May 20, 2013

COMPUTER ENGINEERING DEPARTMENT

COE 405

DESIGN & MODELING OF DIGITAL SYSTEMS

Final Exam

Second Semester (122)

Time: 7:00-10:00 PM

OPEN BOOK EXAM

Student Name : _KEY_____

Student ID. : _____

Question	Max Points	Score
Q1	28	
Q2	20	
Q3	36	
Q4	16	
Total	100	

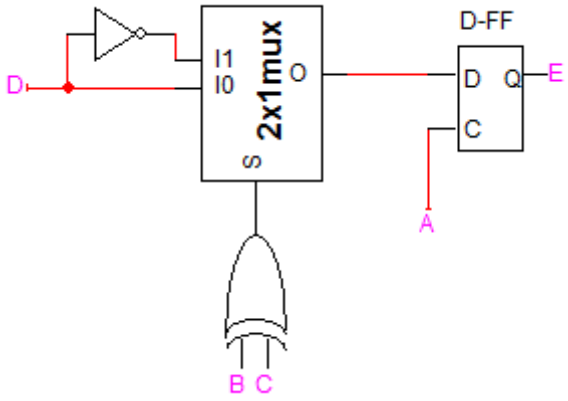
Dr. Aiman El-Maleh

[28 Points]

(Q1) Determine possible circuits that will be synthesize by each of the following modules:

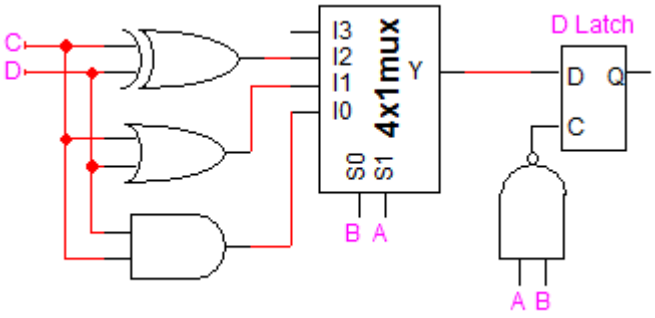
(i)

```
module FinalQ1_1 (output reg E, input A, B, C, D);

always @(posedge A)
        if (B==C) E <= D;
        else      E <= ~ D;
endmodule
```



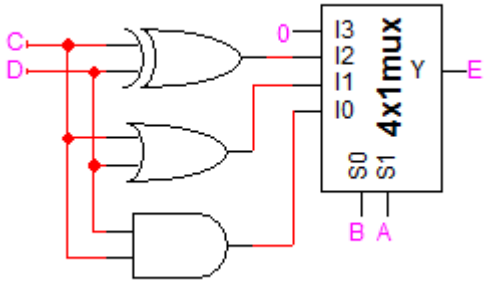
(ii)

```
module FinalQ1_2 (output reg E, input A, B, C, D);

always @(A, B, C, D)
        case ({A,B})
        2'b00:  E = C & D;
        2'b01:  E = C | D;
        2'b10:  E = C ^ D;
        endcase
endmodule
```



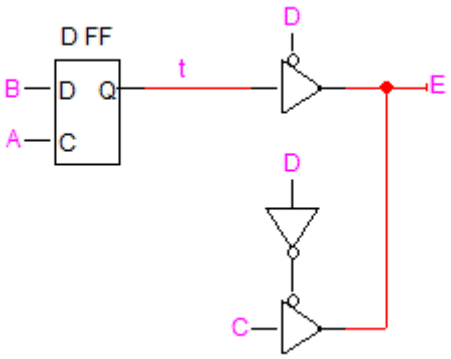
(iii)

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

always @(A, B, C, D) begin
        E = 0;
        case ({A,B})
        2'b00:  E = C & D;
        2'b01:  E = C | D;
        2'b10:  E = C ^ D;
        endcase
end
endmodule
```



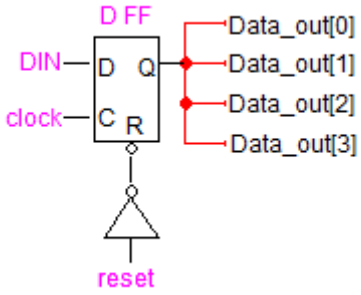
(iv)

