CSE312: Electronic Design Automation Fall 2022

Project (1): ATM - based bank system

Presented to:

Dr. Eman Mohamed

Eng. Adham

Submitted by:

Abdulrahman Maged Abdulrahman: 20P7467

Hazem Zaki Aboukhalil : 20P7516

Mariam Hany Fouad: 20P8966

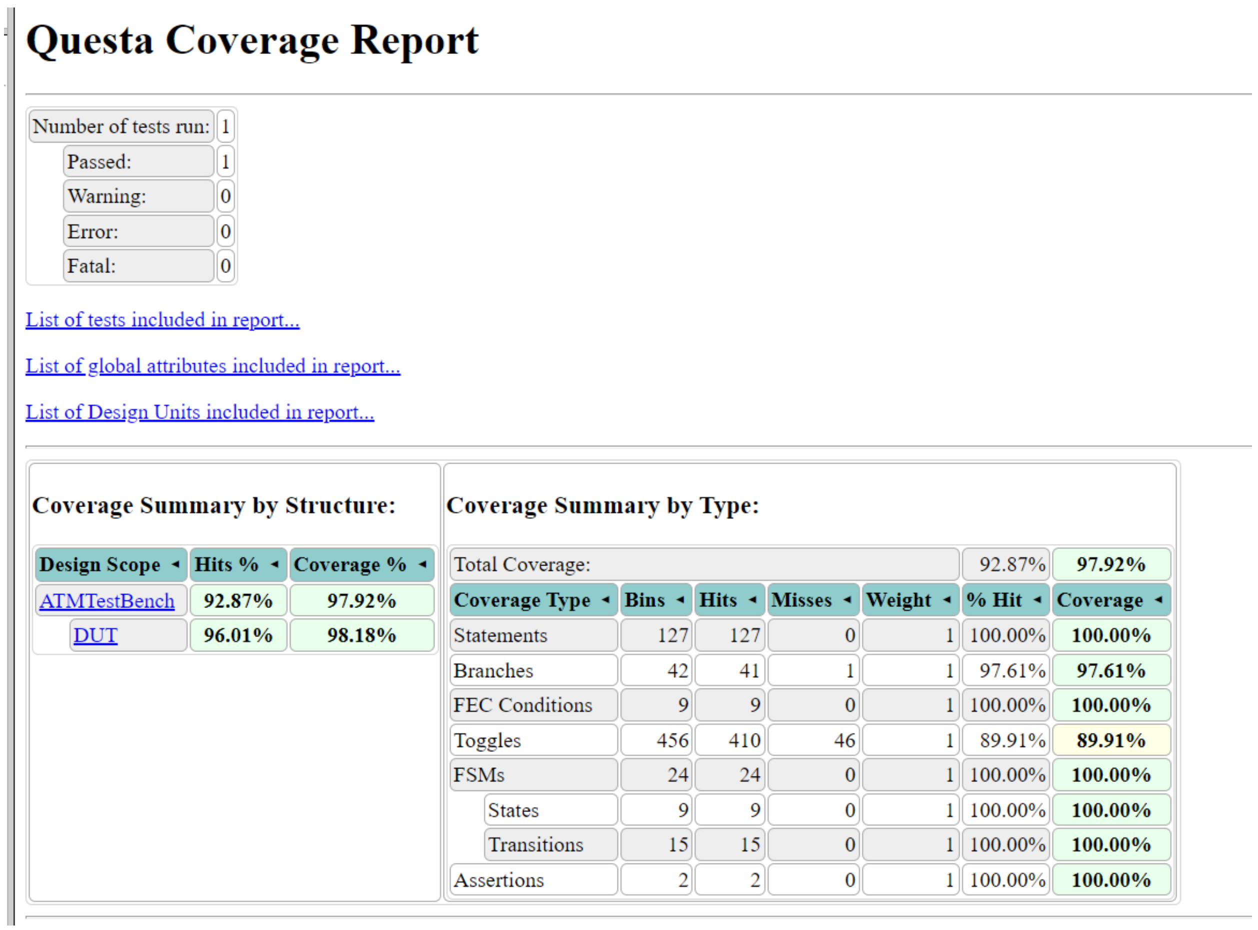
Mina George Fawzy Girgis: 20P4198

Shady Emad Sabry: 20P7239

Matthew sherif shalaby : 20p6785

Karim Bassel Samir: 20P6794

fady fady fouad : 20p7341



Questa Coverage Report

Number of tests run:	1
Passed:	1
Warning:	0
Error:	0
Fatal:	0

List of tests included in report...

List of global attributes included in report...

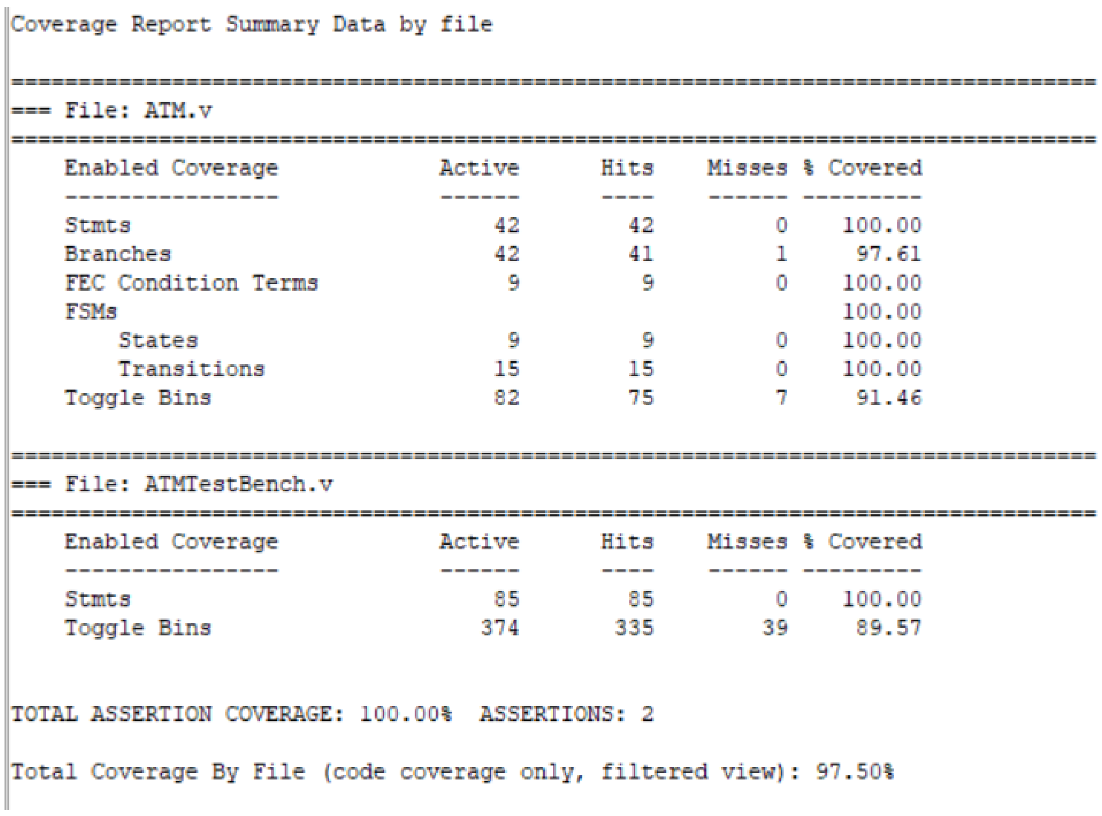
List of Design Units included in report...

