# ELEC5620M Embedded Systems Design

MINI PROJECT

# GAME: SQUASH RALLY

SHRIKANT GHOSHAL

201265711

el18s2g@leeds.ac.uk

(Report word count: 2630)

# Contents

# Abstract and Introduction

This is a report which describes the construction and implementation of a “Pong”-inspired squash game scored on a rally-based system, on the Intel/Altera DE1-SoC development board paired with an LT24 display module. The peripherals used as part of this project include the KEY buttons, slide switches for inputs and output onto the Red LEDs, 7-segment displays and the LT24 screen.

# Description

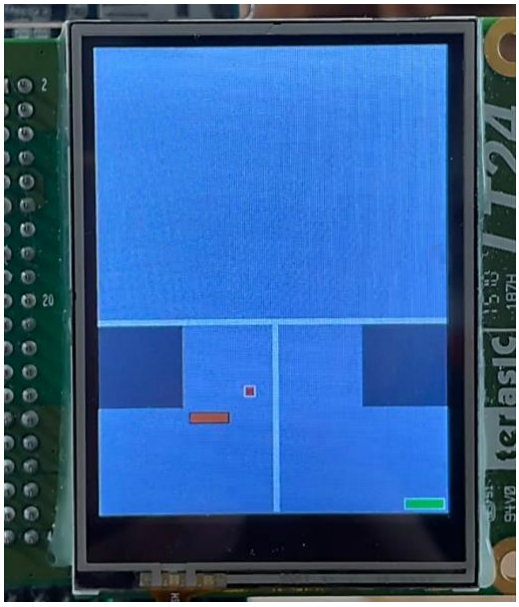
## Game elements:

- There are 2 players represented by rectangular "rackets" on either side of the court. Only one may be active at any given time - the movement of which can be controlled by the 4 KEY buttons.
- The directional controls are as follows:
    - KEY[3] = Left
    - KEY[2] = Up
    - KEY[1] = Down
    - KEY[0] = Right
- A Ball which interacts with the boundaries of the LT24 display module and an active player racket.
- A player has the following attributes created as a struct of type "squashPlayer" (defined using typedef): -
    - x and y coordinates (left and top extremes respectively).
    - height and width (a custom function is used to create a rectangular object based on these entries).
    - speed of movement (planned for future development - to be manipulated by the slide-switches).
    - score
    - active status
- The main game ball is also a self-defined struct with similar parameters: -
    - x and y coordinates (left and top extremes respectively).
    - size (a custom function is used to create a rectangular object based on these entries: supposed to be a circular object but has been classified as a future checkpoint).
    - speed of movement (Currently set during initialisation, manipulation by the slide-switches is planned for future development for higher difficulty gameplay).

## Programming logic

- The game starts with two players on the screen, represented by a rectangular "racket" on either side of the court. By default, the first player is the one on the left.
- A ball is tossed in a random direction and the first player is allowed to move the racket in order to intercept the ball before it hits the lower edge of the screen.

- In the case the player succeeds in doing so, the ball's direction changes according to the angle of impact and this gameplay continues, adding a single point to the respective player each time they can contact the ball.
- In the situation that the ball touches the lower edge of the display, both the player statuses are toggled, and the racket of the other player becomes active, gaining access to movement by the directional controls, while the former player remains inactive until the ball touches the lower edge of the LT24 display once more.
- This gameplay continues until either player reaches a pre-set maximum score controlled by the slide switches on the DE1-SoC board (this needs to be done before the programme is loaded onto the board - a corresponding red LED lights up any switches that have been turned high). If no switches are pushed up at the time of execution, the game continues until a player reaches 11 points, like any standard racket sport game. The 7 segment LCD flashes an "End" notification before declaring the winner. The board must be refreshed to start the game again



*Gameplay elements on the LT24*

## Game Construction

The gameplay logic has been implemented through a module structure, with three main modules: a racket, a ball and the game logic which uses the other two to create a viable game logic. However, the primary port of operation is through a function written to create a rectangular area on the LT24 screen – which is built on the functionality of the LT24 IP core provided in the teaching material.

```
signed int displayRectangle(
      unsigned int xleft,
      unsigned int ytop,
      unsigned int width,
      unsigned int height,
      unsigned int fillColour){
            signed int status;
            unsigned int i, j;
            //Reset watchdog
            ResetWDT();
            //Define Window
            status = LT24_setWindow(xleft,ytop,width,height);
            if (status != LT24_SUCCESS) return status;
            for (j = 0;j < height;j++){
                  for (i = 0;i < width;i++){
                        LT24_write(true, fillColour);
                  }
            }
            //Done
            return LT24_SUCCESS;
}
```

This creates a rectangle of a set colour with the provided dimensions, which has been used inside the main game loop to create the "squashPlayer" and the "ball" objects, and can be used in tandem with the **exitOnFail**() function as used in the teaching material of Unit 3.

The main display uses multiple instances of the **displayRectangle**() function to create the overall court surface with a grey background, white boundary lines and a slightly darker shade of grey for the service areas formatted to the dimensions of a classic squash court.

