Chapter 9

Chapter 9. Useful Modelling Techniques

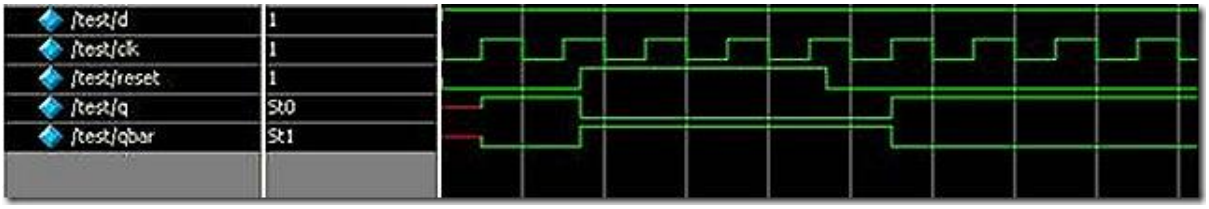
9.7 Exercises

1. Using assign and deassign statements, design a positive edge-triggered D-flipflop with asynchronous clear (q=0) and preset (q=1).

My answer:

```
1   //ex9-1 a possitive edge-triggered D_flipflop with asynchronous clear and preet.
2
3   module edge_dff(q,qbar,d,clk,reset);
4
5   //I/O prots
6   output reg q,qbar;
7   input d,clk,reset;
8
9   always @(posedge clk)
10  begin
11    q=d;
12    qbar=~d;
13  end
14
15  always @(reset)
16    if(reset)
17    begin
18      assign q=1'b0;
19      assign qbar=1'b1;
20    end
21    else
22    begin
23      deassign q;
24      deassign qbar;
25    end
26
27  endmodule
```

```
 1  module test;
 2
 3  reg d,clk,reset;
 4  wire q,qbar;
 5
 6  edge_dff d1(q,qbar,d,clk,reset);
 7
 8  initial
 9  begin
10    d=1'b1;
11    reset=1'b0;
12    #34 reset=1'b1;
13    #60 reset=1'b0;
14    #150 reset=1'b1;
15  end
16
17  initial
18  begin
19    clk=1'b0;
20    forever #10 clk=~clk;
21  end
22
23  endmodule
```



2. Using primitive gates, design a 1-bit full adder FA. Instantiate the full adder inside a stimulus module. Force the sum output to a & b & c_in for the time between 15 and 35 units.

My answer:

```
1   //ex9_2 a 1-bit full adder
2
3   module fa(a,b,cin,cout,sum);
4   output  sum,cout;
5   input a,b,cin;
6
7   wire c1,c2,c3,s1,s2,s3,s4;
8
9   and a1(c1,b,cin),
10      a2(c2,a,b),
11      a3(c3,a,cin);
12
13  or o1(cout,c1,c2,c3); //generate cout
14
15  and a4(s1,~a,~b,cin),
16      a5(s2,~a,b,~cin),
17      a6(s3,a,b,cin),
18      a7(s4,a,~b,~cin);
19
20  or o2(sum,s1,s2,s3,s4); //generate sum
21
22  endmodule
```

```
1   //stimulus
2
3   module test;
4
5   reg a,b,cin;
6   wire cout,sum;
7
8   fa f1(a,b,cin,cout,sum);
9
10  initial
11  begin
12   a=1'b0;
13   b=1'b1;
14   cin=1'b0;
15   #50 cin=1'b1;
16  end
17
18  initial
19  begin
20    #15 force sum=a&b&cin;
21    #20 release sum;
22  end
23
24  initial
25    $monitor($time , " sum= %b",sum);
26
27  endmodule
```

```
 0 sum= 1
15 sum= 0
35 sum= 1
50 sum= 0
```

3. A 1-bit full adder FA is defined with gates and with delay parameters as shown below.

// Define a 1-bit full adder

module fulladd(sum,c_out,a,b,c_in);

parameter d_sum=0,d_cout=0;

//I/O port declarations

output sum,c_out;

input a,b,c_in;

//Internal nets

wire s1,c1,c2;

//Instantiate logic gate primitives

xor(s1,a,b);

and(c1,a,b);

xor #(d_sum) (sum,s1,c_in); //delay on output sum is d_sum

and (c2,s1,c_in);

or #(d_out) (c_out,c2,c1); //delay on output c_out is d_cout

endmodule

Define a 4-bit full adder fulladd4 as shown in example 5-8, but pass the following parameter values to the instances, using the two methods discussed in the book.

