# 实验 2：CPU 功能部件设计

张子康 PB22020660

2024 年 04 月 03 日

# 1 实验目的与内容

## 1.1 寄存器堆设计

设计符合要求的 32 位寄存器堆，并进行仿真。

## 1.2 ALU 设计

设计符合要求的 32 位 ALU，并进行仿真。你需要自行编写合适的仿真文件，验证每一种运算模式下 ALU 计算的正确性。

## 1.3 在线计算器

使用任务 2 中的 ALU，在 FPGAOL 上搭建一个简单的计算器。

## 1.4 初始化存储器

创建一个新的项目，例化合适大小的存储器 IP 核（分布式、块式均可），将上一次实验生成的指令段 COE 文件导入到 IP 核中，并向助教展示。

# 2 逻辑设计

## 2.1 寄存器堆设计

```
module REG_FILE (
    // 时钟信号
    input [0 : 0] clk,
    // 寄存器读地址0，五位宽，可选择32个寄存器中的任意一个
    input [4 : 0] rf_ra0,
    // 寄存器读地址1，同样五位宽，用于第二个读取端口
    input [4 : 0] rf_ra1,
    // 寄存器写地址，五位宽，指定要写入数据的寄存器位置
    input [4 : 0] rf_wa,
    // 写使能信号，当该信号为高时，允许写操作
```

```
11      input [0 : 0] rf_we,
12      // 数据写入寄存器的数据输入，32位宽
13      input [31 : 0] rf_wd,
14      // 从寄存器读取的数据输出0，对应ra0指定的寄存器内容
15      output [31 : 0] rf_rd0,
16      // 从寄存器读取的数据输出1，对应ra1指定的寄存器内容
17      output [31 : 0] rf_rd1);
18
19      // 定义一个32x32位的寄存器文件
20      // 总共可以存储32个32位的寄存器数据
21      reg [31 : 0] reg_file [0 : 31];
22
23      // 用于初始化寄存器
24      integer i;
25      initial begin
26          for (i = 0; i < 32; i = i + 1)
27              reg_file[i] = 0;
28      end
29
30      // 读取指定寄存器的数据
31      assign rf_rd0 = reg_file[rf_ra0];
32      assign rf_rd1 = reg_file[rf_ra1];
33
34      // 向指定寄存器写入数据
35      always @(posedge clk) begin
36          // 指定的寄存器是否为可写入且不是0号寄存器则写入数
                据
37          if (rf_we && rf_wa != 5'd00000)
38              reg_file[rf_wa] <= rf_wd;
39          else
40              reg_file[rf_wa] <= reg_file[rf_wa];
41      end
42
```

```
endmodule
```

以上代码实现了一个寄存器堆，支持 RV32I 指令集，具有以下特点：

1. 0 号寄存器始终为 0，永远无法被更改；
2. 时钟上升沿到来时，如果写使能信号有效，则进行写入操作，否则不进行写入操作；
3. 寄存器堆的读操作是时钟异步的（实际上是组合逻辑），即只要地址给定，对应寄存器的数值就能读出，而无需等待时钟边沿的到来。

## 2.2 ALU 设计

```
// 定义相应的运算操作码

// 加法
`define ADD                 5'B00000
// 减法
`define SUB                 5'B00010
// 有符号小于
`define SLT                 5'B00100
// 无符号小于
`define SLTU                5'B00101
// 按位与
`define AND                 5'B01001
// 按位或
`define OR                  5'B01010
// 按位异或
`define XOR                 5'B01011
// 左移
`define SLL                 5'B01110
// 逻辑右移
`define SRL                 5'B01111
// 算术右移
```

```
22 `define SRA                  5'B10000
23 // 选择第一个操作数
24 `define SRC0                 5'B10001
25 // 选择第二个操作数
26 `define SRC1                 5'B10010
27
28 module ALU (input [31 : 0] alu_src0, // 第一个操作数
29             input [31 : 0] alu_src1, // 第二个操作数
30             input [4 : 0] alu_op,    // 操作码
31             output reg [31 : 0] alu_res); // 运算结果
32
33 // 内部辅助寄存器，存储操作数的副本，用于计算
34 reg signed[31:0] tem0;
35 reg signed[31:0] tem1;
36
37 // 时序逻辑块，根据操作码执行相应的运算
38 always @(*) begin
39     tem0 = alu_src0; // 复制第一个操作数
40     tem1 = alu_src1; // 复制第二个操作数
41
42     // 根据操作码选择执行哪种运算
43     case(alu_op)
44         `ADD  : alu_res = tem0 + tem1; // 加法
45         `SUB  : alu_res = tem0 - tem1; // 减法
46         `SLT  : alu_res = (tem0 < tem1) ? 32'h00000001 :
              32'h00000000; // 有符号小于
47         `SLTU : alu_res = (alu_src0 < alu_src1) ? 32'
              h00000001 : 32'h00000000; // 无符号小于
48         `AND  : alu_res = tem0 & tem1; // 按位与
49         `OR   : alu_res = tem0 | tem1; // 按位或
50         `XOR  : alu_res = tem0 ^ tem1; // 按位异或
51         `SLL  : alu_res = tem0 << tem1[4:0]; // 左移
52         `SRL  : alu_res = tem0 >> tem1[4:0]; // 逻辑右移
```

```
        `SRA  : alu_res = tem0 >>> tem1[4:0]; // 算术右移
        `SRC0 : alu_res = alu_src0; // 直接选择第一个操作数
        `SRC1 : alu_res = alu_src1; // 直接选择第二个操作数

        // 默认情况下，若操作码不在上述列表中，则将结果清零
        default : alu_res = 32'H0;
    endcase
end

endmodule
```

