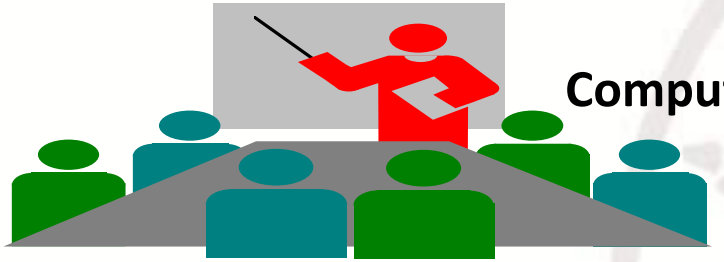


Computer Organization & Design
实验与课程设计

实验四

集成替换CPU核

-IP核设计CPU/IP2CPU

-逻辑实验模块优化三

施青松
Asso. Prof. Shi Qingsong
College of Computer Science and Technology, Zhejiang University
zjsqs@zju.edu.cn

Course Outline

- 实验目的与实验环境
- 实验任务
- 实验原理
- 实验操作与实现

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：数据通路与控制器
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

实验环境

实验设备

1. 计算机（Intel Core i5以上，4GB内存以上）系统

2. Spartan-3 Starter Kit Board/Sword开发板

3. Xilinx ISE14.4及以上开发工具

材料

无

Course Outline

- 实验目的与实验环境
- 实验任务
- 实验原理
- 实验操作与实现

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

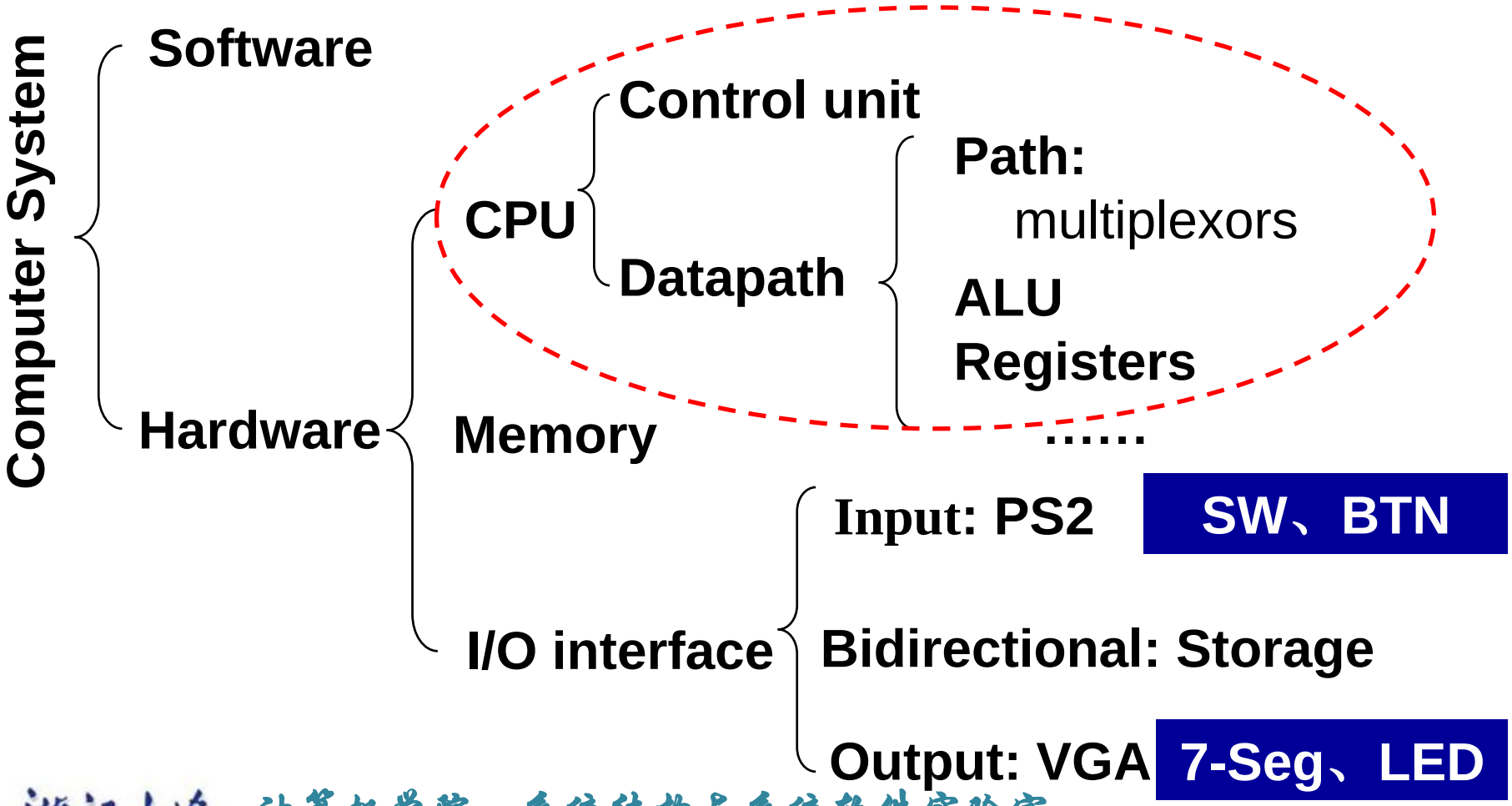
4. 熟练掌握IP核的使用方法

Course Outline

- 实验目的与实验环境
- 实验任务
- 实验原理
- 实验操作与实现

Computer Organization

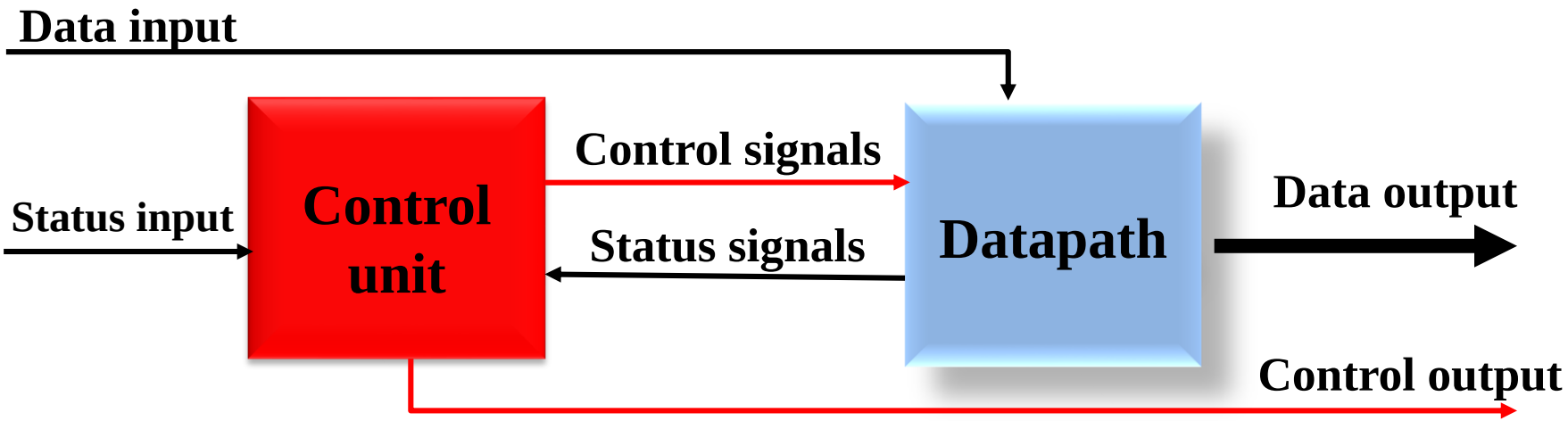
Decomposability of computer systems



Digital circuits vs CPU organization

Digital circuit

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

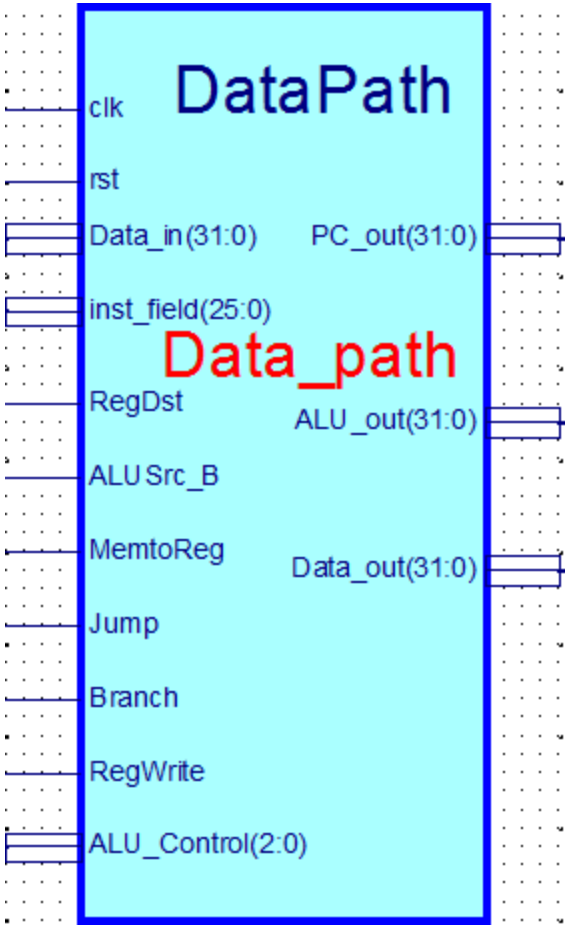


Computer organization

- Special circuits that processes logical action with instructions -Software

CPU部件之1-数据通路：Data_path

- **Data_path**
  - 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- Data_path**
  - 核调用模块Data_path.ngc
  - 核接口信号模块(空文档)：Data_path.v
  - 核模块符号文档：Data_path.sym

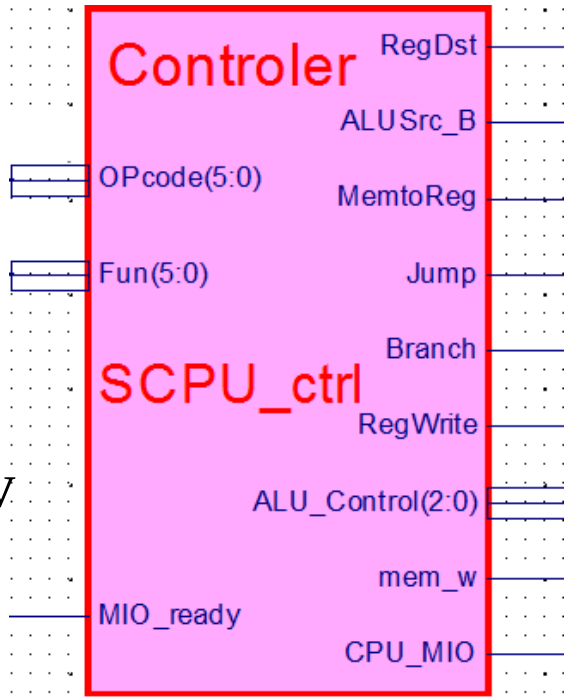


数据通路空模块- Data_path.v

```
module      Data_path( input clk,                    //寄存器时钟
                       input rst,                    //寄存器复位
                       input[25:0]inst_field,        //指令数据域
                       input RegDst,
                       input ALUSrc_B,
                       input MemtoReg,
                       input Jump,
                       input Branch,
                       input RegWrite,
                       input[31:0]Data_in,
                       input[2:0]ALU_Control,

                       output[31:0]ALU_out,
                       output[31:0]Data_out,
                       output[31:0]PC_out
                       );
endmodule
```

CPU部件之2-控制器：SCPU_ctrl

- **SCPU_ctrl**
  - CPU主要部件之一
  - 寄存器传输控制技术中的运算和通路控制器：
- **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- SCPU_ctrl**
  - 核调用模块SCPU_ctrl.ngc
  - 核接口信号模块(空文档)：SCPU_ctrl.v
  - 核模块符号文档：SCPU_ctrl.sym



