디지털시스템설계 실습 보고서

4. 이진수 연산

20210716 최대현

1. 실습 개요

이번 실습에서는 이진수의 연산을 다룬다. 이진수 덧셈에 사용되는 반가산기를 구현하고, 이를 활용하여 전가산기를 구현한다. 또한 전가산기를 가지고 5-bit 리플 가산/감산기를 구현하고 곱셈기를 구현한다.

2. 이론적 배경

1) 반가산기

반가산기(half adder)는 이진수의 한자리수를 연산하고, 자리올림수 출력(carry out)에 따라 자리올림수를 출력하는 연산을 수행한다. Input은 1bit 수 A와 B를 받고, output으로 C(carry-out)와 S(Sum)을 출력한다.

2) 전가산기

전가산기(full adder)는 이진수의 한 자릿수를 연산하고, 하위의 자리올림수 입력을 포함하여 출력하는 연산을 수행한다. 하위의 자리올림수 출력을 상위의 자리올림수 입력에 연결함으로써 두 자릿수 이상의 이진수 덧셈이 가능해진다. 하나의 전가산기는 두개의 반가산기와 한 개의 OR gate로 제작할 수 있다.

3) N 비트 리플 가산기, 감산기

전가산기 여러 개를 이용해서 임의의 비트 수를 더하는 회로를 만들 수 있다. 전가산기의 3가지 입력 중 하나인 Cin을 이전 비트의 Cout으로 받는 형식으로 제작 가능하다. Carry-out이 ripple처럼 다음으로 옮겨간다고 하여 ripple carry 가산기라고 한다. 이 때 첫 번째 전가산기의 Cin은 0이고, 따라서 반가산기로 대체 가능하다.

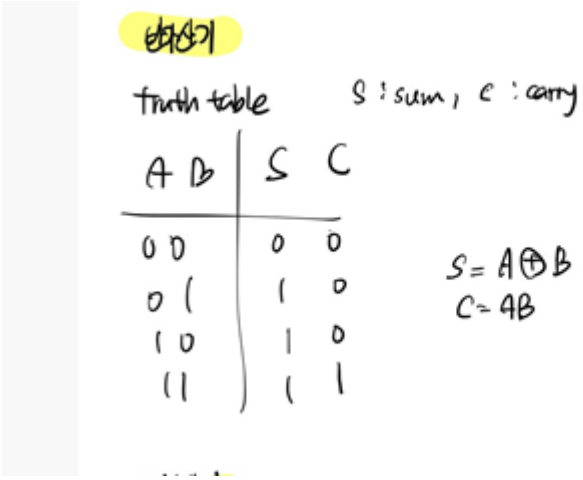
비슷한 원리로 임의의 비트 수를 빼는 감산기도 만들 수 있다. 리플 전가산기 형태에서 빼줄 수에 2의 보수를 취하기 위해 그 수의 각 자릿수에 not 연산을 취하고, 최하위 비트의 Cin을 1로 설정한다. (어떤 수 B의 2의 보수는 'B+1이라고 할 수 있기 때문이다.)

