SCHOOL OF ELECTRONICS AND ELECTRICAL ENGINEERING

ELEC 5620 – MINIPROJECT

Embedded Microprocessor System Design

INTERACTIVE GAMING ENVIRONMENT

ON

DE1-SOC DEVELOPMENT BOARD

BY

NNADIKA CHIMA DANIEL

201077064

AND

OGUNTOLA AZEEM DAMILOLA

201162945

MAY 2018

ABSTRACT

In this paper, the programming of the DE1-SoC development board for an interactive gaming environment was detailed. The game demonstrated the versatility of the SoC device, utilizing various hardware peripherals. The game was built in Altera SoC Embedded Development Studio employing various coding and debugging techniques. Its functionality was verified on the DE1-SoC device.

1. INTRODUCTION

The De1-SoC development board is a hard processor system consisting of a Cyclone V SoC, an FPGA, and various hardware peripherals. The aim of this project is to program the board to demonstrate the versatility of the Cyclone V SoC device. This report describes the development of an Interactive Gaming Environment on the DE1-SoC. It also details the testing and verification of the code. The gaming environment was programmed in C language and consists of two games; Hero game and Snake game, and a menu from which the user can select a game.

The hero game is a one player shooter game designed to test the player's speed and reaction time. The player chooses a game avatar and starts the game with five live bars. The avatar is positioned at the bottom of the screen and can be moved in the horizontal axis. The game is played by shooting beams towards falling enemy ships. Each hit is recorded as a kill and adds to the player score. If the enemy reaches the bottom of the screen, the player losses a live bar. The game is over when the player runs out of life bars.

The snake game is a single player game in which the player manoeuvres a constantly moving line within an environment. The game is played by directing the line to a food item on the screen. When the line collides with a food item, it increases in length and a new food item is spawned at a random location on the screen. The game is over when the line collides with an obstacle or itself.

Features of the interactive gaming environment includes; saving of high scores to the SD card, playing of audio in the game background, display of images, shapes and text on the LCD, display of text on the seven-segment displays, and the use of buttons for inputs. Techniques used in the design include; peripheral interrupts, task scheduling using timer polling, non-blocking polling of inputs, and use of structs.

The rest of this report is organized as follows. Section two describes the key features and techniques involved its design. Section three describes the game code structure. In section four, the process of testing and verification of the code is described, and the report is concluded in section five.

2. CODE ARCHITECTURE

2.1. Timer Polling

The gaming environment makes use of the HPS Private Timer for task scheduling. For this purpose, the private timer is configured to be automatically reloaded when the counter reaches zero and the timer is enabled at the start of the game. The timer is polled by using if statements to compare the difference between the current timer value and the last value recorded against the desired period.

Task scheduling was achieved by creating task execution functions (identified by the suffix **_execute**) which take in as parameter, the address of other functions to be run. The execution functions poll the private timer in a non-blocking manner and execute the addressed function at the specified time interval.

2.2. Object Oriented programming

For cases where multiple objects were required to operate and be displayed on the LCD screen at the same time, an object-oriented approach was utilized in the program. This was achieved using C **struct** (structures). The structures were created using **typedef** and assigned inheritable properties.

Enemy ships and shooter beams in the hero game are generated using structures with properties that specify their x and y coordinates on the LCD. The structures were instantiated as arrays which allowed for efficient handling of multiple enemy ships and shooter beams.

2.3. HPS_IRQ Interrupts

Two peripheral interrupts were employed in the interactive gaming environment. These are the HPS_TIMER0_IRQ interrupt and the KEYS_IRQ interrupt. Control of the interrupts is achieved using the **HPS_IRQ** driver which provides functions for enabling and disabling interrupts.

The HPS_TIMER0_IRQ interrupt is used to run audio functions for both the hero game and the snake game. It is employed in **Hero_launchScene()** and **Hero_selectScene()** to run the audio functions **Hero_launchAudio()** and **Hero_backAudio()** respectively. It is also used in **Snake_Enviro()** to run the **Snake_backAudio()** function.

The HPS_TIMER0 has a clock speed of 100MHz. A driver, **HPS_Timers** was created to interface with the timer and configure it for interrupt. It includes the functions **Timer0_disable()** and **Timer0_enable()** for disabling and enabling the timer respectively. It also includes **Timer0_load()** for loading the timer with a desired value and **Timer0_clearInterrupt()** for clearing the timer interrupt flag.

To use the timer interrupt, the HPS_TIMER0 is first configured by disabling it using **Timer0_disable()**, loading it with a value of 2083 using **Timer0_load()** to give an interval of 20.83 µs (48 kHz), and then enabling it using **Timer0_enable()**. The IRQ interrupt is then enabled by calling the **HPS_IRQ_registerHandler()** function and passing as parameters, the timer ID and a pointer to the desired audio function to be run. The timer interrupt flag is cleared in the audio function using **Timer0_clearInterrupt()**. The interrupt is disabled by calling the **HPS_IRQ_unregisterHandler()** function and passing the timer ID as parameter.

The KEYS_IRQ interrupt is used to read the push button inputs on key release. It is used in the hero game function **Hero_selectCursor()** to run the function **Hero_selectNav()** which allows the user to navigate and select a character using the buttons. It is also employed in the game menu function **Game_selectCursor()** to run the function **Game_selectNav()** which allows the user to select a game using the buttons.

A driver, **IO_Peripherals** was created to interface with the push buttons and configure it for interrupt. It includes an initialisation function which enables interrupts for all the buttons, and the function **IO_getKeyPressID()** which returns the ID of the key that was pressed and clears the KEYS interrupt register. The KEYS_IRQ interrupt is enabled and a pointer to the desired function, attached using **HPS_IRQ_registerHandler()**. The interrupt is disabled using **HPS_IRQ_unregisterHandler().**

2.4. Audio Functionality

Audio effects were added to both the hero game and the snake game. The WM8731 audio CODEC is used to output audio through the line-out port of the De1-SoC, to speakers or headphones. The CODEC is interfaced with using the **WM8731_AudioEngine** driver which

is a modified version of the **DE1SoC_WM8731** driver, extended to include additional functions. The key functions are; **Audio_writeSpace()** which checks to see if there is space in the left and right FIFOs, **Audio_writeToLeft()** and **Audio_writeToRight()** which copies audio samples to the left and right DACs respectively, and **Audio_outVolume()** which is used to set the output volume between levels from 0 to 10.

MP3 versions of the audio files (with sampling frequency of 48 kHz) were decompressed into a two-column array (stereo-left and right) of the audio samples using a MATLAB function (Appendix C) which generates a C source and header file of the array.

The hero game audio effects are handled by two functions **Hero_launchAudio()** and **Hero_backAudio()** which are attached in **Hero_launchScene()** and **Hero_selectScene()** respectively to the HPS_TIMER0_IRQ interrupt and set to run at an interval of 20.83 µs or a rate of 48kHz. The snake game audio effects are handled by the function **Snake_backAudio** located in the *Audio.c* file of the snake game. This function is also attached to the HPS_TIMER0_IRQ interrupt in **Snake_Enviro()** and is set to run at the same interval as the other two functions.

The **Hero_launchAudio()** function plays the audio file *avengers_Audio1*. It uses the global variables **hero_audio** to count samples and **hero_totalAudio** which holds the total number of samples. The **Audio_writeSpace()** function is used to check the FIFO space of the CODEC. If the FIFOs have space, the left and right samples are loaded to the respective DACs using the **Audio_writeToLeft()** and **Audio_writeToRight()** functions, and **hero_audio** is incremented. Since the function is running at a rate of 48 kHz, the DACs are loaded at the audio sample rate, thereby playing the audio. When **hero_audio** equals **hero_totalAudio**, the function is unregistered from the IRQ interrupt.

The **Hero_backAudio()** plays the *avengers_Audio2* file which is the background music of the game and the *avengers_Audio3* file, the shot sound effect. It works in a similar way as **Hero_launchAudio()** except that when **hero_audio** equals **hero_totalAudio**, it is set to zero, thereby looping the audio. Also, when a shot is fired, the shot sound effect is mixed with the background music by adding both samples before loading them to the DACs.

The **Snake_backAudio()** function plays the audio file *snake_Audio* which is the background music of the snake game. It also works the same way as **Hero_launchAudio()** but uses the global variables **snake_audio** and **snake_totalAudio** instead.

2.5. Interfacing with the SD card

The five highest scores of the Hero game are stored in descending order as lines of strings in a file ***scores1.txt*** on the micro SD card. The SD card is accessed using the FatFs driver, which is a FAT file system module that provides various filesystem functions. The FatFs file system is created and mounted in the *main.c*.

The high scores are saved to and retrieved from the SD card using the function **Hero_highScores()** . This function is called to update the high scores when the player losses all life-bars. It creates a file pointer and opens ***scores1.txt*** in a read only mode (FA_READ). The high scores are then read as strings, converted to integers, and stored in an array **hero_HiScores**. Each element of **hero_HiScores** are compared with the player's score to determine if the player has a score higher than the previous values. The **hero_HiScores** array is updated accordingly.

The updated high scores are written back to the SD card by opening ***scores1.txt*** in a create-always mode (FA_CREATE_ALWAYS). This mode ensures that the previous file is always truncated and overwritten. The scores

are then converted into strings and printed to ***scores1.txt***.

2.6. Drivers

The gaming environment employed a handful of drivers to ensure specific operations. These drivers are listed and explained below.

The first is a driver for the timers called **HPS_Timers,**. The **HPS_IRQ** and **HPS_12C** were used for button interrupting and audio setting in the audio driver respectively. The audrio driver used to interface with the audio CODEC on the board was the **WM8731_AudioEngine**. **HPS_usleep** was used as waiting functions within the code, to suspend all operations for a specified amount of time. A peripheral interface driver called the **IO_Peripherals** was written to interface with the buttons on the board. Interacting with the LCD module was enabled by using the **DE1SoC_LT24** driver. Interfacing with and external SD card was made possible using **FatFS** driver.

3. GAME STRUCTURE

This section describes the structure of the interactive gaming environment. It consists of a game menu where the user can select a game to play. The games in the environment include a retro snake game (like the popular 'Snake' on old Nokia phones) and a shooter game (themed after the popular marvel Avengers movies).

3.1. Game Menu

At the start of the game, the player is visited with a menu scene containing images and texts defining the available games. The scene also includes a cursor which the player moves left or right using KEY3 and KEY2 of the peripheral buttons. The desired game is selected using KEY0. The backend of the game menu operates using four major functions.

The **Game_Menu()** function sets a stage for the menu scene by clearing the display and assigning the default values of the selection variables **game_selection** and **game_selected**. The function also calls the **Game_menuScene()** to display the image selections on the screen, for both the hero game and the Snake game.

The selection cursor is displayed using the function **Game_selectCursor()** to draw a box around the images on the screen. It also attaches a navigation function, **Game_selectNav()** to the KEYS_IRQ peripheral interrupt, which is run when the user presses a button. The navigation function tracks the values of the selection integers and allows the push buttons to be used for toggling the cursor and selecting a game.

3.2. Hero Game

The hero game is run by **Hero_game()**, which is called in the **main** function if **game_selection** equals HEROGAME. It calls a set of functions to set up the game environment if a new game is requested. It then calls the function **Hero_inGame()** to handle the dynamics of the game.

3.2.1 Hero Environment

The hero game environment is setup using three functions which are called in **Hero_game()** if a new game is requested. It includes a launch scene which introduces the game and a selection scene that allows the

user to select a character. It also sets up the player character based on the user selection.

The launch scene is handled by the **Hero_launchScene()** function which displays a slide show of images on the LCD while playing a launch audio. The images were decompressed into raw RGB565 format using a MATLAB function (Appendix C) that generates a C source and header file containing the array. The slide show is displayed by polling the HPS private timer to change the image on the LCD every 0.87 seconds. The function plays the launch audio by attaching the audio function **Hero_launchAudio()** to the HPS_TIMER0_IRQ interrupt to run at a rate of 48 kHz. The interrupt is unregistered when the audio has finished playing.

The **Hero_selectScene()** function is called to handle the selection scene once the launch scene ends. This function plays the game background music, displays images of player characters, and allows the user to select a character. The background music is played by attaching the background audio function **Hero_backAudio()** to the HPS_TIMER0_IRQ interrupt. The interrupt is unregistered when the game is exited. Character selection is handled by the function **Hero_selectCursor()**. It blinks a box around the currently selected character image and allows the user to select with the push buttons by attaching the function **Hero_selectNav()** to the KEYS_IRQ interrupt.

Once the user has selected a character, the function **Hero_init()** is called to setup the player according to the user's selection and initialise the game variables to default values. It also initialises the shooter beams, enemy ships, and draws the player life bar. This completes the environment setup.

3.2.2 Hero Dynamics

The game dynamics is handled by the function **Hero_inGame()**. It handles the game-play by running a set of functions using the HPS private timer non-blocking polling. If the player runs out of live-bars, a function **Hero_reset()** is called to run the game reset scene. This function waits for a key interrupt to either restart the current game or navigate to the game menu. In **Hero_inGame()**, during game play, four functions are called to process the motion of the player's avatar, the shooting process, the enemy ships and the life-bar.

Movement of the avatar on the screen are handled by two functions. These are **Hero_dynamics()** and **Hero_move()**. The former simply polls the player inputs by setting a flag when the buttons are pressed it is pressed. The flags are then used throughout the code to signify a button press and then are turned off (a form of debouncing). In **Hero_move()**, when a button is pressed to move the avatar (KEY3 for left and KEY2 for right), the LCD image buffer function is called with the x-coordinate value incremented or decremented towards the direction of motion.

The shooting mechanism of the game employs an intricate algorithm using **struct**. Each shooting beam is a structure with x and y coordinates, and two Booleans (one to signify when a bullet is active and the other for when the bullet hits). In **Hero_shooter()**, when KEY0 is pressed, an instance of a beam is created, with the properties of the **struct** unique to that instance. A maximum of twenty-seven beams can be on the screen at the same time. The active beams have their respective y values incremented in a loop. A beam is rendered inactive whenever it hits the top of the screen or hits an enemy.

In **Hero_fleet(),** the same method of **struct** employed for the shooter is repeated. Instances of enemy fleets are created from this structure and spawned on the screen at regular intervals. All instances of the enemy fleet on the screen move downwards towards the player at the bottom of the screen.

Both the fleet and shooter beam structures have a collision Boolean that is turned true whenever a beam hits the fleet. Individually, the collision Boolean of the beam instances are turned on when the beams hit the top of the screen, and that of the fleet, when the enemy ships hit the bottom of the screen. Once an instance of a beam or an enemy ship has its collision set to true, that instance is cleared from the screen. **Hero_fleet()** and **Hero_shooter()** are both called in **Hero_inGame()**.

The final function that adds to the dynamics of the hero game is **Hero_life()**. This function keeps track of the player life-bar and the score. The score is incremented whenever a collision occurs between a fleet ship and a beam. This function prints the scores on the LCD. Five green boxes are also drawn on the screen using **Graphics_drawBox()** function to signify the player life-bar. When the player runs out of live-bars, the function **Hero_highScore()** is called to read the high scores from the SD card and compare it with the player's score. The high scores are updated accordingly and written back to the SD card. The **Hero_printOver()** function is called to write the high scores on the LCD.

