# Design and Simulation of a Automated Teller Machine (ATM) using Verilog

Sujal Manavadariya
*DA-IICT, Gandhinagar*
Rajkot, India
202201084@daiict.ac.in

Sneh Joshi
*DA-IICT, Gandhinagar*
Bhavnagar, India
202201048@daiict.ac.in

***Abstract*—This project presents the design and simulation of a digital ATM system using Verilog HDL. The ATM supports various operations including account authentication, balance inquiry, deposits, withdrawals, PIN changes, and fund transfers. Users input their account number and PIN, select transactions, and specify amounts. The system validates inputs, checks for sufficient balance, and processes transactions while ensuring accuracy and security. A Finite State Machine (FSM) controls all operations, managing states for account verification, transaction execution, and error handling, ensuring smooth and secure transitions. The design is verified through Verilog simulations, accompanied by FSM diagrams, RTL schematics, and QFLOW synthesis. This project demonstrates the practical application of digital design principles in a real-world financial system.**

## I. Introduction

In a world where microcontrollers are embedded in everything from coffee machines to cars, the idea of rethinking even the most familiar systems, like ATMs through a hardware-first lens becomes both relevant and exciting.

We created an FSM-based ATM controller for this project that is independent of external processors and complex software. Rather, it uses just digital logic to manage every task, including PIN verification(with a three-attempt lockout), language selection, PIN change , Withdrawal , Deposit and transaction processing. The architecture avoids live software execution by managing ATM functions entirely in hardware, creating a more secure, quicker, and flexible system based on the concepts of finite state machines.

The workflow for this project is as follows:

- **Finite State Machine Design**: The first stage is to divide the ATM's activities into discrete states, with a particular emphasis on creating an FSM that can handle changes between these states in response to different inputs.
- **RTL code in Verilog:** Next, we implemented the FSM logic using Verilog at the Register-Transfer Level (RTL), defining the logic of how each state and its transitions would be represented and controlled.
- **Process of Verilog code to Physical Chip Layout using Qflow:** Finally, the Verilog code was processed through the Qflow toolchain, for converting RTL into gate level netlist through the process of synthesis, placement, and routing, ultimately producing the chip layout.

## II. Finite State Machine

A Finite State Machine (FSM) is crucial for modeling systems with a limited number of states, allowing for clear state transitions based on inputs. It simplifies complex behaviors, making it ideal for applications like control systems, user interfaces, and protocols. In the context of an ATM system, an FSM is particularly important because it organizes the various stages of the ATM process—such as authentication, balance inquiry, withdrawal, and deposit—into discrete states, ensuring a structured, predictable flow. This approach improves system reliability, simplifies debugging, and enhances user experience by managing transitions smoothly and efficiently.

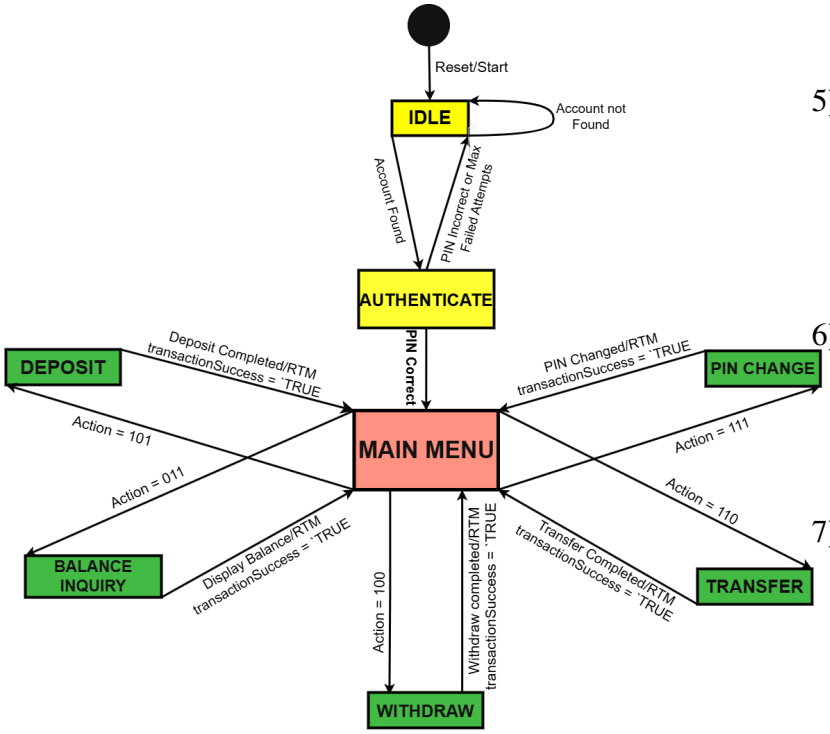


Fig. 1: ATM FSM

This FSM models how an ATM system responds step by step to user inputs. The transitions are based on what the user does.

