《数字逻辑电路》实验报告

第 11 次实验：存储器实验

1. 实验目的

本次实验的目的包括：

1. 数字电路设计：学习存储器的设计方法；
2. FPGA 上的数字设计：学习使用不同的方法设计 FPGA 片上存储器，学习使用片内 M4K 存储器；

2. 实验原理（背景知识）

DE2-70 平台使用的是 Cyclone II EP2C70F896C6 FPGA。EP2C70 系列 FPGA 片内含有多列 M4K 存储器。每一个 M4K 存储器块含有 4608bit（4096 个数据位和 512 个奇偶校验位）。M4K 存储器块内含有同步写入的输入寄存器和输出寄存器用于流水线设计和提供系统性能。输出寄存器可以旁路，但输入寄存器不行。每个 M4K 块可以有不同的配置方法，包括真双口 RAM、简单双口 RAM、单口 RAM、ROM 或者 FIFO 缓存。

为了能够使 Quartus II 编译器自动识别出设计的是存储器而使用片内 M4K 存储器块，需要使用标准 Verilog HDL 风格进行设计。在 Quartus II 中可以使用提供的模板来实现。

Verilog HDL 提供了一种用于模拟的描述语句：initial 语句块。该语句块的功能是在模拟的 0 时刻，该语句块被执行一次，以后再也不执行。利用 initial 语句可以在模拟的开始时刻将所需要的信号、存储单元初始化为所需要的值。initial 语句也常用于测试平台的设计中。因为模拟器会执行其中的语句一次且仅一次，所以如果在 initail 语句块中插入测试代码，就能利用模拟器模拟电路在不同输入下的工作情况。因此 initial 语句也常用于测试平台。需要注意的是 initial 一般只用于模拟测试，在实际的电路中是不可综合的。

但在 FPGA 设计中，可以使用 mif 文件对存储器单元进行初始化。根据 Altera 官方说明：

Memory Initialization File (.mif)

An ASCII text file (with the extension .mif) that specifies the initial content of a memory block (CAM, RAM, or ROM), that is, the initial values for each address. This file is used during project compilation and/or simulation.

利用 initial 语句对 RAM/ROM 设计进行初始化时，Quartus II 编译器会利用类似 mif 文件的机制，对 FPGA 芯片中的 M4K 存储单元进行初始化。因此在 FPGA 设计中，储存单元的初始化虽然不能综合成电路，但是能够在电路系统中实现。

3. 实验器材/环境

1) 软件环境：

设计\编译环境：Quartus II 9.0 Build 132 SJ Full Version --Windows Vista

实验室下载环境：Quartus II 9.0 Build 132 SJ Full Version – Windows XP.

2) 硬件环境：

DE2-70 开发平台;

FPGA 芯片： Cyclone II EP2C70F896C6.

4. 实验设计思路（验收实验）

A.设计目标、模型描述

用 Verilog HDL 实现一个具有 32*8bit 容量的 RAM。能够通过地址来访问、修改 RAM 中相应单元。先向

RAM 输入特定的数据，然后能够通过 LED 或者 HEX 作为输出显示端，逐个读出显示。RAM 具有初始化的功能，在电路刚开始运行时，所有单元均可以被初始化为一个指定的值。

B.数字抽象

RAM 具有 3 项输入、1 项输出。

1. 输入：
   - 地址 addr（5bit）：用于指定访问的 RAM 单元的地址；
   - 数据输入 data（8bit）：用于修改指定地址中保存的内容；
   - 写入使能 we：当写使能有效时，写入数据，当其无效时，读出数据；
   - 时钟信号 clk；
2. 输出：
   - 数据输出 q：用于输出指定地址单元中存放的数据；

表格 1:输入输出信号与DE2-70平台信号对应关系表

信号名称	*DE2-70平台信号*
addr	iSW[12:8]
data	iSW[7:0]
we	iSW[13]
clk	iCLK_50
q	oLEDG[7:0]

C.建立模型

在 Verilog HDL 中可以用多维数组定义存储器。一个 32 字节存储器块可以定义为 32*8 的数组，在 Verilog HDL 中可以使用如下的语句：

Reg[7:0] memory_array[31:0];

在 Verilog-1995 标准中这样定义后就不能再对存储块中的一位进行单独操作，每次操作需要以 8 位为一组进行操作。但在 Verilog-2001 标准中，允许对数组的每一位进行单独操作。在 Cyclone Ii 系列 FPGA 中，这组数组可以由触发器来实现也可以由 M4K 存储器块实现。为了保证使用 M4K 存储器块实现 RAM，必须采用满足 Verilog HDL 代码风格的语言形式来定义 RAM，Quartus II 软件在进行编译时会自动推断出该代码描述的是一个 RAM 块，从而使用 M4K 实现。

下面利用真值表的方式定义输入与输出之间的关系。使用 M[addr]表示 addr 地址所对应的存储器单元中的内容。存储器按照字节编址。

表格 2:存储器行为表

addr	data	we	clk	M[addr]	q
X	X	1	↑	data	data
X	X	0	↑	M[addr]	M[addr]

这样定义的存储器使 M4K 存储器块工作在单口 RAM 模式下。

为了方便阅读输出的内容，q[7:0]分别送往两个扩展 BCD 译码器译码，使其转变为 16 进制输出。使用两个七段译码器(oHEX0_D,oHEX1_D)显示。

对于存储单元内容的初始化问题，使用 initial 语句对 ram 单元初始化。初始化的值使用 INI_VAL 来表示。

存储器的 Verilog HDL 实现如下。

single_port_ram.v

```
// Quartus II Verilog Template
// Single port RAM with single read/write address

module single_port_ram
#(parameter DATA_WIDTH=8, parameter ADDR_WIDTH=5, parameter INI_VAL=3)
(
    input [(DATA_WIDTH-1):0] data,
    input [(ADDR_WIDTH-1):0] addr,
    input we, clk,
    output [(DATA_WIDTH-1):0] q
);

    // Declare the RAM variable
    reg [DATA_WIDTH-1:0] ram[2**ADDR_WIDTH-1:0];

    // Variable to hold the registered read address
    reg [ADDR_WIDTH-1:0] addr_reg;


    integer i;


    initial
        begin:ini
        for (i=0;i<2**ADDR_WIDTH;i=i+1)
            ram[i] = INI_VAL;
        end


    always @ (posedge clk)
    begin
        // Write
        if (we)
            ram[addr] <= data;

        addr_reg <= addr;
    end

    // Continuous assignment implies read returns NEW data.
    // This is the natural behavior of the TriMatrix memory
    // blocks in Single Port mode.
    assign q = ram[addr_reg];

endmodule
```

扩展 16 进制 BCD 译码器的 Verilog HDL 描述如下。

bcd7seg.v

```
    module bcd7seg(bcd,En,H);
```

```
    // Input Port(s)
    input [3:0] bcd;
    input En;

    // Output Port(s)
    output [6:0] H;
    reg [6:0] H;

    // Additional Module Item(s)

    //Output Logic
    always @ (bcd,En)
    begin
    if (En)
        case (bcd)
        0:H = 7'b1000000;
        1:H = 7'b1111001;
        2:H = 7'b0100100;
        3:H = 7'b0110000;
        4:H = 7'b0011001;
        5:H = 7'b0010010;
        6:H = 7'b0000010;
        7:H = 7'b1111000;
        8:H = 7'b0000000;
        9:H = 7'b0010000;
        10:H = 7'b0001000;//A
        11:H = 7'b0000011;//b
        12:H = 7'b1000110;//C
        13:H = 7'b0100001;//d
        14:H = 7'b0010110;//E
        15:H = 7'b0001110;//F
        default: H = 7'b1111111;
        endcase
    else
        H = 7'b1111111;
    end

endmodule
```

