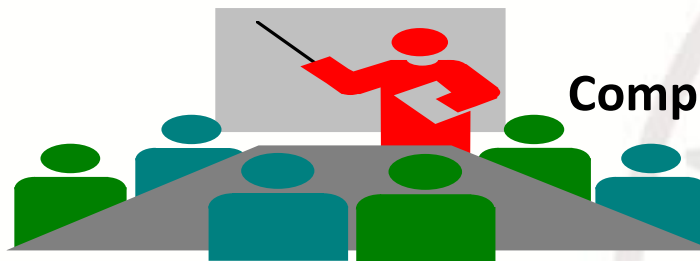




浙江大学  
ZHEJIANG UNIVERSITY



Computer Organization & Design

# Computer Organization & Design

## 实验与课程设计

### 实验八

### CPU设计-中断

施青松

Asso. Prof. Shi Qingsong

College of Computer Science and Technology, Zhejiang University

[zjsqs@zju.edu.cn](mailto:zjsqs@zju.edu.cn)

---

# Course Outline





# 实验目的

1. 深入理解CPU结构
3. 学习如何提高CPU使用效率
3. 学习CPU中断工作原理
4. 扩展设计简单中断
5. 设计中断测试程序



# 实验环境

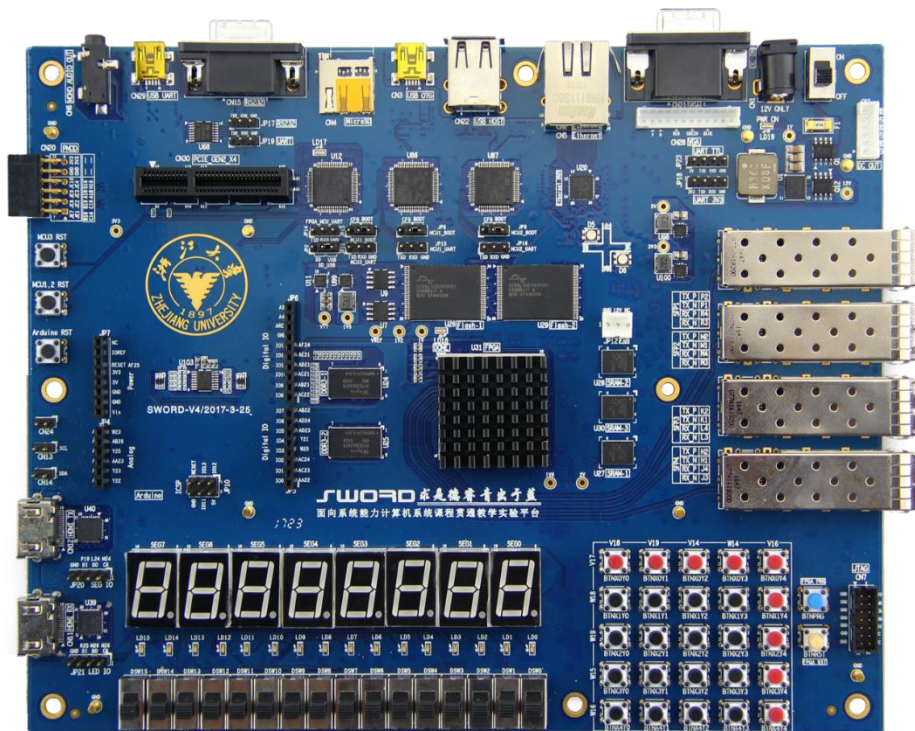
## □ 实验设备

1. 计算机（Intel Core i5以上，4GB内存以上）系统
2. 计算机软硬件课程贯通教学实验系统(SWORD4.0)
3. Xilinx ISE14.7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XC7K325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

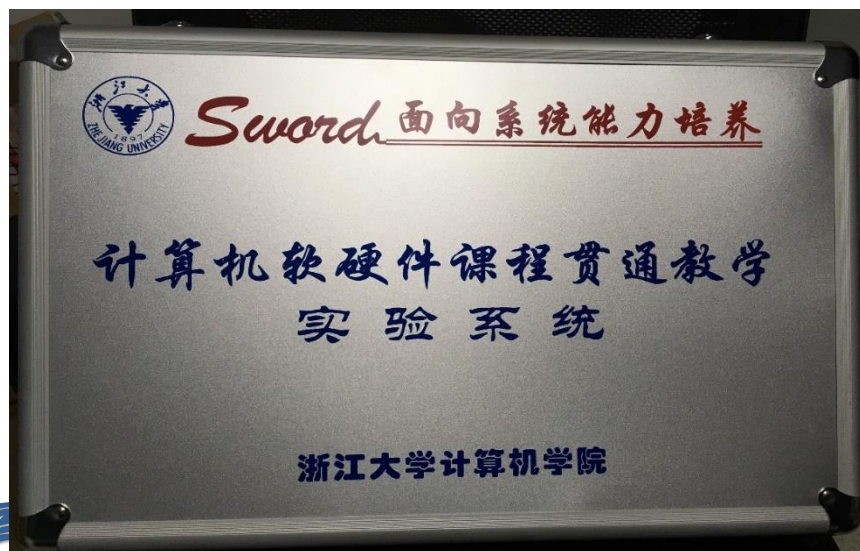
12位VGA接口 (RGB656)、USB-HID (键盘)

