컴퓨터 공학 기초 실험2 보고서

실험제목: Subtraction & arithmetic logic unit (ALU)

실험일자: 2023년 09월 25일 (월)

제출일자: 2023년 10월 05일 (목)

학 과: 컴퓨터공학과

담당교수: 이준환 교수님

실습분반: 월요일 0, 1, 2

학 번: 2022202065

성 명: 박나림

1. 제목 및 목적

A. 제목

Subtraction & arithmetic logic unit (ALU)

B. 목적

ALU의 원리를 공부하고 MUX를 이용하여 설계하는 방법을 이해하도록 한다. flag들에 대한 개념을 이해하고 4-bits ALU를 설계하도록 한다. 이를 이용하여 32-bits ALU를 설계하고 검증해보도록 한다.

2. 원리(배경지식)

A. ALU

덧셈과 뺄셈같은 산술 연산과 함께 NOT A, NOT B, AND, OR, XOR, XNOR등 논리 연산까지 할 수 있는 산술 논리 장치(Arithmetic and Logical Unit)이다. 레지스터에 저장된 데이터를 대상으로 연산이 수행되며, 레지스터 간의 데이터를 주고받게 된다. 뺄셈을 할 때에는 2의 보수를 취한 뒤 더함으로써 가산기로도 뺄셈 연산을 할 수 있도록 설계된다. 연산의 종류에는 곱셈, 나눗셈, 증감, 시프트 등 다양하게 있지만 이번 실습의 ALU는 처음에 나온 총 8개의 연산만을 하는 ALU로, operator인 opcode를 받아서 8-bits MUX로 연산의 종류를 결정하게 된다.

또한 ALU는 특정 조건이 만족되었을 때 상태를 나타내는 flag 정보들을 출력한다. flag는 비교 연산에서 사용될 수 있다. 이번 실습의 ALU에선 총 4개의 flag들로, C(carry), N(negative), Z(zero), V(overflow)가 있다. C는 연산 결과로 carry가 발생하는 경우에 업데이트 되며, N은 연산 결과의 sign bit(최상위 bit)가 1인 경우 음수를 나타낸다는 의미에서 업데이트 된다. Z는 연산 결과 모든 bit가 0인 경우 값도 0이라는 의미로 업데이트 되며, V는 연산 결과 최대 표현 범위를 넘어갈 때 결과 값이 정확하지 않을 수 있다는 의미로 업데이트 된다.

-carry와 overflow의 차이점

캐리는 두 수의 합으로 인해 최상위 비트(MSB)에서 그 다음 비트로 자리올림이 발생하는 것을 말한다. 오버플로우는 최대 비트 수로 표현할 수 있는 범위를 넘어가는 결과 값이 나올 때를 말한다. 주로 양수와 양수를 더했을 때, 음수와 음수를 더했을 때 숫자가 너무 커져버린 경우 부호 값이 다르게 나올 수 있다. 예시로 양수끼리 더

했는데 음수 결과가 나온 경우, ex: 00000001+01111111=10000000 이렇게 되었을 때 오버플로우가 발생했다고 볼 수 있다. 더 정확하게 판단하기 위해서는, MSB를 c2로 설정하고 그 전 비트를 c1이라 하였을 때, 덧셈 연산을 수행하면서 둘의 캐리 값이 같으면 단순히 캐리가 발생한 것이므로 오버플로우가 아니게 되고, 둘의 값이 다르면 그 결과로 부호가 달라지기 때문에 오버플로우 조건을 만족하게 된다. 따라서 V flag를 검사할 땐 c2와 c1을 XOR하여 두 값이 차이가 날 때만 조건을 만족한도록 설정한다. unsigned 연산에선 오버플로우가 발생하지 않지만, signed 연산이 되면 2의 보수 결과의 부호를 검사함으로써 발생할 수 있다. 이러한 연산의 결과가 정확한지 확인하기 위해 V flag가 있는 것이다.

B. blocking & non-blocking

베릴로그에서 값을 할당할 때 두가지 방법이 사용될 수 있는데, '='을 사용하는 blocking 방법과 '<='을 사용하는 non-blocking 방법이 있다. 이러한 문법을 통해 사용자가 값을 할당하는 방법을 컴파일에게 명시적으로 알려주는 것이다. 단, 같은 always문에서 둘을 혼용해서 사용하지 않도록 한다.