RAM，译码器与 DE2-70 平台信号之间连接的原理图见下图。

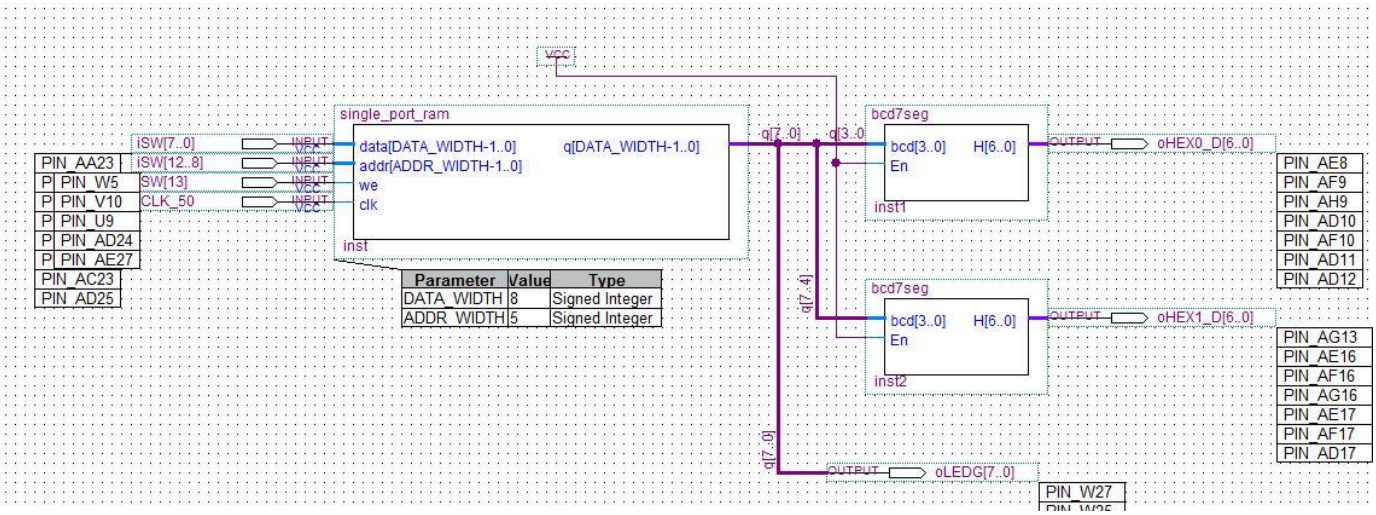


图 1: 存储器、显示模块与 DE2-70 平台信号的连接原理图

5. 实验过程（验收实验过程）

A.利用模板设计存储器

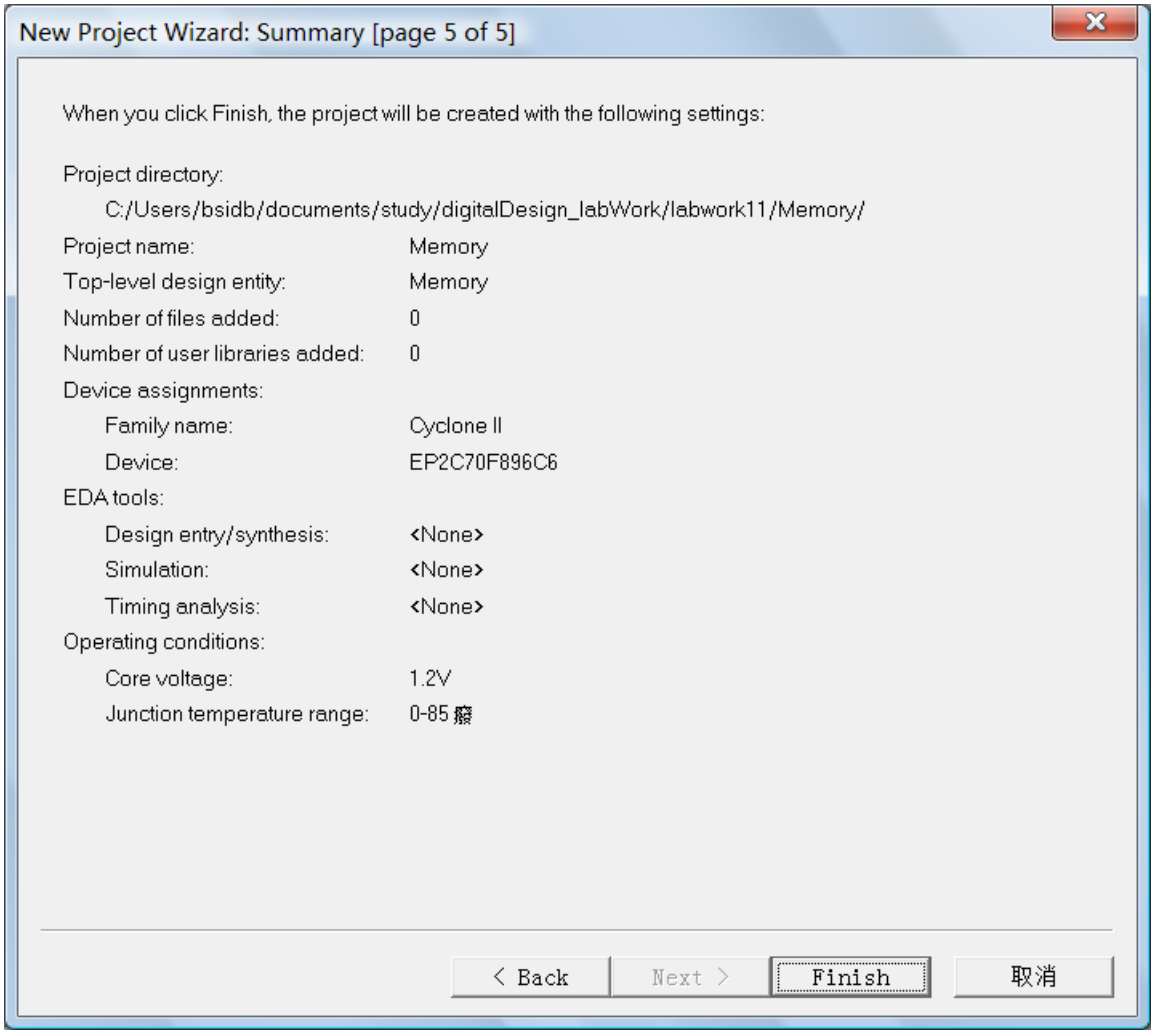


图 2：工程新建小结

利用模板编写 RAM 模块。

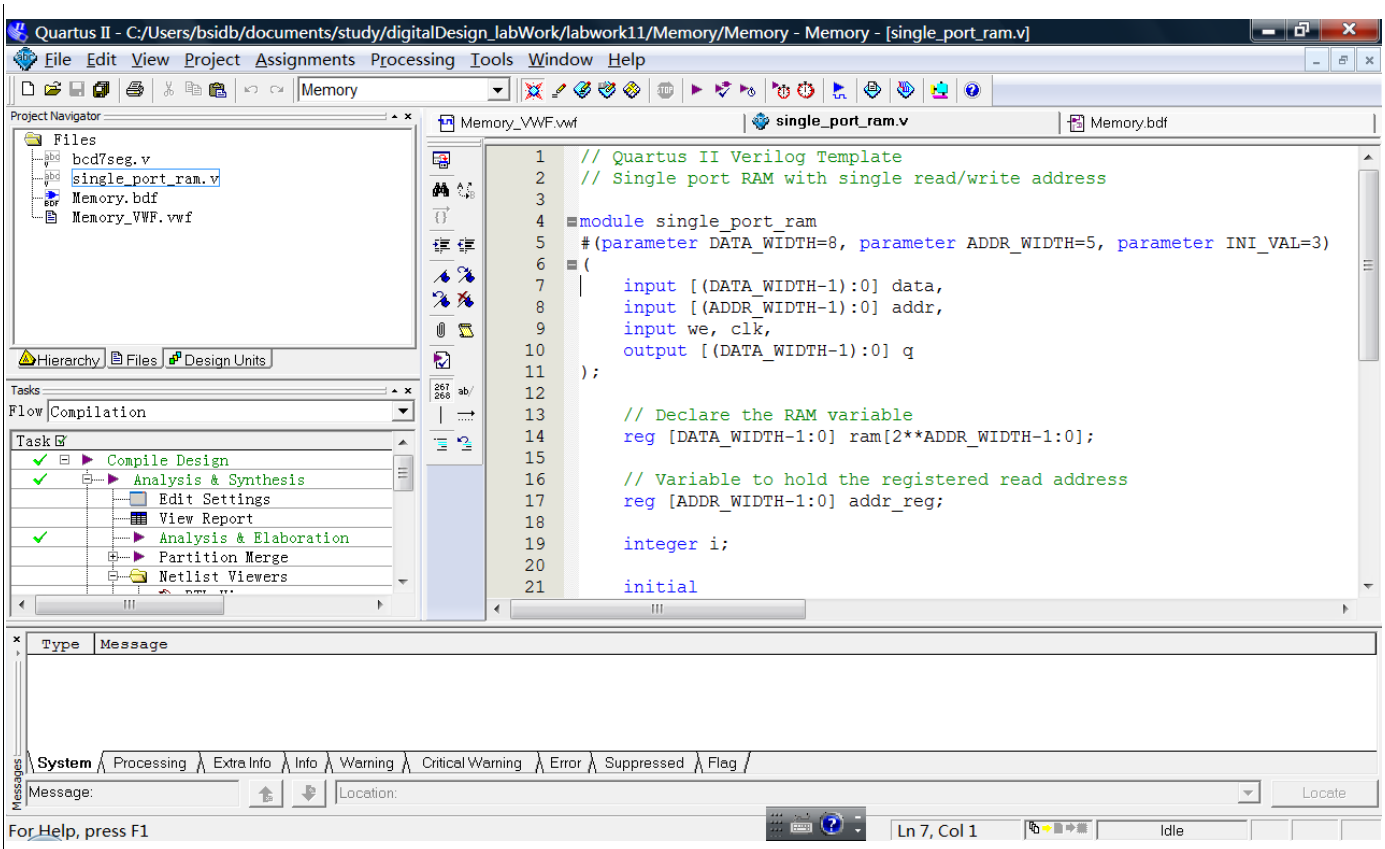


图 3: 编写 RAM 模块

编写译码器显示模块。

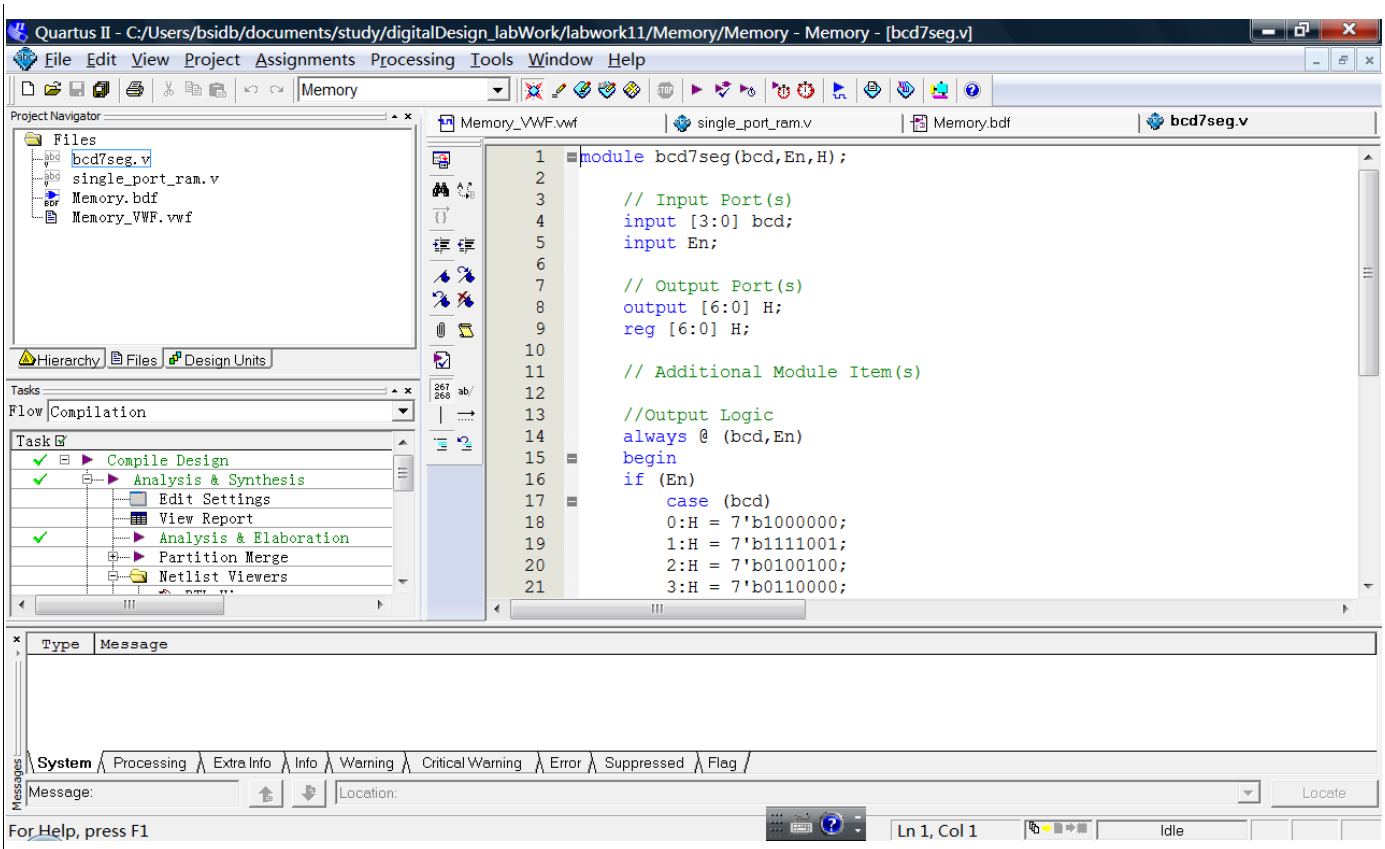


图 4: 编写译码器显示模块

将模块与 DE2-70 平台信号向连接。

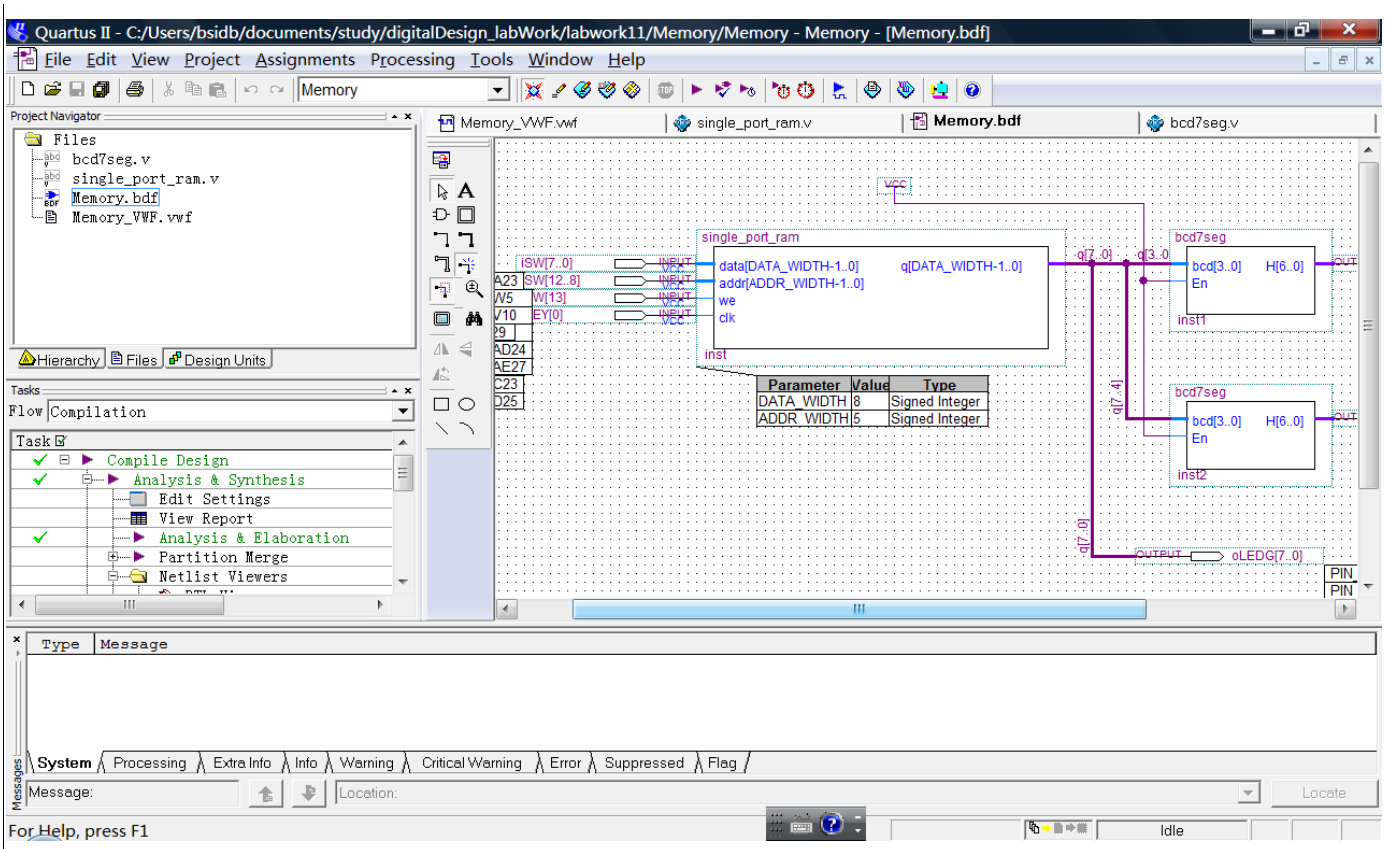


图 5：模块与 DE2-70 平台信号连接

分析/综合实验设计。

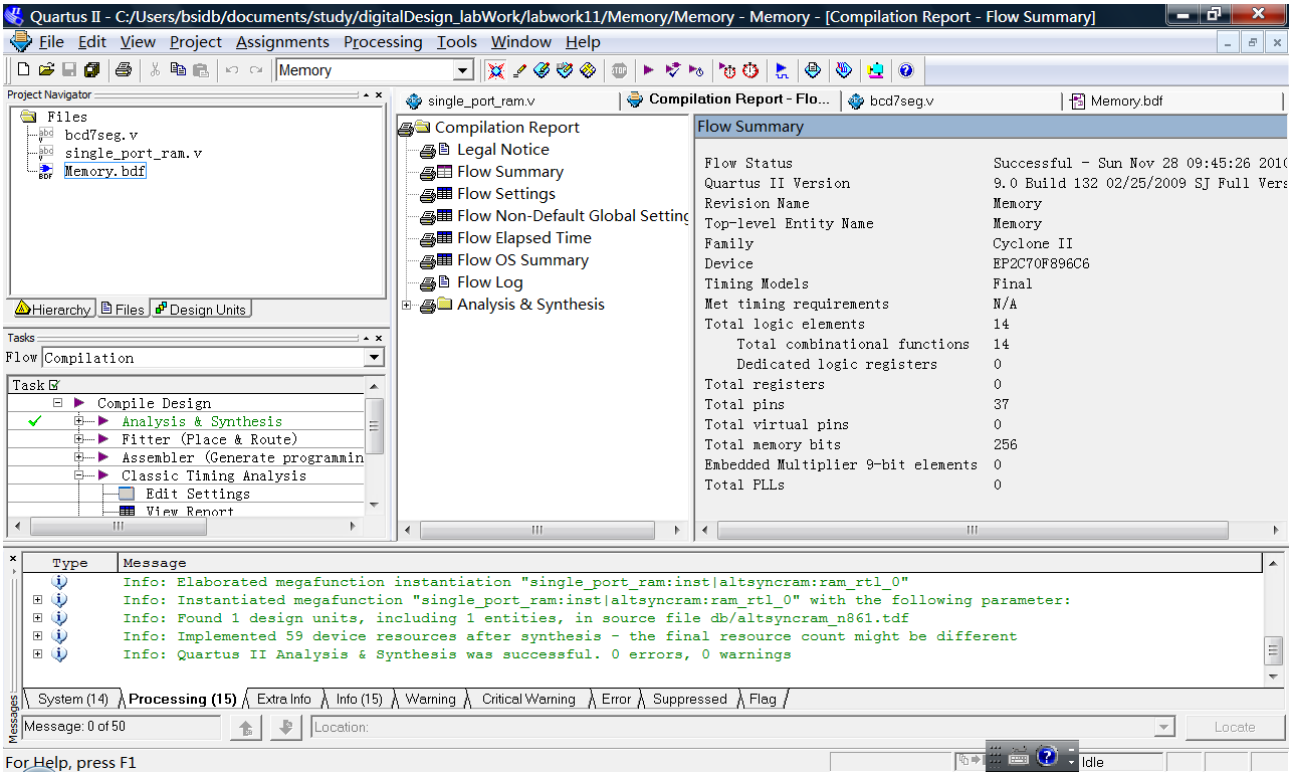


图 6: 分析/综合成功

