SystemVerilog

实验12：ROM + RAM

xgsun@fudan.edu.cn

孙晓光

2024-11-18

1

ROM

ROMs

方法1：数据直接写到代码中

异步

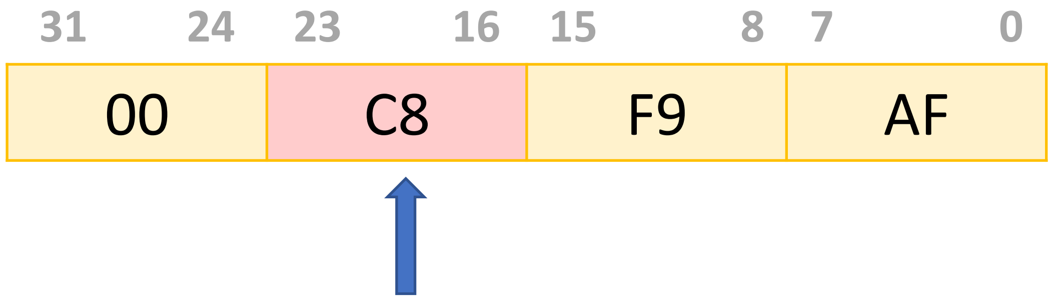
```
module ROM_asynRead(
    input  logic [2:0] addr,
    output logic [7:0] data );



    always_comb                      // image ROM
        case (addr)
            3’h0: data = 8’b0011_1100; //   ****
            3’h1: data = 8’b0111_1110; //  ******
            3’h2: data = 8’b1111_1111; // ********
            3’h3: data = 8’b1111_1111; // ********
            3’h4: data = 8’b1111_1111; // ********
            3’h5: data = 8’b1111_1111; // ********
            3’h6: data = 8’b0111_1110; //  ******
            3’h7: data = 8’b0011_1100; //   ****
        endcase

    logic rom_bit;                // ROM中每个像素值
    assign rom_bit  = data[3]; // ROM addr行中第3列值
endmodule
```

同步

```
module ROM_synRead( input  logic clk,
                    input  logic [2:0] addr,
                    output logic [7:0] data );
    logic [2:0] addr_reg;
    always_ff @(posedge clk)
        addr_reg <= addr;

    always_comb                      // image ROM
        case (addr_reg)
            3’h0: data = 8’b0011_1100; //   ****
            3’h1: data = 8’b0111_1110; //  ******
            3’h2: data = 8’b1111_1111; // ********
            3’h3: data = 8’b1111_1111; // ********
            3’h4: data = 8’b1111_1111; // ********
            3’h5: data = 8’b1111_1111; // ********
            3’h6: data = 8’b0111_1110; //  ******
            3’h7: data = 8’b0011_1100; //   ****
        endcase

    logic rom_bit;                // ROM中每个像素值
    assign rom_bit  = data[3]; // ROM addr行中第3列值
endmodule
```

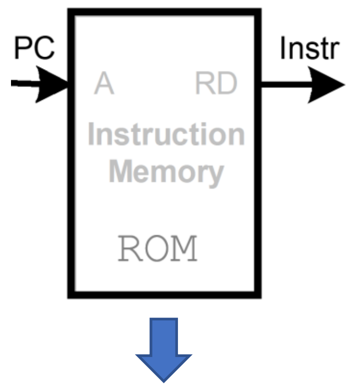
方法2：初始化时赋值

```
// 4x8bit ROM
module ROM2(
    input  logic [1:0] addr,      // 2^2 = 4
    output logic [7:0] oneWord );

    parameter data  = 32'h00_C8_F9_AF; //left index
    parameter N_BITS  = 8; //no. of bits in rom word
    parameter N_WORDS = 4; //no. of words in rom
    parameter TOTAL   = N_BITS * N_WORDS - 1;

    //       位宽                字宽
    logic [N_BITS-1 : 0] rom [0 : N_WORDS-1];

    integer i;
    initial
        for(i=0; i<N_WORDS; i=i+1)
            rom[i] = data[(TOTAL - N_BITS * i) -: N_BITS];
//若i=1, 则 rom[1] = data[( 31   - 8      * 1) -: 8]=data[23-:8]=data[23:16]

    assign oneWord = rom[addr];
endmodule
```



方法3：从文件中读取数据

如，从**指令存储器ROM**中取出指令



Verilog有2个**系统任务**，从**文件**中读取数据到**存储器**

```
$readmemb("<数据文件名>",<存储器名>,<起始地址>,<终止地址>);
$readmemh("<数据文件名>",<存储器名>,<起始地址>,<终止地址>);
```

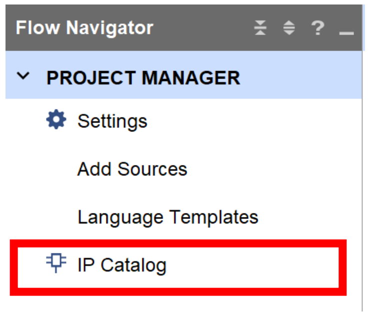
```verilog
// ROM: 64x32bit
module iMemory(input  logic [5:0]  addr ,
               output logic [31:0] data );

   //     位宽        字宽
   logic [31:0] ROM [63:0];     2^6 =64

   initial
      $readmemh("Test.dat", ROM);

   assign data = ROM[addr];
endmodule
```

Test.dat

```
20020005
2003000C
2067FFF7
00E22025
00642824
00A42820
10A7000A
0064202A
10800001
20050000
00E2202A
00853820
00E23822
AC670044
8C020050
08000011
20020001
AC020054
```

