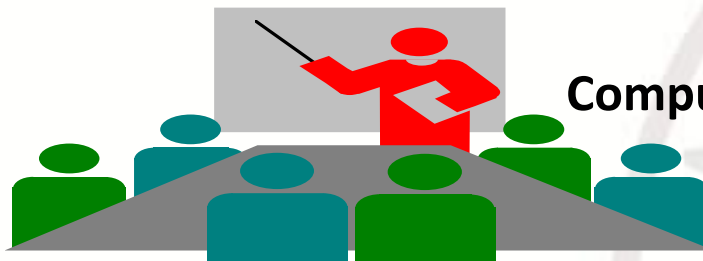




浙江大学
ZHEJIANG UNIVERSITY



Computer Organization & Design

Computer Organization & Design

实验与课程设计

实验九

多周期IP核集成CPU

--建立多周期CPU调试、测试和应用环境

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. 深入理解**CPU**结构
3. 学习CPU性能优化:多周期
3. 建立多周期**CPU**测试应用环境
4. **IP**核深入应用

□ 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

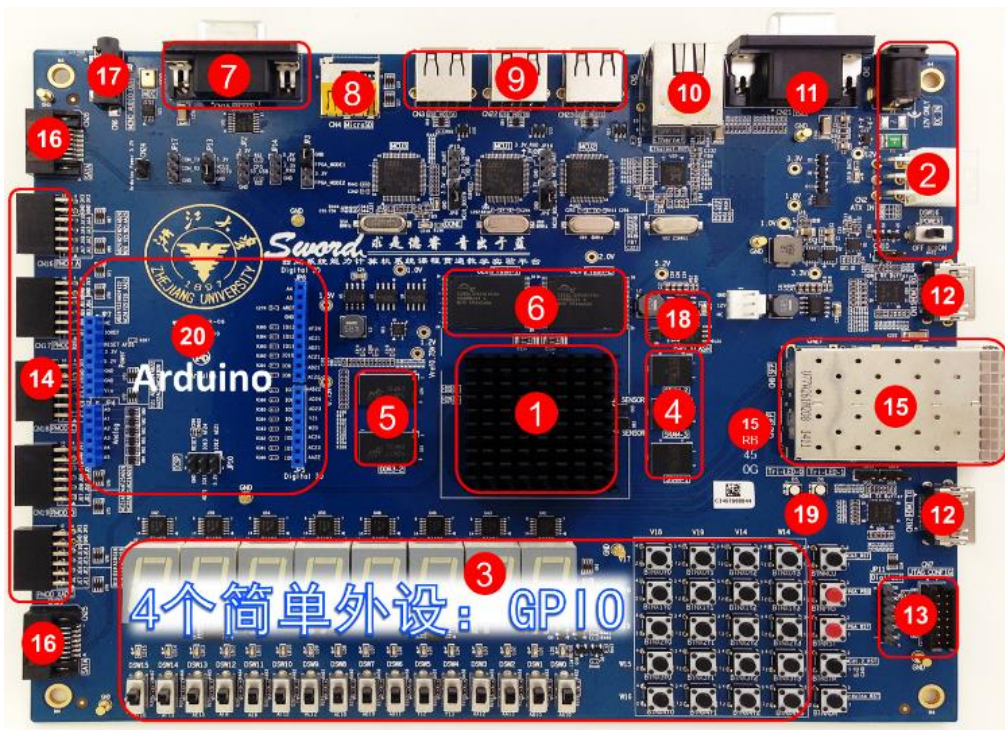
6MB SRAM静态存储器: 支持32Data, 16位TAG

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

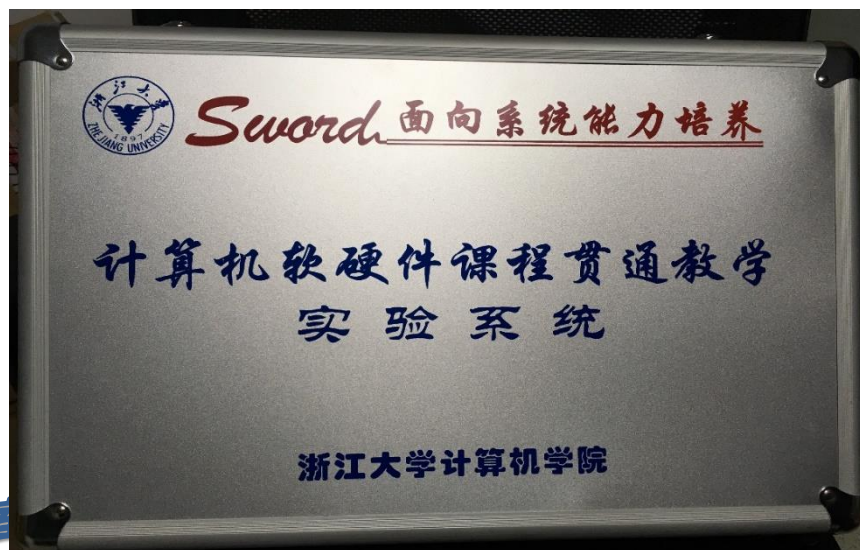
12位VGA接口 (RGB656)、USB-HID (键盘)

▼ 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. 搭建多周期CPU测试应用环境

- 用结构描述重建Exp03顶层模块
 - 用多周期CPU核替换单周期

2. 用多周期数据通路和控制器核集成CPU核

- 除CPU外复用Exp03的部件模块

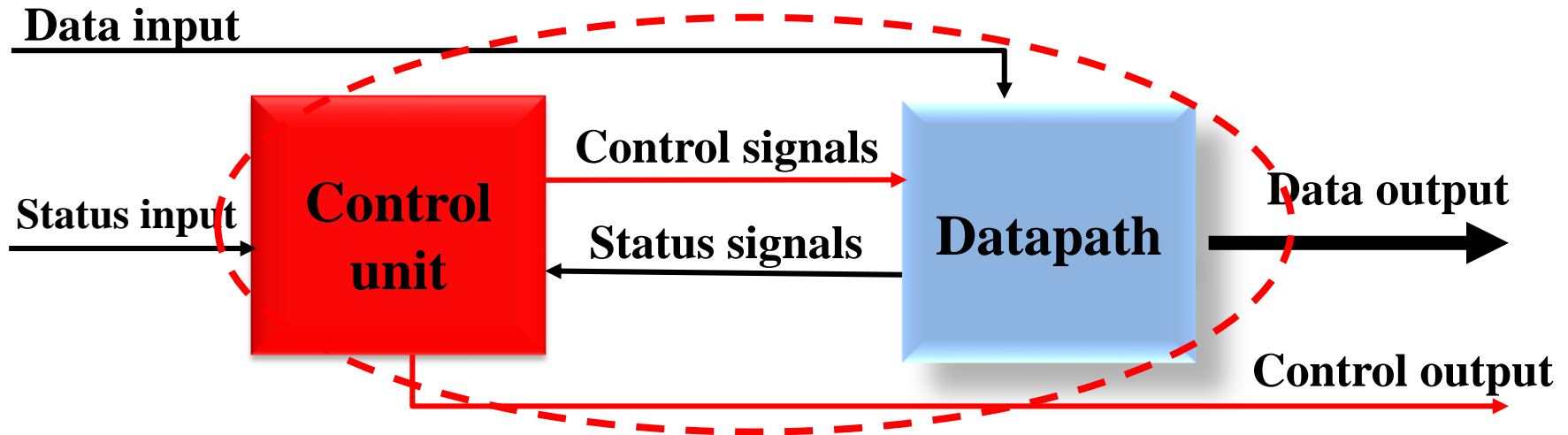
Course Outline



CPU organization

□ Digital circuit

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



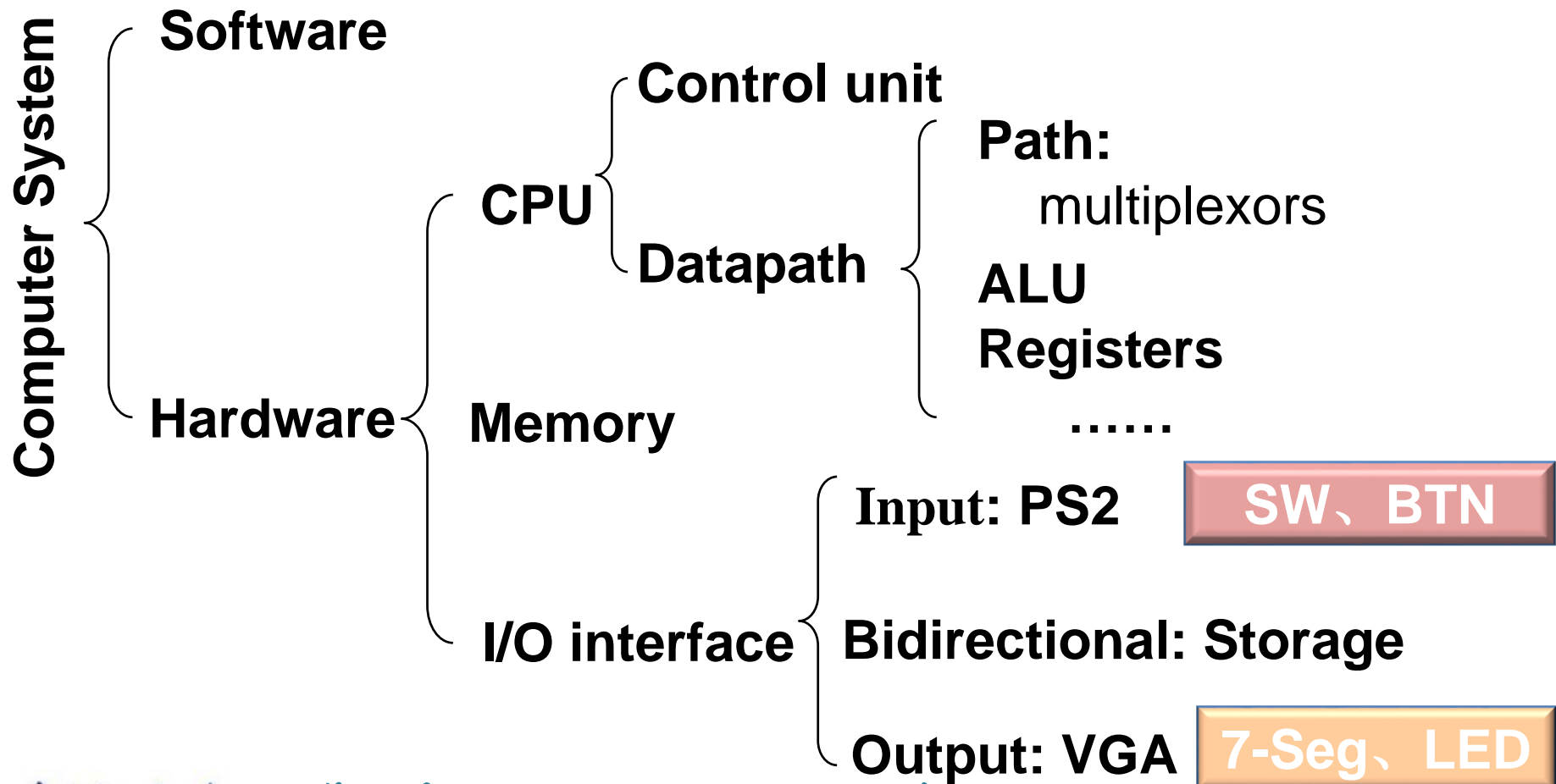
□ Computer organization

- Special circuits that processes logical action with instructions
-Software



Computer Organization

□ Decomposability of computer systems







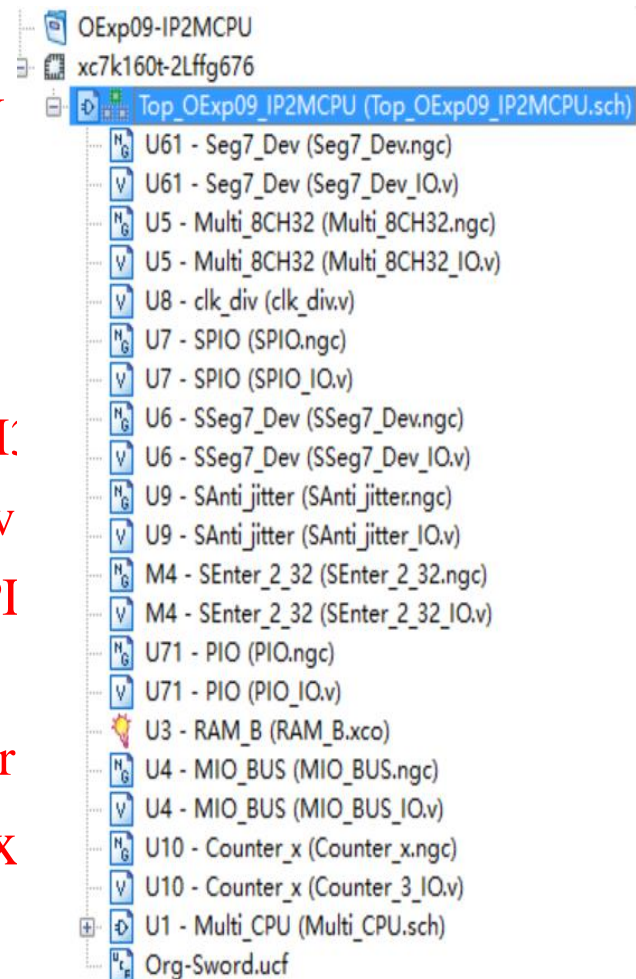
系统分解为九个子模块

□ 用此9个模块，用结构描述建立SOC测试构架

□ 集成实现多周期处理器SOC

- U1: MCPU -Muliti-CPU
- U2: ROM -ROM_D
- U3: RAM -RAM_B
- U4: 总线(含外设3~4) -MIO_BUS
- U5: 7段显示接口 -Multi_8CH
- U6: 外设1- 7段显示设备 -SSeg_Dev
- U7: 外设2-GPIO接口及LED -SPI
- U8: 辅助模块一，通用分频模块 -clk_div
- U9: 辅助模块二，机械去抖模块 -SAnti_jitter
- U10: 通用计数器 -Counter_x

