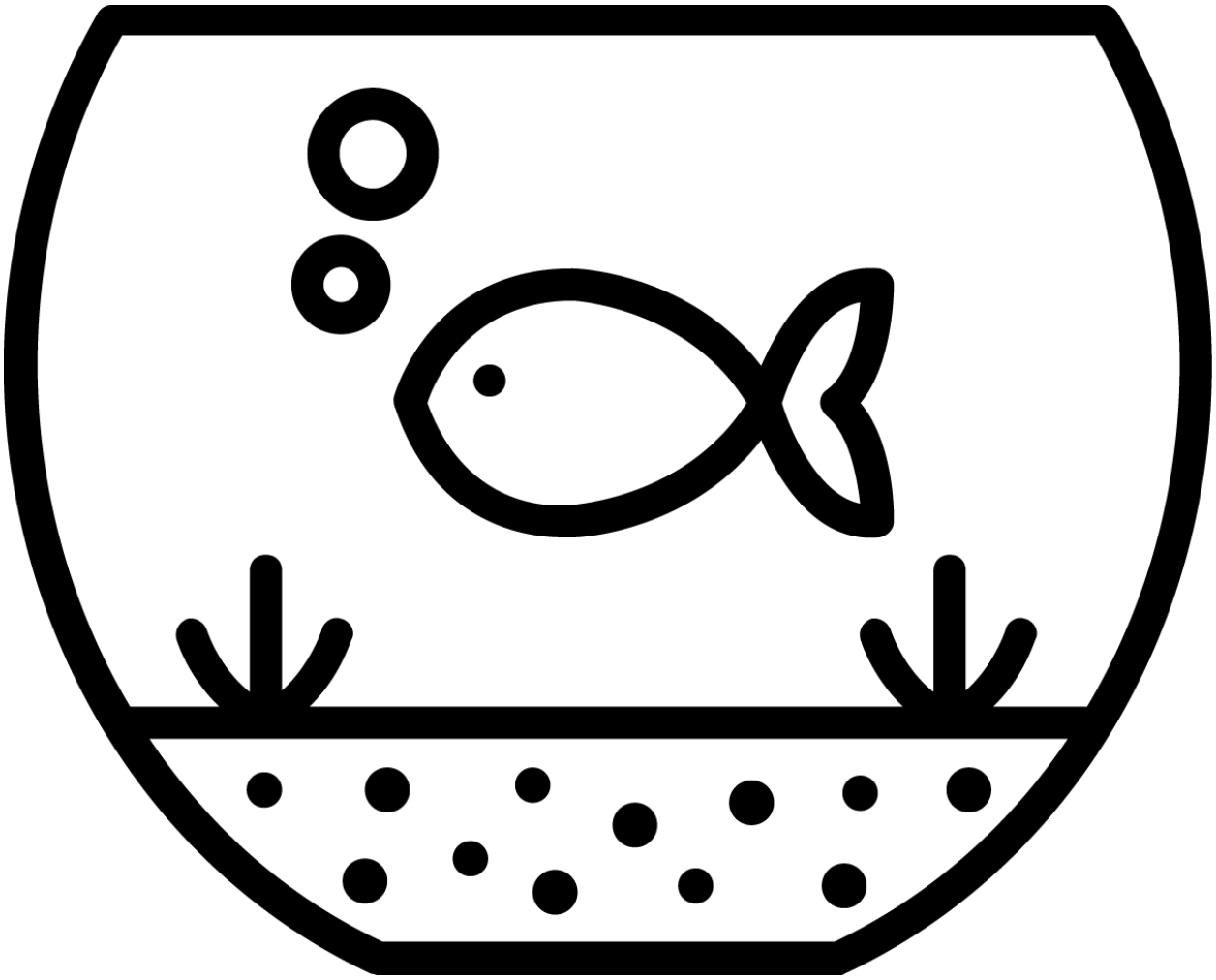
Automatic Fish Feeder

Joe Eckstein
Nicole Rexwinkel

CPE 329-09 Spring 2017 Gerfen



Objective:

The purpose of this project is to integrate a servo motor with the MSP432 Real Time Clock and a user interface consisting of an LCD Display and Keypad. These components will be used to create an automatic fish feeding device. The device will operate by taking user input to set the real time clock and desired feeding times. Additional features of this device include a menu that allows the user to reset the time of day, number of feedings, and feeding time.

System Requirements:

- The system shall have an LCD display.
    - The LCD shall display two lines.
    - The LCD shall display 16 characters per line.
    - The LCD shall operate in 4 bit nibble mode.
    - The characters displayed shall be 5 pixels wide and 8 pixels tall.
- The LCD shall display the the time of day and time last fed during normal operation.
- The LCD shall display instructions that prompt the user for input when needed.
    - The LCD shall display a blinking cursor when user input is expected.
- The system shall have a 12 key keypad.
- The system shall measure time using the MSP432 Real Time Clock.
    - The time of day and feeding times shall be entered by the user in the format of HH:MM.
    - The time of day and feeding times shall be entered by the user in the format of Military time (see Table 1).
    - The system shall be capable of storing up to 3 feeding times.
- The system shall utilize a servomotor capable of turning up to 180 degrees.
- The system shall contain a food storage container and food dispensing system.
    - The food dispensing system will be operated by the servomotor.
- The system shall implement a menu sequence.
    - The menu sequence shall allow the user to feed their fish on demand.
    - The menu sequence shall allow the user to reset the Real Time Clock.
    - The menu sequence shall allow the user to reset their desired feeding times.
- The system shall be powered by a battery.

Table 1: Conversion for Civilian Time to Military Time (AM and PM)

Civilian Time (AM)	Military Time	Civilian Time (PM)	Military Time
12:00	00:00	1:00	13:00
1:00	01:00	2:00	14:00
2:00	02:00	3:00	15:00
3:00	03:00	4:00	16:00
4:00	04:00	5:00	17:00
5:00	05:00	6:00	18:00
6:00	06:00	7:00	19:00
7:00	07:00	8:00	20:00
8:00	08:00	9:00	21:00
9:00	09:00	10:00	22:00
10:00	10:00	11:00	23:00
11:00	11:00	12:00*	24:00

(*) Table 1 shows the conversion from Civilian to Military Time. The Real Time clock on the MSP432 is capable of taking in times ranging from 00:00 to 23:59. 24:00 is an alias of 00:00; therefore, if a user wishes to feed the fish at 12:00am (Midnight) the correct input would be 00:00.

System Specifications:

Table 2: System Specifications for Automatic Fish Feeder

Device	Parameter	Value
MSP432	Variant	MSP-EXP432P401R
	Input Voltage	4.17 [V]
	DCO	3[MHz]
LCD	VCC	5 [V]
	Vo	0 [V]
	Number of Lines	2 Lines
	Font Size	5x8 pixels
	Characters per Line	16 characters
	Mode of Operation	4 bit Nibble
Servomotor	Degrees of Rotation	180 [deg]
	Operating Voltage	5 [V]
	PWM Period	20ms
	Duty Cycle	5-10%
Keypad	Connected Resistance	20[Ω]
Battery Pack	Operating Voltage	3.7V
	Rated Capacity	12000mAh

System Architecture

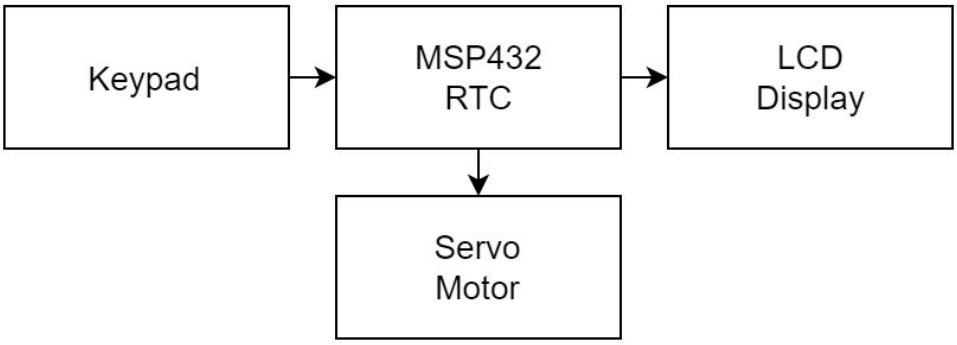


Figure 1: High-Level Hardware Diagram

Figure 1 shows the high level hardware diagram for the automatic fish feeder. Major components for this product include the keypad, MSP432, LCD Display, and the Servo Motor. The keypad is used to take in user input to set up the Real Time Clock as instructions are displayed on the LCD. User input from the keypad is processed by the MSP432 and used to determine when the servo motor should rotate.

