EGR226 Proposal: Slot Machine

by

Joshua Johnston and Jordan Hayes

EGR 226 Introduction to Embedded Systems
Section 901
Date Submitted: March 6, 2019
Instructor: Dr. Nabeeh Kandalaft

Objectives

For this project a slot machine was to be designed, coded, built, and wired that uses the 16x4 LCD for user interface, a keypad for numerical entries, pushbuttons for user input and gameplay, and LEDs and speakers for game feedback in different gameplay conditions.

The heart of the project was to be the TI MSP432 Launchpad microcontroller board. The MSP432 was coded in Code Composer Studio and used to control different peripherals to create the features that complete these objectives. A 10K potentiometer was used to control LCD brightness; this level of brightness was measured using ADC and displayed with a 7-Segment Display. There were options to include mechanisms to detect a coin being entered, empty the project of coins, and to put the project into an organized and attractive body.

The MSP432 is to be used as an embedded controller. Parts included are to be from the lab kit, and anything added must not exceed $50.

Parts included

Stronghero3D Printing Rainbow PLA Filament
- 1.75mm, 1 Kg, ±.05mm
- $26.99

Door Barrel Bolt, Brass
- $6.99

Brass Hinges
- $6.99

Brass Rod- Rectangular (2 pk)
- $1.19

Super Glue
- $3.00

Creality Ender 3D Printer
- Used to print the housing, mounts, coin box, and ramp

FS90 Micro Servo
- 0~120 degrees
- No-Load current: 200 mA (@6 V)
- Stall current: 600 mA (@6 V)

HD44780 16x 4 LCD

5mm Colored LEDs

Resistors
- 100 Ohm
- 1K Ohm

10 K Potentiometer

2N7000 Transistors

Breadboards

TI MSP432

Piezo Buzzer
- >30mA run current

7 Segment Display

Standard 12 Key Keypad

1825910-6- TE Tactile Pushbuttons

22 Gauge Jumper Wires

Soldering Kit

Procedure

Upon bootup, the slot machine should display a welcome screen with team member names and a clever game title for 3 seconds.

Next, a Main Menu is displayed with three options to be chosen with the keypad: Sounds, Enter Credits, and Play. The numbers associated on the keypad are 1, 2, and 3 in the order above. Sounds menu should allow the user to choose from a few predetermined victory sounds, Enter Credits menu should allow the user to enter credits into the game using the keypad, and Play button starts the game.

When Sounds is selected, a screen is displayed prompting the user to select from your defined sounds (at least 2) using the keypad. When the user selects a sound, that sound should be demonstrated. When the user selects *, the user is returned to the Main Menu and the currently selected sound is accepted. To create sounds, Eq. 2 has to be used.

$$\frac{F_{clk}}{F_{note}} \quad (1)$$

When Enter Credits is selected, a screen is displayed prompting the user to enter credits. Integer input should be displayed on screen. User should not be allowed to enter more than a 3-digit number (999) to allow display room on the Game Screen. User input ignored when *-key

pressed, sending back to Main Menu without updating the player credits (CR). User input accepted when #-key pressed, adding input to players Credits (CR) before returning to the Main Menu.

When Play is selected, the Game Screen should be displayed with your slot machine reels: Main Menu, and game information of credits and current bet. CR: Credits displays how many credits the player currently has in the game. After each spin, the credits will be updated. Menu: Main menu button (*) should return the player back to the main menu, allowing to change brightness, add credits, or return to the game screen. BET: Bet displays how much the player currently has wagered. After each spin, the BET is reset to zero.

Two pushbuttons are used for modifying the bet, BET+ and BET-. BET+ increments by 1 up to a max-bet of 5 credits and cannot surpass the amount the player currently has available in credits. BET- decrements by 1 but cannot go below 0.

Another pushbutton is used to start the current round, Play. Selecting Play will do nothing if a wager is not placed by the user. Once a valid wager is placed and Play is pressed, the reels should begin cycling randomly through the available characters. The reels stop with one of two conditions: 5 seconds pass, or Play is pressed again by the player.

When the reels stop, the win conditions are validated to see if the user has won or lost, updating the credits accordingly. If the player wins, all characters' match on at least one of your reels. If one reel matches, the bet is added to the credits. If two reels' match, two times the bet is added to the credits. The player loses if none of the reel's match. The flow and sequence of the program can be referenced in Figure 1.

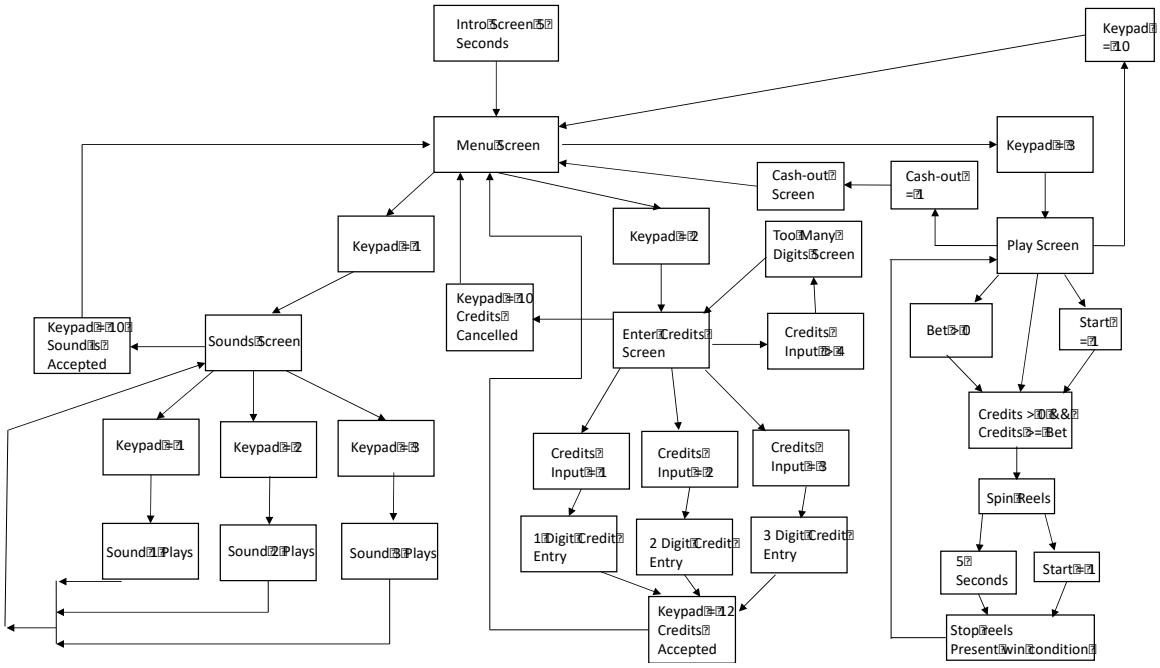

Figure 1: Program Flow Chart

If the user wins, the chosen, or default, victory sound should be played, along with a coded pattern of on LEDs to celebrate the win. The LCD should also display “WIN!” on the line(s) that won. If the player loses, another light pattern will be run and another sound could be played.

One final pushbutton is used to Cash Out from the game, displaying a final cash out screen with a goodbye message. There is an option for this button to empty a coin box or the machine of coins.

The LCD Backlight should be controlled with a Potentiometer. The value of this Potentiometer should be sampled with the ADC and used as a 10-unit scale from of brightness. The Brightness ratio determined by the ADC should be displayed on a 7-segment Display, representing the 10-unit scale as 0-9.

Calculations will have to be made for sending currents to different components using Eq. 1 and inputting resistors into circuits to maintain correct current.

$$V = IR \qquad (2)$$

The project should be put into an attractive, functional form and display. The wiring of the slot machine can be referenced in Figure 2 the wiring schematic. The project should be able to be plugged into a USB wall charger and operate as your slot machine. A feature will also be added to open the coin slot to take out the penny when the user cash out the credits.

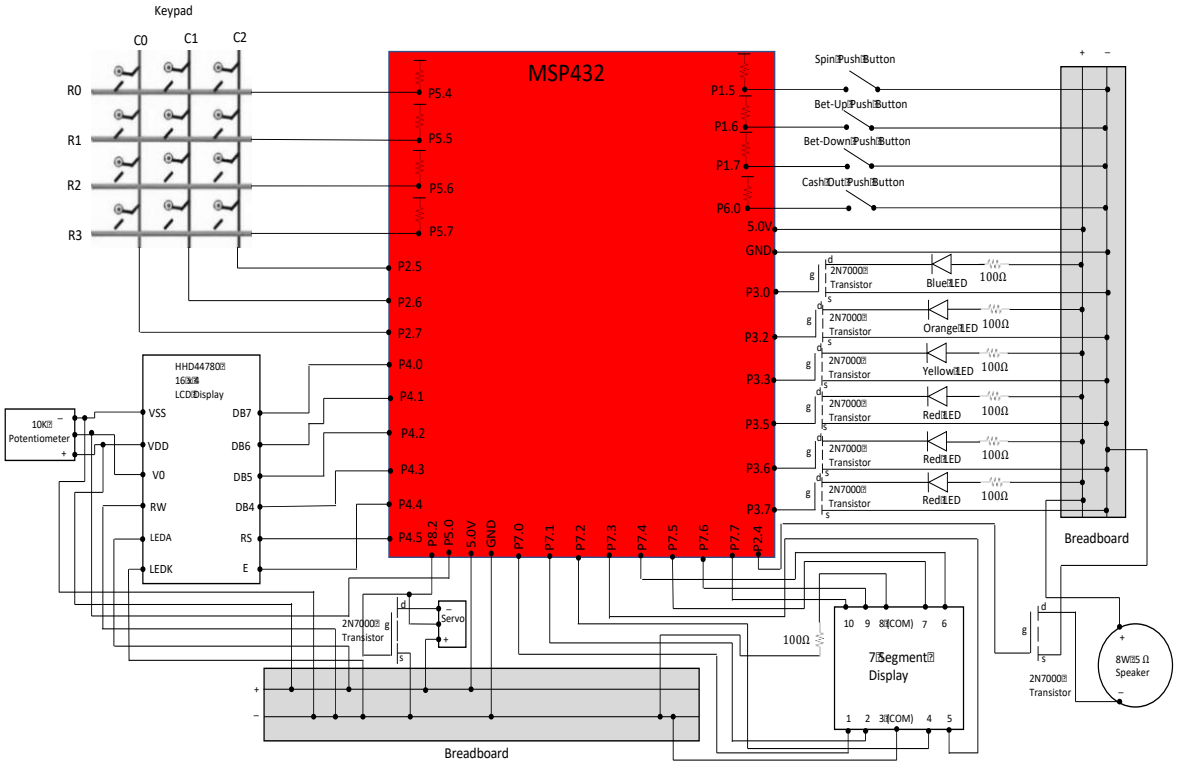

Figure 2: Slot Machine Wiring Schematic

Results

In order to interface with any user, the LCD was used to display menus throughout the program. The LCD pin-out is shown in Table A-1. A library was used and included in the programs called LCD_Library.h as well as LCD_Library.c that included functions to send data and commands to the LCD easily. The first screen is a simple series of characters set to the LCD for 3 seconds using the SysTick clock.

Table A-1 - 16x4 LCD Pin-out

P4.0	D4
P4.1	D5
P4.2	D6
P4.3	D7
P4.4	Enable
P4.5	RS

A potentiometer was used to control the contrast of the LCD and the voltage output was read and converted with an ADC, pin-out shown in Table A-6. The ADC conversion represented the voltage sent through the potentiometer, and a 7 Segment Display (pin-out shown in Table A-5) would display a digit 0-9 corresponding to the voltage level. Setting a pin high on the 7 Segment lights up a segment of a digit connected to the pin.

Table A-5 - 7 Segment Display Pin-out

P7.0	PIN 1
P7.1	PIN 2
P7.2	PIN 4
P7.3	PIN 5
P7.4	PIN 6
P7.5	PIN 7
P7.6	PIN 9
P7.7	PIN 10

Table A-6 – Potentiometer/ADC Pin-out

P5.0	PIN 2
V0 LCD	PIN 2

The slot machine requires user input either from the Standard 12 Key Keypad or from Push Buttons to advance the application (moving through menus, setting credits, starting the spinning reel) so to ensure proper results, a GPIO interrupt was enabled on each of the pushbutton and Keypad pins shown in Table A-2 and Table A-3 below. User input from the keypad was saved in a global variable to set conditions for which menu displayed. Credits were also entered through the keypad and saved as a global variable. Flag variables were created for interrupts to initiate proper commands to the microcontroller. Incorrect user input would display a message or be ignored completely by disabling pending interrupts.

Table A-2 - Standard 12 Key Keypad Pin-out

P5.4	ROW 0
P5.5	ROW 1
P5.6	ROW 2
P5.7	ROW 3
P2.5	COL 0
P2.6	COL 1
P2.7	COL 2

Table A-3 - Pushbutton Pin-out

P1.5	SPIN
P1.6	BET+
P1.7	BET-
P6.0	CASHOUT

After the Welcome Screen expires, the Menu Screen is displayed offering 3 gameplay options: Sounds, Enter Credits, and Play. Credits must be entered and accepted to play the game. The program checks the credit input to ensure it is between 0 and 999. A successful credit entry sends the program back to Menu 2 where the user can now Play. Selecting Play will take you to the Play Screen; credits will be displayed, as well as the current Bet, custom characters, an option to return to the Menu Screen, and a "GOOD LUCK" message. The Bet is a global variable that increases on this screen by pressing BET+, decreases when BET- is pressed, and is set back to zero after a successful spin.

Pressing the Play Button begins a random display of characters to simulate a "spin" and pressing Play again will stop it; it will stop by default in 5 seconds using the Systick timer. The user wins if either one or two lines of characters' match, and loses otherwise by comparing the characters in the lines of the LCD to determine if they ae equal. If the player loses, credits are lost equal to

the Bet. If the player matches one line, credits are added equal to the bet. If both lines match, the player receives two times the Bet in additional credits.

LED patterns were created in sync with its corresponding sound, whether it be the losing, winning chime, or one of the two other custom sounds. Each LED required a 100 Ohm resistor and a 2N700 transistor to drive current. Pinouts for the LEDs is shown below in Table A-4 and were connected to an internally mounted breadboard. Delays between pitches were generated with a SysTick delay to hold notes for certain lengths of time.

Table A-4 - LED Pin-out

P3.0	BLUE
P3.2	ORANGE
P3.3	YELLOW
P3.5	RED1
P3.6	RED2
P3.7	RED3

Both sounds and the servo were controlled using TimerA PWM output. A library was created to hold sequences of output frequencies, or pitches, for the speaker. The speaker's pinout is shown below in Table A-7. Sounds are demoed if the Sounds menu is selected and the sound is selected and the sound played for after a winning spin can be selected. The selected sound is played by sending the note frequency into the speaker function. A desired pitch could be generated using Eq. 1 to get the period for the timer and dividing that number by two, making that the duty cycle.

Table A-7 Audio Speaker Pin-out

P2.4	PWM PIN

A servo was used to empty the coin box of coins when the CASHOUT button is pressed by enabling the interrupt that changes the duty cycle of the Timer A output connected to the FS90 Servo, changing its position enough to allow coins to fall through. Servo pin-out is shown below in Table A-8

Table A-8 Servo Pin-out

P8.2	Speaker Vin

For the housing of the project, a design was created on SolidWorks. Three parts made up the assembly: a roof, a front face with a slanted platform, and the back. The .stl files were saved to an SD card and 3D printed at home using a Creality Ender 3 3D Printer with Stronghero3D Printing Rainbow PLA Filament to give it an attractive and colorful setting. The parts were sanded. The body is held together with brass hinges on one side, and a brass barrel lock on the other, as well as a hook latch. Holes were drilled for the 7 Segment Display and the 10k Potentiometer and were filed into squares. Both the former components had to be soldered to a 22-gauge jumper wire on each lead. The buttons needed two wires soldered and holes drilled for the tactile pushbutton part to stick up through the hole; mounts were printed and drilled in to secure the buttons. Button labels were printed that would also help secure the brass rods. These rectangular brass rods were cut to size and super glued to the button caps provided in the kit and

set into the button labels, through the drilled holes, to make contact with the tactile pushbutton secured on the underside. The LEDs were set into the roof of the housing through drilled holes, and jumper wires were soldered to the leads. Fixtures for the LCD and the Keypad to sit in were made in the .stl file; the LCD is fastened in its fixture with screws; the Keypad, with glue. The MSP432 itself was also screwed into the interior of the housing and a hole for a MicroUSB was built into the .stl file to allow the slot machine to be plugged into any USB port.