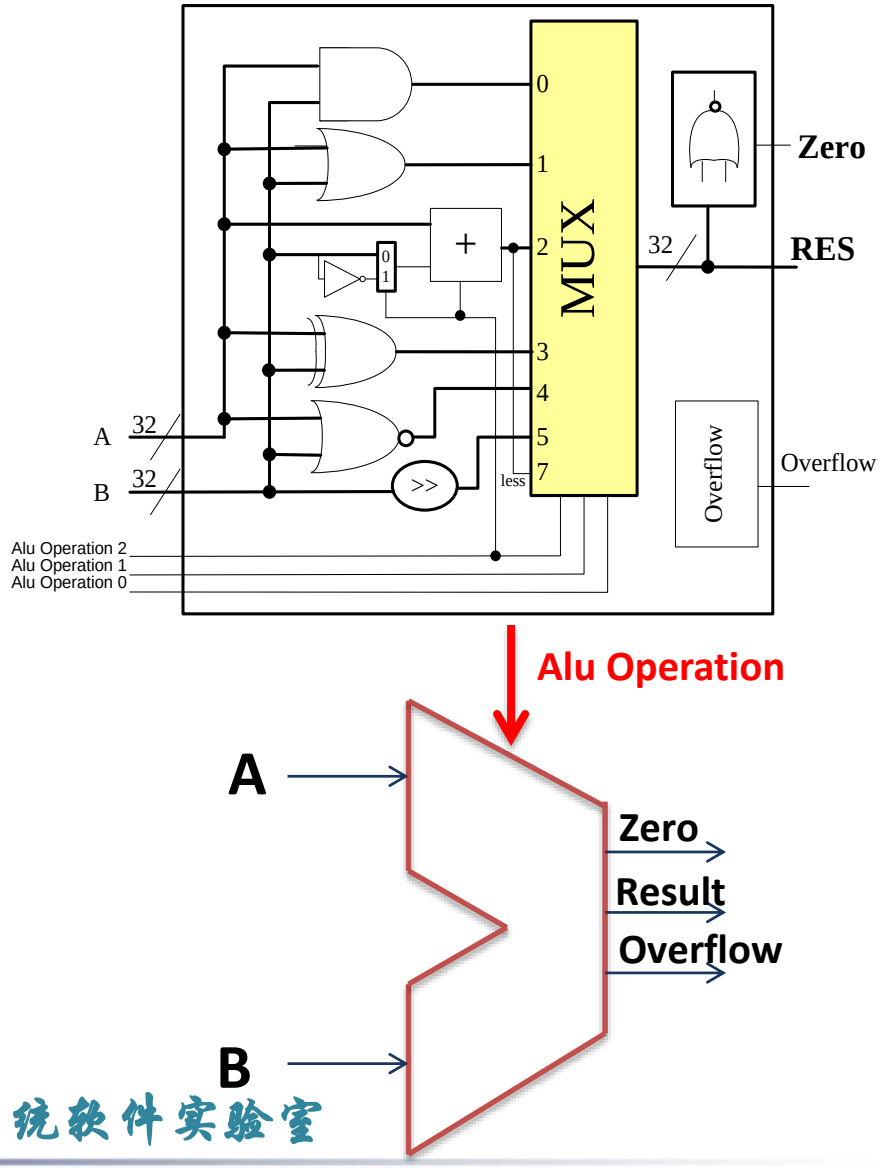
控制器接口文档- SCPU_ctrl.v

```
module      SCPU_ctrl( input[5:0]OPcode,     //OPcode
                       input[5:0]Fun,        //Function
                       input MIO_ready,      //CPU Wait

                       output reg RegDst,
                       output reg ALUSrc_B,
                       output reg MemtoReg,
                       output reg Jump,
                       output reg Branch,
                       output reg RegWrite,
                       output reg mem_w,
                       output reg [2:0]ALU_Control,
                       output reg CPU_MIO
                       );

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 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



硬件描述参考代码

```
module alu(input A, B,
           input[2:0] ALU_operation,
           output[31:0] res,
           input zero, overflow);
 wire [31:0] res_and,res_or,res_add,res_sub,res_nor,res_slt;
 reg [31:0] res;
   parameter one = 32'h00000001, zero_0 = 32'h00000000;
      assign res_and = A&B;
      assign res_or = A|B;
      assign res_add = A+B;
      assign res_sub = A-B;
      assign res_slt =(A < B) ? one : zero_0;
      always @ (A or B or ALU_operation)
            case (ALU_operation)
            3'b000: res=res_and;
            3'b001: res=res_or;
            3'b010: res=res_add;
            3'b110: res=res_sub;
            3'b100: res=~(A | B);
            3'b111: res=res_slt;
            default: res=32'hx;
            endcase
     assign zero = (res==0)? 1: 0;
endmodule
```

How do you write with overflow code ?

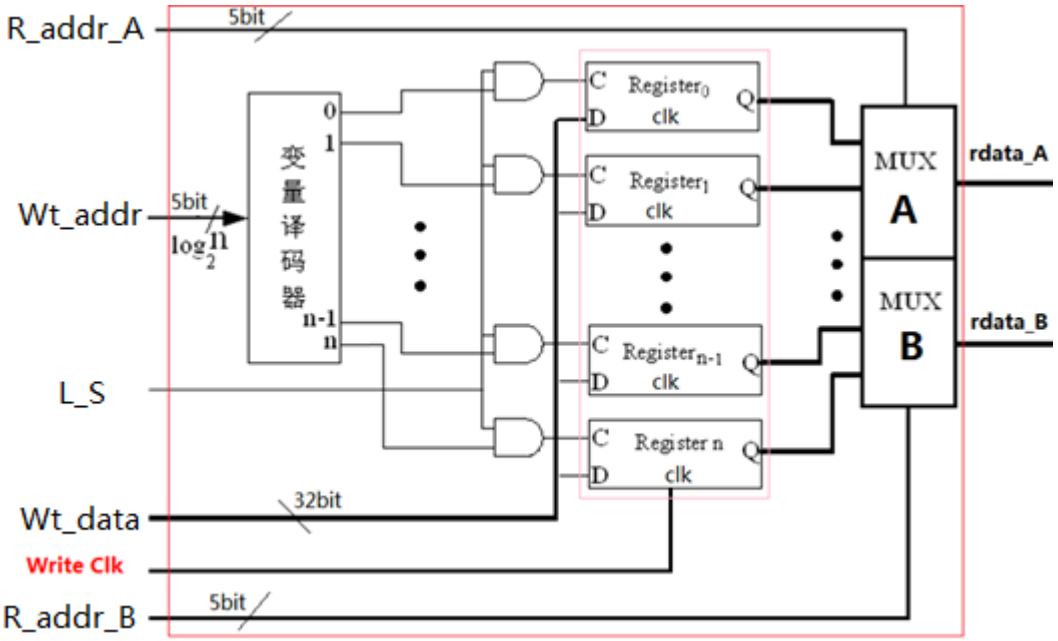
What is the difference The codes in the Synthesize?

