컴퓨터 공학 기초 실험2 보고서

실험제목: Subtractor

& Arithmetic Logic Unit (ALU)

실험일자: 2022년 10월 4일 (화)

제출일자: 2022년 10월 6일 (목)

학 과: 컴퓨터공학과

담당교수: 공영호 교수님

실습분반: 화요일 0, 1, 2

학 번: 2021202058

성 명: 송채영

1. 제목 및 목적

A. 제목

Subtractor & Arithmetic Logic Unit (ALU)

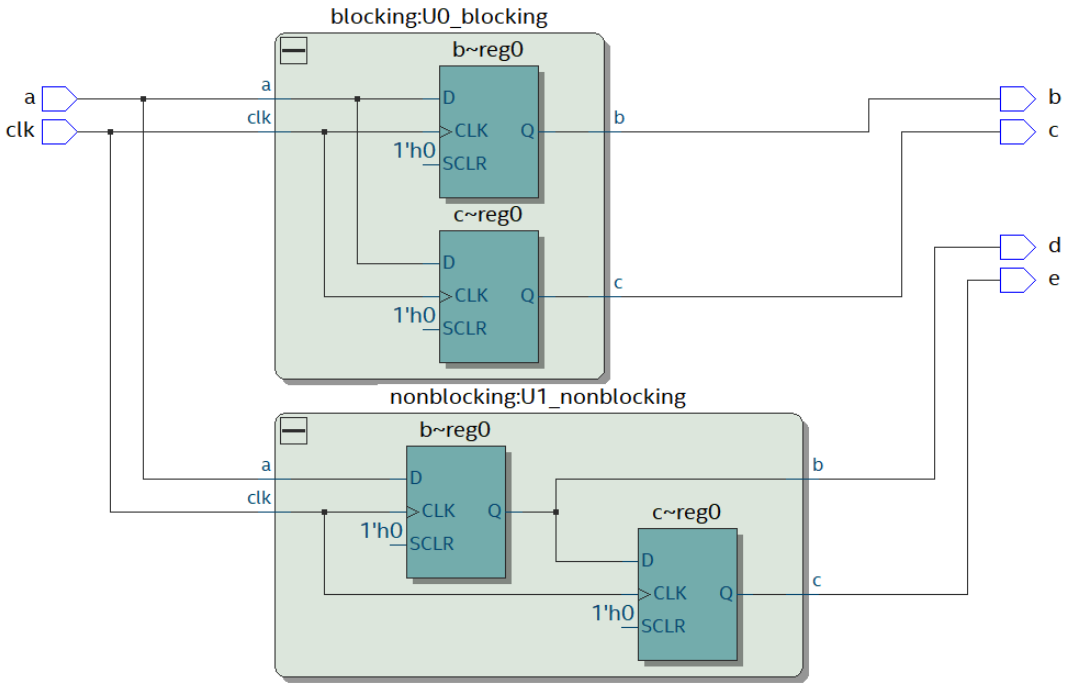
B. 목적

두 숫자의 산술연산과 논리연산을 계산하는 디지털 회로인 산술 논리 장치, Arithmetic Logic Unit에 대해 알아본다. Operator인 3-bit Opcode의 값에 따라 4bits 산술 논리 장치와 32bits 산술 논리 장치를 구현한다. 또한 산술 논리 장치를 검사하는 목적으로 쓰이는 output, C, N, Z, V, ALU Status Flags에 대해 알아본다.

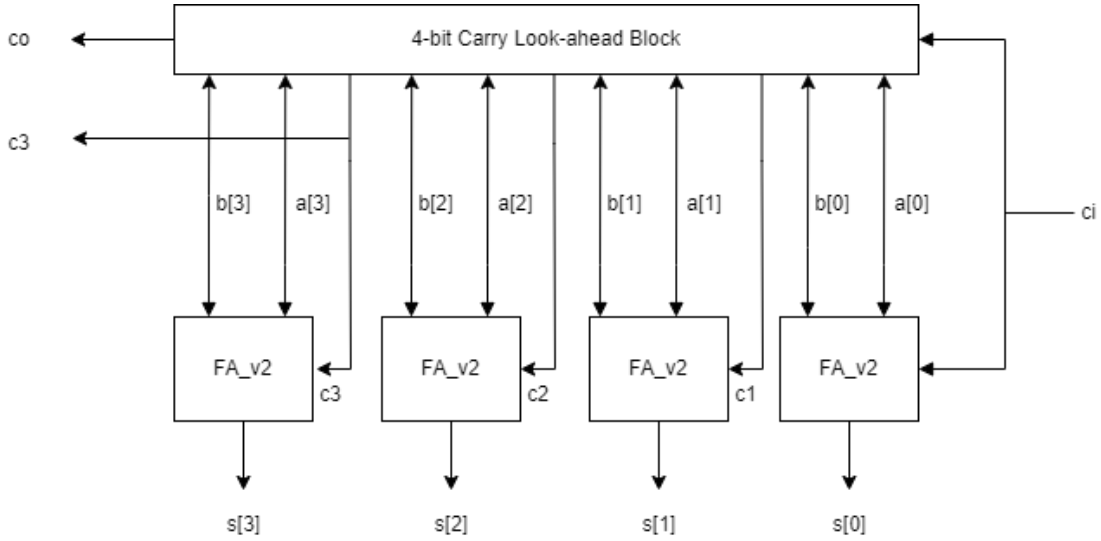
2. 원리(배경지식)

