中山大学数据科学与计算机学院本科生实验报告

（2019 学年秋季学期）

课程名称：**计算机组成原理实验**　　任课教师：　　助教：

年级&班级	19 计科超算	专业(方向)	
学号	18324034	姓名	林天皓
电话		Email	linth5@mail2.sysu.edu.cn
开始日期	2020.11.13	完成日期	2020.11.20

一、　实验题目

实验 8 寄存器堆设计 ALU 实验

二、　实验目的

1.学习和使用 Verlilog HDL 进行电路的设计方法

2.掌握灵活的运用 Verilog HDL 进行各种描述与建模的技巧和方法

3.学习寄存器堆的数据传送与读写工作原理，掌握寄存器堆的设计方法

三、实验内容

1 实验原理

本次实验要求实现一个寄存器堆，并且通过控制信号，和之前已经实现过的 alu 模块交互，包括读取寄存器堆内的数据，将数据传送到运算器，根据结果将数据写回寄存器堆的操作。在本次实验中，为了使得在运算过程中保持平稳，寄存器堆需要设置时钟信号控制，只有在信号的上升沿过程中数据才会发生变化，保障运算过过程中数据的稳定性。分别完成两个模块后，通过一个 top 模块将 alu 模块和寄存器堆模块之间的连线连起来，然后使用仿真文件测试，分析测试的输出数据和波形图。

2. **实验步骤**

第一部分：ALU 模块的实现：

alu 模块包含两个输入的操作数，一个控制信号，一个输出结果。在这里我直接实现 老师在第九周要求实现的带有 ZF,CF,OF,SF,PF 的 alu。得益于 verilog 语言已经提供了+,-,|,&,<<位移运算符，alu 模块实现起来很容易，使用 case 语句根据控制信号直接分为 6 种情况即可。根据控制信号的真值表：

控制信号	操作
0000	按位与
0001	按位或
0010	按位异或
0011	按位或非
0100	加法
0101	减法
0110	A<B?1:0
0111	左移操作

表 1-ALU 控制信号真值表

实现代码如下：

```
module ALU(OP,A,B,F,ZF,CF,OF,SF,PF);
parameter SIZE = 32;//运算位数
input [3:0] OP;//运算操作
input [SIZE:1] A;//左运算数
input [SIZE:1] B;//右运算数
output [SIZE:1] F;//运算结果
output      ZF, //0 标志位，运算结果为 0(全零)则置 1，否则置 0
```

```
            CF, //进借位标志位，取最高位进位 C,加法时 C=1 则 CF=1 表示有进位,减法时 C=0 则
CF=1 表示有借位
            OF, //溢出标志位，对有符号数运算有意义，溢出则 OF=1，否则为 0
            SF, //符号标志位，与 F 的最高位相同
            PF; //奇偶标志位，F 有奇数个 1，则 PF=1，否则为 0
reg [SIZE:1] F;
reg C,ZF,CF,OF,SF,PF;//C 为最高位进位
always@(*)
begin
    C=0;
    case(OP)
        4'b0000:begin F=A&B; end    //按位与
        4'b0001:begin F=A|B; end    //按位或
        4'b0010:begin F=A^B; end    //按位异或
        4'b0011:begin F=~(A|B); end //按位或非
        4'b0100:begin {C,F}=A+B; end //加法
        4'b0101:begin {C,F}=A-B; end //减法
        4'b0110:begin F=A<B; end//A<B 则 F=1，否则 F=0
        4'b0111:begin F=B<<A; end   //将 B 左移 A 位
    endcase
    ZF = F==0;//F 全为 0，则 ZF=1
    CF = C; //进位借位标志
    OF = A[SIZE]^B[SIZE]^F[SIZE]^C;//溢出标志公式
    SF = F[SIZE];//符号标志,取 F 的最高位
    PF = ^F;//奇偶标志，F 有奇数个 1，则 F=1；偶数个 1，则 F=0
end
endmodule
```

