컴퓨터 공학 기초 실험2 보고서

실험제목: Subtractor & Arithmetic Logic Unit

실험일자: 2023년 09월 23일 (금)

제출일자: 2023년 10월 12일 (목)

학 과: 컴퓨터공학과

담당교수: 이형근 교수님

실습분반: 금요일 0, 1, 2

학 번: 2020202037

성 명: 염정호

1. 제목 및 목적

A. 제목

Subtractor & Arithmetic Logic Unit

B. 목적

덧셈, 뺄셈 연산을 수행하는 4bit-Subtractor와 산술 논리연산을 수행하는 4bit Arithmetic Logic Unit (ALU)을 구현 해보고 이를 이용한 32bit ALU를 구현한다. 또한 베릴로그에서 Blocking과 non-Blocking의 차이를 알아본다.

2. 원리(배경지식)

SUB:

ALU: 덧셈, 뺄셈, 곱셈, 나눗셈과 같은 두 숫자 연산과 AND, OR ,NOT, XOR과 같은 논리연산을 수행하는 디지털 회로이다. 이러한 연산을 수행하기 위해 ALU내에서 전가산기, 반가산기와 같은 회로들을 포함하고 있다.

사용자의 요구에 따라 연산을 수행하고, 그 결과를 레지스터에 저장하거나 다음단계로 전달해 복합연산을 수행 할 수 있다. 하지만 연산이 복잡해 질수록 복잡도, 가격, 전력소모면 등에서 실용성이 떨어지게 된다.

FLAG: ALU산술 연산수행 후에 연산 결과에 대한 여러가지 상태 정보를 나타내는 flag를 생성하게 된다. 프로그램 제어, 조건, 분기 및 다른 연산에 이용가능하다.

종류

Zero Flag : 연산 결과가 0일 때 세트된다. 즉 ALU의 출력이 0인경우 Zeor Flag가 1이된다.

Sign Flag : 연산 결과의 최상위 비트(MSB, Most Significant Bit)가 1일 때 세팅 된다. 이를 통해 연산결과가 음수인지 확인할 수 있다.

Overflow FlAG : 연산결과가 표현할 수 있는 범위를 넘어간 경우 세팅 된다.

Carry Flag : 덧셈연산에서 자리 올림이 발생한 경우 세팅된다.

Parity Flag : 연산 결과의 1비트의 개수가 짝수일 때 세팅된다. 1의 개수가 짝수이면 1이 된다.

Carry의 경우 덧셈에서 발생된 자리올림수 즉 8bit 덧셈에서

1101

1001 -> 이 10010이 되는 것 처럼 주어진 8bit를 벗어나지 않는 경우이고

Overflow는 127 + 1처럼 8bit에서 표현할 수 있는 범위(-128~127)을 벗어난 경우에 나타난다.

Blocking : 베릴로그 시뮬레이션을 돌릴 때 순차적으로 할당이 실행되는 것 즉 한 줄의 코드가 끝나기 전에 다음 줄의 코드를 실행시키지 않는다.

"="연산자를 이용해서 표현된다.

한번에 하나의 할당문만 실행함으로 코드블록 내에서 순차적으로 실행된다.

Non-Blocking : 할당이 병렬로 실행된다. 한 줄의 코드가 실행 중일 때 다음 줄의 코드도 실행 될 수 있다.

"<="연산자를 이용해서 표현된다.

Non-Blocking 할당을 사용하면 여러 할당 문이 동시에 실행될 수 있으므로 코드 블록 내에서 병렬적으로 실행된다.

예시)

3. 설계 세부사항

설계한 내용에 대하여 자세히 작성합니다. 설계 세부사항은 구현한 하드웨어를 설계한 방법과 입출력(in/out), 진리표(truth table), 상태천이도(state diagram) 등을 이용하여 설명하고, 해당 하드웨어 동작에 대하여 설명합니다.

(1장에서 2장 사이로 정리하여 작성하며, source code는 첨부하지 말 것)

(입출력, 진리표, 상태천이도 등을 작성할 때, 조교 강의자료에서 그대로 캡쳐하여 삽입하지 말 것)

Opcode	Operation
3b'000	NOT A
3b'001	NOTB
3b'010	And
3b'011	Or
3b'100	Exclusive Or
3b'101	Exclusive Nor
3b'110	Addition
3b'111	Subtrarction

Operator Opcode

A = 0101

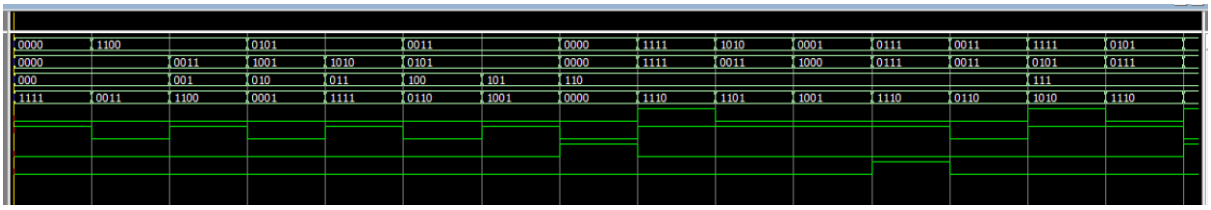
B = 1010

opcode	Y	C	N	Z	V
3b'000	1010	0	1	0	0
3b'001	0101	0	0	0	0
3b'010	0000	0	0	1	0
3b'011	1111	0	1	0	0
3b'100	1111	0	1	0	0
3b'101	0000	0	0	1	0
3b'110	1111	1	1	0	0
3b'111	1011	1	1	0	0

