



杭州电子科技大学
HANGZHOU DIANZI UNIVERSITY

实验项目



- ❖ 容易出现的问题：
- ❖ 模块调用（引用实例）时，将其放在always语句块中：**出错，不允许**
 - **错误观点：**认为程序是按序执行，**实际上**是硬件电路模块与模块之间的信号连接。
 - 串行执行和并行执行的正确理解：

程序1：阻塞赋值

```
always @(*)
```

```
begin
```

```
    reg1 = in1;
```

```
    reg2 = in2 ^ in3;
```

```
    reg3 = reg1; //reg1的新值
```

```
end
```

程序2：非阻塞赋值

```
always @(posedge clk)
```

```
begin
```

```
    reg1 <= in1;
```

```
    reg2 <= in2 ^ in3;
```

```
    reg3 <= reg1; //reg1的旧值
```

```
end
```

❖ ! 和~的区别

$$ZF = \overline{F_{31} + F_{30} + \cdots + F_1 + F_0}$$

❖ ZF的实现

❖ OF只在进行算术加和算术减时有效，其他OF=0

❖ 整个工程中一个模块：包括输入数据选择、ALU运算功能、输出显示；

- 建议：最好将ALU定义为一个独立模块，以便后续实验使用。
- 部件实验一般至少分为2个模块：
 - 部件功能模块：核心功能，可供后续设计使用；
 - 验证模块：用于验证部件功能是否正确，做实验时使用。

实验四 寄存器堆设计

实验五 存储器设计

实验六 MIPS汇编器与模拟器实验

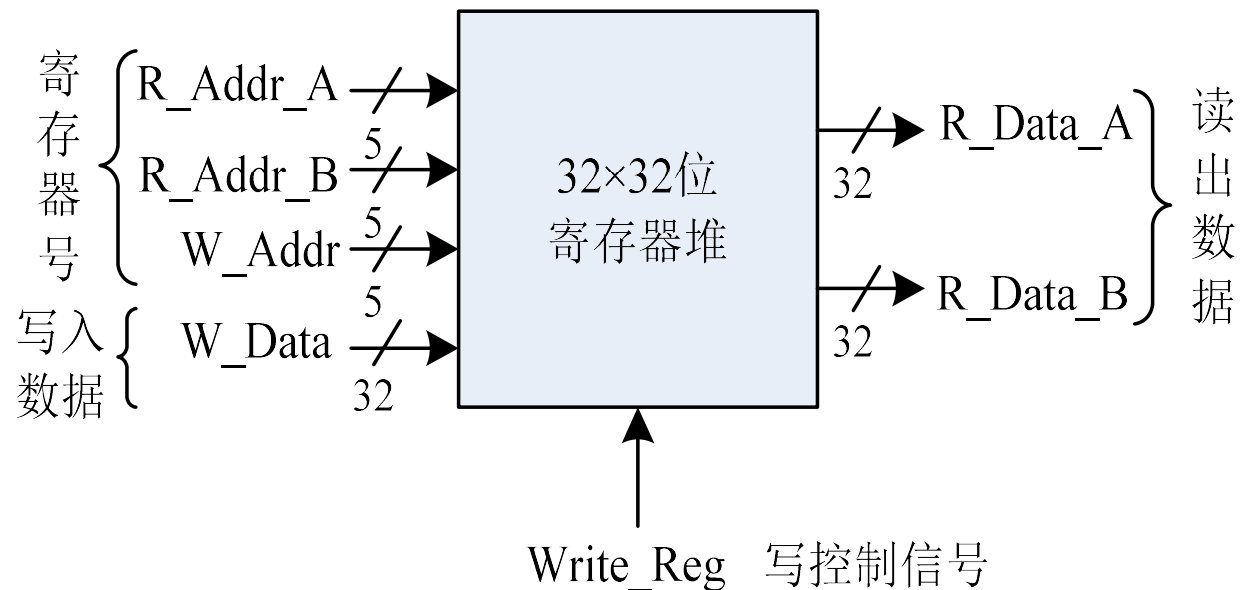
❖ 1、实验目的

- 学习使用Verilog HDL语言进行**时序电路**的设计方法；
- 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**寄存器堆的数据传送与读写工作原理**，掌握寄存器堆的设计方法。

实验四 寄存器堆设计

❖ 2、实验内容与原理

- 设计一个**32×32位**的寄存器堆（即含有32个寄存器，每个寄存器32位）

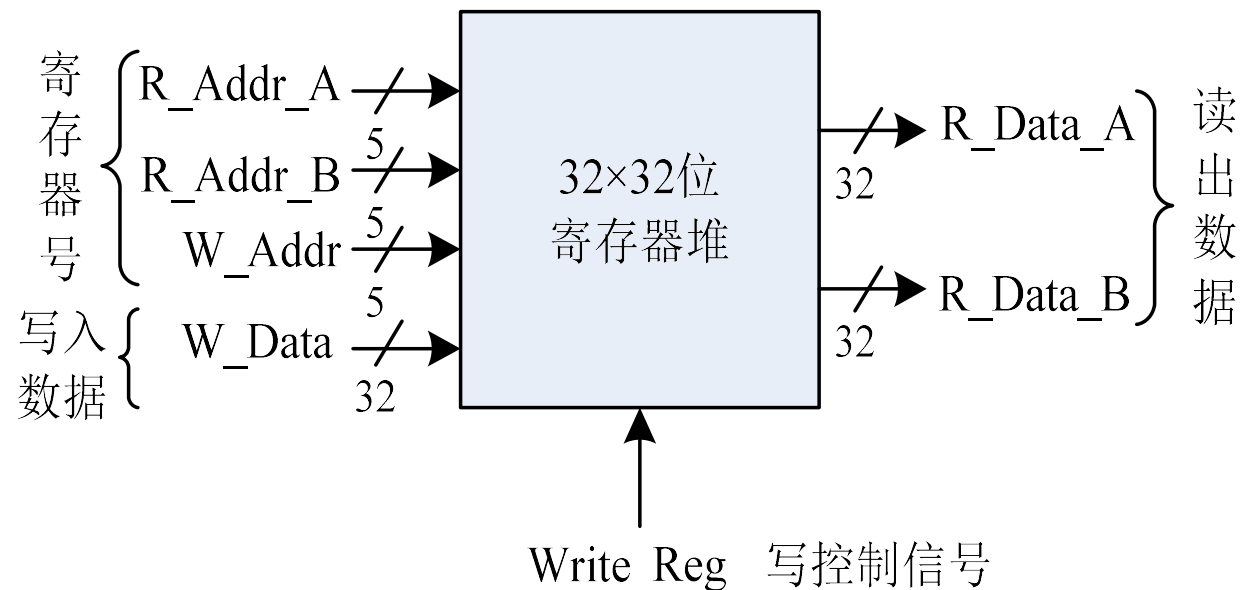


输入输出端口示意图

- ❖ 双端口读：2个读端口
- ❖ 单端口写：1个写端口

实验四 寄存器堆设计

- ❖ **读访问操作：无需时钟同步**，只要给出寄存器地址，即可读出寄存器中的数据。
- ❖ **写访问操作：需要时钟同步**，所有写入操作的输入信号必须在时钟边沿来临时，已经有效（**Write_Reg=1、地址和数据**）。