-blocking

기본적으로 사용되는 '=' 할당은 동일한 시간내에서 라인 별로 순차적으로 실행이 된다. 앞의 코드가 먼저 처리되기 전까지는 다음 코드가 처리될 수 없다. 실행 자체를 막아버리기 때문에 blocking이라 한다. 예제 코드를 실행해보면, blocking module에서는 순차적으로 실행되기 때문에 b=a; c=b;로 하면 b엔 a값이 들어가고, c에 b값을 넣음으로써 같은 a값이 할당된 걸 볼 수 있다.

-non-blocking

'<=' 할당을 사용하면 동일한 시간내에서 동시에 실행이 된다. clock이 업데이트 되면서 동시에 여러 할당을 처리해야 하는 Sequential logic, Latch등에서 효과적인 방법으로 쓰일 수 있다. 예제 코드처럼 b<=a; c<=b;가 실행된다면, 동시에 처리되기 때문에 b=a가 되지만 c는 아직 b의 값을 모르기 때문에 c=x로 처리가 된다.

3. 설계 세부사항

32-bits ALU를 설계하기 위하여 우선 4-bits CLA에서 flag V를 확인하는 용도로 carry out 뿐만 아니라 carry[3]까지 출력하도록 바꾼다. 32-bits CLA에서도 마지막 8번째 CLA를 앞서 바꾼 CLA로 변경한다. 그리고 4-bits ALU, 32-bits ALU를 설계함과 함께 flag를 계산하는 각각의 module도 설계하여 합치도록 한다.

1) opcode

Opcode	Operation
3'b000	NOT A
3'b001	NOT B
3'b010	AND
3'b011	OR
3'b100	XOR
3'b101	XNOR
3'b110	ADD
3'b111	SUB

2) Flag calculator 32 bits

-32 bits I/O 입출력

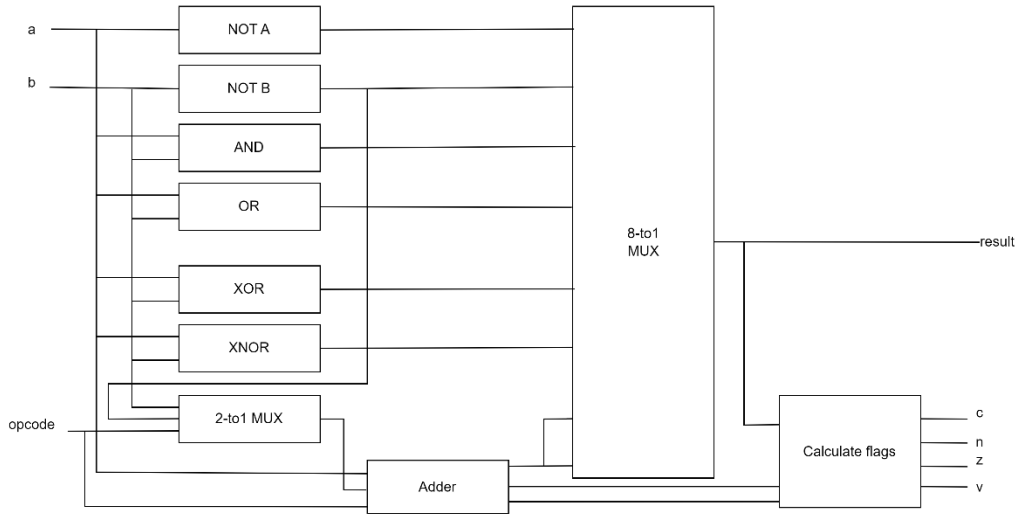
Input	op (3bits, operation code)
	result (32 bits, ALU result)
	co_add (1 bit, carry out)
	c3_add (1 bit, [MSB-1])
output	c (1 bit, carry)
	n (1 bit, negative)
	z (1 bit, zero)
	v (1 bit, overflow)

3) 32-bits ALU

-I/O 입출력

Input	a (32 bits, data)
	b (32 bits, data)
	op (3bits, operation code)
output	result (32 bits, ALU result)
	c (1 bit, carry)
	n (1 bit, negative)
	z (1 bit, zero)
	v (1 bit, overflow)

