

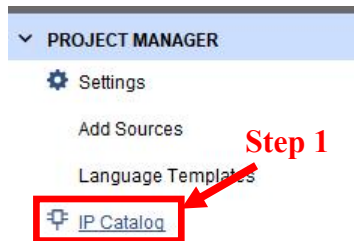
# Memory in xilinx artix7 FPGA

## RAM vs ROM

- RAM: read and write
  - ROM: read only
- 
- IP core - memory related
    - Block(used more often) vs Distribute
- 
- IO ports
    - clk, enable, read/write
    - BUS
      - address(determine which “word” in Memory)
      - data ( read/write the “word” from/to Memory)
- 
- Tips
    - specify the type of “device” in vivado project before using IP cores.

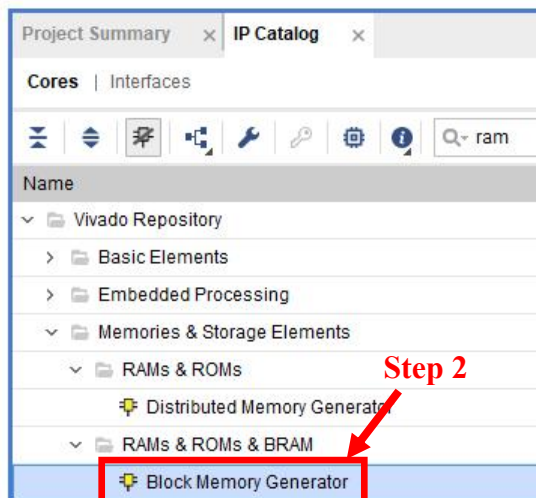
# Using IP core: Block Memory(1)

Using the IP core 'Block Memory' of Xilinx to implement the Data-memory.



Import the IP core in vivado project

1) in “PROJECT MANAGER” window  
click “IP Catalog”



2) in “IP Catalog” window

> Vivado Repository

> Memories & Storage Elements

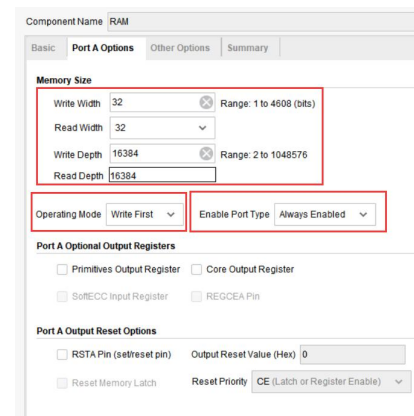
> RAMs & ROMs & BRAM

> **Block Memory Generator**

# Using IP core: Block Memory(2)

## Step3: Customize the IP core

- set **name**(component name),  
**type**(**RAM/ROM**)
- set features of the ROM(**width** and **depth**), **operation mode** and **register output**
- set **initial file**



Component Name: RAM

Basic | Port A Options | Other Options | Summary

Memory Size

Write Width	32	Range: 1 to 4096 (bits)
Read Width	32	
Write Depth	16384	Range: 2 to 1048576
Read Depth	16384	

Operating Mode: Write First | Enable Port Type: Always Enabled

Port A Optional Output Registers

☐ Primitives Output Register ☐ Core Output Register

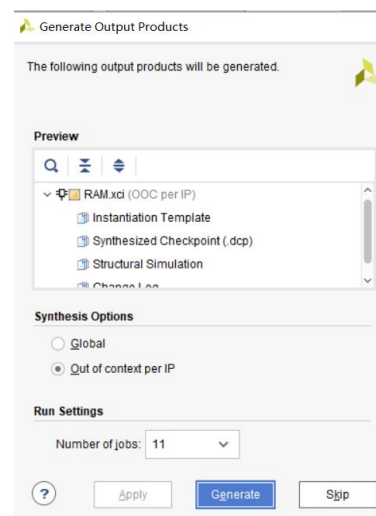
☐ SoftECC Input Register ☐ REGCEA Pin

Port A Output Reset Options

☐ RSTA Pin (setreset pin) ☐ Output Reset Value (Hex) 0

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

## Step4: Generate the IP core, then it will be added to vivado project automatically



Generate Output Products

The following output products will be generated.

