



# 南京大學

## 《数字电路与数字系统实验》实验报告

实验七： 存储器

姓名： 毛彦杰

学号： 191220081

班级： 数字电路与数字系统实验 2 班

院系： 计算机科学与技术系

邮箱： 1363818182@qq.com

实验时间： 2020/10/18

## 预习报告：

### 一、存储器

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储单元。

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  $15.12\mu s$  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。

表 7-3: 存储器实例代码

```

1 module v_rams_8 (clk, we, inaddr, outaddr, din, dout0, dout1, dout2);
2   input clk;
3   input we;
4   input [2:0] inaddr;
5   input [2:0] outaddr;
6   input [7:0] din;
7   output [7:0] dout0, dout1, dout2;
8
9   reg [7:0] ram [7:0];
10  reg [7:0] dout0, dout1;
11
12  initial
13  begin
14    ram[7] = 8'hf0; ram[6] = 8'h23; ram[5] = 8'h20; ram[4] = 8'h50;
15    ram[3] = 8'h03; ram[2] = 8'h21; ram[1] = 8'h82; ram[0] = 8'h0D;
16  end
17
18  always @(posedge clk)
19  begin
20    if (we)
21      ram[inaddr] <= din;
22    else
23      dout0 <= ram[outaddr];
24  end
25  always @(negedge clk)
26  begin
27    if (!we)
28      dout1 <= ram[outaddr];
29  end
30  assign dout2 = ram[outaddr];
31 endmodule

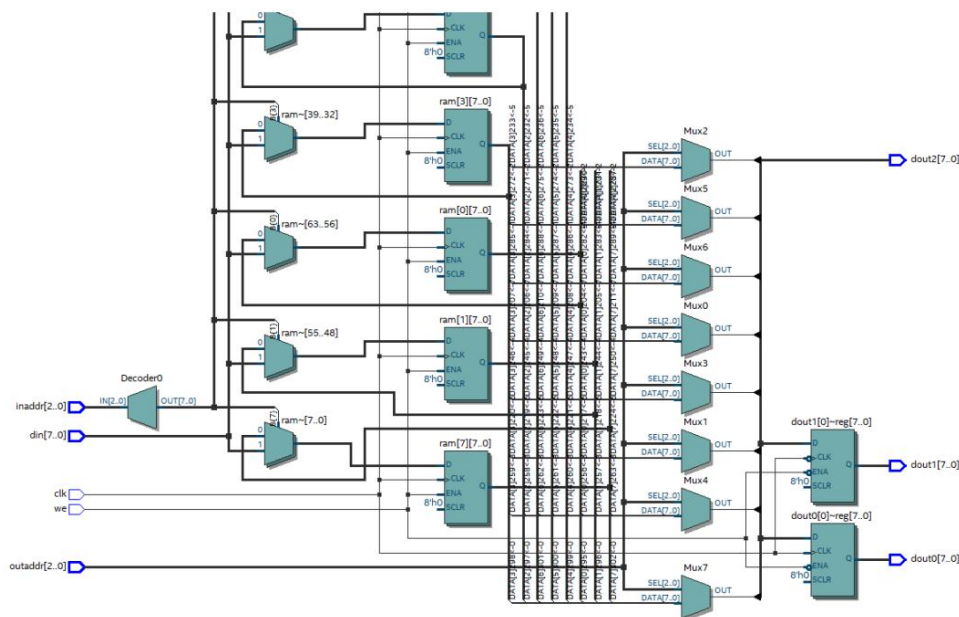
```

