

实验二 寄存器堆与存储器 及其应用

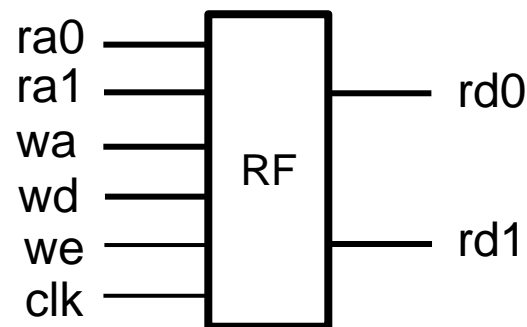
实验目标

- 掌握寄存器堆（**Register File**）和存储器的功能、时序及其应用
- 熟练掌握数据通路和控制器的设计和描述方法

实验内容

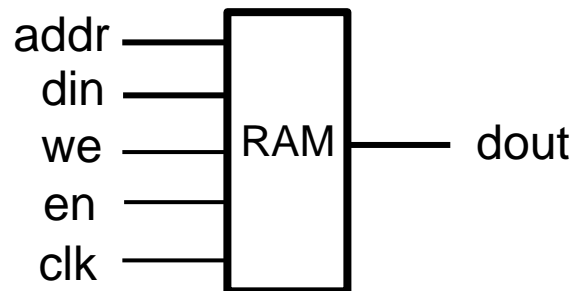
1. 寄存器堆（Register File）

- clk: 时钟
- ra0, rd0: 异步读端口0
- ra1, rd1: 异步读端口1
- wa, wd, we: 同步写端口



2. RAM存储器

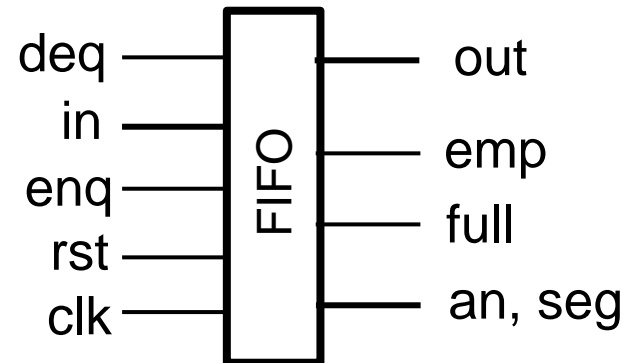
- clk: 时钟
- en: 总使能
- we: 写使能
- addr: 读/写地址
- din: 输入数据
- dout: 输出数据



实验内容（续）

3. 利用寄存器堆实现FIFO队列

- enq, deq: 入队列和出队列使能，假定两者是互斥的，高电平有效且均要求一次有效仅允许操作一项数据
- in, out: 入/出队列数据
- full, emp: 队列满/空标志，在满或空时忽略入/出队操作
- an, seg: 数码管控制信号，显示队列数据
- clk, rst: 时钟，复位



寄存器堆

- 也称寄存器文件(Register File)

