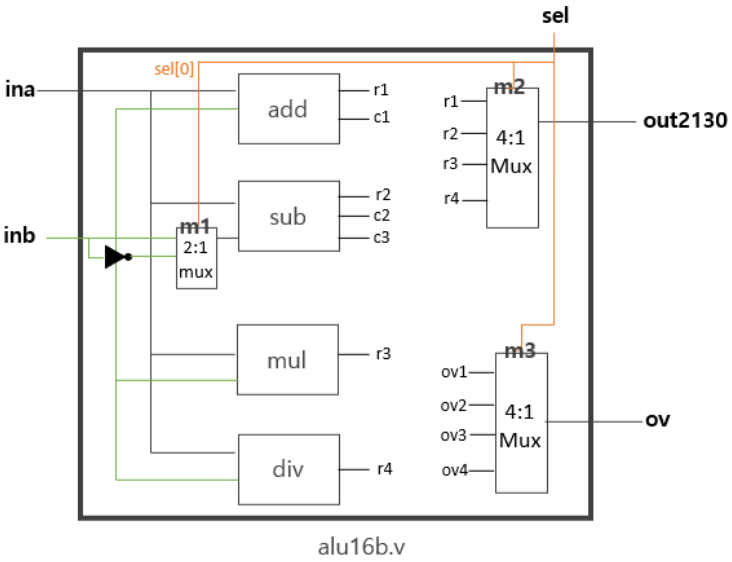
1. 전체 구조

[PORTS]

- 16-bit inputs : ina, inb
- 2-bit input : sel
- 16-bit output : out2130
- 1-bit output : ov

[WIRES]

- r1, r2, r3, r4 : 연산 결과
- ov1, ov2, ov3, ov4 : 오버플로우



alu16b.v

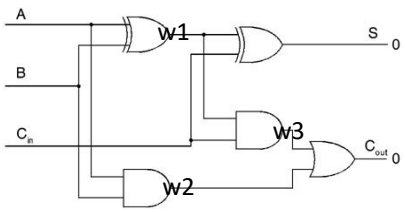
[설명]

adder / subtractor / multiplier / divider를 모듈로 구현하였습니다.
각 연산 결과와 오버플로우는 4:1 mux를 이용해 sel 값에 따라 달리 선택되도록 구현하였습니다.
Subtractor에 연결되는 입력값 b의 경우 2:1 mux를 이용해
sel[0]=1일 때, subtractor에 ~inb가 인가되어 2의 보수가 계산되도록 하였습니다.

2. Adder

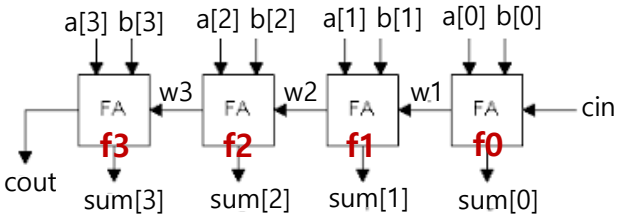
[설명] 그림과 같이 단순하게 (1-bit ≫ 4-bit ≫ 16-bit)로 합쳐지게 설계하였습니다

Step1) 1-bit full adder 설계



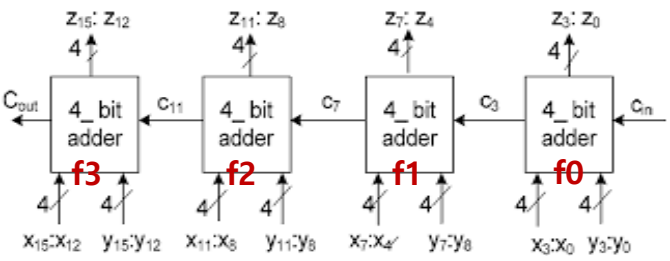
▲ fa1b.v

Step2) 1-bit full adder 4개를 연결하여
4-bit full adder 설계



▲ fa4b.v

Step3) 4-bit full adder 4개를 연결하여
16-bit full adder설계



▲ fa16b.v

3. Subtractor

[설명]

기존 Full Adder를 재활용하여 출력이 (a + (-b))가 되도록 설계하였습니다.

