




2020-10-18

## 数字电路实验七 寄存器

秦嘉余

191220088

1348288404@qq.com



# 目录

- 1.实验目的 .....1
- 2.实验原理 .....1
- 3.实验环境/器材 .....2
- 4. 存储器实例分析 .....2
  - 1.实验思路 .....2
  - 2. 实验代码.....2
  - 3.RTL 视图 .....4
  - 4.仿真结果 .....4
  - 5.实验结果 .....5
- 5. 两个 16\*8 存储器.....5
  - 1.设计思路 .....5
  - 2.实验代码 .....5
  - 3.RTL 视图 .....8
  - 4.引脚分配图 .....8
  - 5.实验结果 .....8
- 6.思考题 .....9
- 7.实验实验中遇到的问题及解决办法.....9

---

## 1.实验目的

---

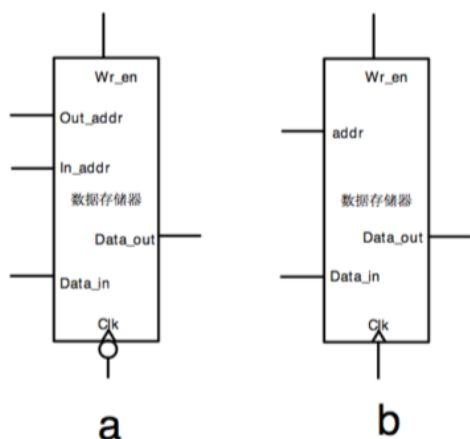
本实验的目的是了解 FPGA 的片上存储器的特性，分析存储器的工作时序 和结构，并学习如何设计存储器。

---

## 2.实验原理

---

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，如图 7 1 所 示。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。



**写数据：**在时钟 (clk) 有效沿 (上升或下降沿)，如果写使能 (Wr\_en，也可以没有使能端) 有效，则读取输入总线 (Data\_in) 上的数据，将其存储到 输入地址线 (In\_addr) 所指的存储单元中。

**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。

---

### 3.实验环境/器材

---

本次实验的环境为 Quartus17.1 版本  
本次实验的器材为 DE10 Standard 开发板

---

### 4. 存储器实例分析

---

#### 1.实验思路

本次实验的目的是验证存储器的读取功能的时序关系，关键在于对于 vt 测试文件的编写，要合适的编写测试文件来实验三种输出的不同之处。在本实验中，先验证每个存储器的值，之后修改某个值后再查看这个值，以此来观察不同的输出的时序差别

#### 2. 实验代码

摘抄自文档，故不作赘述

```

module v_ams_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] =
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

