May 18, 2016

COMPUTER ENGINEERING DEPARTMENT

COE 405

DESIGN & MODELING OF DIGITAL SYSTEMS

Final Exam

Second Semester (152)

Time: 7:00-10:00 PM

Student Name : __KEY____

Student ID. : ____

Question	Max Points	Score
Q1	24	
Q2	14	
Q3	8	
Q4	14	
Q5	27	
Q6	13	
Total	100	

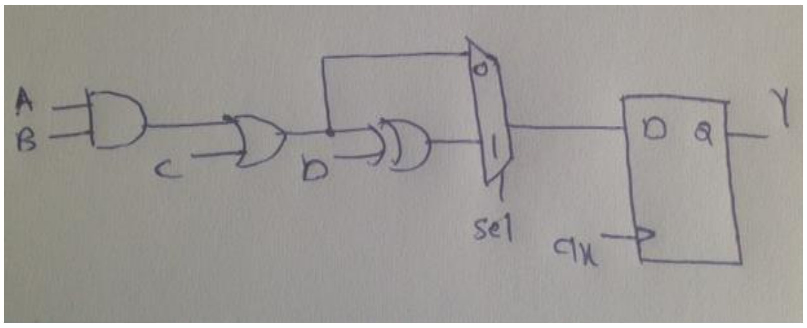
Dr. Aiman El-Maleh

[24 Points]

(Q1) Determine possible circuits that will be synthesized by each of the following modules. Assume that Asynchronous Reset and Set come built-in with FFs. However, for Synchrnous Reset and Set, you need to add the necessary logic.

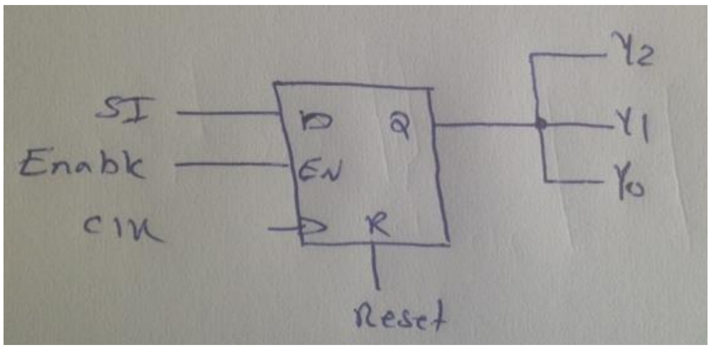
(i)

```
module FinalQ1_1 (output reg Y, input A, B, C, D, Sel, CLK);
  reg X;
  always @ (posedge CLK) begin
      X = A & B; Y = X | C;
      if (Sel) Y = Y ^ D;
  end
endmodule
```

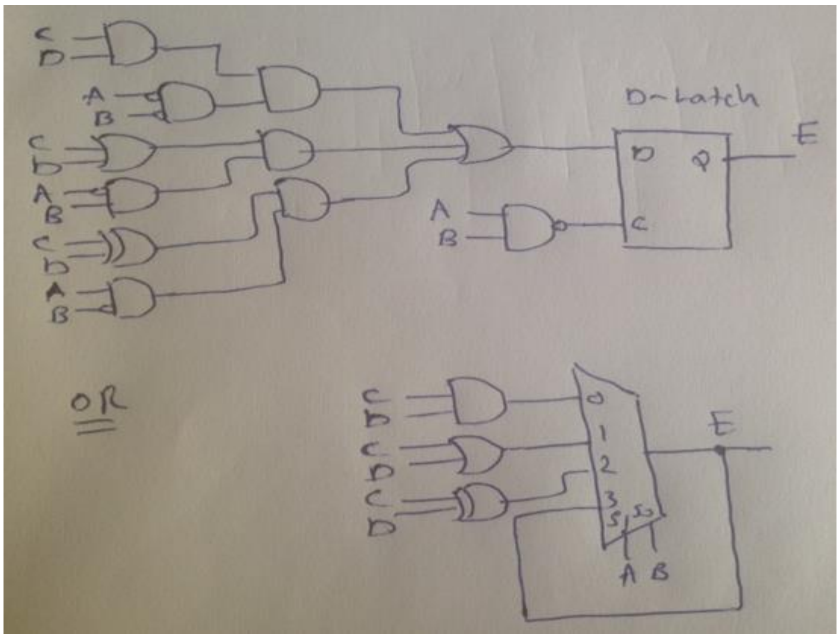


(ii)

