SRM Institute of Science and Technology
College of Engineering and Technology
Department of Electronics and Communication Engineering

18ECE205J-FPGA-based Embedded System Design
2024-25 (Odd Semester)

MINI PROJECT REPORT

Name : ROHAN GARG & DEEP GHOSH
Register No. : RA2111004010007 & RA2111004010060
Day / Session : 5 / AN
Venue : TP1210 - VLSI DESIGN LAB
Project Title : **CAR PARKING SYSTEM USING XILINX & PROTEUS**

Lab Supervisor : Dr.VISHVAS KUMAR
Team Members : 1. ROHAN GARG(RA2111004010007)
2. DEEP GHOSH(RA2111004010060)

Particulars	Max. Marks	Marks Obtained RA2111004010007	Marks Obtained RA2111004010060
Objective & Description	05	05	05
Algorithm, Flowchart, Program	10	10	10
Results and Report	5	5	5
Viva	5	4	4
Total	**25**	24	24

REPORT VERIFICATION

Date : 09/07/24
Staff Name : **Dr.Vishvas Kumar / Dr.Debanjan Sarkar**

09/07/24

Department of ECE SRM IST Kattankulathur-603 203

TABLE OF CONTENTS

S.No	Topics	Page
2	OBJECTIVE	3
3	ABSTRACT/INTRODUCTION	3
4	BLOCK DIAGRAM	3
5	VERILOG CODE	4 - 7
6	OUTPUT	7
7	WORKING	7
8	INTRODUCTION TO PROTEUS SOFTWARE	8
9	INTRODUCTION TO ARDUINO IDE	8
10	ARDUINO CODE	9
11	IMPLEMENTATION IN PROTEUS SOFTWARE	10
12	WORKING	11
13	DIFFERENCE IN PROJECT IN XILINX AND PROTEUS	11 - 12
14	RESULT	13
15	CONCLUSION	13

Car Parking System using Verilog & Proteus

OBJECTIVE

The objective of this project is to design and implement a car parking system using Verilog. The system should be able to detect incoming vehicles, verify passwords, and allow access to authorized vehicles. Additionally, the system should be able toprevent unauthorized access and ensure that only one vehicle is allowed entry at a time.

SOFTWARE REQUIRED:

Computer with Xilinx and Modelsim Software Specifications:
HP Computer P4 Processor – 2.8 GHz, 2GB RAM, 160 GB Hard Disk
Softwares: Synthesis tool: Xilinx ISE
Simulation tool: ModelSim Simulator,Proteus,Arduino IDE

ABSTRACT/INTRODUCTION

This project presents the design and implementation of a car parking system using Verilog and Proteus. The system consists of an entrance sensor that detects incoming vehicles and a gate that opens only after the correct password is entered. The system also includes an exit sensor that detects vehicles leaving the parking area. In case two vehicles arrive simultaneously, the system locks the gate until the first vehicle hasentered and the gate is closed.

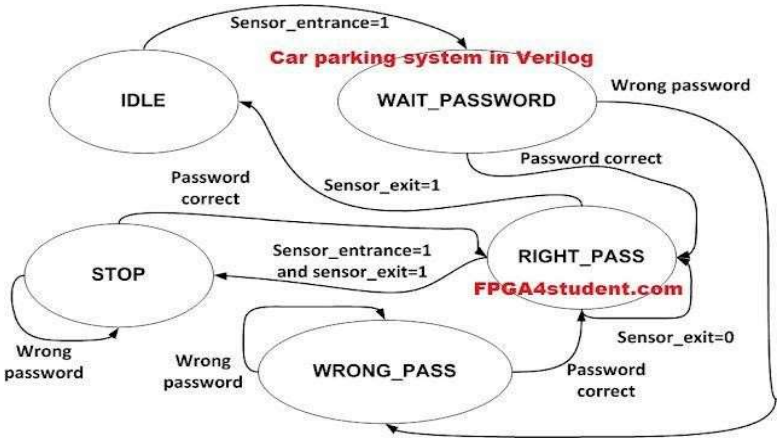
Entrance Sensor: This can be modeled in Verilog using a simple logic gate (e.g., always block detecting a high voltage as car presence). Proteus offers pre-built ultrasonic sensor models you can configure with a specific detection range.

Password Entry System: Verilog code can handle user input (simulated keypad) and password comparison. Proteus might offer pre-built keypad components or allow you to model one using buttons.