Coverage Summary by Structure:

Design Scope	Hits %	Coverage %
ATMTestBench	92.87%	97.92%
DUT	96.01%	98.18%

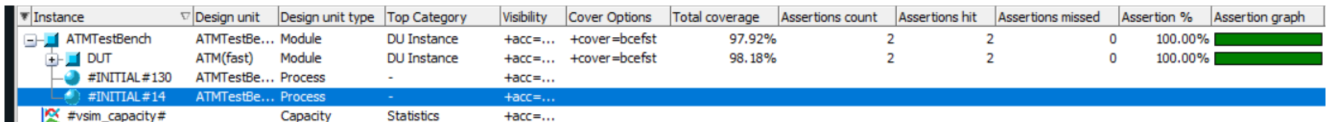
Coverage Summary by Type:

Total Coverage:					92.87%	97.92%
Coverage Type	**Bins**	**Hits**	**Misses**	**Weight**	**% Hit**	**Coverage**
Statements	127	127	0	1	100.00%	100.00%
Branches	42	41	1	1	97.61%	97.61%
FEC Conditions	9	9	0	1	100.00%	100.00%
Toggles	456	410	46	1	89.91%	89.91%
FSMs	24	24	0	1	100.00%	100.00%
States	9	9	0	1	100.00%	100.00%
Transitions	15	15	0	1	100.00%	100.00%
Assertions	2	2	0	1	100.00%	100.00%



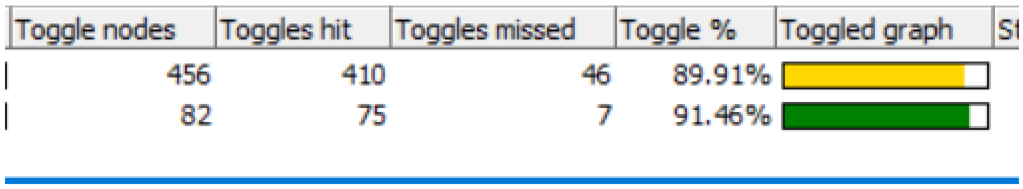
```
Coverage Report Summary Data by file

================================================================================
=== File: ATM.v
================================================================================
    Enabled Coverage            Active      Hits    Misses % Covered
    ----------------            ------      ----    ------ ---------
    Stmts                           42        42         0    100.00
    Branches                        42        41         1     97.61
    FEC Condition Terms              9         9         0    100.00
    FSMs                                                      100.00
        States                       9         9         0    100.00
        Transitions                 15        15         0    100.00
    Toggle Bins                     82        75         7     91.46

================================================================================
=== File: ATMTestBench.v
================================================================================
    Enabled Coverage            Active      Hits    Misses % Covered
    ----------------            ------      ----    ------ ---------
    Stmts                           85        85         0    100.00
    Toggle Bins                    374       335        39     89.57


TOTAL ASSERTION COVERAGE: 100.00%  ASSERTIONS: 2

Total Coverage By File (code coverage only, filtered view): 97.50%
```

Assertion Coverage



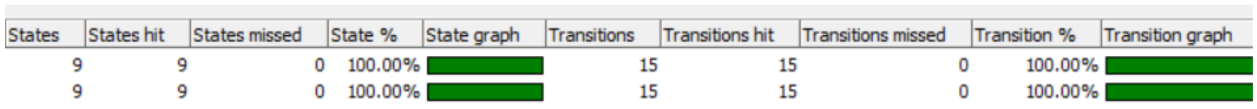
Instance	Design unit	Design unit type	Top Category	Visibility	Cover Options	Total coverage	Assertions count	Assertions hit	Assertions missed	Assertion %	Assertion graph
ATMTestBench	ATMTestBe…	Module	DU Instance	+acc=…	+cover=bcefst	97.92%	2	2	0	100.00%	🟩
DUT	ATM(fast)	Module	DU Instance	+acc=…	+cover=bcefst	98.18%	2	2	0	100.00%	🟩
#INITIAL#130	ATMTestBe…	Process	-	+acc=…							
#INITIAL#14	ATMTestBe…	Process	-	+acc=…							
#vsim_capacity#		Capacity	Statistics	+acc=…							

Toggle Coverage



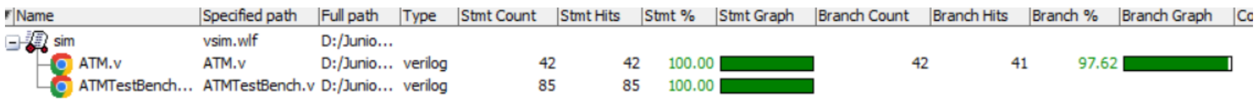
Toggle nodes	Toggles hit	Toggles missed	Toggle %	Toggled graph	St
456	410	46	89.91%	🟨	
82	75	7	91.46%	🟩	

Transition Coverage



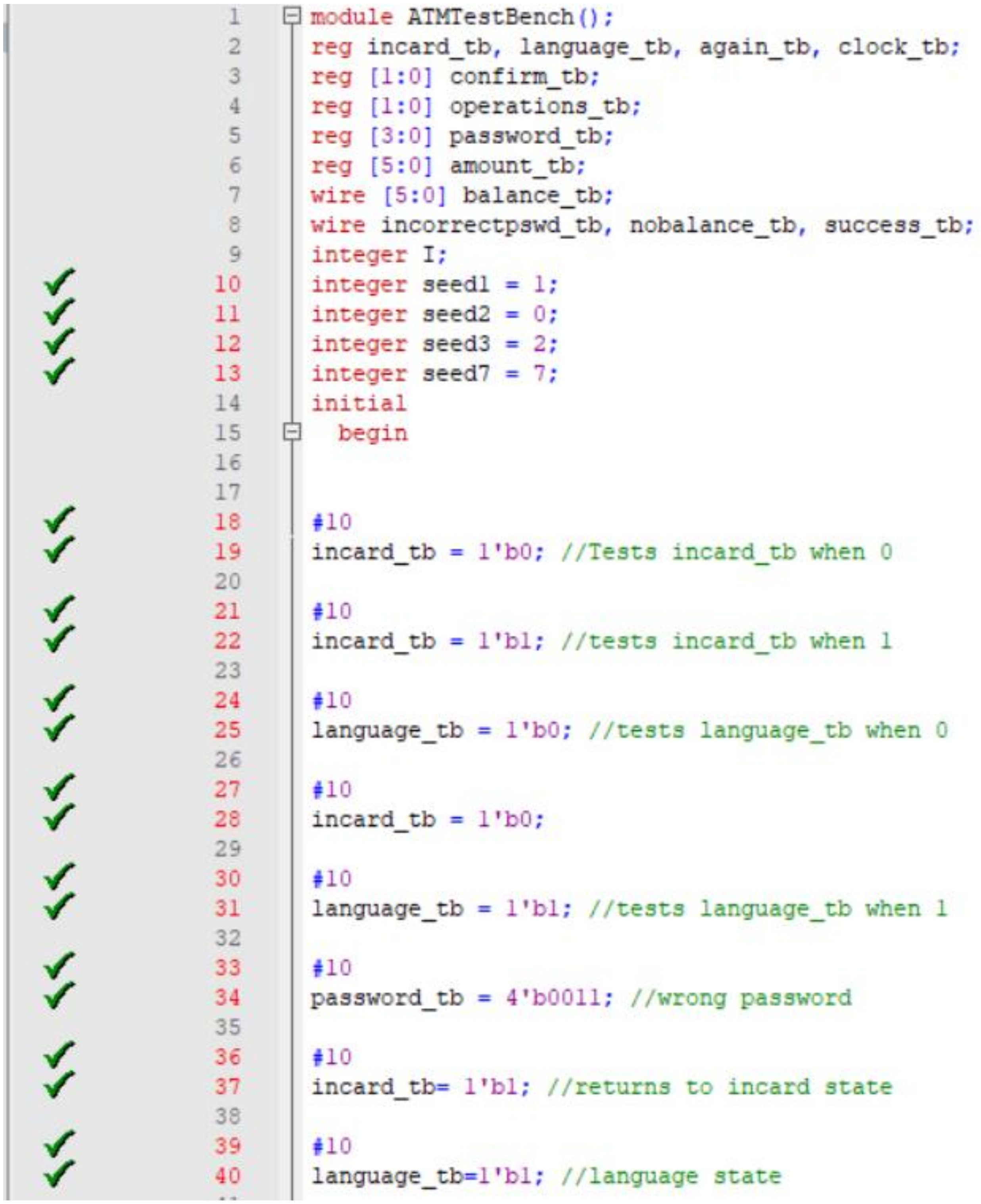
States	States hit	States missed	State %	State graph	Transitions	Transitions hit	Transitions missed	Transition %	Transition graph
9	9	0	100.00%	🟩	15	15	0	100.00%	🟩
9	9	0	100.00%	🟩	15	15	0	100.00%	🟩