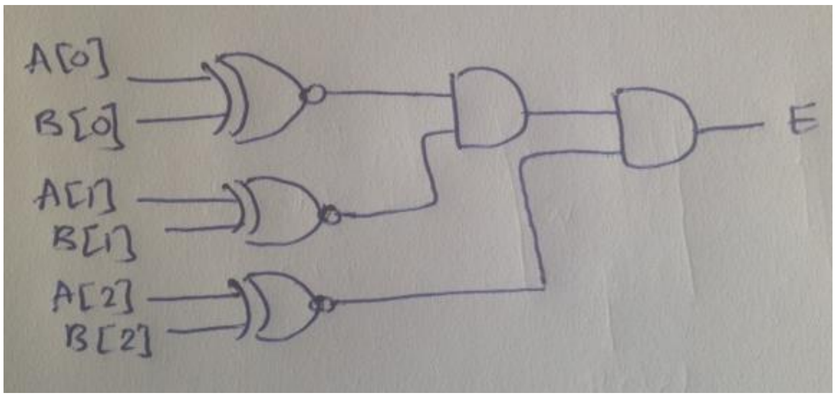
```
module FinalQ1_2 (output reg [2:0] Y, input SI, Reset, Enable, CLK);
   always @ (posedge CLK,  posedge Reset) begin
        if (Reset) Y = 0;
        else if (Enable) begin
                        Y[0] = SI;
                        Y[1] = Y[0];
                        Y[2] = Y[1];
             end
   end
endmodule
```



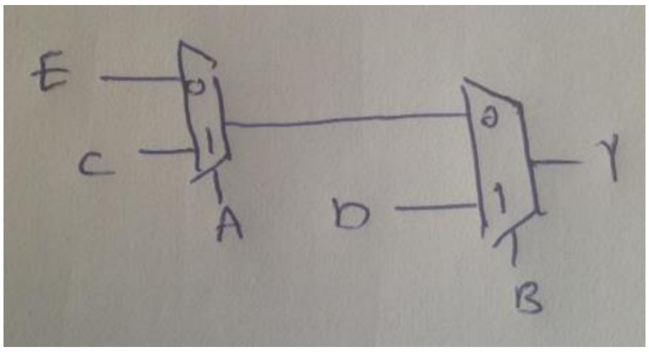
(iii)
```
module FinalQ1_3 (output reg E, input A, B, C, D);

always @(A,B, C, D)
      if (!A && !B)        E = C & D;
      else if (!A && B)  E = C  |  D;
      else if (A && !B)  E = C  ^ D;

endmodule
```



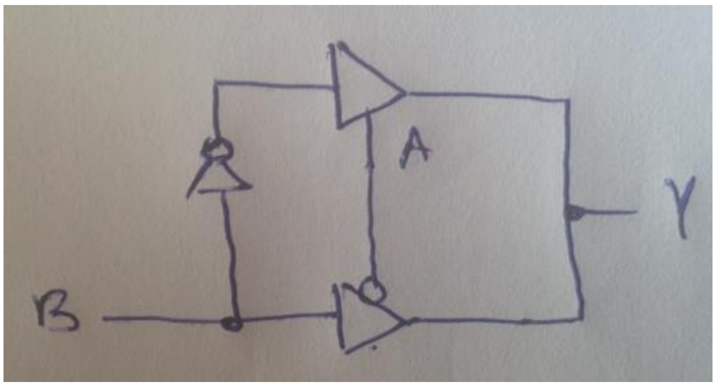
(iv)
```
module FinalQ1_4 #(parameter n=3)(output  reg E, input [n-1:0] A, B);
   integer k;
   always @ (A, B) begin
      E = 1;
      for (k=0; k<n; k=k+1)
          E = E & (A[k] ~^ B[k]);
   end
endmodule
```



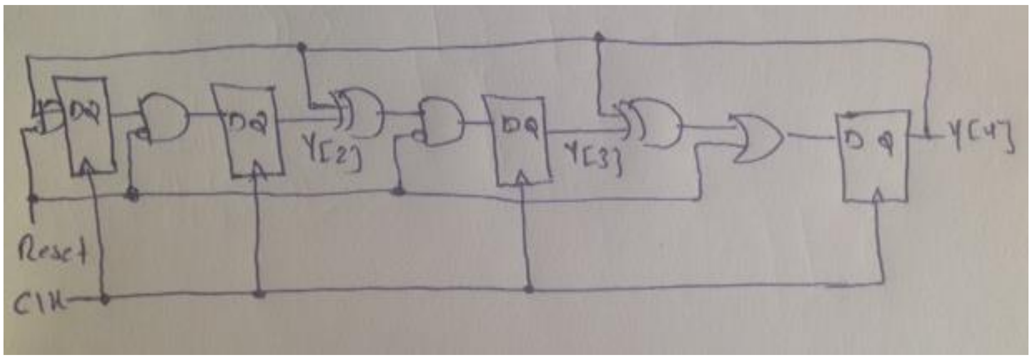
(v)
```
module FinalQ1_5 (output reg Y, input A, B, C, D, E);
  always @ (A, B, C, D, E) begin
      Y = E;
      if (A) Y = C;
      if (B) Y = D;
  end
endmodule
```