이때 sel 값이 2'b01이 되어 b의 2's complement를 계산해야 할 때

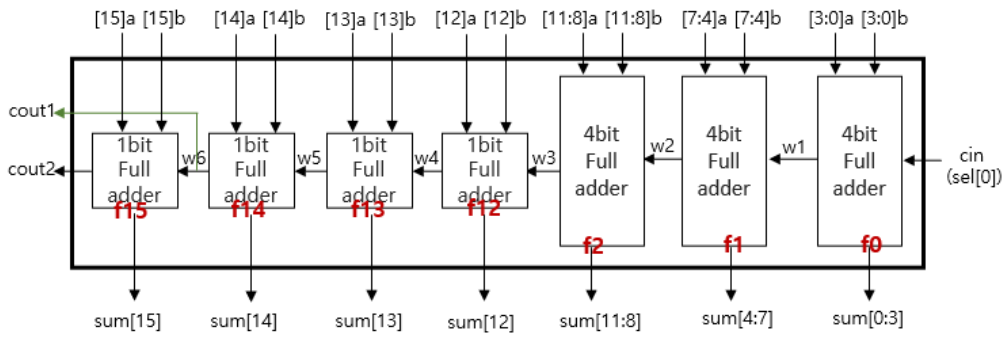
해당 subtractor내에서 ~b를 계산하도록 하지 않고,

top모듈에서 2:1 mux를 통해 sel 값에 따라 b또는 ~b가 subtractor에 입력되도록 하였습니다.

따라서 해당 subtractor 자체는 앞서 소개한 adder와 거의 같은 구조를 가지고 있습니다.

