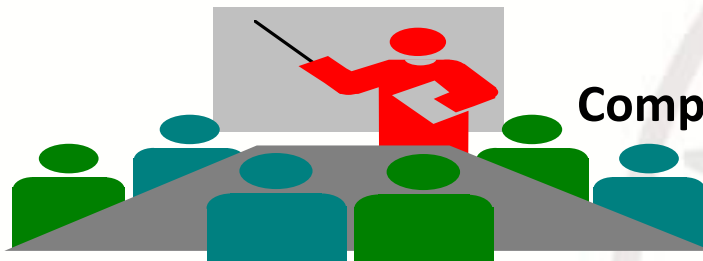




浙江大学
ZHEJIANG UNIVERSITY



Computer Organization & Design

Computer Organization & Design 实验与课程设计

实验一

多路选择器与CPU辅助模块设计

--数字逻辑实验输出模块扩展--

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. 熟练掌握EDA开发工具和开发流程
2. 复习数字逻辑设计实现方法
3. 扩展优化逻辑实验基本模块
4. 优化计算机系统实现的辅助模块
5. 了解计算机硬件系统中到的最基本模块



实验环境

□ 实验设备

1. 计算机（Intel Core i5以上，4GB内存以上）系统
2. Spartan-3 Starter Kit Board/Sword开发板
3. Xilinx ISE14.4及以上开发工具

□ 材料

无

Course Outline

A vertical diagram showing four steps of a course outline. Each step consists of a white circle on the left and a colored rectangular bar on the right. The circles are connected by a vertical line. The first circle has a diagonal line passing through it. The bars are blue, yellow, blue, and blue from top to bottom.

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. 整理设计逻辑实验输出模块

- 多路选择器、基本算术逻辑运算模块、数据扩展模块

2. 整理逻辑实验输出的辅助模块

- 消除机械抖动模块、通用分频模块

3. 设计存储器IP模块

- 32位ROM、32位RAM

4. 设计CPU调试测试显示通道模块

- 在逻辑实验Exp13基础上重建

Course Outline





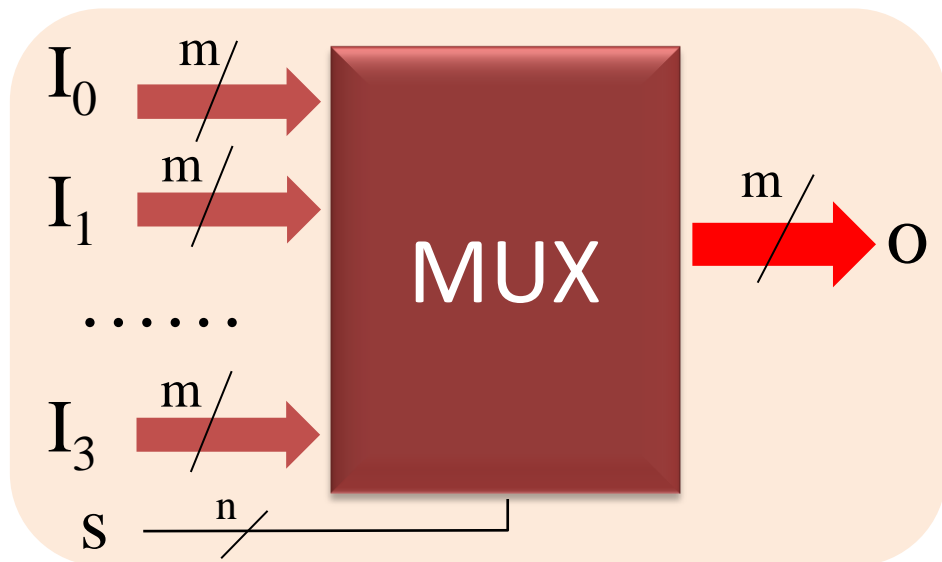
逻辑实验输出模块优化

—组成实验的基本逻辑器件

逻辑实验输出模块优化：多路器

□ 多数选择器

- 逻辑结构如下图所示，在数字逻辑实验课设计过多种
- 在CPU等部件设计将用到的重要模块
- 本课程将用到的多数选择器有：
 - 2选1：5位、32位、8位等
 - 4选1：5位、32位
 - 8选1：8位、32位

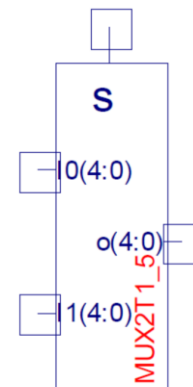




逻辑实验输出模块优化：多路器一

□ 5位2选一

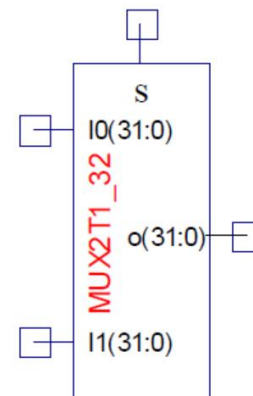
```
module MUX2T1_5(input [4:0] I0,  
                input [4:0] I1,  
                input sel,  
                output reg [4:0] o  
                );  
    .....  
endmodule
```



MUX2T1_5.sym

□ 32位2选一

```
module MUX2T1_32(input [31:0] I0,  
                 input [31:0] I1,  
                 input sel,  
                 output reg [31:0] o  
                 );  
    .....  
endmodule
```



MUX2T1_32.sym



逻辑实验输出模块优化：多路器二

□ 5位4选一

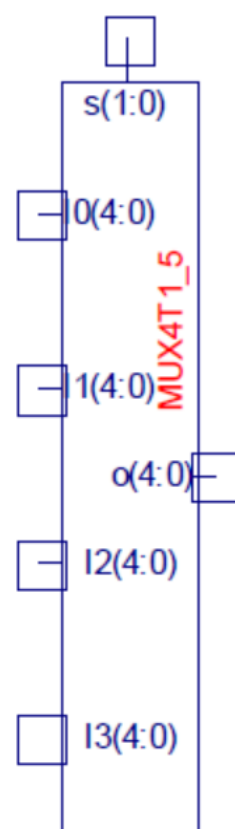
```
module MUX4T1_5(input [1:0] s,  
                input [4:0] I0,  
                input [4:0] I1,  
                input [4:0] I2,  
                input [4:0] I3,  
                output reg[4:0] o  
                );
```

```
.....  
endmodule
```

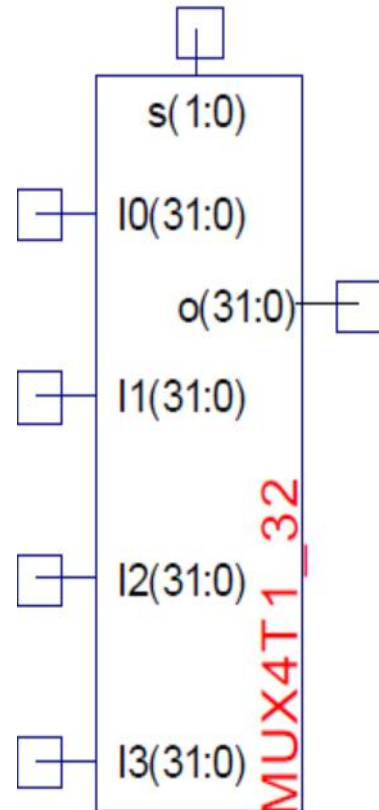
□ 32位4选一

```
module MUX4T1_32(input [1:0] s,  
                 input [31:0] a,  
                 input [31:0] b,  
                 input [31:0] c,  
                 input [31:0] d,  
                 output reg [31:0] o  
                 );
```

```
.....  
endmodule
```



MUX4T1_5.sym



MUX4T1_32.sym

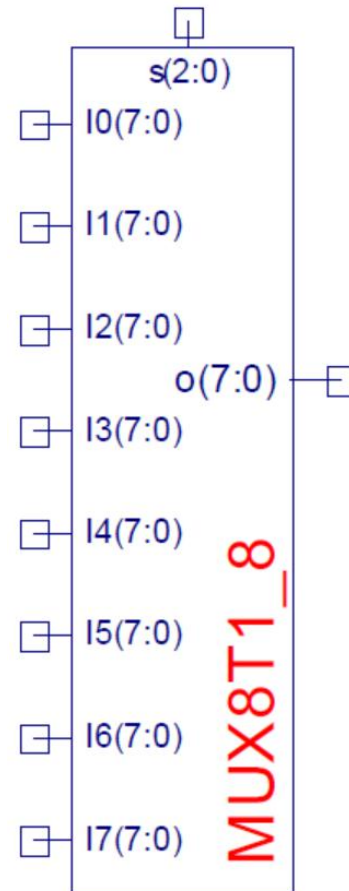
逻辑实验输出模块优化：多路器三



□ 32位8选一

- 逻辑图形符号文件：**MUX8T1_32.sym**

```
module MUX8T1_32(input [31:0] I0,  
                 input [31:0] I1,  
                 input [31:0] I2,  
                 input [31:0] I3,  
                 input [31:0] I4,  
                 input [31:0] I5,  
                 input [31:0] I6,  
                 input [31:0] I7,  
                 input [2:0] s,  
                 output reg [31:0] o  
                );  
    .....  
endmodule
```



逻辑实验输出模块优化：算术函数



□ 32位加法器模块-无进位:**add_32**

- 逻辑图形符号文件: **add_32.sym**

```
module add_32(input [31:0] a,  
              input [31:0] b,  
              output [31:0] c  
              );
```

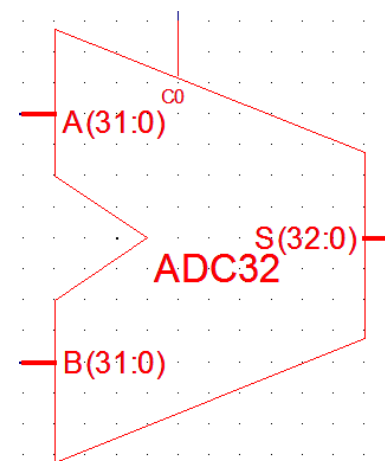
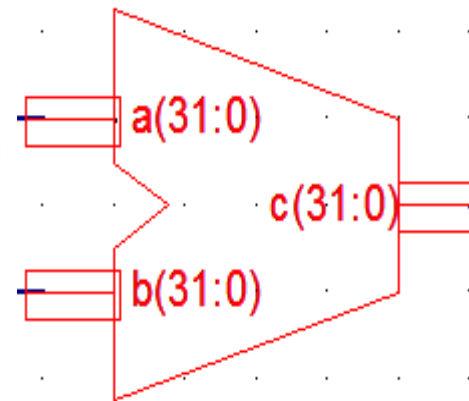
.....

```
endmodule
```

□ 32位加减器模块-带进位:**ADC32**

- 逻辑图形符号文件: **ADC32.sym**

```
module ADC32(input [31:0] A,  
             input [31:0] B,  
             input C0,           //C0=1减法  
             output [32:0] S     //S[32]进位  
             );
```



逻辑实验输出模块优化：逻辑函数



□ 32位“与”运算模块：and32

- 逻辑图形符号文件：and32.sym

```
module and32(input [31:0] A,  
             input [31:0] B,  
             output [31:0] res  
);
```

.....

```
endmodule
```

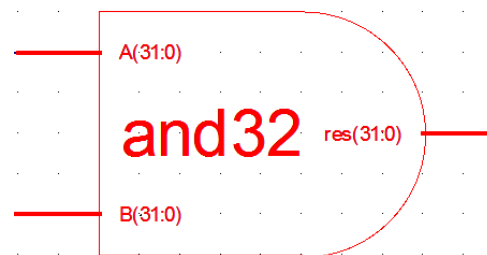
32位“或”运算模块：or32

- 逻辑图形符号文件：and32.sym

```
module or32(input [31:0] A,  
            input [31:0] B,  
            output [31:0] res  
);
```

.....

```
endmodule
```



逻辑实验输出模块优化：逻辑函数



□ 32位“或非”运算模块：nor32

- 逻辑图形符号文件：nor32.sym

```
module nor32(input [31:0] A,  
             input [31:0] B,  
             output [31:0] res  
);
```

.....

```
endmodule
```

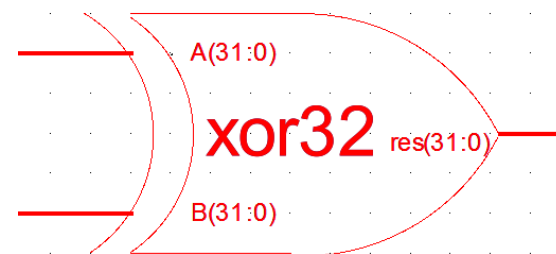
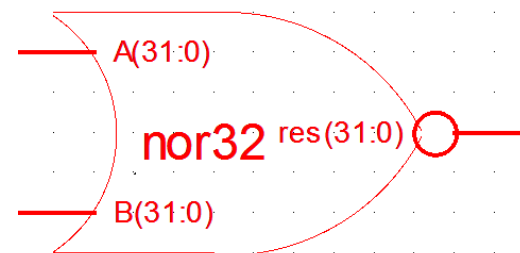
□ 32位“异或”运算模块：xor32

- 逻辑图形符号文件：xor32.sym

```
module xor32(input [31:0] A,  
             input [31:0] B,  
             output [31:0] res  
);
```

.....

```
endmodule
```



逻辑实验输出模块优化：逻辑函数



□ 32位数逻辑右移：**srl32**

- 逻辑图形符号文件：**srl.sym**
- 移位量由B[10:6]决定
- **SP3开发板固定右移1位**

```
module srl32(input [31:0] A,  
             input [31:0] B,  
             output [31:0] res  
             );
```

.....

endmodule

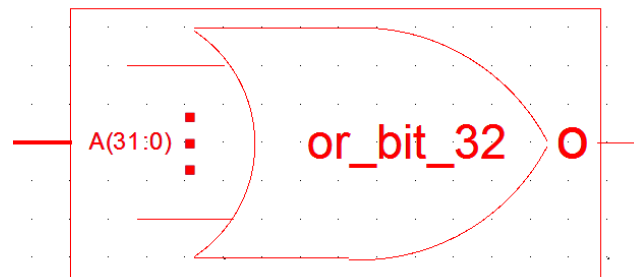
□ 32位数自”或(全零判断)”运算模块：**or_bit_32**

- 逻辑图形符号文件：**or_bit_32.sym**

```
module or_bit_32( input [31:0] A,  
                  output o  
                  );
```

.....

endmodule





逻辑实验输出模块优化：位扩展

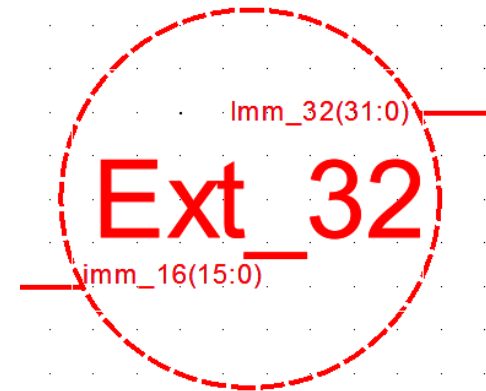
□ 16-32位数算术扩展：Ext_32

- 16位符号数扩展为32位符号数
- 逻辑图形符号文件：Ext_32.sym

```
module Ext_32(input [15:0] imm_16,  
              output[31:0] Imm_32  
              );
```

.....

```
endmodule
```

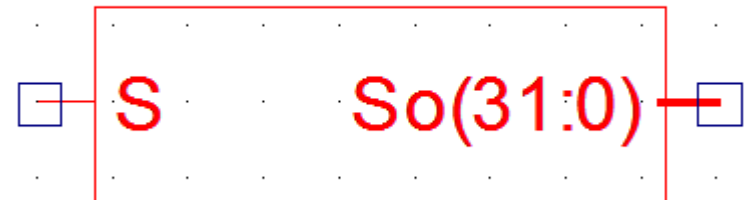


Ext_32.sym

□ 1位信号算术扩展成32位：SignalExt_32

- 逻辑图形符号文件：SignalExt_32.sym

```
module SignalExt_32 (input S,  
                    output [31:0]S  
                    );  
    assign So = {32{S}};  
endmodule
```





逻辑实验输出模块优化

—组成实验使用的辅助逻辑部件



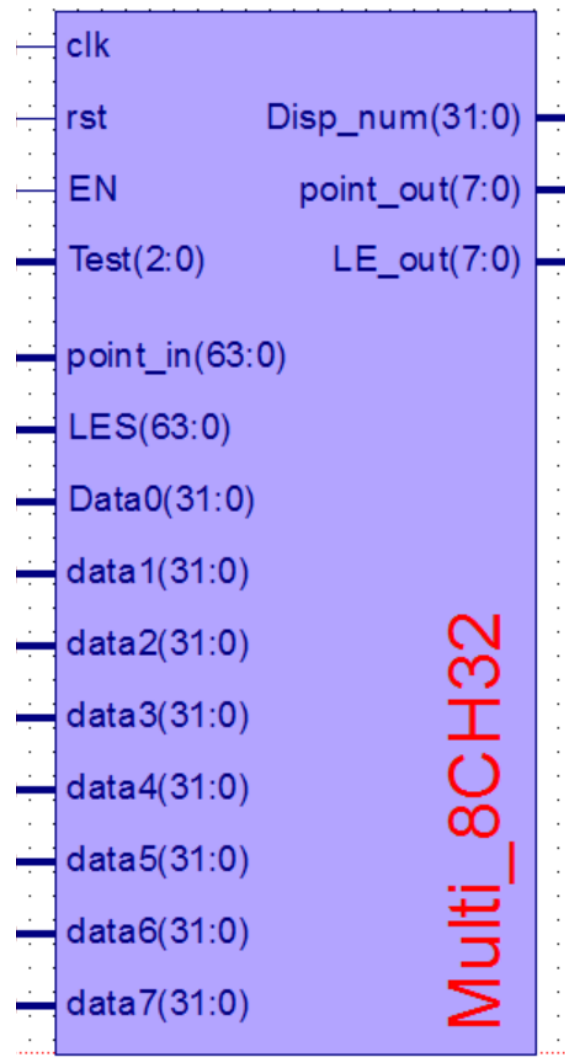
八数据通路模块：Multi_8CH32

□ 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
 - Point_in(63:0): 小数点输入
 - 每个通道8位，共64位
 - LES(63:0): 使能LE (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



Multi_8CH32.sym



八路数据通道模块参考描述： 端口描述

```
module      Multi_8CH32(input  clk,
                        input  rst,
                        input  EN,
                        input [2:0]Test,
                        input [63:0]point_in,
                        input [63:0]blink_in,
                        input [31:0] Data0,
                        input [31:0] Test_data1,
                        input [31:0] Test_data2,
                        input [31:0] Test_data3,
                        input [31:0] Test_data4,
                        input [31:0] Test_data5,
                        input [31:0] Test_data6,
                        input [31:0] Test_data7,
                        output [7:0] point_out,
                        output [7:0] blink_out,
                        output [31:0]Disp_num
                        );

//Write EN
//ALU&Clock, SW[7:5]
//针对8位显示输入各8个小数点
//针对8位显示输入各8个闪烁位
//disp_cpudata

reg[31:0] disp_data = 32'hAA5555AA;
reg[7:0]  cpu_blink = 8'b11111111, cpu_point = 4'b00000000;
```

.....调用三个MUX8T1_32和通道0处理

endmodel



32位数据八通道模块：调用MUX8T1_32

◎数据通道：

不一样哦

```
MUX8T1_32    MUX1_DispData(.IO disp_data),
               .I1(Test_data1),
               .I2(Test_data2),
               .I3(Test_data3),
               .I4(Test_data4),
               .I5(Test_data5),
               .I6(Test_data6),
               .I7(Test_data7),
               .s(Test),           //显示信号选择, Test=SW[7:5] 控制
               .o(Disp_num)       //七段码显示信息
            );
```

◎使能通道：

```
MUX8T1_8     MUX2_Blink(.IO(cpu_blink),
                        .I1(LES[15:8]),
                        .I2(LES[23:16]),
                        .I3(LES[31:24]),
                        .I4(LES[39:32]),
                        .I5(LES[47:40]),
                        .I6(LES[55:48]),
                        .I7(LES[63:56]),
                        .s(Test),           //显示信号选择, Test=SW[7:5] 控制
                        .o(LE_out)         //七段码小数点显示信息
            );
```



◎小数点通道:

有个小错误?