【参考】教材P276-278

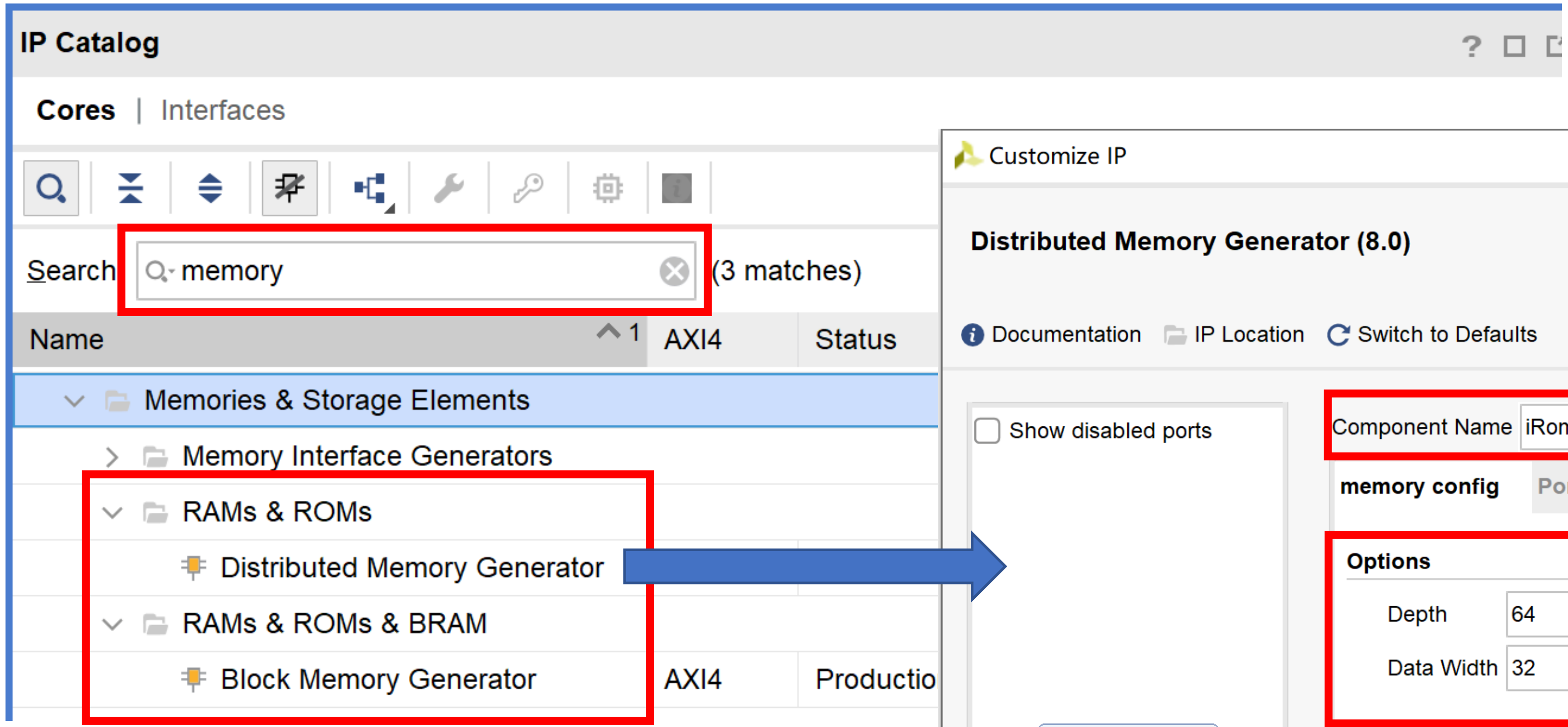
方法4：IP核


Flow Navigator

PROJECT MANAGER

- Settings
- Add Sources
- Language Templates
- IP Catalog

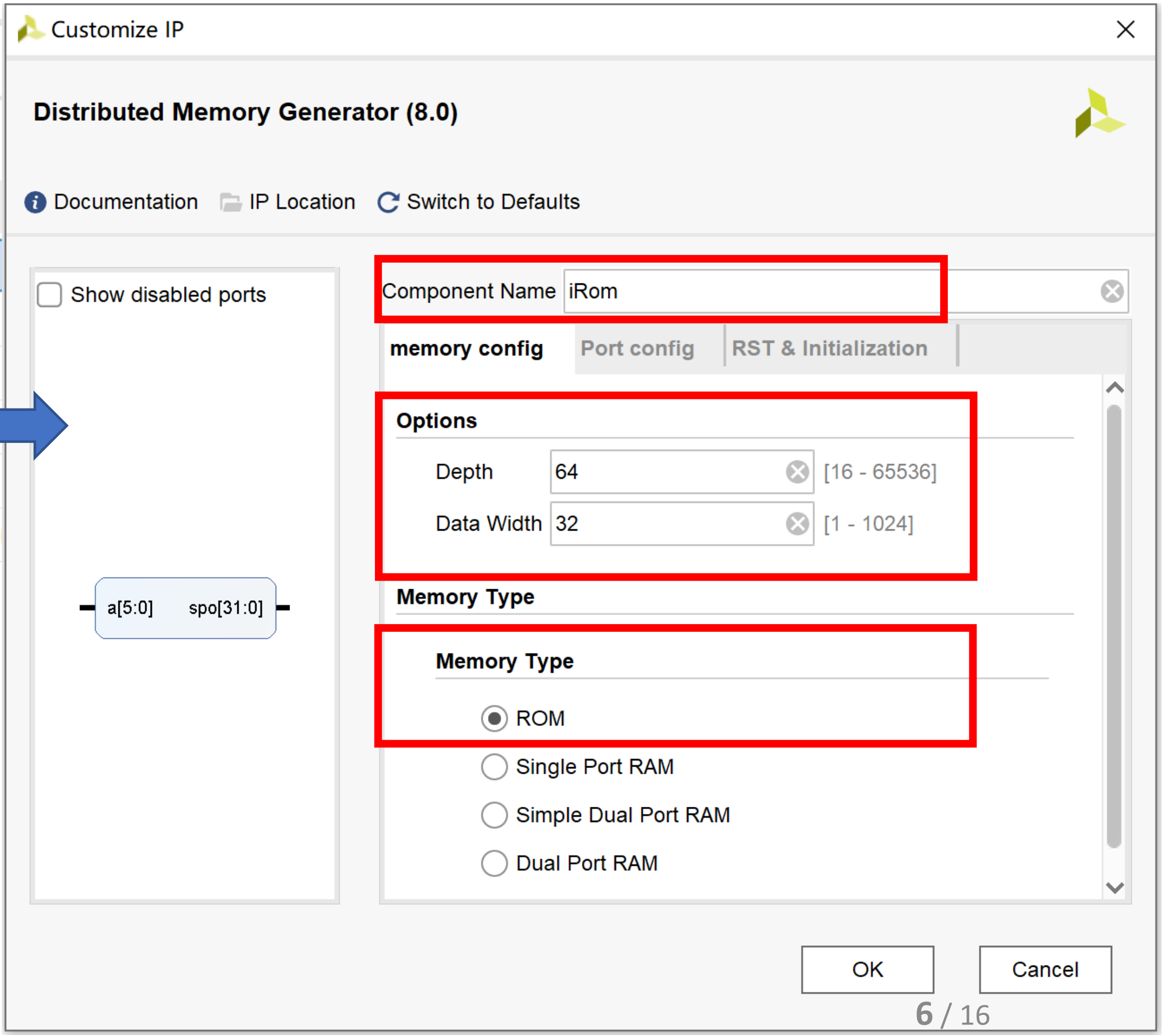
单击IP目录


IP Catalog

Cores | Interfaces

Search: memory (3 matches)

Name	AXI4	Status
Memories & Storage Elements		
Memory Interface Generators		
RAMs & ROMs		
Distributed Memory Generator		
RAMs & ROMs & BRAM		
Block Memory Generator	AXI4	Productio


Customize IP

Distributed Memory Generator (8.0)

Documentation | IP Location | Switch to Defaults

☐ Show disabled ports

a[5:0] spo[31:0]

Component Name: iRom

memory config | Port config | RST & Initialization

Options

Depth	64	[16 - 65536]
Data Width	32	[1 - 1024]

