컴퓨터 공학 기초 실험2 보고서

실험제목: Subtractor & Arithmetic Logic Unit

실험일자: 2022년 10월 4일 (화)

제출일자: 2022년 10월 5일 (수)

학 과: 컴퓨터정보공학부

담당교수: 공영호 교수님

실습분반: 화요일 0,1,2

학 번: 2019202050

성 명: 이강현

1. 제목 및 목적

A. 제목

Subtractor & Arithmetic Logic Unit

B. 목적

산술 논리 장치인 Arithmetic Logic Unit를 이해하고 입력(a,b)을 받아 opcode의 값에 따라 8가지(000~111)의 연산을 시행하는 4bits ALU와 32bits ALU를 설계해 본다. ALU Status Flags에 대해 알아본다.또한 self-checking testbench를 통한 설계 검증을 알아본다.

2. 원리(배경지식)

Flag란 연산결과에 따른 상태를 알려주는데 C,N,Z,V로 구성된다.

C는 carry의 약자로 자리 올림이 일어났는지에 대한 정보를 가진다.

N은 negative의 약자로 수가 음수인지에 대한 정보를 가진다.

Z는 zero의 약자로 수가 0인지에 대한 정보를 가진다.

V는 overflow의 약자로 수가 범위를 벗어났는지에 대한 정보를 가진다.

그중 C와 V의 차이점은 carry는 일어나더라도 연산결과가 정상적인 값을 가진다. 예를 들면 001과 001을 연산했을 때 010이 되는 것, 또한 010과 110을 연산했을 때 000이되며 3bit를 넘어선 자리올림수가 발생하지만 2와 -2를 더한 0이라는 정상적인 결과를 얻게된다.

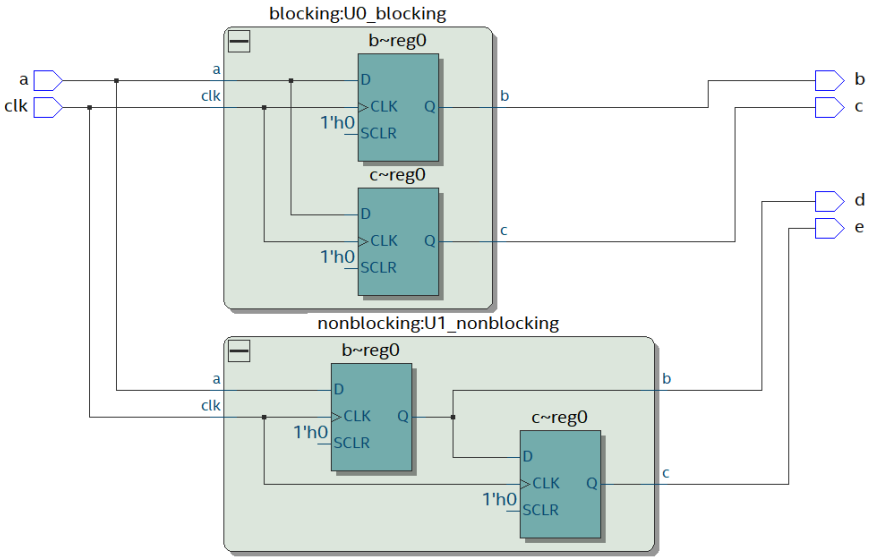
반대로 overflow는 연산결과가 비트의 범위를 벗어나게되면서 비정상적인 결과값을 가지게 되는 것을 의미하는데 101(-3)과 110(-2)을 더했을 때 011이라는 값을 가지고 똑같이 자리올림수를 가지지만 3비트는 two's complement표현방식에 의하면 -4부터 3까지의 범위이고 위의 결과값은 -5이므로 3이라는 결과는 옳지 못한 결과값이다. 따라서 이와 같은 경우는 overflow이다. 결과적으로 carry와 overflow의 차이를 알 수 있는 직관적인 방법은 비트를 벗어난 자리 올림수의 전 캐리와 비트수를 벗어나는 캐리가 둘다 일어나면 carry, 그렇지 않다면 overflow라는 것을 알 수 있다.