对实验设计进行功能仿真，先编写测试文件。

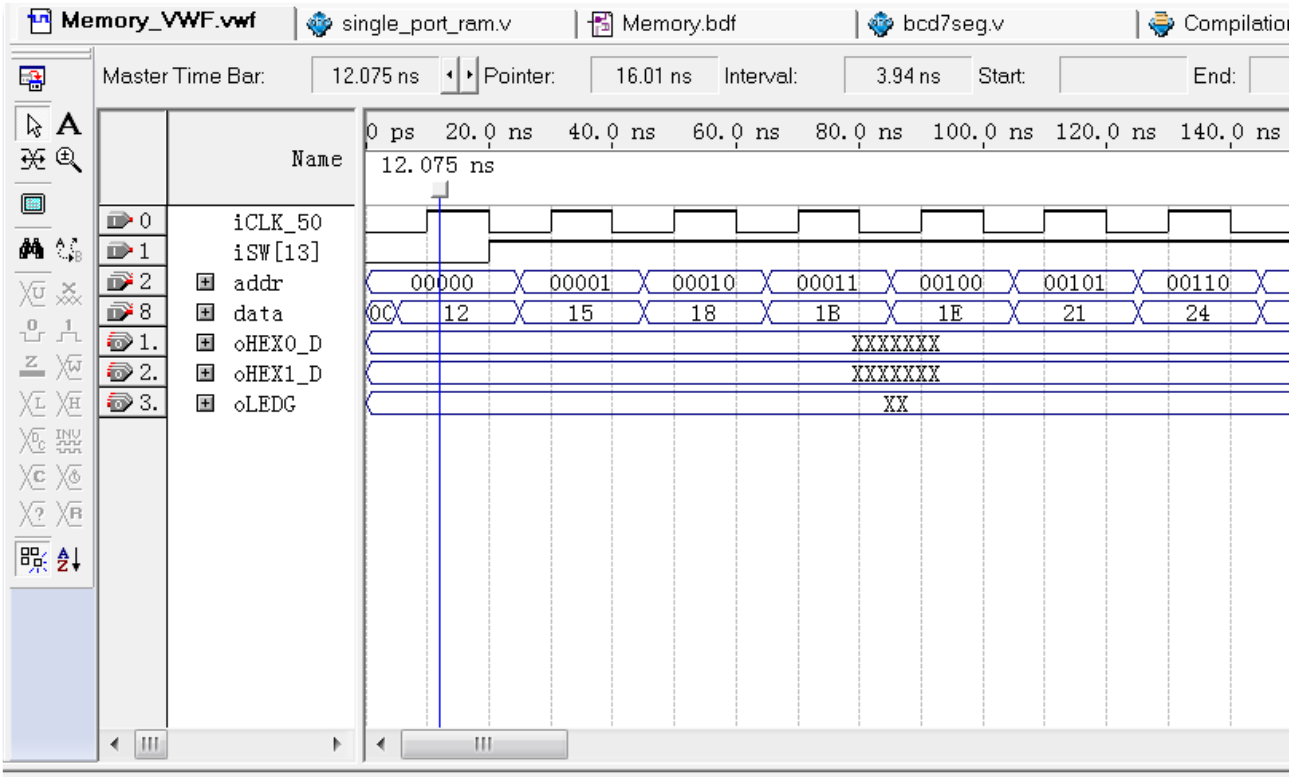


图 7: 编写功能仿真测试文件

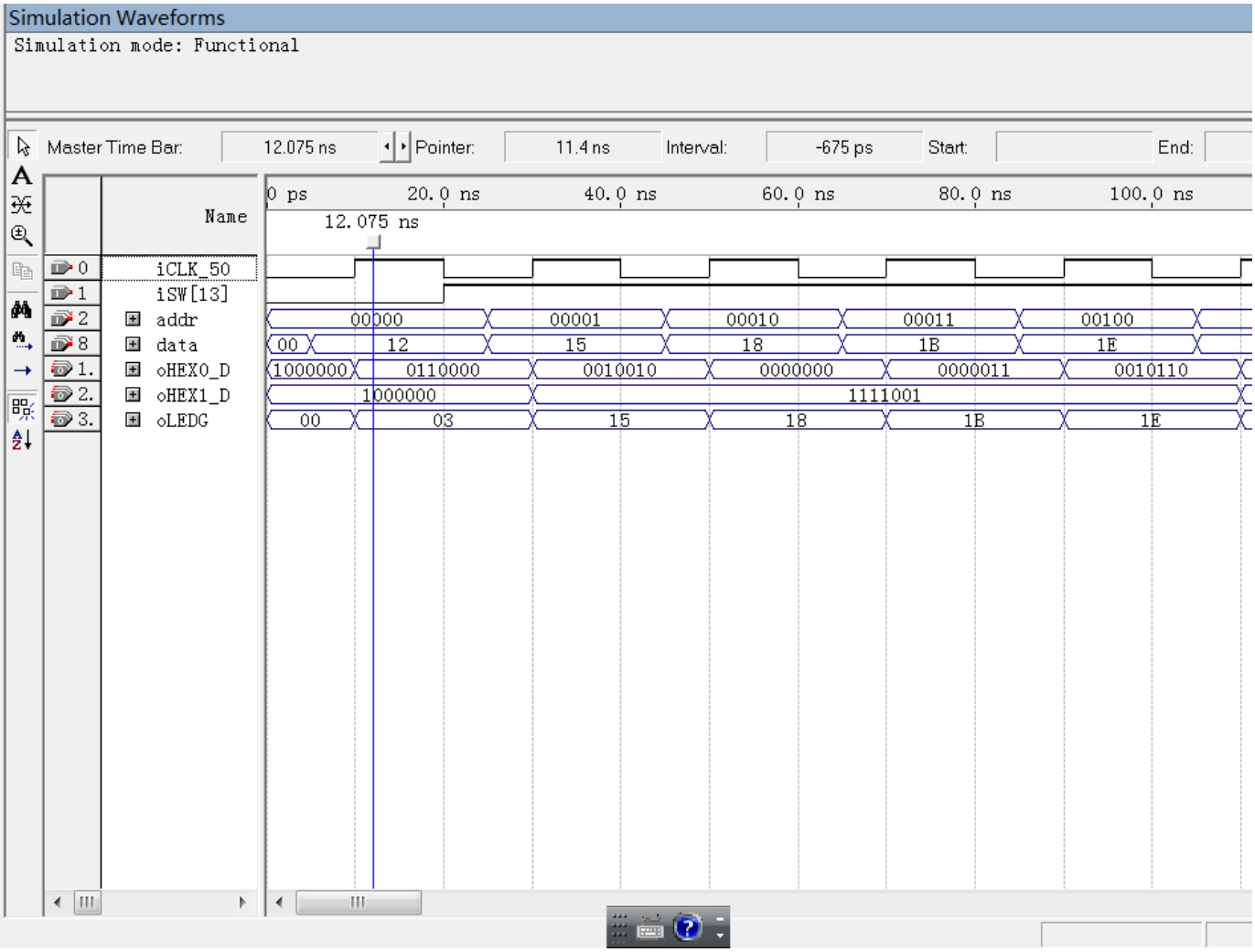

Simulation Waveforms
Simulation mode: Functional
Master Time Bar: 12.075 ns
Pointer: 11.4 ns
Interval: -675 ps
Start:
End:
Name
0 ps
20.0 ns
40.0 ns
60.0 ns
80.0 ns
100.0 ns
12.075 ns
0 iCLK_50
1 iSW[13]
2 addr
8 data
1. oHEX0_D
2. oHEX1_D
3. oLEDG
00000
00001
00010
00011
00100
00
12
15
18
1B
1E
1000000
0110000
0010010
0000000
0000011
0010110
1000000
1111001
00
03
15
18
1B
1E