Conclusion

For this project we created a slot machine application using the TI MSP432 Launchpad as an embedded controller. For the most part, the parts and components used were from the EGR226 lab kit, and extra features cost $45.16 in total. This is under the $50 limit. The project was completed in a group of two and turned out really well. The housing is sturdy and colorful; however, if the bed of the 3D printer had been heated for a longer amount of time and leveled a little better, the prints would not have warped so much. Issues arose with mounting buttons and the potentiometer because tape was unattractive and would not hold the component well, and glue would glue them shut. Thus, parts were well fastened with the mounts, screws, and adhesives where appropriate holding things in place. Fixtures for both the keypad and the 16x4 LCD were made well and fit nicely. The code is extremely well commented and organized into many different libraries with many different functions. With these libraries included, our main source code is short and sweet. Double clicking or hovering over functions will display or take you to the source of the function if further information is needed, and the files are included in the report in Appendix B. Wires were taped and organized to avoid tangles and interference. Another problem came when the servo was implemented. Alone, the servo would function perfectly as asked, making a full 90-degree rotation, but when integrated into the circuit it seemed to not receive enough power or would not respond at all. This would only improve if wires were rearranged. It should be suggested that components be contained and wires covered to avoid the interference. Additional power sources might also improve performance by maintaining current to each component at all times much easier.

Appendix A

Table A-1 - 16x4 LCD Pin-out

P4.0	D4
P4.1	D5
P4.2	D6
P4.3	D7
P4.4	Enable
P4.5	RS

Table A-2 - Standard 12 Key Keypad Pin-Out

P5.4	ROW 0
P5.5	ROW 1
P5.6	ROW 2
P5.7	ROW 3
P2.5	COL 0
P2.6	COL 1
P2.7	COL 2

Table A-3 - Pushbutton Pin-Out

P1.5	SPIN
P1.6	BET+
P1.7	BET-
P6.0	CASHOUT

Table A-4 - LED Pin-out

P3.0	BLUE
P3.2	ORANGE
P3.3	YELLOW
P3.5	RED1
P3.6	RED2
P3.7	RED3

Table A-5 - 7 Segment Display Pin-out

P7.0	PIN 1
P7.1	PIN 2
P7.2	PIN 4
P7.3	PIN 5
P7.4	PIN 6
P7.5	PIN 7
P7.6	PIN 9
P7.7	PIN 10

Table A-6 – Potentiometer/ADC Pin-out

P5.0	PIN 2
V0 LCD	PIN 2

Table A-7 Audio Speaker Pin-out

P2.4	PWM PIN

Table A-8 Servo Pin-out

P8.2	Speaker Vin

Screens and Design

Figure 3: Intro screen and front of slot machine

Figure 4: Menu Screen

Figure 5: Win Sounds Screen

Figure 6: Enter Credits Screen

Figure 7: Bet Exceeding Limit Screen

Figure 8: Play Screen (Bet = 0)

Figure 9: Play Screen (Bet = 5)

Figure 10: Play Screen (Loss)

Figure 11: Win Screen

Figure 12: Cash-Out Screen

Appendix B

Code and libraries for Slot Machine

Main.c File

```
/*****************************************************************
 *Filename: EGR226Sec901_Johnston_Hayes_Final_Project.c
 *Title: Slot Machine
 *Author: Josh Johnston and Jordan Hayes
 *Date: 04/16/2019
 *Instructor: Dr. Nabeeh Kandalaft
 *Description: Program contains custom libraries that
 *operates buttons, a keypad, 16 x 4 LCD display, LEDs,
 *7 - Segment display, ADC conversions, and a piezo speaker.
 *All combined to make a user friendly functional slot machine.
 *
 *Libraries include:
 *  Slot_Machine.h
 *  LCD_Library.h
 *  SysTick_Library.h
 *  ADC_Library.h
 *  Tunes_Library.h
*****************************************************************/
#include "msp.h"
#include <stdio.h>
#include "Slot_Machine.h"
#include "LCD_Library.h"
#include "SysTick_Library.h"
#include "ADC_Library.h"
#include "Tunes_Library.h"
#include <time.h>

 void main(void){

/********************** Function Initializations ************************
*****************************************************************************/
   lcdInit();
   Intro_Screen();
   Menu_Screen();
   SysTick_delay_us(10);
   SysTickInit_NoInterrupts();
   KeyPad_Init();
   Buttons_init();
   LED_init();
   Character_init();
```

```
  ADC_pin_init();
  sevenSegment_init();
  ADC_init();
  Servo();

  TIMER32_2->CONTROL = 0b11100011;
  NVIC_EnableIRQ(PORT1_IRQn);
  NVIC_EnableIRQ(PORT5_IRQn);
  NVIC_EnableIRQ(PORT6_IRQn);
  _enable_IRQ();
/********************** Function Initializations ************************
*****************************************************************************/

  WDT_A->CTL = WDT_A_CTL_PW | WDT_A_CTL_HOLD;     // stop watchdog timer

   while(1)
   {

      Menu_Options();        //Menu selections
      Sounds_Options();      //Sound selections
      Credits_Input();       //Credits menu to input desired credits from 0 - 999
      Spin_and_Play();       //To go into the play function that spins reels and gives win outcome
      Cash_Out();            //Allows for cash out at any time
      Home_Option();         //Option to return to the menu screen with the * key
      ADC_Reading();         //Does ADC conversion for the 7 - segment display from
potentiometer readings

   }
}
```

