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

An internship report (SIP 2921) submitted by

STUDENTS NAME - Reg.No URK21CS3007

in partial fulfillment for the award of the degree of

BACHELOR OF TECHNOLOGY
in
COMPUTER SCIENCE AND ENGINEERING

under the supervision of

Dr.Rajeswari

DIVISION OF COMPUTER SCIENCE AND ENGINEERING
KARUNYA INSTITUTE OF TECHNOLOGY AND SCIENCES
(Declared as Deemed to be University under Sec-3 of the UGC Act, 1956)
Karunya Nagar, Coimbatore - 641 114. INDIA

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 COMPUTER SCIENCE AND ENGINEERING

BONAFIDE CERTIFICATE

This is to certify that the report entitled, "Design And Implementation Of Any Time Electricity Bill Payment Machine Controller " is a bonafide record of Internship work done at Intel Unnati Training Program during the academic year 2023-2024 by

JOSEPH STEPHY J (URK21CS3007)

in partial fulfillment of the requirements for the award of the degree of Bachelor of Technology in Computer Science and Engineering of Karunya Institute of Technology and Sciences.

Submitted for the Viva Voce held on 10.10.2023

Guide Signature

Dr.Rajeswari

ACKNOWLEDGEMENT

First and foremost, We praise and thank ALMIGTHY GOD whose blessings have bestowed in us the will power and confidence to carry out our internship.

We are grateful to our beloved founders **Late. Dr. D.G.S. Dhinakaran, C.A.I.I.B, Ph.D** and **Dr. Paul Dhinakaran, M.B.A, Ph.D**, for their love and always remembering us in their prayers.

We entend Our tanks to **Dr. Prince Arulraj, M.E., Ph.D., Ph.D.,** our honorable vice chancellor, **Dr. E. J. James, Ph.D.,** and **Dr. Ridling Margaret Waller, Ph.D.,** our honorable Pro-Vice Chancellor(s) and **Dr. R. Elijah Blessing, Ph.D.,** our respected Registrar for giving me this opportunity to do the internship.

We would like to thank **Dr. Ciza Thomas, M.E., Ph.D.,** Dean, School of Engineering and Technology for her direction and invaluable support to complete the same.

We would like to place my heart-felt thanks and gratitude to **Dr. J. Immanuel John Raja, M.E., Ph.D.,** Head of the Division, Computer Science and Engineering for his encouragement and guidance.

We feel it a pleasure to be indebted to, **Dr. Rajeswari**, Designation, Division of CSE & **Mr.k.Padmanaban** Intel Designation for their invaluable support, advice and encouragement.

We also thank all the staff members of the School of CSE for extending their helping hands to make this in Internship a successful one.

We would also like to thank all my friends and my parents who have prayed and helped us during the Internship.

DESIGN AND IMPLEMENTATION OF ELECTRICITY BILL PAYMENT

Overview Of Project

Electricity consumers are often faced with the problem of inaccuracy and delay in monthly billing due to some drawbacks. Thus, it is essential to have an efficient system for such purposes via electronic platform with consideration to proximity. The proposed system automates the conventional process of paying electricity bill by visiting the Electricity Board which is tiresome and time consuming. It is also designed to automate the electricity bill calculation and payment for user convenience. The system is developed with Verilog HDL as the base programming language which can be used as a language to to write text models that describe a logic circuit The system would be having two logins: the administrative and user login. The administrator can view the user's account details and can add the customer's information of consuming units of energy of the current month in their account. The Admin must feed the system with the electricity usage data into respective user’s account. The system then calculates the electricity bill for every user and updates the information into their account every month. Users can then view their electricity bill and pay before the month end.

CONTENTS

1.INTRODUCTION

Electricity Billing System is a software-based application.

i. This project aims at serving the department of electricity by computerizing the billing system.

ii. It mainly focuses on the calculation of units consumed during the specified time and the money to be charged by the electricity offices.

iii. This computerized system will make the overall billing system easy, accessible, comfortable, and effective for consumers.

To design the billing system more service oriented and simple, the following features have been implemented in the project. The application has high speed of performance with accuracy and efficiency.

The software provides facility of data sharing, it does not require any staff as in the conventional system. Once it is installed on the system only the meter readings are to be given by the admin where customer can view all details, it has the provision of security restriction.

The system excludes the need of maintaining paper electricity bill, administrator does not have to keep annual track of the users, users can pay the amount without visiting the office. Thus, it saves human efforts and resources.

1.1 Preamble:

We, the owners of our project, respect all customers and make them happy with our service.

The main aim of our project is to satisfy customer by saving their time by payment process, maintaining records, and allowing the customer to view his/her records and permitting them to update their details.

The firm handles all the work manually, which is very tedious and mismatched.

The objectives of our project are as follows:

- To keep the information of customer.
- To keep the information of consuming unit energy of current month.
- To keep the information of consuming unit energy of previous month.
- To calculate the units consumed every month regularly.
- To generate the bills adding penalty and rent.
- To save the time by implementing payment process online.

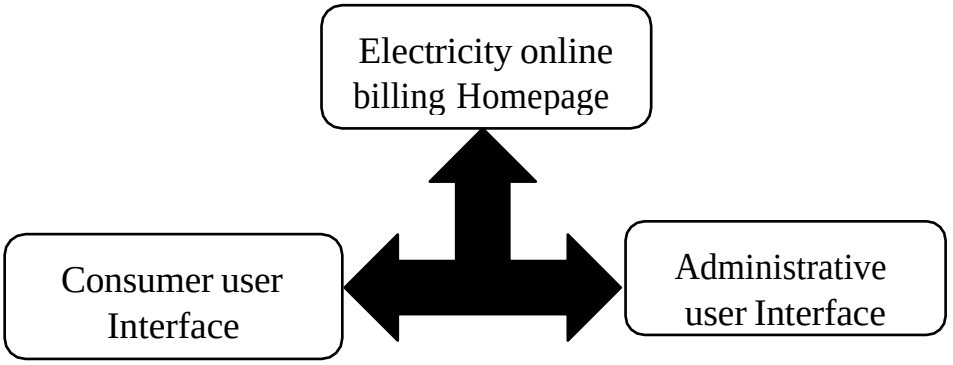


Fig 1.1.1: Block diagram showing the proposed online Electricity billing system.

