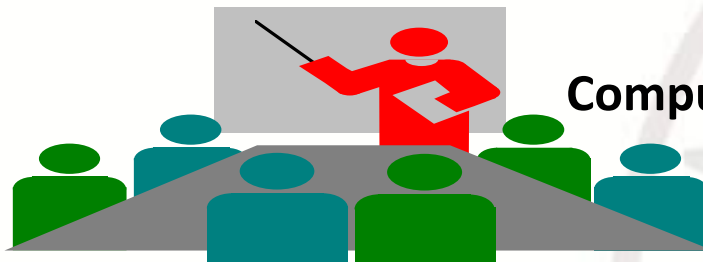




浙江大学
ZHEJIANG UNIVERSITY



Computer Organization & Design

Computer Organization & Design 实验与课程设计

实验六

CPU设计-控制器

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. 运用寄存器传输控制技术
2. 掌握CPU的核心：指令执行过程与控制流关系
3. 设计控制器
4. 学习测试方案的设计
5. 学习测试程序的设计



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

A vertical diagram showing four steps of a course outline. Each step is represented by a white circle on the left, connected by a vertical line, with a corresponding colored rectangular bar to its right. The bars are blue, yellow, blue, and blue from top to bottom. The text inside the bars is white.

实验目的与实验环境

实验任务

实验原理

实验操作与实现

1. 设计9+条指令的控制器

- 用硬件描述语言设计实现控制器
 - 根据Exp05数据通路及指令编码完成控制信号真值表
- 替换Exp05的控制器核
- 此实验在Exp05的基础上完成

2. 设计控制器测试方案：

- OP译码测试：R-格式、访存指令、分支指令，转移指令
- 运算控制测试：Function译码测试

3. 设计控制器测试程序

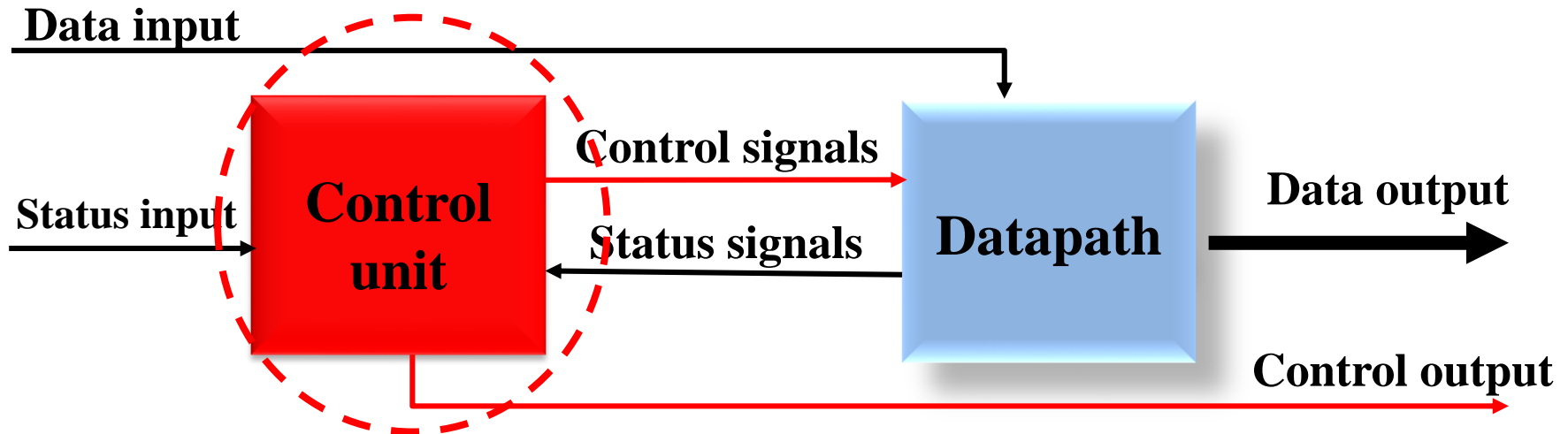
Course Outline



CPU organization

□ Digital circuit

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

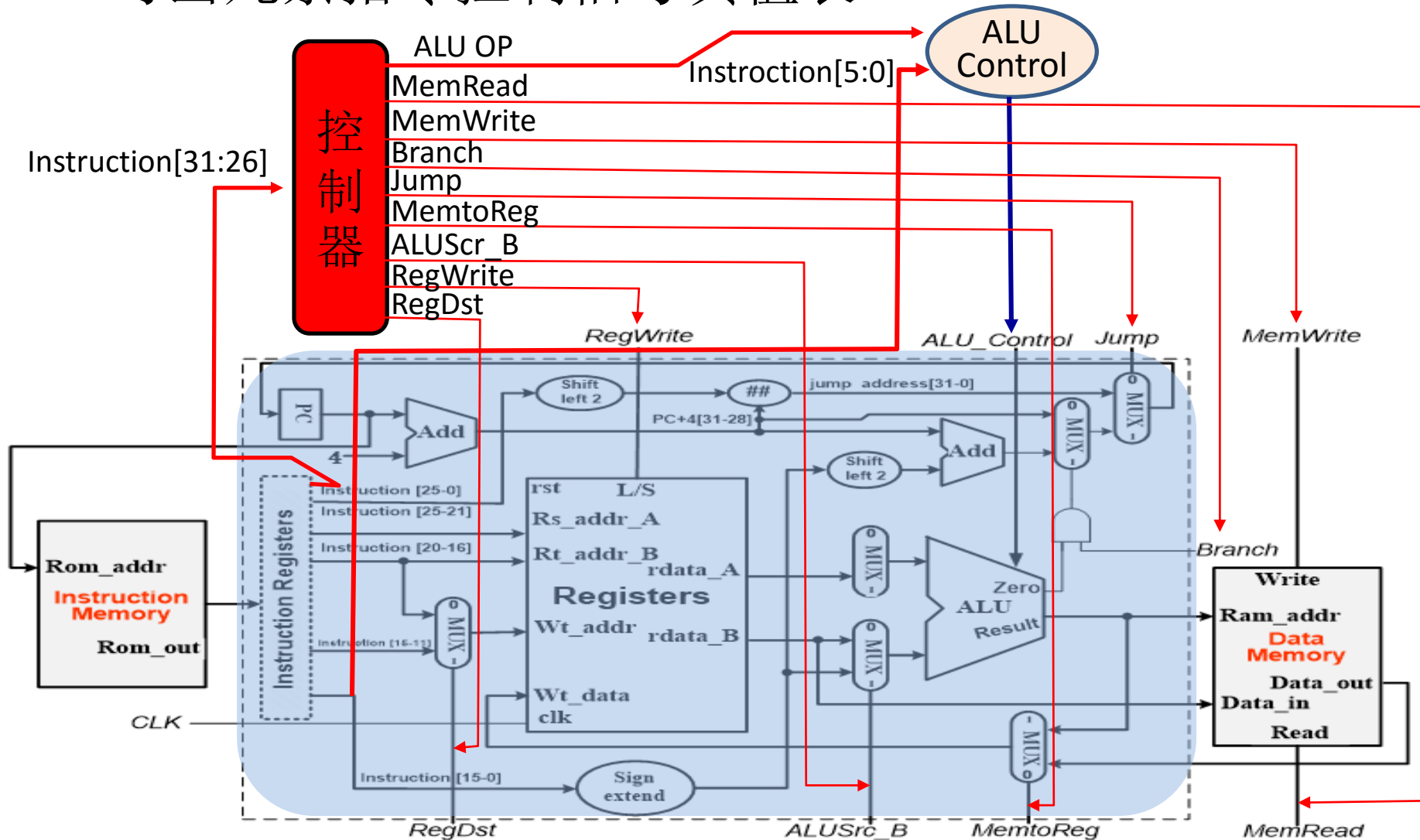


□ Computer organization

- Special circuits that processes logical action with instructions
-Software

控制对象：数据通路结构

□ 写出九条指令控制信号真值表





控制信号定义

□ 通路与控制

| 信号 | 源数目 | 功能定义 | 赋值0时动作 | 赋值1时动作 |
|-------------|---------|-------------|----------|---------------------|
| ALUSrc_B | 2 | ALU端口B输入选择 | 选择寄存器B数据 | 选择32位立即数 (符号扩展后) |
| RegDst | 2 | 寄存器写地址选择 | 选择指令rt域 | 选择指令rs域 |
| MemtoReg | 2 | 寄存器写入数据选择 | 选择存储器数据 | 选择ALU输出 |
