



Lab 4&5 Report

Name: Mert Can Bilgin

Student ID: 2453025

EXPERIMENTAL RESULTS

4.2 LAB 4

4.2.2 D FLIP-FLOP WITH ASYNCHRONOUS RESET AND SYNCHRONOUS LOAD

QUESTION 4.2.2.1

Draw a schematic to show how you would add combinational logic along with two new inputs (R and L) to a conventional D Flip-Flop to have the Reset and Load functions as shown in Figure 1. Note Load input take effect synchronously on the rising edge of the clock and Reset input is an active low input that takes effect asynchronously.

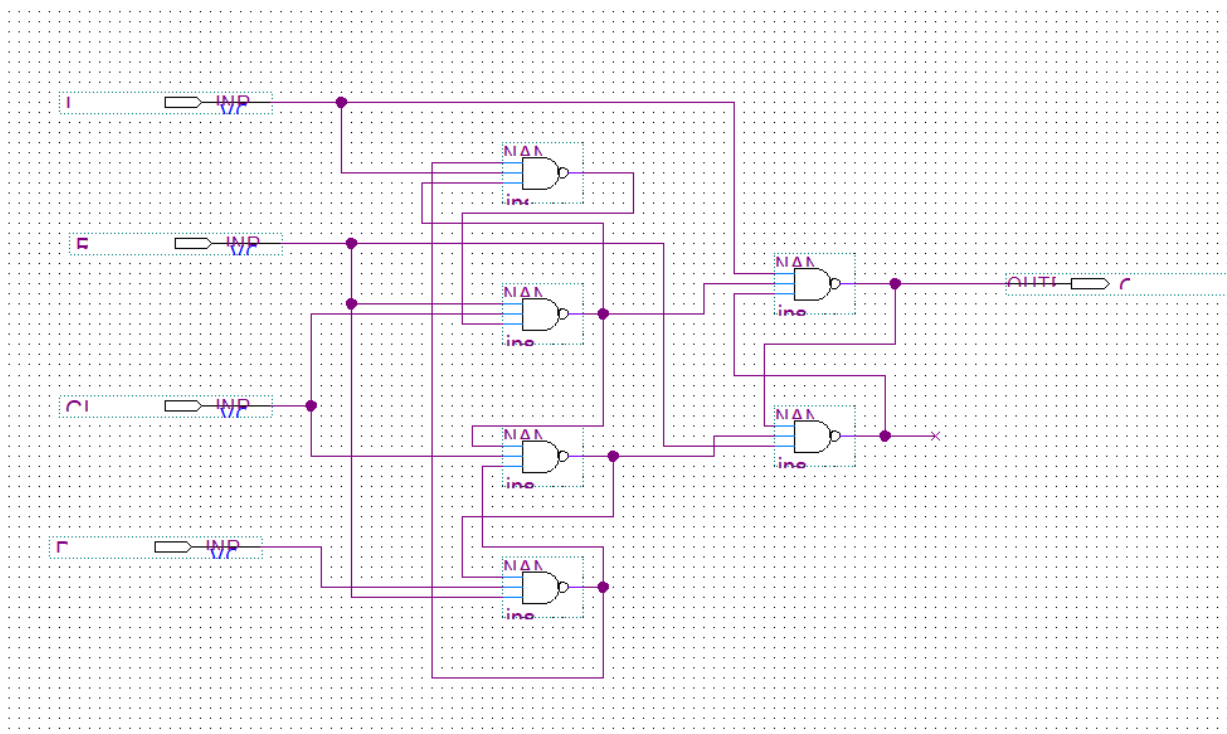


Figure 1: Schematic

QUESTION 4.2.2.2

Use your answer from 2.2.2.1 to modify the Verilog code provided for the D Flip-Flop in Section 4.4 to implement the new asynchronous reset and synchronous load functions.

```
1 module d_flip_flop(Q,D,R,L,Clk);
2
3 input D,R,L,Clk;
4 output Q;
5
6 reg Q = 0;
7 always @(posedge Clk or negedge R) begin
8   if(R == 0) Q = 0;
9   else if (L)
10    Q = D;
11   else
12    Q = Q;
13 end
14 endmodule
15
```

Figure 2: Verilog Code of DFF

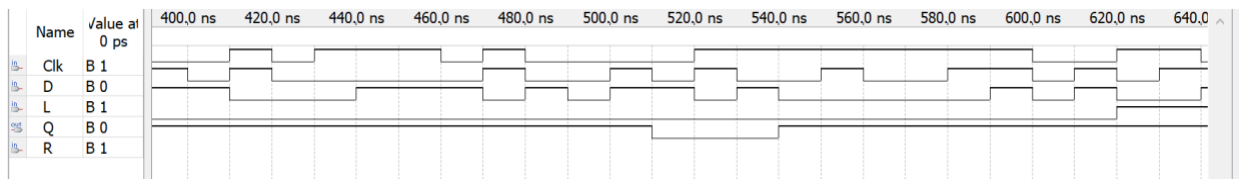


Figure 3: Waveform of DFF

QUESTION 4.2.2.3

Write a Verilog Testbench code to simulate and verify the functionality in ModelSim®

```
1 module d_flip_flop_tb();
2
3 reg D,Clk,R,L;
4 wire Q;
5
6 d_flip_flop DUT(Q,D,R,L,Clk);
7
8 always #400 Clk = ~Clk;
9 initial
10 begin
11   Clk = 0;
12   D = 1; R = 0; L = 1; #200;
13   D = 1; R = 1; L = 0; #200;
14   D = 1; R = 1; L = 1; #200;
15   D = 1; R = 1; L = 1; #200;
16   D = 1; R = 0; L = 1; #200;
17   D = 0; R = 1; L = 0; #200;
18   D = 1; R = 1; L = 1; #200;
19   D = 1; R = 1; L = 1; #200;
20   D = 0; R = 1; L = 1; #200;
21   D = 0; R = 1; L = 1; #200;
22 end
23 endmodule
24
```

