	Academic Year:	2022/2023	Term:	spring 2023	
	Course Code:	Elective-4	Course Title:	EDA	

Cairo University
Faculty of Engineering
Electronics and Communications Engineering Department – 4th Year

ATM - based bank system

Submitted to: Dr. Eman

Name	BN	Sec
عمر ايمن عبدالمتعال	13	3
محمد أحمد محمد أبوسعدة	32	3
محمد ايمن عبدالعليم احمد	34	3
محمد عادل كامل	35	3
مينا حنا فايز وهبة	35	4

Table of Contents

Abstract

The ATM System is the project which is used to access their bank accounts to make cash withdrawals and other transactions. Whenever user wants to make transaction, he can enter their ATM card and verified PIN, then menu of options will appear to choose the appropriate option. ATM System has capability to enter the withdrawal amounts and deposit amounts of money, display the account balance.

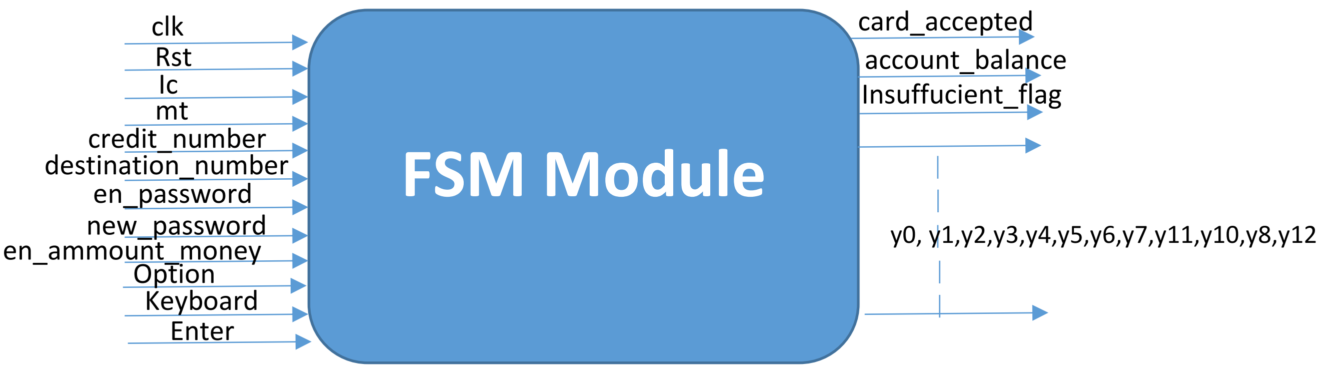
Introduction

The automated teller machine (ATM) is an automatic banking machine which allows the user to complete basic transactions without any help from bank representatives. There are two types of automated teller machines. The basic one allows the customer to only draw cash and receive a report of the account balance. Another one is a more complex machine which accepts deposits, provides credit card payment facilities, and reports account information.

The user is also able to perform one or more transactions. Security is the foundation of a good ATM system. This system will provide for secure connections between users and the bank servers. The whole process will be automated right from PIN validation to transaction completion. The card details and PIN database will be a secure module that will not be open to routine maintenance, the only possibility of access to this database will be through queries(questions) raised from an ATM in the presence of a valid bank ATM card. ATM Simulation System will enable two important features of an ATM, reduction of human error in the banking system and the possibility of24 hour personal banking. In earlier years all the transactions were to be done manually, still done but very rarely, as it is a very difficult task. So now banks use this to give their customers easy and faster transactions. This makes transactions easier and faster.

In this project we are implementing the core of the bank ATM design as well as verification environment.