```
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32×32bit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



Course Outline

- 实验目的与实验环境
- 实验任务
- 实验原理
- 实验操作与实现

设计工程：OExp04-IP2CPU

- **分解CPU为二个IP核**
  - 在Exp03工基础上用二个IP核构建CPU
  - 顶层模块延用Exp03
    - 模块名：Top_OExp04_IP2CPU.sch
- **逻辑实验输出模块优化**
  - ALU模块优化
  - Register Files模块优化
  - 优化目标：满足MIPS处理器的要求

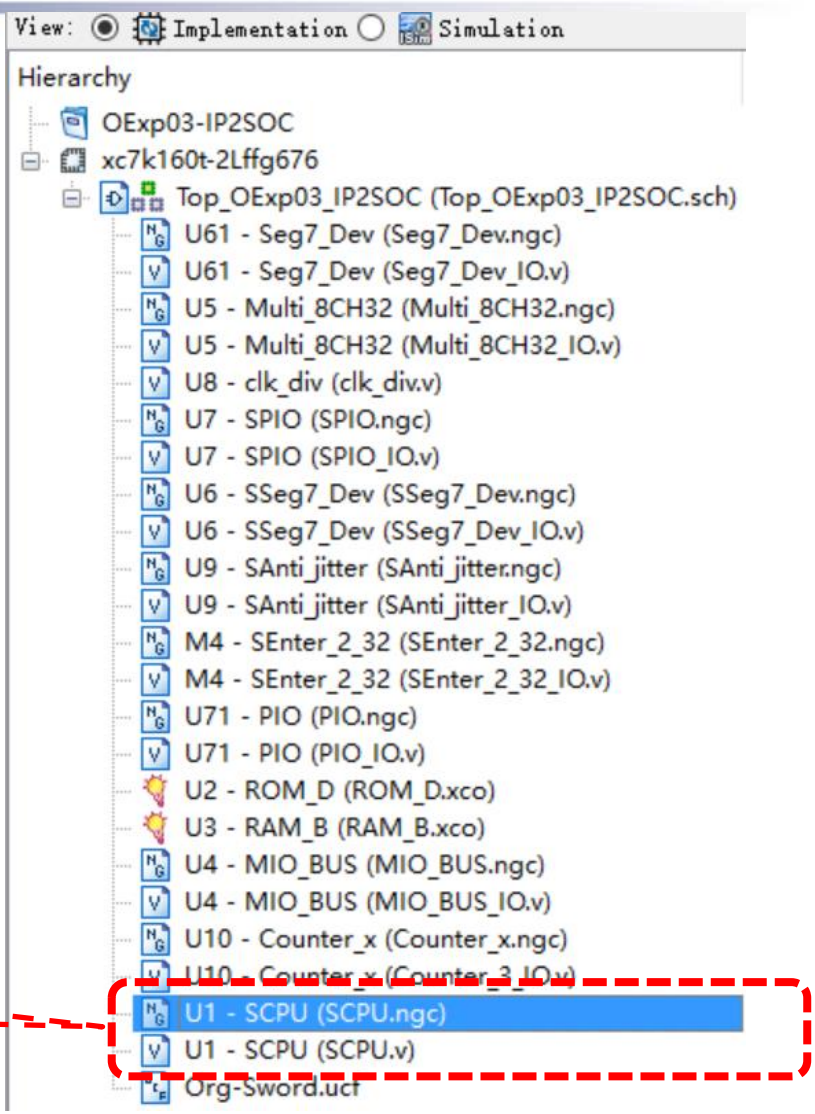
设计要点

IP核设计CPU

--用二个第三方IP核集成CPU

清理Exp03工程

- **移除工程中的CPU核**
  - Exp03工程中移除CPU核关联
- **删除工程中CPU核文件**
  - SCPU.ngc 和 SCPU.v 文件
  - 在Project菜单中运行：
    Cleanup Project Files …
- **建议用Exp03资源重建工程**
  - 除CPU核
  - 命名：OExp04-IP2CPU



