Final Report - Media Center

COE718 - Embedded Systems Design

Fall 2024

Eric Muzzo

Student # 501019745

Department of Computer and Electrical Engineering

Toronto Metropolitan University

1. Abstract

The media player being developed in this project consists of several modules that interface with different physical components on the MCB1700 development board. The concepts and methodologies employed learned in the course were used to achieve a media center with a photo gallery, MP3 player, and a game center. Each of these core modules utilizes the LCD screen, and joystick on the development board. This report serves a cumulative summary of the media center project built in the lab, which includes the full design methodology and results.

2. Introduction

Embedded systems are typically real-time information systems that involve a programmable computer to control some larger system or product. This course has examined the organization of such systems, programmable on-chip technologies, and real-time systems. Today, embedded systems are used in just about every electronic device in everyday settings. Throughout the duration of the course, the architecture of the ARM Cortex-M3 chip was studied and experimented on in the labs with the MCB1700 development board. Using the knowledge gained from class, an interactive media center application was developed.

3. Methodology

The media center as an entire application consists of several sub-applications; the main menu, the photo gallery, the MP3 player, and the game center. Since each sub-application contains its own logic, functionality and controls, a modular approach was taken to divide the project into smaller components. The source code consists of individual, well defined, C files for each sub-application. Each of which contain an entire control structure with their own main functions to run their specific program.

The modular design approach taken made the code much more readable and simpler to debug. There were very similar logic structures for each component, like menu functionality for example, which allowed me to apply my work from one module to another. The operational flow was an overall success and more importantly, very easy to trace when reading the code because of the modular nature of my source code.

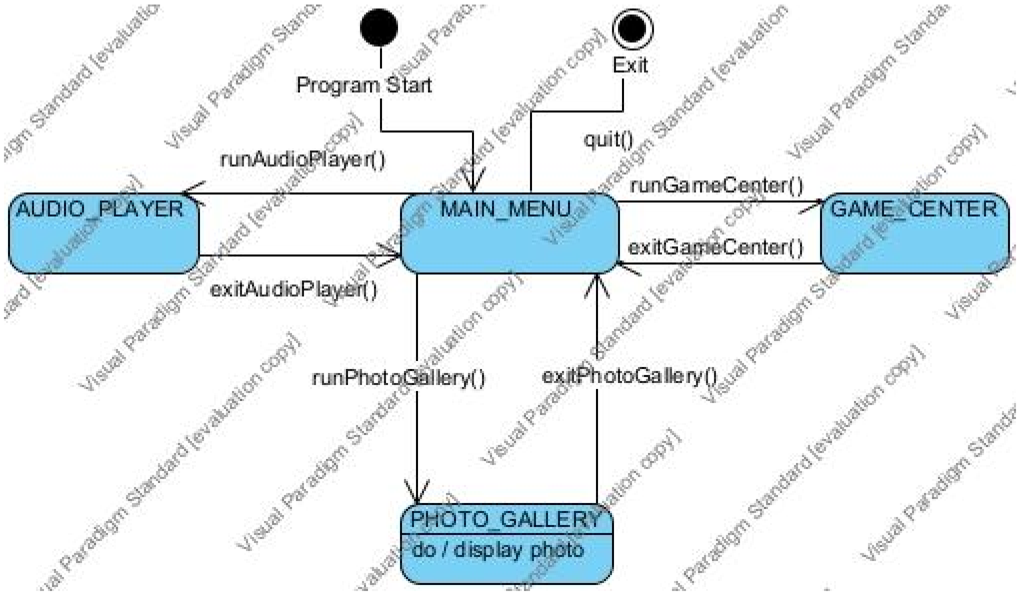

Figure 1: State Diagram

A state machine will be used to globally determine the state of the application. With this approach, the main C file will be able to easily remain in an infinite loop while constantly monitoring the application state. This allows for easy task switching to various sub-applications depending on the system state.

3. System Design

My preliminary design choice of using a state machine (Figure 1) to implement the application flow logic proved to be a wise decision. The state machine was used to globally determine the application state. While I initially assumed that this would only provide the benefit of code cleanliness, there proved to be a much more rewarding benefit to this during the testing phase of the project. For example, when issues occurred with initializing the audio driver, instead of having to comment out massive blocks of code in a single application file, I simply disabled the other states in my main.c file to eliminate the possibility of interference from other components. This allowed me to execute the program directly to the point where the issue occurred.

The media center is broken up into 4 core components; the main menu, the photo gallery, the MP3 player, and the game center. The source code is structured in a similar approach to achieve modularity. Each component has its own controller C file that handles all of the logic and I/O for that program. The state machine running in the main.c file calls the main function of the active state's controller file. When that sub-application is finished running, it returns the MAIN_MENU_STATE back to the main.c controller, who then loads the main menu back into the view. In this design approach, I allowed the main function to effectively monitor and task switch, instead of embedding that logic in each sub-application.

The main function initializes the app and all of the hardware components. An infinite loop runs the program continuously while keeping track of the current application state. By calling the main functions of the other components (ie. photo gallery), control is passed to that sub-component and the state of the application is updated. This is implemented by having a pre-defined enumeration type for the application state. Each component's main function returns an AppState.

The main menu is relatively simple and contains a basic UI listing the 3 application options. This module is responsible for tracking the user's current selection while responding to inputs from the joystick. When the select button is triggered, this module's main function will return the selected AppState back to the main controller, which will then pass control to the specified app. A challenge I faced in the selection logic (both in the menu module and other sub-applications containing a selection mechanism) was implementing a cyclic selection structure. I wanted the user to be able to continuously scroll through options in the menu, which automatically cycles back to the beginning of the list when the end is reached.

The photo gallery module performs similarly to the menu module, thus it was easy to piggyback code from the menu.c file. I used the same menu logic to display a list of images to preview, identical to the list of applications in the main menu. In this application, instead of returning the next application state when the joystick was pressed, I called a utility function, image_preview(), to run a subroutine that displayed the c encoded image file on the display. In order to return to the main menu, I used a small arrow image in the top left corner of the LCD to go backwards. If the user wishes to return to the main menu, they can do so using the joystick, in which case this module's main function will return the MainMenu AppState. Again, the media center's main loop observes the state change and passes control back to the main menu.

The MP3 player module connects the USB audio driver to the PC to act as an audio output device. This module handles the logic of connecting to and disconnecting from the PC when entered into this state. The development board's onboard potentiometer is used to control the volume output, with a similar implementation as done in the course labs.

Finally, the game center sub-application presents another menu for game selection. Again, the menu functionality is similar to the main menu and photo gallery. I decided to change my choice of game from flappy bird to tic tac toe since it required less LCD implementation. Based on the success of my modular design, when designing the game center I came to the realization that this was an identical logical program to the media center as a whole; a menu with options to call little sub-applications, in this case games. As a result, I further broke down the game center component into the game center menu (game_center.c) and an individual file for the game logic (tictactoe.c).

The game center design is identical to my explanations above. Where I particularly achieved decomposition is in the tic tac toe program. Beginning with the logic of a single game of tic tac toe, I created a routine (run_tictactoe()), that handles the initialization of the game board and LCD. Considering that a user may want to play several games in a row, I developed a higher level routine, tictactoe_main(), to run new games by calling the run_tictactoe() function. This way, when a game is over, control does not yet get passed back to the game center, but instead stays within the tic tac toe game module until the tictactoe_main() function relinquishes control of the application. This is shown in Figure 2.

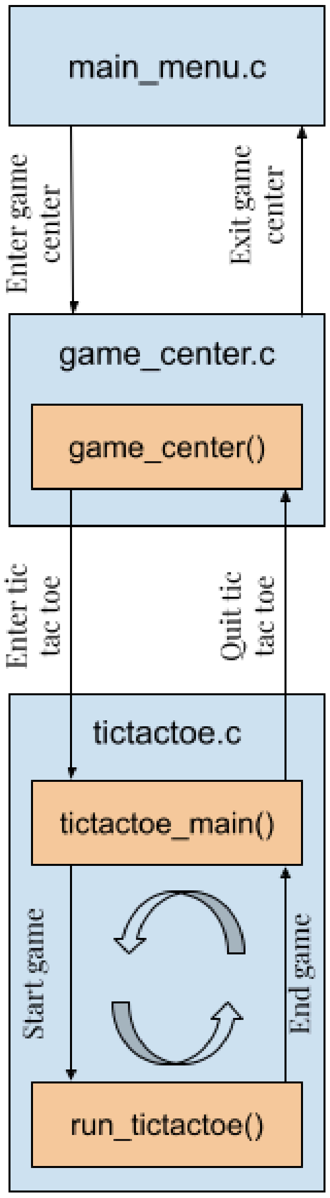

Figure 2

4. Experimental Results

Upon testing the media center application in the lab, a majority of the design I had implemented in simulation worked correctly. There were a few components that needed bug fixing.

The menu selection function, which appears in all modules, had bugs in my original design. I was improperly keeping track of the joystick value leading to the board always placing the cursor on the initialized selection index. I was able to fix this by simplifying to a simple switch case within an infinite for loop.

The audio player module was also a very buggy area that never quite worked properly. It seemed that it was random as to when this performed correctly. In all the in-lab testing conducted, there was one occurrence where the audio output was coming through the MCB1700 development board. Without changing any code at all, when I attempted this again, I got no sound output. This was an area I was unable to correct.

Aside from the above two major issues, the rest of the application performed correctly according to the project requirements.

5. Conclusion

Overall, the media center application was a successful implementation with very minor issues in the end product. The application was completely functional in terms of navigation and state logic. The application utilizes various components on the MCB1700 development board such as the joystick and LCD screen. A state machine proved to be an effective method of managing the application state throughout execution. The photo gallery was able to display C encoded image files, the audio player was able to play audio from the PC at least once, and the game center successfully allowed a game selection menu and one game implementation, tic tac toe. To conclude, the entire project developed my understanding of embedded systems design.

6. References

[1] G. N. Khan, “COE 718 Course Website,” COE 718: Embedded Systems Design, https://www.ecb.torontomu.ca/~courses/coe718/index.html (accessed Nov. 1, 2024).

[1] E. Muzzo, “Final Media Center Project for COE718 - Embedded Systems Design,” GitHub, https://github.com/EricMuzzo/COE718-MediaCenter (accessed Nov. 28, 2024).

