FUNDAMENTAL OF DIGITAL SYSTEM FINAL PROJECT REPORT

DEPARTMENT OF ELECTRICAL ENGINEERING

UNIVERSITAS INDONESIA

MULTI-LEVEL PARKING SYSTEM

GROUP A6

ABDUL FIKIH KURNIA (2106731200)

AQIB RAHMAN (2106731226)

BERNANDA NAUTVAL R.I.W (2106708463)

RAFIE AMANDIO FAUZAN (2106731232)

PREFACE

We give thanks and praise to the presence of God YME who has given His grace and guidance so that we can complete the final project of Digital System Design Practicum entitled "Multi-Level Parking System".

In today's development, the development of technology will continue to grow, making technology an indispensable part of everyday life because technology has entered our lives. Technology must be used wisely and well to have a positive impact on life. This final project is expected to be one of the applications in the current technological development and can be further developed so that it can work better in the future.

We would like to thank the members of group A-6 PSD 01 who helped in preparing this report so that the report can be completed well. Thanks also to Miranty Anjani Putri as the mentor of group A-6 PSD 01 who has supported us so that we can complete this report on time.

We are aware that this project still has shortcomings. Therefore, suggestions and criticisms are very much needed to make us better in the future. Hopefully this report can be useful in the field of education, especially in the field of technology.

Depok, December 09, 2022

Group A-6

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

1.1 BACKGROUND

Currently, many urban areas face challenges with parking due to a growing population and an increasing number of vehicles. Traditional surface-level parking lots are often unable to keep up with the demand for parking, leading to traffic congestion, long search times for parking spots, and frustration for drivers. That's why we create multi-level parking systems.

Multi-level parking systems offer a solution to these challenges by providing additional parking capacity in a smaller footprint. These systems typically consist of several levels of parking stacked on top of each other, with ramps or elevators to access the different levels. This allows for more cars to be parked in a given area compared to a traditional surface-level parking lot.

In addition to increased parking capacity, multi-level parking systems can also provide additional controlled access. They can also make it easier for drivers to find a parking spot, thanks to features like directional signs and automatic parking guidance systems. This can help to reduce traffic congestion and improve the overall parking experience.

Furthermore, a multi-level parking system can help to reduce the amount of land and resources required for parking, which can have a positive impact on the environment. This can be especially important in urban areas where space is limited and the demand for parking is high.

1.2 PROJECT DESCRIPTION

1.2.1 FUNCTIONS

This project aims to develop a VHDL-based simulation of a multi-level parking system. The system will be able to simulate the behavior of a parking lot with multiple floors and different capacities for each floor. The main function of the multi-level parking system is to provide a convenient and organized way to park cars in a large parking lot with multiple floors.

1.2.2 FEATURE

A. Encrypt Password

Encrypting a password with a 32-bit key means that the password is converted into a scrambled, unreadable form using a mathematical algorithm and a key that is 32 bits long. This makes it difficult for anyone who does not have the key to decipher the encrypted password and access the underlying information. To encrypt a password with a 32-bit key, the encryption algorithm takes the password and the key as input, and applies a series of mathematical operations to the password to generate an encrypted version of the password. The key is used to control the encryption process and determine the specific details of how the password is scrambled.

B. Overload Parking

Overloads in a multi-level parking lot refer to situations where the number of cars trying to park in the lot exceeds the available parking spots. This can happen when the parking lot is full, or when the demand for parking is higher than the capacity of the lot. Overloads in a parking system can be indicated by the use of lights. For example, the lights on the entrance to the parking system may turn red, indicating that the system is full and no more cars can be accommodated.

C. Automatic Parking Assignment

Automatic parking assignment refers to the use of technology to automatically assign parking spots to cars in a parking system. Can use algorithms or rules to determine which parking spots are available and assign them to incoming cars. The main advantage of automatic parking assignment is that it can help to optimize the use of the parking system and prevent congestion. By automatically assigning parking spots to cars, it can ensure that the available parking spots are used efficiently and that cars are parked in the most convenient

and efficient locations. This can reduce the time and effort required to find a parking spot, and can help to prevent traffic jams and delays.

D. Calculate The Parking Fee According To The Duration

Calculating the fee for parking refers to the process of determining the amount of money that a driver should pay to park their car in a parking system. The fee can be based on various factors, such as the duration of the parking. To calculate the fee for parking, the parking system may use inputs, such as the time that the car entered the parking system and the time that the car left the parking system. Once the inputs have been collected and processed, the system can use the rules or algorithms to calculate the fee for parking. The result of the calculation can be displayed to the driver and it can be automatically applied to the payment process.

1.2.3 USER FLOW

A. Car-In Flow

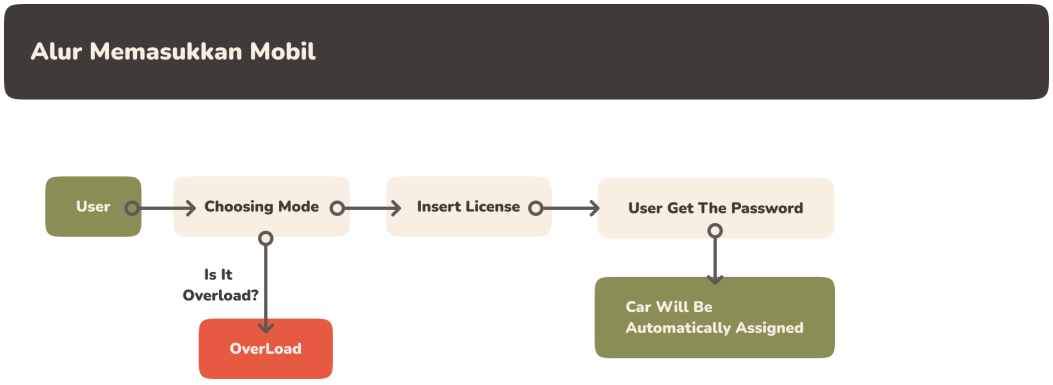


Fig 1. Car-In Flow

If a user wanted to park in this multi-level parking users have to choose input mode and insert their license plate, after inserting their license plate they will get their password for taking the car and their car will be automatically assigned to the empty parking lot.

B. Car-Out Flow

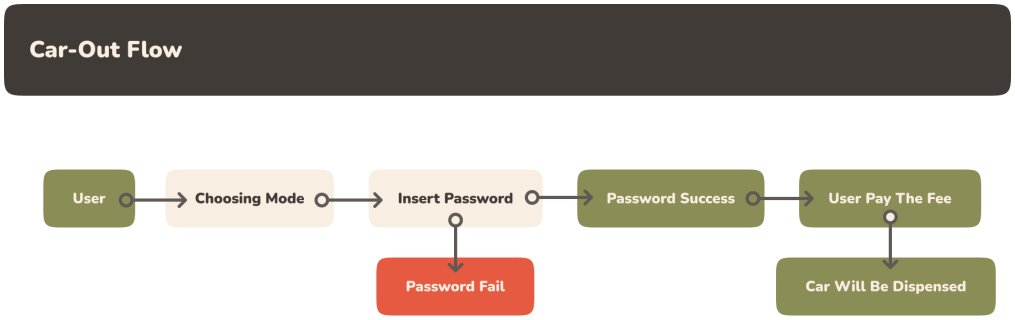


Fig 2. Car-Out Flow

if a user wanted to take their car out they have to select the out mode and insert their previously provided password before getting their car out, if the password is right and the user already paid the price then the car will automatically be assigned to the base floor for the user to pick up. If the password is wrong then the circuit will reset.

1.3 OBJECTIVES

The objectives of this project are as follows:

1. To develop a VHDL model of a multi-level parking system and to evaluate its performance.
2. To integrate the VHDL model of a multi-level parking system with other VHDL models of parking-related systems, such as password and automatic parking guidance systems, and to evaluate the performance of the integrated system.