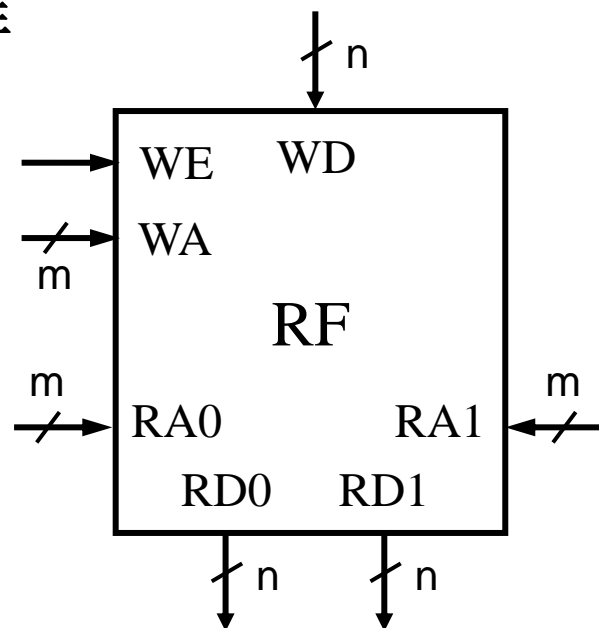
例如，三端口的 $2^m \times n$ 位寄存器堆

1个写端口

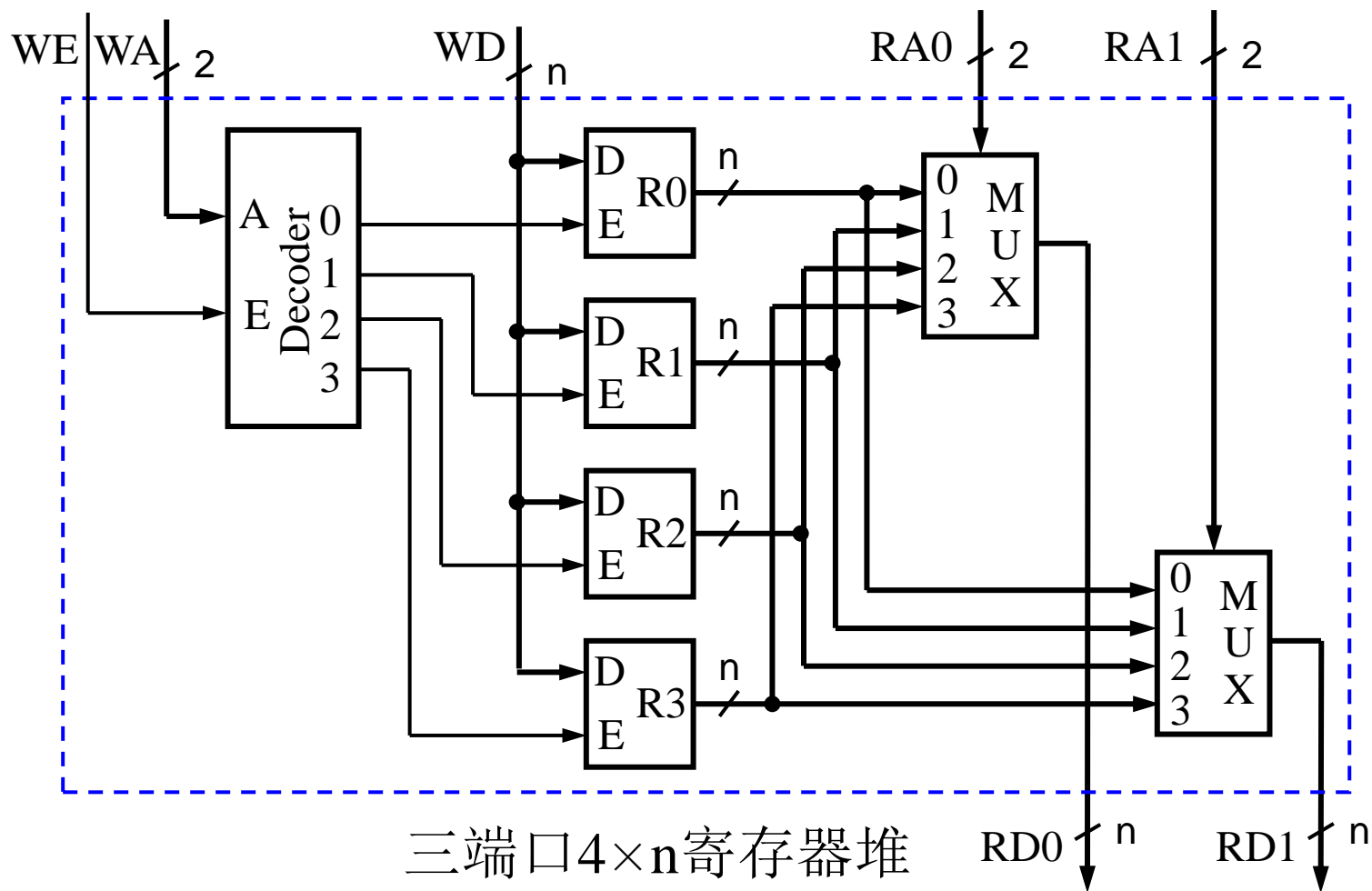
- WA: 写地址
- WD: 写入数据
- WE: 写使能

2个读端口

- RA0、RA1: 读地址
- RD0、RD1: 读出数据



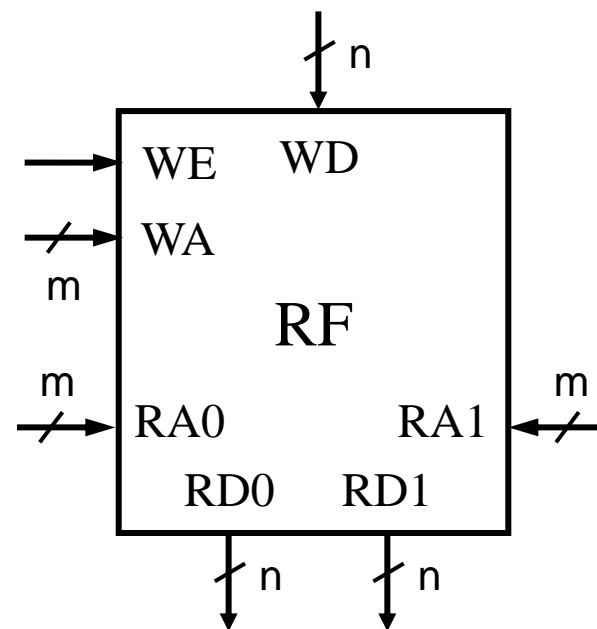
寄存器堆 (续)



寄存器堆 (续)

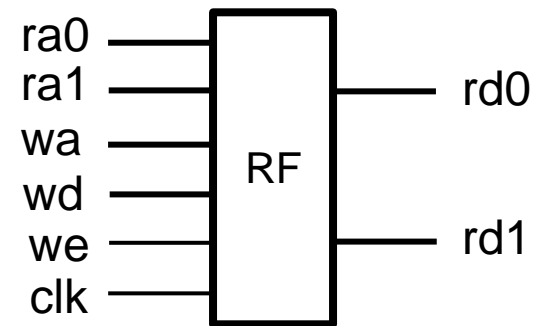
// 三端口8×4寄存器堆行为描述

```
wire [2:0] wa, ra0, ra1;  
wire [3:0] wd, rd0, rd1;  
reg [3:0] regfile[0:7];  
  
assign rd0 = regfile[ra0],  
        rd1 = regfile[ra1];  
  
always @ (posedeg clk) begin  
    if (we) regfile[wa] <= wd;  
end
```



寄存器堆端口定义

```
module register_file           //32 x WIDTH 寄存器堆
    #(parameter WIDTH = 32)   //数据宽度
    (clk,                      //时钟（上升沿有效）
    input [4:0] ra0,           //读端口 0 地址
    output [WIDTH-1:0] rd0,    //读端口 0 数据
    input [4:0] ra1,           //读端口 1 地址
    output [WIDTH-1:0] rd1,    //读端口 1 数据
    input [4:0] wa,            //写端口地址
    input we,                  //写使能，高电平有效
    input [WIDTH-1:0] wd       //写端口数据
    );
```



存储器IP核

- **Vivado**中有存储器IP核可以直接使用
- 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

存储器IP核例化

- **Flow Navigator >> Project Manager >> IP Catalog**
 - Memories & Storage Elements >> RAMs & ROMs >> Distributed Memory Generator
 - 或者 Basic Elements >> Memory Elements >> Distributed Memory Generator
 - Memory config >> Memory Type: [Simple Dual Port RAM](#)
 - RST & Initialization >> [Load COE File](#)

同步写端口：a (地址), d (数据), we (写使能), clk

异步读端口：dpra (地址), dpo (数据)

存储器IP核例化 (续)

