École Polytechnique Fédérale de Lausanne

Andrew Dobis, SCIPER 272002

System-on-Chip Project - Spring 2020
Lab 4

CS-309 Projet de Systems-on-Chip

8th June 2020

Contents

1 Introduction

Embedded systems can be found everywhere, from basic traffic lights all the way to cell phones. However these systems are quite useless if they aren't running any software on them. That's why the goal of this lab is to write an application for our constructed embedded Linux system, that uses some of the peripherals that we have setup during the course of the semester. Many projects are possible for this lab, however due to the state of the currently used Cyclone V board, the choice remains quite limited. The board does not have the necessary pins to attach an LCD onto it or to connect the camera to it. Our current target board is also missing a VGA port. All of this means that the framebuffer manager and VGA sequencer will not be necessary for this lab. That is why the best solution would be to write an application that uses the attached joysticks. After that entire thought process, it has been decided that the application will be a *PONG* game written using the `ncurses` library. The result is illustrated in figure 1.

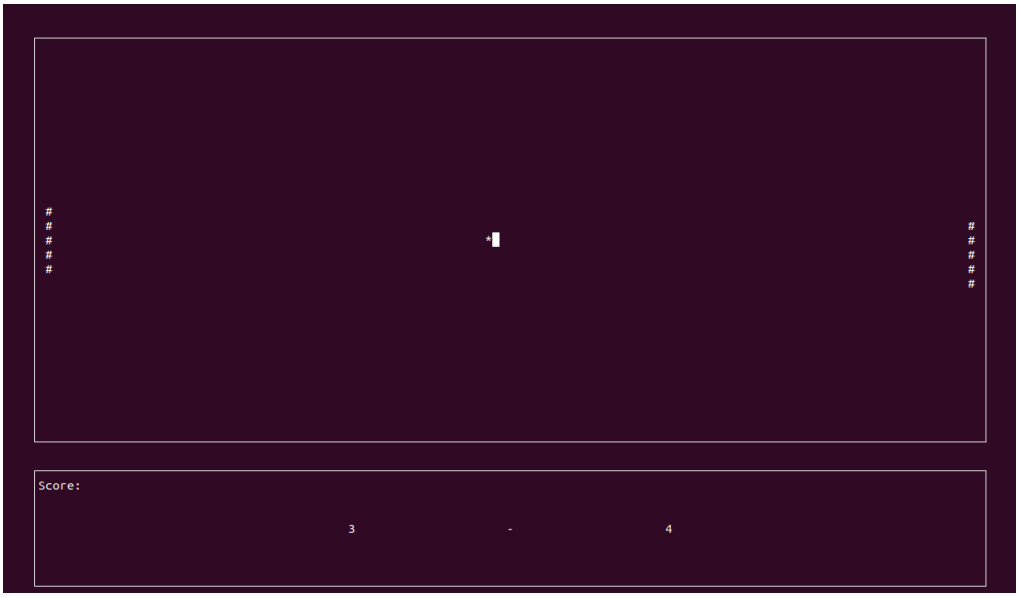

Figure 1: Screenshot from our PONG game.

2 System Architecture

Before starting the project, we need to review how our entire system is structured and figure out how we would like to run our application.

2.1 Setting up the Environment

First off, differently than what we've been doing in the previous labs, we want our application to run on our board's HPS rather than a softcore processor. We thus have two main solutions for that: one would be to cross compile our application for ARM on a separate computer and then simply give the board a compiled binary, an other solution would be to simply compile our application directly on the board. Let's briefly analyse the two solutions. The first would probably be more optimal, since our host computer has a much better processor than the HPS, however it would also require us to have a cross compiled version of `ncurses`, which we would have to compile from source. The second

solution, which is what we will chose to follow, requires us to install a C compiler on our board and to do that we would need more space than the initial `512MB`. Since our SD card has a total of 8GB of memory, we can easily afford to raise the size of the last partition up to `2048MB` in order to have plenty of space to run our application. Once that part is modified in the `create_linux_system` shell script, we can start running it to create our environment.