Exp03需要清理的核

拷贝二个IP核的Symbol文件到当前工程目录：

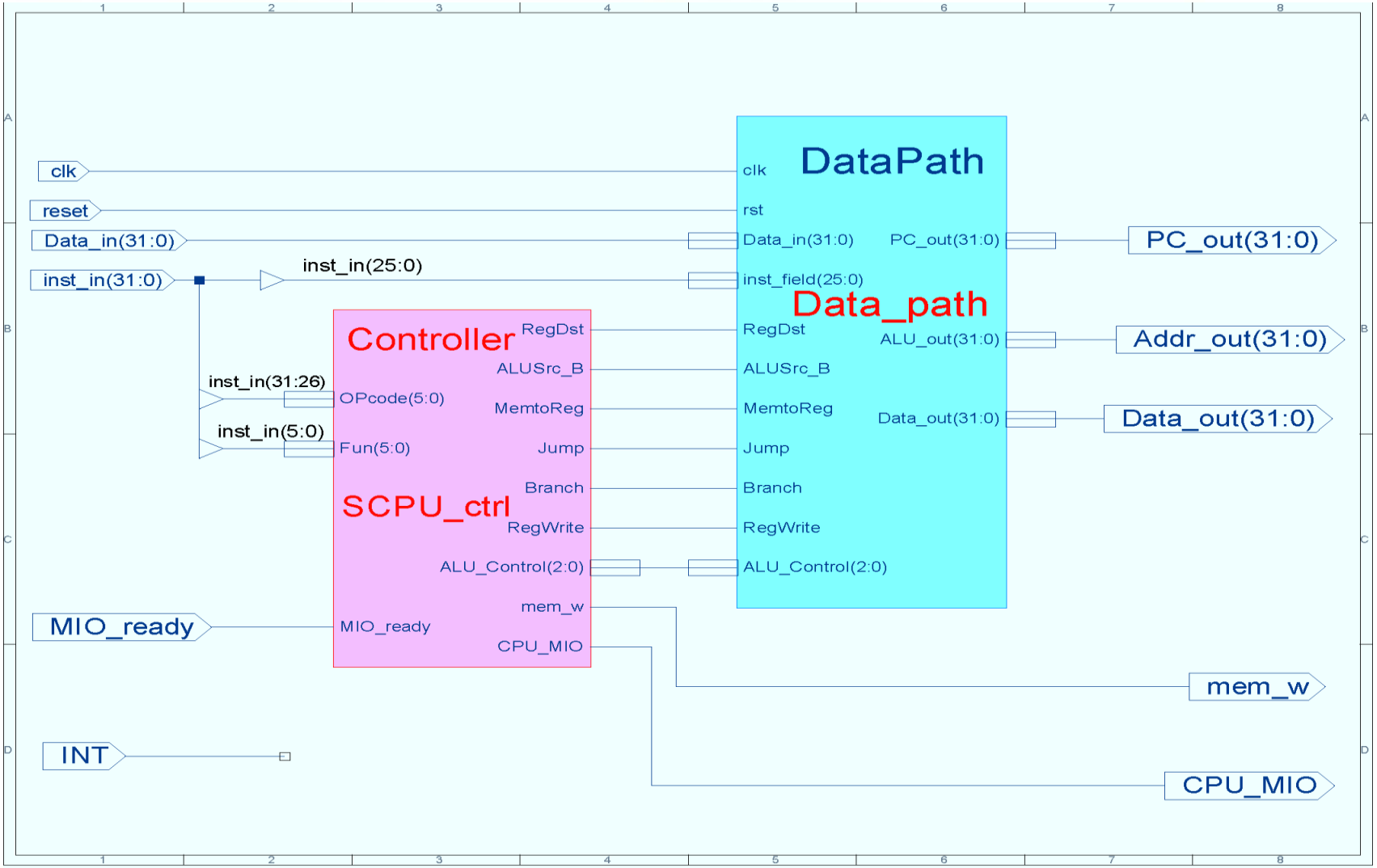
增加SCPU_ctrl.sym、Data_path.sym

拷贝二个IP软核.ngc文档到当前工程目录：

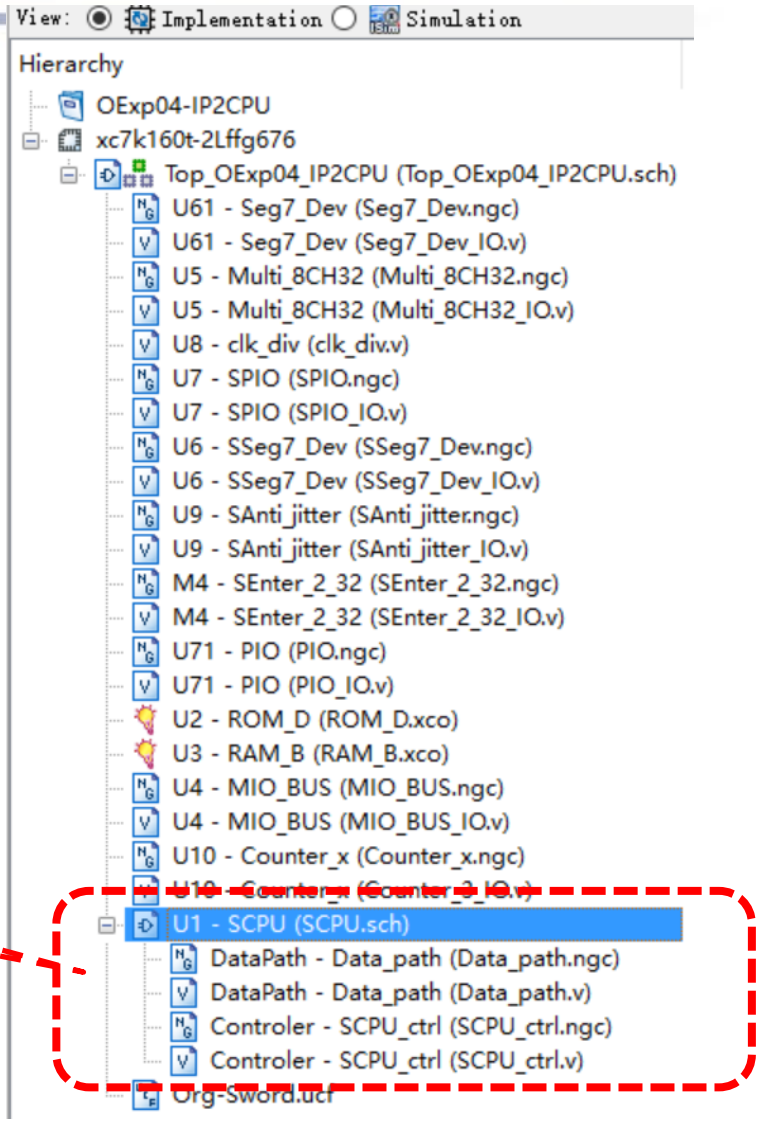
SCPU_ctrl.ngc、Data_path.ngc

接口文件SCPU_ctrl.v、Data_path.v

用逻辑原理图输入CPU设计



Exp04完成CPU设计后
的模块调用关系

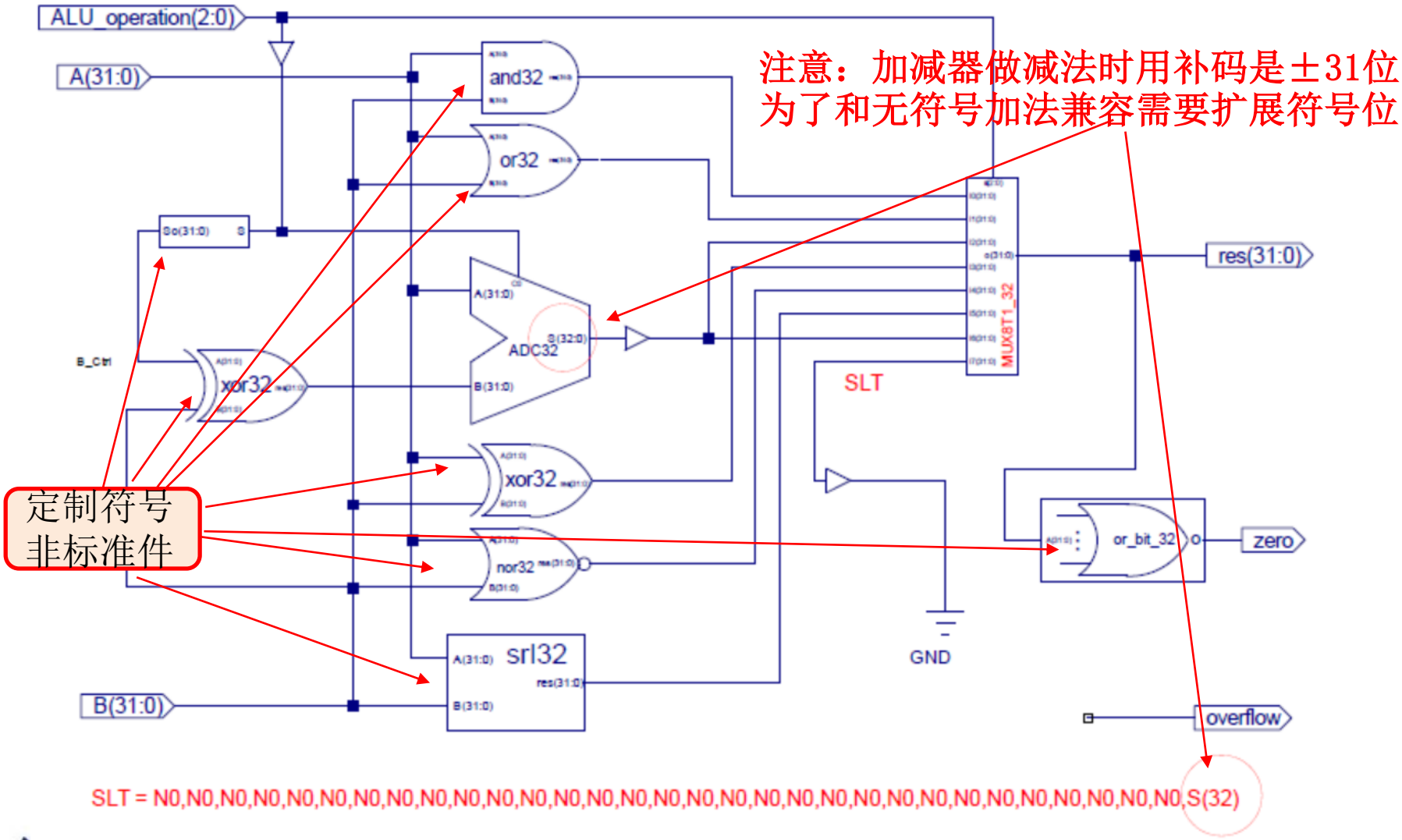


逻辑原理图输入设计ALU

此部件是逻辑实验的ALU进一步改造供Exp05使用

为了减少相互影响请单独建立工程

逻辑原理图输入


ALU_operation(2:0)
A(31:0)
B(31:0)
B_Ctrl
and32
or32
xor32
xor32
