VIA UNIVERSITY COLLEGE
ICT ENGINEERING

Project Report

SEP4E

Stefan-Daniel Horvath
Ovidiu Muresan

August 2018

Table of content

Abstract

This report describes the functionality and development process of a Pong game and how a real-time embedded system is designed and implemented.

The purpose of the project is to combine electronics and real-time programming in order to gain a more practical understanding of the principles.

The requirements for the projects were to make a real-time embedded application:

- using freeRTOS;
- using at least three tasks,two of which need to be hard real time tasks;
- allowing two users to interact with the application, one using the Joystick connected to the board, and one using the PC;
- and the PC application needs to send and receives data from the board, using a USB connection;

The program that was developed is Pong,a real time application that two users can play at the same time.

The hardware for the project:Atmel-ICE,Atmega2560-AVR-Microcontroller, a display, a reset button, a joystick, and a PC with usb connection.

The system was developed during a 18 days period. The first step was to make requirements and analysis; the second step to make the software design, and the last step was to implement and the test the product. The game is playable and the requirements have been met.

Introduction

An embedded system is a computer system with a dedicated function within a larger mechanical or electrical system, often with real-time computing constraints.

In computer science, real-time computing (RTC), or reactive computing describes hardware and software systems subject to a "real-time constraint". Real-time programs must guarantee response within specified time constraints, often referred to as "deadlines".

A video game is an electronic game that involves interaction with a user interface to generate visual feedback on a video device such as a TV screen or computer monitor. Some theorists categorize video games as an art form, but this designation is controversial.

A game controller is a device used with games or entertainment systems to provide input to a video game, typically to control an object or character in the game

Requirements

Functional requirements:

- The game starts when the game is powered.
- Moving the joystick can move the paddle that's on the screen.
- The game resets when the reset button is pressed, and starts again normally.
- The ball moves after the game started, and doesn't move when the game is finished.
- The second players paddle can be moved from the keyboard.
- The board sends and receives data from a connected PC.
- The screen refreshes with the current positions of the paddles and ball.

Use case diagram
The requirements mentioned above are combined to create the use case diagram. The use case diagram gives an overview how the user interacts with the system and also illustrate the functionality of the system to the user.

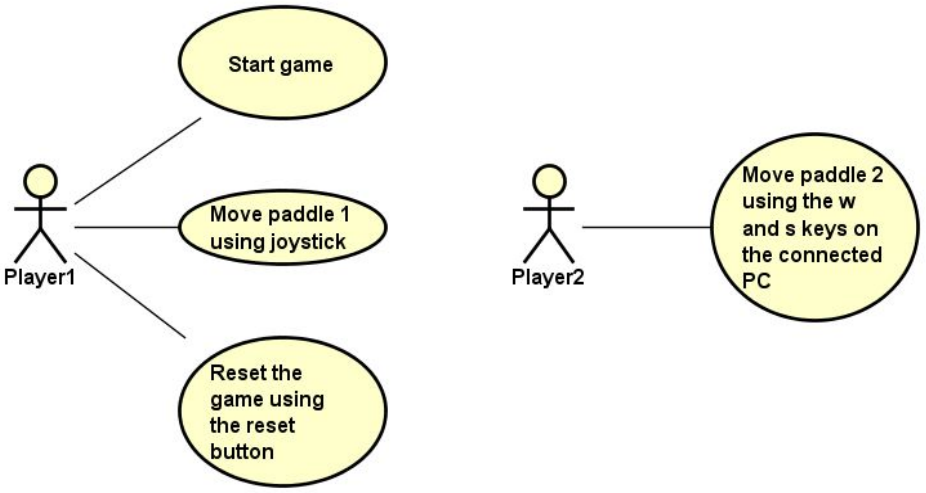

Figure 1 – Use Case Diagram

Use case description

The use case diagram is described in more detail by the use case description. In figure 2 we can see the description for moving the paddle.

ITEM	VALUE
UseCase	Move paddle 1 using joystick
Summary	The user moves the paddle left and right using the joystick on the board.
Actor	Player1
Precondition	The game started, and it did not end yet.
Postcondition	The paddle moved
Base Sequence	1.Player holds the joystick in the desired direction. 2.If the direction is valid, the paddle will move.
Branch Sequence	After the paddle moved, it can move again.
Exception Sequence	If the paddle is at the screen endge, it cannot move past it.
Sub UseCase	
Note	

Figure 2 – Use Case Description

Non-functional requirements:

- The game must be implemented using C and FreeRTOS.
- The game logic must be on board.
- The game must be displayed on the DOT-Matrix display.
- The game must be a real-time program and must contain at least three real-time tasks, and at least two of the tasks must be hard real-time.
- The game must be played by two players. One using the board joystick, and the other one using the pc.
- The PC application can send and receive data from the board.
- The PC application must communicate with the board using USB.

