南京大学

数电实验五：寄存器组与存储器

课程名称：数字逻辑与计算机组成实验

姓名：孙文博

学号：201830210

班级：数电一班

邮箱：201830210@smail.nju.edu.cn

实验时间：2022.3.30

一、 实验目的

1. 复习数字电路中寄存器堆和存储器的相关知识；

2. 了解 FPGA 的触发器及片上存储器的特性；

3. 分析存储器的工作时序和结构；

4. 设计一个 16*8 寄存器组和基于 IP 核的单口 RAM 存储器。

二、 实验环境

设计\编译环境：Quartus (Quartus Prime 17.1) Lite Edition

开发平台：DE10-Standard

FPGA 芯片：Cyclone II 5CSXFC6D6

三、 实验原理

1. 寄存器

寄存器与存储器都是数字系统中的记忆器件，用来保存程序和数据，计算机内部除了主存以外，还有许多记录状态的通用寄存器，如程序计数器 PC 等,CPU 的状态也由这些通用寄存器组中的内容决定。FPGA 芯片通过触发器实现寄存器的功能，其逻辑如下：

```
module register1(load,clk,clr,inp,q);
  input  load,clr,clk,inp;
  output reg q;

  always @(posedge clk)
    if (clr==1)
        q <= 0;
    else if (load == 1)
         q <= inp;
endmodule
```

这个一位寄存器还包括清零端和写使能，只有写使能有效时才会写入数据，以防数据被篡改。多个寄存器组合在一起就构成了寄存器组，用 Verilog 语言中的数组是实现，代码类似上面的一位寄存器。

2. 存储器

存储器是一组存储单元，用于在计算机中存储二进制的数据，存储器的端口包括： 输入端、输出端和控制端口。输入端口包括：读/写地址端口、数据输入端口等；输出端口一般指的是数据输出端口；控制端口包括时钟端和读/写控制端口。一般读取数据不受时钟和使能端控制，只要输出地址有效，就立即将此地址所指的单元中的数据送到输出端口上；写入数据需要写使能有效，且在一个时钟的有效沿（上升或下降沿）时才会将数据写入到对应地址的存储单元中去。

在 Verilog HDL 中，可以用多维数组定义存储器。例如，假设需要一个 32 字节的 8 位存储器块，即此存储器共有 32 个存储单元，每个存储单元可以存储一个 8 位的二进制数。这样的存储器可以定义为 32×8 的数组，在 Verilog 语言中可以作如下变量声明：

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

值得注意的是，虽然寄存器和存储器都是用来存储状态信息的，但是它们在用途和实现上有较大的区别，寄存器一般要求存取速度快、并行访问要求高，所以通常寄存器的容量较小；主存一般容量较大，但是读写时间较长，并且读写过程有严格的时序要求。在 Verilog 中，