1.2 Problem Statement:

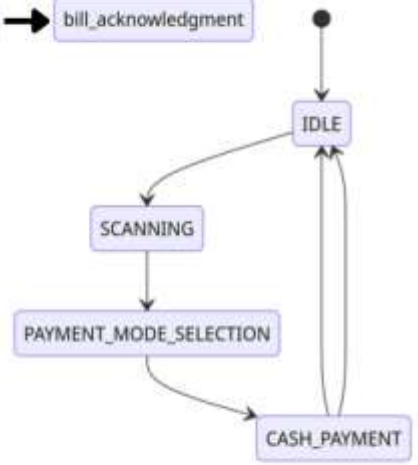
The manual system is suffering from a series of drawbacks. Since whole of the bills is to be maintained with hands the process of keeping and maintaining the information is very tedious and lengthy to customer. It is very time consuming and laborious process because, staff need to be visited the customers place every month to give the bills and to receive the payments. For this reason, we have provided features Present system is partially automated(computerized), existing system is quite laborious as one must enter same information at different places.so by this the users can pay using credit and debit card also and torn and wet cash are not accepted in this machine.

1.3 Proposed Solution:

- This project system excludes the need of maintaining paper electricity bill as all the electricity bill records are managed electronically.
- Administrator doesn't have to keep a manual track of the users. The system automatically calculates fine.
- Users don't have to keep a manual track of the users. The system automatically calculates fine.
- Thus, it saves human efforts and resources.

2.ANALYSIS AND SYSTEM REQUIREMENT

2.1 Block Diagram:



2.2 Feasibility Study:

Feasibility study is the phase in which the analyst checks that the candidatesystem is feasible for the organization or not.

This entails identification, description & evaluation of the system. Feasibility study is done to select thebest system that meets the performance requirement.

If the feasibility study is to serve as a decision document, it must answer keyquestions.

1.Is there a new and better way to do the job that will benefit the user?

2.What are the costs and savings of the alternatives?

3.What is recommended?

The most successful system projects are not necessarily the biggest or mostvisible in the business but rather those truly meet user's expectations.

Feasibility considerations:

Three key considerations are involved in the feasibility study. They are as follows:-

Economic Feasibility:

Economic analysis is the most frequently used method for evaluating the effectiveness of the candidate system.

We analyse the candidate system is feasible as than the manual system because it saves the money, time and manpower. It also feasible according to cost benefits analysis

Technical Feasibility:

Technical feasibility centers around the technology used. It means the c c candidate system is technically feasible i.e. it don't have any technical fault and work properly in the given environment. Our system is technically feasible; it is providing us required output.

Hardware Requirements:

- Hardware specification: Intel Processor
- 32 MB RAM or Higher
- 1.2 GB Hard Disk or Greater
- Video Display Unit
- Keyboard
- Mouse

Software Requirements:

- Operating System: Windows 10 or 11
- Software: Quartus Prime Lite Edison
- Front End: Verilog HDL

2.3 Algorithm(The Whole Project is Explained Step by Step):

Step-wise algorithm for the ATP (Any Time Electricity Bill Payment) machine controller:

1.Initialize the variables and registers: -Initialize the state variable to IDLE state. -Set the output registers (dis, successful, paidamt) to their initial values. -Initialize the totalcurrency, storedExcessAmount, and storedInsufficientAmount registers to zero. -Initialize the currentAmount, excessAmount, insufficientAmount, amt, and due variables to zero.

2. Define the state transition logic: -On every positive edge of the clock (clk), check if the system is in the reset state (rst). -If rst is active, set the state to IDLE. -If rst is not active, set the state to the next_state determined in the previous cycle.

3.Define the state-based behavior: -Use a combinational always @(*) block to determine the behavior based on the current state. -Inside the always @(*) block, use a case statement to evaluate the current state. -Based on the current state, define the behavior for each state. ANY TIME ELECTRICITY BILL PAYMENT MACHINE.

4. IDLE state: -Set the dis, successful, totalcurrency, and paidamt outputs to their default values. -Check the choice input to determine the next state: -If choice is 2'b01 (DD_IN), transition to the DD_IN state. -If choice is 2'b10 (CURRENCY_IN), transition to the CURRENCY_IN state.

5. DD_IN state: -Set the currentAmount to the ddamt input. -Transition to the CALCULATIONS state.

6. CURRENCY_IN state: -Based on the currency input, update the

currentAmount with the corresponding currency value. -If the currency input is invalid, keep the currentAmount unchanged. -Transition to the CURRENCY_IN state to allow for continuous currency inputs. -If the currency input is 4'b1000 (end of currency input), transition to the CALCULATIONS state.

7.CALCULATIONS state: -Calculate the due amount by adding the paymentAmount and storedInsufficientAmount. -Calculate the total amount (amt) by adding the currentAmount and storedExcessAmount. -Compare the amt with the due amount: -If amt is greater than or equal to due, calculate the excessAmount, set the insufficientAmount to zero, -update the storedExcessAmount, and transition to the SUCCESS state. -If amt is less than due, calculate the insufficientAmount, set the excessAmount to zero, -update the storedInsufficientAmount, and transition to the FAILED state.

8 .SUCCESS state: -Set the successful output to 1, dis output to 0. -Update the totalcurrency with the currentAmount. -Update the paidamt with the paymentAmount. -Transition back to the IDLE state.

9. FAILED state: -Set the successful output to 0, dis output to 1. -Update the totalcurrency with the currentAmount. -Update the paidamt with the paymentAmount. -Transition back to the IDLE state.

10.Default case: -If the current state is not explicitly handled, transition back to the IDLE state.

3.IMPLIMENTATION

3.1 Implementation of operations(Algorithmn):

1. Design and Simulation: Write the Verilog code for the ATP machine controller module and create a testbench for simulation to verify its functionality.

2. Synthesis: Use a synthesis tool to convert the Verilog code into a gate-level netlist, optimizing the design for the target technology.

