Table of Contents

1. Abstract

This report outlines the design and implementation of a Media Center using the MCB1700 development board and uVision IDE as part of the Embedded Systems Design course (COE718). The Media Center features a photo gallery, an MP3 player, and an interactive game, all navigated through the joystick and displayed on the onboard LCD screen. By combining hardware and software, this project demonstrates the integration of embedded systems concepts to create an interactive multimedia application.

2. Introduction

The knowledge and skills obtained through the labs completed in COE718 aided with the success of the media center project. The main requirements for this application includes integrating 3 main features: the photo gallery to view photos, the MP3 player to stream audio through the board and an interactive game which in this case is 2048. The photo gallery displays bitmap images which was done by converting PNG to a c file using the software tool GIMP. The MP3 player was designed to stream audio from the PC to the board through USB and the potentiometer on the board was used to control the volume of the audio. Finally, the game 2048 implemented is a sliding tile puzzle game that allows the user to combine numbered tiles to achieve a score of 32. All components of the Media Center are seamlessly navigated using the MCB1700's joystick, allowing users to interact with the system through the onboard LCD screen. This report provides an in-depth analysis of the integration process, design methodology, and outcomes achieved during the development of the Media Center project.

3. Past Work & Review

Throughout this course, 5 labs were completed which all contributed to the success of this project. The labs helped improve the skills required to work with the Keil uVision & ARM Cortex M3 environment, specifically executing code on the LPC1768 microcontroller. From the labs, we learned how to program and interact with peripherals such as the joystick and LCD display, which were essential for the Media Center project. These labs provided hands-on experience on how hardware components should be controlled and how menus and outputs can be displayed on the LCD screen. Furthermore, the labs also improved our coding skills with C language, dealing with modules in programming and debugging codes that were very effective in dealing with multiple integration and operational issues of the Media Center application.

4. Methodology

The project was broken down into 4 stages: implementing the main media center menu, photo gallery, MP3 player and game. Further detail for the following stages can be seen as follows.

4.1 Main Menu

The purpose of the main menu is to allow the user to navigate through the 3 features of the media center: Photo Gallery, MP3 Player, and Game. This was implemented by creating an array to store the menu items, which was later then called in the displayMenu() function. In order to

switch between menu items, a switch case was implemented based on the index of the array where index 0 displays the photo gallery option, index 1 displays the MP3 option and index 2 displays the game option. This methodology can be seen in the flow chart in section 5.

Another switch case was implemented to actually run each of the menu features. If KBD_SELECT was invoked while the index was at either 0, 1 or 2, the corresponding menu item function would run. More details on each function's functionality is described in 4.2, 4.3 and 4.4.

Fig 1: *Main Menu*

4.2 Photo Gallery

When the user selects ‘Photo Gallery’ from the main menu, the program then runs photo.c. This results in the LCD display being cleared to them displaying the first image in an array of images. The GLCD_Bitmap() function allows the user to view the images in the photo gallery and a switch case is implemented to navigate through the array of images. When the user provides an input that corresponds to KBD_LEFT and KBD_RIGHT, the image array is then incremented or decremented accordingly, allowing the user to switch between images using the on board joystick. When the program recognizes a KBD_SELECT, meaning the user has pressed down on the joystick, the LCD display will clear and return to the main menu.

4.3 MP3 Player

When the user selects ‘MP3 Player’ from the main menu, the program runs usbdmain.c. This results in the LCD display to clear and display a welcome to the music center message along with an image of a music note. At the bottom of the screen, the hex value of the potentiometer value is displayed and changes when the potentiometer is increased or decreases, mimicking the

functionality of a volume button. The MP3 player program first needs to initialize all the peripherals and then enters an infinite loop that is continuously retrieving audio data from the computer for audio playback. Once the select button is pressed, this causes the program to exit the infinite loop, clears the display and returns the user to the main menu.

4.4 Game

When the user selects ‘Game’ from the main menu, the program runs game.c. This results in the LCD display to clear and display a message on how to play the game. The interactive game chosen was a variation of 2048 where in order to win the game, the user must achieve a tile of 32. This allows the game to be won in a shorter amount of time, allowing the user to enjoy the game in a shorter amount of time.

During this message display, the program invokes a short delay to allow the user to read through the instructions and after the delay, the LCD is cleared and the game tiles appear. A 4 by 4 grid layout appears with 2 tiles placed on the board at random. The program creates a random seed using the clock to place 2 tiles at random. The user then can move the joystick up, down, left or right to shift the tiles and merge two together. This is done in the program by first shifting the tiles in the direction of the joystick and then calling a function to merge the tiles if they are next to each other. After the user moves the joystick, the LCD display will clear and redraw the grid with the tiles placed in the new location with the updated values. This continues until the user achieves a tile of 32, which then results in a ‘you have won’ message to display. Another delay is invoked during this, so that the user has time to read the prompt. Then the flow of control takes the user back to the main menu of the media center to choose their next task that they would like to run.

5. Design

For the media center design the primary aspect of the embedded system is that it is a menu driven application that uses user input from the on board joystick to navigate and select the tasks the user wants to use. The design starts by initializing peripherals such as the joystick, potentiometer conversion, and the LCD followed by displaying the initial main menu for the user to see. After this point in the process the user can select among three options: a photo gallery, an MP3 player, and a game. The selection is made by moving the joystick in either the up or down direction, this increments or decrements the disp_val value. Depending on the disp_val value, the system updates the display to reflect the current menu selection, allowing users to choose and confirm their desired option by pressing select.