Figure 4: TestBench of DFF

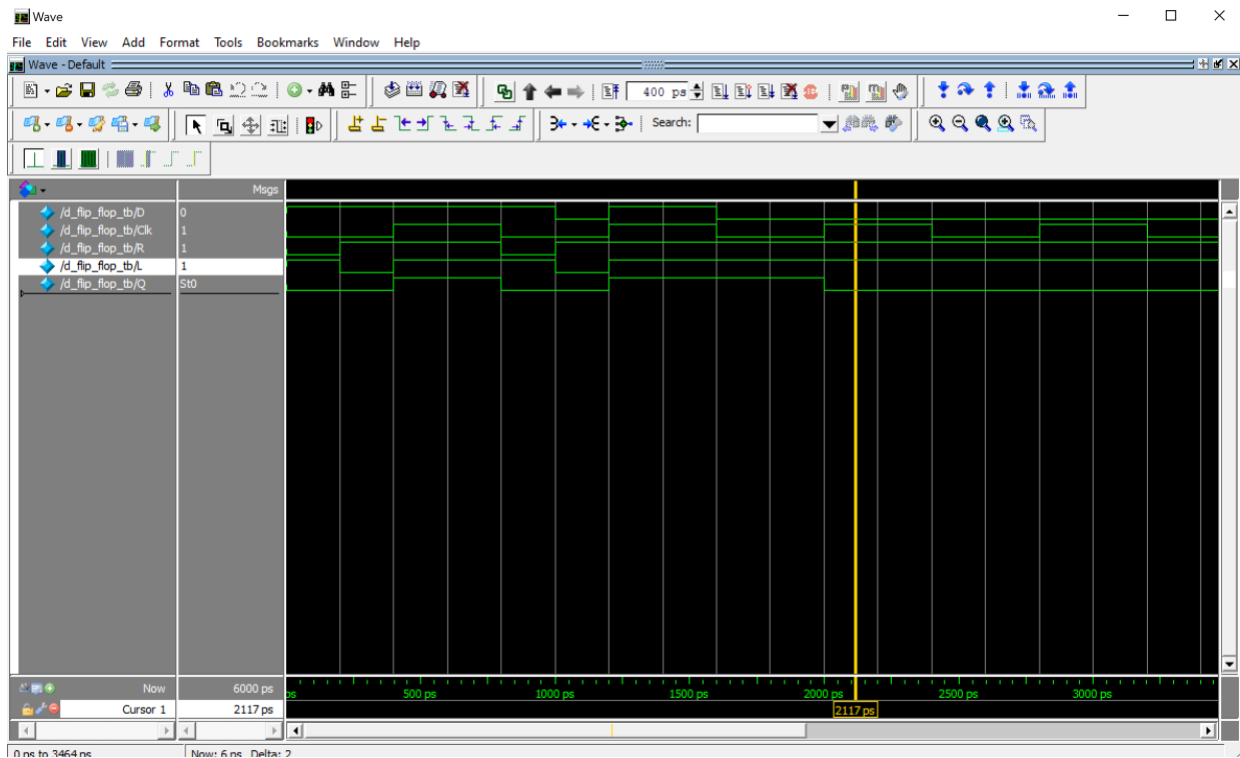


Figure 5: Simulation of DFF

Comments:

- I have implemented the D Flip-Flop with Reset and Load inputs. The Reset input is asynchronous and the Load input is synchronous. It means that R affect the design independtly from Clk, and L is depend on Clk.
- We can easily see that $D(t+1) = D$ in the waveform.
- For example, when clock rising and the L is '1', we get the output as '1'.
- In the yellow line in the Figure 5, we see that D is 0, Clk is 1, R is 1 and L is 1. Clock is rising, and the reset is 1(Active low), so we need to load the data. So output is '0' as we can see.
- This part is essential because, in its absence, the circuit reduces to a combinational circuit.
- By the help of it, we can store one-bit information.

4.2.3 4-BIT SHIFT REGISTER WITH ASYNCHRONOUS RESET AND SYNCHRONOUS LOAD

QUESTION 4.2.3.1

Use the **new D Flip-Flop** in Figure 1 and design a 4-bit scalable bit-sliced synchronous shift register (with SI input and SO output) with synchronous Load and asynchronous Reset. The symbol and a table that describes the operation of the register are depicted in Figure 2.

QUESTION 4.2.3.2

The register has four input and one output. Write a structural (hierarchical) Verilog HDL code by explicitly declaring the D Flip-Flop created in 4.2.2.

```
1 module Shift_Register(SI, L, R, Clk, Q, SO);
2
3   output [3:0] Q;
4   output SO;
5
6   input SI;
7   input L, R, Clk;
8
9
10
11  d_flip_flop U0(Q[3], SI, R, L, Clk);
12  d_flip_flop U1(Q[2], Q[3], R, L, Clk);
13  d_flip_flop U2(Q[1], Q[2], R, L, Clk);
14  d_flip_flop U3(Q[0], Q[1], R, L, Clk);
15  assign SO = Q[0];
16
17 endmodule
```

Figure 6: Verilog Code of Shift Register

QUESTION 4.2.3.3

Use the following test patterns with the clock periods provided in Table 1. Write a Verilog Testbench code to simulate and verify the functionality in ModelSim®.

```
1 module Shift_Register_tb();
2
3   reg R, L, SI, Clk;
4   wire [3:0] Q;
5   wire SO;
6
7   Shift_Register DUT(SI, L, R, Clk, Q, SO);
8   always
9   begin
10    Clk = 1; #300; Clk = 0; #300;
11  end
12  initial
13  begin
14    R = 0; L = 0; SI = 0; #300;
15    R = 1; L = 1; SI = 1; #300;
16    R = 1; L = 1; SI = 0; #300;
17    R = 1; L = 1; SI = 1; #300;
18    R = 0; L = 1; SI = 0; #300;
19    R = 1; L = 1; SI = 0; #300;
20    R = 1; L = 1; SI = 1; #300;
21  end
22 endmodule
23
```

