컴퓨터구조실험

과제 이름: Booth algorithm

담당교수: 이성원

학 과: 컴퓨터정보공학부

학 번: 2019202021

이 름: 정 성 엽

제출일: 2023/3/30

1. Introduction
   A. 과제 소개
      - Verilog로 Booth algorithm 을 구현하고, Icarus Verilog로 input 값을 바꿔가며 컴파일 후에 gtkwave로 파형을 확인하며 해당 결과값에 대한 분석을 한다. 또한 Logisim-evolution을 이용하여 booth algorithm을 설계하고, 입력을 넣어보며 동작 결과를 확인하고 동작 결과가 왜 그렇게 나오는지 설명한다.

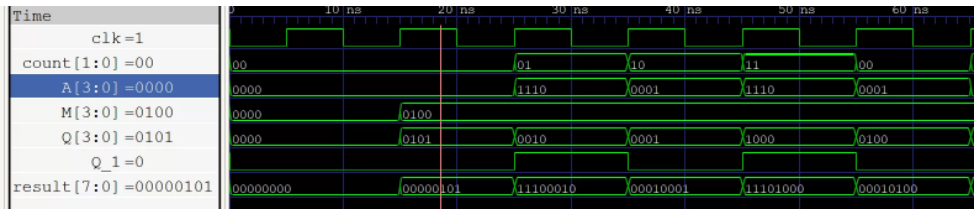
2. 결과화면
   A. Verilog
      - 코드 확인

```
always @(posedge clk or negedge reset) begin
    if(~reset)
    begin
        A <= 4'b0;
        M <= 4'b0;
        Q <= 4'b0;
        Q_1 <= 1'b0;
        count <= 2'b0;
    end

    else if(~start)
    begin
        A <= 4'b0;
        M <= input1;
        Q <= input2;
        Q_1 <= 1'b0;
        count <= 2'b0;
    end

    else
    begin
        case ({Q[0],Q_1})
            2'b0_1: {A,Q,Q_1}<={add[3], add, Q};
            2'b1_0: {A,Q,Q_1}<={sub[3], sub, Q};
            default: {A,Q,Q_1}<={A[3], A, Q};
        endcase

        count = count +1'b1;
    end
end
```

clk가 1로 되거나 reset이 0이 될 때 작동하는 코드로 만약 reset이 0이라면 A, M, Q, Q_1 모두 0으로 초기화 한다. 아니고 만약 start가 0이라면 M과 Q에 각각 input1과 input2를 저장하고 다른 A, Q, Q_1은 0으로 설정한다. 아니라면 만약 {Q[0],Q_1} 조합이 2'b0_1이라면 더하고 arithmetic shift 연산, 2'b1_0이라면 빼고 arithmetic shift 연산, 둘 다 아니라면 shift 연산만 진행한다. count는 1씩 증가 시킨다.

      - 파형 확인

1) 4 * 5

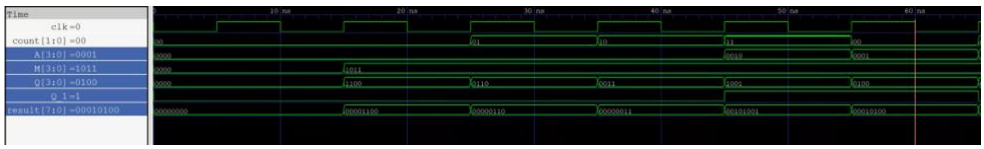


4와 5를 곱하였을 때 파형의 결과이다. M은 multiplicand이고 4, Q는 multiplier로 5이다. 이 booth 알고리즘 과정을 나타내면 아래와 같다

iteration	step	M	A	Q	Q_1
0		0100	0000	0101	0
1	{Q[0],Q_1}={1,0} , sub and shift	0100	1100	0101	0
		0100	1110	0010	1
2	{Q[0],Q_1}={0,1}, add and shift	0100	0010	0010	1
		0100	0001	0001	0
3	{Q[0],Q_1}={1,0} , sub and shift	0100	1101	0001	0
		0100	1110	1000	1
4	{Q[0],Q_1}={0,1}, add and shift	0100	0010	1000	1
		0100	0001	0100	0

위 과정이 진행되어 gtkwave와 같은 파형이 나타났다. gtkwave에서 나타난 값은 각 iteration 별로 마지막 과정의 값들이다. 이는 위의 표에서 회색으로 나타내었다. 또한 multiplier은 4bit 이므로 1~4번 과정이 끝나면 결과를 알 수 있다. 다만 tb_Booth.v에서는 10ns+10ns+640ns=660ns 동안 진행되어도 마지막에 같은 결과가 나오는데 이는 정말 우연의 일치로 00010100을 shift right 하였을 때 A가 0000 Q가 0101 나오는 경우가 있기 때문에 60ns 동안 진행하였을 때 660ns 동안 11번 반복되어 결과가 나타난 것이다.

2) -5 * -4



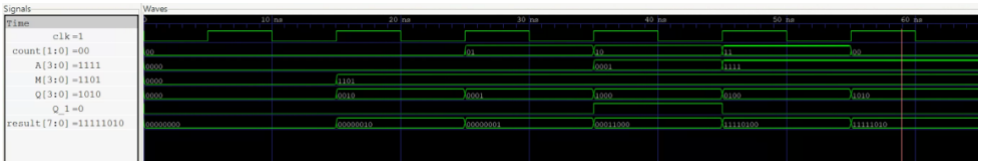
-5와 -4를 곱하였을 때 파형의 결과이다. 이 booth 알고리즘 과정을 나타내면 아래와 같다.

iteration	step	M	A	Q	Q_1
0		1011	0000	1100	0
1	{Q[0],Q_1}={0,0} , shift only	1011	0000	0110	0
2	{Q[0],Q_1}={0,0}, shift only	1011	0000	0011	0
3	{Q[0],Q_1}={1,0} , sub and shift	1011	0101	0011	0
		1011	0010	1001	1
4	{Q[0],Q_1}={1,1}, shift only	1011	0001	0100	1

-5와 -4를 곱하였을 때도 위 과정과 같은 순서로 파형이 정상적으로 나옴을 확인 하였다. 이진수 00010100은 십진수로 20이므로 정답이다. gtkwave에서 파형을 확인하였을 때 60ns까지의 값은 정상적으로 나오지만 660ns 같은 경우 다

른 값이 나오는 것을 확인할 수 있다.

