2010 - 2 - midterm 2015/04/02 박경린

1. a. $63 \cdot 2^{-6} - 1$ ⇒ integer 6bit

sign ⇒ 1bit (MSB)

0.___ ⇒ decimal 3bit

1 bit for preventing overflow

\+ (carry, bit extension)

---

11 bit ⇒ # of full adders are required

ex)

$$\begin{array}{r} 0\,111111 \quad 6bit+1bit \\ +\ 0\,111111 \quad 6bit+1bit \\ \hline 0\,1000000 \quad 1bit+7bit \end{array}$$

sign

b.

To calculate to n^{th} decimal point ⇒ $x2^n \div 2^n$

$(23.n6)_{10} = 23 + 0.n6 = 23 + 6.08 \div 2^3 = 23 + 6 \div 2^3$

$= (00010111.110)_2$ (∵ 11bit)

$(34.8n2)_{10} = 34 + 0.8n2 = 34 + 6.4n2 \div 2^3 = 34 + n \div 2^3$

$= (00100010.111)_2$ ↓ 2's complement

$(-34.8n2)_{10} = (11011101.001)_2$

$(31.25)_{10} = 31 + 0.25 = 31 + 2 \div 2^3$

$= (00011111.010)_2$ ↓ 2's complement

$(-31.25)_{10} = (11100000.110)_2$

$(56.125)_{10} = 56 + 0.125 = 56 + 1 \div 2^3$

$= (00111000.001)_2$

:) $23.n6 - 34.8n2 = -11.112$

11 of full adders (11bit)

$$\begin{array}{r} 00010111.110 \quad (23.n6)_{10} \\ +\ 11011101.001 \quad (-34.8n2)_{10} \\ \hline (1)11110100.111)_2 \end{array}$$

↓ 2's complement

$(00001011.001)_2$

$= 11.125$

∴ $(11101100.111)_2 = -11.125$

It has discrepency with calculation in decimal and binary

This is because During the conversion from decimal to binary decimal number were rounded to integers, and 3 bit binary decimal number only can be represented by 0.125 size

:) $-31.25 - (-56.125)$

$= -31.25 + 56.125 = 24.875$

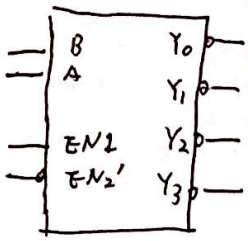
$$\begin{array}{r} 11100000.110 \quad (-31.25)_{10} \\ +\ 00111000.001 \quad (+56.125)_{10} \\ \hline (1)00011000.111 \end{array}$$

discard end carry

$(00011000.111)_2 = (24.875)_{10}$

Scanned with CamScanner

2.



2x4 decoder (complement) with EN1, EN2 switch

Decoder works only when EN1 = 1, EN2' = 0

Truth table (B is MSB)

EN1	EN2'	B	A	Y0	Y1	Y2	Y3
0	x	x	x	1	1	1	1
0	1	x	x	1	1	1	1
1	0	0	0	0	1	1	1
1	0	0	1	1	0	1	1
1	0	1	0	1	1	0	1
1	0	1	1	1	1	1	0

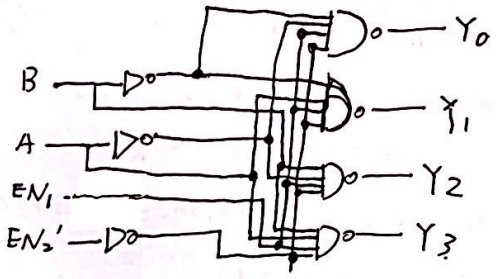
don't care condition

$Y_0 = (EN_2 (EN_2')' B' A')'$

$Y_1 = (EN_1 (EN_2')' B' A)'$

$Y_2 = (EN_1 (EN_2')' B A')'$

$Y_3 = (EN_1 (EN_2')' B A)'$



※ Verilog

1. Gate-level modeling

```
module dec_2(
  output [0:3] Y,
  input A, B, EN1, EN2_Prime
);
wire A_not, B_Not, EN2_Prime_not;

not
   G0 (A_not, A),
   G1 (B_not, B);
not (EN2_Prime_not, EN2_Prime); // optional gate instance name (primitives)
nand
   G2 (Y[0], B_not, A_not, EN1, EN2_Prime_not),
   G3 (Y[1], B_not, A, EN1, EN2_Prime_not),
   G4 (Y[2], B, A_not, EN1, EN2_Prime_not),
   G5 (Y[3], B, A, EN1, EN2_Prime_not);
end module
```

