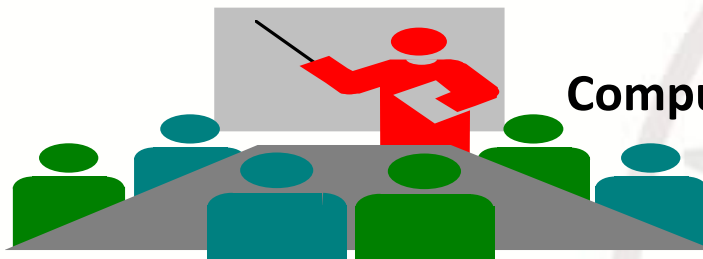




浙江大学
ZHEJIANG UNIVERSITY



Computer Organization & Design

Computer Organization & Design 实验与课程设计

实验四

CPU子核集成替换

---优化CPU调试、测试和应用环境
-逻辑实验模块优化四

施青松

Asso. Prof. Shi Qingsong

College of Computer Science and Technology, Zhejiang University

zjsqs@zju.edu.cn



Course Outline



实验目的



1. 复习寄存器传输控制技术
2. 掌握CPU的核心组成：数据通路与控制
3. 设计数据通路的功能部件
4. 进一步了解计算机系统的基本结构
5. 熟练掌握IP核的使用方法

□ 实验设备

1. 计算机（Intel Core i5以上，4GB内存以上）系统
2. 计算机软硬件课程贯通教学实验系统(Sword)
3. Xilinx ISE14.4及以上开发工具

□ 材料

无

计算机软硬件课程贯通教学实验系统

贯通教学实验平台主要参数

▼ 核心芯片

Xilinx Kintex™-7系列的XC7K160/325资源:

162,240个, Slice: 25350, 片内存储: 11.7Mb

▼ 存储体系 支持32位存储层次体系结构

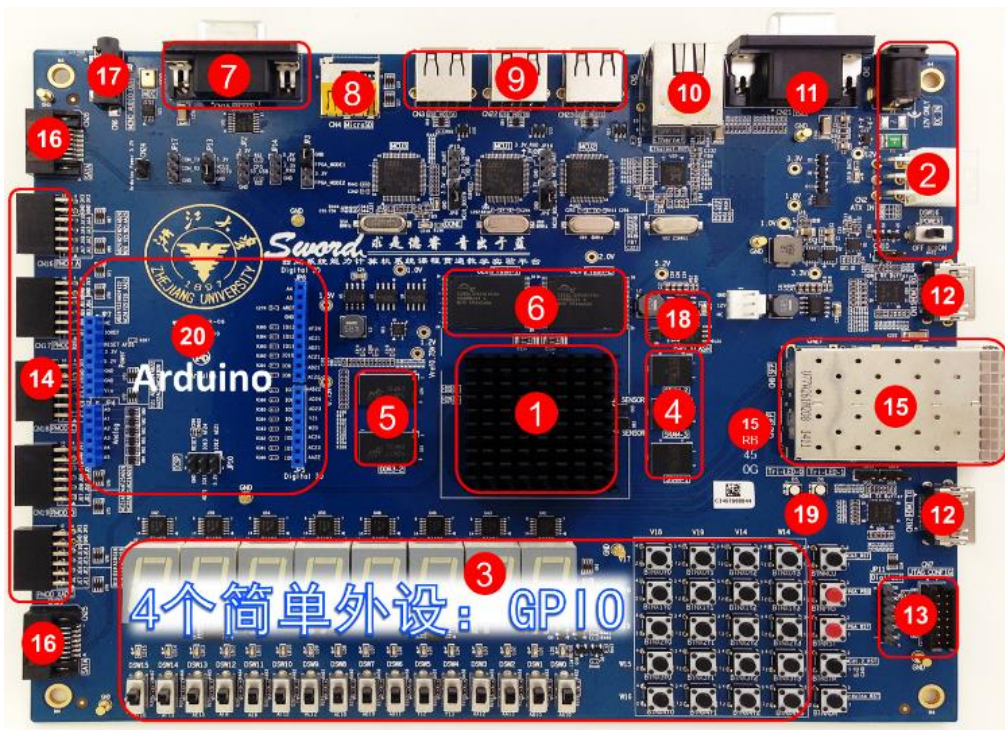
6MB SRAM静态存储器: 支持32Data, 16位TAG

512M BDDR3动态存储: 支持32Data

32MB NOR Flash存储: 支持32位Data

▼ 基本接口 支持微机原理、SOC或微处理器简单应用

4×5+1矩阵按键、16位滑动开关、16位LED、8位七段数码管



▼ 标准接口 支持基本计算机系统实现

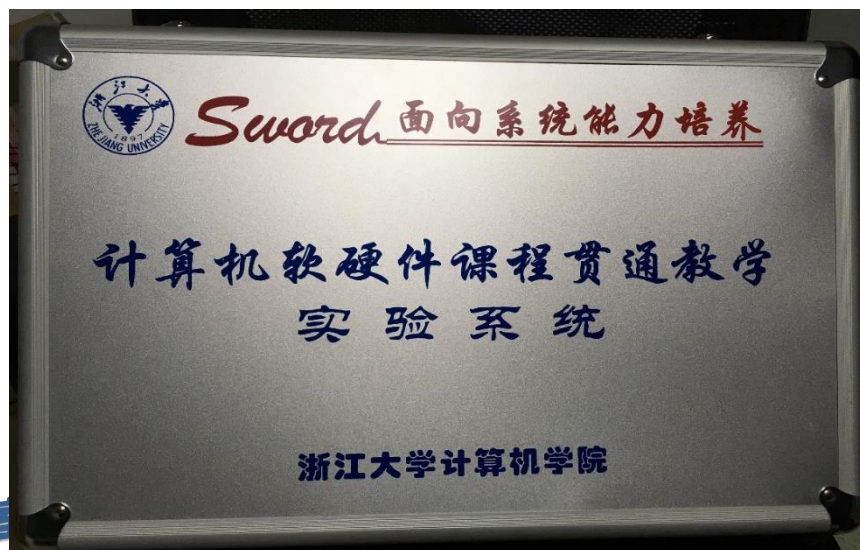
12位VGA接口 (RGB656)、USB-HID (键盘)

▼ 通讯接口 支持数据传输、调试和网络

UART接口、10M/100M/1000M以太网、SFP光纤接口

▼ 扩展接口 支持外存、多媒体和个性化设备

MicroSD(TF)、PMOD、HDMI、Arduino



Course Outline



实验任务



1. 用IP核集成CPU并替换实验三的CPU核

- 选用教材提供的IP核集成实现CPU
- 此实验在Exp03的基础上完成

3. 设计数据通路子部件并作时序仿真:

- ALU
- Register Files

4. 熟练掌握IP核的使用方法

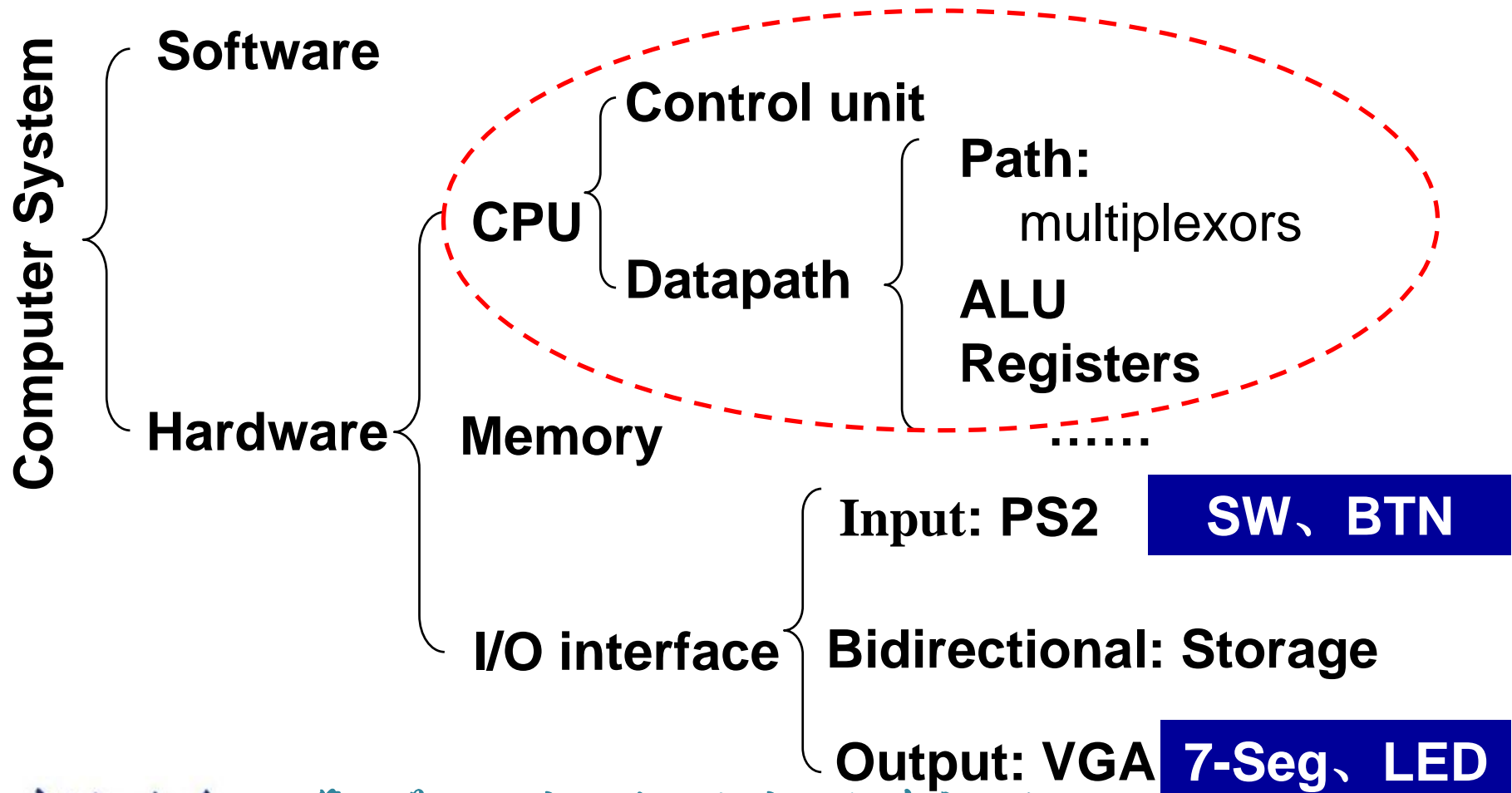


Course Outline



Computer Organization

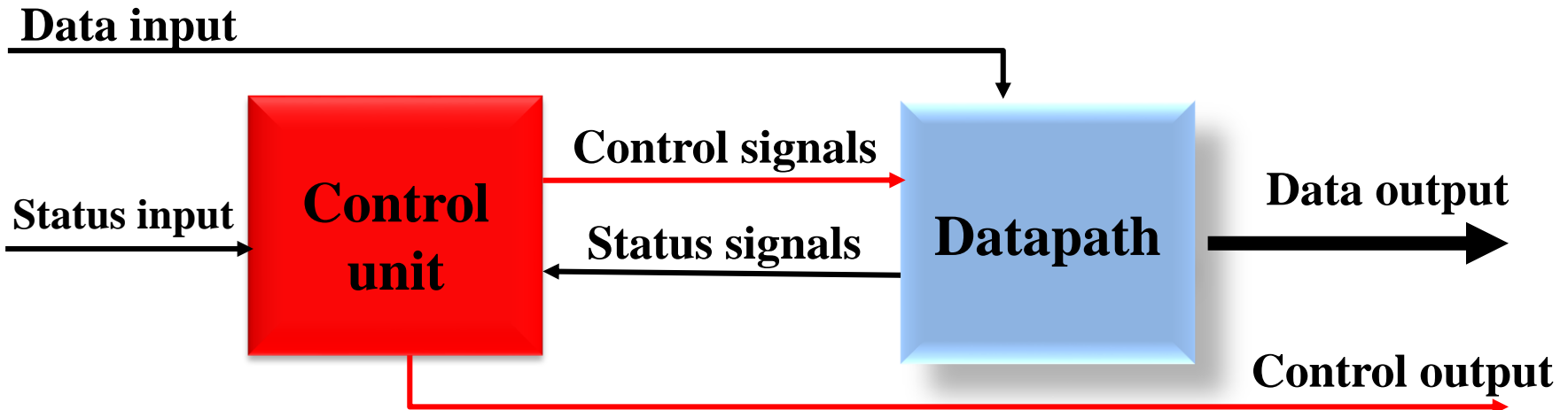
□ Decomposability of computer systems



Digital circuits vs CPU organization

□ Digital circuit

- General circuits that controls logical event with logical gates -
-Hardware



□ Computer organization

- Special circuits that processes logical action with instructions
-Software

CPU部件之1-数据通路:

RSDP9



□ Datapath

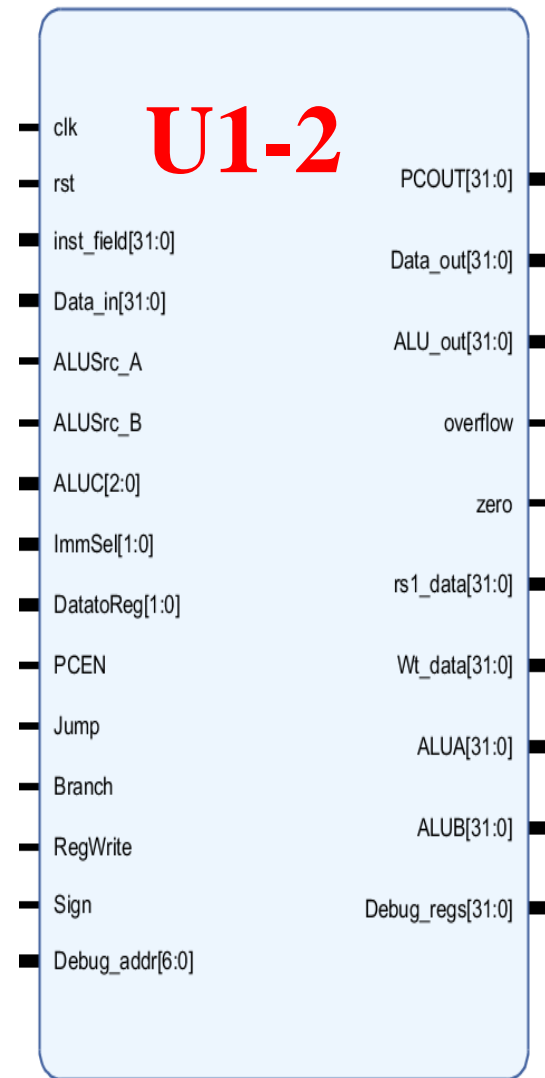
- CPU主要部件之一
- 寄存器传输控制对象: 通用数据通路

□ 基本功能

- 具有通用计算功能的算术逻辑部件
- 具有通用目的寄存器
- 具有通用计数所需的尽可能的路径

□ 本实验用IP 软核- RSDP9

- 核调用模块RSDP9.edf
- 核端口信号定义模块(空文档):
 - RSDP9.v



RSDP9



数据通路端口定义- RSDP9.v

```
module RSDP9(input wire clk,           //寄存器时钟
             input wire rst,           //寄存器复位
             input wire[31:0] inst_field, //指令数据域
             input wire[31:0] Data_in,  //CPU数据输入
             input wire ALUSrc_A,       //寄存器A通道控制
             input wire ALUSrc_B,       //寄存器B通道控制
             input wire[2:0] ALUC,       //ALU操作功能控制
             input wire[1:0] ImmSel,     //RISV-V多立即数选择
             input wire[1:0] DatatoReg,  //REG写数据通道选择
             input PCEN,                 //PC使能信号
             input wire Jump,            //UJ跳转控制
             input wire Branch,          //SB跳转控制
             input wire RegWrite,        //寄存器写信号
             input Sign,                 //符号标志（保留）

             output wire[31:0] PCOUT,     //PC地址输出
             output wire[31:0] Data_out,  //CPU数据输出
             output wire[31:0] ALU_out,   //ALU运算结果输出
             output overflow,             //溢出(保留)
             output zero,                 //ALU操作为"0"

             //Debug signals
             output rsl_data,             //rsl寄存器输出
             output Wt_data,              //寄存器写数据
             output ALUA,                  //ALU A通道输入
             output ALUB,                  //ALU B通道输入
             input [6:0] Debug_addr,      //测试定位
             output [31:0] Debug_regs     //测试、调试信号
);

endmodule
```



CPU部件之2-控制器: SCtrl

□ Controller

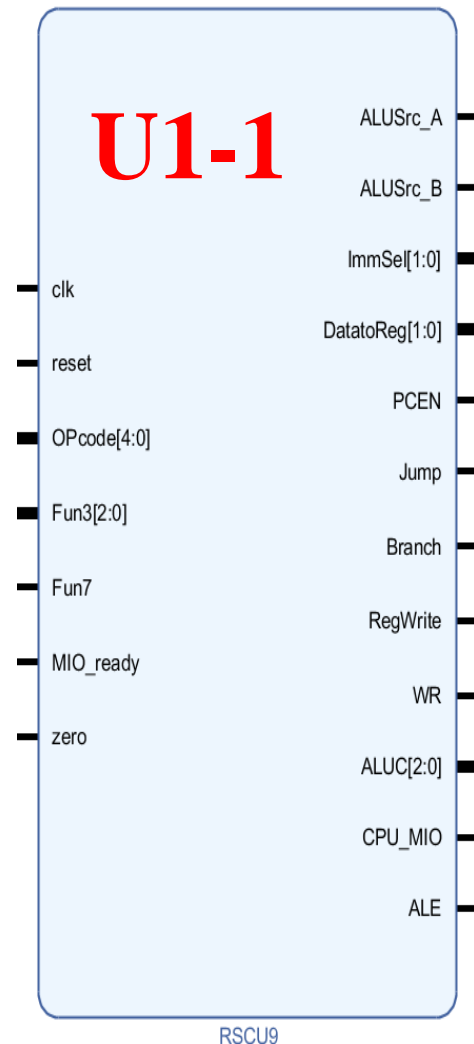
- CPU主要部件之一
- 寄存器传输控制单元:
控制运算和通路控制器

□ 基本功能

- 指令译码
- 产生操作控制信号: ALU运算控制
- 产生指令所需的路径选择

□ 本实验用IP 软核- RSCU9

- 核调用模块RSCU9.edf
- 核端口信号定义模块(空文档):
 - RSCU9.v





控制器端口定义- RSCU9.v

```
module RSCU9(input clk,
             input reset,
             input[4:0]OPcode,           //OPcode: inst[6:2]
             input[2:0]Fun3,             //Function: inst[14:12]
             input Fun7,                 //Function: inst[30]
             input MIO_ready,           //CPU Wait, 复杂时序交互(保留)
             input zero,                //ALU result = 0

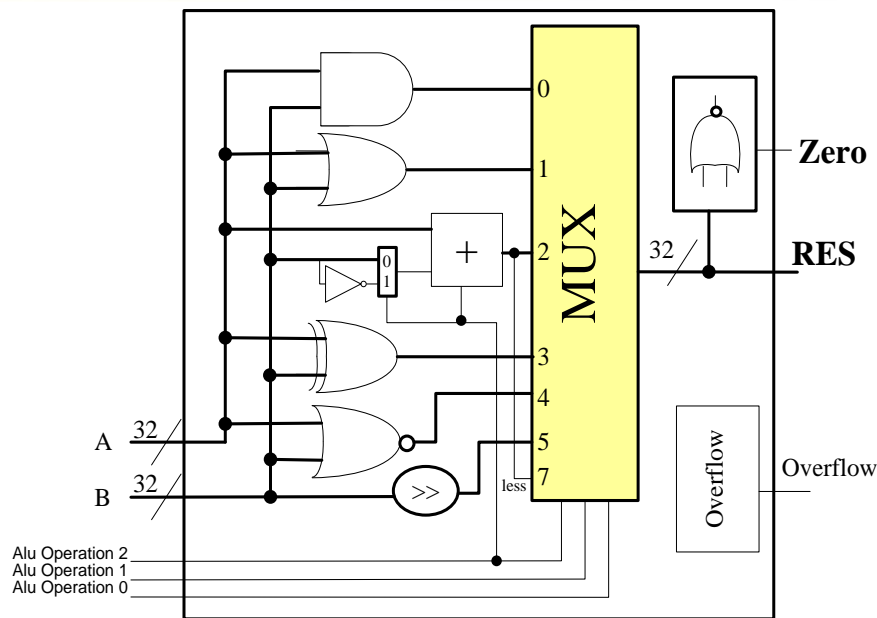
             output reg ALUSrc_A,        //ALU源操作数1选择
             output reg ALUSrc_B,        //ALU源操作数2选择
             output reg [1:0]ImmSel,     //立即数选择控制
             output reg [1:0]DatatoReg,  //写回控制选择
             output PCEN,
             output reg Jump,            //UJ跳转控制
             output reg Branch,          //SB分支跳转控制
             output reg RegWrite,        //寄存器堆写使能
             output reg WR,              //存储器读写使能
             output reg [2:0]ALUC,       //ALU控制
             output reg CPU_MIO,         //存储器操作信号
             output ALE                   //存储器访问有效
);

endmodule
```

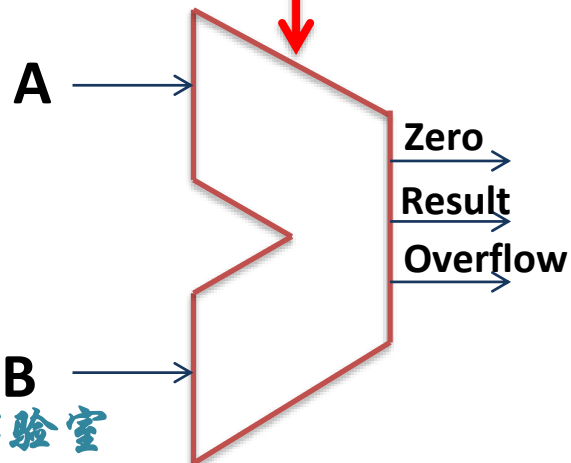

数据通路的功能部件之一：ALU

□ 实现5个基本运算

- 整理逻辑实验八的ALU
- 逻辑图输入并仿真



Alu Operation



ALU Control Lines	Function	note
000	And	兼容
001	Or	兼容
010	Add	兼容
110	Sub	兼容
111	Set on less than	
100	nor	扩展
101	srl	扩展
011	xor	扩展



ALU硬件描述参考代码

```
1  module alu(input [31:0] A,
2             input [31:0] B,
3             input [2:0]ALU_operation,           //ALU_operation后续改为ALUC
4             output reg [31:0] res,
5             output zero,
6             output overflow How do you write with overflow code ?
7             );
8
9  wire[31:0] res_and,res_or,res_add,res_sub,res_nor,res_slt,res_xor,res_srl;
10 parameter one = 32'h00000001, zero_0 = 32'h00000000;
11 assign res_and = A & B;
12 assign res_or  = A | B;
13 assign res_add = A + B;
14 assign res_sub = A - B;
15 assign res_nor = ~(A | B);
16 assign res_slt = (A < B) ? one : zero_0;
17 assign res_xor = A ^ B;
18 assign res_srl = B >> 1;
19
20 always @*
21     case (ALU_operation)
22         3'b000: res=res_and;
23         3'b001: res=res_or;
24         3'b010: res=res_add;
25         3'b110: res=res_sub;
26         3'b100: res=res_nor;
27         3'b111: res=res_slt;
28         3'b011: res=res_xor;
29         3'b101: res=res_srl;
30         default: res=res_add;
31     endcase
32 assign zero = (res==0)? 1: 0;
33 endmodule
```

How do you write with overflow code ?