Instance	Delay Values
fa0	d_sum=1,d_cout-1
fa1	d_sum=2,d_cout=2
fa2	d_sum=2,d_cout=2
fa3	d_sum=3,d_cout=3

a. Build the fulladd4 module with defparm statements to change instance parameter values. Simulate the 4-bit full adder using the stimulus shown is example 5-9 . Explain the effect of the full adder delays on the times when outputs of the adder appear. (Use delays of 20 instead of 5 used in this stimulus.)

b. Build the fulladd4 with delay values passed to instances fa0, fa1, fa2, fa3 during instantiation. Resimulate the 4-bit adder, using the stimulus above. Check if the results are identical.

My answer:

```
 1  // 4-bit full adder
 2  module fulladd4(sum,c_out,a,b,c_in);
 3
 4  output [3:0] sum;
 5  output c_out;
 6  input [3:0] a,b;
 7  input c_in;
 8
 9  wire c1,c2,c3;
10
11  defparam fa0.d_sum=1,fa0.d_cout=1,
12           fa1.d_sum=2,fa1.d_cout=2,
13           fa2.d_sum=3,fa2.d_cout=3,
14           fa3.d_sum=4,fa3.d_cout=4;
15
16  fulladd fa0 (sum[0],c1,a[0],b[0],c_in);
17  fulladd fa1 (sum[1],c2,a[1],b[1],c1);
18  fulladd fa2 (sum[2],c3,a[2],b[2],c2);
19  fulladd fa3 (sum[3],c_out,a[3],b[3],c3);
20
21  endmodule
```

```
1   //stimulus
2   module stimulus;
3
4   reg [3:0] A,B;
5   reg C_IN;
6   wire [3:0] SUM;
7   wire C_OUT;
8
9   fulladd4 f1(SUM,C_OUT,A,B,C_IN);
10
11  initial
12  begin
13    $monitor($time, " A= %b, B=%b, C_IN=%b,--- C_OUT=%b, SUM= %b\n",
14                    A,B,C_IN,C_OUT,SUM);
15  end
16
17  initial
18  begin
19    A=4'd0; B=4'd0; C_IN=1'b0;
20    #20 A=4'd3; B=4'd4;
21    #20 A=4'd2; B=4'd5;
22    #20 A=4'd9; B=4'd9;
23    #20 A=4'd10;B=4'd15;
24    #20 A=4'd10;B=4'd5;C_IN=1'b1;
25  end
26
27  endmodule
```

```
  0 A= 0000, B=0000, C_IN=0,--- C_OUT=x, SUM= xxxx
  1 A= 0000, B=0000, C_IN=0,--- C_OUT=x, SUM= xxx0
  3 A= 0000, B=0000, C_IN=0,--- C_OUT=x, SUM= xx00
  4 A= 0000, B=0000, C_IN=0,--- C_OUT=0, SUM= xx00
  5 A= 0000, B=0000, C_IN=0,--- C_OUT=0, SUM= x000
  7 A= 0000, B=0000, C_IN=0,--- C_OUT=0, SUM= 0000
 20 A= 0011, B=0100, C_IN=0,--- C_OUT=0, SUM= 0000
 21 A= 0011, B=0100, C_IN=0,--- C_OUT=0, SUM= 0001
 22 A= 0011, B=0100, C_IN=0,--- C_OUT=0, SUM= 0011
 23 A= 0011, B=0100, C_IN=0,--- C_OUT=0, SUM= 0111
 40 A= 0010, B=0101, C_IN=0,--- C_OUT=0, SUM= 0111
 60 A= 1001, B=1001, C_IN=0,--- C_OUT=0, SUM= 0111