3) -3 * 2

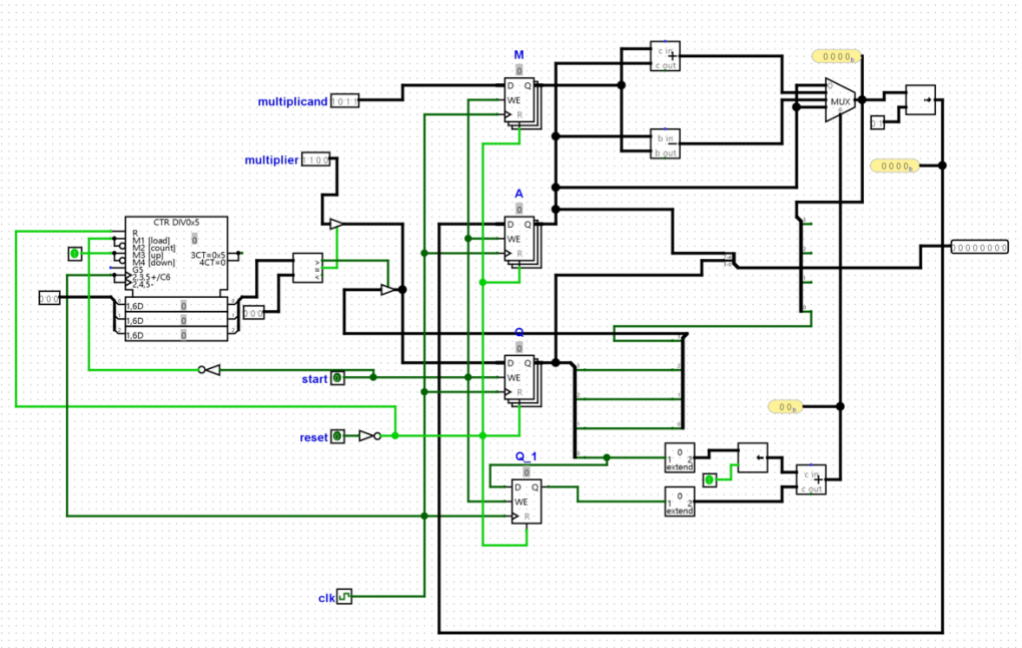


-3와 2를 곱하였을 때 파형의 결과이다. 이 booth 알고리즘 과정을 나타내면 아래와 같다.

iteration	step	M	A	Q	Q_1
0		1101	0000	0010	0
1	{Q[0],Q_1}={0,0} , shift only	1101	0000	0001	0
2	{Q[0],Q_1}={1,0}, sub	1101	0011	0001	0
	and shift	1101	0001	1000	1
3	{Q[0],Q_1}={0,1} , add	1101	1110	1000	1
	and shift	1101	1111	0100	0
4	{Q[0],Q_1}={0,0}, shift only	1101	1111	1010	0

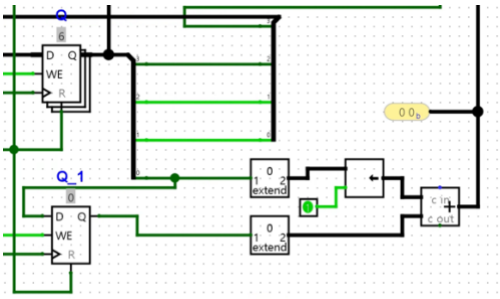
-3과 2를 곱하였을 때도 위 과정과 같은 순서로 파형이 정상적으로 나옴을 확인하였다. 이진수 11111010은 십진수로 -6이므로 정답이다.

B. Logisim-evolution

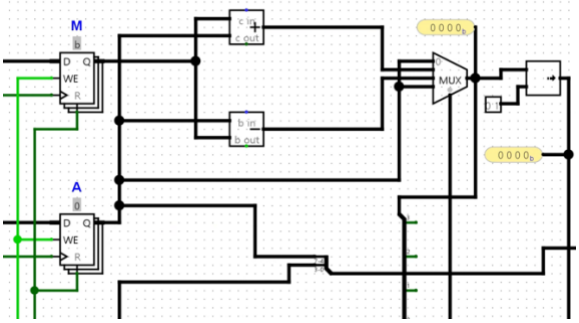


Logisim-evolution으로 2-Radix Booth multiplication을 구현한 회로이다. multiplicand는 M으로 바로 들어가지만 multiplier는 Counter에서 0일 때만 Q로 이동하도록 Controlled Buffer를 사용하여 값을 전달하고 0보다 큰 경우에는 새로운 Q값이 Q로

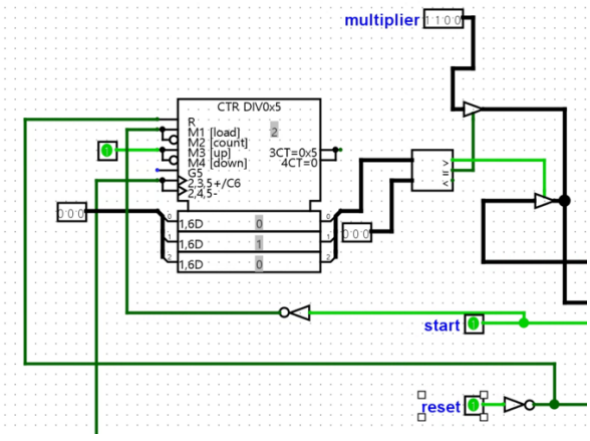
입력되도록 회로를 구성하였다. clk는 각 register와 counter로 입력되도록 하여 모두 rising edge에서 동작하도록 하였다. start는 각 register의 we로 연결되어 1로 set 되었을 때만 작동하도록 하였으며 counter로는 not 게이트를 두어 0으로 Reset 하였을 때는 load로 1로 set 되었을 때는 count로 두어 숫자가 증가하도록 하였다. reset은 falling edge에서 작동해야하므로 시작에 not 게이트를 두어 각 register와 counter에 연결하였다.