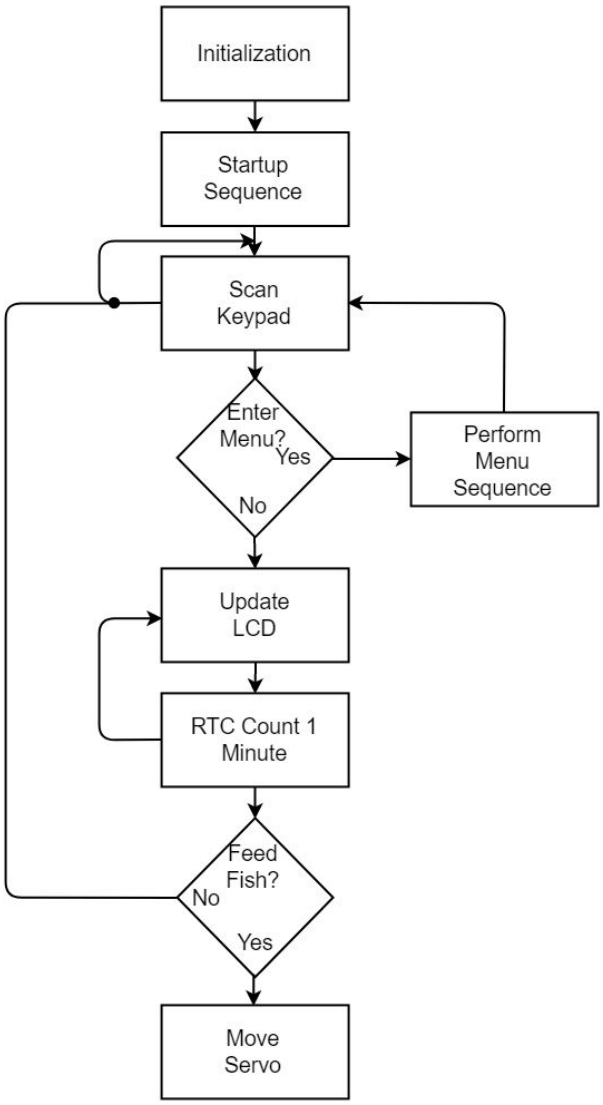


Figure 2: High Level Software Flowchart for Automatic Fish Feeder

Figure 2 shows a high level flowchart which describes the process used to operate the automatic fish feeder. First, all devices and variables used in software are initialized. The device moves through a start up sequence which takes in user input to set the Real Time Clock, number of feedings per day, and feeding times. After the startup sequence is complete, the keypad is scanned for user input to open the menu sequence. If the '#' key is pressed, the menu sequence is entered. When the menu sequence has been completed, device returns to scanning the keypad. If no key is pressed to enter the menu sequence, the LCD display is updated every minute to reflect current time and time last fed. Every minute, the MSP432 compares the Real Time Clock to the user specified feeding time and determines whether or not is time to feed the fish. Feeding occurs by turning a servo to push out food from a container.

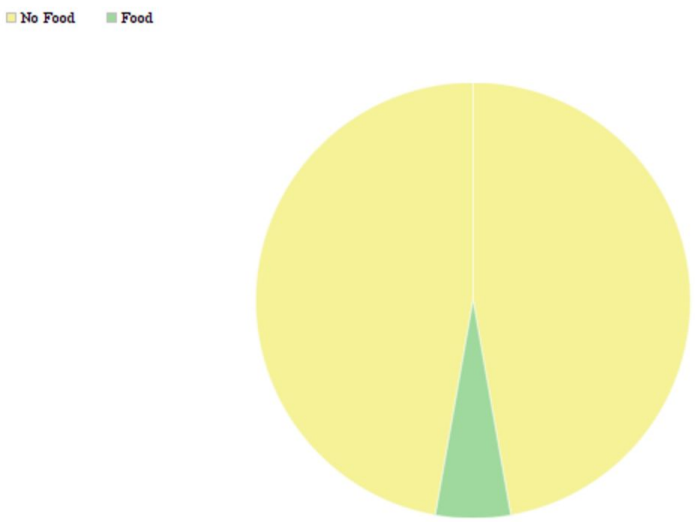


Figure 3: Food Storage Container Cross-Sectional View

Figure 3 shows a cross sectional view of the food storage container. The user is expected to fill the smaller portion of the cylinder with their choice of fish food. A small dispensing hole is located off center of the food holding portion. When the servo is activated, the section containing food is swept over the food dispensing hole. The servo positions were calibrated such that the section containing food would not rest over the food dispensing hole, instead it is quickly brushed over the dispensing location so that only a small amount of food is able to fall out at every rotation.

Component Design:

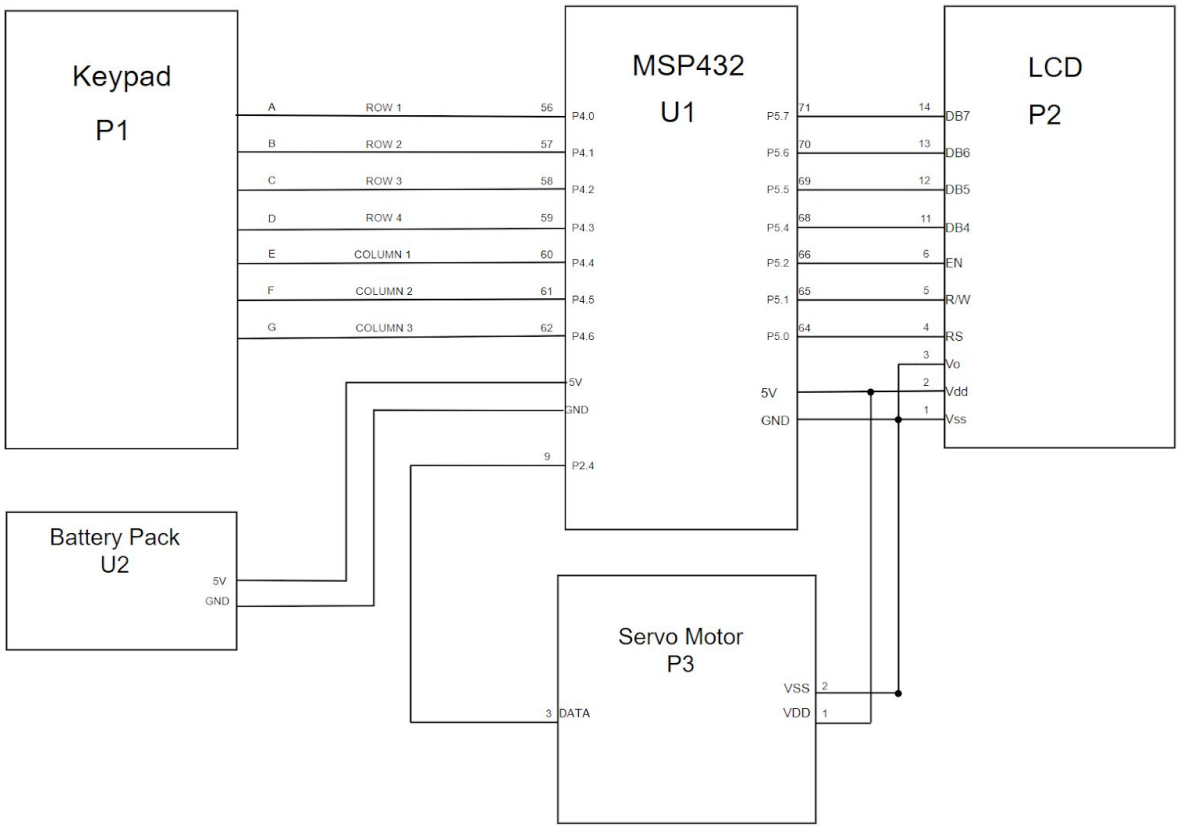


Figure 4: Wiring Diagram for Automatic Fish Feeder

Figure 4 shows the wiring diagram for the automatic fish feeder, and displays the connections made to the MSP432 from peripheral devices and the battery pack. The peripherals include a 12-Key keypad, LCD Display, and servo motor.