Branch Coverage

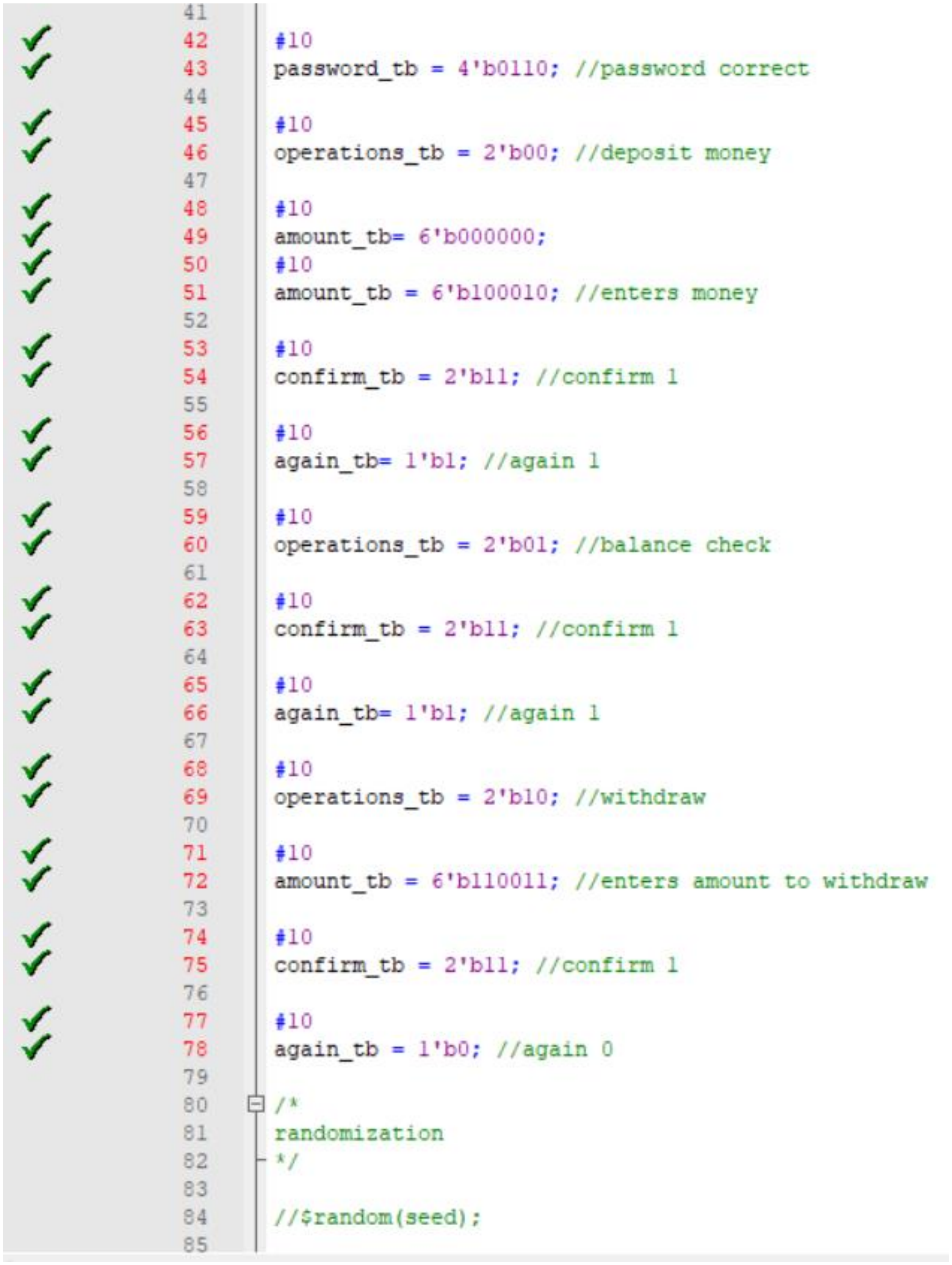


Name	Specified path	Full path	Type	Stmt Count	Stmt Hits	Stmt %	Stmt Graph	Branch Count	Branch Hits	Branch %	Branch Graph	Co
sim	vsim.wlf	D:/Junio…										
ATM.v	ATM.v	D:/Junio…	verilog	42	42	100.00	🟩	42	41	97.62	🟩	
ATMTestBench…	ATMTestBench.v	D:/Junio…	verilog	85	85	100.00	🟩					

