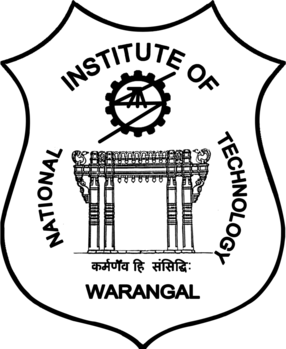
MINI PROJECT REPORT ON

RESTAURANT ONLINE FOOD ORDERING SYSTEM



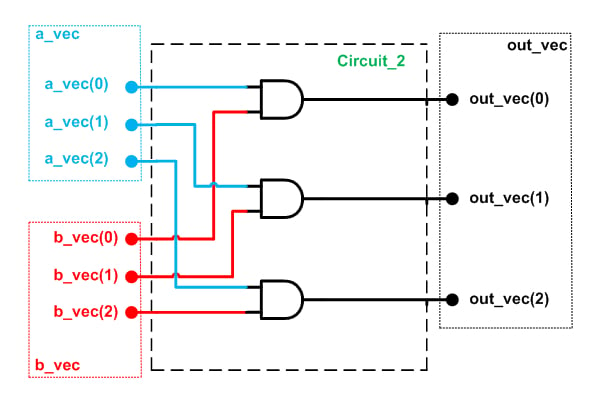
Submitted by:

* Pooja Gupta(21ECB0A44)
* Anjali Pravallika(21ECB0A45)
* P. Srinidhi(21ECB0A46)

Supervisors:

* Mrs. Prithvi Mam
* Mr. Narendar Sir

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING, NATIONAL INSTITUTE OF TECHNOLOGY, WARANGAL



What is Verilog?

Verilog is a HARDWARE DESCRIPTION LANGUAGE (HDL). It is a language used for describing a digital system like a network switch or a microprocessor or a memory or a flip−flop. It means, by using an HDL we can describe any digital hardware at any level. Designs, which are described in HDL are independent of technology, very easy for designing and debugging, and are normally more useful than schematics, particularly for large circuits.

Verilog supports a design at many levels of abstraction. The major three are-

* Behavioral level
* Register-transfer level
* Gate level

Behavioral level

This level describes a system by concurrent algorithms (Behavioural). Every algorithm is sequential, which means it consists of a set of instructions that are executed one by one. Functions, tasks, and blocks are the main elements. There is no regard for the structural realization of the design.

Gate Level

Within the logical level, the characteristics of a system are described by logical links and their timing properties. All signals are discrete signals. They can only have definite logical values (`0', `1', `X', `Z`). The usable operations are predefined logic primitives (basic gates). Gate-level modeling may not be the right idea for logic design. Gate-level code is generated using tools like synthesis tools and his netlist is used for gate-level simulation and for the backend. Except when they serve to separate tokens.

Comments

There are two forms to represent the comments

* 1) Single-line comments begin with the token // and end with a carriage return.

Ex.: //this is single-line syntax

* 2) Multiline comments begin with the token /\* and end with token \*/

Ex.: /\* This is multiline Syntax\*/

### Numbers

You can specify a number in binary, octal, decimal, or hexadecimal format. Negative numbers are represented in 2’s complement numbers. Verilog allows integers, real numbers, and signed & unsigned numbers.

The syntax is given by − <size> <radix> <value>

Size or unsized number can be defined in <Size> and <radix> defines whether it is binary, octal, hexadecimal, or decimal.

### Identifiers

The identifier is the name used to define the object, such as a function, module, or register. Identifiers should begin with alphabetical characters or underscore characters. Ex. A\_Z, a\_z,\_

Identifiers are a combination of alphabetic, numeric, underscore, and $ characters. They can be up to 1024 characters long.

### Operators

Operators are special characters used to put conditions or to operate the variables. There are one, two, and sometimes three characters used to perform operations on variables.

Ex. >, +, ~, &! =.

### Verilog Keywords

Words that have special meanings in Verilog are called Verilog keywords. For example, assign, case, while, wire, reg, and, or, nand, and module. They should not be used as identifiers. Verilog keywords also include compiler directives, and system tasks and functions.

## Gate Level Modelling

Verilog has built-in primitives like logic gates, transmission gates, and switches. These are rarely used for design work but they are used in the post-synthesis world for modeling ASIC/FPGA cells.

Gate-level modeling exhibits two properties −

**Drive strength** − The strength of the output gates is defined by drive strength. The output is strongest if there is a direct connection to the source. The strength decreases if the connection is via a conducting transistor and least when connected via a pull-up/down resistive. The drive strength is usually not specified, in which case the strengths default to strong1 and strong0.

**Delays** − If delays are not specified, then the gates do not have propagation delays; if two delays are specified, then the first one represents the rise delay and the second one is the fall delay; if only one delay is specified, then both, rise and fall are equal. Delays can be ignored in synthesis.

## Data Types

## Verilog consists of, mainly, four basic values. All Verilog data types, which are used in Verilog store these values −

0 (logic zero, or false condition)

1 (logic one, or true condition)

x (unknown logic value)

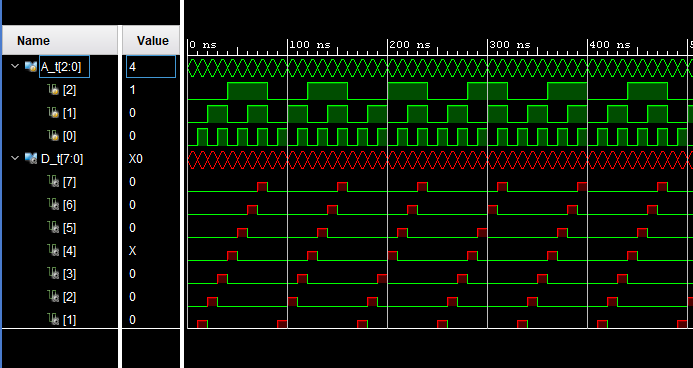
z (high impedance state)

use of x and z is very limited for synthesis.

### Wire: A wire is used to represent a physical wire in a circuit and it is used for the connection of gates or modules. The value of a wire can only be read and not assigned in a function or block. A wire cannot store value but is always driven by a continuous assignment statement or by connecting the wire to the output of a gate/module.

### Register: A reg (register) is a data object, which is holding the value from one procedural assignment to the next one and is used only in different functions and procedural blocks. Reg is a simple Verilog, variable-type register and can’t imply a physical register. In multi-bit registers, the data is stored in the form of unsigned numbers and sign extension is not used.

### Input, Output, Inout: These keywords are used to declare the input, output, and bidirectional ports of a task or module. Here input and inout ports are of wire type and the output port is configured to be of wire, reg, wand, wor, or tri type. Always, the default is wire type.



ABSTRACT:



Nowadays, people are more regular to dine-in at restaurants for their meals. The online food ordering system provides convenience for customers that are nothing special but the generally busy people of society. RESTAURANT FOOD ORDERING overcomes the demerits of the manual hotel or mess system and the old-fashioned queuing system. This system enhances access to readymade foods for people.

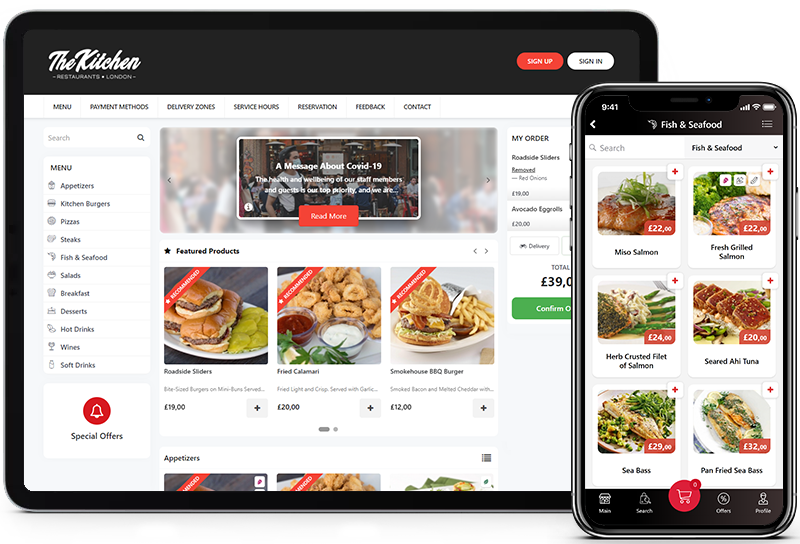
Therefore, this system improves the speed of getting food to a person’s plate and the quality and manner of taking the order from the customer. It provides a better communication platform. The online food ordering system provides the menu online and the customers can easily place the order by selecting.

Also, with RESTAURANT FOOD ORDERING, people can easily track their orders, and the admin can maintain the customer’s database and advance the food delivery system. This food ordering system allows the user to select the desired food items from a list of available menu items provided by the local hotel or restaurant. The user can place orders for the food items of their liking from the list. The payment can be made online or pay-on-delivery system. The user’s details are maintained confidential because it maintains a separate account for each user. An id and password are provided for each user. Therefore it provides a more secure and safe ordering system.

THEORY:

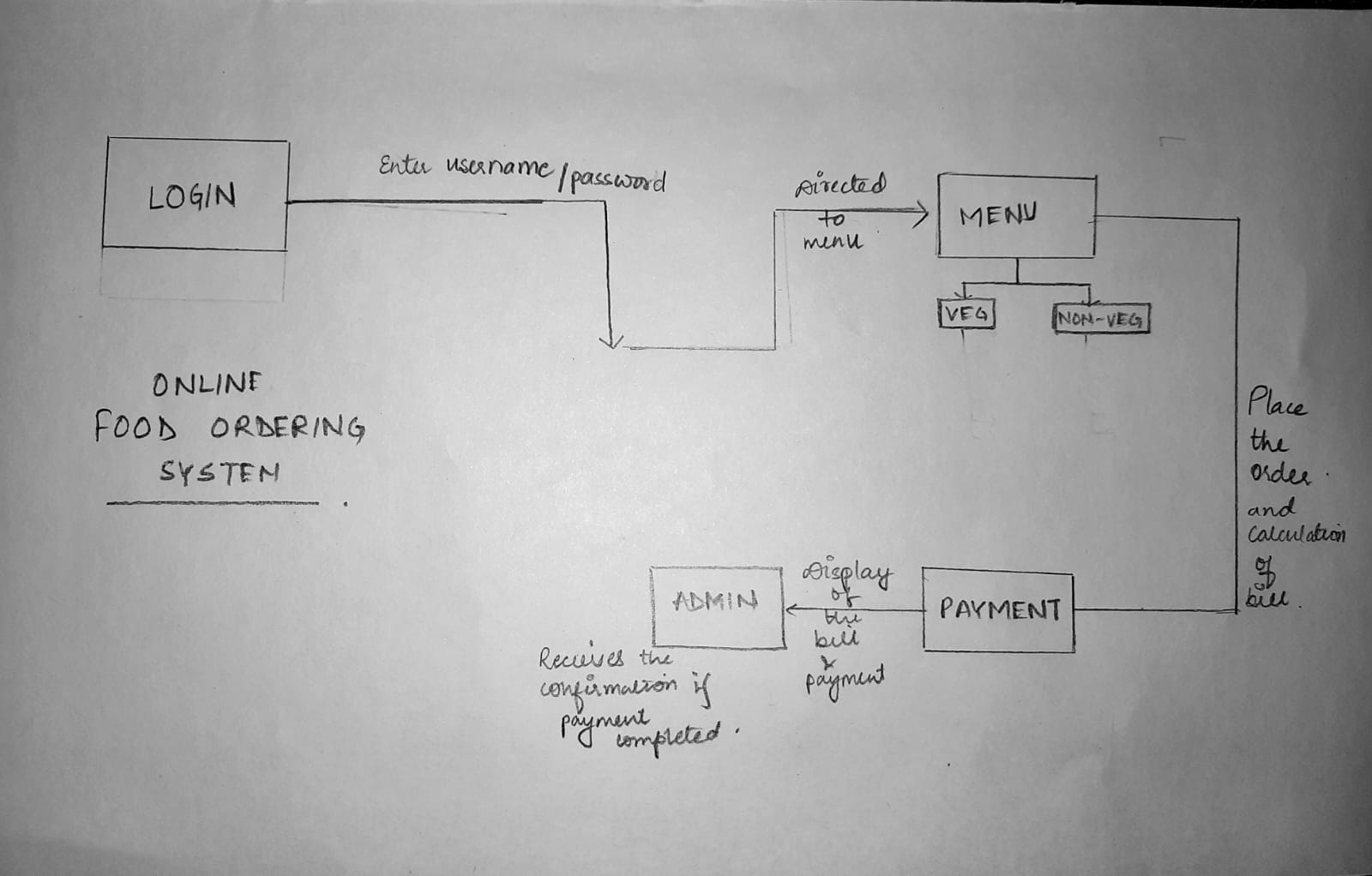
INTRODUCTION:

In every field of human life, automation has become more critical. But there are also many areas where more common methods are being used. The restaurant ordering system is one such area. This project aims to create a touch-based food ordering method as well as a secure payment process that can be used to change the conventional ordering process. In general, the food ordering procedure is actually given in the menu card format in a restaurant, then the customer will have to choose the food item, so the server would have to come back and take their orders, which is really a long method of processing.This project recommends a completely integrated ordering system, in which a more user-friendly touch-screen menu card replaces the conventional paper-based menu ordering process.



How about you enter a restaurant and order the food just in a few minutes without waiting for the waiter to come and take your order? Isn’t it cool?

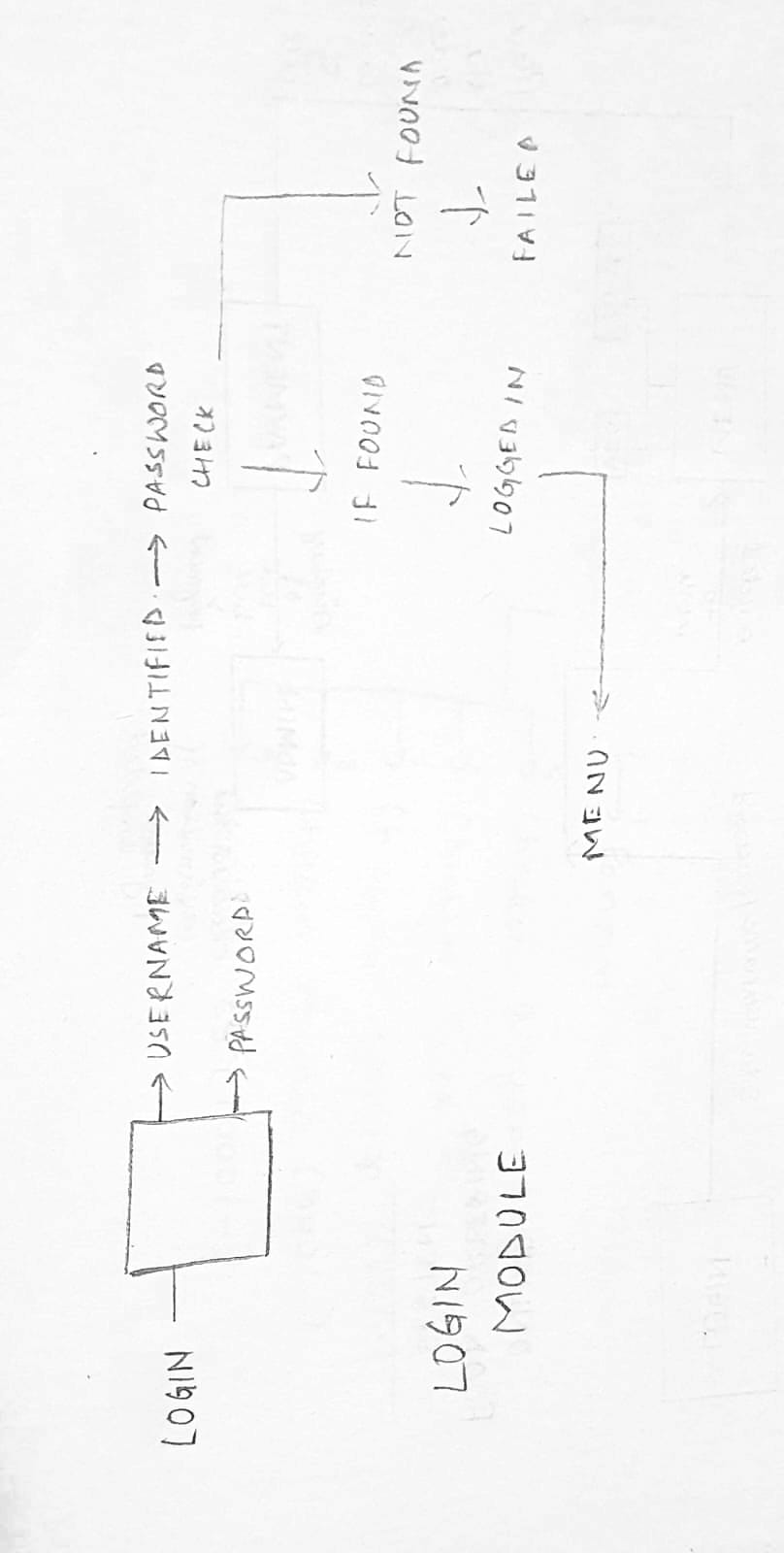
This project is made using Verilog HDL using Xilinx Vivado 2014.2. It consists of three individual modules namely Login\_page, Menu, and Payment. Our main module is Menu which has different options to select their favorite dishes it can be single or multiple. To make it more useful I made it in such a way they can order the same item more than once too.

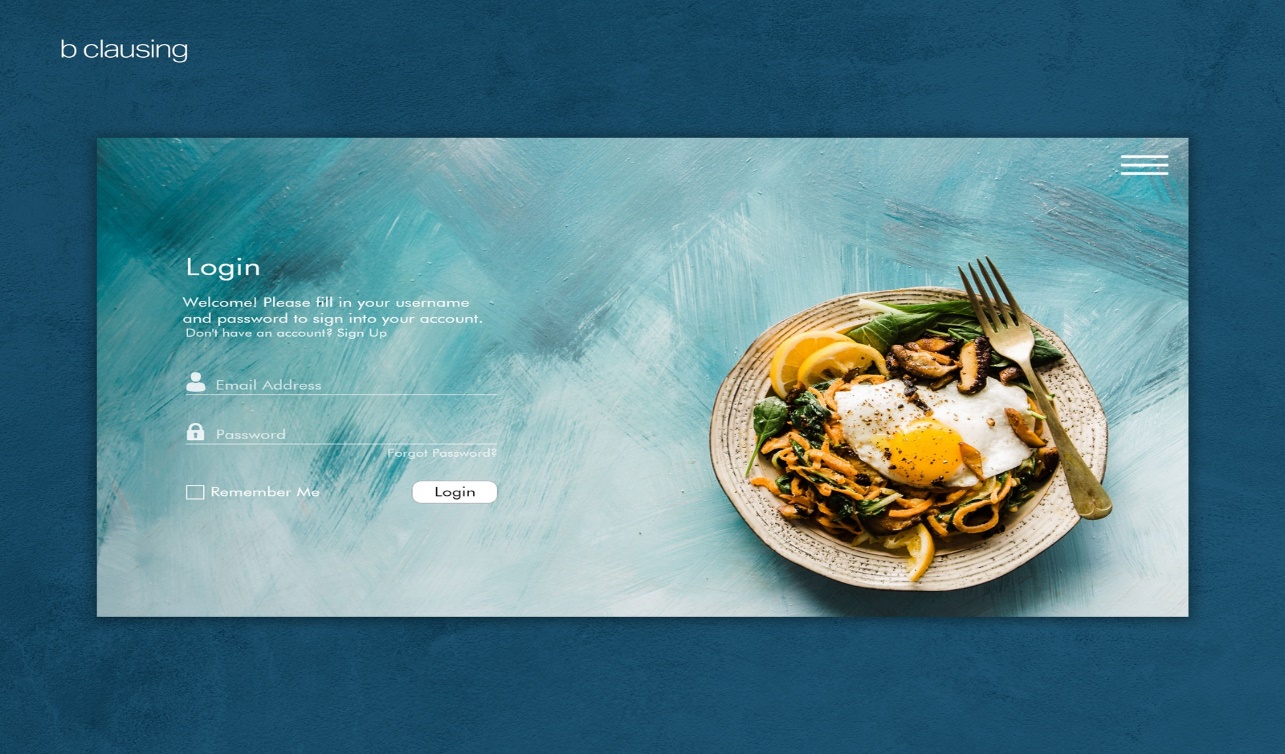


MODULES :

LOGIN\_PAGE:

As from above, we started building the login page first, Every user has a different login id and password. Every customer will have an account to which they can log in or if a new customer comes they can create an account and then place an order. We created a database for the customers where they have login\_id or username and also a password. To make it safer and much easier to remember for the customer, they can give their cell phone number as login\_id or username.