虽然寄存器组和存储器的描述都是二维数组的方式，但是，编译和综合过程中会根据代码访问的要求来选择具体的实现方式。具体分析可参照思考题。

四、 实验过程

1. 基于 FPGA 逻辑单元实现寄存器堆

根据上述知识，我们可以用 Verilog 语言中的数组模拟一个寄存器堆，代码如下：

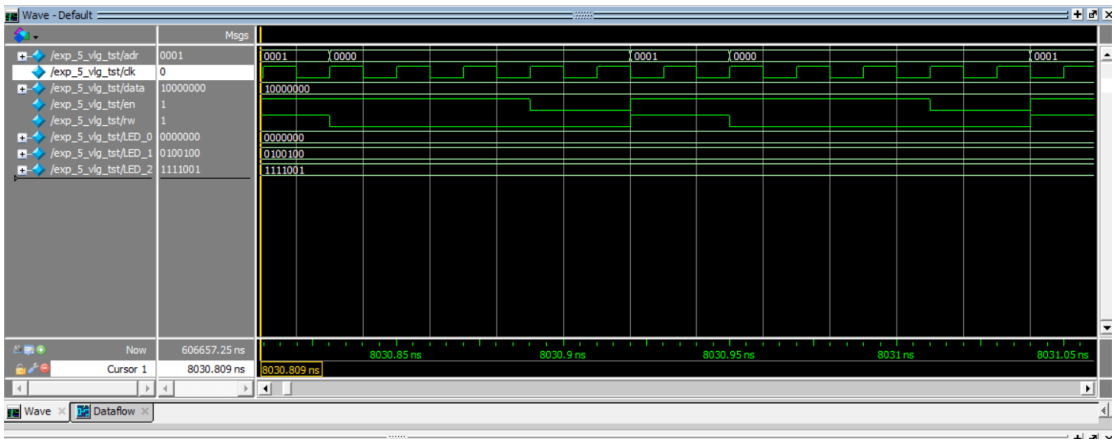
```
module exp_5(
    input  [3:0]  adr,      // 读口/写口地址
    input  [7:0]  data,     // 写入的数据,write data
    input  en,              // 写使能,write enable
    input  rw,              // 判断当前模式为读/写, 0为读取,1为写入
    input  clk,             //时钟
    output [6:0] LED_0,
    output [6:0] LED_1,
    output [6:0] LED_2
    );
```

输入端包括四位读写公用地址，八位数据，写使能，读写选择端和时钟信号，输入是三个七段数码管，当读取操作时显示当前地址的数据，写入操作时显示将要写入的数据，具体实现代码如下：

```
// initial once
    if(i)begin
        for(i=0;i<16;i=i+1)
            register[i]=i;
        i=0;
    end

    if(rw==0) begin
        out=register[adr]; // 显示读口地址对应的寄存器内容
    end

    else begin
        out=data; // 显示将要写入的数据
        if(en)
            register[adr]=data;
    end
```

```
case(out%10)
    0:led0=7'b1000000;
    1:led0=7'b1111001;
    2:led0=7'b0100100;
    3:led0=7'b0110000;
    4:led0=7'b0011001;
    5:led0=7'b0010010;
    6:led0=7'b0000010;
    7:led0=7'b1111000;
    8:led0=7'b0000000;
    9:led0=7'b0010000;
endcase

case((out/10)%10)
    0:led1=7'b1000000;
    1:led1=7'b1111001;
    2:led1=7'b0100100;
    3:led1=7'b0110000;
    4:led1=7'b0011001;
    5:led1=7'b0010010;
    6:led1=7'b0000010;
    7:led1=7'b1111000;
    8:led1=7'b0000000;
    9:led1=7'b0010000;
endcase

case(out/100)
```

进行 testbench 仿真检测的结果如下：



对其进行引脚分配，并烧录到开发板上，等待验收。

2. 基于 IP 核实现存储器

接下来我们基于 IP 核实现一个存储器，并且将之前实现的寄存

器堆和 RAM 放在一个工程中，使得此两个物理上完全不同的存储器共用时钟、读写地址，并且请将两个存储器读出的结果分别用 2 个七段数码管显示。

首先使用 initial 语块对我们的寄存器堆进行初始化，这里注释部分是另一种初始化方法，我们使用文件形式对 ram1 赋初值，代码和文件内容如下：

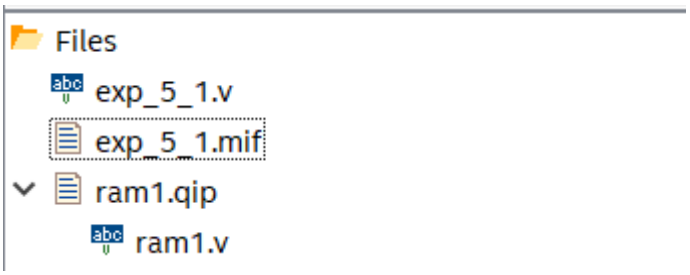
```
initial
   begin
      //ram[7] = 8'hf0; ram[6] = 8'h23; ram[5] = 8'h20; ram[4] = 8'h50;
      //ram[3] = 8'h03; ram[2] = 8'h21; ram[1] = 8'h82; ram[0] = 8'h0D;
      $readmemh("D:/My_design/exp_5_1/mem1.txt", ram1, 0, 15);
   end
```

