Karunya INSTITUTE OF TECHNOLOGY AND SCIENCES
(Declared as Deemed to be University under Sec.3 of the UGC Act, 1956)
MoE, UGC & AICTE Approved
NAAC A++ Accredited

Division of Electronics and Communication Engineering
2024-2025 (ODD SEM)

REPORT

for

FPGA BASED SYSTEM DESIGN-PROJECT BASED COURSE

Title of the Report: Car Parking System Using Verilog On FPGA

A report submitted by

Name of the Student	*Kerlin E, Annie Reachel A, S Blessy Bebina*
Register Number	**URK22EC1071, URK22EC1051, URK22EC1046**
Subject Name	*FPGA Based System Design*
Subject Code	*22EC2020*
Date of Report submission	*30/09/2024*

Total marks: /10 Marks

Signature of Faculty with date:

Karunya INSTITUTE OF TECHNOLOGY AND SCIENCES
(Declared as Deemed to be University under Sec.3 of the UGC Act, 1956)
MoE, UGC & AICTE Approved
NAAC A++ Accredited

TABLE OF CONTENTS

CHAPTER	TITLE	PAGE NO.
1.	INTRODUCTION	3
2.	IMPLEMENTATION	4
3.	TESTING THE OUTPUT	8
4.	CONCLUSION	11

CHAPTER 1

INTRODUCTION

A **car parking system using Verilog on FPGA**, utilizing a single switch and a seven-segment display, is designed to manage and display the number of available parking spots in a parking lot.

In this system:

1. **Switch:** A single switch represents both car entry and exit. When the switch is turned on, it indicates that a car has entered the parking lot and is parked. When the switch is turned off, it represents that the car has left the parking lot.

2. **Counter System**: The system uses a counter that decreases when the switch is turned on (indicating a car has occupied a space) and increases when the switch is turned off (indicating a car has left, freeing up a space). The counter ensures the real-time update of available parking spots.

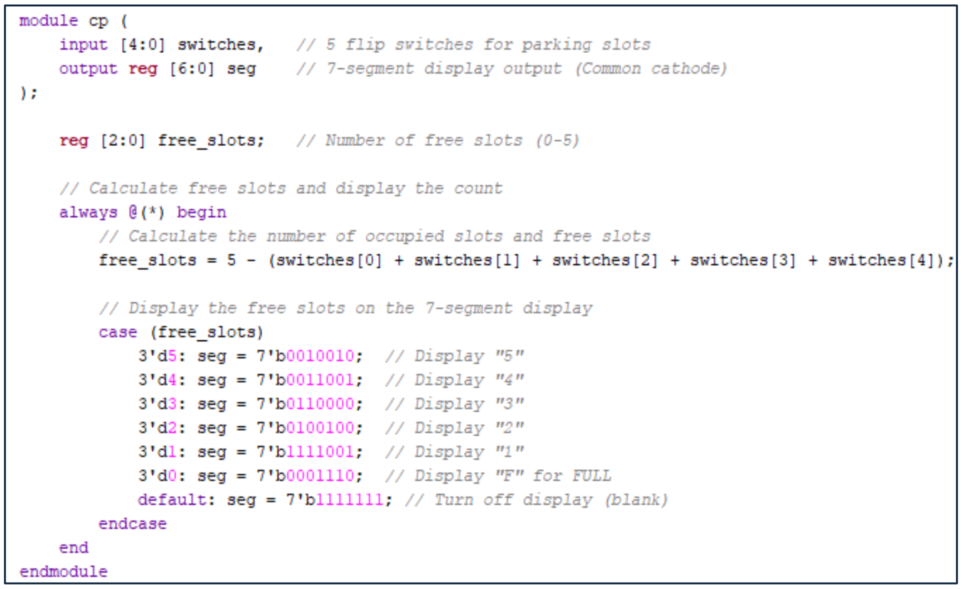
3. **Seven-Segment Display**: The available number of parking spots is displayed on the seven-segment display. It dynamically reflects the current status of the parking lot as cars are parked or removed.

