B. Tech. 3rd Semester

Design of Digital Systems (CS201)

Mini-Project Final Report

On

Automated Parking Detector and Ticket Price Calculator

Rohith Shinoj Kumar

(211CS245, 8848790113, rohith.211cs245@nitk.edu.in)

Rishi Diwaker

(211CS243, 9968965073, rishidiwaker.211cs243@nitk.edu.in)

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

NATIONAL INSTITUTE OF TECHNOLOGY KARNATAKA,

SURATHKAL, MANGALORE - 575025

January, 2023

1. Abstract

Places like Airports, malls, festive centres, etc always attract a lot of people who come from faraway places for various such as travel, entertainment, etc. Many of these people tend to come in their own private vehicles as our public transport infrastructure is not always reliable and people are more comfortable to travel in their own vehicles. However, this poses a serious problem of parking and related management issues as people will need spots to park their cars.

In most places that provide parking options, there is often a motive to create a time-based ticket service for parking the cars. The price to be paid is proportional to the time the car is parked there. However there need to be a minimum amount set as well to account for people who park for very small periods of time. To avoid errors and logistic issues, we have designed an automated model that checks for spots and prints the ticket on its own in an efficient manner.

Our project aims to provide an easy and efficient means for the calculation of parking ticket price at busy centres such as malls, airports etc. by implementing a digital systems model. This ensures that the process is done smoothly without wasting time and reduces the scope of complications due to human error.

2. Introduction

The automated parking system will first check the availability of parking spots and then take parking time as input parameter. The model will automate a function that produces the amount that the customer must pay for parking the vehicle in the parking spot for that time period. Before a car enters the parking area, we first check if there are spots vacant for the car to park. If even one (or more) spots exists vacant from the output of the detector, we let the car go in, else it is blocked. Once the car is in the slot the time is being monitored and before the car leaves, the time it has stayed there is fed as input into the ticket calculator model. The model then automatically performs the required operations based on the programmed metrics and outputs to the customer the amount he has to pay.

Principle

To detect an empty parking spot, we use a Multi bit input representing each spot, where 0 indicates that a parking spot is empty and 1 if it is occupied. Ticket price is decided by a metric that can be set by the user. In our project, the first hour or less in which the vehicle is parked will incur a fixed rate of 'p' Rs. and every subsequent minute a rate of 'q' Rs. The time and ticket decimal inputs are converted to binary and used to calculate the fare. Final ticket price is converted back to its decimal equivalent and provided as output to the customer. For the sake of this model and the technical constraints we limit to small values but can be scaled up to any range required.

Approach and Methodology

Our model has two distinct parts – the detector and the calculator. The detector inputs a 1 if at least one slot is empty and a car can park. We did this by marking each vacant spot by a logic 1, and filled by logic 0. Using a NOT gate for each input and a multibit AND (or a multibit NAND, whichever is available), this output can be achieved as required.

For the calculator the time input in decimal is taken in a digit-by-digit format and converted into BCD using digital circuits. Subsequent multiplication and addition of the BCD digits will give us the binary equivalent. Similar setting is done for the 'p' and q' values. This binary equivalent is compared to 0b111100 (60) using a comparator which bifurcates into 2 separate circuits. If the value is less than 60, we only have to output the 'p' value, but if it is more , then we perform subtraction with 0b111100 and the resultant undergoes binary multiplication with 'q' and cost 'p' is added back. We make use of a divider and a separately designed converter circuit to get a decimal output back.

Components Required

1. Basic logic gates (AND, OR, NOT, etc)
2. 8 bit Multiplier
3. 8 bit full adders
4. 8 bit full subtractors
5. Hex digit display

Advantages

6. Automated parking system requires minimal human intervention and can perform the tasks of detection and calculation all by itself.

7. Reduces chances of human error in manually calculating the cost.

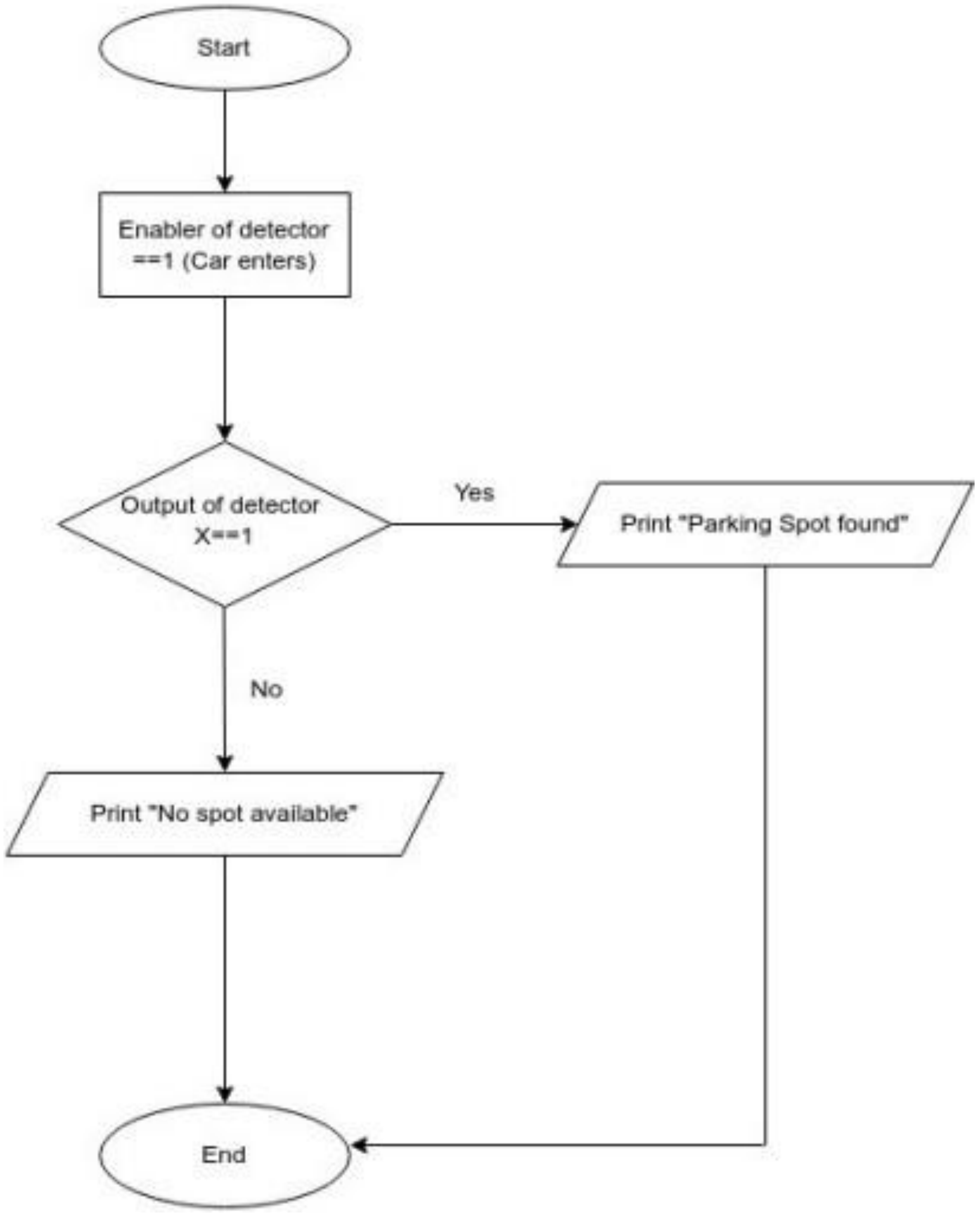
8. Cost effective and very less management required.