```

vt 测试文件代码如下:

```

        .outaddr(outaddr),
        .we(we)
    );
initial
begin
    clk=0; we=0;
        outaddr=0; #18
        outaddr=1; #18
        outaddr=2; #18
        outaddr=3; #18
        outaddr=4; #18
        outaddr=5; #18
        outaddr=6; #18
        outaddr=7; #18

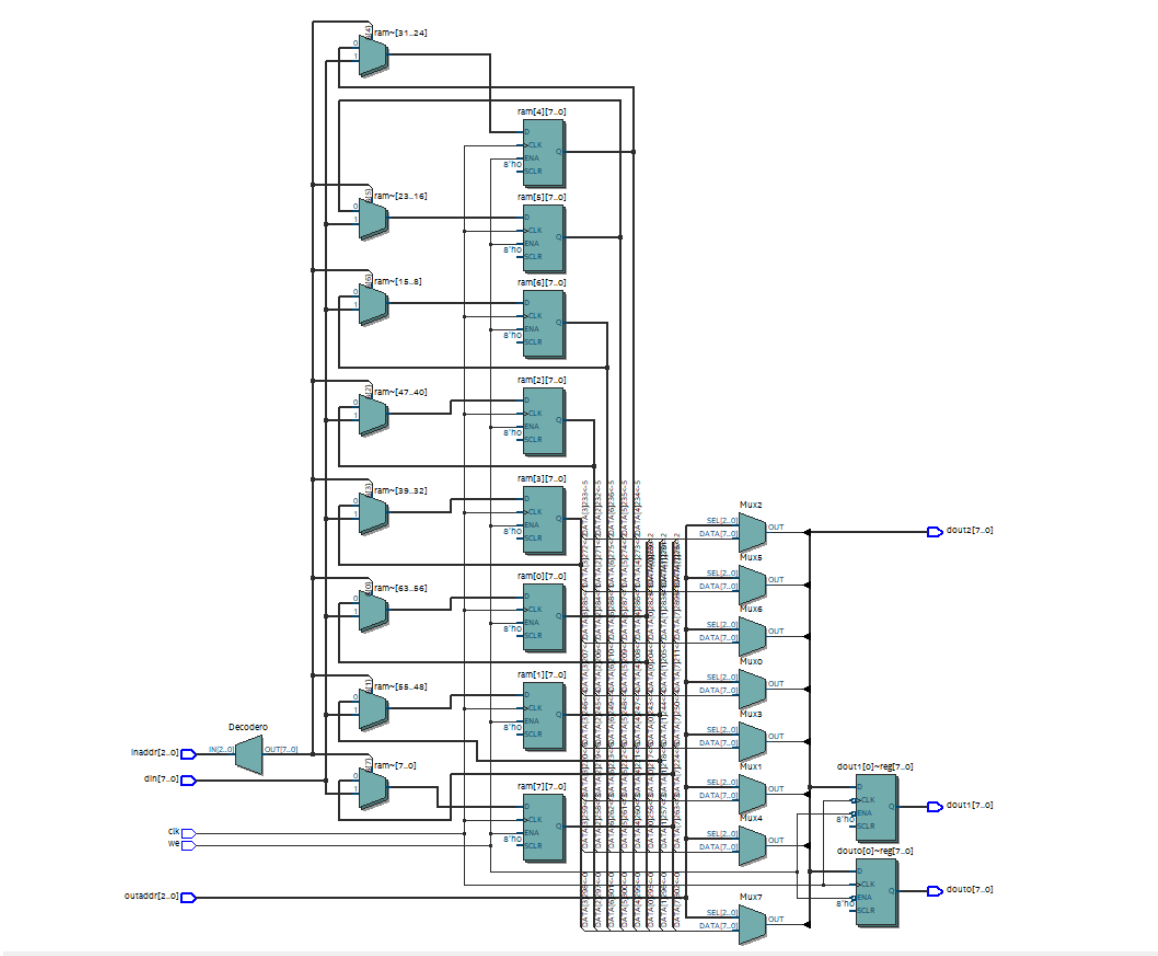
        we=1;
        inaddr=2; din=8'hff; #18
        outaddr=2; #8
        we=0; #18
        we=1;
        inaddr=5; din=8'h0b; #18
        outaddr=5; #8
        we=0; #18
    $stop;
end

always
begin
    #5 clk=~clk;
end

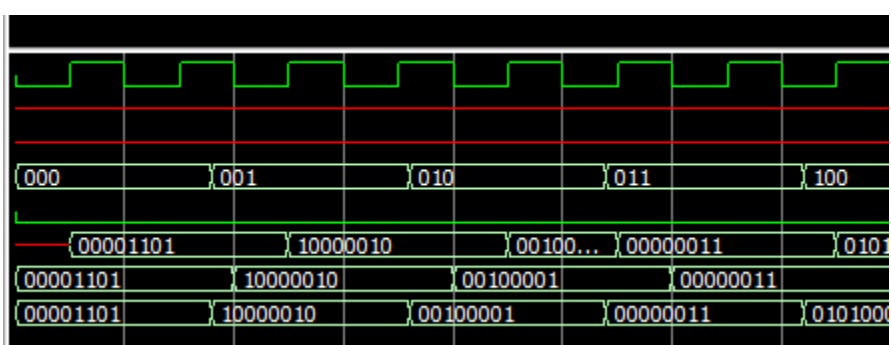
endmodule

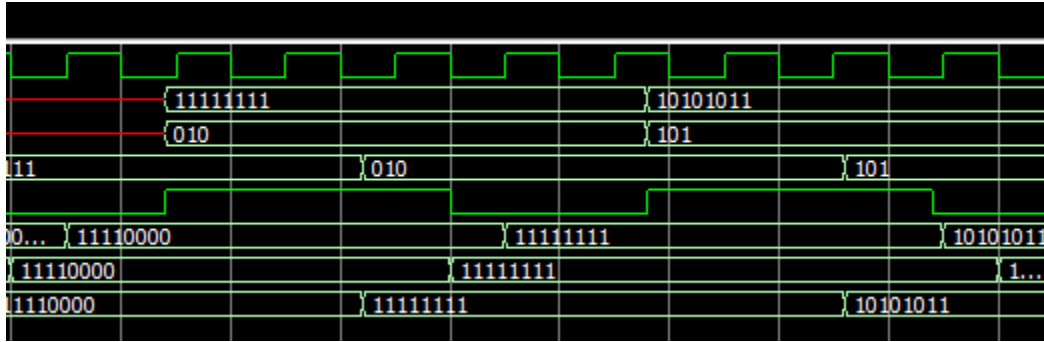
```

