



杭州电子科技大学
HANGZHOU DIANZI UNIVERSITY

实验项目

A composite image showing a computer keyboard and mouse in a blue and green color scheme, overlaid with binary code (0s and 1s) and a faint grid pattern.

主讲教师：章复嘉

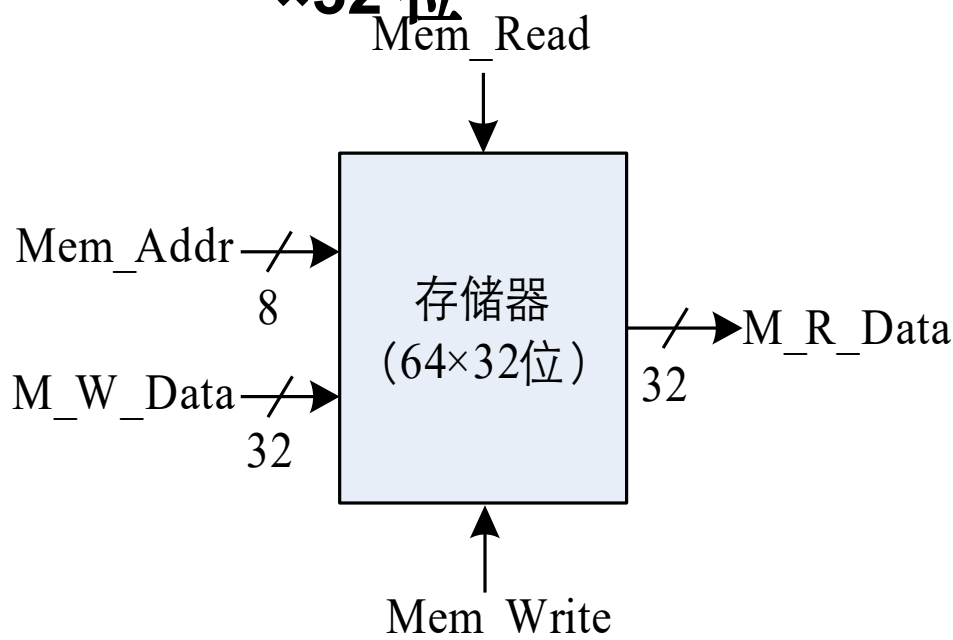
❖ 1、实验目的

- 掌握灵活运用 Verilog HDL 语言进行各种描述与建模的技巧和方法；
- 学习在 ISE 中使用 **Memory IP 核** 生成存储器的方法；
- 学习存储器的结构及读写原理，掌握存储器的设计方法。

实验五 存储器设计

❖ 2、实验内容与原理

- 为 MIPS 处理器设计一个 **256×8 位** 的物理存储器，具有读写功能，按**字节编址**，按**字访问**，即 **64×32 位**



字地址	MSB 位序 LSB			
	31	24	23	16 15 8 7 0
0	3	2	1	0
4	7	6	5	4
...
56	59	58	57	56
60	63	62	61	60

字地址小端格式

实验五 存储器设计

❖2、实验内容与原理

存储器功能表

输入信号				输出信号	操作
Mem_Read	Mem_Write	Mem_Address	M_Write_Data	M_Read_Data	
0	0	——	——	——	无操作
1	0	有效地址	——	读出数据	读操作
0	1	有效地址	有效数据	——	写操作

❖ 2、实验内容与原理

- **注意：**虽然存储器是按照每个单元 32 位来组织的，但是字地址是按照 0、4、8……等 4 的整数倍来递增的，给出的 8 位存储器地址，只按照高 6 位访问存储器，而低 2 位必须为 00。
- MIPS CPU 只有 LWL（取左半字）、LWR（取右半字）、SWL（存左半字）、SWR（存右半字）可以不按 4 字节边界访问存储器。

❖ 2、实验内容与原理

- 在实现方法上，存储器可以使用两种方法构造：
 - 方法一：采用 Verilog 语言中存储器类型（即 reg 的数组类型）定义，并自行管理；
 - 方法二：使用 FPGA 内置的存储器 IP 核来实现。

实验五 存储器设计

■ 两者的区别：

■ 前者（采用 Verilog 语言中存储器类型）：

- 存储器模块需要简单编程来完成读写操作，且其中的数据（或指令）的初始化，也需要在模块内编程赋值完成；

■ 后者（使用 FPGA 内置的存储器 IP 核）

- 无需编程，通过向导自动生成了一个存储模块，只需引用其实例即可，且其中的数据（或指令）的初始化操作可以和外部的一个格式化文件关联，ISE 会自动从该文件装载程序或数据。另外，由 Memory IP 核实现的存储器模块性能更好、更可靠。

❖ 实验内容与原理

■ 以 RAM 存储块为例：方法二

(1) 新建并编辑一个关联文档。

工程目录下新建 *.coe 的纯文本文件（如 Test_Mem.coe）
格式如下：

```
memory_initialization_radix=<Radix>;  
memory_initialization_vector=<data1>, <data2>, ... <data  
n>;
```

第一行说明数据格式：即第二行的 <data> 采用的进制；

第二行定义初始化向量。

实验五 存储器设计

❖ 譬如：

`memory_initialization_radix=16`

`memory_initialization_vector=00000820, 00632020,
00010fff, 20006789, FFFF0000, 0000FFFF, 88888888,
99999999, aaaaaaaaa, bbbbbbbb;`