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 到底要输入到哪个存储器中。

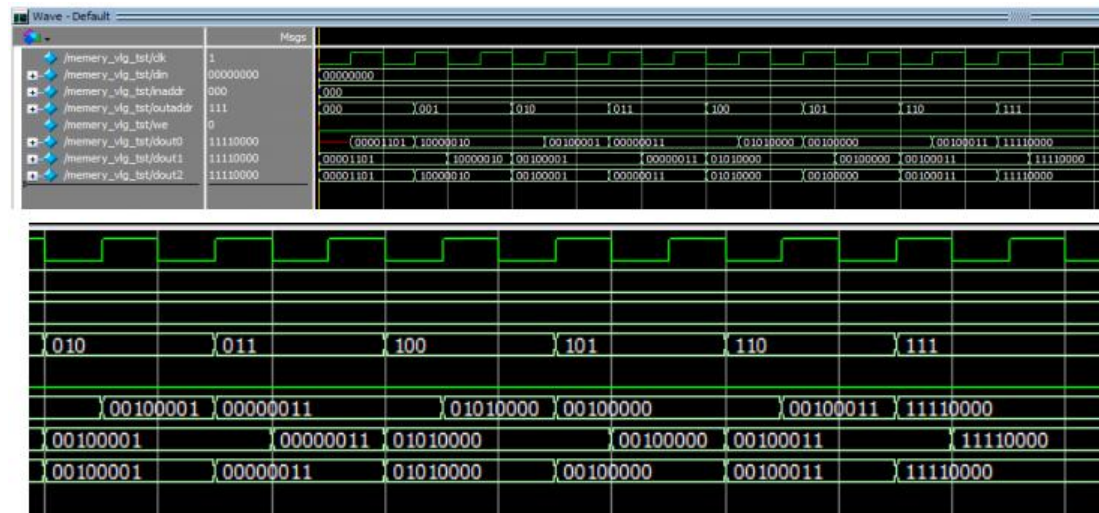
而在输出的时候，outaddr 相当于是选择器，因为 8 个存储器，其中每个存储器的第 0 位数据（合起来共 8 位数据），第一位数据等全部合在一起了，然后由 outaddr 进行选择到底输出哪个存储器的相应的 8 位数据，然后就可以得到那个存储器的数据了



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。

下编写激励代码，观察仿真结果来验证上述过程：

```
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
```



从仿真结果看，dout0 在时钟上升沿与使能端无效时才会输出数据，而 dout1 要在时钟下降沿加上使能端无效时才会输出数据，而 dout2 只要输出地址有效就会输出数据。与上面的描述相符。

## ▪ 思考题

表 7-2: RAM 代码

```
1 module ram #(
2     parameter RAM_WIDTH = 32,
3     parameter RAM_ADDR_WIDTH = 10
4 )(
5     input clk,
6     input we,
7     input [RAM_WIDTH-1:0] din,
8     input [RAM_ADDR_WIDTH-1:0] inaddr,
9     input [RAM_ADDR_WIDTH-1:0] outaddr,
10    output [RAM_WIDTH-1:0] dout
11 );
12
13    reg [RAM_WIDTH:0] ram [(2**RAM_ADDR_WIDTH)-1:0];
14
15    always @(posedge clk)
16        if (we)
17            ram[inaddr] <= din;
18
19    assign dout = ram[outaddr];
20
21 endmodule
```

### 思考题

如果将表 7-2 中存储器实现部分改为

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

该存储器的行为是否会发生变化?

回答: 会发生变化。修改前存储器读数据是不受时钟和使能端控制的, 由于 assign 是连续赋值语句, 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。

## 实验报告：

### 7.4 两个存储器的实现

#### 一、实验目的

在一个工程中完成如下两个存储器。两个存储器的大小均为  $16 \times 8$ , 即每个存储器共有 16 个存储单元, 每个存储单元都是 8 位的, 均可以进行读写。

- **RAM1**: 采用下面的方式进行初始化, 输出端有输出缓存, 输出地址有效后, 等时钟信号的上升沿到来时才输出数据。

```
1 initial
2 begin
3 $readmemh("D:/digital_logic/mem1.txt", ram, 0, 15);
4 end
```

初始化数值为

```
1 @0 00
2 @1 01
3 @2 02
4 @3 03
5 @4 04
6 @5 05
7 @6 06
8 @7 07
9 @8 08
10 @9 09
11 @a 0a
12 @b 0b
13 @c 0c
14 @d 0d
15 @e 0e
16 @f 0f
```

- **RAM2:** 利用 IP 核设计一个双口存储器，利用 .mif 文件进行初始化，十六个单元的初始化值分别为：0xf0, 0xf1, 0xf2, 0xf3, 0xf4, 0xf5, 0xf6, 0xf7, 0xf8, 0xf9, 0xfa, 0xfb, 0xfc, 0xfd, 0xfe, 0xff。

此两个物理上完全不同的存储器共用时钟、读写地址和写使能信号，当写使能有效时，在时钟信号的有效沿写入数据；当写使能信号无效时，在时钟信号的有效沿输出数据。适当选择时钟信号和写使能信号，以能够分别对此两个存储器进行读写。

