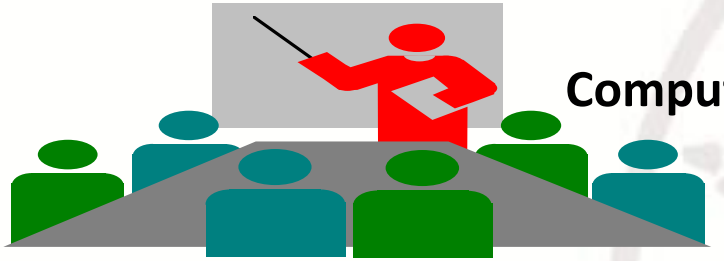


Computer Organization & Design
实验与课程设计

实验三

IP核集成SOC设计

--建立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

- 实验目的与实验环境
- 实验任务
- 实验原理
- 实验操作与实现

实验目的

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

3. Xilinx ISE14.4及以上开发工具

材料

无

Course Outline

- 实验目的与实验环境
- 实验任务
- 实验原理
- 实验操作与实现

实验任务

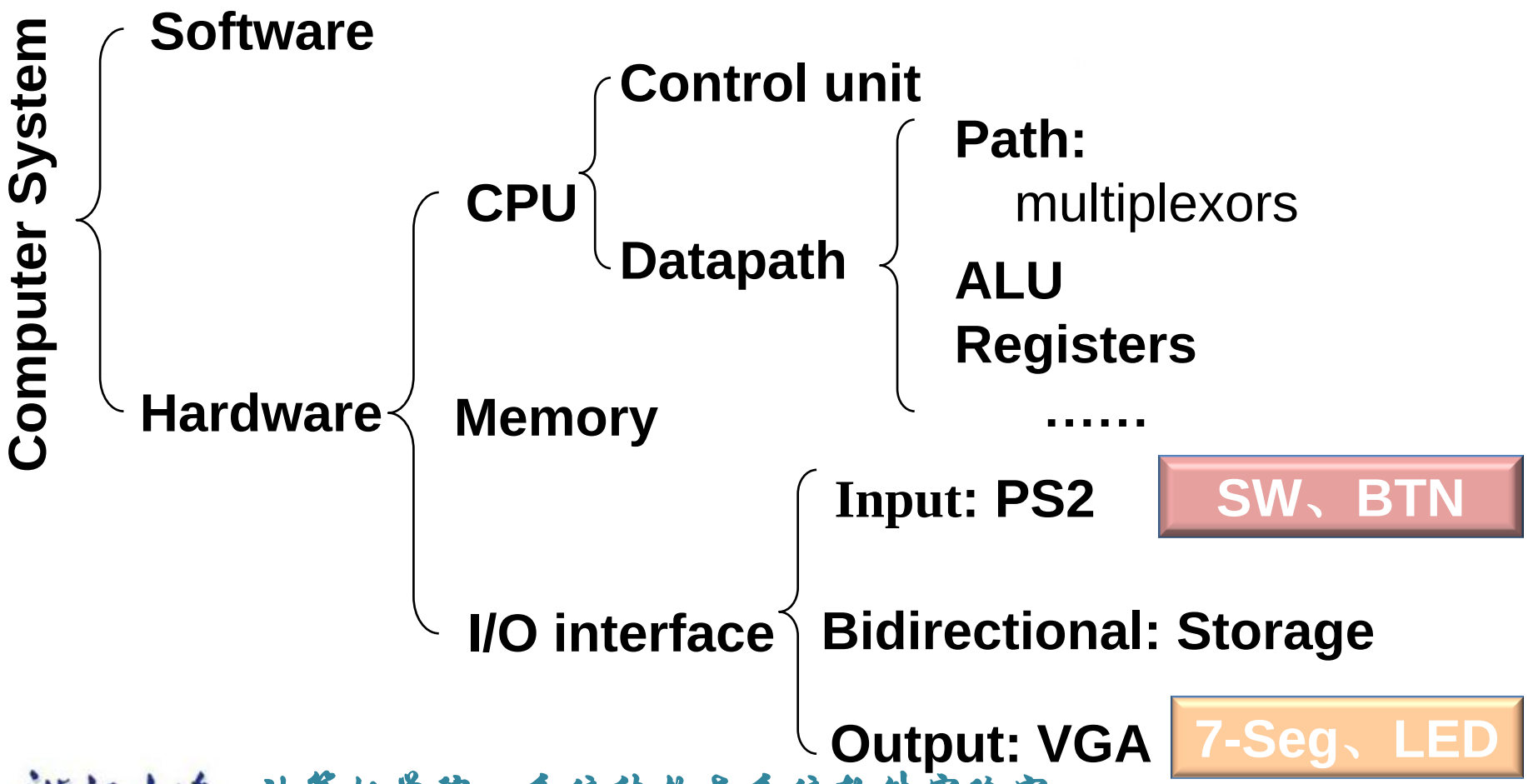
1. 分析基本和接口IP核
2. 设计存储器IP模块
3. 练习掌握IP核的使用方法
4. 选用第三方IP核和已有模块集成实现SOC
   - 此实验顶层用原理图设计实现

Course Outline

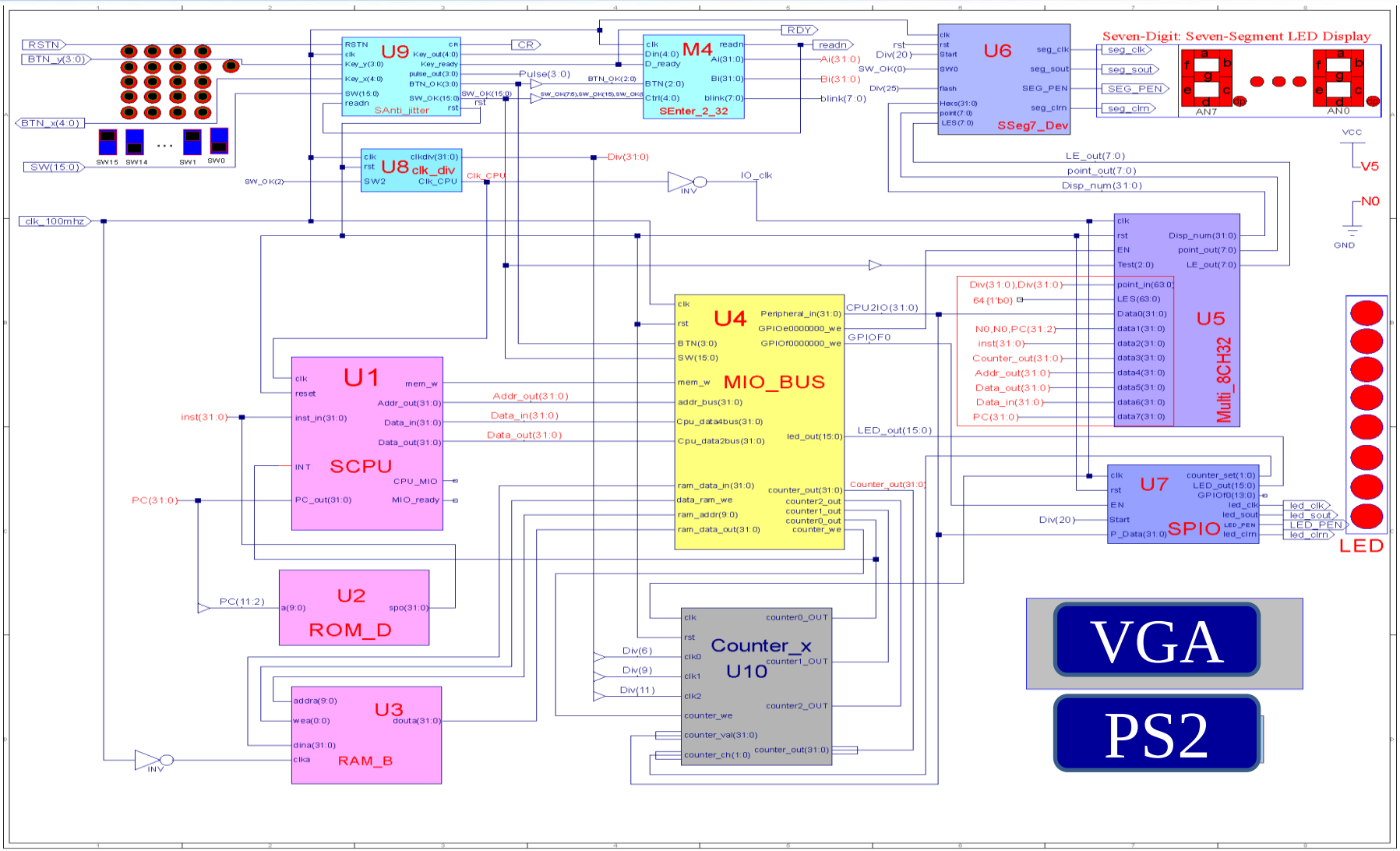
- 实验目的与实验环境
- 实验任务
- 实验原理
- 实验操作与实现

Computer Organization

Decomposability of computer systems

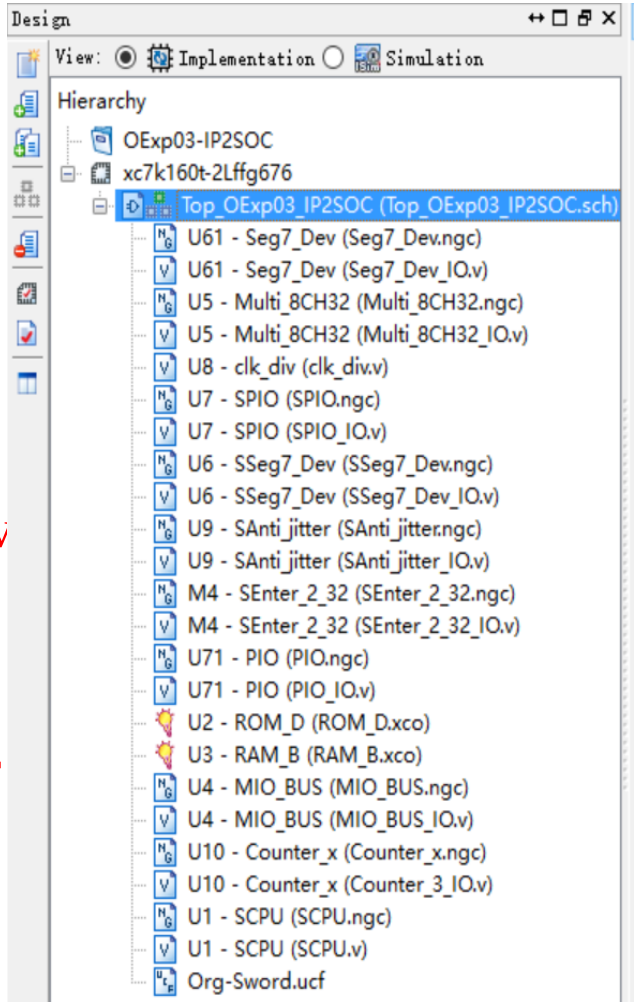


本课程实现的SOC或计算机系统

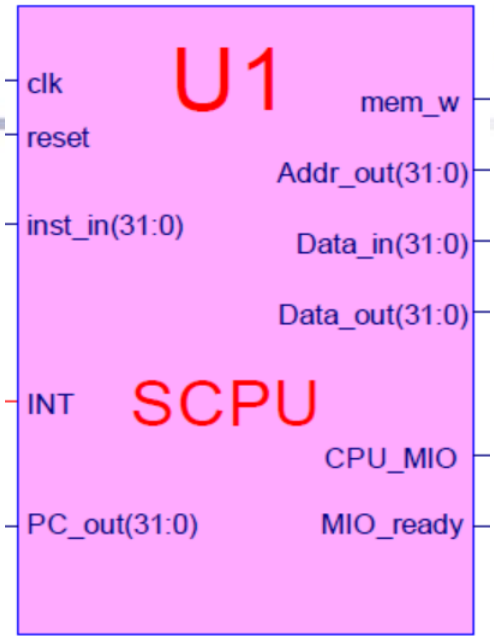


系统分解为十个子模块

- 用此10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 -SSeg7_Dev
  - U7： 外设2-GPIO接口及LED -SPIO
  - 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