实验四 寄存器堆设计

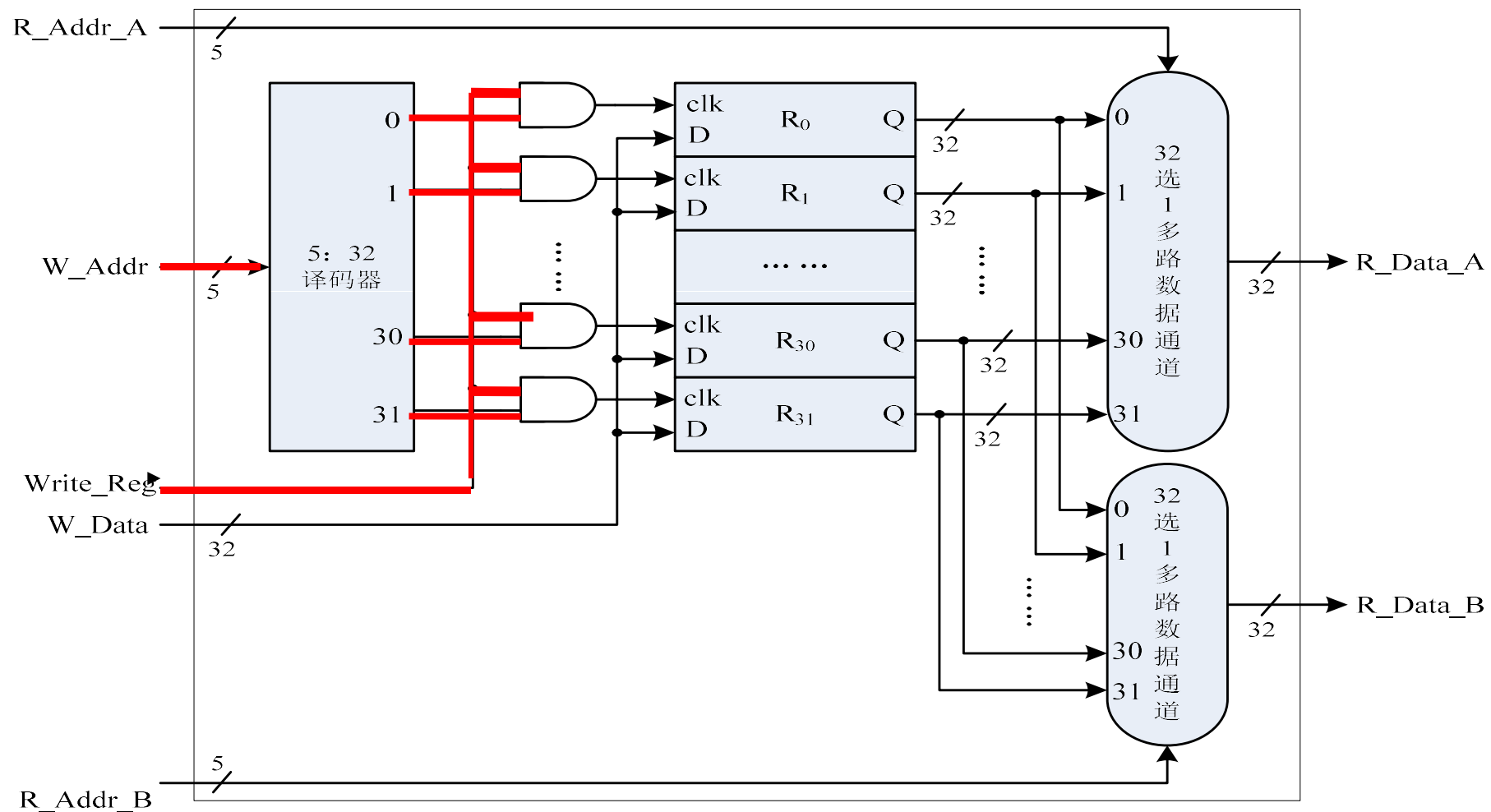
❖ 2、实验内容与原理

寄存器堆功能表

输入信号					输出信号		操作
R_Addr_A	R_Addr_B	Write_Reg	W_Addr	W_Data	R_Data_A	R_Data_B	
寄存器号	—	—	—	—	A口数据	—	读A口
—	寄存器号	—	—	—	—	B口数据	读B口
—	—	1	寄存器号	写入数据	—	—	写操作

实验四 寄存器堆设计

❖ 2、实验内容与原理 寄存器堆逻辑结构图



❖ 实验内容与原理

■ 实验实现：

■ 寄存器堆：reg类型信号的数组

```
reg [31:0] REG_Files[0:31];
```

■ 读操作：组合逻辑电路

```
assign R_Data_A = REG_Files[R_Addr_A];
```

```
assign R_Data_B = REG_Files[R_Addr_B];
```

■ 写操作：时序逻辑电路

- 需要Reset信号：用于初始化寄存器（全部清零）

- 需要clk信号：用于写入寄存器

实验四 寄存器堆设计

```
always @(posedge Clk or posedge Reset)
begin
    if(Reset) //高电平有效, =1则初始化
        .....//初始化32个寄存器
    else
        begin
            if (Write_Reg)
                .....//写入寄存器;
        end
    end
end
```

❖ 3、实验要求

- 编程实现基本的寄存器堆模块，并通过仿真验证；
- 编写一个实验验证的顶层模块，调用该寄存器堆模块
- 参考方法如下：
 - 使用5位开关提供读写的寄存器地址；
 - 1位开关提供Write_Reg信号；指定Write_Reg=0时执行读操作；=1时执行写操作；
 - 2位开关作为复用控制：若为读操作时，用于选择读出的32位数据的某个字节到8位LED灯显示；若为写操作，则选择4个指定数据之一作为写入数据。
 - 1个按钮提供Clk；1个按钮提供Reset；一个按钮作为读A端口/B端口的选择；
 - 8位LED灯作为读出数据的字节显示

❖ 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。

实验四 寄存器堆设计

❖ 信号配置表

	信号	配置设备管脚	功能说明
输入 信号	地址信号	5个逻辑开关	提供读A、读B、写地址
	写信号	1个逻辑开关	=1为写操作；=0为读操作
	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
	Clk、Reset	2个按钮	必须连接到时钟引脚BTND (C9) 或BTNR (D9)
	A /B读端口选择	1个按钮	选择读出A端口还是B端口
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

- 在Xilinx ISE中创建工程，编源码，然后编译、综合
- 编写激励代码，观察仿真波形，直至验证正确
- **实验准备：**
 - 设置N3板卡电源开关跳线J1，选择从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。**

实验四 寄存器堆设计

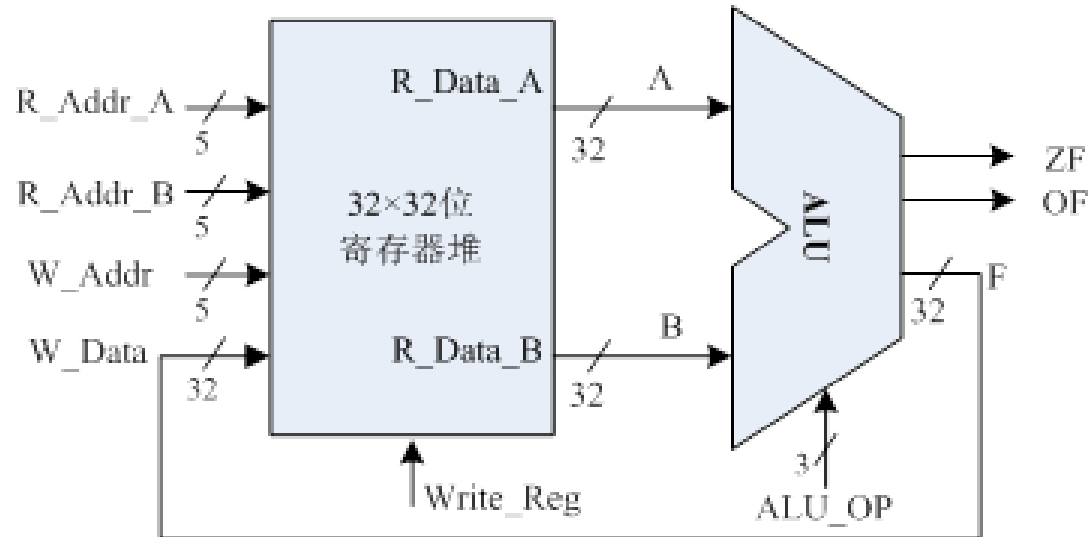
❖ 5、思考与探索：必做（1）和（2）

- （1）选择8个寄存器执行读写操作，将实验结果记录到表中，结果是否符合预期；如果不符，分析原因。
- （2）修改基本寄存器堆模块，实现MIPS计算机的寄存器堆，以供后续MIPS CPU的设计使用。
- （3）谈谈实验中读操作和写操作在时序上有何区别？反映到电路实现上，又有何不同？

实验四 寄存器堆设计

❖ 思考与探索

- (4) 利用实验三ALU模块和本实验的寄存器堆模块，编写一个顶层模块，完成 $R_i \theta R_j \rightarrow R_k$ 的操作（即2个寄存器数据做某种运算，结果送回第3个寄存器中）运算功能 θ 由ALU模块中的ALU_OP信号指定。



