《数字电路与数字系统实验》实验报告

第 8 次实验： 存储器

姓名： 张逸凯

学号： 171840708

院系: 物理 学院

邮箱： 645064582@qq.com

电话： 18051988316

实验时间: 2019 年 5 月 3 日

预习部分

基本定义

2019年春季学期

存储器（Memory）是电子设备中的记忆器件，用来存放程序和数据。电子设备中全部信息，包括输入的原始数据、程序、中间运行结果和最终运行结果都保存在存储器中。

动态存储器工作原理

动态存储器每片只有一条输入数据线，而地址引脚只有 8 条。为了形成 64K 地址，必须在系统地址总线和芯片地址引线之间专门设计一个地址形成电路。使系统地址总线信号能分时地加到 8 个地址的引脚上，借助芯片内部的行锁存器、列锁存器和译码电路选定芯片内的存储单元，锁存信号也靠着外部地址电路产生。

当要从 DRAM 芯片中读出数据时，CPU 首先将行地址加在 A0-A7 上，而后送出 RAS 锁存信号，该信号的下降沿将地址锁存在芯片内部。接着将列地址加到芯片的 A0-A7 上，再送 CAS 锁存信号，也是在信号的下降沿将列地址锁存在芯片内部。然后保持 WE=1，则在 CAS 有效期间数据输出并保持。

当需要把数据写入芯片时，行列地址先后将 RAS 和 CAS 锁存在芯片内部，然后，WE 有效，加上要写入的数据，则将该数据写入选中的存贮单元。

由于电容不可能长期保持电荷不变，必须定时对动态存储电路的各存储单元执行重读操作，以保持电荷稳定，这个过程称为动态存储器刷新。PC/XT 机中 DRAM 的刷新是利用 DMA 实现的。首先应用可编程定时器 8253 的计数器 1，每隔 15.12μs 产生一次 DMA 请求，该请求加在 DMA 控制器的 0 通道上。当 DMA 控制器 0 通道的请求得到响应时，DMA 控制器送到刷新地址信号，对动态存储器执行读操作，每读一次刷新一行。

对于存储器，其读写时序非常重要，也是实践中容易出错的地方。读取数据时在哪个时间点数据有效，写入数据过多久可以读取这些都要在设计时反复检查。

- **PDF 中思考题**

☞ 思考题

如果将表 7-2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?

先看看原来的代码:

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

   assign dout = ram[outaddr];
```

根据 PDF 我们知道，原来代码中存储器读数据是不受时钟和使能端的控制的，因为 assign 是连续赋值语句，所以只要输出地址有效，输出地址所指向单元的数据就会被立即送到输出总线上。而***在修改代码以后存储器的读数据操作便受到了时钟和使能端的控制，只有在时钟有效并且使能端无效的情况下***输出地址所指向单元的数据才会被送到输出总线上。改之前和改之后有一定的差别.

下面对存储器的实现做一个简单的预习

表 7-3: 存储器实例代码

```
module v_rams_8 (clk, we, inaddr,outaddr, din, dout0,dout1,dout2);
input clk;
input we;
input [2:0] inaddr;
input [2:0] outaddr;
input [7:0] din;
output [7:0] dout0,dout1,dout2;

reg [7:0] ram [7:0];
reg [7:0] dout0,dout1;