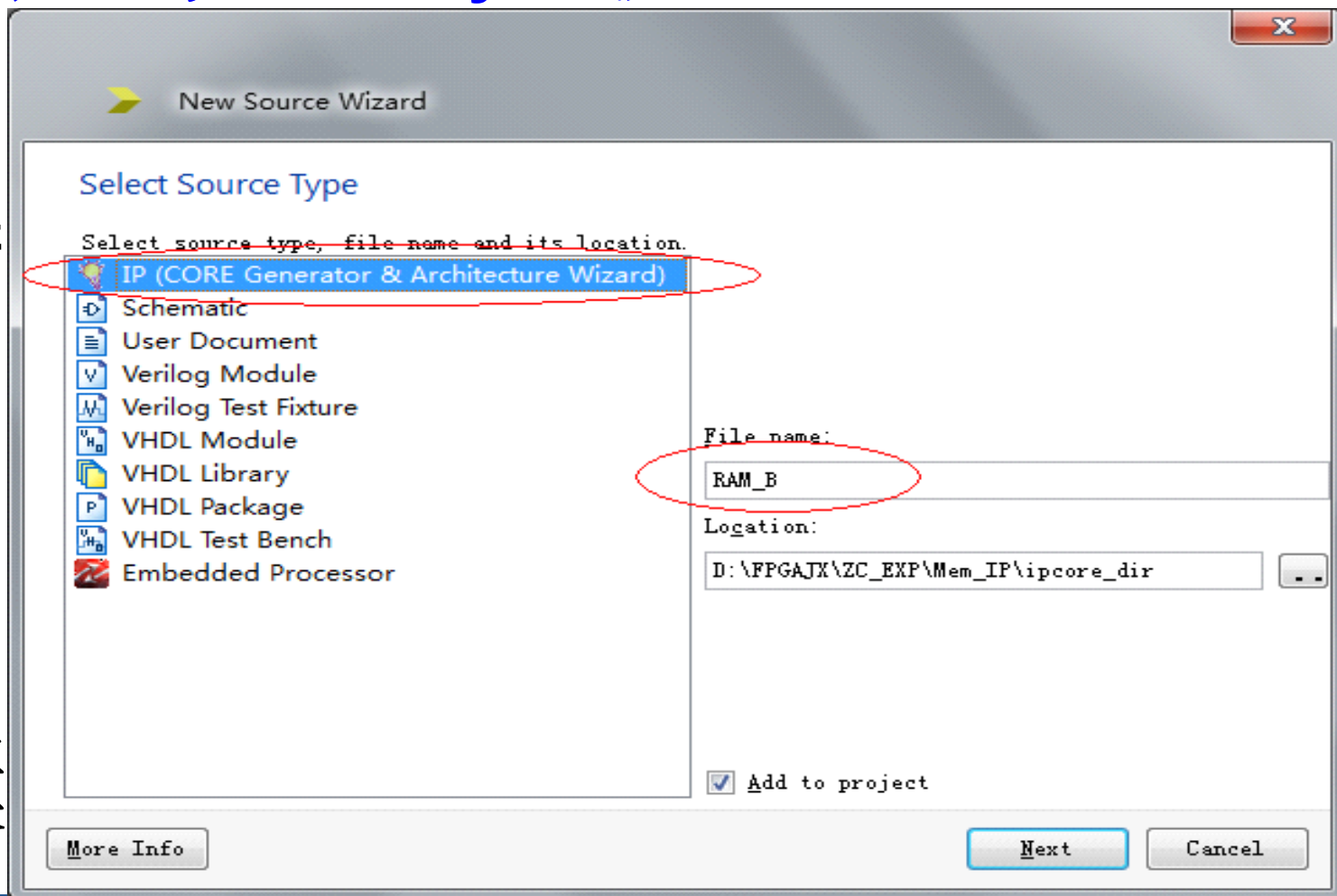
含义：预备要初始化存储器的前十个单元的数据，**数据在第二行“=”右边，且以16进制表示**，各数据以**逗号分隔，以分号结束**，数据依次装入地址为0开始的各单元。

实验五 存储器设计

(2) 新建一个 Memory IP 核。

Project→**New Source** 菜单，弹出 NewSource Wizard 对话框，**Select Source Type** 列表中，选择 **IP**，输入存储器 IP 核的名称（本例 RAM_B）

