Lab2: ALU

1) 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
```

2) 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

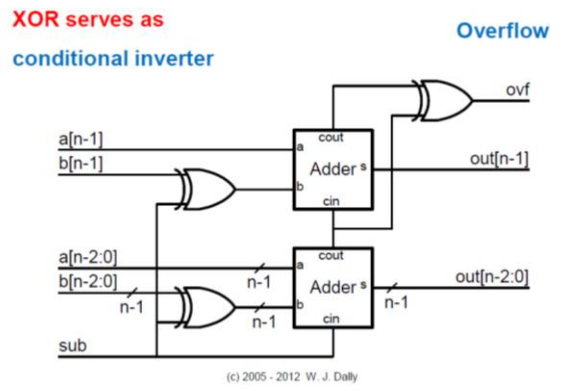
ALU 部分：

```
Y = {31'b0, out_AND}; //Y[31:0] filled woth 0s, output as Y[0]
Zero = ~Y[0];  //Only care about Y[0]
```

3) sel 0110 ~ 1000 (AdderSubtractor)

Concept

- Bit Slice 寫法：使用老師講義中的概念為基礎。



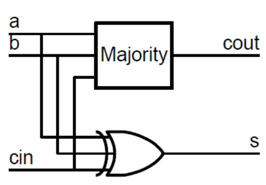
```
// multi-bit adder - with vectors
module Adder2(a,b,cin,cout,s) ;
  parameter n = 8 ;
  input [n-1:0] a, b ;
  input cin ;
  output [n-1:0] s ;
  output cout ;

  wire [n-1:0] p = a ^ b ;
  wire [n-1:0] g = a & b ;
  wire [n:0]   c = {g | (p & c[n-1:0]), cin} ;
  wire [n-1:0] s = p ^ c[n-1:0] ;
  wire         cout = c[n] ;
endmodule
```

```
// add a+b or subtract a-b, check for overflow
module AddSub(a,b,sub,s,ovf) ;
  parameter n = 8 ;
  input [n-1:0] a, b ;
  input sub ;           // subtract if sub=1, otherwise add
  output [n-1:0] s ;
  output ovf ;          // 1 if overflow
  wire c1, c2 ;         // carry out of last two bits
  wire ovf = c1 ^ c2 ;  // overflow if signs don't match

  // add non sign bits
  Adder1 #(n-1) ai(a[n-2:0],b[n-2:0]^{n-1{sub}},sub,c1,s[n-2:0]) ;
  // add sign bits
  Adder1 #(1)   as(a[n-1],b[n-1]^sub,c1,c2,s[n-1]) ;
```

- 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：

Full adder from a truth table

c[i]	b[i]	a[i]	count	c[i+1]	s[i]
0	0	0	0	0	0
0	0	1	1	0	1
0	1	0	1	0	1
0	1	1	2	1	0
1	0	0	1	0	1
1	0	1	2	1	0
1	1	0	2	1	0
1	1	1	3	1	1



```
// full adder - logical
module FullAdder2(a,b,cin,cout,s) ;
  input a,b,cin ;
  output cout,s ;  // carry and sum
  wire s = a ^ b ^ cin ; // 3-input XOR
  wire cout = (a & b)|(a & cin)|(b & cin) ; // majority
endmodule
```

