**Computer Organization**

**Architecture diagram:**

1. 1-bit ALU

****

1. 32-bit ALU

****

**Detailed description of the implementation:**

我先製作一個**Full\_Adder**的module來實現**1-bit ALU**中的**算術運算**，把Full\_Adder放入1-bit ALU的module中，再利用case選擇ALU輸出實現behavioral implementation。

然後再ALU module中宣告32個1-bit ALU。

# module ALU\_1bit Ai (src1, src2, Ainvert, Binvert, Cin, operation, result, cout)

其中：

1-bit ALU的**input**：

Src1[31:0] = src1[31:0]

Src2[31:0] = src2[31:0]

Ainvert [31:0] = ALU\_control [3]

Binvert [31:0] = ALU\_control [2]

Cin [0] = ALU\_control[2] ; Cin [31:1] = CarryOut\_wire [30:0]

operation [1:0] = ALU\_control [1:0]

1-bit ALU的**output**我先用**wire**接住值：

Result\_wire [31:0] = result [31:0]

CarryOut\_wire [31:0] = cout [31:0]

再把wire接住的值拿去做if-else的判斷：

1. **result**輸出的結果分為**SLT**和**Arithmetic Logic**兩大類
2. **SLT**：if(a<b) result = 1; else result = 0

因為SLT本身會執行減法，先把result [31:1] = 0，再把result\_wire的Msb值丟給result就好( result[0] = result\_wire [31] )。如果src1-src2<0的話，result\_wire[31]會等於1；如果src1-src2>0的話，result\_wire [31]會等於0，剛好都能當作result[0]的輸出。

1. **Arithmetic Logic** operation：add, sub, AND, OR, NAND, NOR

直接將wire的值存給output reg ( result [31:0] = result\_wire [31:0] )

1. Zero：if (result[31:0] == 0) zero=1;

將result\_wire的0~31個bit做NOR邏輯運算，當全部result\_wire的位元為0時，zero會等於1。

1. Cout和overflow：

把cout和overflow分為Arithmetic operation和Logic operation討論

1. Logic operation：AND, OR, NAND, NOR

因為邏輯運算的結果不會產生CarryIut和Overflow的情形，所以直接令為cout=0, overflow=0。

1. Arithmetic operation：add, sub, SLT

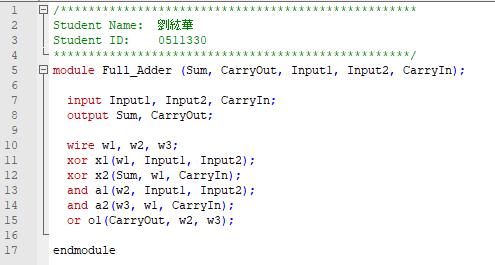
算術運算(包含SLT)就有可能會產生cout和overflow，需要考慮。Cout的想法比較簡單，直接把wire接到output (cout = CarryOut\_wire[31] )

；overflow則是把cout的30和31位元取XOR運算即可 。

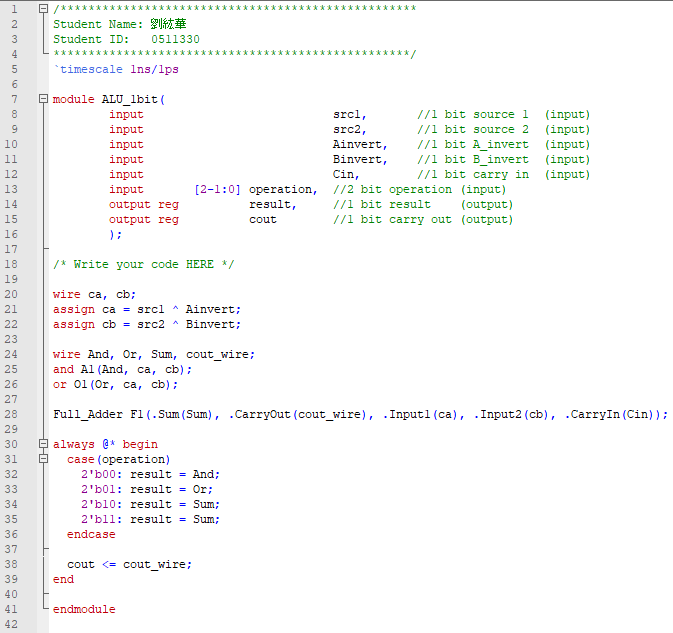
(overflow = CarryOut\_wire[30] ^ CarryOut\_wire[31] )