| Branch | 2 | Beq指令目标地址选择 | 选择PC+4地址 | 选择转移地址 (Zero=1) |
| Jump | 2 | J指令目标地址选择 | 选择J目标地址 | 由Branch决定输出 |
| RegWrite | - | 寄存器写控制 | 禁止寄存器写 | 使能寄存器写 |
| MemWrite | - | 存储器写控制 | 禁止存储器写 | 使能存储器写 |
| MemRead | - | 存储器读控制 | 禁止存储器读 | 使能存储器读 |
| ALU_Control | 000-111 | 3位ALU操作控制 | 参考表Exp04 | Exp04 |



主控制器信号真值表

□ 分析填写控制器输出信号真值表

| OP | Reg Dst | ALU Src | Mem toReg | Reg Write | Mem Read | Mem Write | Branch | Jump | ALU op1 | ALU op0 |
|----------------|---------|---------|-----------|-----------|----------|-----------|--------|------|---------|---------|
| R-格式 000000 | | | | | | | | | | |
| I-格式LW ??? | | | | | | | | | | |
| I-格式SW ??? | | | | | | | | | | |
| I-格式beq ??? | | | | | | | | | | |
| J-格式 ??? | | | | | | | | | | |
| | | | | | | | | | | |
| | | | | | | | | | | |

□ 参考实验四

| Instruction opcode | ALUop | Instruction operation | Funct field | Desired ALU action | ALU_Control |
|--------------------|-------|-----------------------|-------------|--------------------|-------------|
| LW | 00 | Load word | xxxxxx | Load word | 010 |
| SW | 00 | Store word | xxxxxx | Store word | 010 |
| Beq | 01 | branch equal | xxxxxx | branch equal | 110 |
| R-type | 10 | add | 10 0000 | add | 010 |
| R-type | 10 | subtract | 10 0010 | subtract | 110 |
| R-type | 10 | AND | 10 0100 | AND | 000 |
| R-type | 10 | OR | 10 0101 | OR | 001 |
| R-type | 10 | Set on less than | 10 1010 | Set on less than | 111 |
| R-type | 10 | NOR | 10 0111 | NOR | 100 |