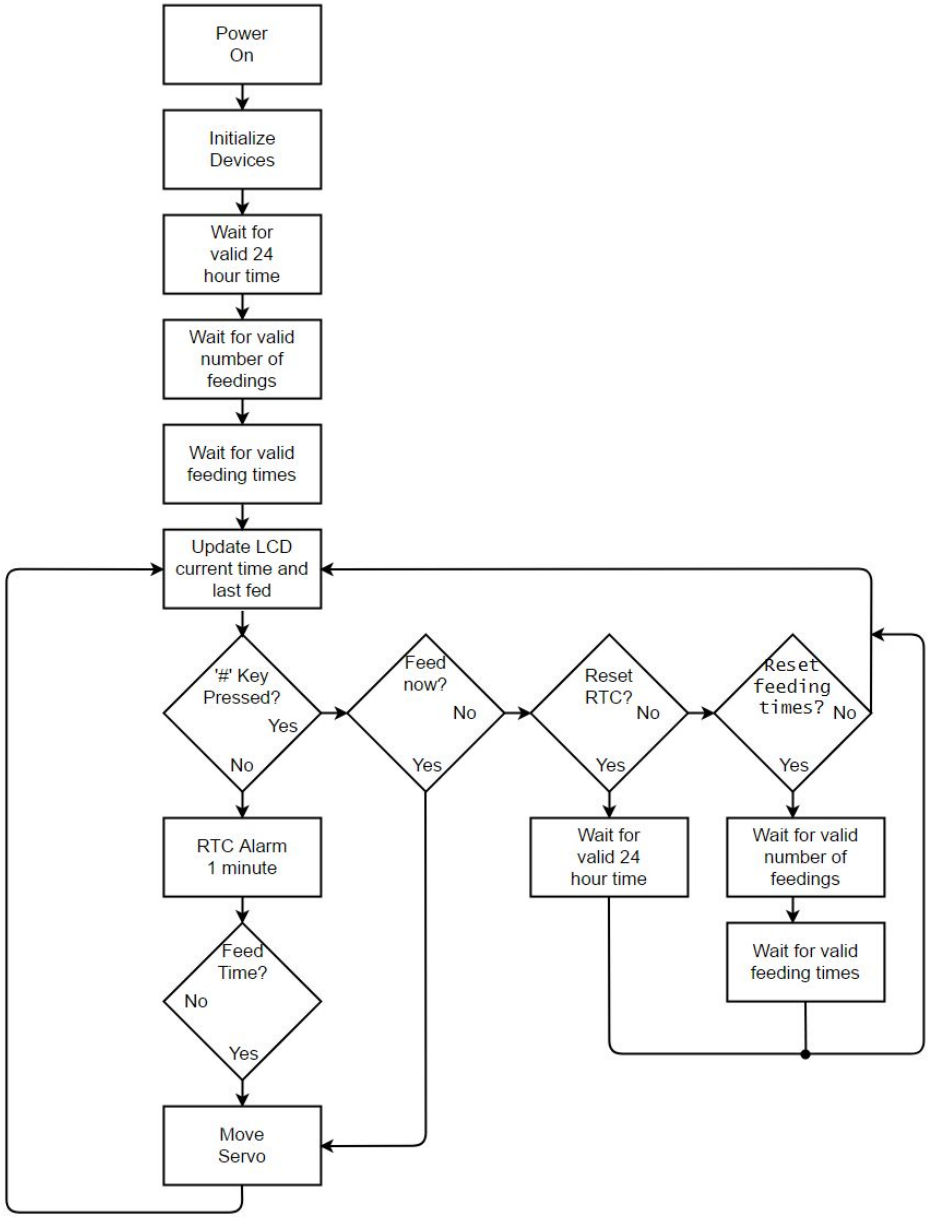


Figure 5: Low-Level Software Flowchart for Automatic Fish Feeder

Figure 5 shows the low level software flow chart for the automatic fish feeder. After powering on the system and initializing devices, the system waits for valid user input to set the Real Time Clock. Valid input is checked by taking in each value pressed on the keypad and determining if it falls within a valid range supported by the system. A similar process is implemented when the user is prompted for the number of feedings per day and desired feeding time. Instructions are displayed on the LCD at every step of the process to assist the user in operating the device.

After setting up the Real Time Clock and the desired feeding times, the user has the option to enter a menu. The menu is entered when the user presses the '#' key any time after completing the startup sequence. The first option on the menu is to feed the fish immediately. If the user selects this option by pressing the '*' key, the current time of the Real Time Clock is saved and the servo is activated to feed the fish. The LCD is updated to reflect the Real Time Clock current time [HH:MM], and the time last fed is also updated.

If the user chooses to move forward on the menu, they will encounter an option to reset the Real Time Clock. If this this option is selected, the user will be prompted to enter a valid time of day as they were in the startup sequence. Moving on to the next and final option will allow the user to reset feeding times. This option asks the user to provide a valid number of feedings per day and daily feeding times for each one.

The menu can be exited by continuing through each option and pressing the '#' key to exit on the last option. The Real Time Clock has an alarm set that will go off every minute. When the one minute alarm goes off, the MSP432 compares the current time with the desired feeding times. If the current time is not a feeding time, the LCD is updated with the current time and time last fed. If the current time is a feeding time, the servo is activated to feed the fish and the LCD is updated with current time and a new 'time last fed'.

Bill Of Materials:

Table 3: Bill of Materials for Automatic Fish Feeder

Item Number	Item Description	Supplier Name	Part Number	Quantity	Cost per Unit	Extended Cost
1	External Battery Supply Device	Texas Instruments	296-46669-ND	1	25.07	25.07
2	Female Jumper Cables	Digikey	1528-1159-ND	1	3.95	3.95
3	Box Packaging	Digikey	HM1095-ND	1	9.85	9.85
4	MSP43201R	Texas Instruments	MSP-EXP432P401R	1	13.03	13.03
5	6”M/F Jumpers	Amazon	HR0291	7	0.0625	0.4375
6	Key Pad	AdaFruit	1824	1	7.50	7.50
7	Tower Pro Servo Motor	Gear Best	SG90S	1	2.45	2.45
Total Cost						**62.29**

Table 3 shows the bill of materials for all components and peripheral devices purchased to assemble the automatic fish feeder. The quantity, cost per unit, and extended cost are included in the table along with the total cost of all materials used.

System Integration:

The peripheral devices used in the automatic fish feeder included a keypad, LCD display, and servomotor. Each of these devices had been used to interface with the MSP432 in previous projects; therefore, source code was modified from previous use to satisfy the system requirements for the automatic fish feeder. Some hardware changes that were made for the project included integrating a new keypad. This process involved testing pins on the keypad to determine which pins were used for rows and which pins for columns. After determining the pinout for the new keypad, source code was modified to support the new device.

The Real Time Clock was a feature on the MSP432 that had not been implemented in previous projects. In order to understand how the Real Time Clock works, the data sheet and technical reference manual for the MSP432 were used to determine how to set up the Real Time Clock and how interrupts for the Real Time Clock could be used to accomplish multiple feedings per day. Based on the information provided, it was determined that the Real Time Clock would be implemented by asking the user to provide all time of day input in a 24 hour format. The interrupt for the Real Time Clock is used to set an alarm every minute. When the alarm for one minute is set, the LCD is updated to show current time and the last time the servo was activated (time last fed). In addition, after the one minute alarm, the current time of the Real Time Clock is compared to the user input for desired feeding times. If the current time of the Real Time clock matches any one of the feeding times, the servo is activated.

While working on the source code for the automatic fish feeder, one of the main goals was to improve the user interface such that any person could operate the system with minimal difficulty. During each step of the startup sequence, the LCD displays instructions for the user to follow. Any time the user is expected to supply input for the device, a flashing cursor appears on screen. Additional steps taken to improve the user interface included adding a menu feature to the device. The menu can be accessed by pressing the '#' key, and additional instructions on screen guide the user through menu operation. Some options that were determined to be useful for the user included options to feed the fish on demand, resetting the Real Time Clock, and resetting the amount of feedings and feeding times per day.