7. Appendix

state.h

```
/*---------------------------------------------------------------------------
 * Name:    state.h
 * Purpose: define application states for use in state machine
 * Note(s):
 *---------------------------------------------------------------------------*/

 #ifndef STATE_H
 #define STATE_H

 typedef enum {
   MAIN_MENU_STATE,
   PHOTO_GALLERY_STATE,
   AUDIO_PLAYER_STATE,
   GAME_CENTER_STATE,
   EXIT_STATE
} AppState;

 #endif
```

main.c

```
/*----------------------------------------------------------------------------
 * Name:    main.c
 * Purpose: manage the application state
 * Note(s):
 *----------------------------------------------------------------------------*/

#include <LPC17xx.H>                    /* NXP LPC17xx definitions            */
#include "string.h"
#include "GLCD.h"
#include "LED.h"
#include "KBD.h"
#include "state.h"
#include "menu.h"
#include "photo_gallery.h"
#include "audio_player.h"
#include "game_center.h"

#define __FI        1                   /* Font index 16x24               */
#define USE_LCD                         0

/*----------------------------------------------------------------------------
  Main Program
 *----------------------------------------------------------------------------*/
int main (void) {

   AppState state = MAIN_MENU_STATE;      //initialize the state to main menu

   GLCD_Init();
   KBD_Init();

   #ifdef USE_LCD
      GLCD_Clear(White);
   #endif

   while(state != EXIT_STATE){

      switch(state){
            case MAIN_MENU_STATE:
                state = main_menu();
                break;

            case PHOTO_GALLERY_STATE:
                state = photo_gallery();
                break;

            case AUDIO_PLAYER_STATE:
                state = audio_player();
                break;

            case GAME_CENTER_STATE:
                state = game_center();
                break;

            default:
                state = EXIT_STATE;
                break;
        }
    }
    return 0;
}
```

menu.c

```
/*----------------------------------------------------------------------------
 * Name:    menu.c
 * Purpose: manage the sub-application states
 * Note(s):
 *----------------------------------------------------------------------------*/

#include <stdio.h>
#include <LPC17xx.H>                    /* NXP LPC17xx definitions            */
#include "string.h"
#include "GLCD.h"
#include "LED.h"
#include "KBD.h"
#include "state.h"
#include "cmsis_os.h"

#define __FI        1                      /* Font index 16x24               */
#define USE_LCD                            0

//ITM Stimulus Port definitions for printf //////////////////
#define ITM_Port8(n)    (*((volatile unsigned char *)(0xE0000000+4*n)))
#define ITM_Port16(n)   (*((volatile unsigned short*)(0xE0000000+4*n)))
#define ITM_Port32(n)   (*((volatile unsigned long *)(0xE0000000+4*n)))

#define DEMCR           (*((volatile unsigned long *)(0xE000EDFC)))
#define TRCENA          0x01000000

struct __FILE { int handle;  };
FILE __stdout;
FILE __stdin;

int fputc(int ch, FILE *f) {
  if (DEMCR & TRCENA) {
    while (ITM_Port32(0) == 0);
    ITM_Port8(0) = ch;
  }
  return(ch);
}
```

```
/*------------------------------------------------------------------------
  Main Menu Display Function
  Updates the LCD to indicate the highlighted selection by the user
 *------------------------------------------------------------------------*/
void updateMainMenuDisplay(int selected){

    switch(selected){

        //Selected = 1; Cursor hovered on Photo gallery
        case 1:
            GLCD_SetBackColor(LightGrey);
            GLCD_DisplayString(4, 0, __FI, (unsigned char *)"   Photo Gallery   ");

            GLCD_SetBackColor(White);
            GLCD_DisplayString(5, 0, __FI, (unsigned char *)"   Audio Player    ");
            GLCD_DisplayString(6, 0, __FI, (unsigned char *)"    Game Center    ");
                        break;

        //Selected = 2; Cursor hovered on audio player
        case 2:
            GLCD_SetBackColor(LightGrey);
            GLCD_DisplayString(5, 0, __FI, (unsigned char *)"   Audio Player    ");

            GLCD_SetBackColor(White);
            GLCD_DisplayString(4, 0, __FI, (unsigned char *)"   Photo Gallery   ");
            GLCD_DisplayString(6, 0, __FI, (unsigned char *)"    Game Center    ");
                        break;

        //Selected = 3; Cursor hovered on Photo gallery
        case 3:
            GLCD_SetBackColor(LightGrey);
            GLCD_DisplayString(6, 0, __FI, "    Game Center    ");

            GLCD_SetBackColor(White);
            GLCD_DisplayString(4, 0, __FI, "   Photo Gallery   ");
            GLCD_DisplayString(5, 0, __FI, "   Audio Player    ");
                        break;
    }
```

```
}

/*----------------------------------------------------------------------------
  Main Menu Function
 *----------------------------------------------------------------------------*/
AppState main_menu(){

   int joystick_val;
   int selected_option = 1;                //The menu option currently selected


   //Setup main menu display
   #ifdef USE_LCD

                KBD_Init();
                GLCD_Clear(White);
      GLCD_SetBackColor(Blue);

      GLCD_SetTextColor(Red);
      GLCD_DisplayString(0, 0, __FI, (unsigned char *)"   Media Center   ");

      GLCD_SetTextColor(Black);
      GLCD_DisplayString(1, 0, __FI, (unsigned char *)"     Main Menu    ");

      GLCD_SetBackColor(White);
      GLCD_SetTextColor(Black);
      GLCD_DisplayString(4, 0, __FI, (unsigned char *)"   Photo Gallery  ");

      GLCD_DisplayString(5, 0, __FI, (unsigned char *)"   Audio Player   ");

      GLCD_DisplayString(6, 0, __FI, (unsigned char *)"   Game Center    ");

   #endif

   for(;;){
                updateMainMenuDisplay(selected_option);

                joystick_val = get_button();
                switch(joystick_val){
                        case KBD_SELECT:
                                switch(selected_option){
                                        case 1:
                                                //photo gallery
                                                return PHOTO_GALLERY_STATE;
                                        case 2:
                                                //audio player
                                                return AUDIO_PLAYER_STATE;
                                        case 3:
                                                //game center
                                                return GAME_CENTER_STATE;
                                }
                                break;
                        case KBD_UP:
                                if(selected_option > 1){
                                        selected_option--;
                                }else{
                                        selected_option = 3;
                                }
                                break;
                        case KBD_DOWN:
                                if(selected_option < 3){
                                        selected_option++;
                                }else{
                                        selected_option = 1;
                                }
                                break;
                }
                osDelay(1000);
        }
}
```

