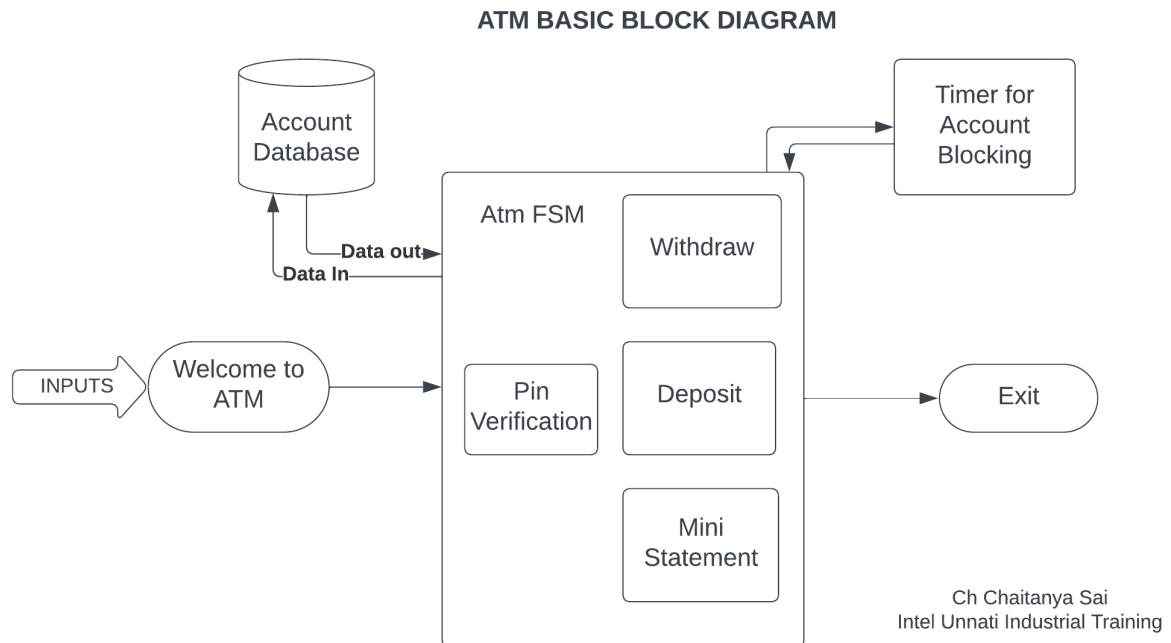# Design and Implementation of Automated Teller Machine (FSM) Controller

SUBMITTED BY

Ch Chaitanya Sai

DEPARTMENT OF ELECTRONICS AND COMMUNICATIONS ENGINEERING

MANIPAL INSTITUTE OF TECHNOLOGY

UDUPI-KARKALA ROAD, ESHWAR NAGAR, MANIPAL, KARNATAKA,576104

Block Diagram:

ATM BASIC BLOCK DIAGRAM
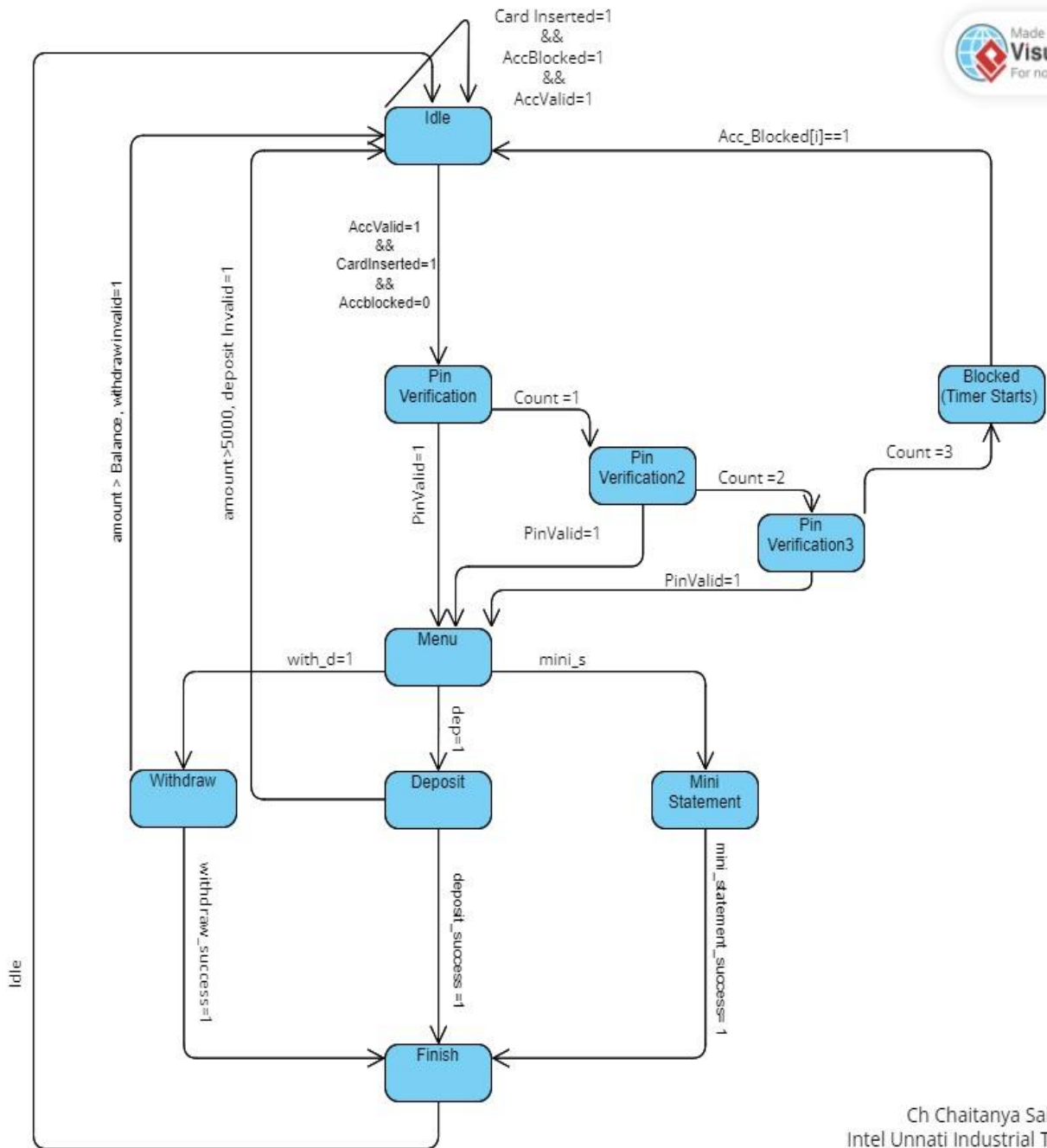
Ch Chaitanya Sai
Intel Unnati Industrial Training

A Basic Model of an ATM was selected for ease of implementation and simulation. The model consists of 5 core ATM functions:

- Pin Verification
- Withdrawal
- Deposit
- Mini Statement
- Account Blocking

The behaviour and operation between these modes are represented by the FSM. FSM is used to describe the sequential logic and transitions of a system.

