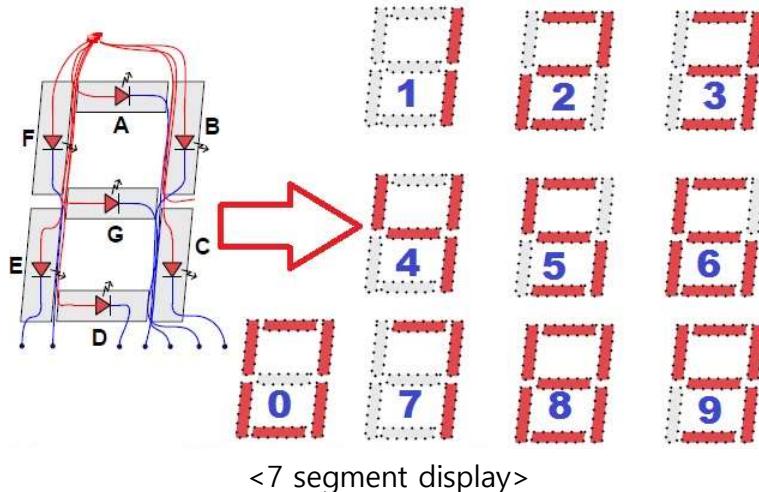


1. Simple ALU with 7 segment display

1) 7 segment display



s	BCD	G	F	E	D	C	B	A
0	0000	0	1	1	1	1	1	1
1	0001	0	0	0	0	1	1	0
2	0010	1	0	1	1	0	1	1
3	0011	1	0	0	1	1	1	1
4	0100	1	1	0	0	1	1	0
5	0101	1	1	0	1	1	0	1
6	0110	1	1	1	1	1	0	1
7	0111	0	0	0	0	1	1	1
8	1000	1	1	1	1	1	1	1
9	1001	1	1	0	1	1	1	1

<각 숫자에 따른 LED 진리표>

7 세그먼트 표시 장치(Seven-segment display)는 표시 장치의 일종으로, 7 개의 획으로 숫자나 문자를 나타낼 수 있다. 단순하기 때문에 전자 회로의 내부적인 수치를 보여 주는데 자주 사용된다. 7 세그먼트 표시 장치의 각 획은 맨 위쪽 가로 획부터 시계 방향으로, 그리고 마지막 가운데 가로 획까지 각각 A부터 G 까지의 이름으로 불린다. 소수를 나타내기 위해서 숫자의 오른쪽 아래에 소수점(DP)이 붙는 경우도 있다.

2) Simple ALU

본 과제에서는 4 비트 Overflow, Zero, Negative 의 경우를 감지하는 기능이 포함된 CLA 기반의 간단한 ALU 를 설계 및 사용한다.

2-0) Simulation using Mathematica

Overflow, Negative flag, Zero flag 의 연산 결과를 가시화하고 테스트하기 위해 다음 Wolfram Language 를 이용하여 CAS Tool 인 Mathematica 에서 다음과 같은 코드를 작성하여 사용하였다.

더하는 두 수를 X, Y 연산 결과를 S, 올림수를 C 라고 하고 LSB 부터 시작하여 각 자리를 표기하여 (예를 들어 X:1011 에서 X3=1, X2=0, X1=0, X0=0 이다)

$$S = X \oplus T \oplus C_{\infty}$$

$$C_{out} = XY + (X \oplus Y)C_{\infty}$$

<Sum 과 Carry 의 계산식>

```
In[1]:= C1 = (X0 && Y0) || (X0 \[And] Y0) && C0
Out[1]= (X0 && Y0) || ((X0 \[And] Y0) && C0)

In[2]:= C2 = (X1 && Y1) || (X1 \[And] Y1) && C1
Out[2]= (X1 && Y1) || ((X1 \[And] Y1) && ((X0 && Y0) || ((X0 \[And] Y0) && C0)))

In[3]:= C3 = (X2 && Y2) || (X2 \[And] Y2) && C2
Out[3]= (X2 && Y2) || ((X2 \[And] Y2) && ((X1 && Y1) || ((X1 \[And] Y1) && ((X0 && Y0) || ((X0 \[And] Y0) && C0)))))

In[4]:= C4 = (X3 && Y3) || (X3 \[And] Y3) && C3
Out[4]= (X3 && Y3) || ((X3 \[And] Y3) && ((X2 && Y2) || ((X2 \[And] Y2) && ((X1 && Y1) || ((X1 \[And] Y1) && ((X0 && Y0) || ((X0 \[And] Y0) && C0))))))

In[5]:= S0 = X0 \[And] Y0 \[And] C0
Out[5]= C0 \[And] X0 \[And] Y0

In[6]:= S1 = X1 \[And] Y1 \[And] C1
Out[6]= X1 \[And] Y1 \[And] ((X0 && Y0) || ((X0 \[And] Y0) && C0))

In[7]:= S2 = X2 \[And] Y2 \[And] C2
Out[7]= X2 \[And] Y2 \[And] ((X1 && Y1) || ((X1 \[And] Y1) && ((X0 && Y0) || ((X0 \[And] Y0) && C0)))))

In[8]:= S3 = X3 \[And] Y3 \[And] C3
Out[8]= X3 \[And] Y3 \[And] ((X2 && Y2) || ((X2 \[And] Y2) && ((X1 && Y1) || ((X1 \[And] Y1) && ((X0 && Y0) || ((X0 \[And] Y0) && C0))))))
```

<Sum 과 Carry 의 계산식을 이용하여 C1~C4, S0~S3 을 나타내는 코드>

```
In[10]:= Overflow1 = (! X3 && ! Y3 && S3) || (X3 && Y3 && ! S3)
Out[10]= (! X3 && ! Y3 && (X3 \[And] Y3 \[And] ((X2 \[And] Y2) || ((X2 \[And] Y2) && ((X1 && Y1) || ((X1 \[And] Y1) && ((X0 && Y0) || ((X0 \[And] Y0) && C0))))))) || (! X3 && Y3 && ! (X3 \[And] Y3 \[And] ((X2 \[And] Y2) || ((X2 \[And] Y2) && ((X1 && Y1) || ((X1 \[And] Y1) && ((X0 && Y0) || ((X0 \[And] Y0) && C0)))))))
```

```
In[11]:= Overflow2 = C4 \[And] C3
Out[11]= ((X2 && Y2) || ((X2 \[And] Y2) && ((X1 && Y1) || ((X1 \[And] Y1) && ((X0 && Y0) || ((X0 \[And] Y0) && C0)))))) \[And]
((X3 && Y3) || ((X3 \[And] Y3) && ((X2 \[And] Y2) || ((X2 \[And] Y2) && ((X1 && Y1) || ((X1 \[And] Y1) && ((X0 && Y0) || ((X0 \[And] Y0) && C0)))))))
```

```
In[12]:= BooleanMinimize[Overflow1] == BooleanMinimize[Overflow2]
[최소화] 부울 [최소화] 부울
Out[12]= True
```

```
In[13]:= Zero = ! S3 && ! S2 && ! S1 && ! S0
Out[13]= ! (X3 \[And] Y3 \[And] ((X2 && Y2) || ((X2 \[And] Y2) && ((X1 && Y1) || ((X1 \[And] Y1) && ((X0 && Y0) || ((X0 \[And] Y0) && C0))))))) \[And]
! (X2 \[And] Y2 \[And] ((X1 && Y1) || ((X1 \[And] Y1) && ((X0 && Y0) || ((X0 \[And] Y0) && C0)))) \[And] ! (X1 \[And] Y1 \[And] ((X0 && Y0) || ((X0 \[And] Y0) && C0)))) \[And] ! (C0 \[And] X0 \[And] Y0)
```

```
In[14]:= NegativeFlag = S3
Out[14]= X3 \[And] Y3 \[And] ((X2 && Y2) || ((X2 \[And] Y2) && ((X1 && Y1) || ((X1 \[And] Y1) && ((X0 && Y0) || ((X0 \[And] Y0) && C0))))))
```

