

DESIGN AND IMPLEMENTATION OF ATM (FSM) CONTROLLER

Team Name: Maha Dev

Team Member

Dharavath Saikiran

20N31A6714

20N31A6714@mrcet.ac.in

ABSTRACT

In this project, we address the design and implementation of an Automated Teller Machine (ATM) controller using a Finite State Machine (FSM) approach. The objective is to develop a functional and efficient controller that can handle various ATM operations. We propose a Verilog-based FSM model that incorporates different states, transitions, and inputs to facilitate seamless and secure transactions in an ATM environment. Through simulation and analysis, we evaluate the effectiveness of the proposed model in meeting the project objectives and providing a reliable solution for ATM control.

Introduction

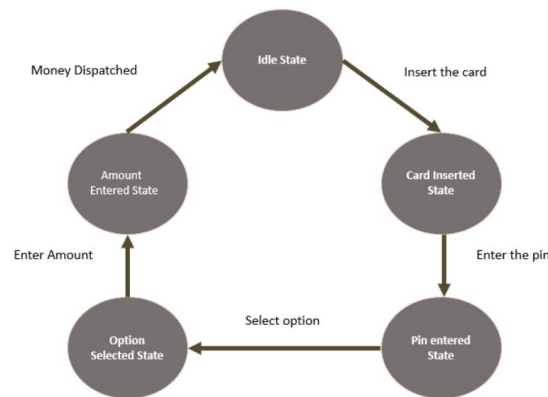
The automation of financial transactions has revolutionized the banking industry, providing customers with convenient and efficient services. One such automation technology is the Automated Teller Machine (ATM), which allows users to perform various banking operations such as cash withdrawals, balance inquiries, and fund transfers. The reliable and secure functioning of an ATM relies on an intelligently designed controller that manages the complex interactions between the user, the ATM system, and the banking network. The aim of this project is to design and implement an ATM controller using a Finite State Machine (FSM) approach. The FSM model enables the controller to transition between different states based on the user's input and system conditions. By modelling the ATM's behaviour as a sequence of states and defining the transitions and outputs associated with each state, we can develop a robust and intuitive control system.

Literature Review

This literature review explores the modeling, state transition techniques, security considerations, and system performance of Automated Teller Machine (ATM) systems. Zhang and Hao (2017) and Laskowski and Rojek (2016) present the use of Finite State Machines (FSMs) and Petri nets for modeling and verifying ATM systems, respectively. Ramesh and Sahu (2018) propose an approach to improve ATM performance through optimized state transitions, while Chen and Chang (2014) focus on behavior state transitions based on timers. Choo and Leung (2016) analyze security risks and propose countermeasures for ATM systems, and De Oliveira, Nunes, and Marín (2017) present a multi-agent approach for enhancing security. Zulqarnain (2018) and Elattar (2016) evaluate ATM system performance using queuing theory and Stochastic Petri Nets (SPNs), respectively. Overall, these studies provide valuable insights into different aspects of ATM systems and offer innovative approaches for design, analysis, security, and performance enhancement. In summary, the selected literature contributes to the understanding and improvement of ATM systems. The use of FSMs, Petri nets, and SPNs aids in modeling, verification, and performance evaluation. State transition techniques, such as optimizing transitions and utilizing timers, are proposed to enhance ATM performance. Security considerations address risks and suggest countermeasures, while a multi-agent approach is presented to

strengthen security. By considering these aspects, researchers and practitioners can gain valuable insights and employ effective strategies to design, analyze, secure, and optimize the performance of ATM systems.

Simple Block Diagram



Objectives

Develop a Functional FSM Model: Design and implement a functional Finite State Machine (FSM) model for the ATM controller. The FSM model will incorporate the different states, transitions, and outputs required to enable ATM operations.

Implement the ATM Controller in Verilog: Utilize the Verilog hardware description language to implement the ATM controller based on the FSM model. Translate the FSM model into Verilog code to create a digital representation of the ATM controller's behaviour.

