May 25, 2017

COMPUTER ENGINEERING DEPARTMENT

COE 405

DESIGN & MODELING OF DIGITAL SYSTEMS

Final Exam

Second Semester (162)

Time: 7:00-10:00 PM

Student Name : _KEY_____

Student ID. : _____

Question	Max Points	Score
Q1	10	
Q2	12	
Q3	10	
Q4	16	
Q5	24	
Total	72	

Dr. Aiman El-Maleh

[10 Points]

(Q1) Consider the logic network defined by the following expression:

$$x = a\,b\,c + a\,b\,d + a\,b'\,c'\,d' + a'\,b\,c'\,d' + a'\,b'\,c + a'\,b'\,d + c\,e + c\,f + d\,e + d\,f$$

Compute the weight of the double cube divisors $d_1 = a\,b + a'\,b'$ and $d_2 = c + d$. Extract the double cube divisor with the highest weight and show the resulting network after extraction and the number of literals saved.

Double Cube Divisor	Base
$d_1 = ab + \bar{a}\bar{b}$	c, d
$\bar{d}_1 = a\bar{b} + \bar{a}b$	$\bar{c}\bar{d}$
$d_2 = c + d$	$ab, e, f, \bar{a}\bar{b}$

weight $(d_1) = 3 * 4 - 3 - 4 + 1 + 1 + 2 = 9$

weight $(d_2) = 4 * 2 - 4 - 2 + 2 + 1 + 1 + 2 + 2 = 10$

Since d_2 has higher weight, it will be extracted.

The resulting network after extraction of d_2 is:

$[1] = c + d$

$x = ab[1] + a\bar{b}\,\overline{[1]} + \bar{a}b\,\overline{[1]} + \bar{a}\bar{b}[1] + [1]e + [1]f$

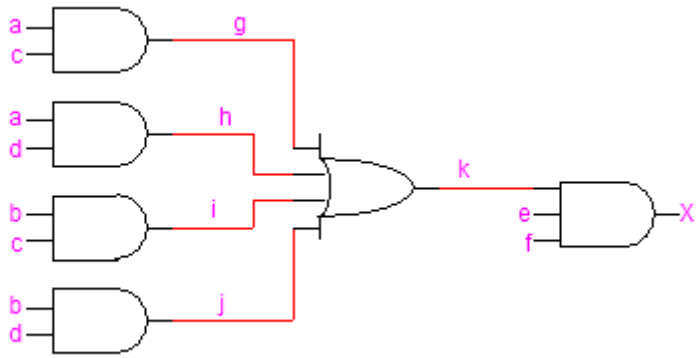
18 literals

Original number of literals = 28 literals

Number of literals saved = 10 literals

[12 Points]

(Q2) Consider the logic network below with inputs *{a, b, c, d, e, f}* 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 except input *a*, which has a data-ready time of 2.

(i) Compute the data ready times, data required times and slacks for all vertices in the network.

(ii) Determine the topological critical path.

(iii) Suggest an implementation of the function *X* 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
$t_a = 2$	$T_a = \min(4-2, 4-2) = 2$	$S_a = 2-2 = 0$
$t_b = 0$	$T_b = \min(4-2, 4-2) = 2$	$S_b = 2-0 = 2$
$t_c = 0$	$T_c = \min(4-2, 4-2) = 2$	$S_c = 2-0 = 2$
$t_d = 0$	$T_d = \min(4-2, 4-2) = 2$	$S_d = 2-0 = 2$
$t_e = 0$	$T_e = 8$	$S_e = 8-0 = 8$
$t_f = 0$	$T_f = 8$	$S_f = 8-0 = 8$
$t_g = 4$	$T_g = 4$	$S_g = 4-4 = 0$
$t_h = 4$	$T_h = 4$	$S_h = 4-4 = 0$
$t_i = 2$	$T_i = 4$	$S_i = 4-2 = 2$
$t_j = 2$	$T_j = 4$	$S_j = 4-2 = 2$
$t_K = 8$	$T_K = 8$	$S_K = 8-8 = 0$
$t_X = 11$	$T_X = 11$	$S_X = 11-11 = 0$

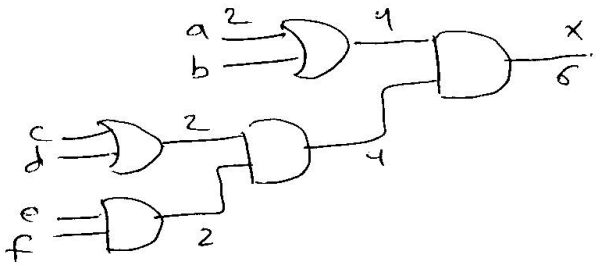
(ii) The topological critical paths are:

{a, g, K, x} and {a, h, K, x}

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

K can be factored into (a+h)(c+d)

Then, x can be implemented as follows:

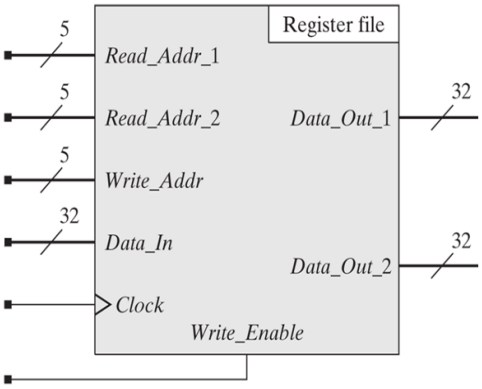


The resulting delay is 6.
Number of literals is 10.

Thus, we have improved the delay from 11 to 6 and area from 15 literals to 10 literals.

[10 Points]

(Q3) It is required to write a Verilog model to model a parametrizable egister file that has two read ports and one write port. The number of address bits for addressing registers and the size of each register should be used as parameters with default values of 5 address bits (i.e., 32 registers) and 32-bits. Register 0 should be always having a constant value of 0 and should not be written to. Your register file should be declared as a two-dimensional array. The block diagram of the regitser file with default parameters is given below:

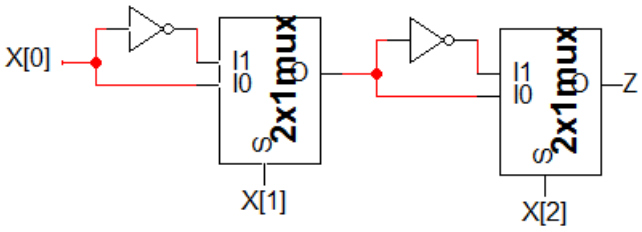



```
module Register_File #(parameter  word_size=32, addr_size=5)
( output  [word_size-1:0] Data_Out_1, Data_Out_2,
input [word_size-1:0] Data_In,
input [addr_size-1:0] Read_Addr_1, Read_Addr_2, Write_Addr,
input  Write_Enable, Clock);

reg [word_size-1:0]  Reg_File[2**addr_size-1:0];

initial begin
  Reg_File[0]=0;
end

assign Data_Out_1 = Reg_File[Read_Addr_1];
assign Data_Out_2= Reg_File[Read_Addr_2];

always @(posedge Clock)
  if (Write_Enable==1'b1 && Write_Addr != 0)
    Reg_File[Write_Addr] <= Data_In;
endmodule
```

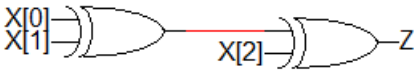
[16 Points]

(Q4) 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 Final_1 #(parameter n=3) (output reg Z, input [n-1:0] X);
    integer i;
    always @(X) begin
      Z=0;
      for (i=0; i<n; i=i+1)
        if (X[i])
          Z = ~Z;
    end
endmodule
```

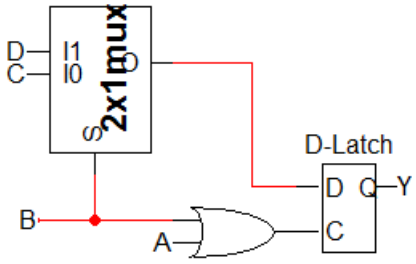


OR



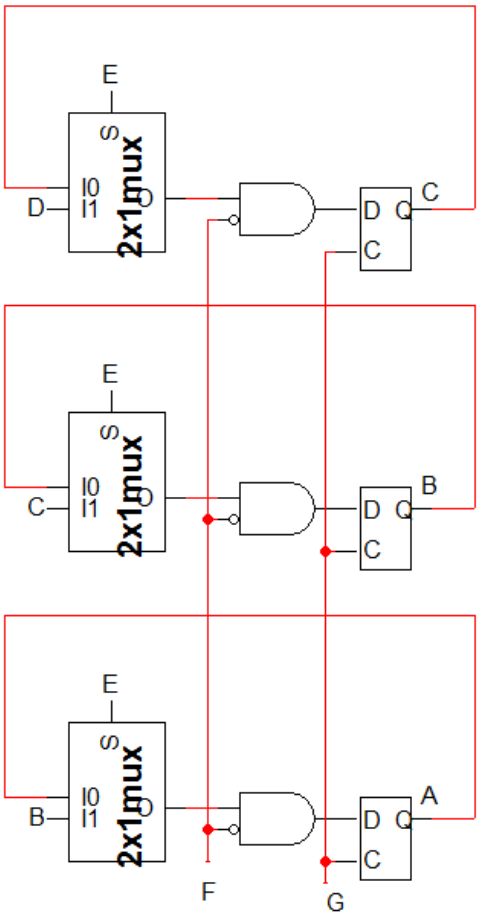
(ii) module Final_2 (output reg Y, input A, B, C, D);