opcode	Y	C	N	Z	V
3b'000	1010	0	1	0	0
3b'001	0101	0	0	0	0
3b'010	0000	0	0	1	0
3b'011	1111	0	1	0	0
3b'100	1111	0	1	0	0
3b'101	0000	0	0	1	0
3b'110	1111	1	1	0	0
3b'111	1011	1	1	0	0

4. 설계 검증 및 실험 결과

A. 시뮬레이션 결과



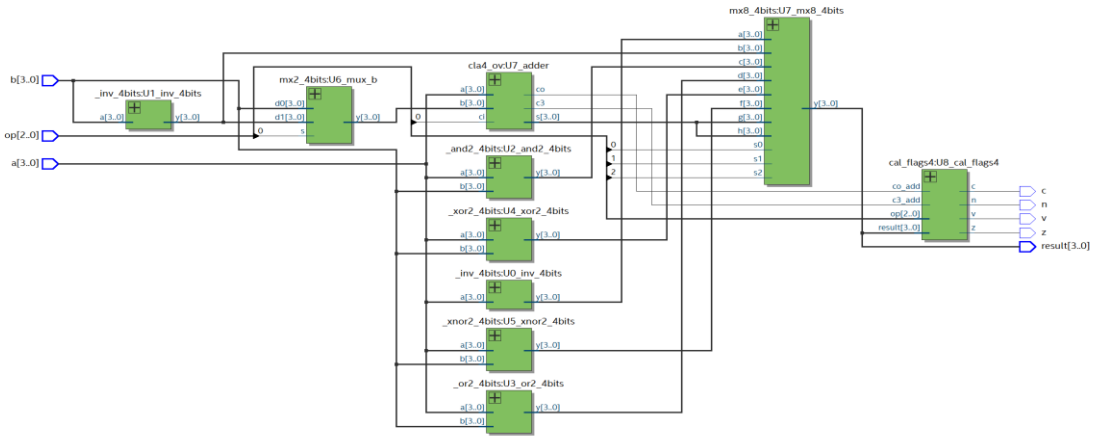
각각의 연산에서 결과 A, B, Opcode를 바꿔가며 연산을 실행하고 결과 값을 확인해 보았다. 논리연산의 경우 논리연산을 진행한 값 그대로 출

력되었으며. 산술 연산의 경우

1111과 1111의 덧셈에선 -1과 -1의 덧셈이기 때문에 오버플로가 발생하지 않고 carry가 발생했다 하지만.

0111 0111의 덧셈의 경우 15+15가 됨으로 4bit에서 표현될 수 있는 숫자 범위를 넘어갔기 때문에 오버플로가 발생한다. 양수끼리의 덧셈에서 음수가 발생한다. 역으로 음수와 음수의 덧셈에서 양수가 발생하는 것 또한 오버플로가 발생한다.

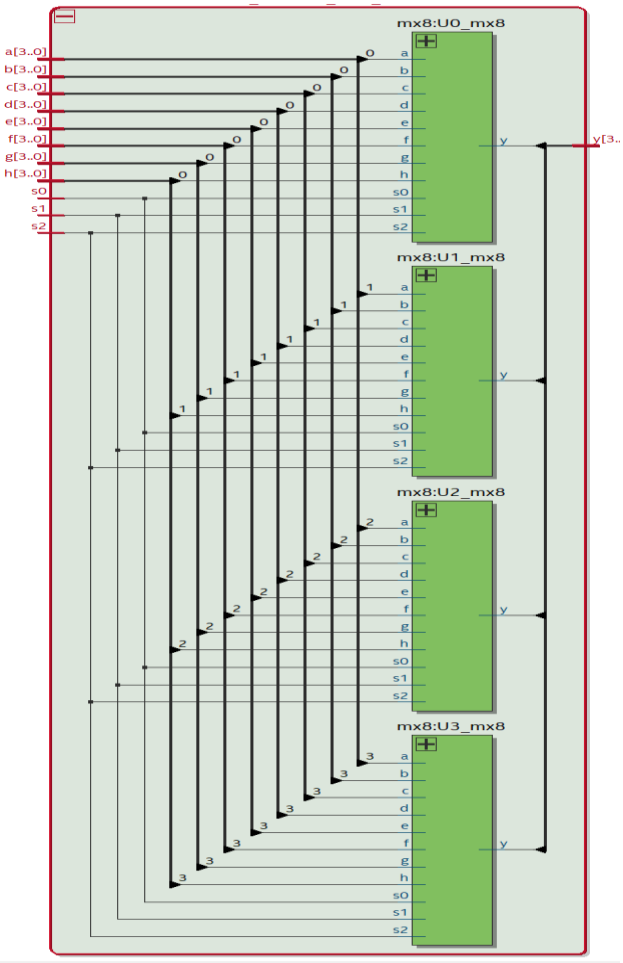
B. 합성(synthesis) 결과



4bit-ALU 구조

각각의 연산 결과 값을 통해 C,N,Z,V와 결과값을 출력한다.

