第一章 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

第二章 实验任务与要求

2.1 实验任务

1. 学习 MIPS 计算机中寄存器堆的设计及原理，如：有多少个寄存器，有无特殊设置的寄存器，mips 指令如何去索引寄存器的等。
2. 自行设计本次实验的方案，画出结构框图，详细标出输入输出端口，本次实验建议设计为异步读同步写的寄存器堆，即读寄存器不需要时钟控制，但写寄存器需时钟控制。
3. 本次实验建议寄存器堆设计为 1 个写端口和 2 个读端口，后续 CPU 实验用到的寄存器堆需要 1 个写端口和 2 个读端口。
4. 根据设计的实验方案，使用 verilog 编写相应代码。
5. 对编写的代码进行仿真，得到正确的波形图。
6. 将以上设计作为一个单独的模块，设计一个外围模块去调用该模块。外围模块中需调用封装好的 LCD 触摸屏模块，显示寄存器堆的读写端口地址和数据，最好能扫描出所有寄存器的值显示在 LCD 触摸屏上，并且需要利用触摸功能输入寄存器堆的读写地址和写数据。
7. 将编写的代码进行综合布局布线，并下载到实验箱中的 FPGA 板子上进行演示。

2.2 实验要求

1. 做好预习：1）掌握寄存器堆的工作原理；2）确定寄存器堆的输入输出端口设计；3）在课前画好寄存器堆的设计框图或实验原理图；
2. 实验实施：1）确认寄存器堆的设计框图的正确性；2）编写 verilog 代码；3）对该模块进行仿真，得出正确的波形，截图作为实验报告结果一项的材料；4）完成调用寄存器堆模块的外围模块的设计，并编写代码；5）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. 实验检查：1）完成上板验证后，让指导老师或助教进行检查，进行现场演示，按照检查人员的要求，对特定寄存器读写，可对演示结果进行拍照作为实验报告结果一项的材料。
4. 实验报告的撰写：1）实验结束后，需按照规定的格式完成实验报告的撰写。

第三章 实验结果

3.1 R-S 触发器

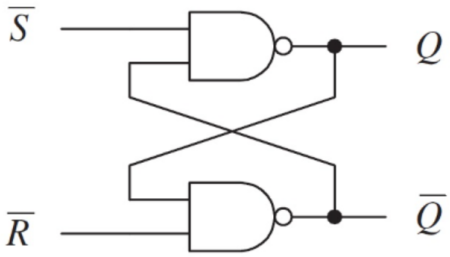


图 3.1 R-S 触发器

```
/**
 * @brief R-S触发器
 * @param S 设置端set
 * @param R 重置端reset
 * @param Q 输出端
 * @param Q_bar 输出端的反相
 * 状态表:
    * R S  | Q_n        Q_bar_n     描述
    * 0 1  | 1      Q_bar_n-1   设置Q为1
    * 1 0  | Q_n-1      1               设置Q_bar为1
    * 1 1  | Q_n-1      Q_bar_n-1       保持前状态
    * 0 0  | 1      1           非法状态（禁止）
 */
module rs_nand_latch(R, S, Q);
    input R, S;
    output Q;

    wire Q_bar;

    nand n1(Q, S, Q_bar);
    nand n2(Q_bar, R, Q);

endmodule // r-s_nand_latch
```

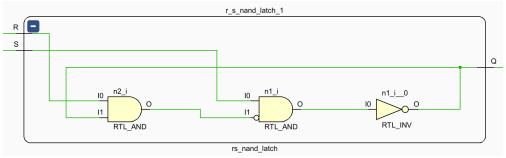
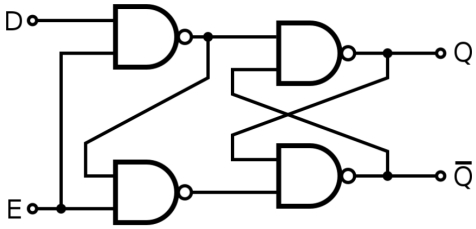