4) MXN 이진 곱셈기

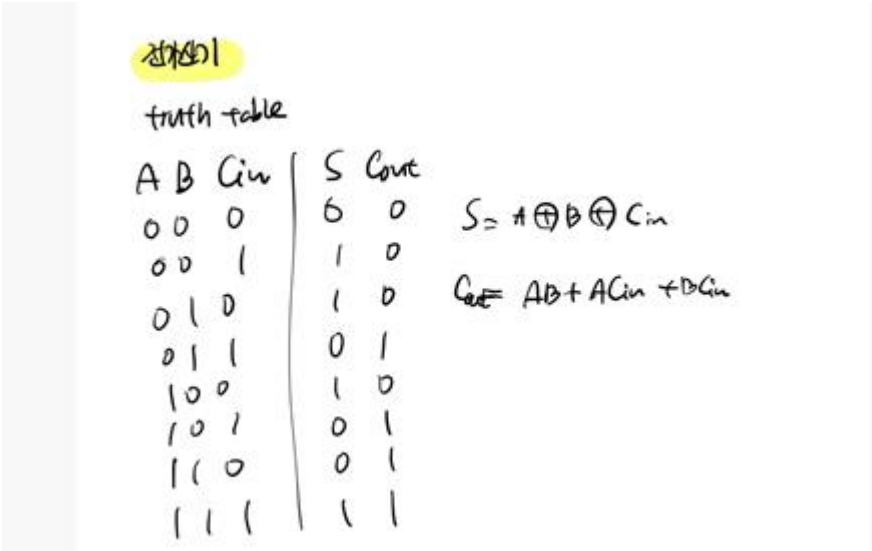
선택 신호에 따라 여러 입력 신호 중 하나를 골라 출력하는 장치이다. 주로 2^n개의 입력 신호 중 한 개를 선택하여 출력한다. 2^n개의 data input에 대해 n bit control input으로 조절하여 사용한다.