Figure 7: TestBench of Shift Register

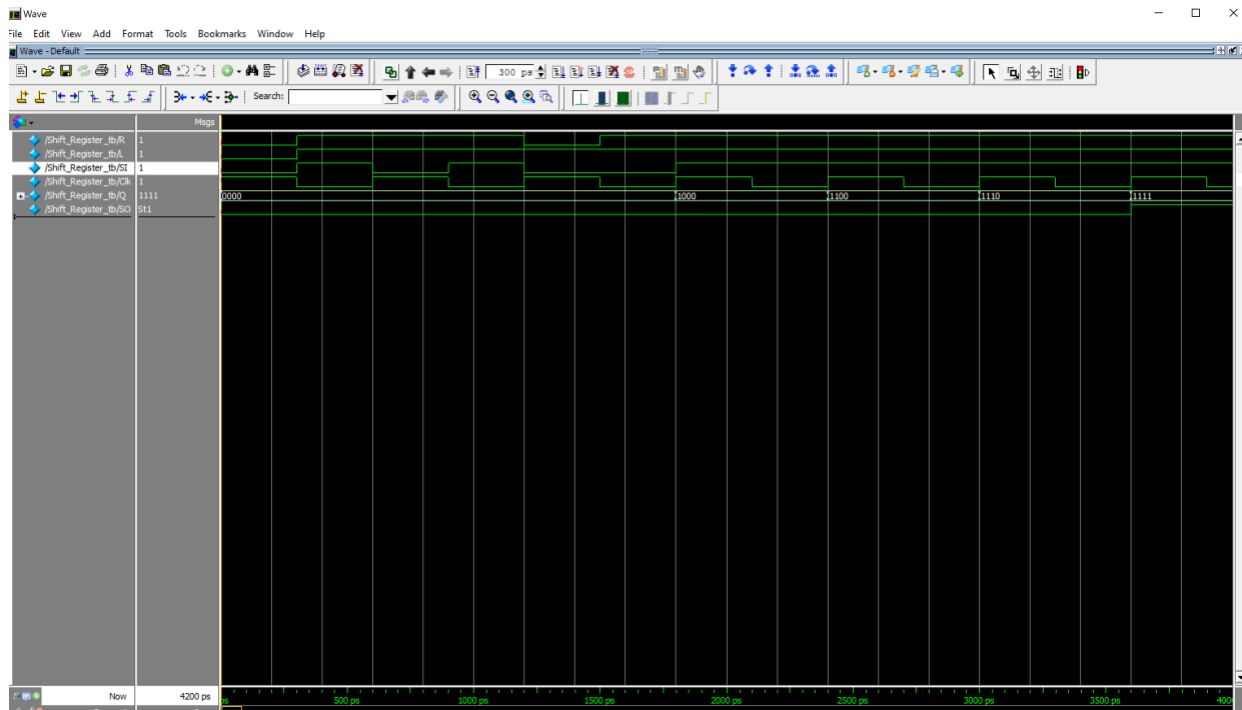


Figure 8: Simulation of Shift Register

Comments:

- I have implemented the shift register by the help of Figure 2(b) in the lab manual.
- After drawing the schematic, I just called the D Flip-Flop in the module.
- Each bit is shifting by the help of FFs.
- At the each clock pulses, one bit of information shifts to the right.
- The SI represents the MSB, and the SO represents the LSB.
- As we can see in the simulation in Figure 9, binary information shifts to the right according to SI.

4.2.4 4-BIT SYNCHRONOUS WRAP-AROUND UP/DOWN COUNTER WITH SYNCNHRONOUS UP/DOWN SELECT AND COUNT-ENABLE AND ASYNCHRONOUS RESET

QUESTION 4.2.4.1

Write a structural (hierarchical) Verilog HDL code by explicitly declaring the D Flip-Flop to create a scalable design of a 4-bit synchronous wrap-around up/down counter with synchronous up/down select and count-enable controls, and asynchronous reset as depicted in Figure 3. **Do not use if statement (behavioural) approach.**

| UP | A | B | C | D | A+ | B+ | C+ | D+ | D3 | D2 | D1 | D0 |
|----|---|---|---|---|----|----|----|----|----|----|----|----|
| 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 |
| 0 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 1 |
| 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 1 | 0 | 0 |
| 0 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 1 |
| 0 | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 1 | 0 |
| 0 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 1 |
| 0 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 0 |
| 0 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 1 |
| 0 | 1 | 0 | 1 | 0 | 1 | 0 | 0 | 1 | 1 | 0 | 1 | 0 |
| 0 | 1 | 0 | 1 | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 1 |
| 0 | 1 | 1 | 0 | 0 | 1 | 0 | 1 | 1 | 1 | 1 | 0 | 0 |
| 0 | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 1 |
| 0 | 1 | 1 | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 0 |
| 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 |
| 1 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 0 |
| 1 | 0 | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 1 |
| 1 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 0 |
| 1 | 0 | 1 | 0 | 1 | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 1 |
| 1 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 0 |
| 1 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 1 |
| 1 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 0 |
| 1 | 1 | 0 | 0 | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 0 | 1 |
| 1 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 1 | 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | 0 | 1 | 1 |
| 1 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 0 |
| 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 |
| 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 |

Figure 9: Truth Table of the Design

| | | | |
|---|---|---|---|
| 1 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 |
| 1 | 1 | 1 | 1 |
| 0 | 1 | 1 | 1 |

ABCD K-Map Up/Down = 0

| | | | |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 |
| 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 1 |

