



**中国科学技术大学**  
University of Science and Technology of China

# 计算机组成原理

## Lab2 CPU功能部件设计

计算机实验教学中心

2024/3/25

# 实验目标

---

- Verilog语法复习
- 掌握寄存器堆 (Register File) 的设计方法
- 掌握算术逻辑单元 (ALU) 的设计方法及简单应用
- 掌握存储器IP核例化及初始化方法

# 实验原理

## 1. Verilog语法复习

### □ Verilog描述注意事项

#### ✓ 组合电路

- 使用assign 或者 always @\* 描述, “=” 赋值;
- 必须配对使用if...else, case语句赋值完全, 避免出现锁存器;
- 避免出现反馈! 例如,  $y = y + x$ ;
- 无需复位, 即组合电路中, 变量无复位信号.

#### ✓ 时序电路

- 使用always @(posedge clk, posedge rst) 描述, “<=” 赋值;
- 边沿敏感变量表中避免出现除时钟和复位外的其他信号;
- 时钟信号避免出现在语句块内.

#### ✓ 多驱动问题

- 模块中所有的assign和always块都是并行执行的;
- 不要在多个并行执行体中对同一变量赋值.

# 实验原理

## 1. Verilog语法复习

### □ 变量类型问题

- ✓ 使用always语句描述的变量，务必声明为reg类型（声明为reg类型的变量，综合后不一定生成寄存器）；
- ✓ 使用assign语句赋值的变量，应声明为wire类型。

示例：

```
wire [7:0] a,b,y;
```

```
reg [7:0] r1, r2;
```

```
assign y = a;  
always @(*)           // (en, a, b)  
    if (en) r1 = a;  
    else r1 = b;
```

```
always @(posedge clk)  
    if (en) r2 <= a;
```

组合电路：

1. 敏感变量不要遗漏，建议用\*
2. 所有条件分支均有赋值
3. 不能含有反馈，如 $r1=r1+1$
4. “=” 赋值

时序电路：

1. “<=” 赋值
2. 边沿敏感变量避免出现除时钟和复位外的其他信号
3. 条件分支可以不完备

# 实验原理

---

## 1. Verilog语法复习

### □ 参数化模块：模块间传递参数

✓ 模块格式定义如下：

```
module module_name
```

```
  #(parameter parameter_list)
```

```
  (端口声明) ;
```

```
  变量声明;
```

```
  逻辑功能描述;
```

```
endmodule
```

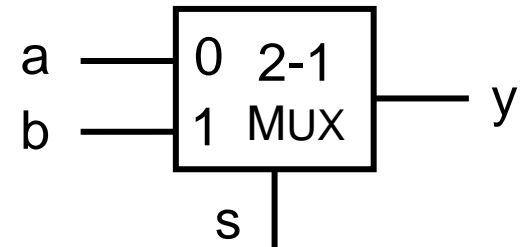
# 实验原理

## 1. Verilog语法复习

### □ 参数化模块：模块间传递参数

#### ✓ 示例1：MUX2

```
module mux2                                //模块名： mux2
    #(parameter MSB = 31,                 //参数声明： 数据最高有效位
      LSB = 0                             //数据最低有效位
    )
    (output [MSB : LSB] y,                 //端口声明： 输出数据
     input [MSB : LSB] a, b,              //两路输入数据
     input s                               //数据选择控制
    );
    assign y = s ? b : a;                  //逻辑功能描述
endmodule
```



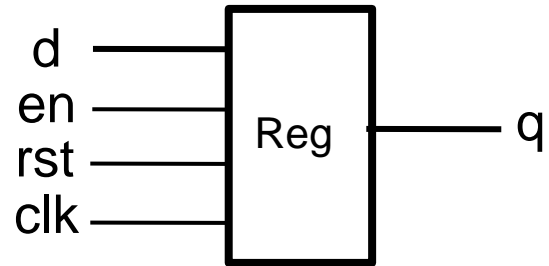
# 实验原理

## 1. Verilog语法复习

### □ 参数化模块：模块间传递参数

#### ✓ 示例2：寄存器

```
module register
    #(parameter WIDTH = 32, RST_VALUE = 0)
    (input  clk, rst, en,
     input  [WIDTH-1 : 0] d,
     output reg [WIDTH-1 : 0] q);
    always @(posedge clk, posedge rst)
        if (rst) q <= RST_VALUE;
        else if (en)
            q <= d;
endmodule
```



- d, q: 输入、输出数据
- clk, rst, en: 时钟、复位、使能

寄存器功能表

rst	clk	en	q	功能
1	x	x	0	复位
0	↑	1	d	置数
0	↑	0	q	保持

# 实验原理

## 1. Verilog语法复习

### □ 带有参数的模块实例化

#### ✓ 模块实例化语句格式：

`module_name` #(parameter\_map) `instance_name` (port\_map);

