컴퓨터 구조 & 실험

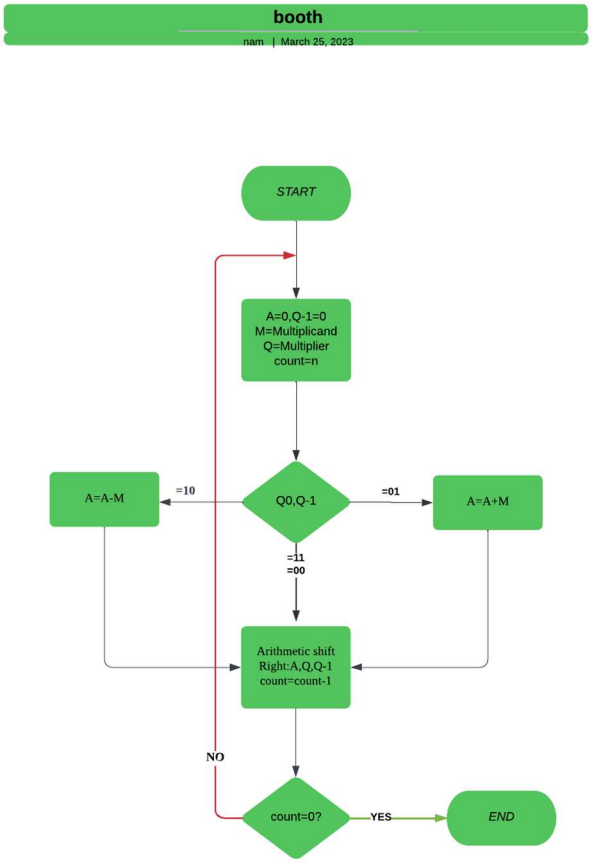
Verilog2 과제

수업 명 : 컴퓨터구조실험
과제 이름 : Verilog2
담당 교수님 : 이성원 교수님
학 번 : 2019202005
이 름 : 남종식

과제 소개

이번 과제는 두 수의 곱셈을 진행하는 booth algorithm을 구현하고 gtkwave를 통해 곱셈결과와 파형을 비교하는 과제입니다.

다음은 booth algorithm의 flow chart입니다.



Booth algorithm을 사용하여 4비트의 두 수의 곱셈을 진행합니다. 이때 Q의 LSB에 한 비트를 덧붙여 이 비트와 Q[0]비트를 비교하여 연산을 진행합니다. 이 덧붙인 비트는 Q-1이라고 하겠습니다. Q[0]Q-1 이 두 비트가 각각 00,11일때 01일 때 10일때를 살펴보겠습니다.

1) Q[0]Q-1=00,11

이때는 arithmetic shift right를 진행해줍니다. arithmetic shift right란 주어진 값을 오른쪽으로 주어진 비트 수만큼 이동시키는 연산입니다. 이때, 빈 자리는 MSB와 같은 값으로 채워집니다. 예를 들어, 2진수로 0000 1111를 arithmetic shift right를

하면 0000 0111이 됩니다. 빈 자리에는 MSB인 0이 채워집니다.

2) Q[0]Q-1=10

이때는 A-M을 진행하고 이 값을 A에 넣습니다. 이때 2의 보수를 이용하여 뺄셈을 진행하며 이후 arithmetic shift right를 진행해줍니다.

3) Q[0]Q-1=01

A+M을 진행 후 결과 값을 A에 넣어준 후 arithmetic shift right를 진행합니다.

