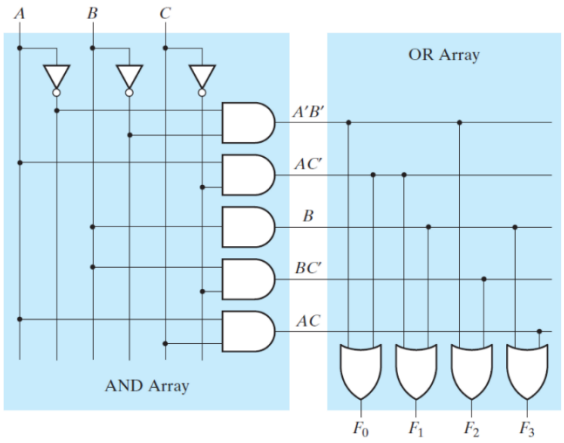
EE201 term project

Compose all the blocks using **Verilog**

202211167 장지원

1. Compose the simple blocks. [40 Pts]

   A. The PLA shown below. [10 Pts]



design.sv: 위 PLA에 따라 and, or, xor gate를 활용해서 gate-level로 코드를 구성해 보았다. 밑은 코드의 일부분이다.

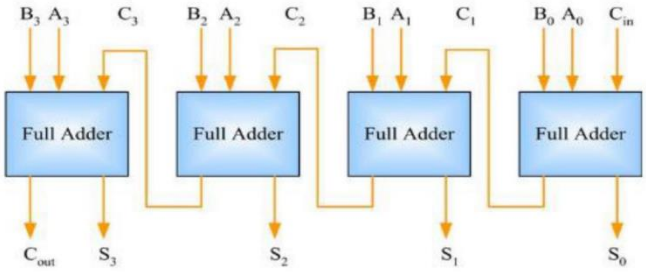
```
and G1 (W1, ~A, ~B);
and G2 (W2, A, ~C);
and G3 (W4, B, ~C);
and G4 (W5, A, C);

or G5 (F0, W1, W2);
or G6 (F1, W2, B);
or G7 (F2, W1, W4);
or G8 (F3, W4, W5);
```

testbench.sv: input으로 가능한 8가지의 경우를 모두 고려하도록 testbench를 작성해보았다. 결과는 밑과 같았고, 이는 실제 위 PLA의 동작과 일치한다.

```
Test Case 1: A=0, B=0, C=0, F0=1, F1=0, F2=1, F3=0
Test Case 2: A=0, B=0, C=1, F0=1, F1=0, F2=1, F3=0
Test Case 3: A=0, B=1, C=0, F0=0, F1=1, F2=1, F3=1
Test Case 4: A=0, B=1, C=1, F0=0, F1=1, F2=0, F3=0
Test Case 5: A=1, B=0, C=0, F0=1, F1=1, F2=0, F3=0
Test Case 6: A=1, B=0, C=1, F0=0, F1=0, F2=0, F3=1
Test Case 7: A=1, B=1, C=0, F0=1, F1=1, F2=1, F3=1
Test Case 8: A=1, B=1, C=1, F0=0, F1=1, F2=0, F3=1
```

B. A full adder and a 4b parallel adder by calling a full adder as a submodule. [10 Pts]



design.sv: full adder를 sub module로 이용하는 4b parallel adder을 구현해보았다. 밑은 코드의 일부분이다.

```
xor G1 (w1, A, B);
xor G2 (S, w1, C_);

and G3 (w2, A, B);
and G4 (w3, w1, C_);
or G5 (C, w2, w3);
```

full adder module의 gate-level 구현이다.

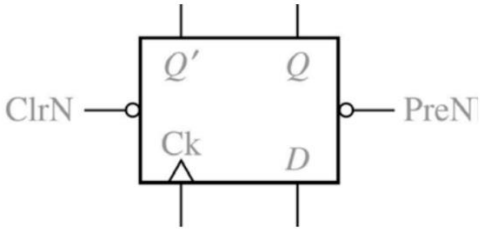
```
assign_1_B_FullAdder U0 (.A(A[0]), .B(B[0]), .C_(1'b0), .S(w_sum[0]), .C(w_carry1));
assign_1_B_FullAdder U1 (.A(A[1]), .B(B[1]), .C_(w_carry1), .S(w_sum[1]), .C(w_carry2));
assign_1_B_FullAdder U2 (.A(A[2]), .B(B[2]), .C_(w_carry2), .S(w_sum[2]), .C(w_carry3));
assign_1_B_FullAdder U3 (.A(A[3]), .B(B[3]), .C_(w_carry3), .S(w_sum[3]), .C(Cout));
```

full adder module을 이용한 4b parallel adder의 구현이다. 이전 carry out를 carry in으로 받으며, 출력은 sum[i]에 저장한다.