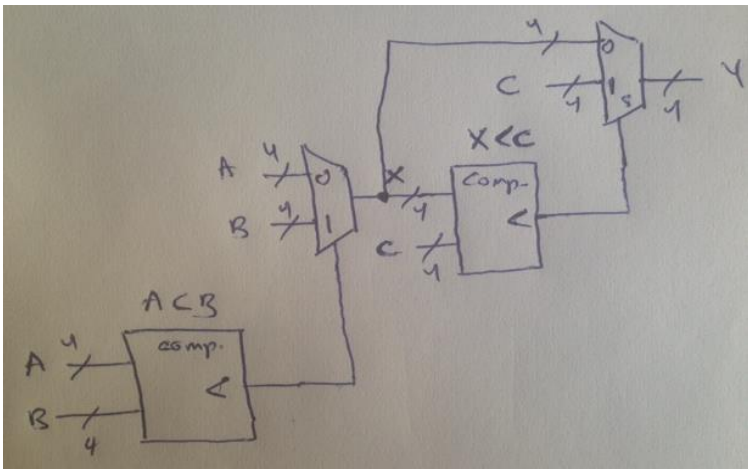
(vi)
```
module FinalQ1_6 (output Y, input A, B);
         assign Y = A? ~B: 1'bz;
         assign Y = !A?  B: 1'bz;
endmodule
```



(vii)
```
module FinalQ1_7 #(parameter  Length=4, initial_state = 4'b0001,
parameter [Length:1]           Tap_Coefficient = 4'b1011)
(input CLK, Reset, output reg [1:Length] Y);
integer k;
always@ (posedge CLK)
   if (Reset) Y<= initial_state;
   else begin
       for (k = 2; k <= Length; k = k+1)
            Y[k] <= Tap_Coefficient[Length-k+1]?Y[k-1]^Y[Length]:Y[k-1];

       Y[1]  <= Y[Length];
  end
endmodule
```

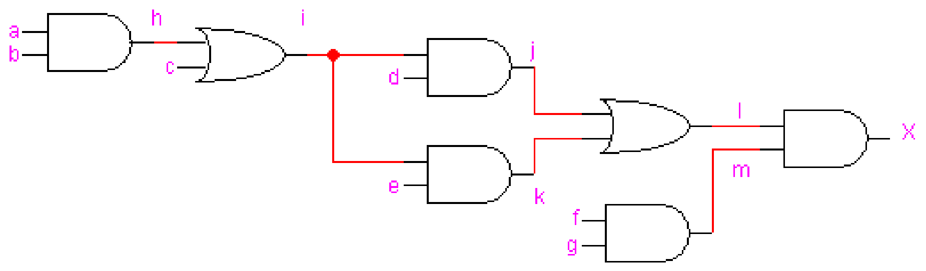


(viii)
```
module FinalQ1_8  ( output [3:0] Y,   input [3:0] A, B, C);

   wire [3:0] X;
   assign X =  GETM (A, B);
   assign Y =  GETM (X, C);

  function [3:0] GETM (input [3:0] O1, O2);
           GETM = (O1 >= O2) ? O1: O2;
  endfunction
endmodule
```



[14 Points]

(Q2) Consider the logic network below with inputs {a, b, c, d, e, f, g} and output {X}:



Assume that the delay of a gate is related to the number of its inputs i.e., the delay of a 2-input AND gate is 2. Also, assume that the input data-ready times are zero for all inputs.

(i) Compute the data ready times and slacks for all gates in the network.

(ii) Determine the topological critical path.

(iii) Suggest an implementation of the function X using only 2-input gates to reduce the delay of the circuit to the minimum possible and determine the maximum propagation delay in the optimized circuit. Has the area been affected?