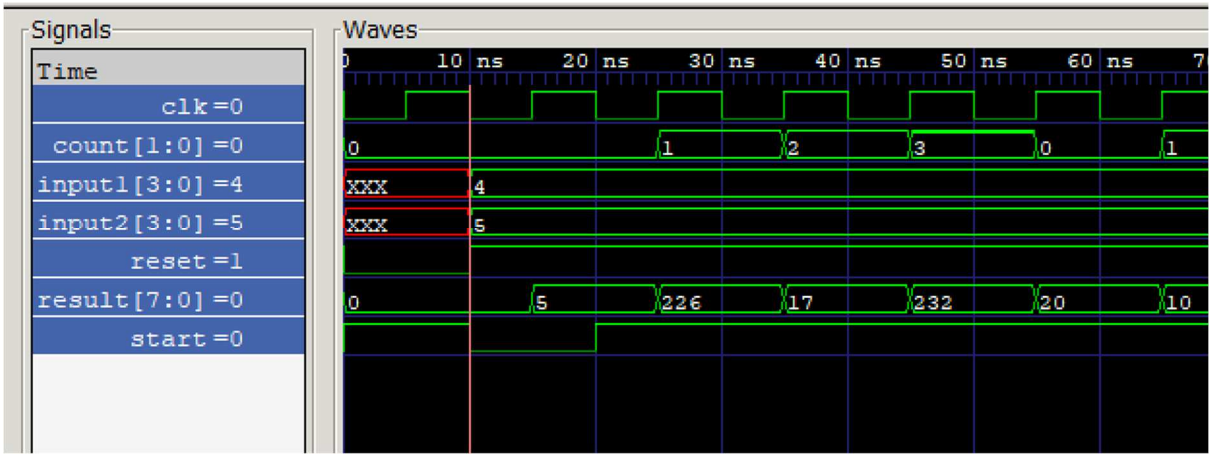
```
always @(posedge clk or negedge reset)
begin
    if(~reset)
    begin
        A<=4'b0;
        M<=4'b0;
        Q<=4'b0;
        Q_1<=4'b0;
        count <=2'b0;
    end

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

```
    else 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

alu adder(A,M,1'b0,add);
alu subtracter(A,~M,1'b1,sub);

assign result ={A,Q};
```

booth.v

```
        reset=1;
        start=1;
        clk=0;
        reset=0;
        #10;