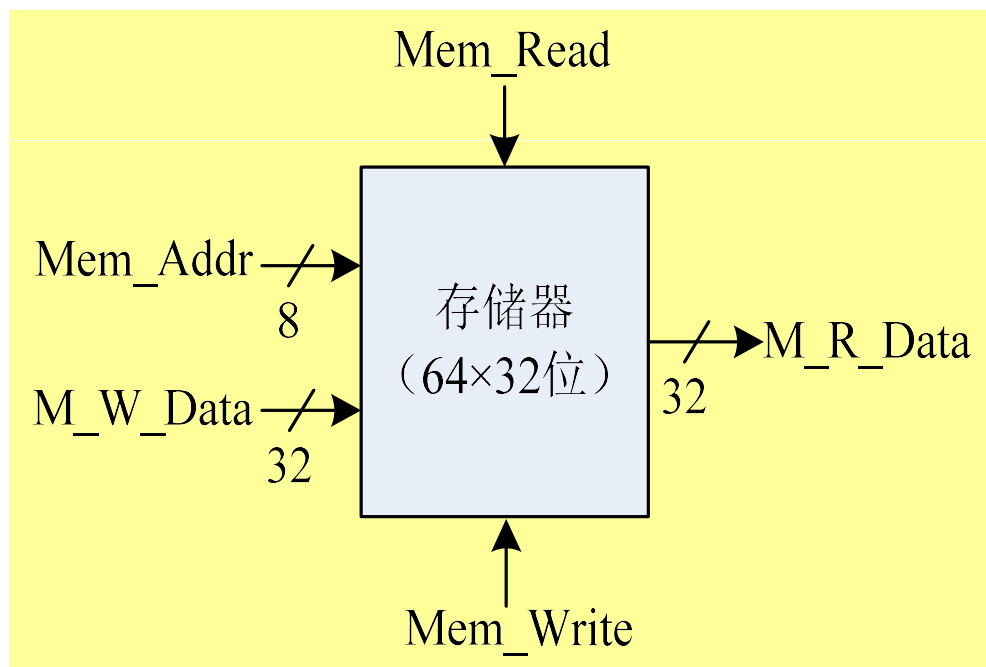
❖ 1、实验目的

- 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_Addr	M_W_Data	M_R_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。

■ 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`

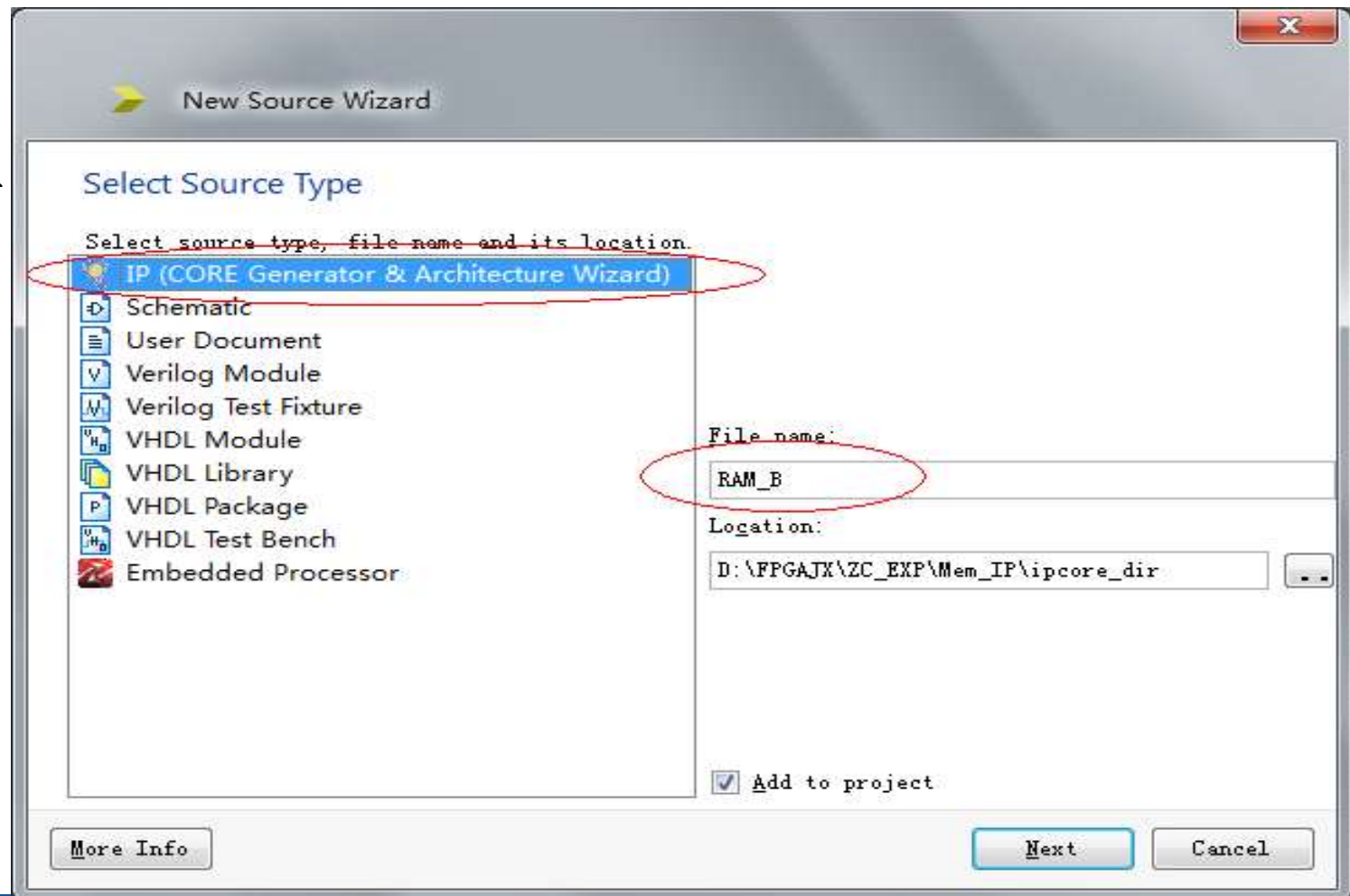
`memory_initialization_vector=00000820, 00632020,
00010fff, 20006789, FFFF0000, 0000FFFF, 88888888,
99999999, aaaaaaaaa, bbbbbbbb;`

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。

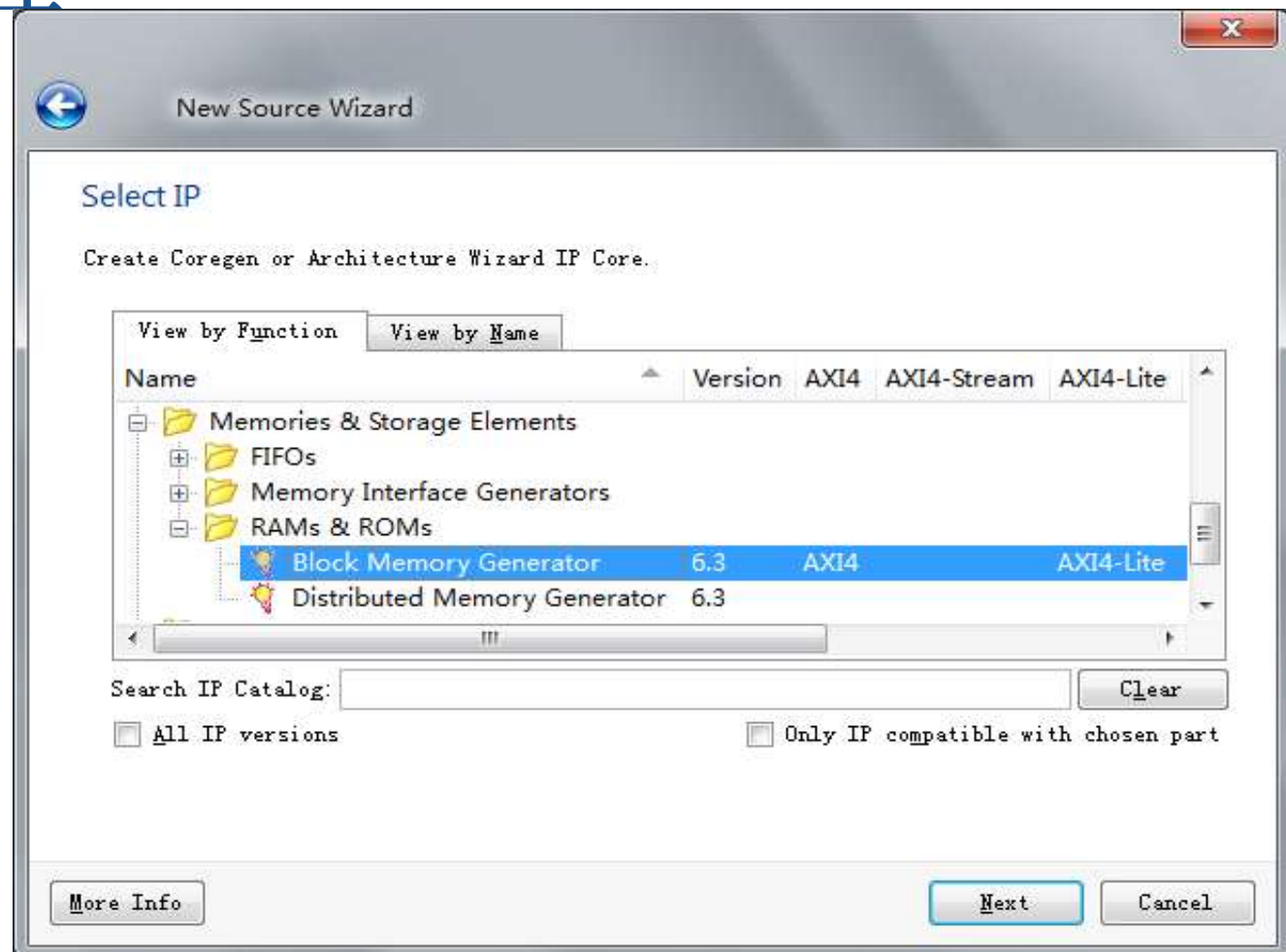
Project→**New Source**菜单，弹出**NewSource Wizard**对话框，**SelectSource Type**列表中，选择**IP**，输入存储器**IP**核的名称（本例**RAM_B**）**IP**核存放位置的缺省路径为工程目录的子目录**\ipcore_dir**下。



