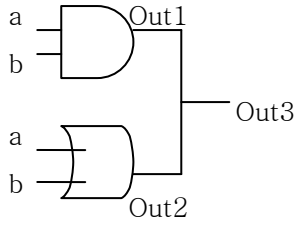
디지털 시스템 설계 중간 고사 정답

2008년 1학기

채점 조교: 김제선, 정진수, 진용석

1. [5점] (a) 아래의 회로를 verilog program한 경우 out1, out2, out3 신호의 값은 무엇인가? 아래 빈칸을 채우시오.



a	x	0	x	0	1	0	z
b	1	0	0	1	1	x	x
Out1							
Out2							
Out3							

Answer)

a	x	0	x	0	1	0	z
b	1	0	0	1	1	x	x
Out1	x	0	0	0	1	0	x
Out2	1	0	x	1	1	x	x
Out3	x	0	x	x	1	x	x

2. [15점] Design a Verilog module of a frequency divider that divides clock by 11 and asserts its output for one cycle, that is, forms an output pulse every 11^{th} pulse of clock. The asynchronous reset signal is active-low and drives the output to '1'.

Answer)

```
module problem2 (resetn,clock_input,clock_output);
input          resetn, clock_input;
output         clock_output;
reg            clock_output;
reg [3:0]     clock_counter;
always @(negedge resetn or posedge clock_input)
begin : clock_output_generation
    if(!resetn)
    begin
        clock_output <= 1'b1;
    end
    else
    begin
        if (clock_counter == 4'ha) clock_output <= 1'b1;
        else clock_output <= 1'b0;
    end
end
always @(negedge resetn or posedge clock_input)
begin : clock_input_pulse_counter
    if(!resetn)
    begin
        clock_counter <= 4'h0;
    end
    else
    begin
        if (clock_counter == 4'ha) clock_counter <= 4'h0;
        else clock_counter <= clock_counter + 4'h1;
    end
end
endmodule
```

3. [10점] Using a cycle behavior (***always***), write a model of a transparent latch having active high *enable* and active low *set* and *reset*. The action of reset is to drive the output of the latch to 0.

Answer)

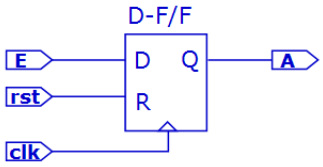
```
module transparent_latch(
        in_data,
        enable,
        set,
        reset,
        out_data
);

 input in_data, enable, set, reset;
ouput out_data;
    reg out_data;

always@(enable or set or reset or in_data)
begin
        if(!reset) out_data <= 0;
        else if(!set) out_data <= 1;
        else if(enable) out_data <= in_data;
end

endmodule
```

4. 아래의 Verilog module을 합성하면 생성되는 회로를 그리시오.

(a) [5점]

```
module problem4a (A, E, clk, rst);
        output   A;
        input    E;
        input    clk, rst;
        reg      A, B, C, D;

        always @ (posedge clk or posedge rst) begin
                if (rst) begin A = 0; B = 0; C = 0; D = 0; end
                else begin
                        D = E;
                        C = D;
                        B = C;
                        A = B;
                end
        end
endmodule
```

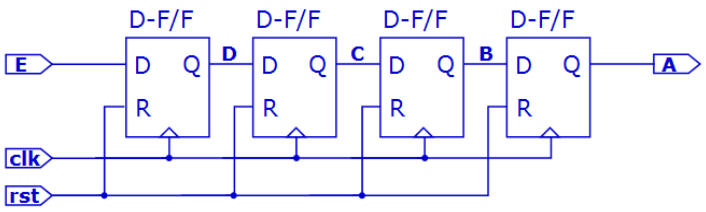
Answer)



(b) [5점]

```
module problem4b (A, E, clk, rst);
    output          A;
    input           E;
    input           clk, rst;
    reg             A, B, C, D;
  always @ (posedge clk or posedge rst) begin
    if (rst) begin A <= 0; B <= 0; C <= 0; D <= 0; end
    else begin
            D <= E;
            C <= D;
            B <= D;
            A <= B;
          end
    end
  endmodule
```

Answer)



5. 아래의 Verilog module이 어떠한 기능을 수행하는지 설명하시오.