Slot_Machine.c

```
/*****************************************************************************
 *****************************************************************************
 *                    Slot_Machine.c
 *            Joshua Johnston and Jordan Hayes
 *                      EGR226
 *            Instructor: Dr. Nabeeh Kandalaft
 *              Created on: March 26, 2019
 *
 *       For use with the MSP432 LaunchPad Development Board
 *
 * Library is used for a custom slot machine gives functions
 * and provides menu screens and functionality for
 * user friendly game play
 *
 * The /// notation makes it so the function description block
 * is visible when you hovering over a function call in any
 * file (this feature is called Intellisense).
 *
 * Functions are each briefly described in comment blocks
 * Functions are commented out throughout program
 * Devices used and pin descriptions are listed below
 *
 * Libraries needed for initializations and other screen functionalities
 *                  LCD_library.h
 *                  SysTick_Library.h
 *
********************** DEVICES AND PARTS ***********************************
* 16 x 4 HHD44780 LCD DISPLAY
* STANDARD 12 KEY KEYPAD
* 4 MOMENTARY PUSH BUTTONS
* 6 LEDs
* 7 SEGMENT DISPLAY
* AUDIO SPEAKER
*
* For more information reference the wiring schematic
*
********************** PIN DESCRIPTIONS ************************************
*
*          ********** 16 X 4 LCD DISPLAY **********
*                   P4.0 -> LCD D4
*                   P4.1 -> LCD D5
*                   P4.2 -> LCD D6
*                   P4.3 -> LCD D7
*                   P4.4 -> LCD E
```

```
*                P4.5 -> LCD RS
*
*        ******* STANDARD 12 KEY KEYPAD *******
*                P5.4 -> ROW 0
*                P5.5 -> ROW 1
*                P5.6 -> ROW 2
*                P5.7 -> ROW 3
*                P2.5 -> COL 0
*                P2.6 -> COL 1
*                P2.7 -> COL 2
*
*        ********** PUSH BUTTONS **********
*                P1.5 -> SPIN
*                P1.6 -> BET-UP
*                P1.7 -> BET-DOWN
*                P6.0 -> CASHOUT
*
*        **********  LEDS  ******************
*                (WIN LEDS)
*               P3.0 -> BLUE LED
*               P3.2 -> ORANGE LED
*               P3.3 -> YELLOW LED
*                (LOSE LEDS)
*               P3.5 -> RED LED
*               P3.6 -> RED LED
*               P3.7 -> RED LED
*
*        ********** 7 SEGMENT DISPLAY **********
*               P7.0 -> Pin 1
*               P7.1 -> Pin 2
*               P7.2 -> Pin 4
*               P7.3 -> Pin 5
*               P7.4 -> Pin 6
*               P7.5 -> Pin 7
*               P7.6 -> Pin 9
*               P7.7 -> Pin 10
*
*        ********** AUDIO SPEAKER **********
*               P2.4 -> PWM Pin
*
*        ********** SERVO MOTOR **********
*               P8.2 -> INPUT Pin
*
*************************************************************************/

//Formatting and commenting completed finish sounds menu
```

```
#include "SysTick_Library.h"
#include "LCD_Library.h"
#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#include "Tunes_Library.h"
#include <time.h>

/*************Global Variables Used Throughout Libraries*****************
*************************************************************************/
uint8_t i = 0, n = 0, x = 0,  j = 0;
int Credits = 0;
int one = 0, two = 0, three = 0;
uint8_t play = 0;

enum Reel_state {DISPLAY_SCREEN, INPUT_BET, SPIN_RESULTS}
Reel_state = DISPLAY_SCREEN;

/*************Global Variables Used Throughout Libraries*****************
*************************************************************************/

/// ************| Menu_Screen |*******************
/// * Brief:  Main menu on LCD Display that
/// *         gives user options to other
/// *         display screen locations
/// * Parm:
/// *     N/A
/// * Return:
/// *      N/A
/// ************************************************/
void Menu_Screen(void){

   if(i != 3  &&  ((BetUp || BetDown || Start) == 1)){
     Bet = 0;
     BetUp = 0;              //Error checking interrupted buttons
     BetDown = 0;              //Resetting if play menu is not present
     Start = 0;
   }

   lcdSetText("MENU", 0, 0);
   lcdSetText("1- Win Sounds", 0, 1);
   lcdSetText("2- Enter Credits", 0, 2);
   lcdSetText("3- Play", 0, 3);
   SysTick_delay_ms(50);
}
```

```
/// ************| Intro_Screen |*********************
/// * Brief: Displays names and the words
/// *   slot machine when devices is powered on.
/// *   Display changes to menu after 3 second delay
/// * Param:
/// *      N/A
/// * Return:
/// *      N/A
/// ************************************************/
void Intro_Screen(void){

   lcdWriteCmd(DISPLAY_100);   //Turning off cursor
   SysTick_delay_us(10);

   lcdSetText("Josh Johnston", 2, 0);
   lcdSetText("Jordan Hayes", 2, 1);
   lcdSetText("SLOT MACHINE", 2, 3);
   SysTick_delay_ms(3000);     //Delays screen for 3 seconds

   lcdWriteCmd(CLEAR);         //Clears Display
   lcdWriteCmd(HOME);          //Sets cursor back to home position
   SysTick_delay_us(10);
}

/// ************| Menu_Options |********************
/// * Brief: Allows user to navigate through
/// *        the different menu screens from
/// *        Menu screen. Uses Keypad
/// *        initialized is SysTick_Library.c
/// *
/// * Global Variables:
/// *         i, num
/// * Return:
/// *      N/A
/// ************************************************/
 void Menu_Options(void){

   if(i == 0 && Read_Keypad()){    //Condition for i variable value and reads for keypad entry

    if( i != 3 && (BetUp == 1)){
       Bet = 0;
       BetUp = 0;               //Error checking interrupts
       BetDown = 0;               //Resetting if play screen is not present
       Start = 0;
  }
```

```
      if(num == 1){             //Goes to sounds menu when num == 1 and i == 0

       lcdWriteCmd(CLEAR);      //Clears display
       lcdWriteCmd(HOME);       //Sets cursor home position
       SysTick_delay_ms(10);

       lcdSetText("1- Song 1 ", 0, 0);
       lcdSetText("2- Song 2", 0, 1);
       lcdSetText("3- Win Bells", 0, 2);
       lcdSetText("*- Confirm Sound", 0, 3);
       SysTick_delay_ms(50);
       i = 1;                   //Sets value of i to 1
      }

      else if(num == 2){        //Goes to Credit enter screen when num == 2 and i == 0

       lcdWriteCmd(CLEAR);      //Clears display
       lcdWriteCmd(HOME);       //Sets cursor to home position
       SysTick_delay_us(10);

       lcdSetText("Enter Credits: ", 0, 0);
       lcdSetText("* = Cancel", 0, 1);
       lcdSetText("# = Accept", 0, 2);
       SysTick_delay_ms(10);
       i = 2;                   //Sets value of i to 2
      }

      else if(num  == 3){       //Condition for 3rd screen
         i = 3;                 //Sets i to 3
         Bet = 0;               //Clears bet
         lcdWriteCmd(CLEAR);    //Clears display
         lcdWriteCmd(HOME);     //Sets cursor to home position
         SysTick_delay_us(10);

      }
    }
}
```

```
/// ************| Bet_Input |*******************
/// * Brief: Stores user bet when conditions
/// *         are met. Uses port interrupts
/// *         initialized in SysTick_Library.c
/// *         on Bet-up and Bet-down buttons
/// *
/// * Global Variables:
/// *         BetUp, BetDown, Bet
/// ****Note: BetUp and BetDown are used as port
/// *    interrupts initialized in SysTick_Library.c
/// * Return:
/// *      N/A
/// ********************************************/
void Bet_Input(void){

  if(BetUp == 1 && Bet < 5 && i == 3){             //Condition for Bet to increment, checks
interrupt and bet value
       SysTick_delay_ms(350);                  //Delays 350ms for debounce purposes

          if(BetUp == 1 && i == 3){              //After delay value of interrupt flag is checked again
            Bet ++;                          //If conditions are met bet is incremented
            BetUp = 0;                         //Interrupt flag is reset
          }
     }

   else if(BetDown == 1 && Bet > 0 && i == 3){     //Conditions for decreasing bet
          SysTick_delay_ms(350);                  //Delays 350ms for debounc

          if(BetDown == 1 && i == 3){             //Checks interrupt flag value again
            Bet--;                           //If conditions are met bet is subtracted by one
            BetDown = 0;                         //Bet down flag is reset
           }
       }
}

/// ************| Home_Option |*******************
/// * Brief: Navigation back to menu screen while
/// *        on any screen by inputing the * on
/// *        the Keypad
/// *
/// * Global Variables:
/// *         num, i, n
/// * Return:
/// *      N/A
/// ********************************************/
void Home_Option(void){
```

```
    if(num == 10){              //Condition if * is pressed

      lcdWriteCmd(CLEAR);       //Clears display
      lcdWriteCmd(HOME);        //Sets cursor to home position
      SysTick_delay_us(10);

      Menu_Screen();            //Displays menu screen on LCD
       i = 0;                   //Resets i to zero
       n = 0;                   //Resets n to zero
    }
}

/// ************| Lose_Lights |*******************
/// * Brief: Displays losing LED sequence when
/// *        characters don't match
/// *
/// * Global Variables:
/// *         N/A
/// * Return:
/// *      N/A
/// ************************************************/
void Lose_Lights(void){

    Speaker(A_4,2);
    P3->OUT |= (BIT5);
    SysTick_delay_ms(500);
    Speaker(0,0);
    SysTick_delay_ms(100);

    P3->OUT = 0;
    P3->OUT |= BIT6;
    Speaker(E_4, 2);
    SysTick_delay_ms(500);
    Speaker(0,0);
    SysTick_delay_ms(200);

    P3->OUT = 0;                    //Turns off LEDs
    P3->OUT |= (BIT5 | BIT6 | BIT7);  //Turns red LEDs on
    Speaker(C_4, 2);
    SysTick_delay_ms(1000);

    Speaker(0,0);
    P3->OUT = 0;

  }
```

```
/// ************| Play_Game |*******************
/// * Brief: Finite State Machine, this allows
/// *        for slot machine reel to randomize.
/// *        As well, displays outcome of reel
/// *        with display message, credit update,
/// *        bet clear, and LED sequence
/// *
/// * Global Variables:
/// *         Reel_state, Credits, Bet, Start, play
/// ****Note: Start is used as a port interrupt
/// *          initialized in SysTick_Library.c
/// * Return:
/// *      N/A
/// ***********************************************/
 void Play_Game(void){

   uint8_t a, b, c, e, f, g;
   char Symbols[5] = {Diamond_index, Stick_Person_index, Dollar_index};
   char row1_1, row1_2, row1_3;
   char row2_1, row2_2, row2_3;



 switch(Reel_state){                    //Finite State Machine

   case DISPLAY_SCREEN:

     if(ButtonPress == 1){               //Checks for keypad press with port interrupt
       Read_Keypad();                   //Reads keypad function for number

       if(num == 10){                  //Condition to go back to menu screen

          lcdWriteCmd(CLEAR);           //Clears display
          lcdWriteCmd(HOME);            //Sets cursor to home position
          SysTick_delay_ms(50);
          Menu_Screen();                //Goes to menu screen
          SysTick_delay_ms(10);
          ButtonPress = 0;              //Resets keypad interrupt flag
          Bet = 0;                    //Resets Bet back to zero
```

```
        }
    }

     if(num == 3 && i == 3){              //Condition to print the play screen to LCD

        lcdSetText("GOOD", 0 , 0);
        lcdSetText("LUCK!", 0, 1);
        Characters();                    //Displays special characters
        lcdSetText("CR", 0, 2);
        lcdSetText("MENU", 6, 2);
        lcdSetText("BET", 12, 2);
        lcdSetInt(Credits, 0,3);
        lcdSetInt(Bet, 13, 3);
        lcdSetText("*", 7, 3);
        SysTick_delay_ms(25);
    }

     Reel_state = INPUT_BET;
     break;


    case INPUT_BET:

     if(Credits > 0 && i == 3){           //Inputting bet condition

        Bet_Input();                     //Goes into bet input function
        lcdSetInt(Bet, 13, 3);           //Displays updated bet value
        SysTick_delay_ms(10);
    }

     Reel_state = SPIN_RESULTS;
     break;


    case SPIN_RESULTS:

       if(Credits > 0 && Credits >= Bet && Bet > 0){

          if(Start == 1){                //If conditions are met game can start if interrupt flag is
triggered
           SysTick_delay_ms(250);        //Delays 250ms for debounce purposes

          if(Start == 1)                 //Reads flag value again

        for(j = 0; j < 30; j++){         //For loop for timing == approx 5 seconds
           Start = 0;                    //Resets start variable
```

```
        Speaker(D_4,2);              //Plays note frequency D_4
        SysTick_delay_ms(60 + (j * 2)); //Delay increases as time increases to hold note longer
for slot machine affect
        Speaker(A_4, 2);             //Plays note A_4 frequency
        SysTick_delay_ms(60 + (j * 2)); //Delay increases as time increases

         a = rand() % 3;             //Setting variables equal to a random number generator
         b = rand() % 3;             //To build a random number sequence
         c = rand() % 3;             //Which will then randomize the position of each character

         e = rand() % 3;
         f = rand() % 3;
         g = rand() % 3;

         row1_1 = Symbols[a];        //Sets char variable to an array set equal to the characters
         row1_2 = Symbols[b];        //The location is determined by the random number
generated
         row1_3 = Symbols[c];

         row2_1 = Symbols[e];
         row2_2 = Symbols[f];
         row2_3 = Symbols[g];

         lcdSetChar(row1_1, 6, 0);    //Prints each character based off their random array
position
         lcdSetChar(row1_2, 8, 0);
         lcdSetChar(row1_3, 10, 0);
         SysTick_delay_us(100);

         lcdSetChar(row2_1, 6, 1);
         lcdSetChar(row2_2, 8, 1);
         lcdSetChar(row2_3, 10, 1);
         SysTick_delay_us(100);



         if(Start == 1){             //Condition to stop the randomization and leave the for loop if
spin button is pressed
            SysTick_delay_ms(250);   //Delays 250ms for debouncing
             if(Start == 1)          //Checks flag value again
               play = 1;             //Sets play to 1 to leave the loop
            break;                   //Break out statement used to leave
         }

      }
```

```
  }

    if(play == 1 || j == 30){          //Condition for if either button was pressed or loop ended

      if((a == b && b == c) && (e == f && f == g)){
                                  //Both rows have wins condition
        Speaker(0,0);

        Credits = Credits + (Bet * Bet);  //Credits are now credits plus bet times itself due to
multiple win spin
        Start = 0;                  //Spin flag is reset

        lcdSetChar(row1_1, 6, 0);
        lcdSetChar(row2_1, 6, 1);
        lcdSetText(" ", 8, 0);
        lcdSetText(" ", 8, 1);
        lcdSetText(" ", 10, 0);
        lcdSetText(" ", 10, 1);
        Speaker(C_4,2);
        SysTick_delay_ms(300);

        Speaker(0,0);
        SysTick_delay_ms(100);
        lcdSetChar(row1_1, 6, 0);
        lcdSetChar(row2_1, 6, 1);
        lcdSetChar(row1_2, 8, 0);
        lcdSetChar(row2_2, 8, 1);
        lcdSetText(" ", 10, 0);
        lcdSetText(" ", 10, 1);
        Speaker(E_4, 2);
        SysTick_delay_ms(300);

        Speaker(0,0);
        SysTick_delay_ms(200);
        lcdSetChar(row1_1, 6, 0);
        lcdSetChar(row2_1, 6, 1);
        lcdSetChar(row1_2, 8, 0);
        lcdSetChar(row2_2, 8, 1);
        lcdSetChar(row1_3, 10, 0);
        lcdSetChar(row2_3, 10, 1);
        Speaker(G_4,2);
        SysTick_delay_ms(300);
        Speaker(0,0);

        lcdWriteCmd(CLEAR);          //Clears display
```

```
            lcdWriteCmd(HOME);              //Sets cursor to home position
            SysTick_delay_ms(50);

            lcdSetText("CONGRADULATIONS!", 0, 0);
            lcdSetText("YOU WIN: ",0, 2);
            lcdSetInt((Bet * Bet), 0, 3);
            lcdSetText("Credits", 3, 3);

            if(song == 1){
               Pirates_C();
            }
            else if(song == 2){
               Despacito();
            }
            else if(song == 3){
               Bells();
            }

           lcdWriteCmd(CLEAR);
           lcdWriteCmd(HOME);
           SysTick_delay_us(10);

           Bet = 0;
           play = 0;
           j = 0;
    }


           else if(a == b && b == c){   //If first row number sequence matches it is a win

            Speaker(0,0);               //Turn off speakers

            Credits += (Bet * 2);        //New value of credits is credits + bet * 2 because of
winnning sequence
            Start = 0;                  //Start interrupt flag is reset

            lcdSetChar(row1_1, 6, 0);     //Display the first column of characters for slot machine
effect
            lcdSetChar(row2_1, 6, 1);
            lcdSetText(" ", 8, 0);
            lcdSetText(" ", 8, 1);
            lcdSetText(" ", 10, 0);
            lcdSetText(" ", 10, 1);
            Speaker(C_4,2);             //Play C_4 note frequency
            SysTick_delay_ms(300);       //For 300ms
```

```
        Speaker(0,0);               //Turn speaker off
        SysTick_delay_ms(100);      //Delay 100ms before displaying next column
        lcdSetChar(row1_1, 6, 0);
        lcdSetChar(row2_1, 6, 1);
        lcdSetChar(row1_2, 8, 0);   //Display the first and second column of characters
        lcdSetChar(row2_2, 8, 1);
        lcdSetText(" ", 10, 0);
        lcdSetText(" ", 10, 1);
        Speaker(E_4, 2);            //Play E_4 note frequency
        SysTick_delay_ms(300);      //Delay 300ms

        Speaker(0,0);               //Turn speaker off
        SysTick_delay_ms(200);      //Delay for 200ms gradually display slower for slot
machine appearance
        lcdSetChar(row1_1, 6, 0);
        lcdSetChar(row2_1, 6, 1);
        lcdSetChar(row1_2, 8, 0);
        lcdSetChar(row2_2, 8, 1);
        lcdSetChar(row1_3, 10, 0);   //Display 3rd column of characters
        lcdSetChar(row2_3, 10, 1);
        Speaker(G_4,2);             //Play G_4 note frequency
        SysTick_delay_ms(300);      //Delay for 300ms

        Speaker(0,0);               //Turn Speaker off

        lcdSetText("YOU",12, 0);
        SysTick_delay_us(10);
        lcdSetText("WIN!", 12, 1);  //Win condition so displays a you win message
        SysTick_delay_us(10);

        lcdWriteCmd(CLEAR);         //Clears display
        lcdWriteCmd(HOME);          //Sets cursor to home position
        SysTick_delay_ms(50);

        lcdSetText("CONGRADULATIONS!", 0, 0);
        lcdSetText("YOU WIN: ",0, 2);
        lcdSetInt((Bet * 2), 0, 3);
        lcdSetText("Credits", 3, 3);


        if(song == 1){

           Pirates_C();
        }

        else if(song == 2){
```

```
        Despacito();
    }

    else if(song == 3){

        Bells();
    }

    lcdWriteCmd(CLEAR);          //Clear display
    lcdWriteCmd(HOME);           //Return cursor to home position
    SysTick_delay_us(10);
    Bet = 0;                     //Reset Bet to zero
    play = 0;                    //Resets play value to zero
    j = 0;                      //Resets j back to zero to repeat loop

}

 else if(e == f && f == g){
                            //Same as top row except bottom row win condition
    Speaker(0,0);

    Credits += (Bet * 2);        //Bet is multiplied by 2 and added to credits
    Start = 0;                   //Reset start interrupt

    lcdSetChar(row1_1, 6, 0);
    lcdSetChar(row2_1, 6, 1);
    lcdSetText(" ", 8, 0);
    lcdSetText(" ", 8, 1);
    lcdSetText(" ", 10, 0);
    lcdSetText(" ", 10, 1);
    Speaker(C_4,2);
    SysTick_delay_ms(300);

    Speaker(0,0);
    SysTick_delay_ms(100);
    lcdSetChar(row1_1, 6, 0);
    lcdSetChar(row2_1, 6, 1);
    lcdSetChar(row1_2, 8, 0);
    lcdSetChar(row2_2, 8, 1);
    lcdSetText(" ", 10, 0);
    lcdSetText(" ", 10, 1);
    Speaker(E_4, 2);
    SysTick_delay_ms(300);

    Speaker(0,0);
```

```
SysTick_delay_ms(200);
lcdSetChar(row1_1, 6, 0);
lcdSetChar(row2_1, 6, 1);
lcdSetChar(row1_2, 8, 0);
lcdSetChar(row2_2, 8, 1);
lcdSetChar(row1_3, 10, 0);
lcdSetChar(row2_3, 10, 1);
Speaker(G_4,2);
SysTick_delay_ms(300);

Speaker(0,0);

lcdSetText("YOU",12, 0);
SysTick_delay_ms(10);
lcdSetText("WIN!", 12, 1);
SysTick_delay_ms(10);

lcdWriteCmd(CLEAR);             //Clears display
lcdWriteCmd(HOME);              //Sets cursor to home position
SysTick_delay_ms(50);

lcdSetText("CONGRADULATIONS!", 0, 0);
lcdSetText("YOU WIN: ",0, 2);
lcdSetInt((Bet * 2), 0, 3);
lcdSetText("Credits", 3, 3);

if(song == 1){

   Pirates_C();
 }

else if(song == 2){

   Despacito();

}

else if(song == 3){

     Bells();
}

lcdWriteCmd(CLEAR);
lcdWriteCmd(HOME);
SysTick_delay_us(10);
Bet = 0;
```

```
        play = 0;
        j = 0;

}


  else{                         //Condition if no rows match

        Speaker(0,0);               //Turn speaker off

        Credits -= Bet;             //Loss so credits equal credits minus bet
        Bet = 0;                   //Bet is reset
        Start = 0;                 //spin flag is reset

        lcdSetChar(row1_1, 6, 0);
        lcdSetChar(row2_1, 6, 1);
        lcdSetText(" ", 8, 0);
        lcdSetText(" ", 8, 1);
        lcdSetText(" ", 10, 0);
        lcdSetText(" ", 10, 1);

        Speaker(C_4,2);
        SysTick_delay_ms(300);

        Speaker(0,0);
        SysTick_delay_ms(100);
        lcdSetChar(row1_1, 6, 0);
        lcdSetChar(row2_1, 6, 1);
        lcdSetChar(row1_2, 8, 0);
        lcdSetChar(row2_2, 8, 1);
        lcdSetText(" ", 10, 0);
        lcdSetText(" ", 10, 1);

        Speaker(E_4, 2);
        SysTick_delay_ms(300);

        Speaker(0,0);
        SysTick_delay_ms(200);
        lcdSetChar(row1_1, 6, 0);
        lcdSetChar(row2_1, 6, 1);
        lcdSetChar(row1_2, 8, 0);
        lcdSetChar(row2_2, 8, 1);
        lcdSetChar(row1_3, 10, 0);
        lcdSetChar(row2_3, 10, 1);

        Speaker(G_4,2);
```

```
            SysTick_delay_ms(300);

            Speaker(0,0);

            lcdSetText("BET", 12, 0);        //Bet lost message is displayed
            SysTick_delay_ms(10);
            lcdSetText("LOST", 12, 1);
            SysTick_delay_ms(10);

            Lose_Lights();                   //Red LED Sequence is displayed
            SysTick_delay_ms(500);
            lcdWriteCmd(CLEAR);
            lcdWriteCmd(HOME);
            SysTick_delay_us(10);

            play = 0;
            j = 0;
          }
       }
 }

  Reel_state = DISPLAY_SCREEN;
  break;
  }
}

/// ************| Spin_and_Play |*****************
/// * Brief: Sets condition to go enter play
/// *              game function.
/// *
/// * Global Variables:
/// *         N/A
/// * Return:
/// *      N/A
/// ***********************************************/
void Spin_and_Play(void){

    if(i == 3){
     Play_Game();
    }
}

/// **********| Sounds_Options |**************
/// * Brief: Used as sound preview
/// *        and win sound selection
/// *
```

```
/// * Global Variables:
/// *        i, num
/// * Return:
/// *      N/A
/// ****************************************/
 void Sounds_Options(void){

   if(i == 1 && Read_Keypad()){     //Reads keypad and looks for number

     if(num == 1){               //condition to play first sound

       Pirates_C();              //Plays song 1
       song = 1;                 //Sets global variable to one
     }

     else if(num == 2){          //Condition from num 2

       Despacito();              //Plays song 2
       song = 2;                 //Sets song to 2
      }

     else if(num == 3){          //Third song option

      Bells();                   //Plays song 3
      song = 3;                  //Sets value of song to 3
     }
   }
}

/// ***************| Credits_Input |*******************
/// * Brief: Stores user keypad credit input
/// *        when acceptance condition is met.
/// *        value is then stored in variable Credits
/// *
/// *Global Variables:
/// *       i, ButtonPress, num, x, n, one, two, three
/// ****Note: ButtonPress is used as port interrupt for
/// *      keypad initialized in SysTick_Libraries.c
/// * Return:
/// *      N/A
/// ****************************************************/
void Credits_Input(void){

   if(i == 2 && Read_Keypad()){          //Looks for i value and value from keypad

     if(ButtonPress == 1) n++;          //If keypad flag is triggered n is incremented
```

```
        if(num == 11){

           num = 0;                 //Sets num to the value of 0 when it is pressed on keypad
           SysTick_delay_us(10);
        }

        if(n == 1 && (num == 12))       //Error check if pound is pressed but no number before the
value will not display

           x = 0;

        if(n == 1 && (num != 12)){      //If digit is entered and pound is not pressed

           lcdSetInt(num, 0, 3);       //Num will display on LCD
           one = num;                  //Variable one set to hold the value of num
           x = 1;                   //A variable is set to one to show it was the first value pressed
        }

        if(n == 2 && (num != 12)){      //If a second digit was entered and pound was not pressed

           lcdSetInt(num, 1, 3);       //The num will be displayed next to the first entry
           two = num;                  //A second variable is set equal to the num
           x = 2;                   //Place holder variable is used to identify which order number was
pressed
        }

        if(n == 3 && (num != 12)){      //Condition for third digit being pressed

           lcdSetInt(num, 2, 3);       //Displays number next to second digit
           three = num;                //Third variable used to hold value of num
           x = 3;                   //Place holder variable used for digit position
        }
      }

        if(num == 12 && x == 1){        //First condition if one number was pressed and pound
was used to accept the value

           Credits = one;              //Credits is set equal to the value of num which was the first
number pressed

          lcdWriteCmd(CLEAR);          //Clears Display
          lcdWriteCmd(HOME);           //Sets cursor back to home position
          SysTick_delay_ms(10);
          Menu_Screen();               //Displays menu screen showing user their credit value has
been accepted
```

```
        SysTick_delay_ms(10);

        i = 0;                  //Resets i
        n = 0;                  //Resets n
        x = 0;                  //Resets x
      }

      if(num == 12 && x == 2){      //Condition for a 2 number entry

        Credits = (one * 10) + two; //First value is multiplied by 10 and added to the second
value entered

        lcdWriteCmd(CLEAR);     //Clears display
        lcdWriteCmd(HOME);      //Returns cursor to home position
        SysTick_delay_us(40);
        Menu_Screen();          //Goes to menu screen to show user credit value has been
accepted
        SysTick_delay_us(40);

        i = 0;                  //Resets i
        n = 0;                  //Resets n
        x = 0;                  //Resets x
      }

      if(num == 12 && x == 3){      //Condition for a 3 digit number

        Credits = (one * 100) + (two * 10) + three;
                        //Multiplies first number by 100 second by 10 and third stays as ones
and adds all together
        lcdWriteCmd(CLEAR);     //Clears Display
        lcdWriteCmd(HOME);      //Returns cursor to home position
        SysTick_delay_ms(10);
        Menu_Screen();          //Returns to menu screen to show value was accepted
        SysTick_delay_us(10);

        i = 0;                  //Resets i
        n = 0;                  //Resets n
        x = 0;                  //Resets x
      }

      if(num == 10 && n > 0 && n < 4){
                        //Condition if a value was entered but * was pressed to cancel
        Credits = 0;            //Resets credits

        Menu_Screen();          //Returns to menu screen
        SysTick_delay_us(10);
```

```
            i = 0;                  //Resets i
            n = 0;                  //Resets n
            x = 0;                  //Resets x
           Credits = 0;               //Confirms credits reset

        }

        else if(n > 4){             //Condition if a number greater than 999 or more then 4 keys
were pressed

            lcdWriteCmd(CLEAR);
            lcdWriteCmd(HOME);
            SysTick_delay_ms(10);

            lcdSetText("LIMIT EXCEEDED", 1, 0);
            lcdSetText("ENTER AMOUNT", 2, 1);
            lcdSetText("WITHIN RANGE:", 2, 2);
            lcdSetText("0 - 999", 4, 3);
            SysTick_delay_ms(5000);

            n = 0;                  //Resets n
            x = 0;                  //Resets x

            lcdWriteCmd(CLEAR);
            lcdWriteCmd(HOME);
            SysTick_delay_ms(10);
                                //Goes back to credit screen when message of exceeding range is
removed
            lcdSetText("Enter Credits: ", 0, 0);
            lcdSetText("* = Cancel", 0, 1);
            lcdSetText("# = Accept", 0, 2);
            SysTick_delay_us(10);

            i = 2;                  //Sets i to 2 to confirm credit screen is valid
      }
}

/// ****************| Cash_Out_Lights |*******************
/// * Brief: Flashes lights after cash out song and
/// *       lights finish
/// *
/// *Global Variables:
/// *       N/A
/// * Return:
/// *      N/A
```

```
/// ****************************************************/
void Cash_Out_Lights(void){

   P3->OUT |= (BIT0 | BIT2 | BIT3 | BIT5 | BIT6 | BIT7);
   SysTick_delay_ms(250);
   P3->OUT = 0;
   SysTick_delay_ms(250);
   P3->OUT |= (BIT0 | BIT2 | BIT3 | BIT5 | BIT6 | BIT7);
   SysTick_delay_ms(250);
   P3->OUT = 0;
   SysTick_delay_ms(250);
   P3->OUT |= (BIT0 | BIT2 | BIT3 | BIT5 | BIT6 | BIT7);
   SysTick_delay_ms(250);
   P3->OUT = 0;
   SysTick_delay_ms(250);
   P3->OUT |= (BIT0 | BIT2 | BIT3 | BIT5 | BIT6 | BIT7);
   SysTick_delay_ms(250);
   P3->OUT = 0;
   SysTick_delay_ms(250);
   P3->OUT |= (BIT0 | BIT2 | BIT3 | BIT5 | BIT6 | BIT7);
   SysTick_delay_ms(250);
   P3->OUT = 0;
   SysTick_delay_ms(250);
   P3->OUT |= (BIT0 | BIT2 | BIT3 | BIT5 | BIT6 | BIT7);
   SysTick_delay_ms(250);
   P3->OUT = 0;
   SysTick_delay_ms(250);
   P3->OUT |= (BIT0 | BIT2 | BIT3 | BIT5 | BIT6 | BIT7);
   SysTick_delay_ms(1000);
   P3->OUT = 0;
   SysTick_delay_ms(1000);
}

/// ***************| Servo |*******************
/// * Brief: Moves the servo to drop the coin
/// *        out of the coin box
/// *
/// *Global Variables:
/// *       N/A
/// * Return:
/// *      N/A
/// ****************************************************/
void Servo(void){

      P8->SEL0 |= BIT2;            //P8.2 set to TA3.2
      P8->SEL1 &= ~BIT2;           //GPIO
```

```
    P8->DIR |= BIT2;            //P8.2 Set to Output

    TIMER_A3->CCR[0] = 6000;     //Period to be set between 1 and 2ms
    TIMER_A3->CCTL[2] = TIMER_A_CCTLN_OUTMOD_7;   //CCR1 reset
    TIMER_A3->CCR[2] = 0;        //PWM duty cycle
    TIMER_A3->CTL = 0x0214;      //SMCLK, Up Mode, Clear TAR to start
}




/// **************| Cash_Out |******************
/// * Brief: Displays cash out screen when user
/// *        pushes cash out button. Cash out
/// *        displays how many credits remain
/// *        and plays a song and light show.
/// *        As well, resets credits back to 0.
/// *
/// * Global Variables:
/// *         CashOut, Credits
/// ****Note: CashOut is used as a port interrupt
/// *         initialized in SysTick_libraries.c
/// * Return:
/// *      N/A
/// ********************************************/
void Cash_Out(void){

   if(CashOut == 1){                //If cashout button interrupt is triggered
      SysTick_delay_ms(250);          //Delay for debounce

      if(CashOut == 1)               //Check flag value again

         lcdWriteCmd(CLEAR);          //Clear display
         lcdWriteCmd(HOME);           //Return cursor to home position

         lcdSetText("Cash Out:", 0, 0);
         lcdSetInt(Credits, 10, 0);     //Display last recorded credits value
         SysTick_delay_ms(10);

         lcdSetText("Thanks 4 Playing", 0, 2);
         SysTick_delay_ms(10);

         lcdSetText("Slot Machine", 2, 3);
         SysTick_delay_ms(10);
         Pirates_C();
```

```
        Cash_Out_Lights();          //Displays cashout lights
        TIMER_A3->CCR[2] = 6000;    //Rotates servo 180 degrees counterclockwise to open
door
        SysTick_delay_ms(2000);     //Delays for 2 seconds to allow full opening of the door
        TIMER_A3->CCR[2] = 3000;    //Rotates servo 180 degrees clockwise to close door
        SysTick_delay_ms(3000);     //Delays 3 seconds to confirm secure shut
        TIMER_A3->CCR[2] = 0;       //Turns servo off

        Credits = 0;                //Resets credit to zero
        i = 0;                      //Resets i to zero

        lcdWriteCmd(CLEAR);         //Clears display
        lcdWriteCmd(HOME);          //Sets cursor back to home position
        SysTick_delay_us(10);

        Menu_Screen();              //Returns to menu screen
        SysTick_delay_ms(10);

        CashOut = 0;                //Resets cashout interrupt flag

    }
}
```