이러한 원리로 overflow가 일어나는 논리는 co[3]^co[2]로 구할 수 있다.

Blocking과 non-blocking에 대해

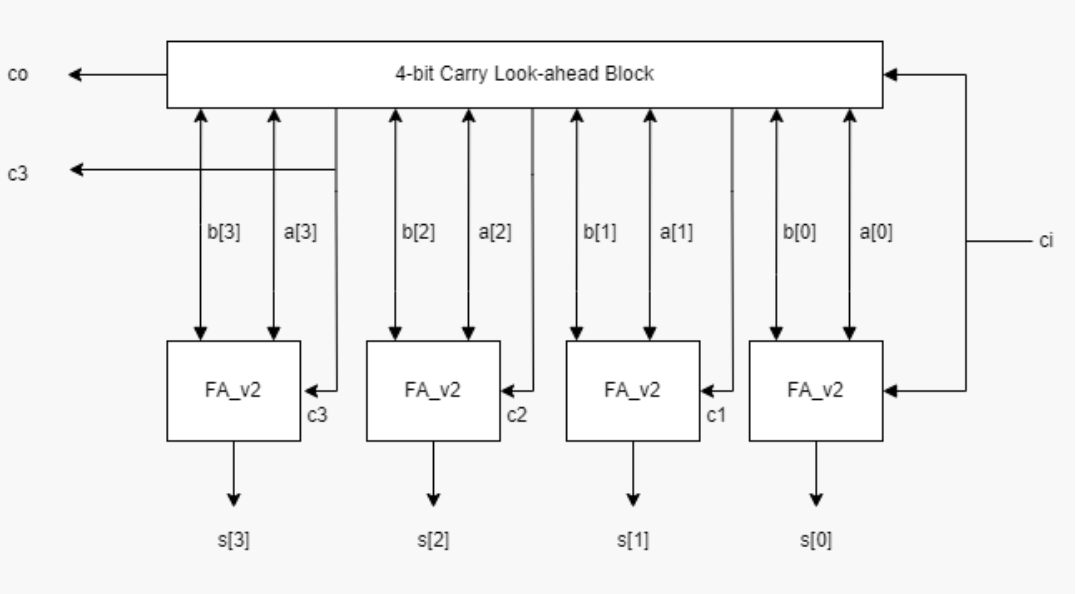
Blocking는 =라는 부호를 사용하여 값을 전달해준다. simulator가 blocking의 경우, one pass로 진행한다. 좌변은 우변이 업데이트 되는 즉시 업데이트 된다. 또한 assignment가 완료될 때까지 절차 내의 실행 흐름이 blocking되는 것이며 한줄의 실행이 끝날때까지 다음 문장이 실행되지 않음을 의미한다.

non-blocking는 <=라는 부호를 사용하여 값을 전달해준다. simulator가 non-blocking의 경우, two pass로 진행된다. 우변은 업데이트되고, 좌변은 오직 예정되어 있다. 좌변은 모듈의 모든 우변이 준비가 되어야 업데이트될 수 있으며 실행 흐름이 blocking 되지 않으므로 non-blocking이라고 한다.

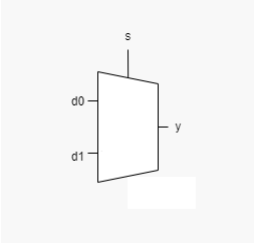


Blocking과 non-blocking 합성결과이며 blocking은 병렬, non-blocking은 직렬로 연결되어 있음을 확인할 수 있다. 또한 blocking에서 b와 c는 동일한 a의 값을 가지지만 non-blocking에서는 b는 a, c는 b의 값을 가진다는 것을 확인할 수 있다.