### ▼ 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：选修

## ■ 基本要求

- 增加SCPU中断功能
  - 兼容SCPU数据通路
  - 增加中断通路
  - 增加中断控制
- 固定中断向量(ARM)
  - 非法指令中断
  - 外部中断
- 此实验在兼容Exp07基础上完成

## ■ 高级要求

- 支持CP0中断相关寄存器
- 支持CP0中断相关指令
- 半兼容模式
  - 统一入口地址：00000004



# Course Outline

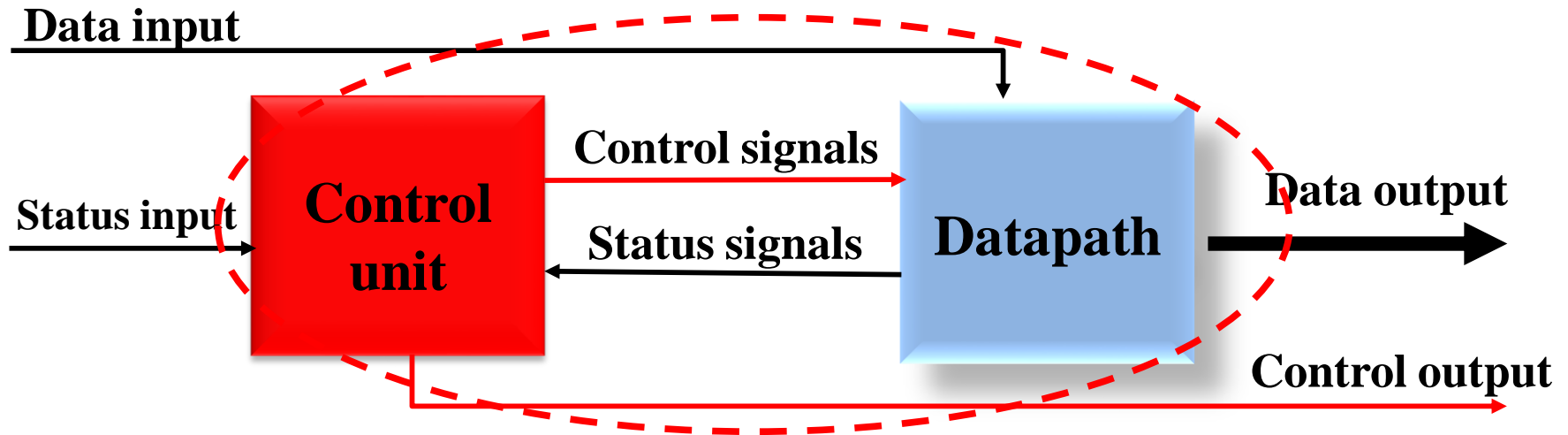




# CPU organization

## □ Digital circuit

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



## □ Computer organization

- Special circuits that processes logical action with instructions  
**-Software**



# MIPS中断结构

## □ 协处理器CP0

### ■ MIPS用来辅助的部件

- 处理异常
- 存储器管理
- 系统配置
- 其他片上功能

## □ 常用CP0寄存器

寄存器	编号	作用
BadVAddr	8	最近内存访问异常的地址
Count	9	高精度内部计时计数器
Compare	11	定时常数匹配比较寄存器
<b>Status(SR)</b>	<b>12</b>	<b>状态寄存器、特权、中断屏蔽及使能等，可位控</b>
<b>Cause</b>	<b>13</b>	<b>中断异常类型及中断持起位</b>
<b>EPC</b>	<b>14</b>	<b>中断返回地址</b>
Config	16	配置寄存器，依赖于具体系统