```
module FinalQ1_4 (output E, input A, B, C, D);

reg t;
always @(posedge A)
        t <= B;

assign E = D?C:1'bz;
assign E = ~D?t:1'bz;

endmodule
```



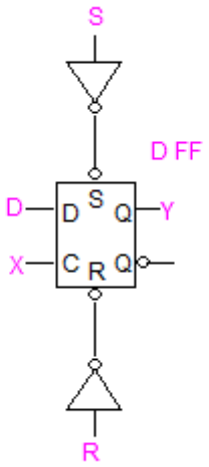
(v)

```
module FinalQ1_5 (output reg [3:0] Data_out, input DIN, clock, reset);
        always @ (posedge clock,  posedge reset) begin
          if (reset) Data_out = 0;
          else begin
                        Data_out[0]=DIN;
                        Data_out[1]=Data_out[0];
                        Data_out[2]=Data_out[1];
                        Data_out[3]=Data_out[2];
          end
        end
endmodule
```



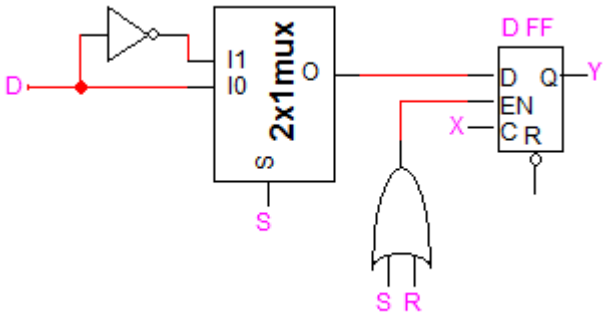
(vi)

```
module FinalQ1_6 (output reg Y, input D, S, R, X);

always @(posedge X, posedge S, posedge R )
  if (R) Y <= 0;
  else if (S) Y <=1;
  else Y <= D;
endmodule
```



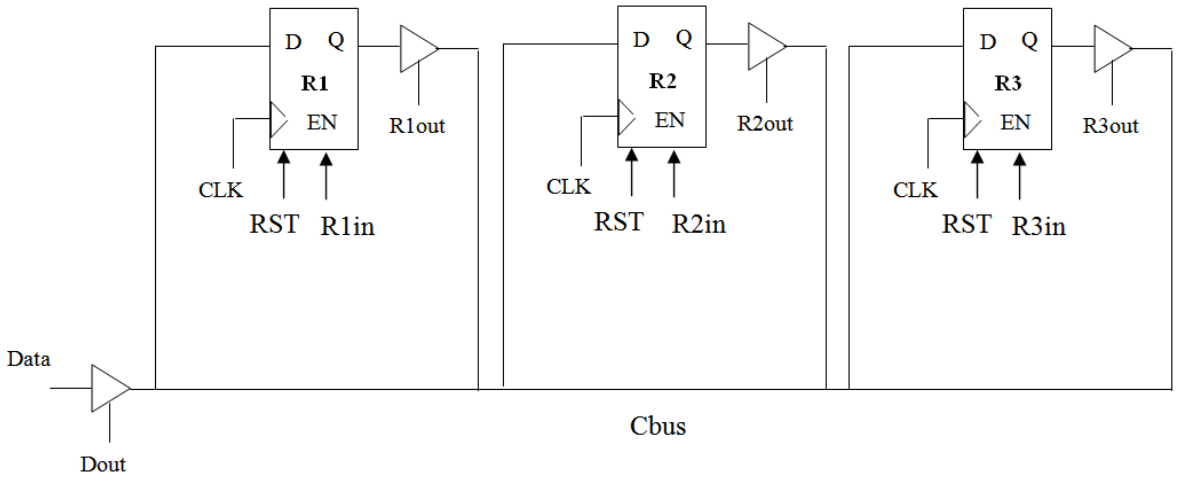
(vii) module FinalQ1_7 (output reg Y, input D, S, R, X);

```
always @(posedge X)        begin
  if (R) Y <= D;
  if (S) Y <=~D;
end
endmodule
```



[20 Points]

(Q2) The three n-bit registers R1, R2, and R3, are connected through a tri-state bus (Cbus) to allow the transfer of the content of any register to any other register as shown below:



(i) Write a Verilog model **Datapath** to model the given datapath showing the inetrface signals assuming R1, R2, R3 as **output** signals, CLK, Reset, R1in, R2in, R3in, R1out, R2out, R3out, Dout and Data as **input** signals. Assume that RST is **Asynchronous** reset that resets the machine when set to 1. Use parametr **n** for determining the width of registers with a default value of 8.