实验五 存储器设计

❖ 选择IP核类型

依次选择
Memories&StorageElements→
RAMs &ROMs→
BlockMemoryGenerator



实验五 存储器设计

(3) Memory IP核参数设置。

Block Memory Generator

Documents View

IP Symbol

LogiCORE 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.

ADDRA[3:0] → DOUTA[15:0]
DINA[15:0] →
ENA →
REGCEA → SBITERR
WEA[0:0] → DBITERR
RSTA → RDADDRECC[3:0]
CLKA →
INJECTSBITERR →
INJECTDBITERR →

选择接口类型Interface Type为: **Native**

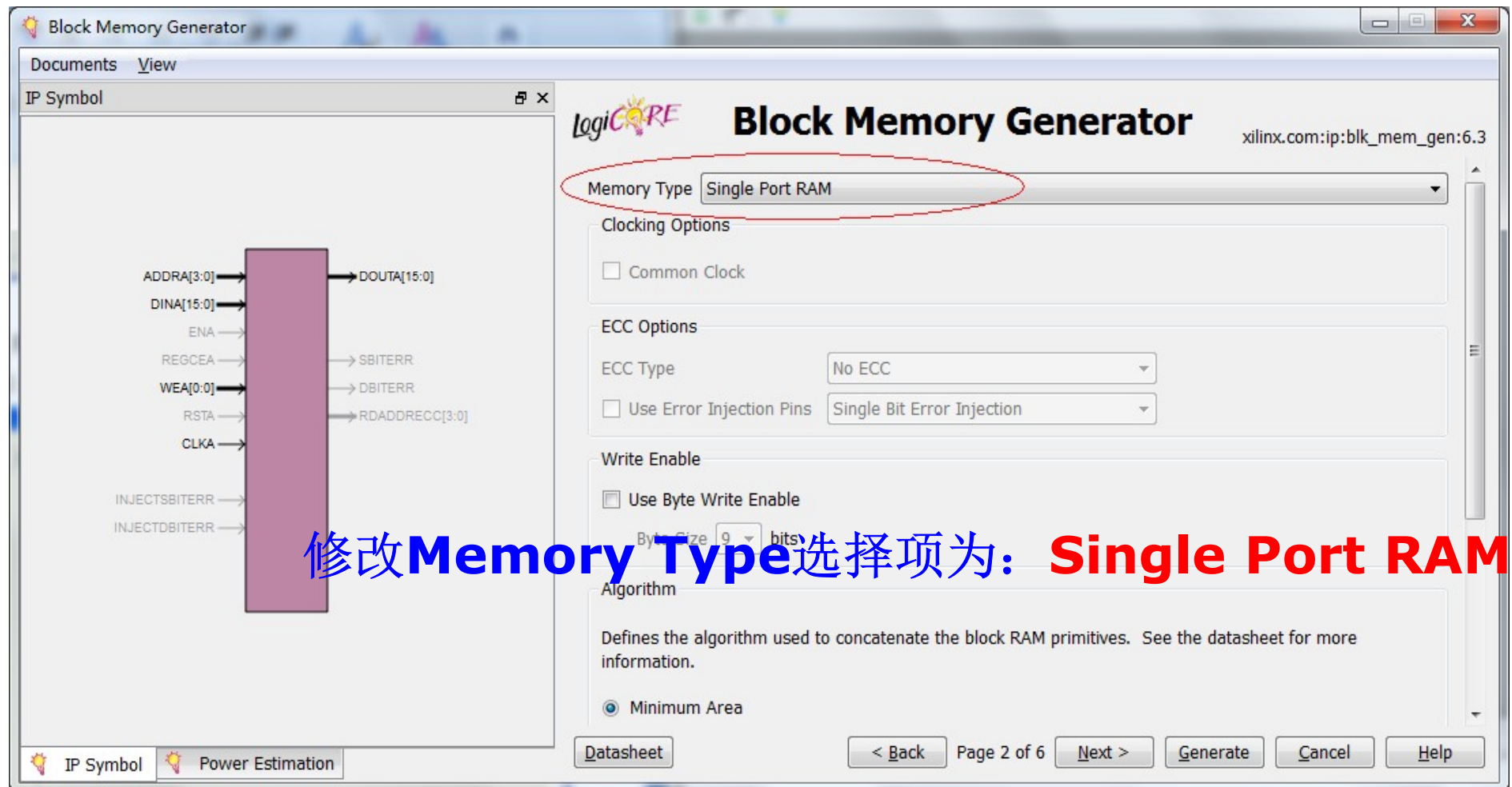
IP Symbol Power Estimation

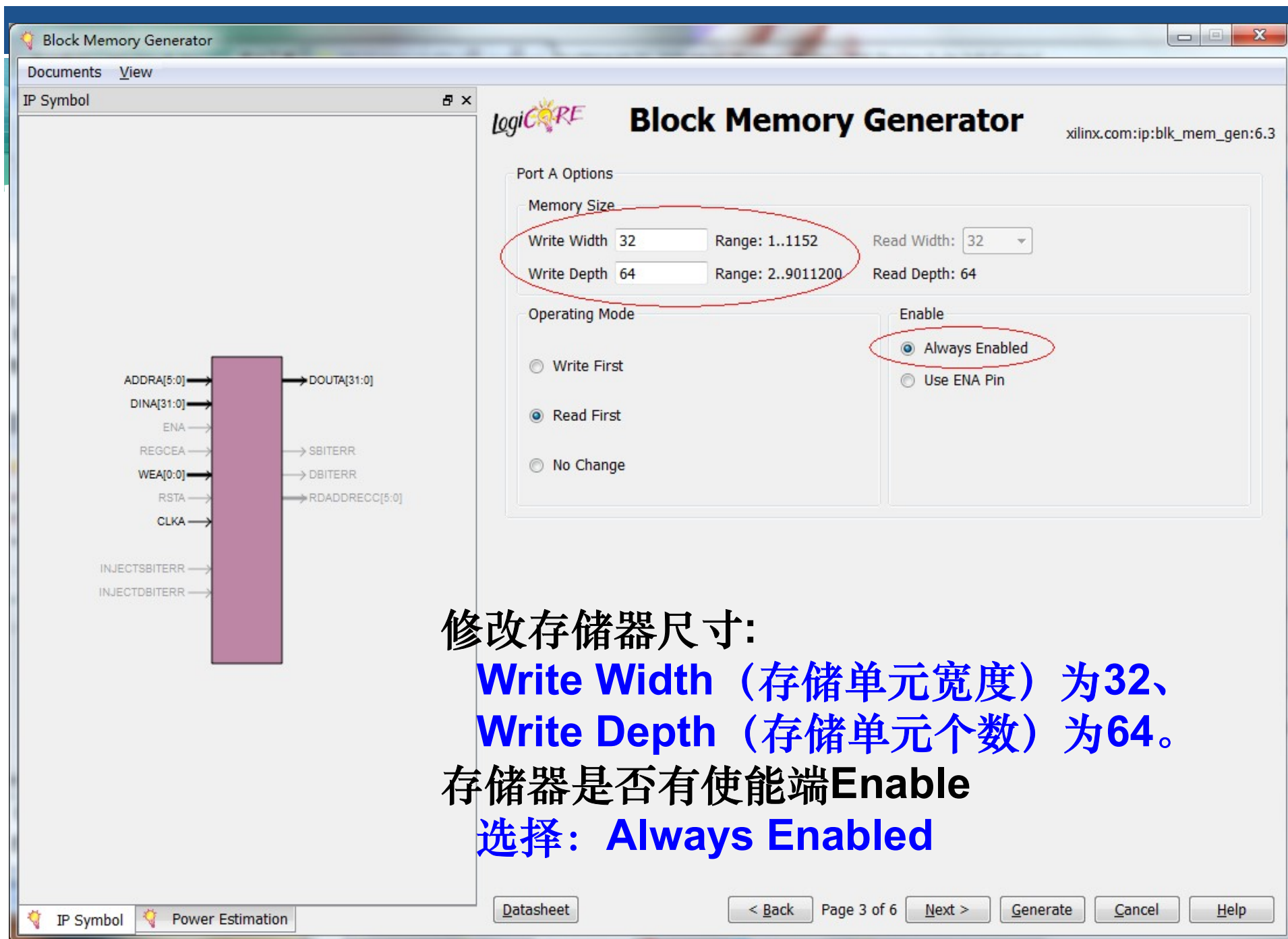
Datasheet

< Back Page 1 of 6 Next > Generate Cancel Help

实验五 存储器设计

(3) Memory IP核参数设置。





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**

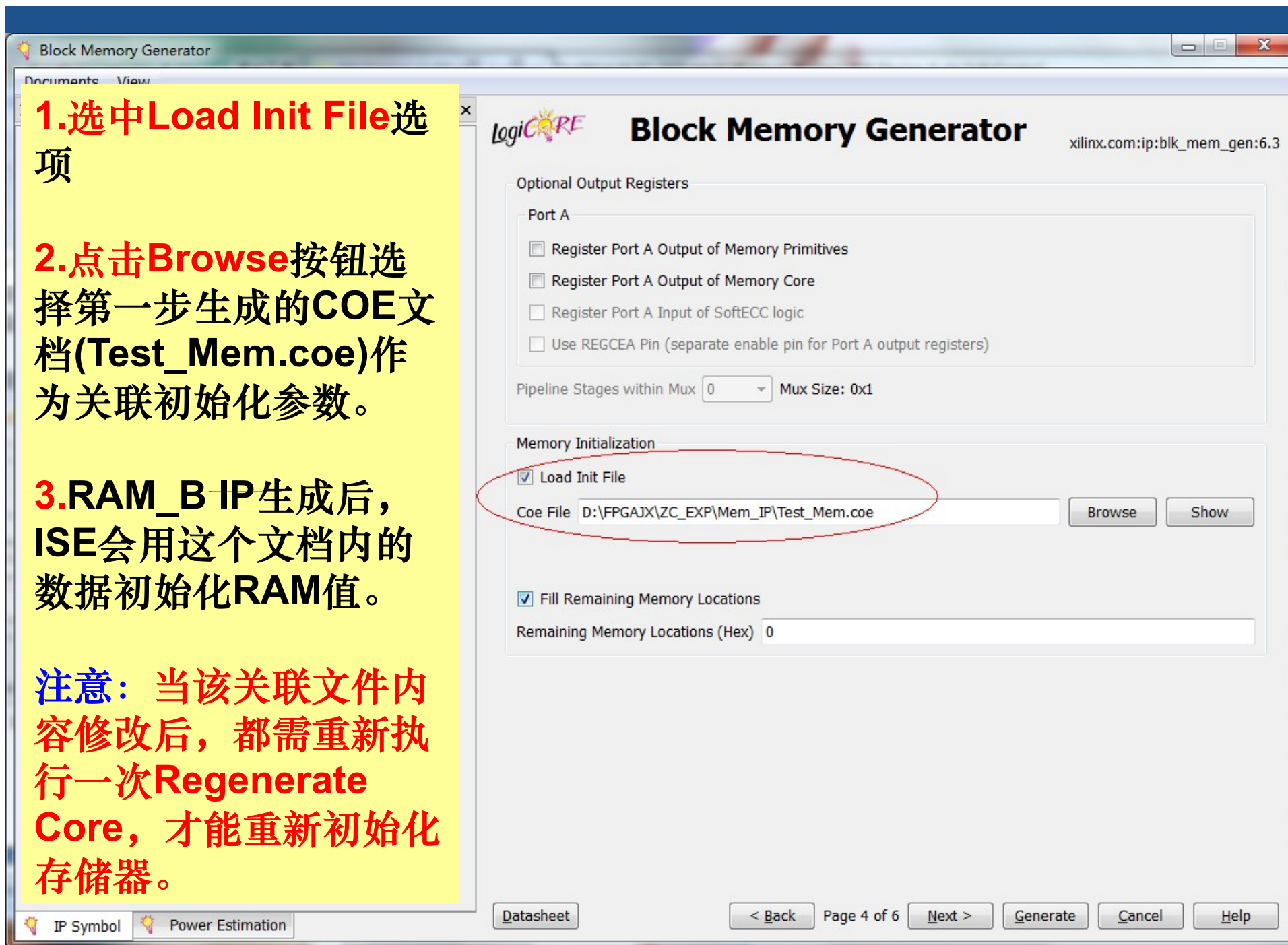
选择: **Always Enabled**

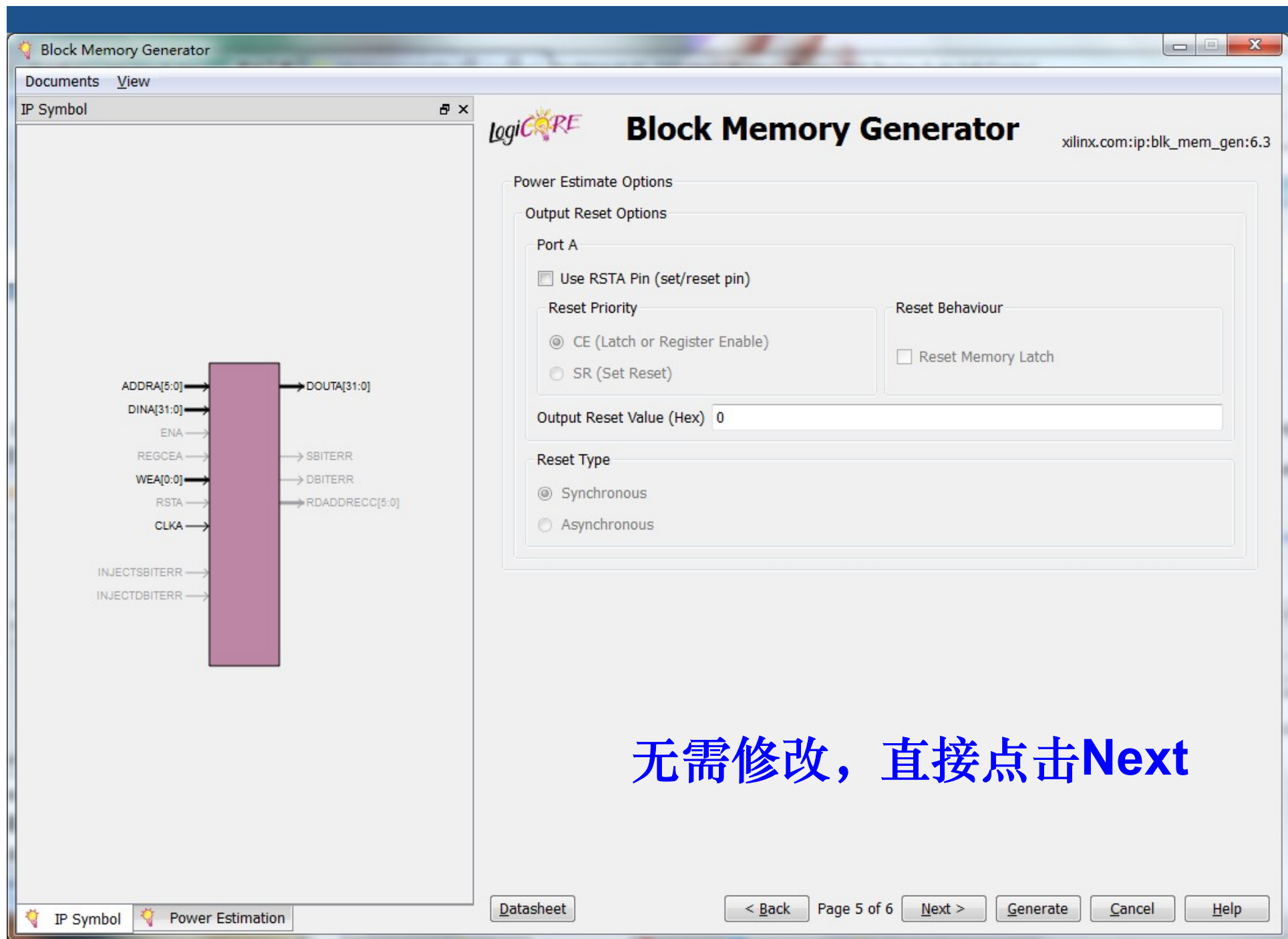
1.选中Load Init File选项

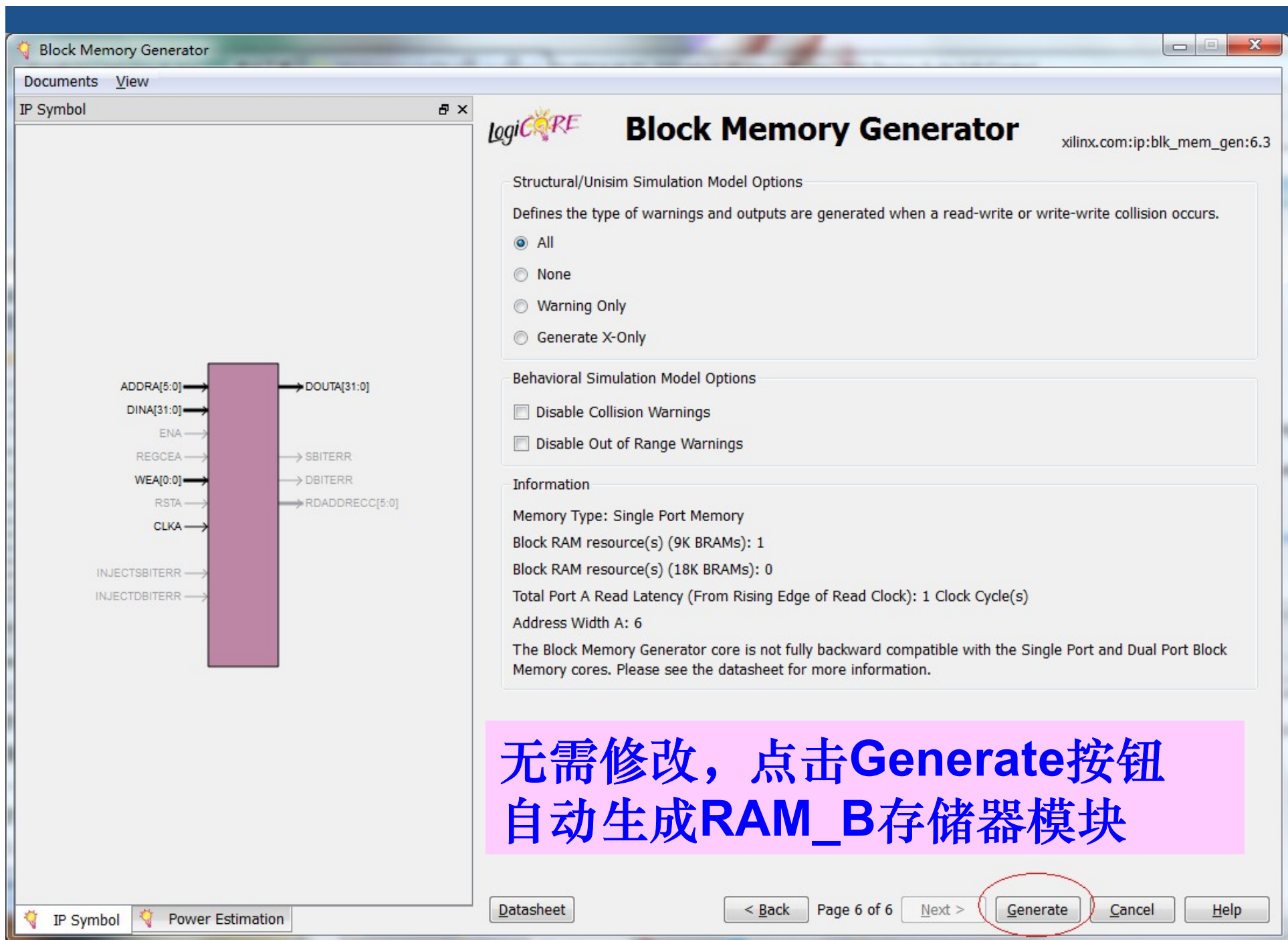