# CPO传输指令

## □ 控制寄存器访问指令

### ■ 读CP0指令mfco

□ mfco rt,rd:  $GPR[rt] \leq CP0[rd]$

Op=6bit	rs=00000	rt=5bit	Rd=5bit	=11个0
0x10	0	rt	rd	00000 000000

### □ 写CP0mtco指令

■ mtc0 rd, rt:  $GPR[rd] \leq CP0[rt]$

Op=6bit	rs=00000	rt=5bit	Rd=5bit	=11位0
0x10	4	rt	rd	00000 000000



# 中断相关指令

## □ 异常返回

### ■ eret

- $PC \leq EPC$ ; (CP0的Cause和Status寄存器有变化)

Op=6bit	1	19bit	FUN
0x0	1	000 0000 0000 0000 0000	011000

## □ 系统调用

### ■ syscall

- $EPC = PC + 4$ ;  $PC \leq$  异常处理地址; Cause和Status寄存器有变化

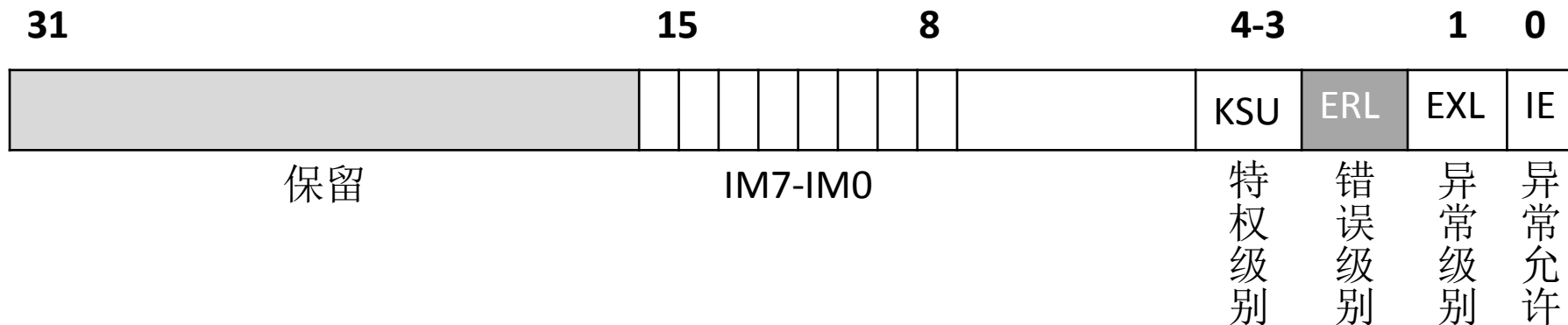
### ■ 参数:

- \$v0=系统调用号: Fig B-9-1
- \$a0~\$a3、\$f12, 返回在\$v0

Op=6bit	20bit	FUN
0x0	0000 0000 0000 0000 0000	0011000



# 状态寄存器: Status(SR)



- **IE:** IE=1全局中断使能
- **EXL:** 异常设置, 强制CPU进入内核模式并关闭中断, 优先级高于IE
- **ERL:** 数据错误设置, 进入内核模式并关闭中断, 优先级高于IE
- **IM7-0:** 8个中断屏蔽位, 6个外中断, 2个软中断。没有优先级
- **KSU:** 00=内核态、01=监管模式、10=用户态。优先级低于EXL和ERL



中断挂起位  
IP0~IP7

异常编码  
ExcCode

浙江大学  
ZheJiang University



# MIPS中断响应

## □ 初始化

- 设置SR: 关中断  $IE=0$ ,  $KSU\ ERL\ EXL=00\ 0\ 0$
- 系统初始化
- 设置SR: 开中断  $IE=1$ , 设置IM7-0

## □ 中断响应

- 硬件保存断点:  $EPC \leftarrow PC+4$
- 硬件修改PC:  $PC \leftarrow$  向量、硬件关中断
- 中断服务: 保护寄存器、开中断\*、服务、恢复寄存器
- 开中断\*
- 返回: `eret` (硬件修改Cause和Status寄存器)





# 中断数据通路和控制器

## □ 数据通路

- 需要增加CP0的Cause和Status寄存器
- 必须增加EPC寄存器及 $EPC \leftarrow PC + 4$ 通路
- 增加mfc0、mtc0和eret指令通道

## □ 控制器

- 增加中断检测电路
- 增加异常检测电路
- 中断响应控制
- CP0传输控制



# 典型处理器中断结构

## □ Intel x86中断结构

- 中断向量：000~3FF，占内存最底1KB空间
  - 每个向量由二个16位生成20位中断地址
  - 共256个中断向量，向量编号 $n=0\sim 255$
  - 分硬中断和软中断，响应过程类同，触发方式不同
  - 硬中断响应由控制芯片8259产生中断号 $n$ (接口原理课深入学习)