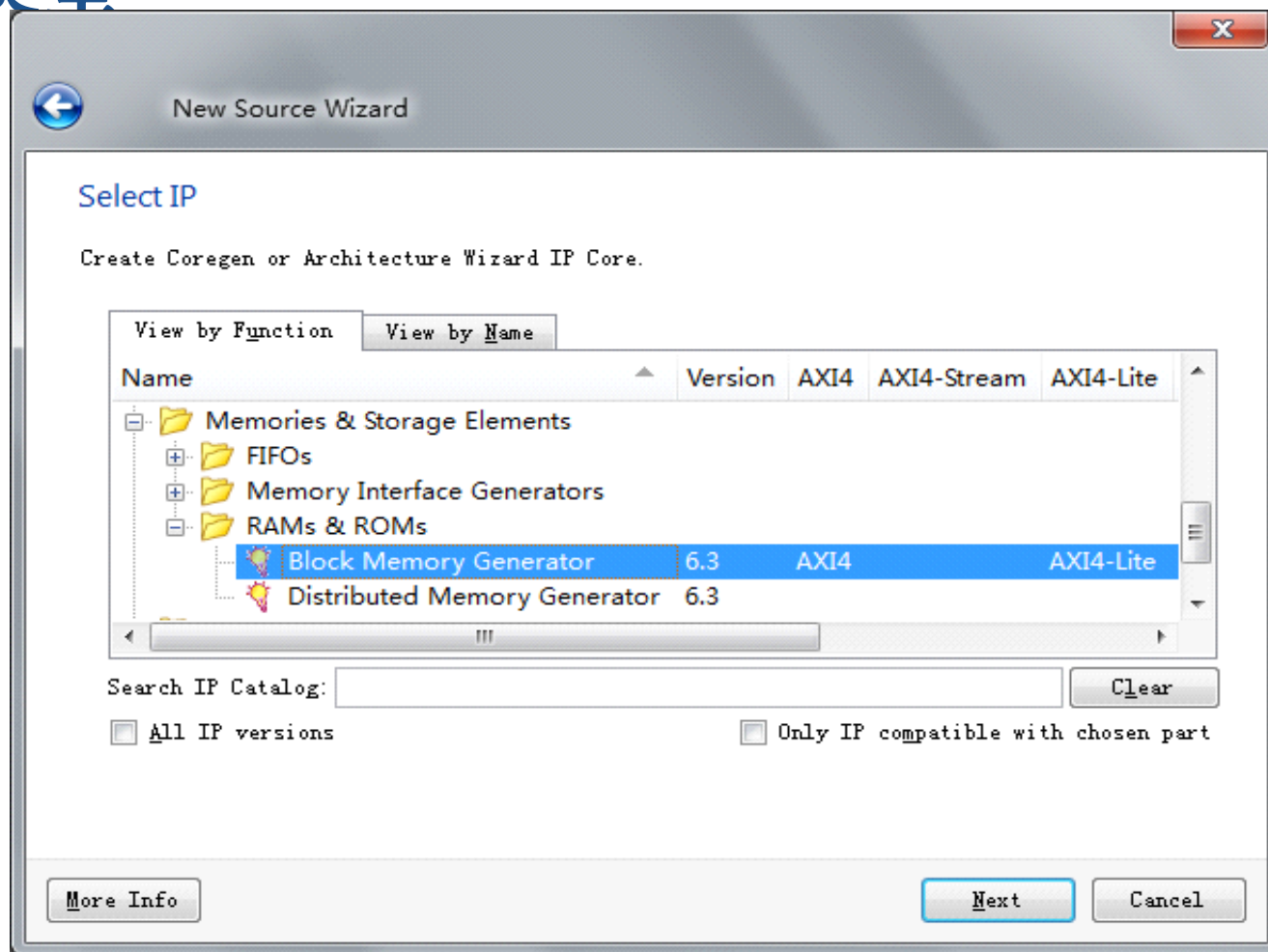
IP 核存放位置的缺省路径为工程目录的子目录 `\ipcore_dir` 下。



实验五 存储器设计

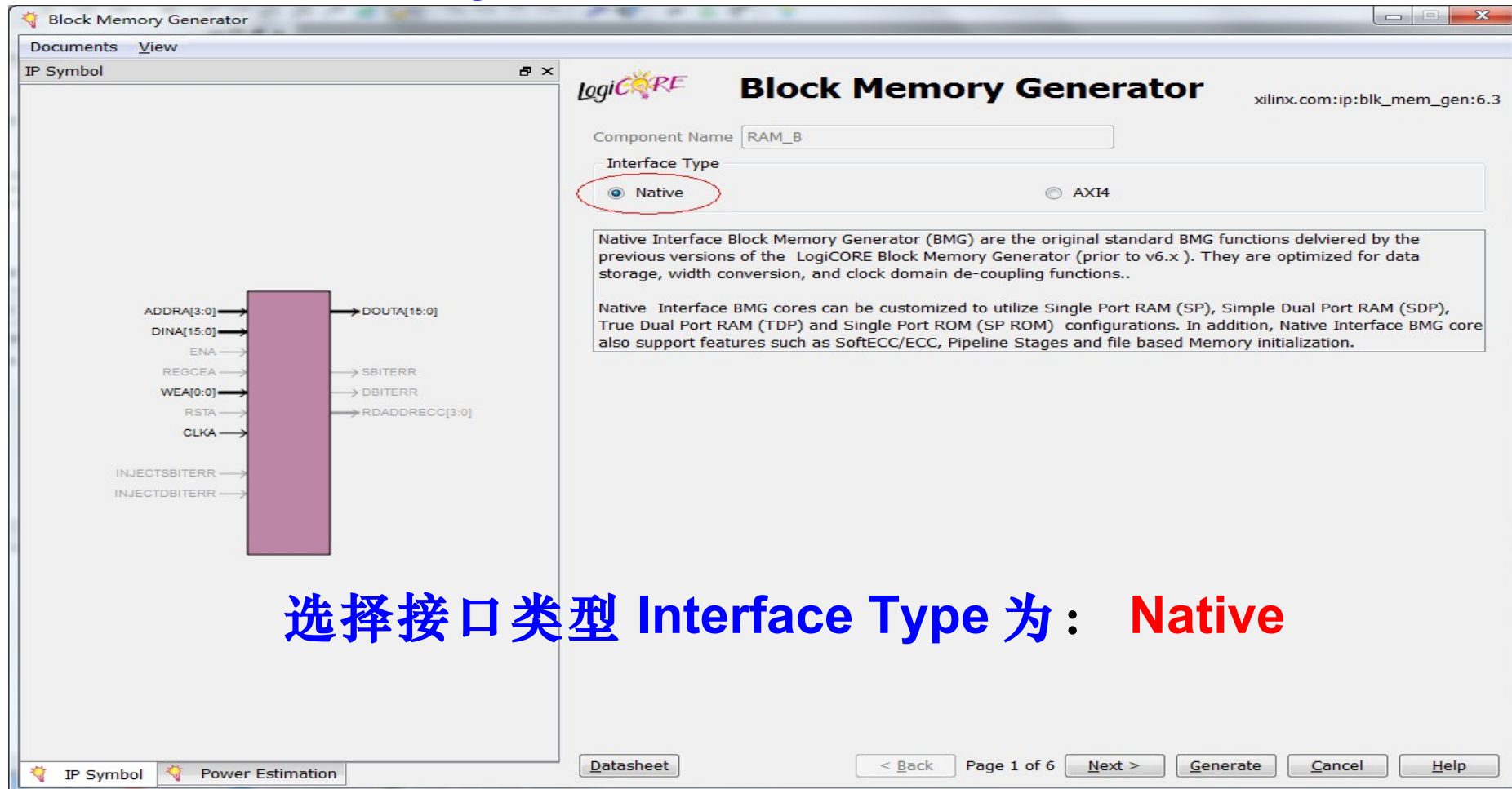
❖ 选择 IP 核类型

依次选择 **Memories & Storage Elements** →
RAMs & ROMs →
Block Memory Generator



实验五 存储器设计

(3) Memory IP 核参数设置。



The screenshot displays the Block Memory Generator (BMG) configuration window. The 'Interface Type' section is highlighted, showing 'Native' selected. The 'Component Name' is 'RAM_B'. The 'IP Symbol' tab is active, showing a block diagram of the memory core with various input and output ports.