2.点击Browse按钮选择第一步生成的COE文档(Test_Mem.coe)作为关联初始化参数。

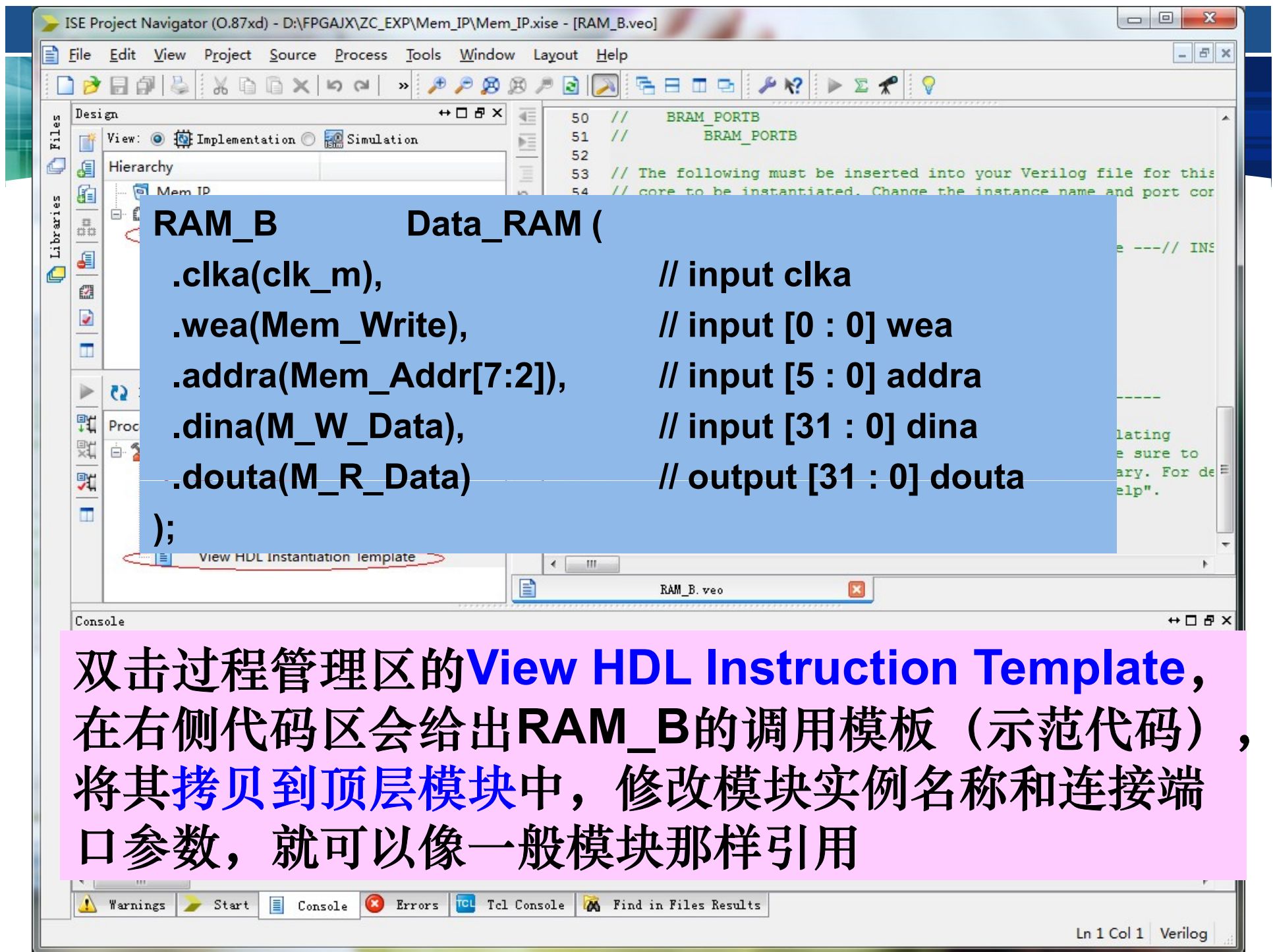
3.RAM_B IP生成后，ISE会用这个文档内的数据初始化RAM值。

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。









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
	Mem_Addr[7:2]	6个逻辑开关	读写存储器地址
输入 信号	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 $R_i \leftarrow (addr) \rightarrow R_j$ 的操作。
尝试编写代码，仿真调试通过。