### Racket module

Now that the court surface has been implemented, the sequential next step was to create a module to create a movable rectangle that reacts to inputs from the KEY buttons on the DE1-SoC board. A struct was created to define a player with data constituting its coordinates, dimensions, speed, score and status (active/inactive), defined as follows:

```
typedef struct squashPlayer
{
        unsigned int xleft;
        unsigned int ytop;
        unsigned int height;
        unsigned int width;
        unsigned int speed;
        unsigned int score;
        bool active;
}squashPlayer;
```

```
void initialisePlayer(
        squashPlayer *racket,
        unsigned int xleft,
        unsigned  int ytop,
        unsigned int width_in,
        unsigned int height_in,
        bool status){
                racket->xleft = xleft;
                racket->ytop = ytop;
                racket->height = height_in;
                racket->width = width_in;
                racket->speed = 1;
                racket->score = 0;
                racket->active = status;
                }
```

The racket module is equipped with the following functions to manipulate elements of any player that has been initialised, which can be done by employing **initialisePlayer()** function with the respective parameters. The initialisation procedure offers customisation on the coordinates, dimensions and the active status of a player while setting the speed and score to a default value.

- **void** togglePlayer(squashPlayer *player1, squashPlayer *player2);
  - Changes the "status" parameter of the two input squashPlayer objects.
- **void** updatePlayerPosition(**unsigned int** keysPressed, squashPlayer *racket);
  - Updates the coordinates of the respective squashPlayer object in response to the KEY button inputs from the DE1-SoC board.
- **void** drawPlayerRacket1(squashPlayer racket);
  - Uses the drawRectangle() function to create a black bordered orange rectangle accessing the parameters of the input squashPlayer object.
- **void** drawPlayerRacket2(squashPlayer racket);
  - Uses the drawRectangle() function to create a white bordered green rectangle accessing the parameters of the input squashPlayer object. Essentially the same logic as the previous function, but offers a differentiation to the two-player capability of the game.

- `void addPoint(squashPlayer *racket);`
    - Increments the "score" parameter of a corresponding squashPlayer object.

### Ball module

The ball module "squashBall" is similar to the racket module, with the primary difference being that it requires the capability to move around the screen automatically when in the main game loop, instead of being manipulated by an input. Moreover, it needs to be reactive to the environment and constraints of the court. In order to facilitate this, two structures were defined in the same module, one depicting a movable ball, and another to quantify a rate-of-change values assisting the ball movement around the court in response to interactions.

```
typedef struct squashBall{
      unsigned int xleft;
      unsigned int ytop;
      unsigned int size;
      unsigned int speed;
}squashBall;
```

```
typedef struct
movementControls{
      unsigned int x;
      unsigned int y;
}movementRate;
```

Once initialised with the appropriate parameters, these two structures work in tandem as part of the following functions:

- `void initialiseBall(`
  `squashPlayer *ball,`
  `movementRate *ballControl,`
  `unsigned int xleft,`
  `unsigned int ytop,`
  `unsigned int size,`
  `unsigned int speed );`
    - Creates an instance of a squashBall with the default parameters.

- `void drawBall(squashBall ball);`
    - Uses instances of the displayRectangle() function to draw a white-outlined red ball on the LT24 screen.

- `void setMovement(movementRate *moveBall, unsigned int x, unsigned int x);`
    - Updates the parameters in a corresponding movementRate object with the input x and y parameters, essentially forming an initialisation function which controls the ball's speed in the gameplay.

- `void updateBallPosition(squashPlayer *ball, movementRate *moveBall);`
    - Changes the coordinates of the squashBall object each time this function is called by a factor of the movementRate parameters.

- `bool checkWallBallBounce(squashBall *ball, movementRate *ballControl);`
    - Checks whether the ball encounters the extremes of the LT24 screen and updates the value of the movementRate object to alter direction (which is implemented by a negation of the respective coordinates). This function returns a Boolean value: *true* if the ball is still

in play, and *false* if it hits the lower extreme of the LT24 screen. This serves as an indicator of when to transfer control from one player to another.

- ```
  void checkRacketBallBounce(
                    squashball *ball,
                    movementRate *ballControl,
                    squashPlayer *racket);
  ```
  - Operates on a similar logic to the previous function but references the immediate location of the corresponding squashPlayer object in order to change the movementRate object parameters before calling the **`updateBallPosition()`** function to reflect the change in the game loop. Subsequently, the function to add a point to the input racket is also called, incrementing the score of the corresponding player.

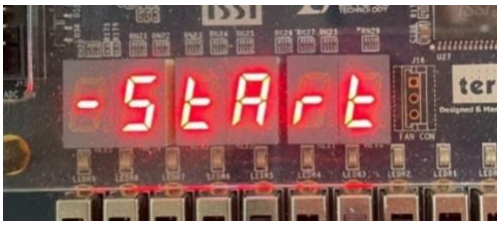
### Pre-game subroutine

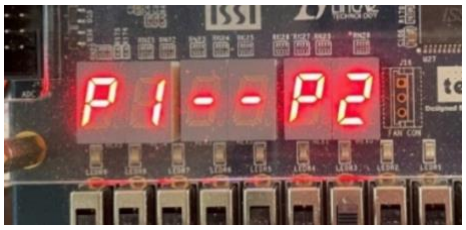
With the racket and ball modules in place, they were now ready to be used by the main game program. The pre-game subroutine allows a few parameters to be changed and set to work with the game loop: the slide switch input was paired to the red LED output to display which switches were high, and the leftmost switch at the high position dictates the maximum number of points that the game would be played until. A function called **`getMaxPoints()`** was written to return an integer value with information relating to the switches which were in a high state when the program is loaded onto the board, which enabled the aforementioned functionality. By default, if no switches are in the high position when the program is loaded onto the board, the game continues until either player gets to a score of 11 – a standard adapted from practices in popular racket sports such as badminton, table tennis and squash.