U1-CPU模块：SCPU



- **MIPS 构架**
  - RISC体系结构
  - 三种指令类型
- **实现基本指令**
  - 设计实现不少于下列指令
    - R-Type： add, sub, and, or, xor, nor, slt, srl*, jr, jalr, eret* ；
    - I-Type：addi, andi, ori, xori, lui, lw, sw, beq, bne, slti
    - J-Type：J, Jal* ；
- **本实验用IP Core- U1**
  - 核调用模块SCPU.ngc
  - 核接口信号模块(空文档)：SCPU.v
  - 核模块符号文档：SCPU.sym

CPU核接口空模块-SCPU.v

```
module   SCPU( input wire clk,
               input wire reset,
               input wire MIO_ready,                  // Not used

               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_w,                     //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
```

U2-指令代码存储模块：ROM_B

- **ROM_D/B**
  - 用实验一设计的模块
  - FPGA内部存储器
    - Block Memory Generator或Distributed Memory Generator
  - 容量
    - 1024×32bit
  - 核模块符号文档：ROM_B.sym
    - 自动生成符号不规则，需要修整

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

- **本实验采用实验一生成的固核**
  - ROM初始化文档：**I9_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

ROM_B调用方式

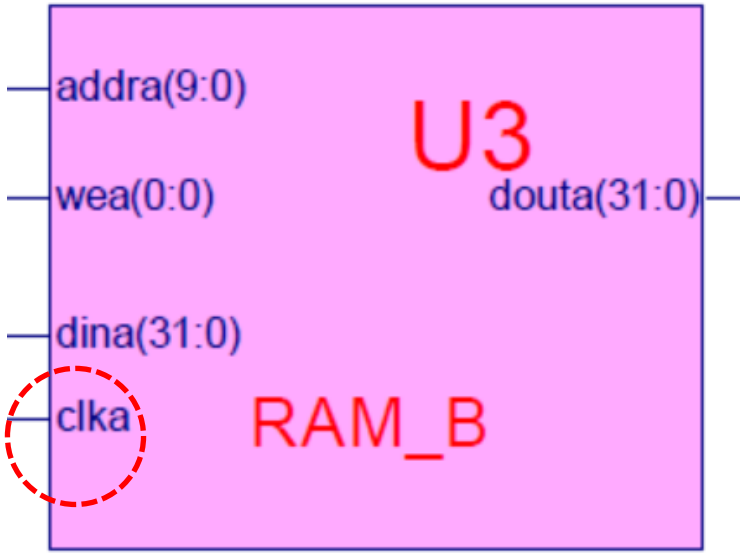
- ROM调用接口信号

```
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
             );
```

- 图形输入调用
  - ROM_B.sym
- 固核调用不需要空模块文档
  - 接口文档

U3-数据存储模块：RAM_B

- **RAM_B**
  - FPGA内部存储器
    - Block Memory Generator
  - 容量
    - 1024×32bit
  - 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

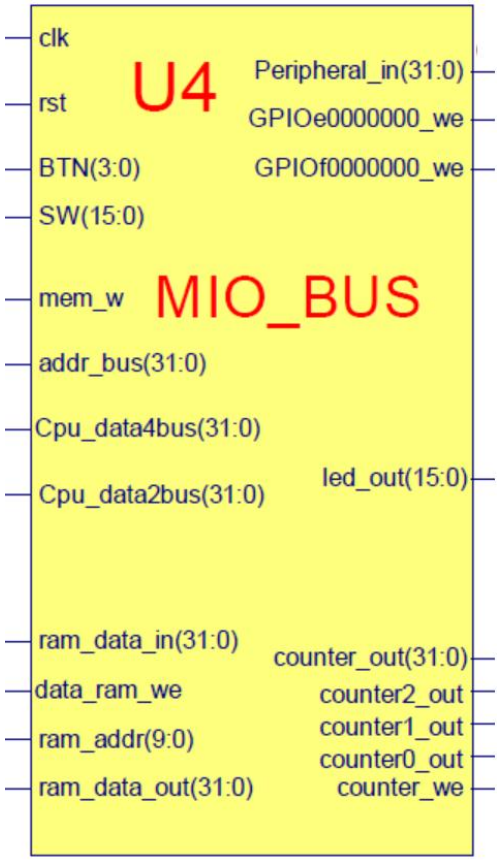
- RAM调用接口信号

```
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
               );
```

- 图形输入调用
  - RAM_B.sym
- 固核调用不需要空模块文档

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



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,             // GPIOffffff00_we
                  output wire GPIOe0000000_we,            // GPIOfffffe00_we
                  output wire counter_we,                 //记数器
                  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_100mhz,                    //主板时钟
                botton_out[3],                 //复位，按钮BTN3
                BTN [3:0] ,                    //4位原始按钮输入
                SW [7:0],                      //8位原始开关输入
                mem_w,                         //存储器读写操作，来自CPU
                Cpu_data2bus [31:0] ,          //CPU输出数据总线
                addr_bus [31:0] ,              //地址总线，来自CPU
                ram_data_out [31:0] ,          //来自RAM数据输出
                led_out [7:0],                 //来自LED设备输出
                counter_out [31:0] ,           //当前通道计数输出，来自计数器外设
                counter0_out,                  //通道0计数结束输出，来自计数器外设
                counter1_out,                  //通道1计数结束输出，来自计数器外设
                counter2_out,                  //通道2计数结束输出，来自计数器外设

                Cpu_data4bus [31:0] ,          //CPU写入数据总线，连接到 CPU
                ram_data_in [31:0] ,           //RAM 写入数据总线，连接到RAM
                ram_addr [9:  0] ,             //RAM访问地址，连接到RAM
                data_ram_we,                   //RAM读写控制， 连接到RAM
                GPIOf0000000_w,                // 设备一LED写信号
                GPIOe0000000_we,               // 设备二7段写信号，连接到U5
                counter_we,                    //记数器写信号，连接到U10
                Peripheral_in [31:0]           //外部设备写数据总线，连接所有写设备
                );

endmodule
```

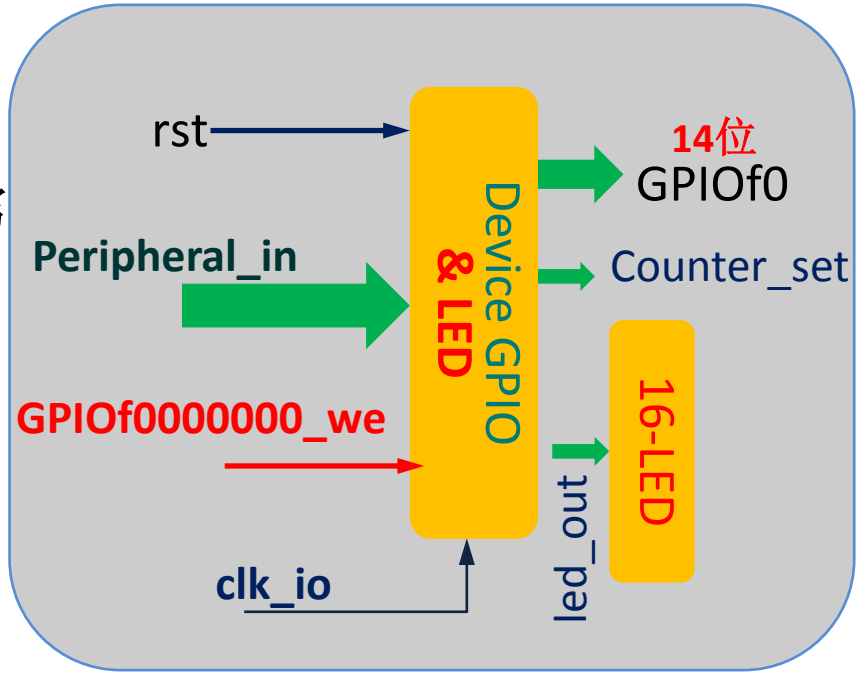
GPIO设备与接口模块

––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

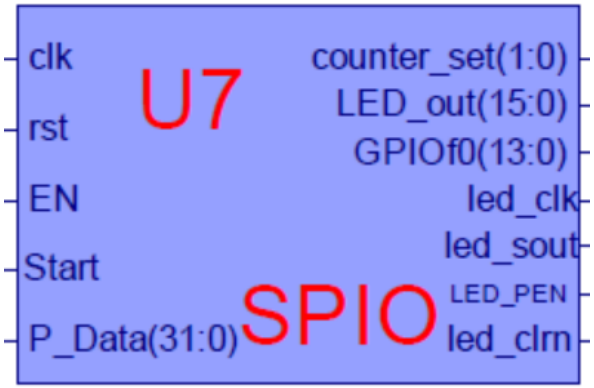
U7-外部设备模块：GPIO接口及设备一SPIO

- GPIO输出设备一
  - 地址范围=f0000000 - fffffff0
  - 读写控制信号：**GPIOf0000000_we**
  - {GPIOf0[13:0],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.ngc
  - 核接口信号模块(空文档)：SPIO_IO.v
  - 核模块符号文档：SPIO.sym

通用接口与设备—IP核调用空模块 -SPIO.v