ALU操作译码化简

对Funct变量化简

- ALUop 单独考虑
 - 化简后合并

$$\begin{aligned}
 ALU_Control_2 &= ALU_{op0} + ALU_{op1} F_1 \\
 ALU_Control_1 &= \overline{ALU_{op1}} + \bar{F}_2 \quad ??? \\
 ALU_Control_0 &= ALU_{op1} (\bar{F}_1 F_0 + F_3)
 \end{aligned}$$

| $F_3 F_2$ \ $F_1 F_0$ | 00 | 01 | 10 | 11 |
|-----------------------|----|----|----|----|
| 00 | 0 | x | x | 1 |
| 01 | 0 | 0 | 1 | x |
| 10 | x | x | x | x |
| 11 | x | x | x | 1 |

Red oval highlights the column where $F_1 F_0 = 10$. An arrow points to a question mark.

| $F_3 F_2$ \ $F_1 F_0$ | 00 | 01 | 10 | 11 |
|-----------------------|----|----|----|----|
| 00 | 1 | x | x | 1 |
| 01 | 0 | 0 | 0 | x |
| 10 | x | x | x | x |
| 11 | 1 | x | x | 1 |

Red oval highlights the row where $F_3 F_2 = 01$. An arrow points to a question mark.

反函数简单

| $F_3 F_2$ \ $F_1 F_0$ | 00 | 01 | 10 | 11 |
|-----------------------|----|----|----|----|
| 00 | 0 | x | x | 0 |
| 01 | 0 | 1 | 0 | x |
| 10 | x | x | x | x |
| 11 | x | x | x | 1 |

Red oval highlights the column where $F_1 F_0 = 10$. Blue oval highlights the row where $F_3 F_2 = 10$. Arrows point to question marks.