```
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

4) 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

5) 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]};
```

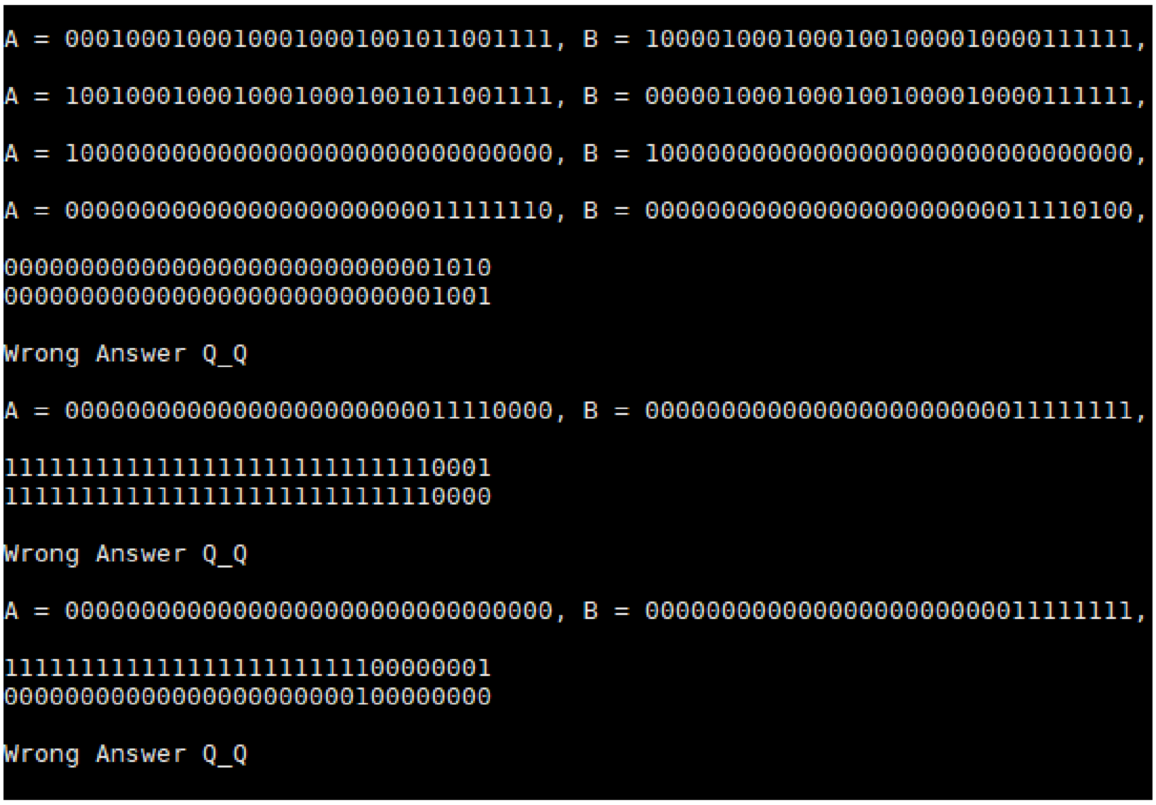
sel 1101 (Arithmetic shift A right by 1-bit)

