**Lab2: ALU**

# ALU module

## Mux

* 在always block裡面用case的寫法

## Module Connection

* 天真地以為module可以寫在always block裡面，但是**不行**，要寫在外面。（always block裡面只能有reg）
* 連接module的寫法有兩種：

一、

AND AandB (.a(A[0]), .b(B[0]), .out(out\_AND));

二、

AND AandB (A[0], B[0], out\_AND);

* 要很注意傳進去的大小幾個bits一定要一樣！

## Cout , Negative, Zero, Overflow

* 一開始Cout , Negative, Zero, Overflow 是wire，後來設成reg，搬到always block裡，每個case分別設定不同的值。
* 常出現的錯誤：會忘記換成reg，就直接寫進always block裡：

ncvlog: \*E,WANOTL (ALU.v,142|19): A net is not a legal lvalue in this context [9.3.1(IEEE)].

* Negative和Zero有比較固定的寫法。

Negative

Negative = 1'b0; *//when Negative not valid*

Negative = Y[31]; *//when Negative valid （ Y[31] is sign bit）*

Zero

Zero = ~Y[0]; *//sel 0000~0101*

Zero = (Y == 32'b0) ? 1'b1 : 1'b0; *//sel 0110~1101*

# sel 0000 ~ 0101 （Gate level Using NAND）

這個是一開始最先寫的部分。（因為期中考考過所以比較熟悉）

一開始不知道題目是否允許利用其它module ，所以全部都用nand gate去寫，但是概念是一樣的。

## 演算過程：

NOT ~A = A nand A

AND A & B = ~(A nand B)

= (A nand B) nand (A nand B)

OR A | B = ~(~A & ~B) = ~A nand ~B

= (A nand A) nand (B nand B)

NOR A nor B = ~(A | B)

= (A | B) nand (A | B)

= (A nand A) nand (B nand B) nand (A nand A) nand (B nand B)

XOR A xor B = (A | B) & (A nand B)

= ((A | B) nand (A nand B)) nand ((A | B) nand (A nand B))

Truth Table for XOR

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| A | B | A xor B | A | B | A nand B | (A | B) & (A nand B) |
| 0 | 0 | 0 | 0 | 1 | 0 |
| 0 | 1 | 1 | 1 | 1 | 1 |
| 1 | 0 | 1 | 1 | 1 | 1 |
| 1 | 1 | 0 | 1 | 0 | 0 |

## Module部分：

* 跟上次的lab比起來，這次學會用一個wire作為幾個gate的input。
* Output的wire以該gate所替代的功能為名，例如 or\_ab , not\_a 等等