3. To investigate the potential applications of the VHDL model of a multi-level parking system in real-world settings and to identify areas for future research and development.

1.4 ROLES AND RESPONSIBILITIES

The roles and responsibilities assigned to the group members are as follows:

Roles	Responsibilities	Person
Idea Creation	Generate and develop ideas and requirements for circuit designs, components, and the parking systems	Abdul Fikih Kurnia, Aqib Rahman, Bernanda Nautval R.I.W, and Rafie Amandio Fauzan
Circuit Design	Design and develop circuits, components, and systems to meet specified requirements.	Abdul Fikih Kurnia, Bernanda Nautval R.I.W, and Rafie Amandio Fauzan
VHDL Programmer	Create code for the main component, and other components. (Everyone has different tasks and you can see it in our GitHub branch https://github.com/Abdfikih/Multi-Level-Parking-System)	Abdul Fikih Kurnia, Bernanda Nautval R.I.W, and Rafie Amandio Fauzan

Report Creation	Write, edit, and proofread the report to ensure it is clear, concise, and accurate	Abdul Fikih Kurnia, Bernanda Nautval R.I.W, and Rafie Amandio Fauzan

Table 1. Roles and Responsibilities

CHAPTER 2

IMPLEMENTATION

2.1 EQUIPMENT

The tools that are going to be used in this project are as follows:

- Replit as a collaborative coding platform https://replit.com
- Git as version control
- Visual Studio Code as a local storage place for coding on each member's laptop / computer
- ModelSim as a VHDL circuit simulator
- Draw.io and Figma as block diagram illustrators

2.2 IMPLEMENTATION

The overall top level circuit is like this

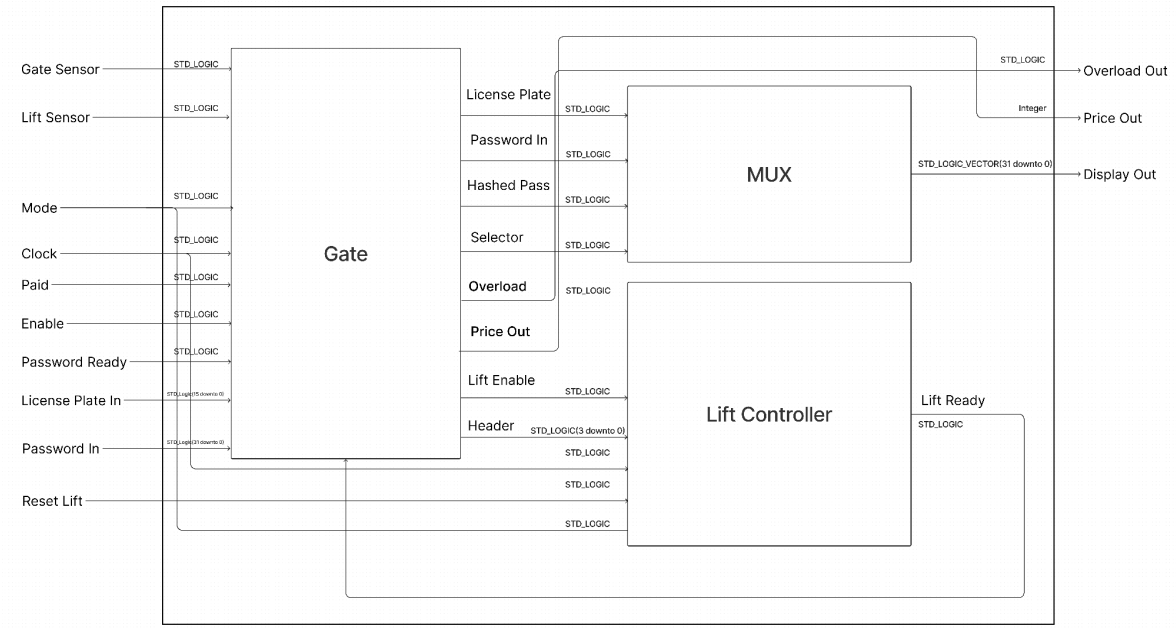


Fig 3. Schematic

There is 3 main component in this circuit which is Gate, Mux, and Lift Controller. Gate is the main component that processes the password, memory, and license plate. When the gate receive an input the gate will process it and when the process come to IN SUCCESS or PASS SUCCESS then the Gate will send a lift enable signal and header for the lift controller that indicate that the car is inside the lift and wanted to send the lift to specific position, the position will be sent in the format of a header like this one below

Fig 4. Header Format

Header will be sent from gate to the lift controller in the format of 4 bit, the first two bit will be read as floor number and the last two as room number. This Circuit technically have 2 Finite State Machine, The Gate and The Lift Controller.

We store the password and plate inside a record and we store the record inside a 4x4 record array named parking_lot.

```
    TYPE room IS RECORD
            room_status : STD_LOGIC;
            fee : INTEGER;
            timer : TIME;
            password : STD_LOGIC_VECTOR(31 DOWNTO 0);
            plate : STD_LOGIC_VECTOR(15 DOWNTO 0);
        END RECORD;


        TYPE parking_lot IS ARRAY (3 DOWNTO 0, 3 DOWNTO 0) OF
room;
```

Room status is used to mark if the parking space is occupied or not, fee is used to store the parking fee, and timer is used to mark the arrival time of the car this field will later be subtracted from the present time to get the price. A password is used to receive the hashed password and store it. A plate is used to store the license plate of the car inside the parking space.

This multi-level parking System has 10 input in the top component which is

No	Input	Function
1.	Gate Sensor	This input is used to change the state from Idle to Select mode, in the implementation this input should be connected to a gate sensor.
2.	Lift Sensor	Lift Sensor should be connected to a sensor near the lift to ensure that the car is already inside the lift so that the lift can now move.
3.	Reset Lift	Input To reset the lift
4.	Mode	is used to decide to pick a car or to park a car, In this circuit Mode is a STD_Logic. 1 in mode represents Car Input and 0 represents Car Output
5.	Clock	Clock input for this circuit
6.	Paid	Is the status of whether the user that successfully inputs the password has paid the bills or not.
7.	Enable	Used to decide if the user already choose the mode.

8.	Password Ready	To know if the user already input the password.
9.	License Plate In	Path to insert the license plate from user
10.	Password In	Path to insert the password from user.

And The top level output as such

No	Input	Function
1.	Overload Out	This output should ideally be connected to some sort of LED or warning to indicate if the parking lot is already full and cannot accept any more cars.
2.	Price	This will provide the calculated price for the buyer
3.	Display Out	this 32 bit output should be connected to 8 seven-segment display, each 4 digit will be connected to seven-segment encoder that receive 4 bit and output to hexadecimal. This is connected to mux which choose what value to display