(i)

Data ready time	Required time	Slack
$t_a = 0$	$\bar{t}_a = 0$	$S_a = 0$
$t_b = 0$	$\bar{t}_b = 0$	$S_b = 0$
$t_c = 0$	$\bar{t}_c = 2$	$S_c = 2-0=2$
$t_d = 0$	$\bar{t}_d = 4$	$S_d = 4-0=4$
$t_e = 0$	$\bar{t}_e = 4$	$S_e = 4-0=4$
$t_f = 0$	$\bar{t}_f = 6$	$S_f = 6-0=6$
$t_g = 0$	$\bar{t}_g = 6$	$S_g = 6-0=6$
$t_h = 2$	$\bar{t}_h = 2$	$S_h = 2-2=0$
$t_i = 4$	$\bar{t}_i = 4$	$S_i = 4-4=0$
$t_j = 6$	$\bar{t}_j = 6$	$S_j = 6-6=0$
$t_k = 6$	$\bar{t}_k = 6$	$S_k = 6-6=0$
$t_l = 8$	$\bar{t}_l = 8$	$S_l = 8-8=0$
$t_m = 2$	$\bar{t}_m = 8$	$S_m = 8-2=6$
$t_x = 10$	$\bar{t}_x = 10$	$S_x = 10-10=0$

(ii) The topological critical paths are:

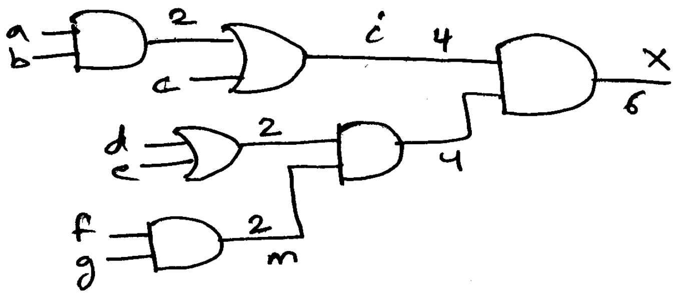
$\{a, h, i, j, l, x\}$
$\{a, h, i, k, l, x\}$
$\{b, h, i, j, l, x\}$
$\{b, h, i, k, l, x\}$

(iii) To optimize the delay of the network, we need to improve the delay of nodes in the critical path.

$l = id + ie \Rightarrow l = i(d+e)$

$x = l \cdot m = i \cdot [(d+e) \cdot m]$

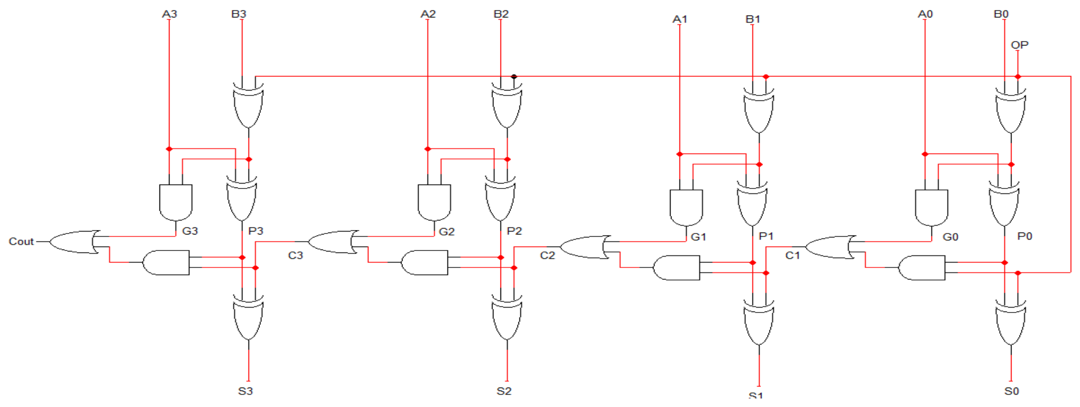
The resulting network:



The maximum propagation delay in the optimized circuit is 6.
Number of literals in the original circuit = 14 literals
Number of literals in optimized circuit = 12 literals
Thus, the area has also been reduced.

[8 Points]

(Q3) It is required to write a parametrizable behavioral Verilog module to model an n-bit adder/subtractor based on the structure given below for a 4-bit adder/subtractor. The circuit receives two n-bit numbers A and B and produces either the addition (A+B) when OP=0 or the subtraction (A-B) when OP=1. The circuit produces an n-bit output S along with Cout. Write a verilog module to generate this structure of the n-bit adder/subtractor. Note that the shown figure is what your ocde should model when n=4.