3. 실험 전 준비

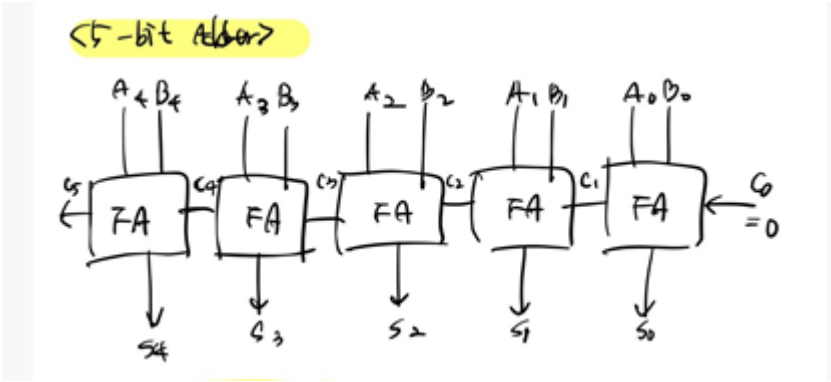
1) 반가산기의 진리표, 식



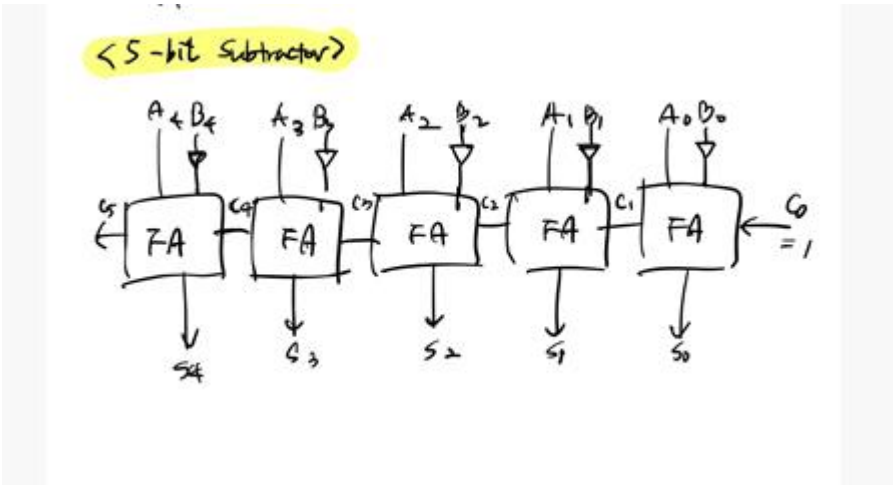
2) 전가산기의 진리표, 식



4) 5-bit ripple Adder

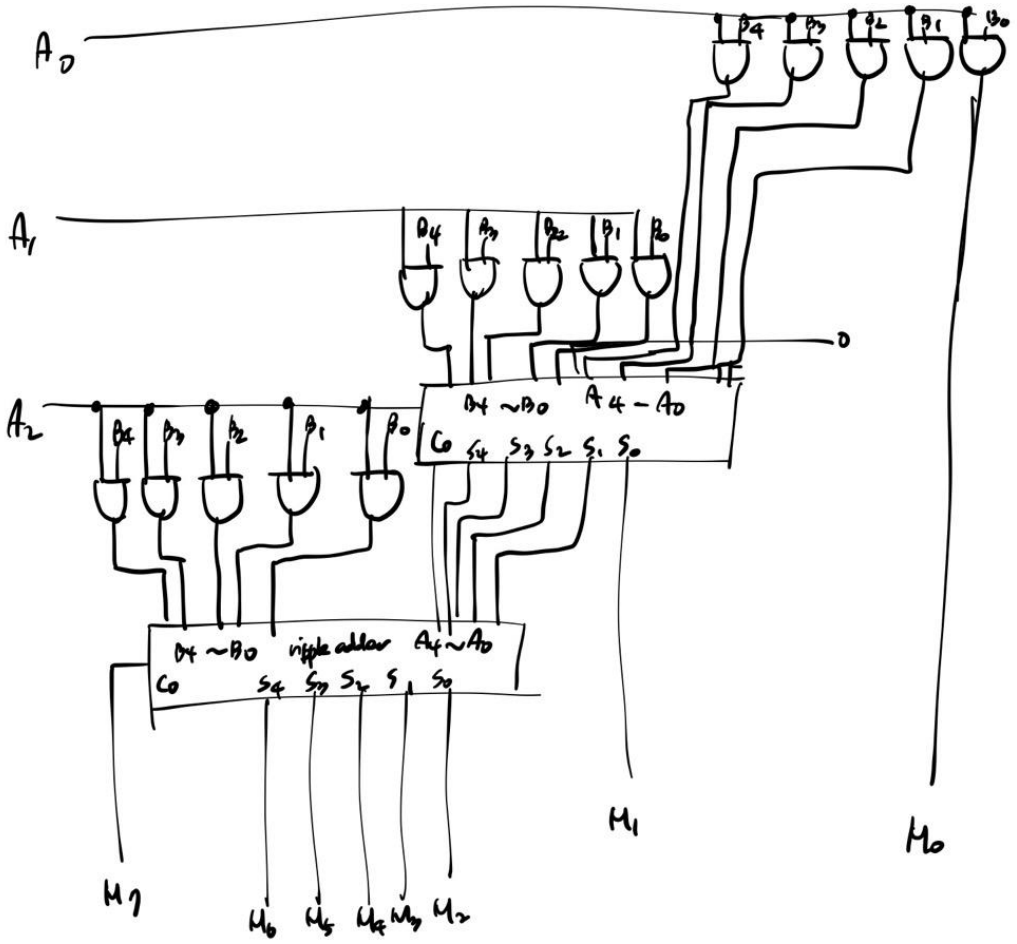


5) 5-bit ripple Subtractor



6) 5X3 binary multiplier

<5x3 이진곱셈기>



4. 실험

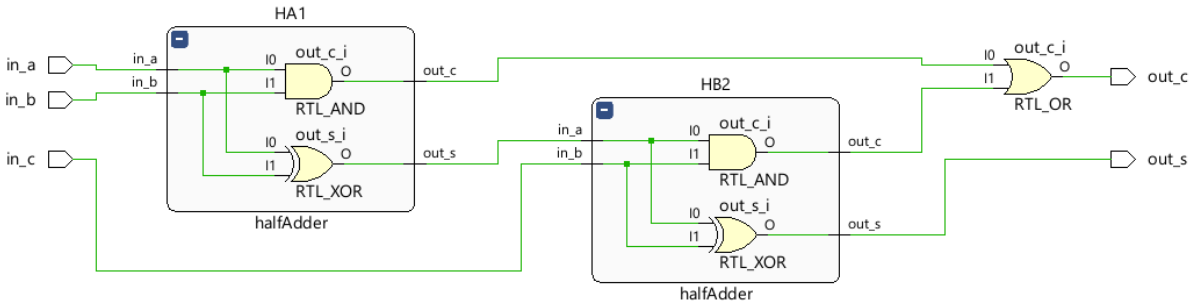
본 실험은 Verilog 프로그래밍으로 회로를 구현하며 진행되었다.

