

2014 Paper E2.1: Digital Electronics II- Solutions

1. (a) This question tests students ability to write Verilog code for a simple counter.

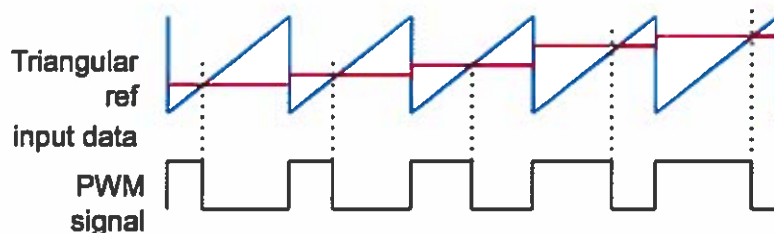
```
module counter_10_12 (clk, r, s, to);  
  
    input      clk;      // system clock  
    input      r;        // synchronous reset input  
    input      s;        // 0 - select 10-bit; 1 - select 12-bit counter  
    output     to;       // timeout signal - goes high for one clock cycle  
  
    reg [11:0] count;    // internal 12-bit counter  
  
    initial count = 12'b0;  
  
    always @ (posedge clk)  
    if (r == 1'b1)  
        count = 12'b0;  
    else begin  
        count <= count + 1'b1;  
        to <= 1'b0;    // timeout normally reset  
        if (s==1'b0) begin    // in 10-bit mode  
            if (count==12'd1023) begin  
                count <= 12'b0;  
                to <= 1'b1;  
            end  
        else    // in 12-bit mode  
            if (count==12'd4095)  
                to <= 1'b1;  
        end  
    end  
end    // always  
  
endmodule
```

This will need 12 LEs for the 12-bit counter. It also needs to detect 12'd1023 and 12'd4095. This requires 13-bit input (12 bits for counter and 1 bit for S), and two outputs. The minimum it would need is 5 LE's (for example, detect lower 10-bit as 1, this needs 4 LE's, and one more to detect the other 2 bits for 12-bit mode). However, anything between 5 and 10 would be acceptable. So accept an answer from 17 to 22 LEs.

[8]

- (b) Bookwork. This question tests student's understanding of PWM DAC which has been covered in the lectures.

Generate a triangular signal (in the form of a 12-bit counter) and compare the input value data_in to that of the counter value. Set pwm_out to be high if the counter value is lower than data_in, otherwise set it to low. The DAC output is the lowpass filtered version of the PWM signal.



```

module pwm_dac (clk, data_in, pwm_out);

    input          clk;          // system clock
    input [11:0]   data_in;      // input data for conversion
    output         pwm_out;       // PWM output

    reg [11:0]     count;         // internal 12-bit counter
    reg            pwm_out;

    initial count = 12'b0;

    always @ (posedge clk) begin
        count <= count + 1'b1;
        if (count > data_in)
            pwm_out <= 1'b0;
        else
            pwm_out <= 1'b1;
        end
    end

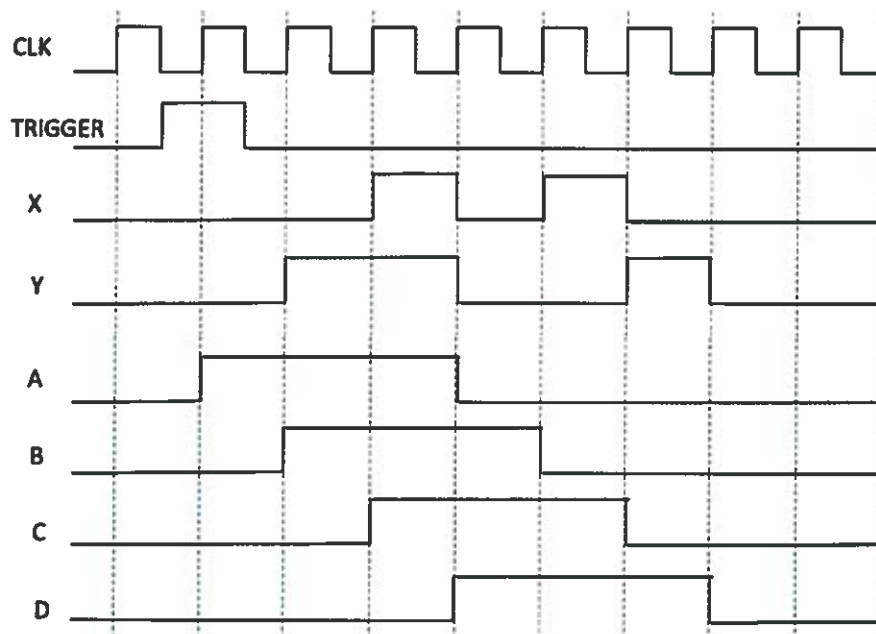
endmodule

```

[8]

- (c) This question tests student's ability to use shift registers to produce control signals for digital systems.

