lab2 实验报告

李毅PB22051031

第一部分 实验内容

1.1 实验题目：CPU功能部件设计

1. 寄存器堆设计及仿真，正确实现寄存器堆的逻辑设计； 完成寄存器堆电路的功能仿真。
2. 32位算术逻辑单元（ALU）设计及仿真， 正确实现ALU的逻辑设计；完成ALU的功能仿真。
3. 算术逻辑单元（ALU）应用-计算器设计，正确实现计算器电路的逻辑设计；完成计算器电路下载测试。
4. 存储器IP核例化及初始化，完成存储器（块式或分布式） IP核例化及初始内容加载。

1.2 实验平台

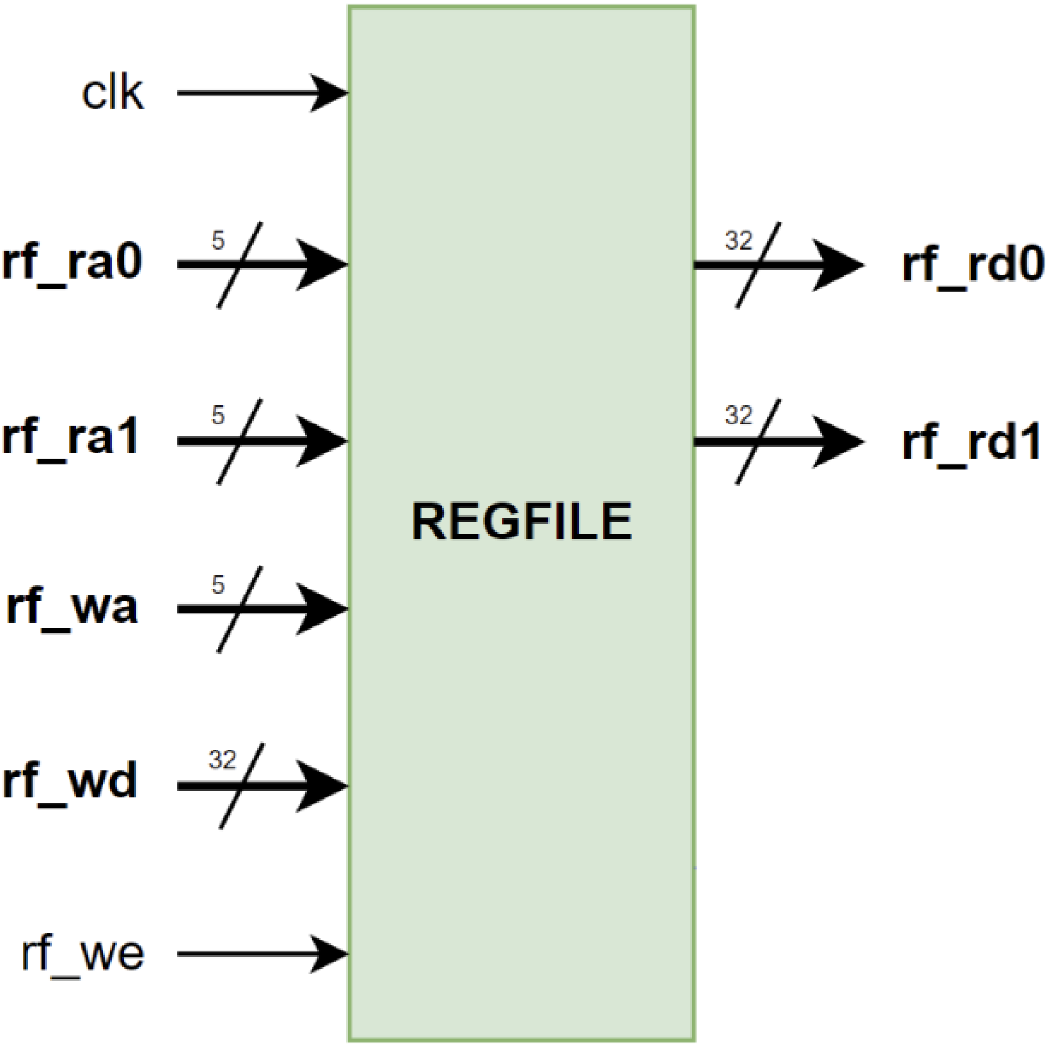
本实验使用vivado 2019.2进行代码编写与仿真

第二部分 实验过程

2.1 实验设计

实验项目1：

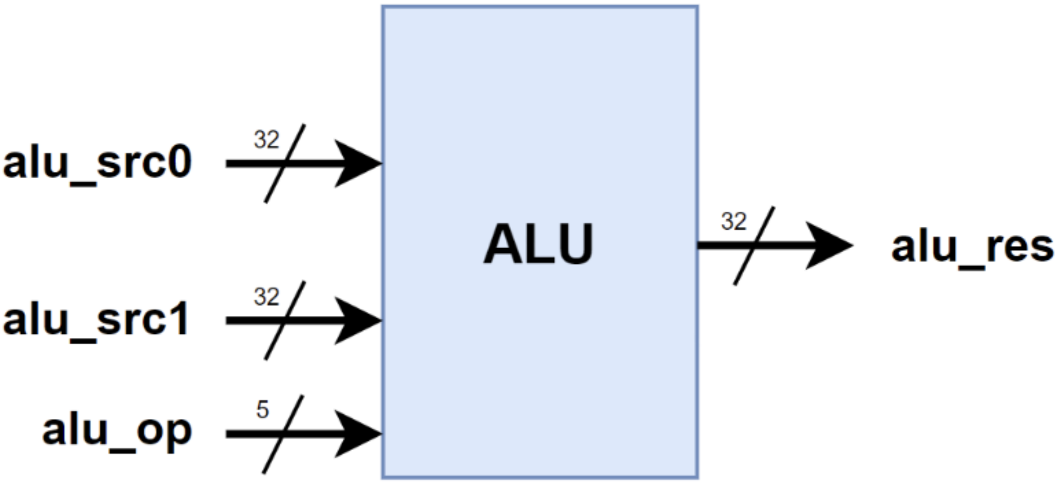
寄存器堆数据通路结构如下图所示：



其中，rf_ra0 - rf_rd0、rf_ra1 - rf_rd1 为数据读取端口；rf_wd - rf_wa 为数据写入端口；rf_we 为写使能信号。

实验项目2：

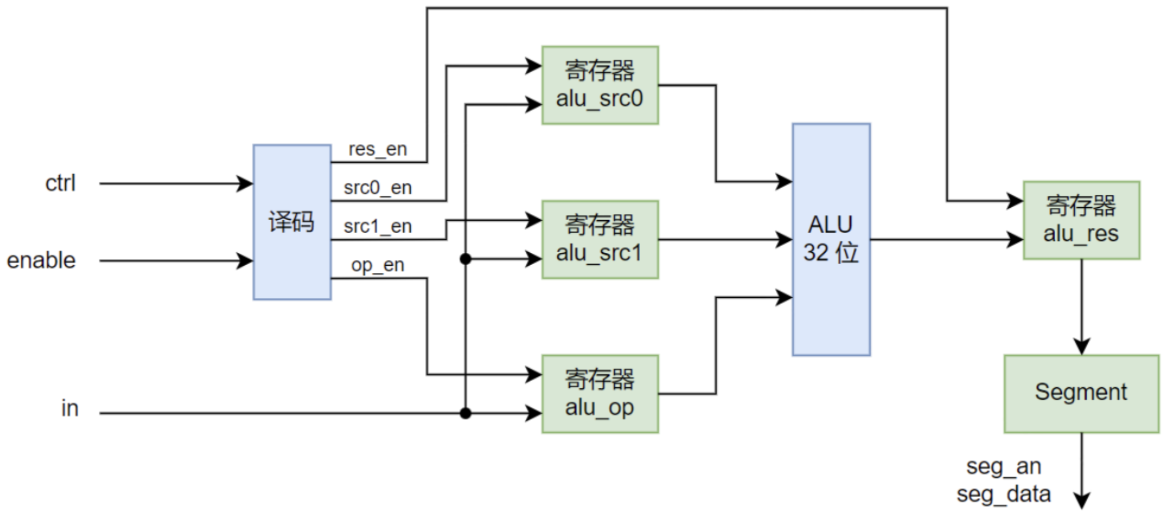
ALU数据通路结构如下图所示：



其中alu_src0,alu_src1为输入，alu_res为输出，alu_op为控制信号。

实验项目3：

运算器数据通路如下图所示：



2.2 核心代码

实验项目1：

```
// 用于初始化寄存器
    integer i;
    initial begin
        for (i = 0; i < 32; i = i + 1)
            reg_file[i] = 0;
    end

    assign rf_rd0=reg_file[rf_ra0];
    assign rf_rd1=reg_file[rf_ra1];

    always @(posedge clk) begin
        if (rf_we) begin
            reg_file[rf_wa] <= rf_wd;
        end
        reg_file[5'b0]<=32'b0;
    end
```

实验项目2：

```
always @(*) begin
        case(alu_op)
            `ADD :
                alu_res = alu_src0 + alu_src1;
            `SUB :
                alu_res = alu_src0 - alu_src1;
            `SLT :
                alu_res = ($signed(alu_src0) < $signed(alu_src1)) ? 32'b1 : 32'b0;
            `SLTU :
