  
241 fclose(timeBetweenMOnesFile);  
242 //printf("finished writing timeBetweenMOnesFile.\n");  
243 return;  
244 }  
245   
246 /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
247 void getTimeBetween(void)  
248 ---------------------------------------------------------------------------------------------  
249 This function retrieves the time between each dispense setting that was saved by the   
250 user.  
251 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
252   
253 void getTimeBetween(void)  
254 {  
255 timeBetweenHTenthsFile = fopen("/home/pi/DigitalThree/final/userSettings/timeBetweenHTenthsFile.txt", "r");  
256 //printf("Reading timeBetweenHTenthsFile \n");  
257 timeBetweenHTenths = fgetc(timeBetweenHTenthsFile);  
258 fclose(timeBetweenHTenthsFile);  
259 timeBetweenHTenths = timeBetweenHTenths - 48;  
260 //printf("timeBetweenHTenths value from file is: %d \n", timeBetweenHTenths);  
261   
262 timeBetweenHOnesFile = fopen("/home/pi/DigitalThree/final/userSettings/timeBetweenHOnesFile.txt", "r");  
263 //printf("Reading timeBetweenHOnesFile \n");  
264 timeBetweenHOnes = fgetc(timeBetweenHOnesFile);  
265 fclose(timeBetweenHOnesFile);  
266 timeBetweenHOnes = timeBetweenHOnes - 48;  
267 //printf("timeBetweenHOnes value from file is: %d \n", timeBetweenHOnes);  
268   
269 timeBetweenMTenthsFile = fopen("/home/pi/DigitalThree/final/userSettings/timeBetweenMTenthsFile.txt", "r");  
270 //printf("Reading timeBetweenMTenthsFile \n");  
271 timeBetweenMTenths = fgetc(timeBetweenMTenthsFile);  
272 fclose(timeBetweenMTenthsFile);  
273 timeBetweenMTenths = timeBetweenMTenths - 48;  
274 //printf("timeBetweenMTenths value from file is: %d \n", timeBetweenMTenths);  
275   
276 timeBetweenMOnesFile = fopen("/home/pi/DigitalThree/final/userSettings/timeBetweenMOnesFile.txt", "r");  
277 //printf("Reading timeBetweenMOnesFile \n");  
278 timeBetweenMOnes = fgetc(timeBetweenMOnesFile);  
279 fclose(timeBetweenMOnesFile);  
280 timeBetweenMOnes = timeBetweenMOnes - 48;  
281 //printf("timeBetweenMOnes value from file is: %d \n", timeBetweenMOnes);  
282 return;  
283 }  
284

1 #include <stdio.h>  
 2 #include <stdlib.h>  
 3 #include <time.h>  
 4 #include <wiringPi.h>  
 5 #include <lcd.h>  
 6 #include "finalMain.h"  
 7   
 8 extern int lcd;  
 9   
 10 extern int hour;  
 11 extern int min;  
 12   
 13 int number;  
 14 int numberStorage;  
 15 int buttonPressed;  
 16 //int cursor = 0;  
 17 //int count = 27;  
 18 int buttonFlag = 0;  
 19 int incFlag = 0;  
 20 int decFlag = 0;  
 21 int toggleAP;  
 22 int currentAP;  
 23 int cursorShiftedF;  
 24 int setModeStep;  
 25   
 26 int hourTenths;  
 27 int hourOnes;  
 28 int minTenths;  
 29 int minOnes;  
 30 char ap;  
 31 int perDay;  
 32 //int timeBetweenHTenths;  
 33 int timeBetweenHTenths;  
 34 int timeBetweenHOnes;  
 35 int timeBetweenMTenths;  
 36 int timeBetweenMOnes;  
 37

