LAB 1. ALU

201800038 박형규, 20180480 성창환

1. Introduction

본 실험에서는 산술 및 논리연산기 (Arithmetic Logic Unit, ALU)의 기능을 Verlilog HDL을 통하여 구현해보며 Verlilog의 기본적인 사용 방법, 문법, 체계 등을 익힘을 목표로 한다. Verlilog의 always문, event 등을 사용하여 논리적인 구조를 만들고 이를 통해 16가지의 연산을 수행하는 ALU를 직접 implement 할 것이다. 이번에 구현할 ALU는 input으로 16-bit signed binary의 A, B input, 그리고 4-bit binary의 FuncCode를 input으로 받고, Output으로 FuncCode에 따른 계산 결과인 C와 덧셈 연산과 뺄셈 연산의 경우 OverflowFlag를 출력하도록 한다.

2. Design

ALU를 디자인하는데 있어서 가장 먼저 initial code로 FuncCode에 따라 이벤트를 발생시켰다. ALU가 수행할 수 있는 16가지 연산에 각각 이벤트 변수를 부여하여 입력받은 FuncCode에 따라 해당되는 이벤트로 이동하도록 하였다. 각 이벤트로 이동하여 always @... begin end 문에 따라 해당되는 연산을 수행하였다. 또한 덧셈과 뺄셈 연산의 경우에는 Carry in과 Carry out에 따라 Overflow가 발생하는지를 구하였고, Overflow가 발생하지 않았을 경우 OverflowFlag가 0을, 발생하였을 경우 1을 나타내도록 하였다.

3. Implementation

case문을 이용해서 FuncCode의 값을 분석해 전달된 연산을 event를 이용해 실행하도록 구현했다.

덧셈과 뺄셈을 제외한 나머지 14가지 FuncCode들은 Varilog의 operator들을 단순히 이용해서 구현을 하였다. 추가적으로 Arithmetic right shift를 구현하기위해 새롭게 16bit signed reg를 만들어서 A를 복사한 다음에 >>> 연산자를 사용해서 구현했다.

덧셈과 뺄셈의 경우 똑같이 operator들을 사용하되, A, B와 C(A+B, A-B)의 MSB를 이용해서 OverflowFlag의 값을 결정했다. 오버플로우 확인은 디시설시간에 배웠던 카노맵을 이용한 방법으로 구한 수식을 적용했다.

z = a+b: (!x & !y & !(!z)) | (!(!x) & !(!y) & !z)이면 overflow

z = a-b: (!x & !(!y) & !(!z)) | (!(!x) & !y & !z)이면 overflow

4. Discussion

Case문을 호출하는데 있어서 인터넷의 많은 예시들이 always @(*) case begin의 형식을 사용해서 이용했는데 이렇게 구현하니 결과값이 매우 이상하게 나왔다. @(*)가 아래에서 구현한 a1 ~ a16까지의 event들을 표함하는 표현이기 때문에 그렇게된다고 판단해서 @()안에 input들만 넣었더니 정상적으로 실행되었다.

Always case begin이 왜 이상한지 이해가 잘 되지 않는다.

5. Conclusion

산술연산기 ALU를 구현하며 Verilog에서 Logic의 흐름이 어떤 방식으로 이루어지는지, case문과 if문을 어떻게 사용하는지 등과 같은 Verilog에서의 기본 Syntax에 대하여 알게 되었다. 또한 ALU를 implement하는데 있어서 더하기와 빼기 연산에서는 각 input의 MSB를 통해 karnough map을 이용하여 overflow를 잡는 방법에 대해 복습하였고, Arithmetic Right Shift의 경우에는 signed register을 새로 생성하여 signed의 부호의 중요성에 대하여 학습하였다. 이에 따라 42개의 모든 Test Case에 대하여 Succed하였다.