nor32
ADC32
A(31:0)
B(31:0)
S(32:0)
srl32
A(31:0)
B(31:0)
res(31:0)
MUX8T1_32
SLT
GND
or_bit_32
zero
overflow
res(31:0)
注意：加减器做减法时用补码是±31位
为了和无符号加法兼容需要扩展符号位
定制符号
非标准件
SLT = N0,N0,N0,N0,N0,N0,N0,N0,N0,N0,N0,N0,N0,N0,N0,N0,N0,N0,N0,N0,N0,N0,N0,N0,N0,N0,N0,N0,N0,N0,N0,S(32)

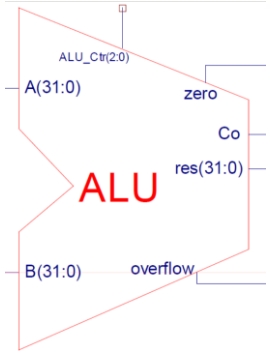
拷贝下列模块符号到ALU工程目录：

（Exp01提供）

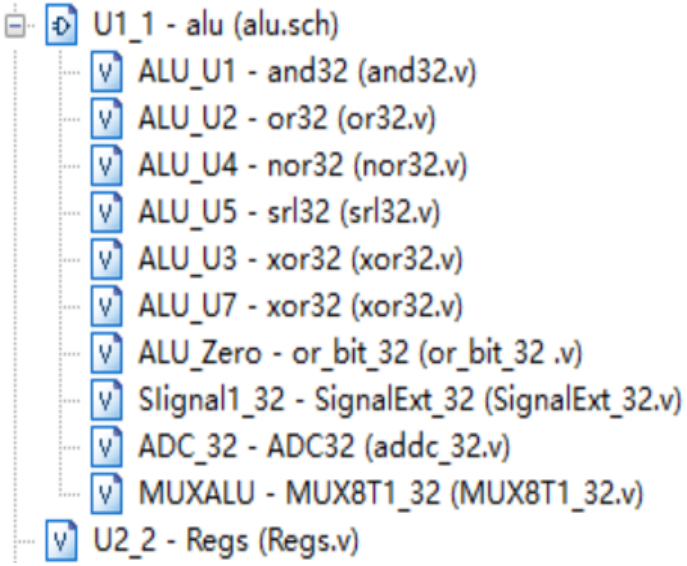
and32、or32、ADC32、xor32、nor32、srl32、SignalExt_32、mux8to1_32、or_bit_32