通道“0”控制:

```
MUX8T1_8      MUX3_Point(.I0(cpu_point),
                        .I1(point_in[15:7]),
                        .I2(point_in[23:16]),
                        .I3(point_in[31:24]),
                        .I4(point_in[39:32]),
                        .I5(point_in[47:40]),
                        .I6(point_in[55:48]),
                        .I7(point_in[63:56]),
                        .s(Test),           //显示信号选择, Test=SW[7:5] 控制
                        .o(point_out)      //七段码显示闪烁位指示
                        );

always@(posedge clk )begin
    if(EN) begin
        disp_data    <= Data0;           //Data0
        cpu_blink    <= blink_in[7:0];
        cpu_point    <= point_in[7:0];
    end
    else begin
        disp_data    <= disp_data;
        cpu_blink    <= cpu_blink;
        cpu_point    <= cpu_point;
    end
end
```


Multi_8CH32调用信号关系



```
Multi_8CH32    U5( .clk(clk_io), .rst(rst),  
                . EN(EN),                      //仅控制通道0  
                . point_in(????????),          //外部输入  
                .LES(????????),                //外部输入  
                .Test(SW_OK[7:5]),              //来自开关去抖  
                .data0(????????????),          //通道0输入  
                .data1(????????????)           //通道1输入  
                .data2(????????????????),      //通道2输入  
                .data3(????????????????),      //通道3输入  
                .data4(????????????????),      //通道4输入  
                .data5(????????????????),      //通道5输入  
                .data6(????????????????),      //通道6输入  
                .data7(????????????????),      //通道7输入  
  
                .point_out(point_out),          //输出到显示模块  
                .blink_out(LE_out),             //输出到显示模块  
                .disp_num(disp_num)            //输出到显示模块  
                );
```



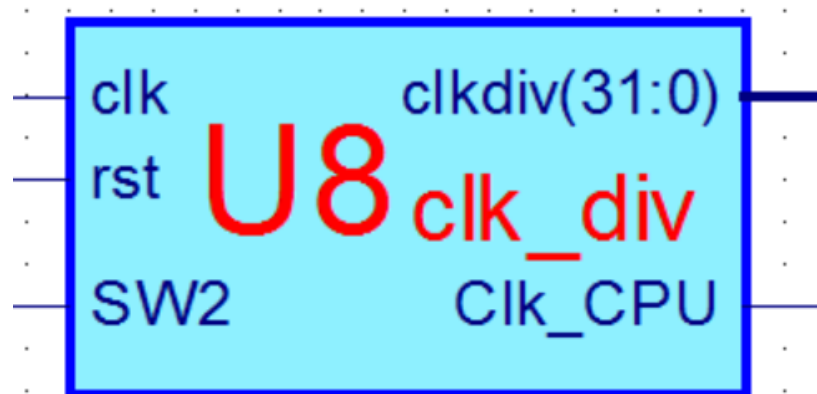
逻辑实验通用分频模块M1优化: `clk_div.v`

□ 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
- 器件编号改为**U8**

□ 基本功能

- 32位计数分频输出: `clkdiv`
- CPU时钟输出: `Clk_CPU`
- `SW[2]`控制`Clk_CPU`输出
 - `SW[2]=0`, 全速频率 (50MHz或25MHz)
 - `SW2=[1]`, 单步频率 (2^{24} 分频, `clkdiv [24]`)
- 核模块符号文档: `clk_div.sym`



`clk_div.sym`

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,                //主板时钟
               input rst,                //复位信号
               input SW2,                //CPU时钟切换
               output reg [31:0]clkdiv,   //32位计数分频输出
               output Clk_CPU            //CPU时钟输出
               );

    always @ (posedge clk or posedge rst) begin
        if (???) clkdiv <= ? ; else clkdiv <= ????????????; end
    assign Clk_CPU=(???) ? clkdiv[24] : clkdiv[2];

endmodule
```



开关去抖动模块M2优化: SAnti_jitter.v

□ 开关机械抖动消除模块(IP Core)

- 用于计算机组成实验辅助模块
- 逻辑实验去抖动模块改造
- 器件编号改为**U9**
- **Sword**平台提供**U9**的IP核

□ 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**(仅Button)
 - **RSTN**: 短按=CR, 长按=rst
 - 其余功能不作要求(阵列键盘属接口课内容)
- 核模块符号文档: SAnti_jitter.sym



开关去抖动模块端口信号

□ 去抖模块端口信号

- 可作为IP核调用空文档：端口文档

```
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



双32位数据输入IP核M4: SEnter_2_32

□ 32位数据输入模块(IP Core)

- 逻辑实验的辅助模块
- 本课程用于部件调试的初始值输入
- 器件编号改为**U10**
- **Sword**平台提供**U10**的IP核

□ 基本功能

- 输入32位二进制数据(SW[15]=0)
 - Inc单键输入: BTN(2)
 - 输出: Ai、Bi
 - 对应显示通道0(SW[7:5]=000)、通道1(SW[7:0]=001)
 - BTN(0)=左移、BTN(1)=右移、修改位由blink指示闪烁

□ 扩展功能

- 阵列式按键扫描码输入: 由SAnti_jitter模块输出Key_out
- Din=5位扫描码, D_ready=1按键有效、readn=0读扫描码
- 核模块符号文档: SEnter_2_32.sym



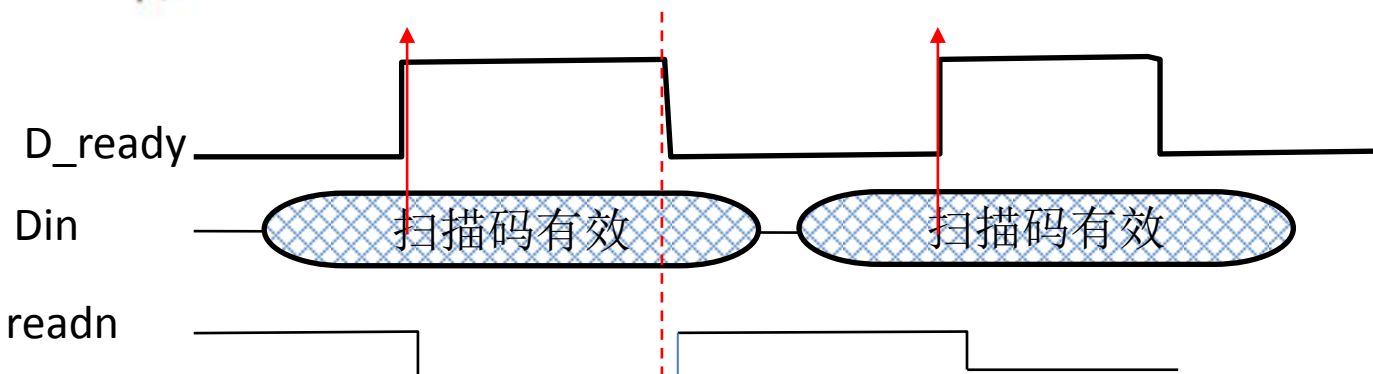
双32位数据输入模块端口信号

□ 双32位数据输入端口信号

- 可作为IP核调用空文档：端口文档

```
module SEnter_2_32(input clk,
                  input[2:0] BTN,           //对应SAnti_jitter列按键
                  input [4:0] Ctrl,         //{SW[7:5],SW[15],SW[0]}
                  input D_ready,           //对应SAnti_jitter扫描码有效
                  input [4:0]Din,
                  output reg readn,         //=0读扫描码
                  output reg[31:0]Ai=32'h87654321, //输出32位数一: Ai
                  output reg[31:0]Bi=32'h12345678, //输出32位数二: Bi
                  output reg [7:0 ]blink    //单键输入指示
);
```

endmodule



七段码显示器IP核M3: SSeg7_Dev



□ 8位七段码显示器(IP Core)

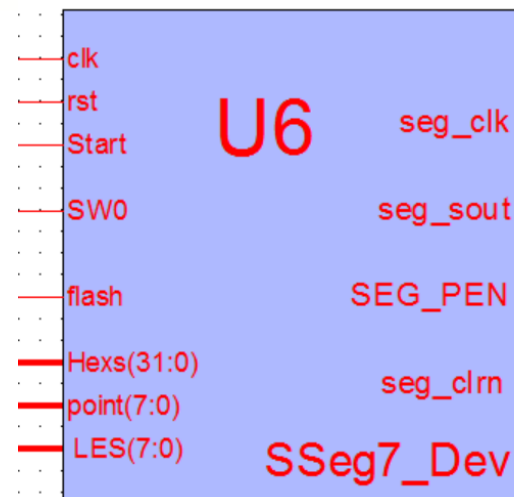
- 逻辑实验的输出显示模块
- 本课程用于调试显示和CPU的简单外设
- 器件编号改为**U6**

□ 基本功能

- 输入32位二进制数据: Hexs
 - SW[0]=1, 显示8位16进制数, SW[0]=0, 显示七段码LED点阵
 - SW[0]=1时: SW[1]=1高16位, SW[1]=0低16位,
 - flash七码闪烁频率, 由通用分频器U8(Div[25])提供, Start串行扫描启动, point: 七段小数点, LES: 七段码使能, 闪烁指示
- 串行输出: seg_clk=时钟, seg_out=串行七段显示数据, SEG_PEN=使能, seg_clrn=清零

□ 核模块符号文档: SSeg7_Dev.sym

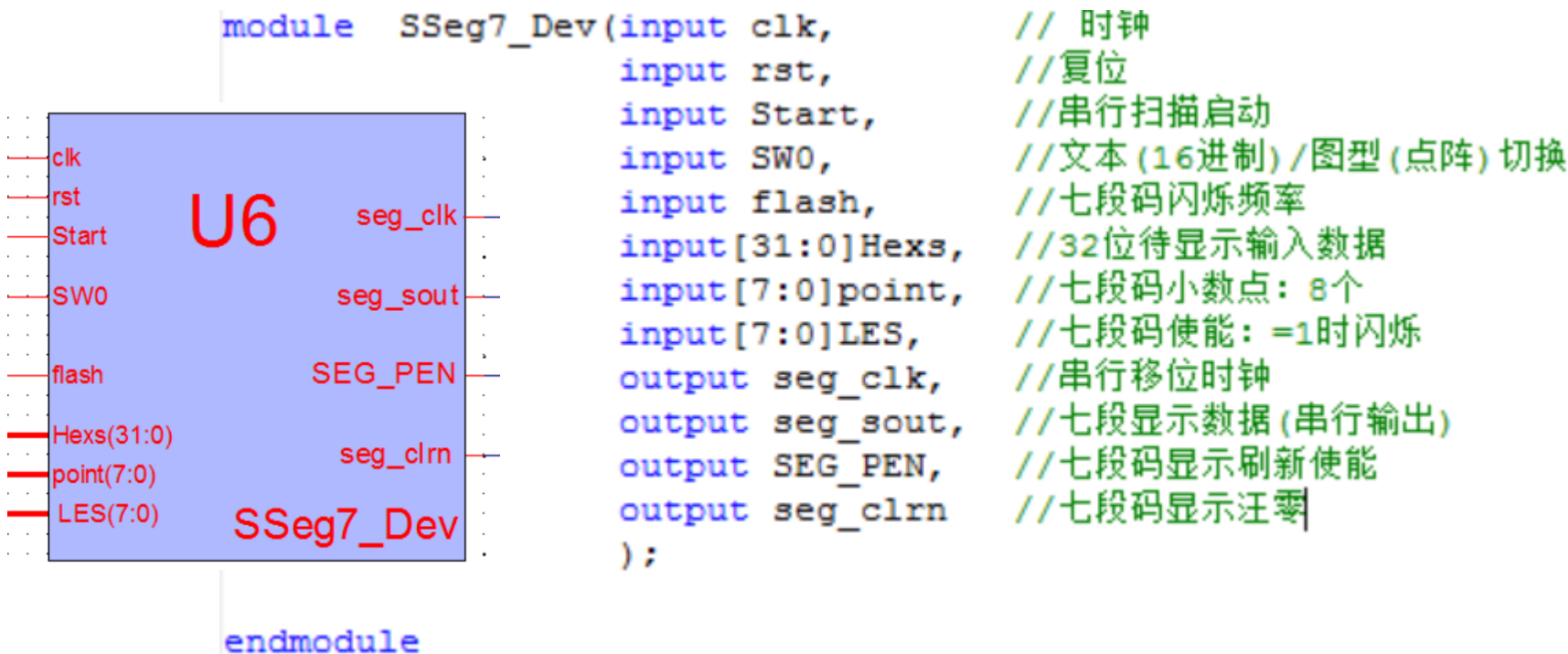
- 由实验二优化扩展, 本实验提供**U6**的IP核



七段码显示器IP核端口信号

□ 七段码显示器IP核端口信号

- 可作为IP核调用空文档：端口文档





LED并行显示模块M6: SPIO

□ 15位LED指示灯控制(IP Core)

- 逻辑实验的输出LED显示模块
 - 相当于通用输入输出接口: GPIO
 - 15位用于LED指示控制, 其余用于扩展

- 器件编号改为**U7**

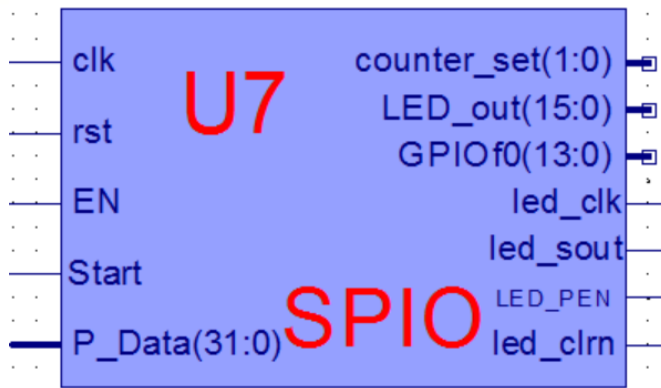
- 本课程用于调试显示和CPU的简单外设

□ 基本功能

- 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

□ 核模块符号文档: SPIO.sym

- 本实验提供**U7**的IP核





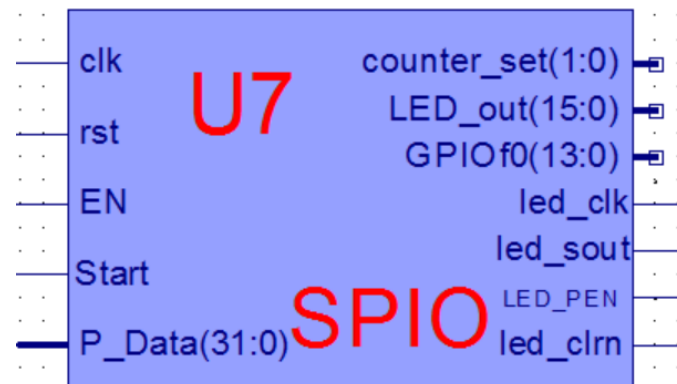
LED并行显示模块IP核端口信号

□ PIO/LED-GPIO IP核端口信号

- 可作为IP核调用空文档：端口文档

```
module      SPIO(input clk,                //时钟
                  input rst,              //复位
                  input Start,            //串行扫描启动
                  input EN,               //PIO/LED显示刷新使能
                  input [31:0] P_Data,    //并行输入，用于串行输出数据
                  output reg[1:0] counter_set, //用于计数/定时模块控制，本实验不用
                  output [15:0] LED_out,  //并行输出数据
                  output wire led_clk,    //串行移位时钟
                  output wire led_sout,   //串行输出
                  output wire led_clrn,   //LED显示清零
                  output wire LED_PEN,    //LED显示刷新使能
                  output reg[13:0] GPIOf0 //待用：GPIO
                );

endmodule
```



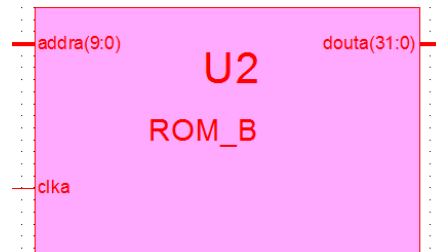
只读存储器IP核M14-1优化:

ROM_32_32



□ 只读存储器

- 用于CPU应用的代码存储器
- 逻辑实验M14-1模块优化
- 模块名改为ROM_B
- 器件编号改为U2



□ 基本功能

- 容量: $1024 \times 32\text{bit}$
- 用FPGA内部存储器实现
 - Block Memory Generator或Distributed Memory Generator
- 核模块符号文档: ROM_B.sym
 - 自动生成符号不规则, 需要修整
- ROM初始化文档暂时不变

用Distributed Memory 没有clk信号
需要编辑删除clka引脚

□ 用ISE工具生成固核

- 用IP Core Generator向导生成
- 核调用模块ROM_B.xco



ROM_B调用端口信号

□ ROM调用接口信号

```
ROM_B    U2 ( .clka(clk_m),           //存储器时钟，主板时钟取反
              .addra(addra),          // 通用分频器输出
              .douta(dispa6)          // ROM输出，至M5显示通道7
              );
addra={ N0,N0,N0,N0,N0,SW(3),clk_div(27:24) }
```

□ 图形输入调用

■ ROM_B.sym



随机存储器IP核M14-2优化: RAM_32_32

□ 随机存储器

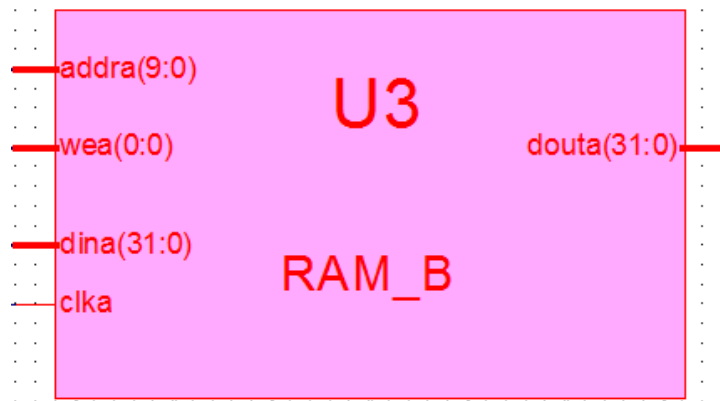
- 用于CPU应用的数据或代码存储器
- 逻辑实验U14-2模块优化
- 模块名改为RAM_B
- 器件编号改为U3