3.RTL 视图



4.仿真结果





## 5.实验结果

dout0 和 dout1 都是在 we=0 是才会赋值，当时钟信号上升沿到来时，dout1 输出，下降沿到达时，dout2 输出，而 dout2 与时钟无关，与 outaddr 保持同步输出

### 5. 两个 16\*8 存储器

## 1.设计思路

本次实验设计两个 16\*8 存储器，而第一个存储器需要自己手动创建，第二个存储器利用 IP 核由系统生成，最后在开发板上验证

## 2.实验代码

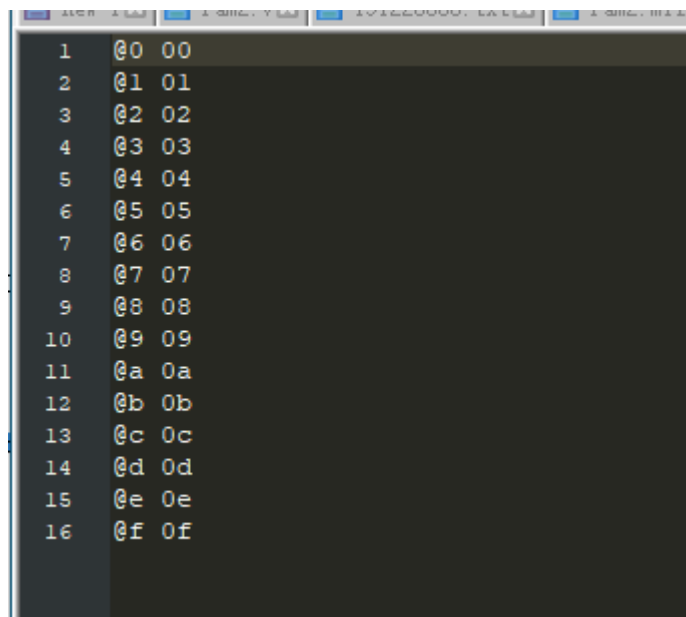
主模块如下，其中 clk 为共用的时钟信号，din 为共用的输入，由于开发板开关不足，所以设置成两位，高位清零，inaddr 和 outaddr 分别为输入输出的地址，we 为写入的使能端，HEX0-HEX3 为输出的显示，其中 HEX0,HEX1 位 ram1 的 16 进制显示，HEX2,HEX3 位 ram2 的 16 进制显示

```
module RAM(clk,din,inaddr,outaddr,we,HEX0,HEX1,HEX2,HEX3);
    input clk;
    input [1:0] din;
    input [3:0] inaddr,outaddr;
    input we;
    output [6:0] HEX0,HEX1,HEX2,HEX3;
    wire [7:0] dout1,dout2;
    ram1 ram_1(clk,din,inaddr,outaddr,we,dout1);
    ram2 ram_2(clk,din,outaddr,inaddr,we,dout2);

    decode_hex decode_hex0(dout1[3:0],HEX0);
    decode_hex decode_hex1(dout1[7:4],HEX1);
    decode_hex decode_hex2(dout2[3:0],HEX2);
    decode_hex decode_hex3(dout2[7:4],HEX3);

endmodule
```