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;
end

always @(posedge clk)
begin
    if (we)
        ram[inaddr] <= din;
    else
        dout0 <= ram[outaddr];
end
always @(negedge clk)
begin
    if (!we)
        dout1 <= ram[outaddr];
end
assign  dout2 = ram[outaddr];
endmodule
```

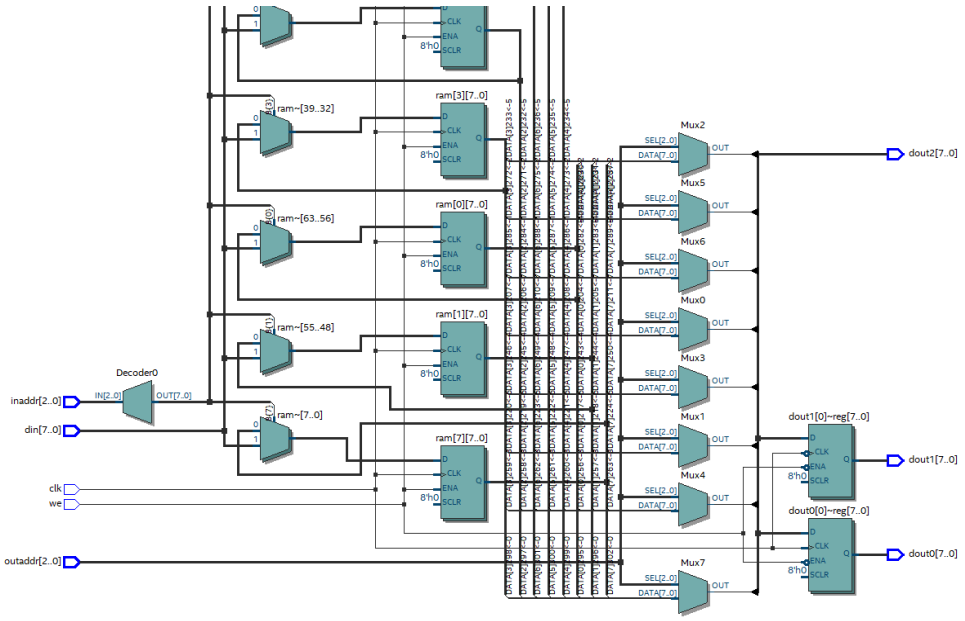
initial 在begin语句块之前 只能一次

1. 输出端 dout0 是在时钟上升沿来临且使能端无效时读取数据的
2. dout1 是在时钟下降沿来临且使能端无效时读取数据
3. dout2 是只要输出地址有效就进行读数据操作.

当时钟上升沿来临且使能端有效时进行的是存储器的写操作。该存储器的初始化是由一个 initial 语句完成的。

理解观察 RTL 图发现该存储器的工作原理是：在时钟上升沿有效且使能端有效时，由 inaddr[2:0]决定输入的数据 din 到底要输入到那个存储器中。

而在输出的时候，outaddr 相当于是选择器，因为 8 个存储器，其中每个存储器的第 0 位数据（合起来共 8 位数据），第一位数据等全部合在一起了，然后由 outaddr 进行选择到底输出哪个存储器的相应的 8 位数据，然后就可以得到那个存储器的数据了



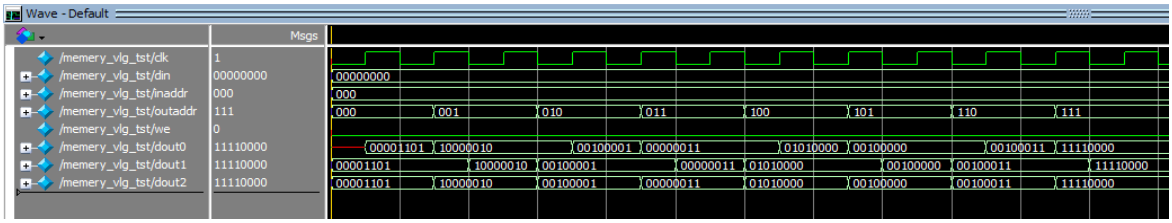
结合这段话理解：

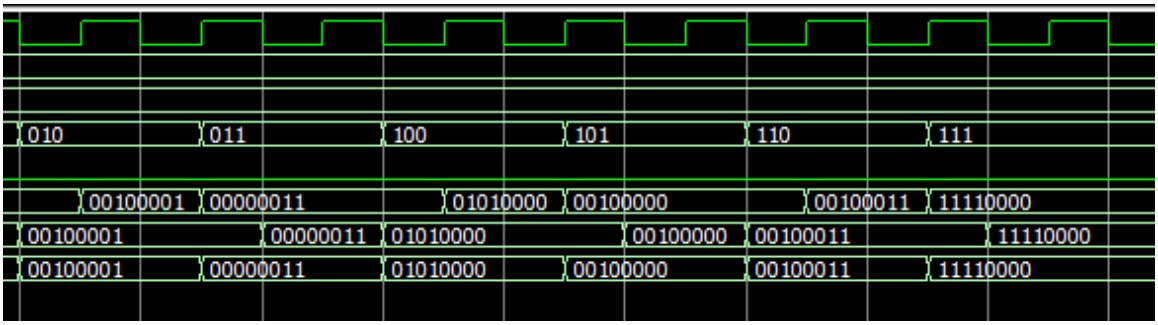
读数据：存储器的输出可以受时钟和使能端的控制，也可以不受时钟和使能端的控制。如果输出受时钟的控制，则在时钟有效沿，将输出地址所指示的单元中的数据，输出到输出总线上（Data_out）；如果不受时钟的控制，则只要输出地址有效，就立即将此地址所指的单元中的数据送到输出总线上。

存储器的三个输出端的有效条件是不一样的，从 RTL 图也可以看出来，其中 dout2 没有约束条件，只要输出地址有效就可以马上将数据输出到总线上，而 dout1 要时钟下降沿加上使能端无效才能输出数据，而 dout0 要时钟有效加上使能端无效才能输出数据。下面验证三个输出端口在时序上的差别，编写测试代码来进行验证：