9. Highly efficient system of managing busy parking lots when the number of cars that go in and out with the time parked is difficult to track.

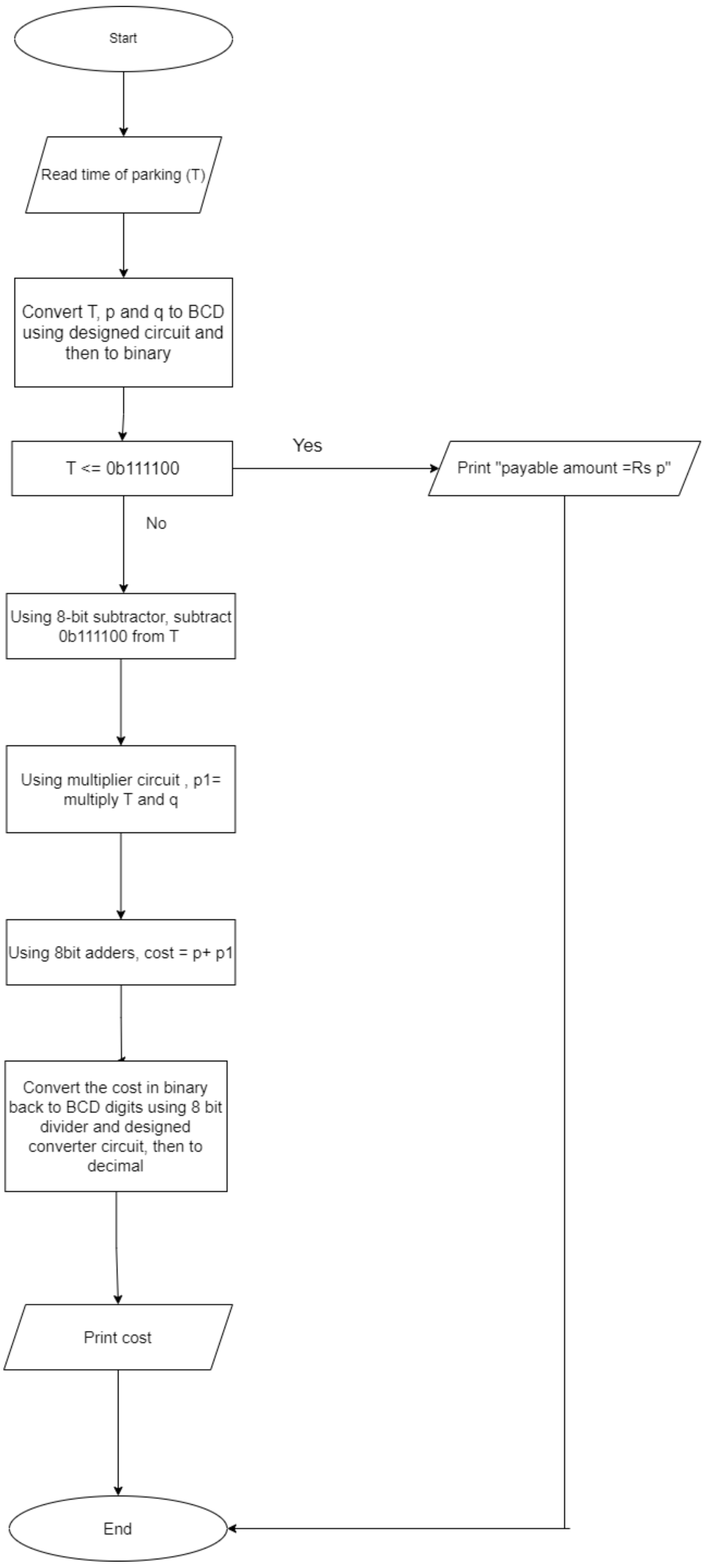
10. The pricing metric is not fixed and can be set by the owner of the parking lot, thus enabling flexibility in choosing the prices. For example, the parking rates of an airport may be much more than that of a mall.