Slot_Machine.h

```
/*****************************************************************************
 *****************************************************************************
 *                    Slot_Machine.c
 *            Joshua Johnston and Jordan Hayes
 *                      EGR226
 *            Instructor: Dr. Nabeeh Kandalaft
 *              Created on: March 26, 2019
 *
 *       For use with the MSP432 LaunchPad Development Board
 *
 * Library is used for a custom slot machine gives functions
 * and provides menu screens and functionality for
 * user friendly game play
 *
 * The /// notation makes it so the function description block
 * is visible when you hovering over a function call in any
 * file (this feature is called Intellisense).
 *
 * Functions are each briefly described in comment blocks
 * Functions are commented out throughout program
 * Devices used and pin descriptions are listed below
 *
 * Libraries needed for initializations and other screen functionalities
 *                    LCD_library.h
 *                    SysTick_Library.h
 *
 ********************** DEVICES AND PARTS ***********************************
 * 16 x 4 HHD44780 LCD DISPLAY
 * STANDARD 12 KEY KEYPAD
 * 4 MOMENTARY PUSH BUTTONS
 * 6 LEDs
 * 7 SEGMENT DISPLAY
 * AUDIO SPEAKER
 *
 * Reference Pin Description Below

*****************************************************************************/

#ifndef SLOT_MACHINE_H_
#define SLOT_MACHINE_H_

/********************** PIN DESCRIPTIONS ***********************************
 *
 *          ********* 16 X 4 LCD DISPLAY *********
```

```
*                P4.0 -> LCD D4
*                P4.1 -> LCD D5
*                P4.2 -> LCD D6
*                P4.3 -> LCD D7
*                P4.4 -> LCD E
*                P4.5 -> LCD RS
*
*         ******* STANDARD 12 KEY KEYPAD *******
*                P5.4 -> ROW 0
*                P5.5 -> ROW 1
*                P5.6 -> ROW 2
*                P5.7 -> ROW 3
*                P2.5 -> COL 0
*                P2.6 -> COL 1
*                P2.7 -> COL 2
*
*        ************ PUSH BUTTONS *************
*                P1.5 -> SPIN
*                P1.6 -> BET-UP
*                P1.7 -> BET-DOWN
*                P6.0 -> CASHOUT
*
*        **************  LEDS  ******************
*                 (WIN LEDS)
*                P3.0 -> BLUE LED
*                P3.2 -> ORANGE LED
*                P3.3 -> YELLOW LED
*                 (LOSE LEDS)
*                P3.5 -> RED LED
*                P3.6 -> RED LED
*                P3.7 -> RED LED
*
*        ********** 7 SEGMENT DISPLAY ************
*               P7.0 -> Pin 1
*               P7.1 -> Pin 2
*               P7.2 -> Pin 4
*               P7.3 -> Pin 5
*               P7.4 -> Pin 6
*               P7.5 -> Pin 7
*               P7.6 -> Pin 9
*               P7.7 -> Pin 10
*
*        *********** AUDIO SPEAKER ************
*               P2.4 -> PWM Pin
*
*************************************************************************/
```

```
/*************Global Variables Used Throughout Libraries*****************
*************************************************************************/
extern uint8_t i, n, x, j;
extern int Credits;
extern int one, two, three;
extern uint8_t play;
/*************Global Variables Used Throughout Libraries*****************
*************************************************************************/

/********************** Function Prototypes ******************************
*************************************************************************/
void Menu_Screen(void);
void Intro_Screen(void);
void Menu_Options(void);
void Sounds_Options(void);
void Credits_Input(void);
void Bet_Input(void);
void Home_Option(void);
void Play_Game(void);
void Lose_Lights(void);
void Cash_Out(void);
void Cash_Out_Lights(void);
void Spin_and_Play(void);
void Servo(void);
/********************** Function Prototypes ******************************
*************************************************************************/

#endif /* SLOT_MACHINE_H_ */
```