上述代码中定义了一个 ALU 模块，其中输入为无符号数，tem0 与 tem1 为有符号副本，回避了一些比较麻烦的手动处理。

## 2.3 在线计算器

```
module TOP (input [0 : 0] clk,// 时钟信号
    input [0 : 0] rst, // 复位信号
    input [0 : 0] enable, // 写使能信号
    input [4 : 0] in, // 输入信号
    input [1 : 0] ctrl, // 控制信号
    output [3 : 0] seg_data,// 用于驱动七段显示器的数据线
    output [2 : 0] seg_an);// 用于驱动七段显示器的段选线

    // 定义ALU相关信号
    reg [31:0] src0;// 第一个操作数
    reg [31:0] src1;// 第二个操作数
    reg [4:0] op;// 要进行的操作
    wire [31:0] res;// 计算结果
    // 实例化ALU模块
    ALU alu(
        .alu_src0(src0),
        .alu_src1(src1),
        .alu_op(op),
```

```
19          .alu_res(res)
20      );
21
22      // 定义寄存器文件相关的信号
23      reg     [ 4 : 0]    ra0, ra1, wa;
24      reg     [ 0 : 0]    we;
25      reg     [31 : 0]    wd;
26      wire    [31 : 0]    rd0;
27      wire    [31 : 0]    rd1;
28      // 实例化寄存器模块
29      REG_FILE regfile (
30          .clk    (clk),
31          .rf_ra0     (ra0),
32          .rf_ra1     (ra1),
33          .rf_wa      (wa),
34          .rf_we      (we),
35          .rf_wd      (wd),
36          .rf_rd0     (rd0),
37          .rf_rd1     (rd1)
38      );
39
40      // 实例化Segment模块
41      Segment segment(
42          .clk(clk),
43          .rst(rst),
44          .output_data(res),
45          .seg_data(seg_data),
46          .seg_an(seg_an)
47      );
48
49      // flage用于确定当前是否进行计算
50      reg flage;
51      // t0和t1用于进行计数，确定当前进行的步骤
```

```
52      reg [2:0] t0;
53      reg [2:0] t1;
54
55      // 初始化
56      initial begin
57          src0=0;
58          src1=0;
59          op=0;
60          ra0=0;
61          ra1=0;
62          wa=0;
63          we=1'b1;
64          wd=0;
65          flage=0;
66          t0=0;
67          t1=0;
68      end
69
70      always @(posedge clk) begin
71          // 处理复位信号，不对寄存器进行操作
72          if(rst)begin
73              src0<=0;
74              src1<=0;
75              op<=0;
76              ra0<=0;
77              ra1<=0;
78              wa<=0;
79              we<=1'b1;
80              wd<=0;
81              flage<=0;
82              t0<=0;
83              t1<=0;
84          end else begin
```

```
85              // 判断当前是否需要从寄存器读取数据进行操作
86              if(flage) begin
87                  // 先从寄存器读取op和第一个操作数
88                  if(t0==3'h0) begin
89                      op<=rd0[4:0];
90                      src0<=rd1;
91                      t1<=t0+1;
92                      t0<=t1+1;
93                  end else begin
94                      // 将读取的寄存器地址指向第二个操作数所
                           在的寄存器
95                      if(t0==3'h1) begin
96                          ra1<=5'h3;
97                          t0<=t1+1;
98                          t1<=t0+1;
99                      end else begin
100                     // 读取第二个操作数并将flage, t0和t1值
                           复位
101                         src1<=rd1;
102                         t0<=0;
103                         t1<=0;
104                         flage<=0;
105                     end
106                 end
107             end else begin
108                 // 判断当前是否按下按钮, 如果按下就储存当前
                       数据
109                 if(enable)
110                     // 判断当前要进行的操作
111                     case(ctrl)
112                         // 输入OP, 进行无符号扩展
113                         2'b00:
114                         begin
```

```
115                         wd<={27'h0,in};
116                         wa<=5'h1;
117                     end
118                     // 输入第一个操作数，并进行符号扩展
119                     2'b01:
120                     begin
121                         wd<={{27{in[4]}},in};
122                         wa<=5'h2;
123                     end
124                     // 输入第二个操作数，并进行符号扩展
125                     2'b10:
126                     begin
127                         wd<={{27{in[4]}},in};
128                         wa<=5'h3;
129                     end
130                     // 进行计算
131                     2'b11:
132                     begin
133                         ra0<=5'h1;
134                         ra1<=5'h2;
135                         flage<=1;
136                         t0<=0;
137                         t1<=0;
138                     end
139                 endcase
140             else begin
141                 src0<=src0;
142                 src1<=src1;
143                 op<=op;
144                 ra0<=ra0;
145                 ra1<=ra1;
146                 wa<=wa;
147                 we<=1'b1;
```

```
148                         wd<=wd;
149                     end
150                 end
151             end
152         end
153 endmodule
```

