2019 Paper E2.1: Digital Electronics II

Answer ALL questions.

There are THREE questions on the paper.

Question ONE counts for 50% of the marks, other questions 25% each

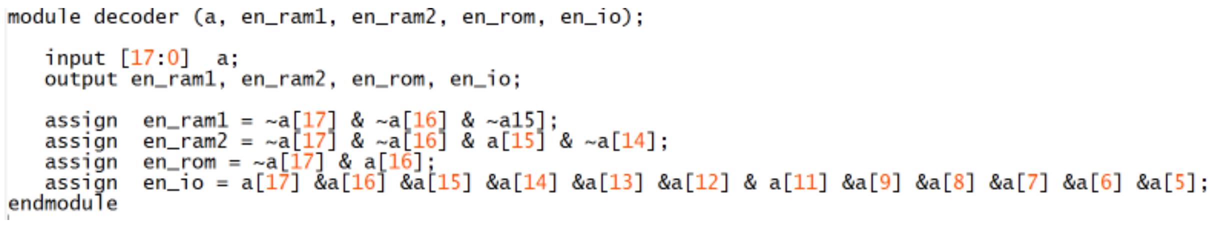
Time allowed: 2 hours

SOLUTIONS

(For Examiners Only)

1. (a) This question tests student's ability to specify simple combinational circuit in Verilog, and their understanding of memory map and address decoding circuits.

(i)



```
module decoder (a, en_ram1, en_ram2, en_rom, en_io);

   input [17:0]  a;
   output en_ram1, en_ram2, en_rom, en_io;

   assign  en_ram1 = ~a[17] & ~a[16] & ~a15];
   assign  en_ram2 = ~a[17] & ~a[16] & a[15] & ~a[14];
   assign  en_rom = ~a[17] & a[16];
   assign  en_io = a[17] &a[16] &a[15] &a[14] &a[13] &a[12] & a[11] &a[9] &a[8] &a[7] &a[6] &a[5];
endmodule
```

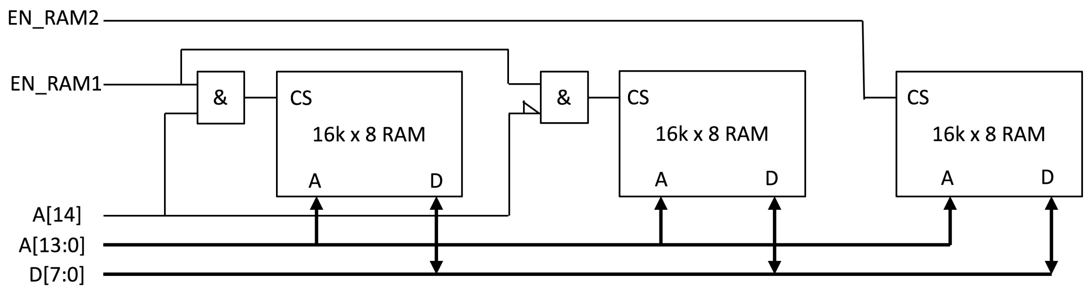
[2]

(ii) 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'h3FFE0	to	18'h3FFFF

[4]

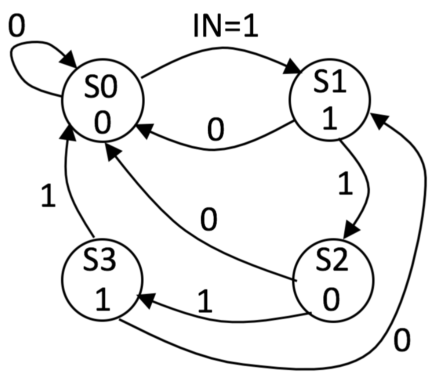
(iii)



[4]

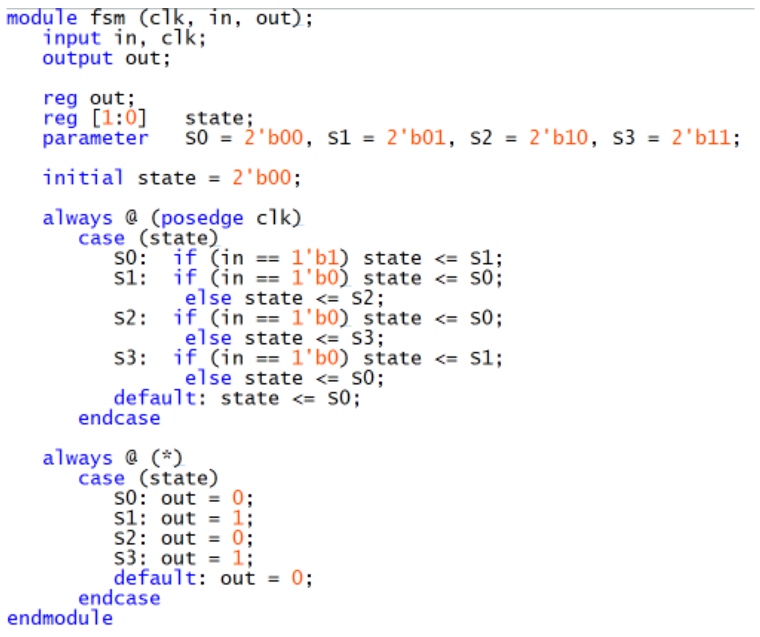
(b) This question tests students understanding of the use of ROM to implement FSM, and how to specify a FSM in terms of state diagram and Verilog HDL.