3. Design

3.1 Flowcharts

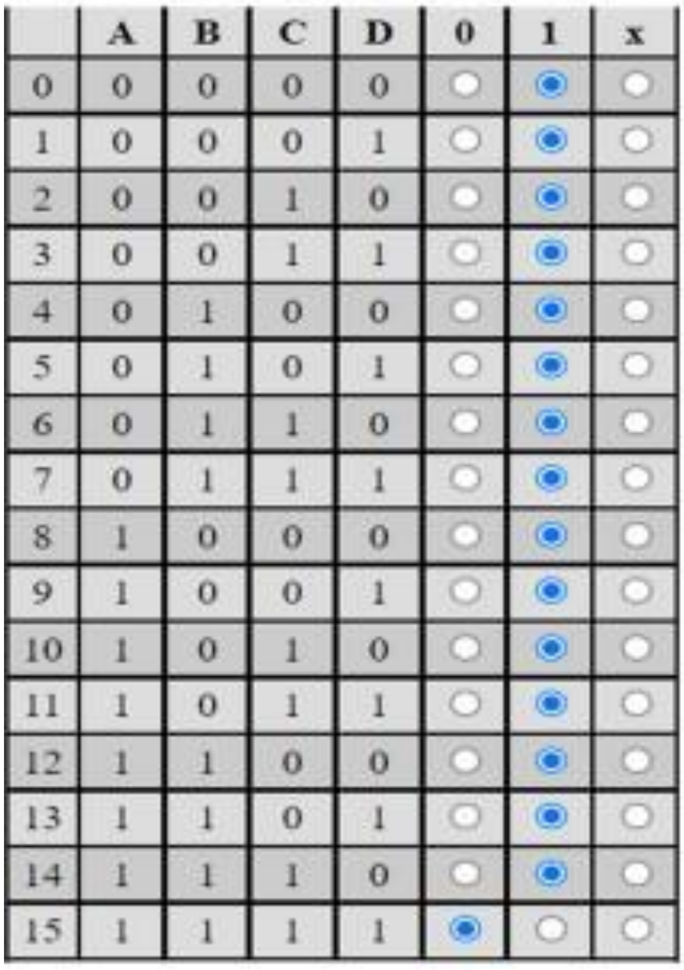

Start
Enabler of detector ==1 (Car enters)
Output of detector X==1
Yes
Print "Parking Spot found"
No
Print "No spot available"
End

l

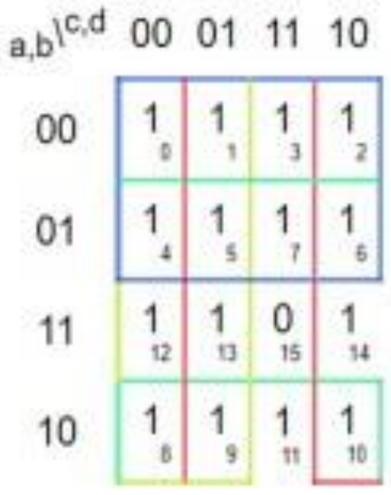

Start
Read time of parking (T)
Convert T, p and q to BCD using designed circuit and then to binary
T <= 0b111100
Yes
Print "payable amount =Rs p"
No
Using 8-bit subtractor, subtract 0b111100 from T
Using multiplier circuit , p1= multiply T and q
Using 8bit adders, cost = p+ p1
Convert the cost in binary back to BCD digits using 8 bit divider and designed converter circuit, then to decimal
Print cost
End

3.2 Truth tables and 3.3 Simplifications

Detector – KMap and Truth table



	A	B	C	D	0	1	x
0	0	0	0	0	○	●	○
1	0	0	0	1	○	●	○
2	0	0	1	0	○	●	○
3	0	0	1	1	○	●	○
4	0	1	0	0	○	●	○
5	0	1	0	1	○	●	○
6	0	1	1	0	○	●	○
7	0	1	1	1	○	●	○
8	1	0	0	0	○	●	○
9	1	0	0	1	○	●	○
10	1	0	1	0	○	●	○
11	1	0	1	1	○	●	○
12	1	1	0	0	○	●	○
13	1	1	0	1	○	●	○
14	1	1	1	0	○	●	○
15	1	1	1	1	●	○	○



a,b\c,d	00	01	11	10
00	1 (0)	1 (1)	1 (3)	1 (2)
01	1 (4)	1 (5)	1 (7)	1 (6)
11	1 (12)	1 (13)	0 (15)	1 (14)
10	1 (8)	1 (9)	1 (11)	1 (10)

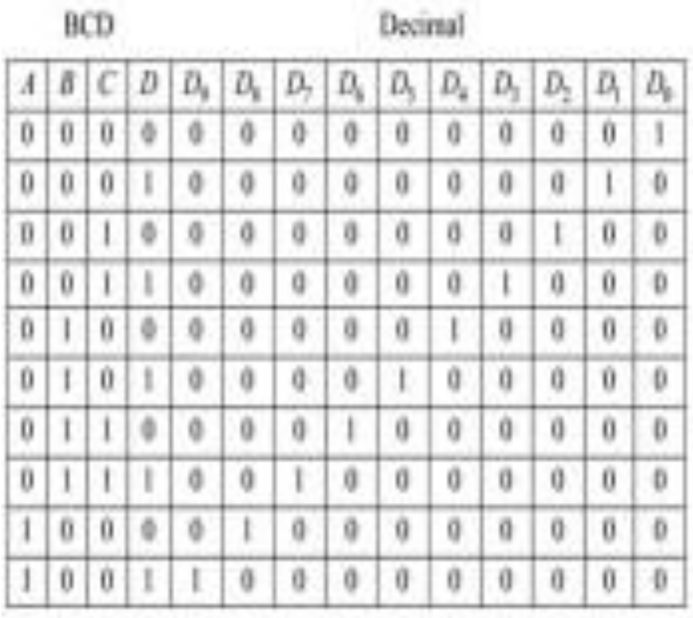
Final Expression from Kmap

For the case of 4 slots in the parking lot X = A'+B'+C'+D'