LOGIN CODE:

`timescale 1ns / 1ps

// Create Date: 21.11.2022 11:41:59

`define true 1`b1//defining true

`define false 1`b0//defining false

`define FIND 1`b0//finding if the username is there in database

`define AUTHENTICATE 1`b1//verifying the passwords

`define WAITING 2`b00//wait until verifyication is done

`define MENU 2`b01//menu option

`define BILL 2`b10//bill option

`define EXIT 2`b11//exit option

module login(//login main code

input [9:0] username,

input [4:0] password,//password input

input action,

input index;//

output success,

);

reg[9:0] user\_data;

reg[4:0] useusername ;

initial begin

user\_data[0]=10`d8914; user\_pin[0]=5`d94595;

user\_data[1]=10`d9672; user\_pin[1]=5`d92906;

user\_data[2]=10`d2472; user\_pin[2]=5`d30122;

end

integer i;

always @ (username or password)

begin

success = `false;

index = 0;

for(i=0;i<3;i=i+1)

begin

if(username == user\_data[i])

begin

if(action == FIND)

begin

success=`true;

index=i;

end

if(action==AUNTHENTICATE)

begin

if(password == user\_pin[i])

begin

success=`true;

index=i;

end

end

end

endmodule

TEST BENCH:

`timescale 1ns / 1ps

// Create Date: 21.11.2022 11:41:55

`define true 1'b1

`define false 1'b0

`define FIND 1'b0

`define AUTHENTICATE 1'b1

`define WAITING 2'b00

`define MENU 2'b01

`define BILL 2'b10

`define EXIT 2'b11

`

module login\_tb();

reg [9:0] username;

reg [4:0] password;

reg action;

reg index;

wire success;

login uut (username, password, action, index, sucess);

initial begin

username =10'b2376; password= 5'b66578; action = FIND; index = 1'b0; #10;

username =10'b2472; password=5'b36894; action = FIND; index = 1'b0; #10;

username =10'b4355; password=5'b67891; action = FIND; index = 1'b0; #10;

username =10'b3989; password=5'b56786; action = FIND; index = 1'b0; #10;

username =10'b2276; password= 5'b66378; action = FIND; index = 1'b0; #10;

username =10'b2872; password=5'b36094; action = FIND; index = 1'b0; #10;

username =10'b4155; password=5'b61891; action = FIND; index = 1'b0; #10;

username =10'b3980; password=5'b56086; action = FIND; index = 1'b0; #10;

username =10'b2871; password=5'b30094; action = FIND; index = 1'b0; #10;

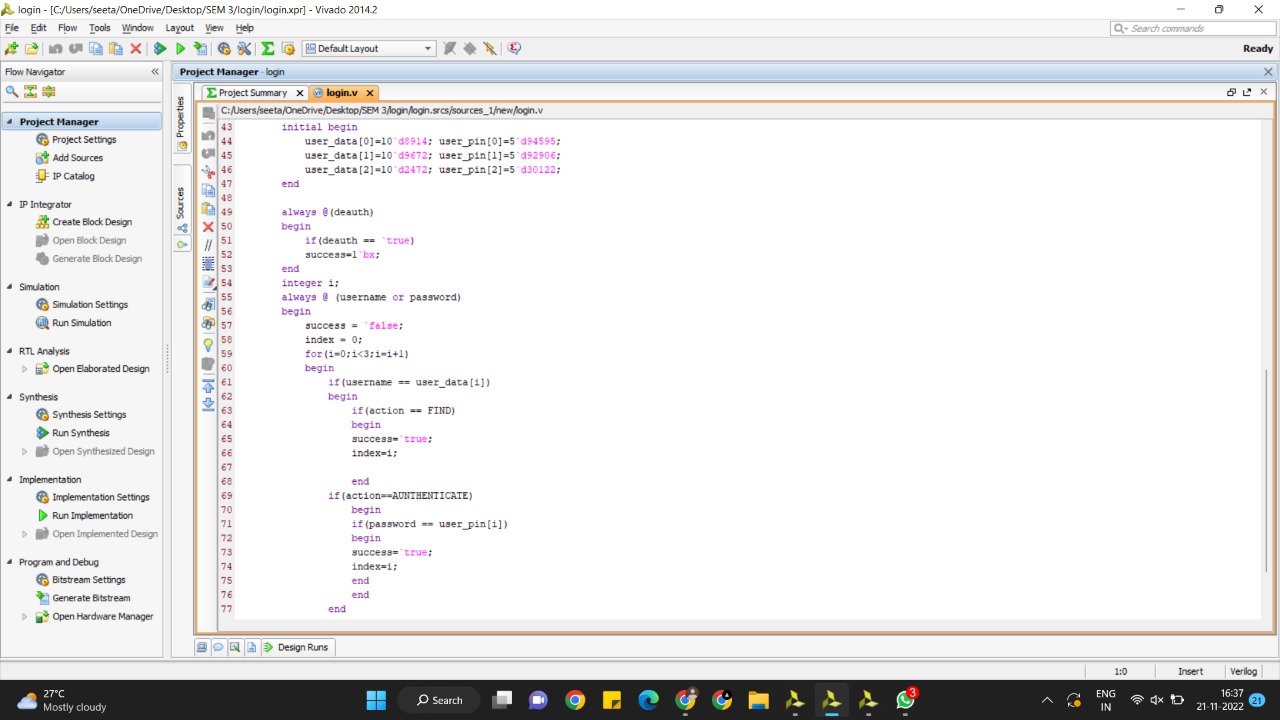
username =10'b4152; password=5'b62891; action = FIND; index = 1'b0; #10;

username =10'b3983; password=5'b59086; action = FIND; index = 1'b0; #10;

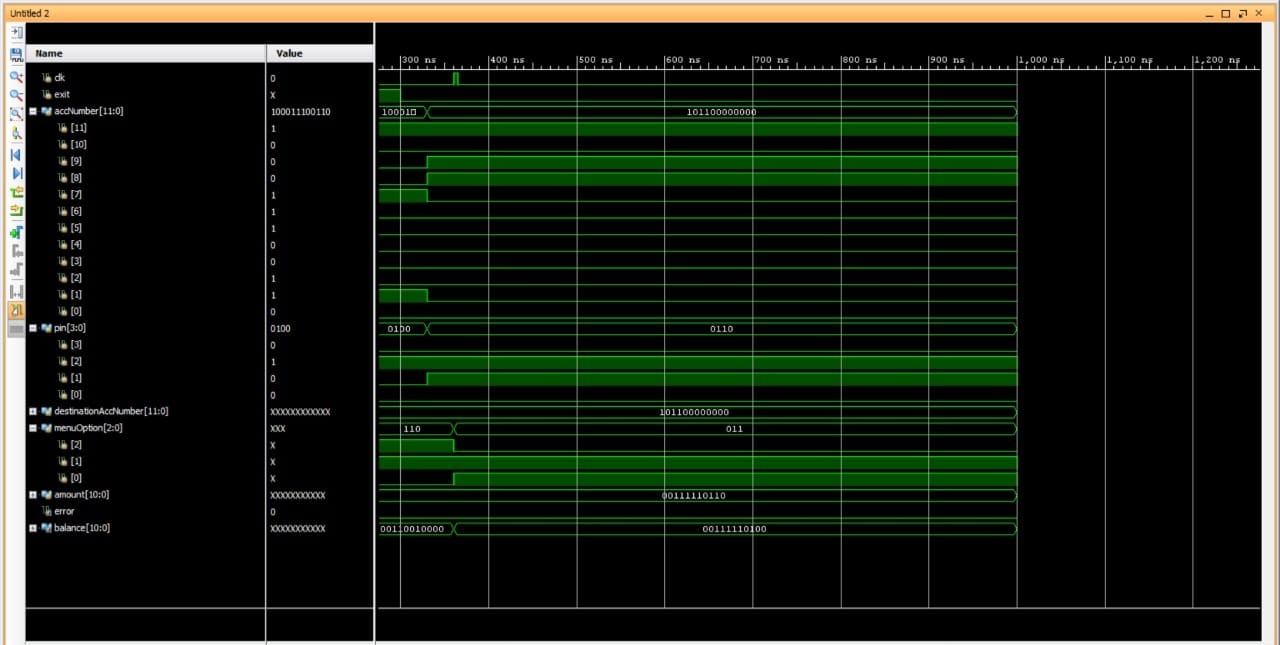
end

endmodule

OUR CODE:



SIMULATION:



MENU:

The menu module consists of the key features of this app, here we have provided facilities to the customers such that they can select the options or items they wanted. They can select single or multiple dishes they require. There are especially two divisions in dishes which are veg and non-veg. In the veg or non-veg, there are five divisions or types of dishes.