```
module Datapath #(parameter n=8)
(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
```

(ii) Write a test bench to do the following assuming that the period of the clock is 100 ns and that the duty cycle is 50%:

- Initialize all the registers to 0 using the RST signal at time = 200 ns.
- Assign RST and all other control signals to 0 at time 300 ns.
- Assign Data=5 and Dout=1 at time 400 ns.
- Copy the value 5 from the bus into register R1 in the next cycle.
- Move the value of R1 into R2 in the following cycle.
- All registers keep their value in the subsequent clock cycles

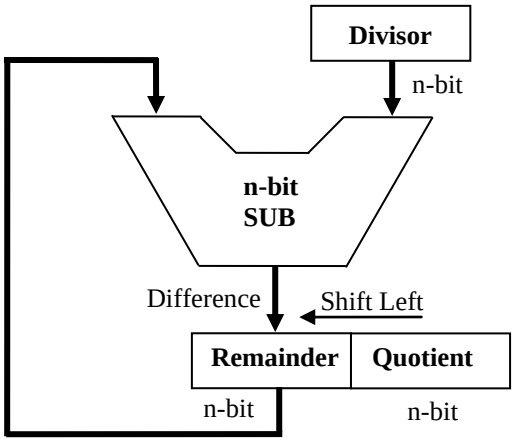
```
module t_Datapath #(parameter n=8)();

wire [n-1:0] R1, R2, R3;
reg CLK, RST, R1in, R2in, R3in, R1out, R2out, R3out, Dout;
reg [n-1:0] Data;

Datapath M1(R1, R2, R3, CLK, RST, R1in, R2in, R3in, R1out, R2out, R3out, Dout, Data);

initial begin CLK = 0; forever #50 CLK = ~CLK; end
initial begin
    #200 RST=1;
    #100 RST=0; R1in=0; R2in=0; R3in=0; R1out=0; R2out=0; R3out=0; Dout=0;
    #100 begin Data='d5; Dout=1; end
    @ (posedge CLK) begin  R1in=1; end
    @ (posedge CLK) begin Dout=0; R1out=1; R1in=0; R2in=1; end
    @ (posedge CLK) begin Dout=0; R1out=0; R1in=0; R2in=0; end
end

endmodule
```

[36 Points]

(Q3) It is required to design a module to perform **unsigned division** of an n-bit dividend number, A, by an n-bit divisor number, B. The divider produces an n-bit quotient and an n-bit remainder. Assume that the divider will be a **sequential divider** and it will set the signal **Ready** to 1 when the quotient and remainder results are ready. The divider has an **Asynchronous Reset** after which in the next clock cycle it starts the division process if the **Start** signal is 1. The quotient and remainder willl maintain their values unless the divider is reset again. The module of the divider is given below where A is the dividend, B is the divider, Q is the quotient and R is the reminder:

```
module UDIV # (parameter n= 4)
(output [n-1:0] Q, R, output Ready,
input [n-1:0] A, B,
input Start, Reset, CLK);

DPath_DIV #(n) DPU(Q, R, CEN, DGEZ, LDR, ClearR, LDQ, SetQ0, LDB, Shift, INC,
CLRC, CLK, A, B);
CU_DIV CU (LDR, ClearR, LDQ, SetQ0, LDB, Shift, INC, CLRC, Ready, Start, CLK,
Reset, DGEZ, CEN);

