University of Liège

ELEN0040 : Digital Electronics

Project Report

Authors :
Grégoire Hendrix s214754
Louis Hogge s192814
Arnaud Innaurato s211234
Macha Krumm s212449

Professor : J-M Redouté
Teaching Assistants : A. Fyon
A. Halin
Year : 2022-2023

1 Introduction

This project aims to implement a simplified version of the game *pong* by using a CPLD. The goal is to bounce the ball on the paddle. The simplified version consists of a single paddle that moves horizontally and a ball that moves vertically, the player has the ability to control the movement of the paddle in order to bounce the ball up. When the ball reaches the top, it bounces back in another column, which is drawn randomly. This version of the game was necessary in order to fit our game on this kind of microchips with only 160 logic gates. The game stops when the player fails 3 times to make the ball bounces on the paddle.

We chose to use a 5x7 LED matrix where the paddle moves in the row of 7 LEDs and the ball in the column of 5 LEDs. The paddle has a width of 2 LEDs and the ball has a size of only 1 LED.

2 Hardware

Here is the electrical schematic of our project :

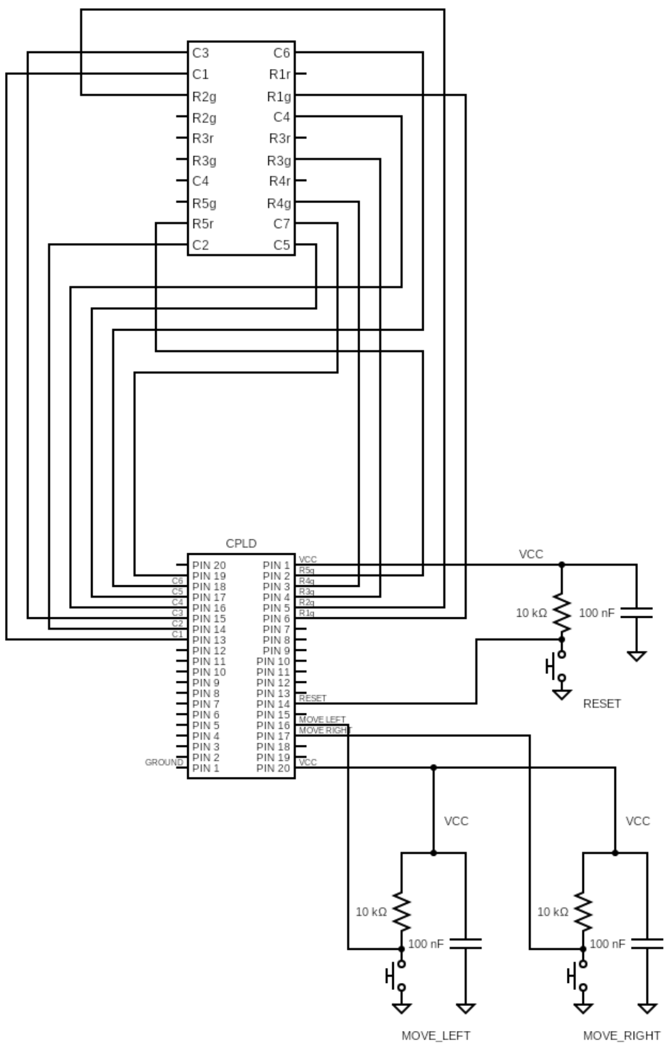


Figure 1 – Electrical schematics

The player needs three buttons to control the game. One to move the paddle to the left, another one to move it to the right and the last one to start/reset the game. Here is the explanation of how they work :

- **Move buttons :** The buttons have to be pushed to move the paddle. It means that it detects when MOVE_LEFT = '0' or if MOVE_RIGHT = '0'. As represented in the schematic, when a button is pushed, the input voltage is the ground so it is why we have '0'. To make sure that we do not have any voltage bounces, each button is in series with a 10 $k\Omega$ resistance and in parallel with a 100 nF capacitor.

- **Reset button :** This button has to be pushed to start playing, to play again if you have lost or to reset the game while you are playing. It is also in series with a 10 $k\Omega$ resistance and in parallel with a 100 nF capacitor. It works like the two other buttons : if you push it, the voltage goes to the ground and the input signal START = '0'. When reset button is pressed, the game starts again with 3 lives.