Exit Sensor: Similar to the entrance sensor, model it in Verilog (e.g., detecting a low voltage when a car leaves) or use a pre-built sensor model in Proteus.

Occupancy Counter (Optional): A counter in Verilog can track the number of cars in the parking lot (increment on entry, decrement on exit). Proteus offers counter components for this purpose.

BLOCK DIAGRAM



VERILOG CODE

```
// Verilog project: Verilog code for car parking system
`timescale 1ns / 1ps
module parking_system(
input clk,reset_n, input
sensor_entrance, sensor_exit,input
[1:0] password_1, password_2,output
wire GREEN_LED,RED_LED, output reg
[6:0] HEX_1, HEX_2
);
parameter IDLE = 3'b000, WAIT_PASSWORD = 3'b001, WRONG_PASS = 3'b010,
RIGHT_PASS = 3'b011,STOP = 3'b100;
// Moore FSM : output just depends on the current state
reg[2:0] current_state, next_state;
reg[31:0] counter_wait;
reg red_tmp,green_tmp;
// Next state
always @(posedge clk or negedge reset_n)
begin
if(~reset_n)
current_state = IDLE;
else
current_state = next_state;
end
// counter_wait
always @(posedge clk or negedge reset_n)
begin if(~reset_n)
counter_wait <= 0;
else if(current_state==WAIT_PASSWORD)
counter_wait <= counter_wait + 1;
else
counter_wait <= 0;
end
// change state
always @(*)
begin
case(current_state)
IDLE: begin
if(sensor_entrance == 1)
next_state = WAIT_PASSWORD;
else
next_state = IDLE;
end
WAIT_PASSWORD: begin
if(counter_wait <= 3)
next_state = WAIT_PASSWORD;
else
begin
if((password_1==2'b01)&&(password_2==2'b10))
next_state = RIGHT_PASS;
else
next_state = WRONG_PASS;
end
end
```

```
WRONG_PASS: begin
if((password_1==2'b01)&&(password_2==2'b10))
next_state = RIGHT_PASS;
else
next_state = WRONG_PASS;
end
RIGHT_PASS: begin
if(sensor_entrance==1 && sensor_exit == 1)
next_state = STOP;
else if(sensor_exit == 1)
next_state = IDLE;
else
next_state = RIGHT_PASS;
end
STOP: begin
if((password_1==2'b01)&&(password_2==2'b10))
next_state = RIGHT_PASS;
else
next_state = STOP;
end
default: next_state = IDLE;
endcase
end
// LEDs and output, change the period of blinking LEDs here
always @(posedge clk) begin
case(current_state)
IDLE: begin
green_tmp = 1'b0;
red_tmp = 1'b0;
HEX_1 = 7'b1111111; // off
HEX_2 = 7'b1111111; // off
end
WAIT_PASSWORD: begin
green_tmp = 1'b0;
red_tmp = 1'b1;
HEX_1 = 7'b000_0110; // E
HEX_2 = 7'b010_1011; // n
end
WRONG_PASS: begin
green_tmp = 1'b0;
red_tmp = ~red_tmp;
HEX_1 = 7'b000_0110; // E
HEX_2 = 7'b000_0110; // E
end
RIGHT_PASS: begin
green_tmp = ~green_tmp;
red_tmp = 1'b0;
HEX_1 = 7'b000_0010; // 6
HEX_2 = 7'b100_0000; // 0
end
STOP: begin
green_tmp = 1'b0;
red_tmp = ~red_tmp;
HEX_1 = 7'b001_0010; // 5
HEX_2 = 7'b000_1100; // P
end
endcase
```