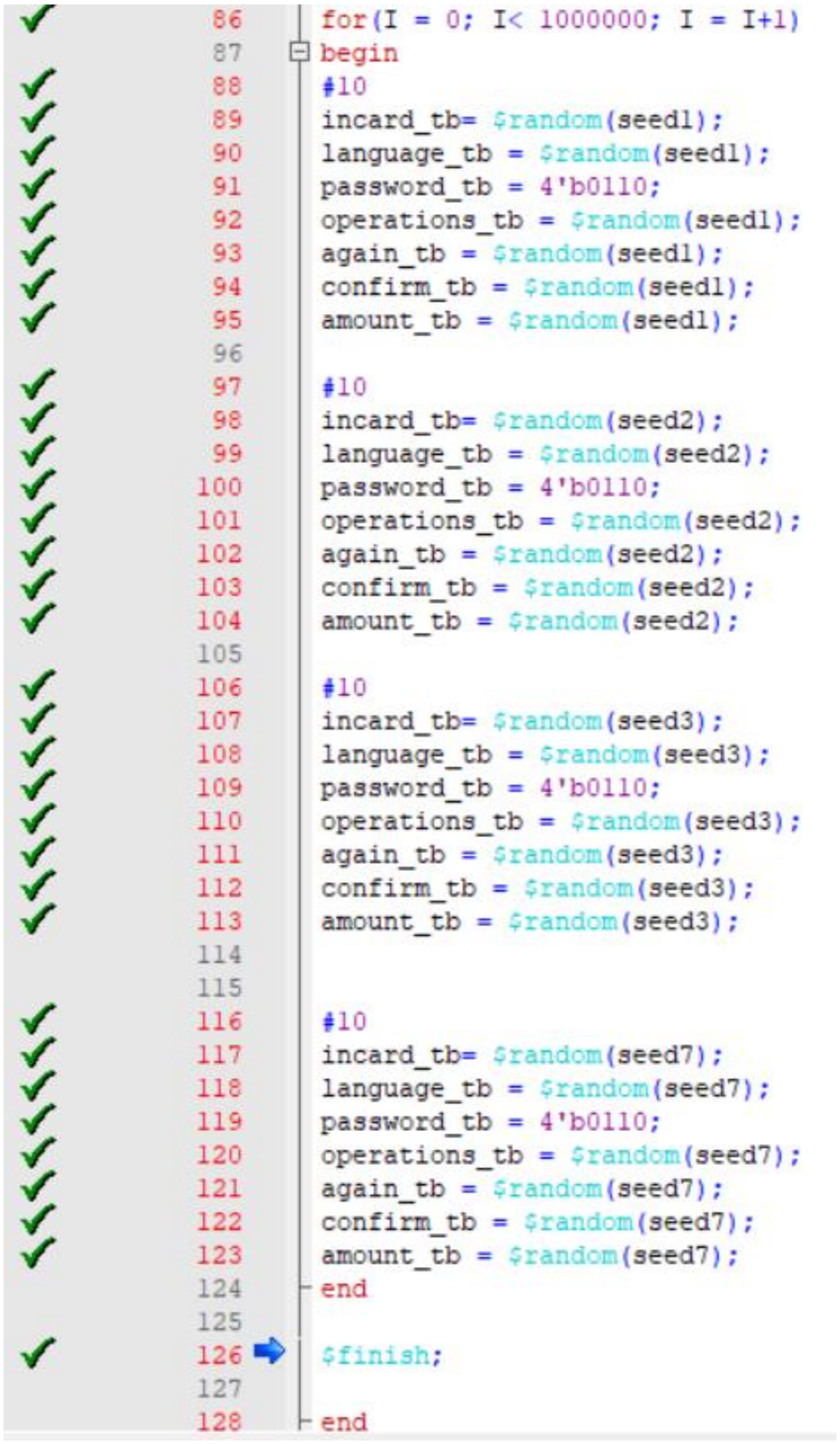
Test Bench Code



```
module ATMTestBench();
reg incard_tb, language_tb, again_tb, clock_tb;
reg [1:0] confirm_tb;
reg [1:0] operations_tb;
reg [3:0] password_tb;
reg [5:0] amount_tb;
wire [5:0] balance_tb;
wire incorrectpswd_tb, nobalance_tb, success_tb;
integer I;
integer seed1 = 1;
integer seed2 = 0;
integer seed3 = 2;
integer seed7 = 7;
initial
  begin


#10
incard_tb = 1'b0; //Tests incard_tb when 0

#10
incard_tb = 1'b1; //tests incard_tb when 1

#10
language_tb = 1'b0; //tests language_tb when 0

#10
incard_tb = 1'b0;

#10
language_tb = 1'b1; //tests language_tb when 1

#10
password_tb = 4'b0011; //wrong password

#10
incard_tb= 1'b1; //returns to incard state

#10
language_tb=1'b1; //language state
```

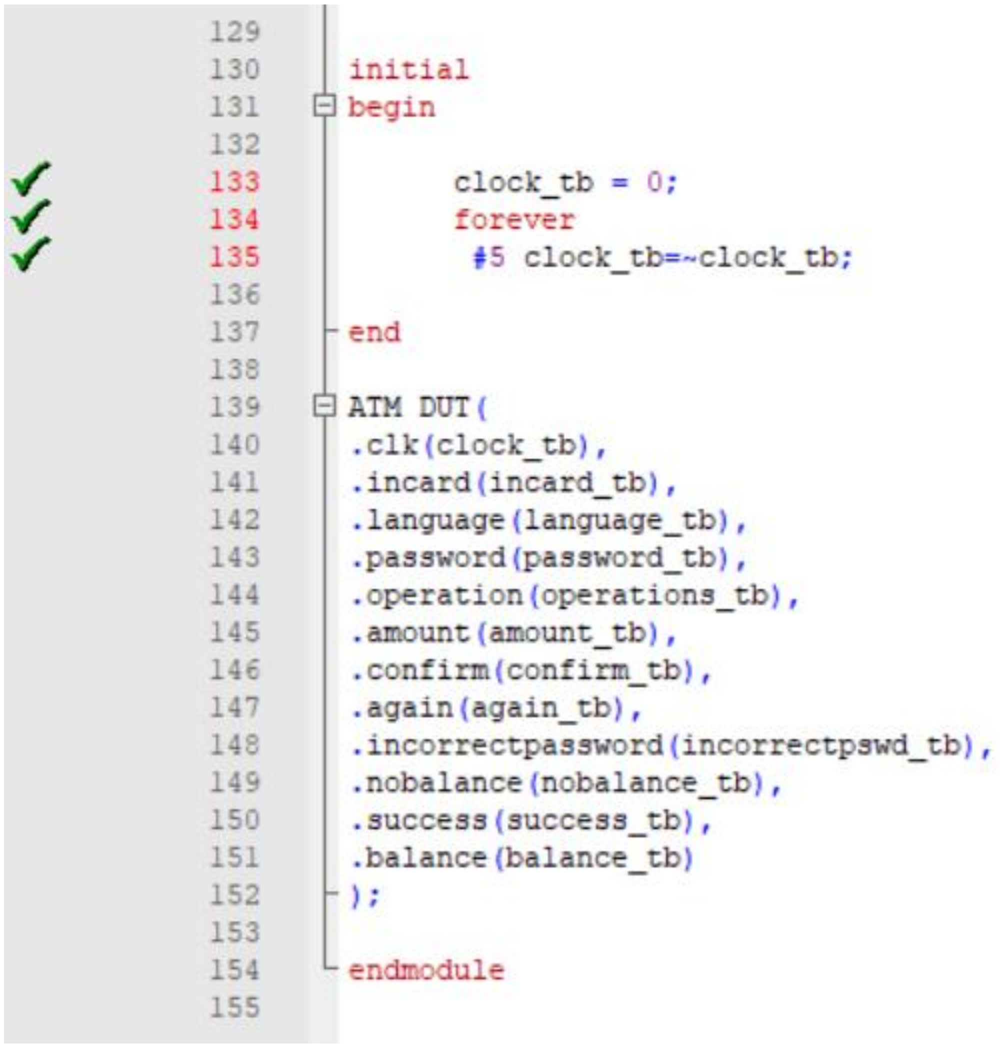


```

#10
password_tb = 4'b0110; //password correct

#10
operations_tb = 2'b00; //deposit money

#10
amount_tb= 6'b000000;
#10
amount_tb = 6'b100010; //enters money

#10
confirm_tb = 2'b11; //confirm 1

#10
again_tb= 1'b1; //again 1

#10
operations_tb = 2'b01; //balance check

#10
confirm_tb = 2'b11; //confirm 1

#10
again_tb= 1'b1; //again 1

#10
operations_tb = 2'b10; //withdraw

#10
amount_tb = 6'b110011; //enters amount to withdraw

#10
confirm_tb = 2'b11; //confirm 1

#10
again_tb = 1'b0; //again 0

/*
randomization
*/

//$random(seed);