图 8: 功能仿真结果 1

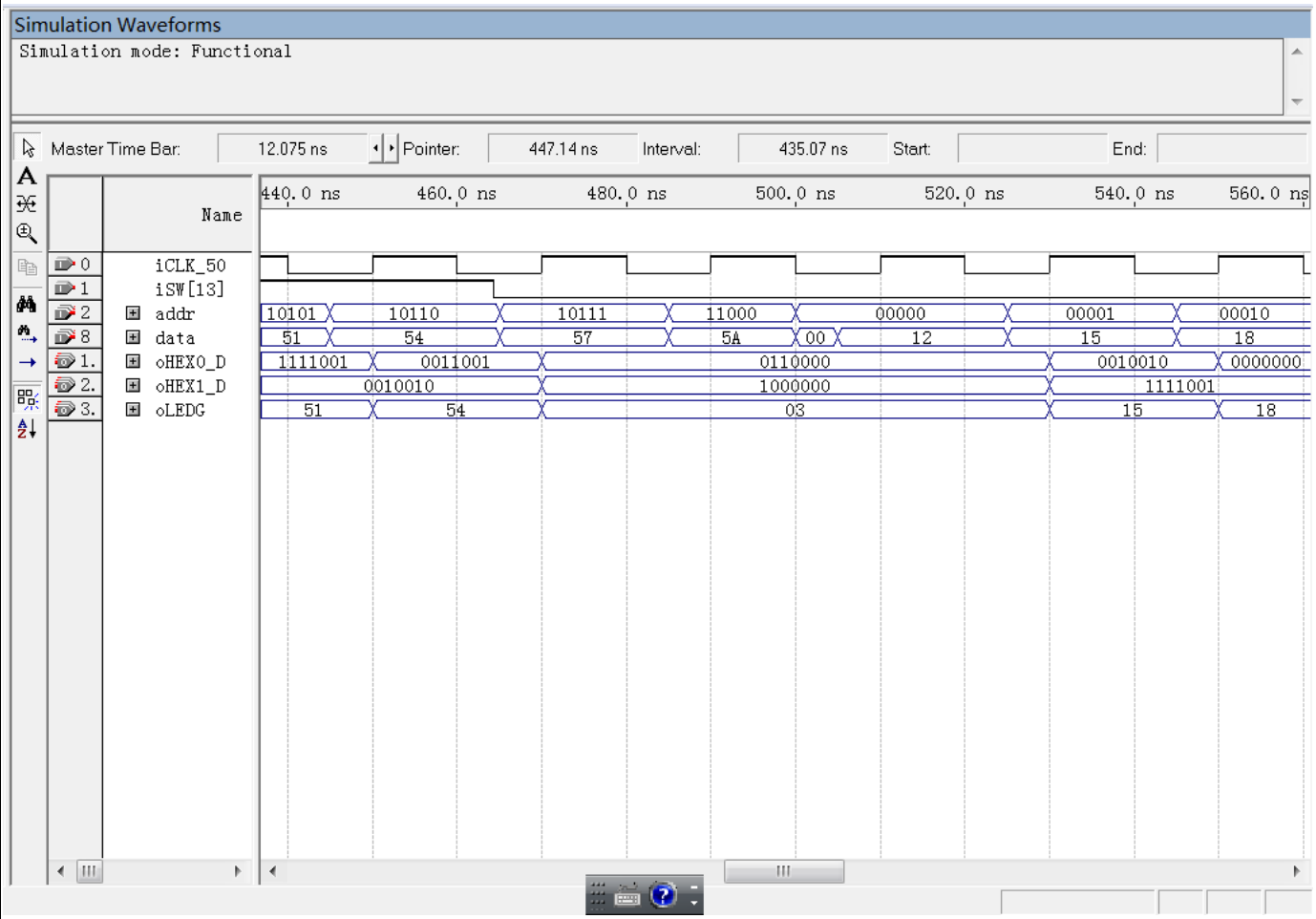


图 9: 功能仿真结果 2

功能仿真符合预期要求。

分配引脚。

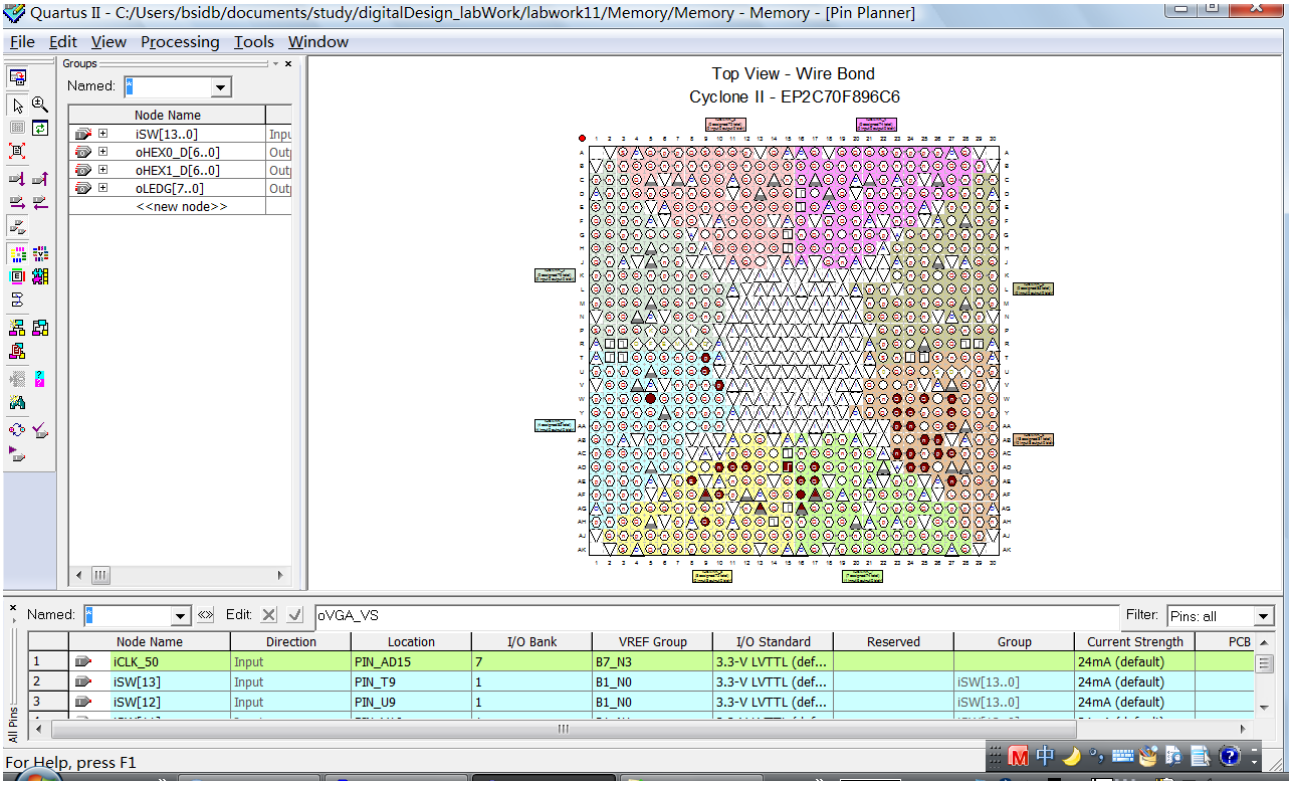


图 10: 分配引脚

对实验设计进行全编译。

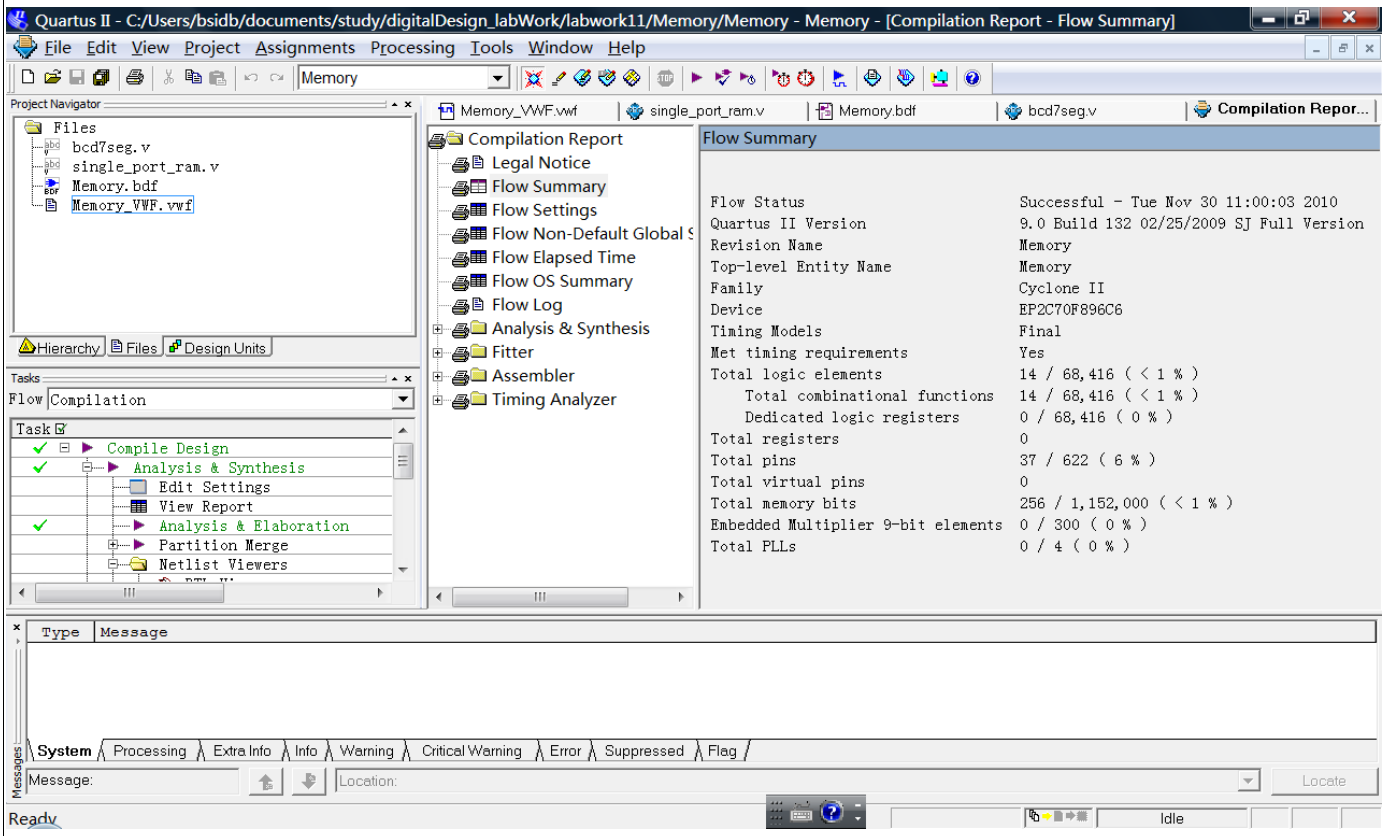


图 11：全编译实验设计

进行时序仿真。

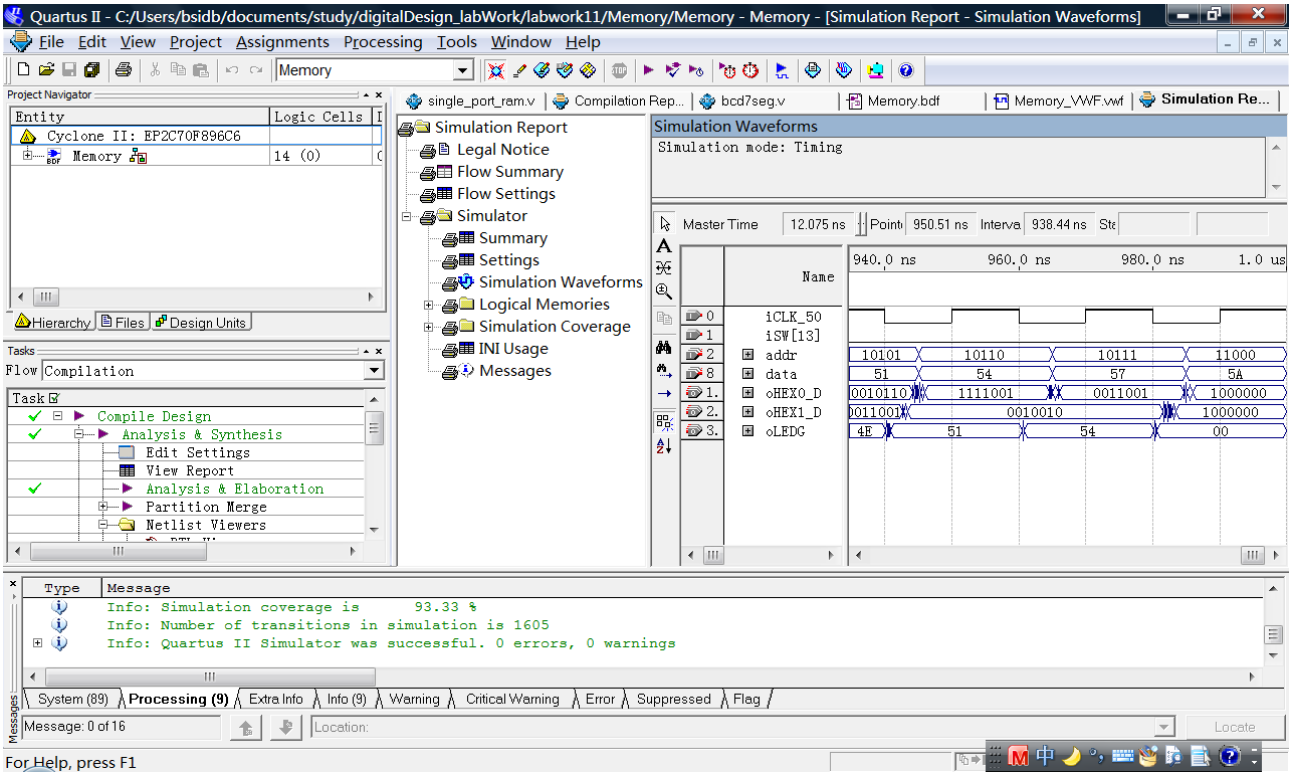


图 12：时序仿真结果，符合要求

B.利用 LPM 库实现设计

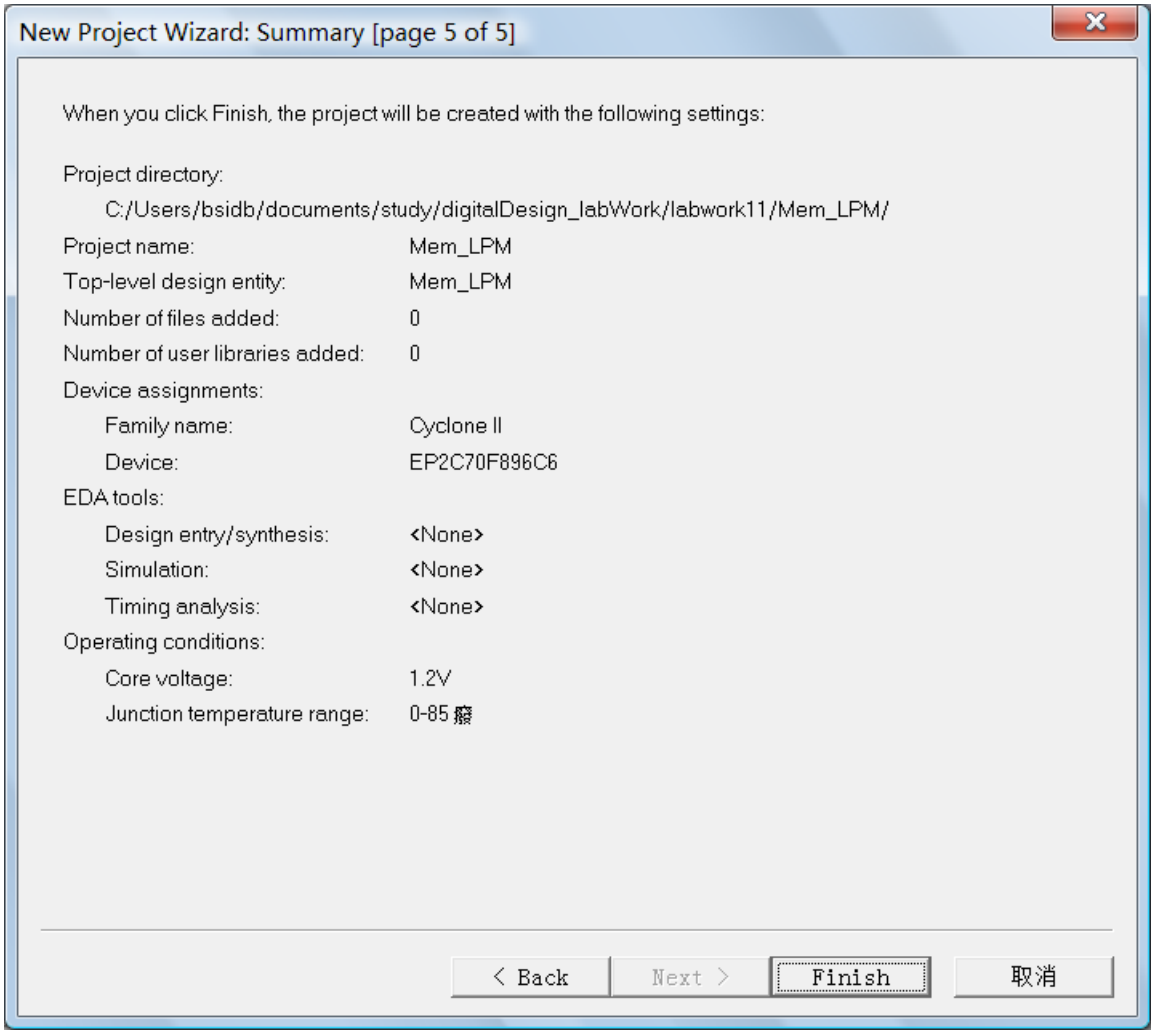


图 13: 工程小结

利用 MegaWizard-Plugin Manager 向导，加入基于 LPM 库的设计。依然选用的是单口 RAM 工作模式。