#### ✓ 端口映射方式：**基于位置或者基于名字，不可混合使用**

- 位置映射：按模块中端口定义的顺序传递

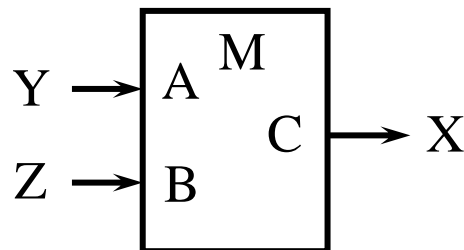
例如：模块定义为 `module M(A, B, C);`

`M M1(Y, Z, X);` //顺序很重要

- 名字映射：`.PortName (value)`

`M M1(.A(Y), .B(Z), .C(X));`

#### ✓ 参数的映射方式类似





# 实验原理

## 1. Verilog语法复习

### □ 带有参数的模块实例化

#### ✓ 示例: MUX4\_1

```
module mux4_1
  (output [7:0] y,
   input [7:0] a, b, c, d,
   input [1:0] S );
```

```
  wire [7:0] k, t;
```

//位置映射

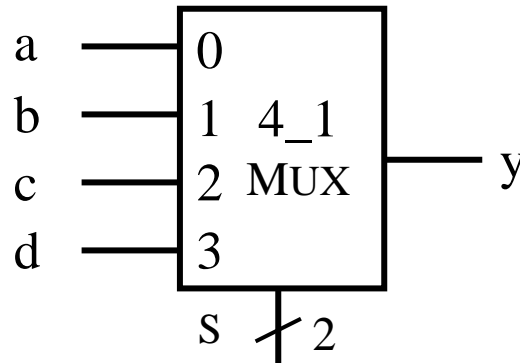
```
  mux2 #(7, 0) M0 (k, a, b, S[0]);
```

```
  mux2 #(7) M1 (t, c, d, S[0]);
```

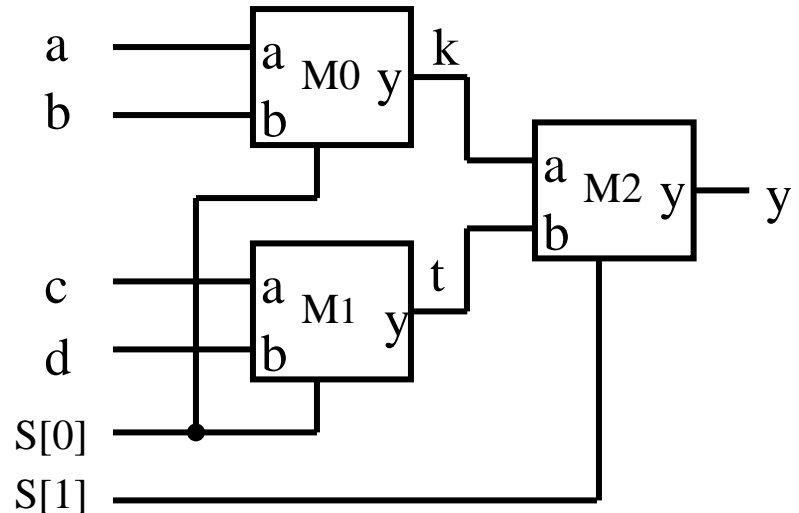
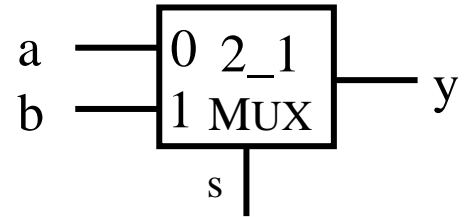
//名字映射

```
  mux2 #(.MSB(7)) M2 (.s(S[1]), .a(k), .b(t), .y(y));
```

```
endmodule
```



```
module mux2 #(parameter MSB=32,LSB=0)
  (output [MSB : LSB] y,
   input [MSB : LSB] a, b,
   input s);
```



# 实验原理

## 1. Verilog语法复习

### □ 仿真文件编写

```
`timescale 1ns/1ps
```

例：时钟生成

```
reg clk;
```

```
// 时钟周期和个数
```

```
parameter CYCLE = 10, Number = 20;
```

```
//clk_gen method1:
```

```
initial begin
```

```
clk = 0;
```

```
repeat (2* Number) //该repeat语句也可生成其他信号
```

```
# CYCLE/2 clk = ~ clk; end
```

```
//clk_gen method2:
```

```
initial begin
```

```
clk = 0;
```

