
Computer Organization & Design实验与课程设计

Lab02

IP核集成SOC设计

--建立CPU调试、测试和应用环境

Ma De (马德)

made@zju.edu.cn

2020

College of Computer Science, Zhejiang University

Course Outline

- 一、实验目的
- 二、实验环境
- 三、实验目标及任务

实验目的

1. 初步了解GPIO接口与设备
2. 了解计算机系统的基本结构
3. 了解计算机各组成部分的关系
4. 了解并掌握IP核的使用方法
5. 了解SOC系统并用IP核实现简单的SOC系统

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. Sword 2.0/Sword4.0开发板
3. Xilinx VIVADO2017.4及以上开发工具

□ 材料

无

实验目标及任务

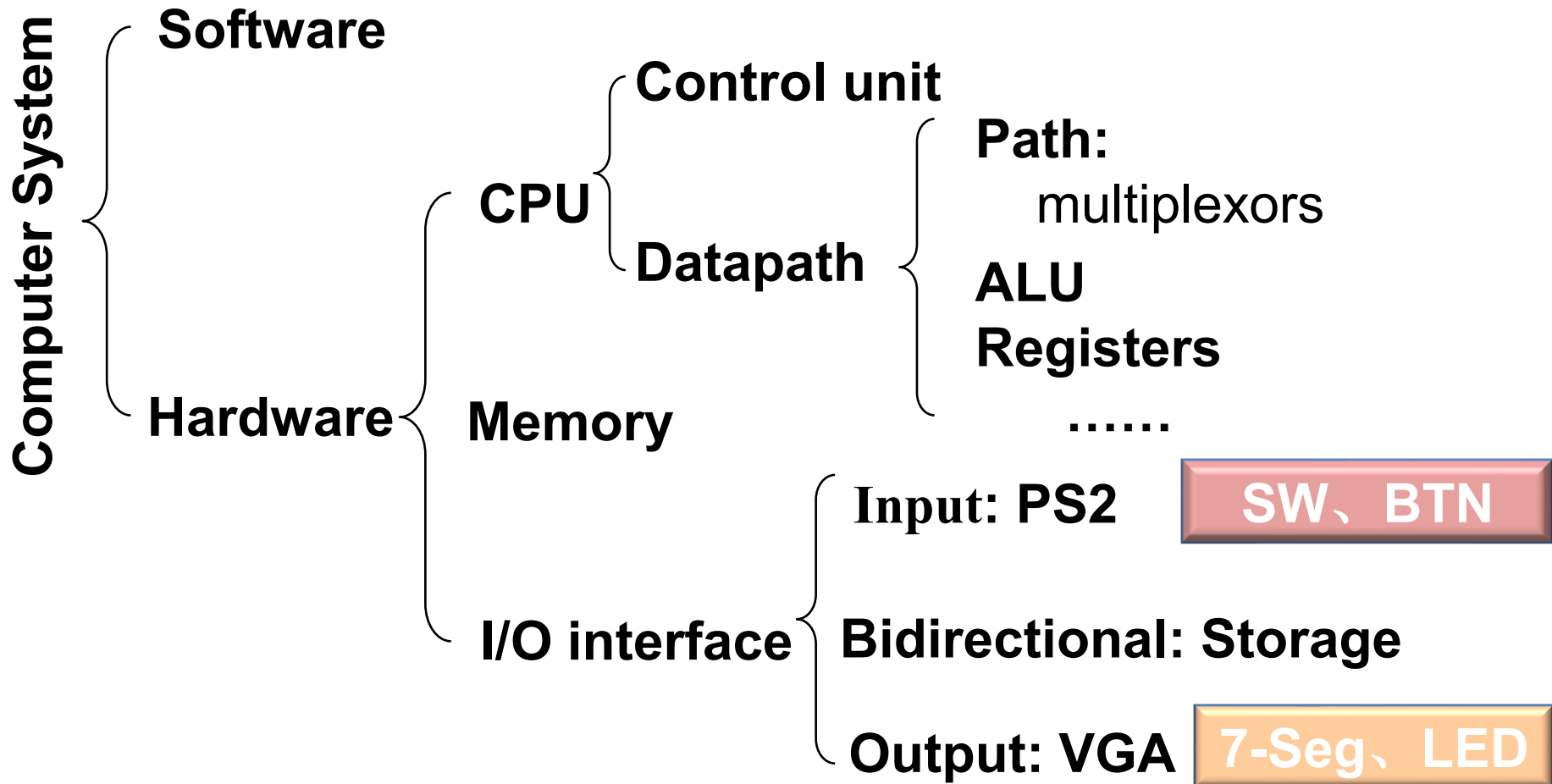
CPU测试环境（基本SOC系统）的建立

- **目标**：熟悉CPU、外设接口和基本功能模块、存储器以及总线各个IP子模块的功能，了解各个IP之间的联系和SOC系统的基本概念
- **任务**：通过第三方IP和已有IP模块建立CPU测试环境（SOC系统的集成实现）-----采用Block Design模块化方式实现，完成由按键消抖、时钟分频、8通道选择、数码管驱动、LED驱动、VGA液晶驱动以及存储ROM、RAM等模块构建的实验平台

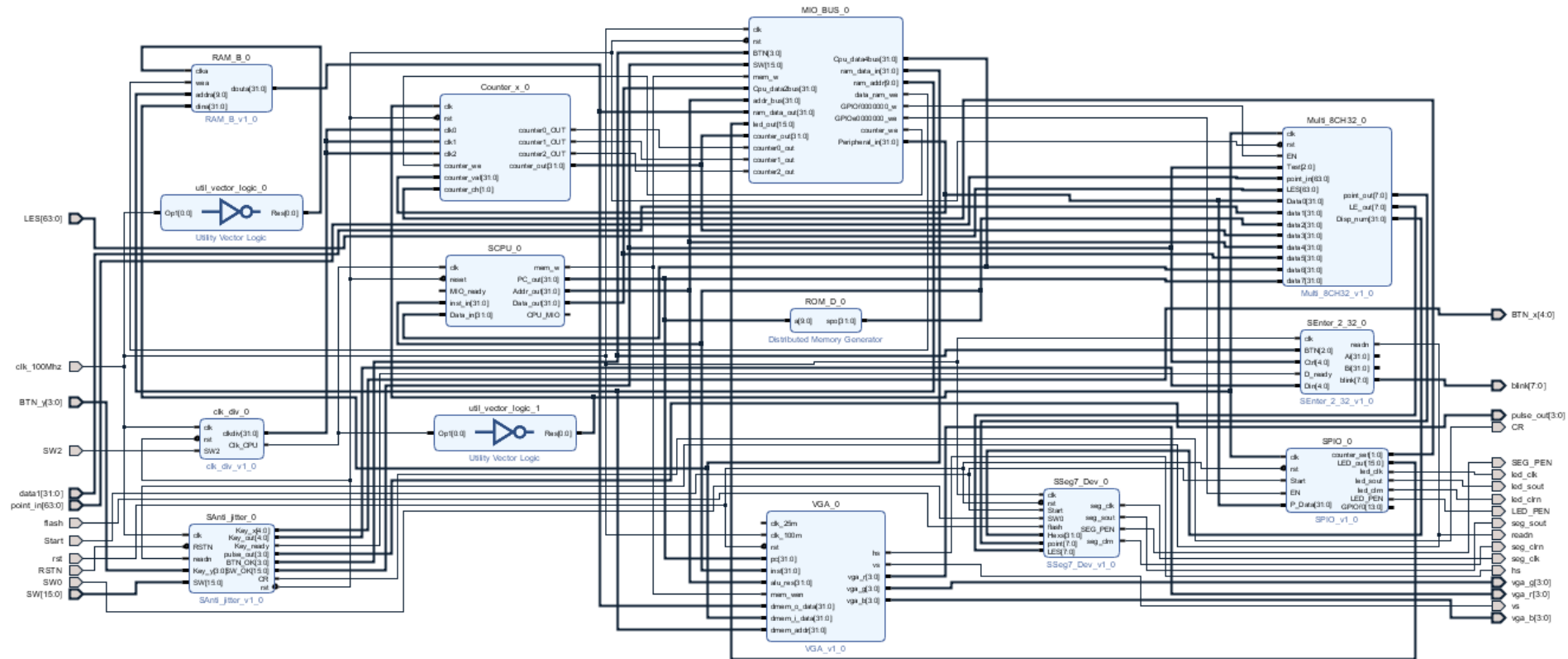
实验系统顶层介绍

Computer Organization

▣ Decomposability of computer systems



本课程实现的SOC或计算机系统



本图仅供概览，详细链接图请参见完整版pdf文档

系统分解为十一个子模块

□ 用此11个模块，互联设计SOC原理图

□ 集成实现SOC

- U1: CPU -SCPU
- U2: ROM -ROM_D
- U3: RAM -RAM_B
- U4: 总线(含外设3~4) -MIO_BUS
- U5: 七段显示接口 -Multi_8CH32
- U6: 外设1- 七段显示设备 -Seg7_Dev
- U7: 外设2-GPIO接口及LED -PIO
- 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 -Anti_jitter
- U10: 通用计数器 -Counter_x
- U11: VGA显示接口 -VGA
- 一些合并、分离、常数IP

(concat\slice\constant lab0使用过)

```
CSSTE_wrapper (CSSTE_wrapper.v) (1)
└─ CSSTE_i: CSSTE (CSSTE.bd) (1)
    └─ CSSTE (CSSTE.v) (30)
        > U1: CSSTE_SCPU_0_0 (CSSTE_S
        > U10: CSSTE_Counter_x_0_0 (CSS
        > U11: CSSTE_VGA_0_0 (CSSTE_VG
        > U2: CSSTE_dist_mem_gen_0_0 (C
        > U3: CSSTE_RAM_B_0_0 (CSSTE_I
        > U4: CSSTE_MIO_BUS_0_0 (CSSTE
        > U5: CSSTE_Multi_8CH32_0_0 (CS
        > U6: CSSTE_SSeg7_Dev_0_0 (CSS
        > U7: CSSTE_SPIO_0_0 (CSSTE_SP
        > U8: CSSTE_clk_div_0_0 (CSSTE_c
        > U9: CSSTE_SAnti_jitter_0_0 (CSST
        .....
        .....
```

系统核心--SCPU介绍

U1-CPU模块：SCPU

□ RISC V 构架

- ⊙ RISC体系结构
- ⊙ 六种指令类型

□ 实现基本指令

- ⊙ 设计实现不少于下列指令

⌘ R-Type: add, sub, and, or, xor, slt, sll, srl, sra, sltu; --10

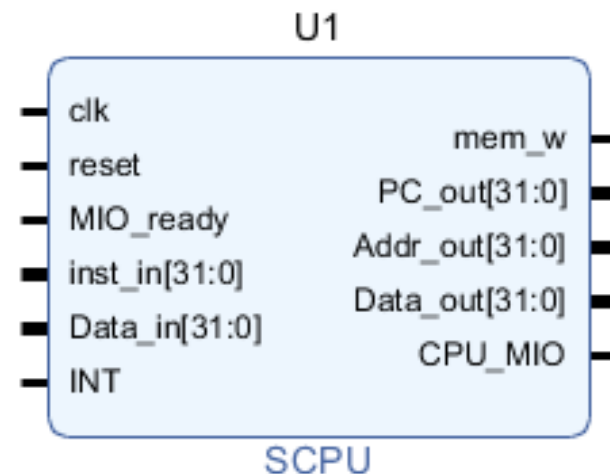
⌘ I-Type: addi, andi, ori, xori, lw, slti, sltiu, slli, srli, srai, jalr -11

⌘ S-Type: sw; --1

⌘ B-Type: beq, bne --2

⌘ U-Type: lui --1

⌘ J-Type: jal; --1



CPU核接口空模块-SCPU.v

```
module SCPU( input wire clk,
              input wire reset,
              input wire [31:0] inst_in,           //指令输入总线
              input wire [31:0] Data_in,           //数据输入总线

              output wire Mem_RW,                  //存储器读写控制
              output wire [31:0] PC_out, //程序空间访问指针
              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
              output wire CPU_MIO,                  // Not used
              input wire INT                        //中断
            );

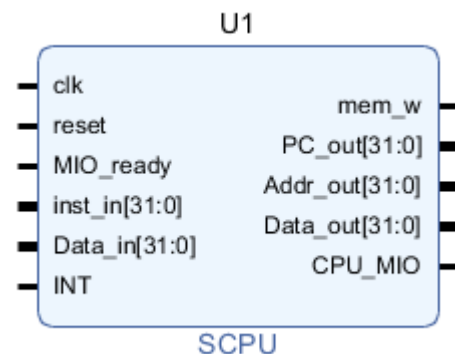
endmodule
```

U1-CPU模块：SCPU

□ RISC32I 构架内核

- ⊙ RISC体系结构
- ⊙ 六种指令类型

□ 本实验系统平台SCPU功能

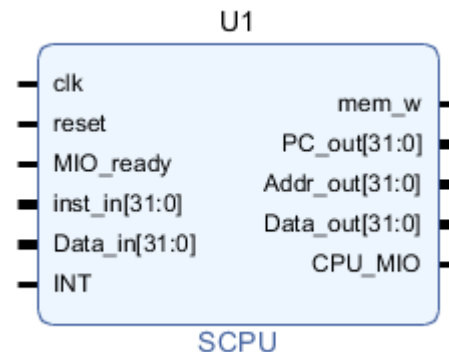


- ⌚ **指令控制**：CPU接收输入的指令inst_in[31:0]并按程序逻辑顺序产生下一条指令地址PC_out[31:0]；所以CPU中应该包含指令指针寄存器(PC)和地址计算单元。
- ⌚ **操作控制**：CPU接收指令后，能产生指令执行过程要求的、指令功能约定的微操作控制信号；所以CPU中应该包含指令译码单元和微操作控制信号的产生单元。
- ⌚ **数据加工**：CPU按照指令的要求，对输入的数据Data_in[31:0]进行相应的加、减、乘、除等算数运算；与、或、异或等逻辑运算；左移、右移等移位操作……后输出结果数据Data_out[31:0]。

U1-CPU模块：SCPU

□ RISC32I 构架内核

- ⊙ RISC体系结构
- ⊙ 六种指令类型



□ 本实验系统平台SCPU功能

- ⌚ **外部访问**：CPU能够进行存储器和I/O访问，通过地址输出 Addr_out[31:0]能够访问到外部存储区域。
- ⌚ **中断处理**：CPU能够处理外部的中断和内部的异常情况，所以应该包含中断控制处理单元，通过INT输入来进行中断操作。
(该部分功能将在后续实验深入学习实现)
- ⌚ **时序控制**：CPU能够提供对所有控制信号的定时控制，所以CPU应该包含时序信号产生电路。

系统存储—ROM\RAM介绍

U2-指令代码存储模块：ROM_B

□ ROM_D/B

用Distributed Memory Generator没有clk信号
请编辑删除clka引脚

- FPGA内部存储器

- Block Memory Generator或Distributed Memory Generator

- 容量

- $1024 \times 32\text{bit}$

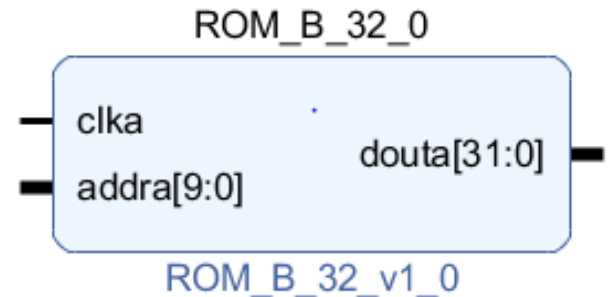
□ 本实验采用实验一方法生成核

- ROM初始化文档：**I_mem.coe**

- 核调用模块ROM_B.xco

- 生成后自动调用关联，不需要空文档

- 建议在本实验的BD环境中重新生成，方便随时更换初始化程序

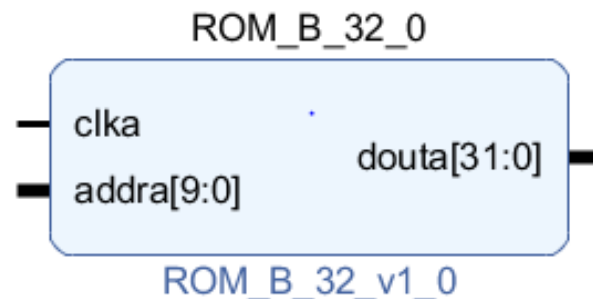


ROM_B调用方式

□ ROM调用接口信号

```
ROM_B    U2 ( .clka(clk_m),           //存储器时钟，与CPU反向
               .addra(PC_out[11:2]),   // ROM地址PC指针，来自CPU
               .douta(inst[31:0])     // ROM输出作为指令输入CPU
            );
```

□ 若是固核调用则不需要空模块文档



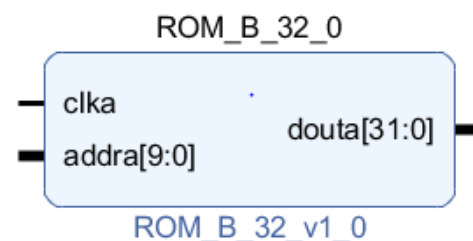
U2-指令代码存储模块：ROM_B

□ ROM_D/B

- 只读存储器，不支持写操作
- 输入地址，输出相应地址空间所存储的数据

□ 本实验平台ROM的作用

- 作为指令代码的存储部件
- 输入来自CPU的PC指针输出，PC_out；在时钟信号的作用下输出对应的指令信息作为CPU的指令输入inst[31:0]
- ROM能否被正确访问，输出的指令信息是否正确是CPU正常运转的关键。



