EE312 Lab Report – Lab 1. ALU

20160030 고은석, 20160680 추헌호

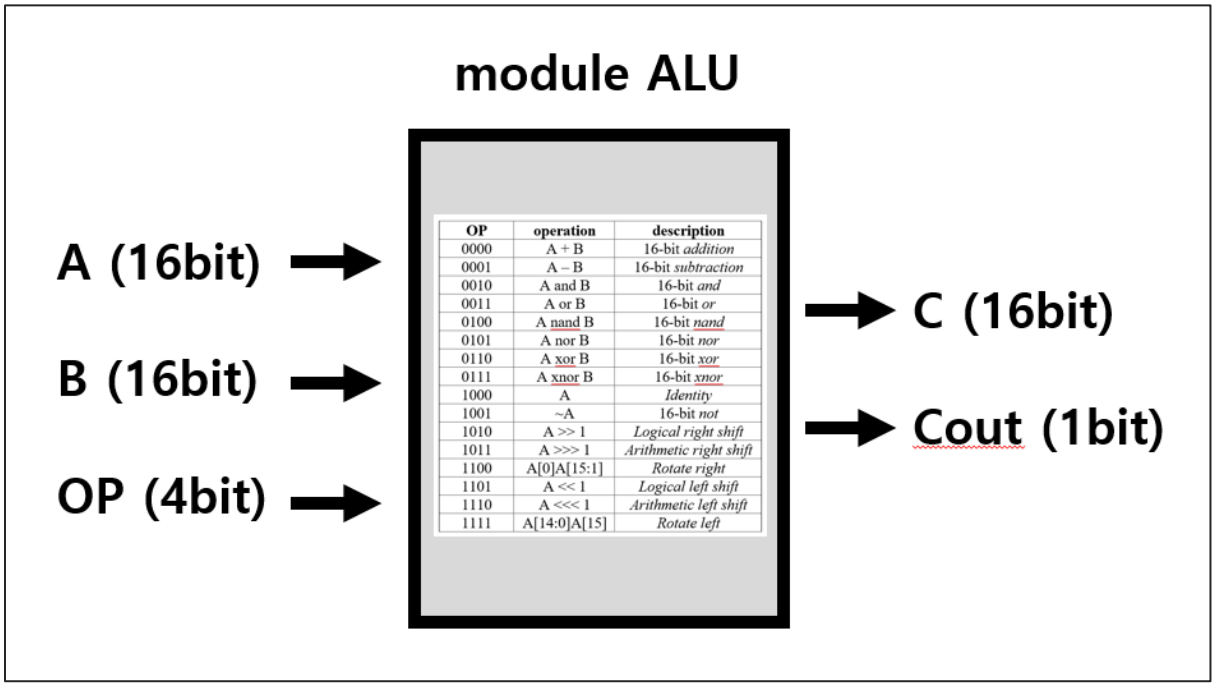
1. Introduction

본 과제에서는 Verilog HDL을 통해 ALU(Arithmetic Logic Unit)을 구현한다. 총 16가지 operation이 포함되어 있으며 각 operation은 두 개 혹은 한 개의 16bit input과 operation의 종류를 알려주는 4bit 신호를 입력받아 연산 결과를 16bit의 output으로 출력한다.

2. Design

ALU의 구현에는 하나의 모듈만이 사용된다. module ALU는 16bit operand인 A, B와 operator의 종류를 알려주는 4bit 신호 OP를 input으로 받아 16bit 연산 결과인 C와 overflow 여부를 알려주는 1bit 신호 Cout을 output으로 출력한다. 이 때 arithmetic operation과 bitwise logic operation에서는 A, B가 모두 사용되지만 shift operation에서는 A만 사용된다.

모듈 내에서는 if문을 통해 어떤 OP 신호가 주어졌는지 확인하고 그에 맞는 연산이 수행된 후 결과값이 C와 Cout에 저장된다.