请合理使用 FPGA 开发板的输入/输出资源，完成此两个存储器的设计。由于开发板上输入数量不够，写入时可以只写入 2 位数据。

## 二、实验原理：动态存储器

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储单元。

### 三、实验设备环境

硬件器材：FPGA 开发板

软件平台：Quartus 开发平台

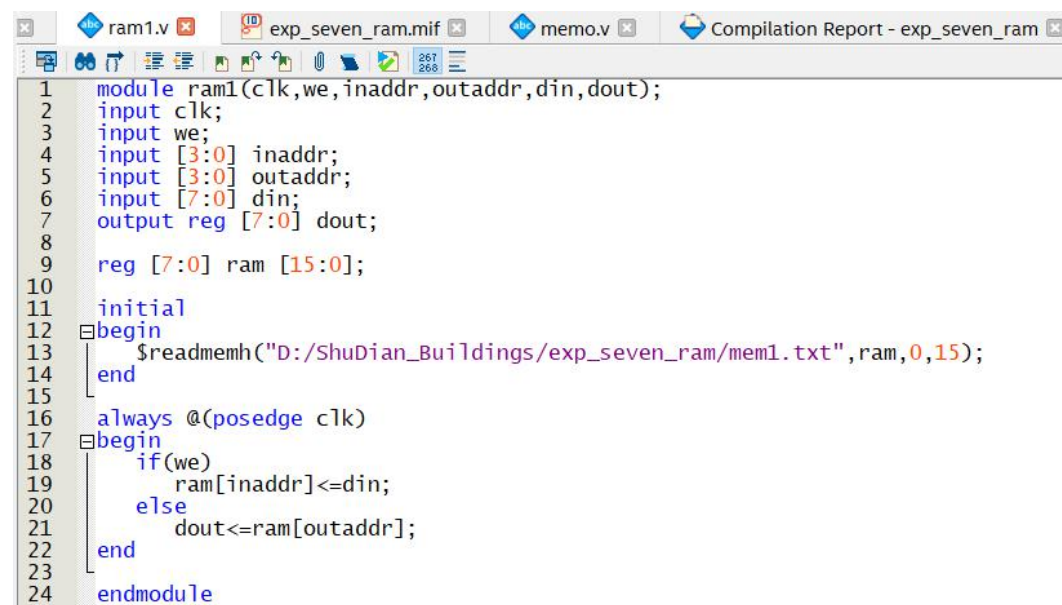
### 四、实验步骤

设计思路：

- RAM1:参考表 7-3 可以得到 RAM1 的设计思路：

设置一个使能端，在使能端有效时进行写操作，在使能端无效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：



```
1 module ram1(clk,we,inaddr,outaddr,din,dout);
2   input clk;
3   input we;
4   input [3:0] inaddr;
5   input [3:0] outaddr;
6   input [7:0] din;
7   output reg [7:0] dout;
8
9   reg [7:0] ram [15:0];
10
11   initial
12   begin
13     $readmemh("D:/ShuDian_Buildings/exp_seven_ram/mem1.txt",ram,0,15);
14   end
15
16   always @(posedge clk)
17   begin
18     if(we)
19       ram[inaddr]<=din;
20     else
21       dout<=ram[outaddr];
22   end
23
24 endmodule
```

- RAM2:参考实验手册给出的步骤即可完成利用 IP 核生成存储器:

.mif 文件用于初始化:

Add	+0	+1	+2	+3	+4	+5	+6	+7	ASCII
0	00	01	02	03	04	05	06	07	.....
8	08	09	10	11	12	13	14	15	.....

主文件设计代码:

```

1  module exp_seven_ram(clk,we,inaddr,outaddr,din,dout1,dout2);
2
3  input clk;
4  input we;
5  input [3:0] inaddr;
6  input [3:0] outaddr;
7  input [7:0] din;
8  output [7:0] dout1,dout2;
9
10 ram1 G(clk,we,inaddr,outaddr,din,dout1);
11 memo F(.clock(clk),.data(din),.rdaddress(inaddr),.wraddress(outaddr),.wren(we),.q(dout2));
12
13 endmodule
14

