[2010년 디지털 시스템 설계 및 실험 중간고사 1 답안지]

출제: 채수익

Verilog 문법 채점 기준 (따로 문제의 채점 기준에 명시되어 있지 않아도 적용되어 있음)
(a) output이 always 문에서 사용된 경우, reg로 선언하지 않은 경우 (-1 pts)
(b) reg, wire를 혼동하여 사용한 경우 (-1 pts)
(c) always @(...)에서 모든 input을 sensitivity list에 집어넣지 않은 경우 (-1 pts)
(d) instantiation 시에 instance 이름이 없는 경우 (-1 pts)
(e) always 문 밖에서 for loop을 사용할 때는 integer가 아닌 genvar로 loop variable을 선언해야 한다. (-1 pts)
(f) 그 외에 세미콜론, 오타 등의 사소한 문법상 실수 (여러 개가 있어도 문제당 -1 pts)
-> for loop에서 i++는 verilog에서 사용되지 않음

1.
A. (5 pts)

```
module mux_2to1(x, a, b, select);
parameter n = 8;
input select;
input [n-1:0] a, b;
output [n-1:0] x;

wire [n-1:0] a, b;
wire [n-1:0] x;
assign x = (select == 1'b0) ? a : b;

endmodule
```

case 문, if/else 등으로 맞게 작성만 하면 5점.
case 문, if/else 사용 시 always 문으로 감싸지 않은 경우 (-1 pts)
잘못된 reg, wire의 선언 (-1 pts)
always @(...)에서 모든 input을 sensitivity list에 집어넣지 않은 경우 (-1 pts)
사소한 문법 오류 (-1 pts)

B. (5 pts)

```
module mux_8to1(x, a, b, c, d, e, f, g, h, select);
parameter n = 8;
input [2:0] select;
input [n-1:0] a, b, c, d, e, f, g, h;
output [n-1:0] x;
```

```
wire [n-1:0] a, b, c, d, e, f, g, h;
wire [n-1:0] x;
// internal wires
wire [n-1:0] tmp0, tmp1, tmp2, tmp3, tmp4, tmp5;
mux_2to1 mux001 (.x(tmp0), .a(a),     .b(b),       .select(select[0]));
mux_2to1 mux002 (.x(tmp1), .a(c),     .b(d),       .select(select[0]));
mux_2to1 mux003 (.x(tmp2), .a(e),     .b(f),       .select(select[0]));
mux_2to1 mux004 (.x(tmp3), .a(g),     .b(h),       .select(select[0]));
mux_2to1 mux010 (.x(tmp4), .a(tmp0),.b(tmp1),  .select(select[1]));
mux_2to1 mux020 (.x(tmp5), .a(tmp2),.b(tmp3),  .select(select[1]));
mux_2to1 mux100 (.x(x),       .a(tmp4),.b(tmp5), .select(select[2]));

endmodule
```

mux_2to1을 사용하지 않은 경우 (-2 pts)
case, if, always 등으로 mux_2to1 instantiation하는 부분을 감싼 경우 (-1 pts)
연결 wire들을 [n-1:0]로 선언하지 않은 경우 (-1 pts)
잘못된 reg, wire의 선언 (-1 pts)
사소한 문법 오류 (-1 pts)

2.
일반적으로 CLA에서 generate가 a&b에 해당하고, propagate가 a^b에 해당한다. 그러나 만일 모든 부분 문제에서 일관성있게 g와 p를 바꿔썼다면 감점 없음. 마찬가지로 g, p를 input a, b로 착각한 경우에도 일관성 있으면 감점 없음.
A. (3 pts)

```
module full_adder(sum, c_out, a, b, c_in);
input a, b, c_in;
output sum, c_out;

wire sum, c_out, tmp1, tmp2, tmp3;
assign sum = a ^ b ^ c_in;
assign tmp1 = a & b;
assign tmp2 = b & c_in;
assign tmp3 = a & c_in;
assign c_out = tmp1 | tmp2 | tmp3;

endmodule
```

