컴퓨터 공학 기초 실험2 보고서

실험제목: Assignment_03 CLA

실험일자: 2020년 09월 28일 (월)

제출일자: 2020년 10월 05일 (월)

학 과: 컴퓨터정보공학부

담당교수: 공진흥 교수님

실습분반: 월요일 0, 1, 2

학 번: 2019202052

성 명: 김 호 성

1. 제목 및 목적

A. 제목

Subtractor & Arithmetic Logic Unit (ALU)

B. 목적

- 4bit ALU에 대해 이해하고, 프로그래밍한다.
- 32bit ALU에 대해 이해하고, 프로그래밍한다.

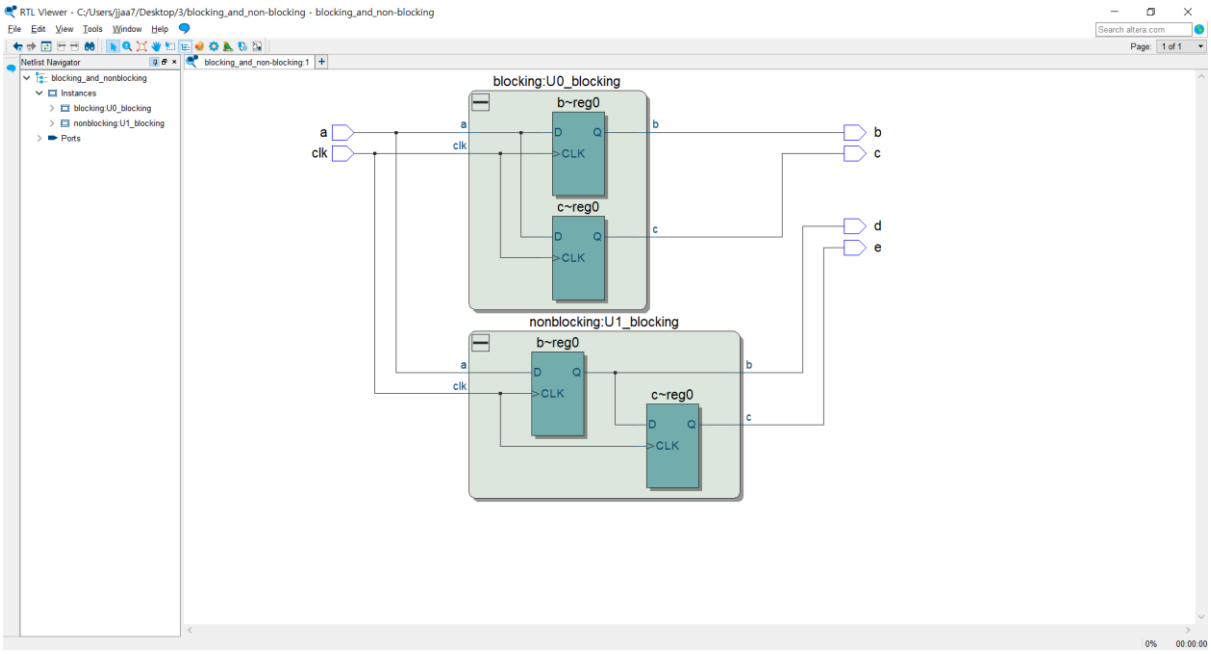
2. 원리(배경지식)

- carry와 overflow의 차이

캐리는 최상위 비트(MSB)에서 그 위의 비트로 자리올림이 발생하는 것을 의미한다. 캐리 자체로는 오류 발생과 관련 없다.

반면 오버플로우는 연산 결과 값이 주어진 비트 수로 표현될 수 있는 범위를 넘긴 것을 의미한다. 수의 범위를 넘었기 때문에 오버플로우는 오류 발생을 의미한다.

- blocking과 non-blocking을 컴파일하고 차이에 대해 설명



호출된 함수가 바로 리턴해서 호출한 함수에게 제어권을 넘겨주고, 호출한 함수가 다른 일을 할 수 있는 기회를 줄 수 있으면 NonBlocking이다.

그렇지 않고 호출된 함수가 자신의 작업을 모두 마칠 때까지 호출한 함수에게 제어권을 넘겨주지 않고 대기하게 만든다면 Blocking이다.

3. 설계 세부사항

4-BIT ALU의 경우 gates.v(여러 게이트), fa_v2.v(1bit full adder), clb4.v(Clock Look Ahead block), cla4_ov.v(4-bit Clock Look Ahead), mx2.v, mx8.v, mx8_4bits, cal_flags(conditional register), alu4.v, tb_alu4.v 총 10가지의 파일이 필요하다.