实验六 MIPS汇编器与模拟器实验

❖ 1、实验目的

- 学习MIPS指令系统，熟悉**MIPS指令格式**及其**汇编指令助记符**，掌握**机器指令编码方法**；
- 学习MIPS汇编程序设计，学会使用MIPS汇编器**将汇编语言程序翻译成二进制文件**；
- 了解使用MIPS教学系统模拟器运行程序的方法；

实验六 MIPS汇编器与模拟器实验

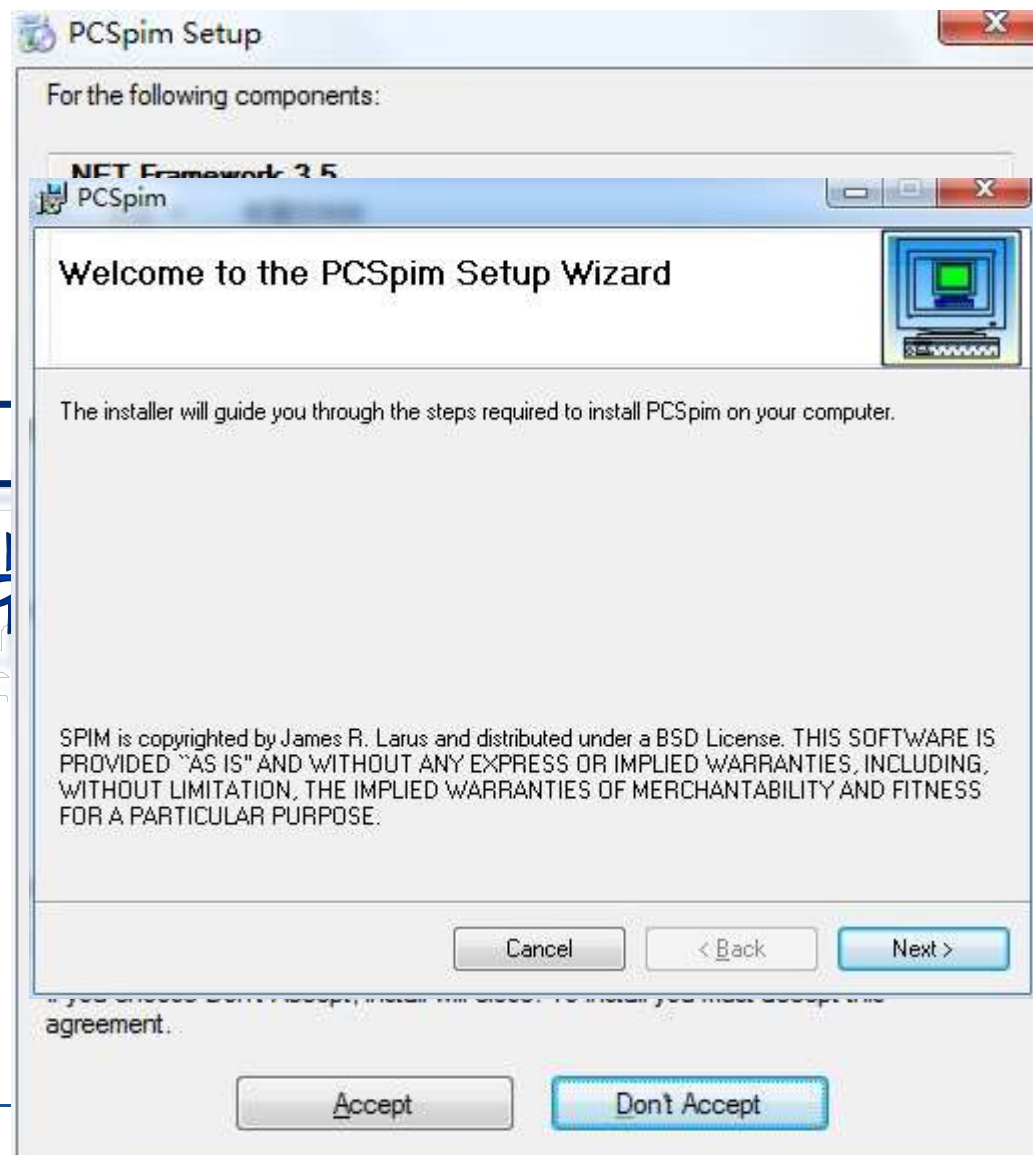
❖ 2、实验内容与原理

a. PCSpim下载及安装:

PCspim模拟器也有很多版本，可以在网上免费下载。注意：最新版本

Qtspim不能用于xp系统，下载时请查看网页上相关提示信息。下载地址：

<http://pages.cs.wisc.edu/~larus/spim.html>



❖ 2、实验

实验结果展示
在四棵树上
由树到树
值如左：
寄存器状
态窗口、
MIPS汇编
程序窗口、
数据区窗
口、信息
输入窗口。

The screenshot displays the PCSpim MIPS simulator interface. The main window shows the following information:

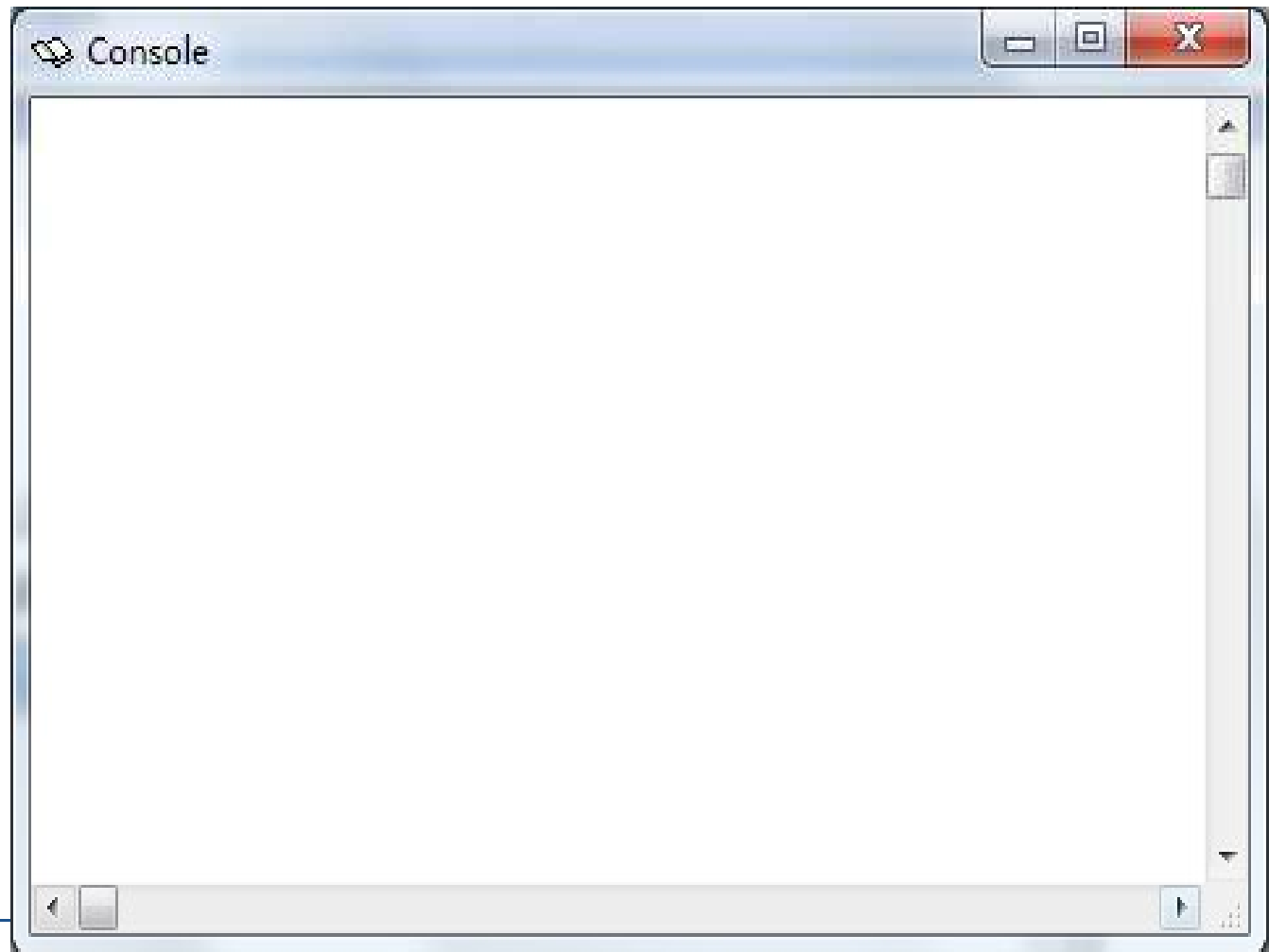
- Registers:**
 - PC = 00000000, EPC = 00000000, Cause = 00000000, BadVAddr = 00000000
 - Status = 3000ff10, HI = 00000000, LO = 00000000
 - General Registers: R0 (r0) to R28 (gp) are listed with their current values.
- Instructions:** A list of instructions is shown, including:
 - 0x00400000: 0x8fa40000 lw \$4, 0(\$29)
 - 0x00400004: 0x27a50004 addiu \$5, \$29, 4
 - 0x00400008: 0x24a60004 addiu \$6, \$5, 4
 - 0x0040000c: 0x00041080 sll \$2, \$4, 2
 - 0x00400010: 0x00c23021 addu \$6, \$6, \$2
 - 0x00400014: 0x00000000 jal 0x00000000 [main]
 - 0x00400018: 0x00000000 nop
 - 0x0040001c: 0x3402000a ori \$2, \$0, 10
- DATA:** A section showing memory addresses and values, including 0x10000000 to 0x10040000.
- STACK:** A section showing memory addresses and values, including 0x7ffff494 to 0x7fffffe1.
- KERNEL DATA:** A section showing memory addresses and values, including 0x90000000 to 0x90000000.
- SPIM Version:** SPIM Version 8.0 of January 8, 2010. Copyright 1990-2010, James R. Larus. All Rights Reserved.
- Message:** A message box at the bottom right displays the text: "For Help, press F1".