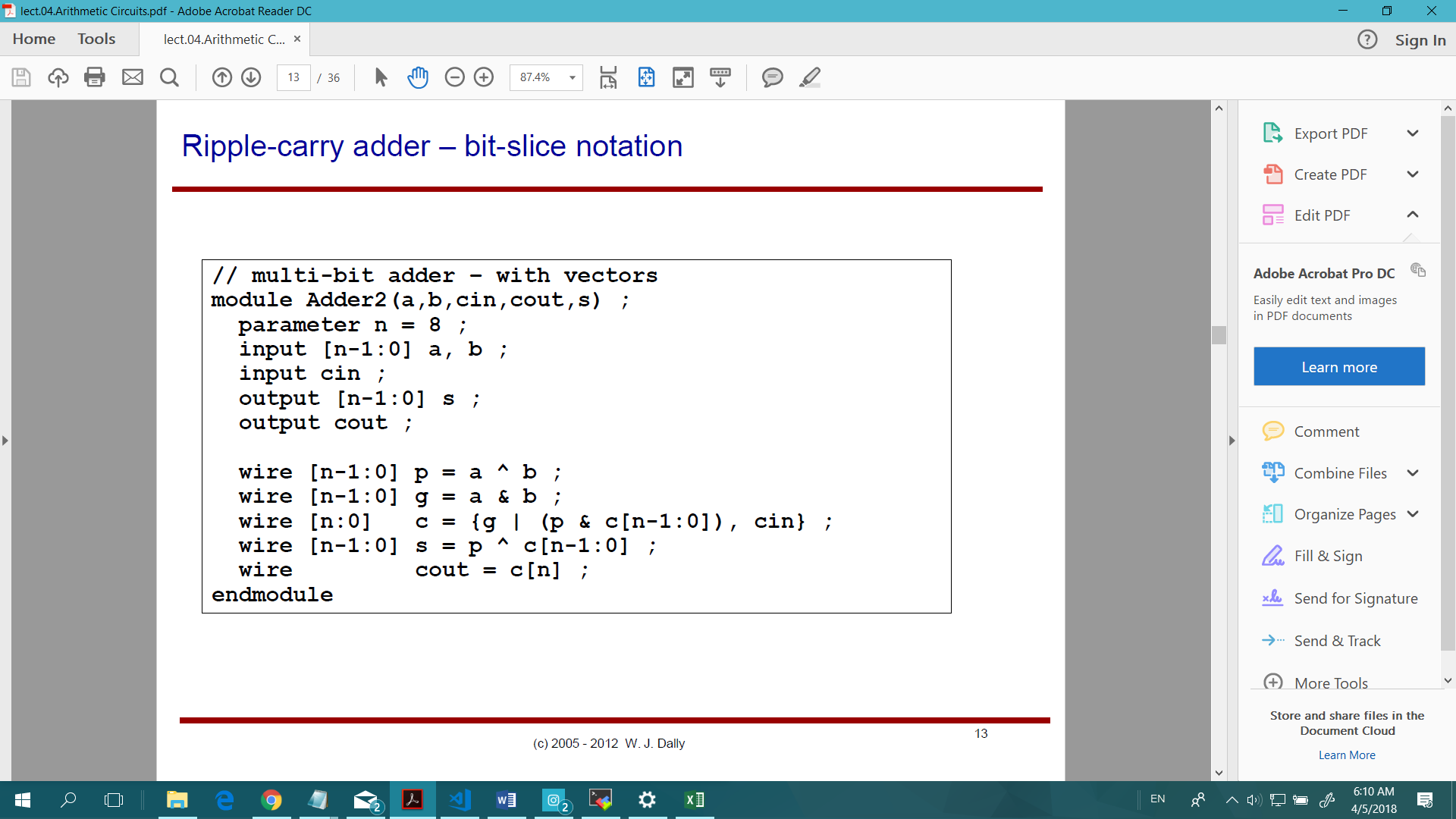
## ALU部分：

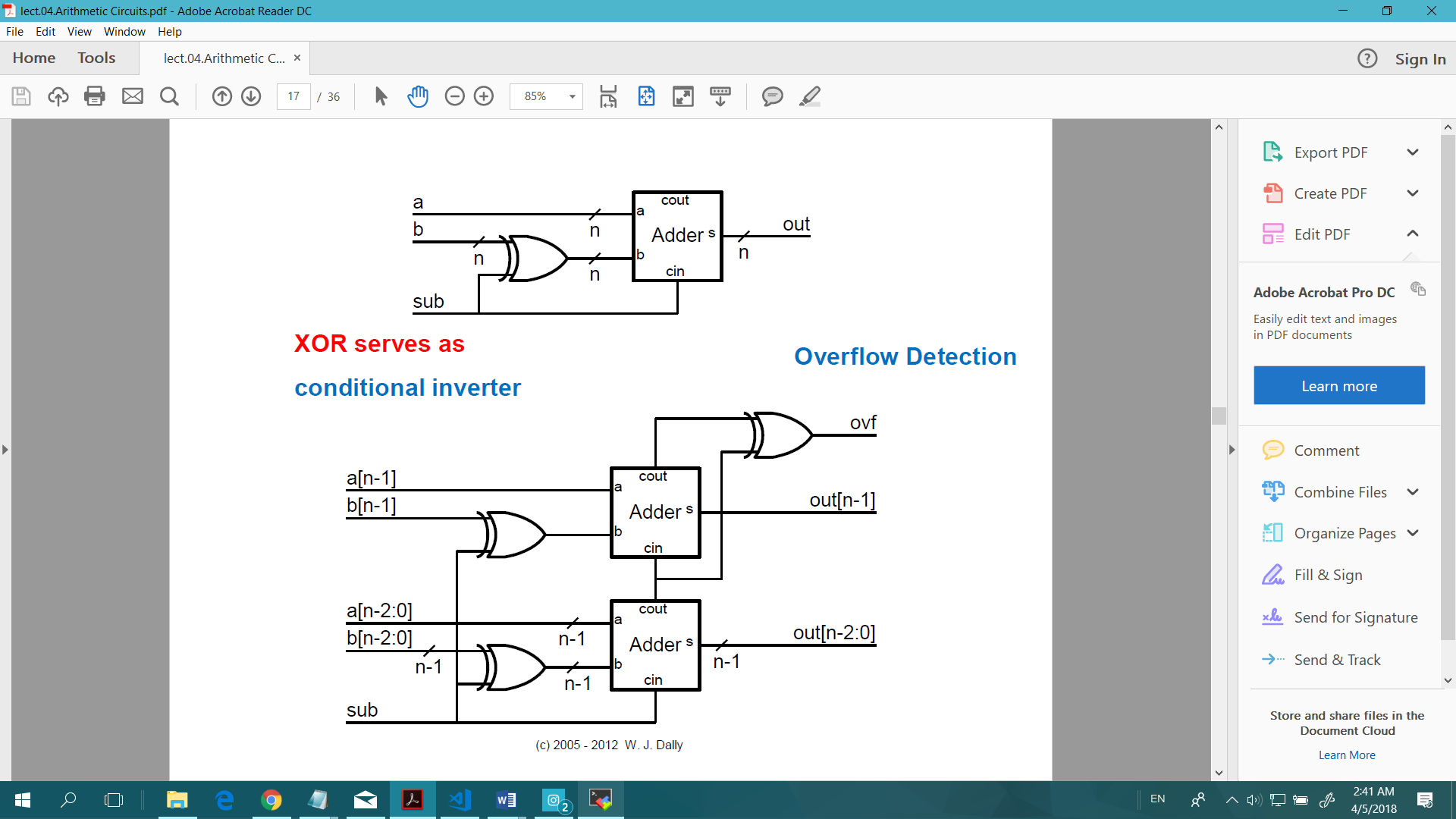
Y = {31'b0, out\_AND}; *//Y[31:0] filled woth 0s, output as Y[0]*