        start=0;
        reset=1;
        input1=4;
        input2=5;
        #10;
        start=1;
        #640; $finish;
    end
```

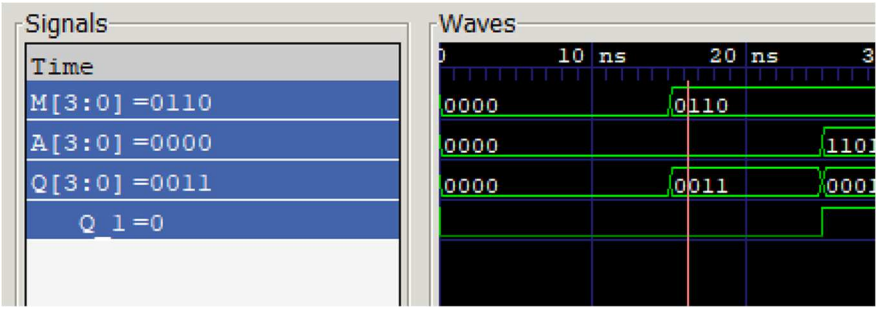
tb_booth.v



일단 코드와 gtkwave를 비교해보겠습니다. 먼저 always문 안에 동작은 posedge clk일 때 또는 negedge reset일 때 동작합니다. gtkwave를 확인해보면 posedge clk이고 reset이 0일 때 초기화가 진행되는 모습을 확인할 수 있습니다. 그 후 reset이 1이 되고 start가 0이 되는 순간 input값이 입력됩니다. 그리고 다음 posedge clk일 때 계산이 시작됩니다. 그 후 총 4cycle이후 4*5의 결과 값인 20이 나온 것을 확인할 수 있습니다. 4bit 두 수의 계산을 진행했으니 총 4cycle 이후 결과 값이 나왔습니다. 자세한 계산 과정은 다음 예시에서 확인해보겠습니다.

다음은 강의자료에 있던 input값이 아닌 다른 input값으로 곱의 결과를 확인해 보겠습니다. 예를 들어 M=6 Q=3일 때 M*Q의 값이 18이 나오는 지 확인해보겠습니다.

M	A	Q	Q-1
0110	0000	0011	0

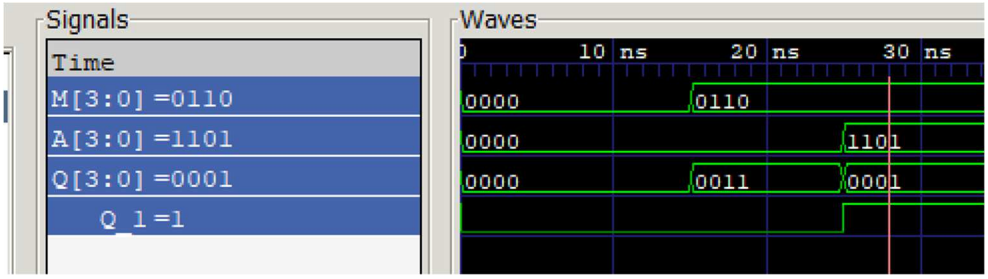


먼저 M=0110, Q=0011 그리고 A와 Q-1은 0으로 초기화 시킵니다.
이제부터 Q[0]과 Q-1을 비교해줍니다. Q[0]는 1 Q-1은 0일 때 SUB를 진행해줍니다. gtkwave에서도 값이 잘 들어온 것을 확인할 수 있습니다.

M	A	Q	Q-1
0110	0000	0011	0
0110	1010	0011	0

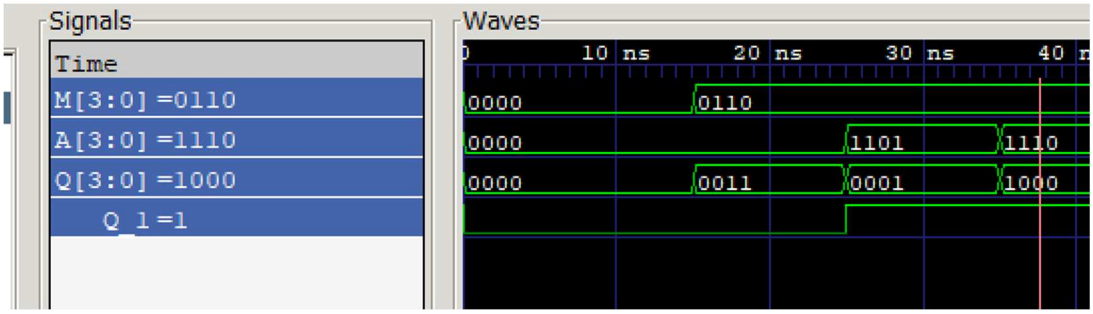
A=A-M을 진행해보면 0000-(0110)을 계산해야 합니다. 이때 0110을 2의 보수로 변환해 0000+1010을 진행합니다. 그 후 A=1010이 됩니다. 그 후 arithmetic shift right를 진행해줍니다.