```
module  SPIO(input clk,                    //io_clk，与CPU反向
             input rst,
             input EN,                     //来自U4
             input [31:0] P_Data,          //来自U4
             input Start,                  //串行输出启动
             output [1:0] counter_set,     //来自U7，后继用
             output [15:0] led_out,        //输出到LED,回读到U4
             output [13:0] GPIOf0          //备用
             output led_clk,               //串行时钟
             output led_sout,              //串行LEDE值
             output LED_PED,               //LED使能
             output led_clrn               //LED清零
             );

endmodule
```

U6-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
- 基本功能(参考OExp02)
  - 8位7段码显示设备
  - 模拟文本，显示8位16进制数：SW[1:0]=x1
  - 模拟图形显示，4位7段用于32个点阵显示，SW[1:0]=x0
- 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

SSeg7_Dev.v

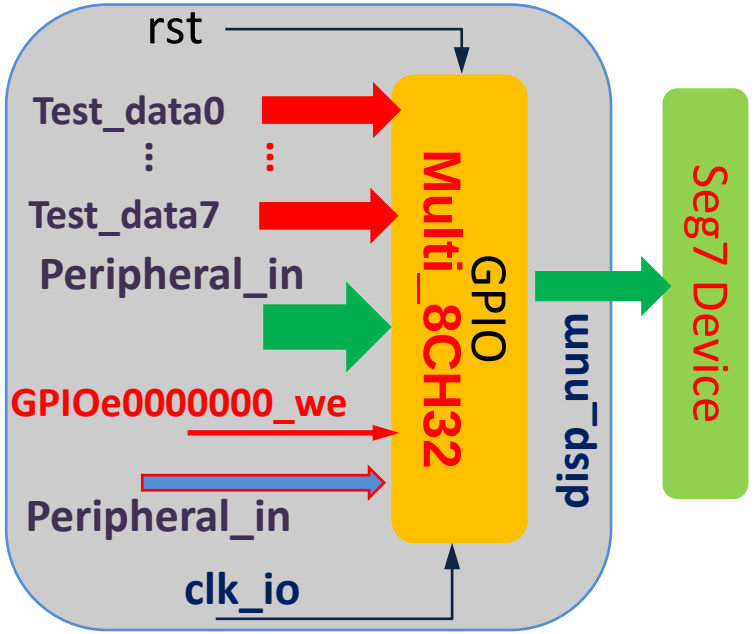
```
module  SSeg7_Dev(input clk,              //时钟
                  input rst,              //复位
                  input Start,            //串行扫描启动
                  input SW0,              //文本(16进制)/图型(点阵)切换
                  input flash,            //七段码闪烁频率
                  input [31:0]Hexs,       //32位待显示输入数据
                  input [7:0]point,       //七段码小数点：8个
                  input [7:0]LES,         //七段码使能：=1时闪烁
                  output seg_clk,         //串行移位时钟
                  output seg_sout,        //七段显示数据(串行输出)
                  output SEG_PEN,         //七段码显示刷新使能
                  output seg_clrn         //七段码显示归零
                 );
endmodule
```

clk
rst
Start
SW0
flash
Hexs

U5-通用设备二接口模块

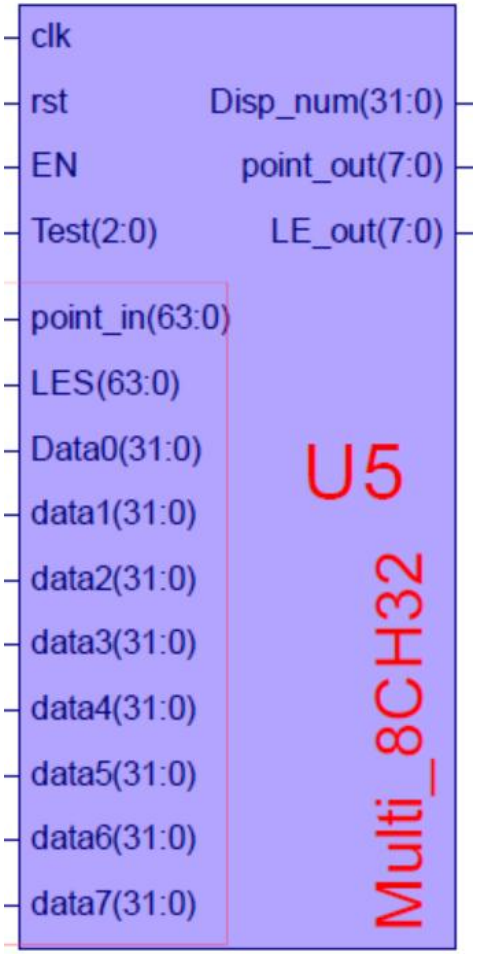
Multi_8CH32

- GPIO输出设备二接口模块
  - 地址范围=E0000000 - EFFFFFFF
  - 读写控制信号：**GPIOe0000000_we**
- 基本功能(参考Exp02)
  - 7段码输出设备接口模块
  - 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EXp02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]LES,     //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] data1,           //通道1
                               input [31:0] data2,           //通道2
                               input [31:0] data3,           //通道3
                               input [31:0]  data4,          //通道4
                               input [31:0]  data5,          //通道5
                               input [31:0]  data6,          //通道6
                               input [31:0]  data7,          //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

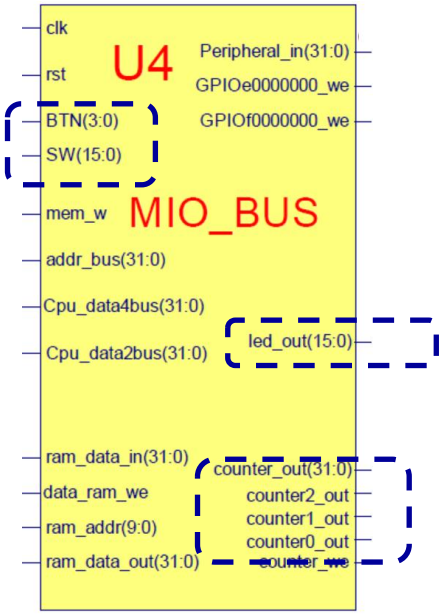
```
Multi_8CH32    U5( .clk(clk_io),  .rst(rst),
                   . EN(GPIOe0000000_we),          //来自U4
                . point_in({Div[31:0],Div[31:0]}),  //外部输入
                   . LES(64'b0),                   //外部输入
                   .Test(SW_OK[7:5]),              //来自U9
                   .Data0(CPU2IO),                 //来自U4
                   .data1({2'b00,PC[31:2]}),       //来自U1
                   .data2(inst),                  //来自CPU
                   .data3(Counter_out),            //来自U10
                   .data4(Addr_out),               //来自CPU
                   .data5(Data_out),              //来自CPU
                   .data6(Data_in),          //送CPU，来自U4
                   .data7(PC),                     //来自CPU

                   .point_out(point_out),          //输出到U6
                   .LE_out(LE_out),                //输出到U6
                   .Disp_num(Disp_num)             //输出到U6
                   );
```

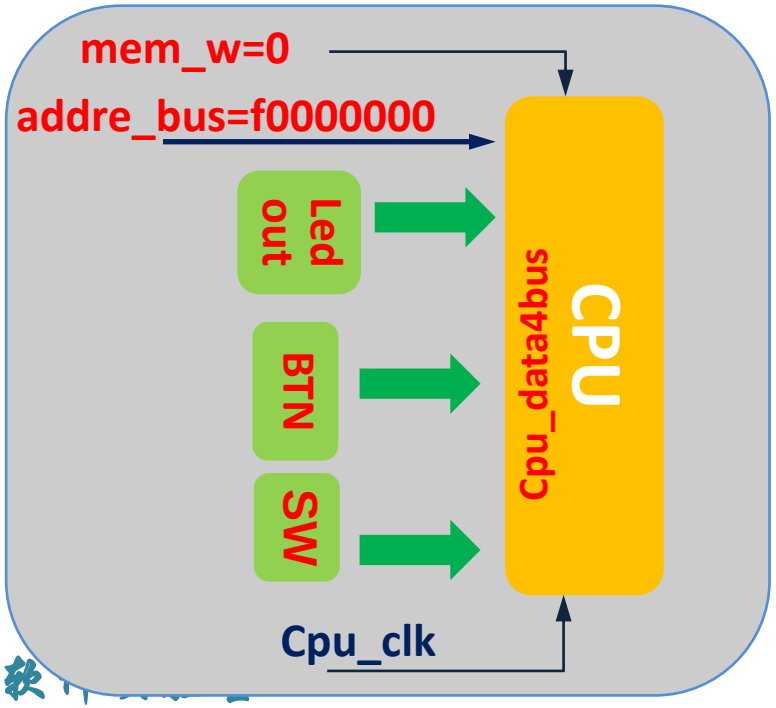
外部设备模块：GPIO接口设备三、四

Device_GPIO_SW_BTN

- **16位Switch输入设备**
  - 地址范围= f0000000-fffffff0，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数据的关系（当**addre_bus=f0000000时**）

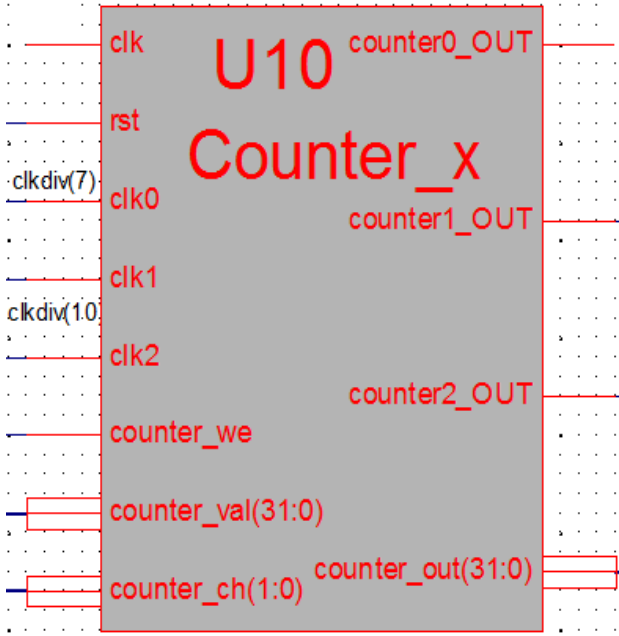
```
Cpu_data4bus = {counter0_out, counter1_out,
                counter2_out, led_out[12:0], SW};
```





U10-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 – 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.ngc
  - 核接口信号模块(空文档)：Counter_x.v
  - 核模块符号文档：Counter_x.sym



通用计数器IP核调用空模块

-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
```