3. Pin Planning: Determine the pin configuration and mapping of the ATP machine controller to the target device, ensuring compatibility and meeting any necessary constraints.

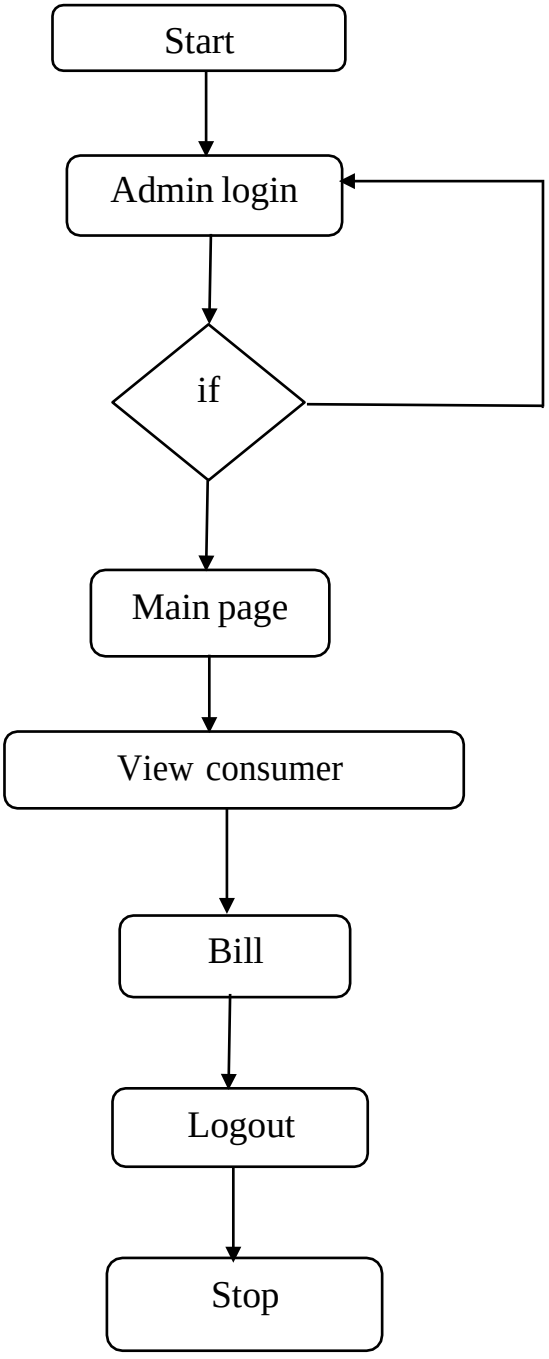
4. Chip Planning: Define the overall architecture and organization of the chip, including placement and routing strategies for the ATP machine controller within the chip.

5. Labs Land Implementation: Perform the physical implementation process, including place and route, to implement the design on the target device.

6. Performance Evaluation: Perform post-layout simulations to verify the functionality and measure key performance metrics such as timing, power consumption, and area utilization.

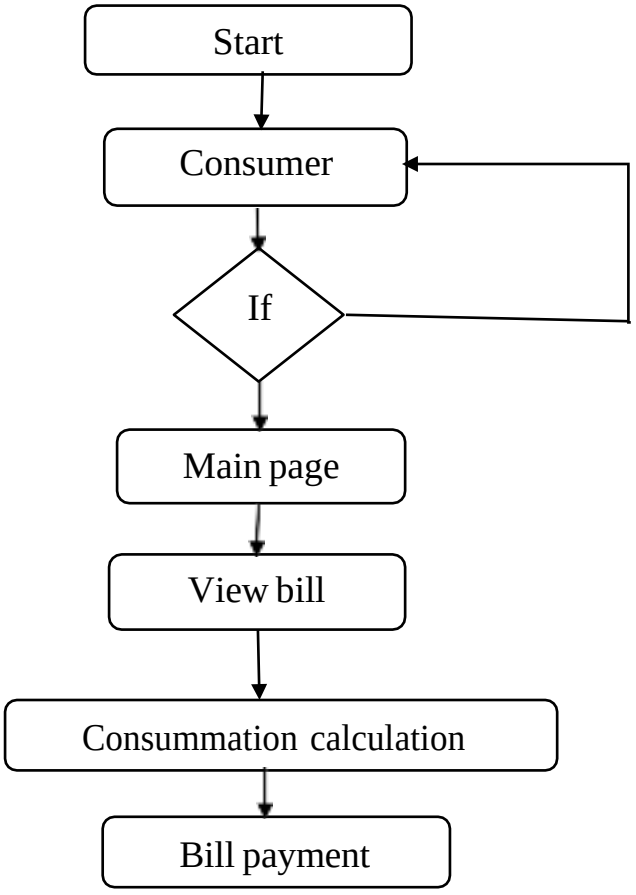
These six steps provide a high-level overview of the main stages involved in the implementation of the ATP machine controller.

