## **实验四 存储器实验**

### 实验目的

掌握存储器的结构和存储器的字位扩展方式。

### 实验原理

图 4.1给出了用256× 4位的存储器“芯片”构成1K× 8位的主存储器的原理框图。整个电路在FPGA内部实现，256× 4位的存储器“芯片”也是用FPGA内部存储资源模拟。由于实验板拨动开关数量有限，数据输入和地址输入共用一组开关，因此设计了一个地址锁存器，实验时先将开关输入的地址保存在地址锁存器中，然后再通过开关输入数据。



图 4.1 主存储器组织实验原理图

1. RAM模块设计

RAM模块容量为：256× 4位，模拟一般的静态RAM芯片的引脚，RAM模块的端口设计有地址输入（ADDR）、数据输入（DIN）、数据输出（DOUT）、写使能（WR）、读使能（RD）、片选（CS）。和普通SRAM芯片不同的是，FPGA内部的RAM模块设计为同步存储器，还有一个时钟输入（CLK）端口。设计代码见1。片选信号控制“芯片”是否工作，片选无效时，不能进行读写操作，DOUT输出为高阻态；片选有效时，允许写入和读出。

2.顶层模块设计

顶层模块按照图4.1原理图设计主存储器。共设计8个256× 4位RAM模块，并设计一个2-4译码器，连接成1K× 8位的主存储器。

### 实验内容

1. **基础实验**

**实验操作和记录**

1）计算地址分配

计算每个RAM块所占用的地址空间。将地址译码器输出的4个选择线所对应的地址范围以16进制形式填入下表。

|  |  |  |
| --- | --- | --- |
|  | 起始地址（H） | 结束地址（H） |
| Y0 | 000 | 0FF |
| Y1 | 100 | 1FF |
| Y2 | 200 | 2FF |
| Y3 | 300 | 3FF |

2）片选信号的产生

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | ADDR | CLK0 | EN | Y3 | Y2 | Y1 | Y0 |
| ① | **00**00000000 |  | 0 | 0 | 0 | 0 | 0 |
| ② | **00**00000000 |  | 1 | 0 | 0 | 0 | 1 |
| ③ | **01**00000000 |  | 1 | 0 | 0 | 1 | 0 |
| ④ | **10**00000000 |  | 1 | 0 | 1 | 0 | 0 |
| ⑤ | **11**00000000 |  | 1 | 1 | 0 | 0 | 0 |

实验现象分析：

（1）阅读程序清单 3.2，并根据第①行的实验结果，如果片选信号EN为0，2-4译码器的输出Y＝\_\_0\_\_\_\_，从而各个RAM块的片选信号CS \_\_\_\_\_无效\_\_\_\_（有效／无效），存储器\_\_\_不可以\_\_\_\_\_\_（可以／不可以）进行读写操作。

（2）根据表中②、③、④、⑤行的结果，当片选信号CS为1时，ADDR[\_9\_][\_8\_]决定了当前访问U0/U1、U2/U3、U4/U5、U6/U7中的哪个存储器模块。

