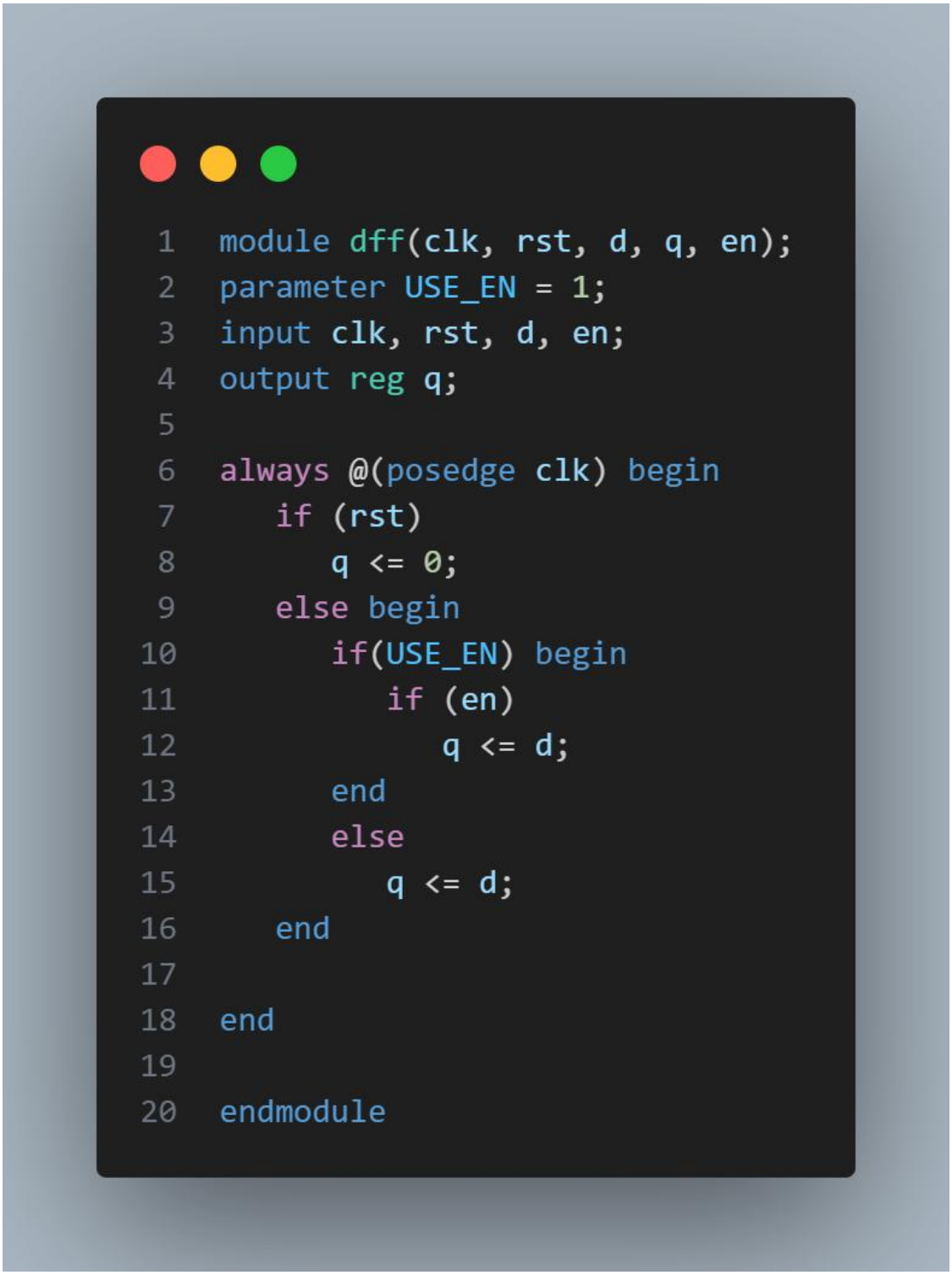
1. Design



```
1  module dff(clk, rst, d, q, en);
2  parameter USE_EN = 1;
3  input clk, rst, d, en;
4  output reg q;
5
6  always @(posedge clk) begin
7     if (rst)
8        q <= 0;
9     else begin
10       if(USE_EN) begin
11          if (en)
12             q <= d;
13       end
14       else
15          q <= d;
16    end
17
18 end
19
20 endmodule
```

