8Bit Full Adder Fnd 설계

김민기

목차

1 | RTL code 설명 (adder_8bit.v, fnd_controller.v)

RTL code 설명 (adder_8bit.v)

```v
module full_adder (
    input  a,
    input  b,
    input  cin,
    output sum,
    output c
);

    wire w_ha_sum, w_ha0_c, w_ha1_c;
    assign c = w_ha0_c | w_ha1_c;  // to full adder output c

    half_adder U_HA0 (
        .a    (a),          // from full adder input a
        .b    (b),
        .sum  (w_ha_sum),
        .carry(w_ha0_c)
    );

    half_adder U_HA1 (
        .a    (w_ha_sum),  // from half adder0 output sum
        .b    (cin),
        .sum  (sum),       // to full adder output sum
        .carry(w_ha1_c)
    );
endmodule

module half_adder (
    input  a,
    input  b,
    output sum,
    output carry
);
    // half adder
    //assign sum   = a ^ b;
    //assign carry = a & b;

    // gate premetive method
    xor (sum, a, b);
    and (carry, a, b);

endmodule
```

(1)

```v
module adder_4bit (
    input  a0, a1, a2, a3,
    input  b0, b1, b2, b3, cin,
    output sum0, sum1, sum2, sum3, c
);

    wire w_fa0_c, w_fa1_c, w_fa2_c;

    full_adder U_FA0 (
        .a  (a0),
        .b  (b0),
        .cin(cin),
        .sum(sum0),
        .c  (w_fa0_c)
    );

    full_adder U_FA1 (
        .a  (a1),
        .b  (b1),
        .cin(w_fa0_c),
        .sum(sum1),
        .c  (w_fa1_c)
    );

    full_adder U_FA2 (
        .a  (a2),
        .b  (b2),
        .cin(w_fa1_c),
        .sum(sum2),
        .c  (w_fa2_c)
    );

    full_adder U_FA3 (
        .a  (a3),
        .b  (b3),
        .cin(w_fa2_c),
        .sum(sum3),
        .c  (c)
    );
endmodule
```

(2)

1 | (1)번 사진은 half_adder와 full_adder의 코드이다.
half_adder는 xor, and 논리 게이트로 구성했고, half_adder를 instance화 하여 full_adder를 구성하였다.

(2)번 사진은 full_adder 4개를 붙여 4bit full adder를 표현했다.
이 역시 full_adder unit 4개를 각각 instance화 하여 구성하였다.

RTL code 설명 (adder_8bit.v)

```v
module adder_8bit (
    input  [7:0] a,
    input  [7:0] b,
    output [7:0] sum,
    output       c
);

    wire w_fa4_0_c;

    adder_4bit U_FA4_0 (
        .a0  (a[0]),
        .a1  (a[1]),
        .a2  (a[2]),
        .a3  (a[3]),
        .b0  (b[0]),
        .b1  (b[1]),
        .b2  (b[2]),
        .b3  (b[3]),
        .cin (1'b0),
        .sum0(sum[0]),
        .sum1(sum[1]),
        .sum2(sum[2]),
        .sum3(sum[3]),
        .c   (w_fa4_0_c)
    );

    adder_4bit U_FA4_1 (
        .a0  (a[4]),
        .a1  (a[5]),
        .a2  (a[6]),
        .a3  (a[7]),
        .b0  (b[4]),
        .b1  (b[5]),
        .b2  (b[6]),
        .b3  (b[7]),
        .cin (w_fa4_0_c),
        .sum0(sum[4]),
        .sum1(sum[5]),
        .sum2(sum[6]),
        .sum3(sum[7]),
        .c   (c)
    );

endmodule
```

```v
`timescale 1ns / 1ps

module top_adder (
    input  [7:0] a,
    input  [7:0] b,
    output [3:0] fnd_digit,
    output [7:0] fnd_data,
    output       c
);

    wire [7:0] w_sum;

    fnd_controller U_FND_CNTL (
        .sum      (w_sum),
        .fnd_digit(fnd_digit),
        .fnd_data (fnd_data)
    );

    adder_8bit U_ADDER (
        .a  (a),
        .b  (b),
        .sum(w_sum),
        .c  (c)
    );

endmodule
```

1 | 왼쪽부터 adder_8bit, top_adder module 코드이다.