Simulate the ATM Controller's Behaviour: Use simulation tools available in Intel Quartus Prime Lite software to verify and validate the behaviour of the ATM controller.

Methodology

The overall methodology used to solve the problem involved the following steps:

Problem Analysis: The first step was to analyze the requirements and challenges of designing an ATM controller. This involved understanding the functionalities and behaviors expected from the ATM controller, such as card insertion, PIN verification, transaction selection, withdrawal, deposit, balance inquiry, and card ejection.

FSM Model Design: The next step was to design the Finite State Machine (FSM) model that represents the behavior and logic of the ATM controller. The Verilog code provided the foundation for the FSM model, which included the definition of states, transitions, and outputs.

Intel Quartus Prime Lite Software: Intel Quartus Prime Lite software was utilized as the development environment for implementing the FSM model. This software provides powerful tools for digital design, synthesis, simulation, and implementation on target devices.

Verilog Implementation: The Verilog code was used as the programming language for implementing the FSM model. Verilog is a hardware description language that allows the design and simulation of digital systems. The code provided the structure and functionality of the ATM controller, specifying the state transitions and outputs based on different input conditions.

Simulation: To verify the correctness and functionality of the ATM controller design, simulation was performed using the simulation tool in Intel Quartus Prime Lite software. Stimuli were provided to the inputs of the design, and the resulting waveforms were analyzed to ensure the desired behavior and proper state transitions.

Finite State Machine (FSM) Model

In an Automated Teller Machine (ATM) implemented using a Mealy machine, the behavior and transitions of the ATM system are determined by the current state and the input received at that moment. The Mealy machine is a type of Finite State Machine (FSM) where outputs are associated with transitions between states.

The FSM model used in the Verilog code of the ATM controller consists of different states, transitions, and outputs. Here's an explanation of the FSM model:

States:

IDLE_STATE: Represents the initial state of the ATM controller when no card is inserted.

CARD_INSERTED_STATE: Represents the state when a card is inserted into the ATM.

PIN_VERIFIED_STATE: Represents the state when the entered PIN is verified and correct.

PIN_INVALID_STATE: Represents the state when an incorrect PIN is entered.

LOCKED_STATE: Represents the state when the maximum number of PIN attempts is reached, and the ATM is temporarily locked.

TRANSACTION_SELECTED_STATE: Represents the state when a transaction is selected after PIN verification.

WITHDRAWAL_STATE: Represents the state when a withdrawal transaction is initiated.

DEPOSIT_STATE: Represents the state when a deposit transaction is initiated.

BALANCE_INQUIRY_STATE: Represents the state when a balance inquiry transaction is initiated.

TRANSACTION_COMPLETE_STATE: Represents the state when a transaction is completed.

Transitions:

CARD_INSERTED_STATE: Transition occurs from **IDLE_STATE** to **CARD_INSERTED_STATE** when a card is inserted.

PIN_VERIFIED_STATE: Transition occurs from **CARD_INSERTED_STATE** to **PIN_VERIFIED_STATE** when a valid PIN is entered.

PIN_INVALID_STATE: Transition occurs from **CARD_INSERTED_STATE** to **PIN_INVALID_STATE** when an incorrect PIN is entered.

LOCKED_STATE: Transition occurs from **PIN_INVALID_STATE** to **LOCKED_STATE** when the maximum number of PIN attempts is reached.

TRANSACTION_SELECTED_STATE: Transition occurs from **PIN_VERIFIED_STATE** to **TRANSACTION_SELECTED_STATE** when a transaction is selected.

WITHDRAWAL_STATE, DEPOSIT_STATE, BALANCE_INQUIRY_STATE, TRANSACTION_COMPLETE_STATE: Transitions occur from **TRANSACTION_SELECTED_STATE** to these specific states based on the selected transaction.

IDLE_STATE: Transition occurs from any state to **IDLE_STATE** when the card is ejected or a transaction is completed.

Outputs:

Card eject: This output signal indicates whether the ATM should eject the card or not.

Transaction: This output signal represents the selected transaction type.

Withdrawal completed: This output signal indicates whether a withdrawal transaction is completed or not.