同步ROM:时钟控制
异步ROM:无需时钟

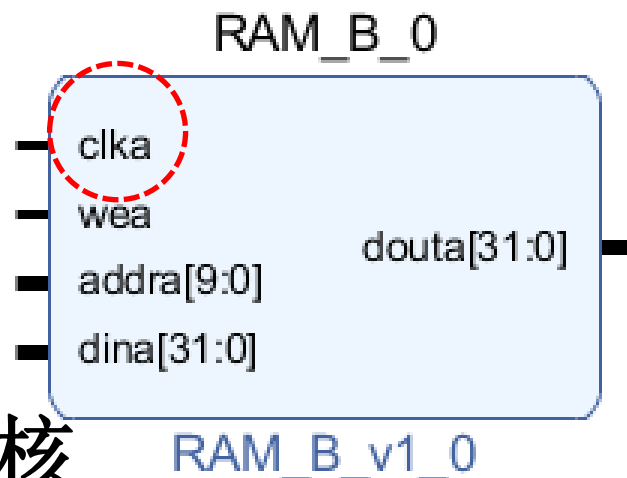
U3-数据存储模块：RAM_B

□ RAM_B

- FPGA内部存储器
 - Block Memory Generator
- 容量
 - $1024 \times 32\text{bit}$

□ 本实验采用实验一生成的固核

- RAM初始化文档：**D_mem.coe**
- 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ROM类同

□ RAM调用接口信号

```
RAM_B    U3 ( .clka(clk_m),           // 存储器时钟，与CPU反向
            .wea(data_ram_we),        // 存储器读写，来自MIO_BUS
            .addra(ram_addr),         // 地址线，来自MIO_BUS
            .dina(ram_data_in),       // 输入数据线，来自MIO_BUS
            .douta(ram_data_out)      // 输出数据线，来自MIO_BUS
            );
```

□ 固核调用不需要空模块文档

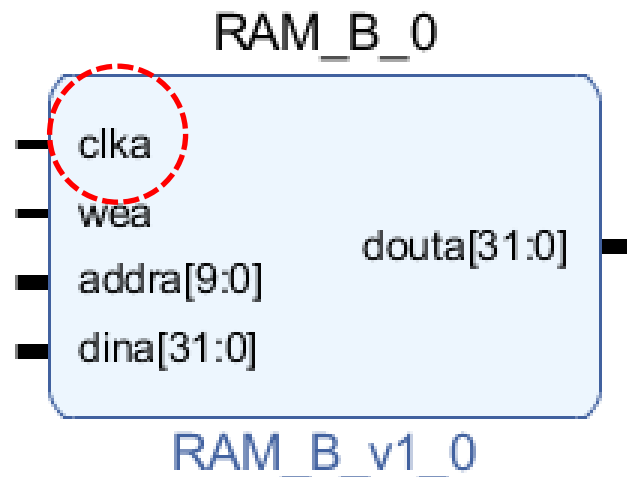
U3-数据存储模块：RAM_B

□ RAM_B

- 随机存储器，支持读写操作
- 有读写使能信号wea

□ 本实验平台RAM的作用

- 作为CPU应用的外部数据存储部件
- 输入地址；wea使能为写；输入数据；则数据写入RAM；
- 输入地址；wea使能为读；输出数据；
- 来自CPU的运算结果可以存放在RAM中；CPU需要的操作数据可以从RAM获取。



单口RAM:同时刻只能进行读或写
双口RAM:同时刻能够进行读和写

系统总线—MIO_BUS介绍

U4-总线接口模块: MIO_BUS

□ MIO_BUS

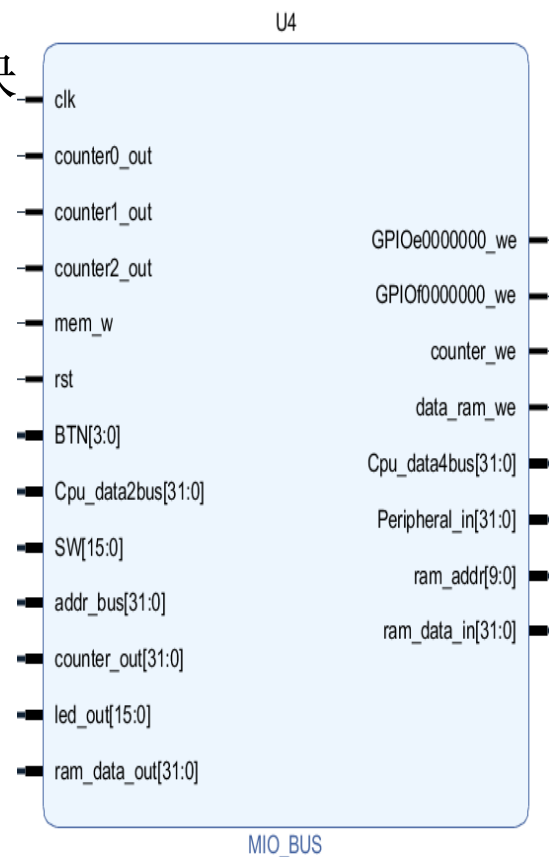
- CPU与外部数据交换接口模块
- 本课程实验将数据交换电路合并成一个模块
 - 非常简单, 但非标准, 扩展不方便
 - 后继课程采用标准总线
 - Wishbone总线

□ 基本功能

- 数据存储、7-Seg、SW、BTN和LED等接口

□ 本实验用IP 软核- U4

- 核调用模块MIO_BUS.vco
- 核接口信号模块(空文档): MIO_BUS_IO.v



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 GPIOf00000000_w,             // GPIOfffff00_we
                output wire GPIOe00000000_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	<pre>clk_100M, botton_out[3], BTN [3:0] , SW [15:0], mem_w, Cpu_data2bus [31:0] , addr_bus [31:0] , ram_data_out [31:0] , led_out [15:0], 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]);</pre>	<pre>//主板时钟 //复位，按钮BTN3 //4位原始按钮输入 //16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</pre>
----------------	-----------	--	---

endmodule

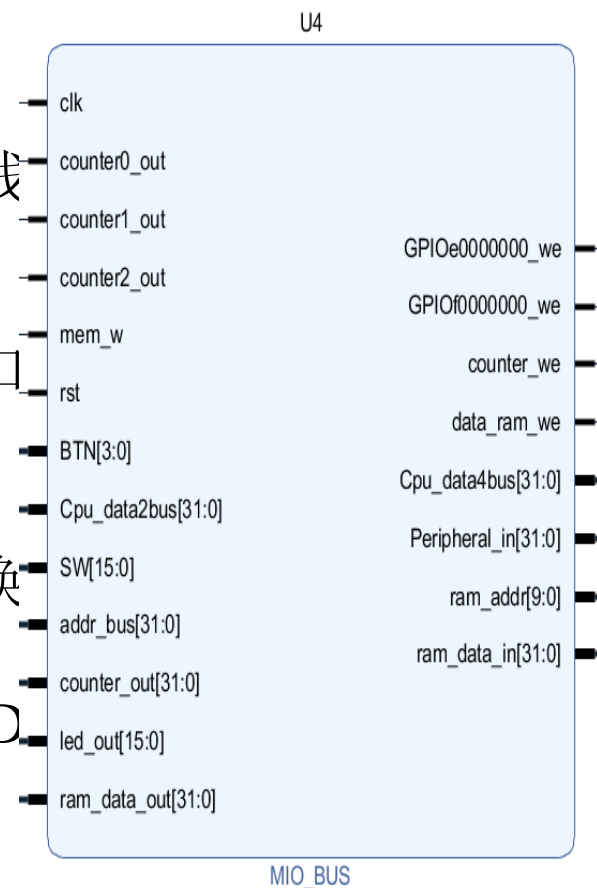
U4-总线接口模块：MIO_BUS

□ MIO_BUS

- 计算机系统中各个功能部件之间传送信息的公共通道
- 按功能划分为数据总线、地址总线和控制总线

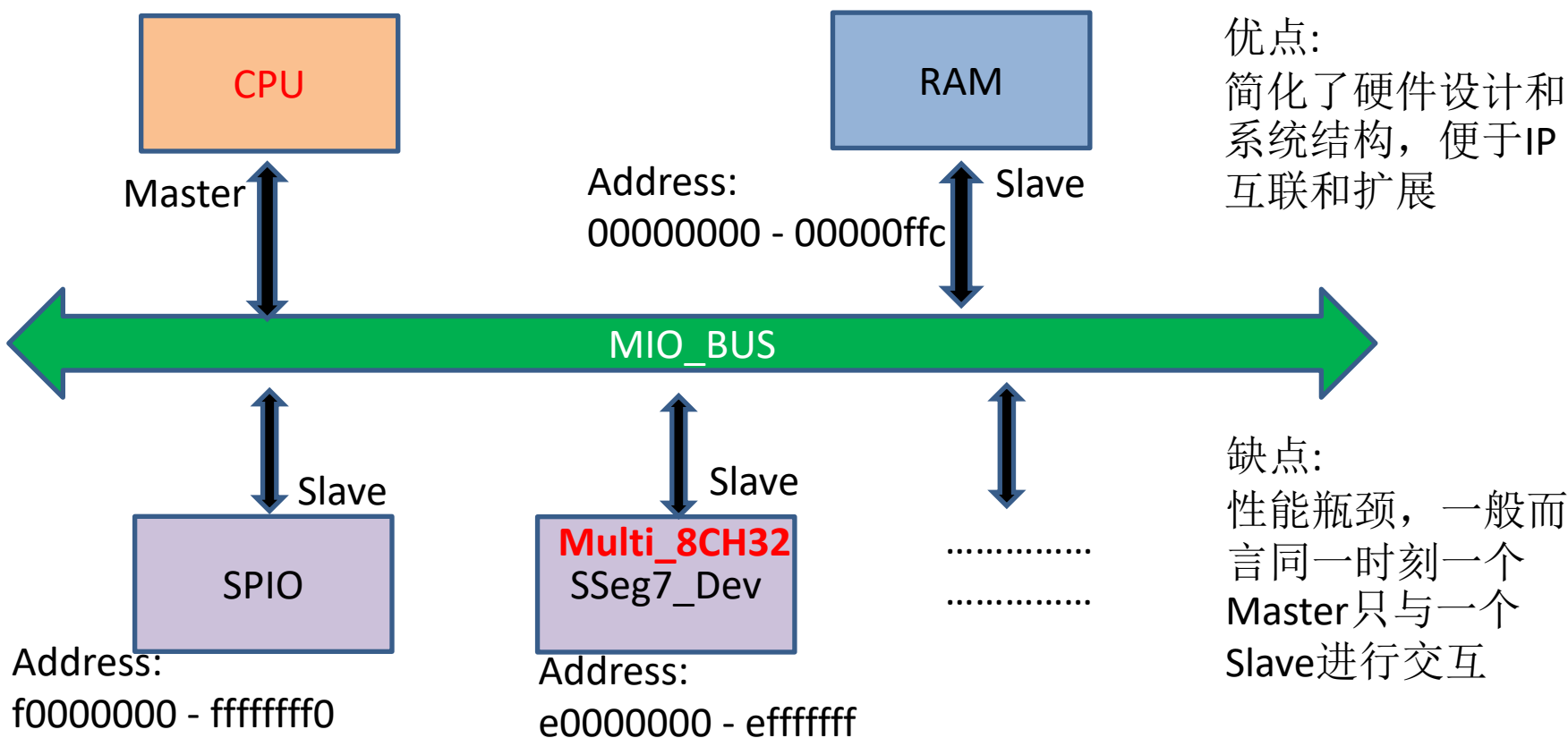
□ 本实验平台总线功能

- 作为通路连接CPU、数据存储器、外设IO接口设备（SW拨码开关、按键BTN、显示设备LED等）
- 完成CPU与数据存储器之间的数据存储和转换操作
- 接收拨码和按键的输入信号，转换完成对LED等显示部件的控制操作



U4-总线接口模块: MIO_BUS

- 本实验构建的系统大致如下图所示，总线作为子系统之间共享的通信链路，其接口分为两种，**Master**(主设备)：主动发起读写操作；**Slave**（从设备）：被动响应读写操作，通过地址映射来选择使用哪一个从设备。因此总线还需具备仲裁功能以决定哪一个主设备发起总线访问；总线需具备译码功能通过地址来选择哪一个从设备来响应。



系统外设—GPIO接口设备介绍

—实验所用的GPIO接口非常简单
除Seg7设备外, 接口与设备合二为一

外部设备模块：GPIO接口及设备--SPIO

□ GPIO输出设备一

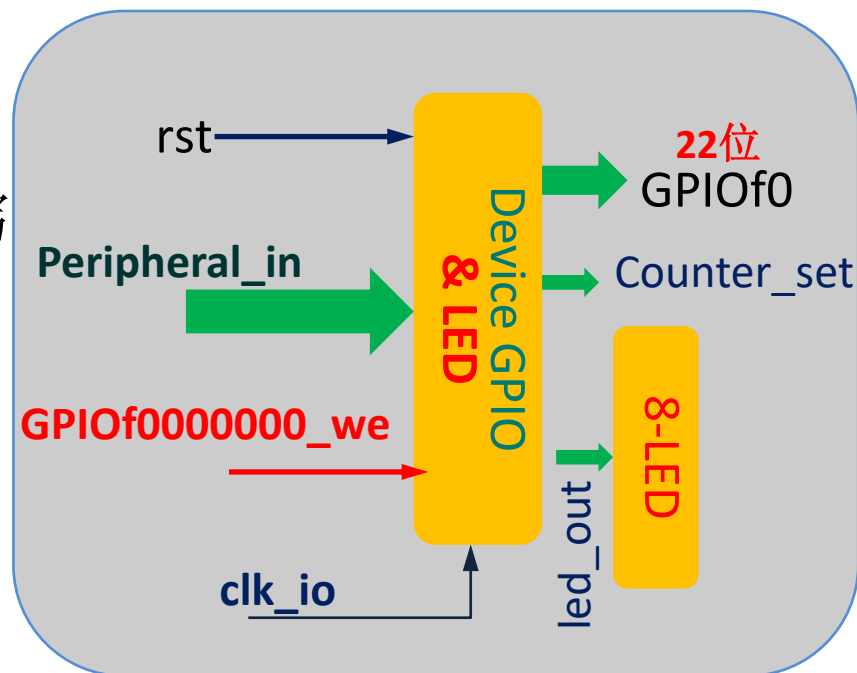
- 地址范围=f0000000 - fffffff0 (ffffff00-ffffff0)
- 读写控制信号：GPIOf0000000_we(GPIOffffff00_we)
- {GPIOf0[21:0],LED,counter_set}

□ 基本功能

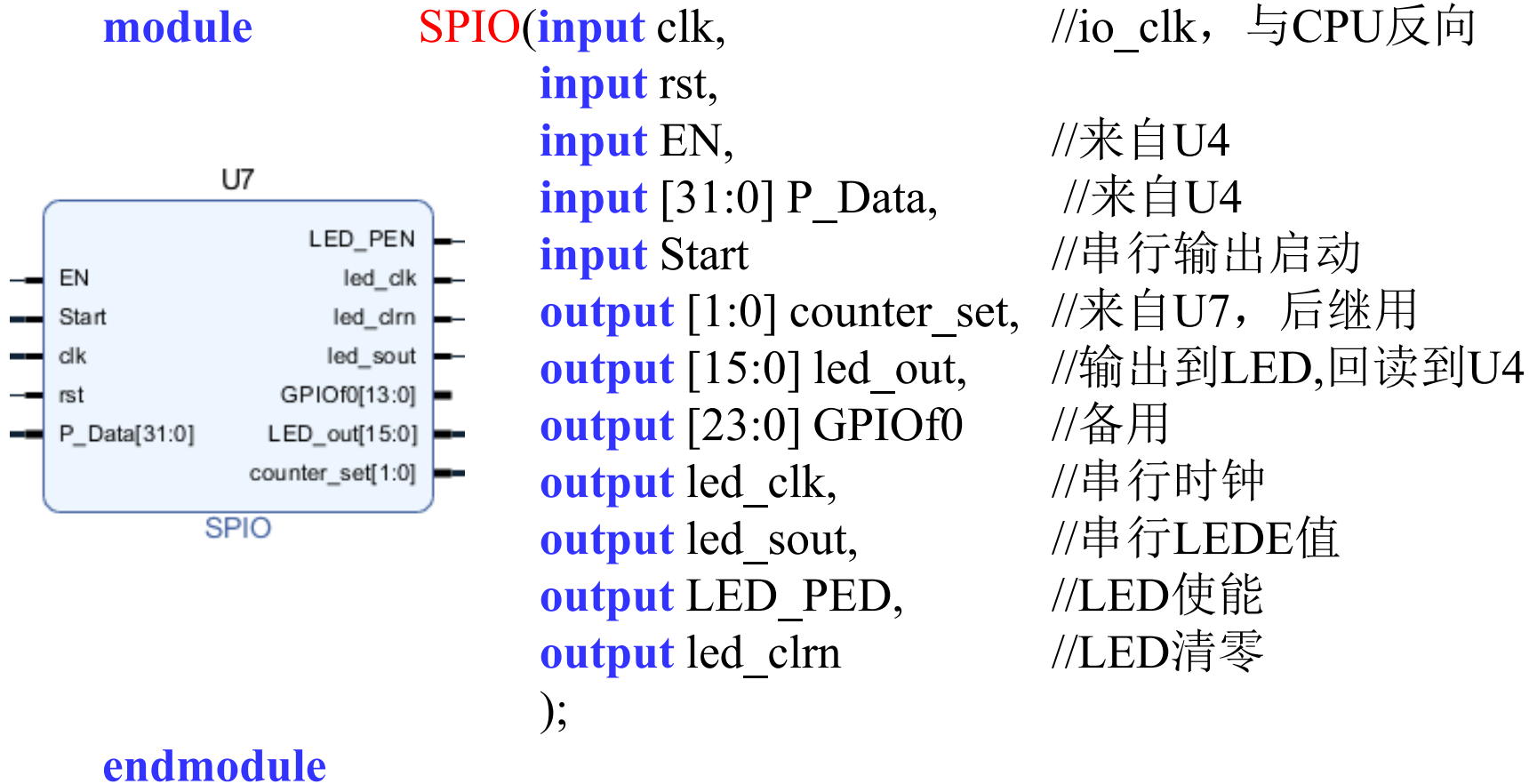
- LEDs设备和计数器控制器读写
- 可回读，检测状态
- 逻辑实验LED模块改造