ALU에서 제공하는 기능으로는 NOT A, NOT B, AND, OR, XOR, XNOR, ADD, SUB이 있으며 SUB의 경우 B값을 반전시킨 후 +1한 값을 A와 더하는 방식으로 설계된다. 여기서 B값을 반전시킨 후 1을 더한 값은
2's complement의 정의로 실제 컴퓨터에서 연산하는 표준 방법에 준한다.
또한, ALU내에서 특정 조건이나 상황이 만족되었다, 이 result 값이 현재 이런 상태임을 나타내기 위해 conditional register모듈을 제작한다.
이 모듈의 경우 연산결과 carry가 발생하는 경우를 나타내는 C, 연산결과의 sign bit(MSB)가 1인, 즉 음수를 나타내는 N, 연산결과가 0을 나타내는 Z, overflow가 일어나 정상적인 값이 아님을 나타내는 V가 있다.
결론적으로 ALU는 제공하는 8개의 기능들 중 op값을 통해 하나를 선택하여 출력해주는 형태이다. (그 결과의 상태를 알 수 있도록 하는 CNZV를 포함해서)

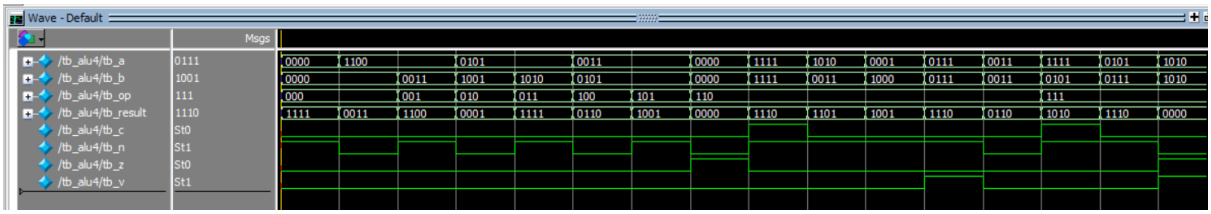
32-BIT ALU는 먼저 만든 4-BIT ALU를 묶어서 만들 수 있는데, 여기서 주의해야할 점은 conditional flag를 만들어주는 모듈에서 MSB bit가 4번째 bit가 아닌 32번째 bit임을 유의해야한다. 또한, 32-BIT ALU의 경우 CLA와 RCA중 CLA를 사용하는 방식의 연산이 더욱 빠르다는 것을 인지하고 설계해야 한다.
4-BIT ALU 8개를 병합해서 만드는 과정이기 때문에 인자 값을 정확히 연결해주는 노력이 필요하다.

4. 설계 검증 및 실험 결과

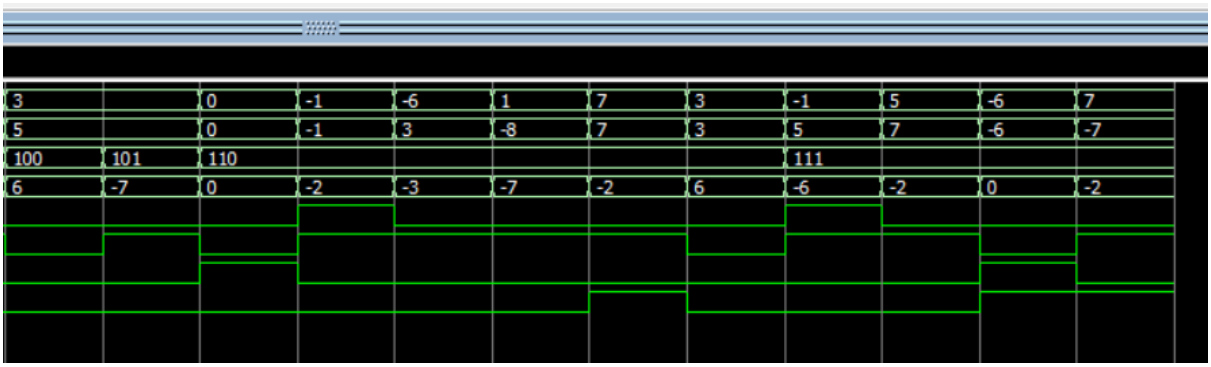
A. 시뮬레이션 결과

4-bit ALU (using self-checking testbench)