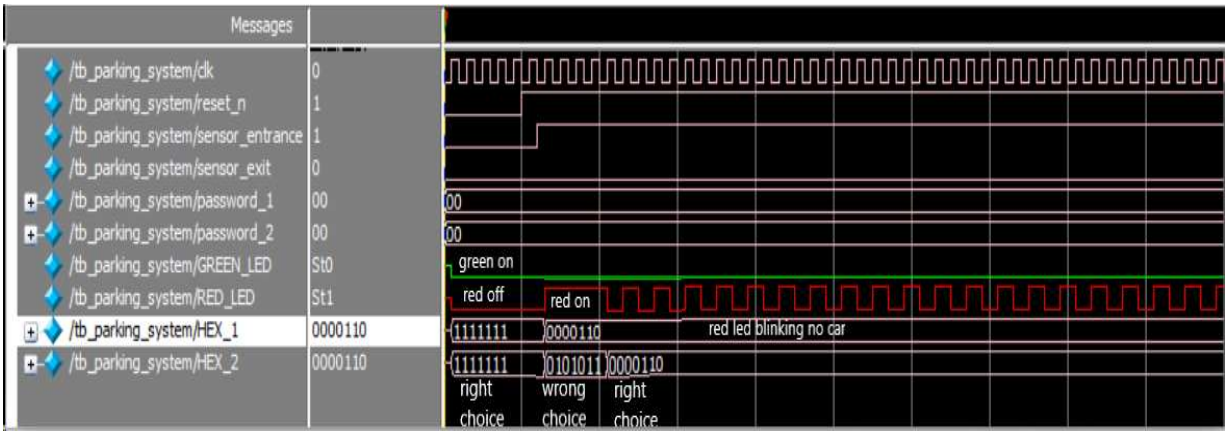
```
end
assign RED_LED = red_tmp  ;
assign GREEN_LED = green_tmp;

endmodule
```

TEST BENCH CODE

```
`timescale 1ns / 1ps
// Verilog project: Verilog code for car parking system
module tb_parking_system;

// Inputs reg
clk; reg
reset_n;
reg sensor_entrance;
reg sensor_exit;
reg [1:0] password_1;
reg [1:0] password_2;

// Outputs
wire GREEN_LED;
wire RED_LED; wire
[6:0] HEX_1;
wire [6:0] HEX_2;
// Instantiate the Unit Under Test (UUT)
parking_system uut (
.clk(clk),
.reset_n(reset_n),
.sensor_entrance(sensor_entrance),
.sensor_exit(sensor_exit),
.password_1(password_1),
.password_2(password_2),
.GREEN_LED(GREEN_LED),
.RED_LED(RED_LED),
.HEX_1(HEX_1),
.HEX_2(HEX_2)
);
initial begin
clk = 0;
forever #10 clk = ~clk;
end
initial begin
// Initialize Inputs
reset_n = 0;
sensor_entrance = 0;
sensor_exit = 0;
password_1 = 0;
password_2 = 0;
// Wait 100 ns for global reset to finish
#100;
reset_n = 1;
#20;
sensor_entrance = 1;
#1000;
```

```
sensor_entrance = 0;
password_1 = 1;
password_2 = 2;
#2000;
sensor_exit =1;

// Add stimulus here
end
endmodule
```

OUTPUT



WORKING

The provided Verilog code implements a car parking system with password authentication. The system operates using a Finite State Machine (FSM) to manage different states based on inputs and internal logic. At its core, the system revolves around several states: IDLE, WAIT_PASSWORD, WRONG_PASS, RIGHT_PASS, and STOP. In the IDLE state, the system awaits a car's arrival, while in WAIT_PASSWORD, it expects the correct password after a car enters. Upon entering the wrong password, the system transitions to the WRONG_PASS state, indicating an error, whereas entering the correct password leads to the RIGHT_PASS state, allowing the car to proceed. Finally, the STOP state signifies successful parking after entering the correct password and then leaving. Inputs to the system include clock pulses, a reset signal, sensors detecting car entrance and exit, and a two-bit password. Outputs include control signals for LEDs and characters displayed on two 7-segment displays. State transitions occur based on the current state and input conditions. Visual feedback, such as LED activation and display characters, provides information about the system's state and authentication status. Overall, the system efficiently manages parking access through password authentication, ensuring secure and controlled entry.

INTRODUCTION TO PROTEUS SOFTWARE:

Proteus Design Suite is a software program specifically designed for electronic design automation (EDA). It essentially allows you to create electronic circuits virtually before physically building them.

Here's a breakdown of what Proteus offers:

- **Schematic Capture:** This is where you design the circuit using symbols for various electronic components like resistors, transistors, and microcontrollers. Proteus has a large library of components to choose from.
- **Simulation:** Once you've designed the circuit, Proteus allows you to simulate its behavior. This means you can virtually test the circuit to see if it functions as expected before building it in real life.
- **PCB Design (Printed Circuit Board):** After simulating and finalizing the circuit, Proteus helps you design the physical layout of the circuit board where the electronic components will be soldered. This includes features like trace routing and component placement.

