ASIC Design of ATM Controller

Sumit Vishwakarma (202201320), Tirth Modi (202201513)
Dhirubhai Ambani Institute of Information and Communication Technology
Gandhinagar, Gujarat, 382007, India
EL527: ASIC Design - Prof. Rutu Parekh

***Abstract*—ATM machines have played a crucial role in the global banking infrastructure since the time they were invented by the Barclays Bank in 1967. In this study report, we begin by examining the working and functioning of ATM machines. Subsequently we try to create a Finite State Machine diagram and a block diagram based on our understanding of the working of the system.**
We then try to implement the same using:

- **Verilog HDL, for core control logic for the ATM Machine**
- **Qflow tool, an EDA tool to generate the physical chip layout**
- **Magic editor, to review the layout in GUI**

Our ultimate goal is to understand the block diagram correctly, prepare a Finite State Machine Diagram for the same and then implement the working of ATM controller in form of RTL code using Verilog HDL. Additionally, we synthesis it to observe and visualize the industry ready physical layout of controller, using Qflow, an EDA tool for ASIC design. Future scope: This design would then be ready for fabrication purpose.

I. Introduction

Automated Teller Machines (ATMs) are an integral part of current banking infrastructure, enabling users to perform self-service financial operations such as withdrawals, balance inquiries, and PIN-based authentication. With increasing emphasis on speed, security, and power efficiency, hardware-level implementations of ATM logic are becoming increasingly relevant. In our report, we aim to demonstrate the working of hardware-based ATM logic by synthesizing this RTL design into an ASIC layout using the open-source Qflow. Upon this, we visualize the layout of the chip which is ready to be fabricated.

II. System Overview

The ASIC implementation of an ATM machine aims to design a hardware-chip that can manage all key operations of ATM machines such as authentication, transaction processing, and security enforcement. As far as the Verilog code is concerned, it's the hardware description language (HDL) we use to describe and implement the digital logic that controls the ATM's core logical operations. Our first step is to understand the core logic of ATM's functioning by breaking it down into discrete stages — such as card detection, PIN verification, and transaction handling. Secondly, we create a flowchart or block diagram to visualize step-by-step operations and define the corresponding FSM states. This structured flow aids in implementing each logical block in the Verilog code efficiently. All the Verilog modules are synthesized using an open-source Qflow tool, which is an EDA tool that converts our written RTL (Register Transfer Level) logic code into a gate-level netlist for layout generation. The final physical layout of the design can then be visualized using the Magic editor, ensuring correct placement and routing of logic gates on silicon. This system overview demonstrates how an ATM machine can be re-imagined as a secure hardware state machine, suitable for real-world ASIC fabrication.

A. *Flow Diagram and FSM Diagram*

Given below is the flowchart that describes the working of an ATM Machine. Based on our understanding of this