以上为计算器的 TOP 模块。

实际上本题并未要求使用寄存器堆，以上代码实现的过于麻烦了。可以直接利用 op,src0 与 src1 进行存储，以此可以极大的简化代码。

## 2.4 初始化存储器

例化合适大小的分布式存储器 IP 核，将上一次实验生成的指令段 COE 文件导入到 IP 核中。图片如下：

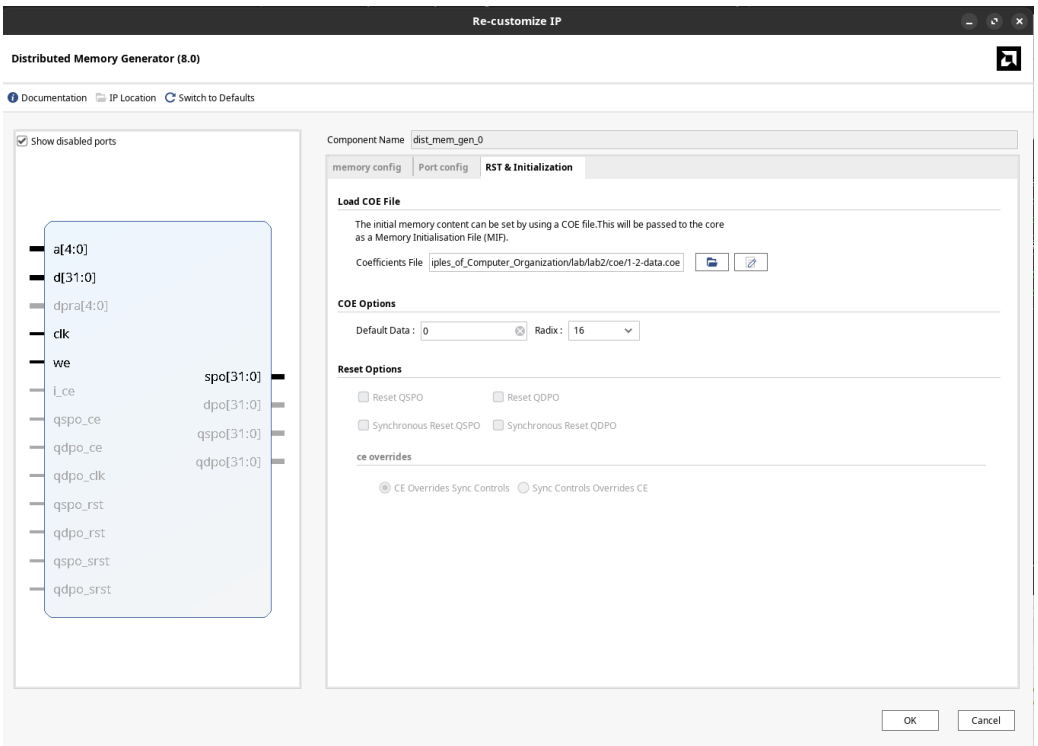


图 1: 导入的 coe 文件

# 3 仿真结果与分析

## 3.1 寄存器堆设计

仿真代码如下:

```
module RegFile_tb ();
    reg                 clk;
    reg     [ 4 : 0]    ra0, ra1, wa;
    reg     [ 0 : 0]    we;
    reg     [31 : 0]    wd;
    wire    [31 : 0]    rd0;
    wire    [31 : 0]    rd1;

    REG_FILE regfile (
        .clk    (clk),
        .rf_ra0    (ra0),
        .rf_ra1    (ra1),
        .rf_wa     (wa),
        .rf_we     (we),
        .rf_wd     (wd),
        .rf_rd0    (rd0),
        .rf_rd1    (rd1)
    );

    initial begin
        clk = 0;
        ra0 = 5'H0; ra1 = 5'H0; wa = 5'H0; we = 1'H0; wd =
           32'H0;

        #12
        ra0 = 5'H0; ra1 = 5'H0; wa = 5'H3; we = 1'H1; wd =
```

```
            32'H12345678;
26
27          #5
28          ra0 = 5'H0; ra1 = 5'H0; wa = 5'H0; we = 1'H0; wd =
               32'H0;
29
30          #5
31          ra0 = 5'H3; ra1 = 5'H2; wa = 5'H2; we = 1'H1; wd =
               32'H87654321;
32
33          #5
34          ra0 = 5'H0; ra1 = 5'H0; wa = 5'H0; we = 1'H0; wd =
               32'H0;
35
36          #5
37          ra0 = 5'H3; ra1 = 5'H0; wa = 5'H0; we = 1'H1; wd =
               32'H87654321;
38