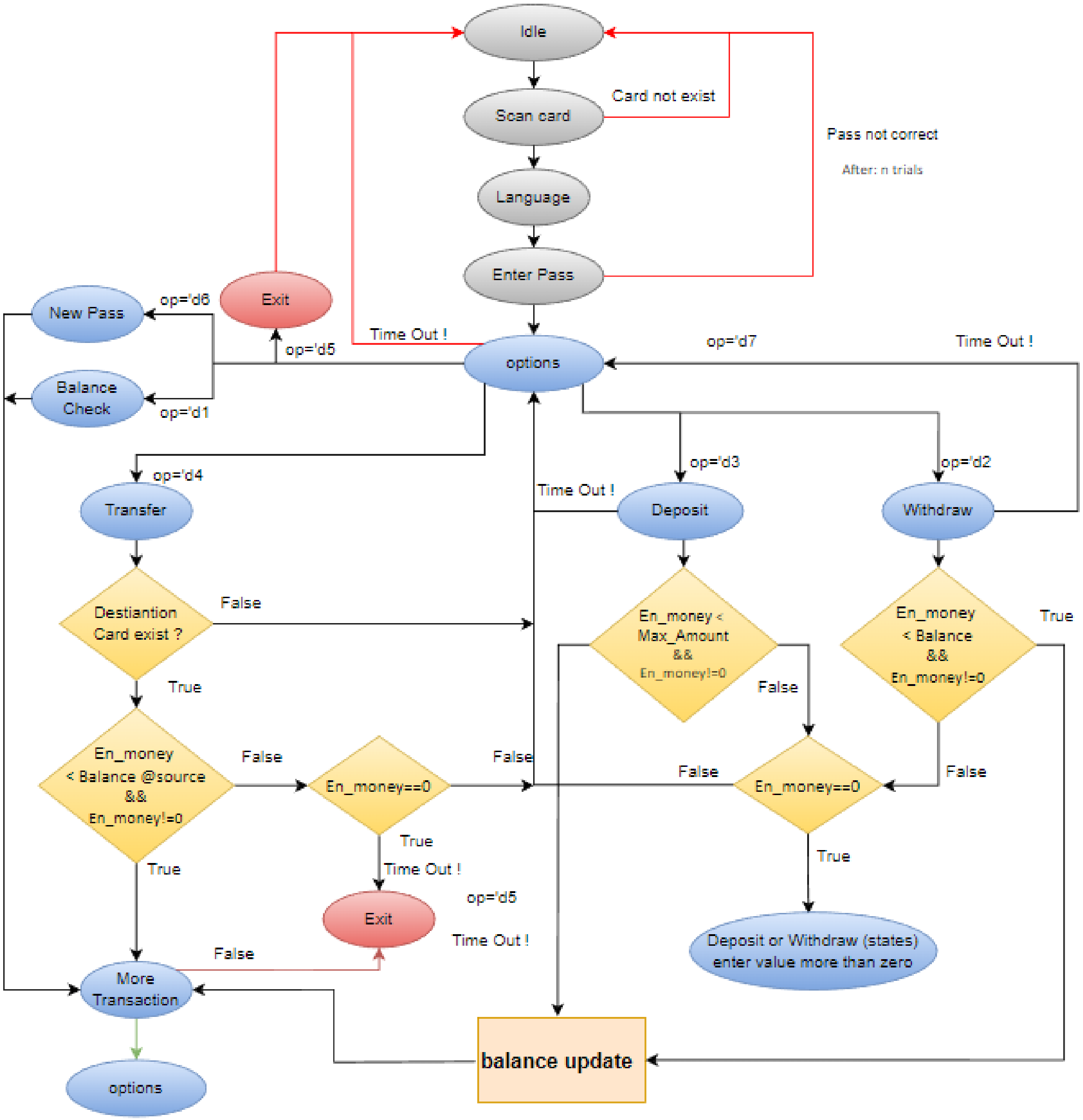
FSM Module



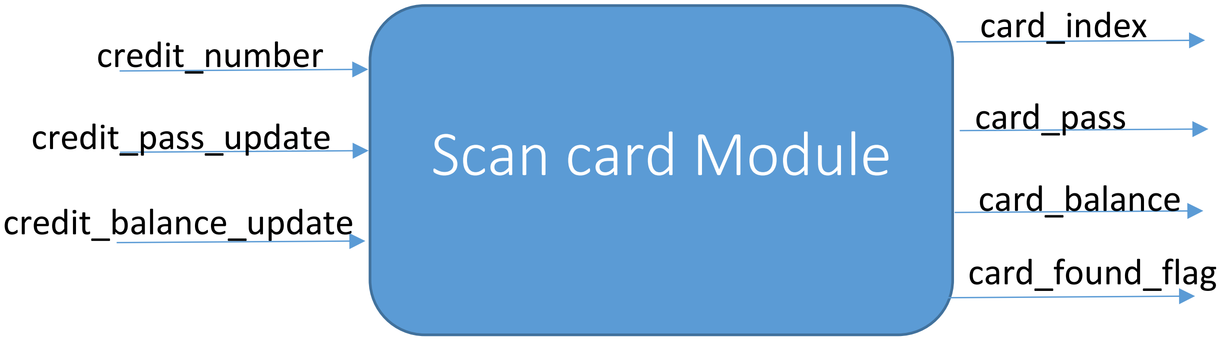
Signal	Type	Width	Description
Clk	**Input**	1	Clock of the system
Rst	**Input**	1	Reset
Ic	**Input**	1	Insert card
mt	**Input**	1	More transaction
credit_number	**Input**	64	Number of credit
destination_number	**Input**	64	Destination number
en_password	**Input**	16	Enter password
new_password	**Input**	16	New password
en_ammount_money	**Input**	16	Enter amount of money
Option	**Input**	3	Screen options
Keyboard	**Input**	4	Keyboard for enter the different options
Enter	**Input**	1	Enter key
card_accepted	**Output**	1	Accepted the card
account_balance	**Output**	16	Account balance
Insuffucient_flag	**Output**	1	Withdraw value more than account balance
y0,y1,y2,y3,y4,y5,y6,y7,y11,y10,y8,y12	**Output**	1	Flags to show the current state

FSM Diagram

Figure 1 FSM diagram



Scan card Module (Behavioral model)



Signal	Type	Width	Description
credit_number	**Input**	64	Number of credit
credit_pass_update	**Input**	16	Update of old password
credit_balance_update	**Input**	14	Update of amount of balance
card_index	**Output**	4	Index of card
card_pass	**Output**	16	Password of the card
card_balance	**Output**	14	Amount of balance
card_found_flag	**Output**	1	Flag refer to that the card that enter is founded

Note:

Behavioral model to act as ATM database.

States of FSM:

S0 (idle state)

The idle state nothing happened on it, and it is the state that I return to it at the end if:

1. The card does not exist.
2. Enter wrong password.
3. Time out and the user does not use any option
4. If enter exit option

S8 (scan card)

The scan card state responsible for handling card and check its existence and get its information from server data base (account balance and pass)

S7 (lang used)

The lang used state decides the language to be used through the transaction. The user in This state selects the required language by pressing any number in the keyboard input.

S9 (enter pass)

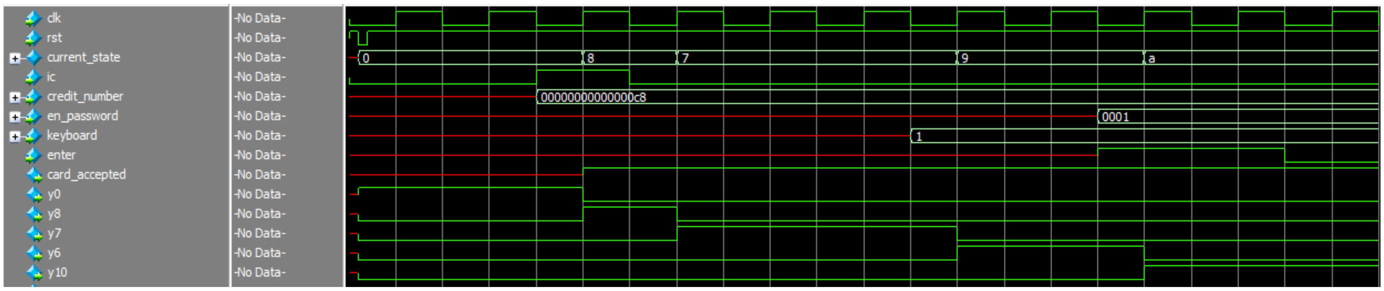
The enter pass state responsible for comparing entered pass with pass from database.

S10 (option select)

The option select state responsible for choosing the desired options:
(Withdraw – deposit- balance check-new pass- transfer -exit – anything else).