□ 本实验用IP 软核- U7

- 核调用模块SPIO.vco
- 核接口信号模块(空文档): SPIO_IO.v



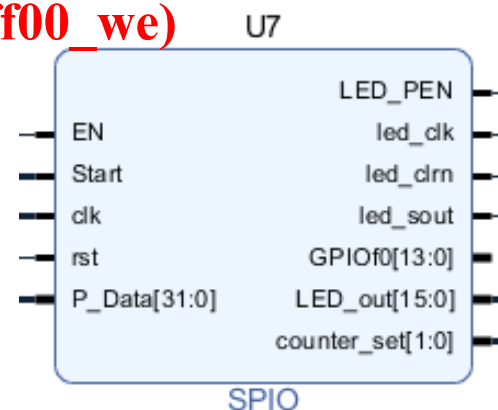
通用接口与设备—IP核端口-SPIO.v



外部设备模块：GPIO接口及设备--SPIO

□ GPIO输出设备一

- 地址范围=f0000000 - fffffff0 (ffffff00-fffffff0)
- 读写控制信号： **GPIOf0000000_we(GPIOffffff00_we)**



□ 本实验平台SPIO作用

- 作为LED显示部件的驱动和计数器Counter的通道控制,本课程用于调试显示和CPU的简单外设
- 输入32位二进制数据： P_Data
 - clk=时钟， EN： 输出使能， Start： 串行扫描启动， rst=复位
- 串行输出： led_clk=时钟， led_sout=串行输出数据， LED_PEN=使能， led_clrn=清零
- 并行输出： LED_out、 counter_set、 GPIOf0

外部设备模块：GPIO设备二-SSeg7_Dev

□ 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)

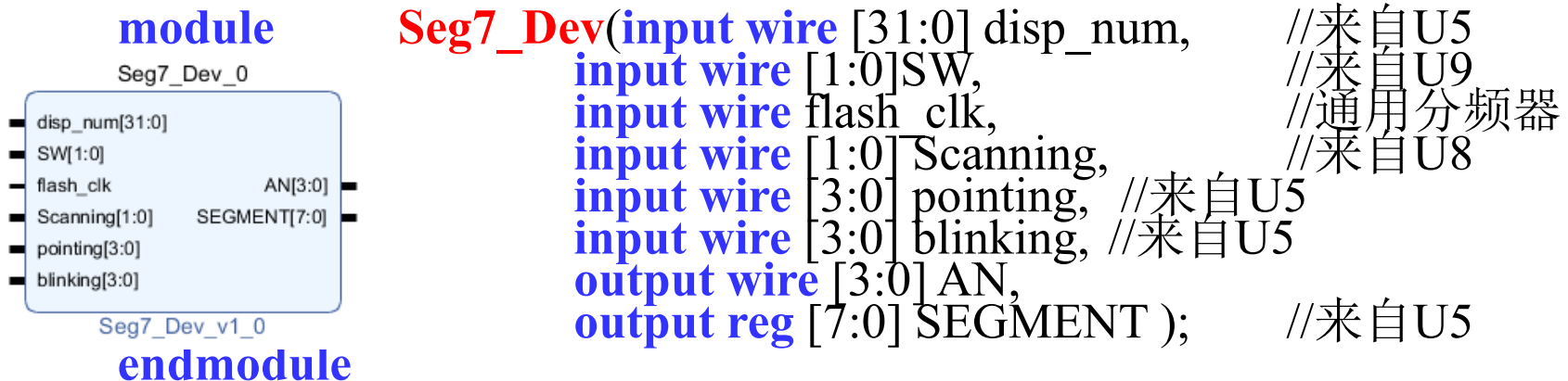
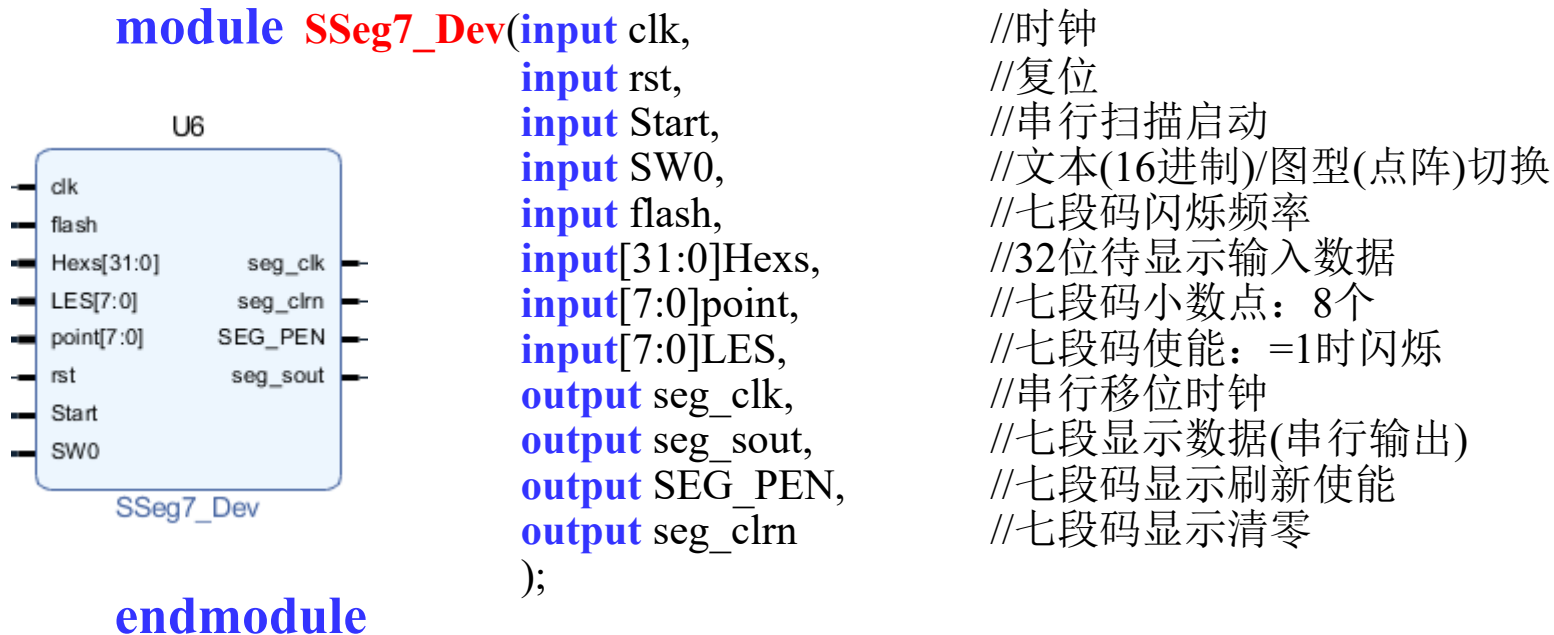
□ 基本功能

- 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； **Sword平台不需要**
 - SW[1:0]=11，显示高4位 **Sword平台不需要**
- 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- **U5**

- 核调用模块SSeg7_Dev.v
- 核接口信号模块(空文档)：SSeg7_Dev.v

通用设备二IP核端口~Seg7_Dev.v



外部设备模块：GPIO设备二 ~SSeg7_Dev

□ 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)

□ 本实验平台Seg7_Dev作用

- 七段数码管的显示驱动
- 输入来自8通道选通Multi_8CH32输出的8位小数点、8位七段码使能、动态刷新频率、32位待显示数据以及拨码**SW[1:0]**的控制信号
- 输出七段显示数据
- 拨码控制
 - SW[0]=1，显示文本
 - SW[0]=0，显示图形
 - SW[1]=1，显示文本高16位
 - SW[1]=0，显示文本低16位

通用设备二接口模块 **Multi_8CH32**

□ GPIO输出设备二接口模块

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

□ 基本功能

- 7段码输出设备接口模块
- 逻辑实验显示通道选择模块改造
- 通道0作为显示设备接口

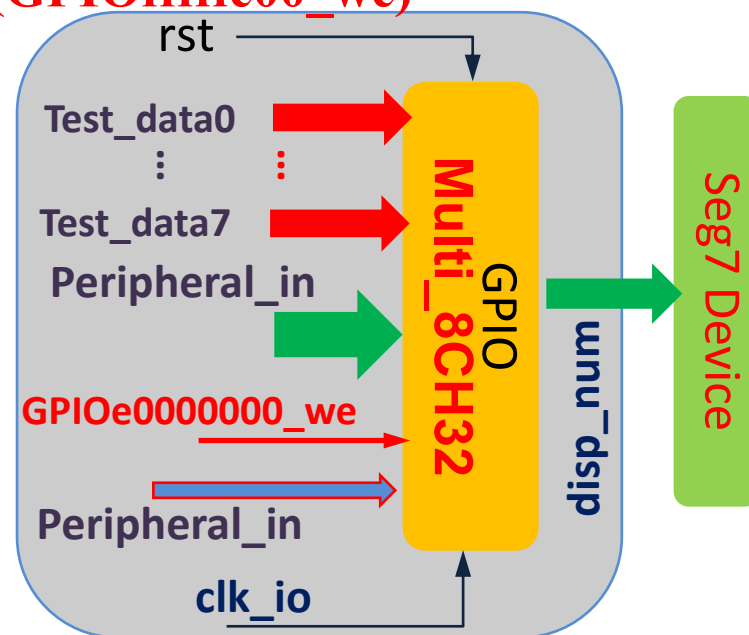
□ **GPIOe0000000_we=1**

□ **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- **U5**

- 核调用模块Multi_8CH32.vco
- 核接口信号模块(空文档): **Multi_8CH32_IO.v**



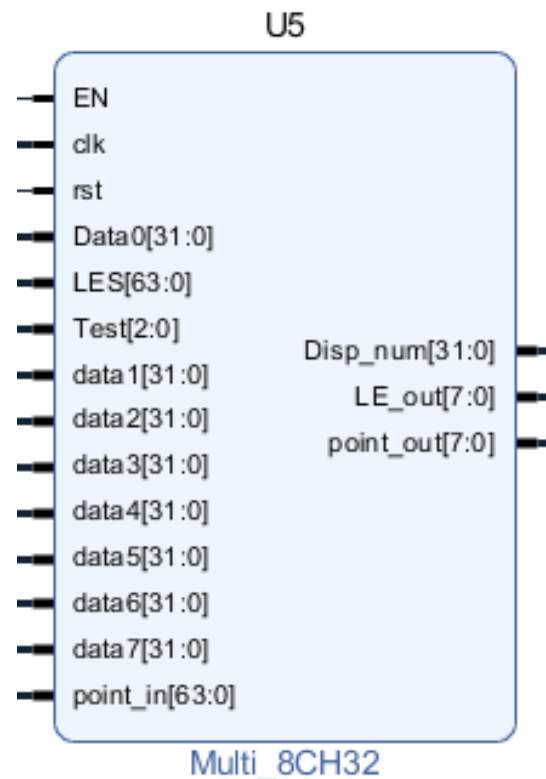
八数据通路模块：Multi_8CH32

□ 多路选择器的简单应用

- 功能：多路信号显示选择控制
 - 用于CPU等各类信号的调试和测试
 - 由1个或多个8选1选择器构成

□ 八路数据通路模块接口

- 与8位七段显示(32位数据)器连接
- I/O接口接口信号功能
 - clk：同步时钟(后期扩展预留)
 - rst：复位信号(后期扩展预留)
 - EN：使能信号(仅控制通道0)
 - SW[7:5]：通道选择控制，控制data[7:1]输出
 - Point_in(63:0)：小数点输入
 - 每个通道8位，共64位
 - LES(63:0)：使能LED (闪烁)控制输入
 - 每个通道8位，共64位
 - Data0-Data7[31:0]：数据输入通道(Data0特殊)
 - LES_out(7:0)：当前使能位输出
 - Point_out(7:0)：当前小数点输出



通用设备二接口-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]blink_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[3:0] point_out, //小数点输出
                    output reg[3:0] blink_out, //闪烁控制输出
                    output [31:0] disp_num //接入7段显示器
                );

    U5: Multi_8CH32
        EN
        clk
        rst
        Data0[31:0]
        LES[63:0]
        Test[2:0]
        data1[31:0]
        data2[31:0]
        data3[31:0]
        data4[31:0]
        data5[31:0]
        data6[31:0]
        data7[31:0]
        point_in[63:0]
        Disp_num[31:0]
        LE_out[7:0]
        point_out[7:0]

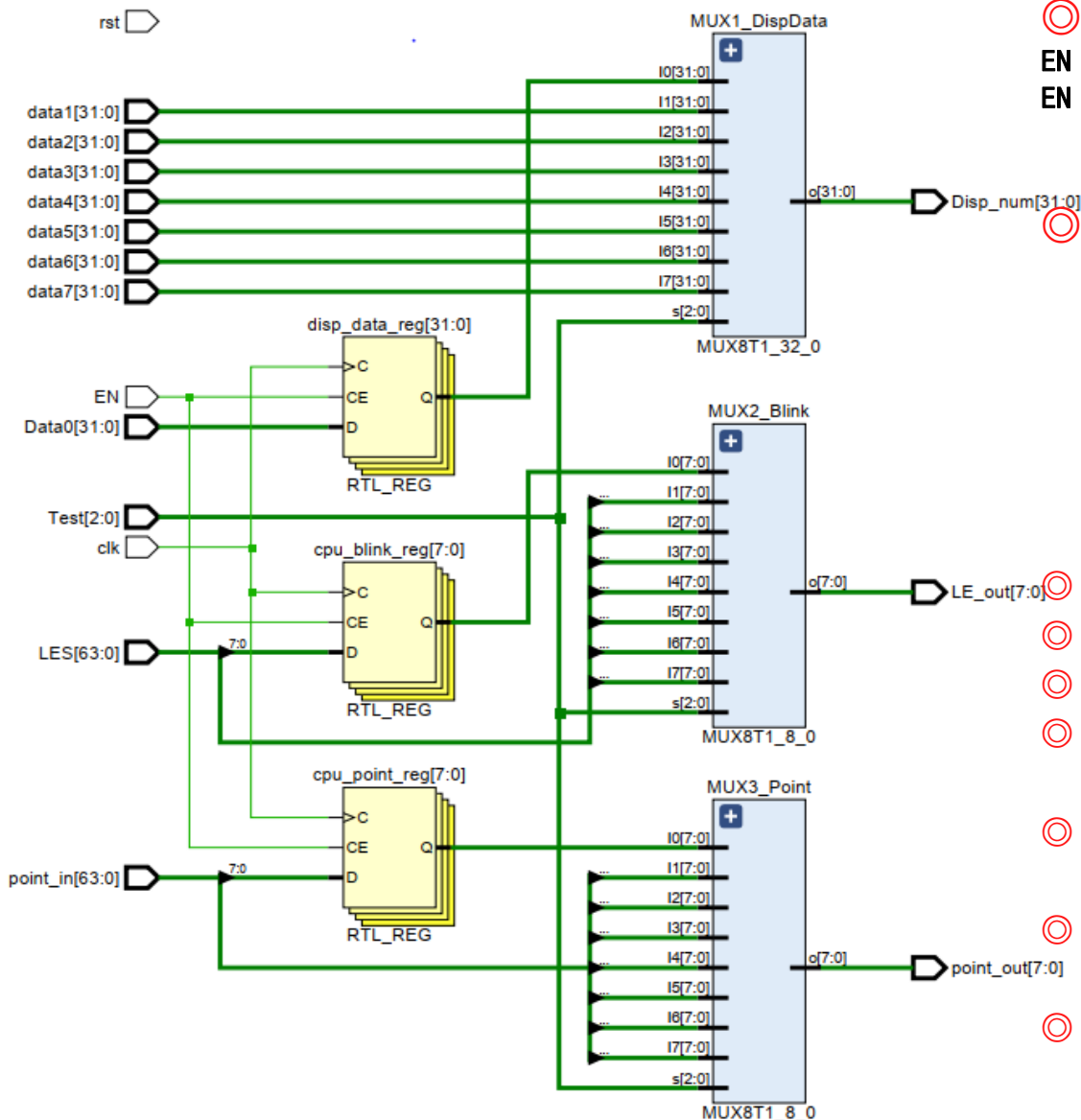
```

Multi_8CH32

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(blink_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  
                  .blink_out(blink_out),           //输出到U6  
                  .disp_num(disp_num)             //输出到U6  
                  );
```

32位数据八通道模块：调用MUX8T1_32



◎ 控制信号EN：控制通道0

EN = 0; 三个黄色的RTL_REG使能端CE=0; Q=0;
EN = 1; 三个黄色的RTL_REG使能端CE=1; Q1=Data0;
Q2=LES[7:0];
Q3=point_in[7:0];

◎ 控制信号Test[2:0]：连接拨码开关SW[7:5]; 共000-111; 8种输入值，选通控制通道0-7. (注：当取000时，选通通道0；而此时其输入数据为Q，又被EN控制；所以通道0输出数据，需要两个使能信号同时有效)

- ◎ 输入Data0[31:0]：通道0输入数据
- ◎ 输入data1-7：通道1-7输入数据
- ◎ 输入LES[63:0]：LED驱动电平信号输入
- ◎ 输入point_in[63:0]：数码管小数点输入
- ◎ 输出MUX1_DispData：被选通通道输出数据
- ◎ 输出MUX2_Blink：被选通的通道输出LED驱动电平信号
- ◎ 输出MUX3_Point：被选通的通道输出小数点

通用设备二接口模块 **Multi_8CH32**

□ GPIO输出设备二接口模块

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

□ 本实验平台**Multi_8CH32**作用

- 作为七段数码管显示信息的输入接口，8通道选通任一输入信息将其转发给数码管显示
- 输入Test[2:0]为选通控制信号，由拨码SW[7:5]控制
 - **SW[7:5]=000; 输出Data0;数据源为总线的Peripheral_in, 源头是Cpu_data2bus**
 - **SW[7:5]=001; 输出data1;数据源为CPU指令字节地址PC_out[31:2]**
 - **SW[7:5]=010; 输出data2;数据源为指令存储器数据输出Inst_in (31:0)**
 - **SW[7:5]=011; 输出data3;数据源为计数器输出Counter_out(31:0)**
 - **SW[7:5]=100; 输出data4;数据源为数据存储器地址输出addr_out (31:0)**
 - **SW[7:5]=101; 输出data5;数据源为Cpu_data2bus,源头是CPU数据输出Data_out**
 - **SW[7:5]=110; 输出data6;数据源为CPU数据输入Cpu_data4bus, 源头是数据存储器数据输出ram_data_out [31:0]**
 - **SW[7:5]=111; 输出data7;数据源为CPU指令字节地址PC_out[31:0]**

