Lab01 ALU

PB15000102 王嵩超

实验内容与要求

设计一算数运算单元ALU

- 采用纯组合逻辑设计
- 32bit位宽
- 完成制定运算功能

模块接口需求

```
module ALU(
  input  signed     [31:0]  alu_a,
  input  signed     [31:0]  alu_b,
  input             [4:0]   alu_op,
  output            [31:0]  alu_out
)
```

操作数与运算

实现以下7种操作：

```
parameter       A_NOP   = 5'h00;        空运算
parameter       A_ADD   = 5'h01;        符号加
parameter       A_SUB   = 5'h02;        符号减
parameter       A_AND   = 5'h03;        与
parameter       A_OR    = 5'h04;        或
parameter       A_XOR   = 5'h05;        异或
parameter       A_NOR   = 5'h06;        或非
```

完成以下运算

- 斐波拉契数列

  2, 2, 4, 6, 10, 16...

- 输入为a, b，其中a=2, b=2
- 调用ALU完成：
  - 输入为a=b=2，输出为16
  - 需要定义一个顶层模块，模块内调用ALU模块N次

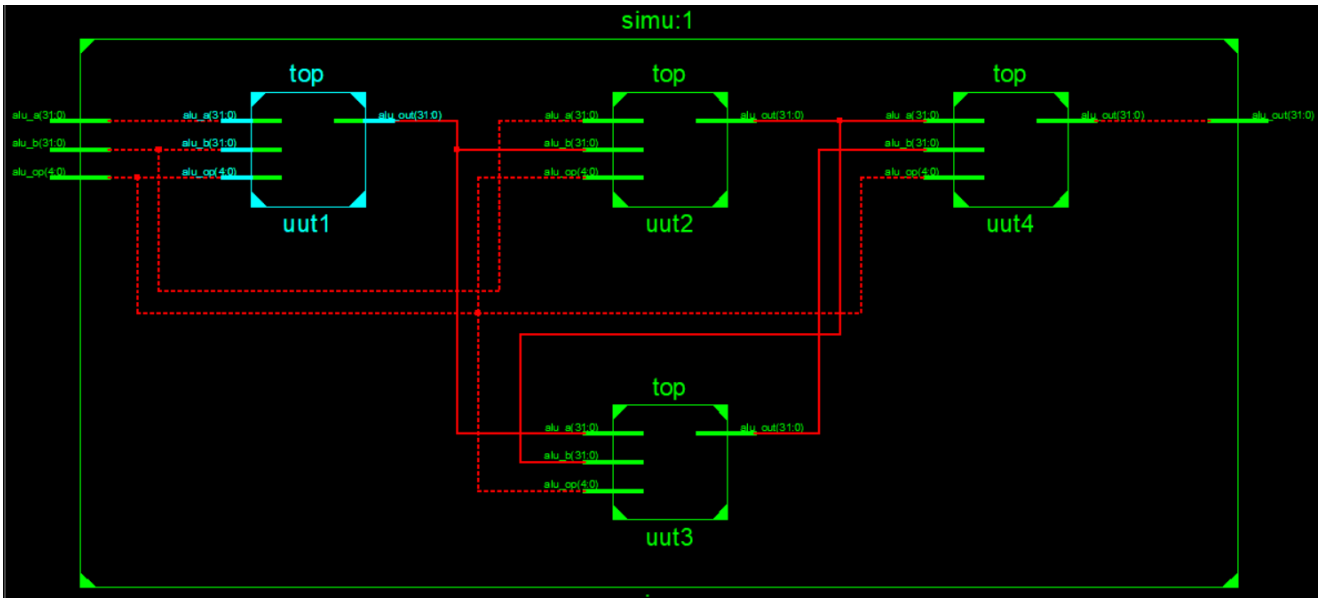
> 注意：要求中提到，运算单元采用纯组合逻辑设计。故在顶层模块应实例化4个ALU模块。在平时用时序逻辑设计中没必要用如此大的开销。

实验设计

- 首先设计ALU模块，该模块是运算单元。

  仅用always模块，if语句和各输入变量的逻辑运算即可完成。
- 再编写top模块，该模块例化了4个ALU模块，并将运算结果依次串接，实现斐波拉契数列的计算。
- 再编写simu仿真模块，该模块提供输入用来仿真。