```
always @(A,B,C,D) begin
   if (A) Y = C;
   if (B) Y = D;
end

endmodule
```

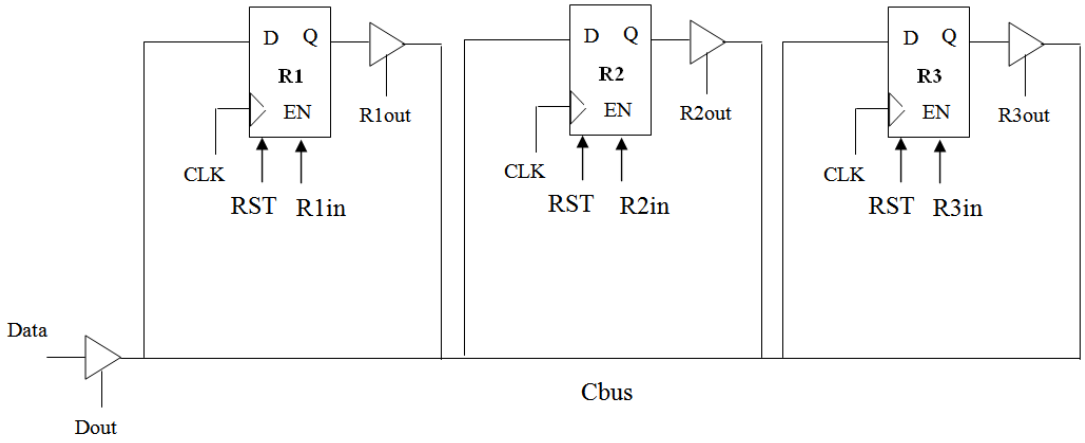


(iii) module Final_3(output reg A, B, C, input D, E, F, CLK);

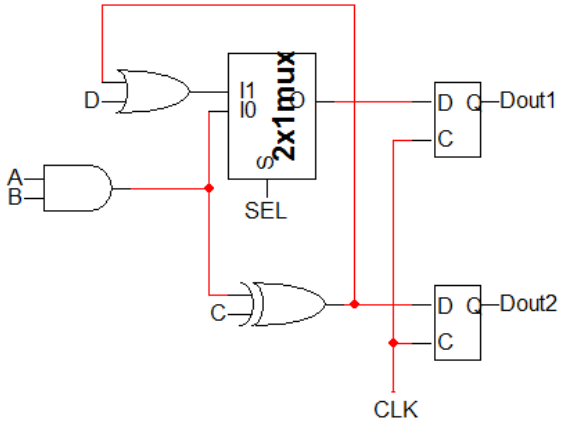
```
always @(posedge CLK)
 if (F) begin
    A<=1'b0; B<=1'b0; C<=1'b0;
 end
 else if (E) begin
          C <= D;
          B <= C;
          A <= B;
 end
endmodule
```



(iv)
```
module Final_4 #(parameter n=4) (output reg [n-1:0] R1, R2, R3,  input CLK, RST,
R1in, R2in, R3in, R1out, R2out, R3out, Dout, input [n-1:0] Data);
  wire [n-1:0] Cbus;
  always @ (posedge CLK, posedge RST) begin
        if (RST) begin R1 <= 0; R2 <= 0; R3 <= 0;   end
        else begin
                if (R1in) R1 <= Cbus;
                if (R2in) R2 <= Cbus;
                if (R3in) R3 <= Cbus;
        end
  end
  assign Cbus = R1out? R1:{n{1'bz}};
  assign Cbus = R2out? R2:{n{1'bz}};
  assign Cbus = R3out? R3:{n{1'bz}};
  assign Cbus = Dout? Data:{n{1'bz}};
endmodule
```



(v)
```
module Final_5 (output reg Dout1, Dout2, input A, B, C, D, SEL, CLK);
  always @ (posedge CLK) begin
      Dout1 = A & B; Dout2 = Dout1 ^ C;
      if (SEL) Dout1 = Dout2 | D;
  end
endmodule
```



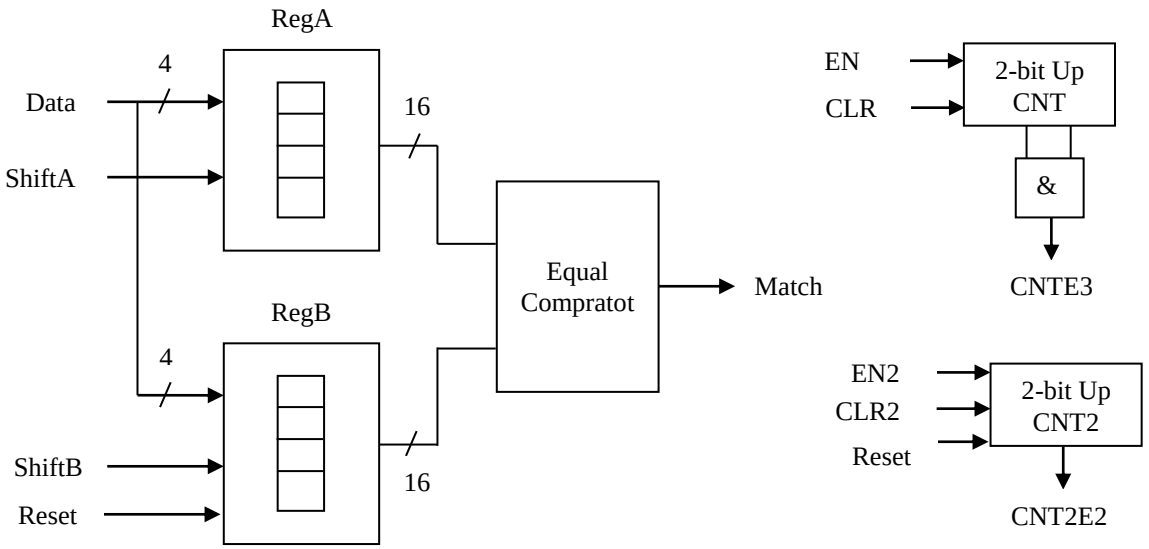
[24 Points]

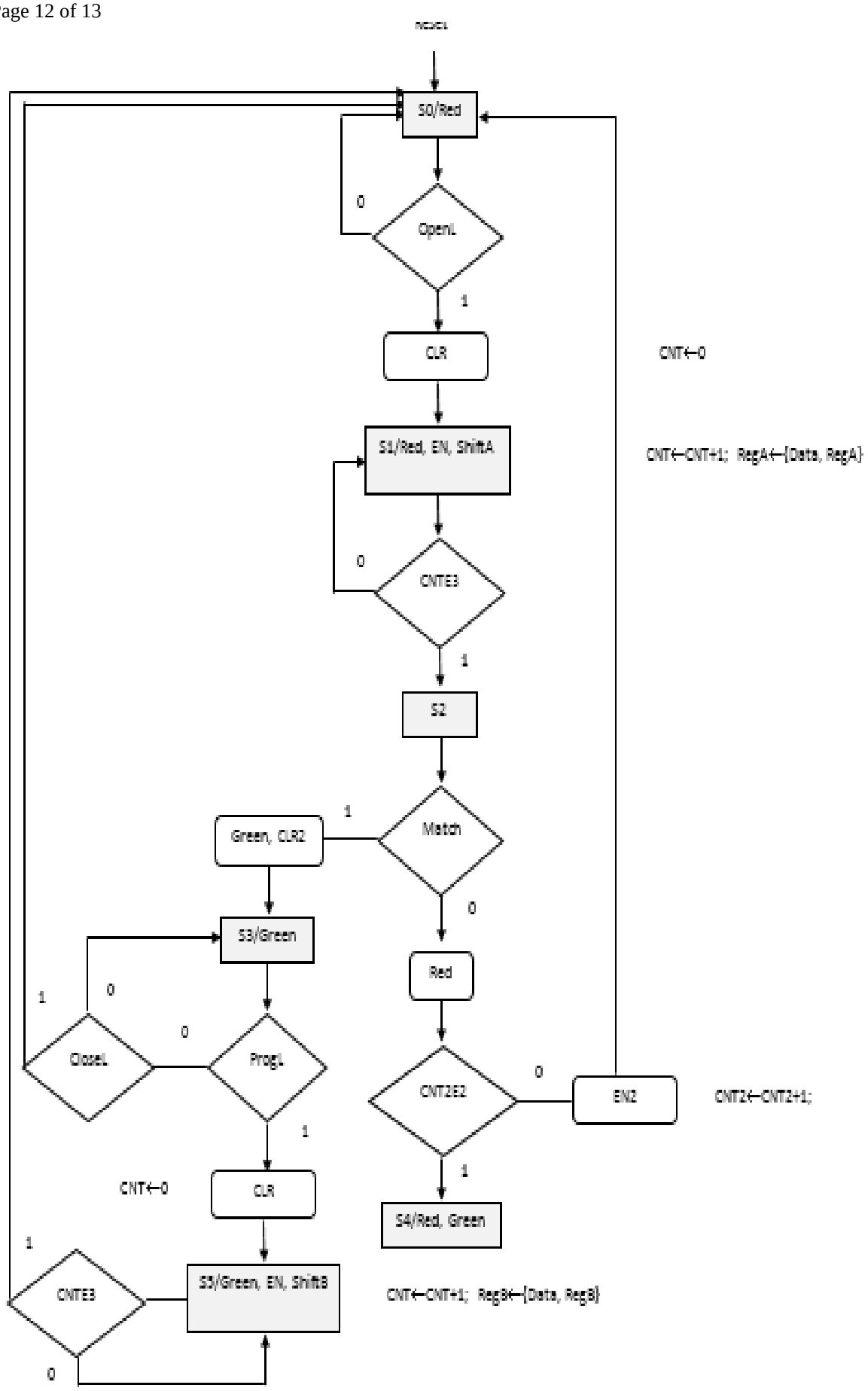