What is the difference The codes in the Synthesize?

```
wire [31:0] SB = ALUC[2] ? ~B : B; //if ALUC[2]=1 then res_add = A - B;
assign {Co,res_add} = A + SB + C0; //ADC: 符号运算加减器
```

//无符号减法

//无符号比较

```
always @ (*)
case (ALU_operation)
3'b000: res = A & B;
3'b001: res = A | B;
3'b010: res = A + B;
3'b110: res = A - B;
3'b100: res = ~(A | B);
3'b111: res = (A < B) ? one : zero_0;
default: res = 32'hx;
endcase
```

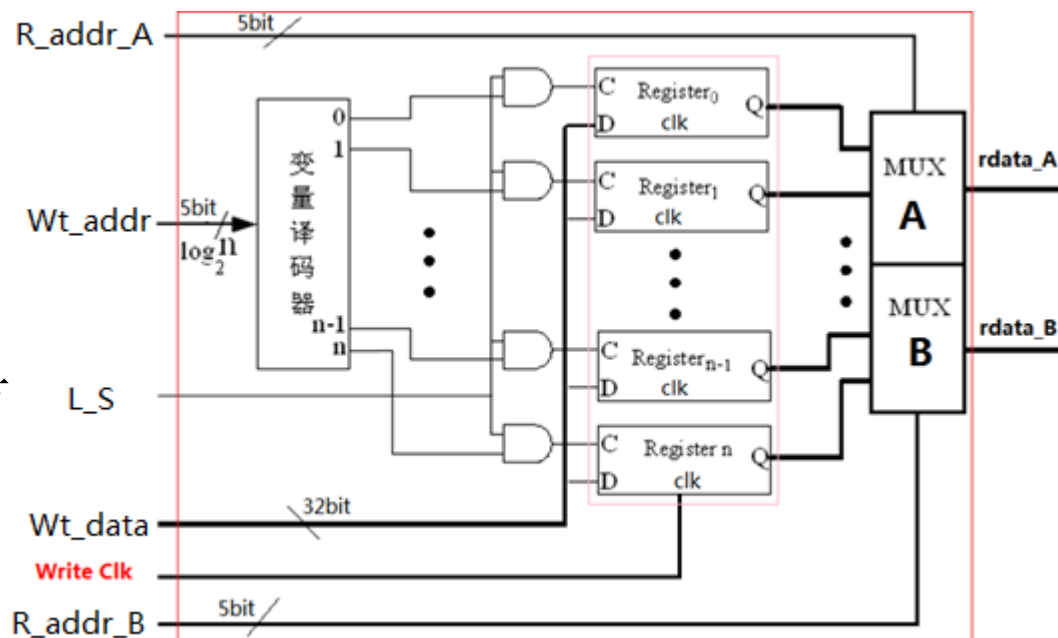
数字系统的功能部件之一：Register files

□ 实现 $32 \times 32\text{bit}$ 寄存器组

- 优化逻辑实验Regs
- 行为描述并仿真结果

□ 端口要求

- 二个读端口：
 - R_addr_A
 - R_addr_B
- 一个写端口，带写信号
 - Wt_addr
 - L_S





非常精练的参考代码

此代码留有BUG，请同学自行编写

```

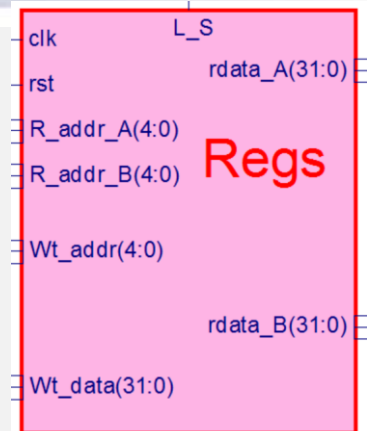
Module regs(input clk, rst, L_S,
            input [4:0] R_addr_A, R_addr_B, Wt_addr,
            input [31:0] wt_data
            output [31:0] rdata_A, rdata_B
            );
reg [31:0] register [1:31];          // r1 - r31
integer i;

assign rdata_A = (Rs_addr_A == 0) ? 0 : register[reg_Rd_addr_A];          // read
assign rdata_B = (Rt_addr_B == 0) ? 0 : register[reg_Rt_addr_B];          // read

always @(posedge clk or posedge rst)
begin if (rst==1) for (i=1; i<32; i=i+1) register[i] <= 0;                // reset
      else if ((Rd_addr != 0) && (we == 1))
          register[Wt_addr] <= wdata;                                     // write
end

assign Debug_regs = (Debug_addr == 0) ? 0 : register[Debug_addr];          //TEST
endmodule

```

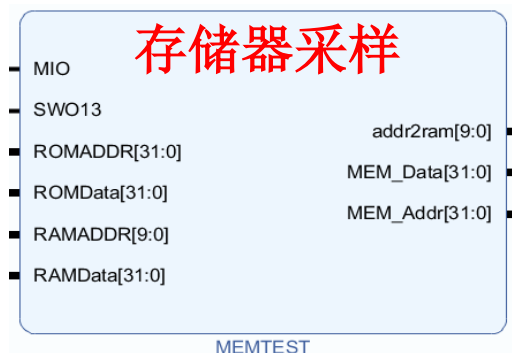


ISE封装逻辑符号

代码来自李亚民教授

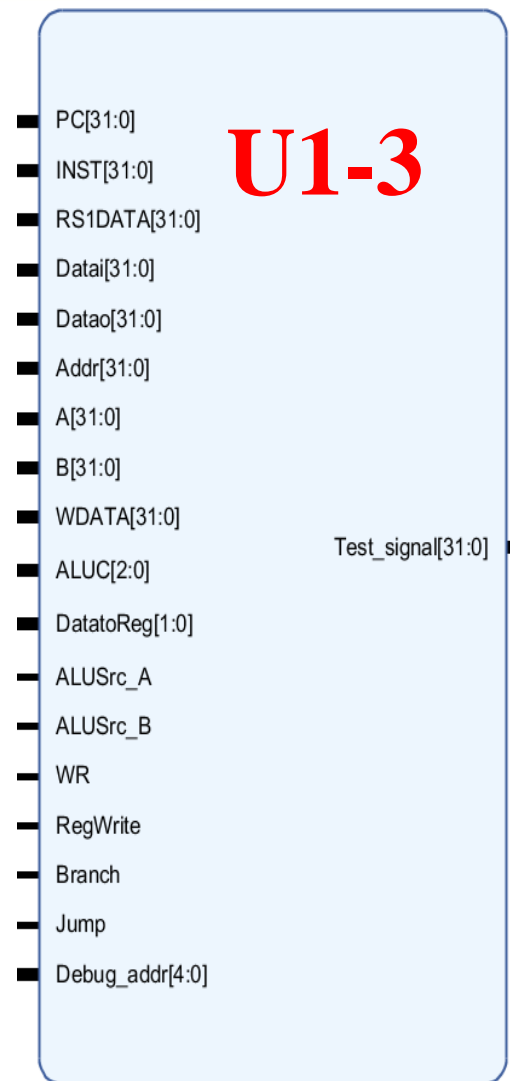
□ CPU测试信号采样

- CPU总线信号
- DP通路数据信号
- DP通路控制信号
- Reg数据(独立)
- 存储器数据(独立)



□ 采样数据定位

- 采样时序由VGA屏幕坐标定位
 - 采样时序(地址)标志: Debug_addr
- 采样信号可根据测试需要调整
 - 时序标志不可轻易修改





测试采样原理

□ 简单原则

- 抽取关键信号显示
- 结合测试程序单步执行。也可自动分析采样数据。
- 尽量不影响CPU状态和控制
 - 以旁路记录为主，干预获取为辅

□ 主要测试信号

- 指令代码：指令是否正常读取和正确
- 指令地址：读取指令与地址是否对应
- 寄存器数据：寄存器数据是指令运行结果重要标志
 - 根据寄存器结果回溯各路信号
- 存储访问信号：存储访问是否正常
- 操作控制：指令每步操作控制信号是否正确
- 操作结果：结果数据、地址和状态是否预期



□ 主要采样输入信号

■ 部分信号内部合成

```
module CPUTEST(input[31:0]PC,           //当前PC指针信号
               input[31:0]INST,         //当前读出指令
               input[31:0]RS1DATA,      //rs1寄存器读出数
               input[31:0]Datai,        //外部输入CPU数据
               input[31:0]Datao,        //CPU输出数据(对应rs2寄存器输出)
               input[31:0]Addr,         //CPU输出地址(对应ALU运算结果)
               input[31:0]A,            //ALU A端口输入数据
               input[31:0]B,            //ALU B端口输入数据
               input[31:0]WDATA,        //寄存器写入数据
               input [2:0]ALUC,         //ALU操作功能编码
               input [1:0]DatatoReg,    //寄存器写通路控制
               input ALUSrc_A,          //寄存器A通道控制
               input ALUSrc_B,          //寄存器B通道控制
               input WR,                //存储器写信号
               input RegWrite,          //寄存器写信号
               input Branch,            //SB转移标志
               input Jump,              //UB转移标志

               input[4:0]Debug_addr,    //采样时序地址
               output reg [31:0] Test_signal //采样输出数据
);
```

CPU采样代码：采样时序

CPUTEST



```
always @* begin
  case (Debug_addr[4:0])
    0: Test_signal = PC;
    1: Test_signal = INST;
    2: Test_signal = {{20{INST[31]}},INST[31:20]}; //imm_12
    3: Test_signal = {{11{INST[31]}},INST[31],INST[19:12],INST[20],INST[30:21],1'b0}}; //UJimm

    4: Test_signal = { {27'b0, INST[19:15]}}; //rs1
    5: Test_signal = RS1DATA; //rs1_data
    6: Test_signal = {{20{INST[31]}}, INST[31:25], INST[11:7]}; //Simm_12
    7: Test_signal = {INST[31:12], 12'h0}; //LU_imm

    8: Test_signal = {27'b0, INST[24:20]}; //rs2
    9: Test_signal = Datao; //Rs2_data
    10: Test_signal = {{19{INST[31]}}, INST[31], INST[7], INST[30:25], INST[11:8],1'b0}}; //SB_imm
    11: Test_signal = {7'h0,WR, 7'h0,RegWrite,13'h0,ALUC}; // control signal

    12: Test_signal = {27'b0, INST[11:7]}; //rd A;
    13: Test_signal = WDATA; //Write:rd-Data
    14: Test_signal = Datai; //MIO to CPU
    15: Test_signal = {7'h0,Branch, 7'h0,Jump, 14'b0, DatatoReg};

    16: Test_signal = A;
    17: Test_signal = Addr; //ALU_out
    18: Test_signal = Datai; //Data to CPU {31'b0, WR};
    19: Test_signal = {27'b0, INST[11:7]}; //Wt

    20: Test_signal = B;
    21: Test_signal = Addr; //CPU Addr
    22: Test_signal = Datao;
    23: Test_signal = WDATA;

    24: Test_signal = {7'b0, ALUSrc_A, 7'b0, ALUSrc_B, 14'b0, DatatoReg};
    default: Test_signal = 32'hAA55_AA55;
  endcase
end

endmodule
```