Preview

- RAM.xci (OOC per IP)
  - Instantiation Template
  - Synthesized Checkpoint (.dcp)
  - Structural Simulation
  - Change Log

Synthesis Options

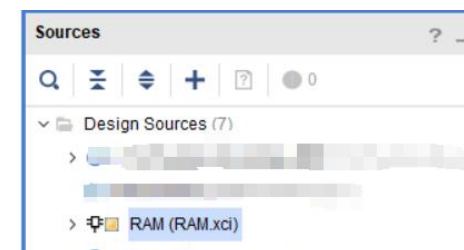
☐ Global

☒ Out of context per IP

Run Settings

Number of jobs: 11

Apply Generate Skip

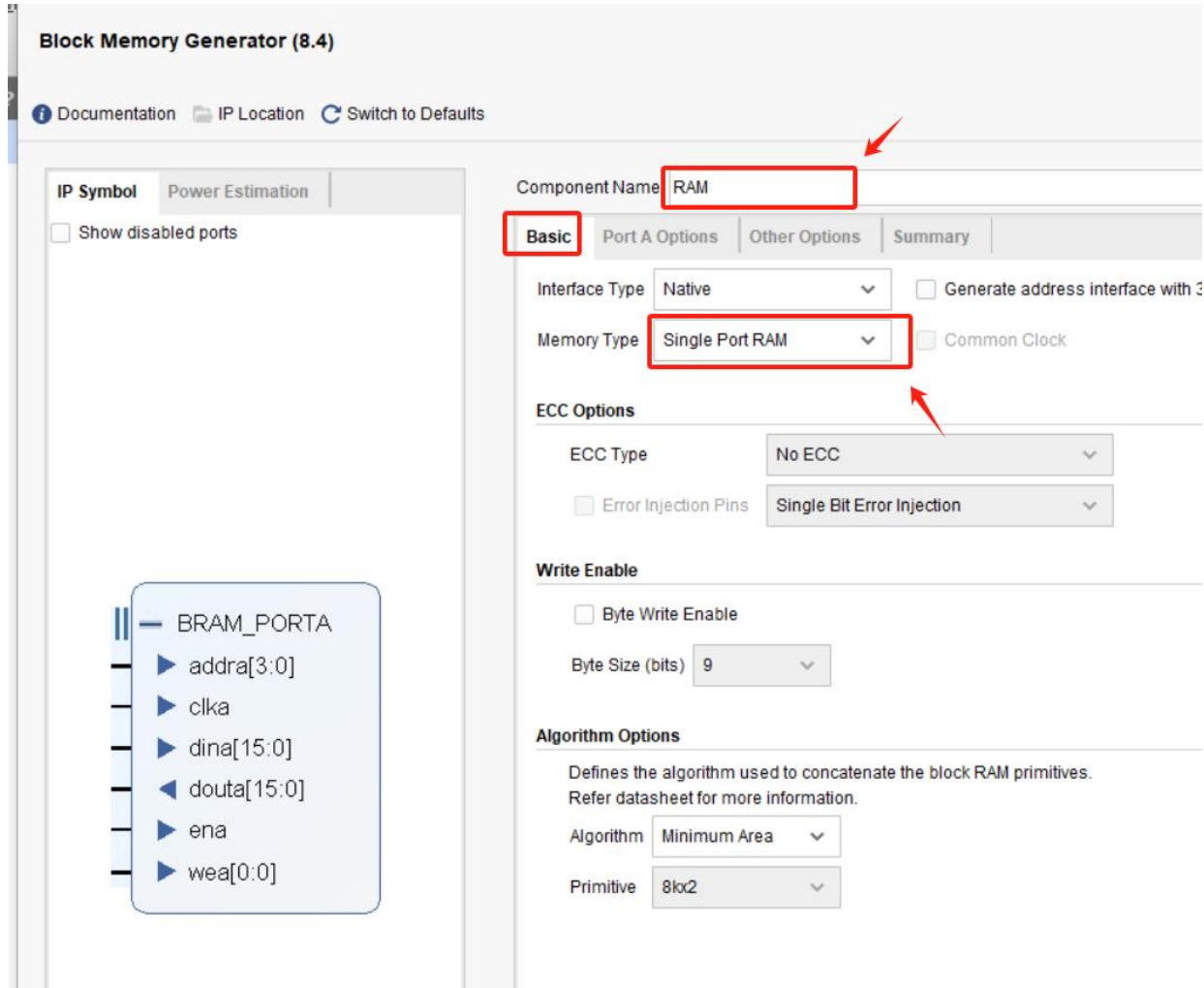


Sources

Design Sources (7)