3. 설계 세부사항



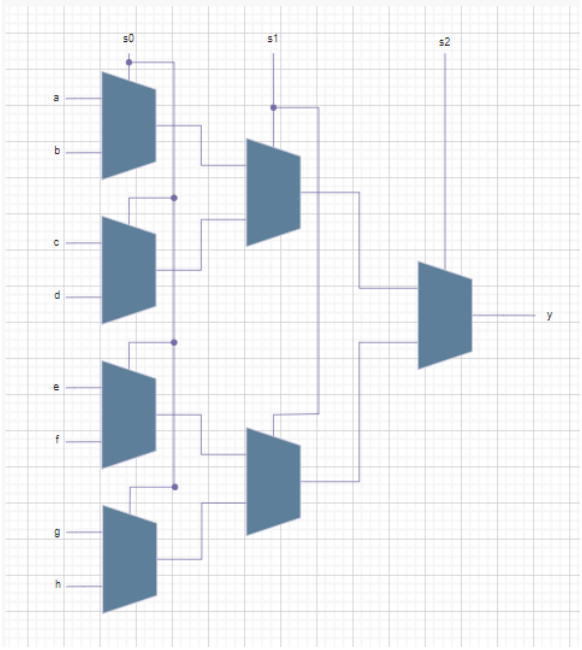
4-bits CLA를 수정하여 만든 회로이다. Overflow를 확인하기 위해 최종 carryout전 캐리또한 output으로 설정해주었다.



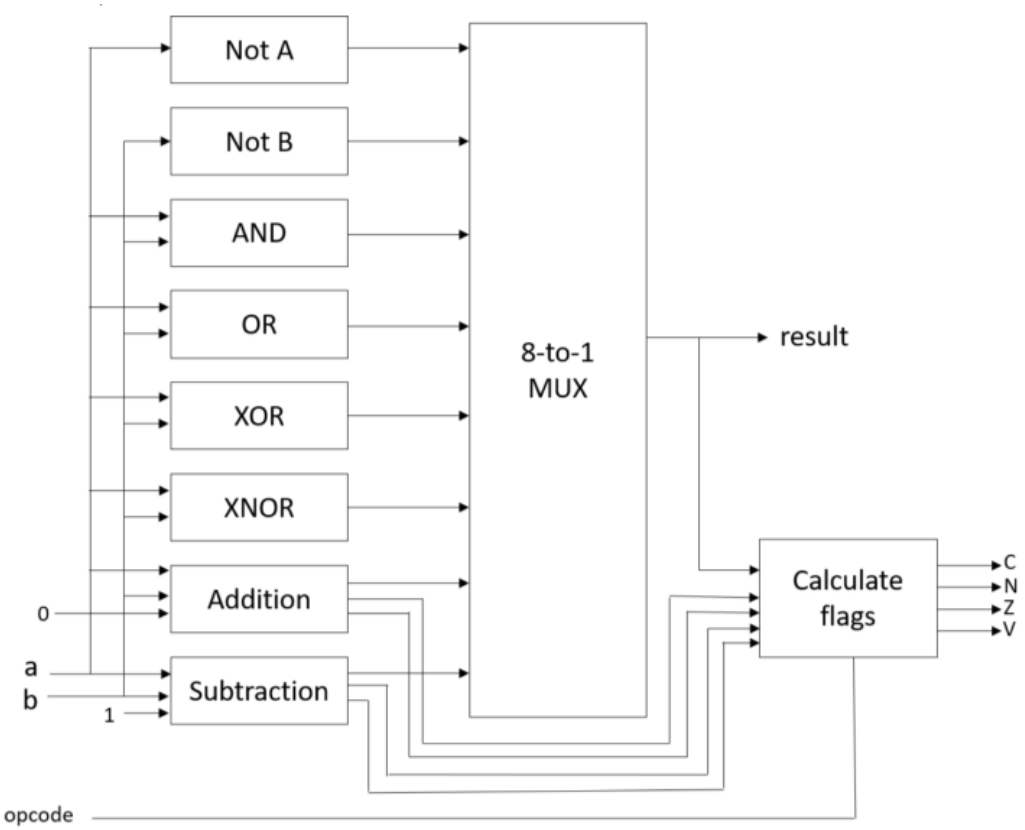
일반적인 2 to 1 mux이다. Inverter와 nand로 구성한다.

이와 같은 진리표를 가진다.

