

Assignment 1 extra

Digital Design Verification

Contents

1	DFF "D Flip Flop"	2
1.1	1 Design	2
1.2	2. Verification Plan	2
1.3	3 Testbench	2
1.4	4 Do File	5
1.5	5 Coverage Report	6

1 DFF ”D Flip Flop”

1.1 1 Design

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

1.2 2. Verification Plan

Label	Design Requirement Description	Stimulus Generation	Functional Coverage	Functionality Check
DFF_USE_EN_1	When USE_EN = 1, q should follow d only when en is high	Exhaustive test patterns (00, 01, 10, 11)	-	A checker in the testbench to validate q updates based on en
DFF_USE_EN_0	When USE_EN = 0, q should always follow d, regardless of en	Exhaustive test patterns (00, 01, 10, 11, x)	-	A checker in the testbench to validate q always takes d, ignoring en
DFF_RESET	When reset is asserted, q should be set to 0	Directed test — apply reset	-	A checker in the testbench to validate q is 0 after reset is asserted

Table 1: DFF Testbench Design Requirements

1.3 3 Testbench

```
1 // Testbench for USE_EN = 1
2 module dff_tb1;
3
4     //-----
5     // Declare signals
6     //-----
7     reg clk ,rst ,d ,en;
8     wire q;
9     parameter USE_EN = 1;
10
11     //-----
12     // Instantiate the DUT (Device Under Test)
13     //-----
14     dff #(USE_EN) dut (.*);
15
16     //-----
17     // Clock generation
18     //-----
19     parameter CLOCK_PERIOD = 10;
20     always begin
21         #(CLOCK_PERIOD/2) clk = ~clk;
22     end
23
24     //-----
25     // Reset task
26     //-----
27     task reset_dut;
28     begin
29         // Assert reset
30         rst = 1; // synchronous reset
31         @(negedge clk);
32
33         // Check output reset to zero
34         if (q != 1'b0) begin
35             $display("Reset test failed: rst=%0b, d=%0b, en=%0b, q=%0b", rst, d, en, q);
36             $stop;
37         end else begin
38             $display("Reset test passed: rst=%0b, d=%0b, en=%0b, q=%0b", rst, d, en, q);
39         end
40
41         // Deassert reset
42         #(CLOCK_PERIOD/2);
43         rst = 0;
44     end
45 endtask
46
47     //-----
48     // Functionality check task
49     //-----
50     task check_functionality;
51         input reg d_in, en_in;
52         begin
53             d = d_in;
54             en = en_in;
55             @(negedge clk);
56             if (en) begin
```

```

57         if (q != d) begin
58             $display("[USE_EN=1]␣Test␣failed:␣rst=%0b,␣d=%0b,␣en=%0b,␣q=%0b", rst, d, en, q);
59             $stop;
60         end else
61             $display("[USE_EN=1]␣All␣tests␣passed!:␣rst=%0b,␣d=%0b,␣en=%0b,␣q=%0b", rst, d, en, q);
62         #(CLOCK_PERIOD/2);
63     end else begin
64         if (q != q) begin
65             $display("[USE_EN=1]␣Test␣failed:␣rst=%0b,␣d=%0b,␣en=%0b,␣q=%0b", rst, d, en, q);
66             $stop;
67         end else
68             $display("[USE_EN=1]␣All␣tests␣passed!:␣rst=%0b,␣d=%0b,␣en=%0b,␣q=%0b", rst, d, en, q);
69         #(CLOCK_PERIOD/2);
70     end
71 end
72 endtask
73
74
75 initial begin
76     //-----
77     // Initialize signals
78     //-----
79     clk = 1'b1;
80     rst = 1'b0;
81     d = 1;
82     en = 1;
83
84     #(CLOCK_PERIOD);
85
86     //-----
87     // Check reset functionality
88     //-----
89     reset_dut;
90
91     // Exhaustive test patterns
92     check_functionality(1'b0,1'b0);
93     check_functionality(1'b0,1'b1);
94     check_functionality(1'b1,1'b0);
95     check_functionality(1'b1,1'b1);
96
97     $finish;
98 end
99
100 endmodule
101
102 //*****
103 //*****
104 //*****
105
106 // Testbench for USE_EN = 0
107 module dff_tb2;
108
109     //-----
110     // Declare signals
111     //-----
112     reg clk ,rst ,d ,en;
113     wire q;
114     parameter USE_EN = 0;
115
116     //-----
117     // Instantiate the DUT (Device Under Test)
118     //-----
119     dff #(USE_EN) dut (.*);
120
121     //-----
122     // Clock generation
123     //-----
124     parameter CLOCK_PERIOD = 10;
125     always begin
126         #(CLOCK_PERIOD/2) clk = ~clk;
127     end
128
129     //-----
130     // Reset task
131     //-----
132     task reset_dut;
133         begin
134             // Assert reset
135             rst = 1;    // synchronous reset
136             @(negedge clk);
137
138             // Check output reset to zero
139             if (q != 1'b0) begin
140                 $display("Reset␣test␣failed:␣rst=%0b,␣d=%0b,␣en=%0b,␣q=%0b", rst, d, en, q);
141                 $stop;
142             end else begin
143                 $display("Reset␣test␣passed:␣rst=%0b,␣d=%0b,␣en=%0b,␣q=%0b", rst, d, en, q);
144             end
145
146             // Deassert reset
147             #(CLOCK_PERIOD/2);
148             rst = 0;
149         end
150     endtask
151
152     //-----
153     // Functionality check task
154     //-----
155     task check_functionality;
156         input reg d_in, en_in;