```



```
for(I = 0; I< 1000000; I = I+1)
begin
#10
incard_tb= $random(seed1);
language_tb = $random(seed1);
password_tb = 4'b0110;
operations_tb = $random(seed1);
again_tb = $random(seed1);
confirm_tb = $random(seed1);
amount_tb = $random(seed1);

#10
incard_tb= $random(seed2);
language_tb = $random(seed2);
password_tb = 4'b0110;
operations_tb = $random(seed2);
again_tb = $random(seed2);
confirm_tb = $random(seed2);
amount_tb = $random(seed2);

#10
incard_tb= $random(seed3);
language_tb = $random(seed3);
password_tb = 4'b0110;
operations_tb = $random(seed3);
again_tb = $random(seed3);
confirm_tb = $random(seed3);
amount_tb = $random(seed3);


#10
incard_tb= $random(seed7);
language_tb = $random(seed7);
password_tb = 4'b0110;
operations_tb = $random(seed7);
again_tb = $random(seed7);
confirm_tb = $random(seed7);
amount_tb = $random(seed7);
end

$finish;

end
```

Randomization



```verilog

initial
begin

      clock_tb = 0;
      forever
       #5 clock_tb=~clock_tb;

end

ATM DUT(
.clk(clock_tb),
.incard(incard_tb),
.language(language_tb),
.password(password_tb),
.operation(operations_tb),
.amount(amount_tb),
.confirm(confirm_tb),
.again(again_tb),
.incorrectpassword(incorrectpswd_tb),
.nobalance(nobalance_tb),
.success(success_tb),
.balance(balance_tb)
);

endmodule
```

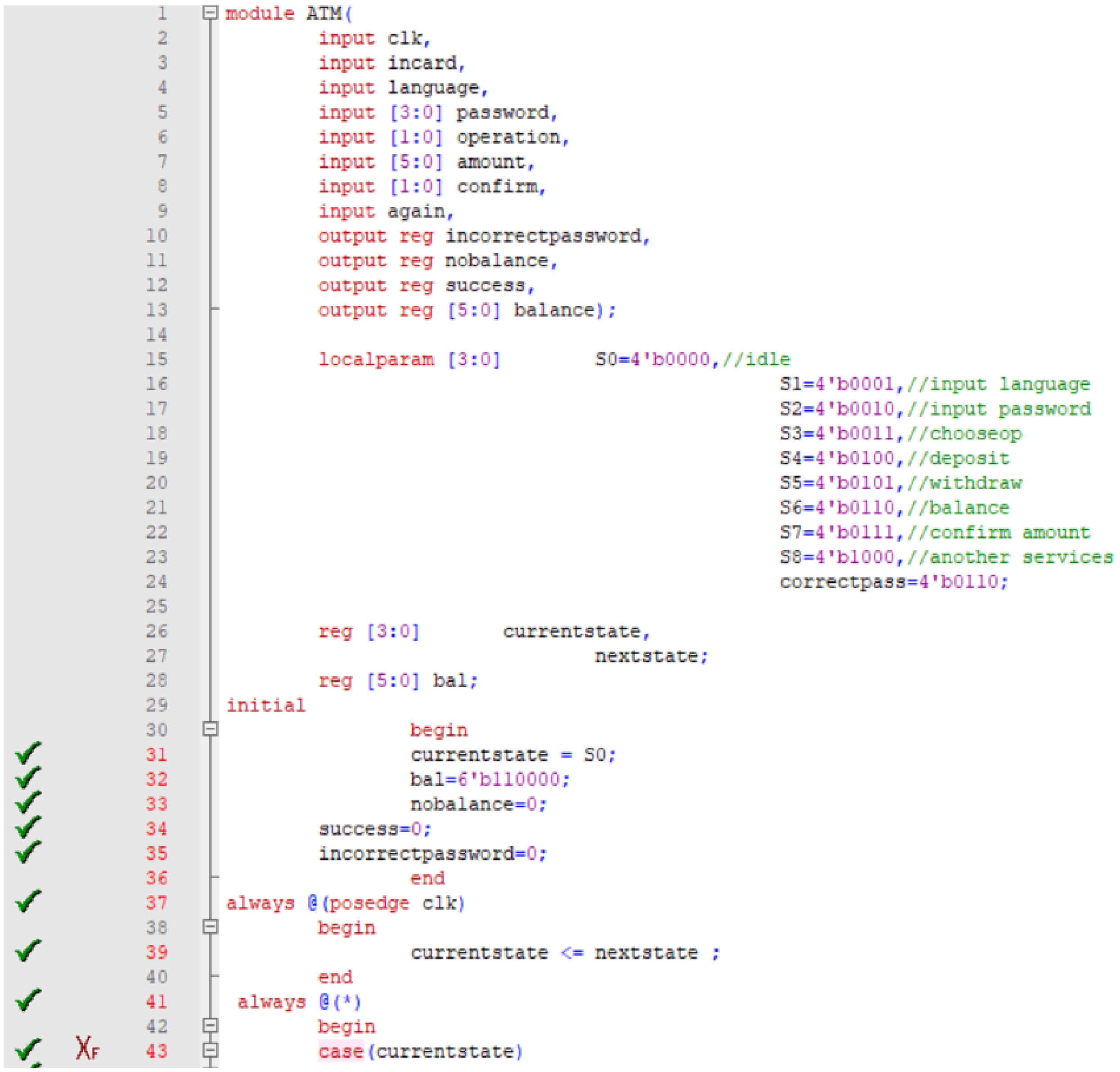
Verification plan

- Regs defined to port map with inputs of the design
- Wires defined to port map with outputs of design
- Integer I is defined as it is used in for loop (randomized input to test output)
- Different seeds are used to generate different random numbers in each iteration in the loop