<Overflow, Zero Flag, Negative Flag 의 정의, 설명 후술>

```

In[15]:= Convert2Param[X_, Y_, cc_] := {X3 -> StringTake[X, {1, 1}] == "1", X2 -> StringTake[X, {2, 2}] == "1", X1 -> StringTake[X, {3, 3}] == "1", X0 -> StringTake[X, {4, 4}] == "1",
                                         [문자열의 부분 추출] [문자열의 부분 추출] [문자열의 부분 추출] [문자열의 부분 추출]
                                         Y3 -> StringTake[Y, {1, 1}] == "1", Y2 -> StringTake[Y, {2, 2}] == "1", Y1 -> StringTake[Y, {3, 3}] == "1", Y0 -> StringTake[Y, {4, 4}] == "1", cc == "1"
                                         [문자열의 부분 추출] [문자열의 부분 추출] [문자열의 부분 추출] [문자열의 부분 추출]
                                         {"0000", "0001", "0010", "0100", "0110", "0111", "1000", "1001", "1010", "1011", "1100", "1101", "1110", "1111"}

In[16]:= BinaryToDecTable = {0, 1, 2, 3, 4, 5, 6, 7, -8, -7, -6, -5, -4, -3, -2, -1}
Out[16]= {0, 1, 2, 3, 4, 5, 6, 7, -8, -7, -6, -5, -4, -3, -2, -1}

In[17]:= ToBinaryString[i_] := StringPadLeft[IntegerString[i, 2], 4, "0"]
                                         [문자열 패딩] [문자열]

In[18]:= NormalizeBinaryDec[v_] := If[v > 7, v - 16, If[v < -8, v + 16, v]]
                                         [문자열]

In[19]:= GetExprValue[EXPR_, X_, Y_] := (EXPR /. Convert2Param[ToBinaryString[X], ToBinaryString[Y], "0"])

In[20]:= ToExprString[X_, Y_] := ToString[GetExprValue[X, Y]]
                                         [문자열]

In[21]:= ToExprFullString[X_, Y_] := ToBinaryString[X] <> "(" <> ToString[BinaryToDecTable[{X + 1}]] <> "+" <> Y <> ")"
                                         [문자열] [문자열] [문자열] [문자열] [문자열] [문자열]

In[22]:= SmartBlend[{list_}] := If[Length[{list}] == 0, White, If[Length[{list}] == 1, list[[1]], Blend[{list}]]]
                                         [문자열] [문자열] [문자열] [문자열] [문자열]

```

<연산 결과 테이블 렌더링용으로 작성한 코드>

2-1) Overflow Detection

두개의 4 비트 부호 정수를 연산할 때, Overflow 가 발생할 수 있는 경우를 따지면 크게 양수 + 양수, 음수 + 음수, 음수 + 양수, 양수 + 음수의 총 4 가지 케이스를 고려할 수 있다. 각 케이스의 대표적인 연산 결과를 나타내면 아래와 같다.

2-1-1) 양수 + 양수의 경우

정상적인 연산이 일어난 경우와 Overflow 가 일어나는 경우를 비교하면

Case: 양수 + 양수 (일반)

X: 0 1 1 0 (DEC:4)

Y: 0 0 0 1 (DEC:0)

$$\begin{array}{ccccccc} & \uparrow & \uparrow & \uparrow & \uparrow \\ C4 & C3 & C2 & C1 \\ =0 & :=0 & :=0 & :=0 \end{array}$$

S: 0 0 1 1 1 (DEC:4)

C4(=0) XOR C3(=0) = 0 <거짓>

Case: 양수 + 양수 (Overflow)

X: 0 1 0 0 (DEC:4)

Y: 0 1 0 0 (DEC:4)

$$\begin{array}{ccccccc} & \uparrow & \uparrow & \uparrow & \uparrow \\ C4 & C3 & C2 & C1 \\ =0 & :=1 & :=0 & :=0 \end{array}$$

S: 0 1 0 0 0 (DEC:-8)

C4(=0) XOR C3(=1) = 1 <참>

X_3, Y_3 이 0이고 S_3 이 1일 때 Overflow 가 일어남을 알 수 있다. 즉 X의 MSB 와 Y의 MSB 가 0이어서 X, Y가 양수인 상황에서 연산 결과인 S의 MSB 가 1이 된 경우, 즉 양수 + 양수 연산을 수행하였는데 음수가 나온 경우 Overflow 가 일어났다고 볼 수 있다.

Overflow1 = (! X3 && ! Y3 && S3) || (X3 && Y3 && ! S3)

이 경우의 조건식은 또 다른 표현으로 $C3=1, C4=0$ 인 경우로 표현할 수 있다. (X와 Y의 덧셈 결과가 음수가 나오기 위해서는 MSB 가 1이 되어야 하는데, 이를 위해서는 X_2, Y_2 간의 연산에서 올림수가 무조건 생겨야 0인 MSB 를 1로 만들 수 있다. 따라서 이 경우 $C3=1$ 이고, X_3, Y_3 간의 연산에서 C3 캐리를 고려하여도 C4 를 1로 만들 수 없기 때문에 $C4=0$ 이 된다.)

2-1-2) 음수 + 음수의 경우

정상적인 연산이 일어난 경우와 Overflow 가 일어나는 경우를 비교하면

Case: 음수 + 음수 (일반)

X: 1 1 1 1 (DEC:-1)

Y: 1 1 1 1 (DEC:-1)

↑ ↑ ↑ ↑

C4 C3 C2 C1

=1:=1:=1:=1

S: 1 1 1 1 0 (DEC:-2)

C4(=1) XOR C3(=1) = 0 <거짓>

Case: 음수 + 음수 (Overflow)

X: 1 0 0 0 (DEC:-8)

Y: 1 0 0 0 (DEC:-8)

↑ ↑ ↑ ↑

C4 C3 C2 C1

=1:=0:=0:=0

S: 1 0 0 0 0 (DEC:0)

C4(=1) XOR C3(=0) = 1 <참>

X_3, Y_3 이 1이고 S_3 이 0일 때 Overflow 가 일어남을 알 수 있다. 즉

X의 MSB 와 Y의 MSB 가 1이어서 X, Y가 음수인 상황에서 연산

결과인 S 의 MSB 가 0 이 된 경우, 즉 음수 + 음수 연산을 수행하였는데 음수가 나온 경우 Overflow 가 일어났다고 볼 수 있다.

$$\text{Overflow1} = (\text{! } X_3 \text{ && ! } Y_3 \text{ && } S_3) \text{ || } (X_3 \text{ && } Y_3 \text{ && ! } S_3)$$

이 경우의 조건식은 또 다른 표현으로 C3=0, C4=1 인 경우로 표현할 수 있다. (X 와 Y 의 MSB 가 1 이고, 이를 더하게 되면 항상 C4=1 이 된다, C4=1 이면서 S3=0(MSB=0, 음수간 덧셈 결과가 양수)인 경우가 본 케이스의 Overflow 조건임을 고려하면 이 경우 C3 은 1 이 될 수 없으므로 C3=0 이다.)

2-1-3) 양수 + 음수, 음수 + 양수의 경우

4 비트 부호 정수의 범위는 -8~7 이므로 가장 작은 음수 + (0 또는 양수)의 경우와, 가장 큰 양수 + (0 또는 음수)의 연산에서 가능한 가장 극단적인 경우는 -8+0, 7+0 이다. 이 두 연산 결과는 범위 안에서 존재하므로 이 경우에서는 Overflow 가 발생하지 않는다.