```

激励代码:

```

27 `timescale 1 ps/ 1 ps
28 module exp_seven_ram_vlg_tst();
29 // constants
30 // general purpose registers
31 reg eachvec;
32 // test vector input registers
33 reg clk;
34 reg [7:0] din;
35 reg [3:0] inaddr;
36 reg [3:0] outaddr;
37 reg we;
38 // wires
39 wire [7:0] dout1;
40 wire [7:0] dout2;
41
42 // assign statements (if any)
43 exp_seven_ram il (
44 // port map - connection between master ports and signals/registers
45 .clk(clk),
46 .din(din),
47 .dout1(dout1),
48 .dout2(dout2),
49 .inaddr(inaddr),
50 .outaddr(outaddr),
51 .we(we)
52 );
53 initial
54 begin
55 // code that executes only once
56 // insert code here --> begin
57 //读数据验证存储器初始化

```

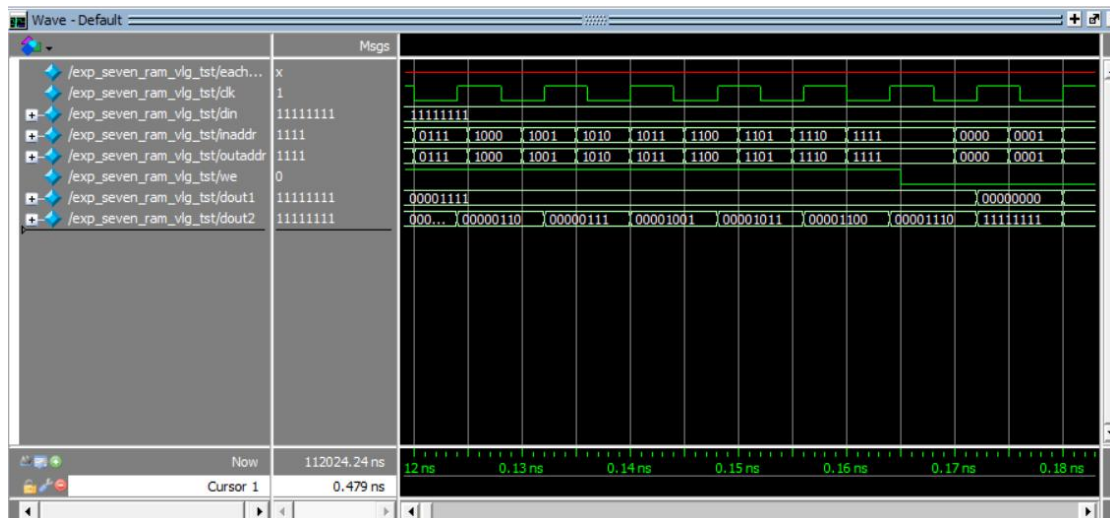
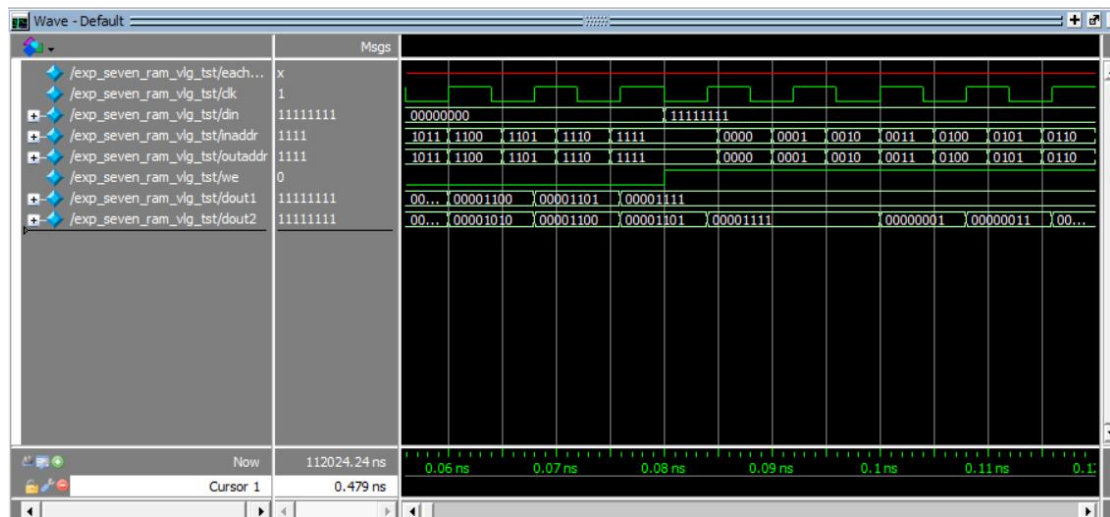
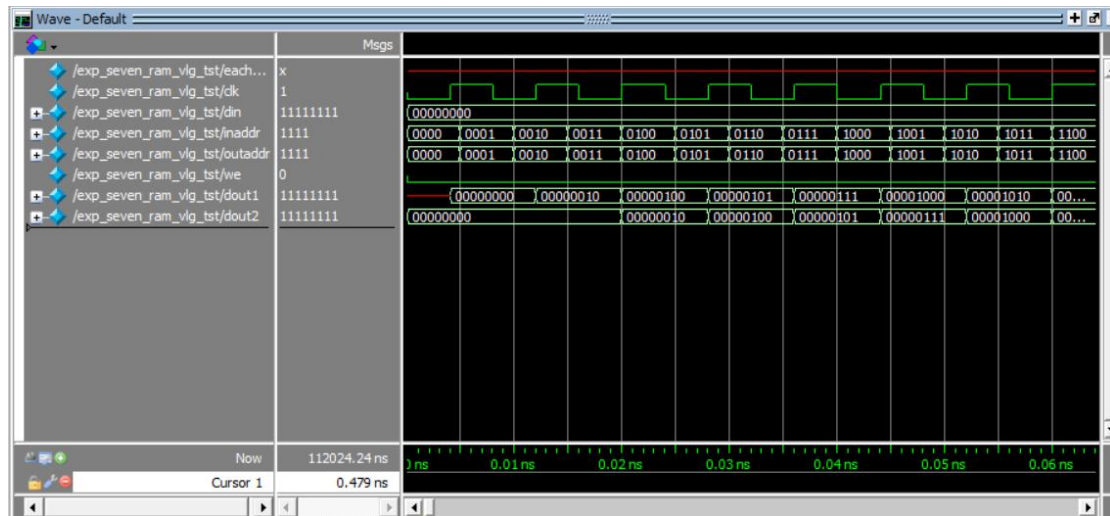
```

58     we=0; clk=0; din=8'b00000000;
59     inaddr=4'b0000; outaddr=4'b0000; #5;
60     inaddr=4'b0001; outaddr=4'b0001; #5;
61     inaddr=4'b0010; outaddr=4'b0010; #5;
62     inaddr=4'b0011; outaddr=4'b0011; #5;
63     inaddr=4'b0100; outaddr=4'b0100; #5;
64     inaddr=4'b0101; outaddr=4'b0101; #5;
65     inaddr=4'b0110; outaddr=4'b0110; #5;
66     inaddr=4'b0111; outaddr=4'b0111; #5;