CF 为加法或者减法情况下的前面一位。其中包含了对 OF 这个溢出检测的部分，使用 OF=input1[31]^input2[31]^aluRes[31]^CF 判断同号运算后的结果是否出现了异号和进位来判断是否溢出。由于仅在加减法的情况下可能会出现溢出，所以在其他运算中我们将 OF 置 0。

最后处理 ZF，SF,PF 的部分。ZF 为判断运算结果是否为 0，SF 为符号位，PF 判断运算结果中奇偶校验，其中^aluRes 这个运算代表了将 aluRes 中的每一位按位逐一进行异或，并将结果取反，容易得到当 aluRes 中有奇数个 1 的情况下，^aluRes 的结果为 1，偶数个 1，则 PF=0。（注：这里老师给的 PF = ~^ aluRes 又是个错误，应该为 PF = ^ aluRes）

总结：alu 模块处理了两个 32 位数字的八种运算。

生成的 RTL 图如下：

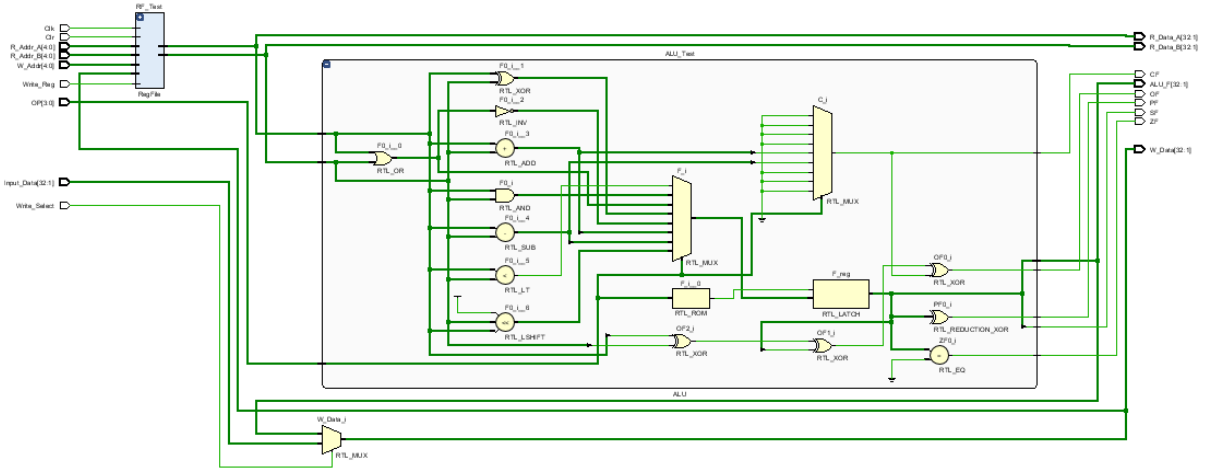

图 1-ALU 模块 RTL 图

第二部分：寄存器堆的实现：

本次寄存器堆的大小位 32 个 32 位的寄存器。

寄存器堆的模块结果如图

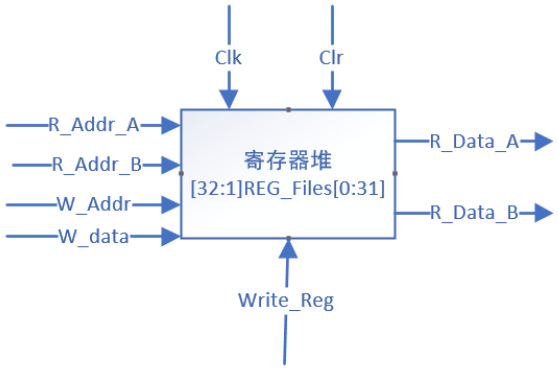

图 2-寄存器堆模块图

需要实现以下几个功能

1.写入功能：寄存器堆的写入受到时钟信号控制，只有时钟信号的上升沿过程，寄存器堆判断写入使能信号是否为高电平，如果为高电平，则根据传入的寄存器写入地址和寄存区写入数据，修改对应的寄存器数据。

