Chapter 7 (Part 3)

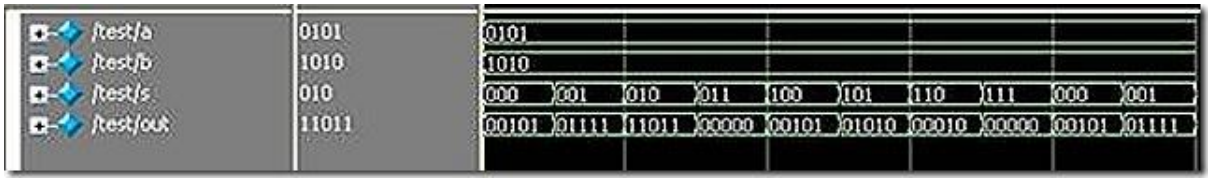
12. Using a case statement, design an 8-function ALU that takes 4-bit inputs a and b and a 3-bit input signal select, and gives a 5-bit output out. The ALU implements the following functions based on a 3-bit input signal select. Ignore any overflow or underflow bits.

Select Signal	Function
3'b000	Out=a
3'b001	Out=a+b
3'b010	Out=a-b
3'b011	Out=a/b
3'b100	Out=a%b(remainder)
3'b101	Out=a<<1
3'b110	Out=a>>1
3'b111	Out=(a>b)(magnitude compare)

My answer:

```
1   //ex7_12 alu
2   module alu_8(out,a,b,s);
3
4   output reg [4:0] out;
5   input [3:0] a,b;
6   input [2:0] s;
7
8   always @(a,b,s)
9   begin
10  case(s)
11  3'b000:out=a;
12  3'b001:out=a+b;
13  3'b010:out=a-b;
14  3'b011:out=a/b;
15  3'b100:out=a%b;
16  3'b101:out=a<<1;
17  3'b110:out=a>>1;
18  3'b111:out=(a>b);
19  endcase
20  end
21
22  endmodule
```

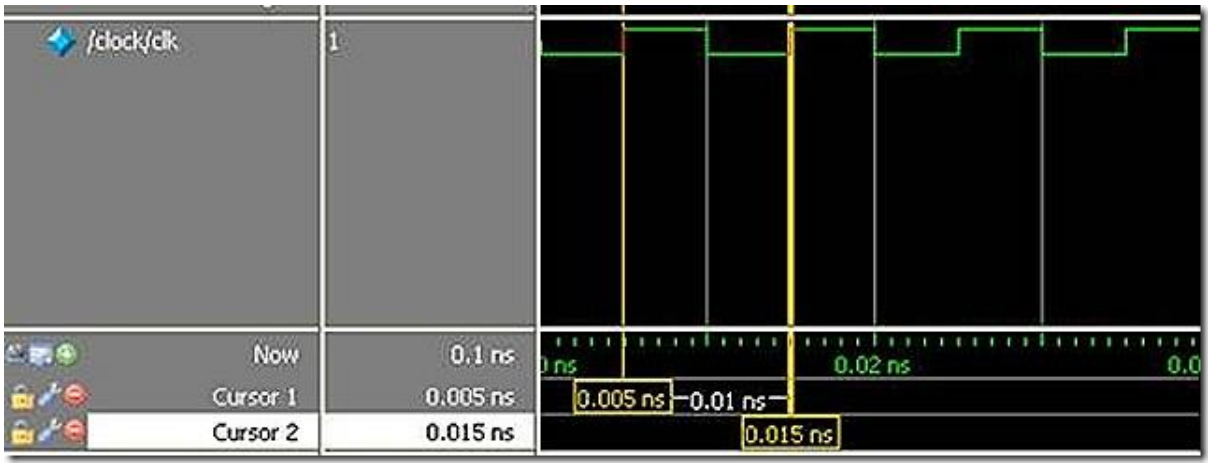
```
1   //stimulus
2   module test;
3   reg [3:0] a,b;
4   reg [2:0] s;
5   wire [4:0] out;
6
7   alu_8 m1(.out(out),.a(a),.b(b),.s(s));
8
9   initial
10  begin
11    a=4'b0101; b=4'b1010;
12    s=3'b000;
13  end
14
15  always
16    #10 s=s+1'b1;
17
18  endmodule
```