(a) [7점]

```
module problem5a (index_value, A_word, trigger);
 output [3: 0]       index_value;
  input   [15: 0]    A_word;
  input              trigger;
  reg     [3: 0]     index_value;

  always @   (trigger)
    begin: search_for_1
      index_value = 0;
      for (index_value = 0; index_value <= 15; index_value = index_value + 1)
        if (A_word[index_value] == 1) disable search_for_1;
    end
endmodule
```

Answer)

Trigger가 변하는 시점에 A_word의 LSB에서 MSB 방향으로 1을 만날 때까지 0의 개수를 센다. 즉, index_value는 LSB로부터 최초의 1의 bit 위치를 나타낸다. 예를 들어 A_word가 16'b0000_0000_0100_0000 이었다면 index_value는 6이 된다. 그러나 A_word가 16'b0000_0000_0000_0000일 때 무한 loop을 돌게 된다. 이는 Index_value가 4bit이기 때문에 발생한다. Index_value가 15일 때 if문을 수행하고 index_value를 1 증가시키면 다시 0이 되기 때문이다.

(b) [8점]

```
module problem5b (sum, data, clk, reset);
  output                  [5: 0]      sum;
  input          [3: 0]   data;
  input                   clk, reset;
  reg                     sum;

always @ (posedge clk) begin:   add_loop
    if (reset) disable add_loop;                          else sum <= data;
      @ (posedge clk) if (reset) disable add_loop;        else sum <= sum + data;
        @ (posedge clk) if (reset) disable add_loop;
                else begin sum <= sum + data; sum <= sum + data; end
          @ (posedge clk) if (reset) disable add_loop;
                else begin sum <= sum + data; sum <= sum + data; sum <= sum + data; end
  end
endmodule
```

Answer)

1. clk의 rising_edge에서 data를 sum에 저장한다.

2. 이후 3 cycle 동안 sum에 data를 더한다.

Problem5b는 1,2번을 반복하면서 4개 data의 합을 4 cycle에 한번씩 반복 계산한다. reset 입력이 들어오면 계산을 중단하고 1부터 다시 시작한다.

6. [5점] Explain why the code fragment shown below will execute endlessly, and recommend an alternative description.

```
reg [3:0] K
for (K=0; K<=15; K=K+ 1) begin
…
end
```

Answer)

K는 4bit이므로 0~15 사이의 값만 가질 수 있다. K가 15가 된 후 다음 루프에서는 overflow가 발생하여 다시 0이 된다. 따라서 주어진 for 구문의 조건문을 항상 만족하므로 for 문이 끝나지 않게 된다.

이를 바로 잡기 위해서는

reg [4:0] K 로 선언하거나

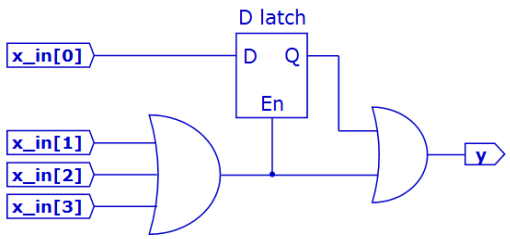
for(K=0; K<15; K=K+1) begin 등으로

원래 코드의 의도에 따라 수정하면 된다.

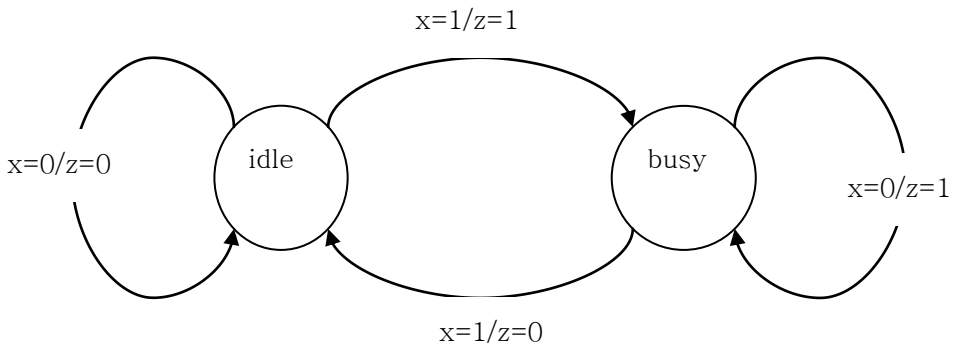
7 [15점] 아래의 Verilog module을 합성하면 생성되는 회로를 그리시오.

```
module problem7 (y, x_in);
   parameter          word_length = 4;
   output             y;
   input              [word_length - 1: 0]          x_in;
   reg                y;
   integer            k;

   always @    (x_in[3:1])
      begin: check_for_1
         y = 0;
         for (k = 0; k <= word_length -1; k = k+1)
            if (x_in[k] == 1)
               begin
                  y = 1;
                  disable check_for_1;
               end
      end
endmodule
```