다만 top 모듈에서 mux가 추가로 연결되며, 추후에 signed overflow를 계산하려면

2개의 carry 정보가 필요하므로 MSB 4bit는 4개의 1-bit Full Adder를 사용하였습니다

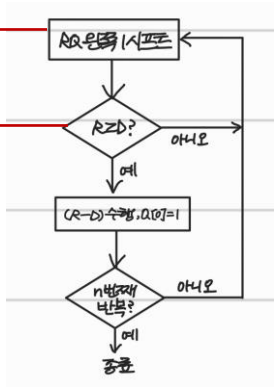


4. Divider

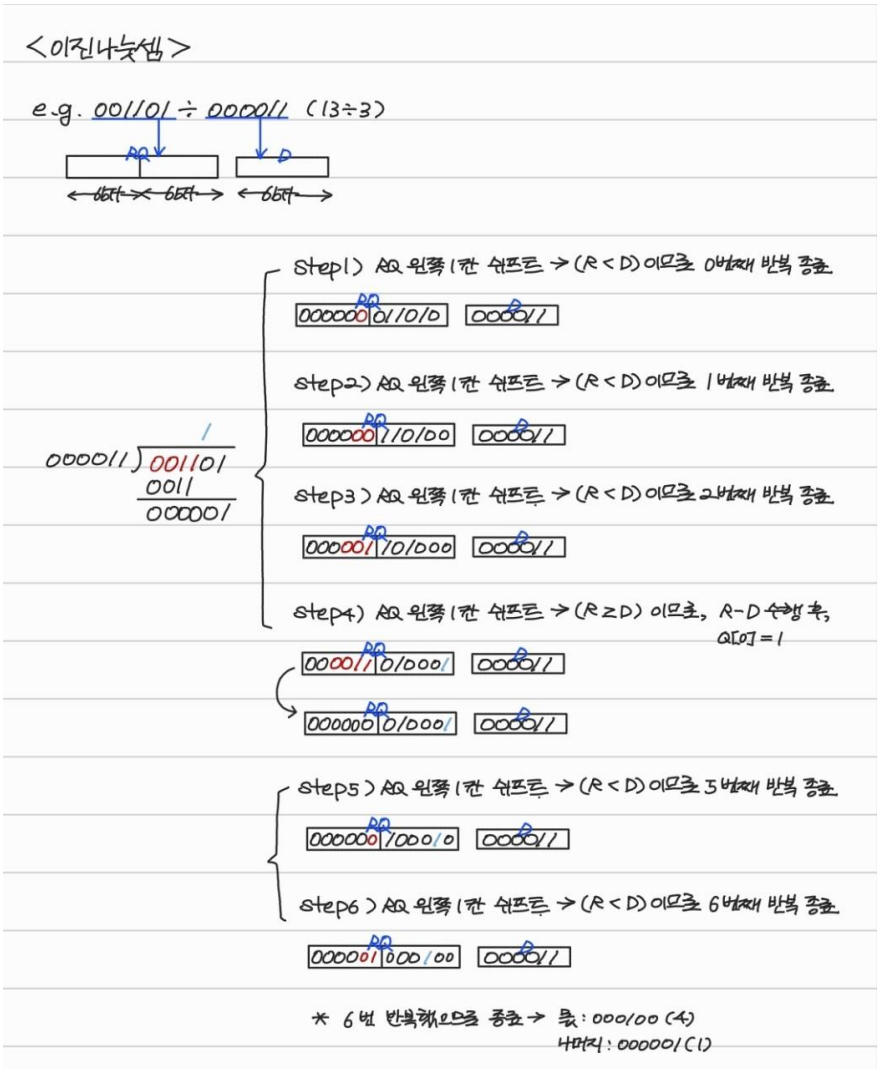
```
module div16b(divided,divisor,out);
  input [15:0] divided,divisor;
  output reg [31:0] out;

  reg [31:0] rq;
  integer i;

  always @(*) begin
    out = {16'b0, divided};
    for(i=0; i<16; i=i+1) begin
        out = out << 1;
        if (out[31:16] >= divisor) begin
          out[31:16] = out[31:16]-divisor;
          out[0] = 1'b1;
        end
    end
  end
endmodule
```



[설명] 해당 나눗셈기의 알고리즘은 "사람이 10진수 나눗셈을 하듯, 2진수에서도 나눗셈을 할 때 dividend에서 divisor만큼 차감해나간다" 라는 아이디어로부터 출발합니다.
이때 divisor를 dividend에서 빼기 위해서는 dividend가 divisor보다 커야 하므로,
알고리즘은 Dividend의 MSB부터LSB까지 훑으며 divisor(D)를 뺄 수 있는 위치(R≥D)를 탐색해나가며 위치를 찾게 되면 divisor를 차감하고, 다음 위치를 탐색하는 과정을 반복합니다. LSB까지 탐색하므로 n-bit 나눗셈에 대해 n번 반복 후 종료합니다. 아래의 사진은 이해를 돕기 위한 예입니다.



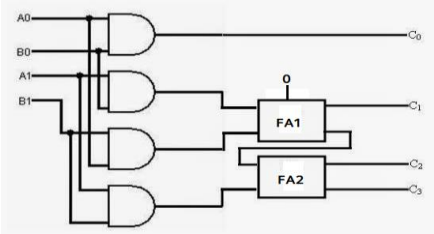
5. Multiplier

[설명]

Multiplier를 만들기 위한 가장 간단한 아이디어는 200개가 넘는 Full Adder를 직접 연결하는 것이지만, 코드가 길고 복잡합니다. 저는 코드를 간단하게 작성하기 위해 아래 그림과 같이 2-bit ≫ 4-bit ≫ 8-bit ≫ 16-bit로 확장하는 방법을 사용하였습니다 Step2) ~ Step4)의 그림을 관찰하면 사용하는 모듈의 입출력 크기만 달라질 뿐 구현 원리가 동일하여 보다 간단한 구현이 가능합니다.,