## □ ARM中断结构

- 固定向量方式(嵌入式课程深入学习)

异常类型	偏移地址(低)	偏移地址(高)	
复位	00000000	FFFF0000	
未定义指令	00000004	FFFF0004	
软中断	00000008	FFFF0008	
预取指令终	0000000C	FFFF000C	
数据终止	00000010	FFFF0010	
保留	00000014	FFFF0014	
中断请求(IRQ)	00000018	FFFF0018	
快速中断请求(FIQ)	0000001C	FFFF001C	

# Course Outline





# 中断设计内容

## □ 确定中断向量模式

- 中断向量：中断入口寻址方法
  - 固定：直接给出中断入口地址
  - 动态：计算获得中断入口地址

## □ 数据通路

- 需要**增加**CP0的Cause和Status寄存器
- 必须**增加**EPC寄存器及 $EPC \leftarrow PC + 4$ 通路
- **增加**mfc0、mtc0和eret指令通道

## □ 控制器

- **增加**中断检测电路
- **增加**异常检测电路
- **增加**中断响应控制
- **增加**CP0传输控制



# 简化中断设计：ARM模式

## □ ARM中断向量表

向量地址	ARM异常名称	ARM系统工作模式	本实验定义
<b>0x0000000</b>	复位	超级用户Svc	内核模式
<b>0x0000004</b>	未定义指令终止	未定义指令终止Und	<b>RI内核模式</b>
<b>0x0000008</b>	软中断（SWI）	超级用户Svc	Sys系统调用
<b>0x000000c</b>	Prefetch abort	指令预取终止Abt	Reserved自定义
<b>0x0000010</b>	Data abort	数据访问终止Abt	Ov
<b>0x0000014</b>	Reserved	Reserved	Reserved自定义
<b>0x0000018</b>	IRQ	外部中断模式IRQ	<b>Int外中断（硬件）</b>
<b>0x000001C</b>	FIQ	快速中断模式FIQ	Reserved自定义

## □ 简化中断设计

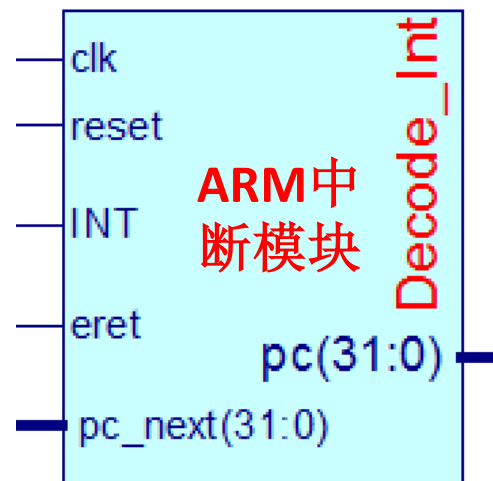
- 采用ARM中断向量(不兼容MIPS)
  - 实现非法指令异常和外中断
  - 设计EPC
- 兼容MIPS\*
  - Cause和Status寄存器
  - 设计mfc0、mtc0指令

# DataPath扩展中断通路

## □ DataPath修改

- 修改PC模块增加(ARM模式)
  - CPU复位时IE=0, EPC=PC=0x00000000
  - IE=中断使能(重要)
  - EPC寄存器, INT触发PC转向中断地址
    - 相当于硬件触发Jal, 用eret返回
  - 增加控制信号INT、RFE/eret
    - INT宽度根据扩展的外中断数量设定
- 修改PC模块增加(MIPS模式)\*
  - CP0简化模块
    - EPC、Cause和Status寄存器
    - 增加CP0数据通道
      - EPC、Cause和Status通道
    - 增加控制信号CP0\_Write
  - 修改PC通道