1) **IDLE**
   - The ATM is inactive, waiting for user input.
   - No transactions are processed during this time.
2) **AUTHENTICATE (PIN Verification)**
   - The user enters their PIN for verification.
   - If PIN is correct, transition to the Main Menu and if PIN is incorrect, transition to the IDLE State.
3) **MAIN MENU**
   - Based on the `Action` input, the system branches into specific functional paths:
     - `Action = 001`: BALANCEINQ
     - `Action = 100`: WITHDRAW
     - `Action = 101`: DEPOSIT
     - `Action = 110`: TRANSFER
     - `Action = 111`: PIN CHANGE
4) **BALANCE INQUIRY**
   - User checks the account balance.
   - After showing the balance, it returns to the Main Menu.
5) **WITHDRAW**
   - Waits for user to enter the amount.
   - User withdraws money from the account.
   - If sufficient balance, withdraw is successful; otherwise, failure is reported and returns to the Main Menu.
6) **DEPOSIT**
   - Waits for user to enter the amount.
   - User deposits money into the account.
   - After the transaction, the system returns to the Main Menu.
7) **TRANSFER**
   - Waits for user to enter the amount and destination account number.
   - Checks if destination account exists and if sufficient balance is available.
   - User transfers money to another account. The system then returns to the Main Menu.
8) **PIN CHANGE**
   - User opts to change their PIN.
   - Waits for user to enter new PIN.
   - After successful change, return to the Main Menu for further actions.

## III. RTL Code in Verilog and Results

The RTL design and simulation waveform are key parts of building digital hardware. Here, we have used Verilog to define the logic and verified how an Automated Teller Machine (ATM) should operate through functional simulation.

```
always @* begin
        nextState = currentState;
        accountFound = `FALSE;
        accIndex = 0;
        destIndex = 4'd15;

        for (i = 0; i < `MAX_ACCOUNTS; i = i + 1) begin
            if (accNumber == acc_database[i])
                begin
                 accIndex = i;
                 accountFound = `TRUE;
            end
        end

        case (currentState)
```

```
            IDLE: if (accountFound) nextState
                = AUTHENTICATE;
                    else nextState = IDLE;

            AUTHENTICATE: begin
                if (pin == pin_database[
                    accIndex] &&
                    failedAttempts[accIndex] <
                     3) begin
                    nextState = MAIN_MENU;
                    failedAttempts[accIndex] =
                         0;
                end else begin
                    if (failedAttempts[
                        accIndex] < 3)
                        failedAttempts[
                            accIndex] =
                            failedAttempts[
                            accIndex] + 1;
                    nextState = IDLE;
                end
            end

            MAIN_MENU: begin
                case (action)
                    3'b011: nextState =
                        BALANCE_INQUIRY;
                    3'b100: nextState =
                        WITHDRAW;
                    3'b101: nextState =
                        DEPOSIT;
                    3'b110: nextState =
                        TRANSFER;
                    3'b111: nextState =
                        CHANGE_PIN;
                    default: nextState =
                        MAIN_MENU;
                endcase
            end

            BALANCE_INQUIRY: begin
                transactionSuccess = `TRUE;
                nextState = MAIN_MENU;
            end

            DEPOSIT: begin
                balance_database[accIndex] =
                    balance_database[accIndex]
                     + amount;
                transactionSuccess = `TRUE;
                nextState = MAIN_MENU;
            end

            WITHDRAW: begin
                if (balance_database[accIndex]
                     >= amount) begin
                    balance_database[accIndex]
                         = balance_database[
                        accIndex] - amount;
                    transactionSuccess = `TRUE
                        ;
                end
                nextState = MAIN_MENU;
            end

            TRANSFER: begin
                for (i = 0; i < `MAX_ACCOUNTS;
                     i = i + 1) begin
                    if (destinationAcc ==
                        acc_database[i])
                        destIndex = i;
                end
                if (destIndex != 4'd15 &&
                    balance_database[accIndex]
                     >= amount) begin
                    balance_database[accIndex]
                         = balance_database[
                        accIndex] - amount;
                    balance_database[destIndex
                        ] = balance_database[
                        destIndex] + amount;
                    transactionSuccess = `TRUE
                        ;
                end
                nextState = MAIN_MENU;
            end

            CHANGE_PIN: begin
                if (pinChange) begin
                    pin_database[accIndex] =
                        newPin;
                    pinSuccess = `TRUE;
                end
                nextState = MAIN_MENU;
            end

            default: nextState = IDLE;
        endcase
    end
endmodule
```