                alu_res = (alu_src0 < alu_src1) ? 32'b1 : 32'b0;
            `AND :
                alu_res = alu_src0 & alu_src1;
            `OR :
                alu_res = alu_src0 | alu_src1;
            `XOR :
                alu_res = alu_src0 ^ alu_src1;
            `SLL :
                alu_res = alu_src0 << alu_src1;
            `SRL :
                alu_res = alu_src0 >> alu_src1;
            `SRA :
                alu_res = $signed(alu_src0) >>> $signed(alu_src1);
            `SRC0 :
                alu_res = alu_src0;
            `SRC1 :
                alu_res = alu_src1;
            default :
                alu_res = 32'H0;
        endcase
end
```

实验项目3:

```
ALU alu(
    .alu_op(alu_op),
    .alu_src0(alu_src0),
    .alu_src1(alu_src1),
    .alu_res(alu_res)
);

Segment segment(
    .clk(clk),
    .rst(rst),
    .output_data(alu_res2),
    .seg_data(seg_data),
    .seg_an(seg_an)
);
always @(posedge clk) begin
    if(rst==1'b1)begin
        alu_op=5'b0;
        alu_src0=32'b0;
        alu_src1=32'b0;
        alu_res2=32'b0;
    end
    if(enable==1'b1)begin
        case (ctrl)
            2'b00:alu_op=in;
            2'b01:alu_src0={{27{in[4]}}, in[4:0]};
            2'b10:alu_src1={{27{in[4]}}, in[4:0]};
            2'b11:alu_res2=alu_res;
            default: alu_op=in;
        endcase
    end
end
```

