Imperial College
London

Revision Lecture 1
(Remote delivery)

28 April 2020

Peter Cheung
Department of Electrical & Electronic Engineering
Imperial College London

URL: www.ee.imperial.ac.uk/pcheung/
E-mail: p.cheung@imperial.ac.uk

2019 Q1 (a) Address decoding (1)

An 8-bit microprocessor system with an 18-bit memory address bus A[17:0] is interfaced to two banks of RAM (RAM1 and RAM2), one bank of ROM, and a space for input and output (IO) as shown in *Figure 1.1a*. The control signals from the microprocessor are omitted for clarity.

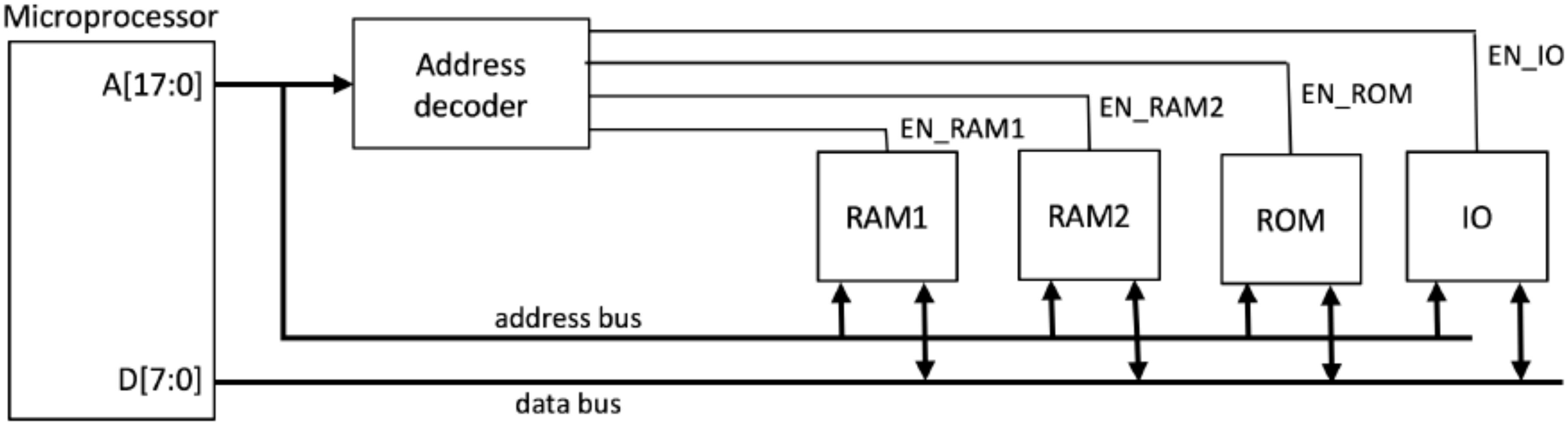
The address decoder module generates four enable signals for the RAM, ROM and IO: EN_RAM1, EN_RAM2, EN_ROM and EN_IO, implementing the following Boolean equations:

EN_RAM1 = ~A17 & ~A16 & ~A15

EN_RAM2 = ~A17 & ~A16 & A15 &~A14

EN_ROM = ~A17 & A16

EN_IO = A17 & A16 & A15 & A14 & A13 & A12 & A11 & A10 & A9 &



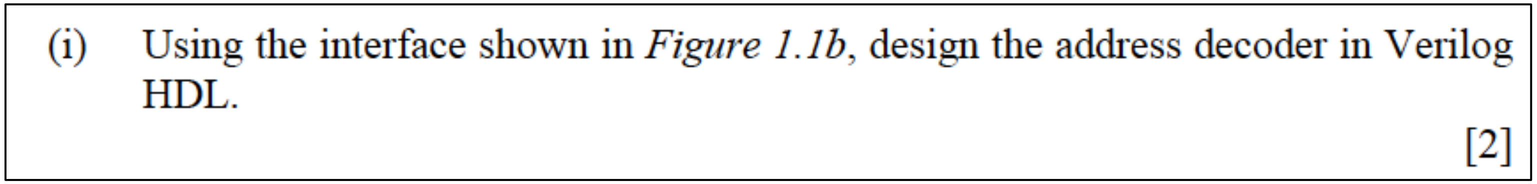
2019 Q1 (a) Address decoding (2)

EN_RAM1 = ~A17 & ~A16 & ~A15

EN_RAM2 = ~A17 & ~A16 & A15 &~A14

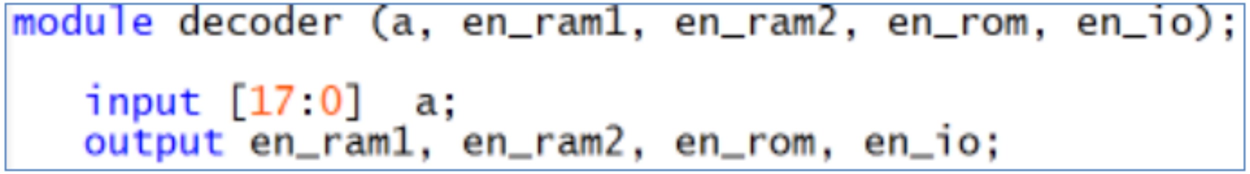
EN_ROM = ~A17 & A16

EN_IO = A17 & A16 & A15 & A14 & A13 & A12 & A11 & A10 & A9 &
A8 & A7 & A6 &A5



(i) Using the interface shown in *Figure 1.1b*, design the address decoder in Verilog HDL.

[2]



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

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

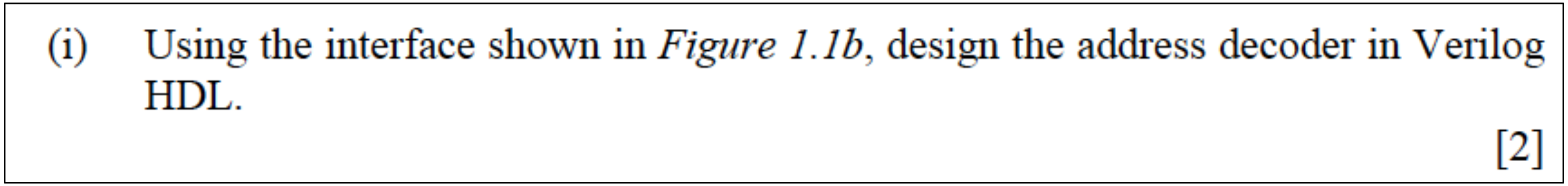
2019 Q1 (a) Address decoding – Solution (i)