Status Flags중 C는 Carry, N은 negative, Z는 zero, V는 overflow를 뜻한다. Carry는 연산 과정 중 자리 올림이 있을 때 발생한다. Negative는 연산 결과의 most significant bit, msb가 1일 때 발생한다. Zero는 연산 결과가 0일 때를 말한다. Overflow는 연산 결과의 값이 나타낼 수 있는 숫자 범위를 넘어갔을 때 발생한다. 여기서 Carry와 Overflow의 차이에 대해 말해보면, 1byte의 두 값을 더했는데 1byte 이상의 값이 나온 경우 carry bit을 set하고, 최상위비트가 동일한 두 수를 더했는데 최상위비트가 바뀌었을 때 overflow bit를 set하면 된다. 또한 양수와 음수를 더하거나 부호가 같은 뺄셈을 할 때는 overflow가 발생하지 않지만, 두 음수를 더해 양수가 나오거나 양수에서 음수를 뺀 결과가 양수일 때 overflow가 발생한다.

Verilog에서의 blocking과 non-blocking에 대해 말해보면, blocking은 = 기호를, non-blocking은 <= 기호를 사용하여 값을 할당한다. Blocking은 한 줄이 완벽하게 될 때 까지 다음 문장으로 넘어가지 않는다. 즉 호출된 함수가 결과값을 바로 반환하지 않는다. Non-blocking은 함수가 결과값을 바로 반환하며 절차 내의 실행 flow는 blocked 되지 않는다. 아래 사진의 경우 과제 파일의 Verilog file 을 compile한 RTL viewer이다. Blocking의 경우 flipflop이 병렬로 연결되어 연결 없이 값이 바로 들어가는 것을 볼 수 있다. Non-blocking의 경우 값이 순차적으로 진행되도록 하기 위해 직렬로 연결되어 값이 연결되어 input으로 들어가는 것을 볼 수 있다.

