

Computer Organization & Design

# Computer Organization & Design

## 实验与课程设计

### 实验三

### CPU辅助模块设计



武汉大学

WUHAN UNIVERSITY

# Course Outline



# 实验目的



1. 扩展优化逻辑实验基本模块
2. 优化计算机系统实现的辅助模块
3. 了解设备与接口、人机交互
4. 了解最简单的接口GPIO
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. Nexys A7开发板
3. Vivado 2018.1 及以上开发工具

## □ 材料

无



# Course Outline



# 实验任务



1. 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
2. 设计存储器IP模块
  - 32位ROM、32位RAM
3. 设计CPU调试测试显示通道模块
4. 优化逻辑实验输出的显示模块
  - 将原理转化为结构化行为描述
  - 增加七段码文本图形显示



# Course Outline





# 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

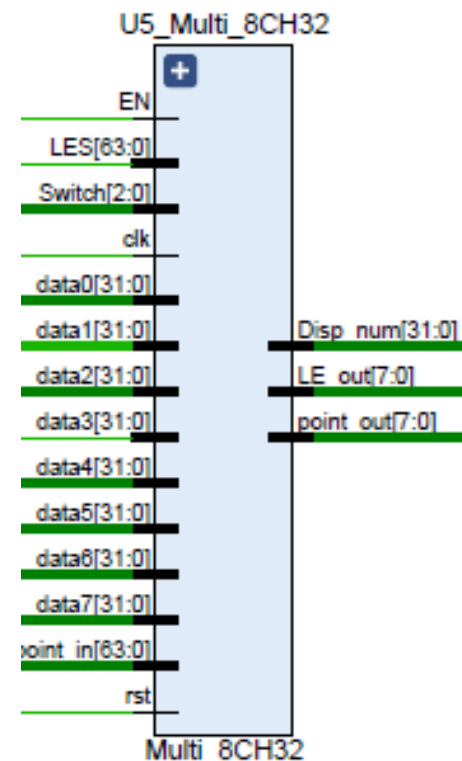


## □ 多路选择器的简单应用

- 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.v



## □ 本实验用IP 软核- **U5**

- 核调用模块Multi\_8CH32.edf
- 核接口信号模块(空文档): Multi\_8CH32.v



# 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?

```
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  
                        );
```

## 通道“0”控制:

```
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  
end
```

# Multi\_8CH32调用信号关系



```
Multi_8CH32    U5( .clk(clk_io), .rst(rst),  
                . EN(EN),  
                . point_in(????????),  
                .LES(????????),  
                .Test(SW_OK[7:5]),  
                .data0(????????????),  
                .data1(????????????)  
                .data2(?????????????),  
                .data3(?????????????),  
                .data4(?????????????),  
                .data5(?????????????),  
                .data6(?????????????),  
                .data7(?????????????),  
  
                .point_out(point_out),  
                .blink_out(LE_out),  
                .disp_num(disp_num)  
                );
```

//仅控制通道0  
//外部输入  
//外部输入  
//来自开关去抖  
//通道0输入  
//通道1输入  
//通道2输入  
//通道3输入  
//通道4输入  
//通道5输入  
//通道6输入  
//通道7输入  
  
//输出到显示模块  
//输出到显示模块  
//输出到显示模块

武汉大学 计算机学院 计算机系统综合设计

# 逻辑实验通用分频模块U8优化: `clk_div.v`

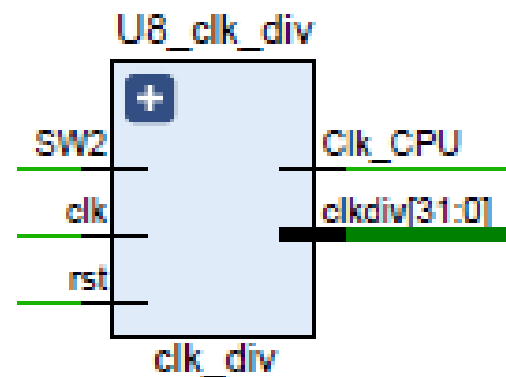


## □ 通用计数分频模块

- 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]`)