Added **begin/end** for every nested if else

2. Testbench
   - Testbench 1

```
module DFF_test1();
    reg clk ;
    reg rst ;
    reg d   ;
    reg en  ;
    wire q  ;

    integer error_count , correct_count ;

    dff #(.USE_EN(1)) DUT (clk, rst, d, q, en) ;

    initial begin
        clk = 0;
        forever begin
            #2 clk = ~clk;
        end
    end

    initial begin
        rst = 0 ;
        en  = 0 ;
        d   = 1 ;
        correct_count = 0 ;
        error_count   = 0 ;
        // TEST_1
        assert_reset () ;
        // TEST_2
        repeat (10) begin
            en = 0 ;
            d  = $random ;
            check_result () ;
        end
        // TEST_3
        repeat (10) begin
            en = 1 ;
            d  = $random ;
            check_result () ;
        end
        // TEST_4
        repeat (10) begin
            en = $random ;
            d  = $random ;
            check_result () ;
        end
        // TEST_5
        assert_reset () ;

        $display ("correct counter = %d , error counter = %d " ,
correct_count , error_count ) ;
        $stop ;

    end

    task check_result ();
        @ (negedge clk) ;
        if (en) begin
            if (q == d) begin
                $display ("test passes") ;
                correct_count = correct_count + 1 ;
            end
            else begin
                $display ("test fail ") ;
                error_count = error_count + 1 ;
            end
        end
        else begin
            if (q == q) begin
                $display ("test passes") ;
                correct_count = correct_count + 1 ;
            end
            else begin
                $display ("test fail ") ;
                error_count = error_count + 1 ;
            end
        end
    endtask

    task assert_reset ();
        rst = 1 ;
        @ (negedge clk) ;
        if (q == 0 ) begin
            $display ("test passes") ;
            correct_count = correct_count + 1 ;
        end
        else begin
            $display ("test fail ") ;
            error_count = error_count + 1 ;
        end
        rst = 0 ;
    endtask
endmodule
```

- Testbench 2

```
module DFF_test2();
    reg clk ;
    reg rst ;
    reg d   ;
    reg en  ;
    wire q  ;

    integer error_count , correct_count ;

    dff #(.USE_EN(0)) DUT (clk, rst, d, q, en) ;

    initial begin
        clk = 0;
        forever begin
            #2 clk = ~clk;
        end
    end

    initial begin
        rst = 0 ;
        en  = 0 ;
        d   = 1 ;
        correct_count = 0 ;
        error_count   = 0 ;
        // TEST_1
        assert_reset () ;
        // TEST_2
        repeat (10) begin
            en = 0 ;
            d  = $random ;
            check_result () ;
        end
        // TEST_3
        repeat (10) begin
            en = 1 ;
            d  = $random ;
            check_result () ;
        end
        // TEST_4
        repeat (10) begin
            en = $random ;
            d  = $random ;
            check_result () ;
        end
        // TEST_5
        assert_reset () ;

        $display ("correct counter = %d , error counter = %d " ,
correct_count , error_count ) ;
        $stop ;

    end

    task check_result ();
        @ (negedge clk) ;
            if (q == d) begin
                $display ("test passes") ;
                correct_count = correct_count + 1 ;
            end
            else begin
                $display ("test fail ") ;
                error_count = error_count + 1 ;
            end
    endtask

    task assert_reset ();
        rst = 1 ;
        @ (negedge clk) ;
        if (q == 0 ) begin
            $display ("test passes") ;
            correct_count = correct_count + 1 ;
        end
        else begin
            $display ("test fail ") ;
            error_count = error_count + 1 ;
        end
        rst = 0 ;
    endtask
endmodule
```