寄存器堆的读写操作代码如下：

```
always @(posedge clk) begin

   if (we_1==0)
      ram1[addr] <= din;

end
```

```
always @(ram1[addr]) begin

        out_1 = ram1[addr];
        case(out_1%10)
            0:led0=7'b1000000;
            1:led0=7'b1111001;
            2:led0=7'b0100100;
            3:led0=7'b0110000;
            4:led0=7'b0011001;
            5:led0=7'b0010010;
            6:led0=7'b0000010;
            7:led0=7'b1111000;
            8:led0=7'b0000000;
            9:led0=7'b0010000;
        endcase

        case((out_1/10)%10)
            0:led1=7'b1000000;
            1:led1=7'b1111001;
            2:led1=7'b0100100;
            3:led1=7'b0110000;
            4:led1=7'b0011001;
            5:led1=7'b0010010;
            6:led1=7'b0000010;
            7:led1=7'b1111000;
            8:led1=7'b0000000;
            9:led1=7'b0010000;
        endcase
```

接下来我们使用 IP 核实现存储器，Quartus 提供了很多非常实用的 IP 核，利用这些 IP 核可以很方便的实现复杂的设计。根据手册上的操作我们得到了 RAM 对应的生产文件：



并在.mif 文件中对其初始化：

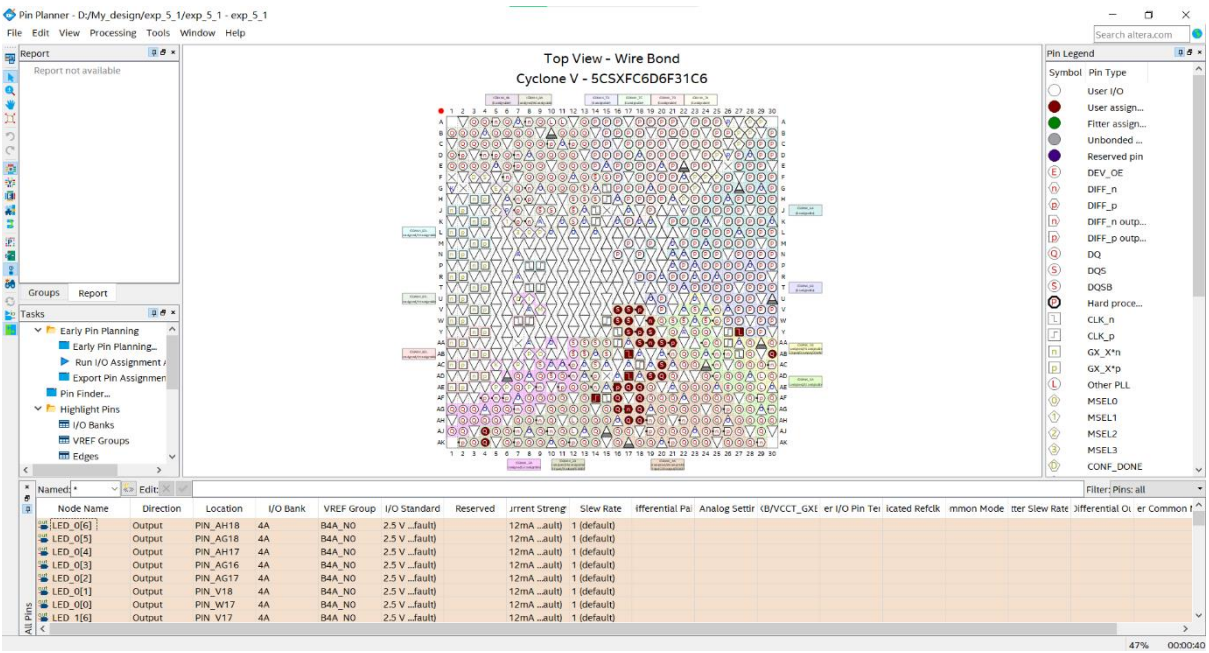
Addr	+0	+1	+2	+3	+4	+5	+6	+7	ASCII
0	240	241	242	243	244	245	246	247	.....
8	248	249	250	251	252	253	254	255	.....