{c_out,s} = a+b+c_in;를 사용한 경우 (2점)
sum이나 c_out 중 하나의 결과가 잘못되었을 경우 (-1점)
잘못된 reg, wire의 선언 (-1 pts)

사소한 문법 오류 (-1 pts) (assign을 해놓고 always로 둘러싼 경우도 포함)

B. (3 pts)

```
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;

reg [n-1:0] c_out;
integer i;

always @(*)
begin
  c_out[0] = g[0] | (c_in & p[0]);
  for (i=1; i<n; i=i+1) begin
    c_out[i] = g[i] | (c_out[i-1] & p[i]);
  end
end

endmodule
```

결과가 잘못된 경우 (0점)
Parameterized module을 작성하지 않은 경우 (-1 pts)
c_out을 1-bit output으로 생각하고 마지막 carry out을 넣은 경우 (-1 pts)
&, | 대신에 곱셈, 덧셈을 썼는데, bit-width가 1이 아닌 경우 (-1 pts)
잘못된 reg, wire의 선언 (-1 pts)
사소한 문법 오류 (-1 pts)

C. (5 pts)

```
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] carry;
wire [n-1:0] g, p;
```

```
integer i;

carry_lookahead #(n) cla_logic0 (.c_out(carry), .g(g), .p(p), .c_in(c_in));

full_adder fa [n-1:0] (S, , A, B, {carry[n-2:0],c_in});
assign g = A & B;
assign p = A ^ B;
assign c_out = carry[n-1];

endmodule
```

full_adder instantiation을 사용하지 않고 풀어 썼을 경우 (5점)
Parameterized module로 작성하지 않은 경우 (-2 pts)
#(4)로 parameterize한 경우 (-1 pts)
잘못된 reg, wire의 선언 (-1 pts)
사소한 문법 오류 (-1 pts) (#(n(n)), (#n)도 포함)

D. (5 pts)

```
module CLA_20bit_adder (S, c_out, A, B, c_in);
input [19:0] A, B;
input c_in;
output [19:0] S;
output c_out;

wire [3:0] group_carry, group_g, group_p;
reg [19:0] g,p;
integer i;

CLA_adder #(5) cadder_0 (S[19:15],c_out,A[19:15],B[19:15],group_carry[2]);
CLA_adder #(5) cadder_1 (S[14:10],         ,A[14:10],B[14:10],group_carry[1]);
CLA_adder #(5) cadder_2 (S[9:5]   ,        ,A[9:5]   ,B[9:5]   ,group_carry[0]);
CLA_adder #(5) cadder_3 (S[4:0]   ,        ,A[4:0]   ,B[4:0]   ,c_in);

carry_lookahead carry_look (.c_out(group_carry), .g(group_g), .p(group_p), .c_in(c_in));

p = A^B;
g = A&B;

assign group_p[0] = p[0]&p[1]&p[2]&p[3]&p[4];
assign group_p[1] = p[5]&p[6]&p[7]&p[8]&p[9];
```

```
assign group_p[2] = p[10]&p[11]&p[12]&p[13]&p[14];
assign group_p[3] = p[15]&p[16]&p[17]&p[18]&p[19];

assign group_g[0] = g[4]|(g[3]&p[4])|(g[2]&p[3]&p[4])|(g[1]&p[2]&p[3]&p[4])|
                    (g[0]&p[1]&p[2]&p[3]&p[4]);
assign group_g[1] = g[9]|(g[8]&p[9])|(g[7]&p[8]&p[9])|(g[6]&p[7]&p[8]&p[9])|
                    (g[5]&p[6]&p[7]&p[8]&p[9]);
assign group_g[2] = g[14]|(g[13]&p[14])|(g[12]&p[13]&p[14])|(g[11]&p[12]&p[13]&p[14])|
                    (g[10]&p[11]&p[12]&p[13]&p[14]);
assign group_g[3] = g[19]|(g[18]&p[19])|(g[17]&p[18]&p[19])|(g[16]&p[17]&p[18]&p[19])|
                    (g[15]&p[16]&p[17]&p[18]&p[19]);
endmodule
```

