컴퓨터 공학 기초 실험2 보고서

실험제목: ALU

실험일자: 2021년 09월 4일 (월)

제출일자: 2021년 10월 10일 (일)

학 과: 컴퓨터정보공학부

담당교수: 공진흥 교수님

실습분반: 월요일 0, 1, 2

학 번: 2018202046

성 명: 이준휘

1. 제목 및 목적
   1. 제목

ALU

* 1. 목적

해당 수업을 통해 ALU가 어떻게 구성되어 있는지 이해한다. 그리고 ALU로부터 나온 C, N, Z, V의 신호가 어떠한 값을 의미하는지 알 수 있다. 이를 self-checking testbench with testvector를 이용하여 검증할 수 있다.

1. 원리(배경지식)
   * 1. Subtracter

Subtracter란 감산기를 의미한다. 해당 회로를 위하여 다른 회로를 구성하지 않고 기존의 가산기를 이용하여 이를 구현할 수 있다. 빼고자 하는 수의 보수 NOT Gate를 이용하여 만든 후 이 수를 다른 수와 더한다. 이때 Carry in 자리에 올림수의 신호를 1로 하여 빼고자 하는 수의 2의 보수를 만들 수 있다. 이로써 하나의 수와 2의 보수를 취한 수를 더하는 것이 감산기의 원리다.

C, N, Z, V

해당 신호는 ALU 상에서 state를 사용해야 할 경우, 특히 ARM과 같은 경우에는 해당 신호를 이용하여 conditional execution을 진행할 수 있도록 돕는 역할을 한다.

C는 Carry를 의미한다. 해당 신호는 마지막 자리에서 Carry가 발생하였는지를 확인하여 출력한다. C = Carry Out이라고 볼 수 있다. 단 해당 신호는 감산기나 가산기에서만 나오도록 설계된다.

N은 Negative를 의미한다. 해당 신호는 마지막 자리의 값이 1인지를 확인하여 이를 출력한다. 즉 N = Result[31]이라고 볼 수 있다.

Z는 Zero를 의미한다. 해당 신호는 모든 자리의 결과가 0인지를 판단하여 이를 출력한다. 즉 Result의 모든 신호를 NOR Gate로 연결한 모습을 취한다. 해당 신호는 A와 B의 값을 빼서 두 값이 같은지를 판단하는 등에 유용하게 사용된다.

V는 Overflow를 의미한다. Overflow는 기존의 값을 연산하였을 때 기존의 표시할 수 있던 숫자의 범위를 초과하여 값이 제대로 표현되지 않는 것을 의미한다. 이러한 상황은 같은 sign을 가진 두 수를 더하거나, 다른 sign을 가진 두 수를 빼는 과정에서만 일어난다. 빼는 과정은 가산기를 이용하여 연산됨으로 우리는 같은 sign을 가진 두 수를 더하는 과정만 이해하면 된다.

만약 양수를 더하는 경우에는 두 수의 [31]은 0으로 생각하면 된다. 이 때 result[31]은 1이 나와야 한다. 이를 정리하면 Carry[31]은 1이 나와야 하고 Carry out은 0이 나와야 한다.

만약 음수를 더하는 경우에는 두 수의 [31]은 1이고 result[31]은 0이 나와야 한다. 이를 정리하면 Carry[31]은 0이 나와야 하고, Carry out은 1나와야 한다.

위의 두 식을 하나로 묶을 경우 우리는 V = Carry out Carry[31]로 이를 정의할 수 있다.

ALU

ALU란 다양한 연산하는 모듈을 의미한다. ALU에는 두 신호, A와 B를 NOT, AND, OR, XOR, XNOR과 같은 부울 연산을 통하여 값을 출력할 수 있다. 그리고 가산기와 감산기를 통하여 이를 값으로 표현할 수 있다. 이러한 신호들을 op을 통해 선별하여 출력할 수 있고, CNZV의 값으로 해당 상태를 표현할 수 있다.

1. 설계 세부사항
   * 1. gates

해당 파일에는 4bit와 32bit를 invert할 수 있는 모듈, 4bit와 32bit를 AND, OR, XOR, XNOR 연산을 할 수 있는 모듈들을 각각 구성하였다.

* + 1. cla4\_ov

해당 모듈은 기존의 cla4모듈을 바탕으로 제작되었으며 출력값으로 c3를 추가한 것만이 달라졌다.

* + 1. cla32\_ov
    2. 해당 모듈은 기존의 cla32모듈을 바탕으로 제작되었으며 마지막 4비트의 모듈만 cla4에서 cla4\_ov로 변경하여 co\_prev를 출력한다.
    3. mx2\_4bits

해당 모듈은 mx2 모듈을 활용하여 제작되었다. 각각의 자리의 비트를 mx2로 연결하여, 총 4개의 mx2 모듈을 이용하여 y값 4bit를 출력한다.

* + 1. mx2\_32bits

해당 모듈은 삼항 연산자를 통해 제작되었다.

* + 1. mx8

해당 모듈 또한 mx2 모듈을 활용하여 제작되었다. 해당 모듈은 a와 b, c와 d, e와 f, g와 h 신호를 s0를 통해 선택하고, 선택된 신호 4개를 2개씩 짝지어서 s1을 통해 선택한다. 마지막으로 나온 2개의 신호를 s2를 이용하여 선택하면 8-to-1 MUX를 구성할 수 있다.