LCD_Library.c

```
/*********************************************************************************
*********
*********************************************************************************
*********
*                       LCD_Library.c
*                  Trevor Ekin (Adapted from Dr. Nabeeh Kandalaft)
*                   EGR226      Date: February-21-2019
*                   (Updated by Josh Johnston Date: 04/02/2019)
*
*  This is a library for the 4x16 LCD.
*
*  All functions are briefly described in their comment blocks.  The /// notation makes
*  it so the function description block is visible when you hovering over a function call
*  in any file (this feature is called Intellisense).
*
*  All pins are set with default values (see below) but they can be easily changed with
*  in LCD_Library.h to any pin configuration (follow instructions in header file)
*
/// * UPDATE
 * Custom character function was added called: Character_init()
 * Initial location to display on 16 x 4 LCD for custom characters
 * Function called: Characters()
*******************************  Pins
*************************************************
*               MSP432 PINS (Default, see LCD_Library.h to change these)
*               P4.0 -> LCD D4
*               P4.1 -> LCD D5
*               P4.2 -> LCD D6
*               P4.3 -> LCD D7
*               P4.4 -> LCD E
*               P4.5 -> LCD RS
*
*********************************************************************************
********/

#include <stdint.h>
#include "LCD_Library.h"
#include <stdio.h>
#include "SysTick_Library.h"

/*************Global Variables Used Throughout Libraries*****************
*************************************************************************/
char Diamond_index;
char Arrows_index;
```

```
char Dollar_index;
char Stick_Person_index;
/*************Global Variables Used Throughout Libraries****************
**********************************************************************/

/// ****| lcdInit | *********************************//*
/// * Brief: Initialize the LCD with chosen connection
/// *         pins.  Send configuration sequence.
/// * param:
/// *      N/A
/// * return:
/// *      N/A
/// ************************************************/
void lcdInit() {
   // Initialize all communication pins as outputs (see LCD_Library.h for PDIR macro
definitions)
   RS_DIR |= RS;
   EN_DIR |= EN;
   D4_DIR |= D4;
   D5_DIR |= D5;
   D6_DIR |= D6;
   D7_DIR |= D7;

   // Initialize all communication pins as low (see LCD_Library.h for Bit Toggling macro
definitions)
   RS_LOW;
   EN_LOW;
   D4_LOW;
   D5_LOW;
   D6_LOW;
   D7_LOW;

   // Send initialization command sequence (see LCD_Library.h for command macro definitions)
   // Many init commands are sent multiple times for redundancy (takes time to latch in LCD)
   SysTick_delay_ms(100);

   // CLEAR command is 0x01
   lcdWriteCmd(CLEAR);        SysTick_delay_ms(100);  //clear screen

   //-------------------------------------------------------------------------------
   // FUNCTION SET COMMAND: 0b001[DL] [N][F] * * -> DL = Data length, N = display
lines, F = character font
   //FSET_000       0x20    -- 4-bit data, 1 line, 5x8 dot font
   //FSET_001       0x24    -- 4-bit data, 1 line, 5x10 dot font
   //FSET_010       0x28    -- 4-bit data, 2 lines, 5x8 dot font
   //FSET_011       0x2C    -- 4-bit data, 2 lines, 5x10 dot font
```

```
   //FSET_100       0x30    -- 8-bit data, 1 line, 5x8 dot font     (standard)
   //FSET_101       0x34    -- 8-bit data, 1 line, 5x10 dot font
   //FSET_110       0x38    -- 8-bit data, 2 lines, 5x8 dot font
   //FSET_111       0x3C    -- 8-bit data, 2 lines, 5x10 dot font
   //-----------------------------------------------------------------------------
   lcdWriteCmd(FSET_100);      SysTick_delay_ms(100);  //start reset
   lcdWriteCmd(FSET_100);      SysTick_delay_ms(100);  //reinforce reset
   lcdWriteCmd(FSET_100);      SysTick_delay_ms(100);  //reset captured
   lcdWriteCmd(FSET_100);      SysTick_delay_ms(100);  //send real "function set" call (same
as others but it is still needed)

   // HOME command is 0x02
   lcdWriteCmd(HOME);          SysTick_delay_ms(100);  //send cursor home

   //-----------------------------------------------------------------------------
   // ENTRY MODE COMMAND: 0b0000 01[I][S] -> I = Increment, S = Shift
   //ENTRYMODE_00    0x04    // no auto increment, no display shift  (DEFAULT)
   //ENTRYMODE_01    0x05    // no auto increment, display shift
   //ENTRYMODE_10    0x06    // auto increment, no display shift
   //ENTRYMODE_11    0x07    // auto increment, display shift
   //-----------------------------------------------------------------------------
   lcdWriteCmd(ENTRYMODE_10);  SysTick_delay_ms(100);  //set up for auto incrementing

   lcdWriteCmd(CLEAR);         SysTick_delay_ms(100);  //clear screen (again)

   //-----------------------------------------------------------------------------
   // DISPLAY CONTORL COMMAND: 0b0000 1[D][C][B] -> D = Display, C = Cursor, B =
Blinking
   //DISPLAY_000     0x08    -- display off, cursor off, blinking off    (DEFAULT)
   //DISPLAY_001     0x09    -- display off, cursor off, blinking on     (not useful)
   //DISPLAY_010     0x0A    -- display off, cursor on, blinking off     (not useful)
   //DISPLAY_011     0x0B    -- display off, cursor on, blinking on      (not useful)
   //DISPLAY_100     0x0C    -- display on, cursor off, blinking off
   //DISPLAY_101     0x0D    -- display on, cursor off, blinking on
   //DISPLAY_110     0x0E    -- display on, cursor on, blinking off
   //DISPLAY_111     0x0F    -- display on, cursor on, blinking on
   //-----------------------------------------------------------------------------
   lcdWriteCmd(DISPLAY_111);   SysTick_delay_ms(100);  //turn on display with blinking
cursor

   // reset CGRAM offset address
   _offset = 0;
}

/// ****| lcdTriggerEN | ****************************//*
/// * Brief: Pulse the enable pin to notify the LCD to
```

```
/// *         latch the current data inputs.
/// * param:
/// *      (unsigned char) data: 8-bit data to send
/// * return:
/// *      N/A
/// ******************************************************/
void lcdTriggerEN() {
   EN_HIGH;
   SysTick_delay_us(50);
   EN_LOW;
   SysTick_delay_us(50);
}

/// ****| lcdWriteData | ****************************//*
/// * Brief: Send data one nibble at a time to LCD via
/// *         SetNibble macro (see LCD_Library.h)
/// * param:
/// *      (unsigned char) data: 8-bit data to send
/// * return:
/// *      N/A
/// ******************************************************/
void lcdWriteData(unsigned char data) {
   RS_HIGH;
   SysTick_delay_us(50);
   SetNibble(data >> 4);   // Upper nibble
   SysTick_delay_us(50);
   lcdTriggerEN();
   SetNibble(data);        // Lower nibble
   SysTick_delay_us(50);
   lcdTriggerEN();
   SysTick_delay_us(50);
   SetNibble(0x00);        // clear output
}

/// ****| lcdWriteCmd | ****************************//*
/// * Brief: Send command one nibble at a time to LCD
/// *         via SetNibble macro (see LCD_Library.h)
/// * param:
/// *      (unsigned char) cmd: 8-bit command to send
/// * return:
/// *      N/A
/// ******************************************************/
void lcdWriteCmd(unsigned char cmd) {
   RS_LOW;
   SysTick_delay_us(50);
   SetNibble(cmd >> 4);     // Upper nibble
```

```
    SysTick_delay_us(50);
    lcdTriggerEN();
    SetNibble(cmd);          // Lower nibble
    SysTick_delay_us(50);
    lcdTriggerEN();
    SysTick_delay_us(50);
    SetNibble(0x00);         // clear output
}

/// ****| lcdSetText | ******************************//*
/// * Brief: Display character string on the LCD at the
/// *         chosen coordinates.
/// * param:
/// *      (char*) text: character string to display
/// *      (int) x:      x-coordinate
/// *      (int) y:      y-coordinate
/// * return:
/// *      N/A
/// ***************************************************/
void lcdSetText(char* text, int x, int y) {
    int i;
    if (x < 16) {
        x |= 0x80;      // Set LCD for first line write
        switch (y){
        case 0:
            x |= 0x00;  // Set LCD for first line write
            break;
        case 1:
            x |= 0x40;  // Set LCD for Second line write
            break;
        case 2:
            x |= 0x10;  // Set LCD for Third line write
            break;
        case 3:
            x |= 0x50;  // Set LCD for Fourth line write
            break;
        case 5:
            x |= 0x20;  // Set LCD for second line write reverse
            break;
        }
        lcdWriteCmd(x);
    }
    i = 0;
    while (text[i] != '\0') {
        lcdWriteData(text[i]);
        i++;
```

```
   }
}

/// ****| lcdSetChar | ******************************//*
/// * Brief: Display character on the LCD at the
/// *         chosen coordinates.
/// * param:
/// *      (char) c:    character to display (can be
/// *                   custom character if c = offset
/// *                   of custom character)
/// *      (int) x:     x-coordinate
/// *      (int) y:     y-coordinate
/// * return:
/// *      N/A
/// ******************************************************/
void lcdSetChar(char c, int x, int y) {
   if (x < 16) {
      x |= 0x80;      // Set LCD for first line write
      switch (y){
      case 0:
         x |= 0x00;  // Set LCD for first line write
         break;
      case 1:
         x |= 0x40;  // Set LCD for Second line write
         break;
      case 2:
         x |= 0x10;  // Set LCD for Third line write
         break;
      case 3:
         x |= 0x50;  // Set LCD for Fourth line write
         break;
      case 5:
         x |= 0x20;  // Set LCD for second line write reverse
         break;
      }
      lcdWriteCmd(x);
   }

   lcdWriteData(c);
}

/// ****| lcdSetInt | ******************************//*
/// * Brief: Convert integer into character string to be
/// *         displayed on LCD at chosen coordinates.
/// * param:
/// *      (int) val: value to convert to display
```

```
/// *      (int) x:   x-coordinate
/// *      (int) y:   y-coordinate
/// * return:
/// *      N/A
/// ************************************************/
void lcdSetInt(int val, int x, int y){
   char number_string[16];
   sprintf   (number_string, "%d\0", val); // Convert the integer to character string
   lcdSetText(number_string, x, y);
}

/// ****| lcdCreateCustomChar | **********************//*
/// * Brief: Creates a custom character in CGRAM based on
/// *         character structure passed.
/// * param:
/// *      (custom_char_t)* cc: custom character struct
/// *                           to place in CGRAM
/// * return:
/// *      (uint8_t) _offset:   offset index of new
/// *                             custom char
/// ************************************************/
uint8_t lcdCreateCustomChar(custom_char_t* cc) {
   lcdWriteCmd(CGRAM+(8*_offset));     // characters placed in intervals of 8 bytes
   lcdWriteData(cc->line0);          // send byte 0 of new character
   lcdWriteData(cc->line1);          // send byte 1 of new character
   lcdWriteData(cc->line2);          // send byte 2 of new character
   lcdWriteData(cc->line3);          // send byte 3 of new character
   lcdWriteData(cc->line4);          // send byte 4 of new character
   lcdWriteData(cc->line5);          // send byte 5 of new character
   lcdWriteData(cc->line6);          // send byte 6 of new character
   lcdWriteData(cc->line7);          // send byte 7 of new character
   return _offset++;                 // return then increment offset value for next character
}


/// ****| lcdClear | *********************************//*
/// * Brief: Clear all visible characters from the
/// *         screen.
/// * param:
/// *      N/A
/// * return:
/// *      N/A
/// ************************************************/
void lcdClear() {

   lcdWriteCmd(CLEAR);
```

```
   SysTick_delay_ms(10);
}

/// ************| Character_init | **********************
/// * Brief: Creates custom characters using structs
/// *        with specific binary numbers that indicates
/// *        specific bits to turn on
/// *
/// * Global Variables:
/// *      Diamond_index, Dollar_index, Stick_person_index
/// * return:
/// *      N/A
/// ****************************************************/
void Character_init(void){

 custom_char_t Diamond_layout = {

          0b00000,
          0b00100,
          0b01110,
          0b11111,
          0b01110,
          0b00100,
          0b00000,
          0b00000
    };

 custom_char_t Dollar_layout = {

          0b00100,
          0b01111,
          0b10100,
          0b01110,
          0b00101,
          0b11110,
          0b00100,
          0b00000
    };

 custom_char_t Stick_Person_layout = {

          0b01110,
          0b01110,
          0b00100,
          0b11111,
          0b00100,
```

```
        0b00100,
        0b01010,
        0b10001

    };


 Diamond_index  = lcdCreateCustomChar(&Diamond_layout);           //Sets the address of
Diamond_layout into Diamond_index
 Dollar_index  = lcdCreateCustomChar(&Dollar_layout);             //Sets the address of
Dollar_layout into Dollar_index
 Stick_Person_index  = lcdCreateCustomChar(&Stick_Person_layout);   //Sets the address of
Stick_Person_layout into Stick_Person_index

}

/// **************| Characters | **********************
/// * Brief: Sets initial position of custom characters
/// *        with a good luck message before.
/// *
/// * Global Variables:
/// *     N/A
/// * return:
/// *      N/A
/// ******************************************************/
void Characters(void){

   lcdSetText("GOOD", 0 ,0);
   lcdSetText("LUCK!", 0, 1);
   SysTick_delay_us(1);

   lcdSetChar(Diamond_index, 6, 0);
   lcdSetChar(Stick_Person_index, 8, 0);
   lcdSetChar(Dollar_index, 10, 0);

   lcdSetChar(Diamond_index, 6, 1);
   lcdSetChar(Dollar_index, 8, 1);
   lcdSetChar(Dollar_index, 10, 1);

}
```

LCD_Library.h

```
/*****************************************************************************
*********
*****************************************************************************
*********
*                       LCD_Library.c
*                       Trevor Ekin (Adapted from Dr. Nabeeh Kandalaft)
*                  EGR226      Date: February-21-2019
*                  (Updated by Josh Johnston Date: 04/02/2019)
*
*  This is a library for the 4x16 LCD.
*
*  All functions are briefly described in their comment blocks.  The /// notation makes
*  it so the function description block is visible when you hovering over a function call
*  in any file (this feature is called Intellisense).
*
*  All pins are set with default values (see below) but they can be easily changed with
*  in LCD_Library.h to any pin configuration (follow instructions in header file)
*
/// * UPDATE
 * Custom character function was added called: Character_init()
 * Initial location to display on 16 x 4 LCD for custom characters
 * Function called: Characters()
*****************************************************************************
**********/

#ifndef LCD_LIBRARY_H_
#define LCD_LIBRARY_H_

//#include "driverlib.h"  // for use with driverlib
#include "msp.h"
#include <stdint.h>

/************  Pins   ********************************
*         MSP432 PINS (Default)
* P4.0 -> LCD D4
* P4.1 -> LCD D5
* P4.2 -> LCD D6
* P4.3 -> LCD D7
* P4.4 -> LCD E
* P4.5 -> LCD RS
* NOTICE:
*   This section is configurable and dynamic.  If you
*   would like to use different pins, make the swaps
*   here.  For example, if you would like D5 to be on
```

```
*   P2.4 instead of P4.1 make following changes:
*       D5_DIR P2DIR
*       D5_OUT P2OUT
*       D5 BIT4
* ************************************************/

// direction registers (modify as needed)
#define EN_DIR  P4DIR
#define RS_DIR  P4DIR
#define D4_DIR  P4DIR
#define D5_DIR  P4DIR
#define D6_DIR  P4DIR
#define D7_DIR  P4DIR

// Port registers (modify as needed)
#define EN_OUT  P4OUT
#define RS_OUT  P4OUT
#define D4_OUT  P4OUT
#define D5_OUT  P4OUT
#define D6_OUT  P4OUT
#define D7_OUT  P4OUT

// Pin BITs (modify as needed)
#define EN  BIT4
#define RS  BIT5
#define D4  BIT0
#define D5  BIT1
#define D6  BIT2
#define D7  BIT3

// Bit Toggling (DO NOT CHANGE)
#define EN_LOW (EN_OUT &= ~EN)
#define EN_HIGH (EN_OUT |= EN)
#define RS_LOW (RS_OUT &= ~RS)
#define RS_HIGH (RS_OUT |= RS)
#define D4_LOW (D4_OUT &= ~D4)
#define D4_HIGH (D4_OUT |= D4)
#define D5_LOW (D5_OUT &= ~D5)
#define D5_HIGH (D5_OUT |= D5)
#define D6_LOW (D6_OUT &= ~D6)
#define D6_HIGH (D6_OUT |= D6)
#define D7_LOW (D7_OUT &= ~D7)
#define D7_HIGH (D7_OUT |= D7)

/// represent provided nibble on data lines use ternary statements and multi-line macro
/// *  note: ternary statements are like an "if / else" statement.
```

```
/// *      if/else example:
/// *
/// *      if(x & 0x01) {
/// *         D4_HIGH;
/// *      } else {
/// *         D4_LOW;
/// *      }
/// *
/// *      Ternary version with exact same result:
/// *      (x & 0x01) ? D4_HIGH : D4_LOW;
/// */
#define SetNibble(x) \
   ((x & 0x01) ? D4_HIGH : D4_LOW); \
   ((x & 0x02) ? D5_HIGH : D5_LOW); \
   ((x & 0x04) ? D6_HIGH : D6_LOW); \
   ((x & 0x08) ? D7_HIGH : D7_LOW);


/************************** Commands ***********************************
*****************************************************************************/
#define CLEAR      0x01
#define HOME       0x02

// ENTRY MODE COMMAND: 0b0000 01[I][S] -- I = Increment, S = Shift
#define ENTRYMODE_00    0x04    // no auto increment, no display shift  (DEFAULT)
#define ENTRYMODE_01    0x05    // no auto increment, display shift
#define ENTRYMODE_10    0x06    // auto increment, no display shift
#define ENTRYMODE_11    0x07    // auto increment, display shift

// DISPLAY CONTORL COMMAND: 0b0000 1[D][C][B] -- D = Display, C = Cursor, B =
Blinking
#define DISPLAY_000     0x08    // display off, cursor off, blinking off    (DEFAULT)
#define DISPLAY_001     0x09    // display off, cursor off, blinking on     (not useful)
#define DISPLAY_010     0x0A    // display off, cursor on, blinking off     (not useful)
#define DISPLAY_011     0x0B    // display off, cursor on, blinking on      (not useful)
#define DISPLAY_100     0x0C    // display on, cursor off, blinking off
#define DISPLAY_101     0x0D    // display on, cursor off, blinking on
#define DISPLAY_110     0x0E    // display on, cursor on, blinking off
#define DISPLAY_111     0x0F    // display on, cursor on, blinking on

// CURSOR/DISPLAY SHIFT COMMAND: 0b0001 [DC][RL] * * -- DC = Display or Cursor,
RL = Right or Left, * = don't care
#define SHIFT_00        0x10;   // cursor shift to the left
#define SHIFT_01        0x14;   // cursor shift to the right
#define SHIFT_10        0x18;   // display shift to the left
#define SHIFT_11        0x1C;   // display shift to the right
```

```
// FUNCTION SET COMMAND: 0b001[DL] [N][F] * * -- DL = Data length, N = display lines,
F = character font
#define FSET_000       0x20    // 4-bit data, 1 line, 5x8 dot font
#define FSET_001       0x24    // 4-bit data, 1 line, 5x10 dot font
#define FSET_010       0x28    // 4-bit data, 2 lines, 5x8 dot font
#define FSET_011       0x2C    // 4-bit data, 2 lines, 5x10 dot font
#define FSET_100       0x30    // 8-bit data, 1 line, 5x8 dot font     (standard)
#define FSET_101       0x34    // 8-bit data, 1 line, 5x10 dot font
#define FSET_110       0x38    // 8-bit data, 2 lines, 5x8 dot font
#define FSET_111       0x3C    // 8-bit data, 2 lines, 5x10 dot font

// CGRAM ADDRESS
#define CGRAM          0x40    // start address for CGRAM data, custom graphics

/************************** Commands *********************************
*********************************************************************/

/********************** Structure Definitions ***************************
*********************************************************************/
/// custom_char_t is a struct containing 8 bytes of data, representing the
///      8 rows of dots that create an LCD character. Each bit is either
///      on (1) or off (0) to display the image you desire.
/// You can make a custom character at (https://omerk.github.io/lcdchargen/)
typedef struct custom_char{
   uint8_t line0;
   uint8_t line1;
   uint8_t line2;
   uint8_t line3;
   uint8_t line4;
   uint8_t line5;
   uint8_t line6;
   uint8_t line7;
}custom_char_t;
/********************** Structure Definitions ***************************
*********************************************************************/

/************************ Global Definitions ****************************
*********************************************************************/
uint8_t _offset;              // offset in CGRAM for new custom characters
/************************ Global Definitions ****************************
*********************************************************************/

/********************** Function Prototypes *****************************
*********************************************************************/
void lcdInit ();
```

```
void lcdClear();
void lcdTriggerEN();
void lcdWriteData(unsigned char data);
void lcdWriteCmd (unsigned char  cmd);
void lcdSetText(char * text, int x, int y);
void lcdSetChar(char c, int x, int y);
void lcdSetInt (int val,     int x, int y);
uint8_t lcdCreateCustomChar(custom_char_t* cc);
void Characters(void);
void Character_init(void);
/********************** Function Prototypes *****************************
*******************************************************************************/

/*************Global Variables Used Throughout Libraries*****************
*******************************************************************************/
extern char Diamond_index;
extern char Arrows_index;
extern char Dollar_index;
extern char Stick_Person_index;
/*************Global Variables Used Throughout Libraries*****************
*******************************************************************************/


#endif /* LIQUID_CRYSTAL_H_ */
```