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：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.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 [31:0]clkdiv,
               output Clk_CPU
              );
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
```

SOC Systemon Chip简介

- **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（网线层或掩膜层）
    - 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)

软IP核 (soft IP Core)

- 软核在EDA 设计领域指的是综合之前的寄存器传输级(RTL) 模型；具体在FPGA 设计中指的是对电路的硬件语言描述，包括逻辑描述、网表和帮助文档等。
- 软核只经过功能仿真，需要经过综合以及布局布线才能使用。
- 优点是灵活性高、可移植性强，允许用户自配置；
- 缺点是对模块的预测性较低，在后续设计中存在发生错误的可能性，有一定的设计风险；
- 软核是IP 核应用最广泛的形式。

固IP核(firm IP core)

- 固核在EDA 设计领域指的是带有平面规划信息的网表；具体在FPGA 设计中可以看做带有布局规划的软核，通常以RTL 代码和对应具体工艺网表的混合形式提供。
- 固核将RTL 描述结合具体标准单元库进行综合优化设计，形成门级网表，再通过布局布线工具即可使用；
- 和软核相比，固核的设计灵活性稍差，但在可靠性上有较大提高；
- 固核也是IP 核的主流形式之一

硬核 (Hard IP Core)

- 硬核在EDA 设计领域指经过验证的设计版图；具体在FPGA 设计中指布局和工艺固定、经过前端和后端验证的设计，设计人员不能对其修改。
- 不能修改的原因有两个：首先是系统设计对各个模块的时序要求很严格，不允许打乱已有的物理版图；其次是保护知识产权的要求，不允许设计人员对其有任何改动。
- IP 硬核的不许修改特点使其复用有一定的困难，因此只能用于某些特定应用，使用范围较窄。

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
  - 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

- 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……**

Course Outline

- 实验目的与实验环境
- 实验任务
- 实验原理
- 实验操作与实现

设计工程：OExp03-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：Top_OExp03_IP2SOC.sch
- **SOC集成技术实现系统构架**
  - 用实验一、二设计的模块和第三方IP核
    - CPU (第三方IP核)：U1
    - ROM (ISE构建核)：U2
    - 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SSeg7_Dev IP）：U6
    - LED显示模块(实验二SPIO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)

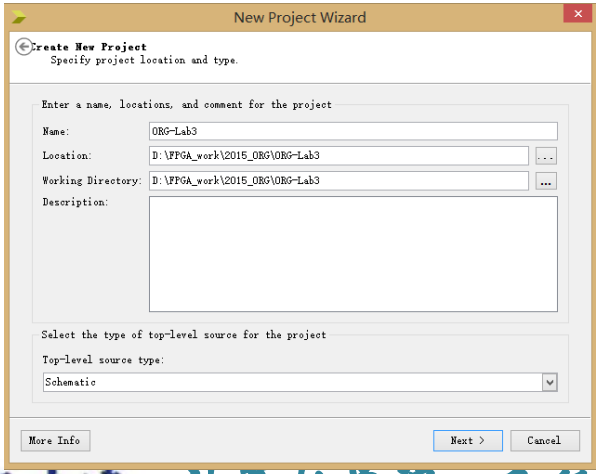
设计要点

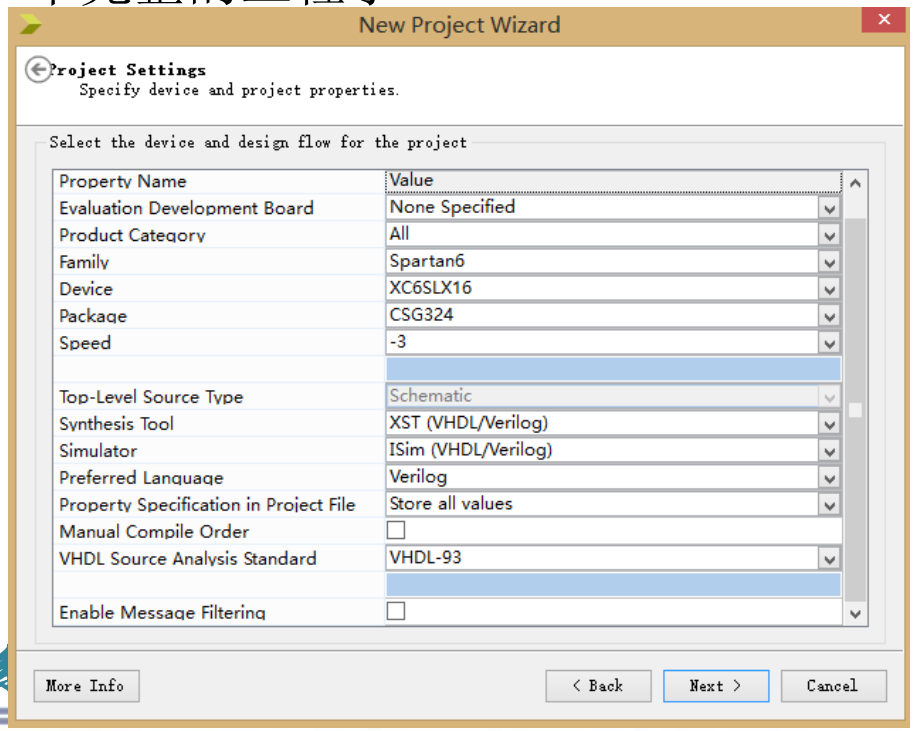
建立SOC应用工程

新建工程OExp03-IP2SOC

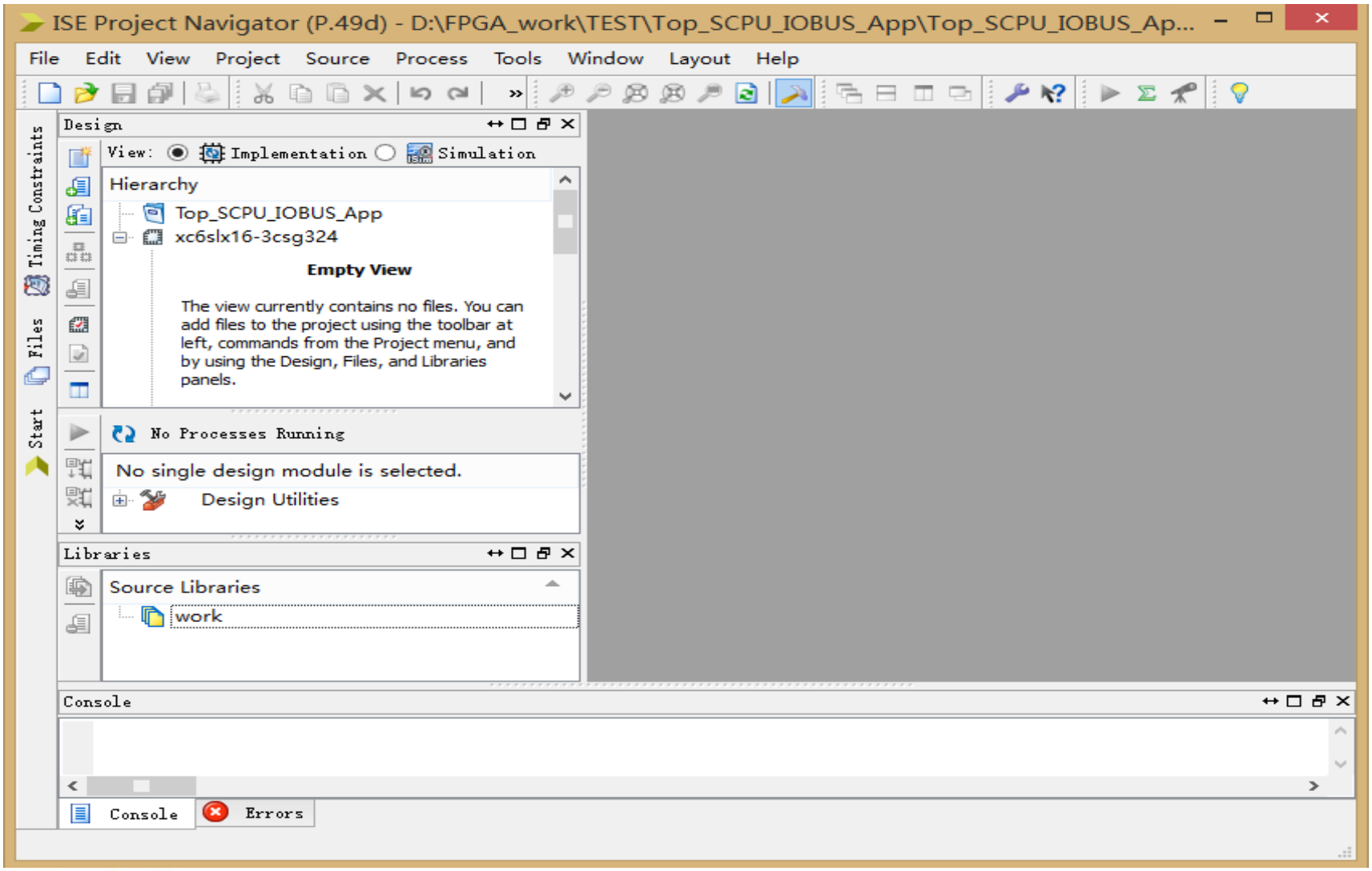
建立SOC应用工程

- 用ISE新建SOC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：**OExp03-IP2SOC**或Top_Simple_CPU_App
  - 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


ISE Project Navigator (P.49d) - D:\FPGA_work\TEST\Top_SCPU_IOBUS_App\Top_SCPU_IOBUS_Ap...
File Edit View Project Source Process Tools Window Layout Help
Design
View: Implementation Simulation
Hierarchy
Top_SCPU_IOBUS_App
xc6slx16-3csg324
Empty View