FINITE STATE MACHINE:

Ch Chaitanya Sai
Intel Unnati Industrial Training

States:

- **Idle**: Initial State no input given.
- **Pin Verification**: Verifies the pin 1st time.
- **Pin Verification2**: Verifies the pin 2nd time.
- **Pin Verification3**: Verifies the pin 3rd time.

- **Menu**: Give the user an option to select either if the three operations withdraw, deposit, Mini statement.

- **Deposit**: amount value is entered which is converted to the actual value and then is then deposited in the account.

- **Withdraw**: amount value is entered which is converted to the actual value and then is then withdrawn from the account.

- **Mini_ Statement**: In this mode it shows 4 recent transactions.

- **Finish**: completed

- **Blocked**: If a pin is given incorrectly 3 times, then the account will be block for 24 hrs.

Transitions:

- **Card Inserted**: gives a signal '1' if a card is inserted.

- **AccValid:** Tells us of the account validation, if the account number is a part of the database or not.

- **Acc Blocked**: gives us the information if the account is blocked or not.

- **PinValid**: If the given input pin is provided matches with the pin in the database corresponding to that account this will become '1'.

- **Count**: can be incremented up to 3, at each increment it goes to Pin verification2 and Pin Verificaiton3 respectively.

- **With_d, dep, mini_s**: signals when triggered moves to withdraw, deposit and mini statement states respectively.

- **Withdraw_success, Deposit_success, mini_statement_success:** these signals trigger when transactions happen successfully.

The **inputs and outputs** can be identified as follows:

Inputs:

1. accNumber: A 2-bit input representing the account number entered by the user.

2. Pin: A 2-bit input representing the PIN entered by the user.

3. card_inserted: A single-bit input indicating whether a card is inserted into the ATM.

4. enter: A single-bit input indicating the user's confirmation or selection of an option.

5. with_d: A single-bit input indicating the user's selection of the Withdraw option.

6. dep: A single-bit input indicating the user's selection of the Deposit option.

7. mini_s: A single-bit input indicating the user's selection of the Mini Statement option.

8. clk: The clock input for the system.

9. reset: A single-bit input for resetting the system.

10. amount_in: A 4-bit input representing the amount entered by the user for withdrawal or deposit.

Outputs:

1. balance: A 16-bit output representing the current balance of the selected account.

2. accountfound: A single-bit output indicating whether the entered account number is found in the database.

3. pinfound: A single-bit output indicating whether the entered PIN is correct.

4. clk_out: A single-bit output representing the divided clock signal for the system.

ASSUMPTIONS:

Considering a simple atm model keeping in mind that we will be implementing it on a DE1 SoC which is a cyclone 5 FPGA (more info about cyclone 5). We be using 10 switches and 4 push buttons and 10 LEDs that this board provides as input and outputs.

Amount and can be input in a denomination of 500 i.e., amount is a 4-bit number which is multiplied with 500, this amount is the actual amount we will be depositing or do the withdrawal. Amount input will be 4 pins.

We have also assumed the account number and pin to be 2 bit each. Since we will be giving the inputs via the switches of the FPGA which takes 2 pin each.

If a wrong pin is entered 3 times the account will be blocked for 24hrs

If amount given as an input is more than a specified amount the ATM won't allow the user to withdraw the amount and goes back to idle state

Made an account database using an Initial block which checks for the corresponding input when an account number and pin is given at the input. This account database also contains a count and account blocked registers corresponding to each account.

For the mini statement withdrawal and deposit transactions are stored in separate register arrays and printed together while simulation. Mini statement shows up to 4 recent transactions.

Clock divider is used to bring down the native clock of 50MHz to 1Hz i.e. 20 ns to 1 sec (for simulation purposes we have taken the native clock frequency).

Approach:

In an ATM system, the outputs often depend on the current state as well as the inputs. For example, the balance output depends on the state of the account and the transactions performed. By using a Mealy implementation, we can directly associate the outputs with the relevant states and inputs, resulting in a more intuitive and straightforward design.

The outputs in the code (balance, accountfound, pinfound, clk_out) are directly dependent on both the current state and the inputs. For example, the balance output is updated based on the state Withdraw, Deposit, and Finish where the balance is modified. Similarly, accountfound and pinfound outputs are determined by the Acc_valid and Pin_valid variables, which are dependent on the current state and input conditions.

Immediate Output Updates: The outputs in the code are updated immediately when specific conditions are met. For instance, in the **Withdraw** state, the **balance** output is updated when a withdrawal is successful. Likewise, the **accountfound** and **pinfound** outputs are updated based on the **Acc_valid** and **Pin_valid** variables, respectively, which change with specific input and state conditions.

Next we analyze and validate the functionality of the ATM module.

1. Idle: This is the initial state of the ATM. It transitions to the Pin_Verification state when a card is inserted and **Acc_valid** is true.

2. Pin_Verification: In this state, the system verifies the PIN entered by the user. It transitions to different states based on the number of incorrect attempts (**count[AccIndex]**).

   If the PIN is correct, it transitions to the Menu state.

   If the first attempt is incorrect, it transitions to Pin_Verification2 state.

   If the second attempt is incorrect, it transitions to Pin_Verification3 state.

   If the third attempt is incorrect, the account is blocked, and it transitions to the Blocked state.

3. Blocked: This state represents the blocked state of an account. The system remains in this state until the block duration (**block_duration**) is reached, at which point it transitions back to the Idle state.

4. Pin_Verification2 and Pin_Verification3: These states handle the second and third attempts at PIN verification, similar to Pin_Verification state. If the PIN is correct, it transitions to the Menu state.

Otherwise, it increments the **count[AccIndex]** and transitions to the next PIN verification state or Blocked state if the maximum attempts are reached.

5. Menu: Once the PIN is verified, the system enters the Menu state, where it waits for the user to select an operation (Withdraw, Deposit, or Mini_Statement). The transitions to the respective states occur based on the **with_d**, **dep**, and **mini_s** inputs.

6. Withdraw: In this state, the system prompts the user to enter the withdrawal amount (**amount_in**). If the entered amount is valid and within the account balance, it updates the balance and transitions to the Finish state. Otherwise, it transitions back to the Idle state or remains in the Withdraw state if the amount is invalid.

7. Deposit: Similar to Withdraw state, this state handles the deposit operation. If the deposit amount is valid, it updates the balance and transitions to the Finish state. Otherwise, it transitions back to the Idle state.

8. Mini_Statement: This state generates a mini-statement displaying recent transaction details. It transitions to the Finish state after generating the mini-statement.

9. Finish: This state is reached after completing a transaction or generating a mini-statement. It transitions back to the Idle state, indicating that the ATM is ready for the next transaction.

10. Blocked Timer: The code includes timers (**timer[1:3]**) to keep track of the blocked duration for individual accounts. The timers are incremented when an account is blocked and reset when the block duration is reached.