Menu – VEG and NON-VEG

VEG – 1. Paneer Fried Rice, denoted as item-0 and costs 100/-

2. Butter Naan, denoted as item-1 and costs 200/-

3. Special Fried Rice, denoted as item-2 and costs 300/-

4. Veg Biryani, denoted as item-3 and costs 400/-

5. Panner Butter Masala, denoted as item-4 and costs 500/-

NON-VEG – 1. Chicken Biryani, denoted as item-5 and costs 600/-

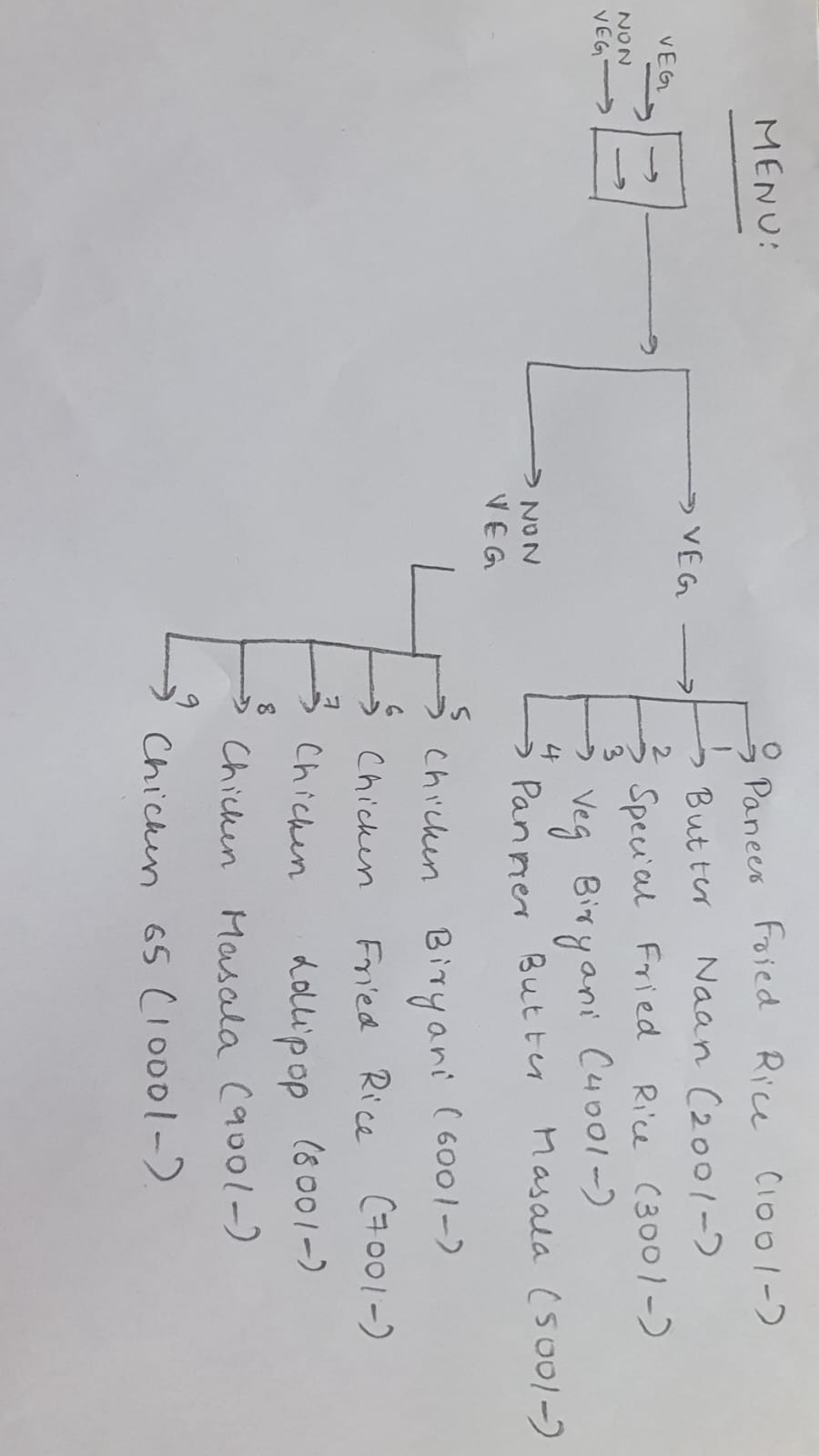
2. Chicken Fried Rice, denoted as item-6 and costs 700/-

3. Chicken Lollipop, denoted as item-7 and costs 800/-

4. Chicken Masala, denoted as item-8 and costs 900/-

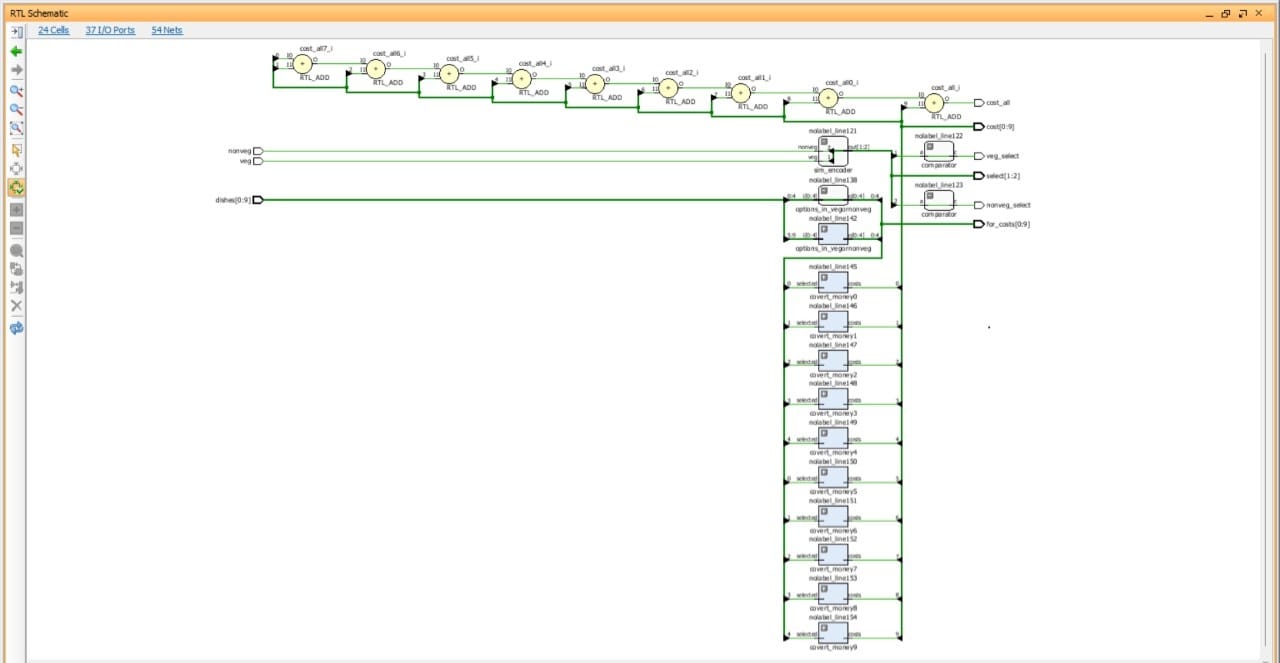
5. Chicken 65, denoted as item-9 and costs 1000/-

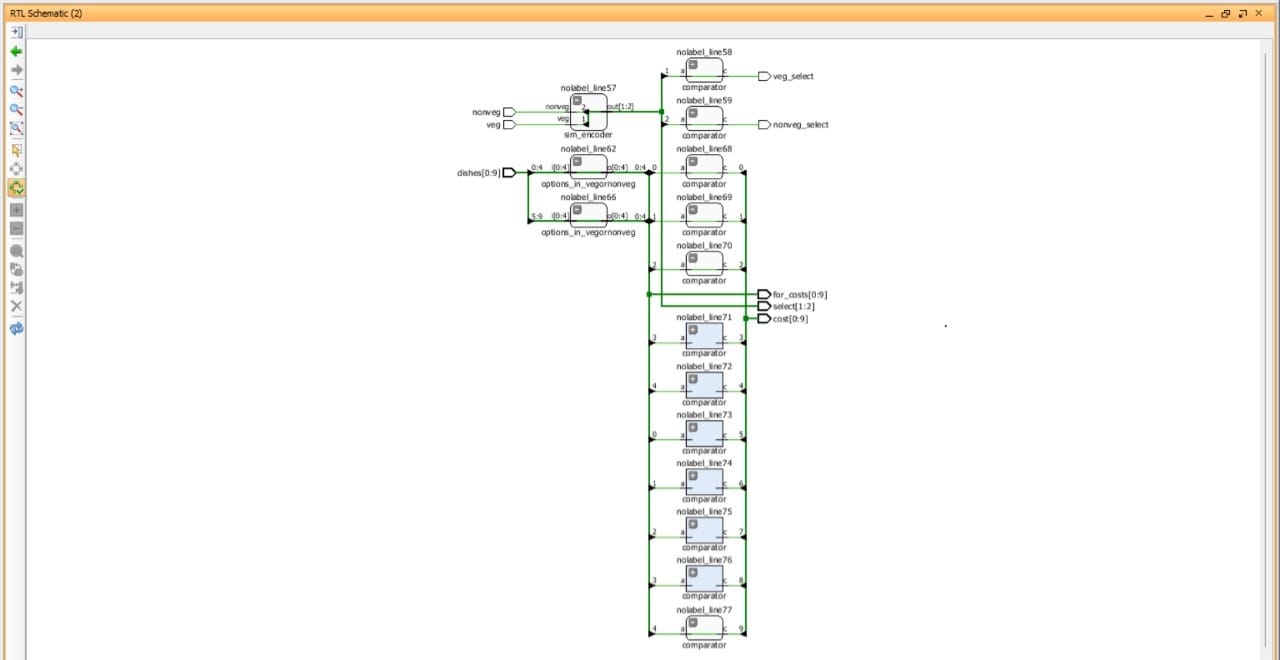
In the menu, the person is free to choose the required item, after the selection the item selected will be displayed to the admin who takes the orders that this person selected. The administration makes sure that if a certain item is not available then the option will be removed.



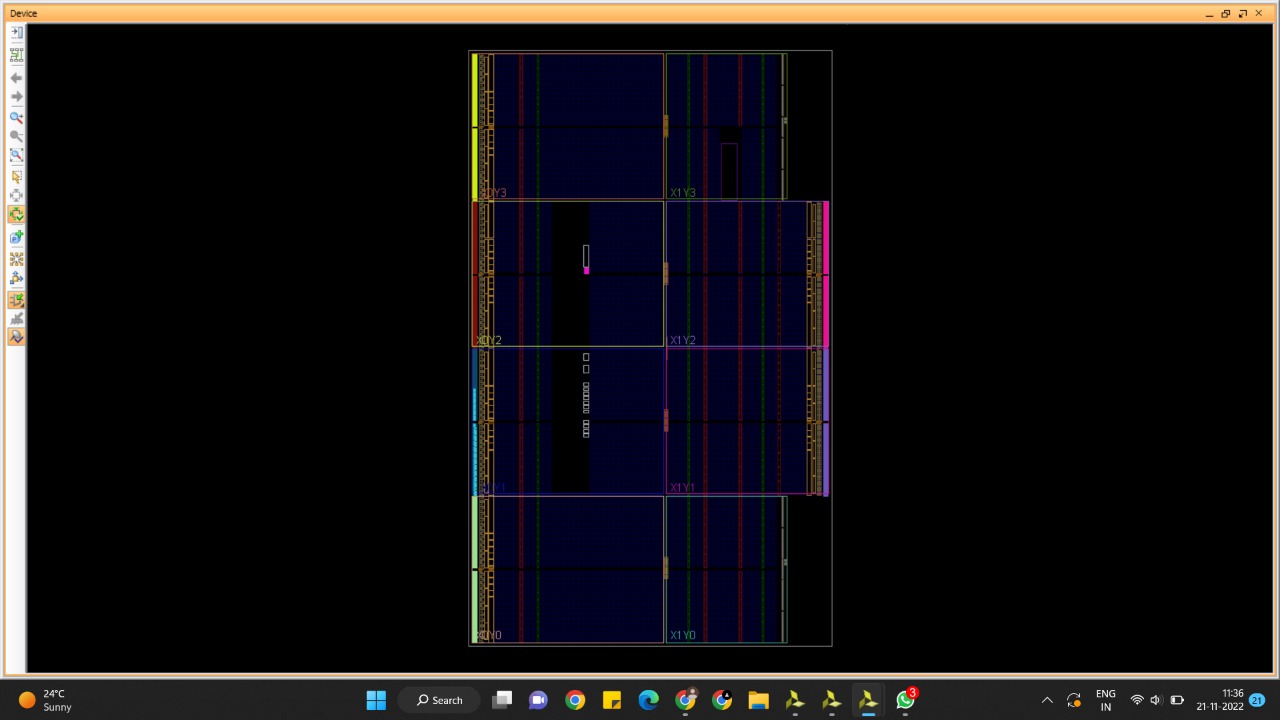


SCHEMATIC:





NETLIST:



MENU CODE :

`timescale 1ns / 1ps

// Create Date: 21.11.2022 11:41:59

module sim\_encoder(

input veg,

input nonveg,

output [1:2]out);

//lets keep i[2] == 0, i[3] == 0 and also i[0] is veg, i[1] is non-veg

assign out[1] = veg;

assign out[2] = nonveg;

endmodule //this module is for getting which option did the customer select

module covert\_money0(

input selected,

output costs);

assign costs0 = 4'd0100 \* (selected == 1);

endmodule//this module is for the item cost

module covert\_money1(

input selected,

output costs);

assign costs1 = 4'd0200 \* (selected == 1);

endmodule//this module is for the item cost

module covert\_money2(

input selected,

output costs);

assign costs2 = 4'd0300 \* (selected == 1);

endmodule//this module is for the item cost

module covert\_money3(

input selected,

output costs);

assign costs3 = 4'd0400 \* (selected == 1);

endmodule//this module is for the item cost

module covert\_money4(

input selected,

output costs);

assign costs4 = 4'd0500 \* (selected == 1);

endmodule//this module is for the item cost

module covert\_money5(

input selected,

output costs);

assign costs5 = 4'd0600 \* (selected == 1);

endmodule//this module is for the item cost

module covert\_money6(

input selected,

output costs);

assign costs6 = 4'd0700 \* (selected == 1);

endmodule//this module is for the item cost

module covert\_money7(

input selected,

output costs);

assign costs7 = 4'd0800 \* (selected == 1);

endmodule//this module is for the item cost

module covert\_money8(

input selected,

output costs);

assign costs8 = 4'd0900 \* (selected == 1);

endmodule//this module is for the item cost

module covert\_money9(

input selected,

output costs);

assign costs9 = 4'd1000 \* (selected == 1);

endmodule//this module is for the item cost

module comparator(

input a,

output c);

assign c =(a==1);

endmodule//this module is for the option he selected

module options\_in\_vegornonveg(

input [0:4]i,

output [0:4]o);

assign o[0] = i[0];//cost 100

assign o[1] = i[1];//cost 200

assign o[2] = i[2];//cost 300

assign o[3] = i[3];//cost 400

assign o[4] = i[4];//cost 500

endmodule//this module is for the option he is interested to select

module menu( //this is the main module

input veg,

input nonveg,

input [0:9]dishes,

output [1:2]select,

output veg\_select,

output nonveg\_select,

output [0:9]for\_costs,

output [0:9]cost

//reg cost\_all

);

reg cost\_all;

initial begin

//reg cost\_all;

sim\_encoder g1 (veg, nonveg, select[1:2]); //the person selected options

comparator g2 (select[1], veg\_select); //selcted option verification

comparator g3 (select[2], nonveg\_select); //selcted option verification

/\* assign for\_costs[0] = 4'd0100;

assign for\_costs[1] = 4'd0200;

assign dishes[2] = 4'd0300;

assign dishes[3] = 4'd0400;

assign dishes[4] = 4'd0500;

assign dishes[5] = 4'd0600;

assign dishes[6] = 4'd0700;

assign dishes[7] = 4'd0800;

assign dishes[8] = 4'd0900;

assign dishes[9] = 4'd1000;

\*/

// if (veg\_select==1) begin

options\_in\_vegornonveg g5 ( dishes[0:4], for\_costs[0:4]); //options in the veg

//end

//if (nonveg\_select==1) begin

options\_in\_vegornonveg g6 ( dishes[5:9], for\_costs[5:9]); //options in nonveg

//end

covert\_money0 g7 (for\_costs[0], cost[0]); //convert to money

covert\_money1 g8 (for\_costs[1], cost[1]); //convert to money

covert\_money2 g9 (for\_costs[2], cost[2]); //convert to money

covert\_money3 g10 (for\_costs[3], cost[3]); //convert to money

covert\_money4 g11 (for\_costs[4], cost[4]); //convert to money

covert\_money5 g12 (for\_costs[5], cost[5]); //convert to money

covert\_money6 g13 (for\_costs[6], cost[6]); //convert to money

covert\_money7 g14 (for\_costs[7], cost[7]); //convert to money

covert\_money8 g15 (for\_costs[8], cost[8]); //convert to money

covert\_money9 g16 (for\_costs[9], cost[9]); //convert to money

//covert\_money0(for\_cost[0], cost[0]);

assign cost\_all = cost[0] + cost[1] + cost[2] + cost[3] + cost[4] + cost[5] + cost[6] + cost[7] + cost[8] + cost[9];

$display(cost\_all);

end

endmodule

TEST BENCH:

`timescale 1ns / 1ps

// Create Date: 21.11.2022 11:41:59

module testbenchmenu();

reg veg;

reg nonveg;

reg [0:9]dishes;

wire [1:2]select;

wire veg\_select;

wire nonveg\_select;

wire [0:9]for\_costs;

wire [0:9]cost;

menu fa0 ( veg, nonveg, dishes[0:9], select[1:2], veg\_select, nonveg\_select, for\_costs[0:9], cost[0:9]);

initial

begin

veg = 1; nonveg = 1; dishes[0:9] = 10'b1000010000; #10;

veg = 1; nonveg = 0; dishes[0:9] = 10'b1110000000;#10;

veg = 1; nonveg = 1; dishes[0:9] = 10'b1000000000;#10;

veg = 0; nonveg = 1; dishes[0:9] = 10'b0111100000;#10;

veg = 1; nonveg = 1; dishes[0:9] = 10'b0011001111;#10;

veg = 0; nonveg = 1; dishes[0:9] = 10'b0000011111;#10;

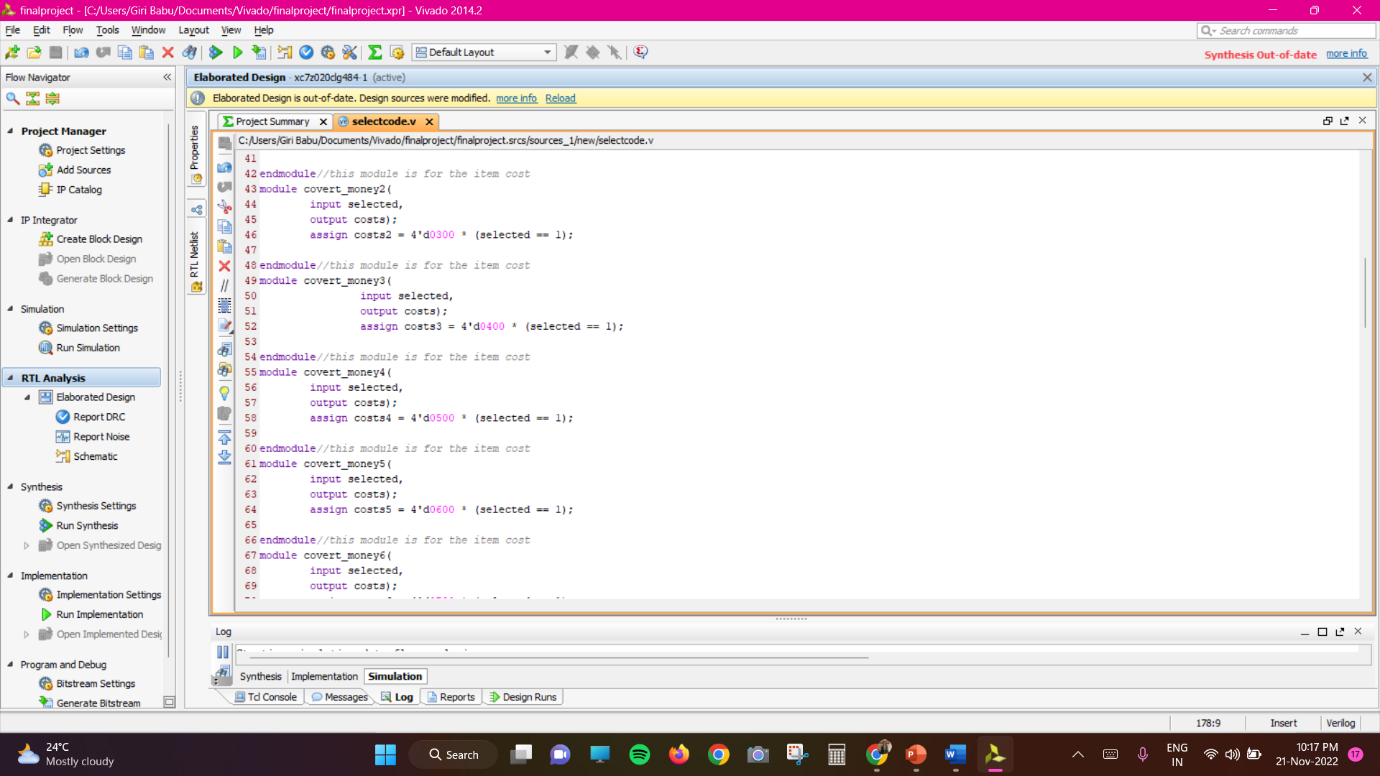
veg = 1; nonveg = 1; dishes[0:9] = 10'b1000011111;#10;

veg = 1; nonveg = 1; dishes[0:9] = 10'b0010011111;#10;