接下来在顶层实体中对此 RAM 进行实例化即可使用，代码如下：

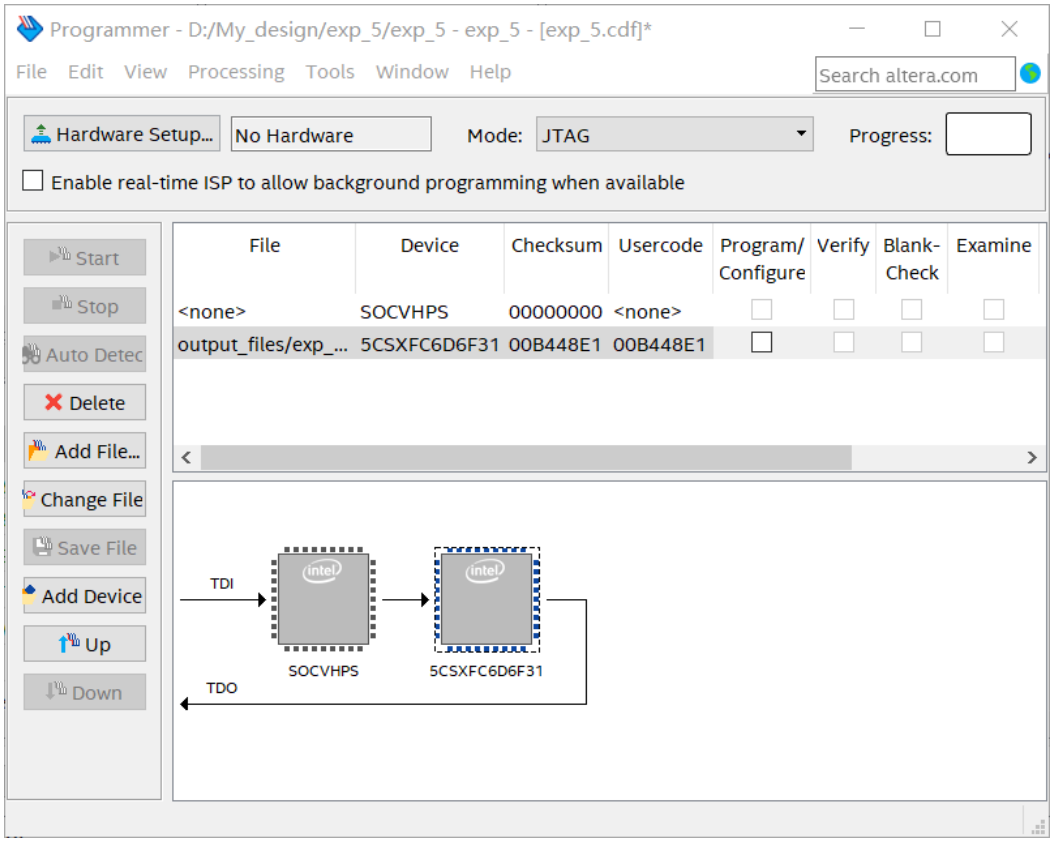
```
ram1 ram2(
    .address(addr),
    .clock(clk),
    .data(din),
    .wren(~we_2),
    .q(out_2)
    );

    reg [7:0] ram1 [15:0];
    reg [7:0] out_1;
    reg [6:0] led0=0;
    reg [6:0] led1=0;
    reg [6:0] led2=0;
    reg [6:0] led3=0;
```