`clk_div.v`

## □ 本实验用- **U8**

- 调用模块`clk_div.v`



# 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
```



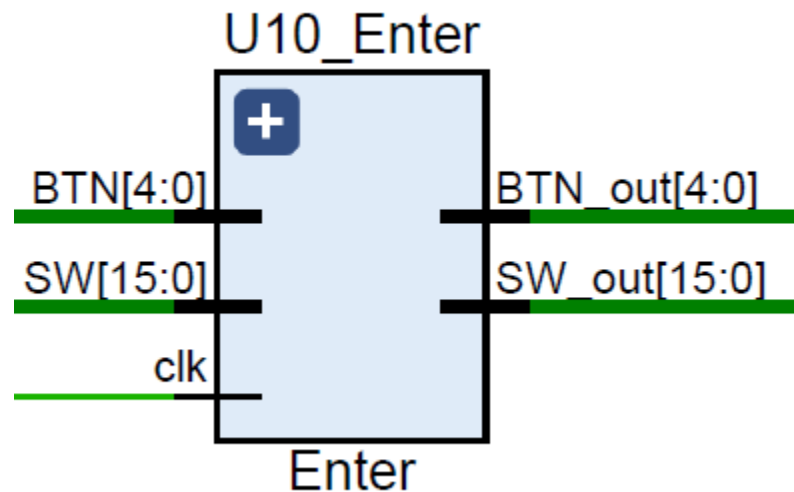


# 双数据输入模块端口U9: Enter



## □ 双数据输入端口信号

```
Enter(input clk,  
      input[4:0] BTN, // 按键  
      input[15:0] SW, // 开关  
      output[4:0] BTN_out,  
      output[15:0] SW_out // 开关输出  
);
```





# LED并行显示模块U7: SPIO

## □ 15位LED指示灯控制(IP Core)

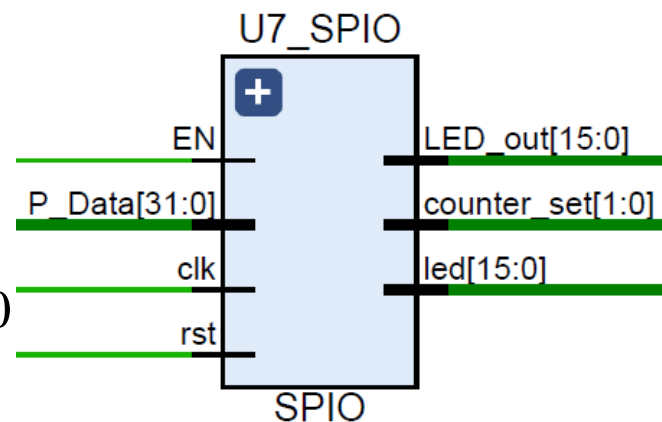
- 逻辑实验的输出LED显示模块
  - 相当于通用输入输出接口: GPIO
  - 15位用于LED指示控制, 其余用于扩展
- 器件编号为**U7**

## □ 基本功能

- 输入32位二进制数据: P\_Data
  - clk=时钟, EN: 输出使能, rst=复位
- 并行输出: LED\_out、counter\_set、GPIOf0

## □ 本实验用IP 软核- **U7**

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



# LED并行显示模块IP核端口信号



## □ PIO/LED-GPIO IP核端口信号

