디지털 시스템 설계 및 실험 답안 **출제: 채수익**

중간고사 1

1.(5pts+5pts = 10pts)

A. (5pts)

```
module mux_4to1(x, a0, a1, a2, a3, select);
parameter n = 8;
input [1:0] select;
input [n-1:0] a0,a1,a2,a3;
output [n-1:0] x;
reg [n-1:0]x;//case 또는 if문 으로 짠 경우 x를 reg로 선언하지 않으면(-1pts)
always @(a0,a1,a2,a3,select)
    case (select)
        2'b00: x = a0;
          2'b01: x = a1;
          2'b10: x = a2;
          2'b11: x = a3;
          //default: x = 0; // using default to include all other possible cases.
      endcase//endcase를 쓰지 않았을 경우( -1pts)
endmodule
```

case문으로 짜지 않고 if,else 구문이나 assign을 이용하여 짜도(assign으로 짠 경우 reg x를 선언해서는 안된다.)상관 없음.

그 외 사소한 문법 오류 (-1pts)

B. (5pts)

```
module mux_16to1(x, a0, a1, a2, a3, a4, a5, a6, a7,
                            a8, a9, a10, a11, a12, a13, a14, a15, select);
parameter n = 8;
input [3:0] select;
input [n-1:0] a0, a1, a2, a3, a4, a5, a6, a7, a8, a9, a10, a11, a12, a13, a14, a15;
output [n-1:0] x;
wire [n-1:0]x0,x1,x2,x3;//wire 선언하지 않았거나 길이를 제대로 선언하지 않았을 경우(-1pts)
mux_4to1 m0(.x(x0), .a0(a0), .a1(a1), .a2(a2), .a3(a3), .select(select[1:0]));
//m0~m4 부분의 select[1:0]이 아닌 [3:0]으로 사용하였을 경우 (-1pts)
mux_4to1 m1(.x(x1), .a0(a4), .a1(a5), .a2(a6), .a3(a7), .select(select[1:0]));
mux_4to1 m2(.x(x2), .a0(a8), .a1(a8), .a2(a9), .a3(a11), .select(select[1:0]));
mux_4to1 m3(.x(x3), .a0(a12), .a1(a13), .a2(a14), .a3(a15), .select(select[1:0]));
mux_4to1 m4(.x(x), .a0(x0), .a1(x1), .a2(x2), .a3(x3), .select(select[3:2]));
//이 부분을 case 문을 이용하여 값을 전달하는 것은 상관 없지만 case문이나 if문 안에
mux_4to1의 모듈 연결 문이 들어있을 경우(컴파일 오류) (-2pts)
endmodule
```

2.(5pts+5pts+5pts+5pts=20pts)

A(5pts)

```
 module full_adder(sum, c_out, a, b, c_in);
 input a, b, c_in;
 output sum, c_out;
 wire w1,w2,w3,w4;
 assign w1 = a^b;
 assign w2 = a&b;
 assign w3 = a&c_in;
 assign w4 = b&c_in;
 assign sum = w1^c_in;
 assign c_out = w2|w3|w4;
 endmodule
//{c_in,s}=a+b+c_in; 과 같이 +로 짠 경우(-1pts)(+를 계산하기 위해 full adder안에 adder가 설계되므로)
//기타 c_in과 s에 제대로 된 결과가 들어가지 않는 논리식 일 경우 (-1pts)
module carry_lookahead(c_out, g, p, c_in);
parameter n = 4;
output [n-1:0] c_out;
input [n-1:0] g;
input [n-1:0] p;
input c_in;
assign c_out[0] = g[0]|(p[0]&c_in);
assign c_out[1] = g[1]|(p[1]&c_out[0]);
assign c_out[2] = g[2]|(p[2]&c_out[1]);
assign c_out[3] = g[3]|(p[3]&c_out[2]);
//약간의 오류가 있을 경우(-1pts)
//c_out을 한 비트라고 가정하고 c_cout에 cout[3]의 내용만 넣었을 경우 (-1pts)
//c_out계산 결과 대부분이 틀리거나 다른 값을 계산했을 경우 (-2pts)
//always문을 사용하였을 때 c_out에 reg 선언을 하지 않았을 경우 (-1pts)
endmodule
```