2.2 Overall Structure

For our application, the only peripheral that we will need are the joysticks. This means that we are going to need to synthesise our MCP3204 and the SPI it needs. As a reminder, MCP3204 is a 12-bit AD Converter that will allow us to interpret the analog signals coming from the joysticks and the SPI (Serial Peripheral Interface) is what will allow us to read the data coming from the AD Converter. Those two components will thus be synthesised on our board's FPGA, while the rest of our code, including the game logic and joystick control software, will be running on our board's HPS. This structure is illustrated in figure 2.

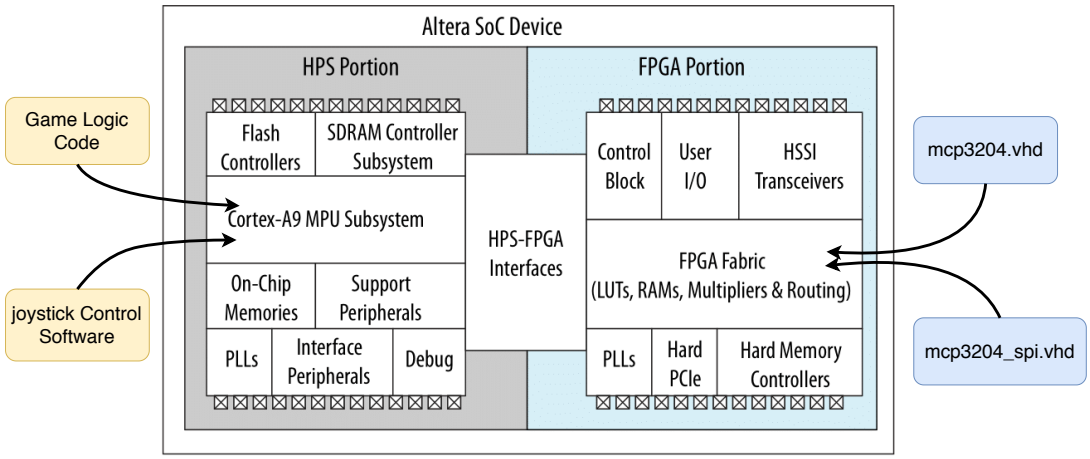

Figure 2: Overall view of the Cyclone V architecture and where our code is executed.

The joysticks will be accessed from the HPS using the MCP3204's serial interface that is read from our joystick's control software. This works because our peripherals are Memory mapped and can thus be accessed from the HPS. This can be done using the following implementation to initialize the hardware:

```
#include "hw_headers/soc_system.h"
//...
//Initialize hardware
joysticks_dev joysticks = joysticks_inst((void *) HPS_0_ARM_A9_0_MCP3204_0_BASE);
joysticks_init(&joysticks);
```

3 Application Architecture

The application we chose to write for this lab is a *PONG* game using `ncurses`. To do that we chose to follow a pseudo-object oriented structure. We thus separated our application into 3 different files

and our main application file. These consist of the `Paddle` structure, which is used to model the paddles controlled by the player, the `Ball`, which models the ball that will be hit around on the screen and a `Game` which contains all of the elements needed to play a round of *PONG*, like the two paddles, the ball and each player's score.

3.1 Programming with ncurses

The main part of application can be found in the `app.c` file. Here you will find most of the code related to handling the `UI`. `Ncurses` organises its applications by window. In our case our application has 2 windows, one containing the main game and one containing our main menu and score board. Our secondary window will simply at first contain its own loop, where the player will select which game mode he wants to play in, and then switches to displaying the score contained in the current `Game` instance. This window is refreshed every time the score is updated. Our main window will display both of our `Paddle` instances and the `Ball` instance. This will be done quite regularly every time one of the paddles or the ball is updated. To avoid jittering, we only update the bottom and top symbols used to draw the paddles. An other problem is the speed of the ball, which at first was extremely high. To mitigate that, we put the process to sleep for $\frac{1}{15}$ sec on every update, limiting the refresh rate to 15Hz. This can be done using the `usleep` function as follows:

```
#define SLEEP_DURATION_US ((1.0/15.0) * 1000000)

//...

//Sleep for a while to avoid an excessive refresh rate
usleep(SLEEP_DURATION_US);
```