3.3. Snake Game

The functions that run the snake game are classified into two sections. The first section deals with the environment of the game and the second section deals with the dynamics of the game.

3.3.1 Snake Environment

In the Snake environment, a new game is initialised using the **Snake_Game()** function. When the function is called, a start sequence sets up the environment using the **Snake_Enviro()** function. This function sets triggers to end the game whenever an end event occurs. During this event, the reset sequence prints 'GAME OVER' on the display and requests for KEY2 to be pressed to continue, or KEY3 to return to the main menu.

In the **Snake_Enviro()** function, the snake is initialised in the environment using **Snake_init()** and a border is drawn on the screen. The background audio function **Snake_backAudio()** is then attached to the **HPS_TIMER0_IRQ** interrupt. This interrupt is de-registered once the game is exited. During gameplay, four functions are employed to process the motion of the snake, the spawning of the food, the snake growth and its destruction. These are **snake_execute()**, **food_execute()**, **Snake_grow()**, and **Snake_destruct()**.

In **Snake_destruct()**, the position of the snake's head is tracked to detect when it collides with an obstacle, or its own body. The **snake_map** array is heavily utilised to match the pixel map of the snake's head and its peripheral. When the pixel occluding its next path is registered as a snake body or an obstacle in the map array, an end event occurs. This end event triggers the reset sequence.

The **snake_execute()** and **food_execute()** are task scheduling functions that run the **Snake_dynamics()** and **Snake_food()** functions respectively at different time intervals.

In the **Snake_food()** function, x and y coordinated are picked at random to spawn a 4x4 pixel sized food, with the top left pixel at the coordinates chosen. The food is only spawned on a spot that is devoid of an obstacle or the snake body. The **Snake_dynamics()** function contains instances of **Snake_direction()**, **Snake_create()** and **Snake_clear()**, all from the snake dynamics section scripts.

3.3.2 Snake Dynamics

The body of the snake is initialised on the screen by the **Snake_init()**. This function first clears the display and sets the **snake_map** with its default values (all 0s). The snake's direction is also set to right immediately a snake is created. The snake body is then created with 1-pixel width and 50-pixel length line. Once the snake is created, the x and y coordinates of the head and tail of the snake are obtained. The snake's position on the screen is also reflected in the **snake_map** array with a value of 1 for every pixel the snake appears.

The **Snake_update()** function specifically draws a pixel head at a new position of the head coordinates, depending on how the snake moves in the environment. The function also clears the snake's tail pixel. The **Snake_create()** and **Snake_clear()** functions respectively move the snake's head and tail forward. In **Snake_create()**, the snake constantly moves in its current path until a button is pressed to change its direction. In **Snake_clear()**, the snake's tail simply moves in the direction of the previous body positions, acting as a cleaner of the snake's body in the environment. This constant creation of the snake's head and clearing of its tail creates the illusion of motion.

The **Snake_grow()** function works in tandem with the **Snake_food()**. When the snake's head hits a food blob in the environment, the snake grows in length 10 pixels in its current direction. A global Boolean is then turned off, telling the **Snake_food()** function to re-spawn a new food blob somewhere else in the field.

The **Snake_direction()** function sets flags for the button inputs. When a button is pushed, the flags are turned off corresponding to that button. These flags are used in **Snake_create()** to change the increment of the x and y coordinates of the snake head. KEY3 is used to move the snake up, KEY2 moves the snake down, KEY1 moves the snake left and KEY0 moves the snake right.

4. TESTING AND VERIFICATION

4.1. Debugging

The code for the interactive gaming environment was compiled and built using the Altera SoC Embedded Design Suite. The compilation also served to check for errors within the code. The DS-5 debugger tool was used to load the program to the DE1-SoC board.

Since a lot of functions were used, break points were inserted at selected points in the code to test the functionality of different blocks of codes. The stack viewer was used to observe the list of the function call stack and step instruction control was used to observe the behaviour of the stack when a call to function is made.

The variables viewer was used to confirm the value of key variables such as **hero_audio** and **snake_audio** in the game audio functions, **game_selection** in **Game_menuScene()**, and **hero_selection** in **Hero_selectScene()**. The variables viewer was also used to track the value of various Booleans used in the code.

For the snake game, a function **Snake_test()** was created to view the current state of the array **snake_map**. This function prints the values of the array to the application console and was used to ensure that the values correctly represented the position of the snake in the environment.

Preliminary tests were undertaken to ensure that the push button inputs produced a correlating movement of the character images on the LCD screen. Considering the results obtained, the intervals of execution of the functions for motion of the images were adjusted to allow smooth transition.

Before integration with the game, the audio functions were tested with several audio clips. It was discovered that uncompressed audio clips required a large amount of memory. This

discovery lead to the use of shorter audio clips and resorting to looping of the clips.

4.2 Hardware Verification

After meticulous debugging of all sections of the program, the final compiled program was loaded to the DE1-SoC device and thoroughly tested. The game performed as expected in all scenarios. Evidences of operation of the game are presented in appendix A.

Attempt was made to load the baremetal image of the game to the SD card. With the DDRRomRamVFP scatter file, the board failed to load the image file from the SD card. The VFP feature of the board was disabled and scatter file was replaced with DDRRomRam. This also produced the same result. This failure of the baremetal image file to load from the SD card may be attributed to its large size (over 16 megabytes).

In the end, the program was run from the DS-5 debugger interface.

5. CONCLUSION

This project entailed the programming of an interactive gaming environment on the DE1-SoC development board. The gaming environment comprised of two games selectable by a user. These games were called HERO: a single person shooter, and the classic SNAKE. The games were displayed on the LT24 LCD module. Both games were programmed to make use of the peripheral buttons on the board to navigate the player (or snake) about the environment.

Programming of the gaming environment utilized external drivers for interfacing with the LCD, audio CODEC, on board timers, external SD card, peripheral buttons and 7 segment displays. The program also made use of code architectures such as task scheduling of functions and the use of structures for object-oriented programming.

Both games were split into 2 sections respectively; the environment and the dynamics of the game. The environment sections detailed the setup of the game fields on the LCD, timing of certain functions, game menus, player selection screens, game over scenes and game reset procedures. The dynamics section of the program highlighted the code processes employed to move the players and objects on the LCD.

The program was tested using the DS-5 debugger tool and verified on the DE1-SoC development board. Issues found during the debugging stage resulted in some modification of the code structure of the affected sections. The game met all specifications within its scope.

6. REFERENCES

[1] Terasic Technologies, *DE1-SoC User Manual 1 www.terasic.com August 5, 2015*. 2015.

[2] V. Soc, “DE1-SoC Computer System with ARM Cortex-A9 DE1-SoC Computer Contents DE1-SoC,” no. April, 2014.

[3] T. Wm, “w Portable Internet Audio CODEC with Headphone Driver and Programmable Sample Rates,” *Audio*, no. April, 2004.

[4] S. Freear, D. Cowell, and T. Carpenter, “Laboratory 1,” *ELEC5566M FPGA Des. Syst. Chip*, 2018.

[5] S. Freear, D. Cowell, and T. Carpenter, “Laboratory, 2.”

[6] S. Freear, D. Cowell, and T. Carpenter, “Laboratory 3,” *ELEC5566M FPGA Des. Syst. Chip*, 2018.

[7] S. Freear, D. Cowell, and T. Carpenter, “Laboratory 4,” *ELEC5566M FPGA Des. Syst. Chip*, 2018.

[8] S. Freear, D. Cowell, and T. Carpenter, “Laboratory 5,” *ELEC5566M FPGA Des. Syst. Chip*, 2018.

[9] S. Freear, D. Cowell, and T. Carpenter, “Laboratory 6,” *ELEC5566M FPGA Des. Syst. Chip*, 2018.

[10] S. Freear, D. Cowell, and T. Carpenter, “Laboratory 8,” *ELEC5566M FPGA Des. Syst. Chip*, 2018.

[11] FatFs, “FatFs - Generic FAT Filesystem Module.” [Online]. Available: http://elm-chan.org/fsw/ff/00index_e.html. [Accessed: 10-May-2018].

[12] Ebay.com, “Thor (black Background) 24 x 36 Poster | eBay.” [Online]. Available: https://www.ebay.com/itm/Thor-black-Background-24-x-36-Poster-/281664811950. [Accessed: 10-May-2018].

[13] Emoji.com, “Snake Emoji (U+1F40D).” [Online]. Available: http://www.iemoji.com/view/emoji/211/animals-nature/snake. [Accessed: 10-May-2018].

[14] Background Check, “black widow background 3 | Background Check All.” [Online]. Available: http://backgroundcheckall.com/black-widow-background-3/. [Accessed: 10-May-2018].

[15] Den of Geek, “The Collectible Side of Captain America | Den of Geek.” [Online]. Available: http://www.denofgeek.com/us/books-comics/captain-america/255174/the-collectible-side-of-captain-america. [Accessed: 10-May-2018].

[16] Sean Fallon, “Hot Toys’ New Black Panther Figure is Glorious.” [Online]. Available: http://comicbook.com/marvel/2018/02/09/hot-toys-black-panther-figure/. [Accessed: 10-May-2018].