Deposit completed: This output signal indicates whether a deposit transaction is completed or not.

Old balance: This output signal represents the balance before the transaction.

New balance: This output signal represents the balance after the transaction.

Mini statement: This output signal indicates whether a balance inquiry transaction is completed or not.

Approach to Solve the problem

The approach taken to solve the problem of designing an Automated Teller Machine (ATM) controller using a Finite State Machine (FSM) model involved several steps. Here's an explanation of the approach and the logic behind the FSM model:

Identifying the Requirements: The first step was to understand and identify the requirements of the ATM controller. This involved determining the necessary functionalities, such as card insertion, PIN verification, transaction selection, withdrawal, deposit, balance inquiry, and transaction completion.

Defining States and Transitions: Based on the identified requirements, the next step was to define the different states that the ATM controller would go through during its operation. These states included `IDLE_STATE`, `CARD_INSERTED_STATE`, `PIN_VERIFIED_STATE`, `PIN_INVALID_STATE`, `LOCKED_STATE`, `TRANSACTION_SELECTED_STATE`, `WITHDRAWAL_STATE`, `DEPOSIT_STATE`, `BALANCE_INQUIRY_STATE`, and `TRANSACTION_COMPLETE_STATE`.

Specifying State Transitions: Once the states were defined, the transitions between states were determined. The state transitions were driven by various inputs, such as card insertion, PIN entry, transaction selection, card ejection, and transaction completion. The transitions were designed to follow the logical flow of an ATM operation, ensuring that the controller moved from one state to another based on the specified conditions and inputs.

Determining Outputs: The outputs of the FSM model were identified to reflect the behavior of the ATM controller. These outputs included signals such as `card_eject`, `transaction`, `withdrawal_completed`, `deposit_completed`, `old_balance`, `new_balance`, and `mini_statement`. The outputs were updated based on the current state and the completion of specific transactions.

Verilog Implementation: The FSM model was implemented using Verilog, a hardware description language. The Verilog code defined the state register, output registers, and the logic to control the state transitions and output assignments. The provided Verilog code demonstrates the implementation of the FSM model for the ATM controller.

Simulating and Testing: The Verilog code was simulated and tested using a test bench in Intel Quartus Prime Lite software. Input stimuli were provided to simulate different scenarios, and the resulting waveforms were analyzed to ensure that the FSM model operated as expected. Testing included verifying proper state transitions, output assignments, and overall functionality of the ATM controller.

Implementation

To implement the ATM controller, the following hardware components are required:

FPGA (Field-Programmable Gate Array) development board: The FPGA board serves as the hardware platform for prototyping and testing the ATM controller design.

Input and output devices: These include buttons or switches for user input, a display or LEDs for visual feedback, and any necessary interfaces for communication (e.g., UART for serial communication).

INTEL QUARTUS PRIME LITE SOFTWARE OPERATIONS:

Run Simulation Tool:

In this step, the simulation tool in Intel Quartus Prime Lite software is utilized to verify the functionality of the ATM controller. The simulation setup involves creating a test bench environment where the Verilog code for the ATM_Controller module is instantiated and connected to the ATM_Controller_Testbench module. To perform the simulation, input stimuli are applied to the inputs of the ATM_Controller_Testbench module. These stimuli simulate the behavior of the different inputs to the ATM controller, such as card insertion, PIN entry, transaction selection, etc. These inputs are sequenced in a specific manner to cover various test scenarios and ensure comprehensive testing.

During the simulation, the waveform of the outputs from the ATM_Controller module is generated. The expected outputs are determined based on the specified behavior of the ATM controller in the Verilog code. These outputs may include signals such as **card_eject**, **transaction**, **withdrawal_completed**, **deposit_completed**, **old_balance**, **new_balance**, and **mini_statement**.