38 /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
 39 void setMode(void)  
 40 ---------------------------------------------------------------------------------------------  
 41 This function services the setModeFlag request which is set by the user and detected in   
 42 the smartDelay() function. This mode of operation contains 3 steps and user may exit   
 43 while in any of the steps. The steps are as follows:   
 44   
 45 1. timeSet(): step in which the user edits the first dispense of the day. This funciton   
 46 provides realtime feedback of what is being inputed and also adjust or coerces what   
 47 the user inputs if the values inputted are considered an invalide entry.   
 48   
 49 2. timesPerDay(): step in which the user edits the amount of dispenses per day.   
 50   
 51 3. timeBetweenDisp(): step in which the user edits the time between each dispense.  
 52 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
 53 void setMode(void)  
 54 {  
 55 setModeStep = 0; //keeps up with the step number in the set mode.  
 56 while(digitalRead(setModeSw) && digitalRead(powerSw) && !digitalRead(resetSw))  
 57 {  
 58 delay(10);  
 59 switch (setModeStep)  
 60 {  
 61 case 0:  
 62 printf("in case zero.\n");  
 63 timeSet();  
 64 break;  
 65   
 66 case 1:  
 67 printf("in case one.\n");  
 68 timesPerDay();  
 69 break;  
 70   
 71 case 2:  
 72 if(perDay == 1) // if only 1 dispens perday, dont need time between.  
 73 {  
 74 setModeStep = 0;  
 75 //timeBetweenHTenths = 0;  
 76 //timeBetweenHTenths = 1;  
 77 break;  
 78 }  
 79 printf("in case two.\n");  
 80 timeBetweenDisp();  
 81 break;  
 82 }  
 83   
 84 while(scanKeypad() == 8) //while the keypad is being pressed, wait until release.  
 85 {  
 86 delay(10);  
 87 }  
 88 }  
 89 //save all inputed settings by user.  
 90 saveHour();  
 91 getHour();  
 92 saveMin();  
 93 getMin();  
 94 saveAP();  
 95 getAP();  
 96   
 97 savePerDay();  
 98 getPerDay();  
 99   
 100 saveTimeBetween();  
 101 getTimeBetween();  
 102   
 103 //clear the lcd and show the default standby display on both lcd and 7 segment.   
 104 lcdClear(lcd);  
 105 defaultDisplay();  
 106 return;  
 107 } //end of setMode();  
 108   
 109   
 110 /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
 111 void timeSet(void)  
 112 ---------------------------------------------------------------------------------------------  
 113 This function contains the step in which the user inputs the desired first dispense of   
 114 the day. The function uses a Switch structure to keep up with the current cursor   
 115 position which the user is editing. This funciton also coerces/adjusts the input if the   
 116 input is considered invalid for that cursor position.   
 117 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
 118 void timeSet(void)  
 119 {  
 120 int cursor = 0;  
 121 int count = 27;  
 122 toggleAP = 0;  
 123   
 124 lcdClear(lcd);  
 125 lcdPosition(lcd, 0, 1);  
 126 lcdPrintf(lcd, "First dispense."); // show timeSet display.  
 127 lcdPosition(lcd, 0, 0);  
 128 getHour(); //retrieve previousely saved settings.   
 129 getMin();  
 130 getAP();  
 131 if(ap == 'A')  
 132 {  
 133 currentAP = 0; //if 0, AM  
 134 }  
 135 else   
 136 {  
 137 currentAP = 1; // if 1, PM  
 138 }  
 139   
 140 printf("Hour Stored: %d%d \n", hourTenths, hourOnes);  
 141 if(hourTenths == 0 && hourOnes == 0)  
 142 {  
 143 lcdPrintf(lcd, " 01:00 AM"); //if file was empty and both hour/min ==0 display 1am  
 144 }  
 145 else   
 146 {  
 147 //display the previously saved settings.  
 148 lcdPrintf(lcd, " %d%d:%d%d %cM", hourTenths, hourOnes, minTenths,   
 149 minOnes, ap);  
 150 }  
 151   
 152 while(digitalRead(setModeSw) && digitalRead(powerSw) && !digitalRead(resetSw))  
 153 {  
 154 count--; // count keeps up with time/iterations  
 155 printf("in set Mode. count: %d \nnumber: %d \n", count, number);  
 156 //delay(20);  
 157 //cursor == 0: adjusting hour tenths  
 158 //cursor == 1: adjusting hour Ones  
 159 //cursor == 2: adjusting Minute tenths  
 160 //cursor == 3: adjusting Minute Ones  
 161 //cursor == 4: adjusting AM/PM  
 162 switch (cursor)  
 163 {  
 164 case 0:  
 165 printf("entered case 0. \n");  
 166 lcdPosition(lcd, 4, 0);   
 167   
 168 if (count == 26) //blink off for 6 iterations (20ms\*6)  
 169 {  
 170 lcdPrintf(lcd, " ");  
 171 }  
 172 else if(count == 20 || buttonFlag == 1) //blink on for 20 iterations (20ms\*20)  
 173 { // or when a button was pressed  
 174 //printf("update LCD")  
 175 if(buttonFlag == 1)   
 176 {  
 177 //if a numbered button was pressed, assign it and display it.  
 178 hourTenths = numberStorage;  
 179 if(hourTenths > 1) //in standard time, hour tenths cannot be >1  
 180 {  
 181 // to avoid frustration by user, if input was > 1, use it for Ones.  
 182 hourOnes = hourTenths;  
 183 hourTenths = 0;  
 184   
 185 //if(hourOnes > 2)  
 186 //{  
 187 //hourOnes = 2;  
 188 //represent the adjustment to the display   
 189 lcdPosition(lcd, 5, 0);  
 190 lcdPrintf(lcd, "%d", hourOnes);  
 191 lcdPosition(lcd, 4, 0);  
 192 cursor++; //a button was pressed so move to the next digit  
 193 //}  
 194 }  
 195 //used to avoid input a number higher than 12 in hours.  
 196 if(hourTenths == 1 && hourOnes > 2)   
 197 {  
 198 hourOnes = 0;  
 199 lcdPosition(lcd, 5, 0);  
 200 lcdPrintf(lcd, "%d", hourOnes);  
 201 lcdPosition(lcd, 4, 0);  
 202 }  
 203 //used to avoid inputing 00 hours. This is not standard time.   
 204 if(hourTenths == 0 && hourOnes == 0)  
 205 {  
 206 hourOnes = 1;  
 207 lcdPosition(lcd, 5, 0);  
 208 lcdPrintf(lcd, "%d", hourOnes);  
 209 lcdPosition(lcd, 4, 0);  
 210 }  
 211 //numberStorage = number;  
 212 lcdPrintf(lcd, "%d", hourTenths);  
 213 //if button was pressed and is possible to do so, move to next digit.  
 214 if(decFlag == 0 && incFlag ==0 && cursor < 5)  
 215 {  
 216   
 217 cursor++;  
 218 }  
 219   
 220 count = 27;  
 221 buttonFlag = 0;  
 222 }  
 223 else  
 224 {  
 225 //if no adjustments had to be made, then just display.  
 226 lcdPrintf(lcd, "%d", hourTenths);  
 227 }  
 228 }  
 229 //check if user pressed "\*" to move cursor position left  
 230 if(decFlag)  
 231 {  
 232 cursor = 4;  
 233 decFlag = 0; //reset flag  
 234 count = 27; //reset count  
 235 }  
 236 //check if user pressed "#" to move cursor position right  
 237 if(incFlag)  
 238 {  
 239 cursor++;  
 240 incFlag = 0;  
 241 count = 27;  
 242 }  
 243 //move to the AM/PM cursor position if user pressed "A" on the Keyboard.  
 244 if (toggleAP == 1)  
 245 {  
 246 cursor = 4;  
 247 }  
 248 //if user changed the cursor position, display the tenths before exiting case  
 249 if(cursor != 0)  
 250 {  
 251 lcdPosition(lcd, 4, 0);  
 252 lcdPrintf(lcd, "%d", hourTenths);  
 253 }  
 254 //reset count if needed.  
 255 if(count == 0)  
 256 {  
 257 count = 27;  
 258 }  
 259 break;  
 260   
 261 //for detailed comments, see case 0. most is same except now adjusting ones   
 262 // of hours  
 263 case 1:  
 264 printf("entered case 1. \n");  
 265 lcdPosition(lcd, 5, 0);  
 266   
 267 if (count == 26)  
 268 {  
 269 lcdPrintf(lcd, " ");  
 270 }  
 271 else if(count == 20 || buttonFlag == 1)  
 272 {  
 273 //printf("update LCD")  
 274 if(buttonFlag == 1)  
 275 {  
 276 hourOnes = numberStorage;  
 277 if(hourTenths == 1 && hourOnes > 2)  
 278 {  
 279 hourOnes = 2;  
 280 }  
 281 if(hourTenths == 0 && hourOnes == 0)  
 282 {  
 283 hourOnes = 1;  
 284 }  
 285 //numberStorage = number;  
 286 lcdPrintf(lcd, "%d", hourOnes);  
 287 if(decFlag == 0 && incFlag ==0 && cursor < 5)  
 288 {  
 289 cursor++;  
 290 }  
 291 count = 27;  
 292 buttonFlag = 0;  
 293 }  
 294 else  
 295 {  
 296 lcdPrintf(lcd, "%d", hourOnes);  
 297 }  
 298 }  
 299   
 300 if(decFlag)  
 301 {  
 302 cursor--;  
 303 decFlag = 0;  
 304 count = 27;  
 305 }  
 306 if(incFlag)  
 307 {  
 308 cursor++;  
 309 incFlag = 0;  
 310 count = 27;  
 311 }  
 312   
 313 if (toggleAP == 1)  
 314 {  
 315 cursor = 4;  
 316 }  
 317 if(cursor != 1)  
 318 {  
 319 lcdPosition(lcd, 5, 0);  
 320 lcdPrintf(lcd, "%d", hourOnes);  
 321 }  
 322   
 323 if(count == 0)  
 324 {  
 325 count = 27;  
 326 }  
 327 break;  
 328 //for detailed comments refer to case 0. most is same except now adjusting tenths  
 329 // place in minutes.   
 330 case 2:  
 331 printf("entered case 2. \n");  
 332 lcdPosition(lcd, 7, 0);  
 333   
 334 if (count == 26)  
 335 {  
 336 lcdPrintf(lcd, " ");  
 337 }  
 338 else if(count == 20 || buttonFlag == 1)  
 339 {  
 340 //printf("update LCD")  
 341 if(buttonFlag == 1)  
 342 {  
 343 minTenths = numberStorage;  
 344 if(minTenths > 5) //minutes cannot go beyond 59.  
 345 {  
 346 minTenths = 5;  
 347 }  
 348 //numberStorage = number;  
 349 lcdPrintf(lcd, "%d", minTenths);  
 350 if(decFlag == 0 && incFlag ==0 && cursor < 5)  
 351 {  
 352 cursor++;  
 353 }  
 354 count = 27;  
 355 buttonFlag = 0;  
 356 }  
 357 else  
 358 {  
 359 lcdPrintf(lcd, "%d", minTenths);  
 360 }  
 361 }  
 362   
 363 if(decFlag)  
 364 {  
 365 cursor--;  
 366 decFlag = 0;  
 367 count = 27;  
 368 }  
 369 if(incFlag)  
 370 {  
 371 cursor++;  
 372 incFlag = 0;  
 373 count = 27;  
 374 }  
 375   
 376 if (toggleAP == 1)  
 377 {  
 378 cursor = 4;  
 379 }  
 380 if(cursor != 2)  
 381 {  
 382 lcdPosition(lcd, 7, 0);  
 383 lcdPrintf(lcd, "%d", minTenths);  
 384 }  
 385   
 386 if(count == 0)  
 387 {  
 388 count = 27;  
 389 }  
 390 break;  
 391   
 392 //for detailed comments refer to case 0. most is same except now adjusting ones   
 393 // place for minutes.   
 394 case 3:  
 395 printf("entered case 3. \n");  
 396 lcdPosition(lcd, 8, 0);  
 397   
 398 if (count == 26)  
 399 {  
 400 lcdPrintf(lcd, " ");  
 401 }  
 402 else if(count == 20 || buttonFlag == 1)  
 403 {  
 404   
 405 //printf("update LCD")  
 406 if(buttonFlag == 1)  
 407 {  
 408 minOnes = numberStorage;  
 409 /\*  
 410 if(!cursorShiftedF)  
 411 {  
 412 minOnes = numberStorage;  
 413 }  
 414 else  
 415 {  
 416 cursorShiftedF = 0;  
 417 }  
 418 \*/  
 419 //numberStorage = number;  
 420 lcdPrintf(lcd, "%d", minOnes);  
 421 if(decFlag == 0 && incFlag ==0 && cursor < 5)  
 422 {  
 423 cursor++;  
 424 }  
 425 count = 27;  
 426 buttonFlag = 0;  
 427 }  
 428 else  
 429 {  
 430 lcdPrintf(lcd, "%d", minOnes);  
 431 }  
 432 }  
 433   
 434 if(decFlag)  
 435 {  
 436 cursor--;  
 437 decFlag = 0;  
 438 count = 27;  
 439 }  
 440 if(incFlag)  
 441 {  
 442 cursor++;  
 443 incFlag = 0;  
 444 count = 27;  
 445 }  
 446   
 447 if (toggleAP == 1)  
 448 {  
 449 cursor = 4;  
 450 }  
 451 if(cursor != 3)  
 452 {  
 453 lcdPosition(lcd, 8, 0);  
 454 lcdPrintf(lcd, "%d", minOnes);  
 455 }  
 456   
 457 if(count == 0)  
 458 {  
 459 count = 27;  
 460 }  
 461 break;   
 462 case 4:  
 463 printf("entered case 4. \n");  
 464 lcdPosition(lcd, 10, 0);  
 465   
 466 // creating a manual toggle  
 467 if(toggleAP)  
 468 {  
 469 if(currentAP == 0)  
 470 {  
 471 currentAP = 1;  
 472 }  
 473 else if (currentAP == 1)  
 474 {  
 475 currentAP = 0;  
 476 }  
 477 }  
 478   
 479 if (count == 26)  
 480 {  
 481 lcdPrintf(lcd, " ");  
 482 }  
 483 else if(count == 20 || toggleAP == 1 || decFlag)  
 484 {  
 485   
 486 if(currentAP == 1)  
 487 {  
 488 ap = 'P';  
 489 lcdPrintf(lcd, "%c", ap);  
 490   
 491 }  
 492 else  
 493 {  
 494 ap = 'A';  
 495 lcdPrintf(lcd, "%c", ap);  
 496 }  
 497   
 498 toggleAP =0;  
 499 }  
 500   
 501 if(count == 0)  
 502 {  
 503 count = 27;  
 504 }  
 505   
 506 if(incFlag)  
 507 {  
 508 cursor = 0;  
 509 incFlag = 0;  
 510 count = 27;  
 511 }  
 512   
 513 if(decFlag)  
 514 {  
 515 cursor--;  
 516 decFlag = 0;  
 517 count = 27;  
 518 cursorShiftedF = 1;  
 519 }  
 520   
 521 if(cursor != 4)  
 522 {  
 523 lcdPosition(lcd, 10, 0);  
 524 lcdPrintf(lcd, "%c", ap);  
 525 }  
 526   
 527 break;   
 528   
 529 }  
 530   
 531 //wait until user releases button on keypad to avoid multiple presses.  
 532 while((buttonPressed = scanKeypad()) > 0)  
 533 {  
 534   
 535 //buttonFlag = 1;  
 536   
 537 //if the button pressed on keypad actually represents a number, then display it.   
 538 number = numberPressed(buttonPressed);  
 539 if(number >= 0)  
 540 {  
 541 buttonFlag = 1;  
 542 }  
 543   
 544 // button "A" on keypad is equal to toggle.   
 545 if(buttonPressed == 4)  
 546 {  
 547 toggleAP = 1;   
 548 }  
 549   
 550 //"\*" button is the move cursor left   
 551 if(buttonPressed == 13 && cursor >= 0)  
 552 {  
 553   
 554 decFlag = 1;  
 555 }  
 556 //"#" button is the move cursor Right  
 557 if (buttonPressed == 15 && cursor <= 5)  
 558 {  
 559 incFlag = 1;  
 560 }  
 561   
 562 // "B" button is the step change button. if pressed exit this funciton.   
 563 if(buttonPressed == 8)  
 564 {  
 565 setModeStep++;  
 566 break;  
 567 }  
 568 }  
 569   
 570 //used to prevent displaying numbers in the AM/PM position.   
 571 if(number >= 0 && cursor != 4)  
 572 {  
 573 numberStorage = number;  
 574 //if(cursor < 5)  
 575 //{  
 576 //cursor++;  
 577 //}  
 578 }  
 579 //printf("numberStorage: %d \n", numberStorage);  
 580   
 581 //The following does a delay compensation from the scanKeypad() funciton.   
 582 // the scanKeypad provides a total of 20ms delay if no button is pressed.   
 583 // However, each row check adds 5 ms delay, depending on the row pressed,  
 584 //delay the follwing amount. reason is to keep a constant amount of delay.  
 585 if(buttonPressed > 12)  
 586 {  
 587 //printf("button pressed: %d \n", button);  
 588 }  
 589 else if(buttonPressed > 8)  
 590 {  
 591 delay(5);  
 592 }  
 593 else if(buttonPressed > 4)  
 594 {  
 595 delay(10);  
 596 }  
 597 else if(buttonPressed > 0)  
 598 {  
 599 delay(15);  
 600 }  
 601   
 602 if(buttonPressed == 8)  
 603 {  
 604 break;  
 605 }  
 606   
 607 }  
 608   
 609 //save settings before exit.   
 610 saveHour();  
 611 getHour();  
 612 saveMin();  
 613 getMin();  
 614 saveAP();  
 615 getAP();  
 616 return;  
 617 } //end of timeSet();  
 618   
 619 /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
 620 void timesPerDay(void)  
 621 ---------------------------------------------------------------------------------------------  
 622 This function contains the 2nd step of the set mode. In this step the user adjust the   
 623 amount of dispenses per day and displays feedback on the lcd screen.  
 624 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
 625 void timesPerDay(void)  
 626 {  
 627 lcdClear(lcd);  
 628 int count = 27;  
 629 getPerDay(); //retrieve settings  
 630 printf("perDayValue grabed is: %d \n", perDay);  
 631 lcdPosition(lcd, 0, 0);  
 632 lcdPrintf(lcd, "dispenses/day: %d", perDay);  
 633 while(digitalRead(setModeSw) && digitalRead(powerSw) && !digitalRead(resetSw))  
 634 {  
 635   
 636 count--;  
 637 printf("count: %d \n", count);  
 638 lcdPosition(lcd, 15, 0);  
 639 if (count == 26)  
 640 {  
 641 lcdPrintf(lcd, " "); //blink off  
 642 }  
 643 else if(count == 20 || buttonFlag == 1) //blink on  
 644 {  
 645 lcdPrintf(lcd, "%d", perDay);  
 646 buttonFlag =0;  
 647 }  
 648   
 649 if(count == 0)  
 650 {  
 651 count = 27;  
 652 }  
 653 //wait till button is released.  
 654 while((buttonPressed = scanKeypad()) > 0)  
 655 {  
 656 number = numberPressed(buttonPressed);  
 657 if(number >= 0)  
 658 {  
 659 buttonFlag = 1;  
 660 perDay = number;  
 661 }  
 662   
 663 if(buttonPressed == 8)  
 664 {  
 665 setModeStep++;  
 666 break;  
 667 }  
 668   
 669 }  
 670   
 671   
 672 //delay compensation  
 673 if(buttonPressed > 12)  
 674 {  
 675 //printf("button pressed: %d \n", button);  
 676 }  
 677 else if(buttonPressed > 8)  
 678 {  
 679 delay(5);  
 680 }  
 681 else if(buttonPressed > 4)  
 682 {  
 683 delay(10);  
 684 }  
 685 else if(buttonPressed > 0)  
 686 {  
 687 delay(15);  
 688 }  
 689   
 690 if(buttonPressed == 8)  
 691 {  
 692 break;  
 693 }  
 694 }  
 695   
 696 //save settings  
 697 savePerDay();  
 698 getPerDay();  
 699   
 700 return;  
 701 } //end of timesPerDay();  
 702   
 703   
 704 /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
 705 void timeBetweenDisp(void)  
 706 ---------------------------------------------------------------------------------------------  
 707 This funciton provedes the third step in the set mode process and it retrieves the   
 708 time between each dispense from the user and provides feedback through the lcd.   
 709 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
 710 void timeBetweenDisp (void)  
 711 {  
 712 int cursor = 0;  
 713 int incFlag;  
 714 lcdClear(lcd);  
 715 int count = 27;  
 716 getTimeBetween(); // retrieve saved settings  
 717 //timeBetweenHTenths = 1;  
 718 //timeBetweenHOnes = 2;  
 719   
 720 lcdPosition(lcd, 0, 0);  
 721 lcdPrintf(lcd, "Time between");  
 722 lcdPosition(lcd, 0, 1);  
 723 lcdPrintf(lcd, "Disp: %d%d:%d%d hrs", timeBetweenHTenths, timeBetweenHOnes, timeBetweenMTenths, timeBetweenMOnes);  
 724   
 725 // the following process is VERY similar to the timeSet() switch structure to acquire a time input from the user.  
 726 while(digitalRead(setModeSw) && digitalRead(powerSw) && !digitalRead(resetSw))  
 727 {  
 728   
 729 count--;  
 730 //printf("count: %d \n", count);  
 731 switch (cursor)  
 732 {  
 733 case 0:  
 734   
 735 lcdPosition(lcd, 7, 1);  
 736 if (count == 26)  
 737 {  
 738 lcdPrintf(lcd, " ");  
 739 }  
 740 else if((count == 20 || buttonFlag == 1))  
 741 {  
 742 if(buttonFlag == 1 && number >= 0)  
 743 {  
 744 timeBetweenHTenths = number;  
 745 }  
 746   
 747 //cursor conditions begin here  
 748 //cant be 0 hours and 0 minutes between dispenses.  
 749 if(timeBetweenHTenths == 0 && timeBetweenHOnes == 0 && timeBetweenMTenths == 0 && timeBetweenMOnes == 0)  
 750 {  
 751 printf("case 0: both zero ones is now 1 \n");  
 752 timeBetweenMOnes = 1;  
 753 lcdPosition(lcd, 11, 1);  
 754 lcdPrintf(lcd, "%d", timeBetweenMOnes);  
 755 lcdPosition(lcd, 10, 1);  
 756 }  
 757 //should not go beyond 24 hours beween each dispense.   
 758 else if(timeBetweenHTenths > 2)  
 759 {  
 760 printf("case 0: tenths greater than 2. ones is now thenths. \n");  
 761 timeBetweenHOnes = timeBetweenHTenths;  
 762 timeBetweenHTenths = 0;  
 763 lcdPrintf(lcd, "%d", timeBetweenHTenths);  
 764 lcdPosition(lcd, 8, 1);  
 765 lcdPrintf(lcd, "%d", timeBetweenHOnes);  
 766 lcdPosition(lcd, 7, 1);  
 767 }  
 768 else if(timeBetweenHTenths == 2 && timeBetweenHOnes > 4)  
 769 {  
 770 lcdPosition(lcd, 8, 1);  
 771 timeBetweenHOnes = 4;  
 772 lcdPrintf(lcd, "%d", timeBetweenHOnes);  
 773 lcdPosition(lcd, 7, 1);  
 774 }  
 775 {  
 776 lcdPosition(lcd, 7, 1);  
 777 printf("case 0: no adjustments, tenths is 0, 1, or 2.\n");  
 778 lcdPrintf(lcd, "%d", timeBetweenHTenths);  
 779 }  
 780   
 781   
 782   
 783 }  
 784   
 785 if(buttonFlag == 1)  
 786 {  
 787 printf("case 0: button was pressed, moving cursor.\n");  
 788 cursor = 1;  
 789 }  
 790   
 791 if(incFlag == 1)  
 792 {  
 793 printf("case 0: cursor flag is set, moving cursor.\n");  
 794 cursor++;  
 795 incFlag = 0;  
 796 }  
 797 if(decFlag == 1)  
 798 {  
 799 printf("case 0: cursor flag is set, moving cursor.\n");  
 800 cursor = 3;  
 801 decFlag = 0;  
 802 }  
 803 if(cursor != 0)  
 804 {  
 805 printf("case 0: cursor is not on case 0. displaying tenths before exit. tenths:%d Ones:%d\n", timeBetweenHTenths,timeBetweenHOnes);  
 806 lcdPosition(lcd, 7, 1);  
 807 lcdPrintf(lcd, "%d", timeBetweenHTenths);  
 808 }  
 809   
 810 if(count == 0)  
 811 {  
 812 count = 27;  
 813 }  
 814   
 815 buttonFlag = 0;  
 816 break;  
 817   
 818 //for detailed comments, see case 0 in timeSet(). This process is very similar.  
 819 case 1:  
 820 lcdPosition(lcd, 8, 1);  
 821 if (count == 26)  
 822 {  
 823 lcdPrintf(lcd, " ");  
 824 }  
 825 else if((count == 20 || buttonFlag == 1))  
 826 {  
 827 //printf("")  
 828 if(buttonFlag == 1 && number >= 0)  
 829 {  
 830 timeBetweenHOnes = number;  
 831 }  
 832   
 833 //cursor conditions begin here  
 834 printf("case 1: tenths:%d ones:%d \n", timeBetweenHTenths, timeBetweenHOnes);  
 835 if(timeBetweenHTenths == 2 && timeBetweenHOnes > 4)  
 836 {  
 837 printf("case 1: total time greater than 24, adjusting..\n");  
 838 timeBetweenHOnes = 4;  
 839 }  
 840 else  
 841 if(timeBetweenHTenths == 0 && timeBetweenHOnes == 0 && timeBetweenMTenths == 0 && timeBetweenMOnes == 0)  
 842 {  
 843 timeBetweenHOnes = 1;  
 844 }  
 845 else  
 846 {  
 847 lcdPrintf(lcd, "%d", timeBetweenHOnes);  
 848 }  
 849 }  
 850 if(buttonFlag == 1)  
 851 {  
 852 printf("case 0: button was pressed, moving cursor.\n");  
 853 cursor++;  
 854 }  
 855 if(incFlag == 1)  
 856 {  
 857 printf("case 1: cursor flag set, moving cursor.\n");  
 858 cursor++;  
 859 incFlag = 0;  
 860 }  
 861 if(decFlag == 1)  
 862 {  
 863 printf("case 0: cursor flag is set, moving cursor.\n");  
 864 cursor--;  
 865 decFlag = 0;  
 866 }  
 867 if(cursor != 1)  
 868 {  
 869 printf("case 1: cursor is not in case 1. displaying ones before exit.\n");  
 870 lcdPosition(lcd, 8, 1);  
 871 lcdPrintf(lcd, "%d", timeBetweenHOnes);  
 872 }  
 873   
 874 if(count == 0)  
 875 {  
 876 count = 27;  
 877 }  
 878 buttonFlag = 0;  
 879 break;  
 880   
 881 //for detailed comments, see case 0 in timeSet(). This process is very similar.  
 882 case 2:  
 883 lcdPosition(lcd, 10, 1);  
 884 if (count == 26)  
 885 {  
 886 lcdPrintf(lcd, " ");  
 887 }  
 888 else if((count == 20 || buttonFlag == 1))  
 889 {  
 890 if(buttonFlag == 1 && number >= 0)  
 891 {  
 892 timeBetweenMTenths = number;  
 893 }  
 894   
 895 //cursor conditions begin here  
 896 if(timeBetweenMTenths == 0 && timeBetweenMOnes == 0)  
 897 {  
 898 printf("case 0: both zero ones is now 1 \n");  
 899 timeBetweenMOnes = 1;  
 900 lcdPosition(lcd, 11, 1);  
 901 lcdPrintf(lcd, "%d", timeBetweenMOnes);  
 902 lcdPosition(lcd, 10, 1);  
 903 }  
 904 else if(timeBetweenMTenths > 5)  
 905 {  
 906 printf("case 0: tenths greater than 5. ones is now thenths. \n");  
 907 timeBetweenMTenths = 5;  
 908 lcdPrintf(lcd, "%d", timeBetweenMTenths);  
 909 }  
 910 else  
 911 {  
 912 lcdPosition(lcd, 10, 1);  
 913 printf("case 0: no adjustments, tenths is 0, 1, or 2.\n");  
 914 lcdPrintf(lcd, "%d", timeBetweenMTenths);  
 915 }  
 916   
 917 }  
 918   
 919 if(buttonFlag == 1)  
 920 {  
 921 printf("case 0: button was pressed, moving cursor.\n");  
 922 cursor++;  
 923 }  
 924   
 925 if(incFlag == 1)  
 926 {  
 927 printf("case 0: cursor flag is set, moving cursor.\n");  
 928 cursor++;  
 929 incFlag = 0;  
 930 }  
 931 if(decFlag == 1)  
 932 {  
 933 printf("case 0: cursor flag is set, moving cursor.\n");  
 934 cursor--;  
 935 decFlag = 0;  
 936 }  
 937 if(cursor != 2)  
 938 {  
 939 printf("case 0: cursor is not on case 0. displaying tenths before exit. tenths:%d Ones:%d\n", timeBetweenMTenths,timeBetweenMOnes);  
 940 lcdPosition(lcd, 10, 1);  
 941 lcdPrintf(lcd, "%d", timeBetweenMTenths);  
 942 }  
 943   
 944 if(count == 0)  
 945 {  
 946 count = 27;  
 947 }  
 948   
 949 buttonFlag = 0;  
 950 break;  
 951   
 952 //for detailed comments, see case 0 in timeSet(). This process is very similar.  
 953 case 3:  
 954 lcdPosition(lcd, 11, 1);  
 955 if (count == 26)  
 956 {  
 957 lcdPrintf(lcd, " ");  
 958 }  
 959 else if((count == 20 || buttonFlag == 1))  
 960 {  
 961 //printf("")  
 962 if(buttonFlag == 1 && number >= 0)  
 963 {  
 964 timeBetweenMOnes = number;  
 965 }  
 966 //printf("case 1: tenths:%d ones:%d \n", timeBetweenMTenths, timeBetweenMOnes);  
 967   
 968 //cursor conditions begin here  
 969 if(timeBetweenHTenths == 0 && timeBetweenHOnes == 0 && timeBetweenMTenths == 0 && timeBetweenMOnes == 0)  
 970 {  
 971 timeBetweenMOnes = 1;  
 972 }  
 973 else  
 974 {  
 975 lcdPrintf(lcd, "%d", timeBetweenMOnes);  
 976 }  
 977 }  
 978   
 979 if(incFlag == 1)  
 980 {  
 981 printf("case 1: cursor flag set, moving cursor.\n");  
 982 cursor = 0;  
 983 incFlag = 0;  
 984 }  
 985 if(decFlag == 1)  
 986 {  
 987 printf("case 0: cursor flag is set, moving cursor.\n");  
 988 cursor--;  
 989 decFlag = 0;  
 990 }  
 991 if(cursor != 3)  
 992 {  
 993 printf("case 1: cursor is not in case 1. displaying ones before exit.\n");  
 994 lcdPosition(lcd, 11, 1);  
 995 lcdPrintf(lcd, "%d", timeBetweenMOnes);  
 996 }  
 997   
 998 if(count == 0)  
 999 {  
1000 count = 27;  
1001 }  
1002 buttonFlag = 0;  
1003 break;  
1004 }  
1005   
1006   
1007 if(count == 0)  
1008 {  
1009 count = 27;  
1010 }  
1011   
1012 //wait until button is released.   
1013 while((buttonPressed = scanKeypad()) > 0)  
1014 {  
1015 number = numberPressed(buttonPressed);  
1016 if(number >= 0)  
1017 {  
1018 buttonFlag = 1;  
1019 //timeBetweenHTenths = number;  
1020 }  
1021   
1022 if(buttonPressed == 8)  
1023 {  
1024 setModeStep = 0;  
1025 break;  
1026 }  
1027   
1028 if(buttonPressed == 15 )  
1029 {  
1030 incFlag = 1;  
1031   
1032 }  
1033 else if(buttonPressed == 13)  
1034 {  
1035 decFlag = 1;  
1036 }  
1037 }  
1038   
1039   
1040 //dealy adjustment  
1041 if(buttonPressed > 12)  
1042 {  
1043 //printf("button pressed: %d \n", button);  
1044 }  
1045 else if(buttonPressed > 8)  
1046 {  
1047 delay(5);  
1048 }  
1049 else if(buttonPressed > 4)  
1050 {  
1051 delay(10);  
1052 }  
1053 else if(buttonPressed > 0)  
1054 {  
1055 delay(15);  
1056 }  
1057   
1058 if(buttonPressed == 8)  
1059 {  
1060 break;  
1061 }  
1062 }  
1063 //save settings.  
1064 saveTimeBetween();  
1065 getTimeBetween();  
1066   
1067 return;  
1068 } //end of timeBetweenDisp();  
1069   
1070 /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
1071 int numberPressed(int buttonPressed)  
1072 ---------------------------------------------------------------------------------------------  
1073 This function returns the actual number that is represented on the keypads labels.   
1074 if a button other than a number is pressed, then it returns -1.  
1075 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
1076 int numberPressed(int buttonPressed)  
1077 {  
1078 switch(buttonPressed)  
1079 {  
1080 case 1:  
1081 return 1;  
1082 break;  
1083 case 2:  
1084 return 2;  
1085 break;  
1086 case 3:  
1087 return 3;  
1088 break;  
1089 case 5:  
1090 return 4;  
1091 break;  
1092 case 6:  
1093 return 5;  
1094 break;  
1095 case 7:  
1096 return 6;  
1097 break;  
1098 case 9:  
1099 return 7;  
1100 break;  
1101 case 10:  
1102 return 8;  
1103 break;  
1104 case 11:  
1105 return 9;  
1106 break;  
1107 case 14:  
1108 return 0;  
1109 break;  
1110   
1111 default:  
1112 return -1;  
1113 }  
1114 }