8bit를 표현하기 위해 input, output을 vector로 표현하였다. (ex: a[0] ~ a[7])
이는 나중에 입출력이 수십, 수백 개일 때의 일일이 선언하지 않아도 되는 수고를 덜어준다.
8bit이기 때문에 4bit full adder unit 2개를 instance화 하여 구성하였다.

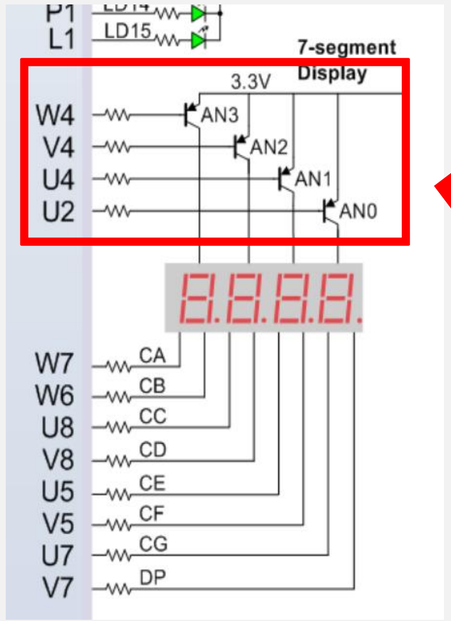
2 | top_adder의 코드는 위에서 설명한 8bit adder와 나중에 설명할 fnd_controller를 통합하고 있는 top_module이다.

입력은 a, b 그대로지만 출력은 sum이 아닌 fnd_digit과 fnd_data가 추가되었다.
이는 fnd_controller의 출력이고, 이 module을 instance화 하여 출력을 연결 해주었다.

전체적인 흐름은 입력이 top_adder안에서 adder_8bit를 거쳐 fnd_controller를 통해 출력으로 나오는 구조를 가진다.

RTL code 설명 (fnd_controller.v)

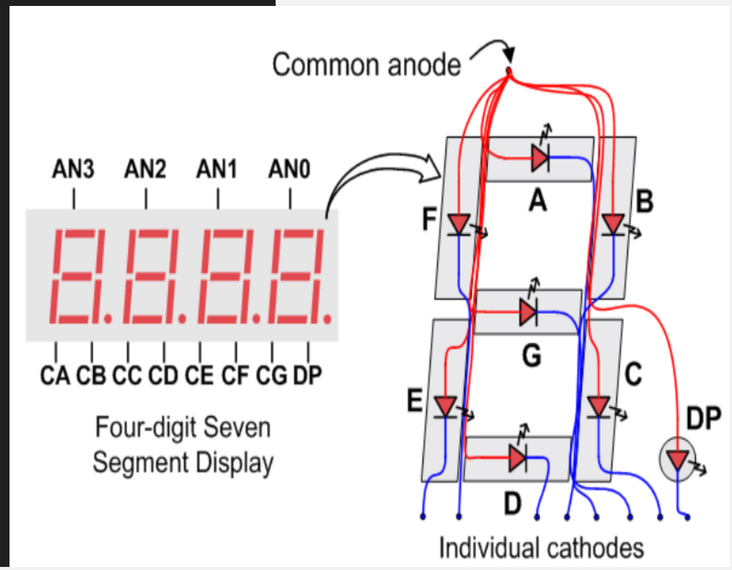
```
`timescale 1ns / 1ps

module fnd_controller (
    input  [7:0] sum,
    output [3:0] fnd_digit,
    output [7:0] fnd_data
);

    assign fnd_digit = 4'b1110;  // to fnd an[3:0]

    bcd U_BCD (
        .bcd      (sum[3:0]),
        .fnd_data(fnd_data)
    );

endmodule

module bcd (
    input      [3:0] bcd,
    output reg [7:0] fnd_data  // always output must be reg type
);

    always @(bcd) begin
        case (bcd)
            4'd0: fnd_data = 8'hC0;
            4'd1: fnd_data = 8'hF9;
            4'd2: fnd_data = 8'hA4;
            4'd3: fnd_data = 8'hB0;
            4'd4: fnd_data = 8'h99;
            4'd5: fnd_data = 8'h92;
            4'd6: fnd_data = 8'h82;
            4'd7: fnd_data = 8'hF8;
            4'd8: fnd_data = 8'h80;
            4'd9: fnd_data = 8'h90;
            default: fnd_data = 8'hFF;
        endcase
    end

endmodule
```