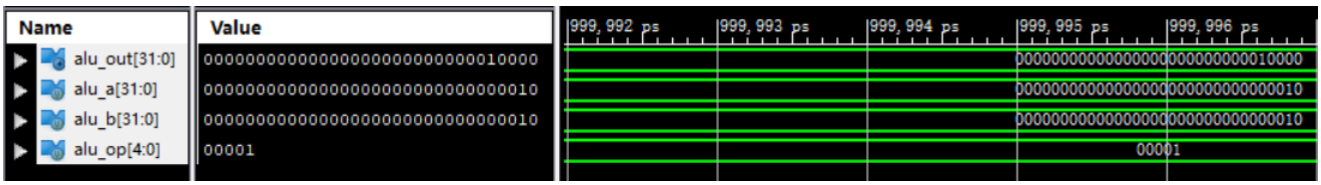
RTL级电路框图（模块名称稍有不同）



仿真结果

斐波拉契数列计算

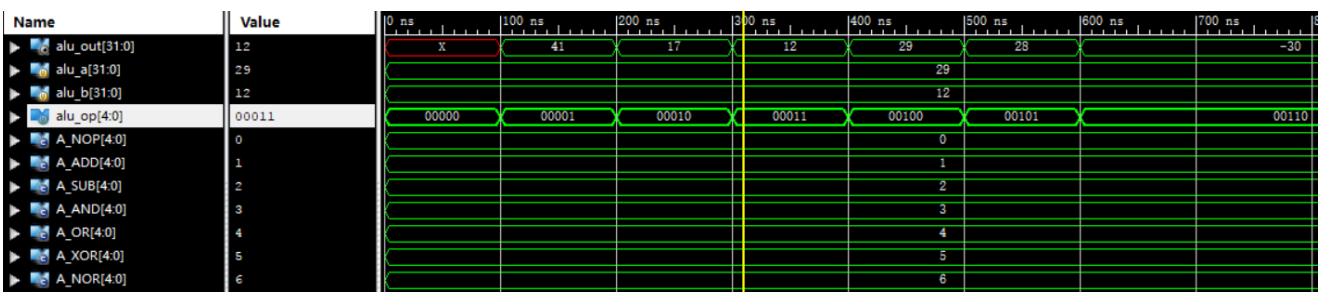
新建一个Verilog Test Fixture源文件，对top模块仿真：



输出结果为16(10000)。

各操作符运算仿真

新建一个Verilog Test Fixture源文件，对alu模块仿真：



操作符每隔100ns切换一次，刚开始的空操作使alu_out置为高阻态。往后依次是符号加、符号减、与运算、或运算、异或运算、或非运算。

源代码

ALU模块：

```
`timescale 1ns / 1ps

module alu(
    input signed [31:0] alu_a,
    input signed [31:0] alu_b,
    input [4:0] alu_op,
    output reg [31:0] alu_out
    );
parameter A_NOP = 5'h00; //空运算
parameter A_ADD = 5'h01; //符号加
parameter A_SUB = 5'h02; //符号减
parameter A_AND = 5'h03; //与
parameter A_OR  = 5'h04; //或
parameter A_XOR = 5'h05; //异或
parameter A_NOR = 5'h06; //或非

always@(*)
begin
    case(alu_op)
     A_NOP:
     begin

     end
     A_ADD:
     begin
         alu_out <= alu_a + alu_b;
     end
     A_SUB:
     begin
         alu_out <= alu_a - alu_b;
     end
     A_AND:
     begin
         alu_out <= alu_a & alu_b;
     end
     A_OR:
     begin
         alu_out <= alu_a | alu_b;
     end
     A_XOR:
     begin
         alu_out <= alu_a ^| alu_b;
    end
     A_NOR:
     begin
         alu_out <= ~(alu_a | alu_b);
     end
     endcase
end
endmodule
```

斐波拉契数列模块：

```
`timescale 1ns / 1ps
module top(
    input signed [31:0] alu_a,
    input signed [31:0] alu_b,
    input [4:0] alu_op,
    output [31:0] alu_out
    );

    // Temps
    wire [31:0] sum1;
    wire [31:0] sum2;
    wire [31:0] sum3;

    // Outputs

    // Instantiate the Unit Under Test (UUT)
    alu uut1 (
         .alu_a(alu_a),
         .alu_b(alu_b),
         .alu_op(alu_op),
         .alu_out(sum1)
    );
    alu uut2 (
         .alu_a(alu_b),
         .alu_b(sum1),
         .alu_op(alu_op),
         .alu_out(sum2)
    );
    alu uut3 (
         .alu_a(sum1),
         .alu_b(sum2),
         .alu_op(alu_op),
         .alu_out(sum3)
    );
    alu uut4 (
         .alu_a(sum2),
         .alu_b(sum3),
         .alu_op(alu_op),
         .alu_out(alu_out)
    );

endmodule
```

斐波拉契数列仿真：

```
`timescale 1ns / 1ps
module simu0;

    // Inputs
    reg [31:0] alu_a;
    reg [31:0] alu_b;
    reg [4:0] alu_op;

    // Outputs
    wire [31:0] alu_out;

    // Instantiate the Unit Under Test (UUT)
    simu uut (
        .alu_a(alu_a),
        .alu_b(alu_b),
        .alu_op(alu_op),
        .alu_out(alu_out)
    );

    initial begin
        // Initialize Inputs
        // Add stimulus here
        alu_a =2;
        alu_b =2;
        alu_op = 5'h01;
    end
endmodule
```

各操作符仿真模块：

```
`timescale 1ns / 1ps

module simueach;

    // Inputs
    reg [31:0] alu_a;
    reg [31:0] alu_b;
    reg [4:0] alu_op;

    // Outputs
    wire [31:0] alu_out;

    parameter A_NOP = 5'h00; //空运算
parameter A_ADD = 5'h01; //符号加
parameter A_SUB = 5'h02; //符号减
parameter A_AND = 5'h03; //与
parameter A_OR  = 5'h04; //或
parameter A_XOR = 5'h05; //异或
parameter A_NOR = 5'h06; //或非
    // Instantiate the Unit Under Test (UUT)
    top uut (
        .alu_a(alu_a),
        .alu_b(alu_b),
        .alu_op(alu_op),
        .alu_out(alu_out)
    );
    initial begin
        // Initialize Inputs
        alu_a = 29;
        alu_b = 12;
        alu_op = A_NOP;

        // Wait 100 ns for global reset to finish
        #100;
      alu_op = A_ADD;
        #100;
        alu_op = A_SUB;
        #100;
        alu_op = A_AND;
        #100;
        alu_op = A_OR;
        #100;
        alu_op = A_XOR;
        #100;
        alu_op = A_NOR;
        // Add stimulus here
    end
endmodule
```