After the peripheral devices were connected and the source code was operational, the chassis for the device, food storage container, and food dispensing system were designed. The chassis consisted of a plastic box which was altered to add mounting holes and cut-outs for the keypad, LCD, and servo wires. The servo was mounted into a cylindrical food storage container, and a food dispensing system was mounted on to the servo. A resealable lid was added to the food storage container so that the user had the option of refilling the food container or checking on the remaining food capacity.

The food dispensing system was designed so that food is held in a small section within the cylinder. When the servo spins, the food containing section is quickly swept over a dispenser opening in the storage device. For every feeding, the servo moves in opposite directions so it does not exceed its 180 degrees of rotation.

Battery-Powered Operation

Current Draw was measured using a digital Multi-meter at the output of the voltage regulators with the peak measurement being taken over multiple cycles.

MSP432 (3.3V Supply)

$I_{Active\,(3MHz)} = 0.76mA$

$I_{Asleep\,(3MHz)} = 0.43mA$

Servo and LCD (5.0V Supply)

$I_{Operating} = 180mA$

$I_{Standby} = 56.6mA$

TPS63002 (5.0V Voltage Regulator)

$\sim 80\%$ Efficient for our Amperage load supplying Servo

TPS63001 (3.3V Voltage Regulator)

$\sim 75\%$ Efficient for our Amperage load supplying MSP432

AEenergy AE503759P6HA

$\sim 12000mAH$ Capacity

I-AV 5.0V DC-DC Converter

Operating

$P = IV = (180mA)(5.0V) = 0.9W$

$P_{DC-DC\,(5V)} = \frac{P_{out\,5V}}{\eta} = \frac{0.9W}{0.80} = 1.125W$

$I_{DC-DC\,(5V)} = \frac{1.125W}{3.7V} = 304.05mA$

Standby

$P = IV = (55.6mA)(5.0V) = 278mW$

$P_{DC-DC\,(5V)} = \frac{P_{out\,5V}}{\eta} = \frac{278mW}{0.80} = 347.5mW$

$I_{DC-DC\,(5V)} = \frac{347.5mW}{3.7V} = 93.92mA$

I-AV 3.3V DC-DC Converter

Active

$$P = IV = (0.76mA)(3.3V) = 2.508mW$$

$$P_{DC-DC\,(3.3V)} = \frac{P_{out\,3.3V}}{\eta} = \frac{2.508mW}{0.75} = 2.2294mW$$

$$I_{DC-DC\,(3.3V)} = \frac{2.2294mW}{3.7V} = 602.52\mu A$$

Asleep

$$P = IV = (0.43mA)(3.3V) = 1.419mW$$

$$P_{DC-DC\,(3.3V)} = \frac{P_{out\,3.3V}}{\eta} = \frac{1.419mW}{0.75} = 1.892mW$$

$$I_{DC-DC\,(3.3V)} = \frac{1.892mW}{3.7V} = 511.35\mu A$$

Average Current Supplied By Battery

The system operates on a 1.6% duty cycle, awake and working for 1 second of every minute.

$$I_{AV\,(ON)} = I_{AV\,(5.0V\ Supply\ Servo\ Active)} + I_{AV\,(3.3V\ Supply\ MSP\ Active)} = 304.05mA + 602.52\mu A = 304.653mA$$

$$I_{AV\,(STANDBY)} = I_{AV\,(5.0V\ Supply\ Servo\ Standby)} + I_{AV\,(3.3V\ Supply\ MSP\ Asleep)} = 93.92mA + 511.35\mu A = 94.43135mA$$

$$I_{AV\,(TOTAL)} = (\tfrac{1}{60})(304.653mA) + (\tfrac{59}{60})(94.43135mA) = 97.94mA$$

Autonomy

$$Time \simeq \frac{Battery\ Capacity\ (mAH)}{I_{AV}(mA)}$$

$$Time \simeq \frac{12000\ (mAH)}{97.94(mA)}$$

$$Time \simeq 122.52\ Hours$$

Discussion & Conclusion:

The automatic fish feeding project was successfully completed by implementing several peripheral devices and integrating the devices with the MSP432 Real Time Clock; however, future iterations of this product would involve improvements to the mechanical design and the power consumption of the device.

The software developed for the automatic fish feeder was able to perform all of the functions laid out in the system requirements. Some features which were added later on in the development process included the menu sequence, and additional user interface improvements. *Displaying instructions and messages on the LCD to guide the user through the startup sequence and menu sequence not only improved the user interface, but it was useful during the debugging process to determine where certain sections of the source code fell through.* The user interface could be improved in future versions of the device to take in the time of day in civilian time as opposed to the 24 hour format it is currently set for. The menu could also be expanded to include more options such as the LCD brightness.

The mechanical concept went through some changes throughout the design process for the automatic fish feeder. Originally, the design for the food dispensing system included a cylinder with multiple food storage sections, and the servo would move in smaller amount to dispense food. The original design was then simplified such that there was only one small section to store food inside the cylinder and the remaining space was left open to prevent food from falling out (see Figure 3). Due to the bulky design of the food dispensing system attached to servo, some design flaws included failure to dispense food. *This problem could be taken care of in future improvements by creating a food storage container that is custom fit for the servo inside, and 3D printing the food dispensing system attached to the servo out of lightweight material so that the servo is not disrupted during activation.*

The amount of current drawn by the servo while in standby mode was not considered while designing the system*. An improvement to the system would to add a transistor to switch off the supply to the servo and stop sending PWM to allow the MSP430 to move into deep sleep in LPM3 or LPM4 to further increase battery life.*

Appendix A - Source Code:

main.c

```
#include "msp.h"
#include "lcd.h"
#include "keypad.h"
#include "servo.h"
#include "rtc.h"
#include "Startup_Sequence.h"

void io_init(void);
void set_feed(void);
void feed_fish(void);
void main_display(void);
void menu_sequence(void);

int num_feed = 0;                       // Stores the number of Scheduled Feedings
int time_stamp[4] = {2,5,6,1};          // Stores the Timestamp of the last Feeding
int feed_time[3][4] = {{2,5,6,1},{2,5,6,1},{2,5,6,1}};  // Array of Scheduled Feedings
int servo_position = 0;                 // Keeps track of servo position within can

// Main system Function for Automatic Fish Feeder
void main(void){

    WDTCTL = WDTPW | WDTHOLD;           // Stop watchdog timer

    //Initialization Sequence
    io_init();      // Initialize both internal and external components

    set_time();     // Ask User to Setup System Clock
    set_feed();     // Ask User to Setup Scheduled Feedings
    delay();        // A Small Delay

       keypad_sleep(); // Setup Keypad for Sleep Mode Operation
       Clear_LCD();    // Clear LCD after Setup Complete

       main_display(); // Draw Sleep Mode UI

    // Wake up on exit from ISR
    SCB->SCR &= SCB_SCR_SLEEPONEXIT_Msk;

    // Main Loop, System is 100% Interrupts Based from this point on
    while(1){
        __sleep();          // System sleeps in LPM0
        __no_operation();   // For debugger
    }
}
```

```
// Function to set up board IO
// No input passed, no input returned
// Modifies System Registers
void io_init(void){

    // Terminate all remaining pins on the device
    P1->DIR |= 0xFF; P1->OUT = 0;
    P2->DIR |= 0xFF; P2->OUT = 0;
    P3->DIR |= 0xFF; P3->OUT = 0;
    //P4->DIR |= 0xFF; P4->OUT = 0; Keypad
    //P5->DIR |= 0xFF; P5->OUT = 0; Display
    P6->DIR |= 0xFF; P6->OUT = 0;
    P7->DIR |= 0xFF; P7->OUT = 0;
    P8->DIR |= 0xFF; P8->OUT = 0;
    P9->DIR |= 0xFF; P9->OUT = 0;
    P10->DIR |= 0xFF; P10->OUT = 0;

    servo_init();   // Setup RTC
    keypad_init();  // Setup Keypad for Data Entry
    LCD_init();     // Setup LCD Display
    Home_LCD();     // Clears and Returns Cursor Home
    rtc_init();     // Setup RTC

}

// Port4 (Keypad) interrupt handler to let us be in low power mode.
// Instead of constantly scanning the keypad we will get an interrupt
// when the # key is pressed
// No input passed, no input returned
// Modifies System Registers
void PORT4_IRQHandler(void){
    NVIC_DisableIRQ(PORT4_IRQn);    // Disable further Keypad Interrupts
    menu_sequence();                // Run System Menu
    delay2(450);                    // Wait before sleeping as a de-bounce
    keypad_sleep();                 // Setup Keypad to Sleep
    P4 -> IFG = 0;                  // Clear Interrupt Flag before returning
}
```

```
// On-board Real Time Clock Interrupt Handler
// No input passed, no input returned
// Modifies Global Variables and System Registers
void RTC_C_IRQHandler(void){
    // Time-Event Interrupt Handler (Every Minute)
    if (RTC_C->CTL0 & RTC_C_CTL0_TEVIFG){
        // Unlock the RTC module and clear time event interrupt flag
        RTC_C->CTL0 = (RTC_C->CTL0 & ~(RTC_C_CTL0_KEY_MASK |  RTC_C_CTL0_TEVIFG)) |
RTC_C_KEY;
        // Re-lock the RTC
        RTC_C->CTL0 = RTC_C->CTL0 & ~(RTC_C_CTL0_KEY_MASK);

        // Check if time is 1st scheduled feeding
        if((RTCHOUR == ((feed_time[0][3] << 4) | (feed_time[0][2])))&&
           (RTCMIN  == ((feed_time[0][1] << 4) | (feed_time[0][0])))){
                feed_fish();

        }
        // Check if time is 2nd scheduled feeding
        else if((RTCHOUR == ((feed_time[1][3] << 4) | (feed_time[1][2])))&&
                (RTCMIN  == ((feed_time[1][1] << 4) | (feed_time[1][0])))){
                feed_fish();

        }
        // Check if time is 3rd scheduled feeding
        else if((RTCHOUR == ((feed_time[2][3] << 4) | (feed_time[2][2])))&&
                (RTCMIN  == ((feed_time[2][1] << 4) | (feed_time[2][0])))){
                feed_fish();

        }

        // Print the current time on the LCD Display (Clock)
        Home_LCD();
        // Isolate each of the digits of the BCD string
        LCD_data(((RTC_C->TIM1&RTC_C_TIM1_HOUR_HD_MASK)>>RTC_C_TIM1_HOUR_HD_OFS)+
'0');
        LCD_data(((RTC_C->TIM1&RTC_C_TIM1_HOUR_LD_MASK)>>RTC_C_TIM1_HOUR_LD_OFS)+
'0');
        LCD_data(':');
        LCD_data(((RTC_C->TIM0&RTC_C_TIM0_MIN_HD_MASK) >>RTC_C_TIM0_MIN_HD_OFS)+ '0');
        LCD_data(((RTC_C->TIM0&RTC_C_TIM0_MIN_LD_MASK) >>RTC_C_TIM0_MIN_LD_OFS)+ '0');
}
}
```

```
// Sets up Scheduled Feeding Times
// No input passed, no input returned
// Modifies Global Variables
void set_feed(void){
    int command = 0;         // Store Keypad Entry
    int i = 0;               // Count iteration
    int ready = 0;           // Boolean to move on to next step
    for(i=0; i < 3; i++){
        feed_time[i][0] = 2;
        feed_time[i][1] = 5;
        feed_time[i][2] = 6;
        feed_time[i][3] = 1;
    }

    // Ask For Number of Feedings
    Clear_LCD();
    LCD_command(0x0F);  // enable display, blink cursor
    LCD_string("# of Feedings?  (1-3)");
    while ((command == 0) || (command > 3)){    //continue asking for feedings until
valid input
        command = keypad_getkey();               //for valid input, set command
    }

    LCD_char(command);                          //write keypad real value to LCD
    num_feed = command;                         //set num_feeds to command value

    command = 0;
    for (i=0; i < num_feed; i++){               //begin asking for feeding times
        ready = 0;
        Clear_LCD();                            //clear the LCD
        LCD_string("Feeding Time #");           //Write string to LCD
        LCD_data(i+1+'0');                      //Write which feeding time is being
set
        LCD_line2();                            //move to line 2
        LCD_string("(hh:mm)");                  //display format for entering time
        delay2(150);                            //short delay
        //Feed Time Hour High Digit
                                                //wait for keypad input
        while(command != 14 && command != 1 && command != 2){
            command = keypad_getkey();          //pull data from keypad
        }
        LCD_char(command);                      //display key pressed
        feed_time[i][3] = keypad_num(command);  //set first key to first hour high
digit
        command = 0;                            // reset command

        delay2(150);                            //debounce

        //Feed Time Hour Low Digit
        while((ready == 0) && (command == 0)){
                command = keypad_getkey();
                if (command != 0){
                    if( feed_time[i][3] == 0 ||  feed_time[i][3] ==1){
                        if(command <= 11 || command == 14){
                            ready = 1;
                        }
                        else{
                            command = 0;
                        }
                    }
                    else if( feed_time[i][3] == 2){
```

```
                        if(command <= 3 || command == 14){
                            ready = 1;
                        }
                        else{
                            command = 0;
                        }
                    }
                    else{
                        ready = 0;
                        command = 0;
                    }
                }
                else{
                    ready = 0;
                    command = 0;
                }
            }
        feed_time[i][2] = keypad_num(command);
        LCD_char(command);
        command = 0;
        LCD_data(':');
        delay2(150);
        //Feed Time Minute High Digit
        while((command >= 7 && command <= 13)|| (command == 0)){
        //continue get_keypad until (0-5)
            command = keypad_getkey();
        }
        LCD_char(command);
        feed_time[i][1] = keypad_num(command);
        command = 0;

        delay2(150);

        while((command >=12 && command <=13)|| (command == 0)){
        //continue get_keypad until(0-9)
            command = keypad_getkey();
        }
        LCD_char(command);
        feed_time[i][0] = keypad_num(command);
        command = 0;
        LCD_command(0x0C);  // enable display, don't blink cursor
        delay2(150);
    }
}

// Function that tells the servo where to move and logs the time the fish was fed at.
void feed_fish(){
    time_stamp[3] = (((RTC_C->TIM1 & RTC_C_TIM1_HOUR_HD_MASK)>>RTC_C_TIM1_HOUR_HD_OFS
));
    time_stamp[2] = (((RTC_C->TIM1 & RTC_C_TIM1_HOUR_LD_MASK)>>RTC_C_TIM1_HOUR_LD_OFS
));
    LCD_data(':');
    time_stamp[1] = (((RTC_C->TIM0 & RTC_C_TIM0_MIN_HD_MASK) >>RTC_C_TIM0_MIN_HD_OFS
));
    time_stamp[0] = (((RTC_C->TIM0 & RTC_C_TIM0_MIN_LD_MASK) >>RTC_C_TIM0_MIN_LD_OFS
));
    Clear_LCD();
    if(servo_position == 0){
        servo_position = 4;
    }
    else if(servo_position == 4){
```

```
        servo_position = 0;
    }
    servo_move(servo_position);
    main_display();

}
// Function to display the menu when the user wakes the system from idle.
void menu_sequence(void){
    keypad_init();
    int command = 0;
    delay2(150);
    Clear_LCD();
    LCD_string("MENU");
    LCD_line2();
    LCD_string("*FEED      NEXT#");
    while(command != 13 && command != 15){
        command = keypad_getkey();
    }
    delay2(150);
    if(command == 13){
        command = 0;
        Clear_LCD();
        Home_LCD();
        feed_fish();
        main_display();
    }
    else if(command == 15){
        command = 0;
        Clear_LCD();
        Home_LCD();
        LCD_string("SET TIME OF DAY");
        LCD_line2();
        LCD_string("*RESET     NEXT#");
        while(command != 13 && command != 15){
                command = keypad_getkey();
            }
        delay2(150);
        if(command == 13){
           command = 0;
           set_time();
           main_display();
        }
        else if(command == 15){
            command = 0;
            Clear_LCD();
            Home_LCD();
            LCD_string("RESET FEED TIMES");
            LCD_line2();
            LCD_string("*RESET     NEXT#");
            while(command != 13 && command != 15){
                    command = keypad_getkey();
                }
            delay2(150);
            if(command == 13){
                command = 0;
                set_feed();
                main_display();
            }
            else if(command == 15){
                command = 0;
                Clear_LCD();
```

```
                main_display();
            }
            }

    }
}
// Print the current time on the LCD Display
void main_display(void){
    Home_LCD();
    LCD_data(((RTC_C->TIM1 & RTC_C_TIM1_HOUR_HD_MASK) >> RTC_C_TIM1_HOUR_HD_OFS )+
'0');
    LCD_data(((RTC_C->TIM1 & RTC_C_TIM1_HOUR_LD_MASK) >> RTC_C_TIM1_HOUR_LD_OFS )+
'0');
    LCD_data(':');
    LCD_data(((RTC_C->TIM0 & RTC_C_TIM0_MIN_HD_MASK)  >> RTC_C_TIM0_MIN_HD_OFS )+
'0');
    LCD_data(((RTC_C->TIM0 & RTC_C_TIM0_MIN_LD_MASK)  >> RTC_C_TIM0_MIN_LD_OFS )+
'0');
    LCD_string("     MENU-#");
    LCD_line2();
    LCD_string("Last Fed: ");

    if(time_stamp[0] == 2 && time_stamp[1] == 5 && time_stamp[2]== 6 &&
time_stamp[3]==1){
        LCD_string("N/A");
    }
    else{
        LCD_data((time_stamp[3]) + '0');
        LCD_data((time_stamp[2]) + '0');
        LCD_data(':');
        LCD_data((time_stamp[1]) + '0');
        LCD_data((time_stamp[0]) + '0');
    }
    keypad_sleep();
}
```