4-1. lab4_1.v

반가산기와 전가산기를 구현한 코드이다.

```
module halfAdder(
    input in_a,
    input in_b,
    output out_s,
    output out_c
    );

    //////////////////////////
    /* Add your code here */
    xor(out_s, in_a, in_b); //s= a xor b
    and(out_c, in_a, in_b); //c = ab
    //////////////////////////

module fullAdder(
    input in_a,
    input in_b,
    input in_c,
    output out_s,
    output out_c
    );

    //////////////////////////
    /* Add your code here */
    wire s1, c1, c2;
    halfAdder HA1(in_a, in_b, s1, c1);
    halfAdder HB2(s1, in_c, out_s, c2);
    or(out_c, c1, c2);
```

<source code>



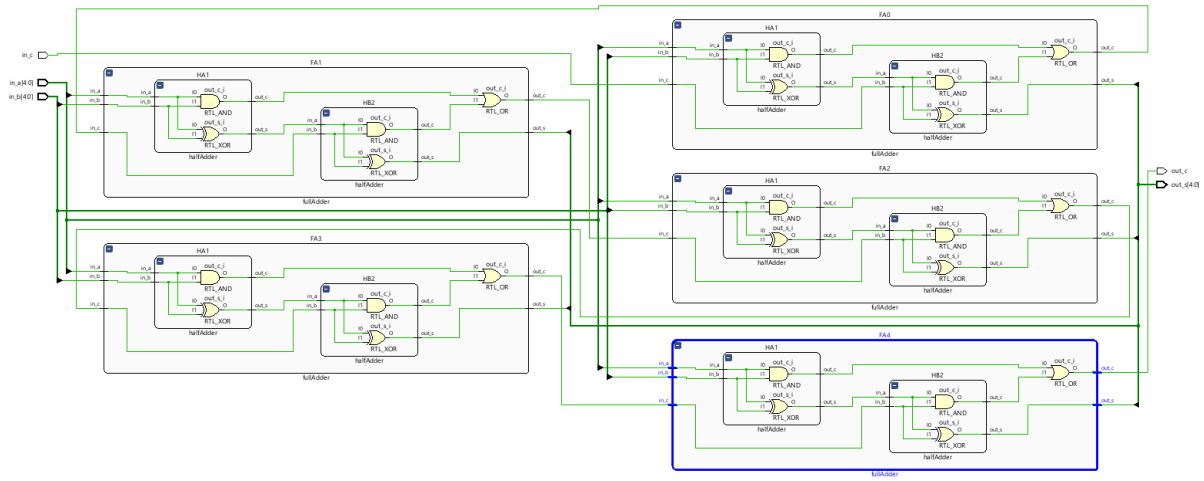
<Schematic 기능으로 출력한 회로도>

4-2. lab4_2.v

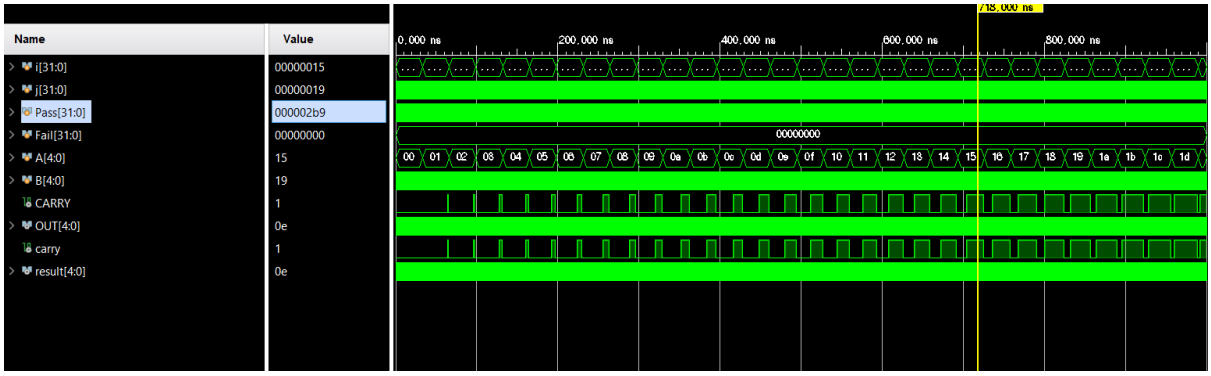
5비트 리플 가산기를 구현한 코드이다.

```