3. Do file

```
# Compile the design and both testbenches
vlib work
vlog dff.v  +cover -covercells
vlog dff_t1_tb.svh  +cover -covercells
vlog dff_t2_tb.svh  +cover -covercells

# --- Run simulation for first testbench ---
vsim -voptargs=+acc work.DFF_test1 -cover
add wave *
coverage save DFF_test1.ucdb -onexit
run -all
quit -sim

# --- Run simulation for second testbench ---
vsim -voptargs=+acc work.DFF_test2 -cover
add wave *
coverage save DFF_test2.ucdb -onexit
run -all
quit -sim

# --- Merge the two coverage databases ---
vcover merge dff_merged.ucdb DFF_test1.ucdb DFF_test2.ucdb -du dff
vcover report dff_merged.ucdb -details -annotate -all
```

4. Coverage file

```
# ================================================================================
# === Instance: /\work.dff
# === Design Unit: work.dff
# ================================================================================
# Branch Coverage:
#     Enabled Coverage              Bins      Hits    Misses  Coverage
#     ----------------              ----      ----    ------  --------
#     Branches                         4         4         0   100.00%
#
# ================================Branch Details================================
#
# Branch Coverage for instance /\work.dff
#
#     Line         Item                      Count     Source
#     ----         ----                      -----     ------
#   File dff.v
# ------------------------------------IF Branch------------------------------------
#     7                                        49     Count coming in to IF
#     7              1                          4        if (rst)
#
#     11             1                         13             if (en)
#
#                                               9     All False Count
#     9              1                         23        else begin
#
# Branch totals: 4 hits of 4 branches = 100.00%
#
#
# Statement Coverage:
#     Enabled Coverage              Bins      Hits    Misses  Coverage
#     ----------------              ----      ----    ------  --------
#     Statements                       4         4         0   100.00%
#
```

```
# ================================Statement Details================================
#
# Statement Coverage for instance /\work.dff  --
#
#     Line         Item                      Count     Source
#     ----         ----                      -----     ------
#   File dff.v
#     1                                                module dff(clk, rst, d, q, en);
#
#     2                                                parameter USE_EN = 1;
#
#     3                                                input clk, rst, d, en;
#
#     4                                                output reg q;
#
#     5
#
#     6              1                         49     always @(posedge clk) begin
#
#     7                                                   if (rst)
#
#     8              1                          4          q <= 0;
#
#     9                                                   else begin
#
#     10                                                     if(USE_EN) begin
#
#     11                                                        if (en)
#
#     12             1                         13               q <= d;
#
#     13                                                    end
#
#     14                                                    else
#
#     15             1                         23             q <= d;
#
```

```
#
# Toggle Coverage:
#     Enabled Coverage              Bins      Hits    Misses  Coverage
#     ----------------              ----      ----    ------  --------
#     Toggles                         10        10         0   100.00%
#
# ================================Toggle Details================================
#
# Toggle Coverage for instance /\work.dff  --
#
#                                              Node      1H->0L      0L->1H  "Coverage"
#                                              ----------------------------------------
#                                               clk           2           2      100.00
#                                                 d           2           2      100.00
#                                                en           2           2      100.00
#                                                 q           2           2      100.00
#                                               rst           2           2      100.00
#
# Total Node Count     =          5
# Toggled Node Count   =          5
# Untoggled Node Count =          0
#
# Toggle Coverage      =     100.00% (10 of 10 bins)
#
#
# Total Coverage By Instance (filtered view): 100.00%
```