When the SVC breakpoint is released, the board first goes through a testing function on the LT24 display module, code in which has been adapted from the teaching material provided in this module, with varying delays. It continues by initialising the game elements and then sets the maximum score of the game according to the state of switches at this time.

### 7-Segment display

Informing a user of this game requires the usage of a text output – which could be implemented in two ways: first being the usage of the six 7-segment displays which could be a altered in an instant independent from the game screen, and the second: creating a set of text output sub-modules by using the **`LT24_drawPixel()`** function. For this project, the former was chosen because of two fundamental reasons – the creation of a text-output based sub-module would have be customized for this application and cannot be edited to adapt to a increasing level of functionality over the course of project development, and the refresh rate of the screen as observed in the development process would result in a flickering on the screen which would potentially cause a significant disturbance to any user attempting to read the content. On the other hand, using the 7 segment displays allows for efficient communication to the user and is not dependent on the frame rate of the display, which allows for system-wide pauses to be made without halting the flow of information to the user.
  ```



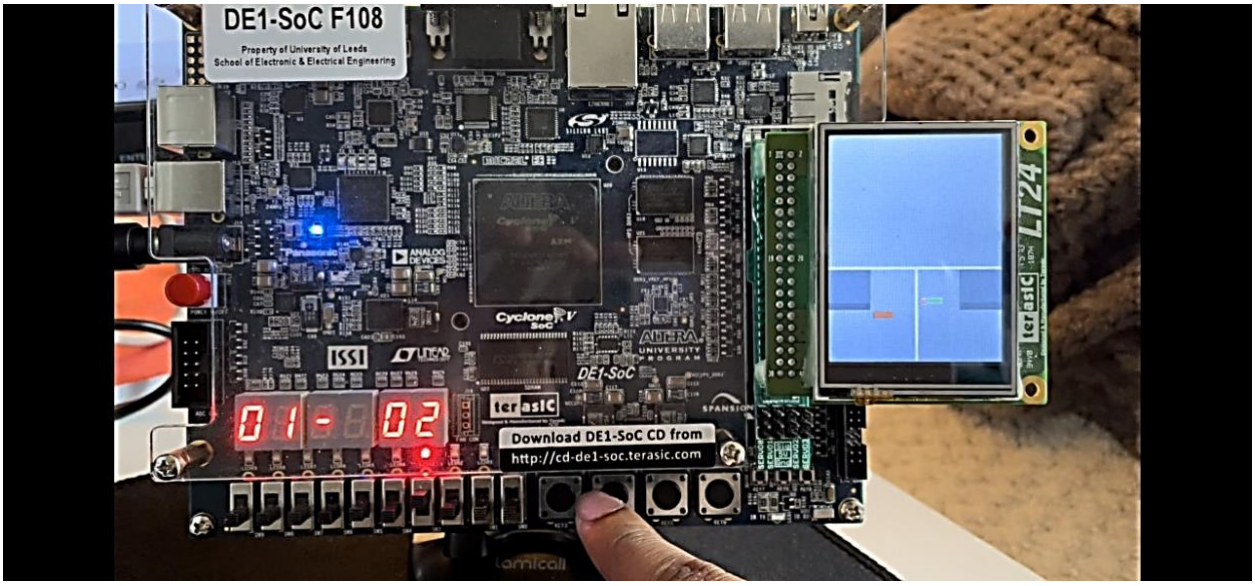




*Starting sequence*



*Ending sequence*

The pictures above show the 7-segment display outputs when the program is loaded onto the board and when the program ends. Each of the displays have a small delay in between to account for readability. During the gameplay, each player's score is displayed on their respective starting side and a "-" appears on one of the middle two displays indicating the active player at any time.




*Score display during gameplay*

### Main game Logic

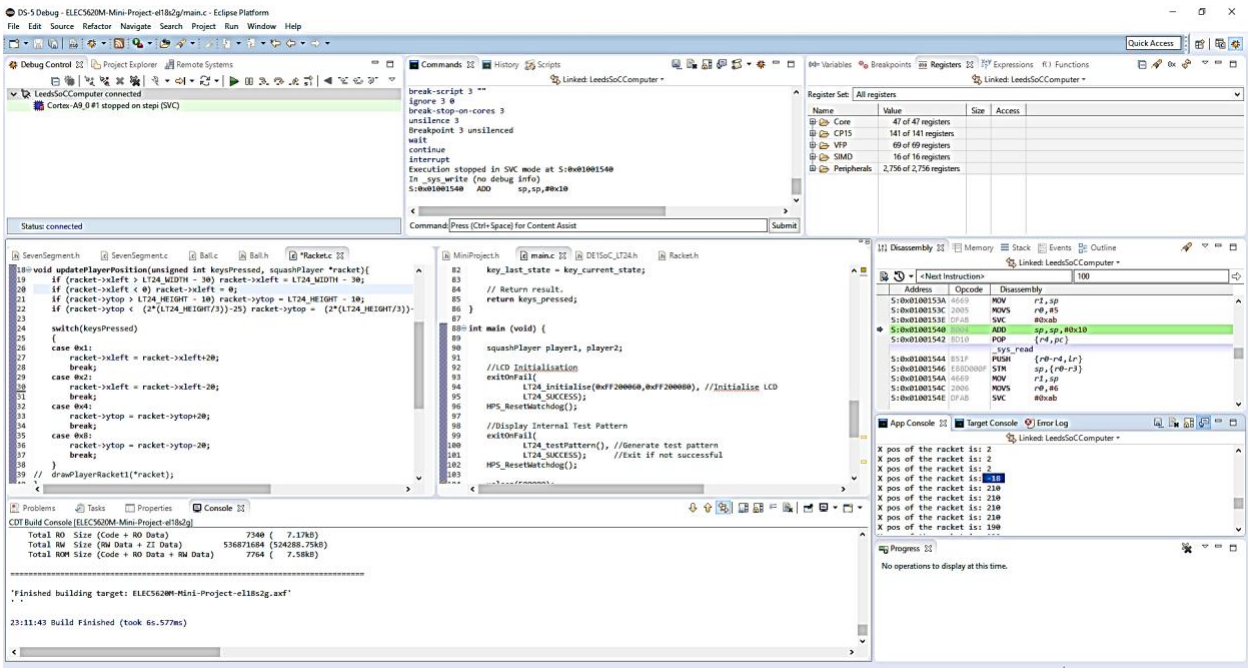
All these components of the project then needed to be placed into a viable sequence which allowed for an error-free and pleasurable gameplay, which was implemented through the main game loop. Contrary to conventional practices of using an infinite loop, this project runs a loop which starts from after the initialisation procedure to the instance where either player reaches the maximum number of points which is dictated by the slide-switches. Following this, the ending sequence on the 7-segment displays is initiated and the program execution is stopped right afterward.

Most of the functionality is achieved through a sequential application of the modules and their respective functions, however the main addition to the game loop is a function called `togglePlayer()` – that accepts two players as input and toggles their "active" status – which is triggered every time the ball bounces against the lower extreme of the screen and the `checkWallBallBounce()` function returns a "false" value. This is paired with a conditional statement that checks the active player and then allows the access of movement controls appropriately while also updating the middle 7-segment displays for indicating the active player.