endmodule
```

Part of the Datapath of the divider is given below:



The algorithm for performing sequential division is as follows:

1. Set Quotient=Dividened, Set Remainder=0.
2. Shift(Remainder,Quotient) Left by 1 bit
3. Difference=Remainder-Divisor
4. If (Difference≥ 0) Then
   Remainder=Difference
   Set Least Significant bit of Quotient to 1.
   End If;
1. If (#iterations<N) Then Goto Step 2.

An example of applying the algorithm for a 4-bit divider with dividened=1110 and divisor=0011 is given below. Note that the Quotient=0100 and the Remainder=0010.

Iteration		Remainder	Quotient	Divisor	Difference
0	Initialize	0 0 0 0	1 1 1 0	0 0 1 1	
1	1: SLL, Difference	0 0 0 1 ←	1 1 0 0	0 0 1 1	1 1 1 0
	2: Diff < 0 => Do Nothing				
2	1: SLL, Difference	0 0 1 1 ←	1 0 0 0	0 0 1 1	0 0 0 0
	2: Rem = Diff, set lsb Quotient	0 0 0 0	1 0 0 1		
3	1: SLL, Difference	0 0 0 1 ←	0 0 1 0	0 0 1 1	1 1 1 0
	2: Diff < 0 => Do Nothing				
4	1: SLL, Difference	0 0 1 0 ←	0 1 0 0	0 0 1 1	1 1 1 1
	2: Diff < 0 => Do Nothing				

The description of various signals is illustrated in the table below:

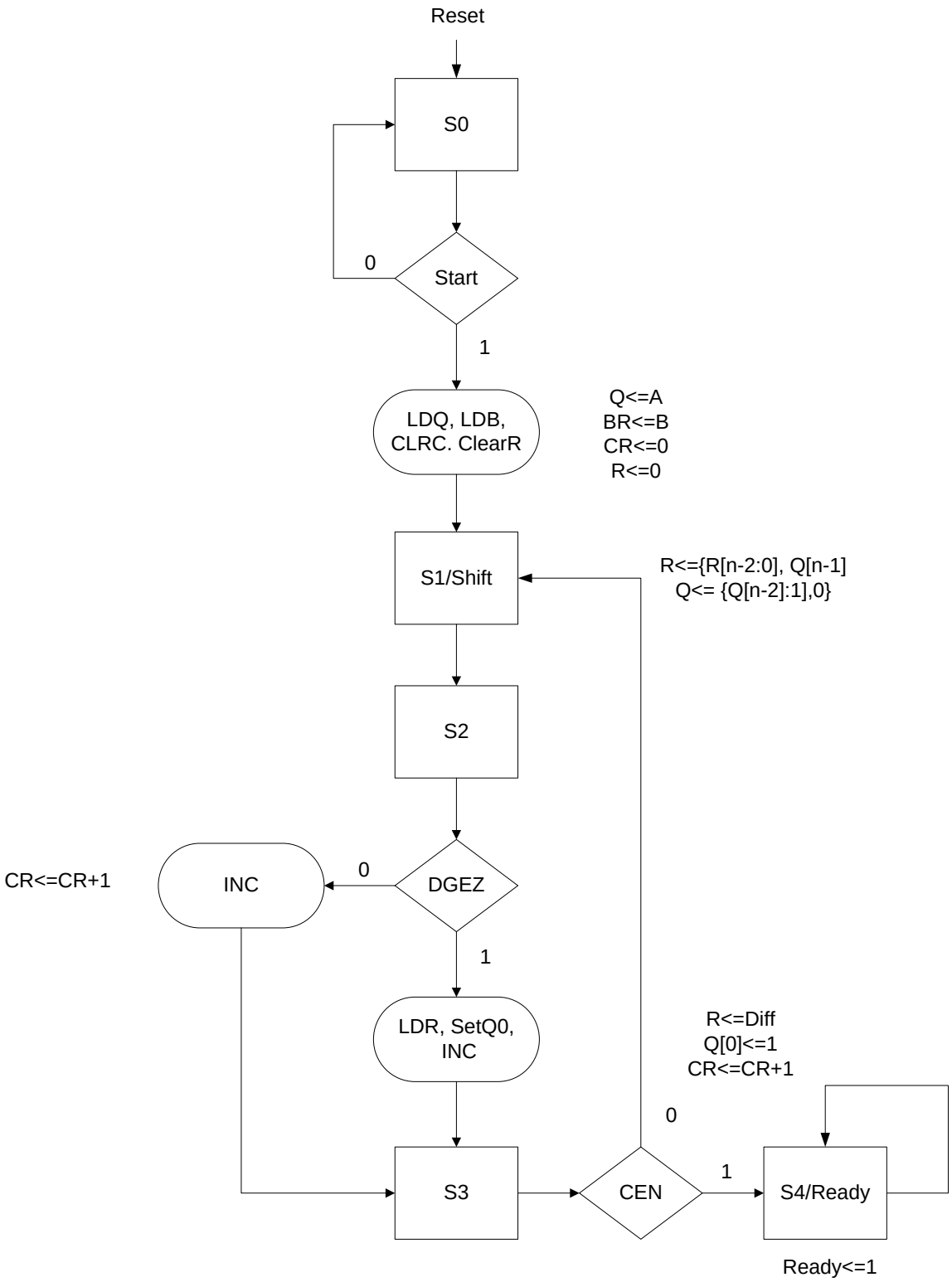
Signal	Role
LDR	Load the remainder register
ClearR	Clear the remainder register
LDQ	Load the quotient register
SetQ0	Set quotion register bit 0 to 1
LDB	Load the B register with the divisor
Shift	Shift the quotient and remainder register one bit to the left
INC	Increment counter
CLRC	Clear counter
DGEZ	Set when difference is ≥ 0
CEN	Set when counter is equal to n

(i) Complete the Verilog model given below for modeling the Datapath of the divider.

