兰州大学信息科学与工程学院
计算机组成原理课程设计实验报告

一、 实验目的

1. 了解只读存储器 ROM 和随机存取存储 RAM 的原理
2. 理解 ROM 读取数据及 RAM 读取、写入数据的过程
3. 理解计算机中存储器地址编址和数据索引方法
4. 理解同步 RAM 和异步 RAM 的区别
5. 掌握调用 Xilinx 库 IP 实例化 RAM 的设计方法
6. 熟悉并运用 Verilog 语言进行电路设计
7. 为后续 CPU 设计实验打下基础

二、 实验器材与设备

2.1 装有 Xilinx vivado 的计算机一台

2.2 LS-CPU-EXB-002 教学系统实验箱一套

三、 实验分析与设计

3.1 实验原理

1. 只读存储器 ROM 和随机存取存储器 RAM 的原理

ROM 以非破坏性读出的方式工作，只能读出无法写入信息，即使断电也不会影响存储的信息丢失，又称为固定存储器。常用于存储各种固定程序和数据，在本次实验中以 ROM 作为 CPU 指令的存储器 instr_ROM。ROM 主要由存储矩阵、地址译码器和输出缓冲器三个部分组成。ROM 的地址译码器是与

门的结合，由两个阵列组成与门阵列和或门阵列。ROM 不需要读取控制线，在任何给定时间，输出线会自动提供又地址选择的 N 位。

而 RAM 也称主存，是于 CPU 直接进行数据交换的内部存储器，可以随时读写，速度较快，通常作为临时数据存储介质，可以从任何指定的地址中写入或读出信息，但是其存储的数据具有易失性，一旦断电则存储的数据将随之丢失。随机存取的含义是数据被读取或写入时，所需的时间与数据所在的位置或写入的位置无关。RAM 由存储矩阵、地址译码器、读写控制器、I/O、片选控制等几部分组成。

2. ROM 读取及 RAM 读取和写入数据的过程原理

ROM 读取数据的时候，只要输入指定的地址码并将使能，则地址指定的存储单元所存储的数据便被读取。其原理是在二极管存储矩阵的阵列中通过地址的行检索和并行的列检索查询到数据并读出。

