

Arithmetic Logic Unit

Aim of the Experiment:

Implement a

16 bit ALU

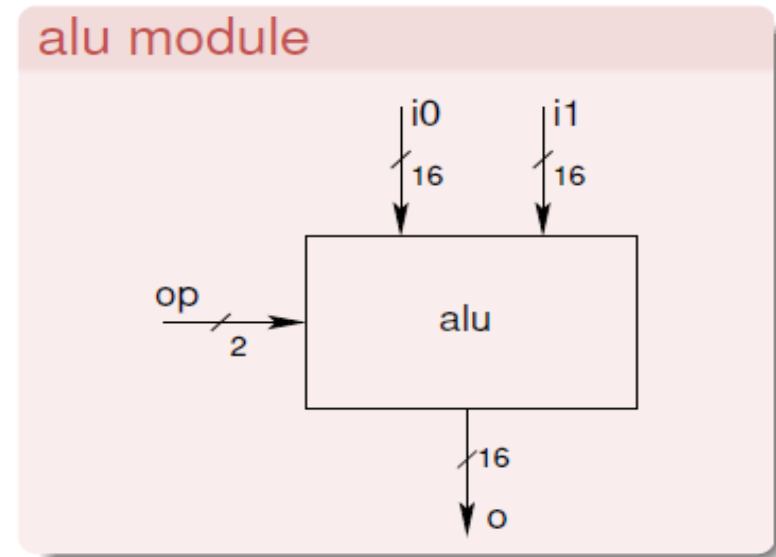
supporting:

Arithmetic operations :

add, subtract

Logic operations :

and, or



The ALU (Arithmetic and Logic Unit) is the heart of a CPU since it performs the arithmetic and logic operations which are the key computations.

Your task in this assignment is to design and simulate a 16-bit¹ ALU which performs the following core operations

- Arithmetic Addition and subtraction (two's complement)
- Logic AND along with OR operation

Each operation would receive as input two 16-bit operands and the output would be a 16-bit result (along with carry and overflow).

The decision on which of the four operations is to be performed would be indicated by a two bit operation input. So the ALU module inputs are:

- i0[15:0] 16-bit operand 0
- i1[15:0] 16-bit operand 1
- op[1:0] 2-bit operation code

Operation Code meaning

| op[1:0] | operation |
|---------|-------------|
| 00 | Addition |
| 01 | Subtraction |
| 10 | and |
| 11 | or |

The ALU module outputs should be:

- **o[15:0]** 16-bit operation result
- **carry** Carry out of MSB
- **overflow** Arithmetic overflow

During logic operations the values of the carry and overflow outputs are irrelevant.

The ALU module inputs are:

| | | |
|-------------------------------|-------------------------------|--|
| i0[15:0] 16-bit operand i0 | i1[15:0] 16-bit operand i1 | |
|-------------------------------|-------------------------------|--|

The opcode bits are op1,op0(2 bit operation code)

| | | |
|----|----------|--|
| 00 | add | |
| 01 | subtract | |
| 10 | and | |
| 11 | or | |

The ALU module outputs are:

| | | |
|---------------------------------------|--|--|
| o[15:0] 16-bit operation result | | |
|---------------------------------------|--|--|

The status bits are

| | | |
|---------------------------|------------------------------------|--|
| Carry Carry out of MSB | Overflow Arithmetic overflow | |
|---------------------------|------------------------------------|--|