```
module DPath_DIV #(parameter n=4)
(output reg [n-1:0] Q, R, output CEN, DGEZ,
input LDR, ClearR, LDQ, SetQ0, LDB, Shift, INC, CLRC, CLK,
input [n-1:0] A, B);

reg [n-1:0] BR, CR;
wire [n-1:0] Diff;

always @ (posedge CLK)
begin
// Divisor Register
```

```
        if (LDB) BR <=  B;

    // Quotient Register
        if (LDQ) Q <=  A;
        else if (Shift) Q <=  {Q[n-2:0],1'b0};
        else if  (SetQ0) Q[0] <= 1'b1;

    // Remainder Register
        if (LDR) R <=  Diff;
        else if (Shift) R <=  {R[n-2:0],Q[n-1]};
        else if  (ClearR) R <= 0;

    // Counter Register
        if (CLRC) CR <=  0;
        else if (INC) CR <=  CR+1;
    end

    assign CEN = (CR==n);
    assign      Diff = R - BR;
    assign      DGEZ = ~ Diff[n-1];

    endmodule
```

(ii) Write an ASMD chart for the **control unit** of the divider.



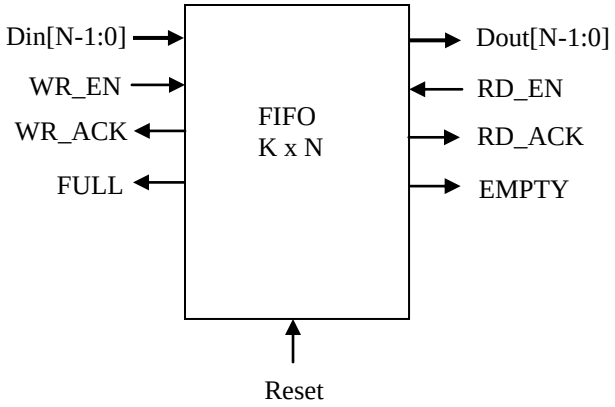
(i) Complete the Verilog model given below for modeling the Control Unit of the divider.

```
module  CU_DIV (output reg LDR, ClearR, LDQ, SetQ0, LDB, Shift, INC, CLRC, Ready, input Start, CLK, Reset, DGEZ, CEN);
// State Codes
parameter  s0=0, s1=1, s2=2, s3=3, s4=4;
reg [2:0] PS, NS;

always @(posedge CLK, posedge Reset)
    if (Reset==1) PS <= s0;
    else PS <= NS;

always @ (PS, Start, DGEZ, CEN) begin
    LDR=0; ClearR=0; LDQ=0; SetQ0=0; LDB=0;
    Shift=0; INC=0; CLRC=0; Ready=0;
    NS=s0;
case (PS)
s0:  if (Start) begin
            LDQ=1; LDB=1; CLRC=1;  ClearR=1;
            NS = s1;
    end
    else NS = s0;
s1:  begin
            Shift=1;
            NS = s2;
    end
s2:  begin
            if (DGEZ) begin
                    LDR=1; SetQ0=1; INC=1;
            end else INC=1;
            NS = s3;
    end
s3:  begin
    if (CEN)
            NS = s4;
```

