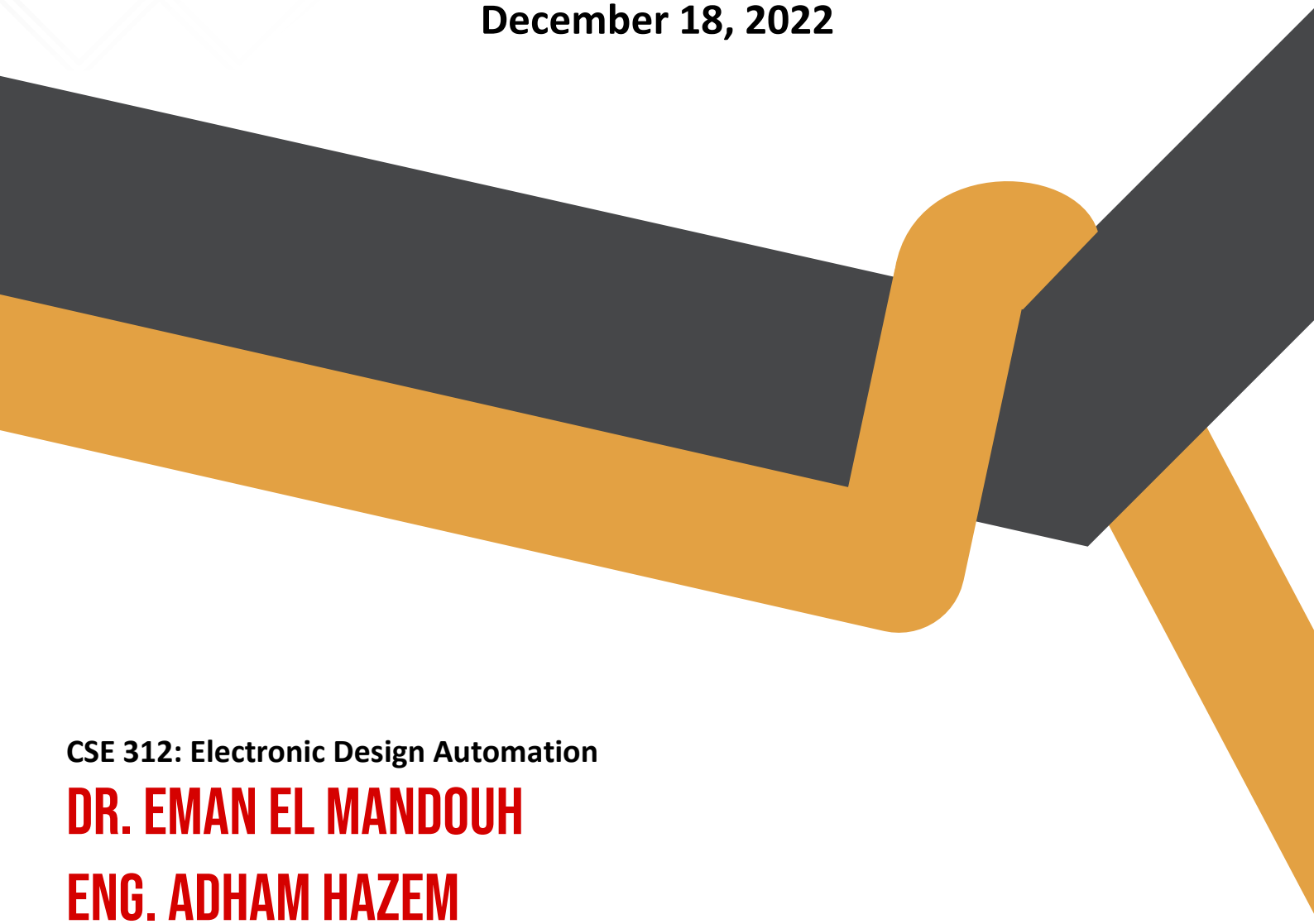
Ain Shams University – Faculty of Engineering

EDA PROJECT (1)