1 | fnd_controller module은 adder에서 받아온 sum을 입력으로 하여 fnd_digit, fnd_data를 출력으로 내보낸다.

fnd_digit은 이는 왼쪽 사진을 보면 알 수 있듯이 W4, V4, U4, U2 Pin에 High(1) 신호가 들어오면 트랜지스터가 비활성화 되어 해당 display에 불이 켜지지 않는 구조를 가지고 있다. 반대로 Low(0) 신호가 들어오면 display에 불이 켜진다.
이번 설계에선 일의 자리만 켜지는 것을 의도 했기 때문에 fnd_digit의 값을 4'b1110으로 할당하였다.

fnd_data는 bcd module에서 나오는 출력 fnd_data를 받아 출력으로 사용한다.



2 | bcd module은 4비트 bcd 값을 7-segment display에 숫자 형태로 표시하기 위한 module이다.
간단하게 설명하자면, 앞에서 8bit로 넘어온 sum 값에서 4bit만 사용하여 case문을 진행하였다. 예를 들어, bcd의 값이 0인 case에서는 fnd_data가 8'hC0(8'b1100_0000)이 된다.
이 역시 fnd_digit과 마찬가지로 1일 때 불이 꺼지는 구조를 가지고 있다.
fnd_data는 DP, G, F, E, D, C, B, A에 맞게 fnd_data[7] ~ fnd_data[0]을 순서대로 Pin할당을 시켜주었고, 그래서 2'b1100_0000이 되면 DP, G를 제외한 모든 곳에 불이 켜지면서 숫자 0의 형태로 나타나게 된다.
나머지 case에 대한 fnd_data값도 정해놓고 그 외의 경우에는 8'hFF로 모든 곳의 불이 꺼지게 된다.

2 | RTL code 설명 (tb_8bit_adder_fnd.v)

RTL code 설명 (tb_8bit_adder_fnd.v)

```
`timescale 1ns / 1ps

module tb_8bit_adder_fnd ();
    reg  [7:0] a;
    reg  [7:0] b;
    wire [3:0] fnd_digit;
    wire [7:0] fnd_data;
    wire       c;

    integer i = 0, j = 0;

    top_adder dut (
        .a        (a),
        .b        (b),
        .fnd_digit(fnd_digit),
        .fnd_data (fnd_data),
        .c        (c)
    );

    initial begin
        #0;
        a = 8'b00;
        b = 8'b00;
        #10;
        for (i = 0; i < 256; i = i + 1) begin
            for (j = 0; j < 256; j = j + 1) begin
                a = i;
                b = j;
                #10;
            end
        end
        $stop;
        #100;
        $finish;
    end
endmodule
```