3）存储器的写操作和读操作过程

256×4 RAM模块的RAM块的读写均由CLK同步，所以设置好地址、数据、读写使能后，要产生一个CLK脉冲才能将数据写入或读出RAM块，时钟信号通过按键产生。

连续往存储器的001H、102H、203H、304H单元，分别写入5AH，6BH，7CH，8DH，然后分别读出观察。

|  | ADDR/ DATA | CLK0 | DATAoe | CS | WR | RD | CLK | DB | Y3 | Y2 | Y1 | Y0 | 实验现象分析 |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| ① | 001H |  | 0 | 1 | **—** | **—** | **——** | FF | 0 | 0 | 0 | 1 | 5AH→DB→（001H） |
| 5AH | —— | 1 | 1 | 1 | 0 |  | 5A | 0 | 0 | 0 | 1 |
| ② | 102H |  | 0 | 1 | **—** | **—** | **——** | FF | 0 | 0 | 1 | 0 | 6BH→DB→(102H) |
| 6BH | —— | 1 | 1 | 1 | 0 |  | 6B | 0 | 0 | 1 | 0 |
| ③ | 203H |  | 0 | 1 | — | — | **——** | FF | 0 | 1 | 0 | 0 | 7CH→DB→(203H) |
| 7CH | —— | 1 | 1 | 1 | 0 |  | 7C | 0 | 1 | 0 | 0 |
| ④ | 304H |  | 0 | 1 | — | — | **——** | FF | 1 | 0 | 0 | 0 | 8DH→DB→(304H) |
| 8DH | —— | 1 | 1 | 1 | 0 |  | 8D | 1 | 0 | 0 | 0 |
| ⑤ | 001H |  | 0 | 1 | 0 | 1 |  | 5A | 0 | 0 | 0 | 1 | （001H）→DB |
| ⑥ | 102H |  | 0 | 1 | 0 | 1 |  | 6B | 0 | 0 | 1 | 0 | (102H)→DB |
| ⑦ | 203H |  | 0 | 1 | 0 | 1 |  | 7C | 0 | 1 | 0 | 0 | (203H)→DB |
| ⑧ | 304H |  | 0 | 1 | 0 | 1 |  | 8D | 1 | 0 | 0 | 0 | (304H)→DB |

000H地址访问的是U1、\_U0\_\_\_ 存储器块，101H地址访问的是\_\_U3\_\_、\_\_U2\_\_存储器块，202H地址访问的是\_\_U5\_\_、\_\_U4\_\_存储器，303H地址访问的是\_U7\_\_\_、\_\_U6\_\_存储器块。