wire c[3:0];

fullAdder FA0(in_a[0], in_b[0], in_c, out_s[0], c[0]); // little endian 방식
fullAdder FA1(in_a[1], in_b[1], c[0], out_s[1], c[1]);
fullAdder FA2(in_a[2], in_b[2], c[1], out_s[2], c[2]);
fullAdder FA3(in_a[3], in_b[3], c[2], out_s[3], c[3]);
fullAdder FA4(in_a[4], in_b[4], c[3], out_s[4], out_c);
```

<source code>



<Schematic 기능으로 출력한 회로도>



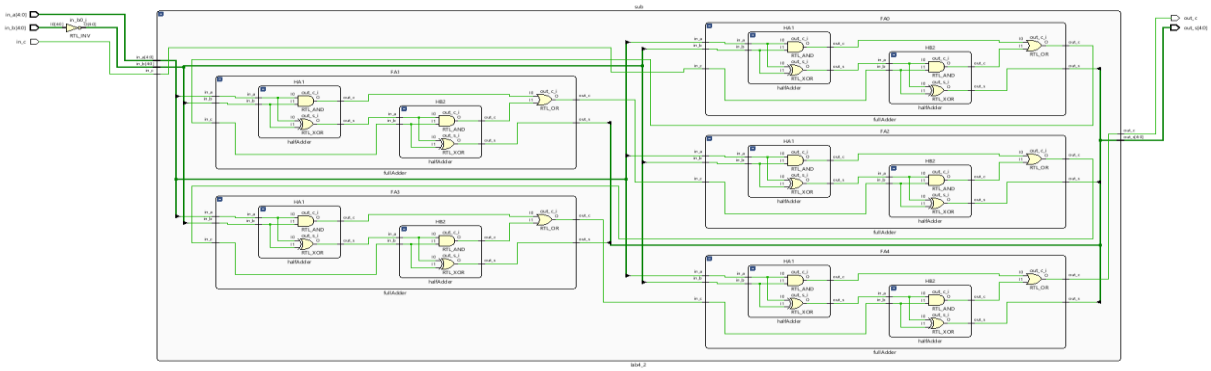
<simulation 결과>

4-3. lab4_3.v

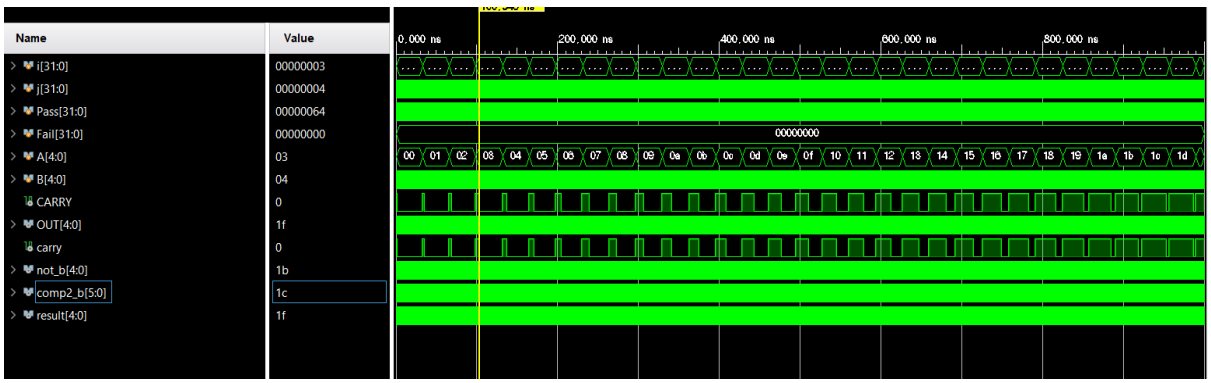
5비트 리플 감산기를 구현한 코드이다.

```
module lab4_3(
    input [4:0] in_a,
    input [4:0] in_b,
    input in_c,
    output [4:0] out_s,
    output out_c
    );

    ////////////////////////
    /* Add your code here */
    lab4_2 sub(in_a, ~in_b, in_c, out_s, out_c);
    ////////////////////////
```