ATM – Based Bank System

December 18, 2022



CSE 312: Electronic Design Automation

DR. EMAN EL MANDOUH

ENG. ADHAM HAZEM

Team No. (10) Members

Design Team:

- Ahmed Wael Samir Abdelmegied 20P7271
- Mohamed Tarek Mohamed Abdalla Ahmed Khafagy 20P6211
- Abdelrahman Mohamed Salaheldin Mohamed 20P8967
- Ahmed Hossam Moussa Sakr 20P1009

Verification Team:

- Osama Khaled Gamal 20P5486
- Mohamed Ali Ahmed 20P7519
- Ziad Mahmoud Gomaa 20P2765
- Ahmed Mostafa Mahmoud 20P7127

Design Document

The following design description is based on the Finite State Machine state diagram which we created and could be found attached and submitted separately with the name of "ATM FSM Diagram.pdf" because it was too large to include as diagrams in this document.

We designed the "ATM – Based Bank System" to contain 30 states using gray state encoding which is although being more complex in next state combinational logic making it slower, but at the same time, the state bits change only one bit at a time between sequential states making it less switching power.

The system starts normally from an idle state, and as long as the "Reset = 0", the system will remain in the idle state as the "Reset" is designed to be active low synchronous. When the "Reset" turns to be of value '1', the ATM system then starts to function based on the first input "Cardless" which represents the type of the transaction whether it is a cardless transaction such as "Currency Exchange" and "Fawry Payments" where then the "Cardless" input will be high and equal to '1' and the system proceeds to the state of "Cardless Transaction", or whether it is a transaction using a bank card such as "View Balance", "Deposit", "Withdraw", "Money Transfer", "Mini-Bank Statement" and "Credit-Card Settlement" where then the input "Cardless" will be low and equal to '0' and the system proceeds instead to the state of "Insert Card". Followingly, the system advances automatically from either states to the state of "Choose Language" representing the language selection phase of the ATM where it proceeds to the next state of "Enter Password Or OTP" when the input named "Language" is high and equal to '1'. Then, a password should be entered to verify the identity of the user, in this design, we chose the correct password to be of 4-bits size equal to "1010" in binary representation. As long as the password is

incorrect, the system remains in the same state waiting for the correct password, when it is entered correctly, the system then proceeds to the main state of the system which is “Choose Transaction”.

There are eight available options of different transactions to be done by the system depending on the inputs of “Opcode” representing the operation code in 4-bit binary arranged gradually from first transaction to the last in gray code representation to achieve less switching power, and the previous input of “Cardless”.

The first transaction which happens when the “Opcode” is equal to “0001” alongside the “Cardless” being equal to ‘0’ is “View Balance” which represents showing the available balance of the user and is done in one step.

The second transaction which happens when the “Opcode” is “0011” alongside “Cardless” being equal to ‘0’ is “Deposit” representing the option of depositing money into the ATM. When the input “Money Deposited” is equal to ‘1’, this means money is deposited and then the system proceeds of the state of “Confirm Deposit” and the transaction is done.

The third transaction which happens when the “Opcode” is equal to “0010” alongside “Cardless” being equal to ‘0’ is “Withdraw” representing the ability to withdraw money from the ATM. Normally there are two means of withdrawing from an ATM, whether selecting an amount from the quick selection menu, or entering the required amount manually, consequently, when the input “Quick List” is low corresponding to ‘0’, the system proceeds to the state of “Enter Value Manually” where it then proceeds to the state of “Check Sufficient Funds” when an input of “Manual Withdraw Value” is high indicating a value has been entered, however, when “Quick List” is high and equal to ‘1’, the system bypasses the state of entering the value manually and proceeds directly to the state of “Check Sufficient Funds”. In “Check Sufficient Funds”, the system checks whether the

requested withdrawal value represented by the 32-bit input of “Withdraw Value” is greater, smaller or equal to the existing balance which we chose to be equal to 1 million in decimal, which is also represented as “32’h000F4240” in hexadecimal. In the case of being greater than the existing balance, then the balance is not sufficient, and the system returns back to the state named “Withdraw”, however, when the withdrawal value is smaller than or equal to the existing balance indicating sufficient balance available, the system shifts to the next transition of “Confirm Withdraw” and the transaction is done.