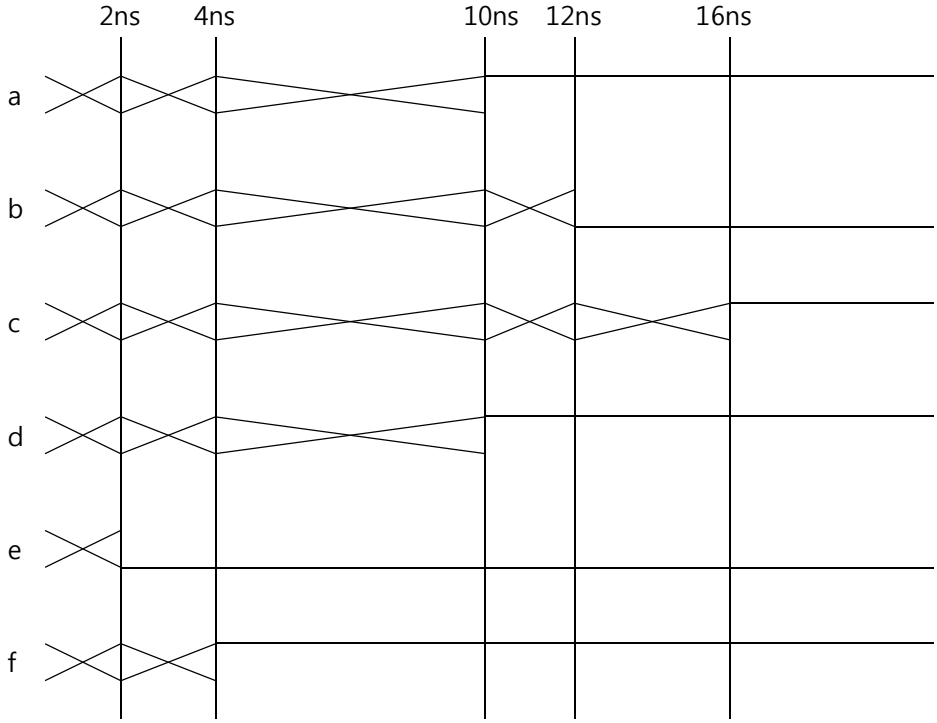
5-bit CLA 간의 carry 전달이 carry-lookahead logic이 아닌 경우, 즉 group_p, group_g를 사용하지 않은 경우 (-2 pts)

5개의 4-bit CLA adder를 쓴 경우, 혹은 instantiation 시에 parameter를 바꾸지 않은 경우 (-2 pts)

잘못된 reg, wire의 선언 (-1 pts)

사소한 문법 오류 (-1 pts)

3. (15 pts)



위에서 X 표시한 부분은 undefined에 해당한다.

하나의 signal에 대해 틀릴 때마다 2점씩 감점.

undefined 표기하지 않았을 경우 2점 감점.

모든 signal에 대해 틀리고, undefined 표기하지 않았을 경우 0점.

4. (5 pts + 5 pts)

두 module에 대한 설명 및 차이점 비교 (2 pts + 2 pts)

모두 있을 경우 (기본 4 pts)

하나에 대해서만 설명한 경우 (기본 2 pts)

아무런 설명도 없는 경우 (기본 0 pts)

내용에 오류가 있는 경우 (-1 pts)

회로 (3 pts + 3 pts)

아래의 block diagram대로 그리거나 혹은 gate level logic으로 그리면 (기본 6 pts)

일부 signal이 빠졌거나 이름을 잘못 쓴 경우 (-1 pts씩)