M	A	Q	Q-1
0110	0000	0011	0
0110	1010	0011	0
0110	1101	0001	1



SUB와 arithmetic shift right를 진행 후 gtkwave입니다. 위 표와 값과 똑같이 진행되고 있는 모습을 확인할 수 있습니다.
Q[0]와 Q-1을 비교해보면 11이니까 arithmetic shift right를 진행합니다.

M	A	Q	Q-1
0110	0000	0011	0
0110	1010	0011	0
0110	1101	0001	1
0110	1110	1000	1



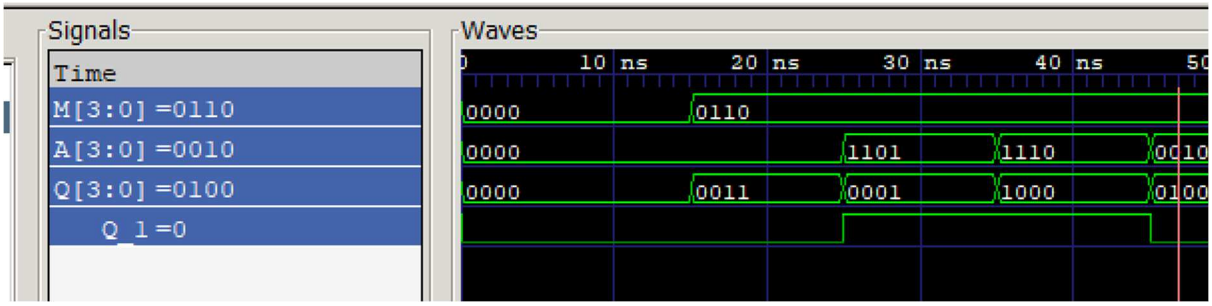
arithmetic shift right를 진행 후 표의 값과 gtkwave값이 같은 모습을 확인할 수 있습니다.
Q[0]와 Q-1을 비교해보면 01이니까 ADD를 진행합니다. 이때 A=1110 M=0110 두 수의 합은 0100이므로 A=0100이 됩니다.

M	A	Q	Q-1
0110	0000	0011	0

0110	1010	0011	0
0110	1101	0001	1
0110	1110	1000	1
0110	0100	1000	1

덧셈이 끝났으니 arithmetic shift right를 진행해줍니다.

M	A	Q	Q-1
0110	0000	0011	0
0110	1010	0011	0
0110	1101	0001	1
0110	1110	1000	1
0110	0100	1000	1
0110	0010	0100	0

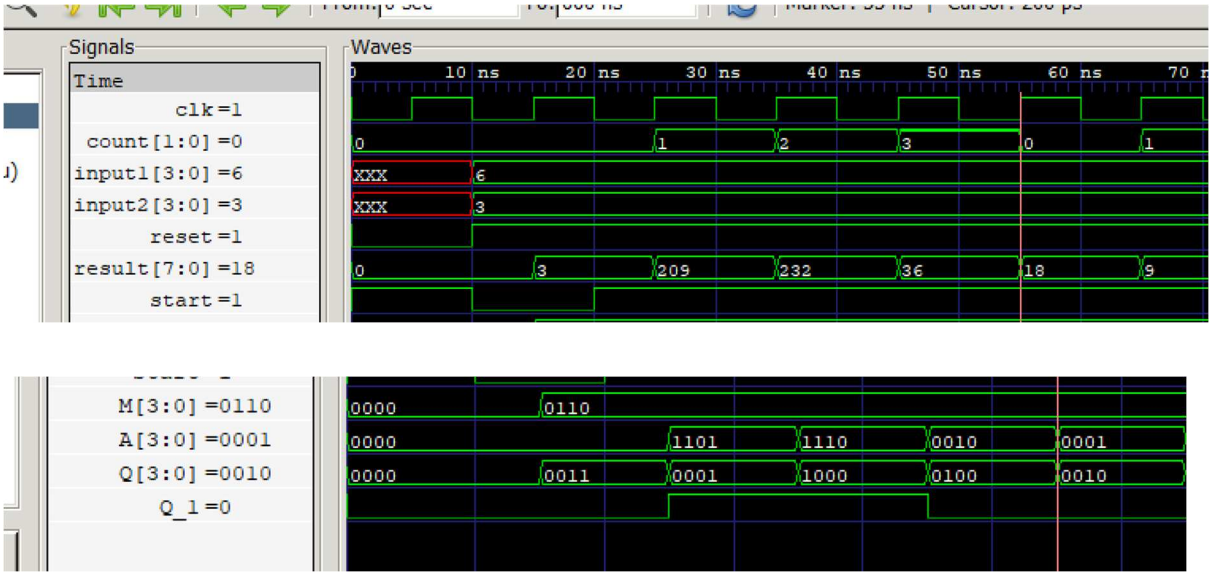


ADD와 arithmetic shift right 연산이 끝난 후 표의 값과 gtkwave의 값이 같은 모습을 확인할 수 있습니다.
Q[0]와 Q-1을 비교해보면 00이니까 arithmetic shift right를 진행해줍니다.

M	A	Q	Q-1
0110	0000	0011	0
0110	1010	0011	0
0110	1101	0001	1

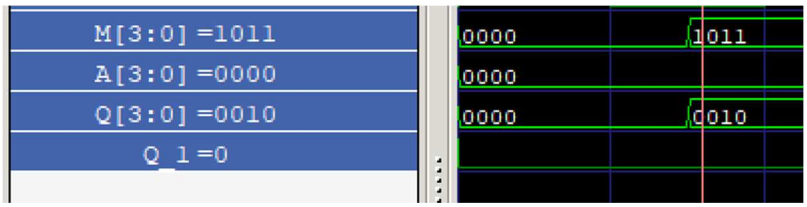
0110	1110	1000	1
0110	0100	1000	1
0110	0010	0100	0
0110	0001	0010	0