Block Memory Generator
xilinx.com:ip:blk_mem_gen:6.3

Component Name: RAM_B

Interface Type:
☒ Native ☐ AXI4

Native Interface Block Memory Generator (BMG) are the original standard BMG functions delivered by the previous versions of the LogiCORE Block Memory Generator (prior to v6.x). They are optimized for data storage, width conversion, and clock domain de-coupling functions..

Native Interface BMG cores can be customized to utilize Single Port RAM (SP), Simple Dual Port RAM (SDP), True Dual Port RAM (TDP) and Single Port ROM (SP ROM) configurations. In addition, Native Interface BMG core also support features such as SoftECC/ECC, Pipeline Stages and file based Memory initialization.

IP Symbol

Power Estimation

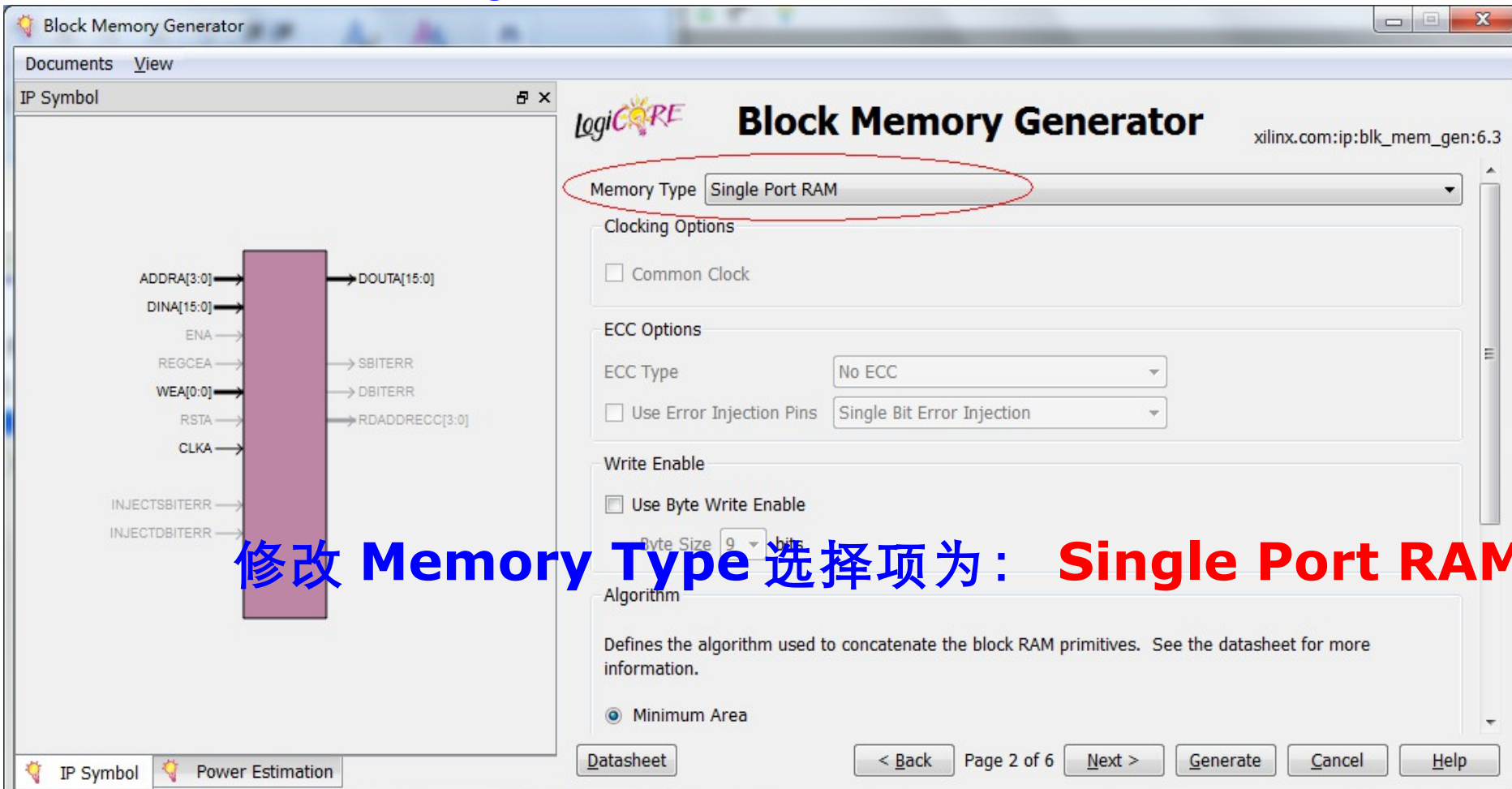
Datasheet

< Back Page 1 of 6 Next > Generate Cancel Help

选择接口类型 Interface Type 为： **Native**

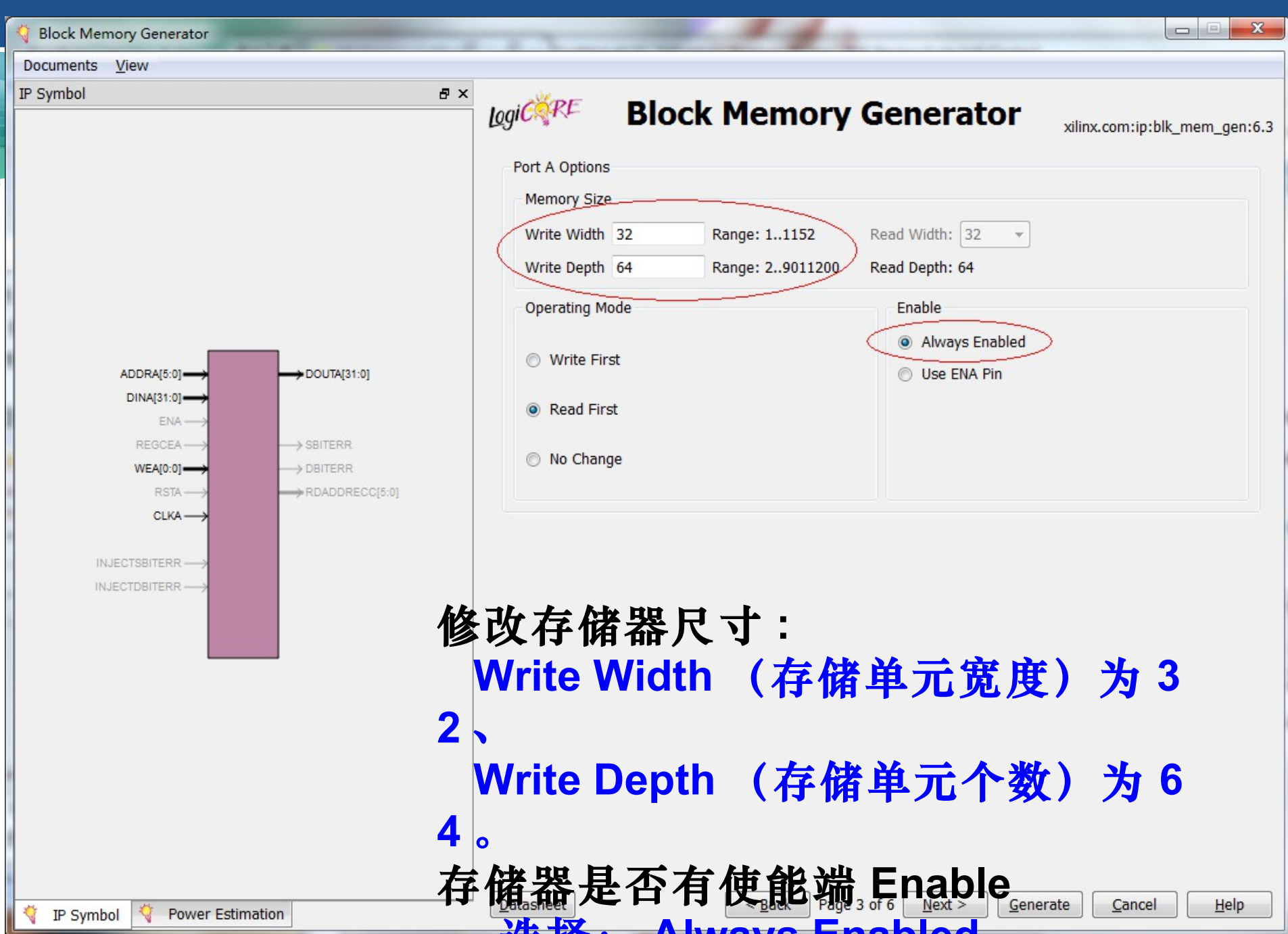
实验五 存储器设计

(3) Memory IP 核参数设置。



The screenshot displays the 'Block Memory Generator' tool interface. On the left, the 'IP Symbol' pane shows a vertical purple rectangle representing the memory block, with input and output pins labeled: ADDR[3:0], DINA[15:0], ENA, REGCEA, WEA[0:0], RSTA, CLKA, DOUTA[15:0], SBITERR, DBITERR, and RDADRECC[3:0]. Below this, there are buttons for 'IP Symbol' and 'Power Estimation'. The main configuration pane on the right is titled 'Block Memory Generator' and includes the URL 'xilinx.com:ip:blk_mem_gen:6.3'. The 'Memory Type' dropdown menu is highlighted with a red circle and set to 'Single Port RAM'. Below this, the 'Clocking Options' section has an unchecked 'Common Clock' checkbox. The 'ECC Options' section shows 'ECC Type' set to 'No ECC' and 'Use Error Injection Pins' unchecked, with 'Single Bit Error Injection' selected in the dropdown. The 'Write Enable' section has 'Use Byte Write Enable' unchecked. The 'Algorithm' section shows 'Minimum Area' selected. At the bottom, there are buttons for '< Back', 'Page 2 of 6', 'Next >', 'Generate', 'Cancel', and 'Help'. A red text overlay is present over the bottom right of the configuration pane.