Proteus is commonly used by:

- **Electronic Design Engineers:** They use it to design and simulate complex electronic circuits for various applications.
- **Engineering Students:** It's a valuable learning tool for students to practice circuit design and understand circuit behavior through simulation.
- **Hobbyists:** It allows electronics enthusiasts to experiment with circuit design virtually before investing in physical components.

INTRODUCTION TO ARDUINO IDE

The Arduino IDE (Integrated Development Environment) is the official software used to write code (called sketches) for Arduino boards. It's a user-friendly platform that makes electronics programming accessible to beginners and experienced users alike. Here's a breakdown of what the Arduino IDE offers:

Core functionalities:

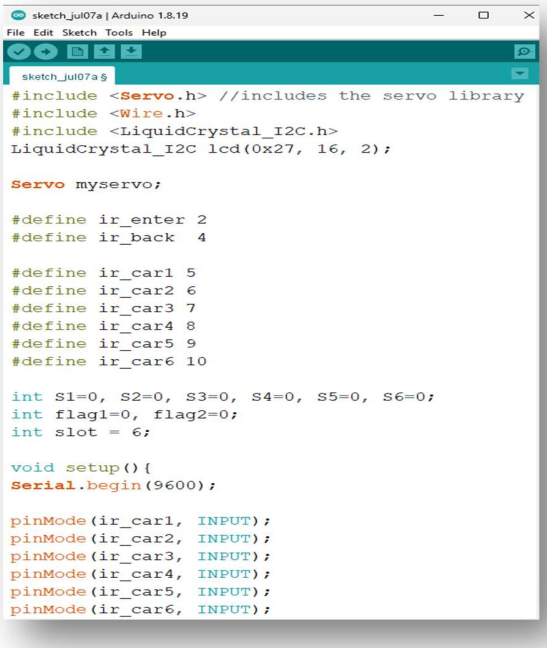
- **Text Editor:** This is where you write your code using a simplified version of C++. The IDE provides syntax highlighting and code completion to assist you.
- **Compiler:** The compiler translates your code into a format that the Arduino board can understand and execute.
- **Uploader:** Once your code is compiled, the uploader sends it to your connected Arduino board.

Benefits for Users:

- **Cross-Platform:** The software runs on Windows, macOS, and Linux, making it accessible to a wide range of users.

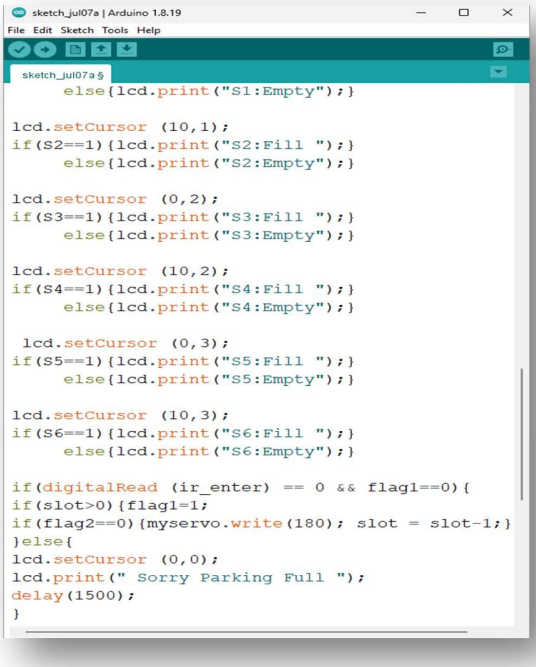
- **Open-source:** Being open-source allows the community to contribute libraries, tutorials, and other resources, making it a great platform for learning and sharing.