B. (5pts)

```
module RCA_adder(S, c_out, A, B, c_in);
parameter n = 4;
input [n-1:0] A, B;
input c_in;
output [n-1:0] S;
output c_out;
wire [2:0]carry;
//carry 값을 전달하는 wire 선언을 하지 않거나 reg로 선언한 경우 (-2pts)
```

```
//모듈의 연결을 always문장 안이나 initial 문장 안에 넣었을 경우 (-1pts)
full_adder m0(S[0], carry[0], A[0], B[0], c_in);
full_adder m1(S[1], carry[1], A[1], B[1], carry[0]);
full_adder m2(S[2], carry[2], A[2], B[2], carry[1]);
full_adder m3(S[3], c_out, A[3], B[3], carry[2]);
//c_cout값을 계산하지 않고 넘어간 경우(-1pts)
//기타 사소한 문법 오류(-1pts)
//말로만 설명한 경우(기본 1.5pts)
endmodule
```

C. (5pts)

```
module CLA_adder(S, c_out, A, B, c_in);
parameter n = 4;
input [n-1:0] A, B;
input c_in;
output [n-1:0] S;
output c_out;

wire[n-1:0] p,g,carry;
//앞에서 짠 carry_lookahead module을 사용하지 않고 여기서 다시 carry를 계산한 경우는 상관 없음
carry_lookahead M0(carry,g,p,c_in);
//각각 p, g, S 값의 계산이 틀렸거나 적지 않은 경우 각각 (-1pts)
assign p[0]=A[0]^B[0];
assign p[1]=A[1]^B[1];
assign p[2]=A[2]^B[2];
assign p[3]=A[3]^B[3];

assign g[0]=A[0]&B[0];
assign g[1]=A[1]&B[1];
assign g[2]=A[2]&B[2];
assign g[3]=A[3]&B[3];

assign S[0] = p[0]^c_in;
assign S[1] = p[1]^carry[0];
assign S[2] = p[2]^carry[1];
assign S[3] = p[3]^carry[2];

assign c_out = carry[3];
```

```
endmodule
```

D. (5pts)

```
module CLA_adder_16bit(S, c_out, A, B, c_in);
input [15:0] A, B;
input c_in;
output [15:0] S;
output c_out;

wire carry[3:0];
//B와 마찬가지의 채점기준을 사용하였고, C를 수정하여 p,g값을 받고 4bit CLA를 단순 연결이 아닌 carry lookahead 모듈을 사용하여 계산한 경우(+1pts)
CLA_adder M0(S[3:0],carry[0],A[3:0],B[3:0],c_in);
CLA_adder M1(S[7:4],carry[1],A[7:4],B[7:4],carry[0]);
CLA_adder M2(S[11:8],carry[2],A[11:8],B[11:8],carry[1]);
CLA_adder M3(S[15:12],carry[3],A[15:12],B[15:12],carry[2]);

assign c_out=carry[3];

endmodule
```

3.

verilog code에 대한 설명 (2pts +2pts)

내용에 A는 latch 또는 logic B는 Flip Flop과 비교하는 내용이 있거나 기타 비슷한 내용(sequential등과 같은 설명)이 있을 경우 (기본 4pts)

둘의 동작에 대한 설명만(enable, reset등) 있고 위 내용이 없을 경우 (기본 3pts)

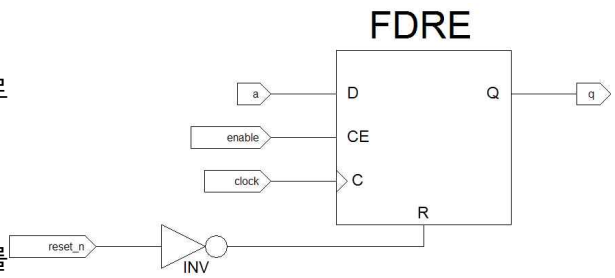
기타 내용에 오류가 있을 경우(-1pts)