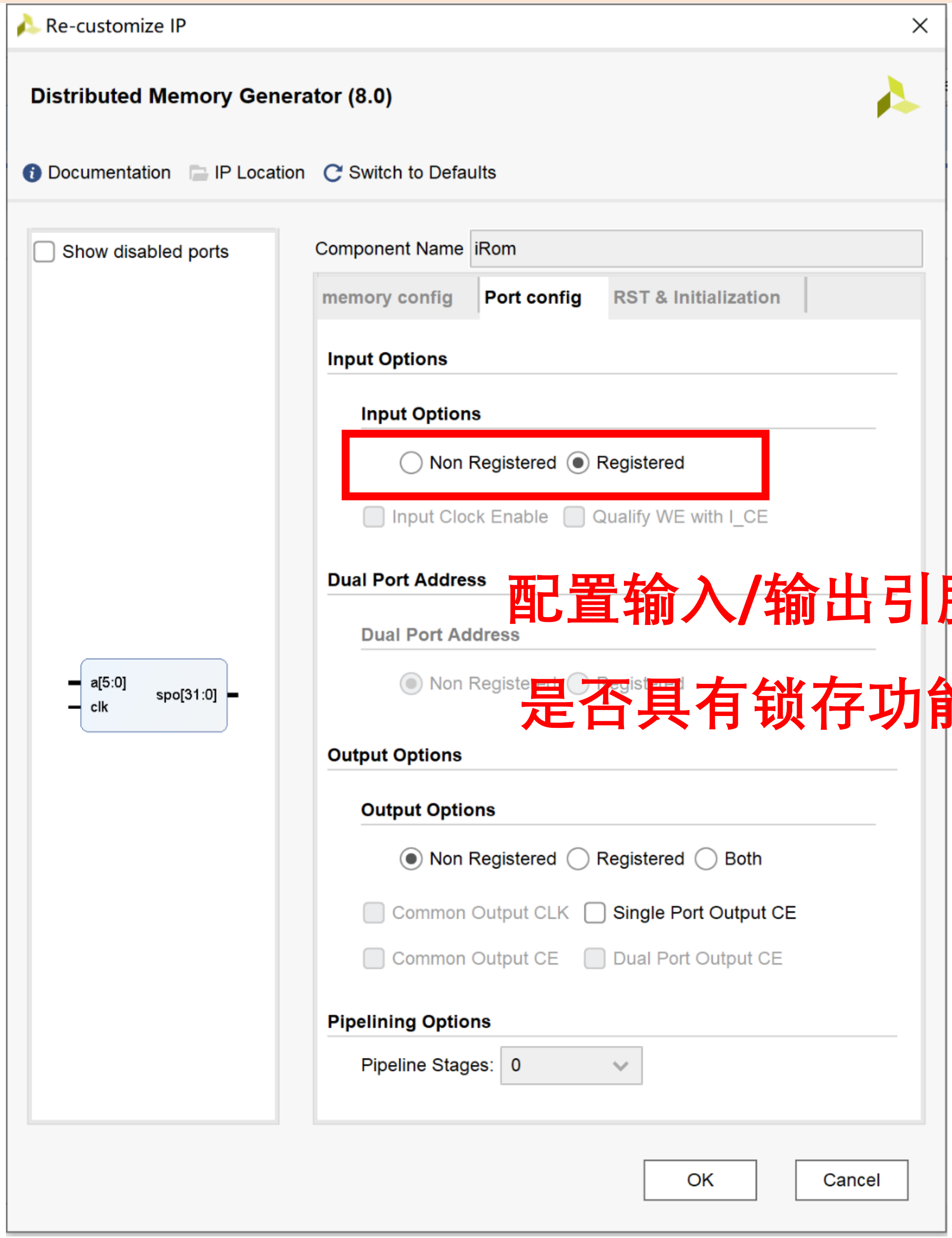
Memory Type

Memory Type

[✓ ROM]
Single Port RAM
Simple Dual Port RAM
Dual Port RAM

OK | Cancel

用 .coe 文件初始化ROM


Re-customize IP

Distributed Memory Generator (8.0)