1 #include <stdio.h>  
 2 #include <stdlib.h>  
 3 #include <time.h>   
 4 #include <lcd.h>  
 5 #include "finalMain.h"  
 6   
 7 extern int lcd;  
 8 time\_t rawtime;  
 9 struct tm \*timestruct;  
 10   
11 extern int hourTenths;  
 12 extern int hourOnes;  
 13 extern int minTenths;  
 14 extern int minOnes;  
 15 extern char ap;  
 16 extern int perDay;  
 17 extern int timeBetweenHTenths;  
 18 extern int timeBetweenHOnes;  
 19 extern int timeBetweenMTenths;  
 20 extern int timeBetweenMOnes;  
 21   
 22 extern int dispenseFlag;  
 23 int dispenseCount;  
 24 char timeAP;  
 25   
 26 /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
 27 int getTimeH(void)  
 28 ---------------------------------------------------------------------------------------------  
 29 This Function retrieves the current Hour of day in standard time (EST).   
 30 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
 31 int getTimeH(void)  
 32 {   
 33 time(&rawtime); //grab epoch time  
 34 timestruct = gmtime(&rawtime); //format into a struct in gm time  
 35 int hour = timestruct->tm\_hour; //retrieve only the hour from the structure.  
 36   
 37   
 38 hour = hour - 4; //adjust for estern standard time (EST)  
 39 if(hour < 0)  
 40 {  
 41 hour = (24-abs(hour));  
 42 }  
 43   
 44 //hour = 24;  
 45 if(hour < 12 || hour == 24) // determines if this hour was in AM or PM.  
 46 {  
 47 timeAP = 'A';  
 48 }  
 49 else if (hour >= 12)  
 50 {  
 51 timeAP = 'P';  
 52 }  
 53   
 54   
 55 //printf("hour from struct: %d\n", hour);  
 56 hour = hour%12; //convert to standard time from military  
 57   
 58 //after modulo 24, results in 0. 0 is not standered time so adjust.  
 59 if (hour == 0)  
 60 {  
 61 hour = 12;  
 62 }  
 63 return hour;  
 64 }  
 65