□ 以上除U1外与单周期共享





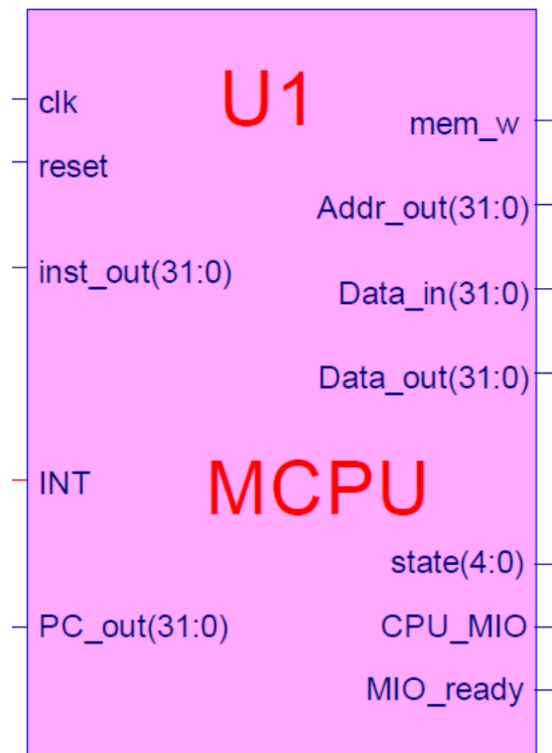
U1-多周期CPU模块: MCPU

□ MIPS 构架

- ⊙ RISC体系结构
- ⊙ 三种指令类型

□ SOC测试模块的处理核心

- 由数据通路和控制器二个核组成
 - 本实验直接用2个IP核集成
 - 本实验也可先用CPU IP核
 - 调试通过后再用2个IP集成
- 本实验可用IP Core- **U1**
 - 核调用模块Multi_CPU.ngc
 - 核接口信号模块(空文档): Multi_CPU.v
 - 核模块符号文档: Multi_SCPU.sym





CPU核接口空模块-MCPU.v

module

MCPU (input wire clk,
input wire reset,
input wire MIO_ready, // be used: =1

output wire[31:0]PC_out, //Test
output[31:0] inst_out, //TEST
output wire mem_w, //存储器读写控制
output wire[31:0]Addr_out, //数据空间访问地址
output wire[31:0]Data_out //数据输出总线
input wire [31:0]Data_in, //数据输入总线
output wire CPU_MIO, // Be used
input wire INT //中断
output[4:0]state //Test
);

注意与单周期区别

endmodule

CPU部件之一-数据通路: MDPath



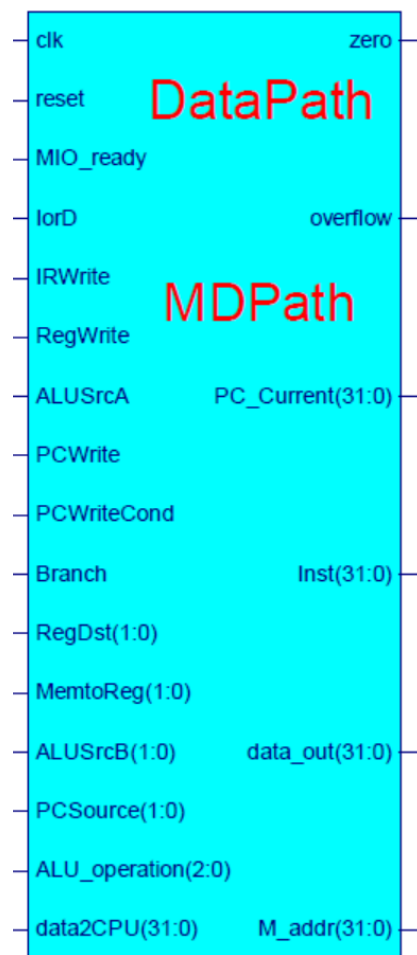
MUX选择更多输入以兼容扩展

□ 本实验用IP 软核- MDPath

- 核调用模块M_atapath.ngc
- 核接口信号模块: M_datapath.v
- 核模块符号文档: M_datapath.sym

□ 重要信号

- Inst: 指令寄存器输出
- PC_Current: 当前PC(PC+4)
- M_addr: 存储器地址
- Branch(教材中的beq):
 - =1: beq
 - =0: bne
- PCWriteCond: Branch指令



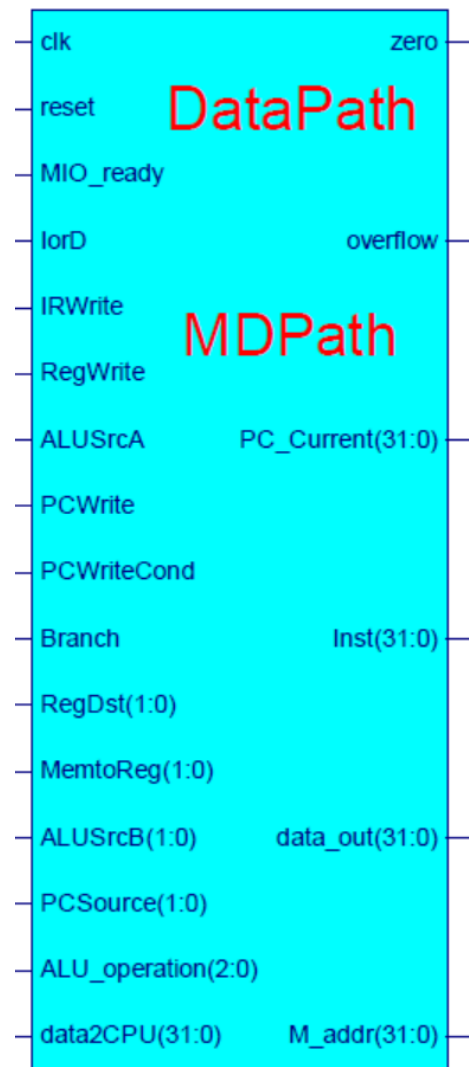


数据通路空模块- MDPath.v

```
module MDPath(input clk,
               input reset,           //=1
               input MIO_ready,
               input IorD,
               input IRWrite,
               input[1:0] RegDst,      //预留到2位
               input RegWrite,
               input[1:0] MemtoReg,    //预留到2位
               input ALUSrcA,
               input[1:0] ALUSrcB,
               input[1:0] PCSource,    //4选1控制
               input PCWrite,
               input PCWriteCond,
               input Branch,
               input[2:0] ALU_operation,

               output[31:0] PC_Current,
               input[31:0] data2CPU,
               output[31:0] Inst,
               output[31:0] data_out,
               output[31:0] M_addr,
               output zero,
               output overflow
               );

endmodule
```





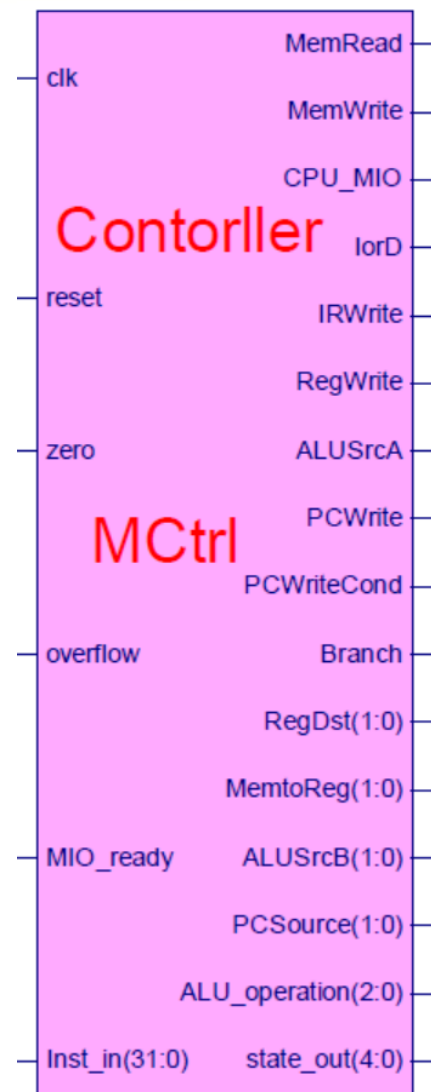
CPU部件之二-控制器: MCtrl

□ 本实验用IP 软核- MCtrl

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

□ 重要信号

- MIO_ready: 外设就绪
 - =0 CPU等待
 - =1 CPU正常运行
 - 本实验恒等于1
- Inst_in: 指令输入, 来自IR输出
- State_out: 状态编码, 用于测试



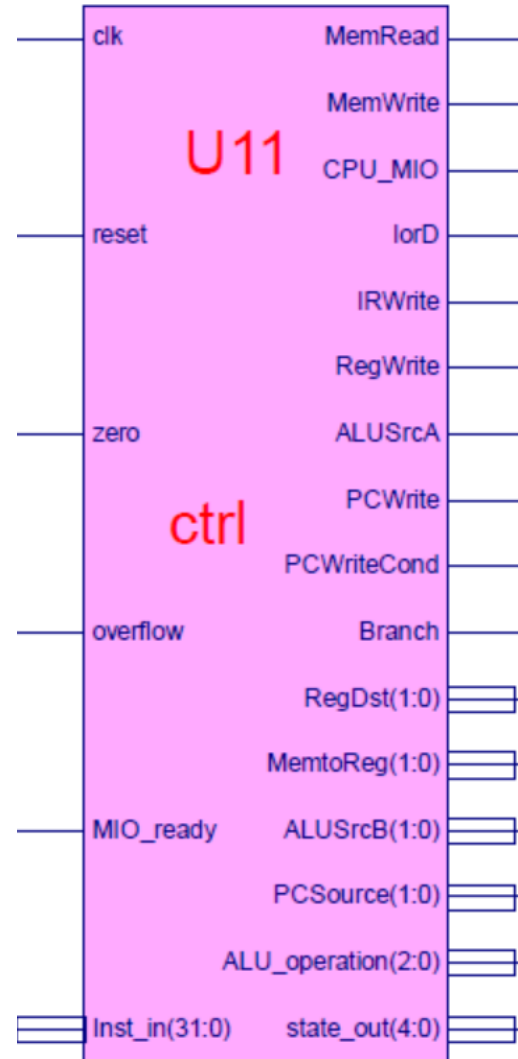


控制器接口文档- ctrl.v

```
module MCtrl(input clk,
             input reset,
             input [31:0] Inst_in,
             input zero,
             input overflow,
             input MIO_ready,
             output reg MemRead,
             output reg MemWrite,
             output reg [2:0] ALU_operation,
             output [4:0] state_out,

             output reg CPU_MIO,
             output reg IorD,
             output reg IRWrite,
             output reg [1:0] RegDst,
             output reg RegWrite,
             output reg [1:0] MemtoReg,
             output reg ALUSrcA,
             output reg [1:0] ALUSrcB,
             output reg [1:0] PCSource,
             output reg PCWrite,
             output reg PCWriteCond,
             output reg Branch
);

endmodule
```