CPU部件之控制器接口： **SCPU_ctrl**

□ **SCPU_ctrl**

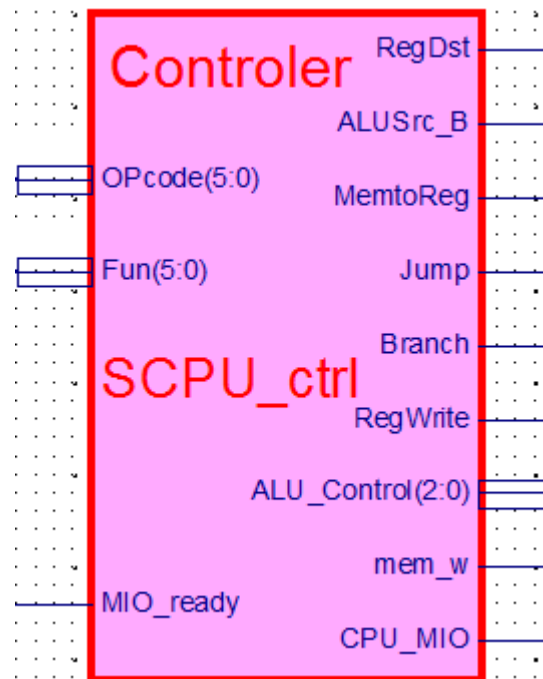
- CPU主要部件之一
- 寄存器传输控制者：编码转换成命令

□ 基本功能

- 微操作控制
- 数据传输通道控制
- 时序控制：单周期时序在那里？

□ 接口要求- **SCPU_ctrl**

- 控制器接口信号如右图
- 模块符号文档：SCPU_ctrl.sym



数据通路接口信号标准- **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
```

Course Outline





CPU之控制器设计

-控制实验五设计的数据通路



设计工程：OExp06-OwnSCPU

◎ 设计CPU之控制器

- ☞ 根据理论课分析讨论设计实验五数据通路的控制器
- ☞ 原理图或HDL描述均可
 - ◎ 但必须用函数表达式结构描述
- ☞ 仿真测试控制器模块

◎ 集成替换验证通过的数据通路模块

- ☞ 替换实验五(Exp05)中的SCPU_ctrl.ngc核
- ☞ 顶层模块沿用Exp05
 - ◎ 模块名：Top_OExp06_OwnSCPU.sch

◎ 测试控制器模块

- ☞ 设计测试程序(MIPS汇编)测试：
- ☞ OP译码测试：
 - ◎ R-格式、访存指令、分支指令，转移指令
- ☞ 运算控制测试：Function译码测试



设计要点

□ 设计主控制器模块

- 写出控制器的函数表达式
- 结构描述实现电路

□ 设计ALU操作译码

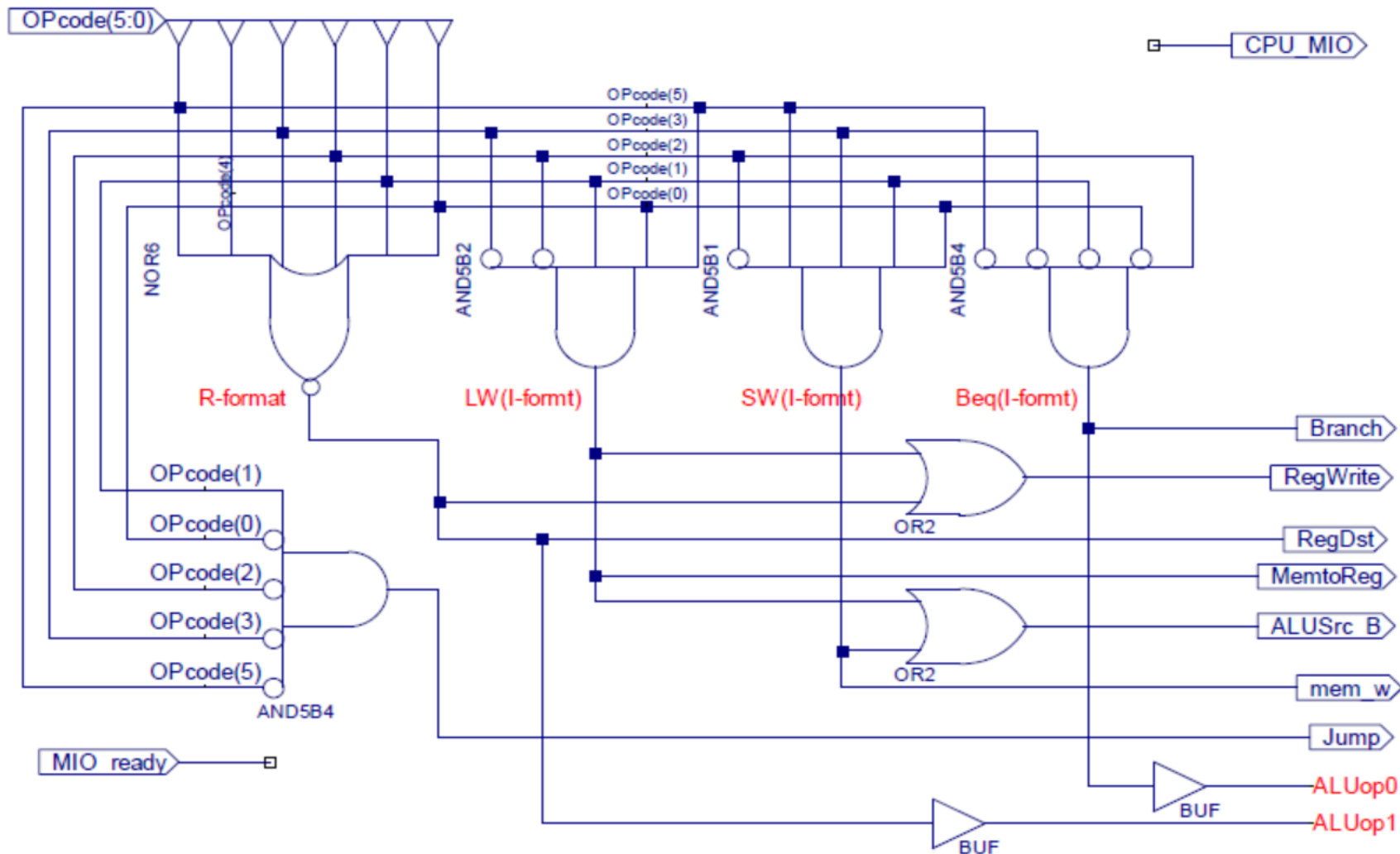
- 写出ALU操作译码函数表达式
- 结构描述实现电路
- 使用DEMO作功能初步调试
 - ALU必须运算包含“nor”操作
 - 否则需要修改或重新设计调试程序

□ 仿真二个控制器电路模块

- 可以单独或合并仿真，但最后要合并为一个控制模块



指令译码-主控制器逻辑电路





主控制器HDL描述结构

□ 指令译码器参考描述