SysTick_Library.c

```
/*****************************************************************************
*********
*****************************************************************************
*********
*                       SysTick_Library.c
*                    Trevor Ekin / Nabeeh Kandalaft
*                   EGR226      Date: March, 6, 2019
*                  (Updated by Joshua Johnston Date: March 26, 2019)
*
*  This is a library for the SysTick Timer Peripheral on the MSP432.
*
*  All functions are briefly described in their comment blocks.  The /// notation makes
*  it so the function description block is visible when you hovering over a function call
*  in any file (this feature is called Intellisense).
*
/// * UPDATE
* Initialization functions added for Standard 12 Key Keypad, Momentary Push Buttons and
LEDs.
* A function used to read the Keypad and return the number pressed
* See Slot_Machine.c Library for pin out and reference schematic for wiring
*
*****************************************************************************
**************/

#include "SysTick_Library.h"
#include "msp.h"
#include <stdio.h>

/*************Global Variables Used Throughout Libraries*****************
*************************************************************************/
volatile uint8_t ButtonPress = 0;
volatile uint8_t BetUp = 0;
volatile uint8_t BetDown = 0;
volatile uint8_t Start = 0;
volatile uint8_t CashOut = 0;
uint8_t num;
volatile int Bet = 0;
/*************Global Variables Used Throughout Libraries*****************
*************************************************************************/

/// ****| SysTickInit_NoInterrupts | ****************//*
/// * Brief: Initialize the SysTick peripheral for use
/// *         without interrupts (busy-waits)
/// * param:
```

```
/// *      N/A
/// * return:
/// *      N/A
/// ***************************************************/
void SysTickInit_NoInterrupts(void){
   SysTick->CTRL  &= ~BIT0;                    //clears enable to stop the counter
   SysTick->LOAD   = 0x00FFFFFF;                  //sets the period... note: (3006600/1000 - 1)
= 1ms
   SysTick->VAL    = 0;                        //clears the value
   SysTick->CTRL   = (STCSR_CLKSRC | STCSR_EN);       //enable SysTick, core clock, no
interrupts, this is the ENABLE and CLKSOURSE
}

/// ****| SysTickInit_WithInterrupts | **************//*
/// * Brief: Initialize the SysTick peripheral for use
/// *         with interrupts (interrupt delays)
/// * param:
/// *      N/A
/// * return:
/// *      N/A
/// ***************************************************/
void SysTickInit_WithInterrupts(void){
   SysTick->CTRL  &= ~BIT0;                          //clears enable to stop the counter
   SysTick->LOAD   = 0x00FFFFFF;                        //sets the period... note:
(3006600/1000 - 1) = 1ms
   SysTick->VAL    = 0;                              //clears the value
   SysTick->CTRL   = (STCSR_CLKSRC | STCSR_INT_EN | STCSR_EN);       // this is the
ENABLE, TICKINT, CLKSOURSE a WITH interrupts Systic->CTRL |= 0x07;
}

/// ****| SysTick_delay_ms | ************************//*
/// * Brief: Use the SysTick timer to delay a specified
/// *         number of milliseconds
/// * param:
/// *      (uint32_t) ms_delay:  number of milliseconds
/// *                           to delay
/// * return:
/// *      N/A
/// ***************************************************/
 void SysTick_delay_ms(uint32_t ms_delay){
   //Delays time_ms number of milliseconds
   //Assume 3MHz clock -> 3000 cycles per millisecond
   SysTick->LOAD  = 3000 * (uint32_t)ms_delay;
   SysTick->VAL   = 0;                          // starts counting from 0
   SysTick->CTRL |= (STCSR_CLKSRC | STCSR_EN);      // ENABLE, CLKSOURSE bits
.... Systic->CTRL |= 0x05;
```

```
    while(!(SysTick->CTRL & ((uint32_t)1)<<16));      // Continue while bit 16 is high or use
....while( (SysTick->CTRL & BIT16) == 0);
   SysTick->CTRL &= ~(STCSR_CLKSRC | STCSR_EN);      // Disable the Systic timer
.... Systic->CTRL =0 ;
}

/// ****| SysTick_delay_us | ************************//*
/// * Brief: Use the SysTick timer to delay a specified
/// *         number of microseconds
/// * param:
/// *      (uint32_t) us_delay:  number of microseconds
/// *                            to delay
/// * return:
/// *      N/A
/// ******************************************************/
void SysTick_delay_us(uint32_t us_delay){
   //Delays time_ms number of milliseconds
   //Assume 3MHz clock -> 3 cycles per microsecond
   SysTick->LOAD  = us_delay*3 - 1;                //counts up to delay
   SysTick->VAL   = 0;                            //starts counting from 0
   SysTick->CTRL |= (STCSR_CLKSRC | STCSR_EN);      // ENABLE, CLKSOURSE bits  ....
Systic->CTRL |= 0x05;
   while(!(SysTick->CTRL & ((uint32_t)1)<<16));     // Continue while bit 16 is high   .... while(
(SysTick->CTRL & BIT16) == 0);
   SysTick->CTRL &= ~(STCSR_CLKSRC | STCSR_EN);     // Disable the Systic timer        ....
Systic->CTRL =0 ;
}

/// **********| PORT5_IRQHandler | ************************
/// * Brief: When Port5 interrupt is triggered
/// *         by pressing the keypad, function and sets
/// *         variable value to one
/// *
/// * Global Variables:
/// *        ButtonPress
/// ****Note: ButtonPress used as a port interrupt
/// *         variable for Keypad recognition
/// * return:
/// *      N/A
/// ******************************************************/
void PORT5_IRQHandler(void){

       if(P5->IFG & (BIT4 | BIT5 | BIT6 | BIT7))  //If keypad is pressed interrupt is triggered

        ButtonPress = 1;                    //Set value of one to volatile variable ButtonPress
```

```
        P5->IFG &= ~(BIT4 | BIT5 | BIT6 | BIT7);   //Clear all flags
}

/// **********| PORT1_IRQHandler | ****************************
/// * Brief: When Port1 interrupt is triggered by pressing
/// *        any of Spin, BetUp, or BetDown sets specific
/// *        variable to the value of one based off which
/// *        button is pressed.
/// *
/// * Global Variables:
/// *        Start, BetUp, BetDown
/// ****Note: Start, BetUp, and BetDown used as port
/// *         interrupt variables
/// * return:
/// *      N/A
/// ************************************************************/
void PORT1_IRQHandler(void){

   if(P1->IFG & BIT5)               //If spin interrupt is triggered

      Start = 1;                  //Set value of one to volatile variable start

   if(P1->IFG & BIT6)               //If bet up interrupt is triggered

      BetUp = 1;                  //Set value of one to volatile variable BetUp

   if(P1->IFG & BIT7)               //If BetDown interrupt is triggered

      BetDown = 1;                //Set value of one to volatile variable BetDown

   P1->IFG &= ~(BIT5 | BIT6 | BIT7);   //Clear all interrupt flags

}

/// **********| PORT6_IRQHandler | ****************************
/// * Brief: When Cashout button is pressed interrupt is
/// *        is triggered and sets CashOut variable to one.
/// *
/// * Global Variables:
/// *        CashOut
/// ****Note: CashOut is used as port interrupt variable
/// *
/// * return:
/// *      N/A
/// ************************************************************/
void PORT6_IRQHandler(void){
```

```
    if(P6->IFG & BIT0)      //If interrupt flag triggers

        CashOut = 1;        //Set value of one to volatile variable

    P6->IFG &= ~BIT0;       //Clear flag
}

/// ******************| KeyPad_Init | ****************************
/// * Brief: Initializes KeyPad, rows have internal pull-up
/// *        resistors and interrupts. Columns initialized
/// *        as GPIO inputs
/// *
/// * Global Variables:
/// *        N/A
/// * return:
/// *      N/A
/// ***************************************************************/
void KeyPad_Init(void){

    P5->SEL0 = 0;
    P5->SEL1 = 0;                           //GPIO
    P5->DIR &= ~(BIT4 | BIT5 | BIT6 | BIT7);       //Port 5 pins set to input
    P5->REN |= (BIT4 | BIT5 | BIT6 | BIT7);        //Enable internal resistor
    P5->OUT |= (BIT4 | BIT5 | BIT6 | BIT7);        //Enable resistor as pull-up
    P5->IES |= (BIT4 | BIT5 | BIT6 | BIT7);        //High to low interrupt trigger for rows
    P5->IFG = 0;                            //Clear flags
    P5->IE |= (BIT4 | BIT5 | BIT6 | BIT7);         //Enable interrupts on rows


    P2->SEL0 = 0;
    P2->SEL1 = 1;                           //GPIO
    P2->DIR &= ~(BIT5 | BIT6 | BIT7);              //Set columns as inputs
}

/// ******************| Read_Keypad | ****************************
/// * Brief: Reads number pressed on keypad based of column
/// *         and row state.
/// *
/// * Global Variables:
/// *        num
/// * return:
/// *      num: number pressed on keypad
/// ***************************************************************/
uint8_t Read_Keypad(void){
```

```
    uint8_t col;
    uint8_t row;

    for(col = 0; col < 3; col++)
    {
        P2->DIR = 0;
        P5->DIR = 0;                        //Set port 4 to input
        P2->DIR |= BIT(5 + col);            //Set columns 0 - 2 to output by incrementing
        P2->OUT &= ~BIT(5 + col);           //Set columns 0 - 2 to LOW by incrementing

        SysTick_delay_ms(10);               //Delays while loop 10ms
        row = P5->IN & 0xF0;                //Read all rows

        while ( !(P5IN & BIT4) | !(P5IN & BIT5) | !( P5IN & BIT6) | !( P5IN & BIT7) );

        if(row != 0xF0) break;              //If low reading some key is pressed
    }

    P2->DIR &= ~(BIT5 | BIT6 | BIT7);       //Set columns to inputs
    if(col == 3)


        return 0;

    if(row == 0xE0) num = col + 1;          //Key press in row 0
    if(row == 0xD0) num = 3 + col + 1;      //Key press in row 2
    if(row == 0xB0) num = 6 + col + 1;      //Key press in row 2
    if(row == 0x70) num = 9 + col + 1;      //Key press in row 3



    return 1;
}

/// ******************| Button_init | ****************************
/// * Brief: Initializes all four buttons: BetUp, BetDown, Spin
/// *        and CashOut as input port interrupts
/// *
/// * Global Variables:
/// *        N/A
/// * return:
/// *        N/A
/// ************************************************************/
void Buttons_init(void){

    P1->SEL0 = 0;
```

```
    P1->SEL1 = 0;                       //Sets port 1 as GPIO
    P1->DIR &= ~(BIT5 | BIT6 | BIT7);       //Set as input
    P1->REN |= (BIT5 | BIT6 | BIT7);        //Enable internal resistor
    P1->OUT |= (BIT5 | BIT6 | BIT7);        //Enable resistor as pull-up
    P1->IES |= (BIT5 | BIT6 | BIT7);        //Set to trigger interrupt from high to low
    P1->IFG = 0;                        //Clears all interrupt flags
    P1->IE |= (BIT5 | BIT6 | BIT7);         //Enables interrupt

    P6->SEL0 = 0;
    P6->SEL1 = 0;                       //GPIO
    P6->DIR &= ~BIT0;                       //Input
    P6->REN |= BIT0;                      //Enable internal resistor
    P6->OUT |= BIT0;                      //Enable resistor as pull-up
    P6->IES |= BIT0;                     //High to low trigger
    P6->IFG = 0;                        //Clear flags
    P6->IE |= BIT0;                       //Enable interrupt
}

/// ******************| LED_init | ***************************
/// * Brief: Initializes all six LEDs initially in the off
/// *        or set to zero.
/// *
/// * Global Variables:
/// *         N/A
/// * return:
/// *         N/A
/// ************************************************************/
void LED_init(void){

    P3->SEL0 = 0;
    P3->SEL1 = 0;                     //Sets Port 3 to GPIO
    P3->DIR |= (BIT0 | BIT2 | BIT3);    //Sets Bit 0, 2, and 3 to output
    P3->DIR |= (BIT5 | BIT6 | BIT7);    //Sets Bit 5, 6, and 7 to output
    P3->OUT = 0;                      //Sets initial state of port to 0
}
```

SysTick_Library.h

```
/*****************************************************************************
*********
*****************************************************************************
*********
*                       SysTick_Library.h
*                    Trevor Ekin / Nabeeh Kandalaft
*                   EGR226      Date: March, 6, 2019
*              (Updated by Joshua Johnston Date: March 26, 2019)
*
*
*  This is a library for the SysTick Timer Peripheral on the MSP432.
*
*  All functions are briefly described in their comment blocks.  The /// notation makes
*  it so the function description block is visible when you hovering over a function call
*  in any file (this feature is called Intellisense).
*
/// * UPDATE
* Initialization functions added for Standard 12 Key Keypad, Momentary Push Buttons and
LEDs.
* A function used to read the Keypad and return the number pressed
* See Slot_Machine.c Library for pin out and reference schematic for wiring
*
*****************************************************************************
*************/

#ifndef SYSTICK_LIBRARY_H_
#define SYSTICK_LIBRARY_H_

#include "msp.h"
#include <stdint.h>

// SysTick Control and Status Register (STCSR)
#define STCSR_COUNT_FG  (0x0100)
#define STCSR_CLKSRC    (0x0004)
#define STCSR_INT_EN    (0x0002)
#define STCSR_EN        (0x0001)

/*************Global Variables Used Throughout Libraries*****************
*************************************************************************/
extern volatile uint8_t ButtonPress;
extern volatile uint8_t BetUp;
extern volatile uint8_t BetDown;
extern volatile uint8_t Start;
extern volatile int Bet;
```

```
extern volatile uint8_t CashOut;
extern uint8_t num;
/*************Global Variables Used Throughout Libraries*****************
*************************************************************************/

/********************** Macro Prototypes ********************************
*************************************************************************
  SysTick Control and Status Register (STCSR) as discussed in lectures

//#define STCSR_COUNT_FG  BIT16
//#define STCSR_CLKSRC    BIT2        // this is the CLKSOURSE bit
//#define STCSR_INT_EN    BIT1        // This is the TICKINT  bit
//#define STCSR_EN        BIT0        // This is the ENABLE bit

********************** Macro Prototypes ********************************
*************************************************************************


********************** Function Prototypes *****************************
*************************************************************************/
void SysTickInit_NoInterrupts  (void);
void SysTickInit_WithInterrupts(void);
void SysTick_delay_ms(volatile uint32_t);
void SysTick_delay_us(volatile uint32_t);
void KeyPad_Init(void);
uint8_t Read_Keypad(void);
void Buttons_init(void);
void LED_init(void);
/********************** Function Prototypes *****************************
*************************************************************************/

#endif /* SYSTICK_LIBRARY_H_ */
```