EN_RAM1 = ~A17 & ~A16 & ~A15

EN_RAM2 = ~A17 & ~A16 & A15 &~A14

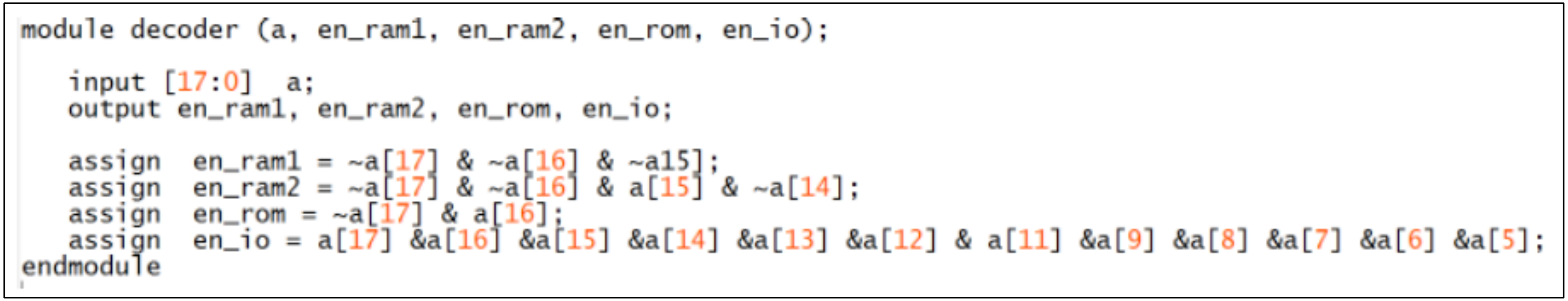
EN_ROM = ~A17 & A16

EN_IO = A17 & A16 & A15 & A14 & A13 & A12 & A11 & A10 & A9 &
A8 & A7 & A6 &A5



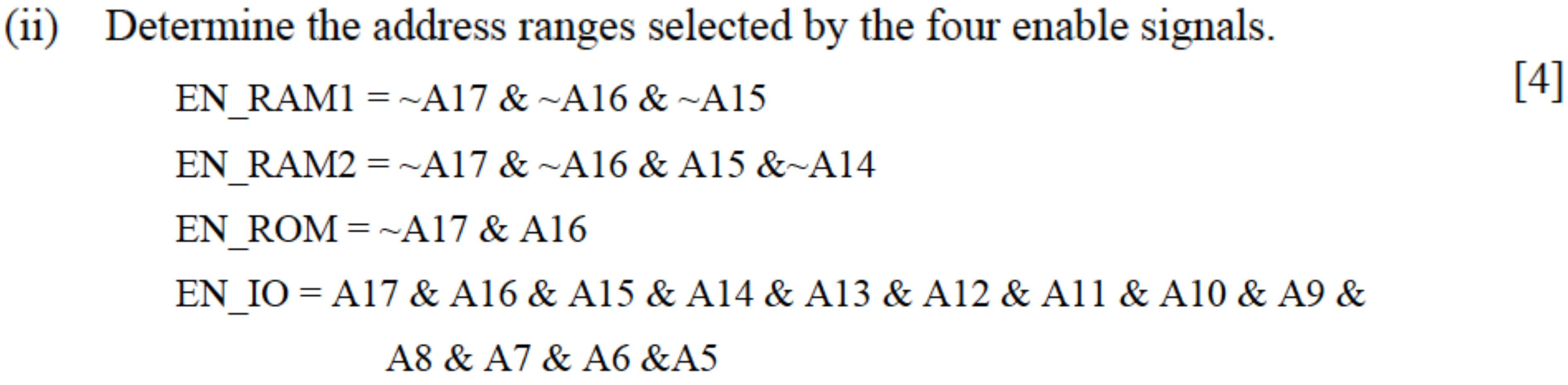
(i) Using the interface shown in *Figure 1.1b*, design the address decoder in Verilog HDL.

[2]



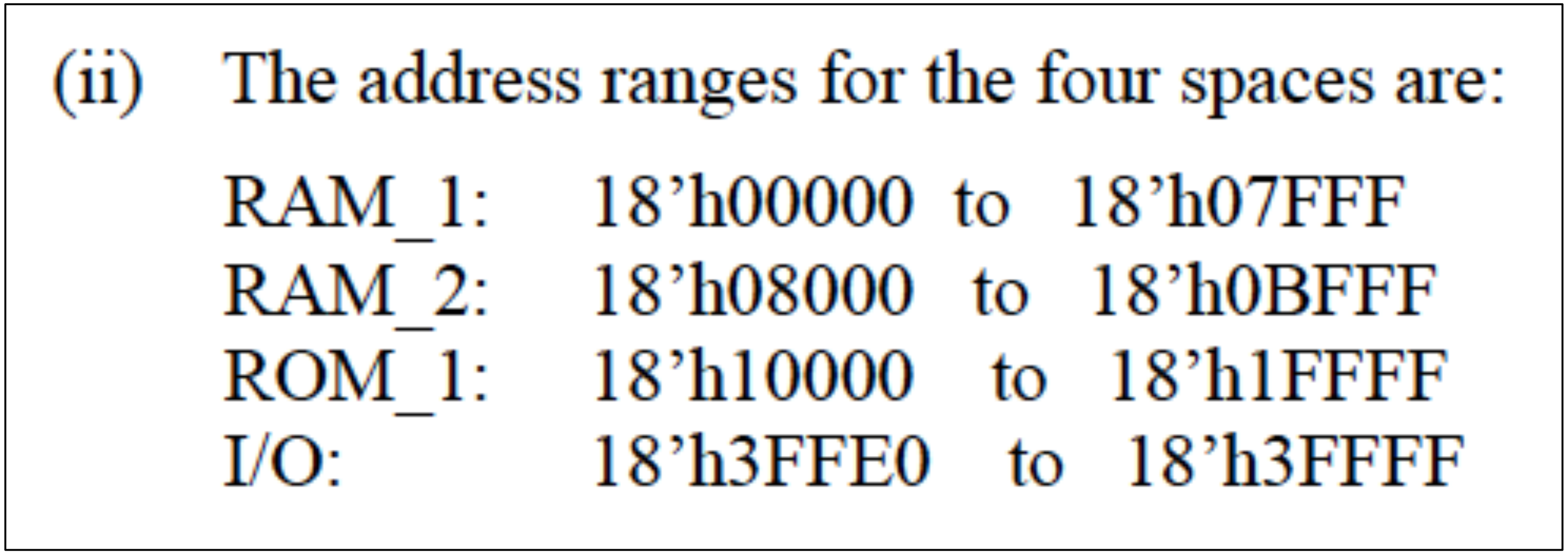
```
module decoder (a, en_ram1, en_ram2, en_rom, en_io);

   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
```