```

```

157 begin
158     d = d_in;
159     en = en_in;
160     @(negedge clk);
161     // check it's completely don't care about [en] signal
162     if (en || !en || en === 1'bx) begin
163         if (q != d) begin
164             $display("[USE_EN=0] Test failed: rst=%0b, d=%0b, en=%0b, q=%0b", rst, d, en, q);
165             $stop;
166         end else
167             $display("[USE_EN=0] All tests passed!: rst=%0b, d=%0b, en=%0b, q=%0b", rst, d, en, q);
168         end
169     #(CLOCK_PERIOD/2);
170 end
171 endtask
172
173
174 initial begin
175     //-----
176     // Initialize signals
177     //-----
178     clk = 1'b1;
179     rst = 1'b0;
180     d = 1;
181     en = 1;
182
183     #(CLOCK_PERIOD);
184
185     //-----
186     // Check reset functionality
187     //-----
188     reset_dut;
189
190     // Exhaustive test patterns
191     check_functionality(1'b0,1'b0);
192     check_functionality(1'b0,1'b1);
193     check_functionality(1'b1,1'b0);
194     check_functionality(1'b1,1'b1);
195     check_functionality(1'b1,1'bx);
196     check_functionality(1'b0,1'bx);
197
198     $finish;
199 end
200
201 endmodule