Operation of previous states:

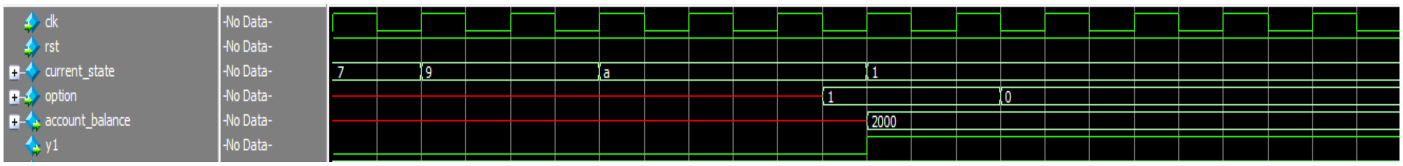
1-First, the operation started from idle state (S0)
2-The user pressed insert card (IC) input and entered card value (card_number)
3-The operation transferred to (scan_card) state to check the existence of the card. (S8)
4-The card number was correct so now we are in (lang_used) state we pressed 1 in the keyboard to select Arabic. (S7)
5-we then move to (enter_pass) state we write the password and press enter if it is true, we move to option select if wrong we have number of trials that we should not exceed. (S9 to S10)
-y0, y8,y7,y6,y10 are output flags indicating which state we are at now.



S1 (balance check)

The balance check state gets all information about updated account balance.

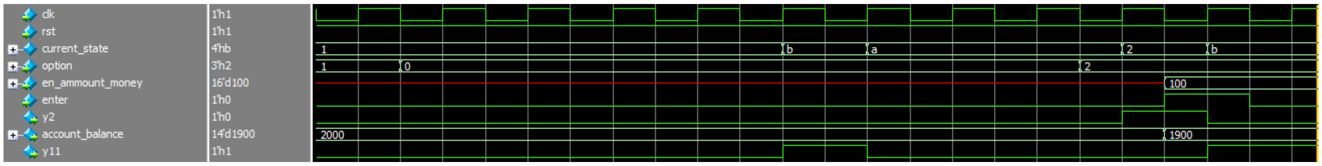
1-During (select_option) state we enter option 1 to select check balance.
2-The output account balance will take the value of the entered card number.
3-y1 flag indicates that we are in this state.



S2 (Withdraw)

The withdraw state gets account balance deducted by the desired entered amount of money. If balance account has the capacity to make this withdraw operation if not go to option state to check account balance.

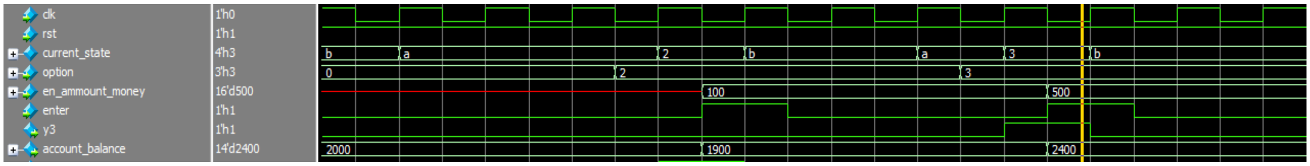
1-From (select_option) state we enter option 2 to select (withdraw) S2.
2-Enter the amount of money(en_amount_money) and press (enter) to save it.
3-output (account_balance) will change in the waveform= 2000-100=1900.but the Withdrawal value must be less than the account balance.
4-y2 flag indicates that we are in this state.



S3 (deposit)

The deposit state gets account balanced increased by the desired entered amount of money. If entered amount has exceed max value of deposit operation go to option state

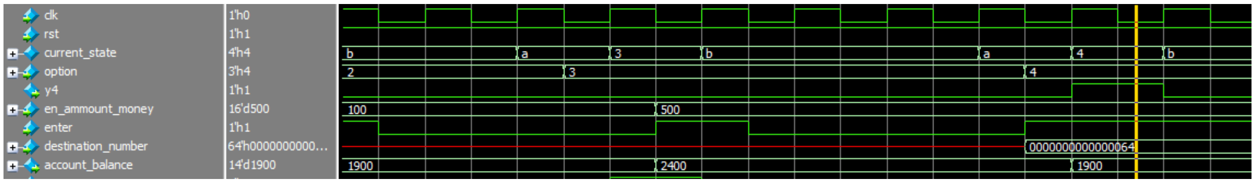
1-From (select_option) state we enter option 3 to select (deposit) S3.
2-Enter the amount of money(en_amount_money) and press (enter) to save it.
3-output (account_balance) will change in the waveform= 2000+500=2400.but the deposit value must be less than the deposit max value.
4-y3 flag indicates that we are in this state.



S4 (transfer)

The transfer state checks the existence of destination account and perform withdraw operation on source account and deposit operation to destination account.

1-From (select_option) state we enter option 4 to select (transfer) S4.
2-Enter the amount of money(en_amount_money) and destination card number and press (enter) to save it.
3-output (account_balance) will change in the waveform= 2400-500=1900.but the deposit value must be less than the deposit max value.
4-y4 flag indicates that we are in this state.

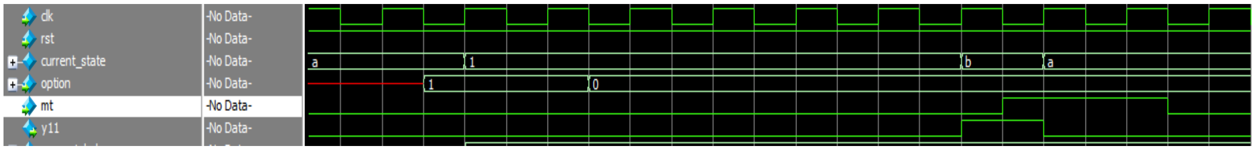


clk	1'h0
rst	1'h1
current_state	4'h4
option	3'h4
y4	1'h1
en_ammount_money	16'd500
enter	1'h1
destination_number	64'h0000000000...
account_balance	14'd1900

S11 (anything else)

The anything else state to let opportunities for user to do more transaction or not.

1-After (check_balance) state we go to (anything_else) state.
2-we press mt (more transaction) if we want to continue and choose another option.
3-y11 flag indicates that we are in this state.