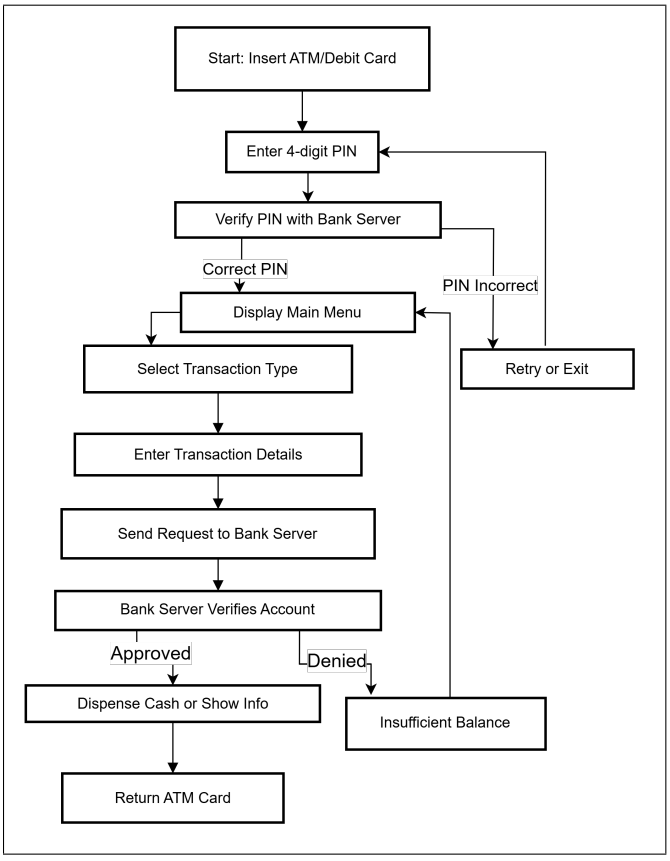

Fig. 1. Flow Chart - ATM Machine

flow diagram, which explains the core logic of working of the machine, we have tried to create an FSM diagram, which clearly displays all the states and possible transitions from one state to another.

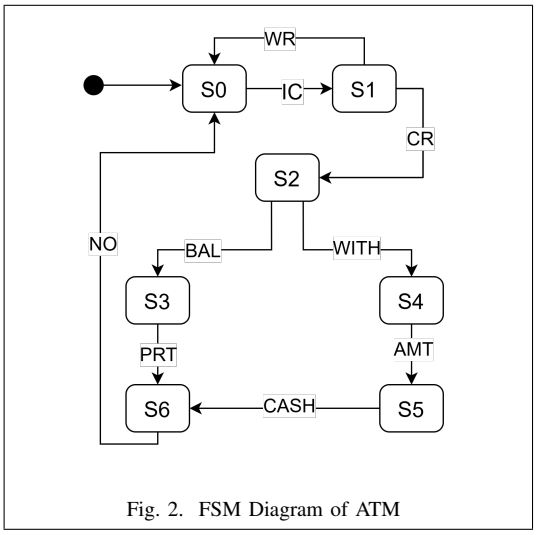

Fig. 2. FSM Diagram of ATM

TABLE I
STATE DESCRIPTIONS

State Code	State Description
S0	Idle State
S1	PIN Enter
S2	Transaction Type
S3	Balance Check
S4	Withdraw
S5	Amount
S6	Balance Display

TABLE II
TRANSITION DESCRIPTIONS

Transition	Description
IC	Insert Card
WR	Wrong PIN
CR	Correct PIN
BAL	Balance Check
WITH	Withdraw Money
PRT	Print Balance
AMT	Amount Entered
CASH	Cash Ejection
NO	Exit

III. RTL IMPLEMENTATION - VERILOG

This section describes how the ATM functionality is modeled using finite state machines (FSM) and digital logic.