```
initial
begin
    we = 0; clk = 0; inaddr = 3'b000; din = 8'b00000000; outaddr = 3'b000; #3;
    outaddr = 3'b001; #3;
    outaddr = 3'b010; #3;
    outaddr = 3'b011; #3;
    outaddr = 3'b100; #3;
    outaddr = 3'b101; #3;
    outaddr = 3'b110; #3;
    outaddr = 3'b111; #3;
end
always
begin
    #1 clk = ~clk;
end
endmodule
```

➢ 仿真结果:





从仿真结果看，三个输出端的工作时序确实是上述分析的那样 dout2 只要输出地址有效就会输出数据，dout1 要在时钟下降沿加上使能端无效时才会输出数据，而 dout0 要在时钟上升沿加上使能端无效时才会输出数据。

开始实验部分

〇． 实验目的

a) 了解FPGA的片上存储器的存储器特性

b) 分析存储器的工作时序和结构

c) 学习如何设计存储器

一． 实验原理（ 知识背景，结合理论课总结 ）

- **DRAM 工作原理：**

动态存储器每片只有一条输入数据线，而地址引脚只有 8 条。为了形成 64K 地址，必须在系统地址总线和芯片地址引线之间专门设计一个地址形成电路。使系统地址总线信号能分时地加到 8 个地址的引脚上，借助芯片内部的行锁存器、列锁存

器和译码电路选定芯片内的存储单元，锁存信号也靠着外部地址电路产生。

当要从 DRAM 芯片中读出数据时，CPU 首先将行地址加在 A0-A7 上，而后送出 RAS 锁存信号，该信号的下降沿将地址锁存在芯片内部。接着将列地址加到芯片的 A0-A7 上，再送 CAS 锁存信号，也是在信号的下降沿将列地址锁存在芯片内部。然后保持 WE=1，则在 CAS 有效期间数据输出并保持。

当需要把数据写入芯片时，行列地址先后将 RAS 和 CAS 锁存在芯片内部，然后，WE 有效，加上要写入的数据，则将该数据写入选中的存贮单元。

二． 实验设备环境

硬件器材：FPGA 开发板．

软件平台：Qaurtus 开发平台．

三． 实验步骤 / 过程（设计思路、设计代码、测试代码、仿真结果和硬件实现等的截图代码等）

实验一：自行设计 8x8 的存储器

➢ 设计思路：

基于 PPT 例子（ 和查找资料 ）

根据表 7-3 可以很容易的得到此次实验的设计思路：

设置一个使能端，在使能端有效时进行写操作，在使能端无时，且读地址有效时进行读操作（实验一中只有一个输出端）即可。然后利用.txt 文件初始化存储器的内容即可。

设计代码

```
module ram1(clk, we, inaddr, outaddr, din, dout);
input clk;
input we;
input [3:0] inaddr;
input [3:0] outaddr;
input [7:0] din;
output reg [7:0] dout;

reg [7:0] ram [7:0];

initial
begin
   $readmemh("C:/Users/Kai/Desktop/memoo.txt", ram, 0, 7);
end

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

endmodule
```

由于实验一和实验二是在同一个工程中，所以测试代码、仿真结果以及硬件实等的截图就放在后面一起做了

实验二：利用 IP 核生成 8×8 的存储器

- *设计思路*

实验二的设计思路就比较简单了，只要按照 PDF 上给出的指导一步一步的做下去即可得到利用 IP 核生成的存储器。然后利用.mif 文件初始化即可

.mif 文件：

dc	+0	+1	+2	+3	+4	+5	+6	+7	ASCII
0	00	01	02	03	04	05	06	07	.....

主文件代码：