2019 Q1 (a) Address decoding – Solution (ii)



(ii) Determine the address ranges selected by the four enable signals. [4]

EN_RAM1 = ~A17 & ~A16 & ~A15
EN_RAM2 = ~A17 & ~A16 & A15 &~A14
EN_ROM = ~A17 & A16
EN_IO = A17 & A16 & A15 & A14 & A13 & A12 & A11 & A10 & A9 &
A8 & A7 & A6 &A5



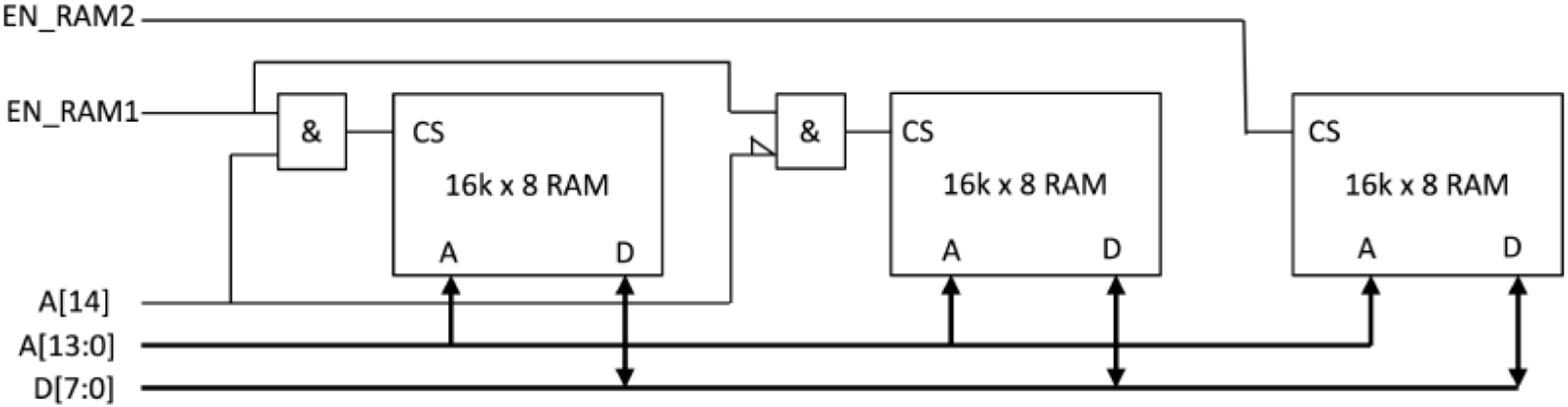
(ii) The address ranges for the four spaces are:

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

2019 Q1 (a) Address decoding – Solution (iii)

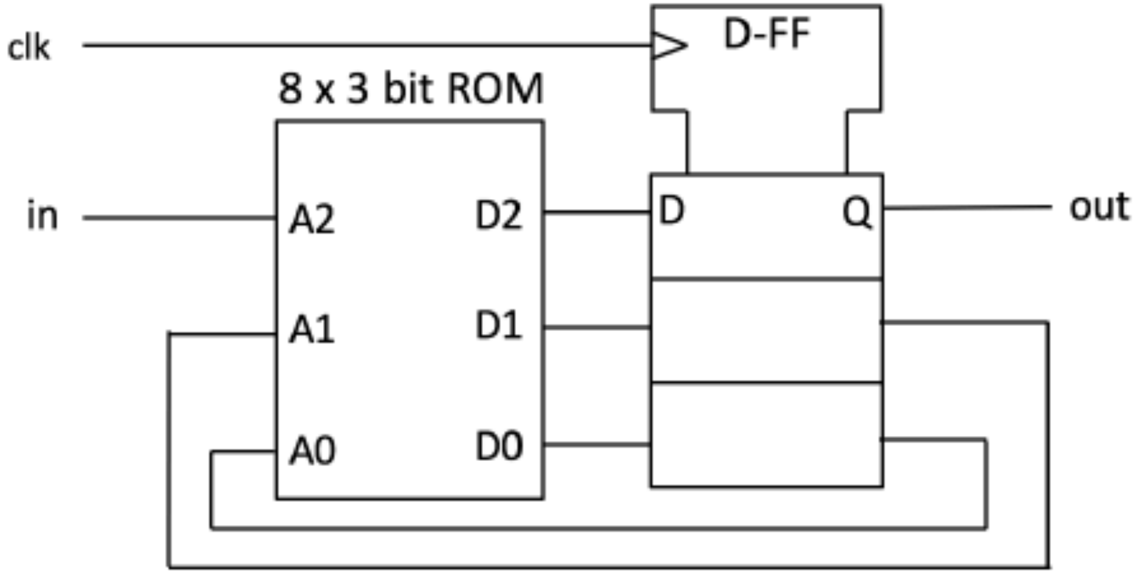
(iii) The two blocks of RAM are to be implemented using only 16k x 8 RAM chips. Draw the circuit diagram for RAM1 and RAM2 showing how the address bus, the data bus, and the enable signals EN_RAM1 and EN_RAM2 are connected to the RAM chips.

RAM_1: 18'h00000 to 18'h07FFF
RAM_2: 18'h08000 to 18'h0BFFF



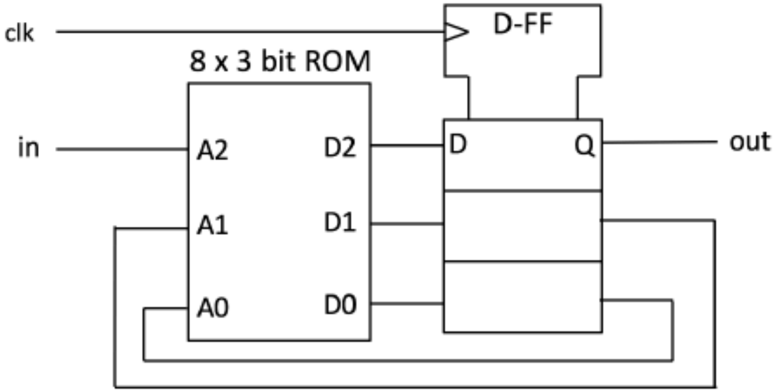
2019 Q1 (b) State Machine (1)

Figure 1.2 shows the circuit that implements a finite state machine (FSM) which has four states, one input ***in***, and one output ***out***. The circuit consists of an 8 x 3-bit ROM and three D-flipflops clocked by a common clock signal ***clk***. The contents of the ROM are shown in the table in *Figure 1.2*. The four states are encoded in binary code as S0, S1, S2 and S3. The D-FFs are initially cleared on power up.