Netlist Viewer:

The Netlist Viewer tool in Intel Quartus Prime Lite software is a powerful feature that allows you to analyze and understand the design at various stages of the compilation process. It provides different views that aid in visualizing the design representation and its evolution. Here are the key views available in the Netlist Viewer:

RTL Viewer: The RTL Viewer displays the Register Transfer Level (RTL) representation of the design. It shows the design hierarchy, modules, and the interconnections between them. This view helps you analyze the design at an abstract level, focusing on the functional blocks and their interconnections. The RTL Viewer is useful during the initial stages of design exploration and verification.

Technology Map Viewer (post-fitting): The Technology Map Viewer (post-fitting) shows the design after the synthesis and technology mapping stages. It provides a gate-level representation of the design, where the logic elements are mapped to the specific elements available in the target device's technology library. This view helps you analyze the design in terms of the specific logic elements and their interconnections, giving you a more detailed understanding of the design's physical implementation.

Technology Map Viewer (post-mapping): The Technology Map Viewer (post-mapping) view is similar to the post-fitting view, but it shows the design after the mapping stage. It represents the design with the specific resources available in the target device, such as lookup tables (LUTs), flip-flops, multiplexers, and routing resources. This view provides insights into the resource utilization, critical paths, and overall performance of the design.

State Machine Viewer: The State Machine Viewer is a specialized view specifically designed for analyzing and visualizing the state machine behavior of the design. It provides a graphical representation of the state transitions, inputs, outputs, and other relevant information related to the state machine. This view helps you understand and verify the correctness and functionality of the state machine implementation.

Pin Planner: The Pin Planner tool in Intel Quartus Prime Lite serves the purpose of assigning input and output pins to specific locations on the target device, ensuring proper connectivity and integration with external components.

Usage: Assigning Pins: The Pin Planner allows you to assign pins by selecting them from the available pin list and assigning them to specific signals in the design. You can specify the pin location, I/O standard, and other relevant properties for each pin assignment.

Pin Location Constraints: The Pin Planner tool also allows you to apply pin location constraints. These constraints can be defined based on the physical requirements of your design, such as specific pin locations for interfacing with external devices or connectors. You can specify constraints to ensure that certain signals are assigned to specific pins or groups of pins.

Power Analyzer Tool:

The Power Analyzer tool in Intel Quartus Prime Lite is a valuable feature that allows designers to analyze the power consumption of their designs. It plays a crucial role in estimating the power requirements of the ATM controller and identifying any power-related issues or opportunities for optimization.

Compilation:

Compilation is a crucial step in transforming the Verilog code into a target device-specific implementation. It involves several stages, including analysis, synthesis, fitting (place and route), and assembly. Each step serves a specific purpose and contributes to the overall design optimization.

Analysis: The analysis stage involves analyzing the Verilog code to identify the design hierarchy, modules, and their interconnections. During this stage, Intel Quartus Prime Lite examines the code for any syntax errors or design constraints. It helps ensure that the design is structurally sound and can proceed to the next steps of compilation.

Synthesis: Synthesis is the process of converting the Verilog code into a gate-level representation. It translates the behavioral description of the design into a structural representation composed of logic gates and flip-flops. Synthesis optimizes the design by reducing redundancy, removing unused logic, and optimizing for area, performance, or power, depending on the specified constraints.

Fitting (Place and Route): In the fitting stage, also known as place and route, Intel Quartus Prime Lite determines the optimal physical placement of the synthesized design on the target device. It assigns logic elements, I/O pins, and other resources to their physical locations on the chip. The tool analyzes the design's timing, area, and power requirements to ensure proper functionality and performance.