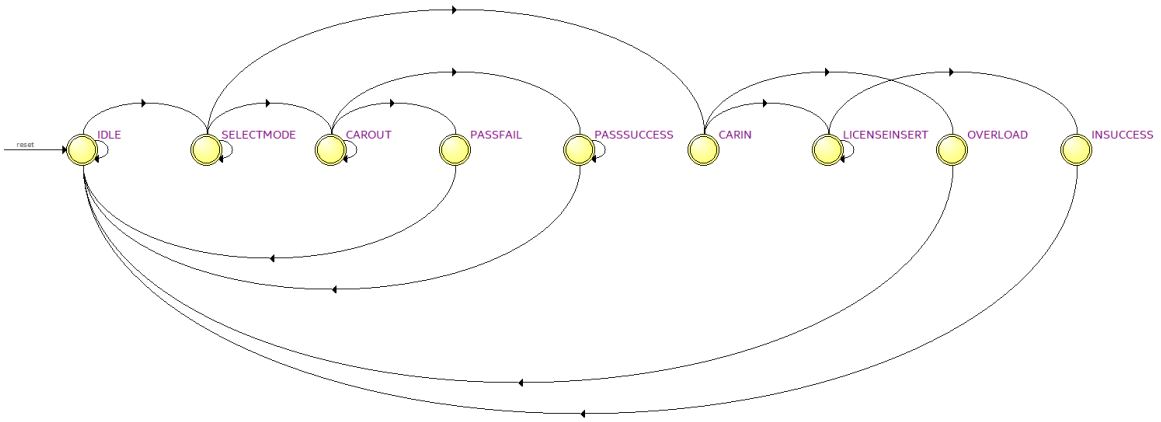


Fig 5. Synthesized State

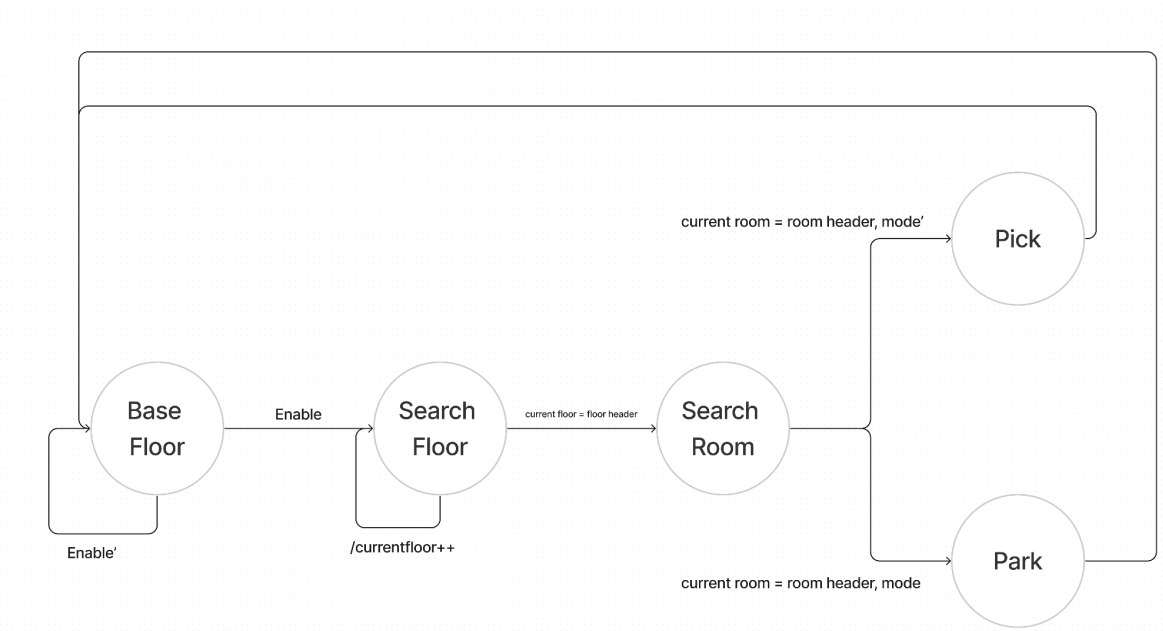


Fig 6. Lift State Diagram

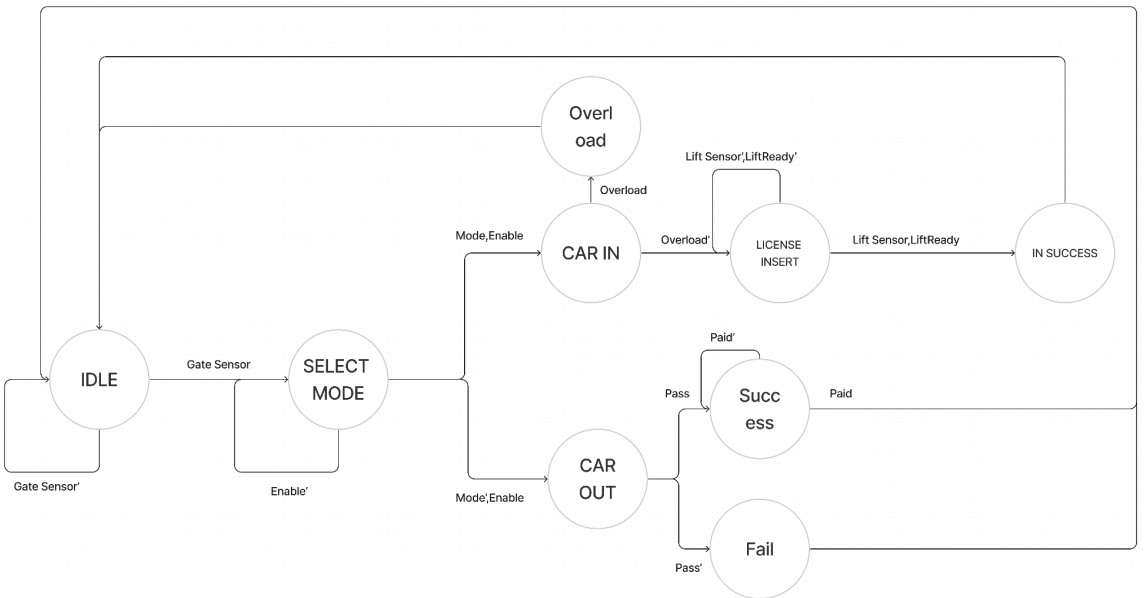


Fig 7. State Diagram

When people enter the parking lot, the gate sensor triggers the state to change to Select mode, and the state will remain there until the enable has the value of 1, indicating that the user has already decided which mode they want and the circuit will

proceed to the selected mode. In the CAR IN state, the program will determine whether or not the parking lot is full; if so, it will return to the IDLE state; if not, the user should enter their license, which will be saved in the memory. After success then the circuit will send a lift enable to the lift and the lift controller will active and run this state to sent the lift to the correct position and back to the base floor, after reaching base floor the lift will send lift ready to the gate so the gate can continue their state. then the state will back to IDLE so the next user can use the elevator. The user will be asked to enter the password in the CAR OUT state, and if they fail, the state will return to IDLE; if they succeed, the state will proceed to the PASS SUCCES state, where they will wait for the user to pay the fee, and once the user has paid the fee, the lift will do its job, and the state will return to IDLE.

CHAPTER 3

TESTING AND ANALYSIS

3.1 TESTING

After merging the program with each component to the main program, which is the top level, the program will be tested to see if the output matches the given input and is in accordance with the algorithm in the program. Testing is performed on the main program using the Finite State Machine that has been combined with the components that have been created.

To test the program, we decided to make 5 possible cases :

a) Car in success
b) Car in but the room overloaded
c) Car out but wrong password
d) Car out but not yet paid
e) Car out success

3.2 RESULT

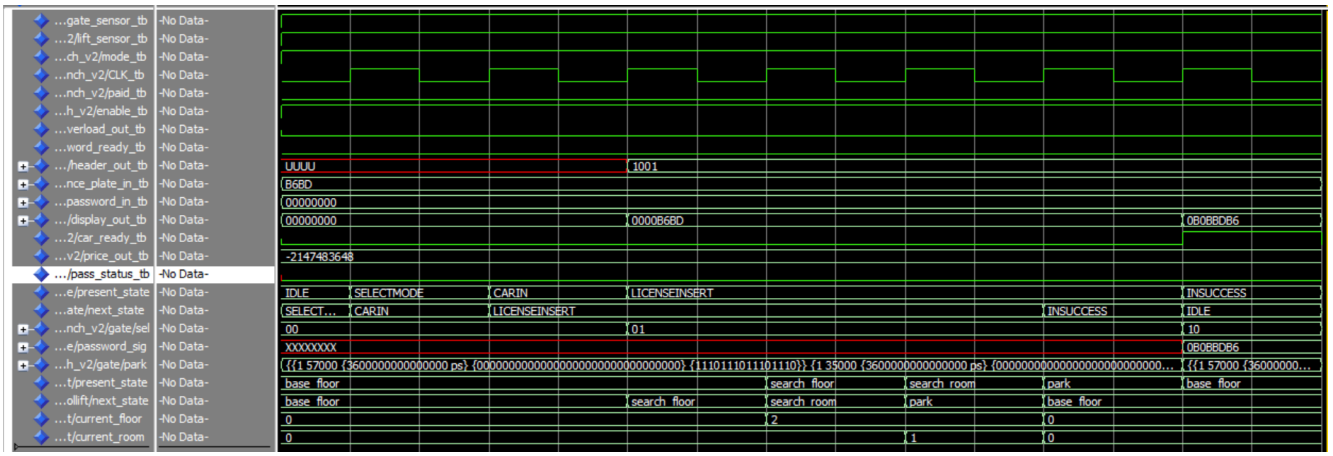


Fig 6. Testing Result for Case “Car in Success”