- **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 [15 : 0] a  
  .d(d),          // input wire [11 : 0] d  
  .dpra(dpra),    // input wire [15 : 0] dpra  
  .clk(clk),      // input wire clk  
  .we(we),        // input wire we  
  .dpo(dpo)       // output wire [11 : 0] dpo  
);
```

实例化模板

COE文件格式

- **An example COE file:**

; Sample Initialization file for a 32x16 distributed ROM

memory_initialization_radix = 16;

memory_initialization_vector =

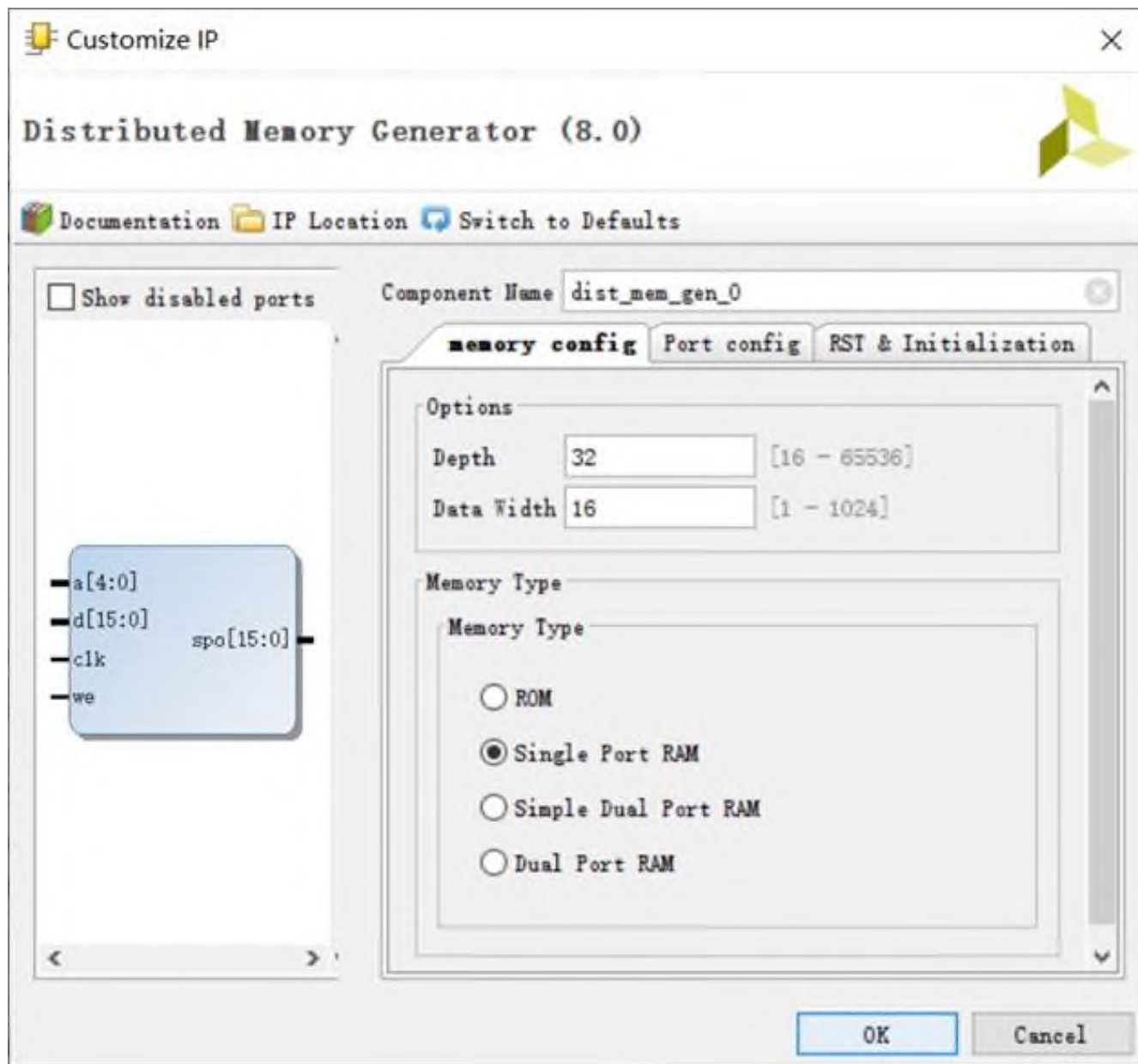
23f4 0721 11ff ABe1 0001 1 0A 0 逗号或空格分隔每项