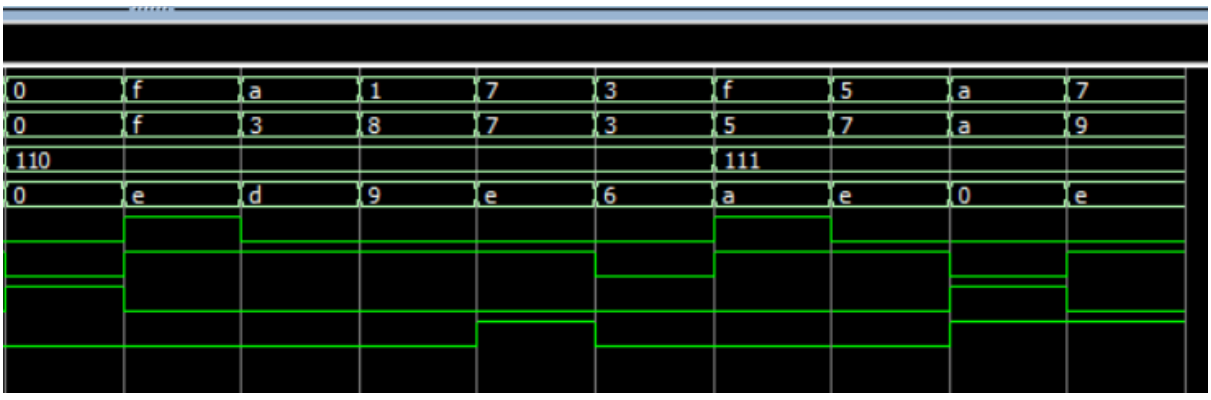
Waveform (binary mode)



Waveform (decimal mode)



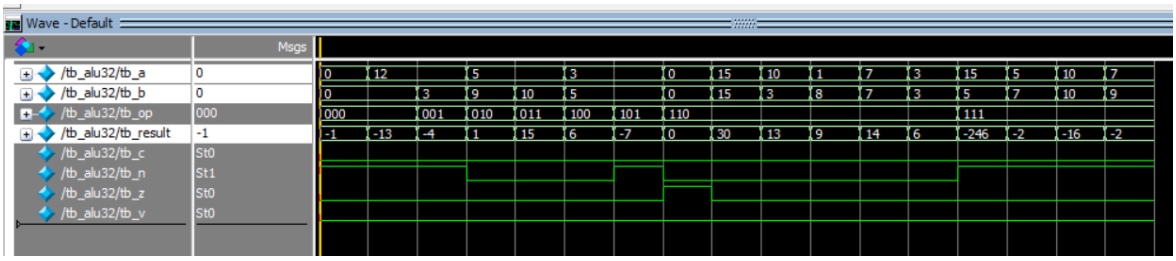
Waveform (hexa mode)



overflow경우를 제외하고 모든 값이 제대로 나오는 것을 확인함.

32-bit ALU (using self-checking testbench with test vectors)