아무 내용도 쓰지 않은 경우(기본 0pts)

회로(3pts +3pts)

A의 회로는 logic으로 그렸을 경우(3 input AND gate)나 logic을 이용하여 그린 경우(기본 3pts)
몇 개의 신호를 빠뜨렸을 경우(-1pts)
단순히 latch라고만 쓴 경우(기본 1pts)

B회로는 오른쪽 그림과 같거나 Flib Flop안의 로직을 쓴 경우(3pts)
각 신호가 하나씩 빠질때 마다(-1pts)
기타 오류(-1pts)
ex) reset을 clk에 넣지 않고 Q와 AND gate를



이용해서 연결하거나 reset을 inverting 시키지 않거나.

4. (2pts+3pts+5pts = 10pts)

A.

The processing of all the active events is called a ***simulation cycle.***

active event의 단어를 사용하여 설명하거나 active event가 수행되는 동안 이라는 의미의 말이 있을 경우(1pts)

Inactive events occur at the current simulation time, but shall be processed after all the active events are processed.

inactive event에 대한 예시를 제시하고 설명하거나 제대로 된 설명일 경우(1pts)

simulation cycle, inactive events에 대한 설명이 부족할 경우 각각 (-0.5pts)

B. (3pts)

a) Active events
b) Inactive events
c) Nonblocking assign event
d) Monitor events

위와 같은 순서가 맞을 경우 (3pts)
틀릴 경우(0pts)

C. (5pts)

```
while (there are events){
        if (no active events){
                if (there are inactive events) {
                        activate all inactive events;
                }
                else if (there are nonblocking assign update events) {
                        activate all nonblocking assign update events;
                }
                else if (there are monitor events) {
                        activate all monitor events;
                }
                else{
                        advance T to the next event time;
                    activate all inactive events for time T;
                }
        }
        E = any active event;
        if (E is an update event) {
                update the modified object;
```

```
            add evaluation events for sensitive processes to event queue;
        }
        else { /* shall be an evaluation event */
                evaluate the process;
                add update events to the event queue;
        }
}
```

위의 pseudo code와 같은 내용을 적은 경우.(기본 5pts)

pseudo 코드에서 약간 부족한 부분이 있는 경우(-1pts)

B의 순서와 비슷하게 순서대로 쓴 경우(-3pts)

5. (10pts)

```
module johnson(clk, start, qout);
parameter N = 4;
input  clk, start;
output reg [0:N-1] qout;

always @(posedge clk or posedge start)
      if (start)
      qout <= {N {1'b0 } };
      else
      qout <= {~qout[N-1], qout[0:N-2] };
endmodule
```

위와 같은 동작을 하거나 case문을 이용하여 제대로 된 동작을 할 경우(기본 10pts)

johnson counter의 동작이 반대로 수행할 경우(-2pts)

ex) 위에서 reg[0:N-1]인데 아래서 {qout[1:N-1],~qout[0]}와 같이

기타 사소한 문법 오류나 사소한 실수(-1pts)

start bit의 경우 자세한 설명이 없으므로 어떻게 시작하도록 짜도 상관없음.

단순한 counter 이거나 알 수없는 동작을 하는 counter일 경우(기본 3pts)

6. (10pts)

```
module priority_en(y,valid,in);
parameter n = 8;
parameter log2n = 3;
input [n-1:0] in;
input valid;
output reg [log2n-1:0] y;

always @(valid, in)
begin
```

```
        if(valid)
        begin
                casex (in)
                        8'b1xxxxxxx : y = 7;
                        8'b01xxxxxx : y = 6;
                        8'b001xxxxx : y = 5;
                        8'b0001xxxx : y = 4;
                        8'b00001xxx : y = 3;
                        8'b000001xx : y = 2;
                        8'b0000001x : y = 1;
                        8'b00000001 : y = 0;
                        default: y = 3'b000;
                endcase
        end
        else
                y=3'b000;
end
endmodule
```

valid bit은 어떻게 사용하도 상관없음.

