COE 718 Final Project Report
MCB1700 Board Media Center

*By Najeeb Almasri - Fall 2024

I. Abstract

This report presents the design and implementation of an advanced embedded system project using the LPC1768 microcontroller, integrating concepts from prior lab work. The project focuses on developing an interactive system featuring audio playback and enhanced graphical user interface (GUI) capabilities.

Key features include real-time audio processing, dynamic volume control using ADC, and an intuitive GLCD-based interface with smooth animations and visual feedback. The methodology adopted involves modular design, leveraging timers, interrupts, and user input handling to ensure system efficiency and responsiveness.

Experimental results demonstrate the successful integration of hardware and software components, achieving low-latency performance and user-friendly interaction. This project serves as a comprehensive showcase of embedded systems principles, highlighting the culmination of iterative learning from previous labs.

II. Introduction

The Media Center project for the MCB1700 development board aims to create an interactive multimedia application that integrates a photo gallery, MP3 player, and game center. This project was designed to showcase the embedded systems programming skills acquired throughout the course, such as peripheral interfacing, interrupt handling, and multitasking using the Keil RTX. The MCB1700 board, equipped with the LPC1768 microcontroller, serves as an ideal platform due to its rich set of peripherals including an LCD screen, joystick, potentiometer, and USB interface.

The application features a main menu interface displayed on the LCD, allowing users to navigate between different modules using the joystick. Each module presents a unique challenge: the photo gallery requires image conversion and rendering, the MP3 player involves streaming audio from a connected PC via USB, and the game center implements an interactive game (Snake) using the joystick for input. These design choices reflect a balance between functionality, user interactivity, and effective use of the board's capabilities.

The implementation process involved breaking down the project into smaller modules, each responsible for a specific task. The main menu was prioritized to establish the primary interface and navigation flow. Once completed, the Snake game was developed as a proof of concept for joystick-based interaction and LCD graphics. The ongoing work includes integrating the MP3 player and photo gallery, focusing on USB audio streaming and efficient image handling. The expected outcome is a fully functional media center application that demonstrates robust design, modular implementation, and seamless user experience.

III. Past Work/Review

This project builds upon the foundational concepts and methodologies explored in previous labs:

Lab 1 Introduction to Embedded Systems Development: Focus: Basic configuration of the LPC1768 board, understanding peripherals, and using the GLCD for simple outputs. Relevance: This lab provided critical insights into initializing and interfacing with the GLCD, a skill directly applied to the visual elements of this project.

Lab 2 Timer and Interrupts: Focus: Configuration and use of timers and interrupts to handle periodic tasks. Relevance: Timers and interrupts from this lab were leveraged to manage periodic updates for animations and audio outputs.

Lab 3a and 3b Analog-to-Digital Conversion (ADC): Focus: Using the ADC to read potentiometer values and manipulating these for control purposes. Relevance: ADC knowledge was used to dynamically control the volume in the audio module, allowing a real-time interaction with hardware.

Lab 4 Advanced GLCD Features: Focus: More complex graphics rendering, including animations and multiple visual layers. Relevance: This lab directly influenced the updated designs and animations displayed on the GLCD, enhancing the user interface aesthetics.

By integrating these foundational skills, this project showcases the culmination of incremental learning and practical applications from prior labs.

IV. Methodology

To design the project, a structured top-down approach was employed:

Requirement Analysis: Identification of essential functionalities, including audio playback, interactive GLCD graphics, and responsive user inputs.

System Design: Division of tasks into modules, such as GLCD Display, Audio Control, and User Input Handling. Establishment of data flows between these modules.

Incremental Development: Testing and debugging each module independently before integrating them.

Iterative Refinement: Continuous improvements based on experimental outcomes, user experience testing, and functionality requirements.

This methodology ensured a robust, modular, and scalable system.

V. System Desgin

The system design of the Media Center application involves both hardware and software components, each playing a crucial role in delivering a smooth, interactive experience. The following sections provide a detailed overview of the hardware setup, software architecture, and flow diagrams for the implemented modules.

Hardware Components:

1. MCB1700 Development Board (LPC1768 Microcontroller): The LPC1768 microcontroller serves as the main processing unit, providing the computational power needed for the application. It includes essential peripherals such as the LCD display, joystick, potentiometer, and USB interface.

2. LCD Display: The LCD is used to display the main menu, images, and game graphics. The application leverages the GLCD driver to interact with the display, allowing for graphical updates and animations.