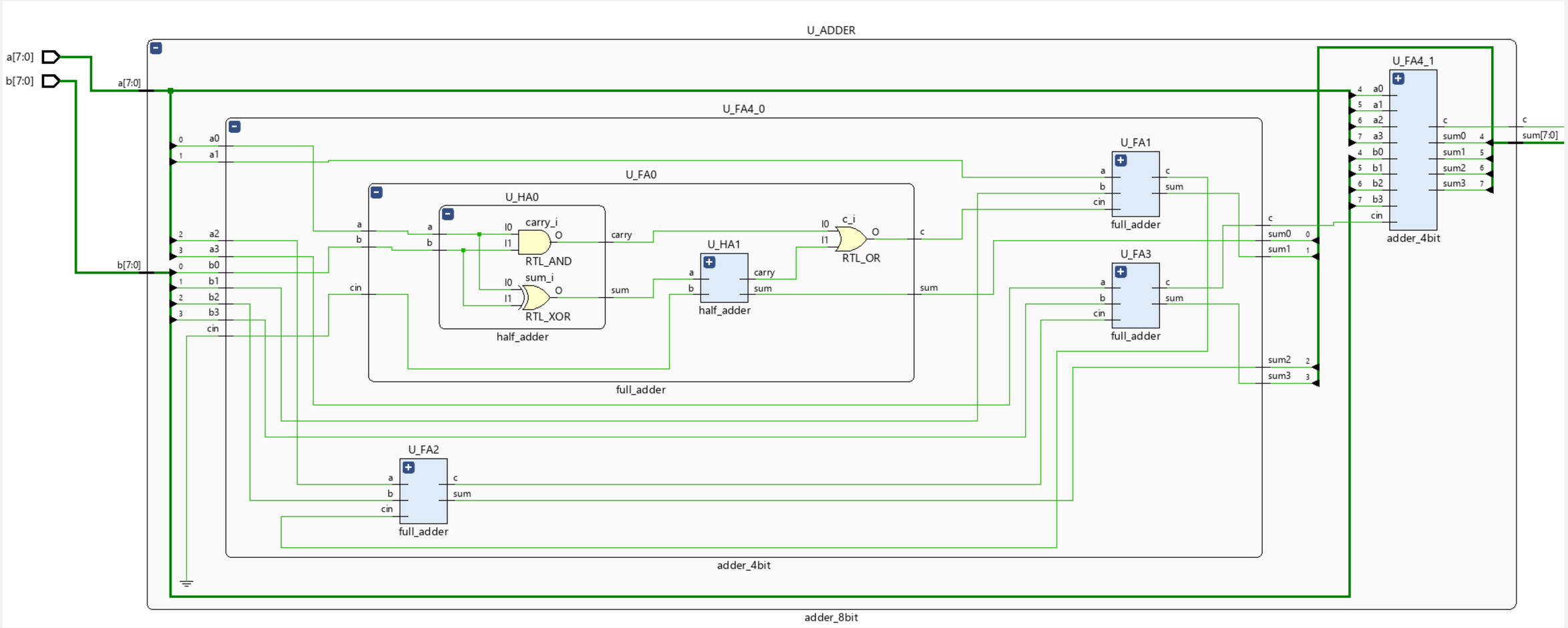
1 | Testbench 파일은 쉽게 말해 내가 설계한 회로가 설계한 대로 동작하는지
가상의 입력을 넣어 출력을 확인하는 Simulation을 위한 파일이다.

2 | DUT(Design Under Test)의 입력인 a, b를 reg로 선언했고,
출력인 fnd_digit, fnd_data, c를 wire로 선언했다.
a, b는 직접 값을 넣어주기 때문에 그 값을 저장하기 위해 reg로 선언했고,
fnd_digit, fnd_data, c는 a, b에 의한 결과 값이고, 이를 연결하기 위한 wire로 선언했다.

3 | Initial begin ~ end 로 delay와 for문을 이용해 simulation을 위한 코드를 작성했다.
simulation시 X값을 방지하기 위해 a, b의 값을 0으로 초기화 시켜주었다.
for문을 이용해 8bit인 a, b의 모든 경우의 수를 계산하며 그 사이에 10ns의 delay를 주었다.
모든 경우가 끝나면 simulation을 멈추고, 100ns 뒤에 $finish를 사용하여 simulation을
완전히 끝냈다.