修改 Memory Type 选择项为: **Single Port RAM**



修改存储器尺寸：

Write Width（存储单元宽度）为 32、

Write Depth（存储单元个数）为 64。

存储器是否有使能端 Enable

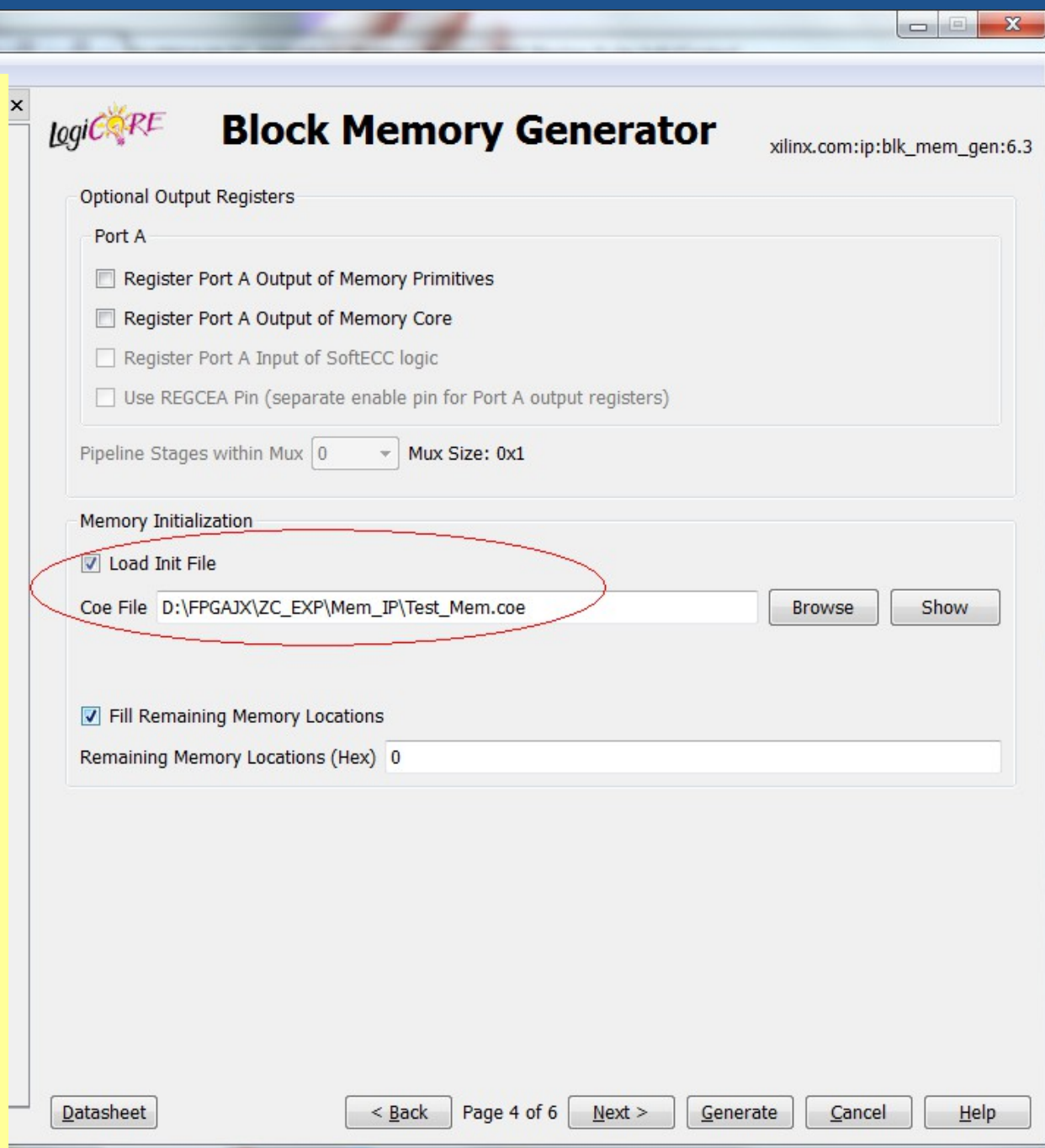
选择：Always Enabled

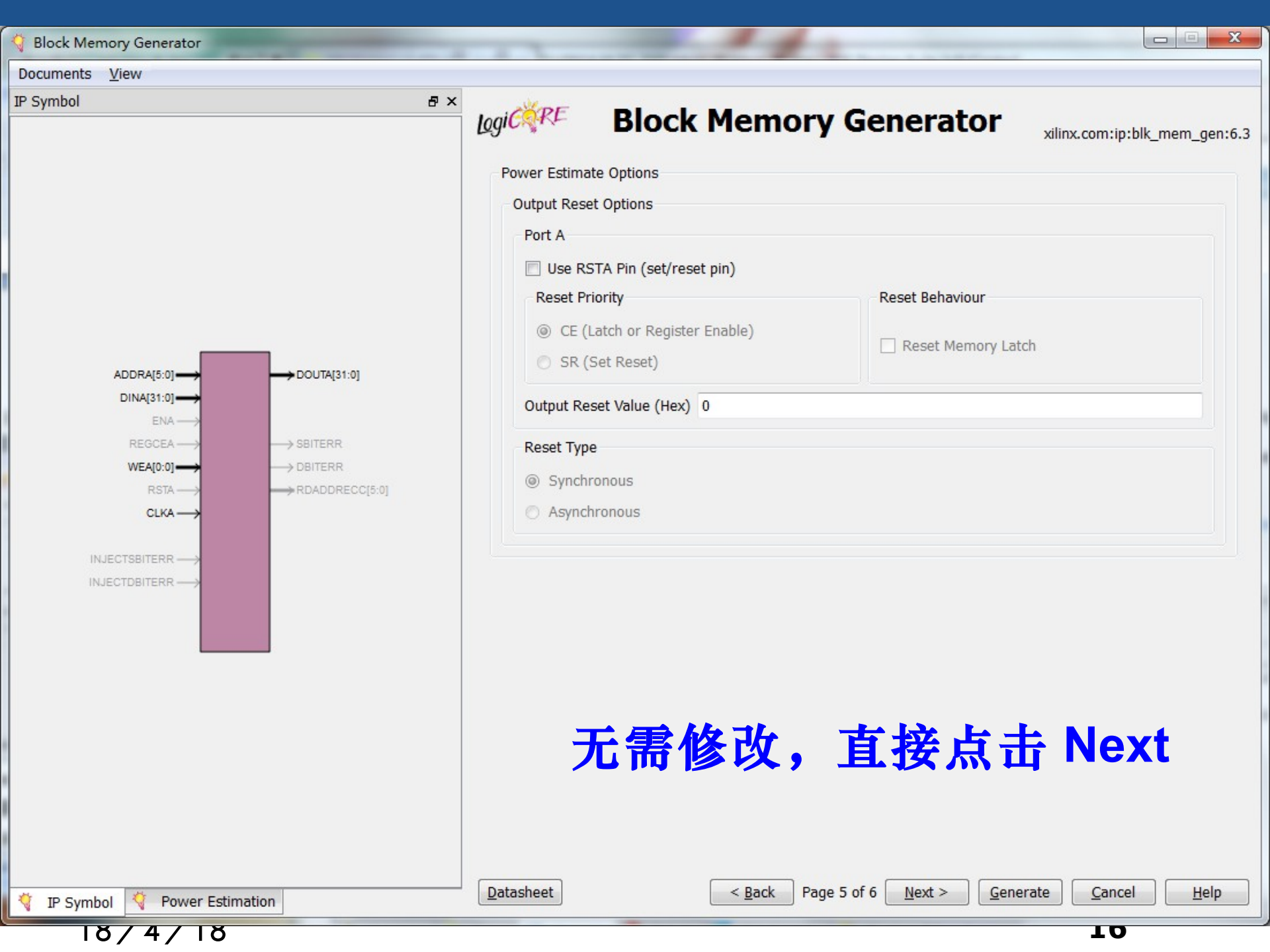
1. 选中 Load Init File 选项

2. 点击 Browse 按钮
选择第一步生成的 COE 文档 (Test_Mem.coe) 作为关联初始化参数。

3. RAM_B IP 生成后，
ISE 会用这个文档内的数据初始化 RAM 值。

注意：当该关联文件
内容修改后，都需重新
执行一次 Regenerate Core，
才能重新初始化存储器。





Block Memory Generator

xilinx.com:ip:blk_mem_gen:6.3

Power Estimate Options

Output Reset Options

Port A

☐ Use RSTA Pin (set/reset pin)