Once a selection is made the flow of control is passed to the specified c file, and the system initializes the required peripherals that are required to make the c file execute properly. For example in the case of selecting the photo gallery, the program begins by initializing the display and joystick. Then the program responds to joystick inputs to navigate between the images. Similarly in the MP3 player the program initializes the potentiometer ADC, joystick and the LCD. The user can adjust the

audio based on the potentiometer value and the program uses this along with streamed audio data from the computer to process audio output. The game handles joystick inputs to update and check game states, such as win conditions. After completing tasks in any module, users can exit and return to the main menu. This design ensures modularity and a clear separation of functionalities.

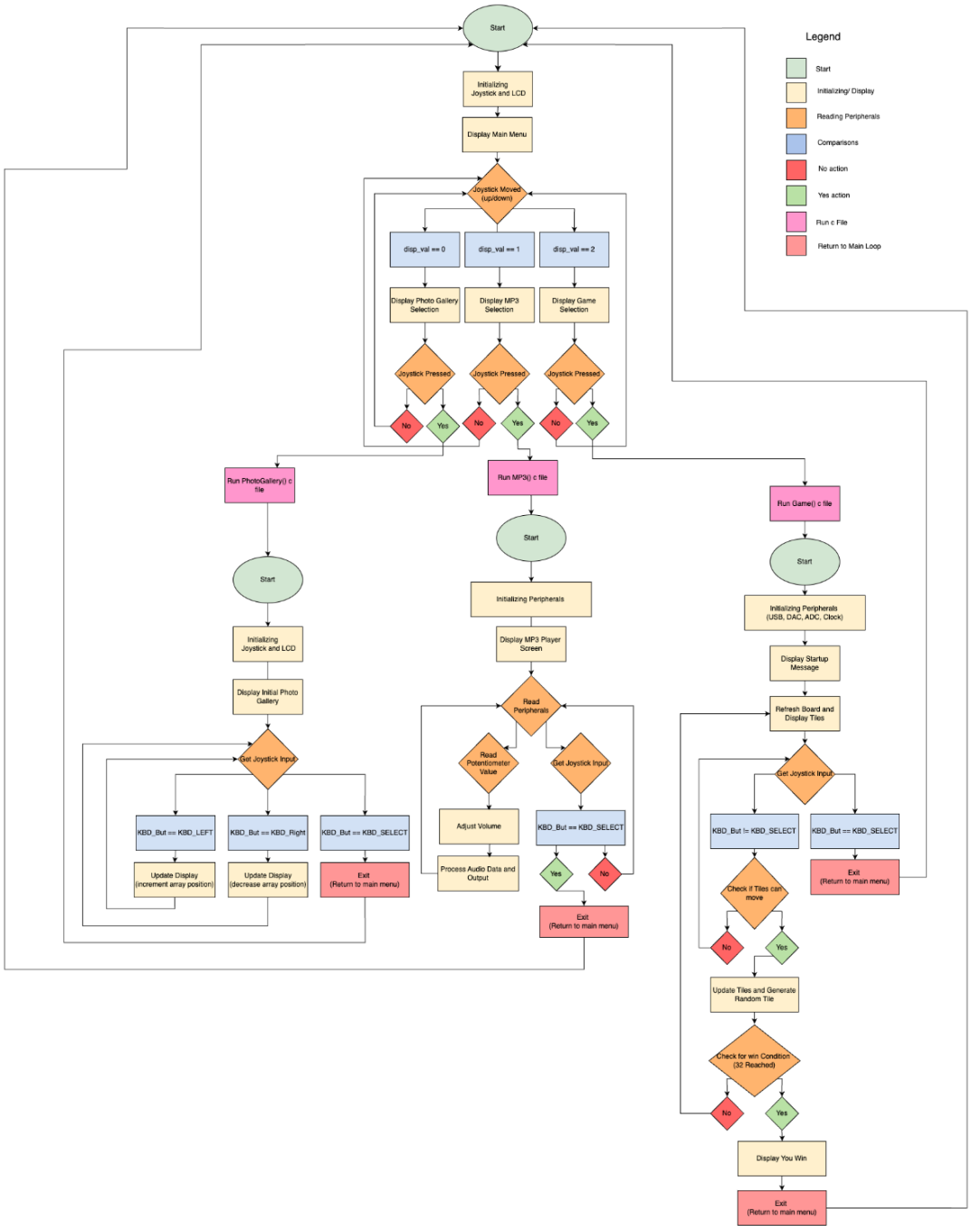


Fig 2: *Flow chart showcasing the flow of the entire media center application*

6. Experimental Results

6.1 Photo gallery

Figure 3 below demonstrates the flow of the photo gallery where there are 3 pictures the user can navigate between using the joystick. These pictures were converted from a png to c file using the softwar4e GIMP, where first the image needed to be scaled down and then flipped horizontally. Then the file was saved as a C file and imported into the project folder. Each image contained its own pixel data pointer that was stored into an array and called in photoGallery.c.