<source code>



<Schematic 기능으로 출력한 회로도>



<simulation 결과>

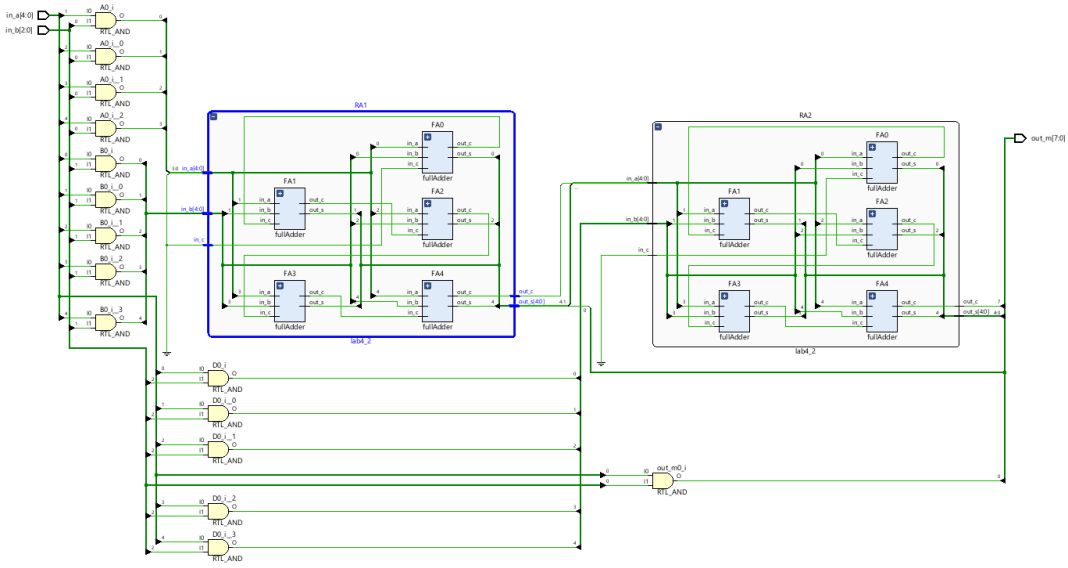
4-4. lab4_4.v

5X3 이진 곱셈기를 구현한 코드이다.