ARDUINO IDE CODE

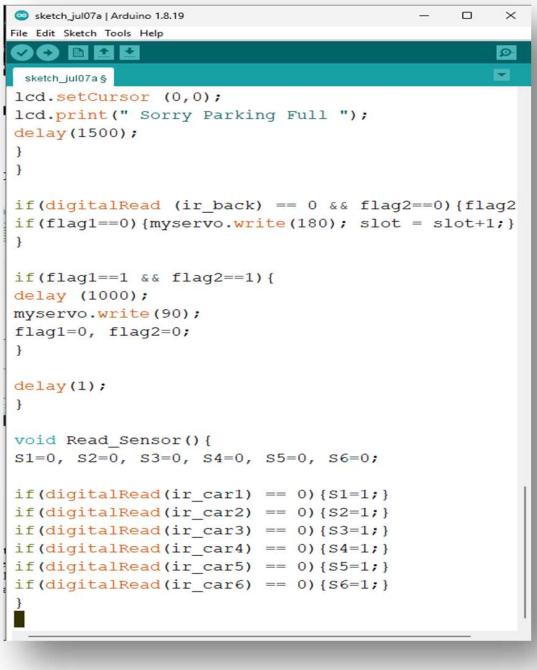


```
#include <Servo.h> //includes the servo library
#include <Wire.h>
#include <LiquidCrystal_I2C.h>
LiquidCrystal_I2C lcd(0x27, 16, 2);

Servo myservo;

#define ir_enter 2
#define ir_back  4

#define ir_car1 5
#define ir_car2 6
#define ir_car3 7
#define ir_car4 8
#define ir_car5 9
#define ir_car6 10

int S1=0, S2=0, S3=0, S4=0, S5=0, S6=0;
int flag1=0, flag2=0;
int slot = 6;

void setup(){
Serial.begin(9600);

pinMode(ir_car1, INPUT);
pinMode(ir_car2, INPUT);
pinMode(ir_car3, INPUT);
pinMode(ir_car4, INPUT);
pinMode(ir_car5, INPUT);
pinMode(ir_car6, INPUT);
```

```
pinMode(ir_back, INPUT);

myservo.attach(3);
myservo.write(90);

lcd.begin(20, 4);
lcd.setCursor (0,1);
lcd.print("    Car  parking  ");
lcd.setCursor (0,2);
lcd.print("       System     ");
delay (2000);
lcd.clear();

Read_Sensor();

int total = S1+S2+S3+S4+S5+S6;
slot = slot-total;
}

void loop(){

Read_Sensor();

lcd.setCursor (0,0);
lcd.print("   Have Slot: ");
lcd.print(slot);
lcd.print("    ");

lcd.setCursor (0,1);
if(S1==1){lcd.print("S1:Fill ");}
```