photo_gallery.c

```
/*----------------------------------------------------------------------------
 * Name:    photo_gallery.c
 * Purpose: manage the photo gallery application
 * Note(s):
 *----------------------------------------------------------------------------*/

#include <LPC17xx.H>                    /* NXP LPC17xx definitions            */
#include "string.h"
#include "GLCD.h"
#include "LED.h"
#include "KBD.h"
#include "state.h"
#include "falcon9.h"
#include "starship.h"
#include "saturnv.h"
#include "arrow.c"
#include "cmsis_os.h"

#define __FI        1                      /* Font index 16x24               */
#define USE_LCD                 0

unsigned char* images[] = {STARSHIP_PIXEL_DATA, FALCON9_PIXEL_DATA,
SATURNV_PIXEL_DATA};

/*----------------------------------------------------------------------------
  Initialize the display for the photo viewer application
 *----------------------------------------------------------------------------*/
void init_display(){

   #ifdef USE_LCD
                    GLCD_Clear(White);
      GLCD_SetBackColor(Blue);

      GLCD_SetTextColor(Red);

      GLCD_Bitmap(0, 0, 16, 16, ARROW_PIXEL_DATA);
      GLCD_DisplayString(0, 3, __FI, (unsigned char *)"Photo Gallery");
```

```
        GLCD_SetBackColor(White);
        GLCD_SetTextColor(Black);
        GLCD_DisplayString(3, 0, __FI, (unsigned char *)"Starship");        //Selection 1
        GLCD_DisplayString(4, 0, __FI, (unsigned char *)"Falcon 9");        //Selection 2
        GLCD_DisplayString(5, 0, __FI, (unsigned char *)"Saturn V");        //Selection 3

    #endif
}

/*-----------------------------------------------------------------------------
  Update the LCD based on selected item
 *------------------------------------------------------------------------------*/
void update_display(int selected){

    switch(selected){

        //Selected = 0; hovered on the back button
        case 0:
            GLCD_SetBackColor(White);

            GLCD_DisplayString(3, 0, __FI, (unsigned char *)"Starship");
            GLCD_DisplayString(4, 0, __FI, (unsigned char *)"Falcon 9");
            GLCD_DisplayString(5, 0, __FI, (unsigned char *)"Saturn V");
                            break;

        //Selected = 1
        case 1:
            GLCD_SetBackColor(LightGrey);
            GLCD_DisplayString(3, 0, __FI, (unsigned char *)"Starship");

            GLCD_SetBackColor(White);
            GLCD_DisplayString(4, 0, __FI, (unsigned char *)"Falcon 9");
            GLCD_DisplayString(5, 0, __FI, (unsigned char *)"Saturn V");
                            break;

        //Selected = 2
        case 2:
            GLCD_SetBackColor(LightGrey);
            GLCD_DisplayString(4, 0, __FI, (unsigned char *)"Falcon 9");

            GLCD_SetBackColor(White);
            GLCD_DisplayString(3, 0, __FI, (unsigned char *)"Starship");
            GLCD_DisplayString(5, 0, __FI, (unsigned char *)"Saturn V");
                        break;

        //Selected = 3
        case 3:
            GLCD_SetBackColor(LightGrey);
            GLCD_DisplayString(5, 0, __FI, (unsigned char *)"Saturn V");

            GLCD_SetBackColor(White);
            GLCD_DisplayString(3, 0, __FI, (unsigned char *)"Starship");
            GLCD_DisplayString(4, 0, __FI, (unsigned char *)"Falcon 9");
                        break;
    }
}

/*----------------------------------------------------------------------------
  Preview's the selected image file
 *----------------------------------------------------------------------------*/
void image_preview(int selected_image){

        int joystick_val;
    int index = selected_image - 1;

    GLCD_Clear(White);
    GLCD_Bitmap(0, 0, 320, 240, images[index]);

    //Wait for left joystick to be pressed to return to photo menu
    for(;;){
                joystick_val = get_button();
                if(joystick_val == KBD_LEFT){
                        init_display();                         //re-initialize display
                        break;
                }
        }
}
```

```
AppState photo_gallery(){

   int joystick_val;
   int selected = 1;

   init_display();

         for(;;){
                  update_display(selected);
                  joystick_val = get_button();

                  switch(joystick_val){
                           case KBD_SELECT:
                                    if(selected == 0){
                                             return MAIN_MENU_STATE;
                                    }else{
                                             image_preview(selected);
                                    }
                                    break;
                           case KBD_UP:
                                    if(selected > 1){
                                             selected--;
                                    }else{
                                             selected = 3;
                                    }
                                    break;
                           case KBD_DOWN:
                                    if(selected < 3){
                                             selected++;
                                    }else{
                                             selected = 1;
                                    }
                                    break;
                           case KBD_LEFT:
                                    selected = 0;
                                    break;
                  }
                  osDelay(1000);
         }
}
```