***Activity Flow chart (administrator)**



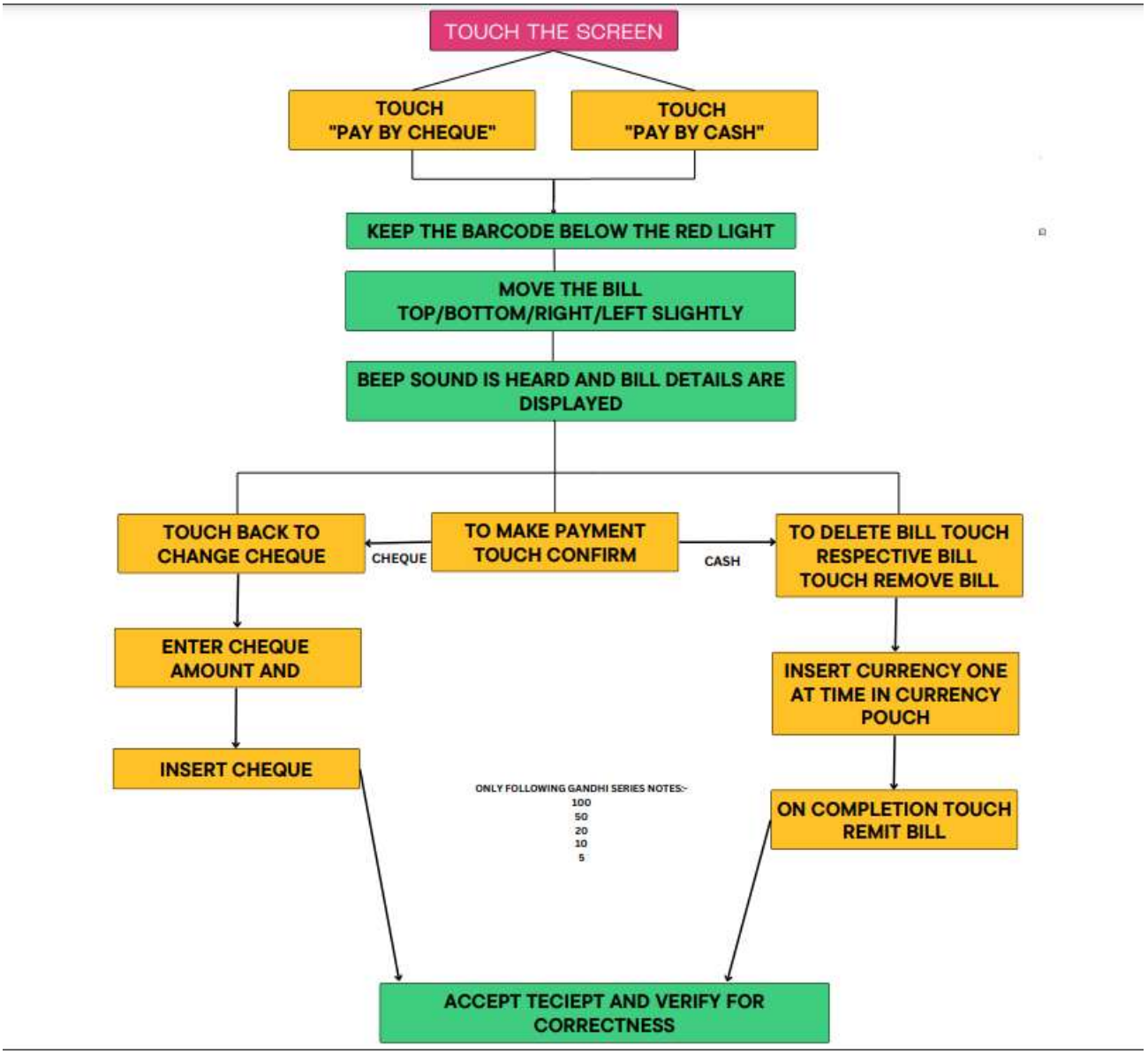
.

***Activity Flow chart(consumer)**



3.2 Data Flow DIAGRAM:

They are the versatile diagramming tools used for structured system analysis. They are specifically used for process modelling which involves graphically representing the function or process, which captures, manipulate, store, and distribute data between a system and its environment and between components within a system. Basically the tool we use in this for drawing diagram is draw.io.



4.TESTING

4.1 Testing process:

Testing is an integral part of software development. Testing process, in a way certifies, whether the product, that is developed, compiles with the standards, that it was designed to. Testing process involves building of test cases, against which, the product has to be tested. In some cases, test cases are done based on the system requirements specified for the product/software, which is to be developed.

4.2 Testing objectives:

The main objectives of testing process are as follows:

• Testing is a process of executing a program with the intent of finding an error.

• A good test case is one that has high probability of finding an as yet undiscovered error.

• A successful test is one that uncovers an as yet undiscovered error.

4.3 Levels of Testing:

Different levels of testing are used in the testing process; each level of testing aims to test different aspects of the system. The basic levels are unit testing, integration testing, system testing and acceptance testing.

4.3.1 Unit Testing:

Unit testing focuses verification effort on the smallest unit of software design the module. The software built, is a collection of individual modules. In this kind of testing exact flow of control for each module was verified. With detailed design consideration used as a guide, important control paths are tested to uncover errors within the boundary of the module.

Table : Negative test case for phone number insertion

Function name	Input	Expected error	Error	Resolved
Input phone number	98977	Phone is invalid	Length of phone number is not equal to 10	Consume()
Input phone number	9877avg	Phone number is invalid	Alphabets are being taken as input for phone number	

Table : Positive test case for phone number insertion

Function name	Input	Expected error	Error	Resolved
Input phone number	8690345678	Expected output is seen		

Table : Negative test case for email insertion

Function name	Input	Expected error	Error	Resolved
Input email	Sail.in	Email is invalid	Email is not in a format given	Consumer()

Table : Positive test case for email insertion

Function name	Input	Expected error	Error	Resolved
Input email	Akil23@gmail.com	Expected output is seen		

Table : Negative test case for customer name insertion

Function name	Input	Expected output	Error	Resolved
Input customer name	Sana123	Name is invalid	Number are being taken as input	Consume()

Table : Positive test case for customer name insertion

Function name	Input	Expected name	Error	Resolved
Input customer name	Joseph	Expected output		

4.3.2 Integration testing:

The second level of testing is called integration testing. In this, many class-tested modules are combined into subsystems, which are then tested. The goal here is to see if all the modules can be integrated properly. We have been identified and debugged.