always문을 적지 않고 바로 casex를 사용하신 경우(-1.5pts)

always문 안에 모든 input을 적지 않은 경우(valid와 in).(posedge valid로 사용하신 경우에는 in은 적지 않으셔도 됨).(-1pts)

casex문의 endcasex라고 적으신 경우(-1pts)

casex문 안에 8'b을 빼 먹으신 경우(-1pts)

x를 ?로 적으신 경우(-1pts)

거꾸로 count하신 경우(-2pts)

ex)8'bxxxxxxxx1

casex문으로 짜지 않으신 경우(-4tps)

기타 단순한 실수나 문법 오류(-1pts)

7. (10pts)

```
module universal(clk, reset_n, s0, s1,rsi, lsi, din, qout);
parameter n = 4;
input clk, reset_n;
input s0, s1; // control
input lsi, rsi; // left shift input, right shift input
input [n-1:0] din;
output reg [n-1:0] qout;

always @(posedge clk or negedge reset_n)
    if (!reset_n) qout <= {N {1'b0 }};
```

```
    else case ( {s1,s0 })
            2'b00: qout <= qout;
            2'b01: qout <= {rsi, qout[N-1:1]};
            2'b10: qout <= {qout[N-2:0], lsi};
            2'b11: qout <= din;
    endcase
endmodule
```

reset은 synchronous나 asynchronous는 상관없음
case문으로 짜지 않고 if else를 이용한 경우도 잘 동작할 경우 상관없음.
rsi나 lsi값에 다른 값을 넣었을 경우(-2pts)
rsi을 lsi로, lsi를 rsi로 쓴 경우(-2pts)
endcase를 적지 않거나 begin을 사용하고 end를 사용하지 않거나 하는 단순한 실수나 문법 오류의 경우(-1pts)

8. (10pts+10pts=20pts)

A. (10pts)

```
module single_cycle(clock,a,b,c,x);
input [7: 0] a, b, c;
input clock;
output [7:0] x;
reg [7:0]x;

always @(posedge clock) begin
     x <= c - (a + b);
end
endmodule
```

reg [7:0]x; 를 쓰지 않고 always 문장 안에 넣을 경우(-1pts)
2cycle에 답이 나오도록 위 문장에 추가한 경우(-2pts)
B답을 여기에 썼을 경우(-3pts)
기타 사소한 실수(-1pts)
ex) 수식을 a-(b+c)로 썼거나.

B.(10pts)

```
module pipeline_3cycle(clock,a,b,c,x);
input [7: 0] a, b, c;
input clock;
output [7:0] x;
reg [7:0] x,qa,qb,qc,qd,qe;

always @(posedge clock) begin
```

```
  qa<=a; qb<=b; qc<=c;
  qd<=(qa+qb);
  qe<=qc;
  x<=qe-qd;
end
endmodule
```

위 문장을 3개의 always문으로 나눈 경우도 같은 동작을 하므로 상관없음.
reg로 always에 있는 output들에 선언을 제대로 하지 않은 경우(-1pts)
qa<=a; qb<=b; qc<=c;가 빠진 경우(-3pts)
qd<=(qa+qb);
qe<=qc;
x<=qe-qd;
가 각각 빠진 경우나 잘못 쓴 경우(-1pts)
문장 중간 중간에 @(posedge clok)을 넣어서 pipeline 동작하지 않은 경우(-5pts)
강제로 3cycle delay를 준 경우(-7pts)
간단한 실수나 문법 오류의 경우(-1pts)

9.

```
module parity(data, parity, even_odd, error);
input [7: 0] data;
input parity;
input even_odd; // 0 for even parity, 1 for odd parity
output error; // 1 for parity error
wire cnt;
assign cnt=^data;
assign error = cnt^parity^even_odd;

endmodule
```

data와 parity bit 안에 있는 1이 홀수개인지 짝수개인지 알아내고
even_odd bit를 이용하여 error를 판별하는 동작을 하는 경우(기타 xor을 사용하거나 count 한 방법)
(기본 10pts)
parity bit, even odd bit를 잘못 이해하고 짠 경우(-5pts)
전부 잘 동작하지만 parity bit를 넣지 않고 data만 이용한 경우(-3pts)
1을 count할 때, 한 output들을 여러 문장에서 동시에 사용한 경우(-3pts)
ex)

```
if(data[8])
cnt=cnt+ 1;
if(data[7])
cnt=cnd+ 1;
```