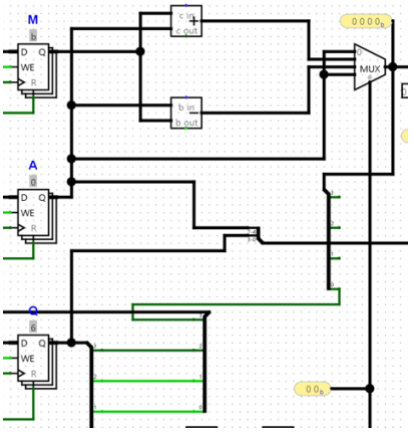
위의 사진을 보면 Q와 Q_1을 2bit로 합치기 위해 1bit를 2bit로 합치기 위해 extender를 사용했으면 extend하는 경우 0으로만 extend하도록 하였다. Q[0]은 extend 이후 left shift를 진행하였고 이 둘을 더하여 2bit로 만들었다.



위에서 만들어진 2bit 값은 이번 사진의 MUX로 들어가며 00 또는 11일 경우 A값이 그대로 나가고 01인 경우 A와 M의 합의 값이 나가고 10인 경우 A에서 M을 뺀 값이 나가도록 하였으며 나간 값은 1만큼 Arithmetic right shift 하도록 설계하였다.



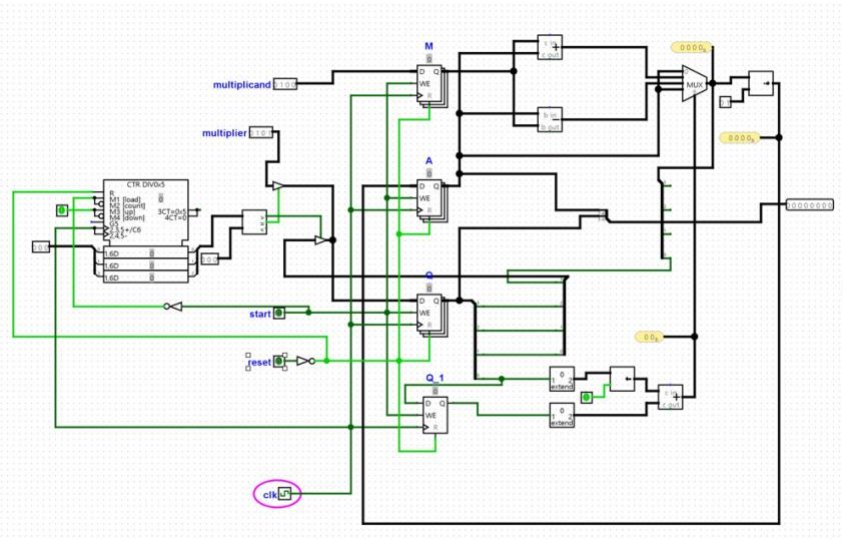
위에서 말했던 counter에서 0일 때만 multiplier 값이 나가도록 하였고 그 이상이라면 Q값이 나가도록 설정하였다.



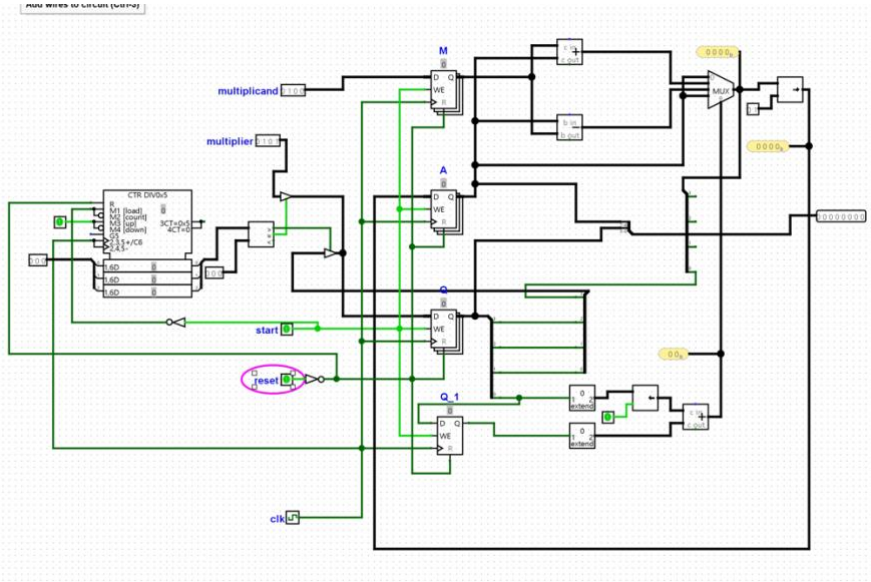
Verilog 코드에서는 A와 M을 shift right 하도록 되어있지만 해당 부분을 둘로 나누었고 spliter를 활용하여 이전 A[0]를 Q[3]으로 두고 이전 Q[3:1]을 Q[2:0]을 두었다.