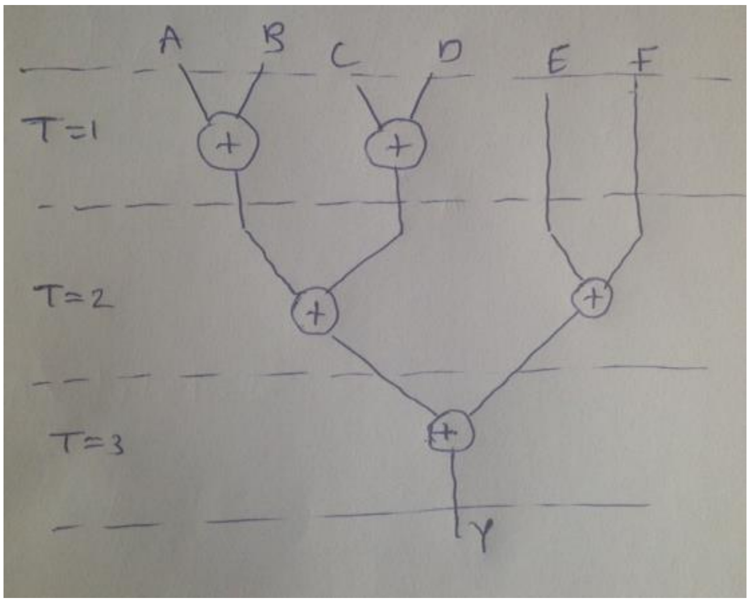


```
module ADDSUB #(parameter n=4)(output reg [n-1:0] S, output Cout,
input [n-1:0] A, B, input OP);


integer i;
reg [n:0] C;
reg [n-1:0] G,P;

assign Cout = C[n];

always @ (A, B, OP) begin

      C[0] = OP;

      for (i=0; i<n; i=i+1) begin

            G[i] = A[i] & (B[i] ^ OP);
            P[i] = A[i] ^ (B[i] ^ OP);
            S[i] = P[i] ^ C[i];
            C[i+1] = G[i] | P[i] & C[i];

      end

end

endmodule
```

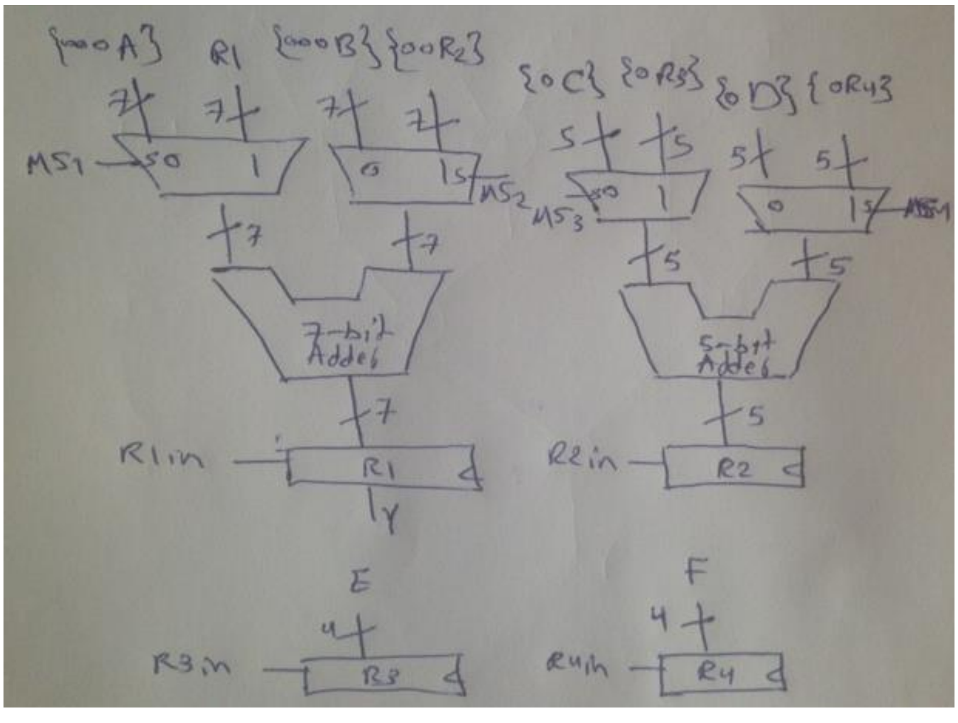
[14 Points]

(Q4) It is required to design a circuit to compute the equation Y=A+B+C+D+E+F, where A, B, C, D, E and F are 4-bit inputs representing unsigned numbers. Assume that inputs are available only during the first cycle when a **Start** signal is asserted. Assume that a **Done** signal will be set when the result is ready and the result will remain held until the next Start operation. Assume that the clock cycle delay is constrained by the delay of the adder.