```
module memo(clk, we, inoutaddr, inoutadd0, din, dout1, dout2);
   input clk;
   input we;
   input [3:0] inoutaddr;
   input [3:0] inoutadd0;
   wire [3:0] addr;
   input [7:0] din;
   output [7:0] dout1;
   output [7:0] dout2;
   assign addr = inoutaddr;
   assign add0 = inoutadd0;
   ram1 S(.clk(clk), .we(we), .inaddr(inoutaddr), .outaddr(addr), .din(din), .dout(dout1));
   ram2port A(.clock(clk), .data(din), .wren(we), .wraddress(inoutadd0), .rdaddress(add0), .q(dout2));
endmodule
```

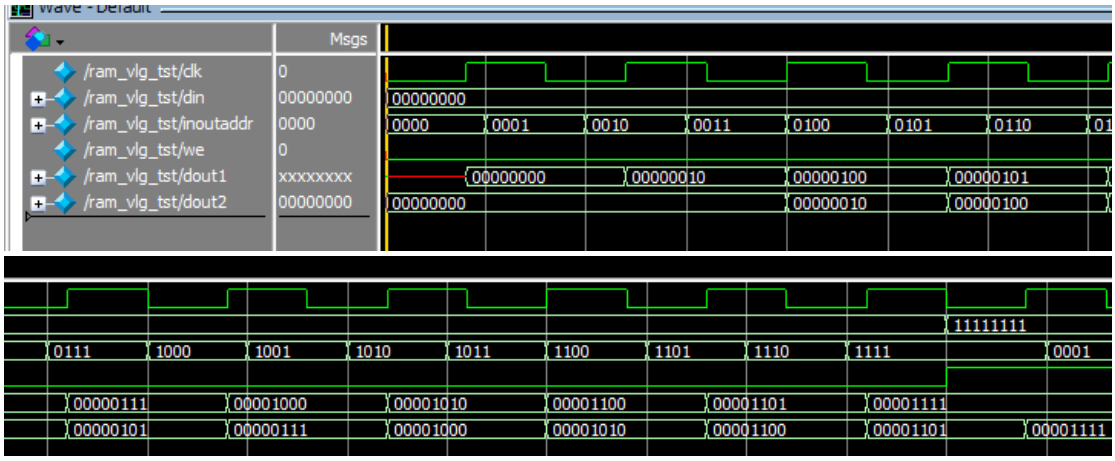
测试代码：

```
begin
    // 读数据验证存储器初始化.
    we = 0; clk = 0; din = 8'b00000000; inoutaddr = 4'b0000; #5;
    inoutaddr = 4'b0001; #5;
    inoutaddr = 4'b0010; #5;
    inoutaddr = 4'b0011; #5;
    inoutaddr = 4'b0100; #5;
    inoutaddr = 4'b0101; #5;
    inoutaddr = 4'b0110; #5;
    inoutaddr = 4'b0111; #5;
    inoutaddr = 4'b1000; #5;
    inoutaddr = 4'b1001; #5;
    inoutaddr = 4'b1010; #5;
    inoutaddr = 4'b1011; #5;
    inoutaddr = 4'b1100; #5;
    inoutaddr = 4'b1101; #5;
    inoutaddr = 4'b1110; #5;
    inoutaddr = 4'b1111; #5;

    // 修改存储器里面的内容
    we = 1; din = 8'b11111111; #5;
    inoutaddr = 4'b0001; #5;
    inoutaddr = 4'b0010; #5;
    inoutaddr = 4'b0011; #5;
    inoutaddr = 4'b0100; #5;
    inoutaddr = 4'b0101; #5;
    inoutaddr = 4'b0110; #5;
    inoutaddr = 4'b0111; #5;
    inoutaddr = 4'b1000; #5;
    inoutaddr = 4'b1001; #5;
    inoutaddr = 4'b1010; #5;
    inoutaddr = 4'b1011; #5;
    inoutaddr = 4'b1100; #5;
    inoutaddr = 4'b1101; #5;
    inoutaddr = 4'b1110; #5;
    inoutaddr = 4'b1111; #5;