Answer)



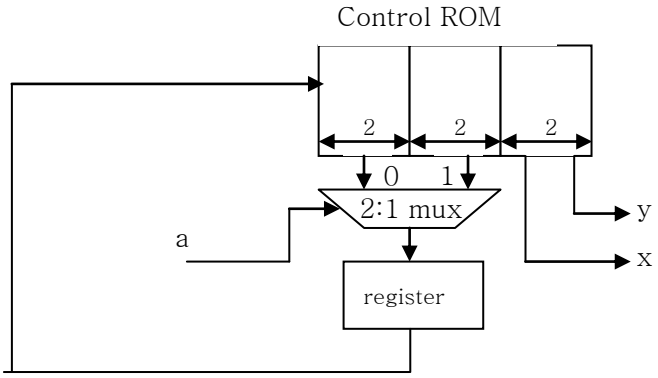
8 [10점]. 아래의 state diagram으로 표현되는 finite state machine을 Verilog로 프로그래밍하시오. 이 FSM의 입력은 x 이고, 출력은 z이다.



Answer)

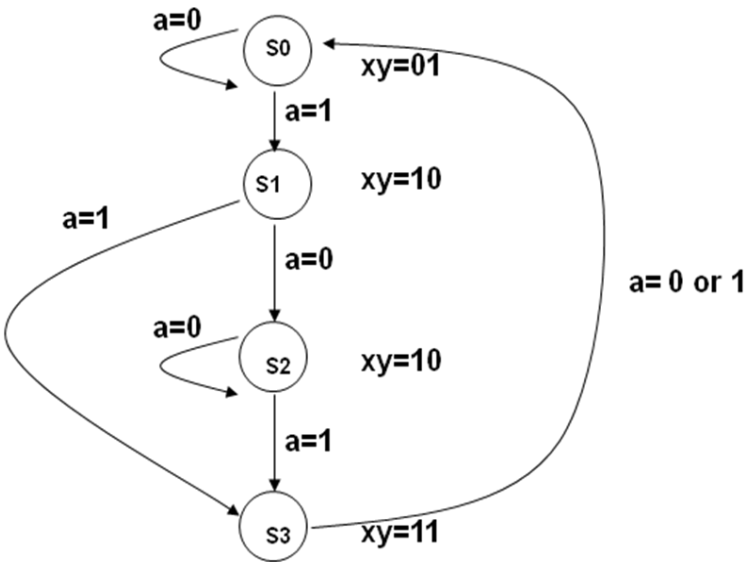
```
module problem8 (resetn, clock, x, z);
input     resetn,clock,x;
output    z;
reg z;
reg current_state,next_state;
parameter IDLE = 0, BUSY = 1;

always @(negedge resetn or posedge clock)
begin : current_state_decision
      if (!resetn) current_state <= IDLE;
      else current_state <= next_state;
end

always @(current_state,x)
begin : next_state_and_output
      case (current_state)
            IDLE      :
                  if (x==1) begin    next_state <= BUSY; z <= 1'b1;    end
                  else begin    next_state <= IDLE; z <= 1'b0;    end
            BUSY      :
                  if (x==1) begin    next_state <= IDLE; z <= 1'b0;    end
                  else begin    next_state <= BUSY; z <= 1'b1;    end
            default :
                  begin    next_state <= IDLE; z <= 1'b0;    end
   endcase
end
endmodule
```

9 (a) [10점]아래 그림은 마이크로프로그래밍 control unit 구조 및 ROM의 content를 보여준다. 아래 그림이 구현하는 control unit을 Finite State Machine으로 표현하시오. 이 Finite state machine의 입력신호는 'a' 이고, 출력 신호는 'x' 및 'y' 이고, state 이름은 임의로 사용하면 됨.