图 3.2 rs 触发器 elaberate design

3.2 D 触发器



A gated D latch based on an $\overline{SR}$ NAND latch

图 3.3 D 触发器

```
/**
 * @brief D触发器
 * @param D 数据端 data
 * @param E 使能端 enable
 * @param Q 输出端
 * @param Q_bar 输出端的反相
    * 状态表:
        * E     D   | Q_n         Q_bar_n      描述
        * 0     X   | Q_n-1       Q_bar_n-1        保持前状态
        * 1     0   | 0       1            重置Q
        * 1     1   | 1       0            设置Q
 */
module d_latch(D, E, Q);
    input D, E;
    output Q;

    wire wire_1, wire_2;

    nand n1(wire_1, D, E);
    nand n2(wire_2, E, wire_1);
```

```
21
22      rs_nand_latch r_s_nand_latch_1(wire_2, wire_1, Q);
23  endmodule // d_latch
```

3.3 D 触发器

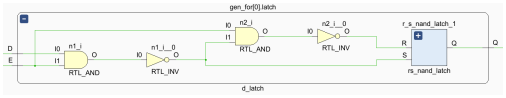


图 3.4 D 触发器 elaberate design

3.4 寄存器

```
1   /**
2    * @brief 32位寄存器
3    */
4   module register_32bit(D, E, Q);
5       input [31:0] D;     // 32-bit data input
6       input E;           // Enable signal (common for all latches)
7       output [31:0] Q;    // 32-bit data output
8
9       // Instantiate 32 D latches, one for each bit
10      // d_latch latches[31:0] (
11      //      .D(D),
12      //      .E(E),
13      //      .Q(Q)
14      // );
15      genvar i;
16      generate
17          for (i = 0; i < 32; i = i + 1) begin: gen_for
18              d_latch latch(
19                  .D(D[i]),
20                  .E(E),
21                  .Q(Q[i])
22              );
23          end
24      endgenerate
```

```
25
26 endmodule // register_32bit
```

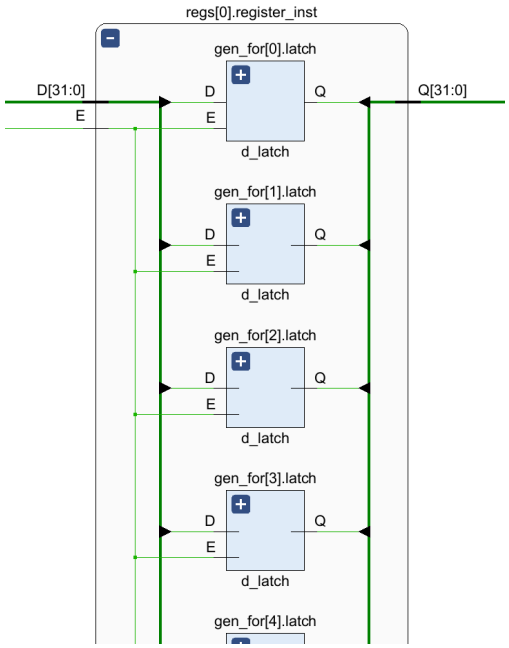


图 3.5 32 位寄存器 elaberated design

3.5 32 位寄存器

3.6 寄存器堆

```
1  module regfile(clk, rst, we, waddr, wdata, raddr1, rdata1, raddr2,
       rdata2, test_addr, test_data);
2      input clk;                  // 时钟信号 clock
3      input rst;                  // 复位信号 reset
4      input we;                   // 写使能信号 write enable
5      input [4:0] waddr;          // 写地址 write address
6      input [31:0] wdata;         // 写数据 write data
7      input [4:0] raddr1;         // 读地址1 read address 1
8      input [4:0] raddr2;         // 读地址2 read address 2
9      input [4:0] test_addr;      // 测试地址 test address
10     output [31:0] rdata1;       // 读数据1 read data 1
11     output [31:0] rdata2;       // 读数据2 read data 2
12     output [31:0] test_data;    // 测试数据 test data
13
14
```

```
15      reg enable[31:0];
16      wire [31:0] reg_outputs[31:0];
17
18      integer i;
19      always @(posedge clk) begin  // 时钟上升沿触发
20          // 复位时，将所有寄存器的使能信号置0
21          if (rst) begin
22              for (i = 0; i < 32; i = i + 1) begin
23                  enable[i] <= 0; // 非阻塞赋值
24              end
25          end else begin
26              // we为1且地址等于waddr时，使能信号为1
27              for (i = 0; i < 32; i = i + 1) begin
28                  if (we && i == waddr) begin
29                      enable[i] <= 1;
30                  end else begin
31                      enable[i] <= 0;
32                  end
33              end
34          end
35      end
36
37      // 32个32位寄存器
38      genvar j;
39      generate
40          for (j = 0; j < 32; j = j + 1) begin : regs
41              register_32bit register_inst (
42                  .D(wdata),
43                  .E(enable[j]),
44                  .Q(reg_outputs[j])
45              );
46          end
47      endgenerate
48
49      // 读数据
50      assign rdata1 = reg_outputs[raddr1];
51      assign rdata2 = reg_outputs[raddr2];
```

```
52
53      // 测试数据，读出寄存器的值显示在触摸屏上
54      assign test_data = reg_outputs[test_addr];
55
56  endmodule //regfile
```