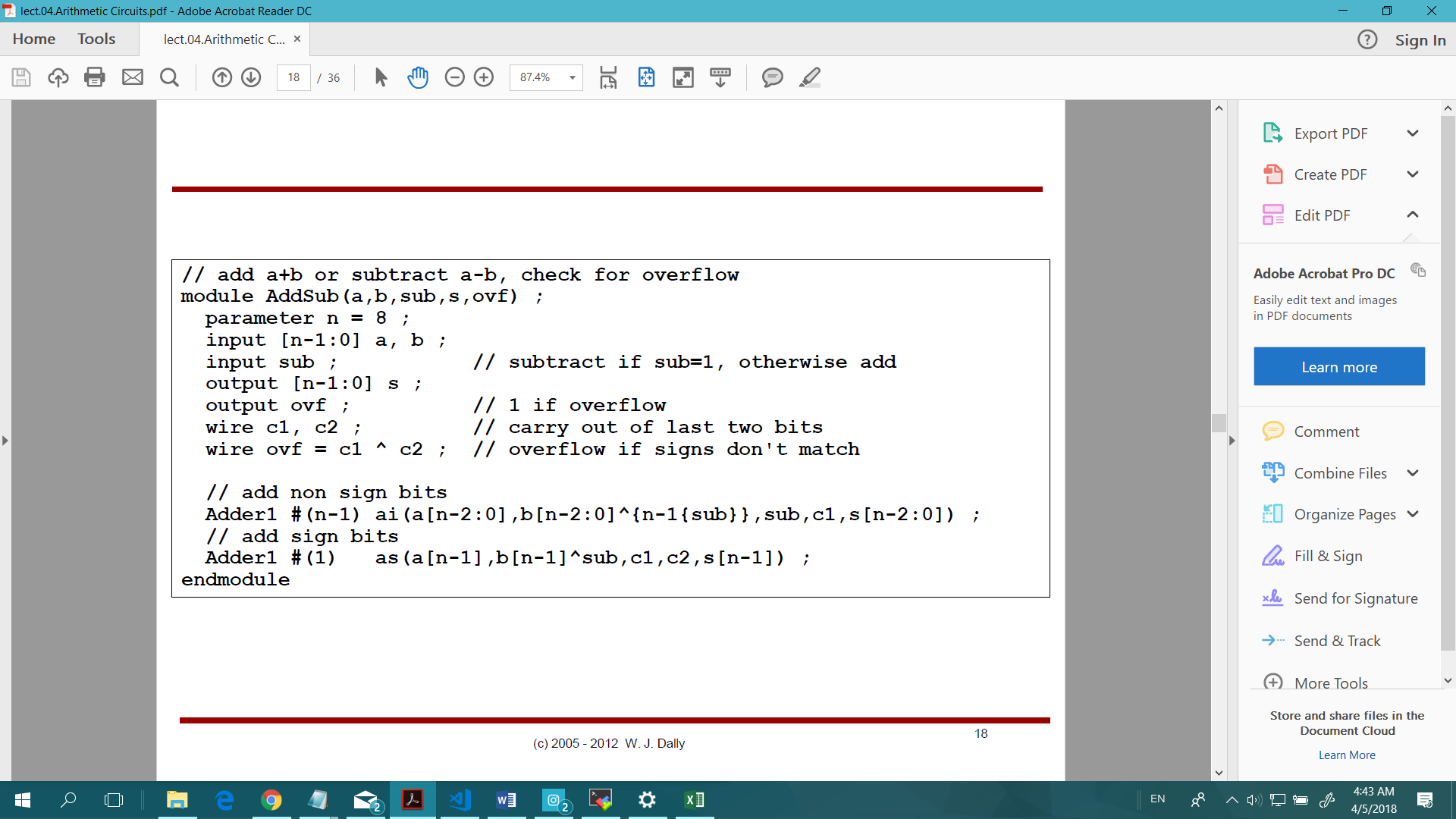
Zero = ~Y[0];  *//Only care about Y[0]*

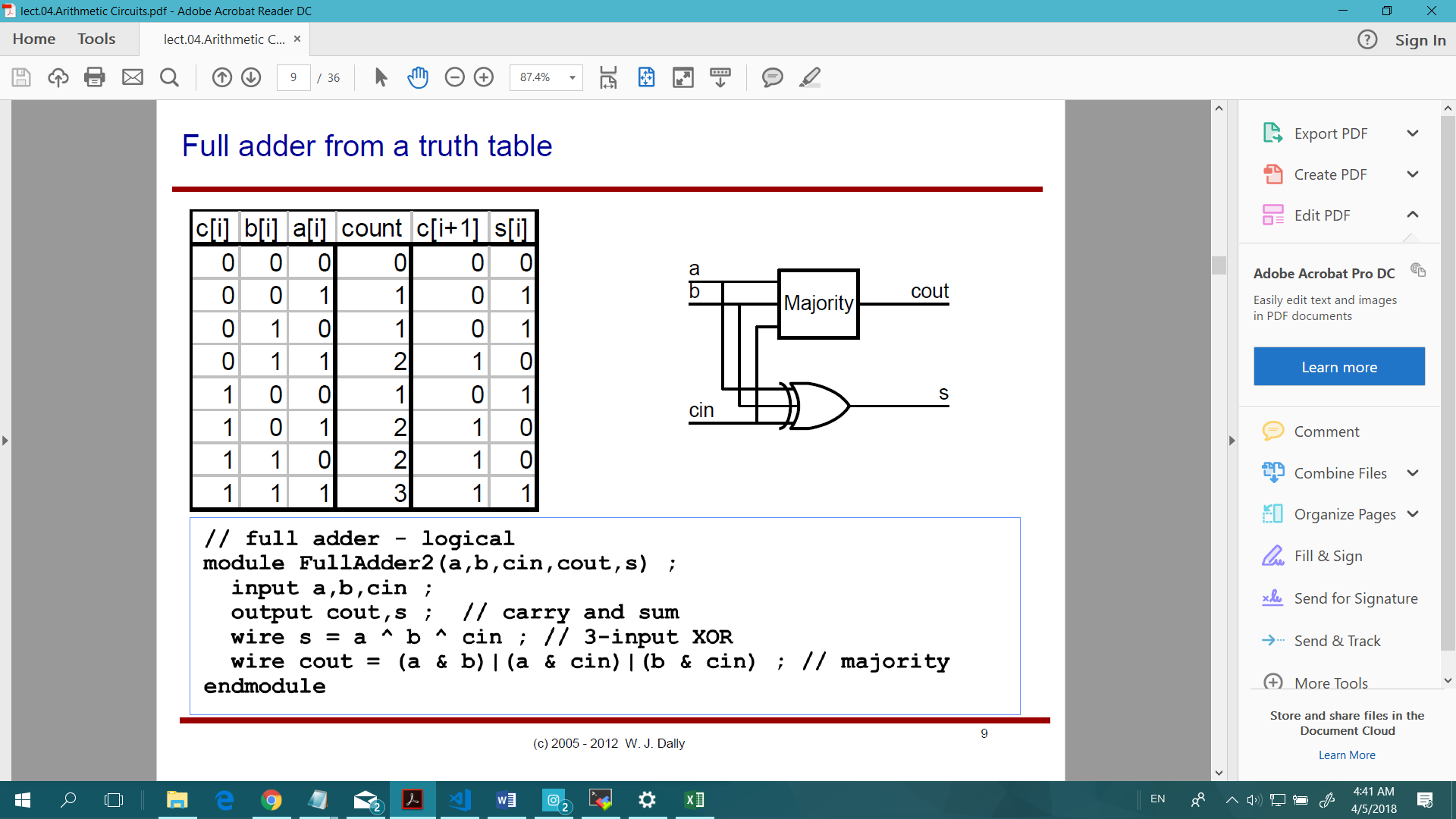
# sel 0110 ~ 1000 (AdderSubtractor)

## Concept

* Bit Slice寫法：使用老師講義中的概念為基礎。





* 原本另外寫一個Full Adder的module，但後來想想寫spec裡面沒有的module好像不太好，所以就把Adder的內容搬進AdderSubtractor裡面。
* 從講義的multi-bit adder的寫法 wire [n:0] c = {g| (p & c[n-1:0]), cin};中學到如何寫bit slice之間的連接。（一開始想用for loop寫但失敗了☹）
* 每個Bit slice內部結構的概念：

module **AdderSubtractor** (A, B, Cin, mode, Cout, Sum);

    input [31:0] A, B;

    input Cin, mode;

    output [31:0] Sum;

    output Cout;

wire [31:0] carry = {(A[30:0] & (B[30:0]^{31{mode}})) | (A[30:0] & carry[30:0]) | ((B[30:0]^{31{mode}}) & carry[30:0]), Cin};

assign Sum[30:0] = A[30:0] ^ (B[30:0]^{31{mode}}) ^ carry[30:0];

assign Cout = (A[31] & (B[31]^mode)) | ((B[31]^mode) & carry[31]) | (carry[31] & A[31]);

endmodule

## Subtraction (2’s completement)

* A – B = A + ~B +1
* Get ~B by XOR with mode (mode == 1 in case of subtraction)
* B[30:0] need to XOR with each bit of mode -> B[30:0]^{31{mode}}