 61 A= 1001, B=1001, C_IN=0,--- C_OUT=0, SUM= 0110
 63 A= 1001, B=1001, C_IN=0,--- C_OUT=0, SUM= 0010
 64 A= 1001, B=1001, C_IN=0,--- C_OUT=1, SUM= 0010
 80 A= 1010, B=1111, C_IN=0,--- C_OUT=1, SUM= 0010
 81 A= 1010, B=1111, C_IN=0,--- C_OUT=1, SUM= 0011
 83 A= 1010, B=1111, C_IN=0,--- C_OUT=1, SUM= 0001
 89 A= 1010, B=1111, C_IN=0,--- C_OUT=1, SUM= 1001
100 A= 1010, B=0101, C_IN=1,--- C_OUT=1, SUM= 1001
101 A= 1010, B=0101, C_IN=1,--- C_OUT=1, SUM= 1000
104 A= 1010, B=0101, C_IN=1,--- C_OUT=1, SUM= 0000
```

```
 1  // 4-bit full adder
 2  module fulladd4(sum,c_out,a,b,c_in);
 3
 4  output [3:0] sum;
 5  output c_out;
 6  input [3:0] a,b;
 7  input c_in;
 8
 9  wire c1,c2,c3;
10
11  /*defparam fa0.d_sum=1,fa0.d_cout=1,
12           fa1.d_sum=2,fa1.d_cout=2,
13           fa2.d_sum=3,fa2.d_cout=3,
14           fa3.d_sum=4,fa3.d_cout=4;*/
15
16  fulladd  #(.d_sum(1),.d_cout(1)) fa0(sum[0],c1,a[0],b[0],c_in);
17  fulladd  #(.d_sum(2),.d_cout(2)) fa1(sum[1],c2,a[1],b[1],c1);
18  fulladd  #(.d_sum(3),.d_cout(3)) fa2(sum[2],c3,a[2],b[2],c2);
19  fulladd  #(.d_sum(4),.d_cout(4)) fa3(sum[3],c_out,a[3],b[3],c3);
20
21  endmodule
```

```
  0 A= 0000, B=0000, C_IN=0,--- C_OUT=x, SUM= xxxx
  1 A= 0000, B=0000, C_IN=0,--- C_OUT=x, SUM= xxx0
  3 A= 0000, B=0000, C_IN=0,--- C_OUT=x, SUM= xx00
  4 A= 0000, B=0000, C_IN=0,--- C_OUT=0, SUM= xx00
  5 A= 0000, B=0000, C_IN=0,--- C_OUT=0, SUM= x000
  7 A= 0000, B=0000, C_IN=0,--- C_OUT=0, SUM= 0000
 20 A= 0011, B=0100, C_IN=0,--- C_OUT=0, SUM= 0000
 21 A= 0011, B=0100, C_IN=0,--- C_OUT=0, SUM= 0001
 22 A= 0011, B=0100, C_IN=0,--- C_OUT=0, SUM= 0011
 23 A= 0011, B=0100, C_IN=0,--- C_OUT=0, SUM= 0111
 40 A= 0010, B=0101, C_IN=0,--- C_OUT=0, SUM= 0111
 60 A= 1001, B=1001, C_IN=0,--- C_OUT=0, SUM= 0111
 61 A= 1001, B=1001, C_IN=0,--- C_OUT=0, SUM= 0110
 63 A= 1001, B=1001, C_IN=0,--- C_OUT=0, SUM= 0010
 64 A= 1001, B=1001, C_IN=0,--- C_OUT=1, SUM= 0010
 80 A= 1010, B=1111, C_IN=0,--- C_OUT=1, SUM= 0010
 81 A= 1010, B=1111, C_IN=0,--- C_OUT=1, SUM= 0011
 83 A= 1010, B=1111, C_IN=0,--- C_OUT=1, SUM= 0001
 89 A= 1010, B=1111, C_IN=0,--- C_OUT=1, SUM= 1001
100 A= 1010, B=0101, C_IN=1,--- C_OUT=1, SUM= 1001
101 A= 1010, B=0101, C_IN=1,--- C_OUT=1, SUM= 1000
104 A= 1010, B=0101, C_IN=1,--- C_OUT=1, SUM= 0000
```