```
`define CPU_ctrl_signals
{RegDst,ALUSrc_B,MemtoReg,RegWrite,MemRead,MemWrite,Branch,Jump,ALUop}
    assign mem_w = MemWrite&&(~MemRead);
    always @* begin
        case(OPcode)
            6'b000000: begin CPU_ctrl_signals = ?; end    //ALU
            6'b100011: begin CPU_ctrl_signals = ?; end    //load
            6'b101011: begin CPU_ctrl_signals = ?; end    //store
            6'b000100: begin CPU_ctrl_signals = ?; end    //beq
            6'b000010: begin CPU_ctrl_signals = ?; end    //jump
            6'h24:      begin CPU_ctrl_signals = ?; end    //slti, 增加ALUop编码
                        .....
            default:    begin CPU_ctrl_signals = ?; end
        endcase
    end
```



控制器仿真激励代码参考

initial begin

// Initialize Inputs

OPcode = 0;

Fun = 0;

MIO_ready = 0;

#40;

// Wait 40 ns for global reset to finish。以上是测试模板代码。

// Add stimulus here

//检查输出信号和关键信号输出是否满足真值表

OPcode = 0; //ALU指令, 检查ALUop=2'b10; RegDst=1; RegWrite=1

Fun = 6'b100000; //add,检查ALU_Control=3'b010

#20;

Fun = 6'b100010; //sub,检查ALU_Control=3'b110

#20;

Fun = 6'b100100; //and,检查ALU_Control=3'b000

#20;

Fun = 6'b100101; //or,检查ALU_Control=3'b001

#20;

Fun = 6'b101010; //slt,检查ALU_Control=3'b111

#20;

Fun = 6'b100111; //nor,检查ALU_Control=3'b100

#20;



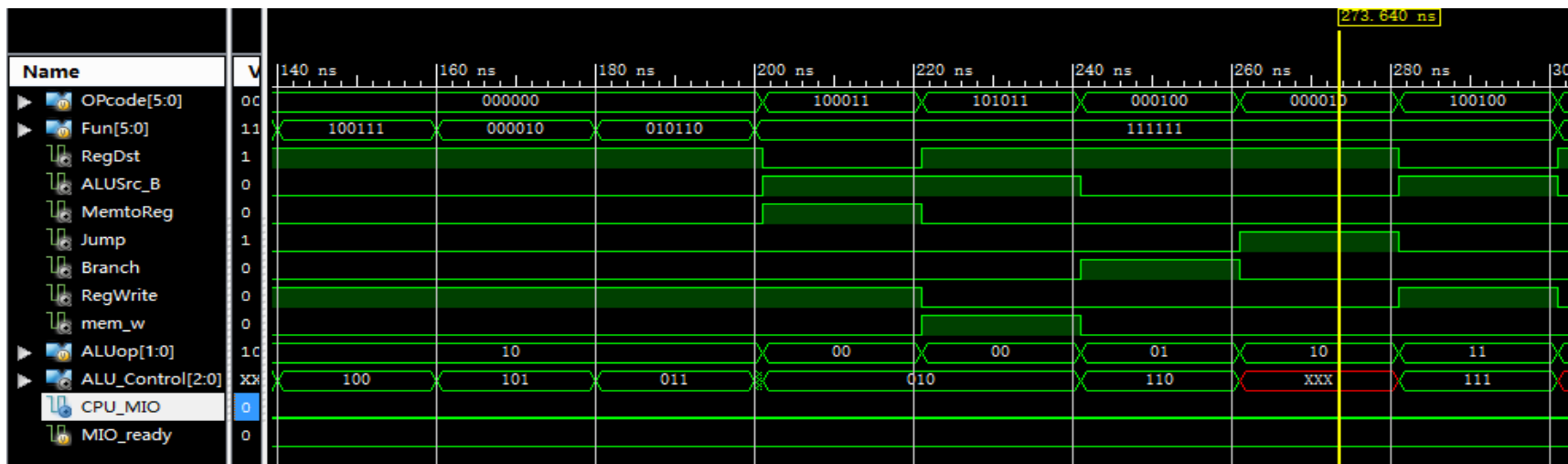
控制器仿真激励代码参考

```
#20;  
Fun = 6'b000010; //srl, 检查ALU_Control=3'b101  
#20;  
Fun = 6'b010110; //xor, 检查ALU_Control=3'b011  
#20;  
Fun = 6'b111111; //间隔  
#1;  
OPcode = 6'b100011; //load指令, 检查ALUop=2'b00, RegDst=0,  
#20; //ALUSrc_B=1, MemtoReg=1, RegWrite=1  
OPcode = 6'b101011;  
#20; //store指令, 检查ALUop=2'b00, mem_w=1, ALUSrc_B=1  
OPcode = 6'b000100; //beq指令, 检查ALUop=2'b01, Branch=1  
#20;  
OPcode = 6'b000010; //jump指令, 检查Jump=1  
#20;  
OPcode = 6'h24; //slti指令, 检查ALUop=2'b11, RegDst=0,  
#20; //ALUSrc_B=1, RegWrite=1  
  
