7.10 Exercises

1. Declare a register called *oscillate*. Initialize it to 0 and make it toggle every 30 time units. Do not use **`always`** statement (Hint: Use the **`forever`** loop).

```
reg oscillate;
initial
 begin
   oscillate = 1'b0;
   forever #30 oscillate = ~oscillate;
 end
```

2. Design a *clock* with time period = 40 and a duty cycle of 25% by using the **`always`** and **`initial`** statements. The value of *clock* at time = 0 should be initialized to 0.

```
module duty_cycle;
    reg clock;
    initial
       begin
          clock = 0;
       end
       always begin
          #1 clock = 0;
          #3 clock = 1;
       end
end module;
```

3. Given below is an **initial** block with blocking procedural assignments. At what simulation time is each statement executed? What are the intermediate and final values of *a*, *b*, *c*, *d*?

```
initial
begin
    a = 1'b0;
    b = #10 1'b1;
    c = #5 1'b0;
    d = #20 {a, b, c};
end
```

35 units

intermediate val: a=0

final => a=0, b=1, c=0 d={0,1,0}

4. Repeat exercise 3 if *nonblocking* procedural assignments were used.

intermediates: a=0 b=0 c=0 d={0, 0, 0}

final => a=0, b=1 c=0 d={a,b,c}

6. What is the final value of *d* in the following example. (Hint: See *intra-assignment delays*).

```
initial
begin
        b = 1'b1; c = 1'b0;
        #10 b = 1'b0;
initial
begin
        d = #25 (b | c);
end
```

#0 b=1 C=0

#10 b=0

#25 b=0, C=0

d=0

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
```

#0 x = 1'b0;
#5 y = 1'b1;
#20 b = y;
#25 a = x;
#45 x = 1'b1;
#50 m = y;
#55 p = x;
#55 a = y;
#85 b = y;

8.5 Exercises

1. Define a **function** to calculate the *factorial* of a 4-bit number. The output is a 32-bit value. Invoke the function by using stimulus and check results.

```
function automatic [0:31] factorial;
    input [0:3] Num;
    begin
      if( Num == 1)
          factorial = 1;
      else
          factorial = Num * factorial (Num-1);
    end
endfunction
```

2. Define a **function** to multiply two 4-bit numbers *a* and *b*. The output is an 8-bit value. Invoke the function by using stimulus and check results.

```
function [7:0] result;
    input [0:3] a;
    input [0:3] b;
  begin
      result = a*b;
  end
endmodule
```

4. Define a **task** to compute the factorial of a 4-bit number. The output is a 32-bit value. The result is assigned to the output after a delay of 10 time units.

```
task factorial;
  output [31:0] result;
  input [3:0] val;
  integer i;
initial
  begin result = 1'b1;
    for(i=0; i<val; i=i+1)
      result = result * val;
  end.
```

5. Define a **task** to compute even parity of a 16-bit number. The result is a 1-bit value that is assigned to the output after three positive edges of clock. (Hint: Use a **repeat** loop in the task).