3.Joystick: The joystick provides user input for navigating the main menu and controlling the Snake game. It supports five directions: up, down, left, right, and center click (select).

4. Potentiometer: The potentiometer is used for volume control in the MP3 player module. It adjusts the output audio level based on its rotation.

5. USB Interface: The USB interface enables audio streaming from a connected PC. It is configured as a USB audio device using the CMSIS USB driver.

Software Components The software architecture is modular, with separate modules for the main menu, photo gallery, MP3 player, and game center. The application uses CMSIS libraries for hardware abstraction and Keil RTX for multitasking support.

1. Main Menu Module: The main menu serves as the entry point of the application, displaying options for the user to select. It is implemented using a state machine that tracks the current selection and updates the display accordingly.

2. Photo Gallery Module: The photo gallery module displays bitmap images stored as C arrays on the LCD. Images are converted using GIMP and included in the project as resources. Users can browse through images using the joystick, with left/right movements for navigation.

3. MP3 Player Module: The MP3 player module streams audio from a PC via USB and plays it through the on-board speaker. The volume is adjusted using the potentiometer. The module displays a splash screen on the LCD during playback and handles joystick input for exiting the player.

4. Game Center Module (Snake Game): The Snake game uses the joystick for movement control and updates the LCD screen with the game state. Collision detection is implemented to end the game if the snake hits the boundary or itself. The game tracks and displays the score, returning to the main menu when the game ends.

Flow Chart Diagram - Media Center:

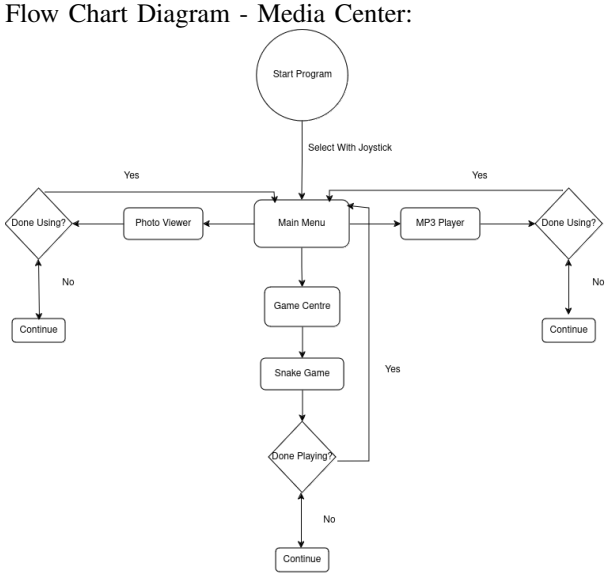

Figure 1: Flow Chart Diagram of the Media Center

Table: Project Plan

Week	Tasks	Description	Deliverables
Week **1** (Completed)	Project Setup & Main Menu Implementation	Implement the LCD menu interface	Functional main menu
Week **2** (Completed)	Game Development (Snake Game)	Develop the chosen game	Functional game
Week **3**	Audio Player Development	Implement the USB-based audio player with volume control	Functional audio player
Week **4**	Photo Gallery Implementation and Final Testing	Develop the photo viewer. Integrate all components, and perform final system tests	Functional photo gallery. Fully functional system, all components tested

This modular approach ensured that each component is isolated, making the system easier to debug and extend!

VI. Experimental Results

The project was evaluated based on functionality, performance, and user experience:

Functionality: The audio playback module successfully processed and played audio data. Dynamic volume adjustment worked seamlessly using the potentiometer. The GLCD displayed animations and interactive feedback without latency.

Performance: Response Time: The system exhibited negligible delays in input recognition and output rendering. Resource Usage: Efficient use of timers and interrupts ensured smooth operation even under load.

Issues and Resolutions: Initial flickering in GLCD animations was resolved by optimizing the rendering logic. Audio playback glitches were mitigated by fine-tuning the timer intervals.

The image below illustrates the project implementation on the lab board, showcasing its functionality and integration.

VII. Conclusion

This lab report documents the development of a Media Center on the MCB1700 board as part of the final project for COE718: Embedded System Design.

The Media Center integrates three key functionalities: a photo gallery, an mp3 player, and a game center with joystick navigation.

The project demonstrates core concepts of embedded programming, peripheral interfacing, and multimedia processing. The software setup involved using the Keil uVision IDE to integrate drivers for the LCD, joystick, USB audio streaming, and onboard speaker.