RAM 也是通过寻址在存储阵列中进行行列的寻址，在寻址之前需要进行片选和定址，但是这两个工作可以有效的同时进行，通过 WE 信号的控制来达到区分读写的目的。

3.2 端口设计

1. ROM ： input [31:0] addr ; output [31:0] inst
2. RAM ： input [31:0] addr ; input [31:0] wdata ; input [31:0]

test_addr ; output [31:0] raddr ; output [31:0] test_data

3. 上板后的拨码

SW18-SW19 ： 01 TADDR

SW18-SW19 ： 10 WDATA

SW18-SW19 ： 11 ADDR

SW22 : 1 写使能

3. 3 设计框图

1. 同步 RAM

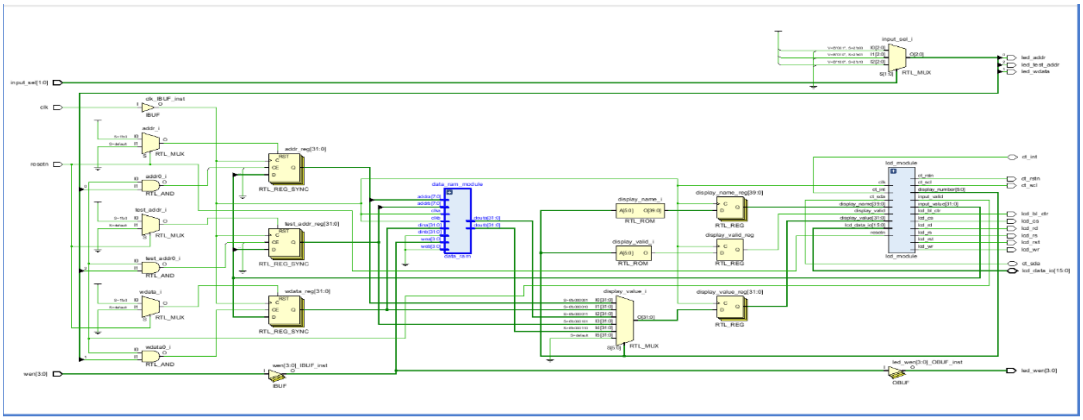


图 3 使用 IP 核同步 RAM 设计框图

2. 同步 ROM

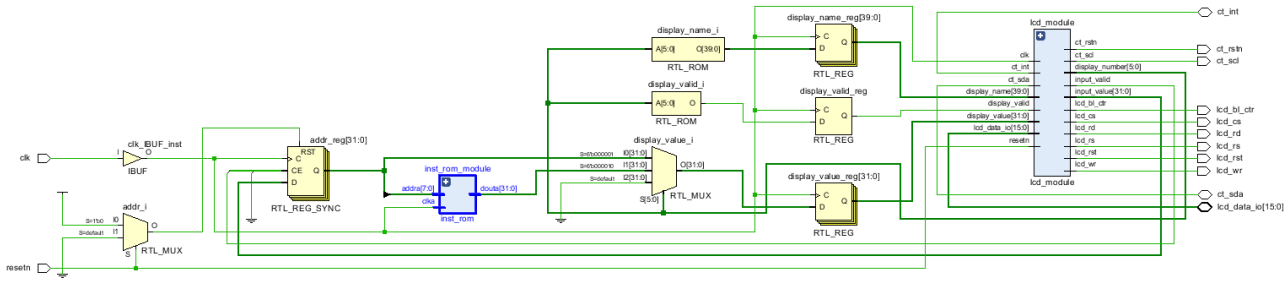


图 4 使用 IP 核同步 ROM 设计框图

3. 异步 RAM

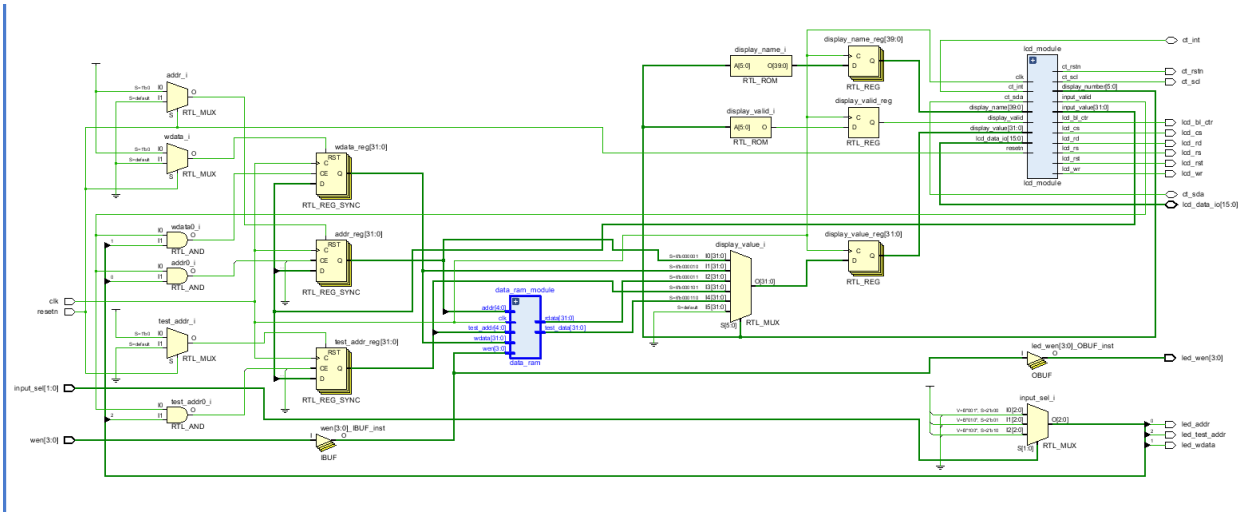


图 5 异步 RAM 设计框图

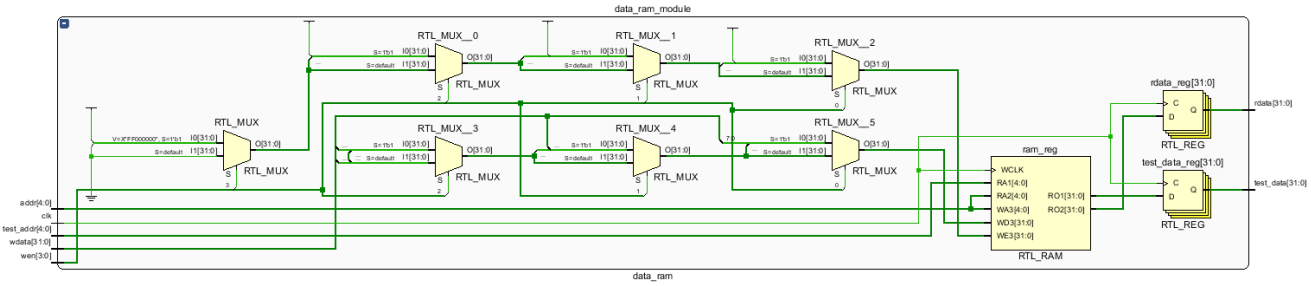