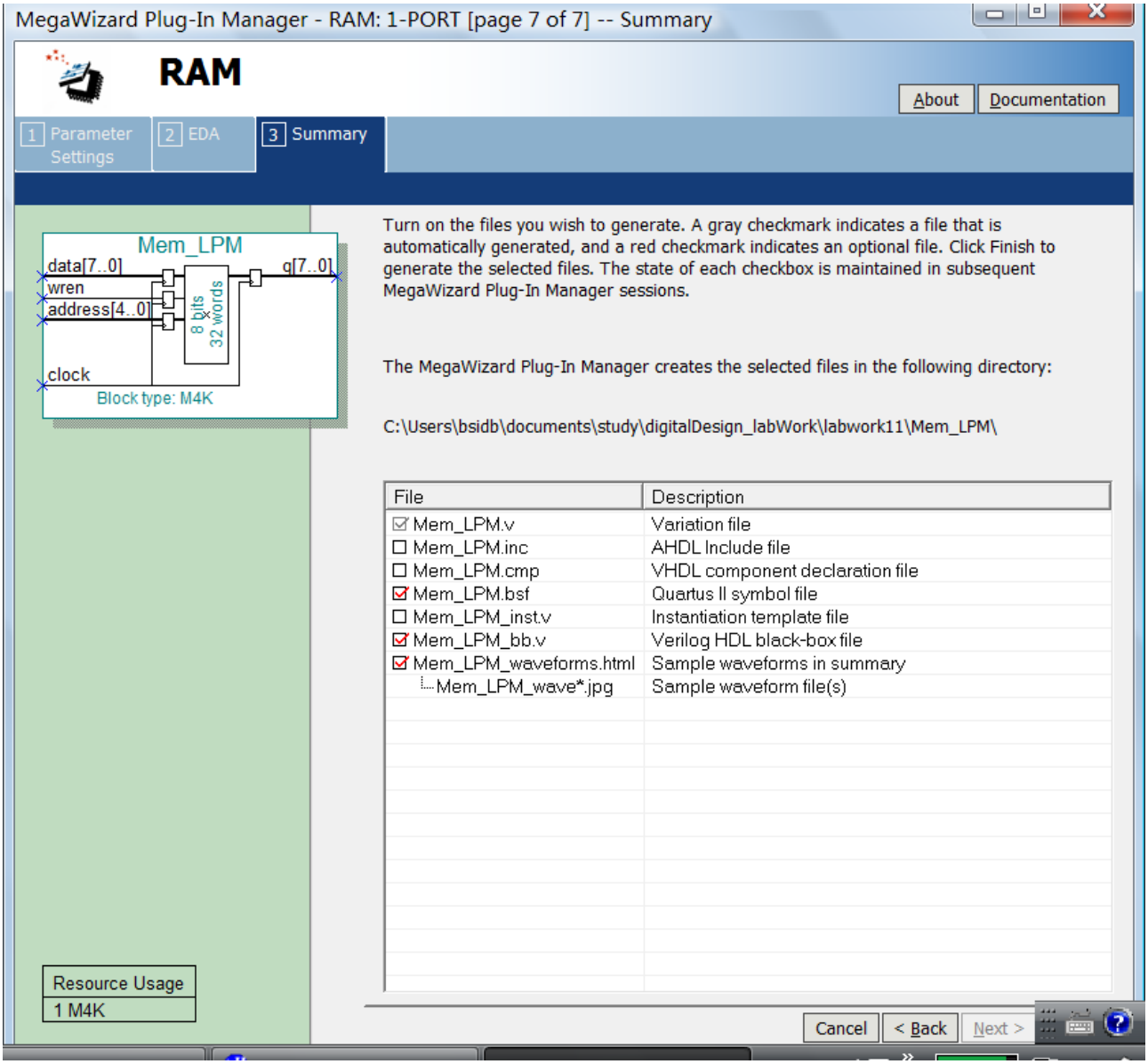


图 14: 利用向导加入 LPM 库设计

将 LPM 库生成的设计与 DE2-70 平台信号连接。

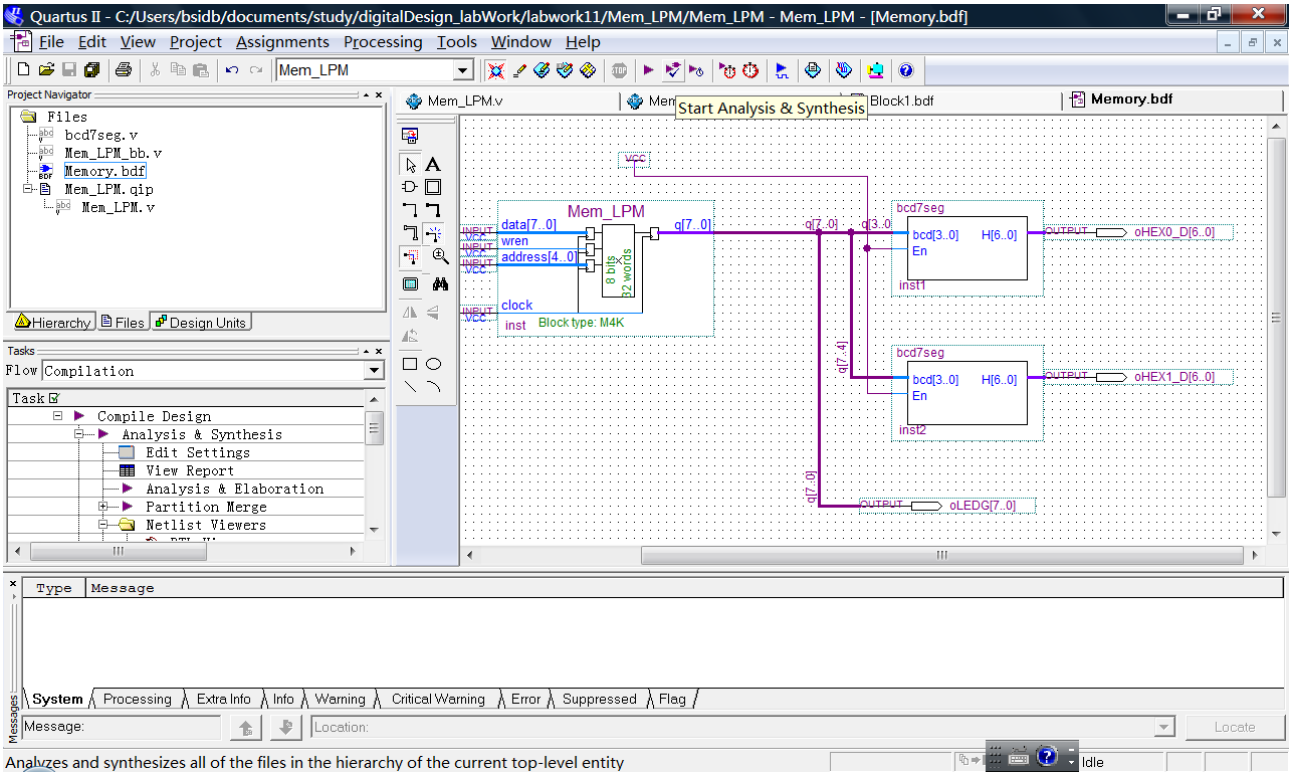


图 15：将实验设计与 DE2-70 平台信号相连

新建 mif 文件，利用它对内存单元初始化。

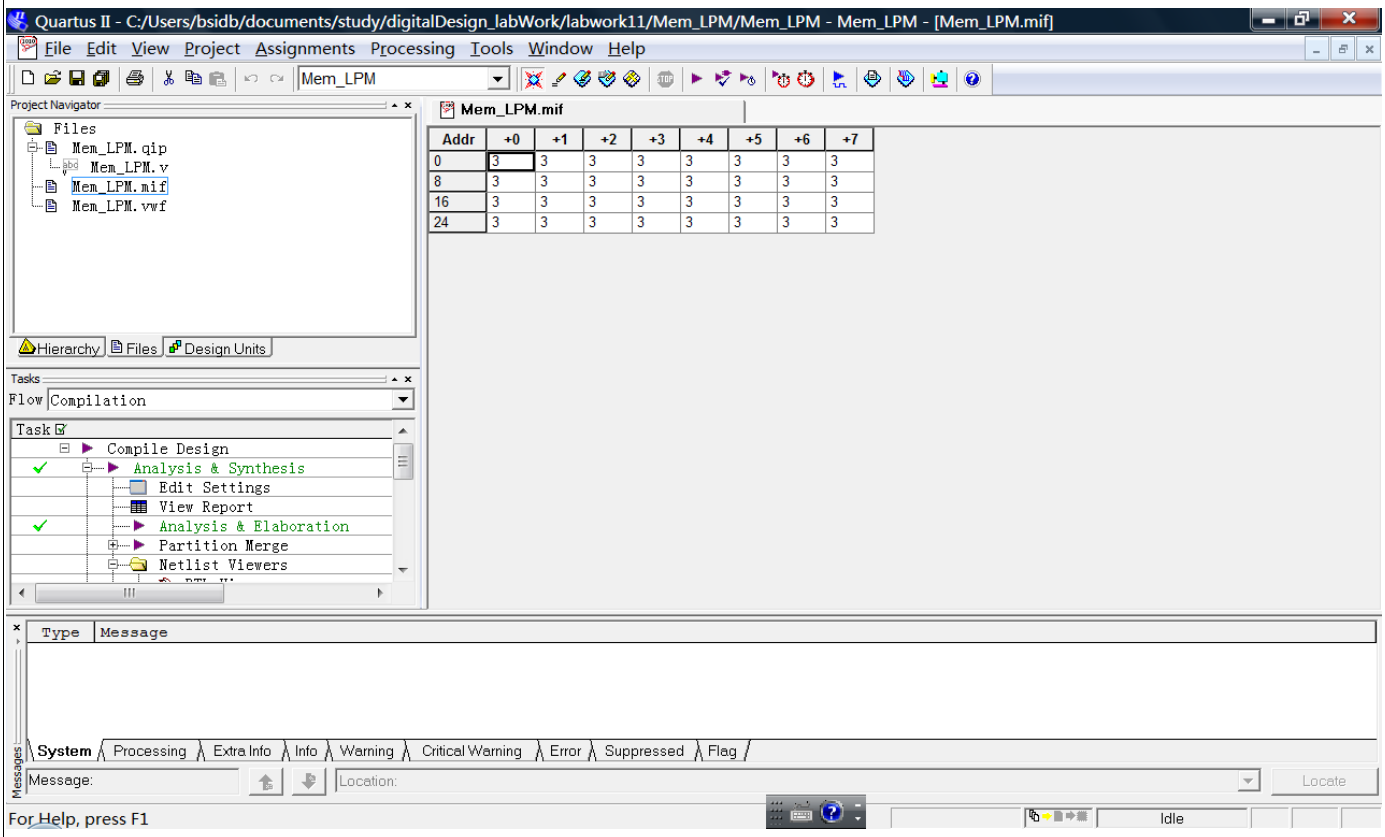


图 16：编辑 mif 文件

先对 LPM 库生成的存储单元进行分析/综合。

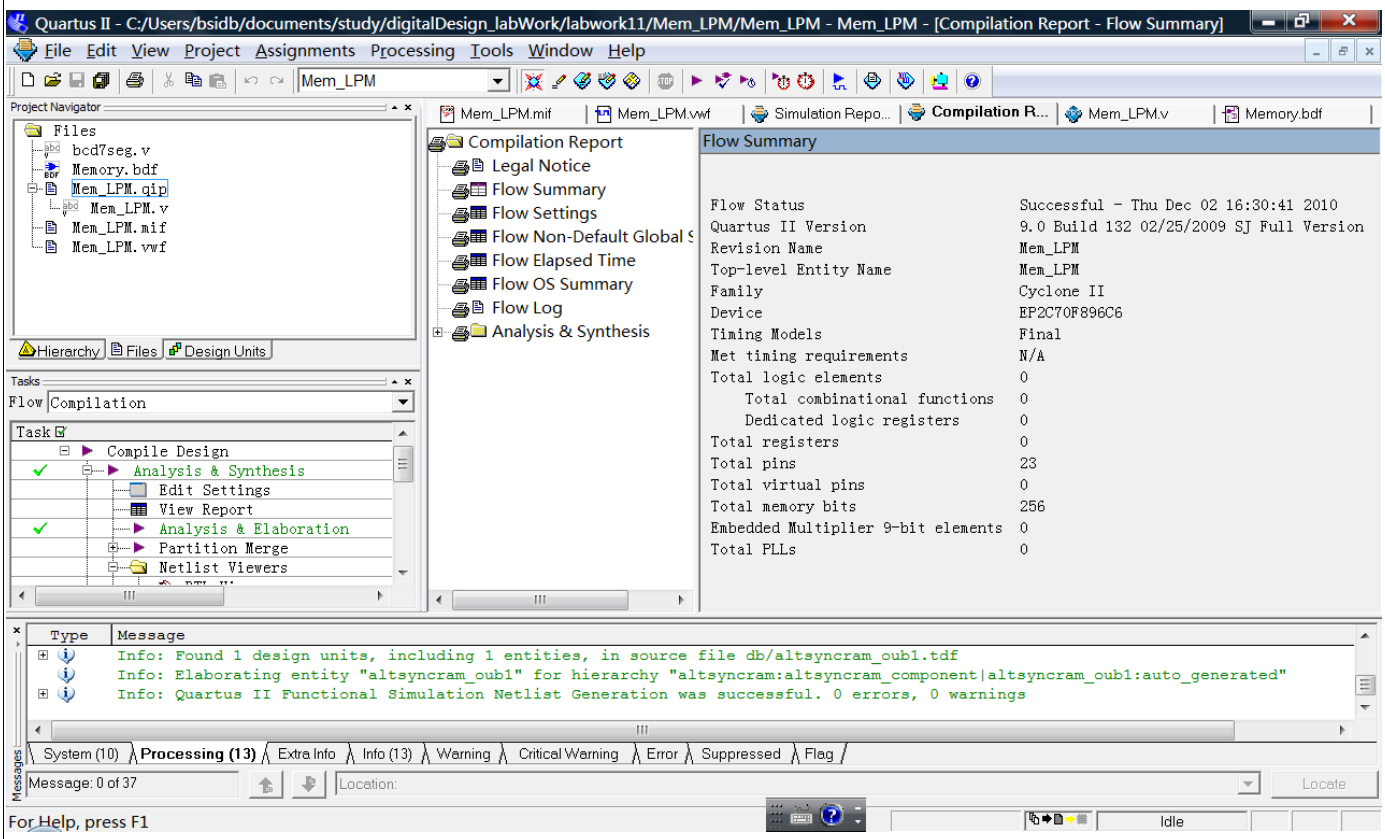


图 17：对 LPM 库分析/综合

进行功能仿真，查看结果。

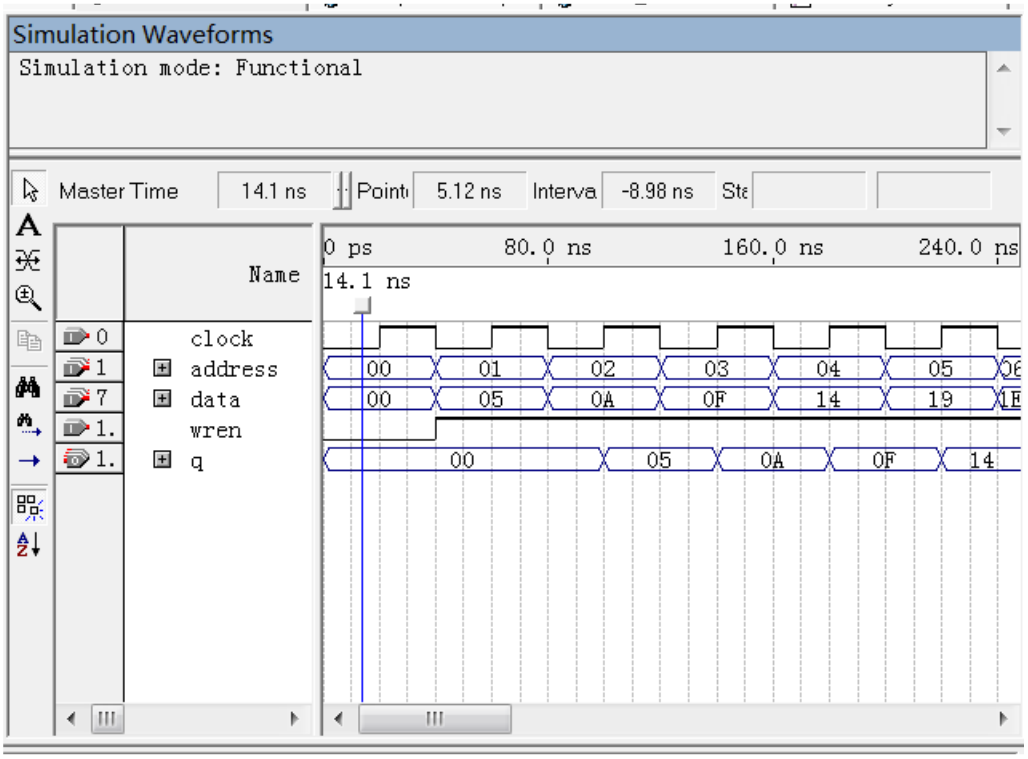