2-1-4) Overflow 검출식

$$\text{Overflow1} = (\text{! } X_3 \text{ && ! } Y_3 \text{ && } S_3) \text{ || } (X_3 \text{ && } Y_3 \text{ && ! } S_3)$$

$$\text{Overflow2} = C_4 \vee C_3$$

2-1-1~2-1-3 의 내용을 취합하여 X3, Y3, S3 을 이용한 검출식과 C4, C3 을 이용한 검출식을 만들 수 있다.

```
In[11]:= Overflow1 = (! X3 && ! Y3 && S3) || (X3 && Y3 && ! S3)
Out[11]= (! X3 && ! Y3 && (X3 \& Y3 \& ((X2 \& Y2) || ((X2 \& Y2) && ((X1 \& Y1) || ((X1 \& Y1) && ((X0 \& Y0) || ((X0 \& Y0) && C0))))))) || (X3 && Y3 && ! (X3 \& Y3 \& ((X2 \& Y2) || ((X2 \& Y2) && ((X1 \& Y1) || ((X1 \& Y1) && ((X0 \& Y0) || ((X0 \& Y0) && C0)))))))
```

```
In[11]:= Overflow2 = C4 \vee C3
Out[11]= ((X2 \& Y2) || ((X2 \& Y2) && ((X1 \& Y1) || ((X1 \& Y1) && ((X0 \& Y0) || ((X0 \& Y0) && C0))))) \vee (X3 \&& Y3) || ((X3 \& Y3) && ((X2 \& Y2) || ((X2 \& Y2) && ((X1 \& Y1) || ((X1 \& Y1) && ((X0 \& Y0) || ((X0 \& Y0) && C0))))))
```

```
In[12]:= BooleanMinimize[Overflow1] == BooleanMinimize[Overflow2]
Out[12]= True
```

두 검출식은 CAS Tool 로 비교하였을 때 동치이며, Overflow1 은 외부 출력으로 C3 을 고려하지 않고 설계한 가산기에서도 쓸 수 있다는 장점이, Overflow2 는 두 캐리(C3, C4) 값만을 이용하여 검출할 수 있다는 장점이 있는 것으로 판단된다.

```
In[1]:= Grid[Table[Item[ToExprToString[X, Y], {Background \rightarrow If[GetExprValue[Overflow1, X, Y], Red, White]}], {X, 0, 16 - 1}, {Y, 0, 16 - 1}], Frame \rightarrow All]
```

< Overflow1 인 경우를 빨간색으로 강조한 전체 연산 결과 표 >

위의 선제 연산 결과 표에서도, 음수+양수, 양수+음수 부분은 강조 표시 되지 않았고, 양수+양수, 음수+음수 경우에서 연산 결과가 Overflow 된 경우만 알맞게 강조 표시 되었음을 확인할 수 있다.

2-2) Negative flag

$S3=1$ 일 때, 즉 연산 결과의 MSB 가 1 일 때 결과 값이 음수이므로 $S3$ 을 Negative flag 로 고려할 수 있다.

In[14]:= NegativeFlag = S:

```
Out[14]= X3 \[And] Y3 \[And] ((X2 \[And] Y2) || ((X2 \[Or] Y2) \[And] ((X1 \[And] Y1) || ((X1 \[Or] Y1) \[And] ((X0 \[And] Y0) || ((X0 \[Or] Y0) \[And] C0)))))))
```

```
Module[{"x", "y"}, Grid[Table[Item[ToExprFullString[x, y], {Background -> If[GetExprValue[NegativeVerlag, x, y], Blue, White]}], {x, 0, 16 - 1}, {y, 0, 16 - 1}], Frame -> All]]
```

<NegativeFlag 인 경우를 파란색으로 강조한 전체 연산 결과 표>

위의 전체 연산 결과 표에서도, 연산 결과가 음수인 경우에서만 올바르게 강조 표시 되었음을 확인 할 수 있다.

2-3) Zero flag

S3, S2, S1, S0 이 모두 0 일 때 연산 값은 0 이다. 따라서 Zero flag 는 다음과 같이 표현할 수 있다.

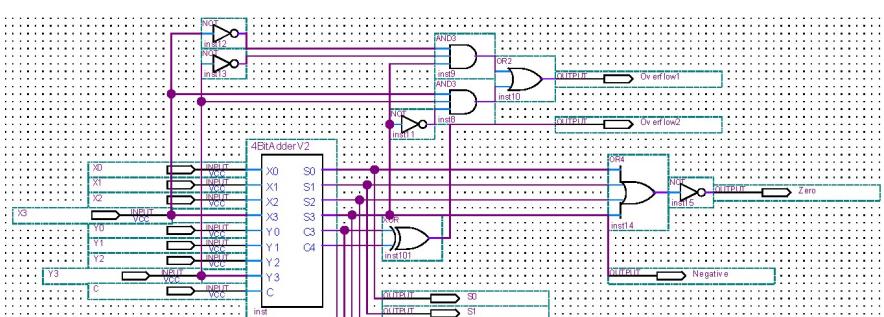
```
In[13]:= Zero = ! S3 && ! S2 && ! S1 && ! S0
Out[13]= !(X3 \[And] Y3 \[And] ((X2 \[And] Y2) \[And] ((X1 \[And] Y1) \[Or] ((X1 \[And] Y1) \[And] ((X0 \[And] Y0) \[Or] ((X0 \[And] Y0) \[And] C0))))) \[And]
           !(X2 \[And] Y2 \[And] ((X1 \[And] Y1) \[And] ((X0 \[And] Y0) \[Or] ((X0 \[And] Y0) \[And] C0)))) \[And] !(X1 \[And] Y1 \[And] ((X0 \[And] Y0) \[Or] ((X0 \[And] Y0) \[And] C0))) \[And] !(C0 \[And] X0 \[And] Y0)
```

(4 항 De Morgan's laws 에 의해 !(S3 || S2 || S1 || S0)로도 표현 가능)