ABCD K-Map Up/Down = 1

$$D3 = U'A'B'C'D' + ABC' + U'AD + UA'BCD + UAB' + ACD'$$

| | | | |
|---|---|---|---|
| 1 | 0 | 0 | 0 |
| 0 | 1 | 1 | 1 |
| 0 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 |

ABCD K-Map Up/Down = 0

| | | | |
|---|---|---|---|
| 0 | 0 | 1 | 0 |
| 1 | 1 | 0 | 1 |
| 1 | 1 | 0 | 1 |
| 0 | 0 | 1 | 0 |

ABCD K-Map Up/Down = 1

$$D2 = BCD' + BC'U + BDU' + B'CDU + B'C'D'U'$$

| | | | |
|---|---|---|---|
| 1 | 0 | 0 | 0 |
| 0 | 1 | 1 | 1 |
| 0 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 |

ABCD K-Map Up/Down = 0

| | | | |
|---|---|---|---|
| 1 | 0 | 0 | 0 |
| 0 | 1 | 1 | 1 |
| 0 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 |

ABCD K-Map Up/Down = 1

$$D1 = U \text{ XNOR } (C \text{ XOR } D)$$

| | | | |
|---|---|---|---|
| 1 | 0 | 0 | 0 |
| 0 | 1 | 1 | 1 |
| 0 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 |

ABCD K-Map Up/Down = 0

| | | | |
|---|---|---|---|
| 1 | 0 | 0 | 0 |
| 0 | 1 | 1 | 1 |
| 0 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 |

ABCD K-Map Up/Down = 1

$D0 = U \text{ XNOR } D$

```

1 module UpDown(updown, E, R, Clk, Q);
2
3 input updown, E, R, Clk;
4 output [3:0]Q;
5 wire W[20:0];
6
7
8 not(W[0],updown);
9 not(W[1],Q[0]);
10 not(W[2],Q[1]);
11 not(W[3],Q[2]);
12 not(W[4], Q[3]);
13
14 //da part
15 and (W[5], W[0], W[4], W[3], W[2], W[1]); //u'a'b'c'd'
16 and (W[6], Q[3], Q[2], W[2]); //abc'
17 and (W[7], W[0], Q[3], Q[0]); //u'ad
18 and (W[8], updown, W[4], Q[2], Q[1], Q[0]); //ua'bcd
19 and (W[9], updown, Q[3], W[3]); //uab'
20 and (W[10], Q[3], Q[1], W[1]); //acd'
21 or (W[11], W[5], W[6], W[7], W[8], W[9], W[10]);
22
23 //db part
24 and (W[12], Q[2], Q[1], W[1]); //bcd'
25 and (W[13], Q[2], W[2], updown); //bc'u
26 and (W[14], Q[2], Q[0], W[0]); //bdu'
27 and (W[15], W[3], Q[1], Q[0], updown); //b'cdu
28 and (W[16], W[3], W[2], W[1], W[0]); //b'c'd'u'
29 or(W[17], W[12], W[13], W[14], W[15], W[16]);
30
31 //dc part
32 xor (W[18], Q[1], Q[0]);
33 xnor(W[19], updown, W[18]);
34
35 //dd part
36 xnor (W[20], updown, Q[0]);
37
38
39
40
41 d_flip_flop U0(Q[0],W[11],R, E, Clk);
42 d_flip_flop U1(Q[1],W[17],R, E, Clk);
43 d_flip_flop U2(Q[2],W[19],R, E, Clk);
44 d_flip_flop U3(Q[3],W[20],R, E, Clk);
45 endmodule

```

Figure 10: Verilog Code of Up/Down Counter

QUESTION 4.2.4.2

Modify your code to connect a 4-bit output to 7-segment decoder designed and used in LAB 2&3

```
1 module UpDown(updown, E, R, Clk, Q, number);
2
3 input updown, E, R, Clk;
4 output [3:0] Q;
5 output [15:0] number;
6 wire W[20:0];
7
8 not(W[0],updown);
9 not(W[1],Q[0]);
10 not(W[2],Q[1]);
11 not(W[3],Q[2]);
12 not(W[4], Q[3]);
13
14 //da part
15 and (W[5], W[0], W[4], W[3], W[2], W[1]); //u'a'b'c'd'
16 and (W[6], Q[3], Q[2], W[2]); //abc'
17 and (W[7], W[0], Q[3], Q[0]); //u'ad
18 and (W[8], updown, W[4], Q[2], Q[1], Q[0]); //ua'bcd
19 and (W[9], updown, Q[3], W[3]); //uab'
20 and (W[10], Q[3], Q[1], W[1]); //acd'
21 or (W[11], W[5], W[6], W[7], W[8], W[9], W[10]);
22
23 //db part
24 and (W[12], Q[2], Q[1], W[1]); //bcd'
25 and (W[13], Q[2], W[2], updown); //bc'u
26 and (W[14], Q[2], Q[0], W[0]); //bdu'
27 and (W[15], W[3], Q[1], Q[0], updown); //b'cdu
28 and (W[16], W[3], W[2], W[1], W[0]); //b'c'd'u'
29 or(W[17], W[12], W[13], W[14], W[15], W[16]);
30
31 //dc part
32 xor (W[18], Q[1], Q[0]);
33 xnor(W[19], updown, W[18]);
34
35 //dd part
36 xnor (W[20], updown, Q[0]);
37
38 d_flip_flop U0(Q[0],W[11],R, E, Clk);
39 d_flip_flop U1(Q[1],W[17],R, E, Clk);
40 d_flip_flop U2(Q[2],W[19],R, E, Clk);
41 d_flip_flop U3(Q[3],W[20],R, E, Clk);
42
43 Decoder D1(Q, number);
44 endmodule
```