오버플로를 검출하기 위해 4BIT CLA에서 carry out과 carry[3]을 출력할 수 있도록 수정했다.

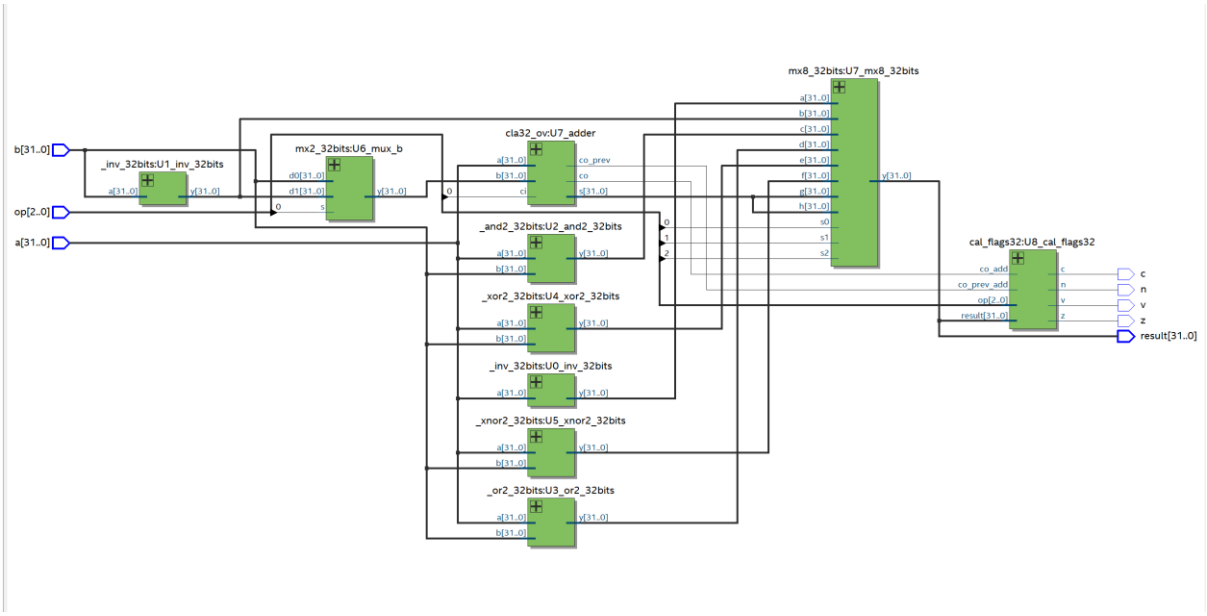


mux8-4bit 구조

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)	10 / 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 %)

사용된 레지스터의 개수: 0 , Pins : 입력a,b (8) + carry(1) + sum(4)+ c3(1)+co(1)

CNZV(4)



32bit ALU 형태는 4BIT ALU와 같으나 각각의 모듈에서의 BIT수가 다르다.

32bit ALU또한 오버플로를 검출하기 위해 32BIT CLA에서 carry out과 carry[31]을 출력할 수 있도록 수정했다.

5. 고찰 및 결론

A. 고찰

4bit ALU와 32bit ALU실험을 진행하였다. 결과 값을 넣고 확인하는 과정에 있어서 CNZV 중 N과 Z부분은 쉽게 결과값 확인이 가능했으나. Carry와 Overflow부분에서 언재 carry가 발생하고 언재 overflow가 발생하는지 확인하기 어려웠다.

확인결과 carry의 경우 4bit alu에서 5bit자리올림 수가 발생 했을 때 carry가 발생함을 확인할 수 있었고, overflow의 경우 양수끼리의 덧셈 혹은 음수끼리의 덧셈에서 부호가 바뀌거나 큰수에서 작은수를 뺐는데 음수가 나오는 경우 등 이상한 결과값이 출력됐을 때 발생한다는 것을 알게되었다.

B. 결론

Alu모듈 하나를 통해 여러가지 산술연산과 논리연산을 수행할 수 있다는 것이 놀라웠다. 각각의 연산을 따로따로 구현하는 것이 아닌 통합해 op코드에 따라 다양한 연산 결과값을 확인 할 수 있었다. 또한 어셈블리 프로그래밍에서도 C,N,Z,V에 따라 다양한 코드를 작성해야 하는 경우가 있었는데, 이번 실험을 통해 그 코드들이 어떻게 생성되는지를 확인 할 수 있었다.

6. 참고문헌

ALU/https://ko.wikipedia.org/wiki/%EC%82%B0%EC%88%A0_%EB%85%BC%EB%A6%AC_%EC%9E%A5%EC%B9%98

이준환/디지털논리회로2/광운대학교/2023

이형근/컴퓨터공학기초실험2/광운대학교/2023