3.2 Handling user input

To handle the user input, we decided to use the joysticks. This was actually easier than using the traditional `ncurses` terminal input, since it allowed us to not have to handle prompting the user since the joysticks input can simply be checked by polling the MCP3204 channels. In the main menu the right joystick can be moved up and down to select different menu options, then moving the right joystick up will confirm the selection, this can, for example, be implemented with the following code:

```
//Get user input
uint32_t jstck_val_l = joysticks_read_left_vertical(joysticks);
uint32_t jstck_val_r = joysticks_read_right_vertical(joysticks);

//Left joystick is all the way up
if(jstck_val_l == JOYSTICKS_MAX_VALUE) {
    highlight = highlight == 0 ? 0 : --highlight;
}
//Left joystick is all the way down
else if(jstck_val_l == JOYSTICKS_MIN_VALUE)
    highlight = (highlight == N_MENU_OPTIONS - 1) ? highlight : ++highlight;
}

//Check for selection confirmation
```

```
if(jstck_val_r == JOYSTICKS_MAX_VALUE) {
    return highlight % N_MEU_OPTIONS;
}
```

Once in the game, the left joystick will control the left paddle and the right joystick the right one. The overall structure of the project is illustrated in figure 3.

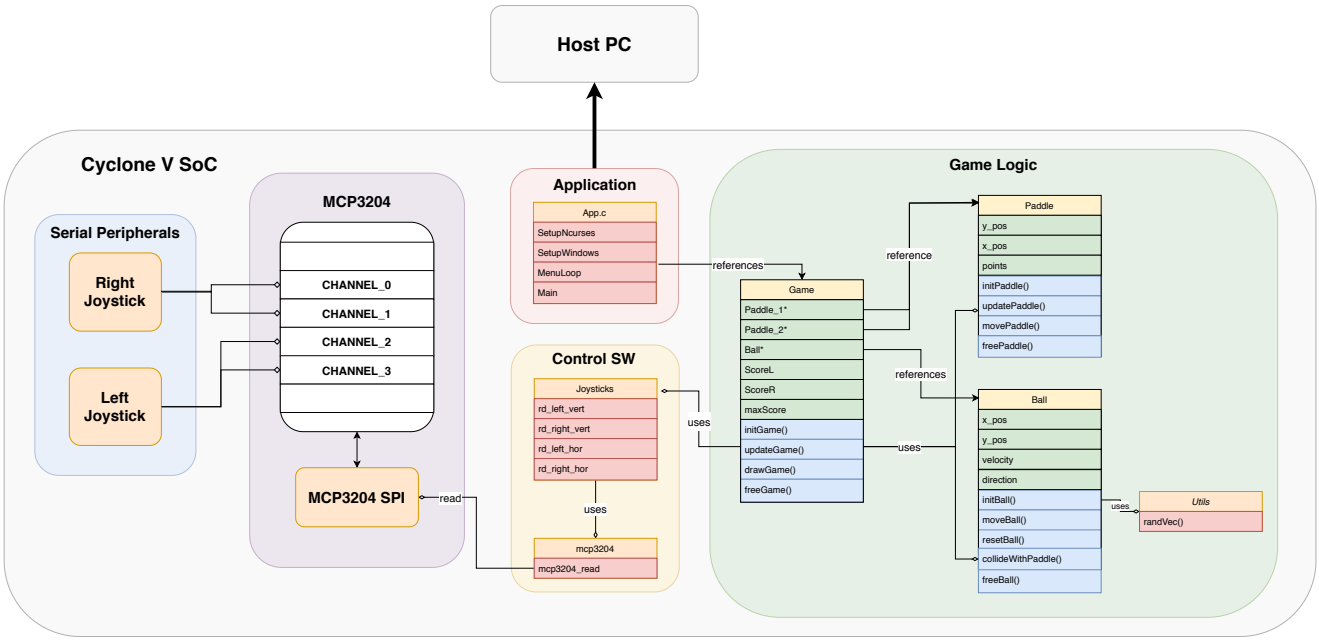

Figure 3: Overall structure of the project.