testbench.sv: 모든 input을 고려하기는 어렵다. 따라서 random으로 8개의 input만 받아서 결과를 출력하도록 코드를 구성해 보았다. 이는 실제 4b parallel adder의 동작과 일치한다.

```
Test Case 1: A=0100, B=0001, Sum=0101, Cout=0000
Test Case 2: A=1001, B=0011, Sum=1100, Cout=0000
Test Case 3: A=1101, B=1101, Sum=1010, Cout=0001
Test Case 4: A=0101, B=0010, Sum=0111, Cout=0000
Test Case 5: A=0001, B=1101, Sum=1110, Cout=0000
Test Case 6: A=0110, B=1101, Sum=0011, Cout=0001
Test Case 7: A=1101, B=1100, Sum=1001, Cout=0001
Test Case 8: A=1001, B=0110, Sum=1111, Cout=0000
```

C. A D flip-flop shown below. [10 Pts]



design.sv: PPT를 바탕으로 asynchronous input이 있는 D flip-flop를 구현해 보았다. 기존 코드에 PreN signal을 추가해 주었다. 밑은 코드의 일부분이다.

```
always @ (posedge Clk, negedge PreN, negedge ClrN
    if (!ClrN) Q <= 1'b0;
    else if (!PreN) Q <= 1'b1;
    else Q <= D;
```

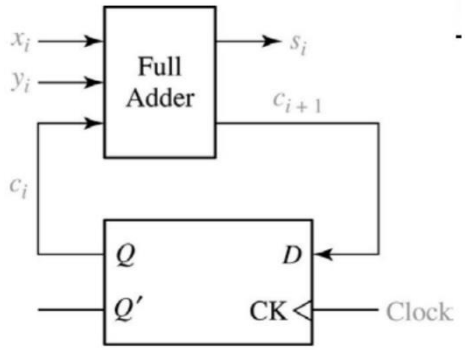
해당 부분은 Clk signal이 positive edge일 때, PreN이 negative edge(bubble이 없다면 positive edge)일 때, ClrN이 negative edge(bubble이 없다면 positive edge)일 때 동작한다.
위 부분이 실행될 때 ClrN이 0이 되면 Q는 0이 된다. PreN이 0이 되면 Q는 0이 된다. 둘 다 0이 아니라면 D flip-flop 원래의 동작을 수행한다(D가 Q로 그대로 전달된다).

testbench.sv: 5가지 경우로 나누어서 test를 진행해 보았다.
Test case 1은 rising edge clock, no preset, no clear인 경우이다. 이 경우에는 D가 그대로 Q에 전달된다.
Test case 2는 rising edge clock, no preset, clear인 경우이다. 이 경우에는 clear signal로 인해 Q가 0이 된다.
Test case 3은 rising edge clock, preset, no clear인 경우이다. 이 경우에는 preset signal로 인해 Q가 1이 된다.
Test case 4은 rising edge clock, no preset, no clear인 경우이다. 이 경우에는 D가 그대로 Q에 전달된다.
Test case 5는 falling edge clock, no preset, no clear인 경우이다. 이 경우에는 falling edge이기 때문에 아무런 변화가 없다.

```
Test Case 1: D=0, Q=0
Test Case 2: D=1, Q=0
Test Case 3: D=0, Q=1
Test Case 4: D=1, Q=1
Test Case 5: D=0, Q=1
```

D. A serial adder by calling the D flip-flop you composed in Prob. 3. [10 Pts] [Your testbench may cover

4b input cases and does NOT need to cover all the input case in this problem.]



design.sv: 1-B와 1-C에서 구현한 full adder, D flip-flop을 사용해 serial adder를 구현해 보았다. 밑은 코드의 일부분이다.

```
assign_1_B_FullAdder U1 (
    .A(in_a[0]),
    .B(in_b[0]),
    .C_(carry_in),
    .S(sum_out[0]),
    .C(carry0)
  );

  assign_1_C U11 (
    .Q(carry1),
    .D(carry0),
    .Clk(clk),
    .PreN(rst),
    .ClrN(~rst)
  );
```

full adder의 연산을 flip-flop이 저장해서 넘겨주는 형태로 코드를 작성해 보았다. 결과는 각각 sum[i]에 저장된다. 위와 같은 full adder/flip-flop set이 3개 있고, 마지막 full adder에서 최종 결과를 출력한다.