The fourth transaction which happens when the “Opcode” is equal to “0110” alongside “Cardless” being equal to ‘1’ is “Exchange Currency” which proceeds to the next state of “Confirm Exchange” when the input named “Currency Deposited” is high being equal to ‘1’ indicating that a currency has been deposited into the ATM and the transaction is done.

The fifth transaction which happens when the “Opcode” is equal to “0111” alongside “Cardless” being equal to ‘1’ is “Fawry Payments” which advances automatically to the next state of “Choose Service”. When the input named “Service Chosen” is high indicating a service has been selected, the system proceeds to the state of “Enter Mobile No” to check whether the mobile number is correct or not. If the mobile number is correct which should be equal to “11’b10000010111”, the system advances to the state of “Confirm Service” and transaction is done, otherwise, the system remains in the state of “Enter Mobile No” waiting for the correct mobile no.

The sixth transaction which happens when the “Opcode” is equal to “0101” alongside “Cardless” being equal to ‘0’ is “Money Transfer” representing money transfer between different accounts. The system advances automatically to the next State of “Bank Account and Transfer Value” which represents entering the receiver’s account number and the transfer value of the transaction and then

checks whether the destinated bank account number is correct or not, if it is correct and equal to “16’hD903”, the system proceeds to the state of “Confirm Balance Available”, otherwise, it remains in the same state waiting for the correct account number”. In the state of “Confirm Balance Available”, if the input “Transfer Value” is greater than the existing balance, the transaction is done and the system advances to “Confirm Transfer” state, otherwise, returns back to the previous state of “Bank Account and Transfer Value”.

The seventh transaction which happens when the “Opcode” is equal to “0100” alongside “Cardless” being equal to ‘0’ is “Mini-Bank Statement” representing showing of a Mini-Bank Statement of the user’s latest transactions and the transaction is done in one step.

The eighth and final transaction which happens when the “Opcode” is equal to “1100” alongside “Cardless” being equal to ‘0’ is “Credit Card Settlement” which represents paying money to settle a credit card debt. When the input named “Credit Card Inserted” is high and equal to ‘1’ indicating a credit card is inserted in the ATM, the system advances to the state of “Deposit Desired Settlement Amount” which then advances to the state of “Confirm Settlement” and transaction is done as long as the “Settlement Amount” input is greater than ‘0’.

After completing either of the eight transactions described above, there is an equivalent output that should turn high when the equivalent transaction is completed, the outputs are "Balance Shown”, “Deposited Successfully”, “Withdrawed Successfully”, “Exchanged Successfully”, “Fawry Services Done”, “Transferred Successfully”, “Bank Statement Produced”, “Debt Paid Off” respectively. Also, there is an input named “Receipt” which represents whether a receipt for the transaction is to be printed or not. Consequently, when the “Receipt” is equal to ‘1’ and the system is in the last state of any of the eight

transactions, the system advances to the state of “Print Receipt” then to the state of “One More Transaction”, however if the “Receipt” is equal to ‘0’ indicating a receipt is not required, the system advances to “One More Transaction” directly. At this pre-final state of “One More Transaction”, if the input of “Another Operation” is equal to ‘1’ indicating another transaction is required by the user, the system reverses back to the state of “Another Operation”, but when “One More Transaction” is equal to ‘0’ indicating the transactions required by the user are finished, the system advances to the last state of “Eject Card” representing the ejection of the bank card from the ATM and the output named “ATM Usage Finished” turns to be high indicating that the use of the atm has finished, and then the system returns back to the idle state.