4 Conclusion

This project help introduce us to programming for an embedded linux system. It was also a nice excuse to learn how to program graphical terminal applications using `ncurses`. The main difficulty for this project was in getting our application to run on the target device. A lot of trial-and-error was needed to figure out how to get such a large application (compared to what we had done in previous labs) using an external library to compile for ARM. The final solution (compiling the application directly on the board) showed to work the best in our case. An other big difficulty came from the fact that none of my computers are capable of running the class's VM correctly (one is incapable of seeing USB peripherals when VTx is active, an other one's quartus compiled all of the Qsys systems with a *time_limited* suffix that made it impossible to convert to a raw binary format and the last one only has a single USB-C, which doesn't help when we want to plug in the 3 cables needed to connect to the board). This last problem has been a big hurdle during the entire semester.

A APPENDIX: Game source code

Here you will find the source code of the entire game

A.1 app.c

```
#include <stdlib.h>
#include <unistd.h>
#include <ncurses.h>
#include <stdio.h>

#include "hw_headers/soc_system.h"
#include "game.h"

#ifndef TEST
    #include "joysticks.h"
#endif

#define WINDOW_SPACING (1)
#define N_MENU_OPTIONS (3)
#define ENTER (10)
#define SLEEP_DURATION_US ((int)((1.0/15.0) * 1000000))

void initCurses() {
    //Init ncurses screen
    initscr();

    //Remove all char echo
    noecho();
}

WINDOW** setupWindows() {
    //Create a new window
    int height, width, init_y, init_x;

    //Retrieve host terminal size
    getmaxyx(stdscr, height, width);
    getbegyx(stdscr, init_y, init_x);

    //Offset by a bit
    init_y += 2;
    init_x += 4;

    //Split the screen into two windows 1/4 -- 3/4
    int mainHeight = 3 * (height - init_y) / 4;
    int mainWidth = width - (init_x * 2);

    int secHeight = height - WINDOW_SPACING - mainHeight - init_y;
```

```
    int secWidth = mainWidth;

    //Create the main window
    WINDOW* mainWin = newwin(mainHeight, mainWidth, init_y, init_x);

    //Move cursur down a bit
    init_y += WINDOW_SPACING + mainHeight;

    //Create secondary window
    WINDOW* secWin = newwin(secHeight, secWidth, init_y, init_x);

    //Refresh screen
    refresh();

    //Draw window borders
    box(mainWin, 0, 0);
    box(secWin, 0, 0);
    wrefresh(mainWin);
    wrefresh(secWin);

    WINDOW** result = calloc(2, sizeof(WINDOW*));
    result[0] = mainWin;
    result[1] = secWin;

    return result;
}

GameMode menuScreen(WINDOW* secWin, joysticks_dev* joysticks) {
    //Clear the screen
    wclear(secWin);
    box(secWin, 0, 0);

    //Different menu options
    char* menuOptions[N_MENU_OPTIONS] = {"1 Player", "2 Players", "CPU vs. CPU"};
    int choice = -1;
    int highlight = 0;

    //Print out menu title
    mvwprintw(secWin, 1, 1, "Main menu: Select your game mode");

    //Menu loop
    while(1) {
        //Print out all menu options
        for(int i = 0; i < N_MENU_OPTIONS; ++i) {
            if(highlight == i) {
                //Highlight the current option
                wattron(secWin, A_REVERSE);
            }
```

```
            //Print the menu option on screen and unhighlight it
            if((i + 3) >= getmaxy(secWin)) {
                mvwprintw(secWin, i + 2, getmaxx(secWin) /2, menuOptions[i]);
            } else {
                mvwprintw(secWin, i + 3, 3, menuOptions[i]);
            }
            wattroff(secWin, A_REVERSE);
        }
        //Refresh the window
        wrefresh(secWin);

        if(joysticks == NULL) {
            //=======================================================
            int c = wgetch(secWin);

            if(c == (int)'w') {