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;
```

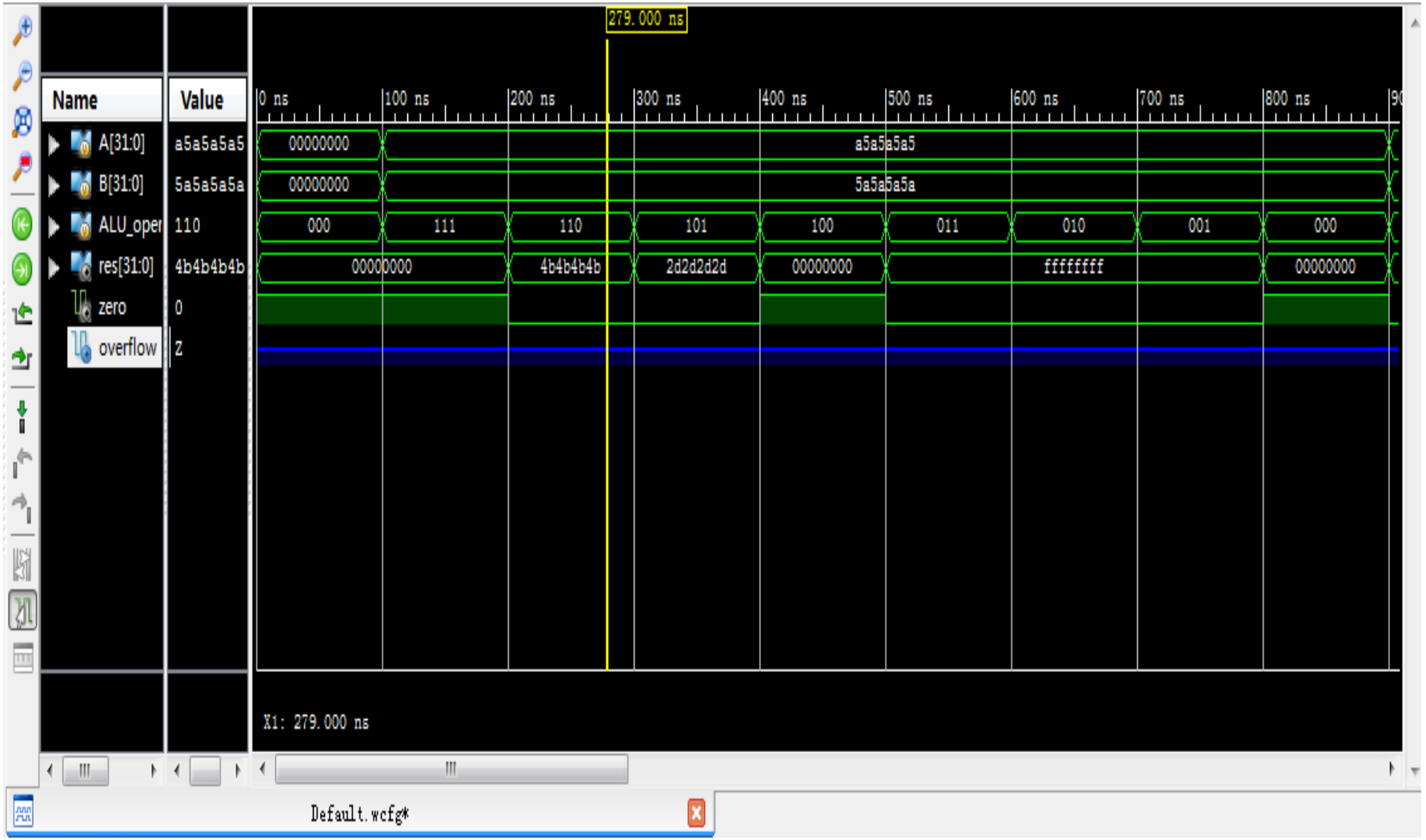


仿真通过后封装逻辑符号

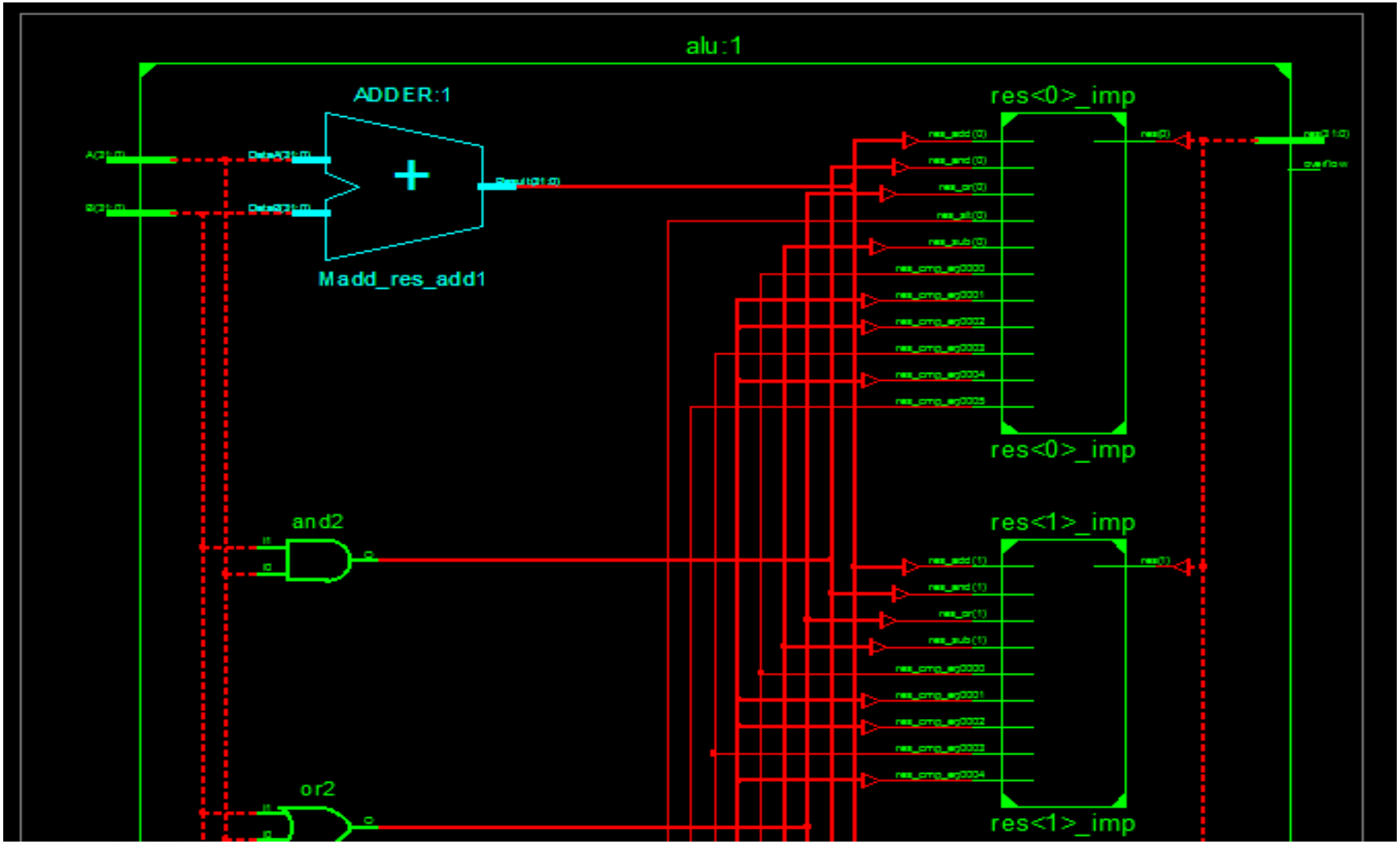


ALU模块调用结构

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Exp05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

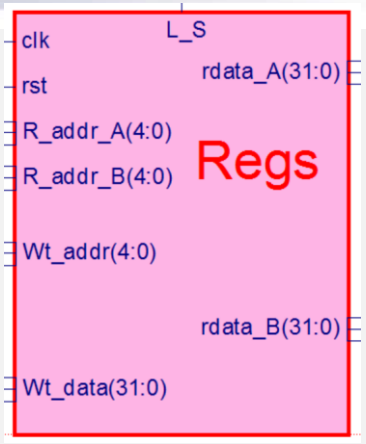
采用多层调用MB_DFF触发器模块实现寄存器

采用结构描述实现Register Files