After Analyzing and checking the functionanaltiy of the FSM it is necessary to cross-reference the code with the expected behavior of an ATM system. Here's a breakdown of the outputs and their expected behavior:

1. **balance**: This output represents the account balance. It should be updated based on successful withdrawal or deposit operations and remain unchanged otherwise. We cross-reference the code to verify that **balance** is correctly updated in the Withdraw and Deposit states, reflecting the correct balance after each transaction.

2. **accountfound**: This output indicates whether the entered account number (**accNumber**) is found in the account database. It should be set to 1 when the account is found and 0 otherwise. We check the code to confirm that **accountfound** is set to 1 in the Pin_Verification state when **Acc_valid** is true, indicating a valid account.

3. **pinfound**: This output indicates whether the entered PIN (**Pin**) matches the PIN associated with the account. It should be set to 1 when the PIN is correct and 0 otherwise. We must review the code to ensure that **pinfound** is set to 1 in the Pin_Verification states (**Pin_Verification**, **Pin_Verification2**, **Pin_Verification3**) when **Pin_valid** is true, indicating a correct PIN.

4. **clk_out**: This output represents the clock signal generated by the clock divider module (**clk_div**). Verify that **clk_out** is correctly assigned the value of **clk_div.clk_out** in the code.

To validate these outputs, we will simulate the code using Quartus Pime Questa Sim, we will also provide input stimulus that covers different scenarios and observe the output values. Verify that the outputs match the expected behavior as described above.

Testbench module for the ATM system includes multiple test cases to simulate different scenarios and verify the functionality of the ATM module. Here's a brief description of the test cases:

1. Test Case 1: PIN Verification (correct at 1st try) and deposit:

    - Insert card, enter account number, enter PIN, and select deposit.

    - Verify if the deposit operation is performed correctly.

2. Test Case 2: PIN Verification (correct at 2nd try) and deposit:

    - Insert card, enter account number, enter incorrect PIN, re-enter PIN, and select deposit.

    - Verify if the deposit operation is performed correctly.

3. Test Case 3: PIN Verification (correct at 3rd try) and deposit:

    - Insert card, enter account number, enter incorrect PIN multiple times, and finally enter the correct PIN.

    - Verify if the deposit operation is performed correctly.

4. Test Case 4: Deposit:

    - Insert card, enter account number, enter PIN, and select deposit.

    - Verify if the deposit operation is performed correctly.

5. Test Case 5: Mini Statement:

    - Insert card, enter account number, enter PIN, and select mini statement.

    - Verify if the mini statement operation is performed correctly.

6. Test Case 6: Making another deposit to check if the mini statement is functioning as intended:

   - Insert card, enter account number, enter PIN, and make a deposit.

   - Verify if the mini statement operation displays the updated transaction.

7. Test Case 7: Mini Statement:

   - Insert card, enter account number, enter PIN, and select mini statement.

   - Verify if the mini statement operation is performed correctly.

8. Test Case 8: Withdrawal 1:

   - Insert card, enter account number, enter PIN, and select withdrawal.

   - Verify if the withdrawal operation is performed correctly.

9. Test Case 9: Withdrawal 2:

   - Insert card, enter account number, enter PIN, and select withdrawal.

   - Verify if the withdrawal operation is performed correctly.

10. Test Case 10: Withdrawal 3:

    - Insert card, enter account number, enter PIN, and select withdrawal.

    - Verify if the withdrawal operation is performed correctly.

11. Test Case 11: Withdrawal 4:

    - Insert card, enter account number, enter PIN, and select withdrawal.

    - Verify if the withdrawal operation is performed correctly.

12. Test Case 12: Mini Statement for withdrawal:

    - Insert card, enter account number, enter PIN, and select mini statement.

    - Verify if the mini statement operation displays the updated transaction after a withdrawal.

13. Test Case 13: Withdrawal 5:

    - Insert card, enter account number, enter PIN, and select withdrawal.

    - Verify if the withdrawal operation is performed correctly.

14. Test Case 14: Mini Statement for withdrawal:

- Insert card, enter account number, enter PIN, and select mini statement.

- Verify if the mini statement operation displays the updated transaction after a withdrawal.

15. Test Case 15: If the deposit amount exceeds 10,000/-:

- Insert card, enter account number, enter PIN, and select deposit with a large amount.

- Verify if the ATM handles the exceeded deposit limit correctly.

16. Test Case 16: If the withdrawal amount exceeds the bank account balance:

- Insert card, enter account number, enter PIN, and select withdrawal with an amount greater than the account balance.

- Verify if the ATM handles the insufficient balance scenario correctly.

17. Test Case 17: Blocking account:

- Insert card, enter account number, enter an incorrect PIN multiple times, and block the account.

- Verify if the ATM blocks the account after multiple failed PIN attempts.

18. Test Case 18: Checking if the account is blocked or not while the timer is running:

- Insert card, enter account number, and check if the account is blocked or not during the timer duration.

- Verify if the ATM handles the account status correctly while the timer is running.

19. Test Case 19: Checking if the account is blocked or not while the timer is running:

- Insert card, enter account number, and check if the account is blocked or not during the timer duration.

- Verify if the ATM handles the account status correctly while the timer is running.

20. Test Case 20: Checking if the account is blocked or not while the timer is running:

- Insert card, enter account number, and check if the account is blocked or not during the timer duration.

- Verify if the ATM handles the account status correctly while the timer is running.

These test cases cover a range of scenarios to ensure the ATM module behaves as expected and produces the correct output for different inputs and actions

Complete Flow of the ATM FSM Code

1. The code starts by initializing the account database, which includes account numbers, PINs, counts, balances, and blocking status. Four accounts are created with their respective details.

2. The main FSM starts executing when the slower clock (clk_out) or the reset signal triggers. Initially, the state is set to Idle.

3. In the Idle state, the code waits for the card_inserted signal and a valid account number (accNumber) to be entered. If a card is inserted and a valid account number is provided, the FSM transitions to the Pin_Verification state.

4. In the Pin_Verification state, the code checks if the account corresponding to the entered account number is blocked. If it is blocked, the FSM transitions back to the Idle state. If not blocked, the code waits for the PIN (Pin) to be entered.

5. Once the PIN is entered (Pin != 2'b00) and the enter signal is high, the code verifies the entered PIN with the stored PIN for the account. If the entered PIN is correct, the Pin_valid signal is set to 1, and the FSM transitions to the Menu state. If the PIN is incorrect, the code increments the count for the account and transitions to Pin_Verification2 state for the second attempt.

6. In Pin_Verification2 state, the code follows a similar procedure to verify the PIN, but this time it checks for the second attempt. If the PIN is correct, the FSM transitions to the Menu state. If not, it transitions to Pin_Verification3 state for the third attempt.

7. In Pin_Verification3 state, the code checks the entered PIN for the third time. If the PIN is correct, the FSM transitions to the Menu state. If the PIN is incorrect, the account is blocked by setting Acc_Blocked to 1, and the FSM transitions to the Blocked state. The corresponding timer (timer) starts counting.