U3-指令代码存储模块: RAM_B

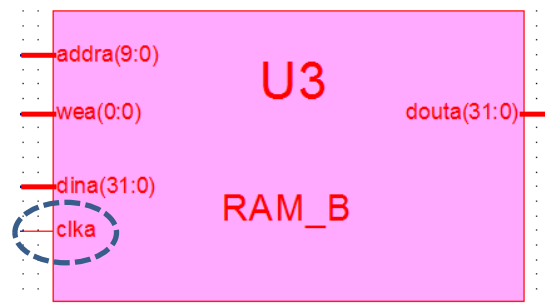
□ RAM_B

用Distributed Memory Generator没有clk信号
请编辑删除clka引脚。SP3平台用不用

- 将Lab3的ROM和RAM合并
 - 数据代码存储共享
- FPGA内部存储器
 - Block Memory Generator或Distributed Memory Generator
- 容量与Lab3的RAM_B相同
 - $1024 \times 32\text{bit}$
- 核模块符号文档: RAM_B.sym
 - 自动生成符号不规则, 需要修整

□ 本实验需要重新生成IP固核

- RAM初始化文档: **mem.coe**
 - 代码与数据合并在一个存储器中
- 核调用模块RAM_B.xco
 - 生成后自动调用关联, 不需要空文档



RAM_B调用方式：与OExp03相同



□ ROM调用接口信号

```
RAM_B    U3 ( .clka(clk_m),           //存储器时钟，与CPU反向
               .addra(ram_addr[9:0]), // RAM地址指针,来自MIO_BUS
               .douta(ram_data_out)   // RAM输出,代码或数据
               );
```

红色与单周期不同均通过MIO_BUS

□ 图形输入调用

- RAM_B.sym

□ 固核调用不需要空模块文档

U3-存储器初始化数据文档: mem.coe

代码与数据共存



```
memory_initialization_radix=16;
```

```
memory_initialization_vector=
```

```
3c03f000, 2014003f, 3c088000, 00632020, 20020001, 00000827, 00205020, 20070003, 00e73827, 20067fff,
00008820, 200502ab, ac650000, 20120002, ac600004, 8c650000, 00a52820, 00a52820, ac650000, ac660004,
3c0dffff, 8c650000, 00a52820, 00a52820, ac650000, 8c650000, 00a52824, 21ad0001, 11680015, 8c650000,
20120018, 00b25824, 11600005, 1172000a, 20120008, 1172000b, ac890000, 08000015, 11410001, 0800002a,
00005027, 014a5020, ac8a0000, 08000015, 8e2902a0, ac890000, 08000015, 8e290260, ac890000, 08000015,
3c0dffff, 014a5020, 01425025, 22310004, 02348824, 21290001, 11210001, 0800003b, 21290005, 8c650000,
00a55820, 016b5820, ac6b0000, ac660004, 8c650000, 00a5824, 1168ffff, 0800001d, 00000000, 00000000,
00000000, 00000000, 00000000, 00000000, 00000000, 00000000, 00000000, 00000000, 00000000, 00000000,
.....
.....
```

代码区: 地址从00000000开始

```
f0000000, 000002AB, 80000000, 0000003F, 00000001, FFFF0000, 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;
```

数据区: 地址起始需要约定



U4-总线接口模块: MIO_BUS

□ MIO_BUS

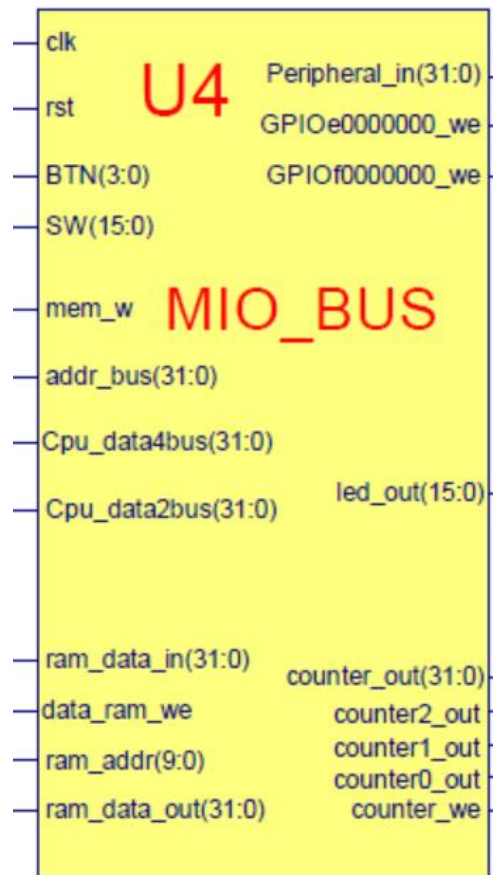
- CPU与外部数据交换接口模块
- 本课程实验将数据交换电路合并成一个模块
 - 非常简单, 但非标准, 扩展不方便
 - 后继课程采用标准总线
 - Wishbone总线

□ 基本功能

- 数据存储、Seg7、SW、BTN和LED等接口

□ 本实验用IP 软核- U4

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



注: 如果增加RAM容量需修改地址译码

IO总线接口空模块-MIO_BUS.v



```
module MIO_BUS( input wire clk, input wire rst,
                input wire [3:0] BTN, input wire [15:0]SW,
                input wire mem_w,
                input wire [31:0] Cpu_data2bus,           //data from CPU
                input wire [31:0] addr_bus,               //addr from CPU
                input wire [31:0] ram_data_out,
                input wire [15:0] led_out,
                input wire [31:0] counter_out,
                input wire counter0_out,
                input wire counter1_out,
                input wire counter2_out,

                output wire [31:0] Cpu_data4bus,           //write to CPU
                output wire [31:0] ram_data_in,            //from CPU write to Memory
                output wire [9: 0] ram_addr,               //Memory Address signals
                output wire data_ram_we,
                output wire GPIOf0000000_w,               // GPIOfffff00_we
                output wire GPIOe0000000_we,              // GPIOfffffe00_we
                output wire counter_we,                   //计数器
                output wire [31:0] Peripheral_in           //送外部设备总线
            );

endmodule
```

MIO_BUS模块调用接口信号关系



MIO_BUS

```
U4( clk_100HzM,  
    rst],  
    BTN [3:0] ,  
    SW [15:0],  
    mem_w,  
    Cpu_data2bus [31:0] ,  
    addr_bus [31:0] ,  
    ram_data_out [31:0] ,  
    led_out [7:0],  
    counter_out [31:0] ,  
    counter0_out,  
    counter1_out,  
    counter2_out,  
  
    Cpu_data4bus [31:0] ,  
    ram_data_in [31:0] ,  
    ram_addr [9: 0] ,  
    data_ram_we,  
    GPIOf0000000_w,  
    GPIOe0000000_we,  
    counter_we,  
    Peripheral_in [31:0]  
);
```

```
//主板时钟  
//复位，按钮BTN3  
//4位原始按钮输入  
//8位原始开关输入  
//存储器读写操作，来自CPU  
//CPU输出数据总线  
//地址总线，来自CPU  
//来自RAM数据输出  
//来自LED设备输出  
//当前通道计数输出，来自计数器外设  
//通道0计数结束输出，来自计数器外设  
//通道1计数结束输出，来自计数器外设  
//通道2计数结束输出，来自计数器外设  
  
//CPU写入数据总线，连接到 CPU  
//RAM 写入数据总线，连接到RAM  
//RAM访问地址，连接到RAM  
//RAM读写控制， 连接到RAM  
// 设备一LED写信号  
// 设备二7段写信号，连接到U5  
//记数器写信号，连接到U10  
//外部设备写数据总线，连接所有写设备
```

endmodule



GPIO设备与接口模块

—非常简单与Exp03相同

—除Seg7设备外, 接口与设备合二为一

U7-外部设备模块：GPIO接口及设备一

GPIO



□ GPIO输出设备一

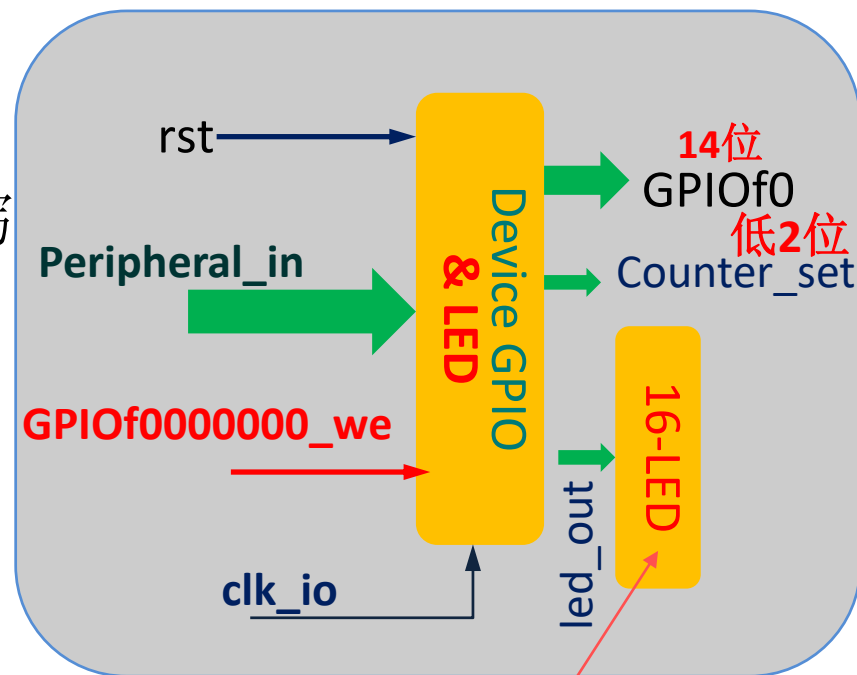
- 地址范围=f0000000 - ffffffff0 (ffffff00-fffffff0)
- 读写控制信号：**GPIOf0000000_we(GPIOffffff00_we)**
- {GPIOf0[21:0],**LED**,counter_set}

□ 基本功能

- LEDs设备和计数器控制器读写
- 可回读，检测状态
- 逻辑实验LED模块改造

□ 本实验用IP 软核- **U7**

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



注意：左移2位后低16位

通用接口与设备—IP核调用空模块

-GPIO.v



module

GPIO(input clk,

input rst,

input EN,

input [31:0] P_Data,

input Start

output [1:0] counter_set,

output [15:0] LED_out,

output [13:0] GPIOf0

output ledclk,

output ledsout,

output LEDED,

output ledclrn

);

//io_clk, 与CPU反向

//来自U4

//来自U4

//串行输出启动

//来自U7, 后继用

//输出到LED, 回读到U4

//备用

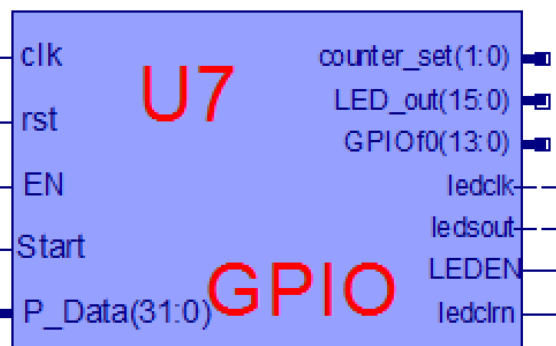
//串行时钟

//串行LEDE值

//LED使能

//LED清零

endmodule





□ 7段码显示输出设备模块

- 需要通过接口模块**Multi_8CH32**与CPU连接
- 地址范围=E0000000 - EFFFFFFF (FFFFFFE00-FFFFFFE0F)

□ 基本功能(参考Exp02)

- 4位7段码显示设备
- 模拟文本，显示8位16进制数：SW[1:0]=x1
 - SW[1:0]=01，显示低4位；
 - SW[1:0]=11，显示高4位
- 模拟图形显示，4位7段用于32个点阵显示，SW[1:0]=x0
- 逻辑实验7段显示模块改造

Sword平台不需要

□ 本实验用IP 软核或OExp02设计模块- **U6**

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



□ 七段码显示输出设备模块

- 需要通过接口模块**Multi_8CH32**与CPU连接
- 地址范围=E0000000 - EFFFFFFF (FFFFFFE00-FFFFFFE0F)

□ 基本功能(参考OExp02)

- 4位7段码显示设备
- 模拟文本，显示8位16进制数：SW[1:0]=x1
 - SW[1:0]=01，显示低4位；
 - SW[1:0]=11，显示高4位
- 模拟图形显示，4位7段用于32个点阵显示，SW[1:0]=x0
- 逻辑实验7段显示模块改造

□ 本实验用IP 软核或Exp02设计的模块- **U5**

- 核调用模块SSeg7_Dev.ngc
- 核接口信号模块(空文档)：SSeg7_Dev.v
- 核模块符号文档：SSeg7_Dev.sym

通用设备二IP核调用空模块



Display.v

```
module Display(input clk,  
               input rst,  
               input Start,  
               input Text  
               input flash,  
               input[31:0]Hexs,  
               input[7:0]point,  
               input[7:0]LES,  
               output segclk,  
               output segsout,  
               output SEGEN,  
               output segclrn  
               );
```