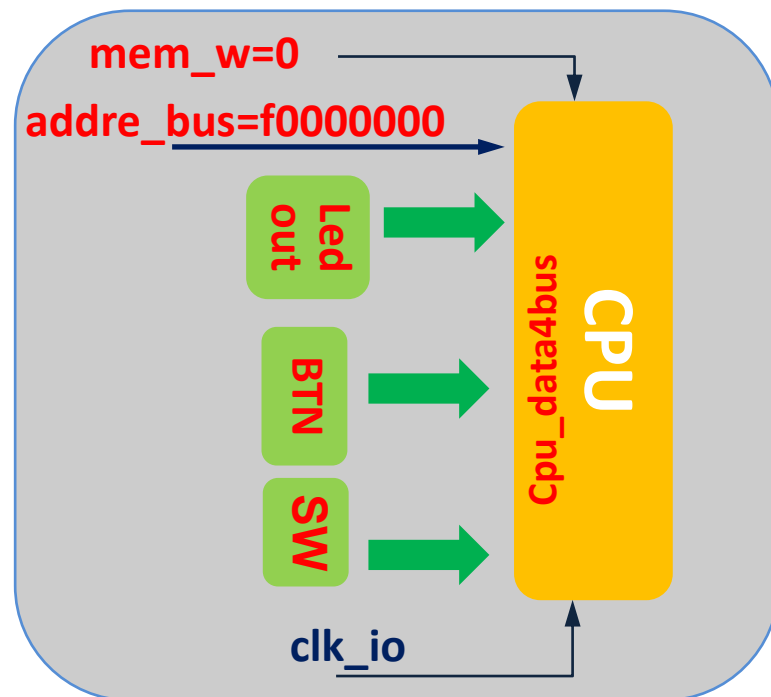
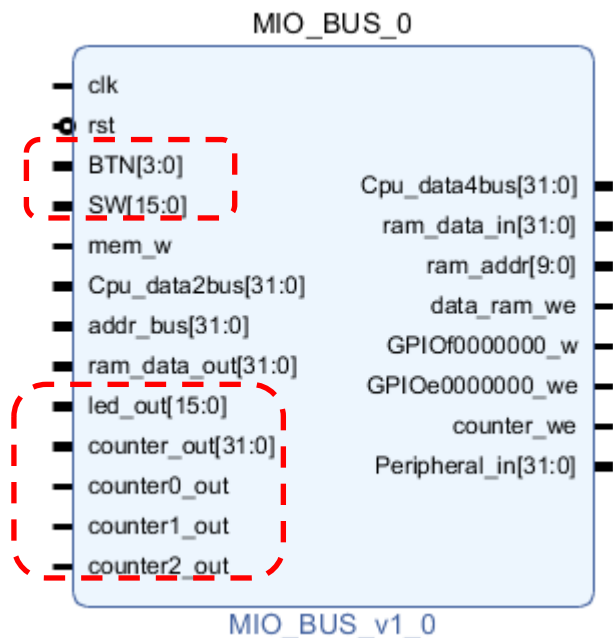
外部设备模块：GPIO接口设备三、四

Device_GPIO_SW_BTN

8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, 1'h000, BTN, SW};
```



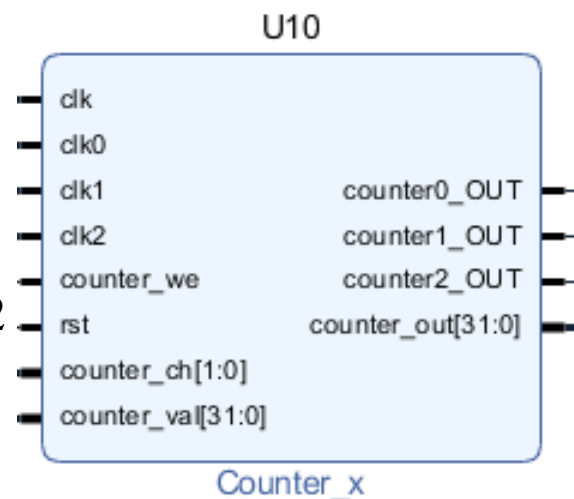
外部设备五：通用计数器模块 Counter_x.v

□ 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_x.v
- 核接口信号模块(空文档): Counter_x.v

通用计数器IP核端口 -Counter_x.v

```

module Counter_x(input clk,           //io_clk
                  input rst,
                  input clk0,          //clk_div[7], 来自U8
                  input clk1,          //clk_div[10], 来自U8
                  input clk2,          //clk_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

```

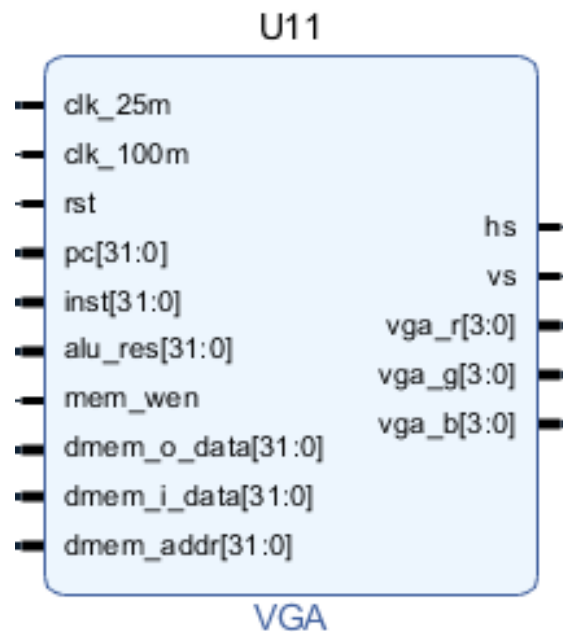
显示调试设备：VGA VGA.v

□ CPU信号调试显示

□ 基本功能

- 输入像素点扫描控制时钟和显示工作时钟。
- 驱动显示CPU调试信息于液晶屏
- 输入各种调试信号数据：
 - Clk_25m=点时钟频率（640x480）
 - Clk_100m=显示时钟频率，
 - pc及其他输入均为调试信号
- 输出VGA标准驱动：
 - Hs 列填充控制
 - Vs 行扫描控制
 - Vga_r;vga_g;vga_b RGB信号

□ 本实验用IP 软核- U11



VGA显示·IP核端口

-VGA.v

```
module VGA(input wire clk_25m,           //扫描控制时钟
            input wire clk_100m,         //显示时钟
            input wire rst,               //复位
            input wire [31:0] pc,         //PC地址待显示
            input wire [31:0] inst,       //指令待显示
            input wire [31:0] alu_res,    //alu输出待显示
            input wire mem_wen,           //存储器控制待显示
            input wire [31:0] dmem_o_data, //存储器数据输出
            input wire [31:0] dmem_i_data, //存储器数据输入
            input wire [31:0] dmem_addr,  //存储器地址
            output wire hs,                //列填充控制
            output wire vs,                //行扫描控制
            output wire [3:0] vga_r,       //RGB信号
            output wire [3:0] vga_g,
            output wire [3:0] vga_b
endmodule
```

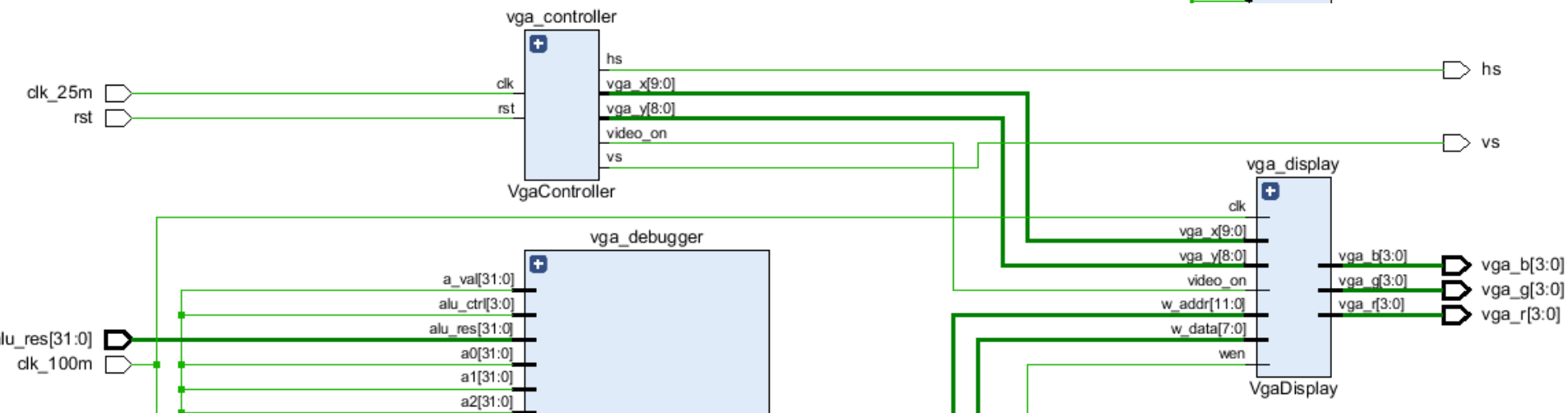
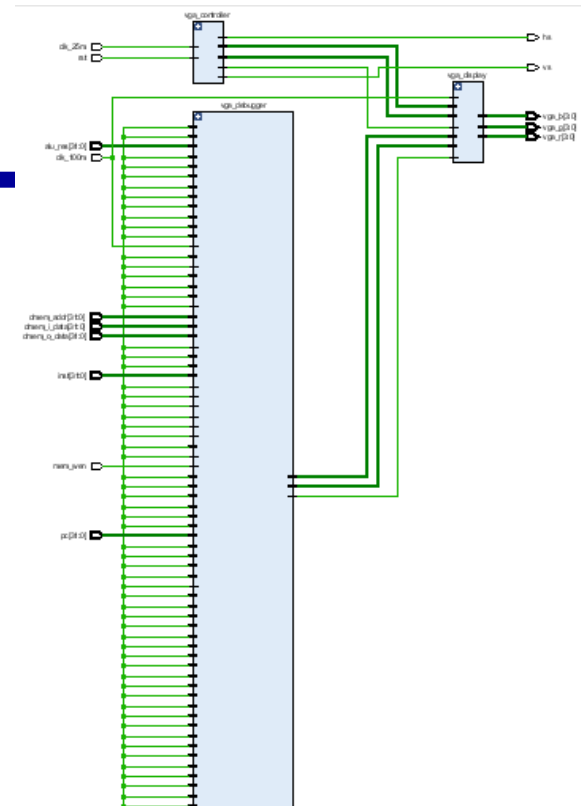
显示驱动可显示其他中间状态，需在CPU输出端口做调整

eg:

```
input wire [4:0] rs1,
input wire [31:0] rs1_val,
input wire [4:0] rs2,
input wire [31:0] rs2_val,
input wire [4:0] rd,
input wire [31:0] reg_i_data,
input wire reg_wen,
```

显示调试设备：VGA

- VGA显示设备模块内部由三个子模块组成：
- Vga_controller: 水平和垂直扫描控制，时钟 25mhz
- Vga_display: 直接接受像素显示信息，输出 RGB驱动信号
- Vga_debugger: 显示信息输入及格式转换；可支持多个调试端口，只需在顶层将端口引出即可；直接连接CPU的输出调试口



系统辅助模块介绍

—通用分频、开关去抖动模块

通用分频模块: **clk_div**

□ 计数分频模块

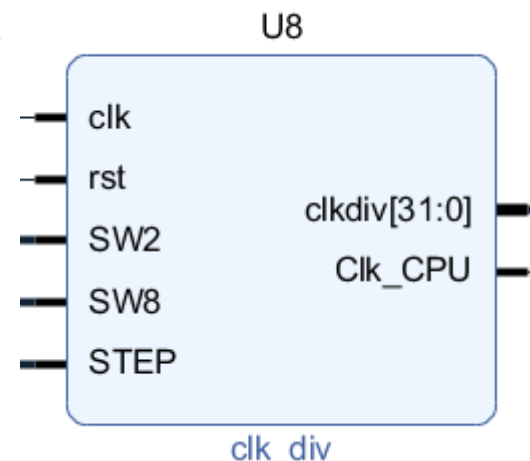
- 用于要求不高的各类计数和分频
 - CPU、IO和存储器等
- 对延时和驱动有要求的需要BUFG缓冲
- 对于时序要求高的需要用DCM实现

□ 基本功能

- 32位计数分频输出: **clkdiv**
- 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.v
- 核接口信号模块(空文档): clk_div.v



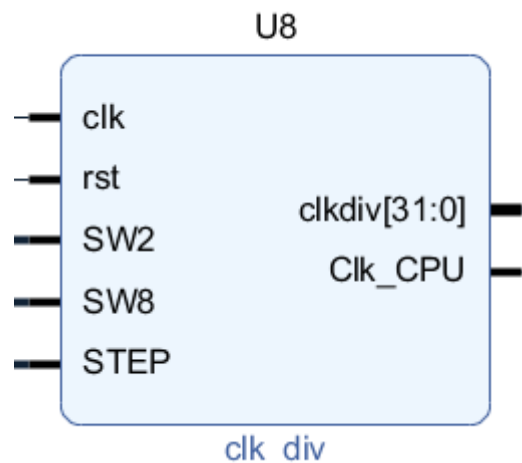
逻辑实验通用分频模块M1优化: `clk_div.v`

通用计数分频模块

- 用于计算机组成实验辅助模块
- 逻辑实验通用计数模块改造
- 增加CPU单步时钟输出

基本功能

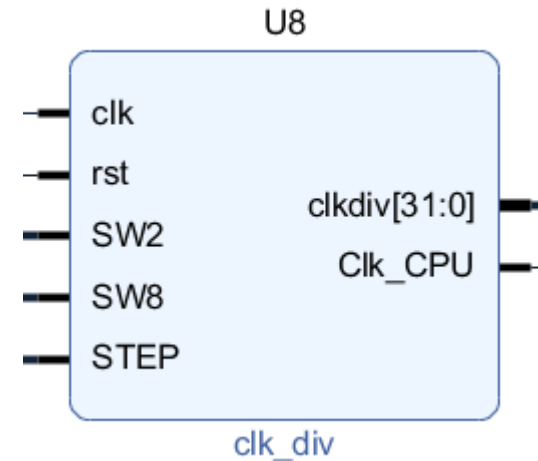
- 32位计数分频输出: `clkdiv`
- CPU时钟输出: `Clk_CPU`
- 拨码开关的2、8号位、按键，`SW[2][8]`控制`Clk_CPU`输出
 - 当拨码开关二、八号位，置低电平即`SW[2][8]=00`；
输出时钟为全速频率（50MHz或25MHz）
 - 当拨码开关二、八号位，分别置高电平即`SW[2][8]=10`；
输出时钟为自动单步频率（ 2^{24} 分频，`clkdiv[24]`）
 - 当拨码开关八号位，置高电平即`SW[2][8]=x1`；
输出时钟为手动单步频率（按键`BYN_Y[0]`输入`STEP`）



通用分频IP核端口 -clk_div.v

```
module clk_div(input clk,  
               input rst,  
               input SW2,SW8,STEP,  
               output [31:0]clkdiv,  
               output Clk_CPU  
               );  
  
