School of Electronic
and Electrical Engineering

UNIVERSITY OF LEEDS

ELEC5620M: Embedded Microprocessor System Design

MINI - PROJECT

Videogame: PONG

	STUDENT 1	STUDENT 2	STUDENT 3
NAME	**NUO XU**	**SANJITH CHANDRAN**	**SHRAJAN BHANDARY**
ID	**201286653**	**201290793**	**201289634**

PONG

1. ABSTRACT

The following project explains a technique to engineer a game engine with DE1-SOC boards and LT24-terasic displays. The game utilizes an external peripheral camera module, touch screen, push buttons and slide switches of the DE1-SOC to accommodate multiple user inputs complimented by the use of text and image to display different output scenarios of the game. Additionally, multiple boards communicate with one another using the GPIO pins.

2. INTRODUCTION

The game is inspired from a combination of the sport of Table Tennis and the arcade game called "Pong". The game consists of two paddles that can be moved up and down along the x-axis of the outer edges of the LCD screen while there are walls along the y-axis of the outer edges of the screen. There is also a net (dashed-line) that runs in parallel to the x-axis, from the middle of one wall to the other. The game starts by releasing the ball at from the centre of the LCD screen and travels at a random angle that belongs in 1 of 4 quadrants. The quadrants are split into angles that lie between 0-90 degrees, 90-180 degrees, 180-270 degrees and 270-360 degrees. The objective of the game is to keep the ball in play as it is able to bounce off the walls and paddles. The ball can only go out of play along the same edge as where the paddles are placed and depending on the side the ball is not kept in play; the opposite paddle/player wins a point. Wins are indicated with a green square on the winner's half of the game screen, whereas losses are indicated with a red square in the loser's half of the screen. The winner's point tally also increases by 1 on the seven-segment display while the loser's score stays the same. The first player to score 10 points (indicated with an A) wins the game and the winner's name is displayed on the LCD screen.

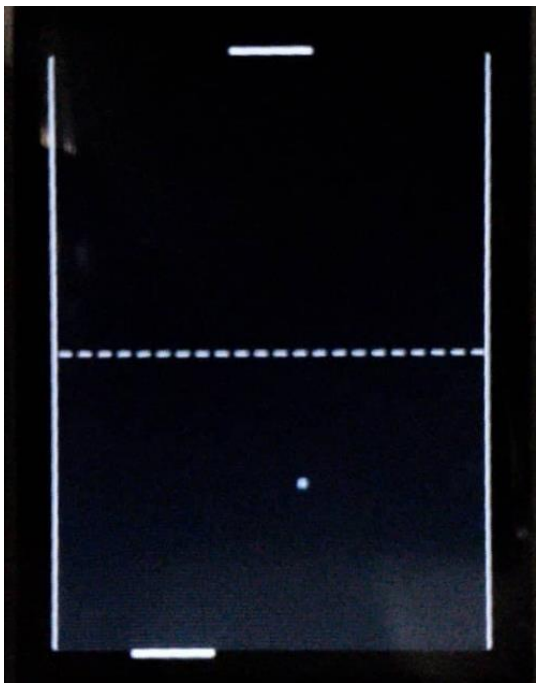

Figure 1: During Gameplay

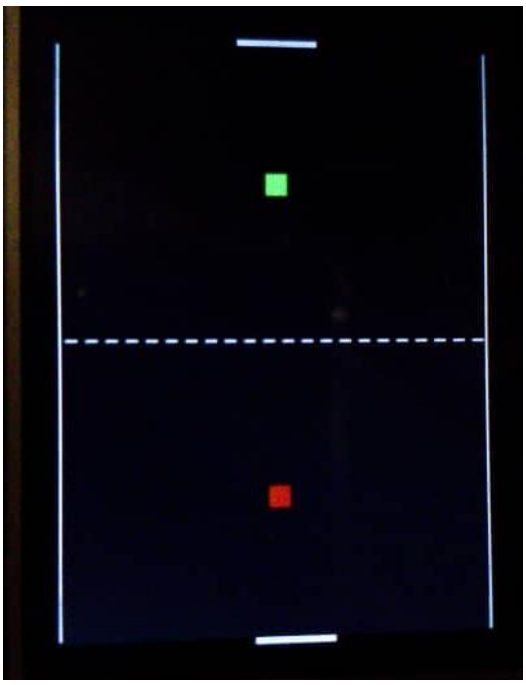

Figure 2: Winning or losing a round

3. GAME DISCUSSION

Pong follows the gameplay of the classic Table-Tennis game that is played between two players. Each player controls a paddle to deflect the ball onto the opponent's area. This report delineates the structure in which the game was developed consisting of retrieving user inputs, displaying the game play on the LCD screen, processing various data to manipulate the flow of the game and hardware implementations of the game. In order to increase efficiency, the game was conceived on two different DE1-SOC boards: Master and Slave. The two players control their respective paddles via two different methods. Player one controls their game-paddle by moving a red coloured object in front of a camera i.e. a red racquet and the second player controls their game-paddle with the help of two push buttons on the DE1-SOC board.

3.1. Slave DE1-SOC Board

The slave board's primary function is to reduce the processing load on the master board. The slave pong board is interfaced with the LCD and touch screen to generate the main screen of the game. The game begins once the touch screen has been tapped and this message is transmitted to the master pong board via the parallel data bus established using the GPIO 1 pins of the two boards. Next, the camera board performs image processing to acquire the location of the red coloured object and transmits the data to the slave board in the form of an 8-bit PWM signal via the GPIO pins. The slave board then transmits the 8-bit paddle-1 data to the master board via 8-bit parallel lines.

3.1.1. Interfacing the LCD and Touch Screen

The LT24 terasic consists of a 240x320 display along with a touch screen. The main screen of the game consists of two parts: image and text. The image was implemented by using concepts learnt in Lab 5 and modified slightly modified to only use half of the LCD Screen [1]. The other half was implemented using drawLine and drawBox functions that were developed for the Assignment. The touch screen of the LT24 board is connected to the DE1-SOC board via the GPIO pins. When the touch screen is tapped, an interrupt signal is transmitted to the GPIO through the 30th bit. A high value on this pin either starts or pauses the game.

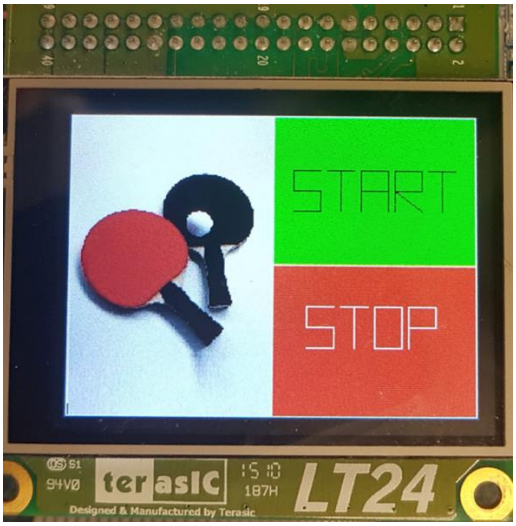

Figure 3: Start Screen with touch feature enabled

3.1.2. Interfacing the Camera Module

The camera module communicates with the slave board using a single 8-bit PWM signal. The data is retrieved by using the ARM A9 Private Timer that has a clock frequency of 225 MHz. The timer is enabled during the positive edge of the PWM signal and is disabled during the negative edge of the PWM signal. When the timer is enabled, it starts counting the number of clock cycles lapsed that corresponds to the 8-bit position of the paddle. This 8-bit data is reflected on LEDs of the slave pong board for the purpose of debugging.

An On-semiconductor full frame transfer global shutter CCD sensor is utilized for machine vision-related capturing. The full frame transfer functions allows the use of the CCD in variable lighting conditions without the assistance of artificial light, which is not permitted due to power consumption and disturbances. The global shutter characteristics is needed for removing any vision artefact caused by rolling shutter and fast-moving objects such as the fast-moving red coloured table tennis racquet. Such artefacts alter the object's actual shape to a horizontally distorted unrecognizable random shape. Tracking fast moving object with inevitable motion blur has proven to be critical for a vision-controlled game [2].

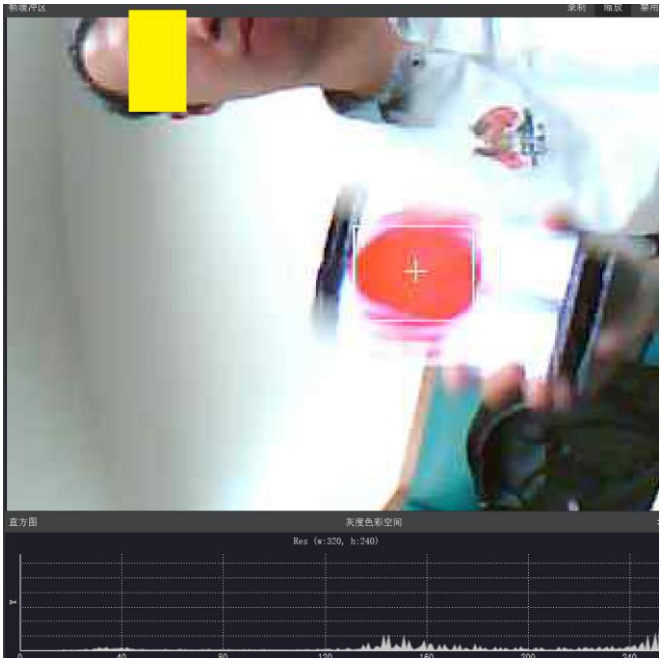

Figure 4. Global shutter with motion blur

STM32F767 is chosen due to its availability and performance for CTOS machine vision kits, although it is not power efficient, for the task of tracking a table tennis racquet, it was found to be more than capable of accomplishing the desired goal. This ARM processor runs FreeRTOS and a Micro Python interpreter on board, which trades a little bit of performance for faster debugging time [3].

- For computer vision, initial thresholding is performed followed by transformation of the raw video stream into different graded binary sequence video by thresholding the raw video with limits such as colour dependant filters. These parameters are adjusted by a machine vision tool-kit in Python.

- After the image has been binary graded, a grouping transform is performed onto the video stream. In each video frame, adjacent pixels of similar colour are grouped into an un-processed group.
- A merging process is needed for a clean result to be consider usable. Each centre coordinate of the computed binary shapes is processed, and each shape, with a centre location comprising of a centre distance smaller than a pre-adjusted value, is merged.
- Finally, after cross-correlation of the merged shape, using a 32x32 pixel shape mask, results in X and Y centre coordinates being calculated with the highest matching index shape. This is then returned using one-wire signalling.