□ 基本功能

- 容量: $1024 \times 32\text{bit}$
- 用FPGA内部存储器实现
 - Block Memory Generator
- 核模块符号文档: RAM_B.sym
 - 自动生成符号不规则, 需要修整
- RAM初始化文档无

□ 用ISE工具生成固核

- 用IP Core Generator向导生成
- 核调用模块RAM_B.xco





RAM_B调用端口信号

□ RAM调用接口信号

```
RAM_B    U3 ( .clka(clk_m),           // 存储器时钟，主板时钟取反
              .wea(SW_OK[4]),         // 存储器读写，来自MIO_BUS
              .addra(addra),          // 通用分频器输出
              .dina(disp6),           // 输入数据线，来自ROM_B
              .douta(disp7)           // 输出数据线，至M5显示通道7
              );

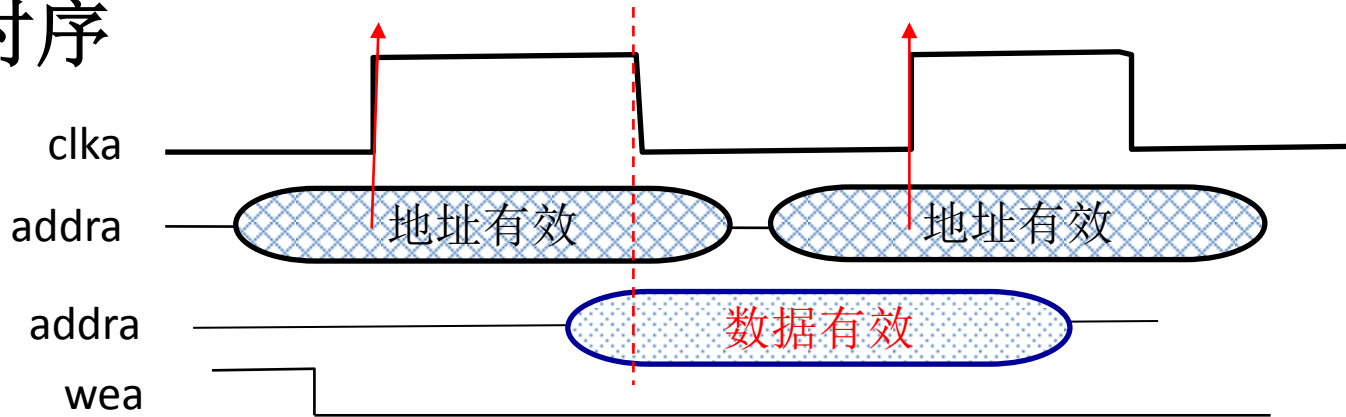
          addra={N0,N0,N0,N0,N0,SW(3),clk_div(27:24)}
```

□ 图形输入调用

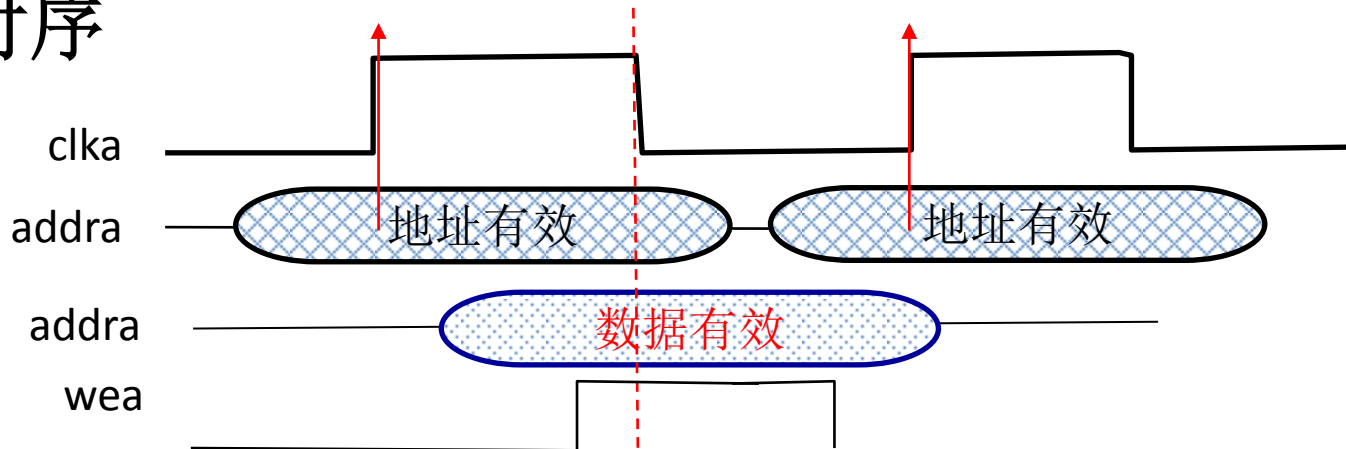
■ RAM_B.sym

Block Memory 时序

□ 读时序



□ 写时序



Course Outline



设计工程一：OExp01-Element



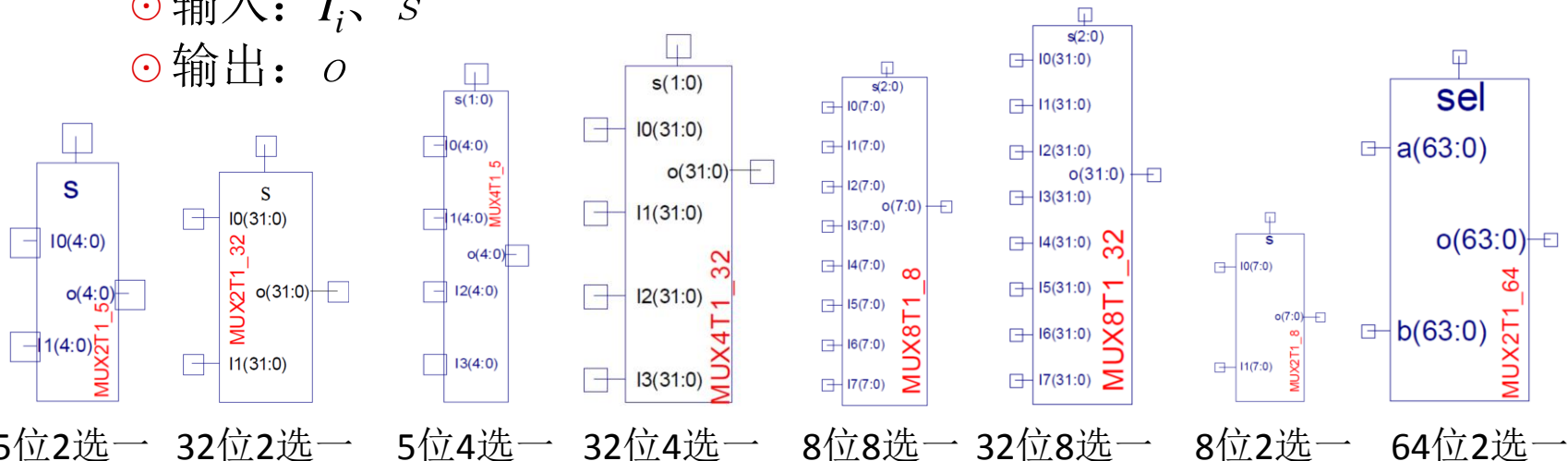
本工程仅做仿真，课外作业

◎ 设计、整理和优化逻辑课实验输出基本逻辑模块

☞ 多路选择器：

⊙ 输入： I_i 、 s

⊙ 输出： o



☞ 多路选择器仿真验证

⊙ 时序仿真激励要点：

◆ 对输入通道作遍历，测试参数用A、5

⊙ 仿真通过后封装，名称：MUX?T1_?，如MUX2T1_8=8位2选一

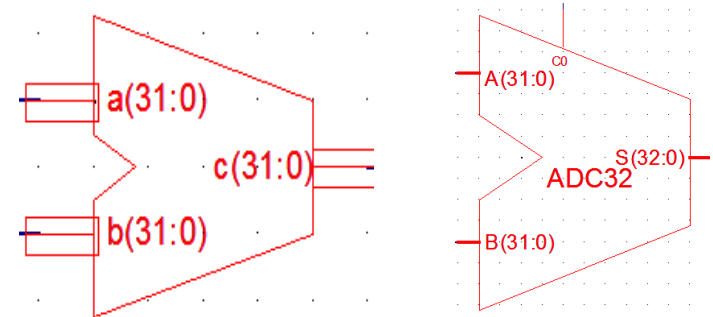
㉔ 算术逻辑函数

⊙ 32位加法器：add32(无进位)

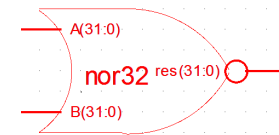
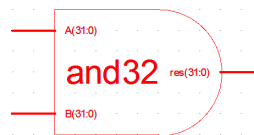
◆ 后期用于有效地址计算

⊙ 32位加减器：ADC23

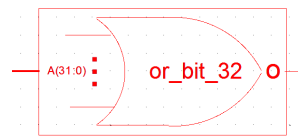
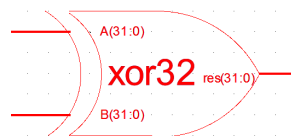
◆ 用于ALU的加减运算



⊙ 32位“与”、“或”、“或非/非”运算



⊙ 32位“异或”、“位或”、“右移”



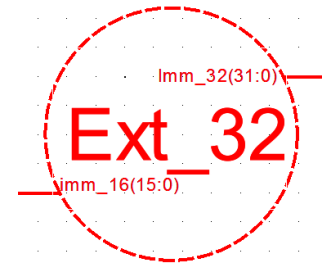
㉔ 算术逻辑函数仿真验证

⊙ 时序仿真激励要点：根据运算特征抽样

⊙ 仿真通过后封装

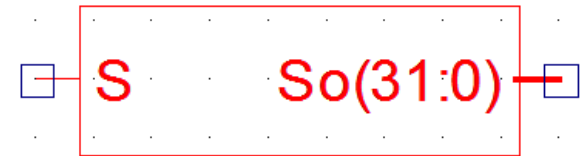
☞ 数据和信号位扩展函数

- 符号数扩展: Ext_32
 - ◆ 16位符号数扩展为32位
- 无符号数扩展: UExt_32
 - ◆ 16位无符号数扩展为32位
- 单信号扩展: SignalExt_32
 - ◆ 一位信号扩展为32位



☞ 位扩展函数仿真

- 时序仿真激励要点: 选择正数、负数
- 仿真通过后封装

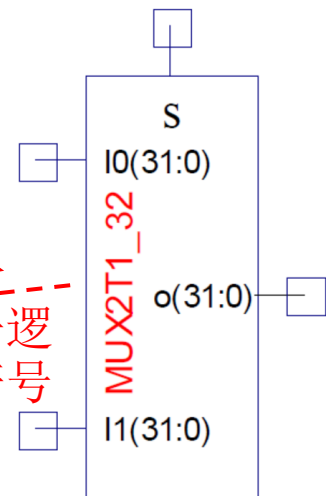


◎ 图型模块调用关系:

☞ 以32位2选一为例

Name	Value	Visible	
InstName	MUX3	<input checked="" type="checkbox"/>	New
SymbolName	MUX2T1_32	<input checked="" type="checkbox"/>	Edit Traits
VeriModel	MUX2T1_32	<input type="checkbox"/>	Delete

模块名对应
双击逻辑符号



拷贝模块的Symbol文件到当前工程根目录:

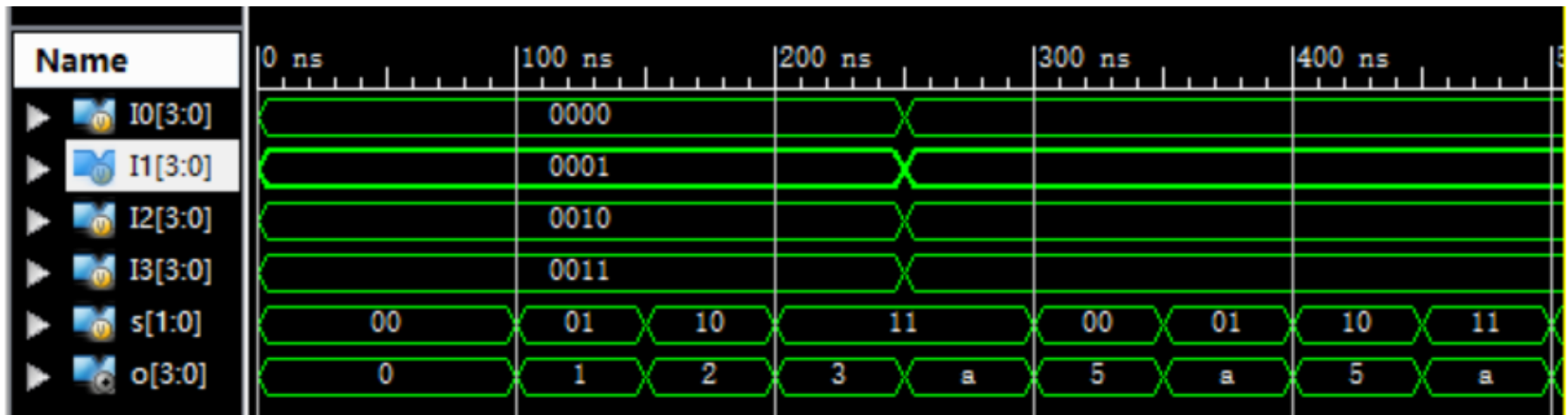
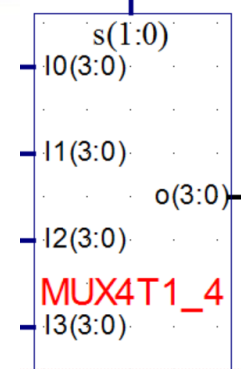
MUX2T1_5、MUX4T1_5、MUX2T1_32、MUX4T1_32
MUX2T1_8、MUX8T1_8、MUX8T1_32、MUX2T1_64
add_32、ADC32、and32、or32、nor32、srl32、xor32、
Ext_32.sym、SignalExt_32.sym、or_bit_32.sym

模块逻辑符号可以自制

仿真参考：以4选1为例

参考激励：

```
initial begin
    s = 0;      #50;      s = 0;
    I0 = 0;     #50;      s = 2;
    I1 = 1;     #50;      s = 1;
    I2 = 2;     #50;      s = 3;
    I3 = 3;     #50;      s = 2;
    #50;        I0 = 4'h5; #50;
    s = 0;       I1 = 4'hA; #50;
    #50;         I2 = 4'h5; #50;
    s = 1;       I3 = 4'hA; #50;
    #50;         s = 0;
    #50;         end
```



◎ 学习RTL综合电路后描述

☞ 打开View RTL Schematic分析学习HDL代码综合后电路描述



设计工程二：OExp01-MUX

◎ 设计八数据通路模块：Multi_8CH32

◎ 设计32位存储器

⌚ ROM: 32×1024 : ROM_D

⌚ RAM: 32×1024 : RAM_B

⊙ B = Block Memory

⊙ D = Distributed Memory

◎ 搭建物理验证输入输出平台

⌚ 调用核或已设计模块实现

⊙ 开关去抖模块(IP核): U9

⊙ 数据输入模块(IP核): M4

⊙ 通用分频模块(clk_div): U8

⊙ 八数据通路模块(Multi_8CH32): U5

⊙ 七段显示模块(SSeg7_Dev 核): U6

⊙ LED显示模块(SPIO核模块): U7



设计要点

◎ 新建工程：OExp01-MUX

◎ 设计八数据通路模块：Multi_8CH32

☞ 参考逻辑实验七

☞ 调用工程一设计的多路选择器

☞ 用HDL结构化调用和行为混合描述实现

☞ 仿真测试

◎ 激励要点：参考8选1：

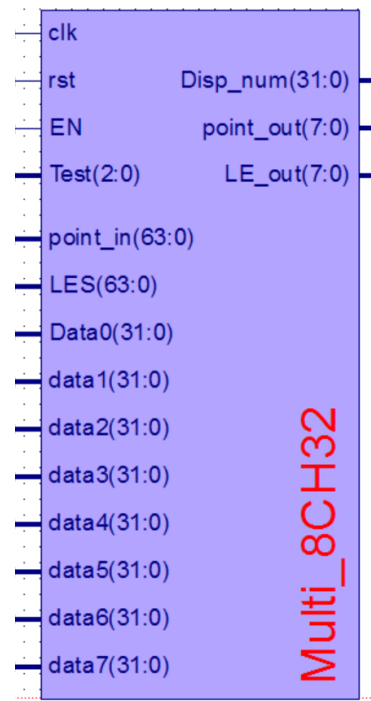
- ◆ 相当于3个8选1共享选择端控制信号
- ◆ 复位测试
- ◆ 使能控制

☞ 仿真通过后封装逻辑符号

◎ Multi_8CH32.sym

☞ 学习RTL综合电路后描述

◎ 打开View RTL Schematic分析学习HDL代码综合后电路描述



◎ 设计32位存储器：参考逻辑实验五、十四

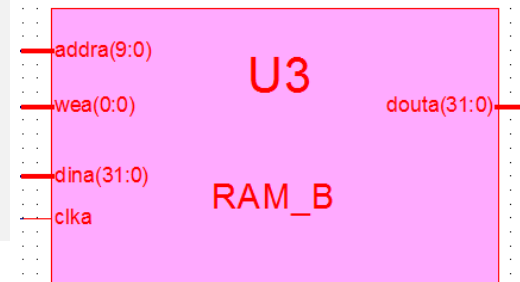
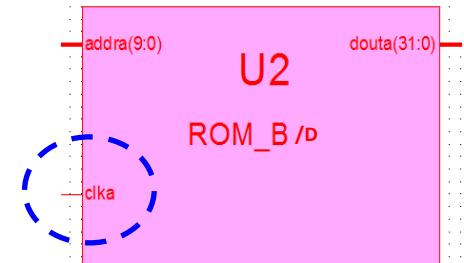
☞ Sword实验平台 ROM用Distributed Memory
RAM用Block Memory

☞ ROM初始化数据：ROM.coe

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

```
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;
```

红色数据是七段LED图形



◎ 完成后修改工程ipcore_dir目录中的模块符号，或替换成自己的



◎ 搭建物理验证输入输出平台

☞ 拷贝需要的模块Symbol文件到当前工程根目录

- ◎ SAnti_jitter.sym(U9): 开关按钮预处理模块
- ◎ SEnter_2_32.sym(U10): 双32位输入模块
- ◎ Multi_8CH32.sym(U5): 八通道选择模块(用于显示)
- ◎ clk_div.sym(U8): 通用分频模块
- ◎ SSeg7_Dev.sym(U6): 七段显示器
- ◎ SPIO.sym(U7): LED/并行输出模块
- ◎ ROM_B.sym(U2): 程序存储器
 - ◆ 删除ipcore_dir目录中的ROM_B.sym文件
- ◎ RAM_B(U3): 主存储器
 - ◆ 删除ipcore_dir目录中的RAM_B.sym文件