1) 4 * 5의 과정

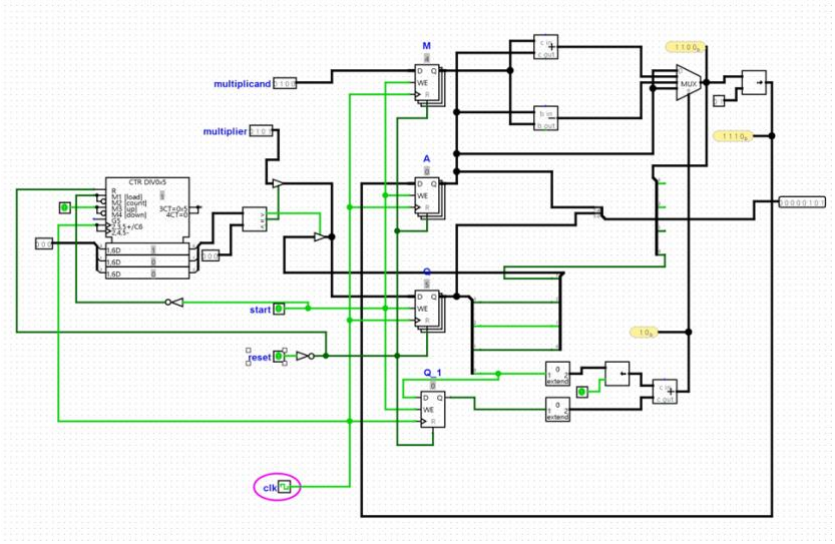
A. Step 0 : input 설정



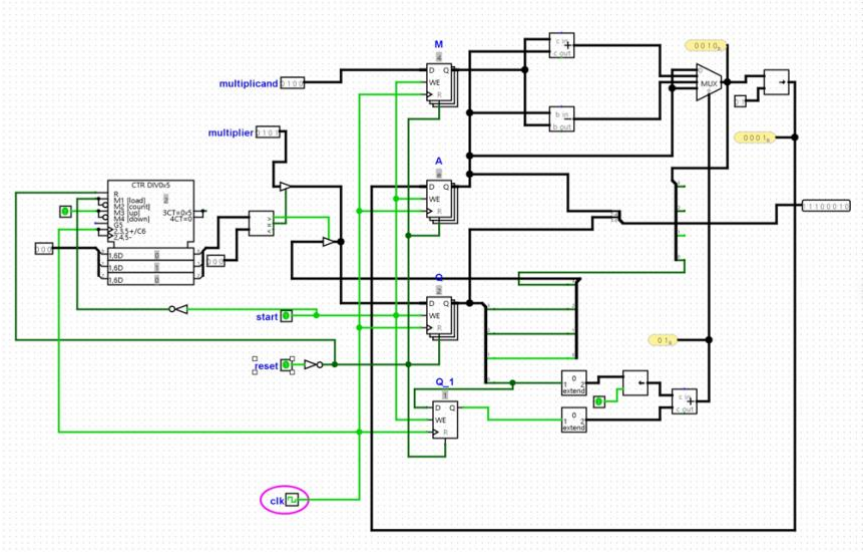
B. Step 1 : reset, start 1로 set



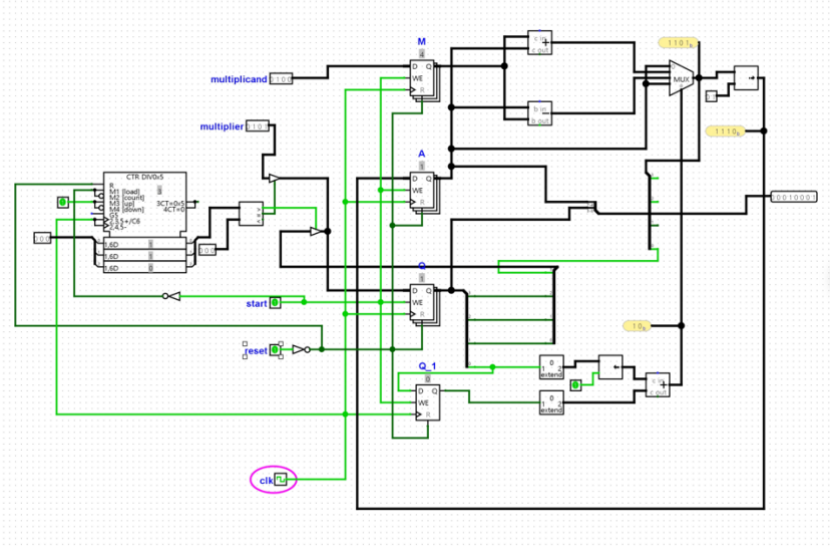
C. Step 2: Cycle 1, input 값이 register A와 Q를 거쳐 output으로 출력되었다.