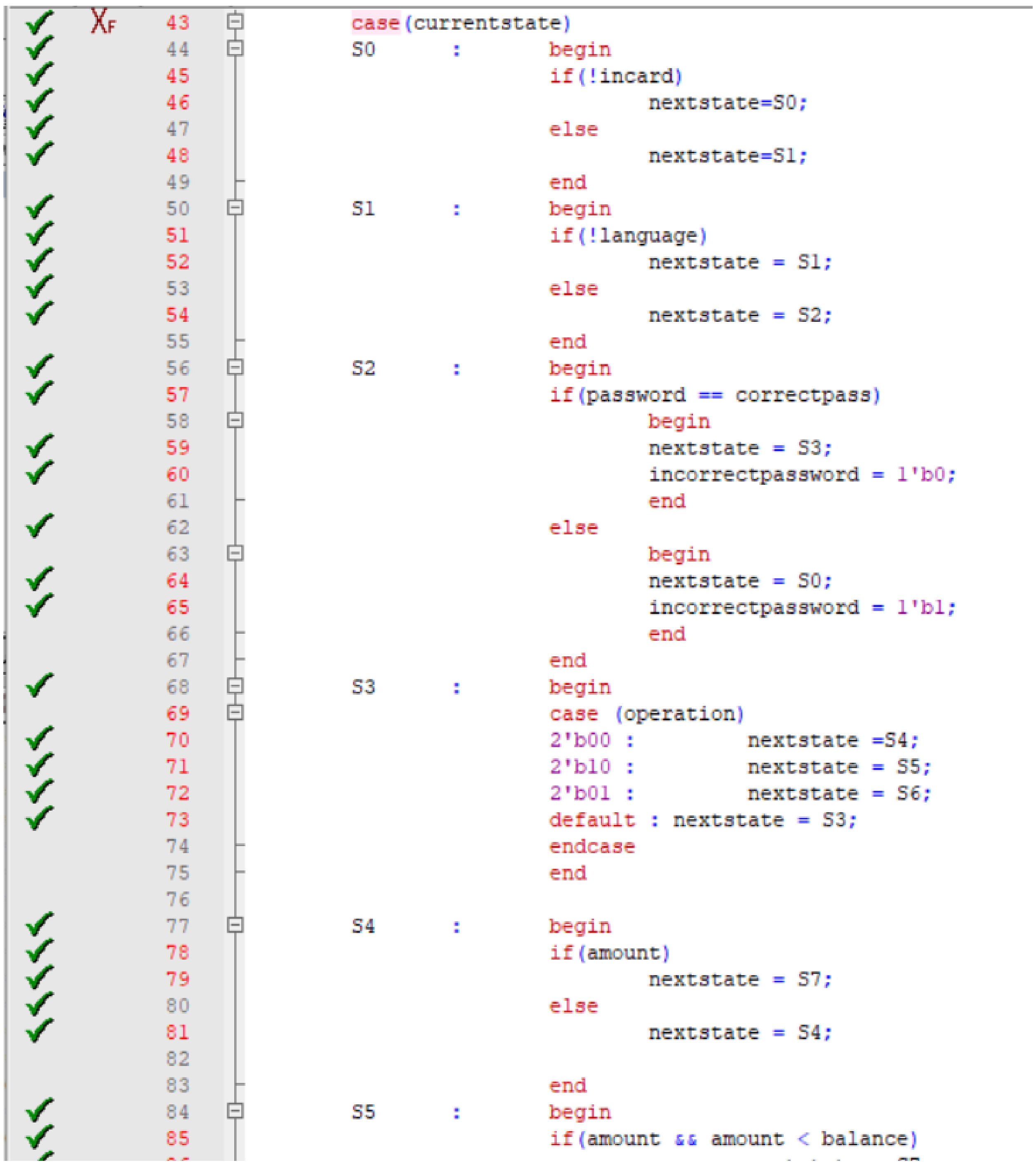
Directed test bench

- Incard initialized by 0 to verify that the current state is an idle state
- Incard initialized by 1 to verify that current state becomes s1
- Language initialized by 0 to verify that it stays in the same state
- Language initialized by 1 to verify that s1 has passed and the current state is now s2
- wrong password entered to check that if the password is incorrect the system moves back to S0
- Right password entered to verify that it moves to s3
- Different operations are entered first deposit operation is checked by initializing the operation to 00 and different amounts are input
- Then again initialized by 1 to check for check balance operation (01) then withdraw operation is checked(10) and amount is input
- Then again is set to 0 to verify that system returns to s0 Randomized
- Loop is set to iterate a fixed number of times to generate different inputs to be tested