Listing 1: FSM Logic for every states

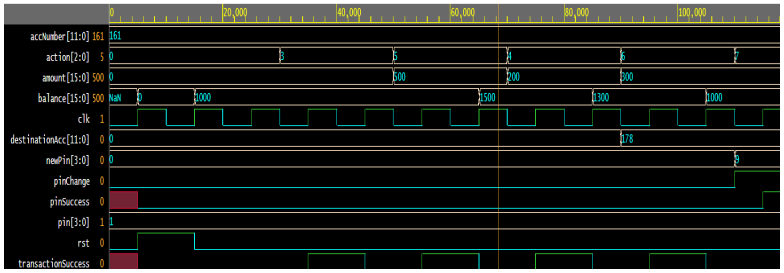


Fig. 2: Waveform for the ATM Module

The figures presented above illustrate the RTL (Register Transfer Level) Verilog code and the corresponding simulation waveform generated using EDA Playground for the ATM Module
These include user interactions such as transfer, PIN change, balance inquiry, deposit, and withdrawal processes, each mapped to a specific FSM state tran-

sition. The waveform validate the correct behavior of the controller under different conditions.

## IV. QFLOW

We utilized Qflow, an open-source digital synthesis toolset, to translate the code into a physical chip architecture after developing and modeling the RTL in Verilog. The entire digital ASIC design process, from synthesis to layout and verification, is automated using Qflow.

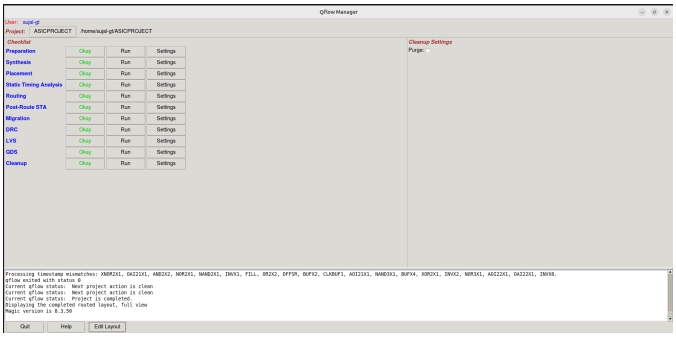


Fig. 3: Successful completion of all stages in the ASIC Design flow

- **Preparation:** Initializes the design setup and ensures all necessary files are available.
- **Synthesis:** Converts RTL code into a gate-level netlist.
- **Placement:** Arranges the logic components on the chip without establishing connections.
- **Static Timing Analysis:** Checks whether signal timings meet design constraints.
- **Routing:** Connects placed gates using metal layers based on the netlist.
- **Post-Route STA:** Re-evaluates timing after routing to ensure accuracy.
- **Migration:** Converts the design into a format compatible with physical layout tools.
- **DRC (Design Rule Check):** Ensures the layout complies with manufacturing design rules.
- **LVS (Layout vs Schematic):** Confirms the layout matches the original schematic/netlist.
- **GDS:** Creates the GDSII layout file used in the chip fabrication process.
- **Cleanup:** Removes temporary files and prepares the project for handoff or re-run.

We used simulation to confirm the ATM Controller's functionality before synthesizing, placing, and routing the design using Qflow. This procedure produced a comprehensive ASIC layout GDSII file that precisely reflected our FSM-based hardware logic and was prepared for production.