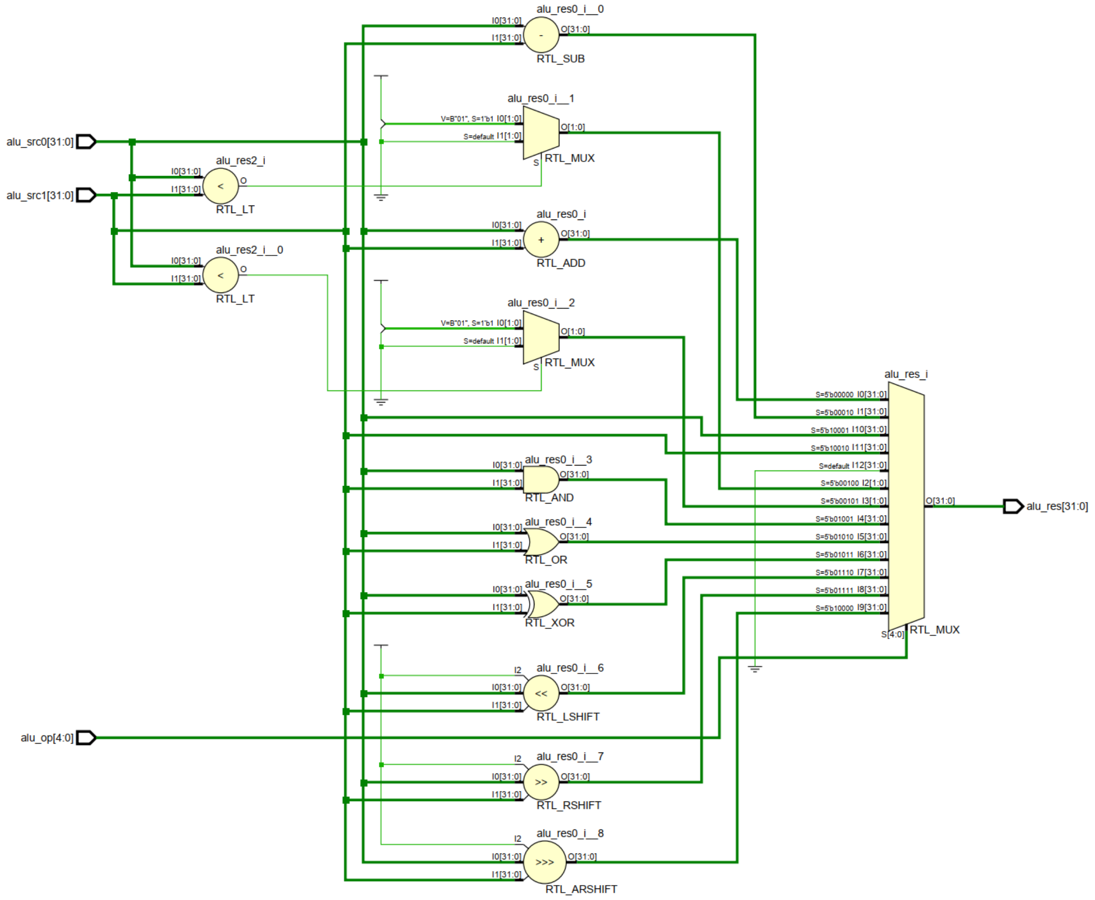
第三部分 实验结果

3.1 电路分析结果

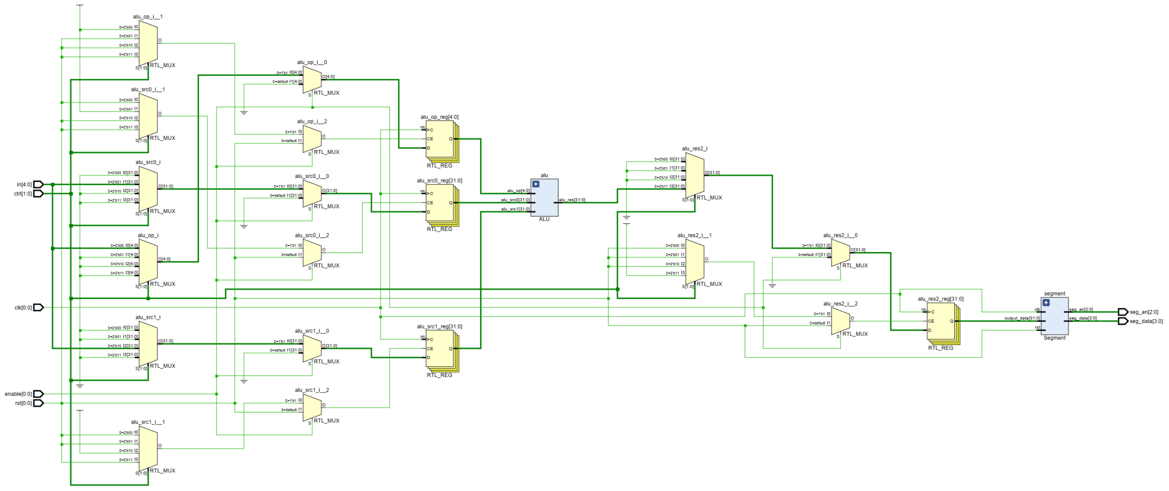
3.1.1 寄存器堆RTL电路：、

文件过大，见实验报告最后一页附图

3.1.2 ALU RTL电路：



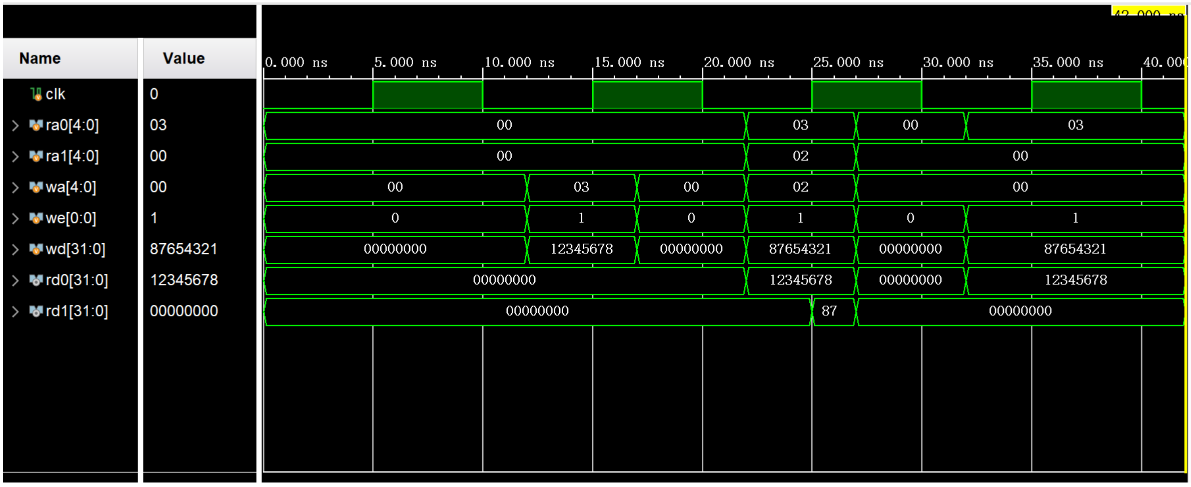
3.1.3 ALU计算器RTL电路：



3.2 电路仿真结果

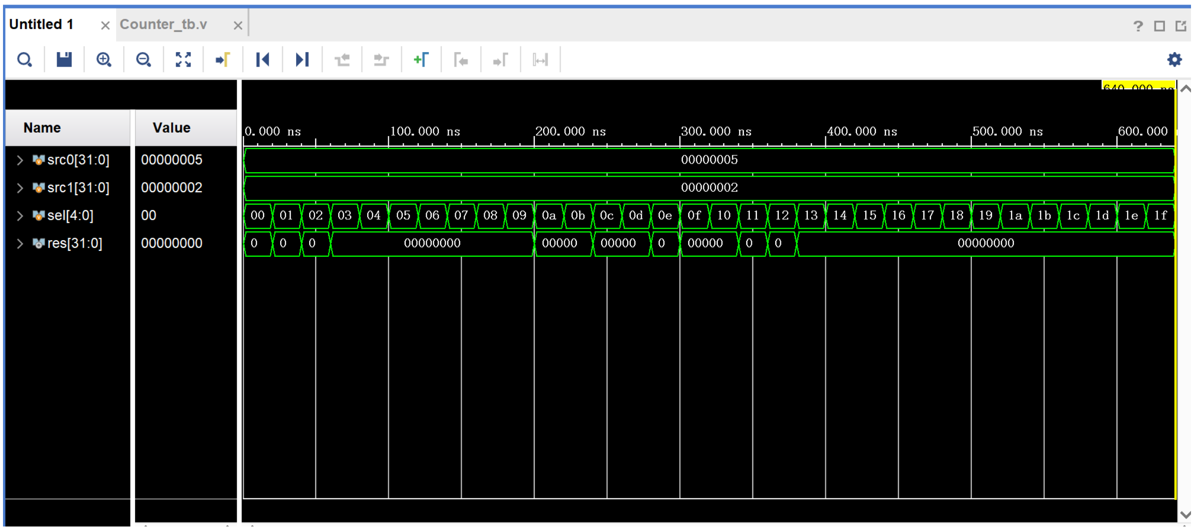
3.2.1 寄存器堆仿真结果

