VHDL-Based ATM System

DONE BY : Mayar Waleed Salah - Fatma Ayoub Ahmed

Mariam Sherif El Sayed - Noura Essam Abdelsattar

Marwa Hussien Ahmed Ali - Youssef Mohamed Ahmed

Mostafa Alaa Eldin Hassan - Mohaned Mohamed Mostafa-

Mohamed Waleed Mohamed - Ahmed Waleed

SUPERVISED BY : DR. Mohamed El Bably

DEPARTMENT : Computers And Systems

YEAR : 3

➢ ***Introduction***

This report describes the development of a Finite State Machine (FSM) for an Automated Teller Machine (ATM) system using VHDL (VHSIC Hardware Description Language). The primary goal is to simulate the functional operation of an ATM, including card authentication, withdrawal, deposit, balance checking, and receipt generation.

➢ ***Objective***

The objective of this project is to implement a robust and efficient ATM system by:

1. **Designing a state-based control system using FSM.**
2. **Ensuring secure user authentication.**
3. **Providing support for key functionalities like withdrawal, deposit, and balance inquiries.**
4. **Maintaining user-friendliness and logical transitions between various states.**

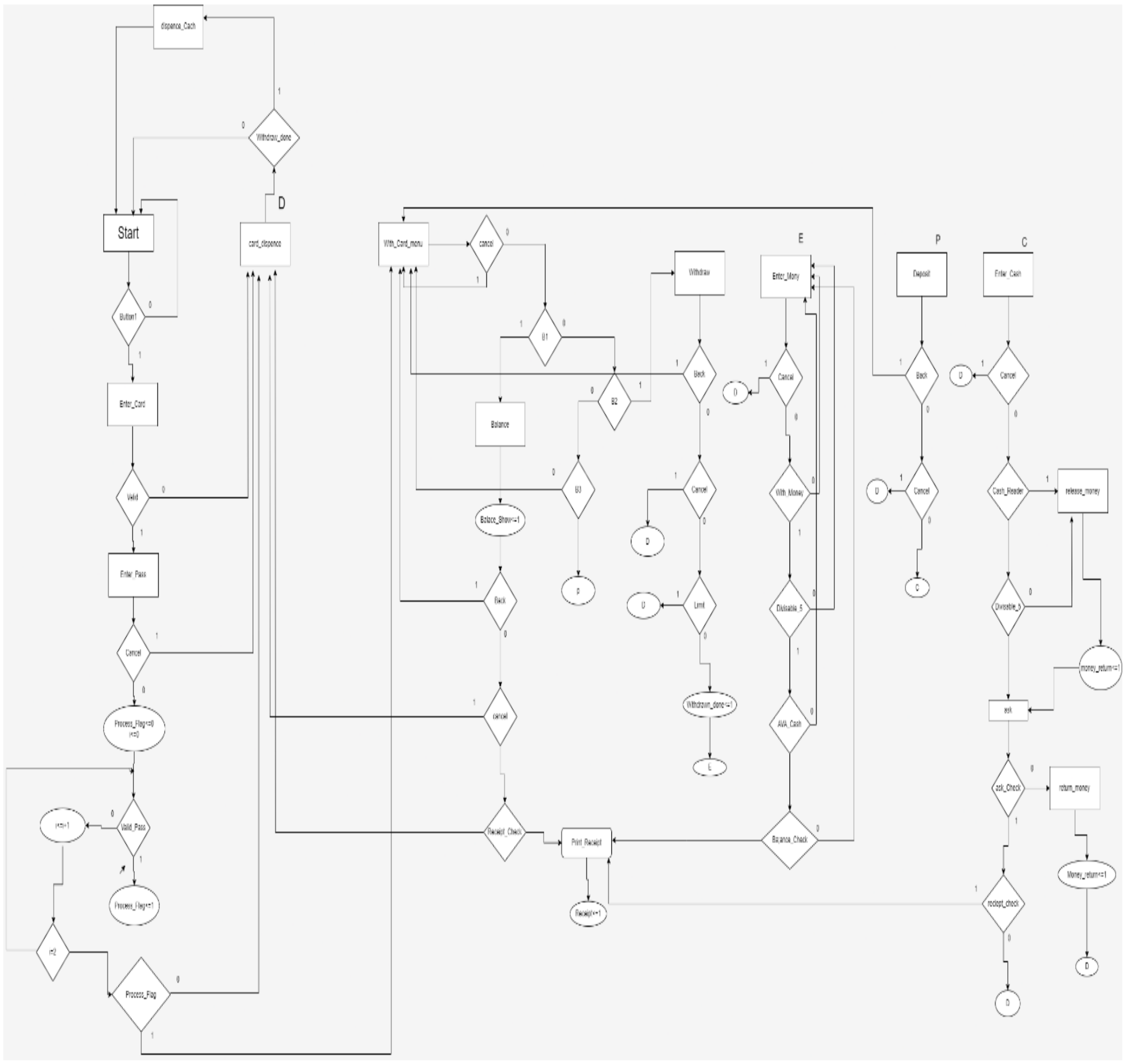
➢ ***System Design***

The system follows a Finite State Machine (FSM) model with predefined states to represent different operational stages of the ATM. The design uses a synchronous state transition model, controlled by a clock and reset signal.

➢ ***Key Features***