Images for the photo gallery were converted from .bmp to C arrays using GIMP, while the mp3 player streamed audio from a PC to the MCB1700 via USB. The hardware setup centered on using the LCD for display, the joystick for navigation, the potentiometer for volume control, and the onboard speaker for audio playback.

Key tasks included building a photo gallery, implementing an mp3 player for USB audio streaming, and designing a game of Snake controlled by the joystick. The project was successful, with each feature functioning as expected: images displayed clearly, audio streamed smoothly, and the game ran without lag.

This project demonstrates the design and implementation of an embedded system, integrating multimedia, user input, and hardware interfacing.

VIII. References

1. ARM Ltd., “LPC1768 User Manual,” [Online]. Available: https://www.nxp.com/docs/en/user-guide/UM10360.pdf.

2. Keil, “CMSIS-RTOS2 User Guide,” [Online]. Available: https://www.keil.com/pack/doc/CMSIS/RTOS2/html/index.html.

3. GIMP Development Team, “Exporting C Source Images,” GIMP Documentation, [Online]. Available: https://www.gimp.org.

4. Keil, “USBAudio Example Project,” COE718 Project Directory, [Course Material], U:718, 2024.

5. NXP Semiconductors, “USB Audio Device Class,” Application Note AN11697, [PDF document]. Available: https://www.nxp.com/docs/en/application-note/AN11697.pdf.

IX Appendix

The following section provides the code for each of the main C files:

// Blinky.c:

```
// LPC17xx definitions
#include <LPC17xx.H>
#include "string.h"
#include "GLCD.h"
#include "LED.h"
#include "KBD.h"
#include "PHOTO_VIEWER.h"
#include "usbdmain.h"
#include "game_on.h"
#include "snake.h"

// Define menu indices for readability
#define PHOTO_VIEWER 1
#define AUDIO_PLAYER 2
#define GAME_CENTRE 3

// Define sub-menu indices for Game Centre
#define MEMORY_GAME 1
#define SNAKE_GAME 2

// Declaration of audio main function
extern int audio_main(void);

// Function Prototypes
void resetMainMenu(void);
void executeTask(int selector);
void displayGameCentreMenu(int selector);
void executeGameCentreTask(int selector);

// Main Function
int main(void)
{
	int selector = 0; // Current menu item
	int joystick_val = 0;
	int joystick_prev = 0;

	// Initialize peripherals
	KBD_Init();
	LED_Init();
	GLCD_Init();
	GLCD_Clear(Black);
```

```
SysTick_Config(SystemCoreClock / 100);

// Display initial menu
resetMainMenu();

// Infinite loop for menu navigation
for (;;) {
joystick_val = get_button(); // Read joystick input

// Handle joystick input only when it changes
if (joystick_val != joystick_prev)
        {
if (joystick_val == KBD_DOWN)
        {
selector = (selector % GAME_CENTRE) + 1; // Increment selector
} else if (joystick_val == KBD_UP)
        {
selector = (selector == 1) ? GAME_CENTRE : selector - 1; // Decrement selector
} else if (joystick_val == KBD_RIGHT)
        {
if (selector == GAME_CENTRE)
                {
        int game_selector = 1; // Default to the first game in Game Centre
        displayGameCentreMenu(game_selector);
        for (;;)
                {
        joystick_val = get_button(); // Handle Game Centre navigation
        if (joystick_val != joystick_prev)
                        {
        if (joystick_val == KBD_DOWN)
                        {
        game_selector = (game_selector % SNAKE_GAME) + 1; // Increment selector
        } else if (joystick_val == KBD_UP)
                        {
        game_selector = (game_selector == 1) ? SNAKE_GAME : game_selector - 1; // Decrement
selector
        } else if (joystick_val == KBD_RIGHT)
                        {
        executeGameCentreTask(game_selector); // Execute selected game
        break; // Return to main menu
        } else if (joystick_val == KBD_LEFT)
                        {
        resetMainMenu(); // Return to main menu
        break;
        }
        displayGameCentreMenu(game_selector); // Update Game Centre menu
        joystick_prev = joystick_val;
        }
        }
```

```
        } else {
                executeTask(selector); // Execute selected task
        }
        }
        joystick_prev = joystick_val;
        }

        // Update menu display based on selector
        switch (selector) {
        case PHOTO_VIEWER:
                {
        GLCD_SetBackColor(Black);
        GLCD_SetTextColor(Red);  // Highlighted color
        GLCD_DisplayString(4, 2, 1, ">> Photo Viewer");
        GLCD_SetTextColor(Blue);
        GLCD_DisplayString(5, 2, 1, "[*] Audio Player");
        GLCD_DisplayString(6, 2, 1, "[*] Game Centre");
        break;
        }
        case AUDIO_PLAYER:
                {
        GLCD_SetBackColor(Black);
        GLCD_SetTextColor(Blue);
        GLCD_DisplayString(4, 2, 1, "[*] Photo Viewer");
        GLCD_SetTextColor(Red);  // Highlighted color
        GLCD_DisplayString(5, 2, 1, ">> Audio Player");
        GLCD_SetTextColor(Blue);
        GLCD_DisplayString(6, 2, 1, "[*] Game Centre");
        break;
        }
        case GAME_CENTRE:
                {
        GLCD_SetBackColor(Black);
        GLCD_SetTextColor(Blue);
        GLCD_DisplayString(4, 2, 1, "[*] Photo Viewer");
        GLCD_DisplayString(5, 2, 1, "[*] Audio Player");
        GLCD_SetTextColor(Red);  // Highlighted color
        GLCD_DisplayString(6, 2, 1, ">> Game Centre");
        break;
        }
        default:
        resetMainMenu();
        break;
        }
        }
}

// Reset menu to initial state
void resetMainMenu(void)
```

```
{
	int i; // Declare loop variable outside the loop

	// Clear screen
	GLCD_Clear(Black);

	// Draw borders
	GLCD_SetTextColor(White);
	for (i = 0; i < 320; i++)
	{
	GLCD_PutPixel(i, 0);	// Top border
	GLCD_PutPixel(i, 239);  // Bottom border
	}
	for (i = 0; i < 240; i++)
	{
	GLCD_PutPixel(0, i);	// Left border
	GLCD_PutPixel(319, i);  // Right border
	}

	// Display header
	GLCD_SetBackColor(Black);
	GLCD_SetTextColor(Yellow);
	GLCD_DisplayString(0, 5, 1, "Final Project");
	GLCD_SetTextColor(Green);
	GLCD_DisplayString(1, 5, 1, "Media Center");

	// Display menu items
	GLCD_SetTextColor(Blue);
	GLCD_DisplayString(4, 2, 1, "[*] Photo Viewer");
	GLCD_DisplayString(5, 2, 1, "[*] Audio Player");
	GLCD_DisplayString(6, 2, 1, "[*] Game Centre");
}

// Execute the selected task
void executeTask(int selector)
{
	GLCD_Clear(White); // Clear screen for task execution
	switch (selector) {
	case PHOTO_VIEWER:
	photo_viewer(1); // Start photo viewer
	break;
	case AUDIO_PLAYER:
	audio_main(); // Start audio player
	break;
	default:
	break;
	}
	resetMainMenu(); // Return to menu after task
}
```

```
// Display the Game Centre menu
void displayGameCentreMenu(int selector)
{
	GLCD_Clear(Black);
	GLCD_SetBackColor(Black);
	GLCD_SetTextColor(Blue);
	GLCD_DisplayString(0, 4, 1, "Game Centre");
	GLCD_DisplayString(4, 2, 1, "Memory Game");
	GLCD_DisplayString(5, 2, 1, "Snake Game");

	// Highlight the selected game
	switch (selector)
	{
	case MEMORY_GAME:
	GLCD_SetBackColor(Red);
	GLCD_SetTextColor(White);
	GLCD_DisplayString(4, 2, 1, "Memory Game");
	GLCD_SetBackColor(Black);
	GLCD_SetTextColor(Blue);
	GLCD_DisplayString(5, 2, 1, "Snake Game");
	break;
	case SNAKE_GAME:
	GLCD_DisplayString(4, 2, 1, "Memory Game");
	GLCD_SetBackColor(Red);
	GLCD_SetTextColor(White);
	GLCD_DisplayString(5, 2, 1, "Snake Game");
	GLCD_SetBackColor(Black);
	GLCD_SetTextColor(Blue);
	break;
	}
}

// Execute the selected game in Game Centre
void executeGameCentreTask(int selector)
{
	GLCD_Clear(White); // Clear screen for task execution
	switch (selector)
	{
	case MEMORY_GAME:
	game_on(); // Start memory game
	break;
	case SNAKE_GAME:
	snakegame(); // Start snake game
	break;
	default:
	break;
	}
	resetMainMenu(); // Return to main menu after task
```

```
}
```