◎ 新建顶层模块输入模板: OExp01_MUX.sch

☞ 参考附录输入顶层逻辑电路描述

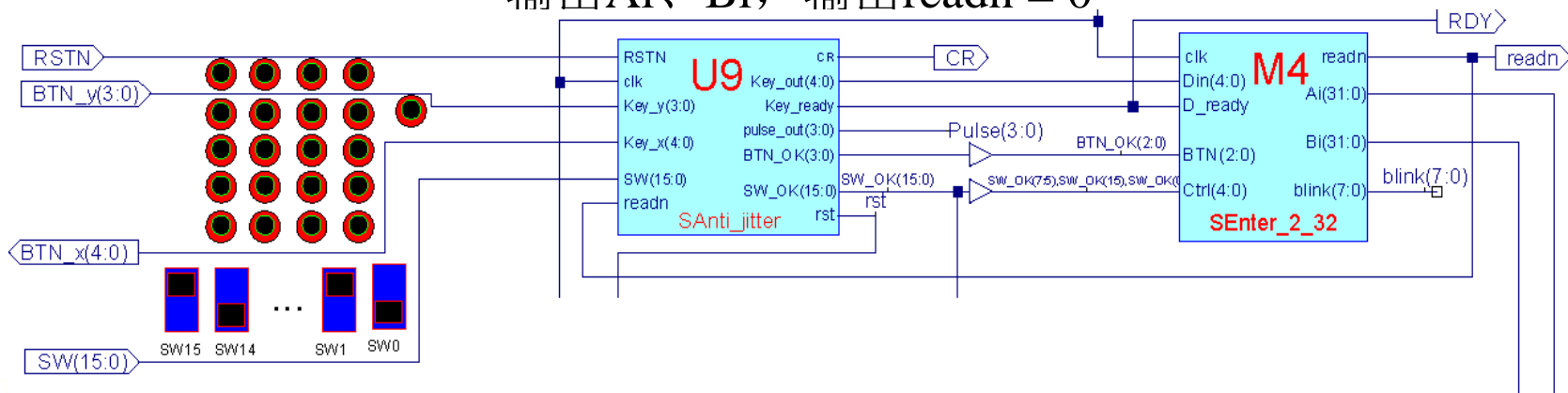
阵列式按键预处理与输入模块连接

◎ 阵列式按键

- ☞ 列输入信号：BTN_y → 去抖处理输出BTN_Ok, pulse_out
- ☞ 行输出信号：BTN_x → 扫描输出Key_out → 0000-1111
- ☞ 阵列扫描码读取握手信号：Key_ready、readn
- ☞ RSTN去抖后：CR=短按RSTN, rst=长按RSTN
- ☞ SW去抖后输出SW_OK

◎ 双32位数据输入模块

- ☞ SW_OK(15)=0: 读取BTN_OK(2:0), 输出Ai、Bi
- ☞ SW_OK(15)=1: D_ready=Key_ready=1, Din读取Key_out, 输出Ai、Bi, 输出readn = 0



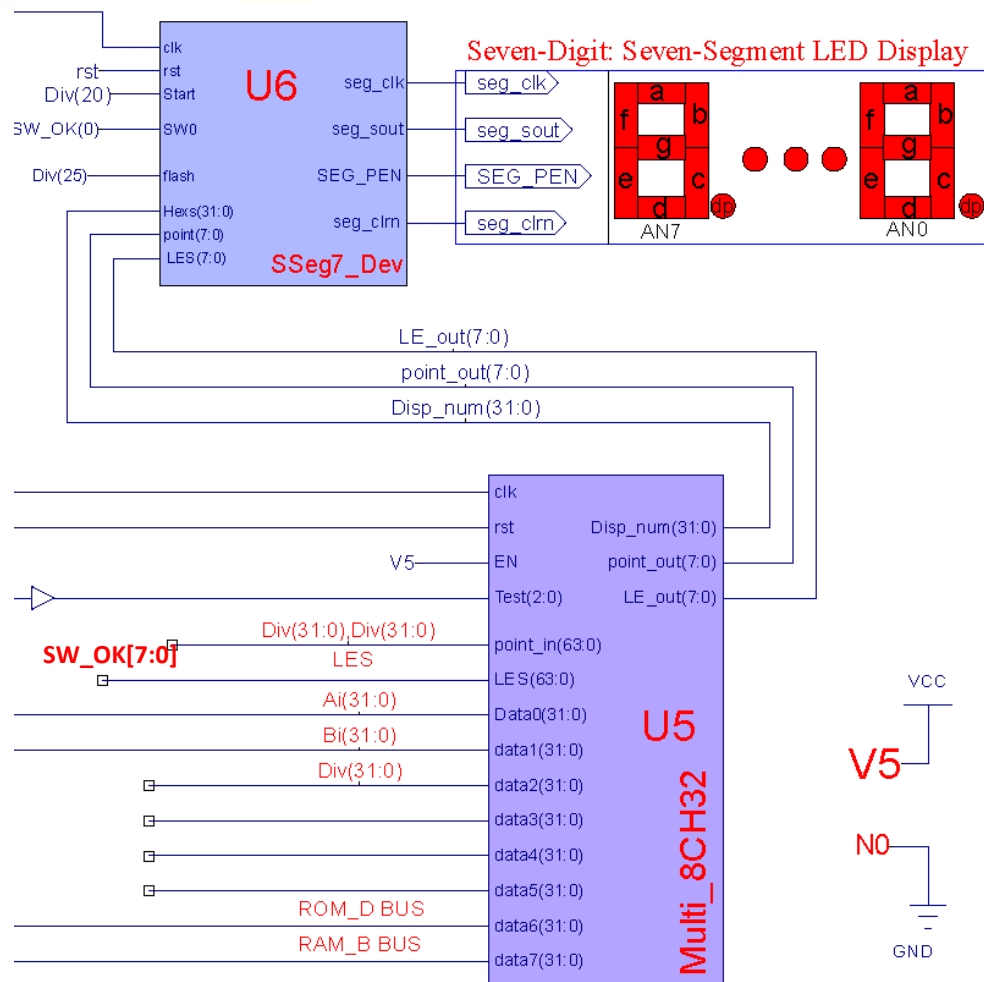
显示信号与七段显示器连接

◎ 七段显示器连接

- ☞ clk=系统时钟
- ☞ rst=长按RSTN
- ☞ Start=Div(20)
- ☞ Hexs=Disp_num
- ☞ point=point_out
- ☞ LES=LE_out

◎ 显示通道连接

- ☞ clk=系统时钟
- ☞ rst=长按RSTN
- ☞ EN=1: 本实验不用
- ☞ Test=SW_OK[7:5]
- ☞ Data0~7: 连接待显示信号



存储器连接

◎ 地址线

☞ ROM、RAM地址线相同

☞ $addre = \{5'b00000, SW[3], Div[27:24]\}$

⊙ $SW[3]=0$ 输出0~F ($SW[0]=1$)

⊙ $SW[3]=1$ 输出点阵 ($SW[0]=0$)

◎ 数据线

☞ ROM输出连接显示通道6,
同时输入RAM

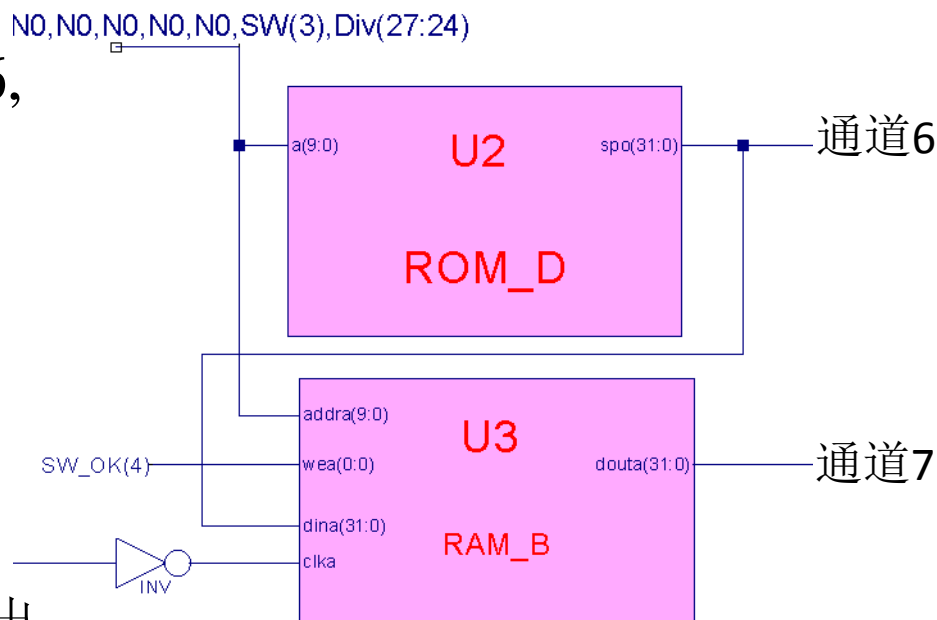
☞ RAM输出数据线连接显
示通道7

◎ 控制线

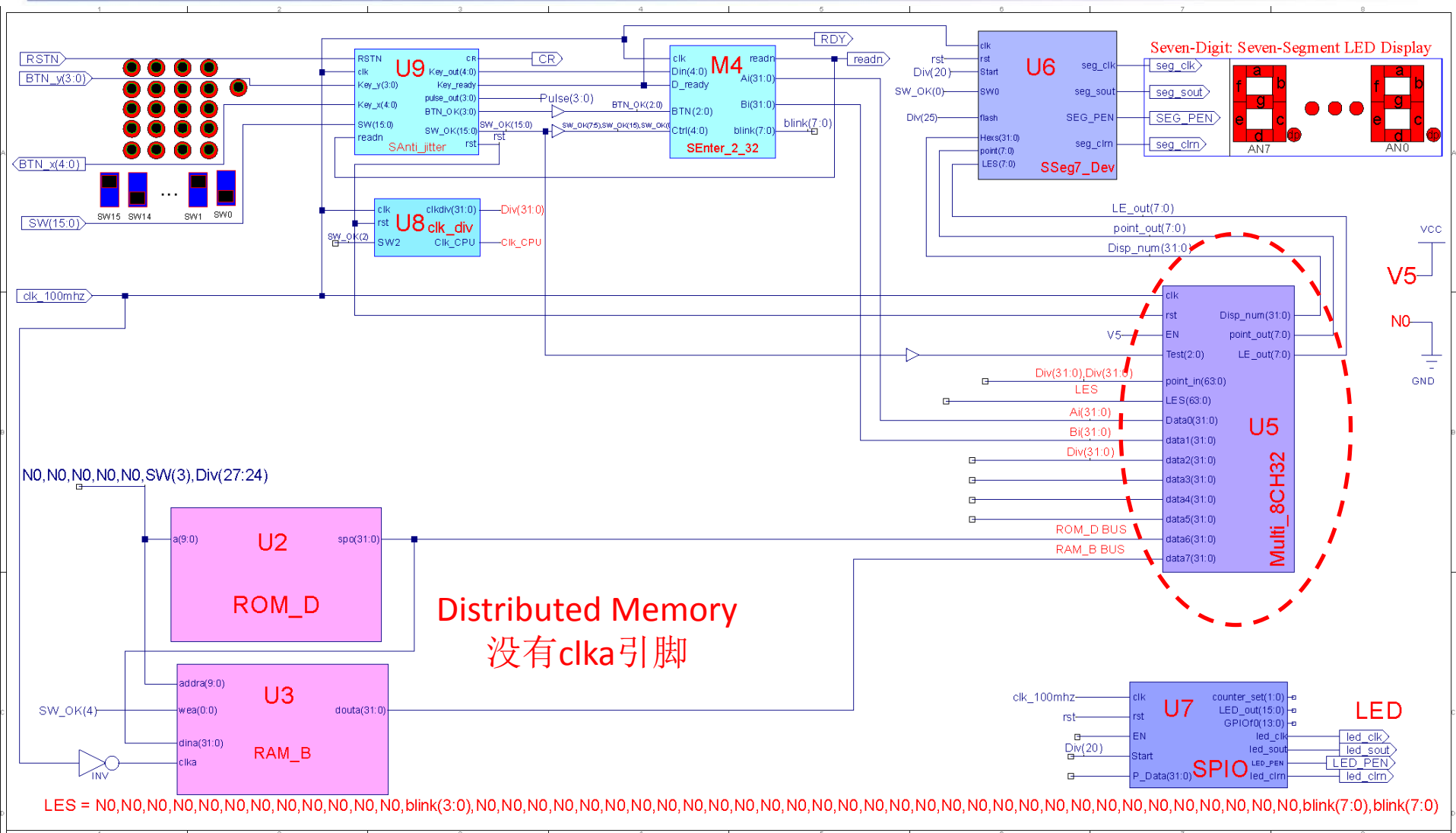
☞ $clka = \sim clk_100MHz$

☞ $wea = SW_OK(4)$

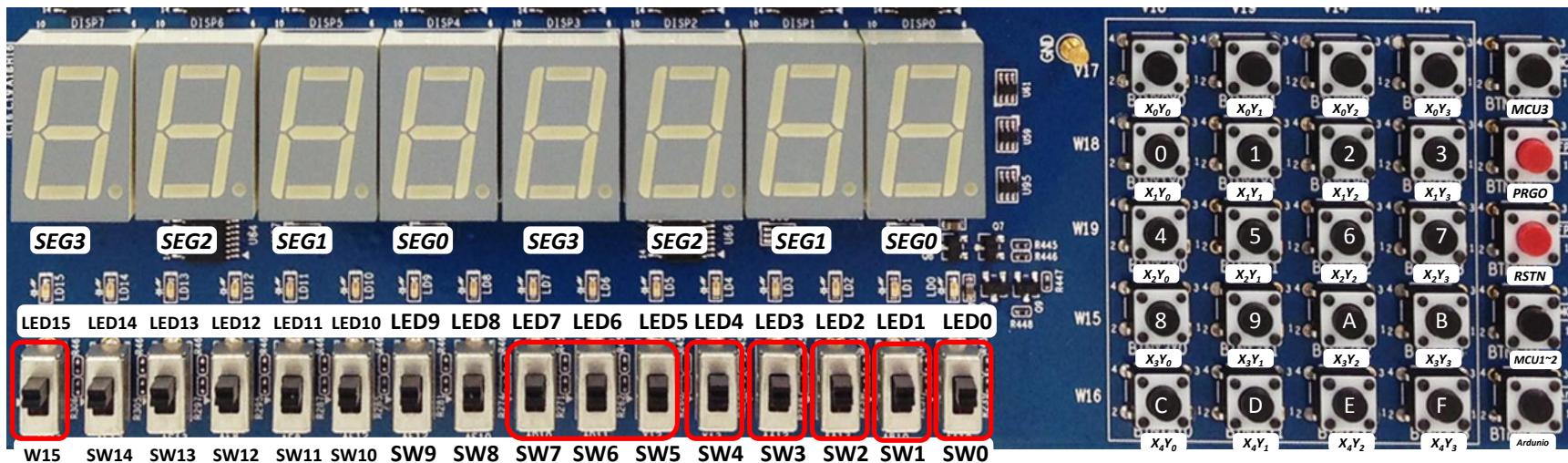
⊙ $SW_OK(4)=1$, RAM有输出



实验一顶层模块逻辑结构：SWORD平台



物理验证-板级GPIO接口功能



U10按键输入方式选择

SW[15]=0, BTN[3:0]

SW[15]=1, Key_out[5:0]

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[3]=ROM地址段选择

SW[4]=RAM写信号

SW[7:5]=显示通道选择

SW[15]=0:

$X_iY[3:0]=BTN[3:0]$

SW[15]=1:

$Key_out=Y[3:0]+i*4$



用户约束

□ UCF引脚定义

#系统时钟

```
NET "clk_100mhz"      LOC = AC18      | IOSTANDARD = LVCMOS18 ;
NET "RSTN"            LOC = W13       | IOSTANDARD = LVCMOS18 ;
NET "clk_100mhz"      TNM_NET = TM_CLK ;
TIMESPEC TS_CLK_100M = PERIOD "TM_CLK" 10 ns HIGH 50%;
```

#LED串行接口

```
NET "led_clk"         LOC = N26       | IOSTANDARD = LVCMOS33 ;
NET "led_clrn"        LOC = N24       | IOSTANDARD = LVCMOS33 ;
NET "led_sout"        LOC = M26       | IOSTANDARD = LVCMOS33 ;
NET "LED_PEN"         LOC = P18       | IOSTANDARD = LVCMOS33 ;
```

#七段码串行接口

```
NET "seg_clk"         LOC = M24       | IOSTANDARD = LVCMOS33 ;
NET "seg_clrn"        LOC = M20       | IOSTANDARD = LVCMOS33 ;
NET "seg_sout"        LOC = L24       | IOSTANDARD = LVCMOS33 ;
NET "SEG_PEN"         LOC = R18       | IOSTANDARD = LVCMOS33 ;
```

#三色信号灯: Tri_LED

```
NET "RDY"             LOC = U21       | IOSTANDARD = LVCMOS33 ;#LED_nR0
NET "readn"           LOC = U22       | IOSTANDARD = LVCMOS33 ;#LED_nG0
NET "CR"              LOC = V22       | IOSTANDARD = LVCMOS33 ;#LED_nB0
#NET "LED_nR1"        LOC = U24       | IOSTANDARD = LVCMOS18 ;
#NET "LED_nG1"        LOC = U25       | IOSTANDARD = LVCMOS18 ;
#NET "LED_nB1"        LOC = V23       | IOSTANDARD = LVCMOS18 ;
```



#阵列式按键

```
NET "BTN_x[0]"      LOC = V17      | IOSTANDARD = LVCMOS18 ;#ROW0
NET "BTN_x[1]"      LOC = W18      | IOSTANDARD = LVCMOS18 ;#ROW1
NET "BTN_x[2]"      LOC = W19      | IOSTANDARD = LVCMOS18 ;#ROW2
NET "BTN_x[3]"      LOC = W15      | IOSTANDARD = LVCMOS18 ;#ROW3
NET "BTN_x[4]"      LOC = W16      | IOSTANDARD = LVCMOS18 ;#ROW4
NET "BTN_y[0]"      LOC = V18      | IOSTANDARD = LVCMOS18 ;#COL0
NET "BTN_y[1]"      LOC = V19      | IOSTANDARD = LVCMOS18 ;#COL1
NET "BTN_y[2]"      LOC = V14      | IOSTANDARD = LVCMOS18 ;#COL2
NET "BTN_y[3]"      LOC = W14      | IOSTANDARD = LVCMOS18 ;#COL3
```