67     inaddr=4'b1000; outaddr=4'b1000; #5;
68     inaddr=4'b1001; outaddr=4'b1001; #5;
69     inaddr=4'b1010; outaddr=4'b1010; #5;
70     inaddr=4'b1011; outaddr=4'b1011; #5;
71     inaddr=4'b1100; outaddr=4'b1100; #5;
72     inaddr=4'b1101; outaddr=4'b1101; #5;
73     inaddr=4'b1110; outaddr=4'b1110; #5;
74     inaddr=4'b1111; outaddr=4'b1111; #5;
75     //修改存储器里面的内容
76     we=1; din=8'b11111111; #5;
77     inaddr=4'b0000; outaddr=4'b0000; #5;
78     inaddr=4'b0001; outaddr=4'b0001; #5;
79     inaddr=4'b0010; outaddr=4'b0010; #5;
80     inaddr=4'b0011; outaddr=4'b0011; #5;
81     inaddr=4'b0100; outaddr=4'b0100; #5;
82     inaddr=4'b0101; outaddr=4'b0101; #5;
83     inaddr=4'b0110; outaddr=4'b0110; #5;
84     inaddr=4'b0111; outaddr=4'b0111; #5;
85     inaddr=4'b1000; outaddr=4'b1000; #5;
86     inaddr=4'b1001; outaddr=4'b1001; #5;
87     inaddr=4'b1010; outaddr=4'b1010; #5;
88     inaddr=4'b1011; outaddr=4'b1011; #5;
89     inaddr=4'b1100; outaddr=4'b1100; #5;
90     inaddr=4'b1101; outaddr=4'b1101; #5;
91     inaddr=4'b1110; outaddr=4'b1110; #5;
92     inaddr=4'b1111; outaddr=4'b1111; #5;

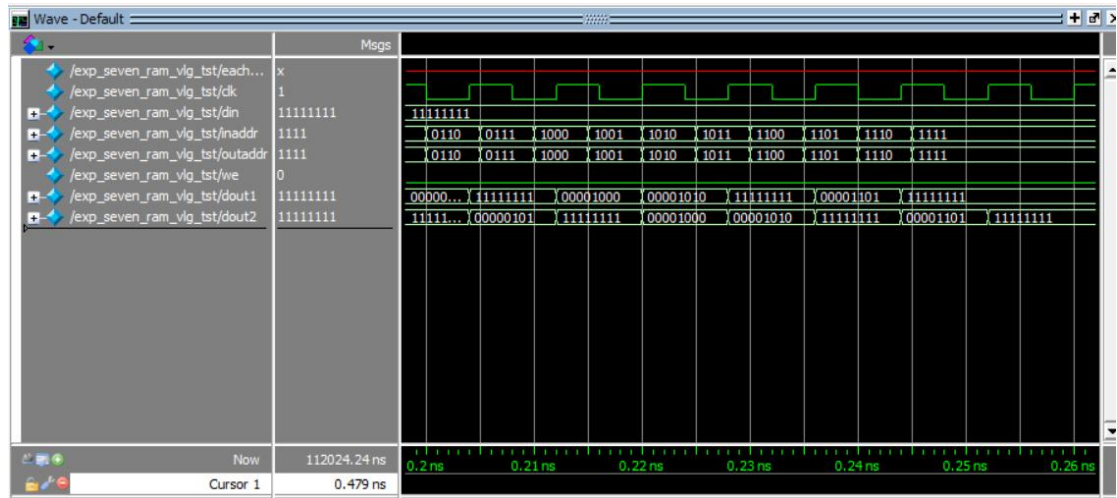