使用参考的仿真文件仿真，结果如下：



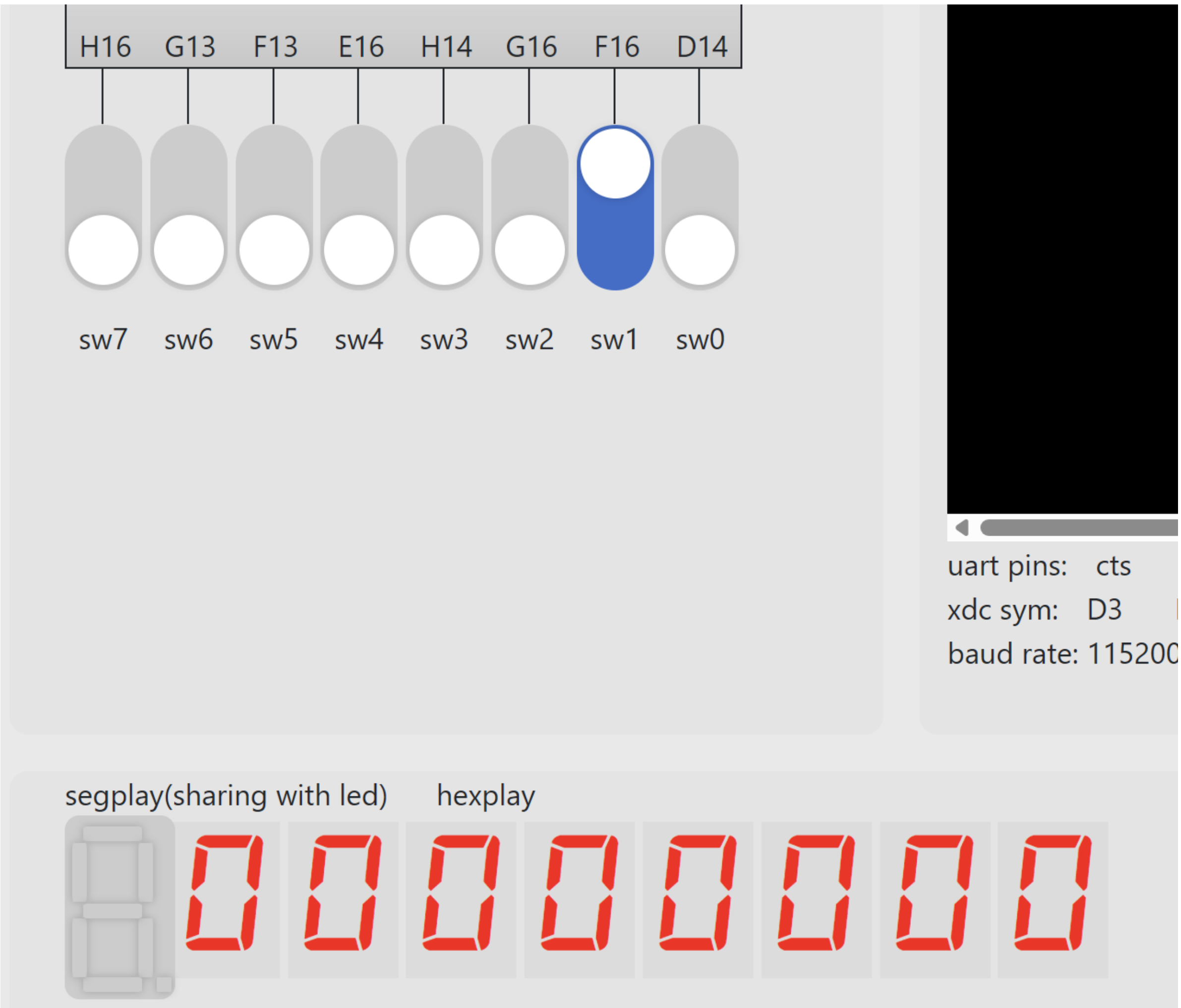
3.2.2 ALU仿真结果

ALU仿真结果如下，仿真文件为将5和2两个数依次进行操作码从00000到11111的运算



3.2.3 ALU计算器仿真结果（FPGAOL平台）

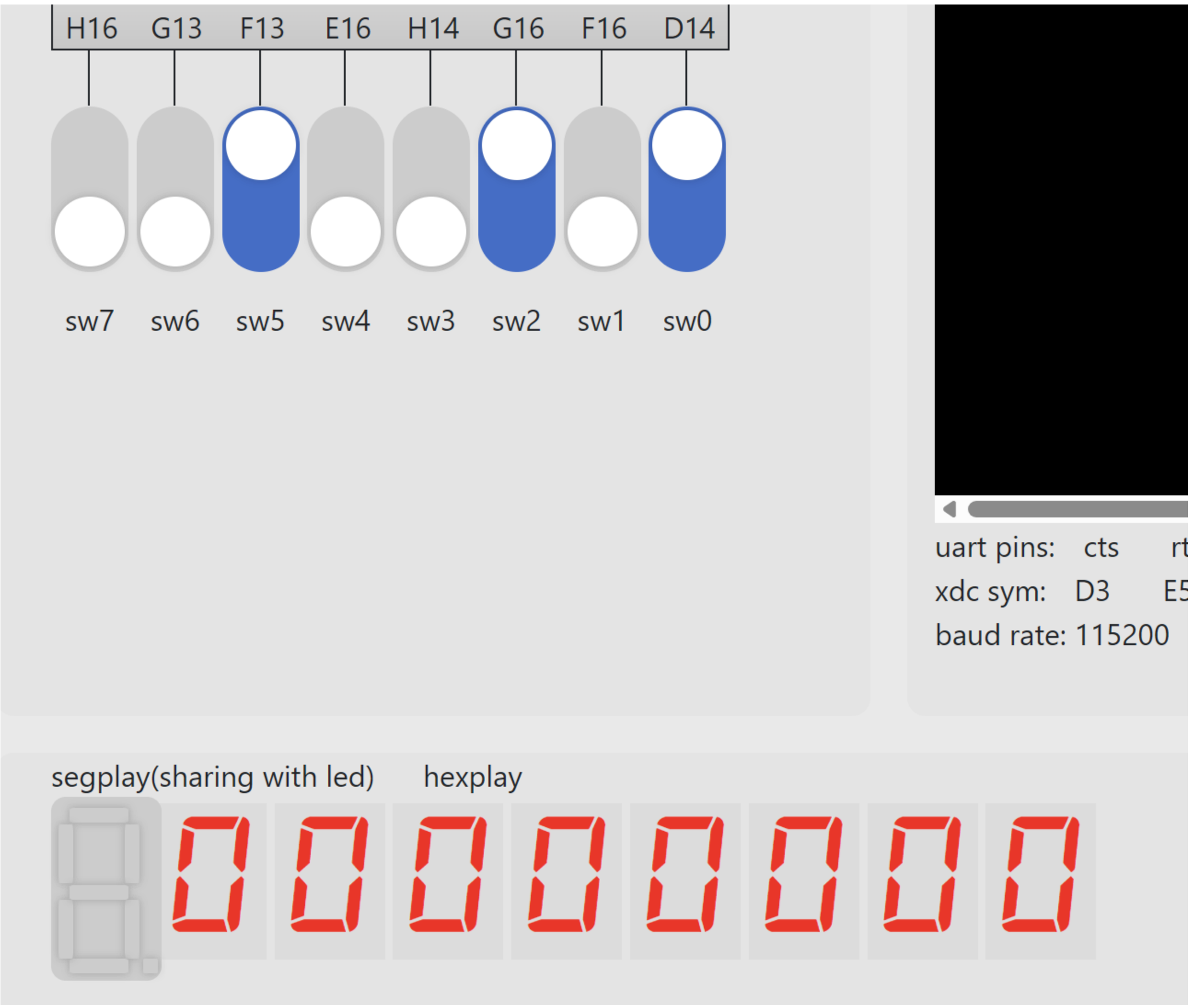
以5-7为例：


H16 G13 F13 E16 H14 G16 F16 D14
sw7 sw6 sw5 sw4 sw3 sw2 sw1 sw0
segplay(sharing with led)
hexplay
00000000