**注意: INT是电平信号, 不要重复响应**





# 控制器扩展中断译码

## □ 控制器修改

### ■ ARM模式(简单)

- 仅增加eret指令
- 中断请求信号触发PC转向，在Datapath模块中修改

### ■ MIPS模式(可独立模块)\*

- 扩展mfc0、mtc0指令译码
- 增加Wt\_Write通道选择控制
- 增加CP0\_Write
- 增加控制信号eret、RI、CP0\_Write

## □ 中断调试

### ■ 首先时序仿真

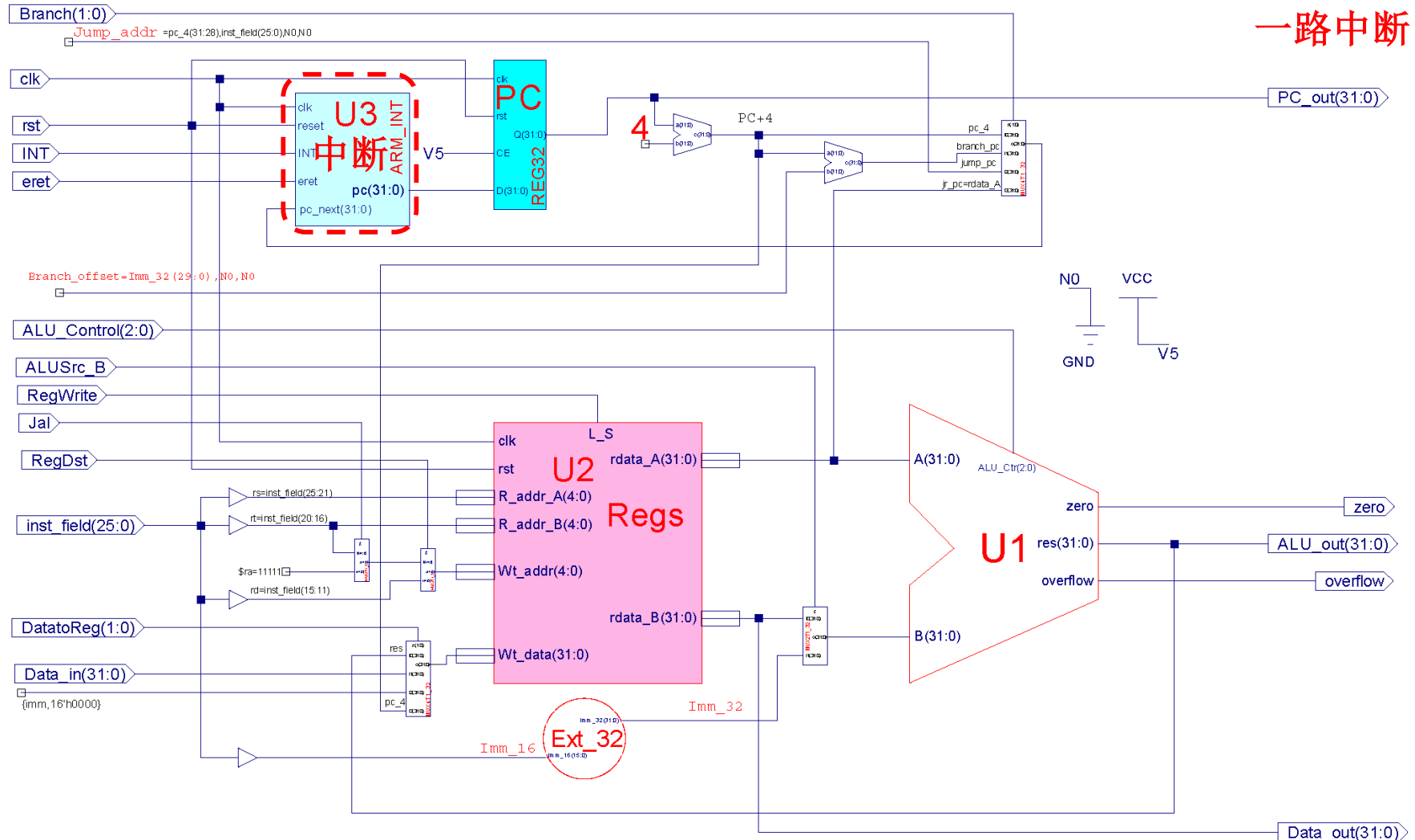
### ■ 物理验证

- 用BTN[0]触发调试：静态或低速
- 用计数器counter1\_OUT调试：动态或高速

**注意动态测试时死锁!!!**



# 增加ARM中断模式的DataPath





# 中断通路模块：检测与锁存

## □ 中断触发检测与服务锁存

```
assign int_clr = reset|int_act;           //clear interrupt Request
always @(posedge INT or                  //interrupt Request
        posedge int_clr )begin           //clear interrupt Request
    if(int_clr==1 )int_req_r<=0;         //clear interrupt Request
    else int_req_r<=1;                   //set interrupt Request
end
```

## □ 断点保护、中断开、并与返回

```
always @(posedge clk or posedge reset ) begin
    if (reset)begin EPC      <= 0;        //EPC=32'h00000000;
                    int_act <= 0;
                    int_en  <= 1;
    end
    else if(int_req_r & int_en)begin //int_req_r: interrupt Request reg
        EPC<= pc_next;              //interrupt return PC
        int_act<=1;                  //interrupt Service
        int_en<=0;                   //interrupt disable
    end
    else begin int_act<=0;
        if(eret) int_en<=1;         //interrupt enable if pc<=EPC;
    end
end
```