8. In the Blocked state, the code checks if the account is still blocked. If it is blocked, the system displays a message stating that the account is blocked, and the FSM transitions back to the Idle state.

9. If the PIN is correct and the FSM transitions to the Menu state, the code checks the input signals with_d, dep, and mini_s to determine the user's selected option: Withdrawal, Deposit, or Mini Statement, respectively.

10. In the Withdraw state, the code waits for the user to enter an amount (amount_in) and press the enter signal. Once an amount is entered, it checks if the requested amount is higher than the account balance. If the balance is insufficient, the withdraw_INVALID signal is set to 1, and the FSM transitions back to the Idle state. Otherwise, the withdrawal is considered valid, and the FSM transitions to the Finish

state after updating the account balance and setting the withdraw_success signal to 1. The withdrawal transaction details are stored in the transaction_withdraw and amount_withdraw arrays.

11. In the Deposit state, the code follows a similar procedure as the Withdraw state but for depositing an amount. It checks if the entered amount exceeds the specified limit. If it does, the deposit_INVALID signal is set to 1, and the FSM transitions back to the Idle state. Otherwise, the deposit is considered valid, and the FSM transitions to the Finish state after updating the account balance and setting the deposit_success signal to 1. The deposit transaction details are stored in the transaction_deposit and amount_deposit arrays.

12. In the Mini_Statement state, the code generates the mini statement by displaying the recent transaction details stored in the transaction_withdraw and amount_withdraw arrays for withdrawals and the transaction_deposit and amount_deposit arrays for deposits.

13. After completing any transaction or generating the mini statement, the FSM transitions to the Finish state. In the Finish state, a message is displayed thanking the user for using the ATM services, and the FSM transitions back to the Idle state.

14. The code also includes logic for a timer (timer) that counts the elapsed time in the Blocked state. Once the timer reaches the specified block duration, the account is unblocked automatically.

15. The output signals accountfound and pinfound indicate whether an account and PIN are valid, respectively.

Files That Have been Uploaded in the GITHUB

- **Code**
  (1) atm.v
  (2) atm_tb.v
  (3) Clock_divider.v
  (4) atm.out.sdc
- **Netlist Viewers**
  (1) RTL Viewer
  (2) Technology map viewer post mapping
  (3) Technology map viewer post Fiting
- **Docs**
  (1) atm-Analysis and Synthesis Report.pdf
  (2) atm-Fitter Report.pdf

(3) atm-Power Analyzer Report.pdf

(4) atm-Timing Analyzer Report.pdf

(5) atm_Pin_Planning.csv

(6) Tb_Simulation_Result.txt

(7) Copy of this report

- **Model**

(1) Fsm

(2) Block Diagram

(3) Wave Form Graph

- **Demo_Videos**

SUMMARY

The provided ATM FSM (Finite State Machine) project is a simulation of an ATM (Automated Teller Machine) system. The project demonstrates the functionality of an ATM, including account validation, PIN verification, balance inquiry, withdrawal, deposit, and mini statement generation.

The project utilizes Verilog HDL (Hardware Description Language) to implement the ATM FSM. It defines the state transitions and behaviour of the ATM system based on various inputs such as account number, PIN, user selections, and timing signals. The FSM design allows the ATM to progress through different states, ensuring proper validation and processing of user actions.

The project incorporates a clock divider module to generate a slower clock signal from the system clock, allowing for timing control and synchronization of operations. It also includes registers and arrays to store and manage account information, PINs, balances, and transaction details.

The outcome of the project is the simulation of an ATM system that can handle basic banking operations. It provides a framework for understanding and implementing the logic required for an ATM, including user authentication, transaction processing, and maintaining account information. This project can serve as a starting point for further development and customization of an actual ATM system.

Result

Understanding Finite State Machines: By working on an ATM FSM project, we gained a deeper understanding of the concept of Finite State Machines. we learned how to model complex systems using states, transitions, and actions, and how to implement them in code and on the FPGA.

System Design and Architecture: Building an ATM system requires careful system design and architecture. we learned how to design and structure software components, handle inputs and outputs, and manage state transitions effectively.

Transaction Processing: The ATM FSM project involves implementing various transactional operations such as account validation, PIN verification, balance inquiry, withdrawal, deposit, and mini statement generation. We gained insights into how these operations are processed and managed in a real-world ATM system.

Problem-Solving and Troubleshooting: While working on the project, you may encounter various challenges, bugs, or issues. These experiences will sharpen our problem-solving and troubleshooting skills as we analyse the problems, identify the root causes, and find effective solutions.

Implementation on FPGA issues:

For the Labs land DE1 SoC I have observed that they have used GPIO ports for the assignment for the switches but not the pins given in the DE1 SoC manual.