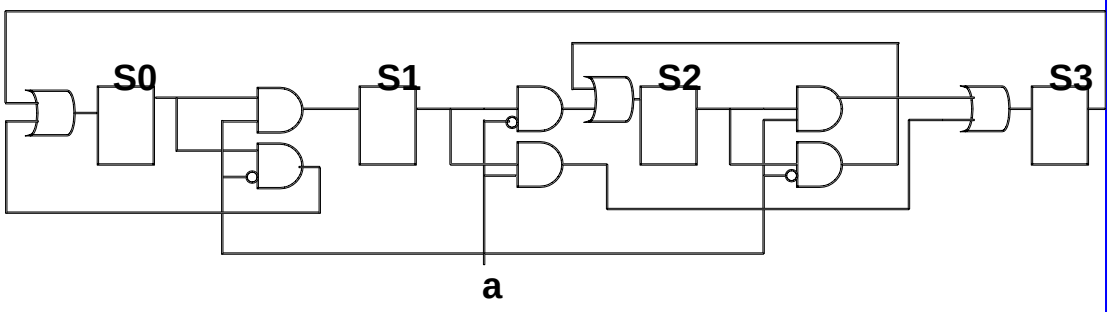
ROM address	ROM contents		
00	00	01	01
01	10	11	10
10	10	11	10
11	00	00	11

Answer)



(b) [5점] 위의 controller를 hardwired 방식으로 구현한 회로를 그리시오.

Answer)



디지털 시스템 설계 기말고사 문제 및 정답

2008년 1학기

1. [10점] (a) Binary SD number system에서 radix는 2이고, 각 digit에서 표현할 수 있는 가장 큰 수가 1인 경우 각 digit에서 두 자리수를 덧셈하는 방식을 아래의 Table P-1과 같이 할 수 있다. 이 경우 carry propagation이 발생할 수 있기 때문에, Table P-2와 같이 더하는 방식을 바꾸면 carry propagation을 방지할 수 있다. radix가 3이고 각 digit에서 표현할 수 있는 가장 큰 수가 2인 SD number system에서, 각 digit에서 두 자리수를 더하는 방식을 Table로 표현하시오. 이때, carry propagation이 발생하지 않도록 Table을 작성하시오. (채점 조교: 김영훈)

Table P-1

$x_i\, y_i$	00	01	$0\bar{1}$	11	$\bar{1}\bar{1}$
c_i	0	1	$\bar{1}$	1	$\bar{1}$
u_i	0	$\bar{1}$	1	0	0

Table P-2

$x_i\, y_i$	00	01	01	$0\bar{1}$	$0\bar{1}$	11	$\bar{1}\bar{1}$
$x_{i-1}\, y_{i-1}$	–	neither is $\bar{1}$	at least one is $\bar{1}$	neither is $\bar{1}$	at least one is $\bar{1}$	–	–
c_i	0	1	0	0	$\bar{1}$	1	$\bar{1}$
u_i	0	$\bar{1}$	1	$\bar{1}$	1	0	0

정답)

Xi Yi	**00**	**01**	**02**	**$0\bar{1}$**	**$0\bar{2}$**	**10**	**11**	**12**	**20**	**21**	**22**	**$2\bar{1}$**	**$2\bar{2}$**
Ci	**0**	**0**	**1**	**0**	$\bar{1}$	**0**	**1**	**1**	**1**	**1**	**1**	**0**	**0**
Ui	**0**	**1**	$\bar{1}$	$\bar{1}$	**1**	**1**	$\bar{1}$	**0**	$\bar{1}$	**0**	**1**	**1**	**0**

Xi Yi	**$\bar{1}$0**	**$\bar{1}$1**	**$\bar{1}$2**	**$\bar{1}\bar{1}$**	**$\bar{1}\bar{2}$**	**$\bar{2}$0**	**$\bar{2}$1**	**$\bar{2}$2**	**$\bar{2}\,\bar{1}$**	**$\bar{2}\,\bar{2}$**
Ci	**0**	**0**	**0**	$\bar{1}$	$\bar{1}$	$\bar{1}$	**0**	**0**	$\bar{1}$	$\bar{1}$
Ui	$\bar{1}$	**0**	**1**	**1**	**0**	**1**	$\bar{1}$	**0**	**0**	$\bar{1}$

- 총 15개의 조합이 존재 => 틀리거나 빠뜨린 항목에 대해 7점부터 1점씩 감점

(b) 위의 (a)에서 작성한 Table을 사용하여 0 1 1 0 + 0 2 1 1을 수행한 과정 및 결과를 보이시오.

정답)

	0	**1**	**1**	**0**
	0	**2**	**1**	**1**
0	**1**	**1**	**0**	**0**
	0	**0**	$\bar{1}$	**1**
0	**1**	**1**	$\bar{1}$	**1**

- 맞출 경우 3점

- 1021, 110$\bar{2}$도 정답 인정

2. [15점] (a) How many denormalized numbers are there in the short (32-bit) IEEE format, and what is their range ? (채점 조교: 현경환)

정답) E = 0, f $\neq$0

Total number of normalized numbers = $(2^{23} - 1) \times 2 = 2^{24} - 2$

Min^{+}: $0.00000000000000000000001 \times 2^{-126} = 2^{-23} \times 2^{-126} = 2^{-149}$

Max^{+}: $0.11111111111111111111111 \times 2^{-126} = (1 - 2^{-23}) \times 2^{-126}$

(b) How many normalized numbers with fixed exponent 1 (including bias) are there in the short (32-bit) IEEE format, and what is their range ?