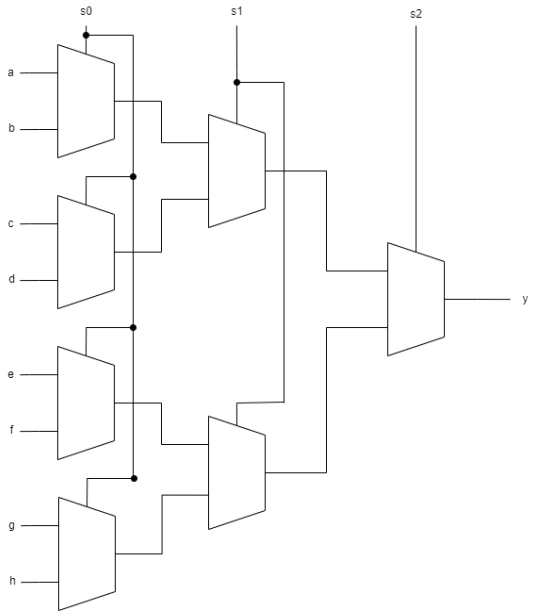


3. 설계 세부사항

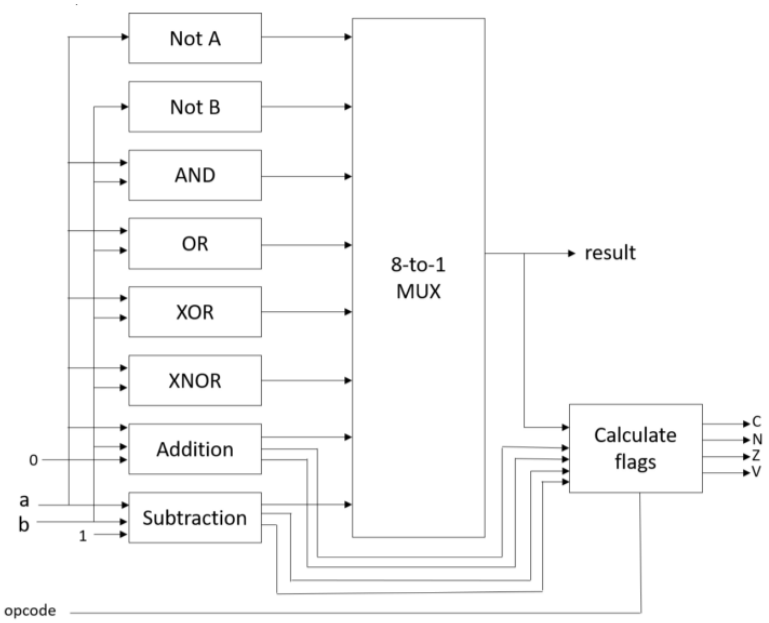


위 사진은 4-bit CLA를 overflow를 검출하기 위해 carry out과 carry[3]을 출력할 수 있도록 수정한 회로이다.

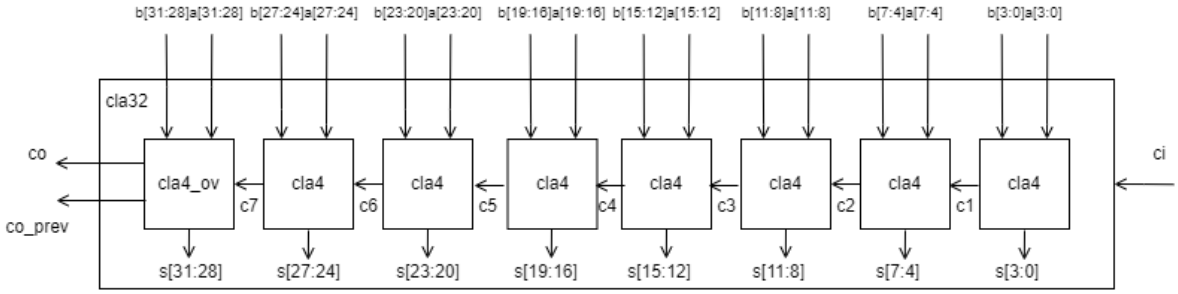
Opcode	Operation
3'b000	Not A
3'b001	Not B
3'b010	And
3'b011	Or
3'b100	Xor
3'b101	Xnor
3'b110	Addition
3'b111	Subtraction



왼쪽의 표는 opcode의 표이며, 오른쪽 사진은 opcode표를 기반으로 만든 alu4의 회로도이다.



8개의 연산의 결과를 Calculate flags로 전달해 즉 C, N, Z, V의 결과를 얻을 수 있다.

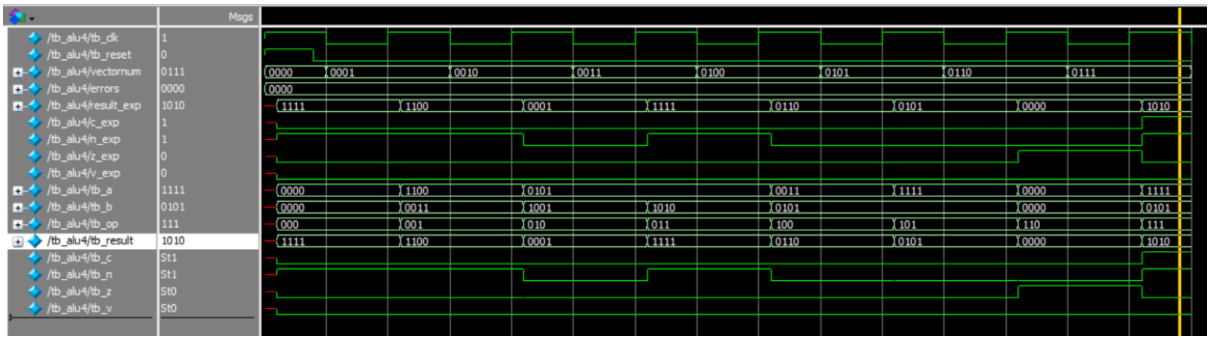


위 사진은 변경된 32-bit CLA로, Carry의 최상위 bit 두 개를 출력해야 하기 때문에 앞선 실습에서 만들었던 32-bit CLA 의 마지막(8 번째) CLA 를 '4-bit CLA to detect overflow'로 변경한다.