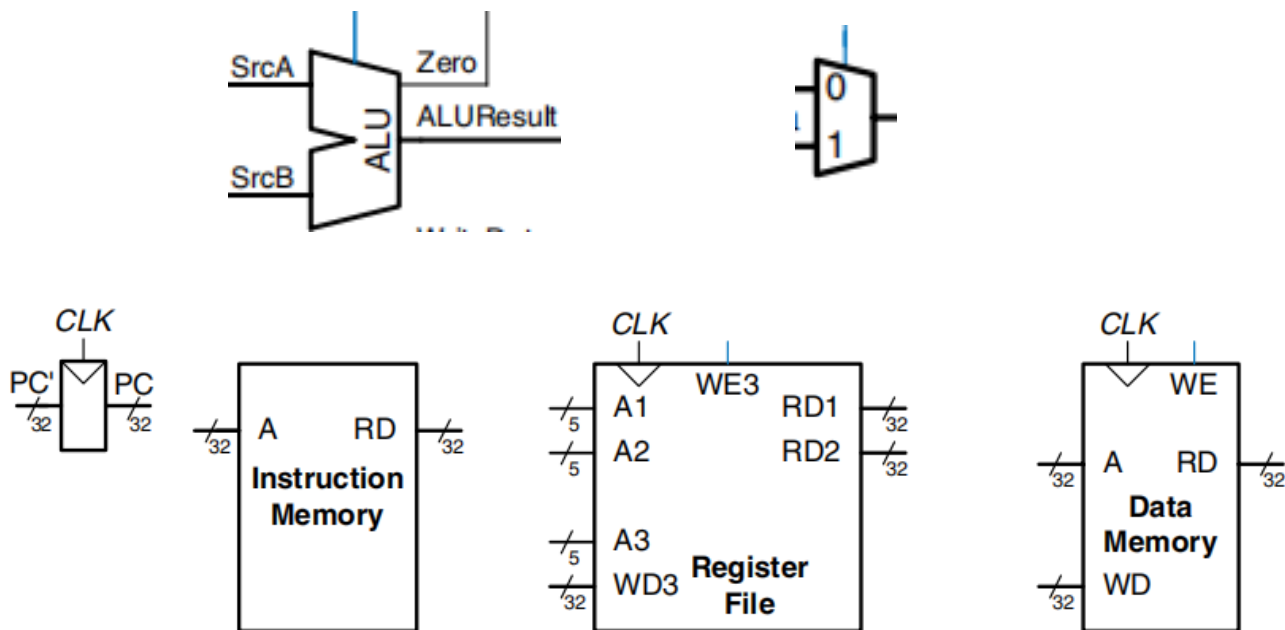
```
forever #CYCLE/2 clk = ~ clk; end
```

**initial和#仅用于仿真，不会产生实际硬件电路**

# 实验原理

## 2.CPU功能部件

- 处理数据的组合单元：与或门，ALU，复用器；
- 存储状态的单元：程序计数器、寄存器堆、**指令存储器**和数据存储器。



# 实验原理

## 2.1 寄存器堆

### □ 寄存器堆介绍

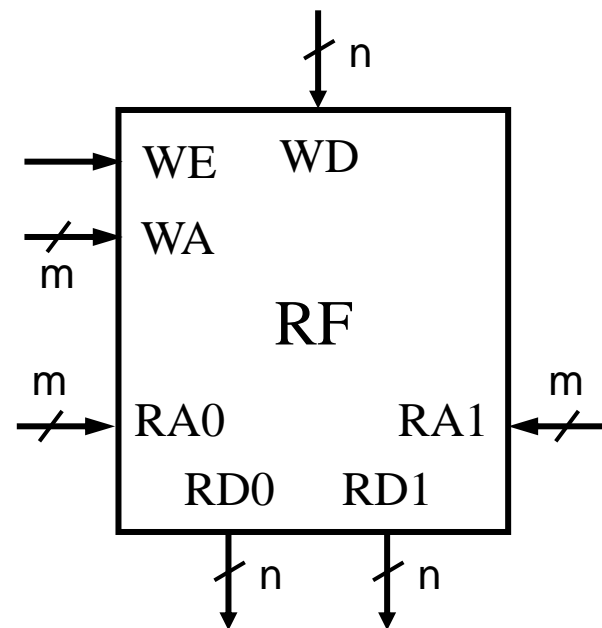
✓ 处理器的32个通用寄存器位于一个叫做寄存器堆 (register file) 的结构中。