For the LED matrix, we used a bi-color 5 x 7 LED matrix (TBC20-11EGWA). We used 12 of the 20 pins available. For the 4 first rows, we used the green LEDs and for the last row (the bottom one) we used the red LEDs.

LED to chip pin assignment		
Name	pin on the CPLD	location on the chip
Row 1 green	pin 12	pin 1
Row 2 green	pin 10	pin 2
Row 3 green	pin 8	pin 3
Row 4 green	pin 6	pin 4
Row 5 red	pin 4	pin 5
Column 1	pin 15	pin 56
Column 2	pin 13	pin 58
Column 3	pin 11	pin 59
Column 4	pin 9	pin 60
Column 5	pin 7	pin 61
Column 6	pin 5	pin 62
Column 7	pin 3	pin 63

3 Main entity and inputs/outputs

Here is the main entity of our game :

```vhdl
entity pong is
    generic(
        -- Constants
        SCREEN_WIDTH: integer := 7;
        SCREEN_HEIGHT: integer := 5;
        PADDLE_WIDTH: integer := 2;
        PADDLE_Y: integer := 0;
          NUMBER_LIVES: integer := 3);
    Port(
        -- Input signals
        CLK_SLOW: in STD_LOGIC;
        CLK_FAST: in STD_LOGIC;

        MOVE_LEFT: in STD_LOGIC;
        MOVE_RIGHT: in STD_LOGIC;

        START: in STD_LOGIC;

        -- Output signals
        LED_ROW: out STD_LOGIC_VECTOR(SCREEN_HEIGHT - 1 downto 0);
        LED_COL: out STD_LOGIC_VECTOR(SCREEN_WIDTH - 1 downto 0);

        LED_LIVES: out STD_LOGIC_VECTOR(NUMBER_LIVES - 1 downto 0));
end pong;
```

3.1 Constants

- SCREEN_WIDTH : equals to the LED matrix's width (7)
- SCREEN_HEIGHT : equals to the LED matrix's height (5)
- PADDLE_WIDTH : the paddle's size has been defined to 2 in this game

- PADDLE_Y : the paddle stays at the bottom so its position is equal to 0
- LIVES : the number of lives has been chosen to be equal to 3

3.2 Inputs

- CLOCK_SLOW : the slowest clock from the CPLD
- CLOCK_FAST : the fastest clock from the CPLD
- MOVE_LEFT : button that moves the paddle one LED to the left
- MOVE_RIGHT : button that moves the paddle one LED to the right
- START : button that starts or restarts the game with 3 lives

3.3 Outputs

- LED_ROW : logic vector which contains the lines of the LED matrix
- LED_COL : logic vector which contains the columns of the LED matrix
- LIVES : logic vector that contains the number of remaining lives

4 Game structure

This schematic explains visually the different steps of the game operations :