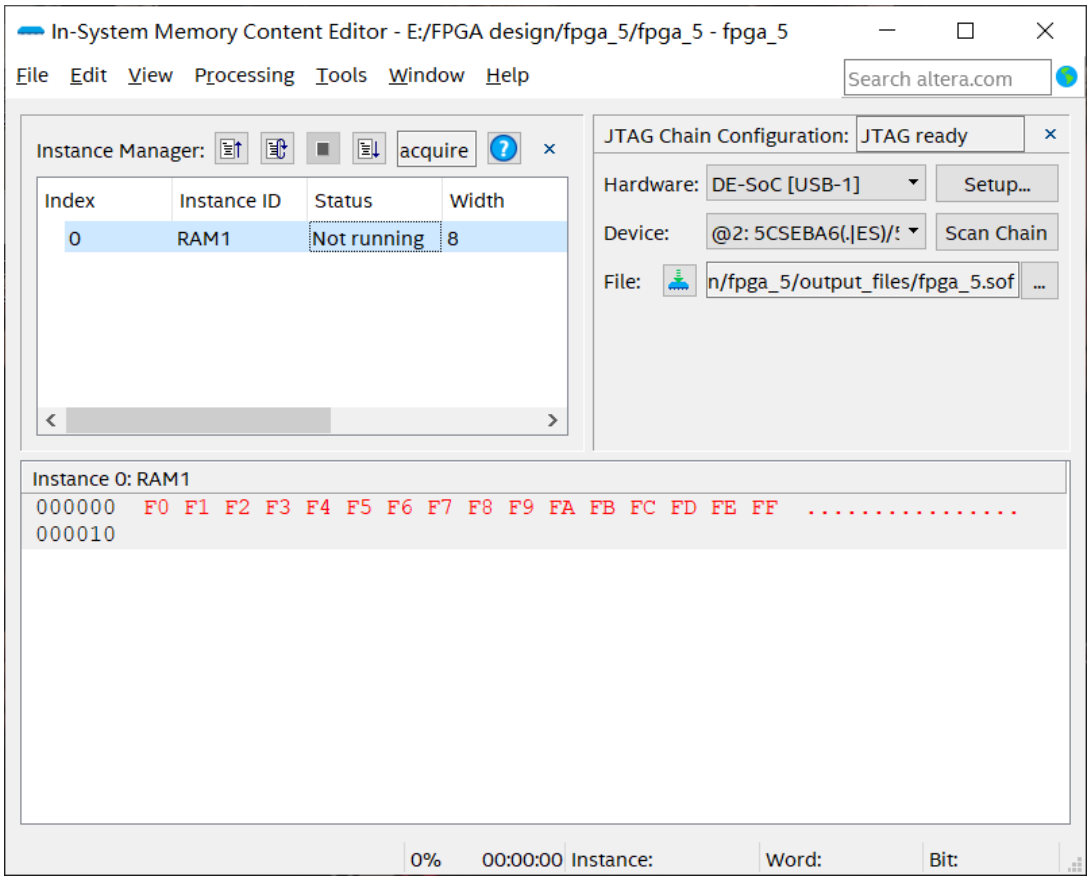
注意我们设计的写使能对应开发板上的按钮，按下去为 0，自然状态为 1，因此我们将使能设置为 0 的时候有效，所以这里实例化的时候用到了取反操作。

然后进行引脚分配，编译运行，并烧录到开发板上：



注意我们还需要对存储器的内容进行动态更新，检测其修改是否真正更新到开发板中。


Programmer - D:/My_design/exp_5/exp_5 - exp_5 - [exp_5.cdf]*
File Edit View Processing Tools Window Help
Search altera.com
Hardware Setup...
No Hardware
Mode: JTAG
Progress:
Enable real-time ISP to allow background programming when available
Start
Stop
Auto Detec
Delete
Add File...
Change File
Save File
Add Device
Up
Down
File
Device
Checksum
Usercode
Program/ Configure
Verify
Blank-Check
Examine
<none>
SOCVHPS
00000000
<none>
output_files/exp_...
5CSXFC6D6F31
00B448E1
00B448E1
TDI
TDO
SOCVHPS
5CSXFC6D6F31


In-System Memory Content Editor - E:/FPGA design/fpga_5/fpga_5 - fpga_5
File Edit View Processing Tools Window Help
Search altera.com
Instance Manager:
acquire
Index
Instance ID
Status
Width
0
RAM1
Not running
8
JTAG Chain Configuration:
JTAG ready
Hardware:
DE-SoC [USB-1]
Setup...
Device:
@2: 5CSEBA6(.|ES)/
Scan Chain
File:
n/fpga_5/output_files/fpga_5.sof
Instance 0: RAM1
000000 F0 F1 F2 F3 F4 F5 F6 F7 F8 F9 FA FB FC FD FE FF
000010
0%
00:00:00
Instance:
Word:
Bit:

五、 实验结果

1. 思考题

☞ **思考题**

上述存储器综合时，综合器是否会用 FPGA 的 RAM 模块来实现这个模块？如果将表5-4中存储器实现部分改为

```
always @(posedge clk)
      if (we)
         ram[inaddr] <= din;
      else
         dout <= ram[outaddr];
```

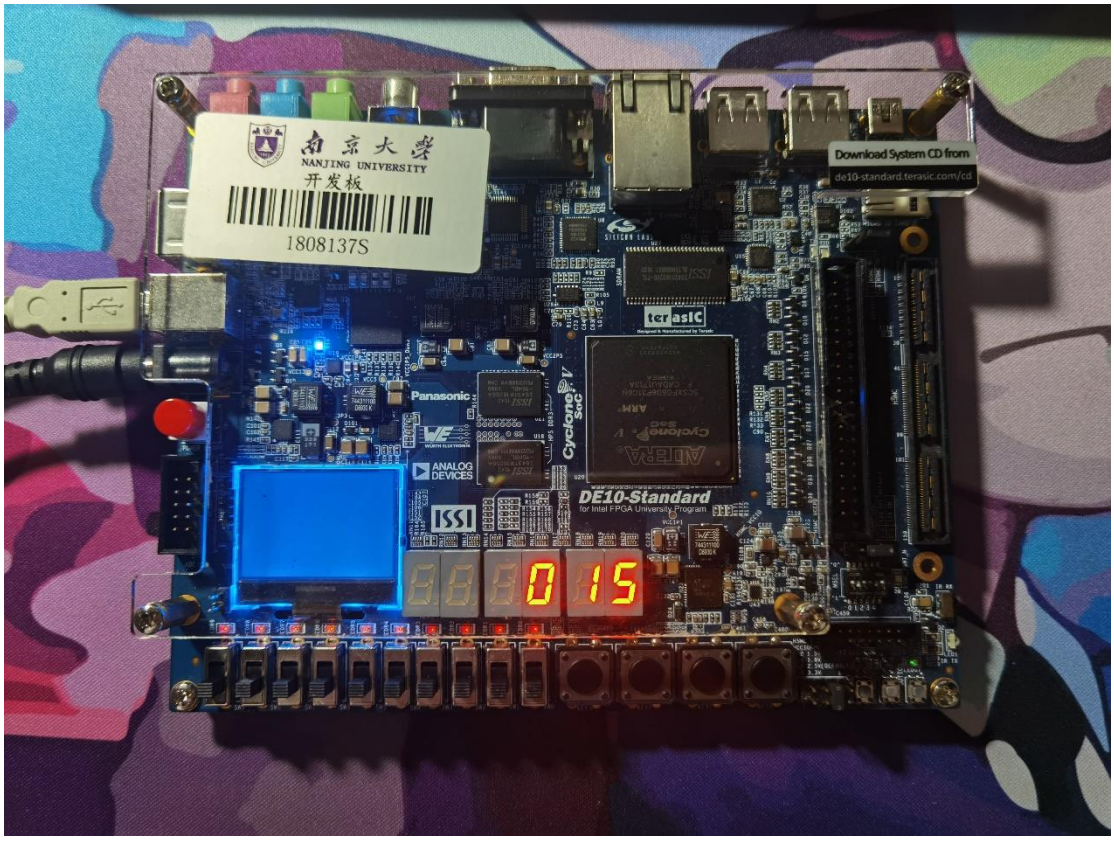
该存储器的行为是否会发生变化？

在存储器示例代码中，读写操作没有严格在时钟信号沿上进行，因此系统会认为该存储单元的读写要求较高，综合器将会直接采用 FPGA 逻辑单元实现，不会用 RAM 模块实现。若读写操作修改为上述代码，则严格在时钟信号沿上读写，又因为存储器的大小是 32*10，故综合器会使用 FPGA 的 RAM 模块来实现。