endmodule
```

拨码SW2,8控制输出时钟
SW2,8=00;CPU全速时钟;
SW2,8=10;CPU自动单步时钟;
SW2,8=x1;CPU手动单步时钟
STEP;



通用分频模块端口信号及描述参考

通用分频器模块行为描述结构

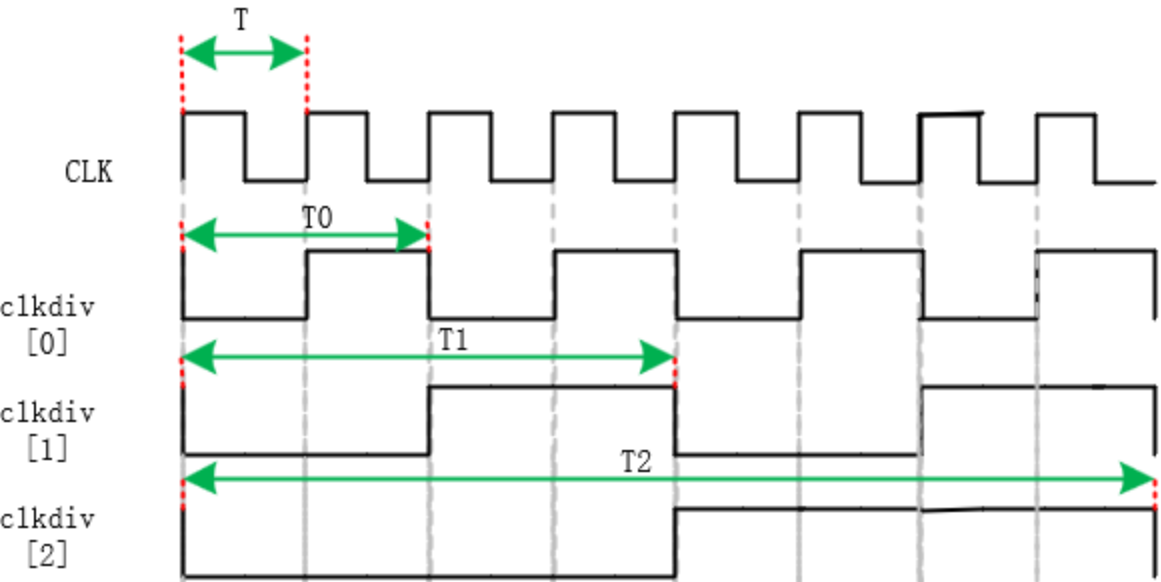
```
module clk_div(input clk,
               input rst,
               input SW2,SW8,STEP
               output reg [31:0]clkdiv,
               output Clk_CPU
               );
    always @ (posedge clk or posedge rst) begin
        if (???) clkdiv <= ? ; else clkdiv <= ??????????; end
    assign Clk_CPU=(?)? STEP:(???) ? clkdiv[24] : clkdiv[2];
endmodule
```

//主板时钟
//复位信号
//CPU时钟切换
//32位计数分频输出
//CPU时钟输出

通过32位计数，每个时钟加一；当达到计数值时，输出时钟翻转一次

八分频
四分频
二分频

clk	clkdiv	Clkdiv[2]	Clkdiv[1]	Clkdiv[0]
	0	0	0	0
	1	0	0	1
	2	0	1	0
	3	0	1	1
	4	1	0	0
	5	1	0	1
	6	1	1	0
	7	1	1	1



开关去抖动模块：SAnti_jitter

□ 开关机械抖动消除模块

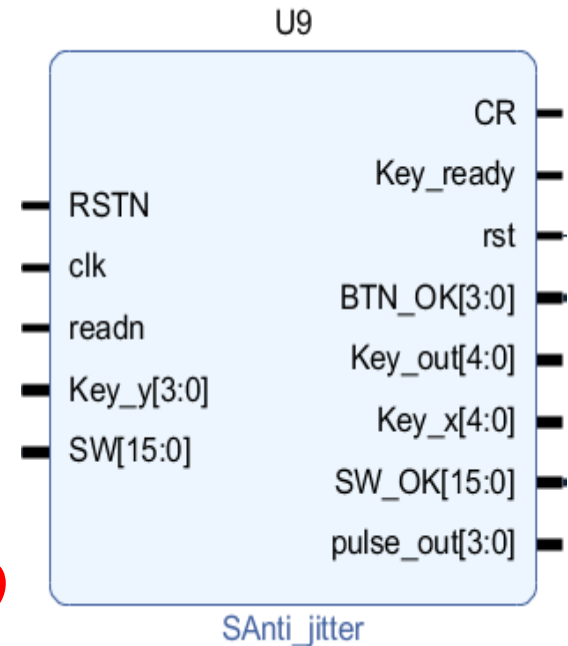
- 用于消除开关和按钮输入信号的机械抖动
 - CPU、IO和存储器等

□ 基本功能

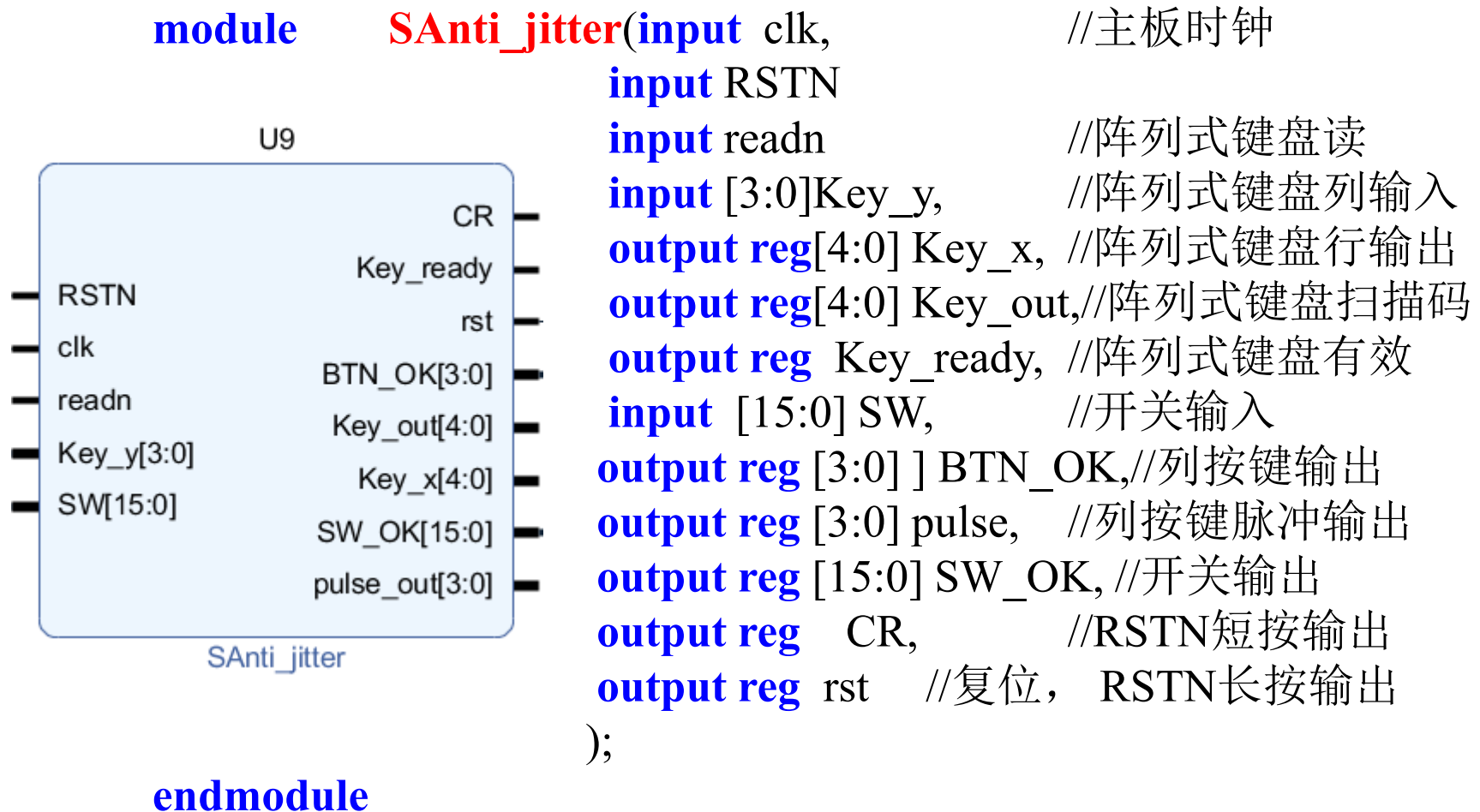
- 输入机械开关量
- 输出滤除机械抖动的逻辑值
 - 电平输出：button_out、SW_OK
 - 脉冲输出：button_pluse
- 逻辑实验模块

□ 本实验可自己设计或用IP 软核- U9

- 核调用模块SAnti_jitter.v
- 核接口信号模块(空文档)：SAnti_jitter.v



开关去抖动IP核端口 -Anti_jitter.v



SOC系统基本概念介绍

SOC Systemon Chip简介

此部分可以了解性介绍

◎ System on Chip (片上系统/系统级芯片)

☞ 从狭义角度讲

- ⊙ 是信息系统的芯片集成，或将系统集成在一块芯片上

☞ 从广义角度讲

- ⊙ SoC是一个微小型系统

◎ SoC技术

☞ SoC是ASIC(Application Specific Integrated Circuits)设计方法学中的新技术

☞ 不是简单芯片(IP Core)功能叠加，而是从整个系统的功能和性能出发，用软硬结合的设计和验证方法，利用IP复用及深亚微米技术，在一个芯片上实现复杂或专用的功能

☞ FPGA上可以实现SOC原型

- ⊙ 计算机专业实现体系结构上的设计与优化
- ⊙ 成熟后由微电子实现底层优化(网线层或腌膜层)
- ⊙ 大批量实现可用做成ASIC

SOC三要素

◎ IP核集成

- ⌚ IP(Intellectual Property)

 - ⊙ (集成电路)知识产权

- ⌚ IP核是具有复杂系统功能的能够独立出售的VLSI模块(硬件描述)

- ⌚ SOC由IP核组装成系统，而不是直接ASIC

◎ IP核复用

- ⌚ SoC中可以有多个MPU、MCU、DSP等或其复合的IP核

◎ IC工艺

- ⌚ 应采用深亚微米以上工艺技术;

SoC芯片设计中的IP模块

◎ IP是SOC的灵魂

- ⌚ SoC设计基础是IP (IntellectualProperty) 复用技术
- ⌚ 已有的IC电路以模块的形式呈现
- ⌚ 在SoC芯片设计中调用
- ⌚ 这些可以被重复使用的IC模块就叫做IP模块(核)
 - ⊙ 一种预先设计好，已经过验证，具有某种确定功能的集成电路、器件或部件

◎ 三种不同形式IP核

- ⌚ 软IP核(soft IP Core)
- ⌚ 固IP核(firm IP core)
- ⌚ 硬IP核(hard IP Core)

SOC设计方法和流程

◎ 系统集成方法

- ㊦ 系统集成法
- ㊦ 部分集成法
- ㊦ IP集成法

◎ 流程

- ㊦ 功能设计
- ㊦ 设计描述和行为级验证

- 依据功能将SOC划分为若干功能模块，并决定实现这些功能将要使用的IP核。
- 设计
 - ◆ 用VHDL或Verilog等硬件描述语言实现各模块的设计。
- 仿真
 - ◆ 利用VHDL或Verilog的电路仿真器，对设计进行功能验证(function simulation或行为验证behavioral simulation)

SoC设计流程-续

㊦ 逻辑综合

- ⊙ 使用逻辑综合工具(synthesizer)进行综合。
- ⊙ 选择适当的逻辑器件库(logic cell library)，作为合成逻辑电路时的参考依据。
- ⊙ 逻辑综合得到门级网表(**课程实验用的核**)

㊦ 门级验证

- ⊙ 寄存器传输级验证(**数字逻辑课知识**)
- ⊙ 确认经综合后的电路是否符合功能需求
- ⊙ 一般利用门电路级验证工具完成。
- ⊙ 此阶段仿真需要考虑门电路的延迟。

SoC设计流程-续

以下是微电子专业做的

☞ 布局和布线

- ⊙ 布局指将设计好的功能模块合理地安排在芯片上，规划好它们的位置。
- ⊙ 布线则指完成各模块之间互连的连线。
- ⊙ 各模块之间的连线，产生的延迟会严重影响SOC的性能

☞ 电路仿真

☞ 基于最终时序的版图后仿真

☞ 确认在考虑门电路延迟和连线延迟的条件之下，电路能否正常运作

☞ 一般是使用SDF（标准延时）文件来输入延时信息

☞ 仿真时间将数倍于先前的仿真。

SOC设计使用的主要语言

◎ VHDL

⌚ 略

◎ VerilogHDL

⌚ 略

◎ System C

⌚ C++: 专用于SOC设计与建模

⌚ 建模元素: 模块、进程、时钟、事件

◎ 其它.....

-
- **任务：**通过第三方IP和已有IP模块建立CPU测试环境（SOC系统的集成实现）-----采用Block Design模块化方式实现

设计工程：OExp02-IP2SOC

◎ 建立CPU调试、测试和应用环境

☞ 顶层用逻辑图实现，调用IP核模块

- ⊙ 模块名：CSSTE.bd(Computer system-single cycle processor test environment)

◎ SOC集成技术实现系统构架

☞ 用实验一设计的模块和第三方IP核

- ⊙ CPU (第三方IP核): U1
- ⊙ ROM (Vivado构建核): U2
- ⊙ RAM (Vivado构建IP核): U3
- ⊙ 总线(第三方IP核): U4
- ⊙ 八数据通路模块(Multi_8CH32): U5
- ⊙ 七段显示模块(第三方IP核): U6
- ⊙ LED显示模块(第三方IP核): U7
- ⊙ 通用分频模块(clk_div): U8
- ⊙ 开关去抖模块(IP核): U9
- ⊙ 定时计数器 (第三方IP): U10
- ⊙ VGA显示模块 (IP核): U11

设计要点

建立SOC应用工程

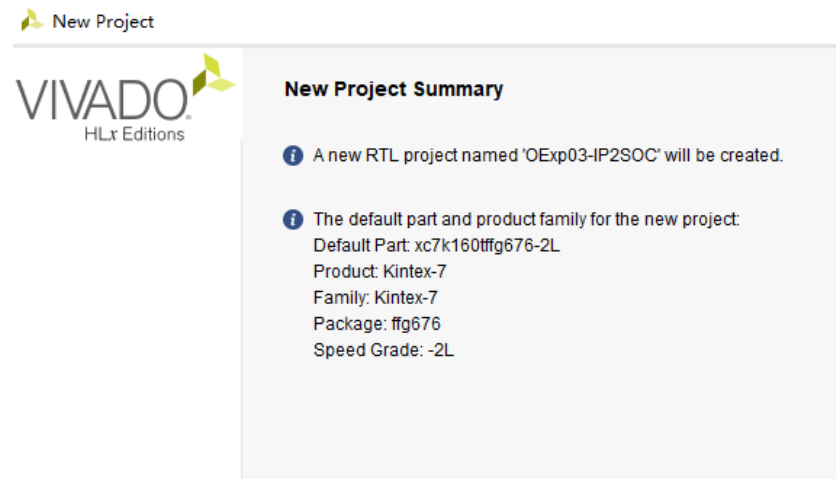
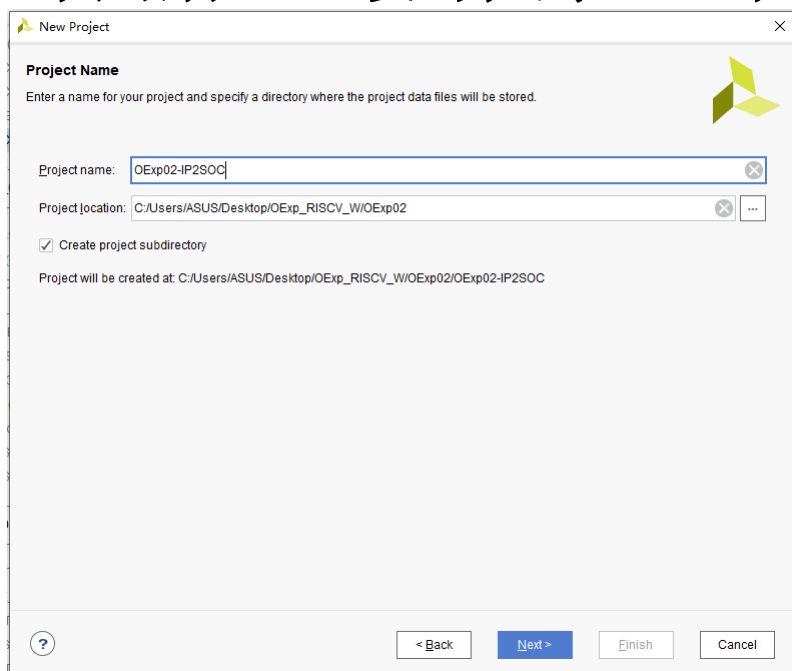
新建工程OExp02-IP2SOC

建立SOC应用工程

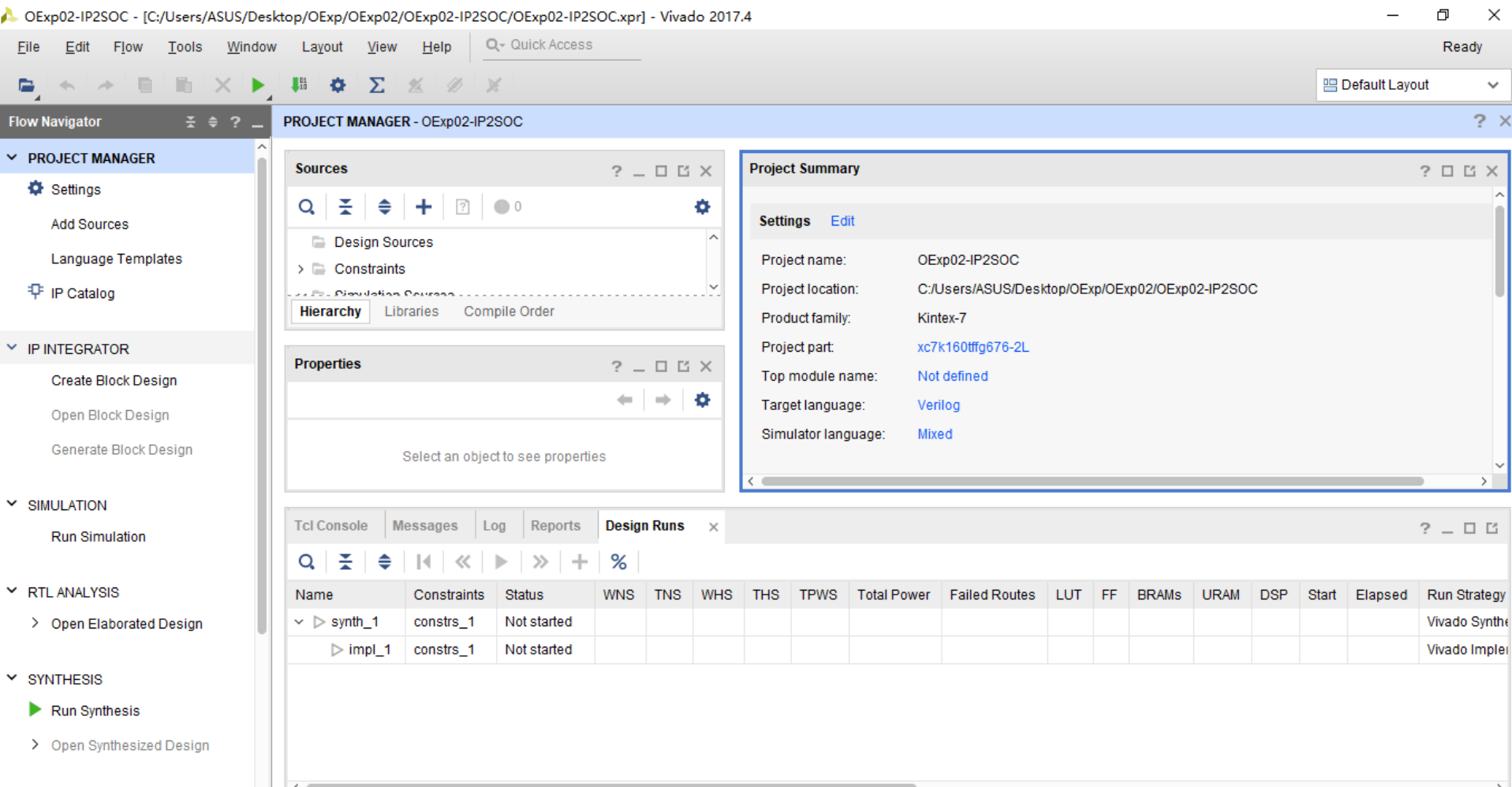
□ 用Vivado新建SOC应用工程

- 双击桌面上“Vivado”图标，启动Vivado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：**OExp02-IP2SOC**
- 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

□ 单周期CPU设计共享此工程



SOC工程模板



设计要点

导入各个IP模块的封装文件到当前工程目录
添加各个IP模块的实际路径到当前工程

› OExp02 › IP

名称	修改日期	类型	大小
Logical	2021/3/2 18:51	文件夹	
MUX	2021/3/2 18:58	文件夹	
scpu	2021/3/3 13:39	文件夹	
Supplementary	2021/3/10 9:40	文件夹	

IP库里边包含基本逻辑模块，多路器，CPU及外设模块，具体添加方法参见Lab0，若有自己设计的模块也请一并添加到IP库，方便后续管理

设计要点

导入用于VGA静态显示的初始化信息文件到工程，并将提供的.mem文件拷贝到D盘下