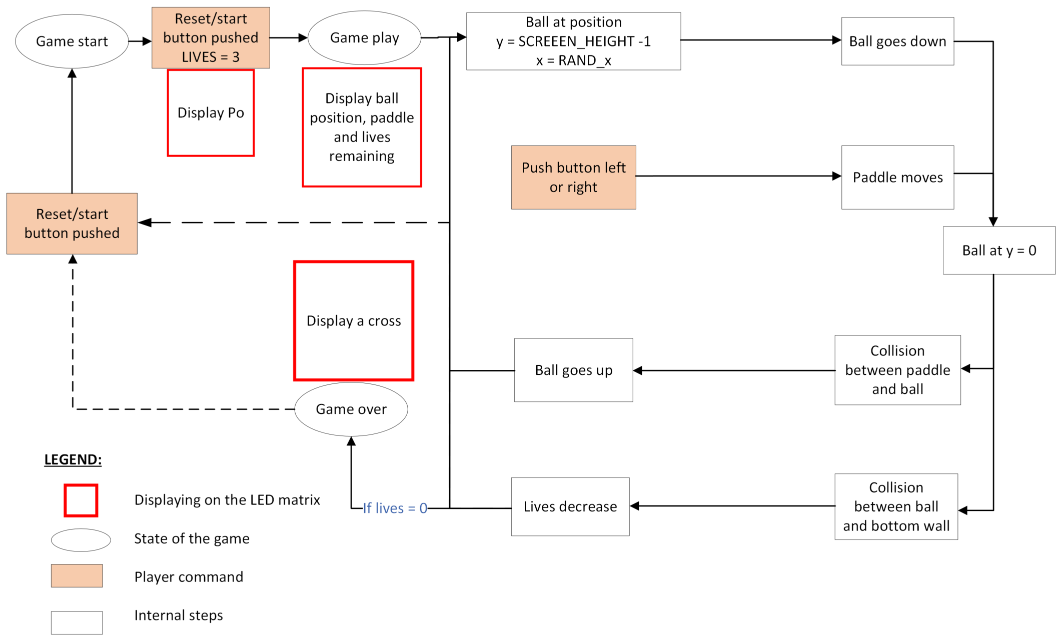

Game start
Reset/start button pushed LIVES = 3
Display Po
Game play
Display ball position, paddle and lives remaining
Ball at position y = SCREEEN_HEIGHT -1 x = RAND_x
Ball goes down
Push button left or right
Paddle moves
Ball at y = 0
Reset/start button pushed
Display a cross
Game over
Ball goes up
Collision between paddle and ball
Lives decrease
Collision between ball and bottom wall
If lives = 0
LEGEND:
Displaying on the LED matrix
State of the game
Player command
Internal steps

FIGURE 2 – Structure of the game

5 Code structure

NB : All the code is given in the section 7.

- **Variables :**
  - BALL_X : the horizontal position of the ball, which can take values between 0 and 6

  - BALL_Y : the vertical position of the ball, which can take values between 0 and 4

- PADDLE_X : the horizontal position of the left corner of the paddle, which can take values between 0 and 5

- RAND_X : a number generated to randomly determine the position of the ball when it hits the top wall

- BALL_DIRECTION : direction of the ball's movement, can be *up* or *down*

- GAME_STATE : indicates the state of the game, it can be *game_start* (before the start of a game), *game_play* (when a game is in progress) or *game_over* (when you have lost)

- LIVES : are decremented when the ball hits the bottom wall, take values from 3 to 0

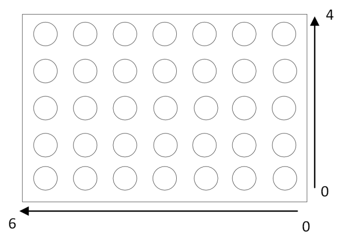


FIGURE 3 – Visualization of the coordinates chosen for the LED matrix

- **Processes :**
  - *Random* : generates a number to randomly determine the position of the ball when it hits the top wall. It is triggered at the rising edge of the CLK_FAST. It goes from 0 to 6 and when it's value is equal to 6, it returns to 0. The computation of the next value of our ball position on the top wall is only done triggering the CLK_SLOW which assure that our number is random.

  - *paddle movement* : if one of the button (left or right) is pressed, we check if the paddle can go in the desired direction. If so, the paddle moves in the direction wanted. It is triggered by the CLK_SLOW.

  - *ball movement* : Triggered by the CLK_SLOW, this process handles the movement of the ball and the collisions with the paddle. There are 4 types of collisions in this implementation :
    - paddle collision : if the ball is at the bottom wall, we check if its horizontal position corresponds to one of the two points of the paddle. If it collides, the ball will go up by changing BALL_DIRECTION to *up*

    - bottom collision : if the ball is at the bottom wall but there is not a collision with the paddle, LIVES is decremented and the ball will start at a random position at the top wall

    - top collision : when the ball is at the top, its position is updated with random position x and the ball will start going down from that new position

    - no collision : handles the cases when the ball is between the top and the bottom wall. It checks the BALL_DIRECTION and increments BALL_Y if it is *up* or decrements it if it is *down*

    When the reset button is pressed, this process is responsible of giving a new random horizontal position at the top row.

- *display* : Before the game starts, '**P**' is shown on the LED matrix. When the game is playing, it displays the correct LEDs and when the game is over, it displays a cross on the matrix. It is triggered by the `CLK_FAST` to have the correct fluency for the LEDs.

- *state management* : manages the states of the games. When the start button is pressed, it starts the game by putting `GAME_STATE` at *game_play*. The game goes on until there are no more lives remaining. Once the game is done, `GAME_STATE` is set at *game_over*.