#switch

```
NET "SW[0]"         LOC = AA10     | IOSTANDARD = LVCMOS15 ;
NET "SW[1]"         LOC = AB10     | IOSTANDARD = LVCMOS15 ;
NET "SW[2]"         LOC = AA13     | IOSTANDARD = LVCMOS15 ;
NET "SW[3]"         LOC = AA12     | IOSTANDARD = LVCMOS15 ;
NET "SW[4]"         LOC = Y13      | IOSTANDARD = LVCMOS15 ;
NET "SW[5]"         LOC = Y12      | IOSTANDARD = LVCMOS15 ;
NET "SW[6]"         LOC = AD11     | IOSTANDARD = LVCMOS15 ;
NET "SW[7]"         LOC = AD10     | IOSTANDARD = LVCMOS15 ;
NET "SW[8]"         LOC = AE10     | IOSTANDARD = LVCMOS15 ;
NET "SW[9]"         LOC = AE12     | IOSTANDARD = LVCMOS15 ;
NET "SW[10]"        LOC = AF12     | IOSTANDARD = LVCMOS15 ;
NET "SW[11]"        LOC = AE8      | IOSTANDARD = LVCMOS15 ;
NET "SW[12]"        LOC = AF8      | IOSTANDARD = LVCMOS15 ;
NET "SW[13]"        LOC = AE13     | IOSTANDARD = LVCMOS15 ;
NET "SW[14]"        LOC = AF13     | IOSTANDARD = LVCMOS15 ;
NET "SW[15]"        LOC = AF10     | IOSTANDARD = LVCMOS15 ;
```



#ArDUNIO-Sword-002-Basic IO

```
NET "Buzzer"          LOC = AF24 | IOSTANDARD = LVCMOS33 ;
NET "SEGMENT[0]"      LOC = AB22 | IOSTANDARD = LVCMOS33 ;#a
NET "SEGMENT[1]"      LOC = AD24 | IOSTANDARD = LVCMOS33 ;#b
NET "SEGMENT[2]"      LOC = AD23 | IOSTANDARD = LVCMOS33 ;
NET "SEGMENT[3]"      LOC = Y21  | IOSTANDARD = LVCMOS33 ;
NET "SEGMENT[4]"      LOC = W20  | IOSTANDARD = LVCMOS33 ;
NET "SEGMENT[5]"      LOC = AC24 | IOSTANDARD = LVCMOS33 ;
NET "SEGMENT[6]"      LOC = AC23 | IOSTANDARD = LVCMOS33 ;#g
NET "SEGMENT[7]"      LOC = AA22 | IOSTANDARD = LVCMOS33 ;#point
```

```
NET "AN[0]"           LOC = AD21 | IOSTANDARD = LVCMOS33 ;
NET "AN[1]"           LOC = AC21 | IOSTANDARD = LVCMOS33 ;
NET "AN[2]"           LOC = AB21 | IOSTANDARD = LVCMOS33 ;
NET "AN[3]"           LOC = AC22 | IOSTANDARD = LVCMOS33 ;
```

```
NET "LED[0]"          LOC = AB26 | IOSTANDARD = LVCMOS33 ;
NET "LED[1]"          LOC = W24  | IOSTANDARD = LVCMOS33 ;
NET "LED[2]"          LOC = W23  | IOSTANDARD = LVCMOS33 ;
NET "LED[3]"          LOC = AB25 | IOSTANDARD = LVCMOS33 ;
NET "LED[4]"          LOC = AA25 | IOSTANDARD = LVCMOS33 ;
NET "LED[5]"          LOC = W21  | IOSTANDARD = LVCMOS33 ;
NET "LED[6]"          LOC = V21  | IOSTANDARD = LVCMOS33 ;
NET "LED[7]"          LOC = W26  | IOSTANDARD = LVCMOS33 ;
```

```
#NET "PS2_clk"        LOC = N18   | IOSTANDARD = LVCMOS33 ;
#NET "PS2_data"       LOC = M19   | IOSTANDARD = LVCMOS33 ;
```




输入设备功能定义

开关定义	=0	=1	备注
SW[0]	图形(七段点阵)	文本(16进制)	
SW[1]	32位二进制高16位	32位二进制低16位	Arduino Sword 002
SW[2]	CPU全速时钟	CPU单步钟	CPU时钟切换
SW[4]	存储器写禁止	存储器写使能	存储单元写控制
SW[7:5]	=000	通道0	Ai
	=001	通道1	Bi
	=010	通道2	SUM(ALU_Out)
	=011	通道3	Sign extension
	=100	通道4	1 bit Ext. to 32 bits
	=101	通道5	通用分频输出
	=110	通道6	ROM_D输出
	=111	通道7	RAM_B输出D(31:0)
按键定义	=0	=1	备注
BTN[0]		正脉冲左移	SW[15]=0,SW[7:5]<=001
BTN[1]		正脉冲右移	SW[15]=0,SW[7:5]<=001
BTN[2]		正脉冲输入修改	SW[15]=0,SW[7:5]<=001
RSTN			长按复位

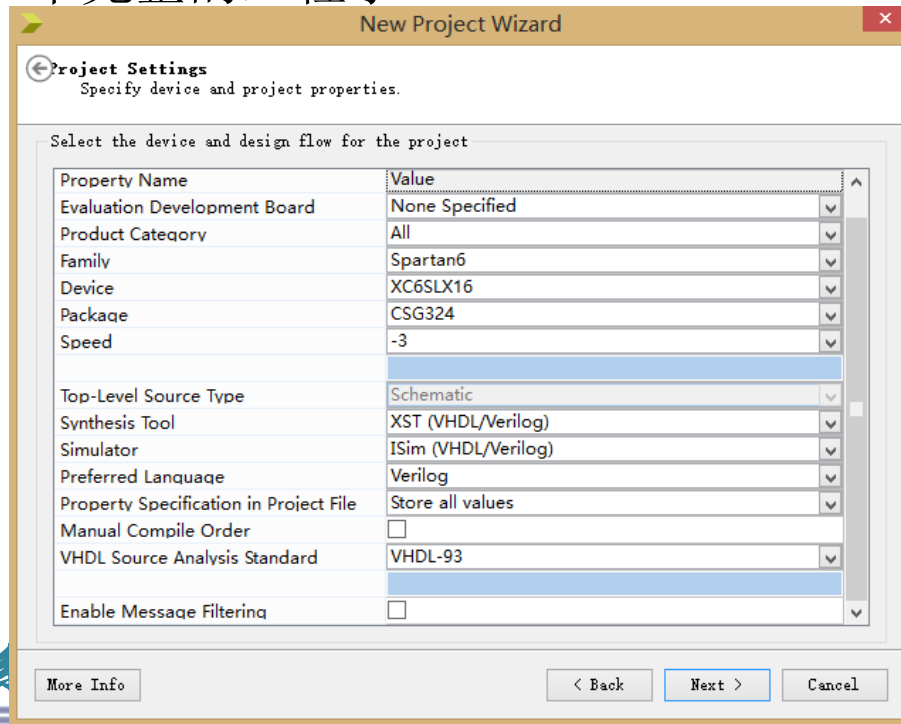
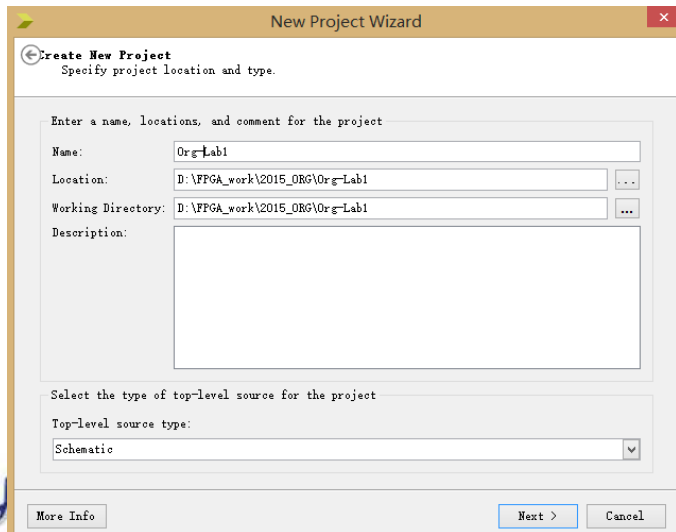
实验一顶层模块输入

建立ISE开发工程

□ 用ISE新建实验一工程

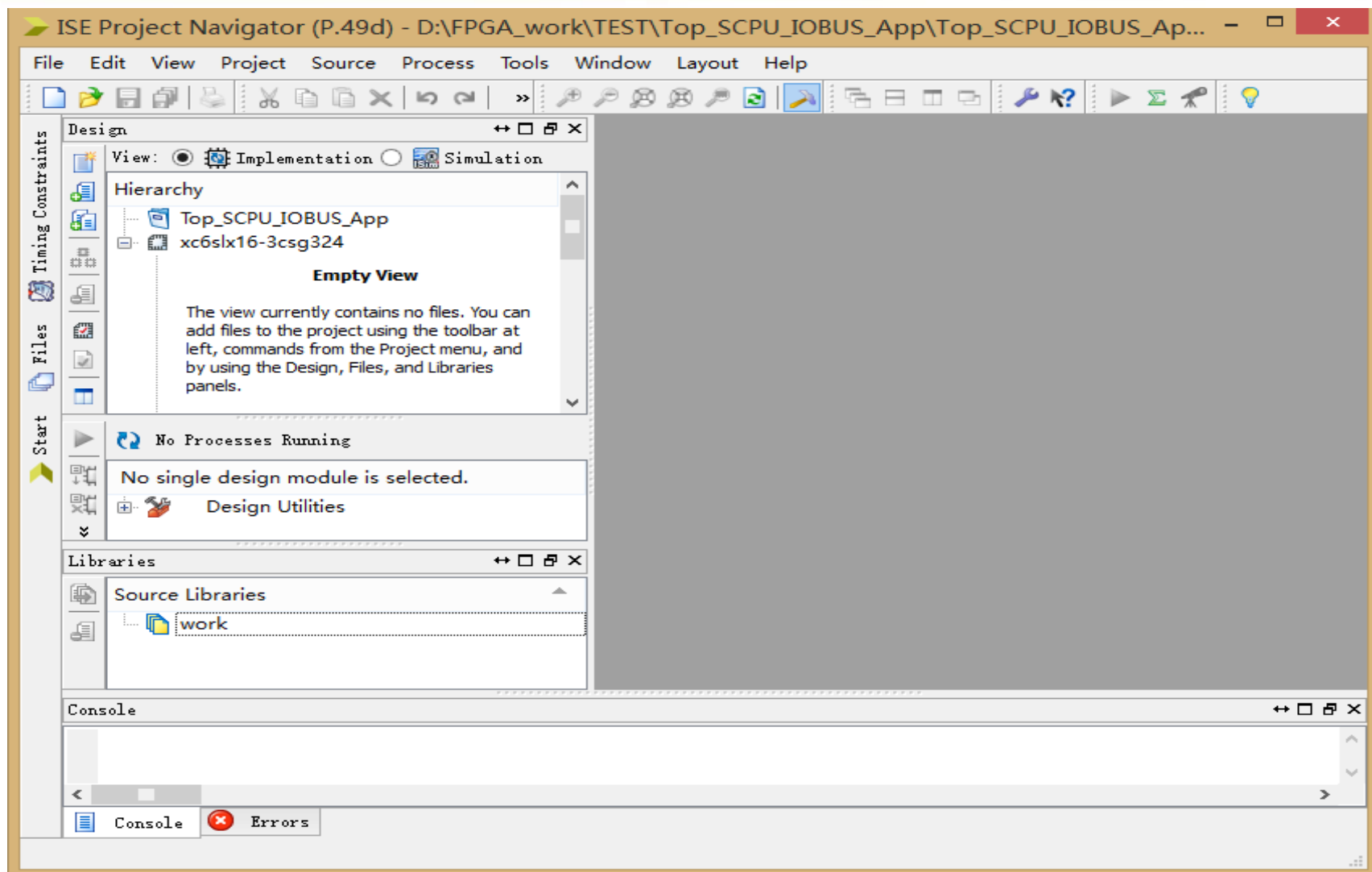
- 双击桌面上“Xilinx ISE”图标，启动ISE软件(也可从开始菜单启动)
- 选择File New Project选项，在弹出的对话框中输入工程名称并指定工程路径。参考工程名：**OExp01-MUX**
- 点击Next按钮进入下一页，选择所使用的芯片及综合、仿真工具。
- 再点击Next按钮进入下一页，这里显示了新建工程的信息，确认无误后，点击Finish就可以建立一个完整的工程了

□ 单周期CPU设计共享此工程





实验一工程模板





拷贝模块的Symbol文件到当前工程根目录：

开关按钮预处理模块：SAnti_jitter.sym(U9)

双32位输入模块：SEnter_2_32.sym(U10)

八通道选择模块：Multi_8CH32.sym(U5)

通用分频模块：clk_div.sym(U8)

七段显示器：SSeg7_Dev.sym(U6)

LED/并行输出模块：SPIO.sym(U7)

程序存储器：ROM_D.sym(U2)

主存储器：RAM_B(U3)



建立实验一顶层模块(原理图输入模板)

□ 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OExp01-???
- 建议修改为..\ **OExp01-??? \Code**