// PhotoViewer.c:

```
// LPC17xx definitions
#include <LPC17xx.H>
#include "PHOTO_VIEWER.h"
#include "GLCD.h"
#include "LED.h"
#include "KBD.h"
#include "blue eyes.c"
#include "re.c"
#include "potofgreed.c"

extern unsigned char be[];
extern unsigned char re[];
extern unsigned char DarkMagician[];
int counterr;

// Main Function
void display_image(int which)
{
	int delay = 0;
	if (which==0)
	{
	GLCD_Clear(Black);
	GLCD_Bitmap(80,60,110,160,(unsigned char *)GIMP_BLUEEYES_pixel_data);
	}
	else if (which==1)
	{
	GLCD_Clear(Black);
	GLCD_Bitmap(80,60,111,160,(unsigned char *)GIMP_RE_pixel_data);
	}
	else if (which==2)
	{
	GLCD_Clear(Black);
	GLCD_Bitmap(80,60,109,160,(unsigned char *)GIMP_POTOFGREED_pixel_data);
	}
}

// Photo Viewer
void photo_viewer (int mode)
{
	int zoom=0;
	int pic    =  0, DELAY = 0;
	int timer   =  0;
	unsigned char *picture_ptr =0;
	int joystick_prev = get_button();
```

```
	int joystick_val = get_button();
	counterr=0;
	display_image(pic);
	while (timer <1)
	{
	joystick_val = get_button();
	if (joystick_val != joystick_prev)
	{
	if (joystick_val == KBD_RIGHT)
		{
		pic = pic+1;
		pic = pic%3;
		display_image(pic);
		zoom = 0;
		}
	else if (joystick_val ==KBD_LEFT)
		{
		pic = pic-1;
		if (pic < 0)
		pic = 2;
		display_image(pic);
		zoom = 0;
		}
	else if (joystick_val ==KBD_SELECT)
	{
		timer ++;
	}
	joystick_prev = joystick_val;
	counterr =0;
	}
	if (counterr >= 50)
	{
	pic = pic+1;
	pic = pic%3;
	display_image(pic);
	counterr =0;
	zoom = 0;

	}
	}
	timer = 0;
	GLCD_Clear(White);
	GLCD_DisplayString (0, 0, 1, "COE 718 Project Demo");
	GLCD_DisplayString (1, 6, 1, "MAIN MENU");
}
```

// Audio.c:

```
/*----------------------------------------------------------------------------
 * 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 "usb.h"
#include "usbcfg.h"
#include "usbhw.h"
#include "usbcore.h"
#include "usbaudio.h"

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 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;

  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 */
}



/*****************************************************************************
**   Main Function  main()
*****************************************************************************/
int main (void)
{
 volatile uint32_t pclkdiv, pclk;

 /* 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 );
}

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

// Snake.c:

```
#include "stdio.h"
#include "stdlib.h"
#include "LPC17xx.h"
#include "KBD.h"
#include "GLCD.h"
#include "LED.h"

#define DELAY_2N 20
#define GRID_WIDTH 20
#define GRID_HEIGHT 10
#define MAX_SNAKE_SIZE 100

// Variables
int xpos, ypos, size, direction, prev_direction, score, speed, collision;
int snake[MAX_SNAKE_SIZE][2];
int food_x, food_y;
int border_enabled = 1;

// Function Prototypes
```

```
void delay(int count);
void initialize_game(void);
void generate_food(void);
void draw_snake(void);
void clear_snake(void);
void update_snake_position(void);
void check_collision(void);
void render_grid(void);
void game_over_screen(void);
void update_score(void);

// Delay Function
void delay(int count)
{
	count <<= DELAY_2N;
	while (count--);
}

// Initialize Game
void initialize_game(void)
{
	GLCD_Clear(Black);
	xpos = GRID_HEIGHT / 2;
	ypos = GRID_WIDTH / 2;
	size = 3;
	direction = 0; // Right
	prev_direction = -1;
	score = 0;
	speed = 15;
	collision = 0;
	generate_food();
	render_grid();
	draw_snake();
	update_score();
}