66 /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
 67 int getTimeHMilit(void)  
 68 ---------------------------------------------------------------------------------------------  
 69 This Function returns the Hour of the day in military time. Very similar to getTimeH()  
 70 except this funciton doesnt not perform a modulo.   
 71 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
 72 int getTimeHMilit(void)  
 73 {   
 74 time(&rawtime);  
 75 timestruct = gmtime(&rawtime);  
 76 int hour = timestruct->tm\_hour;  
 77   
 78   
 79 hour = hour - 4;  
 80 if(hour < 0)  
 81 {  
 82 hour = (24-abs(hour));  
 83 }  
 84   
 85   
 86 //printf("hour from struct: %d\n", hour);  
 87 return hour;  
 88 }  
 89   
 90 /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
 91 int getTimeM(void)  
 92 ---------------------------------------------------------------------------------------------  
 93 This Function retrieves the current minute of the hour.  
 94 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
 95 int getTimeM(void)  
 96 {  
 97 time(&rawtime);  
 98 timestruct = gmtime(&rawtime);  
 99 int minute = timestruct->tm\_min; // retrieve only the minutes of the struct.   
100 return minute;  
101 }  
102   
103 /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
104 int getTimeS(void)  
105 ---------------------------------------------------------------------------------------------  
106 This Function returieves the current seconds of the minute.   
107 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
108 int getTimeS(void)  
109 {  
110 time(&rawtime);  
111 timestruct = gmtime(&rawtime);  
112 int seconds = timestruct->tm\_sec; //retrieve only the minutes of the struct.  
113 return seconds;  
114 }  
115   
116 int cHour; //Current Hour  
117 int cMin; // Current Minutes   
118 int cSec; // current seconds  
119   
/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
121 void dispenseCheck(void)  
122 ---------------------------------------------------------------------------------------------  
123 This function calculates the dispense schedule for the day and determines which schedule   
124 time to use. If the current time is equal to the apropriate scheduled time of dispense   
125 then call the button pattern function which will execute the dispense sequence.  
126 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
127 void dispenseCheck(void)  
128 {  
129 //retrieve all saved settings.   
130 getHour();  
131 getMin();  
132 getAP();  
133 getTimeBetween();  
134 int sHour; //Sceduled hour  
135 int sMin; //Scheduled minutes  
136 // current seconds  
137 //int timeBetweenH = (timeBetweenHTenths\*10)+timeBetweenHOnes;  
138 //combine the tenths and ones place into one variable.   
139 int timeBetweenM = (timeBetweenMTenths\*10)+timeBetweenMOnes;  
140 int timeBetweenH = (timeBetweenHTenths\*10)+timeBetweenHOnes;  
141 printf("Time Between Minutes: %d\n", timeBetweenM);  
142 int scheduleH[perDay]; //array used to store the hours of the schedule.  
143 int scheduleM[perDay]; //array used to store the minutes of the schedule.  
144   
145 char sAP[perDay]; //array used to store AM/PM  
146   
147 sHour = (hourTenths \* 10) + hourOnes; //combine scheduled hour Tenths and Ones   
148 sMin = (minTenths \* 10) + minOnes; //combine scheduled minute tenths and Ones  
149 //getTimeH();  
150 //retrieve current time:  
151 cHour = getTimeH();   
152 cMin = getTimeM();  
153 cSec = getTimeS();  
154   
155 //convert schedule hour to military time.   
156 if(ap == 'P' && sHour != 12)  
157 {  
158 sHour = sHour + 12;  
159 //printf("if PM adjust sHour: %d\n", sHour);  
160 }  
161   
162 if(perDay > 1)  
163 {  
164 int i;  
165   
166   
167 for(i = 0; i < perDay; i++)  
168 {  
169 //calculate each scheduled dispense using for loop iterator and store into array.  
170 scheduleH[i] = sHour + (i \* timeBetweenH);  
171 scheduleM[i] = sMin + (i \* timeBetweenM);  
172 //adjust minutes and hours if minutes add up beyond 59 minutes.   
173 while(scheduleM[i] >= 60)  
174 {  
175 scheduleH[i]++;  
176 scheduleM[i] = scheduleM[i] -60;  
177 }  
178   
179 printf("Dispense scheduleH: I = %d: Hour: %d Minute: %d\n", i, scheduleH[i], scheduleM[i]);  
180 //save the AM/PM setting for each dispense schedule.   
181 if(scheduleH[i] < 12)  
182 {  
183 sAP[i] = 'A';  
184 }  
185 else  
186 {  
187 scheduleH[i] = scheduleH[i] - 12;  
188 sAP[i] = 'P';  
189 }  
190 //printf("perday is equal to 1. \n");  
191 printf("Current: %d:%d.%d %cM \n", cHour, cMin, cSec, timeAP);  
192   
193 printf("Scheduled: %d:%d %cM \n", scheduleH[i], scheduleM[i], sAP[i]);  
194   
195 //compare current time with scheduled time.   
196 if((scheduleH[i] == cHour) && (scheduleM[i] == cMin) && (sAP[i] == timeAP) && (cSec = getTimeS()) == 0)  
197 {  
198

199 while((cSec = getTimeS()) == 0) //using the current second as a button and waiting until changed/released.  
200 {  
201 dispenseCount = i+1; //incriment count to know when the last dispense is.   
202 foodPattern(); //begin dispense process.   
203 }  
204 }  
205 }  
206   
207 }  
208 //same as above just only dispensing at one time of the day.   
209 else if(perDay <= 1)  
210 {  
211 sHour = sHour % 12;  
212 //printf("perday is equal to 1. \n");  
213 printf("Second current: %d:%d.%d %cM \n", cHour, cMin, cSec, timeAP);  
214   
215 printf("Second Scheduled: %d:%d %cM \n", sHour, sMin, ap);  
216 if((sHour == cHour) && (sMin == cMin) && (ap == timeAP) && ((cSec = getTimeS()) < 5))  
217 {  
218 while((cSec = getTimeS()) < 5)  
219 {  
220 dispenseCount++;  
221 foodPattern();  
222 }  
223 }  
224 }  
225   
226   
227 return;  
228 } //end of dispenseCheck()  
229   
230 /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
231 void updateTimer(void)  
232 ---------------------------------------------------------------------------------------------  
233 This Function updates the count down timer on the 7 segment display.   
234 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
235 void updateTimer(void)  
236 {  
237 //retrieve certain saved settings.   
238 getTimeBetween();   
239 getPerDay();  
240 printf("after getTimeBetween(); timeBetweenHTenths: %d timeBetweenHOnes: %d\n", timeBetweenHTenths, timeBetweenHOnes);  
241 //combine tenths and ones for hours and minutes.   
242 int timeBetweenH = (timeBetweenHTenths\*10)+timeBetweenHOnes;  
243 int timeBetweenM = (timeBetweenMTenths\*10)+timeBetweenMOnes;  
244   
245 //acquire current time.  
246 int currentHrM = getTimeHMilit();  
247 int currentMin = getTimeM();  
248 //determine scheduled time.  
249 int sHour = (hourTenths \* 10) + hourOnes;  
250 int sMin = (minTenths \* 10) + minOnes;  
251 //printf("declaration sHour: %d\n", sHour);  
252   
253 //convert scheduled hour to military time.   
254 if(ap == 'P' && sHour != 12)  
255 {  
256 sHour = sHour + 12;  
257 //printf("if PM adjust sHour: %d\n", sHour);  
258 }  
259 //printf("")  
260   
261 int hourToDisp; // variable that will be displayed  
262 int minToDisp; // variable that will be displayed.   
263 int i;  
264 //temperary variables.   
265 int tempHour;   
266 int tempMin;  
267 printf("BEFORE FOR LOOP perDay = %d\n timeBetweenM = %d", perDay, timeBetweenM);  
268 //dispenseCount = 0;  
269   
270 for(i = 0;i < perDay;i++)  
271 {  
272 printf("INSIDE FOR LOOP\n");  
273   
274 //calculating the next required dispense time.   
275 tempHour = sHour + (i \* timeBetweenH);  
276 tempMin = sMin + (i \* timeBetweenM);  
277   
278 //if the last dispense has passed, count down to first dispense tomorrow.  
279 if(dispenseCount == perDay)   
280 {  
281 hourToDisp = (23 - currentHrM) + sHour;  
282 minToDisp = (60 - currentMin) + sMin;  
283 if(minToDisp >= 60)  
284 {  
285 minToDisp = minToDisp - 60;  
286 hourToDisp++;  
287 }  
288 break;  
289 }  
290   
291 printf("\n tempHour: %d\ntempMin: %d \n", tempHour, tempMin);  
292 /\*  
293 while(sMin >= 60)  
294 {  
295 sHour++;  
296 sMin = sMin -60;  
297 }\*/  
298 //printf("sHour in loop: %d\n", sHour);  
299 //printf("UpdateTimer: sHour= %d\nupdateTimer: currentHrM= %d\nTimeBetweenH: %d\n", sHour, currentHrM, timeBetweenH);  
300 if(tempHour > currentHrM) //if scheduled hour is higher the current hour.  
301 {  
302 //must adjust minutes or hour according to the actual time left.   
303 hourToDisp = ((tempHour - currentHrM));  
304 if(tempMin < currentMin)  
305 {  
306 hourToDisp--;  
307 minToDisp = (60 - currentMin) + tempMin;  
308 }  
309 else  
310 {  
311 hourToDisp--;  
312 minToDisp = tempMin-currentMin;  
313 }  
314   
315 if(minToDisp >= 60)  
316 {  
317 hourToDisp++;  
318 minToDisp = minToDisp - 60;  
319 }  
320 //dispenseCount = i;  
321 break;  
322 }  
323 //if hour is equal to current hour but scheduled minutes is later than current minutes.   
324 else if(tempHour == currentHrM && tempMin >= currentMin)  
325 {  
326 printf("DISPENSE COUNT: %d\n", dispenseCount);  
327 if(tempMin == currentMin && tempHour == currentHrM && dispenseCount != perDay)  
328 {  
329 hourToDisp = timeBetweenH;  
330 //printf("MIN TO DISPLAY WHEN TEMP BEFORE ADJUSTMENT = CURRENT: %d\n", minToDisp);  
331 minToDisp = (tempMin + timeBetweenM)-currentMin;  
332 //printf("MIN TO DISPLAY WHEN TEMP AFTER ADJUSTMENT = CURRENT: %d\n", minToDisp);  
333 }  
334 else  
335 {  
336 hourToDisp = 0;  
337 minToDisp = tempMin - currentMin;  
338 }  
339 //dispenseCount = i;  
340 break;  
341 }  
342   
343   
344   
345 }  
346   
347   
348   
349 //begin display process.  
350   
351   
352 //printf("timerupdate(): Hour to display: %d\n", hourToDisp);  
353 //printf("timerupdate(): Minute to display: %d\n", minToDisp);  
354 printf("timerupdate():schedule to use: %d\n", sHour);  
355 //printf("timerupdate():current military hour: %d\n", currentHrM);  
356   
357 //must parse hours and minutes into tenths and ones to display 1 digit at a time on 7 seg  
358 int hrToDispTenths = 0;  
359 int hrToDispOnes;  
360 while(hourToDisp > 9) //parsing hours  
361 {  
362 hrToDispTenths++;  
363 hourToDisp = hourToDisp - 10;  
364 }  
365 hrToDispOnes = hourToDisp;  
366   
367 int minToDispTenths = 0;  
368 int minToDispOnes;  
369 while(minToDisp > 9) //parsing minutes.  
370 {  
371 minToDispTenths++;  
372 minToDisp = minToDisp - 10;  
373 }  
374 minToDispOnes = minToDisp;  
375   
376 //display on 7 segment.   
377 segWrite(0, hrToDispTenths);  
378 segWrite(1, hrToDispOnes);  
379 segWrite(2, minToDispTenths);  
380 segWrite(3, minToDispOnes);  
381   
382 return;  
383 }  
384   
385 /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
386 void updateLcdTime(void)  
387 ---------------------------------------------------------------------------------------------  
388 This Function updates the current time on the first line of the lcd.   
389 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
390 void updateLcdTime(void)  
391 {  
392 lcdPosition(lcd, 10, 0);  
393 lcdPrintf(lcd, " ");  
394 lcdPosition(lcd, 0, 0);  
395 lcdPrintf(lcd, " %d:%d.%d %cM", cHour, cMin, cSec, timeAP);  
396 }  
397   
  
399   
400

1 /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
 2 motorTest2.c  
 3 ---------------------------------------------------------------------------------------------  
 4 Group Name: Automatic Pet Feeder  
 5 Group Members: Raul Rojas,Isrrael Mendoza,Nyron Daniel  
 6   
 7 By: Raul Rojas  
 8 This program demonstrates the funcitonality of our servo motor and its functions which   
 9 will be incorperated in our finished project. The demo of this project will perform   
 10 short dispenses that allow the hall effect sensor to keep track of the position of the   
 11 motor. This is crucial so that the program will know when the motor is stalled and the  
 12 motor will perform a maneuver in an attempt to unjam whatever was causing it to be   
 13 stuck.   
 14   
 15 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
 16   
 17 #include <stdio.h>  
 18 #include <wiringPi.h>  
 19 #include <stdlib.h>  
 20 #include <time.h>  
 21 #include "finalMain.h"  
 22   
 23 //declaring variables needed for global use in file.  
 24 int motorPosition = 0; //Used to store the value of the current position of the motor.  
 25 int previousPosition; //Used to store the value of the previous position of the motor.  
 26 int stallFlag; //A flag which indicates if the motor is stalled.   
 27 extern int dispenseFlag;  
 28 extern int petModeFlag;  
 29   
 30 //main used for testing of functions  
 31   
 79 return 0;  
 80 }  
 81 \*/  
 82   
 83   
 84 /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
 85 void motorMove(int value)  
 86 ---------------------------------------------------------------------------------------------  
 87 By: Raul Rojas  
 88   
 89 This function is the core of the program. It is available to other files in case  
 90 the motor needs to be positioned to a specific location. Using custom software pulse  
 91 width modulation to apply a signal to the motor which corresponds to the position   
 92 desired. The motor itself can only move about 120 degrees thus limiting the input value  
 93 from around 650 - 2200. These values correspond to the HIGH time of the pulse. This   
 94 function then adjust the low time to fit a 20ms window (50Hz).   
 95   
96 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
 97 void motorMove(int value)  
 98 {  
 99 int i;  
100 for(i = 0; i < 10 ; i++) //Loop to repeat the desired PWM signal for a set time.  
101 {  
102 digitalWrite(motorSig, HIGH); //sets the signal pin HIGH   
103 delayMicroseconds(value); //delay time for the HIGH signal   
104 digitalWrite(motorSig, LOW); //Clears the signal pin   
105 delayMicroseconds(20000-value); //adjusts LOW time so the frequency is 50Hz(20ms).  
106 }  
107 return;  
108 }  
109