Step1) 2-bit Multiplier 설계

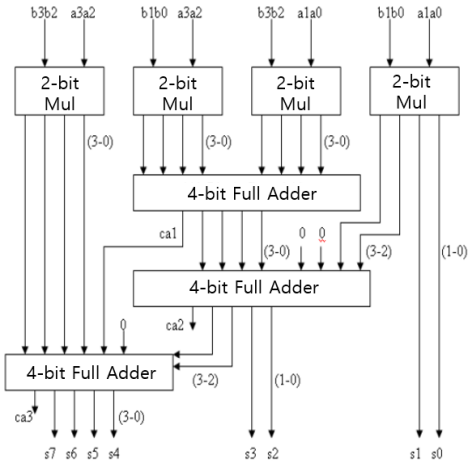
⇒ 단순한 Gate Level Design



▲ mul2b.v

Step2) 4-bit Multiplier 설계

⇒ 2-bit Multiplier & 4-bit Full Adder 재활용

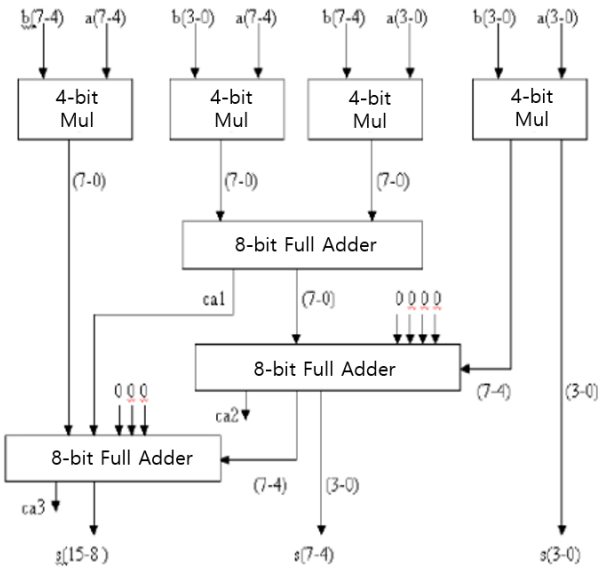


▲ mul4b.v

Step3) 8-bit Multiplier 설계

⇒ 4-bit Multiplier 재활용 & 8-bit Full Adder

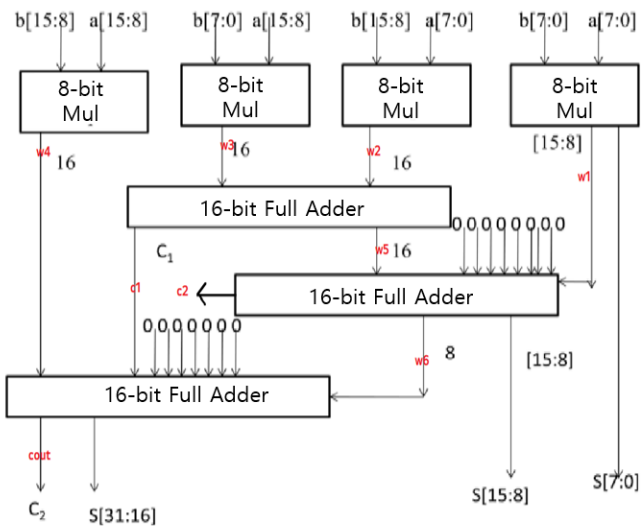
⇒ Multiplier의 구현을 위해 4-bit Full Adder를 이용해 8-bit Full Adder 작성(fa8b.v)