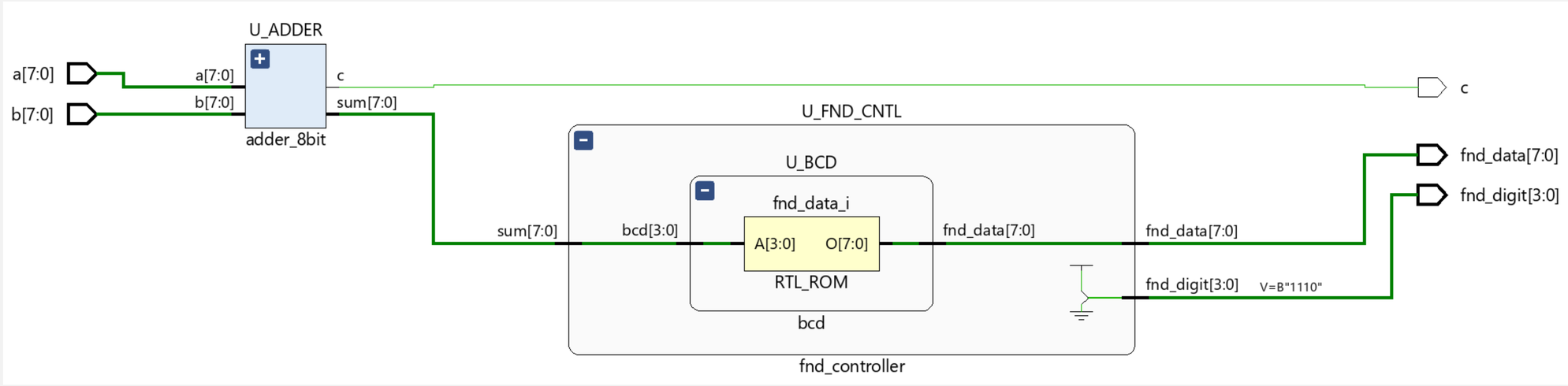
3 | RTL Schematic

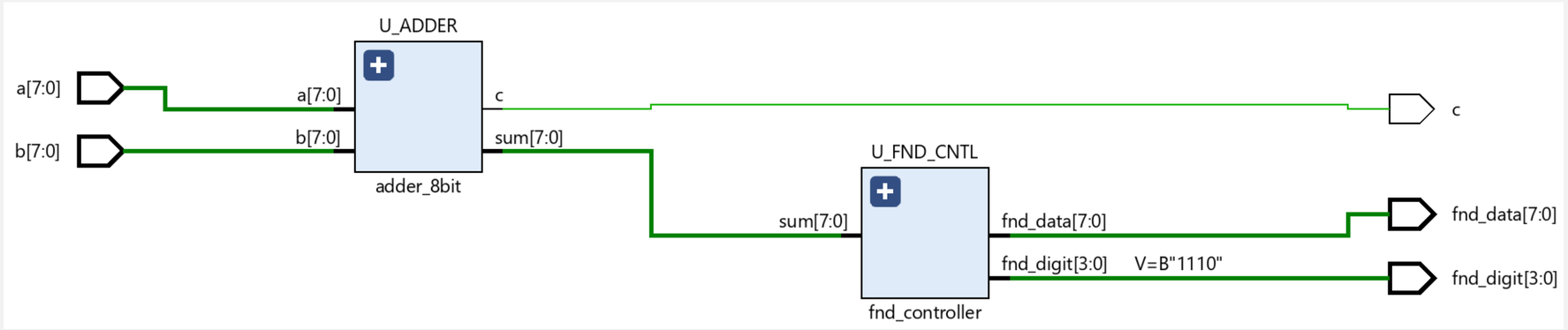
RTL Schematic



1 | 위의 사진은 8bit_full_adder의 schematic 모습이다. 8bit 입력 a, b가 4bit full adder 두 곳에 잘 연결되어 있는 모습을 보인다.
그리고 출력은 c와 8bit sum으로 연결되어 있다. 사진을 보면 adder의 내부는 여러 개의 adder들로 구성되어 있다.