需要注意的是, 本寄存器堆中的 test_addr 和 test_data 是为了方便在触摸屏上显示寄存器的值而添加的.

3.7 tb

```
1   module tb_regfile;
2
3       // Inputs
4       reg clk;
5       reg rst;
6       reg we;
7       reg [4:0] waddr;
8       reg [31:0] wdata;
9       reg [4:0] raddr1;
10      reg [4:0] raddr2;
11
12      // Outputs
13      wire [31:0] rdata1;
14      wire [31:0] rdata2;
15
16      // Instantiate the regfile module
17      regfile uut (
18          .clk(clk),
19          .rst(rst),
20          .we(we),
21          .waddr(waddr),
22          .wdata(wdata),
23          .raddr1(raddr1),
24          .rdata1(rdata1),
25          .raddr2(raddr2),
26          .rdata2(rdata2)
27      );
```

```
28
29      // 生成时钟信号
30      initial begin
31          clk = 0;
32          forever #5 clk = !clk;  // Generate a clock with 10 ns
                period
33      end
34
35      // Test cases
36      initial begin
37          // Initialize Inputs
38          rst = 1; we = 0; waddr = 0; wdata = 0; raddr1 = 0; raddr2 =
                0;
39
40          // Apply reset
41          #20 rst = 0;  // 释放复位信号，确保至少两个时钟周期内的状态
42
43          // Wait for reset to propagate
44          #10;
45
46          // Case 1: 读写同一地址
47          we = 1; waddr = 5; wdata = 32'hAAAA_BBBB;
48          #10;  // Wait a clock cycle for write to complete
49          we = 0; raddr1 = 5; raddr2 = 5;
50          #10;  // Wait a clock cycle to read the data
51
52          // Case 2: 读写不同地址
53          #10; we = 1; waddr = 15; wdata = 32'h1234_5678;
54          #10; we = 0; raddr1 = 15; raddr2 = 5;  // Read new data and
                previous data
55
56          // Case 3: 不写入数据是否保持不变
57          #20; raddr1 = 15; raddr2 = 5;
58
59          #10 $finish;
60      end
61
```

```
62 endmodule // tb_regfile
```

3.8 display

这部分代码是为了在触摸屏上显示寄存器的值, 但是我没能成功的实现它, 不知道为什么程序卡在了生成 bitstream 的地方.

由于这部分代码还有问题, 因此我没有将其放在这里, 想要查看的话可以查看我的 github 仓库, 见附录, 将来有时间我或许会尝试解决这个问题, 并在 github 上更新.

3.9 仿真结果

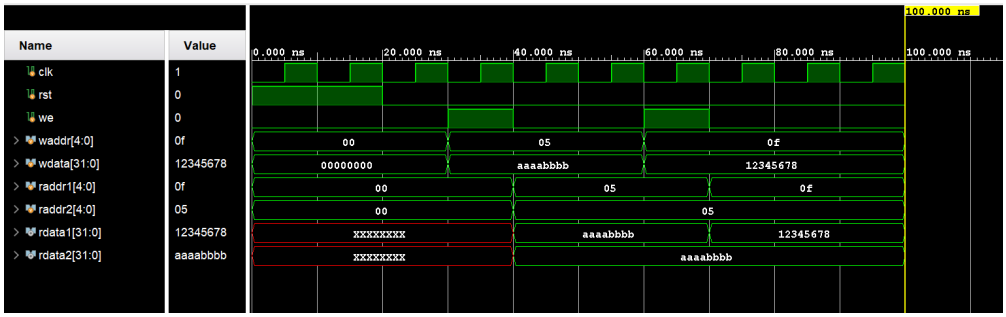


图 3.6 仿真结果

30ns - 40ns: 将 write enable(we) 置为 1, 在 waddr(05) 寄存器写入 wdata(aaaabbbb)