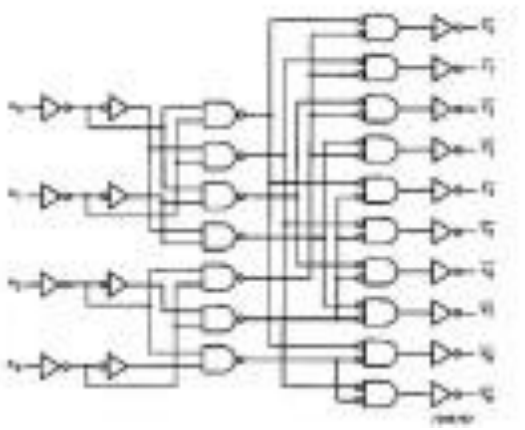
To generalise to n slots labelled A1,A2,A3.....An, we have

X = $\sum_{k=0}^{n} A_k$

BCD to Decimal decoder

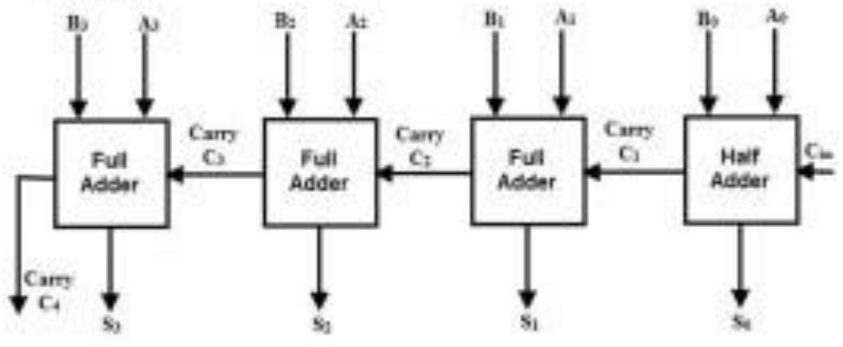


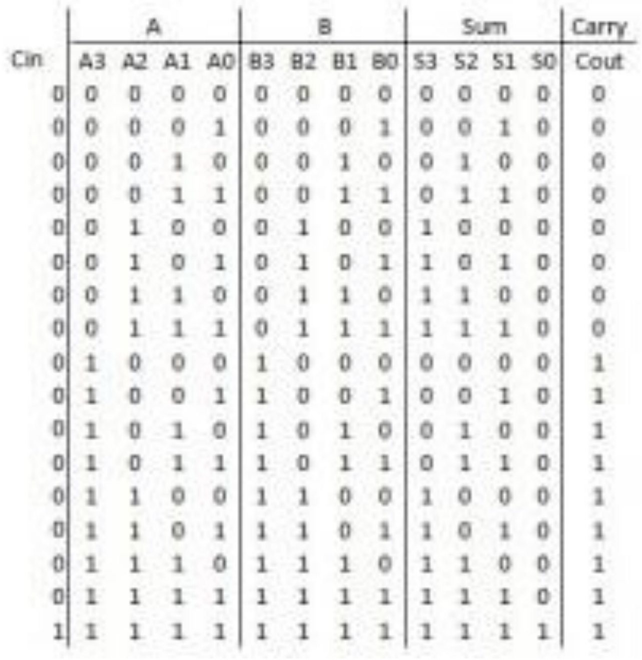
BCD				Decimal									
A	B	C	D	D_9	D_8	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
0	0	0	0	0	0	0	0	0	0	0	0	0	1
0	0	0	1	0	0	0	0	0	0	0	0	1	0
0	0	1	0	0	0	0	0	0	0	0	1	0	0
0	0	1	1	0	0	0	0	0	0	1	0	0	0
0	1	0	0	0	0	0	0	0	1	0	0	0	0
0	1	0	1	0	0	0	0	1	0	0	0	0	0
0	1	1	0	0	0	0	1	0	0	0	0	0	0
0	1	1	1	0	0	1	0	0	0	0	0	0	0
1	0	0	0	0	1	0	0	0	0	0	0	0	0
1	0	0	1	1	0	0	0	0	0	0	0	0	0



