

1. Multiplier 1 && Multiplier 2

只要RTL Schematic即可

```
module mul_1(  
  input [3:0] a,b,c,d,  
  output reg [15:0] out  
);  
  reg [7:0] temp_data_1;  
  reg [11:0] temp_data_2;  
  always@(*) begin  
    temp_data_1 = a*b;  
    temp_data_2 = temp_data_1*c;  
    out = temp_data_2*d;  
  end  
  
endmodule
```

```
module mul_2(  
  input [3:0] a,b,c,d,  
  output reg [15:0] out  
);  
  reg [7:0] temp_data_1;  
  reg [7:0] temp_data_2;  
  always@(*) begin  
    temp_data_1 = a*b;  
    temp_data_2 = c*d;  
    out = temp_data_1*temp_data_2;  
  end  
  
endmodule
```

2. RAM Case (single-port & ip)

1 打开IP核的目录列表

PROJECT MANAGER

Settings

Add Sources

Language Templates

IP Catalog

2 搜索bram,
找到
Block Memory Generator

Project Summary x IP Catalog x

Cores | Interfaces

Search: Q- bram (3 matches)

Name	AXI4	Status	License	VLNV
Vivado Repository				
Embedded Processing				
Memory and Memory Controller				
AXI BRAM Controller	AXI4	Production	Included	xilinx.com:ip:axi_bram_ctrl:4.1
LMB BRAM Controller	AXI4	Production	Included	xilinx.com:ip:lmb_bram_if_cntlr:4.0
Memories & Storage Elements				
RAMs & ROMs & BRAM				
Block Memory Generator	AXI4	Production	Included	xilinx.com:ip:blk_mem_gen:8.4

2. RAM Case (single-port & ip)

3 双击Block Memory Generator，打开下面Basic界面进行配置

Block Memory Generator (8.4)

Documentation IP Location Switch to Defaults

IP Symbol Power Estimation

Show disabled ports

Component Name **bram_sp**

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) 9

Algorithm Options

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

Algorithm Minimum Area

Primitive 8kx2

对IP核命名为bram_sp

选择单口RAM

单口RAM只有一个端口（A端口）：
可以对A端口进行读写。

简化双口RAM有两个端口（A和B端口）：
但是A端口只能进行写入操作，不能进行读出操作，而B端口则只能进行读出操作，不能进行写入操作。

真双口RAM有两个端口（A和B端口）：
A和B端口都能进行读写操作。

2. RAM Case (single-port & ip)

4 切换配置页，打开下面Port A Options界面进行配置，配置完点击OK 5 点击Generate，完成配置

Block Memory Generator (8.4)

Documentation IP Location Switch to Defaults

IP Symbol Power Estimation

☐ Show disabled ports

Component Name

Basic **Port A Options** Other Options Summary

Memory Size

Write Width Range: 1 to 4608 (bits) 配置位宽为8

Read Width

Write Depth Range: 2 to 1048576 深度为16

Read Depth

Operating Mode Enable Port Type

Port A Optional Output Registers 去掉勾选

☐ Primitives Output Register ☐ Core Output Register

☐ Soft ECC Input Register ☐ REGCEA Pin

Port A Output Reset Options

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

☐ Reset Memory Latch Reset Priority

READ Address Change A

☐ Read Address Change A

+ BRAM_PORTA

Generate Output Products

The following output products will be generated.

Preview

☐ ☒ bram_sp.xci (OOC per IP)

- ☐ Instantiation Template
- ☐ Synthesized Checkpoint (.dcp)
- ☐ Structural Simulation
- ☐ Channel Log

Synthesis Options

☐ Global

☒ Out of context per IP

Run Settings

Number of jobs:

2. RAM Case (single-port & ip)

6 回到**Sources**栏，展开**bram_sp**，点击vhd文件，查看端口信息

The screenshot displays the Vivado IDE interface. On the left, the **Sources** window shows the project hierarchy. The **Design Sources (11)** folder is expanded, revealing **bram_sp (bram_sp.xci) (1)**. This folder is further expanded to show **bram_sp(bram_sp_arch) (bram_sp.vhd) (1)**, which is highlighted with a red box. A red arrow points to this file with the text "双击这个文件". Below the Sources window, the **Source File Properties** window for **bram_sp.vhd** is visible, showing it is **Enabled** and located at `e:/VivadoProject/Learning_Basic/ASIC_Class5/ASIC_Class5.srscs/sources_1/ip/bram_sp/synth/bram_sp.vhd`.

On the right, the **Project Summary** window shows the content of **bram_sp.vhd**. The file path is `e:/VivadoProject/Learning_Basic/ASIC_Class5/ASIC_Class5.srscs/sources_1/ip/bram_sp/synth/bram_sp.vhd`. The code defines the ports for the RAM block:

```
60 PORT (  
61     clka : IN STD_LOGIC;  
62     ena  : IN STD_LOGIC;  
63     wea  : IN STD_LOGIC_VECTOR(0 DOWNTO 0);  
64     addra : IN STD_LOGIC_VECTOR(3 DOWNTO 0);  
65     dina  : IN STD_LOGIC_VECTOR(7 DOWNTO 0);  
66     douta : OUT STD_LOGIC_VECTOR(7 DOWNTO 0);  
67 );
```

A red box highlights the port declarations from line 61 to 66, with a red arrow pointing to it from the text "相关的端口信息".