Verification Plan

1. Directed Testing

Firstly, we used direct testing in the testbench to test some transactions using manually entered inputs by us. At the beginning, we initialized all the inputs and then we started changing the inputs according to the transaction to be tested. The first test was the “Withdraw” transaction using a manually input withdrawal value.

```
//Withdraw using manual value test//
Reset_tb                    = 1'b1;
#10
Cardless_tb                 = 1'b0;
#10
Credit_Card_Inserted_tb     = 1'b0;
#10
Language_tb                 = 1'b1;
#10
User_Pass_tb                = 4'b1010;
#10
Opcode_tb                   = 4'b0010;
#10
Quick_List_tb               = 1'b0;
#10
Withdraw_Value_tb           = 32'h00000040;
#10
Manual_Withdraw_Value_tb    = 1'b1;
#10
Receipt_tb                  = 1'b1;
#10
Another_Operation_tb        = 1'b0;
#10
//end of withdraw test
```

The second test was the “View Balance” transaction.

```
//View Balance
Reset_tb                    = 1'b1;
#10
Cardless_tb                 = 1'b0;
#10
Credit_Card_Inserted_tb     = 1'b0;
#10
Language_tb                 = 1'b1;
#10
User_Pass_tb                = 4'b1010;
#10
Opcode_tb                   = 4'b0001;
#10
Receipt_tb                  = 1'b1;
#10
Another_Operation_tb        = 1'b0;
#10
//end of view balance test
```

The third test was the “Deposit” transaction.

```
//Deposit
Reset_tb                    = 1'b1;
#10
Cardless_tb                 = 1'b0;
#10
Language_tb                 = 1'b1;
#10
User_Pass_tb                = 4'b1010;
#10
Opcode_tb                   = 4'b0011;
#10
Money_Deposited_tb          = 1'b1;
#10
Receipt_tb                  = 1'b1;
#10
Another_Operation_tb        = 1'b0;
#10
// end of deposit test
```

The fourth test was the “Exchange Currency” transaction.

```
//Exchange Currency
Reset_tb                    = 1'b1;
#10
Cardless_tb                 = 1'b1;
#10
Language_tb                 = 1'b1;
#10
User_Pass_tb                = 4'b1010;
#10
Opcode_tb                   = 4'b0110;
#10
Currency_Deposited_tb       = 1'b1;
#10
Receipt_tb                  = 1'b1;
#10
Another_Operation_tb        = 1'b0;
#10
// end of Exchange Currency test
```

The fifth test was the “Withdraw” transaction but this time using the quick selection menu.

```
//Withdraw using quick list
Reset_tb                    = 1'b1;
#10
Cardless_tb                 = 1'b0;
#10
Language_tb                 = 1'b1;
#10
User_Pass_tb                = 4'b1010;
#10
Opcode_tb                   = 4'b0010;
#10
Quick_List_tb               = 1'b0;
#10
Withdraw_Value_tb           = 32'h00000040;
#10
Manual_Withdraw_Value_tb    = 1'b1;
#10
Receipt_tb                  = 1'b1;
#10
Another_Operation_tb        = 1'b0;
#10
//end of withdraw using quick list
```

The sixth test was the “Fawry Payments” transaction.

```
//Fawry Payments test
Reset_tb                = 1'b1;
#10
Cardless_tb             = 1'b1;
#10
Language_tb             = 1'b1;
#10
Opcode_tb               = 4'b0111;
#10
Service_Chosen_tb       = 1'b1;
#10
Receipt_tb              = 1'b1;
#10
Another_Operation_tb    = 1'b0;
#10
//end of Fawry Payments test
```

The seventh test was the “Mini-Bank Statement” transaction.

```
//Mini-Bank Statement test
Reset_tb                = 1'b1;
#10
Cardless_tb             = 1'b0;
#10
Language_tb             = 1'b1;
#10
User_Pass_tb            = 4'b1010;
#10
Opcode_tb               = 4'b0100;
#10
Receipt_tb              = 1'b1;
#10
Another_Operation_tb    = 1'b0;
#10
//End of Mini-Bank Statement test
```

The eight test was the “Money Transfer” transaction.

```
//Money Transfer test
Reset_tb                = 1'b1;
#10
Cardless_tb             = 1'b0;
#10
Credit_Card_Inserted_tb     = 1'b0;
#10
User_Pass_tb            = 4'b1010;
#10
Language_tb             = 1'b1;
#10
Opcode_tb               = 4'b0101;
#10
User_Account_No_tb      = 16'hD903;
#10
Transfer_Value_tb       = 32'h00000001;
#10
Receipt_tb              = 1'b1;
#10
Another_Operation_tb    = 1'b0;
#10
//End of Money Transfer test
```

Note : All the directed test cases has passed and verified by tracing the waveform.