This simplified system uses a single switch for both operations, reducing hardware complexity while maintaining accurate parking management. The design can be scaled for different parking lot sizes by adjusting the counter to represent the maximum capacity of the lot.

CHAPTER 2

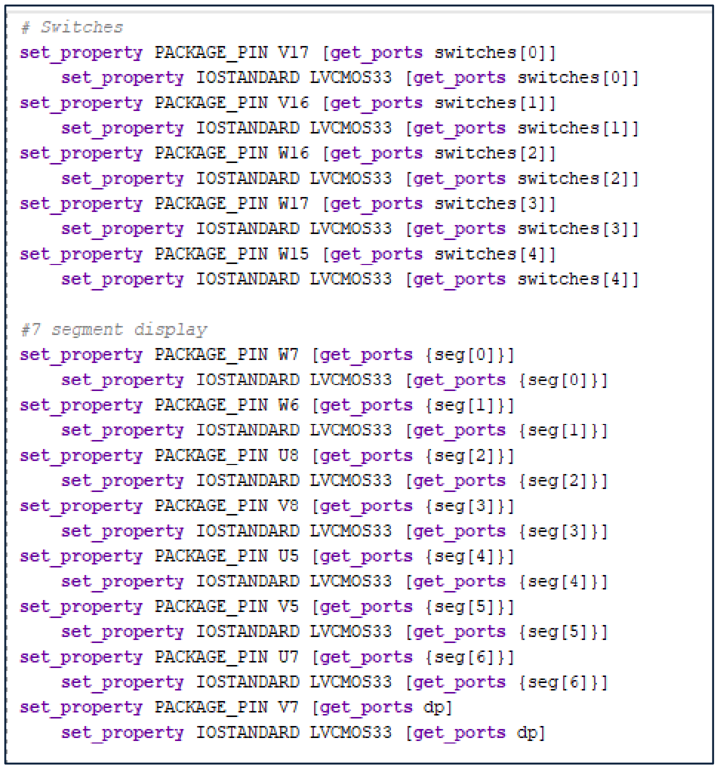
IMPLEMENTATION

CODE:



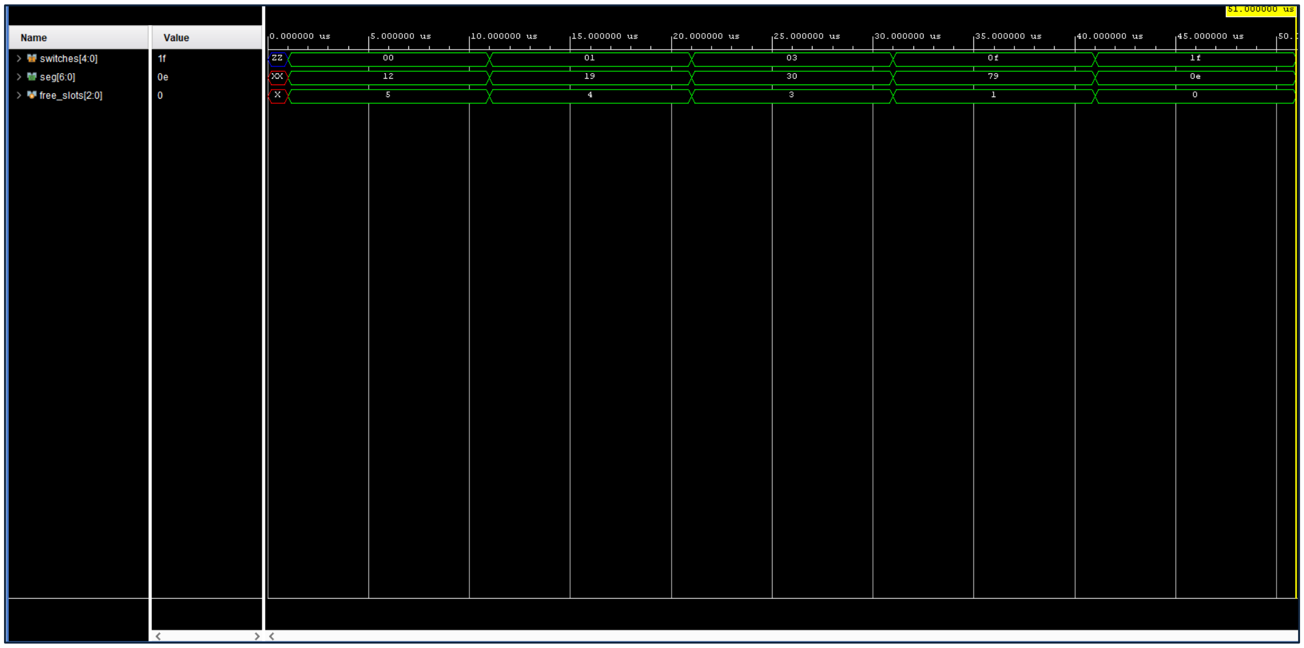
```
module cp (
    input [4:0] switches,   // 5 flip switches for parking slots
    output reg [6:0] seg    // 7-segment display output (Common cathode)
);

    reg [2:0] free_slots;   // Number of free slots (0-5)

    // Calculate free slots and display the count
    always @(*) begin
        // Calculate the number of occupied slots and free slots
        free_slots = 5 - (switches[0] + switches[1] + switches[2] + switches[3] + switches[4]);

        // Display the free slots on the 7-segment display
        case (free_slots)
            3'd5: seg = 7'b0010010;  // Display "5"
            3'd4: seg = 7'b0011001;  // Display "4"
            3'd3: seg = 7'b0110000;  // Display "3"
            3'd2: seg = 7'b0100100;  // Display "2"
            3'd1: seg = 7'b1111001;  // Display "1"
            3'd0: seg = 7'b0001110;  // Display "F" for FULL
            default: seg = 7'b1111111; // Turn off display (blank)
        endcase
    end
endmodule
```