## Testing, Analysis and Reflection

Testing the code involved a repetitive hardware-run analysis for most of the project-development. A very direct approach involving the observation of the enabled functionality. However, that was only possible when all the peripherals were initialised – prior to which, a precision-oriented usage of debugging tools available in the Eclipse DS-5 platform. In this project, once the peripherals were set up, a significant development went to the logic of the game, which called for the hardware-performance analysis with the inclusion of several serial out statements to check the values of variables was as a more appropriate method.

As part of the Electronics and Computer Engineering course, I have had the opportunity to cultivate skills required to write efficient and accurate programs – and as such, with the support of the teaching material available in this module, there was very little error handling to be performed with the project outline I had initially started with. Considering this, one of the main problems which I had to deal with while writing this program was an unexpected overflow while operating the rackets and the x coordinate of the respective racket went below 0. After an analysis by printing the value of the x position to the serial output, it was noticed that the value of "xleft", upon going negative, reverted to the rightmost extreme of the screen with the value being set to 210. While this could easily be replicated on the rightmost extreme with a simple value wrapping conditional statement and act as an added feature of the game rather than a bug, it was also found that a similar error appeared on the "squashBall" object when the size of the ball was increased and some of the boundaries went out of the LT24 extremes. It appeared as if the ball exited the screen on one side and appeared out of the other side, seeming like an infinite playing space. However, this was quite inconsistent and occurred at sporadic intervals. Moreover, while this gameplay was valid and enjoyable, it undermined the scoring system described earlier in this report, while also disregarding a link to the rules and regulations of the real-world squash.



*Screenshot: X value overflowing in the racket when the value is negative*

In order to handle these two fundamental errors in the overflow, a conditional statement was introduced into each of the objects “updatePosition” functions to check the limits of the LT24 screen, subtract the dimensions of the respective object and set that as the new limit, rather than relaying on the extremes of the screen itself. However, this meant that this conditional wrapper would have to be updated every time the size of the objects utilised in this game are edited from the main game loop: which isn’t a very significant issue as to the running performance of the game but leads to an additional consideration across modules when the code is edited instead of a single edit (which is expected in the case of a true modular system).

## Conclusion

This concludes the report detailing the mini-project, which is a “PONG”-type squash game with a rally-based scoring system. ARM DE1-SoC via the Eclipse DS5 platform. It is equipped with the functionality to accept user inputs through the KEY [0-3] pushbuttons which manipulate the positions of rackets in the game to hit a ball against three of the extremes on the LT24 display and results in a penalty if the ball hits the lower extreme – which switches players allowing the user-inputs to manipulate a second racket to attain a similar objective.

# Appendix

## Seven Segment code:

### SevenSegment.h

```
/*
 * SevenSegment.h
 *
 *  Created on: May 1, 2022
 *      Author: shrik
 */