13. Using a while loop, design a clock generator. Initial value of clock is 0. Time period for the clock is 10.

My answer:

```
1   //ex7_13 clock
2   module clock;
3   reg clk;
4
5   initial
6   begin
7      clk=1'b0;
8      while(1)
9      begin
10        $display("clk= %b",clk);
11        #5 clk=~clk;
12     end
13  end
14
15  endmodule
```



14. Using the for loop, initialize locations 0 to 1023 of a 4-bit register array cache_var to 0.

My answer:

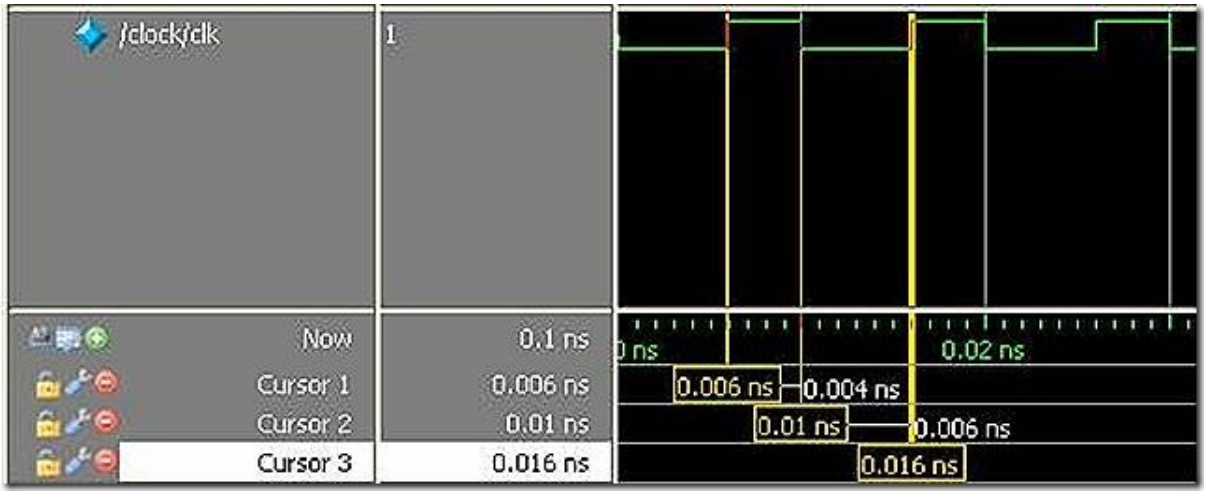
```
1   //ex7_14 for statement
2   module test_for;
3   reg [3:0] cache_var [1023:0];
4
5   integer i;
6
7   initial
8   begin
9     for(i=0;i<1024;i=i+1)
10      begin
11      cache_var[i]=4'b0;
12      $display("cache_var[i=%d]= %b",i,cache_var[i]);
13      end
14  end
15
16  endmodule
```

```
cache_var[i=          0]= 0000
cache_var[i=          1]= 0000
cache_var[i=          2]= 0000
cache_var[i=          3]= 0000
cache_var[i=          4]= 0000
cache_var[i=          5]= 0000
cache_var[i=          6]= 0000
cache_var[i=          7]= 0000
cache_var[i=          8]= 0000
cache_var[i=          9]= 0000
cache_var[i=         10]= 0000
```

15. Using forever statement, design a clock with time period =10 and duty cycl = 40%. Initial value of clock is 0.

My answer:

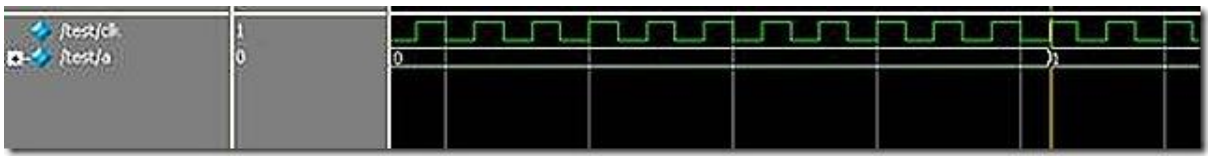
```
1   //ex7_15 forever
2   module clock;
3   reg clk;
4
5   initial
6   begin
7      clk=1'b0;
8      forever
9      begin
10        #6 clk=~clk;
11        #4 clk=~clk;
12     end
13  end
14
15  endmodule
```