Documentation | IP Location | Switch to Defaults

☐ Show disabled ports

Component Name: iRom

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

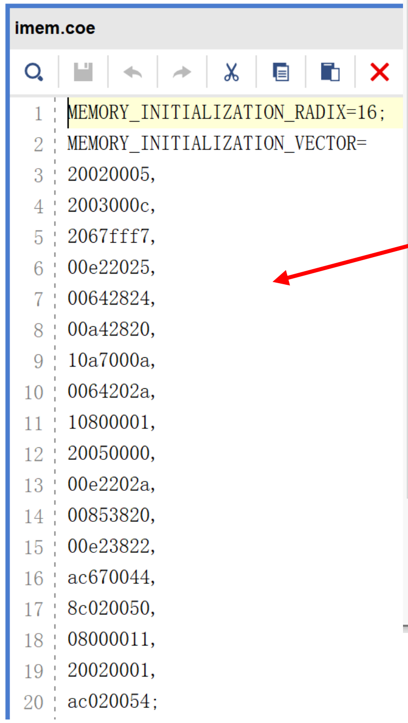
Pipelining Options

Pipeline Stages: 0

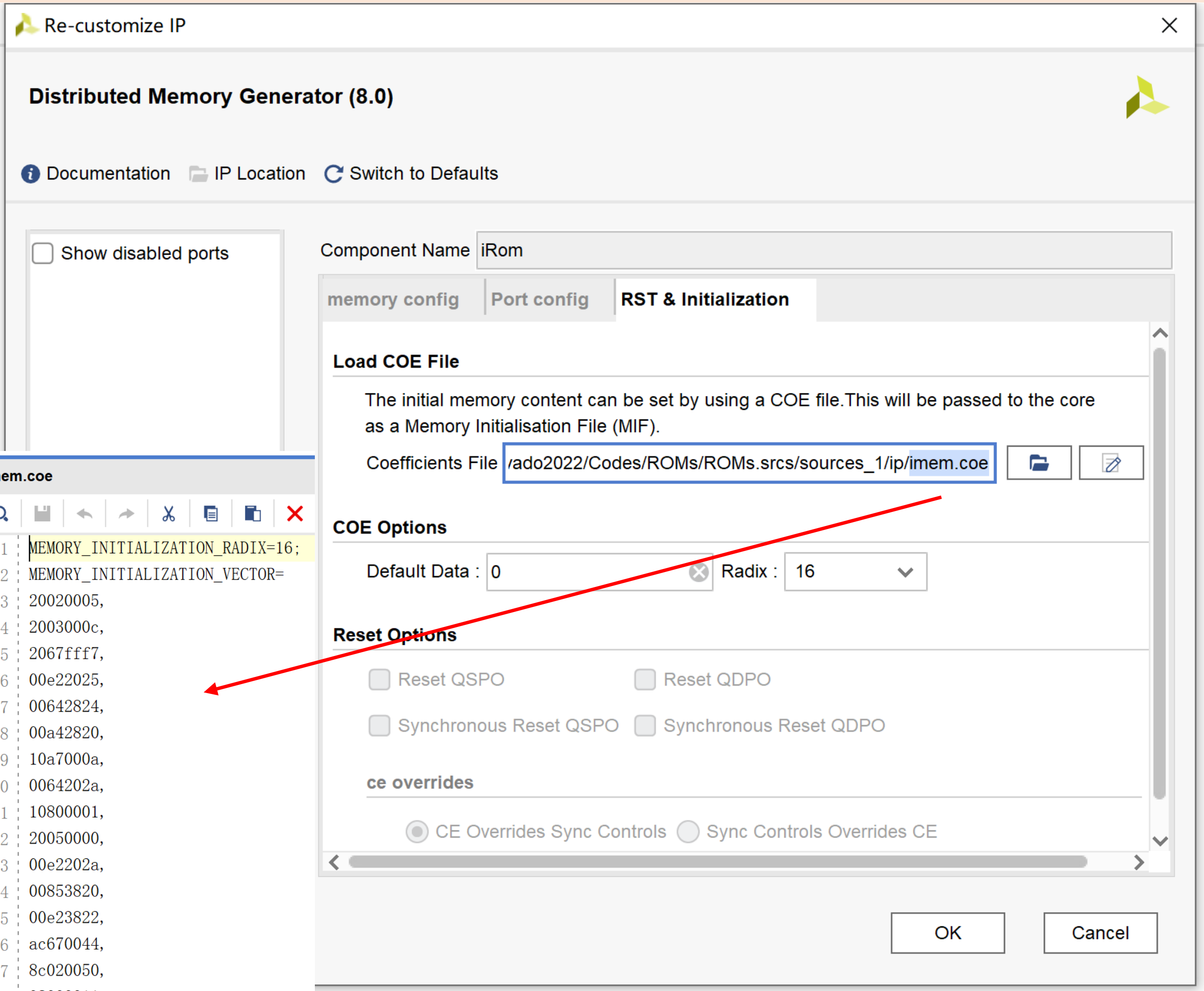
OK | Cancel

配置输入/输出引脚

是否具有锁存功能


imem.coe

```
MEMORY_INITIALIZATION_RADIX=16;
MEMORY_INITIALIZATION_VECTOR=
20020005,
2003000c,
2067fff7,
00e22025,
00642824,
00a42820,
10a7000a,
0064202a,
10800001,
20050000,
00e2202a,
00853820,
00e23822,
ac670044,
8c020050,
08000011,
20020001,
ac020054;
```


Re-customize IP

Distributed Memory Generator (8.0)

Documentation | IP Location | Switch to Defaults

☐ Show disabled ports

Component Name: iRom

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: /ado2022/Codes/ROMs/ROMs.srcs/sources_1/ip/imem.coe

COE Options

Default Data : 0 Radix : 16

Reset Options

☐ Reset QSPO ☐ Reset QDPO

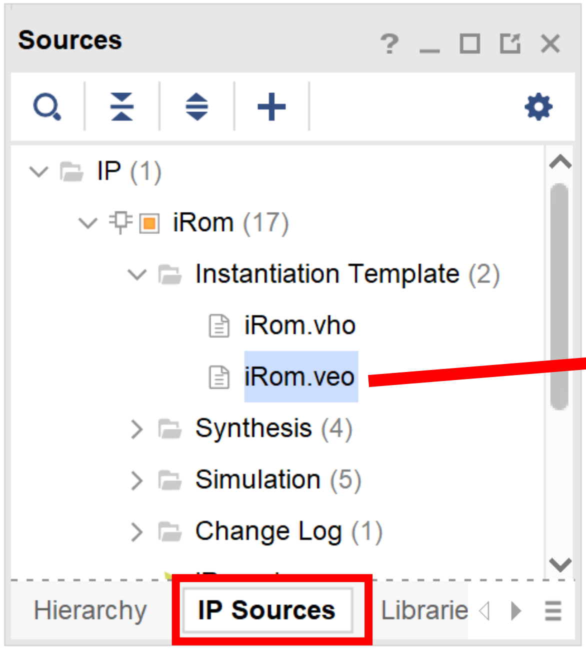
☐ Synchronous Reset QSPO ☐ Synchronous Reset QDPO