```

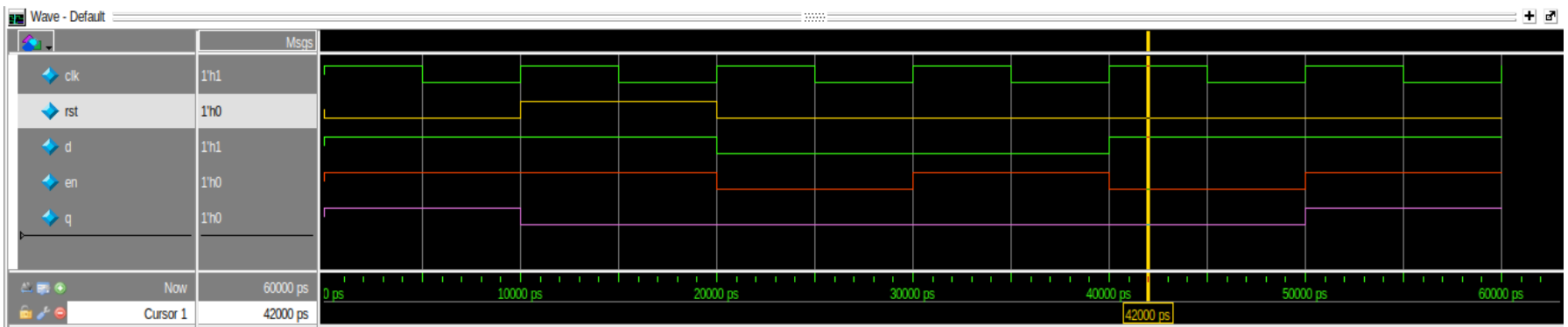


Figure 1: TB1 waveform [use\_en = 1]

```

# vsim -voptargs="+acc" dff_tb1 -coverage
# Start time: 11:31:49 on Mar 06,2025
# ** Note: (vsim-3813) Design is being optimized due to module recompilation...
# Loading sv_std.std
# Loading work.dff_tb1(fast)
# Loading work.dff(fast)
# Reset test passed: rst=1 , d=1 , en=1 , q=0
# [USE_EN=1] All tests passed!: rst=0, d=0, en=0, q=0
# [USE_EN=1] All tests passed!: rst=0, d=0, en=1, q=0
# [USE_EN=1] All tests passed!: rst=0, d=1, en=0, q=0
# [USE_EN=1] All tests passed!: rst=0, d=1, en=1, q=1
# ** Note: $finish      : dff_tb1.sv(97)
# Time: 60 ns Iteration: 0 Instance: /dff_tb1
# 1
# Break in Module dff_tb1 at dff_tb1.sv line 97

```

Figure 2: TB1 transcript [use\_en = 1]

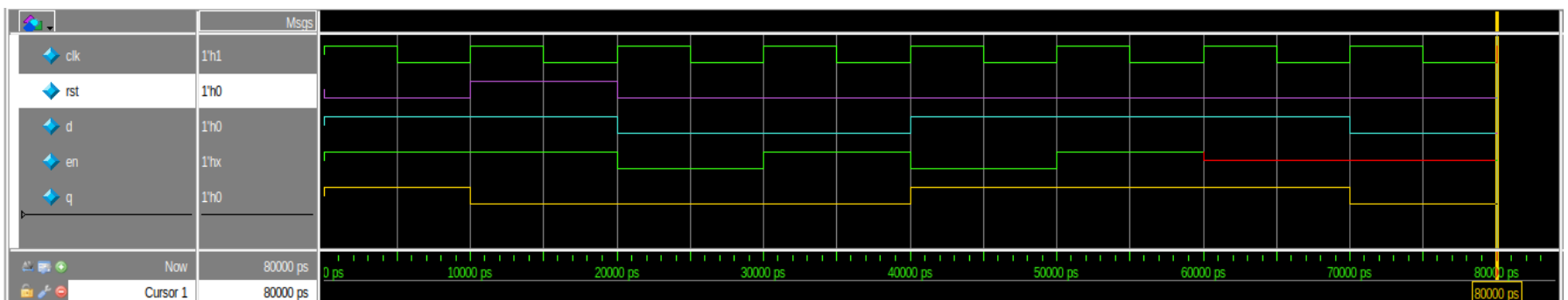


Figure 3: TB2 waveform [use\_en = 0]

```

# vsim -voptargs="+acc" dff_tb2 -coverage
# Start time: 11:49:49 on Mar 06,2025
# ** Note: (vsim-3813) Design is being optimized due to module recompilation...
# Loading sv_std.std
# Loading work.dff_tb2(fast)
# Loading work.dff(fast)
# Reset test passed: rst=1 , d=1 , en=1 , q=0
# [USE_EN=0] All tests passed!: rst=0, d=0, en=0, q=0
# [USE_EN=0] All tests passed!: rst=0, d=0, en=1, q=0
# [USE_EN=0] All tests passed!: rst=0, d=1, en=0, q=1
# [USE_EN=0] All tests passed!: rst=0, d=1, en=1, q=1
# [USE_EN=0] All tests passed!: rst=0, d=1, en=x, q=1
# [USE_EN=0] All tests passed!: rst=0, d=0, en=x, q=0
# ** Note: $finish      : dff_tb2.sv(92)
#   Time: 80 ns   Iteration: 0   Instance: /dff_tb2
# 1
# Break in Module dff tb2 at dff tb2.sv line 92

```

Figure 4: TB2 transcript [use\_en = 0]

