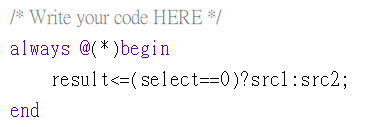
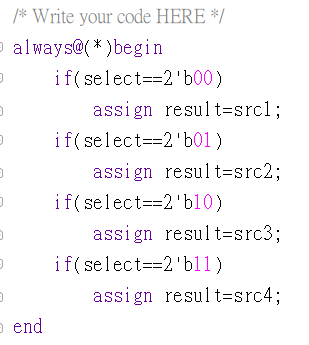
report of lab02-32 bits ALU 109550027 紀竺均

● Detailed description of the implementation

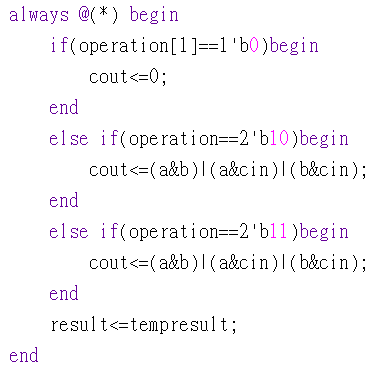
First, I implement 2to1 multiplexer and 4to1 multiplexer. 



Then, I use the multiplexers to implement 1bit ALU.

|  |  |
| --- | --- |
|  |  |
|  | (select line: Binvert) |
|  |  |

In always block, I assign cout register and result register.



After finish 1bit ALU, I try to implement 32 bits ALU.

1. I assign operation, Ainvert, Binvert and cin according to input ALU\_control.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| NAME | ALU\_control | operation | Ainvert | Binvert | cin |
| AND | 0000 | 00 | 0 | 0 | 0 |
| OR | 0001 | 01 | 0 | 0 | 0 |
| ADD | 0010 | 10 | 0 | 0 | 0 |
| SUB | 0110 | 10 | 0 | 1 | 1 |
| SLT | 0111 | 11 | 0 | 1 | 1 |
| NOR | 1100 | 00 | 1 | 1 | 0 |
| NAND | 1101 | 01 | 1 | 1 | 0 |

I turn NOR into “not->and” operation and turn NAND into “not->or” operation because this way is easier to implement and I don’t need to deal with the result. On the other hand, I can’t merge Binvert and cin because they’re not exactly the same during all the operations.

1. I call 1bit\_alu module inside 32bits\_alu for 32 times.

|  |  |  |
| --- | --- | --- |
| alu\_0 |  |  |
| alu\_1~  alu\_31 |  |  |

The input ‘less’ in alu\_0 is (src1[31]^(~src2[31])^tempcout[30]), which we call ‘set’, the result of add operation in alu\_31.

I use generate for loop to generate 31 modules in one command.

1. In always block, I assign four registers: result, zero, carryout, and overflow.

|  |  |  |
| --- | --- | --- |
| result |  | Assign result = wire tempresult. |
| zero |  | If every bit of result is 0, then zero=1. |
| cout |  | Assign cout = wire tempcout when the operation is add or sub. |
| overflow |  | Discuss below. |

overflow occurs when:

(+A)+(+B)=(-Result)

(-A)+(-B)=(+Result)

(+A)-(-B)=(-Result)

(-A)-(+B)=(+Result)

here, (+A) means A is positive.

I simplify the logic by:

if add operation:

if ~(A^B)&(A^Result) then overflow=1

if sub operation:

if (A^B)&(A^Result) then overflow=1

● Implementation results



● Problems encountered and solutions

At first, I get the error cannot be driven by primitives or continuous assignment because I let reg connect to the output of another module. Thus, I put wire in that position and assign reg = wire in always block.

心得：

上學期修DCL寫verilog都不知道自己在寫什麼，這次看著電路圖一步一步把電路設計出來，覺得很有成就感，也看到了自己的進步。謝謝助教給我們很厲害的testbench，對於我debug很有幫助。