图 18: LPM 库功能仿真结果 1

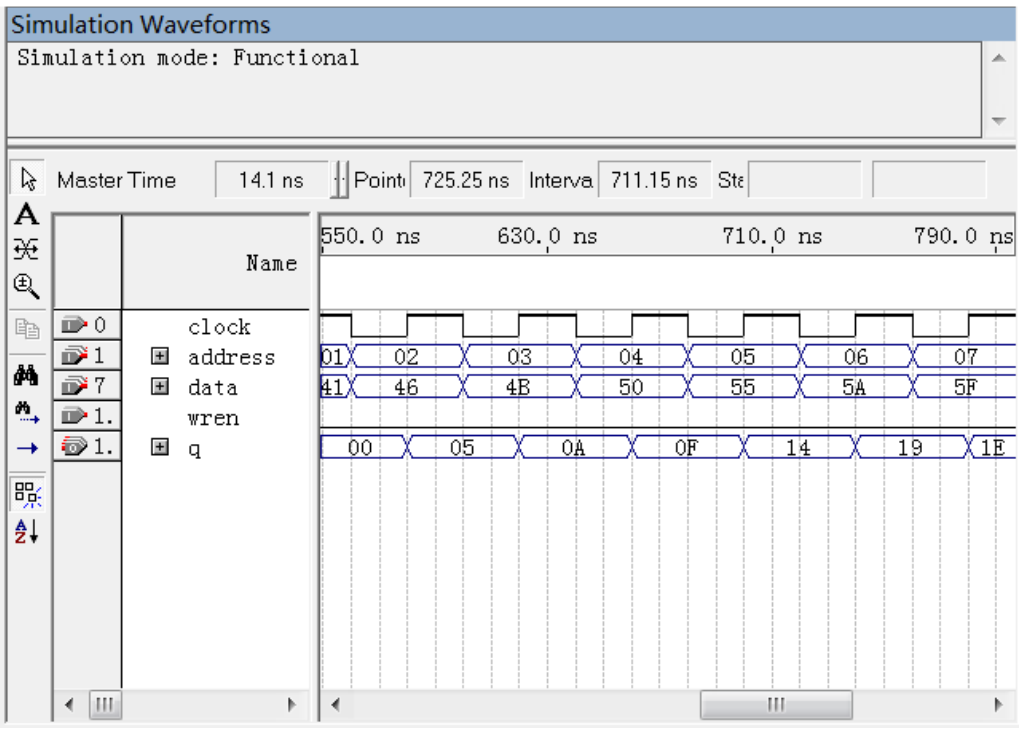


图 19: LPM 库功能仿真结果 2

对全设计进行分析/综合。

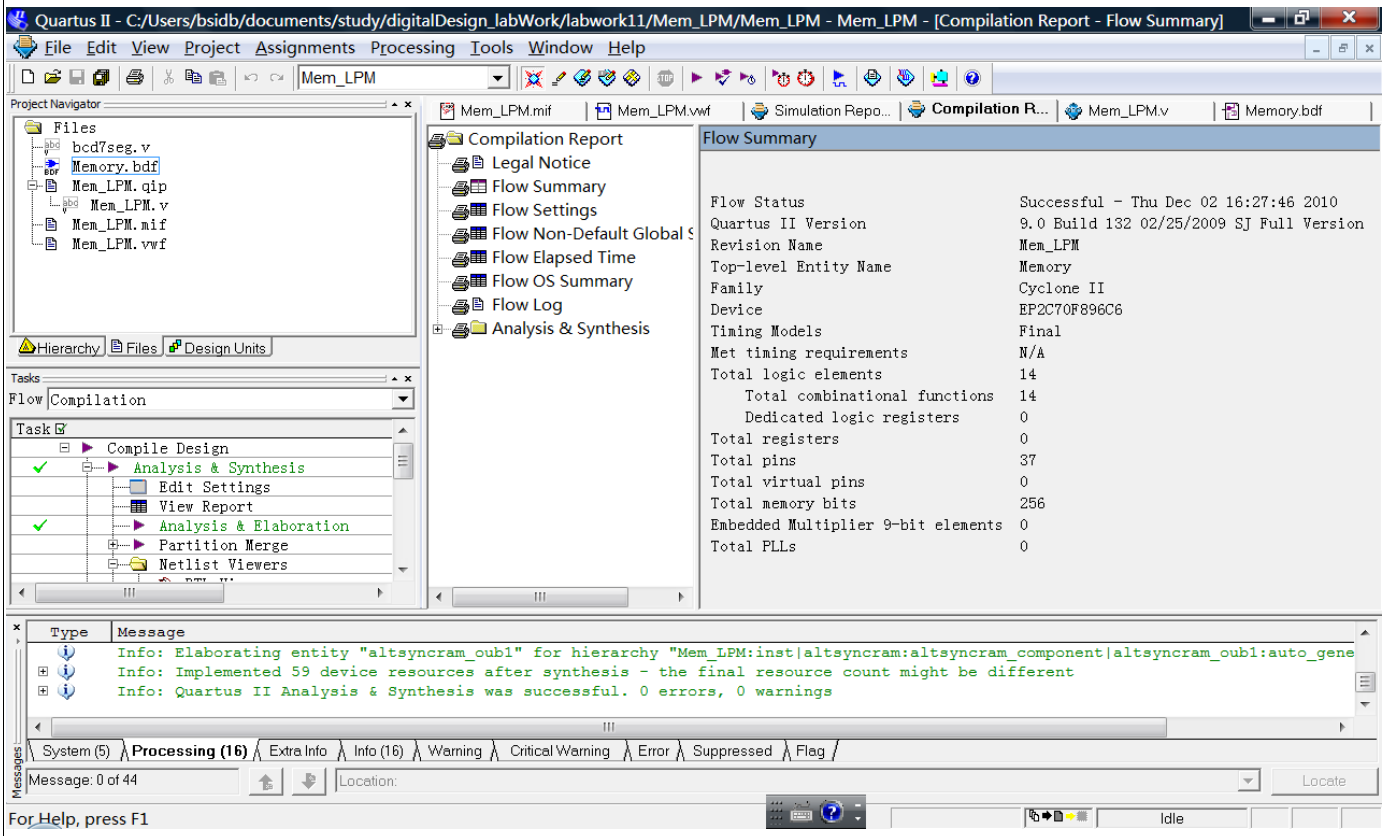


图 20：分析/综合

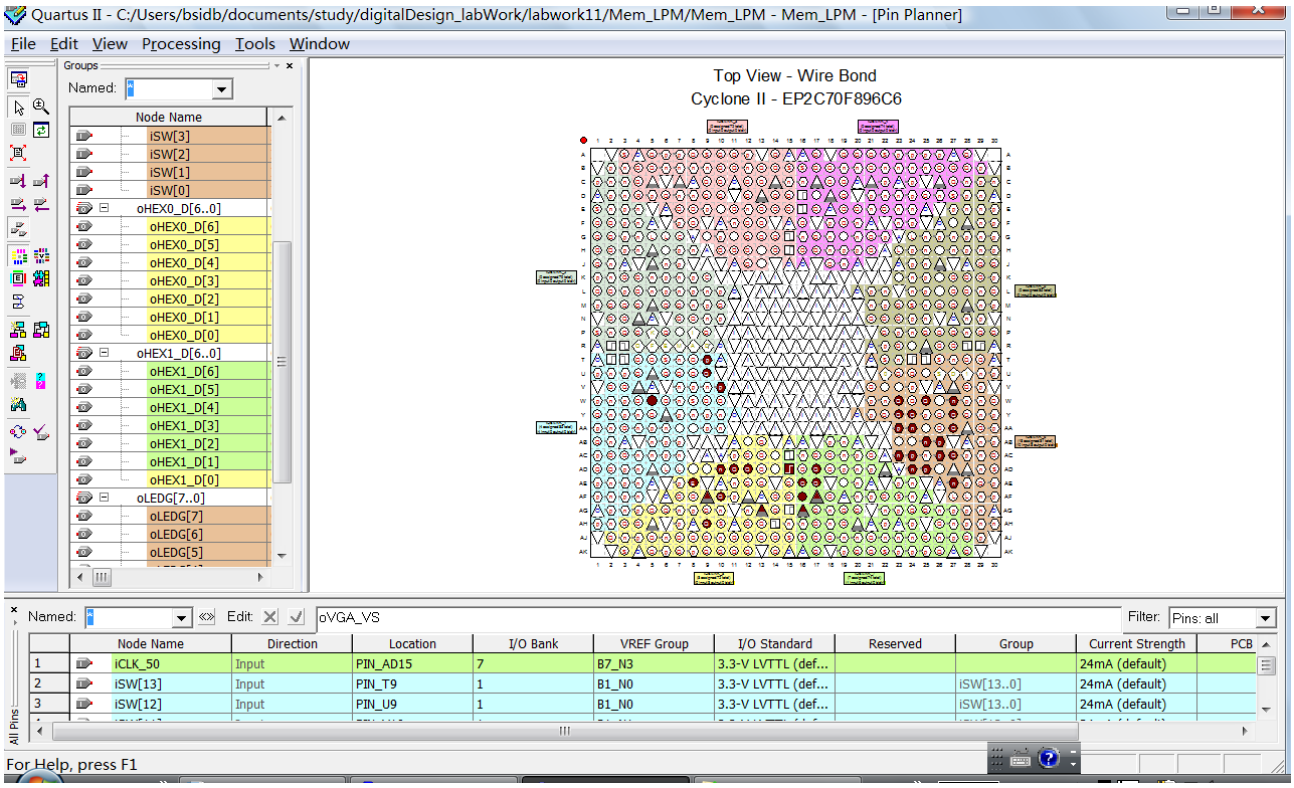


图 21：分配引脚

进行全编译。

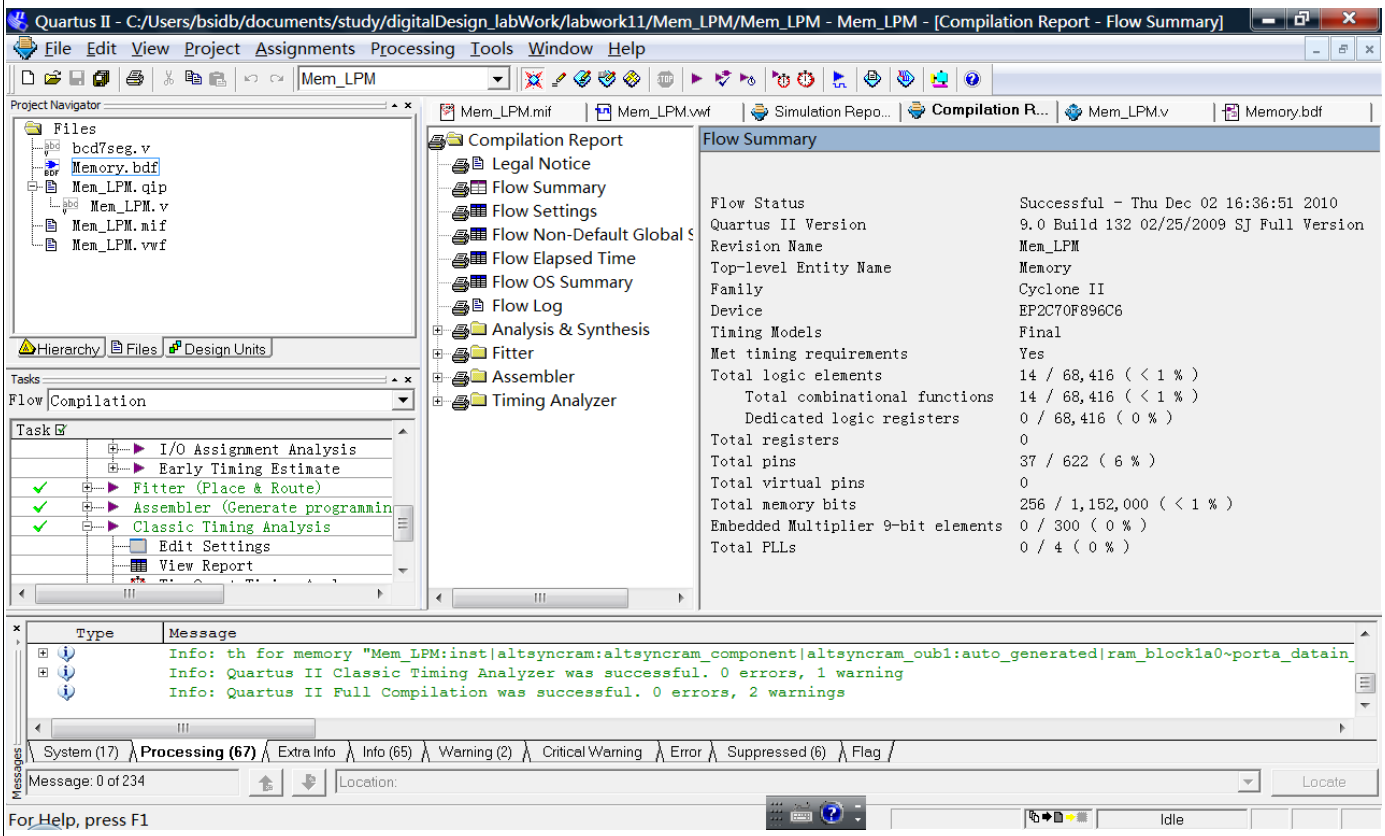


图 22: LPM 库全编译

6. *实验结果*

本次实验学习了存储器的设计方法。并利用了模板和 LPM 库两种方式在 FPGA 上实现了存储器。学习使用了 FPGA 中 M4K 片内存储单元。了解到了 M4K 片内存储单元的一些使用方法。