4. 설계 검증 및 실험 결과

A. 시뮬레이션 결과

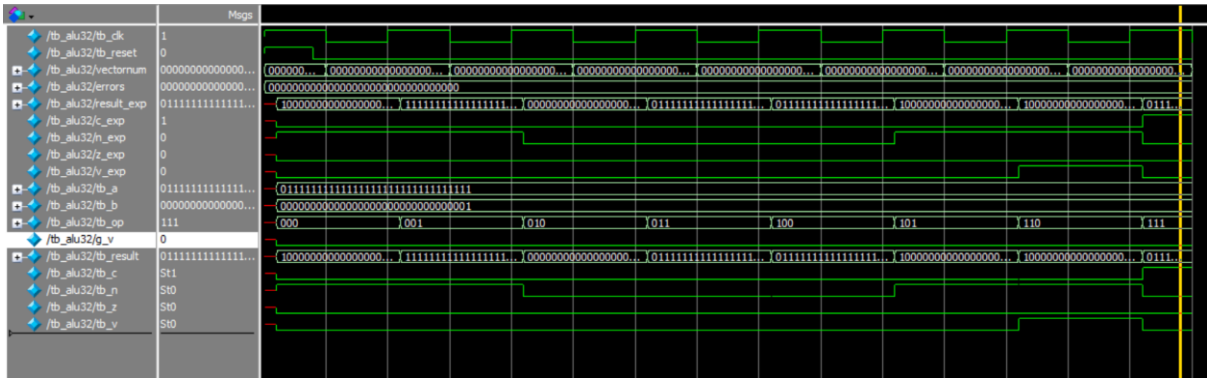
- Alu 4




```
# add wave *
# view structure
# .main_pane.structure.interior.cs.body.struct
# view signals
# .main_pane.objects.interior.cs.body.tree
# run -all
#  8 tests completed with  0 errors
# ** Note: $finish    : C:/intelFPGA_lite/18.1/assignment/2_2021202058_Assignment_04/alu4/tb_alu4.v(48)
#    Time: 75 ns  Iteration: 1  Instance: /tb_alu4
# Error: inputs = a: 0000, b: 0000, op: 000
# outputs = result: 1111, c: 0, n: 1, z: 0, v: 0 (1111 0 1 0 1 expected)
# Error: inputs = a: 1100, b: 0011, op: 001
# outputs = result: 1100, c: 0, n: 1, z: 0, v: 0 (1100 0 1 0 1 expected)
# Error: inputs = a: 0101, b: 1001, op: 010
# outputs = result: 0001, c: 0, n: 0, z: 0, v: 0 (0001 0 0 0 1 expected)
# Error: inputs = a: 0101, b: 1010, op: 011
# outputs = result: 1111, c: 0, n: 1, z: 0, v: 0 (1111 0 1 0 1 expected)
# Error: inputs = a: 0011, b: 0101, op: 100
# outputs = result: 0110, c: 0, n: 0, z: 0, v: 0 (0110 0 0 0 1 expected)
# Error: inputs = a: 1111, b: 0101, op: 101
# outputs = result: 0101, c: 0, n: 0, z: 0, v: 0 (0101 0 0 0 1 expected)
# Error: inputs = a: 0000, b: 0000, op: 110
# outputs = result: 0000, c: 0, n: 0, z: 1, v: 0 (0000 0 0 1 1 expected)
# Error: inputs = a: 1111, b: 0101, op: 111
# outputs = result: 1010, c: 1, n: 1, z: 0, v: 0 (1010 1 1 0 1 expected)
#  8 tests completed with  8 errors
# ** Note: $finish    : C:/intelFPGA_lite/18.1/assignment/2_2021202058_Assignment_04/alu4/tb_alu4.v(48)
#    Time: 75 ns  Iteration: 1  Instance: /tb_alu4
```

Simple testbench의 값을 바탕으로 tb_a, tb_b, opcode, result, flag의 순으로 example.tv를 작성하였다. Testvector을 활용한 self-checking testbench를 통해 0 errors가 나왔고, 올바른 결과값이 나왔음을 알 수 있다. 마지막 사진을 보면 기대 값이 달라 8 errors가 뜨는 것을 알 수 있다.