인덱스	Grid	Table	TableItem	ToExprFullString	X, Y	Background	If[GetExprValue[Zero, X, Y], Green, White]	(X, 0, 16 - 1), (Y, 0, 16 - 1)	Frame	A111
[문자] [한글] [영문] [한국어] [영어] [한국어] [영어] [한국어] [영어] [한국어] [영어]										
[한글]										
0000 (0 - 0000 0 - 0001)	0000 (0 - 0000 0 - 0010)	0000 (0 - 0000 0 - 0011)	0000 (0 - 0000 0 - 0100)	0000 (0 - 0000 0 - 0101)	0000 (0 - 0000 0 - 0110)	0000 (0 - 0000 0 - 0111)	0000 (0 - 0000 0 - 1000)	0000 (0 - 0000 0 - 1001)	0000 (0 - 0000 0 - 1010)	0000 (0 - 0000 0 - 1011)
(0 - 0) - 1	(0 - 0) - 2	(0 - 0) - 3	(0 - 0) - 4	(0 - 0) - 5	(0 - 0) - 6	(0 - 0) - 7	(0 - 0) - 8	(0 - 0) - 9	(0 - 0) - 10	(0 - 0) - 11
0001 (1 - 0000 1 - 0001)	0001 (1 - 0000 1 - 0010)	0001 (1 - 0000 1 - 0011)	0001 (1 - 0000 1 - 0100)	0001 (1 - 0000 1 - 0101)	0001 (1 - 0000 1 - 0110)	0001 (1 - 0000 1 - 0111)	0001 (1 - 0000 1 - 1000)	0001 (1 - 0000 1 - 1001)	0001 (1 - 0000 1 - 1010)	0001 (1 - 0000 1 - 1011)
(0 - 1) - 1	(0 - 1) - 2	(0 - 1) - 3	(0 - 1) - 4	(0 - 1) - 5	(0 - 1) - 6	(0 - 1) - 7	(0 - 1) - 8	(0 - 1) - 9	(0 - 1) - 10	(0 - 1) - 11
0010 (2 - 0000 2 - 0001)	0010 (2 - 0000 2 - 0010)	0010 (2 - 0000 2 - 0011)	0010 (2 - 0000 2 - 0100)	0010 (2 - 0000 2 - 0101)	0010 (2 - 0000 2 - 0110)	0010 (2 - 0000 2 - 0111)	0010 (2 - 0000 2 - 1000)	0010 (2 - 0000 2 - 1001)	0010 (2 - 0000 2 - 1010)	0010 (2 - 0000 2 - 1011)
(0 - 2) - 1	(0 - 2) - 2	(0 - 2) - 3	(0 - 2) - 4	(0 - 2) - 5	(0 - 2) - 6	(0 - 2) - 7	(0 - 2) - 8	(0 - 2) - 9	(0 - 2) - 10	(0 - 2) - 11
0011 (3 - 0000 3 - 0001)	0011 (3 - 0000 3 - 0010)	0011 (3 - 0000 3 - 0011)	0011 (3 - 0000 3 - 0100)	0011 (3 - 0000 3 - 0101)	0011 (3 - 0000 3 - 0110)	0011 (3 - 0000 3 - 0111)	0011 (3 - 0000 3 - 1000)	0011 (3 - 0000 3 - 1001)	0011 (3 - 0000 3 - 1010)	0011 (3 - 0000 3 - 1011)
(0 - 3) - 1	(0 - 3) - 2	(0 - 3) - 3	(0 - 3) - 4	(0 - 3) - 5	(0 - 3) - 6	(0 - 3) - 7	(0 - 3) - 8	(0 - 3) - 9	(0 - 3) - 10	(0 - 3) - 11
0100 (4 - 0000 4 - 0001)	0100 (4 - 0000 4 - 0010)	0100 (4 - 0000 4 - 0011)	0100 (4 - 0000 4 - 0100)	0100 (4 - 0000 4 - 0101)	0100 (4 - 0000 4 - 0110)	0100 (4 - 0000 4 - 0111)	0100 (4 - 0000 4 - 1000)	0100 (4 - 0000 4 - 1001)	0100 (4 - 0000 4 - 1010)	0100 (4 - 0000 4 - 1011)
(0 - 4) - 1	(0 - 4) - 2	(0 - 4) - 3	(0 - 4) - 4	(0 - 4) - 5	(0 - 4) - 6	(0 - 4) - 7	(0 - 4) - 8	(0 - 4) - 9	(0 - 4) - 10	(0 - 4) - 11
0101 (5 - 0000 5 - 0001)	0101 (5 - 0000 5 - 0010)	0101 (5 - 0000 5 - 0011)	0101 (5 - 0000 5 - 0100)	0101 (5 - 0000 5 - 0101)	0101 (5 - 0000 5 - 0110)	0101 (5 - 0000 5 - 0111)	0101 (5 - 0000 5 - 1000)	0101 (5 - 0000 5 - 1001)	0101 (5 - 0000 5 - 1010)	0101 (5 - 0000 5 - 1011)
(0 - 5) - 1	(0 - 5) - 2	(0 - 5) - 3	(0 - 5) - 4	(0 - 5) - 5	(0 - 5) - 6	(0 - 5) - 7	(0 - 5) - 8	(0 - 5) - 9	(0 - 5) - 10	(0 - 5) - 11
0110 (6 - 0000 6 - 0001)	0110 (6 - 0000 6 - 0010)	0110 (6 - 0000 6 - 0011)	0110 (6 - 0000 6 - 0100)	0110 (6 - 0000 6 - 0101)	0110 (6 - 0000 6 - 0110)	0110 (6 - 0000 6 - 0111)	0110 (6 - 0000 6 - 1000)	0110 (6 - 0000 6 - 1001)	0110 (6 - 0000 6 - 1010)	0110 (6 - 0000 6 - 1011)
(0 - 6) - 1	(0 - 6) - 2	(0 - 6) - 3	(0 - 6) - 4	(0 - 6) - 5	(0 - 6) - 6	(0 - 6) - 7	(0 - 6) - 8	(0 - 6) - 9	(0 - 6) - 10	(0 - 6) - 11
0111 (7 - 0000 7 - 0001)	0111 (7 - 0000 7 - 0010)	0111 (7 - 0000 7 - 0011)	0111 (7 - 0000 7 - 0100)	0111 (7 - 0000 7 - 0101)	0111 (7 - 0000 7 - 0110)	0111 (7 - 0000 7 - 0111)	0111 (7 - 0000 7 - 1000)	0111 (7 - 0000 7 - 1001)	0111 (7 - 0000 7 - 1010)	0111 (7 - 0000 7 - 1011)
(0 - 7) - 1	(0 - 7) - 2	(0 - 7) - 3	(0 - 7) - 4	(0 - 7) - 5	(0 - 7) - 6	(0 - 7) - 7	(0 - 7) - 8	(0 - 7) - 9	(0 - 7) - 10	(0 - 7) - 11
1000 (8 - 0000 8 - 0001)	1000 (8 - 0000 8 - 0010)	1000 (8 - 0000 8 - 0011)	1000 (8 - 0000 8 - 0100)	1000 (8 - 0000 8 - 0101)	1000 (8 - 0000 8 - 0110)	1000 (8 - 0000 8 - 0111)	1000 (8 - 0000 8 - 1000)	1000 (8 - 0000 8 - 1001)	1000 (8 - 0000 8 - 1010)	1000 (8 - 0000 8 - 1011)
(0 - 8) - 1	(0 - 8) - 2	(0 - 8) - 3	(0 - 8) - 4	(0 - 8) - 5	(0 - 8) - 6	(0 - 8) - 7	(0 - 8) - 8	(0 - 8) - 9	(0 - 8) - 10	(0 - 8) - 11
1001 (9 - 0000 9 - 0001)	1001 (9 - 0000 9 - 0010)	1001 (9 - 0000 9 - 0011)	1001 (9 - 0000 9 - 0100)	1001 (9 - 0000 9 - 0101)	1001 (9 - 0000 9 - 0110)	1001 (9 - 0000 9 - 0111)	1001 (9 - 0000 9 - 1000)	1001 (9 - 0000 9 - 1001)	1001 (9 - 0000 9 - 1010)	1001 (9 - 0000 9 - 1011)
(0 - 9) - 1	(0 - 9) - 2	(0 - 9) - 3	(0 - 9) - 4	(0 - 9) - 5	(0 - 9) - 6	(0 - 9) - 7	(0 - 9) - 8	(0 - 9) - 9	(0 - 9) - 10	(0 - 9) - 11
1010 (10 - 0000 10 - 0001)	1010 (10 - 0000 10 - 0010)	1010 (10 - 0000 10 - 0011)	1010 (10 - 0000 10 - 0100)	1010 (10 - 0000 10 - 0101)	1010 (10 - 0000 10 - 0110)	1010 (10 - 0000 10 - 0111)	1010 (10 - 0000 10 - 1000)	1010 (10 - 0000 10 - 1001)	1010 (10 - 0000 10 - 1010)	1010 (10 - 0000 10 - 1011)
(0 - 10) - 1	(0 - 10) - 2	(0 - 10) - 3	(0 - 10) - 4	(0 - 10) - 5	(0 - 10) - 6	(0 - 10) - 7	(0 - 10) - 8	(0 - 10) - 9	(0 - 10) - 10	(0 - 10) - 11
1011 (11 - 0000 11 - 0001)	1011 (11 - 0000 11 - 0010)	1011 (11 - 0000 11 - 0011)	1011 (11 - 0000 11 - 0100)	1011 (11 - 0000 11 - 0101)	1011 (11 - 0000 11 - 0110)	1011 (11 - 0000 11 - 0111)	1011 (11 - 0000 11 - 1000)	1011 (11 - 0000 11 - 1001)	1011 (11 - 0000 11 - 1010)	1011 (11 - 0000 11 - 1011)
(0 - 11) - 1	(0 - 11) - 2	(0 - 11) - 3	(0 - 11) - 4	(0 - 11) - 5	(0 - 11) - 6	(0 - 11) - 7	(0 - 11) - 8	(0 - 11) - 9	(0 - 11) - 10	(0 - 11) - 11
1100 (12 - 0000 12 - 0001)	1100 (12 - 0000 12 - 0010)	1100 (12 - 0000 12 - 0011)	1100 (12 - 0000 12 - 0100)	1100 (12 - 0000 12 - 0101)	1100 (12 - 0000 12 - 0110)	1100 (12 - 0000 12 - 0111)	1100 (12 - 0000 12 - 1000)	1100 (12 - 0000 12 - 1001)	1100 (12 - 0000 12 - 1010)	1100 (12 - 0000 12 - 1011)
(0 - 12) - 1	(0 - 12) - 2	(0 - 12) - 3	(0 - 12) - 4	(0 - 12) - 5	(0 - 12) - 6	(0 - 12) - 7	(0 - 12) - 8	(0 - 12) - 9	(0 - 12) - 10	(0 - 12) - 11
1111 (-1 - 0000 13 - 0001)	1111 (-1 - 0000 13 - 0010)	1111 (-1 - 0000 13 - 0011)	1111 (-1 - 0000 13 - 0100)	1111 (-1 - 0000 13 - 0101)	1111 (-1 - 0000 13 - 0110)	1111 (-1 - 0000 13 - 0111)	1111 (-1 - 0000 13 - 1000)	1111 (-1 - 0000 13 - 1001)	1111 (-1 - 0000 13 - 1010)	1111 (-1 - 0000 13 - 1011)
(0 - 13) - 1	(0 - 13) - 2	(0 - 13) - 3	(0 - 13) - 4	(0 - 13) - 5	(0 - 13) - 6	(0 - 13) - 7	(0 - 13) - 8	(0 - 13) - 9	(0 - 13) - 10	(0 - 13) - 11