정답) E = 1

Total number of normalized numbers = $2 * 2^{23} = 2^{24}$

Min^{+}: $1.00000000000000000000000 \times 2^{-126} = 2\text{-}126$

……

Max^{+}: $1.11111111111111111111111 \times 2^{-126} = (2 - 2^{-23}) \times 2^{-126} = (1\text{-}2^{-24}) \times 2^{-125}$

채점기준

숫자 개수 구하는 항목: 각 3점 * 2

숫자 범위 구하는 항목: 각 4점 * 2

기본점수: 1점

기본점수 (1점)

- **기본적으로 필요한 이론 설명이 있을 경우 부여**
- **Short (32-bit) IEEE format에 대한 개념 언급 시**
- **(De)normalized number에 대한 개념 언급 시**
- **잘못된 설명 있을 경우 점수 없음**

숫자 개수 (3점)

- **Sign bit 고려 (x2) 있어야 함**

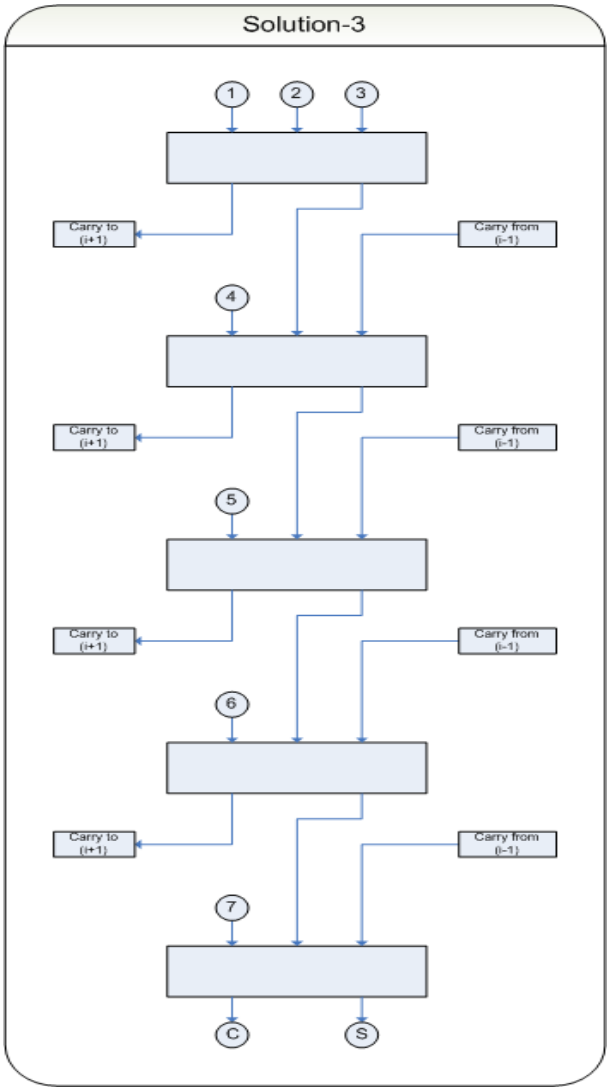
숫자 범위 (4점)

- **최대/최소 각각 2점**
- **단, 풀이 내용이 틀릴 경우 감점**
- **근사값 허용 안 함**

(b)에서 E=1인 경우 E_{true}를 -126이 아니라 1로 한 경우도 정답 처리함

3. [10점] (a) Design a (7;2) compressor with 7 inputs, 2 outputs that has 4 input carries from position (i-1) and generates 4 output carries to position (i+1). Use (3,2) counters and make sure that no long carry-propagation chains are generated. (채점 조교: 김성윤)

정답)