✓ 1个写端口

- WA: 写地址
- WD: 写入数据
- WE: 写使能

✓ 2个读端口

- RA0、RA1: 读地址
- RD0、RD1: 读出数据



三端口的 $2^m \times n$ 位寄存器堆外形图

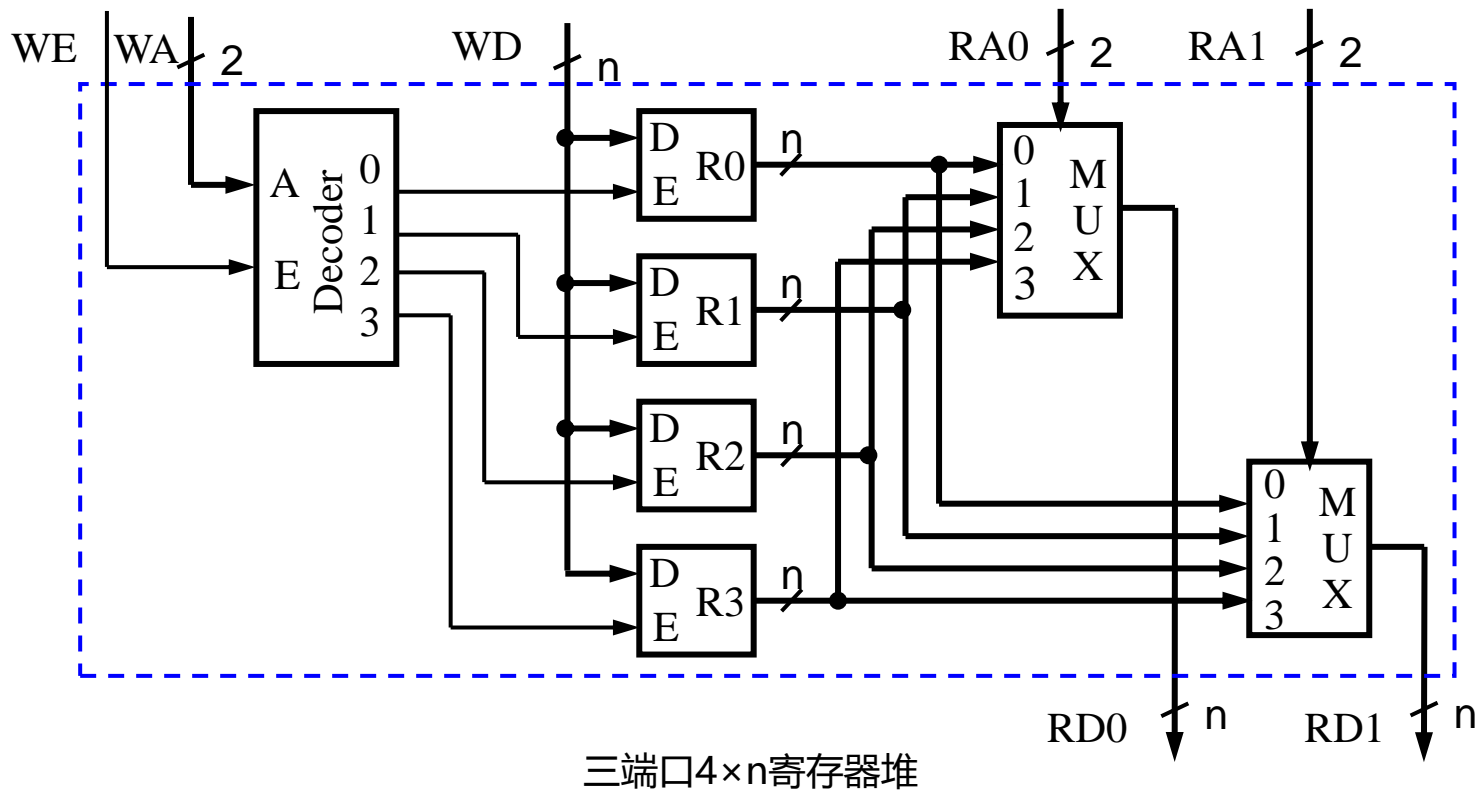
注意:

1. 0号寄存器恒为0，读操作返回0，写操作忽略；
2. 寄存器堆没有复位信号；
3. 读操作：纯组合逻辑；
4. 写操作：时序逻辑。

# 实验原理

## 2.1 寄存器堆

### □ 寄存器堆结构

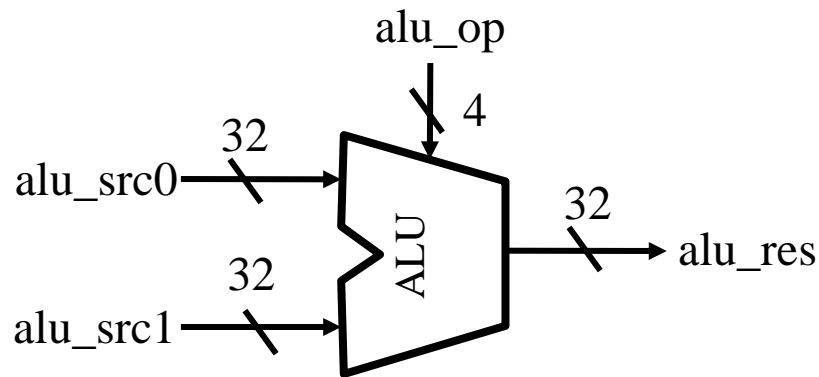


# 实验原理

## 2.2 算术逻辑单元 (ALU)

### □ ALU端口定义

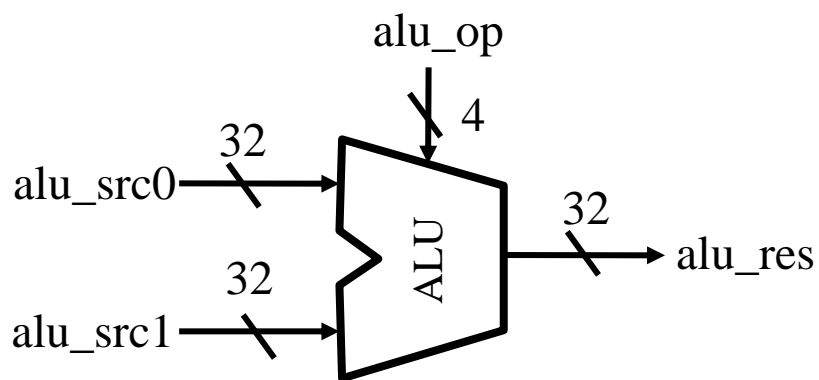
```
module ALU #(parameter WIDTH = 32)    //数据宽度
(
    input [WIDTH-1:0] alu_src0, alu_src1, //两操作数（减运算，alu_src0是被减数）
    input [3:0] alu_op,                    //操作功能（加、减、与、或、异或等）
    output [WIDTH-1:0] alu_res,            //运算结果（和、差 ...）
);
```



# 实验原理

## 2.2 算术逻辑单元 (ALU)

### □ ALU模块功能



ALU模块功能表

Instruction	alu_op	alu_res
ADD	0000	$a + b$
SUB	1000	$a - b$
SLL	0001	$a \ll b$
SLT	0010	$a <_s b$
SLTU	0011	$a <_u b$
XOR	0100	$a \wedge b$
SRL	0101	$a \gg_u b$
SRA	1101	$a \gg_s b$
OR	0110	$a   b$
AND	0111	$a \& b$