    // 再读存储器修改后的内容.
    we = 0; #5;
    inoutaddr = 4'b0001; #5;
```

```
55      // 再读存储器修改后的内容.
56      we = 0; #5;
57      inoutaddr = 4'b0001; #5;
58      inoutaddr = 4'b0010; #5;
59      inoutaddr = 4'b0011; #5;
60      inoutaddr = 4'b0100; #5;
61      inoutaddr = 4'b0101; #5;
62      inoutaddr = 4'b0110; #5;
63      inoutaddr = 4'b0111; #5;
64      inoutaddr = 4'b1000; #5;
65      inoutaddr = 4'b1001; #5;
66      inoutaddr = 4'b1010; #5;
67      inoutaddr = 4'b1011; #5;
68      inoutaddr = 4'b1100; #5;
69      inoutaddr = 4'b1101; #5;
70      inoutaddr = 4'b1110; #5;
71      inoutaddr = 4'b1111; #5;
72  end
73  always
74  begin
75      #4 clk = ~clk;
76  end
77  endmodule
```

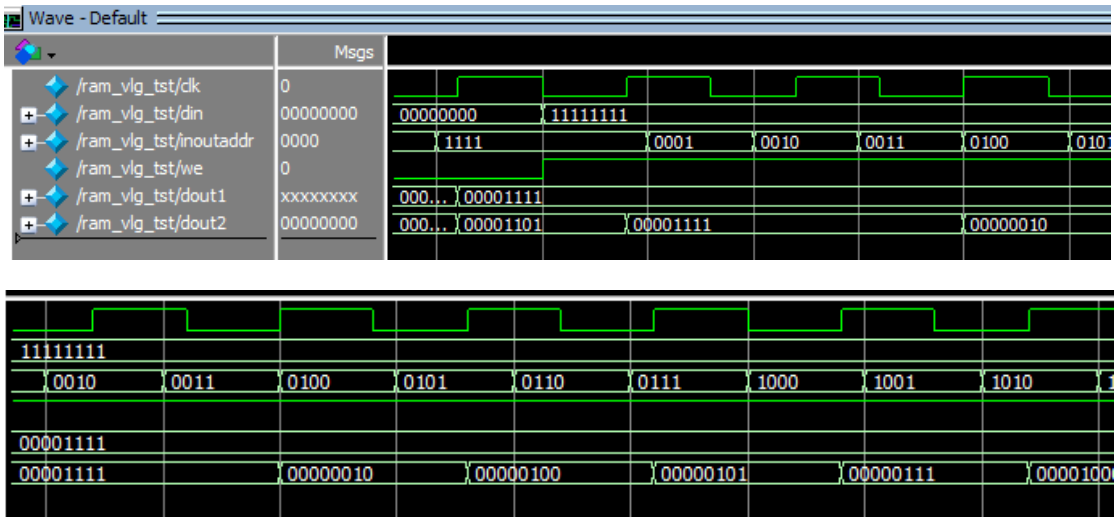
ModelSim 仿真波形：

- 测试代码的思路：先仿真存储器初始化的结果，验证存储器初始化是否正确，然后再进行写操作，之后再进行读操作，验证存储器写操作的正确性。

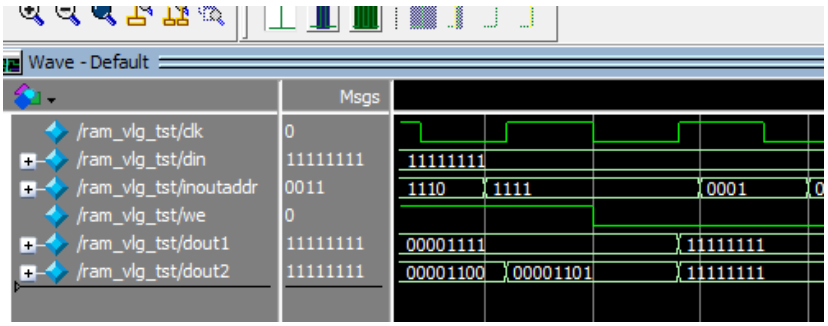