测试信号可以调整，时序必须与测试模块对应



Regs采样代码

□ 寄存器采样

- 寄存器信号对测试调试至关重要
 - 主动读取采样

```
module Regs(input clk,
            .....
            input [4:0] Debug_addr,           // debug address
            output[31:0] Debug_regs           // debug data
            );

reg [31:0] register [1:31];                  // r1 - r31
integer i;
.....

assign Debug_regs = (Debug_addr == 0) ? 0 : register[Debug_addr]; //TEST

endmodule
```

■ CPU采样合成

```
reg[31:0] Test_signal;
assign Debug_data = Debug_addr[5] ? Test_signal : Debug_regs;
```

↓ CPU采样输出

↑ 送测试电路U11

↑ 寄存器采样输出



采样与VGA屏幕定位

寄存器显示定位

0: Test_signal	1: Test_signal	Mod 4 定位	2: Test_signal	3: Test_signal
4: Test_signal	5: Test_signal		6: Test_signal	7: Test_signal
8: Test_signal	9: Test_signal		10: Test_signal	11: Test_signal
12: Test_signal	13: Test_signal		14: Test_signal	15: Test_signal
16: Test_signal	17: Test_signal		18: Test_signal	19: Test_signal
20: Test_signal	21: Test_signal		22: Test_signal	23: Test_signal
24: 当前指令反汇编	25: Test_signal		26: Test_signal	27: Test_signal
28: Test_signal	29: Test_signal		30: Test_signal	31: Test_signal

存储器显示定位



存储器采样代码

□ 存储器旁路记录采样

- 单元庞大，时序复杂
- 影响存储器正常访问
- 监听记录

- 仅记录访问过的单元
- 影响少，结果需要分析

ROM是独立总线：

仅需监听地址和数据总线

```
module MEMTEST(input MIO,
                input SW013,
                input [31:0]ROMADDR,
                input [31:0]ROMData,
                input [9:0]RAMADDR,
                input [31:0]RAMData,
                output [9:0] addr2ram,
                output [31:0]MEM_Data,
                output [31:0]MEM_Addr
                );
    assign addr2ram = MEM_Addr[11:2];
    assign MEM_Addr = SW013 ? MIO ? {20'h00000, RAMADDR, 2'b00} : 32'hFFFFFFFF
//    assign MEM_Addr = SW013 ? MIO ? RAMADDR : 32'hFFFFFFFF
                                : ROMADDR;
    assign MEM_Data = SW013 ? MIO ? RAMData : 32'hAA55AA55
                                : ROMData;
endmodule
```

RAM采样

SW013: 切换RAM和ROM监听

SW014: 翻页

滤除非存储数据



RISC-V CPU HDL描述结构

```
module RSCPU9( input clk,
               input reset, //rst
               .....
               input [6:0]Debug_addr,
               output [31:0]Debug_data);
```

端口描述

..... 信号(变量)定义描述

```
wire [1:0] ImmSel;
```

```
RSCU9 U1_1(.clk(clk),
```

```
.....
        .RegWrite(RegWrite),
        .WR(WR));
```

控制器调用描述

//MemRW

```
wire[31:0]Debug_regs;
```

```
RSDP9 U1_2(.clk(clk),
```

```
.....
        .Debug_addr(Debug_addr),
        .Debug_regs(Debug_regs));
```

数据通路调用描述

```
//DEBUG TEST:
```

```
wire [31:0] Test_signal;
```

```
assign Debug_data = Debug_addr[5] ? Test_signal : Debug_regs;
```

采样信号合成描述

```
CPUTEST U1_3(.PC(PC),
```

```
INST、RS1DATA、Datai、Datao、Addr
A、B、WDATA、ALUC、DatatoReg
ALUSrc_A、ALUSrc_B、WR、RegWrite
Branch、Jump
```

```
.Debug_addr(Debug_addr[4:0]),
.Test_signal(Test_signal)
```