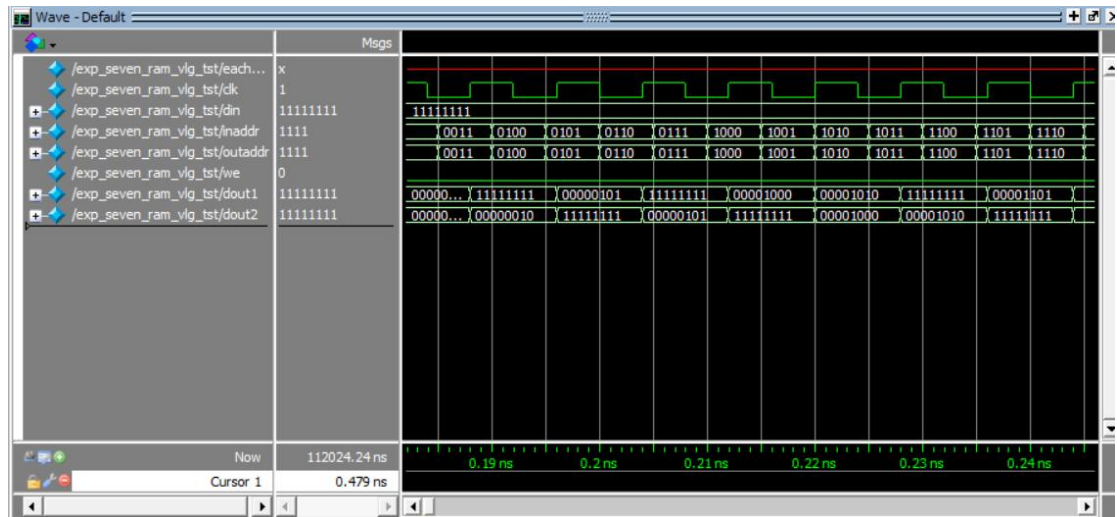
93     //再读存储器修改后的内容
94     we=0; #5;
95     inaddr=4'b0000; outaddr=4'b0000; #5;
96     inaddr=4'b0001; outaddr=4'b0001; #5;
97     inaddr=4'b0010; outaddr=4'b0010; #5;
98     inaddr=4'b0011; outaddr=4'b0011; #5;
99     inaddr=4'b0100; outaddr=4'b0100; #5;
100    inaddr=4'b0101; outaddr=4'b0101; #5;
101    inaddr=4'b0110; outaddr=4'b0110; #5;
102    inaddr=4'b0111; outaddr=4'b0111; #5;
103    inaddr=4'b1000; outaddr=4'b1000; #5;
104    inaddr=4'b1001; outaddr=4'b1001; #5;
105    inaddr=4'b1010; outaddr=4'b1010; #5;
106    inaddr=4'b1011; outaddr=4'b1011; #5;
107    inaddr=4'b1100; outaddr=4'b1100; #5;
108    inaddr=4'b1101; outaddr=4'b1101; #5;
109    inaddr=4'b1110; outaddr=4'b1110; #5;
110    inaddr=4'b1111; outaddr=4'b1111; #5;
111
112    // --> end
113    $display("Running testbench");
114    end
115    always
116    // optional sensitivity list
117    // @(event1 or event2 or .... eventn)
118    //begin
119    // code executes for every event on sensitivity list
120    // insert code here --> begin
121    begin
122    #4 clk=~clk;
123    end
124    //@eachvec;
125    // --> end
126    //end
127    endmodule

