Report: Lab1

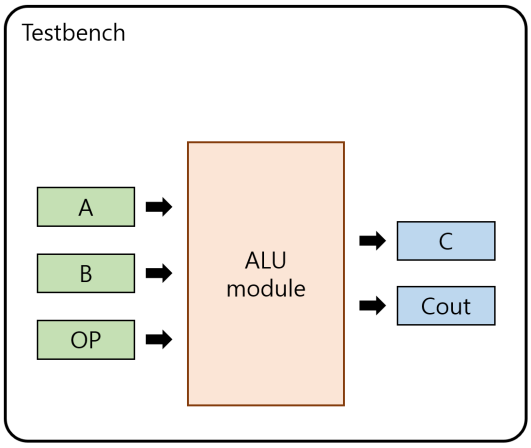
20190533 Hyemin Lee
20190235 Gangtae Park

1. Introduction

Lab 1은 verilog를 이용해서 다양한 연산을 할 수 있는 Arithmetic Logic Unit (ALU)를 제작하는 랩이다. 덧셈과 뺄셈을 비롯해 and, or, nand 등의 gate들과 logical/arithmetic shift 등의 operation들을 구현하였다. Lab 1을 수행하기 위해서는 기초적인 verilog 문법들과 ModelSim을 통해 컴파일하는 방법을 알아야 한다. 또한 overflow를 구현하는 과정에서 truth table과 K-map을 이용해 복잡하지 않게 코드를 작성하는 능력 또한 필요하다. 우리는 각 operation들을 case로 나누어서 각각마다 output을 설정하는 방법으로 ALU를 구현하였다.

2. Design

우리는 하나의 모듈로 ALU를 구현했다. ALU 모듈은 A와 B와 OP(operation)을 입력 받으며, C와 Cout을 출력한다. OP는 16가지의 연산 종류를 나타내며, A와 B는 연산에 사용되는 피연산자(operand)이다. 본 실험에서 구현한 전체 디자인은 아래와 같다.



3. Implementation

ALU 모듈은 호출이 되는 즉시 always 구문을 실행하며, overflow를 나타내는 Cout에 기본값인 0을 대입한다. 그리고 OP값을 확인하여 어떤 연산을 해야하는지 case statement를 이용해서 구분한다. case statement 안에는 각 OP값에 맞는 연산이 구현돼있으며, 규칙에 맞게 C값을 업데이트한다. 이때 OP:0000과 OP:0001인 덧셈과 뺄셈의 경우 overflow를 고려해서 Cout 또한 업데이트해야 한다. 덧셈의 경우, overflow가 발생하는 경우는 input인 A와 B의 msb가 같고 C의 msb가 다른 경우이다. 이 조건을 truth table로 나타내면 다음과 같다.

A	B	C	Cout

0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	0

다음 truth table을 이용해 K-map을 그려보면 다음과 같다.

A\BC	00	01	11	10
0	0	1	0	0
1	0	0	0	1

이를 통해 Cout = A'B'C+ABC' 이라는 결론을 내렸고, verilog를 통해 쉽게 구현 할 수 있었다. 뺄셈의 경우, 임시 레지스터(pB)에 -B 값을 저장한 후에 A+pB라고 가정하고 overflow를 계산하였다. Case statement가 끝나면 ALU 모듈도 종료되며, 최종적인 C와 Cout 값을 반환한다.

4. Evaluation

ModelSim을 통해서 skeleton code로 주어진 ALU_TB.v의 testbench를 컴파일 및 시뮬레이션하기를 반복하여 alu.v가 정확히 구현되었는지 확인하였다. 최종 alu.v는 ALU_TB.v의 모든 testcase(50개)를 통과하였다.

5. Discussion

여러 operation 중 Arithmetic Right Shift를 구현함에 있어서 문제가 있었다.

```
C <= A >>> 1;
```

위의 코드 처럼 verilog에 내장된 Arithmetic Right Shift를 이용해 구현하려고 했으나 test case를 통과하지 못해서 개념에 맞게 직접 구현하여 다음과 같이 표현하였다.

```
C <= {A[15], A[15:1]};
```

Arithmetic Left Shift를 구현할 때는 문제가 없었으나 코드의 통일성을 위해서 같은 방법으로 표현했다.

6. Conclusion

본 실험은 verilog를 이용해서 ALU를 구현하는 실험이었다. 본 실험을 통해 verilog의 기본적인 문법뿐만 아니라 16가지 연산을 verilog로 구현하는 방법을 복습할 수 있었다. 또한 arthmetic right/left shift는 정의를 토대로 직접 구현해보는 경험도 할 수 있었다.