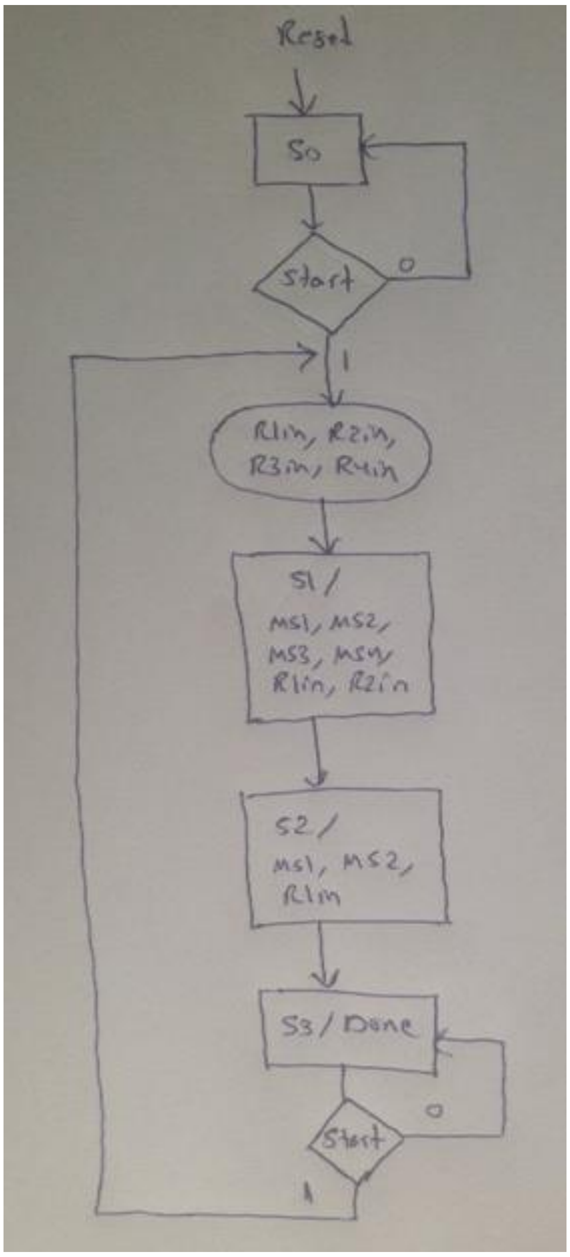
(i) Show a schedule of the operations with minimum latency (i.e., clock cycles) satisfying the area constraint of using a maximum of two adders. Store the output Y in a register.



(ii) Show the DataPath design of your circuit indicating all the control signals and the used adder sizes.



(iii) Show the ASMD diagram of your control unit.