16. Using the repeat loop, delay the statement a= a+1 by 20 positive edges of clock.

My answer:

```
1   //stimulus
2   module test;
3   reg clk;
4   integer a;
5
6   initial
7   begin
8    a=0;
9    clk=1'b0;
10   forever #10 clk=~clk;
11  end
12
13  initial
14    $monitor($time ," a= %d",a);
15
16  always @(posedge clk)
17  begin
18    repeat(20)
19    @(posedge clk) a=a;
20    a=a+1;
21  end
22
23  initial
24   #500 $finish;
25
26  endmodule
```



17. Below is a block with nested sequential and parallel blocks. When does the block finish and what is the order of execution of events? At what simulation times does each statement finish execution?

```
initial
begin
        x = 1'b0;
        #5 y = 1'b1;
        fork
                #20 a = x;
                #15 b = y;
        join
        #40 x = 1'b1;
        fork
                #10 p = x;
                begin
                        #10 a = y;
                        #30 b = x;
                end
                #5 m = y;
        join
end
```

My answer:

```
1   //ex7_17
2   module test;
3   reg x,y,a,b,p,m,c;
4
5   initial
6   begin
7     x=1'b0;
8     c=1'bx;
9     #5 y = 1'b1;
10          fork
11                  #20 a = x;
12                  #15 b = y;
13          join
14          #40 x = 1'b1;
15          fork
16                  #10 p = x;
17                  begin
18                          #10 a = y;
19                          #30 b = c;
20                  end
21                  #5 m = y;
22          join
23  end
24
25  initial
26    $monitor($time, " x=%b,y=%b, a= %b, b=%b, p=%b, m=%b",x,y,a,b,p,m);
27
28  endmodule
```

```
  0 x=0,y=x, a= x, b=x, p=x, m=x
  5 x=0,y=1, a= x, b=x, p=x, m=x
 20 x=0,y=1, a= x, b=1, p=x, m=x
 25 x=0,y=1, a= 0, b=1, p=x, m=x
 65 x=1,y=1, a= 0, b=1, p=x, m=x
 70 x=1,y=1, a= 0, b=1, p=x, m=1
 75 x=1,y=1, a= 1, b=1, p=1, m=1
105 x=1,y=1, a= 1, b=x, p=1, m=1
```

18. Design an 8-bit counter by using forever loop, named block, and disabling of named block. The counter starts counting at count=5 and finishes at count=67. The count is incremented at positive edge of clock. The clock has a time period of 10. The counter counts through the loop only once and then is disabled.(Hint: Use the disable statement).

My answer:

```
1   //ex7_18
2   module test;
3   reg clock;
4   reg [7:0] count;
5
6   initial
7   begin
8     clock=1'b0;
9     forever #5 clock=~clock;
10  end
11
12  initial
13    begin
14      count=8'd5;
15      begin:block1
16        forever
17        begin
18                @(posedge clock) count=count+1;
19                if(count>66 || count<5)
20                   disable block1;
21        end
22      end
23    end
24
25  initial
26    $monitor($time ," count= %d",count);
27
28  endmodule
```

0 count= 5

5 count= 6

15 count= 7

25 count= 8

35 count= 9

45 count= 10

55 count= 11

65 count= 12

75 count= 13

85 count= 14

95 count= 15

105 count= 16

115 count= 17

125 count= 18

135 count= 19

145 count= 20

155 count= 21

165 count= 22

175 count= 23

185 count= 24

195 count= 25

205 count= 26

215 count= 27

225 count= 28

235 count= 29

245 count= 30

255 count= 31

265 count= 32

275 count= 33

285 count= 34

295 count= 35

305 count= 36

315 count= 37

325 count= 38

335 count= 39

345 count= 40

355 count= 41

365 count= 42

375 count= 43

385 count= 44

395 count= 45

405 count= 46

415 count= 47

425 count= 48

435 count= 49

445 count= 50

455 count= 51

465 count= 52

475 count= 53

485 count= 54

495 count= 55

505 count= 56

515 count= 57

525 count= 58

535 count= 59

545 count= 60

555 count= 61

565 count= 62

575 count= 63

585 count= 64

595 count= 65

605 count= 66

615 count= 67