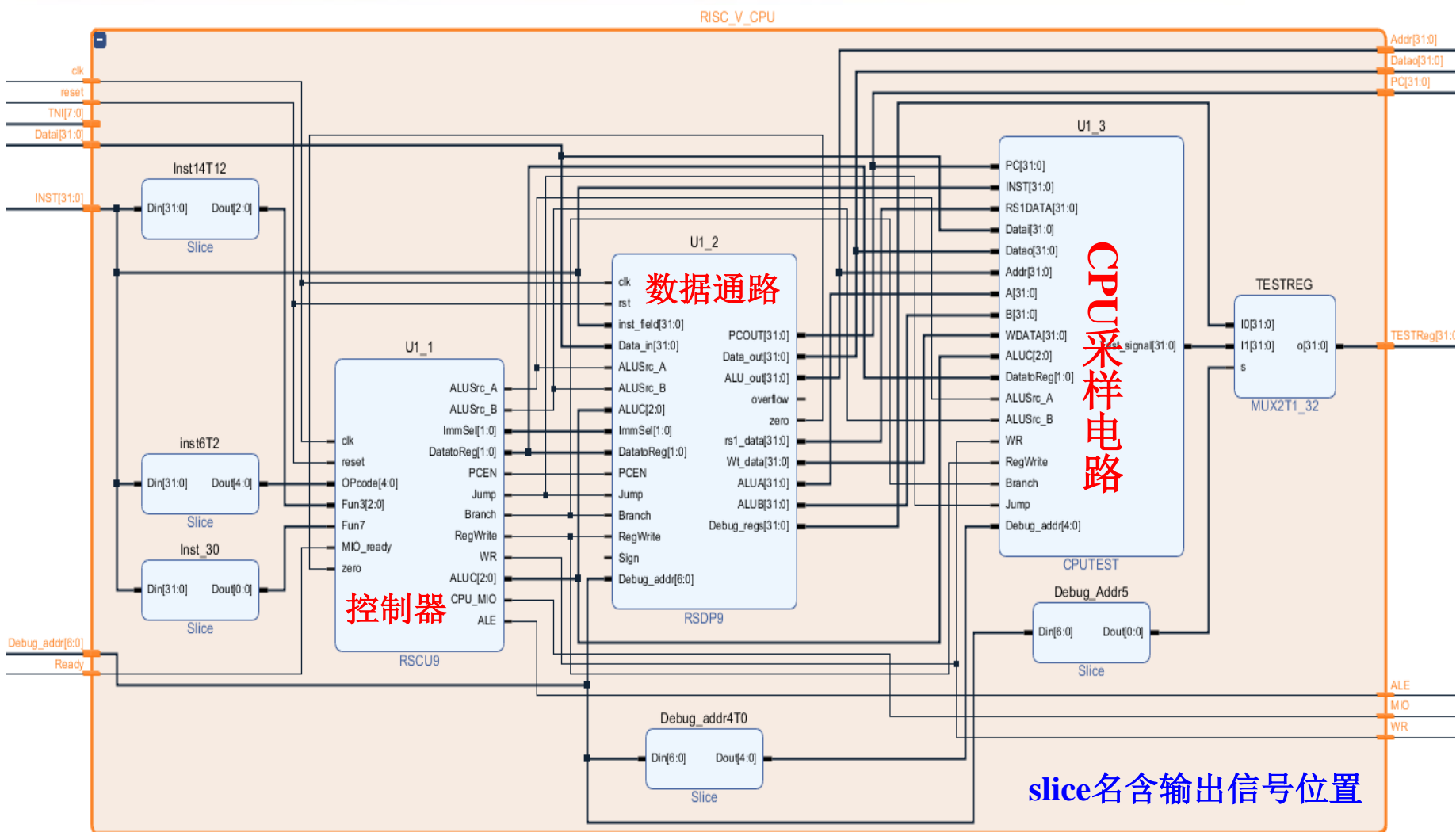
采样调用描述

```
);
```

```
endmodule
```




IP2CPU BD描述结构参考





Course Outline



IP核设计CPU

--用二个第三方IP核集成CPU



设计要点：建立设计工程

◎ 建立工程

☞ OExp04-CSTEH-IPCPU

- ⊙ 在实验三OExp04-CSTEH基础上替换**RCPU9**
- ⊙ 用HDL描述调用EDF核重建ESDC测试环境
- ⊙ 增加CPU测试采样代码描述

◎ 优化数据通路部件

☞ ALU模块优化：修改为符号数、移位操作

☞ Register Files模块优化

- ⊙ 目标：满足RISC-V处理器的要求
- ⊙ 插入REGs采样描述

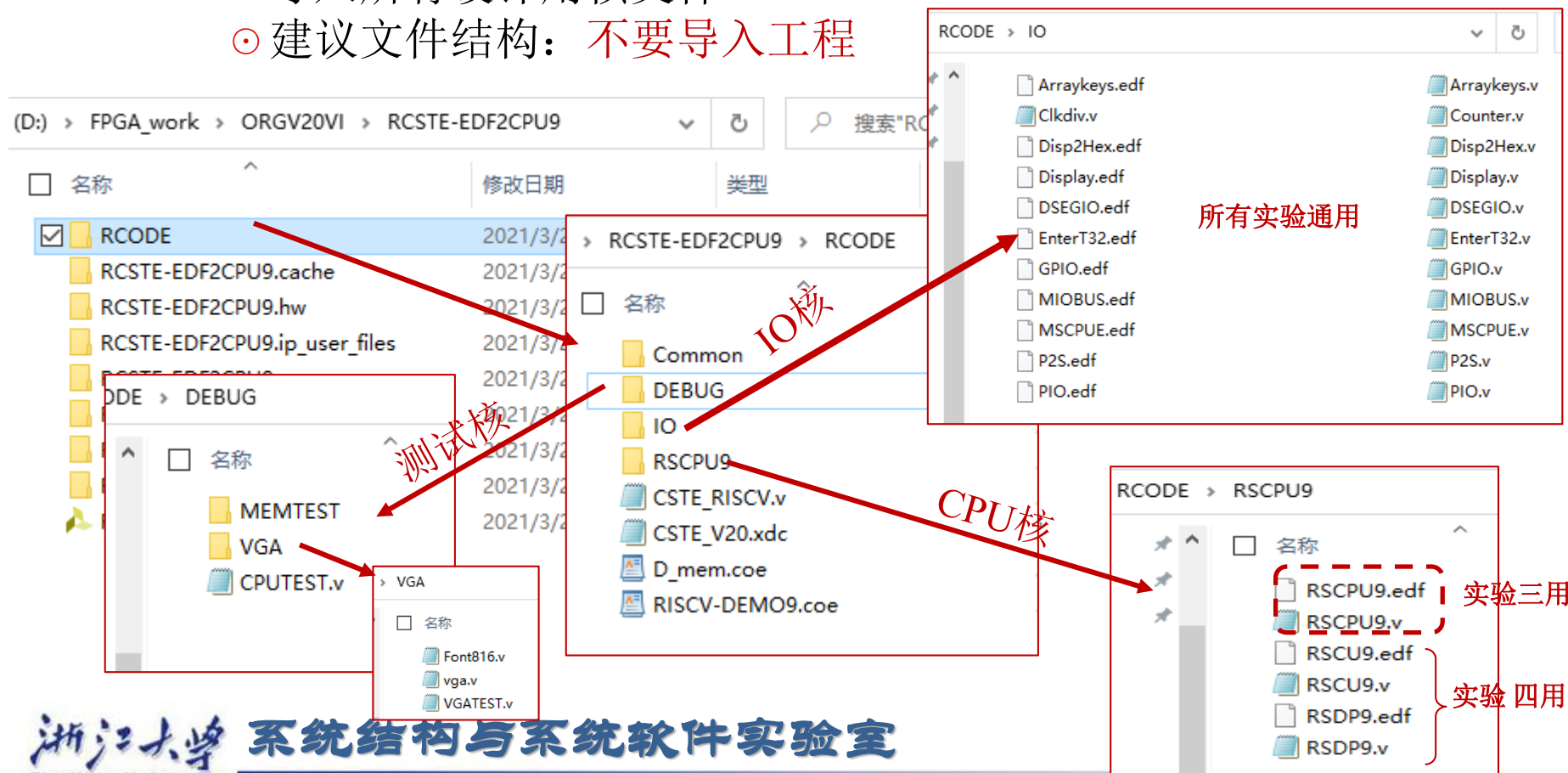
设计要点: HDL描述

◎ 用第三方EDF网表核和EXp01设计核

Ⓔ 在工程根目录建立核和代码文件夹

- 导入所有设计用核文件
- 建议文件结构: **不要导入工程**

实验四有更新



所有实验通用

IO核

CPU核

测试核

实验三用

实验四用

设计要点：BD描述



每次建立新工程后需将核加载到系统！

- 从学在浙大下载IPCORE核压缩包
- 解压到自己计算机的FPGA工作目录下：
 - 非工程目录，可与工程目录并级
 - 根据实验需要逐步提供教学核(有更新)
 - 若用自己设计的核，则核命名：同名核+加学号后四位

工作目录

MYIP ← 自己设计的核目录
ORGIP
ZPORT

自定义接口模板

课程提供的核目录

VGA_work > ORGV20VI > IPCORE > ORGIP

ArrayKeys
Counter
Disp2Hex
DIVOUT
EnterT32
MIOBUS
P2S
RSCPUE
TESTOUT

CLKDIV
DEBUG
Display
DSEGIO
GPIO
MSCPUE
PIO
SWOUT
VGA

课程核(有更新):
每个核1个目录

- 复制相关文档到工程根目录

- 存储器初始文件
- 引脚约束文件复制



设计要点：存储模块：ROM

◎ 设计32位指令存储器：

☞ SWORD实验平台 ROM用Distributed Memory

□ ROM初始化文件(RISCV-DEMO9.coe)

□ 这是一段功能测试程序：CPU仿真另行设计

```
memory_initialization_radix=16;  
memory_initialization_vector=  
0200006F,00000033,00000033,00000033,00000033,00000033,00000033,00000033,00C02283,00502333,  
006303B3,00638E33,00738733,01CE02B3,005282B3,01C28EB3,01DE8F33,01EF0F33,01CF0433,01EF0F33,  
01EF0F33,01DF0FB3,01EF0F33,01EF0F33,01EF0F33,01EF0F33,01EF0F33,01EF0F33,01EF0F33,01EF0F33,  
01EF0F33,01EF0F33,01EF0F33,01EF0F33,01EF0F33,01EF0F33,01EF0F33,01EF0F33,01EF0F33,01EF0F33,  
01EF0F33,01EF0F33,01EF0F33,01EF0F33,01EF0F33,01EF04B3,01E4E633,00948933,012902B3,005282B3,  
406006B3,00C4A223,0004A583,00B585B3,00B585B3,00B4A023,006A8AB3,01592023,01402B03,0004A583,  
00B585B3,00B585B3,00B4A023,0004A583,0055FC33,006B0B33,040B0E63,0004A583,00E70BB3,017B8CB3,  
019B8BB3,0175FC33,000C0C63,037C0463,00E70BB3,037C0663,01592023,FB9FF06F,00D78463,0080006F,  
00D687B3,00F92023,FA5FF06F,0609AA83,01592023,F99FF06F,0209AA83,01592023,F8DFF06F,01402B03,  
00F787B3,0067E7B3,00E989B3,0089F9B3,006A8AB3,00DA8463,00C0006F,00E00AB3,006A8AB3,0004A583,  
00B58C33,018C0C33,0184A023,00C4A223,F6DFF06F;
```

注意：这是RISC-V代码与OExp02不同





设计要点存储模块：RAM

◎ 设计32位数据存储器：

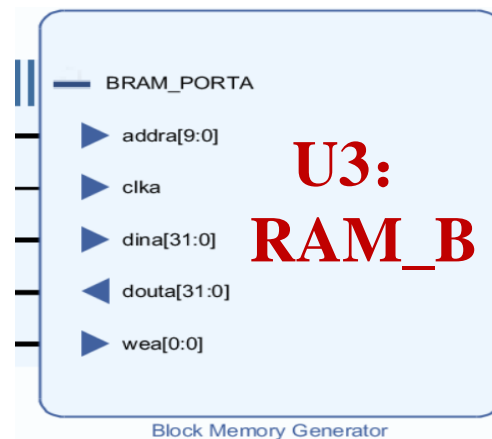
☞ SWORD实验平台 RAM用Block Memory

☞ RAM初始化数据：D_mem.coe

注意：清除BLOCK MEMORY所有输入输出寄存器

```
memory_initialization_radix=16;  
memory_initialization_vector=  
00000000, 11111111, 22222222, 33333333, 44444444, 55555555,  
66666666, 77777777, 88888888, 99999999, aaaaaaaa, bbbbbbbb,  
cccccccc, dddddddd, eeeeeeee, ffffffff, 557EF7E0, D7BDFBD9,  
D7BDFBD9, DFCFFCFB, DFCFBFFF, F7F3DFFF, FFFFDF3D, FFFF9DB9,  
FFFFBCFB, DFCFFCFB, DFCFBFFF, D7DB9FFF, D7BDFBD9, D7BDFBD9,  
FFFF07E0, 007E0FFF, 03bdf020, 03def820, 08002300;
```

RAM初始化数据。红色数据为七段LED图形



☞ 设计流程参考马德老师Lab00PPT



设计要点：硬件描述输入

◎ 阅读核接口要求

☞ 参考实验2~3PPT(本实验核接口有调整)