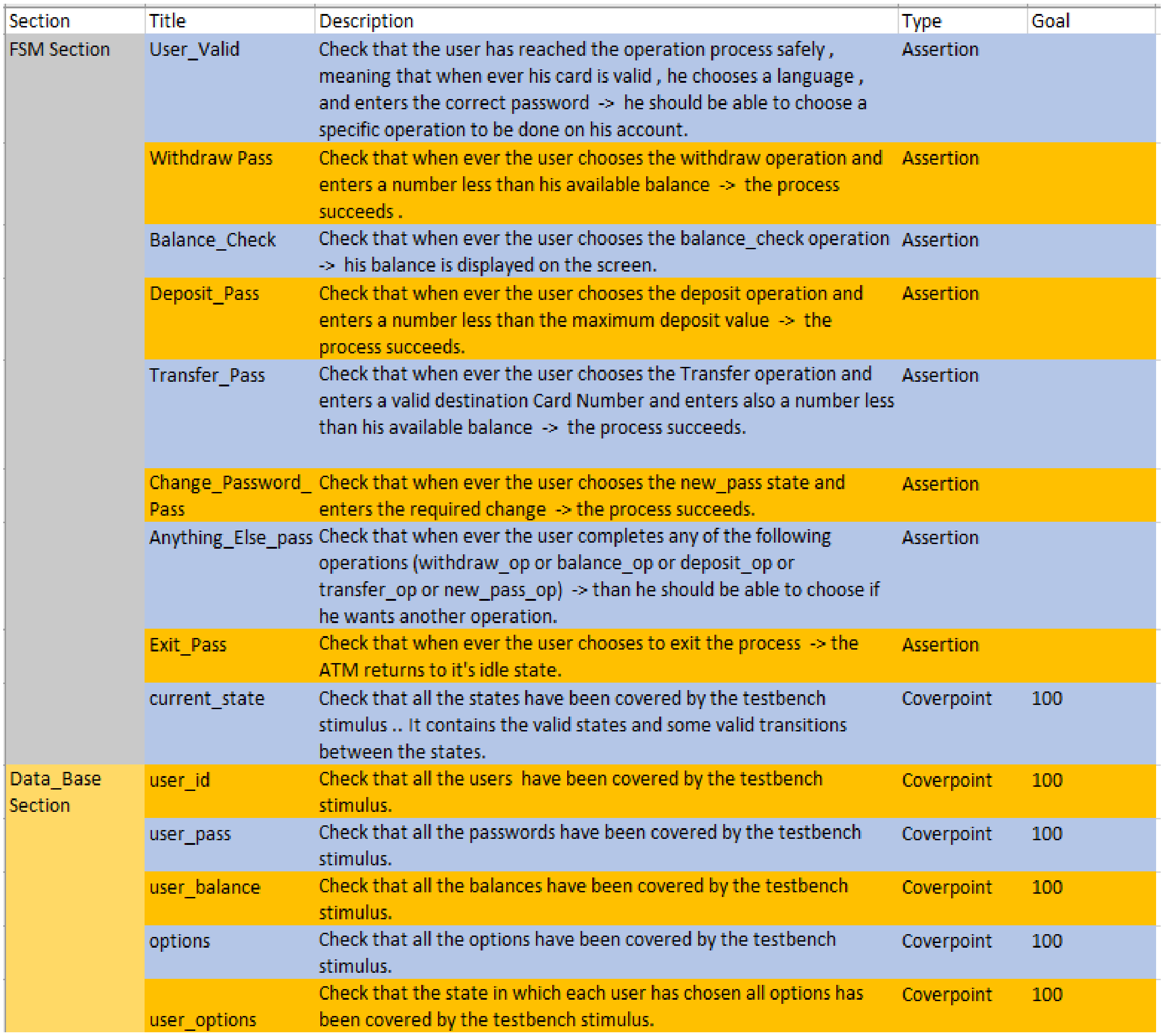
clk	-No Data-
rst	-No Data-
current_state	-No Data-
option	-No Data-
mt	-No Data-
y11	-No Data-

S5 (exit)

The exit state option to finish transaction and go to idle state.

Verification Plan

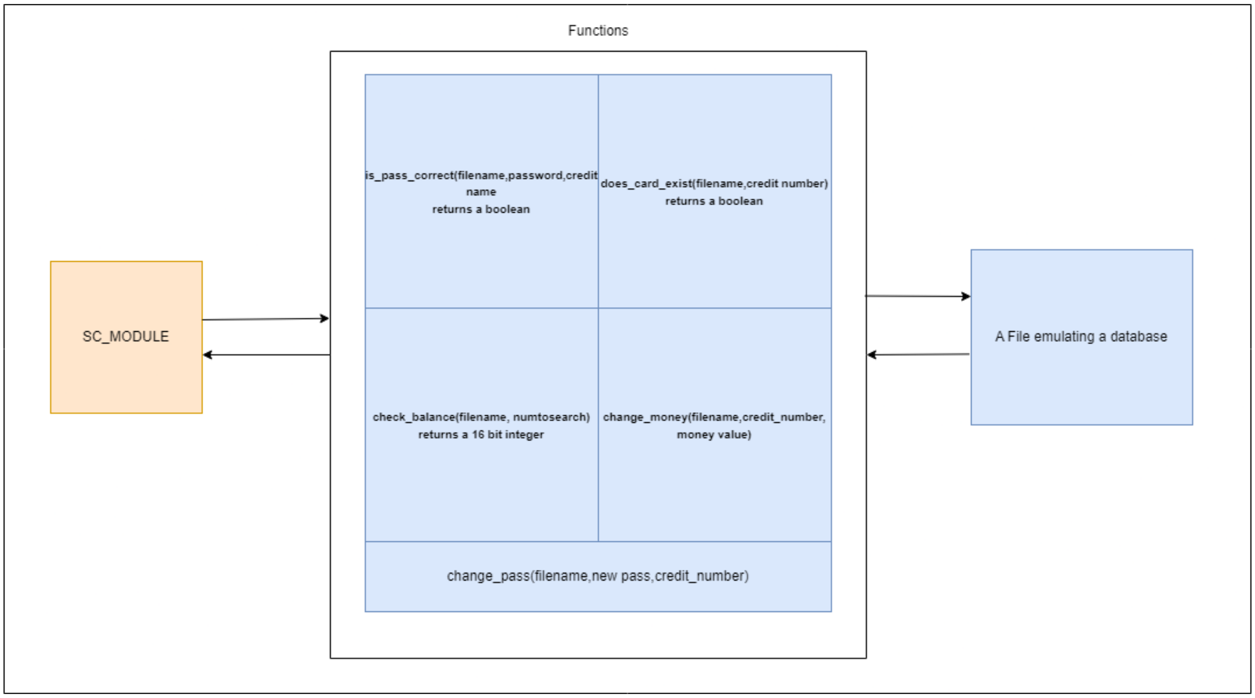
After reading the design specs we wrote a plan for the verification process that would guarantee that the design performs all required operations correctly by using the assertion propertied mentioned below as well as the cover states.