Adder

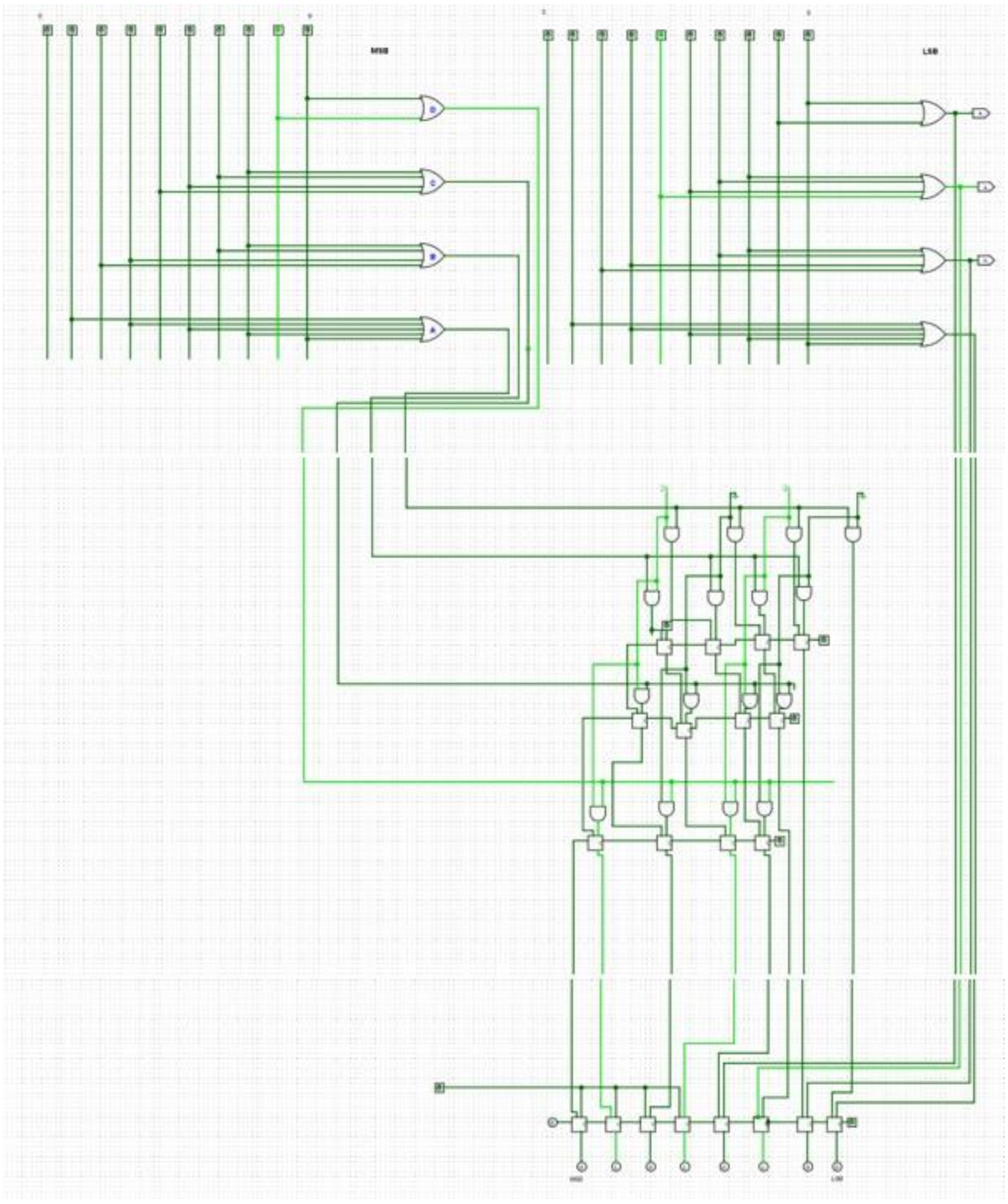
4 bit Adder

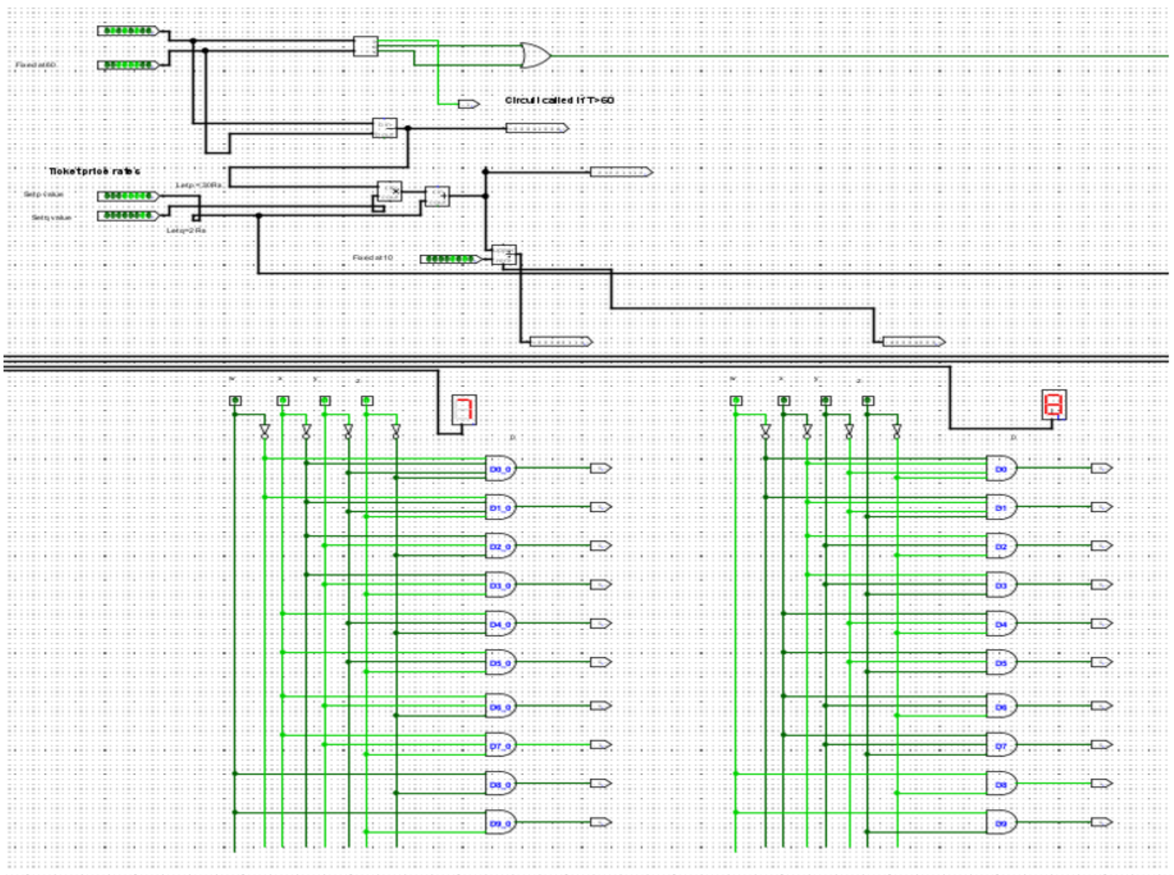


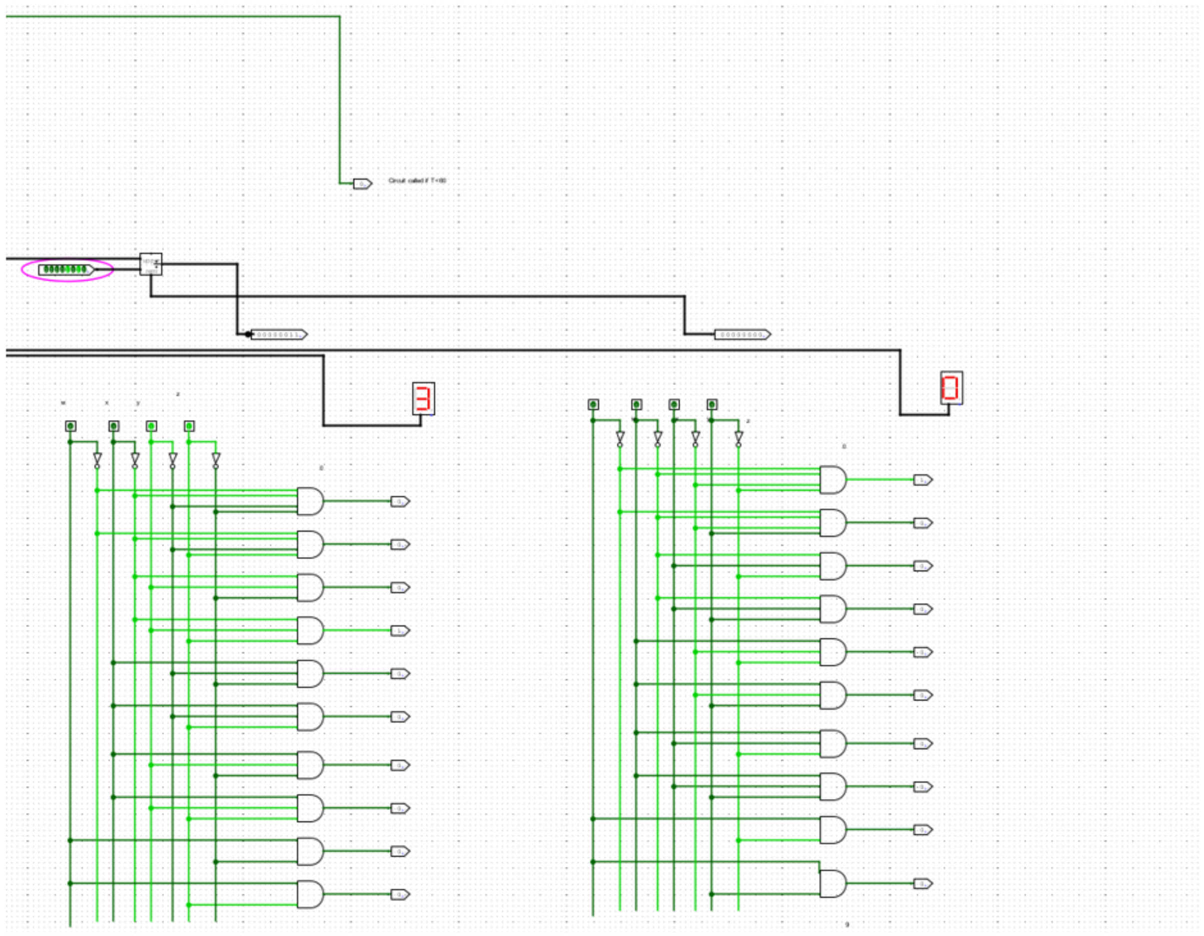


	A				B				Sum				Carry
Cin	A3	A2	A1	A0	B3	B2	B1	B0	S3	S2	S1	S0	Cout
0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	1	0	0	0	1	0	0	1	0	0
0	0	0	1	0	0	0	1	0	0	1	0	0	0
0	0	0	1	1	0	0	1	1	0	1	1	0	0
0	0	1	0	0	0	1	0	0	1	0	0	0	0
0	0	1	0	1	0	1	0	1	1	0	1	0	0
0	0	1	1	0	0	1	1	0	1	1	0	0	0
0	0	1	1	1	0	1	1	1	1	1	1	0	0
0	1	0	0	0	1	0	0	0	0	0	0	0	1
0	1	0	0	1	1	0	0	1	0	0	1	0	1
0	1	0	1	0	1	0	1	0	0	1	0	0	1
0	1	0	1	1	1	0	1	1	0	1	1	0	1
0	1	1	0	0	1	1	0	0	1	0	0	0	1
0	1	1	0	1	1	1	0	1	1	0	1	0	1
0	1	1	1	0	1	1	1	0	1	1	0	0	1
0	1	1	1	1	1	1	1	1	1	1	1	0	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1

3.4 Logisim




Fixed at 60
Circuit called if T>60
Ticket price rate's
Set p value
Set q value
Let p<30Rs
Let q=2 Rs
Fixed at 10
w
x
y
z
D0_0
D1_0
D2_0
D3_0
D4_0
D5_0
D6_0
D7_0
D8_0
D9_0
D0
D1