CONSTRAINT FILES:



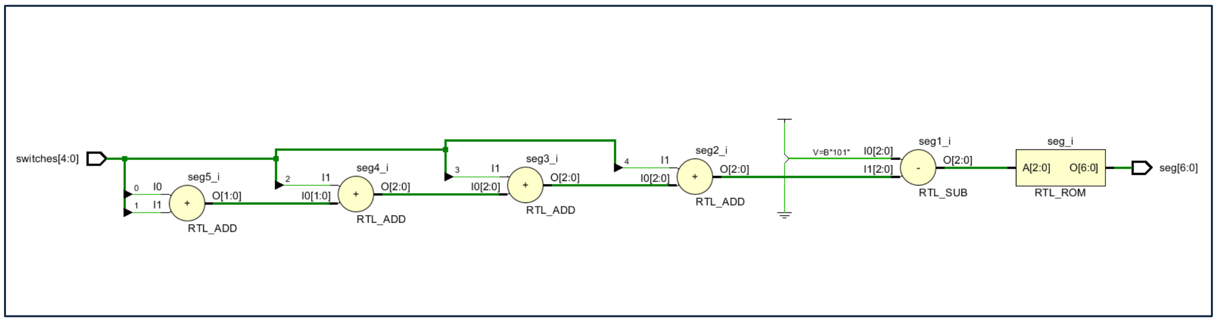
```
# Switches
set_property PACKAGE_PIN V17 [get_ports switches[0]]
    set_property IOSTANDARD LVCMOS33 [get_ports switches[0]]
set_property PACKAGE_PIN V16 [get_ports switches[1]]
    set_property IOSTANDARD LVCMOS33 [get_ports switches[1]]
set_property PACKAGE_PIN W16 [get_ports switches[2]]
    set_property IOSTANDARD LVCMOS33 [get_ports switches[2]]
set_property PACKAGE_PIN W17 [get_ports switches[3]]
    set_property IOSTANDARD LVCMOS33 [get_ports switches[3]]
set_property PACKAGE_PIN W15 [get_ports switches[4]]
    set_property IOSTANDARD LVCMOS33 [get_ports switches[4]]

#7 segment display
set_property PACKAGE_PIN W7 [get_ports {seg[0]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {seg[0]}]
set_property PACKAGE_PIN W6 [get_ports {seg[1]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {seg[1]}]
set_property PACKAGE_PIN U8 [get_ports {seg[2]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {seg[2]}]
set_property PACKAGE_PIN V8 [get_ports {seg[3]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {seg[3]}]
set_property PACKAGE_PIN U5 [get_ports {seg[4]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {seg[4]}]
set_property PACKAGE_PIN V5 [get_ports {seg[5]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {seg[5]}]
set_property PACKAGE_PIN U7 [get_ports {seg[6]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {seg[6]}]
set_property PACKAGE_PIN V7 [get_ports dp]
    set_property IOSTANDARD LVCMOS33 [get_ports dp]
```