//时钟
//复位
//串行扫描启动
//文本(16进制)/图型(点阵)切换
//七段码闪烁频率
//32位待显示输入数据，接U5
//七段码小数点：8个
//七段码使能：=1时闪烁
//串行移位时钟
//七段显示数据(串行输出)
//七段码显示刷新使能
//七段码显示清零

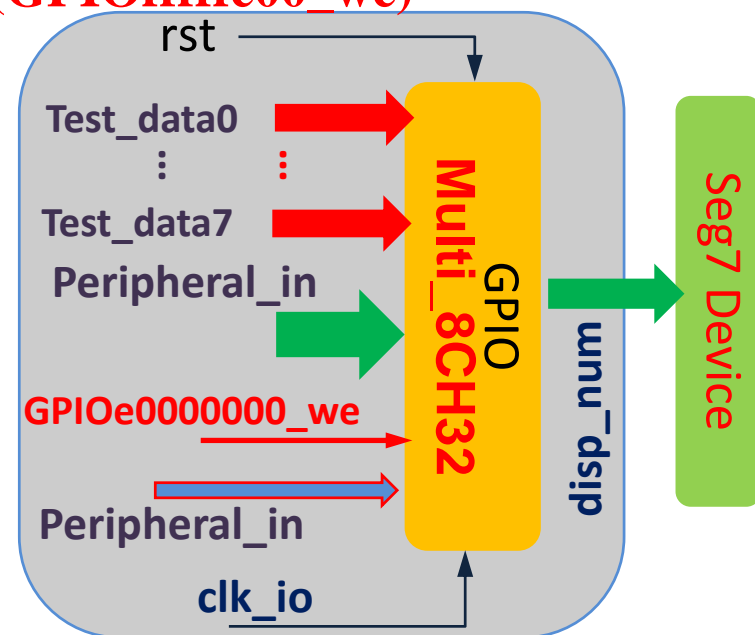
```
endmodule
```

□ GPIO输出设备二接口模块

- 地址范围=E0000000 - EFFFFFFF (FFFFFFE00-FFFFFFE00)
- 读写控制信号: **GPIOe0000000_we(GPIOfffffe00_we)**

□ 基本功能(参考Exp01)

- 七段码输出设备接口模块
- 逻辑实验显示通道模块改造
- 通道0作为显示设备接口
 - **GPIOe0000000_we=1**
 - **CLK上升沿**
- 通道1-7作为调试测试信号显示



□ 本实验用IP 软核或EXp01设计的模块- **U6**

- 核调用模块Multi_8CH32.ngc
- 核接口信号模块(空文档): **Multi_8CH32_IO.v**
- 核模块符号文档: Multi_8CH32.sym

通用设备二接口调用空模块

-Multi_8CH32_IO.v



module

Multi_8CH32 (input clk,

//io_clk, 同步CPU

input rst,

input EN,

//=1, 通道0显示

input[63:0]point_in, //针对8个显示通道各8个小数点

input[63:0]LE_in, //针对8个通道各8位闪烁控制

input [2:0] Test, //通道选择SW[7:5]

input [31:0] Data0, //通道0

input [31:0] data0, //通道1

input [31:0] data1, //通道2

input [31:0] data2, //通道3

input [31:0] data3, //通道4

input [31:0] data4, //通道5

input [31:0] data5, //通道6

input [31:0] data6, //通道7

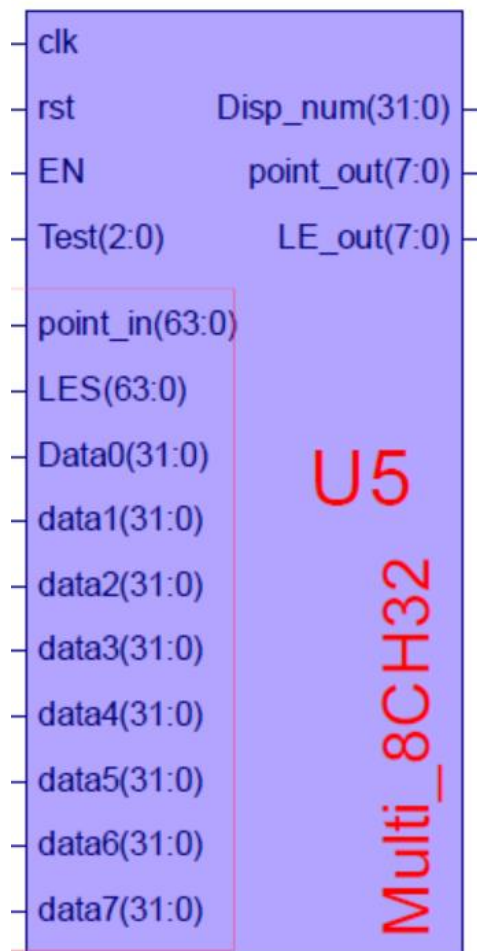
output reg[7:0] point_out, //小数点输出

output reg[7:0] LE_out, //闪烁控制输出

output [31:0] Disp_num //接入7段显示器

);

endmodule



Multi_8CH32调用信号关系



```
Multi_8CH32    U5( .clk(clk_io), .rst(rst),  
                  .EN(GPIOe0000000_we),           //来自U4  
                  .point_in(point_in),             //外部输入  
                  .blink_in(LE_in),                 //外部输入  
                  .Test(SW_OK[7:5]),                //来自U9  
                  .Data0(Peripheral_in),            //来自U4  
                  .data1({2'b00,PC_out[31:2]}),     //来自U1  
                  .data2(counter_out),              //来自U10  
                  .data3(Inst),                     //Inst, 来自CPU  
                  .data4(addr_bus),                  //来自CPU  
                  .data5(Cpu_data2bus),             //来自CPU  
                  .data6(Cpu_data4bus),             //来自CPU  
                  .data7(PC_out),                   //来自CPU;  
  
                  .point_out(point_out),            //输出到U6  
                  .LE_out(blink_out),               //输出到U6  
                  .disp_num(disp_num)              //输出到U6  
                  );
```

外部设备模块：GPIO接口设备三、四

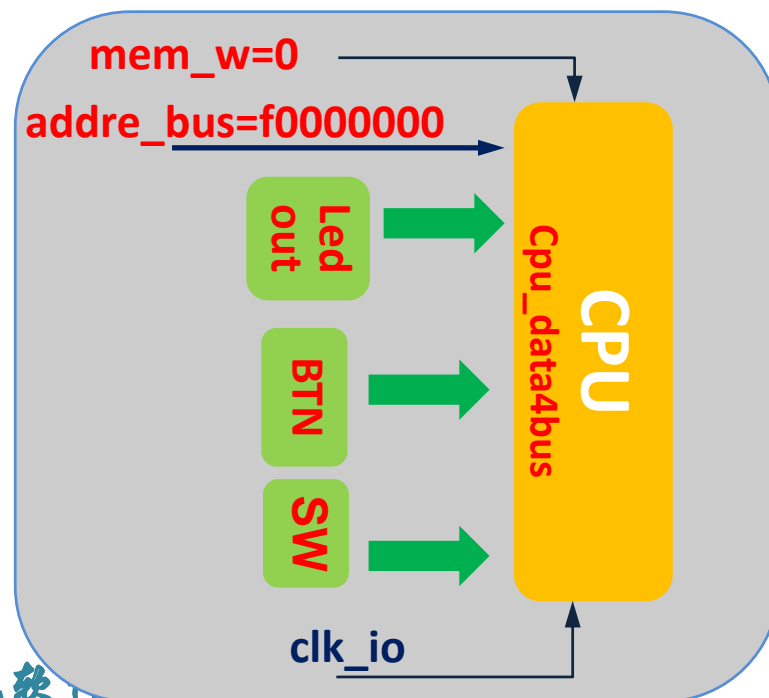
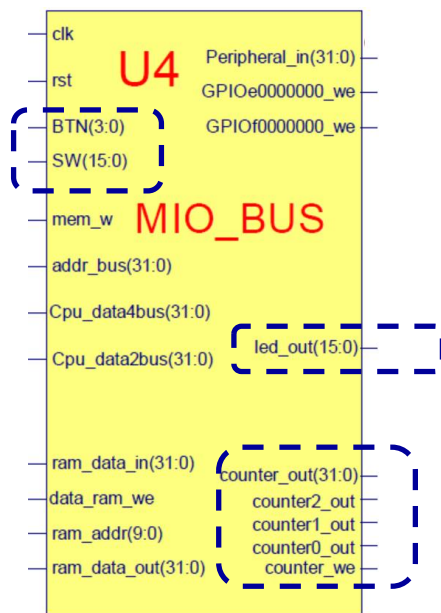
GPIO_SW_BTN



15位Switch和4位Button输入设备

- 地址范围= f0000000-ffffffff0, A[2]=0
- 这二个设备非常简单直接包含在U4, MIO_BUS模块中
- 与CPU数据线关系（当**addre_bus=f0000000**时）

Cpu_data4bus = {counter0_out, counter1_out, counter2_out, led_out[9:0], **BTN, SW**}; 注意：低16位



U10-外部设备五：通用计数器模块

Counter.v



□ 通用计数器设备，双向

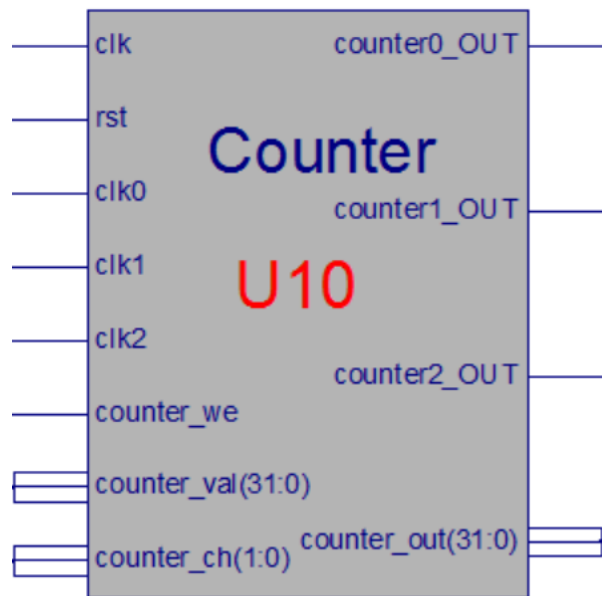
- 地址范围=F0000004 – FFFFFFFF4 (FFFFFFF04-FFFFFFF4)
- 读写控制信号：counter_we

□ 基本功能

- 三通道独立计数器，可用于程序定时。
- 输出用于计数通道设置或计数值初始化
 - counter_set=00、01、10对应计数通道0、1、2
 - counter_set=11对应计数通道工作设置
- 计数器部分兼容8253

□ 本实验用IP 软核- U10

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



通用计数器IP核调用空模块

-Counter_x.v



```
module Counter(input clk,                                     //io_clk
               input rst,
               input clk0,                                     //Div[7], 来自U8
               input clk1,                                     //D], 来自U8
               input clk2,                                     //Div[10], 来自U8
               input counter_we,                               //计数器写控制, 来自U4
               input [31:0] counter_val,                       //计数器输入数据, 来自U4
               input [1:0] counter_ch,                         //计数器通道控制, 来自U7

               output counter0_OUT,                             //输出到U4
               output counter1_OUT,                             //输出到U4
               output counter2_OUT,                             //输出到U4
               output [31:0] counter_out                       //输出到U4
               );

endmodule
```



SOC系统实现辅助模块

U8: 通用分频模块

U9: 开关去抖动模块



U8-通用分频模块: `clk_div`

□ 计数分频模块

- 用于要求不高的各类计数和分频
 - CPU、IO和存储器等
- 对延时和驱动有要求的需要BUFG缓冲
- 对于时序要求高的需要用DCM实现

□ 基本功能

- 32位计数分频输出: `Div`
- CPU时钟输出: `Clk_CPU`
- 逻辑实验通用计数模块改造

□ 本实验自己设计核(逻辑电路输出)- `U8`

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

通用分频IP核调用空模块

-clk_div.v



```
module clk_div(input clk,  
               input rst,  
               input SW2,  
               output reg[31:0]clkdiv,  
               output Clk_CPU  
               );
```