// Generate Food at Random Position
void generate_food(void)
{
	int valid = 0, i;
	while (!valid)
	{
	food_x = rand() % GRID_HEIGHT;
	food_y = rand() % GRID_WIDTH;
	valid = 1;
	for (i = 0; i < size; i++) {
	if (snake[i][0] == food_x && snake[i][1] == food_y)
		{
	valid = 0;
```

```
            break;
            }
            }
            }
            GLCD_SetTextColor(Red);
            GLCD_DisplayChar(food_x, food_y, 1, 0x94);
}

// Render Grid Borders
void render_grid(void)
{
            int x, y;
            if (border_enabled)
            {
            GLCD_SetTextColor(White);
            for (x = 0; x < GRID_HEIGHT; x++)
            {
            GLCD_DisplayChar(x, 0, 1, '|');
            GLCD_DisplayChar(x, GRID_WIDTH - 1, 1, '|');
            }
            for (y = 0; y < GRID_WIDTH; y++)
            {
            GLCD_DisplayChar(0, y, 1, '-');
            GLCD_DisplayChar(GRID_HEIGHT - 1, y, 1, '-');
            }
            }
}

// Draw Snake
void draw_snake(void)
            {
            int i;
            for (i = 0; i < size; i++)
            {
            if (i == 0)
            {
            GLCD_SetTextColor(Green);
            GLCD_DisplayChar(snake[i][0], snake[i][1], 1, 0x88);
            } else
            {
            GLCD_SetTextColor(Blue);
            GLCD_DisplayChar(snake[i][0], snake[i][1], 1, 0x87);
            }
            }
}

// Clear Snake
void clear_snake(void)
            {
```

```
        int i;
        for (i = 0; i < size; i++)
        {
        GLCD_SetTextColor(Black);
        GLCD_DisplayChar(snake[i][0], snake[i][1], 1, ' ');
        }
}

// Update Snake Position
void update_snake_position(void)
        {
        int i;
        clear_snake();
        for (i = size - 1; i > 0; i--)
        {
        snake[i][0] = snake[i - 1][0];
        snake[i][1] = snake[i - 1][1];
        }
        if (direction == 0) ypos++;
        else if (direction == 1) ypos--;
        else if (direction == 2) xpos++;
        else if (direction == 3) xpos--;

        snake[0][0] = xpos;
        snake[0][1] = ypos;
        draw_snake();
}

// Check for Collisions
void check_collision(void)
        {
        int i;
        // Collision with borders
        if (border_enabled) {
        if (xpos < 1 || xpos >= GRID_HEIGHT - 1 || ypos < 1 || ypos >= GRID_WIDTH - 1)
        {
        collision = 1;
        }
        }

        // Collision with self
        for (i = 1; i < size; i++)
        {
        if (snake[0][0] == snake[i][0] && snake[0][1] == snake[i][1])
        {
        collision = 1;
        }
        }
```

```
        // Eating food
        if (snake[0][0] == food_x && snake[0][1] == food_y)
        {
        size++;
        score += 10;
        if (speed > 5) speed--;
        generate_food();
        update_score();
        }
}

// Update Score Display
void update_score(void)
{
        char str[20];
        GLCD_SetTextColor(Yellow);
        GLCD_DisplayString(0, 0, 1, "Score: ");
        sprintf(str, "%d", score);
        GLCD_DisplayString(0, 7, 1, (unsigned char *)str);
}

// Display Game Over Screen
void game_over_screen(void)
{
        char str[20];
        GLCD_Clear(Red);
        GLCD_SetTextColor(White);
        GLCD_DisplayString(5, 5, 1, "GAME OVER");
        sprintf(str, "Score: %d", score);
        GLCD_DisplayString(6, 5, 1, (unsigned char *)str);
}

// Main Snake Game Function
int snakegame(void) {
        initialize_game();

        while (!collision)
        {
        int joystick_val = get_button();
        if (joystick_val != prev_direction)
        {
        if (joystick_val == KBD_UP && direction != 2) direction = 3;
        else if (joystick_val == KBD_DOWN && direction != 3) direction = 2;
        else if (joystick_val == KBD_LEFT && direction != 0) direction = 1;
        else if (joystick_val == KBD_RIGHT && direction != 1) direction = 0;
        prev_direction = joystick_val;
        }

        update_snake_position();
```

```
        check_collision();
        delay(speed);
        }

        game_over_screen();
        return 0;
}
```