* + 1. mx8\_4bit

해당 모듈은 mx8 모듈을 활용하여 제작되었다. 해당 모듈은 mx2\_4bit와 같이 4개의 mx8 모듈을 사용하여 각각의 비트에 연결하여 구성하였다.

mx8\_32bit

해당 모듈은 mx2\_32bits를 이용하여 제작되었다. 해당 회로는 mx8과 같이 7개의 mx8\_32 중 4개는 s0, 2개는 s1, 1개는 s2에 연결하여 제작되었다.

* + 1. cla\_flags4 & cla\_flags32

해당 모듈은 C, N, Z, V를 출력하는 모듈들이다. C값으로 가산 또는 감산이 진행될 때만 co\_add를 출력하고 이외에는 0을 출력한다. n값은 result의 가장 마지막 위치를 출력하며, z는 result의 값이 모두 0인지 판단하여 출력한다. 마지막으로 v는 가산이나 감산 상황에서만 진행되며 co\_add와 이전 자리의 케리를 XOR 연산한 값을 출력한다.

* + 1. alu4 & alu32

해당 모듈들은 각각의 비트에 맞는 a와 b의 NOT, AND, OR, XOR, XNOR 연산을 위에서 구성한 모듈들을 통해 연결한다. 그 중 NOT Gate를 거친 b 신호는 일반 b 신호와 MUX를 통해 s0로 구분되어 cla에 연결된다. cla는 a와 MUX를 거친 b신호, 그리고 s0를 자리수로 입력받고, 연산 결과를 출력한다. 이러한 연산 결과는 8-to-1 MUX를 통해 3bit의 수로 구별되어 출력된다. 이렇게 출력되는 값의 상태를 확인하기 위하여 cla\_flag와 이 신호를 연결한다.

1. 설계 검증 및 실험 결과
   1. 시뮬레이션 결과

|  |
| --- |
|  |
|  |
| 해당 testbench는 alu4를 검증하는 것이다. 처음 두 값인 0000/0000, 0000/1111을 a를 NOT, b를 NOT으로 연결하여 invert가 정상적으로 되는지 확인하고 n과 z가 정상적으로 작동하는지 확인한다. 3번째부터 6번째의 값은 0001/0011의 신호에 AND, OR, XOR, XNOR의 연산이 정상적으로 되는지 확인하였다. 7번째 값은 일반적인 양의 정수의 덧셈이 잘 되는지 확인하였고, 8번째에는 음의 정수가 정상적으로 더해지고 C가 발생하는지 확인하였으며, 9번째에서는 overflow가 일어나는 값을 더하여 v신호가 정상적으로 동작하는지 확인하였다. 마지막으로는 감산기가 정상적으로 출력되는지 확인하였다. 위 10개의 신호는 example.tv를 통해 입력하는 방법으로 self-checking testbench with testvectors를 이용하였다. 해당 코드에서 error는 0으로 나왔다. |
|  |
|  |
| 해당 testbench는 alu32를 검증하는 것이다. 처음 두 값인 00000000/00000000, 00000000/ffffffff을 a를 NOT, b를 NOT으로 연결하여 invert가 정상적으로 되는지 확인하고 n과 z가 정상적으로 작동하는지 확인한다. 3번째부터 6번째의 값은 00000001/00000003의 신호에 AND, OR, XOR, XNOR의 연산이 정상적으로 되는지 확인하였다. 7번째 값은 일반적인 양의 정수의 덧셈이 잘 되는지 확인하였고, 8번째에는 음의 정수가 정상적으로 더해지고 C가 발생하는지 확인하였으며, 9번째에서는 overflow가 일어나는 값을 더하여 v신호가 정상적으로 동작하는지 확인하였다. 마지막으로는 감산기가 정상적으로 출력되는지 확인하였다. 위 10개의 신호는 example.tv를 통해 입력하는 방법으로 self-checking testbench with testvectors를 이용하였다. 해당 코드에서 error는 0으로 나왔다. |

* 1. 합성(synthesis) 결과

|  |  |
| --- | --- |
| alu4 | |
|  |  |
|  |  |
| alu32 | |
|  |  |
|  |  |
| rtl for bocking and nonblocking | |
|  | |
|  |  |

해당 모듈의 설계를 확인하면 각각의 모듈이 정상적으로 설계되어 연결되었음을 알 수 있다.

* 1. FPGA board targeting 결과

1. 고찰 및 결론
   1. 고찰

이번 주차의 과제는 기존의 과제를 활용하면서 더욱더 응용을 할 수 있었던 시간이었다. 해당 과제를 진행하면서 testbench에서 어려움을 느꼈다. 특히 파일을 읽어올 때 파일이 읽히지 않는 문제가 있었다. 해당 문제는 내가 저장해놓은 tv 파일의 위치를 지정해주지 않아서 생긴 문제였다. ./../../을 통해 위치를 시뮬레이션 안 폴더에서 밖을 지정해줌으로써 이를 해결하였다.

* 1. 결론

1. 해당 실험을 통해 alu가 어떻게 구성되어 있는지 알 수 있다. 또한 c, n, z, v 신호에 대해 명확히 이해할 수 있는 시간이었다. 추가적으로 non-Blocking과 Blocking의 차이는 blocking의 경우, a = b, b = c와 같이 연결할 경우 a가 b와 c에 연결되는 모습을 보이지만, non-blocking의 경우, a <= b, b <= c와 같이 연결할 경우 a에 b가 연결되고 b에 c가 연결되는 모습을 볼 수 있다.
2. 참고문헌

이준환 교수님/디지털논리회로1/광운대학교(컴퓨터정보공학부)/2021

이준환 교수님/디지털논리회로2/광운대학교(컴퓨터정보공학부)/2021