#### 1.4 4 Do File

```

vlib work
vlog dff.v dff_tb1.sv dff_tb2.sv +cover -covercells

# Run testbench for USE_EN = 1
proc run_tb1 {} {
    vsim -voptargs="+acc" dff_tb1 -cover
    add wave *
    coverage save dff_tb1.ucdb -du dff -onexit
    run -all
}

# Run testbench for USE_EN = 0
proc run_tb2 {} {
    vsim -voptargs="+acc" dff_tb2 -cover
    add wave *
    coverage save dff_tb2.ucdb -du dff -onexit
    run -all
}

# Choose testbench to run
puts "Select testbench to run: 1 for USE_EN=1, 2 for USE_EN=0"
set choice [gets stdin]
if {$choice == 1} {
    run_tb1
} elseif {$choice == 2} {
    run_tb2
} else {
    puts "Invalid choice. Exiting."
}

# note : this do file to run tb according to input argument when run command do run.txt inside questasim

```

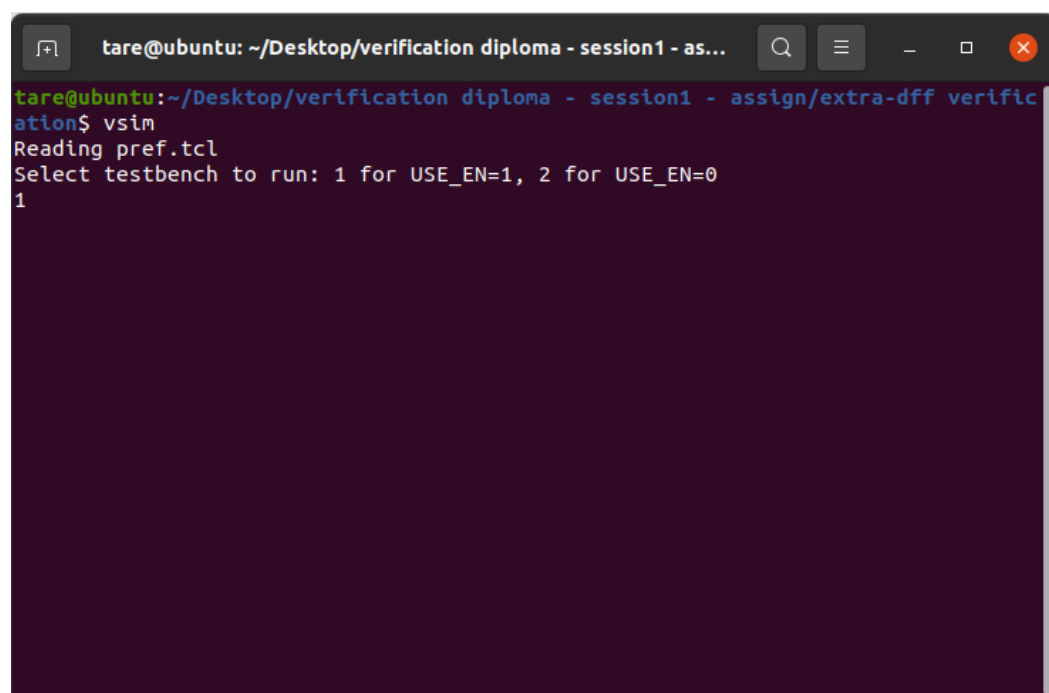


Figure 5: first run dor TB1 using ubuntu 20.04 termianl

```
tare@ubuntu: ~/Desktop/verification diploma - session1 - as...
tare@ubuntu:~/Desktop/verification diploma - session1 - assign/extra-dff verific
ation$ vsim
Reading pref.tcl
Select testbench to run: 1 for USE_EN=1, 2 for USE_EN=0
1
Select testbench to run: 1 for USE_EN=1, 2 for USE_EN=0
2
█
```

Figure 6: second run for TB2 using ubuntu 20.04 termianl

1.5 5 Coverage Report

Coverage Report by instance with details

===== 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		12	Count coming in to IF	
7	1	2	if (rst) begin	
11	1	3	if (en)	
		2	All False Count	
9	1	5	end 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	12	always @(posedge clk) begin	
7			if (rst) begin	
8	1	2	q <= 0;	
9			end else begin	
10			if(USE_EN) begin	
11			if (en)	
12	1	3	q <= d;	
13			end else	
14	1	5	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%