ram1 的代码如下，寄存器内的内容为 mem1.txt 内的内容，输出也为时钟有效



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

```
module ram1(clk,din,inaddr,outaddr,we,dout);
    input clk;
    input [7:0] din;
    input [3:0] inaddr,outaddr;
    input we;
    output reg [7:0] dout;
    reg [7:0] ram [15:0];

    initial begin
        $readmemh("D:/data/Homework/FPGA/Exp/Exp7/mem1.txt",ram,0,15);
    end

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

ram2 的代码如下，由系统自动生成：

```

input    clock;
input [7:0] data;
input [3:0] rdaddress;
input [3:0] wraddress;
input    wren;
output [7:0] q;
`ifndef ALTERA_RESERVED_QIS
// synopsys translate_off
endif
    tri1    clock;
    tri0    wren;
`ifndef ALTERA_RESERVED_QIS
// synopsys translate_on
endif

wire [7:0] sub_wire0;
wire [7:0] q = sub_wire0[7:0];

altsyncram altsyncram_component (
    .address_a (wraddress),
    .address_b (rdaddress),
    .clock0 (clock),
    .data_a (data),
    .wren_a (wren),
    .q_b (sub_wire0),
    .aclr0 (1'b0),
    .aclr1 (1'b0),
    .addressstall_a (1'b0),
    .addressstall_b (1'b0),
    .byteena_a (1'b1),
    .byteena_b (1'b1),
    .clock1 (1'b1),
    .clocken0 (1'b1),
    .clocken1 (1'b1),
    .clocken2 (1'b1),
    .clocken3 (1'b1),
    .data_b ({8{1'b1}}),
    .eccstatus (),

```

decode\_hex 代码如下，功能为把 2 进制输出转化为 16 进制在七段译码管显示

```

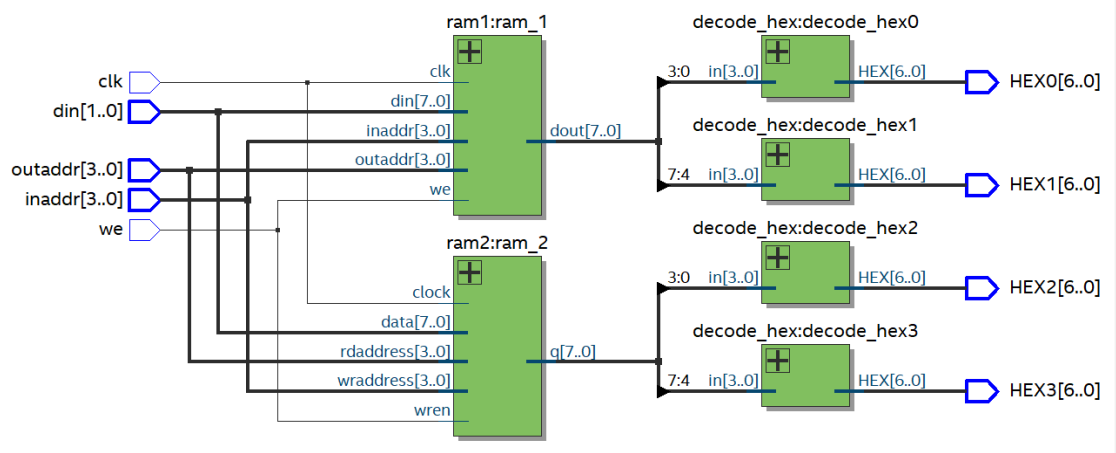
module decode_hex(in,HEX);
    input [3:0] in;
    output reg [6:0] HEX;