## Cin

* Cin設為(sel[0]|sel[3]|Cin)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | sel[0] | sel[3] | Cin | +1 | sel[0]|sel[3]|Cin |
| Addition | 0 (sel=0110) | 0 | 0 | 0 |
| 0 (sel=0110) | 1 | 1 | 1 |
| Subtraction | 1 (sel=0111, 1000) | 0 (題目不會有Cin) | 1 A-B = A+(~B+1) | 1 |

* 把carry[0] 設為Cin方便寫expression

## Overflow

Overflow cases:

|  |  |  |  |
| --- | --- | --- | --- |
| A[31] | 1 |  | 0 |
| B[31] | 1 |  | 0 |
| Y[31] | 0 |  | 1 |

A[31] same as A[31] Y[31] not same as A[31]

Overflow = (!(B[31]^A[31])) & (A[31]^Y[31]);

Overflow = (!(!B[31]^A[31])) & (A[31]^Y[31]);

B reverse

## Absolute Value

Y = (Sum[31] == 0) ? Sum: ~Sum + 1;

In 2’s completement: -A = ~A + 1

# sel 1001 (Multiply)

題目沒有說不可以直接用乘的，所以 Y = A \* B 。

## Overflow

Overflow = A[15] ^ B[15] ^ Y[31];

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | A[15] | B[15] | Y[31] | Overflow | A[15]^B[15]^Y[15] |
| 正正得正 (0) | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 | 1 |
| 正負得負 (1) | 0 | 1 | 0 | 1 | 1 |
| 0 | 1 | 1 | 0 | 0 |
| 負正得負 (1) | 1 | 0 | 0 | 1 | 1 |
| 1 | 0 | 1 | 0 | 0 |
| 負負得正(0) | 1 | 1 | 0 | 0 | 0 |
| 1 | 1 | 1 | 1 | 1 |

# sel 1010 ~ 1101 (Shift)

目前找到兩種寫法：

sel 1010, 1011 (shift A left by 1-bit)

Y = A << 1'b1;

Y = {A[30:0], 1'b0};

sel 1100 (Logic shift A right by 1-bit)

Y = A >> 1'b1;

Y = {1'b0, A[31:1]};

sel 1101 (Arithmetic shift A right by 1-bit)

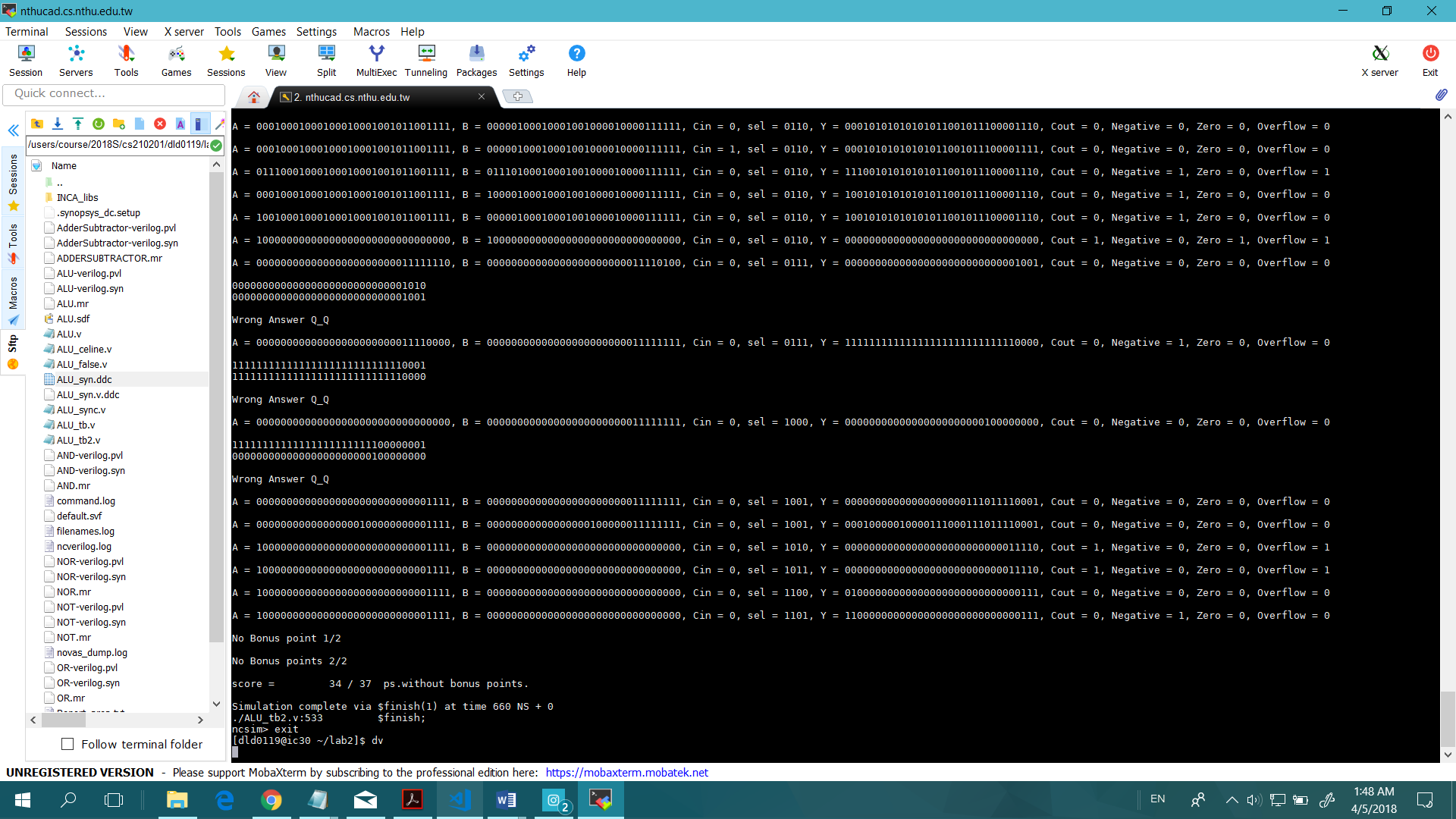
Y = $signed(A) >>> 1'b1; *//learned from internet*