```
SPIO(input clk,          //时钟
      input rst,          //复位
      input EN,           //PIO/LED显示刷新使能
      input [31:0] P_Data, //并行输入
      output reg[1:0] counter_set, //用于计数/定时模块控制
      output reg[15:0] LED_out,    //并行输出数据
      output reg[15:0] led,        // LED输出
      output reg[13:0] GPIOf0 //待用： GPIO
    );
```





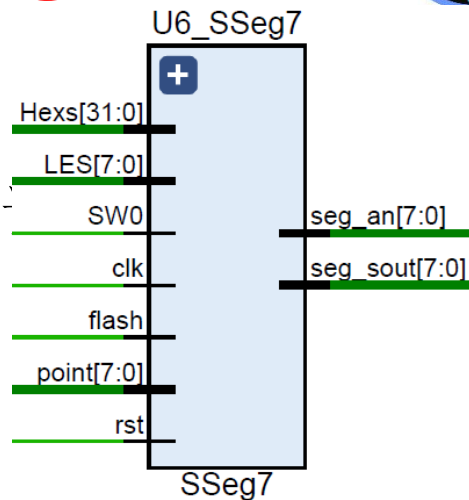
# 七段码显示器IP核U6: SSeg7

## □ 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
- 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七段码闪烁频率, 由通用分频器U8(Div[25])提供, point: 七段小数点, LES: 七段码使能, 闪烁指示
- 串行输出: seg\_an=使能, seg\_sout=串行七段显示数据,



## □ 本实验用IP 软核- **U6**

- 核调用模块SSeg7.edf
- 核接口信号模块(空文档): SSeg7.v





# 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