APPENDIX

APPENDIX A: HARDWARE VERIFICATION RESULTS

Fig. A1: Game Menu Scene

Fig. A2: Hero Game Launch Scene

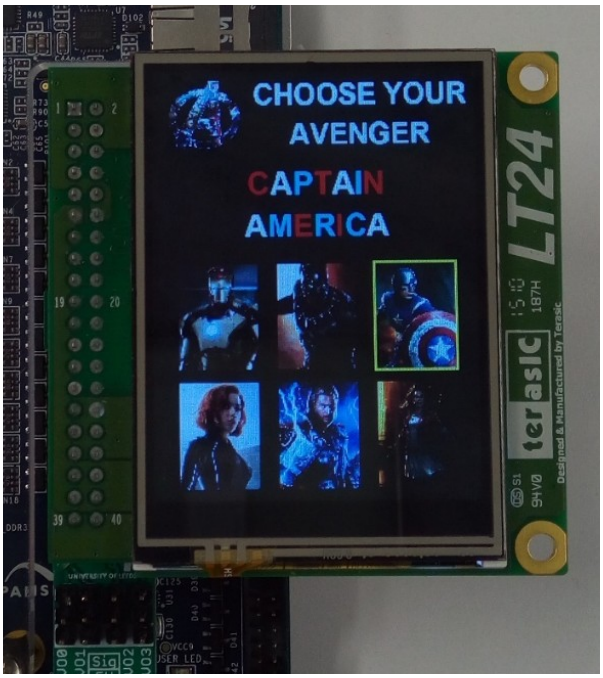

Fig. A3: Hero Game Selection Scene

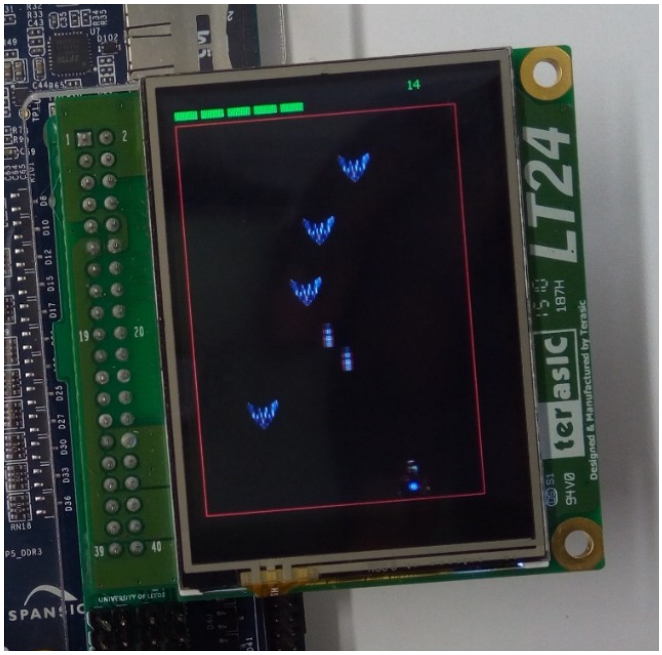

Fig. A4: Hero Game Play

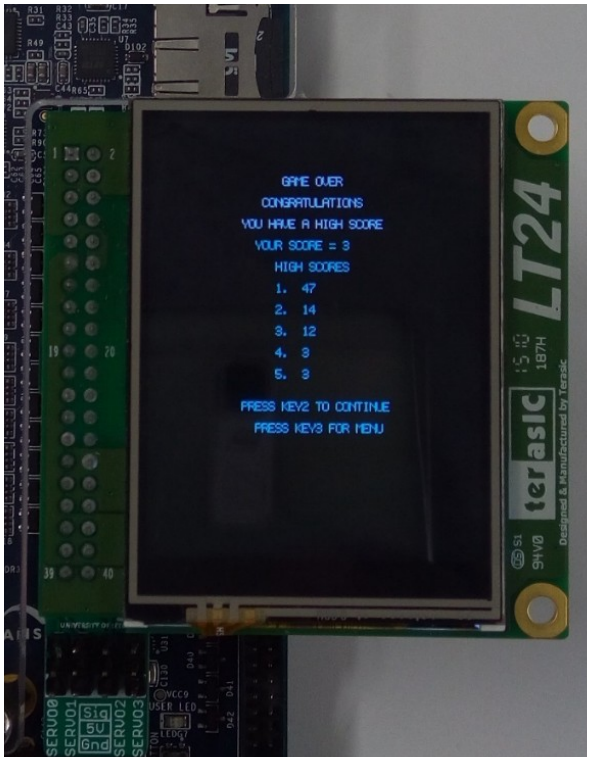

Fig. A5: High Score Scene

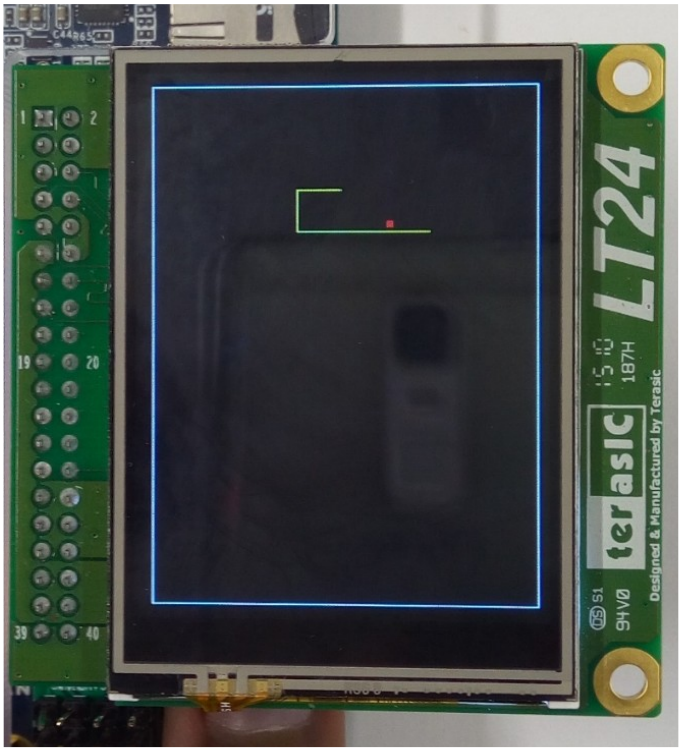

Fig. A6: Snake Game Play

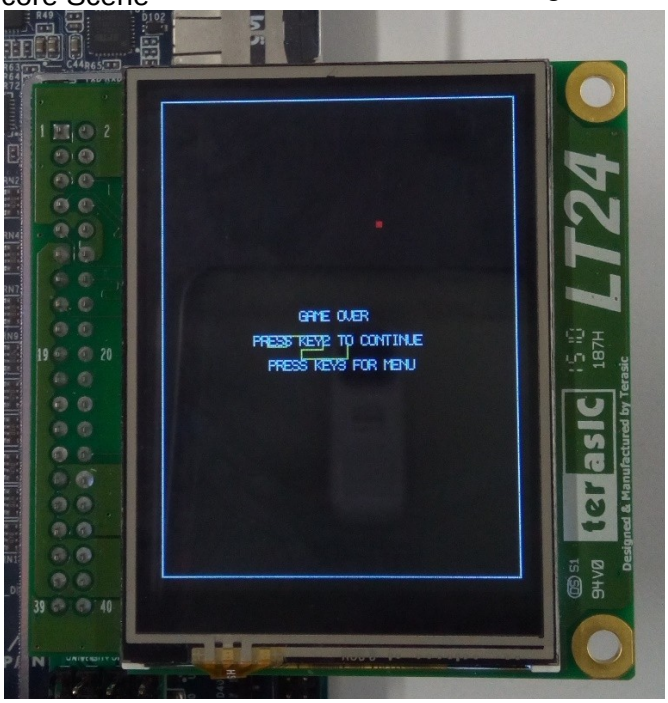

Fig. A7: Snake Game Over Scene

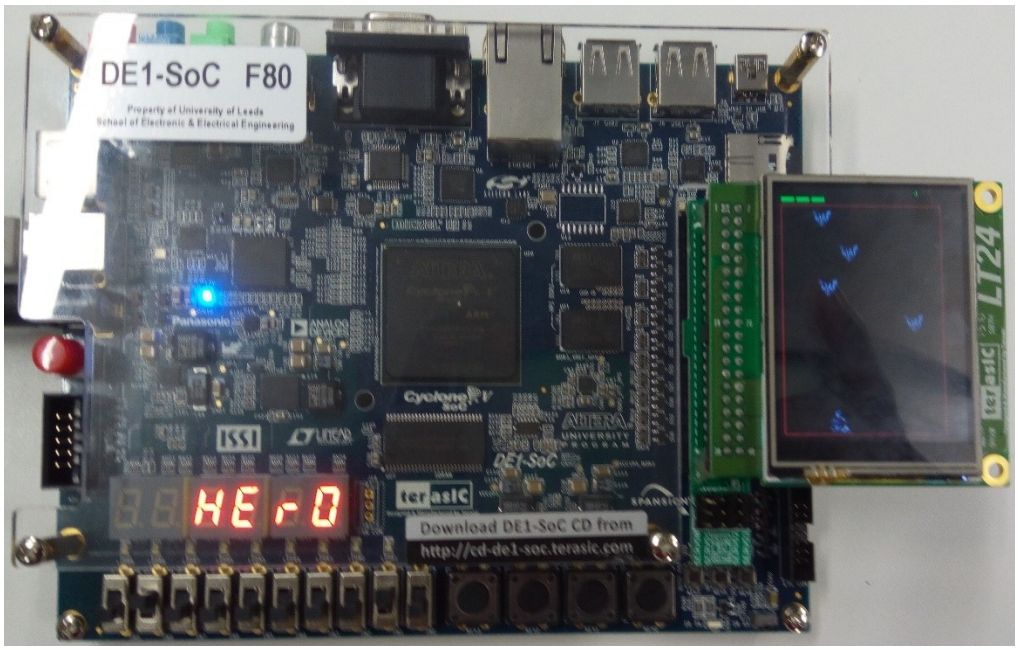

Fig. A8: Ingame showing seven-segment display

APPENDIX B: PROGRAM CODES

```
/*
 * main.c
 *
 * Created on: 11 Apr 2018
 *  Authors:
 *
 *      Azeem Oguntola  SID: 201162945
 *      Chima Nnadika   SID: 201077064
 *
 * Description
 * ------------------------------------------------------------------
 * This is an interactive gaming environment
 * It consists of two games; Hero/Avengers shooter game and
 * a snake game the user can select which game to play
 *
 * -- Features
 *
 * - Saving of high scores to SD Card
 * - Playing of audio in game background
 * - Display of Images and Shapes on the LCD
 * - Display of text on seven-segment
 * - Text display on LCD using lookup table
 * - Use of buttons for inputs
 *
 * -- Software Design Techniques
 *
 * - Use of Peripheral Interrupts (KEYS and TIMER0)
 * - Task Scheduling using Timer Polling
 * - Use of Structs
 * - Non-blocking Polling of inputs
 */

// Include required header files

#include "HeroGame/Hero_Enviro.h"   // Include hero game
#include "SnakeGame/Snake_Enviro.h" // include snake game

// Exit on fail function prototype
void exitOnFail(signed int status, signed int successStatus);

// Create FATFS file system for accessing the SD card
FATFS FatFs;

// Main function
int main(void){

    // mount file system
    f_mount(&FatFs, "", 0);
    ResetWDT();

    // Initialise the LCD graphics engine
    exitOnFail(
            Graphics_initialise(0xFF200060,0xFF200080),
            LT24_SUCCESS);
```

```
    ResetWDT();

    // Initialise the Private timer and timer0
    exitOnFail(
            Timer_initialise(),
            TIMER_SUCCESS);
    // configure the private timer
    exitOnFail(
            Timer_setControlRegister (false, true, true),
            TIMER_SUCCESS);
    // Load the private timer
    exitOnFail(
            Timer_setLoadValue (0xFFFFFFFF),
            TIMER_SUCCESS);
    // Set the prescaler value to 0
    exitOnFail(
            Timer_setPrescalerValue (0),
            TIMER_SUCCESS);
    ResetWDT();

    // Initialise the audio codec
    exitOnFail(
            Audio_initialise(),
            WM8731_SUCCESS);
    //Clear both FIFOs
    Audio_clearFIFO(true,true);
    ResetWDT();
    // Initialise the KEYS driver for IRQ
    IO_initialise();
    // Initialise IRQ
    HPS_IRQ_initialise();
    ResetWDT();
    Audio_outVolume (8);          // Set audio volume to level 8
    ResetWDT();

    // Main run loop
    while(1){

        // See GameMenu, HeroGame and SnakeGame folders for function details.

        // Run the game menu function
        Game_Menu();
        // if the hero game was selected, run the hero game
        if (game_selection == HEROGAME) Hero_game();
        // else if the snake game was selected, run the snake game
        else if (game_selection == SNAKEGAME) Snake_Game();
        ResetWDT(); // reset watchdog
    }


}


// Exit on fail function
void exitOnFail(signed int status, signed int successStatus){
    if (status != successStatus) {
```

```
        exit((int)status); //Add breakpoint here to catch failure
    }
}


/*
 * GameMenu.h
 *
 * Created on: 11 Apr 2018
 *  Authors:
 *
 *      Azeem Oguntola  SID: 201162945
 *      Chima Nnadika   SID: 201077064
 *
 * Description
 * ------------------------------------------------------------------
 * This file contains functions that run the game main menu scene
 *
 */
#ifndef GAMEMENU_H_
#define GAMEMENU_H_

/* Inclusions */

// Include the necessary drivers
#include "../HPS_Watchdog/HPS_Watchdog.h"
#include "../HPS_Timers/HPS_Timers.h"                    // Timer driver for private
timer and timer 0
#include "../BasicFont/BasicFont.h"
#include "../LT24_GraphicsEngine/LT24_GraphicsEngine.h"
#include "../IO_Peripherals/IO_Peripherals.h"              // Functions for KEY
interrupt
#include "../HPS_IRQ/HPS_IRQ.h"
#include "../HPS_usleep/HPS_usleep.h"

// Include standard libraries
#include <stdbool.h>
#include <math.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>

// Include images
#include "images/menu_title.h"
#include "images/menu_lSnake.h"
#include "images/menu_lAvengers.h"
#include "images/menu_tSnake.h"
#include "images/menu_tAvengers.h"

// Define Macros to indicate game selection
#define HEROGAME       1
#define SNAKEGAME      2

// Define Text for display on 7-segment //

#define A           0x77
```

```
#define C           0x39
#define E           0x79
#define H           0x76
#define L           0x38
#define N           0x37
#define O           0x3F
#define R           0x50
#define S           0x6D
#define T           0x78

// External Globals
extern unsigned int game_selection;
extern unsigned int game_selected;
extern bool newMenu;

/* Function prototypes */

// Wrapper Function to run the game menu scene
void Game_Menu(void);

// Function to play the game manu scene
void Game_menuScene (void);

// Function to select a game
void Game_selectCursor (void);

// Function to navigate through characters
void Game_selectNav (void);

// Function to Display Location on Seven Segment
void Game_hexDisplay(void);

#endif /* GAMEMENU_H_ */



/*
 * GameMenu.c
 *
 * Created on: 11 Apr 2018
 *  Authors:
 *
 *      Azeem Oguntola  SID: 201162945
 *      Chima Nnadika   SID: 201077064
 *
 * Description
 * ------------------------------------------------------------------
 * This file contains functions that run the game main menu scene
 *
 */

#include "GameMenu.h"

// External Globals

bool newMenu = true;
unsigned int game_selection;
```

```
unsigned int game_selected;

/* Wrapper Function to run the game menu scene */
void Game_Menu(void){
    // display on hex
    // This block runs the game menu scene if newMenu is requested
    if(newMenu) {
        game_selection = 0;
        game_selected = HEROGAME;
        Game_hexDisplay();
        Game_menuScene();
        LT24_clearDisplay(LT24_BLACK);
        ResetWDT();
        newMenu = false;
    }
}

/* Function to play the game manu scene */
void Game_menuScene (void) {
    // This function displays the title of the game environment
    // and allows the user to select between the available games

    // Clear display and display the game logos
    LT24_clearDisplay(LT24_BLACK);
    ResetWDT();
    LT24_copyFrameBuffer(menu_title,20,10,200,50);
    ResetWDT();
    LT24_copyFrameBuffer(menu_lAvengers,25,150,90,90);
    ResetWDT();
    LT24_copyFrameBuffer(menu_lSnake,125,150,90,90);
    ResetWDT();

    usleep(500000); // wait a while
    // Play the cursor select function
    Game_selectCursor();
}

/* Function to select a game */
void Game_selectCursor (void) {
    // This function allows the user to select a game. It displays the
    // game names and the selection cursor as the user moves through the games.
    // The HPS private timer is used to blink the cursor

    // local variables
    unsigned int currentTimerValue, lastTimeValue;
    const unsigned int timePeriod = 80000000;   // 0.35 secs
    unsigned short blinkColour, colour;     // colour values
    bool swap = false;      // To know when to change cursor colour

    game_selection = 0;
    // Attach the navigation fucntion to the KEYS_IRQ.
    // The navigation function allows the user to move
    // through the games
    HPS_IRQ_registerHandler(KEYS_IRQ, Game_selectNav);
    ResetWDT();
    // Begin private timer polling
```

```
    lastTimeValue = Timer_currentValue();
    // While a selection has not been made, we wait for the user to
    // make a selection. When a selection is made, the variable
    // game_selection will no longer be zero and this loop will end.

    while (game_selection == 0) {
        currentTimerValue = Timer_currentValue();   // Get current timer value
        // Check if is time to run the function
        if ((lastTimeValue - currentTimerValue) >= timePeriod) {
            // Change the cursor colour according to value of swap
            if (swap) blinkColour = LT24_YELLOW;
            else blinkColour = LT24_GREEN;

            // Begin to display game names and cursor.
            // We set cursor colour to blinkcolour for the currently selected
            // character. Otherwise, we set it to the background which is black.
            // The cursor is box bordering the images that blinks
            // by changing from yellow to green and back

            // Herogame
            if (game_selected == HEROGAME) {
                LT24_copyFrameBuffer(menu_tAvengers,45,80,150,30);
                ResetWDT();
                colour = blinkColour;
            } else {
                colour = LT24_BLACK;
            }
            // draw cursor
            Graphics_drawBox(colour,false,0,23,148,116,241);
            Graphics_drawBox(colour,false,0,24,149,115,240);
            ResetWDT();

            // snake game
            if (game_selected == SNAKEGAME) {
                LT24_copyFrameBuffer(menu_tSnake,45,80,150,30);
                ResetWDT();
                colour = blinkColour;
            } else {
                colour = LT24_BLACK;
            }
            // draw cursor
            Graphics_drawBox(colour,false,0,123,148,216,241);
            Graphics_drawBox(colour,false,0,124,149,215,240);
            ResetWDT();

            swap = !swap;       // toggle swap
            // update the lastTimeValue to when the program was last run
            lastTimeValue -= timePeriod;
        }

        // Clear the private timer interrupt flag
        Timer_clearInterrupt();
        ResetWDT();
    }
}
```

```c
/* Function to navigate through characters */
void Game_selectNav (void) {
    // This function allows the user to move through games and select
    // a game. The function is attached to the KEYS_IRQ interrupt

    // See IO_Peripherals.c for more details on the IO functions
    // get the key that was pressed
    unsigned int keyID = IO_getKeyPressID();

    // use key3 for previous
    if (keyID == 3) {
        game_selected = 1;
    }
    // use key2 for next
    else if (keyID == 2) {
        game_selected = 2;
    }
    // use key0 to select game
    else if (keyID == 0) {
        // unregister this function from the interrupts
        HPS_IRQ_unregisterHandler(KEYS_IRQ);
        // assign the selected game to the selection variable
        game_selection = game_selected;
    }
}


/* Function to Display Location on Seven Segment */
void Game_hexDisplay(void) {
    volatile int *HEX3_0_ptr = (int *) 0xFF200020;
    volatile int *HEX5_4_ptr = (int *) 0xFF200030;

    // Display HERO
    if (game_selection == HEROGAME) {
        *HEX3_0_ptr = (H << 24)|(E << 16)|(R << 8)|(O);
        *HEX5_4_ptr = 0;
    // Display SNAKE - use C as K
    } else if (game_selection == SNAKEGAME) {
        *HEX3_0_ptr = (N << 24)|(A << 16)|(C << 8)|(E);
        *HEX5_4_ptr = (S);
    // Display SELECT
    } else {
        *HEX3_0_ptr = (L << 24)|(E << 16)|(C << 8)|(T);
        *HEX5_4_ptr = (S << 8)|(E);
    }


}



/*
 * Hero_Enviro.h
 *
 * Created on: 11 Apr 2018
 *  Authors:
 *
```

```
 *      Azeem Oguntola  SID: 201162945
 *      Chima Nnadika   SID: 201077064
 *
 * Description
 * ------------------------------------------------------------------
 * This file contains functions that setup the environment
 * for the avengers game and synchronize the game play.
 * It uses functions from hero.h and inherits its external variables.
 *
 */

#ifndef HERO_ENVIRO_H_
#define HERO_ENVIRO_H_

/* Inclusions */

#include "Hero.h"

// Include the launch scene images
#include "Images/avengers_Limage0.h"
#include "Images/avengers_Limage1.h"
#include "Images/avengers_Limage2.h"
#include "Images/avengers_Limage3.h"
#include "Images/avengers_Limage4.h"
#include "Images/avengers_Limage5.h"
#include "Images/avengers_Limage6.h"

// Include hero images for selection scene
#include "images/avengers_sLogo.h"
#include "images/avengers_sTitle.h"
#include "images/avengers_sCap.h"
#include "images/avengers_sIron.h"
#include "images/avengers_sPanther.h"
#include "images/avengers_sThor.h"
#include "images/avengers_sScarlet.h"
#include "images/avengers_sWidow.h"

// Include hero title images
#include "images/avengers_sCapText.h"
#include "images/avengers_sIronText.h"
#include "images/avengers_sPantherText.h"
#include "images/avengers_sThorText.h"
#include "images/avengers_sScarletText.h"
#include "images/avengers_sWidowText.h"

/* Function Prototypes */

// Wrapper function to run the Avengers game
void Hero_game(void);

// Function to run the avengers game
void Hero_inGame(void);

// Function to play the game launch scene
void Hero_launchScene (void);
```

```
// Function to play the game hero selection scene
void Hero_selectScene (void);

// Function to select game character (hero)
void Hero_selectCursor (void);

// Function to navigate through characters
void Hero_selectNav (void);


#endif /* HERO_ENVIRO_H_ */



/*
 * Hero_Enviro.c
 *
 * Created on: 11 Apr 2018
 *  Authors:
 *
 *      Azeem Oguntola  SID: 201162945
 *      Chima Nnadika   SID: 201077064
 *
 * Description
 * ------------------------------------------------------------------------
 * This file contains functions that setup the environment
 * for the avengers game and synchronize the game play.
 * It uses functions from hero.h and inherits its external variables.
 *
 */


#include "Hero_Enviro.h"


/* Wrapper function to run the Avengers game */
void Hero_game(void){
    // This is the function called in the main.c
    // If a new launch has been requested, run launch sequence
    if(hero_launch) {
        // Display Hero text on seven-seg
        Game_hexDisplay();
        Hero_launchScene();
        Hero_selectScene();
        Hero_init();
        hero_launch = false;
        }
    // Run the game
    Hero_inGame();
}

/* Function to run the avengers game */
void Hero_inGame(void){
    // This function runs the avengers game.
    // The HPS private timer is used to run the
    // game functions at specific intervals
```

```
    // If hero_dead is true, run the reset function
    if(hero_dead){
        Hero_reset();
        ResetWDT();
    }else{
    // Update current timer value and run the game functions

        hero_currentTime = Timer_currentValue();
        execute_hero(&Hero_motion);
        execute_shooter(&Hero_shooter);
        execute_fleet(&Hero_fleet);
        execute_life(&Hero_life);
    }
    // Clear the timer interrupt and reset the watch dog
    Timer_clearInterrupt();
    ResetWDT();
}

/* Function to play the game launch scene */
void Hero_launchScene (void) {
    // The launch scene is a slide show of images and a launch music.
    // The HPS private timer is used to slide through the images.
    // The HPS timer0 IRQ interrupt is used to run the music

    // local variables
    unsigned int currentTimerValue, lastTimeValue, img = 0;
    const unsigned int timePeriod = 200000000; //slide show image interval. 0.87 secs

    // Get total samples in launch audio file
    hero_totalAudio = sizeof(avengers_Audio1)/sizeof(avengers_Audio1[0]);
    // Initialize current sample counter to zero
    hero_audio = 0;
    // clear the display and reset the watch dog
    LT24_clearDisplay(LT24_BLACK);
    ResetWDT();

    // To play the launch music, we attach the Hero_launchAudio() function to timer0
    // IRQ interrupt. This function loads the audio codec with the next audio sample.
    // First we disable the timer, load it with our desired period, and then enable the
timer
    // The timer operates at 100MHz. 2083 achieves a frequency of 48KHz (audio sampling
frequency)
    // See HPS_Timers.c for more details about the timer functions

    Timer0_disable();       // disable timer0
    Timer0_load(2083);      // load with our desired value
    Timer0_enable();        // enable the timer

    // We now attach our audio function to timer0 IRQ