Analysis

In this section, we look closely at the project description in order to extract the requirements both functional and non-functional, the system is going to fulfil. Then with the use case descriptions and use case diagram the steps to be taken in order to achieve the requirements will be show.

Pong Description

Pong is one of the earliest arcade video games,it's a two-dimensional sports game that simulates table tennis. The player controls an in-game paddle by moving it vertically across the left or right side of the screen. They can compete against another player controlling a second paddle on the opposing side. Players use the paddles to hit a ball back and forth. The goal is for each player to reach a certain number points before the opponent; points are earned when one fails to return the ball to the other.

Activity Diagram

Each text box represents an action taken by Game Console and PC application, where the rhombus represents a decision point, where more outcomes are possible from one action.

The activity diagram shown in figure 3 illustrates the steps taken by the game, beginning with powering it on (black circle). Each text box is an action taken by the game, where the rhomb is a decision point, where more outcomes are possible.

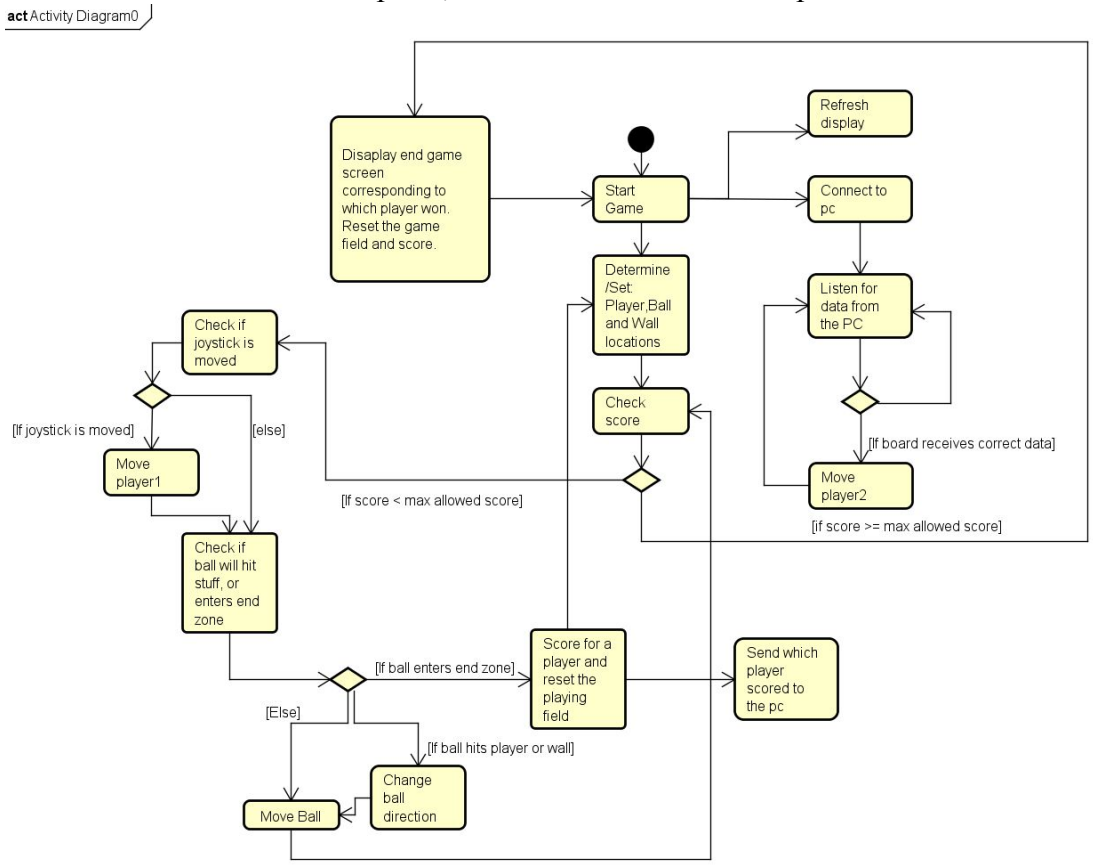

Figure 3 – Activity diagram

Game Console Hardware

Figure 4 shows the hardware that was used in the project: the board and its components, the ATMEL-ICE, and the R2R connector.

The board is equipped with a processor ATmega324PA. The microchip maximum clock frequency is 20MHz, program memory size 32kB and Data RAM Size 2k. The board has joystick Series JS 5208 that has 5-positions (up, down, left, right, enter) and a display DOT-matrix LED (20 x 14).

The ATMEL-ICE is a powerful development tool for debugging and programming SAM and AVR microcontrollers with on-chip debug capability.

Figure 4 – Avr microcontroller

Design

This part of the report concerns the design choices made regarding the technical aspects of the system and is divided into three parts: Tasks, Protocols and Serial Communication.

Architecture

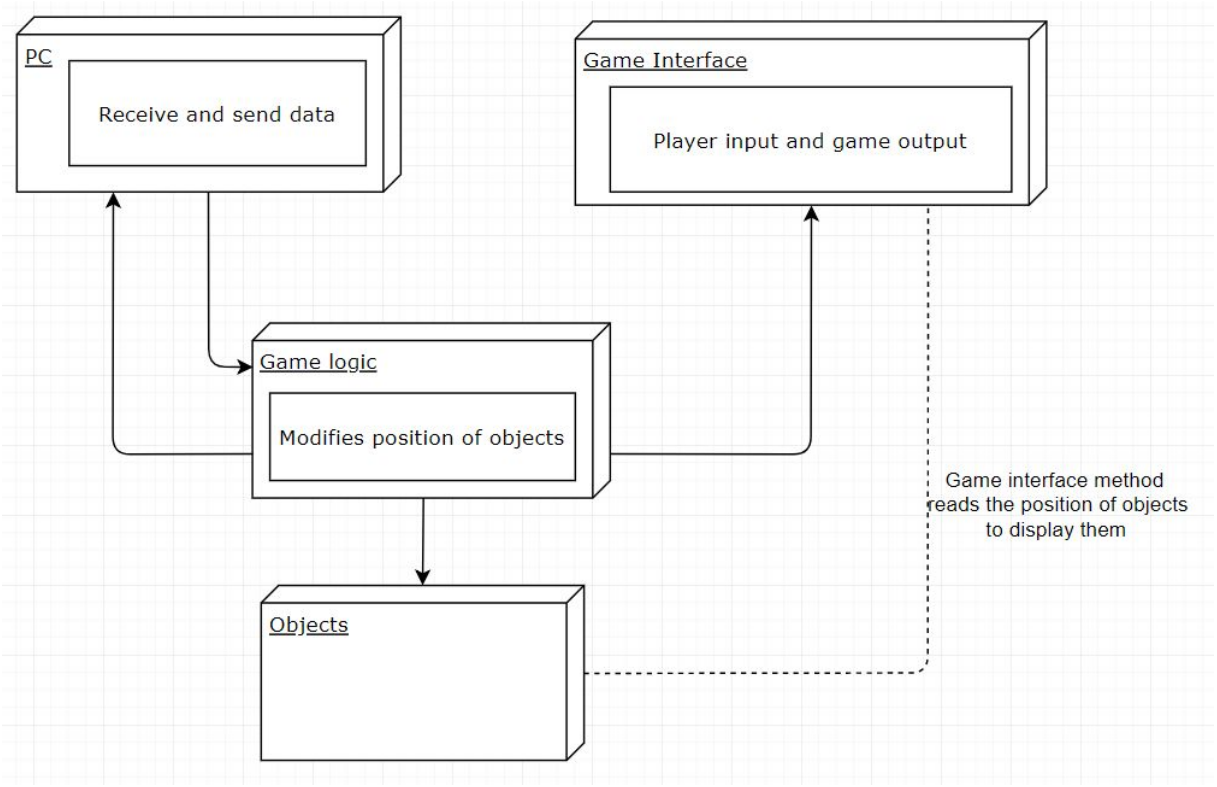

Figure 5 - Architecture pattern

Technologies

The technologies used in this project are: FreeRTOS, assembly, C, USB.

The application is coded in C, using Atmel Studio. FreeRTOS is used as the operating system on which the tasks are run. Assembly is the language in which the drivers for the ATMEL board are written.