waveform

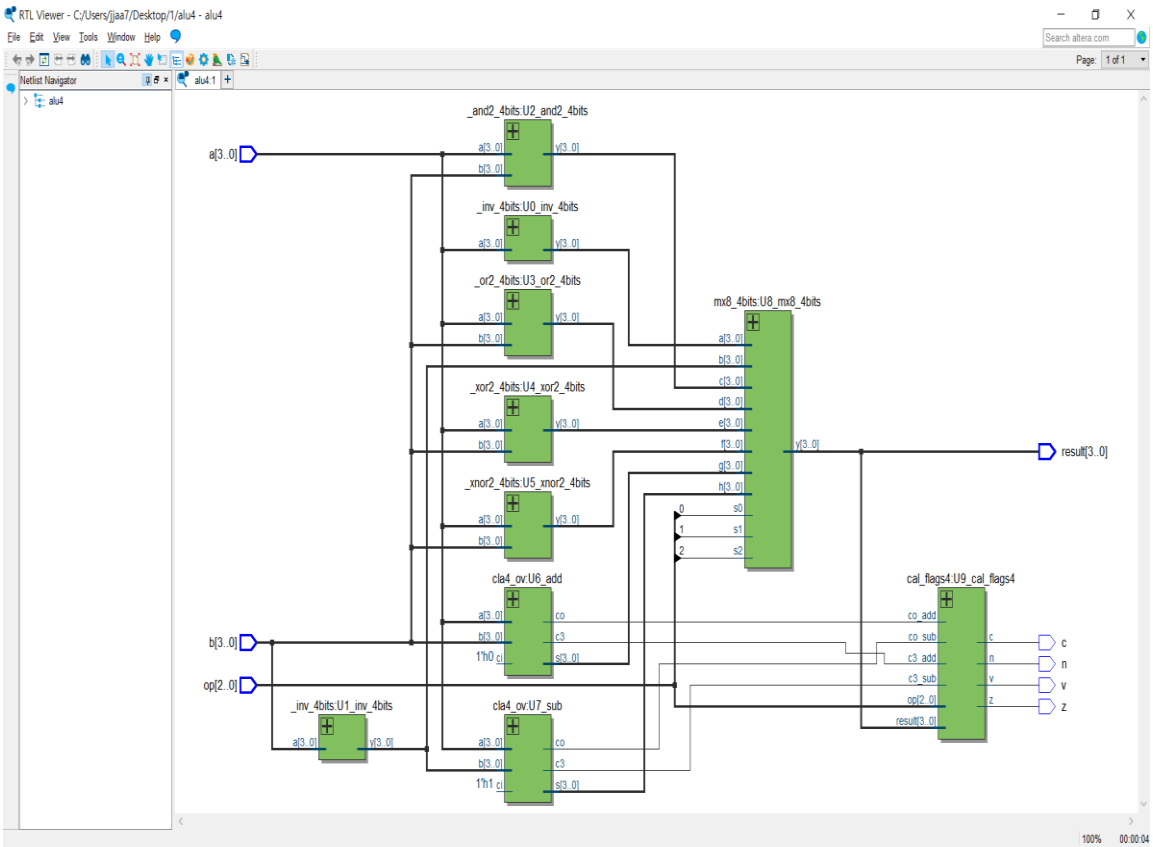


SUB에서 값이 이상해진 것을 확인할 수 있다.