39          #10
40          $finish;
41      end
42      always #5 clk = ~clk;
43  endmodule
```

仿真结果如图：

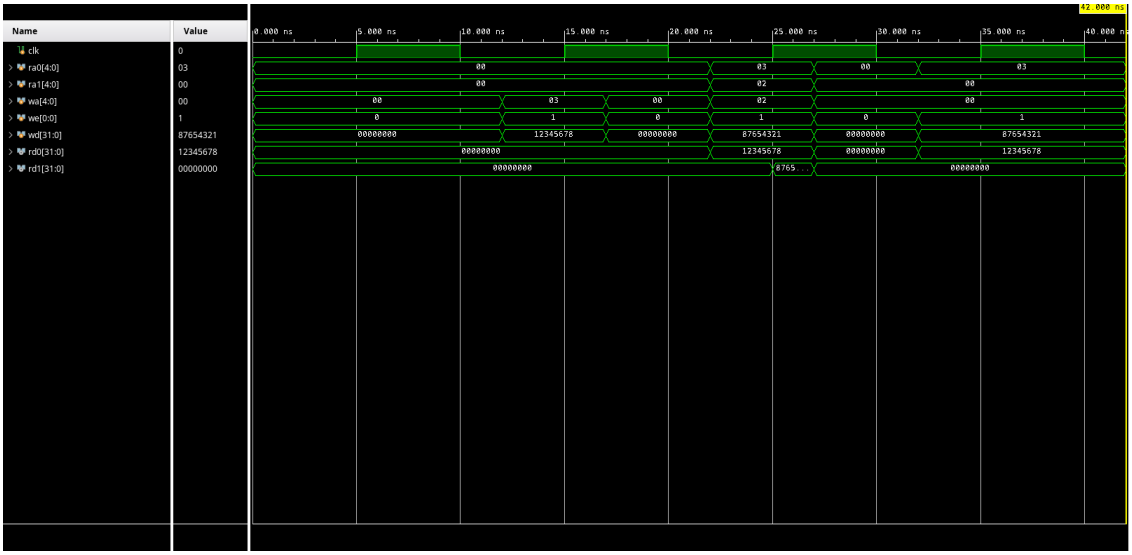


图 2: 寄存器堆仿真结果

## 3.2 ALU 模块

仿真代码如下:

```
`timescale 1ns / 1ps
module alu_testbench();
    parameter clk_sep  = 1;
    parameter time_sep = 10;
    reg [31:0] src0;
    reg [31:0] src1;
    reg [4:0] op;
    wire [31:0] res;
    reg clk;
    ALU alu(
        .alu_src0(src0),
        .alu_src1(src1),
        .alu_op(op),
        .alu_res(res)
    );
    initial begin
```

```
17          clk=0;
18          forever begin
19              #time_sep
20              clk=~clk;
21          end
22      end
23      initial begin
24          src0=32'h81111111;
25          src1=32'h11111111;
26          op=5'B00000;
27          #clk_sep
28          op=5'B00010;
29          #clk_sep
30          op=5'B00100;
31          #clk_sep
32          op=5'B00101;
33          #clk_sep
34          op=5'B01001;
35          #clk_sep
36          op=5'B01010;
37          #clk_sep
38          op=5'B01011;
39          #clk_sep
40          op=5'B01110;
41          #clk_sep
42          op=5'B01111;
43          #clk_sep
44          op=5'B10000;
45          #clk_sep
46          op=5'B10001;
47          #clk_sep
48          op=5'B10010;
49          #clk_sep
```

```
50          $finish;
51      end
52  endmodule
```