```
S Seg7(input clk,           //时钟
        input rst,         //复位
        input SW0,         //文本(16进制)/图形(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 [7:0]seg_an, //
        output reg [7:0]seg_sout //七段码显示数据
    );
```

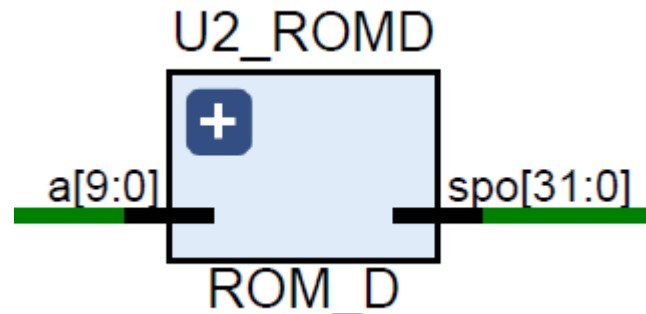


# 只读存储器IP核:



## □ 只读存储器

- 用于CPU应用的代码存储器
- 模块名改为ROM\_D
- 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
  - Distributed Memory Generator
- 核模块符号文档: ROM\_D.xci
  - 自动生成符号不规则, 需要修整
- ROM初始化文档暂时不变

## □ 用Vivado工具生成固核

- 用IP Catalog向导生成
- 核调用模块ROM\_D.xci





# 随机存储器IP核: RAM

## □ 随机存储器

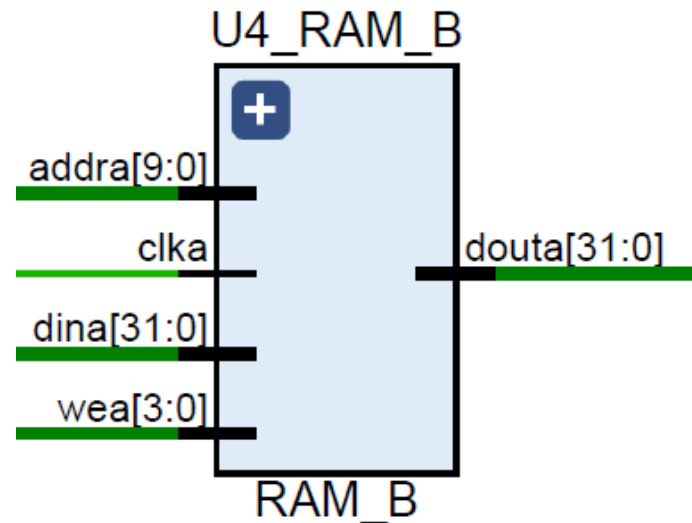
- 用于CPU应用的数据或代码存储器
- 模块名改为RAM\_B
- 器件编号改为U4

## □ 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.xci
  - 自动生成符号不规则, 需要修整
- RAM初始化文档无

## □ 用Vivado工具生成固核

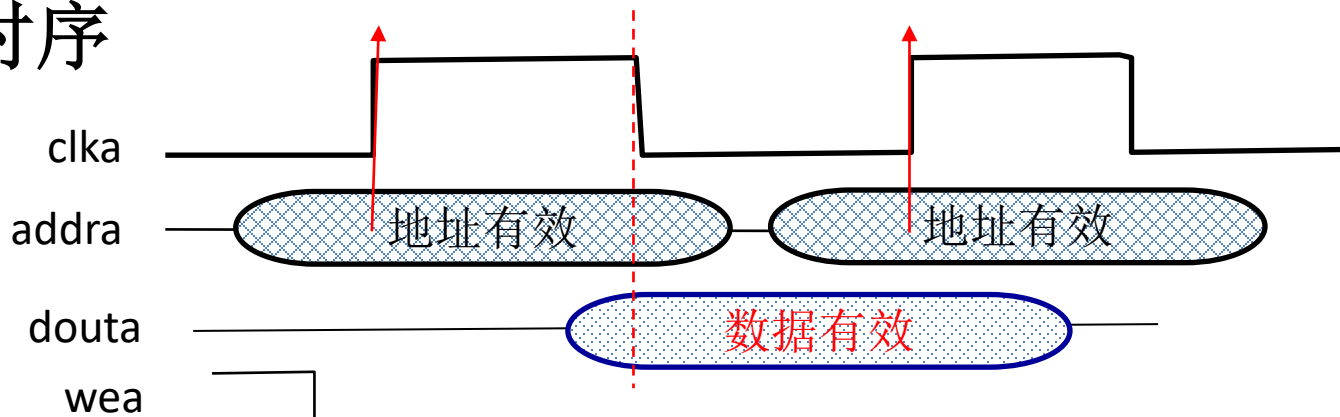
- 用IP Catalog向导生成
- 核调用模块ROM\_D.xci



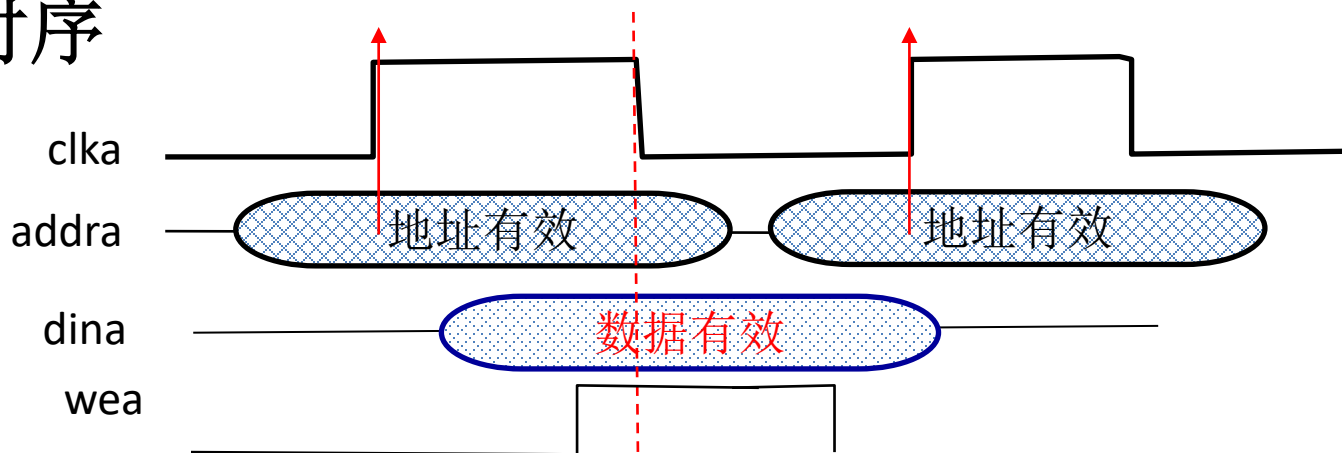
# Block Memory 时序



## □ 读时序



## □ 写时序







# RAM的字节写

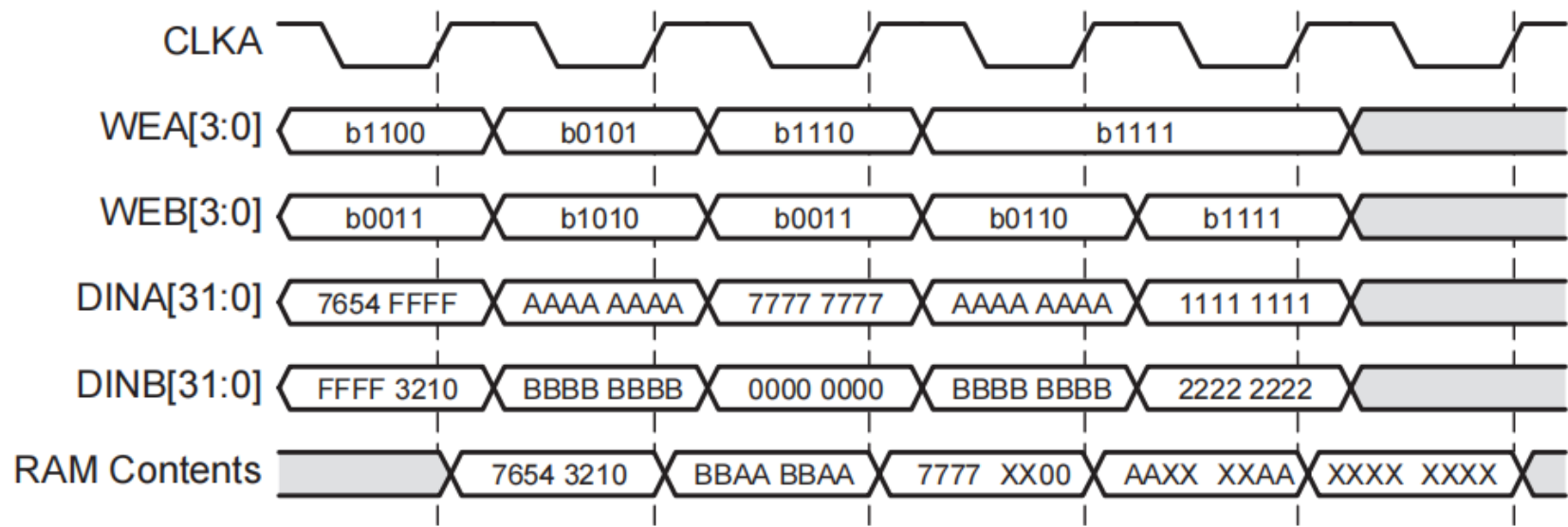


Figure 3-15: Write-Write Collision Example



# Course Outline



# ROM IP核生成



Flow Navigator