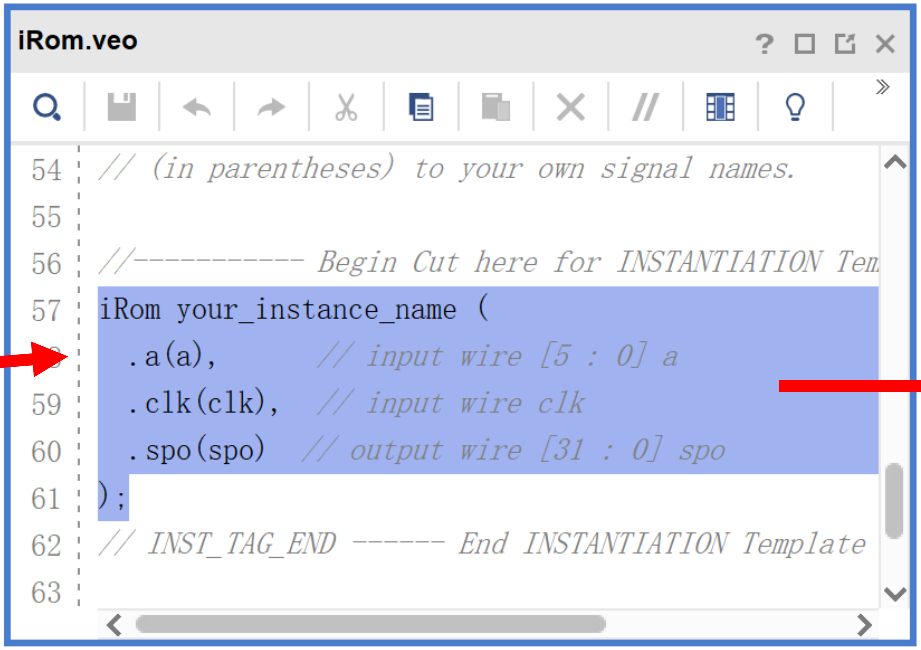
ce overrides

[✓ CE Overrides Sync Controls] Sync Controls Overrides CE

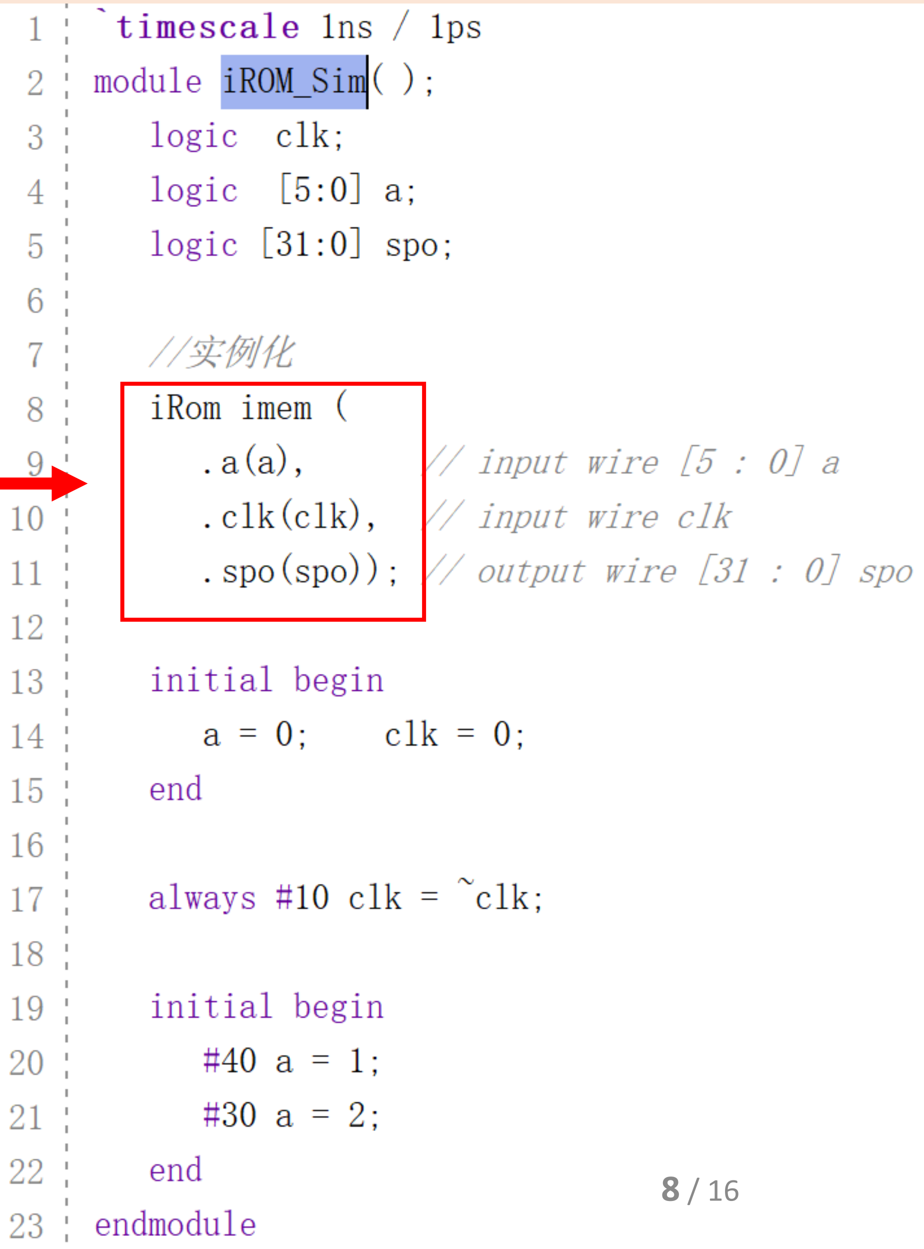
OK | Cancel

仿真

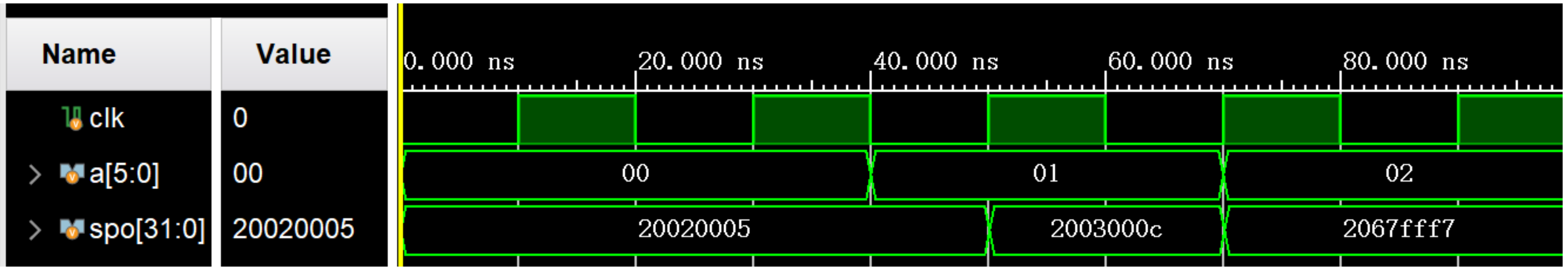



iRom.veo

```
// (in parentheses) to your own signal names.

//----------- Begin Cut here for INSTANTIATION Tem
iRom your_instance_name (
  .a(a),      // input wire [5 : 0] a
  .clk(clk),  // input wire clk
  .spo(spo)  // output wire [31 : 0] spo
);
// INST_TAG_END ------ End INSTANTIATION Template
```



```
`timescale 1ns / 1ps
module iROM_Sim( );
   logic  clk;
   logic  [5:0] a;
   logic [31:0] spo;

   //实例化
   iRom imem (
      .a(a),       // input wire [5 : 0] a
      .clk(clk),   // input wire clk
      .spo(spo));  // output wire [31 : 0] spo

   initial begin
      a = 0;   clk = 0;
   end

   always #10 clk = ~clk;

   initial begin
      #40 a = 1;
      #30 a = 2;
   end
endmodule
```



Nexys4 上的存储空间

- Artix-7 芯片有两种存储空间 (embedded memory)

  ① **Distributed RAM** 1,188Kb

  由多个逻辑单元中的LUTs构成，故容量小

  ② **Block RAM** 4,860Kb

- The Nexys4 DDR board contains two external memories:

  ① *1Gib* (128MiB) DDR2 **SDRAM**,

  ② *128Mib* (16MiB) non-volatile(非易失) serial **Flash** device.

  - **以10为底的指数**：1KB=10^3=1000，1MB=10^6=1000KB
  - **以 2为底的指数**：1KiB=2^10=1024, 1MiB=2^20=1024KiB

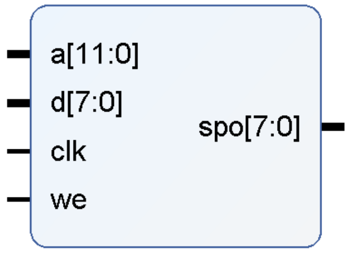
2

RAM

RAMs

单口 RAM：1套地址总线，1套数据总线，读、写分开

```
module singlePort_RAM_asynRead
    #( parameter ADDR_WIDTH = 12, DATA_WIDTH = 8 )
     (input  logic clk,
      input  logic we,       // Write enable
      input  logic [ADDR_WIDTH-1 : 0] addr,
      input  logic [DATA_WIDTH-1 : 0] din,
      output logic [DATA_WIDTH-1 : 0] dout );

    //        位宽                    字宽
    logic [DATA_WIDTH-1 : 0] RAM [2**ADDR_WIDTH-1 : 0];

    always_ff @(posedge clk)
        if (we) RAM[addr] <= din;   // write operation

    assign      dout = RAM[addr];   // read operation 异步
endmodule
```



```
module singlePort_RAM_synRead
    #( parameter ADDR_WIDTH = 12, DATA_WIDTH = 8 )
     (input  logic clk,
      input  logic we,       // Write enable
      input  logic [ADDR_WIDTH-1 : 0] addr,
      input  logic [DATA_WIDTH-1 : 0] din,
      output logic [DATA_WIDTH-1 : 0] dout );

    //        位宽                    字宽
    logic [DATA_WIDTH-1: 0] RAM [2**ADDR_WIDTH-1 : 0];
    logic [ADDR_WIDTH-1: 0] addr_reg ;

    always_ff @(posedge clk)
    begin   //=== 先写后读 ===
        if (we) RAM[addr] <= din;   // write operation
        addr_reg <= addr;           //同步地址
    end

    assign   dout = RAM[addr_reg]; // read operation 同步
endmodule
```

Due to the internal structure, an asynchronous read operation can be realized **only** by the distributed RAM.