110 /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
111 void motorDispense(void)  
112 ---------------------------------------------------------------------------------------------  
113 By: Raul Rojas  
114   
115 This function provides a smart method of dispensing. It attempts to determine the current  
116 position of the motor and then moves the motor accordingly. This is important becuase   
117 since the motor can only rotate 120 degrees, it must know what position the motor is in   
118 to rotate in the right direction. It also uses the hall effect sensor to detect whether   
119 the motor has been stalled after each movement. To accomplish this, the motor must move   
120 in 90 degree increments whilst checking the hall effect sensor between those movements  
121 to make sure the motor actually moved.   
122   
123 note: The hall effect sensor is positioned at 90 degrees. Therefore, when the magnet is   
124 at 0 or 180 (120 actual) degrees, the hall effect sensor should not detect a magnet.   
125 For convention, consider 650 as 0 degrees, 1500 as 90 degrees, and 2200 as 180 degrees.  
126   
127 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
128 void motorDispense(void)  
129 {  
130 int count = 0; //count which controls the amount of desired dispenses.   
131   
132 //On boot motorPosition is 0 that way the program can attempt to determine position.  
133 if(motorPosition == 0)  
134 {  
135 if(!digitalRead(hes)) //Is the motor position in 90 degrees?  
136 {  
137 motorPosition = 1501;   
138 }  
139 else  
140 {  
141 motorMove(1501); //if not at 90 deg, move to it as a starting position.   
142 delay(100);  
143 motorPosition = 1501;  
144 count++; //This will count as a dispense.   
145 printf("count: %d \n", count);  
146   
147 }  
148 }  
149 while (count < 2) //dispense only twice.  
150 {  
151 delay(500); //pause between dispenses.  
152   
153 //The following Switch block rotates the motor according to its previous positions.  
154 switch (motorPosition)  
155 {  
156 case 650:  
157 motorMove(1499); //use 1499 as an indicator that it moved from 650.  
158 motorPosition = 1499; //store its known position.  
159 previousPosition = 650; //store its previously known position.  
160 delay(100);  
161 if(digitalRead(hes)) //check if motor is stalled.   
162 {  
163 stallFlag = 1; //motor is stalled, set flag.  
164 printf("motor is stalled. \n");  
165 motorPosition = previousPosition; //if stalled, motor didnt move.  
166 }  
167 else  
168 {  
169 count++; //if not stalled, it counts as a successful dispense.  
170 }  
171 printf("count: %d \n", count);  
172 break;  
173   
174 case 1499: //indicates the previous position was at 650.  
175 motorMove(2200); //rotate motor so it continues its rotation of 180 deg.  
176 motorPosition = 2200;  
177 previousPosition = 1499;  
178 delay(100);  
179 if(!digitalRead(hes))  
180 {  
181 stallFlag = 1;  
182 printf("motor is stalled. \n");  
183 motorPosition = previousPosition;  
184 }  
185 else  
186 {  
187 count++;  
188 }  
189   
190 printf("count: %d \n", count);  
191 break;  
192   
193 case 1500: //this case only possible if using the manual input method of demo.  
194 motorMove(650);  
195 motorPosition = 650;  
196 previousPosition = 1500;  
197 delay(100);  
198 if(!digitalRead(hes))  
199 {  
200 stallFlag = 1;  
201 printf("motor is stalled. \n");  
202 motorPosition = previousPosition;  
203 }  
204 else  
205 {  
206 count++;  
207 }  
208   
209 printf("count: %d \n", count);  
210 break;  
211   
212 case 1501: //indicates that previous position of motor was 2200.  
213 motorMove(650);  
214 motorPosition = 650;  
215 previousPosition = 1501;  
216 delay(100);  
217 if(!digitalRead(hes))  
218 {  
219 stallFlag = 1;  
220 printf("motor is stalled. \n");  
221 motorPosition = previousPosition;  
222 }  
223 else  
224 {  
225 count++;  
226 }  
227   
228 printf("count: %d \n", count);  
229 break;  
230   
231 case 2200:  
232 motorMove(1501);  
233 motorPosition = 1501;  
234 previousPosition = 2200;  
235 delay(100);  
236 if(digitalRead(hes))  
237 {  
238 stallFlag = 1;  
239 printf("motor is stalled. \n");  
240 motorPosition = previousPosition;  
241 }  
242 else  
243 {  
244 count++;  
245 }  
246   
247 printf("count: %d \n", count);  
248 break;  
249   
250 default: //only used in the manual input method of motor demo.   
251 motorMove(1501);  
252 motorPosition = 1501;  
253 previousPosition = 1501;  
254 delay(100);  
255 if(digitalRead(hes))  
256 {  
257 stallFlag = 1;  
258 printf("motor is stalled. \n");  
259 motorPosition = previousPosition;  
260 }  
261 else  
262 {  
263 count++;  
264 }  
265   
266 printf("count: %d \n", count);  
267   
268 }  
269 delay(500); //delay time between dispenses.  
270   
271 //if motor is position at 90 deg, turn off led. else, keep it on.   
272 if(digitalRead(hes)==0)   
273 {  
274 digitalWrite(motorStatLed,0);  
275 }  
276 else  
277 {  
278 digitalWrite(motorStatLed,1);  
279 }  
280   
281 if(stallFlag)  
282 {  
283 motorUnstall();  
284 }  
285 }  
286 }  
287   
288   
289 /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
290 void motorUnstall(void)  
291 ---------------------------------------------------------------------------------------------  
292 By: Raul Rojas  
293   
294 This short function performs a short maneuver to attempt to remove any thing that is   
295 causing the motor to be stuck. It rotates back and forth quickly for a self fix solution.   
296   
297 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
298 void motorUnstall(void)  
299 {  
300 printf("unstalling.. \n");  
301 delay(500);  
302 int stall; //count for loop.  
303   
304 for(stall = 0; stall < 2; stall++)  
305 {  
306 motorMove(motorPosition + 300); //rotate motor slightly one way from current position  
307 delay(20);  
308 motorMove(motorPosition - 300); //rotate slightly other way from original position  
309 delay(20);  
310 motorMove(motorPosition); //return to original position.  
311 }  
312 stallFlag = 0; //release flag.  
313 }  
314   
315 void vibMove(void)  
316 {  
317 printf("Entered vibMove.");  
318 motorPosition = 650;  
319 motorMove(motorPosition);  
320 delay(1000);  
321 int adder = 700;  
322 int subtr = 600;  
323 //while(motorPosition < 2100);  
324 int i;  
325   
326 int modeAmount;  
327 if(petModeFlag)  
328 {  
329 modeAmount = 120;  
330 }  
331 else  
332 {  
333 modeAmount = 60;  
334 }  
335   
336 while(updateWeight() < modeAmount)  
337 {  
338 for(i=0; i<50;i++)  
339 {  
340 if(updateWeight() >= modeAmount)  
341 {  
342 break;  
343 }  
344 printf("%d \n", i);  
345 //delay(50);  
346 motorMove(motorPosition += adder);  
347 printf("%d \n", motorPosition);  
348 if(motorPosition > 2200)  
349 {  
350 break;  
351 }  
352 //delay(50);  
353 motorMove(motorPosition -= subtr);  
354 }  
355   
356 if(updateWeight() >= modeAmount)  
357 {  
358 dispenseFlag = 0;  
359 return;  
360 }  
361   
362 for(i=0; i<50;i++)  
363 {  
364 if(updateWeight() >= modeAmount)  
365 {  
366 break;  
367 }  
368 printf("%d \n", i);  
369 //delay(50);  
370 motorMove(motorPosition -= adder);  
371 printf("%d \n", motorPosition);  
372 if(motorPosition <650)  
373 {  
374 break;  
375 }  
376 //delay(50);  
377 motorMove(motorPosition += subtr);  
378 }  
379 }  
380 dispenseFlag = 0;  
381 return;  
382 }  
383

1 #include <wiringPi.h>  
 2 #include <stdio.h>  
 3 #include <stdlib.h>  
 4 #include <lcd.h>  
 5 #include "finalMain.h"  
 6   
 7 extern int lcd;  
 8 int weight;  
 9 /\*  
 10 int main(void)  
 11 {  
 12 hx711Setup();  
 13   
 14 while (1)  
 15 {  
 16 updateWeight();  
 17 delay(20);  
 18   
 19 }  
 20 return 0;  
 21 }  
 22 \*/  
 23 /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
 24 HX711 Setup Function  
 25 by Isrrael Mendoza  
 26   
 27 This function is used to intialize the Raspberry Pie and the hx711. Sets up pin modes  
 28 and puts hx711 in normal operating conditions.  
 29   
 30 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
 31 void hx711Setup(void)  
 32 {  
 33 wiringPiSetupGpio();  
 34 pinMode (DOUT,INPUT); //setting up pin modes  
 35 pinMode (PD\_SCK, OUTPUT);  
 36 digitalWrite (PD\_SCK, LOW); //setting hx711 in normal mode  
 37 }  
 38 /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
 39 BitBang Function   
 40 by Isrrael Mendoza   
 41   
 42 This function is used to retrieve the 24 bit output from the hx711. The "Bit Banging"  
 43 method is used to obtain bits. Once Dout goes low the hx711 is ready for data is ready  
 44 for retrieval. While Dout is low we will pulse PD\_SCK and with every pulse a bit shifted  
 45 our from the dout pin. Repeat 24 times to obatin our 24 bits and then pulse the PD\_SCK   
 46 pin once more to select channel A with a gain of 128.  
 47   
 48 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
 49 //long startTime;  
 50 //long timeOut;  
 51 //int storedData = 0;  
 52   
 53 int offset=-10;  
 54 int getMass() //bit banging will be implemented to  
 55 { //retrieve data  
 56 printf("inside getMass().\n");  
 57 int i;  
 58 int data=0;  
 59 //startTime = micros();  
 60 while (digitalRead(DOUT)); //waits until hx711 is ready for data retrieval  
 61   
 62 /\*  
 63 timeOut = micros() - startTime;  
 64 if(timeOut >= 100000)  
 65 {  
 66 return storedData;  
 67 }  
 68 \*/  
 69   
 70   
 71 for (i=0;i<25;i++)  
 72 { //Pulse PD\_SCK to shift bits one by one, starting with  
 73   
 74 digitalWrite(PD\_SCK, HIGH); //MSB  
 75 data = ((data<<1)|digitalRead(DOUT)); //store bits into data variable  
 76 digitalWrite(PD\_SCK, LOW);  
 77 }  
 78   
 79 if (data >= 0x800000)   
 80 data= (data|0xFF000000);  
 81   
 82   
 83 data =(0.001196359\*data)+offset;  
 84 data = (data \*2.127659574);  
 85   
 86 //printf("data: %d", data);  
 87 //if((data-storedData) > 100 || (storedData - data) > 100)  
 88 //{  
 89 //data = storedData;  
 90 //}  
 91 printf("exit getMass(). data: %d \n", data);  
 92 //storedData = data;  
 93 return data; //bit signed inter  
 94 }  
 95   
 96 /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
 97 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
 98   
 99 int tare()  
100 {  
101 offset = (getMass()-offset); //Tares the balance   
102   
103 return offset;  
104 }  
105 //int storedWeight = 0;  
106   
107 /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
108 int updateWeight(void)  
109 ---------------------------------------------------------------------------------------------  
110 This function quickly grabs the current weight of the bowl and updates it on the lcd  
111 as well as return the weight value from where it was called.   
112 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
113 int updateWeight(void)  
114 {   
115   
116 weight = getMass();  
117   
118   
119 //position lcd curser to a position which will not interfere with the constant characters  
120 lcdPosition(lcd, 10,1);  
121   
122 //clear any existing characters in the data section of the first line of the LCD.  
123 lcdPrintf(lcd, " ");   
124   
125 //Reposition LCD curser  
126 lcdPosition(lcd, 10,1);  
127 //printf("Weight: %d g\n", weight);  
128   
129 //Print data into grams.  
130 lcdPrintf(lcd, "%d", weight);  
131   
132 return weight;  
133 }  
134   
135 int avgWeight(void)  
136 {  
137 int i;  
138   
139 int measure1 = 0;  
140 int measure2 = 0;  
141 int measure3 = 0;  
142   
143 weight = 0;  
144 // call function which reads and converts the data into g's. Funciton located in hx711.c  
145 for(i = 0; i < 3; i++)  
146 {  
147 switch (i)  
148 {  
149 //delay(5);  
150 case 0:  
151 measure1 = getMass();  
152 break;  
153   
154 case 1:   
155 measure2 = getMass();  
156 break;  
157   
158 case 2:  
159 measure3 = getMass();  
160 break;  
161 }  
162 }  
163   
164 if(((measure1 > measure2) && (measure1 - measure2) < 50) || ((measure1 < measure2) && (measure1 - measure2) > -50))  
165 {  
166 return ((measure1+measure2)/2);  
167 }  
168 else if(((measure1 > measure3) && (measure1 - measure3) < 50) || ((measure1 < measure3) && (measure1 - measure3) > -50))  
169 {  
170 return ((measure1 + measure3)/2);  
171 }  
172 else if(((measure2 > measure3) && (measure2 - measure3) < 50) || ((measure2 < measure3) && (measure2 - measure3) > -50))  
173 {  
174 return ((measure2 + measure2)/2);  
175 }  
176 else   
177 {  
178 return -999;  
179 }  
180 }