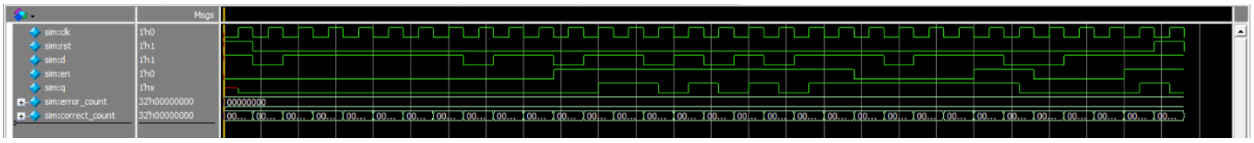
5. Verification plan
    - Testbench 1

	A	B	C	D	E
1	**Label**	**Design Requirement Description**	**Stimulus Generation**	**Functional Coverage**	**Functionality Check**
2	TEST_1	When the reset is asserted, the output value should be low	Directed at the start of the simulation	-	A checker in the testbench to make sure the output is correct
3	TEST_2	Assign en with 0 and assign d with 10 random values , the output should doesn't change	Randomized	-	A checker in the testbench to make sure the output is correct
4	TEST_3	Assign en with 0 and assign d with 10 random values , the output should be the value of input (d)	Randomized	-	A checker in the testbench to make sure the output is correct
5	TEST_4	Assign en with 10 random values and assign d with 10 random values , the output should the value of input (d) or doesn't change according to value of (en)	Randomized	-	A checker in the testbench to make sure the output is correct
6	TEST_5	When the reset is asserted, the output value should be low	Directed at the end of the simulation	-	A checker in the testbench to make sure the output is correct

    - Testbench 2

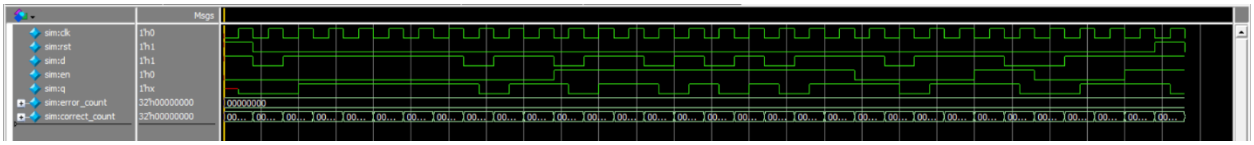
	A	B	C	D	E
1	**Label**	**Design Requirement Description**	**Stimulus Generation**	**Functional Coverage**	**Functionality Check**
2	TEST_1	When the reset is asserted, the output value should be low	Directed at the start of the simulation	-	A checker in the testbench to make sure the output is correct
3	TEST_2	Assign en with 0 and assign d with 10 random values , the output should be the value of input (d)	Randomized	-	A checker in the testbench to make sure the output is correct
4	TEST_3	Assign en with 0 and assign d with 10 random values , the output should be the value of input (d)	Randomized	-	A checker in the testbench to make sure the output is correct
5	TEST_4	Assign en with 10 random values and assign d with 10 random values ,the output should be the value of input (d)	Randomized	-	A checker in the testbench to make sure the output is correct
6	TEST_5	When the reset is asserted, the output value should be low	Directed at the end of the simulation	-	A checker in the testbench to make sure the output is correct

6. Questasim snippets
   - Testbench 1



```
# test passes
# test passes
# test passes
# test passes
# test passes
# test passes
# test passes
# test passes
# correct counter =          32 , error counter =           0
# ** Note: $stop    : dff_t1_tb.svh(48)
#    Time: 128 ns  Iteration: 1  Instance: /DFF_test1
# Break in Module DFF_test1 at dff_t1_tb.svh line 48
```

   - Testbench 2



```
# test passes
# test passes
# test passes
# test passes
# test passes
# test passes
# test passes
# test passes
# correct counter =          32 , error counter =           0
# ** Note: $stop    : dff_t2_tb.svh(49)
#    Time: 128 ns  Iteration: 1  Instance: /DFF_test2
# Break in Module DFF_test2 at dff_t2_tb.svh line 49
```