    HPS_IRQ_registerHandler(HPS_TIMER0_IRQ, Hero_launchAudio);
    ResetWDT();

    // Now we begin the slide show and stay at the last image until
    // the launch audio has finished playing. The HPS private timer
    // is used to periodically update the current image
```

```
    lastTimeValue = Timer_currentValue();   // Set lasttimevalue to the current timer
value
    // While the audio is still playing, run the slide show.
    while (hero_audio < hero_totalAudio) {
        currentTimerValue = Timer_currentValue();   // Get current timer value
        // Check if it is time to display next image
        if ((lastTimeValue - currentTimerValue) >= timePeriod) {
            // Display the slide show image indicated by variable img
            switch (img) {
                case 0:
                    LT24_copyFrameBuffer(avengers_Limage0,0,90,240,134);
                    break;
                case 1:
                    LT24_copyFrameBuffer(avengers_Limage1,0,90,240,134);
                    break;
                case 2:
                    LT24_copyFrameBuffer(avengers_Limage2,0,90,240,134);
                break;
                case 3:
                    LT24_copyFrameBuffer(avengers_Limage3,0,90,240,134);
                break;
                case 4:
                    LT24_copyFrameBuffer(avengers_Limage4,0,90,240,134);
                break;
                case 5:
                    LT24_copyFrameBuffer(avengers_Limage5,0,90,240,134);
                break;
                default:
                    LT24_copyFrameBuffer(avengers_Limage6,0,0,240,320);
            }
            // increment img if we are not on the last image.
            // Otherwise, we want it to stay at the last image
            if(img < 6) img++;
            // update the last time value to when program was last run
            lastTimeValue -= timePeriod;
        }

        // Clear the private timer interrupt flag
        Timer_clearInterrupt();
        ResetWDT();
    }
}

/* Function to play the game hero selection scene */
void Hero_selectScene (void) {
    // This function displays images the game characters (heroes),
    // attaches the background music, and allows the player to
    // choose a character using the keys.
    // The KEYS_IRQ is used to read the player action

    // Clear the display to black
    LT24_clearDisplay(LT24_BLACK);
    ResetWDT();
    // Display all character images
    LT24_copyFrameBuffer(avengers_sLogo,0,0,70,60);
    ResetWDT();
```

```
    LT24_copyFrameBuffer(avengers_sTitle,70,0,170,60);
    ResetWDT();
    LT24_copyFrameBuffer(avengers_sIron,15,145,60,80);
    ResetWDT();
    LT24_copyFrameBuffer(avengers_sPanther,90,145,60,80);
    ResetWDT();
    LT24_copyFrameBuffer(avengers_sCap,165,145,60,80);
    ResetWDT();
    LT24_copyFrameBuffer(avengers_sWidow,15,235,60,80);
    ResetWDT();
    LT24_copyFrameBuffer(avengers_sThor,90,235,60,80);
    ResetWDT();
    LT24_copyFrameBuffer(avengers_sScarlet,165,235,60,80);
    ResetWDT();

    // Set the background audio sample counter to zero
    // and get the total audio samples of the background audio
    hero_audio = 0;
    hero_totalAudio = sizeof(avengers_Audio2)/sizeof(avengers_Audio2[0]); // Total
samples in audio file

    // To play the background music, we attach the Hero_backAudio() function to timer0
    // IRQ interrupt. This function loads the audio codec with the next audio sample.
    // First we disable the timer, load it with our desired period, and then enable the
timer
    // The timer operates at 100MHz. 2083 achieves a frequency of 48KHz (audio sampling
frequency)
    // See HPS_Timers.c for more details about the timer functions

    Timer0_disable();       // disable timer0
    Timer0_load(2083);      // load with our desired value
    Timer0_enable();        // enable the timer

    // We now attach our audio function to timer0 IRQ
    HPS_IRQ_registerHandler(HPS_TIMER0_IRQ, Hero_backAudio);
    ResetWDT();

    // Then we run the select cursor function
    Hero_selectCursor();

}

/* Function to select game character (hero) */
void Hero_selectCursor (void) {
    // This function allows the user to select a character. It displays the
    // character names and the selection cursor as the user moves through characters.
    // The HPS private timer is used to blink the cursor

    // local variables
    unsigned int currentTimerValue, lastTimeValue;
    const unsigned int timePeriod = 80000000;   // 0.35 secs
    unsigned short blinkColour, colour;     // colour values
    bool swap = false;      // To know when to change cursor colour

    // set the hero selection to zero
    hero_selection = 0;
```

```
// Attach the navigation fucntion to the KEYS_IRQ.
// The navigation function allows the user to move
// through characters
HPS_IRQ_registerHandler(KEYS_IRQ, Hero_selectNav);
ResetWDT();

// Begin private timer polling
lastTimeValue = Timer_currentValue();
// While a selection has not been made, we wait for the user to
// make a character selection. When a selection is made, the variable
// hero_selection will no longer be zero and this loop will end.
while (hero_selection == 0) {
    currentTimerValue = Timer_currentValue();   // Get current timer value
    // Check if is time to run the function
    if ((lastTimeValue - currentTimerValue) >= timePeriod) {
        // Change the cursor colour according to value of swap
        if (swap) blinkColour = LT24_YELLOW;
        else blinkColour = LT24_GREEN;

        // Begin to display chracter names and cursor.
        // We set cursor colour to blinkcolour for the currently selected
        // character. Otherwise, we set it to the background which is black.
        // The cursor is set of lines bordering the character images that blinks
        // by changing from yellow to green and back

        // Characters

        // Iron man
        if (hero_selected == HERO_IRON) {
            LT24_copyFrameBuffer(avengers_sIronText,60,70,120,60);
            ResetWDT();
            colour = blinkColour;
        } else {
            colour = LT24_BLACK;
        }
        // draw the cursor
        Graphics_drawLine (colour, 13, 144, 76, 144);
        Graphics_drawLine (colour, 13, 143, 76, 143);
        Graphics_drawLine (colour, 13, 225, 76, 226);
        Graphics_drawLine (colour, 13, 226, 76, 225);
        Graphics_drawLine (colour, 13, 143, 13, 226);
        Graphics_drawLine (colour, 14, 143, 14, 226);
        Graphics_drawLine (colour, 75, 143, 75, 226);
        Graphics_drawLine (colour, 76, 143, 76, 226);
        ResetWDT();

        // Black panther
        if (hero_selected == HERO_PANTHER) {
            LT24_copyFrameBuffer(avengers_sPantherText,60,70,120,60);
            ResetWDT();
            colour = blinkColour;
        } else {
            colour = LT24_BLACK;
        }
        // draw the cursor
```

```
Graphics_drawLine (colour, 88, 144, 151, 144);
Graphics_drawLine (colour, 88, 143, 151, 143);
Graphics_drawLine (colour, 88, 225, 151, 226);
Graphics_drawLine (colour, 88, 226, 151, 225);
Graphics_drawLine (colour, 88, 143, 88, 226);
Graphics_drawLine (colour, 89, 143, 89, 226);
Graphics_drawLine (colour, 150, 143, 150, 226);
Graphics_drawLine (colour, 151, 143, 151, 226);
ResetWDT();

// Captain America
if (hero_selected == HERO_CAP) {
    LT24_copyFrameBuffer(avengers_sCapText,60,70,120,60);
    ResetWDT();
    colour = blinkColour;
} else {
    colour = LT24_BLACK;
}
// draw the cursor
Graphics_drawLine (colour, 163, 144, 226, 144);
Graphics_drawLine (colour, 163, 143, 226, 143);
Graphics_drawLine (colour, 163, 225, 226, 226);
Graphics_drawLine (colour, 163, 226, 226, 225);
Graphics_drawLine (colour, 163, 143, 163, 226);
Graphics_drawLine (colour, 164, 143, 164, 226);
Graphics_drawLine (colour, 225, 143, 225, 226);
Graphics_drawLine (colour, 226, 143, 226, 226);
ResetWDT();

// Blackwidow
if (hero_selected == HERO_WIDOW) {
    LT24_copyFrameBuffer(avengers_sWidowText,60,70,120,60);
    ResetWDT();
    colour = blinkColour;
} else {
    colour = LT24_BLACK;
}
// draw the cursor
Graphics_drawLine (colour, 13, 234, 76, 234);
Graphics_drawLine (colour, 13, 233, 76, 233);
Graphics_drawLine (colour, 13, 315, 76, 316);
Graphics_drawLine (colour, 13, 316, 76, 315);
Graphics_drawLine (colour, 13, 233, 13, 316);
Graphics_drawLine (colour, 14, 233, 14, 316);
Graphics_drawLine (colour, 75, 233, 75, 316);
Graphics_drawLine (colour, 76, 233, 76, 316);
ResetWDT();

// Lord of thunder
if (hero_selected == HERO_THOR) {
    LT24_copyFrameBuffer(avengers_sThorText,60,70,120,60);
    ResetWDT();
    colour = blinkColour;
} else {
    colour = LT24_BLACK;
}
// draw the cursor
```

```
            Graphics_drawLine (colour, 88, 234, 151, 234);
            Graphics_drawLine (colour, 88, 233, 151, 233);
            Graphics_drawLine (colour, 88, 315, 151, 316);
            Graphics_drawLine (colour, 88, 316, 151, 315);
            Graphics_drawLine (colour, 88, 233, 88, 316);
            Graphics_drawLine (colour, 89, 233, 89, 316);
            Graphics_drawLine (colour, 150, 233, 150, 316);
            Graphics_drawLine (colour, 151, 233, 151, 316);
            ResetWDT();

            // Scarlet witch
            if (hero_selected == HERO_SCARLET) {
                LT24_copyFrameBuffer(avengers_sScarletText,60,70,120,60);
                ResetWDT();
                colour = blinkColour;
            } else {
                colour = LT24_BLACK;
            }
            // draw the cursor
            Graphics_drawLine (colour, 163, 234, 226, 234);
            Graphics_drawLine (colour, 163, 233, 226, 233);
            Graphics_drawLine (colour, 163, 315, 226, 316);
            Graphics_drawLine (colour, 163, 316, 226, 315);
            Graphics_drawLine (colour, 163, 233, 163, 316);
            Graphics_drawLine (colour, 164, 233, 164, 316);
            Graphics_drawLine (colour, 225, 233, 225, 316);
            Graphics_drawLine (colour, 226, 233, 226, 316);
            ResetWDT();

            swap = !swap;  // toggle swap to change blink colour
            // update the lastTimeValue to when the program was last run
            lastTimeValue -= timePeriod;
        }

        // Clear the private timer interrupt flag
        Timer_clearInterrupt();
        ResetWDT();
    }
}

/* Function to navigate through characters */
void Hero_selectNav (void) {
    // This function allows the user to move through characters and select
    // a character. The function is attached to the KEYS_IRQ interrupt


    // See IO_Peripherals.c for more details on the IO functions
    // get the key that was pressed

    unsigned int keyID = IO_getKeyPressID();
    // use key3 for previous
    if (keyID == 3) {
        if (hero_selected == 1) hero_selected = 6;
        else hero_selected --;
    }
    // use key2 for next
```

```
    else if (keyID == 2) {
        if (hero_selected == 6) hero_selected = 1;
        else hero_selected ++;
    }
    // use key1 to select character
    else if (keyID == 0) {
        // assign the selected hero to the selection variable
        hero_selection = hero_selected;
        // Then unregister this function from the KEYS_IRQ
        HPS_IRQ_unregisterHandler(KEYS_IRQ);
    }
}
```

```
/*
 * Hero.h
 *
 * Created on: 11 Apr 2018
 *  Authors:
 *
 *      Azeem Oguntola  SID: 201162945
 *      Chima Nnadika   SID: 201077064
 *
 * Description
 * ------------------------------------------------------------------
 * This script contains functions used to setup and drive the dynamics of
 * the hero player and enemy animations for the avengers game
 * It is a child script of Hero_Enviro.
 */


#ifndef HERO_H_
#define HERO_H_

/* Inclusions */

// Include the necessary drivers
#include "../HPS_Watchdog/HPS_Watchdog.h"
#include "../HPS_usleep/HPS_usleep.h"
#include "../HPS_Timers/HPS_Timers.h"                    // Timer driver for private
timer and timer 0
#include "../BasicFont/BasicFont.h"
#include "../FatFS/diskio.h"                                // File system functions to
access SD card
#include "../FatFS/ff.h"                                  // File system functions to
access SD card
#include "../LT24_GraphicsEngine/LT24_GraphicsEngine.h"
#include "../IO_Peripherals/IO_Peripherals.h"              // Functions for KEY
interrupt

// Include other header files
#include "../GameMenu/GameMenu.h"
#include "Audio.h"

// Include standard libraries
#include <stdbool.h>
```

```c
#include <math.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>

// Include hero images for game play
#include "images/avengers_pIron.h"
#include "images/avengers_pPanther.h"
#include "images/avengers_pCap.h"
#include "images/avengers_pWidow.h"
#include "images/avengers_pThor.h"
#include "images/avengers_pScarlet.h"

// Include enemy and bullet images
#include "images/avengers_IronShot.h"
#include "images/avengers_enemy1.h"

// Define Macros to indicate character selection
#define HERO_IRON         1
#define HERO_PANTHER      2
#define HERO_CAP          3
#define HERO_WIDOW        4
#define HERO_THOR         5
#define HERO_SCARLET      6

// Define maximum and minimum player position on the LCD
#define HERO_MAXLEFT      5
#define HERO_MAXRIGHT     185

/* External Global variables */

// Variables for execution timing
extern unsigned int hero_lastTime[5];
extern unsigned int hero_periods[5];
extern unsigned int hero_currentTime;

// Other external globals
extern bool hero_launch;
extern bool hero_dead;
extern unsigned int hero_selected;
extern unsigned int hero_selection;

// Typedef functions for timed executions
typedef void (*h_func)(void); //no inputs

/* Function Prototypes */

// Function to initialise the Hero game
void Hero_init(void);

// Function to move player left or right
void Hero_move(void);

// Function to update the button states
void Hero_dynamics(void);

// Function to activate a shot and move it
```

```
void Hero_shooter(void);

// Function to spawn and move enemy ship
void Hero_fleet(void);

// Function to keep track of player life
void Hero_life(void);

// Function to save highscores to SD card
void Hero_highScores (void);

// Function to print game over texts
void Hero_printOver (void);

// Function to play the game reset scene
void Hero_reset(void);

// Wrapper function for the player motion functions
void Hero_motion(void);

// Function to execute Hero_shooter function at specified intervals
void execute_hero(h_func p);

// Function to execute Hero_motion() function at specified intervals
void execute_shooter(h_func p);

// Function to execute fleet function at specified intervals
void execute_fleet(h_func p);

// Function to execute Hero_life() function at specified intervals
void execute_life(h_func p);
#endif /* HERO_H_ */



/*
 * Hero.c
 *
 * Created on: 11 Apr 2018
 *  Authors:
 *
 *      Azeem Oguntola  SID: 201162945
 *      Chima Nnadika   SID: 201077064
 *
 * Description
 * ------------------------------------------------------------------------
 * This script contains functions used to setup and drive the dynamics of
 * the hero player and enemy animations for the avengers game
 * It is a child script of Hero_Enviro.
 */


#include "Hero.h"

//initialise peripheral buttons
volatile unsigned int *BUTNS = (unsigned int *) 0xFF200050;
```

```
/* External Global Variables */

//variables for execution timing
unsigned int hero_lastTime[5] = {0,0,0,0,0};
unsigned int hero_periods[5] = {900000,1500000,270000000,6000000,900000};
unsigned int hero_currentTime = 0;
// Other externals
unsigned int hero_selection = 0;    //player selection check
unsigned int hero_selected = HERO_IRON; //default player selected
bool hero_launch = true;        // indicates a new launch
bool hero_dead;                 // if player is dead

/* Globals - non external */

bool hero_hit = false;    // if player is hit
unsigned int hero_xpose, hero_ypose; // player position
unsigned short *hero_active; //pointer to active player image
unsigned short hero_button[4] = {0,0,0,0}; //button states
bool spawnShip = false; //check if enemy ships are spawned
bool isHiScore = false; //Checks if player score is a high score

// scoring variables
unsigned int hero_score;
char hero_scoretxt[5]; //string of score
unsigned int hero_life; //holds number of lives
char hero_HiScoresTxt [5][10]; //string of high score