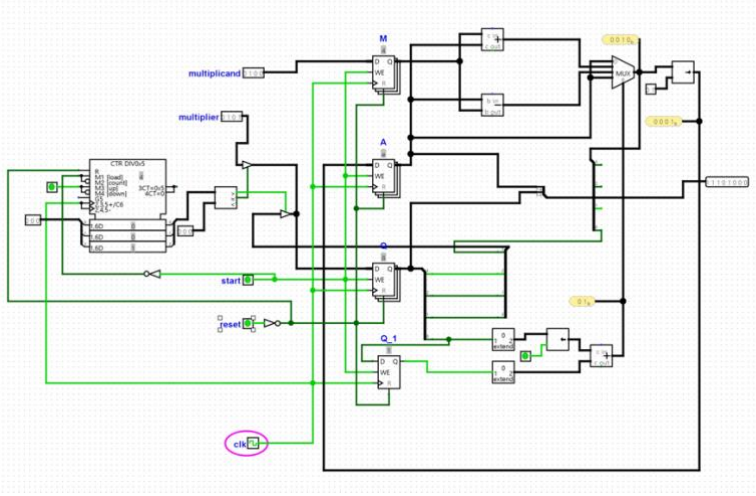
D. Step 3: Cycle 2



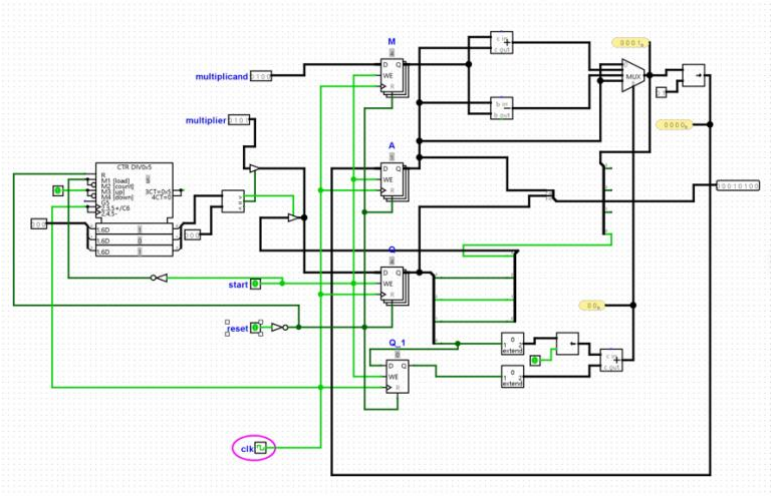
E. Step 4: Cycle 3



F. Step 5: Cycle 4



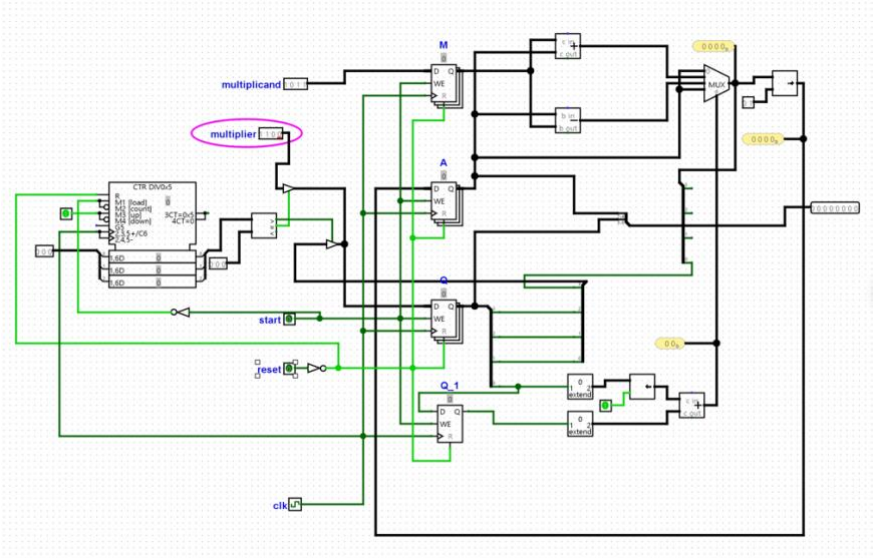
G. Step 5: Cycle 5



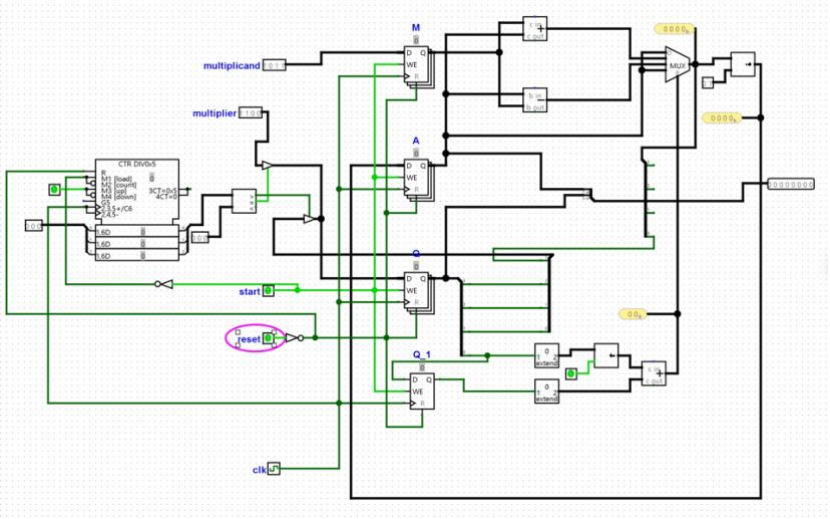
5 cycle 동안 multiplication이 완료되어 00010100인 20이 출력되었다.