1 /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
 2 \* rangingMod.c  
 3 \* Group 4: Automatic Pet Feeder  
 4 \* Group Members: Raul Rojas, Nyron Daniel & Isrreal Mendoza  
 5 \*   
 6 \* This program when executed will use an ultrasonic Ranging Module (HC-SR04) to find the  
 7 \* distance of the furthest object in centimeters. This program will be used to detect the  
 8 \* food level of our Pet Feeder.   
 9 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
 10 #include <stdio.h>  
 11 #include <wiringPi.h>  
 12 #include <stdlib.h>  
 13 #include <lcd.h>  
 14 #include "finalMain.h"  
 15   
 16 //#define TRIGPIN 4  
 17 //#define ECHOPIN 6  
 18   
 19 //int getCM();  
 20 void rangingModSetup();  
 21 long travelTime;  
 22 void callTest(void);  
 23 long startTime;  
 24 extern int lcd;  
 25 /\*  
 26 int main (void)  
 27 { //Calling Ultrasonic Ranging Module Function.  
 28 rangingModSetup();  
 29   
 30 printf("Distance: %dcm\n", getCM());  
 31 return 0;  
 32 }  
 33 \*/  
 34   
 35 void rangingModSetup ()   
 36 {  
 37 wiringPiSetupGpio();  
 38 pinMode(TRIGPIN, OUTPUT);  
 39 pinMode(ECHOPIN, INPUT);  
 40   
 41 //TRIGPIN must start low in order to get a clear signal.  
 42 digitalWrite(TRIGPIN, LOW);  
 43 delay(30);  
 44 }  
 45   
 46 int getCM()  
 47 {  
 48 printf("in getCM\n");  
 49 //Sending Trigger pulse.  
 50 printf("sending trigpin signal.\n");  
 51 digitalWrite(TRIGPIN, HIGH);  
 52 printf("trigpin high.\n");  
 53 delayMicroseconds(20);  
 54 digitalWrite(TRIGPIN, LOW);  
 55 printf("trigpin low. trigger signal sent.\nStarting wait. \nwaiting for signal. \n");  
 56 //printf("starting wait.");  
 57 //printf("waiting for signal\n");  
 58   
 59 startTime = micros();  
 60   
 61 callTest();  
 62   
 63 //Waiting for ECHOPIN to end.  
 64 startTime = micros();  
 65   
 66 while (digitalRead(ECHOPIN) == HIGH);  
 67   
 68 travelTime = micros() - startTime;  
 69 //Attaining the distance in centimeters.  
 70 int distance = travelTime / 58;  
 71   
 72 printf("exit getCM. Distance: %d", distance);  
 73 return distance;  
 74 }  
 75   
 76 void callTest(void)  
 77 {  
 78   
 79 while (digitalRead(ECHOPIN) == LOW)  
 80 {  
 81 printf("test");  
 82 travelTime = micros() - startTime;  
 83 if(travelTime > 500)  
 84 {  
 85 break;  
 86 }  
 87 }  
 88 return;  
 89 }  
 90   
 91 void foodCheck(void)  
 92 {  
 93 int distance = getCM();  
 94 if(distance >= 12)  
 95 {  
 96 digitalWrite(foodStatLed, LOW);  
 97 }  
 98 else  
 99 {  
100 digitalWrite(foodStatLed, HIGH);  
101 }  
102   
103 }

Schematics



Figure 10 – Power Distribution Schematic



Figure 11 – Breakout Schematic



Figure 12 – Keypad Schematic



Figure 13 – Back Panel Switches Schematic



Figure 14 – 7 Segment Schematic



Figure 15 – Ranging Module Schematic



Figure 16 – LCD/MCP23017 Schematic



Figure 17 – Amplifier Board (Hx711) Schematic



Figure 18 – Hall Effect Schematic



Figure 19 – Servo Schematic

Block Diagram

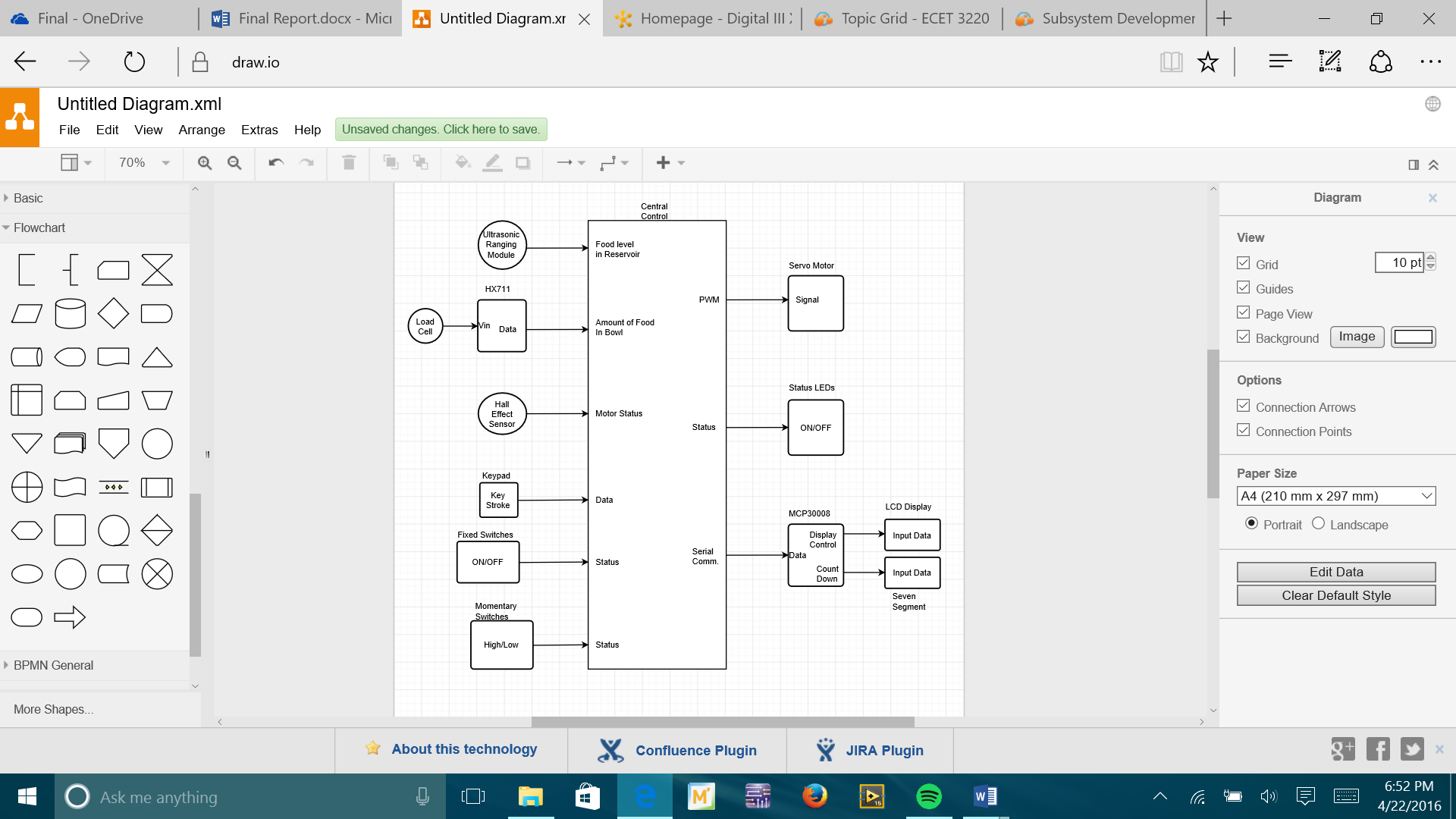


Figure 20: Automated Pet Feeder Block Diagram

Subsystem Interface Diagram

Table 2: Power Distribution Board

|  |  |  |  |
| --- | --- | --- | --- |
| **Power Distribution Board** | | | |
| Connector # | Pin # | Function | Description |
| J1 – Micro USB Adapter | Pin 1 | Vcc | Provides the +5 V dc our system needs to be functional. As well as a common ground |
| Pin 2 | Rx | NC |
| Pin 3 | Tx | NC |
| Pin 4 | NC | NC |
| Pin 5 | GND | Ground |
| J2 - USB Adapter | Pin 1 | USB / Vcc | Provides the +5 V dc that the pie need to be operational |
| Pin 2 | USB / Data- | NC |
| Pin 3 | USB / Data+ | NC |
| Pin 4 | USB / GND | Provides ground to the Pie |
| J3 – 3.3V Screw Terminal Block | Pin 1 | Seven Segment Board | Provides +3.3 V dc to the Status LEDs on the front. The two Status LEDs are: Food and Motor Status |
| Pin 2 | Seven Segment | +3.3 V dc / Vcc |
| Pin 3 | Green LED | +3.3 V dc / Vcc |
| Pin 4 | Red LED | +3.3 V dc / Vcc |
| Pin 5 | Blue LED | +3.3 V dc / Vcc |
| Pin 6 | Dog Petal 1 | +3.3 V dc / Vcc |
| Pin 7 | Dog Petal 2 | +3.3 V dc / Vcc |
| Pin 8 | Dog Petal 3 | +3.3 V dc / Vcc |
| Pin 9 | SwitchBoard | Provides +3.3 V dc to the Switch Board. This board consist of three Switches: Power, Set Mode, and Pet Mode |
| Pin 10 | Reset Switch | +3.3V dc / Vcc |
| Pin 11 | HX711 | +3.3 V dc / Vcc |
| Pin 12 | MCP3008 | +3.3 V dc / Vcc |
| Pin 13 | Hall Effect Sensor | +3.3V dc / Vcc |
| J4 – 5V Spring Terminal Block | Pin 1 | Servo Motor | +5 V dc / Vcc |
| Pin 2 | LCD | +5 V dc / Vcc |
| Pin 3 | Ranging Module | +5 V dc / Vcc |
| Pin 4 | Hall Effect Sensor | +5 V dc / Vcc |
|  | Pin 1 | Hx711 | Ground |
| Pin 2 | Seven Segment | Ground |
| Pin 3 | Hall Effect Sensor | Ground |
| Pin 4 | LCD Board | Provides ground to two components on the LCD board. These two components are: The MCP3008 and the LCD |
| Pin 5 | Reset Switch | Ground |
| Pin 6 | Ranging Module | Ground |
| Pin 7 | Servo Motor | Ground |

Table 3: Raspberry Pi

|  |  |  |  |
| --- | --- | --- | --- |
| **Raspberry Pie** | | | |
| Connector # | Pin # | Function | Description |
| J1 - Micro USB 2.0 Type B Connector | Pin 1 | RPi Vcc | +5 V dc |
| Pin 2 | RPi D- | NC |
| Pin 3 | RPi D+ | NC |
| Pin 4 | RPi ID | NC |
| Pin 5 | RPi GND | Ground |
| J2 - T- Cobbler Breakout | Pin 1 | 3.3 V | NC |
| Pin 2 | 5 V | NC |
| Pin 3 | GPIO 2 | SDA: Seven Segment Display / MCP3008 |
| Pin 4 | 5 V | NC |
| Pin 5 | GPIO 3 | SCL: Seven Segment Display / MCP3008 |
| Pin 6 | GND | NC |
| Pin 7 | GPIO 4 | Ranging Module Trigger / Pin 3 / Boolean Logic |
| Pin 8 | GPIO 14 | Motor Status LED / Boolean Logic |
| Pin 9 | GND | NC |
| Pin 10 | GPIO 15 | Food Status LED / Boolean Logic |
| Pin 11 | GPIO 17 | First Dog Petal (Red) / Boolean Logic / Active Low |
| Pin 12 | GPIO 18 | Servo Motor Pulse / Pin 3 / PWM |
| Pin 13 | GPIO 27 | Second Dog Petal (Blue) / Boolean Logic / Active Low |
| Pin 14 | GND | NC |
| Pin 15 | GPIO 22 | Third Dog Petal (Green) / Boolean; Active Low |
| Pin 16 | GPIO 23 | Reset Switch / Boolean / Active High |
| Pin 17 | 3.3 V | NC |
| Pin 18 | GPIO 24 | Power Switch / Boolean / Active Low |
| Pin 19 | GPIO 10 | Keypad Column 2 / Boolean / Active Low |
| Pin 20 | GND | NC |
| Pin 21 | GPIO 9 | Keypad Column 4 / Boolean / Active Low |
| Pin 22 | GPIO 25 | Set Mode Switch / Boolean / Active Low |
| Pin 23 | GPIO 11 | NC |
| Pin 24 | GPIO 8 | NC |
| Pin 25 | GND | NC |
| Pin 26 | GPIO 7 | Hx711 CLK / Pin 2/ Serial Data |
| Pin 27 | ID SD | NC |
| Pin 28 | ID SC | NC |
| Pin 29 | GPIO 5 | Pet Mode Switch / Pin 1 / Boolean / Active Low |
| Pin 30 | GND | NC |
| Pin 31 | GPIO 6 | Ranging Module Echo / Pin 2 / Boolean Logic |
| Pin 32 | GPIO 12 | HX711 DT / Pin 3/ Serial Data |
| Pin 33 | GPIO 13 | Hall Effect Sensor / Pin 2 / Boolean Logic |
| Pin 34 | GND | NC |
| Pin 35 | GPIO 19 | Keypad Column 3 / Boolean / Active Low |
| Pin 36 | GPIO 16 | Red LED / Boolean / Active Low |
| Pin 37 | GPIO 26 | Blue LED / Pin 2 / Boolean / Active Low |
| Pin 38 | GPIO 20 | Green LED / Pin 2 / Boolean / Active Low |
| Pin 39 | GND | NC |
| Pin 40 | GPIO 21 | Keypad Column 1 / Boolean / Active Low |