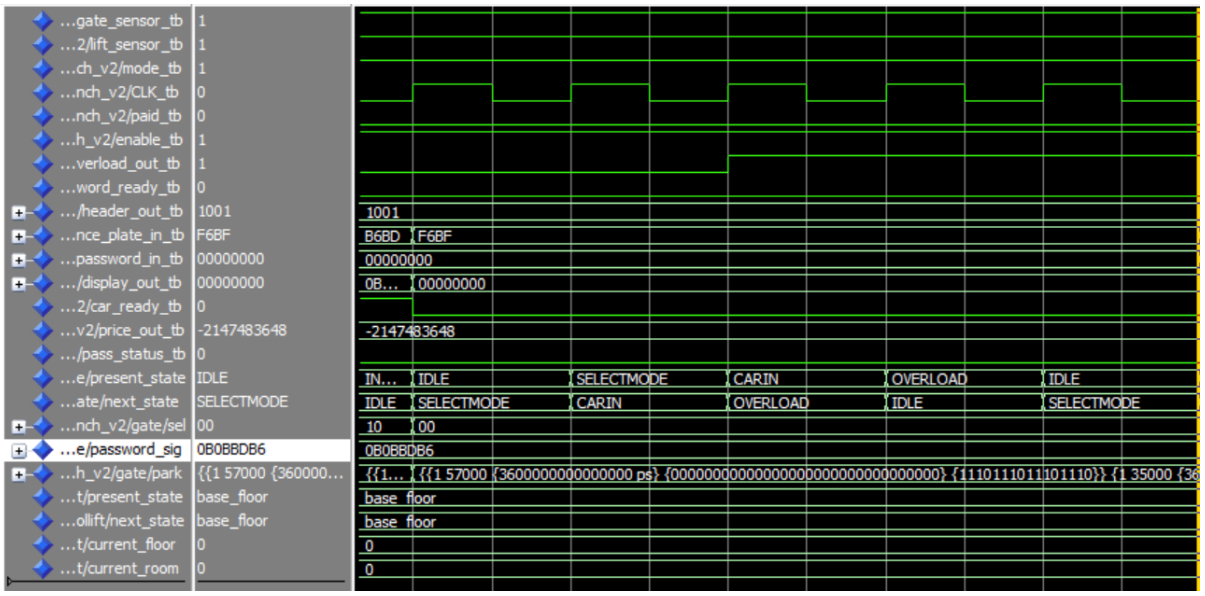


Fig 7. Testing Result for Case “Car in but Overloaded”

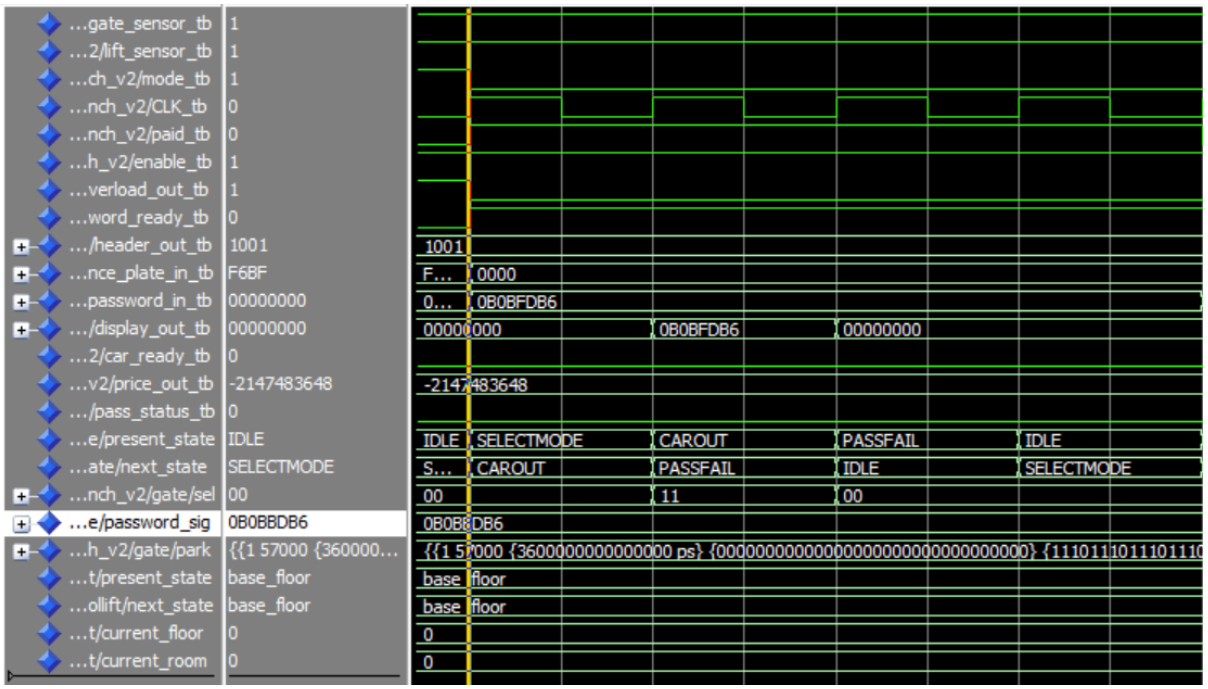


Fig 8. Testing Result for Case “Car out but wrong password”