SIMULATION:



SYNTHESIZE THE DESIGN:

• After writing the code, click **Run Synthesis** in the **Flow Navigator**. • If there are any errors in your code, resolve them and rerun the synthesis.

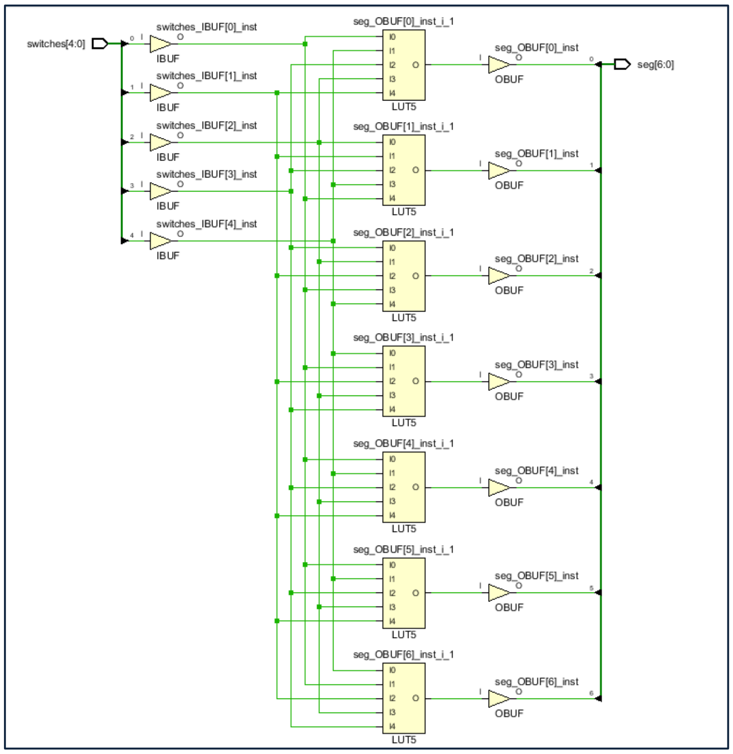


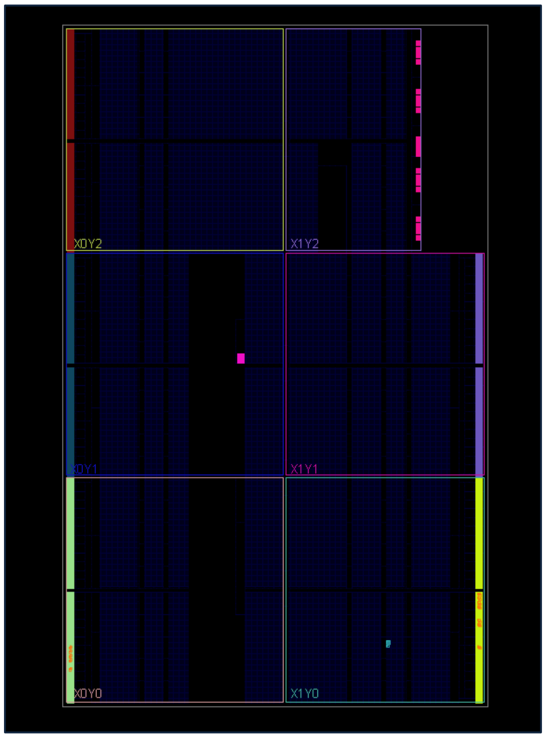
Netlist

IMPLEMENT THE DESIGN:

Once synthesis is complete, click **Run Implementation** in the **Flow Navigator**.