- **User Authentication: Validates the card and password before granting access.**
- **Menu Options: Provides separate states for balance inquiry, withdrawal, and deposit.**
- **Error Handling: Handles invalid inputs, incorrect passwords, and cancels gracefully.**
- **Receipt Printing: Offers the option to print a transaction receipt.**

➢ ***Flow chart***


dispence_Cach
Withdraw_done
Start
card_dispence
D
Button1
Enter_Card
Valid
Enter_Pass
Cancel
Process_Flag<=0
Valid_Pass
Process_Flag<=1
Process_Flag
With_Card_menu
cancel
B1
B2
B3
Balance
Balace_Show<=1
Back
Receipt_Check
Print Receipt
Receipt<=1
Withdraw
Limit
Withdrawn_done<=1
E
Enter_Mony
With_Money
Divisable_5
AVA_Cash
Balance_Check
P
Deposit
C
Enter_Cash
Cash_Reader
release_money
Divisable_5
money_return<=1
ask
ask_Check
return_money
Money_return<=1
receipt_check

➢*Code Implementation*

1.Libraries :

```
library ieee;
use ieee.std_logic_1164.all;
```

2.Entity:

```
entity ATM
is

  port (
    button1, button2, button3, back, cancel, reset, clk , receipt_check, limit,
with_money, divisble_5, ava_cash, ask_check:in std_logic;
    reciept, money_return, Balance_show, cash_show                            :
out std_logic;
    cash_reader                           : in std_logic;
    valid_pass:in std_logic_vector(2 downto 0);
    valid, balance_check:in std_logic;
    prs_o, nxt_o: inout STD_LOGIC_VECTOR(4 downto 0)
  );
end ATM;
```