audio_player.c

```
/*----------------------------------------------------------------------------
 * Name:    audio_player.c
 * Purpose: manage the audio player application
 * Note(s):
 *----------------------------------------------------------------------------*/

#include <LPC17xx.h>                    /* NXP LPC17xx definitions            */
#include "string.h"
#include "GLCD.h"
#include "LED.h"
#include "KBD.h"
#include "state.h"
#include <stdio.h>

#include "type.h"
#include "usb.h"
#include "usbcfg.h"
#include "usbhw.h"
#include "usbcore.h"
#include "usbaudio.h"
#include "cmsis_os.h"


#define __FI        1                      /* Font index 16x24               */
#define USE_LCD                             0

#define ITM_Port8(n)    (*((volatile unsigned char *)(0xE0000000+4*n)))
#define ITM_Port16(n)   (*((volatile unsigned short*)(0xE0000000+4*n)))
#define ITM_Port32(n)   (*((volatile unsigned long *)(0xE0000000+4*n)))

#define DEMCR           (*((volatile unsigned long *)(0xE000EDFC)))
#define TRCENA          0x01000000

extern  void SystemClockUpdate(void);
extern uint32_t SystemFrequency;
uint8_t  Mute;                                 /* Mute State */
uint32_t Volume;                                 /* Volume Level */
```

```
#if USB_DMA
   uint32_t *InfoBuf = (uint32_t *)(DMA_BUF_ADR);
   short *DataBuf = (short *)(DMA_BUF_ADR + 4*P_C);
#else
   uint32_t InfoBuf[P_C];
   short DataBuf[B_S];                    /* Data Buffer */
#endif

uint16_t  DataOut;                        /* Data Out Index */
uint16_t  DataIn;                         /* Data In Index */

uint8_t   DataRun;                        /* Data Stream Run State */
uint16_t  PotVal;                         /* Potenciometer Value */
uint32_t  VUM;                            /* VU Meter */
uint32_t  Tick;                           /* Time Tick */


/*----------------------------------------------------------------------------
  Get Potentiometer value
 *----------------------------------------------------------------------------*/
void get_potval (void) {
  uint32_t val;

  LPC_ADC->CR |= 0x01000000;              /* Start A/D Conversion */
  do {
    val = LPC_ADC->GDR;                   /* Read A/D Data Register */
  } while ((val & 0x80000000) == 0);      /* Wait for end of A/D Conversion */
  LPC_ADC->CR &= ~0x01000000;             /* Stop A/D Conversion */
  PotVal = ((val >> 8) & 0xF8) +          /* Extract Potenciometer Value */
         ((val >> 7) & 0x08);
}


/*----------------------------------------------------------------------------
  Timer Counter 0 Interrupt Service Routine
  executed each 31.25us (32kHz frequency)
 *----------------------------------------------------------------------------*/
void TIMER0_IRQHandler(void){
   long  val;
```

```
int32_t joystick_val;
uint32_t cnt;

if (DataRun) {                          /* Data Stream is running */
   val = DataBuf[DataOut];                   /* Get Audio Sample */
   cnt = (DataIn - DataOut) & (B_S - 1);       /* Buffer Data Count */
   if (cnt == (B_S - P_C*P_S)) {              /* Too much Data in Buffer */
      DataOut++;                        /* Skip one Sample */
   }
   if (cnt > (P_C*P_S)) {                  /* Still enough Data in Buffer */
      DataOut++;                        /* Update Data Out Index */
   }

   DataOut &= B_S - 1;                    /* Adjust Buffer Out Index */

   if (val < 0) VUM -= val;                 /* Accumulate Neg Value */

   else       VUM += val;                 /* Accumulate Pos Value */

   val  *= Volume;                       /* Apply Volume Level */
   val >>= 16;                          /* Adjust Value */
   val  += 0x8000;                       /* Add Bias */
   val  &= 0xFFFF;                        /* Mask Value */
}
else{
   val = 0x8000;                        /* DAC Middle Point */
}

if (Mute) {
   val = 0x8000;                        /* DAC Middle Point */
}

LPC_DAC->CR = val & 0xFFC0;                  /* Set Speaker Output */

if ((Tick++ & 0x03FF) == 0) {               /* On every 1024th Tick */
   get_potval();                        /* Get Potenciometer Value */
   if (VolCur == 0x8000) {                /* Check for Minimum Level */
      Volume = 0;                       /* No Sound */
   } else {
      Volume = VolCur * PotVal;             /* Chained Volume Level */
      }
      val = VUM >> 20;                          /* Scale Accumulated Value */
      VUM = 0;                             /* Clear VUM */
      if (val > 7) val = 7;                 /* Limit Value */
   }

   LPC_TIM0->IR = 1;                        /* Clear Interrupt Flag */

   joystick_val = get_button();

   if(joystick_val == KBD_LEFT){
      NVIC_DisableIRQ(TIMER0_IRQn);
      NVIC_DisableIRQ(USB_IRQn);
   }
}


/*----------------------------------------------------------------------------
  Main Audio Function
 *----------------------------------------------------------------------------*/
AppState audio_player(void){

   volatile uint32_t pclkdiv, pclk;

   GLCD_Clear(White);
   GLCD_SetBackColor(Blue);
   GLCD_SetTextColor(White);
   GLCD_DisplayString(0, 0, __FI, (unsigned char *)"     MP3 Player     ");

   GLCD_SetBackColor(White);
   GLCD_SetTextColor(Black);
   GLCD_DisplayString(1, 0, __FI, (unsigned char *)"Push left to return ");

   //could possibly insert a music icon here

   /* SystemClockUpdate() updates the SystemFrequency variable */
   SystemClockUpdate();

   LPC_PINCON->PINSEL1 &=~((0x03<<18)|(0x03<<20));
```

```
/* P0.25, A0.0, function 01, P0.26 AOUT, function 10 */
LPC_PINCON->PINSEL1 |= ((0x01<<18)|(0x02<<20));

/* Enable CLOCK into ADC controller */
LPC_SC->PCONP |= (1 << 12);

LPC_ADC->CR = 0x00200E04;          /* ADC: 10-bit AIN2 @ 4MHz */
LPC_DAC->CR = 0x00008000;          /* DAC Output set to Middle Point */


/* By default, the PCLKSELx value is zero, thus, the PCLK for
all the peripherals is 1/4 of the SystemFrequency. */
/* Bit 2~3 is for TIMER0 */
pclkdiv = (LPC_SC->PCLKSEL0 >> 2) & 0x03;
switch( pclkdiv ){
  case 0x00:
  default:
   pclk = SystemFrequency/4;
  break;
  case 0x01:
   pclk = SystemFrequency;
  break;
  case 0x02:
   pclk = SystemFrequency/2;
  break;
  case 0x03:
   pclk = SystemFrequency/8;
  break;
}

LPC_TIM0->MR0 = pclk/DATA_FREQ - 1;      /* TC0 Match Value 0 */
LPC_TIM0->MCR = 3;                       /* TCO Interrupt and Reset on MR0
*/
LPC_TIM0->TCR = 1;                       /* TC0 Enable */
NVIC_EnableIRQ(TIMER0_IRQn);

USB_Init();                    /* USB Initialization */
USB_Connect(TRUE);             /* USB Connect */

return MAIN_MENU_STATE;
```

```
}
```