#### **2. 提高实验（用Logisim实现存储器扩展（位扩展、字扩展））**。

1）**存储器扩展（位扩展）**

#### Step 1 设计

设计要求：现有8\*8 位的RAM存储器模块，请扩展为8\*32 位RAM存储空间；

（1）放置元件

根据设计要求，扩展8\*32 位RAM存储空间需要4个8\*8 位的RAM存储器模块，如图4.2所示，添加4个存储器元件。

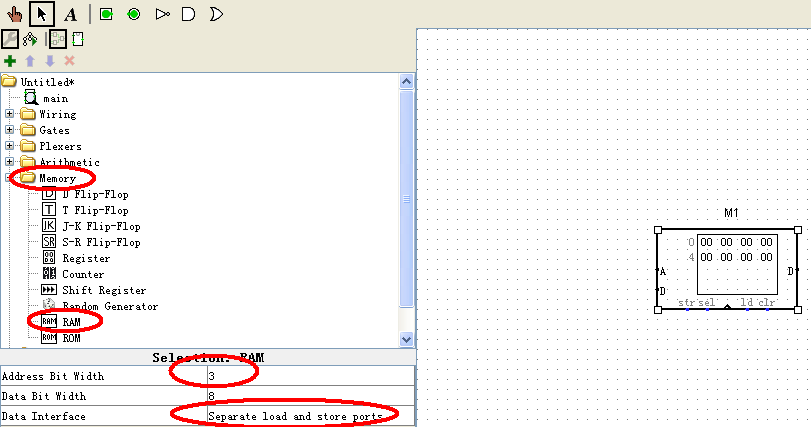


图4.2 放置存储器元件

添加输入、输出端口和时钟：

输入端口：

address：存储器地址；

Input：输入数据

WE：写使能

OE：存储器输出使能

输出端口：

output：存储器输出

时钟： clk

（2）连线

元件放置后，连线完成如图4.3所示。

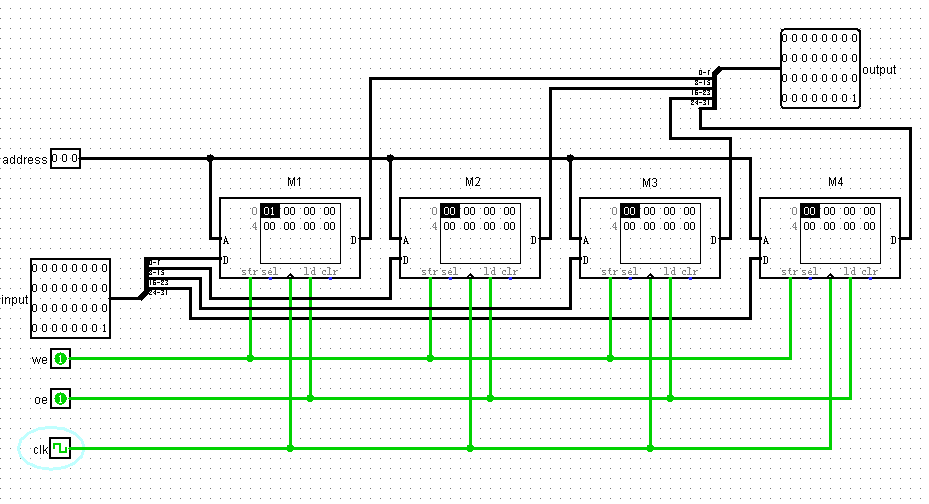


图4.3 位扩展逻辑图

说明：上图中，需要将32位的数据，分成4组8位数据分别连接到4个存储器模块上，因此放置了2个总线元件。关于总线放置以及配置如图4.4所示。

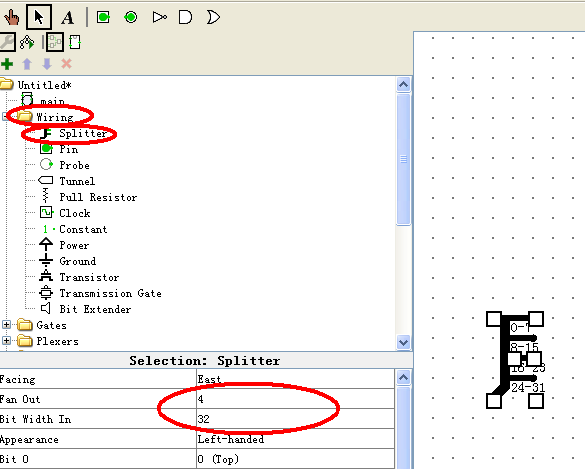
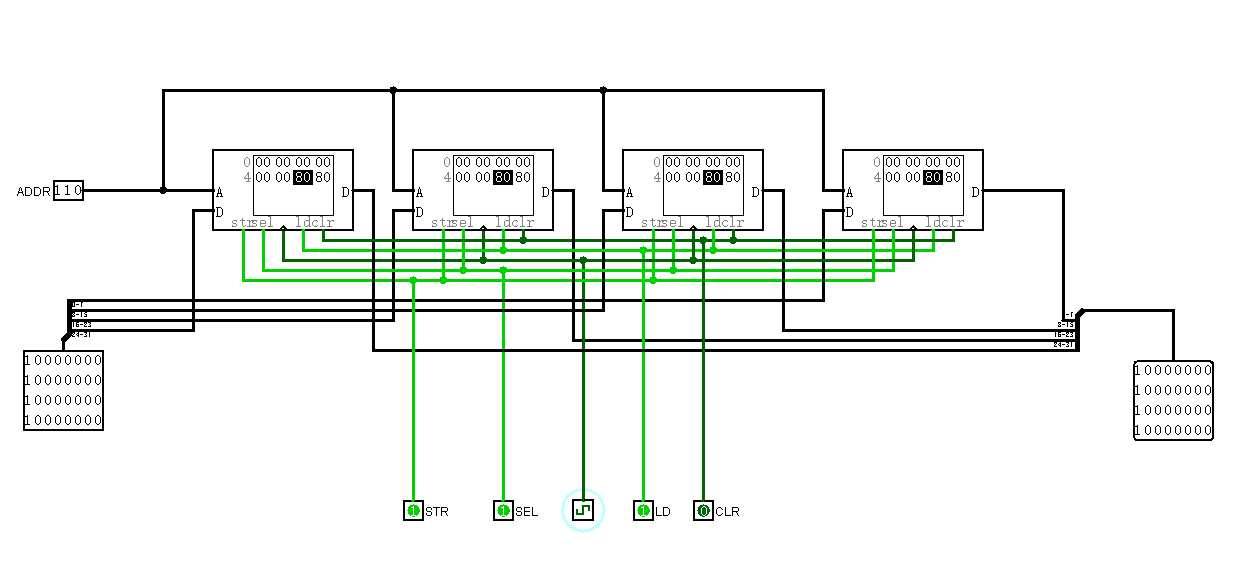


图4.4 数据总线设置

**电路图：**



#### Step 2 测试

设置WE为 1、OE为0， Address为”010”， Input为”00001000 00000100 00000010 00000001”。通过点击 CLK来查看结果，如图4.5所示。

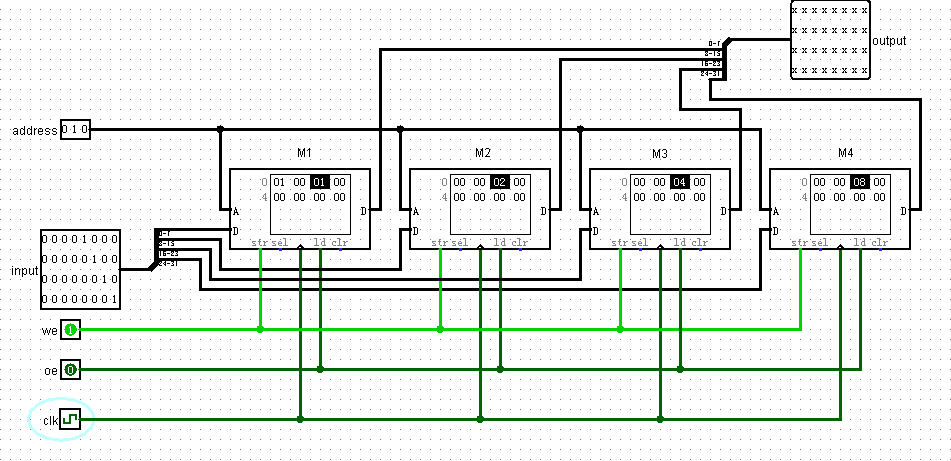
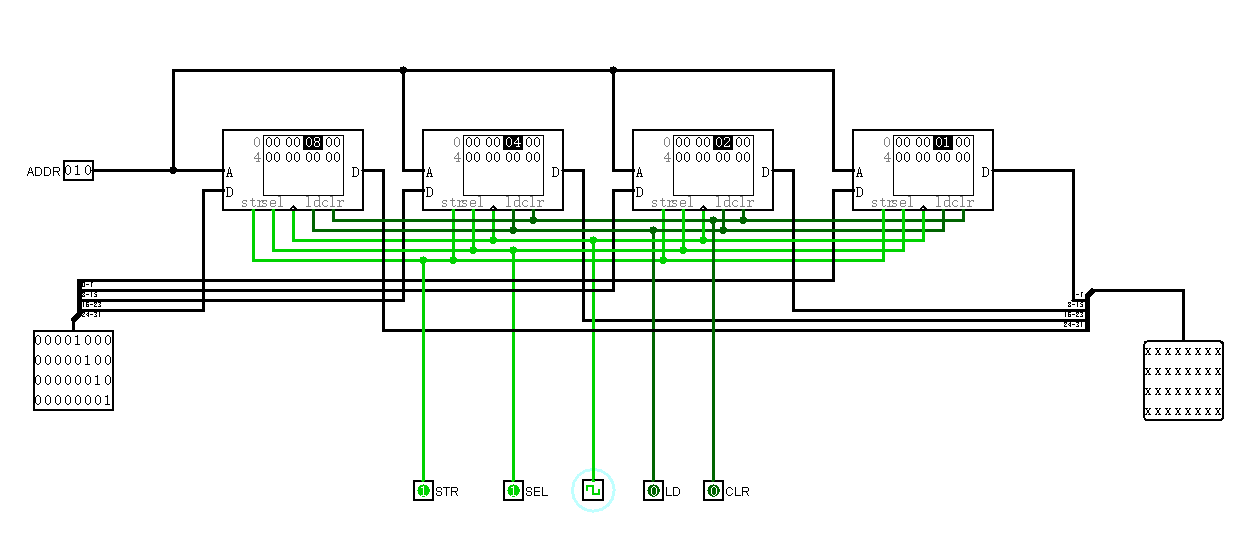


图4.5测试结果

**测试结果：**



**2）存储器扩展（字扩展）**

#### Step 1 设计

设计要求：现有 位的RAM存储器模块，请扩展为 位RAM存储空间；

根据设计要求，扩展32 位RAM存储空间需要4个 位的RAM存储器模块。

为实现对每个存储器模块的读写操作，需要利用地址的高位（A5-A4）以及2-4译码器对4个存储器模块进行片选，译码器的输出分别与oe,we信号相与，就可以控制每个存储器模块的读、写操作。如图4.6所示，是字扩展的逻辑图。

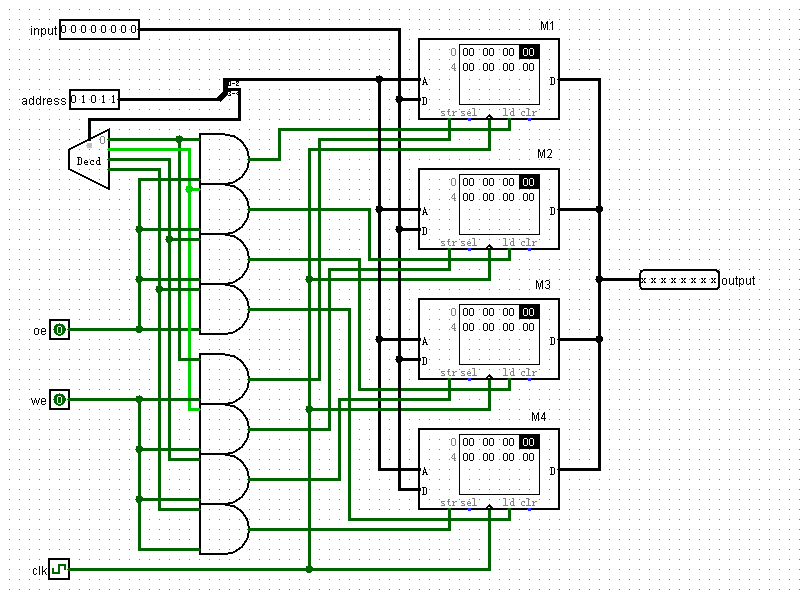
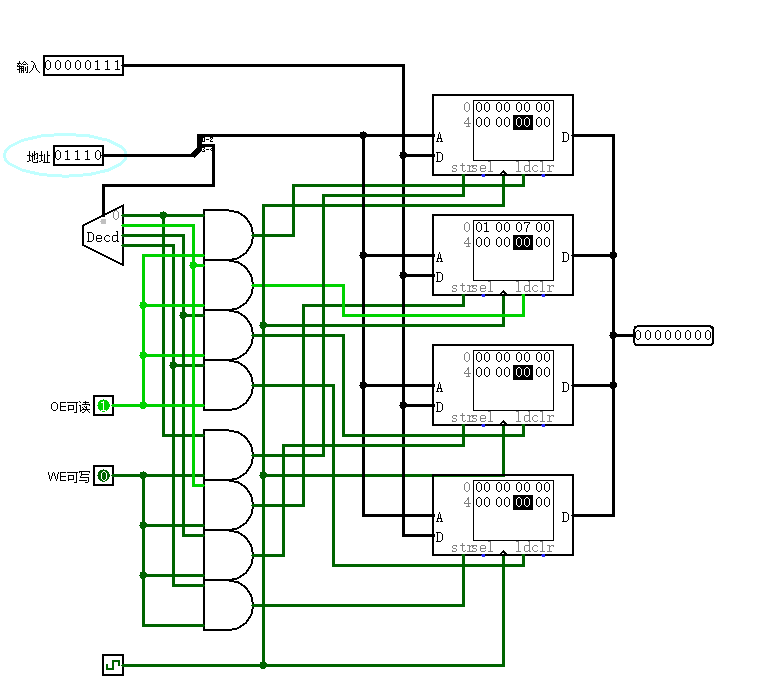


图4.6扩展连线

**电路图：**



#### Step 2 测试

设置WE为 1、OE为0时，Input为”00001111”，Address为”01011”，通过点击 CLK来查看结果如图4.7所示。

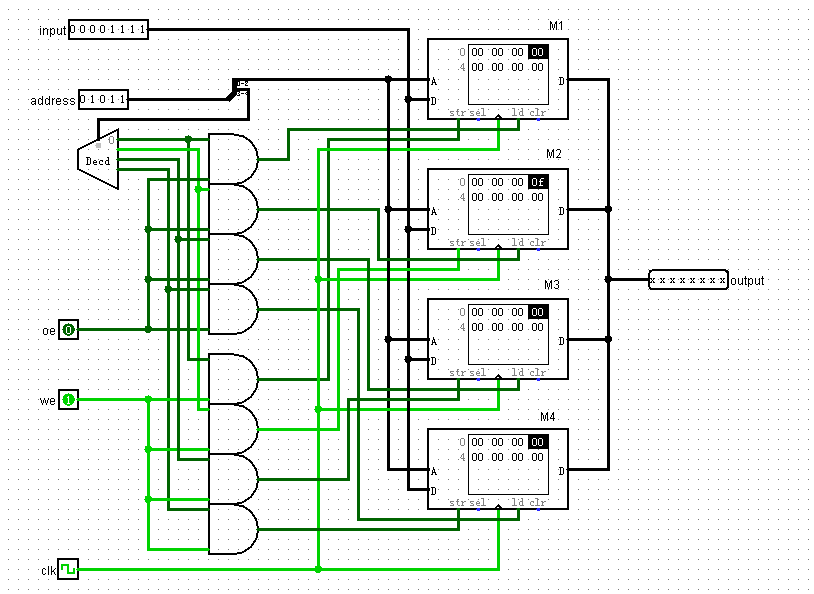
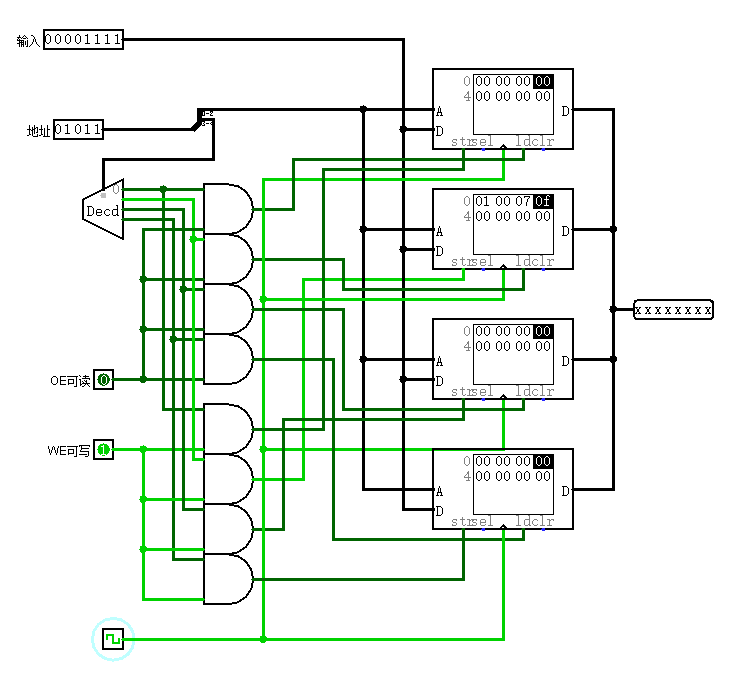