```



































## ModelSim 仿真波形：







## 引脚分配:

Node Name	Direction	Location	I/O Bank	VREF Group	I/O Standard
 clk	Input	PIN_AJ4	3B	B3B_N0	2.5 V ...fault)
 din[7]	Input				2.5 V ...fault)
 din[6]	Input				2.5 V ...fault)
 din[5]	Input				2.5 V ...fault)
 din[4]	Input				2.5 V ...fault)
 din[3]	Input				2.5 V ...fault)
 din[2]	Input				2.5 V ...fault)
 din[1]	Input	PIN_Y27	5B	B5B_N0	2.5 V ...fault)
 din[0]	Input	PIN_AB30	5B	B5B_N0	2.5 V ...fault)
 dout1[7]	Output				2.5 V ...fault)
 dout1[6]	Output				2.5 V ...fault)
 dout1[5]	Output				2.5 V ...fault)
 dout1[4]	Output				2.5 V ...fault)
 dout1[3]	Output	PIN_AC22	4A	B4A_N0	2.5 V ...fault)
 dout1[2]	Output	PIN_AB22	5A	B5A_N0	2.5 V ...fault)
 dout1[1]	Output	PIN_AF24	4A	B4A_N0	2.5 V ...fault)
 dout1[0]	Output	PIN_AE24	4A	B4A_N0	2.5 V ...fault)
 dout2[7]	Output				2.5 V ...fault)
 dout2[6]	Output				2.5 V ...fault)
 dout2[5]	Output				2.5 V ...fault)
 dout2[4]	Output				2.5 V ...fault)
 dout2[3]	Output	PIN_AD24	4A	B4A_N0	2.5 V ...fault)
 dout2[2]	Output	PIN_AC23	4A	B4A_N0	2.5 V ...fault)
 dout2[1]	Output	PIN_AB23	5A	B5A_N0	2.5 V ...fault)
 dout2[0]	Output	PIN_AA24	5A	B5A_N0	2.5 V ...fault)
 inaddr[3]	Input	PIN_AA30	5B	B5B_N0	2.5 V ...fault)
 inaddr[2]	Input	PIN_AC29	5B	B5B_N0	2.5 V ...fault)
 inaddr[1]	Input	PIN_AD30	5B	B5B_N0	2.5 V ...fault)
 inaddr[0]	Input	PIN_AC28	5B	B5B_N0	2.5 V ...fault)
 outaddr[3]	Input	PIN_V25	5B	B5B_N0	2.5 V ...fault)
 outaddr[2]	Input	PIN_W25	5B	B5B_N0	2.5 V ...fault)
 outaddr[1]	Input	PIN_AC30	5B	B5B_N0	2.5 V ...fault)
 outaddr[0]	Input	PIN_AB28	5B	B5B_N0	2.5 V ...fault)
 we	Input	PIN_AK4	3B	B3B_N0	2.5 V ...fault)

KEY[0]时钟信号 clk

LED[3:0]:dout2[3:0]

LED[9:6]:dout1[3:0]

SW[1:0]:din[1:0]

SW[9:6]:inaddr[3:0]

SW[5:2]:outaddr[3:0]