Use four stage shift register with outputs A, B, C and D as shown.

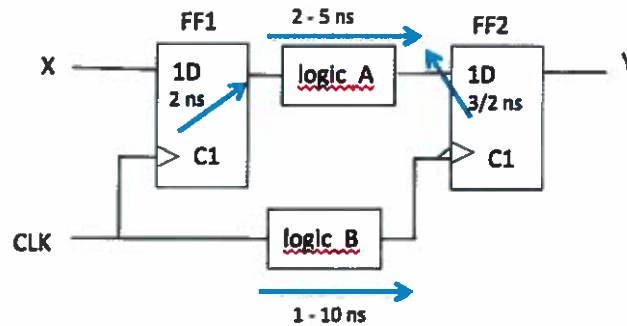


Then $X = A \& C + \sim B \& C$

$Y = A \& B + \sim C \& D$

[8]

(d) This question tests student's ability to work out digital circuit timing constraints.



Setup time constraint:

$$tc-q(\max) + \text{logic_A}(\max) + t_{\text{setup}} < \frac{1}{2} T + \text{logic_B}(\min)$$

$$2 + 5 + 3 < \frac{1}{2} T + 1, \text{ therefore } T > 18\text{ns and } F_{\max}(\text{setup}) < 55.56\text{MHz}$$

Hold time constraint:

$$T + tc-q(\min) + \text{logic_A}(\min) > \frac{1}{2} T + \text{logic_B}(\max) + t_{\text{hold}}, \text{ therefore } T > 16\text{ns.}$$

Hold time is never violated.

[8]

(e) This question tests student's understanding of memory map and address decoding circuits.

The address ranges for the four spaces are:

RAM_1: 18'h00000 to 18'h07FFF

RAM_2: 18'h08000 to 18'h0BFFF

ROM_1: 18'h10000 to 18'h1FFFF

I/O: 18'h3FF80 to 18'h3FF9F

Therefore RAM_1 CS = $\sim A_{17} \& \sim A_{16} \& \sim A_{15}$

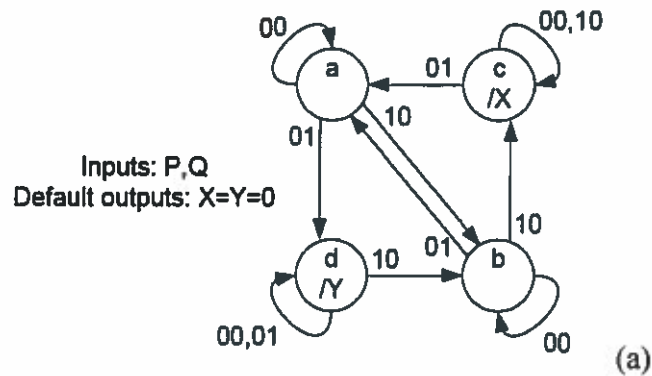
RAM_2 CS = $\sim A_{17} \& \sim A_{16} \& A_{15} \& \sim A_{14}$

ROM_1 CS = $\sim A_{17} \& A_{16}$

I/O CS = $A_{17} \& A_{16} \& A_{15} \& A_{14} \& A_{13} \& A_{12} \& A_{11} \& A_9 \& A_8$
 $\& A_7 \& A_6 \& A_5$

[8]

2. This question tests student's ability to design a reasonably complicated FSM.



[15]