USB, or universal serial bus, is an industry standard for cables, connectors and protocols for connection, communication and power supply between personal computers and their peripheral devices. In this project, usb is used to power the board and establish communication between it and the PC, and also to power the Atmel-ICE and program the board.

Class Diagram

In the figure below you can see the diagrams both for pc side and microcontroller side.

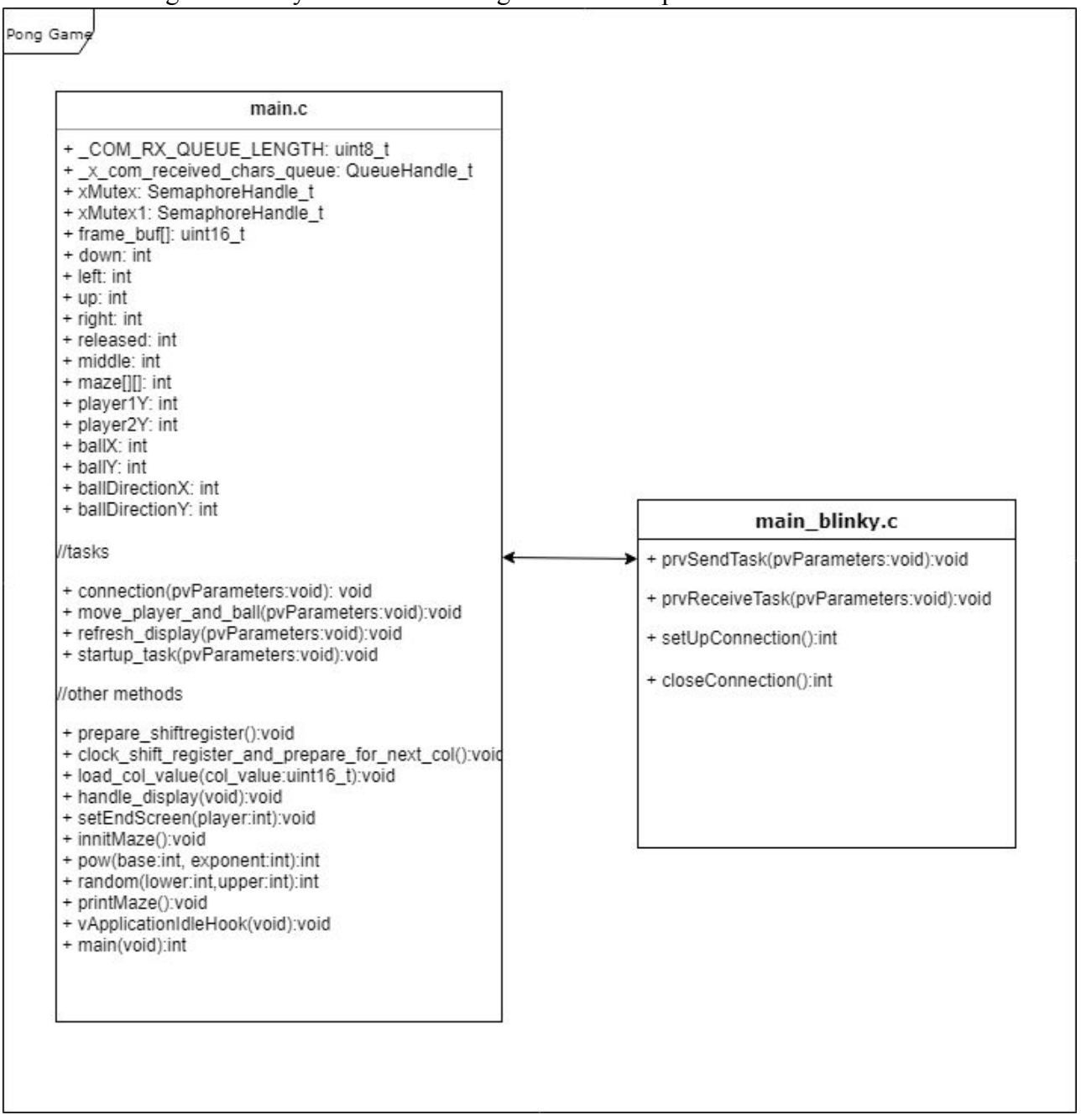

Figure 6 - Class diagram

User interface

The game is displayed on the led dot matrix(10x14) attached to the microcontroller. The players can interact with the game by using the joystick and the keyboard.