对实验进行功能仿真得到了如下结果。

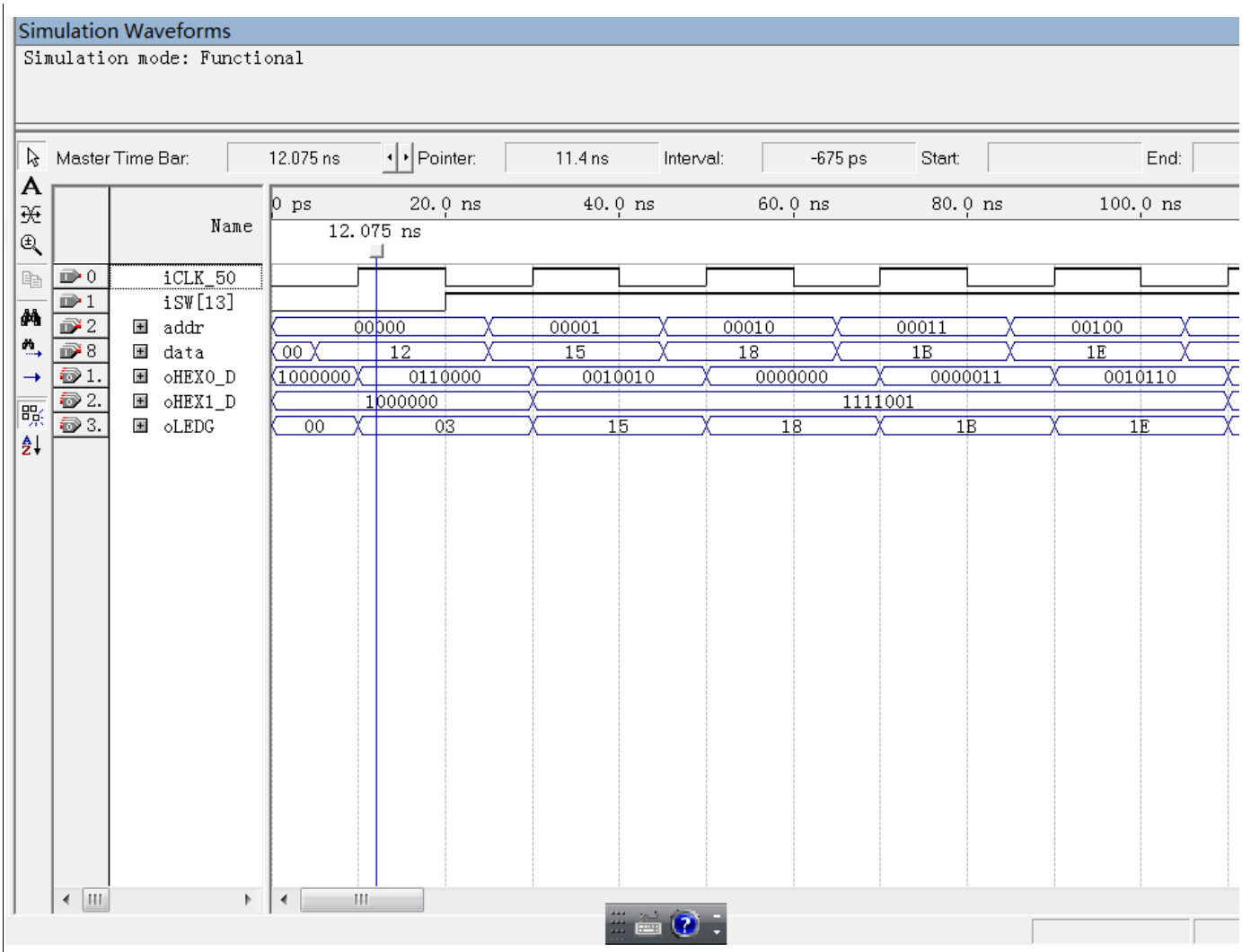


图 23: 模板设计功能仿真结果 1

在没有写入之前，第一个时钟上升沿到来时，从 0 号单元中读出的结果是 3，确实为初始化的值。在以后的时钟上升沿进行写入。可以看到输出端 oLEDG 随着写入的值在同一时钟上升沿发生变化，并没有延迟。这样的结果符合 Verilog HDL 中对 q 的描述：

```
// Continuous assignment implies read returns NEW data.
  // This is the natural behavior of the TriMatrix memory
  // blocks in Single Port mode.
  assign q = ram[addr_reg];
```