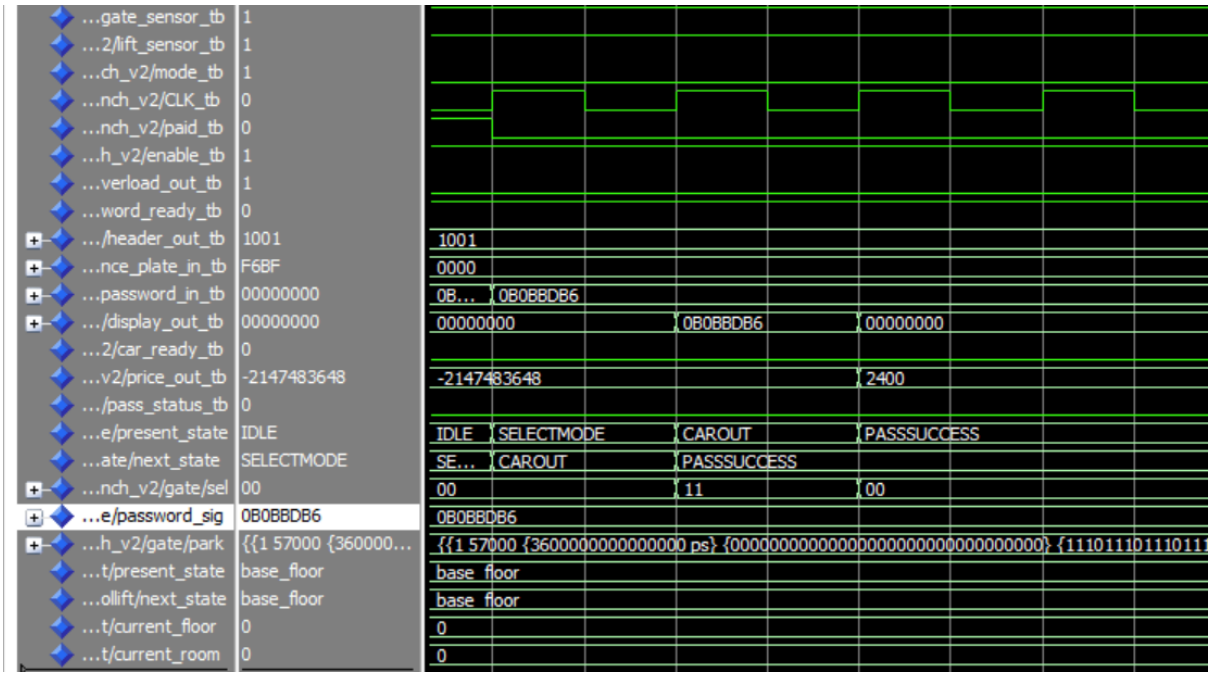


Fig 9. Testing Result for Case “Car out but not yet paid”

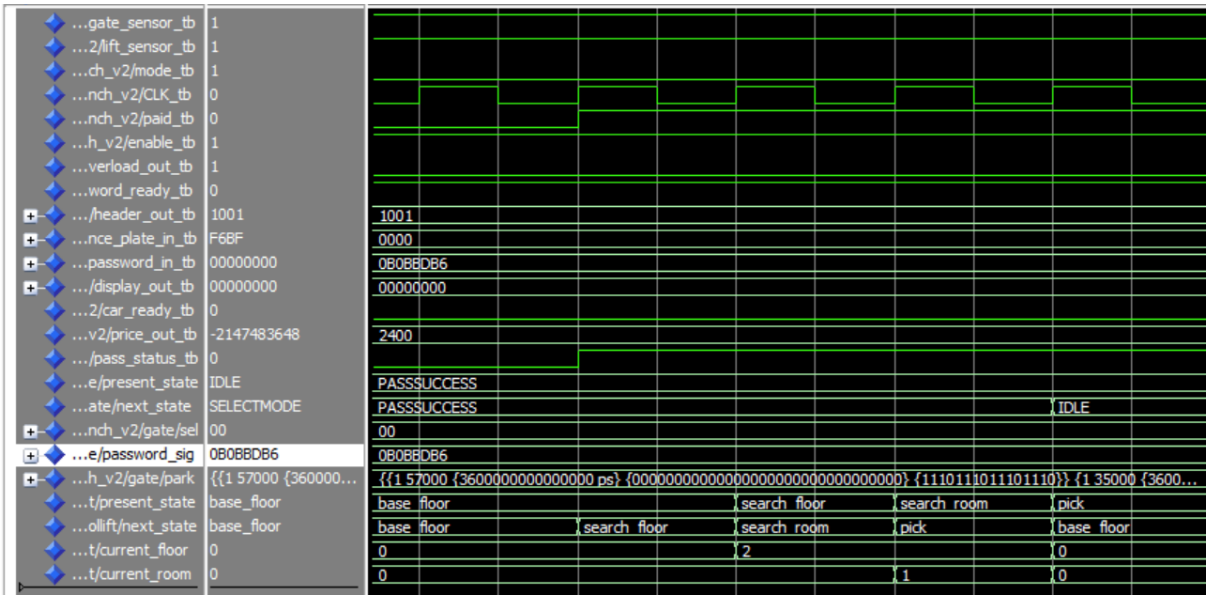


Fig 10. Testing Result for Case “Car out Success”

```
# ** Note: [PARKING ROOM SLOT 15/16]

# ** Note: The car has been successfully parked in the room 2, 1
#    Time: 1499 ps  Iteration: 0  Instance: /testbench_v2

# ** Note: The car cannot be parked because the room is overloaded
#    Time: 2500 ps  Iteration: 0  Instance: /testbench_v2

# ** Note: The car cannot be picked because the password doesn't match
#    Time: 3300 ps  Iteration: 0  Instance: /testbench_v2

# ** Note: Price: 2400

# ** Note: The car cannot be picked because the parking fee has not been paid

# ** Note: The car has been picked successfully from the room
```

Fig 10. Testing Result in Transcript Page

Based on the simulation results as shown in the above figure, it can be said that the program can run as it should. The input will be the right reference for the desired output results so it can be declared that this program is successful in carrying out its performance.

3.3 ANALYSIS

First, we set scenario that the 15/16 are already occupied. Then we run signal simulation that cover all the 5 different case

a. Car In Success

   In this test the plate and password is safely stored in the memory and in the LICENSE INSERT the lift state is working properly also the header is working fine as it should.

b. Car in but the room overloaded

   In this test, the parking space is already full and the next car input should not be assigned to any parking spot. The result is as expected: the car cannot go in and goes to the "overload" state, and the overload out is valued at 1.

c. Car out but wrong password

   In this test case the program work successfully where the state after user inputting false password goes to PASS FAIL and then back to IDLE

d. Car out but not yet paid

   When the password input matches the password stored in the memory, the car will be in a waiting state until the fee is paid by the user, and then it will wait for the lift to be ready before going to the next state.

e. Car out success

   This case is when the password matched and the parking fee already paid.

CHAPTER 4

CONCLUSION

This project aims to develop a VHDL-based simulation of a multi-level parking system. The system will be able to simulate the behavior of a parking lot with multiple floors and different capacities for each floor. The main function of the multi-level parking system is to provide a convenient and organized way to park cars in a large parking lot with multiple floors.

The system has a feature where when a car enters the gate, a check will be made on the availability of parking spaces. If the parking space is full, the lights will turn on and if it is available, the lights will turn off. When the parking space is available, the input of the license plate number will be made. When the license plate number is entered, the license plate number will be converted into a 32-bit password where the encryption algorithm has been determined in the code. Both will be displayed on the seven-segment alternately, which is regulated by the multiplexer and entered into the record so that it will be recorded and stored. After the car has been successfully placed in the available parking space, the time of entry will be stored and the time will continue to run until the car is ready to be taken. When the car is ready to be taken, the user will be asked to enter the license plate number and password of the car. Then, the duration of the parking time will stop and be multiplied by the unit parking fee. After that, the parking fee to be paid will be displayed, then payment is made automatically and the car successfully exits. The memory of the existing car will be erased and ready to be filled with cars that want to enter.

REFERENCES

[1] I. Jahan, P. Das, and M. Ahsan, “CAR PARKING SYSTEM DESIGN IN VHDL,” 2019. Accessed: Dec. 2, 2022. [Online]. Available: http://dspace.daffodilvarsity.edu.bd:8080/bitstream/handle/123456789/318/P12854%20%2837%25%29.pdf?sequence=1&isAllowed=y

[2] Anon, “Automated Parking System - Final Report,” Accessed: Dec. 3, 2022. [Online]. Available:http://www.ece.ualberta.ca/~elliott/ee552/projects/1998_w/Automated_Parking/final.html

[3] S. Sharmila. Devi, B. A. J.J.R, D. M, and K. A.I, “Car Parking System Using FPGA,”. Accessed: Dec. 5, 2022. [Online]. International Research Journal on Advanced Science Hub, vol. 2, no. 8, pp. 88–93, Sep. 2020, doi: 10.47392/irjash.2020.99. Available: https://rspsciencehub.com/pdf_99_4a8d46a09be536c53c1ded0b2b440182.html