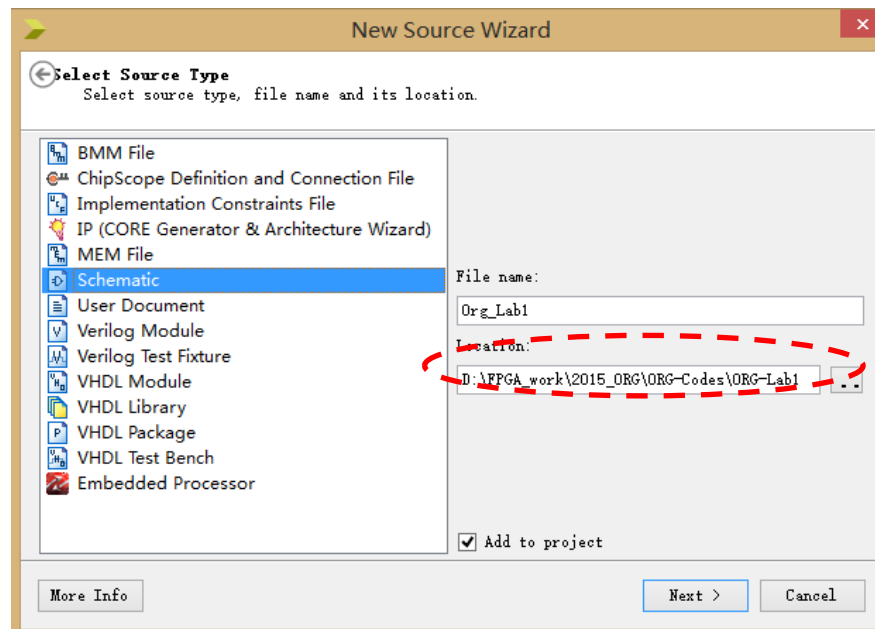
□ 注意：为了方便管理，将每个实验的代码存放在**独立目录**中！

- 同时注意同名.sch与.v文件的冲突

□ 点击Next生成

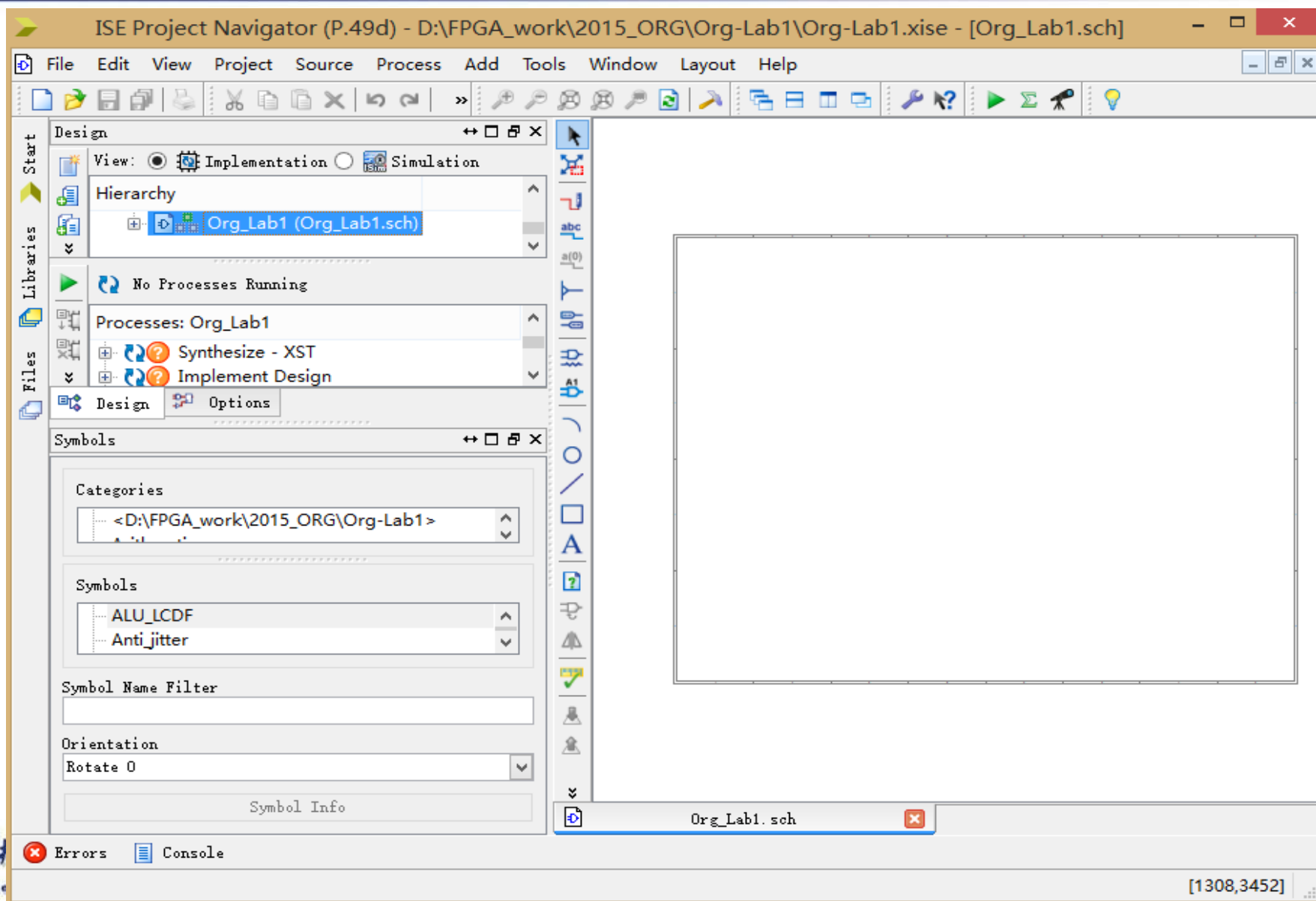
-原理图输入模板

- 根据顶层逻辑图输入





原理图输入窗口与环境





逻辑图输入顶层逻辑

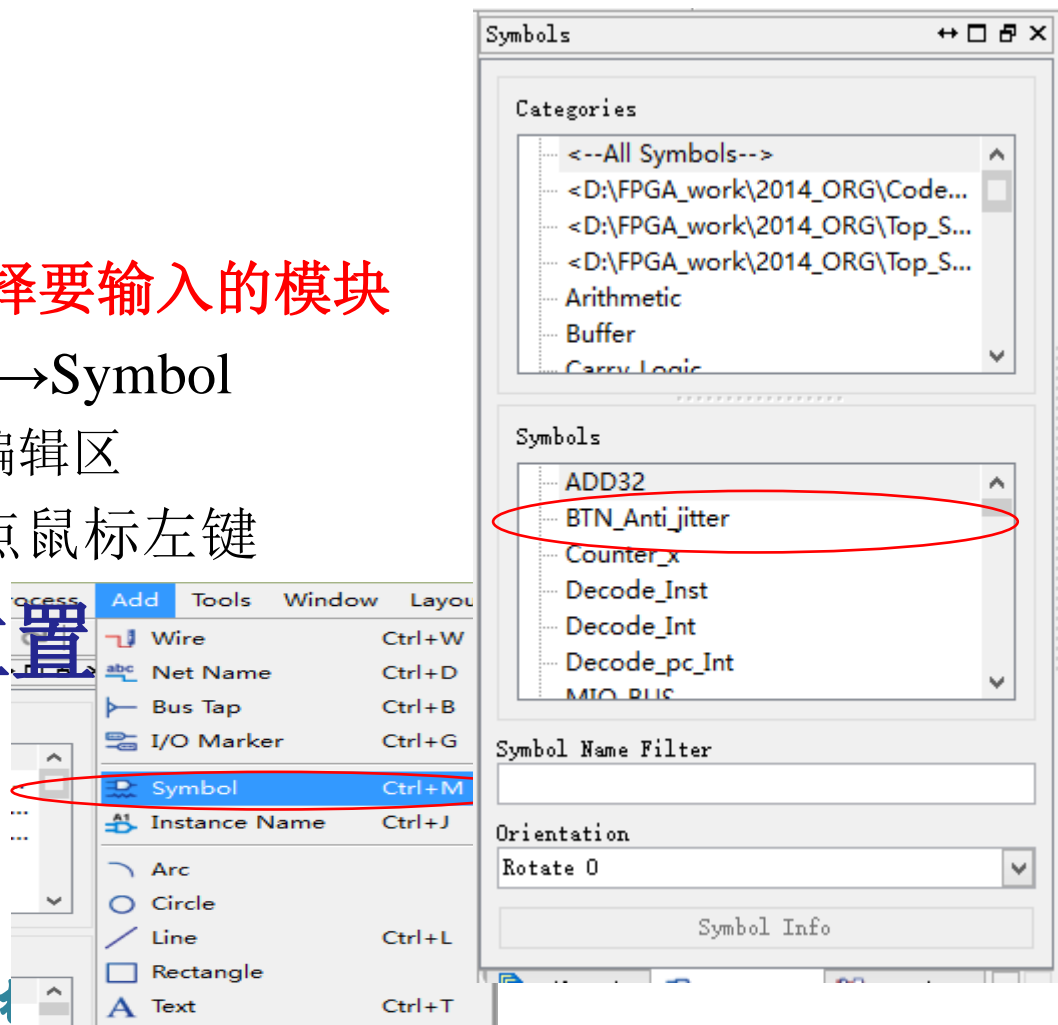
输入实验一顶层模块

□ 在原理图输入窗口输入顶层模块

- 激活Symbol表单容器1
- 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U8、U9模块

ISE Project Navigator (P.49d) - D:\FPGA_work\2015_ORG\Org-Lab1\Org-Lab1.xise - [Org_Lab1.sch*]

File Edit View Project Source Process Add Tools Window Layout Help

Options

Select Options

When you click on a branch:

- ☒ Select the entire branch
- ☐ Select the line segment

When you move an object:

- ☒ Keep the connections to other objects

Design Options

Symbols

Categories

<D:\FPGA_work\2015_ORG\Org-Lab1>

Symbols

ALU_LCDF

Anti_jitter

Symbol Name Filter

Orientation

Rotate 0

Symbol Info

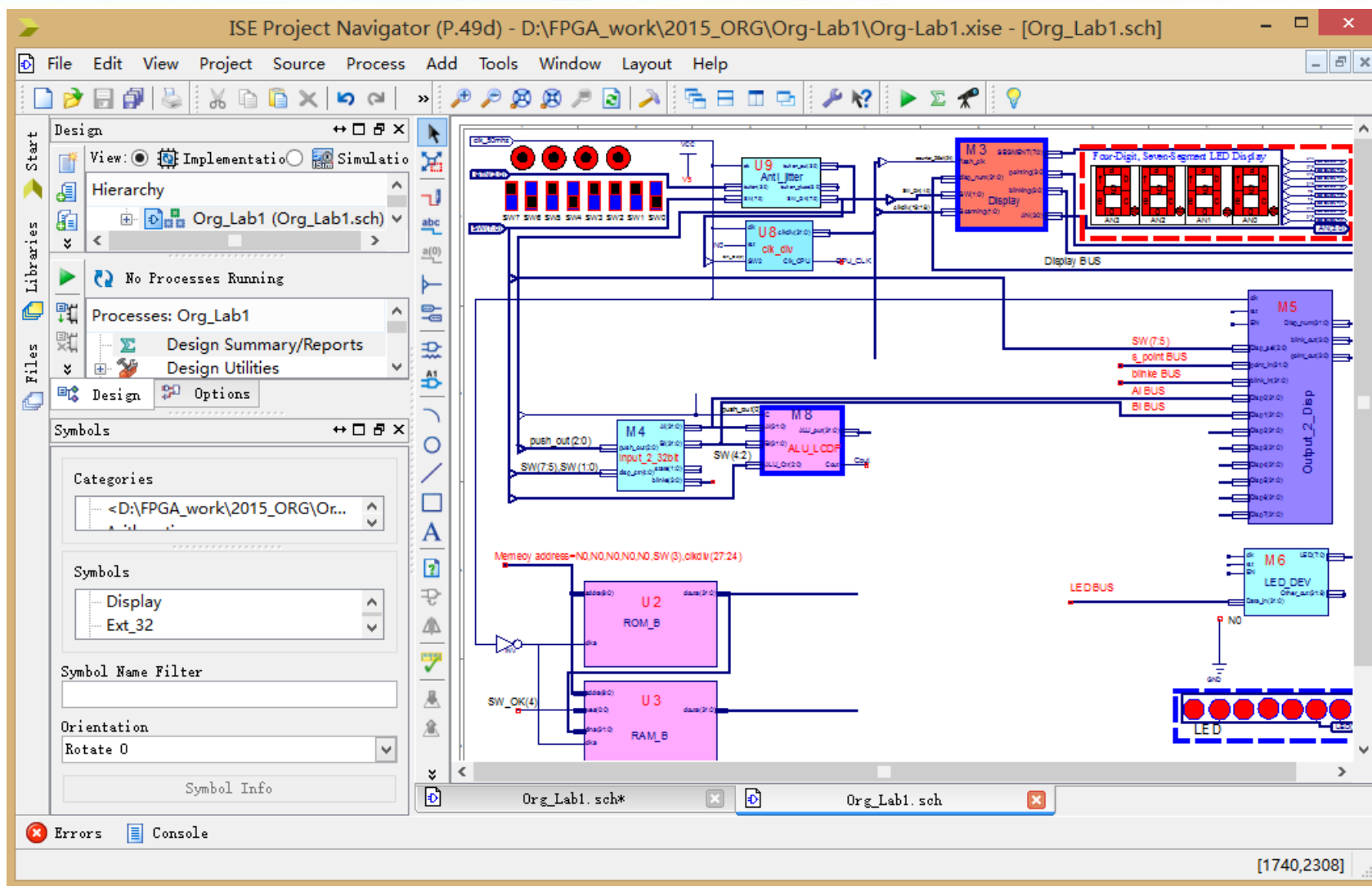
Errors Console

Org_Lab1.sch* Org_Lab1.sch

[1068,1132]



在逻辑Label2中修改输入：连接若干信号

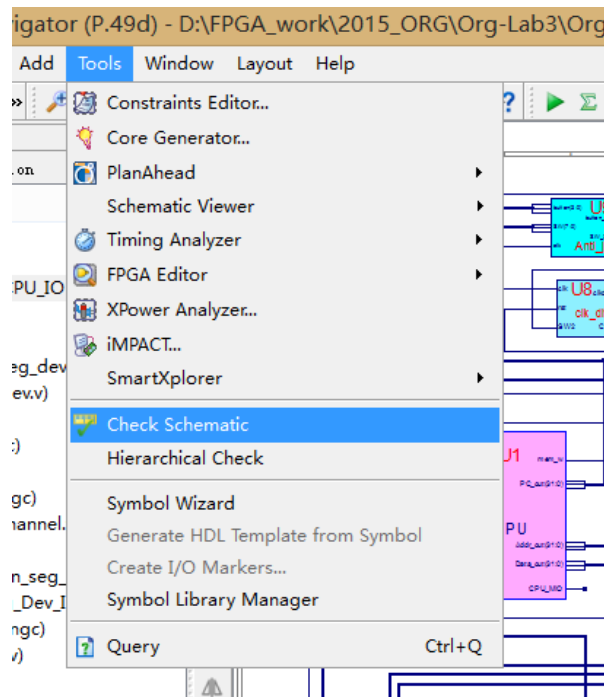




原理图检查

□ 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

- 点击Add Source 添加调用模块
- 顶层窗口放置模块Symbol后会直接调用对应模块
- 综合器会根据端口模块连接信号
 - 被调用模块名必须与图形模块定义一致
 - 被调用.v文件中的端口信号必须与图形模块定义一致



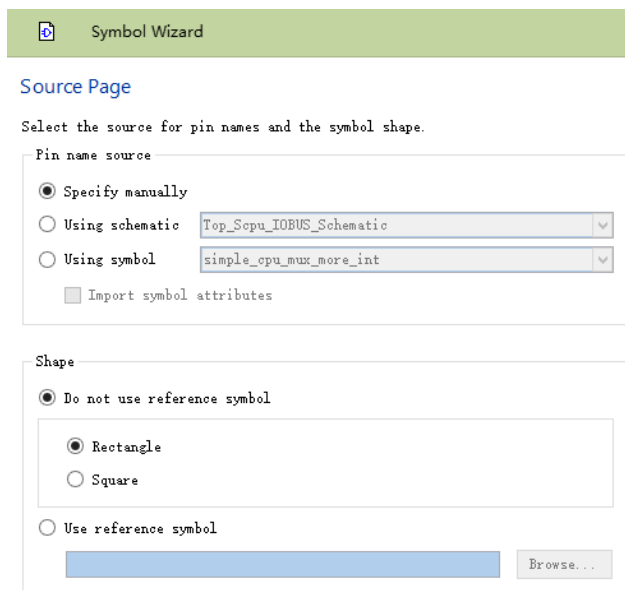
设计编辑模块逻辑符号

-如果需要

设计新逻辑模块图形符号

□ 在原理图输入编辑模式（窗口）

- 在菜单栏：选择Tool→Symbol
 - 弹出Symbol设计向导窗口
 - 根据Wizard指示设计Symbol参数



Symbol Wizard

Source Page

Select the source for pin names and the symbol shape.

Pin name source

☒ Specify manually

☐ Using schematic

☐ Using symbol

☐ Import symbol attributes

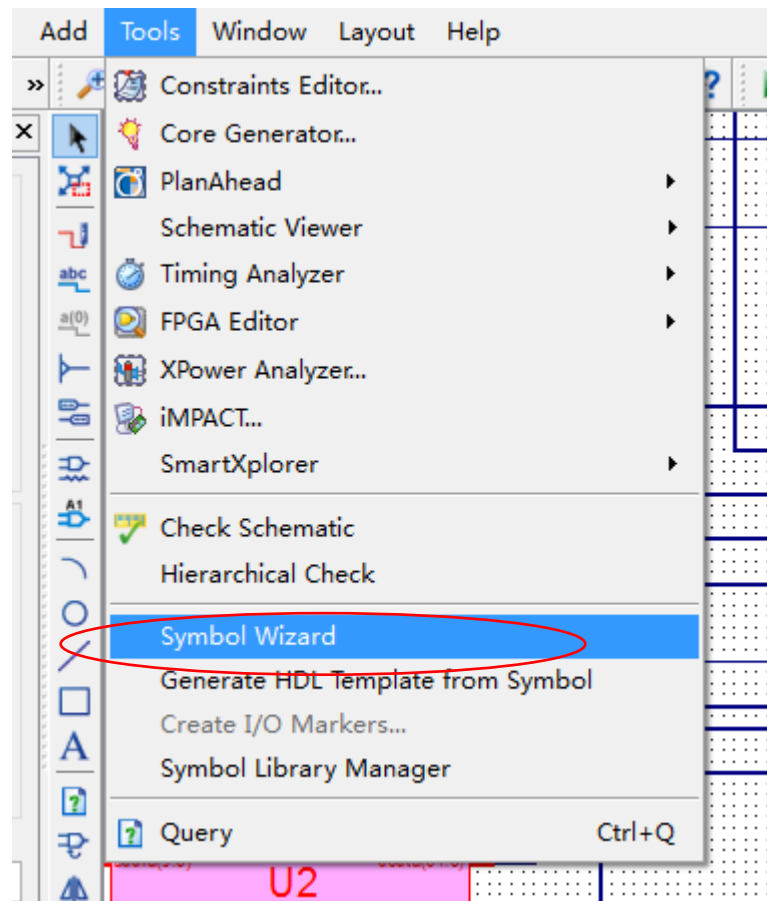
Shape

☒ Do not use reference symbol

☒ Rectangle

☐ Square

☐ Use reference symbol



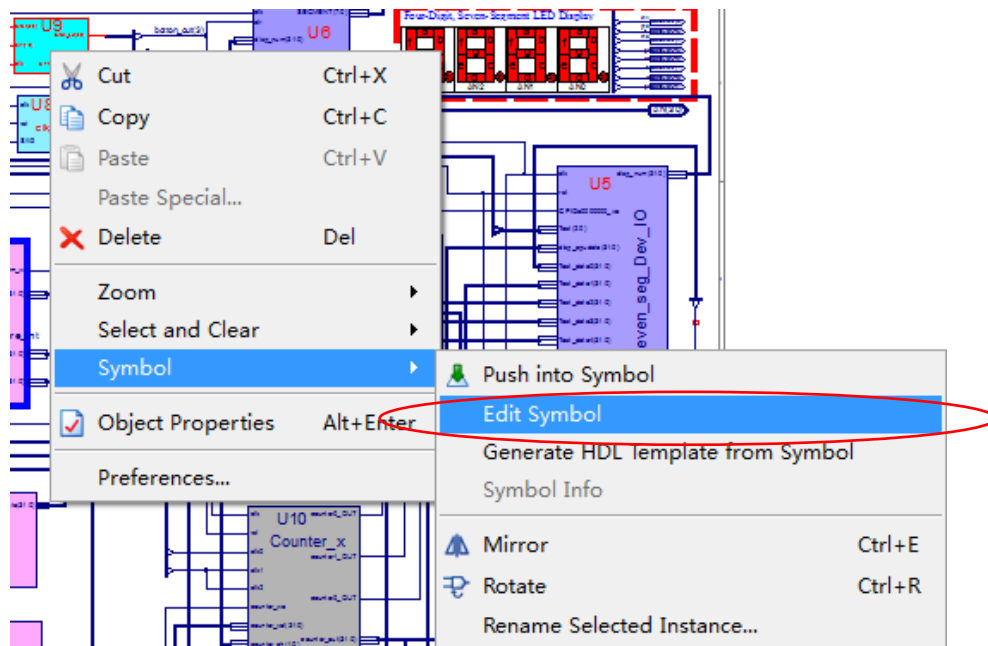
编辑修改逻辑图形符号

□ 进行图形符号编辑窗口

■ 在原理图编辑窗口中

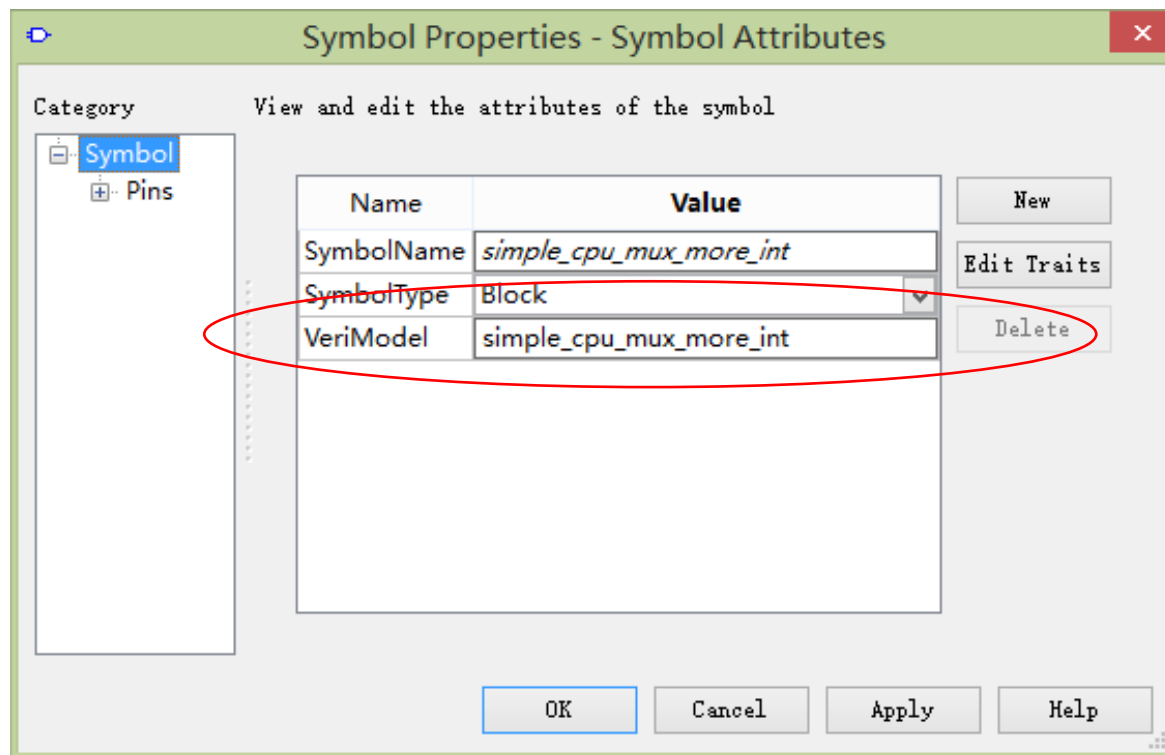
□ 选择要编辑的Symbol模块

- 点鼠标右键，选择Symbol中的edit Symbol
- 弹出Symbol编辑窗口
- 根据需要编辑



给图形符号增加调用属性

- 编辑增加“VeriModel”属性项
 - 属性值必须与将调用的“模块名”相同
 - 此例中为“simple_cpu_mux_more_int”





实现逻辑实验输出模块

-供后继组成实验使用



独立模块设计与仿真

□ 独立模块设计

- 用行为描述设计实现“PPT实验原理”部分介绍的模块

□ 独立模块仿真

- 设计仿真激励(输入)代码
- 分别仿真独立模块

□ 仿真通过后制作必要的逻辑符号

- 设计独立模块的逻辑符号(.sym)