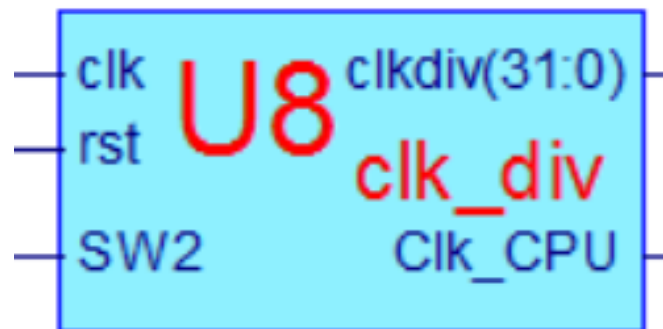
// Clock divider-时钟分频器

```
    always @ (posedge clk or posedge rst) begin  
        if (rst) clkdiv <= 0;  
        else clkdiv <= clkdiv + 1'b1;
```

```
    end
```

```
    assign Clk_CPU=(SW2)? clkdiv[24] : clkdiv[1];
```

```
endmodule
```





U9-开关去抖动模块：SAnti_jitter

- 开关机械抖动消除模块
 - 用于消除开关和按钮输入信号的机械抖动
 - CPU、IO和存储器等
- 基本功能
 - 输入机械开关量
 - 输出滤除机械抖动的逻辑值
 - 电平输出：button_out、SW_OK
 - 脉冲输出：button_pluse
 - 逻辑实验模块
- 本实验可自己设计或用IP 软核- U9
 - 核调用模块SAnti_jitter.ngc
 - 核接口信号模块(空文档)：SAnti_jitter.v
 - 核模块符号文档：SAnti_jitter.sym

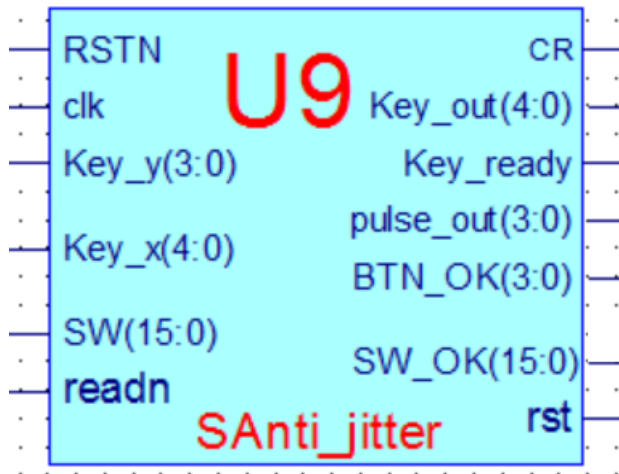
开关去抖动IP核调用空模块

-Anti_jitter.v



```
module      SAnti_jitter(input  clk,                //主板时钟
                        input  RSTN
                        input  readn                //阵列式键盘读
                        input  [3:0]Key_y,         //阵列式键盘列输入
                        output reg[4:0] Key_x,      //阵列式键盘行输出
                        output reg[4:0] Key_out,    //阵列式键盘扫描码
                        output reg  Key_ready,      //阵列式键盘有效
                        input  [15:0] SW,          //开关输入
                        output reg [3:0] ] BTN_OK,  //列按键输出
                        output reg [3:0] pulse,     //列按键脉冲输出
                        output reg [15:0] SW_OK,    //开关输出
                        output reg  CR,            //RSTN短按输出
                        output reg  rst           //复位, RSTN长按输出
);

endmodule
```



Course Outline





设计工程：OExp09-MCPU2SOC

◎ 建立CPU调试、测试和应用环境

⌚ 顶层用HDL实现，调用IP核模块

○ 模块名：Top_OExp09_IP2MCPU.sch

◎ SOC集成技术实现测试系统构架

⌚ 复用实验三模块，除SCPU外

○ CPU (第三方IP核): U1

○ RAM (ISE构建IP核): U3

○ 总线(第三方IP核): U4

○ 八数据通路模块(实验一Multi_8CH32): U5

○ 七段显示模块(实验二Display IP): U6

○ LED显示模块(实验二GPIO模块): U7

○ 通用分频模块(clk_div): U8

○ 开关去抖模块(IP核): U9

○ 数据输入模块(IP核): M4(目前没有使用)

◎ 分解CPU为二个IP核

⌚ SOC调试通过后用二个IP核构建MCPU

建立多周期SOC测试应用环境

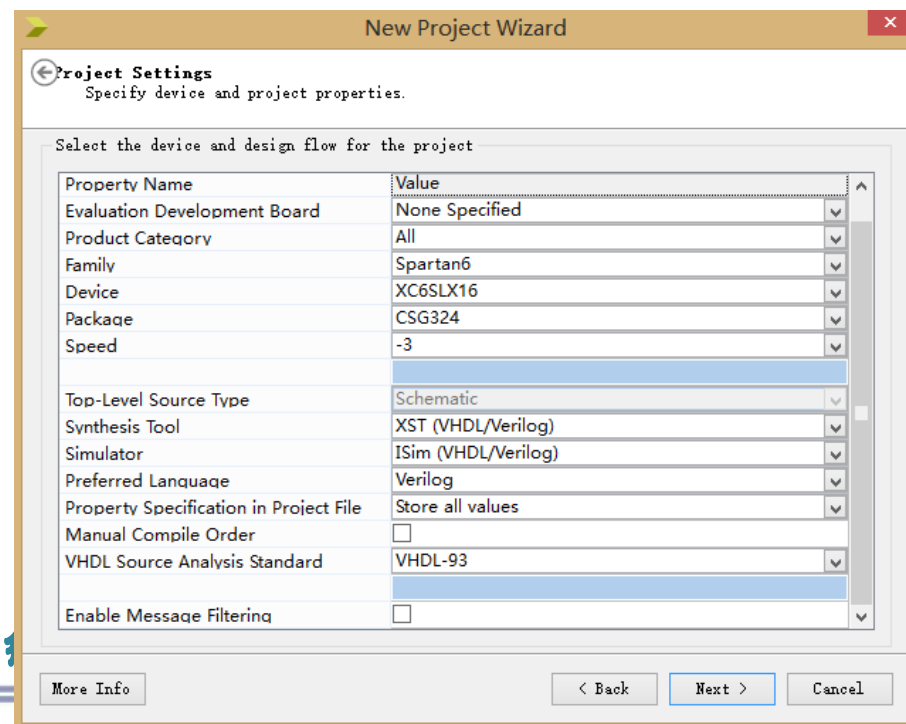
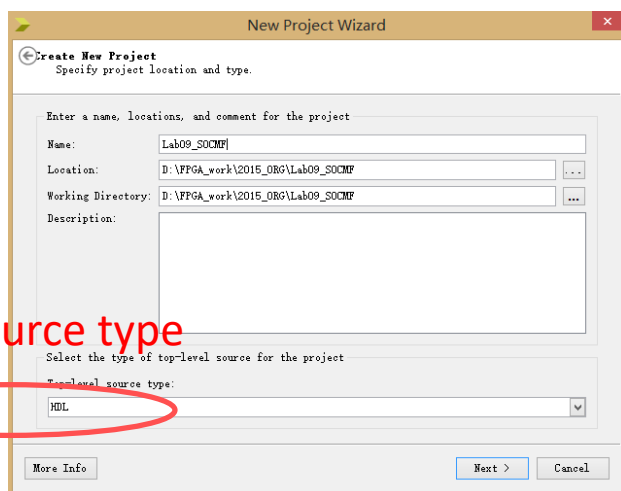
--新建工程OExp09-MCPU2SOC

建立多周期SOC测试应用工程

□ 用ISE新建SOC测试应用工程

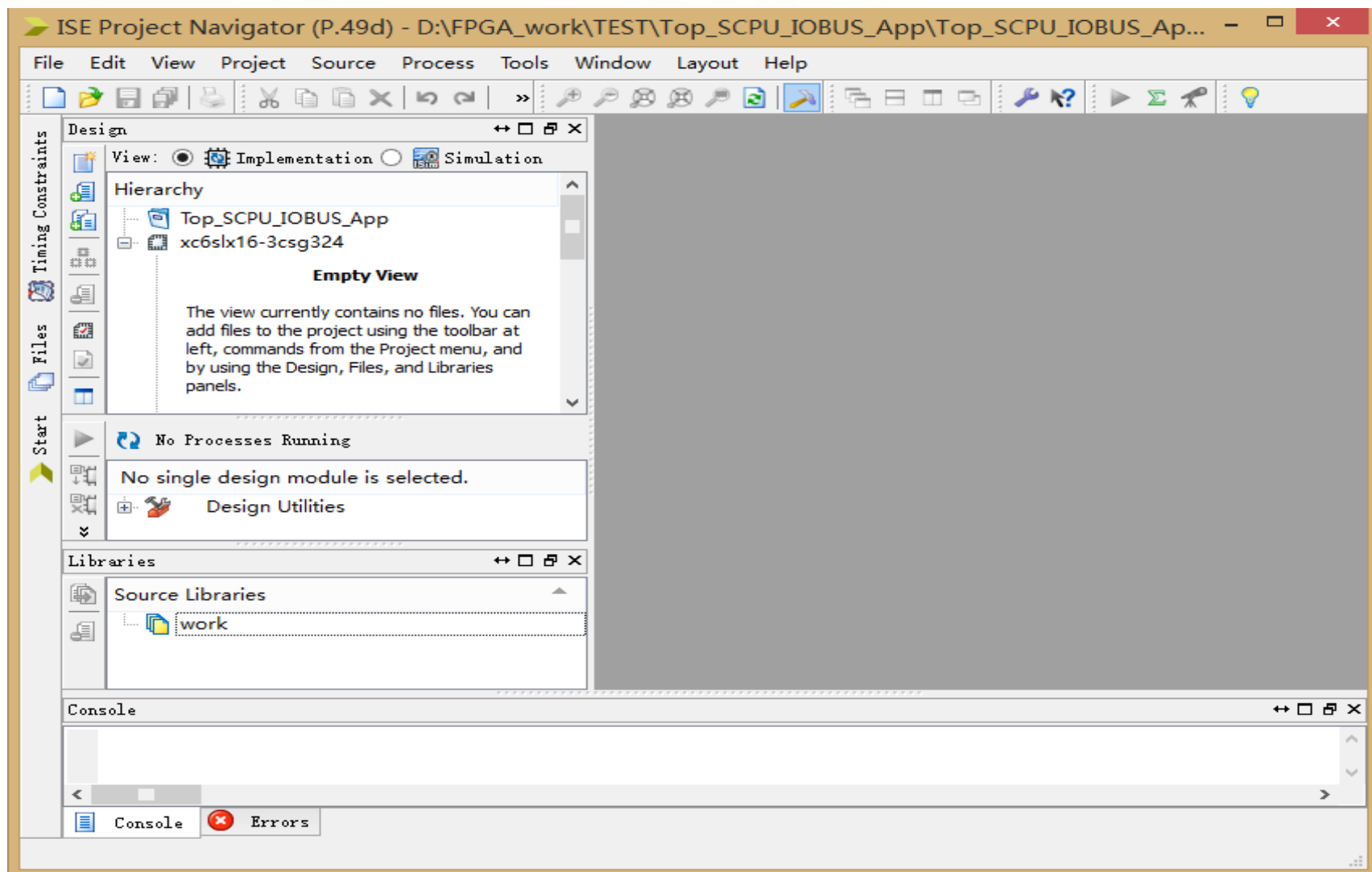
- 双击桌面上“Xilinx ISE”图标，启动ISE软件(也可从开始菜单启动)
- 选择File New Project选项，在弹出的对话框中输入工程名称并指定工程路径。参考工程名：**ORG-Exp09**或**OExp09-MCPU2SOC**
- 点击Next按钮进入下一页，选择所使用的芯片及综合、仿真工具。
- 再点击Next按钮进入下一页，这里显示了新建工程的信息，确认无误后，点击Finish就可以建立一个完整的工程了