B. 합성(synthesis) 결과

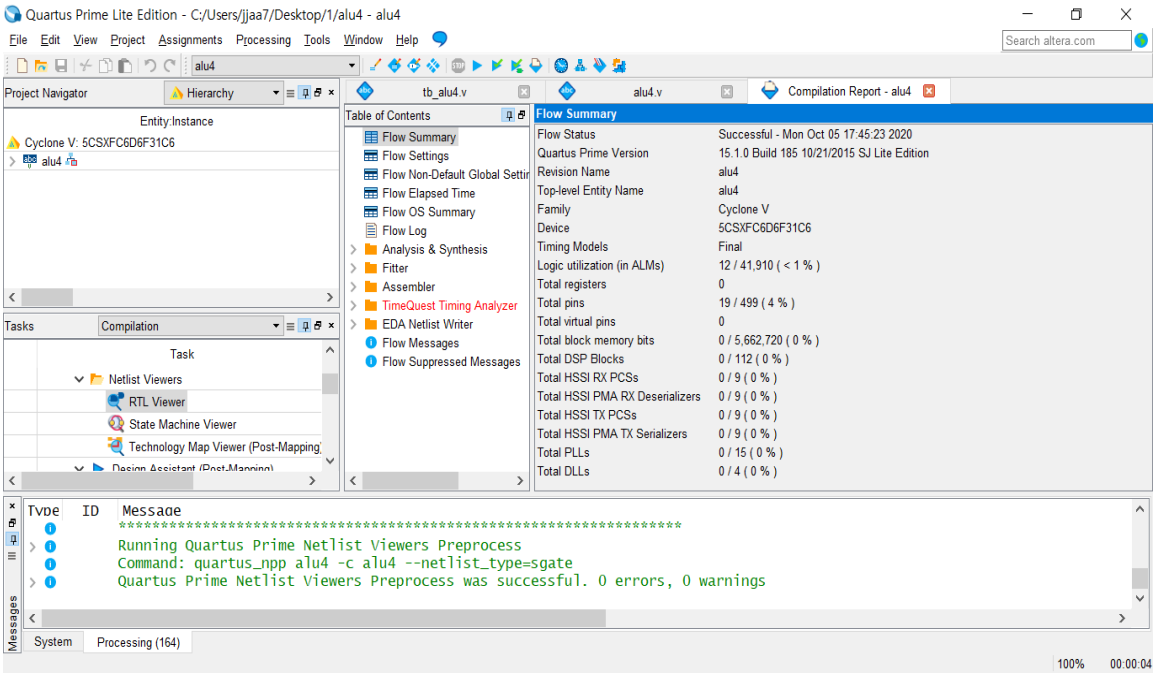
4-bit ALU

- RTL viewer



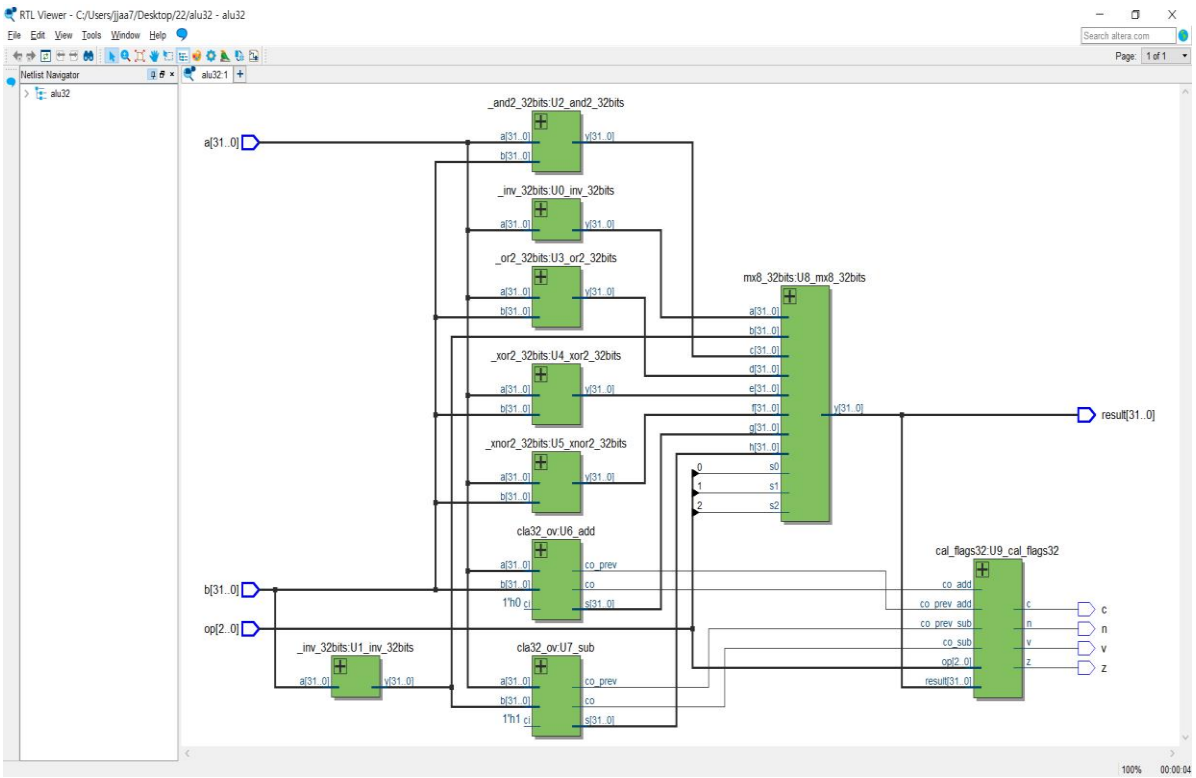
A, B에 숫자를 받고, 어떤 기능을 수행할지 결정해주는 op값을 넣은 후 일단 모든 기능들에 값이 들어간다. 이후 mx8에서 op값에 의해 특정 기능을 수행한 값을 도출해낸데. 또한, ADD와 SUB의 경우에는 conditional register값이 필요하기 때문에 Cal_flags4 모듈에 값을 넣어주고 C, N, Z, V값이 나온 모습이다.