D2
D3
D4
D5
D6
D7
D8
D9


Circuit called if T<60
w
x
y
z

4. Verilog Code

4.1 Verilog Code for detector - Dataflow

```
module detector (
        input A,
        input B,
        input C,
        input D,

        output Z
);

assign Z = ~A | ~B | ~C | ~D;
endmodule

module main;
reg A;
reg B;
reg C;
reg D;
wire Z;
boolen uut(
.A(A),
.B(B),
.C(C),
.D(D),
.Z(Z)
);
initial begin
$monitor("t= %d | input =%d%d%d%d | output =%d", $time,A,B,C,D,Z);
$dumpfile("detector.vcd");
$dumpvars(0, main);

initial begin
     A=0; B=0; C=0; D=0;
  #1 A=0; B=0; C=0; D=1;
  #1 A=0; B=0; C=1; D=0;
  #1 A=0; B=0; C=1; D=1;
  #1 A=0; B=1; C=0; D=0;
  #1 A=0; B=1; C=0; D=1;
  #1 A=0; B=1; C=1; D=0;
  #1 A=0; B=1; C=1; D=1;
  #1 A=1; B=0; C=0; D=0;
  #1 A=1; B=0; C=0; D=1;
  #1 A=1; B=0; C=1; D=0;
  #1 A=1; B=0; C=1; D=1;
  #1 A=1; B=1; C=0; D=0;
  #1 A=1; B=1; C=0; D=1;
  #1 A=1; B=1; C=1; D=0;
  #1 A=1; B=1; C=1; D=1;
 end
endmodule
```