6 Simulation

This section shows a simulation of our game done in ModelSim. It shows the proper functioning of `LIVES`. The `CLK_SLOW` is set to 10ns and the `CLK_FAST` to 0.1ns to correctly display the LEDs. To simulate the movement of the paddle, the clock of `MOVE_LEFT` is set at 0.5 sec and the clock of `MOVE_RIGHT` at 0.4 sec. `GAME_STATE` is at game_play. The simulation lasts 3 seconds. It behaves like expected.

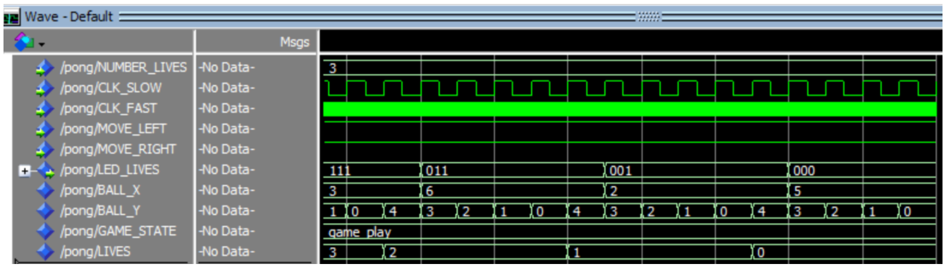

Wave - Default
Msgs
/pong/NUMBER_LIVES -No Data- 3
/pong/CLK_SLOW -No Data-
/pong/CLK_FAST -No Data-
/pong/MOVE_LEFT -No Data-
/pong/MOVE_RIGHT -No Data-
/pong/LED_LIVES -No Data- 111 011 001 000
/pong/BALL_X -No Data- 3 6 2 5
/pong/BALL_Y -No Data- 1 0 4 3 2 1 0 4 3 2 1 0 4 3 2 1 0
/pong/GAME_STATE -No Data- game_play
/pong/LIVES -No Data- 3 2 1 0

FIGURE 4 – Simulation

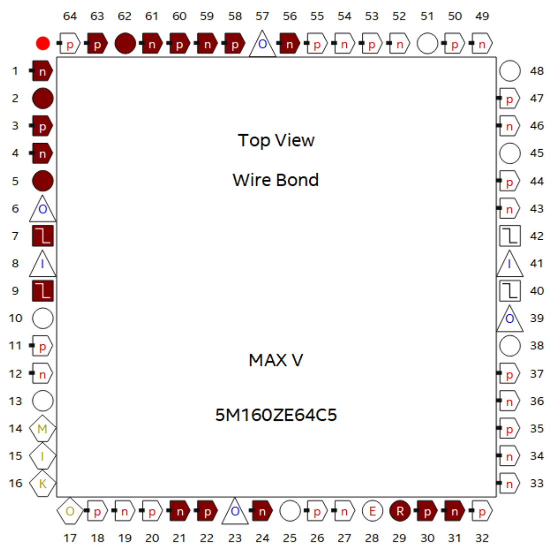
It shows that `LIVES` decreases and that `LED_LIVES` is correctly decremented. The `LIVES` go correctly from 3 to 0 and are decremented when `BALL_Y` is at 0. `BALL_X` is a random number and `BALL_Y` is between 0 and 4.

7 Code

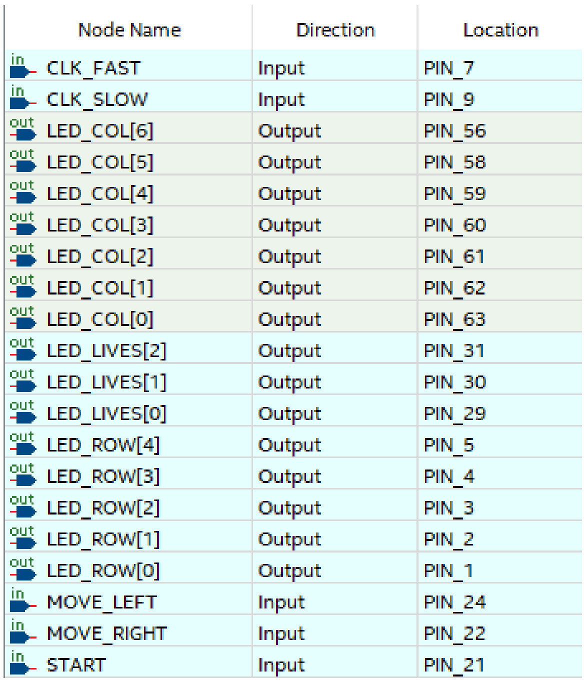
The code we implemented uses 94 logic gates of our chip , which corresponds to 59% of its capacity. We use 20 pins of the 54 available :