- Alu 32




```
# add wave *
# view structure
# .main_pane.structure.interior.cs.body.struct
# view signals
# .main_pane.objects.interior.cs.body.tree
# run -all
#          8 tests completed with          0 errors
# ** Note: $finish    : C:/intelFPGA_lite/18.1/assignment/2_2021202058_Assignment_04/alu32/tb_alu32.v(49)
#    Time: 75 ns  Iteration: 1  Instance: /tb_alu32
# Error: inputs = a: 7fffffff, b: 00000001, op: 000
# outputs = result: 80000000, c: 0, n: 1, z: 0, v: 0 (80000000 0 0 1 1 expected)
# Error: inputs = a: 7fffffff, b: 00000001, op: 001
# outputs = result: fffffffe, c: 0, n: 1, z: 0, v: 0 (fffffffe 0 0 1 1 expected)
# Error: inputs = a: 7fffffff, b: 00000001, op: 010
# outputs = result: 00000001, c: 0, n: 0, z: 0, v: 0 (00000001 0 0 0 1 expected)
# Error: inputs = a: 7fffffff, b: 00000001, op: 011
# outputs = result: 7fffffff, c: 0, n: 0, z: 0, v: 0 (7fffffff 0 0 0 1 expected)
# Error: inputs = a: 7fffffff, b: 00000001, op: 100
# outputs = result: 7ffffffe, c: 0, n: 0, z: 0, v: 0 (7ffffffe 0 0 0 1 expected)
# Error: inputs = a: 7fffffff, b: 00000001, op: 101
# outputs = result: 80000001, c: 0, n: 1, z: 0, v: 0 (80000001 0 0 1 1 expected)
# Error: inputs = a: 7fffffff, b: 00000001, op: 110
# outputs = result: 80000000, c: 0, n: 1, z: 0, v: 1 (80000000 0 1 0 0 expected)
# Error: inputs = a: 7fffffff, b: 00000001, op: 111
# outputs = result: 7ffffffe, c: 1, n: 0, z: 0, v: 0 (7ffffffe 0 1 1 1 expected)
#          8 tests completed with          8 errors
# ** Note: $finish    : C:/intelFPGA_lite/18.1/assignment/2_2021202058_Assignment_04/alu32/tb_alu32.v(49)
#    Time: 75 ns  Iteration: 1  Instance: /tb_alu32
```

Simple testbench의 값을 바탕으로 tb_a, tb_b, opcode, result, flag의 순으로 example.tv를 작성하였다. Testvector을 활용한 self-checking testbench를 통해 0 errors가 나왔고, 올바른 결과값이 나왔음을 알 수 있다. 마지막 사진을 보면 기대 값이 달라 8 errors가 뜨는 것을 알 수 있다.

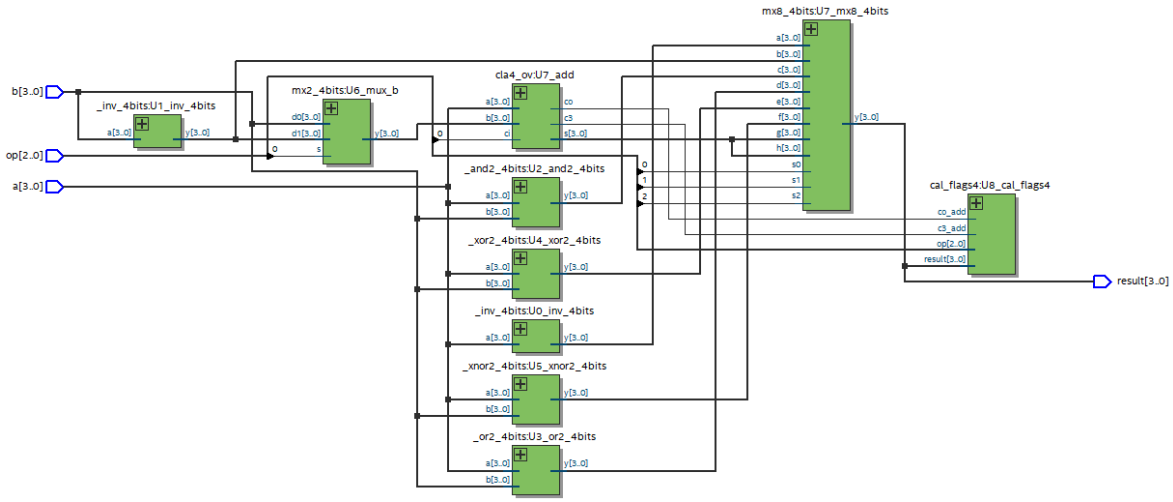
하지만 alu32의 testbench의 경우 alu4와는 다르게 뒤에서부터 계산하였을 때 opcode로 인해 다음과 같은 오류가 생긴다.