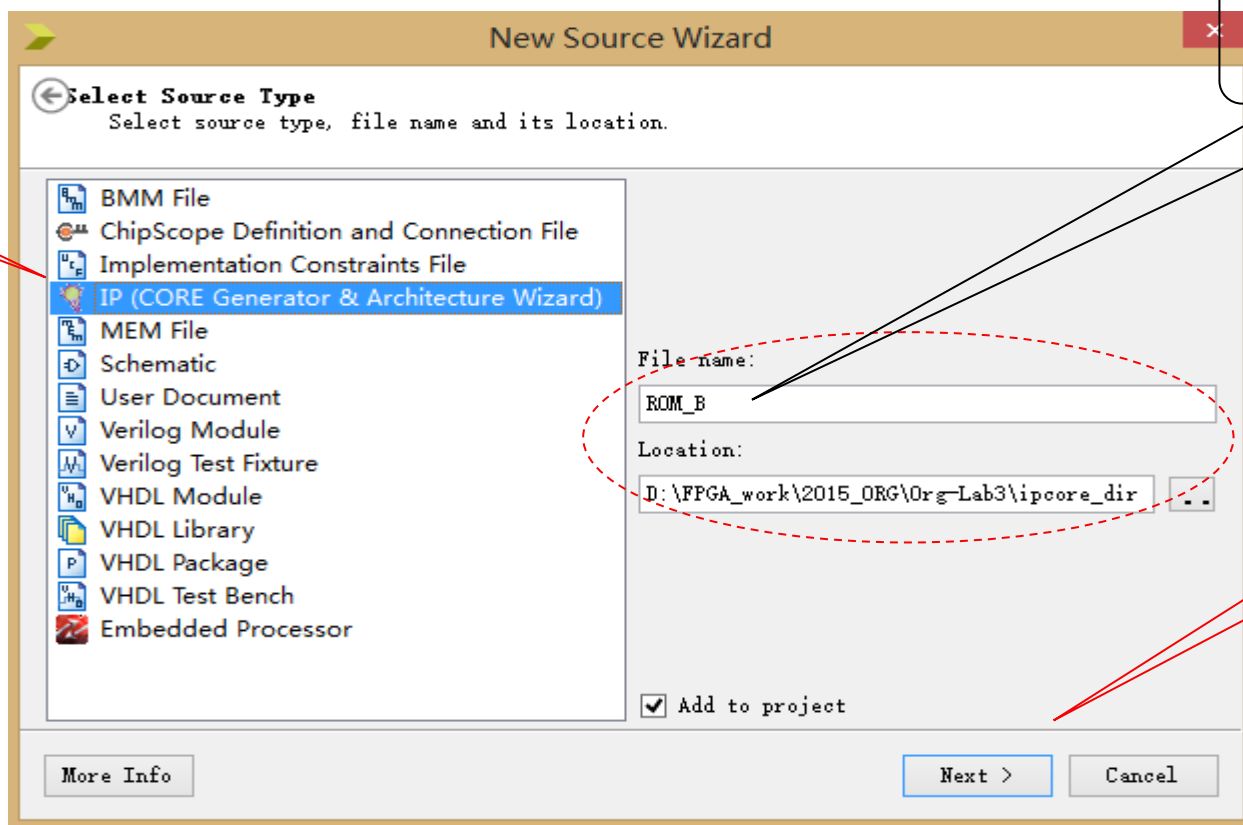


存储器IP核生成

ROM_D IP Core-U2

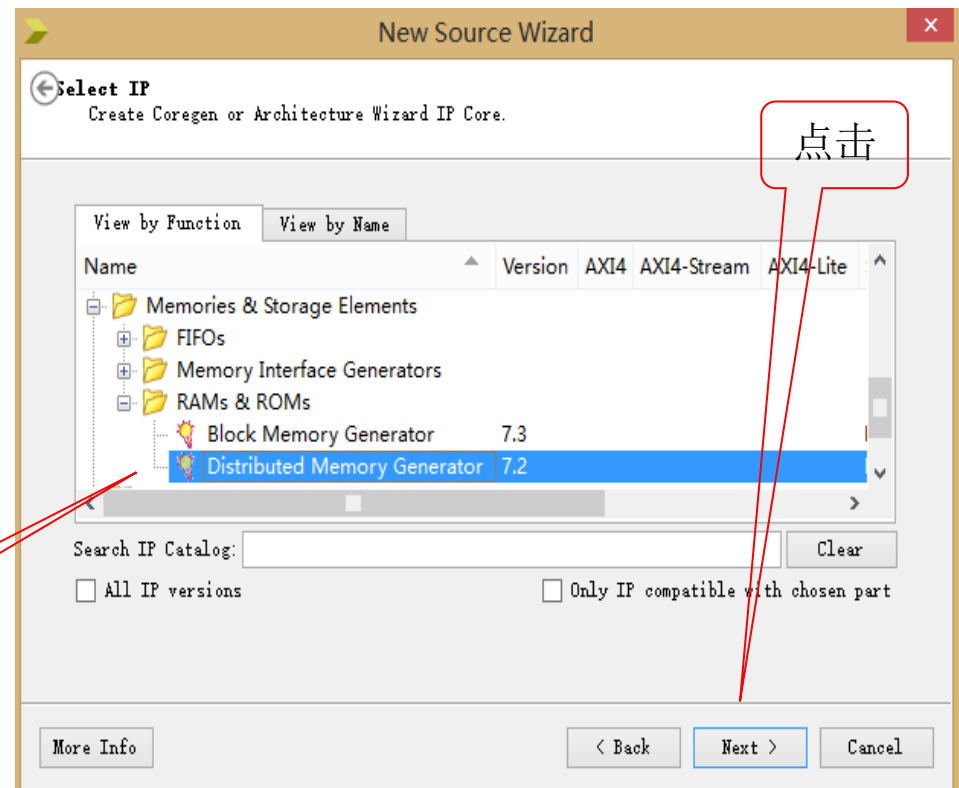
□ 用IP核生成器建立 $1024 \times 32\text{bit}$ 的ROM核

- 【第一步】在ISE集成菜单上从Project选择New Source, 将弹出如如下新模块源模板生成向导窗口。



- 【第二步】 点击Next弹出IP核选择窗口
 - 选择核: Memories & Storage Elements → RAMs & ROMs
 - 选择Distributed Memory Generator 7.2
或Block Memory Generator (注意: 有时钟)

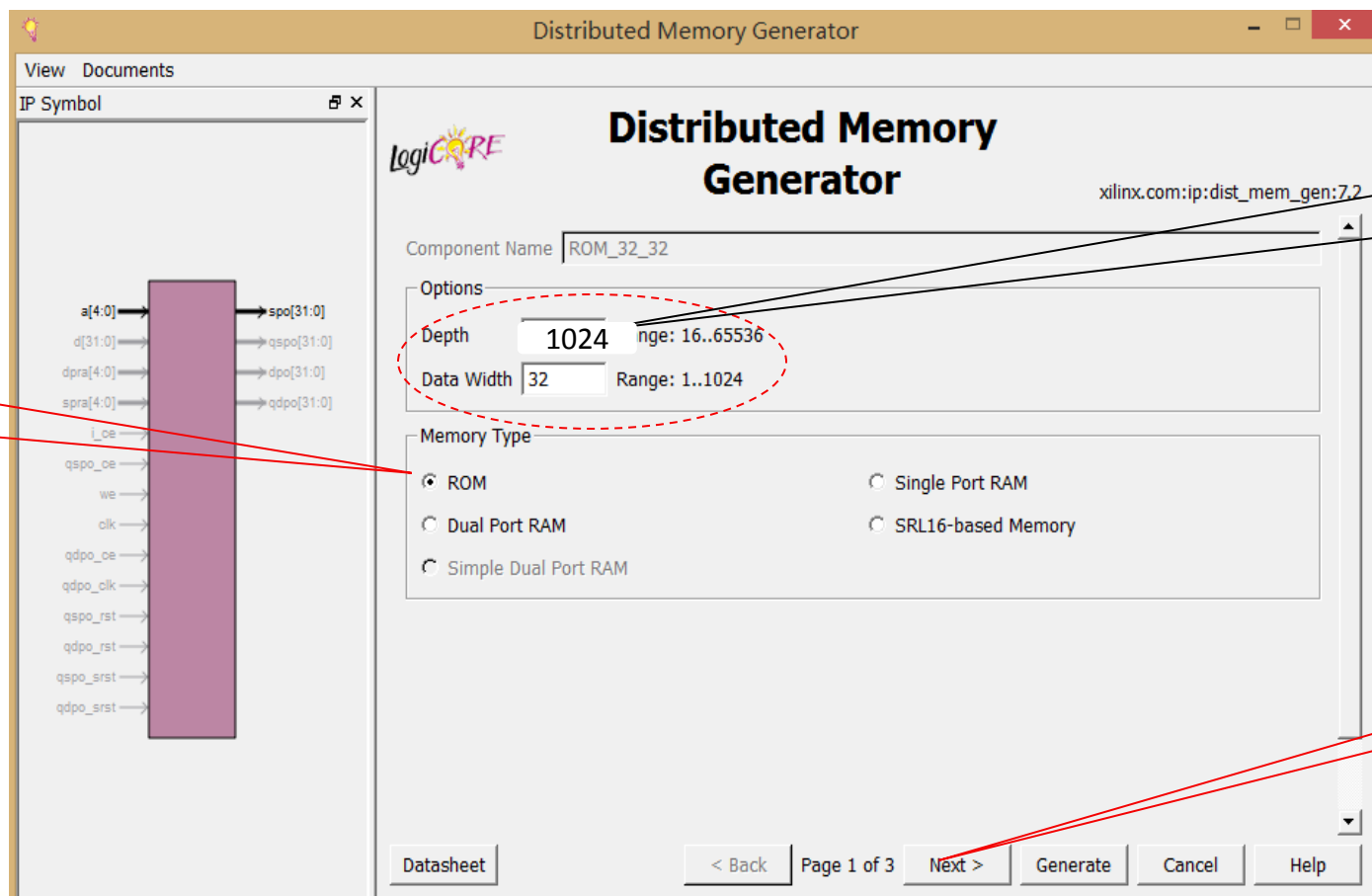
- **Distributed Memory :**
符合异步访问要求
- **Block Memory Generator:**
是同步访问



【第三步】核参数设置

弹出窗口第1页

注意：无时钟



填入:
Depth 1024
Data Width 32

选择:
ROM

点击

④ 【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（ROM.coe）
- ❑ 其余不用修改

点击Browse关联 Init File

Distributed Memory Generator

Distributed Memory Generator

xilinx.com:ip:dist_mem_gen:7.2

Load COE File

If desired the initial memory content can be set by using a COE file. This will be passed to the core as a Memory Initialisation File (MIF).

Coefficients File : I9_men.coe

Browse...

Show...

COE Options

Default Data : 0

Radix : 16

Reset Options

☐ Reset QSP0

☐ Reset QDPO

☐ Synchronous Reset QSP0

☐ Synchronous Reset QDPO

☒ CE Overrides Sync Controls

☐ Sync Controls Overrides CE

点击
查验

查看Datasheet

Datasheet

< Back

Page 3 of 3

Next >

Generate

Cancel

Help

最后
点击
生成

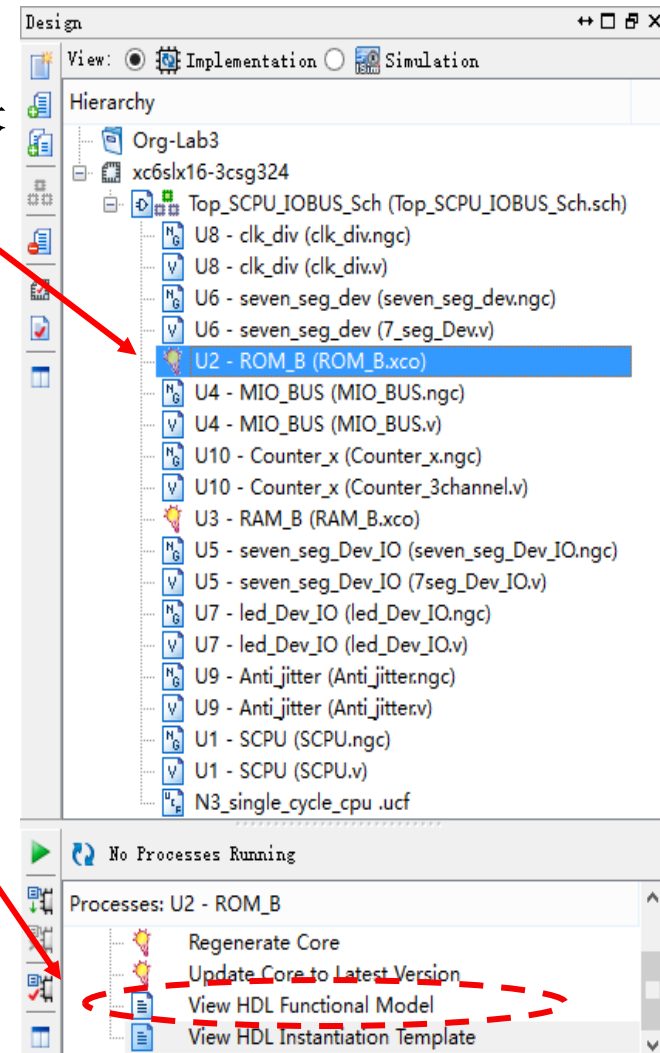
☞ 【第五步】生成后调用

- ▣ 核生成后在设计窗口出现ROM_B.xco模块
- ▣ 点击View HDL Instantiation Template查看核调用结构：

```
ROM_B your_instance_name (  
    .a(a),                // input [9 : 0] a  
    .spo(spo)             // output [31 : 0] spo  
);
```

- ▣ 在当前工程的ipcore_dir目录下有核的图形符号：ROM_B.sym
 - ◆ 这个图形符号非常大，需要修改

☞ 此ROM核在顶层直接调用





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

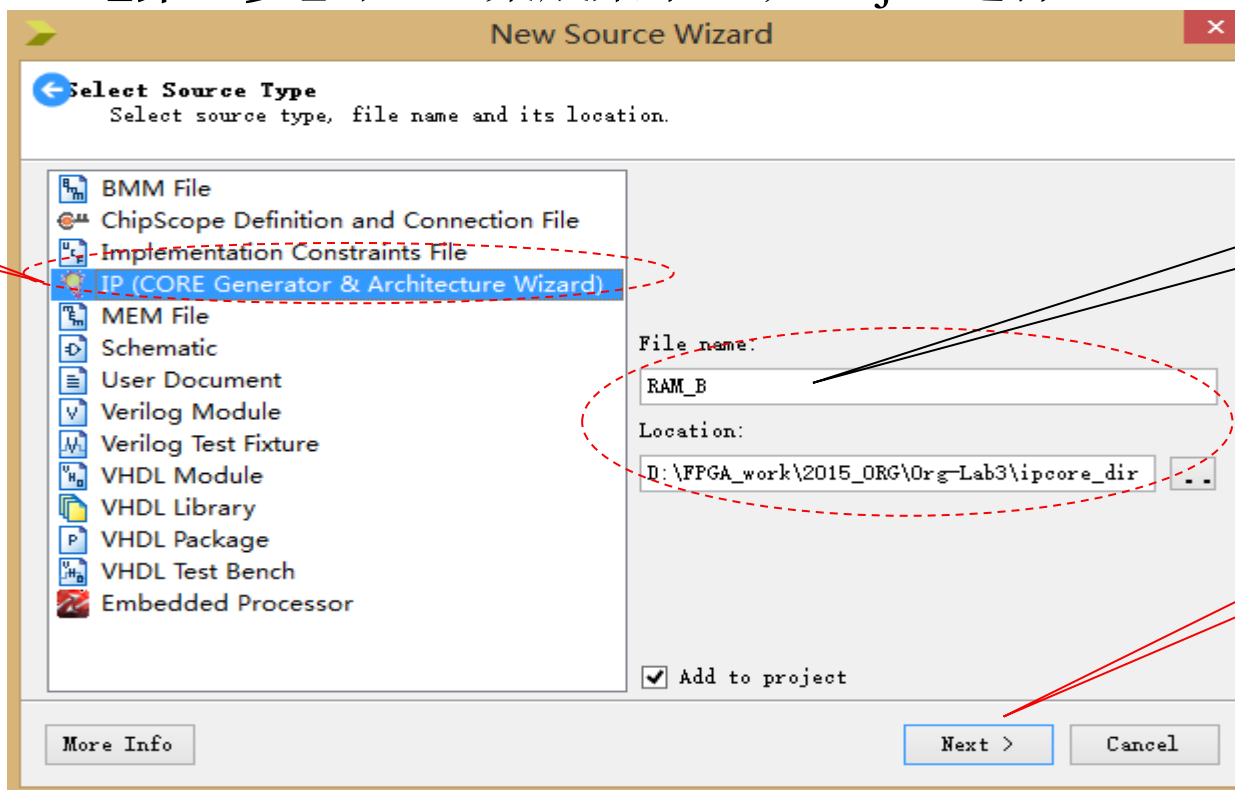
```
memory_initialization_radix=16;
memory_initialization_vector=
00000000, 11111111, 22222222, 33333333, 44444444, 55555555,
66666666, 77777777, 88888888, 99999999, aaaaaaaaaa, bbbbbbbbbb,
cccccccc, dddddddd, eeeeeeee, ffffffff, 557EF7E0, D7BDFBD9,
D7DBFDB9, DFCFFCFB, DFCFBFFF, F7F3DFFF, FFFFDF3D, FFFF9DB9,
FFFFBCFB, DFCFFCFB, DFCFBFFF, D7DB9FFF, D7DBFDB9, D7BDFBD9,
FFFF07E0, 007E0FFF, 03bdf020, 03def820, 08002300;
```