▲ mul8b.v

Step4) 16-bit Multiplier 설계

⇒ 8-bit Multiplier & 16-bit Full Adder 재활용



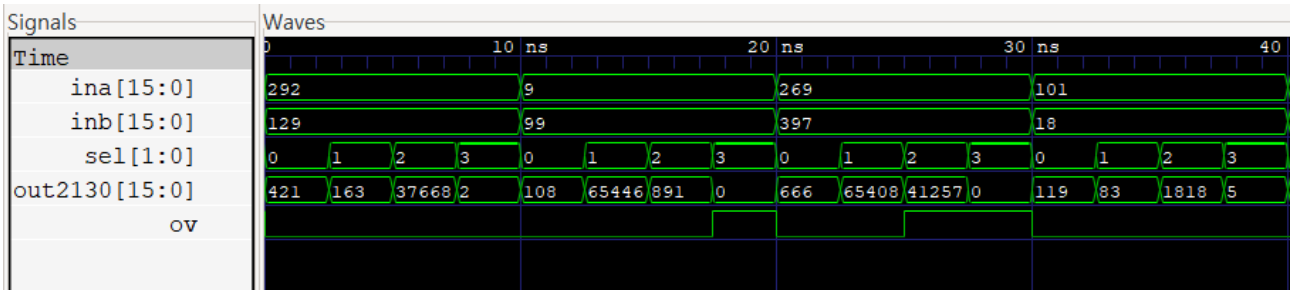
▲ mul16b.v

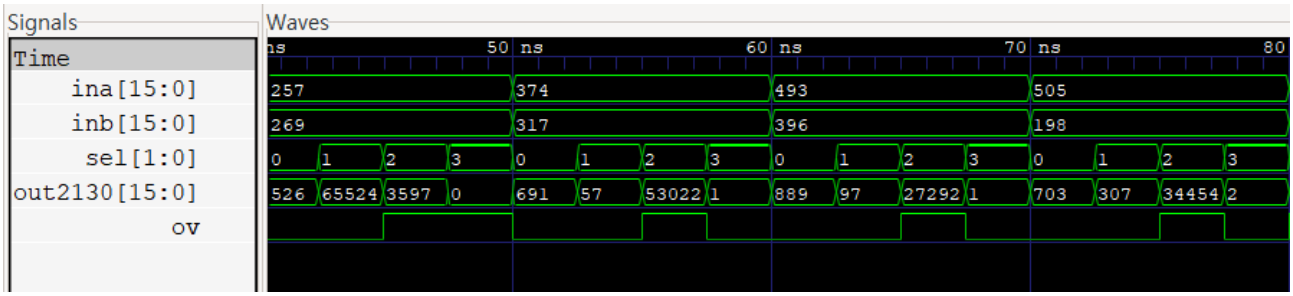
6. 테스트벤치 결과

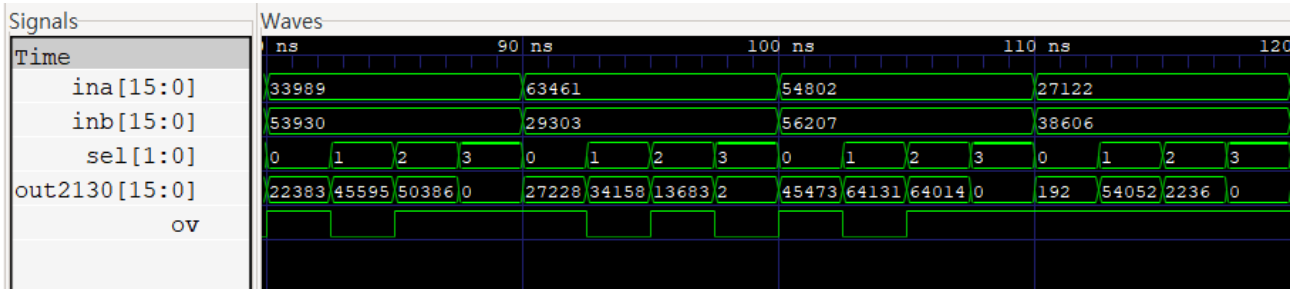
- Waveform

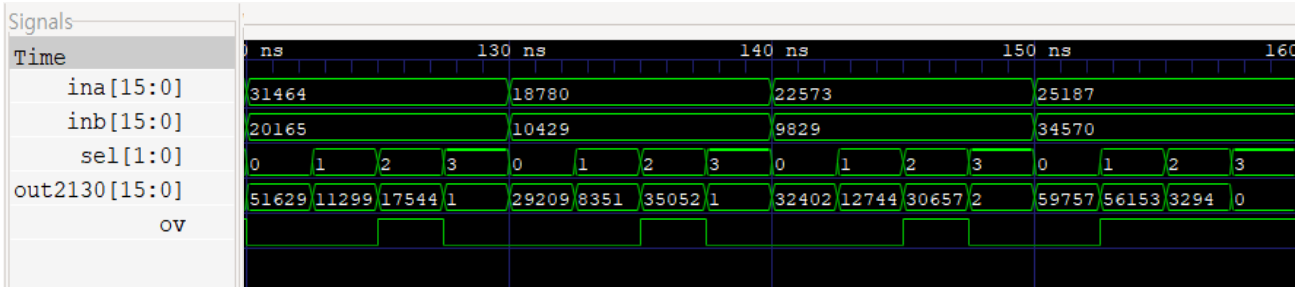
[설명] 1~8번째 : $urandom % 512;

9~16번째 : $urandom;









- **Adder :** 표현 범위(0~65535)를 벗어나는 경우, 기대했던 출력과는 다른 값이 나오는 동시에 오버플로우가 감지됨을 확인할 수 있다. 그 외의 경우에는 정확하게 계산된다

Input1	Input2	Correct Output	My Output	My Overflow
292	129	421	421	0
9	99	108	108	0
269	397	666	666	0
101	18	119	119	0
257	269	526	526	0
374	317	691	691	0
493	396	889	889	0
505	198	703	703	0
33989	53930	87919	22383	1
63461	29303	92764	27228	1
54802	56207	111009	45473	1
27122	38606	65728	192	1
31464	20165	51629	51629	0
18780	10429	29209	29209	0
22573	9829	32402	32402	0
25187	34570	59757	59757	0