(i)



[5]

(ii)

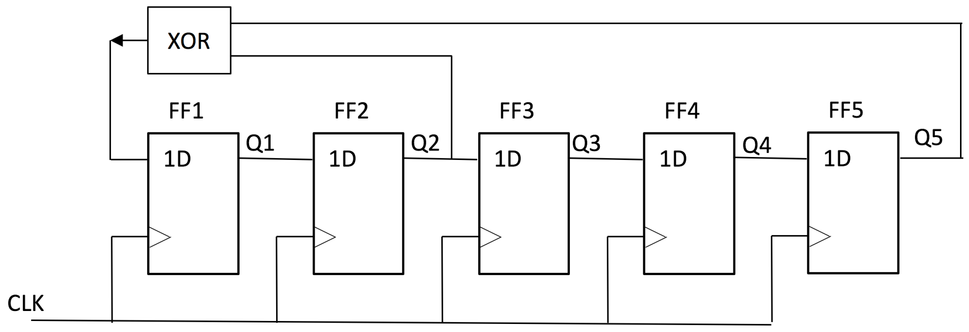


```
module fsm (clk, in, out);
   input in, clk;
   output out;

   reg out;
   reg [1:0]   state;
   parameter   S0 = 2'b00, S1 = 2'b01, S2 = 2'b10, S3 = 2'b11;

   initial state = 2'b00;

   always @ (posedge clk)
      case (state)
         S0:  if (in == 1'b1) state <= S1;
         S1:  if (in == 1'b0) state <= S0;
               else state <= S2;
         S2:  if (in == 1'b0) state <= S0;
               else state <= S3;
         S3:  if (in == 1'b0) state <= S1;
               else state <= S0;
         default: state <= S0;
      endcase

   always @ (*)
      case (state)
         S0: out = 0;
         S1: out = 1;
         S2: out = 0;
         S3: out = 1;
         default: out = 0;
      endcase
endmodule
```

[5]

(c) This question tests student's ability to read Verilog code and their understanding of using LFSR to implement a pseudo-random binary sequence generator circuit.

(i)



[4]

(ii)

Q5:Q1	Q5^Q2
0 0 0 0 1	0
0 0 0 1 0	1
0 0 1 0 1	0
0 1 0 1 0	1
1 0 1 0 1	1
0 1 0 1 1	1
1 0 1 1 1	0

[4]

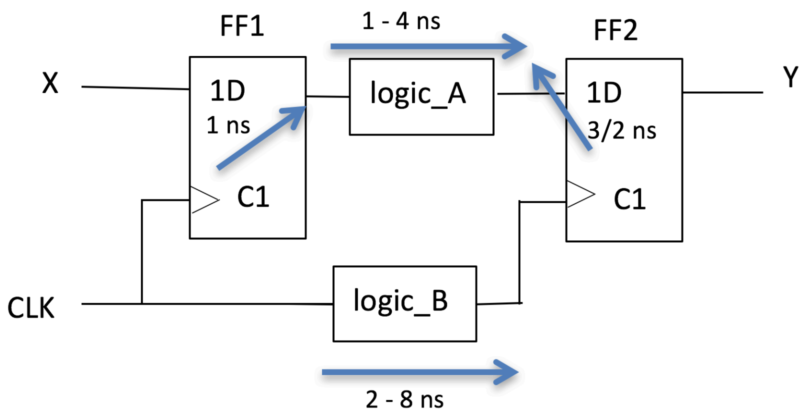
(iii) The primitive polynomial implemented is:

$$1 + x^2 + x^5$$

[2]

(d) This tests student's understanding of timing constraints in digital circuits.