I am assuming they have done something similar for the push buttons also, but I don't have the list of the pin configuration details that were used to configure the Labs land FPGA, the pins in the manual and pins used by Labs Land to interface switches and push buttons might be different.
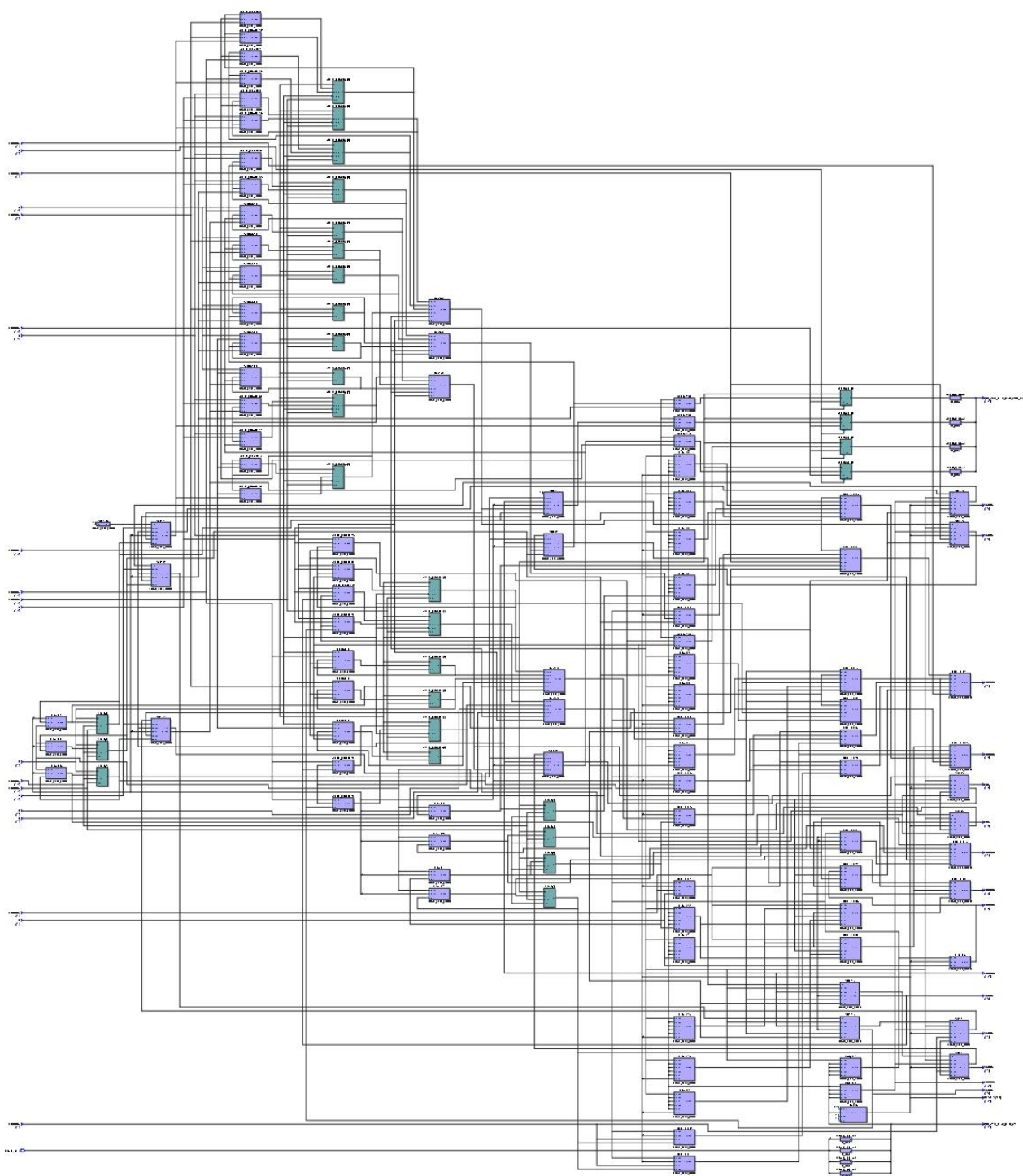This is an issue I am facing at the final step of the project.
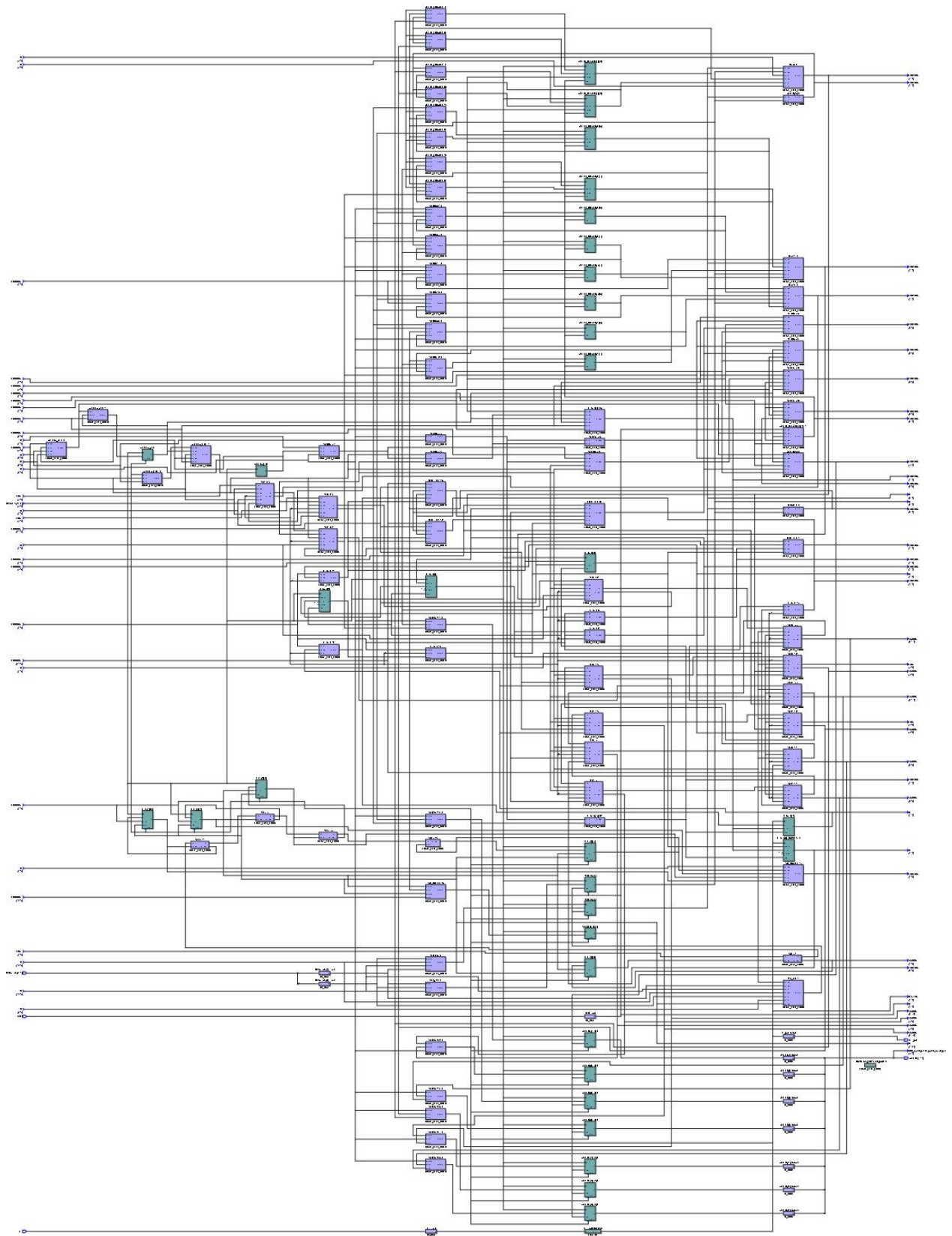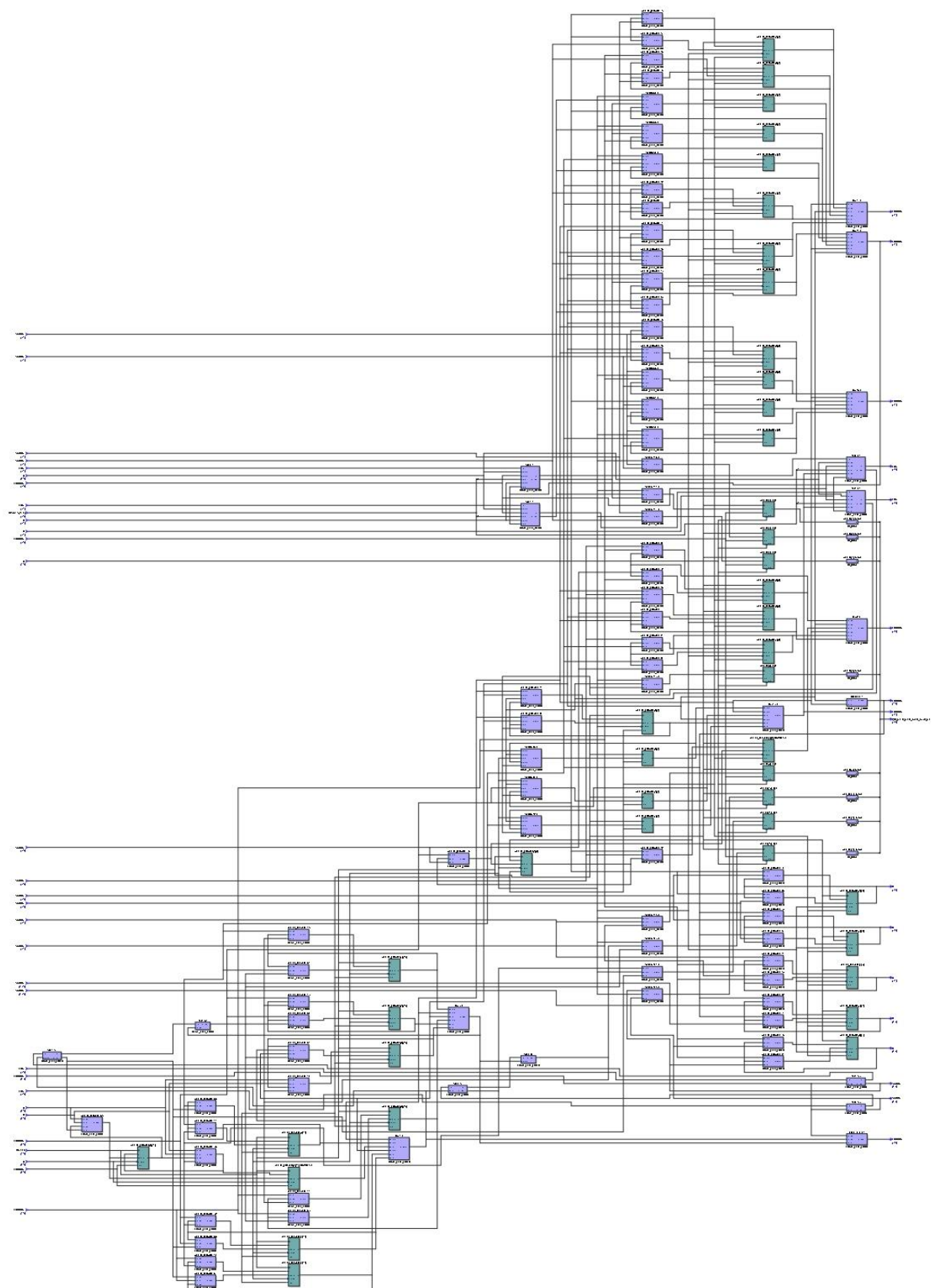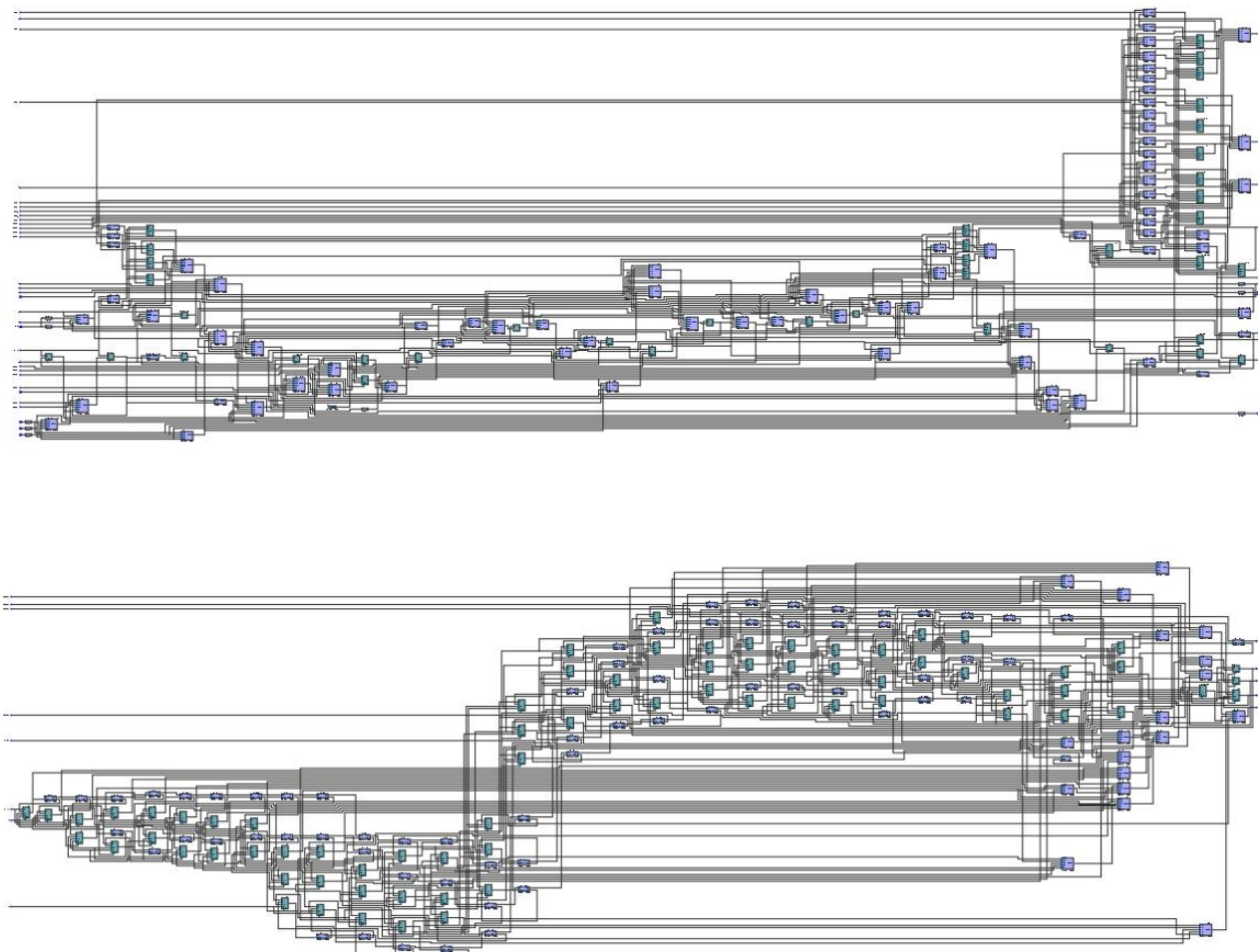
