# 计算机组成原理 Project2 – ALU 实验

吴育昕 刘啸宇 宋方睿

## 目录



## 1 实验目的

1. 熟悉硬件描述语言及开发环境,了解硬件系统开发的基本过程
2. 掌握 ALU 基本设计方法和简单运算器的数据传送通路
3. 验证 ALU 的功能

## 2 实验工具及环境

**操作系统** Windows XP (虚拟机)

**软件** Xilinx ISE 14.6

**语言** Verilog

# 3 实验原理

本实验内容是根据算术逻辑运算单元的功能表,通过状态机状态的变化,达到改变控制信号的组合的目的,从而实现不同的算术与逻辑运算功能,并将结果与标志位显示出来.实验中的 ALU 可以实现基本的算术运算、逻辑运算、移位运算等,功能如下表所示:

| 操作码 | 功能 | 描述 |
|---|---|---|
| ADD | A+B | 加法 |
| SUB | A-B | 减法 |
| AND | A and B | 与 |
| OR | A or B | 或 |
| XOR | A xor B | 异或 |
| NOT | not A | 非 |
| SLL | A sll B | 逻辑左移 |
| SRL | A sll B | 逻辑右移 |
| SRA | A sra B | 算术右移 |
| ROL | A rol B | 循环左移 |

# 4 模块设计

顶层 alu 分为三个模块:

1. 状态机 inputState: 接收 clk, 在读入/输出的四种状态中切换, 并向 core 和 selector 发送指令.
2. 计算核心 core: 当状态机状态进入计算时, 对输入数据执行计算.
3. 选择器 selector: 当状态机进入某个输出状态时, 选择相应的 flag 或 result 进行输出.

相关接口如`alu.v`所示:

src/alu.v

```
module alu(
  input clk,
  input rst,
  input [15:0] data,
  output [15:0] display
);

wire [15:0] dataA, dataB;
wire [15:0] result, flags;
wire [3:0] op;
wire doCal, showFlag;

inputState inputStateM (
  clk,
  rst,
```

```
16    data,
17    dataA,
18    dataB,
19    op,
20    doCal,
21    showFlag
22  );
23
24  core coreM (
25    doCal,
26    rst,
27    dataA,
28    dataB,
29    op,
30    result,
31    flags
32  );
33
34  selector selectorM (
35    showFlag,
36    result,
37    flags,
38    display
39  );
40
41  endmodule
```

各模块详细代码见附录.

## 5 实验过程

首先测试发现三个人的 Linux 系统上的 Xilinx 软件都无法正常连接实验平台, 于是开始安装 Windows XP 虚拟机并安装 Xilinx, 之后开始代码调试.

代码烧入后实验机上没有任何反应, 经过分析后认为应该是 Reset 的状态不对. 通过修改代码输出 Reset 值, 发现 Reset 按下为 0, 与代码期待行为相反, 因而实验平台一直处于 Reset 状态.

之后的主要问题是不清楚平台上管脚高低位的对应顺序, 因此无法观测程序行为. 做了几次小实验后清楚了平台设置, 再将原始代码烧入后行为便正常了. 做了一些测试后均未出现问题, 于是结束实验.

## 6 经验及总结

这次实验总体比较简单, 我们在实验前期主要复习巩固了 verilog 语法, 也熟悉了仿真软件的使用. 经过仿真, 可以尽早消灭 bug, 大大减少实验中的调试时间.

实验中在环境配置上出了些问题, 但也很快通过虚拟机得到了解决.

另外, 这次的实验, 我们将各部分功能进行了模块化设计, 锻炼了初步的模块设计能力, 希望能为今后进一步的实验有所帮助.

# 7 思考题

1. ALU 所使用的电路是组合逻辑电路还是时序逻辑电路?

   因为输入机拥有四种状态，其行为与所处状态有关，因此为时序逻辑电路。

2. 如果给定了 A 和 B 的初值, 每次运算后都把结果写入 B 中再进行下次运算, 这样的一个带暂存功能的 ALU 要增加一些什么电路来实现?

   给定了 A 和 B 的初值需要两个可读寄存器，在进入输入阶段时先进行赋值。结果写入 B 说明需要用寄存器存下运算的结果，然后再输入阶段再次赋值给 B。具体的电路实现可以使用 D 触发器进行实现。

# 8 实验代码

src/inputState.v

```
module inputState(
  input clk,
  input rst,
  input [15:0] data,
  output reg [15:0] dataA, dataB,
  output reg [3:0] op,
  output reg doCal, showFlag
);

parameter SIZE = 2;
parameter DATAA = 2'b00, DATAB = 2'b01, OP = 2'b10, SF = 2'b11;
reg [SIZE - 1 : 0] state;

always @ (negedge rst or posedge clk)
begin : FSM_SEQ
  if (rst == 0) begin
    state <= #1 DATAA;
  end
  else
  begin
    case (state)
      DATAA: state <= DATAB;
      DATAB: state <= OP;
      OP: state <= SF;
      SF: state <= DATAA;
    endcase
  end
end

always @ (negedge rst or posedge clk)
begin : OUTPUT_LOGIC
  if (rst == 0) begin
    dataA <= 0;
    dataB <= 0;
    doCal <= 0;
    showFlag <= 0;
```

```
37    end
38    else
39    begin
40      case (state)
41        DATAA: dataA <= data;
42        DATAB: dataB <= data;
43        OP:
44        begin
45          op <= data[3:0];
46          showFlag <= 0;
47          doCal <= 1;
48        end
49        SF:
50        begin
51          doCal <= 0;
52          showFlag <= 1;
53        end
54      endcase
55    end
56  end
57  endmodule
```

src/core.v

```
1   module core (
2     clk,
3     rst,
4     dataA,
5     dataB,
6     op,
7     result,
8     flags
9   );
10
11  input clk;
12  input rst;
13  input dataA;
14  input dataB;
15  input op;
16
17  output result;
18  output flags;
19
20  wire clk;
21  wire rst;
22  wire [15:0] dataA, dataB;
23  wire [3:0] op;
24  reg [15:0] result;
25  reg [15:0] flags;
26
27  parameter ADD = 4'b0000,
28    SUB = 4'b0001,
29    AND = 4'b1000,
30    OR  = 4'b1001,
```

```
31    XOR = 4'b1010,
32    NOT = 4'b1011,
33    SLL = 4'b1100,
34    SRL = 4'b1101,
35    SRA = 4'b0010,
36    ROL = 4'b0011;
37
38
39  always @ (negedge rst or posedge clk)
40  begin : CORE
41    if (rst == 0) begin
42      result <= 0;
43    end
44    else
45    begin
46      case (op)
47        ADD: result <= dataA + dataB;
48        SUB: result <= dataA - dataB;
49        OR:  result <= dataA | dataB;
50        XOR: result <= dataA ^ dataB;
51        NOT:
52          if (dataA == 0) begin
53            result <= 1;
54          end else begin
55            result <= 0;
56          end
57        SLL: result <= dataA << dataB;
58        SRL: result <= dataA >> dataB;
59        SRA: result <= dataA >>> dataB;
60        ROL: result <= (dataA << dataB) | dataA >> (16 - dataB);
61          default: result <= op;
62      endcase
63    end
64  end
65
66  endmodule
```

src/selector.v

```
1   module selector(
2     input control,
3     input [15:0] dataA, dataB,
4     output reg [15:0] data
5   );
6
7   always @*
8   begin
9     case (control)
10      0: data <= dataA;
11      1: data <= dataB;
12    endcase
13  end
14  endmodule
```