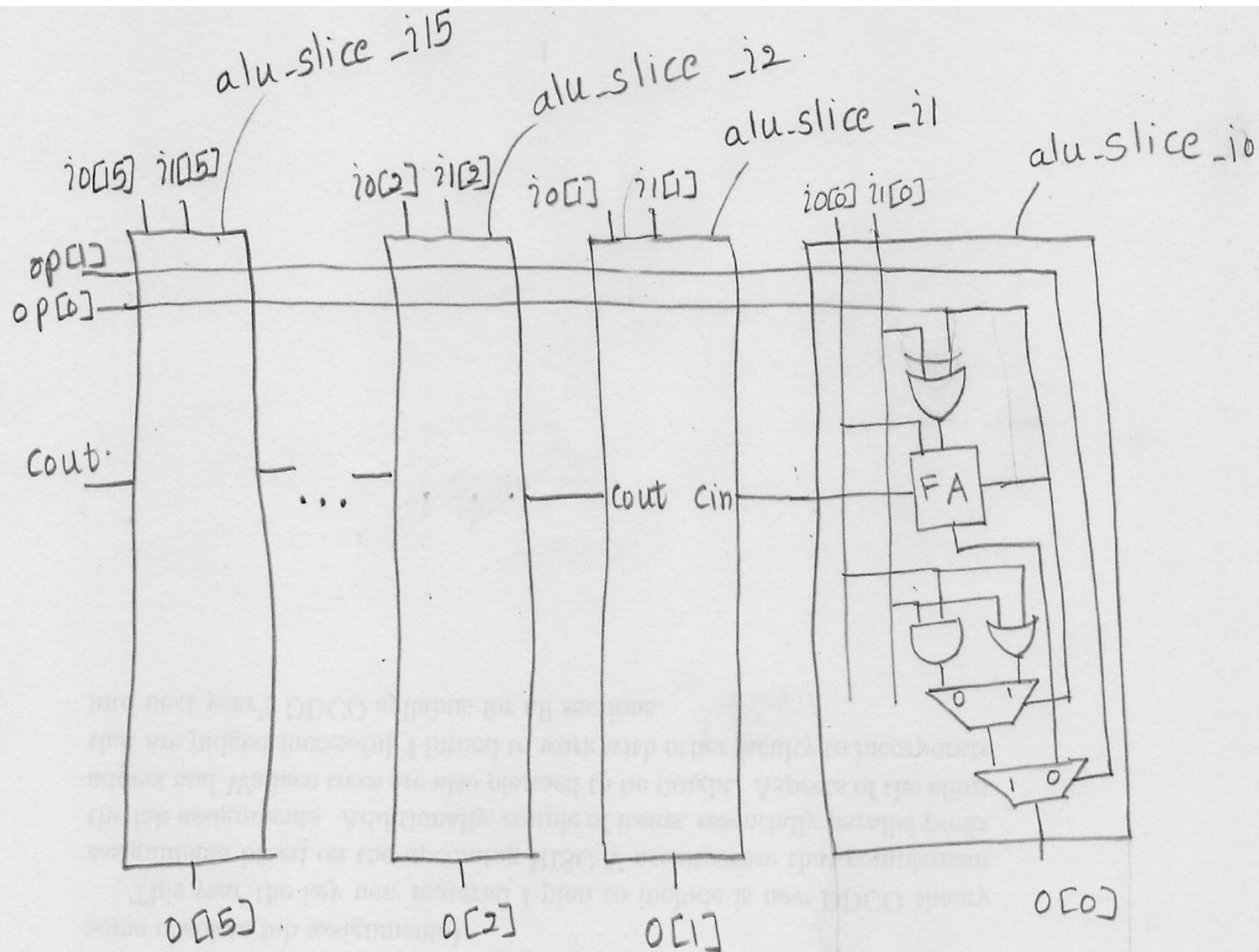
- Once you have designed the ALU logic, it needs to be specified using the Verilog syntax .
- The entire ALU should be composed of purely combinational logic elements that have been discussed in class.
- Also, only the Verilog module definition and instantiation syntax discussed in class should be used.
- Once the ALU is designed, it needs to be simulated to verify proper functionality of its various operations.
- Appropriate inputs need to be applied to the ALU.

The inputs are specified as test vectors, four of which are specified as samples on lines 14 to 17 of tb_alu.v.

The samples test only the arithmetic functionality.