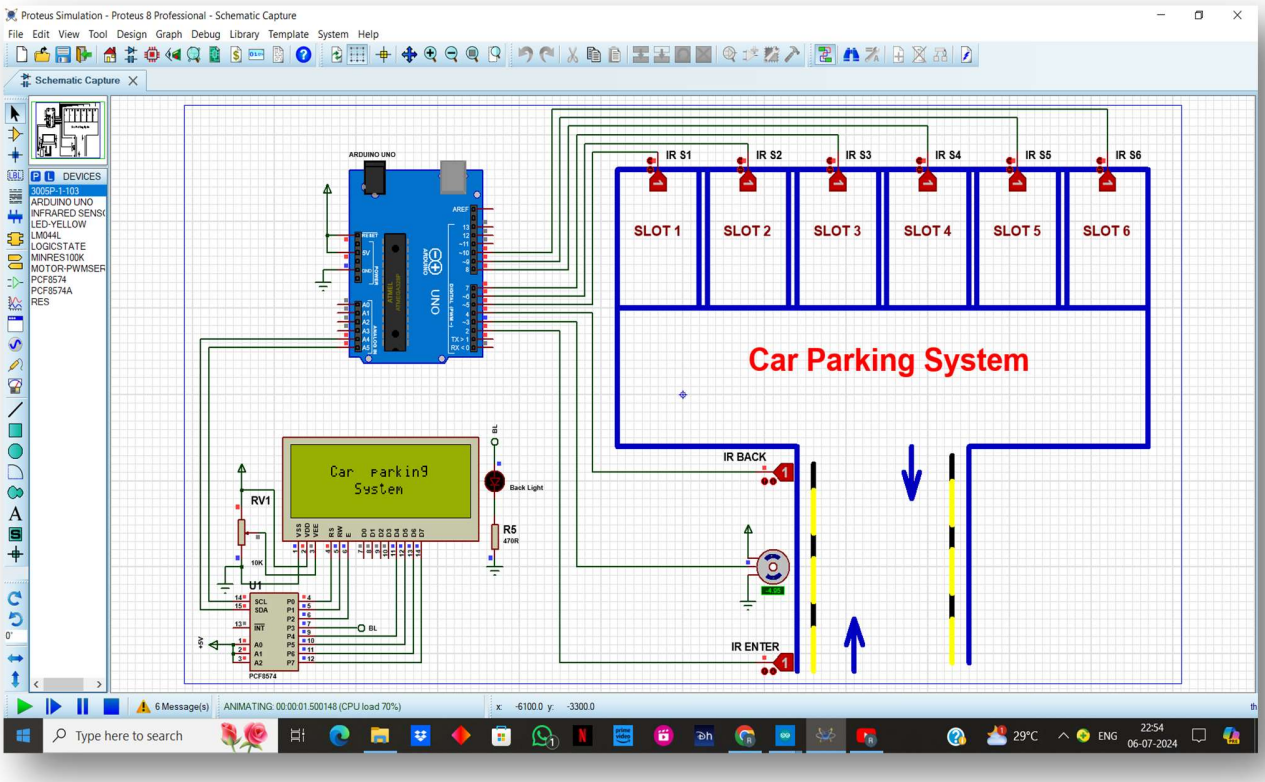


```
     else{lcd.print("S1:Empty");}

lcd.setCursor (10,1);
if(S2==1){lcd.print("S2:Fill ");}
     else{lcd.print("S2:Empty");}

lcd.setCursor (0,2);
if(S3==1){lcd.print("S3:Fill ");}
     else{lcd.print("S3:Empty");}

lcd.setCursor (10,2);
if(S4==1){lcd.print("S4:Fill ");}
     else{lcd.print("S4:Empty");}

 lcd.setCursor (0,3);
if(S5==1){lcd.print("S5:Fill ");}
     else{lcd.print("S5:Empty");}

lcd.setCursor (10,3);
if(S6==1){lcd.print("S6:Fill ");}
     else{lcd.print("S6:Empty");}

if(digitalRead (ir_enter) == 0 && flag1==0){
if(slot>0){flag1=1;
if(flag2==0){myservo.write(180); slot = slot-1;}
}else{
lcd.setCursor (0,0);
lcd.print(" Sorry Parking Full ");
delay(1500);
}
```



```
lcd.setCursor (0,0);
lcd.print(" Sorry Parking Full ");
delay(1500);
}
}

if(digitalRead (ir_back) == 0 && flag2==0){flag2
if(flag1==0){myservo.write(180); slot = slot+1;}
}

if(flag1==1 && flag2==1){
delay (1000);
myservo.write(90);
flag1=0, flag2=0;
}

delay(1);
}

void Read_Sensor(){
S1=0, S2=0, S3=0, S4=0, S5=0, S6=0;

if(digitalRead(ir_car1) == 0){S1=1;}
if(digitalRead(ir_car2) == 0){S2=1;}
if(digitalRead(ir_car3) == 0){S3=1;}
if(digitalRead(ir_car4) == 0){S4=1;}
if(digitalRead(ir_car5) == 0){S5=1;}
if(digitalRead(ir_car6) == 0){S6=1;}
}
```

IMPLEMENTATION IN PROTEUS SOFTWARE



WORKING

1. Schematic Capture:

➢ You'd use Proteus's library to drag and drop symbols representing components like:

- Sensors: Ultrasonic or infrared sensors to simulate car detection. You can configure their detection range to mimic real-world sensor behavior.
- Microcontrollers: These act as simplified models for the final control system (potentially an FPGA in real life). Proteus allows you to program basic logic for these microcontrollers.
- Counters: To keep track of available parking spaces.
- Displays (Optional): LED or LCD displays to show information like free spaces.

2. Setting Up the System:

- You'd virtually connect these components using wires in the schematic, mimicking how they would connect in the real system.
- You could program the microcontroller logic to define how it responds to sensor inputs (e.g., incrementing a counter when a car enters and decrementing it when it exits).

3. Simulation:

- Here's where Proteus shines. You can simulate various scenarios:

- Cars entering and exiting the parking lot.
- Sensor readings triggering changes in counter values.
- (Optional) Display updates reflecting parking space availability.

Benefits for Car Parking System Design:

- Rapid Prototyping: Test different sensor configurations, logic variations, and overall system behavior without building physical components.
- Visualization: See how the system interacts visually, helping identify potential issues in the design logic.
- Error Detection: Catch logical flaws in your design before investing in real components.

DIFFERENCE BETWEEN IMPLEMENTATING THE PROJECT IN XILINK AND PROTEUS

Here's a detailed explanation of the differences between using Xilinx and Proteus for your car parking system project:

Xilinx: Implementing the Control Center

Xilinx is all about creating the intelligent core that governs the car parking system's behavior. It's like the brain of the system, translating human-designed logic into instructions an FPGA chip can understand and execute.