□ 多周期CPU设计共享此工程





SOC测试应用框架工程模板





本实验采用结构化HDL描述不需要拷贝模块符号图

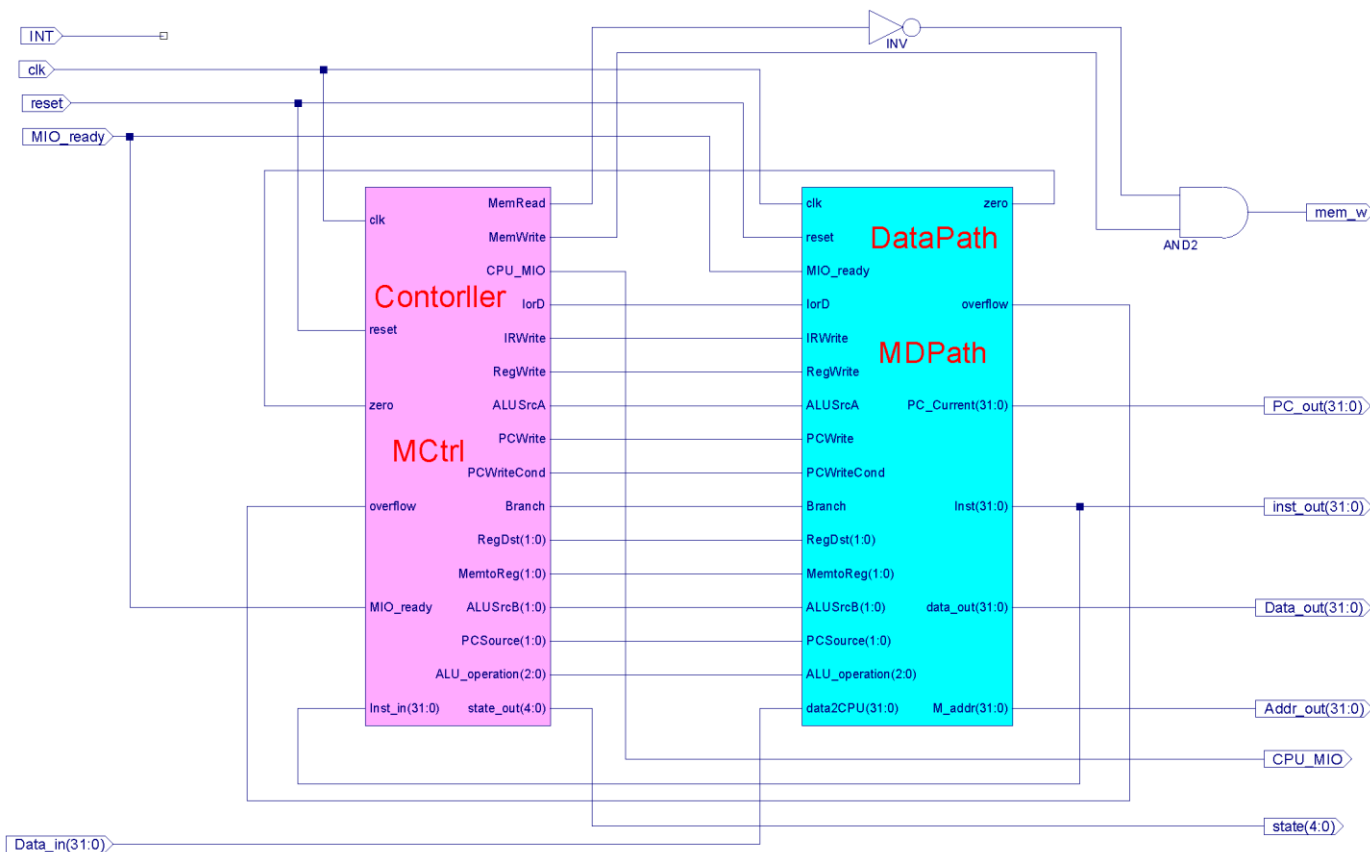
**拷贝U1、U3~U10、多周期控制器、数据通路软核
(.ngc)文档到当前工程目录**

——已经自己设计完成的除外
——通用分频模块建议采用自己设计的

设计多周期CPU核调用模块

□ 用HDL描述IP核调用实现CPU

- 建议模块名：Multi_CPU.v或MCPMU.v





核集成CPU描述结构参考

```
module      MCPU(input clk,                                //muliti_CPU
                  input reset,
                  input MIO_ready,
                  output[31:0] PC_out,                      //TEST
                  output[31:0] inst_out,                   //TEST
                  output mem_w,
                  output[31:0] Addr_out,
                  output[31:0] Data_out,
                  input [31:0] Data_in,
                  output CPU_MIO,
                  input INT,
                  output[4:0]state                          //Test
                );
```

..... 变量申明：所有模块端口申明为**wire**时要特别注意信号宽度

..... 初值、模块连接过渡信号，信号转换、总线转换等

..... 模块调用：2个模块调用

endmodule



重要信号及模块调用结构

```
.....  
assign mem_w=MemWrite&&(~MemRead);  
assign PC_out=PC_Current;
```

```
MCtrl x_ctrl(.clk(clk),  
              .reset(reset),  
              .Inst_in(inst_out),  
              .....  
              .PCWrite(PCWrite),  
              .PCWriteCond(PCWriteCond),  
              .Branch(Branch)  
              );
```

```
MDPath x_datapath(.clk(clk),  
                  .reset(reset),  
                  .MIO_ready(MIO_ready),  
                  .....  
                  .M_addr(Addr_out),  
                  .zero(zero),  
                  .overflow(overflow)  
                  );
```

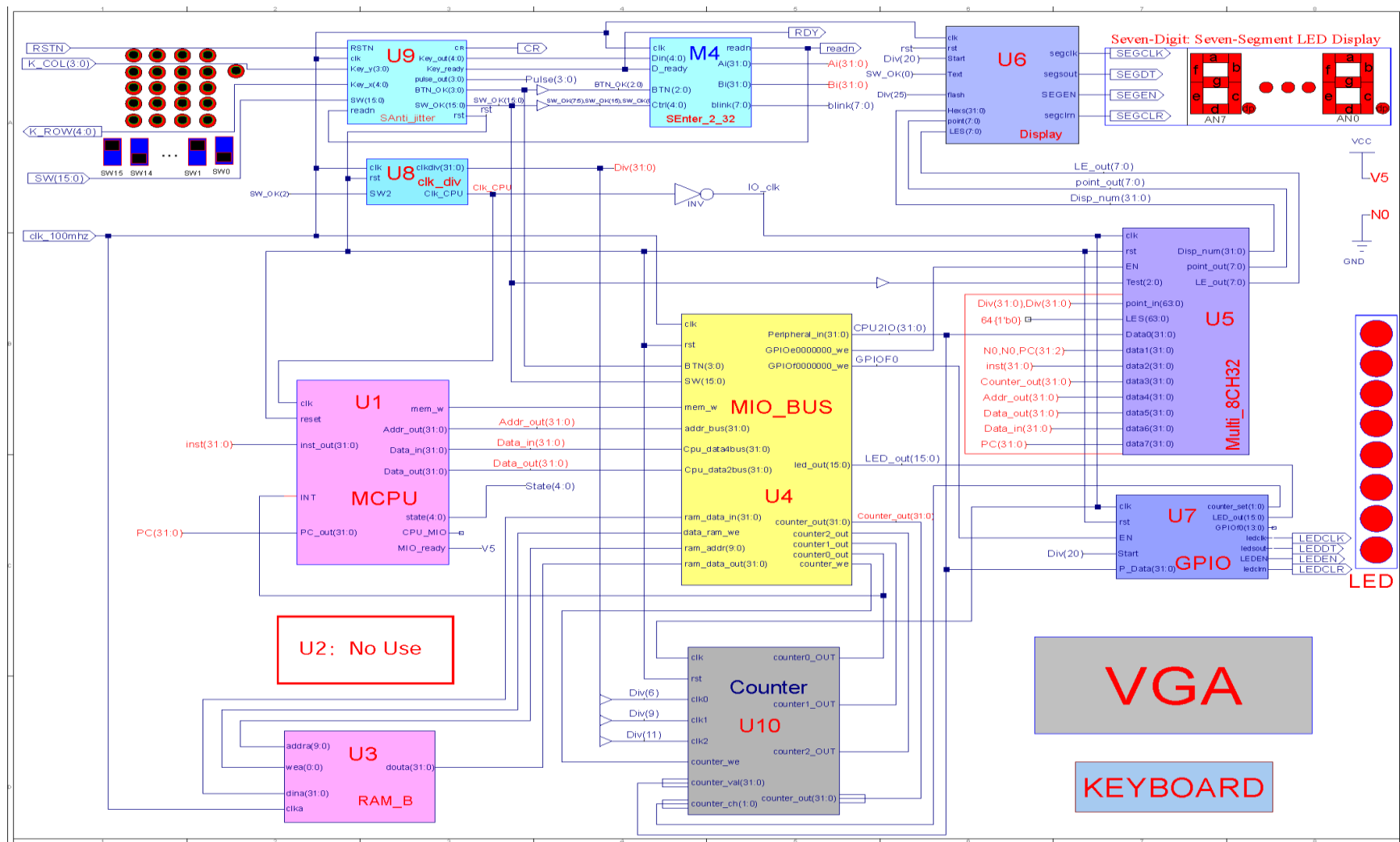


SoC测试构架顶层结构描述

--用HDL实现，调用IP核模块

根据下列逻辑原理图设计结构描述代码

很简单：就是端口定义和二级模块调用





建立结构描述输入模板（顶层模块）

□ 建立顶层模块

- 在Project弹出的菜单中选择New Source命令
- 选择：Verilog Module
- 缺省目录是工程目录Exp09_MCPU2SOC
- 建议修改为Exp09_MCPU2SOC\Code

□ 注意：为了方便管理，将所有代码存放在独立目录中！

- 同时注意同名.sch与.v文件的冲突

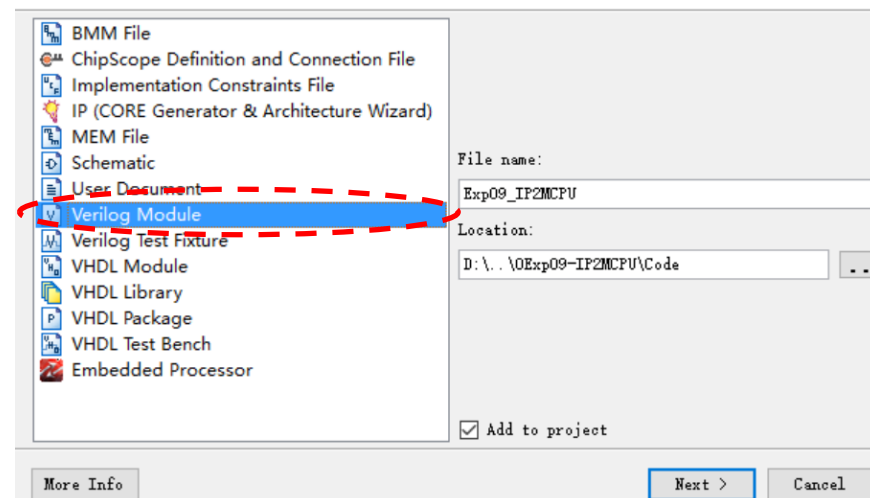
New Source Wizard

←Select Source Type

Select source type, file name and its location.

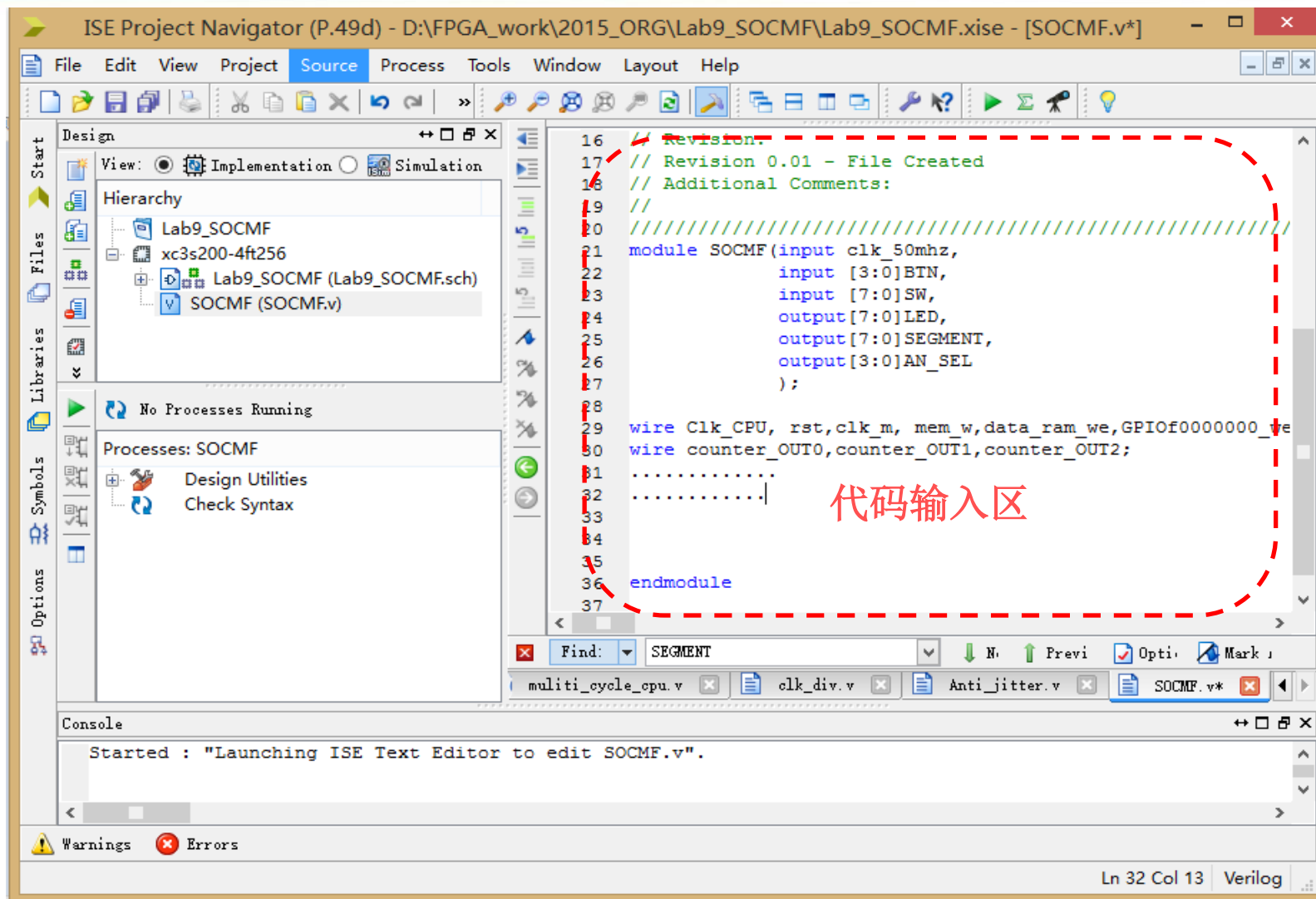
□ 根据原理图设计描述代码

- 后面以SOCMF.v为例说明





硬件描述代码输入窗口与环境





完成输入后第二层模块层次关系

ISE Project Navigator (P.49d) - D:\FPGA_work\2015_ORG\Lab9_SOCMF\Lab9_SOCMF.xise - [SOCMF.v]