◎ 根据接口属性和测试环境要求输入描述

☞ 参考实验二连接示意和PDF文档输入描述

☞ 理解每一个语句或连线的意义和目的

◎ 实验三的BLOCK DESIGN描述有调整

☞ 目前BD描述结构经测试正确

◎ Vivadoa工具BD描述或存在BUG或属性还不了解，存在描述综合不稳定性。根据下载验证排查问题。

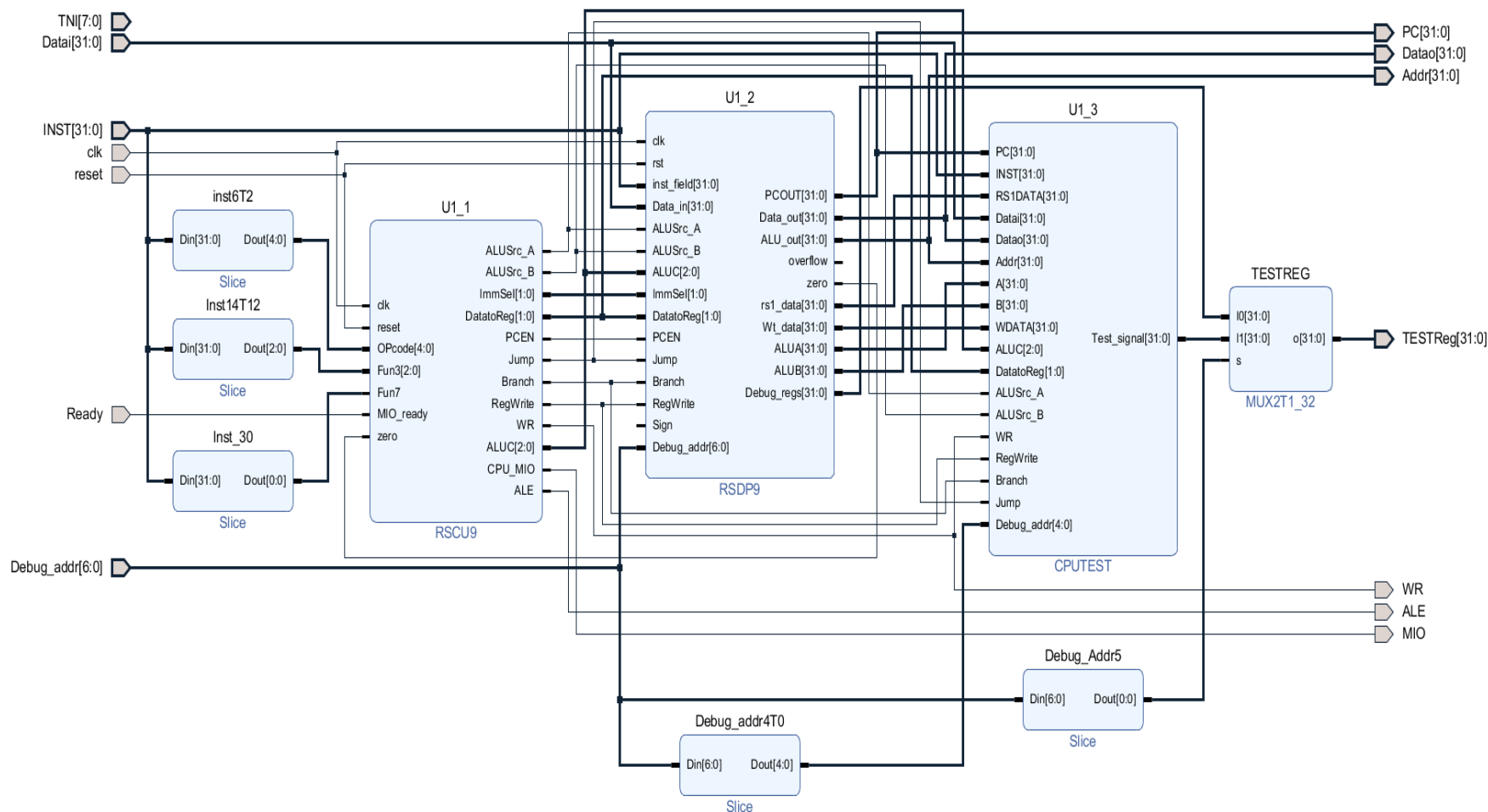
◎ 实验四修改了部分核代码

☞ 设计描述时注意



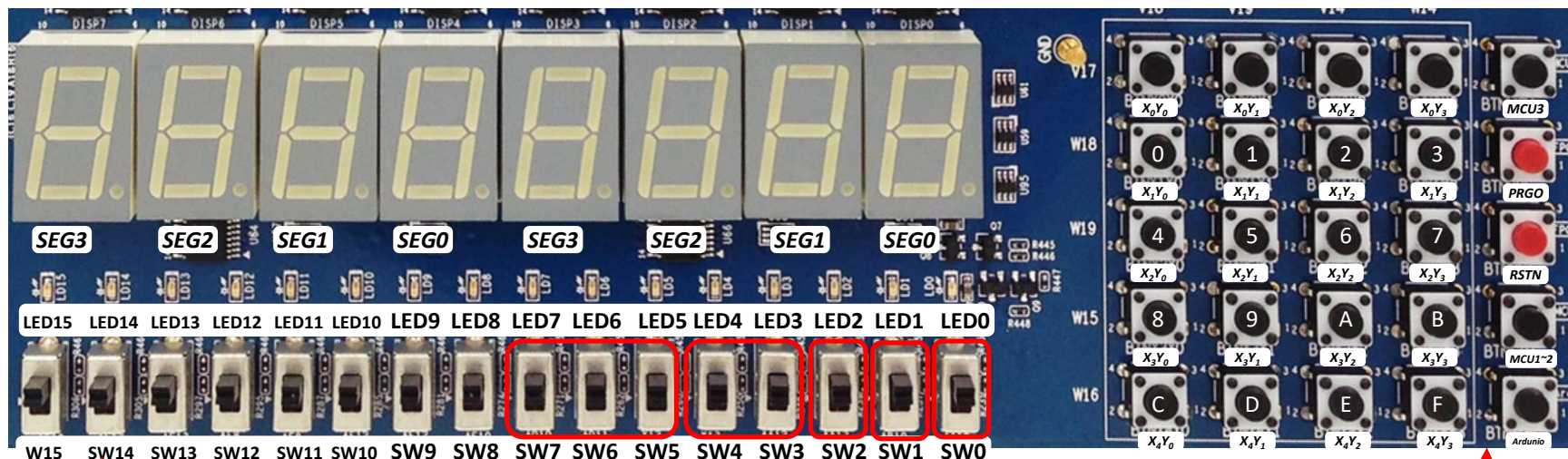
设计要点：实验四设计主要内容结构

□ 用HDL描述实现



设计要点：物理验证接口（详细参见实验二）

物理验证同实验二



SW[0]=文本图形选择

SW[1]=高低16位选择

SW[2]=CPU单步时钟选择

没有使用

SW[4:3]=00, 点阵显示程序：跑马灯
SW[4:3]=00, 点阵显示程序：矩形变幻
SW[4:3]=01, 内存数据显示程序：0~F
SW[4:3]=10, 当前寄存器+1显示

SW[7:5]=显示通道选择
SW[7:5]=000: CPU程序运行输出
SW[7:5]=001: 测试PC字地址
SW[7:5]=010: 测试指令字
SW[7:5]=011: 测试计数器
SW[7:5]=100: 测试RAM地址
SW[7:5]=101: 测试CPU数据输出
SW[7:5]=110: 测试CPU数据输入

SW[13]=0选择测试ROM
SW[13]=1选择测试RAM
SW[14]=0/1测试数据翻页



实验四的测试显示

Zhejiang University Computer Organization Experimental SOC Test Environment (With RISC-V)

x0: zero	00000000	x01: ra	00000000	x02: sp	00000000	x03: gp	00000000
x04: tp	00000000	x05: t0	80000000	x06: t1	00000001	x07: t2	00000002
x8: fps0	0000003F	x09: s1	F0000000	x10: a0	00000000	x11: a1	80000004
x12: a2	F8000000	x13: a3	FFFFFFFF	x14: a4	00000004	x15: a5	FFFFFFBFF
x16: a6	00000000	x17: a7	00000000	x18: s2	E0000000	x19: s3	00000028
x20: s4	00000000	x21: s5	0000314B	x22: s6	FFFE8FBD	x23: s7	00000018
x24: s8	80000000	x25: s9	00000010	x26: s10	00000000	x27: s11	00000000
x28: t3	00000003	x29: t4	0000000F	x30: t5	78000000	x31: t6	000000FF
I-Point	0000010C	I--CODE	0004A583	Imm--12	00000000	UJim-20	0004A000
rs1Addr	00000009	rs1Data	F0000000	SImm-12	0000000B	LUim-20	0004A000
rs2Addr	00000000	rs2Data	00000000	SBim-12	0000080A	W-R-AUC	00010002
rd-Addr	0000000B	rd/W-Da	80000004	MIO-CPU	80000004	B/J-D2R	00000001
ALU-Ain	F0000000	ALU-out	F0000000	CPU-Dai	80000004	WB-Addr	0000000B
ALU-Bin	00000000	CPUAddr	F0000000	CPU-Dao	00000000	WB-DATA	80000004
lw x0B, x09, 000H		-----	AA55AA55	-----	AA55AA55	-----	AA55AA55
RESERVE	AA55AA55	RESERVE	AA55AA55	RESERVE	AA55AA55	RESERVE	AA55AA55
CODE-00	02002063	CODE-01	00000000	CODE-02	00000000	CODE-03	00000000
CODE-04	00000000	CODE-05	00000E00	CODE-06	00000000	CODE-07	00000000
CODE-08	00C02283	CODE-09	0043AE33	CODE-0A	006303B3	CODE-0B	00638E33
CODE-0C	00738733	CODE-0D	01C282B3	CODE-0E	005282B3	CODE-0F	01C28EB3
CODE-10	01DE8F33	CODE-11	01EF0F33	CODE-12	01CF0433	CODE-13	01EF0F33
CODE-14	01EF0F33	CODE-15	01CF0533	CODE-16	01EF0F33	CODE-17	01EF0F33
CODE-18	01EF0F33	CODE-19	01EF0F33	CODE-1A	01EF0F33	CODE-1B	01EF0F33
CODE-1C	01EF0F33	CODE-1D	01EF0F03	CODE-1E	01EF0F33	CODE-1F	01EF0F33
CODE-20	01EF0F33	CODE-21	01EF0F33	CODE-22	01EF0F33	CODE-23	01EF0F33
CODE-24	01EF0F33	CODE-25	01F70933	CODE-26	01EF0F33	CODE-27	01EF0F33