Section	Title	Description	Type	Goal
FSM Section	User_Valid	Check that the user has reached the operation process safely , meaning that when ever his card is valid , he chooses a language , and enters the correct password -> he should be able to choose a specific operation to be done on his account.	Assertion	
	Withdraw Pass	Check that when ever the user chooses the withdraw operation and enters a number less than his available balance -> the process succeeds .	Assertion	
	Balance_Check	Check that when ever the user chooses the balance_check operation -> his balance is displayed on the screen.	Assertion	
	Deposit_Pass	Check that when ever the user chooses the deposit operation and enters a number less than the maximum deposit value -> the process succeeds.	Assertion	
	Transfer_Pass	Check that when ever the user chooses the Transfer operation and enters a valid destination Card Number and enters also a number less than his available balance -> the process succeeds.	Assertion	
	Change_Password_Pass	Check that when ever the user chooses the new_pass state and enters the required change -> the process succeeds.	Assertion	
	Anything_Else_pass	Check that when ever the user completes any of the following operations (withdraw_op or balance_op or deposit_op or transfer_op or new_pass_op) -> than he should be able to choose if he wants another operation.	Assertion	
	Exit_Pass	Check that when ever the user chooses to exit the process -> the ATM returns to it's idle state.	Assertion	
	current_state	Check that all the states have been covered by the testbench stimulus .. It contains the valid states and some valid transitions between the states.	Coverpoint	100
Data_Base Section	user_id	Check that all the users have been covered by the testbench stimulus.	Coverpoint	100
	user_pass	Check that all the passwords have been covered by the testbench stimulus.	Coverpoint	100
	user_balance	Check that all the balances have been covered by the testbench stimulus.	Coverpoint	100
	options	Check that all the options have been covered by the testbench stimulus.	Coverpoint	100
	user_options	Check that the state in which each user has chosen all options has been covered by the testbench stimulus.	Coverpoint	100

Reference model

The Reference model was designed based on the previous implantation , it was designed using c++ and systemc library, it have the components as following.


Functions
is_pass_correct(filename,password,credit name
returns a boolean
does_card_exist(filename,credit number)
returns a boolean
check_balance(filename, numtosearch)
returns a 16 bit integer
change_money(filename,credit_number, money value)
change_pass(filename,new pass,credit_number)
SC_MODULE
A File emulating a database

The main module of ATM is defined using systemc macro and have the same states as mentioned before, there is an external file which contains all the credit details and used to emulate the database and the ATM module is interfacing with the file using those functions.

does_card_exist()

checks whether the card exists or not

Is_pass_correct()

It checks if the password is correct, it takes filename, password and credit numberand returns a bool which tells if the password is correct or not.

Check_balance()

This returns the current balance for a credit number.

Change_money()

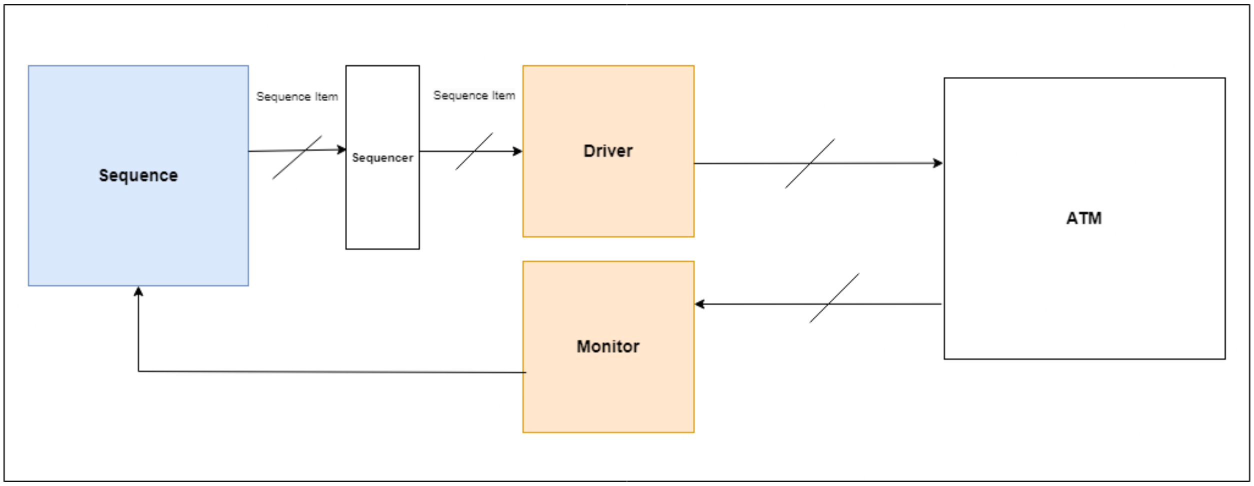
This can change the money balance In a user account whether in a deposit , transfer or withdraw operation.

Change_pass()

Used to change password for a given new password

Verification Process:

The verification Process starts with creating an input Stimulus, that could cover all the possible states and this was done using the UVM methodology and system Verilog classes but making a complete randomization would take a lot of simulation time and is mostly wouldn't get us to the wanted coverage not even close to it so a constraint randomization was made so that we should be able to attend this coverage but how the hierarchy of injecting the stimulus was made.



The sequence first generates a sequence item to send it to the DUT this sequence item is a class which contains all the signals in the designed and it is randomized or not, the following signals that are in an instance of sequence item of a class is as following

Signal	Type	Constraints
rst	Input - direct only	No constraints
ic	Input - direct only	No constraints
enter	Input - direct only	No constraints
mt	Input- Randomized	No constraints
credit_number, destination_number	Input - Randomized	Constraints inside a given array of Credit Numbers
en_password	Input - Randomized	Constraints inside a given array of passwords with probability of success higher
new_password	Input - Randomized	No constraints
en_ammount_money	Input – Randomized	Constraints to be less than 10,000 EGP
option	Input - Randomized	No constraints
keyboard	Input – Randomized	No constraints

And the rest of the signals are defined as static output but why its static and why there is a connection between the monitor and the sequence class, that connection is actually a flag based that the monitor rises every time it reads values from the DUT and put in the output of the sequence item class and based on the state of the machine the sequence or the generator decides which constraint randomization values should be put.