非常精练的参考代码

此代码留有BUG，请同学自行编写

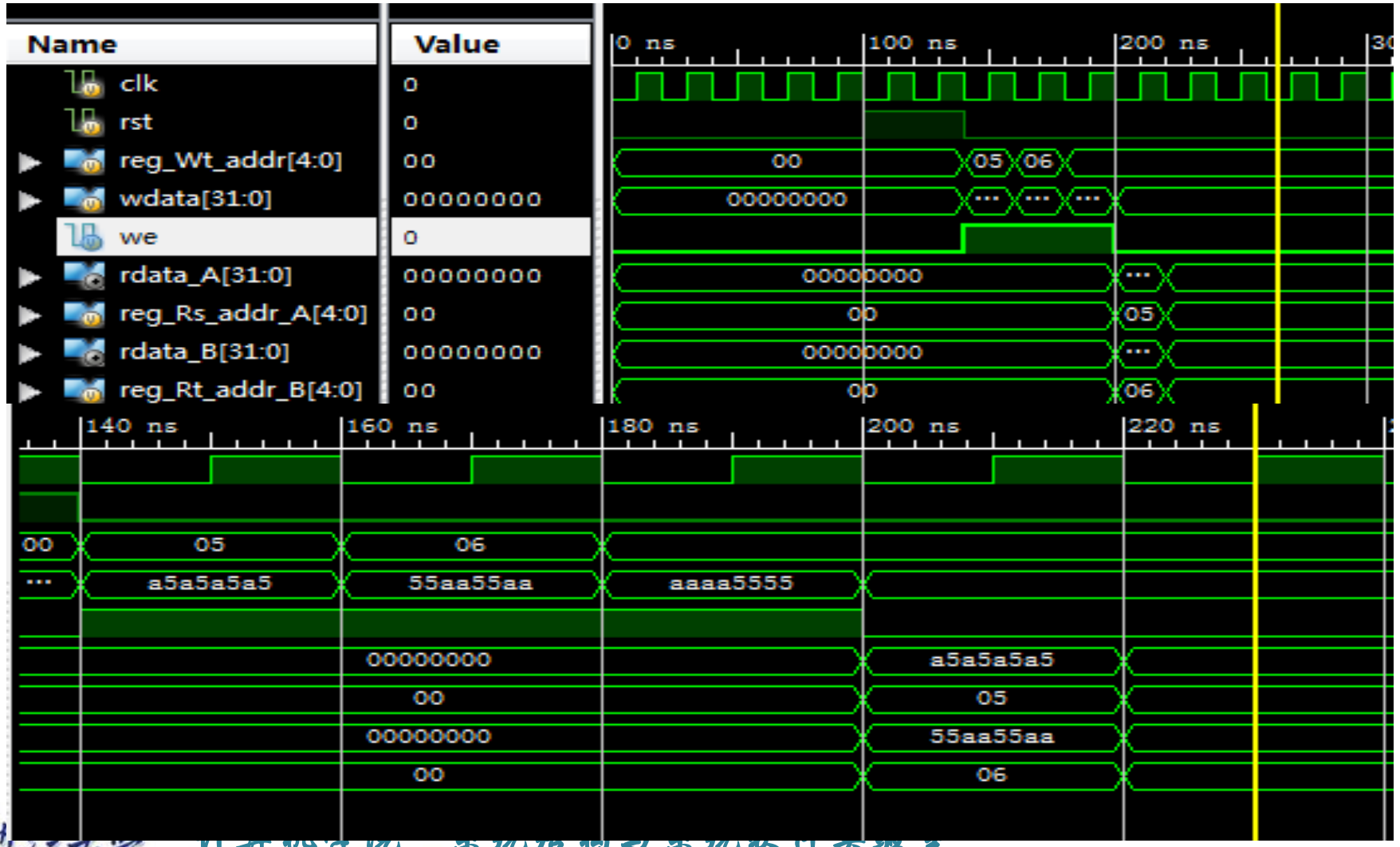
```
Module  regs(input clk,  rst,  L_S,
                  input [4:0]      R_addr_A, R_addr_B, Wt_addr,
                  input [31:0]     wt_data
                  output [31:0]  rdata_A,  rdata_B
                );
reg [31:0] register [1:31];                    // r1 - r31
    integer i;