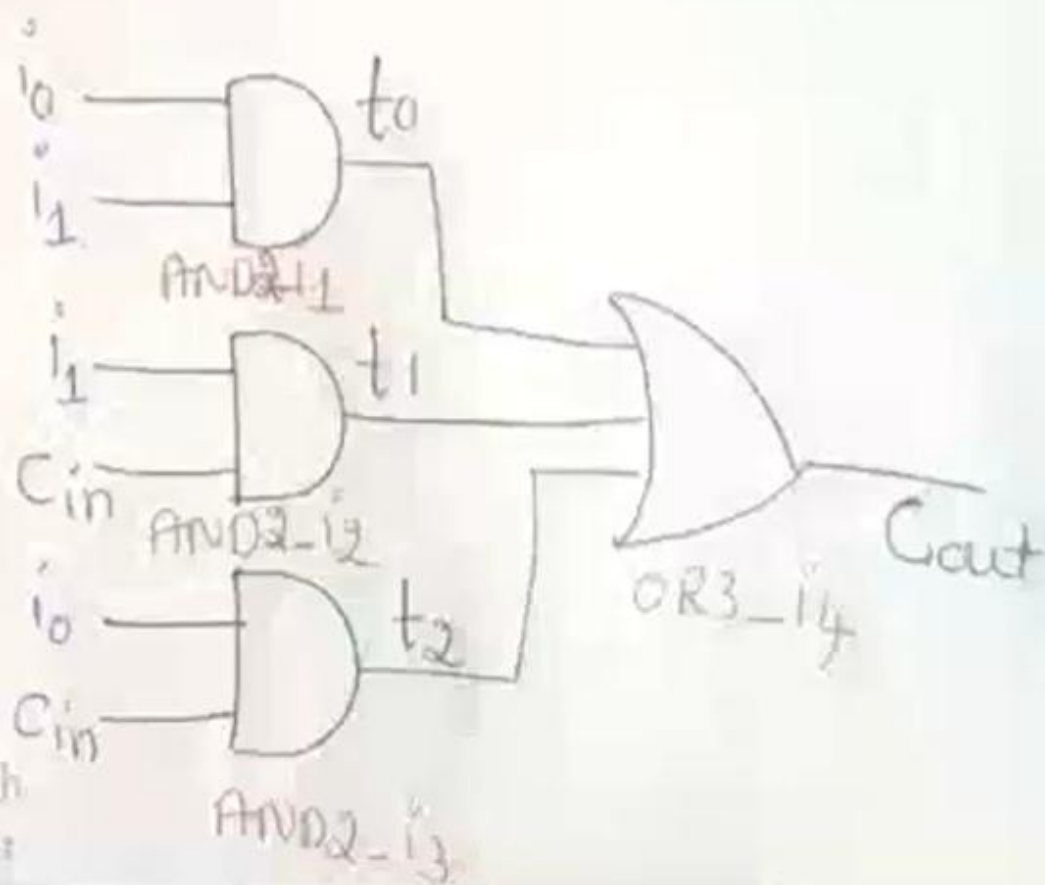
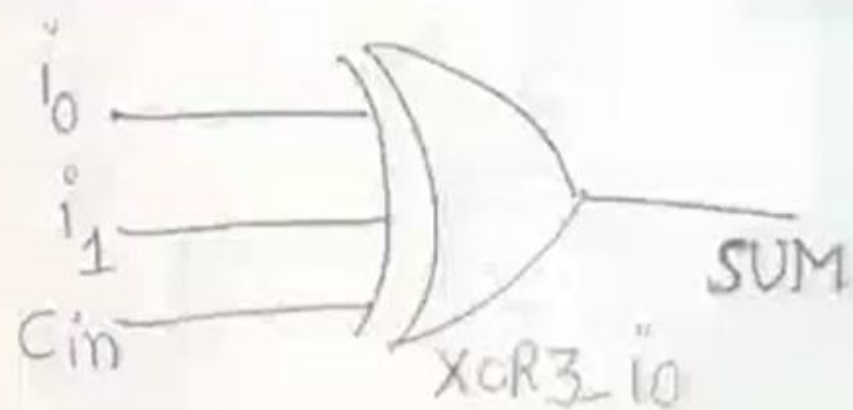
Additional test vectors need to be added not only for logic but arithmetic operations as well.