Y = {A[31], A[31:1]};

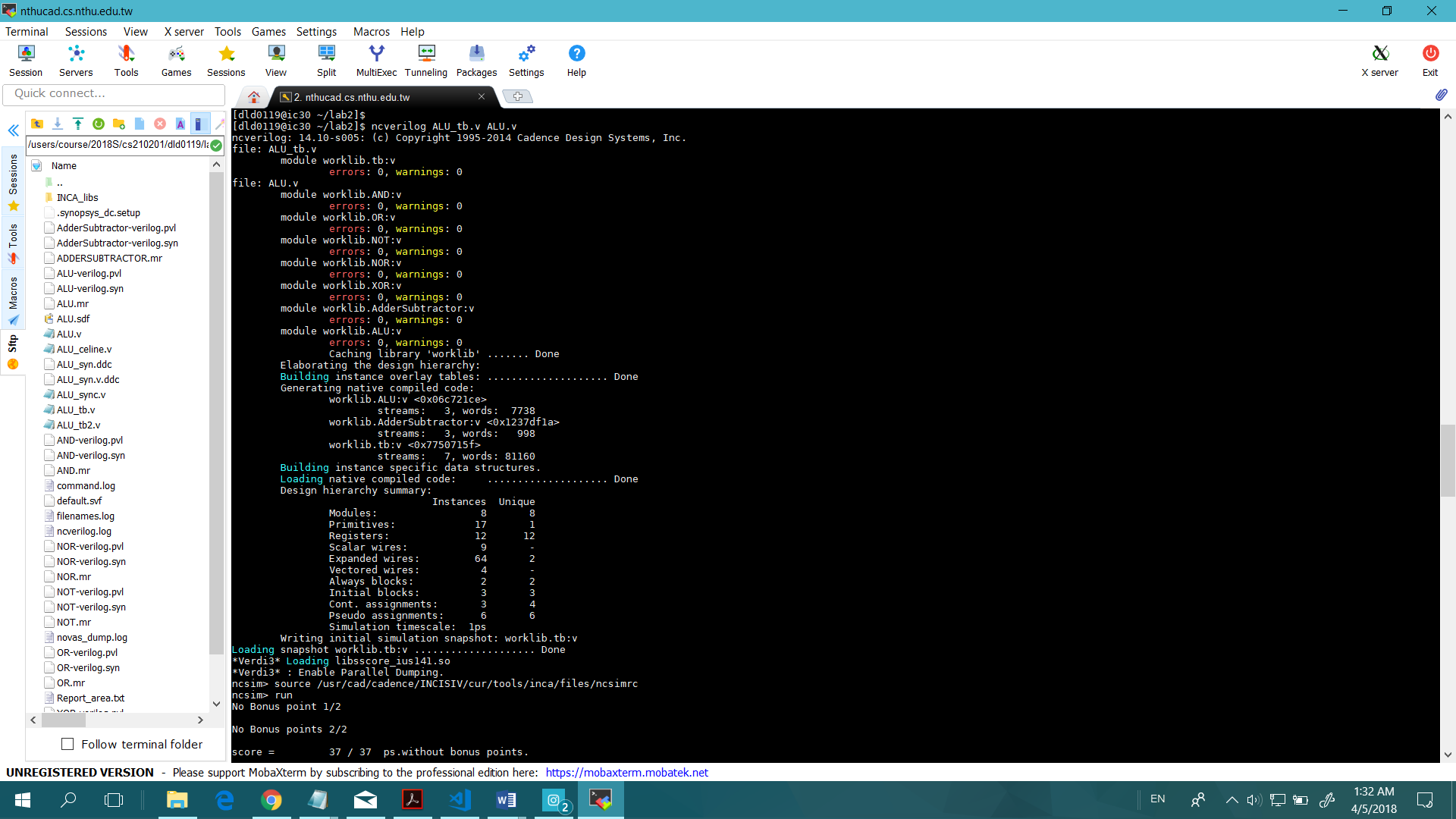
Verilog寫法的多樣化很有趣。