(Q5) You are required to design a programmable digital lock circuit. The lock has 5 inputs: OpenL, CloseL, ProgL, Reset and 4-bit serial input. The lock is closed when either the Reset input is pressed or when the CloseL input is pressed when the lock is open. The lock has also two outputs Red and Green. If the lock is opened, Green is ON and if it is closed, Red is ON. If the lock is jammed both Red and Green are ON. The circuit receives the input serially representing four BCD digits received serially digit by digit starting from the least significant digit (i.e., 4-bits every clock cycle). If the input combination matches a 4-BCD digit stored password, the lock is opened, otherwise the lock remains closed. The user should be able to re-program the lock to store a new password. Durting the attempt of opening the lock, if 3 wrong 4-BCD digit combinations are entered (i.e., 3 incorrect attempts of entering the password), the lock jams and needs to be reset. Assume that the lock can be only reset by an operator and that when the lock is reset it will assume a stored password of 0000.
The user cannot program the lock unless the lock is open. Once the lock is programmed, it will close automatically. Assume that when the user presses OpenL or ProgL, the four digits will be transmitted to the lock starting from the next cycle and that the input will be sent in 4 consecutive cycles. Every time the user attepmts to open the lock, he has to press the Open input and then supply the four digits serially starting from the next clock cycle. Assume that Reset is synchronous.