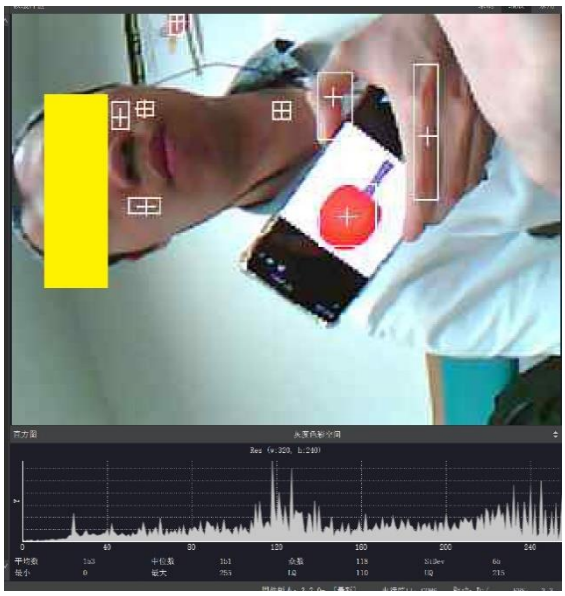

Figure 5. Before parameter optimization

The machine vision's parameters are optimized for the project's specific use case and tested with complex background and lighting conditions. For example, in Figure 7, there is someone's face, and two extra red patch looking logo, which in back-lit conditions, the machine vision successfully picked out the correct bat shaped object displayed on a phone screen.

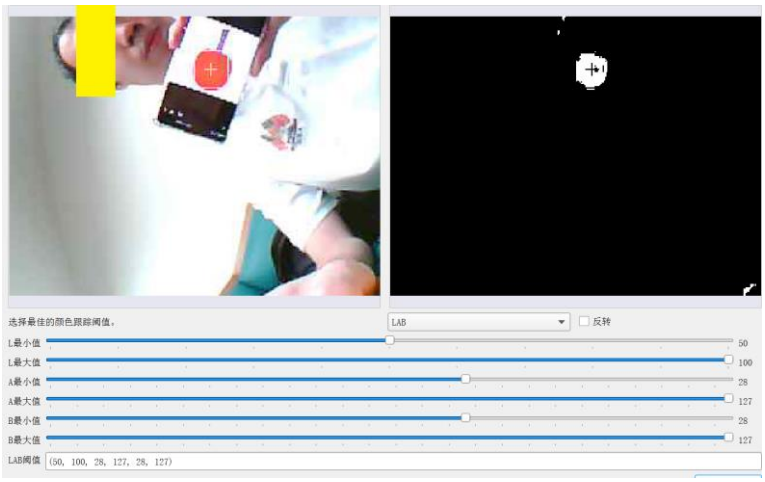

Figure 6. Adaptive Colour Pre-processor

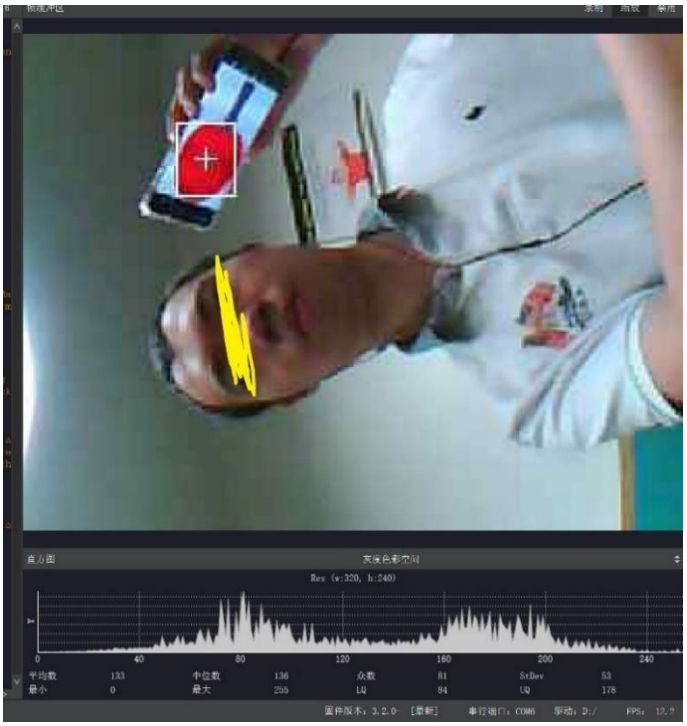

Figure 7. After optimization

Figure 8 shows a successful implementation and testing of the vision system when using a picture of a paddle for controlling the game-paddle in the Pong project.

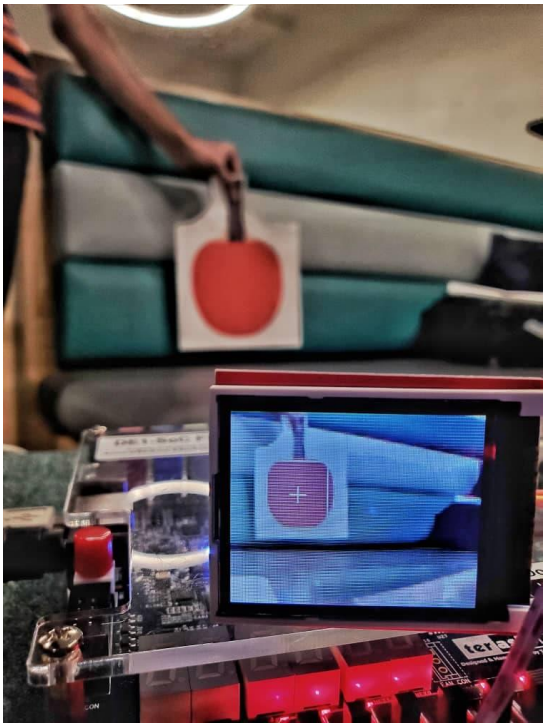

Figure 8. Implementation with DE1-SoC for Pong game

3.1.3. Interfacing LCD via PS-2 port

A second LCD screen is interfaced with the slave pong board to display the rules of the game, how to play the game as well as the length of time the game was played for.

Figure 9. Additional LCD Screen connected to DE1-SoC board via PS2 port

3.2. Master DE1-SOC board

The master pong board is the main control unit of the entire game. It houses the core outline of the game with functions such as obtaining inputs from the players, displaying patterns on the LCD screen and managing the overall flow of the game.

3.2.1. Game Screen

The main constituents that determine the flow of the game are the paddles and the ball. The two paddles are rectangular boxes with length and width of 10 pixels and 2 pixels respectively. The ball is a square box with length of 3 pixels. The paddles and ball are confined within a rectangular box of size 300x220 pixels, which represents the gaming area. The game doesn't start until the touch screen has been tapped. Once the screen is pressed, the game starts with each player score at 0 on the seven-segment display, paddles in the centre position at each side and the ball at the centre location of the gaming screen.

3.2.2. Paddles

The first player paddle is controlled by the moving the red object in front of the camera. The location of the second player paddle is decided by two push buttons: up (KEY-1) and down (KEY-0). In order to ensure inputs from DE1-SoC, to move the paddles, do not hinder the ball's movement, a technique called polling is used to ensure smooth software flow control. When the players press the keys/buttons or move the red coloured object, the paddles are drawn and erased after every iteration. This results in a pattern that resembles

the movement of the paddles. These paddles cannot move out of the gaming area even if the players apply brute force. The paddles are drawn on the LCD using the drawBox function of the graphics engine library.

3.2.3. Ball

The ball is drawn using a for-loop condition with the LT24_drawPixel command to draw a ball that is 3x3 pixels in size. The ball's movement is accomplished in a similar manner to that of the paddles' whereby the ball is drawn and erased as well.

3.2.4. Scoring, New round and Game speed

Each player scores a point when the opponent fails to touch the ball using their respective paddle. This triggers an increase in the score of the player and pauses the game for 3 seconds before the commencement of the next round. The first person is score 10 points is declared the winner. In addition, the speed of the movement of the ball can be altered based on the status of the slide switches of the master DE1-SOC board. The more switches that are high, the faster the speed of the ball.

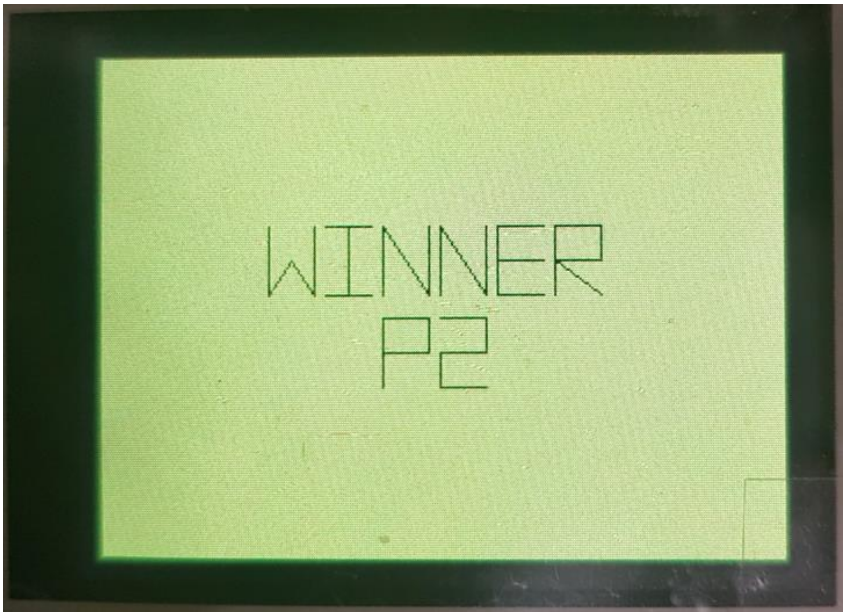

Figure 10. Winner Screen

3.3. Master and Slave communication

Data is transmitted across the two game boards via the GPIO 1 peripherals. The base address of this peripheral is 0xFF200070. However, based on their required function, these pins have to be configured as being either an input or an output. A pin is defined as an input by assigning a value of 0 to the corresponding bit position of the control register. Accordingly, if the assigned value is 1, then the corresponding pin is an output.

Example: *GPIO_pointer = 0xFF200070.

***(GPIO_pointer+4) = 0xFFFFFFFE.**

The above example sets the least significant bit as an input and the remaining 31 bits as outputs. This is the control register of the GPIO peripheral of the slave board. The signal from the camera module is connected to

pin 0 of the slave board's GPIO 1. In contrast, all pins of the GPIO peripheral on the master pong board are assigned as inputs to receive paddle data from the slave board.