원래 밑과 같이 하나의 full adder와 하나의 D flip-flop만을 이용하도록 코드를 구성했었다. 하지만 in_a[i], in_b[i], sum_out[i]과 같이 indexing 하는 부분에서 error가 생겨서 아래 코드로는 serial adder를 구현하지 못했다.

```
assign_1_C U0 ( .Q(carry_in), .D(carry_out), .Clk(clk), .PreN(rst), .ClrN(~rst) );

assign_1_B_FullAdder U1
```

```
( .A(in_a[i]), .B(in_b[i]), .C_(carry_in), .S(sum_out[i]), .C(carry_out) );

always @(posedge clk or posedge rst)
  begin if (rst) begin
  sum_out <= 4'b0; carry_in <= 1'b0;
  end else begin
  i <= i+1; end
  end
```

testbench. sv: 8개의 input만 받아서 결과를 출력하도록 코드를 구성해 보았다.

```
Test Case 1: Input A: 0000, Input B: 0011, Sum: 0011, Carry_out: 0
Test Case 2: Input A: 1100, Input B: 0011, Sum: 1111, Carry_out: 0
Test Case 3: Input A: 1000, Input B: 0011, Sum: 1011, Carry_out: 0
Test Case 4: Input A: 0011, Input B: 0011, Sum: 0xx0, Carry_out: 0
Test Case 5: Input A: 1000, Input B: 1001, Sum: 0001, Carry_out: 1
Test Case 6: Input A: 1111, Input B: 1111, Sum: xxx0, Carry_out: 1
Test Case 7: Input A: 0000, Input B: 1111, Sum: 1111, Carry_out: 0
Test Case 8: Input A: 0001, Input B: 1011, Sum: 1xx0, Carry_out: 0
```

test case를 보면 carry가 다음 bit로 이동하지 못하는 걸 확인할 수 있다(carry가 잘 전달되지 못했기 때문에 don't care term이 생겼다). 이는 carry를 reg로 정의하지 않고 wire로 정의했기 때문에 생기는 문제이다. 하지만 reg로 정의하면

```
Variable "carry0" is driven by an invalid combination of structural and
  procedural drivers. Variables driven by a structural driver cannot have
any
  other drivers.
```

위와 같은 오류가 생긴다. 아마 같은 변수(carry)를 가진 full adder와 flip flop이 clk rising edge에서 함께 update 되면서 발생하는 문제로 보인다. 이를 해결할 방법은 찾지 못하였다.

인터넷을 통해 정상적으로 동작하는 serial adder code를 가져와 보았다(해당 코드는 파일 밑에 첨부하였다). 인터넷의 코드는 한 bit씩 받아서 결과를 출력하는 형식이다. 밑은 0111+0101 연산 결과이다.

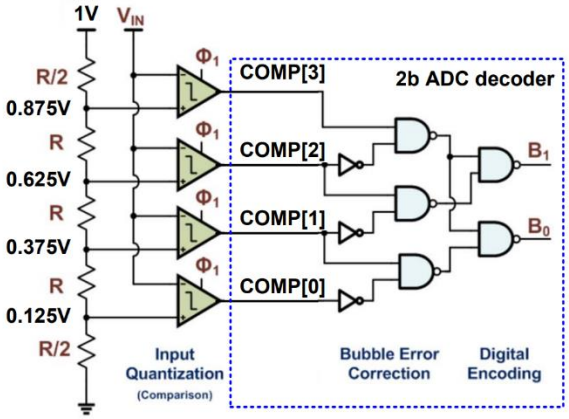
코드 출처: [Verilog Coding Tips and Tricks: Verilog code for an N-bit Serial Adder with Testbench code (verilogcodes.blogspot.com)]

```
4-bit addition result: s = 0
4-bit addition result: s = 0
4-bit addition result: s = 1
4-bit addition result: s = 1
```

1100으로 잘 계산되었음을 확인할 수 있다.