Rtc.h

```
/*
 * rtc.h
 *
 *  Created on: June 2, 2017
 *      Author: Joe Eckstein
 */

#ifndef RTC_H_
#define RTC_H_

void rtc_init(void);
void rtc_time(int hour, int minute, int second, int year, int month, int day, int
week_day);

#endif /* RTC_H_ */
```

Rtc.c

```
/*
 * rtc.c
 *
 *  Created on: June 2, 2017
 *      Author: Joe Eckstein
 */

#include "msp.h"
#include "rtc.h"

/*
 * Initializes the RTC.
 * No Input, Nothing Returned
 * Modifies System Registers
 */
void rtc_init(void){
    // Configure Port J
    PJ->DIR |= (BIT2 | BIT3);
    PJ->OUT &= ~(BIT2 | BIT3);
    PJ->SEL0 |= BIT0 | BIT1;                // set LFXT pin as second function
    CS->KEY = CS_KEY_VAL ;                  // Unlock CS module for register access
    CS->CTL2 |= CS_CTL2_LFXT_EN;            // LFXT on

    // Loop until XT1, XT2 & DCO fault flag is cleared
    do
    {
       // Clear XT2,XT1,DCO fault flags
       CS->CLRIFG |= CS_CLRIFG_CLR_DCOR_OPNIFG | CS_CLRIFG_CLR_HFXTIFG |
               CS_CLRIFG_CLR_LFXTIFG | CS_CLRIFG_CLR_FCNTLFIFG;
       SYSCTL->NMI_CTLSTAT &= ~ SYSCTL_NMI_CTLSTAT_CS_SRC;
    } while ((SYSCTL->NMI_CTLSTAT | SYSCTL_NMI_CTLSTAT_CS_FLG)
            && (CS->IFG & CS_IFG_LFXTIFG)); // Test oscillator fault flag

    // Select ACLK as LFXTCLK
    CS->CTL1 &= ~(CS_CTL1_SELA_MASK) | CS_CTL1_SELA_0;
    CS->KEY = 0;                            // Lock CS module from unintended accesses
}

/*
 * Function to set the time in the RTC.
 * Accepts integers of each of the inputs in BCD NOT DECIMAL, BCD!
 * Nothing Returned
 * Modifies System Registers
 */
// NOTE SYSTEM IS EXPECTING BCD INPUT AND WE ONLY DEAL WITH BCD MODE
void rtc_time(int hour, int minute, int second, int year, int month, int day, int
week_day){
    // Configure RTC_C
    RTC_C->CTL0 = RTCKEY_VAL|               // Unlock RTC key protected registers
            RTC_C_CTL0_TEVIE;               // Enable RTC time event interrupt
    RTC_C->CTL13 |= RTC_C_CTL13_BCD |       // BCD mode
            RTC_C_CTL13_TEV_0 |             // Set RTCTEV for 1 minute alarm event
interrupt
            RTC_C_CTL13_HOLD;               // RTC hold

    RTC_C->YEAR = year;                    // Year = 0x2016
    RTC_C->DATE = (month << RTC_C_DATE_MON_OFS) | // Month = 0x10 = October
            (day | RTC_C_DATE_DAY_OFS);     // Day = 0x07 = 7th
    RTC_C->TIM1 = (week_day << RTC_C_TIM1_DOW_OFS) | // Day of week = 0x05 = Friday
            (hour << RTC_C_TIM1_HOUR_OFS);  // Hour = 0x11
```

```
    RTC_C->TIM0 = (minute << RTC_C_TIM0_MIN_OFS) | // Minute = 0x59
            (second << RTC_C_TIM0_SEC_OFS);    // Seconds = 0x45

    RTC_C->CTL13 &= ~(RTC_C_CTL13_HOLD);     // Start RTC calendar mode
    RTC_C->CTL0 &= ~(RTC_C_CTL0_KEY_MASK);  // Lock the RTC registers

    // Enable global interrupt
    __enable_irq();

    NVIC->ISER[0] = 1 << ((RTC_C_IRQn) & 31);
}
```

Lcd.h

```
/*
 * lcd.h
 *
 *  Created on: May 22, 2017
 *      Author: nmrex
 */

#ifndef LCD_H_
#define LCD_H_

#include "msp.h"

/* LCD Commands in 4-bit mode */

#define RS 1
#define RW 2
#define EN 4

void delayMs(int n);
void LCD_write(unsigned char data, unsigned char control);
void LCD_command(unsigned char command);
void LCD_data(unsigned char data);
void LCD_init(void);
void Clear_LCD(void);
void Home_LCD(void);
void LCD_line2(void);
void LCD_string(char * phrase);

#endif /* LCD_H_ */
```

Lcd.c

```
/*
 * lcd.c
 *
 *  Created on: May 22, 2017
 *      Author: nmrex
 */
#include "lcd.h"

/* www.MicroDigitalEd.com
 * p3_3.c: Initialize the LCD using 4-bit data mode.
 * Data and control pins share Port 4.
 * This program does not poll the status of the LCD.
 * It uses delay to wait out the time LCD controller is busy.
 * Timing is more relax than the HD44780 datasheet to accommodate the
 * variations among the LCD modules.
 * You may want to adjust the amount of delay for your LCD controller.
 *
 * Tested with Keil 5.20 and MSP432 Device Family Pack V2.2.0
 * on XMS432P401R Rev C.
 */

/* LCD Commands in 4-bit mode */

/*
 * Routine to setup the LCD on Port 5
 * No input passed, no input returned
 * Modifies System Registers
 */
void LCD_init(void) {
    P5->DIR = 0xF7;     // Make P5 outputs
    delayMs(30);        // Init sequence
    LCD_write(0x30, 0);
    delayMs(30);
    LCD_write(0x30, 0);
    delayMs(1);
    LCD_write(0x30, 0);
    delayMs(3);
    LCD_write(0x20, 0); // 4-bit mode
    delayMs(3);

    LCD_command(0x28);  // 4-bit mode, 2-line, 5x7 characters
    LCD_command(0x06);  // Move Cursor after Characters
    LCD_command(0x01);  // Clear and Home Display
    LCD_command(0x0C);  // enable display, don't blink cursor
}

/*
 * Routine to write to the LCD on Port 5
 * Accepts a , no input returned
 * Modifies System Registers
 */
/* With 4-bit mode, each command or data is sent twice with upper
 * nibble first then lower nibble.
 */
void LCD_write(unsigned char data, unsigned char control) {
    data &= 0xF8;                   // clear lower for control (Leave Data Bits)
    control &= 0x07;                // clear upper for data (Leave Control Bits)
    P5->OUT = data | control;       // Setup Outputs
    P5->OUT = data | control | EN;  // Pulse Enable High
    delayMs(0);
```

```
    P5->OUT &= data & 0xF8;         // clear control bits and enable (Dont touch bit
3)
    P5->OUT &= 0x08;                // Clear Outport except for bit 3
}

/*
 * Routine to write to the LCD on Port 5
 * No input passed, no input returned
 * Modifies System Registers
 */
void LCD_command(unsigned char command) {
    LCD_write(command & 0xF0, 0);   // Write Upper Nibble
    LCD_write(command << 4, 0);     // Write Lower Nibble

    if (command < 4)
        delayMs(4);                 // Commands 1-3 Require more time
    else
        delayMs(1);                 // Other command require less time
}
// Write data such as a character to the display
// accepts char, returns nothing
void LCD_data(unsigned char data) {

    LCD_write(data & 0xF0, RS); // Upper Nibble First
    LCD_write(data << 4, RS);   // Lower Nibble Second

    delayMs(1);

}

/* delay milliseconds when system clock is at 3 MHz */
void delayMs(int n) {
    // A delay
    int i, j;

    for (j = 0; j < n; j++)
        for (i = 750; i > 0; i--);
}
// Clear the LCD
// No Input, No Return
void Clear_LCD(void) {
    LCD_write(0x00, 0x00);  // Pass Upper Bits
    LCD_write(0x10, 0x00);  // Pass Lower Bits
    delayMs(4);
}

// Move to First Location of Display
// No Input, No Return
void Home_LCD(void) {
    LCD_write(0x80, 0x00);  // Pass Upper Bits
    LCD_write(0x00, 0x00);  // Pass Lower Bits
    delayMs(1);
}
// Move to Second Line Of Display
// No Input, No Return
void LCD_line2(void) {

    LCD_write(0xC0, 0x00);  // Pass Upper Bits
    LCD_write(0x00, 0x00);  // Pass Lower Bits
    delayMs(1);
}
```

```
/*
 * Function accepts a string of up to 32 chars to write to the LCD
 * It begins at the current RAM position. To re-rewrite the entire display,
 * clear and home using the applicable commands before calling
 * No Return
 */
void LCD_string(char * phrase){
int char_count = 0;
int i = 0;
    while(phrase[i] != '\0'){
        char_count ++;
        if(char_count<16){
        LCD_data(phrase[i]);
        i++;
        }
        else if(char_count == 16){
            LCD_data(phrase[i]);
            i++;
            LCD_line2();
        }
        else{
        LCD_data(phrase[i]);
        i++;
        }
    }
}
```