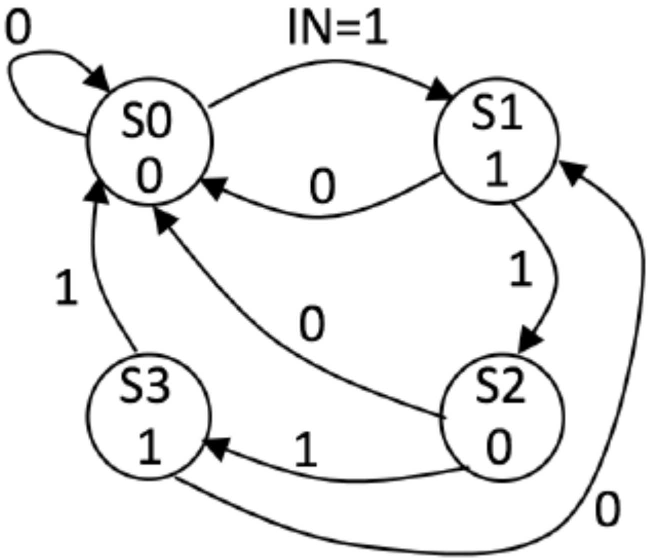
Address A[2:0]	Data D[2:0]
000	000
001	000
010	000
011	101
100	101
101	010
110	111
111	000

2019 Q1 (b) State Machine - Solution (i)



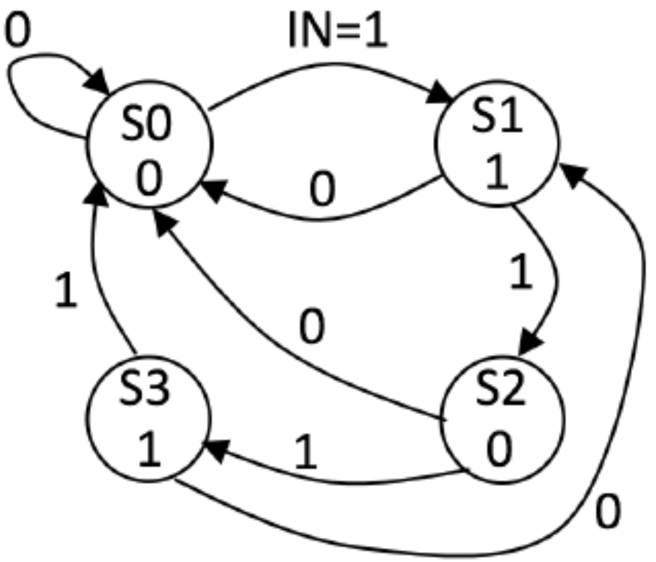
Address A[2:0]	Data D[2:0]
000	000
001	000
010	000
011	101
100	101
101	010
110	111
111	000

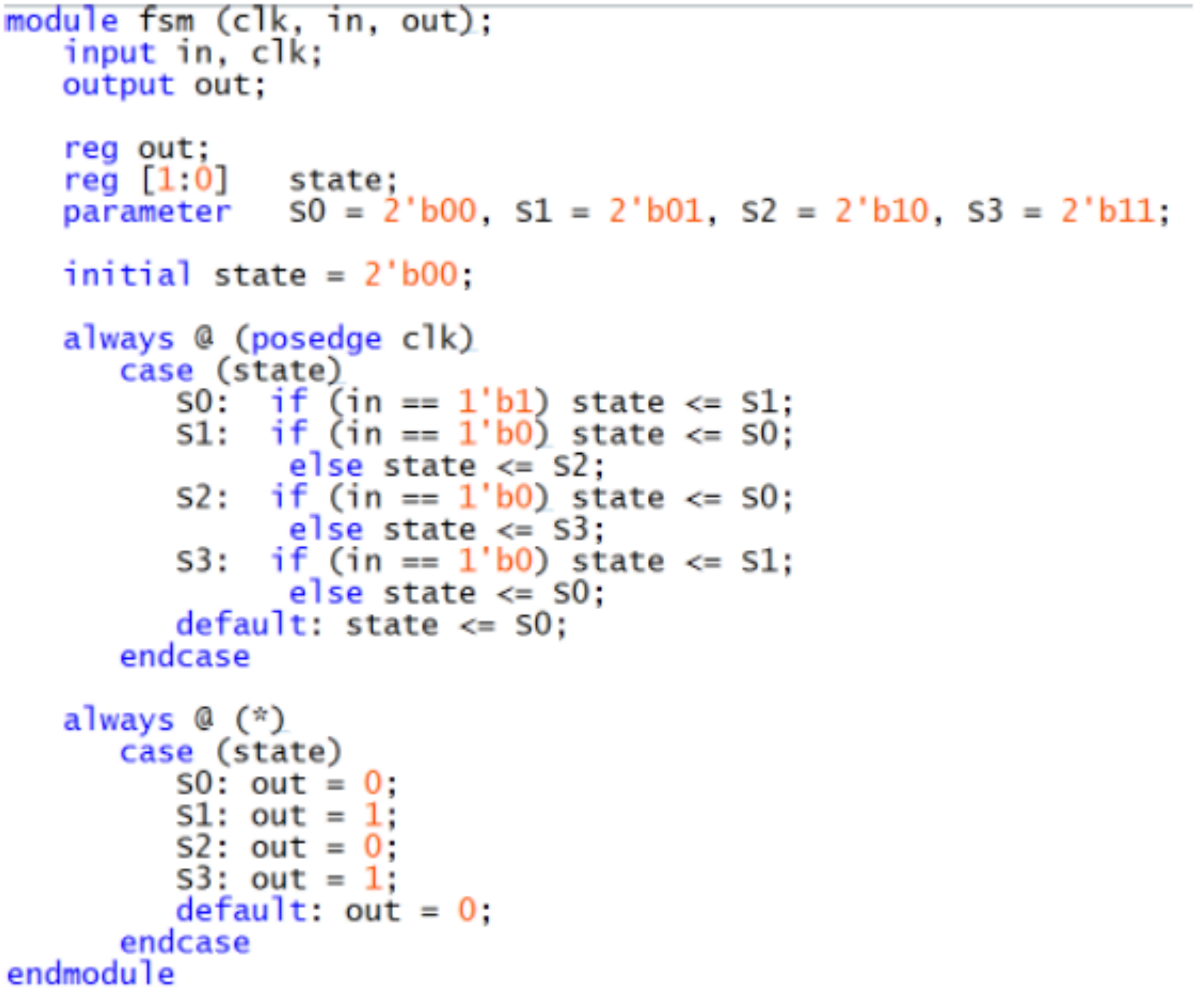
(i) Derive the state diagram for the FSM.