PROJECT MANAGER - SCPU\_SOC

Sources

Design Sources (4)

- Verilog (1)
  - ctrl\_encode\_def.v
- IP2SOC\_Top (SCPU\_Top.v) (7)
  - U\_CLKDIV : clk\_div (clk\_div.v)
  - U\_SCPU : sccpu (sccpu.v) (9)
    - U\_IM : xil\_defaultlib.imem
    - U\_DM : dm (dm.v)
    - U\_MIO : MIO\_BUS (MIO\_BUS.v)
    - U\_Multi : Multi\_CH32 (Multi\_CH32.v)
    - U\_7SEG : seg7x16 (seg7x16.v)
  - mux16 (mux.v)
  - mux8 (mux.v)
- Constraints
- Simulation Sources (4)
  - sim\_1 (4)

Project Summary | IP Catalog

Cores | Interfaces

distributed

Name

- Vivado Repository
  - Basic Elements
    - Memory Elements
      - Distributed Memory Generator
  - Memories & Storage Elements
    - RAMs & ROMs
      - Distributed Memory Generator



# 配置ROM IP核 - memory config

Customize IP

**Distributed Memory Generator (8.0)**

Documentation IP Location Switch to Defaults

☐ Show disabled ports

Component Name ROM\_D

**memory config** Port config RST & Initialization

**Options**

Depth 1024 [16 - 65536]

Data Width 32 [1 - 1024]

**Memory Type**

**Memory Type**

☒ ROM