3.4. Libraries of Game_Engine.c

This function contains a list of sub-routines that are briefly described below. There is also a list of predefined global and local variables that are responsible for setting the initial position and colour of the ball, paddles, walls and net.

signed int Paddle_X_Initialize: X is denoted by either 1 or 2

This function is for initializing the location and colour of the paddle, based on two x-y coordinates (top left and bottom right).

signed int Paddle_X_Draw: X is denoted by either 1 or 2

Given the top left and bottom right coordinates of each paddle, this function draws each paddle using a boundary colour.

signed int Graphics_drawDash

This function is responsible for drawing a dashed line along the centre of the LCD screen to represent the net for the Pong game. This function uses the Graphics_drawLine function.

signed int Move_Paddle_1

This function uses the camera to move the paddle during gameplay. The signal is received via the GPIO pins.

signed int Move_Paddle_2

This function is used for moving one of the paddles by using the keys/buttons on the DE1-SoC board, specifically KEY-0 and KEY-1.

signed int Graphics_drawBall

This function draws the ball on the screen using an assigned colour via the LT24_drawPixel function.

signed int Graphics_Erase

This function erases the ball by writing to pixels at the same location that the drawBall function uses and implementing a pixel colour that matches the background colour of the game. This is also accomplished by using the LT24_drawPixel function.

signed int Move_Ball

This function uses an initial angle (with respect to the x-axis) to determine the angle the ball travels. For the remainder of the program, in the event the ball hits either the wall or either paddle, the x-y coordinates of the ball determines the angle of departure for the trajectory of the ball. Each wall and paddle has two possible exit angles that are set as quadrant values. These quadrant values represent angles that lie in each 90-

degree quadrant. This function is also responsible to calculating an initial and final coordinate for the ball to travel. This initial point is the starting point of travel and the final is the point at which the ball contacts either the wall or the paddle. The next initial point of the ball is then set to the previous final point and so forth. The score is also updated if the paddle misses the ball.

void Update_Score_X: X is denoted by either 1 or 2

The score is updated in this function through the issue of the seven-segment display. HEX0 and HEX5 shows the scores that go from 0 to 9 (an A depicts a winner) while HEX1-4 is responsible for displaying the word "Pong". There are two functions; one for each player's score.

unsigned int round_off

There is no built-in function for rounding an integer which is usually based on the first decimal value. Therefore, a custom function was developed for rounding the integer value if the first decimal place had a value that was either higher or lower than 0.5. This is used to help with calculating the y-coordinate of the ball during movement.

void Display_Seven_Segment

This function contains a list of case statements for HEX0-5. This is implemented so as to make the process of displaying numbers and characters much easier. Cases 0-15 are used for the regular 0-9 and A-F characters, while case 16 has been assigned custom characters such as P, o, n and g.

Figure 11. Seven segment displaying the score and name of the game

3.5. Audio

An 8-bit sound effect is generated for the pong-game to mimic an actual arcade machine via a numerically controlled oscillator, digital chirping controller and AM modulator. The sound is generated by an FPGA and recorded into non-volatile RAM on an audio ASIC. A TTL control signal is sent to the ASIC whenever sound effect is needed. This architecture (using an audio ASIC) drastically reduces the gaming ARM's unpredictable load, which gives us a constant frames-per-second running speed and a smooth gaming experience.

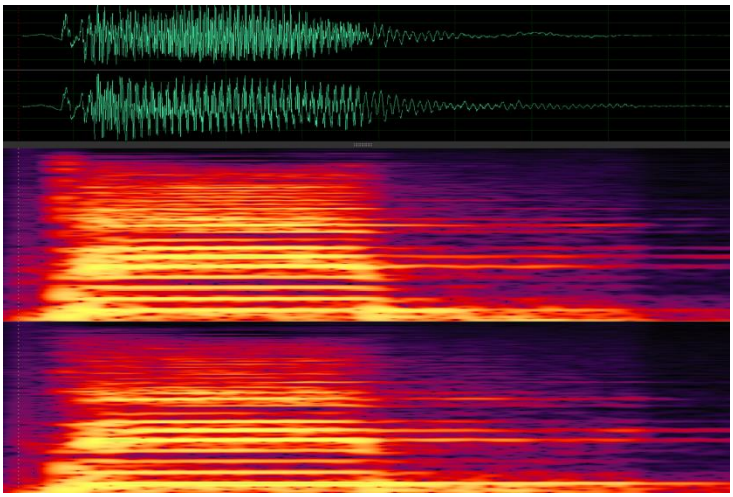

Figure 12. Sample of audio effect in both time and frequency domain.

4. Hardware Implementation

The setup of the entire game consists of two DE1-SoC boards and several hardware peripheries. The first board comprises of an LCD screen with the touch feature implemented. This LCD screen is split into two parts, one shows the display of a .jpg image and the other part of the screen shows a "START" and "STOP" button for starting and stopping game. This DE1-SoC board is also responsible in handling the inputs of a camera system which is able to track any red coloured object i.e. an actual table tennis racquet. The movement of this red object is used to control the movement of one of the paddles during the game. An additional screen is also attached via the PS2 port to display the general rules of the Pong game and input controller instructions on how to play the game. On the second DE1-SoC board, another player is able to manipulate the other paddle via the keys/buttons on the DE1-SoC board, specifically KEY0 and KEY1. The score of the game is also displayed on the seven-segment display to allow the users to know their points tally while playing the game. The players can also adjust the speed at which the ball travels by setting the slide-switches from low to high i.e. the more switches that are set high, the faster the ball travels and the more difficult the gameplay. It is on the second DE1-SoC's LCD screen that the actual gameplay can be seen. There is also a speaker that is attached to the SERVO pins of this LCD board to make a sound every time the ball hits the wall or the paddle. Figures 13 and 14, seen below, show the flowchart of the hardware implementation as well as the final picture of the entire set-up of the game.

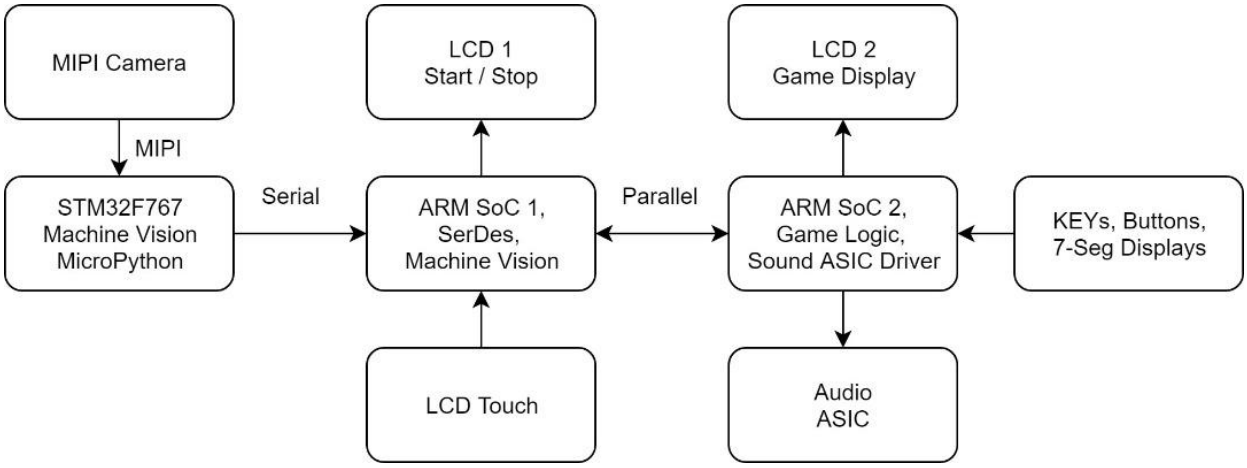

Figure 13. Hardware Implementation Flowchart

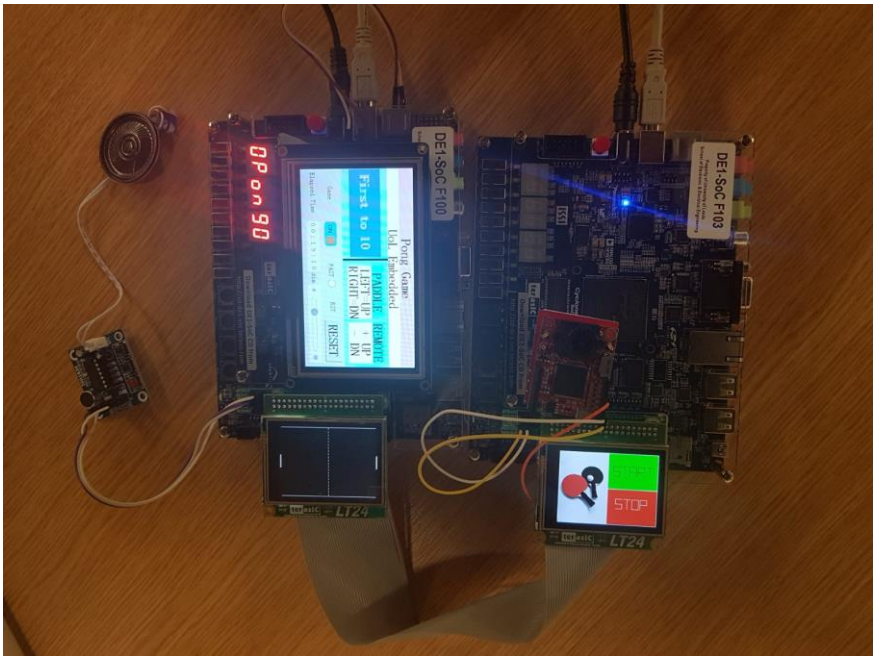

Figure 14. Final assembly and testing of the Pong project

5. OPTIMIZATIONS

The gameplay consists of multiple mathematical (trigonometric, addition, division and multiplication) instructions that include floating point numbers. Since, performing floating point calculations using software is instruction expensive, a hardware vector floating point unit is implemented. Furthermore, the game makes use of large number of instructions and this introduces time lag in the software flow. To make the game efficient, optimisations are done when compiling the source files. After floating point and compiler optimisations, the performance of the game improved as evident in the faster movement of the ball as compared to no optimisations.

```
================================================================================

    Total RO  Size (Code + RO Data)                19344 (  18.89kB)
    Total RW  Size (RW Data + ZI Data)         536871760 (524288.83kB)
    Total ROM Size (Code + RO Data + RW Data)      19868 (  19.40kB)

================================================================================
```

Figure 15. Total RO size without default compiler optimisations.

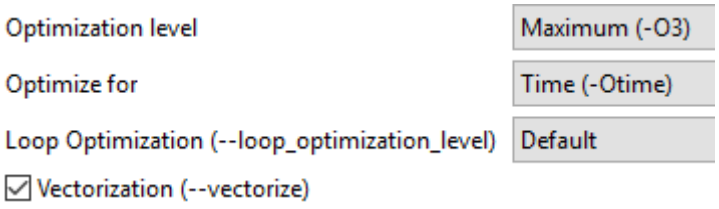

Figure 16. Maximum compiler level and time optimisations.

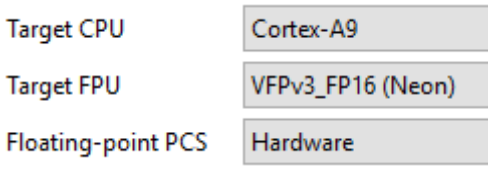

Figure 17. Neon Engine – Vector floating point

```
================================================================================

    Total RO  Size (Code + RO Data)                17368 (  16.96kB)
    Total RW  Size (RW Data + ZI Data)         536871756 (524288.82kB)
    Total ROM Size (Code + RO Data + RW Data)      17888 (  17.47kB)

================================================================================
```

Figure 18. Total RO size with maximum compiler level and time optimisations.