Input			Output
s	d0	d1	Y
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

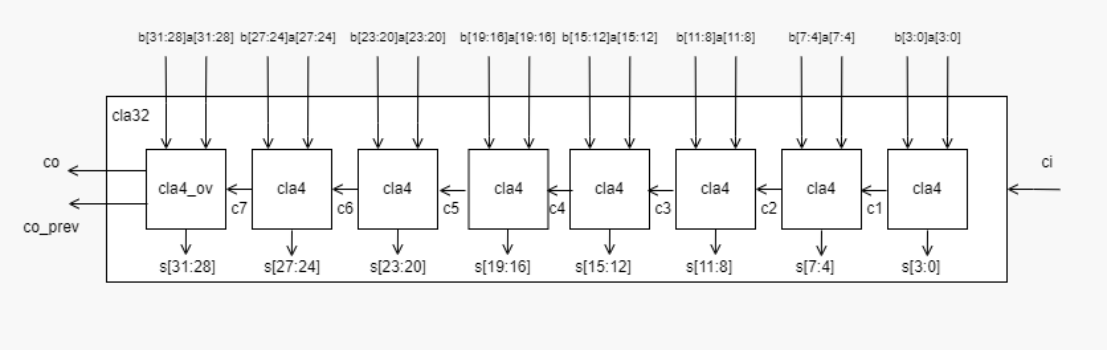


8 to 1 mux이다. 2 to 1 mux 7개를 사용하여 만들었다. Opcode가 3비트이므로 8개의 연산 결과중 어떤 것을 출력값으로 정할지 선택하기 위하여 만든 회로이다. 위의 2 to 1 mux를 7개 사용하여 만들었다.



아래와 같은 명령어들에 따라 result를 가지며 가산기,감산기 그리고 result값에 따라 flag를 설정한다.

Opcode	Operation
3'b000	Not A
3'b001	Not B
3'b010	And
3'b011	Or
3'b100	Xor
3'b101	Xnor
3'b110	Addition
3'b111	Subtraction



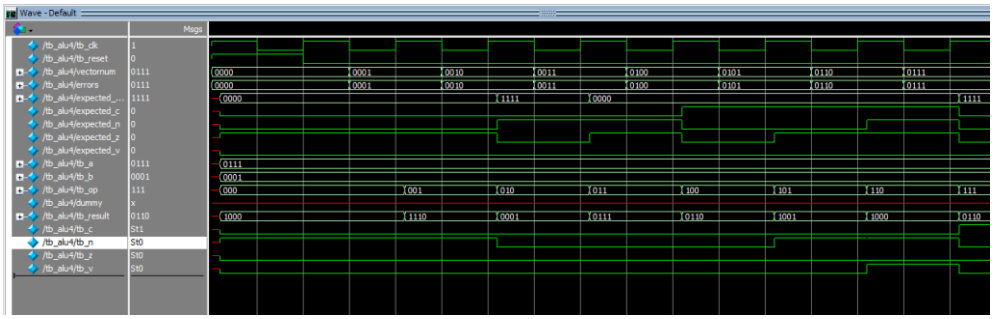
Cla32의 회로의 구현모습이다.

4. 설계 검증 및 실험 결과

A. 시뮬레이션 결과

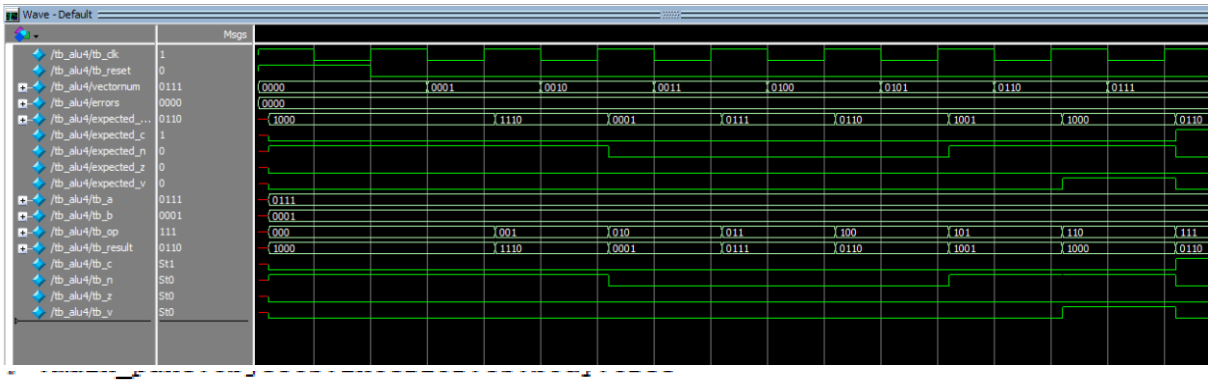
이번 시뮬레이션은 self-checking testbench with testvectors 기법을 사용하여 회로를 검증하였다.