```
Memory File (2)
  font_8x16.mem
  vga_debugger.mem

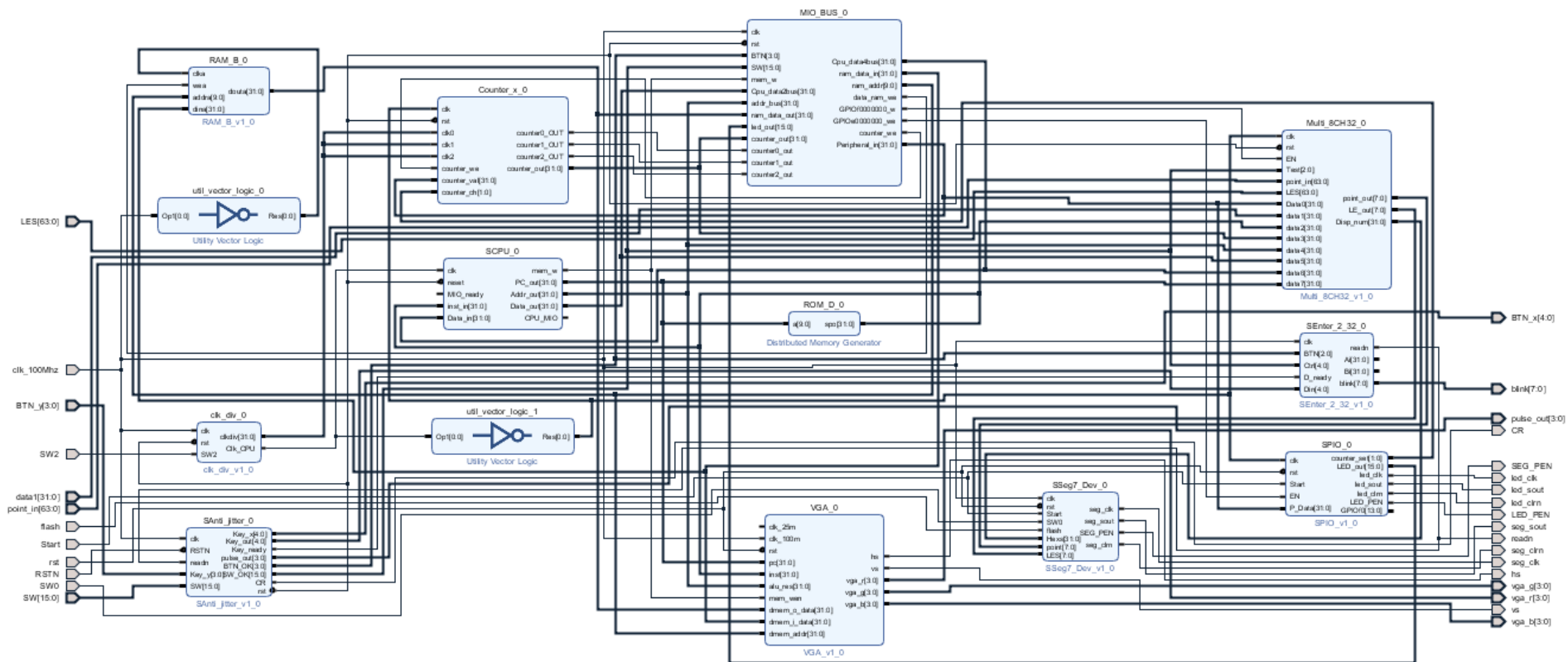
initial $readmemh("D://vga_debugger.mem", display_data);
initial $readmemh("D://font_8x16.mem", fonts_data);
```

也可自己更改路径，放入实际的路径之下

若更改VGA的端口及其他功能，请重新进行IP封装，并在BD工程中进行IP状态更新

原理图输入SoC顶层逻辑

SOC顶层逻辑图



本图仅供概览，详细连接图请参见完整版pdf文档

建立SOC原理图输入模板（顶层模块）

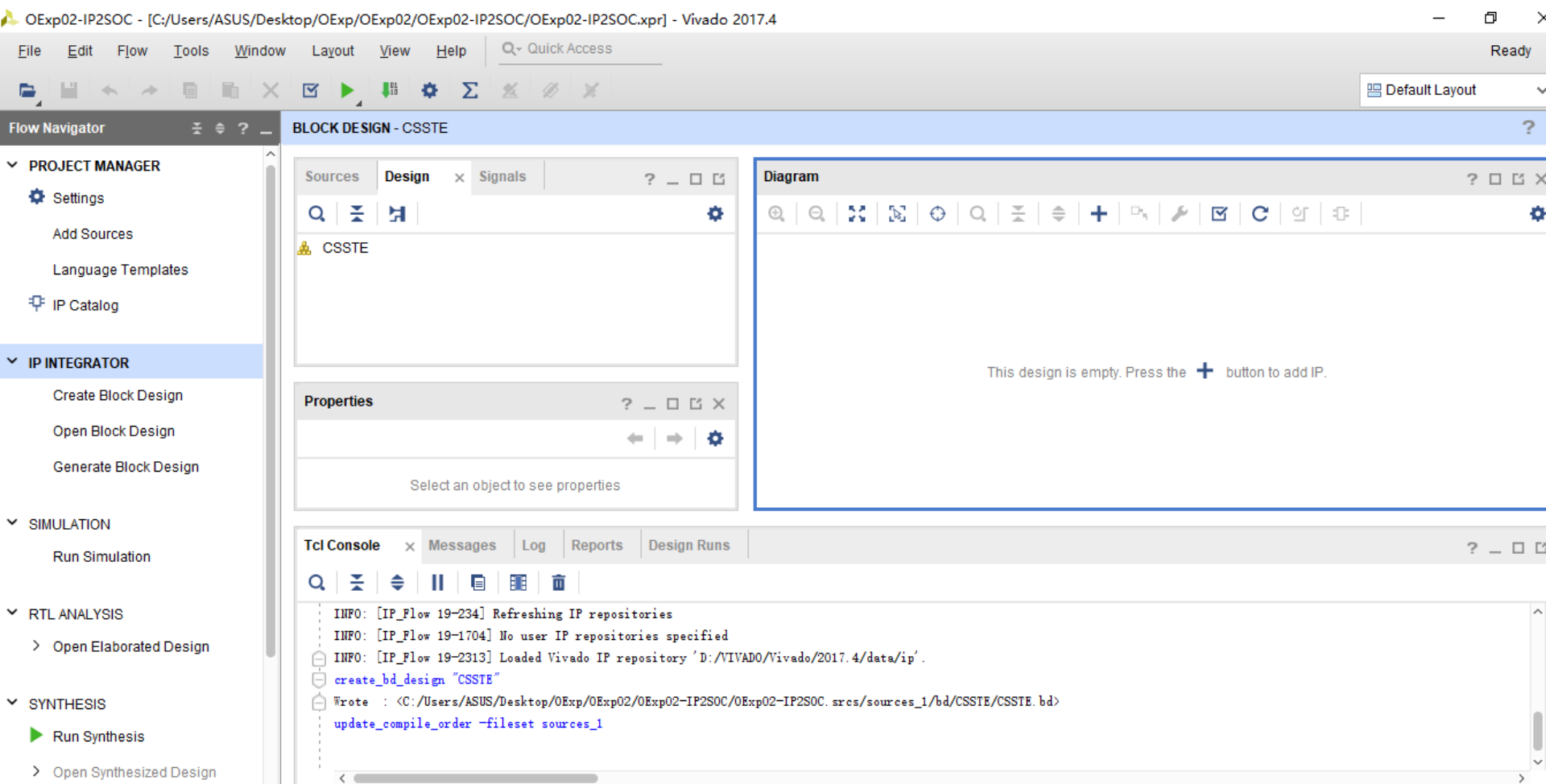
□ 建立顶层模块

- 单击 Create Block Design建立BD工程
- 工程名CSSTE.bd

□ 输入SOC顶层原理图

- 根据SOC顶层逻辑图连接各个模块

原理图输入窗口与环境



放置了CPU模块

BLOCK DESIGN - CSSTE *

Sources Design x Signals ? _ □ □

CSSTE

> U1 (SCPU:1.0)

Block Properties

U1

Name: U1

General Properties IP

Tcl Console x Messages Log

Diagram

Designer Assistance available. [Run Connection Automation](#)

U1

MIO_ready CPU_MIO

clk MemRW

rst Addr_out[31:0]

Data_in[31:0] Data_out[31:0]

inst_in[31:0] PC_out[31:0]

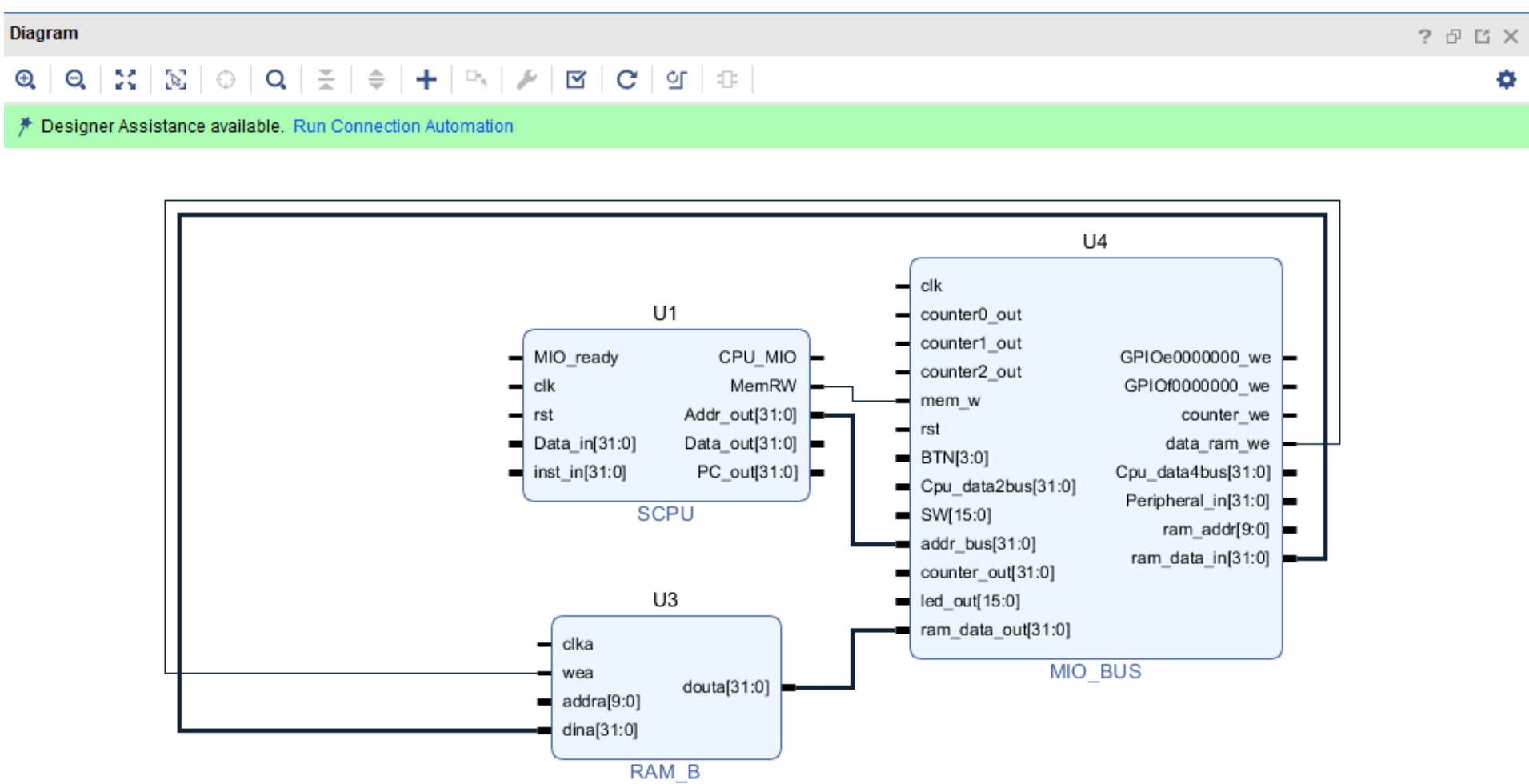
SCPU

Search: Q- |

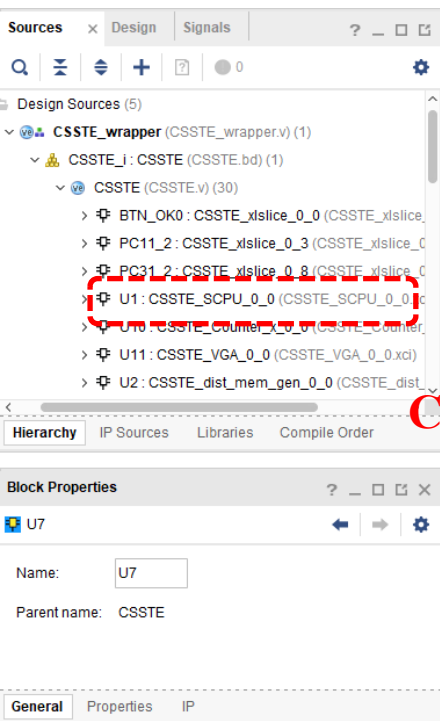
- 1G/2.5G Ethernet PCS/PMA or SGMII
- 2D Graphics Accelerator Bit Block Transfer
- 3GPP LTE Channel Estimator
- 3GPP LTE MIMO Decoder
- 3GPP LTE MIMO Encoder
- 3GPPLTE Turbo Encoder
- 3GPP Mixed Mode Turbo Decoder
- 3GPP Turbo Encoder
- 7 Series Integrated Block for PCI Express

ENTER to select, ESC to cancel, Ctrl+Q for IP details

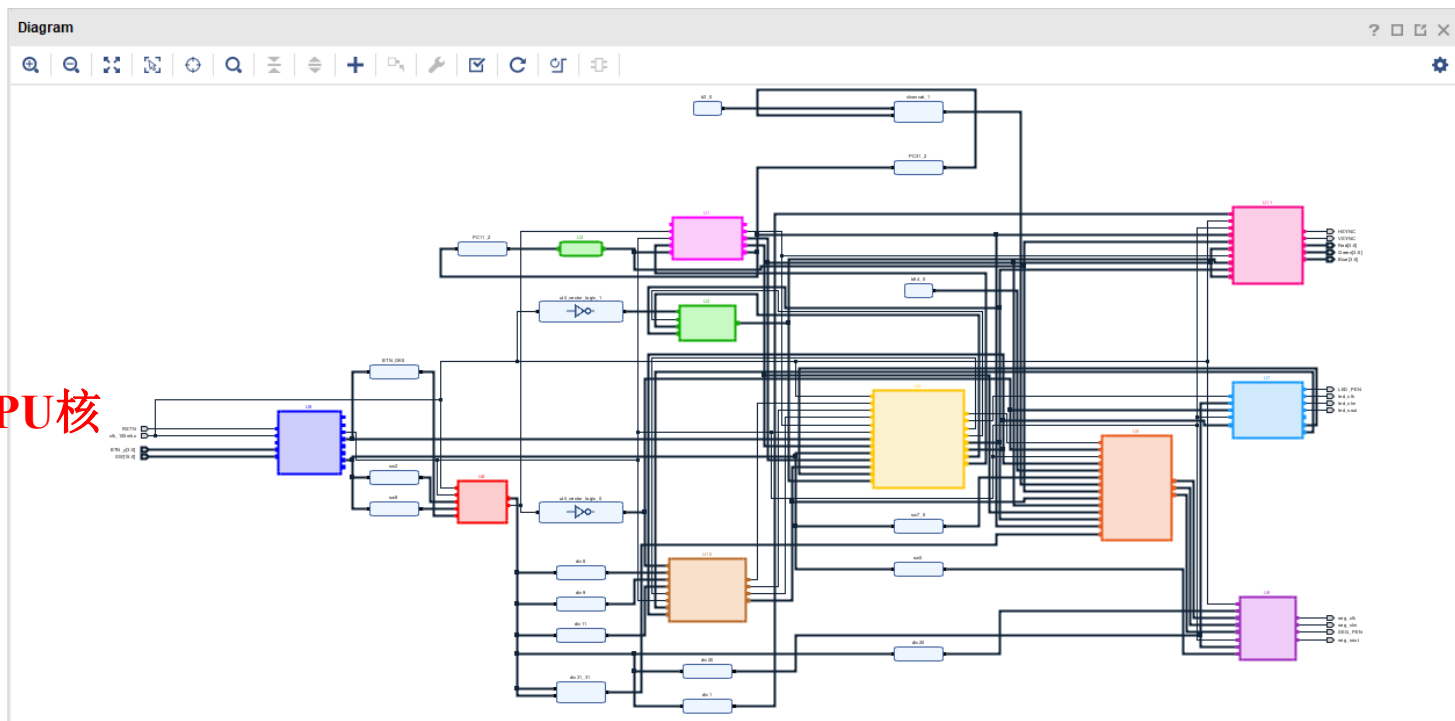
输入存储器和总线模块并连接若干信号



完成输入后第二层模块层次关系



CPU核



原理图检查

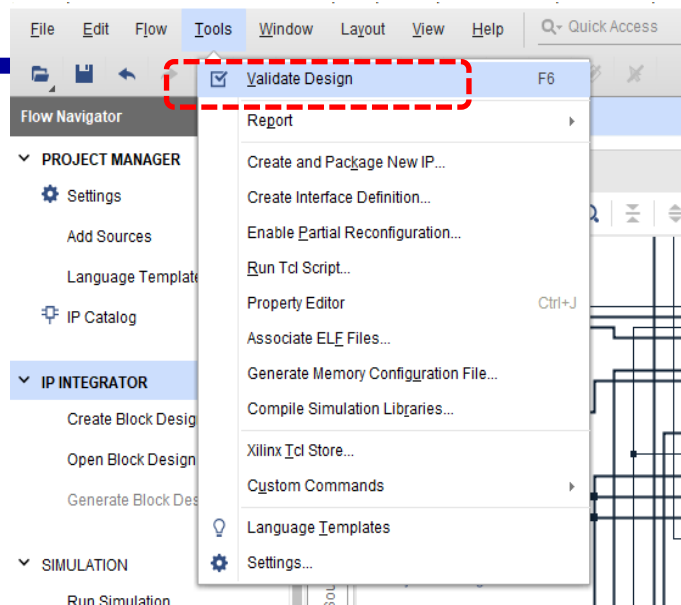
□ 模块信号连接检测

- 在菜单栏：

选择Tools→Validate Design

- 编辑器自动检查原理图信号连接

- 不会检查电路逻辑功能
- 仅检查信号连接是否满足规则
- 特别注意总线连接（利用Slice\Concat IP）
 - 位宽
 - 顺序

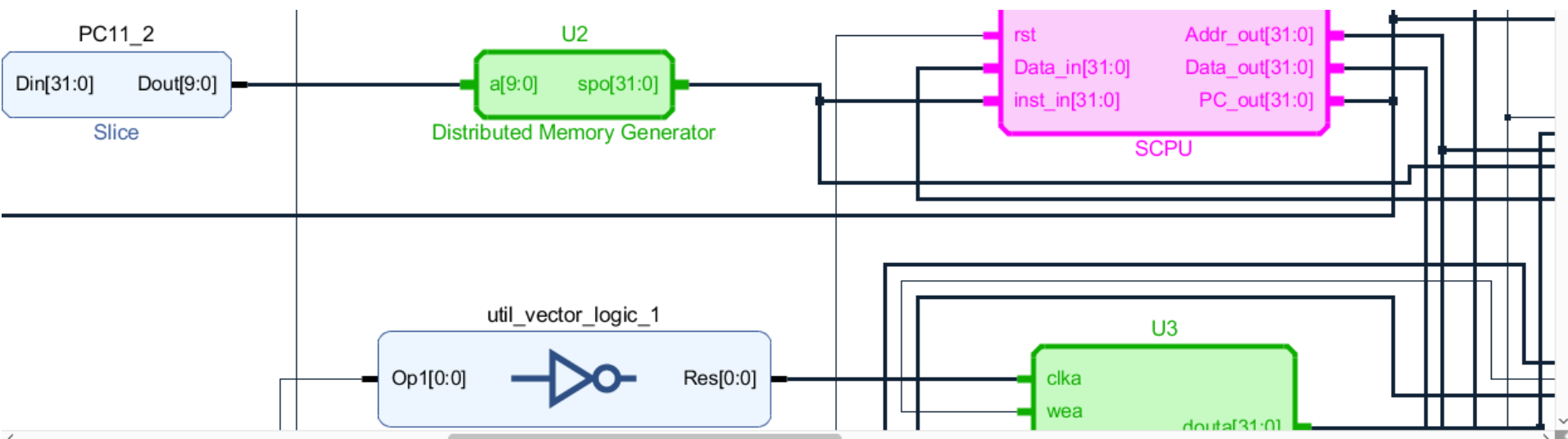
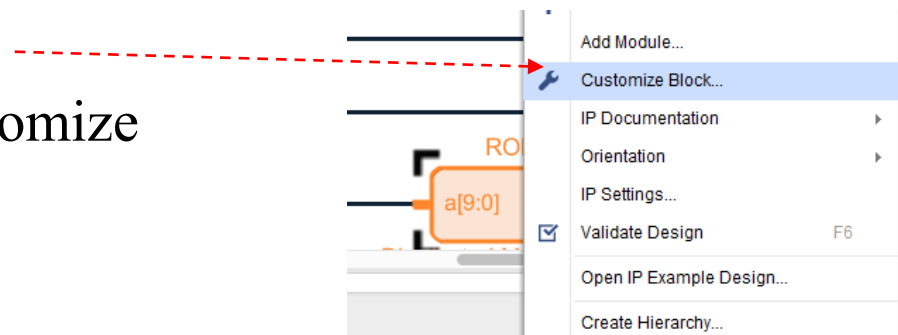


存储器IP核重新初始化

ROM_B重新初始化

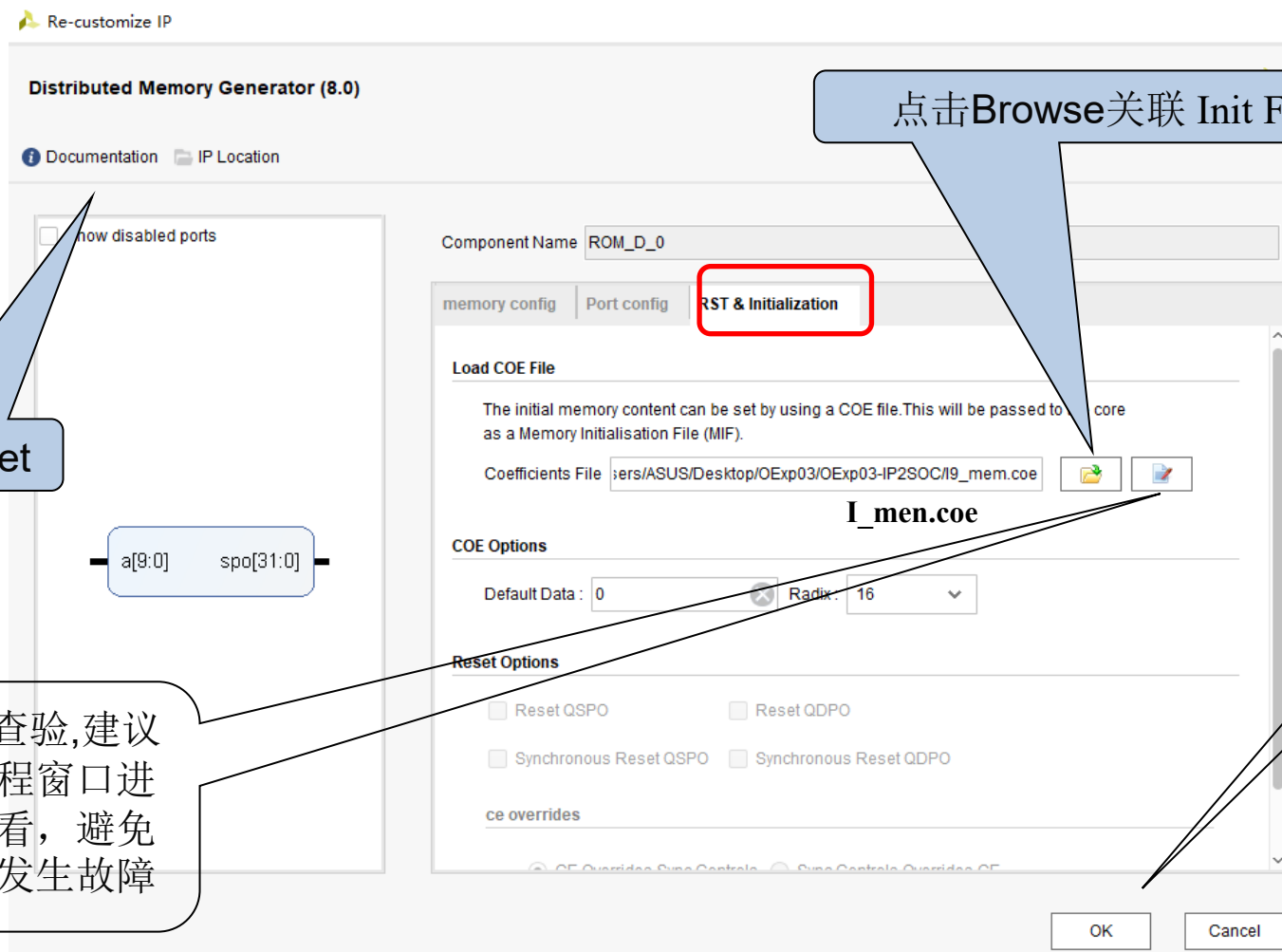
□ 用 *I9_men.coe* 初始化ROM

- 双击U2进入核管理向导
 - 也可以选中U2模块，右键customize
- 进入核管理窗口



关联初始化文件并生成ROMIP核

- ❑ 点击“Browse...”选择初始化关联文件（**I_men.coe**）
- ❑ 其余不用修改，点击**Generate** 重新生成ROM核



ROM初始化文件：.coe

□ ROM.coe格式

- 可以用Vivado打开编辑，也可以用普通文本编辑工具
- 格式如下：
 - 第一行：说明是初始化参数向量采用16进制（也可以2进制）
 - 第二行：初始化向量名
 - 第三行开始：初始化向量元素，用逗号“，”分隔，分号结束
 - 文件头、尾部可以用“#”号加注释，中间不可以