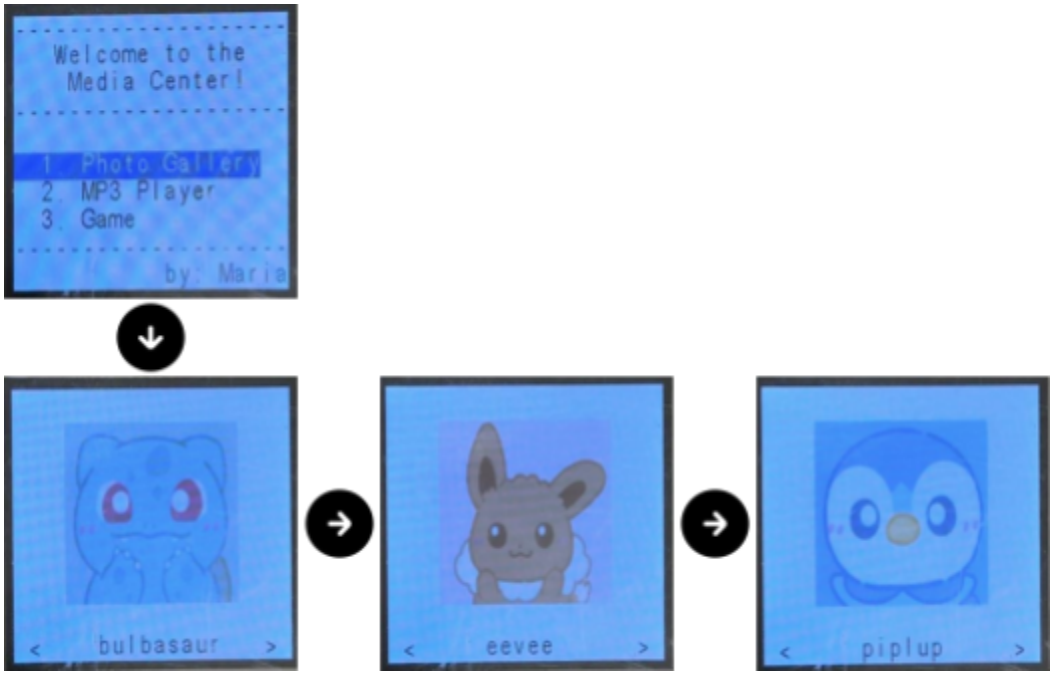


Fig 3: *Flow of the Photo Gallery Feature*

6.2 MP3 player

Figure 4 below shows the results for the MP3 player. When the user selects the MP3 player, a message on the computer prompts the user to connect to the sound card on the computer to enable audio playback. The LCD screen displays a welcome message as well as an image. The bottom of the screen shows the potentiometer value as a hex number which updates as the potentiometer is moved on the board.