<Zero 인 경우를 파란색으로 강조한 전체 연산 결과 표>

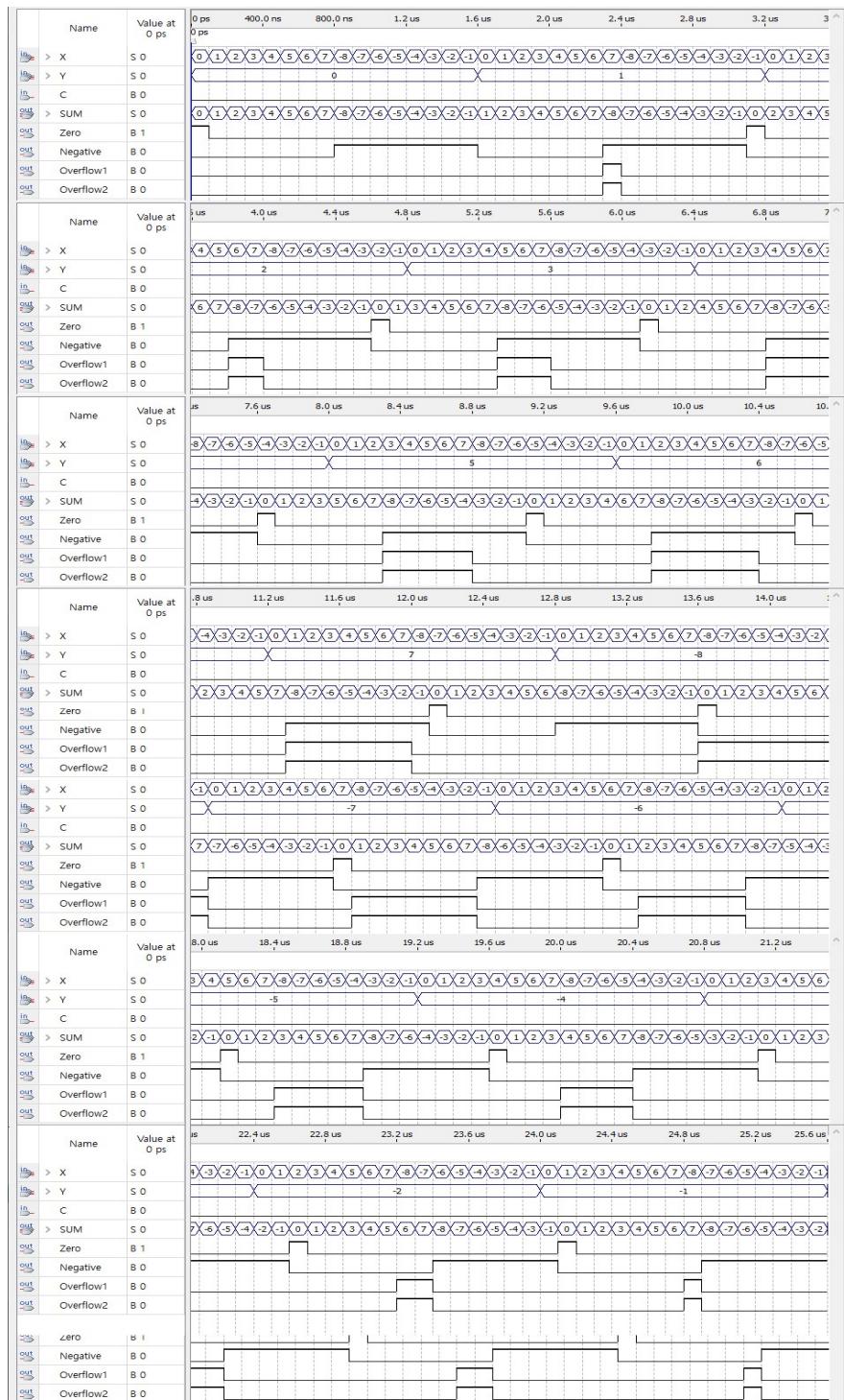
2-4) CAD with Simulation



<4BitRCA with Flag Circuit>

Overflow1 과 Overflow2 의 값을 모두 확인하기 위해 기존 조합논리에 대한 학습(1) 과제에서 사용하였던 4BitAdder 를 수정하여 C3 을 외부 출력으로 사용하는 4BitAdderV2 를 만들어 회로에 사용하였다.

2-5) Function Simulation

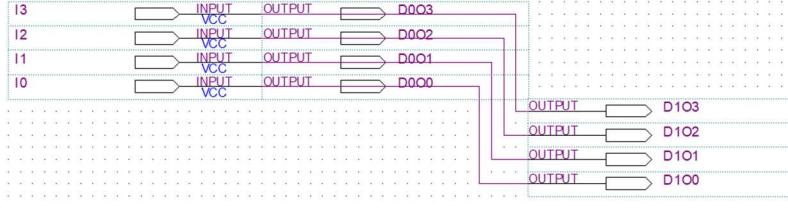


<4BitRCA with Flag Circuit, Functional Simulation>

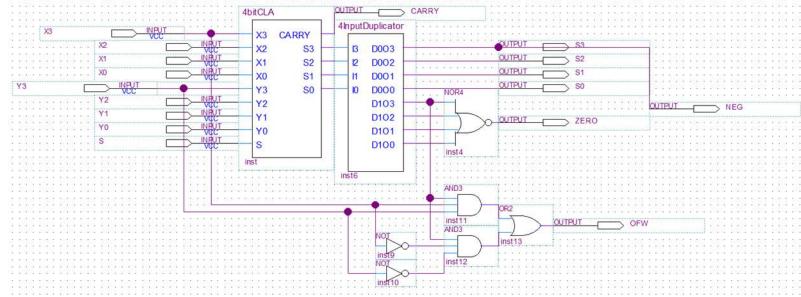
<Overflow(Red), Zero(Green), Negative(Blue)로 표시한 전체 연산 결과>

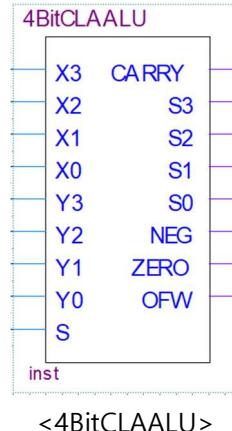
본 회로의 결과를 살펴보면, 전체 연산 결과 표와 대조하였을 때 주어진 X, Y에 대하여 Zero, Negative, Overflow가 정상적으로 검출됨을 확인할 수 있었다. 또한 Overflow1과 Overflow2가 동치임을 다시 확인할 수 있었다.

2-6) ALU with CLA



<4bitDuplicator>





<4BitCLAALU>

2-1~2-5 의 내용과, 기존 CLA 과제의 회로를 참조하여 본 과제에서 사용할 ALU 를 설계하였다.