仿真结果如图：


图 3: ALU 模块仿真结果

# 4 电路设计与分析

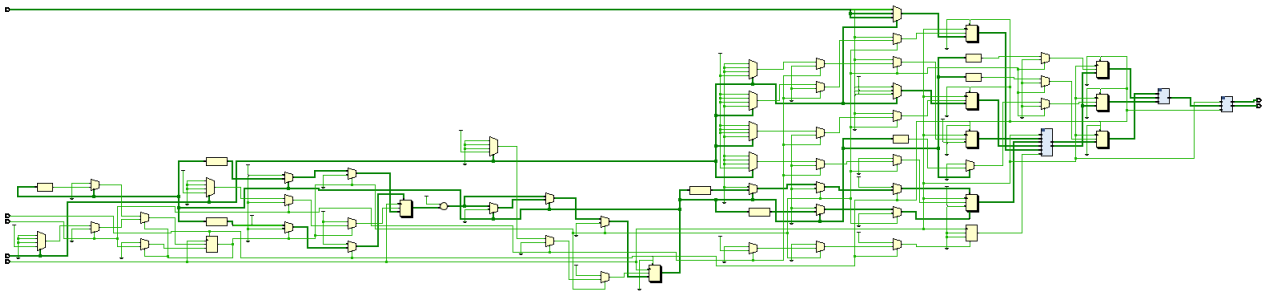

图 4: 完整的电路图

# 5 测试结果与分析

## 5.1 在线计算器

将编译好的 bit 文件导入 FPAGOL, 并且输入 $OP = 5'b00010, SRC0 = 5'b00001, SRC1 = 5'b111111$，计算结果如图所示：

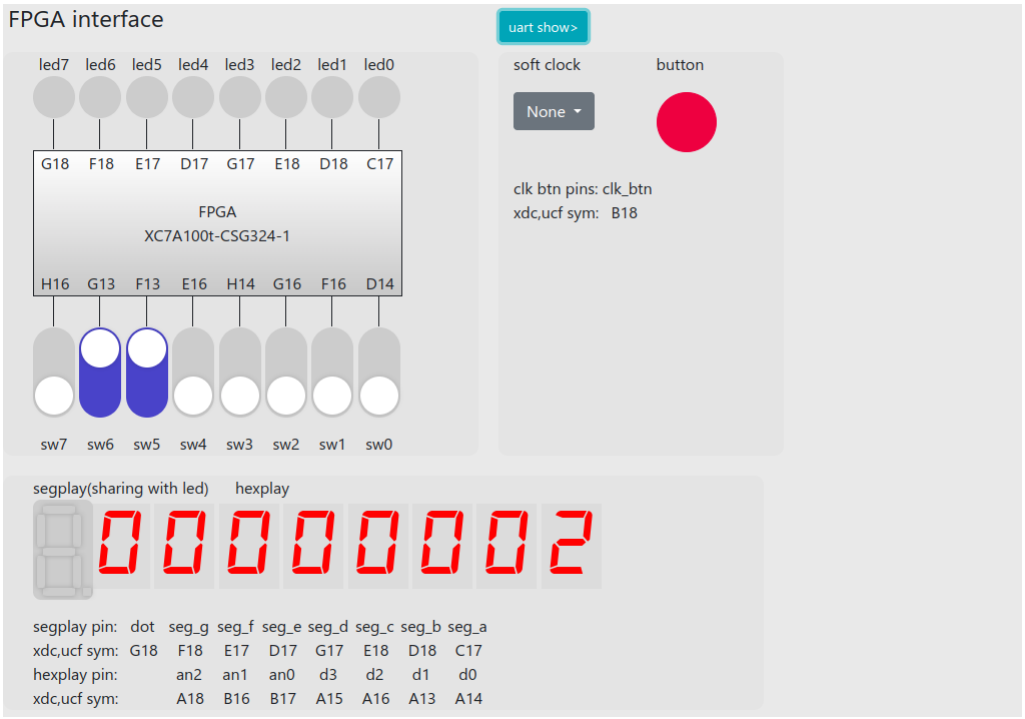


图 5: FPAG 运行结果

# 6 总结

通过本次实验初步掌握了 verilog，并完成了寄存器堆、ALU 模块、在线计算器的设计与 ip 核初始化。