unsigned int hero_HiScores [5];


// Structure of bullet element
// contains instance properties of bullet positions, active and collision statuse
typedef struct{
    unsigned int beam_x;
    unsigned int beam_y;
    bool active;
    bool collide;
} shooter;

// Structure of enemy ship element
// contains instance properties of ship positions, active and collision statuses
typedef struct{
    unsigned int ship_x;
    unsigned int ship_y;
    bool active;
    bool collide;
} spaceship;

unsigned int shoot_id = 0; //instance id of shooter structures
unsigned int ship_id = 0; //instance id of ship structures

// on a straight line, only 27 beams can fit the screen, so it makes sense
// to make that the instance limit at any given time
shooter beam[27];

// an arbitrary number of ship instance limits on the screen
// (ideally 35 ships will never be on the screen simultaneously... ideally).
spaceship fleet[35];
```

```
/* Function to initialise the Hero game */
void Hero_init(void){
    // This function initialises the game variables to the default values
    // and sets up the player character and the game border
    unsigned int i;
    // Clear the display
    LT24_clearDisplay(LT24_BLACK);
    ResetWDT();

    // initialise shooter beam variables
    for(i=0; i<27; i++){
        beam[i].beam_y = 270;
        beam[i].active = false;
        beam[i].collide = false;
    }

    // initialise enemy spaceships
    for(i=0; i<35; i++){
        fleet[i].ship_y = 21;
        fleet[i].active = false;
        fleet[i].collide = false;
    }

    // Setup the player image according to the character selection
    switch (hero_selection) {
    case HERO_IRON:
        hero_active = avengers_pIron;
        break;
    case HERO_PANTHER:
        hero_active = avengers_pPanther;
        break;
    case HERO_CAP:
        hero_active = avengers_pCap;
        break;
    case HERO_WIDOW:
        hero_active = avengers_pWidow;
        break;
    case HERO_THOR:
        hero_active = avengers_pThor;
        break;
    case HERO_SCARLET:
        hero_active = avengers_pScarlet;
        break;
    default:
        hero_active = avengers_pIron;
    }

    // Set game variables to default
    hero_dead = false;
    hero_life = 5; //default number of lives
    hero_score = 0;
    //default position of player (uper left side of image position in screen)
    hero_xpose = 95;
    hero_ypose = 284;
```

```
    // Draw the Gameplay environment border
    Graphics_drawBox(LT24_RED, false, 0, 4, 20, 215, 315);
    ResetWDT();
    // Draw the Player picture
    LT24_copyFrameBuffer(hero_active, hero_xpose, hero_ypose,30,30);
    ResetWDT();

    //Draw Player life bar
    Graphics_drawBox(LT24_GREEN,1,LT24_GREEN,4,10,20,15);
    Graphics_drawBox(LT24_GREEN,1,LT24_GREEN,24,10,40,15);
    Graphics_drawBox(LT24_GREEN,1,LT24_GREEN,44,10,60,15);
    Graphics_drawBox(LT24_GREEN,1,LT24_GREEN,64,10,80,15);
    Graphics_drawBox(LT24_GREEN,1,LT24_GREEN,84,10,100,15);

    //set lastTime variables for timing executions
    hero_lastTime[0] = Timer_currentValue();
    hero_lastTime[1] = Timer_currentValue();
    hero_lastTime[2] = Timer_currentValue();
    hero_lastTime[3] = Timer_currentValue();
    hero_lastTime[4] = Timer_currentValue();

}

/* Function to move player left or right */
void Hero_move(void){
    // This checks if the move left or move right button state is true
    // is pressed and moves the player accordingly
    if(hero_button[0] && hero_xpose > HERO_MAXLEFT){
        LT24_copyFrameBuffer(hero_active, hero_xpose--, hero_ypose,30,30);
        ResetWDT();
        hero_button[0] = 0;
    }
    else if(hero_button[1] && hero_xpose < HERO_MAXRIGHT){
        LT24_copyFrameBuffer(hero_active, hero_xpose++, hero_ypose,30,30);
        ResetWDT();

        hero_button[1] = 0;
    }

}

/* Function to update the button states */
void Hero_dynamics(void){
    // This function handles Button dynamics of the player on the screen
    // Button 0 shoots
    // Button 2 and 3 moves player right and left.

    if(*BUTNS & 0x8){
        //hero is moving left
        hero_button[0] = 1;
    }
    if(*BUTNS & 0x4){
        //hero is moving right
        hero_button[1] = 1;
    }
```

```
    if(*BUTNS & 0x1){
        //hero is shooting
        hero_button[2] = 1;
    }
    if(*BUTNS & 0x2){
        //hero is blasting
        hero_button[3] = 1;
    }
}

/* Function to activate a shot and move it */
void Hero_shooter(void){
    // This function sets the dynamics of the beam shot by the player.
    // It creates an instance of the bullet object and assigns unique coordinates to
it.
    // When the bullet instance hits an enemy or has been exhausted, the function
clears
    // that particular instance of it.

    unsigned int i, j;

    //when the shooter button is pressed, activate a beam
    if(hero_button[2] && !(*BUTNS & 0X1)){
        beam[shoot_id].active = true;
        beam[shoot_id].beam_x = hero_xpose+10;
        LT24_copyFrameBuffer(avengers_IronShot, beam[shoot_id].beam_x,
beam[shoot_id].beam_y,10,10);
        shoot_id++;
        if(shoot_id == 27) shoot_id = 0; //reset the object identifier when all
instances are exhausted
        heroShot_audio = 0; //start the shooter audio sequence
        hero_button[2] = 0;
    }

    //all active beams in field should move upwards in field
    for(i=0; i < 27; i++){
        if(beam[i].active){ //if particular instance is active, move it upwards
            LT24_copyFrameBuffer(avengers_IronShot, beam[i].beam_x,
beam[i].beam_y--,10,10);
            ResetWDT();
            //check for active beams colliding with a ship
            for(j=0; j < 35; j++){
                if(fleet[j].active){
                    if(beam[i].beam_y == fleet[j].ship_y+29){
                        if((beam[i].beam_x+8 >= fleet[j].ship_x) && (beam[i].beam_x+1
<= fleet[j].ship_x+29)){
                            beam[i].collide = true;
                            fleet[j].collide = true;
                            Graphics_drawBox(LT24_BLACK,1,LT24_BLACK,150,0,239,19);
                            hero_score++; //add score when bullet hits ship
                        }
                    }
                }

            }
```

```
        }
        //beam disappears when it meets the end of the screen
        if(beam[i].beam_y == 21){
            beam[i].collide = true;
        }
        if(beam[i].collide){ //when a collision occurs, this resets the instance of
that bullet and resets it

Graphics_drawBox(LT24_BLACK,1,LT24_BLACK,beam[i].beam_x,beam[i].beam_y,beam[i].beam_x+9
,beam[i].beam_y+9);
            beam[i].active = false;
            beam[i].beam_y = 270;
            beam[i].collide = false;
        }
    }
    ResetWDT();
}

/* Function to spawn and move enemy ship */
void Hero_fleet(void){
    // This function executes the dynamics of the enemy fleet.
    // Much like the bullet structures, each enemy ship is an instance of the fleet
structure.
    // When the enemy reaches the end of the screen (bottom) or is hit by a bullet, the
ship
    // instance is destroyed.
    unsigned int i;
    if(spawnShip){ //it is time for a new ship instance to be spawned on screen
        fleet[ship_id].active = true;
        fleet[ship_id].ship_x = (rand()%(HERO_MAXRIGHT-HERO_MAXLEFT)+ HERO_MAXLEFT);
        LT24_copyFrameBuffer(avengers_enemy1, fleet[ship_id].ship_x,
fleet[ship_id].ship_y,30,30);
        ship_id++;

        if(ship_id == 35) ship_id = 0; //reset the object identifier when all instances
are exhausted

        spawnShip = false; //toggle this so it is turned on again for a new ship to
spawn
    }

    for(i=0; i < 35; i++){

        if(fleet[i].active){ //move ship downwards towards floor when it is on screen
            LT24_copyFrameBuffer(avengers_enemy1, fleet[i].ship_x, fleet[i].ship_y+
+,30,30);
            ResetWDT();
        }
        if(fleet[i].ship_y == 285){ //ship has reached end of screen.
            fleet[i].collide = true;
            hero_life--; //life of hero reduces when ship reaches the end of screen

        }

        if(fleet[i].collide){ //Reset instance of ship to default when it hits the end
of the screen or is hit by a bullet.
```

```
Graphics_drawBox(LT24_BLACK,1,LT24_BLACK,fleet[i].ship_x,fleet[i].ship_y,fleet[i].ship_
x+29,fleet[i].ship_y+29);
            fleet[i].active = false;
            fleet[i].ship_y = 21;
            fleet[i].collide = false;
            // redraw the player for persistence even if hit.
            LT24_copyFrameBuffer(hero_active, hero_xpose, hero_ypose,30,30);
            ResetWDT();
        }
    }
}

/* Function to keep track of player life*/
void Hero_life(void){
    // This function keeps track of the player life
    // It draws a life bar and keeps the player scores whenever it
    // destroys a ship.
    // Print the score
    sprintf(hero_scoretxt, "%d", hero_score);
    Graphics_printText(LT24_GREEN,36,0, hero_scoretxt);

    if(hero_life == 4) Graphics_drawBox(LT24_BLACK,1,LT24_BLACK,84,10,100,15);
    else if(hero_life == 3) Graphics_drawBox(LT24_BLACK,1,LT24_BLACK,64,10,80,15);
    else if(hero_life == 2) Graphics_drawBox(LT24_BLACK,1,LT24_BLACK,44,10,60,15);
    else if(hero_life == 1) Graphics_drawBox(LT24_BLACK,1,LT24_BLACK,24,10,40,15);
    else if(hero_life == 0){// when player is dead, run end of life routine
        hero_dead = true;
        Hero_highScores(); //run high score check and save it to file
        LT24_clearDisplay(LT24_BLACK);
        usleep(1000000);
        Hero_printOver(); //print end of life sequence
        ResetWDT();
    }

}

/* Function to save highscores to SD card */
void Hero_highScores (void) {
    // This function reads the high scores from the SD card,
    // Checks if the player score is a high score, and updates the high scores
    // and writes them back to the sd card
    // Five high scores are saved

    FIL fil;    // Create file pointer
    unsigned int i = 0;
    char buffer[200];   // String to store the high scores

    ResetWDT();
    // Open the high score file and read the values into
    // hero_HiScores array.
    f_open(&fil, "scores1.txt", FA_READ);
    ResetWDT();
    while (f_gets(buffer, sizeof buffer, &fil)) {
        hero_HiScores[i] = atoi (buffer);   //atoi converts the string to integer
        if (i < sizeof (hero_HiScores)) i++;
```

```
    }
    f_close(&fil);  // close the file
    ResetWDT();
    isHiScore = true;   // assume player has a highscore

    // Check if the player's score is a high score and update the
    // High score array accordingly
    if (hero_score > hero_HiScores[0]) {
        hero_HiScores [4] = hero_HiScores [3];
        hero_HiScores [3] = hero_HiScores [2];
        hero_HiScores [2] = hero_HiScores [1];
        hero_HiScores [1] = hero_HiScores [0];
        hero_HiScores [0] = hero_score;
    }
    else if (hero_score > hero_HiScores[1]) {
        hero_HiScores [4] = hero_HiScores [3];
        hero_HiScores [3] = hero_HiScores [2];
        hero_HiScores [2] = hero_HiScores [1];
        hero_HiScores [1] = hero_score;
    }
    else if (hero_score > hero_HiScores[2]) {
        hero_HiScores [4] = hero_HiScores [3];
        hero_HiScores [3] = hero_HiScores [2];
        hero_HiScores [2] = hero_score;
    }
    else if (hero_score > hero_HiScores[3]) {
        hero_HiScores [4] = hero_HiScores [3];
        hero_HiScores [3] = hero_score;
    }
    else if (hero_score > hero_HiScores[4]) hero_HiScores [4] = hero_score;
    else isHiScore = false; // If not a high score, set isHiScore to false

    // Create a new high score file and write the updated high scores to it
    f_open(&fil, "scores1.txt", FA_CREATE_ALWAYS | FA_WRITE | FA_READ);
    ResetWDT();
    f_printf(&fil, "%d\n%d\n%d\n%d\n%d\n", hero_HiScores[0], hero_HiScores[1],
hero_HiScores[2], hero_HiScores[3], hero_HiScores[4]);
    ResetWDT();
    f_close(&fil);  // close the file

    // Convert the high scores to strings so the can be printed on the LCD
    sprintf(hero_HiScoresTxt[0], "%d", hero_HiScores[0]);
    sprintf(hero_HiScoresTxt[1], "%d", hero_HiScores[1]);
    sprintf(hero_HiScoresTxt[2], "%d", hero_HiScores[2]);
    sprintf(hero_HiScoresTxt[3], "%d", hero_HiScores[3]);
    sprintf(hero_HiScoresTxt[4], "%d", hero_HiScores[4]);

}

/* Function to print game over texts */
void Hero_printOver (void) {
    // This function prints the end of game text
    Graphics_printText(LT24_WHITE,19,5, "GAME OVER");

    if (isHiScore) {
        Graphics_printText(LT24_WHITE,16,7, "CONGRATULATIONS");
```

```
        Graphics_printText(LT24_WHITE,13,9, "YOU HAVE A HIGH SCORE");
    } else {
        Graphics_printText(LT24_CYAN,22,7, "OOPS!");
        Graphics_printText(LT24_CYAN,10,9, "YOU DON'T HAVE A HIGH SCORE");
    }
    ResetWDT();

    Graphics_printText(LT24_CYAN,15,11, "YOUR SCORE = ");
    Graphics_printText(LT24_CYAN,28,11, hero_scoretxt);
    ResetWDT();
    Graphics_printText(LT24_CYAN,18,13, "HIGH SCORES");
    Graphics_printText(LT24_CYAN,18,15, "1.");
    Graphics_printText(LT24_CYAN,22,15, hero_HiScoresTxt[0]);
    Graphics_printText(LT24_CYAN,18,17, "2.");
    Graphics_printText(LT24_CYAN,22,17, hero_HiScoresTxt[1]);
    Graphics_printText(LT24_CYAN,18,19, "3.");
    Graphics_printText(LT24_CYAN,22,19, hero_HiScoresTxt[2]);
    Graphics_printText(LT24_CYAN,18,21, "4.");
    Graphics_printText(LT24_CYAN,22,21, hero_HiScoresTxt[3]);
    Graphics_printText(LT24_CYAN,18,23, "5.");
    Graphics_printText(LT24_CYAN,22,23, hero_HiScoresTxt[4]);
    ResetWDT();

    Graphics_printText(LT24_CYAN,13,26, "PRESS KEY2 TO CONTINUE");
    Graphics_printText(LT24_CYAN,15,28, "PRESS KEY3 FOR MENU");

    ResetWDT();

}

/* Function to play the game reset scene */
void Hero_reset(void){
    // This function polls the key inputs and either restarts the game
    // or goes back to the main menu

    // Poll key inputs
    if(*BUTNS == 0x8) hero_button[0] = 1;
    if(*BUTNS == 0x4) hero_button[1] = 1;

    // Use key 2 to restart the game
    if(hero_button[1] & *BUTNS != 0x4) {
        hero_button[1] = 0;
        hero_button[0] = 0;
        Hero_init();
    // use key 3 to go to main menu
    } else if (hero_button[0] & *BUTNS != 0x8) {
        game_selection = 0;
        // unregister background music function from timer0 IRQ
        HPS_IRQ_unregisterHandler(HPS_TIMER0_IRQ);
        hero_button[0] = 0;
        hero_button[1] = 0;
        newMenu = true;
        hero_launch = true;

    }
    ResetWDT();
```

```
}

/* Wrapper function for the player motion functions */
void Hero_motion(void){
    Hero_dynamics();
    Hero_move();
}

/* Function to execute Hero_shooter function at specified intervals */
void execute_shooter(h_func p){
    if((hero_lastTime[0] - hero_currentTime) >= hero_periods[0]){
        //This function takes in Hero_shooter and runs it at the
        //rate of period[0]
        p();

        hero_lastTime[0] -= hero_periods[0];
    }
}

/* Function to execute Hero_motion() function at specified intervals */
void execute_hero(h_func p){
    if((hero_lastTime[1] - hero_currentTime) >= hero_periods[1]){
        //This function takes in Hero_motion() and runs it at the
        //rate of period[1]
        p();


        hero_lastTime[1] -= hero_periods[1];
    }
}

/* Function to execute fleet function at specified intervals */
void execute_fleet(h_func p){
    if((hero_lastTime[2] - hero_currentTime) >= hero_periods[2]){
        //This is the rate at which the fleets are spawned on the screen
        //period[2]
        spawnShip = true;

        hero_lastTime[2] -= hero_periods[2];
    }

    if((hero_lastTime[3] - hero_currentTime) >= hero_periods[3]){
        //This function takes in Hero_fleet() and runs it at the
        //rate of period[3]
        p();

        hero_lastTime[3] -= hero_periods[3];
    }

}

/* Function to execute Hero_life() function at specified intervals */
void execute_life(h_func p){
    if((hero_lastTime[4] - hero_currentTime) >= hero_periods[4]){
        //This function takes in Hero_life() and runs it at the
        //rate of period[4]
```

```
        p();

        hero_lastTime[4] -= hero_periods[4];
    }
}


/*
 * Audio.h
 *
 * Created on: 11 Apr 2018
 *  Authors:
 *
 *      Azeem Oguntola  SID: 201162945
 *      Chima Nnadika   SID: 201077064
 *
 * Description