- **Subtraction :** Subtraction의 경우, 각 입력값은 2's complement로 해석한다. 따라서 값의 표현 범위는 (-32,768~32,767)가 된다. 아래 표에서 음수값의 경우 괄호 안의 숫자에 표시하였다. 결과가 표현 범위를 벗어나는 경우, 오버플로가 발생함을 확인할 수 있다

Input1	Input2	Correct Output	My Output	My Overflow
292	129	163	163	0
9	99	(-90) 65446	65446	0
269	397	(-128) 65408	65408	0
101	18	83	83	0
257	269	(-12) 65524	65524	0
374	317	57	57	0
493	396	97	97	0
505	198	163	307	0
(-31547) 33989	(-11606) 53930	(-19941) 45595	45595	0
(-2075) 63461	29303	(-31378) 34158	34158	0
(-10734) 54802	(-9329) 56207	(-1405) 64131	64131	0
27122	(-26930) 38606	(+54052) 54052	54052	1
31464	20165	(-17019) 48517	11299	0
18780	10429	(8351) 8351	8351	0
22573	9829	(+12744) 12744	12744	0
25187	(-30966) 34570	(+56153) 56153	56153	1

- **Multiplication :** 표현 범위(0~65535)를 벗어나는 경우, 기대했던 출력과는 다른 값이 나오는 동시에 오버플로우가 감지된다. 곱셈의 경우, 입력 숫자의 범위를 줄였음에도 연산의 특징 때문에 쉽게 오버플로우가 발생함을 확인할 수 있다.