총 4번의 shift연산이 끝난 후 AQ의 연산결과가 8'b000100010이 나온 것을 확인할 수 있습니다. 이를 10진수로 변환하면 18이므로 처음 input값이였던 6과3의 곱의 결과가 나온 것을 확인할 수 있습니다.



총 연산이 끝난 후 표의 값과 gtkwave의 값이 같은 모습을 확인할 수 있습니다. count가 0이 되어 연산이 끝났다는 것을 알 수 있습니다.
다음으로 값이 결과가 음수일 때도 진행해보았습니다.
M=-5 Q=2일 때 M*Q의 값이 -10이 나오는 지 확인해보겠습니다.

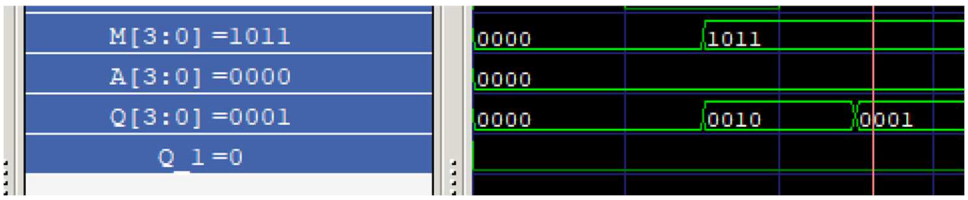
M	A	Q	Q-1
1011	0000	0010	0



표와 같이 input값을 설정해주었습니다.

Q[0]와 Q-1을 비교해보면 00이므로 arithmetic shift right를 진행합니다.

M	A	Q	Q-1
1011	0000	0010	0
1011	0000	0001	0



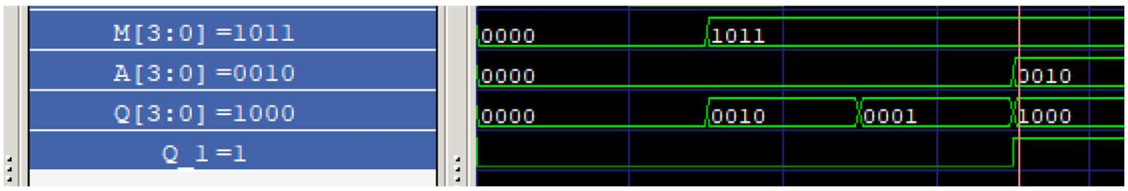
arithmetic shift right이후의 표의 값과 gtkwave로 확인한 값이 모두 일치하는 모습을 확인할 수 있습니다.

Q[0]와 Q-1을 비교해보면 10이므로 SUB를 진행합니다.

M	A	Q	Q-1
1011	0000	0010	0
1011	0000	0001	0
1011	0101	0001	0

SUB가 끝났으니 arithmetic shift right를 진행합니다.

M	A	Q	Q-1
1011	0000	0010	0
1011	0000	0001	0
1011	0101	0001	0
1011	0010	1000	1



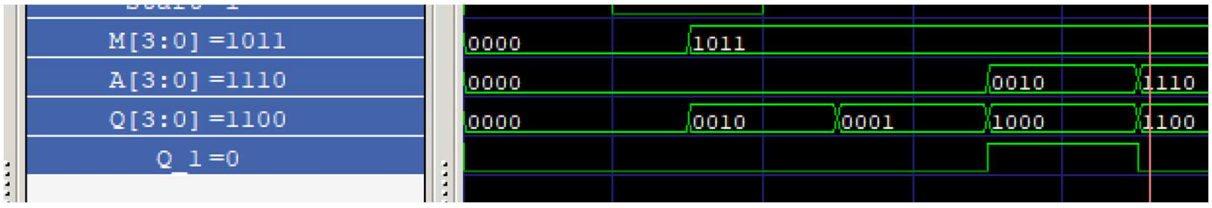
SUB와 arithmetic shift right연산이 끝난 후 값과 gtkwave에 나타나는 값입니다.

arithmetic shift right가 끝난 후 Q[0]와 Q-1을 비교해보면 01이므로 ADD를 진행합니다.

M	A	Q	Q-1
1011	0000	0010	0
1011	0000	0001	0
1011	0101	0001	0
1011	0010	1000	1