game_center.c

```
/*----------------------------------------------------------------------------
 * Name:    game_center.c
 * Purpose: manage the game center application
 * Note(s):
 *----------------------------------------------------------------------------*/

#include <LPC17xx.H>                    /* NXP LPC17xx definitions           */
#include "string.h"
#include "GLCD.h"
#include "LED.h"
#include "KBD.h"
#include "state.h"
#include "tictactoe.h"
#include "arrow.c"
#include "cmsis_os.h"

#define __FI        1                      /* Font index 16x24               */
#define USE_LCD                 0


/*----------------------------------------------------------------------------
  Initialize the display for the game center main menu
 *----------------------------------------------------------------------------*/
void init_gc_display(){

   #ifdef USE_LCD
                    GLCD_Clear(White);
      GLCD_SetBackColor(Blue);

      GLCD_SetTextColor(Red);

      GLCD_Bitmap(0, 0, 16, 16, ARROW_PIXEL_DATA);
      GLCD_DisplayString(0, 3, __FI, (unsigned char *)"Game Center");


      GLCD_SetBackColor(White);
```

```
        GLCD_SetTextColor(Black);
        GLCD_DisplayString(3, 0, __FI, (unsigned char *)"     Tic Tac Toe      ");    //Selection 1

        //Use these if you have more than one game
        //GLCD_DisplayString(4, 0, __FI, (unsigned char *)"Game 2");                //Selection 2
        //GLCD_DisplayString(5, 0, __FI, (unsigned char *)"Game 3");                //Selection 3

    #endif
}

/*---------------------------------------------------------------------------
  Update the LCD based on selected item
 *---------------------------------------------------------------------------*/
void update_gc_display(int selected){

    switch(selected){

        //Selected = 0; hovered on the back button
        case 0:
            GLCD_SetBackColor(White);

            GLCD_DisplayString(3, 0, __FI, (unsigned char *)"     Tic Tac Toe      ");
            //GLCD_DisplayString(4, 0, __FI, (unsigned char *)"Game 2");
            //GLCD_DisplayString(5, 0, __FI, (unsigned char *)"Game 3");
            break;

        //Selected = 1
        case 1:
            GLCD_SetBackColor(LightGrey);
            GLCD_DisplayString(3, 0, __FI, (unsigned char *)"     Tic Tac Toe      ");

            GLCD_SetBackColor(White);
            //GLCD_DisplayString(4, 0, __FI, (unsigned char *)"Game 2");
            //GLCD_DisplayString(5, 0, __FI, (unsigned char *)"Game 3");
            break;

        /*
        Left here in case more games are added

        //Selected = 2
        case 2:
            GLCD_SetBackColor(LightGrey);
            GLCD_DisplayString(4, 0, __FI, (unsigned char *)"Falcon 9");

            GLCD_SetBackColor(White);
            GLCD_DisplayString(3, 0, __FI, (unsigned char *)"Starship");
            GLCD_DisplayString(5, 0, __FI, (unsigned char *)"Saturn V");

        //Selected = 3
        case 3:
            GLCD_SetBackColor(LightGrey);
            GLCD_DisplayString(5, 0, __FI, (unsigned char *)"Saturn V");

            GLCD_SetBackColor(White);
            GLCD_DisplayString(3, 0, __FI, (unsigned char *)"Starship");
            GLCD_DisplayString(4, 0, __FI, (unsigned char *)"Falcon 9");
        */
    }
}

AppState game_center(void){

    int joystick_val;                  //The joystick value
    int selected = 1;                  //The menu item selected

    init_gc_display();

    for(;;){
        update_gc_display(selected);
        joystick_val = get_button();

        switch(joystick_val){
            case KBD_SELECT:
              if(selected == 0){
                return MAIN_MENU_STATE;
              }else{
                tictactoe_main();
                init_gc_display();
                osDelay(10000);
              }
```

```
                break;
            case KBD_DOWN:
                selected = 1;
                break;
            case KBD_LEFT:
                selected = 0;
                break;
        }
        osDelay(1000);
    }
}
```