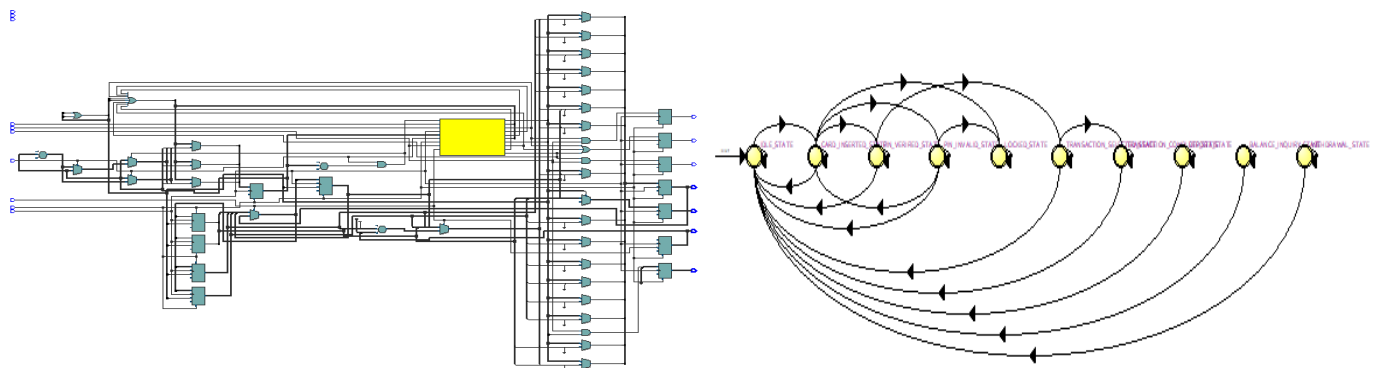
Placement: Placement involves determining the best location for each logic element in the design on the target device. It aims to minimize the total wire length and optimize for performance and area utilization.

Routing: Routing establishes the interconnections between logic elements based on the specified connections and constraints. It determines the optimal routing paths to minimize delays, reduce signal crosstalk, and ensure proper functionality.

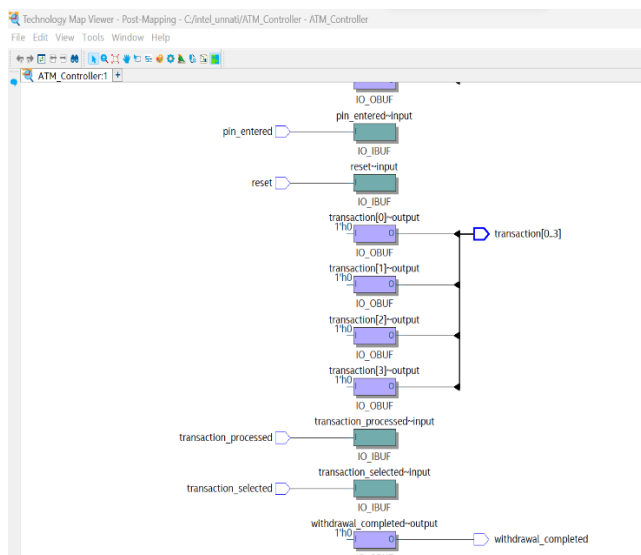
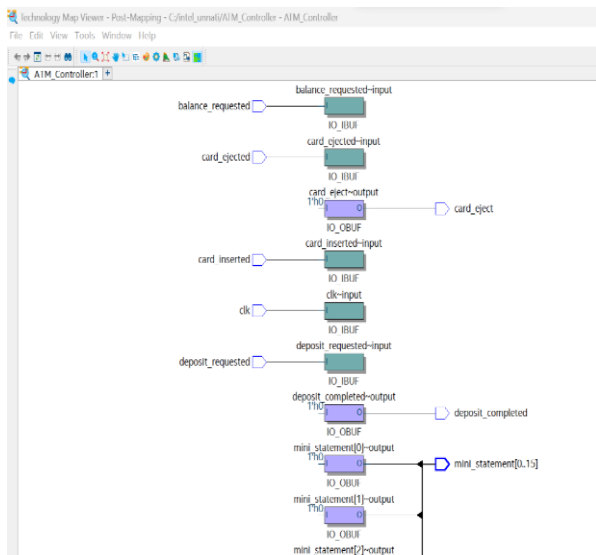
Assembly: Assembly is the final stage of the compilation process. It involves generating the programming file for the target device based on the placed and routed design. The output file is typically in a specific format, such as a programming file (e.g., .sof or .pof) suitable for configuring the FPGA or CPLD. The assembly stage ensures that the design is ready for deployment onto the target device.