6. CONCLUSION

The project successfully demonstrates that all objectives have been met and this is reflected in Figure 14 that shows the final working hardware implementation. The hardware has been rigorously tested with different levels of optimisations to ensure an efficient and smooth game-play.

7. REFERENCES

[1]. Rinky Dinky Electronics, Image Converter. [Online] 2019. [Accessed 13 May 2019]. Available from: http://www.rinkydinkelectronics.com/t_imageconverter565.php

[2]. Lan-ying, W. A. N. G. (2012). Porting and Implementation of ucGUI Based on STM32 Embedded System [J]. Journal of Sichuan University of Science & Engineering (Natural Science Edition), 1.

[3]. Ding, L., Song, Z., Xu, M., & Xu, C. (2013). Design of embedded measurement and control system based on STM32. Journal of Central South University (Science and Technology), 44, 260-265.

APPENDIX – A

Appendix - A consists of all the .c files used in the project.

SECTION 1: pong.h file that is used for displaying the image. (pong.c is 2000 lines long)

```
/*
 * pong.h
 *
 * An image of the DE1-SoC Board.
 */

#ifndef PONG_H_
#define PONG_H_

extern const unsigned short pong[76800];

#endif /* PONG_H_ */
```

SECTION 2: main.c file of the slave pong board.

```
////////////////////////////////////////////////////////////////////////////
/*
 *  Program for the slave board of PONG.
 *  -------------------------------------
 *  File Name     : main.c
 *  Target Device : ARM Cortex-A9 processor
 *
 *  Created on    : April 23, 2019
 *  Code Author(s): Nuo, Sanjith Chandran & Shrajan Bhandary
 *  Location      : University of Leeds
 *  Module              : ELEC5620M Embedded Microprocessor System Design
 *
 *  Description of the file:
 *      The driver is used to control different aspects of the game such as
 *       read the player's input using the camera and touch screen.
 */
////////////////////////////////////////////////////////////////////////////

#include "Graphics_Engine/Graphics_Engine.h"
       // Importing the graphics engine library for drawing geometrical shapes.
#include "DE1SoC_LT24/DE1SoC_LT24.h"
              // Importing the LCD library for interfacing with the LT24 terasic LCD.
#include "HPS_Watchdog/HPS_Watchdog.h"
                      // Importing the watchdog timer library to ensure the program doesn't
stay in an infinite loop.
#include "HPS_usleep/HPS_usleep.h"
                      // Importing the sleep library to create instances of time delay.
#include <stdlib.h>
                                      // Importing the standard library functions to perform
necessary actions.
#include <time.h>
                                      // Importing the time library so that its values can be
used as seed value for generating random numbers.
#include "pong.h"
                                      // Importing the pong library that has the bit map of the
image to be displayed.

void exitOnFail(signed int status, signed int successStatus)                            //
Exit on fail sub-routine is used to ensure that the processor doesn't malfunction.
{
    if (status != successStatus)
    {
        exit((int)status);
                            // Add breakpoint here to catch failure
    }
}


/* Main Function. */
int main(void)
{
```

```
        /* The timer of the ARM processor is used to the detect the PWM signals from the camera
module. */
        /* ARM A9 Private Timer Load. */
        volatile unsigned int *private_timer_load = (unsigned int *) 0xFFFEC600;

        /* ARM A9 Private Timer Value. */
    volatile unsigned int *private_timer_value = (unsigned int *) 0xFFFEC604;

        /* ARM A9 Private Timer Control. */
    volatile unsigned int *private_timer_control = (unsigned int *) 0xFFFEC608;

        /* ARM A9 Private Timer Interrupt. */
    volatile unsigned int *private_timer_interrupt = (unsigned int *) 0xFFFEC60C;

        ////////////////////////////////////////////////////////////////////////////

        /* The touch screen is used to start or stop the game. */
        /* Touch screen Port Base Address. */
        volatile unsigned int *TOUCH_SCREEN_ptr  = (unsigned int *) 0xFF200060;

        /* The GPIO port is used to communicate between the master board and slave board. */
        /* GPIO Port Base Address. */
        volatile unsigned char *GPIO_ptr  = (unsigned char *) 0xFF200070;

        /* The LEDs on the slave pong board are diplay the PWM signal from the camera and are
used for debugging purpose. */
        /* LED Base Address. */
        volatile unsigned int *LED_ptr  = (unsigned int *) 0xFF200000;

        /* Variables to detect rising and falling edge of the PWM signal. */
        /* The value of the GPIO input. */
        unsigned int current GPIO value = 0;
        unsigned int previous_GPIO_value = 0;

        /* Variables to determine the ON time of the PWM signal. */
        /* Hold Initial Value, Final Value */
    unsigned int initial_count = 0;
        unsigned int final_count = 0;

        /* Variable to hold the value of the touch screen. */
        /* Touch screen value. */
        unsigned int touch_screen_value = 0;

        /* Start or Stop mode of the game that is determined by the touch screen.
                mode = 0 => stop mode and mode = 0 => start mode*/
        int mode = 0;

        /* Initialise the LCD Display and exit if not successful. */
    exitOnFail( LT24_initialise(0xFF200060,0xFF200080),  LT24_SUCCESS); HPS_ResetWatchdog();
        exitOnFail( LT24_copyFrameBuffer(pong,10,160,220,150), LT24_SUCCESS);
HPS_ResetWatchdog();

        ////////////////////////////////////////////////////////////////////////////////////////
///
        //////////////////////////  DRAW THE MAIN SCREEN OF THE GAME
//////////////////////////////
        ////////////////////////////////////////////////////////////////////////////////////////
///

        Graphics_drawBox(10,10,230,310,LT24_WHITE,true,0x39E7); ResetWDT();

        //START
        Graphics_drawBox(10,10,120,160,LT24_WHITE,false,LT24_GREEN); ResetWDT();
        //STOP
        Graphics_drawBox(120,10,230,160,LT24_WHITE,false,LT24_RED); ResetWDT();

        //S=1
        Graphics_drawLetter(50,125,80,145,1,LT24_BLACK); ResetWDT();
        //T=2
        Graphics_drawLetter(50,100,80,120,2,LT24_BLACK); ResetWDT();
        //A=3
        Graphics_drawLetter(50,75,80,95,3,LT24_BLACK); ResetWDT();
        //R=4
        Graphics_drawLetter(50,50,80,70,4,LT24_BLACK); ResetWDT();
        //T=2
        Graphics_drawLetter(50,25,80,45,2,LT24_BLACK); ResetWDT();
```

```
        //S=1
        Graphics_drawLetter(150,115,180,135,1,LT24_WHITE); ResetWDT();
        //T=2
        Graphics_drawLetter(150,90,180,110,2,LT24_WHITE); ResetWDT();
        //O=5
        Graphics_drawLetter(150,65,180,85,5,LT24_WHITE); ResetWDT();
        //P=6
        Graphics_drawLetter(150,40,180,60,6,LT24_WHITE); ResetWDT();

        ////////////////////////////////////////////////////////////////////////////

        /* Loading the timer load with a value. */
        *private_timer_load = 100000000;

        /* Setting the first pin as Input and the remaining as output. */
        *(GPIO_ptr+4)  = 0xFFFFFFFE;

        /* Checking if the touch screen has been tapped. */
        touch_screen_value = *TOUCH_SCREEN_ptr & 0x20000000;

        /* Don't start the game until the touch screen has been tapped. */
        while ( touch_screen_value != 0 )
        {
                touch_screen_value = *TOUCH_SCREEN_ptr & 0x20000000;
                // Keep monitoring the touch screen for taps.
                *GPIO_ptr = 0;
                                                        // Inform the current mode to the master
board.
                HPS_ResetWatchdog();
                                                // Reset the watch dog timer periodically.
        }

        /* Mode = start. */
        mode = mode ^ 0x1;
                                                        // Change mode to start.

        *GPIO_ptr = 0xFF;
                                                        // Inform the master board that the mode
is start game.
        usleep(5000);
                                                        // Do nothing for 5 ms so that the master
board has received the data.

    /* Main Run Loop. */
    while (1)
        {
                touch_screen_value = *TOUCH_SCREEN_ptr & 0x20000000;
                // Keep monitoring the touch screen for taps.
                if ( touch_screen_value == 0 )
                                                // Change modes after each tap of the touch
screen.
                {
                        /* Invert the mode. */
                        mode = mode ^ 0x1;
                                                        // Change modes to stop or start.
                        usleep(50000);

                }

                /* In start mode the slave pong board must communicate with the camera module
and retrieve the location of the paddle. */
                if ( mode == 1 )
                {
                        current_GPIO_value = *GPIO_ptr & 0x1;
                                // Measure the signal from the camera module.

                        if ( current_GPIO_value == 1 && previous_GPIO_value == 0 )
                // Check for rising edge of the PWM signal.
                        {
                                initial_count = *private_timer_value;

                                *private_timer_control = (0 << 8) | (0 << 2) | (1 << 1) | (1 <<
0);     // Start the timer so that the pulse width can be measured.

                                while ( current_GPIO_value == 1 )

                                {
```

```
                                        current_GPIO_value = *GPIO_ptr & 0x1;

                                        HPS_ResetWatchdog();

                                        if ( current_GPIO_value == 0 )
                                        {
                                                *private_timer_control = (0 << 8) | (0 << 2) | (1
<< 1) | (0 << 0);
                                                final_count = initial_count -
*private_timer_value;
                                                *GPIO_ptr = final_count / 100;
                                                // The pulse width conveying the location of the
paddle is sent to the master board as 8-bit parallel signal.
                                                *LED_ptr = final_count / 100;
                                                HPS_ResetWatchdog();
                                        }

                                        previous_GPIO_value = current_GPIO_value;

                                }
                        }

                        previous_GPIO_value = current_GPIO_value;
                }

                else if ( mode == 0 )
                                                // If the mode is stop notify the master pong
board about the same.
                {
                        *GPIO_ptr = 0x00000000;
                }

                /* Finally reset the watchdog. */
        HPS_ResetWatchdog();
    }
}
```