- This step maps your design to the actual FPGA resources.





GENERATE BITSTREAM:

After the implementation is completed, click **Generate Bit stream**. This generates the bit file, which is required to program your FPGA.

• If prompted to save the design, click **Yes.**

CONFIGURE THE FPGA (DOWNLOAD BITSTREAM):

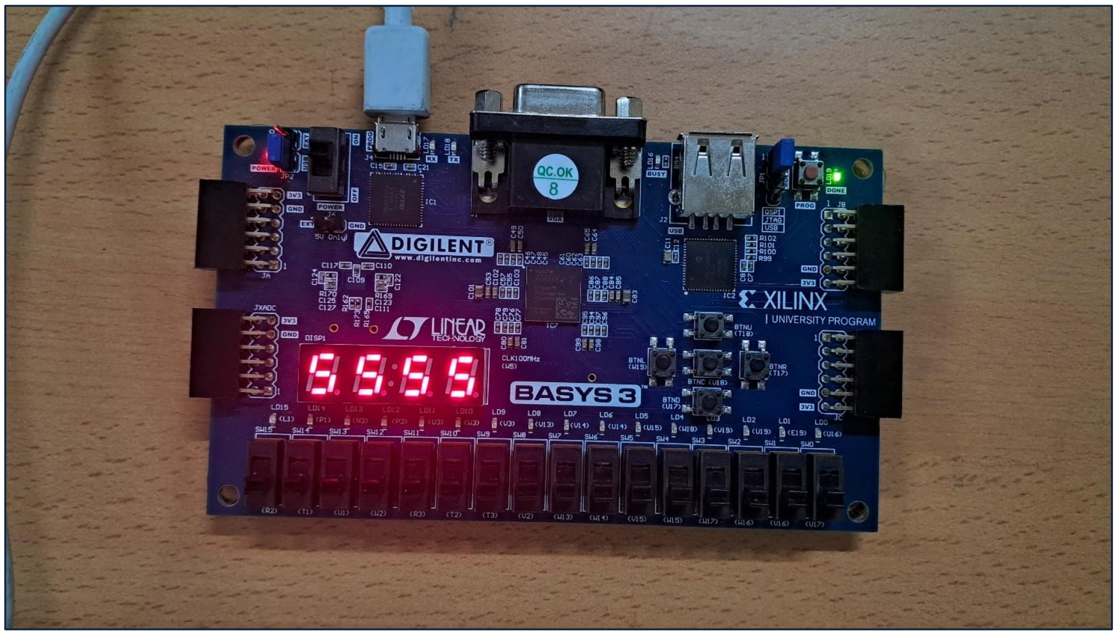
Once the bit stream file is generated, open the **Hardware Manager** by clicking **Open Hardware Manager** in the Flow Navigator.

• Click **Open Target** > **Auto Connect** to connect to your FPGA board.