2. 上板验收（寄存器堆）

我们实现了 16×8 寄存器堆，读取时不需要受时钟控制，直接输出数据，写入时在写使能有效且时钟上升沿时写入。

下图为读取地址为 1111（15）时对应的寄存器内容，初值为 15。完整验收视频已在附件中上传到 cms 网站中。

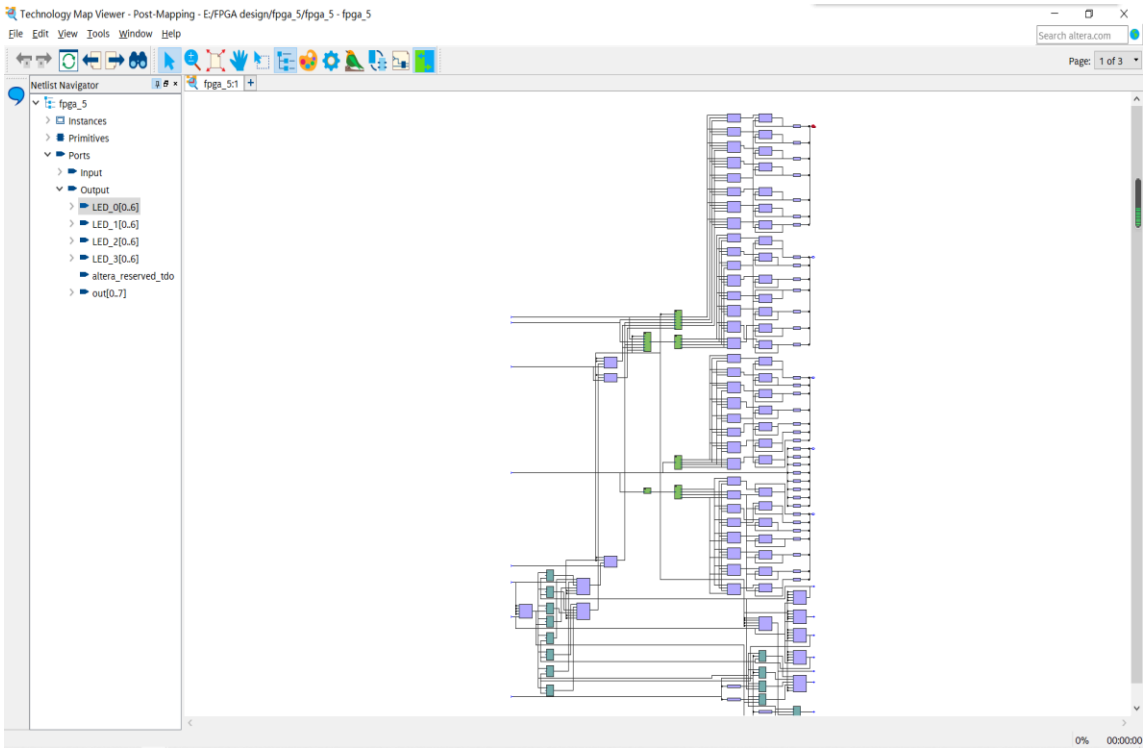


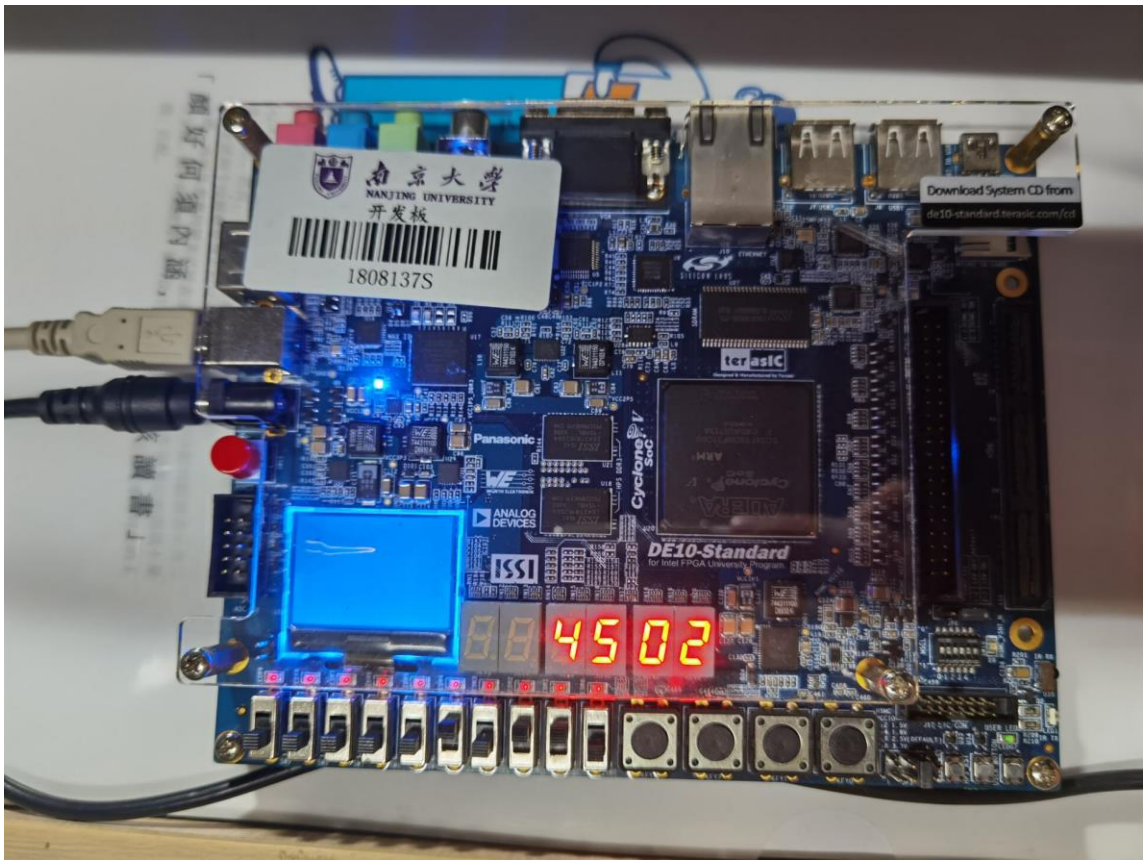
3. 上板验收（存储器）

我们同时实现了一个基于 IP 核的单口存储器，并已初始化以下初值（具体步骤在实验过程部分）：

0xf0, 0xf1, 0xf2, 0xf3, 0xf4, 0xf5, 0xf6,0xf7,0xf8, 0xf9, 0xfa, 0xfb, 0xfc, 0xfd, 0xfe, 0xff

- 上述过程中已在 In-System Memory Content Editor 来修改 RAM 中的数据，以验证修改已经更新到开发板上；
- 将时钟设置为开发板上的按钮，观察到寄存器堆中读取不需要等待时钟，写入需要一个周期；IP 核存储器需要两个时钟周期进行读取写入操作。
- 根据树形结构（如下图）观察两种存储器，分别用了 FPGA 的逻辑电路和 RAM 两种方法。


Technology Map Viewer - Post-Mapping - E:/FPGA design/fpga_5/fpga_5 - fpga_5
File Edit View Tools Window Help
Netlist Navigator
fpga_5
Instances
Primitives
Ports
Input
Output
LED_0[0..6]
LED_1[0..6]
LED_2[0..6]
LED_3[0..6]
altera_reserved_tdo
out[0..7]
fpga_5:1
Page: 1 of 3
0%
00:00:00


南京大学
NANJING UNIVERSITY
开发板
1808137S
Download System CD from
de10-standard.terasic.com/fcd
terasIC
Panasonic
ANALOG DEVICES
ISSI
Cyclone V SoC
DE10-Standard
for Intel FPGA University Program
4502

六、 总结与反思

本次实验中我们掌握了 FPGA 的触发器及片上存储器的特性，并利用 FPGA IP 核和逻辑单元分别实现了寄存器堆和存储器，最后验证我们的存储器单元。但在调用 IP 核模块的时候遇到了许多问题，后来通过查找报错标头找到对应解决方法，总体来说实验比较顺利，希望再接再厉！