3.Architecture:

```
architecture behaviour of ATM is
  type state is (
    Start, -- 0
    ask, -- 1
    Enter_Card, -- 2
    Enter_Pass, -- 3
    Enter_Money, -- 4
    Enter_Cash, -- 5
    With_Card_menu, -- 6
    Balance, -- 7
    withdraw, -- 8
    deposit, -- 9
    print_reciept, -- 10
    Card_dispence, -- 11
    Enter_Phone, -- 12
    WIthout_Card_Menu, -- 13
```

```
        return_money, -- 14
        dispense_cash, -- 15
        release_money); -- 16
    signal prs, nxt : state;

  begin
    seq : process (clk, reset)
    begin
      if reset = '1' then
        prs <= Start;
      else
        if (rising_edge(clk)) then
          prs <= nxt;
          prs_o <= nxt_o;
        end if;
      end if;
    end process seq;

    comb : process (cancel, back, button1, button2, button3, reset, clk,
  receipt_check, limit, with_money, divisble_5, ava_cash, cash_reader, valid_pass,
  valid, balance_check)
        variable withdrawn_done: std_logic := '0';
        variable pass_flag: std_logic := '0';

    begin
      if prs = start then      -- Start State
          Balance_show <= '0';
          reciept <= '0';
          cash_show<='0';
          money_return <= '0';
          withdrawn_done:='0';
        if button1 = '1' then -- with card
          nxt <= Enter_Card;
          nxt_o <= "00010";
        else
          nxt <= Start;
          nxt_o<="00000";
        end if;

      elsif prs = Enter_Card then -- Enter Card State
          if valid='1' then
              nxt<=Enter_Pass;
              nxt_o<="00011";
          else
              nxt<= Card_dispence;
              nxt_o<="01011";
          end if;

      elsif prs= Card_dispence then --Card dispense State
          if withdrawn_done='1' then
              nxt<=dispense_cash;
              nxt_o <= "01111";
          else
              nxt<=Start;
              nxt_o <= "00000";
          end if;

      elsif prs=dispense_cash then
          cash_show<='1';
          nxt<=Start;
          nxt_o <= "00000";

      elsif prs=Enter_Pass then -- Entering password
        if cancel = '1' then
          nxt <= Card_dispence;
          nxt_o<="01011";
        else
          pass_flag:='0';
            L: for i in 0 to 2 Loop
                if valid_pass(i)='1' then
                  pass_flag:='1';
                  exit;
                end if;
            end  loop L;

          if pass_flag='1' then
            nxt<= With_Card_menu;
            nxt_o <= "00110";
            else
            nxt<=Card_dispence;
            nxt_o<="01011";
          end if;
        end if;
      elsif prs = with_Card_menu then -- Menu State
          if cancel = '1' then
            nxt <= Card_dispence;
            nxt_o<="01011";
          elsif button1 = '1' and button2 = '0' and button3 = '0' then -- balance
              nxt <= Balance;
              nxt_o <= "00111";
          elsif button2 = '1' and button1 = '0' and button3 = '0' then -- withdraw
              nxt <= withdraw;
              nxt_o <= "01000";
          elsif button3 = '1' and button1 = '0' and button2 = '0' then -- deposite
              nxt <= deposit;
              nxt_o <= "01001";
          else
              nxt <= with_Card_menu;
              nxt_o <= "00110";
          end if;


      elsif prs = Balance then -- Check Balance State
            Balance_show <= '1';

            if back = '1' then
              nxt <= With_Card_menu;
              nxt_o <= "00110";
            elsif cancel = '1' then
              nxt <= Card_dispence;
              nxt_o<="01011";
            elsif receipt_check = '1' then
              --reciept<='1';
              nxt<=print_reciept;
              nxt_o<="01010";
            else
              reciept <= '0';
              nxt<=card_dispence;
              nxt_o<="01011";
            end if;

      elsif prs = withdraw then -- Withdraw State
          if back = '1' then
            nxt <= With_Card_menu;
            nxt_o <= "00110";
          elsif cancel = '1' then
            nxt <= Card_dispence;
            nxt_o<="01011";
          elsif limit='1' then
            nxt<=card_dispence;
            nxt_o<="01011";
          else
            nxt<=Enter_Money;
            nxt_o <= "00100";
            withdrawn_done := '1';
          end if;

      elsif prs = Enter_Money then -- Enter Money State in withdrawing
          if cancel = '1' then
            nxt <= Card_dispence;
            nxt_o<="01011";
          elsif with_money='1' and divisble_5='1' and ava_cash='1' and 
  balance_check='1' then
            if receipt_check='1' then
              nxt <= print_reciept;
              nxt_o <= "01010";
            else
              nxt <= Card_dispence;
              nxt_o<="01011";
            end if;
          else
            nxt <= Enter_Money;
            nxt_o <= "00100";
          end if;

      elsif prs = print_reciept then --Print Receipt & Card Dispense State
          reciept<='1';
          nxt<=Card_dispence;
          nxt_o<="01011";
        end if;

      elsif prs = deposit then -- Deposit State
          if back = '1' then
            nxt <= With_Card_menu;
            nxt_o <= "00110";
          elsif cancel = '1' then
            nxt <= Card_dispence;
            nxt_o<="01011";
          else
            nxt<=Enter_Cash;
            nxt_o <= "00101";
        end if;

      elsif prs = Enter_Cash then -- Enter_Cash State in Deposit
        if cancel = '1' then
          nxt <= Card_dispence;
          nxt_o<="01011";
        elsif cash_reader='1' or divisble_5 = '0' then
            nxt <= release_money;
            nxt_o <= "10000";
          else
            nxt<= ask;
            nxt_o <= "00001";
        end if;

      elsif prs = release_money then -- release State : return bad money
        money_return<='1'; -- bad money output
        nxt<=ask;
        nxt_o <= "00001";

      elsif prs = ask then -- ASK State after Deposit
        if ask_check ='1'and receipt_check ='1' then
          nxt <= print_reciept;
          nxt_o <= "01010";
        elsif ask_check ='1'and receipt_check ='0' then
          nxt<=Card_dispence;
          nxt_o<="01011";
        else
          nxt <= return_money;
          nxt_o <= "01110";
        end if;

      elsif prs= return_money then -- return_money State: if user cancel the
  deposit process
          money_return<='1';
          nxt<=Card_dispence;
          nxt_o<="01011";
      end if;
      end process comb;
  end behaviour;
```

➢ ***Entity Declaration***

The ATM entity defines the input and output ports:

- **Inputs: Button signals (button1, button2, button3), control signals (cancel, back, reset), and validation signals (valid_pass, valid, balance_check).**
- **Outputs: Signals for displaying balance, showing cash, printing receipts, and returning money.**
- **Bidirectional Ports: prs_o and nxt_o are used to track the current and next state.**