- RAM (RAM.xci)

# Customize RAM IP core (1)



**Block Memory Generator (8.4)**

Documentation IP Location Switch to Defaults

IP Symbol Power Estimation

☐ Show disabled ports

Component Name: RAM

**Basic** Port A Options Other Options Summary

Interface Type: Native ☐ Generate address interface with 3

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

BRAM\_PORTA

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

In **Basic** configuration page:

1) Specify the **"Memory Type"** as **"Single Port RAM"**

2) Specify the **"component name"**, here is **"RAM"**

**NOTE:** the **component name** could be any string except the keyword in vivado and verilog

# Customize RAM IP core (2)

Component Name: RAM

Basic | **Port A Options** | Other Options | Summary

**Memory Size**

Write Width: 32 (Range: 1 to 4608 (bits))

Read Width: 32

Write Depth: 16384 (Range: 2 to 1048576)

Read Depth: 16384

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

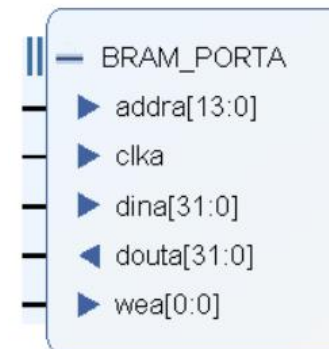
**Port A Output Reset Options**

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

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

## 3) PortA Options settings:

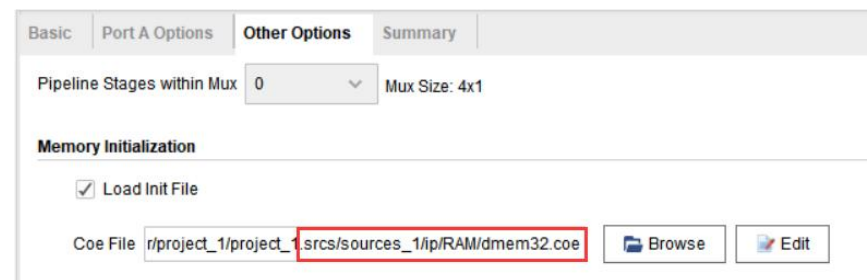
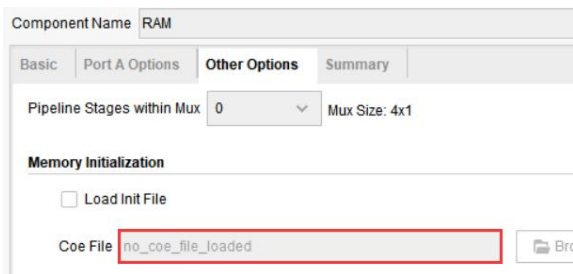
- Data read and write **bit width**:  
**32 bits (4Byte)**
- Write/Read **Depth**: **16384 (64KB)**
- **NOTE: Both width and depth configurations here are just a demo, need to be determined by designer**
- Operating Mode: **Write First**
- Enable Port Type: **Always Enabled**
- PortA Optional Output Registers: **NOT SET**



# Customize RAM IP core (3)

## 4) Other Options settings:

- 1. When **specifying the initialization file** for customize the RAM on the 1st time, the IP core RAM just customized **WITHOUT initial file** and **corresponding path**, so set it to **no initial file** when creating RAM.
- 2. **After** the RAM IP core created
  - 2-1. **COPY** the initialization file **dmem32.coe** to **projectName.srscs/sources\_1/ip/ComponentName**. (“projectName.srscs” is under the project folder, “componentName” here is ‘RAM’)
  - 2-2. Double-click the newly created RAM IP core, **RESET** it with the **initialization file**, select the **dmem32.coe** file that has been in the directory of **projectName.srscs/sources\_1/ip/RAM**.



NOTE: “**dmem32.coe**” here is just a demo, the coe file need to be generated by the designer

# Design Module With Memory IP Instanced

After the generation of the IP core, it would be added as a module to the **Design sources** of the current project.

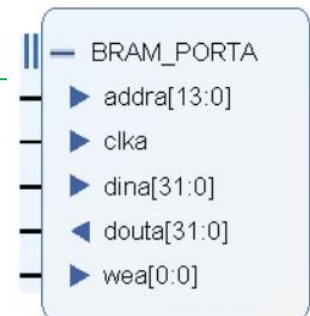
Instance the IP core as the other modules in verilog, bind its ports.