1011	1101	1000	1

ADD가 끝났으니 arithmetic shift right를 진행합니다.

M	A	Q	Q-1
1011	0000	0010	0
1011	0000	0001	0
1011	0110	0001	0
1011	0011	0000	1
1011	1101	1000	1
1011	1110	1100	0



ADD와 arithmetic shift right연산이 끝난 후 값과 gtkwave에 나타나는 값입니다

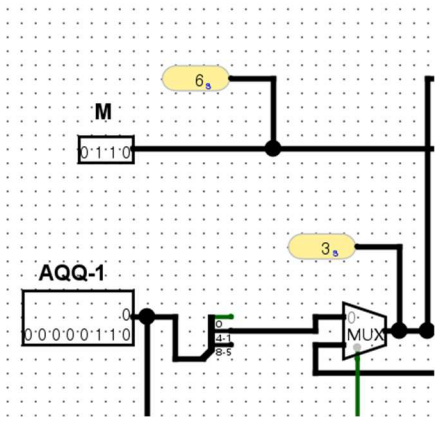
arithmetic shift right가 끝난 후 Q[0]와 Q-1을 비교해보면 00이므로 arithmetic shift right를 진행합니다.

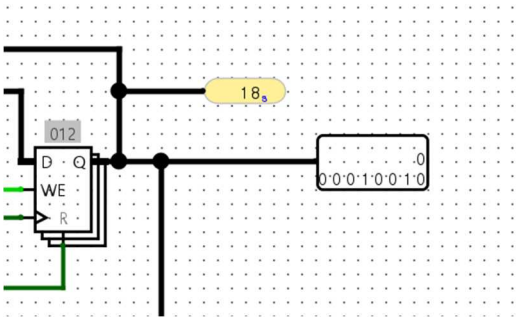
M	A	Q	Q-1
1011	0000	0010	0
1011	0000	0001	0
1011	0110	0001	0
1011	0011	0000	1
1011	1101	0000	1
1011	1110	1100	0
1011	1111	0110	0

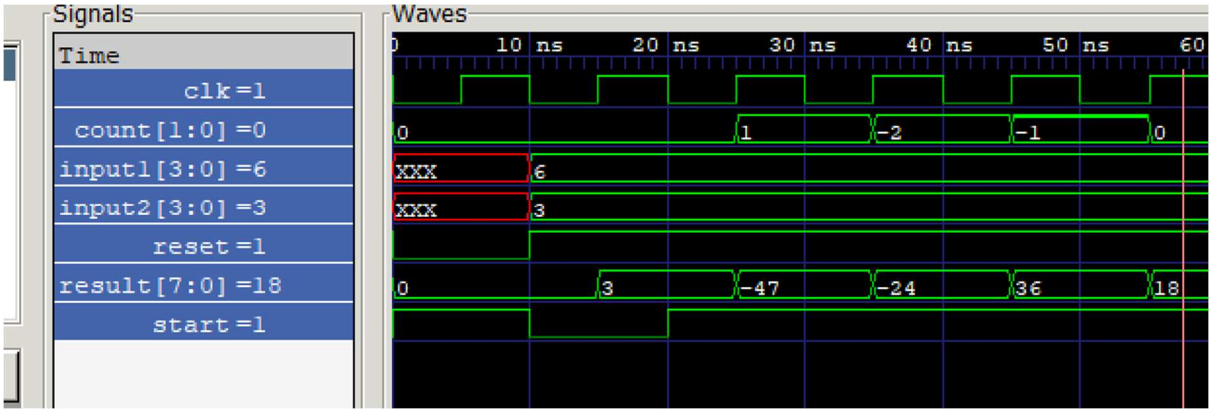
모든 연산이 끝난 후 결과 값과 gtkwave에 나타난 파형이 모두 일치하는 모습을 확인할 수 있습니다. 결과 값으로 AQ는 8'b11110110이 나왔습니다. 이는 10진수로 -10이므로 -5*2의 결과 값이 제대로 나온 것을 확인할 수 있습니다.

다음은 booth 알고리즘을 Logisim evolution으로 구현 관련 설명입니다. Logisim evolution에서는 booth algorithm을 구현할 때 AQQ-1부분을 한 비트로 모아서 연산을 진행했습니다. 먼저 A와 Q의 덧셈 및 뺄셈 연산을 진행할 때는 splitter를 이용해 9비트중 8~5비트를 뽑아 A를 얻어내고 4~1비트를 뽑아 Q를 얻어내어 계산했습니다. 이 연산을 진행하면 결과 값을 4비트로 나올 테니 여기서 또 한 번 splitter를 이용해 A부분에 4비트를 채우고 M부분 4비트 Q-1부분 1비트를 가져와 이번에는 splitter로 9비트로 만들어줬습니다. 