➢ ***Architecture***

The FSM architecture comprises the following:

1. **States: The system has 17 states including Start, Enter_Card, Enter_Pass, With_Card_menu, Balance, withdraw, deposit, Card_dispence, etc.**
2. **State Transition Logic: Managed using a synchronous process (seq) and combinational logic (comb). The seq process updates the present state (prs) based on clock signals, while comb calculates the next state (nxt) based on inputs and current state.**
3. **Validation Checks: Password validation and conditions for money withdrawal or deposit ensure the system operates securely and reliably.**

➢ ***State Descriptions***

- **Start: Initial state where the user selects operations like "with card" or "without card."**
- **Enter_Card: Validates the user card.**
- **Enter_Pass: Authenticates the user password.**
- **With_Card_menu: Displays menu options for balance, withdrawal, and deposit.**
- **Balance: Displays the current account balance.**
- **withdraw: Initiates the withdrawal process, including amount entry and validation.**

- **deposit: Handles cash deposits and checks the validity of the inserted money.**
- **Card_dispence: Dispenses the card after the transaction.**
- **print_reciept: Prints a receipt for the transaction.**

➢ ***Results***

The implemented FSM simulates a fully functional ATM system, demonstrating:

1. **Smooth state transitions based on user inputs.**
2. **Error-free operation with secure user authentication.**
3. **Logical handling of corner cases, such as invalid passwords or incorrect input amounts.**

➢ ***Test Bench code:***

1.Libraries :

```
library ieee;
 use ieee.std_logic_1164.all;
```

2.Entity:

```
entity ATM_TB is
end ATM_TB;
```

3.Architecture:

```
architecture behavoiur of ATM_TB is
    component ATM is
        port (
            button1, button2, button3, back, cancel, reset, clk , receipt_check,
limit, with_money, divisble_5, ava_cash, ask_check:in std_logic;
            reciept, money_return, Balance_show,
cash_show                            : out std_logic;
            cash_reader                             : in std_logic;
            valid_pass:in std_logic_vector(2 downto 0);
            valid, balance_check:in std_logic;
            prs_o, nxt_o: inout STD_LOGIC_VECTOR(4 downto 0)
          );
    end component;
    signal prs_o, nxt_o: std_logic_vector(4 downto 0);
    signal cash_show, valid, balance_check, cash_reader, button1, button2,
button3, back, cancel, reset, clk , receipt_check, limit, with_money, divisble_5,
ava_cash, ask_check: std_logic;
    signal valid_pass: std_logic_vector(2 downto 0);
    signal reciept, money_return, Balance_show: std_logic;
    begin
        port1: ATM port map(button1, button2, button3, back, cancel, reset, clk ,
receipt_check, limit, with_money, divisble_5, ava_cash, ask_check, reciept,
money_return, Balance_show, cash_show, cash_reader, valid_pass, valid,
balance_check, prs_o, nxt_o);
        -- generation of clock
        clk_process : PROCESS
        BEGIN
            l:for i in 0 to 400 loop
                if clk = '0' then
                    clk <= '1';
                else
                    clk <= '0';
                end if;
                WAIT FOR 5 ns;
            end loop l;
            wait;
        END PROCESS clk_process;

        comb : process
            begin
            reset <= '1'; -- reset
            WAIT FOR 10 ns;
            reset <= '0'; -- choose Enter card
            button1 <= '1';
            button2 <= '0';
            button3 <= '0';
            wait for 10 ns; -- valid card input
            valid <= '1';
            wait for 10 ns; -- valid pass input
            valid_pass <= "100";
            wait for 10 ns; -- choose balance from card menu
            button1 <= '1';
            button2 <= '0';
            button3 <= '0';
            receipt_check <= '1';
            wait for 10 ns; -- want receipt
            wait for 10 ns; -- dispense
            wait for 10 ns; -- start
            WAIT FOR 10 ns;
```

```
button1 <= '1';
button2 <= '0';
button3 <= '0';
wait for 10 ns; -- valid card input
valid <= '1';
wait for 10 ns; -- valid pass input
valid_pass <= "100";
wait for 10 ns;
button1 <= '0';
button2 <= '0';
button3 <= '1';
wait for 10 ns; -- deposit
wait for 10 ns; -- enter cash
cash_reader <= '0';
divisble_5 <= '1';
wait for 10 ns;
ask_check <= '1';
receipt_check <= '1';
wait for 10 ns;
wait for 10 ns; -- want receipt
wait for 10 ns; -- dispense

button1 <= '1';
button2 <= '0';
button3 <= '0';
wait for 10 ns; -- valid card input
valid <= '1';
wait for 10 ns; -- valid pass input
valid_pass <= "100";
wait for 10 ns;
button1 <= '0';
button2 <= '1';
button3 <= '0';
wait for 10 ns; -- withdraw
wait for 10 ns; -- enter money
with_money <= '1';
divisble_5 <= '1';
ava_cash <= '1';
balance_check <= '1';
receipt_check <= '0';
wait for 10 ns;
wait for 10 ns;
wait for 10 ns; -- dispense
```

```
            button1 <= '1';
            button2 <= '0';
            button3 <= '0';
            wait for 10 ns; -- valid card input
            valid <= '1';
            wait for 10 ns; -- valid pass input
            valid_pass <= "000"; --failed password
            wait for 10 ns; -- dispense

            button1 <= '1';
            button2 <= '0';
            button3 <= '0';
            wait for 10 ns; -- valid card input
            valid <= '1';
            wait for 10 ns; -- valid pass input
            valid_pass <= "100";
            wait for 10 ns;
            button1 <= '0';
            button2 <= '1';
            button3 <= '0';
            wait for 10 ns; -- withdraw
            wait for 10 ns; -- enter money
            with_money <= '1';
            divisble_5 <= '1';
            ava_cash <= '1';
            balance_check <= '1';
            receipt_check <= '0';
            wait for 10 ns;
            wait for 10 ns;
            wait for 10 ns; -- dispense

            wait;
        end process comb;
end behavoiur;
```