Table : Test case on basis of generation of bill

Function name	Input	Expected error	Error	Resolved
Negative searching of total bill	1234(meter no) January(month)	Details seen but not total bill	Output not seen	Consume ()
Positive searching of total bill	1234(meter no) January(month)	Must display full generated bill		

Appendix:

Source code 1:(For Test Branch) ie)Pin Diagram:

```
module ATP_Machine_Electricity_Bill_Payment (
    input wire clk,
    input wire reset,
    input wire card_inserted,     // Card  insertion detection
    input wire [7:0] card_data,   // Card data (e.g., customer ID)
    input wire [3:0] pin,         // PIN for authorization
```

```
    input wire payment_1000,      // Payment in Rs 1000
    input wire payment_500,       // Payment in Rs 500
    input wire payment_100,       // Payment in Rs 100
    input wire payment_50,        // Payment in Rs 50
    output reg [7:0] display,     // Display output
    output reg payment_success, // Payment success flag
    output reg payment_fail,      // Payment failure flag
    output reg payment_timeout    // Payment timeout flag
);
    // Internal signal declarations
    reg [3:0] state;
    reg authorized;
    reg payment_completed;
    reg  excess_payment;
    reg [12:0] counter;
    reg [7:0] bill_amount;
    reg [7:0] payment_amount;
    reg [7:0] remaining_amount;
    reg [7:0] history_amount;

    // Constants
    localparam [7:0] BILL_AMOUNT = 8'hF4; // Total bill amount in
cents
    localparam [12:0] TIMEOUT_COUNTER = 13'd39062; //
Timeout duration in clock cycles

    // Clock process
    always @(posedge clk or posedge reset) begin
        if (reset) begin
            state <= 4'b0000;
            authorized <= 1'b0;
```

```
            payment_completed <= 1'b0;
            excess_payment <= 1'b0;
            counter <= 13'd0;
            bill_amount <= 8'd0;
            payment_amount <= 8'd0;
            remaining_amount <= 8'd0;
            history_amount <= 8'd0;
        end else begin
            case (state)
                4'b0000: begin // Idle state
                    if (card_inserted) begin
                        state <= 4'b0010; // Transition to data entry state
                    end
                end
                4'b0010: begin // Data entry state
                    if (1'b1) begin
                        state <= 4'b0011; // Transition to validation state
                    end
                end
                4'b0011: begin // Validation state
                    if (1'b1) begin
                        state <= 4'b0100; // Transition to bill process state
                        bill_amount <= BILL_AMOUNT;
                        remaining_amount <= bill_amount;
                    end
                end
                4'b0100: begin // Bill process state
                    if (1'b1) begin
                        state <= 4'b0101; // Transition to payment by cash
state
                        payment_amount <= 8'd0;
```

```
            end
        end
        4'b0101: begin // Payment by cash state
            if (payment_1000 || payment_500 || payment_100 ||
payment_50) begin
                payment_amount <= payment_amount +
(payment_1000 * 1000) + (payment_500 * 500) + (payment_100 *
100) + (payment_50 * 50);
                remaining_amount <= remaining_amount -
payment_amount;
            end
            if (remaining_amount == 0) begin
                payment_completed <= 1'b1;
            end
            if (1'b1) begin
                state <= 4'b0110; // Transition to acknowledge state
            end
        end
        4'b0110: begin // Acknowledge state
            if (1'b1) begin
                state <= 4'b0111; // Transition to transaction process
state
            end
        end
        4'b0111: begin // Transaction process state
            counter <= counter + 1;
            if (counter == TIMEOUT_COUNTER) begin
                state <= 4'b1101; // Transition to timeout state
                payment_timeout <= 1'b1; // Set timeout flag
            end else if (payment_completed) begin
                state <= 4'b1000; // Transition to payment confirm
```

```
state
                end else begin
                    state <= 4'b1110; // Transition to fail state
                end
            end
            4'b1000: begin // Payment confirm state
                if (excess_payment) begin
                    state <= 4'b1001; // Transition to reduce if excess
state
                end else begin
                    state <= 4'b1011; // Transition to receipt state
                end
            end
            4'b1001: begin // Reduce if excess state
                if (1'b1) begin
                    state <= 4'b1010; // Transition to reduction process
state
                end
            end
            4'b1010: begin // Reduction process state
                if (1'b1) begin
                    state <= 4'b1011; // Transition to receipt state
                end
            end
            4'b1011: begin // Receipt state
                if (1'b1) begin
                    state <= 4'b1100; // Transition to history state
                end
            end
            4'b1100: begin // History state
                if (1'b1) begin
```

```
                state <= 4'b0000; // Transition back to idle state
            end
        end
        4'b1101: begin // Timeout state
            if (1'b1) begin
                state <= 4'b0000; // Transition back to idle state
            end
        end
        4'b1110: begin // Fail state
            if (1'b1) begin
                state <= 4'b0000; // Transition back to idle state
            end
        end
        default: state <= 4'b0000; // Default case to handle any unexpected state
    endcase
  end
end

// Display output assignment
always @* begin
  case (state)
    4'b0010, 4'b0100: display = card_data; // Display card data during data entry and bill process states
    4'b0101: display = remaining_amount; // Display remaining amount during payment by cash state
    4'b0110: display = bill_amount; // Display bill amount during acknowledge state
    4'b1000, 4'b1001, 4'b1010: display = payment_amount; // Display current payment amount during payment confirm and reduction process states
```

```
            4'b1011: display = {excess_payment, payment_amount}; // Display excess payment and payment amounts during receipt state
            4'b1100: display = history_amount; // Display history amount during history state
            default: display = 8'd0; // Display 0 in other states
        endcase
    end

    // Payment success and failure flag assignment
    always @* begin
        case (state)
            4'b1000, 4'b1011: begin
                payment_success = 1'b1;
                payment_fail = 1'b0;
            end
            4'b1110: begin
                payment_success = 1'b0;
                payment_fail = 1'b1;
            end
            default: begin
                payment_success = 1'b0;
                payment_fail = 1'b0;
            end
        endcase
    end

endmodule
```