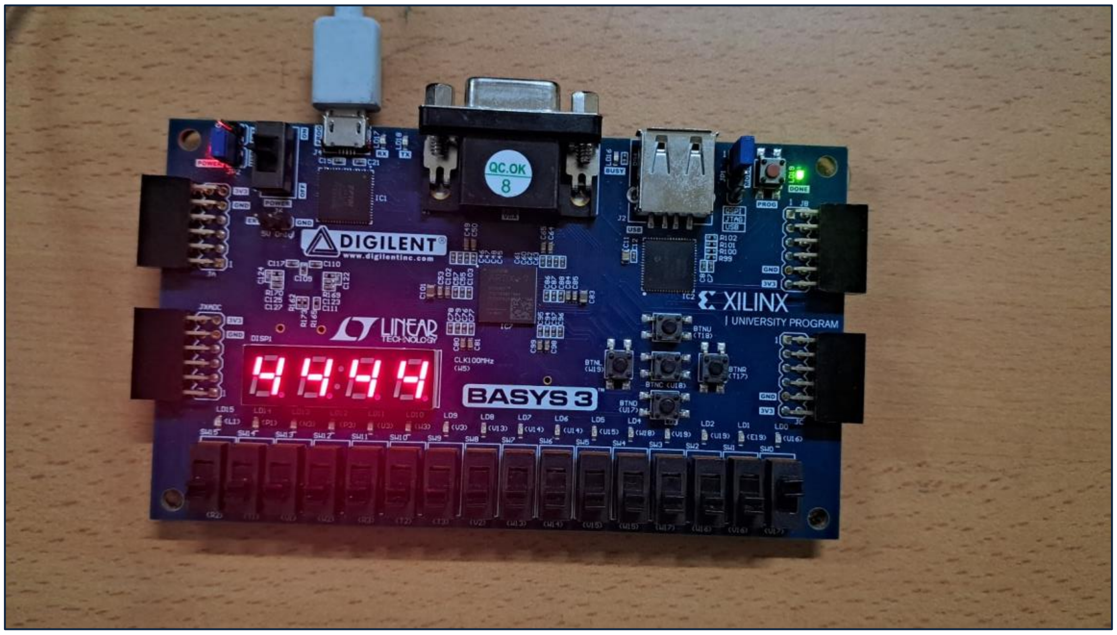
• After the board is connected, click **Program Device** and select the generated bit stream file (bit).