(i) Design the data path unit for the digital lock circuit.

(ii) Obtain the ASMD chart of the control unit that will control the operation of the digital lock circuit.

(iii) Write a single behavioral Verilog module for modeling your datapath. Do not write sepearte Verilog modules for individual compoenents and instantiate them.




S0/Red
OpenL
0
1
CLR
CNT←0
S1/Red, EN, ShiftA
CNT←CNT+1; RegA←{Data, RegA}
CNTE3
0
1
S2
Match
1
Green, CLR2
0
Red
S3/Green
1
0
CloseL
0
ProgL
1
CNT←0
CLR
CNT2E2
0
EN2
CNT2←CNT2+1;
1
S4/Red, Green
1
CNTE3
S5/Green, EN, ShiftB
CNT←CNT+1; RegB←{Data, RegB}
0

```
module Lock_DP
(output Match, CNTE3, CNT2E2,  input [3:0] Data, input ShiftA,
ShiftB, EN, CLR, EN2, CLR2, Reset, CLK);


reg [15:0] RegA, RegB;
reg [1:0] CNT, CNT2;


// RegA

always @(posedge CLK) begin
   if (ShiftA)
     RegA <= {Data , RegA[15:4]};

end

// RegB

always @(posedge CLK) begin
if (Reset)
     RegB <= 16'b0;
else if (ShiftB)
     RegB <= {Data, RegB[15:4]};

end

// Match

assign Match = (RegA == RegB);


// 1st Counter

always @(posedge CLK) begin
   if (CLR)
      CNT <= 2'b0;
   else if (EN)
      CNT <= CNT + 1;
end
assign CNTE3 = CNT[1] & CNT[0];


// 2nd Counter

always @(posedge CLK) begin
      if (CLR2 || Reset)
         CNT2 <= 2'b0;
      else if (EN2)
          CNT2 <= CNT2 + 1;
end
assign CNT2E2 = CNT2[1];

endmodule
```