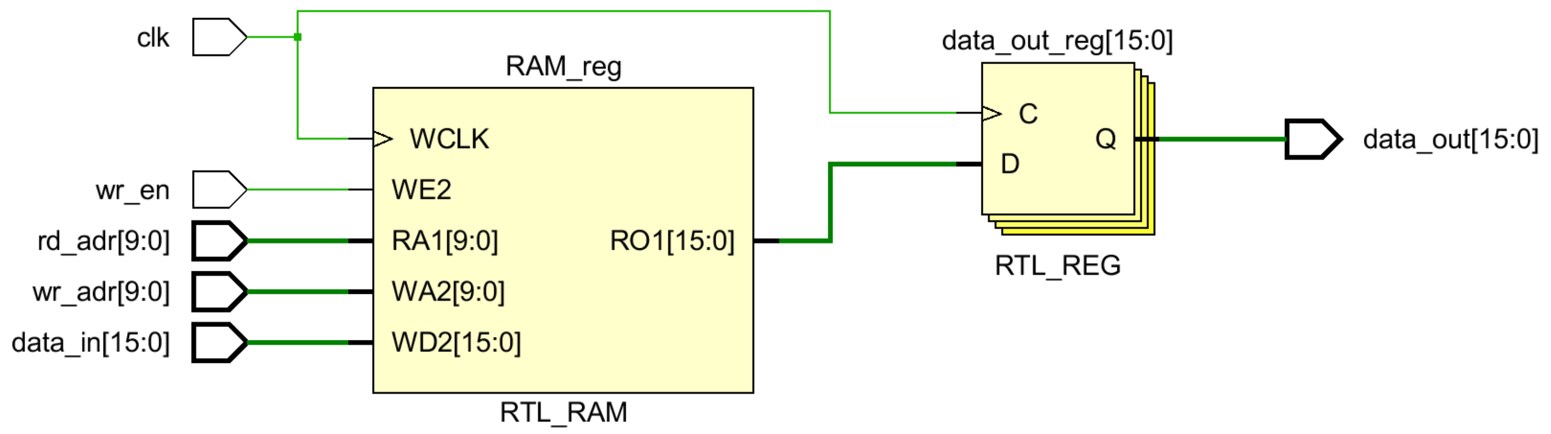
Simple Dual-port RAM：2套地址总线，1套数据总线

```
 1  // read 和 write是并行的
 2  module SimpleDualPortRAM1(
 3      input logic        clk, wr_en,
 4      input logic  [ 9:0] rd_adr,
 5      input logic  [ 9:0] wr_adr,
 6      input logic  [15:0] data_in,
 7      output logic [15:0] data_out );
 8
 9      //      位宽         字宽
10      logic [15:0] RAM [1023:0];
11
12      always_ff @( posedge clk )
13      begin
14          data_out <= RAM[rd_adr];            //Read  RAM
15          if (wr_en)  RAM[wr_adr] <= data_in; //Write RAM
16      end
17  endmodule
```

一读、一写

读、写并行：先保存原值，再写入

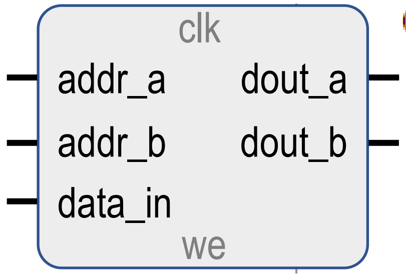
RAM中**原有的值**先存入**data_out_reg**中输出，然后 data_in 再保存到RAM中。



并行：14、15行可以颠倒。

Dual-port RAM：2套地址总线，1.5套数据总线

```
module DualPort_RAM_asynRead
    #(parameter ADDR_WIDTH = 6, DATA_WIDTH = 8 )
    ( input  logic clk,
      input  logic we,   // Write enable
      input  logic [ADDR_WIDTH-1: 0] addr_a, addr_b ,
      input  logic [DATA_WIDTH-1: 0] din_a,
      output logic [DATA_WIDTH-1: 0] dout_a, dout_b );

    //          位宽                    字宽
    logic [DATA_WIDTH-1: 0] RAM [2**ADDR_WIDTH-1: 0];

    always_ff @(posedge clk)
        if (we) RAM[addr_a] <= din_a; // write operation

    assign dout_a = RAM[addr_a];    // read operations 1
    assign dout_b = RAM[addr_b];    // read operations 2
endmodule
```



```
module DualPort_RAM_synRead
    #(parameter ADDR_WIDTH = 6, DATA_WIDTH = 8 )
    ( input  logic clk,
      input  logic we,    // Write enable
      input  logic [ADDR_WIDTH-1: 0] addr_a, addr_b,
      input  logic [DATA_WIDTH-1: 0] din_a,
      output logic [DATA_WIDTH-1: 0] dout_a, dout_b);

    //          位宽                    字宽
    logic [DATA_WIDTH-1: 0] RAM [2**ADDR_WIDTH-1: 0];
    logic [ADDR_WIDTH-1: 0] addr_a_reg, addr_b_reg;

    always_ff @(posedge clk)
    begin
        if (we) RAM[addr_a] <= din_a; // write operation
        addr_a_reg <= addr_a;
        addr_b_reg <= addr_b;
    end

    assign dout_a = RAM[addr_a_reg];  // read operations 1
    assign dout_b = RAM[addr_b_reg];  // read operations 2
endmodule
```

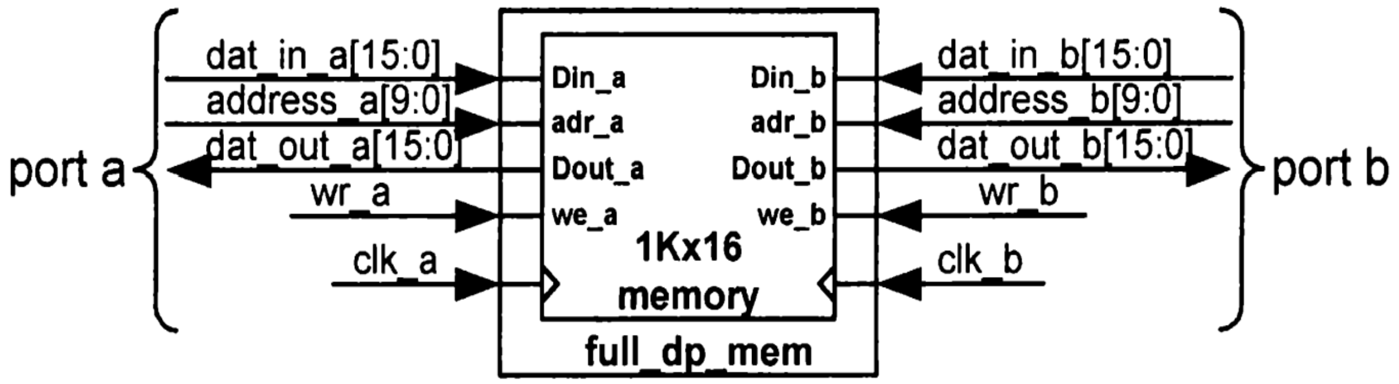
两个口都可以读写，但不能对同一个地址同时写。

Can be realized only by distributed RAM, thus its size is limited.