3) BCDScaler

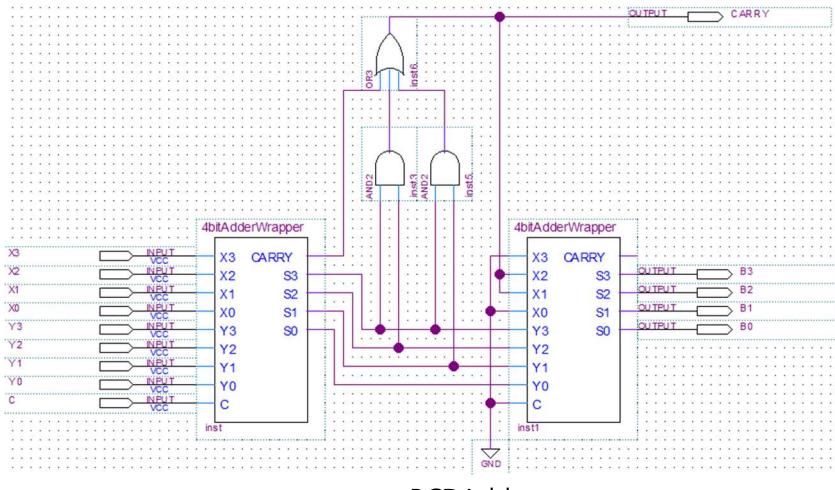
본 과제에서는 BCD 입력을 받아 BCD Carry 와 B3, B2, B1, B0 의 이진 BCD 값을 출력하는 BCDScaler 를 설계 및 활용하였다.

이는 BCD 값이 10 이상일 때 1 의 BCD Carry 값과 입력 BCD 값에서 10 을 뺀 값을 반환하는 회로이다.

BCD Value	BCD CARRY	Scaled BCD Value
0	0	0
1	0	1
2	0	2
3	0	3
4	0	4
5	0	5
6	0	6
7	0	7
8	0	8
9	0	9
10	1	0
11	1	1
12	1	2
13	1	3
14	1	4
15	1	5

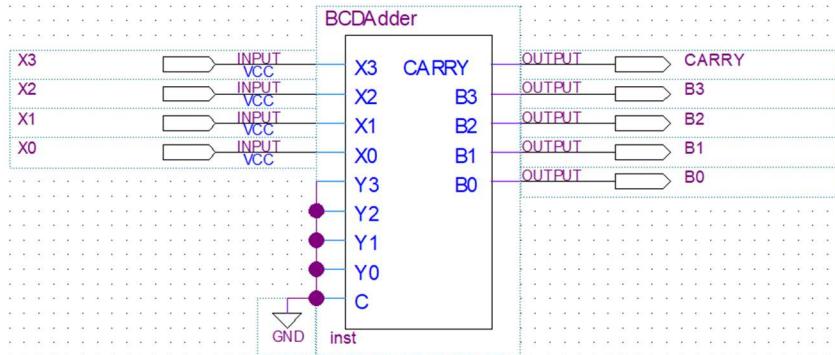
<BCDScaler 의 출력표>

해당 회로를 만들기 위해서는 BCDAdder 를 활용할 수 있다. BCDAdder 는 두 BCD 값을 더한 BCD 값을 출력하는 회로이다. 기존 과제에서 활용한 RCA 4bitAdder 를 이용하여 BCDAdder 를 설계하였다.



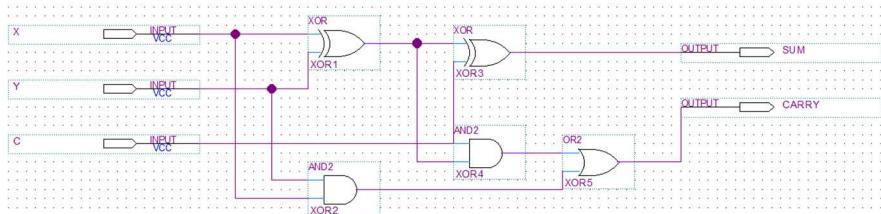
<BCDAdder>

BCDAdder는 두 수를 더한 값이 10 이상이면 CARRY를 1로 처리하기 때문에 한 입력에는 0을, 한 입력에는 스케일할 BCD 값을 주면 BCDScaler로 활용할 수 있다.