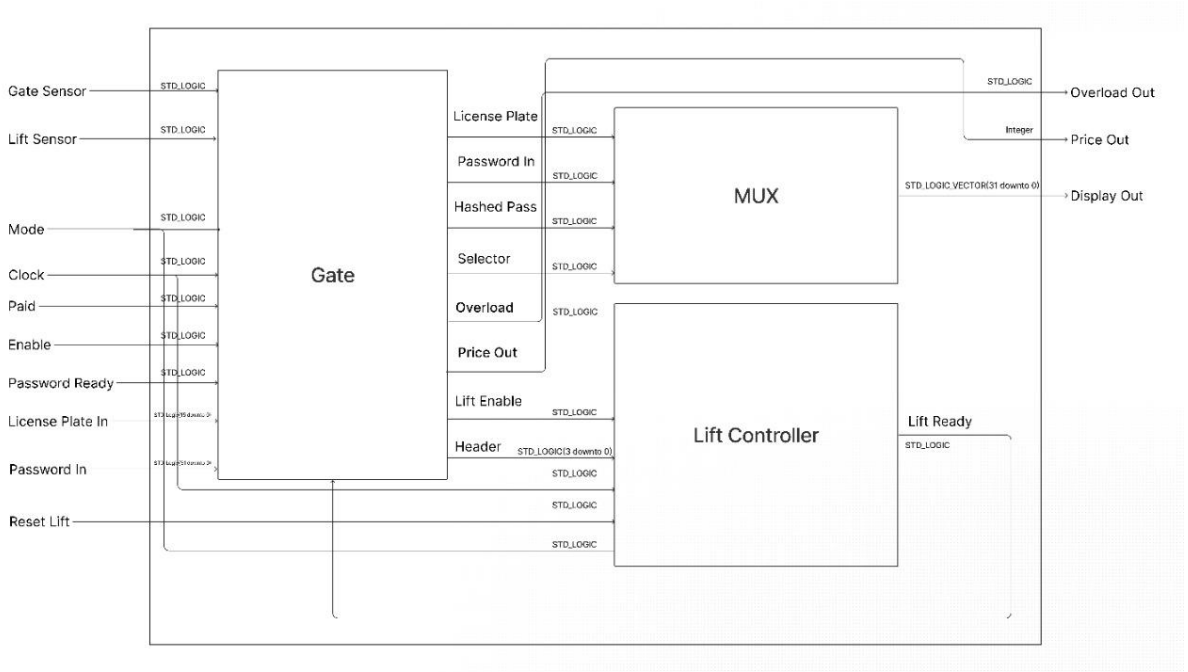
[4] G. Ggyu, A. Gunasekara, and R. Kathriarachchi, “A Smart Vehicle Parking Management Solution,” 2015. Accessed: Dec. 5, 2022. [Online]. Available: http://ir.kdu.ac.lk/bitstream/handle/345/1048/com-054.pdf?sequence=1&isAllowed=y

[5] P. Mirunalini, B. Bharathi, N. Ananthamurugan, S. Suresh, and S. Gopal, “Multi-Level Smart Parking System,” IEEE Xplore, Feb. 01, 2018. Accessed: Dec. 5, 2022. [Online]. Available : https://ieeexplore.ieee.org/document/8452835

[6] Digital Laboratory DTE UI. 2022.. “Modul 2-9 Digital System Design”. Accessed: Dec. 5, 2022. [Online]. Available : https://emas.ui.ac.id