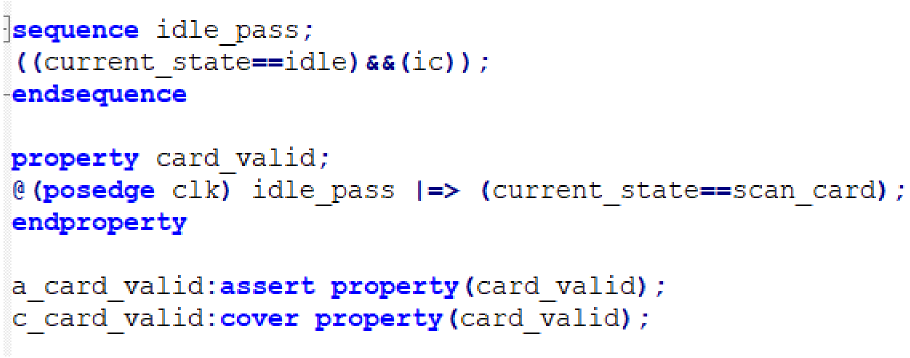
But what are the sequence that the generator could possibly generate, the following table represent the sequences which can the sequence generate.

First state	2nd state	3rd state	4th state	5th state	6th state	7th state	8th state	Probability
Idle	Scan card	Enter lang	Enter pass	Enter option	Balance check	Anything else	Enter option	Normal
Idle	Scan card	Enter lang	Enter pass	Enter option	Balance check	Anything else	Idle	Normal
Idle	Scan card	Enter lang	Enter pass	Enter option	withdraw	Anything else	Enter option	Normal
Idle	Scan card	Enter lang	Enter Pass	Enter option	withdraw	Idle		Normal
Idle	Scan card	Enter lang	Enter pass	Enter option	withdraw	Anything else		Normal
Idle	Scan card	Enter lang	Enter pass	Enter option	New pass	Anything else	Enter option	Normal
Idle	Scan card	Enter lang	Enter pass	Enter option	New pass	Anything else	Idle	Normal
Idle	Scan card	Enter lang	Enter pass	Enter option	New pass	Idle		Low -may not occure
Idle	Scan card	Enter lang	Enter pass	Enter option	deposit	Anything else	Enter option	Normal
Idle	Scan card	Enter lang	Enter pass	Enter option	deposit	Idle		Normal
Idle	Scan card	Enter lang	Enter pass	Enter option	deposit	Anything else	Idle	Normal
Idle	Scan card	Idle						Not applied
Idle	Scan card	Enter lang	Enter pass	Enter option	Idle			Low – may not occure
Idle	Scan card	Enter lang	Enter pass	Enter option	Transfer op	Anything else	Idle	Normal
Idle	Scan card	Enter lang	Enter pass	Enter option	Transfer op	Idle		Normal
Idle	Scan card	Enter lang	Enter pass	Enter option	Transfer op	Anything else	Enter option	Normal
Idle	Scan card	Enter lang	Enter pass	Enter option	Exit	Idle		Normal

Assertions Section:

We have written 12 assertions properties that verify the control unit which is a Finite State Machine. We wrote properties that check for each and every state that whenever the right conditions occur this state does it’s operation correctly and goes to the expected next state.

Example:



```
sequence idle_pass;
 ((current_state==idle)&&(ic));
endsequence

property card_valid;
@(posedge clk) idle_pass |=> (current_state==scan_card);
endproperty

a_card_valid:assert property(card_valid);
c_card_valid:cover property(card_valid);
```

This is the most simple property , it checks that whenever the current state is idle and a user inserts his card, we move to the scan card state.

Problems and Solutions:

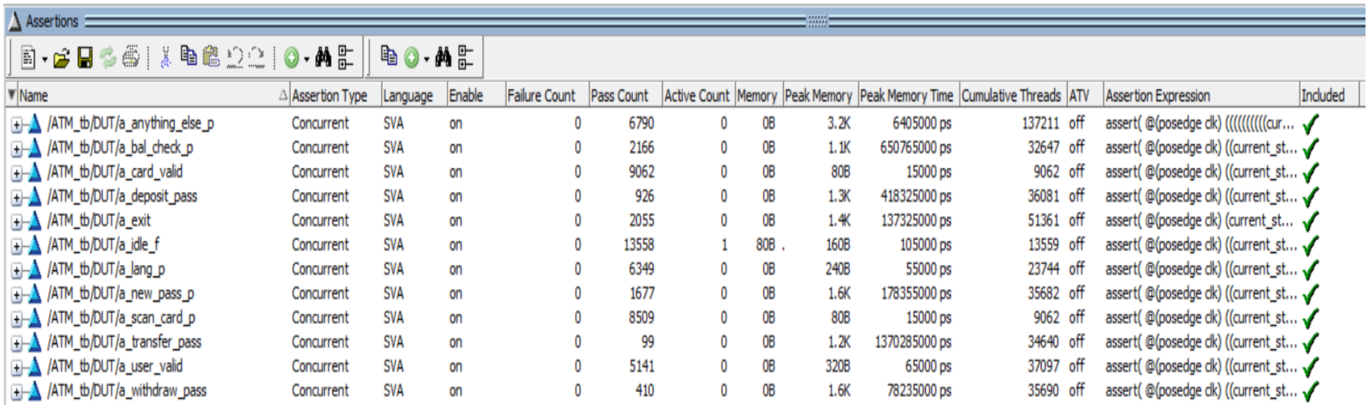
Problem_1: At the beginning we wrote large and overlapping properties, which were difficult to read and debug.

Solution: We followed some guidelines written in the lecture slides which solved the situation such as:

- Keep sequences and properties as simple as possible.
- Use sequences to simplify properties.
- Build complex properties out of simple, and short sequences.

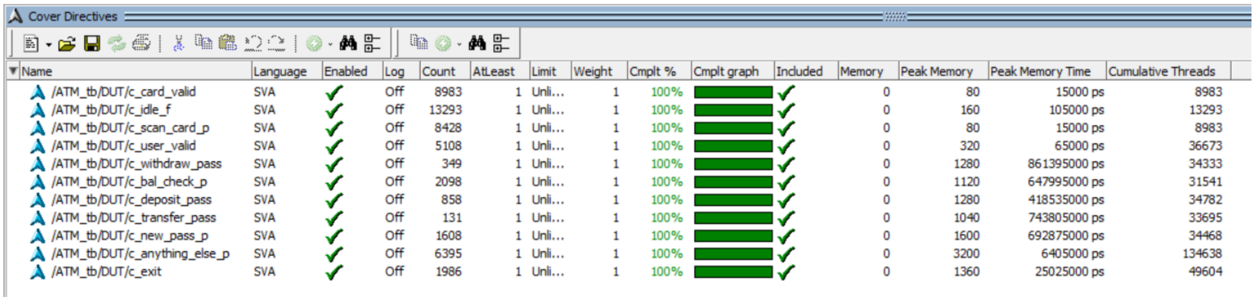
Assertions Results:

We have 12 assertion properties .. All of them passed ..