Scanned with CamScanner

2. Data-flow modeling

```
module dec_2 (
  output [0:3] Y,
  input  B, A, EN1, EN2_prime
);

  assign  Y[0] = ~( ~B & ~A &, EN1, & ~EN2_Prime),
          Y[1] = ~(~B & A & EN1 & ~EN2_prime),
          Y[2] = ~( B & ~A & EN1 & ~EN2_prime),
          Y[3] = ~( B & A & EN1 & ~EN2_prime);

end module
```

3. Behavioral modeling

```
module dec_2 (
  output reg [0:3] Y,
  input  B, A, EN1, EN2_prime
);

  always @ ( B, A, EN1, EN2-Prime)
    case ( { EN1, EN2-Prime, B, A} )
        4'b 1000 :  Y = 4'b 0111;
        4'b 1001 :  Y = 4'b 1011;
        4'b 1010 ;  Y = 4'b 1101;
        4'b 1011 :  Y = 4'b 1110;
        default  :  Y = 4'b 1111;
    endcase
  end module
```

3. $f(x, y, z) = x'y'z + y'z + xz + xy'z$

a. input : x, y, z
   output : f

b. literal - a variable or its complement
   10개

c. product term - one or more literals connected by AND(·)
   $x'y'z$, $y'z$, xz, $xy'z$

   4개

d. canonical sop - sum of minterms

※ minterm - a product term that includes all the variable either complement or not

Scanned with CamScanner

Solution 1

$x'y'z + y'z + x'z + xy'z'$

$= x'y'z + (x+x')y'z + x'(y+y')z + xy'z'$ ($\because x+x'=1$)

$= x'y'z + xy'z + x'y'z + x'yz + xy'z + xy'z'$

$= x'y'z + xy'z + x'yz + xy'z'$ ($\because x+x=x$)

Solution 2

$x'y'z \Rightarrow 001 \Rightarrow 1$

$xy'z' \Rightarrow 100 \Rightarrow 4$

$y'z \Rightarrow \{001, 101 \Rightarrow 1, 5$

$x'z \Rightarrow \{001, 011 \Rightarrow 1, 3$

$f = \Sigma(1,3,4,5)$

$= x'y'z + x'yz + xy'z' + xy'z$

x \ yz	00	01	11	10
0	0	1	1*	0
1	1*	1	0	0

e. minterm ($\because$(d)- definition)

$x'y'z,\ xy'z,\ x'yz,\ xy'z'$

g. canonical pos - product of maxterm

$f = \Sigma(1,3,4,5) = \Pi(0,2,6,7)$

$= (x+y+z)(x+y'+z)(x+y'+z)(x'+y'+z')$

h. Using K-map in (d)

marked * $\Rightarrow$ EPI : $x'z, xy'$

PI : $x'z, xy', y'z$

※ $f = x'y'z + y'z + x'z + xy'z'$

$= x'z(1+y') + xy'(z+1)$

$= x'z + xy'$

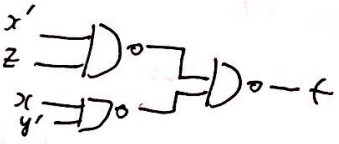
$f(x,y,z) =$ EPI (+ PI)

$= x'z + xy'$

; Following h EPI : $x'z, xy'$

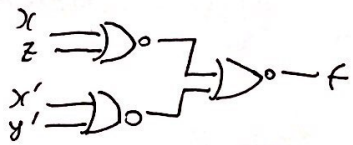
PI : $x'z, xy', y'z$

; $f = x'z + xy' = ((x'z + xy')')' = ((x'z)'.(xy')')'$



k. $f = (x+z)(x'+y')$ ($\because$ k-map o)

$= ((x+z)' + (x'+y')')'$



Scanned with CamScanner

4. Latch → output changes as input changes (non-clocked memory)
Flip-Flop → output changes as clockedge (clocked memory)
(shortly after)
(clock transition)

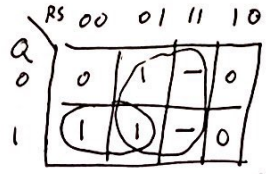
※ 4 kinds of Flip-Flop (RS, D, JK, T), characteristic table, K-map
excitation table, verilog (Behavioral modeling) (asynchronous reset)

(assume Flip-Flop is positive triggered

RS Flip-Flop

CLK	RS	Q(t+1) (characteristic table)
0	--	Q
1	--	Q
↑	00	Q (no change)
↑	01	1 (set)
↑	10	0 (Reset)
↑	11	- or? (illegal input) (unpredictable)