RTL Viewer:
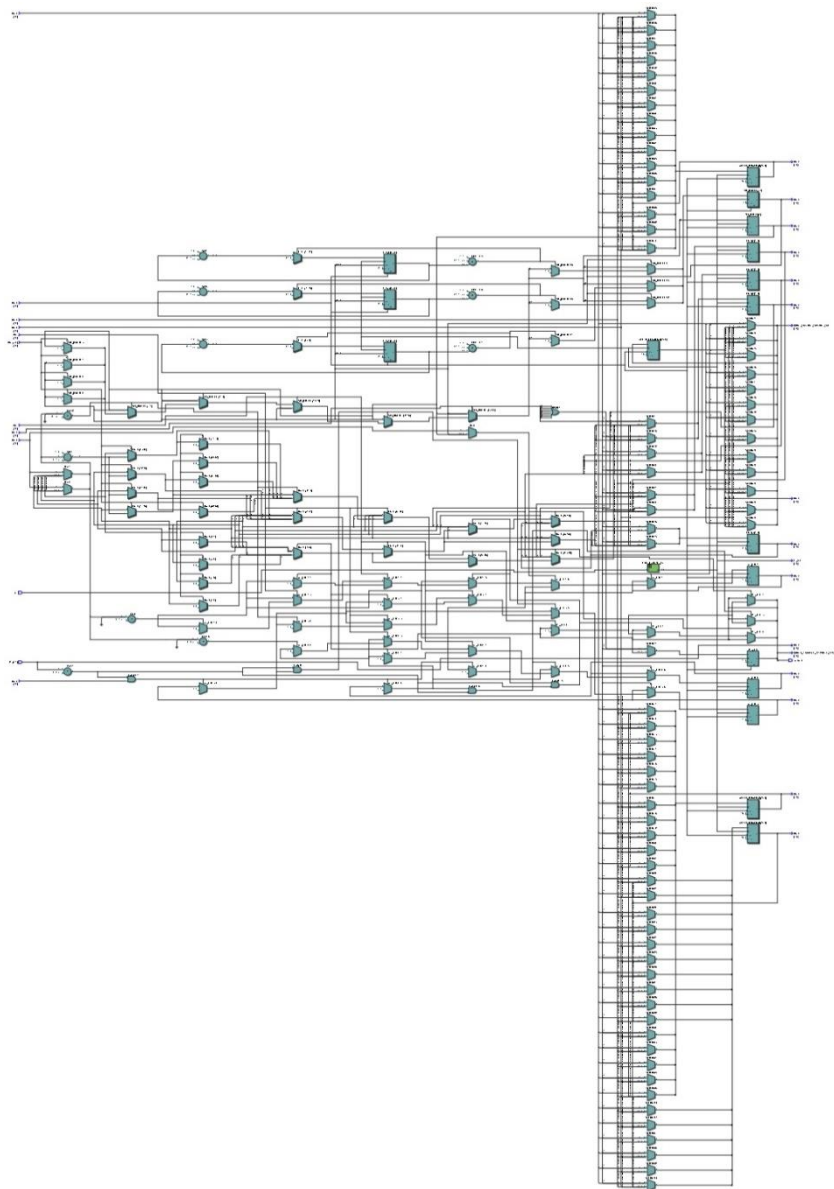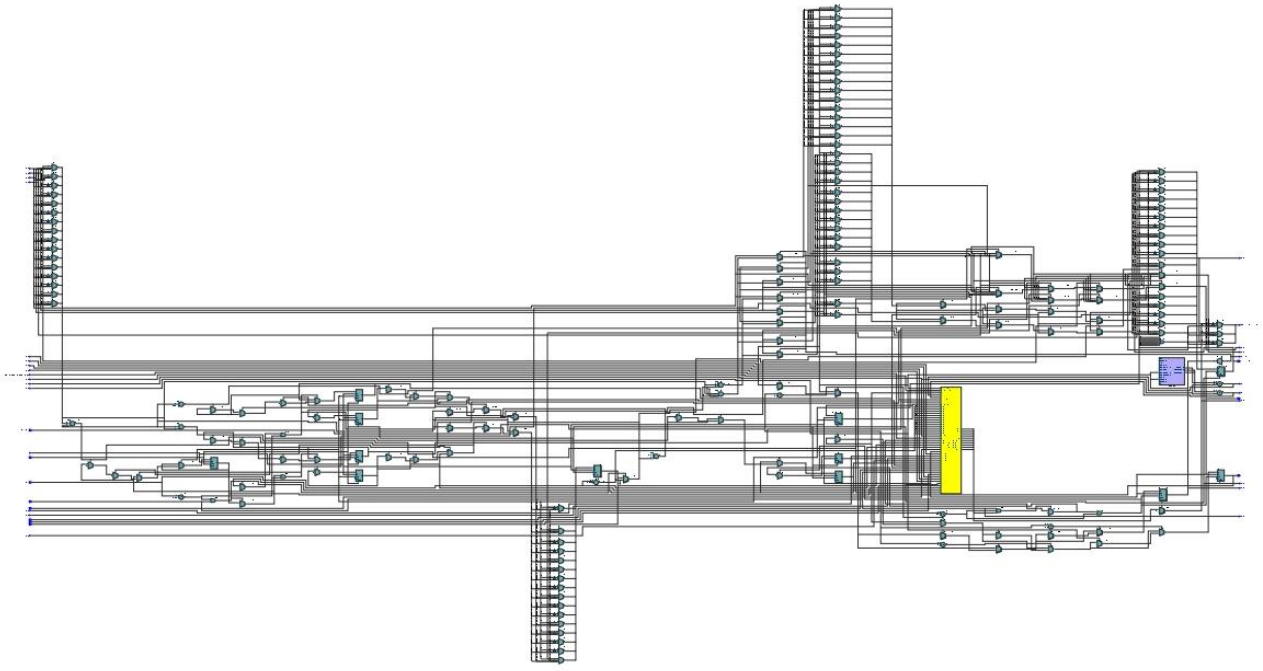
Technology map viewer post mapping
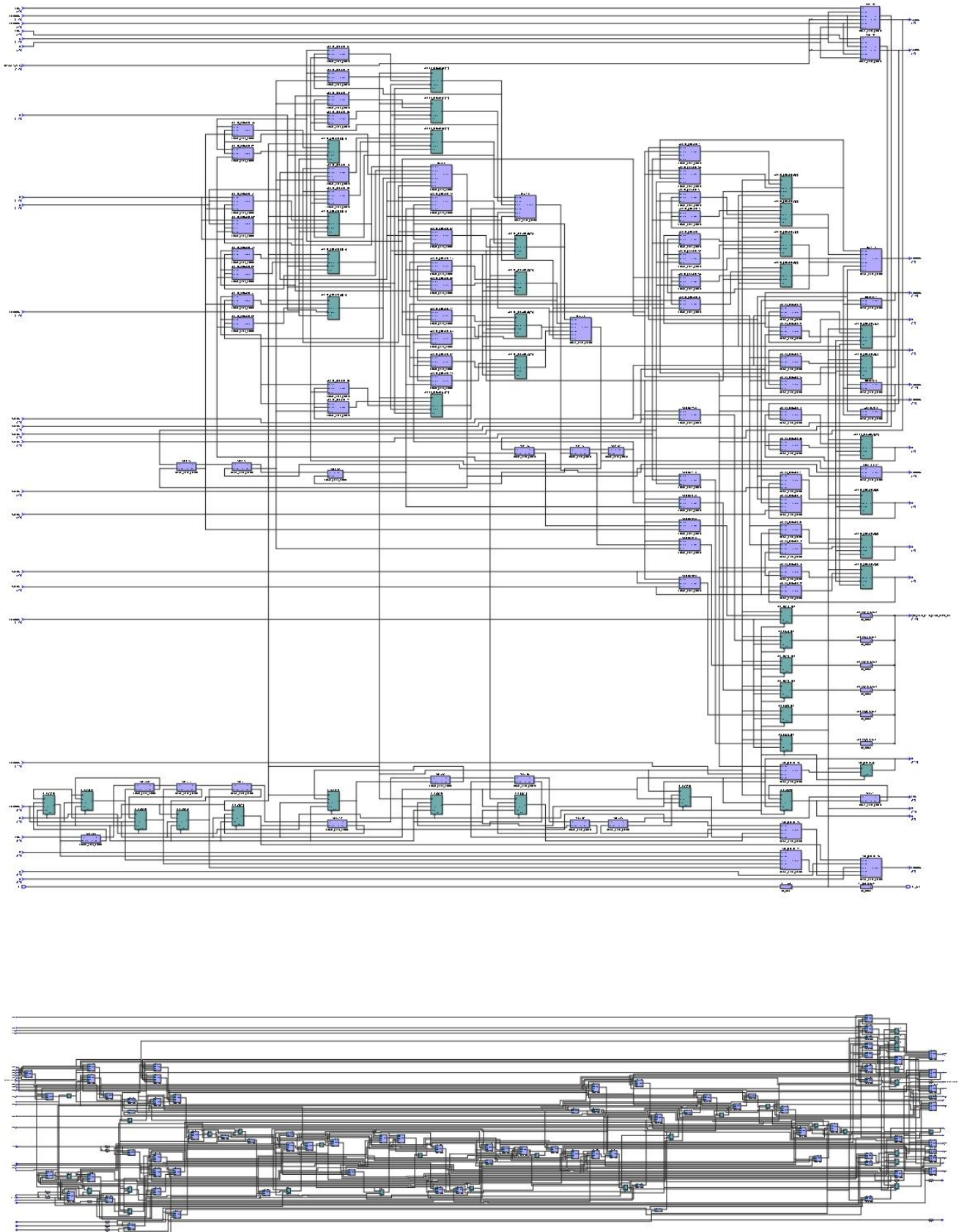
Technology Map Viewer Post Fitting