2.清零功能：同时有一个清零信号，当遇到清零信号的上升沿时，将所有的 32 个寄存器内储存的值全部清零。

3.读取功能：将输入的两个内存地址，通过 assign 寻址并输出对应信号。

实现代码如下：

```
module RegFile(
           Clk,Clr,Write_Reg,R_Addr_A,R_Addr_B,W_Addr,W_Data,R_Data_A,R_Data_B
       );
parameter ADDR = 5;//寄存器编码/地址位宽
parameter NUMB = 1<<ADDR;//寄存器个数
parameter SIZE = 32;//寄存器数据位宽
input Clk;//写入时钟信号
input Clr;//清零信号
input Write_Reg;//写控制信号
input [ADDR-1:0]R_Addr_A;//A 端口读寄存器地址
input [ADDR-1:0]R_Addr_B;//B 端口读寄存器地址
input [ADDR-1:0]W_Addr;//写寄存器地址
input [SIZE:1]W_Data;//写入数据
output [SIZE:1]R_Data_A;//A 端口读出数据
output [SIZE:1]R_Data_B;//B 端口读出数据
reg [SIZE:1]REG_Files[0:NUMB-1];//寄存器堆本体
integer i;//用于遍历 NUMB 个寄存器
initial//初始化 NUMB 个寄存器，全为 0
    for(i=0;i<NUMB;i=i+1)
        REG_Files[i]<=i;
always@(posedge Clk or posedge Clr)//时钟信号或清零信号上升沿
begin
    if(Clr)//清零
        for(i=0;i<NUMB;i=i+1)
            REG_Files[i]<=0;
    else//不清零,检测写控制，高电平则写入寄存器
        if(Write_Reg)
            REG_Files[W_Addr]<=W_Data;
end         //读操作没有使能或时钟信号控制，使用数据流建模(组合逻辑电路,读不需要时钟控制)
assign R_Data_A=REG_Files[R_Addr_A];
assign R_Data_B=REG_Files[R_Addr_B];
endmodule
```

生成的 RTL 图如下

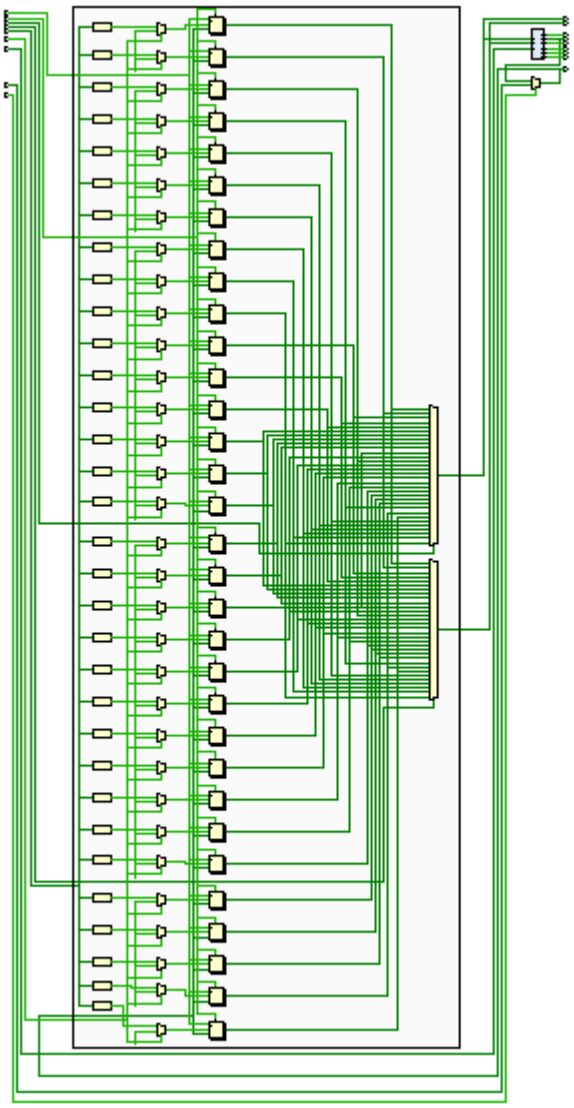

图 3-寄存器堆模块 RTL 图

其中中间靠右的部分一堆连线就是实现

```
assign R_Data_A=REG_Files[R_Addr_A];
assign R_Data_B=REG_Files[R_Addr_B];
```