23f4 0721 11ff ABe1 0001 1 0A 0 数据（不允许为负数）

23f4 721 11ff ABe1 0001 1 A 0

23f4 721 11ff ABe1 0001 1 A 0;

分布式存储器IP



分布式存储器IP

Component Name: dist_mem_gen_0

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

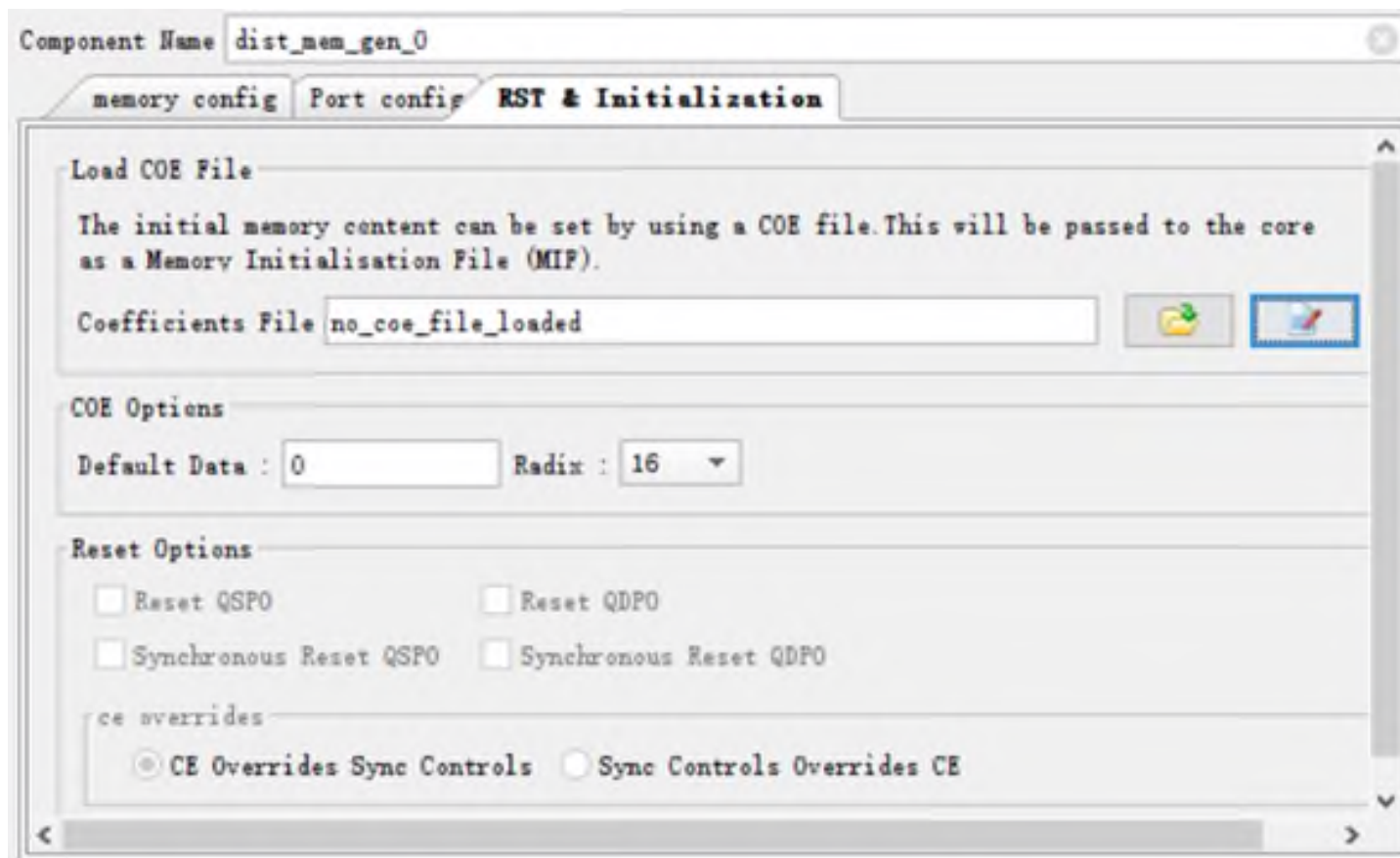
☐ Common Output CLK ☐ Single Port Output CE

☐ Common Output CE ☐ Dual Port Output CE

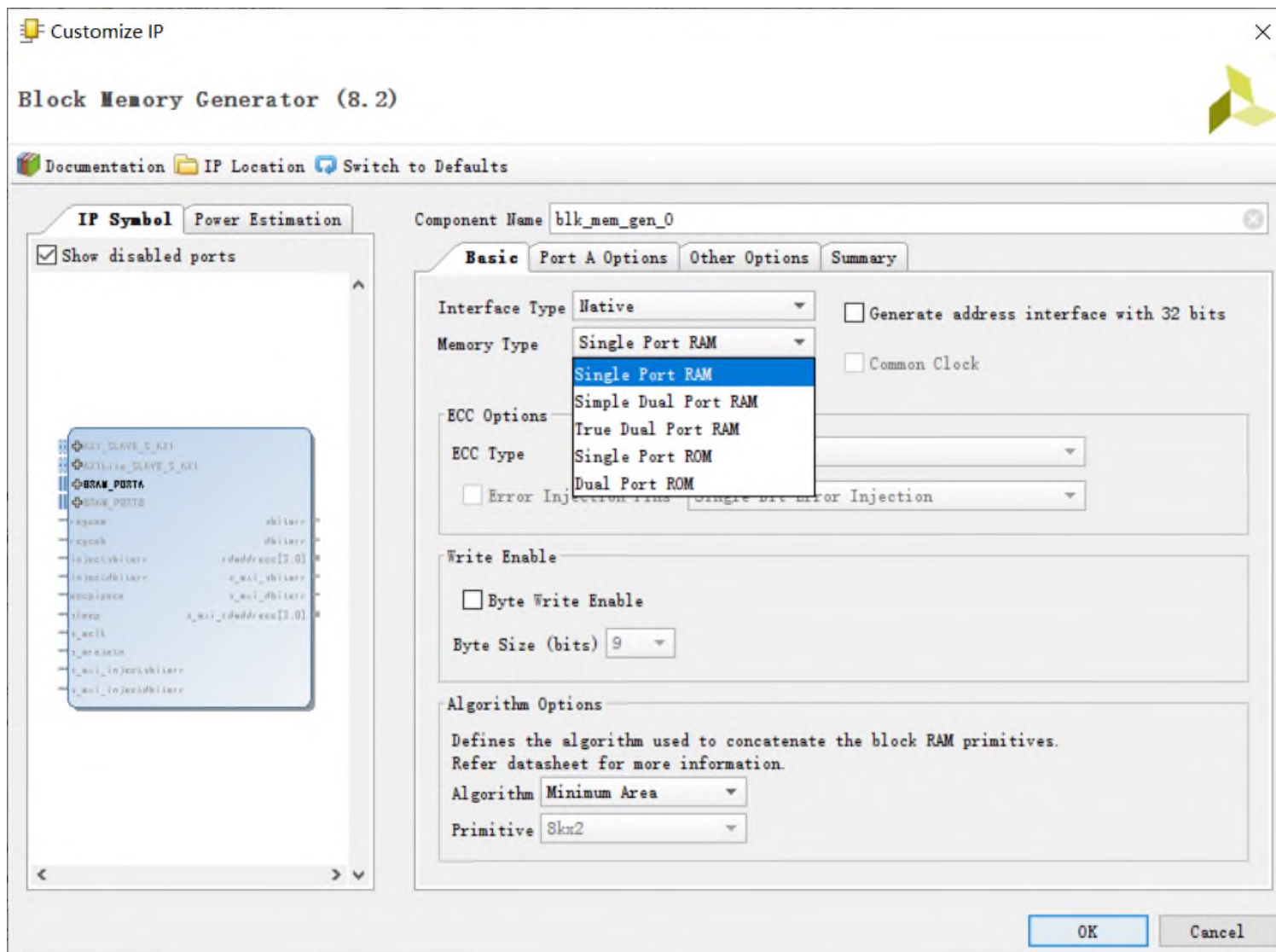
Pipelining Options

Pipeline Stages: 0

分布式存储器IP



块式存储器IP



块式存储器IP

Basic **Port A Options** Other Options Summary

Memory Size

Write Width 16 Range: 1 to 4608 (bits)