(i)



Setup time constraint:

tc-q(max) + logic_A(max) + t_setup < T + logic_B(min)

[4]

(ii)

Given the various timing specification, we get:

$1 + 4 + 3 \leq T + 2$, therefore $T \geq 6$ns and Fmax (setup) $\leq$ 166.7 MHz

[2]

(iii)

Hold time constraint:

T + tc-q(min) + logic_A(min) $\geq$ logic_B(max) + t_hold

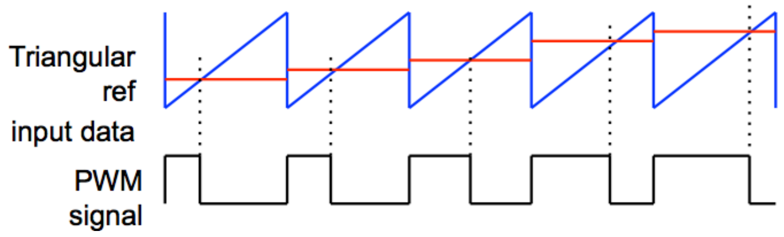
$6 + 1 + 1 \geq 8 + 2$, which does not hold! Therefore there can be hold time violation

[4]

(e) This tests student's understanding of PWM DAC principles and implementation.

(i) 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.



[5]

(ii)

```
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

endmodule
```

[5]

2. This question tests student's ability to apply what they have learned in the Lab to a practical problem in sine and cosine generation.

   (a) Dual-port ROM consists of two independent ports, with separate address and data buses. In this way, the contents of the ROM can be assessed by two sources at two addresses simultaneously. Since the ROM contains 1024 samples of a complete sinewave, we can produce a sinewave by sequentially reading the contents of the ROM, increasing the address value using a 10-bit counter at each counter clock cycle. The quadrature signal (i.e. the sinewave that is 90 degrees apart) can be produce by offsetting the address value by ¼ of a cycle or 256 location.

   [10]

   (b) The frequency of the sine and cosine signals is given by:

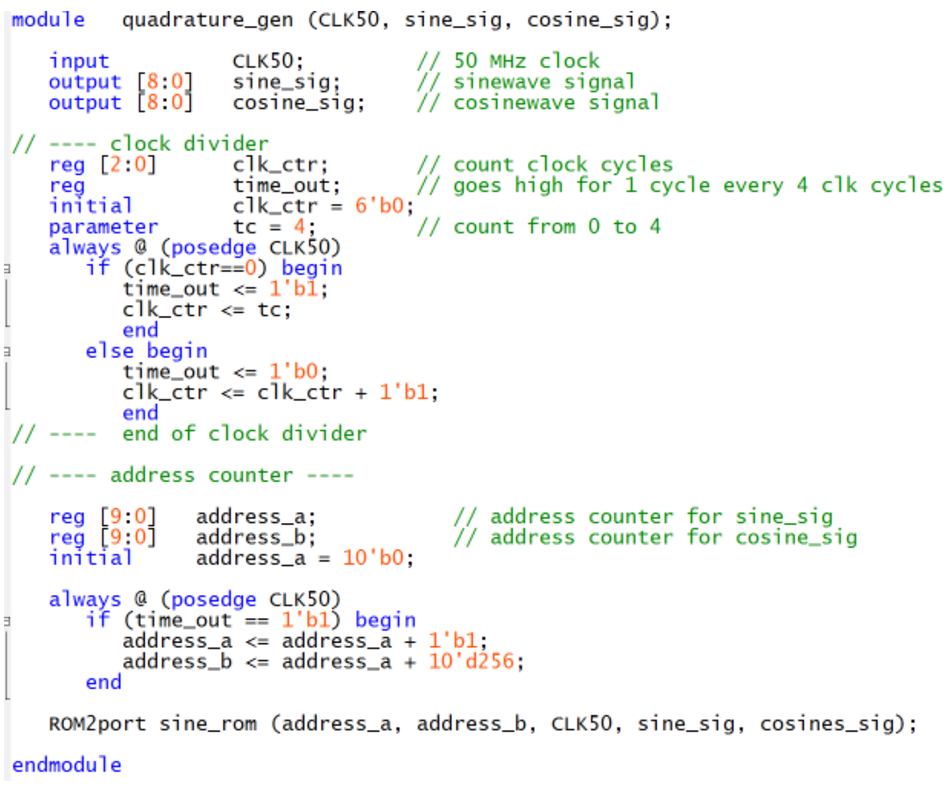
   fsig = fsamp/1024 = (50MHz/K) / 1024.

   For fsig to be as close to 10kHz as possible, we need fsamp to be 10.24MHz. Therefore the closest we can get to is to divide the 50MHz clock by 5.

   Hence K = 5 and N = 256.