                highlight = highlight == 0 ? 0 : --highlight;
            } else if(c == (int)'s') {
                highlight = (highlight == N_MENU_OPTIONS - 1) ? highlight : ++highlight;
            }

            //Check for selection confirmation
            if(c == ENTER) {
                return highlight % N_MENU_OPTIONS;
            }
            //=======================================================
        } else {
        #ifndef TEST
            //Get user input
            uint32_t jstck_val_l = joysticks_read_left_vertical(joysticks);
            uint32_t jstck_val_r = joysticks_read_right_vertical(joysticks);

            //Left joystick is all the way up
            if(jstck_val_l == JOYSTICKS_MAX_VALUE) {
                highlight = highlight == 0 ? 0 : --highlight;
            }
            //Left joystick is all the way down
            else if(jstck_val_l == JOYSTICKS_MIN_VALUE) {
                highlight = (highlight == N_MENU_OPTIONS - 1) ? highlight : ++highlight;
            }

            //Check for selection confirmation
            if(jstck_val_r == JOYSTICKS_MAX_VALUE) {
                return highlight % 2;
            }
        #endif
        }
    }
```

```
}

void drawScores(Game_t* game, WINDOW* secWin) {
    //Sanity Check
    if(game != NULL && secWin != NULL) {
        //Compute drawing coordinates
        int mid_y = (getmaxy(secWin)) / 2;
        int first_x = (getmaxx(secWin)) / 3;
        int second_x = 2 * first_x;

        //Convert the scores to character strings
        char score1[3];
        char score2[3];
        sprintf(score1, "%d", (game->scoreL));
        sprintf(score2, "%d", (game->scoreR));

        //Draw the scores
        mvwprintw(secWin, mid_y, first_x, score1);
        mvwprintw(secWin, mid_y, first_x + ((second_x - first_x) / 2), "-");
        mvwprintw(secWin, mid_y, second_x, score2);
    }
}

void playGame(Game_t* game, WINDOW* mainWin, WINDOW* secWin, joysticks_dev* joysticks) {
    //Clear both windows
    wclear(mainWin);
    wclear(secWin);
    box(mainWin, 0, 0);
    box(secWin, 0, 0);

    //Disable input for the secondary window
    keypad(secWin, false);

    //Init Score Board
    mvwprintw(secWin, 1, 1, "Score:");

    //Main game loop
    while(1) {
        if(game->scoreL >= game->maxScore ||
            game->scoreR >= game->maxScore) {
                break;
        }
        //Show the scores
        drawScores(game, secWin);
        wrefresh(secWin);

        //Update the game
        updateGame(game, mainWin, joysticks);
```

```
        //Draw the game
        drawGame(game, mainWin);

        //Refresh the window
        wrefresh(mainWin);

        //Sleep for a while to avoid an excessive refresh rate
        usleep(SLEEP_DURATION_US);
    }
}

int main(int argc, char** argv) {
    //Initialize ncurses
    initCurses();

#ifndef TEST
    //Initialize hardware
    joysticks_dev joysticks = joysticks_inst((void *) HPS_0_ARM_A9_0_MCP3204_0_BASE);
    joysticks_init(&joysticks);
#endif
    //Setup main window
    WINDOW** wins = setupWindows();
    WINDOW* mainWin = wins[0];
    WINDOW* secWin = wins[1];

    //Free useless temp result array
    free(wins);
    wins = NULL;

    //==============UPDATE==================
    //Show the menu screen
    GameMode mode = menuScreen(secWin, /*&joysticks*/ NULL);

    //Create a new game
    Game_t* game = initGame(mainWin, mode);

    //Show the main screen
    playGame(game, mainWin, secWin, /*&joysticks*/NULL);
    //======================================
    char winner = game->scoreL > game->scoreR;

    //Free the game
    freeGame(game);
    game = NULL;

    //End ncurses
    endwin();
```

```
    //Show who won
    if(winner) {
        printf("Player 1 is the winner ! \n");
    } else {
        printf("Player 2 is the winner ! \n");
    }

    return EXIT_SUCCESS;
}
```