# 中断通路模块：PC通路

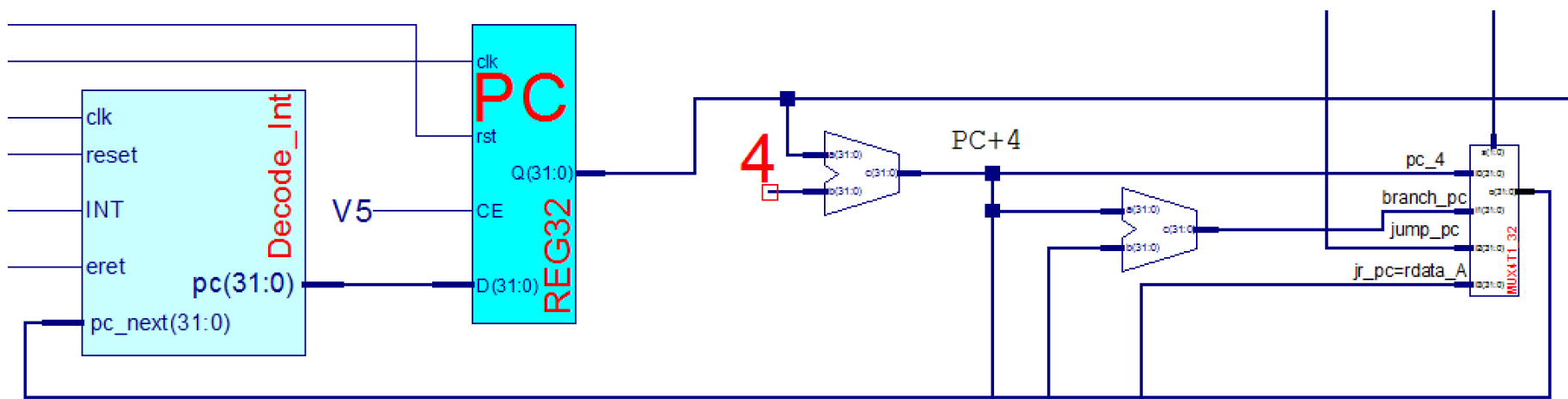
## □ PC输出通路

```

always @*begin
    if (reset==1)pc<=32'h00000000;
    else if(int_req_r & int_en )
        pc<=32'h00000004;           //interrupt Vector
    else if(eret)pc <= EPC;         //interrupt return
    else pc <= pc_next;            //next instruction
end

```

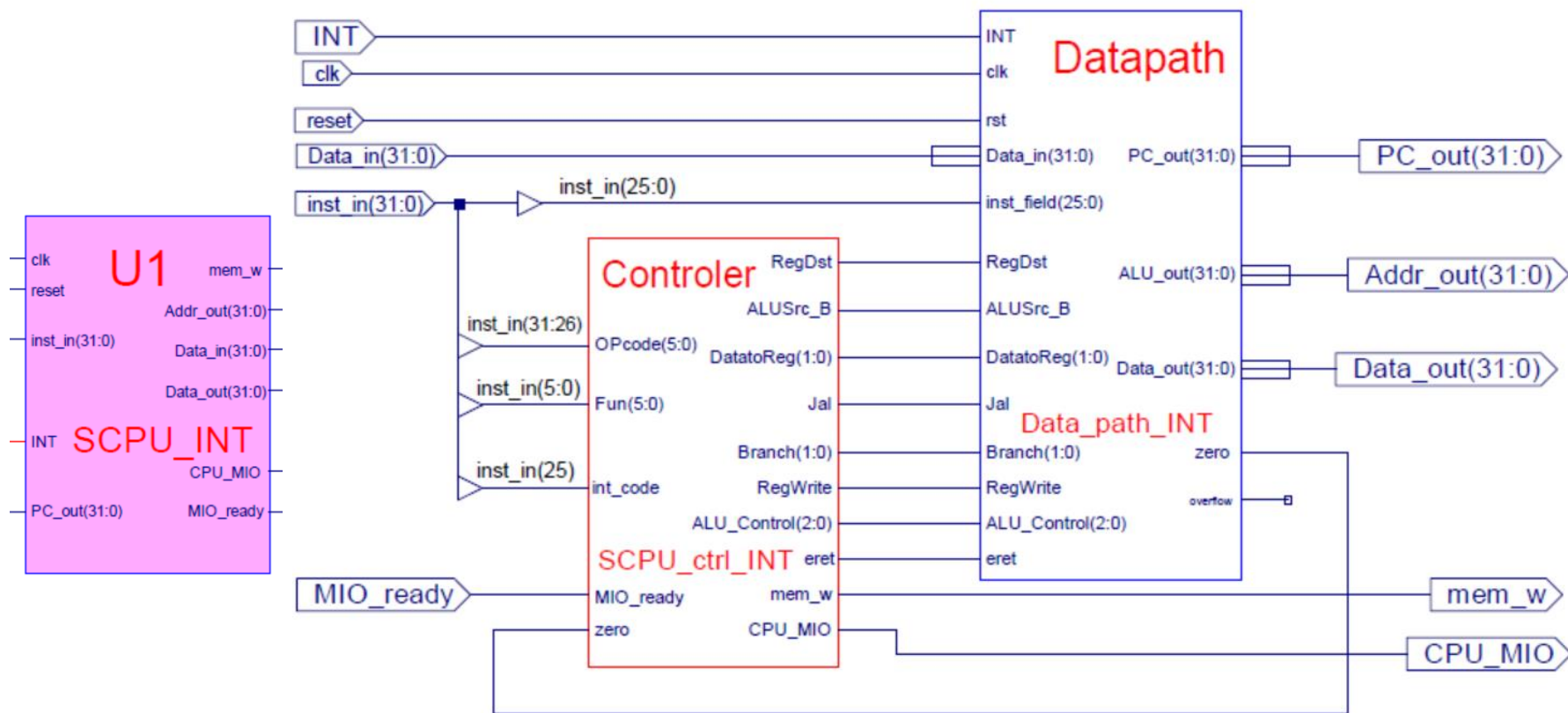
[ 仅实现一路中断，请同学们扩展 ]



