計算機結構 HW1

姓名：莊育權
學號：B03901142
系級：電機三

I、作業報告

一、 8-bit Arithmetic Logic Unit

1. alu_rtl.v & alu_rtl_tb2.v

(1) How to design:

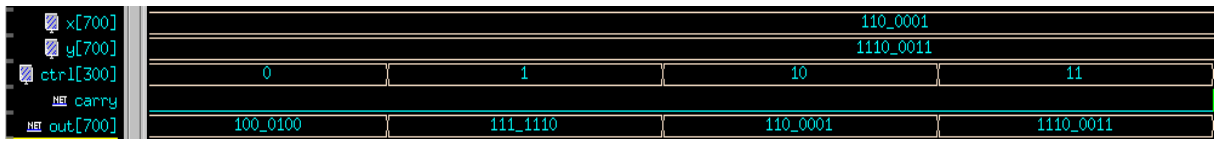
用(? :)語法使其電路像是一個 16-to-1 的 MUX，再將其值 assign 給{carry,out[7:0]}。但由於 x 與 y 為 signed 8bits，需要考慮到第 9bit 的 carry，於是在做加減法運算的時候，需先將 x 與 y 做 sign extension，{x[7],x}與{y[7],y}，要不然電腦會自動給 0 到 x 和 y 的第 9bit，會使得計算出來的值是錯的。

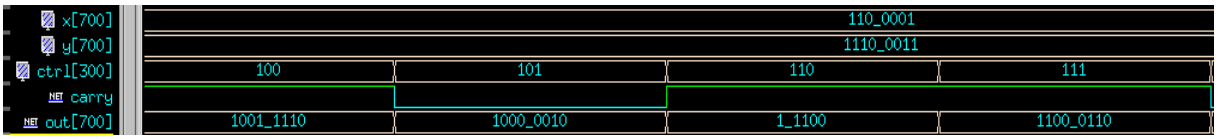
(2) How to verify:

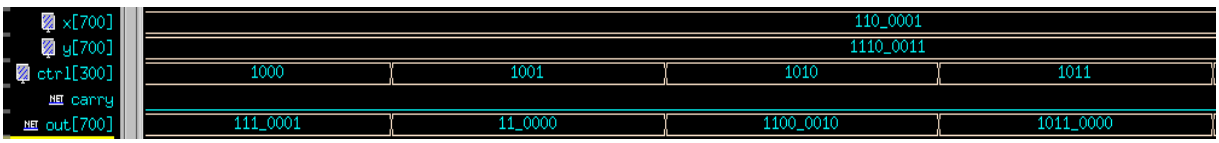
在 test bench 中給定 x=0110_0001 和 y=1110_0011，再藉由手算算出各個 out 的正確值。若 alu_rtl.v 所跑出來的 out 與手算的答案相符，就會顯示出 PASS，反之則為 FAIL。而之所以會選擇這兩組數字，是因為這樣做 x+y 和 x-y 始就會需要用到 carry，因此可藉此知道自己的 code 在這兩種運算時，carry 的值會不會出錯。

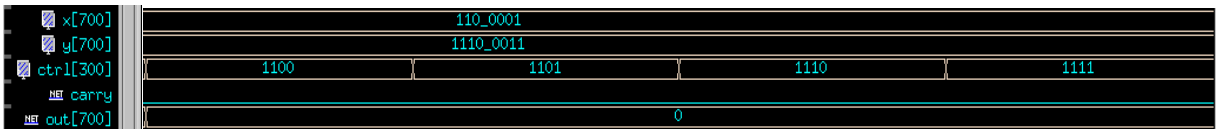
(3) Waveform:

```
ncsim> run
PASS --- 0000 add
PASS --- 0001 sub
PASS --- 0010 and
PASS --- 0011 or
PASS --- 0100 not
PASS --- 0101 xnor
PASS --- 0110 nor
PASS --- 0111 sll
PASS --- 1000 srl
PASS --- 1001 sra
PASS --- 1010 rl
PASS --- 1011 rr
PASS --- 1100 equal
PASS --- 1101 NOP
PASS --- 1110 NOP
PASS --- 1111 NOP
Simulation complete
./alu_rtl_tb2.v:125
ncsim> exit
```