K-map



Q(t+1) = S + R'Q

excitation table

Q(t)	Q(t+1)	S R	
0	0	0 x	no change 00
0	1	1 0	Reset 01
1	0	0 1	– set 10
1	1	x 0	– no change 00, set 10

```
module RSFF (
  output reg Q,
  input S, R, clk, rst
);
always @ (posedge clk, negedge rst)
  if (~rst) Q <= 1'b0; // if(rst==0)
  else case({S,R})
    2'b00: Q <= Q;
    2'b01: Q <= 1'b0;
    2'b10: Q <= 1'b1;
    2'b11: Q <= 1'bZ; // High Impedence
  endcase
endmodule
```

or

```
always @ (posedge clk, negedge rst)
if (~rst) Q <= 1'b0;
else begin
  if(S!=R) begin
  // S=1,R=0 or S=0,R=1
     Q <= S;
  end
  else if(S==1 && R==1) begin
     Q <= 1'bZ;
  end
end
```

Scanned with CamScanner

D - Flip Flop

Clk	D	Q(t+1) (characteristic table)
0	–	q
1	–	q
↑	0	0 (Reset)
↑	1	1 (set)

K-map

Q \ D	0	1
0		1
1		1

$Q(t+1) = D$

excitation table

Q(t)	Q(t+1)	D
0	0	0
0	1	1
1	0	0
1	1	1

```
module DFF (
  output reg Q,
  input D, clk, rst
);
  always @(posedge clk, negedge rst)
    if(~rst) Q <= 1'b0;
    else     Q <= D;
endmodule
```

JK - Flip Flop

characteristic table

Clk	J	K	Q(t+1)
0	–	–	q
1	–	–	q
↑	0	0	q (no change)
↑	0	1	0 (Reset)
↑	1	0	1 (set)
↑	1	1	Q' (complement)

K-map

Q \ JK	00	01	11	10
0	0	0	1	1
1	1	0	0	1

$Q = JQ' + K'Q$

excitation table

Q(t)	Q(t+1)	J	K	
0	0	0	x	no change 00
0	1	1	x	reset 01
1	0	x	1	set 10
1	1	x	0	complement 11
				Reset 01
				complement 11
				no change 00
				set 10

```
module JKFF (
  output reg Q,
  input J, K, clk, rst
);
  always @(posedge clk, negedge rst)
    if(~rst) Q <= 1'b0;
    else case ({J,K})
      2'b00 : Q <= Q;
      2'b01 : Q <= 1'b0;
      2'b10 : Q <= 1'b1;
      2'b11 : Q <= ~Q;
    endcase
endmodule
```

```
wire JK;
assign JK = (J & ~Q) | (~K & Q);
DFF D1(Q, JK, clk, rst);
```

Scanned with CamScanner

T-Flip Flop

Characteristic table

CLK	T	Q(t+1)	
0	=	Q	
1	=	Q	
↑	0	Q	(no change)
↑	1	Q'	(complement)

k-map

Q \ T	0	1
0	0	1
1	1	0

$Q(t+1) = TQ' + T'Q$

$= T \oplus Q$

Excitation Table

Q(t)	Q(t+1)	T
0	0	0
0	1	1
1	0	1
1	1	0

```
module TFF(
  output reg Q;
  input T, clk, rst
);
  wire D;
  assign D = T^Q;
  DFF DI(Q, D, clk, rst);
end module.
```

```
JKFF(Q, T, T, clk, rst);
```

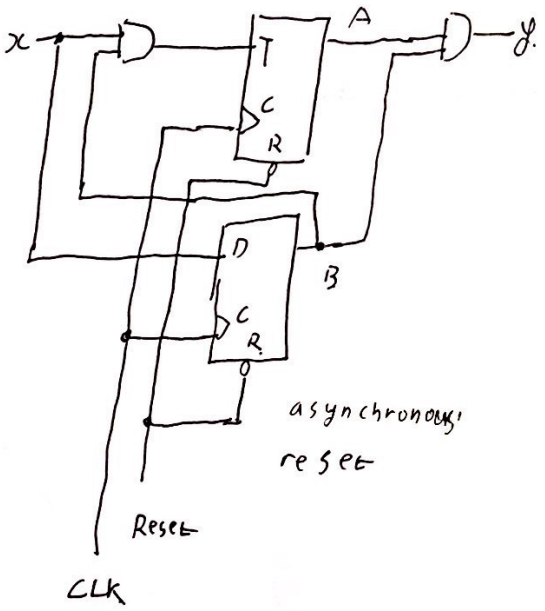
```
always @(posedge clk, negedge rst)
  if(~rst) Q <= 1'b0;
  else begin
    if(~T) Q <= ~Q;
  end
```