图 6 data_ram 的内部设计框图

4. 异步 ROM

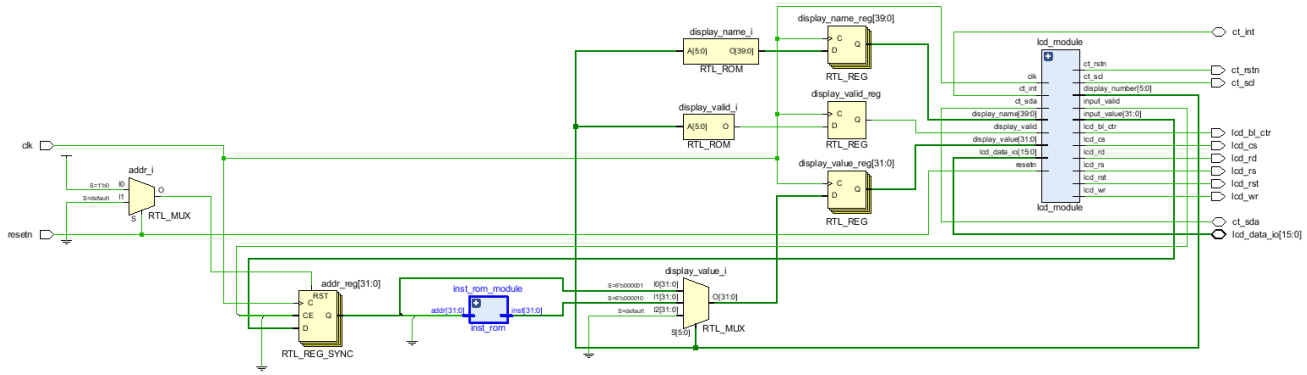


图 7 异步 ROM 设计框图

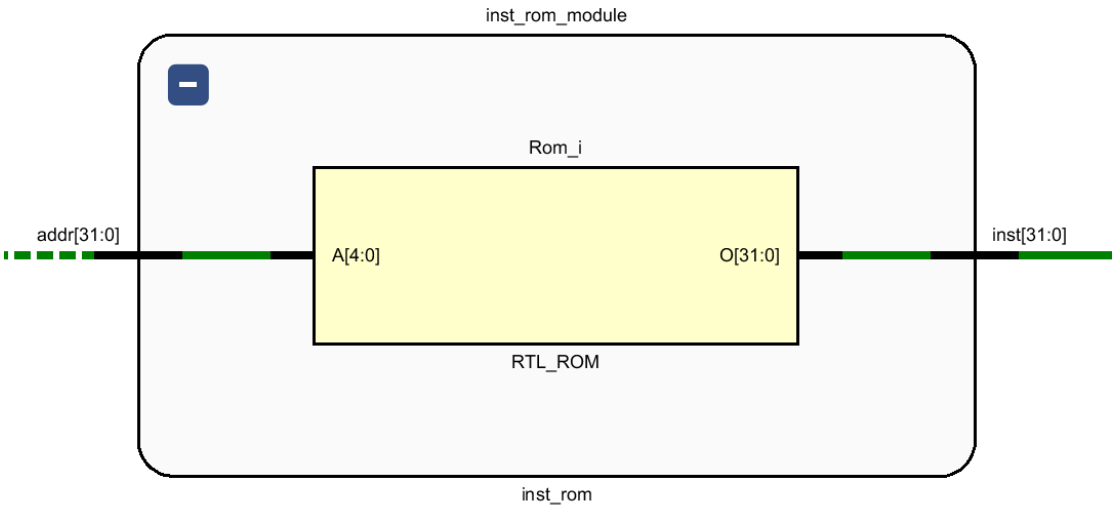


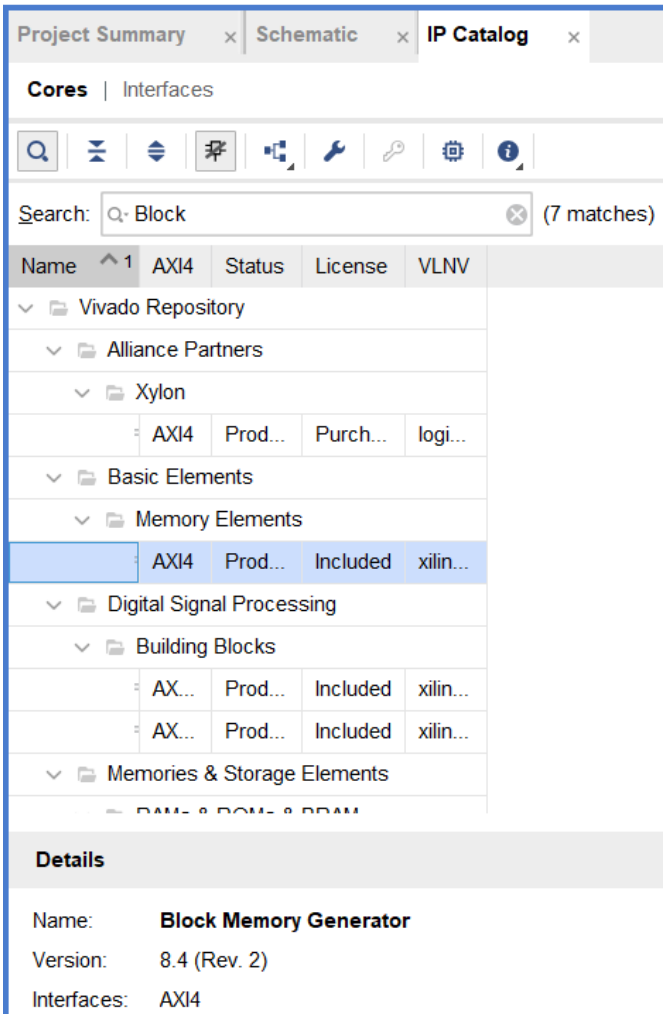
图 8 inst_rom 的内部设计框图

四、 实验步骤

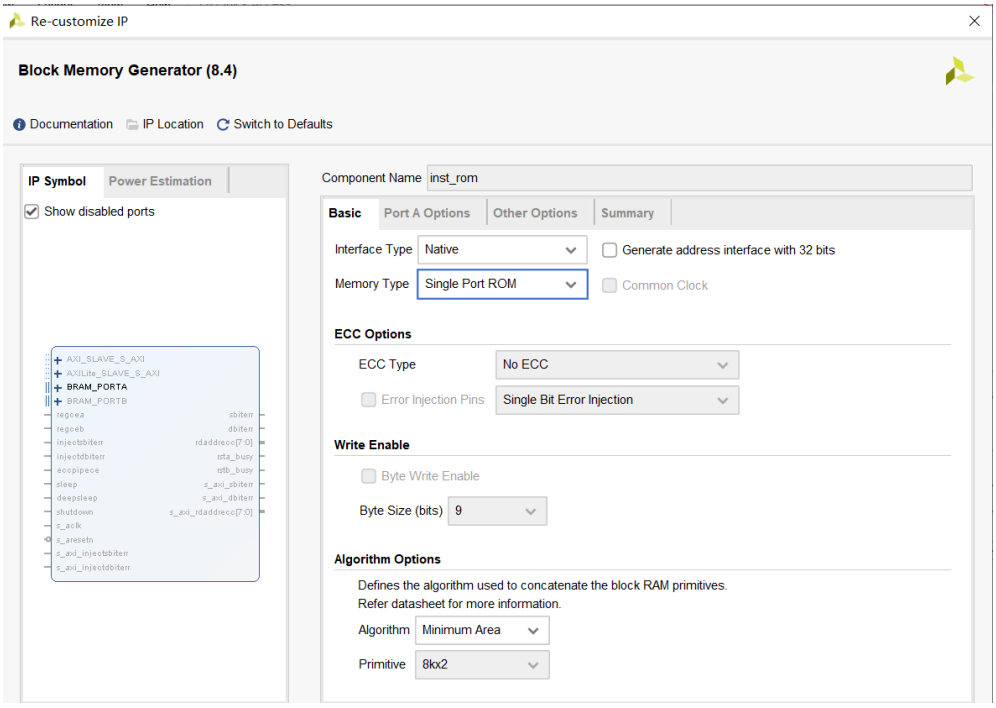
4. 1 代码实现

1. 使用 IP 核实例化同步 ROM