    always @(in)
    begin
        case (in)
            0: HEX=7'b1000000;
            1: HEX=7'b11111001;
            2: HEX=7'b0100100;
            3: HEX=7'b0110000;
            4: HEX=7'b0011001;
            5: HEX=7'b0010010;
            6: HEX=7'b0000010;
            7: HEX=7'b11111000;
            8: HEX=7'b0000000;
            9: HEX=7'b0010000;
            10: HEX=7'b0001000;
            11: HEX=7'b0000011;
            12: HEX=7'b1000110;
            13: HEX=7'b0100001;
            14: HEX=7'b0000110;
            15: HEX=7'b0001110;
            default: HEX=7'b1111111;
        endcase
    end
endmodule

```



### 3.RTL 视图



### 4.引脚分配图

out	HEX0[6]	Output	PIN_AH18	out	HEX3[6]	Output	PIN_AD20
out	HEX0[5]	Output	PIN_AG18	out	HEX3[5]	Output	PIN_AA19
out	HEX0[4]	Output	PIN_AH17	out	HEX3[4]	Output	PIN_AC20
out	HEX0[3]	Output	PIN_AG16	out	HEX3[3]	Output	PIN_AA20
out	HEX0[2]	Output	PIN_AG17	out	HEX3[2]	Output	PIN_AD19
out	HEX0[1]	Output	PIN_V18	out	HEX3[1]	Output	PIN_W19
out	HEX0[0]	Output	PIN_W17	out	HEX3[0]	Output	PIN_Y19
out	HEX1[6]	Output	PIN_V17	in	clk	Input	PIN_AJ4
out	HEX1[5]	Output	PIN_AE17	in	din[1]	Input	PIN_Y27
out	HEX1[4]	Output	PIN_AE18	in	din[0]	Input	PIN_AB30
out	HEX1[3]	Output	PIN_AD17	in	inaddr[3]	Input	PIN_V25
out	HEX1[2]	Output	PIN_AE16	in	inaddr[2]	Input	PIN_W25
out	HEX1[1]	Output	PIN_V16	in	inaddr[1]	Input	PIN_AC30
out	HEX1[0]	Output	PIN_AF16	in	inaddr[0]	Input	PIN_AB28
out	HEX2[6]	Output	PIN_W16	in	outaddr[3]	Input	PIN_AA30
out	HEX2[5]	Output	PIN_AF18	in	outaddr[2]	Input	PIN_AC29
out	HEX2[4]	Output	PIN_Y18	in	outaddr[1]	Input	PIN_AD30
out	HEX2[3]	Output	PIN_Y17	in	outaddr[0]	Input	PIN_AC28
out	HEX2[2]	Output	PIN_AA18	in	we	Input	PIN_AK4
out	HEX2[1]	Output	PIN_AB17				
out	HEX2[0]	Output	PIN_AA21				

### 5.实验结果

已在开发板上验收

---

## 6.思考题

---

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

assign dout = ram[outaddr];
```

### 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?

原来的代码输出与时钟信号无关，与 outaddr 保持同步，现在限制在 always 语句中，时钟信号有效才会改变

---

## 7.实验实验中遇到的问题及解决办法

---

在实验中发现，IP 核生成的 RAM2 中，每次对 outaddr 修改后，读取其他寄存器的值，或者修改当前寄存器的值要重新输出时，RAM2 总是在两个时钟周期后才会改变为实际的值。后来发现这一现象的产生与 IP 核系统生成的内部的时序逻辑有关，系统生成的逻辑中含有锁存器，每次时钟信号到来时，总是先锁存当前的值在寄存器中，在下一个时钟周期才输出。从这个现象中，更加能体会到：

“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

反复检查和验证。”

这句话的意义了