A demo is given on the following picture:



*//Create a instance of RAM(IP core), binding the ports*

```
RAM ram (  
    .clka(clk),                // input wire clka  
    .wea(memWrite),            // input wire [0 : 0] wea  
    .addra(address[15:2]),      // input wire [13 : 0] addra  
    .dina(writeData),          // input wire [31 : 0] dina  
    .douta(readData)           // output wire [31 : 0] douta  
);
```





# Function Verification by simulation

```
//The testbench module for dmemory32
module ramTb( );
reg clock = 1'b0;
reg memWrite = 1'b0;
reg [31:0] addr = 32'h0000_0010;
reg [31:0] writeData = 32'ha000_0000;
wire [31:0] readData;

dmemory32 uram
    (~clock,memWrite,addr,writeData,readData);
always #50 clock = ~clock;

initial fork
    #120    memWrite = 1'b1;
    #200
        writeData = 32'h0000_00f5;
    #400
        memWrite = 1'b0;
    // ... to be completed
join

endmodule
```

NOTE:

**Using bind port with name is Suggested!!**

1) Set “**memWrite**” to 1'b0 means to read the data from the RAM unit identified by “**addr**”.

2) Set “**memWrite**” to 1'b1 and “**writeData**” to 0x0000\_00f5 which means to write data 0xa000\_00f5 to the RAM unit identified by “**addr**”.

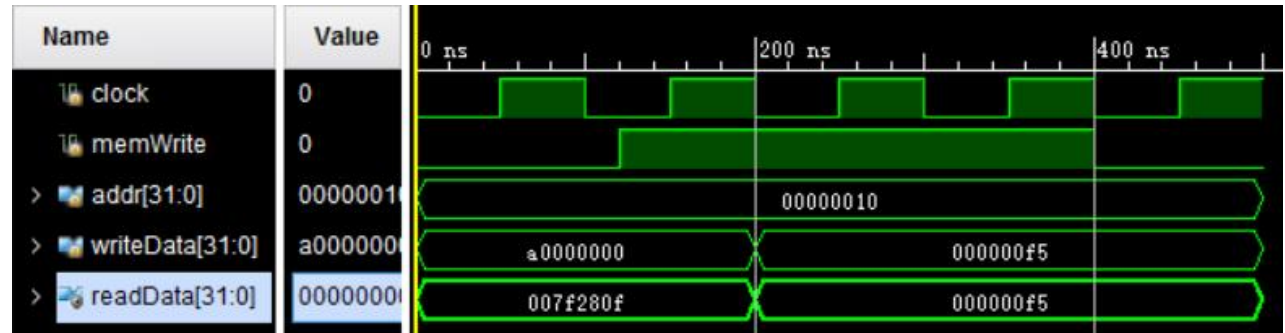


# Function Verification by simulation continued

```

1 memory_initialization_radix = 16;
2 memory_initialization_vector =
3 007f2812,
4 007f2811,
5 007f2810,
6 007f2810,
7 007f280f,
8 00000001,
9 00000002,
10 00000003,
11 00000005,
12 00000006,
13 00000007,
14 0000ffff,
15 00000000,
16 00000000,

```



Memory Size

Write Width 32 Range: 1 to 4608 (bits)

Read Width 32

**NOTE:** The bit width of each row of data in the COE file is consistent with the read and write bit width set in RAM settings.

e.g. the bitwidth of (007f2812) in hexadecimal is 32 which is same with “the read and write bit width” set in RAM settings

# Customize Memory IP core(ROM) (1)

Component Name: **prgrom**

**Basic** | Port A Options | Other Options | Summary

Interface Type: **Native** ☐ Generate address interface with 32 bits

Memory Type: **Single Port ROM** ☐ 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**

Component Name: **prgrom**

**Basic** | **Port A Options** | Other Options | Summary

**Memory Size**

Port A Width: **32** Range: 1 to 4608 (bits)

Port A Depth: **16384** Range: 2 to 1048576

The Width and Depth values are used for Read Operation in Port A

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

Component Name: **prgrom**

**Basic** | Port A Options | **Other Options** | Summary

Pipeline Stages within Mux: **0** Mux Size: 4x1

**Memory Initialization**

☐ Load Init File

Coe File: **no\_coe\_file\_loaded**

☐ Fill Remaining Memory Locations

Remaining Memory Locations (Hex): **0**

**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: **All**

**Behavioral Simulation Model Options**

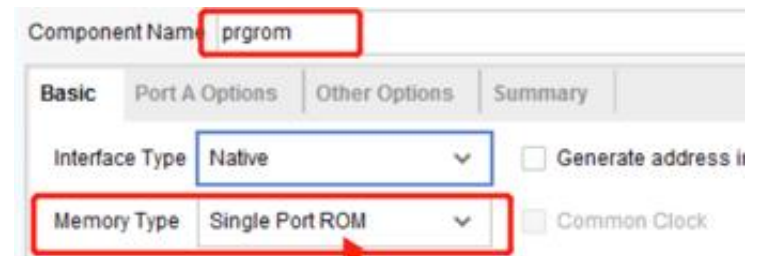
☐ Disable Collision Warnings ☐ Disable Out of Range Warnings

The steps about customization on block memory ROM is almost same as which of RAM.

# Instance the IP core (ROM)

// instance the IP core in verilog

```
prgrom instmem(  
    .clka(clock),  
    .addra(PC[15:2]),  
    .douta(Instruction)  
);
```



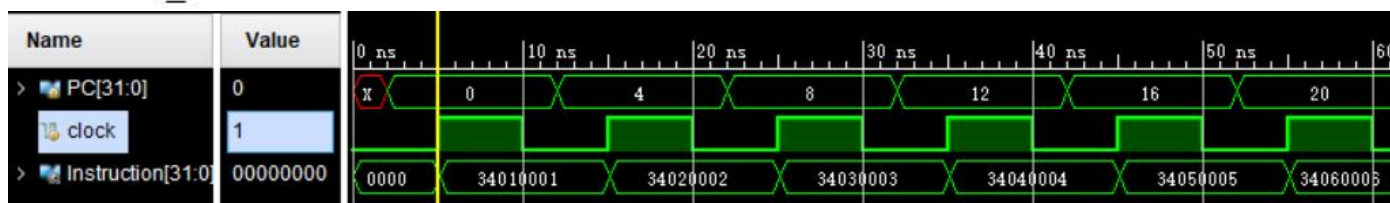


# The Function Verification of “prgrom”

NOTE: The bit width of each row of data in the COE file is consistent with the read and write bit width set in ROM settings.  
e.g. the bitwidth of (007f2812) in hexadecimal is 32 which is same with “the read and write bit width” set in ROM settings

prgmip32.coe

```
1 memory_initialization_radix = 16;  
2 memory_initialization_vector =  
3 34010001,  
4 34020002,  
5 34030003,  
6 34040004,  
7 34050005,  
8 34060006,  
9 34070007,  
10 34080008,  
11 34090009,  
12 340a000a,  
13 340b000b,  
14 340c000c,
```



```
module prgrom_tb( );    //a reference for the testbench ?  
    reg[31:0] PC;  
    reg clock=1'b0;  
    wire [31:0] Instruction;  
    prgrom instmem(.clka(clock),.addra(PC[15:2]),.douta(Instruction));  
    always #5 clock = ~clock;  
    initial begin  
        clock = 1'b0;  
        #2 PC = 32'h0000_0000;  
        repeat(5) begin  
            #10 PC = PC+4;  
            #10 $finish;  
        end  
    end  
endmodule
```

# TIPs

- Supplementary explanation on the addresses used for RAM and ROM instantiation in this courseware:
  - In standard 32-bit CPUs and computer architectures, memory is addressed in bytes, and to improve efficiency, the basic unit of CPU access to storage units is 4 bytes. Therefore, when accessing this storage module for read and write, the address used is a multiple of 4. (as the demo shown in this slides) (The above is the knowledge involved in the composition principles of the next semester)
  - In your project this semester (if needed), this standard can be temporarily ignored. You can design the basic unit bit width for storage units and the bit width for accessing data by yourself
- 补充说明：本课件中 关于 RAM 和 ROM 实例化时使用的地址的补充说明
  - 在标准的32位CPU以及计算机架构中，存储器是以字节为单位进行编址，而为提高效率，CPU访问存储单元的基本单位是4字节，因此针对这块存储模块进行读写访问时，使用的地址为4的倍数。（以上为下学期组成原理所涉及知识）本节使用的编址方式即为这种。
  - 在本学期同学们的project中，如果有需要使用存储类的IP核，可以暂时不参考这种标准，你可以自行设计存储单元的基本单元位宽和访问数据的位宽。