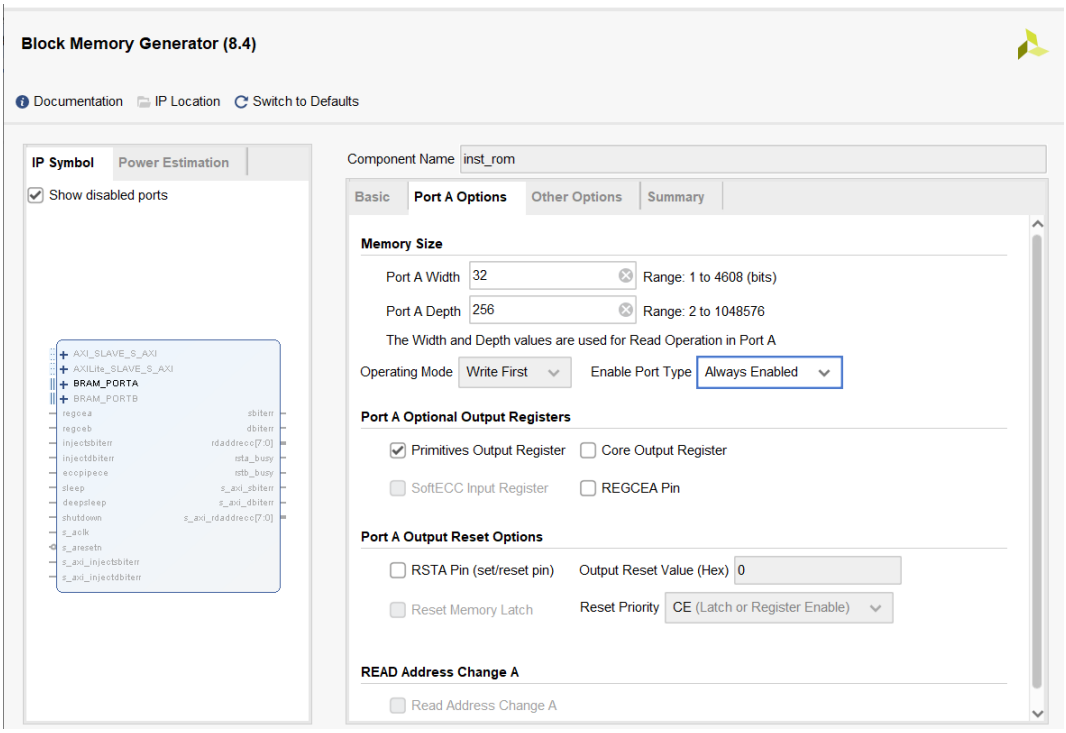
a. 打开 vivado 中的 IP catalog 并在输入框中输入 Block Memory Generator 选中



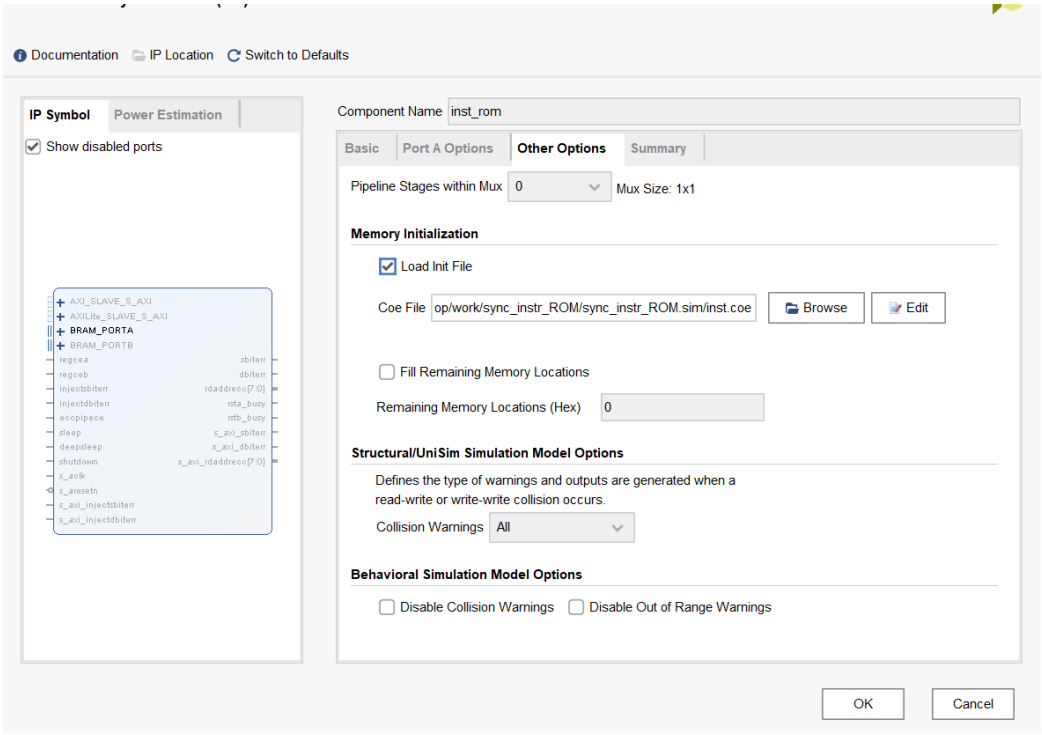
b. 双击后选择单端口 Single Port ROM



c. 并设置 Port A 的深度和宽度，并将使能信号 Enable Port Type 设置为 Always Enabled



d. 点击 Other Options 选项卡勾选 Load Init File 将初始化文件.coe 导入为 ROM 的初始文件，点击 generate 生成 ROM