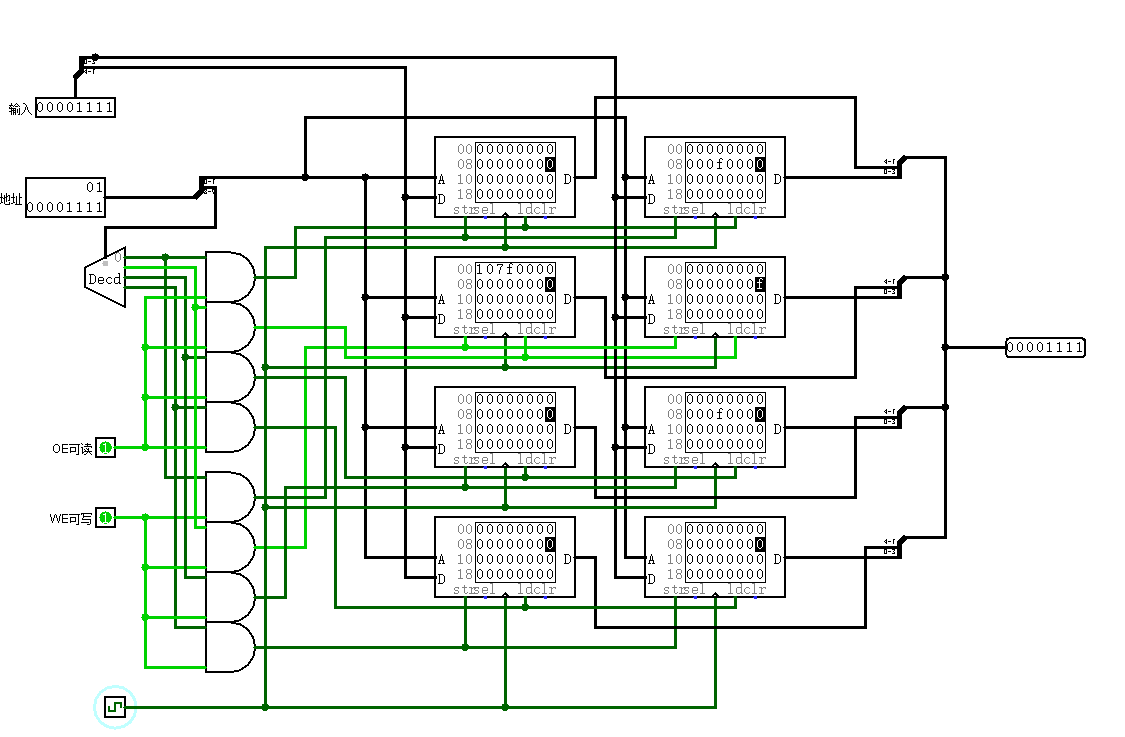
图4.7扩展测试

**测试结果：**



### 选做题

1. 利用logisim软件，把256×4 b的RAM，扩展为1K×8b的RAM。



1. 利用Verilog硬件描述语言实现一个图 4.1原理图所示的256× 4b的RAM模块，然后通过设计一个2-4译码器，把256× 4b的RAM模块扩展成1K× 8b的RAM，并利用Modelsim进行仿真测试。