#ifndef SEVENSEGMENT_SEVENSEGMENT_H_
#define SEVENSEGMENT_SEVENSEGMENT_H_

void sevenSegmentWrite(unsigned int display, unsigned char value);
void SevenSegmentOutput (unsigned int display, unsigned char value);
void scoreDisplay(unsigned int display, unsigned char value);
void singleDisplay(unsigned int display, unsigned char value);
#endif /* SEVENSEGMENT_SEVENSEGMENT_H_ */
```

### SevenSegment.c

```
/*
 * SevenSegment.c
 *
 *  Created on: May 1, 2022
 *      Author: shrik
 */


#include "SevenSegment.h"

volatile unsigned char *sevenseg_base_lo_ptr = (volatile unsigned char *) 0xFF200020;
volatile unsigned char *sevenseg_base_hi_ptr = (volatile unsigned char *) 0xFF200030;

// There are four HEX displays attached to the low (first) address.
#define SEVENSEG_N_DISPLAYS_LO 4

// There are two HEX displays attached to the high (second) address.
#define SEVENSEG_N_DISPLAYS_HI 2

void sevenSegmentWrite(unsigned int display, unsigned char value) {
    if (display < SEVENSEG_N_DISPLAYS_LO) {
        // Using byte addressing to access directly for low address
        sevenseg_base_lo_ptr[display] = value;
    } else {
        // For a high address, shift down to enable byte addressing
        display = display - SEVENSEG_N_DISPLAYS_LO;
        sevenseg_base_hi_ptr[display] = value;
    }
}

/*
```

```
 * Function that correspondingly turns on the displays on the seven segment
 * display according to the display address and the value when called.
 * Uses the sevenSegmentWrite(unsigned int display, unsigned char value)
 * function.
 */
void SevenSegmentOutput (unsigned int display, unsigned char value){

	switch(value)
		{
			case 0xF:
				sevenSegmentWrite(display,0x00);
				break;
			case 0x0 : // input is 0
				sevenSegmentWrite(display, 0x3F);
				break;
			case 0x1 : // input is 1
				sevenSegmentWrite(display, 0x6 );
				break;
			case 0x2 : // input is 2
				sevenSegmentWrite(display, 0x5B);
				break;
			case 0x3 : // input is 3
				sevenSegmentWrite(display, 0x4F);
				break;
			case 0x4 : // input is 4
				sevenSegmentWrite(display, 0x66);
				break;
			case 0x5 : // input is 5
				sevenSegmentWrite(display, 0x6D);
				break;
			case 0x6 : // input is 6
				sevenSegmentWrite(display, 0x7D);
				break;
			case 0x7 : // input is 7
				sevenSegmentWrite(display, 0x7 );
				break;
			case 0x8 : // input is 8
				sevenSegmentWrite(display, 0x7F);
				break;
			case 0x9 : // input is 9
				sevenSegmentWrite(display, 0x67);
				break;
			default  : // input out-of-bounds - only displays a dash "-".
				sevenSegmentWrite(display, 0x40);
		}
}

void scoreDisplay(unsigned int display, unsigned char value){
	int firstDigit = value % 10; // Get the first digit (LSB)
	int secondDigit = value / 10; // Get the second digit (MSB)

	SevenSegmentOutput(display,firstDigit);
	SevenSegmentOutput(display+1,secondDigit);
}
```

```
void singleDisplay(unsigned int display, unsigned char value){

	SevenSegmentOutput(display,value);
}
```

## Racket Module

### Racket.h

```
/*
 * Racket.h
 *
 *  Created on: May 1, 2022
 *      Author: shrik
 */

#ifndef RACKET_H_
#define RACKET_H_

#include "MiniProject.h"

typedef struct squashPlayer {
	unsigned int xleft;
	unsigned int ytop;
	unsigned int height;
	unsigned int width;
	unsigned int speed;
	unsigned int score;
	bool active;
}squashPlayer;

void initialisePlayer(squashPlayer *racket, unsigned int xleft, unsigned  int ytop,
unsigned int width_in,  unsigned int height_in, bool status);
void togglePlayer(squashPlayer *player1, squashPlayer *player2);
void updatePlayerPosition(unsigned int keysPressed, squashPlayer *racket);

void drawPlayerRacket1(squashPlayer racket);
void drawPlayerRacket2(squashPlayer racket);

void addPoint(squashPlayer *racket);

#endif /* RACKET_H_ */
```

### Racket.c

```
/*
 * Racket.c
 *
 *  Created on: Apr 30, 2022
 *      Author: shrik
 */
#include "Racket.h"
```

```
void initialisePlayer(squashPlayer *racket, unsigned int xleft, unsigned  int ytop,
unsigned int width_in,  unsigned int height_in, bool status){
	racket->xleft = xleft;
	racket->ytop = ytop;
	racket->height = height_in;
	racket->width = width_in;
	racket->speed = 1;
	racket->score = 0;
	racket->active = status;
}

/* Toggle the active/inactive statuses of any two input players:
 * essentially act as a hand-over program
 */
void togglePlayer(squashPlayer *player1, squashPlayer *player2){
	if (player1->active == true){
		player1->active = false;
		player2->active = true;
	} else {
		player1->active = true;
		player2->active = false;
	}