A.2 game.c

```
#include <stdlib.h>
#include <ncurses.h>
#include "game.h"

#define EDGE_OFFSET 2
#define MAX_SCORE 5

Game_t* initGame(WINDOW* mainWin, GameMode mode) {
    //Compute the paddle positions
    int start_x = 0;
    int end_x = getmaxx(mainWin);

    nodelay(mainWin, true);

    //Initialize both paddles
    Paddle_t* paddle1 = initPaddle(mainWin, start_x + EDGE_OFFSET);
    Paddle_t* paddle2 = initPaddle(mainWin, end_x - EDGE_OFFSET - 1);
    GameMode g_mode = mode; //mode is 0 if 1 PLAYER & 1 if 2 PLAYERS

    //Initialize the ball
    int mid_x = (end_x - start_x) / 2;
    int mid_y = (getmaxy(mainWin)) / 2;
    Ball_t* ball = initBall(mid_x, mid_y, mainWin);

    //Allocate and initialize the game
    Game_t* game = calloc(1, sizeof(Game_t));
    Game_t game_ = {paddle1, paddle2, g_mode, ball, 0, 0, MAX_SCORE};

    *game = game_;
    return game;
}

void updatePaddleAI(Game_t* game, WINDOW* mainWin, int player) {
    //Check the ball's next position
    int new_y = game->ball->y_pos + (game->ball->direction[1] * game->ball->velocity);
```

```
    int y_pos;
    Paddle_t* paddle;
    char moveCond;

    if(player == 1) {
        y_pos = game->paddle_1->y_pos;
        paddle = game->paddle_1;
        moveCond = game->ball->direction[0] < 0;
    } else {
        y_pos = game->paddle_2->y_pos;
        paddle = game->paddle_2;
        moveCond = game->ball->direction[0] > 0;
    }

    //Only move the paddle if the ball is coming in its direction
    if(moveCond) {
        //Move towards the ball
        if(new_y > y_pos) {
            movePaddle(paddle, mainWin, DOWN);
        }

        if(new_y < y_pos) {
            movePaddle(paddle, mainWin, UP);
        }
    }
}

void updateGame(Game_t* game, WINDOW* mainWin, joysticks_dev* joysticks) {
    //Clear the frame
    wclear(mainWin);
    box(mainWin, 0, 0);

    //Update both paddles
    if(game->mode == CPU_V_CPU) {
        updatePaddleAI(game, mainWin, 1);
    } else {
        updatePaddle(game->paddle_1, mainWin, 1, joysticks);
    }

    if(game->mode != P1_V_P2) {
        updatePaddleAI(game, mainWin, 2);
    } else {
        updatePaddle(game->paddle_2, mainWin, 2, joysticks);
    }

    //Udpate the ball
    moveBall(game->ball, mainWin, game->paddle_1, game->paddle_2);
```

```
    //Update game score
    game->scoreL = game->paddle_1->points;
    game->scoreR = game->paddle_2->points;
}

void drawGame(Game_t* game, WINDOW* mainWin) {
    //Draw both paddles
    for(int i = 0; i < PADDLE_SIZE; ++i) {
        mvwprintw(mainWin, game->paddle_1->y_pos - (PADDLE_SIZE / 2) + i,
                        game->paddle_1->x_pos, "#");
        mvwprintw(mainWin, game->paddle_2->y_pos - (PADDLE_SIZE / 2) + i,
                        game->paddle_2->x_pos, "#");
    }

    mvwprintw(mainWin, game->ball->y_pos, game->ball->x_pos, "*");
}

void freeGame(Game_t* game) {
    //Free both paddles
    freePaddle(game->paddle_1);
    freePaddle(game->paddle_2);
    game->paddle_1 = NULL;
    game->paddle_2 = NULL;

    //Free the ball
    freeBall(game->ball);
    game->ball = NULL;

    //Free the game structure
    free(game);
}
```

A.3 paddle.c

```
#include <stdlib.h>
#include "paddle.h"

#define MOVED 1
#define NO_MOVE 0

Paddle_t* initPaddle(WINDOW* mainWin, int init_x) {
    //Allocate memory for the paddle
    Paddle_t* paddle = calloc(1, sizeof(Paddle_t));

    int mid_y = (getmaxy(mainWin)) / 2;

    //Create the paddle
```

```
    Paddle_t paddle_ = {
        mid_y,
        init_x,
        0
    };

    *paddle = paddle_;

    return paddle;
}

void movePaddle(Paddle_t* paddle, WINDOW* mainWin, MoveDir dir) {
    switch(dir) {
        case UP:
            //Check that our paddle is still in the window bounds
            if(paddle->y_pos - 3 > 0) {
                paddle->y_pos -= 1;
            }
            break;
        case DOWN:
            //Check that our paddle is still in the window bounds
            if(paddle->y_pos + 2 < getmaxy(mainWin)) {
                paddle->y_pos += 1;
            }
            break;
        default:
            break;
    }
}

void updatePaddle(Paddle_t* paddle, WINDOW* mainWin, int player, joysticks_dev* joysticks) {
    switch(player) {
        case 1:
            if(joysticks != NULL) {
            #ifndef TEST
                //Check left joystick
                uint32_t jstck_val_l = joysticks_read_left_vertical(joysticks);

                if(jstck_val_l == JOYSTICKS_MAX_VALUE) {
                    //Move the paddle up
                    movePaddle(paddle, mainWin, UP);
                } else if(jstck_val_l == JOYSTICKS_MIN_VALUE){
                    //Move the paddle down
                    movePaddle(paddle, mainWin, DOWN);
                }
            #endif
            } else {
                int c = wgetch(mainWin);
```

```
                if(c == (int)'w') {
                    //Move the paddle up
                    movePaddle(paddle, mainWin, UP);
                } else if(c == (int)'s') {
                    //Move the paddle down
                    movePaddle(paddle, mainWin, DOWN);
                }
            }
            break;
        case 2:
            if(joysticks != NULL) {
            #ifndef TEST
                //Check right joystick
                uint32_t jstck_val_r = joysticks_read_right_vertical(joysticks);

                if(jstck_val_r == JOYSTICKS_MAX_VALUE) {
                    //Move the paddle up
                    movePaddle(paddle, mainWin, UP);
                } else if(jstck_val_r == JOYSTICKS_MIN_VALUE){
                    //Move the paddle down
                    movePaddle(paddle, mainWin, DOWN);
                }
            #endif
            } else {
                int c = wgetch(mainWin);

                if(c == (int)'i') {
                    //Move the paddle up
                    movePaddle(paddle, mainWin, UP);
                } else if(c == (int)'k') {
                    //Move the paddle down
                    movePaddle(paddle, mainWin, DOWN);
                }
            }
            break;
        default:
            break;
    }
}

void freePaddle(Paddle_t* paddle) {
    //Deallocate memory associated to the paddle
    free(paddle);
}
```