2. Randomized Testing

Secondly, we used randomized testing to random the inputs as much as possible where we created two nested for loops. The first loop was to generate the seeds used in the random function for 70 different seeds ranging from 0 till 69 to use different seeds in randomizing to achieve as much diversity as possible and the second for loop was to use the generated seeds in randomizing the inputs for 100 thousand time.

```
//Randomized Testing
    for(j=0 ; j<70 ; j=j+1)
    begin

        seed =j;
        $display("seed is %d" , seed);

            for(i=0 ; i<99999 ; i=i+1)
            begin

                #10
                Reset_tb                =$random(seed);
                Cardless_tb             =$random(seed);
                Language_tb             =$random(seed);
                Money_Deposited_tb      =$random(seed);
                Quick_List_tb           =$random(seed);
                Manual_Withdraw_Value_tb=$random(seed);
                Currency_Deposited_tb   =$random(seed);
                Service_Chosen_tb       =$random(seed);
                Credit_Card_Inserted_tb =$random(seed);
                Receipt_tb              =$random(seed);
                Another_Operation_tb    =$random(seed);
                Opcode_tb               =$random(seed);
                Withdraw_Value_tb       =$random(seed);
                Transfer_Value_tb       =$random(seed);
                Settlement_Amount_tb    =$random(seed);

end
            end
```

3. Assertions

Thirdly, we used psl assertions in the design module to assert certain functionalities in the ATM.

The first assertion was to test the transition between the state of "Enter Value Manually" and the state of "Check Sufficient Funds" included in the flow of the "Withdraw" transaction. The assertion was made to test out that if the current state is "Enter Value Manually" and the input of "Manual Withdraw Value" becomes equal to '1' alongside the "Reset" being equal to '1', the next state should be "Check Sufficient Funds" at the next positive edge of the clock.

```
//psl assert always((((current_state==Enter_Value_Manually)&&Manual_Withdraw_Value)&&Reset) -> next (Check_Sufficient_Funds))@(posedge Clock);
```

The second assertion was to test the transition between the state of "One More Transaction" and the state of "Choose Transaction" which happens when one transaction is finished and another transaction is required. The assertion was made to test out that if the current state is "One More Transaction" and the input of "Another Operation" becomes equal to '1' alongside the "Reset" being equal to '1', the next state should be "Choose Transaction" at the next positive edge of the clock.

```
//psl assert always((((current_state==One_More_Transaction)&&Another_Operation)&&Reset) -> next (Choose_Transaction))@(posedge Clock);
```

The third assertion was to test the transition between the state of "Withdraw" and the state of "Check Sufficient Funds" included in the flow of the "Withdraw" transaction. The assertion was made to test out that if the current state is "Withdraw" and the input of "Quick List" becomes equal to '1' alongside the "Reset" being equal to '1', the next state should be "Check Sufficient Funds" at the next positive edge of the clock.

```
//psl assert always((((current_state==Withdraw)&&Quick_List)&&Reset) -> next (Check_Sufficient_Funds))@(posedge Clock);
```

The fourth and final assertion was to test the state of “Enter Pin or OTP”. The assertion was made to test out that if the current state is “Enter Pin or OTP” and the entered password is incorrect, the system should remain in the same state.

```
//psl assert always((((current_state==Enter_Pin_Or_OTP)&&(Correct_Pass==0)&&Reset)) -> next (Enter_Pin_Or_OTP))@(posedge Clock);
```