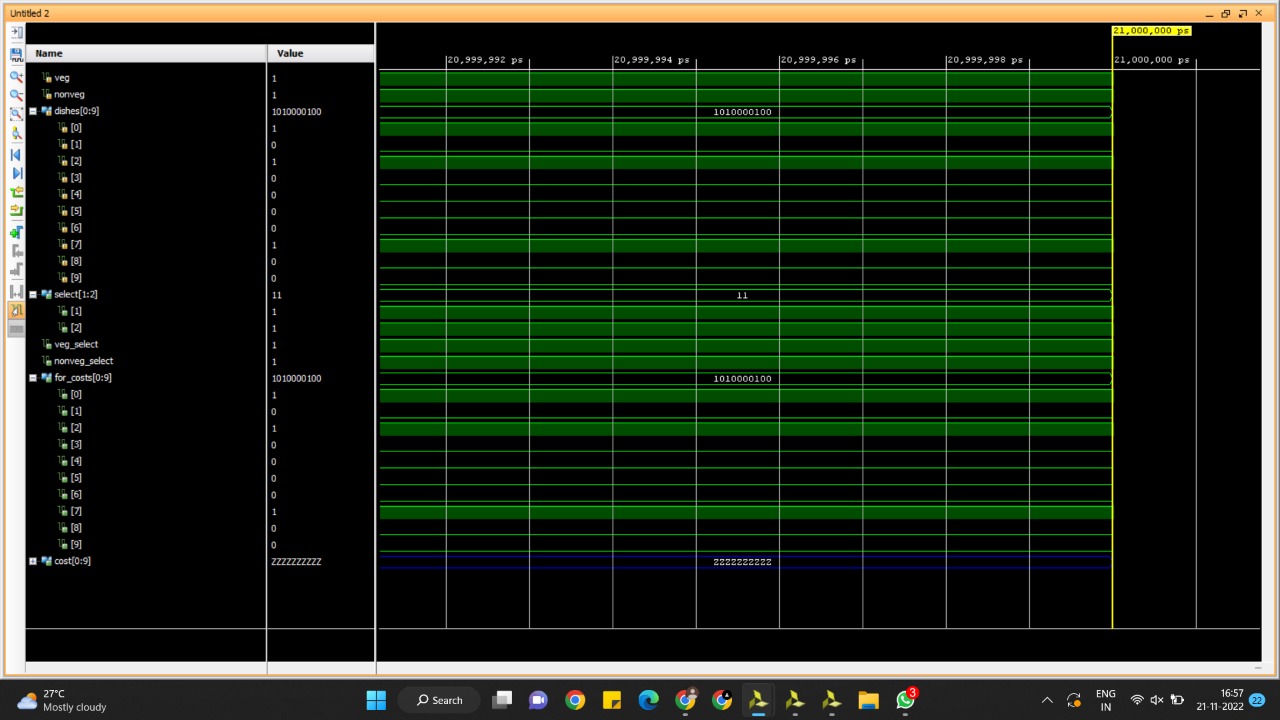
veg = 0; nonveg = 1; dishes[0:9] = 10'b000001111;#10;

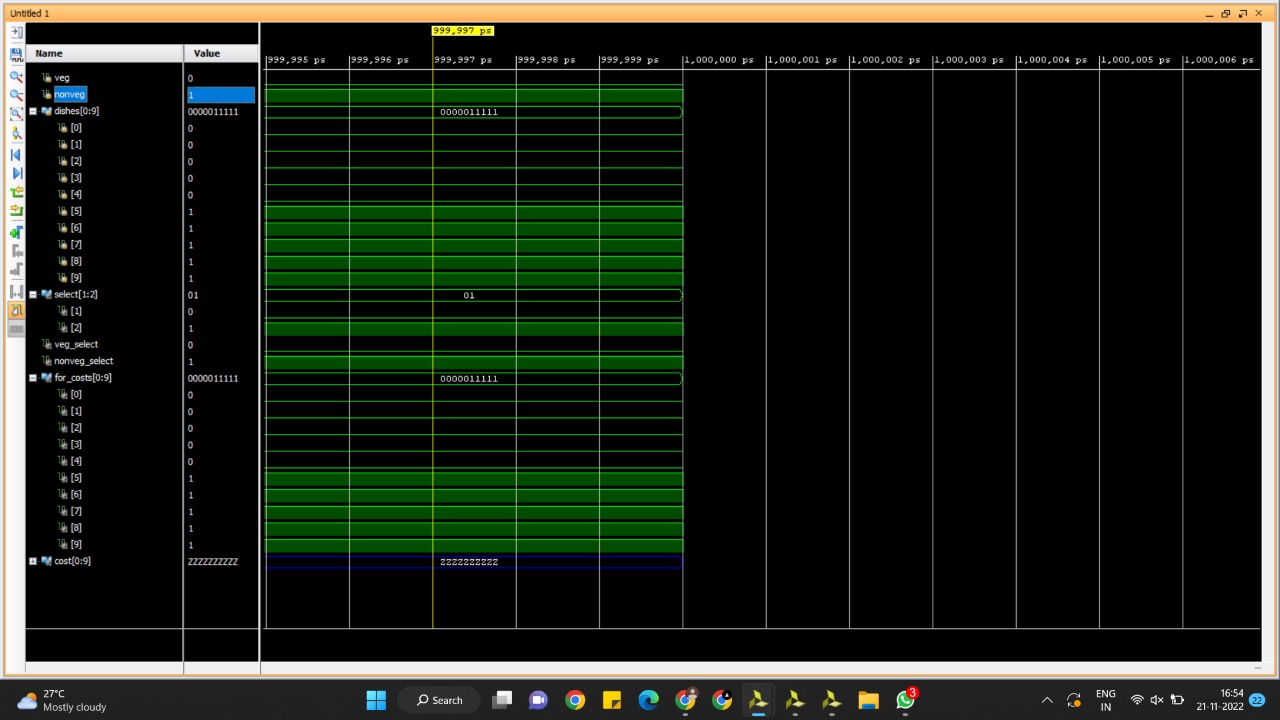
end

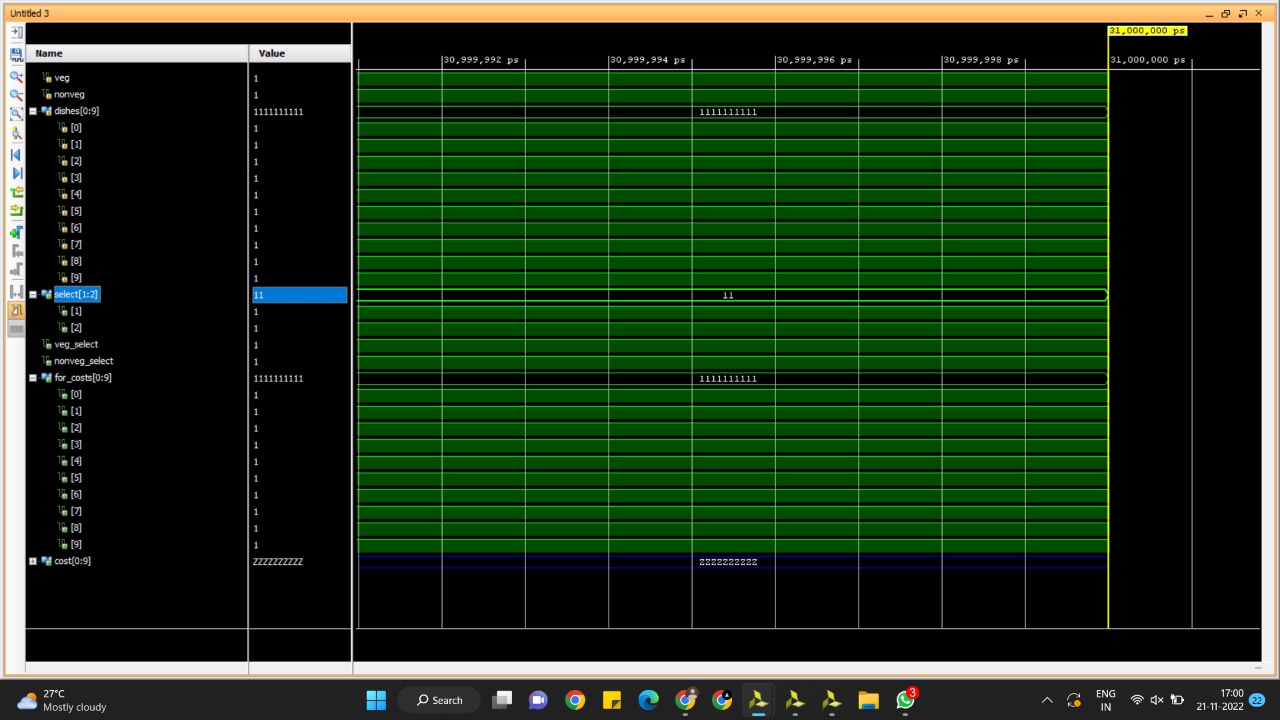
endmodule

OUR CODE: 

SIMULATION:

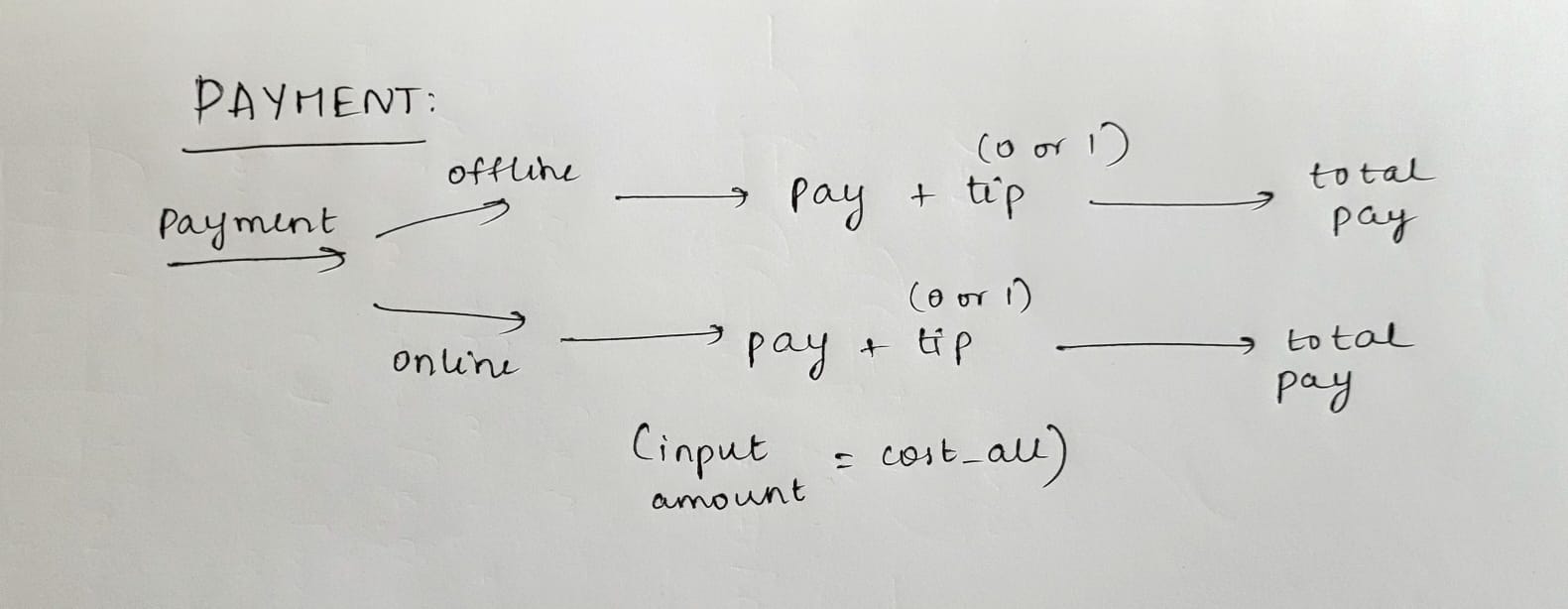






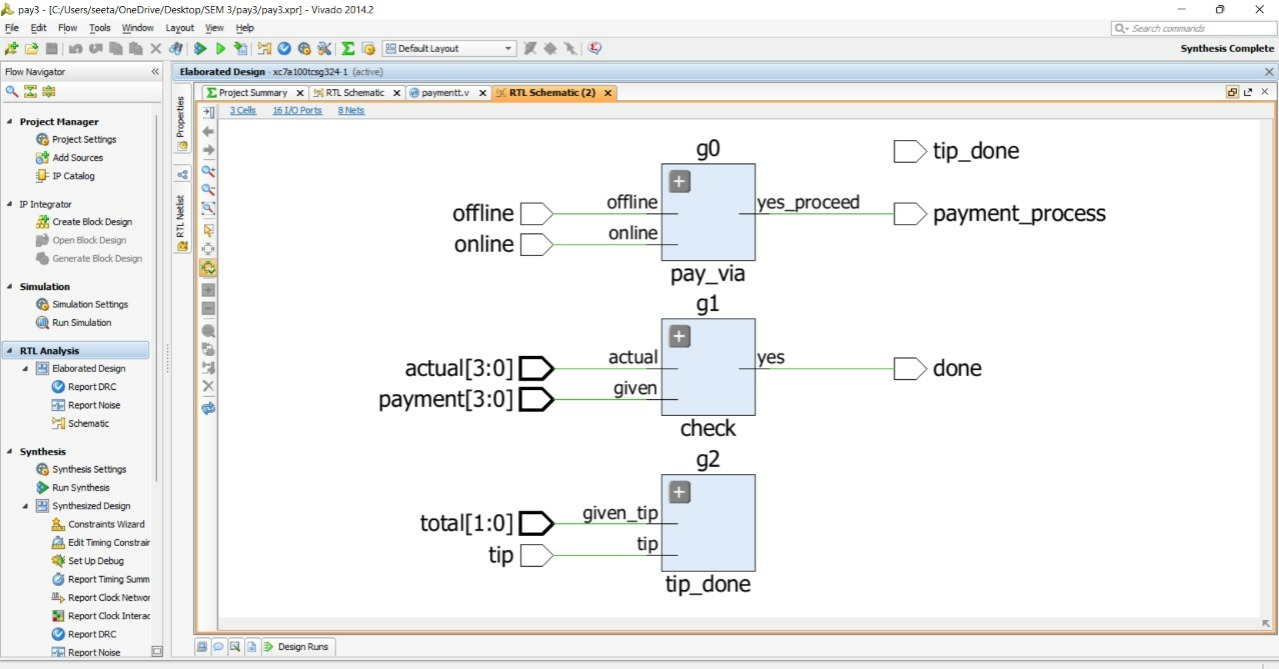
PAYMENT:

Payment is for displaying the total amount they have to pay after serving the items. If the customer wants to give a tip to the waiter they can include it in the total payment. The module is convenient enough to let the customer pay the money offline or online. If the mode is offline then the module exists, if online then he has to pay. There is one more advantage to visiting often to the restaurant as the prayer payment the person can give them an offer of 500/- off or etc, depending on the before-visit payment. This data is stored in the database.

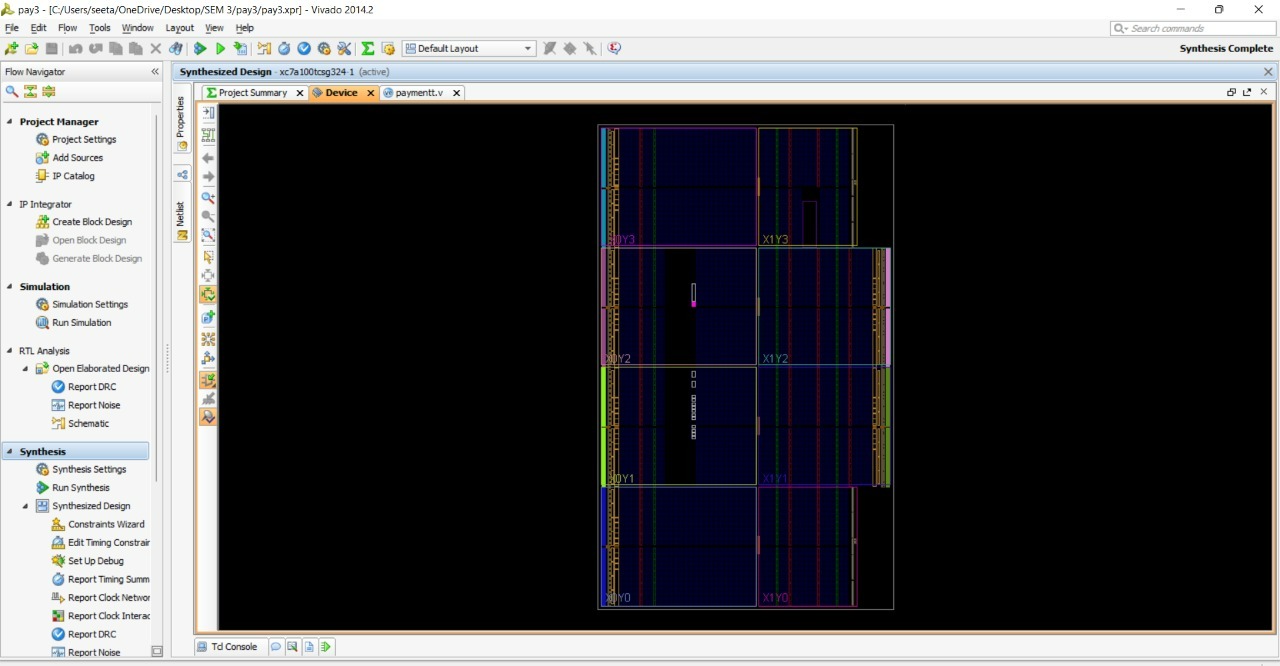




SCHEMATIC:



NETLISTS:



PAYMENT CODE:

`timescale 1ns / 1ps

// Create Date: 21.11.2022 11:41:59

module check(

input given,

input actual,

output yes);

assign yes = (given == actual);

endmodule

module pay\_via(

input online,

input offline,

output yes\_proceed);

assign yes\_procced = (offline == 0) \* (online == 1);

endmodule

module tip\_done(

input tip,

input given\_tip,

output total\_tip);

assign total\_tip = given\_tip \* (tip == 1);

endmodule

module payment(

input [3:0]payment,//for paying

input online, //if online payment

input offline, //if offline payment

output payment\_process, //for the process

input tip, //tip to server

input [3:0]actual,

output done,

output tip\_done,

input [1:0]total

);

pay\_via g0 (online, offline, payment\_process);

check g1 (payment[3:0], actual[3:0], done);

tip\_done g2 (tip, total[1:0], done\_tip);

endmodule

TEST BENCH:

`timescale 1ns / 1ps

// Create Date: 21.11.2022 11:41:59

module testbenchh();

reg [3:0]payment;

reg online;

reg offline;

reg [3:0]actual;

reg [1:0]total;

reg tip;

wire payment\_process;

wire done;

wire tip\_done;

payment fa0 ([3:0]payment, online, offline, payment\_process, tip, [3:0]actual, done, tip\_done, [1:0]total);

initial

begin

[3:0]payment = 4'd1010; online = 1; offline = 0; [3:0]actual = 4'd1200; [1:0]total = 2'd10; tip = 1; #10; online

[3:0]payment = 4'd1110; online = 0; offline = 1; [3:0]actual = 4'd1100; [1:0]total = 2'd11; tip = 1; #10;

[3:0]payment = 4'd1010; online = 0; offline = 1; [3:0]actual = 4'd1010; [1:0]total = 2'd10; tip = 0; #10;

[3:0]payment = 4'd1100; online = 1; offline = 0; [3:0]actual = 4'd1000; [1:0]total = 2'd11; tip = 1; #10;

[3:0]payment = 4'd1110; online = 1; offline = 0; [3:0]actual = 4'd1100; [1:0]total = 2'd01; tip = 1; #10;

[3:0]payment = 4'd1000; = 1; offline = 0; [3:0]actual = 4'd1000; [1:0]total = 2'd10; tip = 1; #10;

[3:0]payment = 4'd1011; online = 1; offline = 0; [3:0]actual = 4'd1111; [1:0]total = 2'd10; tip = 1; #10;

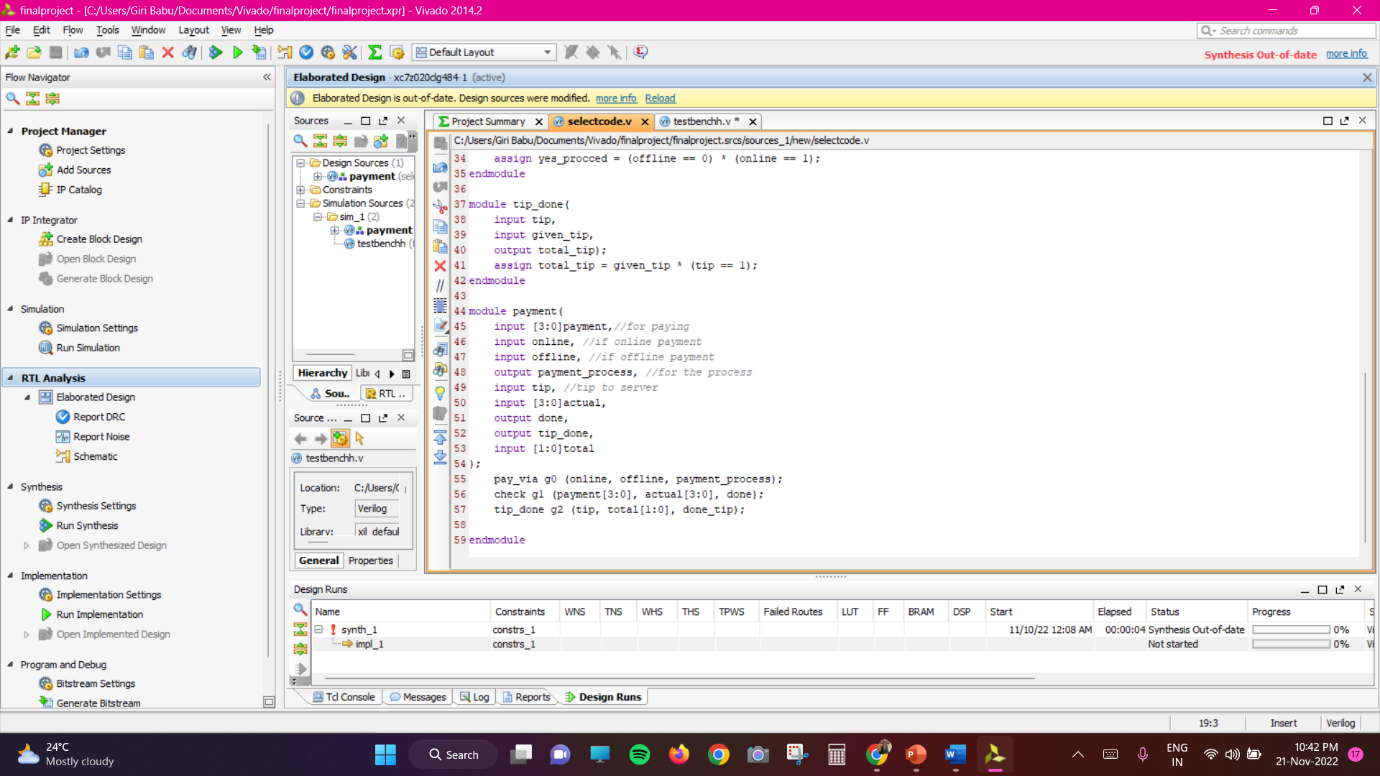
[3:0]payment = 4'd1100; online = 0; offline = 1; [3:0]actual = 4'd1100; [1:0]total = 2'd10; tip = 1; #10;