2019 Q1 (b) State Machine - Solution (ii)

(ii) Starting with the state diagram in (i), design a new version of the FSM in Verilog HDL.

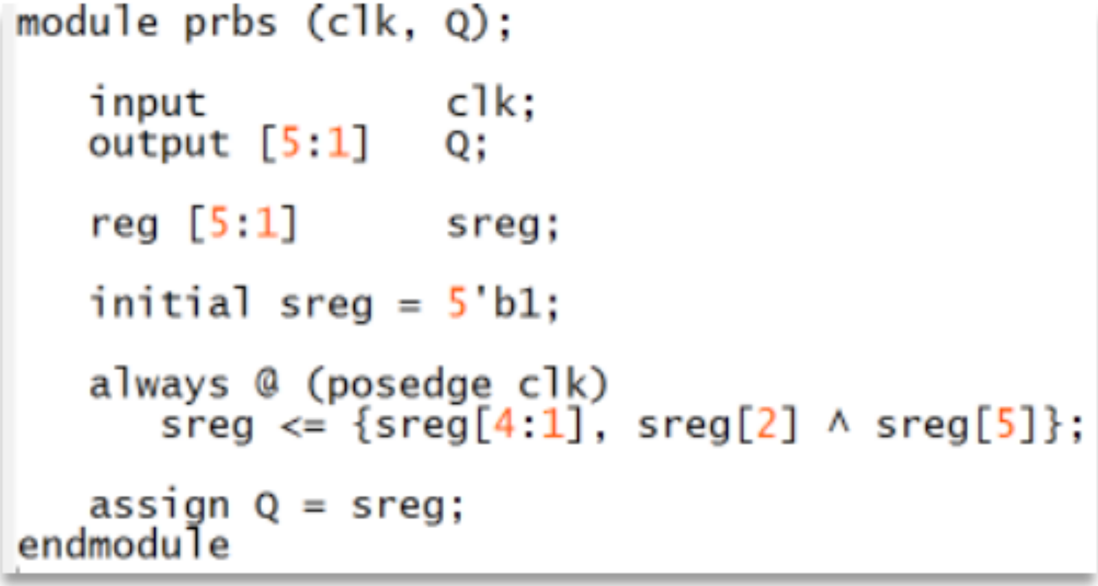




```
module fsm (clk, in, out);
   input in, clk;
   output out;

   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
```

2019 Q1 (c) Linear Feedback Shift Register (1)

Figure 1.3 shows the Verilog HDL implementation of a 5-bit pseudo-random binary sequence (PRBS) generator circuit.



```
module prbs (clk, Q);

   input           clk;
   output [5:1]    Q;

   reg [5:1]       sreg;

   initial sreg = 5'b1;

   always @ (posedge clk)
      sreg <= {sreg[4:1], sreg[2] ^ sreg[5]};

   assign Q = sreg;
endmodule
```

(i) Draw the circuit schematic diagram for the PRBS generator.

[4]

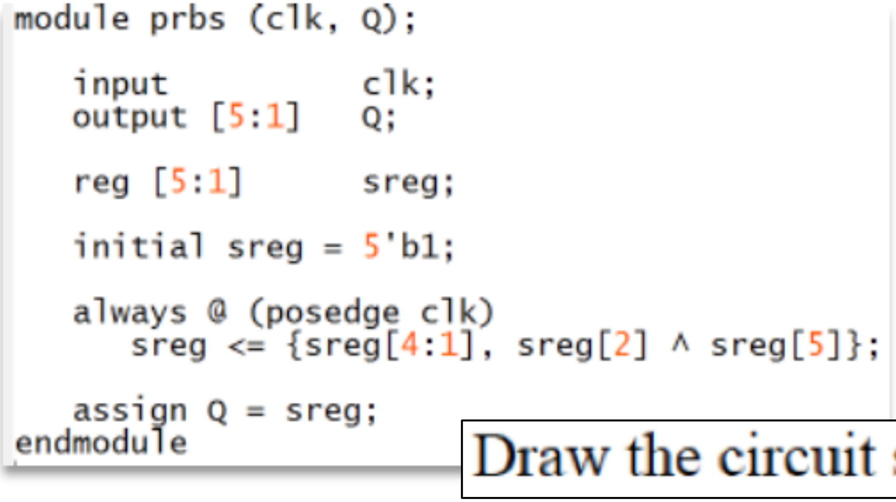
(ii) Determine the output value Q5:1 for the first 6 clock cycles.

[4]

(iii) What is the primitive polynomial of this PRBS generator?

[2]

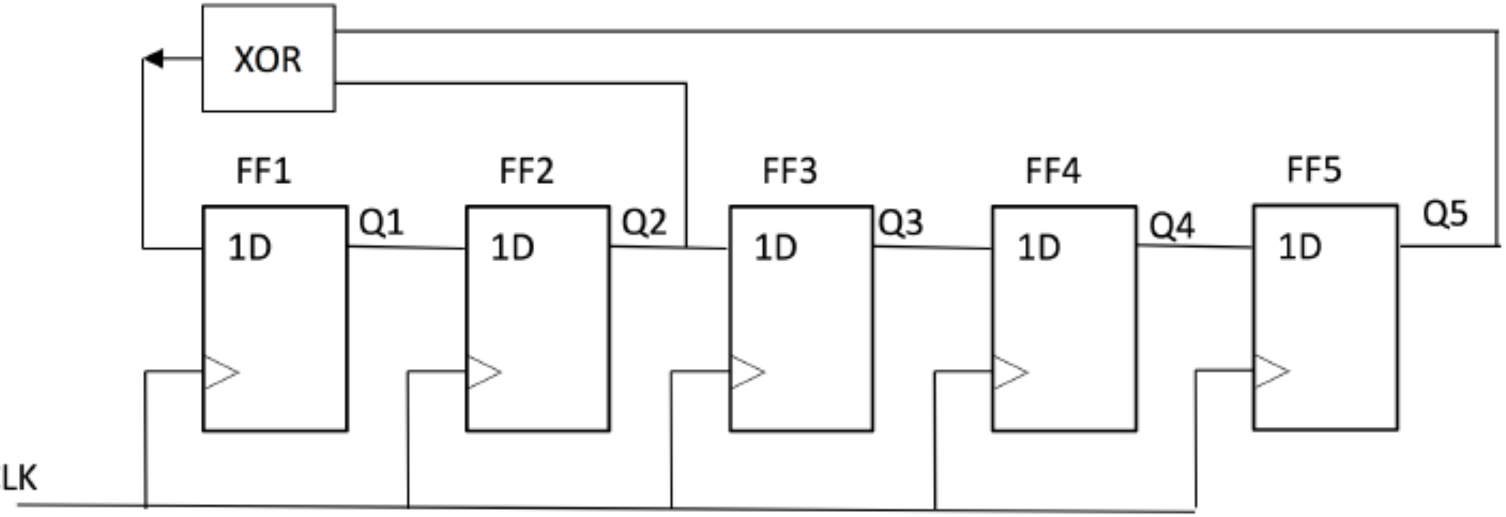
2019 Q1 (c) LFSR – solution (i)



```
module prbs (clk, Q);

   input          clk;
   output [5:1]   Q;

   reg [5:1]      sreg;

   initial sreg = 5'b1;

   always @ (posedge clk)
      sreg <= {sreg[4:1], sreg[2] ^ sreg[5]};

   assign Q = sreg;
endmodule
```