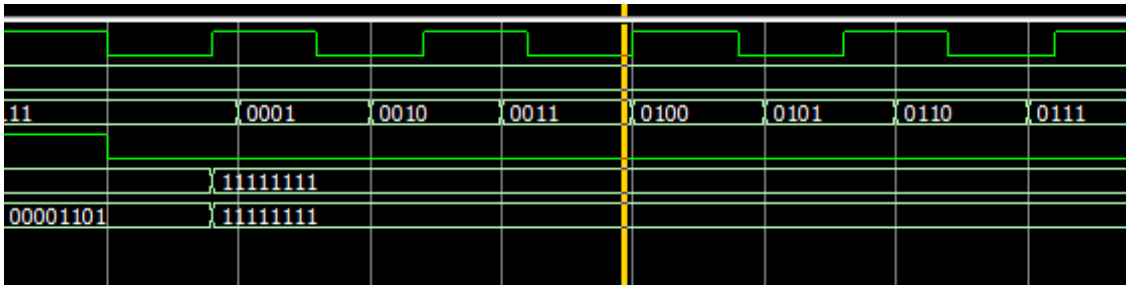
因为利用 IP 核生成的存储器是有一个锁存的，所以读需要两个周期。

之后使能端变为：1，进行存储器的写操作。



通过写地址将存储器的每个内容都写为 1111 1111。
然后再将使能端变为：0，进行读操作，验证存储器的写操作的正确性。





从上述仿真结果可以看出进行写操作后，两个存储器中的所有内容都被修改为了 1111 1111，从而说明存储器的写操作是正确的。

➢ 硬件实现：

引脚分配：

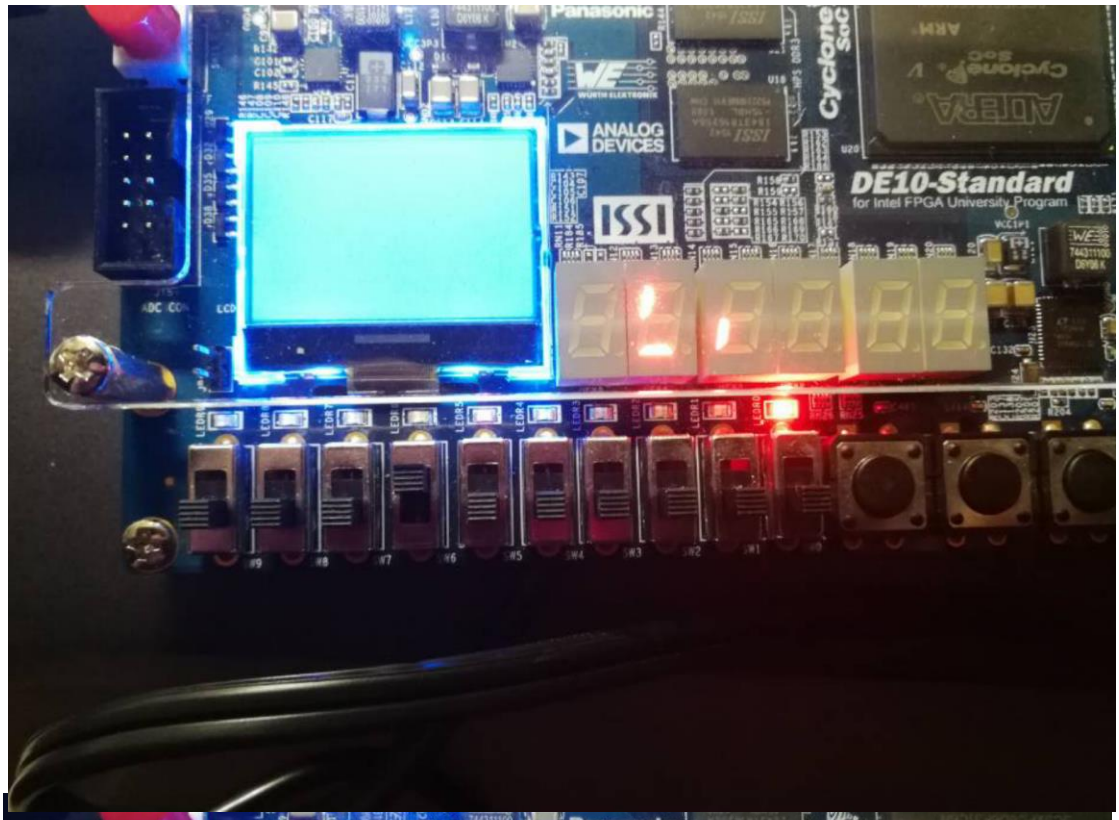
引脚说明见开发板实现部分

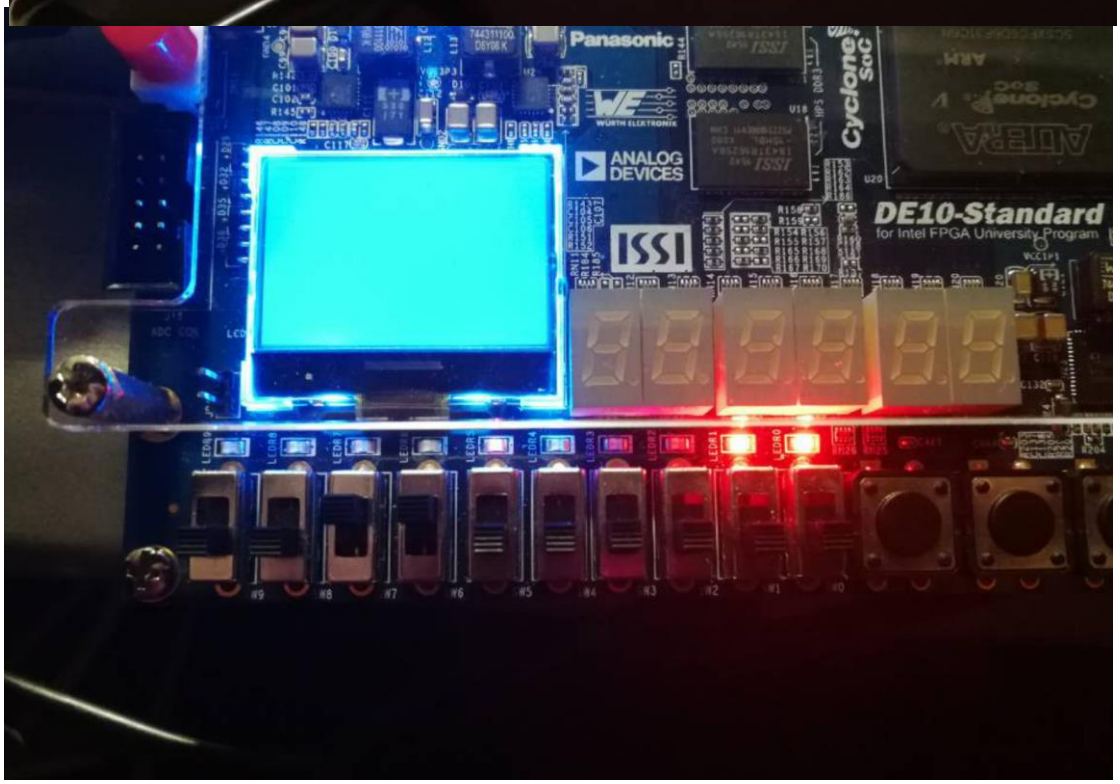
Node Name	Direction	Location	I/O Ban	EF Grc	Stand	eserve	nt S
clk	Input	PIN_AJ4	3B	B...0	2.5...lt)		12...
din[7]	Input				2.5...lt)		12...
din[6]	Input				2.5...lt)		12...
din[5]	Input				2.5...lt)		12...
din[4]	Input				2.5...lt)		12...
din[3]	Input				2.5...lt)		12...
din[2]	Input				2.5...lt)		12...
din[1]	Input	PIN_Y27	5B	B...0	2.5...lt)		12...
din[0]	Input	PIN_AB30	5B	B...0	2.5...lt)		12...
dout1[7]	Output				2.5...lt)		12...
dout1[6]	Output				2.5...lt)		12...
dout1[5]	Output				2.5...lt)		12...