实验六 MIPS汇编器与模拟器实验

❖ 2、实验内容与原理

除此之外，还有一个独立的控制台窗口

Console，用于程序信息的输入、输出，如图



实验六 MIPS汇编器与模拟器实验

❖ 2、实验内容与原理

(1) 编辑程序

由于PCSpim没有自带的编辑器，用户可以选择自己喜欢的editor编辑文件，后缀名为.asm或者为.s。在这里我们用系统自带的文本编辑器，命名为test.asm。具体程序如下：

a. PCSpim应用实例：

读入两个整数，进行比较，并输出较大的数。

```
main: li $v0,5
```

```
syscall
```

```
move $t0, $v0
```

```
li $v0,5
```

```
syscall
```

```
move $t1, $v0
```

```
bgt $t0, $t1, t0_bigger
```

```
move $t2,$t1
```

```
b endif
```

```
t0_bigger: move $t2, $t0
```

```
endif: move $a0,$t2
```

```
li $v0,1
```

```
syscall
```

```
jr $ra;
```

PCSpim

File Simulator Window Help

Open... Ctrl+O

Save Log File... Ctrl+S

Exit Alt+F4

PC = 00000000 Cause = 00000000 BadVAddr= 00000000

= 00000000 LO = 00000000

General Registers

R0 (r0) = 00000000 R8 (t0) = 00000000 R16 (s0) = 00000000 R24 (t8) = 00000000

R1 (at) = 00000000 R9 (t1) = 00000000 R17 (s1) = 00000000 R25 (t9) = 00000000

R2 (v0) = 00000000 R10 (t2) = 00000000 R18 (s2) = 00000000 R26 (k0) = 00000000

R3 (v1) = 00000000 R11 (t3) = 00000000 R19 (s3) = 00000000 R27 (k1) = 00000000

[0x00400000] 0x8fa40000 lw \$4, 0(\$29) ; 183: lw \$a0 0(\$sp)

[0x00400004] 0x27a50004 addiu \$5, \$29, 4 ; 184: addiu \$a1 \$sp 4

[0x00400008] 0x24a60004 addiu \$6, \$5, 4 ; 185: addiu \$a2 \$a1 4

[0x0040000c] 0x00041080 sll \$2, \$4, 2 ; 186: sll \$v0 \$a0 2

[0x00400010] 0x00c23021 addu \$6, \$6, \$2 ; 187: addu \$a2 \$a2 \$v0

[0x00400014] 0x0c100009 jal 0x00400024 [main] ; 188: jal main

[0x00400018] 0x00000000 nop ; 189: nop

DATA

[0x10000000]...[0x10040000] 0x00000000

STACK

[0x7ffff494] 0x00000000 0x00000000 0x7fffffe1

KERNEL DATA

Copyright 1997, Morgan Kaufmann Publishers, Inc.

See the file README for a full copyright notice.

Loaded: C:\Program Files (x86)\PCSpim\exceptions.s

C:\Users\Administrator\Desktop\test.asm successfully loaded

Attempt to execute non-instruction at 0x00400060

Attempt to execute non-instruction at 0x00400060


For Help, press F1

PC=0x00000000 EPC=0x00000000 Cause=0x00000000

DELAY

实验六 MIPS汇编器与模拟器实验

❖ 6.6.3 实验要求

- 按照上述实验内容中的例子，编辑**test.asm**文件，在**PCSpim**中打开它，并使用单步执行和连续执行方式运行该程序；
- 将下列程序输入一个**R_CPU_Test.asm**文件，并在**PCSpim**中单步运行，观察各个寄存器的值，是否和预期的一致； 
- 将上述程序的指令代码逐条摘录出来，拷贝至**ROM IP**核关联文件*.coe中，以备后续实验使用；
- 撰写实验报告：含执行结果截图、实验结果记录表、实验分析和生成的*.coe文件内容，以及你对本实验的“思考与探索”部分所作的思考与探索；

实验六 MIPS汇编器与模拟器实验

```
nor $1, $0, $0;    #$1=FFFF_FFFF
sltu $2, $0, $1;    #$2=0000_0001
add $3, $2, $2;     #$3=0000_0002
add $4, $3, $2;     #$4=0000_0003
add $5, $4, $3;     #$5=0000_0005
add $6, $5, $3;     #$6=0000_0007
sllv $7, $6, $2;     #$7=0000_000E
add $9, $5, $6;     #$9=0000_000C
sllv $8, $6, $9;     #$8=0000_7000
xor $9, $1, $8;     #$9=FFFF_8FFF
add $10, $9, $1;    #$10=FFFF_8FFE
sub $11, $8, $7;    #$11=0000_6FF2
sub $12, $7, $8;    #$12=FFFF_900E
and $13, $9, $12;   #$13=FFFF_800E
or $14, $9, $12;    #$14=FFFF_9FFF
or $15, $6, $7;     #$15=0000_000F
```

```
nor $16, $6, $7;    #$16=FFFF_FFF0
add $17, $7, $3;    #$17=0000_0010
sllv $18, $8, $17;   #$18=7000_0000
sllv $19, $3, $17;   #$19=0002_0000
sllv $20, $19, $7;   #$20=8000_0000
add $21, $20, $1;    #$21=7FFF_FFFF
or $22, $18, $21;    #$22=7FFF_FFFF
add $23, $20, $22;   #$23=FFFF_FFFF
sub $24, $20, $22;   #$24=0000_0001
sub $25, $22, $20;   #$25=FFFF_FFFF
xor $26, $18, $1;    #$26=8FFF_FFFF
sltu $27, $22, $20;  #$27=0000_0001
sltu $28, $26, $20;  #$28=0000_0000
add $29, $22, $2;    #$29=8000_0000
sub $30, $20, $2;    #$30=7FFF_FFFF
add $31, $11, $26;   #$31=9000_6FF1
```



实验六 MIPS汇编器与模拟器实验

❖ 6.6.4 实验步骤

- (1) 使用记事本程序或任何纯文本编辑器，编辑**test.asm**文件，输入前述内容；
- (2) 运行**PCSpim**程序，在其中打开**test.asm**，先连续执行，输入起始地址**0x0040 0000**，再单步运行，按照需要在控制台输入2个数据，执行完毕，观察结果。
- (3) 使用记事本程序或任何纯文本编辑器，编辑**R_CPU_Test.asm**文件，输入规定指令。
- (4) 在**PCSpim**程序中打开**R_CPU_Test.asm**，同上单步执行，记录执行结果。

实验六 MIPS汇编器与模拟器实验

❖ 6.6.5 思考与探索：必做（1）

- (1) 将R_CPU_Test.asm汇编程序执行的结果记录到表6.13中，分析你的实验结果是否正确；如果不正确，请分析原因；
- (2) 谈谈你在实验中，碰到了什么问题？又是怎么解决的？
- (3) PCSpim能模拟执行MIPS的汇编程序。还有其他真正意义上的MIPS汇编器，将汇编源程序直接翻译成一个只含机器码（指令编码）的代码文件，甚至还有将机器码直接自动生成*.coe的小工具，查找相关资料，把你的新发现和大家分享，写在实验报告里。



The End!