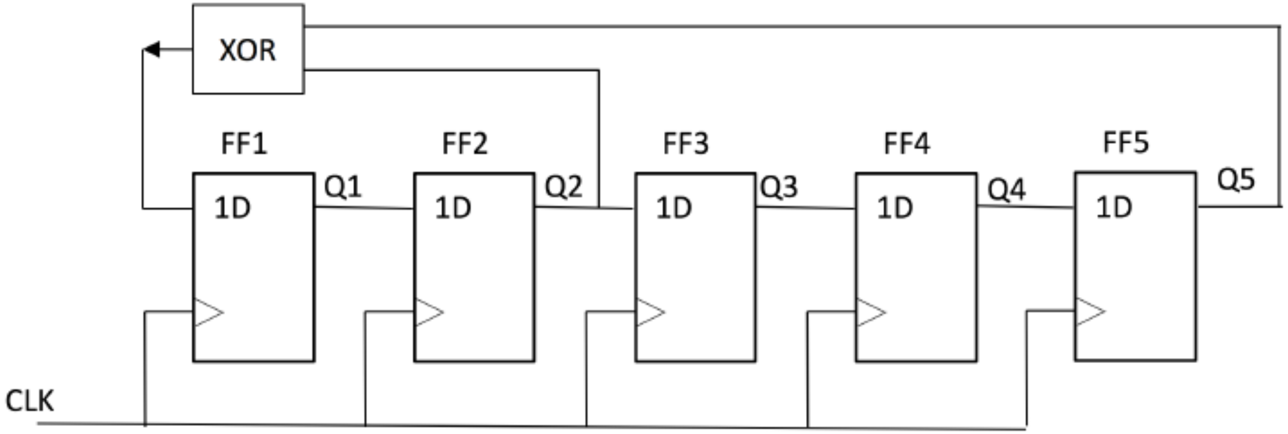
Draw the circuit schematic diagram for the PRBS generator.

(i)



2019 Q1 (c) LFSR – solution (ii & iii)

(ii) Determine the output value Q5:1 for the first 6 clock cycles.



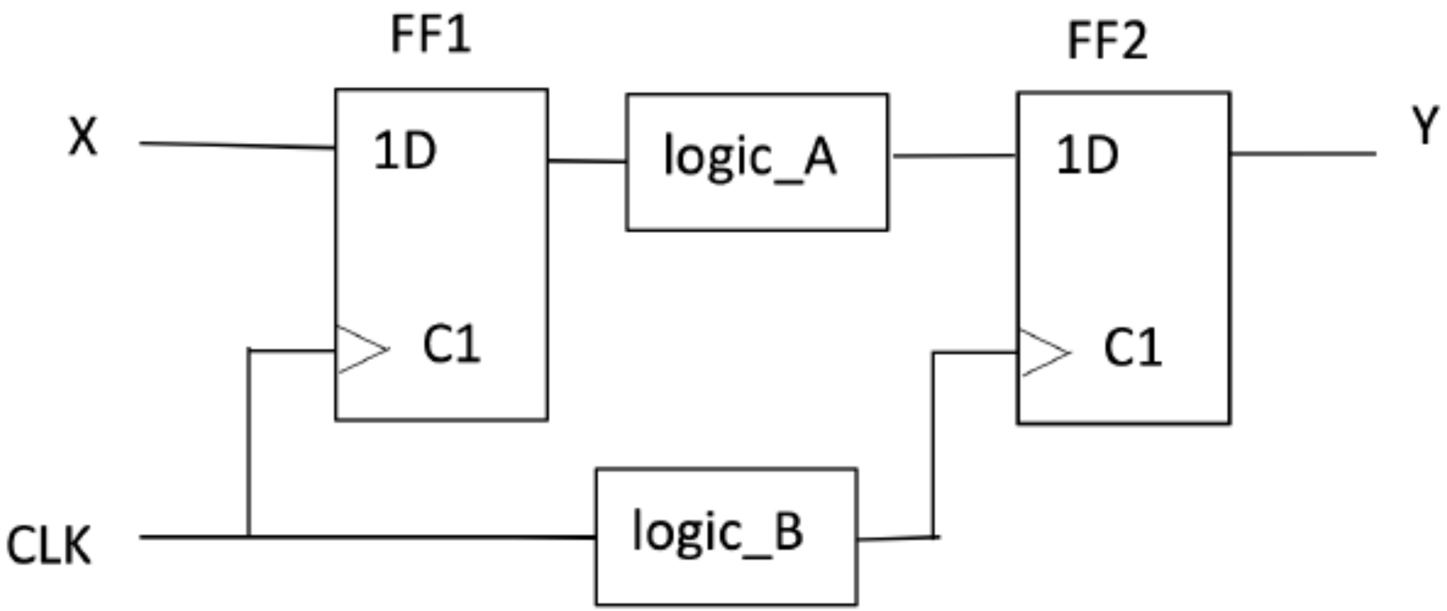
Q5:Q1	Q5^Q2
0 0 0 0 1	0
0 0 0 1 0	1
0 0 1 0 1	0
0 1 0 1 0	1
1 0 1 0 1	1
0 1 0 1 1	1
1 0 1 1 1	0