 * ------------------------------------------------------------------------
 * This file contains the audio functions for the avengers game.
 * The audio files have been decompressed using matlab and converted
 * to a C-array of the samples.
 *
 */

#ifndef AUDIO_H_
#define AUDIO_H_

/* Inclusions */

// Include the necessary drivers
#include "../WM8731_AudioEngine/WM8731_AudioEngine.h"   // Audio driver
#include "../HPS_IRQ/HPS_IRQ.h"                         // HPS interrupt driver
#include "../HPS_Timers/HPS_Timers.h"                   // Timer driver for private
timer and timer 0

// Include the audio files
#include "Sounds/avengers_Audio1.h"         // Launch Scene audio
#include "Sounds/avengers_Audio2.h"         // Game background audio
#include "Sounds/avengers_Audio3.h"         // shot audio effect

// External Global Variables
extern unsigned int heroShot_audio;
extern unsigned int heroShot_totalAudio;
extern unsigned int hero_audio;
extern unsigned int hero_totalAudio;

/* Function prototypes */

// Function to play the launch scene audio
void Hero_launchAudio (void);

// Function to play the background audio and shot audio effect
void Hero_backAudio (void);


#endif /* AUDIO_H_ */
```

```c
/*
 * Audio.c
 *
 * Created on: 11 Apr 2018
 *  Authors:
 *
 *      Azeem Oguntola  SID: 201162945
 *      Chima Nnadika   SID: 201077064
 *
 * Description
 * ----------------------------------------------------------------------
 * This file contains the audio functions for the avengers game.
 * The audio files have been decompressed using matlab and converted
 * to a C-array of the samples.
 *
 */


#include "Audio.h"

/* External Global Variables */

// Total samples and current sample counter of background music
unsigned int hero_totalAudio;
unsigned int hero_audio;

// Total samples and current sample counter of shot audio
unsigned int heroShot_audio = sizeof(avengers_Audio3)/sizeof(avengers_Audio3[0]);
unsigned int heroShot_totalAudio = sizeof(avengers_Audio3)/sizeof(avengers_Audio3[0]);


/* Function to play the launch audio */
void Hero_launchAudio (void) {
    // This function is attached to timer0 interrupt to run at
    // 48KHz or 1/48000 seconds interval which is the audio sampling rate.
    // It loads audio samples to the audio codec

    double ampl = 1000000000.0;     //Music Amplitude
    // We don't want to repeat the audio. So once the current sample equals the total
samples,
    // unregister the interrupt, making timer0 available for further use as well.
    if (hero_totalAudio == hero_audio) {
        HPS_IRQ_unregisterHandler(HPS_TIMER0_IRQ);
    }
    // Check if there is space in the left and right audio fifos.
    // See WM8731_AudioEngine.c for more details
    if (Audio_writeSpace()) {
        // The audio file is a stereo file and thus has two channels
        // write the current sample to the left and right channels
        Audio_writeToLeft((signed int)(ampl*avengers_Audio1[hero_audio][0])); //col 0
left channel
        Audio_writeToRight((signed int)(ampl*avengers_Audio1[hero_audio][1]));//col 1
right channel
       hero_audio++; // Increment the sample counter
```

```
    }
    // Clear Timer0 interrupt. See HPS_Timers.c for more details
    Timer0_clearInterrupt();
}

/* Function to play the game background music and shot audio */
void Hero_backAudio (void) {
    // This function is attached to timer0 interrupt to run at
    // 48KHz or 1/48000 seconds interval which is the audio sampling rate.
    // It loads the background music audio samples to the audio codec.
    // If a shot is fired, it mixes the back ground music with the shot audio

    // local Variables
    double ampl1 = 700000000.0;     // Background music amplitude
    double ampl2 = 1200000000.0;    // Shot audio amplitude
    double shotAudio[2];            // sample holder for shot audio
    // initialize sample holder to zero
    shotAudio[0] = 0;
    shotAudio[1] = 0;

    // We want the background music to run in a loop. So once we get to the last sample
    // of the back ground music, we start again from sample 0.
    if (hero_audio >= hero_totalAudio) hero_audio = 0;

    // Check if there is space in the left and right audio fifos.
    // See WM8731_AudioEngine.c for more details
    if (Audio_writeSpace()) {

    // If the current sample counter of the shot audio is less than the total
    // samples, we assign the current sample to the sample holder. The sample counter
    // the shot audio is set to zero when a shot is fired.
        if (heroShot_audio < heroShot_totalAudio) {
            shotAudio[0] = avengers_Audio3[heroShot_audio][0];
            shotAudio[1] = avengers_Audio3[heroShot_audio][1];
            heroShot_audio ++;          // increment the sample counter of the shot
audio
        }
        // Then we add the background audio sample and the shot audio sample holder
        // and write them to the left and right channels
        Audio_writeToLeft((signed int)(ampl1*(avengers_Audio2[hero_audio][0])+
(ampl2*shotAudio[0])));  //col 0 left channel
        Audio_writeToRight((signed int)(ampl1*(avengers_Audio2[hero_audio][1])+
(ampl2*shotAudio[1])));//col 1 right channel
        hero_audio++;                   // Increment the sample counter of the
background audio
    }
    // Clear Timer0 interrupt. See HPS_Timers.c for more details
    Timer0_clearInterrupt();
}


/*
 * Snake_Enviro.h
 *
 * Created on: 11 Apr 2018
 *  Authors:
```

```
 *
 *      Azeem Oguntola  SID: 201162945
 *      Chima Nnadika   SID: 201077064
 *
 * Description
 * ------------------------------------------------------------------
 * This script contains functions used to setup and drive the environment of an
 * animated snake. It is uses snake.h
 *
 */


#ifndef SNAKE_Enviro_H_
#define SNAKE_Enviro_H_

/* Inclusions */

//inherit snake functions and map
#include "Snake.h"
#include "Snake_Graphics.h"

//define environment limits
#define X_MAX           222
#define X_MIN           14
#define Y_MAX           302
#define Y_MIN           14

/* External Global variables */
//variables for execution timing
extern unsigned int snake_lastTime[2];
extern unsigned int snake_periods[2];
extern unsigned int snake_currentTime;

extern bool snake_newGame;

// Typedef functions for timed executions
typedef void (*s_func)(void); //no inputs

/* Function prototypes */

// Wrapper Function to run the snake game
void Snake_Game(void);

// Function to run the snake game
void Snake_Enviro(void);

// Function to set a snake in the environment
void Snake_dynamics(void);

// Function to destroy the snake when it hits itself or a wall
void Snake_destruct(void);

// Function to spawn the food
void Snake_food(void);

//these two functions get random x and y coordinates
```

```
void get_y(void);
void get_x(void);


//timer execution functions
void snake_execute(s_func p);
void food_execute(s_func p);


#endif /* SNAKE_Enviro_H_ */


/*
 * Snake_Enviro.c
 *
 * Created on: 11 Apr 2018
 *  Authors:
 *
 *      Azeem Oguntola  SID: 201162945
 *      Chima Nnadika   SID: 201077064
 *
 * Description
 * ------------------------------------------------------------------
 * This script contains functions used to setup and drive the environment of an
 * animated snake. It is uses snake.h
 *
 */


#include "Snake_Enviro.h"
#include "Audio.h"

/* External Global variables */

//variables for execution timing
unsigned int snake_lastTime[2] = {0,0};
unsigned int snake_periods[2] = {9000000,1000000};
unsigned int snake_currentTime = 0;

bool snake_newGame;

bool snake_startgame = true; //runs the begining of the game
unsigned short snake_button [2]; //button states

/* Wrapper Function to run the snake game */
void Snake_Game(void){
    // This function runs the functions that make up the snake game.
    // It runs the reset screen if the game is over
    if(snake_startgame){ //at start of game
        // Display SNAKE on seven-seg
        Game_hexDisplay();
        Graphics_printText(colours[4],17,20, " S N A K E  ");
        usleep(2000000);
        snake_newGame = true; //when this is true, it runs the environment instance
once
        snake_startgame = false;
```

```
    }

    Snake_Enviro(); //environment instance, runs the game until broken by an end of
life event

    if(snake_reset){
        //at end of life, show the game over screen
        Graphics_printText(colours[4],19,18, "GAME OVER");
        Graphics_printText(colours[4],13,20, "PRESS KEY2 TO CONTINUE");
        Graphics_printText(colours[4],15,22, "PRESS KEY3 FOR MENU");

        if(*BUTN == 0x8) snake_button[0] = 1;
        if(*BUTN == 0x4) snake_button[1] = 1;

        if(snake_button[1] & *BUTN != 0x4) { //button 2 runs snake game again
            //reset button states
            snake_button[0] = 0;
            snake_button[1] = 0;
            //start new snake game
            snake_newGame = true;
            snake_reset = false;

        } else if (snake_button[0]& *BUTN != 0x8) { //button 3 runs game menu
            game_selection = 0;
            HPS_IRQ_unregisterHandler(HPS_TIMER0_IRQ);
            //reset button states
            snake_button[0] = 0;
            snake_button[1] = 0;
            newMenu = true; //menu screen will run
            snake_startgame = true; //toggle on for when user enters snake game again
            snake_reset = false;
        }
        ResetWDT();
    }

}

/* Function to run the snake game */
void Snake_Enviro(void){

    //Creates game environment once
    if(snake_newGame){

        Snake_init(); //call snake instance
        Snake_drawBox(colours[4],0,0,5,5,235,315); //setup environment wall

        // set variables to default
        snake_reset = false;
        snake_initialised = false;
        food_isSpawned = false;
        food_isClear = true;
        snake_isPlaying = true;
        snake_newGame = false;

        // Set the background audio sample counter to zero
        // and get the total audio samples of the background audio
```

```
        snake_audio = 0;
        snake_totalAudio = sizeof(snake_Audio)/sizeof(snake_Audio[0]); // Total samples
in audio file

        // To play the background music, we attach the Hero_backAudio() function to
timer0
        // IRQ interrupt. This function loads the audio codec with the next audio
sample.
        // First we disable the timer, load it with our desired period, and then enable
the timer
        // The timer operates at 100MHz. 2083 achieves a frequency of 48KHz (audio
sampling frequency)
        // See HPS_Timers.c for more details about the timer functions

        Timer0_disable();       // disable timer0
        Timer0_load(2083);      // load with our desired value
        Timer0_enable();        // enable the timer

        // We now attach our audio function to timer0 IRQ
        HPS_IRQ_registerHandler(HPS_TIMER0_IRQ, Snake_backAudio);
        ResetWDT();

        snake_lastTime[0] = Timer_currentValue();
        snake_lastTime[1] = Timer_currentValue();
    }

    //while game is playing the snake dynamics
    // should run (snake body and food)
    if(snake_isPlaying){
        snake_currentTime = Timer_currentValue();
        Snake_grow();
        Snake_destruct();
        snake_execute(&Snake_dynamics);
        food_execute(&Snake_food);
    }
    Timer_clearInterrupt();
    ResetWDT();
}

/* Function to set a snake in the environment */
void Snake_dynamics(void){
    Snake_direction();
    Snake_create(colours[1],colours[7]);
    Snake_clear(colours[1],colours[7]);

}

/* Function to destroy the snake when it hits itself or a wall */
void Snake_destruct(void){

    //Detect an obstacle in all four directions
    if(snake_dir[0] && snake_map[head_xpose][head_ypose-1] == 8){
        snake_reset = true;
        snake_isPlaying = false;
        usleep(1000000);
    }
```

```
    else if(snake_dir[1] && snake_map[head_xpose][head_ypose+1] == 8){
        snake_reset = true;
        snake_isPlaying = false;
        usleep(1000000);
    }
    else if(snake_dir[2] && snake_map[head_xpose-1][head_ypose] == 8){
        snake_reset = true;
        snake_isPlaying = false;
        usleep(1000000);
    }
    else if(snake_dir[3] && snake_map[head_xpose+1][head_ypose] == 8){
        snake_reset = true;
        snake_isPlaying = false;
        usleep(1000000);
    }

    //Detect a snake body in all four directions
    else if(snake_dir[0] && snake_map[head_xpose][head_ypose-1] == 1){
        snake_reset = true;
        snake_isPlaying = false;
        usleep(1000000);
    }
    else if(snake_dir[1] && snake_map[head_xpose][head_ypose+1] == 1){
        snake_reset = true;
        snake_isPlaying = false;
        usleep(1000000);
    }
    else if(snake_dir[2] && snake_map[head_xpose-1][head_ypose] == 1){
        snake_reset = true;
        snake_isPlaying = false;
        usleep(1000000);
    }
    else if(snake_dir[3] && snake_map[head_xpose+1][head_ypose] == 1){
        snake_reset = true;
        snake_isPlaying = false;
        usleep(1000000);
    }

}

/* Function to spawn the food  */
void Snake_food(void){
    //this function is to spawn food at certain coordinates in screen
    //it has to be random and not at the points where
    //the snake already exists
    //food coordinates
    unsigned int i, j;

    //check field 3x3 from chosen coordinate for snake
    //or obstacle
    if(!food_isSpawned){
        get_x();
        get_y();
        for(i= food_ypose; i<food_ypose+FOOD_SIZE; i++){
            for(j = food_xpose; j< food_xpose+FOOD_SIZE; j++){
                if(snake_map[j][i] != 0 ){
                    food_isClear = false;
```

```
                }
            }
        }
    }
    //create the food
    if(food_isClear && !food_isSpawned){
        for(i = food_ypose; i < food_ypose+FOOD_SIZE; i++){
            for(j = food_xpose; j< food_xpose+FOOD_SIZE; j++){
                LT24_drawPixel(colours[0],j,i);
                snake_map[j][i] = 10;
            }
        }
        food_isSpawned = true;
    }

}

//get random x coordinates within environment limits
void get_x(void){
    unsigned int x;
    x = (rand()%(X_MAX-X_MIN)+ X_MIN);

    //we want coordinate to start at an even track
    if(x%2==0){
        food_xpose = x;
    }else{
        x++;
        food_xpose = x;
    }
}

//get random y coordinates within environment limits
void get_y(void){
    unsigned int y;
    y = (rand()%(Y_MAX-Y_MIN) + Y_MIN);

    //we want coordinate to start at an even track
    if(y%2==0){
        food_ypose = y;
    }else{
        y++;
        food_ypose = y;
    }
}
/* Function to exexute Snake_dynamics() */
void snake_execute(s_func p){
    if((snake_lastTime[0] - snake_currentTime) >= snake_periods[0]){
        //This  execution function takes in Snake_dynamics() and runs
        //it at an appropriate rate (9Mcycles/231.25MHz)
        p();

        //Snake_test();
        snake_lastTime[0] -= snake_periods[0];
    }
}

/* Function to exexute Snake_food() */
```

```
void food_execute(s_func p){
    if((snake_lastTime[1] - snake_currentTime) >= snake_periods[1]){
        //This execution function takes in Snake_food() and runs
        //at a rate much faster than the snake motion. (1Mcycles/231.25MHz)
        p();

        //Snake_test();
        snake_lastTime[1] -= snake_periods[1];
    }
}

/*
 * Snake.h
 *
 * Created on: 11 Apr 2018
 *  Authors:
 *
 *      Azeem Oguntola  SID: 201162945
 *      Chima Nnadika   SID: 201077064
 *
 * Description
 * ------------------------------------------------------------------------
 * This script contains functions used to setup and drive the dynamics of
 * a snake animation for the game 'Snake'. It is a child script of the
 * Snake Environment Snake_Enviro.
 */


#ifndef SNAKE_H_
#define SNAKE_H_


/* Inclusions */

// Include the necessary drivers
#include "../DE1SoC_LT24/DE1SoC_LT24.h"
#include "../HPS_Watchdog/HPS_Watchdog.h"
#include "../HPS_usleep/HPS_usleep.h"
#include "../WM8731_AudioEngine/WM8731_AudioEngine.h"
#include "../HPS_IRQ/HPS_IRQ.h"
#include "../HPS_Timers/HPS_Timers.h"
#include "../BasicFont/BasicFont.h"
#include "../GameMenu/GameMenu.h"

// Include standard libraries
#include <stdbool.h>
#include <math.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>


//define the default snake length at start
//and the size of the food at start