Figure 11: Up/Down with Decoder

QUESTION 4.2.4.3

Use the following test patterns with the clock periods provided in Table 2. Write a Verilog Testbench code to simulate and verify the functionality in ModelSim®.

```
1  module UpDown_tb ();
2      reg Clk, R,E, updown;
3      wire [3:0] Q;
4      wire [15:0] number;
5      UpDown DUT(updown,E, R, Clk, Q, number);always
6      begin
7          Clk = 1; #2000; Clk = 0; #2000;
8      end
9      initial
10     begin
11         R =0; updown=0;E=0; #2000;
12         R =1; updown=0;E=0; #2000;
13         R =1; updown=1;E=1; #2000;
14         R =1; updown=0;E=1; #2000;
15     end
16 endmodule
17
18
```

Figure 12: TestBench of Up/Down Counter

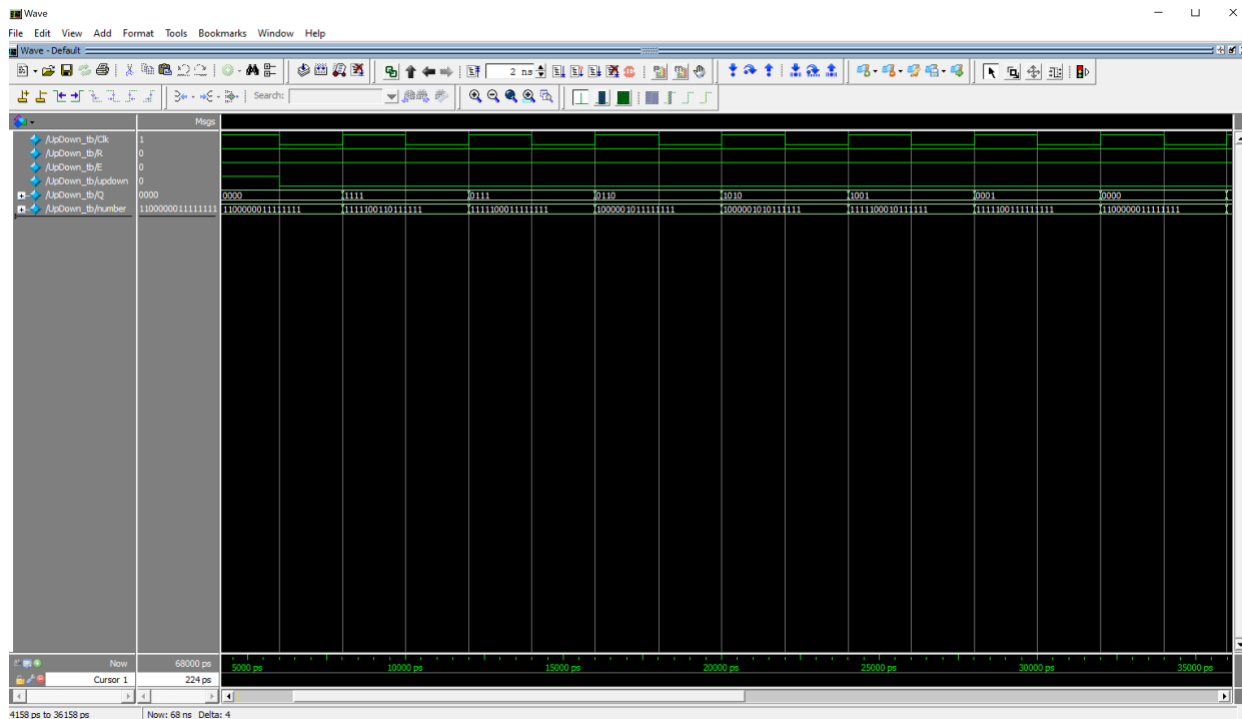


Figure 13: Simulation of Up/Down Counter

Comments:

- I have implemented the wrap-around up/down counter. The logic is that when the up is '1', the number increases by one, otherwise it decreases by one.
- To implement it, I started with deriving a truth table, then I drew K-Maps and got the simplified boolean expressions.
- According to them I wrote the Verilog Code, then I tested it in Modelsim.
- After that, I connected the output to Decode we did in Lab2&3.
- As we can see in the simulation in Figure 13, the output increases when up is 1, otherwise it decreases at the clock pulses.

4.3 LAB 5

4.3.1 3-BIT DOWN COUNTER USING D-TYPE FLIP-FLOP

Design a 3-bit **down** counter with asynchronous clear using rising-edge triggered D-type flip-flops. The count should wrap around from "000" to "111".

| A | B | C | A+ | B+ | C+ | D _A | D _B | D _C | J _A | K _A | J _B | K _B | J _C | K _C |
|---|---|---|----|----|----|----------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|
| 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | X | 1 | X | 1 | X |
| 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | X | 0 | X | X | 1 |
| 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | X | X | 1 | 1 | X |
| 0 | 1 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | X | X | 0 | X | 1 |
| 1 | 0 | 0 | 0 | 1 | 1 | 0 | 1 | 1 | X | 1 | 1 | X | 1 | X |
| 1 | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | X | 0 | 0 | X | X | 1 |
| 1 | 1 | 0 | 1 | 0 | 1 | 1 | 0 | 1 | X | 0 | X | 1 | 1 | X |
| 1 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 0 | X | 0 | X | 0 | X | 1 |