The primitive polynomial implemented is:

$$\boxed{1 + x^2 + x^5}$$

2019 Q1 (d) Digital Timing

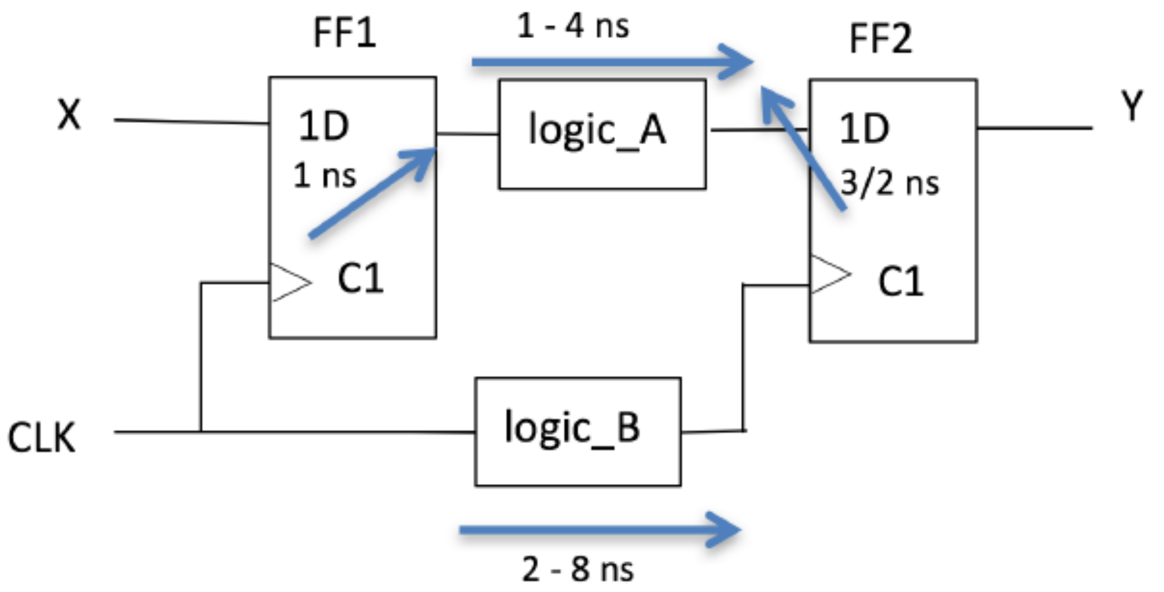
Figure 1.4 shows a circuit with two D-flipflops FF1 and FF2 with setup and hold times of 3 ns and 2 ns respectively, and a clock-to-Q output delay of 1 ns. The clock signal CLK has a 1:1 mark-space ratio. The D input of FF2 is driven by logic_A, which has a propagation delay between 1 ns and 4 ns. The clock input to FF2 is driven by logic_B, which has a propagation delay between 2 ns and 8 ns.



2019 Q1 (d) Digital Timing – solution (i)

(i) Derive the setup time constraint for D input of FF2 as an inequality.

[4]



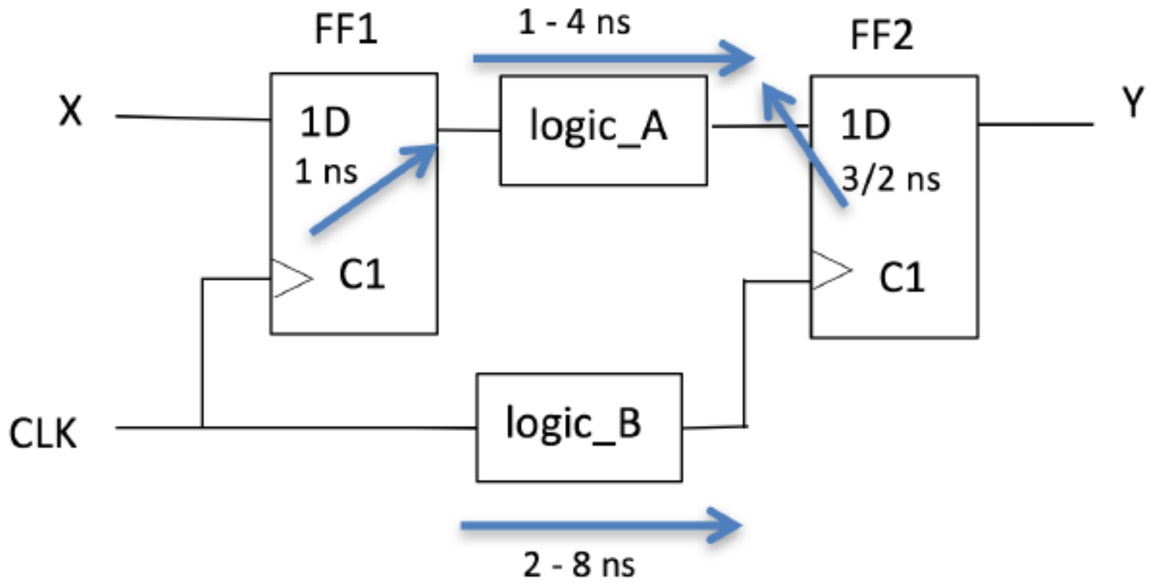
Setup time constraint:

$$tc\text{-}q(\max) + logic\_A(\max) + t\_setup < T + logic\_B(\min)$$

2019 Q1 (d) Digital Timing – solution (ii)

(ii) By considering the setup time constraint only, derive the maximum operating frequency f_{max} of CLK.

[2]



$$t_{c\text{-}q}(max) + logic\_A(max) + t\_setup < T + logic\_B(min)$$

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

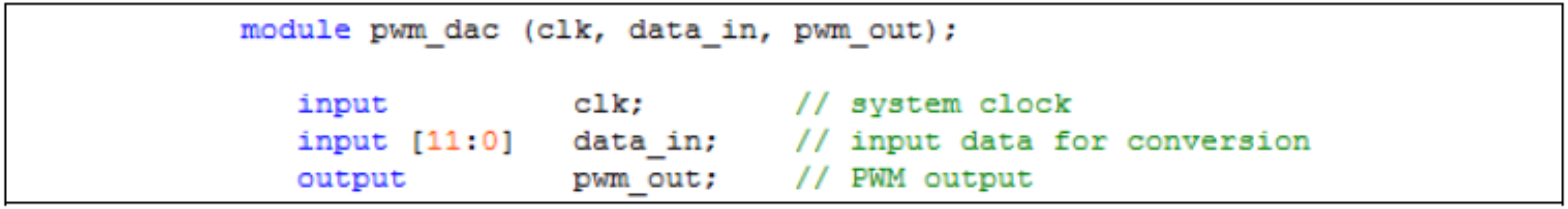
2019 Q1 (e) PWM and DAC

(i) Explain the principle of operation of a 12-bit pulse-width modulation (PWM) digital-to-analogue converter (DAC).

[5]

(ii) Design in Verilog HDL a PWM DAC using the interface shown in *Figure 1.5*.