Reset Priority

- ☒ CE (Latch or Register Enable)
- ☐ SR (Set Reset)

Reset Behaviour

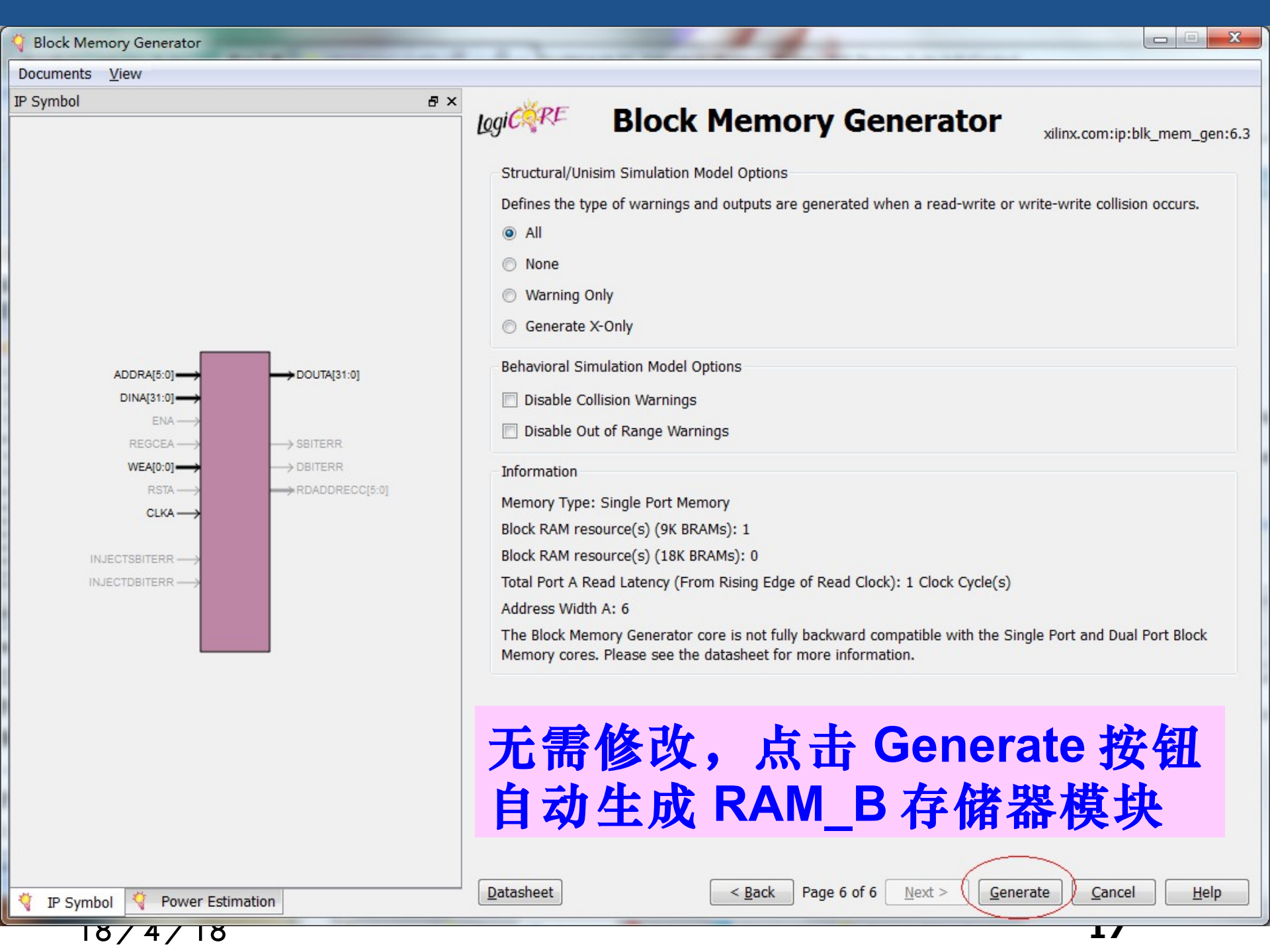
☐ Reset Memory Latch

Output Reset Value (Hex) 0

Reset Type

- ☒ Synchronous
- ☐ Asynchronous

无需修改，直接点击 Next



Block Memory Generator

xilinx.com:ip:blk_mem_gen:6.3

Structural/Unisim Simulation Model Options

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

- ☒ All
- ☐ None
- ☐ Warning Only
- ☐ Generate X-Only

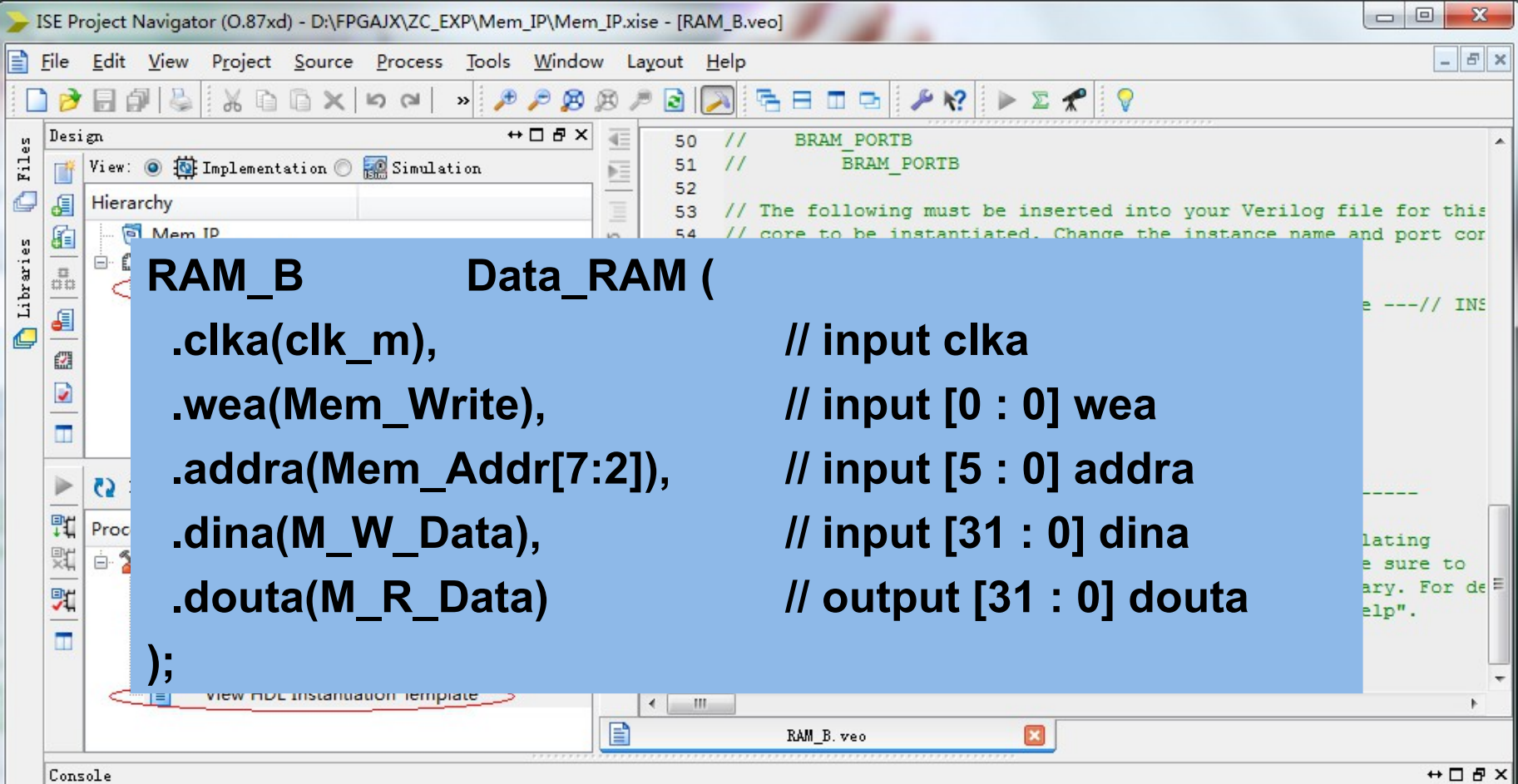
Behavioral Simulation Model Options

- ☐ Disable Collision Warnings
- ☐ Disable Out of Range Warnings

Information

Memory Type: Single Port Memory
Block RAM resource(s) (9K BRAMs): 1
Block RAM resource(s) (18K BRAMs): 0
Total Port A Read Latency (From Rising Edge of Read Clock): 1 Clock Cycle(s)
Address Width A: 6
The Block Memory Generator core is not fully backward compatible with the Single Port and Dual Port Block Memory cores. Please see the datasheet for more information.

无需修改，点击 **Generate** 按钮
自动生成 **RAM_B** 存储器模块



双击过程管理区的 **View HDL Instruction Template**，在右侧代码区会给出 RAM_B 的调用模板（示范代码），将其**拷贝到顶层模块**中，修改模块实例名称和连接端口参数，就可以像一般模块那样引用