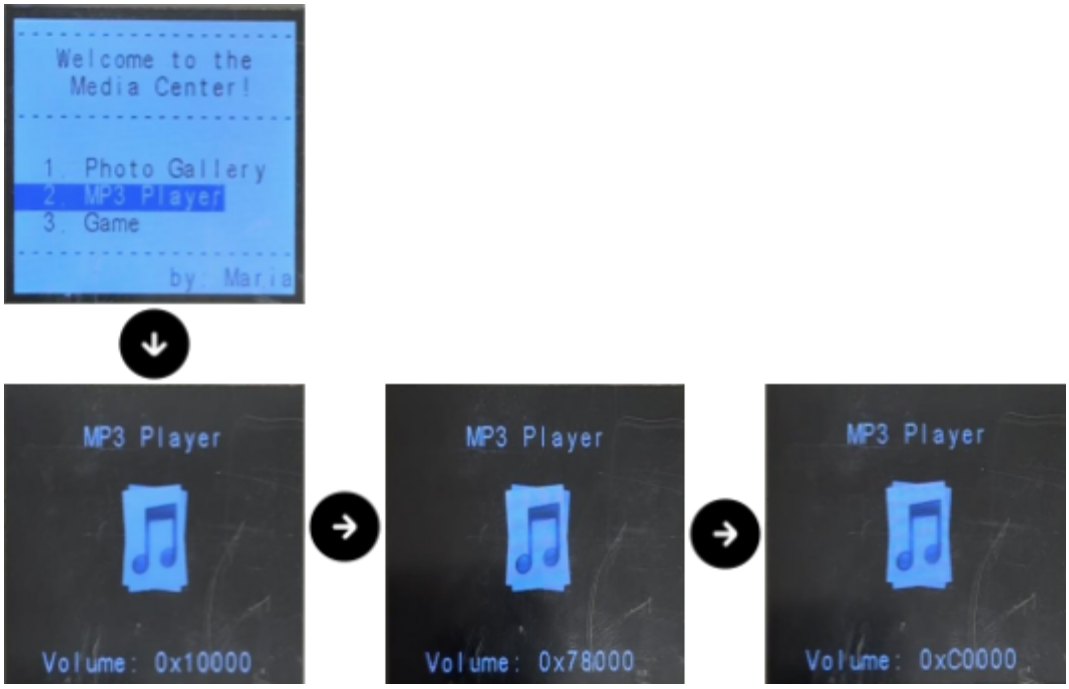
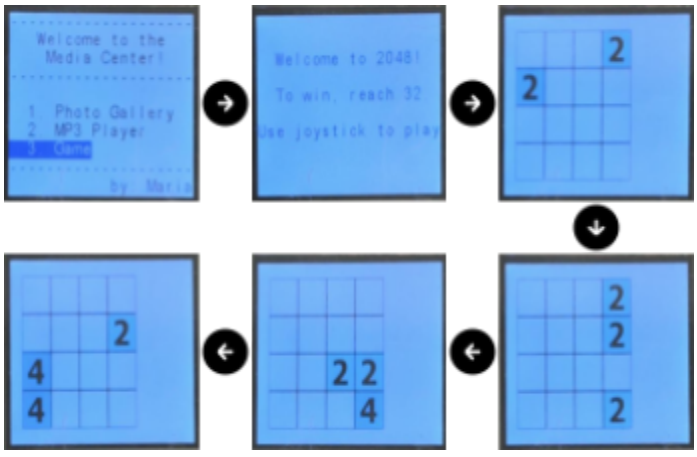
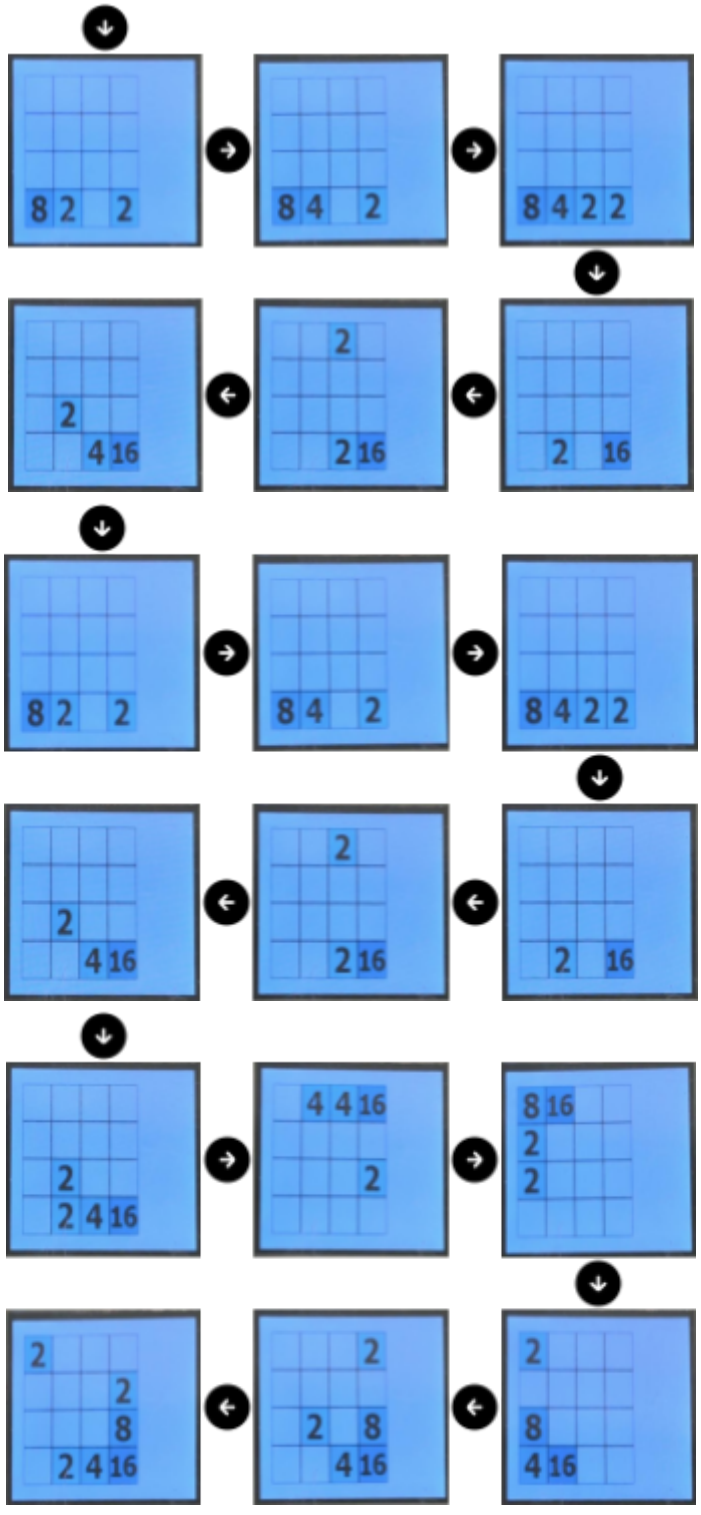


Fig 4: *Flow MP3 player Feature where the volume value can be seen changing*

6.3 Game

This game is a variation of the classic 2048 game, implemented in a simplified manner where the user wins when reaching a tile with the value of 32. The user controls the game using the joystick, moving tiles in four directions (up, down, left, right). Tiles with the same number merge when they collide during movement, creating a new tile with a doubled value. The goal is to strategically merge tiles to achieve the target tile of 32 without filling up the grid.




8 2 2
8 4 2
8 4 2 2
2 16
2
2 16
2
4 16
8 2 2
8 4 2
8 4 2 2
2 16
2
4 16
4 4 16
2
8 16
2
2
2
2 4 16
2
2 8
4 16
2
8
4 16
2
2
8
2 4 16