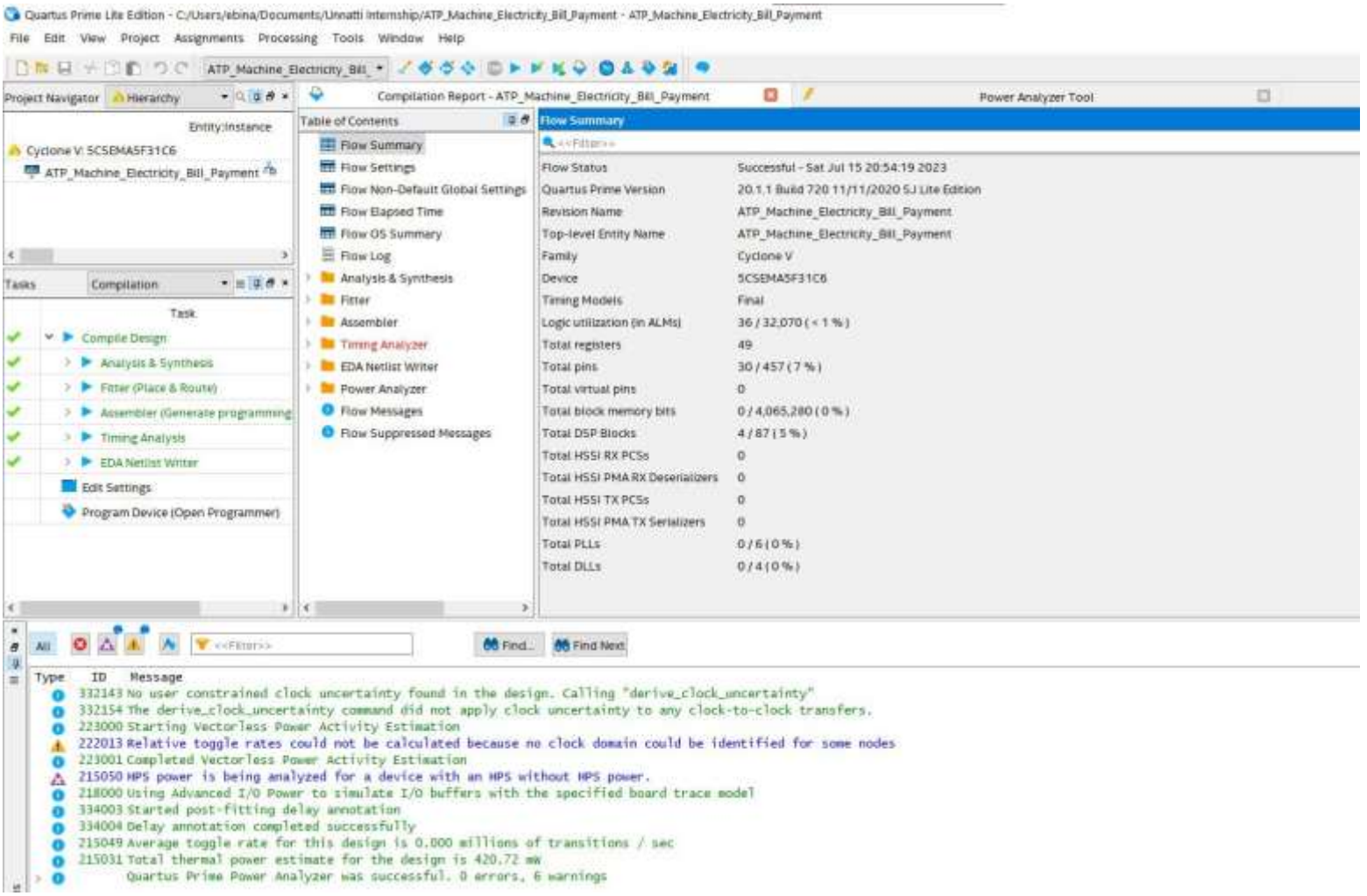
Source code2:(Verilog HDL Code)

```
module ATP_Machine_Electricity_Bill_Payment_TB;
    // Inputs
    reg clk;
    reg reset;
    reg card_inserted;
    reg [7:0] card_data;
    reg [3:0] pin;
    reg payment_1000;
    reg payment_500;
    reg payment_100;
    reg payment_50;

    // Outputs
    wire [7:0] display;
    wire payment_success;
    wire payment_fail;
    wire payment_timeout;

    // Instantiate the ATP Machine module
    ATP_Machine_Electricity_Bill_Payment ATP_Machine (
        .clk(clk),
        .reset(reset),
        .card_inserted(card_inserted),
        .card_data(card_data),
        .pin(pin),
        .payment_1000(payment_1000),
        .payment_500(payment_500),
        .payment_100(payment_100),
        .payment_50(payment_50),
```