Be sure to change value of TESTVECS on line 2 whenever the number of test vectors is modified.

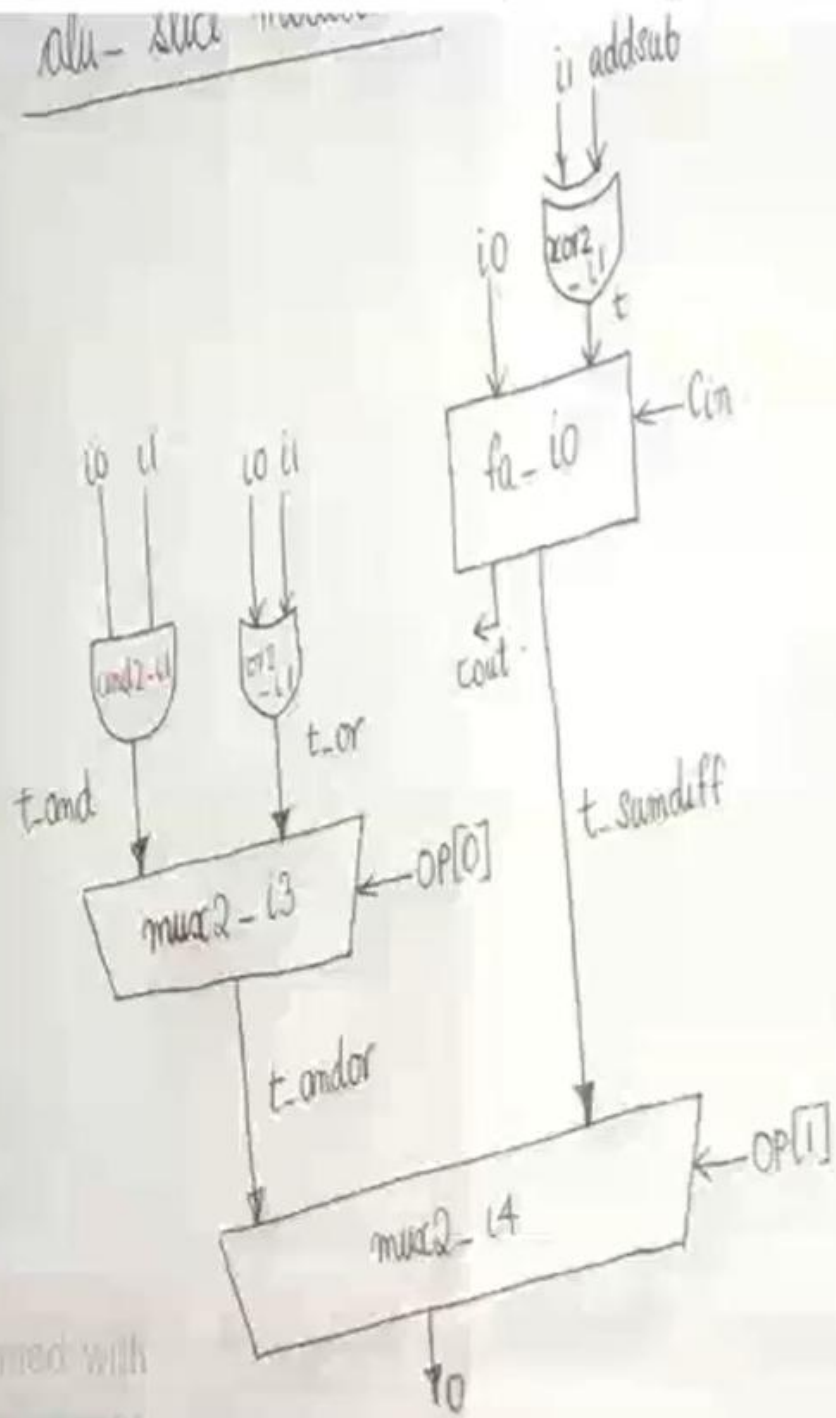


Source File alu.v

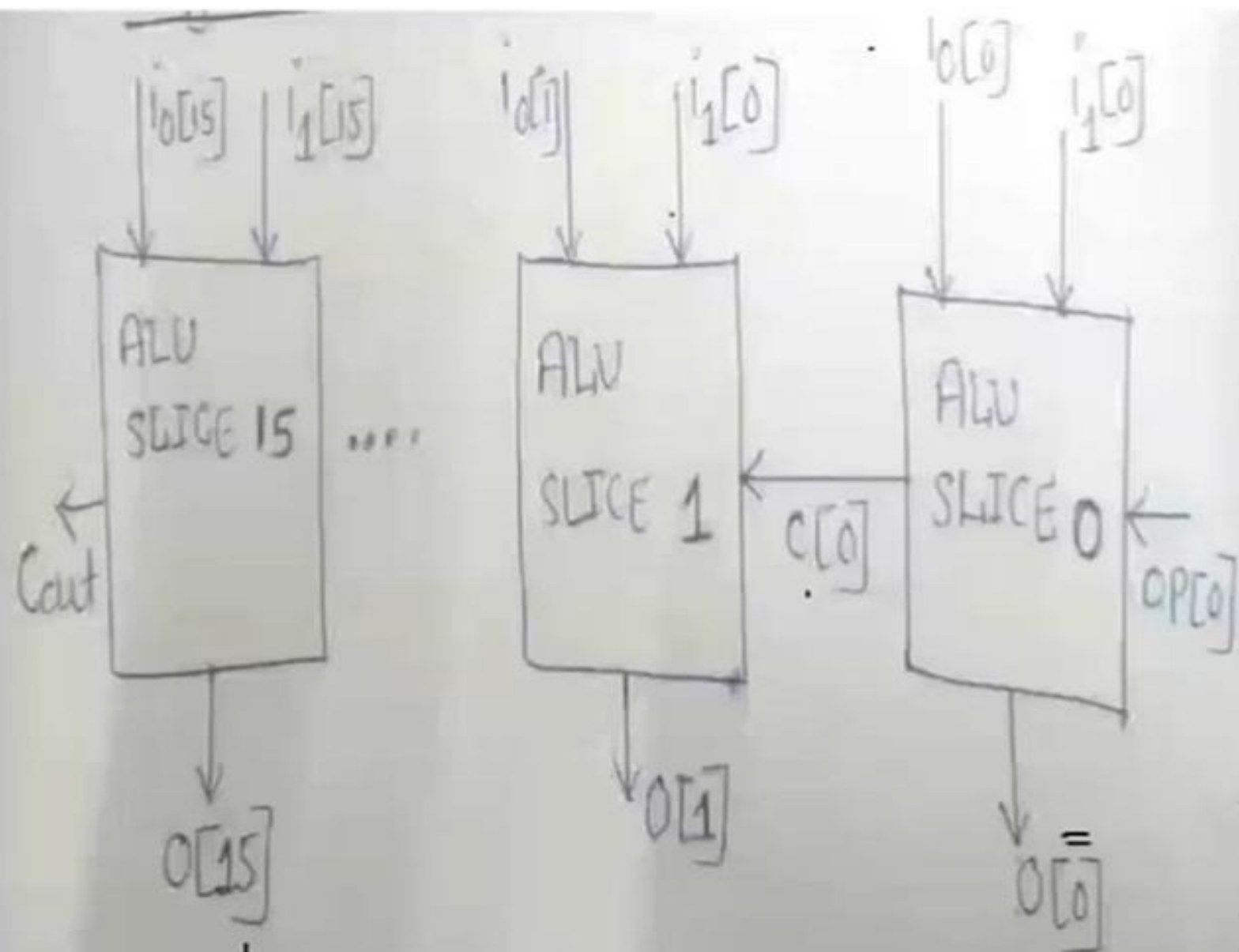
```
module fa (input wire i0, i1, cin, output wire sum, cout);  
  wire t0, t1, t2;  
  xor3 _i0 (-----);  
  and2 _i1 (-----);  
    and2 _i2 (-----);  
  and2 _i3 (-----);  
  or3 _i4 (-----);  
endmodule  
  
module addsub (input wire addsub, i0, i1, cin, output wire  
sumdiff, cout);  
  wire t;  
  fa _i0 (-----);  
  xor2 _i1 (-----);endmodule
```