Tasks

The real-time system is programmed in C using FreeRTOS. A real time application that uses a RTOS, can be structured as a set of independent tasks. Each task executes within its own context with no other dependencies on other tasks within the system or the RTOS scheduler itself. Only one task within the application can be executing at any point in time and the RTOS scheduler is responsible for deciding which task should run first based on it's priority.

The states of a FreeRTOS task can be:

- **Running**
  -task is executing using the CPU
- **Ready**
  -task may run but is waiting for the CPU to be available
- **Blocked**
  -task is delayed (timing/event) or is waiting for another task (synchronization)
- **Suspended**
  -task may enter this state only through specific calls
  -there is no associated timeout
  -not considered in scheduling

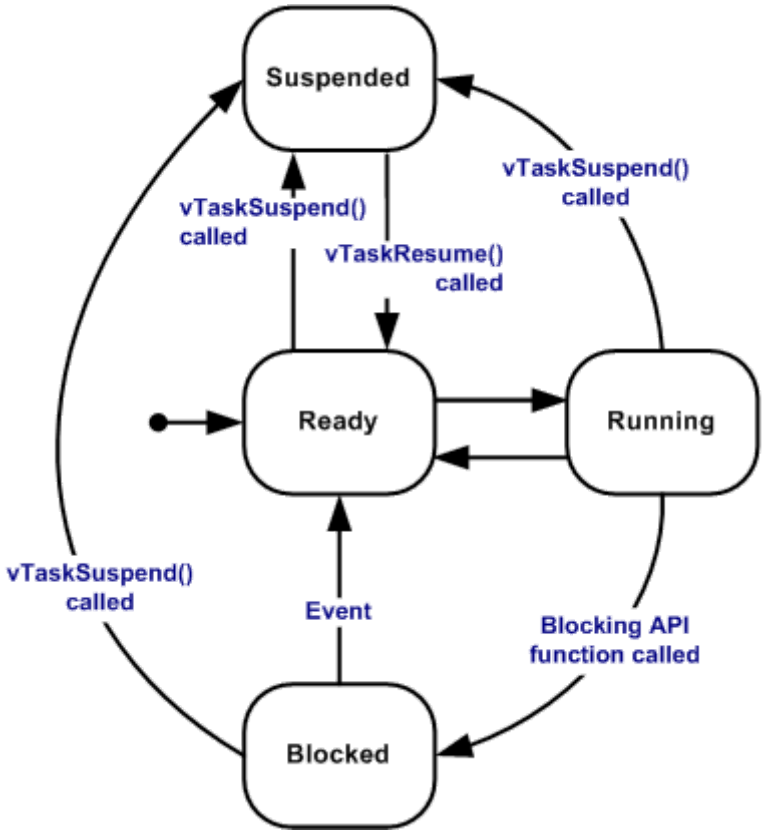

Figure 7 - Valid task state transitions

Task functionality

In order to code the game three tasks make up the core of the real-time program.The functionality of the real-time system for Pong game can be described by the following tasks:

```
BaseType_t t1 = xTaskCreate(connection, (const char *)"Connection", configMINIMAL_STACK_SIZE, (void *)NULL, tskIDLE_PRIORITY+3, NULL);
BaseType_t t2 = xTaskCreate(move_player_and_ball, (const char *)"Move", configMINIMAL_STACK_SIZE, (void *)NULL, tskIDLE_PRIORITY+3, NULL);
BaseType_t t3 = xTaskCreate(refresh_display, (const char *)"Refresh", configMINIMAL_STACK_SIZE, (void *)NULL, tskIDLE_PRIORITY+1, NULL);
```

Figure 8 - Tasks

Task 1:

Connection task.

If there is something sent to the board, check what is sent and move player 2 if what is received is correct.

Task 2:

Game logic task.

Determines player and ball position, and also contains the logic that allows player 1 to move.

Task 3:

Refresh display task.

Prints the position of the player, ball and walls.

Game state:

- Board is unpowered. Game is off.
- Board is powered. Game is on.
- Player scores, positions get reset and game goes on.
- Score limit is reached. Game winner is announced and after a delay game starts again.

Scheduling

To provide scheduling guarantees for the system and taking in consideration the three periodic tasks with execution time less than their periods, Fixed-Priority Scheduling is used. The systems is using two hard-real-time tasks that have the same priority and one soft time task that has low priority.

