组成原理实验课程第_三_次实报告

实验名称	寄存器堆实现			班级	张金老师
学生姓名	杨冰雪	学号	2110508	指导老师	董前琨
实验地点	实验楼 A306		实验时间	2023.4.17	

1、 实验目的

1. 熟悉并掌握 MIPS 计算机中寄存器堆的原理和设计方法。
2. 初步了解 MIPS 指令结构和源操作数/目的操作数的概念。
3. 熟悉并运用 verilog 语言进行电路设计。
4. 为后续设计 cpu 的实验打下基础。

2、 实验内容说明

1） 掌握寄存器堆的工作原理，确定寄存器堆的输入输出端口设计，画好寄存器堆的设计框图或实验原理图。

2） 了解 FPGA 板，确定设计中与 FPGA 板上交互的接口。

3） 编写 verilog 代码，要求使用 4 位 wen 控制信号，对应写入 wdata 的四个字节；要求使用 2 位 ren 控制信号，控制读出数据的高 16 位和低 16 位。

4） 完成调用寄存器堆模块的外围模块的设计，并编写代码，对代码进行综合布局布线下载到实验箱里 FPGA 板上，进行上板验证。

3、 实验原理图

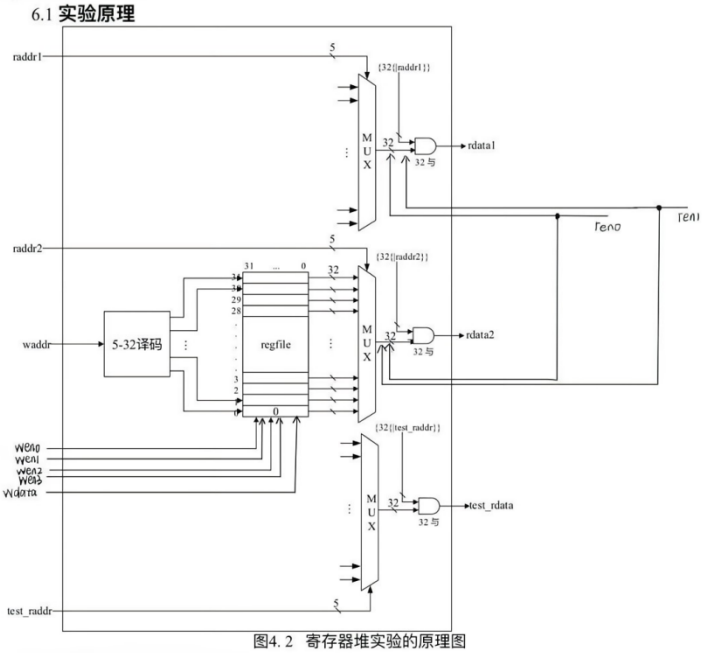


图4. 2 寄存器堆实验的原理图

该寄存器堆共有一个写端口和三个读端口，其中一个读端口为调试读端口，用于扫描读出 32 个寄存器。写端口有四个使能信号分别为 wen0、wen1、wen2、wen3，分别写入 wdata 的[7:0]、[15:8]、[23:16]、[31:24]位。两个读端口也两个控制信号 ren0、ren1，同时控制两个读端口，分比读出 rdata 的[15:0]、[31:16]位。

4、 实验步骤

（分布介绍依次完成了哪些代码修改，从而实现了什么样的功能）

1. 在 regfile 模块中定义了 4 个 wen 控制信号（wen0，wen1，wen2，wen3）和 2 个 ren 控制信号（ren0，ren1）。

```
module regfile(
    input           clk,
    input           wen0,
    input           wen1,
    input           wen2,
    input           wen3,
    input           ren0,
    input           ren1,
    input     [4 :0] raddr1,
```

2. 当 wen0 输入为高信号时，把写入数据 wdata 的第一个字节写入寄存器中，当 wen1 输入为高信号时，把写入数据 wdata 的第二个字节写入寄存器中，当 wen2 输入为高信号时，把写入数据 wdata 的第三个字节写入寄存器中，当 wen3 输入为高信号时，把写入数据 wdata 的第四个字节写入寄存器中。

```
if (wen0)
begin
    rf[waddr][7:0] <= wdata[7:0];
end
if (wen1)
begin
    rf[waddr][15:8] <= wdata[15:8];
end
if (wen2)
begin
    rf[waddr][23:16] <= wdata[23:16];
end
if (wen3)
begin
    rf[waddr][31:24] <= wdata[31:24];
end
```