SECTION 3: Graphics_Engine.h is used for drawing shapes on the LCD screen

```
////////////////////////////////////////////////////////////////////////////
/*
 *  Driver for LT24 Display Graphics Engine
 *  ---------------------------------------
 *  File Name     : Graphics_Engine.h
 *  Target Device : ARM Cortex-A9 processor
 *
 *  Created on    : March 15, 2019
 *  Code Author   : S. Bhandary
 *  Location     : University of Leeds
 *  Module               : ELEC5620M Embedded Microprocessor System Design
 *
 *  Description of the file:
 *      The driver is used to draw different geometrical shapes on the LT24
 *      display. This driver imports other libraries to access hardware
 *      resources and communicate with them.
 */
////////////////////////////////////////////////////////////////////////////

#ifndef GRAPHICS_ENGINE_H_
#define GRAPHICS_ENGINE_H_

/* Importing the required libraries. */
/* Boolean variable type "bool" and "true"/"false" constants. */
#include <stdbool.h>


/* Error Codes*/
#define GE_SUCCESS       0

/* Creating a function that returns maximum and minimum of two numbers. */
#define MAX(number_1, number_2) (((number_1) > (number_2)) ? (number_1) : (number_2))
#define MIN(number_1, number_2) (((number_1) < (number_2)) ? (number_1) : (number_2))

////////////////////////////////////// DECLARATION OF FUNCTIONS
//////////////////////////////////////
```

```
/* Function to draw a box with two coordinates: top-left (x1,y1) and bottom_right (x2,y2) with
a boundary colour.
   The function has a boolean argument that decides whether the box should be filled or not
with the colour as that of the boundary.*/
signed int Graphics_drawBox ( unsigned int x1, unsigned int y1, unsigned int x2, unsigned int
y2, unsigned short colour, bool noFill, unsigned short fillColour );

/* Function to draw a circle with two centre: (x,y) and radius with a boundary colour.
   The function has a boolean argument that decides whether the circle should be filled or not
with with a particular colour mentioned.*/
signed int Graphics_drawCircle ( unsigned int x, unsigned int y, unsigned int r, unsigned short
colour, bool noFill, unsigned short fillColour );

/* Graphic function to draw a straight line. Returns 0 if successful .*/
signed int Graphics_drawLine ( unsigned int x1, unsigned int y1, unsigned int x2, unsigned int
y2, unsigned short colour );

/* Function to draw a triangle with three vertices: (x1,y1), (x2,y2) and (x3,y3) with a
boundary colour.
   The function has a boolean argument that decides whether the triangle should be filled or
not with with a particular colour mentioned.*/
signed int Graphics_drawTriangle( unsigned int x1, unsigned int y1, unsigned int x2, unsigned
int y2, unsigned int x3, unsigned int y3, unsigned short colour, bool noFill, unsigned short
fillColour );

/* Function to initialize the LCD. */
void Graphics_initialise( unsigned int lcd_pio_base, unsigned int lcd_hw_base );

/* Graphic function to determine whether a coordinate lies inside the triangle to fill
triangle.*/
bool Graphics_Engine_Point_Test ( unsigned int point_x, unsigned int point_y, unsigned int x1,
unsigned int y1, unsigned int x2, unsigned int y2, unsigned int x3, unsigned int y3);

/* Function to swap axes i.e., swap x and y points */
unsigned int* Graphics_Engine_Swap_Vertices ( unsigned int point_x, unsigned int point_y );

/* Graphic function to draw letters on the LCD screen. */
signed int Graphics drawLetter(unsigned int x1, unsigned int y1, unsigned int x2,  unsigned int
y2, unsigned int score, unsigned short colour);

#endif /* GRAPHICS_ENGINE_H_ */
```

SECTION 4: Game_Engine.h is used for controls of the game.

```
////////////////////////////////////////////////////////////////////////////////
/*
 *  Game Engine for PONG.
 *  ---------------------------------------
 *  File Name     : Game_Engine.h
 *  Target Device : ARM Cortex-A9 processor
 *
 *  Created on    : April 23, 2019
 *  Code Author(s): Nuo, Sanjith Chandran & Shrajan Bhandary
 *  Location      : University of Leeds
 *  Module              : ELEC5620M Embedded Microprocessor System Design
 *
 *  Description of the file:
 *      The driver is used to control different aspects of the game such as
 *      read the player's input, manipulate the position of the paddle,
 *             movement of the ball, players' scores and game-play of PONG. This
 *             driver imports other libraries to access hardware resources and
 *             communicate with them.
 */
////////////////////////////////////////////////////////////////////////////////

#ifndef GAME_ENGINE_H_
#define GAME_ENGINE_H_

/* Importing the required libraries. */
/* Boolean variable type "bool" and "true"/"false" constants. */
#include <stdbool.h>


/* Error Codes*/
#define GAME_SUCCESS       0
```

```
/* Creating a function that returns maximum and minimum of two numbers. */
#define MAX(number_1, number_2) (((number_1) > (number_2)) ? (number_1) : (number_2))
#define MIN(number_1, number_2) (((number_1) < (number_2)) ? (number_1) : (number_2))

////////////////////////////////////////// INITIAL CONSTANT POSITIONS
//////////////////////////////////////

/* A paddle is a rectangle with length 20 pixels and width 5 width. Therefore, it has a right
top coordinate and left-bottom coordinate. */

/* Constant paddle dimensions. */
#define PADDLE_LENGTH 40
#define PADDLE_WIDTH  2

#define PADDLE_1_CENTER_X     119
#define PADDLE_2_CENTER_X     119

#define MINIMUM_PADDLE_X 11
#define MAXIMUM_PADDLE_X 229

#define PADDLE_INCREMENT 1
#define PADDLE_DECREMENT 1

/* Paddle 1 (x1,y1) point. */
#define PADDLE_1_START_X_1    PADDLE_1_CENTER_X - ( PADDLE_LENGTH / 2 )
#define PADDLE_1_START_Y_1    7

/* Paddle 1 (x2,y2) point. */
#define PADDLE_1_START_X_2    PADDLE_1_CENTER_X + ( PADDLE_LENGTH / 2 )
#define PADDLE_1_START_Y_2    9

/* Paddle 2 (x1,y1) point. */
#define PADDLE_2_START_X_1    PADDLE_2_CENTER_X - ( PADDLE_LENGTH / 2 )
#define PADDLE_2_START_Y_1    311

/* Paddle 2 (x2,y2) point. */
#define PADDLE_2_START_X_2    PADDLE_2_CENTER_X + ( PADDLE_LENGTH / 2 )
#define PADDLE_2_START_Y_2    313

/* The colour of the background. */
#define GAME_BACKGROUND_GREY (0x39E7)

////////////////////////////////////////// DECLARATION OF FUNCTIONS
//////////////////////////////////////

/* Function to initialize PADDLE 1 with two coordinates: top-left (x1,y1) and bottom_right
(x2,y2) with a boundary colour. */
signed int Paddle_1_Initialize ( void );

/* Function to initialize PADDLE 2 with two coordinates: top-left (x1,y1) and bottom_right
(x2,y2) with a boundary colour. */
signed int Paddle_2_Initialize ( void );

/* Function to draw PADDLE 1 with two coordinates: top-left (x,y) and fixed paddle dimensions
with a boundary colour. */
signed int Paddle_1_Draw ( unsigned int PADDLE_1_X, unsigned int PADDLE_1_Y );

/* Function to draw PADDLE 2 with two coordinates: top-left (x,y) and fixed paddle dimensions
with a boundary colour. */
signed int Paddle_2_Draw ( unsigned int PADDLE_2_X, unsigned int PADDLE_2_Y );

/* Function to draw the dashed line with points: (x1,y1) and (x2,y2) with a boundary colour. */
signed int Graphics_drawDash(unsigned int x1, unsigned int y1, unsigned int x2,  unsigned int
y2, unsigned short colour);

/* Function to move paddle 1 based on the buttons. */
signed int Move_Paddle_1 ( void );

/* Function to move paddle 2 based on the buttons. */
signed int Move_Paddle_2 ( void );

/* Function to draw ball on the screen. */
signed int Graphics_drawBall(unsigned int x1, unsigned int y1, unsigned int x2,  unsigned int
y2, unsigned short colour);

/* Function to erase ball from the screen. */
```

```
signed int Graphics_Erase(unsigned int x1, unsigned int y1, unsigned int x2,  unsigned int y2,
unsigned short fillColour);

/* Function to move the ball around the screen. */
signed int Move_Ball ( void );

/* Round off function. */
unsigned int round_off(float num);

/* Seven Segment displays. */
void Display_Seven_Segment(int DISPLAY_NUMBER, int HEX_NUMBER);

/* Function to update scores of players. */
void Update_Score_1 ( void );
void Update_Score_2 ( void );

#endif /* GAME_ENGINE_H_ */
```