- The application is assumed to consist of a fixed set of tasks
- All tasks are periodic, with known periods
- The tasks are completely independent of each other
- All tasks have a deadline equal to their period (that is, each task must complete before it is next released)

Task	Period, T	Computation time C	Priority, P

1. connection	200	10	3
2.game_logic	200	62	3
3.display	100	13	1

The computation time was measured using the picoscope.

Utilization-Based Analysis

$$U \equiv \sum_{i=1}^{N} \frac{C_i}{T_i} \leq N(2^{1/N} - 1)$$

$$U \leq 0.69 \text{ as } N \to \infty$$

• For D=T task sets only.
• A simple sufficient but not necessary schedulability test exists.
The utilization bound for 3 tasks is 78%
U=C/T
U total is 49% which is below the utilization bound for 3 tasks => test passed

Response Time Analysis

Is sufficient and necessary(exact)

If the task set passes the test they will meet all their deadlines; if they fail the test then, at run-time, a task will miss its deadline (unless the computation time estimations themselves turn out to be pessimistic). Ri<Di, as opposed to RMS where a system may meet its deadlines despite a negative analysis result.

$$R_i = C_i + \sum_{j \in hp(i)} \left\lceil \frac{R_i}{T_j} \right\rceil C_j$$

Task	Period, T	Computation time C	Priority, P	Response time, R
1. connection	200	10	3	10
2.game_logic	200	62	3	62
3.display	100	13	1	20.31

The response time for Game Console tasks is less then Period => Test passed

Protocol

USB has detection of errors via a CRC on each packet; if corruption is detected, the USB controller will drop the packet and the host will have to (automatically) resend it - unless it is an Isochronous type endpoint. For Isochronous endpoints, corrupt packets are just dropped and not re-sent.

Implementation

The game consists of 3 tasks. 2 hard real time tasks: the connection task and the game logic task; and one soft real time task: the display task.

1. The connection task (figure 11) listens for incoming data from the PC. This task has a high priority because it is responsible for the movement of player 2, and any delay on this task can cause player 2 to loose unfairly.
   In this task, if there is something received, and if it's something expected, then player 2's position is changed accordingly.

2. The game logic task is responsible for:
   a. initializing the playing field.
   b. setting the initial random position of the ball.
   c. setting the initial direction of the ball.
   d. setting the initial position for the players.
   e. keeping score.
   f. resetting the game after the score is reached.
   g. displaying the end game.
   h. player, ball and wall hit detection.

3. The refresh display method is responsible for updating the screen with the current position of the players and ball every now and then, so that the players are able to play. The task simply calls the method in figure 10.

This is the main method for the game, the other tasks need to be started from the startup task because freeRTOS.

```
int main(void)
{
    init_board();

    // Shift register Enable output (G=0)
    PORTD &= ~_BV(PORTD6);

    //creates tasks
    BaseType_t t1 = xTaskCreate(startup_task, (const char *)"Startup", configMINIMAL_STACK_SIZE, (void *)NULL, tskIDLE_PRIORITY, NULL);

    // Start the display handler timer
    init_display_timer(handle_display);


    sei();
    //Start the scheduler
    vTaskStartScheduler();
    //Should never reach here
    for(;;);
}
```

Figure 9- Main method

```
void printMaze()  // PRINTS MAZE  ////////////////////////////////////////////////////////////////////////////////
{

    for (int j = 0; j < 14; j++) {
        int dec = 0;
        int power = 0;
        for (int i = 0; i < 10; i++)
        {
            if (maze[i][j] == 1  || maze[i][j] == 3 || (j==ballX && ballY== i)) {
                //(j==0 && player1Y == i) || (j==0 && player1Y+1 == i) || (j==0 && player1Y+1 == i) || (j==13 && player2Y == i) || (j==13 && player2Y+1 == i) || (j==13 && player2Y+1 == i)
                dec += pow(2, power);
            }
            if(j==0 || j==13){
                maze[i][j]  =2;
                if((j==0 && player1Y == i) || (j==0 && player1Y+1 == i) || (j==0 && player1Y+1 == i) || (j==13 && player2Y == i) || (j==13 && player2Y+1 == i) || (j==13 && player2Y+1 == i)){
                    maze[i][j]  =3;
                }
            }

                power++;
        }
        frame_buf[j]=dec;
    }
}
```

Figure 10- Method for displaying

```
void connection(void *pvParameters)
{
    // The parameters are not used
    ( void ) pvParameters;

    #if (configUSE_APPLICATION_TASK_TAG == 1)...
    #endif

    BaseType_t result = 0;
    uint8_t byte;

    while(1)
    {
        if( xSemaphoreTake( xMutex1, 250) ){
            result = xQueueReceive(_x_com_received_chars_queue, &byte, 1000L); //this is what it receives
            xSemaphoreGive(xMutex1);}
        if (result) { //if there is a result
            //simple check if what was received corresponds to known values, do a thing.
            if(byte=='s')
                if (player2Y != 0)
                    player2Y = player2Y - 1;
            if(byte=='w')
                if (player2Y != 8)
                    player2Y = player2Y + 1;

            }
        //vTaskDelay(200);
    }
}
```

Figure 11- Method for receiving data

```
if(PinCValue==up)
    if (player1Y != 0){
            player1Y = player1Y - 1;}
        //  player2Y = player2Y - 1;} //temporary until PC connectivity is implemented

if(PinCValue==down)
    if (player1Y != 8){
            player1Y = player1Y + 1;}
        //  player2Y = player2Y + 1;} //temporary until PC connectivity is implemented
```

Figure 12- Moving the player

```
else if(maze[ballY+ballDirectionY][ballX+ballDirectionX]==3)
    {
        ballDirectionX=ballDirectionX*(-1);
    }
```

Figure 13- Player hit detection

Test

The system was unit tested during the development process. Before starting on a new method or feature, the current one needs to work.

Test Specification

- The game starts when the game is powered.
  - Tested by hard coding the screen to light up when the program is running.
- Moving the joystick can move the paddle that's on the screen.
  - Tested by moving a dot using the joystick. Then evolved into the player controls the game uses.
- The game resets when the reset button is pressed, and starts again normally.
  - The reset is a built in functionality of the board, the program simply starts again.
- The ball moves after the game started, and doesn't move when the game is finished.
  - The direction of the ball is also the 'speed'. If it is set to 0, the ball doesn't move, if it's 2, it goes 2 pixels at a time, but can also go through walls.
- The second players paddle can be moved from the keyboard.
  - Tested using cmd, with the command "echo w > COM3", in our case COM3
- The board sends and receives data from a connected PC.
  - Tested using HTERM, where the board is programmed to send back whatever bits it receives.
- The screen refreshes with the current positions of the paddles and ball.
  - Tested with the ball speed set to 0.

Discussion

There is a lot of delay for the PC application, when it listens if the board sent a message. This causes player2 to have some delay and jumpiness in their gameplay. If the task responsible for listening is inactive, there is no delay.

There is a problem with the corners of the play area as it gets intersected by the 'walls' and by the 'finish line' area, this was fixed on one side but not on the other.

Another bug that we have, and that wasn't fixed is that if the PC application sends very many messages to the board through the serial connection, the application just stops. It doesn't freeze, but instead the game screen changes and nothing else will happen until you

reset the game. The reason might be that the queue for receiving information is going out of bounds.

An issue with the display that was seen, is that there is a faint shadow of the paddles: for example , if the player on the left side is moving up and down, on the right side, you can faintly see the leds lighting up where the player would be. This is not an issue but it should be mentioned.

Initially the game was supposed to be a kind of maze game played by two players, but this idea was scrapped because the screen is not complex enough and you couldn't tell the player apart from the walls, or other objects. This is the reason the method responsible for displaying stuff is called 'Print maze'.

Conclusions

The project was successfully completed, all the requirements have been implemented and tested.

The players are able to control their paddles, the ball starts at a random point and direction in the center of the play area, the hit collision (mostly) works, the score is being kept and the game ends when the score reaches a limit. The board communicates with the PC using the USB, the PC app does its job of registering player 2s keypresses, and by displaying information given to it by the board.

Sources of information

https://en.wikipedia.org/wiki/Pong
https://www.avrfreaks.net/forum/does-usb-have-error-correction-it
http://www.microchip.com/mplab/avr-support/atmel-studio-7

Appendices

The purpose of your appendixes is to provide extra information to the expert reader. List the appendices in order of mention.
Examples of appendices
* Project Description
* User Guide
* Source code – source documentation
* Diagrams
* Data sheets