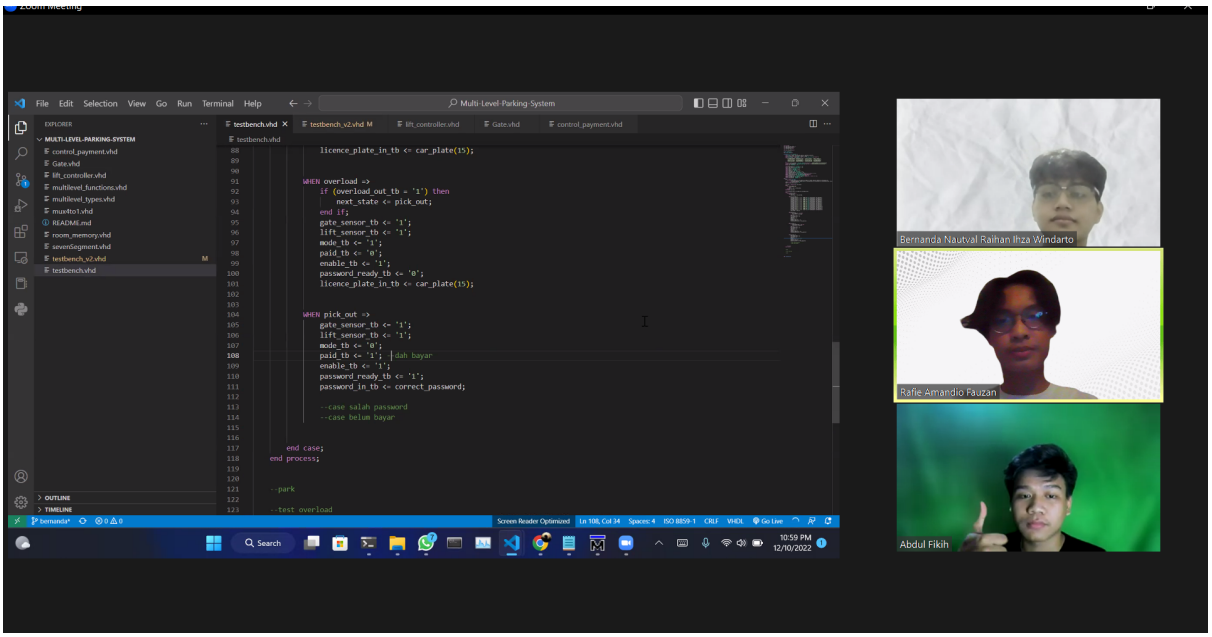
APPENDICES

Appendix A: Project Schematic

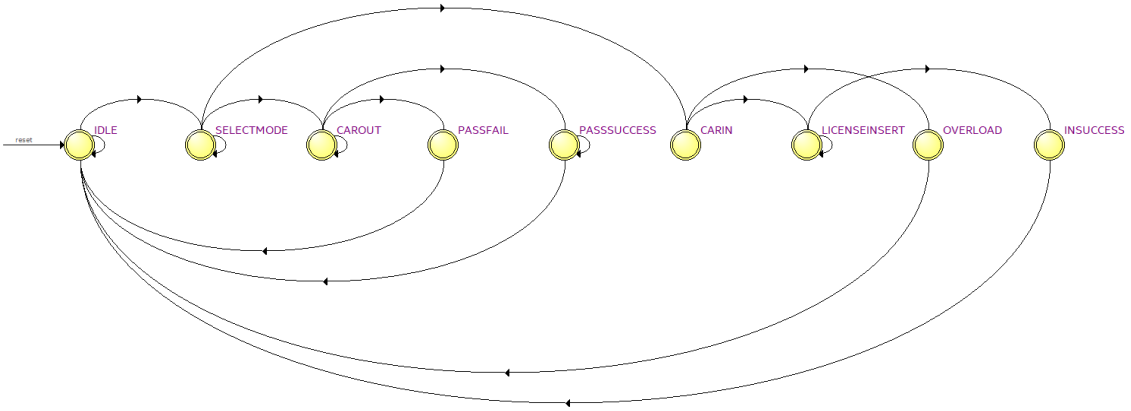


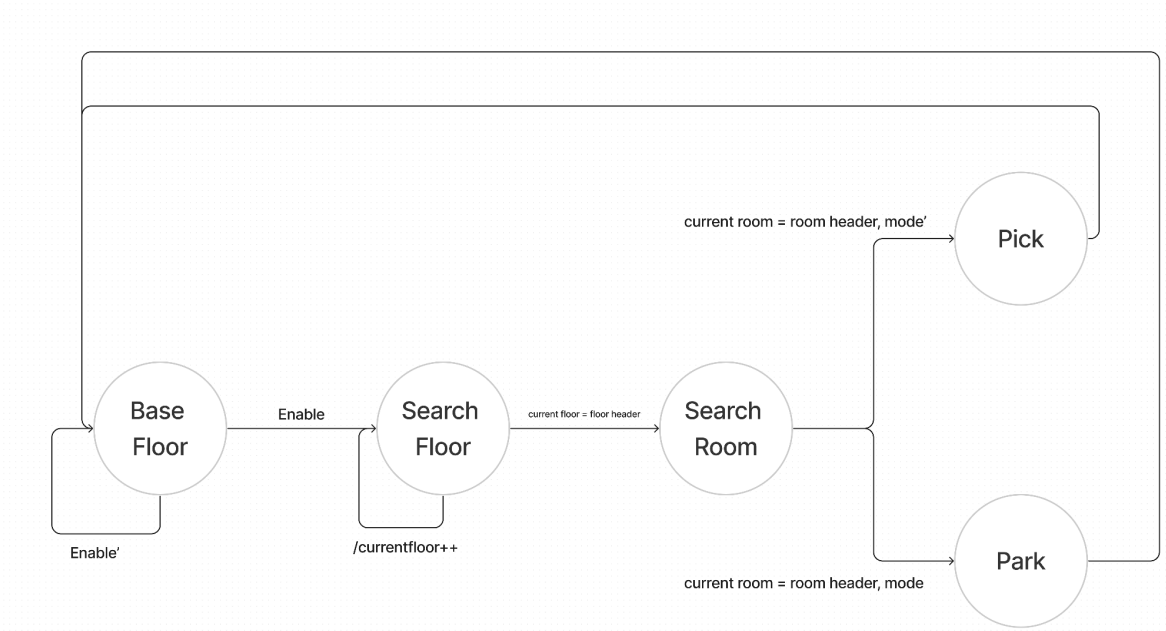
Appendix B: Documentation

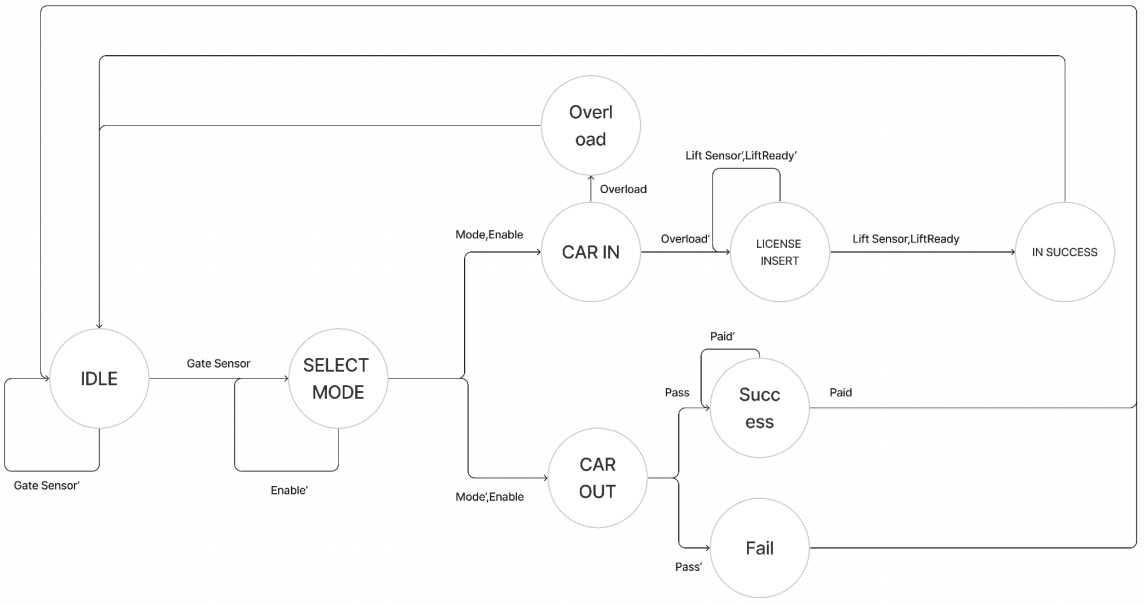
Put the documentation (photos) during the making of the project