Table 4: Breakout Board

|  |  |  |  |
| --- | --- | --- | --- |
| **Breakout Board** | | | |
| Connector # | Pin # | Function | Description |
| J1 - T- Cobbler Breakout |  | Refer to J2 on RPi Board |  |
| J2 – 13 Pin Spring Terminal | Pin 1 | GPIO 17 | First Dog Pedal (Red) |
| Pin 2 | GPIO 27 | Second Dog Pedal (Blue) |
| Pin 3 | GPIO 22 | Third Dog Pedal (Green) |
| Pin 4 | GPIO 26 | Red LED |
| Pin 5 | GPIO 16 | Blue LED |
| Pin 6 | GPIO 20 | Green LED |
| Pin 7 | GPIO 24 | Power Switch |
| Pin 8 | GPIO 5 | Pet Mode Switch |
| Pin 9 | GPIO 25 | Set Mode Switch |
| Pin 10 | GPIO 10 | Keypad Column 2 |
| Pin 11 | GPIO 19 | Keypad Column 1 |
| Pin 12 | GPIO 23 | Reset Switch |
| Pin 13 | GPIO 14 | Motor Status LED |
| J3 – 13 Pin Spring Terminal | Pin 1 | GPIO 15 | Food Status LED |
| Pin 2 | GPIO 9 | Keypad Column 1 |
| Pin 3 | GPIO 21 | Keypad Column 4 |
| Pin 4 | GPIO 12 | HX711 Data |
| Pin 5 | GPIO 7 | HX711 Clock |
| Pin 6 | GPIO 13 | Hall effect Sensor |
| Pin 7 | GPIO 4 | Ranging Trigger |
| Pin 8 | GPIO 6 | Ranging Echo |
| Pin 9 | GPIO 18 | Servo Motor Signal |
| Pin 10 | GPIO 3 | Seven Segment I2C Clock |
| Pin 11 | GPIO 2 | Seven Segment I2C Data |
| Pin 12 | GPIO 3 | LCD I2C Clock |
| Pin 13 | GPIO 2 | LCD I2C Data |

Table 5: Keypad Board

|  |  |  |  |
| --- | --- | --- | --- |
| **Keypad Board** | | | |
| Connector # | Pin # | Function | Description |
| J1 – 8 Pin Spring Terminal | Pin 1 | Row 1 | Active Low |
| Pin 2 | Row 2 | Active Low |
| Pin 3 | Row 3 | Active Low |
| Pin 4 | Row 4 | Active Low |
| Pin 5 | Column 1 | Active Low |
| Pin 6 | Column 2 | Active Low |
| Pin 7 | Column 3 | Active Low |
| Pin 8 | Column 4 | Active Low |

|  |  |  |  |
| --- | --- | --- | --- |
| **Back Panel SwitchBoard** | | | |
| Connector # | Pin # | Function | Description |
| J1 – 5 Pin Spring Terminals | Pin 1 | Vcc | 3.3 Vdc |
| Pin 2 | GND | Ground |
| Pin 3 | Power Switch | Must be high to be functioning. If low, then the system has power but will not perform functionalities / Active Low |
| Pin 4 | Pet Mode Switch | Two Pet Modes: Cat Mode and Dog Mode. When it is time to dispense food it will dispense an amount respectively to the mode / Active Low |
| Pin 5 | Set Mode Switch | Allows the user to set the desired dispense schedule / Active Low |

Table 6: Back Panel SwitchBoard

|  |  |  |  |
| --- | --- | --- | --- |
| **7 Segment/LEDs Board** | | | |
| Connector # | Pin # | Function | Description |
| J1 – 6 pin Spring Terminal | Pin 1 | Vcc | 3.3 V |
| Pin 2 | Food Status LED | Notifies the user that the food level is getting too low. Will blink when the food level reaches 20% and stay on when it reaches 10% |
| Pin 3 | Motor Status LED | Will notify the user when the motor is stalled. After the RPi senses the Motor is stuck it will attempt to un-stall itself. If unsuccessful LED will turn on |
| Pin 4 | GND | Ground |
| Pin 5 | Serial Clock | Provides the Clock signal to the Seven Segment, that is needed for I2C protocol |
| Pin 6 | Serial Data | Provides the Data Signal to the Seven Segment, that is needed for I2C protocol |

Table 7 : 7 Segment/LEDs Board

|  |  |  |  |
| --- | --- | --- | --- |
| **Ranging Module Board** | | | |
|  | Pin # | Function | Description |
| J1 - 4 pin Spring Terminal Block | Pin 1 | Vcc | 5 V dc |
| Pin 2 | Trigger | When pulsed, Ranging Module produces a sound wave and pulls Echo Pin high |
| Pin 3 | Echo | Once Echo Pin is Pulled high, it stays high until the soundwave returns. Distance is calculated accordingly with the time it stayed high |
| Pin 4 | GND | Ground |

Table 8: Ranging Module Board

Table 9: LCD/ MCP23017 Board

|  |  |  |  |
| --- | --- | --- | --- |
| **LCD/ MCP23017 Board** | | | |
| Connecter # | Pin # | Function | Description |
| J1 - 5 Pin Spring Terminal | Pin 1 | Vcc (5V) | Provides the LCD Vcc |
| Pin 2 | GND | Provides ground to both the LCD and MCP23017 |
| Pin 3 | Vcc (3.3V) | Provides the MCP23017 Vcc |
| Pin 4 | Serial Data | MCP23017 Data pin since it uses I2C protocol |
| Pin 5 | Serial Clock | MCP23017 Clock Pin for I2C Protocol |
| J2 – 4 Pin Spring Terminal | Pin 1 | Keypad Row 1 | Active Low / Boolean Logic |
| Pin 2 | Keypad Row 2 | Active Low / Boolean Logic |
| Pin 3 | Keypad Row 3 | Active Low / Boolean Logic |
| Pin 4 | Keypad Row 4 | Active Low / Boolean Logic |

|  |  |  |  |
| --- | --- | --- | --- |
| **Amplifier Board** | | | |
| Connector # | Pin # | Function | Description |
| J1 - 4 pin Spring Terminal | Pin 1 | Vcc | Supplies the 5 V Vcc for the HX711 |
| Pin 2 | Do | Data Pin used to retrieve measurements. Data is retrieved with bit banging method |
| Pin 3 | PD\_CLK | PD\_CLK pin also used to in the retrieval of data. When puled it shifts one bit at a time through Do Pin |
| Pin 4 | GND | Ground |
| J2 – 4 pin Screw Terminal | Pin 1 | E+ | Excitation (+) voltage. The voltage applied to the input terminals of the Load Cell |
| Pin 2 | E- | Excitation (-) voltage. The voltage applied to the input terminals of the Load Cell |
| Pin 3 | A+ | Channel A (+) input. Since HX711 has to input, we’re choosing the programmable input |
| Pin 4 | A- | Channel A (+) input. Since HX711 has to input, we’re choosing the programmable input |

Table 10: Load Cell Amplifier (Hx711) Board

|  |  |  |  |
| --- | --- | --- | --- |
| **Hall Effect Board** | | | |
| Connector # | Pin # | Function | Description |
| J1 - 4 | 1 | Vcc (5V) | Provides the Vcc for the Hall Effect Sensor |
| 2 | GND | Ground |
| 3 | Signal | Hall effect sensor detects if there is a magnetic field near it. Will be used to notify the user if the motor is stalled |
| 4 | Vcc (3.3V) | 3.3 V is used to make the Hall effect active low. When there is no magnetic field the output signal is high |

Table 11: Hall Effect Board

Wiring Diagram

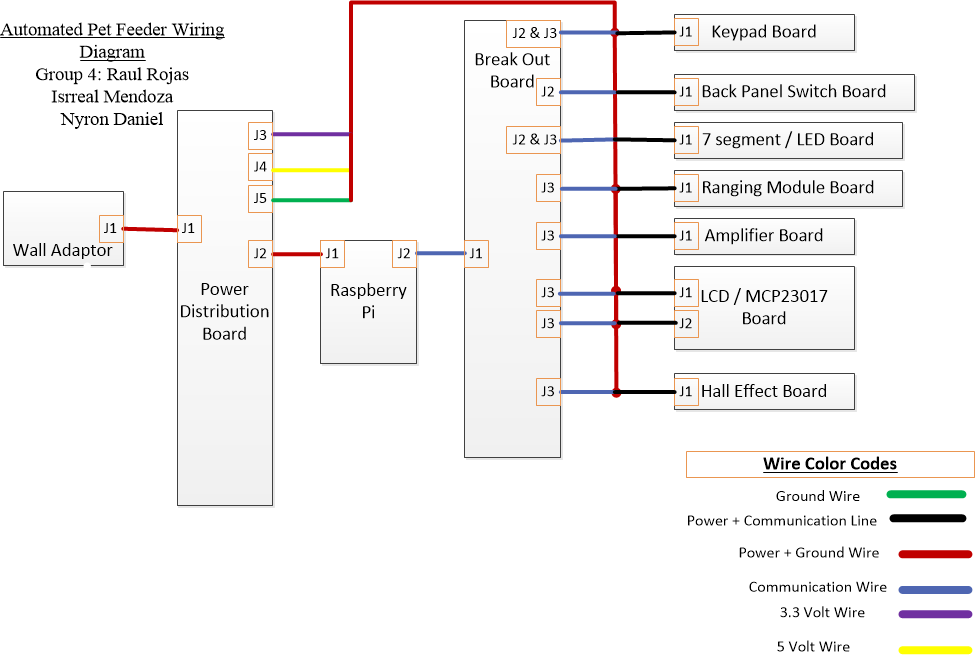


Figure 21: Automated Pet Feeder Wiring Diagram

Board Layout Diagram

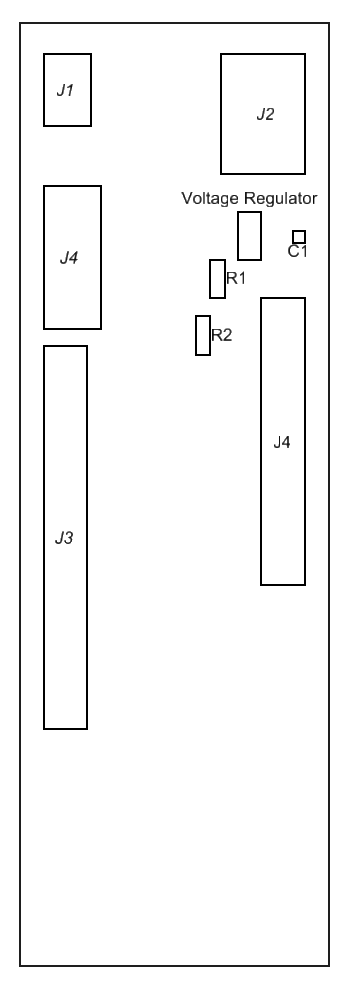


Figure 22: Power Distribution Board

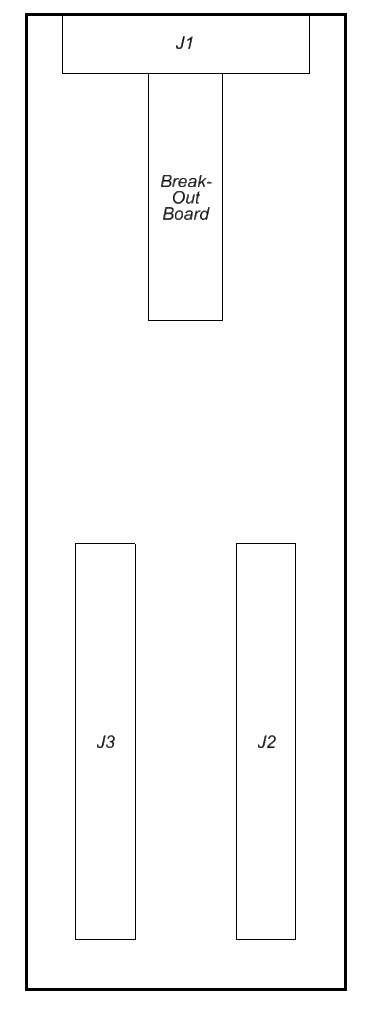


Figure 23: Breakout Board.

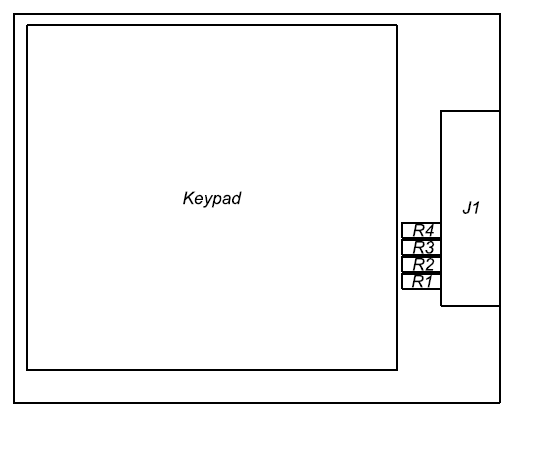


Figure 24: Keypad Board.