The view currently contains no files. You can add files to the project using the toolbar at left, commands from the Project menu, and by using the Design, Files, and Libraries panels.
No Processes Running
No single design module is selected.
Design Utilities
Libraries
Source Libraries
work
Console
Console
Errors
Timing Constraints
Files
Start

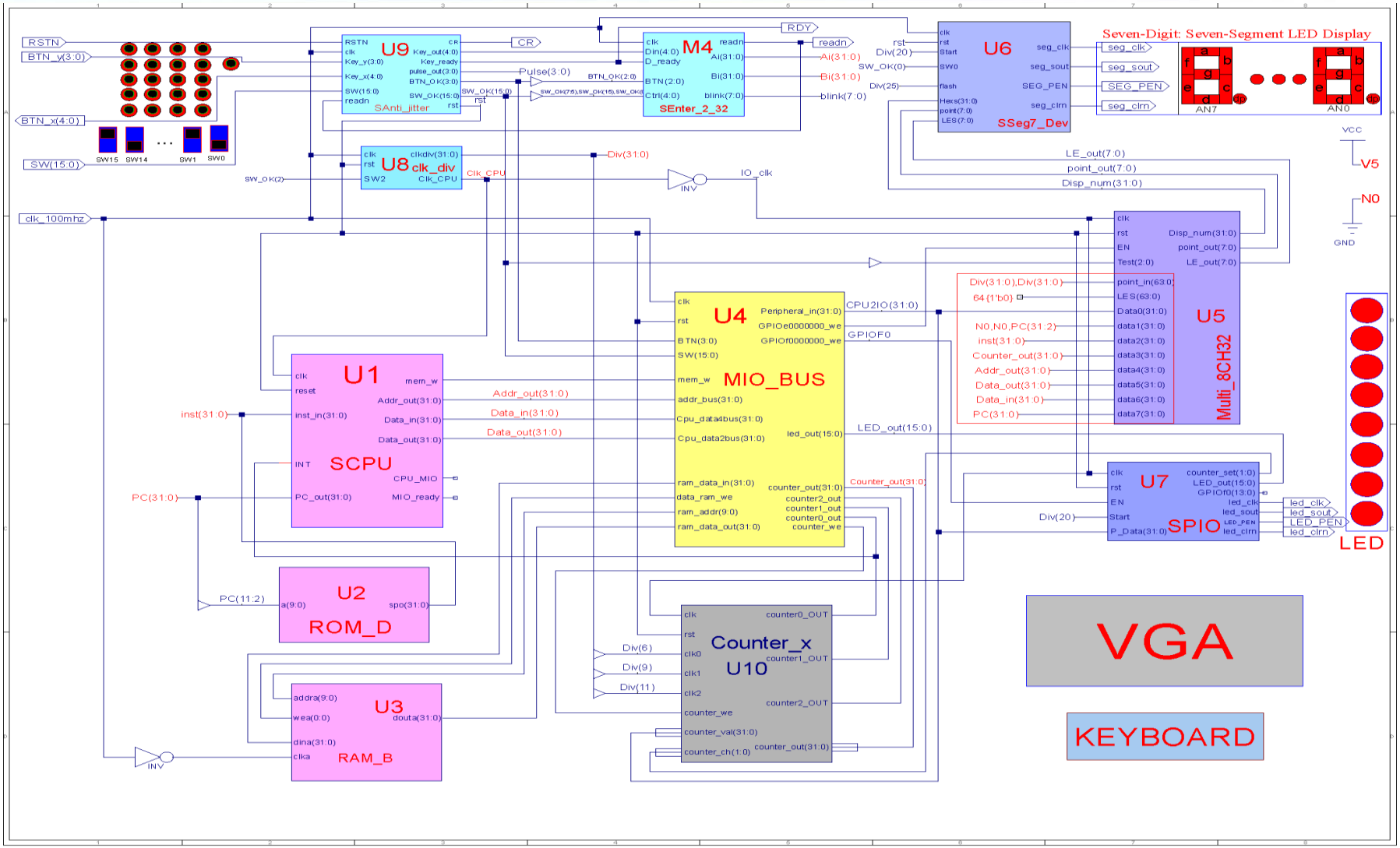
拷贝U1~U10模块的Symbol文件到当前工程目录

拷贝U1~U10模块软核.ngc文档到当前工程目录

原理图输入SoC顶层逻辑

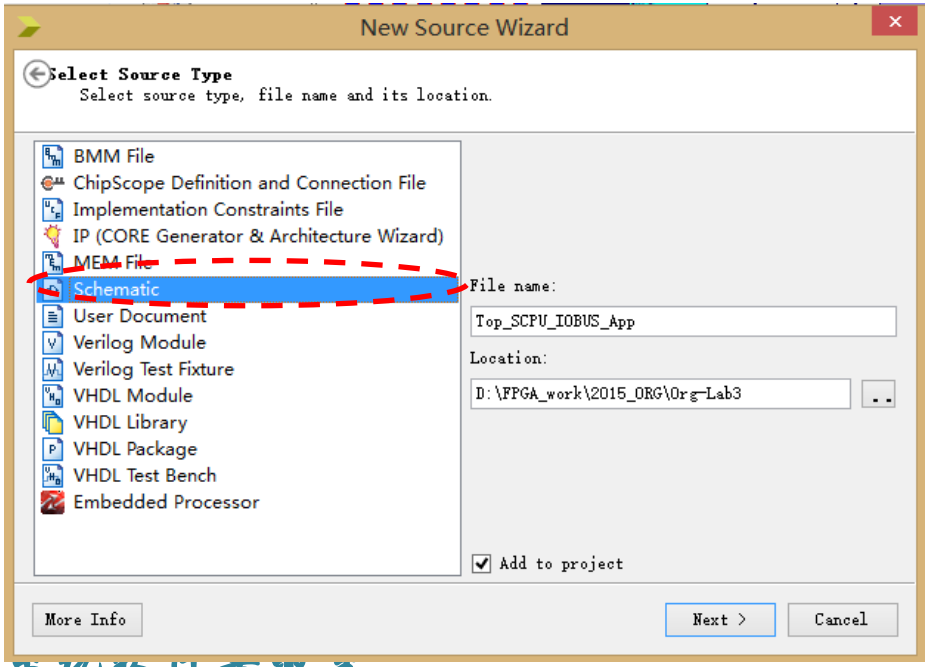
--在实验二的基址上加入U1、U2、U3、U4和U10

SOC顶层逻辑图

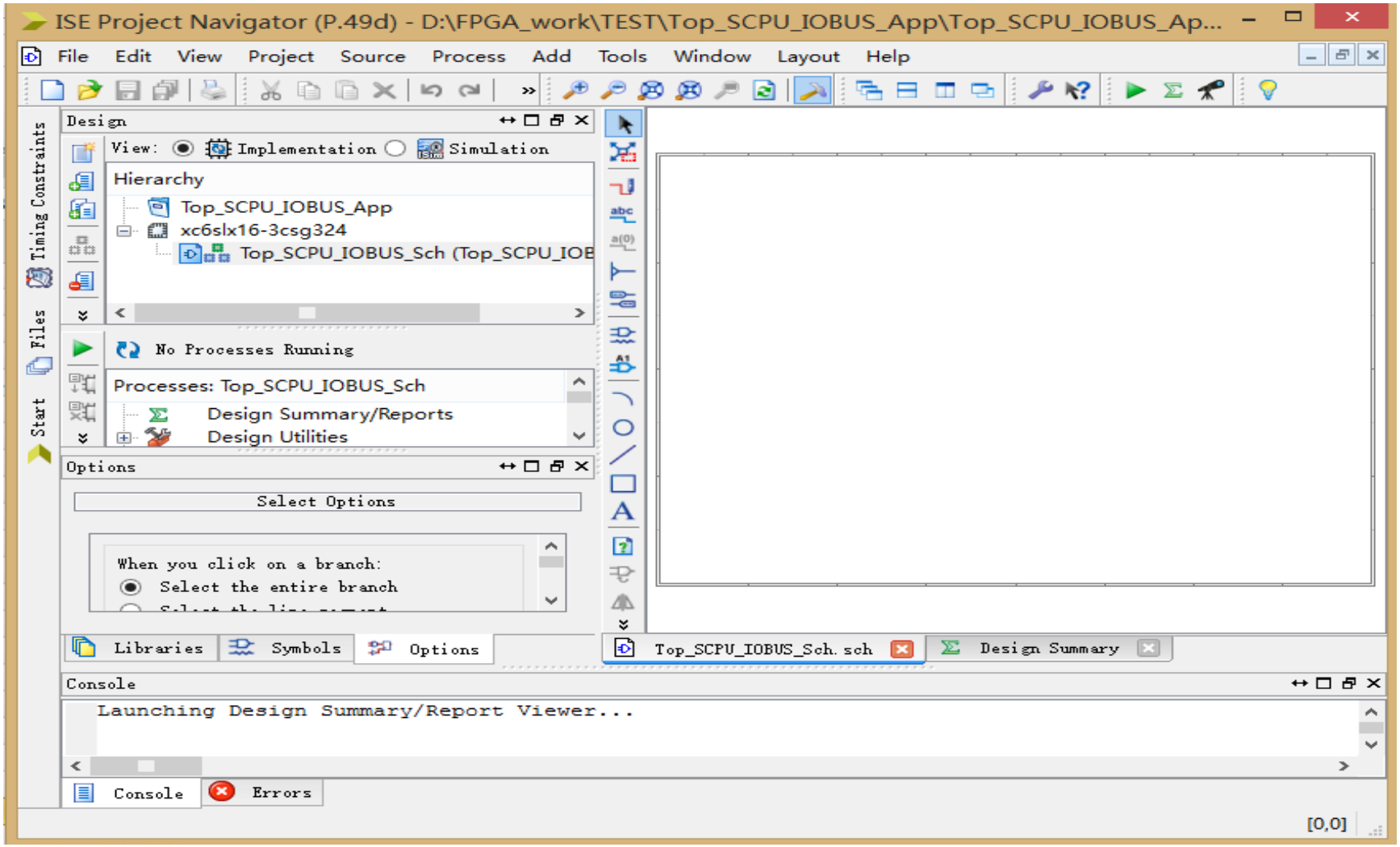

RSTN
BTN_y(3:0)
BTN_x(4:0)
SW(15:0)
SW15 SW14 SW1 SW0
clk_100mhz
U9
SAnti_jitter
CR
Pulse(3:0)
BTN_OK(2:0)
SW_OK(15:0)
RDY
readn
M4
SEnter_2_32
Ai(31:0)
Bi(31:0)
blink(7:0)
U6
SSeg7_Dev
Seven-Digit: Seven-Segment LED Display
seg_clk
seg_sout
SEG_PEN
seg_clrn
AN7
AN0
VCC
V5
GND
N0
U8
clk_div
clkdiv(31:0)
Clk_CPU
Div(31:0)
INV
IO_clk
LE_out(7:0)
point_out(7:0)
Disp_num(31:0)
U4
MIO_BUS
Peripheral_in(31:0)
GPIOe0000000_we
GPIOf0000000_we
BTN(3:0)
SW(15:0)
mem_w
addr_bus(31:0)
Cpu_data4bus(31:0)
Cpu_data2bus(31:0)
led_out(15:0)
ram_data_in(31:0)
data_ram_we
ram_addr(9:0)
ram_data_out(31:0)
counter_out(31:0)
counter0_out
counter1_out
counter2_out
counter_we
CPU2IO(31:0)
GPIOF0
U5
Multi_8CH32
U1
SCPU
inst(31:0)
PC(31:0)
Addr_out(31:0)
Data_in(31:0)
Data_out(31:0)
CPU_MIO
MIO_ready
U2
ROM_D
PC(11:2)
U3
RAM_B
U7
SPIO
LED_out(15:0)
Counter_out(31:0)
Counter_x
U10
Div(6)
Div(9)
Div(11)
LED
VGA
KEYBOARD

建立SOC原理图输入模板（顶层模块）

- 建立顶层模块
  - 在Project弹出的菜单中选择New Source命令
  - 选择原理图输入法(Schematic)
  - 缺省目录是工程目录OExp02-IP2SOC
  - 建议修改为**simple_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
- 输入SOC顶层原理图
  - 根据SOC顶层逻辑图

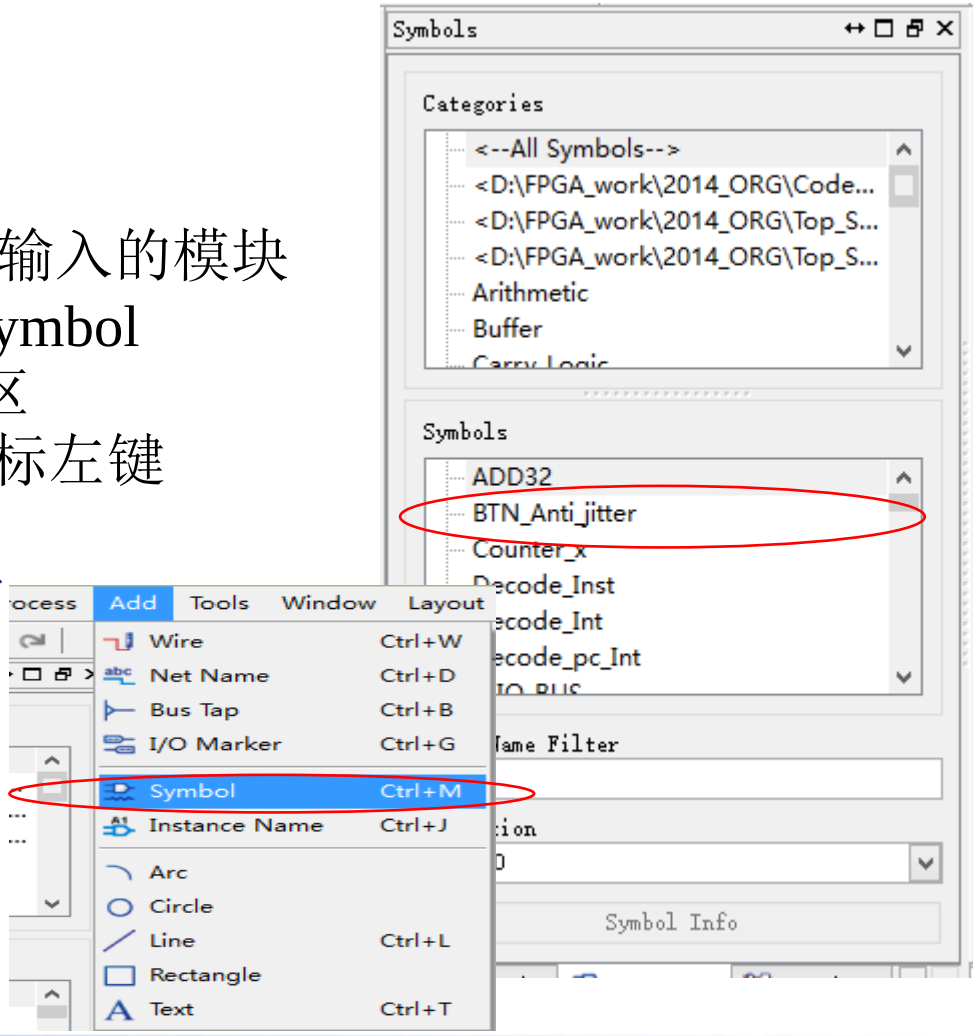


原理图输入窗口与环境

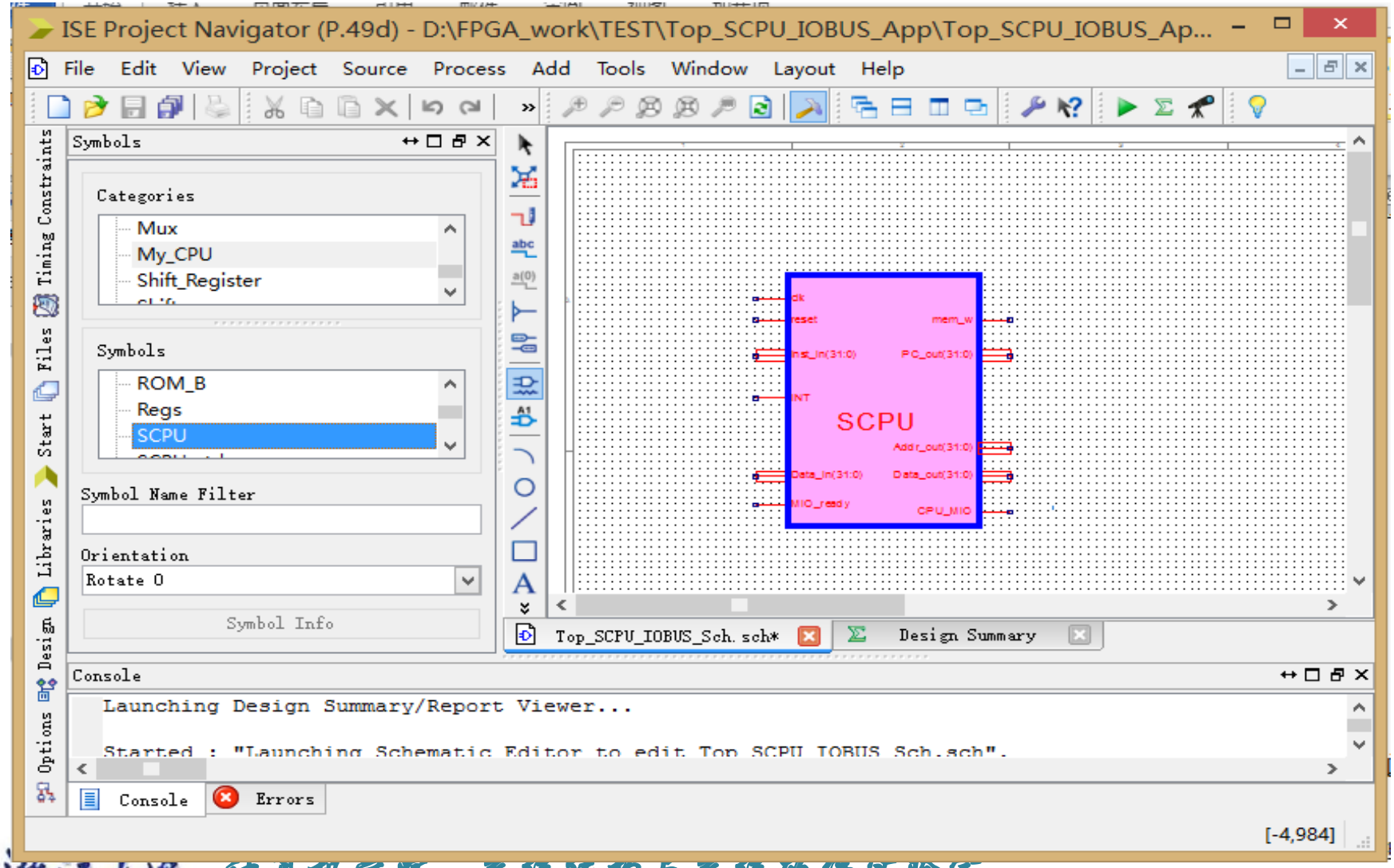

ISE Project Navigator (P.49d) - D:\FPGA_work\TEST\Top_SCPU_IOBUS_App\Top_SCPU_IOBUS_Ap...
File Edit View Project Source Process Add Tools Window Layout Help
Design
View: Implementation Simulation
Hierarchy
Top_SCPU_IOBUS_App
xc6slx16-3csg324
Top_SCPU_IOBUS_Sch (Top_SCPU_IOB
No Processes Running
Processes: Top_SCPU_IOBUS_Sch
Design Summary/Reports
Design Utilities
Options
Select Options
When you click on a branch:
Select the entire branch
Libraries
Symbols
Options
Top_SCPU_IOBUS_Sch.sch
Design Summary