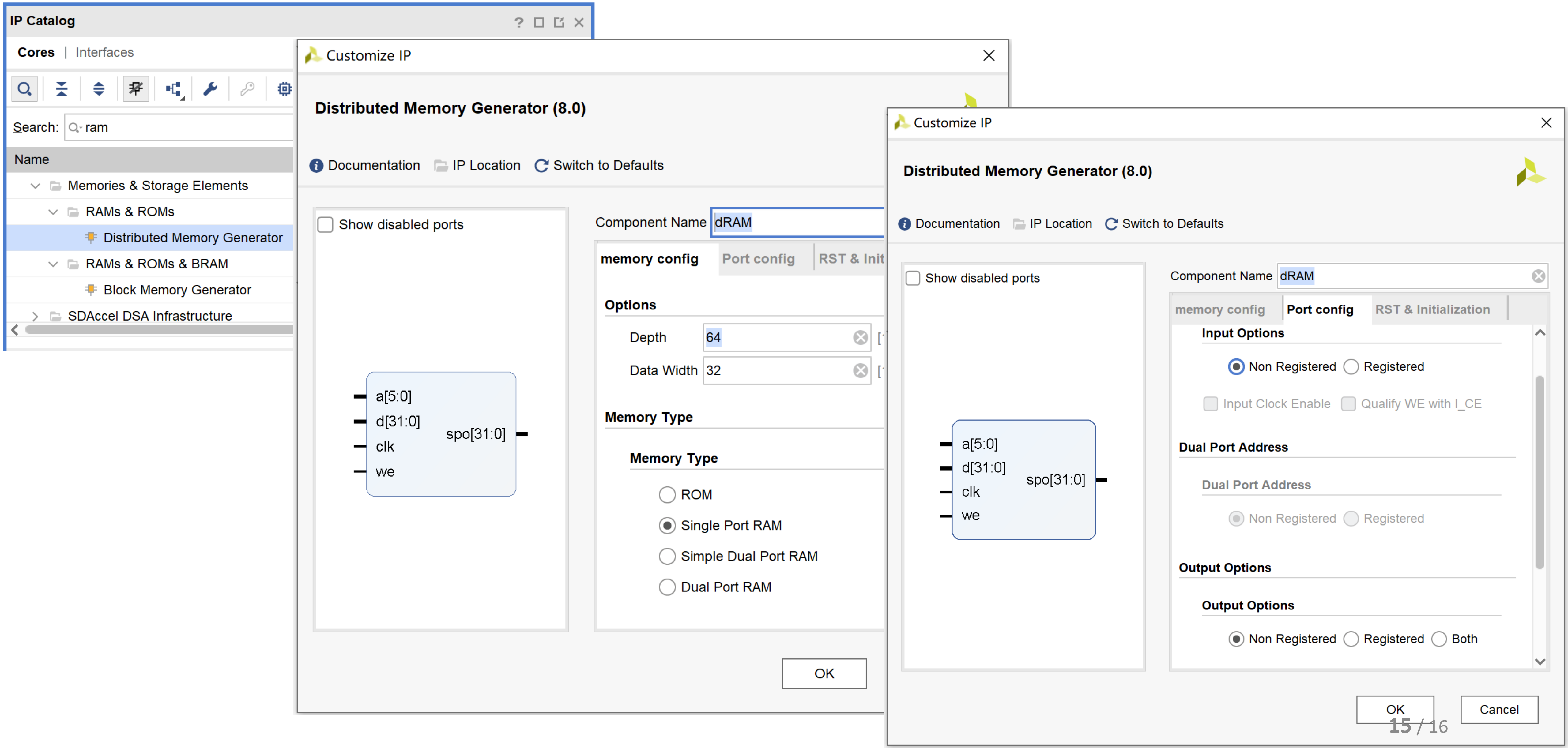
Full Dual-port RAM：地址线、数据线、时钟都2套

```
module FullDualPortMemory(
    input logic         clk_a,  clk_b,
    input logic         wr_a,   wr_b,
    input logic [ 9:0]  address_a,  address_b,
    input logic [15:0]  data_in_a,  data_in_b,
    output logic[15:0]  dat_out_a,  dat_out_b );

    logic [15:0] RAM [1023:0];

    // Port a
    always_ff @( posedge clk_a )
    begin
        dat_out_a           <= RAM[address_a];
        if (wr_a)
        begin
            dat_out_a       <= data_in_a;
            RAM[address_a]  <= data_in_a;
        end
    end

    // Port b
    always_ff @( posedge clk_b )
    begin
        dat_out_b           <= RAM[address_b];
        if (wr_b)
        begin
            dat_out_b       <= data_in_b;
            RAM[address_b]  <= data_in_b;
        end
    end
endmodule
```

先输出
再写入



IP核设计RAM



IP Catalog

Cores | Interfaces

Search: ram

Name
- Memories & Storage Elements
  - RAMs & ROMs
    - Distributed Memory Generator
  - RAMs & ROMs & BRAM
    - Block Memory Generator
- SDAccel DSA Infrastructure

Customize IP

Distributed Memory Generator (8.0)

Documentation | IP Location | Switch to Defaults

Show disabled ports

Component Name: dRAM

memory config | Port config | RST & Init

Options

Depth: 64

Data Width: 32

Memory Type

Memory Type
- ROM
- [✓ Single Port RAM]
- Simple Dual Port RAM
- Dual Port RAM

OK

Customize IP

Distributed Memory Generator (8.0)

Documentation | IP Location | Switch to Defaults

Show disabled ports

Component Name: dRAM

memory config | Port config | RST & Initialization

Input Options

[✓ Non Registered] Registered

Input Clock Enable | Qualify WE with I_CE

Dual Port Address

Dual Port Address

Non Registered | Registered

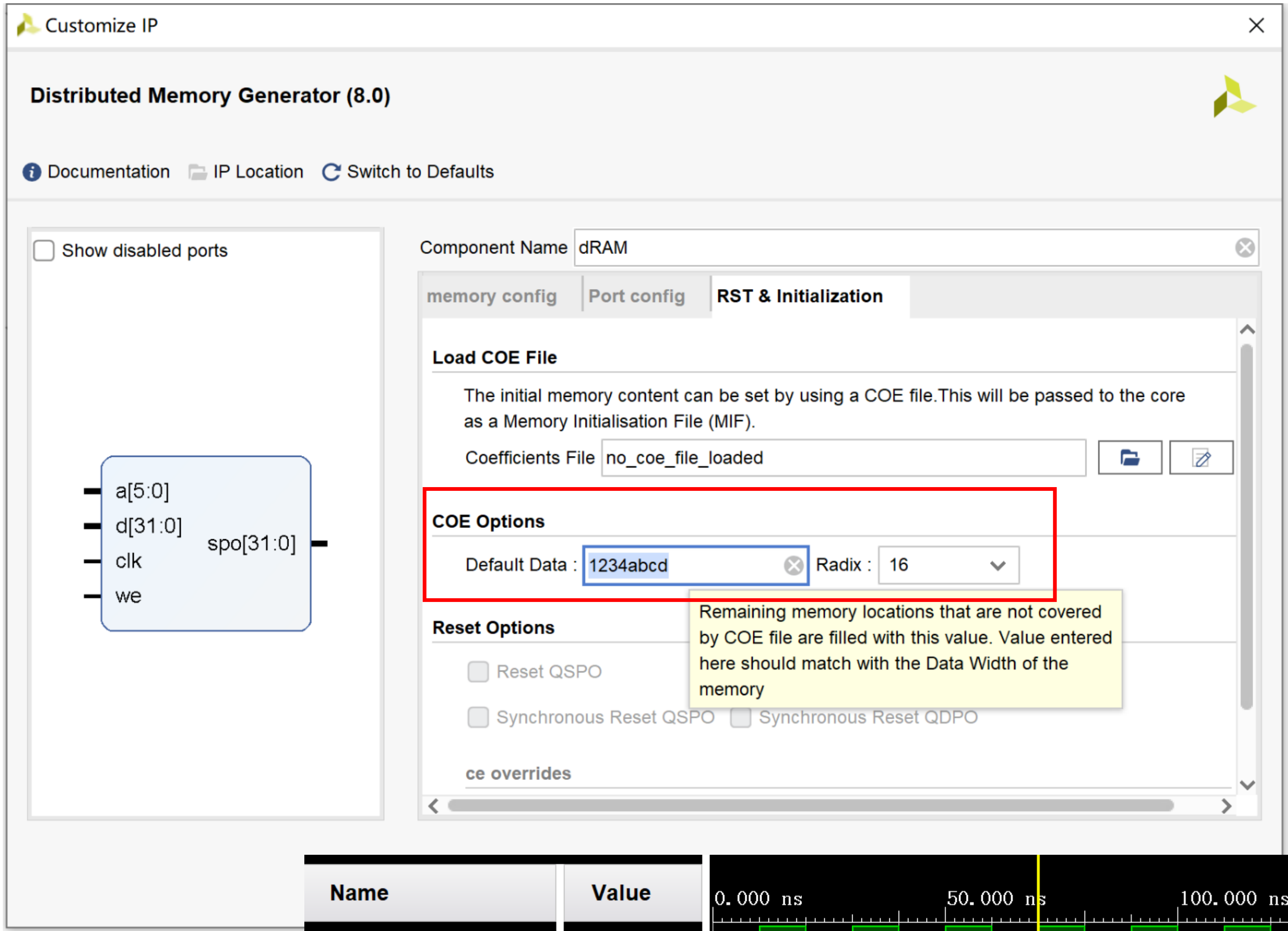
Output Options

Output Options

[✓ Non Registered] Registered Both

OK | Cancel

仿真



Customize IP

Distributed Memory Generator (8.0)