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

-- Pong game
entity pong is
    generic(
        -- Constants
        SCREEN_WIDTH: integer := 7;
        SCREEN_HEIGHT: integer := 5;
        PADDLE_WIDTH: integer := 2;
        PADDLE_Y: integer := 0;
        NUMBER_LIVES: integer := 3);
    Port(
        -- Input signals
        CLK_SLOW: in STD_LOGIC;
        CLK_FAST: in STD_LOGIC;

        MOVE_LEFT: in STD_LOGIC;
        MOVE_RIGHT: in STD_LOGIC;

        START: in STD_LOGIC;

        -- Output signals
        LED_ROW: out STD_LOGIC_VECTOR(SCREEN_HEIGHT - 1 downto 0);
        LED_COL: out STD_LOGIC_VECTOR(SCREEN_WIDTH - 1 downto 0);

        LED_LIVES: out STD_LOGIC_VECTOR(NUMBER_LIVES - 1 downto 0));
end pong;

architecture behavioral of pong is
    -- Internal signals for game logic
    signal BALL_X: integer range 0 to SCREEN_WIDTH - 1 := 0;
    signal BALL_Y: integer range 0 to SCREEN_HEIGHT - 1 := SCREEN_HEIGHT - 1;

    signal PADDLE_X: integer range 0 to SCREEN_WIDTH - PADDLE_WIDTH := 0;

    signal RAND_X: integer range 0 to SCREEN_WIDTH - 1 := 0;
    type direction is (up, down);
    signal BALL_DIRECTION: direction := down;

    type state is (game_start, game_play, game_over);
    signal GAME_STATE: state := game_start;

    signal LIVES: integer range 0 to NUMBER_LIVES := NUMBER_LIVES;
begin
    -- Random number generator process
    random: process(CLK_FAST)
    begin
        if rising_edge(CLK_FAST) then
            -- Playing the game
            if GAME_STATE = game_play then
                if(RAND_X = SCREEN_WIDTH - 1) then
                    RAND_X <= 0;
                else
                    RAND_X <= RAND_X + 1;
                end if;
            end if;
        end if;
    end process random;

    -- Paddle movement process
    paddle_movement: process(CLK_SLOW)
    begin
        if rising_edge(CLK_SLOW) then
            -- Playing the game
            if GAME_STATE = game_play then
                if MOVE_LEFT = '0' then
                    if not (PADDLE_X = 0) then
                        PADDLE_X <= PADDLE_X - 1;
                    end if;
                elsif MOVE_RIGHT = '0' then
                    if not (PADDLE_X = SCREEN_WIDTH - PADDLE_WIDTH) then
                        PADDLE_X <= PADDLE_X + 1;
                    end if;
                end if;
            end if;
        end if;
    end process paddle_movement;

    -- Ball movement process
    ball_movement: process(CLK_SLOW)
    begin
        if rising_edge(CLK_SLOW) then
            -- Playing the game
            if GAME_STATE = game_play then
                if BALL_Y = 0 then
                    -- Paddle colision
                    if BALL_X = PADDLE_X or BALL_X = PADDLE_X+1 then
                        BALL_DIRECTION <= up;
                        BALL_Y <= BALL_Y + 1;
                    -- Bottom colision
                    else
                        LIVES <= LIVES - 1;
                        BALL_X <= RAND_X;
                        BALL_Y <= SCREEN_HEIGHT - 1;
                    end if;
                -- Top colision
                elsif BALL_Y = SCREEN_HEIGHT then
                    BALL_DIRECTION <= down;
                    BALL_X <= RAND_X;
                    BALL_Y <= BALL_Y - 1;
                -- No colision
                else
                    case BALL_DIRECTION is
                        when up =>
                            BALL_Y <= BALL_Y + 1;
                        when down =>
                            BALL_Y <= BALL_Y - 1;
                    end case;
                end if;
            end if;