Console
Launching Design Summary/Report Viewer...
Console
Errors
[0,0]
Timing Constraints
Files
Start

输入SoC顶层分解模块

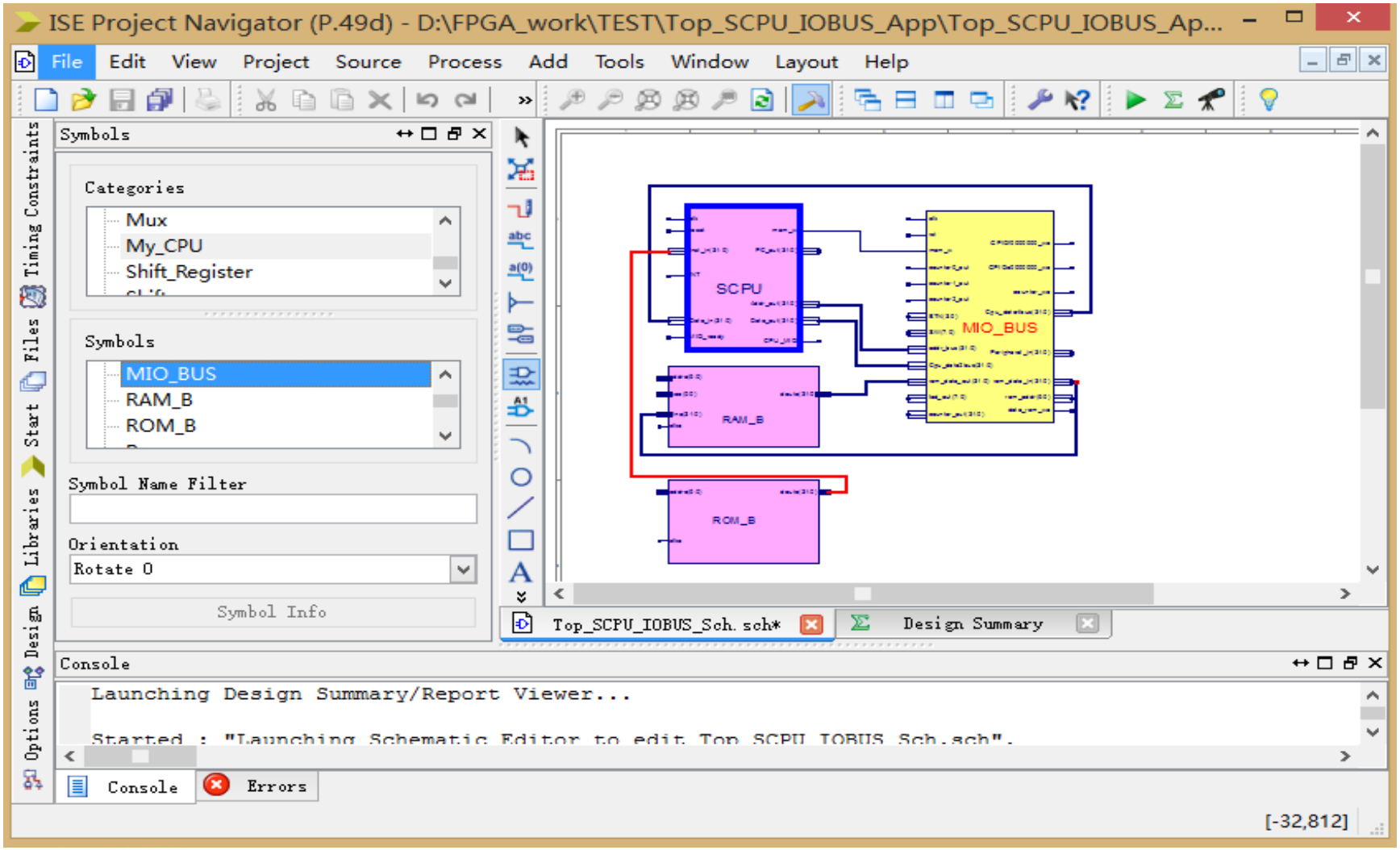
- 在原理图输入窗口输入SoC子模块
  - 激活Symbo表单容器l
  - 在Categories窗口中
    - 选择Symbol目录
    - 在Symbol窗口中选择要输入的模块
    - 在菜单栏：选择add→Symbol
      - 或光标移至图形编辑区
    - 在编辑区适当位置点鼠标左键
- 注意模块在窗口位置
  - 模块连线后移动困难
  - 必须合理安排空间



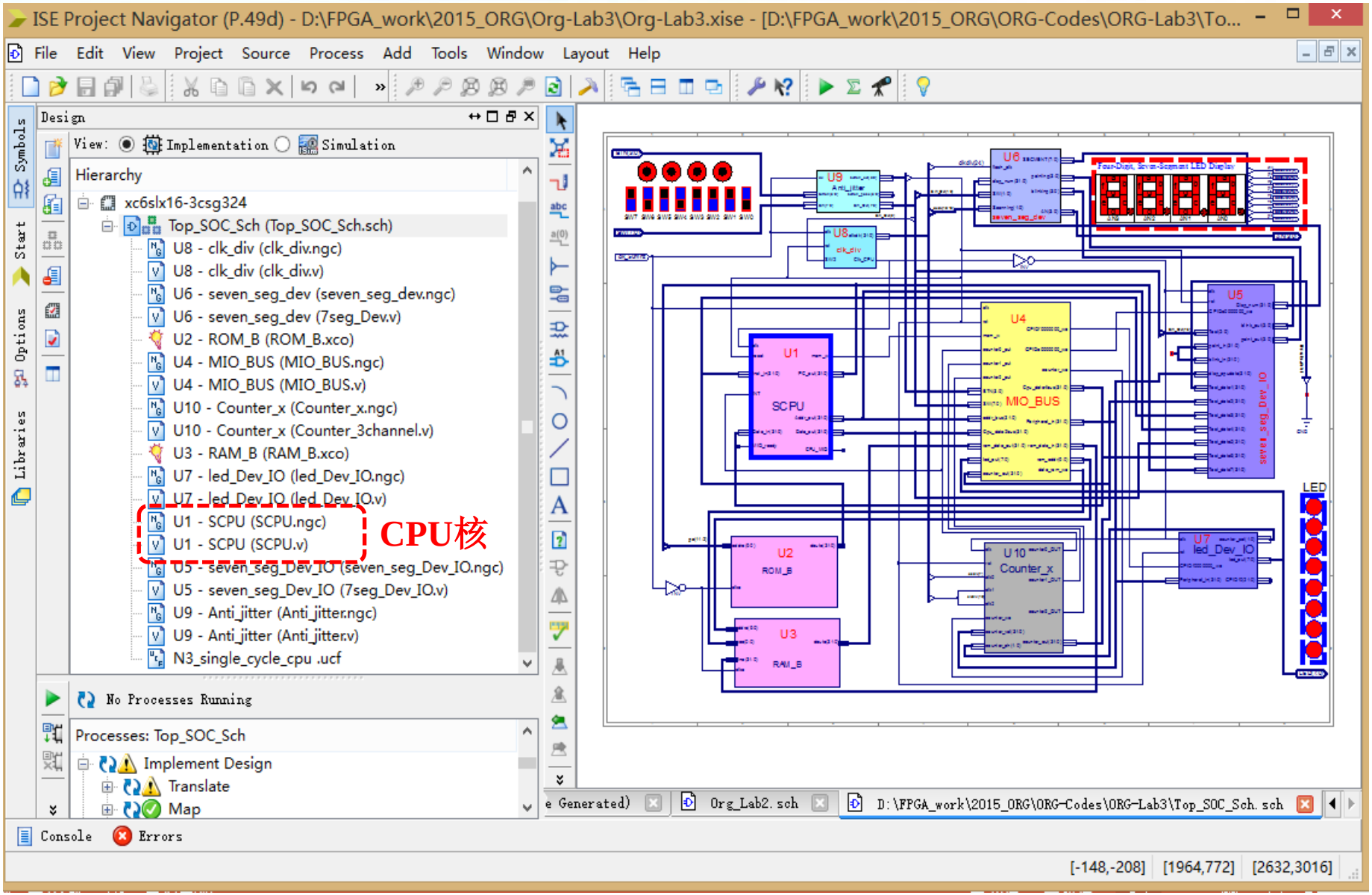
放置了CPU模块


ISE Project Navigator (P.49d) - D:\FPGA_work\TEST\Top_SCPU_IOBUS_App\Top_SCPU_IOBUS_Ap...
File Edit View Project Source Process Add Tools Window Layout Help
Symbols
Categories
Mux
My_CPU
Shift_Register
Symbols
ROM_B
Regs
SCPU
Symbol Name Filter
Orientation
Rotate 0
Symbol Info
clk
reset
mem_w
Inst_in(31:0)
PC_out(31:0)
INT
SCPU
Addr_out(31:0)
Data_in(31:0)
Data_out(31:0)
MIO_ready
CPU_MIO
Top_SCPU_IOBUS_Sch.sch*
Design Summary
Console
Launching Design Summary/Report Viewer...
Started : "Launching Schematic Editor to edit Top_SCPU_IOBUS_Sch.sch".
Console
Errors
[-4,984]
Timing Constraints
Files
Start
Libraries
Design
Options

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

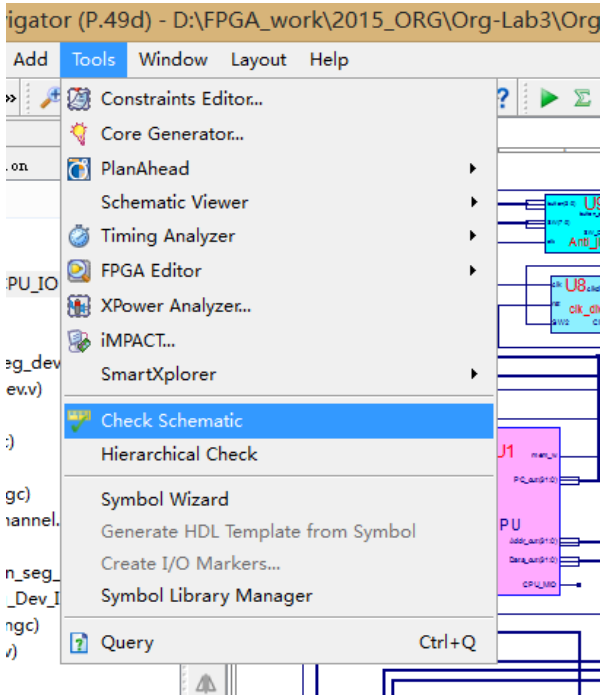


原理图检查

模块信号连接检测

- 激活原理图编辑窗口
- 在菜单栏：

  选择Tools→Check Schematic
- 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
    - 错位
    - 别名



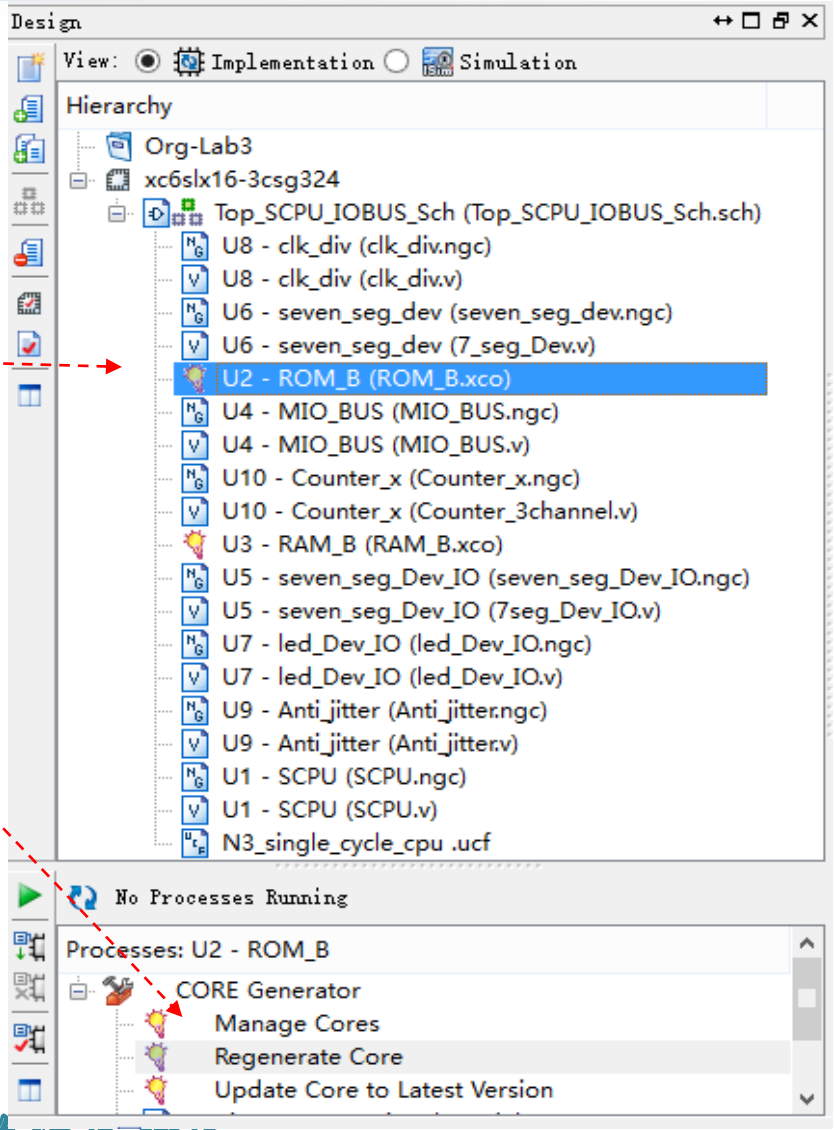
关联顶层调用模块

- 连接模块的接口信号
  - 模块放置后根据顶层分解图连接各模块
  - 信号连线时注意各信号之间的合理布线距离
    - 当连线较近时注意不要同时选中多个信号节点：NET
- 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

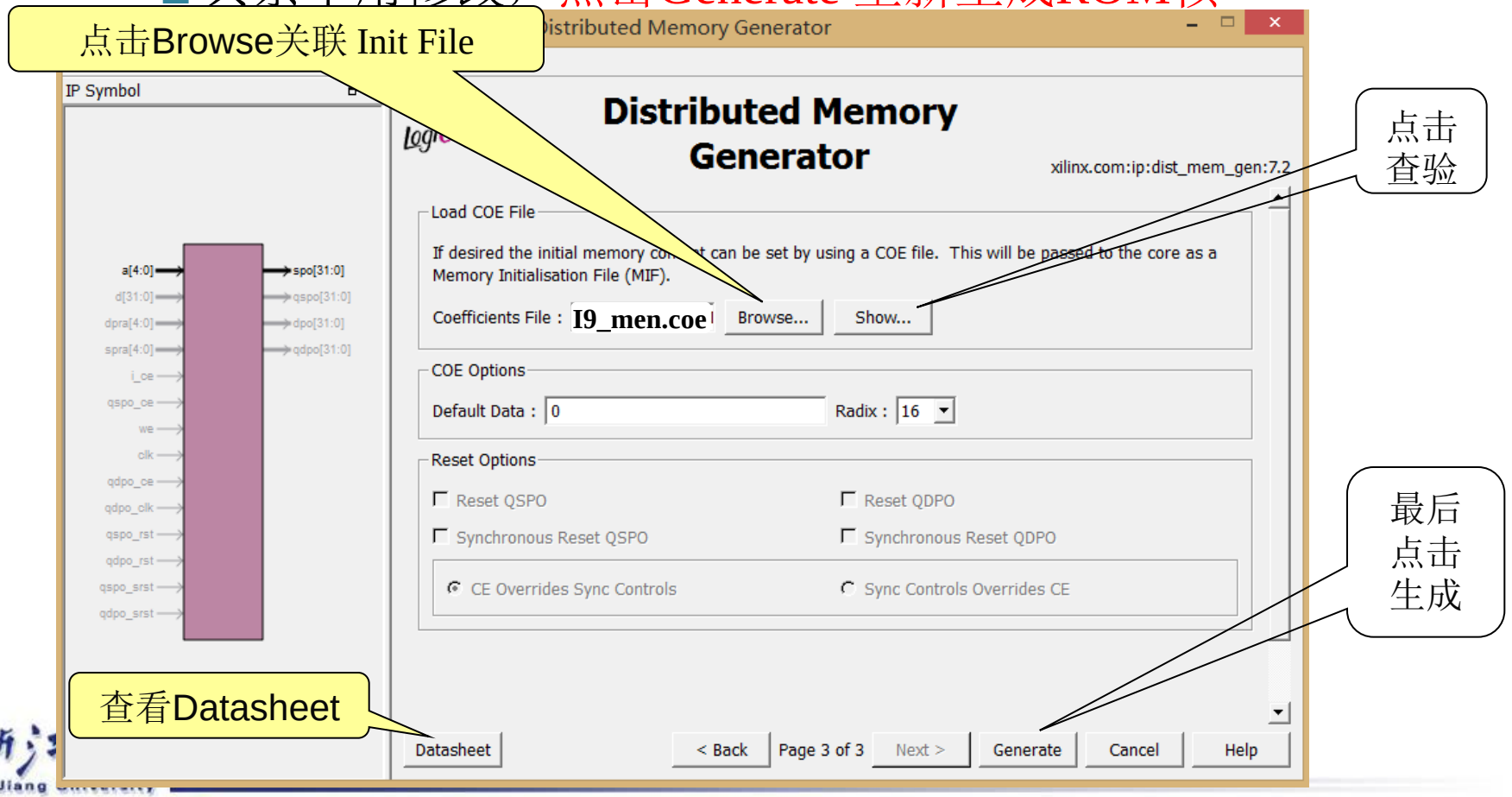
存储器IP核重新初始化

ROM_B重新初始化

- 用I9_men.coe初始化ROM
- 在设计窗口双击U2
  - 双击ROM_B进入核管理向导
    - 也可以选中ROM_B模块
      - 在Processes Running窗口双击Manager Core
  - 在核管理窗口(与核生成窗口相同)点击Next，进入核参数配置第3页(或第4页)
    - 相当于核生成〖第四步〗



- 〖第四步〗关联初始化文件并生成ROMIP核
  - 点击Next，跳过第2页 弹出窗口第3页
  - 点击“Browse...”选择初始化关联文件（**I9_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ISE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=
00000827, 0001102a, 00421820, 00622020, 00832820, 00a43020,
00c53820, 00e64020, 01074820, 01285020, 01495820, 016a6020,
018b6820, 01ac7020, 01cd7820, 01ee8020, 020f8820, 02309020,
02519820, 0272a020, 0293a820, 02b4b020, 02d5b820, 02f6c020,
0317c820, 0338d020, 0359d820, 037ae020, 039be820, 03bcf020,
03ddf820, 08000000;
```