-회로도



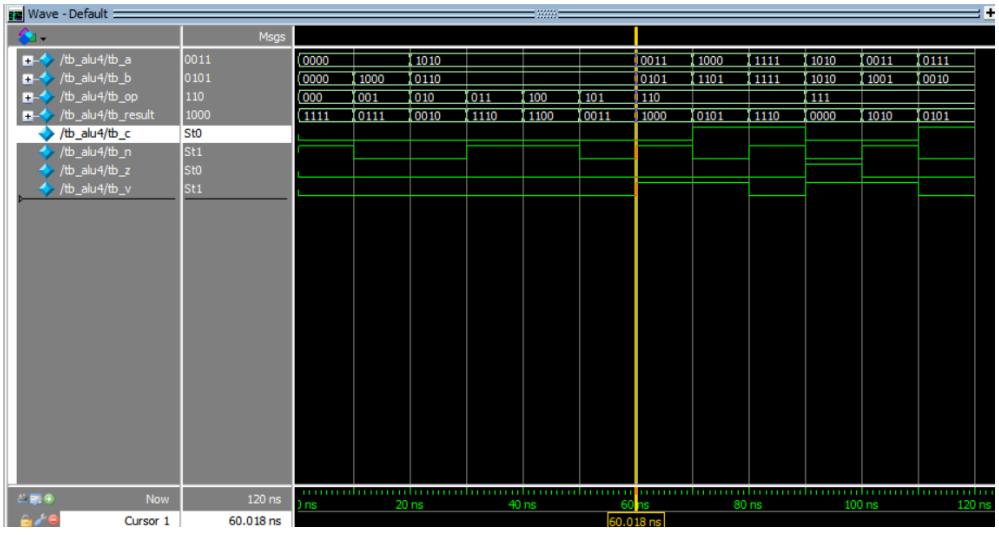
4. 설계 검증 및 실험 결과

A. 시뮬레이션 결과

4-bits ALU와 32-bits ALU를 각각의 테스트벤치로 검증해 본 결과이다. 8개의 연산 종류별로 계산이 되었으며, 최상위 비트가 1로 음수 결과가 나온 경우 N flag, 덧셈이나 뺄셈에서 carry가 발생한 경우 C flag, 결과가 전부 0이 나온 경우 Z flag, 표현 범위를 벗어난 경우 V flag가 뜬 걸 확인할 수 있다.

1) 4-bits ALU

-binary



2) 32-bits ALU

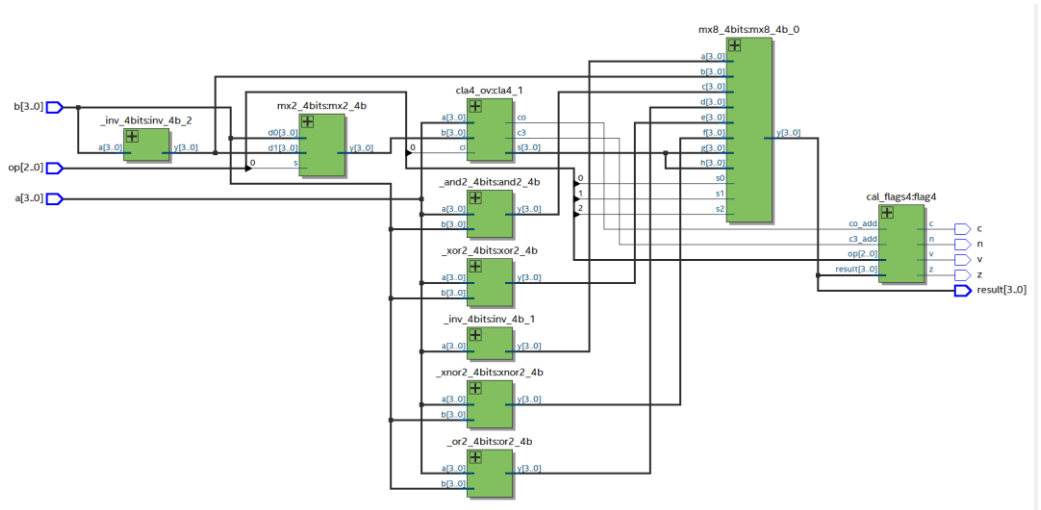
-hexadecimal

B. 합성(synthesis) 결과

1) 4-bits ALU

4-bits ALU를 설계했을 때의 합성 결과로, RTL viewer를 통해 회로도를 확인할 수 있다. 또한 Flow summary를 보면 회로의 크기가 10인걸 알 수 있다.