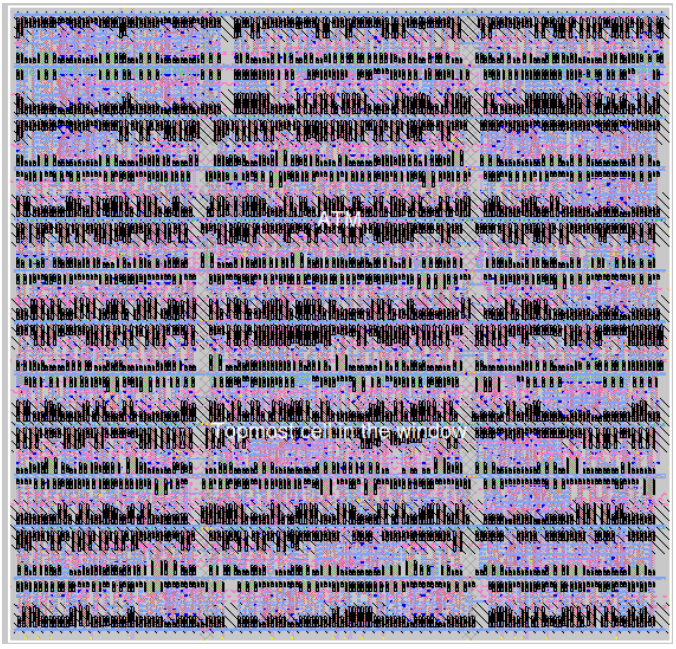

Fig. 4: Synthesized Netlist View of ATM Module

The final ATM Module layout illustrates gate placement and routing, confirming successful synthesis and integration using the Qflow toolchain.

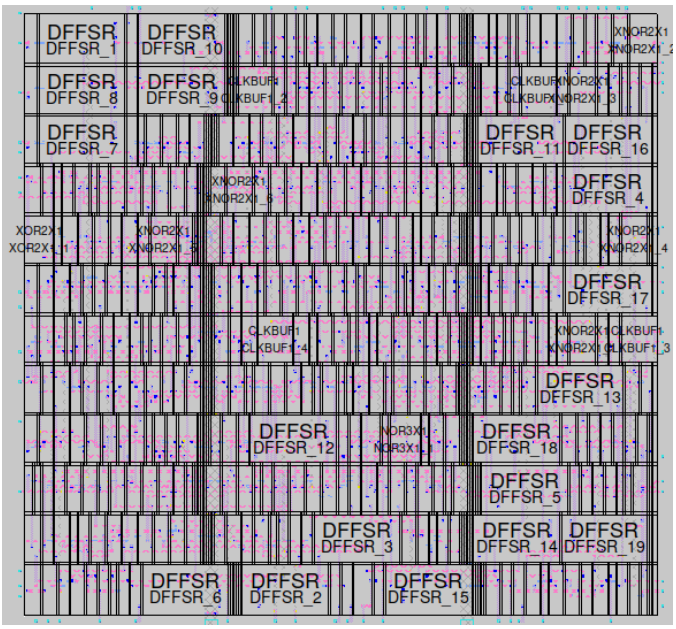


Fig. 5: Final Physical Layout of ATM Module

## V. FUTURE

The current ATM design establishes a reliable functional base; however, several enhancements can be incorporated to expand its capabilities and improve user experience. One significant improvement is the integration of biometric authentication methods—such as fingerprint or facial recognition—to strengthen system security and streamline access. Additionally, replacing traditional input methods with a touchscreen interface can offer a more intuitive and user-friendly interaction.

The system may also be extended to support multi-currency transactions, enabling users to deposit or withdraw in various denominations, thereby increasing its applicability in global contexts. Enhanced change management mechanisms can be implemented to ensure accurate and efficient coin or note dispensing. Furthermore, incorporating power optimization techniques—such as dynamic clock gating or low-power standby modes—can reduce energy consumption, making the design more suitable for embedded or battery-powered deployments. These enhancements would collectively contribute to a more secure, efficient, and adaptable ATM system for real-world applications.

## REFERENCES


[1] Iqbalur Rahman Rokon, Toufiq Rahman, Md. Murtoza Ali Quader, and Mukit Alam. "Hardware Implementation of Watchdog Timer for Application in ATM Machine Using Verilog and FPGA." International Conference on Electronics, Biomedical Engineering and its Applications (ICEBEA'2012), Jan. 7-8, 2012.

[2] Pong P. Chu "RTL Hardware Design Using VHDL, Coding For Efficiency, portability and Scalability", Willy Interscience a john wilay and sons, inc., Publication, pp no:23-160.

[3] Stanley MAZOR and Patricia LINGSTRAAT, "A guide to VHDL (2nd Edition)", copyright 1993, Kluwer Achedamic Publishers, pp no: 1-1 to 7-16.

[4] Qflow: An Open-Source Digital Synthesis Flow, OpenCircuitDesign.com

[5] Magic VLSI Layout Tool, OpenCircuitDesign.com