2. The main project [60 Pts]

A. Compose the 2b flash ADC decoder shown below. You MUST use gate-level implementation. (Refer to page 37 in the Chapter 5 slide.) The output of a comparator will be HIGH when V+ is higher than V-. For example, in case VIN is 0.15V, the COMP[3:0] and B[1:0] will be 1110 and 01, respectively.



design.sv: COMP[3:0]를 input으로 하고, B[1:0]를 output으로 하는 코드를 작성해 보았다. 이는 gate-level로 구현하였다. 밑은 코드의 일부분이다.

```
nand G1 (W1, COMP[3], ~COMP[2]);
nand G2 (W2, COMP[2], ~COMP[1]);
nand G3 (W3, COMP[1], ~COMP[0]);
nand G4 (B[1], W1, W2);
nand G5 (B[0], W1, W3);
```

testbench.sv: COMP의 모든 경우를 고려하는 testbench를 구성해 보았다. 밑의 2-B에서 작성한 truth table과 같은 결과가 나왔음을 확인할 수 있다.

```
Test Case 0: COMP = 1111, B = 00
Test Case 1: COMP = 1110, B = 01
Test Case 2: COMP = 1100, B = 10
Test Case 3: COMP = 1000, B = 11
Test Case 4: COMP = 0000, B = 00
```

B. Compose the truth table regarding all the cases of COMP[3:0] and the resultant B[1:0].

V_{in}	COMP[3]	COMP[2]	COMP[1]	COMP[0]	B[1]	B[0]
~ 0.125V	1	1	1	1	0	0
0.125V ~ 0.375V	1	1	1	0	0	1
0.375V ~ 0.625V	1	1	0	0	1	0

0.625V ~ 0.825V	1	0	0	0	1	1
0.875V ~	0	0	0	0	0	0

COMP[3:0] input에 따른 B[1:0]의 output을 나타낸 truth table이다.

C. Explain why bubble error correction is necessary in the flash ADC.

범위 경계 근처의 전압은 noise로 인해 값이 경계를 넘어가는 등의 문제가 발생할 수 있다. bubble error correction은 이러한 동작에서 문제를 감지하고 값을 보정한다. 이를 위해 이는 필수적이다.

3. Source

- *(for gate level) Verilog Gate Level Examples (chipverify.com)*
- *(for serial adder) Verilog Coding Tips and Tricks: Verilog code for an N-bit Serial Adder with Testbench code (verilogcodes.blogspot.com)*

4. Code

serial adder 추가 코드이다.

```
module serial_adder
    (   input clk,reset,   //clock and reset
        input a,b,cin,   //note that cin is used for only first iteration.
        output reg s,cout   //note that s comes out at every clock cycle and cout is valid only for last clock cycle.
        );

reg c,flag;

always@(posedge clk or posedge reset)
begin
    if(reset == 1) begin //active high reset
        s = 0;
        cout = c;
        flag = 0;
    end else begin
        if(flag == 0) begin
            c = cin;   //on first iteration after reset, assign cin to c.
            flag = 1;   //then make flag 1, so that this if statement isnt executed any more.
        end
        cout = 0;
        s = a ^ b ^ c;   //SUM
        c = (a & b) | (c & b) | (a & c);   //CARRY
    end
end

endmodule


module tb;

    // Inputs
    reg clk;
    reg reset;
    reg a;
    reg b;
    reg cin;

    // Outputs
    wire s;
    wire cout;

    // Instantiate the Unit Under Test (UUT)
    serial_adder uut (
        .clk(clk),
        .reset(reset),
        .a(a),
        .b(b),
        .cin(cin),
```

```
            .s(s),
            .cout(cout)
        );

//generate clock with 10 ns clock period.
        always
            #5 clk = ~clk;

        initial begin
            // Initialize Inputs
            clk = 1;
            reset = 0;
            a = 0;
            b = 0;
            cin = 0;
            reset = 1;
            #20;
            reset = 0;
            a = 1; b = 1; cin = 0;        #10;
                $display("4-bit addition result: s = %b", s);
            a = 1; b = 0; cin = 0;    #10;
                $display("4-bit addition result: s = %b", s);
            a = 1; b = 1; cin = 0;    #10;
                $display("4-bit addition result: s = %b", s);
            a = 0; b = 0; cin = 0;    #10;
                $display("4-bit addition result: s = %b", s);


        end

endmodule
```