uart pins: cts

xdc sym: D3

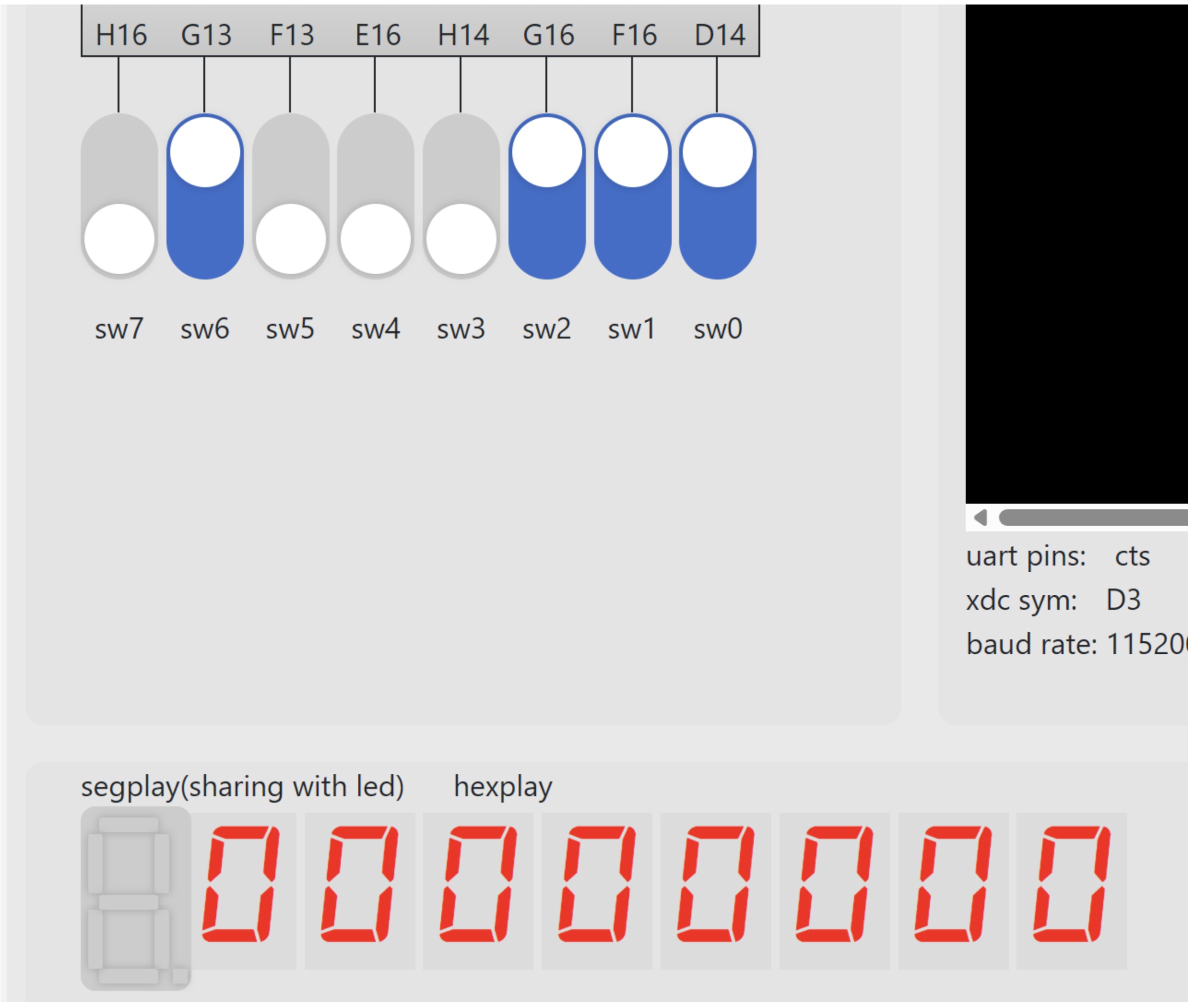
baud rate: 11520


H16 G13 F13 E16 H14 G16 F16 D14
sw7 sw6 sw5 sw4 sw3 sw2 sw1 sw0
segplay(sharing with led)
hexplay
00000000

uart pins: cts rt

xdc sym: D3 E5

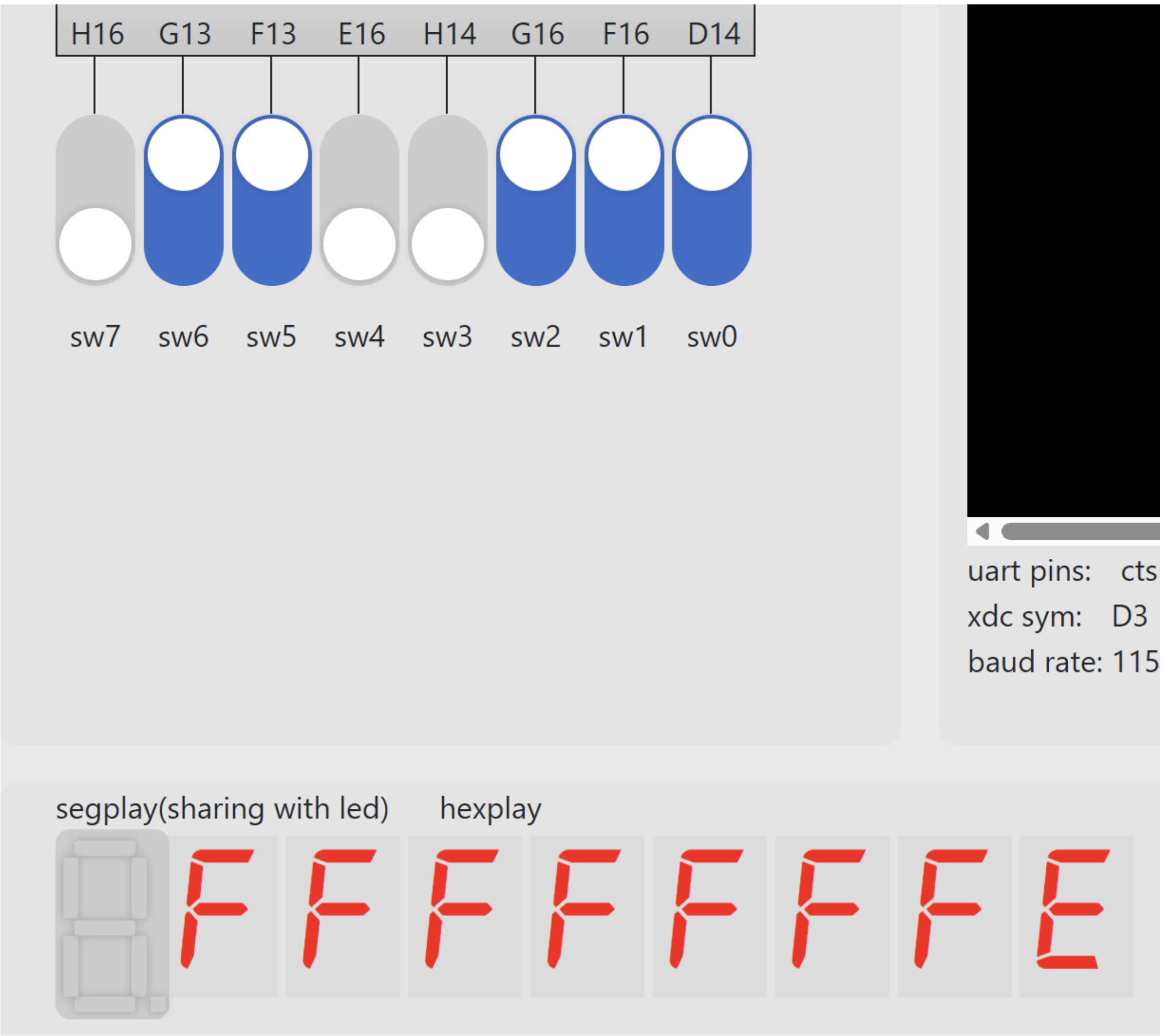
baud rate: 115200


H16 G13 F13 E16 H14 G16 F16 D14
sw7 sw6 sw5 sw4 sw3 sw2 sw1 sw0
segplay(sharing with led)
hexplay
00000000