OP	operation	description
0000	A + B	16-bit *addition*
0001	A – B	16-bit *subtraction*
0010	A and B	16-bit *and*
0011	A or B	16-bit *or*
0100	A nand B	16-bit *nand*
0101	A nor B	16-bit *nor*
0110	A xor B	16-bit *xor*
0111	A xnor B	16-bit *xnor*
1000	A	*Identity*
1001	~A	16-bit *not*
1010	A >> 1	*Logical right shift*
1011	A >>> 1	*Arithmetic right shift*
1100	A[0]A[15:1]	*Rotate right*
1101	A << 1	*Logical left shift*
1110	A <<< 1	*Arithmetic left shift*
1111	A[14:0]A[15]	*Rotate left*

3. Implementation

1) Arithmetic Operations

덧셈과 뺄셈은 단순하게 Verilog HDL에 내장된 +, - 연산자를 통해 구현했다. 단 이 때 A와 B는 signed value이기 때문에 overflow를 고려해 주어야 한다. overflow가 발생했는지의 여부는 두 단계를 통해 확인할 수 있다. 덧셈의 경우, 먼저 A와 B의 MSB를 비교하여 둘의 부호가 같은지 체크한다. 만약 같다면 다음으로 연산 결과인 C의 MSB와 A, B의 MSB를 비교하여 부호가 다른지 체크한다. 이 두 단계가 모두 참이라면 overflow가 발생한 것이다. 뺄셈의 경우에는 반대로 A, B의 부호가 다르고, B의 부호와 C의 부호가 같을 때 overflow가 발생한 것으로 처리한다. overflow가 발생했다면 Cout을 1로 설정하고 아니라면 0으로 설정한다.

2) Bitwise Logic Operations

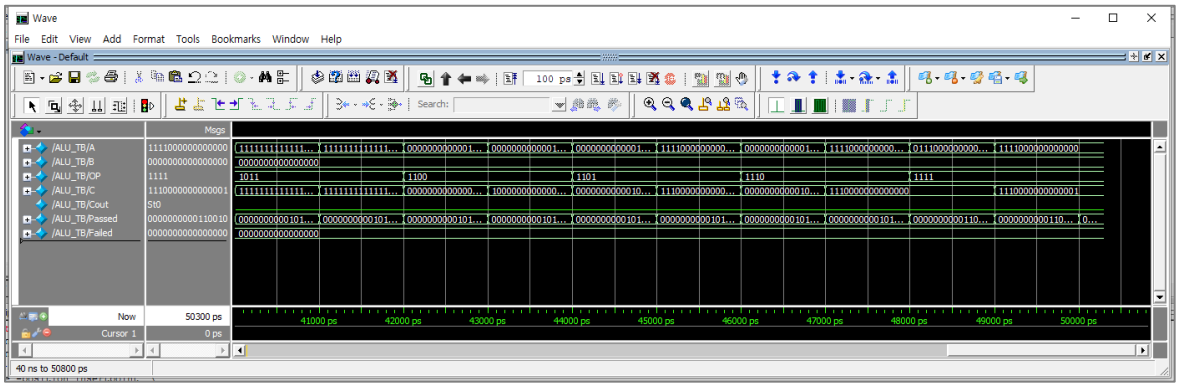
Bitwise 논리 연산에서는 overflow를 고려할 필요가 없기 때문에 Cout은 항상 0으로 설정했고, and, or, nand, nor, xor, xnor은 Verilog HDL에 내장된 연산자인 &, |, ~, ^을 조합하여 구현했다.

3) Shift Operations

마찬가지로 Verilog HDL에 내장된 >>, >>>, <<, <<< 연산자를 이용해 구현했다. 이 때 >>, <<은 각각 right, left 방향의 logical shift로 operand의 부호를 고려하지 않는다. >>>, <<<은 right, left 방향의 arithmetic shift로 operand의 부호를 고려한다. Arithmetic right shift의 경우 A의 MSB가 1이었다면 C의 MSB가 0으로 출력되기 때문에, 그것을 다시 1로 설정해야 한다. Rotate의 경우 1bit logical shift를 적용한 후 A의 LSB를 C의 MSB에 대입하거나(right) 반대로 A의 MSB를 C의 LSB에 대입함으로써(left) 구현한다. Overflow는 고려할 필요가 없기 때문에 Cout은 항상 0으로 설정한다.