4.2 Verilog Code for detector - Gatelevel

```
module detector (
        input A,
        input B,
        input C,
        input D,

        output Z
        output oab, ocd
);

or(oab, ~A, ~B);
or(ocd, ~C, ~D);
or(Z, oab, ocd);
endmodule

module main;
reg A;
reg B;
reg C;
reg D;
wire Z;
boolen uut(
.A(A),
.B(B),
.C(C),
.D(D),
.Z(Z)
);
initial begin
$monitor("t= %d | input =%d%d%d%d | output =%d", $time,A,B,C,D,Z);
$dumpfile("detector.vcd");
$dumpvars(0, main);

initial begin
     A=0; B=0; C=0; D=0;
  #1 A=0; B=0; C=0; D=1;
  #1 A=0; B=0; C=1; D=0;
  #1 A=0; B=0; C=1; D=1;
  #1 A=0; B=1; C=0; D=0;
  #1 A=0; B=1; C=0; D=1;
  #1 A=0; B=1; C=1; D=0;
  #1 A=0; B=1; C=1; D=1;
  #1 A=1; B=0; C=0; D=0;
  #1 A=1; B=0; C=0; D=1;
  #1 A=1; B=0; C=1; D=0;
  #1 A=1; B=0; C=1; D=1;
  #1 A=1; B=1; C=0; D=0;
  #1 A=1; B=1; C=0; D=1;
  #1 A=1; B=1; C=1; D=0;
  #1 A=1; B=1; C=1; D=1;
 end
endmodule
```

4.3Verilog Code for Calculator - Behavioral

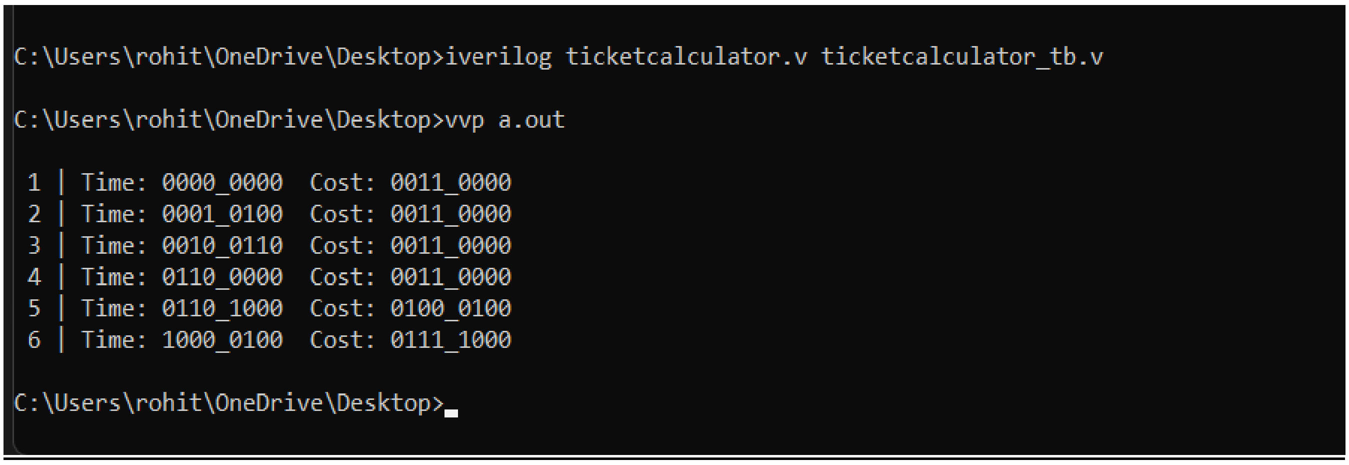
```
module time_converter(input  [3:0] time1, output reg [3:0] result);
  reg [3:0] time_bin;
  reg [3:0] time_sub_60;
  reg [3:0] time_mul_2;
  reg [3:0] time_add_30;

  // Conversion modules
  assign time_bin = decimal_to_binary(time1);
  assign time_sub_60 = binary_subtractor(time_bin, 8'b0011_1100);
  assign time_mul_2 = binary_multiplier(time_sub_60, 2'b10);
  assign time_add_30 = binary_adder(time_mul_2, 8'b0001_1110)
  assign result = binary_to_decimal(time_add_30);
endmodule

module decimal_to_binary(input  [3:0] decimal, output  reg[3:0] binary);
  always @* begin
    case (decimal)
      4'b0000: binary = 4'b0000;
      4'b0001: binary = 4'b0001;
      4'b0010: binary = 4'b0010;
      4'b0011: binary = 4'b0011;
      4'b0100: binary = 4'b0100;
      4'b0101: binary = 4'b0101;
      4'b0110: binary = 4'b0110;
      4'b0111: binary = 4'b0111;
      4'b1000: binary = 4'b1000;
      4'b1001: binary = 4'b1001;
      default: binary = 4'bxxxx; // default value for invalid input
    endcase
  end
endmodule

module binary_subtractor(input [3:0] a, input [3:0] b, output reg[3:0] result);
  reg [3:0] a_sub_b;
  reg carry;

  always @* begin
    a_sub_b = a - b;
    carry = 0;

    // Perform 1-bit binary subtraction with carry
    for (integer i = 0; i < 4; i = i + 1) begin
      if (a[i] == 1 && b[i] == 1) begin
        result[i] = carry;
        carry = 1;
      end else if (a[i] == 0 && b[i] == 0) begin
        result[i] = carry;
        carry = 0;
      end else if (a[i] == 1 && b[i] == 0) begin
        result[i] = a_sub_b[i];
        carry = 0;
      end else if (a[i] == 0 && b[i] == 1) begin
        result[i] = a_sub_b[i];
        carry = 1;
      end
    end
  end
endmodule

module binary_multiplier(input [3:0] a, input [1:0] b, output reg[3:0] result);
  reg [3:0] a_mul_b;
  reg [1:0] shift_count;

  always @* begin
    a_mul_b = 0;
    shift_count = 0;

    // Perform binary multiplication using shift and add
    while (shift_count < 2) begin
      if (b[shift_count] == 1) begin
        a_mul_b = a_mul_b + (a << shift_count);
      end
      shift_count = shift_count + 1;
    end
    result = a_mul_b;
  end
endmodule

module binary_adder(input [3:0] a, input [3:0] b, output reg[3:0] result);
  reg [3:0] a_add_b;
  reg carry;

  always @* begin
    a_add_b = a + b;
    carry = 0;

    // Perform 1-bit binary addition with carry
    for (integer i = 0; i < 4; i = i + 1) begin
      if (a[i] == 1 && b[i] == 1) begin
        result[i] = carry;
        carry = 1;
      end else if (a[i] == 0 && b[i] == 0) begin
        result[i] = carry;
        carry = 0;
      end else begin
        result[i] = a_add_b[i];
        carry = 0;
      end
    end
  end
endmodule

module binary_to_decimal(input [3:0] binary, output reg[3:0] decimal);
  always @* begin
    case (binary)
```

```
      4'b0000: decimal = 4'b0000;
      4'b0001: decimal = 4'b0001;
      4'b0010: decimal = 4'b0010;
      4'b0011: decimal = 4'b0011;
      4'b0100: decimal = 4'b0100;
      4'b0101: decimal = 4'b0101;
      4'b0110: decimal = 4'b0110;
      4'b0111: decimal = 4'b0111;
      4'b1000: decimal = 4'b1000;
      4'b1001: decimal = 4'b1001;
      default: decimal = 4'bxxxx; // default value for invalid input
    endcase
  end
endmodule
```