Keypad.h

```
/*
 * keypad.h
 *
 *  Created on: May 22, 2017
 *      Author: nmrex
 */

#ifndef KEYPAD_H_
#define KEYPAD_H_



#include "msp.h"
#include "lcd.h"


void delay(void);
void keypad_init(void);
void keypad_sleep(void);
char keypad_getkey(void);
void LED_init(void);
void LED_set(int value);
void LCD_char(int value);
int keypad_num(int value);
void delay2(int n);


#endif /* KEYPAD_H_ */
```

Keypad.c

```
#include "keypad.h"


/* this function initializes Port 4 that is connected to the keypad.
 * All pins are configured as GPIO input pin. The column pins have
 * the pull-up resistors enabled.
 */
void keypad_init(void) {
    P4->DIR = 0;
    P4->REN = 0xF0;     /* enable pull resistor for column pins */
    P4->OUT = 0xF0;     /* make column pins pull-ups */
}

/*
 * This function sets up the keypad to respond to an interrupt from the user pressing
 * The # key.
 */
void keypad_sleep(void){
       P4->DIR = (~BIT6) | 0x0F;                        // Setup 6 as Input 0-3 as Output
    P4->OUT = BIT6;                                          // 6 pull-up all others
low
    P4->REN = BIT6;                             // Enable pull-up resistor (P4.6 output
high)
    P4->IES = BIT6;                             // Interrupt on high-to-low transition
    P4->IFG = 0;                                // Clear all P1 interrupt flags
    P4->IE = BIT6;                              // Enable interrupt for P4.6

    NVIC_EnableIRQ(PORT4_IRQn);
}

/*
 * This is a non-blocking function to read the keypad.
 * If a key is pressed, it returns a unique code for the key. Otherwise,
 * a zero is returned.
 * The upper nibble of Port 4 is used as input and connected to the columns.
 * Pull-up resistors are enabled so when the keys are not pressed, these pins
 * are pulled high.
 * The lower nibble of Port 4 is used as output that drives the keypad rows.
 * First all rows are driven low and the input pins are read. If no key is pressed,
 * they will read as all one because of the pull up resistors. If they are not
 * all one, some key is pressed.
 * If some key is pressed, the program proceeds to drive one row low at a time and
 * leave the rest of the rows inactive (float) then read the input pins.
 * Knowing which row is active and which column is active, the program
 * can decide which key is pressed.
 *
 * Only one row is driven so that if multiple keys are pressed and row pins are
shorted,
 * the microcontroller will not be damaged. When the row is being deactivated,
 * it is driven high first otherwise the stray capacitance may keep the inactive
 * row low for some time.)
 *
 *  --KEYPAD--  --OUTPUT--
 *  *--------*  *--------*
 *  |-1--2--3|  |-1--2--3|
 *  |--------|  |--------|
 *  |-4--5--6|  |-5--6--7|
 *  |--------|  |--------|
 *  |-7--8--9|  |-9-10-11|
 *  |--------|  |--------|
```

```
 *  |-*--0--#|  |13-14-15|
 *  *--------*  *--------*
 *
 */
char keypad_getkey(void) {
    int row, col;
    const char row_select[] = {0x01, 0x02, 0x04, 0x08}; /* one row is active */

    /* check to see any key pressed */
    P4->DIR |= 0x0F;            /* make all row pins output */
    P4->OUT &= ~0x0F;           /* drive all row pins low */
    delay();                    /* wait for signals to settle */
    col = P4->IN & 0xF0;        /* read all column pins */
    P4->OUT |= 0x0F;            /* drive all rows high before disable them */
    P4->DIR &= ~0x0F;           /* disable all row pins drive */
    if (col == 0xF0)            /* if all columns are high */
        return 0;               /* no key pressed */

    /* If a key is pressed, it gets here to find out which key.
     * It activates one row at a time and read the input to see
     * which column is active. */
    for (row = 0; row < 4; row++) {
        P4->DIR &= 0x0F;                /* disable all rows */
        P4->DIR |= row_select[row];     /* enable one row at a time */
        P4->OUT &= ~row_select[row];    /* drive the active row low */
        delay();                        /* wait for signal to settle */
        col = P4->IN & 0xF0;            /* read all columns */
        P4->OUT |= row_select[row];     /* drive the active row high */
        if (col != 0xF0) break;         /* if one of the input is low, some key is
pressed. */
    }
    P4->OUT |= 0x0F;                    /* drive all rows high before disable them */
    P4->DIR &= 0x0F;                    /* disable all rows */
    if (row == 4)
        return 0;                       /* if we get here, no key is pressed */

    /* gets here when one of the rows has key pressed, check which column it is */
    if (col == 0xE0) return row * 4 + 1;    /* key in column 0 */
    if (col == 0xD0) return row * 4 + 2;    /* key in column 1 */
    if (col == 0xB0) return row * 4 + 3;    /* key in column 2 */
    if (col == 0x70) return row * 4 + 4;    /* key in column 3 */

    return 0;   /* just to be safe */
}

/*
 * Function that takes the "Scan Code" from the keypad and sends the appropriate
 * ASCII character to the LCD
 */
void LCD_char(int value) {
    value &= 0x0F;          // Only looking at bottom bit
    // Translate Key Number into ASCII and output to the LCD
    switch(value) {
    case(1) : LCD_data('1');
        break;
    case(2) : LCD_data('2');
        break;
    case(3) : LCD_data('3');
        break;
    case(4) : LCD_data('A');
        break;
```

```
    case(5) : LCD_data('4');
        break;
    case(6) : LCD_data('5');
        break;
    case(7) : LCD_data('6');
        break;
    case(8) : LCD_data('B');
        break;
    case(9) : LCD_data('7');
        break;
    case(10) : LCD_data('8');
        break;
    case(11) : LCD_data('9');
        break;
    case(12) : LCD_data('D');
        break;
    case(13) : LCD_data('*');
        break;
    case(14) : LCD_data('0');
        break;
    case(15) : LCD_data('#');
        break;
    case(16) : LCD_data('E');
        break;
    }
}

/* make a small delay
 * No input passed, no input returned
 */
void delay(void) {
}

/*
 * Function that Translates the "Scan Code" and returns the number hit.
 * It will return 15 if a letter, *, #, or other invalid input is received.
 */
int keypad_num(int value){
    value &= 0x0F;          // Only looking at bottom bit
    // Translate Key Number into ASCII and output to the LCD
    switch(value) {
    case(1) : return 1;
    case(2) : return 2;
    case(3) : return 3;
    case(4) : return 15;
    case(5) : return 4;
    case(6) : return 5;
    case(7) : return 6;
    case(8) : return 15;
    case(9) : return 7;
    case(10) : return 8;
    case(11) : return 9;
    case(12) : return 15;
    case(13) : return 15;
    case(14) : return 0;
    case(15) : return 15;
    case(16) : return 15;
    }
    return 15;
}
```

```
/*
 * A delay used for keypad debounce.
 * Takes input of number of cycles to run returns nothing
 */
void delay2(int n){
    int i, j;
    for (j = 0; j < n; j++)
        for(i = 750; i > 0; i--);
}
```