# Testbench

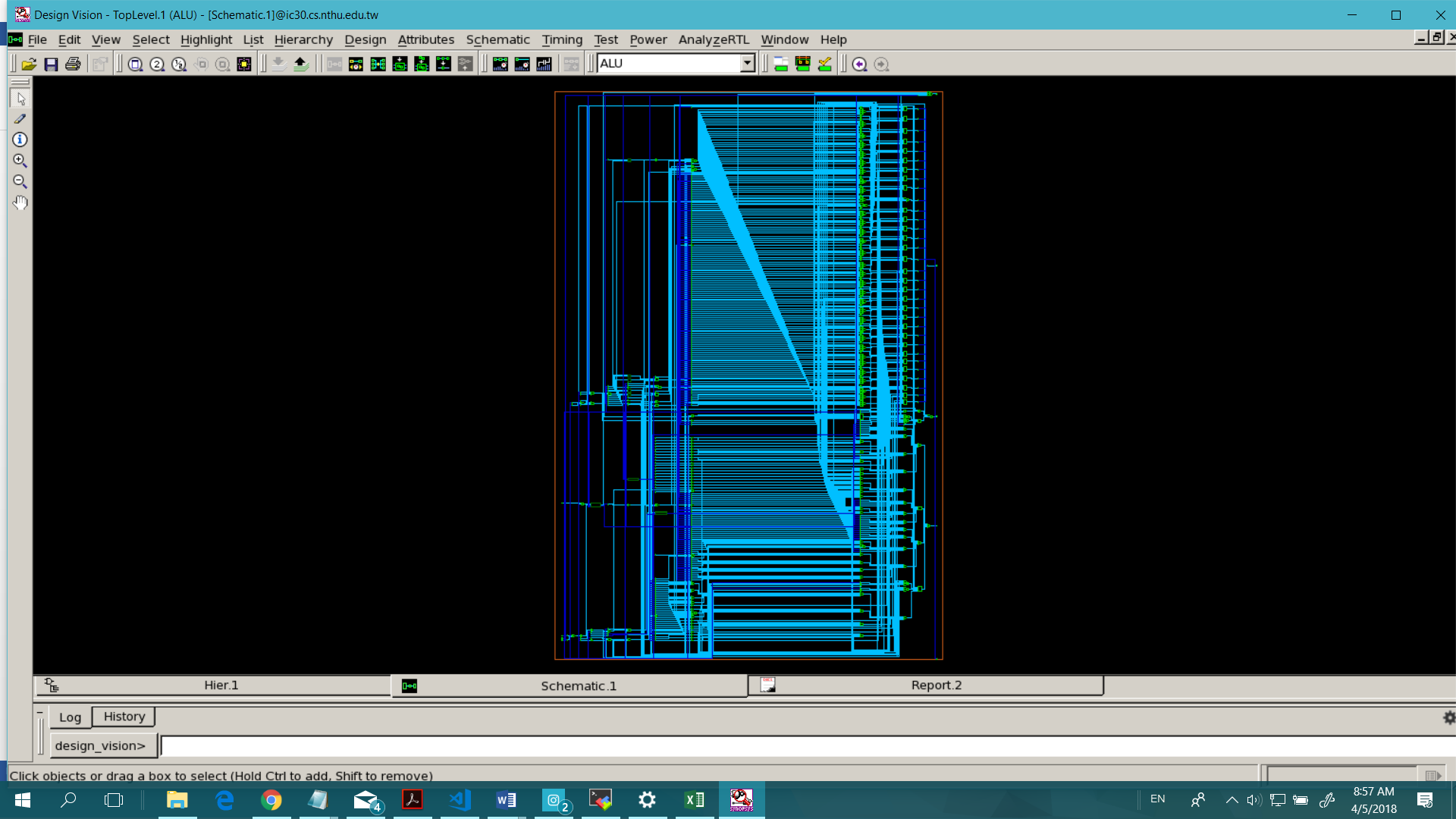
* 為了debug方便，稍微修改了tb以顯示答案對比（上為正確答案，下為輸出答案）