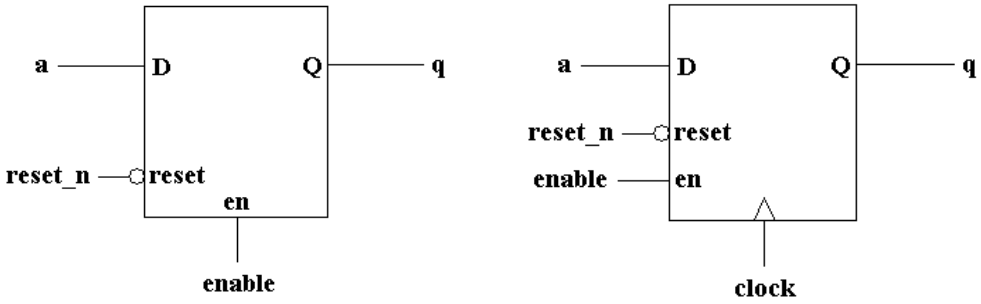
reset이 active low임을 고려하지 않은 경우 (-1 pts씩)

Latch의 schematic을 or, and 형태로 그린 경우 (-1 pts)

Latch의 block diagram이 잘못된 경우 (-1 pts) (세모꼴의 표시는 clock input을 의미 – 즉, flipflop)

Flipflop의 block diagram이 잘못된 경우 (-1 pts) (세모꼴의 표시는 clock input을 의미한다)

기타 오류 (-1 pts씩)



5. (3 pts + 3 pts + 5 pts)

A. (3 pts)

Simulation Cycle => Complete processing of all currently **active events** Inactive event => Events that occur at the **current simulation time**, but **will be processed after** all active events have been processed

위에 boldface 및 밑줄로 강조한 부분에 대한 내용이 있으면 된다. (각 1.5 pts)

강조 부분에 대해 명시하지는 않았으나, 기타 예시를 제시하고 설명하거나 제대로 된 설명을 한 경우 (각 1 pts)

B. (3 pts)

active event, inactive event, non-blocking assignment event, monitor event

순서가 하나라도 틀리면 0점.

C. (5 pts)

```
while (there are events)
  if (no active events)
    if (there are inactive events) {
      activate all inactive events;
    } else if (there are non-blocking assign update events) {
      activate all non-blocking 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 events;
  if (E is an update event) {
    update the modified object;
    add evaluation events for the sensitive processes to event queue;
  } else { // evaluation event
    evaluate the process;
    add all update events to the event queue;
  }
}
```

while loop이 있음 : 1점

if (no active events) 문이 있음 : 1점

inactive event, non-blocking assignment, monitor에 관한 if 문이 순서대로 있음 : 1점

monitor 이후의 else에서 다음 event time으로 넘어간다는 내용이 있음 : 1점

update/evaluation event에 관한 if 문이 있음 : 1점

6. (10 pts)

parameter인 n은 항상 4의 배수로 주어지는 것을 가정한다.

```
module BCD_counter(clk, start, qout);
parameter n = 8;
input clk, start;
output reg [n-1:0] qout;

wire [n-1:0] next_qout;
wire [n/4:0] carry;
```

```
always @(posedge clk)
begin
  if (start == 1'b1) begin
    qout <= {n{1'b0}};
  end else begin
    qout <= next_qout;
  end
end

genvar i;
generate for (i=0; i<n/4; i=i+1) begin: sub_BCD
  if (i == 0)
    BCD_1digit bcd (qout[i*4+3:i*4],1'b1,next_qout[i*4+3:i*4],carry[i]);
  else
    BCD_1digit bcd (qout[i*4+3:i*4],carry[i-1],next_qout[i*4+3:i*4],carry[i]);
end
endgenerate
endmodule

module BCD_1digit(curr_cnt, c_in, next_cnt, c_out);
input c_in;
input [3:0] curr_cnt;
output c_out;
output [3:0] next_cnt;

reg c_out;
reg [3:0] next_cnt;
reg [3:0] tmp;

always @(*)
begin
  tmp = curr_cnt + c_in;
  if (tmp > 4'd9) begin
    next_cnt = 4'h0;
    c_out = 1'b1;
  end else begin
    next_cnt = tmp;
    c_out = 1'b0;
```

```
  end
end
endmodule
```