- **위와 같이 (i+1) position의 carry 4개가 chain을 만들지 않고 연결되어 있으면 10점. (위의 그림과 다르더라도 설명된 조건을 충족하면 10점)**
- **(i+1) position의 carry 4개를 이용했으나, chain이 생성되거나 연결에 실수가 있는 경우 7점.**
- **(i+2) position의 carry를 이용한 경우 chain없이 동작하면 5점**
- **(i+2) position의 carry를 이용했으나, chain이 생성되거나 연결에 실수가 있는 경우 3점**
- **(3,2) counter를 5개 사용하지 않은 경우 0점**
- **(i+3) position의 carry가 생성된 경우 0점**
- **Position이 같지 않은 carry를 하나의 (3,2) counter에 연결한 경우 0점**

4. [15점] (a) Write down the rules for a radix-8 modified Booth's algorithm (Table 형식으로 작성하면 됨). (채점 조교: 김응섭)

정답)

bi	bi-1	bi-2	bi-3	xi	xi-1	xi-2	
0	0	0	0	0	0	0	0
0	0	1	0	0	0	1	+A
0	1	0	0	0	1	0	+2A
0	1	1	0	0	1	1	+3A
1	0	0	0	1′	0	0	-4A
1	0	1	0	0	1′	1′	-3A
1	1	0	0	0	1′	0	-2A
1	1	1	0	0	0	1′	-A
0	0	0	1	0	0	1	+A
0	0	1	1	0	1	0	+2A
0	1	0	1	0	1	1	+3A
0	1	1	1	1	0	0	+4A
1	0	0	1	0	1′	1′	-3A
1	0	1	1	0	1′	0	-2A
1	1	0	1	0	0	1′	-A
1	1	1	1	0	0	0	0

another form

bi	bi-1	bi-2	bi-3	xi	xi-1	xi-2	
0	0	0	0	0	0	0	0
0	0	0	1	0	0	1	+A
0	0	1	0	0	0	1	+A
0	0	1	1	0	1	0	+2A
0	1	0	0	0	1	0	+2A
0	1	0	1	0	1	1	+3A
0	1	1	0	0	1	1	+3A
0	1	1	1	1	0	0	+4A
1	0	0	0	1′	0	0	-4A
1	0	0	1	0	1′	1′	-3A
1	0	1	0	0	1′	1′	-3A
1	0	1	1	0	1′	0	-2A
1	1	0	0	0	1′	0	-2A
1	1	0	1	0	0	1′	-A
1	1	1	0	0	0	1′	-A
1	1	1	1	0	0	0	0

채점기준: 1 row 당 0.5점 -> 총 점: 8점 (.5는 반올림)

(b) 아래의 왼쪽 예에서 6-bit 2's complement number인 010110과 001011를 Radix-4 modified Booth algorithm을 사용하여 하여 곱하는 과정을 보인다. 이 과정에서는 negative partial product를 2's complement 연산을 통하여 구할 수 있다고 가정하였고, 또한 negative partial product의 sign bit extension을 줄였다. 위의 (a)에서 구한 radix-8 modified Booth's algorithm을 사용하는 경우 negative sign-bit extension을 막기 위한 scheme에 대해서 설명하시오. 이 scheme을 적용하여 아래의 example (010110 x 001111) 을 수행하는 과정 및 결과를 보이시오.

```
      0 1 0 1 1 0                0 1 0 1 1 0
    x 0 0 1 0 1 1              x 0 0 1 1 1 1
  0 1 1 0 1 0 1 0
1 0 0 1 0 1 0
1 1 0 1 1 0
0 0 1 1 1 1 0 0 1 0
```

1)3 점

Ss ssx xxx xxx

1 1Sx xxx xxx

Sx xxx xxx

(S: inverted sign-bit, s: sign-bit, x: bit={0, 1})

-> 말로 설명해도 됨.

2) 4 점

010 110

x 001 111

R 010 001 (1 점)

---

01 111 101 010 (1 점)

110 101 100 (1 점)

---

000 101 001 010 (1점)

5. [10점] Interrupt controller와 Timer의 기능에 대해서 각각 간단히 설명하시오. (채점 조교: 김제선)

정답)

Interrupt controller : 주변 장치에서 발생된 interrupt는 CPU로 바로 전달되지 않고 interrupt controller에게 전달되고, 이 interrupt controller는 발생된 interrupt 종류를 CPU 에 알려주고 CPU는 interrupt의 종류에 따라 interrupt를 발생시킨 장치를 제어한다. Prioritizing and propagating interrupt requests from internal or external devices to the IU. (interrupt controller은 interrupt를 CPU에 알려줄 뿐, 직접 처리하지 않음.)

Timer : Prescaler is decremented on each clock. When underflow, timer tick generated. Timer is decremented each time prescaler generates a timer tick. When underflow, interrupt generated. If the reload bit is set, reloaded with the value of the timer reload register.