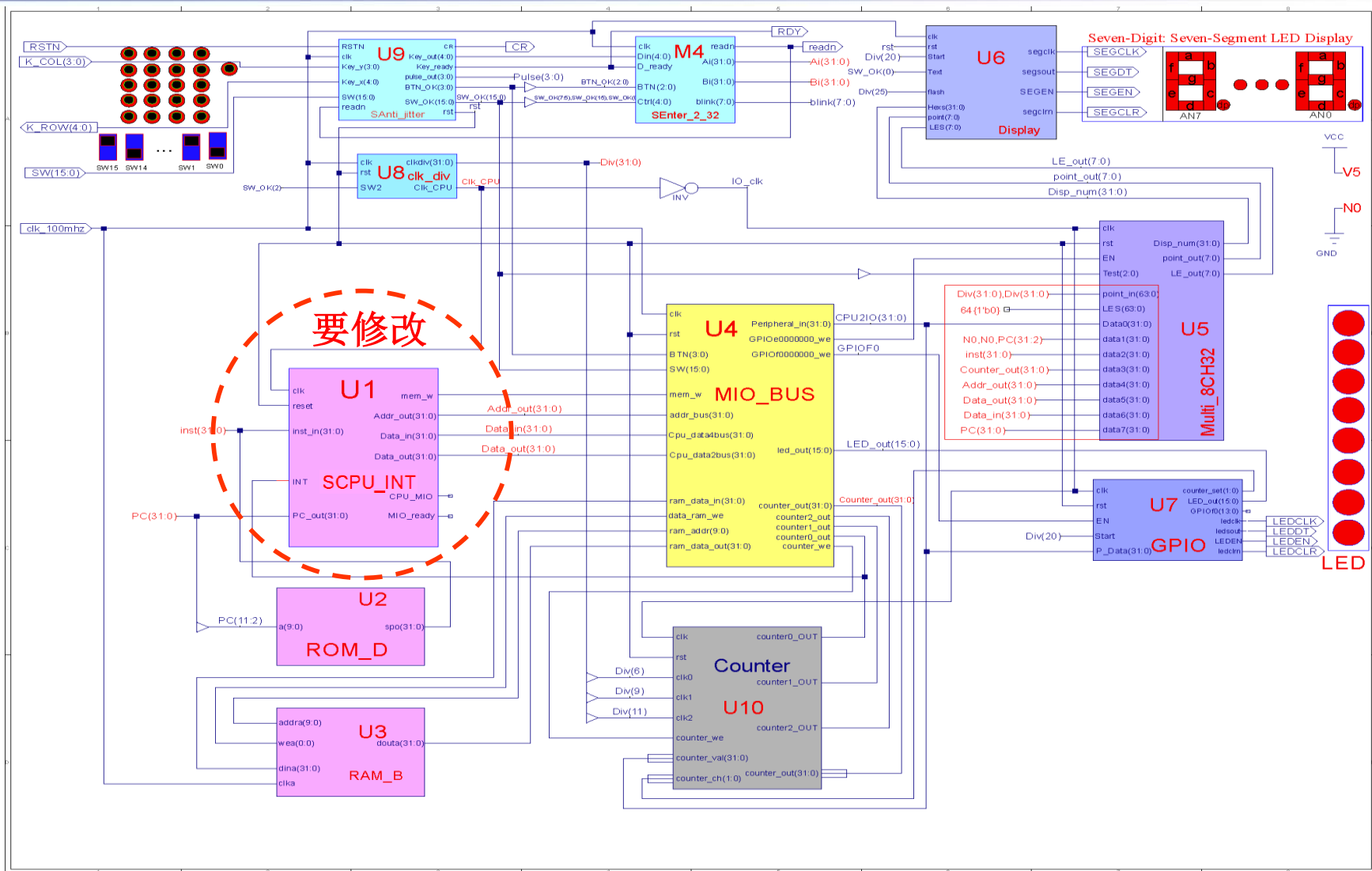
# 增加中断后的CPU模块

## □ 注意修改模块逻辑符号

⌘ SCPU\_INT.sym、SCPU\_ctrl\_INT.sym和Data\_path\_INT.sym



# 利用SOC作为调试测试环境





# 修改功能测试程序：判断子代码

loop1:

```
lw $a1, 0($v1);    //读GPIO端口F0000000状态，同前。
add $a1, $a1, $a1;
add $a1, $a1, $a1;  //左移2位将SW与LED对齐
sw $a1, 0($v1);    //再将新$a1:r5写到GPIO端口F0000000,写到LED
lw $a1, 0($v1);    //再读GPIO端口F0000000状态
and $t3,$a1,$t0;    //与80000000相与，即：取最高位=out0,屏蔽其余位
beq $t3,$t0,C_init; //硬件计数。out0=1,Counter通道0溢出,中断转C_init
add $t5, $t5, $v0;  //程序计数延时(加1)
// beq $t5, $zero,C_init; //若程序计数$t5:r13=0,转C_init
```

SW状态  
循环显示

三种定时选择

```
l_next:    // 延时未到，继续：判断7段码显示模式：SW[4:3]
lw $a1, 0($v1);    //再读GPIO端口F0000000开关SW
add $s2, $t6, $t6;  //因$t6:r14=4, 故$s2:r18=00000008
add $s6, $s2, $s2;  // $s6:r22=00000010
add $s2, $s2, $s6;  // $s2:r18=00000018(00011000)
and $t3, $a1, $s2;  //取SW[4:3]到$t3
beq $t3, $zero, L20; //SW[4:3]=00,7段显示"点"左移反复
beq $t3, $s2, L21;  //SW[4:3]=11, 显示七段图形
add $s2, $t6, $t6;  // $s2:r18=8
beq $t3, $s2, L22;  //SW[4:3]=01,七段显示预置数字
```

.....



# 修改定时子代码： 中断返回

```