ADC_Library.c

```
/*******************************************************************************
 *******************************************************************************
 *                   ADC_Library.c
 *           Joshua Johnston and Jordan Hayes
 *                     EGR226
 *           Instructor: Dr. Nabeeh Kandalaft
 *             Created on: March 26, 2019
 *
 *      For use with the MSP432 LaunchPad Development Board
 *
 * Library is used for ADC conversion with 7 Segment Display and potentiometer
 *
 * The /// notation makes it so the function description block
 * is visible when you hovering over a function call in any
 * file (this feature is called Intellisense).
 *
 * Functions are each briefly described in comment blocks
 * Functions are commented out throughout program
 * Pin descriptions are listed below
 ********************** PIN DESCRIPTIONS
*****************************************
 *
 *        *********** Potentiometer ***************
 *                P5.0 -> Pin 2
 *                V0 LCD -> Pin 2
 *
 *        ********** 7 SEGMENT DISPLAY ************
 *                P7.0 -> Pin 1
 *                P7.1 -> Pin 2
 *                P7.2 -> Pin 4
 *                P7.3 -> Pin 5
 *                P7.4 -> Pin 6
 *                P7.5 -> Pin 7
 *                P7.6 -> Pin 9
 *                P7.7 -> Pin 10
 *

*******************************************************************************
****/

#include "msp.h"
#include "SysTick_Library.h"
#include "ADC_Library.h"
#include <stdio.h>
```

```
#include <stdint.h>

/// ******************| sevenSegment_init | **********************
/// * Brief: Initializes all four buttons: BetUp, BetDown, Spin
/// *        and CashOut as input port interrupts
/// *
/// * Global Variables:
/// *        N/A
/// * return:
/// *        N/A
/// ************************************************************/
void sevenSegment_init(void) {

   P7 -> SEL0 = 0;
   P7 -> SEL1 = 0;
   P7 -> DIR |= (BIT0 | BIT1 | BIT2 | BIT3 | BIT4 | BIT5 | BIT6 | BIT7);
   P7 -> OUT = 0;
}

/// ******************| ADC_pin_init | **********************
/// * Brief: Initializes P5.0 for ADC
/// *
/// * Global Variables:
/// *        N/A
/// * return:
/// *        N/A
/// ************************************************************/
void ADC_pin_init(void) {

   P5->SEL0 |= BIT0;   //Set P5.0 for ADC
   P5-> SEL0 |= BIT0;
}

/// ******************| ADC_init | **********************
/// * Brief: Initializes ADC
/// *
/// * Global Variables:
/// *        N/A
/// * return:
/// *        N/A
/// ************************************************************/
void ADC_init(void) {

   ADC14-> CTL0 &= ~ADC14_CTL0_ENC;    //disable, no conversions runnning
   ADC14-> CTL0 |= 0x04200210;        //creates the ADC using SMCLK, 1 turns it on, 2 sec
cycle
```

```
    ADC14-> CTL1 = 0x00000030;          //14 bit resolution
    ADC14-> CTL1 |= 0x0000000;
    ADC14-> MCTL[0] =0x05;
    ADC14-> CTL0 |= ADC14_CTL0_ENC;     //enable, start conversion
}

/// ******************| ADC_Reading | **********************
/// * Brief: Takes in ADC raw value converts it to voltage
/// * and displays numbers 0 - 9 on 7 - Segment display
/// * based on voltage conditions.
/// *
/// * Global Variables:
/// *        N/A
/// * return:
/// *        N/A
/// ************************************************************/
void ADC_Reading(void) {

    static volatile uint16_t result;
    float V;

    ADC14->CTL0 |= ADC14_CTL0_SC;

    while(!ADC14->IFGR0 & BIT0);       //wait until conversion is finished

    result = ADC14 -> MEM[0];          //result equals converted value
    V = ((result * 5.0) / 16384);      //Voltage conversion from raw ADC readings

    if(V <= .5) {
      P7 -> OUT = 0;
      P7 -> OUT |= NINE;               //Displays defined 9 on 7 - Segment display
      SysTick_delay_ms(100);
    }

    if(V > .5 && V <= 1.0) {
      P7 -> OUT = 0;
      P7 -> OUT |= EIGHT;              //Displays defined 8 on 7 - Segment display
      SysTick_delay_ms(100);
    }

    if(V >1.0 && V <= 1.5) {
      P7 -> OUT = 0;
      P7 -> OUT |= SEVEN;              //Displays defined 7 on 7 - Segment display
      SysTick_delay_ms(100);
    }
```

```
    if(V > 1.5 && V <= 2.0) {
      P7 -> OUT = 0;
      P7 -> OUT |= SIX;              //Displays defined 6 on 7 - Segment display
      SysTick_delay_ms(100);
    }

    if(V > 2.0 && V <= 2.5) {
      P7 -> OUT = 0;
      P7 -> OUT |= FIVE;             //Displays defined 5 on 7 - Segment display
      SysTick_delay_ms(100);
    }

    if(V > 2.5 && V <= 3.0) {
      P7 -> OUT = 0;
      P7 -> OUT |= FOUR;             //Displays defined 4 on 7 - Segment display
      SysTick_delay_ms(100);
    }

    if(V > 3.0 && V <= 3.5) {
      P7 -> OUT = 0;
      P7 -> OUT |= THREE;            //Displays defined 3 on 7 - Segment display
      SysTick_delay_ms(100);
    }

    if(V > 3.5 && V <= 4.0) {
      P7 -> OUT = 0;
      P7 -> OUT |= TWO;              //Displays defined 2 on 7 - Segment display
      SysTick_delay_ms(100);
    }

    if(V > 4.0 && V <= 4.5) {
      P7 -> OUT = 0;
      P7 -> OUT |= ONE;              //Displays defined 1 on 7 - Segment display
      SysTick_delay_ms(100);
    }

    if(V > 4.5) {
      P7 -> OUT = 0;
      P7 -> OUT |= ZERO;             //Displays defined 0 on 7 - Segment display
      SysTick_delay_ms(100);
    }
}
```

AC_Library.h

```
/*****************************************************************************
 *****************************************************************************
 *                  ADC_Library.c
 *           Joshua Johnston and Jordan Hayes
 *                    EGR226
 *          Instructor: Dr. Nabeeh Kandalaft
 *            Created on: March 26, 2019
 *
 *      For use with the MSP432 LaunchPad Development Board
 *
 * Library is used for ADC conversion with 7 Segment Display and potentiometer
 *
 * The /// notation makes it so the function description block
 * is visible when you hovering over a function call in any
 * file (this feature is called Intellisense).
 *
 * Functions are each briefly described in comment blocks
 * Functions are commented out throughout program
 * Pin descriptions are listed below

*****************************************************************************
*/

#ifndef ADC_LIBRARY_H_
#define ADC_LIBRARY_H_

 /********************** PIN DESCRIPTIONS
****************************************
 *
 *        *********** Potentiometer ***************
 *                P2.4 -> Pin 2
 *               V0 LCD -> Pin 2
 *
 *        ********** 7 SEGMENT DISPLAY ************
 *                P7.0 -> Pin 1
 *                P7.1 -> Pin 2
 *                P7.2 -> Pin 4
 *                P7.3 -> Pin 5
 *                P7.4 -> Pin 6
 *                P7.5 -> Pin 7
 *                P7.6 -> Pin 9
 *                P7.7 -> Pin 10
 *
```

```
**********************************************************************************
****/

#include "msp.h"
#include <stdint.h>

//7 Segment display
#define ZERO  0x77            //0x01110111
#define ONE   0x14           //0x00010100
#define TWO   0xB3            //0x10110001
#define THREE 0xB6            //0x10110110
#define FOUR  0xD4            //0x11010100
#define FIVE  0xE6           //0x11100110
#define SIX   0xC7           //0x11000111
#define SEVEN 0x34            //0x00110100
#define EIGHT 0xF7            //0x11110111
#define NINE  0xF4           //0x11110100


/********************** Function Prototypes ******************************
*********************************************************************************/
void sevenSegment_init(void);
void ADC_Reading(void);
void ADC_pin_init(void);
void ADC_init(void);
/********************** Function Prototypes ******************************
*********************************************************************************/

#endif /* ADC_LIBRARY_H_ */
```