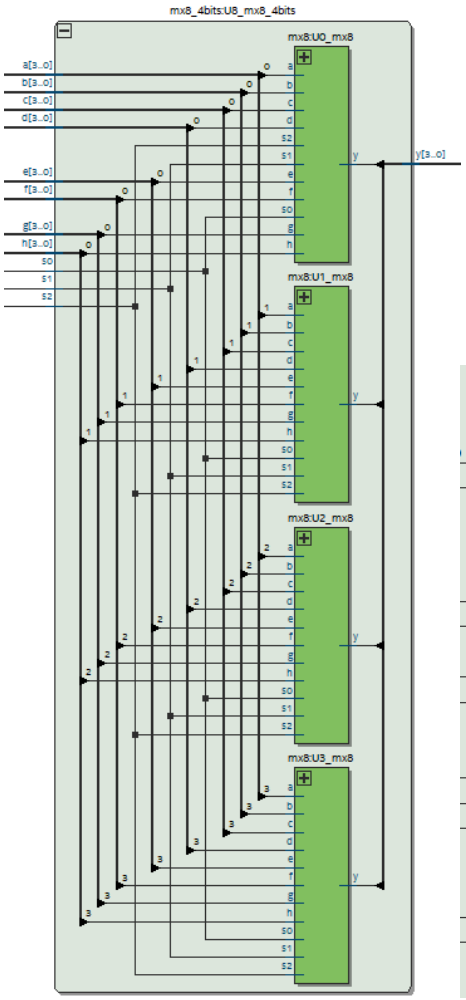
```
# Error: inputs = a: fffffffe, b: 00000002, op: 000
# outputs = result: 00000001, c: 0, n: 0, z: 0, v: 0 (80000000 0 1 0 0 expected)
# Error: inputs = a: fffffffe, b: 00000002, op: 001
# outputs = result: fffffffd, c: 0, n: 1, z: 0, v: 0 (fffffffe 0 1 0 0 expected)
# Error: inputs = a: fffffffe, b: 00000002, op: 010
# outputs = result: 00000002, c: 0, n: 0, z: 0, v: 0 (00000001 0 0 0 0 expected)
# Error: inputs = a: fffffffe, b: 00000002, op: 011
# outputs = result: fffffffe, c: 0, n: 1, z: 0, v: 0 (7fffffff 0 0 0 0 expected)
# Error: inputs = a: fffffffe, b: 00000002, op: 100
# outputs = result: fffffffc, c: 0, n: 1, z: 0, v: 0 (7ffffffe 0 0 0 0 expected)
# Error: inputs = a: fffffffe, b: 00000002, op: 101
# outputs = result: 00000003, c: 0, n: 0, z: 0, v: 0 (80000001 0 1 0 0 expected)
# Error: inputs = a: fffffffe, b: 00000002, op: 110
# outputs = result: 00000000, c: 1, n: 0, z: 1, v: 0 (80000000 0 1 0 1 expected)
# Error: inputs = a: fffffffe, b: 00000002, op: 111
# outputs = result: fffffffc, c: 1, n: 1, z: 0, v: 0 (7ffffffe 1 0 0 0 expected)
#          8 tests completed with          8 errors
# ** Note: $finish    : C:/intelFPGA_lite/18.1/assignment/2_2021202058_Assignment_04/alu32/tb_alu32.v(72)
#    Time: 75 ns  Iteration: 1  Instance: /tb_alu32
# 1
```

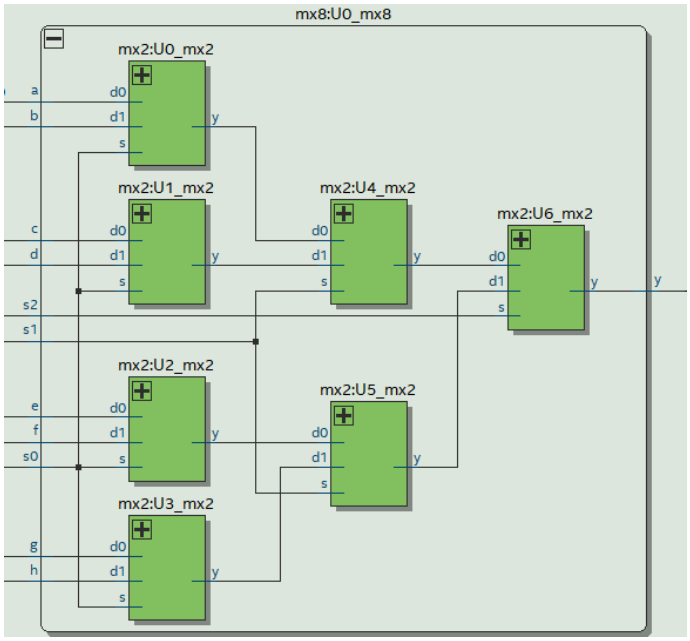
이를 방지하기 위해 비트 수를 맞춰주는 용도인 쓰레기값 g_v를 넣어주었다.