dout1[4]	Output				2.5...lt)		12...
dout1[3]	Output	PIN_AC22	4A	B...0	2.5...lt)		12...
dout1[2]	Output	PIN_AB22	5A	B...0	2.5...lt)		12...
dout1[1]	Output	PIN_AF24	4A	B...0	2.5...lt)		12...
dout1[0]	Output	PIN_AE24	4A	B...0	2.5...lt)		12...
dout2[7]	Output				2.5...lt)		12...
dout2[6]	Output				2.5...lt)		12...
dout2[5]	Output				2.5...lt)		12...
dout2[4]	Output				2.5...lt)		12...
dout2[3]	Output	PIN_AD24	4A	B...0	2.5...lt)		12...
dout2[2]	Output	PIN_AC23	4A	B...0	2.5...lt)		12...
dout2[1]	Output	PIN_AB23	5A	B...0	2.5...lt)		12...
dout2[0]	Output	PIN_AA24	5A	B...0	2.5...lt)		12...
inoutadd0[3]	Input	PIN_AA30	5B	B...0	2.5...lt)		12...
inoutadd0[2]	Input	PIN_AC29	5B	B...0	2.5...lt)		12...
inoutadd0[1]	Input	PIN_AD30	5B	B...0	2.5...lt)		12...
inoutadd0[0]	Input	PIN_AC28	5B	B...0	2.5...lt)		12...
inoutaddr[3]	Input	PIN_V25	5B	B...0	2.5...lt)		12...