# 实验原理

## 2.3存储器IP核

### □ 存储器IP核介绍

- ✓ 两种IP类型：分布式 (Distributed) 、块式 (Block) 存储器
- ✓ 定制化方式：ROM/RAM、单端口/简单双端口/真正双端口等
- ✓ 分布式存储器端口 (单端口)
  - 同步写端口：a (地址), d (数据), we (写使能), clk
  - 异步读端口：a (地址), spo (数据)
- ✓ 块式存储器端口 (单端口)
  - 同步写端口：addr (地址), din (数据), we (写使能), clk
  - 同步读端口：addr (地址), dout (数据), clk
  - 使能端口：en (读、写使能)



# 实验原理

## 2.3 存储器IP核

### □ IP核生成方式

**Flow Navigator >> Project Manager >> IP Catalog**

#### ✓ 途径1:

- Memories & Storage Elements >> RAMs & ROMs >> Distributed Memory Generator (分布式存储器)
- Memories & Storage Elements >> RAMs & ROMs& BRAMs >> Block Memory Generator (块式存储器)

#### ✓ 途径2:

- Basic Elements >> Memory Elements >> Distributed Memory Generator(分布式存储器)
- Basic Elements >> Memory Elements >> Block Memory Generator(块式存储器)

# 实验原理

## 2.3 存储器IP核

### □ IP核生成方式

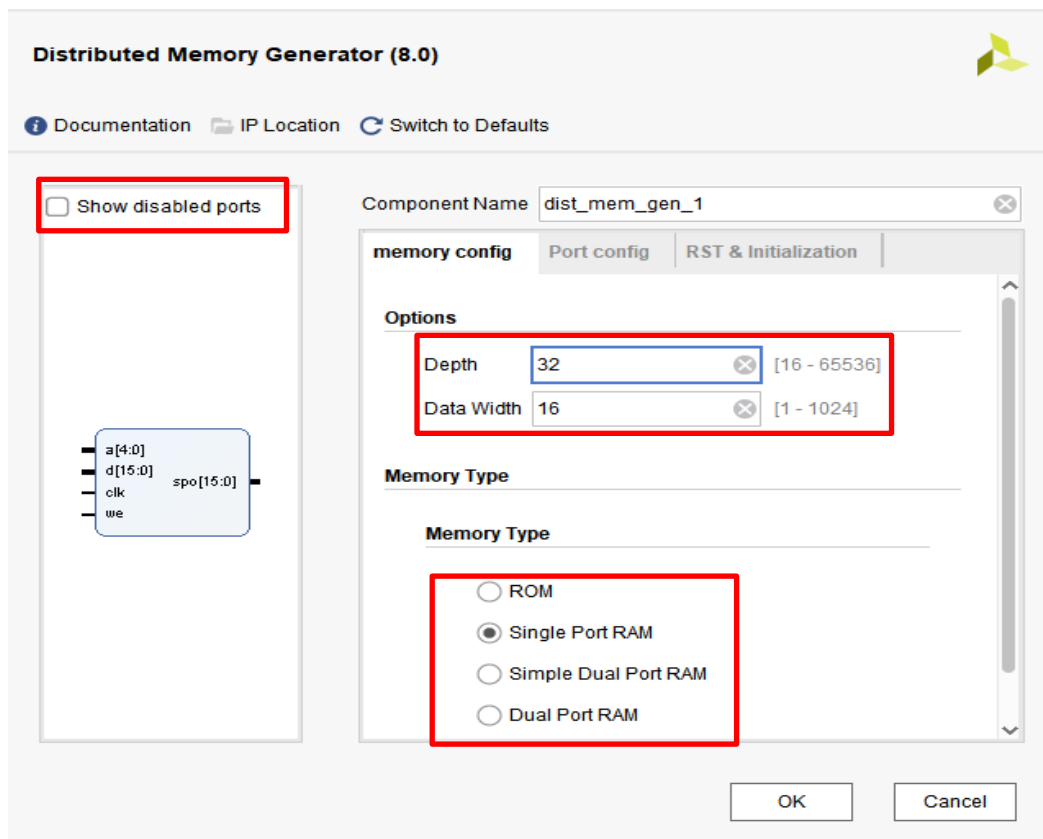
The screenshot displays the Vivado IP Catalog interface. On the left, the 'Flow Navigator' pane shows the 'IP Catalog' option selected under the 'IP INTEGRATOR' section. The main pane shows a search for 'mem' with 7 matches. The search results are organized into a tree structure. The 'Basic Elements' folder is expanded, showing 'Memory Elements' which contains 'Block Memory Generator' and 'Distributed Memory Generator'. The 'Memories & Storage Elements' folder is also expanded, showing 'ECC', 'FIFOs', 'Memory Interface Generators', 'RAMs & ROMs' (which is highlighted), and 'RAMs & ROMs & BRAM'. The 'RAMs & ROMs' folder contains 'Distributed Memory Generator' and 'Block Memory Generator'. The 'Block Memory Generator' is selected, and its details are shown in the table below.