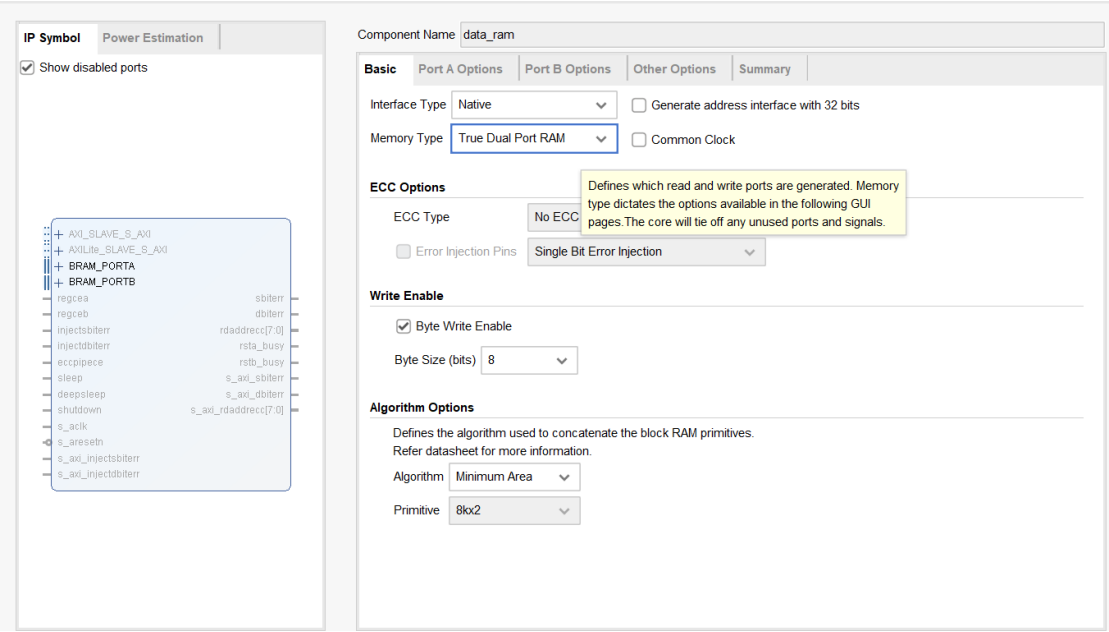


e. 将相应的 inst_rom_display.v 文件、lcd_module 文件、.xdc 管脚约束文件添加入工程后便可以进行文件的烧录

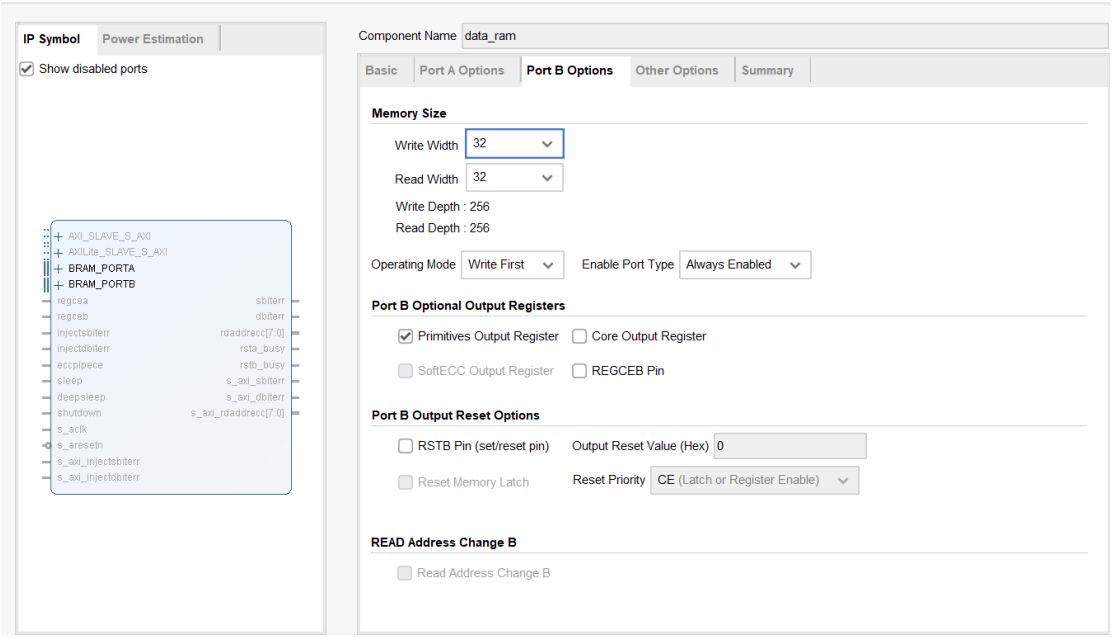
2. 使用 IP 核实例化同步 RAM

a. 第一步同上述 ROM 的第一步操作

b. 在 memory type 中选择 True Dual Port RAM，两个端口一个作为正常的读写端口，一个作为调试的读端口；勾选 Write Enable，并在 byte size 中填写 8bits,激活写使能