❖ 3、实验要求

- 按照**方法一**，编程实现基本的存储器模块，并通过仿真验证
- 按照**方法二**，生成一个 RAM_B 存储器模块，关联文件中输入 64 个 32 位数据，16 进制表示
- 编写一个实验验证的顶层模块，调用方法二生成的存储器模块；
- **课前任务**：**编程、仿真、验证**，确保逻辑正确性；

❖3、实验要求

- 实验室任务：
 - 配置管脚：见下表
 - 生成 *.bit 文件，下载到 Nexys3 实验板中。
 - 完成板级验证。
- 撰写实验报告。

实验五 存储器设计

❖ 信号配置表

	信号	配置设备 管脚	功能说明
输入 信号	Mem_Addr[7:2]	6 个逻辑开 关	读写存储器地址
	选择信号	2 个逻辑开 关	读操作时，选择显示的字节 ； 写操作时，选择要写入的数据
	Mem_Write	1 个按钮	=1 为写操作； =0 为读操作
	Clk	1 个按钮	时钟引脚
输出 信号	LED[7:0]	8 个 LED 灯	显示读出数据的字节

❖ 4、实验步骤

- 使用合适的建模方法和语句进行编程、仿真；
- 启动计算机，拷贝工程文件到硬盘上；
- 实验准备：
 - 设置 N3 板卡电源跳线 J1，从 USB 取电；
 - 用 USB 电缆连接 PC 机和 N3 板卡；
 - 开 N3 实验板的电源开关；
- 在 PC 机上打开工程文件，进行管脚配置。
- 生成编程文件 *.bit，下载到板卡中。
- 实验。


❖ 5、思考与探索：必做（1）、（2）

- （1）选择 8 个存储器单元执行**读操作**，将实验结果记录到表中的第二列和第三列，分析你的读出数据是否和初始化关联文件中的数据一致；若不一致，分析原因。
- （2）对上面的 8 个存储器单元执行**写操作，覆盖初始化数据，然后再执行读操作**，将读出数据记录到表的第四列和第五列中。这些单元的数据有否改写？分析读出数据是否和写入数据一致；如果不一致，请分析原因。

❖ 5、思考与探索：

- （3）若设计实现一个 **ROM**，考虑端口信号有何不同？尝试分别使用方法一和方法二设计实现一个 ROM
- （4）考虑调用实验三所实现的基本 ALU 模块、实验四实现的寄存器堆模块和本实验所实现的存储器模块，编写一个顶层模块，完成 $R_i \leftarrow (addr)$ → R_j 的操作。尝试编写代码，仿真调试通过。





The End!