Parameterized module로 작성하지 않은 경우 (-3 pts)
0~9까지의 BCD counter만을 구현한 경우 (-3 pts)
Output이 BCD 형태로 바뀌지 않은 경우 (-3 pts)
BCD(Binary Coded Decimal) counter가 아닌 다른 counter인 경우 (기본 2 pts)

7. (10 pts)

```
module Ring_counter(clk, start, qout);
parameter n = 8;
input clk, start;
output reg [n-1:0] qout;

always @(posedge clk or posedge start)
begin
  if (start) qout <= {1'b1, {n-1{1'b0}}};
  else       qout <= {qout[0], qout[n-1:1]};
end

endmodule
```

Parameterized module로 작성하지 않은 경우, 즉, 8-bit ring counter를 작성한 경우 (-3 pts)
Start 시에 qout을 0으로 한 경우 (-1 pts)
Rotational shift 형식으로 순환되지 않는 경우 (-1 pts)

8. (10 pts)

이 문제는 casex를 제대로 활용하기 위해서는 n, log2n 등에 대해 추가적인 가정이 필요하므로, parameterized module이 아닌 n=8인 특정 경우에 대한 해답을 제시한다.

```
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 @(*)
begin
  if (valid)
  begin
```

```
      casex(in)
        8'b1xxxxxxx: y=3'd7;
        8'b01xxxxxx: y=3'd6;
        8'b001xxxxx: y=3'd5;
        8'b0001xxxx: y=3'd4;
        8'b00001xxx: y=3'd3;
        8'b000001xx: y=3'd2;
        8'b0000001x: y=3'd1;
        8'b00000001: y=3'd0;
        default: y=3'd0;
      endcase
   end
   else y = 3'b000;
end

endmodule
```

Parameterized module로 작성한 경우 (10점+2점 : 추가점수 2점)
casex를 사용하지 않고 parameterized module로 작성한 경우 (기본 10점)
Priority가 반대로 된 경우 (-1 pts) (즉, 8'b1xxxxxxx에 y=3'd0를 한 경우)
사소한 문법 실수 (-1 pts)

9. (10 pts)

```
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;

reg [n-1:0] parallel_out;

always @(posedge clk or negedge reset_n)
begin
   if (!reset_n) qout <= {n{1'b0}};
   else qout <= parallel_out;
end

always @(*)
```

```
begin
  case ( {s1, s0} )
    2'b00: parallel_out = qout;
    2'b01: parallel_out = {rsi, qout[n-1:1]};
    2'b10: parallel_out = {qout[n-2:0], lsi};
    2'b11: parallel_out = din;
  endcase
end

endmodule
```

Parameterized module로 작성하지 않은 경우, 즉, 4-bit universal shift register를 작성한 경우 (-3 pts)

rsi, lsi를 고려하지 않은 경우 (-2 pts)

10.

A. (10 pts)

```
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 <= a * (b – c);
end

endmodule
```

사소한 문법 오류 (-1 pts)

계산이 single cycle에 이루어지지 않는 경우 (-2 pts)

잘못된 reg, wire의 선언 (-1 pts)

기타 사소한 실수 (-1 pts씩)

B. (10 pts)

```
module pipeline_3cycle(clock, a, b, c, x);
input [7:0] a, b, c;
input clock;
output [7:0] x;
```

```
reg [7:0] x;
reg [7:0] tmpa, tmpb, tmpc, tmpa2, tmp_sub, tmp_mul;

always @(posedge clock)
begin
  tmpa <= a; tmpb <= b; tmpc <= c;
  tmpa2 <= tmpa; tmp_sub <= tmpb – tmpc;
  tmp_mul <= tmpa2 * tmp_sub;
  x <= tmp_mul;
end

endmodule
```

잘못된 reg, wire의 선언 (-1 pts)
pipeline이 되어 있지 않은 경우 (-2 pts) (예를 들어, 3번의 @(posedge)를 사용한 경우)
latency가 3 cycle이 아닌 경우 (-2 pts) (예를 들어, blocking assignment를 사용한 경우)
3-stage pipeline을 고려할 때 결과가 잘못된 값인 경우 (-2 pts) (예를 들어, tmpa2 없이 tmp_mul 계산에서 tmpa를 그대로 사용하는 등)
기타 사소한 실수 및 문법 오류 (-1 pts)