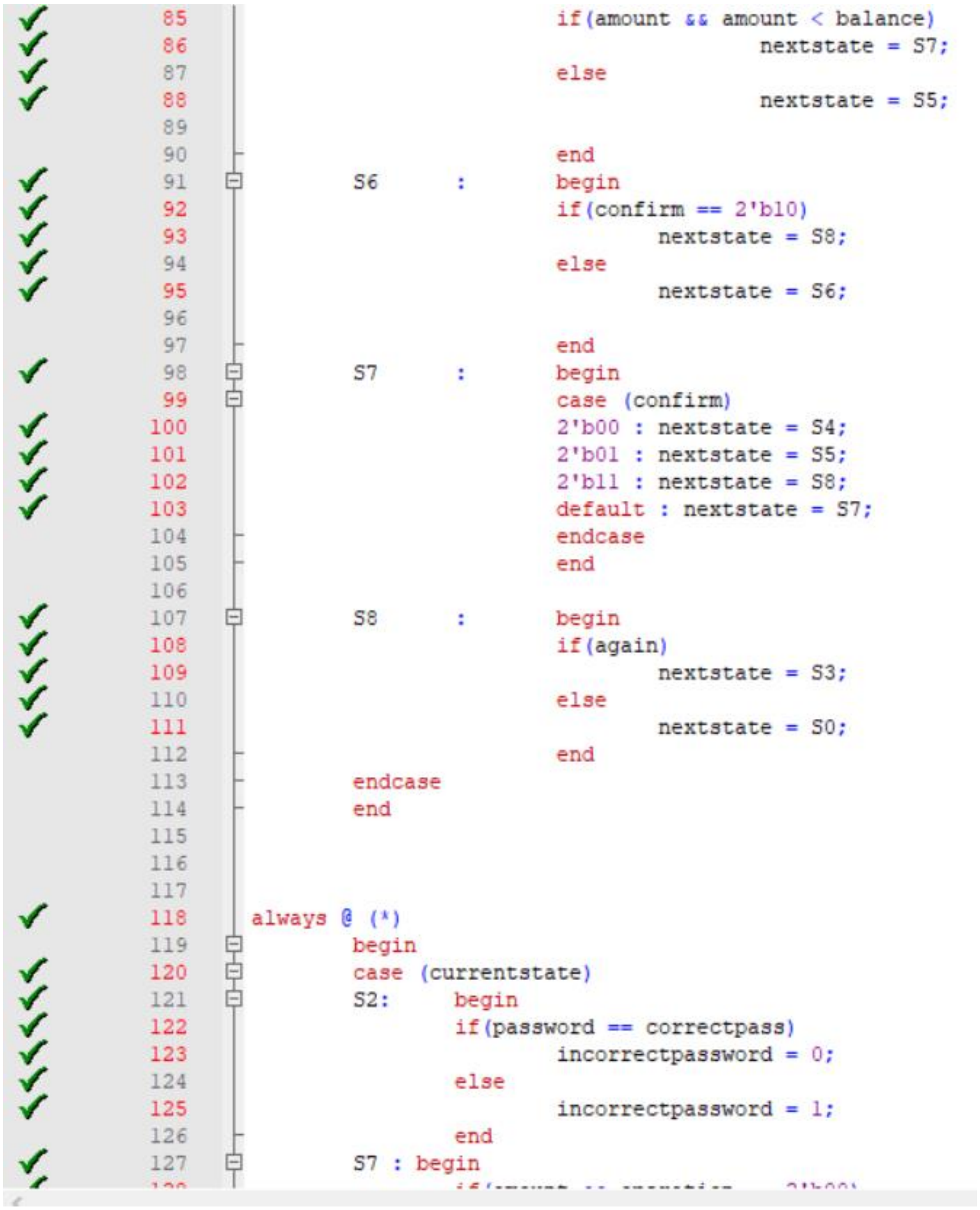
Design RTL Code



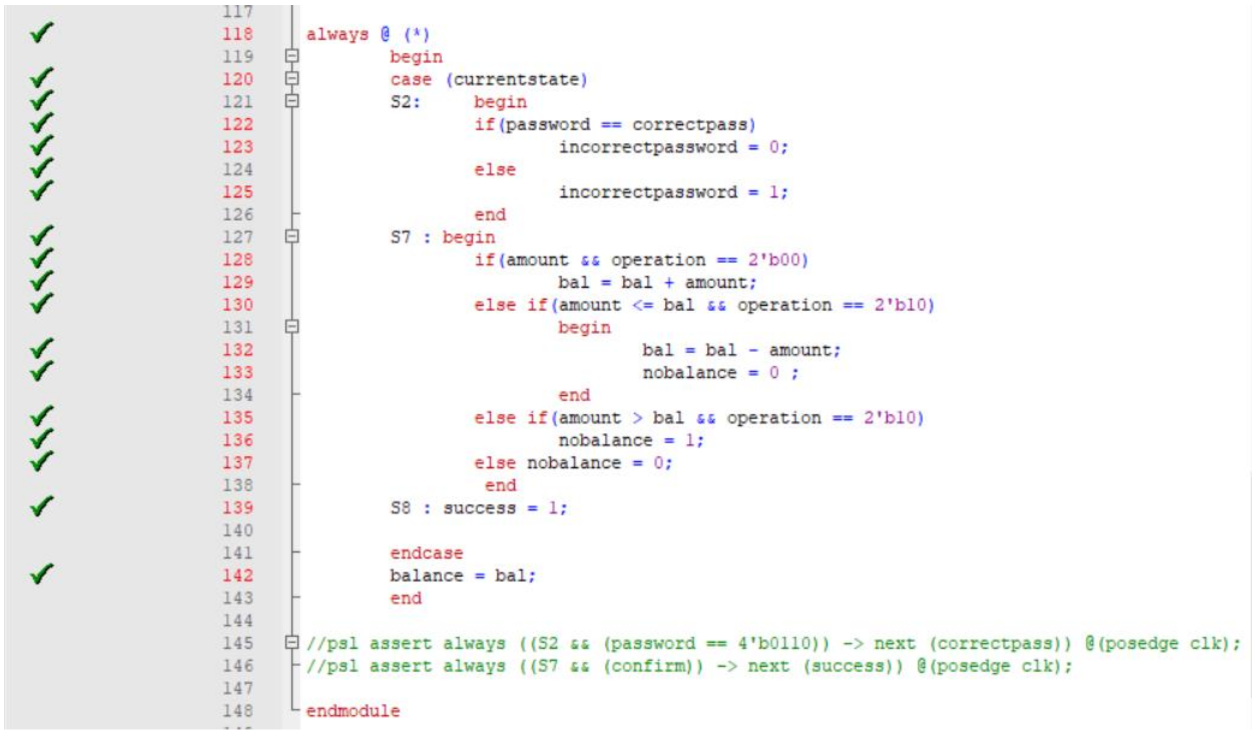
```verilog
module ATM(
        input clk,
        input incard,
        input language,
        input [3:0] password,
        input [1:0] operation,
        input [5:0] amount,
        input [1:0] confirm,
        input again,
        output reg incorrectpassword,
        output reg nobalance,
        output reg success,
        output reg [5:0] balance);

        localparam [3:0]        S0=4'b0000,//idle
                                        S1=4'b0001,//input language
                                        S2=4'b0010,//input password
                                        S3=4'b0011,//chooseop
                                        S4=4'b0100,//deposit
                                        S5=4'b0101,//withdraw
                                        S6=4'b0110,//balance
                                        S7=4'b0111,//confirm amount
                                        S8=4'b1000,//another services
                                        correctpass=4'b0110;

        reg [3:0]       currentstate,
                                nextstate;
        reg [5:0] bal;
initial
                begin
                currentstate = S0;
                bal=6'b110000;
                nobalance=0;
        success=0;
        incorrectpassword=0;
                end
always @(posedge clk)
        begin
                currentstate <= nextstate ;
        end
 always @(*)
        begin
        case(currentstate)
```



```
case(currentstate)
S0      :       begin
                if(!incard)
                        nextstate=S0;
                else
                        nextstate=S1;
                end
S1      :       begin
                if(!language)
                        nextstate = S1;
                else
                        nextstate = S2;
                end
S2      :       begin
                if(password == correctpass)
                        begin
                        nextstate = S3;
                        incorrectpassword = 1'b0;
                        end
                else
                        begin
                        nextstate = S0;
                        incorrectpassword = 1'b1;
                        end
                end
S3      :       begin
                case (operation)
                2'b00 :         nextstate =S4;
                2'b10 :         nextstate = S5;
                2'b01 :         nextstate = S6;
                default : nextstate = S3;
                endcase
                end

S4      :       begin
                if(amount)
                        nextstate = S7;
                else
                        nextstate = S4;

                end
S5      :       begin
                if(amount && amount < balance)
```



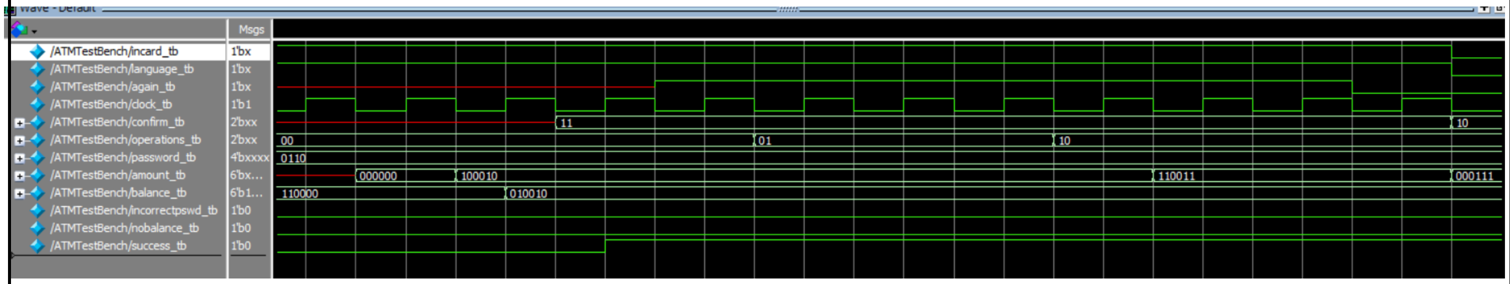
```
                                if(amount && amount < balance)
                                        nextstate = S7;
                                else
                                        nextstate = S5;

                                end
                S6      :       begin
                                if(confirm == 2'b10)
                                        nextstate = S8;
                                else
                                        nextstate = S6;

                                end
                S7      :       begin
                                case (confirm)
                                2'b00 : nextstate = S4;
                                2'b01 : nextstate = S5;
                                2'b11 : nextstate = S8;
                                default : nextstate = S7;
                                endcase
                                end

                S8      :       begin
                                if(again)
                                        nextstate = S3;
                                else
                                        nextstate = S0;
                                end
                endcase
                end



always @ (*)
                begin
                case (currentstate)
                S2:     begin
                        if(password == correctpass)
                                incorrectpassword = 0;
                        else
                                incorrectpassword = 1;
                        end
                S7 : begin
```