<ALU4>



```
# Error: inputs = a: 0111, b: 0001, op: 000
# outputs = result: 1000, c: 0, n: 1, z: 0, v: 0 (0000 0 0 1 0 expected)
# Error: inputs = a: 0111, b: 0001, op: 001
# outputs = result: 1110, c: 0, n: 1, z: 0, v: 0 (0000 0 0 1 0 expected)
# Error: inputs = a: 0111, b: 0001, op: 010
# outputs = result: 0001, c: 0, n: 0, z: 0, v: 0 (1111 0 1 0 0 expected)
# Error: inputs = a: 0111, b: 0001, op: 011
# outputs = result: 0111, c: 0, n: 0, z: 0, v: 0 (0000 0 1 1 0 expected)
# Error: inputs = a: 0111, b: 0001, op: 100
# outputs = result: 0110, c: 0, n: 0, z: 0, v: 0 (0000 1 0 0 0 expected)
# Error: inputs = a: 0111, b: 0001, op: 101
# outputs = result: 1001, c: 0, n: 1, z: 0, v: 0 (0000 1 0 1 0 expected)
# Error: inputs = a: 0111, b: 0001, op: 110
# outputs = result: 1000, c: 0, n: 1, z: 0, v: 1 (0000 1 1 1 0 expected)
# Error: inputs = a: 0111, b: 0001, op: 111
# outputs = result: 0110, c: 1, n: 0, z: 0, v: 0 (1111 0 0 0 0 expected)
#  8 tests completed with  8 errors
# ** Note: $finish    : C:/intelFPGA_lite/18.1/assign4/alu4/tb_alu4.v(46)
#    Time: 85 ns  Iteration: 1  Instance: /tb_alu4
```

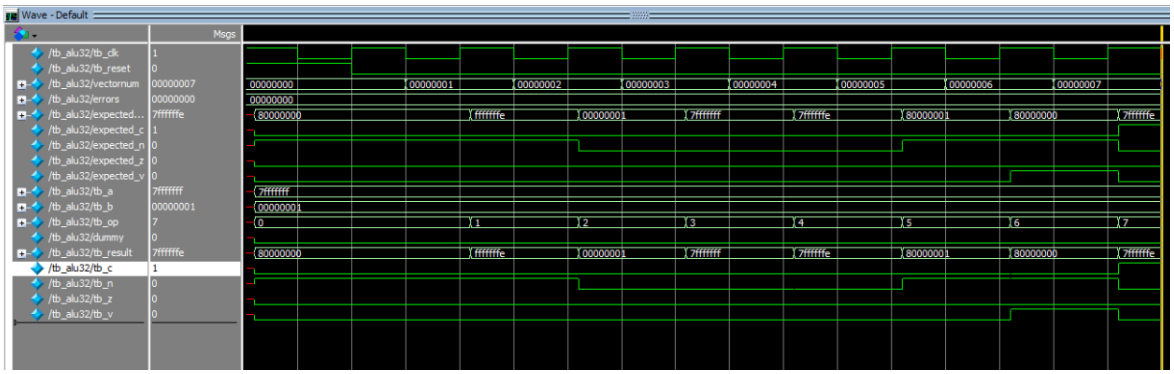
올바르지 않은 테스트벤치값을 넣었을 때 이렇게 error가 나오는 것을 확인할 수 있다. dummy변수는 vector배열이 받는 인자보다 너무 클때를 생각해서 값을 빼내는 용도로 사용했다.




```
# run -all
#  8 tests completed with  0 errors
# ** Note: $finish    : C:/intelFPGA_lite/18.1/assign4/alu4/tb_alu4.v(45)
#    Time: 85 ns  Iteration: 1  Instance: /tb_alu4
# 1
```