➢ ***Structure of the Test Bench***

- **Entity Declaration**

  The ATM_TB entity serves as the test bench and does not have ports because it is used solely for simulation purposes.

- **Architecture**

  The architecture behavoiur includes:

4. **Component Instantiation:**

   The ATM component is instantiated with all input and output signals connected to corresponding test signals declared in the test bench.

5. **Signal Declarations:**

   Signals are defined to simulate inputs and outputs of the ATM system.

6. **Examples:**

   prs_o and nxt_o simulate the current and next states.

   button1, button2, button3 simulate user inputs.

   cash_reader, with_money, and divisible_5 represent conditions for cash transactions.

7. **Clock Generation:**

   A process (clk_process) generates a clock signal with a 10 ns period (5 ns high, 5 ns low).

   The clock drives the ATM's synchronous operations.

8. **Simulation Process:**

   A comb process simulates various ATM scenarios by manipulating input signals and observing outputs.

- **Sequence of Operations:**

  Reset:

  The system is reset initially (reset <= '1') and then released (reset <= '0').

  Scenario 1: Valid card, valid password, balance inquiry.

  Scenario 2: Valid card, valid password, deposit operation.

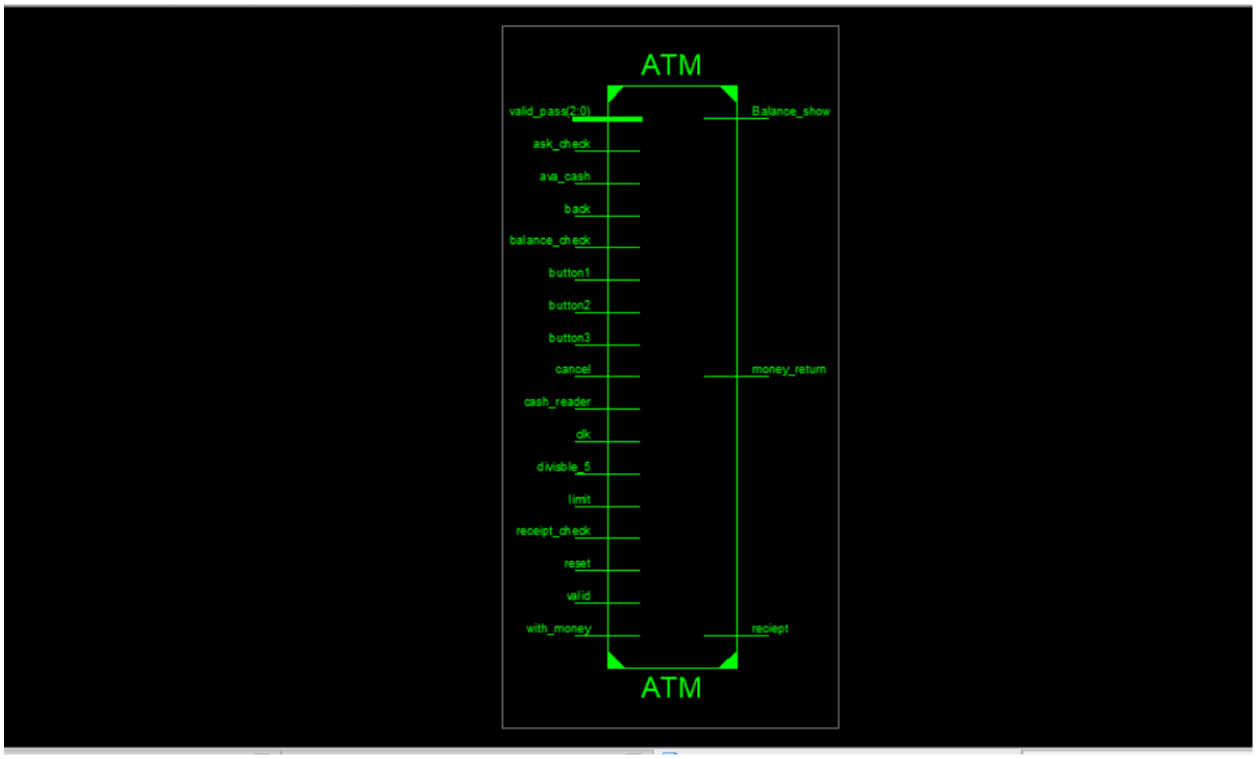
  Scenario 3: Valid card, valid password, withdrawal operation.

