Logic Design Lab Report: Week 6

2013-11826 임주경

1. Introduction

Verilog를 이용해서, 1-bit arithmetic logic unit (ALU)를 구현해본다.

2. Implementation

구현하고자 하는 ALU는 1-bit inputs (ain, bin) 과 2-bit control inputs ([1:0] mode, [1:0] opcode) 그리고 1-bit outputs (cout, result)로 구성된다. Control input인 mode 와 opcode에 따라서 operator가 달라진다. 구현한 코드는 아래의 그림 1과 같다.

```
1  `timescale 1ns / 1ps
2
3  module alu
4  (
5     input [1:0] mode, opcode,
6     input ain, bin,
7     output reg result, cout
8  );
9
10 always @(*) begin
11    case({mode, opcode})
12       4'b0000 : {cout, result} = {1'bx, ain};
13       4'b0001 : {cout, result} = {1'bx, ~ain};
14       4'b0010 : {cout, result} = {1'bx, ((ain * ~bin) + (~ain * bin))};
15       4'b0011 : {cout, result} = {1'bx, ~((ain * ~bin) + (~ain * bin))};
16       4'b0100 : {cout, result} = {1'b0, ain};
17       4'b0101 : {cout, result} = ~{1'b0, ain};
18       4'b0110 : {cout, result} = ain + bin;
19       4'b0111 : {cout, result} = {(~ain) * bin, (ain * bin) + (~ain * ~bin)};
20       4'b1000 : {cout, result} = ain + 1'b1;
21       4'b1001 : {cout, result} = (~{1'b0, ain}) + 2'b1;
22       4'b1010 : {cout, result} = ain + bin + 1'b1;
23       4'b1011 : {cout, result} = bin + ((~{1'b0, ain}) + 2'b1);
24       default : {cout, result} = {1'bx, 1'bx};
25    endcase
26
27 end
28
29 endmodule
```

그림 1. ALU 코드 (behavioral description)

Verilog 모듈을 이용해서 behavioral description 표현을 사용하였다. Always 구문을 사용하기 위해서 output은 register로 선언하였다. case구문을 사용하여 {mode, opcode}를 인자로 받도록 했다. Mode가 2'b00일 때 Bit operations, 2'b01 또는 2'b10일 때 Arithmetic operations, 2'b11일 때 Don't care로 구성된다. 구현은 실습 6주차 강의자료의 ALU Specification을 참고하였다. 각 case는 ALU specification의 comment를 참고하였으며, XOR, XNOR연산은 truth table을 이용하여 구현하였

다. Don't care는 default를 이용해서 구현하였다. 이때, {mode, opcode} 가 4'b0111일 때, Complement and add 연산에서 {cout, result} = (~ain) + bin; 과 같이 구현할 경우 문제가 생김을 발생하였는데, 이러한 원인은 ain을 1-bit input으로 인식하지 못해서 생긴 것으로 추측된다. 예를 들어, ain = 0, bin = 0일 경우 올바른 output은 cout = 0, result = 1이어야 한다. 하지만, (~ain) = (~0) = 2'b11로 인식되어, 결과가 cout = 1, result = 1로 출력됨을 테스트 벤치 코드를 활용해서 확인할 수 있었다. 따라서 이러한 문제는 truth table을 만들어서 코드의 19줄과 같이 해결할 수 있었다.

3. Result

Mode, opcode, ain, bin의 모든 조합에 대해서 올바른 결과값을 얻어낼 수 있었다. Testbench 코드를 이용해 이를 확인할 수 있었다.

4. Conclusion/Discussion

Behavioral description의 case구문을 사용해서 간단히 코드를 마칠 수 있었다. Arithmetic operations의 Complement and add와 같이 초기에 bitwise 연산이 의도와 다르게 연산 되는 일이 발생했는데, 이는 {cout, result} = (~ain) + bin으로 구현할 경우, (~ain)을 2-bit로 인식해 bitwise연산이 진행돼서 발생하는 문제임을 추측해 볼 수 있었다. 이와 같은 상황에서는 truth table을 사용해서 S-o-P 표현으로 output을 구현해도 문제없음을 직접 확인할 수 있었다.