Name	Assertion Type	Language	Enable	Failure Count	Pass Count	Active Count	Memory	Peak Memory	Peak Memory Time	Cumulative Threads	ATV	Assertion Expression	Included
/ATM_tb/DUT/a_anything_else_p	Concurrent	SVA	on	0	6790	0	0B	3.2K	6405000 ps	137211	off	assert(@(posedge clk) (((((((((cur...	✓
/ATM_tb/DUT/a_bal_check_p	Concurrent	SVA	on	0	2166	0	0B	1.1K	650765000 ps	32647	off	assert(@(posedge clk) ((current_st...	✓
/ATM_tb/DUT/a_card_valid	Concurrent	SVA	on	0	9062	0	0B	80B	15000 ps	9062	off	assert(@(posedge clk) ((current_st...	✓
/ATM_tb/DUT/a_deposit_pass	Concurrent	SVA	on	0	926	0	0B	1.3K	418325000 ps	36081	off	assert(@(posedge clk) ((current_st...	✓
/ATM_tb/DUT/a_exit	Concurrent	SVA	on	0	2055	0	0B	1.4K	137325000 ps	51361	off	assert(@(posedge clk) (current_st...	✓
/ATM_tb/DUT/a_idle_f	Concurrent	SVA	on	0	13558	1	80B	160B	105000 ps	13559	off	assert(@(posedge clk) ((current_st...	✓
/ATM_tb/DUT/a_lang_p	Concurrent	SVA	on	0	6349	0	0B	240B	55000 ps	23744	off	assert(@(posedge clk) ((current_st...	✓
/ATM_tb/DUT/a_new_pass_p	Concurrent	SVA	on	0	1677	0	0B	1.6K	178355000 ps	35682	off	assert(@(posedge clk) ((current_st...	✓
/ATM_tb/DUT/a_scan_card_p	Concurrent	SVA	on	0	8509	0	0B	80B	15000 ps	9062	off	assert(@(posedge clk) ((current_st...	✓
/ATM_tb/DUT/a_transfer_pass	Concurrent	SVA	on	0	99	0	0B	1.2K	1370285000 ps	34640	off	assert(@(posedge clk) ((current_st...	✓
/ATM_tb/DUT/a_user_valid	Concurrent	SVA	on	0	5141	0	0B	320B	65000 ps	37097	off	assert(@(posedge clk) ((current_st...	✓
/ATM_tb/DUT/a_withdraw_pass	Concurrent	SVA	on	0	410	0	0B	1.6K	78235000 ps	35690	off	assert(@(posedge clk) ((current_st...	✓

We ensured that all of them have been covered as well to make sure there are vacuous states ..

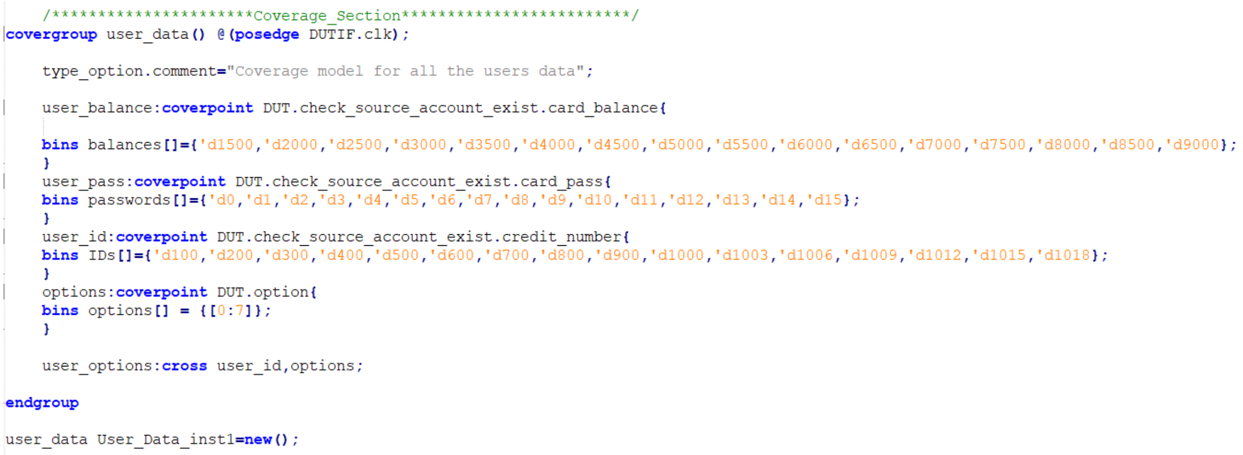


Name	Language	Enabled	Log	Count	AtLeast	Limit	Weight	Cmplt %	Cmplt graph	Included	Memory	Peak Memory	Peak Memory Time	Cumulative Threads
/ATM_tb/DUT/c_card_valid	SVA	✓	Off	8983	1	Unli...	1	100%	🟩	✓	0	80	15000 ps	8983
/ATM_tb/DUT/c_idle_f	SVA	✓	Off	13293	1	Unli...	1	100%	🟩	✓	0	160	105000 ps	13293
/ATM_tb/DUT/c_scan_card_p	SVA	✓	Off	8428	1	Unli...	1	100%	🟩	✓	0	80	15000 ps	8983
/ATM_tb/DUT/c_user_valid	SVA	✓	Off	5108	1	Unli...	1	100%	🟩	✓	0	320	65000 ps	36673
/ATM_tb/DUT/c_withdraw_pass	SVA	✓	Off	349	1	Unli...	1	100%	🟩	✓	0	1280	861395000 ps	34333
/ATM_tb/DUT/c_bal_check_p	SVA	✓	Off	2098	1	Unli...	1	100%	🟩	✓	0	1120	647995000 ps	31541
/ATM_tb/DUT/c_deposit_pass	SVA	✓	Off	858	1	Unli...	1	100%	🟩	✓	0	1280	418535000 ps	34782
/ATM_tb/DUT/c_transfer_pass	SVA	✓	Off	131	1	Unli...	1	100%	🟩	✓	0	1040	743805000 ps	33695
/ATM_tb/DUT/c_new_pass_p	SVA	✓	Off	1608	1	Unli...	1	100%	🟩	✓	0	1600	692875000 ps	34468
/ATM_tb/DUT/c_anything_else_p	SVA	✓	Off	6395	1	Unli...	1	100%	🟩	✓	0	3200	6405000 ps	134638
/ATM_tb/DUT/c_exit	SVA	✓	Off	1986	1	Unli...	1	100%	🟩	✓	0	1360	25025000 ps	49604