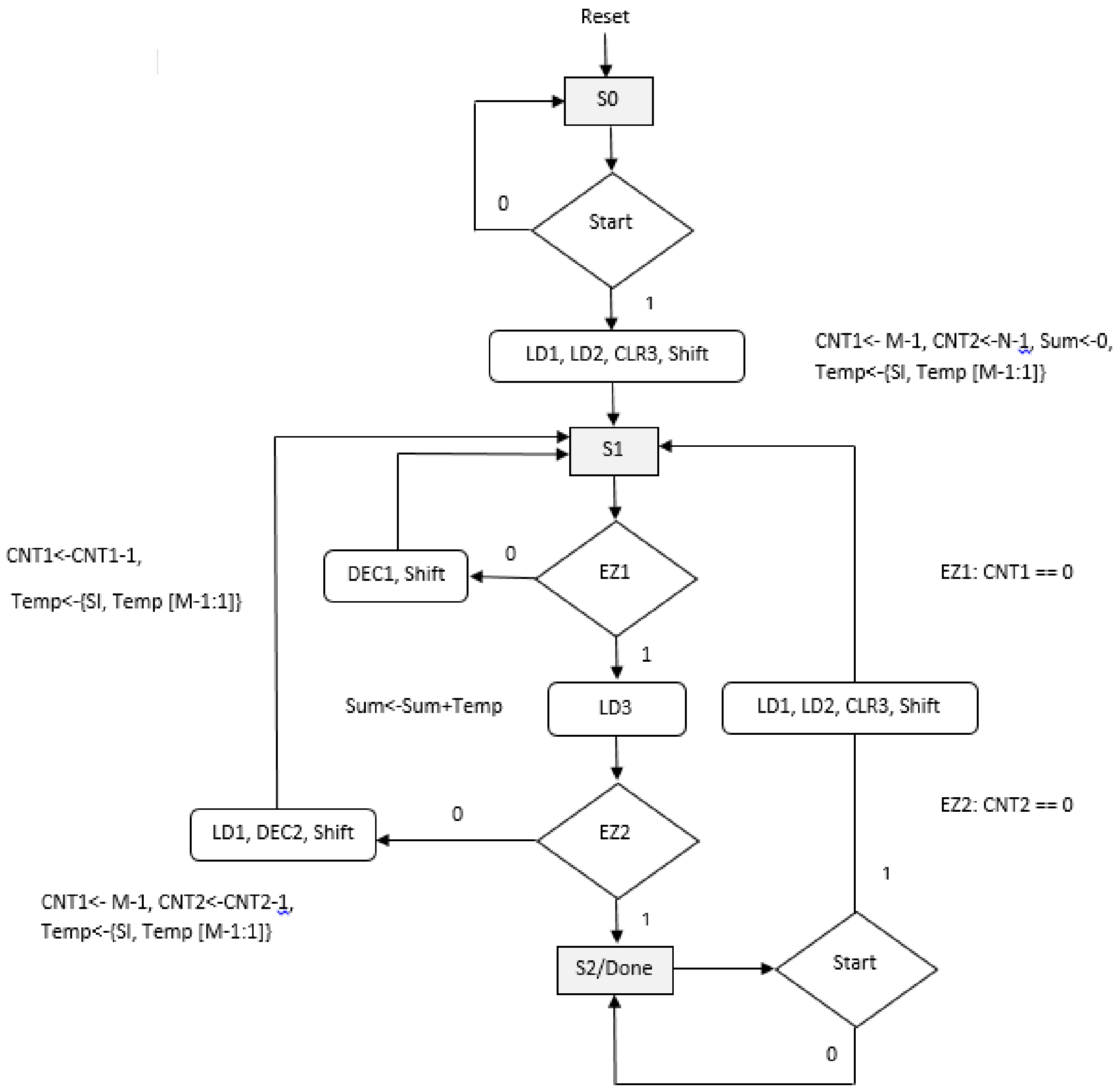
R1 ← A+B. R2 ← C+D
R3 ← E, R4 ← F

R1 ← R1+R2.
R2 ← R3+R4

R1 ← R1+R2.

[27 Points]

(Q5) Given below is the ASMD chart of a circuit that computes the sum of N M-bit numbers serially. Once a start signal is asserted, the circuit reads N M-bit numbers serially (i.e., starting with the first number, it is read 1-bit at a time (from LSB to MSB) through a serial input SI, then the 2nd number is read and so on). The first bit of the first number is applied along with the Start signal. It produces the sum of the entered numbers and asserts a Done signal once the result is ready.



(i) Write a parametrizable behavioral Verilog module, with parameters N and M, to model the data path for this circuit. Assume that you have the following log2 function:

```
function integer log2 (input integer n);
      integer i;
      begin
     log2 = 1;
     for (i=0; 2**i<n; i=i+1)
          log2 = i+1;
       end
endfunction
```

```
module FINALQ5DP #(parameter N=4, M=2) (output reg
[log2(N*(2**M-1))-1:0] Sum, output EZ1, EZ2, input SI, LD1,
LD2, LD3, CLR3, Shift, DEC1, DEC2, CLK);

reg [M-1:0] Temp;
reg [log2(M)-1:0] CNT1;
reg [log2(N)-1:0] CNT2;

assign EZ1 = ~| CNT1;
assign EZ2 = ~| CNT2;

always @(posedge CLK) begin

if (LD1)
      CNT1 <= M-1;

else if (DEC1)
      CNT1 <= CNT1 - 1;

if (LD2)
      CNT2 <= N-1;

else if (DEC2)
      CNT2 <= CNT2 - 1;

if (Shift)
      Temp <= {SI, Temp[M-1:1]};

if (CLR3)
      Sum <= 0;

else if (LD3)
      Sum <= Sum + Temp;

end

function integer log2 (input integer n);
      integer i;
      begin
         log2 = 1;
         for (i=0; 2**i<n; i=i+1)
            log2 = i+1;
      end
endfunction

endmodule
```

(ii) Write a behavioral Verilog module to model the ASMD chart of this circuit.

```
module FinalQ5CU_CU (output reg LD1, LD2, LD3, CLR3, Shift,
DEC1, DEC2, Done, input Start, EZ1, EZ2, Reset, CLK);

parameter S0 = 2'b00, S1=2'b01, S2=2'b10;

reg [1:0] state, next_state;

always @(posedge CLK, posedge Reset)
   if (Reset) state <= S0;
```

```
    else state <= next_state;

always @(state, Start, EZ1, EZ2) begin
    LD1=0; LD2=0; LD3=0; CLR3=0; Shift=0; DEC1=0; DEC2=0;
Done=0;
    case (state)
        S0:
          if (Start)  begin
            LD1=1; LD2=1; CLR3=1; Shift=1;  next_state=S1;  end
          else next_state=S0;
         S1:
           if (EZ1)  begin
              LD3=1;
              if (EZ2)
                    next_state=S2;
              else begin
                       LD1=1; DEC2=1; Shift=1; next_state=S1;
                    end

           end
           else begin
                   DEC1=1; Shift=1; next_state=S1;
                 end
          S2: begin Done=1;
           if (Start)  begin
              LD1=1; LD2=1; CLR3=1; Shift=1; next_state=S1;
           end
           else next_state=S2;
           end
          default: begin
                        next_state='bx; LD1='bx; LD2='bx; LD3='bx;
                        CLR3='bx; Shift='bx; DEC1='bx; DEC2='bx;
                              Done='bx;
                    end

    endcase
end

endmodule
```