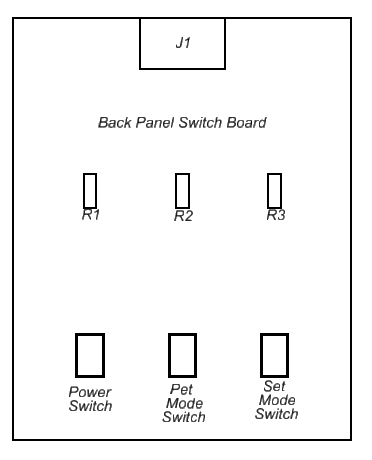


Figure 25: Back Panel Switch Board.

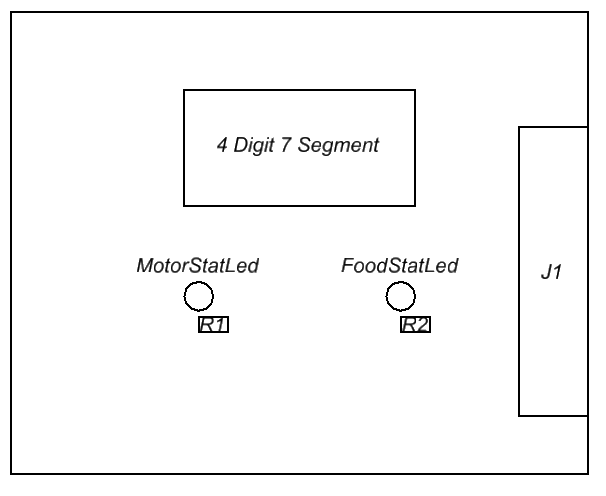


Figure 26: 4 Digit 7 Segment Board.

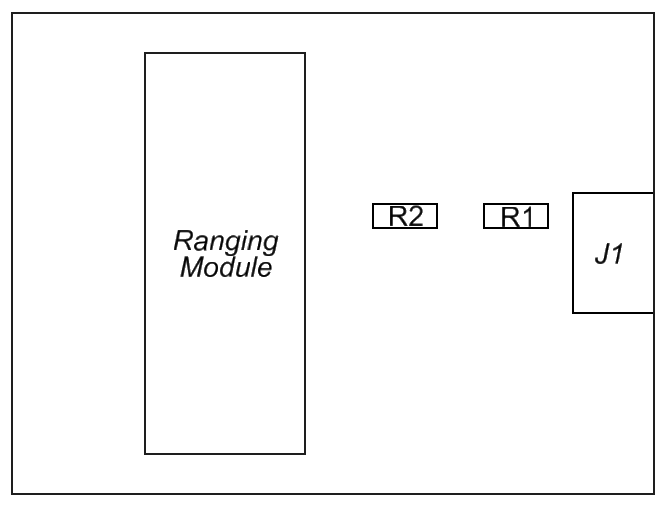


Figure 27: Ranging Module Board.

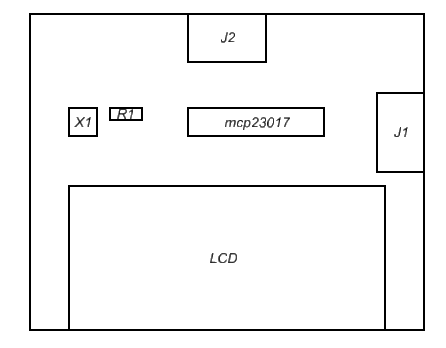


Figure 28: LCD Board.

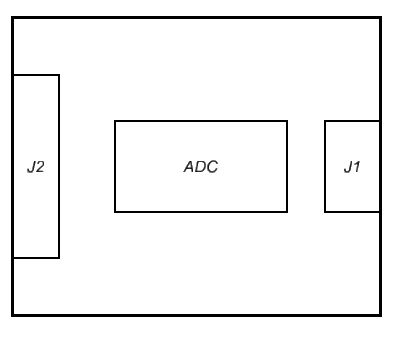


Figure 29: Amplifier Board.

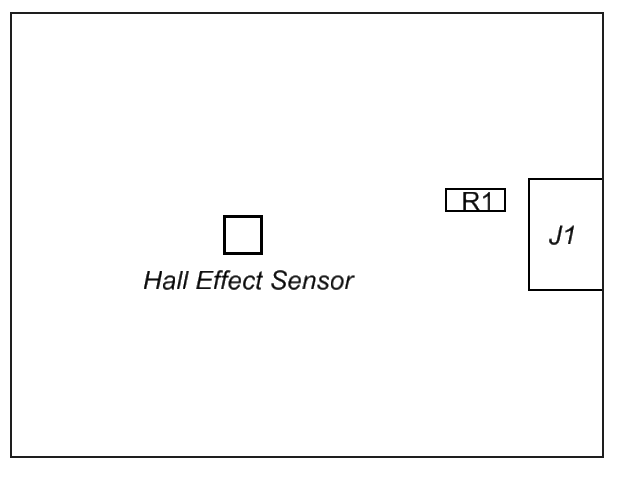


Figure 30: Hall Effect Board.

PARTS LIST

Table 12: Automated Pet Feeder Parts List

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Reference | Part Name | Description | Part # | Unit Cost ($) | Quantity | Total ($) |
|  |  |  |  |  |  |  |
| Breakout Board | Connector | Configurable Spring Terminal Blocks | a15091600ux0313 | 0.20 | 26 | 5.20 |
| Pi T-Cobbler | Assembled Pi T-Cobbler Plus -GPIO Breakout for RPi A+/B +/Pi 2/Pi 3. | 2028 | 13.93 | 1 | 13.93 |
| Proto Board | Adafruit Perma-Proto Raspberry Pi Breadboard PCB Kit | 1135 | 7.95 | 1 | 7.95 |
| Resistor | 4.7kΩ | MF1/4LCT52R222J | 0.10 | 4 | 0.40 |
|  |  |  |  |  |  | 27.48 |
| Hall Effect Board | Connector | Configurable Spring Terminal Blocks | a15091600ux0313 | 0.20 | 4 | 0.80 |
| Hall Effect Sensor | transducer that varies its output voltage in response to a magnetic field | SS441R | 5.00 | 1 | 5.00 |
| Proto Board | MCM Electronics PC Breadboard | 932525 | 0.99 | 1 | 0.99 |
| Resistor | 4.7kΩ | MF1/4LCT52R222J | 0.10 | 1 | 0.10 |
|  |  |  |  |  |  | 34.37 |
| Hx711 Board | Connector | Configurable Spring Terminal Blocks | a15091600ux0313 | 0.20 | 8 | 1.60 |
| Load Cell Amplifier | 24-bit Dual Channel ADC | IM131029002 | 7.00 | 1 | 7.00 |
| Proto Board | MCM Electronics PC Breadboard | 932525 | 0.99 | 1 | 0.99 |
|  |  |  |  |  |  | 43.96 |
| Keypad Board | Connector | Configurable Spring Terminal Blocks | a15091600ux0313 | 0.20 | 8 | 1.60 |
| Keypad | 16-Button Numeric Keypad | AK-1607-N-BBW-R | 9.25 | 1 | 9.25 |
| Proto Board | MCM Electronics PC Breadboard | 932525 | 1.20 | 1 | 1.20 |
|  |  |  |  |  |  | 56.01 |
| LCD Board | Connector | Configurable Spring Terminal Blocks | a15091600ux0313 | 0.20 | 9 | 1.80 |
| LCD | LCD Display 16x2 Parallel 5 volt White Backlit Transmissive | NHD-0216K1Z-NSW-BBW-L | 10.95 | 1 | 10.95 |
| MCP23017 | IC I/O EXPANDER / I2C / 16B 28SDIP | MCP23017-E/SP-ND | 1.20 | 1 | 1.20 |
| Potentiometer | trimmable potentiometer | COM-09806 | 0.95 | 1 | 0.95 |
| Proto Board | MCM Electronics PC Breadboard | 932525 | 1.99 | 1 | 1.99 |
| Resistor | (1x 1kΩ) & (2x 4.7kΩ) | MF1/4LCT52R222J | 0.10 | 3 | 0.30 |
|  |  |  |  |  |  | 73.20 |
| Power Distribution Board | Voltage Regulator | VOLTAGE REGULATOR. NEGATIVE, ADJUSTABLE. NTE-957 | Model #NTE957 | 3.09 | 1 | 3.09 |
| Capacitor | 0.1µF | COM-08375 [RoHS](https://www.sparkfun.com/static/rohs/) | 0.25 | 1 | 0.25 |
| Connector | Configurable Spring Terminal Blocks | a15091600ux0313 | 0.20 | 15 | 3.00 |
| Micro USB 3.0 Breakout Board | USB Female Receptacle Breakout board | BRK-uUSBm | 6.99 | 1 | 6.99 |
| Proto Board | Adafruit Perma-Proto Raspberry Pi Breadboard PCB Kit | 1135 | 7.95 | 1 | 7.95 |
| Resistor | (1x220Ω) & (1x352Ω) | MF1/4LCT52R222J | 0.10 | 3 | 0.30 |
| 0.3USB Breakout Board | USB Female Receptacle Breakout board | BRK-USB-A | 7.99 | 1 | 7.99 |
|  |  |  |  |  |  | 102.77 |
| Ranging Module Board | Connector | Configurable Spring Terminal Blocks | a15091600ux0313 | 0.20 | 4 | 0.80 |
| Ultrasonic Ranging Module | provides 2cm - 400cm non-contact measurement function | 20-019-100 | 5.55 | 1 | 5.55 |
| Proto Board | MCM Electronics PC Breadboard | 932525 | 0.99 | 1 | 0.99 |
| Resistor | (1x 1kΩ) & (1x 2kΩ) | MF1/4LCT52R222J | 0.10 | 2 | 0.20 |
|  |  |  |  |  |  | 110.31 |
| Seven Segment Board | Connector | Configurable Spring Terminal Blocks | a15091600ux0313 | 0.20 | 7 | 1.40 |
| LED | Standard LED - Through Hole Red Tinted Diffused | TLHR4600 | 0.44 | 2 | 0.88 |
| Proto Board | MCM Electronics PC Breadboard | 932525 | 1.20 | 1 | 1.20 |
| Resistor | 220 Ω | MF1/4LCT52R222J | 0.10 | 2 | 0.20 |
| Seven Segment | Red alpha-numeric 4-digit 7-segment display with TTL, SPI or I2C Serial Interface. | COM-11441 | 12.95 | 1 | 12.95 |
|  |  |  |  |  |  | 126.94 |
| Switches Board | Connector | Configurable Spring Terminal Blocks | a15091600ux0313 | 0.20 | 5 | 1.00 |
| Proto Board | MCM Electronics PC Breadboard | 932525 | 1.99 | 1 | 1.99 |
| Resistor | 620Ω | MF1/4LCT52R222J | 0.10 | 3 | 0.30 |
| Fixed Switch | Slide Switches SPDT 2POS TOP VERT T/H SLIDE SWITCH | [SLS121PC04](https://www.mouser.com/Search/ProductDetail.aspx?R=SLS121PC04virtualkey50660000virtualkey506-SLS121PC04) | 0.96 | 3 | 2.88 |
| Momentary Switch | SWITCH PUSH SPST-NO 3A 120V | EG1932-ND | 2.79 | 1 | 2.79 |
|  |  |  |  |  |  | 135.90 |
| Miscellaneous | Arcade Buttons | 60mm large arcade button. Used to induce pet interaction (red, blue, green) | 1190 | 5.95 | 3 | 17.85 |
| Bolts and Nuts | Machine Screws Round Head Combo #6-32 x 3/8 in | 528490 | 1.18 | 10 | 11.80 |
| Corner Braces | Zinc-plated inside corner brace. Used to keep the wood structure together. | 20482 | (4 pack) 3.21 | 6 | 19.26 |
| Food Dispenser | Dry Food Dispenser - used to encase the motor and hall effect sensor. The fins were modified to dispense pet food. | 1461 | 22.99 | 1 | 22.99 |
| Hinges | Narrow Utility Hinges | 24097 | 1.97 | 3 | 5.91 |
| Weighing Load Cell | Electronic Balance Weighing load cell sensor (0-5kg) | B006W21DUO | 7.89 | 1 | 7.89 |
| Servo Motor | High Powered, High Torque, Metal Gear Servo motor | MG92B | 11.95 | 1 | 11.95 |
| Hardboard Tempered Wood | (0.125in. X 23.75 in. X 47.75 in.) Sheet of wood that was used to build the physical structure. | 7005015 | 4.97 | 3 | 14.91 |
| Wire | 22-gauge solid Hook-up wire 22-foot 6 color kit. | WK-106 | 18.25 | 2 | 36.50 |
| Wood Pillar (support Beams) | 2 in. x 2 in. x 12 ft. Rough Green Western Red Cedar Lumber | 395337 | 5.95 | 2 | 11.90 |
| Wood Stick (base) | 2 “ x 2” x 8’ Pressure-Treated Lumber | 1461 | 3.37 | 1 | 3.37 |
|  | Window Bolt (Lock) | 2" wide white window bolt. Used to lock the inside wiring. | 13518 | 2.78 | 2 | 5.56 |
| Total |  |  |  |  |  | 305.88 |