☐ Single Port RAM

☐ Simple Dual Port RAM

☐ Dual Port RAM

OK Cancel

a[9:0] spo[31:0]

# 配置ROM IP核 - RST & Initialization



Customize IP

## Distributed Memory Generator (8.0)

Documentation IP Location Switch to Defaults



☐ Show disabled ports

Component Name: ROM\_D



memory config Port config **RST & Initialization**

**Load COE File**

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:   

**COE Options**

Default Data:   Radix:  

**Reset Options**

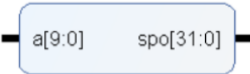
☐ Reset QSPO ☐ Reset QDPO

☐ Synchronous Reset QSPO ☐ Synchronous Reset QDPO

**ce overrides**

☒ CE Overrides Sync Controls ☐ Sync Controls Overrides CE

OK Cancel



The diagram shows a rectangular component labeled 'ROM\_D'. It has two input ports on the left side: 'a[9:0]' and 'spo[31:0]'. The component is connected to a larger block on the right, which is the configuration window.

# COE文件



- COE (Coefficient) 文件是ROM的初始化需要使用来传递参数
- 可通过编辑venus中venus出Hex文件编辑而成  
加上两行  
memory\_initialization\_radix=16;  
memory\_initialization\_vector=  
其余各行加,  
最后一行加;