}

/* Main function that enables racket movement on the LT24 screen in response
 * to the KEY[0-3] inputs and also limits movement to the player area on the
 * screen
 */
void updatePlayerPosition(unsigned int keysPressed, squashPlayer *racket){

	if (racket->xleft > LT24_WIDTH - 30) racket->xleft = LT24_WIDTH - 30; //Screen
right extreme limit
	if (racket->xleft < 2) racket->xleft = 2; //Screen left extreme limit
	if (racket->ytop > LT24_HEIGHT - 10) racket->ytop = LT24_HEIGHT - 10; //Screen
bottom extreme limit
	if (racket->ytop <  (2*(LT24_HEIGHT/3))-25) racket->ytop =
(2*(LT24_HEIGHT/3))-25; // Maximum playing area limit

	//Reaction to Key presses
	switch(keysPressed)
	{
	case 0x1:
		racket->xleft = racket->xleft+20;
		break;
	case 0x2:
		racket->ytop = racket->ytop+20;
		break;
	case 0x4:
		racket->ytop = racket->ytop-20;
		break;
	case 0x8:
		//The value of xleft kept overflowing after becoming negative resulting
a loop to the right side of the screen
		//so a protective wrapper has been added.
```

```
			if (racket->xleft > 20){
				racket->xleft = racket->xleft-20;
			}else racket->xleft = 2;
			break;
		}

}

//draw an initialised racket with a Black-Orange colour scheme
void drawPlayerRacket1(squashPlayer racket){
		displayRectangle(racket.xleft, racket.ytop, racket.width, racket.height,
LT24_BLACK);
		displayRectangle((racket.xleft)+1, (racket.ytop)+1, (racket.width)-2,
(racket.height)-2, 0xFB83);
}

//draw an initialised racket with a White-Green colour scheme
void drawPlayerRacket2(squashPlayer racket){
		displayRectangle(racket.xleft, racket.ytop, racket.width, racket.height,
LT24_WHITE);
		displayRectangle((racket.xleft)+1, (racket.ytop)+1, (racket.width)-2,
(racket.height)-2, 0x2EA4);
}

//Adds a single point to a respective player
void addPoint(squashPlayer *racket){
		racket->score = racket->score+1;
}
```

## Ball Module

### Ball.h

```
/*
 * Ball.h
 *
 *  Created on: Apr 30, 2022
 *      Author: shrik
 */

#ifndef BALL_BALL_H_
#define BALL_BALL_H_

#include "MiniProject.h"
#include "Racket.h"

typedef struct squashBall{
		unsigned int xleft;
		unsigned int ytop;
		unsigned int size;
		unsigned int speed;
}squashBall;

typedef struct movementControls{
		unsigned int x;
```

```
	unsigned int y;
}movementRate;

void initialiseBall(squashBall *ball,  movementRate *ballControl, unsigned int xleft,
unsigned int ytop, unsigned int size, unsigned int speed);
void drawBall(squashBall ball);
void setMovement(movementRate *moveBall, unsigned int x, unsigned int y);
void updateBallPosition(squashBall *ball, movementRate *moveBall);

bool checkWallBallBounce(squashBall *ball, movementRate *ballControl);

#endif /* BALL_BALL_H_ */
```

Ball.c

```
/*
 * Ball.c
 *
 *  Created on: Apr 30, 2022
 *      Author: shrik
 */
#include "Ball.h"

void initialiseBall(squashBall *ball, movementRate *ballControl, unsigned int xleft,
unsigned int ytop, unsigned int size, unsigned int speed){
	int direction;

	ball->xleft = xleft;
	ball->ytop = ytop;
	ball->size = size;
	ball->speed = speed;

	//Ball goes toward top half of the screen
	ballControl->x=speed;
	ballControl->y=-speed;

}

void drawBall(squashBall ball){
	displayRectangle(ball.xleft, ball.ytop, ball.size, ball.size, LT24_WHITE);
	displayRectangle(ball.xleft+1, ball.ytop+1, ball.size-2, ball.size-2,
LT24_RED);
}

//Sets movement parameters with controllable rate along the x and y axes separately
void setMovement(movementRate *moveBall, unsigned int x, unsigned int y){
	moveBall->x = x;
	moveBall->y = y;
}

//Uses the movementRate struct to operate the ball movement by editing its
coordinates every time it is called.
void updateBallPosition(squashBall *ball, movementRate *moveBall){
	ball->xleft += moveBall->x;
	ball->ytop += moveBall->y;
```

```
}

//checks the ball-contact with the LT24 wall-boundaries
bool checkWallBallBounce(squashBall *ball, movementRate *ballControl){
//     if (ballHitBottom == true) ballHitBottom = false;

       if (ball->ytop <= 4){
              ball->ytop = 2;
              ballControl->y = - ballControl->y;
              return true;
       }

       else if ((ball->ytop+ball->size) >= LT24_HEIGHT-4 ){
              ball->ytop = (LT24_HEIGHT - 2) - ball->size;
              ballControl->y = - ballControl->y;
              return false;
       }

       else if (ball->xleft <= 4){
              ball->xleft = 2;
              ballControl->x = - ballControl->x;
              return true;
       }

       else if ((ball->xleft+ball->size) >= LT24_WIDTH-4 ){
              ball->xleft = (LT24_WIDTH - 2) - ball->size;
              ballControl->x = - ballControl->x;
              return true;
       }
       else return true;
}