Appendix C : State







Appendix D : Main Source Code

```
LIBRARY ieee;

USE ieee.std_logic_1164.ALL;

USE ieee.numeric_std.ALL;

USE ieee.math_real.ALL;
```

```
USE work.multilevel_types.ALL;
USE work.multilevel_functions.ALL;
USE work.room_memory.ALL;

ENTITY Gate IS
    PORT (
        gate_sensor : IN STD_LOGIC;
        lift_sensor : IN STD_LOGIC;
        mode : IN STD_LOGIC;
        clk : IN STD_LOGIC;
        paid : IN STD_LOGIC;
        enable : IN STD_LOGIC;
        overload_out : OUT STD_LOGIC;
        password_ready : IN STD_LOGIC;
        header_out : OUT STD_LOGIC_VECTOR(3 DOWNTO 0);
        licence_plate_in : IN STD_LOGIC_VECTOR(15 DOWNTO 0);
        password_in : IN STD_LOGIC_VECTOR(31 DOWNTO 0);
        display_out : OUT STD_LOGIC_VECTOR(31 DOWNTO 0);
        car_ready : OUT STD_LOGIC;
        price_out : OUT INTEGER;
        pass_status : OUT STD_LOGIC := '0'
        -- time_stamp : out integer
    );
END ENTITY;

ARCHITECTURE rtl OF Gate IS
    COMPONENT mux_4to1 IS
        PORT (
```

```
            PlatIn, PlatOut : IN STD_LOGIC_VECTOR (15 DOWNTO 0);
            PassIn, PassOut : IN STD_LOGIC_VECTOR (31 DOWNTO 0);
            Sel : IN STD_LOGIC_VECTOR (1 DOWNTO 0);
            Zout : OUT STD_LOGIC_VECTOR (31 DOWNTO 0)
        );
    END COMPONENT;

    COMPONENT sevenSegment IS
        PORT (
            Bin : IN STD_LOGIC_VECTOR(3 DOWNTO 0);
            Outer : OUT STD_LOGIC_VECTOR(6 DOWNTO 0)
        );
    END COMPONENT;

    COMPONENT lift_controller IS
        PORT (
            clk : IN STD_LOGIC;
            position_header : IN STD_LOGIC_VECTOR(3 DOWNTO 0) :=
(OTHERS => '0');
            enable : IN STD_LOGIC := '0';
            mode : IN STD_LOGIC := '0';
            ready : INOUT STD_LOGIC := '0'

        );
    END COMPONENT lift_controller;

    --State machine
    CONSTANT offscreen : STD_LOGIC_VECTOR(15 DOWNTO 0) := (OTHERS =>
'0');
```

```
    SIGNAL present_state, next_state : state_type;
    SIGNAL position_header : STD_LOGIC_VECTOR(3 DOWNTO 0);
    SIGNAL enablelift : STD_LOGIC := '0';
    SIGNAL sel : STD_LOGIC_VECTOR(1 DOWNTO 0) := "00";
    SIGNAL lifting_state : STD_LOGIC_VECTOR(1 DOWNTO 0) := (OTHERS
=> '0');
    SIGNAL liftready : STD_LOGIC;
    SIGNAL password_sig : STD_LOGIC_VECTOR(31 DOWNTO 0);
    SIGNAL park : parking_lot := parking_array;
    SIGNAL header : STD_LOGIC_VECTOR(3 DOWNTO 0);
    SIGNAL priceout : INTEGER;


BEGIN
    mux : mux_4to1 PORT MAP(
        offscreen, licence_plate_in, password_sig, password_in, sel,
display_out
    );


    controllift : lift_controller PORT MAP(clk, header, enablelift,
mode, liftready);


    --synchronize process
    sync_proc : PROCESS (clk, next_state)
    BEGIN
        IF rising_edge(clk) THEN
            present_state <= next_state;
        END IF;
    END PROCESS;


    --comb process
```

```
    comb_proc : PROCESS (present_state, gate_sensor, lift_sensor,
mode, enable, password_ready, licence_plate_in, password_in, paid,
liftready)
        VARIABLE price, timeelapsed : INTEGER := 0;
        VARIABLE overload_var : STD_LOGIC;
        VARIABLE pass_stat_var : STD_LOGIC;
    BEGIN
        sel <= "00";
        park <= parking_array;
        header_out <= header;
        overload_out <= overload_var;
        pass_status <= pass_stat_var;
        CASE present_state IS
            WHEN IDLE =>
                car_ready <= '0';
                overload_var := '0';
                pass_stat_var := '0';
                IF gate_sensor = '1' THEN
                    next_state <= SELECTMODE;
                ELSE
                    next_state <= IDLE;
                END IF;
            WHEN SELECTMODE =>
                IF enable = '1' AND mode = '1' THEN
                    next_state <= CARIN;

                ELSIF enable = '1' AND mode = '0' THEN
                    next_state <= CAROUT;
                ELSE
                    next_state <= SELECTMODE;
```

```
                END IF;


            WHEN CAROUT =>
                IF password_ready = '1' THEN
                    sel <= "11";
                    next_state <= PASSSUCCESS;
                    IF (parking_array(0, 0).password = password_in)
THEN
                        header_out <= "0000";
                    ELSIF (parking_array(0, 1).password =
password_in) THEN
                        header <= "0001";
                    ELSIF (parking_array(0, 2).password =
password_in) THEN
                        header <= "0010";
                    ELSIF (parking_array(0, 3).password =
password_in) THEN
                        header <= "0011";
                    ELSIF (parking_array(1, 0).password =
password_in) THEN
                        header <= "0100";
                    ELSIF (parking_array(1, 1).password =
password_in) THEN
                        header <= "0101";
                    ELSIF (parking_array(1, 2).password =
password_in) THEN
                        header <= "0110";
                    ELSIF (parking_array(1, 3).password =
password_in) THEN
                        header <= "0111";
                    ELSIF (parking_array(2, 0).password =
password_in) THEN
```

```
                    header <= "1000";
                ELSIF (parking_array(2, 1).password =
password_in) THEN
                    header <= "1001";
                ELSIF (parking_array(2, 2).password =
password_in) THEN
                    header <= "1010";
                ELSIF (parking_array(2, 3).password =
password_in) THEN
                    header <= "1011";
                ELSIF (parking_array(3, 0).password =
password_in) THEN
                    header <= "1100";
                ELSIF (parking_array(3, 1).password =
password_in) THEN
                    header <= "1101";
                ELSIF (parking_array(3, 2).password =
password_in) THEN
                    header <= "1110";
                ELSIF (parking_array(3, 3).password =
password_in) THEN
                    header <= "1111";
                ELSE
                    next_state <= PASSFAIL;
                END IF;

            ELSE
                next_state <= CAROUT;
            END IF;

        WHEN CARIN =>
```

```
                IF (parking_array(0, 0).room_status = '0') THEN
                    header <= "0000";
                ELSIF (parking_array(0, 1).room_status = '0') THEN
                    header <= "0001";
                ELSIF (parking_array(0, 2).room_status = '0') THEN
                    header <= "0010";
                ELSIF (parking_array(0, 3).room_status = '0') THEN
                    header <= "0011";
                ELSIF (parking_array(1, 0).room_status = '0') THEN
                    header <= "0100";
                ELSIF (parking_array(1, 1).room_status = '0') THEN
                    header <= "0101";
                ELSIF (parking_array(1, 2).room_status = '0') THEN
                    header <= "0110";
                ELSIF (parking_array(1, 3).room_status = '0') THEN
                    header <= "0111";
                ELSIF (parking_array(2, 0).room_status = '0') THEN
                    header <= "1000";
                ELSIF (parking_array(2, 1).room_status = '0') THEN
                    header <= "1001";
                ELSIF (parking_array(2, 2).room_status = '0') THEN
                    header <= "1010";
                ELSIF (parking_array(2, 3).room_status = '0') THEN
                    header <= "1011";
                ELSIF (parking_array(3, 0).room_status = '0') THEN
                    header <= "1100";
                ELSIF (parking_array(3, 1).room_status = '0') THEN
                    header <= "1101";
                ELSIF (parking_array(3, 2).room_status = '0') THEN
```

```
                    header <= "1110";
                ELSIF (parking_array(3, 3).room_status = '0') THEN
                    header <= "1111";
                ELSE
                    overload_var := '1';
                    overload_out <= overload_var;
                END IF;

                IF overload_var = '1' THEN
                    next_state <= OVERLOAD;
                ELSE
                    next_state <= LICENSEINSERT;
                END IF;

            WHEN LICENSEINSERT =>
                IF lift_sensor = '1' AND NOT (licence_plate_in =
"0000000000000000") THEN
                    sel <= "01";
                    enablelift <= '1';
                    IF liftready = '1' THEN
                        next_state <= INSUCCESS;
                    ELSE
                        next_state <= LICENSEINSERT;
                    END IF;
                ELSE
                    next_state <= LICENSEINSERT;
                END IF;

            WHEN INSUCCESS =>
```

```
                car_ready <= '1';
                enablelift <= '0';
                sel <= "10";
                parking_array(to_integer(unsigned(header(3 DOWNTO
2))), to_integer(unsigned(header(1 DOWNTO 0)))).room_status := '1';
                parking_array(to_integer(unsigned(header(3 DOWNTO
2))), to_integer(unsigned(header(1 DOWNTO 0)))).plate :=
licence_plate_in;
                parking_array(to_integer(unsigned(header(3 DOWNTO
2))), to_integer(unsigned(header(1 DOWNTO 0)))).timer := now;
                password_sig <= hash(licence_plate_in);
                parking_array(to_integer(unsigned(header(3 DOWNTO
2))), to_integer(unsigned(header(1 DOWNTO 0)))).password :=
password_sig;

                next_state <= IDLE;

            WHEN PASSFAIL =>
                pass_stat_var := '0';
                next_state <= IDLE;

            WHEN PASSSUCCESS =>
                IF (pass_stat_var = '0') THEN
                    price := (now -
parking_array(to_integer(unsigned(header(3 DOWNTO 2))),
to_integer(unsigned(header(1 DOWNTO 0)))).timer) / 1 ps;
                    REPORT "Price: " & INTEGER'image(price);
                    price_out <= price;
                END IF;
                pass_stat_var := '1';
                IF paid = '1' THEN
                    enablelift <= '1';
```

```
                    IF liftready = '1' THEN

                        parking_array(to_integer(unsigned(header(3
DOWNTO 2))), to_integer(unsigned(header(1 DOWNTO 0)))).room_status
:= '0';

                        parking_array(to_integer(unsigned(header(3
DOWNTO 2))), to_integer(unsigned(header(1 DOWNTO 0)))).plate :=
(OTHERS => '0');

                        parking_array(to_integer(unsigned(header(3
DOWNTO 2))), to_integer(unsigned(header(1 DOWNTO 0)))).timer := 0
ns;

                        parking_array(to_integer(unsigned(header(3
DOWNTO 2))), to_integer(unsigned(header(1 DOWNTO 0)))).password :=
(OTHERS => '0');

                        next_state <= IDLE;

                    ELSE

                        next_state <= PASSSUCCESS;

                    END IF;

                ELSE

                    next_state <= PASSSUCCESS;

                END IF;


            WHEN OVERLOAD =>

                next_state <= IDLE;


        END CASE;

    END PROCESS;


END ARCHITECTURE;
```