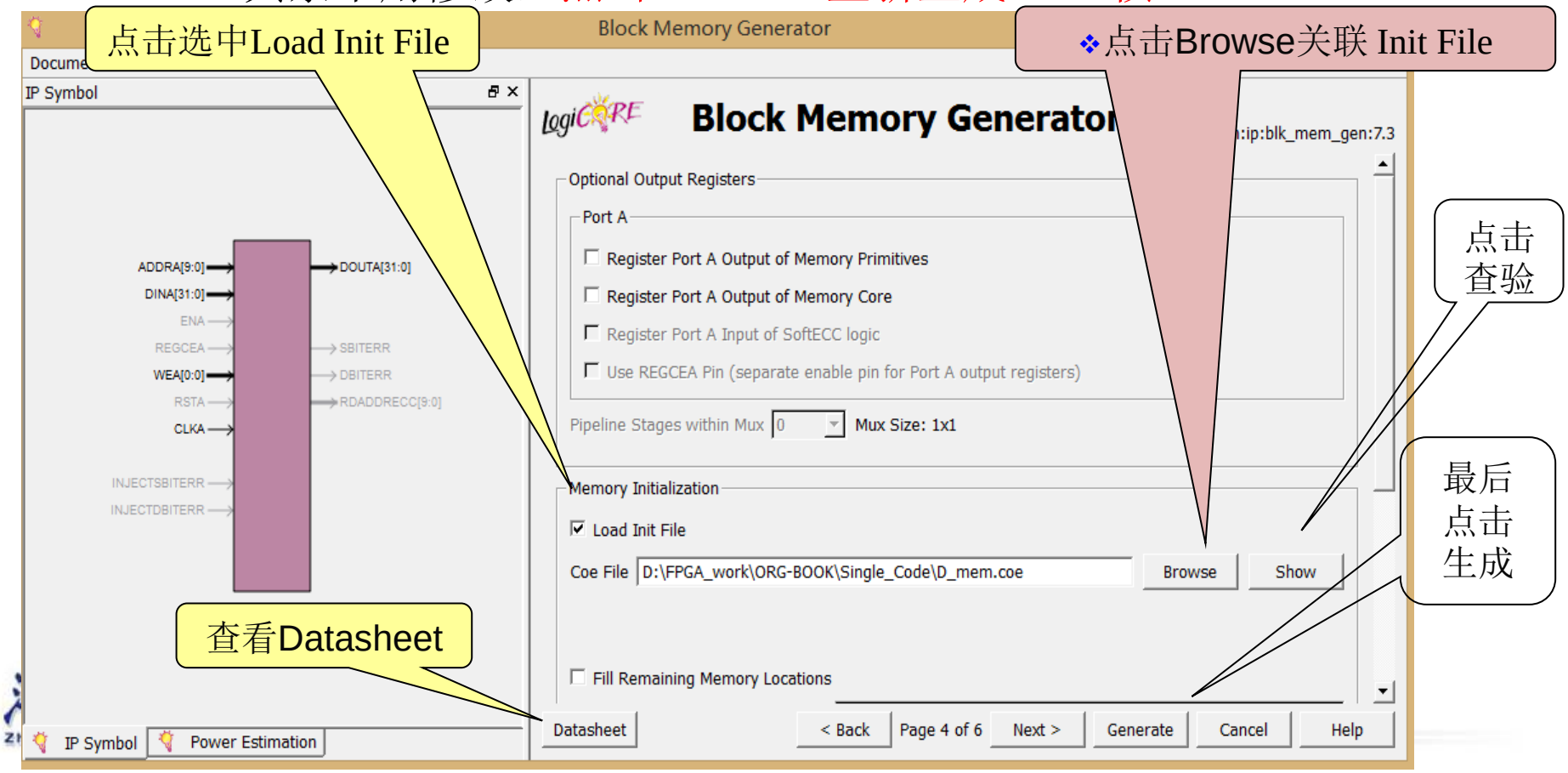
- **以上数据一段简单的指令测试**
  - CPU仿真用上述数据
  - 下载时用I9_mem.coe