c. 在 Port A Options 中设置读写的深度和宽度，深度为 256，宽度为 32 位，后续 CPU 实验是基于 32 位数据进行运算的，所以这里设置宽度为 32 位；Enable Port Type 设置为 Always Enable，对于 Port B Options 也做同样的设置，然后点击 generate 生成对应的 RAM



d. 将相应的 data_ram_display.v 文件、lcd_module 文件、.xdc 管脚约束文件添加入工程后便可以进行文件的烧录

3. 使用 Verilog 语言编写异步 ROM

基本思路：指令的条数不限，这里出于方便同样提供了 32 个 32 位的寄存器来保存指令。首先使用 assign 语句将指令分别存入到二维 reg 中，由于不需要时钟控制，所以只要有地址来，那么就将数据读出到 inst 中。

4. 使用 Verilog 语言编写异步 RAM

基本思路：通过四位的字节写使能控制数据的写入，使用二维的 reg 模拟数据存储器，读地址和测试地址给定则给出该地址存储的数据，由 rdata 和 test_data 带出。而由于实际 MIPS 指令中地址位为 5 位，所以这里实际读的地址只有 5 位，也就是说存储器中仅需要 32 个寄存器即可。

在这里分别给出编写的异步 ROM 和异步 RAM 的代码：

```
    assign Rom[5'h09]=32'h1022000A;// beq $1,$2,10
    assign Rom[5'h0A]=32'h0800000D;// J 0D
    assign Rom[5'h0B]=32'hXXXXXXXX;
    assign Rom[5'h0C]=32'hXXXXXXXX;
    assign Rom[5'h0D]=32'hAD02000A;// sw $1 10($8) memory[$8+10]=10
    assign Rom[5'h0E]=32'h8D04000A;//lw $1 10($8) $4=12
    assign Rom[5'h0F]=32'h00521826;//xor $1,$2,$3   $1=$2^$3
    assign Rom[5'h10]=32'h00021900;//sll $1,$2,4
    assign Rom[5'h11]=32'h00021902;//srl $1,$2,4
    assign Rom[5'h12]=32'h00021903;//sra $1,$2,4
    assign Rom[5'h13]=32'h3047000A;//andi $1,$2,A
    assign Rom[5'h14]=32'h382300EF;//xori $1,$2,0xef
    assign Rom[5'h15]=32'h3C011234;//lui $1,0x1234
    assign Rom[5'h16]=32'h0C00001A;//Jal 1A
    assign Rom[5'h17]=32'h0800001A;// J 1A
    assign Rom[5'h18]=32'hXXXXXXXX;
    assign Rom[5'h19]=32'hXXXXXXXX;
    assign Rom[5'h1A]=32'h03E00008;//Jr 16
    assign Rom[5'h1B]=32'hXXXXXXXX;
    assign Rom[5'h1C]=32'hXXXXXXXX;
    assign Rom[5'h1D]=32'hXXXXXXXX;
    assign Rom[5'h1E]=32'hXXXXXXXX;
    assign Rom[5'h1F]=32'hXXXXXXXX;

    always @(*)
    begin
        inst<= Rom[addr];
    end
endmodule
```

图 9 异步 ROM 的实现代码(部分)

```
module data_ram(clk,wen,addr,wdata,rdata,test_addr,test_data);
    input clk;
    input [3:0] wen;
    input [4:0] addr,test_addr;
    input [31:0] wdata;
    output reg [31:0] rdata,test_data;
    reg [31:0] ram [0:255];//宽度为32位，深度为256位和同步RAM保持一致
    integer i;
    initial begin
        for(i=0;i<256;i=i+1)
            ram[i]=0;//初始化
    end
    always@(posedge clk) begin
        if(wen[3])ram[addr][31:24]<=wdata[31:24];//写信号来临才写
        if(wen[2])ram[addr][23:16]<=wdata[23:16];
        if(wen[1])ram[addr][15:8]<=wdata[15:8];
        if(wen[0])ram[addr][7:0]<=wdata[7:0];
        rdata <= ram[addr];//读信号来临才读
        test_data <= ram[test_addr];//测试端口
    end
endmodule
```

图 10 异步 RAM 的实现代码

4.2 仿真与综合

编写完代码后就可以进行 simulation 和 implement 了

1. 首先测试 ROM

这是编写的 tb 文件：

```
module tb;
    reg  [31:0] addr;
    wire [31:0] inst;
    //实例化rom
    inst_rom inst_rom_module(
        .addr  (addr),
        .inst  (inst      )
    );
    integer i;
    initial begin
        addr = 0;
        //将32个寄存器中的指令都打印
        for(i=0;i<32;i=i+1) begin
            #20;
            addr = addr + 5'd1;
        end
    end
endmodule
```