```
Y = $signed(A) >>> 1'b1; //learned from internet
Y = {A[31], A[31:1]};
```

Verilog 寫法的多樣化很有趣。

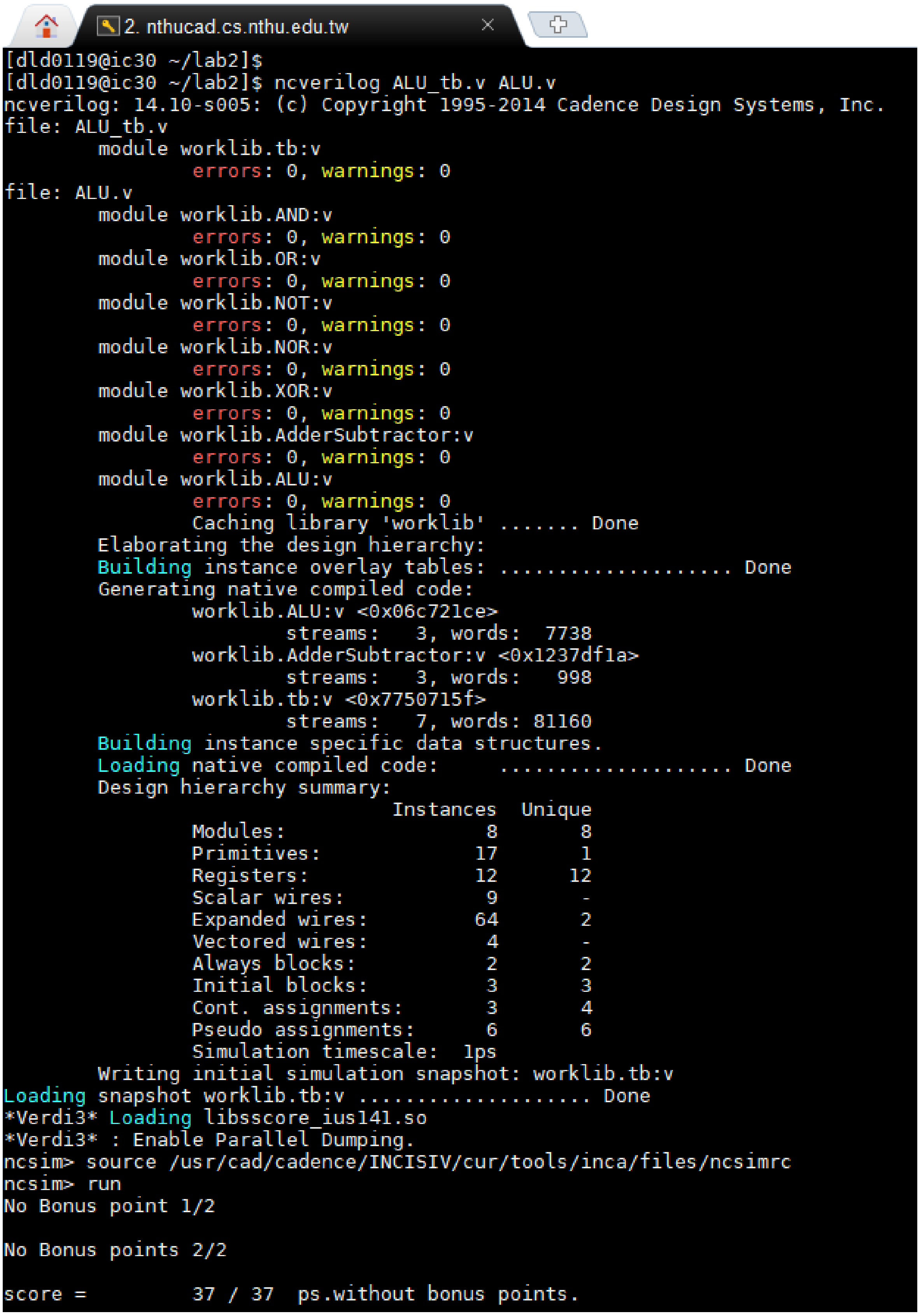
6) Testbench

- 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）



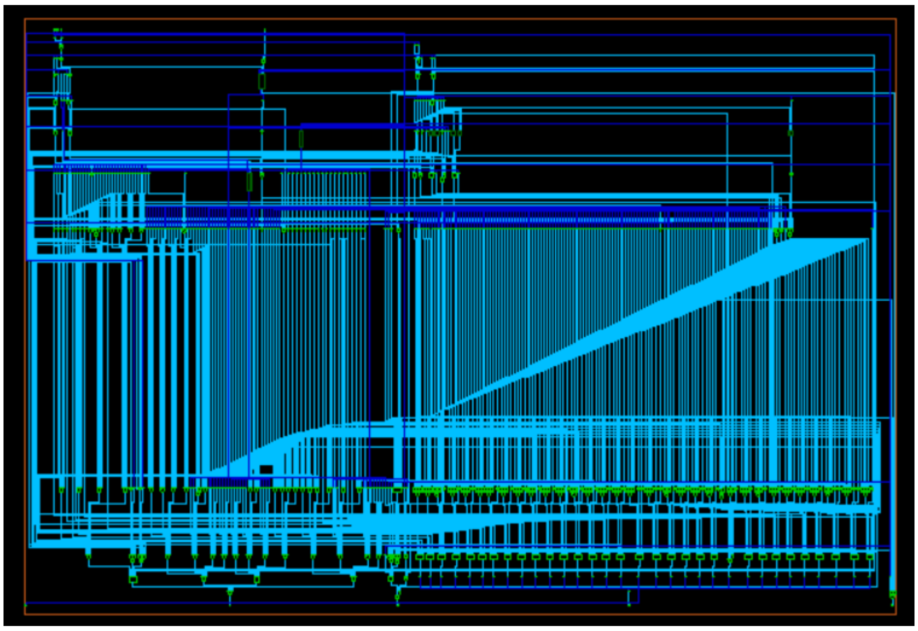
```
A = 00010001000100010001001011001111, B = 10000100010001001000010000111111,