```
memory_initialization_radix=16;  
memory_initialization_vector=  
00100093, 00102133, 002101B3, 00218233, 003202B3, 00428333,  
005303B3, 00638433, 007404B3, 00848533, 009505B3, 00A58633,  
00B606B3, 00C68733, 00D707B3, 00E78833, 00F808B3, 01088933,  
011909B3, 01298A33, 013A0AB3, 014A8B33, 015B0BB3, 016B8C33,  
017C0CB3, 018C8D33, 019D0DB3, 01AD8E33, 01BE0EB3, 01CE8F33,  
01DF0FB3, 0xF80002E3;
```

□ 以上数据一段简单的指令测试

- CPU仿真用上述数据
- 下载时用I_mem.coe

简单的指令测试(暂时只要了解)

#baseAddr 0000

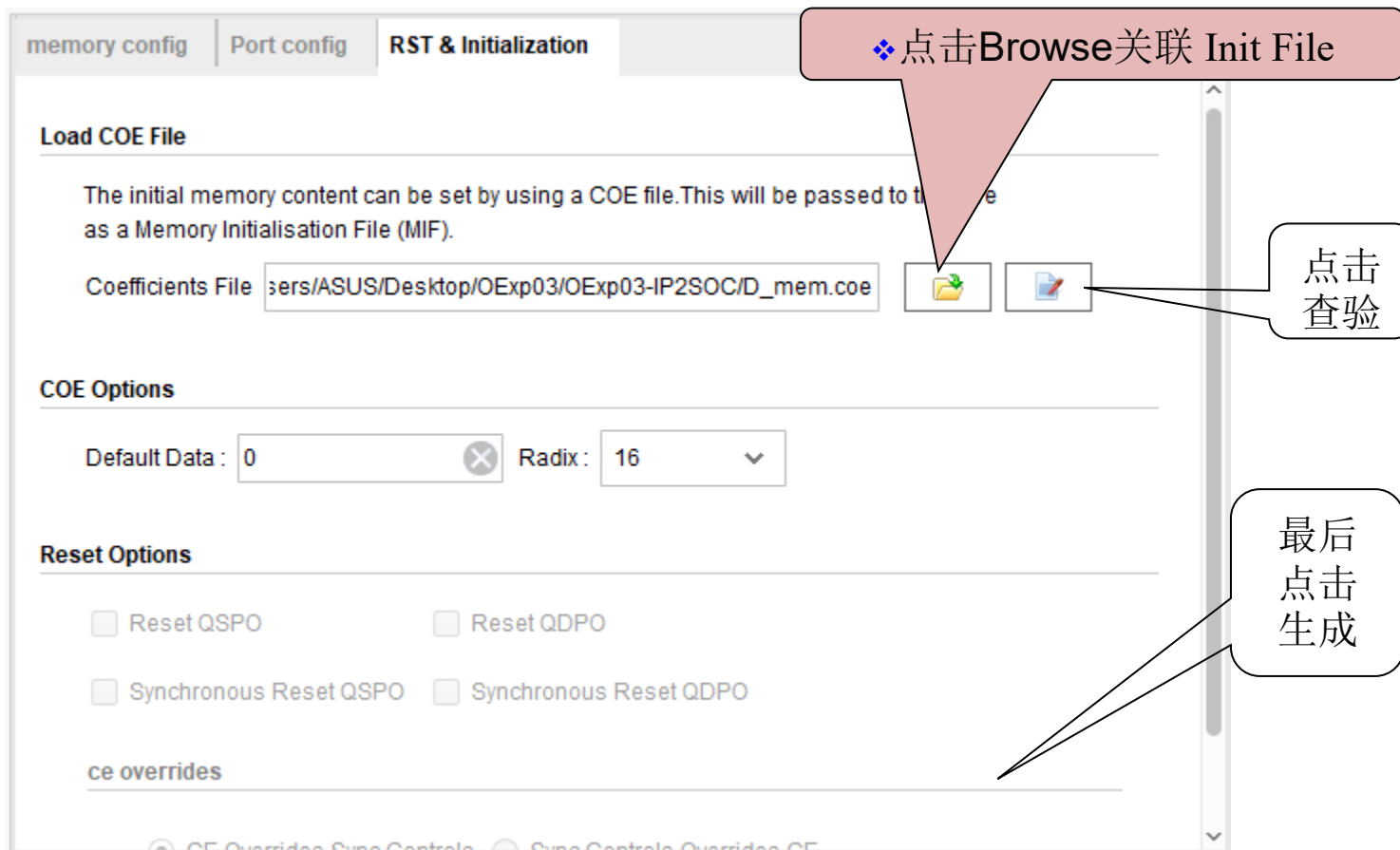
```
loop:   addi x1,x0,1;      //x1=00000001
        slt x2,x0,x1;      //x2=00000001
        add x3,x2,x2;      //x3=00000002
        add x4,x3,x2;      //x4=00000003
        add x5,x4,x3;      //x5=00000005
        add x6,x5,x4;      //x6=00000008
        add x7,x6,x5;      //x7=0000000d
        add x8,x7,x6;      //x8=00000015
        add x9,x8,x7;      //x9=00000022
        add x10,x9,x8;     //x10=00000037
        add x11,x10,x9;    //x11=00000059
        add x12,x11,x10;   //x12=00000090
        add x13,x12,x11;   //x13=000000E9
        add x14,x13,x12;   //x14=00000179
        add x15,x14,x13;   //x15=00000262
```

```
add r16,r15,r14; //r16=000003DB
add r17,r16,r15; //r17=000006D3
add r18,r17,r16; //r18=00000A18
add r19,r18,r17; //r19=000010EB
add r20,r19,r18; //r20=00001B03
add r21,r20,r19; //r21=00003bEE
add r22,r21,r20; //r22=000046F1
add r23,r22,r21; //r23=000080DF
add r24,r23,r22; //r24=0000C9D0
add r25,r24,r23; //r25=00014AAF
add r26,r25,r24; //r26=0001947F
add r27,r26,r25; //r27=0012DF2E
add r28,r27,r26; //r28=001473AD
add r29,r28,r27; //r29=002752DB
add r30,r29,r28; //r30=003BC688
add r31,r30,r29; //r31=00621963
beq x0,x0,loop;
```

RAM_B初始化

□ 与ROM同样方法进入核管理向导，关联初始化文件并生成RAM IP核

- 点击“Browse...”选择初始化关联文件（**D_mem.coe**）
- 其余不用修改，**点击Generate**重新生成ROM核



RAM初始数据--.coe

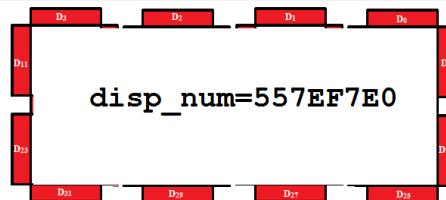
□ D_mem.coe初始数据

```
memory_initialization_radix=16;  
memory_initialization_vector=
```

红色数据是LED图形

```
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;
```

□ 下载和仿真均可用



557EF7E0映射为图形显示 4位七段码的外围边框点亮

RISC-V在线指令集翻译器

<https://venus.cs61c.org/>

Venus

Editor

Simulator

Chocopy


Terminal

Files

URL

Wiki

JVM



```
Venus Web Terminal
Wed Jan 27 2021 20:29:29 GMT+0800 (中国标准时间)
Enter "help" for more information.