3. 当 ren0 为高信号时，第一个读端口读出 rdata1 的低 16 位；当 ren1 为高信号时，第二个读端口读出 rdata1 的高 16 位。

```
always @(*)
begin
    if(ren0)
    begin
    case (raddr1)
        5'd1 : rdata1[15:0] <= rf[1 ][15:0];
        5'd2 : rdata1[15:0] <= rf[2 ][15:0];
        5'd3 : rdata1[15:0] <= rf[3 ][15:0];
        5'd4 : rdata1[15:0] <= rf[4 ][15:0];
        5'd5 : rdata1[15:0] <= rf[5 ][15:0];
        5'd6 : rdata1[15:0] <= rf[6 ][15:0];
        5'd7 : rdata1[15:0] <= rf[7 ][15:0];
        5'd8 : rdata1[15:0] <= rf[8 ][15:0];
        5'd9 : rdata1[15:0] <= rf[9 ][15:0];
        5'd10: rdata1[15:0] <= rf[10][15:0];
        5'd11: rdata1[15:0] <= rf[11][15:0];
        5'd12: rdata1[15:0] <= rf[12][15:0];
        5'd13: rdata1[15:0] <= rf[13][15:0];
        5'd14: rdata1[15:0] <= rf[14][15:0];
        5'd15: rdata1[15:0] <= rf[15][15:0];
        5'd16: rdata1[15:0] <= rf[16][15:0];
        5'd17: rdata1[15:0] <= rf[17][15:0];
        5'd18: rdata1[15:0] <= rf[18][15:0];
        5'd19: rdata1[15:0] <= rf[19][15:0];
        5'd20: rdata1[15:0] <= rf[20][15:0];
        5'd21: rdata1[15:0] <= rf[21][15:0];
        5'd22: rdata1[15:0] <= rf[22][15:0];
        5'd23: rdata1[15:0] <= rf[23][15:0];
        5'd24: rdata1[15:0] <= rf[24][15:0];
        5'd25: rdata1[15:0] <= rf[25][15:0];
        5'd26: rdata1[15:0] <= rf[26][15:0];
        5'd27: rdata1[15:0] <= rf[27][15:0];
        5'd28: rdata1[15:0] <= rf[28][15:0];
        5'd29: rdata1[15:0] <= rf[29][15:0];
        5'd30: rdata1[15:0] <= rf[30][15:0];
        5'd31: rdata1[15:0] <= rf[31][15:0];
        default : rdata1 <= 32'd0;
    endcase
    end
```

```
if(ren1)
begin
case (raddr1)
    5'd1 : rdata1[31:16] <= rf[1 ][31:16];
    5'd2 : rdata1[31:16] <= rf[2 ][31:16];
    5'd3 : rdata1[31:16] <= rf[3 ][31:16];
    5'd4 : rdata1[31:16] <= rf[4 ][31:16];
    5'd5 : rdata1[31:16] <= rf[5 ][31:16];
    5'd6 : rdata1[31:16] <= rf[6 ][31:16];
    5'd7 : rdata1[31:16] <= rf[7 ][31:16];
    5'd8 : rdata1[31:16] <= rf[8 ][31:16];
    5'd9 : rdata1[31:16] <= rf[9 ][31:16];
    5'd10: rdata1[31:16] <= rf[10][31:16];
    5'd11: rdata1[31:16] <= rf[11][31:16];
    5'd12: rdata1[31:16] <= rf[12][31:16];
    5'd13: rdata1[31:16] <= rf[13][31:16];
    5'd14: rdata1[31:16] <= rf[14][31:16];
    5'd15: rdata1[31:16] <= rf[15][31:16];
    5'd16: rdata1[31:16] <= rf[16][31:16];
    5'd17: rdata1[31:16] <= rf[17][31:16];
    5'd18: rdata1[31:16] <= rf[18][31:16];
    5'd19: rdata1[31:16] <= rf[19][31:16];
    5'd20: rdata1[31:16] <= rf[20][31:16];
    5'd21: rdata1[31:16] <= rf[21][31:16];
    5'd22: rdata1[31:16] <= rf[22][31:16];
    5'd23: rdata1[31:16] <= rf[23][31:16];
    5'd24: rdata1[31:16] <= rf[24][31:16];
    5'd25: rdata1[31:16] <= rf[25][31:16];
    5'd26: rdata1[31:16] <= rf[26][31:16];
    5'd27: rdata1[31:16] <= rf[27][31:16];
    5'd28: rdata1[31:16] <= rf[28][31:16];
    5'd29: rdata1[31:16] <= rf[29][31:16];
    5'd30: rdata1[31:16] <= rf[30][31:16];
    5'd31: rdata1[31:16] <= rf[31][31:16];
    default : rdata1 <= 32'd0;
endcase
end
```