Servo.h

```
/*
 * servo.h
 *
 *  Created on: June 2, 2017
 *      Author: Joe Eckstein
 */

#ifndef SERVO_H_
#define SERVO_H_

void servo_init(void);
void servo_move(int position);


#endif /* SERVO_H_ */
```

Servo.c

```
/*
 * servo.c
 *
 *  Created on: June 2, 2017
 *      Author: Joe Eckstein
 */

#include "servo.h"
#include "msp.h"

/*
 * Sets up the Servo using Timer A0
 * Accepts no input and returns nothing
 * Modifies System Registers
 */
void servo_init(void){
    TIMER_A0->CCR[0] = 61000;                       // Set Period of Square Wave
    TIMER_A0->CCR[1] = 2288;                        // Set Initial Duty Cycle
    TIMER_A0->CCTL[1] = TIMER_A_CCTLN_OUTMOD_7;     // Setup Timer_A for Reset/Set
Mode
    TIMER_A0->CTL = TIMER_A_CTL_SSEL__SMCLK |       // SMCLK as Timer A Clock Source
                    TIMER_A_CTL_MC__UP |            // Timer A Up mode
                    TIMER_A_CTL_CLR;                // Clear TAR
    P2->DIR |= BIT4;                                // P7.6~7 set TA1.1~2
    P2->SEL0 |= BIT4;
    P2->SEL1 &= ~(BIT4);
}
// Set Duty Cycle depending on desired position
// Input of integer between 0 and 18 meaning 0 to 180 degrees in increments of 10
// Returns Nothing
// Modifies SYstem Registers
void servo_move(int position){
    switch(position){
            case(0):
                TIMER_A0->CCR[1] = 2000;
                break;
            case(1):
                TIMER_A0->CCR[1] = 2296;
                break;
            case(2):
                TIMER_A0->CCR[1] = 2592;
                break;
            case(3):
                TIMER_A0->CCR[1] = 2888;
                break;
            case(4):
                TIMER_A0->CCR[1] = 3184;
                break;
            case(5):
                TIMER_A0->CCR[1] = 3480;
                break;
            case(6):
                TIMER_A0->CCR[1] = 3776;
                break;
            case(7):
                TIMER_A0->CCR[1] = 4072;
                break;
            case(8):
                TIMER_A0->CCR[1] = 4368;
                break;
```

```
            case(9):
                TIMER_A0->CCR[1] = 4664;
                break;
            case(10):
                TIMER_A0->CCR[1] = 4968;
            break;
            case(11):
                TIMER_A0->CCR[1] = 5272;
            break;
            case(12):
                TIMER_A0->CCR[1] = 5576;
            break;
            case(13):
                TIMER_A0->CCR[1] = 5880;
            break;
            case(14):
                TIMER_A0->CCR[1] = 6184;
            break;
            case(15):
                TIMER_A0->CCR[1] = 6488;
            break;
            case(16):
                TIMER_A0->CCR[1] = 6792;
            break;
            case(17):
                TIMER_A0->CCR[1] = 7096;
                break;
            case(18):
                TIMER_A0->CCR[1] = 7400;
                break;
            }
}
```

Startup_Sequence.c

```
/*
 * Startup_Sequence.h
 *
 *  Created on: May 31, 2017
 *      Author: nmrex
 */

#ifndef STARTUP_SEQUENCE_H_
#define STARTUP_SEQUENCE_H_

#include "lcd.h"
#include "keypad.h"

void set_time(void);

#endif /* STARTUP_SEQUENCE_H_ */
```

Startup_Sequence.h

```
// this file contains functions that will be used when the
// fish feeder is first initialized.
#include "lcd.h"
#include "keypad.h"
#include "rtc.h"

// Allows User to Set System Time
// Accepts no input and returns nothing
// Modifies SYstem Registers
void set_time(void){
       Home_LCD();
       Clear_LCD();
       LCD_string("Time (hh:mm)");
       LCD_line2();
       int HR[2] = {0,0};
       int HR_HD = 0;
       int MIN[2] = {0,0};
       int command = 0;
       int ready = 0;
    LCD_command(0x0F);  // enable display, blink cursor
       // Hour High Digit
       while(command != 14 && command != 1 && command != 2){
              command = keypad_getkey();
       }
       LCD_char(command);
       HR[0] = keypad_num(command);
       HR_HD = HR[0];
       command = 0;
       delay2(150);
       ready = 0;

       // Hour Low Digit
       while((ready == 0) && (command == 0)){
              command = keypad_getkey();
              if (command != 0){
                  if(HR_HD == 0 || HR_HD ==1){
                      if(command <= 11 || command == 14){
                          ready = 1;
                      }
                      else{
                          command = 0;
                      }
                  }
                  else if(HR_HD == 2){
                      if(command <= 3 || command == 14){
                          ready = 1;
                      }
                      else{
                          command = 0;
                      }
                  }
                  else{
                      ready = 0;
                      command = 0;
                  }
              }
              else{
                  ready = 0;
                  command = 0;
              }
```

```
        }
        LCD_char(command);
        HR[1] = keypad_num(command);
        command = 0;
        ready = 0;
        LCD_data(':');

        delay2(150);

        // Minute High Digit
        while((command >= 7 && command <= 13)|| (command == 0)){
               command = keypad_getkey();
        }
        LCD_char(command);
        MIN[0] = keypad_num(command);
        command = 0;

        delay2(150);

        // Minute Low Digit
        while((command >=12 && command <=13)|| (command == 0)){
               command = keypad_getkey();
        }
        LCD_char(command);
        MIN[1] = keypad_num(command);
        command = 0;

        rtc_time((HR[0] << 4)|(HR[1]),(MIN[0] << 4)|(MIN[1]), 0x01, 0x2017, 0x06, 0x03,
0x06);
    LCD_command(0x0C);  // enable display, don't blink cursor
        delay2(150);

}
```

Appendix B - References:

1. Mazidi, Chen, Naimi, Salmanzadeh, “TI MSP432 ARM Programming For Embedded Systems”, 2016.
2. Texas Instruments, “Mixed-Signal Microcontrollers”, MSP432P401R datasheet, March 2015 [Revised March 2017].
3. Texas Instruments, “MSP432P4xx SimpleLink™ Microcontrollers, Technical Reference Manual”, March 2015 [Revised March 2017]
4. Texas Instruments, “MSP432P401R SimpleLink™ Microcontroller LaunchPad™ Development Kit (MSP‑EXP432P401R)”, March 2015 [Revised March 2017]
5. Texas Instruments, “Fuel Tank MKii Battery BoosterPack™ Plug-in Module”, Battery Controller User’s Guide, June 2016
6. Adafruit, “Datasheet 3x4 Phone-style Matrix Keypad”, May 2017
7. Lumex, “LCM-S01602DTR/M 16x2 Character LCD Module, TN, Reflective Datasheet”, September 2002
8. LCD Controller Datasheet
9. Digilent, “PmodCLP™ Parallel LCD Display Module Reference Manual”, April 2008
10. Samsung, “SK0066U LCD Controller Datasheet”
11. Jeff Gerfen, “CPE 329 - Problems of Interest #4 - Batteries and Power Consumption”, Fall 2013
12. Fish Bowl by Made by Made from the Noun Project