OPcode = 6'h3f; //间隔  
Fun = 6'b000000; //间隔
```

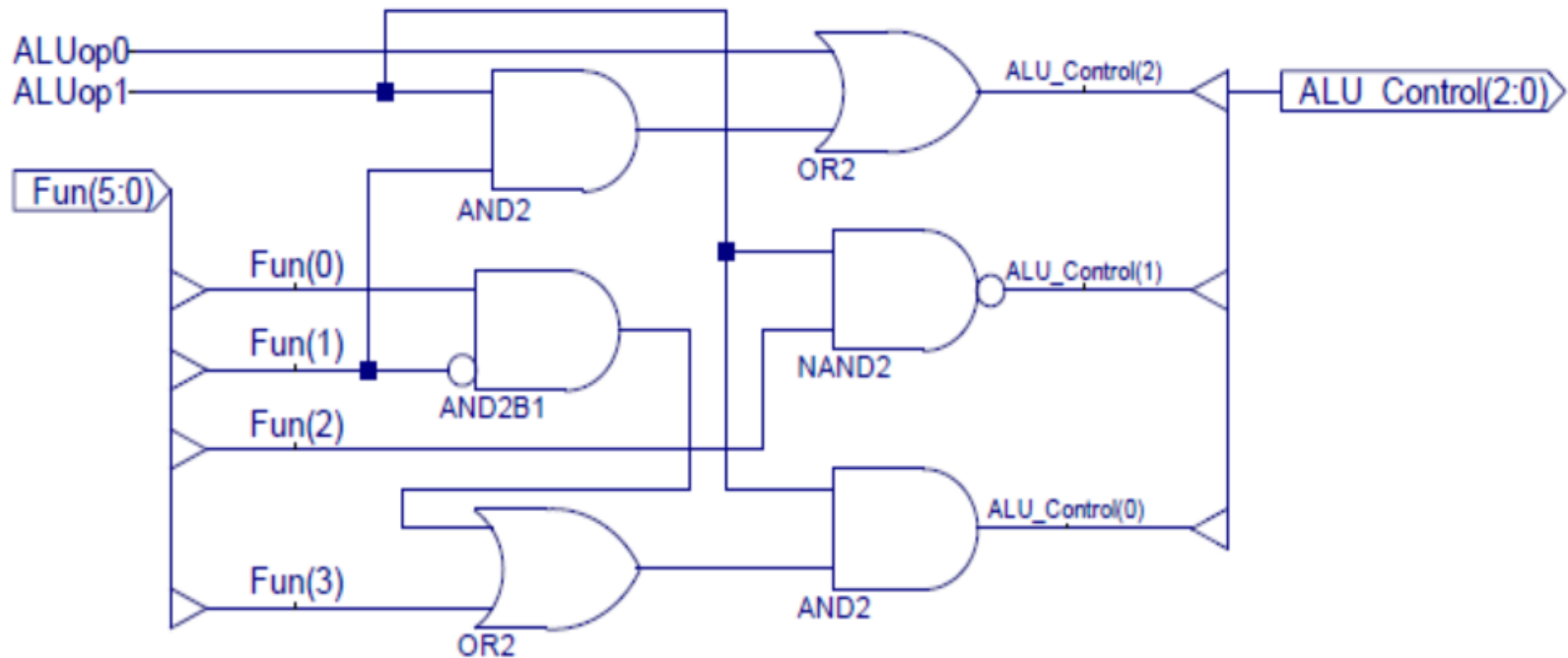
end



控制器模块时序仿真结果



ALU操作译码器逻辑电路





ALU操作译码器HDL描述结构

□ ALU控制器参考描述