4. Evaluation

작성한 코드의 평가는 Operation Module이 작성되어 있는 alu.v 파일을 포함하는 testbench 파일인 ALU_TB.v 파일을 ModelSim 프로그램을 통해서 simulate 하는 방법으로 진행한다.




```
VSIM 3> run -all
# TEST              Add-1 :
# PASSED
# TEST              Add-2 :
# PASSED
# TEST              Add-3 :
# PASSED
# TEST              Add-4 :
# PASSED
# TEST              Add-5 :
# PASSED
# TEST              Add-6 :
# PASSED
# TEST              Add-7 :
# PASSED
# TEST              Add-8 :
# PASSED
# TEST              Add-9 :
# PASSED
# TEST              Sub-1 :
# PASSED
# TEST              Sub-2 :
# PASSED
# TEST              Sub-3 :
# PASSED
# TEST              Sub-4 :
# PASSED
# TEST              Sub-5 :
# PASSED
# TEST              Sub-6 :
# PASSED
# TEST              Sub-7 :
# PASSED
# TEST              And-1 :
# PASSED
# TEST              And-2 :
# PASSED
# TEST               Or-1 :
# PASSED
# TEST               Or-2 :
# PASSED
# TEST             Nand-1 :
# PASSED
# TEST             Nand-2 :
# PASSED
# TEST              Nor-1 :
# PASSED
# TEST              Nor-2 :
# PASSED
# TEST              Xor-1 :
# PASSED
# TEST              Xor-2 :
# PASSED
# TEST             Xnor-1 :
# PASSED
# TEST             Xnor-2 :
# PASSED
# TEST               Id-1 :
# PASSED
# TEST               Id-2 :
# PASSED
# TEST               Id-3 :
# PASSED
# TEST              Not-1 :
# PASSED
# TEST              Not-2 :
# PASSED
# TEST              Not-3 :
# PASSED
# TEST              Lrs-1 :
# PASSED
# TEST              Lrs-2 :
# PASSED
# TEST              Lrs-3 :
# PASSED
# TEST              Lrs-4 :
# PASSED
# TEST              Ars-1 :
# PASSED
# TEST              Ars-2 :
# PASSED
# TEST              Ars-3 :
# PASSED
# TEST              Ars-4 :
# PASSED
# TEST               Rr-1 :
# PASSED
# TEST               Rr-2 :
# PASSED
# TEST              Lls-1 :
# PASSED
# TEST              Lls-2 :
# PASSED
# TEST              Als-1 :
# PASSED
# TEST              Als-2 :
# PASSED
# TEST               Rl-1 :
# PASSED
# TEST               Rl-2 :
# PASSED
# Passed = 50, Failed = 0
# ** Note: $finish    : C:/verilogProject/ALU_TB.v(55)
#    Time: 50 ns  Iteration: 0  Instance: /ALU_TB
```

Simulation 결과는 위와 같았고 ModelSim 프로그램의 transcript에서 볼 수 있듯이 총 50개의 testcase 중 Passed가 50개, Failed가 0개, 즉 모든 ALU_TB.v 파일 내에 있는 모든 testcase를 통과한 것을 볼 수 있었다.

5. Discussion

이번 Lab Assignment는 비교적 간단했기 때문에 Verilog 언어에 대해서는 기존에 많이 사용하는 C, Python 같은 언어와 달라 조금 헷갈리기는 했지만 큰 어려움을 겪지는 않았다. 하지만 본인의 경우(20160030, 고은석) ModelSim 프로그램에서 Simulation을 실행 시에 테스트 결과를 보다 쉽게 관찰하기 위해 쓰여진 '$Display' command가 ModelSim의 transcript 창에서 보이지 않는 문제가 있었다. 이 문제로 test를 실행하고 나서 각 testcase의 Pass/Fail 여부를 판단하기 위해 ModelSim의 Wave Window에 표시되는 Passed/Failed 변수의 증가로 판단을 진행했는데 이번 Assignment처럼 test가 비교적 간단하고 testcase의 개수가 적다면 큰 문제가 안되지만 그렇지 않다면 힘들어질 것이기 때문에 빠른 시일 내에 해결이 필요하다고 생각된다.

6. Conclusion

작성한 alu.v를 이용하여 test 해본 결과 실패한 ALU_TB.v 파일 내에서 실패한 testcase가 하나도 없었다는 것을 통해 alu.v가 적절히 작성되었다고 판단했다. 이번 Assignment를 통해 Verilog의 기본적인 구조 및 Verilog로 작성된 파일을 Simulation 해보기 위한 ModelSim 프로그램의 기본적인 사용방법을 익힐 수 있었다.