Tunes_Library.c

```
/*****************************************************************************
 *****************************************************************************
 *                    Tunes_Library.c
 *            Joshua Johnston and Jordan Hayes
 *                      EGR226
 *            Instructor: Dr. Nabeeh Kandalaft
 *              Created on: April 4, 2019
 *
 *      For use with the MSP432 LaunchPad Development Board
 *
 * Library is used for songs created with PWM from Timer A.
 * LEDs Initialized in SysTick_Library.c and used throughout for effect.
 *
 * The /// notation makes it so the function description block
 * is visible when you hovering over a function call in any
 * file (this feature is called Intellisense).
 *
 * Functions are each briefly described in comment blocks
 * Functions are commented out throughout program
 * Pin descriptions are listed below
 ********************** PIN DESCRIPTIONS
*****************************************
 *
 *         *********** Piezo Speaker ***************
 *                P2.4 -> Speaker input pin
 *

*****************************************************************************
****/

//Needs header block with descriptions of added functions, add function comment headers,
comment and format throughout

#include "msp.h"
#include "Tunes_Library.h"
#include "SysTick_Library.h"
#include <math.h>
#include <stdint.h>
#include <stdio.h>

uint8_t song = 3;      //Global variable for song selection

/// *****************| Speaker | ****************************
/// * Brief: Initializes timer A PWM for speaker, function
```

```
/// * inputs note and 2 for on and 0 for off.
/// *
/// * Global Variables:
/// *       N/A
/// * return:
/// *       N/A
/// ***************************************************************/
void Speaker(int Note, uint8_t on){

   int freq = 3000000 / Note;

   P2->SEL0 |= BIT4;                        //Set TimerA
   P2->SEL1 = 0;
   P2->DIR |= BIT4;                         //Set P2.4 for OutPut

   TIMER_A0->CCR[0] = freq;                    //Period based off note input
   TIMER_A0->CCTL[1] = TIMER_A_CCTLN_OUTMOD_7;    //CCR1 reset
   TIMER_A0->CCR[1] = freq / on;                //PWM duty cycle
   TIMER_A0->CTL = 0x0214;                     //SMCLK, Up Mode, Clear TAR to start
 }

/// *****************| Pirates_C | ********************************
/// * Brief: Plays pirates of Caribbean theme song with variations
/// * of notes and delays. As well displays a matching LED sequence
/// *
/// * Global Variables:
/// *       N/A
/// * return:
/// *       N/A
/// ***************************************************************/
void Pirates_C(void){

      //FIRST PART
        Speaker(E_4, 2);
        P3->OUT |= BIT0;
        SysTick_delay_ms(125);
        Speaker(G_4, 2);
        P3->OUT = 0;
        P3->OUT |= BIT2;
        SysTick_delay_ms(125);
        Speaker(A_4, 2);
        P3->OUT = 0;
        P3->OUT |= (BIT5 | BIT6 | BIT7);
        SysTick_delay_ms(250);
        Speaker(A_4,2);
        P3->OUT = 0;
```

```
P3->OUT |= (BIT5 | BIT6 | BIT7);
SysTick_delay_ms(125);
Speaker(0, 0);
P3->OUT = 0;
SysTick_delay_ms(125);

Speaker(A_4, 2);
P3->OUT = 0;
P3->OUT |= (BIT5 | BIT6 | BIT7);
SysTick_delay_ms(125);
Speaker(B_4, 2);
P3->OUT = 0;
P3->OUT |= BIT3;
SysTick_delay_ms(125);
Speaker(C_5, 2);
P3->OUT = 0;
P3->OUT |= (BIT0 | BIT2 | BIT3);
SysTick_delay_ms(250);
Speaker(C_5,2);
P3->OUT = 0;
P3->OUT |= (BIT0 | BIT2 | BIT3);
SysTick_delay_ms(125);
Speaker(0, 0);
P3->OUT = 0;
SysTick_delay_ms(125);

Speaker(C_5, 2);
P3->OUT = 0;
P3->OUT |= (BIT0 | BIT2 | BIT3);
SysTick_delay_ms(125);
Speaker(D_5, 2);
P3->OUT = 0;
P3->OUT |= (BIT0 | BIT2 | BIT3 | BIT5 | BIT6 | BIT7);
SysTick_delay_ms(125);
Speaker(B_4, 2);
P3->OUT = 0;
P3->OUT |= BIT3;
SysTick_delay_ms(250);
Speaker(B_4,2);
P3->OUT = 0;
P3->OUT |= BIT3;
SysTick_delay_ms(125);
Speaker(0, 0);
P3->OUT = 0;
SysTick_delay_ms(125);
```

```
  Speaker(A_4, 2);
  P3->OUT = 0;
  P3->OUT |= (BIT5 | BIT6 | BIT7);
  SysTick_delay_ms(125);
  Speaker(G_4, 2);
  P3->OUT = 0;
  P3->OUT |= BIT2;
  SysTick_delay_ms(125);
  Speaker(A_4, 2);
  P3->OUT = 0;
  P3->OUT |= (BIT5 | BIT6 | BIT7);
  SysTick_delay_ms(375);
  Speaker(0,0);
  P3->OUT = 0;
  SysTick_delay_ms(125);
//END OF PART 1

//PART 2
  Speaker(E_4, 2);
  P3->OUT = 0;
  P3->OUT |= BIT0;
  SysTick_delay_ms(125);
  Speaker(G_4, 2);
  P3->OUT = 0;
  P3->OUT |= BIT2;
  SysTick_delay_ms(125);
  Speaker(A_4, 2);
  P3->OUT = 0;
  P3->OUT |= (BIT5 | BIT6 | BIT7);
  SysTick_delay_ms(250);
  Speaker(A_4,2);
  P3->OUT = 0;
  P3->OUT |= (BIT5 | BIT6 | BIT7);
  SysTick_delay_ms(125);
  Speaker(0, 0);
  P3->OUT = 0;
  SysTick_delay_ms(125);

  Speaker(A_4, 2);
  P3->OUT = 0;
  P3->OUT |= (BIT5 | BIT6 | BIT7);
  SysTick_delay_ms(125);
  Speaker(B_4, 2);
  P3->OUT = 0;
  P3->OUT |= BIT3;
  SysTick_delay_ms(125);
```

```
Speaker(C_5, 2);
P3->OUT = 0;
P3->OUT |= (BIT0 | BIT2 | BIT3);
SysTick_delay_ms(250);
Speaker(C_5,2);
P3->OUT = 0;
P3->OUT |= (BIT0 | BIT2 | BIT3);
SysTick_delay_ms(125);
Speaker(0, 0);
P3->OUT = 0;
SysTick_delay_ms(125);

Speaker(C_5, 2);
P3->OUT = 0;
P3->OUT |= (BIT0 | BIT2 | BIT3);
SysTick_delay_ms(125);
Speaker(D_5, 2);
P3->OUT = 0;
P3->OUT |= (BIT0 | BIT2 | BIT3 | BIT5 | BIT6 | BIT7);
SysTick_delay_ms(125);
Speaker(B_4, 2);
P3->OUT = 0;
P3->OUT |= BIT3;
SysTick_delay_ms(250);
Speaker(B_4,2);
P3->OUT = 0;
P3->OUT |= BIT3;
SysTick_delay_ms(125);
Speaker(0, 0);
P3->OUT = 0;
SysTick_delay_ms(125);

Speaker(A_4, 2);
P3->OUT = 0;
P3->OUT |= (BIT5 | BIT6 | BIT7);
SysTick_delay_ms(125);
Speaker(G_4, 2);
P3->OUT = 0;
P3->OUT |= BIT2;
SysTick_delay_ms(125);
Speaker(A_4, 2);
P3->OUT = 0;
P3->OUT |= (BIT5 | BIT6 | BIT7);
SysTick_delay_ms(375);
Speaker(0,0);
P3->OUT = 0;
```

```
  SysTick_delay_ms(125);
//END OF PART 2

//PART 3
  Speaker(E_4, 2);
  P3->OUT = 0;
  P3->OUT |= BIT0;
  SysTick_delay_ms(125);
  Speaker(G_4, 2);
  P3->OUT = 0;
  P3->OUT |= BIT2;
  SysTick_delay_ms(125);
  Speaker(A_4, 2);
  P3->OUT = 0;
  P3->OUT |= (BIT5 | BIT6 | BIT7);
  SysTick_delay_ms(250);
  Speaker(A_4,2);
  P3->OUT = 0;
  P3->OUT |= (BIT5 | BIT6 | BIT7);
  SysTick_delay_ms(125);
  Speaker(0, 0);
  P3->OUT = 0;
  SysTick_delay_ms(125);

  Speaker(A_4, 2);
  P3->OUT = 0;
  P3->OUT |= (BIT5 | BIT6 | BIT7);
  SysTick_delay_ms(125);
  Speaker(C_5, 2);
  P3->OUT = 0;
  P3->OUT |= (BIT0 | BIT2 | BIT3);
  SysTick_delay_ms(125);
  Speaker(D_5, 2);
  P3->OUT = 0;
  P3->OUT |= (BIT0 | BIT2 | BIT3 | BIT5 | BIT6 | BIT7);
  SysTick_delay_ms(250);
  Speaker(D_5,2);
  P3->OUT = 0;
  P3->OUT |= (BIT0 | BIT2 | BIT3 | BIT5 | BIT6 | BIT7);
  SysTick_delay_ms(125);
  Speaker(0, 0);
  P3->OUT = 0;
  SysTick_delay_ms(125);

  Speaker(D_5, 2);
  P3->OUT = 0;
```

```
  P3->OUT |= (BIT0 | BIT2 | BIT3 | BIT5 | BIT6 | BIT7);
  SysTick_delay_ms(125);
  Speaker(E_5, 2);
  P3->OUT = 0;
  P3->OUT |= BIT0;
  SysTick_delay_ms(125);
  Speaker(F_5, 2);
  P3->OUT = 0;
  P3->OUT |= (BIT0 | BIT3);
  SysTick_delay_ms(250);
  Speaker(F_5,2);
  P3->OUT = 0;
  P3->OUT |= (BIT0 | BIT3);
  SysTick_delay_ms(125);
  Speaker(0, 0);
  P3->OUT = 0;
  SysTick_delay_ms(125);

  Speaker(E_5, 2);
  P3->OUT = 0;
  P3->OUT |= BIT0;
  SysTick_delay_ms(125);
  Speaker(D_5, 2);
  P3->OUT = 0;
  P3->OUT |= (BIT0 | BIT2 | BIT3 | BIT5 | BIT6 | BIT7);
  SysTick_delay_ms(125);
  Speaker(E_5, 2);
  P3->OUT = 0;
  P3->OUT |= BIT0;
  SysTick_delay_ms(125);
  Speaker(A_4,2);
  P3->OUT = 0;
  P3->OUT |= (BIT5 | BIT6 | BIT7);
  SysTick_delay_ms(250);
  Speaker(0, 0);
  P3->OUT = 0;
  SysTick_delay_ms(125);
 //END OF PART 3

 //PART 4
  Speaker(A_4, 2);
  P3->OUT = 0;
  P3->OUT |= (BIT5 | BIT6 | BIT7);
  SysTick_delay_ms(125);
  Speaker(B_4, 2);
  P3->OUT = 0;
```

```
P3->OUT |= BIT3;
SysTick_delay_ms(125);
Speaker(C_5, 2);
P3->OUT = 0;
P3->OUT |= (BIT0 | BIT2 | BIT3);
SysTick_delay_ms(250);
Speaker(C_5,2);
P3->OUT = 0;
P3->OUT |= (BIT0 | BIT2 | BIT3);
SysTick_delay_ms(125);
Speaker(0, 0);
P3->OUT = 0;
SysTick_delay_ms(125);

Speaker(D_5, 2);
P3->OUT = 0;
P3->OUT |= (BIT0 | BIT2 | BIT3 | BIT5 | BIT6 | BIT7);
SysTick_delay_ms(250);
Speaker(E_5, 2);
P3->OUT = 0;
P3->OUT |= BIT0;
SysTick_delay_ms(125);
Speaker(A_4, 2);
P3->OUT = 0;
P3->OUT |= (BIT5 | BIT6 | BIT7);
SysTick_delay_ms(250);
Speaker(0,0);
P3->OUT = 0;
SysTick_delay_ms(125);

Speaker(A_4, 2);
P3->OUT = 0;
P3->OUT |= (BIT5 | BIT6 | BIT7);
SysTick_delay_ms(125);
Speaker(C_5, 2);
P3->OUT = 0;
P3->OUT |= (BIT0 | BIT2 | BIT3);
SysTick_delay_ms(125);
Speaker(B_4, 2);
P3->OUT = 0;
P3->OUT |= BIT3;
SysTick_delay_ms(250);
Speaker(B_4,2);
P3->OUT = 0;
P3->OUT |= BIT3;
SysTick_delay_ms(125);
```

```
        Speaker(0, 0);
        P3->OUT = 0;
        SysTick_delay_ms(125);

        Speaker(C_5, 2);
        P3->OUT = 0;
        P3->OUT |= (BIT0 | BIT2 | BIT3);
        SysTick_delay_ms(125);
        Speaker(A_4, 2);
        P3->OUT = 0;
        P3->OUT |= (BIT5 | BIT6 | BIT7);
        SysTick_delay_ms(125);
        Speaker(B_4, 2);
        P3->OUT = 0;
        P3->OUT |= (BIT0 | BIT2 | BIT3 | BIT5 | BIT6 | BIT7);
        SysTick_delay_ms(375);
        Speaker(0,0);
        P3->OUT = 0;
        SysTick_delay_ms(1000);
        //END OF PART 4
}

/// ******************| Despacito | *********************************
/// * Brief: Plays song Despacito with variations of notes and delays,
/// * displays matching LED sequence.
/// *
/// * Global Variables:
/// *       N/A
/// * return:
/// *       N/A
/// ***************************************************************************/
void Despacito(void){

   Speaker(D_5, 2);
   P3->OUT |= BIT0;                 //Blue
   SysTick_delay_ms(575);
   Speaker(0,0);
   P3->OUT = 0;                     //Off
   SysTick_delay_ms(200);
   Speaker(C_S5, 2);
   P3->OUT |= BIT2;                 //Orange
   SysTick_delay_ms(575);
   Speaker(0,0);
   P3->OUT = 0;                     //Off
   SysTick_delay_ms(200);
   Speaker(B_4, 2);
```

```
P3->OUT |= BIT3;                //Yellow
SysTick_delay_ms(290);
Speaker(0,0);
P3->OUT = 0;                    //Off
SysTick_delay_ms(100);
Speaker(F_S4, 2);
P3->OUT |= (BIT5 | BIT6 | BIT7);    //Reds
SysTick_delay_ms(290);
Speaker(0, 0);
P3->OUT = 0;                    //Off
SysTick_delay_ms(100);

Speaker(F_S4, 2);
P3->OUT |= (BIT5 | BIT6 | BIT7);    //Reds
SysTick_delay_ms(145);
Speaker(0, 0);
P3->OUT = 0;                    //Off
SysTick_delay_ms(50);
Speaker(F_S4, 2);
P3->OUT |= (BIT5 | BIT6 | BIT7);    //Reds
SysTick_delay_ms(145);
Speaker(0, 0);
P3->OUT = 0;                     //Off
SysTick_delay_ms(50);
Speaker(F_S4, 2);
P3->OUT |= (BIT5 | BIT6 | BIT7);    //Reds
SysTick_delay_ms(145);
Speaker(0, 0);
P3->OUT = 0;                    //Off
SysTick_delay_ms(50);
Speaker(F_S4, 2);
P3->OUT |= (BIT5 | BIT6 | BIT7);    //Reds
SysTick_delay_ms(145);
Speaker(0, 0);
P3->OUT = 0;                    //Off
SysTick_delay_ms(50);
Speaker(F_S4, 2);
P3->OUT |= (BIT5 | BIT6 | BIT7);    //Reds
SysTick_delay_ms(145);
Speaker(0,0);
P3->OUT = 0;                    //Off
SysTick_delay_ms(50);

Speaker(B_4, 2);
P3->OUT |= BIT3;                //Yellow
SysTick_delay_ms(145);
```

```
Speaker(0, 0);
P3->OUT = 0;                    //Off
SysTick_delay_ms(50);
Speaker(B_4, 2);
P3->OUT |= BIT3;                  //Yellow
SysTick_delay_ms(145);
Speaker(0, 0);
P3->OUT = 0;                    //Off
SysTick_delay_ms(50);
Speaker(B_4, 2);
P3->OUT |= BIT3;                  //Yellow
SysTick_delay_ms(145);
Speaker(0, 0);
P3->OUT = 0;                    //Off
SysTick_delay_ms(50);
Speaker(B_4, 2);
P3->OUT |= BIT3;                  //Yellow
SysTick_delay_ms(290);
Speaker(0, 0);
P3->OUT = 0;                    //Off
SysTick_delay_ms(100);

Speaker(A_4, 2);
P3->OUT |= (BIT0 | BIT2 | BIT3 | BIT5 | BIT6 | BIT7); //All
SysTick_delay_ms(145);
Speaker(0,0);
P3->OUT = 0;                    //Off
SysTick_delay_ms(50);
Speaker(B_4, 2);
P3->OUT |= BIT3;                  //Yellow
SysTick_delay_ms(290);
Speaker(0,0);
P3->OUT = 0;                    //Off
SysTick_delay_ms(100);
Speaker(G_4, 2);
P3->OUT |= (BIT0 | BIT7);         //Blue and Red
SysTick_delay_ms(290);
Speaker(0, 0);
P3->OUT = 0;                    //Off
SysTick_delay_ms(100);

Speaker(G_4, 2);
P3->OUT |= (BIT0 | BIT7);         //Blue and Red
SysTick_delay_ms(145);
Speaker(0, 0);
P3->OUT = 0;                    //Off
```

```
SysTick_delay_ms(50);
Speaker(G_4, 2);
P3->OUT |= (BIT0 | BIT7);          //Blue and Red
SysTick_delay_ms(145);
Speaker(0, 0);
P3->OUT = 0;                       //Off
SysTick_delay_ms(50);
Speaker(G_4, 2);
P3->OUT |= (BIT0 | BIT7);          //Blue and Red
SysTick_delay_ms(145);
Speaker(0, 0);
P3->OUT = 0;                       //Off
SysTick_delay_ms(50);
Speaker(G_4, 2);
P3->OUT |= (BIT0 | BIT7);          //Blue and Red
SysTick_delay_ms(145);
Speaker(0, 0);
P3->OUT = 0;                       //Off
SysTick_delay_ms(50);
Speaker(G_4,2);
P3->OUT |= (BIT0 | BIT7);          //Blue and Red
SysTick_delay_ms(145);
Speaker(0,0);
P3->OUT = 0;                       //Off
SysTick_delay_ms(50);

Speaker(B_4, 2);
P3->OUT |= BIT3;                     //Yellow
SysTick_delay_ms(145);
Speaker(0, 0);
P3->OUT = 0;                       //Off
SysTick_delay_ms(50);
Speaker(B_4, 2);
P3->OUT |= BIT3;                     //Yellow
SysTick_delay_ms(145);
Speaker(0, 0);
P3->OUT = 0;                       //Off
SysTick_delay_ms(50);
Speaker(B_4, 2);
P3->OUT |= BIT3;                     //Yellow
SysTick_delay_ms(145);
Speaker(0, 0);
P3->OUT = 0;                       //Off
SysTick_delay_ms(50);
Speaker(B_4, 2);
P3->OUT |= BIT3;                     //Yellow
```

```
SysTick_delay_ms(290);
Speaker(0,0);
P3->OUT = 0;                    //Off
SysTick_delay_ms(100);

Speaker(C_S5, 2);
P3->OUT |= BIT2;                   //Orange
SysTick_delay_ms(145);
Speaker(0,0);
P3->OUT = 0;                    //Off
SysTick_delay_ms(50);
Speaker(D_5, 2);
P3->OUT |= BIT0;                   //Blue
SysTick_delay_ms(290);
Speaker(0,0);
P3->OUT = 0;                    //Off
SysTick_delay_ms(100);

Speaker(A_4, 2);
P3->OUT |= (BIT0 | BIT2 | BIT3 | BIT5 | BIT6 | BIT7); //All
SysTick_delay_ms(290);
Speaker(0,0);
P3->OUT = 0;                    //Off
SysTick_delay_ms(100);

Speaker(A_4, 2);
P3->OUT |= (BIT0 | BIT2 | BIT3 | BIT5 | BIT6 | BIT7); //All
SysTick_delay_ms(145);
Speaker(0, 0);
P3->OUT = 0;                    //Off
SysTick_delay_ms(50);
Speaker(A_4, 2);
P3->OUT |= (BIT0 | BIT2 | BIT3 | BIT5 | BIT6 | BIT7); //All
SysTick_delay_ms(145);
Speaker(0, 0);
P3->OUT = 0;                    //Off
SysTick_delay_ms(50);
Speaker(A_4, 2);
P3->OUT |= (BIT0 | BIT2 | BIT3 | BIT5 | BIT6 | BIT7); //All
SysTick_delay_ms(145);
Speaker(0, 0);
P3->OUT = 0;                    //Off
SysTick_delay_ms(50);
Speaker(A_4, 2);
P3->OUT |= (BIT0 | BIT2 | BIT3 | BIT5 | BIT6 | BIT7); //All
SysTick_delay_ms(145);
```

```
Speaker(0, 0);
P3->OUT = 0;                    //Off
SysTick_delay_ms(50);

Speaker(D_5, 2);
P3->OUT |= BIT0;                  //Blue
SysTick_delay_ms(145);
Speaker(0,0);
P3->OUT = 0;                    //Off
SysTick_delay_ms(50);
Speaker(C_S5, 2);
P3->OUT |= BIT2;                  //Orange
SysTick_delay_ms(145);
Speaker(0,0);
P3->OUT = 0;                    //Off
SysTick_delay_ms(50);
Speaker(D_5, 2);
P3->OUT |= BIT0;                  //Blue
SysTick_delay_ms(145);
Speaker(0,0);
P3->OUT = 0;                    //Off
SysTick_delay_ms(50);
Speaker(C_S5, 2);
P3->OUT |= BIT2;                  //Orange
SysTick_delay_ms(145);
Speaker(0,0);
P3->OUT = 0;                    //Off
SysTick_delay_ms(50);

Speaker(D_5, 2);
P3->OUT |= BIT0;                  //Blue
SysTick_delay_ms(290);
Speaker(0,0);
P3->OUT = 0;                    //Off
SysTick_delay_ms(100);

Speaker(E_5,2);
P3->OUT |= (BIT3 | BIT5);         //Red and Yellow
SysTick_delay_ms(145);
Speaker(0,0);
P3->OUT = 0;                    //Off
SysTick_delay_ms(50);
Speaker(E_5, 2);
P3->OUT |= (BIT3 | BIT5);         //Red and Yellow
SysTick_delay_ms(290);
Speaker(0,0);
```

```
    P3->OUT = 0;                    //Off
    SysTick_delay_ms(100);
    Speaker(C_S5,2);
    P3->OUT |= BIT2;                //Orange
    SysTick_delay_ms(575);
    Speaker(0,0);
    P3->OUT = 0;                    //Off
    SysTick_delay_ms(400);

    Speaker(D_5, 2);
    P3->OUT |= BIT0;                //Blue
    SysTick_delay_ms(575);
    Speaker(0,0);
    P3->OUT = 0;                    //Off
    SysTick_delay_ms(200);
    Speaker(C_S5, 2);
    P3->OUT |= BIT2;                //Orange
    SysTick_delay_ms(575);
    Speaker(0,0);
    P3->OUT = 0;                    //Off
    SysTick_delay_ms(200);
    Speaker(B_4, 2);
    P3->OUT |= BIT3;                //Yellow
    SysTick_delay_ms(290);
    Speaker(0,0);
    P3->OUT = 0;                    //Off
    SysTick_delay_ms(100);
    Speaker(F_S4, 2);
    P3->OUT |= (BIT5 | BIT6 | BIT7);    //Reds
    SysTick_delay_ms(290);
    Speaker(0, 0);
    P3->OUT = 0;                    //Off
    SysTick_delay_ms(100);
}

/// ******************| Bells | *********************************
/// * Brief: Plays song loud bell sounding noise for wins by
/// * using for loop to time it and small delays for faster rate.
/// * Displays matching LED flashing.
/// *
/// * Global Variables:
/// *        N/A
/// * return:
/// *        N/A
/// ***************************************************************/
```

```
void Bells(void){

   int a;

   for(a = 0; a < 50; a ++){              //For loop allows for repeating

      P3->OUT = 0;                     //LEDs off
      P3->OUT |= (BIT5 | BIT6 | BIT7);    //Red LEDS on
      Speaker(D_5, 2);                  //Note frequency D_5 plays
      SysTick_delay_ms(50);              //Delays for 50ms

      P3 -> OUT = 0;                   //LEDs off
      P3->OUT |= (BIT0 | BIT2 | BIT3);    //Multi color LEDs on
      Speaker(C_5, 2);                  //Plays note frequency C_5
      SysTick_delay_ms(50);              //Delays 50ms
   }

      P3->OUT = 0;                     //After completion of for loop LEDs off
      Speaker(0,0);                   //Speaker off
}
```

Tunes_Library.h

```
/*****************************************************************************
 *****************************************************************************
 *                    Tunes_Library.c
 *            Joshua Johnston and Jordan Hayes
 *                      EGR226
 *            Instructor: Dr. Nabeeh Kandalaft
 *              Created on: April 4, 2019
 *
 *      For use with the MSP432 LaunchPad Development Board
 *
 * Library is used for songs created with PWM from Timer A.
 * LEDs Initialized in SysTick_Library.c and used throughout for effect.
 *
 * The /// notation makes it so the function description block
 * is visible when you hovering over a function call in any
 * file (this feature is called Intellisense).
 *
 * Functions are each briefly described in comment blocks
 * Functions are commented out throughout program
 * Pin descriptions are listed below
 *****************************************************************************/

#ifndef TUNES_LIBRARY_H_
#define TUNES_LIBRARY_H_

/************************ PIN DESCRIPTIONS
***************************************
 *
 *         ************ Piezo Speaker ***************
 *                 P2.4 -> Speaker input pin
 *

*****************************************************************************
****/

#include "msp.h"
#include <stdint.h>

//Note frequencies
#define A_2     110
#define A_S2    117
#define B_2     123
#define C_3     131
#define C_S3    139
```

```
#define D_3     147
#define D_S3    156
#define E_3     165
#define F_3     175
#define F_S3    185
#define G_3     196
#define G_S3    208
#define A_3     220
#define A_S3    233
#define B_3     247
#define C_4     262
#define C_S4    277
#define D_4     294
#define D_S4    311
#define E_4     330
#define F_4     349
#define F_S4    370
#define G_4     392
#define G_S4    415
#define A_4     440
#define A_S4    466
#define B_4     494
#define C_5     523
#define C_S5    554
#define D_5     587
#define D_S5    622
#define E_5     659
#define F_5     698
#define F_S5    740
#define G_5     784
#define G_S5    831
#define A_5     880
#define A_S5    932
#define B_5     988

extern uint8_t song;

/********************** Function Prototypes ****************************
*******************************************************************************/
void Speaker(int Note, uint8_t on);
void Pirates_C(void);
void Despacito(void);
void Bells(void);
/********************** Function Prototypes ****************************
*******************************************************************************/
#endif /* TUNES_LIBRARY_H_ */
```