SECTION 5: Game_Engine.c is used for controls of the game.

```
////////////////////////////////////////////////////////////////////////////
/*
 *  Game Engine for PONG.
 *  ---------------------------------------
 *  File Name     : Game_Engine.c
 *  Target Device : ARM Cortex-A9 processor
 *
 *  Created on    : April 23, 2019
 *  Code Author(s): Nuo, Sanjith Chandran & Shrajan Bhandary
 *  Location      : University of Leeds
 *  Module              : ELEC5620M Embedded Microprocessor System Design
 *
 *  Description of the file:
 *      The driver is used to control different aspects of the game such as
 *       read the player's input, manipulate the position of the paddle,
 *             movement of the ball, players' scores and game-play of PONG.
 */
////////////////////////////////////////////////////////////////////////////

#include "Game_Engine.h"
                              // Invoking the main header file.
#include "../Graphics Engine/Graphics Engine.h"
               // Importing the Graphics Engine driver library.
#include "../DE1Soc_LT24/DE1SoC_LT24.h"
                        // Importing the Leeds SoC LT24 driver controller.
#include "../HPS_Watchdog/HPS_Watchdog.h"
               // Importing the Watch dog timer driver controller.
#include "../HPS usleep/HPS usleep.h"
               // Importing the sleep function.
#include "math.h"
                                      // Importing the mathematical library of C.
#include <time.h>
                                      // Importing the time library so that its values can be
used as seed value for generating random numbers.

/* Global variables to hold the current position of paddle 1. */
unsigned current_paddle_1_x_1;
unsigned current_paddle_1_y_1;
unsigned current_paddle_1_x_2;
unsigned current_paddle_1_y_2;

/* Global variables to hold the previous position of paddle 1. */
unsigned previous_paddle_1_x_1;
unsigned previous_paddle_1_y_1;
unsigned previous_paddle_1_x_2;
unsigned previous_paddle_1_y_2;

/* Global variables to hold the current position of paddle 2. */
unsigned current_paddle_2_x_1;
unsigned current_paddle_2_y_1;
unsigned current_paddle_2_x_2;
unsigned current_paddle_2_y_2;

/* Global variables to hold the previous position of paddle 2. */
unsigned previous_paddle_2_x_1;
unsigned previous_paddle_2_y_1;
```

```
unsigned previous_paddle_2_x_2;
unsigned previous_paddle_2_y_2;

/* Global variables to move the paddles. */
unsigned move_paddle_1_x = PADDLE_1_START_X_1;
unsigned move_paddle_2_x = PADDLE_2_START_X_1;

/* Move ball variables. */
#define PI 3.14

signed int quadrant = 1;
signed int inherent_delay = 5000;

unsigned int x1 = 120 ;
unsigned int y1 = 160;

unsigned int x2 = 120;
unsigned int y2 = 160;

unsigned int angle = 60;

unsigned short colour;
unsigned short fillColour;

unsigned int GPIO_value;

/* Base address of the seven segment displays. */
volatile char *HEX_1  = (char *) 0xFF200020;
volatile char *HEX_2  = (char *) 0xFF200021;
volatile char *HEX_3  = (char *) 0xFF200022;
volatile char *HEX_4  = (char *) 0xFF200023;
volatile char *HEX_5  = (char *) 0xFF200030;
volatile char *HEX_6  = (char *) 0xFF200031;

/* Flags for the points. */
bool ball_touch_paddle = true;

/* Variables to store players' scores. */
int player_1_score = 0;
int player_2_score = 0;

/* Function to initialize PADDLE 1 with two coordinates: top-left (x1,y1) and bottom_right
(x2,y2) with a boundary colour. */
signed int Paddle_1_Initialize ( void )
{
	/* Declaring the local variables required for drawing a line */
	signed int Game_status = 0;
                            // Status variable to check for errors.

	current_paddle_1_x_1 = PADDLE_1_START_X_1;
	current_paddle_1_y_1 = PADDLE_1_START_Y_1;
	current_paddle_1_x_2 = PADDLE_1_START_X_2;
	current_paddle_1_y_2 = PADDLE_1_START_Y_2;

	Game_status = Graphics_drawBox ( current_paddle_1_x_1, current_paddle_1_y_1,
current_paddle_1_x_2, current_paddle_1_y_2, LT24_WHITE, false, LT24_WHITE ); // Draw the
paddle.
	if ( Game_status != GAME_SUCCESS) return Game_status;
	// If there is an error, report it.

	/* The current value of the paddle becomes previous value in the next iteration. */
	previous_paddle_1_x_1 = current_paddle_1_x_1;
	previous_paddle_1_y_1 = current_paddle_1_y_1;
	previous_paddle_1_x_2 = current_paddle_1_x_2;
	previous_paddle_1_y_2 = current_paddle_1_y_2;

	return GAME_SUCCESS;
                            // Return the completion of the function.
}

/* Function to initialize PADDLE 2 with two coordinates: top-left (x1,y1) and bottom_right
(x2,y2) with a boundary colour. */
signed int Paddle_2_Initialize ( void )
{
	/* Declaring the local variables required for drawing a line */
	signed int Game_status = 0;
                            // Status variable to check for errors.
```

```
        current_paddle_2_x_1 = PADDLE_2_START_X_1;
        current_paddle_2_y_1 = PADDLE_2_START_Y_1;
        current_paddle_2_x_2 = PADDLE_2_START_X_2;
        current_paddle_2_y_2 = PADDLE_2_START_Y_2;

        Game_status = Graphics_drawBox ( current_paddle_2_x_1, current_paddle_2_y_1,
current_paddle_2_x_2, current_paddle_2_y_2, LT24_WHITE, false, LT24_WHITE ); // Draw the
paddle.
        if ( Game_status != GAME_SUCCESS) return Game_status;
        // If there is an error, report it.

        /* The current value of the paddle becomes previous value in the next iteration. */
        previous_paddle_2_x_1 = current_paddle_2_x_1;
        previous_paddle_2_y_1 = current_paddle_2_y_1;
        previous_paddle_2_x_2 = current_paddle_2_x_2;
        previous_paddle_2_y_2 = current_paddle_2_y_2;

        return GAME_SUCCESS;
                                // Return the completion of the function.
}

/* Function to draw PADDLE 1 with two coordinates: top-left (x,y) and fixed paddle dimensions
with a boundary colour. */
signed int Paddle_1_Draw ( unsigned int PADDLE_1_X, unsigned int PADDLE_1_Y )
{
        /* Declaring the local variables required for drawing a line */
        signed int Game_status = 0;
                                // Status variable to check for errors.

        current_paddle_1_x_1 = PADDLE_1_X;
        current paddle 1 y 1 = PADDLE 1 Y;
        current_paddle_1_x_2 = PADDLE_1_X + PADDLE_LENGTH;
        current_paddle_1_y_2 = PADDLE_1_Y + PADDLE_WIDTH ;

        Game_status = Graphics_drawBox ( previous_paddle_1_x_1, previous_paddle_1_y_1,
previous_paddle_1_x_2, previous_paddle_1_y_2,LT24_BLACK, false, LT24_BLACK); // Erase the
paddle.
        if ( Game_status != GAME_SUCCESS) return Game_status;
        // If there is an error, report it.

        Game_status = Graphics_drawBox ( current_paddle_1_x_1, current_paddle_1_y_1,
current_paddle_1_x_2, current_paddle_1_y_2, LT24_WHITE, false, LT24_WHITE ); // Draw the
paddle.
        if ( Game_status != GAME_SUCCESS) return Game_status;
        // If there is an error, report it.

        /* The current value of the paddle becomes previous value in the next iteration. */
        previous_paddle_1_x_1 = current_paddle_1_x_1;
        previous_paddle_1_y_1 = current_paddle_1_y_1;
        previous_paddle_1_x_2 = current_paddle_1_x_2;
        previous_paddle_1_y_2 = current_paddle_1_y_2;

        return GAME_SUCCESS;
                                // Return the completion of the function.
}

/* Function to draw PADDLE 2 with two coordinates: top-left (x,y) and fixed paddle dimensions
with a boundary colour. */
signed int Paddle_2_Draw ( unsigned int PADDLE_2_X, unsigned int PADDLE_2_Y )
{
        /* Declaring the local variables required for drawing a line */
        signed int Game_status = 0;
                                // Status variable to check for errors.

        current_paddle_2_x_1 = PADDLE_2_X;
        current_paddle_2_y_1 = PADDLE_2_Y;
        current_paddle_2_x_2 = PADDLE_2_X + PADDLE_LENGTH;
        current_paddle_2_y_2 = PADDLE_2_Y + PADDLE_WIDTH ;

        Game_status = Graphics_drawBox ( previous_paddle_2_x_1, previous_paddle_2_y_1,
previous_paddle_2_x_2, previous_paddle_2_y_2, LT24_BLACK, false, LT24_BLACK); // Erase the
paddle.
        if ( Game_status != GAME_SUCCESS) return Game_status;
        // If there is an error, report it.
```

```
        Game_status = Graphics_drawBox ( current_paddle_2_x_1, current_paddle_2_y_1,
current_paddle_2_x_2, current_paddle_2_y_2, LT24_WHITE, false, LT24_WHITE ); // Draw the
paddle.
        if ( Game_status != GAME_SUCCESS) return Game_status;
        // If there is an error, report it.

        /* The current value of the paddle becomes previous value in the next iteration. */
        previous_paddle_2_x_1 = current_paddle_2_x_1;
        previous_paddle_2_y_1 = current_paddle_2_y_1;
        previous_paddle_2_x_2 = current_paddle_2_x_2;
        previous_paddle_2_y_2 = current_paddle_2_y_2;

        return GAME_SUCCESS;
                                // Return the completion of the function.
}

/* Function to draw the dashed line with points: (x1,y1) and (x2,y2) with a boundary colour. */
signed int Graphics_drawDash(unsigned int x1, unsigned int y1, unsigned int x2,  unsigned int
y2, unsigned short colour)
{
        /* Declaring the local variables required for drawing a line */
        signed int Game_status = 0;
                                // Status variable to check for errors.
        signed int count;

        for ( count = x1+1 ; count <= x2-1 ; count+=10 )
        {
                if ( count < x2-5 )
                {
                        Game_status = Graphics_drawLine(count,y1,count+5,y2,colour);
                        if ( Game_status != GAME_SUCCESS) return Game_status;
                        // If there is an error, report it.
                }
        }
        return GAME_SUCCESS;
}

/* Function to move paddle based on the camera. */
signed int Move_Paddle_1 ( void )
{
        /* Declaring the local variables required for drawing a line */
        signed int Game_status = 0;
                                // Status variable to check for errors.

        /* GPIO Port Base Address. */
        volatile unsigned char *GPIO_ptr  = (unsigned char *) 0xFF200070;

        /* LED Base Address. */
        volatile unsigned int *LED_ptr  = (unsigned int *) 0xFF200000;

        /* Variable to hold the value from the slave board */
        unsigned slave_board_value;

        /* Setting all pin as Input. */
        *(GPIO_ptr+4)  = 0x00000001;

        slave_board_value = *GPIO_ptr & 0xFFF;

        move_paddle_1_x = ( slave_board_value * 5 ) / 4;

        if ( move_paddle_1_x <= MINIMUM_PADDLE_X )
        {
                move_paddle_1_x = MINIMUM_PADDLE_X;
        }
        else if ( move_paddle_1_x >= ( MAXIMUM_PADDLE_X - PADDLE_LENGTH ) )
        {
                move_paddle_1_x = MAXIMUM_PADDLE_X - PADDLE_LENGTH;
        }

        //Paddle_1_Draw ( current_paddle_1_x, PADDLE_Y_1 ); HPS_ResetWatchdog();
        Game_status = Paddle_1_Draw ( move_paddle_1_x, PADDLE_1_START_Y_1 );
HPS_ResetWatchdog();
        if ( Game_status != GAME_SUCCESS) return Game_status;
        // If there is an error, report it.

        return GAME_SUCCESS;
}
```

```
/* Function to move paddle based on the buttons. */
signed int Move_Paddle_2 ( void )
{
	/* Declaring the local variables required for drawing a line */
	signed int Game_status = 0;
				// Status variable to check for errors.

	volatile unsigned int *KEY_ptr	= (unsigned int *) 0xFF200050;
	unsigned key_value;

	key_value = *KEY_ptr & 0x3;

	if ( key_value == 1 )
	{

		if ( move_paddle_2_x >= ( MAXIMUM_PADDLE_X - PADDLE_LENGTH ) )
		{
			move_paddle_2_x = MAXIMUM_PADDLE_X - PADDLE_LENGTH;
		}

		else if ( move_paddle_2_x < MAXIMUM_PADDLE_X )
		{
			move_paddle_2_x = move_paddle_2_x + PADDLE_INCREMENT;

			if ( move_paddle_2_x >= ( MAXIMUM_PADDLE_X - PADDLE_LENGTH ) )
			{
				move_paddle_2_x = MAXIMUM_PADDLE_X - PADDLE_LENGTH;
			}
		}

		//Reset watchdog.
		HPS_ResetWatchdog();
	}
	else if ( key_value == 2 )
	{
		if ( move_paddle_2_x <= MINIMUM_PADDLE_X )
		{
			move_paddle_2_x = MINIMUM_PADDLE_X;
		}
		else if ( move_paddle_2_x > MINIMUM_PADDLE_X )
		{
			move_paddle_2_x = move_paddle_2_x - PADDLE_DECREMENT;

			if ( move_paddle_2_x <= MINIMUM_PADDLE_X )
			{
				move_paddle_2_x = MINIMUM_PADDLE_X;
			}
		}

		//Reset watchdog.
		HPS_ResetWatchdog();
	}

	//Paddle_1_Draw ( current_paddle_1_x, PADDLE_Y_1 ); HPS_ResetWatchdog();
	Game_status = Paddle_2_Draw ( move_paddle_2_x, PADDLE_2_START_Y_1 );
HPS_ResetWatchdog();
	if ( Game_status != GAME_SUCCESS) return Game_status;
	// If there is an error, report it.

	return GAME_SUCCESS;
}

/* Function to draw ball on the screen. */
signed int Graphics_drawBall(unsigned int x1, unsigned int y1, unsigned int x2,  unsigned int
y2, unsigned short colour)
{
	signed int i;
	signed int j;
	for(i=y1;i<=y2;i++)
			{
				for(j=x1;j<=x2;j++)
				{
					LT24_drawPixel(colour,j,i);

				}
			}
```

```
        return 0;
}

/* Function to erase ball from the screen. */
signed int Graphics_Erase(unsigned int x1, unsigned int y1, unsigned int x2,  unsigned int y2,
unsigned short fillColour)
{
        signed int i;
        signed int j;
                for(i=y1;i<=y2;i++)
                                {
                                        for(j=x1;j<=x2;j++)
                                        {
                                                LT24_drawPixel(fillColour,j,i);
                                        }
                                }

        return 0;
}

/* Function to move the ball around the screen. */
signed int Move_Ball ( void )
{
        volatile unsigned char *GPIO_ptr = (unsigned char *)0xFF2000C0;
        float hit_angle = angle*(PI/180);

        volatile unsigned int *SW_ptr = (unsigned int *)0xFF200040;
        unsigned int delay = inherent_delay - ( ( *SW_ptr & 0x3FF ) * 3.5) ;

        *(GPIO_ptr+4)   = 0xFF;
        colour = LT24_WHITE;
        fillColour = 0x39E7;

        ball_touch_paddle = true;

        if (quadrant ==1)
        {
                if   (x2>12 && y2>12)
                {

                        *GPIO_ptr = 0x0;
                        x2 = x2-1;
                        y2 = y2 - abs(round_off(1/tan(hit_angle)));
                        Graphics_drawBall(x2,y2,x2+3,y2+3,colour);
                        usleep(delay);
                        Graphics_Erase(x2,y2,x2+3,y2+3,fillColour);
                        Graphics_drawDash(11,160,229,160,LT24_WHITE);
                }

                if (x2 <= 12)
                {
                        *GPIO_ptr = 0xF;