6. [15점] A counter is said to enter an abnormal state if it enters a state that is not explicitly decoded in its next-state function. A self-correcting counter has the ability to recover from an abnormal state. Write a Verilog program modeling a self-correcting 3-bit Johnson counter using the next state 001_2 if the current state is in an abnormal state. (참조-Johnson counter: 000→001→011→111→110→100→000→...). (채점 조교: 진용석)

정답)

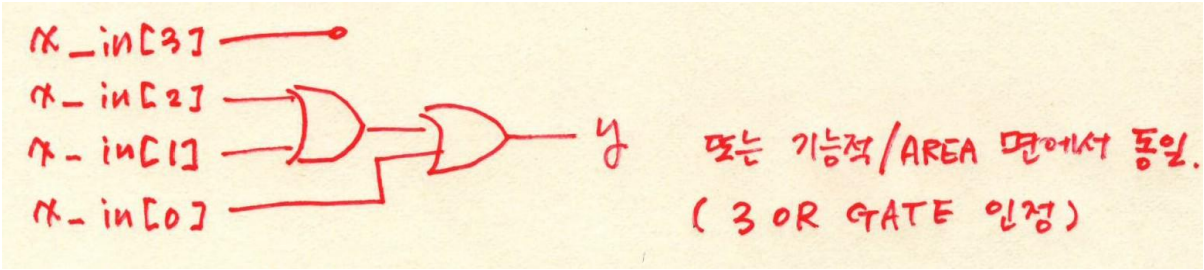
```
module self_correcting_johnson_counter(
        input CLK, RESETn,
        output reg count
);
always@(posedge CLK, negedge RESETn) begin
        if(!RESETn) begin
                count <= 3'b001;
        end else begin
                case(count)
                        3'b000: count <= 3'b001;
                        3'b001: count <= 3'b011;
                        3'b011: count <= 3'b111;
                        3'b111: count <= 3'b110;
                        3'b110: count <= 3'b100;
                        3'b100: count <= 3'b000;
                        default: count <= 3'b001;
                endcase
        end
end
endmodule
```

7 [15점] 아래의 Verilog module을 합성하면 생성되는 회로를 그리시오. (채점 조교: 주종환)

(a)

```
module prob7a (y, x_in);
   parameter          word_length = 4;
   output             y;
   input              [word_length - 1: 0] x_in;
   reg                y;

   always @   x_in begin
      y = 0;
      if (x_in[0] == 1) y=1;
      else if (x_in[1] == 1) y=1;
      else if (x_in[2] == 1) y=1;
   end
endmodule
```

(정답) 6점만점



감점:

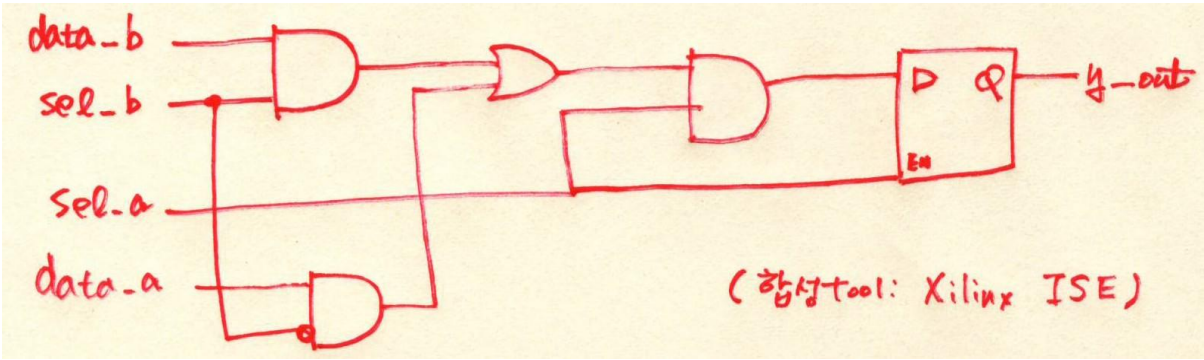
- **Input port인 x_in[3] 생략 (-1점)**
- **기능적으로 동일하지만 큰 Area Overhead (-3점~-5점)**
- **Latch 등 불필요한 하드웨어 합성 (-5점)**

기타: 기능적으로 전혀 오답 (0점)