Read Width 16

Write Depth 32 Range: 2 to 9011200

Read Depth 32

Operating Mode Write First Enable Port Type Use ENA Pin

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)

块式存储器IP

Basic Port A Options **Other Options** Summary

Pipeline Stages within Mux Mux Size: 2x1

Memory Initialization

☐ Load Init File

Coe File

☐ Fill Remaining Memory Locations

Remaining Memory Locations (Hex)

Structural/UniSim Simulation Model Options

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

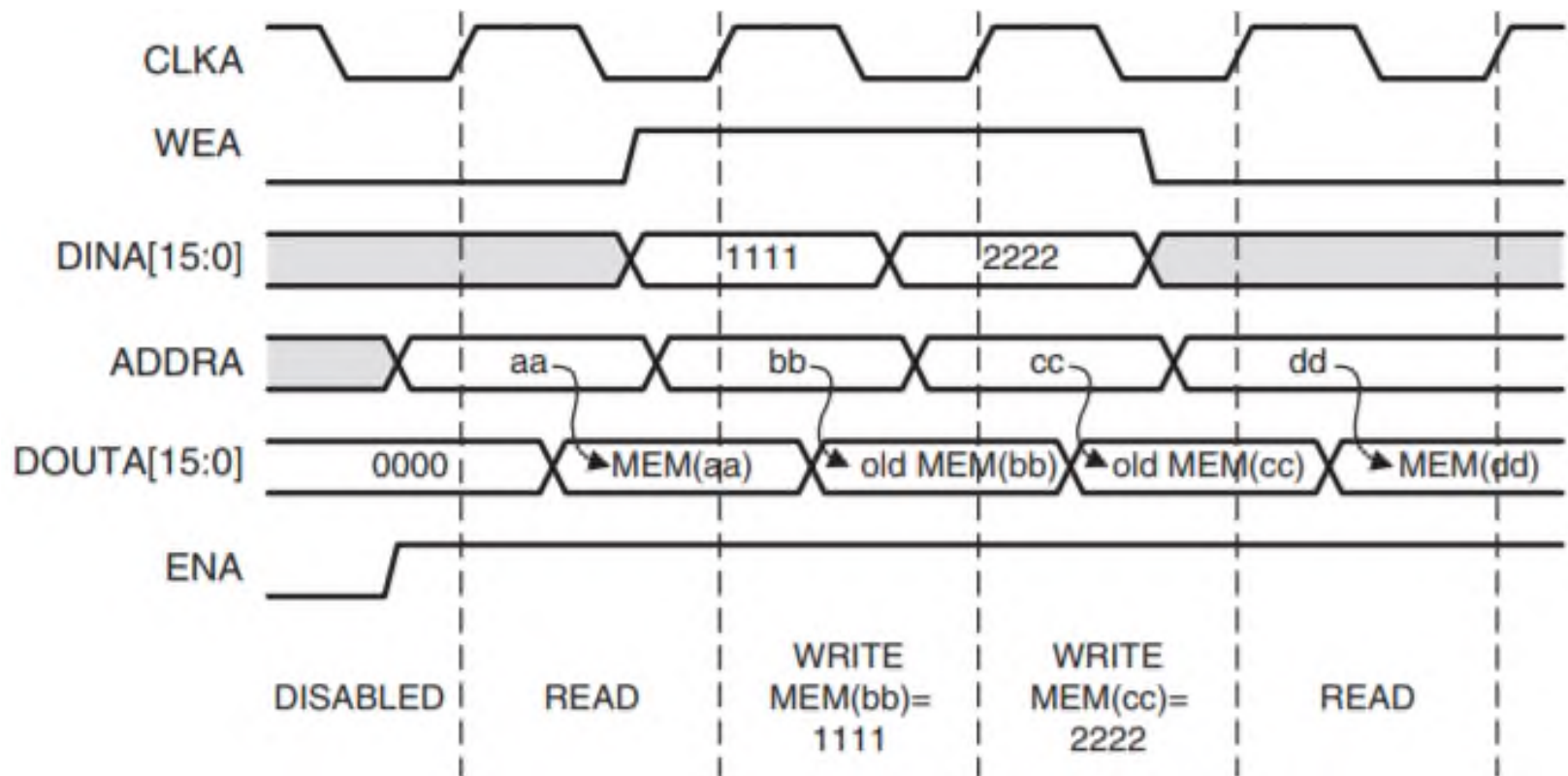
Collision Warnings