uart pins: cts

xdc sym: D3

baud rate: 11520

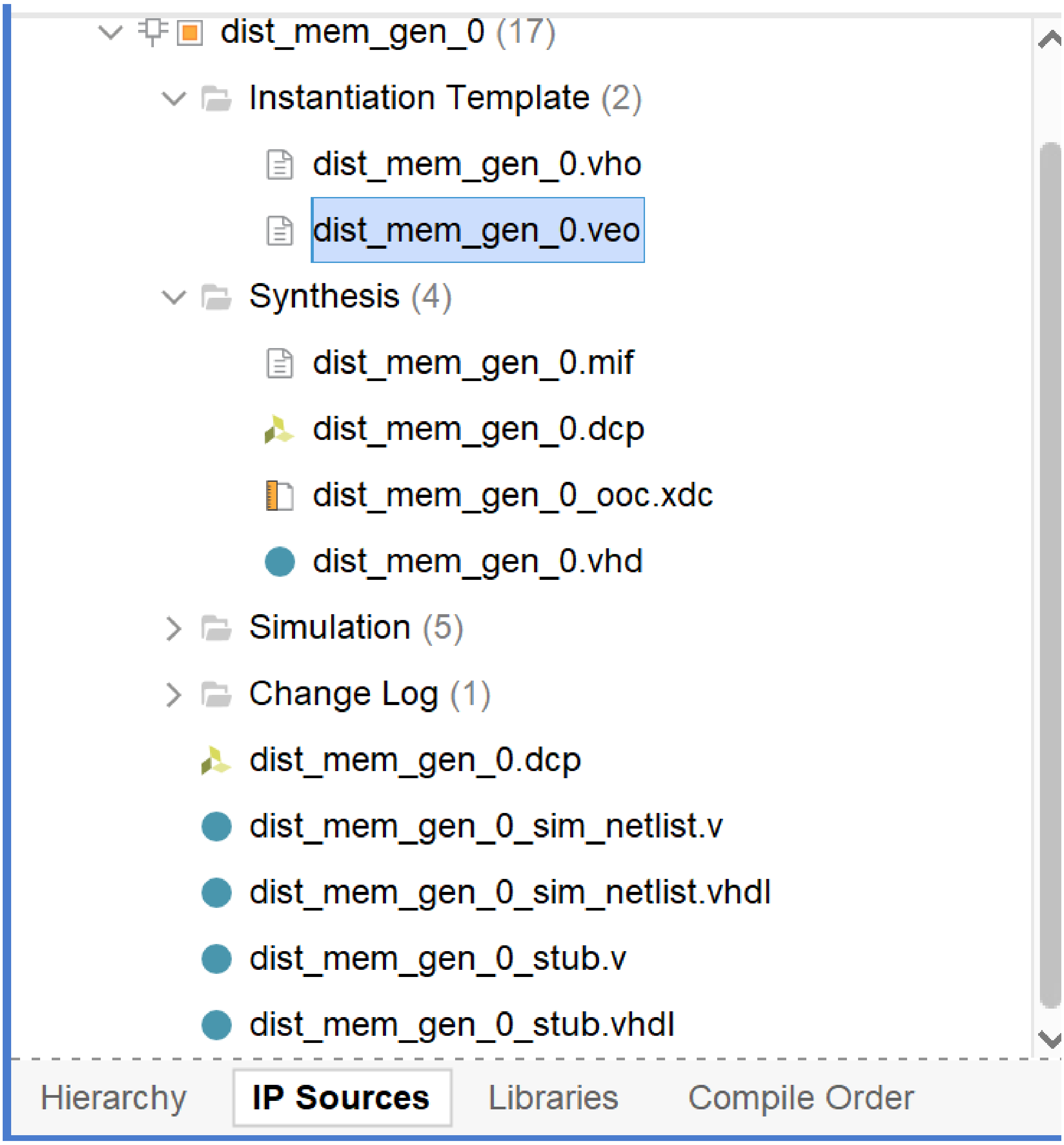

H16 G13 F13 E16 H14 G16 F16 D14
sw7 sw6 sw5 sw4 sw3 sw2 sw1 sw0
segplay(sharing with led)
hexplay
FFFFFFFE

uart pins: cts

xdc sym: D3

baud rate: 115

3.3 存储器IP核例化及初始化



dist_mem_gen_0.veo文件片段：

```
//----------- Begin Cut here for INSTANTIATION Template ---// INST_TAG
dist_mem_gen_0 your_instance_name (
  .a(a),      // input wire [5 : 0] a
  .d(d),      // input wire [15 : 0] d
  .clk(clk),  // input wire clk
  .we(we),    // input wire we
  .spo(spo)  // output wire [15 : 0] spo
);
// INST_TAG_END ------ End INSTANTIATION Template ---------
```

第四部分 心得体会

辅修人首次感受到了verilog的魅力（雾），熟悉了verilog语法，vivado调试和仿真的方法，理解了寄存器堆，ALU的原理，了解了存储器IP核的例化方法。

第五部分 附件

只附上verilog数据通路代码和仿真文件代码，xdc文件省略（因为完全使用的参考代码）

1.regfile.v

```
////////////////////////////////////////////////////////////////////////////////
//
// Company:
// Engineer:
//
// Create Date: 2024/03/31 21:31:05
// Design Name:
// Module Name: Counter
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
////////////////////////////////////////////////////////////////////////////////
//

`timescale 1ns / 1ps
module REG_FILE (
    input                   [ 0 : 0]        clk,

    input                   [ 4 : 0]        rf_ra0,
    input                   [ 4 : 0]        rf_ra1,
    input                   [ 4 : 0]        rf_wa,
    input                   [ 0 : 0]        rf_we,
    input                   [31 : 0]        rf_wd,

    output                  [31 : 0]        rf_rd0,
    output                  [31 : 0]        rf_rd1
);

    reg [31 : 0] reg_file [0 : 31];

    // 用于初始化寄存器
    integer i;
    initial begin
        for (i = 0; i < 32; i = i + 1)
            reg_file[i] = 0;
    end

    assign rf_rd0=reg_file[rf_ra0];
    assign rf_rd1=reg_file[rf_ra1];

    always @(posedge clk) begin
        if (rf_we) begin
```

```
            reg_file[rf_wa] <= rf_wd;
        end
        reg_file[5'b0]<=32'b0;
    end

endmodule
```