File Edit View Project Source Process Tools Window Layout Help

Design View: Implementation Simulation

Hierarchy

- Lab9_SOCMF
 - xc3s200-4ft256
 - SOCMF (SOCMF.v)
 - U6 - seven_seg_dev (seven_seg_dev.ngc)
 - U6 - seven_seg_dev (7seg_Dev.v)
 - U9 - Anti_jitter (Anti_jitter.v)
 - U8 - clk_div (clk_div.v)
 - U1 - Muliti_CPU (Muliti_CPU.ngc) CPU核**
 - U1 - Muliti_CPU (muliti_cycle_cpu.v)
 - U3 - RAM_B (RAM_B.xco)
 - U4 - MIO_BUS (MIO_BUS.ngc)
 - U4 - MIO_BUS (MIO_BUS.v)
 - U7 - led_Dev_IO (led_Dev_IO.ngc)
 - U7 - led_Dev_IO (led_Dev_IO.v)
 - U5 - seven_seg_Dev_IO (seven_seg_Dev_IO.ngc)
 - U5 - seven_seg_Dev_IO (7seg_Dev_IO.v)
 - U10 - Counter_x (Counter_x.ngc)
 - U10 - Counter_x (Counter_3channel.v)
 - SOC_SP3.ucf

No Processes Running

Processes: SOCMF

- Design Summary/Reports
- Design Utilities
- User Constraints

```
106 //+++++muliti_cycle_cpu+++++
107 Muliti_CPU    U1(.clk(Clk_CPU),
108               .reset(rst),
109               .MIO_ready(MIO_ready), //MIO_ready
110
111               // Internal signals:
112               .pc_out(pc),           //Test
113               .Inst(Inst),           //Test
114               .mem_w(mem_w),
115               .Addr_out(addr_bus),
116               .data_out(Cpu_data2bus),
117               .data_in(Cpu_data4bus),
118               .CPU_MIO(CPU_MIO),
119
120               .state(state)           //Test
121               );
122
123 RAM_B          U3(.clka(clk_m),
124               .wea(data_ram_we),
125               .addra(ram_addr),       // Adresse_Bus [9 : 0]
126               .dina(ram_data_in),
127               .douta(ram_data_out)    //Data_Bus [31 : 0]
128               );
129
130 //+++++
131
132 MIO_BUS    U4(.clk(clk_50mhz),
133               .rst(rst),
134               .BTN(bottom_out),
```

Find: SEGMENT Next Previous Options Mark All

F. sch | muliti_cycle_cpu.v | clk_div.v | Anti_jitter.v | SOCMF.v

Ln 123 Col 5 Verilog



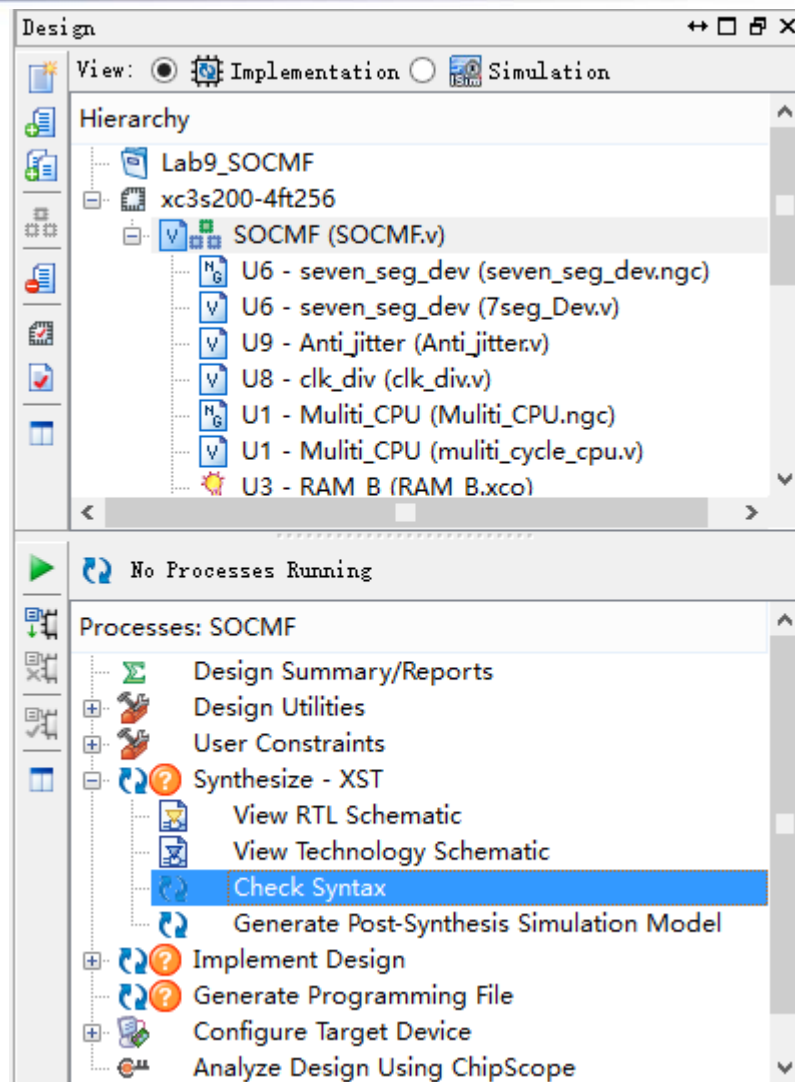
顶层语法检查

□ 模块信号连接检测

- 激活Design窗口
- 点中顶层模块: SOCMF.v
- 在进程运行窗口:

选择Synthesize→Check Syntax

- 综合器检查代码语法
 - 不会检查电路逻辑功能
 - 仅检查代码语法
 - 特别注意总线连接
 - 错位
 - 别名



模块调用与关联

□ 模块调用方法

模块名 调用编号(端口信号列表);

■ 端口信号对应:

- 与模块内端口信号顺序一一对应
- 用括号引用: .模块内信号(输入信号)

如: .clk(clk_100mHz)

□ 顶层调用模块关联

- 顶层窗口放置模块Symbol后会直接调用对应模块
- 建立核端口模块与软核模块关联
 - 只有端口信号的模块, 没有逻辑代码的空文档.v
 - 综合器会根据端口模块连接信号
- 点击Add Source 关联对应的空模块

顶层(SOC测试应用系统)参考描述



```
module      SOCME (input clk_50mhz,  
                  input [3:0]BTN,  
                  input [7:0]SW,  
                  output[7:0]LED,  
                  output[7:0]SEGMENT,  
                  output[3:0]AN_SEL  
                  );
```

```
Wire Clk_CPU, rst, clk_m, mem_w, data_ram_we, GPIOf00000000_we, GPIOe0000000_we, counter_we;  
wire counter_OUT0, counter_OUT1, counter_OUT2;  
wire [1:0]Counter_set;  
wire MIO_ready;  
wire CPU_MIO;
```

..... 变量申明：所有模块端口申明为**wire**，特别注意信号宽度

..... 初值、模块连接过渡信号，信号转换、总线转换等

..... 模块调用：9个模块调用

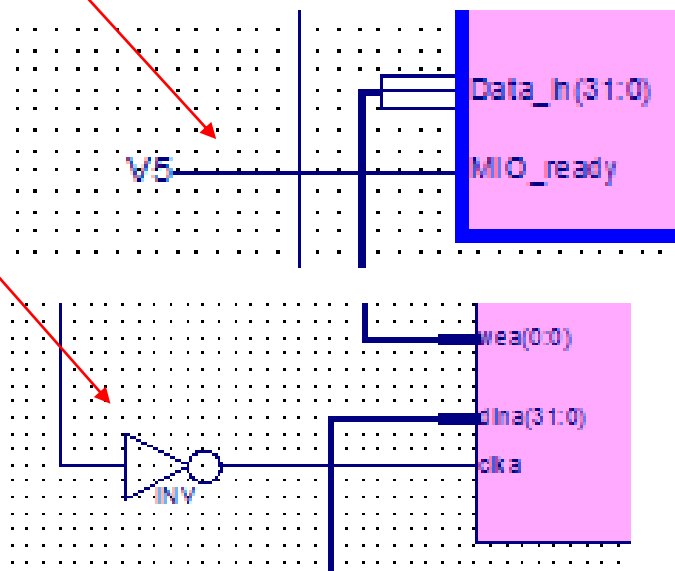
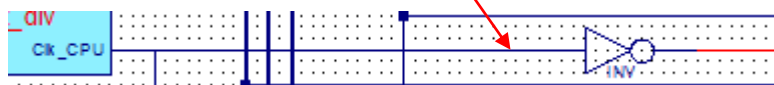
```
endmodule
```

信号传输:



初值、模块连接过渡信号, 信号转换、总线转换等

```
assign MIO_ready=~button_out[1];           //CPU永远不等待, 或=1
assign SW2=SW_OK[2];                       //信号传输
assign LED={led_out[7]|Clk_CPU,led_out[6:0]}; //总线转换
assign clk_m=~clk_50mhz;
.....
assign clk_io=~Clk_CPU;
.....
```



模块调用



```
SAnti_jitter    U9(clk,           //主板时钟
                  RSTN
                  readn
                  Key_y,         //阵列式键盘列输入
                  Key_x,         //阵列式键盘行输出
                  Key_out,       //阵列式键盘扫描码
                  .....

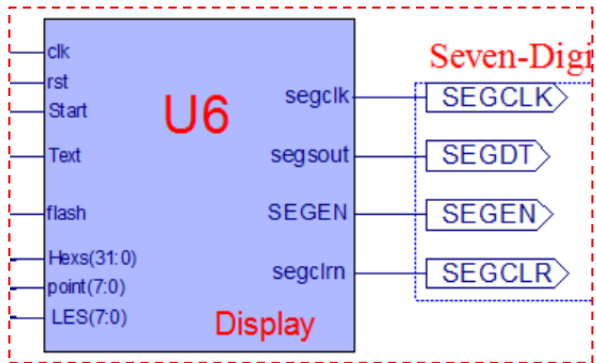
                  rst           //复位， RSTN长按输出
                  );

clk_div         U8(clk_100mhz,    // Clock divider-时钟分频器
                  rst,
                  SW2,
                  clkdiv,
                  Clk_CPU
                  );
```




模块调用

Display



```

U6(.clk(clk_100mhz),
    .rst(rst),
    .....
    .....
    .SEGEN(SEGEN),
    .segclrn(SEGCLR)
);

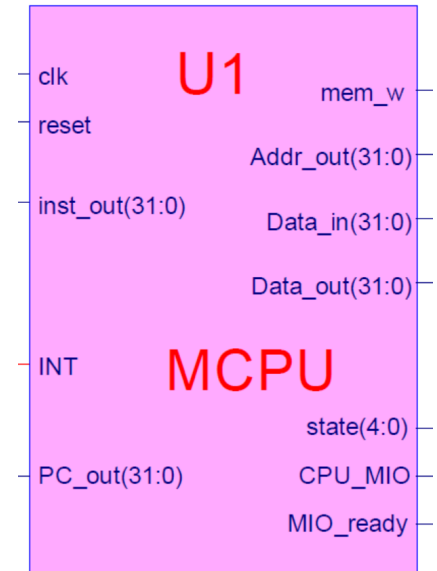
```