4. 当 ren0 为高信号时，第一个读端口读出 rdata2 的低十六位；当 ren1 为高信号

时，第二个读端口读出 rdata2 的高十六位。

```
always @(*)
begin
    if(ren0)
    begin
    case (raddr2)
        5'd1 : rdata2[15:0] <= rf[1 ][15:0];
        5'd2 : rdata2[15:0] <= rf[2 ][15:0];
        5'd3 : rdata2[15:0] <= rf[3 ][15:0];
        5'd4 : rdata2[15:0] <= rf[4 ][15:0];
        5'd5 : rdata2[15:0] <= rf[5 ][15:0];
        5'd6 : rdata2[15:0] <= rf[6 ][15:0];
        5'd7 : rdata2[15:0] <= rf[7 ][15:0];
        5'd8 : rdata2[15:0] <= rf[8 ][15:0];
        5'd9 : rdata2[15:0] <= rf[9 ][15:0];
        5'd10: rdata2[15:0] <= rf[10][15:0];
        5'd10: rdata2[15:0] <= rf[10][15:0];
        5'd11: rdata2[15:0] <= rf[11][15:0];
        5'd12: rdata2[15:0] <= rf[12][15:0];
        5'd13: rdata2[15:0] <= rf[13][15:0];
        5'd14: rdata2[15:0] <= rf[14][15:0];
        5'd15: rdata2[15:0] <= rf[15][15:0];
        5'd16: rdata2[15:0] <= rf[16][15:0];
        5'd17: rdata2[15:0] <= rf[17][15:0];
        5'd18: rdata2[15:0] <= rf[18][15:0];
        5'd19: rdata2[15:0] <= rf[19][15:0];
        5'd20: rdata2[15:0] <= rf[20][15:0];
        5'd21: rdata2[15:0] <= rf[21][15:0];
        5'd22: rdata2[15:0] <= rf[22][15:0];
        5'd23: rdata2[15:0] <= rf[23][15:0];
        5'd24: rdata2[15:0] <= rf[24][15:0];
        5'd25: rdata2[15:0] <= rf[25][15:0];
        5'd26: rdata2[15:0] <= rf[26][15:0];
        5'd27: rdata2[15:0] <= rf[27][15:0];
        5'd28: rdata2[15:0] <= rf[28][15:0];
        5'd29: rdata2[15:0] <= rf[29][15:0];
        5'd30: rdata2[15:0] <= rf[30][15:0];
        5'd31: rdata2[15:0] <= rf[31][15:0];
        default : rdata2 <= 32'd0;
    endcase
    end
```

```
if(ren1)
begin
case (raddr2)
    5'd1 : rdata2[31:16] <= rf[1 ][31:16];
    5'd2 : rdata2[31:16] <= rf[2 ][31:16];
    5'd3 : rdata2[31:16] <= rf[3 ][31:16];
    5'd4 : rdata2[31:16] <= rf[4 ][31:16];
    5'd5 : rdata2[31:16] <= rf[5 ][31:16];
    5'd6 : rdata2[31:16] <= rf[6 ][31:16];
    5'd7 : rdata2[31:16] <= rf[7 ][31:16];
    5'd8 : rdata2[31:16] <= rf[8 ][31:16];
    5'd9 : rdata2[31:16] <= rf[9 ][31:16];
    5'd10: rdata2[31:16] <= rf[10][31:16];
    5'd11: rdata2[31:16] <= rf[11][31:16];
    5'd12: rdata2[31:16] <= rf[12][31:16];
    5'd13: rdata2[31:16] <= rf[13][31:16];
    5'd14: rdata2[31:16] <= rf[14][31:16];
    5'd15: rdata2[31:16] <= rf[15][31:16];
    5'd16: rdata2[31:16] <= rf[16][31:16];
    5'd17: rdata2[31:16] <= rf[17][31:16];
    5'd18: rdata2[31:16] <= rf[18][31:16];
    5'd19: rdata2[31:16] <= rf[19][31:16];
    5'd20: rdata2[31:16] <= rf[20][31:16];
    5'd21: rdata2[31:16] <= rf[21][31:16];
    5'd22: rdata2[31:16] <= rf[22][31:16];
    5'd23: rdata2[31:16] <= rf[23][31:16];
    5'd24: rdata2[31:16] <= rf[24][31:16];
    5'd25: rdata2[31:16] <= rf[25][31:16];
    5'd26: rdata2[31:16] <= rf[26][31:16];
    5'd27: rdata2[31:16] <= rf[27][31:16];
    5'd28: rdata2[31:16] <= rf[28][31:16];
    5'd29: rdata2[31:16] <= rf[29][31:16];
    5'd30: rdata2[31:16] <= rf[30][31:16];
    5'd31: rdata2[31:16] <= rf[31][31:16];
    default : rdata2 <= 32'd0;
endcase
end
```