- **Finite State Machine (FSM):** The core logic of the ATM controller is built by analyzing the FSM with states such as Idle, PIN Enter, Transaction Type, Balance Check, Withdraw, Amount and Balance display. Each state represents a specific phase in the ATM operation, and transitions are triggered based on the user inputs.
- **Inputs and Outputs:** We initialized the following inputs: clock, insert_card, pin_input, transaction_select, amount, and produces outputs like state indicators, and auth_access to indicate the time period for transaction.
- **Timing and Control Logic:** Initialized the synchronous clock signal to control state transitions. Reset logic ensures the system returns to the Idle state upon completion of all transitions.
- **Simulation and Testing:** We have manually done the functional verification using a Verilog testbench .v file which correctly simulated all necessary state transitions like PIN attempts, Transaction selection including Money Withdrawal, Balance Checking, and subsequently cash dispensing upon withdrawal. Additionally, we also tested the freeze condition after three wrong PIN attempts, ensuring the FSM locks the user out for a specific duration before allowing new input.
- **Synthesis Considerations:** Our Verilog RTL is written in a way so as to ensure its compatibility in EDA tools like Qflow to ensure accurate synthesize of our VHDL code and finally to create a proper chip layout design out of it.

```
module ATM_Machine (
    input clk,
    input reset,
    input insert_card,
    input [15:0] pin_input,
    input correct_pin,
    input balance_check,
    input withdraw,
    input print_balance,
    input amount_entered,
    input cash_eject,
    input exit,
    output reg [2:0] state,
    output reg auth_success,
    output reg freeze
);

    parameter S0 = 3'd0; // Idle state
    parameter S1 = 3'd1; // PIN Entry state
    parameter S2 = 3'd2; // Transaction Type state
    parameter S3 = 3'd3; // Balance Check state
    parameter S4 = 3'd4; // Withdraw state
    parameter S5 = 3'd5; // Amount Entered state
    parameter S6 = 3'd6; // Balance Display state

    reg [1:0] wrong_pin_counter;
    reg [31:0] freeze_timer;

    always @(posedge clk or posedge reset) begin
        if (reset) begin
            state <= S0;
            auth_success <= 0;
            freeze <= 0;
            wrong_pin_counter <= 0;
            freeze_timer <= 0;
        end else begin

            if (freeze) begin
                if (freeze_timer < 120000)
                    freeze_timer <= freeze_timer + 1;
                else begin
                    freeze <= 0;
                    wrong_pin_counter <= 0;
                    freeze_timer <= 0;
                end
            end else begin
                case (state)
                    S0: begin
                        auth_success <= 0;
                        if (insert_card)
                            state <= S1;
```

```
                    S1: begin
                        if (correct_pin) begin
                            auth_success <= 1;
                            state <= S2;
                            wrong_pin_counter <= 0;
                        end else begin
                            wrong_pin_counter <= wrong_pin_counter + 1;
                            if (wrong_pin_counter == 2) begin
                                freeze <= 1;
                                freeze_timer <= 0;
                                state <= S0;
                            end
                        end
                    end

                    S2: begin
                        if (balance_check)
                            state <= S3;
                        else if (withdraw)
                            state <= S4;
                        else if (print_balance)
                            state <= S6;
                        else if (exit)
                            state <= S0;
                    end

                    S3: begin
                        if (print_balance)
                            state <= S6;
                        else if (exit)
                            state <= S0;
                    end

                    S4: begin
                        if (amount_entered)
                            state <= S5;
                        else if (exit)
                            state <= S0;
                    end

                    S5: begin
                        if (cash_eject)
                            state <= S0;
                    end

                    S6: begin
                        if (exit)
                            state <= S0;
                    end

                    default: state <= S0;
                endcase
            end
        end
    end
endmodule
```

Fig. 3. Verilog Code - ATM Machine

IV. Chip Layout - Qflow Tool

- **Synthesis:** After synthesis of our RTL code in Verilog, synthesis of the verilog code takes place which converts the written logic into the gate level netlist.
- **Placement:** So after we represented the RTL code as a gate level representation, there is a need to assign these gate level connections to a physical position. This process is referred as placement. Its main motive is to minimize the wire length used in the chip and reduce the delay across the layout.
- **Static Timing Analysis (STA):** This is an important process as it verifies the timing constraints and correctness without running any kind of simulation. It mainly verifies setup time constraints and hold time constraints.
- **Routing:** After the physical positions of these logic gates are assigned, the next step is to interconnect al these cells using metal wires. this process is called routing and is an integral part of ASIC design process.
- **Chip Layout:** Upon completion of all these processes, the design of chip layout is generated correctly by the Qflow tool as shown in Figure 4: Chip Layout. We obtained the following layout upon successful completion of all these steps listed in Figure 5: Qflow Output Log. Our design passed all the stages, which validates our design for ATM controller.