//checks the ball-contact with the respective racket boundaries
void checkRacketBallBounce(squashBall *ball, movementRate *ballControl, squashPlayer
*racket){

       if (
                     (ball->ytop >= racket->ytop) &&
                     (ball->ytop <= (racket->ytop + racket->height + 2 )) &&
                     (ball->xleft >= racket->xleft) &&
                     (ball->xleft <= (racket->xleft + racket->width))
                     ){
              ball->ytop = racket->ytop - ball->size;
              ballControl->y = -ballControl->y;
              addPoint(racket);
       }

       updateBallPosition(ball, ballControl);
}
```

## Main program

### MiniProject.h

```
/*
```

```
 * MiniProject.h
 *
 *  Created on: Apr 30, 2022
 *      Author: shrik
 */

#ifndef MINIPROJECT_H_
#define MINIPROJECT_H_

//Size of the LCD
#define LT24_WIDTH  240
#define LT24_HEIGHT 320

#include "DE1SoC_LT24/DE1SoC_LT24.h"
#include "Ball.h"
#include "Racket.h"
#include "HPS_Watchdog/HPS_watchdog.h"
#include "SevenSegment/SevenSegment.h"

#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>

void exitOnFail(signed int status, signed int successStatus);
signed int displayRectangle(unsigned int xleft, unsigned int ytop, unsigned int
width, unsigned int height, unsigned int fillColour);
void createCourt();
unsigned int getPressedKeys();
unsigned int getMaxPoints();

#endif /* MINIPROJECT_H_ */
```

main.c

```
/*
 * main.c
 *
 *  Created on: Apr 29, 2022
 *      Author: shrik
 */

#include "MiniProject.h"

#define LSB_DISPLAY_LOCATION 0
#define MID_DISPLAY_LOCATION 2
#define MSB_DISPLAY_LOCATION 4

//Used for frame rate analysis and removed for redundancy handling
// 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;

//Pointer to the Red LEDS output on the DE1-SoC board
volatile unsigned int *LEDR_ptr = (int *)0xFF200000; // Red LEDs 0-9

//Pointer to push button inputs on the DE1-SoC board
volatile unsigned int *key_ptr = (unsigned int *)0xFF200050;
unsigned int key_last_state = 0;

//Pointer to slide switches inputs on the DE1-SoC board
volatile unsigned int *sw_ptr = (unsigned int *)0xFF200040;


//Debugging for LCD
void exitOnFail(signed int status, signed int successStatus){
    if (status != successStatus) {
        exit((int)status); //Add breakpoint here to catch failure
    }
}


//Creates a rectangle at xleft and ytop coordinates of specified dimensions and fills
it with the respective colour
signed int displayRectangle(unsigned int xleft, unsigned int ytop, unsigned int
width, unsigned int height, unsigned int fillColour){
      signed int status;
      unsigned int i, j;
      //Reset watchdog
      ResetWDT();
      //Define Window
      status = LT24_setWindow(xleft,ytop,width,height);
      if (status != LT24_SUCCESS) return status;
      for (j = 0;j < height;j++){
              for (i = 0;i < width;i++){
                  LT24_write(true, fillColour);
              }
          }
      //Done
      return LT24_SUCCESS;
}

//Uses the displayRectange() function to create a scaled display of a squash court.
void createCourt(){
      displayRectangle(0,0, LT24_WIDTH, LT24_HEIGHT, 0xBDF7);

      //Left service box
      displayRectangle(0, (2*(LT24_HEIGHT/3))-25, (LT24_WIDTH/4)-2, LT24_WIDTH/4,
0x6B6D);

      //Right service box
      displayRectangle((3*(LT24_WIDTH/4))+2, (2*(LT24_HEIGHT/3))-25, (LT24_WIDTH/4)-
2, LT24_WIDTH/4, 0x6B6D);
```

```
	displayRectangle(0,(2*(LT24_HEIGHT/3))-26,LT24_WIDTH,4,
LT24_WHITE);//Horizontal court line
	displayRectangle((LT24_WIDTH/2), (2*(LT24_HEIGHT/3))-26, 4,LT24_HEIGHT/3+27,
LT24_WHITE);//Vertical court line

}

/*
 * Function that tracks the pushbutton input from users on the DE1-SoC:
 * It returns an unsigned integer value corresponding to the button that
 * has been pressed.
 */
unsigned int getPressedKeys() {

    // Store the current state of the keys.
    unsigned int key_current_state = *key_ptr;

    // If the key was down last cycle, and is up now, mark as pressed.
    unsigned int keys_pressed = (~key_current_state) & (key_last_state);

    // Save the key state for next time, so we can compare the next state to this.
    key_last_state = key_current_state;

    // Return result.
    return keys_pressed;
}

/* Reads an input from the slide-switches when called and returns a corresponding
integer value
 * depicting the maximum number of points. The return value corresponds to the
maximum digit of
 * the slide switches in the "on" position when the program is loaded onto the board.
 */
unsigned int getMaxPoints(){
	unsigned int returnVal;
	unsigned int sw_value = *sw_ptr;

	if(sw_value < 0x2) returnVal = 0;
	else if(sw_value < 0x4) returnVal = 1;
	else if(sw_value < 0x8 ) returnVal = 2;
	else if(sw_value < 0x10) returnVal = 3;
	else if(sw_value < 0x20) returnVal = 4;
	else if(sw_value < 0x40) returnVal = 5;
	else if(sw_value < 0x80) returnVal = 6;
	else if(sw_value < 0x100) returnVal = 7;
	else if(sw_value < 0x200) returnVal = 8;
	else if(sw_value < 0x400) returnVal = 9;
	else returnVal = 10;