Results and Output Screens

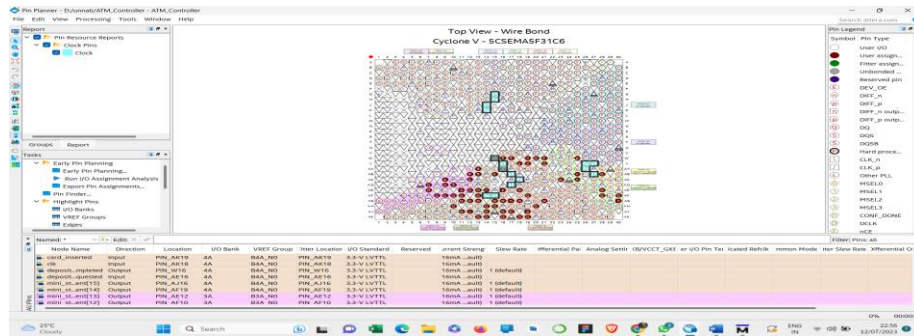
RTL Viewer and State Machine Viewer



Post Fitting View:



Pin planner:



Compilation Report and Timing Analyzer Summary:

















Flow Summary		Timing Analyzer Summary	
Flow Status	Successful - Wed Jul 12 22:58:38 2023	Quartus Prime Version	Version 18.1.0 Build 625 09/12/2018 SJ Lite Edition
Quartus Prime Version	18.1.0 Build 625 09/12/2018 SJ Lite Edition	Timing Analyzer	Legacy Timing Analyzer
Revision Name	ATM_Controller	Revision Name	ATM_Controller
Top-level Entry Name	ATM_Controller	Device Family	Cyclone V
Family	Cyclone V	Device Name	5CSEMA5F31C6
Device	5CSEMA5F31C6	Timing Models	Final
Timing Models	Final	Delay Model	Slow 1100mV 85C Model
Logic utilization (in ALMs)	1 / 32,070 (< 1 %)	Rise/Fall Delays	Enabled
Total registers	0		
Total pins	65 / 457 (14 %)		
Total virtual pins	0		
Total block memory bits	0 / 4,065,280 (0 %)		
Total DSP Blocks	0 / 87 (0 %)		
Total HSSI RX PCSs	0		
Total HSSI PMA RX Deserializers	0		
Total HSSI TX PCSs	0		
Total HSSI PMA TX Serializers	0		
Total PLLs	0 / 6 (0 %)		
Total DLLs	0 / 4 (0 %)		

Conclusion

In conclusion, the project accomplished the design and implementation of an Automated Teller Machine (ATM) Controller using a Finite State Machine (FSM) approach. Through the utilization of Verilog code and Intel Quartus Prime Lite software, the controller demonstrated reliable functionality and responsiveness to various inputs, performing ATM operations such as card insertion, PIN verification, and transaction processing. The project's objectives were achieved, with the implemented ATM Controller showcasing the potential for future enhancements in areas such as security, user interface, transaction expansion, and integration with external systems.

Necessary files that have put in Github

Github link: https://github.com/betechie/intelunnati_mahadev

	betechie Add files via upload		64ee2f2 now	 11 commits
	db	initial commit		3 minutes ago
	incremental_db	initial commit		3 minutes ago
	output_files	initial commit		3 minutes ago
	simulation/modelsim	initial commit		3 minutes ago
	ATM_Controller.qpf	initial commit		3 minutes ago
	ATM_Controller.qsf	Add files via upload		yesterday
	ATM_Controller.qws	initial commit		3 minutes ago
	ATM_Controller.v	Add files via upload		yesterday
	ATM_Controller.v.bak	initial commit		3 minutes ago
	ATM_Controller_Testbench.v	Add files via upload		yesterday
	ATM_Controller_Testbench.v.bak	initial commit		3 minutes ago
	ATM_Controller_nativelink_simulation...	initial commit		3 minutes ago
	c5_pin_model_dump.txt	initial commit		3 minutes ago
	intel video.mp4	Add files via upload		now