alu - slice



```
module alu_slice (input wire [1:0] op, input wire i0, i1,  
cin, output wire o, cout);  
  wire t_sumdiff, t_and, t_or, t_andor;  
  addsub _i0 (-----);  
  and2 _i1 (-----);  
  or2 _i2 (-----);  
  mux2 _i3 (-----);  
  mux2 _i4 (-----);  
endmodule
```



```
module alu (input wire [1:0] op, input wire [15:0] i0, i1,  
output wire [15:0] o, output wire cout);  
wire [14:0] c;  
alu_slice _i0 (-----);  
alu_slice _i1 (-----);  
alu_slice _i2 (-----);  
alu_slice _i3 (-----);  
alu_slice _i4 (-----);  
alu_slice _i5 (-----);  
alu_slice _i6 (-----);  
alu_slice _i7 (-----);  
alu_slice _i8 (-----);  
alu_slice _i9 (-----);  
alu_slice _i10 (-----);
```

```
alu_slice _i11 (-----);  
alu_slice _i12 (-----);  
alu_slice _i13 (-----);  
alu_slice _i14 (-----);  
alu_slice _i15 (-----);  
endmodule
```

Testbench tb_alu.v

```
`timescale 1 ns / 100 ps
`define TESTVECS 16
module tb;
reg clk, reset;
reg [1:0] op;
reg [15:0] i0, i1;
wire [15:0] o;
wire cout;
  reg [33:0]
test_vecs [0:(`TESTVECS-1)];
integer i;
initial
begin
```

```
$dumpfile("tb_alu.vcd");  
$dumpvars(0,tb);  
end  
  initial  
begin  
  reset = 1'b1;  
  #12.5  
  reset = 1'b0;  
end  
  initial clk = 1'b0;  
  always  
  #5  
  clk =~ clk;  
  initial  
begin
```



```
test_vecs[0][33:32] = 2'b00;  
test_vecs[0][31:16] = 16'h0000;  
test_vecs[0][15:0] = 16'h0000;  
test_vecs[1][33:32] = 2'b00;  
test_vecs[1][31:16] = 16'haa55;  
test_vecs[1][15:0] = 16'h55aa;  
test_vecs[2][33:32] = 2'b00;  
test_vecs[2][31:16] = 16'hffff;  
test_vecs[2][15:0] = 16'h0001;  
test_vecs[3][33:32] = 2'b00;  
test_vecs[3][31:16] = 16'h0001;  
test_vecs[3][15:0] = 16'h7fff;  
test_vecs[4][33:32] = 2'b01;  
test_vecs[4][31:16] = 16'h0000;  
test_vecs[4][15:0] = 16'h0000;
```

```
test_vecs[5][33:32] = 2'b01;
  test_vecs[5][31:16] = 16'h55aa;
test_vecs[5][15:0] = 16'h55aa;
test_vecs[6][33:32] = 2'b01;
test_vecs[6][31:16] = 16'hffff;
test_vecs[6][15:0] = 16'h0001;
  test_vecs[7][33:32] = 2'b01;
test_vecs[7][31:16] = 16'h0001;
test_vecs[7][15:0] = 16'h7fff;
test_vecs[8][33:32] = 2'b10;
test_vecs[8][31:16] = 16'h0000;
test_vecs[8][15:0] = 16'h0000;
  test_vecs[9][33:32] = 2'b10;
test_vecs[9][31:16] = 16'h55aa;
test_vecs[9][15:0] = 16'h55aa;
```