```
C:\Users\Q. Liu\Desktop\source\mipstestloop_fpga.coe - Notepad++
文件(F) 编辑(E) 搜索(S) 视图(V) 编码(N) 语言(L) 设置(T) 工具(O)
mipstestloop_fpga.coe
1 memory_initialization_radix=16;
2 memory_initialization_vector=
3 20020005,
4 2003000c,
5 2067fff7,
6 00e22025,
7 00642824,
8 00a42820,
9 10a7000a,
10 0064202a,
11 10800001,
12 20050000,
13 00e2202a,
14 00853820,
15 00e23822,
16 ac670044,
17 8c020050,
18 08000011,
19 20020001,
20 ac020054,
21 200affff,
22 014a5020,
23 014a5020,
24 014a5020,
25 014a5020,
26 014a5020,
27 014a5020,
28 014a5020,
29 014a5020,
30 014a5020,
31 014a5020,
32 014a5020,
33 014a5020,
34 014a5020,
35 014a5020,
36 014a5020,
37 014a5020,
38 214c0004,
39 214e000c,
40 8d8d0000,
41 adcd0000,
42 08000025;
43
```

# 产生RAM IP核



lab02 - [D:/projects/lab02/lab02.xpr] - Vivado 2018.1

File Edit Flow Tools Repgrts Window Layout View Help Q- Quick Access Ready

Flow Navigator PROJECT MANAGER - lab02

PROJECT MANAGER

- Settings
- Add Sources
- Language Templates
- IP Catalog

IP INTEGRATOR

- Create Block Design
- Open Block Design
- Generate Block Design

SIMULATION

- Run Simulation

RTL ANALYSIS

- Open Elaborated Design

SYNTHESIS

- Run Synthesis
- Open Synthesized Design

IMPLEMENTATION

- Run Implementation
- Open Implemented Design

Sources

Design Sources (13)

- top (top.v) (11)
  - U1\_SCPU : xil\_defaultlib.CPU\_0
  - U2\_ROMD : ROM\_D (ROM\_D.xci)

Hierarchy IP Sources Libraries Compile Order

Repository Properties

User Repository

Path : C:/Xilinx/Vivado/2018.1/data/

Number of IPs : 498

General IPs Interfaces

Project Summary top.v IP Catalog

Cores Interfaces

Q- block

Name	AXI4	Status	License	VLNV
Complex Multiplier	AXI4-Stream	Production	Included	xilinx.com:ip:cmpy...
CORDIC	AXI4-Stream	Production	Included	xilinx.com:ip:cordi...

Memories & Storage Elements

RAMs & ROMs & BRAM

Block Memory Generator

AXI4 Production Included xilinx.com:ip:blk\_m

Details

Path: C:/Xilinx/Vivado/2018.1/data/

Number of IPs: 498

Number of interfaces: 201

Tcl Console Messages Log Reports Design Runs

Q- + %

Name	Constraints	Status	WNS	TNS	WHS	THS	TPWS	Total Power	Failed Routes	LUT	FF	BRAMs	URAM	DSP	Start	Elapsed	Run Strategy
synth_1 (active)	constrs_1	Not started															Vivado Synthesis
impl_1	constrs_1	Not started															Vivado Implement
Out-of-Context Module Runs																	
ROM_D_synth_1	ROM_D	synth_design Complete!								105	0	0.00	0	0	6/2...	00:00:53	Vivado Synthesis

# 产生RAM IP核

Customize IP



## Block Memory Generator (8.4)

[Documentation](#) [IP Location](#) [Switch to Defaults](#)

IP Symbol

Power Estimation

☐ Show disabled ports

BRAM\_PORTA

- ▶ addra[9:0]
- ▶ clka
- ▶ dina[15:0]
- ◀ douta[15:0]
- ▶ ena
- ▶ wea[1:0]

Component Name RAM\_B

Basic

Port A Options

Other Options

Summary

Interface Type Native

☐ Generate address interface with 32 bits

Memory Type Single Port RAM

☐ Common Clock

### ECC Options

ECC Type No ECC

☐ Error Injection Pins Single Bit Error Injection

### Write Enable

☒ Byte Write Enable

Byte Size (bits) 8

### Algorithm Options

Defines the algorithm used to concatenate the block RAM primitives.  
Refer datasheet for more information.

Algorithm Minimum Area

Primitive 8kx2

OK

Cancel



# 产生RAM IP核

Customize IP



## Block Memory Generator (8.4)



[Documentation](#) [IP Location](#) [Switch to Defaults](#)

IP Symbol

Power Estimation

☐ Show disabled ports

BRAM\_PORTA

▶ addr[9:0]

▶ clka

▶ dina[31:0]

◀ douta[31:0]

▶ wea[3:0]

Component Name RAM\_B

Basic

Port A Options

Other Options

Summary

### Memory Size

Write Width 32 Range: 8 to 4096 (bits)

Read Width 32

Write Depth 1024 Range: 2 to 1048576

Read Depth 1024

Operating Mode Write First

Enable Port Type Always Enabled

### Port A Optional Output Registers

☒ Primitives Output Register ☐ Core Output Register

☐ SoftECC Input Register ☐ REGCEA Pin

### Port A Output Reset Options

☐ RSTA Pin (set/reset pin) Output Reset Value (Hex) 0

☐ Reset Memory Latch Reset Priority CE (Latch or Register Enable)

### READ Address Change A

☐ Read Address Change A

OK

Cancel

# 产生RAM IP核



Customize IP

## Block Memory Generator (8.4)

Documentation IP Location Switch to Defaults

IP Symbol Power Estimation

☐ Show disabled ports

BRAM\_PORTA

- ▶ addra[9:0]
- ▶ clka
- ▶ dina[31:0]
- ◀ douta[31:0]
- ▶ wea[3:0]

Component Name RAM\_B

Basic Port A Options Other Options Summary

Pipeline Stages within Mux 0 Mux Size: 1x1

**Memory Initialization**

☒ Load Init File

Coe File D:/projects/lab02/lab02.srsrcs/D\_mem.coe Browse Edit

屏幕截图 Ctrl + Alt + A

屏幕录制 Ctrl + Alt + S

屏幕识图 Ctrl + Alt + O

屏幕翻译 Ctrl + Alt + F

✓ 截图时隐藏当前窗口

Remaining Memory Locations

Memory Locations (Hex) 0

**Simulation Model Options**

Define the type of warnings and outputs are generated when a read-write or write-write collision occurs.

Collision Warnings All


**Behavioral Simulation Model Options**


☐ Disable Collision Warnings ☐ Disable Out of Range Warnings

OK Cancel




# 产生RAM IP核





 Generate Output Products ×


The following output products will be generated. 


**Preview**


  

▼  RAM\_B.xci (OOC per IP)

 Instantiation Template

 Synthesized Checkpoint (.dcp)

 Structural Simulation

 Change Log


**Synthesis Options**

☐ Global

☒ Out of context per IP

**Run Settings**

Number of jobs: 7 ▼



Apply

**Generate**

Skip



◎ END