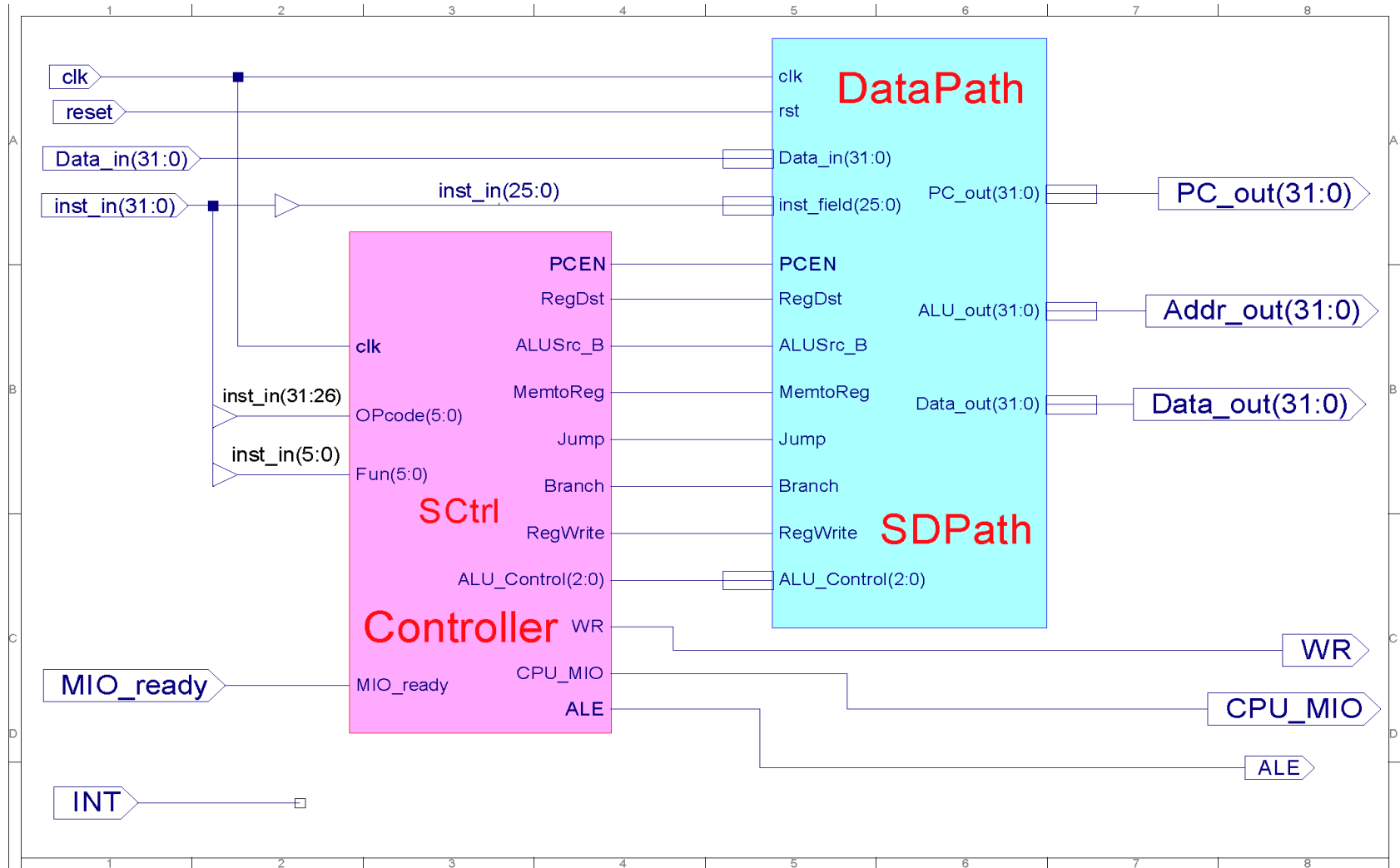
● END



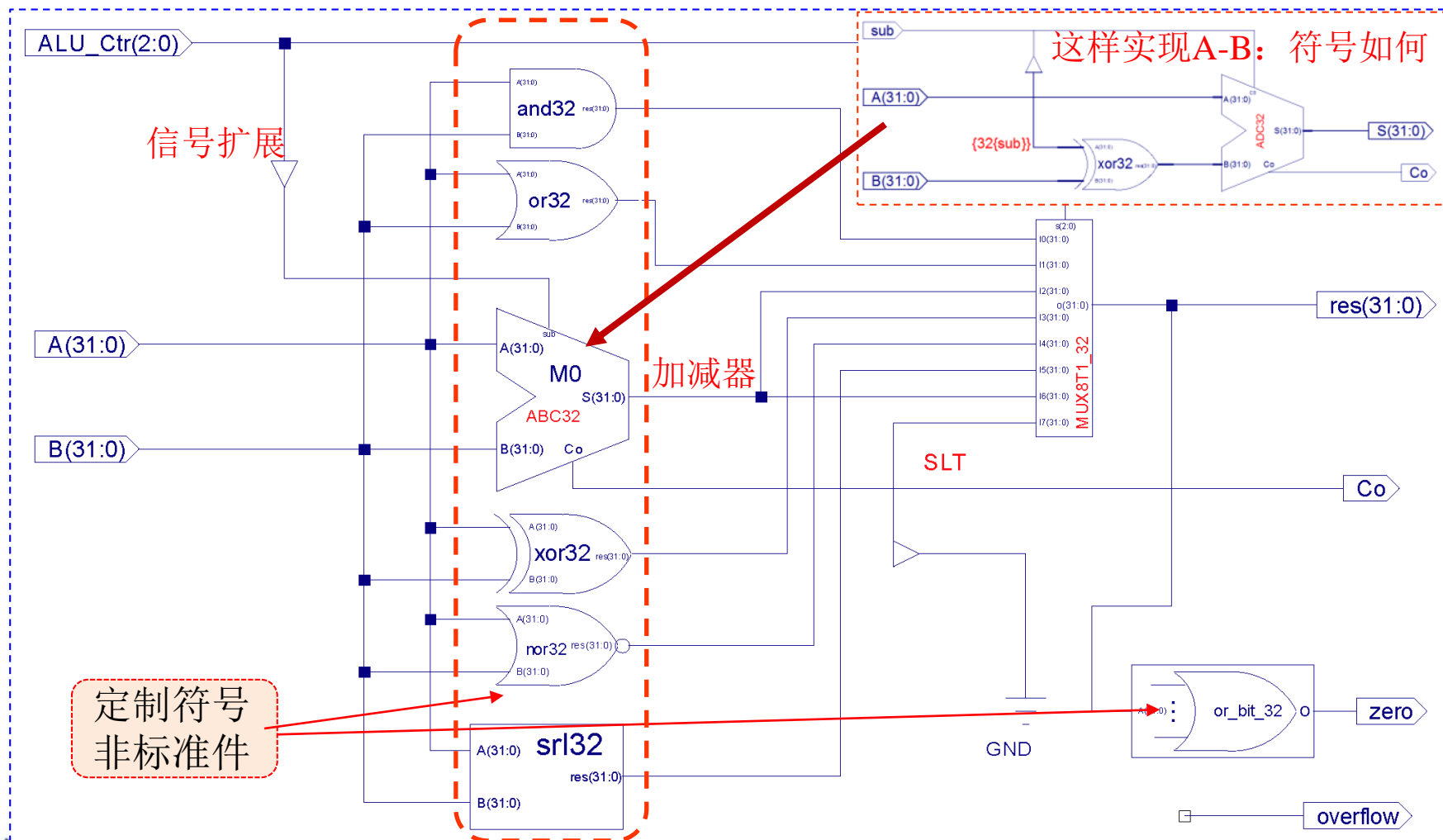
附录



ISE电路描述参考



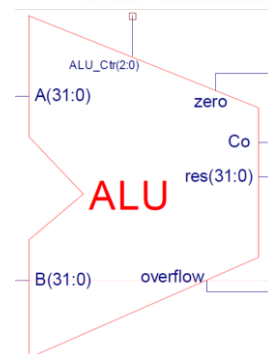
ALU逻辑原理图输入



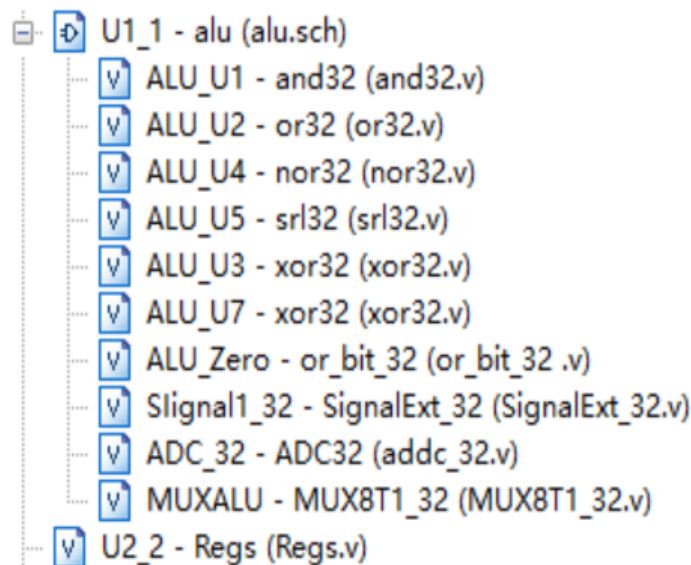
ALU测试激励参考代码

```

A=32'hA5A5A5A5;
B=32'h5A5A5A5A;
ALU_operation =3'b111;
#100;
ALU_operation =3'b110;
#100;
ALU_operation =3'b101;
#100;
ALU_operation =3'b100;
#100;
ALU_operation =3'b011;
#100;
ALU_operation =3'b010;
#100;
ALU_operation =3'b001;
#100;
ALU_operation =3'b000;
#100;
A=32'h01234567;
B=32'h76543210;
ALU_operation =3'b111;
    
```



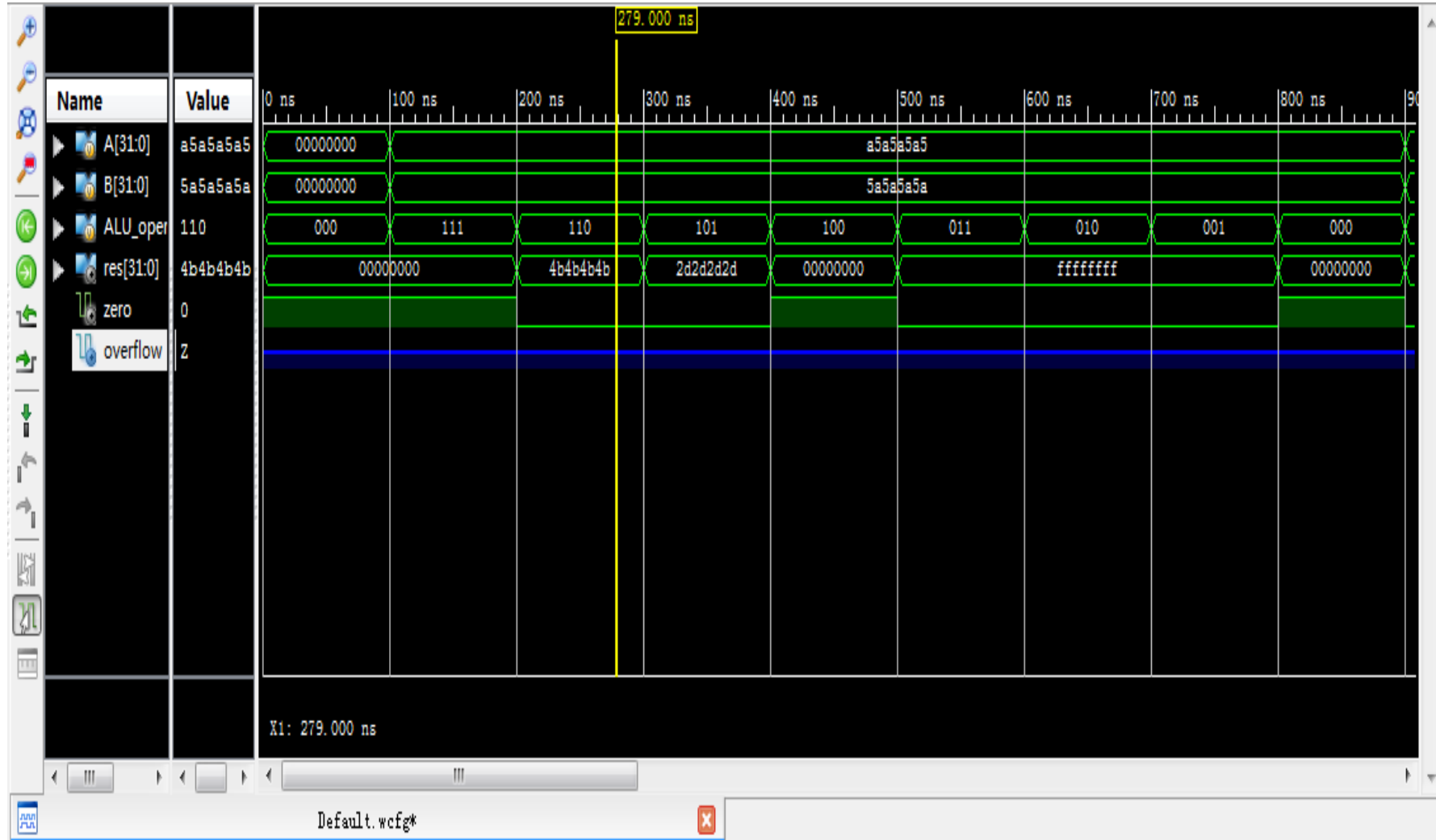
仿真通过后封装逻辑符号



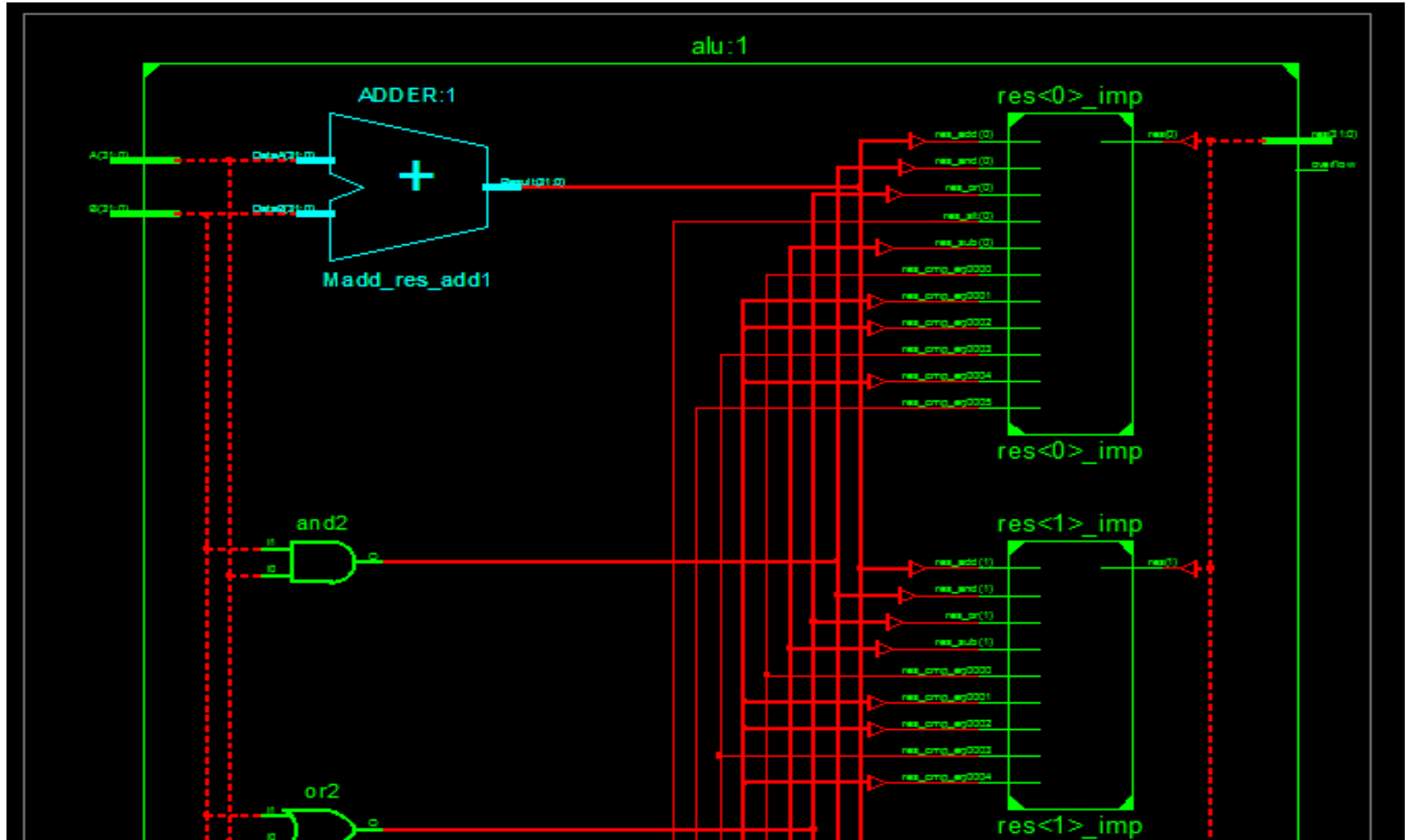
ALU模块调用结构



ALU_Simulation结果参考



RTL-Schematic





行为描述设计 Register files

此部件是逻辑实验Exp10的Regs的优化供OExp05使用
可与ALU共享工程

逻辑Exp10的Regs特点:

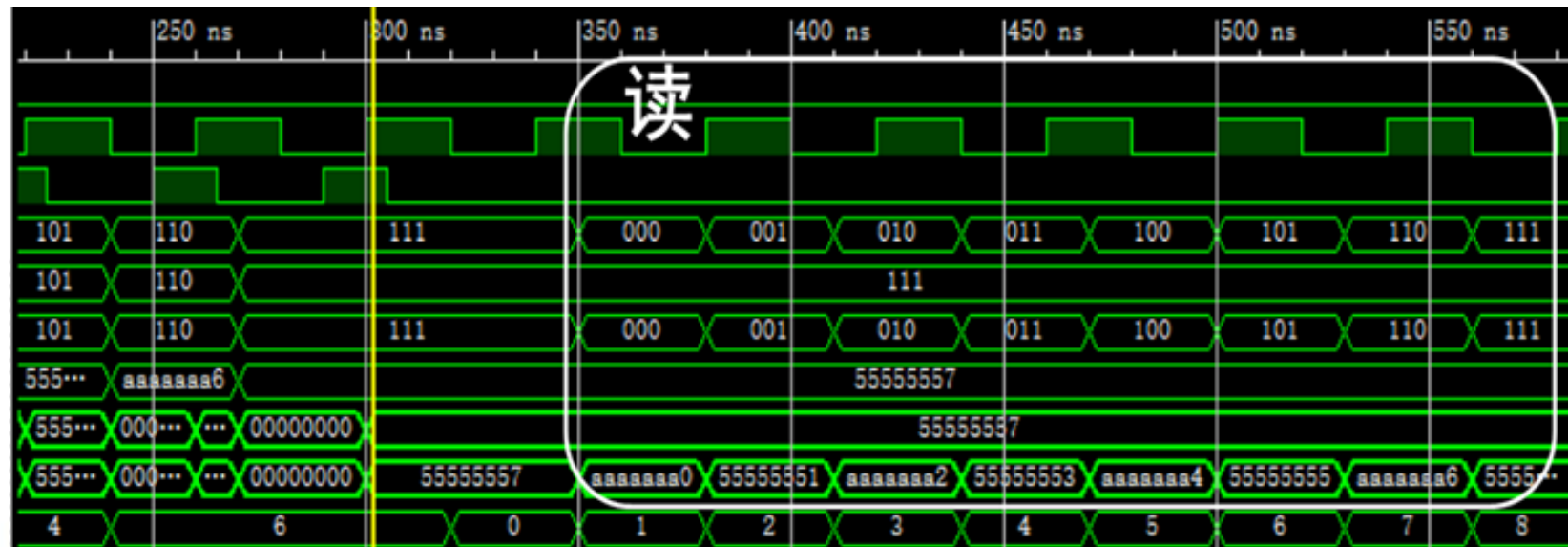
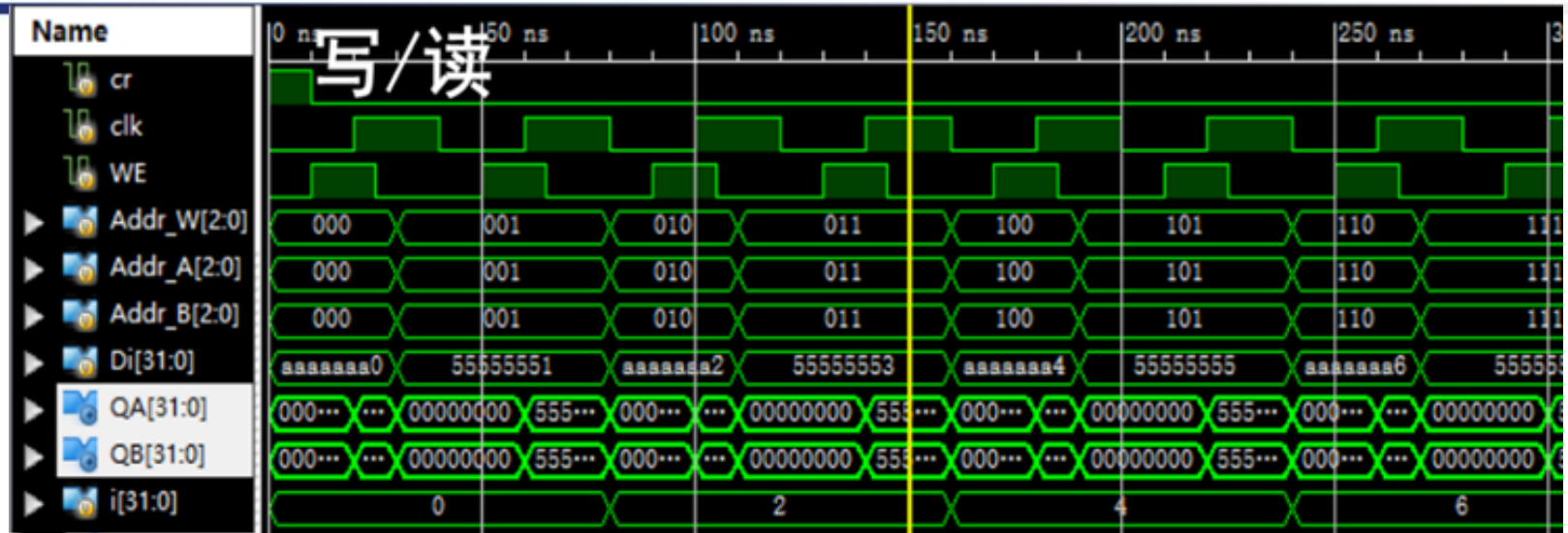
- 采用逻辑门实例描述实现D触发器
- 采用多层调用MB_DFF触发器模块实现寄存器
- 采用结构描述实现Register Files



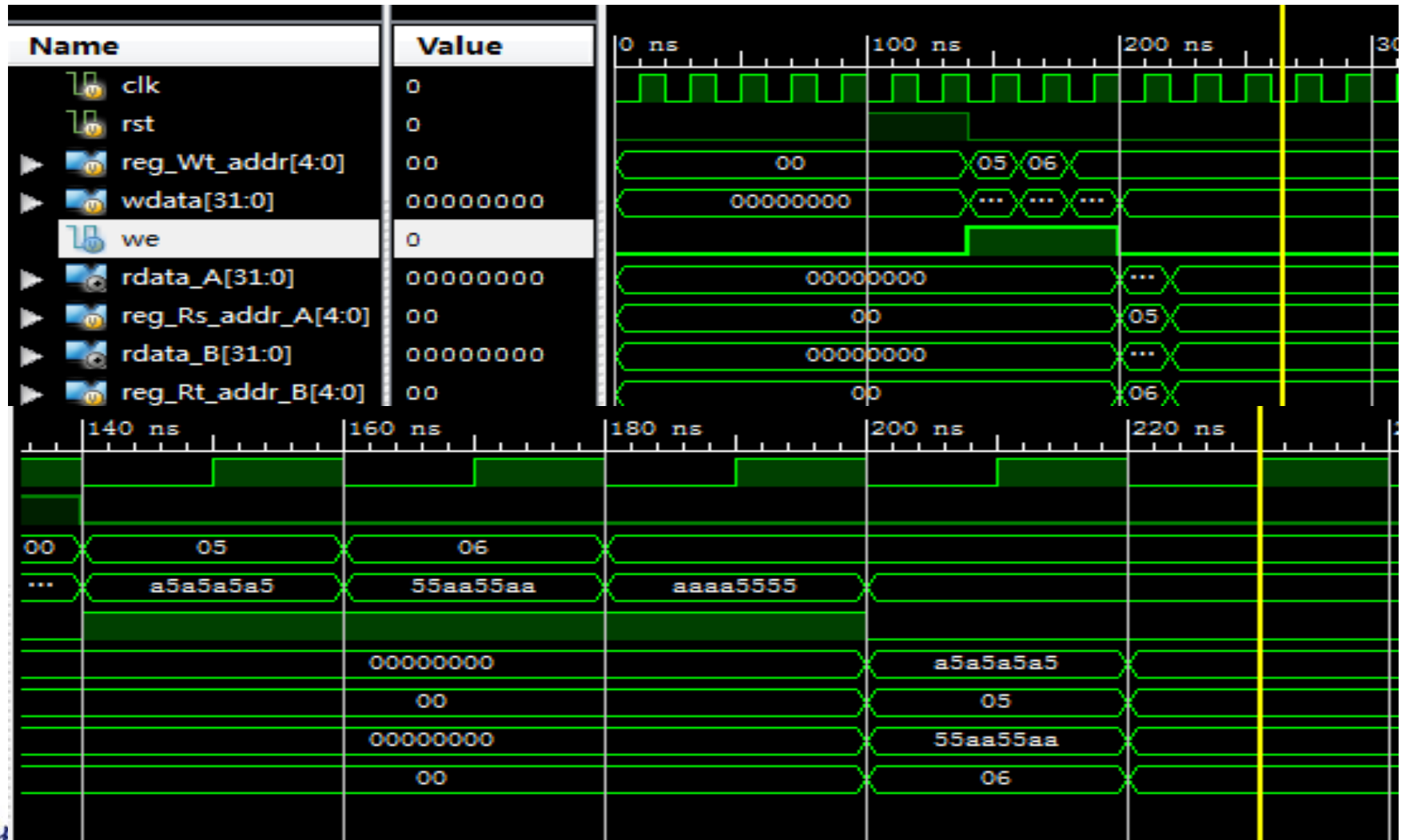
Regfile测试激励参考代码(不含采样)

```
1 integer i=0;
2 initial begin
3     // Initialize Inputs
4     clk = 0;
5     cr = 1;
6     WE = 0;
7     Addr_W = 0;
8     Addr_A = 0;
9     Addr_B = 0;
10    Di = 0;
11    fork
12        forever #20 clk <= ~clk; //寄存器触发时钟
13        #10 cr = 0;
14        begin
15            for (i=0; i<8; i=i+2)begin //地址奇偶交叉遍历：写、读
16                Addr_W <= i; //写地址：偶数地址
17                Addr_A <= i; //A口读地址，注意读出线变化时间
18                Addr_B <= i; //B口读地址，注意读出线变化时间
19                Di <= 32'hAAAAAA0+i; //写入数据：特征数据+标志数据i
20                #10; WE <=1; //写脉冲=1：注意与时钟边沿时序
21                #15; WE <=0; //写脉冲=0
22                #5;
23                Addr_W <= i+1; //写地址：奇数地址
24                Addr_A <= i+1; //A口读地址，注意读出线变化时间
25                Addr_B <= i+1; //B口读地址，注意读出线变化时间
26                Di <= 32'h55555551+i; //写入数据：特征数据+标志数据i
27                #20; WE <=1; //写脉冲=1：注意与时钟边沿时序
28                #15; WE <=0; //写脉冲=0
29                #15;
30            end
31            WE = 0; //写信号恒等于“零”
32            for (i=0; i<8; i=i+1)begin //地址顺序遍历：读
33                #30 Addr_W <= i; //写地址
34                Addr_B <= i; // A口读地址：分析读出线及变化时间
35                Addr_B <= i; // B口读地址：分析读出线及变化时间
36            end
37        end
38    join
39    end
```

regfile仿真结果



修改32个后regfile仿真结果





思考题

- 如何给ALU增加溢出功能
 - 提示：分析运算结果的符号
- 分析逻辑Exp10的Register Files设计
 - 本实验你做了那些优化？
 - 逻辑Exp10的Register Files直接使用，你认为会存在那些问题？