2) -5 * -4

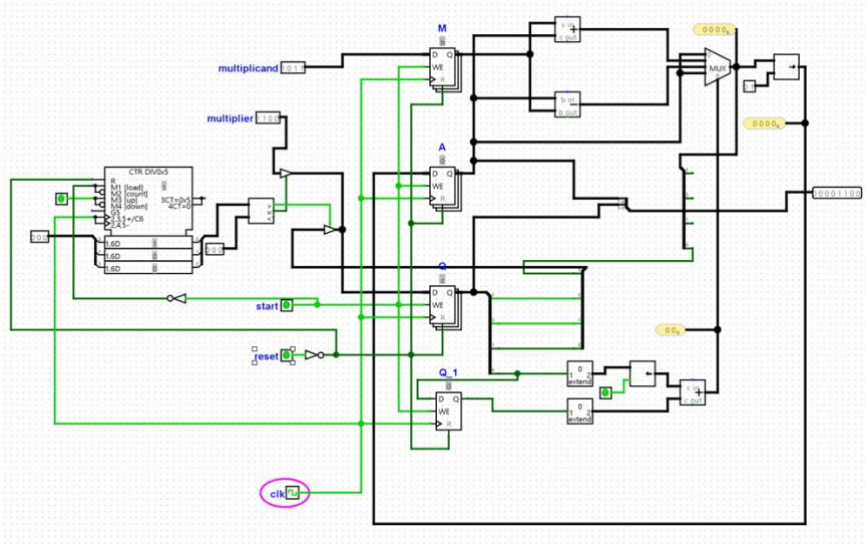
A. Step 0 : input 설정



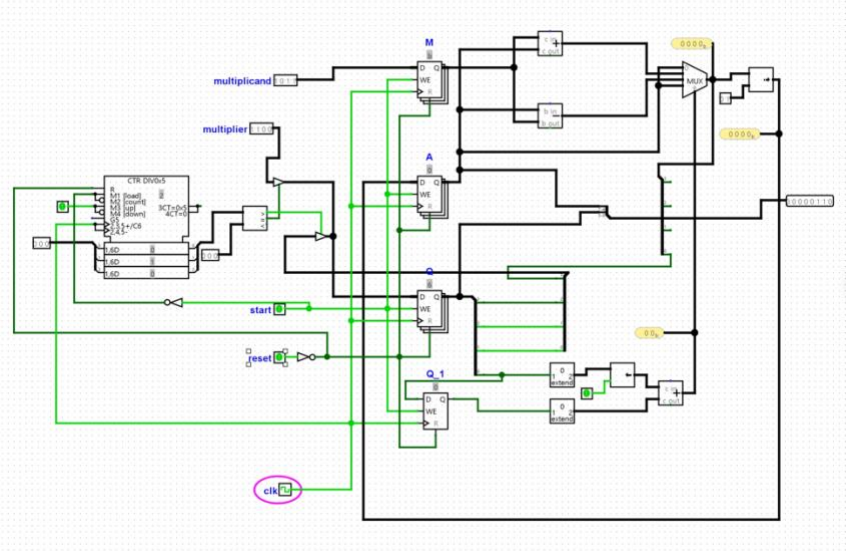
B. Step 1 : reset, start 1로 set



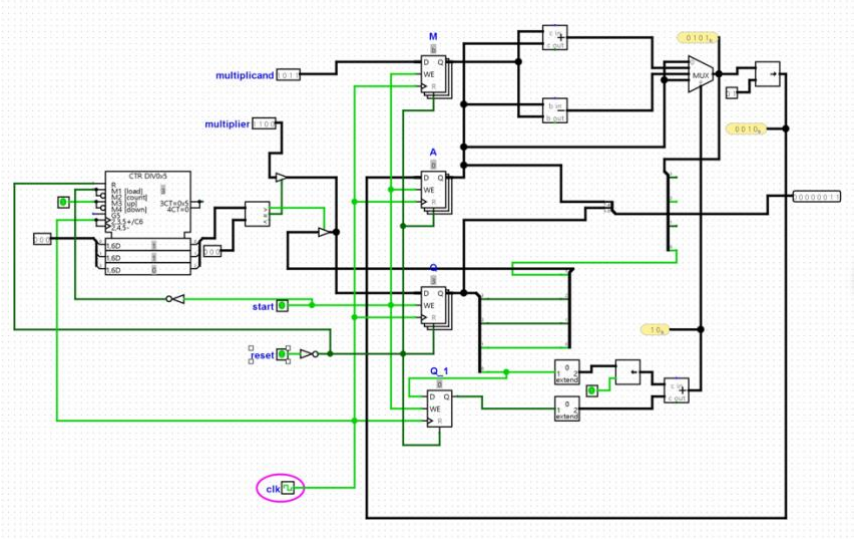
C. Step 2: Cycle 1, input 값이 register A와 Q를 거쳐 output으로 출력되었다.