2. alu.v & alu_tb2.v

(1) How to design:

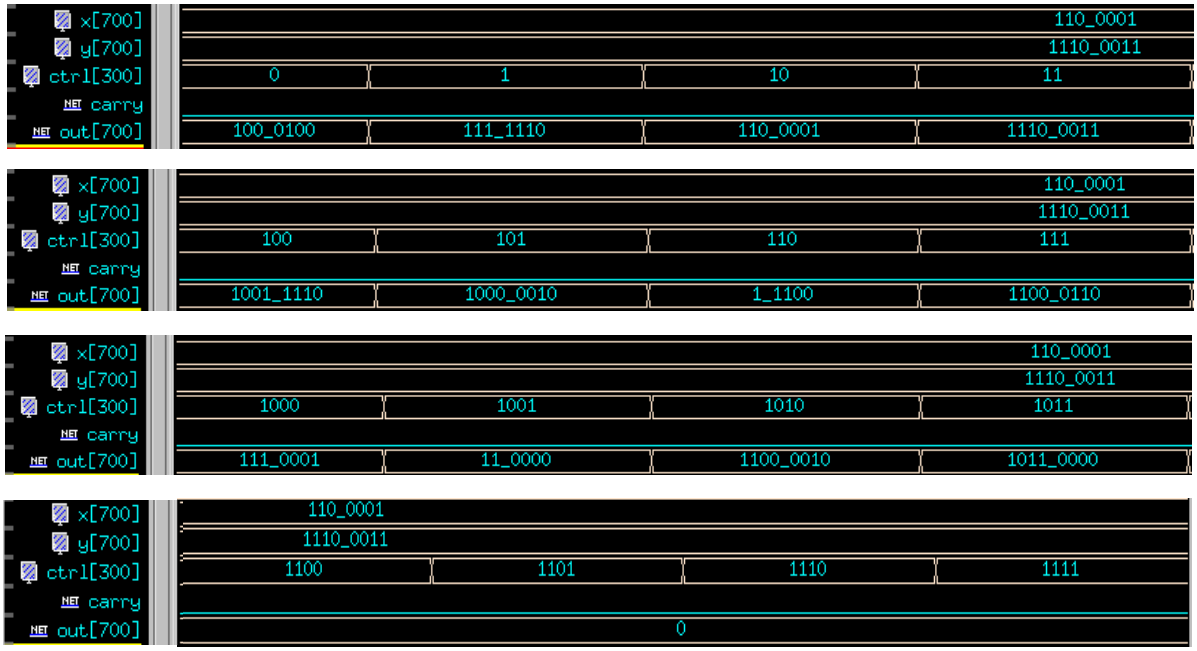
想法與RTL一樣,只是在Behavior Level需使用always block,而always裡面需包含ctrl、x和y,這樣當這三個變數改變的時候,就會產生out。Condition我則使用case的語法。一樣在做加減法的時候,需先講x和y做sign extension。

(2) How to verify:

使用與alu_rtl.v一樣的test bench,若成功則顯示PASS,反之為FAIL。

(3) Waveform:

```
ncsim> run
PASS --- 0000 add
PASS --- 0001 sub
PASS --- 0010 and
PASS --- 0011 or
PASS --- 0100 not
PASS --- 0101 xnor
PASS --- 0110 nor
PASS --- 0111 sll
PASS --- 1000 srl
PASS --- 1001 sra
PASS --- 1010 rl
PASS --- 1011 rr
PASS --- 1100 equal
PASS --- 1101 NOP
PASS --- 1110 NOP
PASS --- 1111 NOP
Simulation complete
./alu_tb2.v:125
ncsim> exit
```



二、 8x8 Register File

1. register_file.v & register_file_tb.v

(1) How to design:

- 先將8個register都default成0。
- 在每個positive edge clock時(always@(posedge clk)),如果(if)WEN 為 1,則根據 RW 的數值(case)來判斷input(busW)應該放入哪個register裡。
- 當RX、RY或暫存器的值改變,根據RX或RY的數值(case)

將暫存器的數值傳出到 busX 或 buxY。

(2) How to verify:

用助教給的 test bench 去做驗證。

```
ncsim> run

Congratulations!! Your Verilog Code is correct!!

Simulation complete via $finish(1) at time 315 NS + 0
./register_file_tb.v:66         $finish;
ncsim> exit
```

三、 Simple Calculator

1. simple_calculator.v & simple_calculator_tb.v

(1) How to design:

- 先建立一個 MUX 的 module
- 再利用前兩題的 module，將 module register_file、module mux 和 module alu 接在一起

(2) How to verify:

用助教給的 test bench 去做驗證。

```
Calculation results: 13 * 12 =  156
*****************************        /|__/|
**                           **      / 0,0  |
**   Congratulations !!      **     /_____   |
** All Patterns Passed!!     **    /^ ^ ^ \  |
**                           **   |^ ^ ^ ^ |w|
*****************************    \m___m__|_|
Simulation complete via $finish(1) at time 350 NS + 0
./simple_calculator_tb.v:322         #(`CYCLE) $finish;
ncsim> exit
```

四、 問題與討論

1.

在第一題的 RT-Level 可以看到 carry 在不是做加減運算時會上升成 1，是因為我 assign 值給{carry,out[7:0]}，因此有 9 個 bit，而例如在做~x 時，x 只有 8 個 bit，電腦會自動補 0 把它補成 9 個 bit，因此在 RT-Level 中有時 carry 會上升成 1。

而在 Behavior Level，因為我只有在做 xy 加減法運算時，會把值 assign 給 carry，故在做其他運算時 carry 電腦皆自動給它為 0。

2.

一開始在做第一題加減法運算時,以為如果 x+y 需要動用到 carry,電腦會自動把它 sign extension 成 9bit 再做加減法運算。但當在跑 test bench 後才發現電腦只會自動補 0,而造成出來的數值會是錯的,必須自己先將 x 和 y 做 sign extension 才行。

3.

由於在一開始以為自己一、二題都做對了,之後第三題其實只是要把它接起來而已,接起來應該就會過了,但最後卻發現有 4 個 errors。打開 nWave 看了 register 那層才發現,當 RX 或 RY 沒改變的時候,輸入給暫存器的值(alu_out)改變,暫存器並不會將值傳給輸出(reg_xout 或 busY),看了 register 的 code 才發現,原本打成像是圖一那樣,而正確應該是要打成圖二這樣才會對。

```
always @(*)  begin
    case (RX)
    3'b000 : busX = r[0];
    3'b001 : busX = r[1];
    3'b010 : busX = r[2];
    3'b011 : busX = r[3];
    3'b100 : busX = r[4];
    3'b101 : busX = r[5];
    3'b110 : busX = r[6];
    3'b111 : busX = r[7];
    default: busX = 0;
    endcase
end
```

圖一、錯誤打法

```
always @(RX,r[0],r[1],r[2],r[3],r[4],r[5],r[6],r[7])  begin
    case (RX)
    3'b000 : busX = r[0];
    3'b001 : busX = r[1];
    3'b010 : busX = r[2];
    3'b011 : busX = r[3];
    3'b100 : busX = r[4];
    3'b101 : busX = r[5];
    3'b110 : busX = r[6];
    3'b111 : busX = r[7];
    default: busX = 0;
    endcase
end
```

圖二、正確打法

由於 always 裡面的*代表的所有變數,是只有一開始宣告為 input 的變數,也就是只有 RX 和 RY 變下面才會啟動,因此應該把 r[0]~r[7] 也都打上去,這樣暫存器的輸入改變,輸出也要跟著改變才對!

五、 心得

之前在積體電路設計課有打過一點點 verilog,但那主要屬於 gate level,而這此是屬於 behavior level 和 RT-level,語法第一次打真的很不習慣。一開始真的不知道甚麼時候要用 wire,甚麼時候要用 reg,if...else 只能放在 always,assign 應該甚麼時候用。但也因為練習過一次,才知道大概的語法應該怎麼打。

可是除了可以 compile 過之外,我覺得最難的是,如何打出可以

synthesizable 的 code，雖然這次作業不要求做到可以合成，但在查資料的過程中，才發現其實有很多邏輯是可以去想的，或是應該避免一些打法，造成合成時會出現很多不該出現的電路，，感覺真的要學好和打好 verilog 還有好長一段路啊!