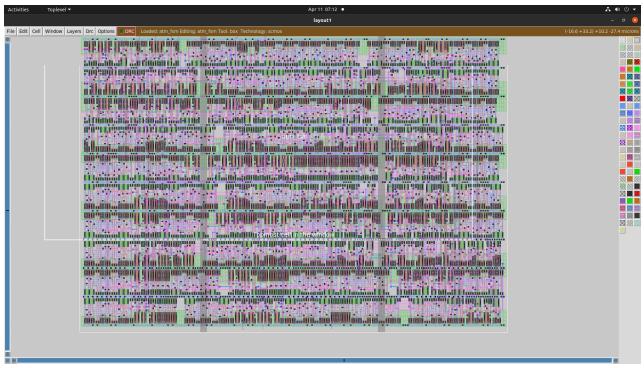

Fig. 4. Chip Layout

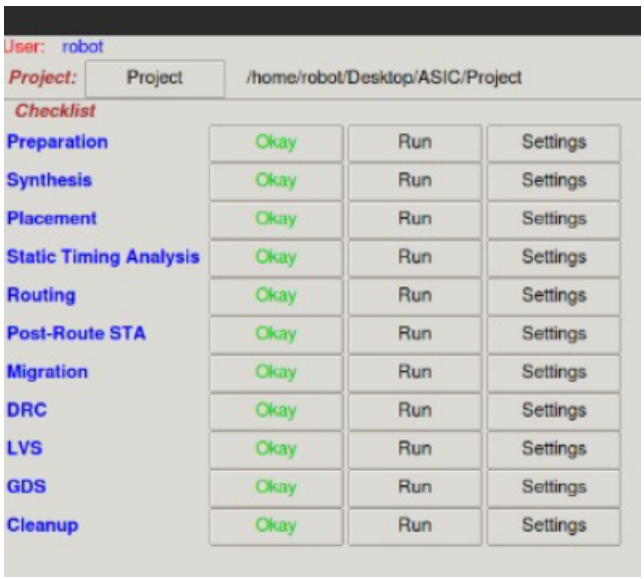

Fig. 5. Qflow Output Log

V. Waveform Analysis

- **Overview:** In this section, we have written the detailed output observation and their respective explanations to clearly understand the working logic of the ATM controller and the transitions taking place across different states mentioned in the FSM diagram.
- **pin_input[15:0]:** As ATM PIN numbers are of 4 digit, we represent it as a 16 bit signal, 4 bits for each 4 digits. It gets activated when the user enters the PIN, as shown in the waveform. Its value remains stable during the authentication process.