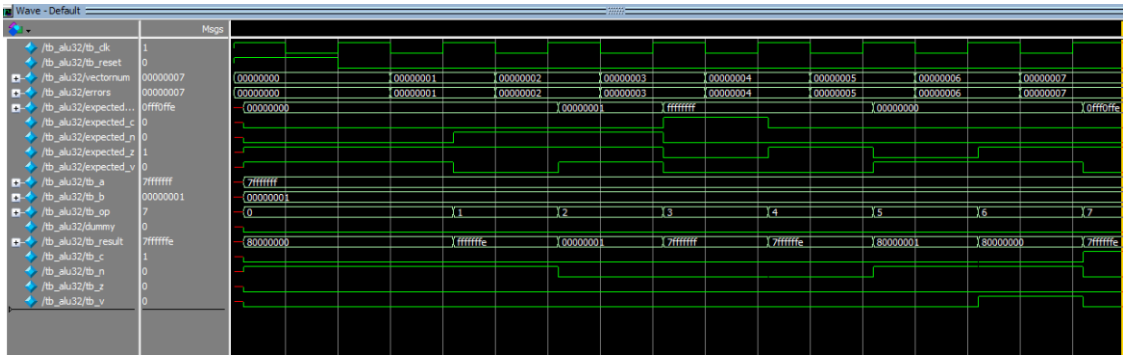
올바른 테스트벤치 예측값을 넣었을 때 잘 예측하고 에러가 발생하지 않는 것을 확인했다.

<ALU32>




```
# view signals
# .main_pane.objects.interior.cs.body.tree
# run -all
#          8 tests completed with          0 errors
# ** Note: $finish    : C:/intelFPGA_lite/18.1/assign4/alu32/tb_alu32.v(46)
#    Time: 85 ns  Iteration: 1  Instance: /tb_alu32
# 1
# Break in Module tb_alu32 at C:/intelFPGA_lite/18.1/assign4/alu32/tb_alu32.v line 46
```

올바른 값을 넣었을 때 테스트 벤치 결과이다. 명령어에 맞게 결과값이 잘 나오고 flag값도 올바르게 측정되는 모습이다.