通过 for 循环输出每个寄存器堆中的 32 位指令，在 simulation 的仿真波形中也能够看到其正确输出了指令

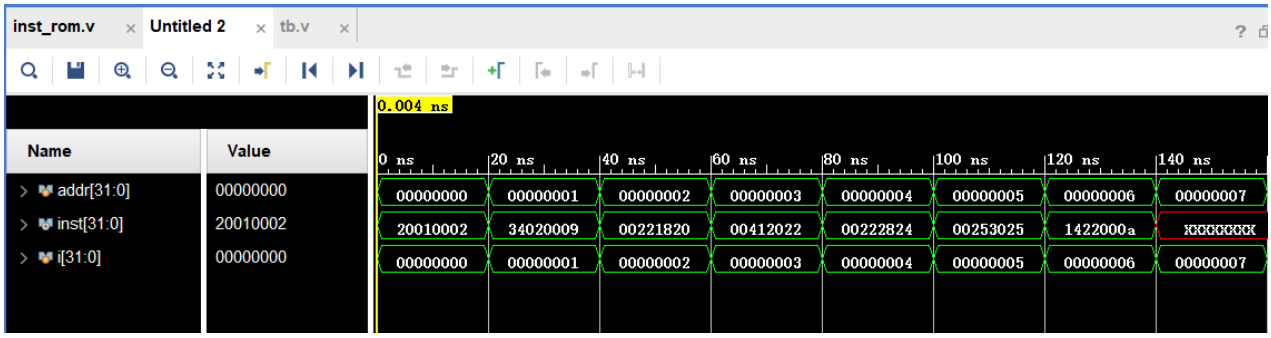


图 11 ROM 的仿真波形图

2. RAM 的仿真测试

首先编写 RAM 的 tb 文件

```
data_ram data_ram_module(
    .clk   (clk   ),
    .wen   (wen   ),
    .addr  (addr),
    .wdata (wdata ),
    .rdata (rdata ),
    .test_addr(test_addr),
    .test_data(test_data)
);
always #20 clk = ~clk;
initial begin
clk = 0;
# 20;
wen = 4'b1111;
addr = 1;
wdata = 10;
# 20;
wen = 4'b0000;
addr = 1;
test_addr = 1;
# 20;
addr = 2;
# 20
addr = 1;
end
```

先将写使能置为 1111，然后将数据 10 写入 1 号寄存器；再将写使能置为 0，使用 addr 和 test_addr 来读取 1 号和 2 号寄存器中的值

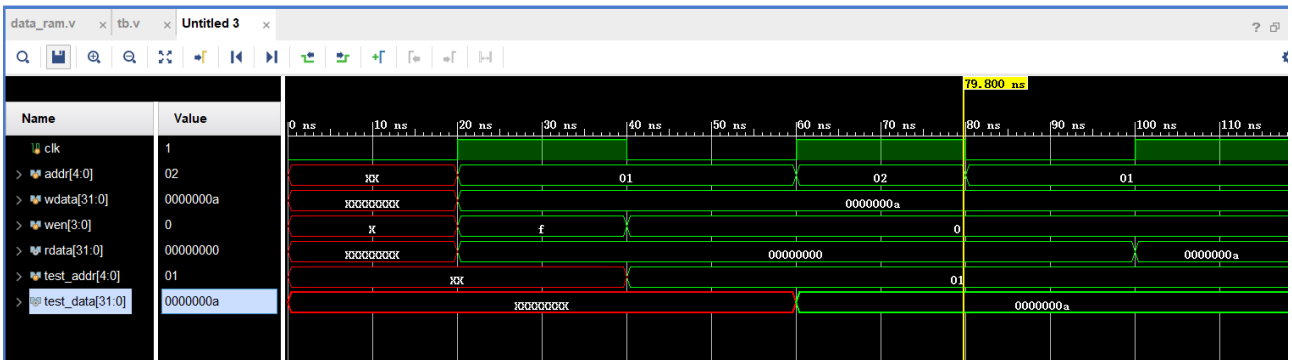


图 11 RAM 的仿真波形图

可以看到 10 正确的写入了 1 号寄存器并被 addr 和 test_addr 正确读出

4.3 上板验证与结果展示

五、 问题回答

六、 收获与体会

1. **理解了 ROM 和 RAM 的使用场景的不同和差异**
2. **理解了同步和异步 RAM 的差别**
3. **尝试调用 Xilinx 库中的 IP 核进行 RAM 的例化，体会到 IP 核的高效与简便，可以很快的实现一些基本的器件，加快开发的时间**
4. **熟悉并运用了 Verilog 语言进行电路进行设计**
5. **掌握了几种实现寄存器堆的方法并了解了相联寄存器的工作原理**