5. regfile_display 顶层文件中也需要定义 4 个 wen 控制信号和 2 个 ren 控制信号，并且需要在寄存器堆模块中把信号传进去。

```
input wen0,
input wen1,
input wen2,
input wen3,
input ren0,
input ren1,
input [1:0] input_sel,
```

```
regfile rf_module(
    .clk   (clk   ),
    .wen0  (wen0  ),
    .wen1  (wen1  ),
    .wen2  (wen2  ),
    .wen3  (wen3  ),
    .ren0  (ren0  ),
    .ren1  (ren1  ),
```

6. 约束文件中需要把传入的变量与拨码开关连接起来，从而能使信号能够拨码开关输入。

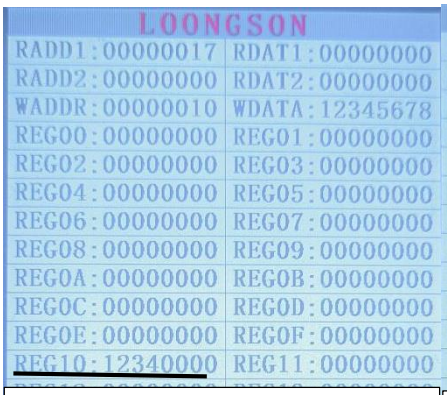
```
set_property PACKAGE_PIN AC22 [get_ports wen3]
set_property PACKAGE_PIN AC23 [get_ports wen2]
set_property PACKAGE_PIN AB6 [get_ports wen1]
set_property PACKAGE_PIN W6 [get_ports wen0]
set_property PACKAGE_PIN AA7 [get_ports ren1]
set_property PACKAGE_PIN Y6 [get_ports ren0]
set_property PACKAGE_PIN AC21 [get_ports input_sel[1]]
set_property PACKAGE_PIN AD24  [get_ports input_sel[0]]

set_property IOSTANDARD LVCMOS33 [get_ports wen0]
set_property IOSTANDARD LVCMOS33 [get_ports wen1]
set_property IOSTANDARD LVCMOS33 [get_ports wen2]
set_property IOSTANDARD LVCMOS33 [get_ports wen3]
set_property IOSTANDARD LVCMOS33 [get_ports ren0]
set_property IOSTANDARD LVCMOS33 [get_ports ren1]
set_property IOSTANDARD LVCMOS33 [get_ports input_sel[1]]
set_property IOSTANDARD LVCMOS33 [get_ports input_sel[0]]
```

5、 实验结果分析

当向地址为 10 的寄存器写入数据 0x12345678 时，此时 wen3、wen2、wen1、wen0 都为 0，所以 REG10 中数据为 0x00000000。

拨动拨码开关时 wen3 为 1，wen2、wen1、wen0 依旧为 0，写入 wdata 的第四个字节，REG10 数据变为 0x12000000。



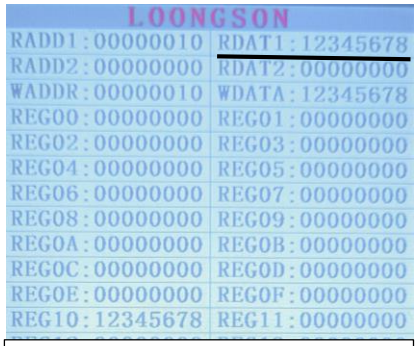
拨动拨码开关 wen2 为 1，wen1、wen0 依旧为 0，写入 wdata 的第三个字节，REG10 数据变为 0x12340000。

拨动拨码开关 wen1 为 1，wen0 依旧为 0，写入 wdata 的第二个字节，此时 REG10 数据变为 0x12345600。

拨动拨码开关 wen0 为 1，写入 wdata 的第一个字节，REG10 数据变为 0x12345678。

第一个读端口读取地址为 10 的寄存器中的数据，此时 ren1、ren0 都为 0，所以读取的 rdata1 为 0x00000000。

拨动拨码开关 ren1 为 1，ren0 依旧为 0，rdata1 读取寄存器中高十六位，rdata1 变为 0x12340000。



拨动拨码开关 ren0 为 1，rdata1 读取寄存器中低十六位，rdata1 变为 0x12345678。

第二个读端口读取地址为 10 的寄存器中的数据，此时 ren1、ren0 都为 0，所以读取的 rdata2 为 0x00000000。

拨动拨码开关 ren1 为 1，ren0 依旧为 0，rdata2 读取寄存器中高十六位，rdata2 变为 0x12340000。

拨动拨码开关 ren0 为 1，rdata2 读取寄存器中低十六位，rdata2 变为 0x12345678。

6、 总结感想

通过本次实验，掌握了寄存器堆的工作原理及设计方法，了解了寄存器写入和读取的具体过程和步骤，也更加能够熟练运用 verilog 语言进行电路设计。