            -- Set up the game for restart
            if GAME_STATE = game_start then
                BALL_Y <= SCREEN_HEIGHT - 1;
                LIVES <= NUMBER_LIVES;
            end if;
        end if;
    end process ball_movement;

    -- Display process
    display: process(CLK_FAST)
        -- Counter to display alternatively
        variable COUNTER : integer range 0 to 3 := 0;
    begin
        if rising_edge(CLK_FAST) then
            case GAME_STATE is
                -- Ready to play the game
                when game_start =>
                    -- Initialize control signals to draw "P"
                    if COUNTER = 3 then
                        LED_COL <= "0010000";
                        LED_ROW <= "10000";
                        COUNTER := 0;
                    end if;

                    if COUNTER = 2 then
                        LED_COL <= "0001100";
                        LED_ROW <= "01011";
                    end if;

                    if COUNTER = 1 then
                        LED_COL <= "0010100";
                        LED_ROW <="00011";
                    end if;

                    COUNTER:= COUNTER + 1;
                -- Playing the game
                when game_play =>
                    if LIVES /= 0 then
                        if COUNTER = 3 then
                            -- Paddle display
                            LED_COL <= "0000000";
                            LED_ROW <= "11110";

                            -- Set control signals for paddle
                            LED_COL(PADDLE_X) <= '1';
                            LED_COL(PADDLE_X + 1) <= '1'; -- Paddle width is 2

                            COUNTER := 0;
                        else
                            -- Ball display
                            LED_COL <= "0000000";
                            LED_ROW <= "11111";

                            -- Set control signals for ball
                            LED_COL(BALL_X) <= '1';
                            LED_ROW(BALL_Y) <= '0';

                            COUNTER := COUNTER + 1;
                        end if;
                    end if;
                -- Game lost
                when game_over =>
                    -- Initialize control signals to draw a cross
                    if COUNTER = 3 then
                        LED_COL <= "0100010";
                        LED_ROW <= "01110";
                        COUNTER := 0;
                    end if;

                    if COUNTER = 2 then
                        LED_COL <= "0010100";
                        LED_ROW <= "10101";
                    end if;

                    if COUNTER = 1 then
                        LED_COL <= "0001000";
                        LED_ROW <= "11011";
                    end if;

                    COUNTER := COUNTER + 1;
            end case;
        end if;
    end process display;

    state_management: process(CLK_SLOW)
    begin
        if rising_edge(CLK_SLOW) then
            case GAME_STATE is

                -- Ready to play the game
                when game_start =>
                    -- Start the game
                    if START = '0' then
                        GAME_STATE <= game_play;
                    end if;

                -- Playing the game
                when game_play =>
                    -- Lives management
                    for i in 0 to NUMBER_LIVES - 1 loop
                        if i < LIVES then
                            LED_LIVES(i) <= '1';
                        else
                            LED_LIVES(i) <= '0';
                        end if;
                    end loop;

                    -- Game lost
                    if LIVES = 0 then
                        GAME_STATE <= game_over;
                    end if;

                    -- Reset the game
                    if START = '0' then
                        GAME_STATE <= game_start;
                    end if;

               -- Game lost
               when game_over =>
                    -- Reset the game
                    if START = '0' then
                        GAME_STATE <= game_start;
                    end if;
            end case;
        end if;
    end process state_management;
end architecture behavioral;
```

8 Pin Planner



(a)



Node Name	Direction	Location
CLK_FAST	Input	PIN_7
CLK_SLOW	Input	PIN_9
LED_COL[6]	Output	PIN_56
LED_COL[5]	Output	PIN_58
LED_COL[4]	Output	PIN_59
LED_COL[3]	Output	PIN_60
LED_COL[2]	Output	PIN_61
LED_COL[1]	Output	PIN_62
LED_COL[0]	Output	PIN_63
LED_LIVES[2]	Output	PIN_31
LED_LIVES[1]	Output	PIN_30
LED_LIVES[0]	Output	PIN_29
LED_ROW[4]	Output	PIN_5
LED_ROW[3]	Output	PIN_4
LED_ROW[2]	Output	PIN_3
LED_ROW[1]	Output	PIN_2
LED_ROW[0]	Output	PIN_1
MOVE_LEFT	Input	PIN_24
MOVE_RIGHT	Input	PIN_22
START	Input	PIN_21

(b)

FIGURE 5 – Pin planner