Digital Logic Design - Chapter 7-Verilog

(In addition to the code include design documentation either as comments in the code or separate document.)

1S. Write a Verilog code that implements function f(A,B,C) = A'.B.C + A.B.C.

Solution:

```
`timescale 1 ns / 1 ps              //set time scale to 1 nano seconds (10^-9)

// Problem 1S.  f(A,B,C) = A'.B.C + A.B.C.
module prob6s (           // module definition
      input wire A,       // input is driven always so wire type is used
      input wire B,
      input wire C,
      output wire f       // this is a combinational logic which means output is
 );                       // continuously being drive so it can also be a wire.

      // assign is used for combinational logic
      assign  f = (~A)&B&C | A&B&C;       //Use logical or "|", and "&", not "~"

endmodule
```

1U. Write Verilog code to implements function f(A,B,C) = A'.B.C' + A'.B'.C + A.B'.

Solution:

2S. Write Verilog code to implement a positive-edge-triggered D latch.

Solution:

```
`timescale 1ns/100ps          // time measurement unit is 1 nsec with 100 ps percision

// Design an rising edge triggered  D flip flop
module D_ff(                  // defines the input and output into module
      input wire clock,
      input wire d,
      output reg q =0         // reg type remembers the value after each update
 );                           // reg type is used for sequential logic

// "always" block executes only during certain events
// In this case, always block executes at the rising edge of clock
always @ (posedge clock) begin
      q = d;                  // blocking assignment, LHS has to be reg type
end
endmodule
```

2U. Write Verilog code to implement a positive-edge-triggered T latch.

Solution:

3S. Write Verilog code to implement a negative-edge-triggered JK flip flop.

Solution:

```
`timescale 1ns/100ps    // time measurement unit is 1 nsec with 100 ps percision

// Design a JK flip flop
module JK_ff(                   // define module
    input wire clock,
    input wire j,
    input wire k,
    output reg q =0
 );

// Body of the design
always @ (negedge clock) begin      // executes code at falling edge of clock
    q <= j&(~q) | (~k)&q;           // make an assignment
end
endmodule
```

3U. Write Verilog code to implement a positive-edge-triggered SR flip flop.

Solution:

4S. Write Verilog code to implement a 4-bit binary up counter.

Solution:

```
`timescale 1ns/100ps    // time measurement unit is 1 nsec with 100 ps percision

// Design a 4-bit up counter
module Counter(                     // define module
    input wire clk,
    output reg [3:0] count =4'b0    // count is declared as a 4-bit array of reg type
 );

    // Body of the design
    always @ (posedge clk) begin   // executes code at rising edge of clock
        // Increment by one with roll over to zero
        count = (count==4'b1111) ? (4'b0) : (count + 4'b1);  // you can implement it using "if"
    end

endmodule
```

4U. Write Verilog code to implement a 4-bit binary down counter

Solution:

5S. Write Verilog code that outputs a signal that has a frequency equal to 1/2, 1/4, 1/8 & 1/16 of input clock frequency depending on the value of two input select bits.

Solution:

```
`timescale 1 ns / 1 ps

// Divide clock frequency  by 2, 4, 8
module freq_sel(
    input wire clk,
    input wire [1:0] sel,
    output reg out = 0
);

integer count=0;

always @ (posedge clk) begin
    count = count + 1;
    case (sel)
        0   : out = ~out;
        1   : if (count == 2) begin
                  out = ~out;
                  count = 0;
              end
        2   : if (count == 4) begin
                  out = ~out;
                  count = 0;
              end
        3   : if (count == 8) begin
                  out = ~out;
                  count = 0;
              end
        default: count = 0;
    endcase
end
endmodule
```

5U. Write Verilog code to implement a 3-bit binary to 8-line decoder (active high)

Solution:

6U. Write Verilog code to implement an 8-line (active low) to 3-bit binary encoder.

Solution:

7U. Write Verilog code that implements the 3-way T-intersection with each side having red, yellow and green lights. Assume that car on the right has priority and only one side has green light at any given time and yellow light is on for 2 to 4 seconds.

Solution:

8U. Write Verilog code to implement an ALU which adds, subtracts and multiplies 8-bit numbers.

Solution: