



컴퓨터공학 기초 실험2

Lab #4

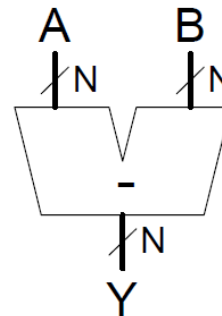
Subtractor & Arithmetic Logic Unit

SUBTRACTOR & ALU

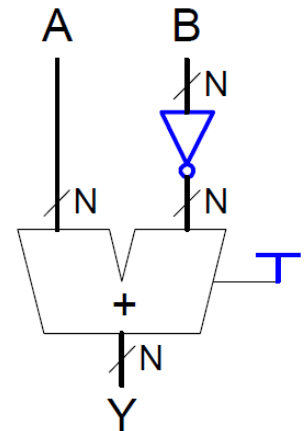
Subtractor

➤ Subtraction in Digital Circuits

- ✓ 대부분의 digital system에서는 subtraction을 위하여 2의 보수(2's complement)를 사용
 - 해당 입력을 invert한 후, 1을 더함
- ✓ 2's complement number 구하는 방법
 - $A - B = A + (-B)$
 - 이때, $-B$ 를 2's complement number로 변경
- ✓ 하나의 adder를 사용하여 덧셈, 뺄셈 연산 모두 수행 가능



Symbol



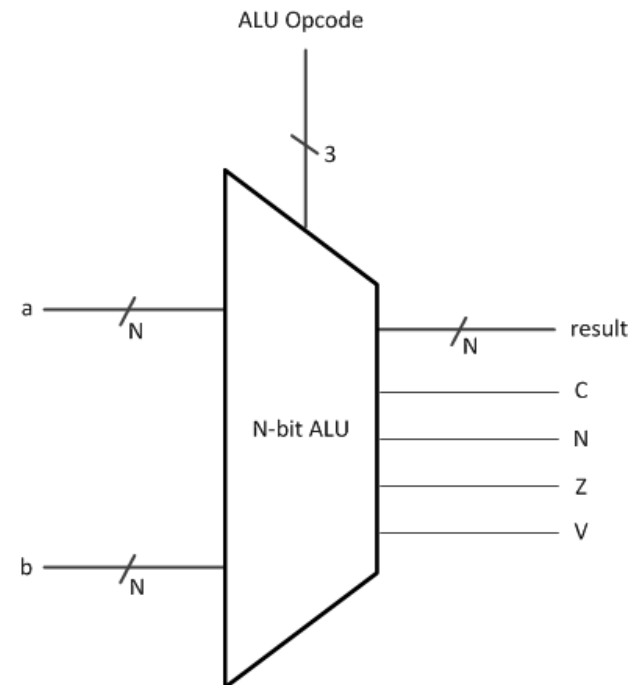
Implementation

Arithmetic Logic Unit

➤ Arithmetic Logic Unit(산술 논리 장치)

- ✓ Arithmetic Logic Unit(산술 논리 장치)는 두 숫자의 산술연산(덧셈, 뺄셈 등등)과 논리 연산(AND, OR, XOR, 등등)을 계산하는 디지털 회로이다.
- ✓ 본 실습에서는 Operator인 3-bit opcode에 따라 여러 연산을 수행

| Opcode | Operation |
|--------|---------------|
| 3'b000 | Not A |
| 3'b001 | Not B |
| 3'b010 | And |
| 3'b011 | Or |
| 3'b100 | Exclusive or |
| 3'b101 | Exclusive nor |
| 3'b110 | Addition |
| 3'b111 | Subtraction |



Symbol

Arithmetic Logic Unit(Cont.)

➤ ALU Status Flags

- ✓ 실습에서 구현하는 ALU는 총 4개의 flag를 갖는다.
- ✓ C : Carry
 - 연산결과 carry가 발생하는 경우
- ✓ N : Negative
 - 연산결과 sign bit가 1인 경우
- ✓ Z : Zero
 - 연산결과가 0인 경우
- ✓ V : Overflow
 - 연산결과 overflow가 발생한 경우($V = \text{carry}[n-1] \text{ exclusive or } \text{carry}[n-2]$)
- ✓ ALU의 status flag는 비교 연산을 하는 데 사용되어 질 수 있다.
 - $\text{Result} = A - B$
 - If $N == 1$, then $A < B$
 - Else If $Z == 1$, then $A == B$
 - Else $Z == 0$, then $A > B$

Today's objective

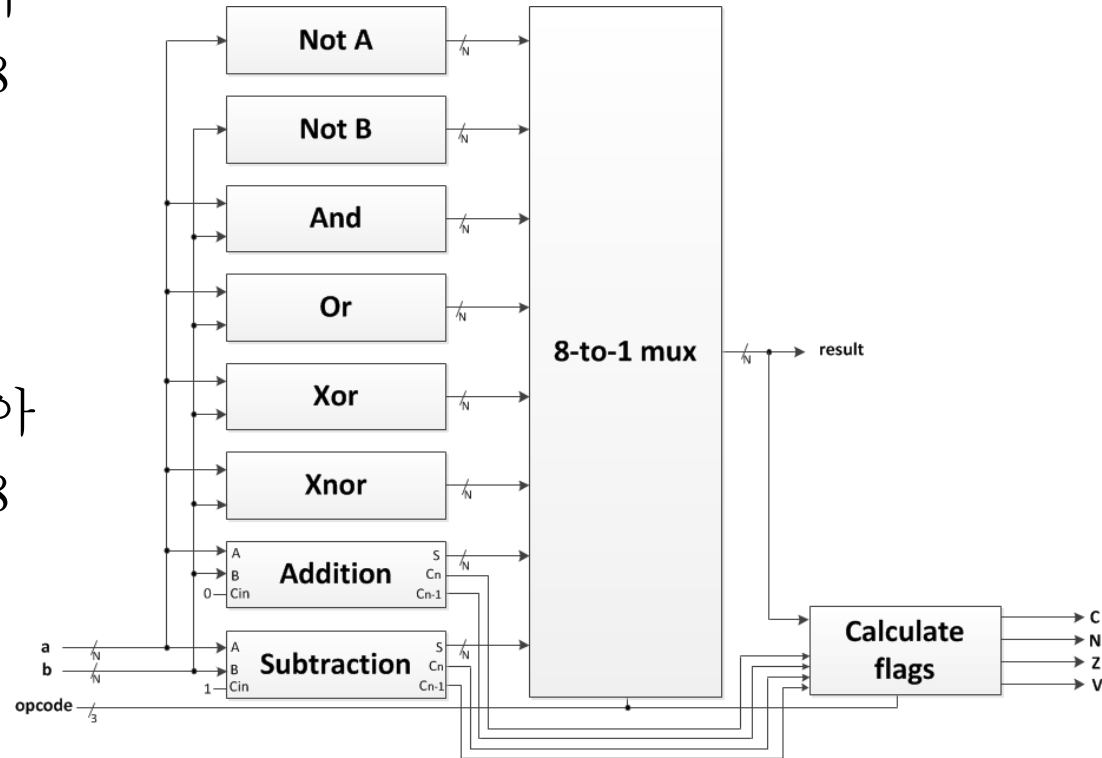
1. 4bits 산술논리 장치

- 4bits의 입력 (a, b) 을 받아 opcode (op)의 값에 따라 8가지의 연산을 시행한다.

2. 32bits 산술논리 장치

- 32bits의 입력 (a, b) 을 받아 opcode (op)의 값에 따라 8가지의 연산을 시행한다.

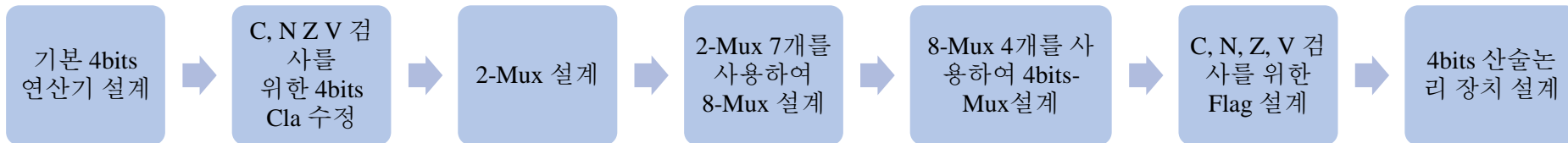
- C, N, Z, V 는 산술논리 장치를 검사하는 목적 으로 쓰이는 Output 이다



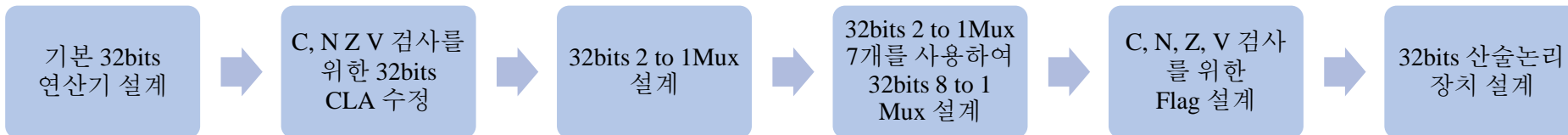
Today's objective

➤ Work flow chart

➤ 4bits 산술논리 장치.



➤ 32bits 산술논리 장치



4-bit Arithmetic Logic Unit

PRACTICE I

Arithmetic Logic Unit

➤ New Project Wizard

- ✓ Project name : alu4
- ✓ Family & Device : Cyclone V 5CSXFC6D6F31C6 (밑에서 6번째)

➤ Verilog file

- ✓ Add files : gates.v, fa_v2.v, clb4.v
- ✓ New files : cla4_ov.v, mx2.v, mx8.v, mx8_4bits.v,
cal_flags4.v, alu4.v, tb_alu4.v

➤ 파일을 추가할 때 해당 프로젝트 폴더에 복사하여 집어넣는다.

Basic Logic Gates (1/3)

➤ Inverter

```
module _inv(a,y);  
input a;  
output y;  
assign y=~a;  
endmodule
```

➤ 2-to-1 nand gate

```
module _nand2(a,b,y);  
input a,b;  
output y;  
assign y=~(a&b);  
endmodule
```

➤ 2-to-1 and gate

```
module _and2(a,b,y);  
input a,b;  
output y;  
assign y=a&b;  
endmodule
```

➤ 2-to-1 or gate

```
module _or2(a,b,y);  
input a,b;  
output y;  
assign y=a|b;  
endmodule
```

➤ 2-to-1 xor gate

```
module _xor2(a,b,y);  
input a, b;  
output y;  
wire inv_a, inv_b;  
wire w0, w1;  
_inv U0_inv(.a(a), .y(inv_a));  
_inv U1_inv(.a(b), .y(inv_b));  
_and2 U2_and2(.a(inv_a), .b(b), .y(w0));  
_and2 U3_and2(.a(a), .b(inv_b), .y(w1));  
_or2 U4_or2(.a(w0), .b(w1), .y(y));  
endmodule
```

(Lab2 – RCA 때 작성했던 logic gates로 **gates.v**에 작성하였다.)

Basic Logic Gates (2/3)

➤ 3-to-1 and gate

```
module _and3(a,b,c,y);  
input a,b,c;  
output y;  
assign y=a&b&c;  
endmodule
```

➤ 4-to-1 and gate

```
module _and4(a,b,c,d,y);  
input a,b,c,d;  
output y;  
assign y=a&b&c&d;  
endmodule
```

➤ 5-to-1 and gate

```
module _and5(a,b,c,d,e,y);  
input a,b,c,d,e;  
output y;  
assign y=a&b&c&d&e;  
endmodule
```

➤ 3-to-1 or gate

```
module _or3(a,b,c,y);  
input a,b,c;  
output y;  
assign y=a|b|c;  
endmodule
```

➤ 4-to-1 or gate

```
module _or4(a,b,c,d,y);  
input a,b,c,d;  
output y;  
assign y=a|b|c|d;  
endmodule
```

➤ 5-to-1 or gate

```
module _or5(a,b,c,d,e,y);  
input a,b,c,d,e;  
output y;  
assign y=a|b|c|d|e;  
endmodule
```

(Lab3 – CLA 때 작성했던 logic gates로 **gates.v**에 추가로 작성하였다.)

Basic Logic Gates (3/3)

➤ 4 bits inverter

```
module _inv_4bits(a,y);  
input [3:0]      a;  
output [3:0]     y;  
assign y=~a;  
endmodule
```

➤ 4 bits 2-to-1 and gate

```
module _and2_4bits(a,b,y);  
input [3:0]      a,b;  
output [3:0]     y;  
assign y=a&b;  
endmodule
```

➤ 4 bits 2-to-1 or gate

```
module _or2_4bits(a,b,y);  
input [3:0]      a,b;  
output [3:0]     y;  
assign y=a|b;  
endmodule
```

➤ 4 bits 2-to-1 exclusive or gate

```
module _xor2_4bits(a,b,y);  
input [3:0]      a,b;  
output [3:0]     y;  
_xor2 U0_xor2(.a(a[0]), .b(b[0]), .y(y[0]));  
_xor2 U1_xor2(.a(a[1]), .b(b[1]), .y(y[1]));  
_xor2 U2_xor2(.a(a[2]), .b(b[2]), .y(y[2]));  
_xor2 U3_xor2(.a(a[3]), .b(b[3]), .y(y[3]));  
endmodule
```

➤ 4 bits 2-to-1 exclusive nor gate

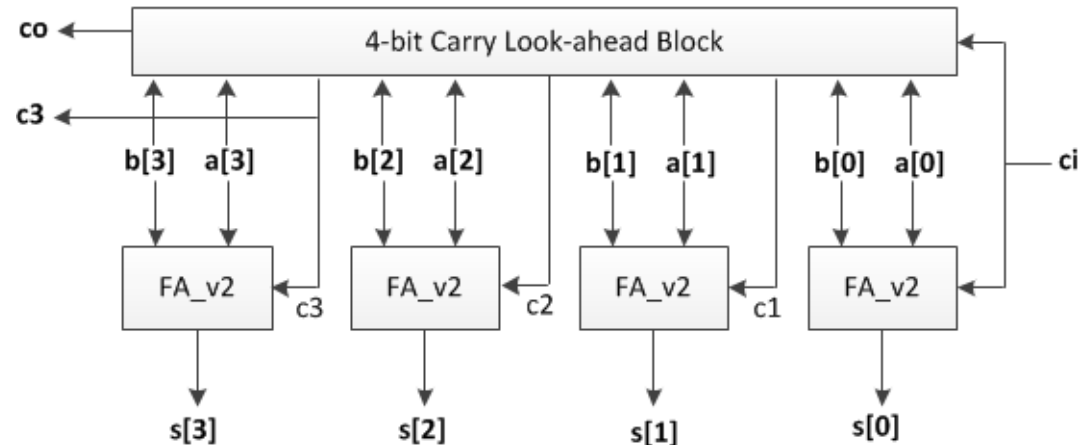
```
module _xnor2_4bits(a,b,y);  
input [3:0]      a,b;  
output [3:0]     y;  
wire [3:0]       w0;  
_xor2_4bits U0_xor2_4bits(.a(a), .b(b), .y(w0));  
_inv_4bits U1_inv_4bits(.a(w0), .y(y));  
endmodule
```

(Lab4 – SUB&ALU에 추가하는 logic gates로 **gates.v**에 추가로 작성한다.)

Modification of 4bits CLA

➤ 4-bits CLA 수정

- ✓ Overflow를 검출하기 위하여 상위 2개의 carry를 출력
- ✓ 지난 수업의 4-bit cla를 수정
- ✓ **File : cla4_ov.v**
- ✓ **Module : cla4_ov**



```
module cla4_ov(a, b, ci, s, c3, co);
    input [3:0] a, b;
    input ci;
    output [3:0] s;
    output c3, co;
```

Instances of fa_v2 and clb4

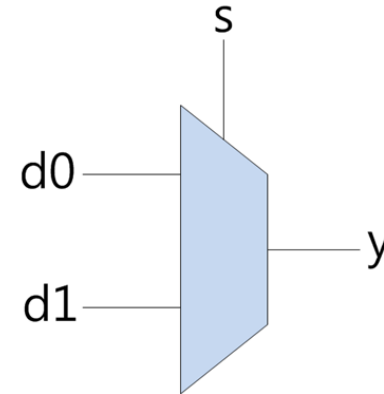
endmodule

1bit 2-to-1 Multiplexer

➤ gates.v에 있는 logic gates 중 inverter와 nand gate를 이용하여 multiplexer를 구현

✓ **File : mx2.v**

✓ **Module : mx2**



| Input | | | Output |
|-------|----|----|--------|
| s | d0 | d1 | y |
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 1 |
| 0 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 1 |
| 1 | 1 | 0 | 0 |
| 1 | 1 | 1 | 1 |

```
module mx2(d0, d1, s, y);  
  input d0, d1;  
  input      s;  
  output     y;  
  wire sb, w0, w1;
```

Instances of
inverter and nand gate

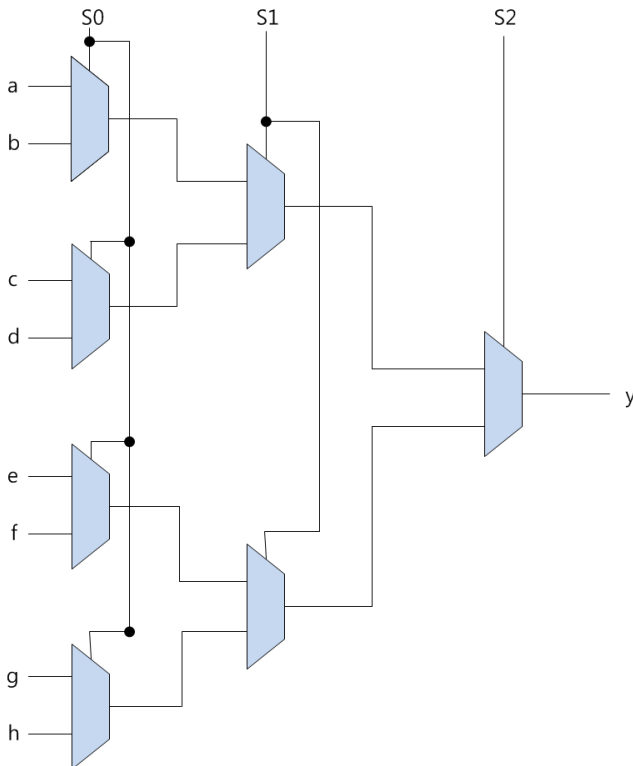
```
endmodule
```

1bit 8-to-1 Multiplexer

- 1bit 2-to-1 multiplexer를 instance하여 1bit 8-to-1 multiplexer를 구현

✓ **File : mx8.v**

✓ **Module : mx8**



```
module mx8(a, b, c, d, e, f, g, h, s2, s1, s0, y);  
  input  a, b, c, d, e, f, g, h;  
  input  s2, s1, s0;  
  output y;
```

```
  wire  w0, w1, w2, w3, w4, w5;
```

Instances of mx2

```
endmodule
```

4bits 8-to-1 Multiplexer

➤ 1bit 8-to-1 multiplexer를 4개 instance하여 4 bits 8-to-1 multiplexer 구현

✓ **File : mx8_4bits.v**

✓ **Module : mx8_4bits**

```
module mx8_4bits(y, a, b, c, d, e, f, g, h, s2, s1, s0);  
    input  [3:0]  a, b, c, d, e, f, g, h;  
    input          s2, s1, s0;  
    output [3:0]  y;
```

Instances of mx8

```
endmodule
```


Calculation of 4bits Flags

- Multiplexer의 결과와 adder/subtraction의 carry 값들을 받아 flag 들을 계산
 - ✓ **File : cal_flags4.v**
 - ✓ **Module : cal_flags4**
 - ✓ Flag 중 c, z, v는 삼항 연산자(conditionaloperator)를 사용하여 구현
 - ✓ Flag 중 n은 result의 MSB를 할당

```
module cal_flags4(op, result, co_add, c3_add, co_sub, c3_sub, c, n, z, v);  
  input  [2:0]  op;  
  input  [3:0]  result;  
  input        co_add, c3_add, co_sub, c3_sub;  
  output        c, n, z, v;
```

Assign (Conditional Operator)

참고, conditional operator

➤ Assign **Variable** = (**condition**) ? **True** : **False**

✓ Example

```
assign c=(op[2:1]!=2'b11)? 1'b0:((op[0]==1'b0)?co_add:co_sub);
```

Verilog

```
if( !( op[2:1] = 2'b11 ) ){
    c = 1'b0
}
else{
    if( op[0] == 1'b0 ){ c = co_add }
    else{ c = co_sub }
}
```

C language

➤ Assign $c = (op[2:1] \neq 2'b11) ? 1'b0 : ((op[0] == 1'b0) ? co_add : co_sub);$ 의 의미

| Op[2] | Op[1] | Op[0] | Target | Return | Reference |
|-------|-------|-------|-------------|--------|------------------|
| 1 | 1 | 0 | Addition | Co_add | Addition 의 Co |
| 1 | 1 | 1 | Subtraction | Co_sub | Subtraciton 의 Co |

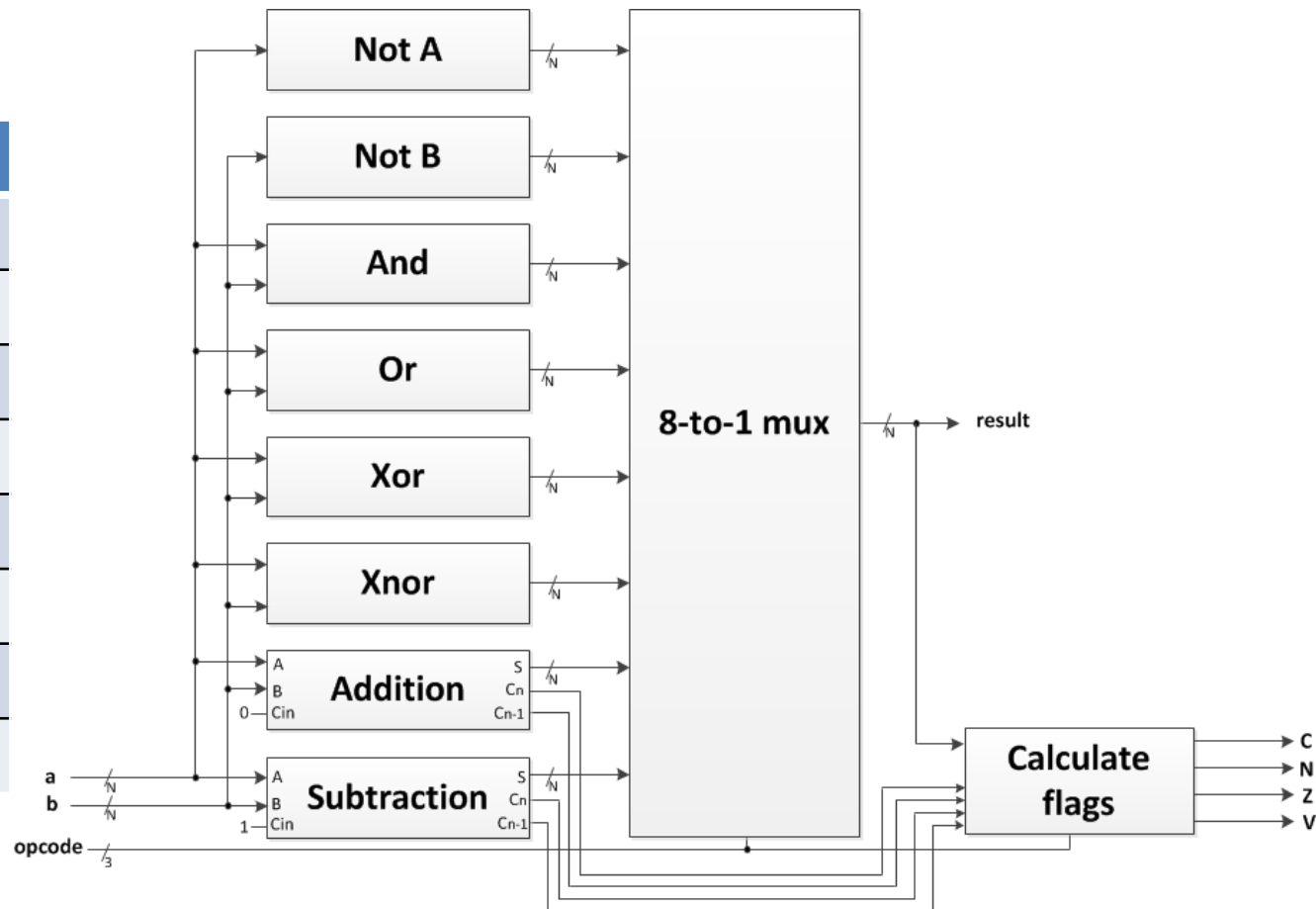
4bits Arithmetic Logic Unit (1/2)

➤ 4bits Arithmetic Logic Unit

✓ **File : alu4.v**

✓ **Module : alu4**

| Opcode | Operation |
|--------|---------------|
| 3'b000 | Not A |
| 3'b001 | Not B |
| 3'b010 | And |
| 3'b011 | Or |
| 3'b100 | Exclusive or |
| 3'b101 | Exclusive nor |
| 3'b110 | Addition |
| 3'b111 | Subtraction |



4bits Arithmetic Logic Unit (2/2)

```
module alu4(a, b, op, result, c, n, z, v);  
  input  [3:0]      a, b;  
  input  [2:0]      op;  
  output [3:0]      result;  
  output           c, n, z, v;  
  wire  [3:0]      w_not_a, w_not_b, w_and, w_or, w_xor, w_xnor, w_add, w_sub;  
  wire           c3_add, co_add, c3_sub, co_sub;
```

```
_inv_4bits  
_inv_4bits  
_and2_4bits  
_or2_4bits  
_xor2_4bits  
_xnor2_4bits  
cla4_ov  
cla4_ov  
mx8_4bits  
  
cal_flags4
```

Instance

```
endmodule
```

Testbench of ALU

```
`timescale 1ns/100ps
module tb_alu4;
    reg    [3:0]      tb_a, tb_b;
    reg    [2:0]      tb_op;
    wire   [3:0]      tb_result;
    wire      tb_c, tb_n, tb_z, tb_v;
    alu4 U0_alu4(.a(tb_a), .b(tb_b), .op(tb_op), .result(tb_result),
                .c(tb_c), .n(tb_n), .z(tb_z), .v(tb_v));
    initial begin
```

Assign (Conditional Operator)

```
end
endmodule
```

Functional Simulation

➤ Waveform

✓ Logical operation(binary display)

| | | | | | | | | | | | | |
|----------------------|------|------|------|------|------|------|------|------|------|-----|--|--|
| ◆ /alu4_tb/tb_a | 0011 | 0000 | 1100 | | | 0101 | | | 0011 | | | |
| ◆ /alu4_tb/tb_b | 0101 | 0000 | | | 0011 | 1001 | 1010 | 0101 | | | | |
| ◆ /alu4_tb/tb_op | 100 | 000 | | | 001 | 010 | 011 | 100 | | 101 | | |
| ◆ /alu4_tb/tb_result | 0110 | 1111 | 0011 | 1100 | 0001 | 1111 | 0110 | | 1001 | | | |
| ◆ /alu4_tb/tb_c | St0 | | | | | | | | | | | |
| ◆ /alu4_tb/tb_n | St0 | ESAL | | | | | | | | | | |
| ◆ /alu4_tb/tb_z | St0 | | | | | | | | | | | |
| ◆ /alu4_tb/tb_v | St0 | | | | | | | | | | | |

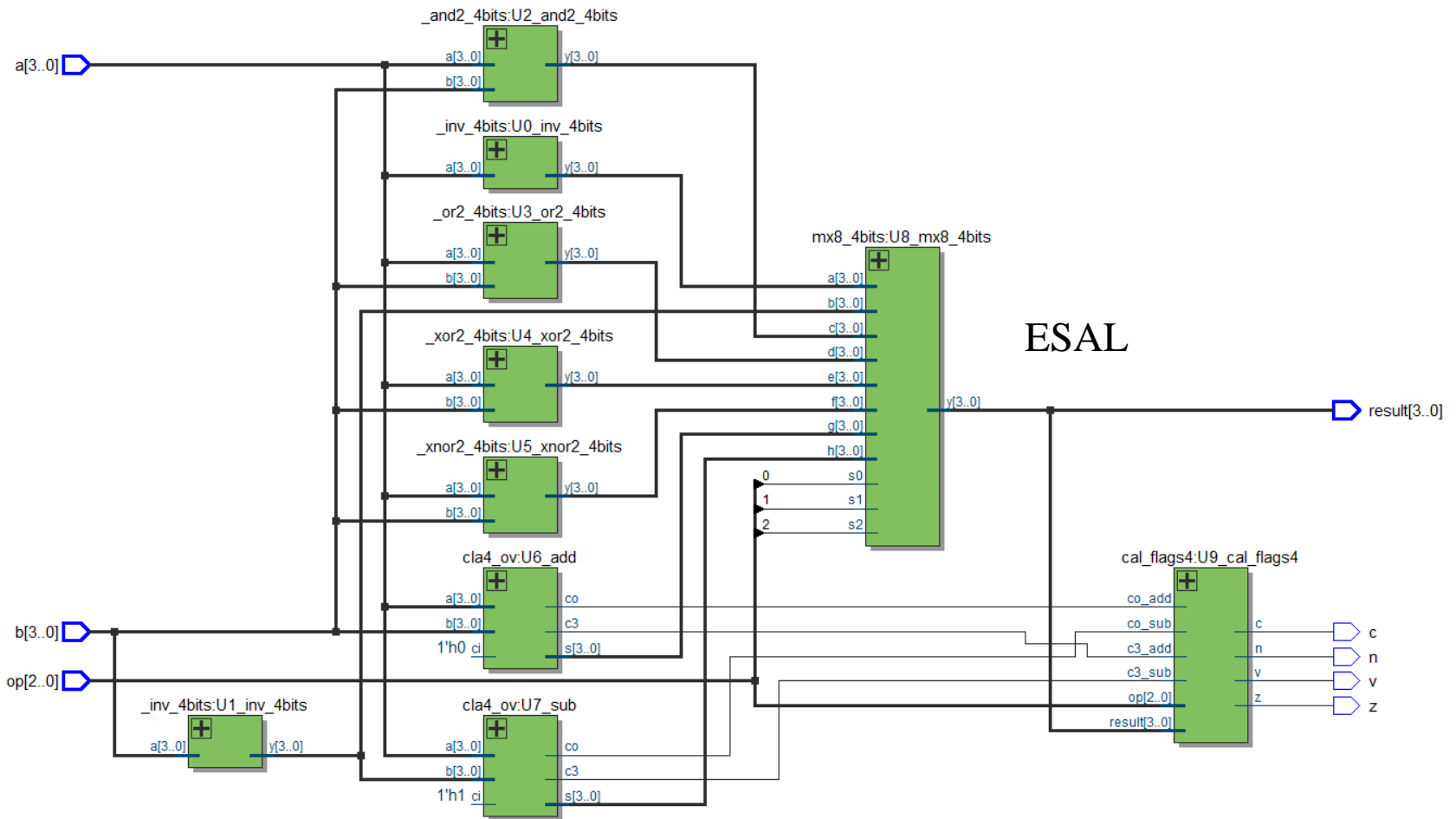
✓ Arithmetic operation(hexadecimal and decimal display)

| | | | | | | | | | | | | |
|----------------------|-----|------|---|---|---|-----|---|---|--|--|--|--|
| ◆ /alu4_tb/tb_a | a | a | 1 | 7 | 3 | f | 5 | a | | | | |
| ◆ /alu4_tb/tb_b | a | 3 | 8 | 7 | 3 | 5 | 7 | a | | | | |
| ◆ /alu4_tb/tb_op | 111 | 110 | | | | 111 | | | | | | |
| ◆ /alu4_tb/tb_result | 0 | d | 9 | e | 6 | a | e | 0 | | | | |
| ◆ /alu4_tb/tb_c | St1 | | | | | | | | | | | |
| ◆ /alu4_tb/tb_n | St0 | ESAL | | | | | | | | | | |
| ◆ /alu4_tb/tb_z | St1 | | | | | | | | | | | |
| ◆ /alu4_tb/tb_v | St0 | | | | | | | | | | | |

| | | | | | | | | | | | | |
|----------------------|-----|------|----|----|---|-----|----|----|--|--|--|--|
| ◆ /alu4_tb/tb_a | 3 | -6 | 1 | 7 | 3 | -1 | 5 | -6 | | | | |
| ◆ /alu4_tb/tb_b | 3 | 3 | -8 | 7 | 3 | 5 | 7 | -6 | | | | |
| ◆ /alu4_tb/tb_op | 110 | 110 | | | | 111 | | | | | | |
| ◆ /alu4_tb/tb_result | 6 | -3 | -7 | -2 | 6 | -6 | -2 | 0 | | | | |
| ◆ /alu4_tb/tb_c | St0 | | | | | | | | | | | |
| ◆ /alu4_tb/tb_n | St0 | ESAL | | | | | | | | | | |
| ◆ /alu4_tb/tb_z | St0 | | | | | | | | | | | |
| ◆ /alu4_tb/tb_v | St0 | | | | | | | | | | | |

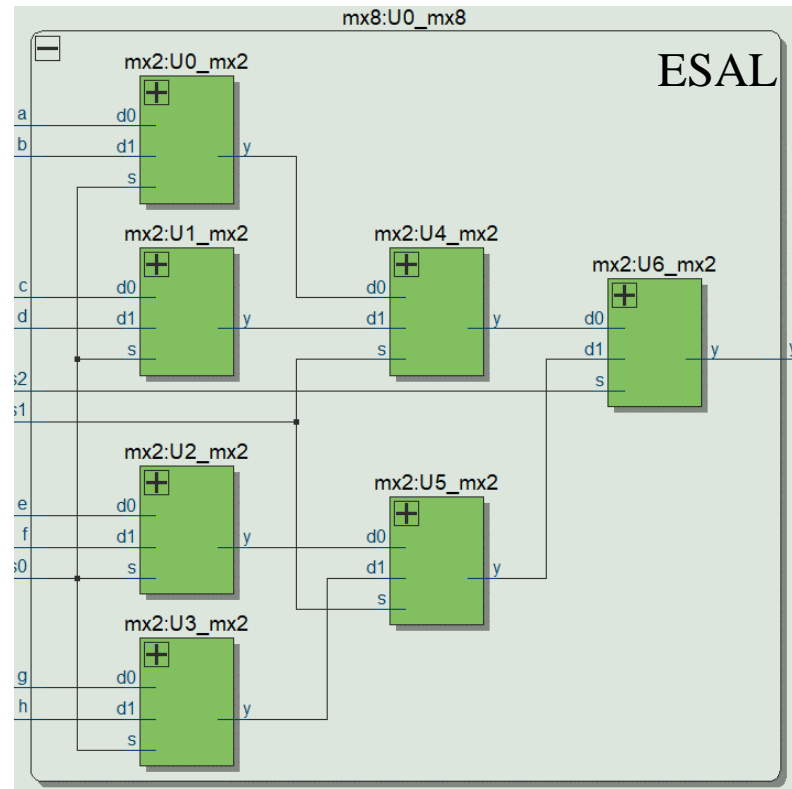
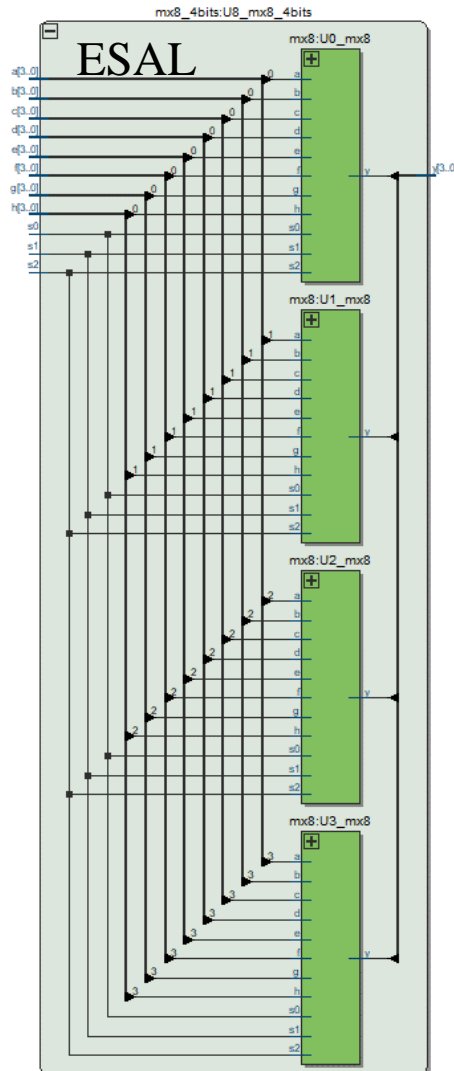
RTL Viewer (1/2)

➤ RTL viewer of ALU4



RTL Viewer (2/2)

➤ RTL viewer of mx8_4bits



Compilation Report

➤ Flow Summary

| Flow Summary | |
|---------------------------------|---|
| Flow Status | Successful - [REDACTED] |
| Quartus Prime Version | 15.1.0 Build 185 10/21/2015 SJ Lite Edition |
| Revision Name | [REDACTED] |
| Top-level Entity Name | alu4 |
| Family | Cyclone V |
| Device | 5CSXFC6D6F31C6 |
| Timing Models | Final |
| Logic utilization (in ALMs) | 12 / 41,910 (< 1 %) |
| Total registers | 0 |
| Total pins | 19 / 499 (4 %) |
| Total virtual pins | 0 |
| Total block memory bits | 0 / 5,662,720 (0 %) |
| Total DSP Blocks | 0 / 112 (0 %) |
| Total HSSI RX PCSs | 0 / 9 (0 %) |
| Total HSSI PMA RX Deserializers | 0 / 9 (0 %) |
| Total HSSI TX PCSs | 0 / 9 (0 %) |
| Total HSSI PMA TX Serializers | 0 / 9 (0 %) |
| Total PLLs | 0 / 15 (0 %) |
| Total DLLs | 0 / 4 (0 %) |

32-bit Arithmetic Logic Unit

PRACTICE II

Arithmetic Logic Unit

➤ New Project Wizard

- ✓ Project name : alu32
- ✓ Family & Device : Cyclone V 5CSXFC6D6F31C6 (밑에서 6번째)

➤ Verilog file

- ✓ Add files : gates.v, fa_v2.v, clb4.v, cla4.v, cla4_ov.v
- ✓ New file : cla32_ov.v, mx2_32bits.v, mx8_32bits.v, cal_flags32.v
alu32.v, tb_alu32.v

Basic Logic Gates (1/3)

➤ 32 bits inverter

```
module _inv_32bits(a,y);  
input [31:0]      a;  
output [31:0]     y;  
assign y=~a;  
endmodule
```

➤ 32 bits 2-to-1 and gate

```
module _and2_32bits(a,b,y);  
input [31:0]      a,b;  
output [31:0]     y;  
assign y=a&b;  
endmodule
```

➤ 32 bits 2-to-1 or gate

```
module _or2_32bits(a,b,y);  
input [31:0]      a,b;  
output [31:0]     y;  
assign y=a|b;  
endmodule
```

(Lab4 – SUB&ALU에 추가하는 logic gates로 **gates.v**에 추가로 작성한다.)

Basic Logic Gates (2/3)

➤ 32 bits exclusive or gate

```
module _xor2_32bits(a,b,y);  
input [31:0] a,b;  
output [31:0] y;  
  
_xor2_4bits U0_xor2_4bits(.a(a[3:0]), .b(b[3:0]), .y(y[3:0]));  
_xor2_4bits U1_xor2_4bits(.a(a[7:4]), .b(b[7:4]), .y(y[7:4]));  
_xor2_4bits U2_xor2_4bits(.a(a[11:8]), .b(b[11:8]), .y(y[11:8]));  
_xor2_4bits U3_xor2_4bits(.a(a[15:12]), .b(b[15:12]), .y(y[15:12]));  
_xor2_4bits U4_xor2_4bits(.a(a[19:16]), .b(b[19:16]), .y(y[19:16]));  
_xor2_4bits U5_xor2_4bits(.a(a[23:20]), .b(b[23:20]), .y(y[23:20]));  
_xor2_4bits U6_xor2_4bits(.a(a[27:24]), .b(b[27:24]), .y(y[27:24]));  
_xor2_4bits U7_xor2_4bits(.a(a[31:28]), .b(b[31:28]), .y(y[31:28]));  
  
endmodule
```

(Lab4 – SUB&ALU에 추가하는 logic gates로 **gates.v**에 추가로 작성한다.)

Basic Logic Gates (3/3)

➤ 32 bits exclusive nor gate

```
module _xnor2_32bits(a,b,y);  
input [31:0]  a,b;  
output [31:0] y;  
  
_xnor2_4bits U0_xnor2_4bits(.a(a[3:0]),    .b(b[3:0]),    .y(y[3:0]));  
_xnor2_4bits U1_xnor2_4bits(.a(a[7:4]),    .b(b[7:4]),    .y(y[7:4]));  
_xnor2_4bits U2_xnor2_4bits(.a(a[11:8]), .b(b[11:8]),    .y(y[11:8]));  
_xnor2_4bits U3_xnor2_4bits(.a(a[15:12]), .b(b[15:12]), .y(y[15:12]));  
_xnor2_4bits U4_xnor2_4bits(.a(a[19:16]), .b(b[19:16]), .y(y[19:16]));  
_xnor2_4bits U5_xnor2_4bits(.a(a[23:20]), .b(b[23:20]), .y(y[23:20]));  
_xnor2_4bits U6_xnor2_4bits(.a(a[27:24]), .b(b[27:24]), .y(y[27:24]));  
_xnor2_4bits U7_xnor2_4bits(.a(a[31:28]), .b(b[31:28]), .y(y[31:28]));  
  
endmodule
```

(Lab4 – SUB&ALU에 추가하는 logic gates로 **gates.v**에 추가로 작성한다.)

Modification of 32bits CLA

➤ 32-bit cla 수정

- ✓ 앞선 4-bit cla의 overflow를 검출하기 위해 수정했던 것처럼 32-bit cla도 수정
- ✓ **File : cla32_ov.v**
- ✓ **Module : cla32_ov**
 - 일반 4-bit cla 7개와 4-bit cla_ov 1개를 instance해서 구현

```
module cla32_ov(a, b, ci, s, co_prev, co);  
  input [31:0] a, b;  
  input ci;  
  output [31:0] s;  
  output co_prev;  
  output co;
```

```
  wire c1, c2, c3, c4, c5, c6, c7;
```

Instance

```
endmodule
```

32bits 8-to-1 Multiplexer

➤ 32bits 2-to-1 multiplexer

- ✓ **File : mx2_32bits.v**
- ✓ **Module : mx2_32bits**
- ✓ 삼항 연산자(conditional operator)
를 사용하여 구현

➤ 32bits 8-to-1 multiplexer

- ✓ **File : mx8_32bits.v**
- ✓ **Module : mx8_32bits**
- ✓ mx_32bits를 instance하여 구현

```
module mx2_32bits(d0, d1, s, y);  
  input  [31:0]  d0, d1;  
  input          s;  
  output [31:0]  y;
```

Conditional Operator

```
endmodule
```

```
module mx8_32bits(a, b, c, d, e, f, g, h, s2, s1, s0, y);  
  input  [31:0]  a, b, c, d, e, f, g, h;  
  input          s2, s1, s0;  
  output [31:0]  y;  
  
  wire          [31:0]  w0, w1, w2, w3, w4, w5;
```

Instance

```
endmodule
```


Calculation of 32bits Flags

- 앞선 alu4에서 했던 방식과 유사하게 구현

```
module cal_flags32(op, result, co_add, co_prev_add, co_sub, co_prev_sub, c, n, z, v);  
  input [2:0]  op;  
  input [31:0] result;  
  input      co_add, co_prev_add, co_sub, co_prev_sub;  
  output      c, n, z, v;
```

Assign (Conditional Operator)

```
endmodule
```

32bits Arithmetic Logic Unit

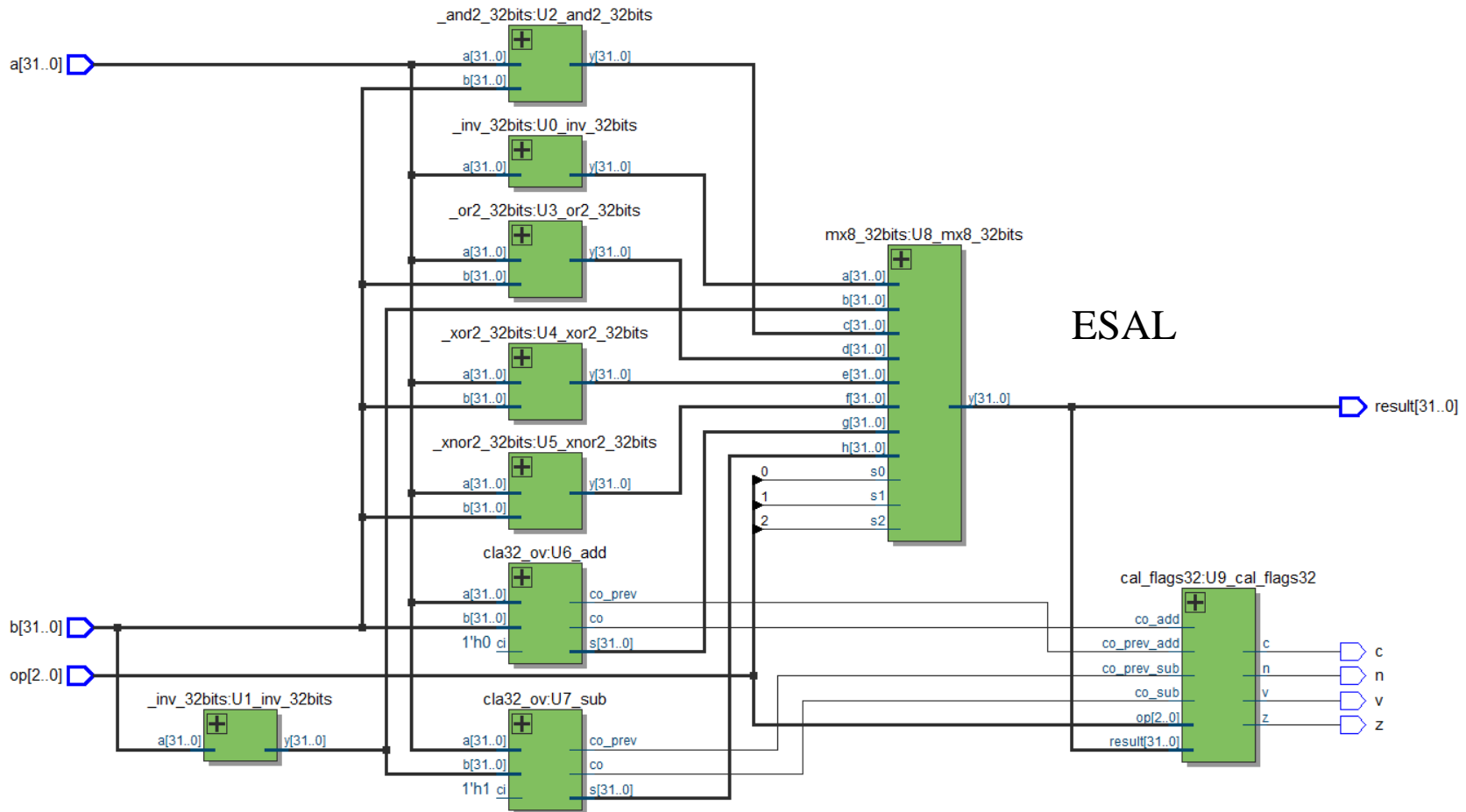
➤ 32-bit ALU

```
module alu32(a, b, op, result, c, n, z, v);  
  input  [31:0]  a, b;  
  input  [2:0]   op;  
  output [31:0]  result;  
  output        c, n, z, v;  
  
  wire  [31:0]  w_not_a, w_not_b, w_and, w_or, w_xor, w_xnor, w_add, w_sub;  
  wire        co_prev_add, co_add, co_prev_sub, co_sub;  
  
  _inv_32bits  
  _inv_32bits  
  _and2_32bits  
  _or2_32bits  
  _xor2_32bits  
  _xnor2_32bits  
  cla32_ov  
  cla32_ov  
  mx8_32bits  
  
  cal_flags32  
  
endmodule
```

Instance

RTL Viewer

➤ RTL Viewer



Compilation Report

➤ Flow Summary

| Flow Summary | |
|---------------------------------|---|
| Flow Status | Successful - [REDACTED] |
| Quartus Prime Version | 15.1.0 Build 185 10/21/2015 SJ Lite Edition |
| Revision Name | [REDACTED] |
| Top-level Entity Name | alu32 |
| Family | Cyclone V |
| Device | 5CSXFC6D6F31C6 |
| Timing Models | Final |
| Logic utilization (in ALMs) | 126 / 41,910 (< 1 %) |
| Total registers | 0 |
| Total pins | 103 / 499 (21 %) |
| Total virtual pins | 0 |
| Total block memory bits | 0 / 5,662,720 (0 %) |
| Total DSP Blocks | 0 / 112 (0 %) |
| Total HSSI RX PCSs | 0 / 9 (0 %) |
| Total HSSI PMA RX Deserializers | 0 / 9 (0 %) |
| Total HSSI TX PCSs | 0 / 9 (0 %) |
| Total HSSI PMA TX Serializers | 0 / 9 (0 %) |
| Total PLLs | 0 / 15 (0 %) |
| Total DLLs | 0 / 4 (0 %) |

Assignment 3

➤ Report

- ✓ 자세한 사항은 lab document 참고

➤ Submission

- ✓ Soft copy
 - 월요일 반 : 10월 5일 밤 23시 59분까지
 - 금요일 반 : 10월 9일 밤 23시 59분까지
- ✓ 늦은 숙제는 제출 이틀 후 까지만 받음(20% 감점)

채점기준

| 세부사항 | | 점수 | 최상 | 상 | 중 | 하 | 최하 |
|---------------|--|-----|----|----|----|---|----|
| 소스코드 | Source code가 잘 작성 되었는가? (Structural design으로 작성되었는가?) | 10 | 10 | 8 | 5 | 3 | 0 |
| | 주석을 적절히 달았는가? (반드시 영어로 주석 작성) | 20 | 20 | 15 | 10 | 5 | 0 |
| 설계검증 (보고서) | 보고서를 성실히 작성하였는가? (보고서 형식에 맞추어 작성) | 30 | 30 | 20 | 10 | 5 | 0 |
| | 합성결과를 설명하였는가? | 10 | 10 | 8 | 5 | 3 | 0 |
| | 검증을 제대로 수행하였는가? (모든 입력 조합, waveform 설명) | 30 | 30 | 20 | 10 | 5 | 0 |
| 총점 | | 100 | | | | | |

References

- Altera Co., www.altera.com/
- 이준환, 디지털논리회로2 강의자료, 광운대학교, 컴퓨터 공학과, 2020

Q&A

THANK YOU