#define SNAKE_LENGTH        50
#define FOOD_SIZE           4
```

```
/* External Global variables */
//These variables are declared in the source file and are visible/used/altered
//anywhere this h file is called
extern unsigned short snake_map[240][320];
extern unsigned short colours[8];
extern bool snake_initialised;
extern volatile unsigned int *BUTN;
extern bool snake_reset;
extern bool snake_isPlaying;
extern signed int head_xpose,head_ypose;
extern signed int tail_xpose, tail_ypose;
extern unsigned short snake_dir[8];
extern signed int food_xpose, food_ypose;
extern bool food_isSpawned;
extern bool food_isClear;
extern bool food_isEaten;

/* Function prototypes */

// Function to initialise the snake
void Snake_init(void);

// Function to update snake position on LCD and map
void Snake_update(unsigned short snakeColour, unsigned short backgroundColour);

// Function to create the snake in motion
void Snake_create(unsigned short snakeColour, unsigned short backgroundColour);

// Function to clear the snake in motion
void Snake_clear(unsigned short snakeColour, unsigned short backgroundColour);

// Function to grow the snake when a food is eaten
void Snake_grow(void);

// Function to change the snake direction
void Snake_direction(void);

// Debugging Function to observe the snake map *Not part of the game
void Snake_test(void);



#endif /* SNAKE_H_ */

/*
 * Snake.c
 *
 * Created on: 11 Apr 2018
 *  Authors:
 *
 *      Azeem Oguntola  SID: 201162945
 *      Chima Nnadika   SID: 201077064
 *
 * Description
 * -------------------------------------------------------------------
```

```
 * This script contains functions used to setup and drive the dynamics of
 * a snake animation for the game 'Snake'. It is a child script of the
 * Snake Environment Snake_Enviro.
 */


#include "Snake.h"

/* External Global variables */

//initialise peripheral buttons
volatile unsigned int *BUTN = (unsigned int *) 0xFF200050;

bool snake_initialised = false;
bool started = true;

//these hold the positions of the snake head and tail
unsigned int start_xpose = 10,start_ypose = 50; //default start positions
signed int head_xpose = 0,head_ypose = 0;
signed int tail_xpose = 0, tail_ypose = 0;
signed int food_xpose = 0, food_ypose = 0;

//this is the map of the snake environment
unsigned short snake_map[240][320];
unsigned short colours[8] =
{LT24_RED,LT24_YELLOW,LT24_GREEN,LT24_CYAN,LT24_WHITE,LT24_BLUE,LT24_MAGENTA,LT24_BLACK
};
//boolean to know when the food has been eaten.
//this is useful to tell the program when to spawn
//new food
bool food_isEaten = false;
//boolean to check if food has been spawned
bool food_isSpawned = false;
//boolean to let us know field is clear
bool food_isClear = true;

// these booleans will let us know when the snake
// hits a wall or it self
bool snake_reset = false;
bool snake_isPlaying = false;

//this holds the direction memory
//0 to 3 holds current direction while 4 to 7 holds
//previous direction before a direction change
unsigned short snake_dir[8] = {0,0,0,1,0,0,0,0};

/* Function to initialise the snake */
void Snake_init(void){
    //This function cares about the snake body and
    //initialises it on the screen.
    //It also resets the LCD and snake map whwnever it is called
    //and places the snake in the default start position with
    //the default snake length.
    if(!snake_initialised){
        unsigned int i = 0, j = 0;
        LT24_clearDisplay(colours[7]);
```

```
        //clear and reset snake map
        for (i = 0; i < 240; i++ ) {
            for (j = 0; j < 320; j++) {
                snake_map [i][j] = 0;
            }
        }
        //reset snake default direction
        snake_dir[0] = 0;snake_dir[1] = 0;
        snake_dir[2] = 0;snake_dir[3] = 1;
        snake_dir[4] = 0;snake_dir[5] = 0;
        snake_dir[6] = 0;snake_dir[7] = 0;

        //game i set to start, it is time to make
        //a snake in the environment
        started = true;

        for(i=0;i<(SNAKE_LENGTH);i++){
            //create a snake with 10 pixel length and update snake memory map
            snake_map[start_xpose+i][start_ypose] = 1;
            LT24_drawPixel(LT24_YELLOW,start_xpose+i,start_ypose);
        }

        //once the game starts, we get the coordinates
        //of the snakes head and tail.
        if(started){
            head_xpose = start_xpose + (SNAKE_LENGTH-1);
            head_ypose = start_ypose;
            tail_xpose = start_xpose;
            tail_ypose = start_ypose;

            //initial positions of set, so we can turn this off
            started = false;
        }

        snake_initialised = true; //snake created
    }

}

/* Function to update snake position on LCD and map */
void Snake_update(unsigned short snakeColour, unsigned short backgroundColour){
    //updates state of the snake in the environment
    //create moving snake
    LT24_drawPixel(snakeColour,head_xpose,head_ypose);
    //update snake memory map
    snake_map[head_xpose][head_ypose] = 1;
    //clear snake tail
    LT24_drawPixel(backgroundColour,tail_xpose,tail_ypose);
    //update snake memory map
    snake_map[tail_xpose][tail_ypose] = 0;
}

/* Function to create the snake in motion */
void Snake_create(unsigned short snakeColour, unsigned short backgroundColour){

    //This function moves the head of the snake forward
```

```
    //check end of screen
    if(head_xpose == 240) head_xpose = 0;
    if(head_xpose < 0) head_xpose = 239;
    if(head_ypose == 320) head_ypose = 0;
    if(head_ypose < 0) head_ypose = 319;

    //change snake direction
    //the &&(head_xpose%2 != 0) makes sure the snake only
    //turns on an even pixel track. This is for error checking purposes
    //and to make sure the snake is always visible on the track
    // (it never touches itself except at the head)
    if(snake_dir[0]){
        if(snake_dir[6] && (head_xpose%2) != 0) head_xpose--;
        else if(snake_dir[7] && (head_xpose%2) != 0) head_xpose++;
        else head_ypose--;
    }
    else if(snake_dir[1]){
        if(snake_dir[6] && (head_xpose%2) != 0) head_xpose--;
        else if(snake_dir[7] && (head_xpose%2) != 0) head_xpose++;
        else head_ypose++;
    }
    else if(snake_dir[2]){
        if(snake_dir[4] && (head_xpose%2) != 0) head_ypose--;
        else if(snake_dir[5] && (head_xpose%2) != 0) head_ypose++;
        else head_xpose--;
    }
    else if(snake_dir[3]){
        if(snake_dir[4] && (head_xpose%2) != 0) head_ypose--;
        else if(snake_dir[5] && (head_xpose%2) != 0) head_ypose++;
        else head_xpose++;
    }
    //update snake state in environment
    Snake_update(snakeColour,backgroundColour);

}

/* Function to clear the snake in motion */
void Snake_clear(unsigned short snakeColour, unsigned short backgroundColour){
    //update snake state in environment
    Snake_update(snakeColour,backgroundColour);

    //check end of screen
    //check for the snakes last known tail position
    //and move towards that point
    if(tail_xpose > -1 && tail_ypose > -1 && tail_xpose < 240 && tail_ypose < 320){
        //this corrects a positioning error in the map i didn't understand
        if(tail_ypose == 319) snake_map[tail_xpose-1][319] = 0;
        if(tail_ypose == 0) snake_map[tail_xpose+1][0] = 0;

        //Allows the function follow the snake trail on the map
        if(snake_map[tail_xpose+1][tail_ypose] == 1) tail_xpose++;
        else if(snake_map[tail_xpose-1][tail_ypose] == 1) tail_xpose--;
        else if(snake_map[tail_xpose][tail_ypose+1] == 1) tail_ypose++;
        else if(snake_map[tail_xpose][tail_ypose-1] == 1) tail_ypose--;
    }
```

```
    //correct for edge looping
    if(tail_xpose == 240) tail_xpose = 0;
    if(tail_xpose == -1) tail_xpose = 239;
    if(tail_ypose == 320) tail_ypose = 0;
    if(tail_ypose == -1) tail_ypose = 319;

}

/* Function to grow the snake when a food is eaten */
void Snake_grow(void){

    unsigned int i = 0, j = 0;

    //if it sees food ahead of it, reset the food pixels
    //The code runs four times to accommodate for the four directions
    //the snake is moving as it finds a food to eat

    //snake is moving up and sees food
    if(snake_dir[0] && snake_map[head_xpose][head_ypose-2] == 10){

        for(i = food_ypose; i<food_ypose+FOOD_SIZE; i++){
            for(j = food_xpose; j< food_xpose+FOOD_SIZE; j++){
                LT24_drawPixel(colours[7],j,i);
                snake_map[j][i] = 0;
            }
        }

        //grow the snake
        for(i=head_ypose; i>head_ypose-10; i--){
            LT24_drawPixel(colours[1],head_xpose,i);
            snake_map[head_xpose][i] = 1;
        }
        head_ypose = head_ypose - 9;
        food_isSpawned = false;
    }
    //snake is moving down and sees food
    else if(snake_dir[1] && snake_map[head_xpose][head_ypose+2] == 10){

        for(i = food_ypose; i<food_ypose+FOOD_SIZE; i++){
            for(j = food_xpose; j< food_xpose+FOOD_SIZE; j++){
                LT24_drawPixel(colours[7],j,i);
                snake_map[j][i] = 0;
            }
        }

        //grow the snake in this direction
        for(i=head_ypose; i<head_ypose+10; i++){
            LT24_drawPixel(colours[1],head_xpose,i);
            snake_map[head_xpose][i] = 1;
        }
        head_ypose = head_ypose + 9;
        food_isSpawned = false;
    }
    //snake is moving left and sees food
    else if(snake_dir[2] && snake_map[head_xpose-2][head_ypose] == 10){

        for(i = food_ypose; i<food_ypose+FOOD_SIZE; i++){
```

```
            for(j = food_xpose; j< food_xpose+FOOD_SIZE; j++){
                LT24_drawPixel(colours[7],j,i);
                snake_map[j][i] = 0;
            }
        }

        for(i=head_xpose; i>head_xpose-10; i--){
            LT24_drawPixel(colours[1],i,head_ypose);
            snake_map[i][head_ypose] = 1;
        }
        head_xpose = head_xpose - 9;
        food_isSpawned = false;
    }
    //snake is moving right and sees food
    else if(snake_dir[3] && snake_map[head_xpose+2][head_ypose] == 10){

        for(i = food_ypose; i<food_ypose+FOOD_SIZE; i++){
            for(j = food_xpose; j< food_xpose+FOOD_SIZE; j++){
                LT24_drawPixel(colours[7],j,i);
                snake_map[j][i] = 0;
            }
        }

        for(i=head_xpose; i<head_xpose+10; i++){
            LT24_drawPixel(colours[1],i,head_ypose);
            snake_map[i][head_ypose] = 1;
        }
        head_xpose = head_xpose + 9;
        food_isSpawned = false;
    }
}

/* Function to change the snake direction */
void Snake_direction(void){
    //This function runs all the time to keep track of the button
    //pressed to change direction of the snake.
    //The snake uses its current direction to then know how its moving
    //The function also saves the previous direction for error purposes in motion
    unsigned int i=0;

    if(*BUTN & 0xF){
        //save previous direction for state check
        for (i=0;i<4;i++){
            snake_dir[i+4] = snake_dir[i];
        }
    }

    if(*BUTN & 0x8){
        //snake is moving up
        if(snake_dir[1]){
            //but not when it's moving down
            snake_dir[0] = 0;
        }else{
            snake_dir[0] = 1;snake_dir[1] = 0;
            snake_dir[2] = 0;snake_dir[3] = 0;
        }
    }
```

```
    if(*BUTN & 0x4){
        //snake is moving down
        if(snake_dir[0]){
            //but not when it's moving up
            snake_dir[1] = 0;
        }else{
            snake_dir[0] = 0;snake_dir[1] = 1;
            snake_dir[2] = 0;snake_dir[3] = 0;
        }
    }
    if(*BUTN & 0x2){
        //snake is moving left
        if(snake_dir[3]){
            //but not when it's moving right
            snake_dir[2] = 0;
        }else{
            snake_dir[0] = 0;snake_dir[1] = 0;
            snake_dir[2] = 1;snake_dir[3] = 0;
        }
    }
    if(*BUTN & 0x1){
        //snake is moving right
        if(snake_dir[2]){
            //but not when moving left
            snake_dir[3] = 0;
        }else{
            snake_dir[0] = 0;snake_dir[1] = 0;
            snake_dir[2] = 0;snake_dir[3] = 1;
        }
    }

}

/* Debugging Function to observe the snake map */
void Snake_test(void){
    // This function prints out the snake map in the console
    // It is not part of the game
    signed int h1, h2, h3, h4;
    signed int t1, t2, t3, t4;
    h1 = head_xpose - 1; t1 = tail_xpose - 1;
    h2 = head_xpose + 1; t2 = tail_xpose + 1;
    h3 = head_ypose - 1; t3 = tail_ypose - 1;
    h4 = head_ypose + 1; t4 = tail_ypose + 1;

    printf("Head Position: \n ------------ \n");
    printf("x: %d, y: %d \n", head_xpose, head_ypose);
    printf("%d   %d  %d  \n", snake_map[h1][h3],snake_map[head_xpose][h3],snake_map[h2]
[h3]);
    printf("%d   %d  %d  \n", snake_map[h1][head_ypose],snake_map[head_xpose]
[head_ypose],snake_map[h2][head_ypose]);
    printf("%d   %d  %d   \n", snake_map[h1][h4],snake_map[head_xpose]
[h4],snake_map[h2][h4]);
    printf("------------\n \n");
    printf("Tail Position: \n ------------ \n");
    printf("x: %d, y: %d \n", tail_xpose, tail_ypose);
    printf("%d   %d  %d  \n", snake_map[t1][t3],snake_map[tail_xpose][t3],snake_map[t2]
[t3]);
```

```
    printf("%d   %d  %d  \n", snake_map[t1][tail_ypose],snake_map[tail_xpose]
[tail_ypose],snake_map[t2][tail_ypose]);
    printf("%d   %d  %d   \n", snake_map[t1][t4],snake_map[tail_xpose]
[t4],snake_map[t2][t4]);

}

/*
 * Audio.h
 *
 * Created on: 11 Apr 2018
 *  Authors:
 *
 *      Azeem Oguntola  SID: 201162945
 *      Chima Nnadika   SID: 201077064
 *
 * Description
 * -------------------------------------------------------------------
 * This file contains the audio function for the snake game.
 * The audio file has been decompressed using matlab and converted
 * to a C-array of the samples.
 *
 */

#ifndef AUDIO_H_
#define AUDIO_H_

/* Inclusions */

// Include the necessary drivers
#include "../WM8731_AudioEngine/WM8731_AudioEngine.h"   // Audio driver
#include "../HPS_IRQ/HPS_IRQ.h"                         // HPS interrupt driver
#include "../HPS_Timers/HPS_Timers.h"                   // Timer driver

// Include the audio file
#include "Sounds/snake_Audio.h"         // Background music

// External Global Variables
extern unsigned int snake_audio;
extern unsigned int snake_totalAudio;

/* Function prototype */

// Function to play the background audio
void Snake_backAudio (void);


#endif /* AUDIO_H_ */

/*
 * Audio.c
 *
 * Created on: 11 Apr 2018
 *  Authors:
 *
 *      Azeem Oguntola  SID: 201162945
```

```
 *      Chima Nnadika   SID: 201077064
 *
 * Description
 * ------------------------------------------------------------------
 * This file contains the audio functions for the avengers game.
 * The audio files have been decompressed using matlab and converted
 * to a C-array of the samples.
 *
 */

#include "Audio.h"

/* External Global Variables */

// Total samples and current sample counter of background music
unsigned int snake_totalAudio;
unsigned int snake_audio;


/* Function to play the game background music and shot audio */
void Snake_backAudio (void) {
    // This function is attached to timer0 interrupt to run at
    // 48KHz or 1/48000 seconds interval which is the audio sampling rate.
    // It loads the background music audio samples to the audio codec.
    // If a shot is fired, it mixes the back ground music with the shot audio

    double ampl = 1000000000.0;     //Music Amplitude

    // We want the background music to run in a loop. So once we get to the last sample
    // of the back ground music, we start again from sample 0.
    if (snake_audio >= snake_totalAudio) snake_audio = 0;

    // Check if there is space in the left and right audio fifos.
    // See WM8731_AudioEngine.c for more details
    if (Audio_writeSpace()) {
       // The audio file is a stereo file and thus has two channels
        // write the current sample to the left and right channels
        Audio_writeToLeft((signed int)(ampl*snake_Audio[snake_audio][0])); //col 0 left
channel
        Audio_writeToRight((signed int)(ampl*snake_Audio[snake_audio][1]));//col 1
right channel
        snake_audio++;                  // Increment the sample counter
    }
    // Clear Timer0 interrupt. See HPS_Timers.c for more details
    Timer0_clearInterrupt();
}
```