(b)

```
module prob7b (y_out, sel_a, sel_b, data_a, data_b);
   output y_out;
   input  sel_a, sel_b, data_a, data_b;
   reg    y_out;

   always @   (sel_a or sel_b or data_a or data_b)

      case ({sel_a, sel_b})
         2'b10: y_out = data_a;
         2'b11: y_out = data_b;
      endcase

endmodule
```

(정답) 9점만점



기본점수:

- **sel_a가 0일때에, Latch의 입력을 끊어 output의 glitch 발생을 최소화하고 output 값을 유지하며, sel_b는 data_a, data_b의 MUXING을 정상적으로 수행함. (9점)**
- **sel_a가 0일때에, 이전의 output 값을 유지하며, sel_b는 data_a, data_b의 MUXING을 정상**

적으로 수행함 (8점)

감점:

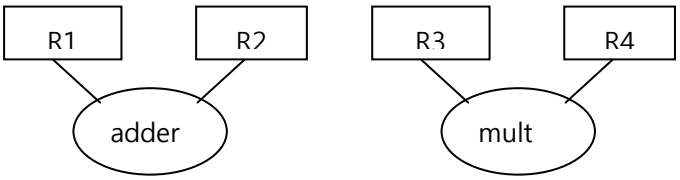
- Latch의 enable wire 연결 등의 실수 (-3점)
- 2:1 MUX대신 3state buffer 사용 (-3점)
- 하드웨어 기능은 동일하지만 불필요한 하드웨어 사용 (-2점)

부분점수: (위의 경우가 아닐 때)

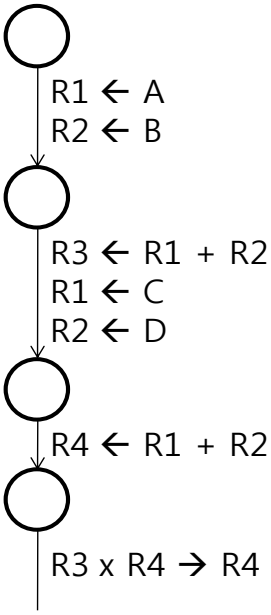
- Feedback 혹은 Latch 합성 (sel_a = 0일 때, 값유지)에 대한 개념이 있는 2:1 MUX (2점)
- 단지 2:1 MUX, latch 없음. (1점)

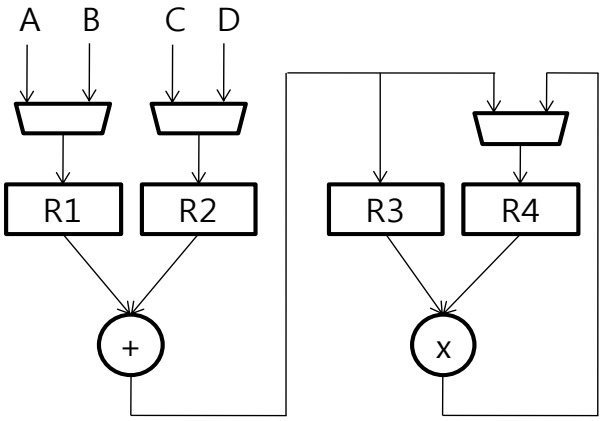
기능적으로 전혀 다른 동작, 혹은 동작 불가능함. (0점)

8 [10점] 아래의 그림처럼 1개의 adder와 1개의 multiplier가 주어졌다고 가정하자. 각각의 adder와 multiplier의 두개의 입력은 각각 register R1, R2 및 R3, R4에 연결되어 있다. 이 adder와 multiplier를 사용하여 연산 F=(A+B)*(C+D)을 수행하기 위한 data path 및 control unit을 설계하시오. F의 최종값은 register R4에 저장되게 하시오. (채점 조교: 이채은)



정답)





<채점 기준>

아래 조건들을 만족 시키지 못할 때 부분 감점

data path(5점):

1) **1 adder, 1 mult 만 사용할 것**
2) **Adder 에는 R1/R2 만 접근 가능. Mult 에는 R3/R4 만 접근 가능**
3) **Mux 나 tri-state buf 등으로 data 흐름을 충분히 논리적으로 그릴 것(Register 에 input or output 이 2개 이상씩 나가지 않도록)**
4) **최종 값은 R4에**

control unit(5점):

1) **Register 의 흐름 또는 control signal 들을 충분이 자세히 기술할 것**
2) **State or clock 을 고려해서 연산 수행. 즉, load 나 add/mult 등이 무조건 한번에 실행되는 '개념적인' 접근이 아니라 '실제적인' 설계를 고려할 것**

과도한 'redundant' 한 동작은 감점 대상