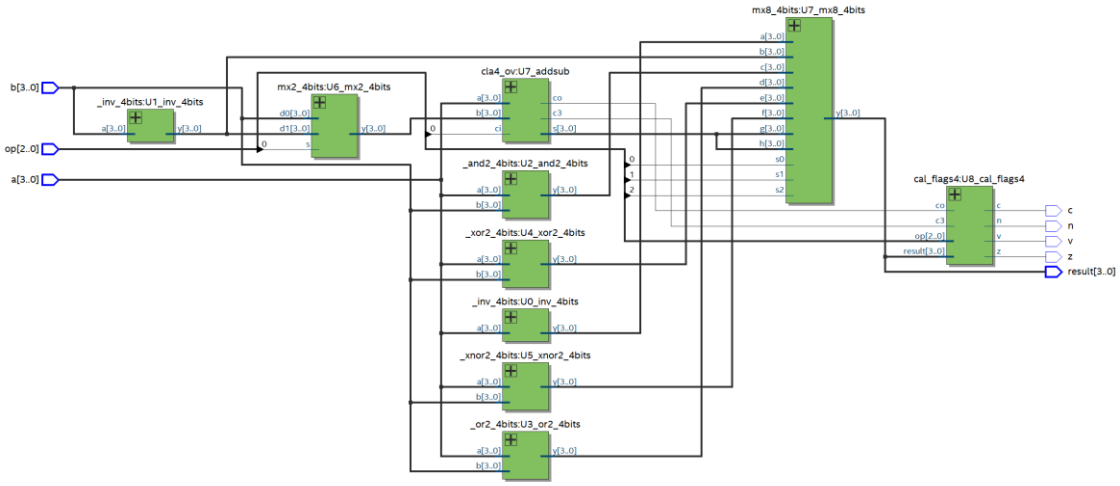



```
# run -all
# Error: inputs = a: 7fffffff, b: 00000001, op: 000
# outputs = result: 80000000, c: 0, n: 1, z: 0, v: 0 (00000000 0 0 1 1 expected)
# Error: inputs = a: 7fffffff, b: 00000001, op: 001
# outputs = result: fffffffe, c: 0, n: 1, z: 0, v: 0 (00000000 0 1 1 0 expected)
# Error: inputs = a: 7fffffff, b: 00000001, op: 010
# outputs = result: 00000001, c: 0, n: 0, z: 0, v: 0 (00000001 0 1 1 1 expected)
# Error: inputs = a: 7fffffff, b: 00000001, op: 011
# outputs = result: 7fffffff, c: 0, n: 0, z: 0, v: 0 (ffffffff 1 0 0 0 expected)
# Error: inputs = a: 7fffffff, b: 00000001, op: 100
# outputs = result: 7ffffffe, c: 0, n: 0, z: 0, v: 0 (ffffffff 0 0 1 0 expected)
# Error: inputs = a: 7fffffff, b: 00000001, op: 101
# outputs = result: 80000001, c: 0, n: 1, z: 0, v: 0 (00000000 0 0 0 1 expected)
# Error: inputs = a: 7fffffff, b: 00000001, op: 110
# outputs = result: 80000000, c: 0, n: 1, z: 0, v: 1 (00000000 0 0 1 1 expected)
# Error: inputs = a: 7fffffff, b: 00000001, op: 111
# outputs = result: 7ffffffe, c: 1, n: 0, z: 0, v: 0 (0fff0ffe 0 0 1 0 expected)
#          8 tests completed with          8 errors
# ** Note: $finish    : C:/intelFPGA_lite/18.1/assign4/alu32/tb_alu32.v(46)
#    Time: 85 ns  Iteration: 1  Instance: /tb_alu32
```

올바르지 않은 테스트벤치값을 넣었을 때는 에러를 확인할 수 있다.

B. 합성(synthesis) 결과

<ALU4>



Alu4의 RTL viewer이다. Inverter, and, or, xor, xnor, add sub의 연산값들이 모두 계산되지만 mux를 통해 명령어에 맞게 결과값으로 출력되고 flag가 계산되는 것을 확인할 수 있다.