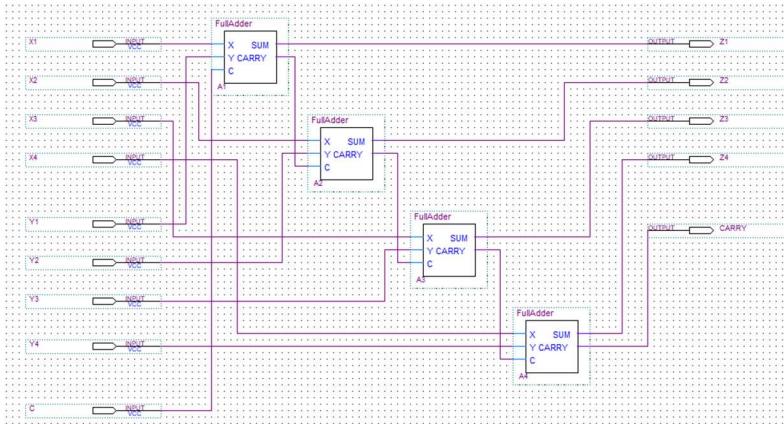


<BCDScaler>

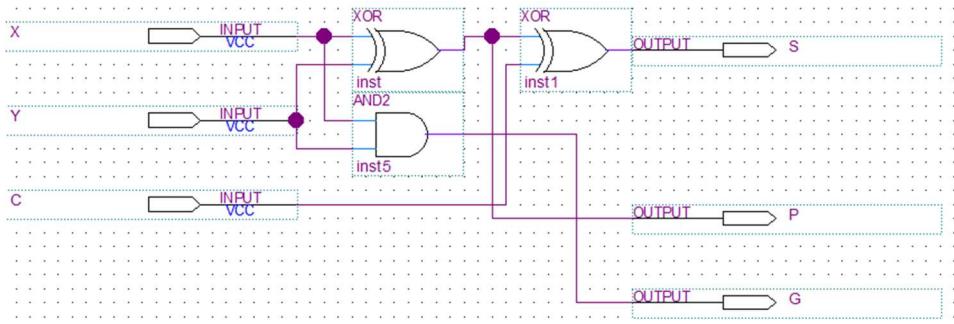
4) 기타 구성 회로 및 완성 회로



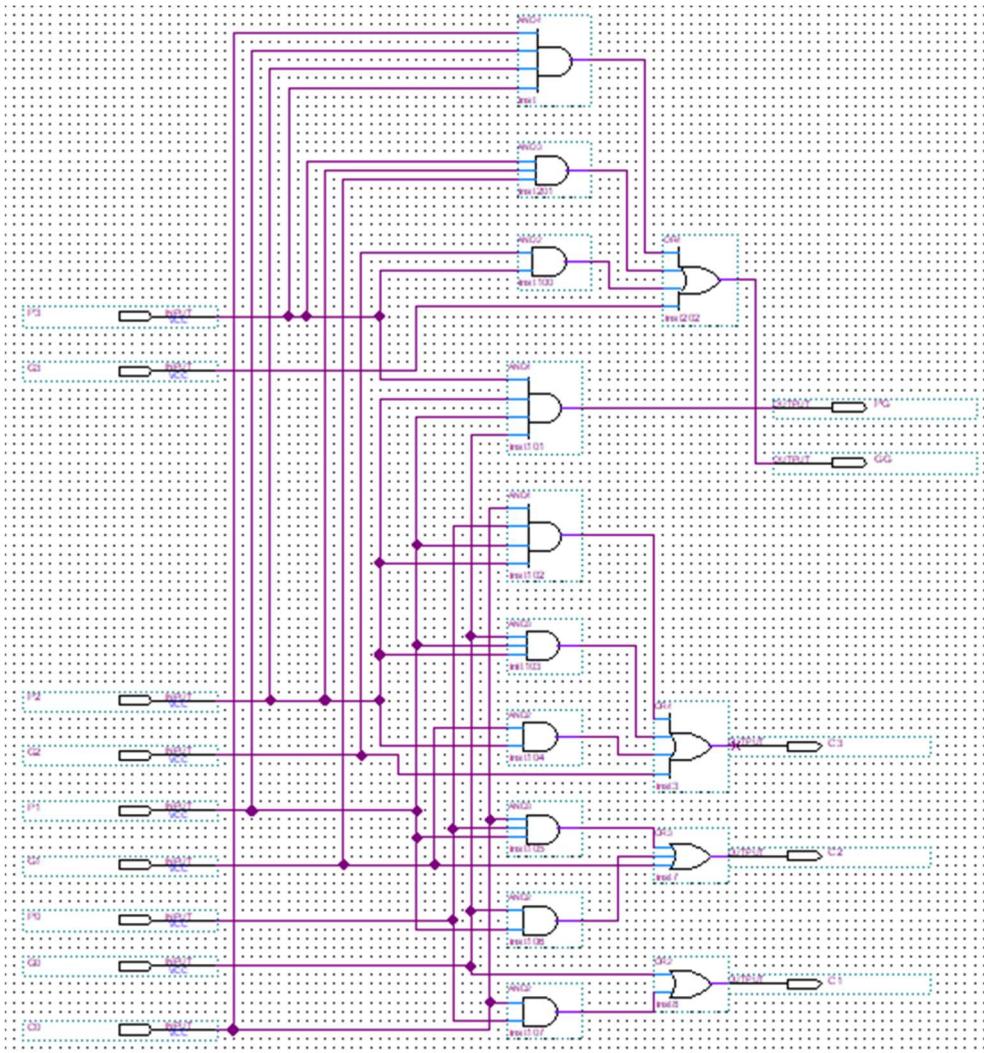
<FullAdder, (전가산기)>



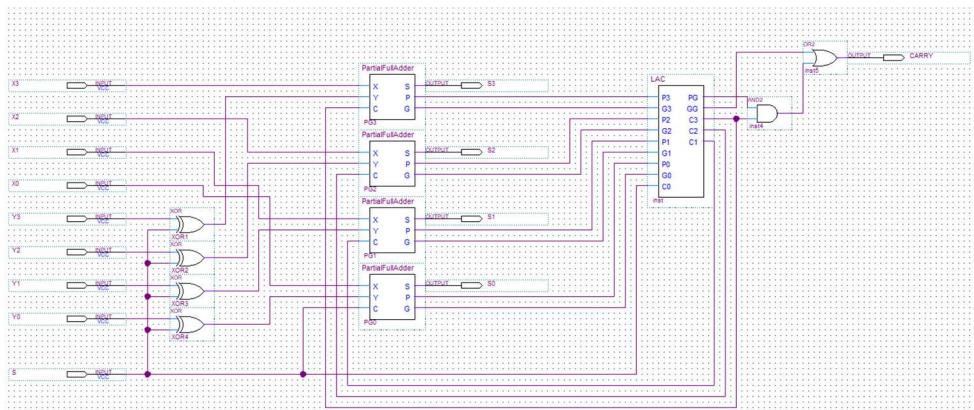
<4bitAdder/4bitAdderWrapper(입력/출력 순서만 다름), 4 비트 RCA 가산기>



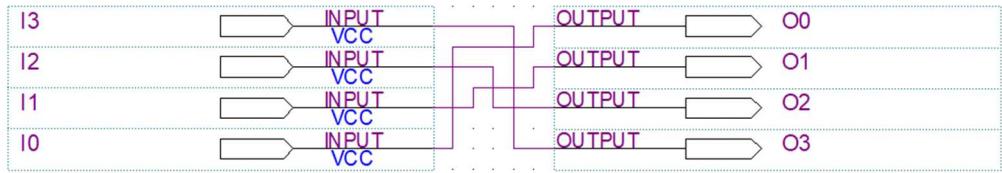
<PartialFullAdder, 부분전가산기>



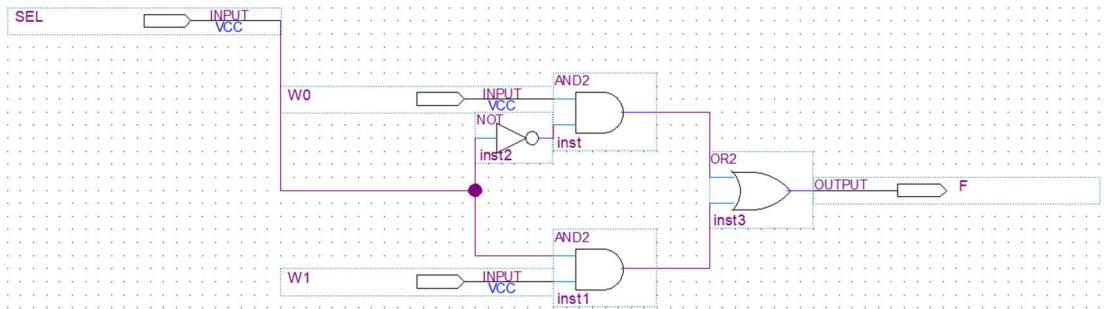
<LAC, CLA 의 캐리 예측 회로>



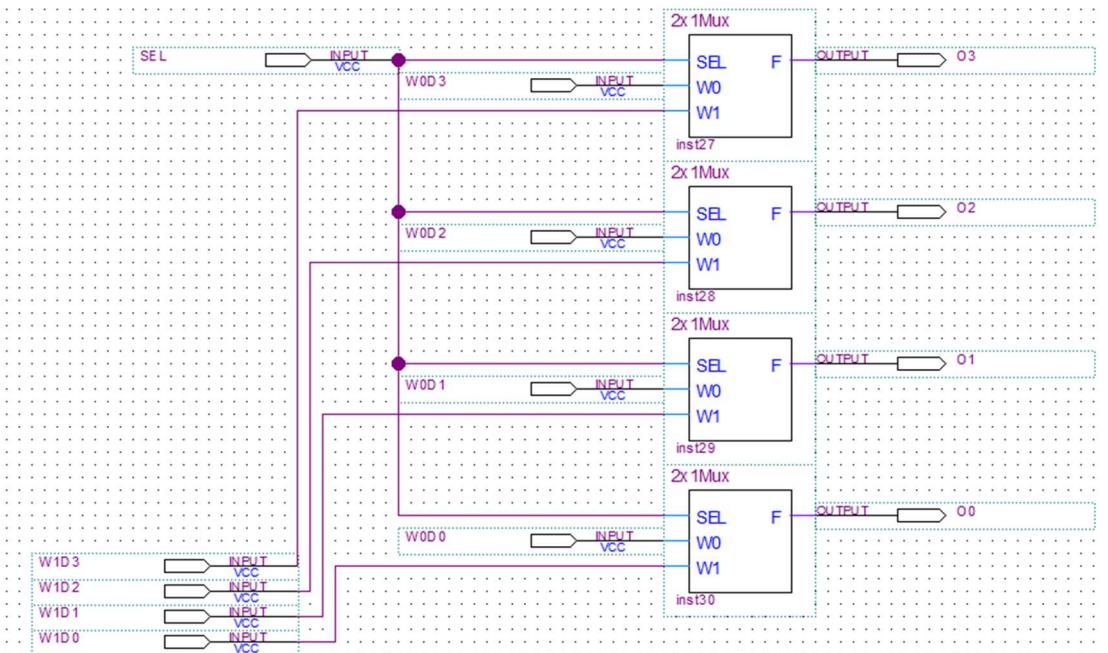
<4bitCLA, 4 비트 CLA 가감산기>



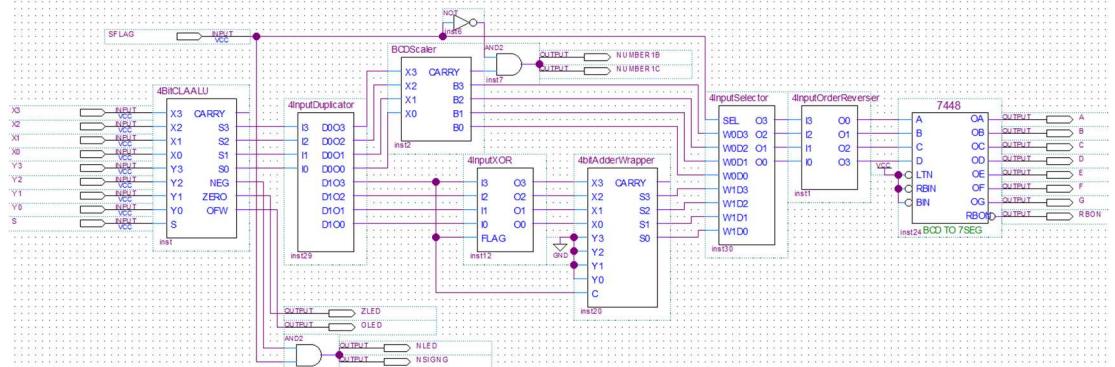
<4InputOrderReverser>



<2x1Mux, 2x1 멀티플렉서>



<4InputSelector, 2x1 Mux 를 이용한 SEL 값에 따라 4 개의 입력을 선택하여 출 출력으로 내보내는 회로>



<최종 설계 회로>

완성 회로는 7 segment display 3 개 (순서대로 음수 출력용, Unsigned 출력용, 숫자 출력용)와 case detection 용 LED 3 개(Zero, Overflow, Negative)를 사용한다.