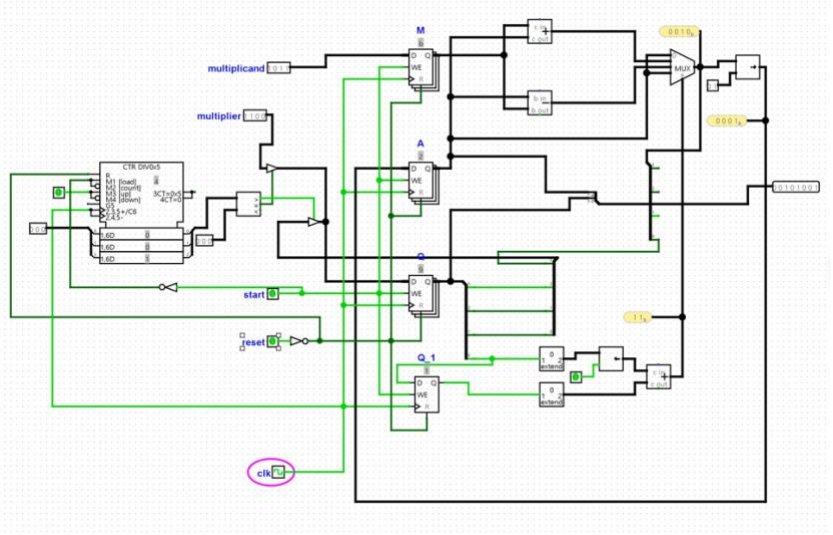
D. Step 3: Cycle 2



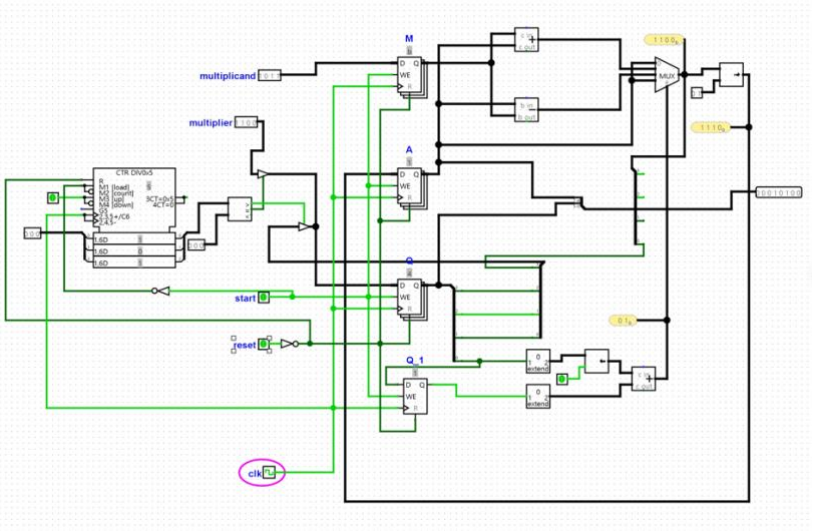
E. Step 4: Cycle 3



F. Step 5: Cycle 4



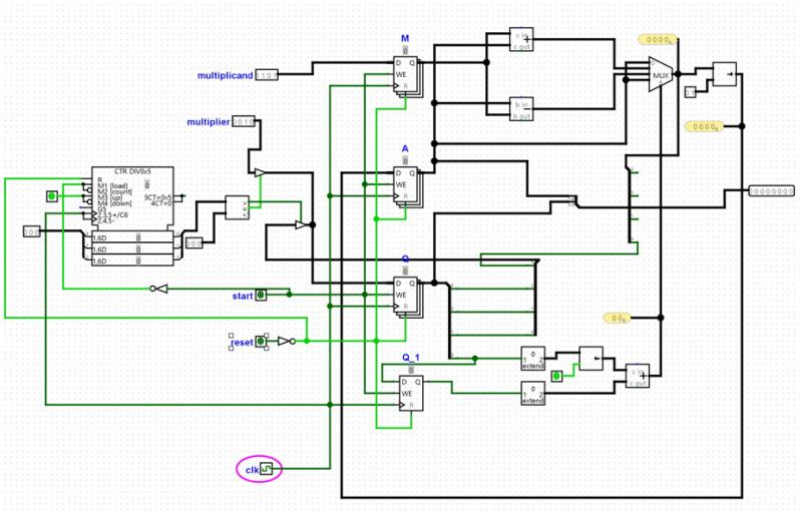
G. Step 6: Cycle 5



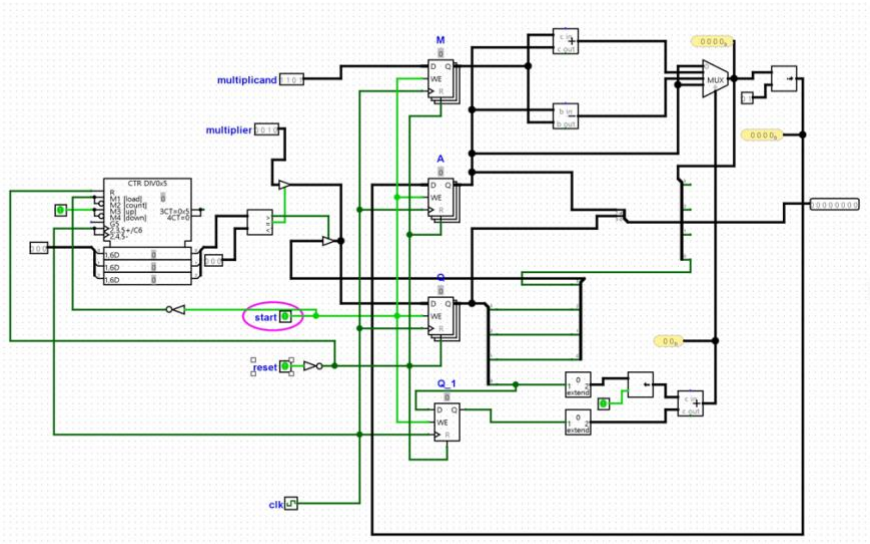
5 cycle 동안 multiplication이 완료되어 00010100인 20이 출력되었다.