   [5]

   (c)

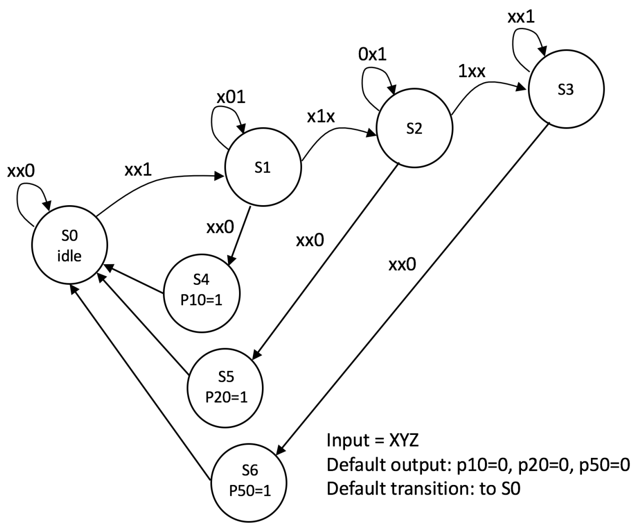


```
module   quadrature_gen (CLK50, sine_sig, cosine_sig);

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

// ---- clock divider
   reg [2:0]      clk_ctr;       // count clock cycles
   reg            time_out;      // goes high for 1 cycle every 4 clk cycles
   initial        clk_ctr = 6'b0;
   parameter      tc = 4;        // count from 0 to 4
   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'd256;
      end

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

endmodule
```

   [10]

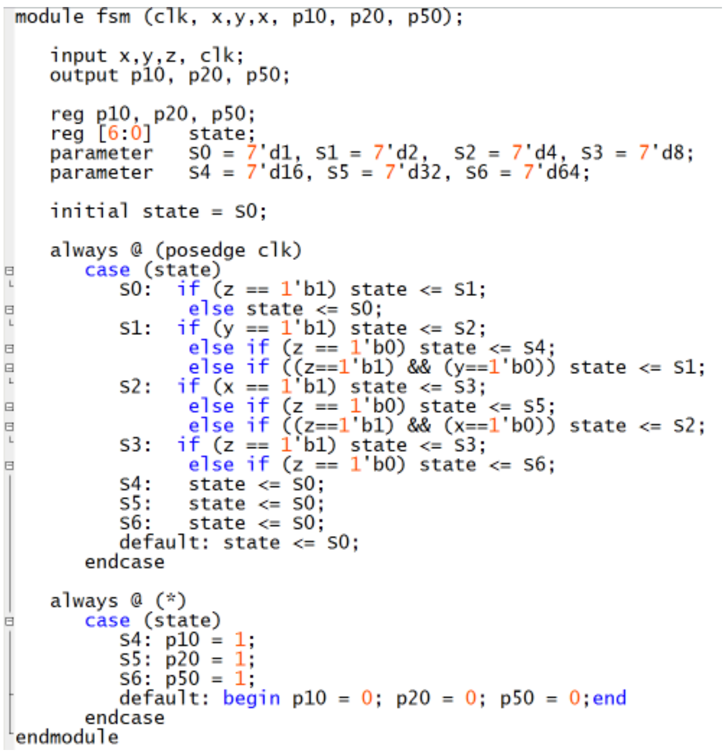
3. This question tests student’s ability to design and code in Verilog HDL a reasonably complex FSM with a practical application.

(a)



[15]

(b)



```
module fsm (clk, x,y,x, p10, p20, p50);

   input x,y,z, clk;
   output p10, p20, p50;

   reg p10, p20, p50;
   reg [6:0]   state;
   parameter   S0 = 7'd1, S1 = 7'd2,  S2 = 7'd4, S3 = 7'd8;
   parameter   S4 = 7'd16, S5 = 7'd32, S6 = 7'd64;

   initial state = S0;

   always @ (posedge clk)
      case (state)
         S0:  if (z == 1'b1) state <= S1;
              else state <= S0;
         S1:  if (y == 1'b1) state <= S2;
              else if (z == 1'b0) state <= S4;
              else if ((z==1'b1) && (y==1'b0)) state <= S1;
         S2:  if (x == 1'b1) state <= S3;
              else if (z == 1'b0) state <= S5;
              else if ((z==1'b1) && (x==1'b0)) state <= S2;
         S3:  if (z == 1'b1) state <= S3;
              else if (z == 1'b0) state <= S6;
         S4:  state <= S0;
         S5:  state <= S0;
         S6:  state <= S0;
         default: state <= S0;
      endcase

   always @ (*)
      case (state)
         S4: p10 = 1;
         S5: p20 = 1;
         S6: p50 = 1;
         default: begin p10 = 0; p20 = 0; p50 = 0;end
      endcase
endmodule
```

[10]