2.regfile_tb.v

```
`timescale 1ns / 1ps
////////////////////////////////////////////////////////////////////////////////
//
// Company: 
// Engineer: 
// 
// Create Date: 2024/03/31 21:50:06
// Design Name: 
// Module Name: Counter_tb
// Project Name: 
// Target Devices: 
// Tool Versions: 
// Description: 
// 
// Dependencies: 
// 
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
// 
////////////////////////////////////////////////////////////////////////////////
//


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;

    #5
    ra0 = 5'H0; ra1 = 5'H0; wa = 5'H0; we = 1'H0; wd = 32'H0;

    #5
    ra0 = 5'H3; ra1 = 5'H2; wa = 5'H2; we = 1'H1; wd = 32'H87654321;

    #5
    ra0 = 5'H0; ra1 = 5'H0; wa = 5'H0; we = 1'H0; wd = 32'H0;

    #5
    ra0 = 5'H3; ra1 = 5'H0; wa = 5'H0; we = 1'H1; wd = 32'H87654321;

    #10
    $finish;
end
always #5 clk = ~clk;
endmodule
```

3.ALU.v

```
`timescale 1ns / 1ps
////////////////////////////////////////////////////////////////////////////////
//
// Company:
// Engineer:
//
// Create Date: 2024/04/01 00:53:57
// Design Name:
// Module Name: Counter
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
////////////////////////////////////////////////////////////////////////////////
//


`define ADD                 5'B00000
`define SUB                 5'B00010
`define SLT                 5'B00100
`define SLTU                5'B00101
`define AND                 5'B01001
```

```
`define OR                  5'B01010
`define XOR                 5'B01011
`define SLL                 5'B01110
`define SRL                 5'B01111
`define SRA                 5'B10000
`define SRC0                5'B10001
`define SRC1                5'B10010


module ALU (
    input                   [31 : 0]            alu_src0,
    input                   [31 : 0]            alu_src1,
    input                   [ 4 : 0]            alu_op,

    output      reg         [31 : 0]            alu_res
);
    always @(*) begin
        case(alu_op)
            `ADD :
                alu_res = alu_src0 + alu_src1;
            `SUB :
                alu_res = alu_src0 - alu_src1;
            `SLT :
                alu_res = ($signed(alu_src0) < $signed(alu_src1)) ? 32'b1 :
32'b0;
            `SLTU :
                alu_res = (alu_src0 < alu_src1) ? 32'b1 : 32'b0;
            `AND :
                alu_res = alu_src0 & alu_src1;
            `OR :
                alu_res = alu_src0 | alu_src1;
            `XOR :
                alu_res = alu_src0 ^ alu_src1;
            `SLL :
                alu_res = alu_src0 << alu_src1;
            `SRL :
                alu_res = alu_src0 >> alu_src1;
            `SRA :
                alu_res = $signed(alu_src0) >>> $signed(alu_src1);
            `SRC0 :
                alu_res = alu_src0;
            `SRC1 :
                alu_res = alu_src1;
            default :
                alu_res = 32'H0;
        endcase
    end
endmodule
```

4.ALU_tb.v

```
`timescale 1ns / 1ps
////////////////////////////////////////////////////////////////////////////////
//
// Company:
// Engineer:
//
// Create Date: 2024/04/01 00:58:11
// Design Name:
// Module Name: Counter_tb
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
////////////////////////////////////////////////////////////////////////////////
//

`define ADD                 5'B00000
`define SUB                 5'B00010
`define SLT                 5'B00100
`define SLTU                5'B00101
`define AND                 5'B01001
`define OR                  5'B01010
`define XOR                 5'B01011
`define SLL                 5'B01110
`define SRL                 5'B01111
`define SRA                 5'B10000
`define SRC0                5'B10001
`define SRC1                5'B10010


module ALU (
    input                   [31 : 0]            alu_src0,
    input                   [31 : 0]            alu_src1,
    input                   [ 4 : 0]            alu_op,

    output      reg         [31 : 0]            alu_res
);
    always @(*) begin
        case(alu_op)
            `ADD :
                alu_res = alu_src0 + alu_src1;
            `SUB :
                alu_res = alu_src0 - alu_src1;
            `SLT :
                alu_res = ($signed(alu_src0) < $signed(alu_src1)) ? 32'b1 :
32'b0;
            `SLTU :
                alu_res = (alu_src0 < alu_src1) ? 32'b1 : 32'b0;
            `AND :
                alu_res = alu_src0 & alu_src1;
            `OR :
                alu_res = alu_src0 | alu_src1;
            `XOR :
                alu_res = alu_src0 ^ alu_src1;
            `SLL :
                alu_res = alu_src0 << alu_src1;
            `SRL :
                alu_res = alu_src0 >> alu_src1;
            `SRA :
                alu_res = alu_src0 >>> alu_src1;
            `SRC0 :
                alu_res = alu_src0;
            `SRC1 :
                alu_res = alu_src1;
            default :
                alu_res = 32'H0;
        endcase
    end
endmodule

module ALU_tb();
//...
    reg [31 : 0] src0;
    reg [31 : 0] src1;
    reg [4 : 0] sel;
    wire [31 : 0] res;

    ALU alu(
                .alu_src0(src0),
                .alu_src1(src1),
                .alu_op(sel),
                .alu_res(res)
    );

    initial begin
        src0=32'H5;
        src1=32'H2;
        sel=5'b0;
        repeat(32) begin
            #20;
            sel = sel + 1;
        end
        $finish;
    end
//...
endmodule
```