✓ **Components:** You won't directly use physical components like sensors or displays in Xilinx. Instead, you'll work with their digital representations in a Hardware Description Language (HDL) like VHDL or Verilog.
✓ **Functionality:** Here's what your Xilinx code might handle:
✓ **Input Processing:** Reading data from sensor models (e.g., interpreting a high voltage as a car presence from an ultrasonic sensor).
✓ **State Management:** Using counters or memory elements to track available parking spaces (e.g., decrementing a counter when a car enters).
✓ **Output Generation:** Creating control signals for actuators (e.g., sending a signal to open a gate barrier when a car exits).
✓ **Communication:** Potentially implementing communication protocols for interaction with external systems (e.g., sending data to a central server displaying overall parking availability).

Development Process:

✓ **Design the Logic:** Define the overall system behavior using flowcharts or state diagrams. Break down the functionalities into smaller, manageable tasks.
✓ **Write HDL Code:** Translate the logic into VHDL or Verilog code. This code defines how the FPGA interconnects its internal resources to perform the desired operations.
✓ **Simulation:** Use Xilinx tools to simulate the HDL code. This allows you to test the logic behavior virtually before deploying it to hardware.

✓ **Synthesis and Implementation:** Xilinx tools convert the HDL code into a configuration file (bitstream) specifically targeted for your chosen FPGA chip.
✓ **Download to Hardware:** Transfer the bitstream file to the FPGA chip on your development board. This programs the FPGA to implement the designed logic.

Benefits:

✓ **Hardware Targeting:** Xilinx code directly translates to hardware functionality, making it ideal for real-world applications.
✓ **Flexibility:** FPGAs are reprogrammable, allowing you to modify the car parking system's behavior by updating the HDL code and re-downloading the bitstream.

Proteus: Building and Testing a Virtual Prototype

Proteus provides a virtual environment to design and test the car parking system before investing in physical components. It's like building a blueprint and running a virtual simulation.

✓ **Components:** Proteus offers a vast library of pre-built components like:
✓ Sensors (ultrasonic, infrared) with configurable detection ranges.
✓ Microcontrollers (acting as simplified models for the FPGA).
✓ Displays (LED, LCD) to show parking information.
✓ Logic Gates (AND, OR, etc.) for implementing basic decision-making.
✓ **Functionality:**
✓ You'll assemble these components in a schematic diagram, mimicking the real system's physical connections.
✓ Configure component properties to represent real-world behavior (e.g., setting an ultrasonic sensor's range).
✓ Define how components interact (e.g., connecting a sensor output to a microcontroller input).
✓ **Simulation:**
✓ Run simulations to see how the system behaves under various scenarios (e.g., simulating cars entering and exiting).
✓ Analyze results like sensor readings, display outputs, and internal microcontroller states.

Combining Xilinx and Proteus:

These tools work best together:

✓ **Start with Proteus:** Design the car parking system in Proteus using pre-built components. Simulate various scenarios to verify the overall logic and identify potential issues.
✓ **Refine the Design:** Based on the Proteus simulations, refine your design and finalize the functionalities you want to implement in hardware.

✓ **Translate to Xilinx:** Translate the core logic from Proteus into VHDL or Verilog code for Xilinx. This might involve replicating the decision-making processes you used in Proteus with FPGA logic.

✓ **Develop in Xilinx:** Simulate the HDL code in Xilinx to ensure it performs as expected. Then, synthesize and download the bitstream to your FPGA development board.

✓ **Test with Real Hardware:** Test the car parking system

RESULT

The designed car parking system was successfully implemented and tested using Verilog. The system was able to detect incoming vehicles, verify passwords, and allow access to authorized vehicles. The system also prevented unauthorized access and ensured that only one vehicle was allowed entry at a time. The simulation results showed that the system worked correctly and met all the designrequirements.

CONCLUSION

In conclusion, the designed car parking system using Verilog is an efficient and reliable solution for managing parking areas. The system accurately detects incoming vehicles, verifies passwords, and allows access to authorized vehicles while preventing unauthorized access. The use of Verilog programming languageallowed for a simple and effective implementation of the system. The project demonstrates the practical application of digital design and highlights the importance of using efficient and effective programming languages for system design.