C_init:                                     //延时结束，修改显示值和定时/延时初始化
    lw $t5, 14($zero);                     //取程序计数延时初始化常数
    add $t2, $t2, $t2;                     // $t2:r10=ffffffc, 7段图形点左移
    or $t2, $t2, $v0;                     // $t2:r10末位置1，对应右上角不显示
    add $s1, $s1, $t6;                     // $t6:r14=00000004, LED图形访存地址加1
    and $s1, $s1, $s4;                     //与3F相与，留下后6位。$s1:r17=000000XX, //屏蔽地址高位
    add $t1, $t1, $v0;                     //r9+1
    beq $t1, $at, L6;                     //若r9=ffffff,重置r9=5
    j L7;

L6:                                         //r9=4
    add $t1, $zero, $t6;                 //重置r9=5
    add $t1, $t1, $v0;

L7:                                         //读GPIO端口F0000000状态
    lw $a1, 0($v1);
    add $t3, $a1, $a1;
    add $t3, $t3, $t3;                     //左移2位将SW与LED对齐，同时
    sw $t3, 0($v1);                       //r5输出到GPIO端口F0000000
    sw $a2, 4($v1);                       // 计数器端口:F0000004，送计数常数
    eret;
```

**[注意动态测试时死锁!!!]**





## □ 思考题

- 如何设计中断静态测试(BTN0)?
- 如何扩展中断通路模块实现多路中断?



● END