3) -3 * 2

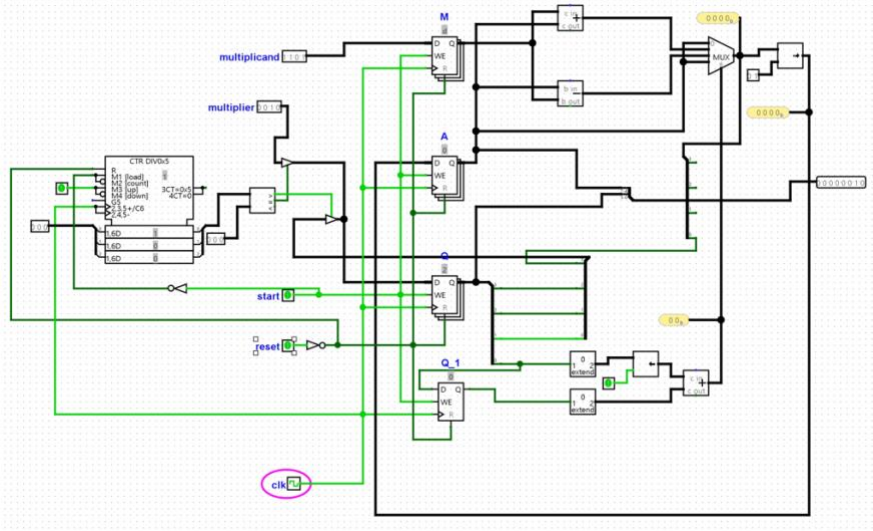
A. Step 0 : input 설정



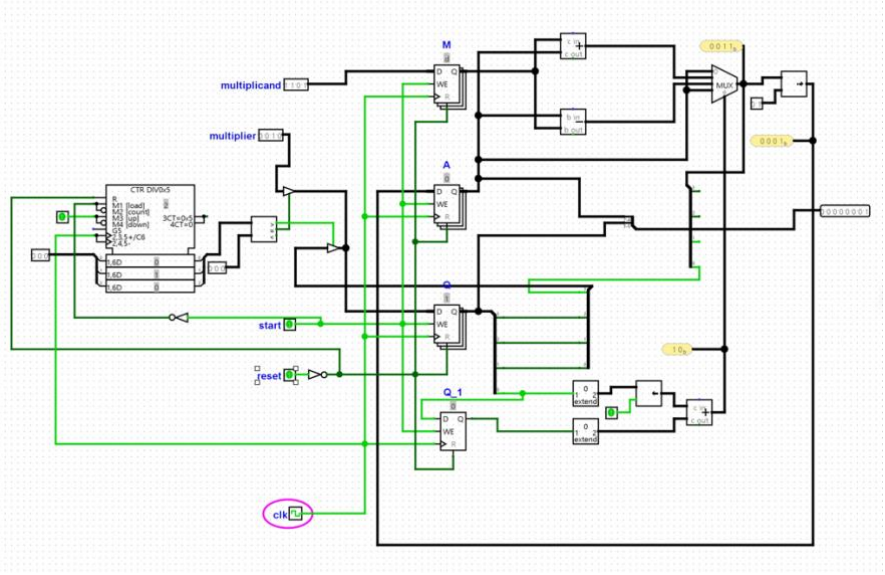
B. Step 1 : reset, start 1로 set



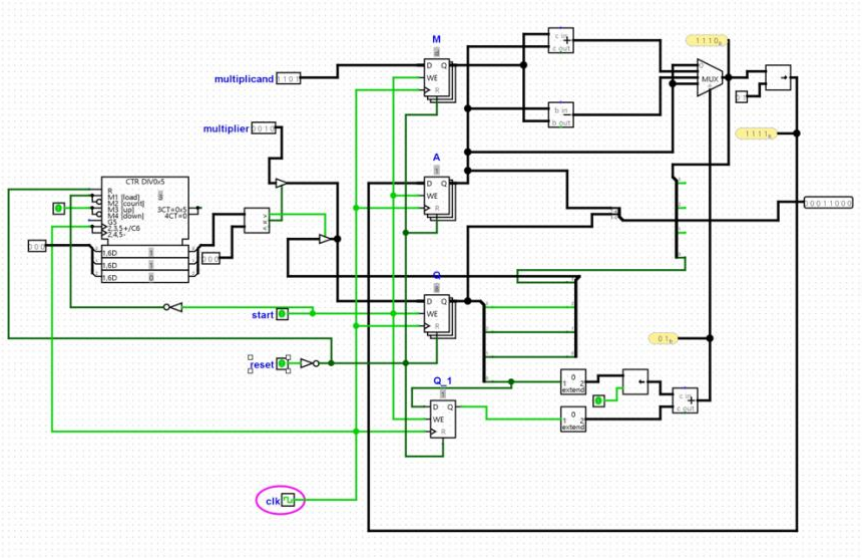
C. Step 2: Cycle 1, input 값이 register A와 Q를 거쳐 output으로 출력되었다.



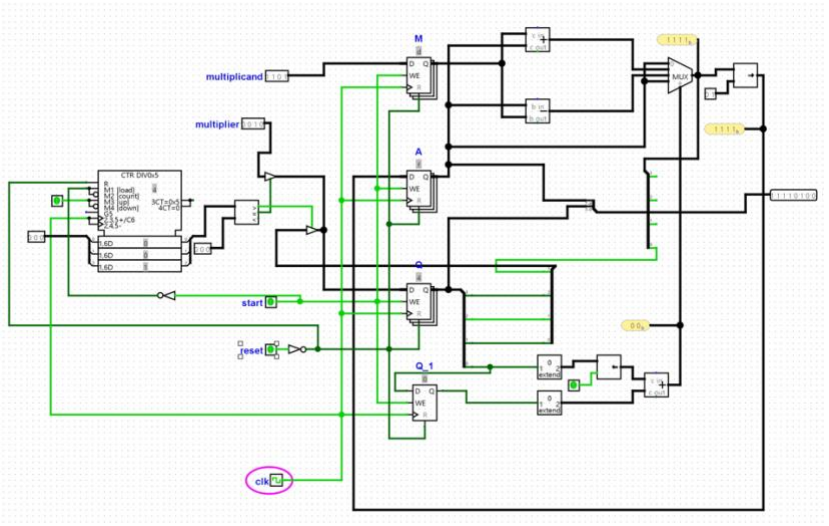
D. Step 3: Cycle 2



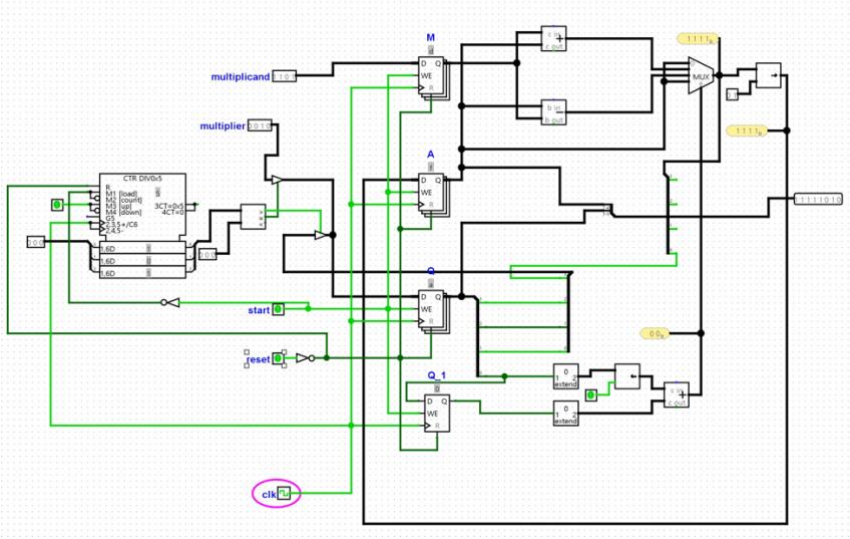
E. Step 4: Cycle 3



F. Step 5: Cycle 4



G. Step 6: Cycle 5



5 cycle 동안 multiplication이 완료되어 11111010인 -6이 출력되었다.

3. 고찰

Verilog로 Booth Algorithm에 대한 Multiplication을 코드로 구현하였으며 이를 gtkwave로 testbench를 작동시켜 4*5, -5*-4, -3*2의 파형의 변화를 확인하였다. 각 cycle 마다 A, Q, Q_1 그리고 result의 결과를 확인하였으며 Booth Algorithm에 의해 각 과정이 진행됨 또한 확인하였다.

이 Verilog를 바탕으로 Logisim evolution을 통해 회로를 구성하였다. 초기 multiplier에 대한 값을 Q에 넣고 그 이후에는 이후의 Q값을 다시 넣어야 하는데 이를 두기 위해서 Counter 값에 따라 입력되는 값이 달라지도록 설정하였다. 이 과정을 이해하기 위해 Logisim evolution에 있는 counter에 넣어야 할 각 input 값과 출력되는 output 값의 이해가 필요했다. 또한 Q[0]와 Q_1에 따라 달라지는 Booth Algorithm에 따라 바뀌도록 Mux를 활용하여 완성하였다.

4. Reference

1) 컴퓨터구조실험 Booth 제안서/ 컴퓨터구조실험 / 광운대학교/ 이성원 교수님/2023