Flow Summary

Flow Summary	
🔍 <<Filter>>	
Flow Status	Successful - Sat Jul 15 13:17:55 2023
Quartus Prime Version	22.1std.1 Build 917 02/14/2023 SC Lite Edition
Revision Name	atm
Top-level Entity Name	atm
Family	Cyclone V
Device	5CSEMA5F31C6
Timing Models	Final
Logic utilization (in ALMs)	242 / 32,070 (< 1 %)
Total registers	213
Total pins	34 / 457 (7 %)
Total virtual pins	0
Total block memory bits	0 / 4,065,280 (0 %)
Total DSP Blocks	1 / 87 (1 %)
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 %)

Analysis and synthesis Summary

Analysis & Synthesis Summary

🔍 <<Filter>>

Analysis & Synthesis Status	Successful - Sat Jul 15 13:16:23 2023
Quartus Prime Version	22.1std.1 Build 917 02/14/2023 SC Lite Edition
Revision Name	atm
Top-level Entity Name	atm
Family	Cyclone V
Logic utilization (in ALMs)	N/A
Total registers	199
Total pins	34
Total virtual pins	0
Total block memory bits	0
Total DSP Blocks	1
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
Total DLLs	0

Resource Usage Summary

Analysis & Synthesis Resource Usage Summary

🔍 <<Filter>>

	Resource	Usage
1	Estimate of Logic utilization (ALMs needed)	247
2		
3	▼ Combinational ALUT usage for logic	419
1	-- 7 input functions	1
2	-- 6 input functions	73
3	-- 5 input functions	84
4	-- 4 input functions	68
5	-- <=3 input functions	193
4		
5	Dedicated logic registers	199
6		
7	I/O pins	34
8		
9	Total DSP Blocks	1
10		
11	Maximum fan-out node	clk~input
12	Maximum fan-out	200
13	Total fan-out	2562
14	Average fan-out	3.73

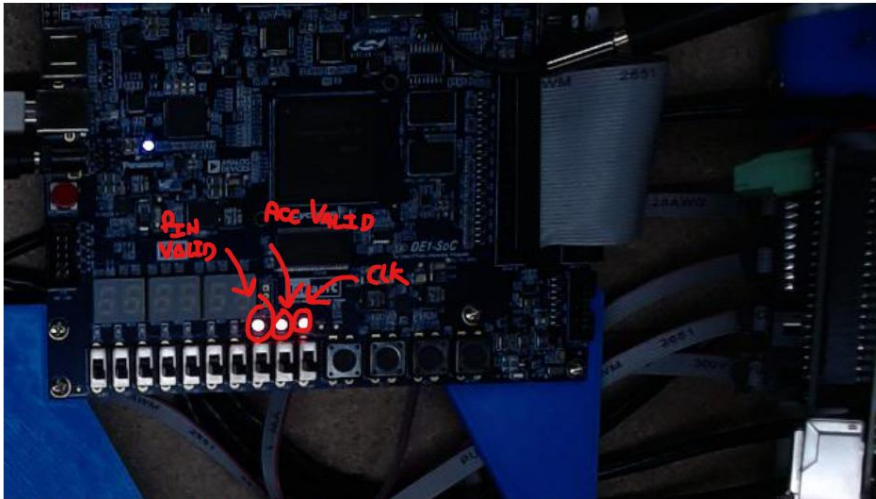# Fitter Summary

Fitter Summary

🔍 <<Filter>>

Fitter Status	Successful - Sat Jul 15 13:17:39 2023
Quartus Prime Version	22.1std.1 Build 917 02/14/2023 SC Lite Edition
Revision Name	atm
Top-level Entity Name	atm
Family	Cyclone V
Device	5CSEMA5F31C6
Timing Models	Final
Logic utilization (in ALMs)	242 / 32,070 (< 1 %)
Total registers	213
Total pins	34 / 457 (7 %)
Total virtual pins	0
Total block memory bits	0 / 4,065,280 (0 %)
Total RAM Blocks	0 / 397 (0 %)
Total DSP Blocks	1 / 87 (1 %)
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 %)

Power Analyzer Summary

Power Analyzer Summary

🔍 <<Filter>>

Power Analyzer Status	Successful - Sat Jul 15 13:44:12 2023
Quartus Prime Version	22.1std.1 Build 917 02/14/2023 SC Lite Edition
Revision Name	atm
Top-level Entity Name	atm
Family	Cyclone V
Device	5CSEMA5F31C6
Power Models	Final
Total FPGA Thermal Power Dissipation	439.25 mW
Core Dynamic Thermal Power Dissipation	1.97 mW
Core Static Thermal Power Dissipation	417.34 mW
I/O Thermal Power Dissipation	19.93 mW
HPS Dynamic (Dual core) Power	802.38 mW
HPS Dynamic (Single core) Power	793.28 mW
Total FPGA and HPS Power	1241.63 mW
Power Estimation Confidence	Low: user provided insufficient toggle rate data

Partial Implementation on FPGA