...
기타 사소한 오류나 실수 (-1pts)

10.

```
module parity10(clock, data, even_odd, error, reset);
input data, clock;
input even_odd; // 0 for even parity, 1 for odd parity
input reset;
output reg error; // 1 for parity error
reg [3:0]state;
reg [8:0]buff;
parameter
  S0=4'd0,  S1=4'd1,  S2=4'd2,  S3=4'd3,
  S4=4'd4,  S5=4'd5,  S6=4'd6,  S7=4'd7,
  PA=4'd8,//paritystate
  IDLE=4'd10;
always@(posedge clock or posedge reset)
begin
        if(reset)
          begin
                state=IDLE;
                buff=0;
                error=0;
          end
        else
        case(state)
                S0: begin
                state = S1;
                buff={buff[7:0],data};
                end
                S1:begin
                state = S2;
                buff={buff[7:0],data};
                end
                S2:begin
                state = S3;
                buff={buff[7:0],data};
                end
                S3:begin
                state = S4;
                buff={buff[7:0],data};
```

```
                    end
                    S4:begin
                    state = S5;
                    buff={buff[7:0],data};
                    end
                    S5:begin
                    state = S6;
                    buff={buff[7:0],data};
                    end
                    S6:begin
                    state = S7;
                    buff={buff[7:0],data};
                    end
                    S7:begin
                    state = PA;
                    buff={buff[7:0],data};
                    end
                    PA: begin
                    state= S0;
                    buff={buff[7:0],data};
                    error= ((^buff)^(even_odd));
                    end
                    IDLE: begin
                    state = S0;
                    buff={8'b0000_0000,data};
                    end
                    default state = IDLE;
          endcase
end
endmodule
```

moore FSM으로 동작하거나 짜지 않은 경우(-3pts)
문제를 이해했지만 9bit를 연속적으로 검사하지 못하거나(중간에 다른 state가 존재) 2 state 혹은 4state를 짜고 count를 추가하지 않아 8bit+1bit를 체크하지 못하는 경우나 약간의 문제 때문에 잘 동작하지 않는 경우.(기본점수 7.5pts)
parity bit에 대한 이해가 부족하고 잘 동작하지 않는 경우(3pts)
기타 사소한 실수나 문법 오류(-1pts)

11. (10pts)

```
module sequence0101(clock, data, z, reset_n);
input data, clock, reset_n;
```

```
output reg z;
reg[1:0] state, next_state;

parameter
A=2'd0,
B=2'd1,
C=2'd2,
D=2'd3;

always@(posedge clock or negedge reset_n)
begin
  if(!reset_n) state<=A;
  else state <= next_state;
end

always@(state or data)
begin
  case(state)
          A: if(data) next_state = A;  else next_state = B;
          B: if(data) next_state = C;  else next_state = B;
          C: if(data) next_state = A;  else next_state = D;
          D: if(data) next_state = C;  else next_state = B;
  endcase
end

always@(state or data)
begin
  case(state)
    A:begin
        if(data) z=0;
        else z=0;
      end
    B:begin
        if(data) z=0;
        else z=0;
      end
    C:begin
        if(data) z=0;
```

```
            else z=0;
          end
      D:begin
            if(data) z=1;
            else z=0;
          end
      endcase
end
endmodule
```

always문 3개를 이용하여 짜지 않는 경우(-2pts)
FSM 형태가 아니고 동작하지 않는 경우(기본 3pts)
moore FSM이 아닌경우(-2pts)
state를 선언시에 길이를 잘못 선언한 경우[1:0](-1pts)
기타 사소한 실수나 문법 오류(-1pts)