KEY[1]:we

## 五：实验过程中遇到的问题及解决


在写存储器时，用 txt 来初始化 ram 的时候没搞清楚格式，刚开始多加了一个空格导致编译不过：

```
@ 0 00  
@ 1 01  
@ 2 02  
@ 3 03  
@ 4 04  
@ 5 05  
@ 6 06  
@ 7 07  
@ 8 08  
@ 9 09  
@ a 0a  
@ b 0b  
@ c 0c  
@ d 0d  
@ e 0e  
@ f 0f
```

后来向同学询问后修改成了正确的格式：

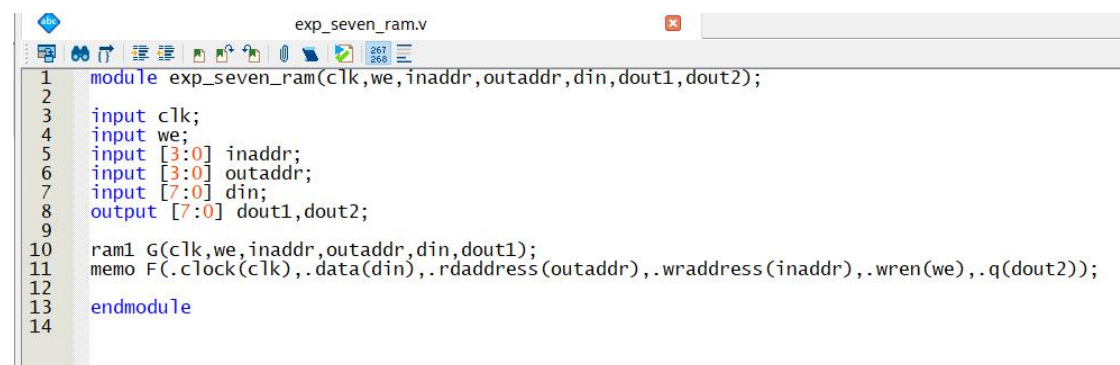
```
@0 00  
@1 01  
@2 02  
@3 03  
@4 04  
@5 05  
@6 06  
@7 07  
@8 08  
@9 09  
@a 0a  
@b 0b  
@c 0c  
@d 0d  
@e 0e  
@f 0f
```

另外在实验过程中遇到的一个问题是 IP 核生成的 RAM 无法显示初始化数据显示混乱的问题，后来经排查发现是主文件设计代码中给 IP 核生成的 RAM 传入的参数出现了谬误：



```
1 module exp_seven_ram(clk,we,inaddr,outaddr,din,dout1,dout2);
2
3 input clk;
4 input we;
5 input [3:0] inaddr;
6 input [3:0] outaddr;
7 input [7:0] din;
8 output [7:0] dout1,dout2;
9
10 ram1 G(clk,we,inaddr,outaddr,din,dout1);
11 memo F(.clock(clk),.data(din),.rdaddress(inaddr),.wraddress(outaddr),.wren(we),.q(dout2));
12
13 endmodule
14
```

起初传参传反了，把输入地址 inaddr 传给了 rdaddress，把 outaddr 传给了 wraddress，后来把 inaddr 传给 wraddress，把 outaddr 传给 rdaddress，更正后就可以正常显示初始化数据了。



```
1 module exp_seven_ram(clk,we,inaddr,outaddr,din,dout1,dout2);
2
3 input clk;
4 input we;
5 input [3:0] inaddr;
6 input [3:0] outaddr;
7 input [7:0] din;
8 output [7:0] dout1,dout2;
9
10 ram1 G(clk,we,inaddr,outaddr,din,dout1);
11 memo F(.clock(clk),.data(din),.rdaddress(outaddr),.wraddress(inaddr),.wren(we),.q(dout2));
12
13 endmodule
14
```

## 六、实验得到的启示

这次更深刻理解了 IP 核的使用，以及具体其与 verilog 语言代码之间的联系。

先规划好各个模块的大框架，再专门实现每一个小功能。自顶向下，逐步求精。并且在实现具体的小功能的时候考虑好使用的变量类型/宽度/个数，语句类型/函数/任务等等问题，用恰当的语句来实现适合的功能，构建数字电路建模的 first principle 思考体系。

## 七、意见与建议

实验手册中的实例对学习 verilog 语句使用与 quartus 相关操作有巨大帮助，希望能在学习新知识的同时接触更多相关例子来加深对新知的理解与应用能力。

希望自己能加强对网上知识的搜索自学能力，这样可以迅速掌握新的知识。