Name	AXI4	Status	License	VLNV
Vivado Repository				
AXI Infrastructure				
Basic Elements				
Memory Elements				
Block Memory Generator	AXI4	Production	Included	xilinx.com:ip:blk_mem_gen:8.4
Distributed Memory Generator		Production	Included	xilinx.com:ip:dist_mem_gen:8.0
Communication & Networking				
Embedded Processing				
Memories & Storage Elements				
ECC		Production	Included	xilinx.com:ip:ecc:2.0
FIFOs				
Memory Interface Generators				
RAMs & ROMs				
Distributed Memory Generator		Production	Included	xilinx.com:ip:dist_mem_gen:8.0
RAMs & ROMs & BRAM				
Block Memory Generator	AXI4	Production	Included	xilinx.com:ip:blk_mem_gen:8.4
Video & Image Processing				

# 实验原理

## 2.3 存储器IP核

### □ 分布式存储器IP核参数设置及初始化



# 实验原理

## 2.3 存储器IP核

### □ 分布式存储器IP核参数设置及初始化

memory config | **Port config** | RST & Initialization

Input Options

Input Options

☒ Non Registered ☐ Registered

☐ Input Clock Enable ☐ Qualify WE with I\_CE

Dual Port Address

Dual Port Address

☒ Non Registered ☐ Registered

Output Options



Output Options

☒ Non Registered ☐ Registered ☐ Both



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 :  

# 实验原理

## 2.3 存储器IP核

### □ 块式存储器IP核参数设置及初始化

**Block Memory Generator (8.4)**

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

**IP Symbol** | **Power Estimation**

☐ Show disabled ports

BRAM\_PORTA

- addr[3:0]
- clka
- dina[15:0]
- douta[15:0]
- ena
- wea[0:0]

Component Name: blk\_mem\_gen\_0

**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: Simple Dual Port RAM

☐ Error Injection

**Write Enable**

☐ Byte Write Enable

Byte Size (bits): 9

**Algorithm Options**

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

Algorithm: Minimum Area

Primitive: 8kx2

# 实验原理

## 2.3 存储器IP核

### □ 块式存储器IP核参数设置及初始化

Basic	Port A Options	Other Options	Summary
-------	----------------	---------------	---------

Memory Size

Write Width	16	Range: 1 to 4608 (bits)
Read Width	16	
Write Depth	16	Range: 2 to 1048576
Read Depth	16	

Operating Mode: Write First

Enable Port Type: Always Enabled

Port A Optional Output Registers

<input type="checkbox"/> Primitives Output Register	<input type="checkbox"/> Core Output Register
<input type="checkbox"/> SoftECC Input Register	<input type="checkbox"/> REGCEA Pin

Basic	Port A Options	Other Options	Summary
-------	----------------	---------------	---------

Pipeline Stages within Mux: 0 Mux Size: 1x1

Memory Initialization

☒ Load Init File

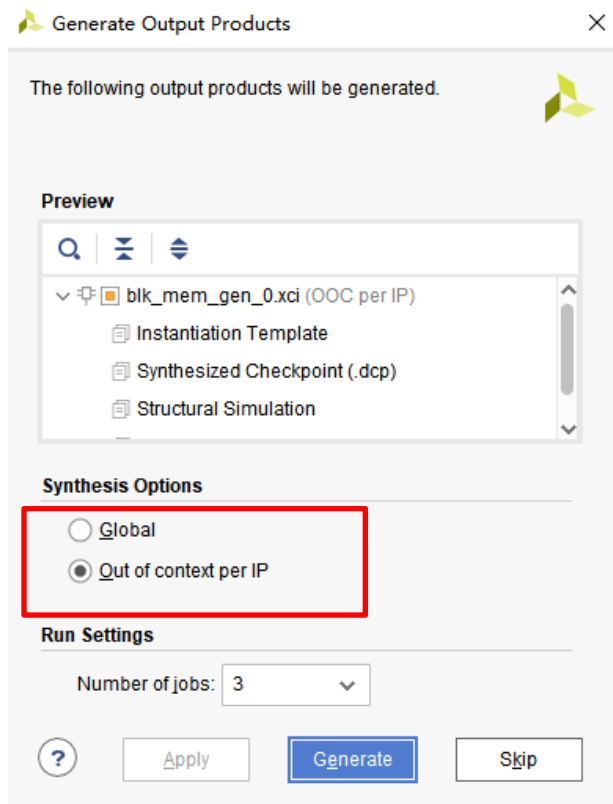
Coe File: o proj/CS EXP2023/LAB2/RAM/BRAM/BRAM/data\_ram.coe

