Lab2 report

作者

周延儒

digital logic design

[教師姓名]

Report:

This is our lab 2 ，in this lab, we're required to implement an ALU with 32bit adder.

First of all ，ALU stands for arithmetic logic unit, in computer architecture, ALU computes the data from the registerfiles, and output the corresponding results and the input is the opcode, which,in here ,we declared as fs.

Moreover , inside the ALU , we have to implement a 32bit adder with the input [31:0]a,b ,cin and the output cout ,[31:0]. Plus because we have to detect whether the result is overflowed , we add a additional output signal **overflow** to detect it.

To begin with the 32bit adder , we build a module ，module Adder32(A,B,Cin,C,Y,overflow)

And we design it with bit-slide method to implement it.

In addition, because we add a extra output overflow, we must changed the previous module a little bit.
Fortunately , to achieve the goal , we only have to add a assignment ,assign overflow= c[n] ^ c[n-1] , where overflow is the c[n] xor c[n-1].

So we have completed the 32bit adder.

Then we can begin to design the ALU;

First of all , after analyzing the ALU, we need the module 32 bit adder.

Plus, because the function depends on the fs(opcode), we can use **case** to implement it.

Because the size of fs(opcode) is 4 bits , there 2^4=16 cases to begin. We declare 4'b0000 to 4'1110 and default,

Noted that in the **case** assignment , we had better to implement all the cases even if we don't need some cases. This is because if we don't implement them, the Verilog tool may synthesis some circuits we don't expected . As a result we declare default instead of 4'1111 although we may have implement all the cases.

This technique is the very important coding style.

In each case, we can design the function we want. And

It's crucial that we can never declare a module in a always block ,so we have to declare the function module outside the always block with a additional wire or reg in it , and the connect the input and output with these wire and reg. So no matter we change the value of input , the output will be the what we desire .

This technique helps to implement the cases related the 32 bit adder.

In the end , we only have to deal with the shift function .

Verilog provides the logical shift(>>) · and arithmetic shift (>>>) to implement . However, in arithmetic shift ,

We must declare as $signed (A)<<<1'b1, and we to have to determine whether the shift will cause overflow , so the overflow signal would be if((A[31] & ~A[30]) || (~A[31] & A[30])).

Fortunately , the >>> won't cause the overflow.

After we implement the design , we can simulate it with waveform .

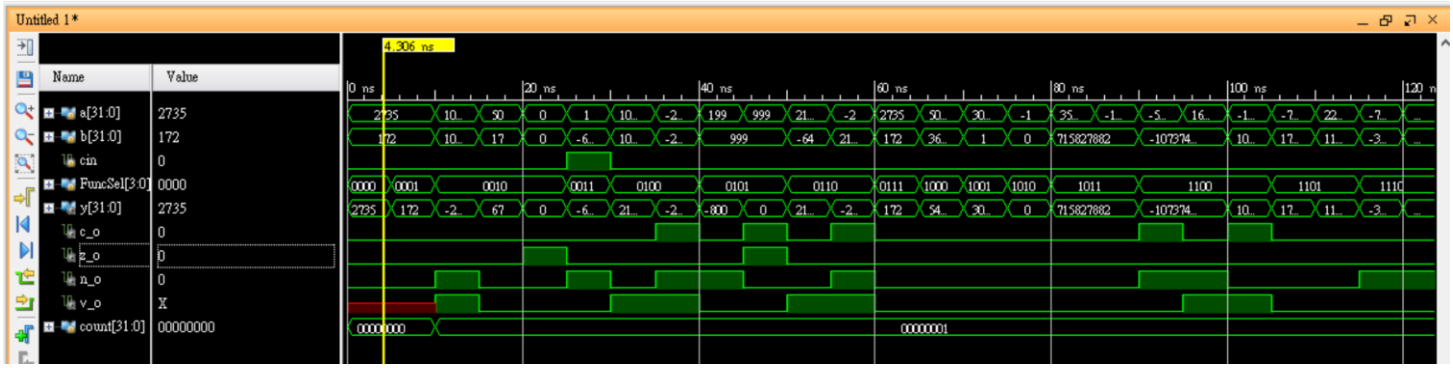
Then we can synthesis the design, in this process , some error may happen instead of in the simulation .

These errors may be due to improper coding style.

For instance , as what we have mentioned, we don't implement all the case in case assignment.Or we use one variable multiple times in a single always block .

These may cause synthesis error ,such as synthesis unexpected circuit or even failure.

Running results : waveform(by vivado Verilog tool)



Summary:

In this lab, we acquired more advanced technique to build our module: ALU with the add32bit. Up to now, we have learned to construct the combinational circuit without any clock via Verilog tool . Additionally we learn to synthesis the electronic circuit . Actually , we may implement it on FPGA to test our design. Finally , we can make our circuit design really IC chip .

Hope we can learn more in this coarse.