                        usleep(10000);
                        quadrant = 2;
                }
                else if(y2 <= 12)
                {
                        if ( x2 > current_paddle_1_x_1 && x2 < current_paddle_1_x_2 )
                        {
                                *GPIO_ptr = 0xF;
                                usleep(10000);
                                quadrant = 4;
                        }
                        else
                        {
                                Update_Score_2();
                                ball_touch_paddle = false;
                        }
                }

        }

        else if (quadrant ==2)
        {
                if (x2<225 && y2>12)
                {
                        *GPIO_ptr = 0x0;
```

```
                        x2 = x2+1;
                        y2 = y2 - abs(round_off(1/tan(hit_angle)));
                        Graphics_drawBall(x2,y2,x2+3,y2+3,colour);
                        usleep(delay);
                        Graphics_Erase(x2,y2,x2+3,y2+3,fillColour);
                        Graphics_drawDash(11,160,229,160,LT24_WHITE);
                }

                if (x2 >= 225)
                {
                        quadrant = 1;
                        *GPIO_ptr = 0xF;
                        usleep(10000);
                }
                else if(y2 <= 12)
                {
                        if ( x2 > current_paddle_1_x_1 && x2 < current_paddle_1_x_2 )
                        {
                                *GPIO_ptr = 0xF;
                                usleep(10000);
                                quadrant = 3;
                        }
                        else
                        {
                                Update_Score_2();
                                ball_touch_paddle = false;
                        }
                }
        }

        //else if (angle>180 && angle<270)
        else if (quadrant ==3)
        {
                if (x2<225 && y2<305)
                {
                        *GPIO_ptr = 0x0;
                        x2 = x2+1;
                        y2 = y2 + abs(round off(1/tan(hit angle)));
                        Graphics_drawBall(x2,y2,x2+3,y2+3,colour);
                        usleep(delay);
                        Graphics_Erase(x2,y2,x2+3,y2+3,fillColour);
                        Graphics_drawDash(11,160,229,160,LT24_WHITE);
                }

                if (x2 >= 225)
                {

                        quadrant = 4;
                        *GPIO_ptr = 0xF;
                        usleep(10000);
                }
                else if(y2 >= 305)
                {
                        if ( x2 > current_paddle_2_x_1 && x2 < current_paddle_2_x_2 )
                        {
                                *GPIO_ptr = 0xF;
                                usleep(10000);
                                quadrant = 2;
                        }
                        else
                        {
                                Update_Score_1();
                                ball_touch_paddle = false;
                        };
                }
        }
                //else  if (angle>270 && angle<360)
        else if (quadrant == 4)
        {
                if (x2>12 && y2<305)
                {
                        *GPIO_ptr = 0x0;
                        x2 = x2-1;
                        y2 = y2 + abs(round_off(1/tan(hit_angle)));
                        Graphics_drawBall(x2,y2,x2+3,y2+3,colour);
                        usleep(delay);
                        Graphics_Erase(x2,y2,x2+3,y2+3,fillColour);
```

```
                        Graphics_drawDash(11,160,229,160,LT24_WHITE);
                }

                if (x2 <= 12)
                {
                        quadrant = 3;
                        *GPIO_ptr = 0xF;
                        usleep(10000);
                }
                else if(y2 >= 305)
                {
                        if ( x2 > current_paddle_2_x_1 && x2 < current_paddle_2_x_2 )
                        {
                                *GPIO_ptr = 0xF;
                                usleep(10000);
                                quadrant = 1;
                        }
                        else
                        {
                                Update_Score_1();
                                ball_touch_paddle = false;
                        };
                }
        }

        HPS_ResetWatchdog();

        if ( ball_touch_paddle == true )
        {
                x1=x2;
                y1=y2;
        }
}

void Update_Score_1 ( void )
{
        Graphics_drawBox ( 115, 75, 125, 85, LT24_GREEN, false, LT24_GREEN ); // Draw the box.
        Graphics_drawBox ( 115, 235, 125, 245, LT24_RED, false, LT24_RED ); // Draw the box.

        player_1_score = player_1_score + 1;

        Display_Seven_Segment(1,player_1_score);
        Display_Seven_Segment(2,16);
        Display_Seven_Segment(3,16);
        Display_Seven_Segment(4,16);
        Display_Seven_Segment(5,16);
        Display_Seven_Segment(6,player_2_score);

        if ( player_1_score >= 10  )
        {
                Graphics_drawBox ( previous_paddle_1_x_1, previous_paddle_1_y_1,
previous_paddle_1_x_2, previous_paddle_1_y_2, LT24_BLACK, false, LT24_BLACK); // Erase paddle
1.
                Graphics_drawBox ( previous_paddle_2_x_1, previous_paddle_2_y_1,
previous_paddle_2_x_2, previous_paddle_2_y_2, LT24_BLACK, false, LT24_BLACK); // Erase paddle
2.

                while (1)
                {
                        Graphics_drawBox(10,10,230,310,LT24_BLACK,false,LT24_YELLOW);
ResetWDT();
                        Graphics_drawLetter(85,215,115,235,7,LT24_BLACK); ResetWDT();
                        Graphics_drawLetter(85,190,115,210,8,LT24_BLACK); ResetWDT();
                        Graphics_drawLetter(85,165,115,185,9,LT24_BLACK); ResetWDT();
                        Graphics_drawLetter(85,140,115,160,9,LT24_BLACK); ResetWDT();
                        Graphics_drawLetter(85,115,115,135,10,LT24_BLACK); ResetWDT();
                        Graphics_drawLetter(85,90,115,110,4,LT24_BLACK); ResetWDT();
                        Graphics_drawLetter(125,165,155,185,6,LT24_BLACK); ResetWDT();

                        //P1
                        Graphics_drawLetter(125,140,155,160,11,LT24_BLACK); ResetWDT();
                }
        }

        usleep(3000000);
```

```
        Graphics_drawBox ( 115, 75, 125, 85, GAME_BACKGROUND_GREY , false, GAME_BACKGROUND_GREY
); // Erase the box.
        Graphics_drawBox ( 115, 235, 125, 245, GAME_BACKGROUND_GREY , false,
GAME_BACKGROUND_GREY  ); // Draw the box.

        x1 = 120;
        y1 = 160;
        x2 = 120;
        y2 = 160;

        srand(time(0));

        quadrant = ((abs( rand())) % 4 ) + 1;
        angle = ((abs( rand())) % 40 )+ 20;
}

void Update_Score_2 ( void )
{
        Graphics_drawBox ( 115, 75, 125, 85, LT24_RED, false, LT24_RED ); // Draw the box.
        Graphics_drawBox ( 115, 235, 125, 245, LT24_GREEN, false, LT24_GREEN ); // Draw the
box.

        player_2_score = player_2_score + 1;

        Display_Seven_Segment(1,player_1_score);
        Display_Seven_Segment(2,16);
        Display_Seven_Segment(3,16);
        Display_Seven_Segment(4,16);
        Display_Seven_Segment(5,16);
        Display_Seven_Segment(6,player_2_score);

        if ( player_2_score >= 10 )
        {
                Graphics_drawBox ( previous_paddle_1_x_1, previous_paddle_1_y_1,
previous_paddle_1_x_2, previous_paddle_1_y_2, LT24_BLACK, false, LT24_BLACK); // Erase paddle
1.
                Graphics_drawBox ( previous_paddle_2_x_1, previous_paddle_2_y_1,
previous_paddle_2_x_2, previous_paddle_2_y_2, LT24_BLACK, false, LT24_BLACK); // Erase paddle
2.

                while (1)
                {
                        Graphics_drawBox(10,10,230,310,LT24_BLACK,false,LT24_YELLOW);
ResetWDT();
                        Graphics_drawLetter(85,215,115,235,7,LT24_BLACK); ResetWDT();
                        Graphics_drawLetter(85,190,115,210,8,LT24_BLACK); ResetWDT();
                        Graphics_drawLetter(85,165,115,185,9,LT24_BLACK); ResetWDT();
                        Graphics_drawLetter(85,140,115,160,9,LT24_BLACK); ResetWDT();
                        Graphics_drawLetter(85,115,115,135,10,LT24_BLACK); ResetWDT();
                        Graphics_drawLetter(85,90,115,110,4,LT24_BLACK); ResetWDT();
                        Graphics_drawLetter(125,165,155,185,6,LT24_BLACK); ResetWDT();

                        //P2
                        Graphics_drawLetter(125,140,155,160,12,LT24_BLACK); ResetWDT();
                }
        }

        usleep(3000000);

        Graphics_drawBox ( 115, 75, 125, 85, GAME_BACKGROUND_GREY , false, GAME_BACKGROUND_GREY
); // Erase the box.
        Graphics_drawBox ( 115, 235, 125, 245, GAME_BACKGROUND_GREY , false,
GAME_BACKGROUND_GREY  ); // Draw the box.

        x1 = 120;
        y1 = 160;
        x2 = 120;
        y2 = 160;

        srand(time(0));

        quadrant = ((abs( rand())) % 4 ) + 1;
        angle = ((abs( rand())) % 40 )+ 20;

}

/* Round off function that converts float value to integer. */
```

```
unsigned int round_off(float num)
{
	int n = (int)num;

	if((num-n)>0.5)
	{
		return n+1;
	}
	else {
		return n;
	}
}

/* Seven Segment displays. */
void Display_Seven_Segment(int DISPLAY_NUMBER, int HEX_NUMBER)
{
	if ( DISPLAY_NUMBER == 1 )
	{
		switch (HEX_NUMBER)
		{
			case 0: *HEX_1 = 63;break;
			case 1: *HEX_1 = 6;break;
			case 2: *HEX_1 = 91;break;
			case 3: *HEX_1 = 79;break;
			case 4: *HEX_1 = 102;break;
			case 5: *HEX_1 = 109;break;
			case 6: *HEX_1 = 125;break;
			case 7: *HEX_1 = 07;break;
			case 8: *HEX_1 = 127;break;
			case 9: *HEX_1 = 103;break;
			case 10: *HEX_1 = 119;break;
			case 11: *HEX_1 = 124;break;
			case 12: *HEX_1 = 57;break;
			case 13: *HEX_1 = 94;break;
			case 14: *HEX_1 = 121;break;
			case 15: *HEX_1 = 113;break;

		}
	}

	else if ( DISPLAY_NUMBER == 2 )
	{

		switch (HEX_NUMBER)
		{
			case 0: *HEX_2 = 63;break;
			case 1: *HEX_2 = 6;break;
			case 2: *HEX_2 = 91;break;
			case 3: *HEX_2 = 79;break;
			case 4: *HEX_2 = 102;break;
			case 5: *HEX_2 = 109;break;
			case 6: *HEX_2 = 125;break;
			case 7: *HEX_2 = 07;break;
			case 8: *HEX_2 = 127;break;
			case 9: *HEX_2 = 103;break;
			case 10: *HEX_2 = 119;break;
			case 11: *HEX_2 = 124;break;
			case 12: *HEX_2 = 57;break;
			case 13: *HEX_2 = 94;break;
			case 14: *HEX_2 = 121;break;
			case 15: *HEX_2 = 113;break;
			case 16: *HEX_2 = 0x6F; break;
		}
	}

	else if ( DISPLAY_NUMBER == 3 )
	{

		switch (HEX_NUMBER)
		{
			case 0: *HEX_3 = 63;break;
			case 1: *HEX_3 = 6;break;
			case 2: *HEX_3 = 91;break;
			case 3: *HEX_3 = 79;break;
			case 4: *HEX_3 = 102;break;
			case 5: *HEX_3 = 109;break;
			case 6: *HEX_3 = 125;break;
```

```
                        case 7: *HEX_3 = 07;break;
                        case 8: *HEX_3 = 127;break;
                        case 9: *HEX_3 = 103;break;
                        case 10: *HEX_3 = 119;break;
                        case 11: *HEX_3 = 124;break;
                        case 12: *HEX_3 = 57;break;
                        case 13: *HEX_3 = 94;break;
                        case 14: *HEX_3 = 121;break;
                        case 15: *HEX_3 = 113;break;
                        case 16: *HEX_3 = 0x54; break;
                }
        }

        else if ( DISPLAY_NUMBER == 4 )
        {

                switch (HEX_NUMBER)
                {
                        case 0: *HEX_4 = 63;break;
                        case 1: *HEX_4 = 6;break;
                        case 2: *HEX_4 = 91;break;
                        case 3: *HEX_4 = 79;break;
                        case 4: *HEX_4 = 102;break;
                        case 5: *HEX_4 = 109;break;
                        case 6: *HEX_4 = 125;break;
                        case 7: *HEX_4 = 07;break;
                        case 8: *HEX_4 = 127;break;
                        case 9: *HEX_4 = 103;break;
                        case 10: *HEX_4 = 119;break;
                        case 11: *HEX_4 = 124;break;
                        case 12: *HEX_4 = 57;break;
                        case 13: *HEX 4 = 94;break;
                        case 14: *HEX_4 = 121;break;
                        case 15: *HEX_4 = 113;break;
                        case 16: *HEX_4 =0x5C; break;
                }
        }

        else if ( DISPLAY_NUMBER == 5 )
        {

                switch (HEX_NUMBER)
                {
                        case 0: *HEX_5 = 63;break;
                        case 1: *HEX_5 = 6;break;
                        case 2: *HEX_5 = 91;break;
                        case 3: *HEX_5 = 79;break;
                        case 4: *HEX_5 = 102;break;
                        case 5: *HEX_5 = 109;break;
                        case 6: *HEX_5 = 125;break;
                        case 7: *HEX_5 = 07;break;
                        case 8: *HEX_5 = 127;break;
                        case 9: *HEX_5 = 103;break;
                        case 10: *HEX_5 = 119;break;
                        case 11: *HEX_5 = 124;break;
                        case 12: *HEX_5 = 57;break;
                        case 13: *HEX_5 = 94;break;
                        case 14: *HEX_5 = 121;break;
                        case 15: *HEX_5 = 113;break;
                        case 16: *HEX_5 = 0x73; break;

                }
        }

        else if ( DISPLAY_NUMBER == 6 )
        {

                switch (HEX_NUMBER)
                {
                        case 0: *HEX_6 = 63;break;
                        case 1: *HEX_6 = 6;break;
                        case 2: *HEX_6 = 91;break;
                        case 3: *HEX_6 = 79;break;
                        case 4: *HEX_6 = 102;break;
                        case 5: *HEX_6 = 109;break;
                        case 6: *HEX_6 = 125;break;
                        case 7: *HEX_6 = 07;break;
```

```
                        case 8: *HEX_6 = 127;break;
                        case 9: *HEX_6 = 103;break;
                        case 10: *HEX_6 = 119;break;
                        case 11: *HEX_6 = 124;break;
                        case 12: *HEX_6 = 57;break;
                        case 13: *HEX_6 = 94;break;
                        case 14: *HEX_6 = 121;break;
                        case 15: *HEX_6 = 113;break;
                }
        }
}
```