이 연산을 하기 위해서는 Q[0]Q-1부분 비트를 판단해 연산 진행을 결정해야 하는데 이때는 4 to 1MUX를 이용해 결정했습니다. 두 비트가 00이면 arithmetic shift right를 진행하고 01이면 SUB, 10이면 ADD 그리고 11이면 arithmetic shift right를 진행하게 했습니다. 그리고 처음에 start가 0일 때 MUX를 이용해 레지스터 초기 input값을 AQQ-1값으로 넣어주었고 그 이후에는 shift를 진행한 값이 저장되게 해주었습니다. 초기 값을 레지스터에 저장하고 그 이후 start가1일때는 연산이 진행된 후의 값을 가지고 A와 Q와 Q-1의 값을 뽑아내어 진행했습니다.

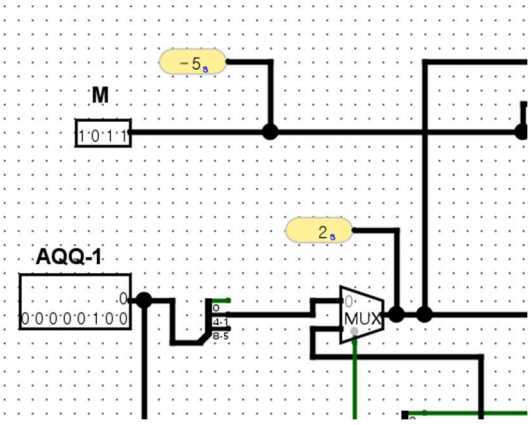
처음 진행했던 6*3 연산결과를 Logisim evolution으로 나타낸 결과입니다.



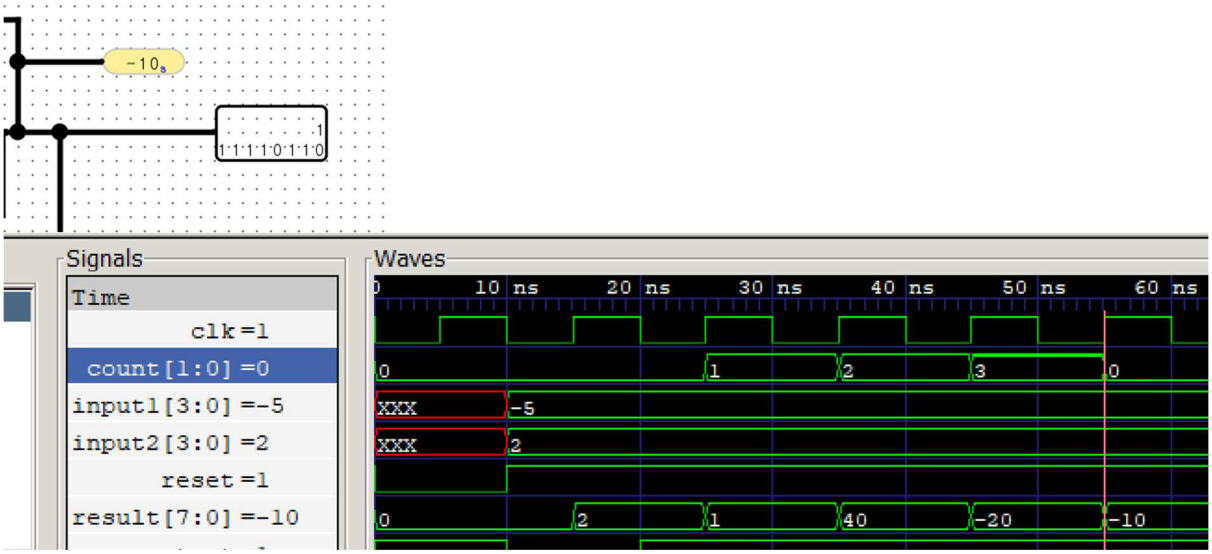




Gtkwave 확인을 통해 결과 값이 같게 나오는 것을 확인할 수 있습니다.



처음 초기 값 상태입니다. -5*2의 연산을 진행해 보았습니다. 처음에 start가 0일 때 초기 값을 레지스터에 저장하고 start를 1로 맞추고 clk을 4번 보내고 5번째 받는 시점에 결과 값이 나온 모습을 확인할 수 있습니다.



gtkwave에서 확인할 수 있듯이 1clk를 통해 input값이 들어오고 다음 clk부터 5번째 clk가 들어온 순간에 결과 값이 같아지는 모습을 확인할 수 있습니다.

고찰

예전에 booth algorithm에 대해서는 본 적이 있지만 자세하게 기억은 안나는 상태였습니다. 강의를 듣고 순서도와 표를 비교하면서 booth algorithm이 조금 생각이 나면서 이해할 수 있었습니다. 강의자료에 있는 예시 말고 과제에서는 제가 다른 예시를 들며 순서도에 따라 표를 작성하고 이 표를 gtkwave의 파형과 비교하면서 확인하니 이해가 더 쉬었고 arithmetic shift right연산 할 때마다 gtkwave의 값이 같아 신기하기도 했습니다.

Reference

Computer Architecture Lab 3주차 강의자료