Behavioral Simulation Model Options

☐ Disable Collision Warnings ☐ Disable Out of Range Warnings

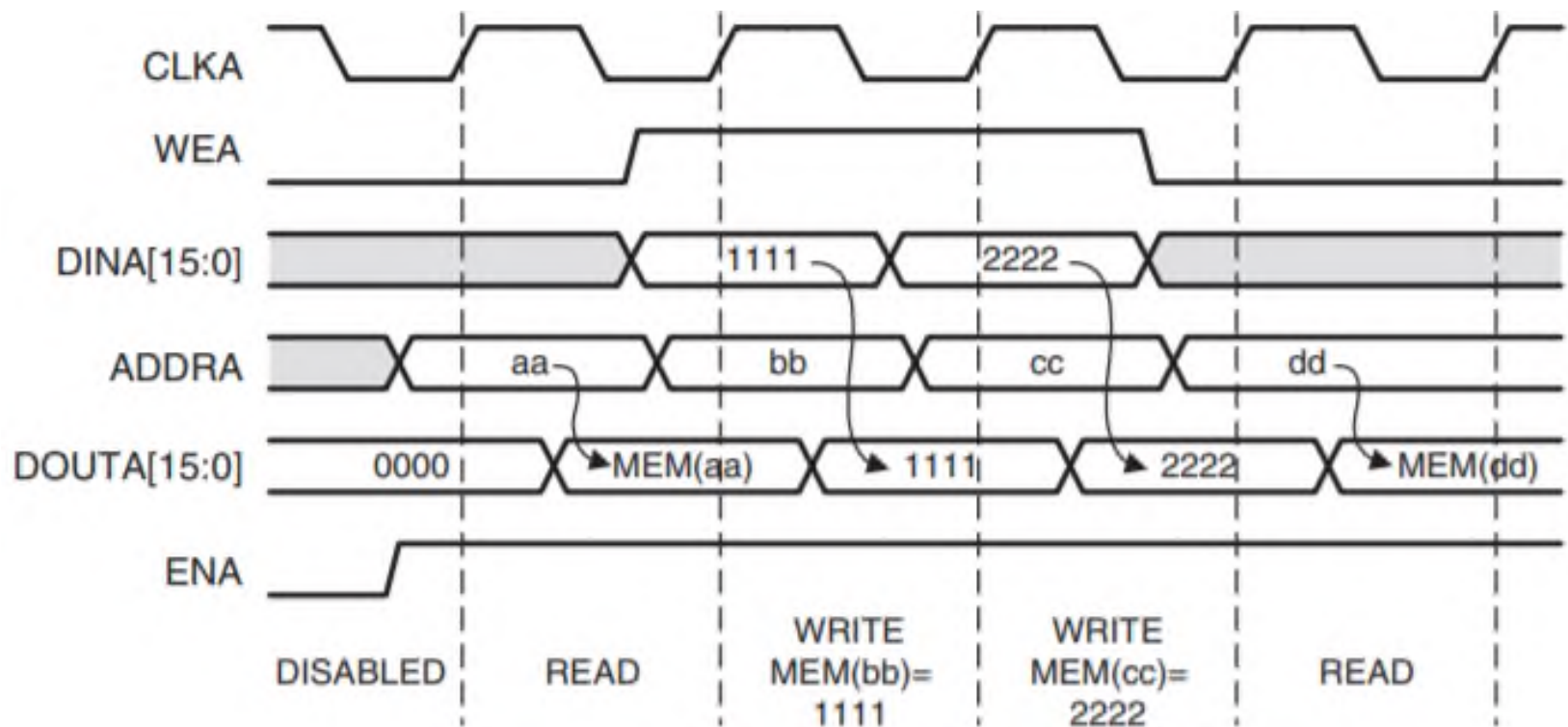
存储器时序

- Read First Mode



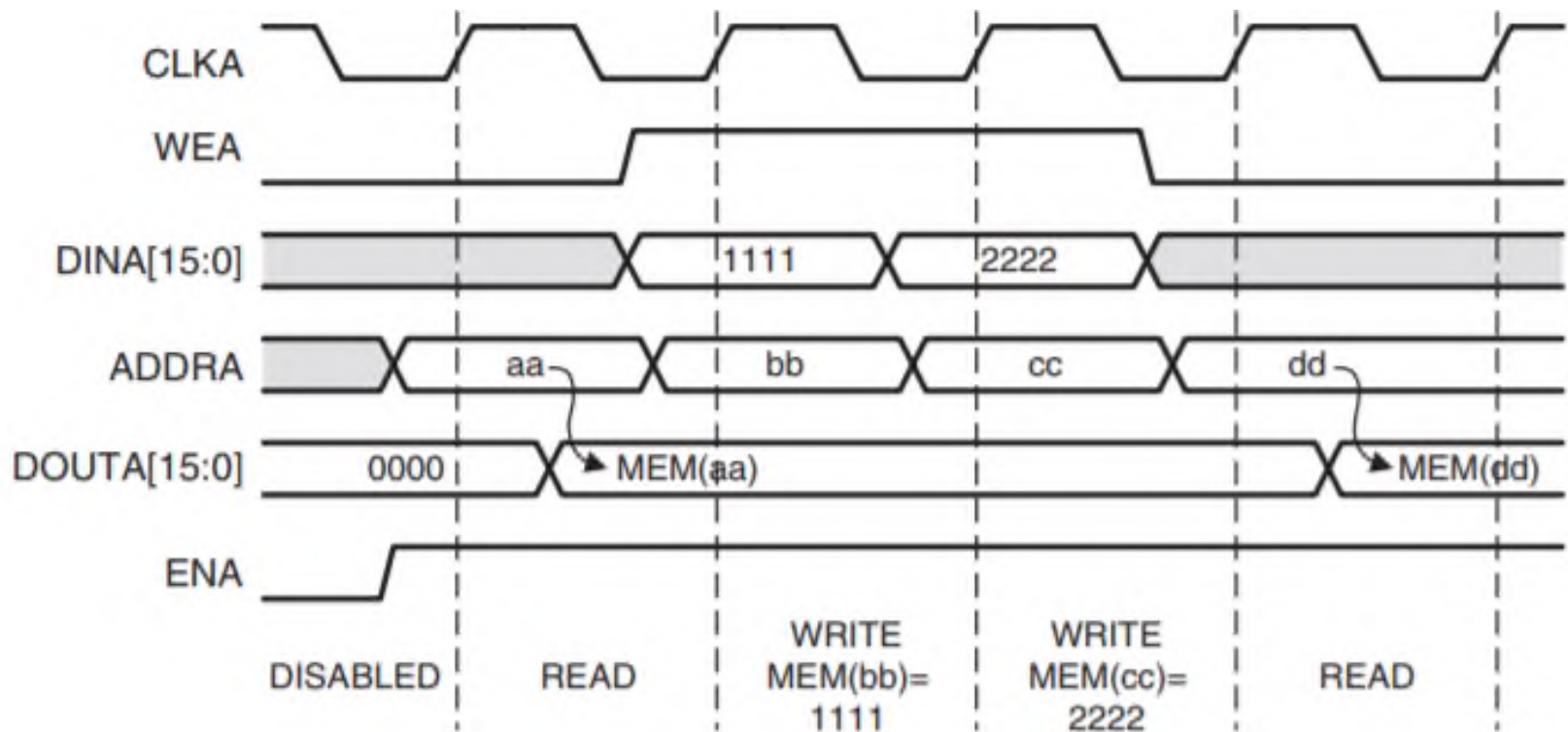
存储器时序 (续1)

- Write First Mode



存储器时序 (续2)

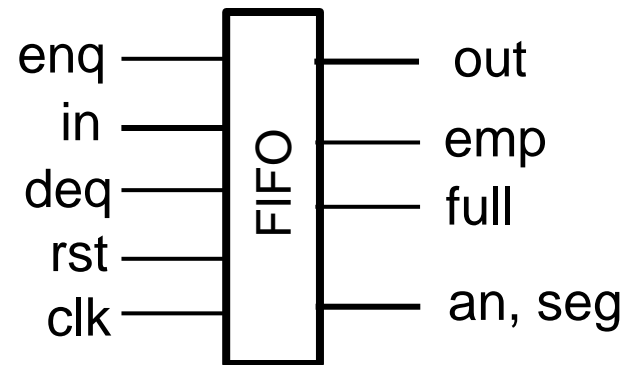
- No Change Mode



FIFO队列

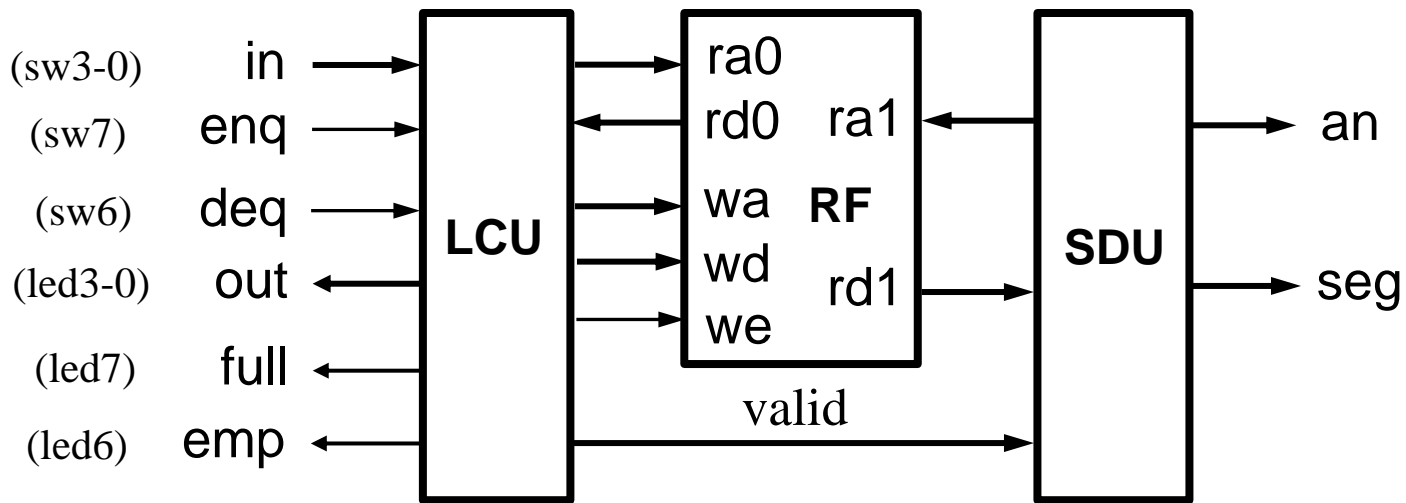
- 用三端口 8×4 寄存器堆实现最大长度为8的FIFO队列

- deq, enq: 出/入队列使用 (互斥), 一次有效仅允许操作一项数据
- out, in: 出/入队列数据
- full, emp: 队列满/空, 满/空时忽略入/出队操作
- an, seg: 数码管控制信号, 显示队列状态



FIFO队列逻辑结构