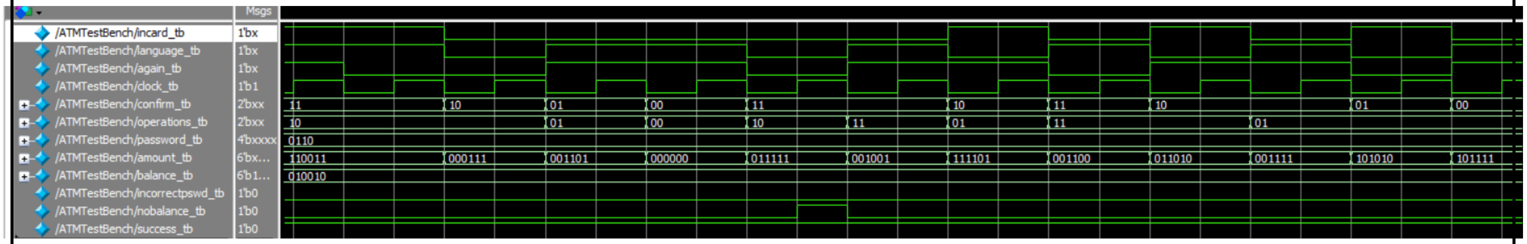


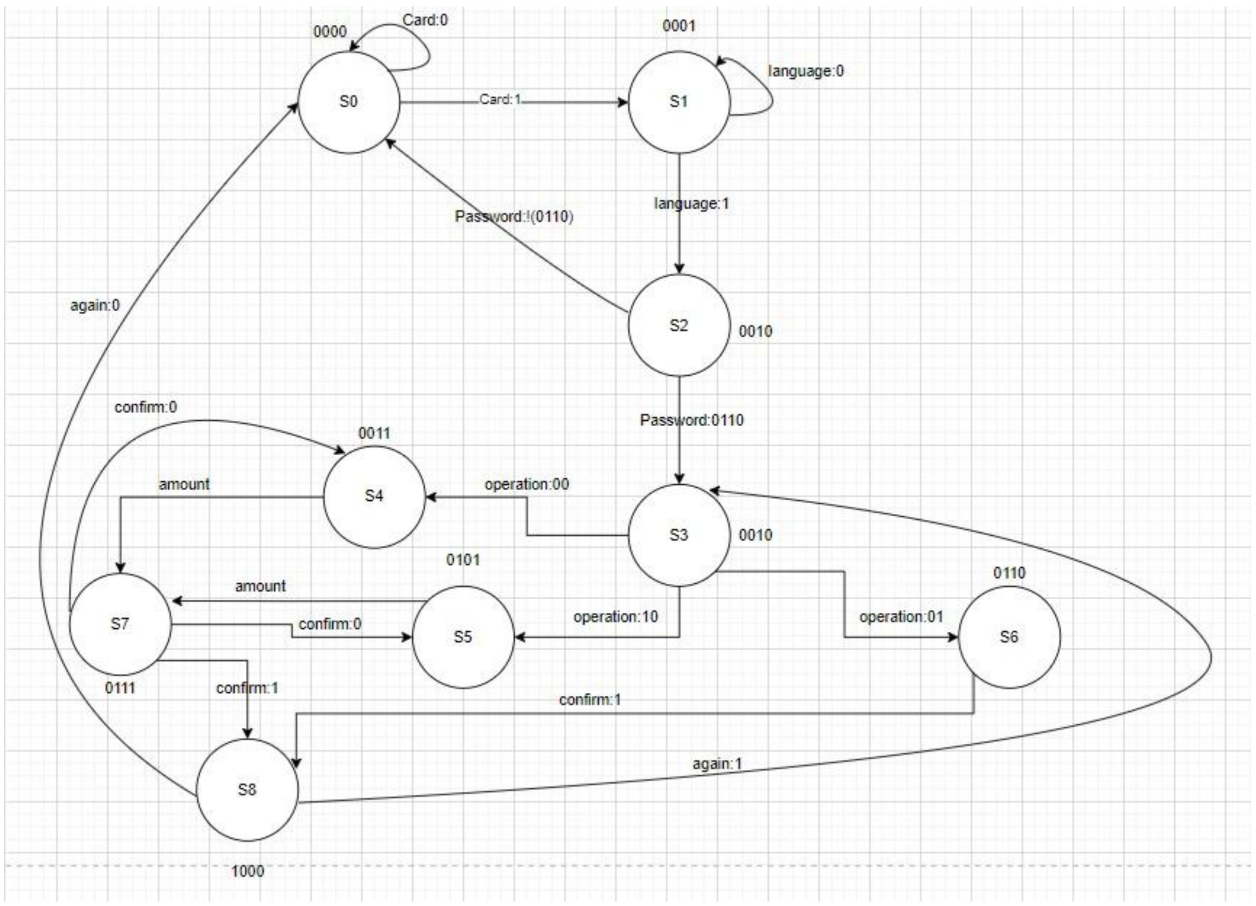
```
always @ (*)
        begin
        case (currentstate)
        S2:     begin
                if(password == correctpass)
                        incorrectpassword = 0;
                else
                        incorrectpassword = 1;
                end
        S7 : begin
                if(amount && operation == 2'b00)
                        bal = bal + amount;
                else if(amount <= bal && operation == 2'b10)
                        begin
                                bal = bal - amount;
                                nobalance = 0 ;
                        end
                else if(amount > bal && operation == 2'b10)
                        nobalance = 1;
                else nobalance = 0;
                 end
        S8 : success = 1;

        endcase
        balance = bal;
        end

//psl assert always ((S2 && (password == 4'b0110)) -> next (correctpass)) @(posedge clk);
//psl assert always ((S7 && (confirm)) -> next (success)) @(posedge clk);

endmodule
```

Assertion

Waveforms samples







State diagram

IDLE (0000) S0

Language->language(0001) S1

Password-> password[3:0](0010) S2

Choose operation→operation[1:0](0011) S3

deposit->00(0100) S4

withdraw->10(0101)S5

balance->01(0110)S6

check amount (0111)S7

another services(1000)S8

State's name & details

State	Input	outcome
S0 (IDLE)	Card = 0	S0 (IDLE)
	Card = 1	S1 (Language)
S1 (Language)	Language = 0	S1 (language)
	Language = 1	S2 (Operations)
S2 (password)	Password != 0110	S0 (IDLE)
	Password == 0110	S3 (Operations)
S3 (operations)	Operation = 00	S4 (deposit)
	Operation = 01	S6 (balance)
	Operation = 10	S5 (withdraw)
S4 (deposit)	Amount = $random	S7 (check amount)
S5 (withdraw)	Amount = $random	S7 (check amount)
S6 (balance)	Confirm = 1	S8 (other services)
S7 (check amount)	Confirm = 0 (if S5)	S5 (withdraw)
	Confirm = 0 (if S4)	S4 (deposit)
	Confirm = 1	S8 (other services)
S8 (other services)	Again = 0	S0 (IDLE)
	Again = 1	S3 (operations)