[user@venus] /#
```

Settings

General

Calling Convention

Tracer

Packages

Simulator Default Args

Text Start

0x00000000

Max History

-1

Save on Close

Save on Close

Aligned Addressing

Force Aligned Addressing?

Mutable Text

Mutable Text?

Only Ecall Exit

Only Ecall Exit?

Default Reg States

Set Registers on Init?

Allow Access

Allow Access Between Stack and Heap?

Max number of steps: (Negative means ignored)

-1

☐ Dark Mode

RISCV在线指令集翻译器

<https://venus.cs61c.org/>

Venus

Editor

Simulator

Chocopy

单击**Editor**进入文本边界模式，输入汇编程序（**注意用#注释**）

```
1 loop:addi x1,x0,1    #x1=00000001
2 slt x2,x0,x1        #x2=00000001
3 add x3,x2,x2        #x3=00000002
4 add x4,x3,x2        #x4=00000003
5 add x5,x4,x3        #x5=00000005
6 add x6,x5,x4        #x6=00000008
7 add x7,x6,x5        #x7=0000000d
8 add x8,x7,x6        #x8=00000015
9 add x9,x8,x7        #x9=00000022
10 add x10,x9,x8      #x10=00000037
11 add x11,x10,x9     #x11=00000059
12 add x12,x11,x10    #x12=00000090
13 add x13,x12,x11    #x13=000000E9
14 add x14,x13,x12    #x14=00000179
15 add x15,x14,x13    #x15=00000262
16 add x16,x15,x14    #x16=000003DB
17 add x17,x16,x15    #x17=000006D3
18 add x18,x17,x16    #x18=00000A18
19 add x19,x18,x17    #x19=000010EB
20 add x20,x19,x18    #x20=00001B03
21 add x21,x20,x19    #x21=00003bEE
22 add x22,x21,x20    #x22=000046F1
23 add x23,x22,x21    #x23=000080DF
24 add x24,x23,x22    #x24=0000C9D0
25 add x25,x24,x23    #x25=00014AAF
26 add x26,x25,x24    #x26=0001947F
27 add x27,x26,x25    #x27=0012DF2E
28 add x28,x27,x26    #x28=001473AD
29 add x29,x28,x27    #x29=002752DB
30 add x30,x29,x28    #x30=003BC688
31 add x31,x30,x29    #x31=00621963
32 beq x0,x0,loop     #pc---->loop
```

RISCV在线指令集翻译器

<https://venus.cs61c.org/>

Venus Editor **Simulator** ChoCopy

Assemble & Simulate from Editor

Cancel

PC	Machine Code	Basic Code	Original Code
0x0	0x00100093	addi x1 x0 1	addi x1,x0,1 #x1=00000001
0x4	0x00102133	slt x2 x0 x1	slt x2,x0,x1 #x2=00000001
0x8	0x002101B3	add x3 x2 x2	add x3,x2,x2 #x3=00000002
0xc	0x00218233	add x4 x3 x2	add x4,x3,x2 #x4=00000003
0x10	0x003202B3	add x5 x4 x3	add x5,x4,x3 #x5=00000005
0x14	0x00428333	add x6 x5 x4	add x6,x5,x4 #x6=00000008
0x18	0x005303B3	add x7 x6 x5	add x7,x6,x5 #x7=0000000d
0x1c	0x00638433	add x8 x7 x6	add x8,x7,x6 #x8=00000015
0x20	0x007404B3	add x9 x8 x7	add x9,x8,x7 #x9=00000022
0x24	0x00848533	add x10 x9 x8	add x10,x9,x8 #x10=00000037
0x28	0x009505B3	add x11 x10 x9	add x11,x10,x9 #x11=00000059
0x2c	0x00A58633	add x12 x11 x10	add x12,x11,x10 #x12=00000090
0x30	0x00B606B3	add x13 x12 x11	add x13,x12,x11 #x13=000000E9

Copy! Download! Clear!

console output

Registers		Memory	Cache
Integer (R)		Floating (F)	
zero	0x00000000		
ra (x1)	0x00000001		
sp (x2)	0x00000001		
gp (x3)	0x00000002		
tp (x4)	0x00000003		
t0 (x5)	0x00000005		
t1 (x6)	0x00000008		
t2 (x7)	0x0000000D		
s0 (x8)	0x00000015		
s1 (x9)	0x00000022		
a0 (x10)	0x00000037		
a1 (x11)	0x00000059		

点击Simulator---

→Assemble&Simulate from Editor

RISCV在线指令集翻译器

<https://venus.cs61c.org/>

VenusEditorSimulatorChocopy

RunStepPrevResetDumpTrace

PC	Machine Code	Basic Code	Original Code
0x0	0x00100093	addi x1 x0 1	loop:addi x1,x0,1 #x1=00000001
0x4	0x00102133	slt x2 x0 x1	slt x2,x0,x1 #x2=00000001
0x8	0x002101B3	add x3 x2 x2	add x3,x2,x2 #x3=00000002
0xc	0x00218233	add x4 x3 x2	add x4,x3,x2 #x4=00000003
0x10	0x003202B3	add x5 x4 x3	add x5,x4,x3 #x5=00000005
0x14	0x00428333	add x6 x5 x4	add x6,x5,x4 #x6=00000008
0x18	0x005303B3	add x7 x6 x5	add x7,x6,x5 #x7=0000000d
0x1c	0x00638433	add x8 x7 x6	add x8,x7,x6 #x8=00000015
0x20	0x007404B3	add x9 x8 x7	add x9,x8,x7 #x9=00000022
0x24	0x00848533	add x10 x9 x8	add x10,x9,x8 #x10=00000037
0x28	0x009505B3	add x11 x10 x9	add x11,x10,x9 #x11=00000059
0x2c	0x00A58633	add x12 x11 x10	add x12,x11,x10 #x12=00000090
0x30	0x00B606B3	add x13 x12 x11	add x13,x12,x11 #x13=000000E9

Copy!Download!Clear!

RegistersMemoryCache

Integer (R)Floating (F)

zero

0x00000000

ra (x1)

0x00000001

sp (x2)

0x00000001

gp (x3)

0x00000002

tp (x4)

0x00000003

t0 (x5)

0x00000005

t1 (x6)

0x00000008

t2 (x7)

0x0000000D

s0 (x8)

0x00000015

s1 (x9)

0x00000022

a0 (x10)

0x00000037

a1 (x11)

0x00000059

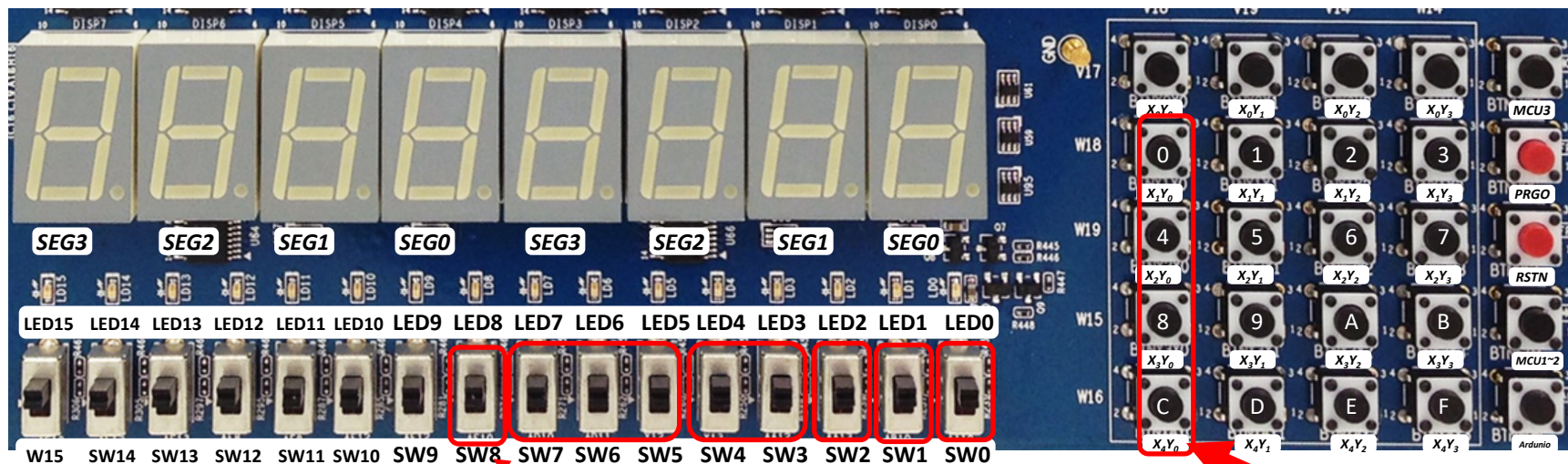
Display SettingsHex

点击左侧全速运行，单步，追踪等功能键，右侧可观察寄存器状态

SoC物理调试验证

-本实验不需要测试，仅用DEMO程序验证功能

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[8][2]=CPU单步时钟选择

BTN_y[0]做单步STEP输入

下载验证SoC

□ 非IP核仿真

- 对自己设计的模块做时序仿真
- 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
 - 用测试代码替换I_mem.coe数据*

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[8]SW[2]	00	CPU全速时钟 100MHZ
SW[8]SW[2]	01	CPU自动单步时钟(2*24分频)
SW[8]SW[2]	1X	CPU手动单步时钟(按键BTN_OK[0])
SW[7:5]	011	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)

Key_y[0]

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[8]SW[2]	00	CPU全速时钟 100MHZ
SW[8]SW[2]	01	CPU自动单步时钟(2*24分频)
SW[8]SW[2]	1X	CPU手动单步时钟(按键BTN_OK[0])
SW[7:5]	001	CPU指令字地址PC_out[31:2]
SW[7:5]	010	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

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[8]SW[2]	00	CPU全速时钟 100MHZ
SW[8]SW[2]	01	CPU自动单步时钟(2*24分频)
SW[8]SW[2]	1X	CPU手动单步时钟(按键BTN_OK[0])

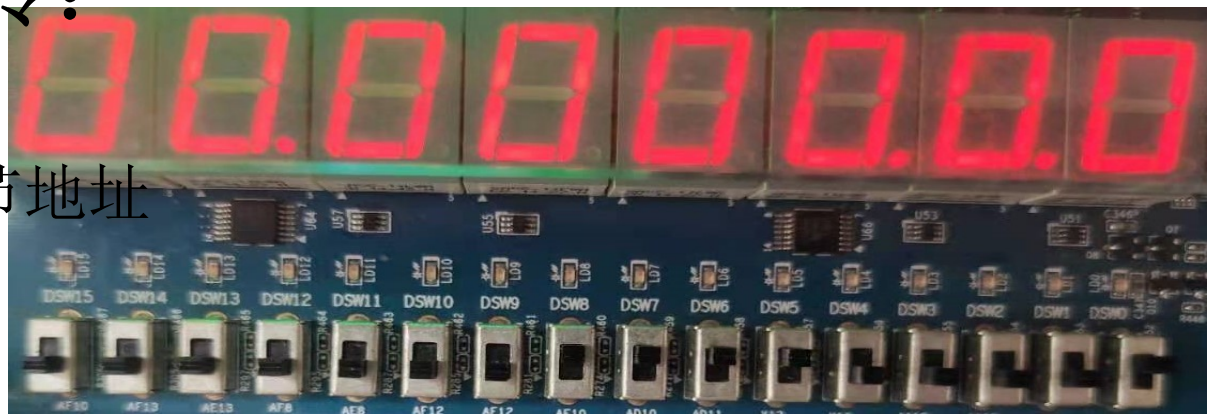
- 液晶显示屏显示PC,inst,等信息

实验效果图

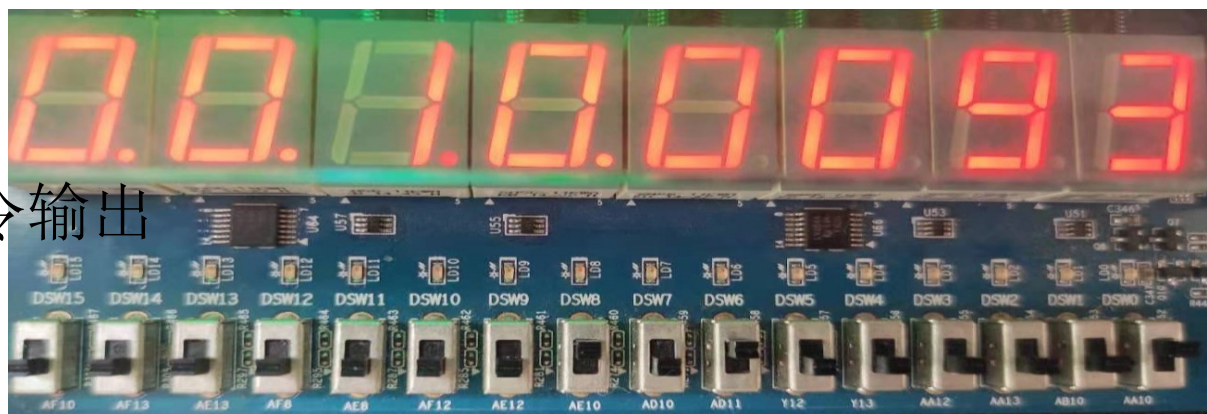
□ 数码管的显示效果图

□ CPU运行第一条指令:

- $SW[7:5] = 111$;
- 选通通道data7指令字节地址
- $PC_out = 00000000$



- $SW[7:5] = 010$;
- 选通通道data2ROM指令输出
- $inst = 00100093$



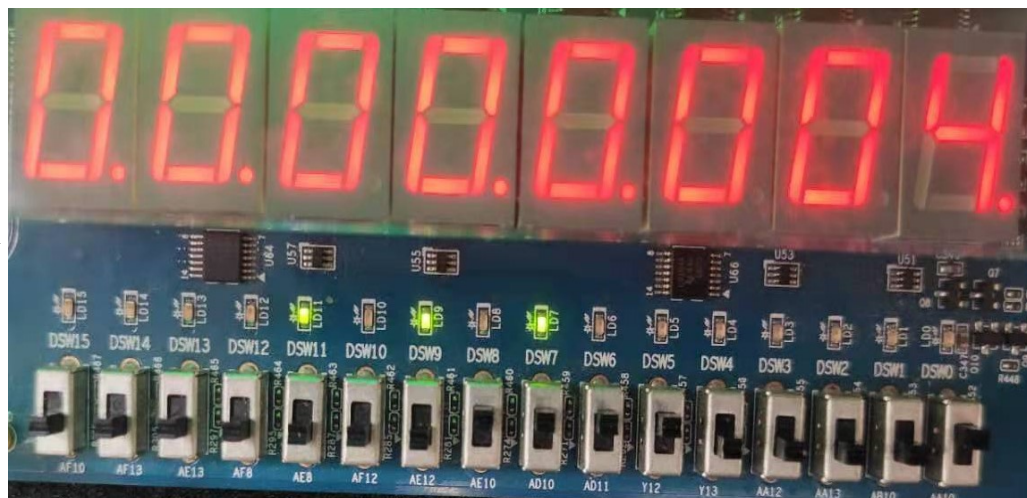
□ 其他通道请依据开关状态进行观察

实验效果图

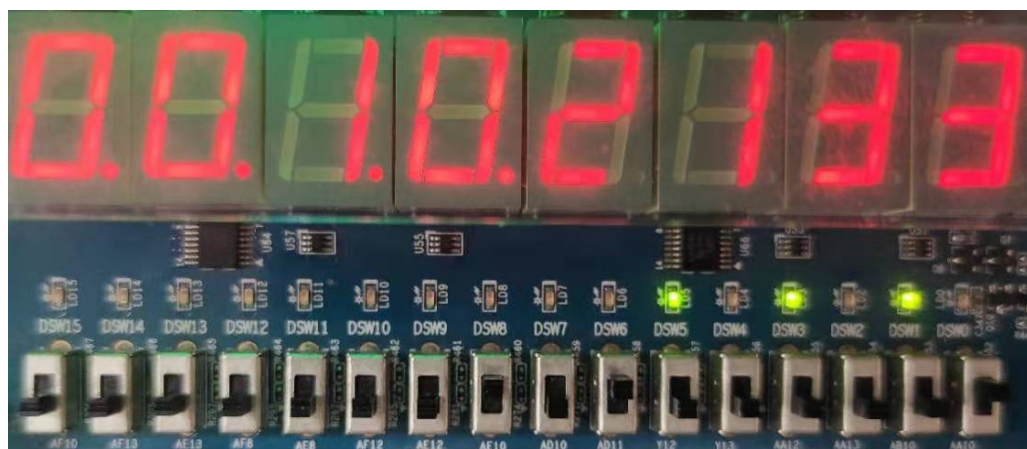
□ 数码管的显示效果图

□ CPU运行第二条指令：

- $SW[7:5] = 111$;
- 选通通道data7指令字节地址
- $PC_out = 00000004$



- $SW[7:5] = 010$;
- 选通通道data2ROM指令输出
- $inst = 00102133$



□ 其他通道请依据开关状态进行观察

实验效果图

□ VGA的调试演示效果图

□ CPU运行第一条指令: 0x00100093 addi x1 x0 1

- PC=00000000
- PC为取指令的地址
- inst=00100093
- Inst为指令的内容
- alu_res=00000001
- alu_res为ALU结果
- dmem_addr=alu_res
= 00000001
- dmem_addr为RAM
输入地址

```
RV32I Single Cycle CPU
pc: 00000000    inst: 00100093

x0: 00000000    ra: 00000000    sp: 00000000    gp: 00000000
t0: 00000000    t1: 00000000    t2: 00000000    s0: 00000000
a0: 00000000    a1: 00000000    a2: 00000000    a3: 00000000
a5: 00000000    a6: 00000000    a7: 00000000    s2: 00000000
s4: 00000000    s5: 00000000    s6: 00000000    s7: 00000000
s9: 00000000    s10: 00000000    s11: 00000000    t3: 00000000
t5: 00000000    t6: 00000000

rs1: 00    rs1_val: 00000000
rs2: 00    rs2_val: 00000000
rd: 00    reg_i_data: 00000000    reg_wen: 0

is_imm: 0    is_auiipc: 0    is_lui: 0    imm: 00000000
a_val: 00000000    b_val: 00000000    alu_ctrl: 0    cmp_ctrl: 0
alu_res: 00000001    cmp_res: 0

is_branch: 0    is_jal: 0    is_jalr: 0
do_branch: 0    pc_branch: 00000000

mem_wen: 0    mem_ren: 0
dmem_o_data: f0000000    dmem_i_data: 00000000    dmem_addr: 0

csr_wen: 0    csr_ind: 000    csr_ctrl: 0    csr_r_data: 00000000
mstatus: 00000000    mcause: 00000000    mepc: 00000000    mtval: 0
mtvec: 00000000    mie: 00000000    mip: 00000000
```


实验效果图

□ VGA的调试演示效果图

□ CPU运行第二条指令: 0x00102133 slt x2 x0 x1

- PC=00000004
- PC为取指令的地址
- inst=00102133
- Inst为指令的内容
- alu_res=00000001
- alu_res为ALU结果
- dmem_addr=alu_res
= 00000001

```
RV32I Single Cycle CPU
pc: 00000004    inst: 00102133

x0: 00000000    ra: 00000000    sp: 00000000    gp: 00000000    tp: 00000000
t0: 00000000    t1: 00000000    t2: 00000000    s0: 00000000    s1: 00000000
a0: 00000000    a1: 00000000    a2: 00000000    a3: 00000000    a4: 00000000
a5: 00000000    a6: 00000000    a7: 00000000    s2: 00000000    s3: 00000000
s4: 00000000    s5: 00000000    s6: 00000000    s7: 00000000    s8: 00000000
s9: 00000000    s10: 00000000    s11: 00000000    t3: 00000000    t4: 00000000
t5: 00000000    t6: 00000000

rs1: 00    rs1_val: 00000000
rs2: 00    rs2_val: 00000000
rd: 00    reg_i_data: 00000000    reg_wen: 0

is_imm: 0    is_auiipc: 0    is_lui: 0    imm: 00000000
a_val: 00000000    b_val: 00000000    alu_ctrl: 0    cmp_ctrl: 0
alu_res: 00000001    cmp_res: 0

is_branch: 0    is_jal: 0    is_jalr: 0
do_branch: 0    pc_branch: 00000000

mem_wen: 0    mem_ren: 0
dmem_o_data: 00000000    dmem_i_data: 00000001    dmem_addr: 00000001

csr_wen: 0    csr_ind: 000    csr_ctrl: 0    csr_r_data: 00000000
mstatus: 00000000    mcause: 00000000    mepc: 00000000    mtval: 00000000
mtvec: 00000000    mie: 00000000    mip: 00000000
```

 **END**