Figure 14: Truth Table of the Design

| | | | |
|---|---|---|---|
| 1 | 0 | 0 | 0 |
| 0 | 1 | 1 | 1 |

ABC K-Map

$$DA = A'B'C' + AC + AB$$

| | | | |
|---|---|---|---|
| 1 | 0 | 0 | 1 |
| 1 | 0 | 0 | 1 |

ABC K-Map

$$DC = C'$$

| | | | |
|---|---|---|---|
| 1 | 0 | 1 | 0 |
| 1 | 0 | 1 | 0 |

ABC K-Map

$$DB = B \text{ XNOR } C$$

```

1  module DownCounterD(CLR, Clk, Q);
2
3  input CLR, Clk;
4  output [2:0]Q;
5  wire [7:0]W;
6
7  not(W[0], Q[0]);
8  not(W[1], Q[1]);
9  not(W[2], Q[2]);
10
11 //a part
12 and(W[3],W[2],W[1],W[0]); //a'b'c'
13 and(W[4], Q[2],Q[0]); //ac
14 and(W[5],Q[2],Q[1]); //ab
15 or(W[6],W[3],W[4],W[5]);
16
17 //b part
18 xnor(W[7],Q[1],Q[0]);
19
20 d_flip_flop F0(Q[2],W[6],CLR, Clk);
21 d_flip_flop F1(Q[1],W[7],CLR, Clk);
22 d_flip_flop F2(Q[0],W[0],CLR, Clk);
23 endmodule
24

```

Figure 15: Verilog Code of Down Counter with D FF

```

1  module DownCounterD_tb();
2
3      wire [2:0]Q;
4      reg CLR, Clk;
5
6      DownCounterD DUT(CLR, Clk, Q);
7
8      initial begin
9          CLR = 1;
10     end
11
12     always begin
13         Clk = 100; #100; Clk = 1; #100;
14     end
15 endmodule
16

```

Figure 16: TestBench of Down Counter with DFF

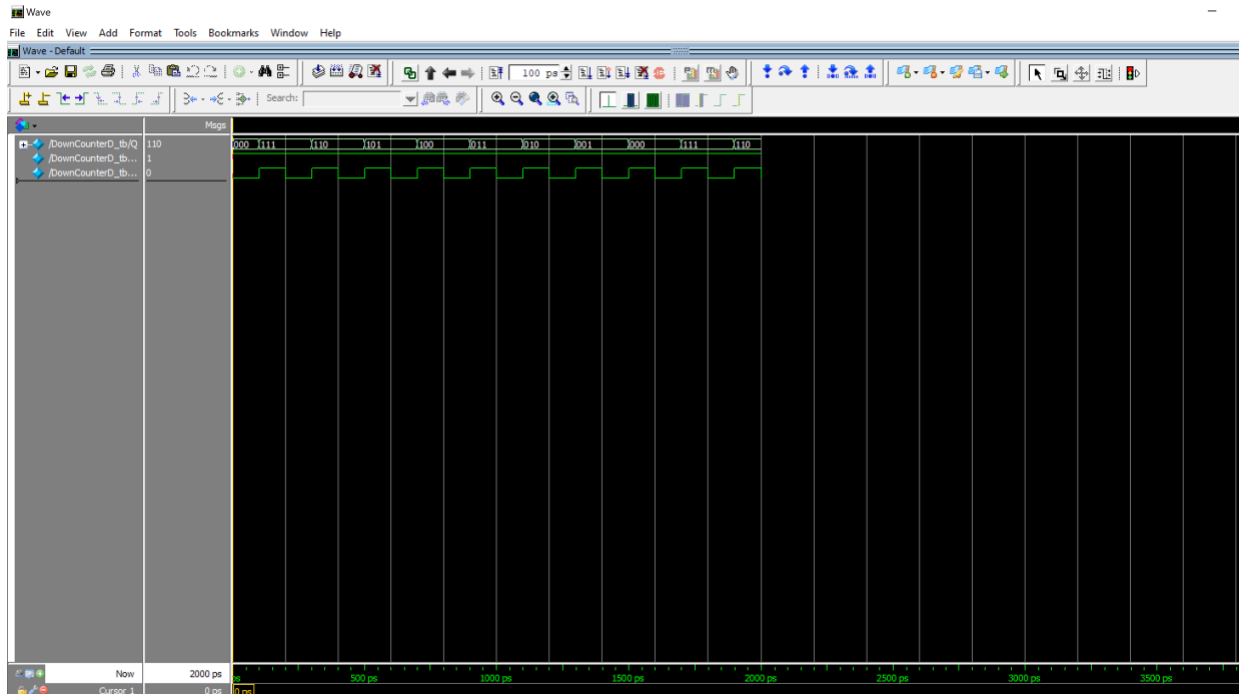


Figure 17: Simulation of DownCounter with DFF

Comments:

- I have implemented the down counter with D FFs.
- Firstly, I derived the truth table, then I drew K-Maps.
- After getting simplified boolean expressions, I wrote the Verilog Code.
- I tested and verified it in Modelsim.
- As we can see in the simulation in Figure 17, it counts down.

4.3.2 3-BIT DOWN COUNTER USING JK-TYPE FLIP-FLOP

Design a 3-bit down counter with asynchronous clear using negative-edge triggered JK-type flip-flops with asynchronous active low clear.

By using the truth table in Figure 14:

| | | | |
|----------|----------|----------|----------|
| 1 | 0 | 0 | 0 |
| X | X | X | X |

ABC K-MAP

$$J_a = B'C'$$

| | | | |
|----------|----------|----------|----------|
| 1 | 0 | X | X |
| 1 | 0 | X | X |

ABC K-MAP

$$J_b = C'$$

| | | | |
|----------|----------|----------|----------|
| 1 | X | X | 1 |
| 1 | X | X | 1 |

ABC K-MAP

$$J_c = 1$$

| | | | |
|----------|----------|----------|----------|
| X | X | X | X |
| 1 | 0 | 0 | 0 |

ABC K-MAP

$$K_a = B'C'$$

| | | | |
|----------|----------|----------|----------|
| X | X | 0 | 1 |
| X | X | 0 | 1 |

ABC K-MAP

$$K_b = C'$$

| | | | |
|----------|----------|----------|----------|
| X | 1 | 1 | X |
| X | 1 | 1 | X |

ABC K-MAP

$$J_c = 1$$


```

1 module DownCounterJK(Q,Clk);
2
3 output [2:0]Q;
4 input Clk;
5 wire [7:0]W;
6
7 not(W[0],Q[1]); //b'
8 not(W[1],Q[0]); //c'
9
10 and(W[2],W[0],W[1]); //JA,KA
11
12 jk_flip_flop F0(W[2],W[2],Q[2],Clk);
13 jk_flip_flop F1(W[1],W[1],Q[1],Clk);
14 jk_flip_flop F2(1,1,Q[0],Clk);
15 endmodule
16