根据模板提供的注释，在单口模式下 q 始终以内存单元的最新的值输出。

在写使能无效后，开始读出存储的值。

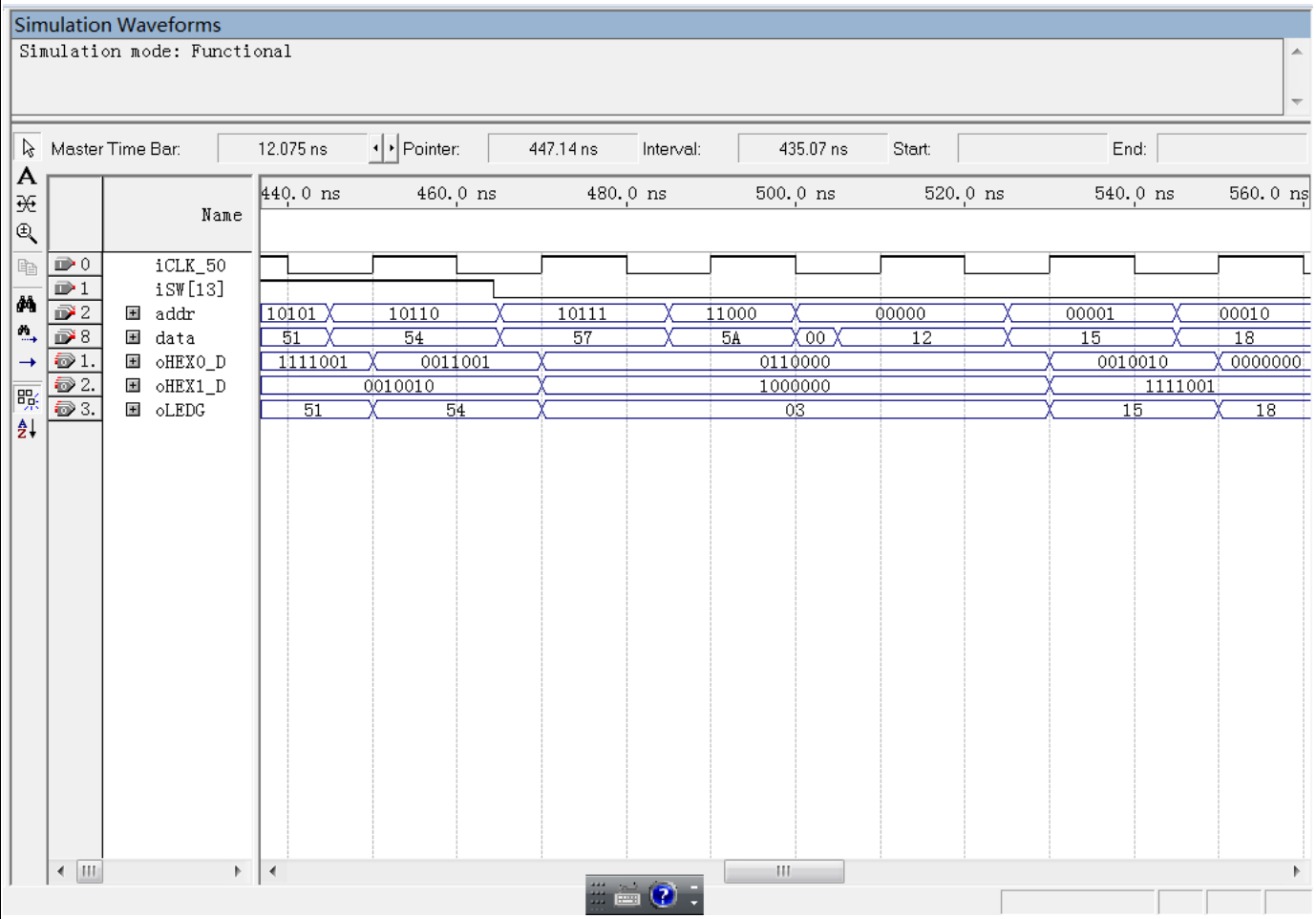


图 24: 功能仿真结果 2

在写使能（iSW[13]）无效后，进入读状态。对于地址 10111，11000，00000 存储的内容均为初始化的值 3H。而对于 00001 等内存单元的内容，现在读出的值为当时写入的值。说明功能仿真的结果符合当初的要求。

对于 LPM 库的设计方法所得到的功能仿真结果为：

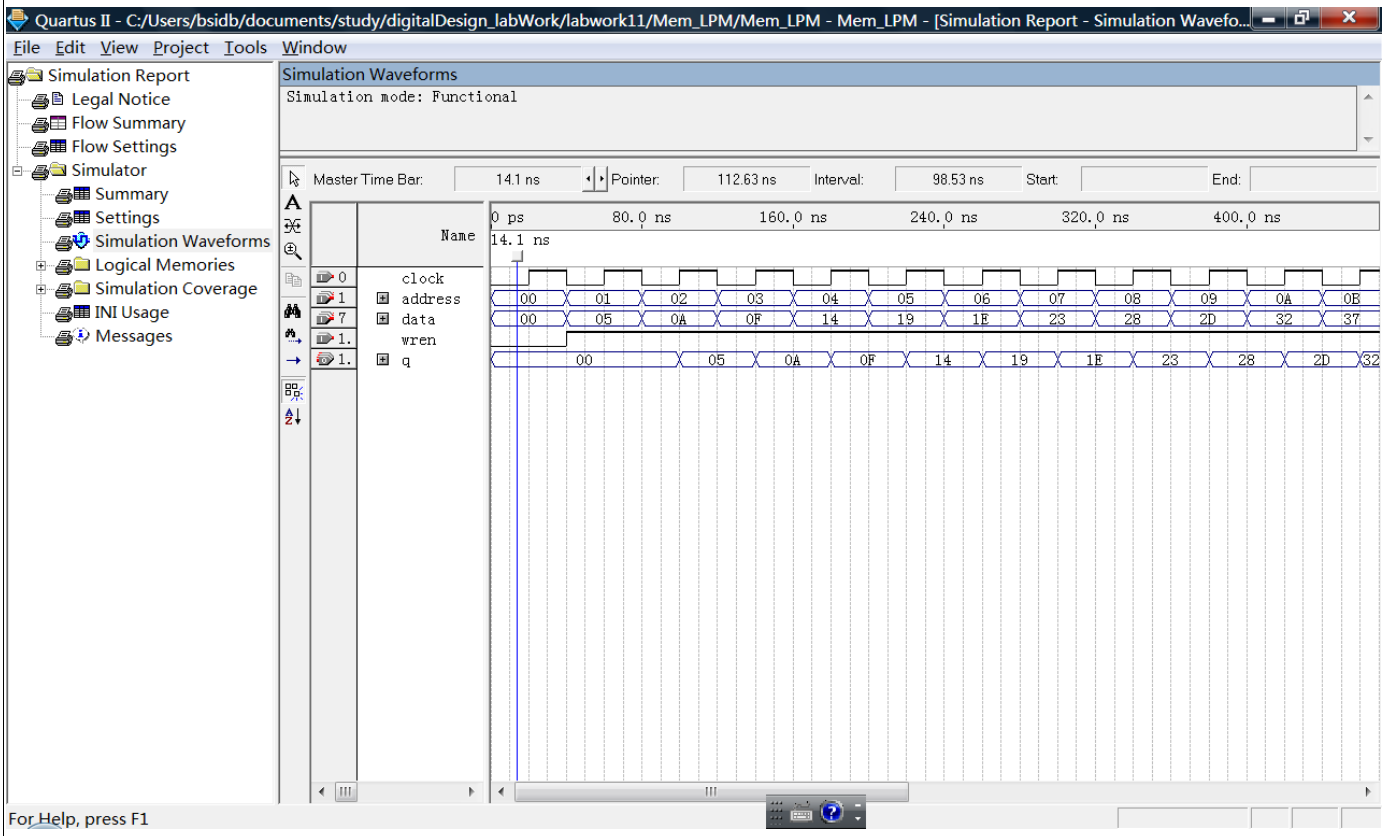


图 25: LPM 库设计功能仿真结果

这个功能仿真是在刚使用 LPM 库生成存储器时进行的仿真结果，没有做显示初始化。从图中可以看到，当写使能有效时第一个时钟到来时输出 q 并没有随输入发生变化，而是缓了一个周期才变化。发生这种情况的原因是在使用向导定制存储器模块时，在输出端是否添加寄存器上的选项上选择了是。因此有这个寄存器作为缓冲，使输出迟后了一个时钟上升沿。

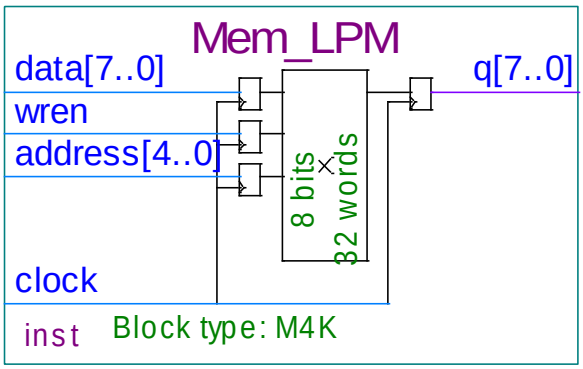


图 26: LPM 库生成的存储器模块逻辑符号

从逻辑符号上看，其输出具有寄存器，因此会迟后一个时钟上升沿。

7. 实验中遇到的问题及解决方案

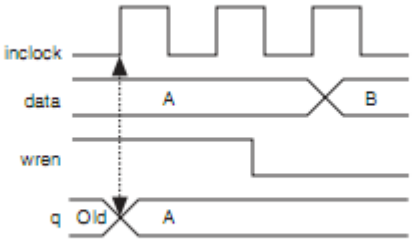
A.双口模式写操作需要两次有效的问题

在做实验中，我选择的 M4K 存储单元工作模式是单口 RAM 模式；有一些同学选择的是双口具有读写两种

时钟的 RAM 模式。在双口模式下，当对同一个单元同时读、写时，出现了一个奇特的现象，在功能仿真中并没有体现出来，就是当第一次写时钟上升沿到来后马上进行读操作发现输出仍然为该单元原来的老值，只有再来一个写操作上升沿后再读出，输出才随输入发生变化。或者连续进行两次写操作后再进行读操作这样所得到的值才为刚刚写入的值。发生这种情况的原因和 M4K 存储器的行为相关。这种情况是符合 M4K 存储器的设计的。

在计算机组成原理的课程上，我们学习过尽量避免对同一个内存单元同时进行读、写，以避免出现非预期的值。同样对存储器单元的使用应该尽量避免同时对一个地址同时进行读、写，这样可能会带来预料之外的情况。但是同时读、写在实际应用中可能无法避免，因此有必要对这种情形下 M4K 存储器的行为进行约定。查阅 Altera 公司提供的官方文档 *Arria GX Device Handbook, Volume 2 ,Chapter 6 TriMatrix Embedded Memory Blocks in Arria GX Devices*(文档编号 AGX52006-1.2)【该文档已经包含在实验报告的压缩包中】。其中关于上述情形下 M4K 存储器的行为进行了约定。文档中“Read-During-Write Operation at the Same Address”一节同时给出了有关的功能波形图。

Figure 6–21. Arria GX Same-Port Read-During-Write Functionality *Note (1)*



Note to Figure 6–21:
(1) Outputs are not registered.

图 27: 单口模式下的同时读写时的功能波形图

可以看到在单口模式下输出在内存单元写入的同时发生变化，输出没有被缓冲。

而在双口模式下，存储器的行为由一个参数决定。根据原文：

The READ_DURING_WRITE_MODE_MIXED_PORTS parameter for M512

and M4K memory blocks determines whether to output the old data at

the address or a "don’t care" value. Setting this parameter to OLD_DATA

outputs the old data at that address. Setting this parameter to DONT_CARE

outputs a "don’t care" or unknown value.

在双口同时读、写时 M4K 存储器的行为，由参数 READ_DURING_WRITE_MODE_MIXED_PORTS 决定。两种情形下的功能波形图见下。

Figure 6–22. Arria GX Mixed-Port Read-During-Write: OLD_DATA

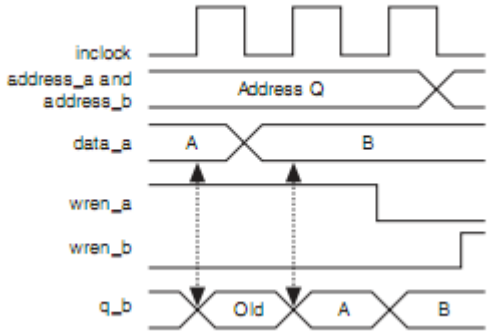


图 28: 双口模式下同时读、写情况的功能波形图（OLD_DATA)

在 OLD_DATA 参数设置下，输出会被缓存。即先输出旧的值，在下一个时钟输出新的值。

而在 DONT_CARE 设置下，在对同一个地址同时进行读、写时读操作会输出一个不确定的值。但此时写操作是生效的。

Figure 6–23. Arria GX Mixed-Port Read-During-Write: DONT_CARE

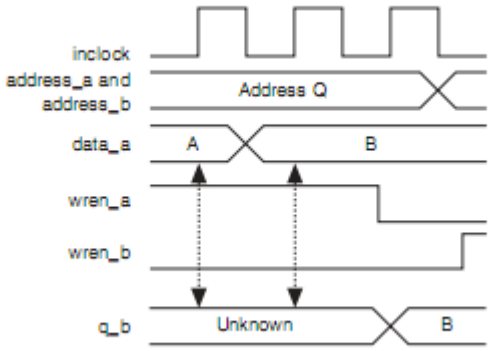


图 29: 双口模式下同时读、写的功能波形图 (DONT_CARE)

实验中出现的现象和官方提供的功能波形图是一致的，说明这是正常现象。

8. 实验的启示/意见和建议

在 Quartus II 软件提供的模板中，提供了许多的注释信息。仔细阅读这些注释信息可以更好的理解电路的行为。在硬件描述语言并不能完全描述电路行为的情况下，Altera 公司提供了有关的手册可以在需要时进行查阅。

功能波形图一般能够确定的给出某些条件下电路的行为。在设计中如果电路的某项特性无法很好地用语言表示出来，可以借助于波形图的方式。

附：

总用时：约8h30min

预习时间

预习实验 0h

验收实验 2h

预习实验报告 2h

课堂验收实验：

课堂实验时间 1h30min

完善实验报告 3h