```
always @* begin
```

```
  case(ALUOp)
```

```
    2'b00: ALU_Control = ? ;
```

```
//add计算地址
```

```
    2'b01: ALU_Control = ? ;
```

```
//sub比较条件
```

```
    2'b10:
```

```
      case(Fun)
```

```
        6'b100000: ALU_Control = 3'b010 ;    //add
```

```
        6'b100010: ALU_Control = ? ;          //sub
```

```
        6'b100100: ALU_Control = ? ;          //and
```

```
        6'b100101: ALU_Control = ? ;          //or
```

```
        6'b101010: ALU_Control = ? ;          //slt
```

```
        6'b100111: ALU_Control = ? ;          //nor:~(A | B)
```

```
        6'b000010: ALU_Control = ? ;          //srl
```

```
        6'b010110: ALU_Control = ? ;          //xor
```

```
        .....  
      default: ALU_Control=3'bx;
```

```
    endcase
```

```
    2'b11: ALU_Control = ? ;
```

```
//slti
```

```
  endcase
```

控制器集成替换

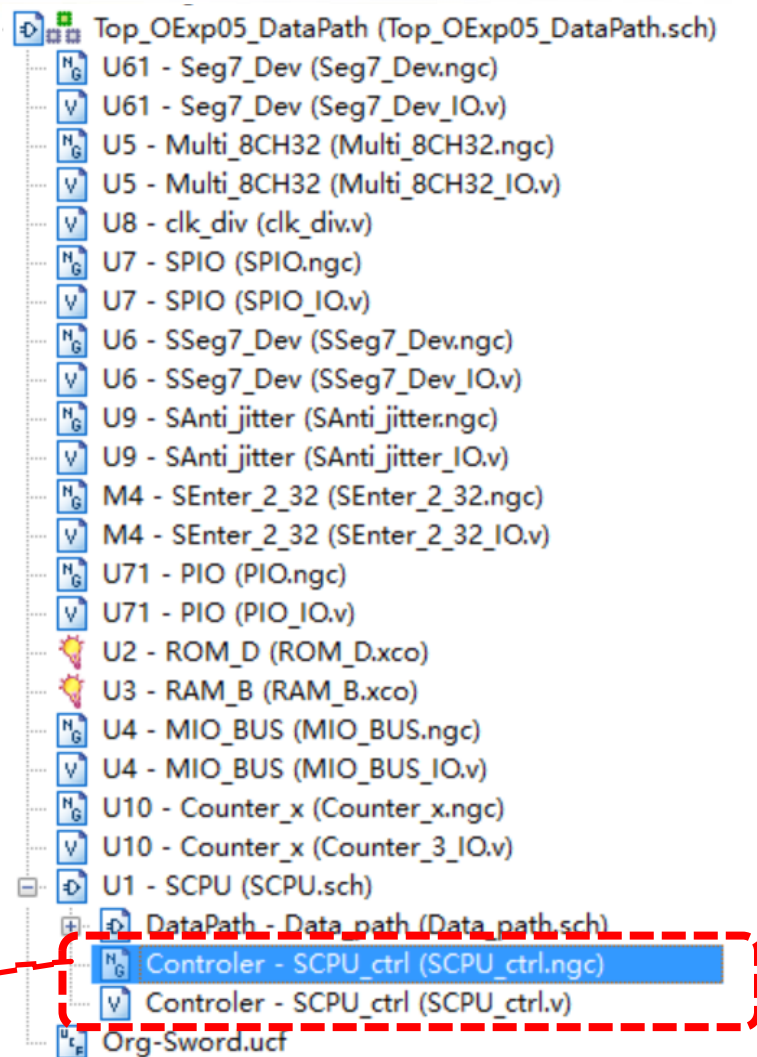
□ 集成替换

- 仿真正确后替换Exp05的控制器IP核

□ 移除工程中的控制器核

- Exp05工程中移控制器核关联
- 删除工程中控制器核文件
 - SCPU_ctrl.ngc文件
 - 在Project菜单中运行:
Cleanup Project Files ...
- 建议用Exp05资源重建工程
 - 除SCPU_ctrl.ngc核

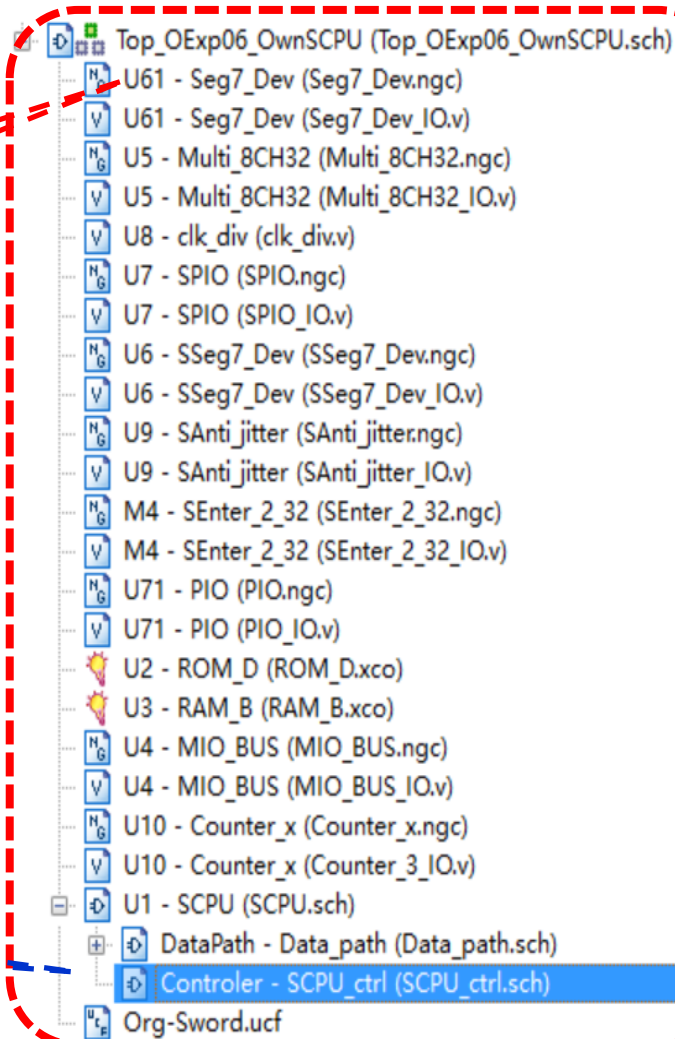
Exp05需要清理的核



集成替换SCPU_ctrl核后的模块层次结构

Exp06完成数据通路替换后的模块调用关系

替换后的控制器模块



□ 使用**DEMO**程序目测控制器功能(同实验五)

■ DEMO接口功能

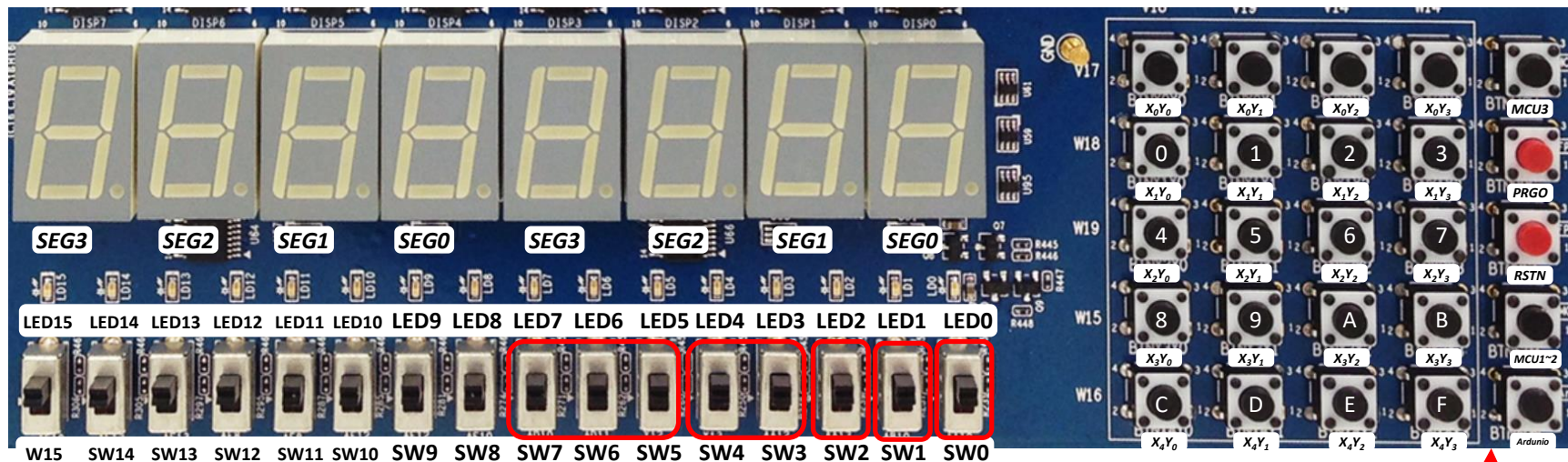
□ SW[7:5]=000, SW[2]=0(全速运行)

- SW[4:3]=00, SW[0]=0, 点阵显示程序: 跑马灯
- SW[4:3]=00, SW[0]=0, 点阵显示程序: 矩形变幻
- SW[4:3]=01, SW[0]=1, 内存数据显示程序: 0~F
- SW[4:3]=10, SW[0]=1, 当前寄存器R9+1显示

□ 用汇编语言设计测试程序

- 测试ALU指令(R-格式译码、Function译码)
- 测试LW/SW指令(I-格式译码)
- 测试分支指令(I-格式译码)
- 测试转移指令(J-格式译码)

物理验证-DEMO接口功能



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

没有使用

DEMO功能, 测试程序可以替换成自己的功能

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



测试程序参考：ALU指令

□ 设计ALU指令测试程序替换DEMO程序

- ALU、Regs测试参考设计，测试结果通过CPU输出信号单步观察
- SW[7:5]=100, Addr_out = ALU输出
- SW[7:5]=101, Data_out= 寄存器B输出