tictactoe.c

```
/*----------------------------------------------------------------------------
 * Name:    tictactoe.c
 * Purpose: Houses the game logic for tic tac toe
 * Note(s):
 *----------------------------------------------------------------------------*/

#include <LPC17xx.H>                    /* NXP LPC17xx definitions            */
#include "string.h"
#include "GLCD.h"
#include "LED.h"
#include "KBD.h"
#include "state.h"
#include <stdbool.h>
#include "cmsis_os.h"

#include "tictactoe_o.h"
#include "tictactoe_x.h"

#define __FI        1                      /* Font index 16x24               */
#define USE_LCD                          0

/*----------------------------------------------------------------------------
 *                      Game Variables
 *----------------------------------------------------------------------------*/
#define GRID_ROWS   3
#define GRID_COLS   3

char board[GRID_ROWS][GRID_COLS];           //Game board
char current_player;                                   //Current player X or O
int selected_row;                          //Selected row on the game board
int selected_col;                          //Selected column on the game board
bool game_over;                            //True = game over, False = game in progress


/*----------------------------------------------------------------------------
 *                      Game Setup
 *  Setup the board
 *-------------------------------------------------------------------------*/

//  Setup the game board
void init_board(){

    int i, j;

    for (i = 0; i < GRID_ROWS; i++) {
        for (j = 0; j < GRID_COLS; j++) {
            board[i][j] = ' ';
        }
    }
}

//  Draw the game grid
void draw_grid(){

    int x, y;

    GLCD_Clear(Black);
    GLCD_SetBackColor(Black);
    GLCD_SetTextColor(Red);

    //Draw vertical lines at x = 106 & x = 213 (Screen width 320/3 ~= every 106 pixels)
    for(y = 0; y < 240; y++){

        //Will draw pixels on the left and right of x=106&213 to give the lines some thickness
        GLCD_PutPixel(105, y);
        GLCD_PutPixel(106, y);
        GLCD_PutPixel(107, y);

        GLCD_PutPixel(212, y);
        GLCD_PutPixel(213, y);
        GLCD_PutPixel(214, y);
    }

    //Draw horizontal lines at y = 80 & y = 160 (Screen height 240/3 ~= every 80 pixels)
    for(x = 0; x < 320; x++){

        GLCD_PutPixel(x, 79);
        GLCD_PutPixel(x, 80);
        GLCD_PutPixel(x, 81);

        GLCD_PutPixel(x, 159);
        GLCD_PutPixel(x, 160);
        GLCD_PutPixel(x, 161);
    }

    /*
    //Alternative visual: make the gameboard square (length of horizontal & vertical lines the
same)
    for(int y = 0; y < 240; y++){
        GLCD_PutPixel(106, y);
        GLCD_PutPixel(213, y);
    }

    //Draw horizontal lines at y = 80 & y = 160 (Screen height 240/3 ~= every 80 pixels)
    for(int x = 80; x < 240; x++){
        GLCD_PutPixel(x, 80);
        GLCD_PutPixel(x, 160);
    }
    */
}

/*----------------------------------------------------------------------------
 *                         Auxillary LCD functions
 *---------------------------------------------------------------------------*/

//Initialize tic tac toe main menu display
void init_ttt_lcd(){

    GLCD_Clear(Black);
    GLCD_SetBackColor(Black);
    GLCD_SetTextColor(White);

    GLCD_DisplayString(2, 0, __FI, (unsigned char *)"    Tic Tac Toe     ");
    GLCD_DisplayString(4, 0, __FI, (unsigned char *)"Push SELECT To Start");
    GLCD_DisplayString(6, 0, __FI, (unsigned char *)"Push LEFT To Go Back");
}
```

```
//  Highlights the cell that the cursor is on by drawing a box around the cell
void highlight_cell(int row, int col){

    int x, y;

    int start_x = col * 106;
    int end_x = start_x + 106;
    int start_y = row * 80;
    int end_y = start_y + 80;

    GLCD_SetTextColor(Green);
    for(x = start_x; x < end_x; x++){
        GLCD_PutPixel(x, start_y);              //Simultaneously highlight top and bottom edges
        GLCD_PutPixel(x, end_y);
    }

    for(y = start_y; y < end_y; y++){
        GLCD_PutPixel(start_x, y);              //Simultaneously highlight left and right edges
        GLCD_PutPixel(end_x, y);
    }
}

//  Draw the current board state to the LCD
void draw_board(){

    int i, j, x, y;

    for(i = 0; i < GRID_ROWS; i++){
        for(j = 0; j < GRID_COLS; j++){

            x = (j * 106) + 18;
            y = (i * 80) + 5;

            if(board[i][j] != ' '){
                switch(board[i][j]){
                    case 'X':
                        GLCD_Bitmap(x, y, TICTACTOE_X_WIDTH, TICTACTOE_X_HEIGHT,
TICTACTOE_X_PIXEL_DATA);
                        break;
                    case 'O':
```

```
                    GLCD_Bitmap(x, y, TICTACTOE_O_WIDTH, TICTACTOE_O_HEIGHT,
TICTACTOE_O_PIXEL_DATA);
                                                        break;
                }
            }
        }
    }
}


/*--------------------------------------------------------------------------