5.ALU_cal.v

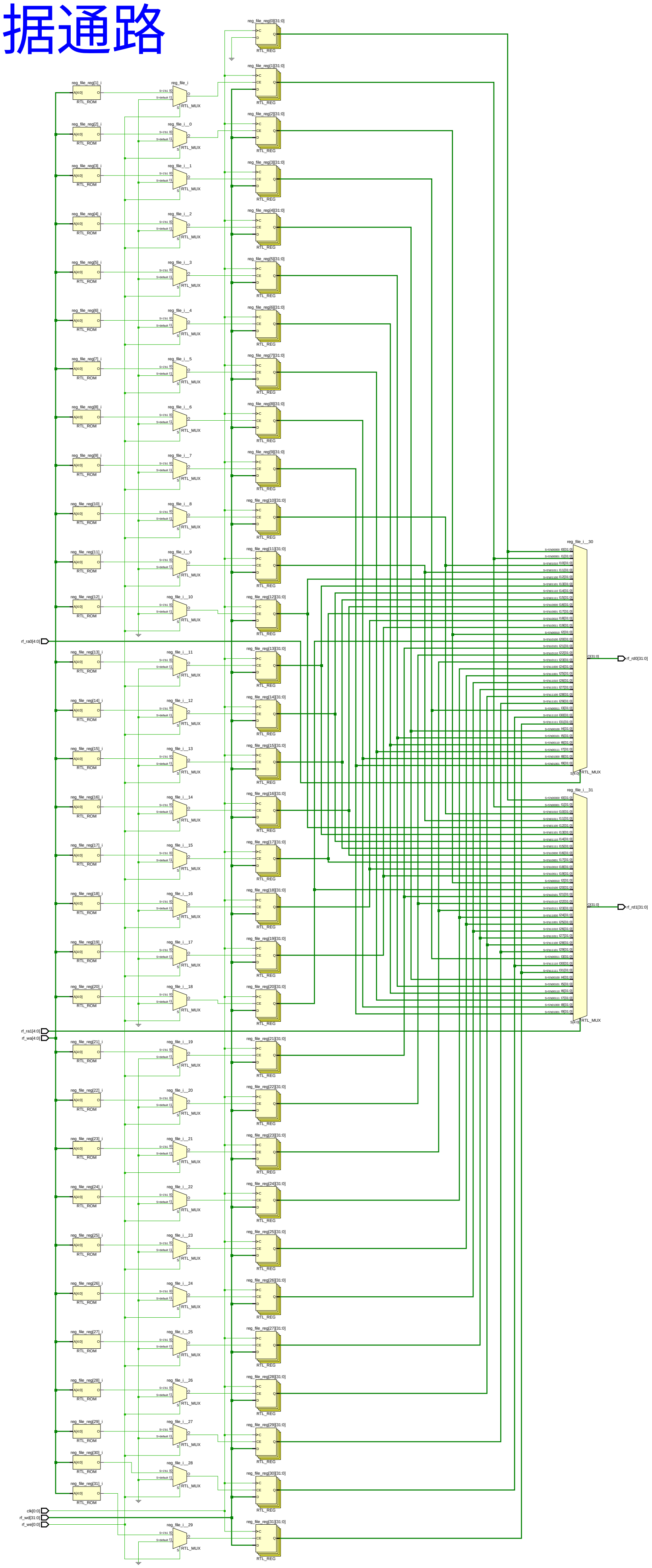
```
`timescale 1ns / 1ps
////////////////////////////////////////////////////////////////////////////////
//
// Company:
```

```verilog
// Engineer:
//
// Create Date: 2024/04/01 16:25:07
// Design Name:
// Module Name: Counter1
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
////////////////////////////////////////////////////////////////////////////////
//


`define ADD                 5'B00000
`define SUB                 5'B00010
`define SLT                 5'B00100
`define SLTU                5'B00101
`define AND                 5'B01001
`define OR                  5'B01010
`define XOR                 5'B01011
`define SLL                 5'B01110
`define SRL                 5'B01111
`define SRA                 5'B10000
`define SRC0                5'B10001
`define SRC1                5'B10010


module TOP (
    input                   [ 0 : 0]            clk,
    input                   [ 0 : 0]            rst,

    input                   [ 0 : 0]            enable,
    input                   [ 4 : 0]            in,
    input                   [ 1 : 0]            ctrl,

    output                  [ 3 : 0]            seg_data,
    output                  [ 2 : 0]            seg_an
);
reg  [4:0]  alu_op;
reg  [31:0] alu_src0;
reg  [31:0] alu_src1;
wire [31:0] alu_res;
reg [31:0] alu_res2;

ALU alu(
    .alu_op(alu_op),
    .alu_src0(alu_src0),
    .alu_src1(alu_src1),
    .alu_res(alu_res)
);
```

```
Segment segment(
    .clk(clk),
    .rst(rst),
    .output_data(alu_res2),
    .seg_data(seg_data),
    .seg_an(seg_an)
);
always @(posedge clk) begin
    if(rst==1'b1)begin
        alu_op=5'b0;
        alu_src0=32'b0;
        alu_src1=32'b0;
        alu_res2=32'b0;
    end
    if(enable==1'b1)begin
        case (ctrl)
            2'b00:alu_op=in;
            2'b01:alu_src0={{27{in[4]}}, in[4:0]};
            2'b10:alu_src1={{27{in[4]}}, in[4:0]};
            2'b11:alu_res2=alu_res;
            default: alu_op=in;
        endcase
    end
end
endmodule

module ALU (
    input                   [31 : 0]            alu_src0,
    input                   [31 : 0]            alu_src1,
    input                   [ 4 : 0]            alu_op,

    output      reg         [31 : 0]            alu_res
);
    always @(*) begin
        case(alu_op)
            `ADD :
                alu_res = alu_src0 + alu_src1;
            `SUB :
                alu_res = alu_src0 - alu_src1;
            `SLT :
                alu_res = ($signed(alu_src0) < $signed(alu_src1)) ? 32'b1 :
32'b0;
            `SLTU :
                alu_res = (alu_src0 < alu_src1) ? 32'b1 : 32'b0;
            `AND :
                alu_res = alu_src0 & alu_src1;
            `OR :
                alu_res = alu_src0 | alu_src1;
            `XOR :
                alu_res = alu_src0 ^ alu_src1;
            `SLL :
                alu_res = alu_src0 << alu_src1;
            `SRL :
                alu_res = alu_src0 >> alu_src1;
            `SRA :
                alu_res = alu_src0 >>> alu_src1;
            `SRC0 :
                alu_res = alu_src0;
            `SRC1 :
                alu_res = alu_src1;
            default :
                alu_res = 32'H0;
        endcase
    end
endmodule

module Segment(
    input                   [ 0 : 0]            clk,
    input                   [ 0 : 0]            rst,
    input                   [31 : 0]            output_data,
    output          reg     [ 3 : 0]            seg_data,
    output          reg     [ 2 : 0]            seg_an
);

parameter COUNT_NUM = 50_000_000 / 400;         // 100MHz to 400Hz
parameter SEG_NUM = 8;                          // Number of segments

reg [31:0] counter;
always @(posedge clk) begin
    if (rst)
        counter <= 0;
    else if (counter >= COUNT_NUM)
        counter <= 0;
    else
        counter <= counter + 1;
end

reg [2:0] seg_id;
always @(posedge clk) begin
    if (rst)
        seg_id <= 0;
    else if (counter == COUNT_NUM) begin
        if (seg_id >= SEG_NUM - 1)
            seg_id <= 0;
        else
            seg_id <= seg_id + 1;
    end
end

always @(*) begin
    seg_data = 0;
    case (seg_an)
        'd0     : seg_data = output_data[3:0];
        'd1     : seg_data = output_data[7:4];
        'd2     : seg_data = output_data[11:8];
        'd3     : seg_data = output_data[15:12];
        'd4     : seg_data = output_data[19:16];
        'd5     : seg_data = output_data[23:20];
        'd6     : seg_data = output_data[27:24];
        'd7     : seg_data = output_data[31:28];
        default : seg_data = 0;
    endcase
end

always @(*) begin
    seg_an = seg_id;
```

```
end
endmodule
```

寄存器堆数据通路


reg_file_reg[0][31:0]
RTL_REG
RTL_ROM
RTL_MUX
rf_ra0[4:0]
rf_ra1[4:0]
rf_wa[4:0]
clk[0:0]
rf_wd[31:0]
rf_we[0:0]
reg_file_i__30
reg_file_i__31
rf_rd0[31:0]
rf_rd1[31:0]