	*LEDR_ptr = sw_value;
	return returnVal;
}

/*
 * Initialises the game elements: two players and a ball moving at a certain speed.
```

```
 * The seven-segment displays show a word "-StArT" for a short interval before the
game starts
 */
void initGame(squashPlayer *player1, squashPlayer *player2, squashBall *mainBall,
movementRate *initialSpeed){
	createCourt();

	initialisePlayer(player1, 2, LT24_HEIGHT-10, 28, 8, true);
	initialisePlayer(player2, LT24_WIDTH-30, LT24_HEIGHT-10, 28, 8, false );

	initialiseBall(mainBall, initialSpeed, LT24_WIDTH/2, LT24_HEIGHT/2, 8, 2);

	drawPlayerRacket1(*player1);
	drawPlayerRacket2(*player2);
	drawBall(*mainBall);


	//write -StArT to the 7seg Displays
	singleDisplay(5, 100);
	sevenSegmentWrite(4, 0x6D);
	sevenSegmentWrite(3, 0x78);
	sevenSegmentWrite(2, 0x77);
	sevenSegmentWrite(1, 0x50);
	sevenSegmentWrite(0, 0x78);
	usleep(5000000);

	//write P1--P2 on the 7seg displays
	sevenSegmentWrite(5, 0x73);
	singleDisplay(4, 0x1);
	singleDisplay(3, 100);
	singleDisplay(2, 100);
	sevenSegmentWrite(1, 0x73);
	singleDisplay(0, 0x2);
	usleep(5000000);
}


/*
 * To be used after the game loop finishes: compares the player scores and
 * declares the winner on the 7-segment displays before terminating the program.
 */
void playerWin(squashPlayer *player1, squashPlayer *player2){

	if(player1->score > player2 ->score){
		//P1-WIn
		sevenSegmentWrite(5, 0x73);
		singleDisplay(4, 0x1);
		singleDisplay(3, 100);
		sevenSegmentWrite(2, 0x1D);
		sevenSegmentWrite(1, 0x30);
		sevenSegmentWrite(0, 0x54);
	}else {
		//P2-WIn
		sevenSegmentWrite(5, 0x73);
		singleDisplay(4, 0x2);
```

```
            singleDisplay(3, 100);
            sevenSegmentWrite(2, 0x1D);
            sevenSegmentWrite(1, 0x30);
            sevenSegmentWrite(0, 0x54);
      }

}

void endingSequence(squashPlayer *player1, squashPlayer *player2){

      //WRITE --EnD- on the 7seg displays
      usleep(5000000);
      singleDisplay(5, 100);
      singleDisplay(4, 100);
      sevenSegmentWrite(3, 0x79 );
      sevenSegmentWrite(2, 0x54);
      sevenSegmentWrite(1, 0x5E);
      singleDisplay(0, 100);
      usleep(5000000);
      //Display winner on 7-segment displays
      playerWin(player1, player2);
}

int main (void) {

      squashPlayer player1, player2;
      squashBall mainBall;
      movementRate initialSpeed = {1,1};
      unsigned int maxPoints;

      //LCD Initialisation
      exitOnFail(
                  LT24_initialise(0xFF200060,0xFF200080), //Initialise LCD
                  LT24_SUCCESS);
      HPS_ResetWatchdog();

      //Display Internal Test Pattern
      exitOnFail(
                  LT24_testPattern(), //Generate test pattern
                  LT24_SUCCESS);      //Exit if not successful
      HPS_ResetWatchdog();

      usleep(500000);
      HPS_ResetWatchdog();

      //Game elements initialised
      initGame(&player1, &player2, &mainBall, &initialSpeed);
      usleep(500000);
      //If no switches are on the "On" position (also indicated by LEDR) then the
game defaults to an 11 point maximum.
      if(getMaxPoints() == 0) maxPoints = 10;
      else maxPoints = getMaxPoints();


      //main game loop
```

```
	while(player1.score <= maxPoints && player2.score <= maxPoints){
		createCourt();
		updateBallPosition(&mainBall, &initialSpeed);
		if (!checkWallBallBounce(&mainBall, &initialSpeed)){
			togglePlayer(&player1, &player2);
			usleep(5000000);
		}

		if (player1.active){
			updatePlayerPosition(getPressedKeys(), &player1);
			checkRacketBallBounce(&mainBall, &initialSpeed, &player1);
			singleDisplay(3, 100);//"-" indicating the active player
			singleDisplay(2, 0xF);//Turn the other 7seg display off (custom
code)
		}

		if (player2.active){
			updatePlayerPosition(getPressedKeys(), &player2);
			checkRacketBallBounce(&mainBall, &initialSpeed, &player2);
			singleDisplay(2, 100);//"-" indicating the active player
			singleDisplay(3, 0xF);//Turn the other 7seg display off (custom
code)
		}

		drawPlayerRacket1(player1);
		drawPlayerRacket2(player2);
		drawBall(mainBall);
		scoreDisplay(MSB_DISPLAY_LOCATION, player1.score);
		scoreDisplay(LSB_DISPLAY_LOCATION, player2.score);

		HPS_ResetWatchdog();
		}

	endingSequence(&player1, &player2);
}
```