4. Create a design that uses the full adder example above. Use a conditional compilation (`ifdef). compile the fulladd4 with defparam statements if the text macro DPARAM is defined by the `define statement; otherwise, compile the fulladd4 with module instance parameter values.

My answer:

```
1   //ex9-4 ifdef
2
3   `ifdef DPARAM
4   module fulladd4_d;
5   ...
6   endmodule
7   `else
8   module fulladd4_p;
9   ...
10  endmodule
11  `endif
```

6. What will be the output of the $display statement shown below?

```
1   //Ex9-6
2
3   module top;
4   A a1();
5   endmodule
6
7   module A;
8   B b1();
9   endmodule
10
11  module B;
12  initial
13    $display("I am inside instance %m");
14  endmodule
```

My answer:

I am inside instance top.a1.b1

7. Consider the 4-bit full adder in example 6-4. Write a stimulus file to do random testing of the full adder. Use a random number generator to generate a 32-bit random number. Pick bit 3:0 and apply them to input a; pick bits 7:4 and apply them to input b. Use bit 8 and apply it to c_in. Apply 20 random test vectors and observe the output

My answer:

```
1   //stimulus
2
3   module test;
4
5   reg [3:0] a,b;
6   reg c_in;
7   wire c_out;
8   wire [3:0] sum;
9
10  reg [31:0] addr;
11
12  fulladd4 f1(sum,c_out,a,b,c_in);
13
14  initial
15  begin
16
17    repeat (20)
18    begin
19     #10 addr=$random;
20     a=addr[3:0];
21     b=addr[7:4];
22     c_in=addr[8];
23    end
24  end
25
26  initial
27    $monitor($time , " a= %b, b= %b, c_in= %b, c_out= %b, sum= %b",
28                  a,b,c_in,c_out,sum);
29
30  endmodule
```

```
  0 a= xxxx, b= xxxx, c_in= x, c_out= x, sum= xxxx
 10 a= 0100, b= 0010, c_in= 1, c_out= 0, sum= 0111
 20 a= 0001, b= 1000, c_in= 0, c_out= 0, sum= 1001
 30 a= 1001, b= 0000, c_in= 0, c_out= 0, sum= 1001
 40 a= 0011, b= 0110, c_in= 0, c_out= 0, sum= 1001
 50 a= 1101, b= 0000, c_in= 1, c_out= 0, sum= 1110
 60 a= 1101, b= 1000, c_in= 1, c_out= 1, sum= 0110
 70 a= 0101, b= 0110, c_in= 0, c_out= 0, sum= 1011
 80 a= 0010, b= 0001, c_in= 0, c_out= 0, sum= 0011
 90 a= 0001, b= 0000, c_in= 1, c_out= 0, sum= 0010
100 a= 1101, b= 0000, c_in= 1, c_out= 0, sum= 1110
110 a= 0110, b= 0111, c_in= 1, c_out= 0, sum= 1110
120 a= 1101, b= 0011, c_in= 1, c_out= 1, sum= 0001
130 a= 1101, b= 1110, c_in= 1, c_out= 1, sum= 1100
140 a= 1100, b= 1000, c_in= 1, c_out= 1, sum= 0101
150 a= 1001, b= 1111, c_in= 1, c_out= 1, sum= 1001
160 a= 0110, b= 1100, c_in= 0, c_out= 1, sum= 0010
170 a= 0101, b= 1100, c_in= 0, c_out= 1, sum= 0001
180 a= 1010, b= 1010, c_in= 0, c_out= 1, sum= 0100
190 a= 0101, b= 1110, c_in= 1, c_out= 1, sum= 0100
200 a= 0111, b= 0111, c_in= 0, c_out= 0, sum= 1110
```

8. Use the 8-bit memory initialization example in example 9-14 . Modify the file to read data in hexadecimal. Write a new data file with the following addresses and data values. Unspecified locations are not initialized.

Location Address	Data
1	33
2	66
4	z0
5	0z
6	01

My answer:

```
1   //ex9-8
2   module test;
3
4   reg [7:0] memory [0:7];
5   integer i;
6
7   initial
8   begin
9     $readmemh("init.dat",memory);
10
11    for(i=0;i<8;i=i+1)
12      $display("Memory [%0d] = %h",i,memory[i]);
13  end
14
15  endmodule
```

```
# Memory [0] = xx
# Memory [1] = 33
# Memory [2] = 66
# Memory [3] = xx
# Memory [4] = z0
# Memory [5] = 0z
# Memory [6] = 01
# Memory [7] = xx
```

9. Write an initial block that controls the VCD file. The initial block must do the following:

l Set myfile.dmp as the output VCD file.

l Dump all variables two levels deep in module instance top.a1.b1.c1.

l Stop dumping to VCD at time 200.

l Start dumping to VCD at time 400.

l Stop dumping to VCD at time 500.

l Create a checkpoint. Dump the current value of all VCD variables to the dumpfile.

My answer:

```
 1    //ex9-9
 2
 3    initial
 4      $dumpfile("myfile.dmp");
 5
 6    initial
 7      $dumpvars(2,top.a1.b1.c1);
 8
 9    initial
10    begin
11      #200 $dumpoff;
12      #200 $dumpon;
13      #100 $dumpoff;
14    end
15
16    initial
17      $dumpall;
```