  Scenario 4: Valid card, invalid password, failed authentication.

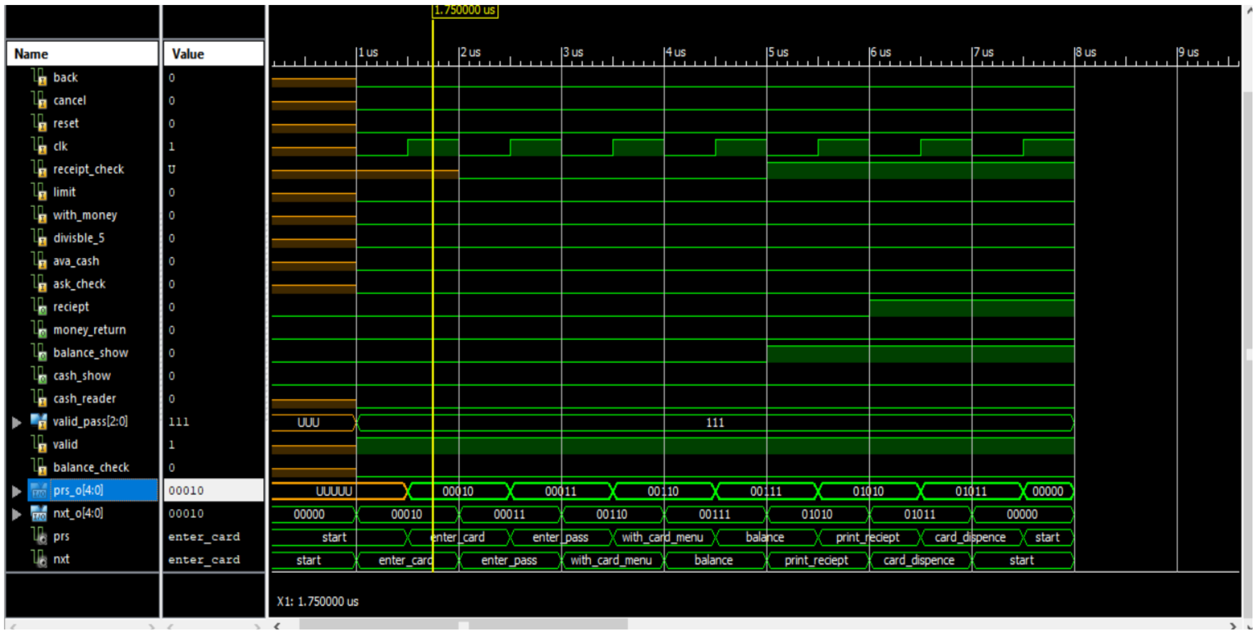
  Scenario 5: Additional withdrawal test with conditions for successful operation.

  Each scenario includes appropriate delays (wait for 10 ns) to mimic real-time behavior and ensure synchronization.