A = 10010001000100010001001011001111, B = 00000100010001001000010000111111,

A = 10000000000000000000000000000000, B = 10000000000000000000000000000000,

A = 00000000000000000000000001111110, B = 00000000000000000000000011110100,

00000000000000000000000000001010
00000000000000000000000000001001

Wrong Answer Q_Q

A = 00000000000000000000000011110000, B = 00000000000000000000000011111111,

11111111111111111111111111110001
11111111111111111111111111110000

Wrong Answer Q_Q

A = 00000000000000000000000000000000, B = 00000000000000000000000011111111,

11111111111111111111111100000001
00000000000000000000000100000000

Wrong Answer Q_Q
```

7) NCVerilog 模擬結果



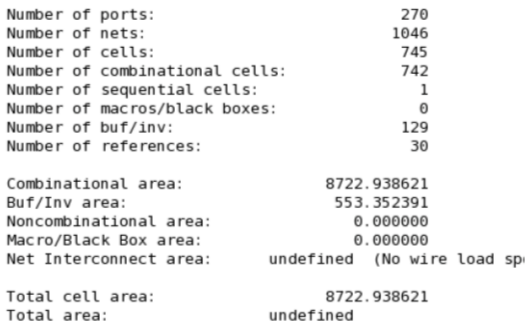
```
[dld0119@ic30 ~/lab2]$
[dld0119@ic30 ~/lab2]$ ncverilog ALU_tb.v ALU.v
ncverilog: 14.10-s005: (c) Copyright 1995-2014 Cadence Design Systems, Inc.
file: ALU_tb.v
        module worklib.tb:v
                errors: 0, warnings: 0
file: ALU.v
        module worklib.AND:v
                errors: 0, warnings: 0
        module worklib.OR:v
                errors: 0, warnings: 0
        module worklib.NOT:v
                errors: 0, warnings: 0
        module worklib.NOR:v
                errors: 0, warnings: 0
        module worklib.XOR:v
                errors: 0, warnings: 0
        module worklib.AdderSubtractor:v
                errors: 0, warnings: 0
        module worklib.ALU:v
                errors: 0, warnings: 0
                Caching library 'worklib' ....... Done
        Elaborating the design hierarchy:
        Building instance overlay tables: .................... Done
        Generating native compiled code:
                worklib.ALU:v <0x06c721ce>
                        streams:   3, words:  7738
                worklib.AdderSubtractor:v <0x1237df1a>
                        streams:   3, words:   998
                worklib.tb:v <0x7750715f>
                        streams:   7, words: 81160
        Building instance specific data structures.
        Loading native compiled code:     .................... Done
        Design hierarchy summary:
                                 Instances  Unique
                Modules:                8       8
                Primitives:            17       1
                Registers:             12      12
                Scalar wires:           9       -
                Expanded wires:        64       2
                Vectored wires:         4       -
                Always blocks:          2       2
                Initial blocks:         3       3
                Cont. assignments:      3       4
                Pseudo assignments:     6       6
                Simulation timescale:  1ps
        Writing initial simulation snapshot: worklib.tb:v
Loading snapshot worklib.tb:v .................... Done
*Verdi3* Loading libsscore_ius141.so
*Verdi3* : Enable Parallel Dumping.
ncsim> source /usr/cad/cadence/INCISIV/cur/tools/inca/files/ncsimrc
ncsim> run
No Bonus point 1/2

No Bonus points 2/2

score =         37 / 37  ps.without bonus points.
```

8) 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:



```
Number of ports:                          270
Number of nets:                          1046
Number of cells:                          745
Number of combinational cells:            742
Number of sequential cells:                 1
Number of macros/black boxes:               0
Number of buf/inv:                        129
Number of references:                      30

Combinational area:               8722.938621
Buf/Inv area:                      553.352391
Noncombinational area:               0.000000
Macro/Black Box area:                0.000000
Net Interconnect area:      undefined  (No wire load sp

Total cell area:                  8722.938621
Total area:                 undefined
```

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
```