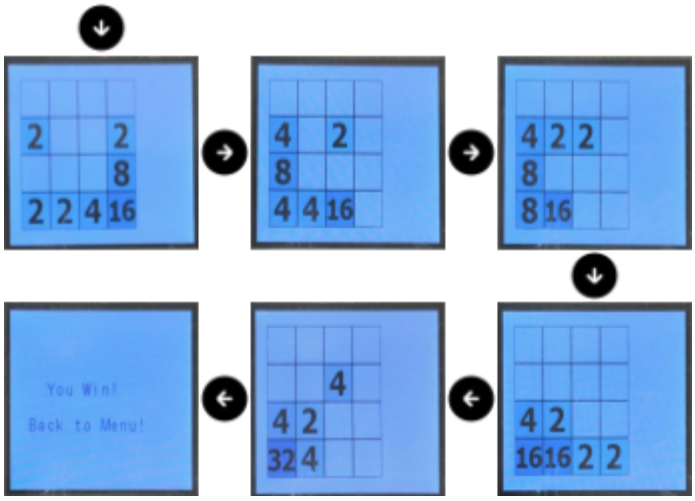


Fig 5: *Continuation of the flow of the Game 2048 feature*

7. Conclusion

The completion of this project demonstrated the ability to combine software and hardware to produce an embedded system for the application of a media center. We were able to successfully learn how to set up audio drivers to stream audio to the onboard speaker, and how to display images on the LCD using GLCD_Bitmap(). Additionally, we improved our proficiency with graphical user interfaces utilizing the MCB1700 board by developing a game on the LCD. The game taught me the importance of scheduling tasks, as if conditions are checked at the wrong time or the display is not updated at the current time the overall user interface becomes poor and incorrect. All things considered, this study demonstrates how embedded system concepts can be used in real-world situations.

8. References

[1] Media Center Laboratory Manual, COE718 Embedded System Design, Toronto Metropolitan University, Toronto, ON, Canada. [Online]. Available: https://www.ee.torontomu.ca/~courses/coe718/labs/Media-Center.pdf

Appendix

Appendix A: Blinky.c (Main Menu)

```
/*----------------------------------------------------------------------------
* Name:    Blinky.c
* Purpose: LED Flasher and Graphic Demo
* Note(s):
*----------------------------------------------------------------------------
* This file is part of the uVision/ARM development tools.
* This software may only be used under the terms of a valid, current,
* end user licence from KEIL for a compatible version of KEIL software
* development tools. Nothing else gives you the right to use this software.
*
* This software is supplied "AS IS" without warranties of any kind.
*
* Copyright (c) 2008-2011 Keil - An ARM Company. All rights reserved.
*----------------------------------------------------------------------------*/

#include <LPC17xx.H>                    /* NXP LPC17xx definitions           */
#include "GLCD.h"
#include "KBD.h"
#include "photoGallery.h"

#define __FI        1                   /* Font index 16x24                  */
#define MENU_OPTIONS 3

// Menu items
const char *menu_items[MENU_OPTIONS] = {
  "  1. Photo Gallery",
  "  2. MP3 Player",
  "  3. Game"
};
int disp_val = 0;  // Current selection in the menu

// Function prototypes
void displayMenu(int selected);
void handleSelection(void);
void navigateMenu(void);
int MP3(void);      // Placeholder for MP3 player function
int game(void);     // Placeholder for game function
```

```
/*---------------------------------------------------------------------
 Display Menu Function
*---------------------------------------------------------------------*/
void displayMenu(int selected) {

    int i;

  // Display header
  GLCD_SetBackColor(White);
          GLCD_SetTextColor(Black);
          GLCD_DisplayString(0, 0, __FI, "--------------------");
          GLCD_DisplayString(1, 0, __FI, "   Welcome to the   ");
          GLCD_DisplayString(2, 0, __FI, "    Media Center!   ");
          GLCD_DisplayString(3, 0, __FI, "--------------------");
          GLCD_SetTextColor(DarkGrey);
          GLCD_DisplayString(8, 0, __FI, "--------------------");
          GLCD_DisplayString(9, 0, __FI, "          by: Maria");



  // Display menu items
  for (i = 0; i < MENU_OPTIONS; i++) {
    if (i == selected) {
      GLCD_SetBackColor(Blue);
      GLCD_SetTextColor(White);
    } else {
      GLCD_SetBackColor(White);
      GLCD_SetTextColor(Black);
    }
    GLCD_DisplayString(5 + i, 0, __FI, (unsigned char*)menu_items[i]);
  }
}

/*---------------------------------------------------------------------
 Handle Selection Function
*---------------------------------------------------------------------*/
void handleSelection(void) {
  GLCD_Clear(White);
  switch (disp_val) {
    case 0:
      photoGallery(); // Call to photo function
```

```
            break;
        case 1:
            MP3(); // Call to MP3 function
            break;
        case 2:
            game(); // Call to game function
            break;
    }
}

/*---------------------------------------------------------------------
 Navigate Menu Function
*---------------------------------------------------------------------*/
void navigateMenu(void) {
    int KBD_but = get_button();

    switch (KBD_but) {
        case KBD_SELECT:
            handleSelection();
            break;
        case KBD_UP:
            disp_val = (disp_val > 0) ? disp_val - 1 : MENU_OPTIONS - 1;
            displayMenu(disp_val);
            break;
        case KBD_DOWN:
            disp_val = (disp_val < MENU_OPTIONS - 1) ? disp_val + 1 : 0;
            displayMenu(disp_val);
            break;
    }
}

/*---------------------------------------------------------------------
 Main Functionality
*---------------------------------------------------------------------*/
int mainFunctionality(void) {
    KBD_Init();
    GLCD_Init();
    GLCD_Clear(White);
    displayMenu(disp_val);  // Display initial menu

    while (1) {
        navigateMenu();
```

```
  }
}

/*----------------------------------------------------------------------
 Main Program
*----------------------------------------------------------------------*/
int main(void) {
  mainFunctionality();
}
```