Input1	Input2	Correct Output	My Output	My Overflow
292	129	37668	37668	0
9	99	891	891	0
269	397	106,793	41257	1
101	18	1818	1818	0
257	269	69133	3597	1
374	317	118,558	53022	1
493	396	195,228	27292	1
505	198	99,990	34454	1
33989	53930	1,833,026,770	50386	1
63461	29303	1,859,597,683	13683	1
54802	56207	3,080,256,014	64014	1
27122	38606	1,047,071,932	2236	1
31464	20165	634,471,560	17544	1
18780	10429	195,856,620	35052	1
22573	9829	221,870,017	30657	1
25187	34570	870,714,590	3294	1

- **Division :** 몫이 0이 될 때 오버플로우가 감지된다.

Input1	Input2	Correct Output	My Output	My Overflow
292	129	2	2	0
9	99	0	0	1
269	397	0	0	1
101	18	5	5	0
257	269	0	0	1
374	317	1	1	0
493	396	1	1	0
505	198	2	2	0
33989	53930	0	0	1
63461	29303	2	2	0
54802	56207	0	0	1
27122	38606	0	0	1
31464	20165	1	1	0
18780	10429	1	1	0
22573	9829	2	2	0
25187	34570	0	0	1

7. 합성 결과 캡쳐

```
Writing verilog file '/home/KNUEEhdd1/comp311/comp10/work_dir/proj1/dc/alu16b.syn.v'.
Warning: Verilog 'assign' or 'tran' statements are written out. (VO-4)
Warning: Verilog writer has added 16 nets to module alu16b using SYNOPSYS_UNCONNECTED_ as
prefix.  Please use the change_names command to make the correct changes before invoking t
he verilog writer.  (VO-11)
1
write_sdc $currentDesign.sdc
1
report_qor
Information: Updating design information... (UID-85)

****************************************
Report : qor
Design : alu16b
Version: O-2018.06-SP4
Date   : Sat Apr 30 19:32:07 2022
****************************************


  Timing Path Group (none)
  -----------------------------------
  Levels of Logic:             195.00
  Critical Path Length:        392.48
  Critical Path Slack:         uninit
  Critical Path Clk Period:       n/a
  Total Negative Slack:          0.00
  No. of Violating Paths:        0.00
  Worst Hold Violation:          0.00
  Total Hold Violation:          0.00
  No. of Hold Violations:        0.00
  -----------------------------------


  Cell Count
  -----------------------------------
  Hierarchical Cell Count:        708
  Hierarchical Port Count:       6187
  Leaf Cell Count:               2938
  Buf/Inv Cell Count:             354
  Buf Cell Count:                   2
  Inv Cell Count:                 352
  CT Buf/Inv Cell Count:            0
  Combinational Cell Count:      2938
  Sequential Cell Count:            0
  Macro Count:                      0
  -----------------------------------


  Area
  -----------------------------------
  Combinational Area:     9195.946434
  Noncombinational Area:     0.000000
  Buf/Inv Area:            453.647042
  Total Buffer Area:             4.07
  Total Inverter Area:         449.58
  Macro/Black Box Area:      0.000000
  Net Area:               1775.220428
  -----------------------------------
  Cell Area:              9195.946434
  Design Area:           10971.166863


  Design Rules
  -----------------------------------
  Total Number of Nets:          3257
  Nets With Violations:             0
  Max Trans Violations:             0
  Max Cap Violations:               0
  -----------------------------------


  Hostname: knuee-srv2

  Compile CPU Statistics
  ----------------------------------------
  Resource Sharing:                   1.90
  Logic Optimization:                 3.45
  Mapping Optimization:               1.68
  ----------------------------------------
  Overall Compile Time:               9.16
  Overall Compile Wall Clock Time:    7.05

  --------------------------------------------------------------------

  Design  WNS: 0.00  TNS: 0.00  Number of Violating Paths: 0


  Design (Hold)  WNS: 0.00  TNS: 0.00  Number of Violating Paths: 0

  --------------------------------------------------------------------


1
exit

Thank you...
[comp10@knuee-srv2 dc]$
```