# NCVerilog 模擬結果

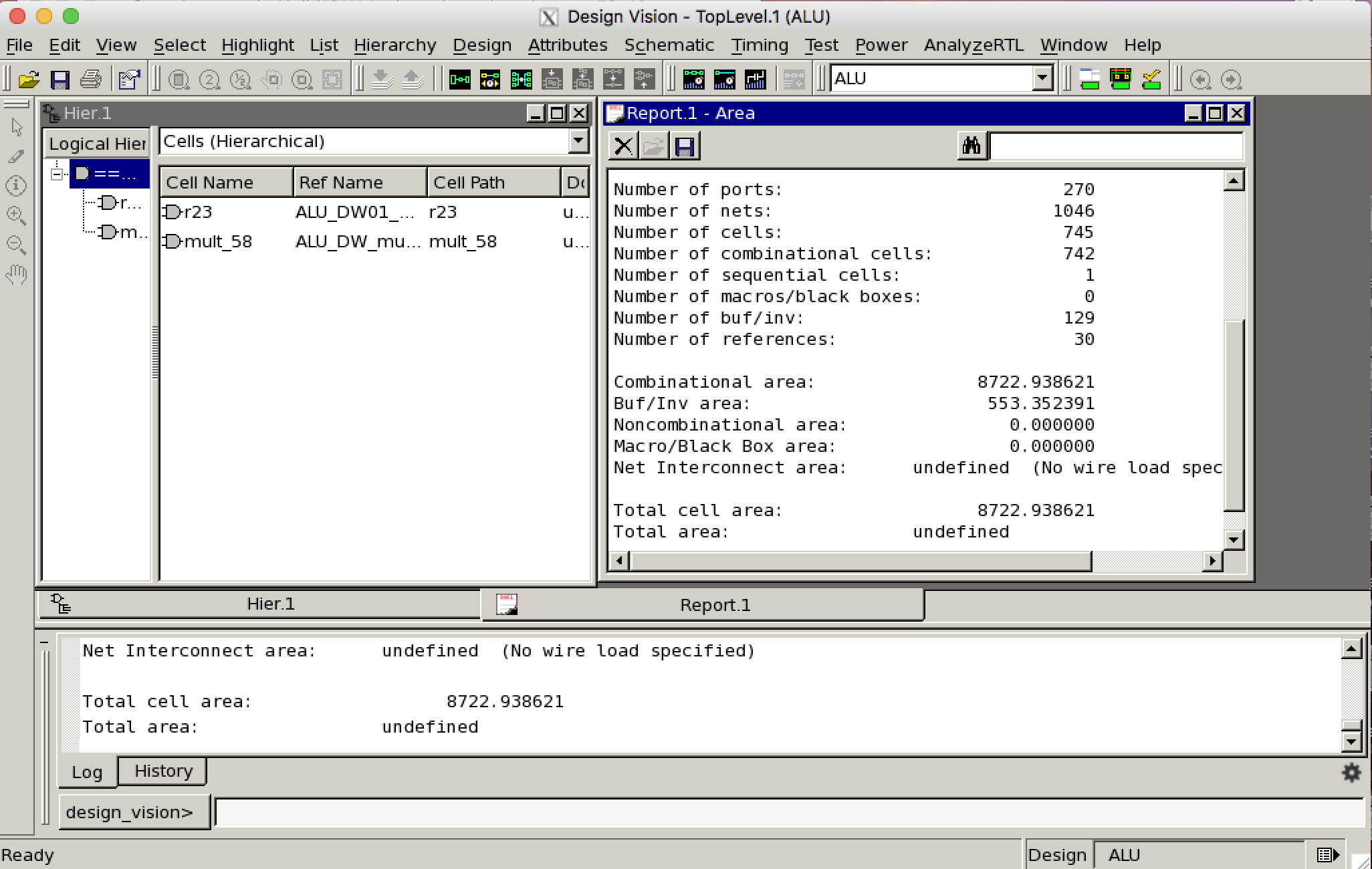


# Design Vision



一開始看到自己的電路是可行的很有成就感。但後來Report Area中看出電路的大小跟gate的數量，發現我寫出來的電路比助教的大兩倍以上。我想到老師上課說考量到經濟效益等，gate的數量應該是越小越好，那我應該是一個很爛的工程師吧？這樣一想就覺得不開心。

*助教的：*



*我的：*

Number of ports: 444

Number of nets: 2005

Number of cells: 1466

Number of combinational cells: 1456

Number of sequential cells: 2

Number of macros/black boxes: 0

Number of buf/inv: 281

Number of references: 39

Combinational area: 18116.350106

Buf/Inv area: 1420.723778