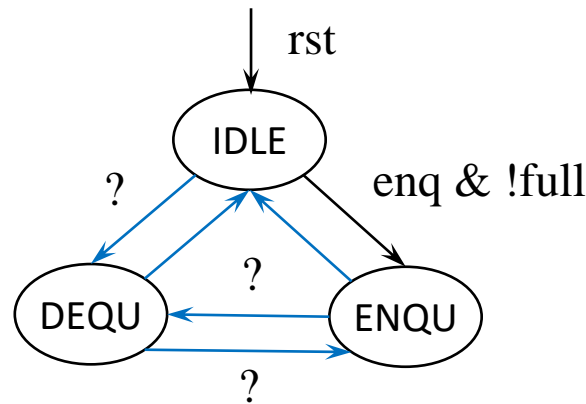
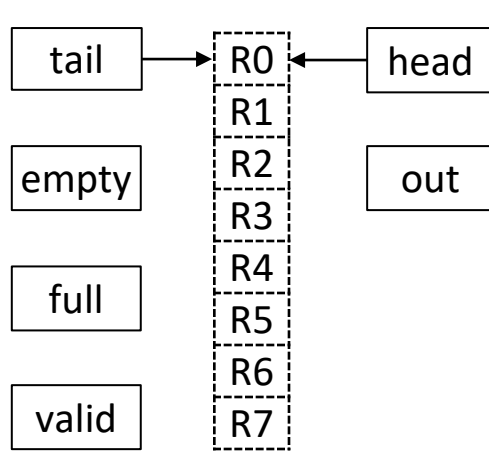
- 队列控制单元 **LCU (List Control Unit)**
 - 处理出/入队操作，显示队列空/满状态
- 数码管显示单元 **SDU (Segment Display Unit)**
 - 显示队列数据内容



* 省略了clk (100MHz) 和 rst (button)

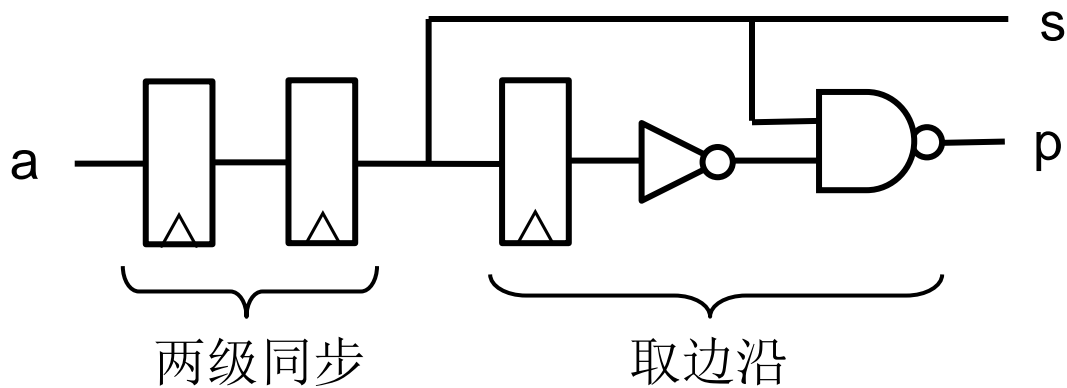
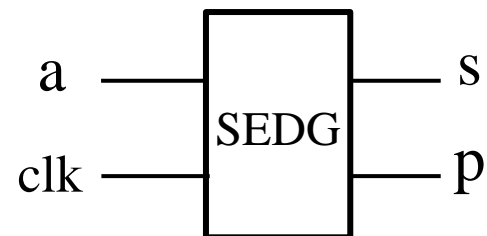
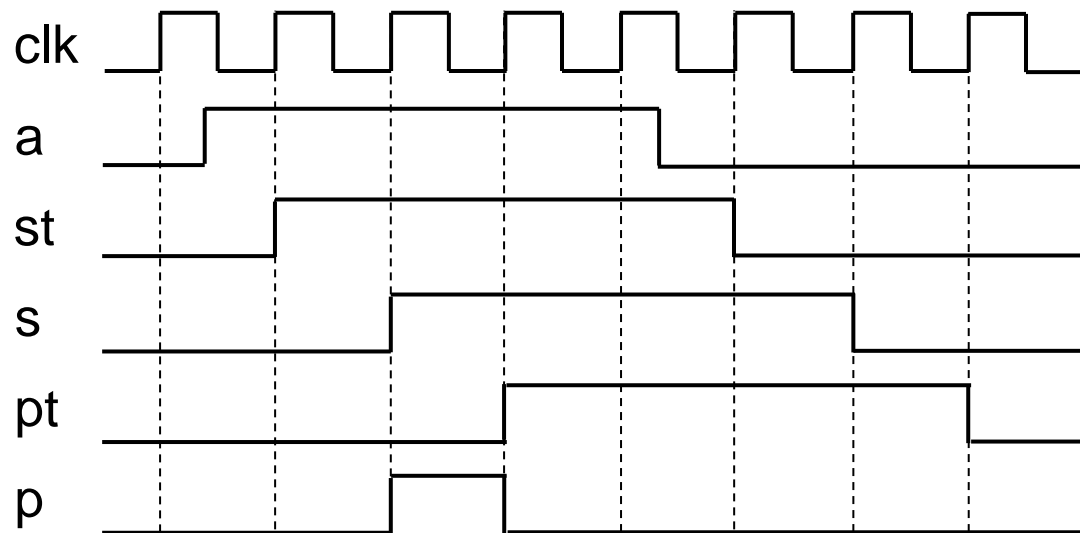
队列控制单元

- **head**: 3位, 队头, 指向出队列数据
- **tail**: 3位, 队尾, 指向入队列位置
- **full, empty**: 各1位, 满和空标志
- **valid**: 8位, 有效标志, 第*i*位对应*R_i*状态
- **out**: 4位, 出队列数据



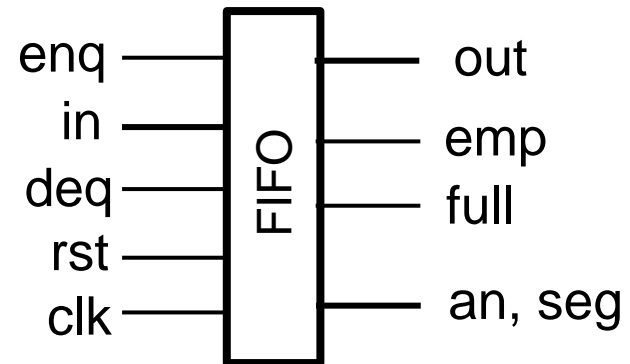
$RF[T] \leftarrow in, V[T] \leftarrow 1,$
 $T \leftarrow T+1,$
 $F \leftarrow (T+1) == H,$
 $E \leftarrow 0;$

异步信号同步和取边沿



FIFO端口定义

```
module fifo (  
    input clk, rst,    //时钟（上升沿有效）、同步复位（高电平有效）  
    input enq,         //入队列使能，高电平有效  
    input [3:0] in,    //入队列数据  
    input deq,         //出队列使能，高电平有效  
    output [3:0] out,  //出队列数据  
    output [2:0] an,    //数码管选择  
    output [3:0] seg   //数码管数据  
);
```



实验步骤

1. 行为方式参数化描述寄存器堆，功能仿真
2. IP例化分布式和块式16 x 8位单端口RAM，功能仿真和对比
3. 设计FIFO队列电路的数据通路和控制器，结构化方式描述数据通路，Moore型FSM描述控制器，功能仿真
4. FIFO队列电路下载至FPGA中测试

The End