.

```
test_vecs[10][33:32] = 2'b10;  
test_vecs[10][31:16] = 16'hffff  
test_vecs[10][15:0] = 16'h0001;  
test_vecs[11][33:32] = 2'b10;  
test_vecs[11][31:16] = 16'h0001;  
test_vecs[11][15:0] = 16'h7fff;  
test_vecs[12][33:32] = 2'b11;  
test_vecs[12][31:16] = 16'h0000;  
test_vecs[12][15:0] = 16'h0000;  
test_vecs[13][33:32] = 2'b11;  
test_vecs[13][31:16] = 16'haa55;  
test_vecs[13][15:0] = 16'h55aa;  
test_vecs[14][33:32] = 2'b11;  
test_vecs[14][31:16] = 16'hffff;  
test_vecs[14][15:0] = 16'h0001;
```

```
test_vecs[15][33:32] = 2'b11;
test_vecs[15][31:16] = 16'h0001;
test_vecs[15][15:0] = 16'h7fff;
end initial {op, i0, i1} = 0;
  alu alu_0 (op, i0, i1, o, cout);
initial begin
  #6
  for(i=0;i<`TESTVECS ; i=i+1)
    begin
      #10
      {op, i0, i1}=test_vecs[i];
    end
  #100 $finish;
end
endmodule
```

DESIGN AND SIMULATION

- To simulate the ALU with the test vectors, run the commands:
`iverilog -o tb_alu lib.v alu.v tb_alu.v`
- `vvp tb_alu`
- Above commands should produce the `tb_alu.vcd` file.
- To view the waveforms, run the command:
`gtkwave tb_alu.vcd`

| | op[1:0] | i0[15:0] | i1[15:0] | Output |
|-------------|---------|----------|-----------|--------|
| TESTVECTOR1 | 2'b00 | 16'h0000 | 16'h0000 | |
| TESTVECTOR2 | 2'b00 | 16'haa55 | 16'h55aa | |
| TESTVECTOR3 | 2'b00 | 16'hffff | 16'h0001 | |
| TESTVECTOR3 | 2'b00 | 16'h0001 | 16'h7fff | |
| TESTVECTOR4 | 2'b01 | 16'h0000 | 16'h0000 | |
| TESTVECTOR5 | 2'b01 | 16'haa55 | 16'h55aa | |
| TESTVECTOR6 | 2'b01 | 16'hffff | 16'h0001; | |
| TESTVECTOR7 | 2'b01 | 16'h0001 | 16'h7fff; | |

| | op[1:0] | i0[15:0] | i1[15:0] | Output |
|---------------|---------|----------|-----------|--------|
| TESTVECTOR8 | 2'b10 | 16'h0000 | 16'h0000 | |
| TESTVECTOR9 | 2'b10 | 16'haa55 | 16'h55aa | |
| TESTVECTOR10 | 2'b10 | 16'hffff | 16'h0001 | |
| TESTVECTOR11 | 2'b10 | 16'h0001 | 16'h7fff | |
| TESTVECTOR 12 | 2'b11 | 16'h0000 | 16'h0000 | |
| TESTVECTOR 13 | 2'b11 | 16'haa55 | 16'h55aa | |
| TESTVECTOR 14 | 2'b11 | 16'hffff | 16'h0001; | |
| TESTVECTOR 15 | 2'b11 | 16'h0001 | 16'h7fff; | |