2. RAM Case (single-port & ip)

7 新建bram_ipcore.v文件，例化bram_sp ip

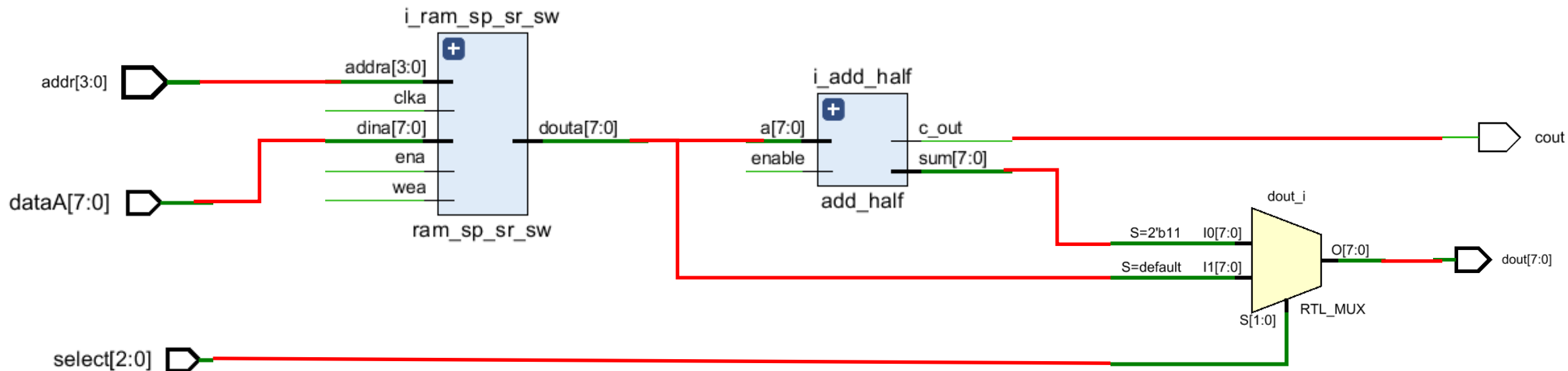
```
module bram_ipcore#(  
    parameter DATA_WIDTH = 8,  
    parameter ADDR_WIDTH = 4,  
    parameter RAM_DEPTH = (1 << ADDR_WIDTH)  
)(  
    //-----Input Ports-----  
    input wire          clka,    //Clock Input  
    input wire [ADDR_WIDTH-1:0] addra, //Address Input  
    input wire [DATA_WIDTH-1:0] dina, //Data Input  
    input wire          ena,    //Chip Select  
    input wire          wea,    //Write Enable / Read Enable  
    //-----Output Ports-----  
    output wire [DATA_WIDTH-1:0] douta  
);  
  
    bram_sp i_bram_sp(  
          
    );  
endmodule
```

根据第6步查看的端口信息，完成例化

8 新建bram_ipcore_tb.v文件，
将ram_sp_sr_sw_tb.v中的例化模块对应修改就行，其他保持不变

3. RAM Case2 (ram & adder & mux)

1 根据下图，完成相应代码部分



i_ram_sp是RAM case完成的，可以直接使用
add_half.v是之前作业完成过的，这里直接给出，见文件资料

3. RAM Case2 (ram & adder & mux)

```
`define READ    2'b01
`define WRITE   2'b10
`define READ_ADD 2'b11
module simple_project#(
    parameter DATA_WIDTH = 8,
    parameter ADDR_WIDTH = 4,
    parameter RAM_DEPTH = (1 << (ADDR_WIDTH))
)(
    input                clk,
    input                rst_n,
    input  [1:0]         select,
    input  [DATA_WIDTH-1:0] dataA,
    input  [ADDR_WIDTH-1:0] addr,

    output [DATA_WIDTH-1:0] dout,
    output cout

);

//Inner signals
wire  [DATA_WIDTH-1:0] dout_buf_ram;
wire  [DATA_WIDTH-1:0] dout_buf_add;
reg    ena;
reg    wea;
reg    enable;
```

```
//MUX
//Interconnections
ram_sp_sr_sw #(DATA_WIDTH, ADDR_WIDTH, RAM_DEPTH)
i_ram_sp_sr_sw(
    .clka      (clk ),
    .addra     (    ),
    .dina      (    ),
    .ena       (ena  ),
    .wea       (wea  ),
    .douta     (    )
);

add_half #(DATA_WIDTH)
i_add_half(
    .a         (    ),
    .enable    (enable),
    .sum       (    ),
    .c_out     ([    ])
);
```


3. RAM Case2 (ram & adder & mux)

```
68 //control
69 always @(posedge clk or negedge rst_n) begin
70     if (!rst_n) begin
71         ena         <= 1'b0;
72         wea         <= 1'b0;
73         enable      <= 1'b0;
74     end
75     else begin
76         case (select)
77             `READ: begin
78                 ena         <= 1'b1;
79                 wea         <= 1'b0;
80                 enable      <= 1'b0;
81             end
82             `WRITE:begin
83                 ena         <= 1'b1;
84                 wea         <= 1'b1;
85                 enable      <= 1'b0;
86             end
77
```

```
87     `READ_ADD:begin
88         ena         <= 1'b1;
89         wea         <= 1'b0;
90         enable      <= 1'b1;
91     end
92     default: begin
93         ena         <= 1'b0;
94         wea         <= 1'b0;
95         enable      <= 1'b0;
96     end
97     endcase
98 end
99 end
100 endmodule
```

3. RAM Case2 (ram & adder & mux)

2

tb_simple_project.v跟之前的testbench文件基本一致，这里直接给出，见文件资料
修改下文件路径即可

```
67  initial begin
68      #20;
69      $readmemb("E:/InputData.txt",input_data);
70  end
71
72
73  integer fp_rd_data;
74  initial begin
75      fp_rd_data = $fopen("E:/OutputData.txt","w");
76  end
77
```

OutputData_Groundtruth.txt

```
00010001
0
00001010
0
00000111
0
00000000
1
11111111
0
,
```

要求最后输出的OutputData.txt文件内容跟OutputData_groundtruth.txt内容一致