Coverage Section:

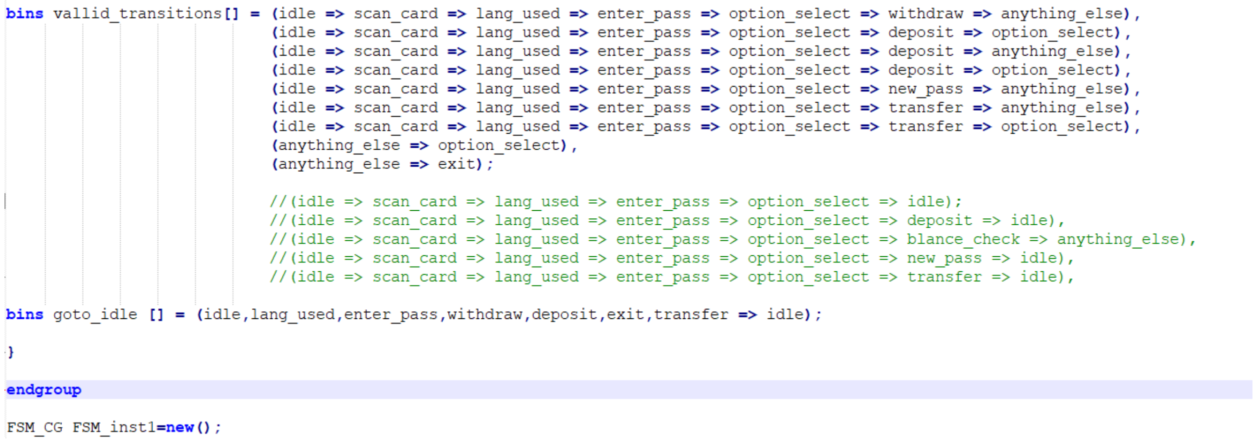
We have 2 covergroups , below is a brief discussion about each one:

1. User_Data Covergroup: In this covergroup we check that all the data in the users database have been accessed at least once by the testbench stimulus .. This makes us sure that all users have entered their credit numbers correctly and entered a correct password for their accounts and lastly checked their balances correctly ..



```
    /**********************Coverage_Section************************/
covergroup user_data() @(posedge DUTIF.clk);

    type_option.comment="Coverage model for all the users data";

    user_balance:coverpoint DUT.check_source_account_exist.card_balance{

    bins balances[]={'d1500,'d2000,'d2500,'d3000,'d3500,'d4000,'d4500,'d5000,'d5500,'d6000,'d6500,'d7000,'d7500,'d8000,'d8500,'d9000};
    }
    user_pass:coverpoint DUT.check_source_account_exist.card_pass{
    bins passwords[]={'d0,'d1,'d2,'d3,'d4,'d5,'d6,'d7,'d8,'d9,'d10,'d11,'d12,'d13,'d14,'d15};
    }
    user_id:coverpoint DUT.check_source_account_exist.credit_number{
    bins IDs[]={'d100,'d200,'d300,'d400,'d500,'d600,'d700,'d800,'d900,'d1000,'d1003,'d1006,'d1009,'d1012,'d1015,'d1018};
    }
    options:coverpoint DUT.option{
    bins options[] = {[0:7]};
    }

    user_options:cross user_id,options;

endgroup

user_data User_Data_inst1=new();
```

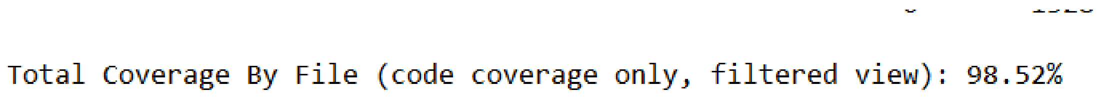
2. FSM_Covergroup: In this covergroup we check that all states have been covered by the testbench stimulus , as well as some transitions that we care about.



```
bins vallid_transitions[] = (idle => scan_card => lang_used => enter_pass => option_select => withdraw => anything_else),
                            (idle => scan_card => lang_used => enter_pass => option_select => deposit => option_select),
                            (idle => scan_card => lang_used => enter_pass => option_select => deposit => anything_else),
                            (idle => scan_card => lang_used => enter_pass => option_select => deposit => option_select),
                            (idle => scan_card => lang_used => enter_pass => option_select => new_pass => anything_else),
                            (idle => scan_card => lang_used => enter_pass => option_select => transfer => anything_else),
                            (idle => scan_card => lang_used => enter_pass => option_select => transfer => option_select),
                            (anything_else => option_select),
                            (anything_else => exit);

                            //(idle => scan_card => lang_used => enter_pass => option_select => idle);
                            //(idle => scan_card => lang_used => enter_pass => option_select => deposit => idle),
                            //(idle => scan_card => lang_used => enter_pass => option_select => blance_check => anything_else),
                            //(idle => scan_card => lang_used => enter_pass => option_select => new_pass => idle),
                            //(idle => scan_card => lang_used => enter_pass => option_select => transfer => idle),

bins goto_idle [] = (idle,lang_used,enter_pass,withdraw,deposit,exit,transfer => idle);

}

endgroup

FSM_CG FSM_inst1=new();
```

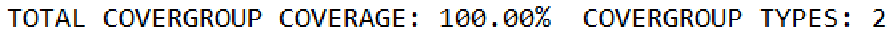
Coverage Results:

With respect to the code coverage , the percentage got from the coverage report is as follows:



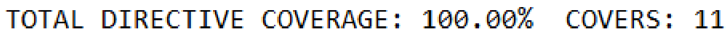
```
Total Coverage By File (code coverage only, filtered view): 98.52%
```

With respect to the total Functional Coverage the results got from the coverage report is as follows:



```
TOTAL COVERGROUP COVERAGE: 100.00%  COVERGROUP TYPES: 2
```

With respect to the assertion properties coverage, the results got from the coverage report are as follows:



```
TOTAL DIRECTIVE COVERAGE: 100.00%  COVERS: 11
```

Pre_100%_Coverage:

At the beginning the coverage percentage was not 100 % , we found that some transitions in the FSM have not been covered by the testbench stimulus, so we changed the input constraints in the testbench to solve this problem, and at some other cases we changed the priority of some inputs that would cover the uncovered states.