11. (10 pts)

```
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 error;
wire total_parity;

assign total_parity = (^data) ^ parity;
assign error = data_parity ^ even_odd;

endmodule
```

사소한 실수 및 문법 오류 (-1 pts)

12. (10 pts) 최초의 순간에만 IDLE state에 있으며, 이 때 입력된 data는 무시한다. 이후에는 data로 필요한 값이 매 cycle마다 입력된다고 가정한다.

```
module parity(clock, data, even_odd, error);
input data, clock;
input even_odd; // 0 for even parity, 1 for odd parity
```

```
output reg error;

reg [3:0] curr_state;
reg [3:0] next_state;
reg curr_value;
reg next_value;
parameter
  IDLE = 4'd0,
  D1 = 4'd1, D2 = 4'd2, D3 = 4'd3, D4 = 4'd4,
  D5 = 4'd5, D6 = 4'd6, D7 = 4'd7, D8 = 4'd8,
  P1 = 4'd9; // parity

initial
begin
  next_state <= IDLE;
  next_value <= 1'b0;
end

always @(posedge clock)
begin
  curr_state <= next_state;
  curr_value <= next_value;
end

always @(*)
begin
  error = 1'bx;
  case (curr_state)
    IDLE: begin
      next_state = D1; next_value = 1'b0;
    end
    D1: begin
      next_state = D2; next_value = data; error <= curr_value;
    end
    D2: begin
      next_state = D3; next_value = curr_value ^ data;
    end
    D3: begin
      next_state = D4; next_value = curr_value ^ data;
```

```
    end
    D4: begin
      next_state = D5; next_value = curr_value ^ data;
    end
    D5: begin
      next_state = D6; next_value = curr_value ^ data;
    end
    D6: begin
      next_state = D7; next_value = curr_value ^ data;
    end
    D7: begin
      next_state = D8; next_value = curr_value ^ data;
    end
    D8: begin
      next_state = P1; next_value = curr_value ^ data;
    end
    P1: begin
      next_state = D1; next_value = curr_value ^ data ^ even_odd;
    end
    default: begin
      next_state = IDLE; next_value = 1'b0;
    end
  endcase
end

endmodule
```

다른 state를 추가적으로 도입하는 등의 경우에도 결과만 정확하면 기본 10pts.
Moore machine이므로, error는 current state에만 dependent하고 input에는 independent해야 한다. 이 조건을 어겼을 경우 (-2 pts)
기타 사소한 실수 및 문법 오류 (-1 pts)

13. (10 pts)

```
module sequence0101(clock, data, z, reset_n);
input data, clock, reset_n;
output reg z; // 1 if detected

reg [1:0] curr_state, next_state;
parameter
  A = 2'b00, B = 2'b01, C = 2'b10, D = 2'b11;
```

```
always @(posedge clock)
begin
  if (reset_n == 1'b0) begin
    curr_state <= A;
  end else begin
    curr_state <= next_state;
  end
end

always @(*)
begin
  case (curr_state)
    A: if (data == 1'b1) next_state = A; else next_state = B;
    B: if (data == 1'b1) next_state = C; else next_state = B;
    C: if (data == 1'b1) next_state = A; else next_state = D;
    D: if (data == 1'b1) next_state = C; else next_state = B;
  endcase
end

always @(*)
begin
  case (curr_state)
    A: z = 0;
    B: z = 0;
    C: z = 0;
    D: if (data == 1'b1) z = 1; else z = 0;
  endcase
end

endmodule
```

Mealy machine이므로, z는 input과 current state에만 dependent하다. 이를 어겼을 경우 (-2 pts)

always 문이 3개가 아닌 경우 (-2 pts)

Parameter 문을 잘못 사용한 경우 (-8 pts)