    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

endmodule
```



仿真通过后封装逻辑符号

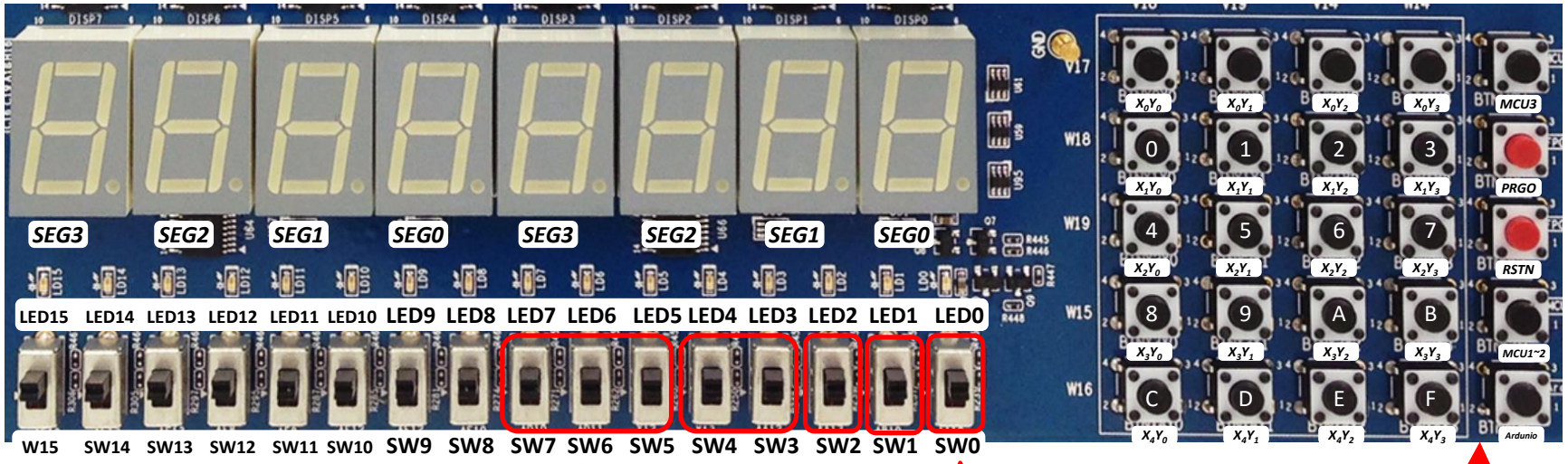
代码来自李亚民教授

regfile仿真结果



物理验证-DEMO接口功能

- 物理验证同实验三



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[0]=文本图形选择

SW[1]=高低16位选择

SW[2]=CPU单步时钟选择

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

没有使用

思考题

- **如何结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？


END