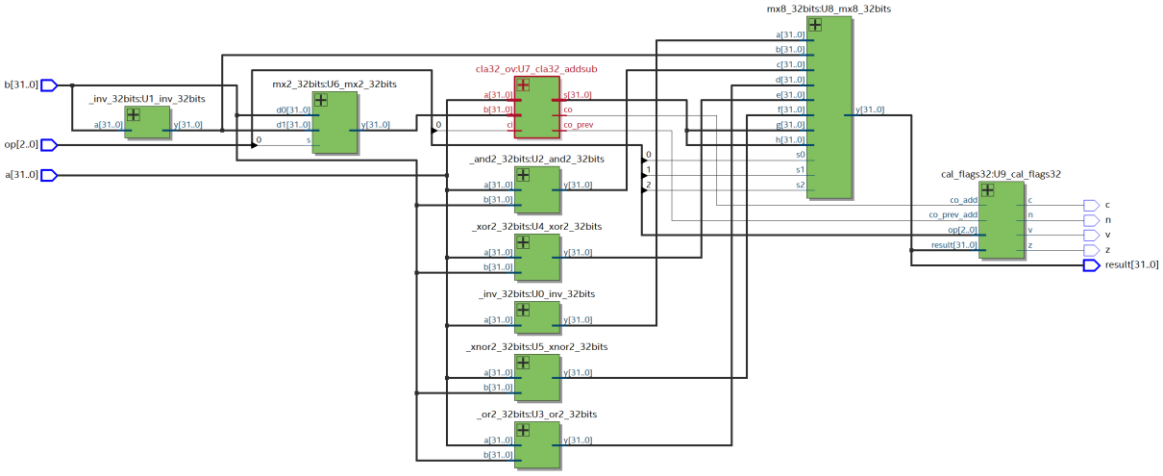
Flow Summary

<<Filter>>

Flow Status	Successful - Tue Oct 04 23:27:29 2022
Quartus Prime Version	18.1.0 Build 625 09/12/2018 SJ Lite Edition
Revision Name	alu4
Top-level Entity Name	alu4
Family	Cyclone V
Device	5CSXFC6D6F31C6
Timing Models	Final
Logic utilization (in ALMs)	N/A
Total registers	0
Total pins	19
Total virtual pins	0
Total block memory bits	0
Total DSP Blocks	0
Total HSSI RX PCSs	0
Total HSSI PMA RX Deserializers	0
Total HSSI TX PCSs	0
Total HSSI PMA TX Serializers	0
Total PLLs	0
Total DLLs	0

Flow summary가 올바른 흐름을 보여준다.

<ALU32>



Alu32의 RTL viewer이다. 다양한 연산들 값이 alu4와 마찬가지로 mux를 통해 명령어에 맞게 출력되는 것을 눈으로 확인가능하다.

Flow Summary

<<Filter>>

Flow Status	Successful - Tue Oct 04 23:05:38 2022
Quartus Prime Version	18.1.0 Build 625 09/12/2018 SJ Lite Edition
Revision Name	alu32
Top-level Entity Name	alu32
Family	Cyclone V
Device	5CSXFC6D6F31C6
Timing Models	Final
Logic utilization (in ALMs)	101 / 41,910 (< 1 %)
Total registers	0
Total pins	103 / 499 (21 %)
Total virtual pins	0
Total block memory bits	0 / 5,662,720 (0 %)
Total DSP Blocks	0 / 112 (0 %)
Total HSSI RX PCSs	0 / 9 (0 %)
Total HSSI PMA RX Deserializers	0 / 9 (0 %)
Total HSSI TX PCSs	0 / 9 (0 %)
Total HSSI PMA TX Serializers	0 / 9 (0 %)
Total PLLs	0 / 15 (0 %)
Total DLLs	0 / 4 (0 %)

Flow summary가 올바른 흐름을 보여준다.

5. 고찰 및 결론

A. 고찰

self checking testbench 기법을 사용해 보았는데 기존의 testbench 보다 코드가 길고 복잡하나 직접 예측한 값들이 에러가 나는지 눈으로 확인할 수 있어 회로를 검증하는 데는 많이 효과적인 것 같았다. Flag라는 것이 연산결과에 대한 정보를 담아주기에 이러한 정보들을 이용하면 다양한 조건문을 활용하여 나중에 복잡한 회로를 구현할 때 더욱 용이하게 잘 사용할 수 있을 것 같다.

B. 결론

기본적인 gate들에서 ha,fa모듈들을 구현하고 이를 통해 RLA,CLA를 구현하고 그것들을 이용해서 이번시간에 배운 ALU까지 만들어보았다. 이는 작은 모듈들을 재사용하여 큰 모듈들을 구현할 수 있고 모듈들의 재사용성이 높기 때문에 굉장히 효율적이라는 생각이 들게 하는 것 같다. 또한 이번 실험에서는 mux의 장점을 극대화한 실험이라는 생각이 들었는데 다양한 연산들을 필요한 상황에 뽑아서 쓸 수 있게 하는 mux의 유용함을 확인할 수 있었다.

베릴로그 문법적인 측면에서도 많은 것을 배울 수 있었는데 c언어에서 사용했던 삼항 연산자나 if문을 통한 조건문의 분기등은 베릴로그로 앞으로 회로를 구현할 때 유용하게 사용될 것 같다.

6. 참고문헌

공영호 교수님/컴퓨터공학기초실험2/2022