B. 합성(synthesis) 결과

- Alu4







위의 첫 번째 사진은 ALU4 의 RTL viewer 이다. 8 input mux 의 input 으로 들어간 값들로 flags 를 계산하는 것을 알 수 있다. 아래 왼쪽 사진은 8 input mux 인 mx8_4bit 를, 오른쪽 사진은 확대한 사진이다.

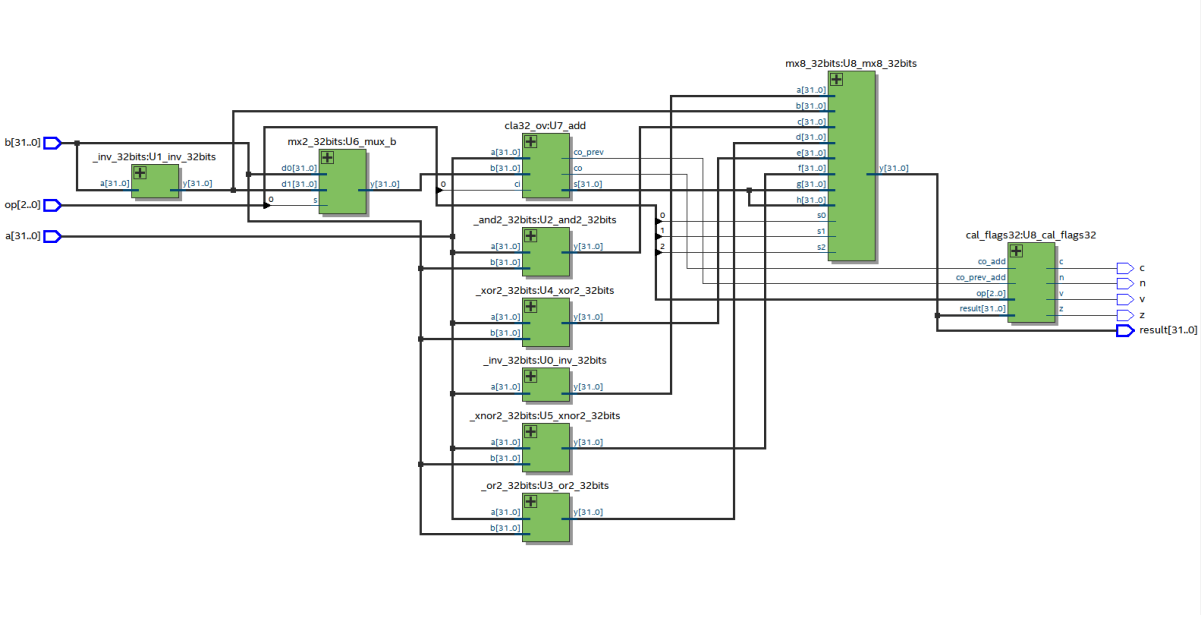
Flow Summary

<<Filter>>

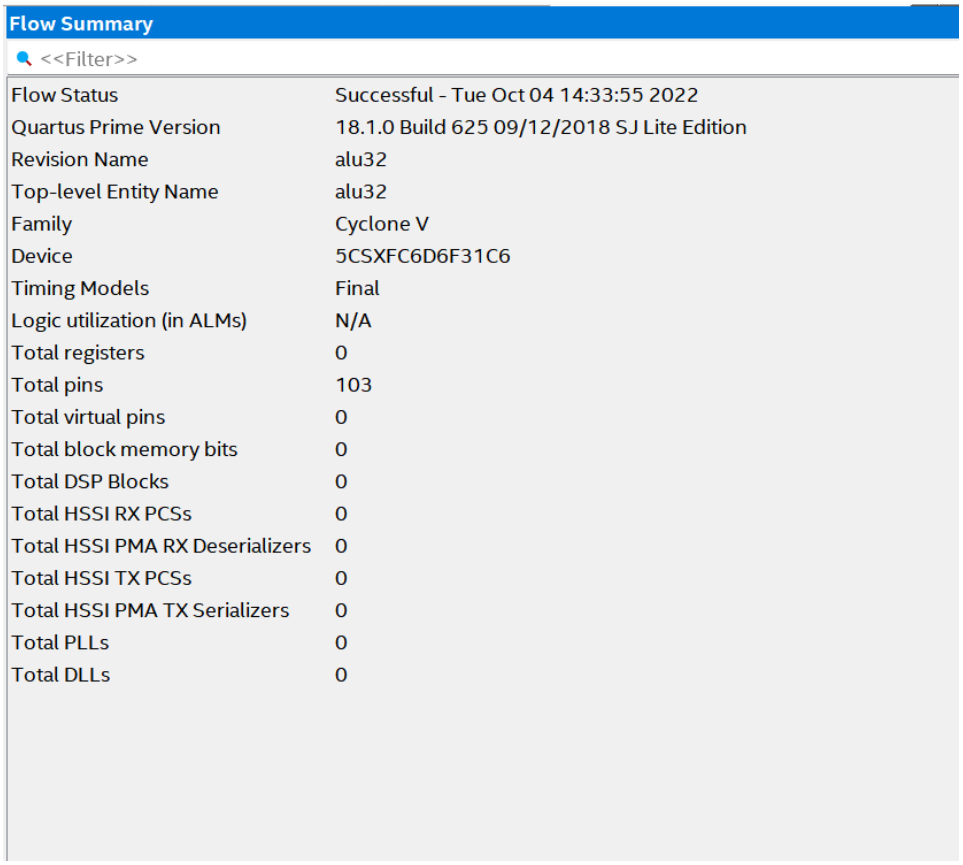
Flow Status	Successful - Tue Oct 04 10:53:14 2022
Quartus Prime Version	18.1.0 Build 625 09/12/2018 SJ Lite Edition
Revision Name	alu4
Top-level Entity Name	alu4
Family	Cyclone V
Device	5CSXFC6D6F31C6
Timing Models	Final
Logic utilization (in ALMs)	7 / 41,910 (< 1 %)
Total registers	0
Total pins	19 / 499 (4 %)
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 %)

Alu4의 flow summary이다. 총 19개의 pin을 사용한 것을 알 수 있으며 성공적으로 컴파일이 완료된 것을 알 수 있다.

- Alu32



위의 사진은 ALU32의 RTL viewer이다. ALU4와 마찬가지로 8 input mux의 input으로 들어간 값들로 flags를 계산하는 것을 알 수 있으며, 각 gate들은 32bits의 input을 가지는 것을 알 수 있다.



Flow Summary

<<Filter>>

Flow Status	Successful - Tue Oct 04 14:33:55 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)	N/A
Total registers	0
Total pins	103