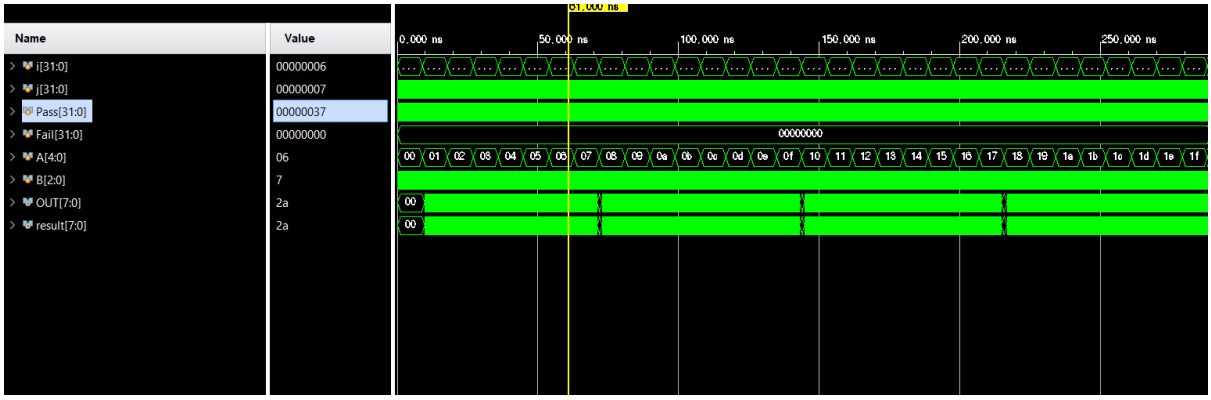
```
/* Add your code here */
 assign out_m[0]=in_a[0]&in_b[0]; // 최하위 비트, 별도의 carry가 필요하지 않음

 wire [4:0] A;
 wire [4:0] B;


 assign A[0]=(in_a[1]&in_b[0]);
 assign A[1]=(in_a[2]&in_b[0]);
 assign A[2]=(in_a[3]&in_b[0]);
 assign A[3]=(in_a[4]&in_b[0]);
 assign A[4]=0;

 assign B[0]=(in_a[0]&in_b[1]);
 assign B[1]=(in_a[1]&in_b[1]);
 assign B[2]=(in_a[2]&in_b[1]);
 assign B[3]=(in_a[3]&in_b[1]);
 assign B[4]=(in_a[4]&in_b[1]);

 wire c1;
 wire [4:0] s1; //위  ripple adder의 sum값 저장

 wire cin1;
 assign cin1 = 0;
 lab4_2 RA1 (A, B, cin1, s1, c1); // 그림에서 위에 해당하는 ripple adder

 assign out_m[1]=s1[0]; //두 번째로 작은 비트 설정
```

```
wire c1;
wire [4:0] s1; //위  ripple adder의 sum값 저장

wire cin1;
assign cin1 = 0;
lab4_2 RA1 (A, B, cin1, s1, c1); // 그림에서 위에 해당하는 ripple adder

assign out_m[1]=s1[0]; //두 번째로 작은 비트 설정

wire [4:0] C;

assign C[0]=s1[1]; // 위의 ripple adder 값을 끌어씀
assign C[1]=s1[2];
assign C[2]=s1[3];
assign C[3]=s1[4];
assign C[4]=c1;

wire [4:0] D;

assign D[0]=(in_a[0]&in_b[2]);
assign D[1]=(in_a[1]&in_b[2]);
assign D[2]=(in_a[2]&in_b[2]);
assign D[3]=(in_a[3]&in_b[2]);
assign D[4]=(in_a[4]&in_b[2]);


wire c2;///carry 저장할 wire
wire [4:0] s2; //아래 ripple adder의 sum값 저장
```

<source code>



<Schematic 기능으로 출력한 회로도>



<simulation 결과>

5. 결론

Lab 4_1에서는 halfAdder 모듈을 만들고, 이를 이용해서 fullAdder를 구현할 수 있었다. Lab4_2에서는 구현한 fullAder 여러 개로 5-bit ripple adder를 만들어볼 수 있었고, 이를 응용하여 lab4_3에서는 5-bit ripple subtractor를 구현했다. 4_4에서는 마지막으로 4_2에서 구현한 5-bit ripple adder로 5x3 multiplier를 구현했다.

6. 고찰

이번 실습에서는 이론적으로 학습했던 전가산기와 리플 가산기 등 2진수 연산기를 구현해볼 수 있었다. 또한, 이를 활용하여 다소 생소했던 이진 곱셈기까지 구현해볼 수 있었다. 이번 실습은 지금까지의 실습들과 다르게 각 lab 문제들이 유기적인 구조로 구성되어 있어서, lab 4_2를 코딩할 때 오류가 발생해서 그 뒤를 진행하지 못했던 좋지 않은 기억이 있다. 내가 만든 module을 가지고 새로운 module을 만들어본다는 점에서 유익한 시간이었다.