RTL Schematic



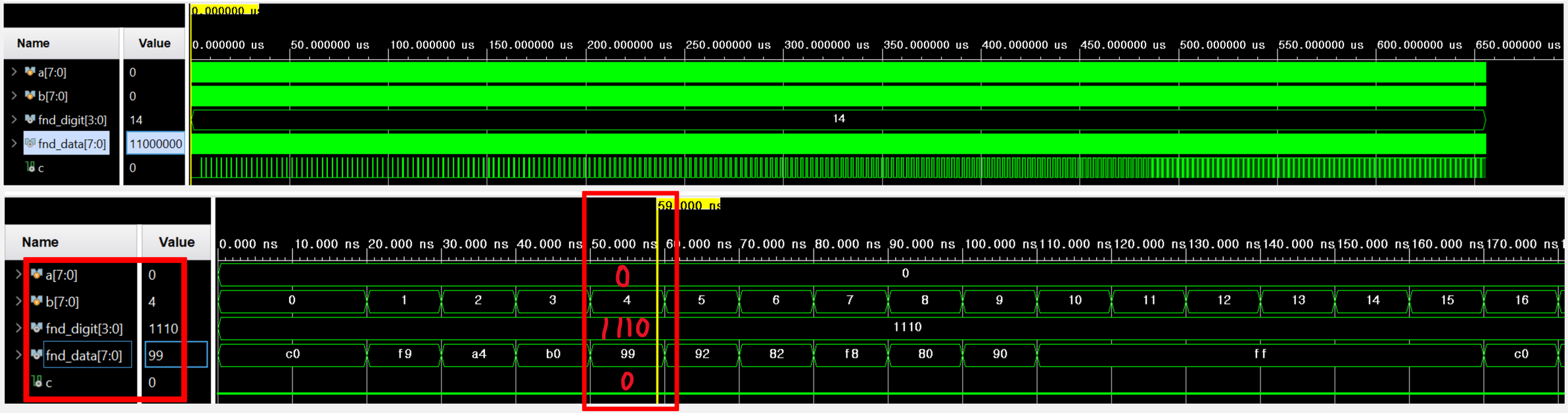


1 | adder에서 나온 c는 그대로 출력으로 나오게 되고, 8bit sum은 fnd_controller로 연결된다. 그 안에는 4bit bcd입력과 bcd module을 통해 fnd_data가 출력으로 나오게 된다.

2 | 아래 사진은 전체 RTL schematic의 모습이다. 입출력 신호가 잘 연결되었는지는 simulation을 통해 확인할 수 있었다.