-RTL viewer



-Flow summary

Flow Summary

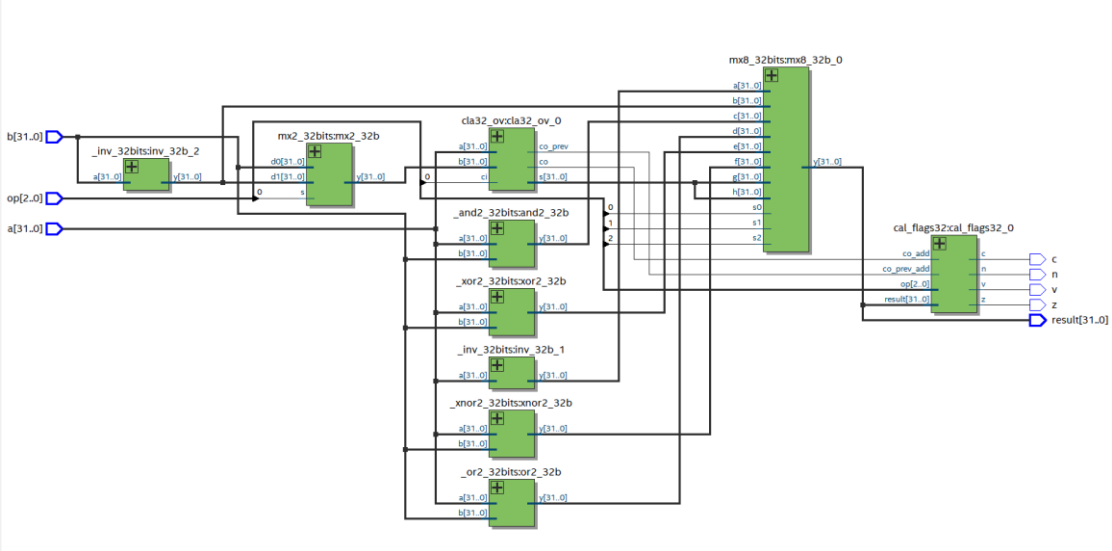
<<Filter>>

Flow Status	Successful - Mon Sep 25 14:03:10 2023
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 %)

2) 32-bits ALU

32-bits ALU를 설계했을 때의 합성 결과로, RTL viewer를 통해 회로도를 확인할 수 있다. 또한 Flow summary를 보면 회로의 크기가 4-bits ALU보다 더 큰 89인걸 알 수 있다.

-RTL viewer



-Flow summary

Flow Summary

<<Filter>>

Flow Status	Successful - Tue Sep 26 13:49:12 2023
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)	89 / 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 %)

5. 고찰 및 결론

A. 고찰

ALU를 검증할 때, 기본적인 테스트벤치를 이용하여 웨이브폼으로 검증하는 데엔 성공했으나 디지털논리회로에서 배운 'using self-checking testbench with testvectors'기법으로 하지는 못하였다. test.tv파일을 생성하여 tb_a, tb_b, tb_op, tb_resEx로 input과 예상 output값을 넣은 다음 테스트벤치 코드에서 파일을 읽어와 벡터 배열에 저장하는 방법으로 구현하였는데, 파일은 읽어오는 것 같지만 그걸 저장하는 데에 실패한 것 같다. 웨이브폼으로 결과를 확인해봐도 값들이 전부 x로 나왔었기 때문이다. 이 부분은 더 공부가 필요한 것으로 보인다.

B. 결론

ALU의 원리를 이해하며 어떠한 방식으로 작동되는지 알 수 있었다. op코드를 받아서 연산을 분류별로 나누고, 뺄셈 또한 덧셈기로 구현한다는 점, 지금까지 설계했던 MUX나 CLA가 필요하다는 점 등 직접 실습을 해보면서 더 잘 이해할 수 있었다. 나중에 더 큰 설계를 할 때 이러한 ALU도 같이 필요할 것 같다는 생각이 들었다.

또한 blocking과 nonblocking에 대해 새롭게 알게되었는데, '='와 '<='로 연산자에 따라 할당하는 순서가 달라진다는 점을 자세히 깨달을 수 있었다.

6. 참고문헌

David Money Harris and Sarah L. Harris / Digital Design and Computer Architecture / Elsevier / 2007