◎ 此数据用于下板测试

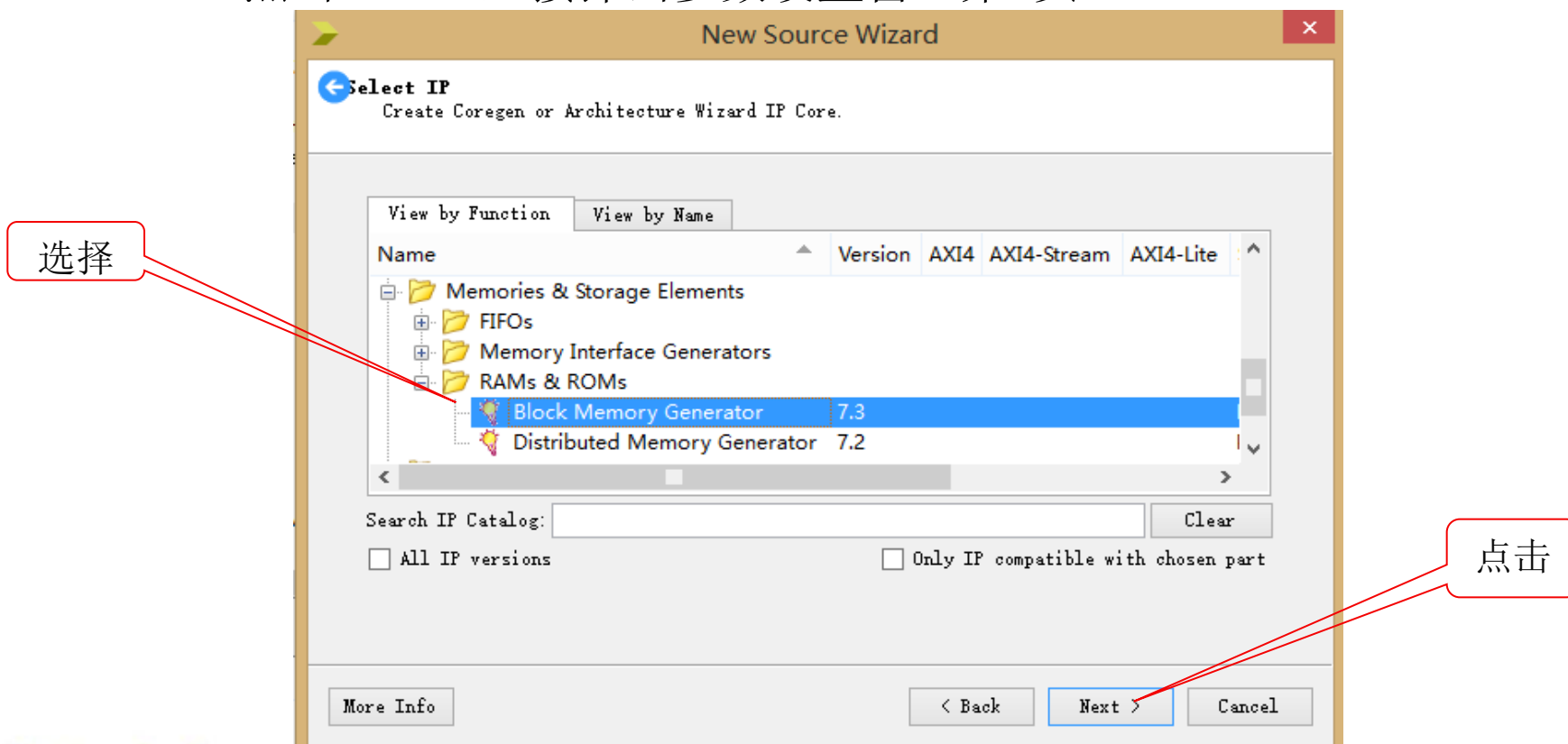
红色数据是LED图形

RAM_B IP Core-U3

- 与ROM相同用ISE核生成器实现RAM
- 用IP核生成器建立 $1024 \times 32\text{bit}$ RAM核
 - 【第一步】在ISE集成菜单上从Project选择New Source，将弹

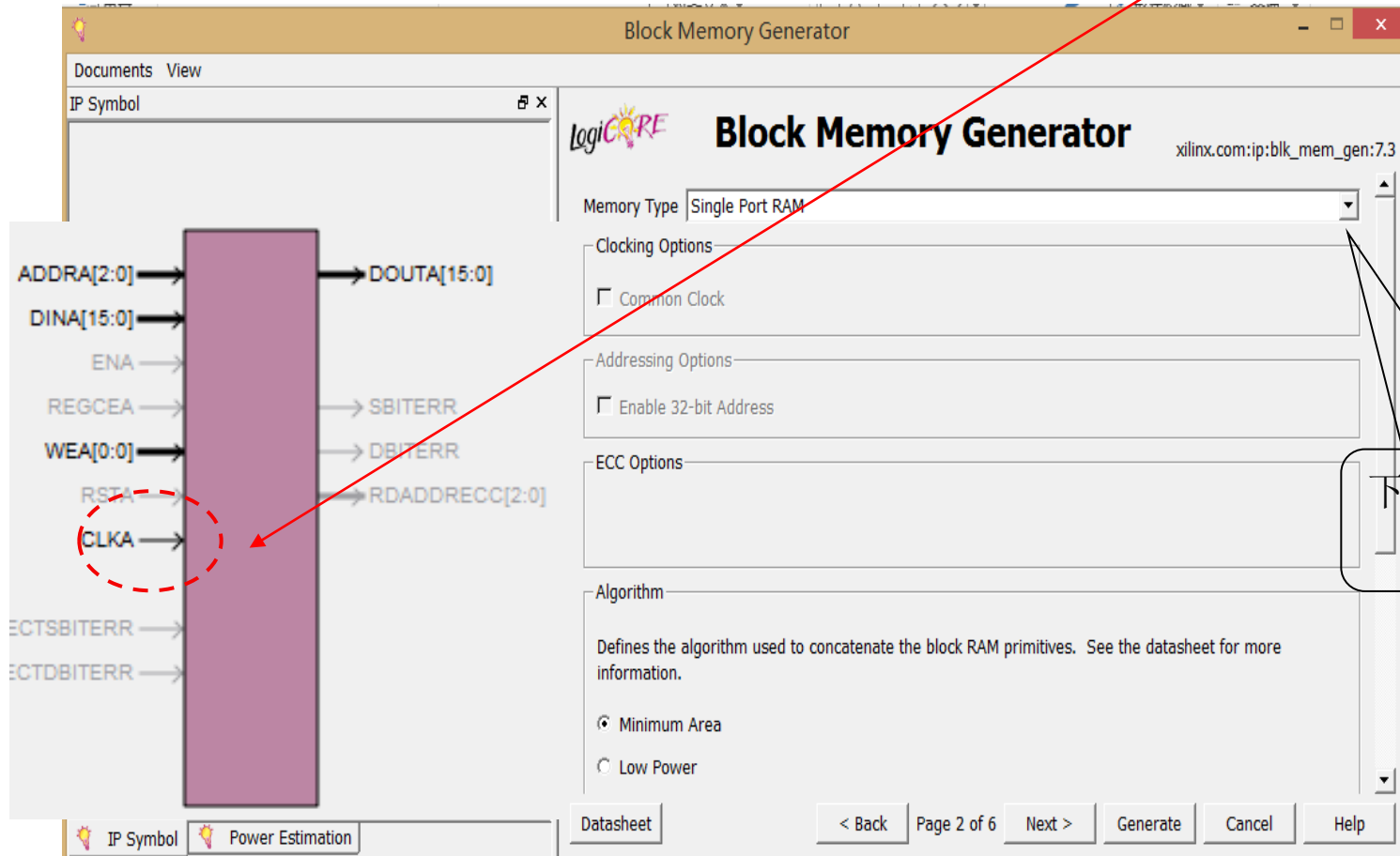


- 【第二步】 点击Next弹出IP核选择窗口
 - 选择核： Memories & Storage Elements → RAMs & ROMs
 - 选择**BLOCK Memory Generator 7.3** 注意：与ROM选择不同
 - 点击Next，直接弹出参数设置窗口第2页

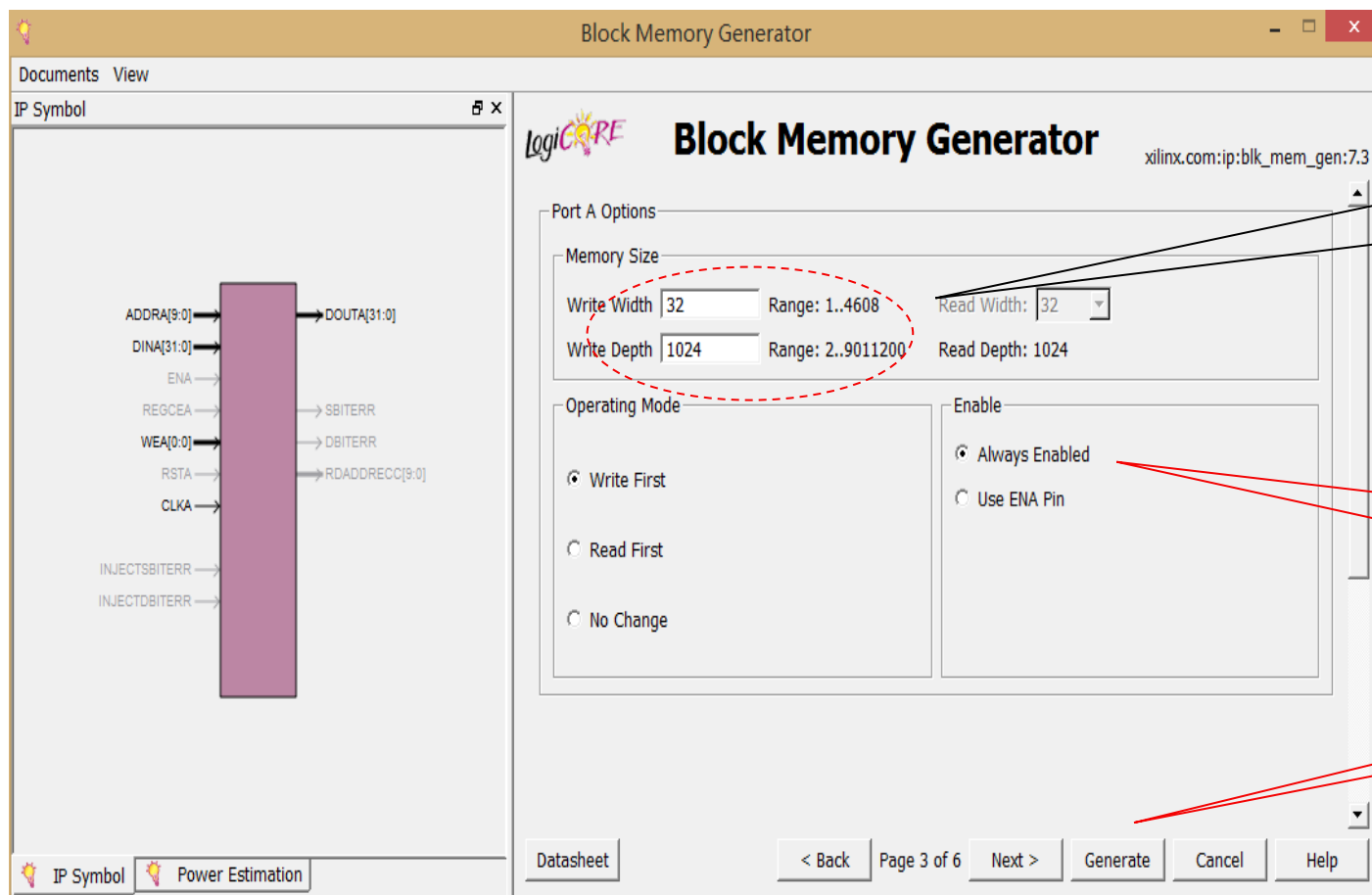


■ 【第三步】核参数设置

- 点击Next，直接弹出窗口第2页 **注意：有时钟**



- 【第三步】核参数设置
 - 点击Next，弹出窗口第3页



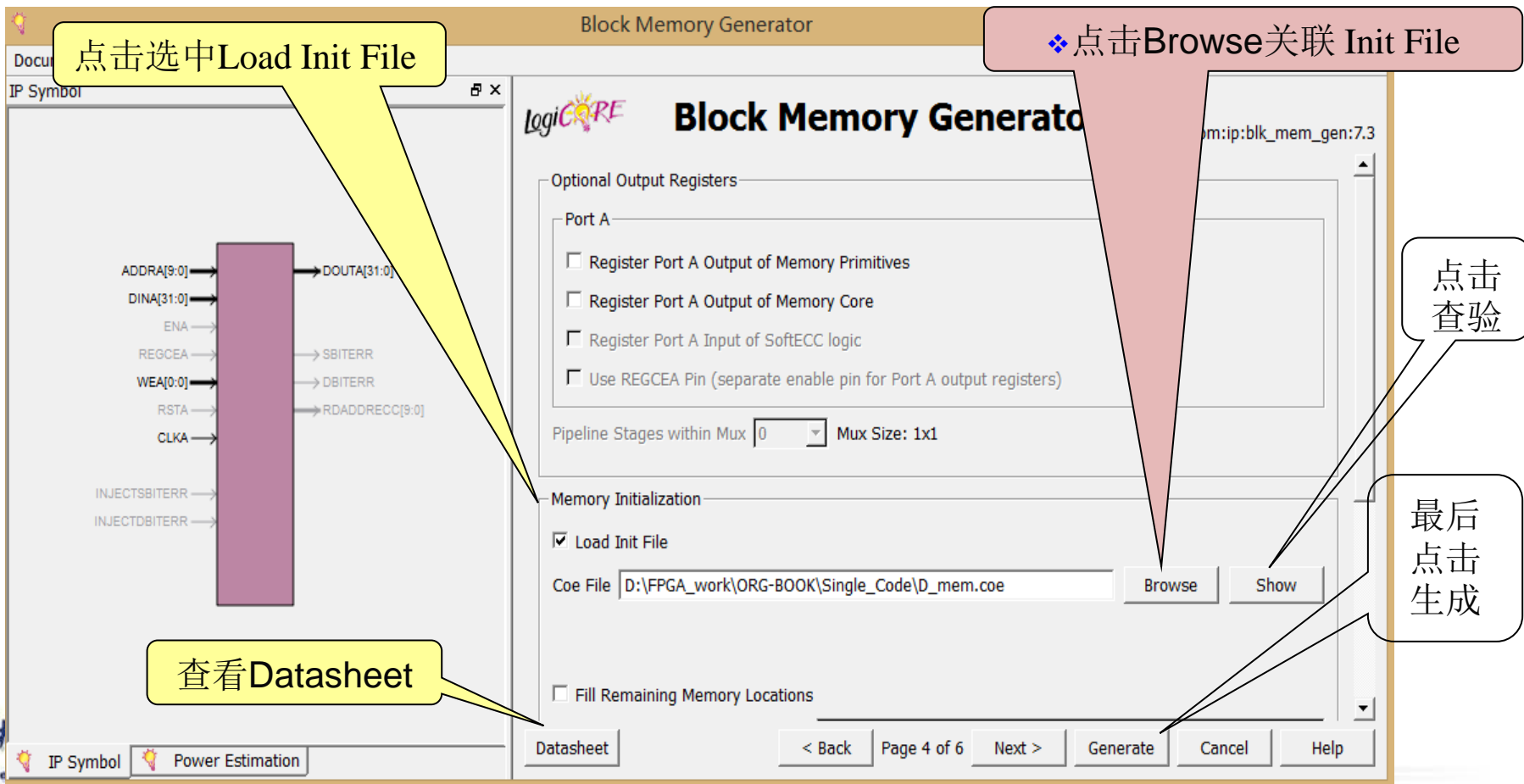
填入:
Write Width 32
Write Depth 1024

选择:
Always Enabled

点击

本实验RAM无初始化数据

- 【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（本实验无）
 - 其余不用修改



The screenshot shows the 'Block Memory Generator' tool interface. The left pane displays the IP Symbol block with various input and output ports. The right pane shows the configuration options for the memory core.

Annotations:

- 点击选中Load Init File**: Points to the 'Load Init File' checkbox in the 'Memory Initialization' section.
- 查看Datasheet**: Points to the 'Datasheet' button at the bottom left.
- ❖点击Browse关联 Init File**: Points to the 'Browse' button next to the 'Coe File' field.
- 点击查验**: Points to the 'Show' button next to the 'Coe File' field.
- 最后点击生成**: Points to the 'Generate' button at the bottom right.

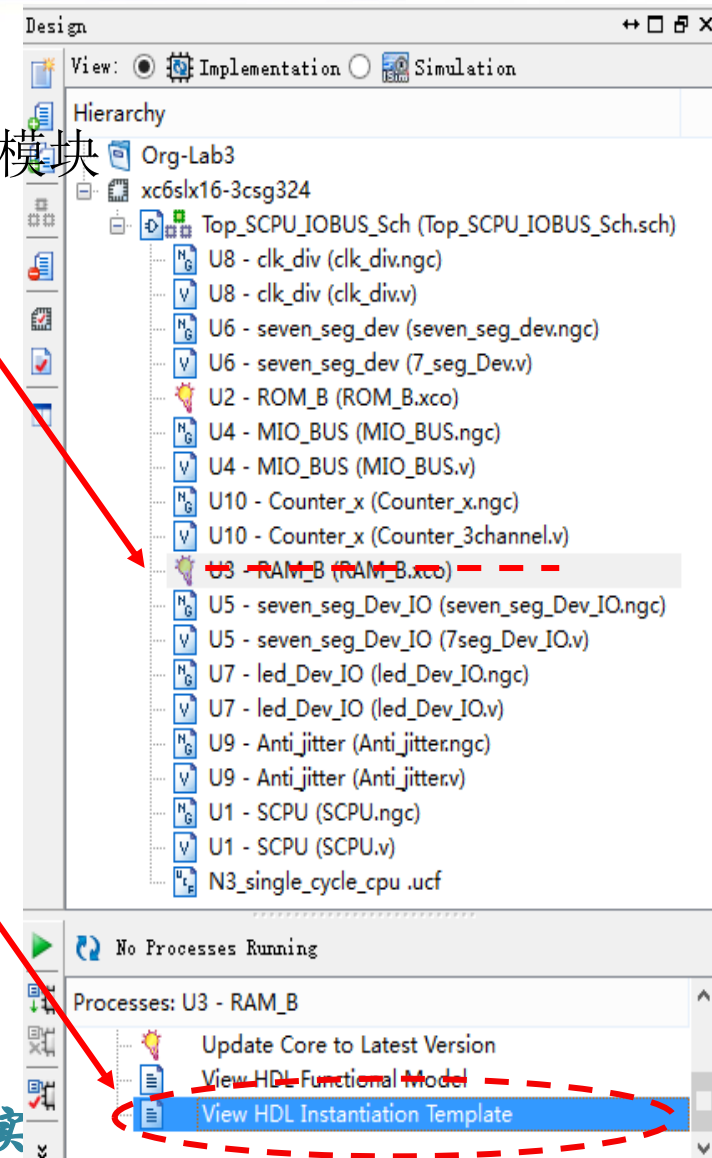
Interface Details:

- Optional Output Registers**: Includes checkboxes for 'Register Port A Output of Memory Primitives', 'Register Port A Output of Memory Core', 'Register Port A Input of SoftECC logic', and 'Use REGCEA Pin (separate enable pin for Port A output registers)'.
- Pipeline Stages within Mux**: Set to 0.
- Mux Size**: Set to 1x1.
- Memory Initialization**: The 'Load Init File' checkbox is checked. The 'Coe File' field contains 'D:\FPGA_work\ORG-BOOK\Single_Code\D_mem.coe'.
- Buttons**: 'Browse', 'Show', 'Generate', 'Cancel', and 'Help' are visible at the bottom.

- 【第五步】生成后调用
 - 核生成后在设计窗口出现RAM_B.xco模块
 - 点击View HDL Instantiation Template查看核调用结构:

```
RAM_B U3 (  
    .clka(clk_m),           // input clka  
    .wea(data_ram_we),     // input [0:0] wea  
    .addra(ram_addr),      // input [9:0] addra  
    .dina(ram_data_in),    // input [31:0] dina  
    .douta(ram_data_out) // output [31:0] douta  
);
```

- 在当前工程的ipcore_dir目录下有核的图形符号: RAM_32_32.sym
 - 这个图形符号非常大, 需要修改
- 此RAM核在顶层模块中调用





● END