(iii) Write a Verilog test bench to test the overall circuit modeled using the following module:

```
module FinalQ5 #(parameter N=8, M=4) (output
[log2(N*(2**M-1))-1:0] Sum, output Done, input SI, Start,
Reset, CLK);
```

Your test bench should test the module for computing the sum of the following three 2-bit numbers: 1, 2, and 3. Your test bench should generate the CLK signal assuming a clock period of 100ns with 50% duty cycle.

```
module FinalQ5 #(parameter N=8, M=4) (output [log2(N*(2**M-
1))-1:0] Sum, output Done, input SI, Start, Reset, CLK);
```

```
FINALQ5DP #(N, M) M1 (Sum, EZ1, EZ2, SI, LD1, LD2, LD3, CLR3,
Shift, DEC1, DEC2, CLK);

FinalQ5CU_CU M2 (LD1, LD2, LD3, CLR3, Shift, DEC1, DEC2, Done,
Start, EZ1, EZ2, Reset, CLK);

function integer log2 (input integer n);
      integer i;
      begin
     log2 = 1;
     for (i=0; 2**i<n; i=i+1)
          log2 = i+1;
       end
endfunction

endmodule


module FinalQ5_Test();

parameter N=3, M=2;

wire [log2(N*(2**M-1))-1:0] Sum;
wire Done;
reg SI, Start, Reset, CLK;


FinalQ5 #(N, M) M1 (Sum, Done, SI, Start, Reset, CLK);

initial begin
CLK = 0;
forever
#50 CLK = ~ CLK;
end

initial begin
Reset=1;
#100 Reset=0; Start=1; SI=1;
#100 Start=0; SI=0;
#100 SI=0;
#100 SI=1;
#100 SI=1;
#100 SI=1;
end

function integer log2 (input integer n);
      integer i;
      begin
     log2 = 1;
     for (i=0; 2**i<n; i=i+1)
          log2 = i+1;
       end
endfunction


endmodule
```

[13 Points]

(Q6) It is required to write a parametrizable Verilog module to model a Queue data structure (First In First Out) with N elements each with M bits. The Queue has M-bit DataOut signal which outputs the content of the Queue when the Queue is read by setting signal Read=1. It also has M-bit DataIn signal which holds the value to be written to the Queue by setting signal Write=1. The Queue has also three additional output signals: Empty to indicate that the Queue is empty, Full to indicate that the queue is full and Error. The Error signal is set when the Queue is requested to be read while it is empty or when the Queue is requested to be written while it is full. In the case of Error, no action will be done by the Queue. Assume also that when both Read and Write signals are set, the Queue will neither be read nor written. Assume that the Queue has synchronous reset which will reset the Queue to the empty state when asserted.

```
module Queue #(parameter N=4, M=4)(output reg [M-1:0] DataOut,
output Full, Empty, Error,  input [M-1:0] DataIn, input Read,
Write, Reset, CLK);

reg [log2(N):0] CNT;
reg [log2(N)-1:0] First, Last;
reg [M-1:0] Queue [N-1:0];

assign Error = (Full && Write) || (Empty && Read);
assign Empty = (CNT==0);
assign Full  = (CNT==N);


always @(posedge CLK) begin

if (Reset) begin
  CNT = 0;
  First = 0;
  Last = 0;
end else begin

       if (Read && !Write && !Empty) begin
            DataOut = Queue[First];
            First = First + 1;
            if (First == N) First = 0;
             CNT = CNT - 1;
       end

       if (Write && !Read && !Full) begin
            Queue[Last] = DataIn;
            Last = Last + 1;
            if (Last == N) Last = 0;
            CNT = CNT + 1;
       end

     end

end
```

```
function integer log2 (input integer n);
      integer i;
      begin
  log2 = 1;
  for (i=0; 2**i<n; i=i+1)
       log2 = i+1;
       end
endfunction


endmodule
```