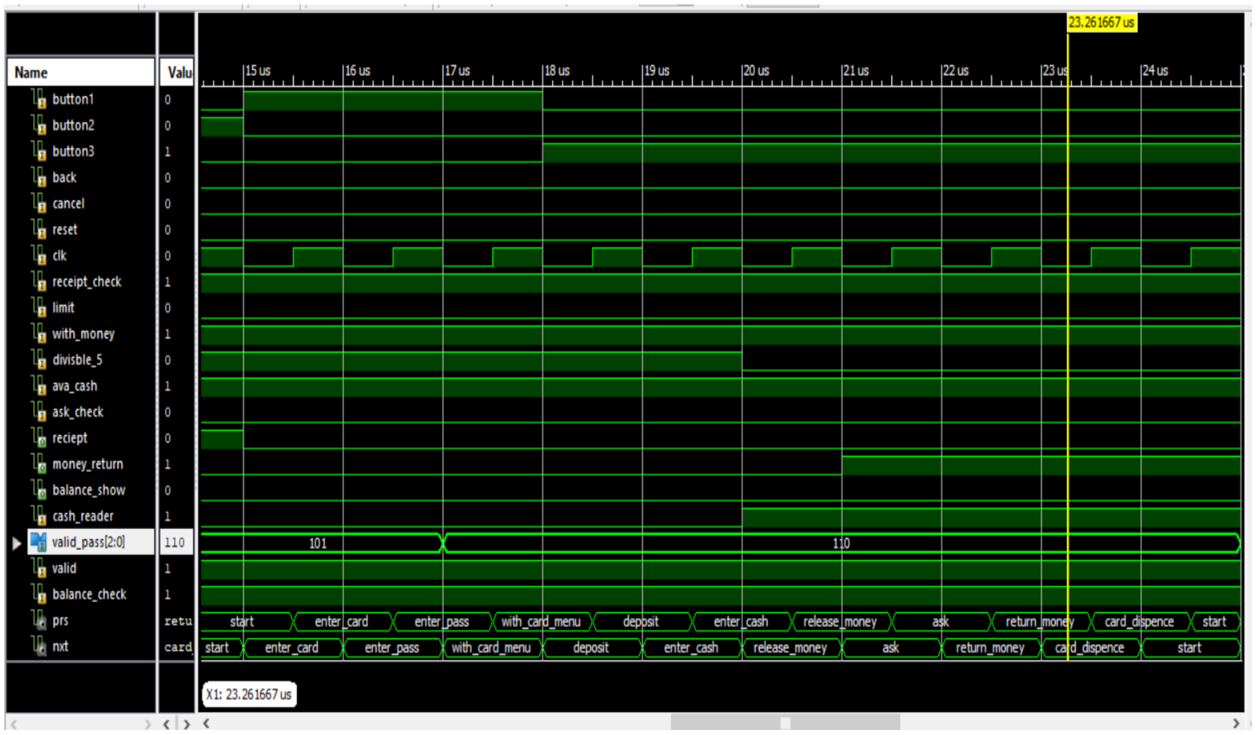
➢ Chip :