4 | Simulation Waveform

Simulation Waveform

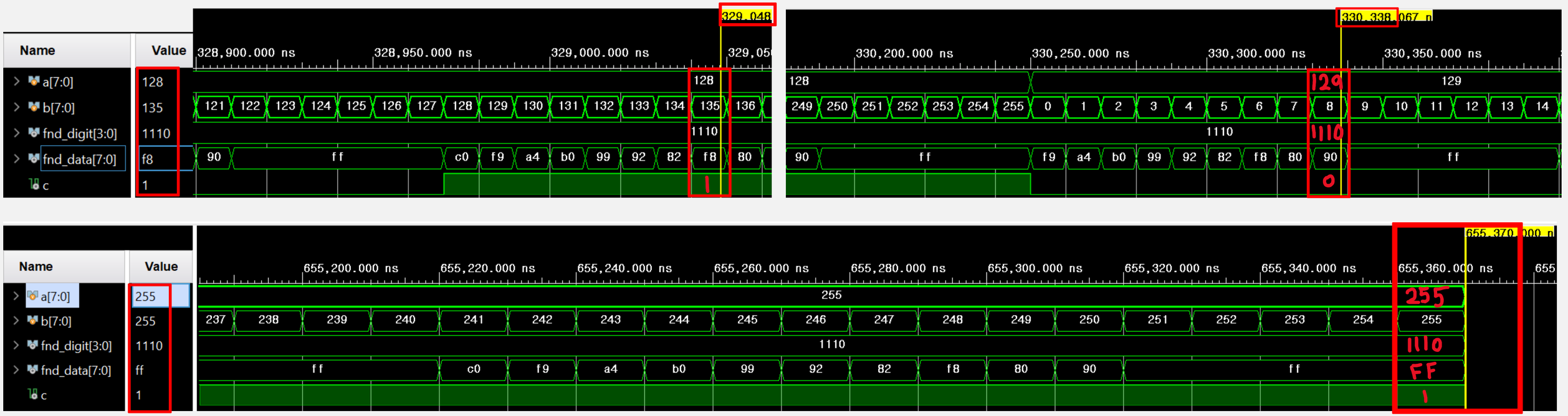


1 | 위의 사진은 simulation을 실행했을 때 나오는 전체 waveform의 모습이다.

2 | 59ns에서 simulation 분석 :
a = 0, b = 4, fnd_digit = 4'b1110, fnd_data = 8'h99(8'b1001_1001), c = 0
a(0000_0000) + b(0000_0100) = sum(0000_0100)의 값이 하위 4bit bcd(0100)가 되어 이는 4'd4이고,
fnd_data가 8'h99가 나오게 된다. 설계한대로 simulation이 올바르게 나오는 모습을 보인다.
fnd_digit은 이번 설계에서 일의 자리만 보도록 의도했기 때문에 simulation이 끝날 때까지 4'b1110을 유지한다.
이번 연산에서 carry는 일어나지 않았으므로 c의 값이 0으로 파형에 나타난다.

```
always @(bcd) begin
    case (bcd)
        4'd0: fnd_data = 8'hC0;
        4'd1: fnd_data = 8'hF9;
        4'd2: fnd_data = 8'hA4;
        4'd3: fnd_data = 8'hB0;
        4'd4: fnd_data = 8'h99;
        4'd5: fnd_data = 8'h92;
        4'd6: fnd_data = 8'h82;
        4'd7: fnd_data = 8'hF8;
        4'd8: fnd_data = 8'h80;
        4'd9: fnd_data = 8'h90;
        default: fnd_data = 8'hFF;
    endcase
end
```

Simulation Waveform



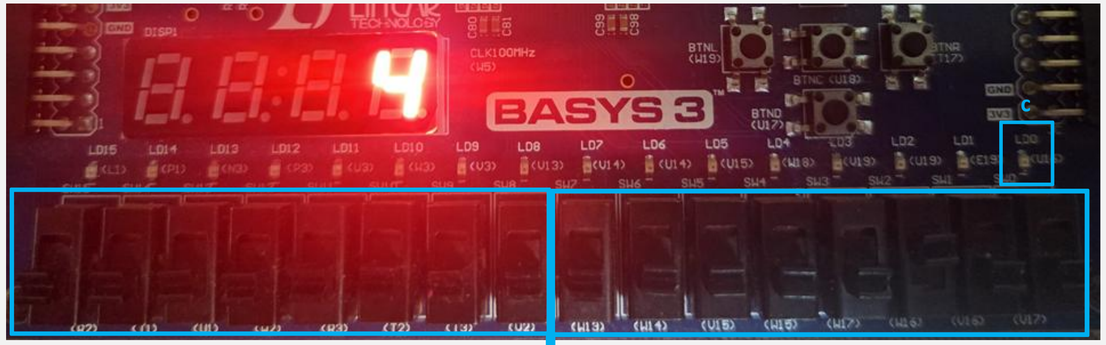
1 | 329,000ns에서 simulation 분석 :
a = 128, b = 135, fnd_digit = 4'b1110, fnd_data = 8'hf8(8'b1111_1000), c = 1
a(1000_0000) + b(1000_0111) = sum(0000_0111)의 값이 하위 4bit bcd(0111)가 되어 이는 4'd7이고,
fnd_data가 8'hf8이 나오게 된다. 설계한대로 simulation이 올바르게 나오는 모습을 보인다.
이번 연산에서는 carry가 발생하였기 때문에 c의 값이 1으로 파형에 나타난다.