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

Alu32의 flow summary이다. 총 103개의 pin을 사용한 것을 알 수 있으며 성공적으로 컴파일이 완료된 것을 알 수 있다.

5. 고찰 및 결론

A. 고찰

Testvector를 이용한 testbench를 처음 접하니 익숙하지 않아 더 어렵고 시간도 오래 걸렸지만, 디지털논리회로 강의와 강의자료를 여러 번 읽어보고 이해하고 코드를 짜니 조금은 수월했다. 하지만 error의 개수를 알려주니 틀렸는지 맞았는지 쉽게 알 수 있어 좋았다. 또한 example.tv파일을 어디에 만들어줘야 하는 점과, alu32의 self testbench에서 비트 수가 맞지 않아 오류가 나는 점에 대해 물어보고 도움을 받아 진행하였다. 다음 실습 때 testvector을 이용한 testbench를 작성한다면 좀 더 수월하게 진행할 수 있을 것 같다.

B. 결론

2, 3주차 때 짰던 코드를 다시 불러오고 추가하고 수정해서 새로운 코드를 만들어내서 실습을 진행할수록 verilog에 익숙해지는 것 같다. 또한 다른 수업에서 배우는 것, 예를 들

어 carry와 overflow 같은 개념들이 겹쳐 개념을 이해하고 이론적인 부분에 대해 작성할 때 수월하게 진행할 수 있었던 것 같다. Carry와 overflow 같은 경우 직접 숫자로 예시를 들어가며 개념에 이해를 도왔다. Testvector를 이용한 testbench에 대해 처음 접해보았다. 기존에는 simple testbench를 사용하여 waveform을 확인하였는데 self checking testbench를 이용하니 코드에서 테스트를 하였을 때 잘못되거나 잘됐을 경우를 출력해주어 디버깅을 용이하게 한다는 점을 알았다. 또한 self checking testbench의 경우 positive edge와 negative edge의 동작을 정의해 input에 할당하고, input은 tv파일에서 온다는 것을 알 수 있었다.

6. 참고문헌

공영호 교수님/디지털논리회로2 강의자료/광운대학교 컴퓨터 공학과 2022 강의자료
공영호 교수님/컴퓨터공학기초실험2/광운대학교/2022 강의자료