**Implementation results:**

1. Full\_Adder

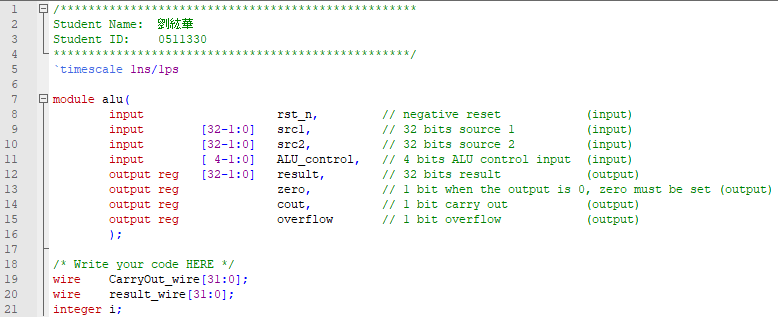


1. 1-bit ALU



1. 32-bit ALU

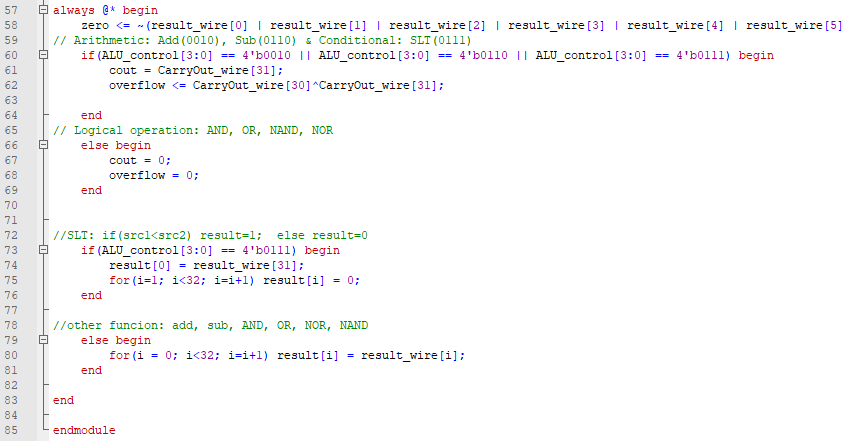
(Part 1) 宣告wire和I/O



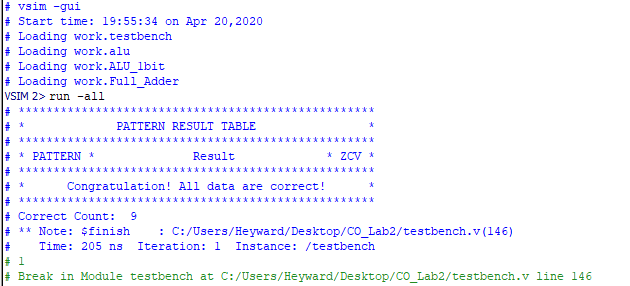
(Part 2) 宣告 32個 1-bit ALU



(Part 3) behavirol implementation of **ZCV** and **SLT** function.



1. Testbench result



**Problems encountered and solutions:**

1. 遇到最難的部分是怎麼執行SLT function，雖然知道把(src1-src2)減法結果的Msb拉回result的Lsb做輸出，但實作起來還是很困難。幸好後來想到把SLT和其他function分成兩類，再把result依不同情況執行。

最後的解法為：

1. Result\_wire先接住1-bit ALU的運算結果。
2. 利用ALU\_control判斷是否為SLT
3. 如果是執行SLT：result [31:1] = 0 ; result [0] = result\_wire [31]
4. 其他的：直接把wire接給output reg ( result [31:0] = result\_wire [31:0] )

2. 執行減法時：src1 + src2的1’s complement + 1，其中+1從哪裡來我想

了很久才發現，把ALU\_control[2] (Binvert) 拿去給最低為位元ALU的Cin當input，只要執行減法時，Binvert都會等於1，剛好能當+1的來源。

**Lesson learnt (if any):**

Verilog好難！

差點寫不完作業之後要提早開始寫！

**Comment:**

這次作業最難的部分其實是不熟悉verilog的語法，希望之後能更清楚語法，而且完全搞懂blocking和non-blocking的邏輯意義和它們在verilog裡面語法區別。