简单的指令测试(暂时只要了解)

```
#baseAddr 0000
loop:   nor r1,r0,r0;          //r1=FFFFFFFF
        slt r2,r0,r1;          //r2=00000001
        add r3,r2,r2;          //r3=00000002
        add r4,r3,r2;          //r4=00000003
        add r5,r4,r3;          //r5=00000005
        add r6,r5,r4;          //r6=00000008
        add r7,r6,r5;          //r7=0000000d
        add r8,r7,r6;          //r8=00000015
        add r9,r8,r7;          //r9=00000022
        add r10,r9,r8;         //r10=00000037
        add r11,r10,r9;        //r11=00000059
        add r12,r11,r10;       //r12=00000090
        add r13,r12,r11;       //r13=000000E9
        add r14,r13,r12;       //r14=00000179
        add r15,r14,r13;       //r15=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
        j loop;
```

RAM_B初始化

- 与ROM同样方法进入核管理向导，点击Next进入第4页
  - 〖第四步〗关联初始化文件并生成RAM IP核
    - 点击Next，弹出窗口第4页
    - 点击“Browse...”选择初始化关联文件（**D_men.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=
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;
```

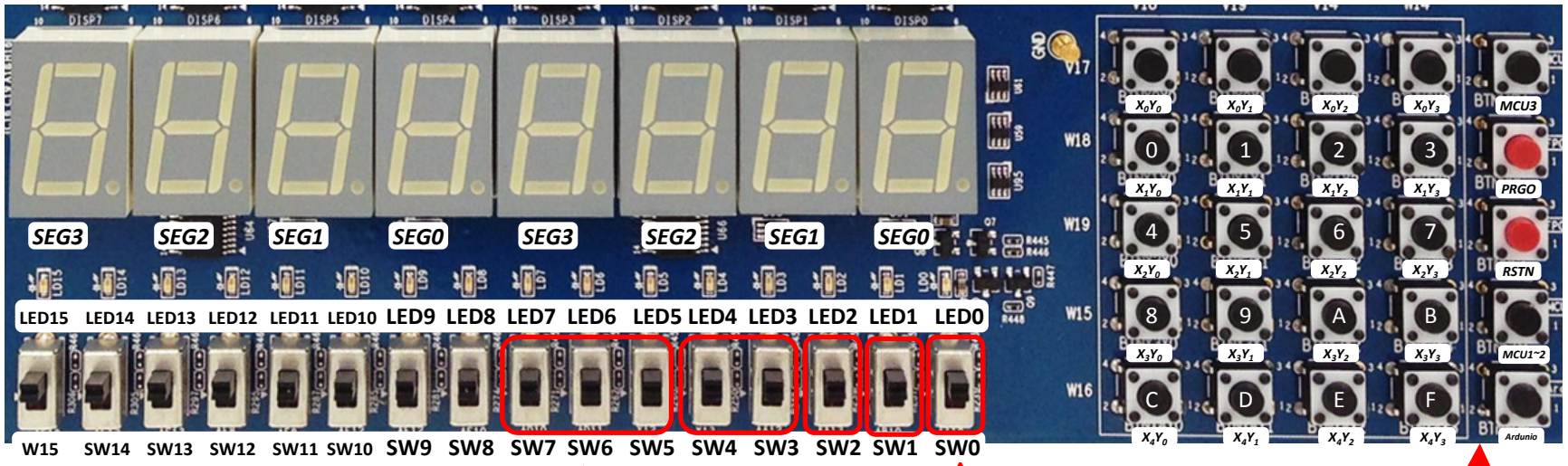
下载和仿真均可用

红色数据是LED图形

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[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

SW[2]=CPU单步时钟选择

~~SW[1]=高低16位选择~~

SW[0]=文本图形选择

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
  - 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I9_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）

- 文本功能测试

开关	位置	功能
SW[1:0]	X1	七段码文本显示
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）

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