Browse Edit

# 实验原理

## 2.3 存储器IP核

### □ IP核综合方式



#### ✓ Global模式:

- 全局综合：每次工程综合时，IP核源码都会被综合；
- 不会产生.dcp文件；
- 综合的时间长。

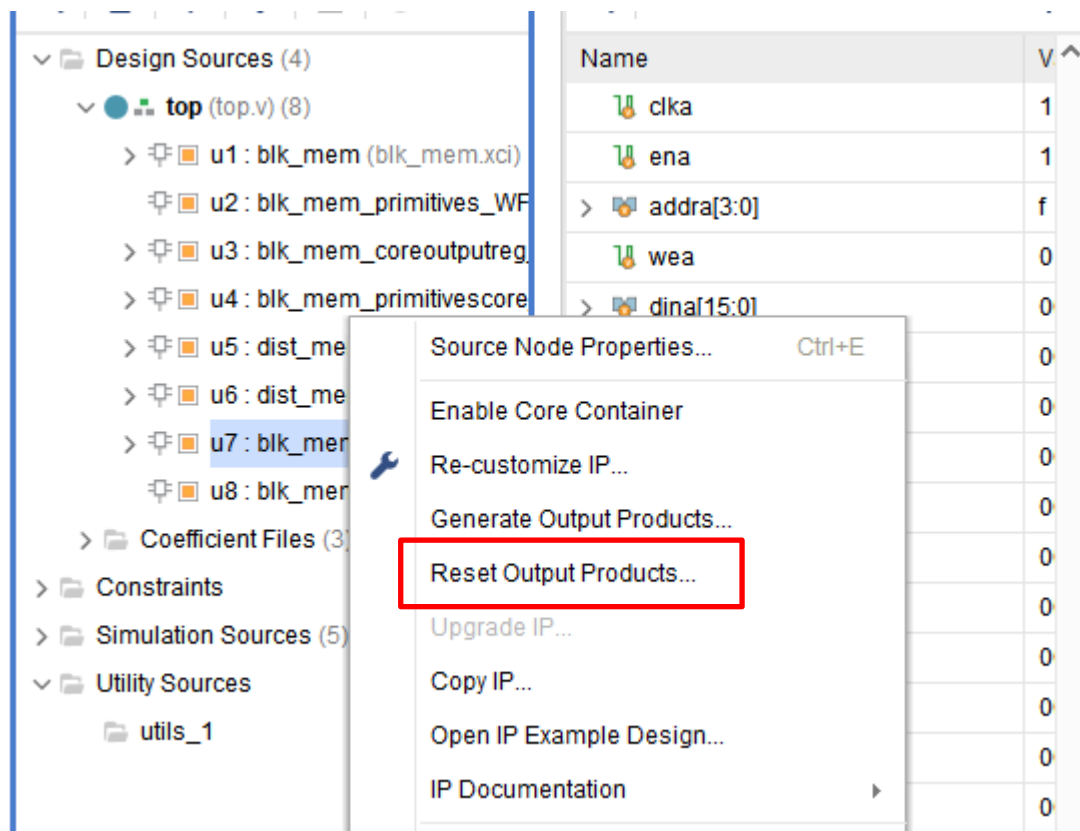
#### ✓ OOC模式:

- 对IP进行单独综合，生成.dcp文件；
- 工程用到IP时，只需从.dcp文件中解析出对应IP的网表文件，而不再对IP核的源文件重新综合；
- 可以加快综合的速度；
- 生成IP核后，如果还需要对IP的参数或初始值进行修改，则需要先对IP核进行复位，然后重新选择OOC模式生成IP核。

# 实验原理

## 2.3 存储器IP核

## □ IP核综合方式





# 实验原理

## 2.3 存储器IP核

### □ 实例化IP核

Project Manager – display >> Sources >> IP Sources

– IP >> dist\_mem\_gen\_0 >> Instantiation Template >> dist\_mem\_gen\_0.vco

```
dist_mem_gen_0  your_instance_name (  
    .a(a),           // input wire [3 : 0] a  
    .d(d),           // input wire [7 : 0] d  
    .clk(clk),       // input wire clk  
    .we(we),         // input wire we  
    .spo(spo)        // output wire [7 : 0] spo  
);
```

# 实验原理

## 2.3 存储器IP核

### □ 实例化IP核

The screenshot displays the Vivado IDE interface during the instantiation of a memory IP core. On the left, the 'Sources' window shows a project tree where the 'dist\_mem\_gen\_0' IP core is expanded. The 'dist\_mem\_gen\_0.veo' file is highlighted with a red box. Below the tree, the 'IP Sources' tab is selected and also highlighted with a red box. On the right, the 'dist\_mem\_gen\_0.veo' file is open, showing Verilog code. A red box highlights the instantiation template section, which includes comments and the actual instantiation code for 'dist\_mem\_gen\_0'.

```
46 //
47 // DO NOT MODIFY THIS FILE.
48
49 // IP VLNV: xilinx.com:ip:dist_mem_gen:8.0
50 // IP Revision: 13
51
52 // The following must be inserted into your Verilog file for
53 // core to be instantiated. Change the instance name and port
54 // (in parentheses) to your own signal names.
55
56 //----- Begin Cut here for INSTANTIATION Template -----//
57 dist_mem_gen_0 your_instance_name (
58   .a(a),      // input wire [3 : 0] a
59   .d(d),      // input wire [7 : 0] d
60   .clk(clk),  // input wire clk
61   .we(we),    // input wire we
62   .spo(spo)   // output wire [7 : 0] spo
63 );
64 // INST_TAG_END ----- End INSTANTIATION Template -----
65
```

# 实验原理

## 3. FPGAOL实验平台使用

- 登录平台网站：fpgaol.ustc.edu.cn，使用统一身份认证登录，或者直接以游客身份登录

### Login to FPGAOL

体验FPGAOL新界面！

科大统一身份认证登录

其他学校认证登录

游客访问 / Visitor Login

# 实验原理

## 3. FPGAOL实验平台使用

□ 设备获取：点击“acquire”按钮获取一个FPGA结点

✓ 成功后在下方link栏将显示链接，通过链接进入设备操作界面

✓ 默认使用时长20分钟(自动释放结点，点击“release”按钮手动释放)