CHAPTER 3

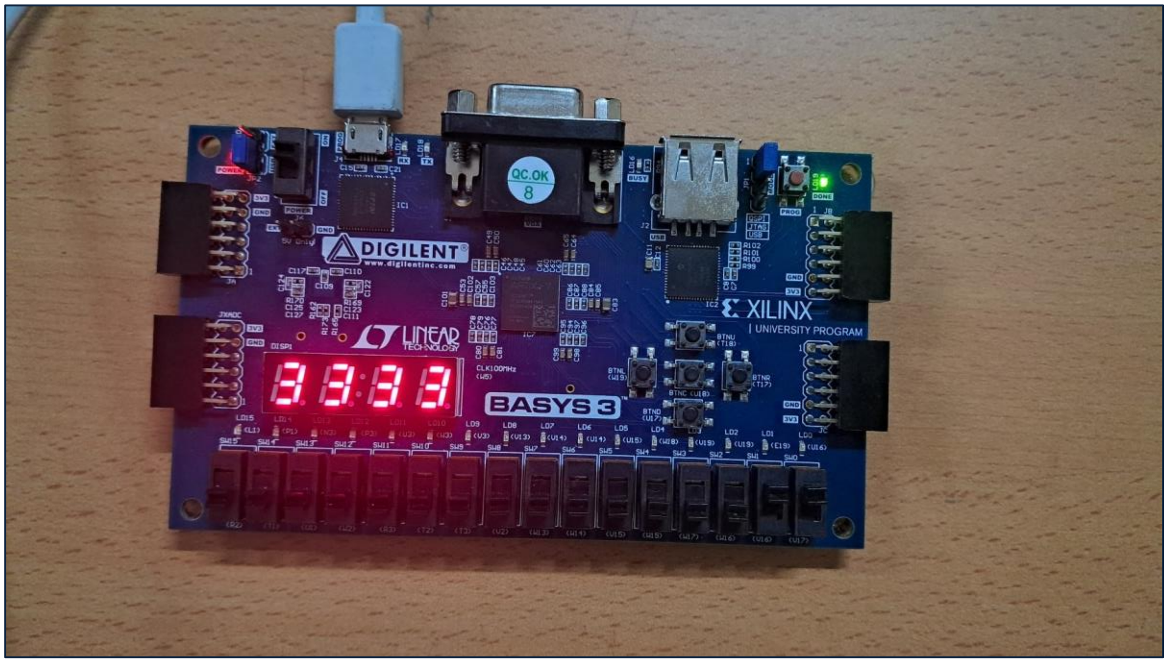
TESTING OF OUTPUT

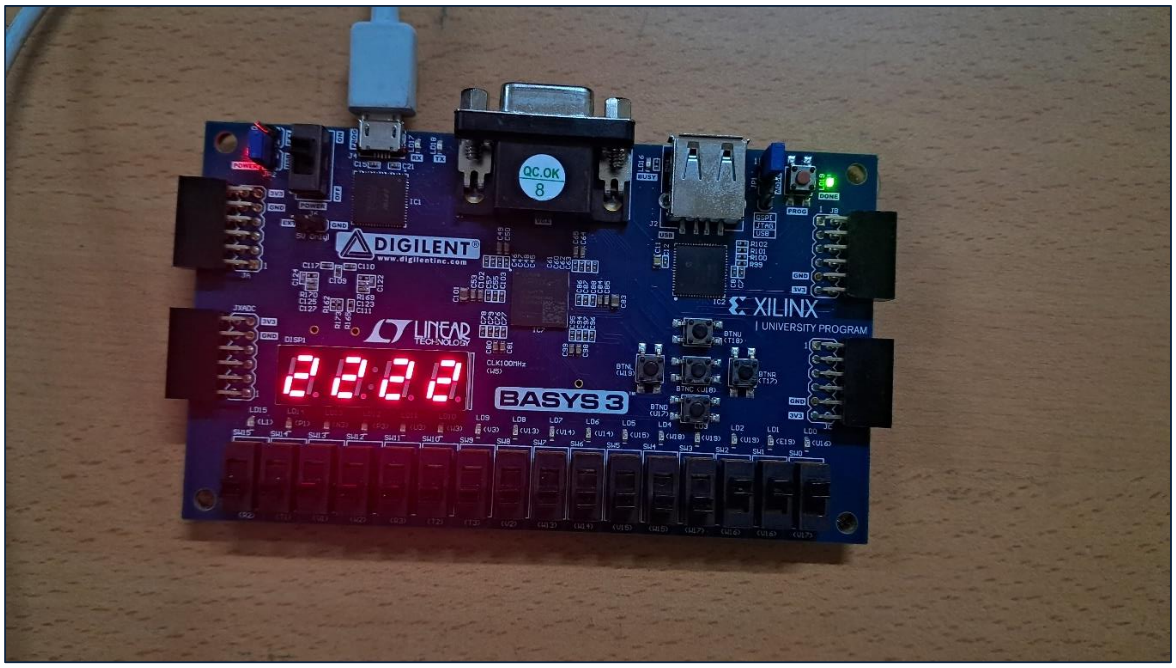


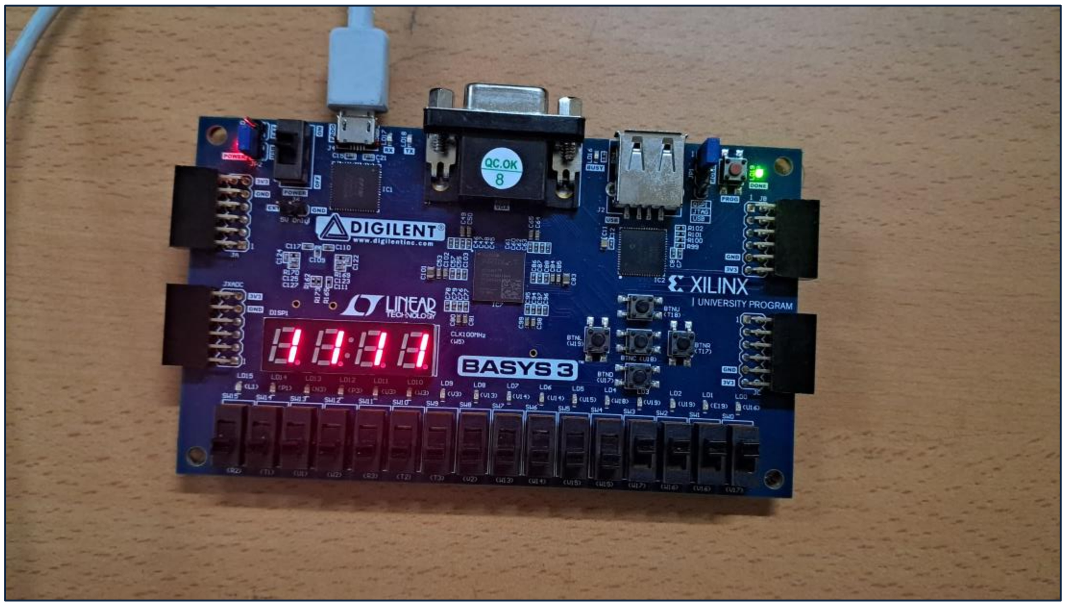
No cars parked, all switches are off, and the count is 5 (all available).