SECTION 6: main.c of the master pong board

```
////////////////////////////////////////////////////////////////////////////////
/*
 *  Program for the master board of PONG.
 *  -------------------------------------
 *  File Name     : main.c
 *  Target Device : ARM Cortex-A9 processor
 *
 *  Created on    : April 23, 2019
 *  Code Author(s): Nuo, Sanjith Chandran & Shrajan Bhandary
 *  Location      : University of Leeds
 *  Module              : ELEC5620M Embedded Microprocessor System Design
 *
 *  Description of the file:
 *      The driver is used to control different aspects of the game such as
 *       read the player's input, manipulate the position of the paddle,
 *             movement of the ball, players' scores and game-play of PONG.
 */
////////////////////////////////////////////////////////////////////////////////

#include "Game_Engine/Game_Engine.h"
                // Invoking the main header file.
#include "Graphics_Engine/Graphics_Engine.h"
        // Importing the graphics engine library for drawing geometrical shapes.
#include "DE1SoC_LT24/DE1SoC_LT24.h"
                // Importing the LCD library for interfacing with the LT24 terasic LCD.
#include "HPS_Watchdog/HPS_Watchdog.h"
                        // Importing the watchdog timer library to ensure the program doesn't
stay in an infinite loop.
#include "HPS_usleep/HPS_usleep.h"
                        // Importing the sleep library to create instances of time delay.
#include <stdlib.h>
                                        // Importing the standard library functions to perform
necessary actions.
#include <time.h>
                                        // Importing the time library so that its values can be
used as seed value for generating random numbers.

void exitOnFail(signed int status, signed int successStatus)                                //
Exit on fail sub-routine is used to ensure that the processor doesn't malfunction.
{
    if (status != successStatus)
    {
        exit((int)status);
                                // Add breakpoint here to catch failure
    }
}

/* Main Function.*/
int main(void)
{
        /* GPIO Port Base Address. */
        volatile unsigned char *GPIO_ptr  = (unsigned char *) 0xFF200070;          // The GPIO
of the master board is connected to the slave board for board-to-board communication.

        /* Variable to hold the value from the slave board */
        unsigned int slave_board_data = 0;
                        // This information holds the value of the touch screen and the paddle 1
control values from the slave board.

        /* Setting all pin as Input. */
        *(GPIO_ptr+4)  = 0x000000;
                                // The master board only receives values from the slave board.
```

```
    /* Initialise the LCD Display and exit if not successful. */
    exitOnFail( LT24_initialise(0xFF200060,0xFF200080),  LT24_SUCCESS);

        HPS_ResetWatchdog();
                                // Reset the watch dog timer so that it doesn't run out and
restarts the processor.

        /* Drawing the gaming area. White Border on two sides and grey sides on two sides. Grey
Fill. */
        Graphics_drawBox(10,10,230,310,LT24_WHITE,false,0x39E7); HPS_ResetWatchdog();
        Graphics_drawLine(10,10,230,10,0x39E7); HPS_ResetWatchdog();
        Graphics_drawLine(10,310,230,310,0x39E7); HPS_ResetWatchdog();

        /* Line. Drawing a Dashed Net between the two players. */
        Graphics_drawDash(11,160,229,160,LT24_WHITE); HPS_ResetWatchdog();

    /* Initialize the two paddles with their starting positions. */
    Paddle_1_Initialize(); HPS_ResetWatchdog();
    Paddle_2_Initialize(); HPS_ResetWatchdog();

        /* The seven-segment display is used to display player scores and the word Pong. */
        Display_Seven_Segment(1,0);
        Display_Seven_Segment(2,16);
        Display_Seven_Segment(3,16);
        Display_Seven_Segment(4,16);
        Display_Seven_Segment(5,16);
        Display_Seven_Segment(6,0);
        HPS_ResetWatchdog();

        /* Infinite Loop. */
        while (1)
        {
                slave_board_data = *GPIO_ptr & 0xFFFFFFF0;
                // Keep storing the data from the slave board as it can stop and start the game.

                while ( slave_board_data == 0 )
                        // If the slave board data is 0, then the game should be paused.
                {
                        slave_board_data = *GPIO_ptr & 0xFFFFFFF0;
                // Keep checking for new data from the slave board that can change from stop
mode to start mode and vice-versa.
                        HPS_ResetWatchdog();
                }

                Move_Ball();
                                        // If start mode, then keep moving the ball around the
gaming area.
                Move_Paddle_1();
                                        // Move paddle 1 continuously in the start mode based on
the user input.
                Move_Paddle_2();
                                        // Move paddle 2 continuously in the start mode based on
the user input.
                HPS_ResetWatchdog();
                                // Reset the watch dog timer.
        }
}
```

APPENDIX – B

Appendix – B consists of external boards used in the game.

Section 1: OpenMV camera module from website: https://openmv.io/products/openmv-cam-m7

Section 2: OpenMV camera module from website: https://www.electronicshub.org/interfacing-isd1820-voice-recorder-module-with-arduino/