```

Figure 18: Verilog Code of Down Counter with JK FF

C:/Users/mertc/Desktop/DownCounterJK/Modelsim/DownCounterJK_tb.

| Ln# | |
|-----|---------------------------------|
| 1 | module DownCounterJK_tb(); |
| 2 | |
| 3 | wire [2:0]Q; |
| 4 | reg Clk; |
| 5 | |
| 6 | DownCounterJK DUT(Q,Clk); |
| 7 | always begin |
| 8 | Clk = 100; #100; Clk = 1; #100; |
| 9 | end |
| 10 | endmodule |
| 11 | |

Figure 19: TestBench of Down Counter with JK FF

```

1 module jk_flip_flop(J,K,Q,Clk);
2
3 output Q;
4
5 input J,K,Clk;
6 reg Q = 0;
7
8 always@(negedge Clk) begin
9   if(J == 0)
10    begin
11      if(K == 0) Q = Q;
12      else Q = 0;
13    end
14   else begin
15     if (K == 0) Q = 1;
16     else Q = ~Q;
17   end
18 end
19 endmodule
20

```

Figure 20: Verilog Code of JK FF

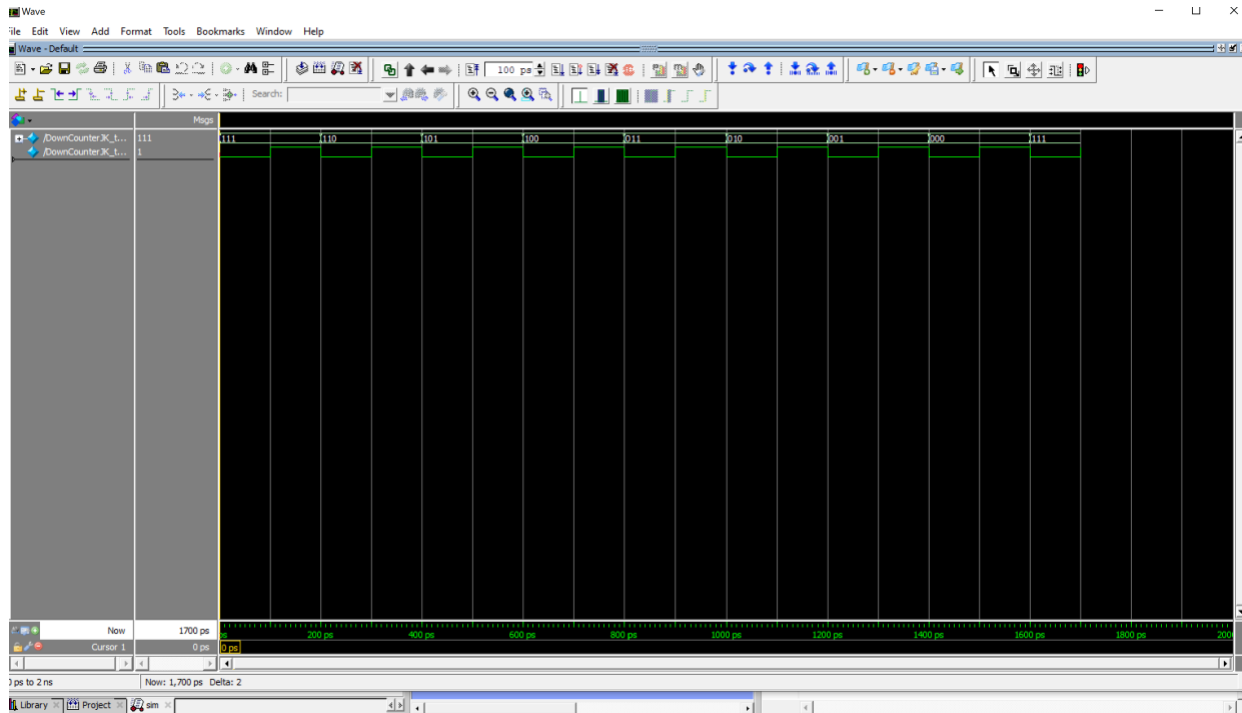


Figure 21:Simulation of Down Counter with JK FF

Comments:

- I have implemented the down counter with JK FFs.
- Firstly, I derived the truth table, then I drew K-Maps.
- After getting simplified boolean expressions, I wrote the Verilog Code.
- I tested and verified it in Modelsim.
- As we can see in the simulation in Figure 21, it counts down.