Appendix B: photoGallery.c

```
/*----------------------------------------------------------------------
* Name:    Blinky.c
* Purpose: LED Flasher and Graphic Demo
* Note(s):
*----------------------------------------------------------------------
* This file is part of the uVision/ARM development tools.
* This software may only be used under the terms of a valid, current,
* end user licence from KEIL for a compatible version of KEIL software
* development tools. Nothing else gives you the right to use this software.
*
* This software is supplied "AS IS" without warranties of any kind.
*
* Copyright (c) 2008-2011 Keil - An ARM Company. All rights reserved.
*----------------------------------------------------------------------*/

#include <LPC17xx.H>                    /* NXP LPC17xx definitions          */
#include "string.h"
#include "GLCD.h"
#include "LED.h"
#include "KBD.h"
#include "bulbasaur.c"
#include "eevee.c"
#include "piplup.c"
extern main(void);

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

/*----------------------------------------------------------------------
 Main Program
```

```
*-------------------------------------------------------------------------*/
int photoGallery (void) {
 int currentButtonState = 0;
 int lastButtonState = -1;
 int pic = 0;
  int KBD_but;
  unsigned char* images[] = {(unsigned char *)&bulbasaur_pixel_data,
                             (unsigned char *)&eevee_pixel_data,
                             (unsigned char *)&piplup_pixel_data
                            };

  int totalImages = sizeof(images) / sizeof(images[0]);

  KBD_Init();
 LED_Init ();
 GLCD_Init();

  for (;;) {
     currentButtonState = get_button();

     // Check if the button state has changed since the last loop iteration
     if (currentButtonState != lastButtonState) {
        switch (currentButtonState) {
           case KBD_RIGHT:
              pic = (pic + 1) % totalImages;  // right, wrap around if necessary
              break;
           case KBD_LEFT:
              pic = (pic - 1 + totalImages) % totalImages;  // left, wrap around if necessary
              break;
                    case KBD_SELECT:
                         GLCD_Clear (White);
                         main();
                         return 0;
                         break;
        }

                if(currentButtonState == KBD_UP | currentButtonState == KBD_DOWN){
                }
                else {
                   switch (pic){
                      case 0:
                         GLCD_Clear(White);  //bulb
```

```
                    GLCD_SetTextColor(Black);
                    GLCD_DisplayString(9, 0, __FI, " <   bulbasaur   >");
                    break;

                  case 1:
                    GLCD_Clear(White); //eve
                    GLCD_SetTextColor(Black);
                    GLCD_DisplayString(9, 0, __FI, " <    eevee     >");

                    break;
                  case 2:
                    GLCD_Clear(White); //pip
                    GLCD_SetTextColor(Black);
                    GLCD_DisplayString(9, 0, __FI, " <    piplup    >");
                    break;
                }

                GLCD_Bitmap(60, 30, 200, 160, images[pic]);

        lastButtonState = currentButtonState;
    }
  }
}
    }
```

Appendix C: usbdMain.c (MP3 Player)

```
/*----------------------------------------------------------------------------
* Name:    usbmain.c
* Purpose: USB Audio Class Demo
* Version: V1.20
*----------------------------------------------------------------------------
*      This software is supplied "AS IS" without any warranties, express,
*      implied or statutory, including but not limited to the implied
*      warranties of fitness for purpose, satisfactory quality and
*      noninfringement. Keil extends you a royalty-free right to reproduce
*      and distribute executable files created using this software for use
*      on NXP Semiconductors LPC microcontroller devices only. Nothing else
*      gives you the right to use this software.
*
* Copyright (c) 2009 Keil - An ARM Company. All rights reserved.
```

```
*-----------------------------------------------------------------------------*/

#include "LPC17xx.h"                    /* LPC17xx definitions */
#include "type.h"

#include "GLCD.h"
#include "KBD.h"
//#include "piplup.c"
#include "usb.h"

#include "usbcfg.h"
#include "usbhw.h"
#include "usbcore.h"
#include "usbaudio.h"


#include "music.c"

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

unsigned char* images[] = {(unsigned char *)&TMP_pixel_data};

int currentButtonState = 0;
int lastButtonState = -1;
//extern main(void);

extern  void SystemClockUpdate(void);
extern uint32_t SystemFrequency;
uint8_t  Mute;                          /* Mute State */
uint32_t Volume;                        /* Volume Level */
char text[10];
#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 Potenciometer 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;
 uint32_t cnt;
 int KBD_but;

 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 */
    sprintf(text, "0x%04X", Volume);
    GLCD_DisplayString(9, 10, __FI, (unsigned char*)text);
  }
  val = VUM >> 20;              /* Scale Accumulated Value */
  VUM = 0;                      /* Clear VUM */
  if (val > 7) val = 7;         /* Limit Value */
 }

 LPC_TIM0->IR = 1;              /* Clear Interrupt Flag */
 KBD_but = get_button();
 if(KBD_but == KBD_SELECT){
  NVIC_SystemReset();
 }
}
```

```
/*****************************************************************************
**   Main Function  main()
*****************************************************************************/

int MP3 (void){
 volatile uint32_t pclkdiv, pclk;
//  int currentButtonState = 0;
//  int lastButtonState = -1;
//
//  KBD_Init();
 GLCD_Init();
 GLCD_Clear  (Black);
 GLCD_SetBackColor(Black);
 GLCD_SetTextColor(White);
 GLCD_Bitmap(110, 70, 100, 100, images[0]);
 GLCD_DisplayString(1, 0, __FI, "     MP3 Player     ");
 GLCD_DisplayString(9, 0, __FI, "  Volume:       ");

  /* 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 */
 /********* The main Function is an endless loop **********/
while(1);


    // Check if the button state has changed since the last loop iteration


}

/*****************************************************************************
**                      End Of File
*****************************************************************************/
```