inoutaddr[2]	Input	PIN_W25	5B	B...0	2.5...lt)		12...
inoutaddr[1]	Input	PIN_AC30	5B	B...0	2.5...lt)		12...
inoutaddr[0]	Input	PIN_AB28	5B	B...0	2.5...lt)		12...
we	Input	PIN_AK4	3B	B...0	2.5...lt)		12...

All Pins

➢ 开发板实现

因为助教哥已经验收过了，具体功能的一一验证这里就略过了哈，下面是主要的几个功能。



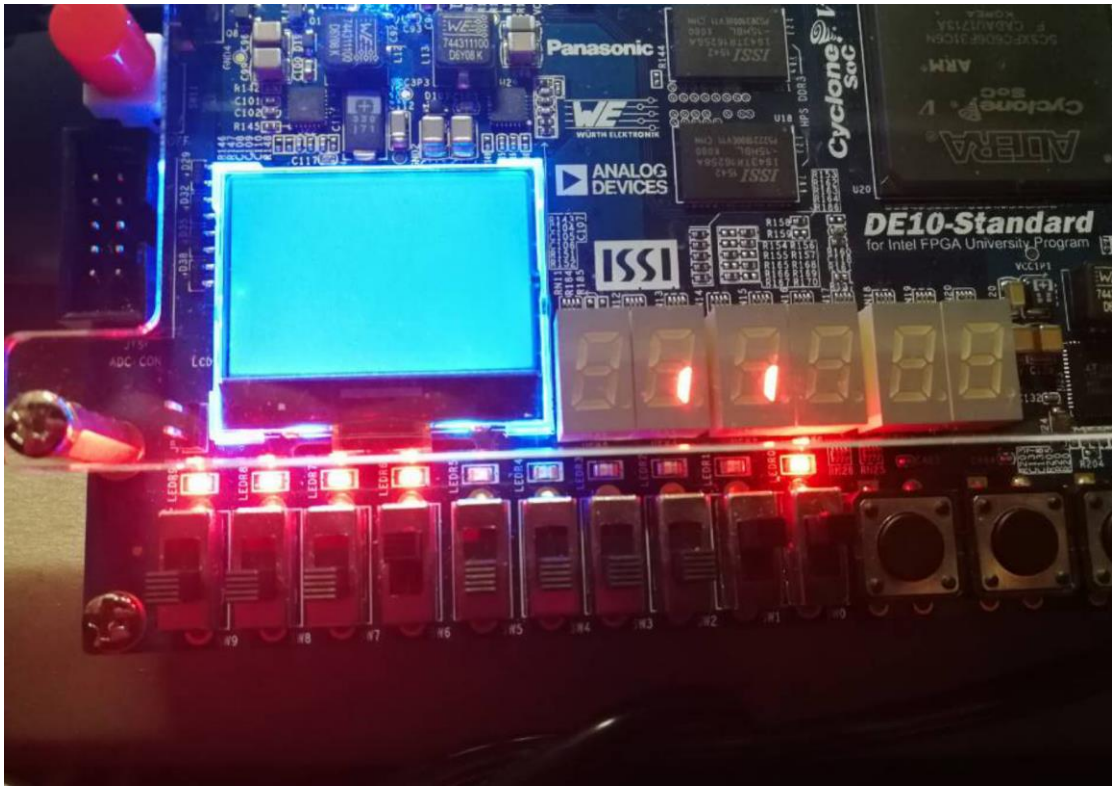


引脚说明：

开关 9~6 是自己写的存储器的 addr_in，开关 5~2 是 IP 核生成的存储器 addr_in. 开关 1~0 是 data_in

按钮 1 是使能端，按钮 0 是 clk，在 LED 灯处显示输出.

以上两张图是数据查询.



上面是将地址处的内存写为 1.

四． 实验中遇到的问题及解决方案（请具体的描述问题和解决方法）

a) LED 有一个问题就是在进行写操作时，我们自己编写的存储器没有亮灯，***而利用 IP 核生成的存储器却灯亮了，***上网查阅资料和询问

老师才发现其实是因为利用 IP 核生成的存储器的使能端只对写操作控制有效，对读操作无效，也即无论使能端怎样都会进行读操作。

b) 通过询问老师和上网查阅资料，对 IP 核存储器进行分析后才知道 IP 核存储器的原理（*里面有个锁存器，读操作需要两个时钟，然后使能端只可以控制写操作*，也即只有使能端有效时才能写，而使能端对读操作是不控制的

五． 实验得到的启示（积极思考）

实验过程中还是要细心一些，不要急，越急越做不好.

在实验中一定要将实验原理弄清楚，效率也比较高.

六． 意见和建议等

这次都挺好的，实现的东西也很有意思！哈哈感觉不错.