4.3.3 4-BIT SYNCHRONOUS PARALLEL LOAD SHIFT REGISTER WITH COUNTER AND ASYNCHRONOUS RESET

QUESTION 4.3.3.1

Write a structural (hierarchical) Verilog code to read an 8-bit message from parallel input (user switches) into a shift-register and use the shift function together with a counter to determine the number of '1's in the message. You have to create a top-level design and initialize shift register and counter explicitly. **Do not use if statement (behavioural) approach.** (no need for designing the 7-segment LED decoder here, the decoder will be used during the experimental work.)

```

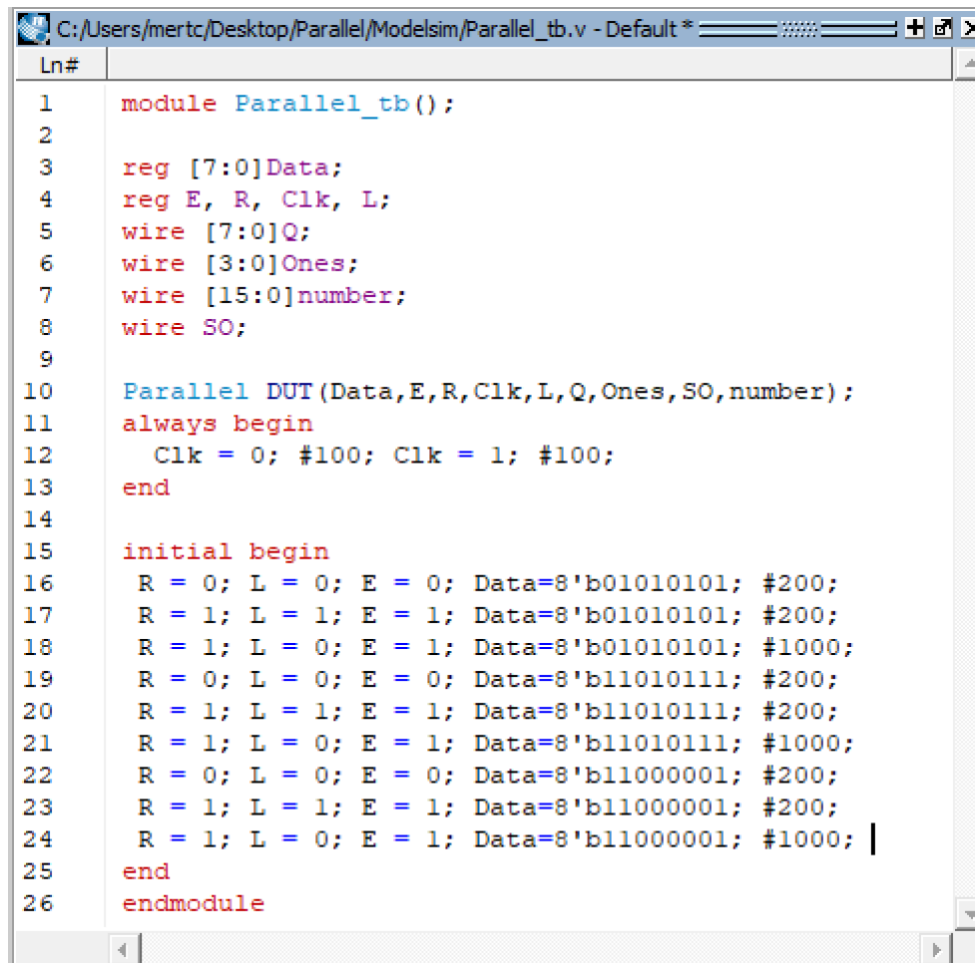
C:/Users/mertc/Desktop/Parallel/Modelsim/Parallel.v - Default
Ln#
1  module Parallel (Data,E,R,Clk,L,Q,Ones,SO,number) ;
2
3  input [7:0]Data;
4  input E, R, Clk, L;
5  output [7:0]Q;
6  output [3:0]Ones;
7  output [15:0]number;
8  output SO;
9
10 Shift_Register S0 (Data[0],L,R,Clk,Q[0],SO) ;
11 Shift_Register S1 (Data[1],L,R,Clk,Q[1],SO) ;
12 Shift_Register S2 (Data[2],L,R,Clk,Q[2],SO) ;
13 Shift_Register S3 (Data[3],L,R,Clk,Q[3],SO) ;
14
15 //count the 1's, enable up
16 UpDown U0 (Q[0], E, R, Clk, Ones, number) ;
17 UpDown U1 (Q[1], E, R, Clk, Ones, number) ;
18 UpDown U2 (Q[2], E, R, Clk, Ones, number) ;
19 UpDown U3 (Q[3], E, R, Clk, Ones, number) ;
20
21 //shift again
22 Shift_Register S4 (Data[4],L,R,Clk,Q[4],SO) ;
23 Shift_Register S5 (Data[5],L,R,Clk,Q[5],SO) ;
24 Shift_Register S6 (Data[6],L,R,Clk,Q[6],SO) ;
25 Shift_Register S7 (Data[7],L,R,Clk,Q[7],SO) ;
26
27 //count again
28 UpDown U4 (Q[4], E, R, Clk, Ones, number) ;
29 UpDown U5 (Q[5], E, R, Clk, Ones, number) ;
30 UpDown U6 (Q[6], E, R, Clk, Ones, number) ;
31 UpDown U7 (Q[7], E, R, Clk, Ones, number) ;
32 assign SO = Q[0];
33
34 endmodule

```

Figure 22: Verilog Code

QUESTION 4.3.3.3

Use the following test patterns with the clock periods provided in Table 2. Write a Verilog Testbench code to simulate and verify the functionality in ModelSim®.



```
1  module Parallel_tb();
2
3      reg [7:0]Data;
4      reg E, R, Clk, L;
5      wire [7:0]Q;
6      wire [3:0]Ones;
7      wire [15:0]number;
8      wire SO;
9
10     Parallel DUT(Data,E,R,Clk,L,Q,Ones,SO,number);
11     always begin
12         Clk = 0; #100; Clk = 1; #100;
13     end
14
15     initial begin
16         R = 0; L = 0; E = 0; Data=8'b01010101; #200;
17         R = 1; L = 1; E = 1; Data=8'b01010101; #200;
18         R = 1; L = 0; E = 1; Data=8'b01010101; #1000;
19         R = 0; L = 0; E = 0; Data=8'b11010111; #200;
20         R = 1; L = 1; E = 1; Data=8'b11010111; #200;
21         R = 1; L = 0; E = 1; Data=8'b11010111; #1000;
22         R = 0; L = 0; E = 0; Data=8'b11000001; #200;
23         R = 1; L = 1; E = 1; Data=8'b11000001; #200;
24         R = 1; L = 0; E = 1; Data=8'b11000001; #1000; |
25     end
26 endmodule
```

Figure 23:TestBench