APPENDIX C: MATLAB FUNCTIONS

```
% function to extract audio files
% place audio to be extracted in same directory as this function
% Example use AudioExtractor('audio.mp3', 'audio1');
% you will get an audio1.c file and audio1.h file with variable name audio1
```

```
function AudioExtractor(inputFileName, variableName)

    % read the audio file
    [audioData, SR] = audioread(inputFileName);
    % get the array size
    [row,col] = size(audioData);

    % If stereo (2 channels)
    if col == 2
        % Create C file
        fd=fopen([variableName '.c'],'wt');
        % print variable name
        fprintf(fd,['const double ' variableName ' [%d][%d]={'],row, col);
        % begin printing the array
        for chunkStart = 1:1:(row-1)
            fprintf(fd,'\n ');
            fprintf(fd,' {%d, %d},',audioData(chunkStart, 1), audioData(chunkStart,
2));
        end
        fprintf(fd,'\n ');
        fprintf(fd,' {%d, %d}',audioData(row, 1), audioData(row, 2));
        fprintf(fd,'\n }; \n\n');
        fclose(fd);

        % Create H file
        fd=fopen([variableName '.h'],'wt');
        fprintf(fd,['#ifndef ' upper(variableName) '_H_\n#define ' upper(variableName)
'_H_\n']);
        fprintf(fd,['extern const double ' variableName ' [%d][%d];\n\n\n'],row, col);
        fprintf(fd,'#endif \n\n');
        fclose(fd);

    end

    % if mono (single channel)
    if col == 1
        %c file
        fd=fopen([variableName '.c'],'wt');
        fprintf(fd,['const double ' variableName ' [%d]={'],row);
        for chunkStart = 1:1:(row-1)
            fprintf(fd,'\n ');
            fprintf(fd,' %d,',audioData(chunkStart));
        end
        fprintf(fd,'\n ');
        fprintf(fd,' %d};\n\n',audioData(row));
        fprintf(fd,'\n');
        fclose(fd);

        % Create H file
        fd=fopen([variableName '.h'],'wt');
        fprintf(fd,['#ifndef ' upper(variableName) '_H_\n#define ' upper(variableName)
'_H_\n']);
        fprintf(fd,['extern const double ' variableName ' [%d];\n\n\n'],row);
        fprintf(fd,'#endif \n\n');
        fclose(fd);
    end
```

```
end


%Converts Image to RGB565 C file. Make sure image correct size.
function Convert565(inputFileName, variableName)
    A = imread( inputFileName );
    h = double(A)/255; %Convert from 0-255 to 0-1
    rgb565=uint16( 2048*uint16(h(:,:,1)*31) + ...
    32*uint16(h(:,:,2)*63) + ...
    uint16(h(:,:,3)*31))';
    arraySize = numel(rgb565);
    fd=fopen([variableName '.c'],'wt');
    fprintf(fd,['const unsigned short ' variableName ' [%d]={'],arraySize );
    for chunkStart = 1:16:(arraySize-1)
        chunkEnd = chunkStart + min(15,arraySize-1-chunkStart);
        fprintf(fd,'\n ');
        fprintf(fd,' 0x%04X,',rgb565(chunkStart:chunkEnd));
    end

    if (mod(arraySize,16) == 1)
        fprintf(fd,'\n ');
    end
    fprintf(fd,' 0x%04X\n};\n',rgb565(end));
    fclose(fd);

    fd=fopen([variableName '.h'],'wt');
    fprintf(fd,['#ifndef ' upper(variableName) '_H_\n#define ' upper(variableName)
'_H_\n']);
    fprintf(fd,['extern const unsigned short ' variableName ' [%d];\n'],arraySize );
    fprintf(fd,'#endif');
    fclose(fd);


end
```