[3:0]payment = 4'd1011; online = 1; offline = 0; [3:0]actual = 4'd1000; [1:0]total = 2'd10; tip = 1; #10;

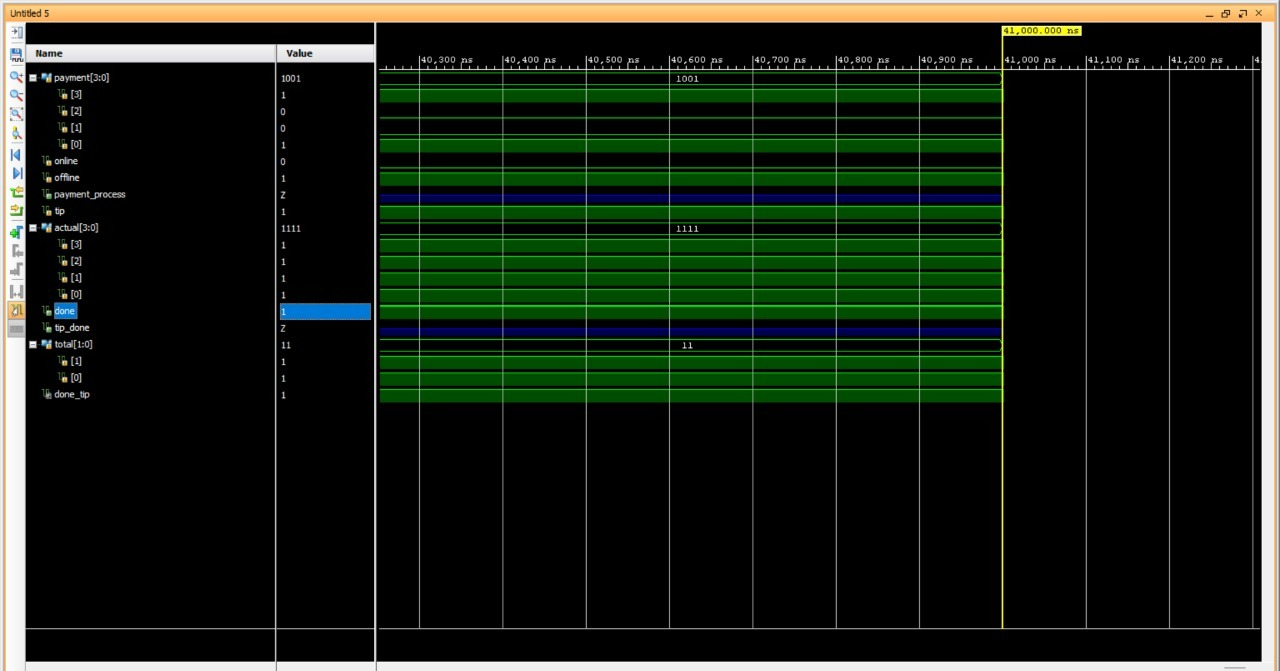
end

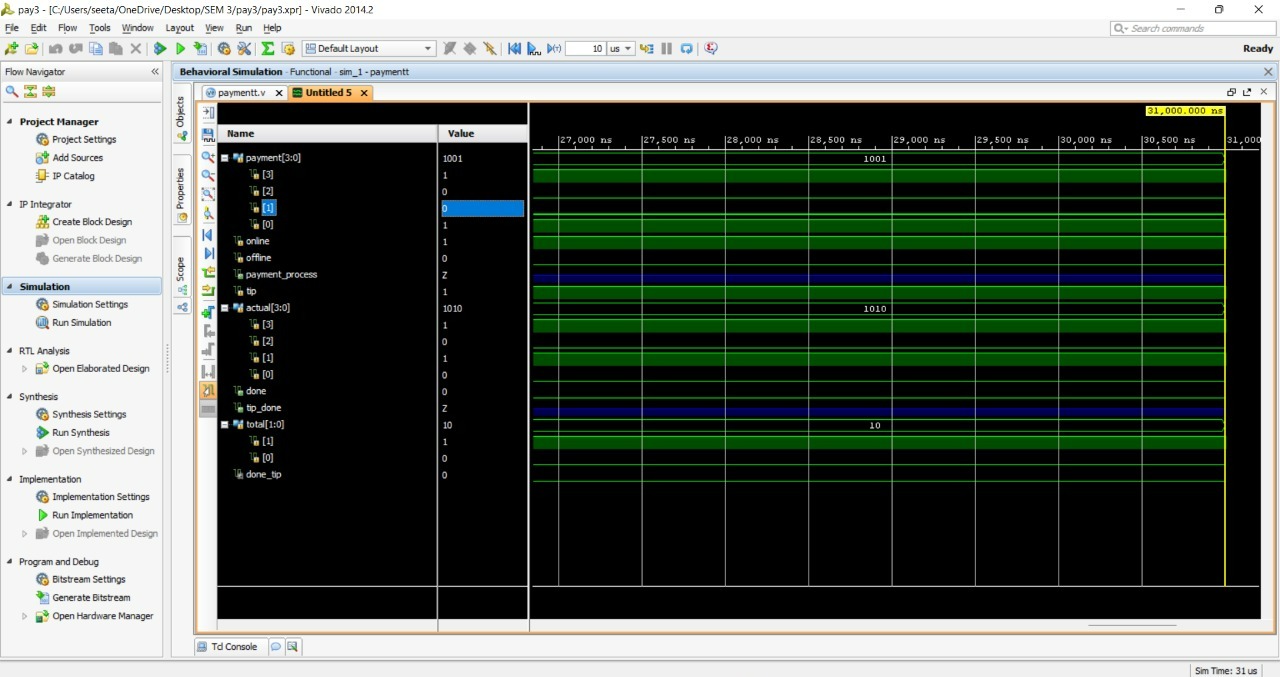
endmodule

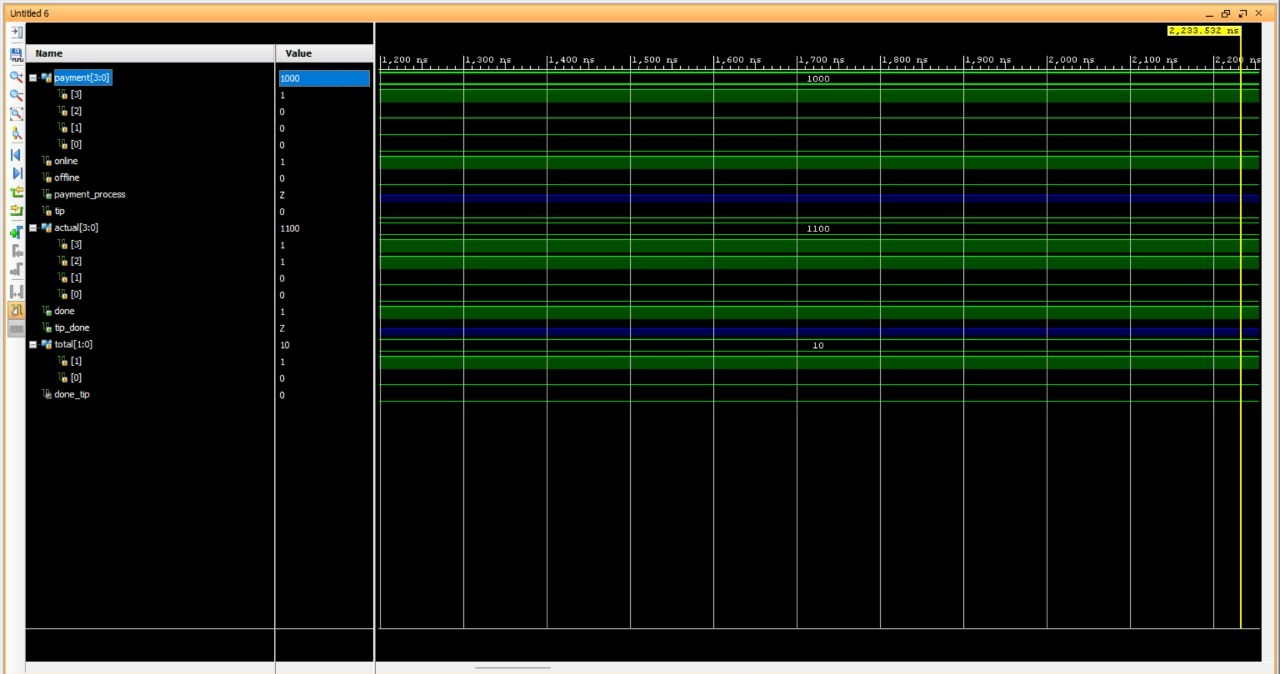
OUR CODE:



SIMULATION:







CONCLUSION:

1. This method helps us to decrease human efforts and save a lot of time.

2. This also has advantages like automation, as the system itself is automatic it helps a lot of data to be stored in very less time compared to manual actions.

3. This system also attracts most of the customers as per their comfort.

4. There is no worry of losing any instruction given by the customer.

REFERENCES:

-We got the inspiration to complete this project from observing the problems we face while ordering any food in restaurants.

-We are thankful to all the professors who guided and helped us to make this project successful.

-Below are the platforms to learn Verilog better.

<https://www.tutorialspoint.com/vlsi_design/vlsi_design_verilog_introduction.htm>

<https://www.javatpoint.com/verilog>

THANK YOU!