- Flow Summary



32-bit ALU

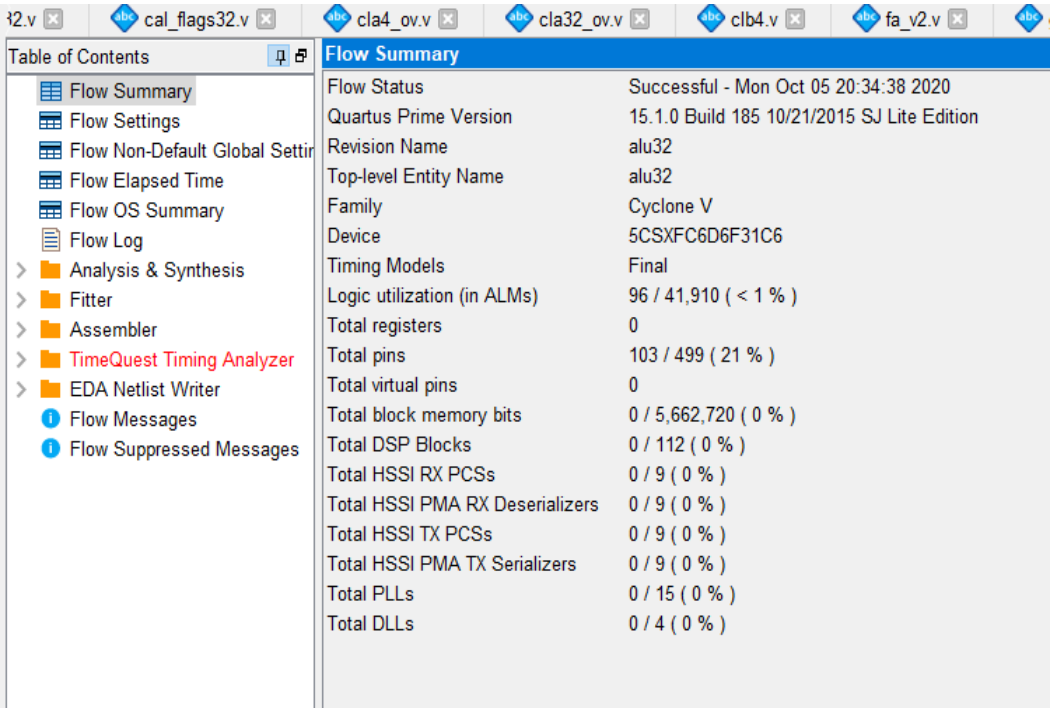
- RTL viewer



32-BIT ALU의 경우 4-BIT ALU와 유사한 함성결과를 가진다.

그러나 BIT수가 늘어남에 따라 각각의 모듈을 열어보면 더욱 많은 logic을 사용함을 알 수 있다. 또한 condition register에서 bit수에 맞게 코드를 수정해줘야 하며, 4bit씩 인자에 맞게 bit를 연결해주어야 한다.

- flow summary



5. 고찰 및 결론

A. 고찰

- 현재의 ALU 구조에서 두 개의 adder가 아닌 한 개의 adder만을 사용하여 addition과 subtraction을 하는 방법: A – B = A + (-B)인 점을 이용하면 된다. 그렇게 되면 2's complement 체계에 의해서 B와 -B의 관계는 B에 NOT을 취한 값에 +1한 값이 -B이므로 인자 값에 a, not b, ci(ci = 1)을 adder에 넣어주면 된다.

- 32-BIT ALU의 경우 SUB에서 제대로 값이 나오지 않았는데, tb를 보니, 값이 정상적으로 계산되다가, 9번째 bit와 5번째 bit에서부터 값이 바뀌어서 나오기 시작하는 것을 확인할 수 있었다. 고치는 것 까진 실패했지만, Timing에서 시간이 꼬인 것으로 생각된다. 왜냐하면 ADD의 경우 정상적으로 나오는 것을 확인할 수 있었고, SUB은 A – B = A + (-B)를 응용해서 인자 값을 바꾸어 넣은 것이기 때문이다. (또, 정상적으로 나오는 SUB도 있었다.)

- 32-BIT ALU에서 testbench를 self-checking할 때 vector를 사용해서 하라고 했으나, 사용하는 방법을 도무지 이해못해 CLA4와 동일한 방법을 사용했다. 그럼에도 불구하고 self-checking에서 error가 나는 부분들이 많았는데, 1번째의 경우는 result에 어떤 값을 넣어도 error로 뜨는 경우였다. 수기로 계산한 결과와는 SUB을 제외하곤 전부 옳게 나왔다. 2번째의 경우는 SUB의 경우 5, 9번째 register에서 정상적인 값으로 계산되다가, 갑자기

그 이후 bit부터 값이 전반 되는 경우이다. 아직 원인을 찾지 못했으나 값이 반전되기 이전으로만 비교한다면 정상적으로 값이 나오기 때문에, CLB나 TIMING에서 문제가 있는 것으로 생각된다.

B. 결론

- ALU는 6개의 logic과 덧셈 뺄셈의 기능을 선택적으로 수행할 수 있게 해주는 module 이다.

- C, N, Z, V는 산술 논리 장치를 검사하는 목적으로 쓰이는 Output이다.

- ALU는 하나의 adder를 사용하여 덧셈, 뺄셈 연산이 모두 가능하다.

6. 참고문헌

이준환/디지털논리회로2/광운대학교(컴퓨터정보공학부)/2020
공영호/디지털논리회로2/광운대학교(컴퓨터정보공학부)/2020