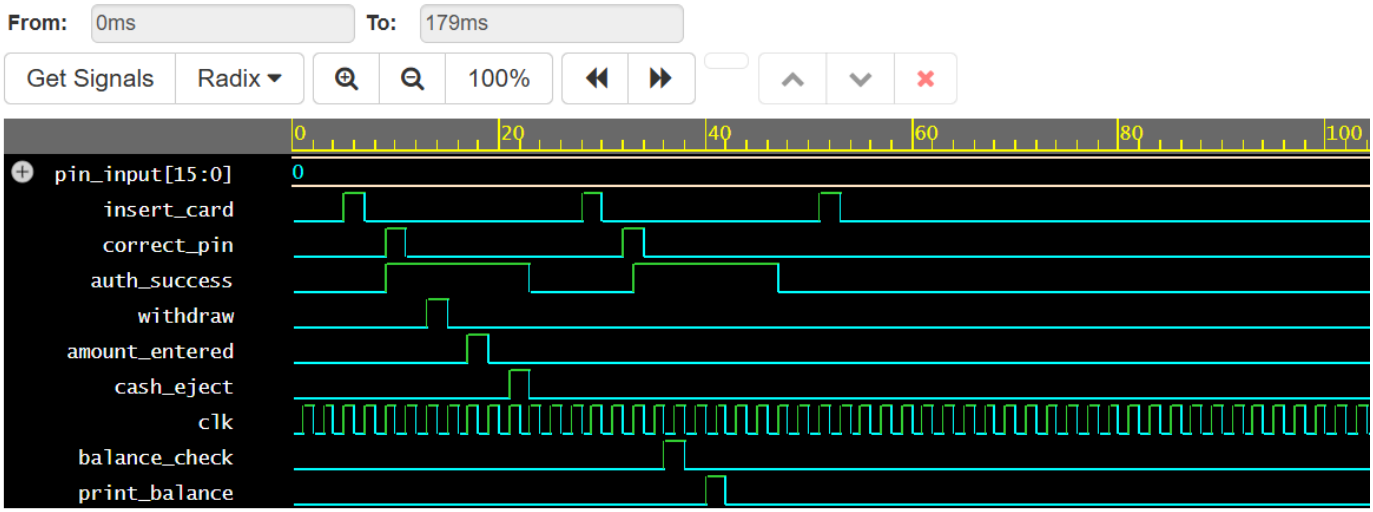

Fig. 6. Waveform Output

- **insert_card:** The insert card signal is triggered when the user inserts the ATM card in the machine card-input. As observed from the waveform, it correctly gets triggered upon insertion of cards at several time points.
- **correct_pin:** This signal is triggered high when the user inputs the correct PIN number for the respective ATM card.
- **auth_success:** If the user inputs the correct PIN number, the auth_success signal is triggered and it remains high during the entire session whether the user prefers either withdrawal or balance check. Basically, it confirms that authentication has been successful and subsequently, enables access to further operations.
- **withdraw:** This signal is triggered when a user wants to request a withdrawal. In the waveform, it is clearly activated after authentication, showing a user's decision to withdraw cash.
- **amount_entered:** After the withdrawal is requested, the amount to be withdrawn is entered, indicated by this signal going high. It triggers the FSM to process the transaction value.
- **cash_eject:** After the withdrawal request and amount are verified, this signal pulses high, showing that the cash is being ejected. This is the output action of the FSM based on previous inputs.
- **clk:** It represents a regular clock signal. The transitions of all control signals are synchronous to the clock edges.
- **balance_check:** As evident from the waveform, this signal becomes high after the "insert card" state, which shows that the user has chosen to check their balance instead of withdrawal.
- **print_balance:** Finally, this signal goes high after the check balance state, for printing the balance slip, a common final step in an ATM session.

The waveform validates the proper sequencing of ATM states: from card insertion and PIN verification to withdrawal and balance checking. All signals behave as expected in a synchronized manner, proving the functional correctness of our controller design.

A. *Testbench*

- **Overview:** This section highlights the scenario in which a user enters an incorrect PIN three consecutive times. According to our design, this results in triggering the freeze timer for 120 seconds. The waveform output confirms this behavior.
- **insert_card:** This signal goes high when the card is inserted. It triggers to begin the authentication process. In our test, this was activated prior to each PIN attempt. So for three attempts, 3 insert cards were triggered.
- **correct_pin:** This remains low throughout the three incorrect attempts because we are checking the freeze functionality. So 3 consecutive incorrect PIN numbers lead to this state.
- **wrong_pin_count:** With each incorrect PIN, this counter increments. After the third wrong attempt, the count reaches three, which triggers the freeze state.
- **freeze_triggered:** As shown in Figure 7, this signal indicates the system has entered the freeze state. Once high, this signal prevents further interaction, disabling PIN input and other necessary transitions for a fixed duration i.e 120 seconds in our design.
- **freeze_timer:** After three failed PIN entries, this timer is activated. The waveform confirms its activation by remaining high for the simulated duration representing 2 minutes. After expiration, normal operation resumes.