 *                         Game Logic Functions
 *--------------------------------------------------------------------------*/

//  Check for a winner
bool check_for_winner(){

    int i;

    //Checking rows and columns
    for(i = 0; i < GRID_ROWS; i++) {
        if (board[i][0] == current_player && board[i][1] == current_player && board[i][2] ==
current_player){
                                return true;
                        }
        if (board[0][i] == current_player && board[1][i] == current_player && board[2][i] ==
current_player){
                                return true;
                        }
    }

    //Checking diagonals
    if (board[0][0] == current_player && board[1][1] == current_player && board[2][2] ==
current_player){
                        return true;
            }
    if (board[0][2] == current_player && board[1][1] == current_player && board[2][0] ==
current_player){
        return true;
    }
    return false;
}

//  Check if board is full
bool is_board_full(){

    int i, j;

    for (i = 0; i < GRID_ROWS; i++) {
        for (j = 0; j < GRID_COLS; j++) {
            if (board[i][j] == ' '){
                return false;
            }
        }
    }
    return true;
}


/*--------------------------------------------------------------------------
 *                          Main Game Loop
 * Notes: sets up an instance of the tic tac toe game
 *--------------------------------------------------------------------------*/

void run_tictactoe(){

    int joystick_val;
    selected_row = 1;
    selected_col = 1;
    game_over = false;
    current_player = 'X';

    init_board();
    draw_board();
    draw_grid();

    while(!game_over){

        highlight_cell(selected_row, selected_col);
```

```
//-----------Input Handler--------------------------
joystick_val = get_button();

switch(joystick_val){

   case KBD_UP:
      if(selected_row > 0){
         selected_row--;
      }else{
         selected_row = 2;
      }
      draw_grid();
      draw_board();
      highlight_cell(selected_row, selected_col);
      break;

   case KBD_DOWN:
      if(selected_row < (GRID_ROWS - 1)){
         selected_row++;
      }else{
         selected_row = 0;
      }
      draw_grid();
      draw_board();
      highlight_cell(selected_row, selected_col);
      break;

   case KBD_LEFT:
      if(selected_col > 0){
         selected_col--;
      }else{
         selected_col = 2;
      }
      draw_grid();
      draw_board();
      highlight_cell(selected_row, selected_col);
      break;

   case KBD_RIGHT:
```

```
            if(selected_col < (GRID_COLS - 1)){
               selected_col++;
            }else{
               selected_col = 0;
            }
            draw_grid();
            draw_board();
            highlight_cell(selected_row, selected_col);
            break;

         case KBD_SELECT:
            if(board[selected_row][selected_col] == ' '){                    //if cell is empty, allow user to
select

               board[selected_row][selected_col] = current_player;         //Save current player char
to game board
               draw_grid();
                                                  draw_board();

               if(check_for_winner()){
                  GLCD_SetBackColor(Red);
                  GLCD_SetTextColor(Black);
                  GLCD_DisplayString(6, 0, __FI, (unsigned char *)"      Winner!       ");

                  //Small delay
                  osDelay(5000);
                  game_over = true;
                  return;
               }

               else if(is_board_full()){

                  GLCD_SetBackColor(Red);
                  GLCD_SetTextColor(Black);
                  GLCD_DisplayString(6, 0, __FI, (unsigned char *)"         Tie        ");

                  //Small delay
                  osDelay(5000);
                  game_over = true;
                  return;
```

```
                }
                else{
                    if(current_player == 'X'){
                        current_player = 'O';
                    }
                    else{
                        current_player = 'X';
                    }
                }
            }
            break;
        }
    }

}

/*---------------------------------------------------------------------------
 *                        Tic Tac Toe Main Loop
 * Notes: the loop that starts a game, prompts users to play again, or gives the
 *        option to exit back to the game center
 *---------------------------------------------------------------------------*/

void tictactoe_main(){

    int joystick_val;
    bool running = true;

    init_ttt_lcd();

    while(running){

        joystick_val = get_button();

        switch(joystick_val){
            case KBD_SELECT:
                run_tictactoe();
                init_ttt_lcd();
                break;

            case KBD_LEFT:
                running = false;
                return;
            }
        }

    }
```