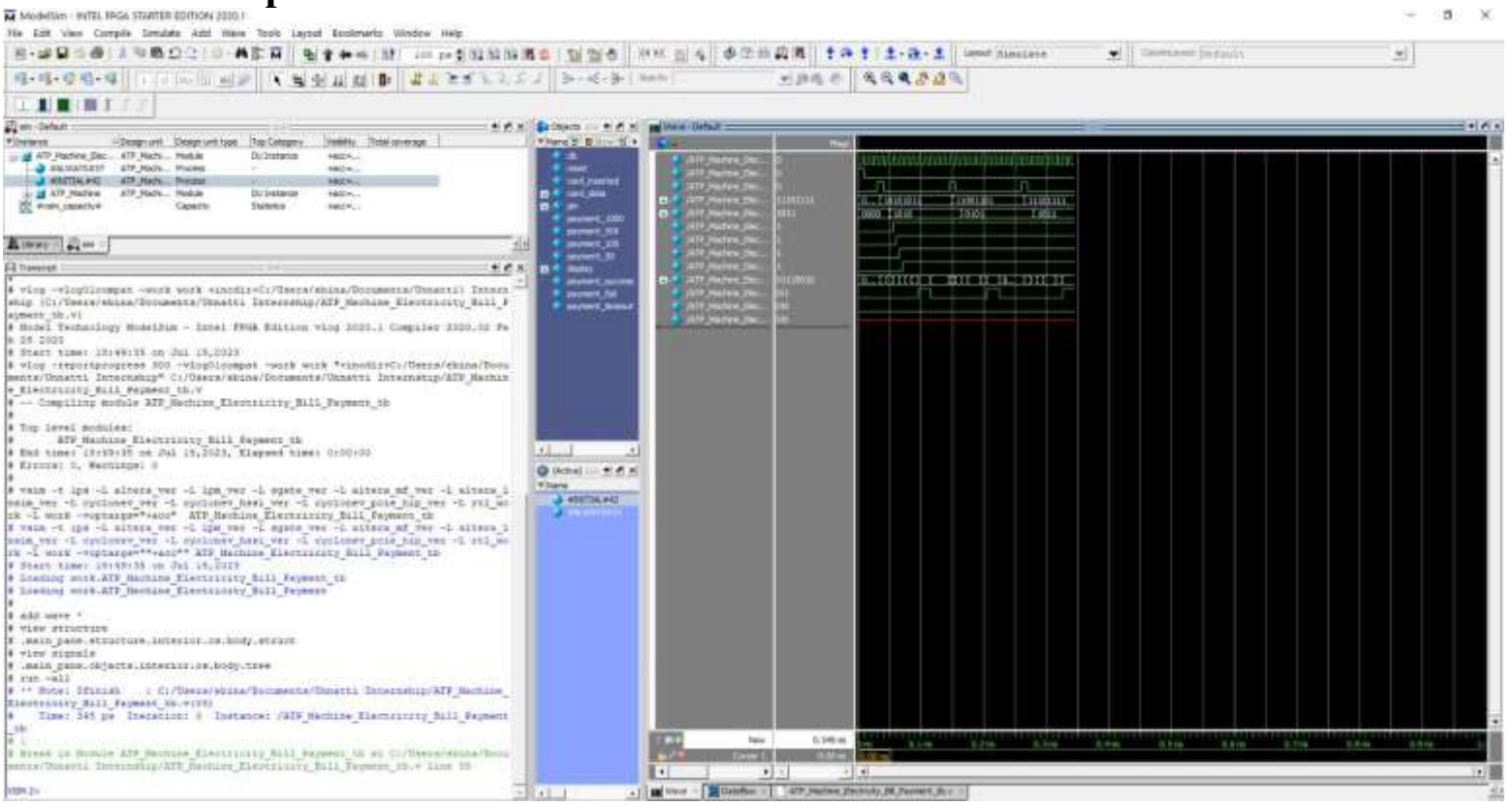
```
    .display(display),
    .payment_success(payment_success),
    .payment_fail(payment_fail),
    .payment_timeout(payment_timeout)
);

// Clock generation
always begin
    #5 clk = ~clk;
end

// Initialize inputs
initial begin
    clk = 0;
    reset = 1;
    card_inserted = 0;
    card_data = 0;
    pin = 0;
    payment_1000 = 0;
    payment_500 = 0;
    payment_100 = 0;
    payment_50 = 0;

    #10 reset = 0; // Deassert reset after 10 time units

    // Scenario 1: Successful payment
    #20 card_inserted = 1;
    #5 card_data = 8'hAB;
    #5 card_inserted = 0;
    #10 pin = 4'b1010; // Correct PIN
    #5 payment_1000 = 1; // Rs 1000 payment
```

```
        #5 payment_500 = 1; // Rs 500 payment
        #5 payment_100 = 1; // Rs 100 payment
        #5 payment_50 = 1; // Rs 50 payment
        #50;
        #5; // Add any additional waiting time for observation

        // Scenario 2: Failed payment
        #20 card_inserted = 1;
        #5 card_data = 8'hCD;
        #5 card_inserted = 0;
        #10 pin = 4'b0101; // Incorrect PIN
        #5 payment_1000 = 1; // Rs 1000 payment
        #5 payment_500 = 1; // Rs 500 payment
        #5 payment_100 = 1; // Rs 100 payment
        #5 payment_50 = 1; // Rs 50 payment
        #50;
        #5; // Add any additional waiting time for observation

        // Scenario 3: Timeout
        #20 card_inserted = 1;
        #5 card_data = 8'hEF;
        #5 card_inserted = 0;
        #10 pin = 4'b1011; // Correct PIN
        #50; // Wait for timeout
        #5; // Add any additional waiting time for observation

        // End simulation
        #10 $finish;
    end
endmodule
```

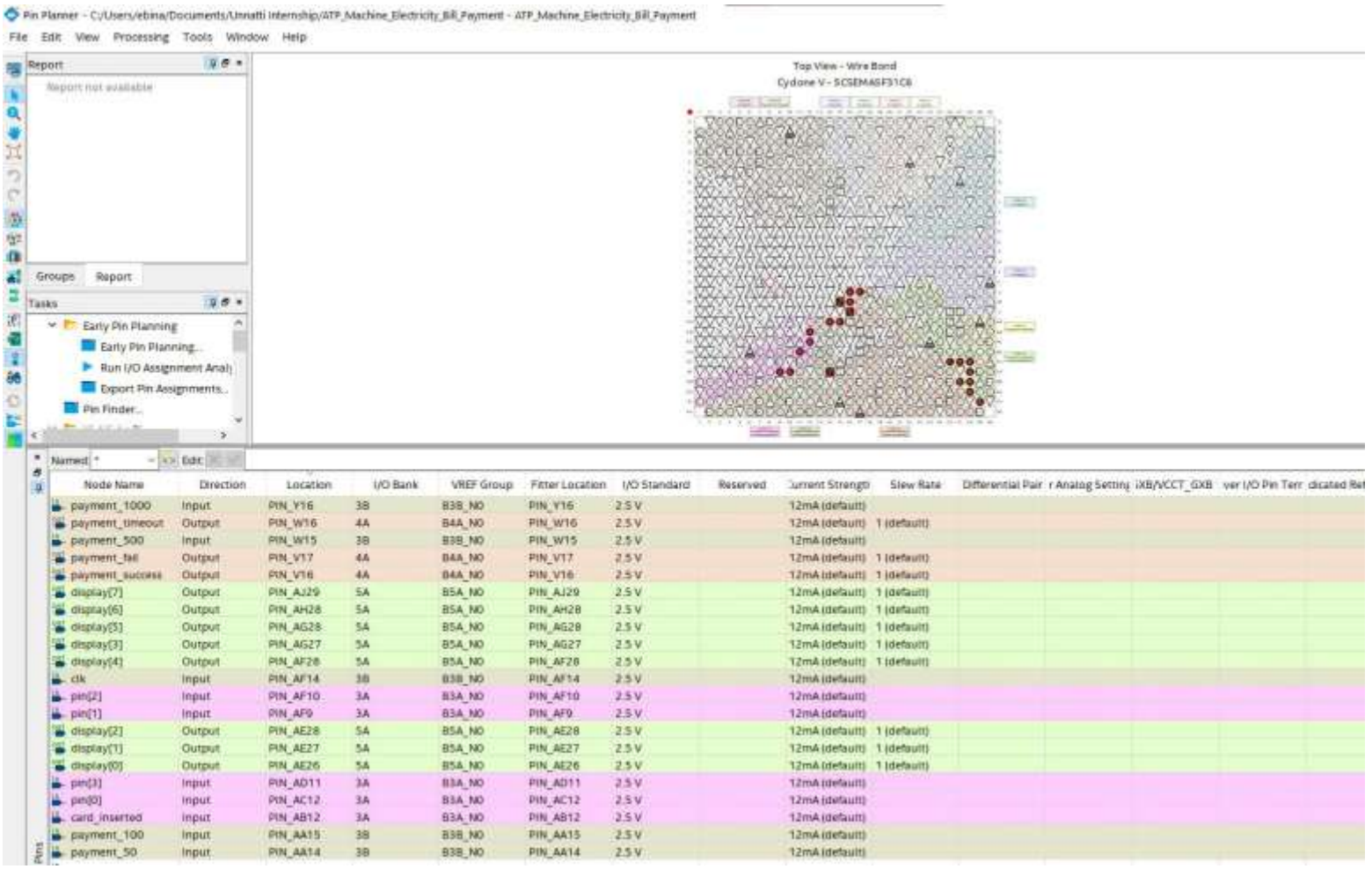
Output Screenshots:



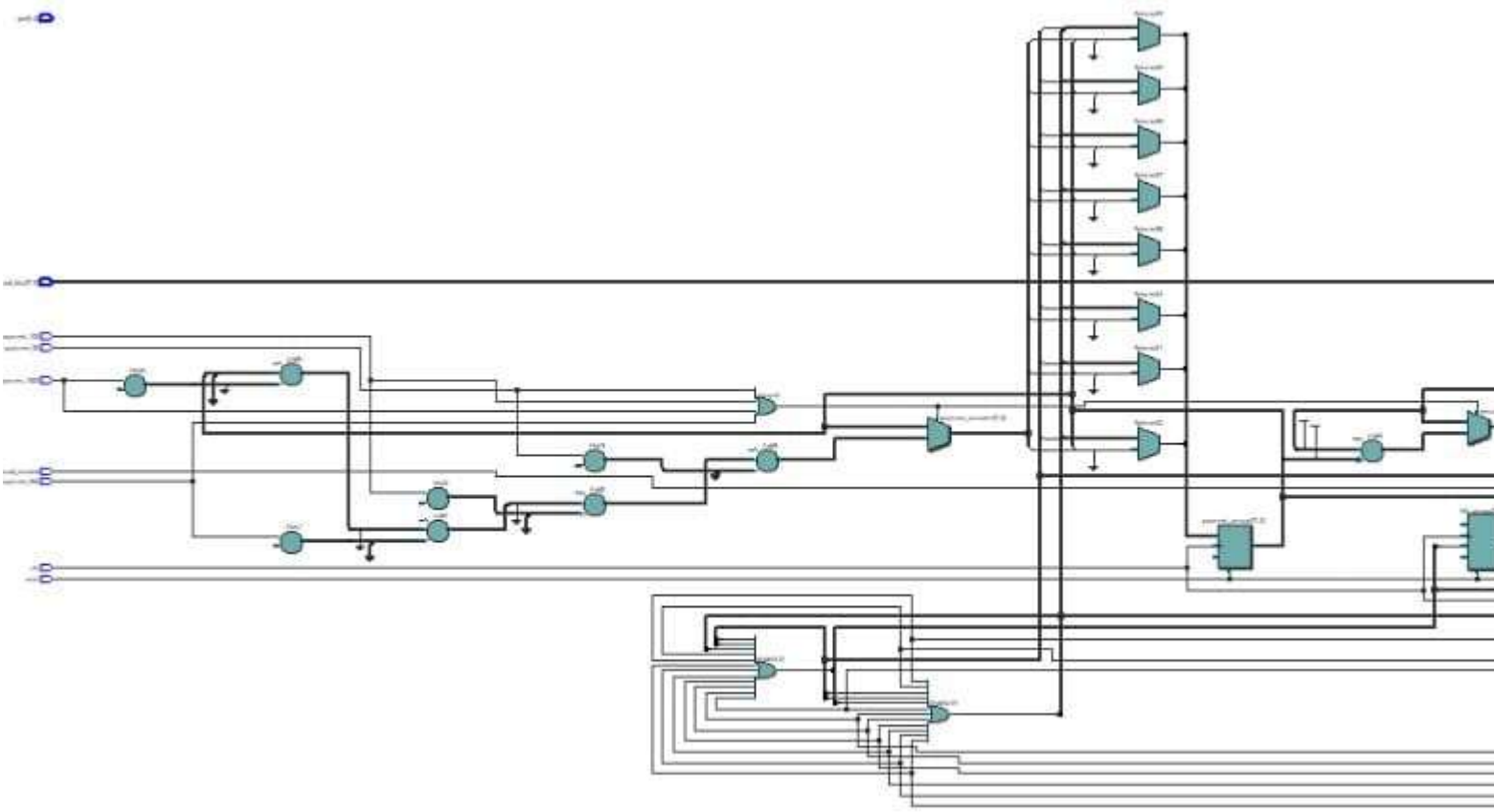
Data Flow Output:



Pin Diagram Output:



State Machine Diagram:



CONCLUSION

In conclusion, the implemented electricity payment system using Verilog HDL successfully handles barcode scanning, enables payment mode selection, and facilitates cash payment processing. The system exhibits robust behavior, accurately calculates amounts due and paid, and efficiently dispenses change. It provides a reliable and efficient solution for processing electricity bill payments. Future improvements can include support for additional payment modes and enhanced security measures. Overall, the project demonstrates the effectiveness of Verilog HDL in designing and implementing electronic payment systems.

Future Analysis

We have left all the options open so that if there is any other future requirement in the system by the user for the enhancement of the system then it is possible to implement them. We hope that the project will serve its purpose for which it is develop there by underlining success of process