源码：

**256× 4b的RAM模块：**

module RAM\_256(A,Din,Dout,clk,str,sel,ld,clr);

input [7:0] A;//地址

input [3:0] Din;

input clk,str,sel,ld,clr;

output reg[3:0] Dout;

integer i;

reg [3:0] data[255:0]; //memory型，256个存储单元的存储器data，每个存储单元是一个4 //位寄存器

initial begin

Dout = 4'bz;

end

always@(posedge clk or posedge clr)

begin

if(sel == 0) begin Dout = 4'bxxxx; end

else

begin

if(clr == 1)

begin

for(i = 0; i < 255; i = i+1)

data[i] = 0;

end

else if(str == 1)

begin

data[A] = Din;

end

else if(ld == 1)

begin

Dout = data[A];

end

else begin Dout = 4'bz; end

end

end

endmodule

**2\_4译码器：**

module yima2\_4(in,out);

input [1:0]in;

output reg[3:0] out = 1;

always@(in)

begin

case(in)

2'b00: out = 4'b0001;

2'b01: out = 4'b0010;

2'b10: out = 4'b0100;

2'b11: out = 4'b1000;

endcase

end

Endmodule

**顶层文件，1K× 8b的RAM：**

module RAM\_1k(A,Din,Dout,clk,str,ld,sel,clr,out);

input [9:0] A;

input [7:0] Din;

input clk,str,ld,sel,clr;

output [7:0]Dout;

output [3:0]out;

yima2\_4 yima1(.in(A[9:8]),.out(out));

RAM\_256 RAM0(.A(A[7:0]),.Din(Din[7:4]),.clk(clk),.str(str &(out == 4'b0001)),.ld(ld &(out == 4'b0001)),.sel(sel),.clr(clr),.Dout(Dout[7:4]));

RAM\_256 RAM1(.A(A[7:0]),.Din(Din[3:0]),.clk(clk),.str(str &(out == 4'b0001)),.ld(ld &(out == 4'b0001)),.sel(sel),.clr(clr),.Dout(Dout[3:0]));

RAM\_256 RAM2(.A(A[7:0]),.Din(Din[7:4]),.clk(clk),.str(str &(out == 4'b0010)),.ld(ld &(out == 4'b0010)),.sel(sel),.clr(clr),.Dout(Dout[7:4]));

RAM\_256 RAM3(.A(A[7:0]),.Din(Din[3:0]),.clk(clk),.str(str &(out == 4'b0010)),.ld(ld &(out == 4'b0010)),.sel(sel),.clr(clr),.Dout(Dout[3:0]));

RAM\_256 RAM4(.A(A[7:0]),.Din(Din[7:4]),.clk(clk),.str(str &(out == 4'b0100)),.ld(ld &(out == 4'b0100)),.sel(sel),.clr(clr),.Dout(Dout[7:4]));

RAM\_256 RAM5(.A(A[7:0]),.Din(Din[3:0]),.clk(clk),.str(str &(out == 4'b0100)),.ld(ld &(out == 4'b0100)),.sel(sel),.clr(clr),.Dout(Dout[3:0]));

RAM\_256 RAM6(.A(A[7:0]),.Din(Din[7:4]),.clk(clk),.str(str &(out == 4'b1000)),.ld(ld &(out == 4'b1000)),.sel(sel),.clr(clr),.Dout(Dout[7:4]));

RAM\_256 RAM7(.A(A[7:0]),.Din(Din[3:0]),.clk(clk),.str(str &(out == 4'b1000)),.ld(ld &(out == 4'b1000)),.sel(sel),.clr(clr),.Dout(Dout[3:0]));

endmodule

测试截图：