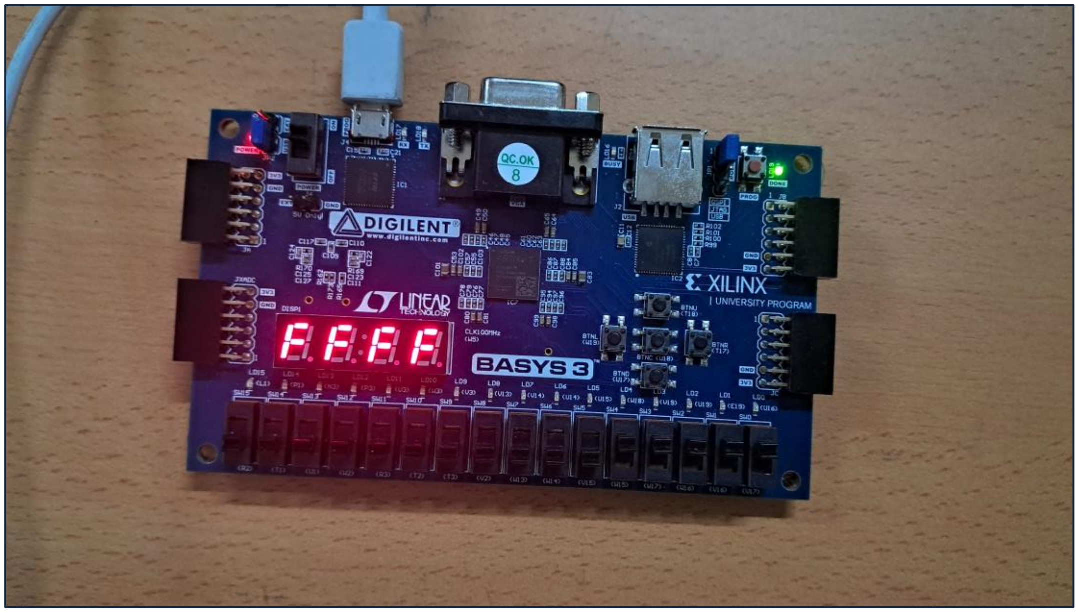
One car is parked, one switch is on, and the count is 4 (four available).


Two cars are parked, two switches are on, and the count is 3 (three available).


Three cars are parked, three switches are on, and the count is 2 (two available).



Four cars are parked, four switches are on, and the count is 1 (one available).



All spaces are occupied, all switches are on, and the count is F (full).

CHAPTER 4

CONCLUSION

The implementation of a car parking system using FPGA technology, such as the Basys 3 board, represents a significant advancement in managing parking resources efficiently. This system not only provides real-time visibility of available parking spaces but also automates the monitoring process, reducing congestion and improving the overall user experience. By integrating various features, such as LED indicators and seven-segment displays, the system offers a user-friendly interface that effectively communicates parking availability.

The flexibility of FPGA technology allows for further enhancements, such as IoT integration, automated fee calculation, and advanced data analytics, making this solution suitable for a wide range of applications, including smart cities, shopping centres, and residential areas. As urban populations continue to grow and parking becomes increasingly scarce, the deployment of intelligent parking solutions like this will be essential in optimizing space utilization and facilitating smoother traffic flow.

Overall, the FPGA-based car parking system serves as a foundational step towards smarter, more efficient parking management solutions, paving the way for future innovations in transportation and urban planning.