Appendix D: game.c

```
#include <LPC17xx.H>
#include "GLCD.h"
#include "KBD.h"
#include <stdlib.h>

// Bitmap Files
#include "2.c"
#include "4.c"
#include "8.c"
#include "16.c"
#include "32.c"
```

```
// Include other bitmaps (e.g., 8.c, 16.c, etc.)

#define GRID_SIZE 4      // Size of the 2048 grid (4x4)
#define CELL_SIZE 50     // Size of each cell on the LCD
#define DEBOUNCE_DELAY 200 // Debounce delay for joystick inputs
unsigned char* bitmap;
int board[GRID_SIZE][GRID_SIZE]; // Game board

// Timer Initialization for Random Seed
void initTimer() {
  LPC_SC->PCONP |= (1 << 1); // Power on Timer 0
  LPC_TIM0->TCR = 0x02;      // Reset Timer
  LPC_TIM0->TCR = 0x01;      // Enable Timer
}

void seedRandom() {
  srand(LPC_TIM0->TC); // Use Timer 0's counter as a seed
}

// Utility Functions
int randomRange(int min, int max) {
  return (rand() % (max - min + 1)) + min; // Random number between min and max
}

void delay(int ms) {
  int count = ms * 12000; // Approximate delay for LPC17xx at 12 MHz clock
  while (count--) {
    __NOP();
  }
}

int debounceJoystick() {
  int stableInput = get_button();
  delay(DEBOUNCE_DELAY);
  if (stableInput == get_button()) {
    return stableInput;
  }
  return 0; // No stable input
}

// Function to Get Bitmap for a Number
unsigned char* getBitmapForValue(int value) {
```

```
  switch (value) {
    case 2:
            bitmap=(unsigned char*)TWO_pixel_data;
            break;
    case 4:
            bitmap=(unsigned char*)FOUR_pixel_data;
            break;
        case 8:
            bitmap=(unsigned char*)EIGHT_pixel_data;
            break;
        case 16:
            bitmap=(unsigned char*)SIXTEEN_pixel_data;
            break;
        case 32:
            bitmap=(unsigned char*)THIRTYTWO_pixel_data;
            break;
    default: return NULL; // No bitmap for empty tiles
  }
}


// Function to Display Startup Message
void displayStartupMessage() {
  GLCD_Clear(White);
  GLCD_SetBackColor(White);
  GLCD_SetTextColor(Blue);
  GLCD_DisplayString(2, 2, 1, (unsigned char *)"Welcome to 2048!");
  GLCD_DisplayString(4, 2, 1, (unsigned char *)"To win, reach 32.");
  GLCD_DisplayString(6, 0, 1, (unsigned char *)"Use joystick to play");
  delay(7000); // Wait for 3 seconds
}

// Function to Display Win Message
void displayWinMessage() {
  GLCD_Clear(White);
  GLCD_SetBackColor(White);
  GLCD_SetTextColor(Blue);
  GLCD_DisplayString(4, 4, 1, (unsigned char *)"You Win!");
  GLCD_DisplayString(6, 2, 1, (unsigned char *)"Back to Menu!");
  delay(5000); // Wait for 3 seconds
}
```

```
// Game Board Initialization
void addRandomTile() {
  int emptyCells = 0;
  int emptyPositions[GRID_SIZE * GRID_SIZE][2];
  int i, j, randomIndex, row, col;

  // Collect empty positions
  for (i = 0; i < GRID_SIZE; i++) {
    for (j = 0; j < GRID_SIZE; j++) {
      if (board[i][j] == 0) {
        emptyPositions[emptyCells][0] = i;
        emptyPositions[emptyCells][1] = j;
        emptyCells++;
      }
    }
  }

  if (emptyCells == 0) return; // No empty cells

  // Select a random empty position
  randomIndex = randomRange(0, emptyCells - 1);
  row = emptyPositions[randomIndex][0];
  col = emptyPositions[randomIndex][1];

  // Place a 2 (90%) or 4 (10%)
  board[row][col] = (randomRange(0, 9) < 9) ? 2 : 4;
}

void initBoard() {
  int i, j;
  for (i = 0; i < GRID_SIZE; i++) {
    for (j = 0; j < GRID_SIZE; j++) {
      board[i][j] = 0; // Set all cells to 0
    }
  }
  // Add two initial tiles
  addRandomTile();
  addRandomTile();
}

void GLCD_FillRect(int x, int y, int width, int height) {
```

```
  int i,j;
  for (i = x; i < x + width; i++) {
    for (j = y; j < y + height; j++) {
      GLCD_PutPixel(i, j); // Draw a pixel at (i, j)
    }
  }
}


// Display the Game Board
void displayBoard() {
  int i, j;
  GLCD_Clear(White); // Clear the entire screen

  for (i = 0; i < GRID_SIZE; i++) {
    for (j = 0; j < GRID_SIZE; j++) {
      int x = j * CELL_SIZE + 20; // X position of tile
      int y = i * CELL_SIZE + 20; // Y position of tile

      // Draw tile background
      GLCD_SetTextColor(White);
      GLCD_FillRect(x, y, CELL_SIZE, CELL_SIZE); // Draw tile background

      // Get bitmap for the tile value
      bitmap = getBitmapForValue(board[i][j]);

      // Draw bitmap if it exists
      if (bitmap != NULL) {
        GLCD_Bitmap(x, y, CELL_SIZE, CELL_SIZE, bitmap);
      }

      // Draw border around tile
      GLCD_SetTextColor(Black);