```
#baseAddr 0000
loop:   nor r1,r0,r0;           //r1=FFFFFFFF
        slt r2,r0,r1;          //r2=00000001
        add r3,r2,r2;          //r3=00000002
        add r4,r3,r3;          //r4=00000004
        add r5,r4,r2;          //r5=00000005
        add r6,r5,r5;          //r6=0000000A
        nor r7,r5,r5;          //r7=FFFFFFFA
        sub r8,r7,r5;          //r8=FFFFFFF5
        and r9,r8,r5;          //r9=00000005
        and r10,r8,r6;         //r10=00000000
        or r11,r5,r6;          //r11=0000000F
        or r12,r11,r7;         //r12=0000000A
        slt r13,r5,r7;         //r13=00000000
        .....
j loop;
```



测试程序参考： LW/SW

□ 设计LW/SW程序替换DEMO程序

- 参考Lab5通道测试设计。测试结果通过CPU输出信号单步观察
- 存储器地址通过Addr_out通道4观察： 14+\$zero

#baseAddr 0000

```
start:                                     //通道结果由后一条指令读操作数观察
      lw  r5, 14($zero);                  //取测试常数55555555。存储器读通道

start_A:
      add r1, r5, $zero;                  //r1: 寄存器写通道。R5:寄存器读通道A输出
      nor r2, $zero, r1;                 //r1: 寄存器读通道B输出。R2:ALU输出通道
      lw  r5, 48($zero); //取测试常数AAAAAAAA。立即数通道:00000048
      beq r2, r5 test_sw;                //循环测试
      j   start;                          //循环测试。立即数通道: 00000014
test_sw: .....                          //增加写SW测试, 如14和48单元交换
      j   start;                          //循环测试。立即数通道: 00000014
```

□ 测试的完备性

- 上述测试正确仅表明地址计算、存储单元和总线传输部分正确
- 要测试其完全正确, 必须遍历所有可能的情况



动态LW/SW测试

□ 利用七段显示设备可以设计动态测试程序

- 7段码显示器的地址是E0000000/FFFFFFE0
- LED显示地址是F0000000/FFFFFF00
- SW指令输出测试结果: sw
- 请设计存储器模块测试程序
 - 测试结果在7段显示器上指示

□ RAM初始化数据同Lab5

F0000000, 000002AB, 80000000, 0000003F, 00000001, FFF70000,
0000FFFF, 80000000, 00000000, 11111111, 22222222, 33333333,
44444444, 55555555, 66666666, 77777777, 88888888, 99999999,
aaaaaaaa, bbbbbbbb, cccccccc, dddddddd, eeeeeeee, FFFFFFFF,
557EF7E0, D7BDFBD9, D7DBFDB9, DFCFFCFB, DFCFBFFF, F7F3DFFF,
FFFFDF3D, FFFF9DB9, FFFFBCFB, DFCFFCFB, DFCFBFFF, D7DB9FFF,
D7DBFDB9, D7BDFBD9, FFFF07E0, 007E0FFF, 03bdf020, 03def820,
08002300;



设计测试记录表格

- ALU指令测试结果记录
 - 自行设计记录表格
- LW/SW指令测试结果记录
 - 自行设计记录表格
- 动态存储模块测试记录
 - 自行设计记录表格



思考题

- 单周期控制器时序体现在那里？
- 设计bne指令需要增加控制信号吗？
- 扩展下列指令，控制器将作如何修改：
 - R-Type: srl*, jr, jalr, eret*;
 - I-Type: addi, andi, ori, xori, lui, bne, slti
 - J-Type: Jal;
- 此时用二级译码有优势吗？
- 动态存储模块测试七段显示会出现什么问题？



● END