Calculator Testbench

```
module time_converter_tb;
  reg [3:0] time1;
  wire [3:0] result;

  // Instantiate the time_converter module
  time_converter tconv(
    time1,result
  );

  initial begin
    // Test input values
    time1 = 8'b0000_0000 ; //Time =0s
    #1;
    $display("%t | Time: %b    Cost: %b", $time, time1,result);

    time1 = 8'b0001_0100; // Time = 18s
    #1;
    $display("%t | Time: %b    Cost: %b", $time, time1,result);

    time1 = 8'b0010_0110; // Time = 26s
    #1
    $display("%t | Time: %b    Cost: %b", $time, time1,result);

    time1 = 8'b0110_0000; //Time = 60s
    #1;
    $display("%t | Time: %b    Cost: %b", $time, time1,result);

    time1 = 8'b0110_1000; //Time = 67s
    #1;
    $display("%t | Time: %b    Cost: %b", $time, time1,result);

    time1 = 8'b1000_0100; //Time = 84s
    #1;
    $display("%t | Time: %b    Cost: %b", $time, time1,result);


    $finish;
  end
endmodule
```

Output



```
C:\Users\rohit\OneDrive\Desktop>iverilog ticketcalculator.v ticketcalculator_tb.v

C:\Users\rohit\OneDrive\Desktop>vvp a.out

 1 | Time: 0000_0000  Cost: 0011_0000
 2 | Time: 0001_0100  Cost: 0011_0000
 3 | Time: 0010_0110  Cost: 0011_0000
 4 | Time: 0110_0000  Cost: 0011_0000
 5 | Time: 0110_1000  Cost: 0100_0100
 6 | Time: 1000_0100  Cost: 0111_1000

C:\Users\rohit\OneDrive\Desktop>
```

5. 1 Conclusions

This project successfully detects an empty parking spot in any area with a designated parking so that a user knows before-hand if parking is available. If it is available, it then calculates the parking ticket price which uses many concepts of digital electronics. Using this model, everything runs automatically so that there is minimal need for extra work. Chances of error are also minimal and the process is overall more efficient than non-automated systems.

This project is a mini-representation of a parking ticket calculator that can be used in real world. For the sake of demonstration of the model, we are limited to a small value of input, hence a small output is obtained. With proper resources such as time and logistics the scope of the model can be widened to fit real-life needs in different settings. Further the hourly and minutely rates in the model are not fixed and are free to be changed depending on need.

5.2 Future Work

Since the project is built on a highly scaled down version, it is limited to smaller inputs of time, cost and also in features. The detector despite being a very simple circuit is capable of higher order expansion such that its practical scope may be widened. For larger scale practical applications, the model may be designed with more logic elements and thus be made fit to include capabilities beyond discussed in our model. It can then be expanded to any range of price and time, thus being of immense use in practical settings where an orderly use of the parking system is needed.

6. References

i. Digital Logic And Computer Design By M. Morris Mano.

2.https://www.hel.fi/static/kanslia/Innovaatiorahasto/charles-thesis-formated-covered.pdf

3. https://www.researchgate.net/publication/304337388_Design_and_simulation_of_120_capacity_automobile_parking_control_system_using_updown_decade_counters

4.https://user.eng.umd.edu/ austin/ense623.d/projects06.d/ParkingLotProject.pdf

5. https://www.elprocus.com/electrical-project-ideas-for-engineering-students/