[5]

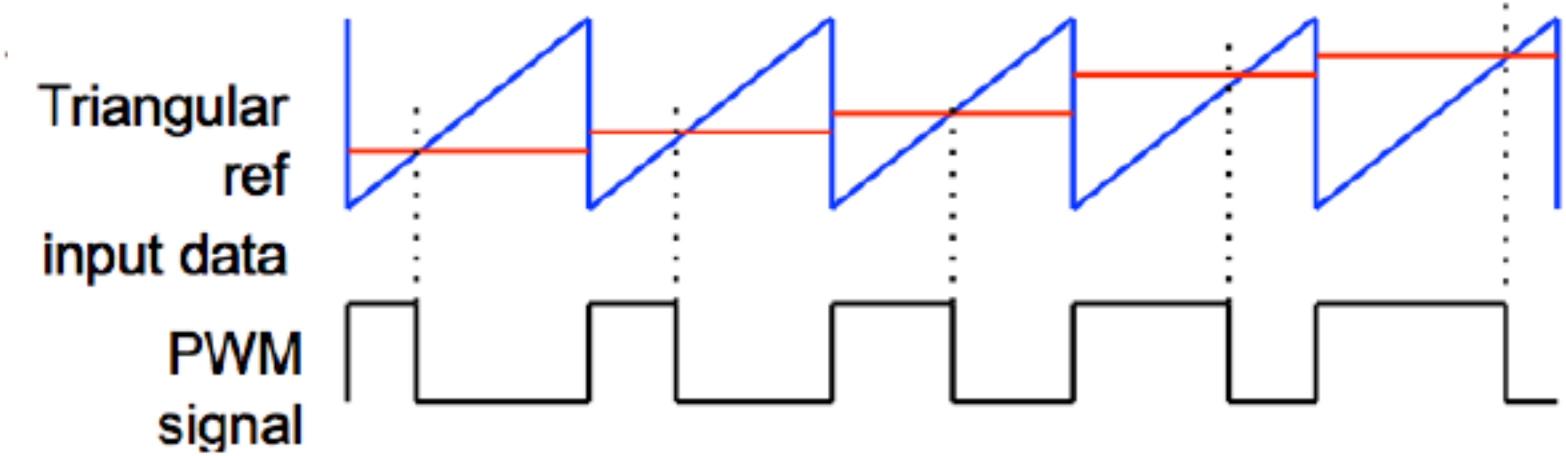


```
module pwm_dac (clk, data_in, pwm_out);

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

2019 Q1 (e) PWM and DAC - solution

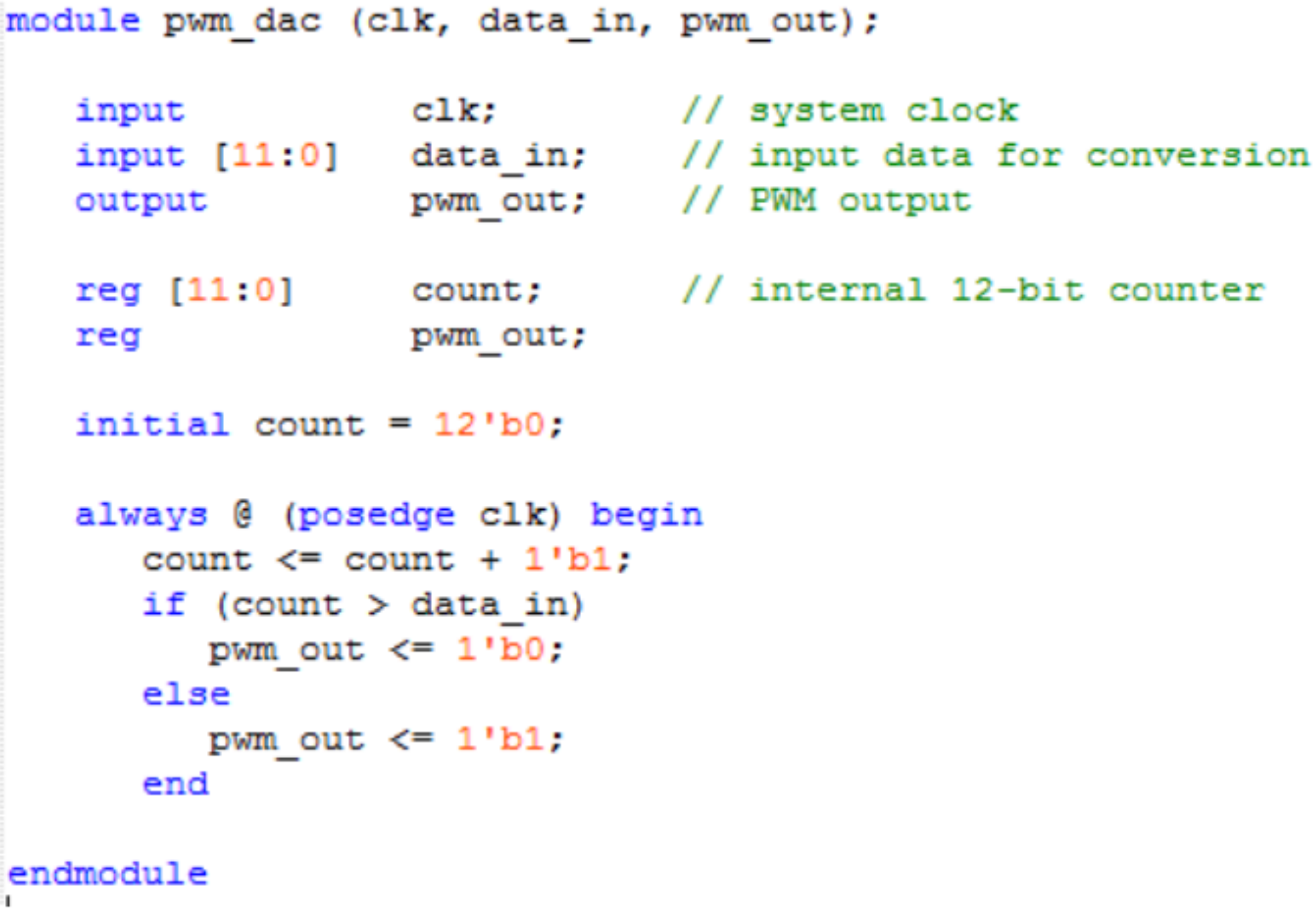
(i) Explain the principle of operation of a 12-bit pulse-width modulation (PWM) digital-to-analogue converter (DAC).

[5]



2019 Q1 (e) PWM and DAC - solution

(ii) Design in Verilog HDL a PWM DAC using the interface shown in *Figure 1.5*.
[5]



```
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
```