컴퓨터 구조

2013210061

채윤병

Lab Session 7

**alu1bit.v**

module alu1bit(result,cout,a,b,less,cin,binvert,operation);

input a,b,less,cin,binvert,operation;

output reg cout, result;

reg andout, orout, addout, afterinvert;

always @(operation or a or b or cin)

begin

case(binvert)

2'b0: afterinvert = b;

2'b1: afterinvert = ~b;

endcase

andout = a & afterinvert;

orout = a | afterinvert;

{cout, addout} = a + afterinvert + cin;

case (operation)

2'b00: result = andout;

2'b01: result = orout;

2'b10: result = addout;

2'b11: result = less;

endcase

end

endmodule

**msbalu1bit.v**

module msbalu1bit(set,result,cout,a,b,less,cin,binvert,operation);

input a,b,less,cin,binvert,operation;

output reg cout, result, set;

reg andout, orout, addout, afterinvert;

always @(operation or a or b or cin)

begin

case(binvert)

1'b0: afterinvert = b;

1'b1: afterinvert = ~b;

endcase

andout = a & afterinvert;

orout = a | afterinvert;

{cout, addout} = a + afterinvert + cin;

set = addout;

case (operation)

2'b00: result = andout;

2'b01: result = orout;

2'b10: result = addout;

2'b11: result = less;

endcase

end

endmodule

**alu4bit.v**

`include "msbalu1bit.v"

`include "alu1bit.v"

module alu4bit (A, B, CIN, BINV, OP, COUT, Y);

input [3:0] A,B;

input CIN,BINV,OP;

output [3:0]COUT;

wire set;

output [3:0] Y;

alu1bit alu1bit00 (Y[0],COUT[0],A[0],B[0],set,CIN,BINV,OP);

alu1bit alu1bit01 (Y[1],COUT[1],A[1],B[1],0,COUT[0],BINV,OP);

alu1bit alu1bit02 (Y[2],COUT[2],A[2],B[2],0,COUT[1],BINV,OP);

msbalu1bit alu1bit03 (set,Y[3],COUT[3],A[3],B[3],less,COUT[2],BINV,OP);

endmodule

**tbalu4bit.v**

`include "alu4bit.v"

module tb4alu;

reg [3:0] A, B;

reg CIN, BINV;

reg [1:0] OP;

wire COUT;

wire [3:0] Y;

alu4bit alu4bit00 (A, B, CIN, BINV, OP, COUT, Y);

initial begin

A = 4'b0000;

B = 4'b0000;

CIN = 1'b0;

BINV = 1'b0;

OP = 2'b00;

#10

A = 4'b0101;

B = 4'b1111;

OP = 2'b00;

#10

A = 4'b1010;

B = 4'b1111;

OP = 2'b00;

#10

A = 4'b0101;

B = 4'b0000;

OP = 2'b01;

#10

A = 4'b1010;

B = 4'b0000;

OP = 2'b01;

#10

A = 4'b0111;

B = 4'b0101;

CIN = 1'b0;

BINV = 1'b0;

OP = 2'b10;

#10

A = 4'b0100;

B = 4'b0010;

CIN = 1'b1;

BINV = 1'b0;

OP = 2'b10;

#10

A = 4'b0101;

B = 4'b0010;

CIN = 1'b1;

BINV = 1'b1;

OP = 2'b10;

#10

A = 4'b0100;

B = 4'b0101;

CIN = 1'b1;

BINV = 1'b1;

OP = 2'b10;

#10

A = 4'b0111;

B = 4'b0110;

CIN = 1'b1;

BINV = 1'b1;

OP = 2'b11;

#10

A = 4'b0011;

B = 4'b0111;

CIN = 1'b1;

BINV = 1'b1;

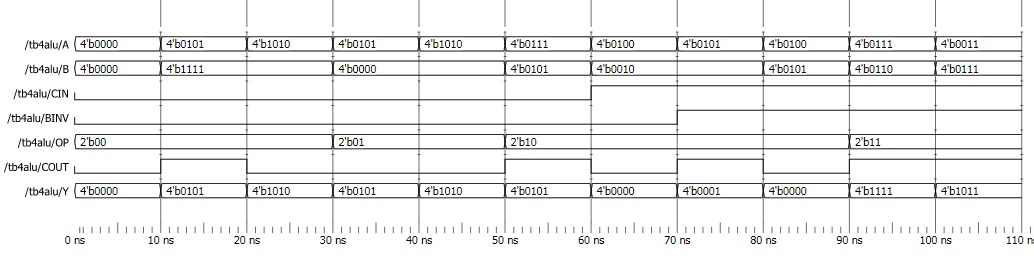
OP = 2'b11;

#10

$stop;

end

endmodule



이 ALU는 4비트를 계산한다. ALU는 3개의 1비트 가산기와 1개의 최상위 비트 가산기로 이루어져있다. AND, OR, 가산기-감산기, set less than 명령을 구현했다. MUX가 네 가지 값 중 하나를 골라 result로 출력한다. AND는 {BINVERT,OPERATION}이 000일 때, OR은 001일 때, add는 010일 때, sub는 110일 때, SLT는 111일 때 계산된다. 위의 표는 BINVERT와 OPERARTION 입력을 받고 AB와 B를 4비트 연산한 것이다.

always @(operation or a or b or cin) 구문을 이용한 Behavioral Level Modeling을 이해하기가 어려웠는데 값 들이 변할 때마다 반복된다는 것을 표를 이용해 쉽게 확인할 수 있었다. 이 모듈의 모든 입력을 괄호 안에 넣기 때문에 구문을 쓰기 전과 후는 변화가 없을 것이라고 예상했지만 수정을 하니 오류가 떴다. ALU를 설계할 때와 우리가 이전에 설계한 Verilog와는 다른 것이 무엇인지 이해하기가 어려웠다. 이해하도록 노력하면 나중에 Clk 개념을 도입할 때 다양하게 쓰일 수 있을 것 같다.

또 Behavioral Level Modeling을 하려면 출력 값들이 output이 아니라 output reg로 선언돼야 하는 것을 몰라서 고생을 했었는데 수정을 하니 잘 돌아가는 것을 확인할 수 있었다.