Documentation IP Location Switch to Defaults

Show disabled ports

Component Name dRAM

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 no_coe_file_loaded

COE Options

Default Data : 1234abcd Radix : 16

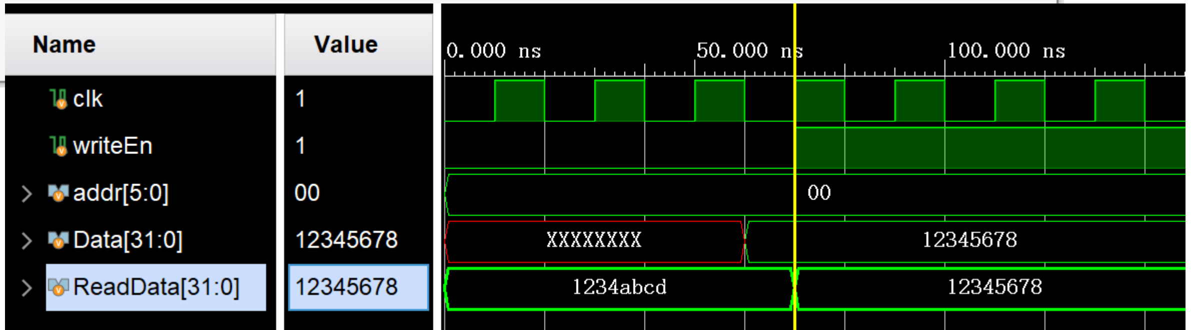
Remaining memory locations that are not covered by COE file are filled with this value. Value entered here should match with the Data Width of the memory

Reset Options

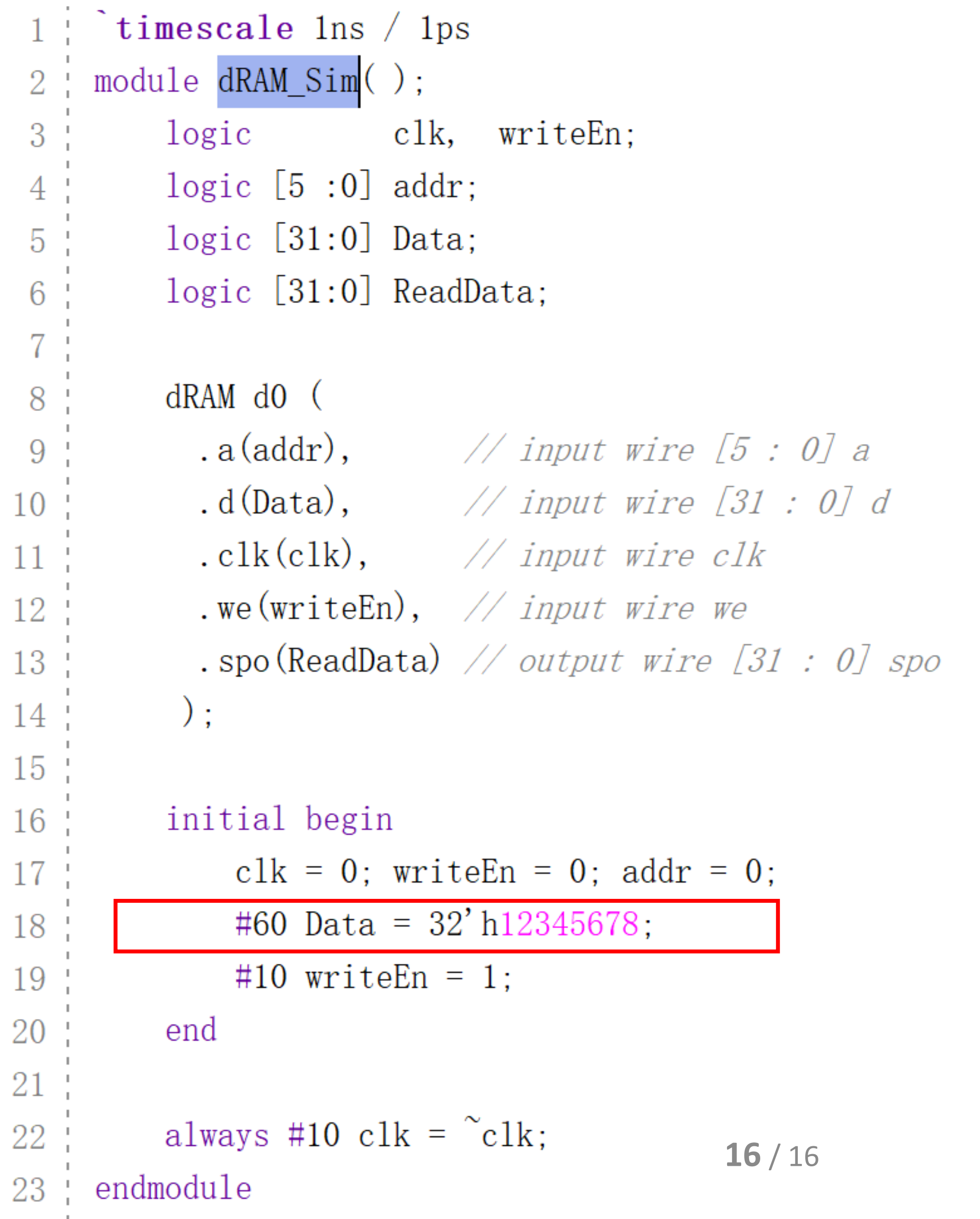
Reset QSPO

Synchronous Reset QSPO Synchronous Reset QDPO

ce overrides



Name	Value
clk	1
writeEn	1
addr[5:0]	00
Data[31:0]	12345678
ReadData[31:0]	12345678



```
`timescale 1ns / 1ps
module dRAM_Sim( );
    logic        clk,  writeEn;
    logic [5 :0] addr;
    logic [31:0] Data;
    logic [31:0] ReadData;

    dRAM d0 (
      .a(addr),       // input wire [5 : 0] a
      .d(Data),       // input wire [31 : 0] d
      .clk(clk),      // input wire clk
      .we(writeEn),   // input wire we
      .spo(ReadData)  // output wire [31 : 0] spo
    );

    initial begin
        clk = 0; writeEn = 0; addr = 0;
        #60 Data = 32'h12345678;
        #10 writeEn = 1;
    end

    always #10 clk = ~clk;
endmodule
```