5. Circuit Analysis

circuit diagram → state equations → state table → state diagram

state equations

$y = AB$ (moore machine output is a function of the present state only)



$T = Bx$

$D = x$

$B(t+1) = x$

$A(t+1) = A(t) \oplus T = A(t)(Bx)' + A'(t)Bx$

$= A(t)B'(t) + A(t)x' + A'(t)B(t)x.$

Scanned with CamScanner

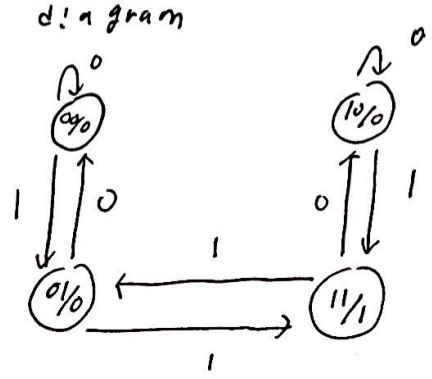
※ A⊕B ⇒ Xor is 1 only when the number of 1 input is odd

A⊕BX =1 { A=0 BX=1 ⇒ A B X: 0 1 1
A=1 BX=0 ⇒ 1 0 0
1 0 1
1 1 0

State table

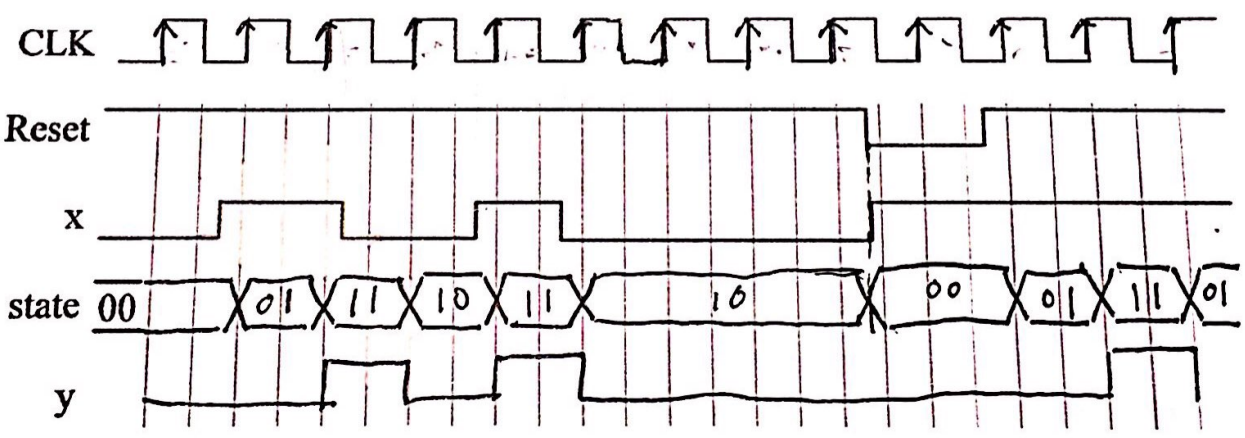
Present state		input	next state		output
A(t)	B(t)	x	A(t+1)	B(t+1)	y
0	0	0	0	0	0
0	0	1	0	1	0
0	1	0	0	0	0
0	1	1	1	1	0
1	0	0	1	0	0
1	0	1	1	1	0
1	1	0	1	0	1
1	1	1	0	1	1

State diagram



positive edge triggered and asynchronous reset

moore model



Scanned with CamScanner

※ verilog moore machine.

```
module moore_1 (
  output reg y,
  input x, clk, rst
);
  reg [1:0] state;
  parameter S0=2'b00, S1=2'b01, S2=2'b10, S3=2'b11;

  always @ (posedge clk, negedge rst)
    if (~rst) state <= S0;
    else case (state)
      S0: if (~x) state<=S0; else state<=S1;
      S1: if (~x) state<=S0; else state<=S3;
      S2: if (~x) state<=S2; else state<=S3;
      S3: if (~x) state<=S2; else state<=S1;
    endcase
    assign y = (state==S3);
endmodule
```

Scanned with CamScanner