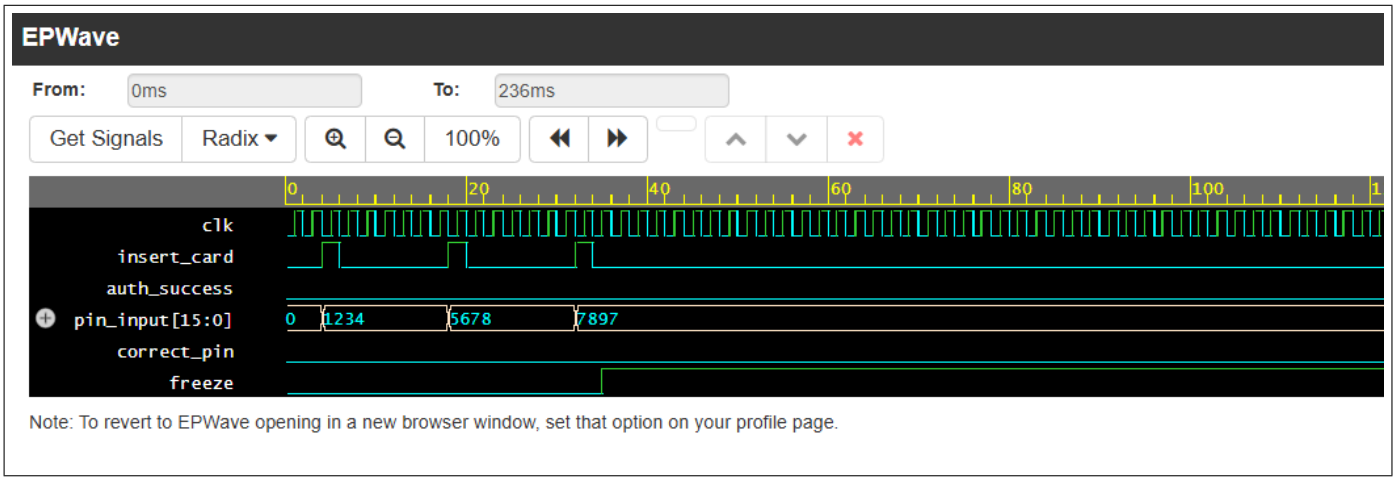

Fig. 7. Freeze Timer Waveform Output

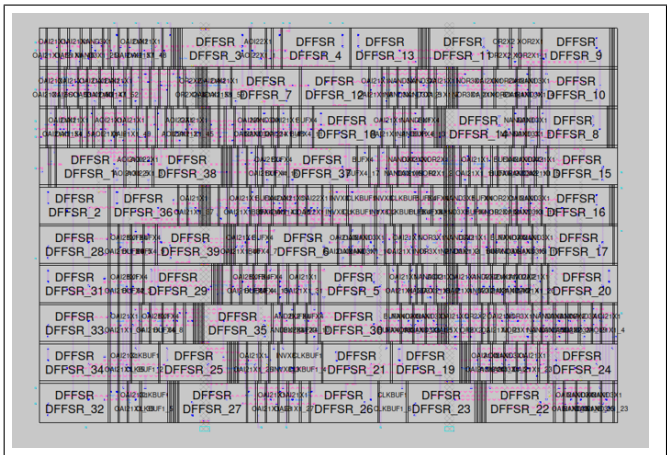

Fig. 8. GDS Layout

VI. GDS Layout using Magic in Qflow

After running all the steps in the Qflow toolchain, we obtained a .gds file in our layout folder. Basically, GDS refers to graphic data stream file which represents the layout of our designed chip in a color coded format. It's main feature is that it ensures that the core logic and functionality of the chip are physically realized.

It displays metal layers, POLY (Polysilicon), Active Areas(diffusion), Contact cuts and other crucial components of the phuyiscal chip. Fig 8: GDS layout image shows these important components of chip and the interconnections between transistors and logic elements.

The blocks present in the layout image, like DFFSR (D Flip-Flop with Set/Reset) and BUF(Buffer) etc. represents the standard cells used in the physical chip of this controller. These cells are pre-designed and optimized for various functions.

VII. Conclusion

In this project, we aimed to design and implement a functional ATM controller, firstly by creating an FSM diagram to understand each states and transitions. Then, using Verilog HDL, we covered the behavioral aspect of chip design. Through this project, our primary goal was to understand and then simulate the operation of an ATM machine, including card insertion, PIN verification, authentication, withdrawal, balance checking and cash ejection. We successfully modeled the functionality using a Verilog Code, verified it through multiple simulation waveforms. Following this validation, the design was synthesized to a visualize a complete ASIC design flow using the Qflow tool, which included crucial steps like synthesis, placement, routing, and layout generation. Each stage was carefully analyzed, ensuring timing correctness through static timing analysis. The final layout met all design constraints, and the observed waveform confirmed accurate behavior, validating the correctness of our ASIC design for the controller. This project provided a practical understanding of digital hardware design, simulation, and backend layout flow in ASIC Design of a chip with a real-world application.

References

[1] P. Divya, A. Lavanya, and T. C. Sekhar, "An ASIC Implementation of Automated Teller Machine Controller for Secured Financial Transactions," *International Journal of Innovative Research in Science, Engineering and Technology*, vol. 6, no. 4, pp. 6010–6015, Apr. 2017.

[2] B. Srivathsan, "Lecture 2: Modeling Code Behaviour," *Model Checking and Systems Verification*, Chennai Mathematical Institute, July–Nov. 2015.