(b)

```

module FSM_detector (CLK, P, Q, X, Y);
    input  CLK;      // clock input - all transitions happens shortly after rising edge
    input  P,Q;      // FSM input signals
    output X,Y;      // FSM output signals

    parameter STATE_a = 4'b0001, STATE_b = 4'b0010, STATE_c = 4'b0100, STATE_d = 4'b1000;
    reg [3:0] state;

    initial begin
        state = STATE_a;
        X = 1'b0;
        Y = 1'b0;
    end

    wire [1:0] in;

    assign in = {P,Q};

    always @ (posedge CLK)
        case (state)
            STATE_a: case (in)
                2'b00: state <= STATE_a;
                2'b01: state <= STATE_d;
                2'b10: state <= STATE_b;
                default: state <= STATE_a;
            endcase
            STATE_b: case (in)
                2'b00: state <= STATE_b;
                2'b01: state <= STATE_a;
                2'b10: state <= STATE_c;
                default: state <= STATE_b;
            endcase
            STATE_c: case (in)
                2'b00: state <= STATE_c;
                2'b01: state <= STATE_a;
                2'b10: state <= STATE_c;
                default: state <= STATE_c;
            endcase
            STATE_d: case (in)
                2'b00: state <= STATE_d;
                2'b01: state <= STATE_d;
                2'b10: state <= STATE_b;
                default: state <= STATE_d;
            endcase
            default: ;
        endcase

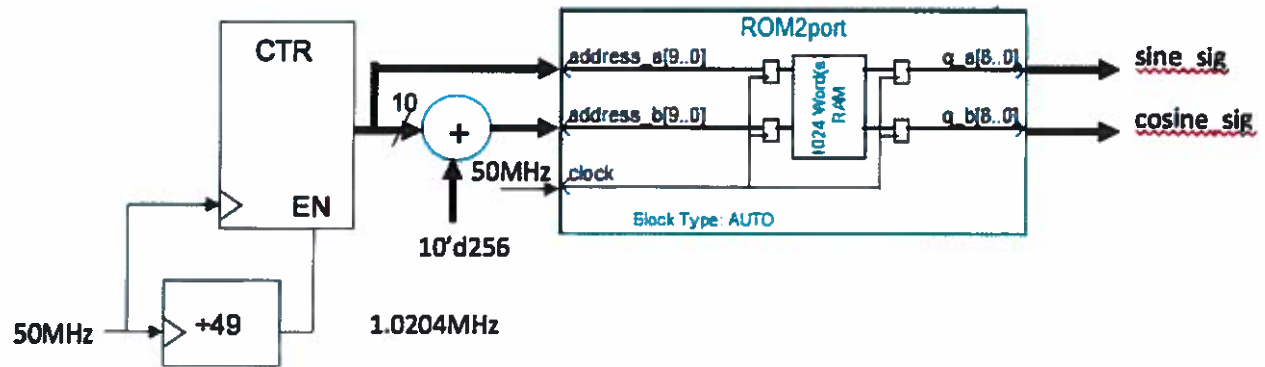
        X = (state == STATE_c);
        Y = (state == STATE_d);
    endmodule
  
```

```

always @ (state)
    case (state)
        STATE_a: begin X = 0; Y = 0; end
        STATE_b: begin X = 1; Y = 0; end
        STATE_c: begin X = 0; Y = 0; end
        STATE_d: begin X = 0; Y = 1; end
        default: begin X = 0; Y = 0; end
    endcase
  
```

[15]

3. (a)



[15]

(b)

```

module quadrature_gen (CLK50, sine_sig, cosine_sig);

    input          CLK50;          // 50 MHz clock
    output [8:0]    sine_sig;       // sinewave signal
    output [8:0]    cosine_sig;     // cosinewave signal

    // ---- clock divider
    reg [5:0]       clk_ctr;        // count clock cycles
    reg            time_out;        // goes high for 1 cycle every 49 clk cycles
    initial         clk_ctr = 6'b0;
    parameter      tc = 48;        // count from 0 to 48
    always @ (posedge CLK50)
    |   if (clk_ctr==0) begin
        time_out <= 1'b1;
        clk_ctr <= tc;
        end
    |   else begin
        time_out <= 1'b0;
        clk_ctr <= clk_ctr + 1'b1;
        end
    // ---- end of clock divider

    // ---- address counter ----

    reg [9:0]       address_a;      // address counter for sine_sig
    reg [9:0]       address_b;      // address counter for cosine_sig
    initial         address_a = 10'b0;

    always @ (posedge CLK50)
    |   if (time_out == 1'b1) begin
        address_a <= address_a + 1'b1;
        address_b <= address_a + 10'd246;
        end
    end

    ROM2port sine_rom (address_a, address_b, CLK50, sine_sig, cosines_sig);

endmodule

```

$$f_{out} = f_{samp}/1024 = (50\text{MHz}/49) / 1024 = 996.5 \text{ Hz.}$$

[15]