A.4 ball.c

```c
#include <time.h>
#include <stdlib.h>
#include <unistd.h>
#include "utils.h"
#include "ball.h"

#define MAX_VELOCITY 5
#define DIR_MODULUS 3
#define A_LITTLE 1000000

Ball_t* initBall(int center_x, int center_y, WINDOW* mainWin) {
    //Initialize random number generator
    srand(time(NULL));

    //Allocate memory for the ball
    Ball_t* ball = calloc(1, sizeof(Ball_t));

    //Initialize a random direction
    int* dir = calloc(2, sizeof(int));
    randVec(dir, DIR_MODULUS);

    //Create the ball
    Ball_t ball_ = {
        center_x,
        center_y,
        1, //rand() % MAX_VELOCITY,
        dir
    };
    //Set pointer to be on ball
    *ball = ball_;
    return ball;
}

void resetBallPos(Ball_t* ball, WINDOW* mainWin) {
    //Reset the ball's position
    ball->x_pos = getmaxx(mainWin) / 2;
    ball->y_pos = getmaxy(mainWin) / 2;

    //Compute a random initial direction
    randVec(ball->direction, DIR_MODULUS);
}

void collideWithPaddle(Ball_t* ball, Paddle_t* paddle, int* new_x, int* new_y) {
    while(*new_x == paddle->x_pos) {
        //Update direction
        int x_dir = -(ball->direction[0]);
```

```
        randVec(ball->direction, DIR_MODULUS);
        ball->direction[0] = x_dir;

        //Compute new postitions
        *new_x = ball->x_pos + (ball->direction[0] * ball->velocity);
        *new_y = ball->y_pos + (ball->direction[1] * ball->velocity);
    }
}

void moveBall(Ball_t* ball, WINDOW* mainWin, Paddle_t* paddle_1, Paddle_t* paddle_2) {

    //Compute the new postition and set it
    int new_x = ball->x_pos + (ball->direction[0] * ball->velocity);
    int new_y = ball->y_pos + (ball->direction[1] * ball->velocity);

    //Check for horizontal wall collisions
    if(new_x <= 0) {
        //Increase right score
        paddle_2->points += 1;

        //Reset the ball's position
        ball->x_pos = getmaxx(mainWin) / 2;
        ball->y_pos = getmaxy(mainWin) / 2;

        //Compute a random initial direction
        randVec(ball->direction, DIR_MODULUS);

        //Wait a little
        usleep(A_LITTLE);

        return;
    }

    if(new_x >= getmaxx(mainWin)) {
        //Increase right score
        paddle_1->points += 1;

        //Reset the ball's position
        ball->x_pos = getmaxx(mainWin) / 2;
        ball->y_pos = getmaxy(mainWin) / 2;

        //Compute a random initial direction
        randVec(ball->direction, DIR_MODULUS);

        //Wait a little
        usleep(A_LITTLE);

        return;
```

```
    }
    //Check for vertical wall collisions
    if(new_y <= 0 || new_y >= getmaxy(mainWin)) {
        ball->direction[1] *= -1;
        new_y = ball->y_pos + (ball->direction[1] * ball->velocity);
    }

    if(new_x == paddle_1->x_pos) {
        //Check for left paddle collisions
        if(new_y >= (paddle_1->y_pos - (PADDLE_SIZE / 2)) &&
            new_y <= (paddle_1->y_pos + (PADDLE_SIZE / 2))) {

            //Collide with the paddle
            collideWithPaddle(ball, paddle_1, &new_x, &new_y);
        }
    }

    if(new_x == paddle_2->x_pos) {
        //Check for right paddle collisions
        if(new_y >= (paddle_2->y_pos - (PADDLE_SIZE / 2)) &&
            new_y <= (paddle_2->y_pos + (PADDLE_SIZE / 2))) {

            //Collide with the paddle
            collideWithPaddle(ball, paddle_2, &new_x, &new_y);
        }
    }

    ball->x_pos = new_x;
    ball->y_pos = new_y;
}

void freeBall(Ball_t* ball) {
    free(ball->direction);
    ball->direction = NULL;

    //Free the ball structure
    free(ball);
}
```

A.5 utils.c

```
#include <math.h>
#include <stdlib.h>
#include "utils.h"

void randVec(int* vec, int mod) {
    vec[0] = rand() % mod;
    vec[1] = rand() % mod;
```

```
    //Make sure that the direction is valid
    while(vec[0] == 0) {
        vec[0] = rand() % mod;
        vec[1] = rand() % mod;
    }
}
```

A.6 mcp3204.c

```
#include <assert.h>

#ifndef TEST
    #include "io_custom.h"
#endif
#include "mcp3204.h"
#define MCP3204_NUM_CHANNELS (4)

mcp3204_dev mcp3204_inst(void *base) {
    mcp3204_dev dev;
    dev.base = base;

    return dev;
}

void mcp3204_init(mcp3204_dev *dev) {
    return;
}

uint32_t mcp3204_read(mcp3204_dev *dev, uint32_t channel) {
    #ifndef TEST
    switch(channel) {
        case 0:
            return ioc_read_32(dev->base, MCP3204_CHANNEL_0_OFST);
        case 1:
            return ioc_read_32(dev->base, MCP3204_CHANNEL_1_OFST);
        case 2:
            return ioc_read_32(dev->base, MCP3204_CHANNEL_2_OFST);
        case 3:
            return ioc_read_32(dev->base, MCP3204_CHANNEL_3_OFST);
        default:
            return 0;
    }
    #else
    return 0;
    #endif
}
```