SFLAG: Singed flag, 이 입력을 1로 주게되면 Unsigned 출력을 실시한다.

X0~X3: 입력 숫자 X

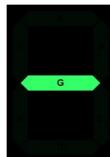
Y0~Y3: 입력 숫자 Y

S: 1 일 때 가산을 수행하게 하는 Subtract 입력

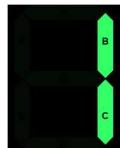
ZLED: 연산 결과가 0 일 때 점등되는 LED

OLED: 연산 결과가 오버플로우일 때 점등되는 LED

NLED: 연산 결과가 음수일 때 점등되는 LED



NSIGNNG: 음수를 표시하는 – 부호



NUMBER1B: Unsigned 출력시 1의 윗부분

NUMBER1C: Unsigned 출력시 1의 아랫부분

A~G: 숫자 출력용 7 segment display 출력

2. 실험 결과

1) 현장 실습 결과



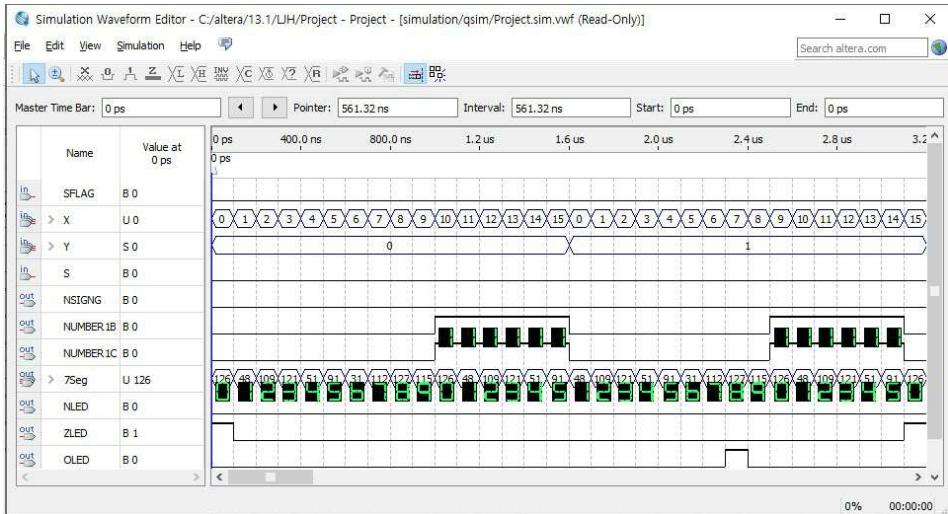
가산 모드(S=0)에서 $0001(1) + 0010(2) = 3$ 을 출력하는 모습



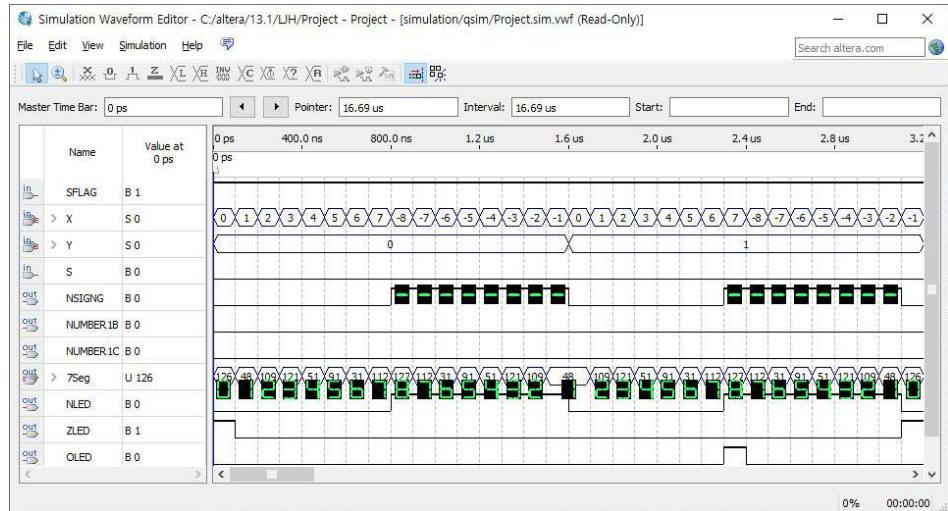
감산 모드(S=1)에서 $0001(1) - 0001(1) = 0$ 을 출력하는 모습 (빨간색 네모 부분은 S 스위치)

실습 시간이 1 시간으로 빠듯하여 현장에서는 가산/감산 기능만을 구현하였습니다. 같은 숫자가 중복으로 출력되는 부분은 7 Segment Pin 의 SEL 값을 미처 제대로 설정하지 못했습니다.

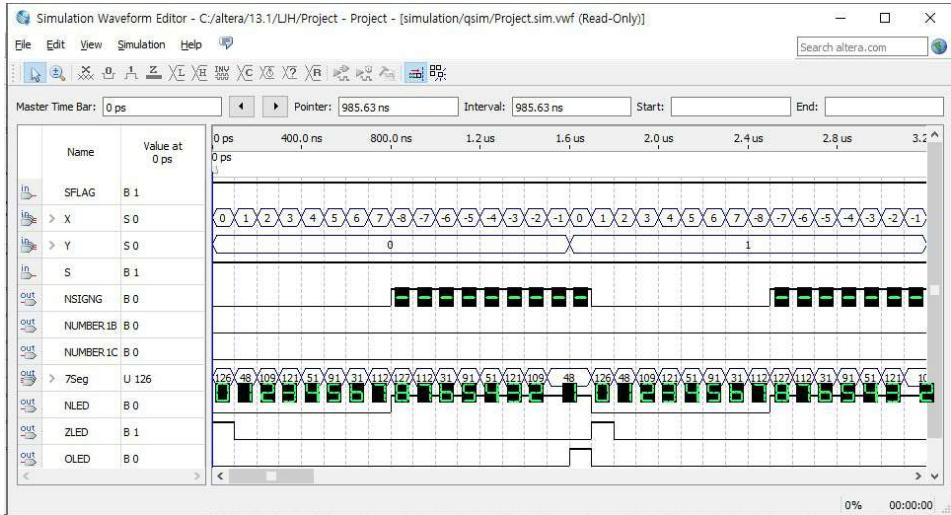
2) 완성 회로 시뮬레이션 결과



<Unsigned(SFLAG=0) & 가산(S=0)>



<Signed(SFLAG=1) & 가산(S=0)>



<Signed(SFLAG=1) & 감산(S=1)>

3. 느낀 점

Overflow, Zero, Negative의 검출 방법을 공부하면서 비정상 연산을 어떤 방법으로 검사하는지 이해할 수 있어서 좋았다.

시간이 부족하여 현장 실습에서 음수 표시등을 구현하지 못한 것이 너무 아쉬워서, 음수/양수 출력 모드를 설계하여 기능을 좀 더 업그레이드 해보았다. 본 과제 하면서 어려움이 많았으나 원하는 대로 작동하는 것을 보니 교수님이 수업 시간에 말씀하셨던 짜릿한 컴파일 성공의 쾌감을 느낄 수 있었다.