```
        else
                NS = s1;
         end
    s4: begin
                Ready=1;
                NS = s4;
       end
    default: NS=s0;
    endcase
    end
    endmodule
```

[16 Points]

(Q4) It is required to model an **Asynchronous FIFO (First-In-First-Out) memory queue**. The FIFO has a parametrizable memory depth of upto **K** loctions with a parametrizable data width of **N** bits. The FIFO interface is given below:



The Reset signal is an **Asynchronous reset** and when set to 1 it will consider the content of FIFO empty and set the EMPTY flag to one and all other flags to 0. A handshaking mechanism is used for both writing and reading from the FIFO using the signal WR_EN, WR_ACK, RD_EN and RD_ACK. When WR_EN=1, the data in DIN input will be written to the available location in the FIFO as long as the FULL signal is not set to 1. If the FIFO is FULL the request is ignored and not acknolwdged. It is assumed that the WR_EN signal will remain 1 until a WR_ACK is set to 1 by the FIFO. After that, the WR_EN signal will go to 0. A similar handshaking mechanism is used for reading from the FIFO. When RD_EN=1, the data in the proper location will be output to DOUT as long as the EMPTY signal is not set to 1. Then, the RD_ACK signal is set to 1. If the FIFO is EMPTY the request is ignored and not acknolwdged. It is very important to note that the **FIFO should be able to read and write simultaneously** if needed.
A read pointer is used to point at the location to be read from and a write pointer is used to point at the location to be written to. A counter is used to keep track of when the FIFO is full or empty.
Part of the FIFO mdoule is given below and you need to complete the following missing parts:

(i) Declare all the required variables for modeling the FIFO.

(ii) The Read and Counter processes are given to you. Describe the write process for writing to the FIFO.

```
module FIFO #(parameter N=8, K=4, KS=2)
(output reg [N-1:0] DOUT,
output reg WR_ACK, RD_ACK, FULL, EMPTY,
input [N-1:0] DIN,
input Reset, WR_EN, RD_EN);

// Define required variables here
```

```
reg [KS-1:0] ReadP, WriteP;
reg [KS:0]  Count;
reg INC, DEC, INCA, DECA;

reg [N-1:0] FIFO[K-1:0];


// Read Process

       always @ (Reset, RD_EN, Count) begin
          if (Reset) begin
            RD_ACK = 0;
            ReadP = 0;
            EMPTY = 1;
            DEC = 0;
          end
          else begin
                    if (RD_EN)
                              if (Count > 0) begin
                                        DOUT = FIFO[ReadP];

                                        if (ReadP < K-1)
                                                  ReadP = ReadP + 1;
                                        else
                                                  ReadP = 0;

                                        RD_ACK = 1;
                                        DEC = 1;
                                        @ (posedge DECA)    DEC = 0;

                                        @ (negedge RD_EN)  RD_ACK = 0;
                              end
                    if (Count==0)
                              EMPTY = 1;
                    else
                              EMPTY = 0;
                    end
     end // Read Process

// Write Process

          always @ (Reset, WR_EN, Count) begin
       if (Reset) begin
            WR_ACK = 0;
            WriteP = 0;
            FULL = 0;
            INC = 0;
       end
       else         begin
                    if (WR_EN) begin
                              if (Count < K) begin
```

```
                                FIFO[WriteP] = DIN;

                                if (WriteP < K-1)
                                        WriteP = WriteP + 1;
                                else
                                        WriteP = 0;

                                WR_ACK = 1;
                                INC = 1;
                                @ (posedge INCA)     INC = 0;

                                @ (negedge WR_EN)WR_ACK = 0;
                        end

                end
                if (Count==K)
                        FULL = 1;
                else
                        FULL = 0;
        end
    end // Write Process


// Counter Process

        always @ (Reset, INC, DEC) begin
                if (Reset==1) begin
                        Count = 0;
                        INCA = 0;
                        DECA = 0;
                end
                else begin
                        if (INC) begin
                                Count = Count + 1;
                                INCA = 1;
                                @ (negedge INC) INCA = 0;
                        end
                        if (DEC) begin
                                Count = Count - 1;
                                DECA = 1;
                                @ (negedge DEC) DECA = 0;
                        end
                end
        end // Counter Process

endmodule
```