40ns - 60ns: 两个读数据口 raddr1, raddr2 同时置为 05, rdata1, rdata2 成功读取数据 aaaabbbb

60ns - 70ns: 将 we 置为 1, 在 waddr(0f) 寄存器中写入 wdata(12345678)

70ns - 100ns: 两个独守巨口 raddr1, raddr2 一个置为 0f, 一个置为 05, rdata1, rdata2 成功分别读取数据 12345678 和 aaaabbbb

3.10 FPGA 实现

由于 display 的部分我没有完成, 因此我的代码并不能在 FPGA 触控板上运行. 这里的图片是老师给的代码在 FPGA 上的运行结果.

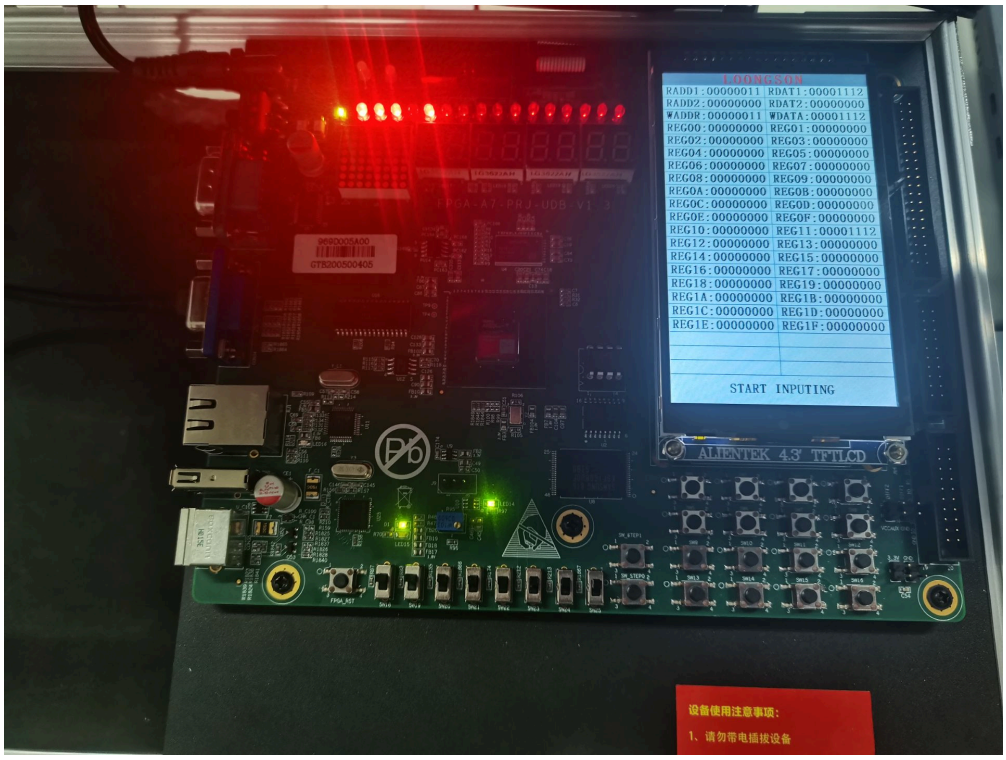

LOONGSON
RADD1:00000011 RDAT1:00001112
RADD2:00000000 RDAT2:00000000
WADDR:00000011 WDATA:00001112
REG00:00000000 REG01:00000000
REG02:00000000 REG03:00000000
REG04:00000000 REG05:00000000
REG06:00000000 REG07:00000000
REG08:00000000 REG09:00000000
REG0A:00000000 REG0B:00000000
REG0C:00000000 REG0D:00000000
REG0E:00000000 REG0F:00000000
REG10:00000000 REG11:00001112
REG12:00000000 REG13:00000000
REG14:00000000 REG15:00000000
REG16:00000000 REG17:00000000
REG18:00000000 REG19:00000000
REG1A:00000000 REG1B:00000000
REG1C:00000000 REG1D:00000000
REG1E:00000000 REG1F:00000000
START INPUTING
ALIENTEK 4.3' TFTLCD
设备使用注意事项:
1、请勿带电插拔设备

图 3.7 FPGA 实现

附　　录

参考链接：https://github.com/zehua0417/ComputerOrganizationAndArchitecture_exp