      GLCD_FillRect(x, y, CELL_SIZE, 1);              // Top border
      GLCD_FillRect(x, y + CELL_SIZE - 1, CELL_SIZE, 1); // Bottom border
      GLCD_FillRect(x, y, 1, CELL_SIZE);              // Left border
      GLCD_FillRect(x + CELL_SIZE - 1, y, 1, CELL_SIZE); // Right border
    }
  }
}

int slideLeft() {
```

```
  int moved = 0;
    int i,j;
  for (i = 0; i < GRID_SIZE; i++) {
    for (j = 1; j < GRID_SIZE; j++) {
      if (board[i][j] != 0) {
        int k = j;
        while (k > 0 && board[i][k - 1] == 0) {
          board[i][k - 1] = board[i][k];
          board[i][k] = 0;
          k--;
          moved = 1;
        }
      }
    }
  }
  return moved;
}

int slideRight() {
  int moved = 0;
    int i,j;
  for (i = 0; i < GRID_SIZE; i++) {
    for (j = GRID_SIZE - 2; j >= 0; j--) {
      if (board[i][j] != 0) {
        int k = j;
        while (k < GRID_SIZE - 1 && board[i][k + 1] == 0) {
          board[i][k + 1] = board[i][k];
          board[i][k] = 0;
          k++;
          moved = 1;
        }
      }
    }
  }
  return moved;
}

int slideUp() {
  int moved = 0;
    int j,i;
  for (j = 0; j < GRID_SIZE; j++) {
    for (i = 1; i < GRID_SIZE; i++) {
```

```
        if (board[i][j] != 0) {
          int k = i;
          while (k > 0 && board[k - 1][j] == 0) {
            board[k - 1][j] = board[k][j];
            board[k][j] = 0;
            k--;
            moved = 1;
          }
        }
      }
    }
    return moved;
}

int slideDown() {
    int moved = 0;
      int i,j;
    for (j = 0; j < GRID_SIZE; j++) {
      for (i = GRID_SIZE - 2; i >= 0; i--) {
        if (board[i][j] != 0) {
          int k = i;
          while (k < GRID_SIZE - 1 && board[k + 1][j] == 0) {
            board[k + 1][j] = board[k][j];
            board[k][j] = 0;
            k++;
            moved = 1;
          }
        }
      }
    }
    return moved;
}

void mergeLeft() {
    int i,j;
    for (i = 0; i < GRID_SIZE; i++) {
      for (j = 0; j < GRID_SIZE - 1; j++) {
        if (board[i][j] != 0 && board[i][j] == board[i][j + 1]) {
          board[i][j] *= 2;
          board[i][j + 1] = 0;
        }
      }
```

```
    }
  }

  void mergeRight() {
    int i, j;
    for (i = 0; i < GRID_SIZE; i++) {
      for (j = GRID_SIZE - 1; j > 0; j--) {
        if (board[i][j] != 0 && board[i][j] == board[i][j - 1]) {
          board[i][j] *= 2;
          board[i][j - 1] = 0;
        }
      }
    }
  }

  void mergeUp() {
    int i, j;
    for (j = 0; j < GRID_SIZE; j++) {
      for (i = 0; i < GRID_SIZE - 1; i++) {
        if (board[i][j] != 0 && board[i][j] == board[i + 1][j]) {
          board[i][j] *= 2;
          board[i + 1][j] = 0;
        }
      }
    }
  }

  void mergeDown() {
    int i, j;
    for (j = 0; j < GRID_SIZE; j++) {
      for (i = GRID_SIZE - 1; i > 0; i--) {
        if (board[i][j] != 0 && board[i][j] == board[i - 1][j]) {
          board[i][j] *= 2;
          board[i - 1][j] = 0;
        }
      }
    }
  }

  int checkWinCondition() {
    int i, j;
    for (i = 0; i < GRID_SIZE; i++) {
```

```
      for (j = 0; j < GRID_SIZE; j++) {
        if (board[i][j] == 32) {
          return 1; // Win condition met
        }
      }
    }
    return 0; // No win condition met
}


// Game Loop
int game(void) {
    GLCD_Init();
    KBD_Init();
    initTimer();   // Initialize Timer 0
    seedRandom();  // Seed random number generator
      displayStartupMessage();
    initBoard();
    displayBoard();

    while (1) {
      int buttonPress = debounceJoystick();
      int moved = 0;

          if (buttonPress == KBD_SELECT) {
        NVIC_SystemReset(); // Reset the microcontroller
                return 0;
      }

      if (buttonPress == KBD_UP) {
        moved = slideUp();
        mergeUp();
        moved |= slideUp();
      } else if (buttonPress == KBD_DOWN) {
        moved = slideDown();
        mergeDown();
        moved |= slideDown();
      } else if (buttonPress == KBD_LEFT) {
        moved = slideLeft();
        mergeLeft();
        moved |= slideLeft();
      } else if (buttonPress == KBD_RIGHT) {
```

```
            moved = slideRight();
            mergeRight();
            moved |= slideRight();
        }

        if (moved) {
            addRandomTile();
            displayBoard();
        }

            if (checkWinCondition()) {
            displayWinMessage();
            NVIC_SystemReset();
            return 0;
        }

    }

    return 0;
}
```