655,000ns에서 simulation 분석 :
a = 255, b = 255, fnd_digit = 4'b1110, fnd_data = 8'hf8(8'b1111_1000), c = 1
a(1111_1111) + b(1111_1111) = sum(1111_1110)의 값이 하위 4bit bcd(1110)가 되어 이는 4'd14이고,
bcd case에 값이 존재하지 않아 default로 fnd_data가 8'hff가 나오게 된다.
설계한대로 simulation이 올바르게 나오는 모습을 보인다.
이번 연산에서는 carry가 발생하였기 때문에 c의 값이 1으로 파형에 나타난다.

```
always @(bcd) begin
    case (bcd)
        4'd0: fnd_data = 8'hC0;
        4'd1: fnd_data = 8'hF9;
        4'd2: fnd_data = 8'hA4;
        4'd3: fnd_data = 8'hB0;
        4'd4: fnd_data = 8'h99;
        4'd5: fnd_data = 8'h92;
        4'd6: fnd_data = 8'h82;
        4'd7: fnd_data = 8'hF8;
        4'd8: fnd_data = 8'h80;
        4'd9: fnd_data = 8'h90;
        default: fnd_data = 8'hFF;
    endcase
end
```

5 | Result

Result



0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
a[7]	a[6]	a[5]	a[4]	a[3]	a[2]	a[1]	a[0]	b[7]	b[6]	b[5]	b[4]	b[3]	b[2]	b[1]	b[0]

0000_0000 + 0000_0100 = 0000_0100(4) -> bcd(0100) ->
bcd(4) -> fnd_data = 8'h99(8'b1001_1001) 이므로
DP, E, D, A를 제외한 모든 곳에 불이 켜지고 숫자 4의 형태를 가진다.
추가로 carry는 발생하지 않아 LD0에는 불이 켜지지 않는다.

0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
a[7]	a[6]	a[5]	a[4]	a[3]	a[2]	a[1]	a[0]	b[7]	b[6]	b[5]	b[4]	b[3]	b[2]	b[1]	b[0]

1000_0000 + 1000_0111 = 0001_0000_0111(263) -> bcd(0111) ->
bcd(7) -> fnd_data = 8'hf8(8'b1111_1000) 이므로
DP, G, F, E, D를 제외한 모든 곳에 불이 켜지고 숫자 7의 형태를 가진다.
추가로 carry가 발생하여 LD0에 불이 켜진다.

1	0	0	0	0	0	0	1	0	0	0	0	1	0	0	0
a[7]	a[6]	a[5]	a[4]	a[3]	a[2]	a[1]	a[0]	b[7]	b[6]	b[5]	b[4]	b[3]	b[2]	b[1]	b[0]

1000_0001 + 0000_1000 = 1000_1001(137) -> bcd(1001) ->
bcd(9) -> fnd_data = 8'h90(8'b1001_0000) 이므로
DP, E를 제외한 모든 곳에 불이 켜지고 숫자 9의 형태를 가진다.
추가로 carry는 발생하지 않아 LD0에는 불이 켜지지 않는다.

0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
a[7]	a[6]	a[5]	a[4]	a[3]	a[2]	a[1]	a[0]	b[7]	b[6]	b[5]	b[4]	b[3]	b[2]	b[1]	b[0]

1111_1111 + 1111_1111 = 0001_1111_1110(510) -> bcd(1110) ->
bcd(14) -> fnd_data = 8'hff(8'b1111_1111) 이므로
DP, G, F, E, D, C, B, A 모든 곳에 불이 꺼지게 된다.
추가로 carry가 발생하여 LD0에 불이 켜진다.

감사합니다