体验FPGAOL新界面！						
#	Device Type	Vacant/Total	Manual	xdc	Use	
1	FPGAOL 1.0	69 / 70	<a href="#">FPGAOL v2.1 manual</a>	<a href="#">fpgaol1.xdc</a>	<a href="#">acquire</a>	<a href="#">release</a>
Device ID		133				
Acquire Time		March 21, 2024, 4:23 p.m.				
Expiration Time		March 21, 2024, 4:43 p.m.				
Panel Link		<a href="http://202.38.79.134:12133/?token=HA3TMYRYMZTGEMRUHAYTGODFGMYTGODDMYYWKMBQHFRWCMJQGY3GEYZXHA2GCYRQP0614">http://202.38.79.134:12133/?token=HA3TMYRYMZTGEMRUHAYTGODFGMYTGODDMYYWKMBQHFRWCMJQGY3GEYZXHA2GCYRQP0614</a>				
XVC Server		202.38.79.134:-2				

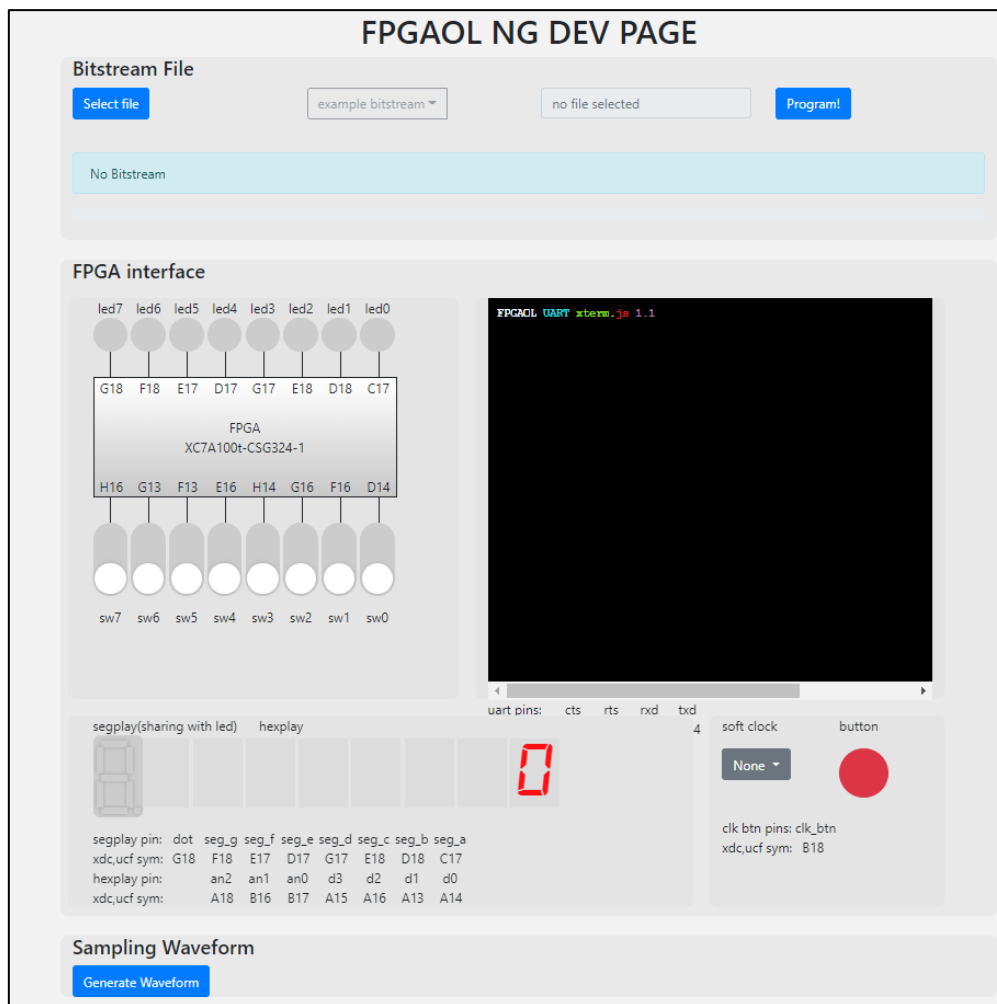
[FPGAOL v2.1 使用说明 \(WIP\)](#)

# 实验原理

## 3. FPGAOL实验平台使用

### □ 烧写FPGA

- ✓ 点击 “Select file” 按钮
- ✓ 选择需要烧写的bit文件
- ✓ 点击 “Program! ”



# 实验内容

---

## 1. 寄存器堆设计及仿真

- 正确实现寄存器堆的逻辑设计；
- 完成寄存器堆电路的功能仿真。

## 2. 32位算术逻辑单元 (ALU) 设计及仿真

- 正确实现ALU的逻辑设计；
- 完成ALU的功能仿真。

## 3. 算术逻辑单元 (ALU) 应用-计算器设计

- 正确实现计算器电路的逻辑设计；
- 完成计算器电路下载测试。

## 4. 存储器IP核例化及初始化

- 完成存储器（块式或分布式）IP核例化及初始内容加载。

**The End**