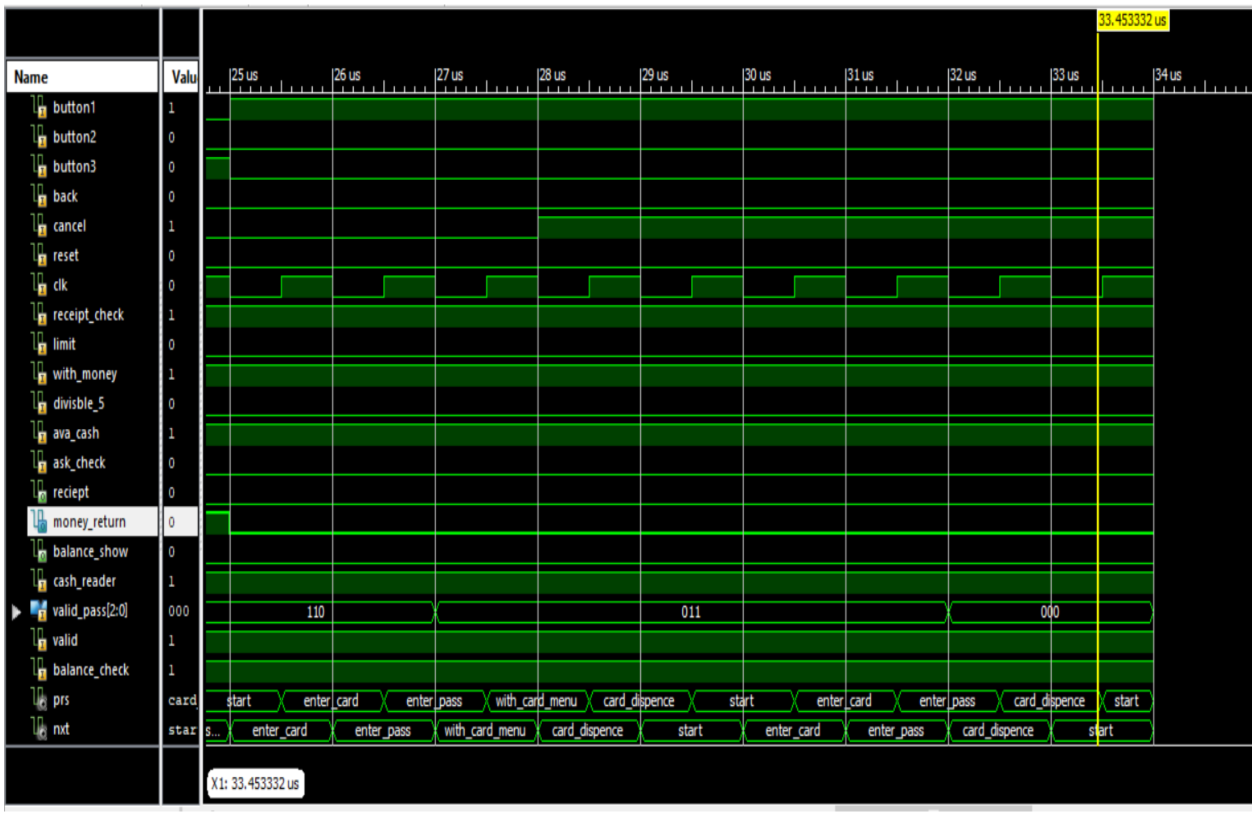
➢ Sample of simulation



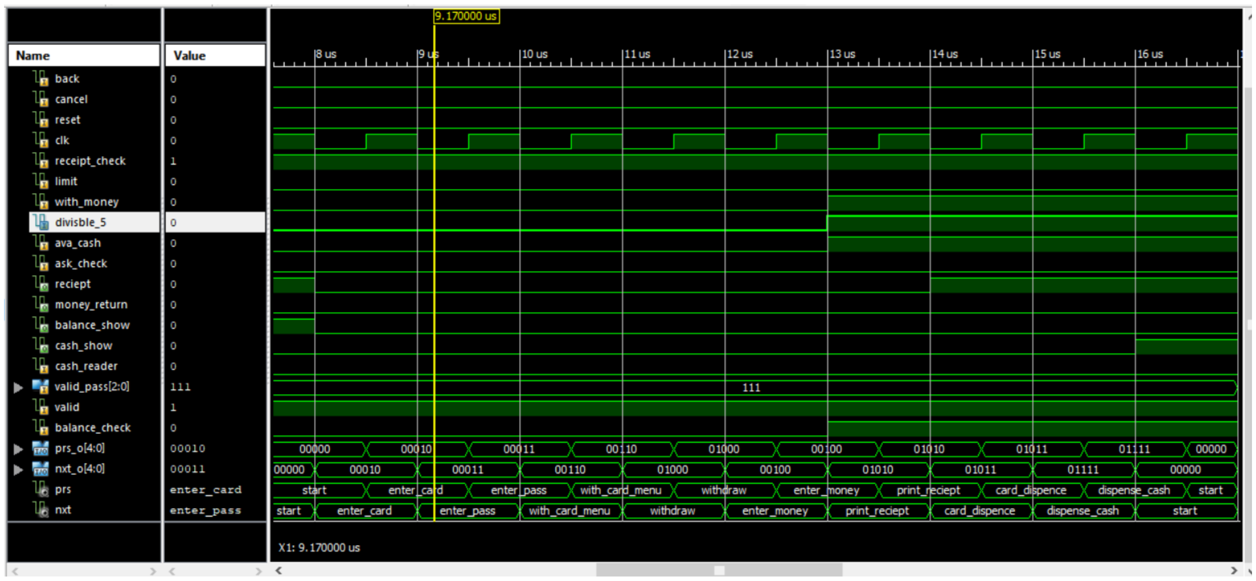
balance



deposite



pass



Withdrawn

➢ ***Conclusion***

This project successfully implemented a VHDL-based FSM for an ATM system. The design ensures secure, user-friendly operation while meeting the functional requirements of modern ATM machines. Future work may include enhancing the system with features like multi-language support, improved user interfaces, and integration with external banking networks.

Code Summary

The VHDL code demonstrates the system's logic using synchronous and combinational processes. Key variables and signals manage state transitions, user inputs, and outputs for seamless operation.