MCPU

```

U1(.clk(Clk_CPU),
    .reset(rst),
    .MIO_ready(MIO_ready),
    .....
    .....
    .state(state)
);
//MIO_ready
//Test

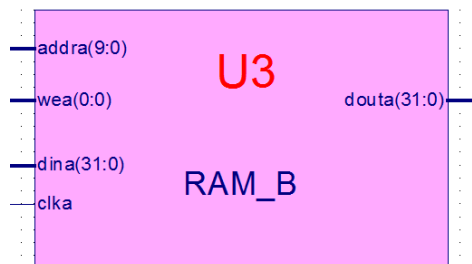
```



模块调用



RAM_B



```
U3(.clka(clk_m),
    .wea(data_ram_we),
    .addra(ram_addr),
    .dina(ram_data_in),
    .douta(ram_data_out)
);
```

// Addre_Bus [9 : 0]

//Data_Bus [31 : 0]

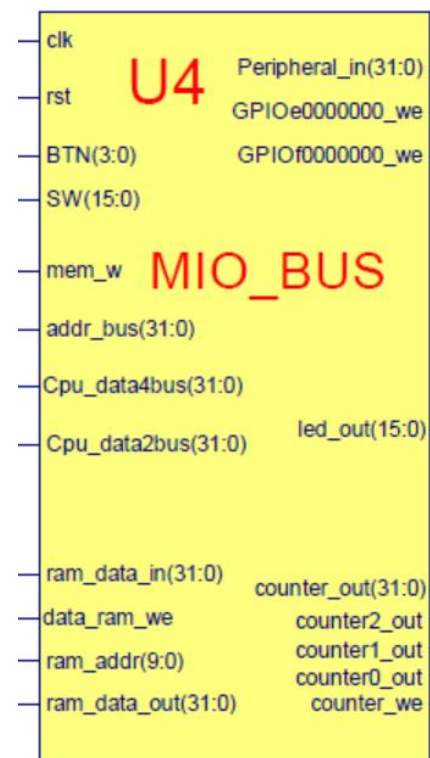
MIO_BUS

```
U4( .clk(clk_50mhz),
```

.....

.....

```
.Peripheral_in(Peripheral_in)
);
```



模块调用



GPIO

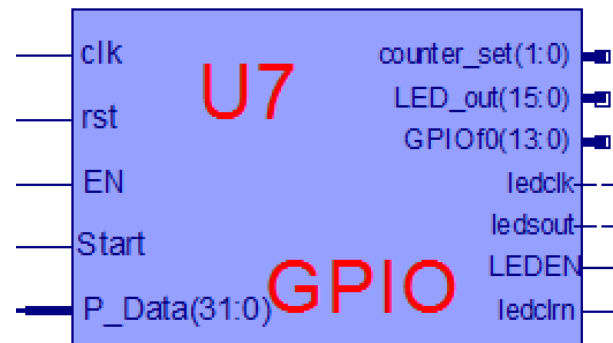
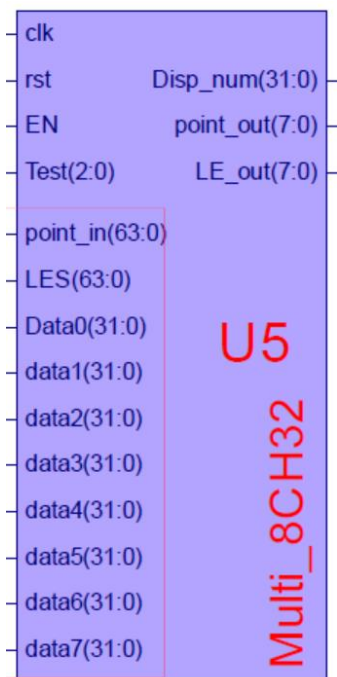
U7(

);

Multi_8CH32

U5(

```
.clk(?),
.rst(rst),
.EN(?),
.Test(SW_OK[7:5]),
.Data0(?),
.data1(?),
.data2(counter_out),
.data3(?),
.data4(?),
.data5(?),
.data6(?),
.data6(?),
.....
);
```



```
//CPU data output
//pc[31:2]
//counter
//Inst
//addr_bus
//Cpu_data2bus;
//Cpu_data4bus;
//pc;
```

模块调用

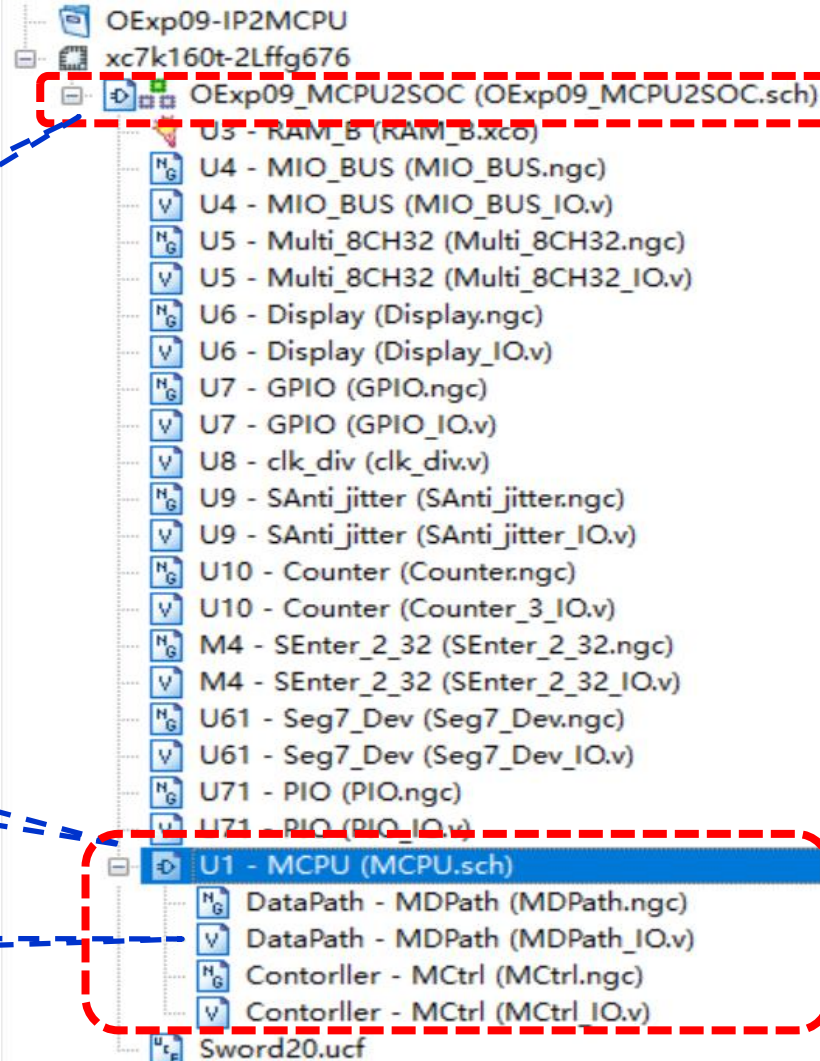
Counter

```
U10(.clk(clk_io),  
    .clk0(clkdiv[9]),  
    .clk1(clkdiv[10]),  
    .clk2(clkdiv[10]),  
    .counter_we(counter_we),  
    .counter_val(Peripheral_in),  
    .counter_ch(Counter_set),  
    .counter0_OUT(counter_OUT0),  
    .counter1_OUT(counter_OUT1),  
    .counter2_OUT(counter_OUT2),  
    .counter_out(counter_out)  
);
```

endmodule

完成后的层次关联

Hierarchy



顶层改用HDL描述

CPU封装层用电路或HDL描述均可

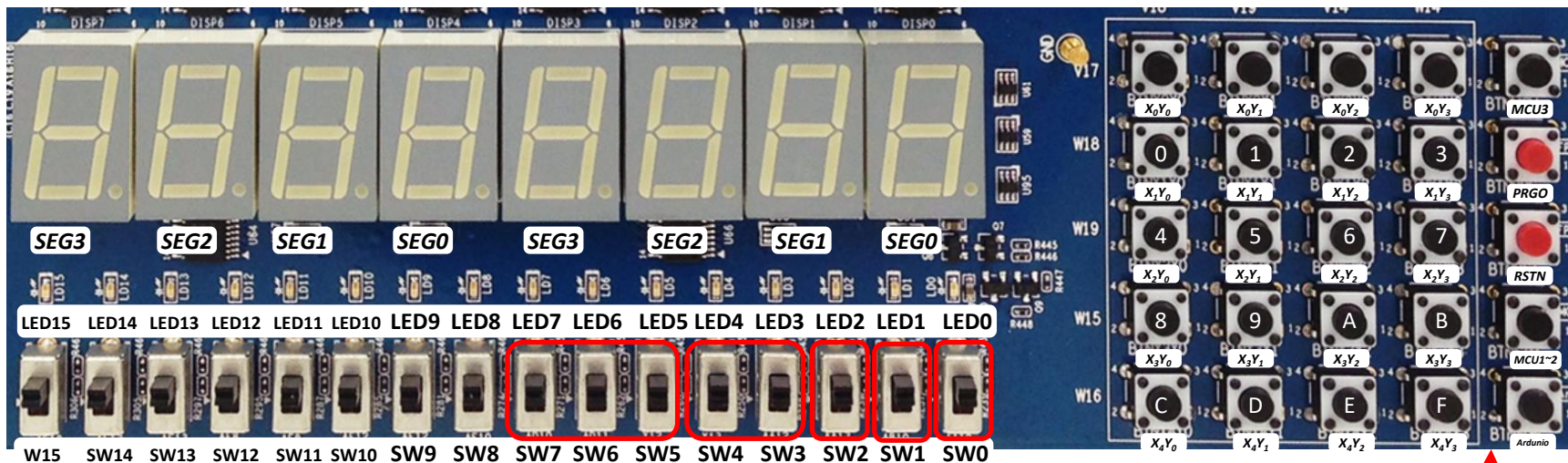
IP核集成的CPU



SoC物理调试验证

- 本实验不需要测试，仅用DEMO程序验证功能
- 过程与实验三、四相同

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

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



下载验证SoC

□ 非IP核仿真

- 对自己设计的模块做时序仿真(单周期时仿真过的略)
- 第三方IP核不做仿真(固核无法做仿真)

□ SOC物理验证

- 下载流文件.bit
- 验证调试SOC功能
 - 功能不正确时排查错误
- 定性观测SOC关键信号
 - 本实验只要求定性观测
 - 测试代码和数据用mem.coe数据*



测试开关设置

■ 图形功能测试

开关	位置	功能
SW[1:0]	X0	七段码图形显示
SW[2]	0	CPU全速时钟
SW[4:3]	00	7段码从上至下亮点循环右移
SW[4:3]	11	7段码矩形从下到大循环显示
SW[7:5]	000	作为外设使用 (E0000000/FFFFFFE0)

■ 文本功能测试

开关	位置	功能
SW[1:0]	01	七段码文本显示 (低16位)
	11	七段码文本显示 (高16位)
SW[2]	0	CPU全速时钟
SW[4:3]	01	7段码显示RAM数字
SW[4:3]	10	7段码显示累加
SW[7:5]	000	作为外设使用 (E0000000/FFFFFFE0)



仅定性观测

□ SOC信号测试

- CPU全速运行
- 测试开关设置

开关	位置	功能
SW[1:0]	01	七段码文本显示（低16位）
SW[1:0]	11	七段码文本显示（高16位）
SW[2]	0	CPU全速时钟
SW[7:5]	010	Counter值输出
SW[7:5]	100	CPU数据存储地址addr_bus（ALU）
SW[7:5]	101	CPU数据输出Cpu_data2bus（寄存器B）
SW[7:5]	110	CPU数据输入Cpu_data4bus（RAM输出）



仅定性观测

□ SOC信号测试

- CPU单步运行
- 测试开关设置
- 设计测试程序替换DEMO程序*

开关	位置	功能
SW[1:0]	01	七段码文本显示（低16位）
SW[1:0]	11	七段码文本显示（高16位）
SW[2]	1	CPU单步时钟
SW[7:5]	001	CPU指令字地址PC_out[31:2]
SW[7:5]	011	ROM指令输出Inst_in
SW[7:5]	100	CPU数据存储地址addr_bus(ALU输出)
SW[7:5]	101	CPU数据输出Cpu_data2bus(寄存器B)
SW[7:5]	110	CPU数据输入Cpu_data4bus(RAM输出)
SW[7:5]	111	CPU指令字节地址PC_out



● END