这两句代码的部分，一开始不是很理解怎样通过 assign 将一个会变化的寻址通过组合电路的方式综合，通过查看生成的 RTL 图发现 vivado 自动生成了一个 MUX 数据选择器模块来完成寻址取数据的操作。

第三部分：顶层模块的实现：

接下来通过 top 模块将 alu 模块和寄存器堆模块连接，同时加入一个外部输入数据方便我们自己写入测试数据。

实现代码如下：

```
module RF_ALU(
           Clk,Clr,Write_Reg,Write_Select,//控制信号
           R_Addr_A,R_Addr_B,W_Addr,//读写地址
           Input_Data,R_Data_A,R_Data_B,//数据 IO
           OP,ZF,CF,OF,SF,PF,ALU_F//ALU 运算
           ,W_Data );
parameter ADDR = 5;//地址位宽
parameter SIZE = 32;//数据位宽          //寄存器堆
input Clk, Clr;//写入时钟信号，清零信号
input Write_Reg;//写控制信号
input [ADDR-1:0]R_Addr_A;//A 读端口寄存器地址
input [ADDR-1:0]R_Addr_B;//B 读端口寄存器地址
input [ADDR-1:0]W_Addr;//写寄存器地址
input [SIZE:1]Input_Data;//外部输入数据

output [SIZE:1]R_Data_A;//A 端口读出数据
output [SIZE:1]R_Data_B;//B 端口读出数据          //ALU
input [3:0] OP;//运算符编码
output ZF,//零标志
       CF,//进借位标志(只对无符号数运算有意义)
       OF,//溢出标志(只对有符号数运算有意义)
       SF,//符号标志(只对有符号数运算有意义)
       PF;//奇偶标志
output [SIZE:1] ALU_F;//运算结果 F
input wire Write_Select;//写入数据选择信号
wire [SIZE:1]ALU_F;//ALU 运算结果中间变量
output  reg [SIZE:1]W_Data;//写入数据     //Write_Select 高电平则写外部输入，否则写运
算结果
always@(*)
begin
    case (Write_Select)
        1'b0: W_Data=ALU_F;
        1'b1: W_Data=Input_Data;
    endcase
end
RegFile RF_Test(         //输入
            .Clk(Clk),//时钟信号
            .Clr(Clr),//清零信号
            .Write_Reg(Write_Reg),//写入控制
            .R_Addr_A(R_Addr_A),//A 端口读地址
            .R_Addr_B(R_Addr_B),//B 端口读地址
            .W_Addr(W_Addr),//写入地址
            .W_Data(W_Data),//写入数据，由外部或 ALU 输入                  //输出
            .R_Data_A(R_Data_A),//A 端口读出数据
```

```
            .R_Data_B(R_Data_B)//B 端口读出数据
        );              //实例化 ALU 模块
ALU ALU_Test(     //输入
        .OP(OP),//运算符
        .A(R_Data_A),//从寄存器读 A 操作数
        .B(R_Data_B),//从寄存器读 B 操作数
        .F(ALU_F),//ALU_F 作为中间变量暂存运算结果，与 Input_Data 选择输入寄存
器                  //输出
        .ZF(ZF),//零标志
        .CF(CF),//进借位标志(只对无符号数运算有意义)
        .OF(OF),//溢出标志(只对有符号数运算有意义)
        .SF(SF),//符号标志(只对有符号数运算有意义)
        .PF(PF)//奇偶标志
    );
endmodule
```

生成的 RTL 图如下：

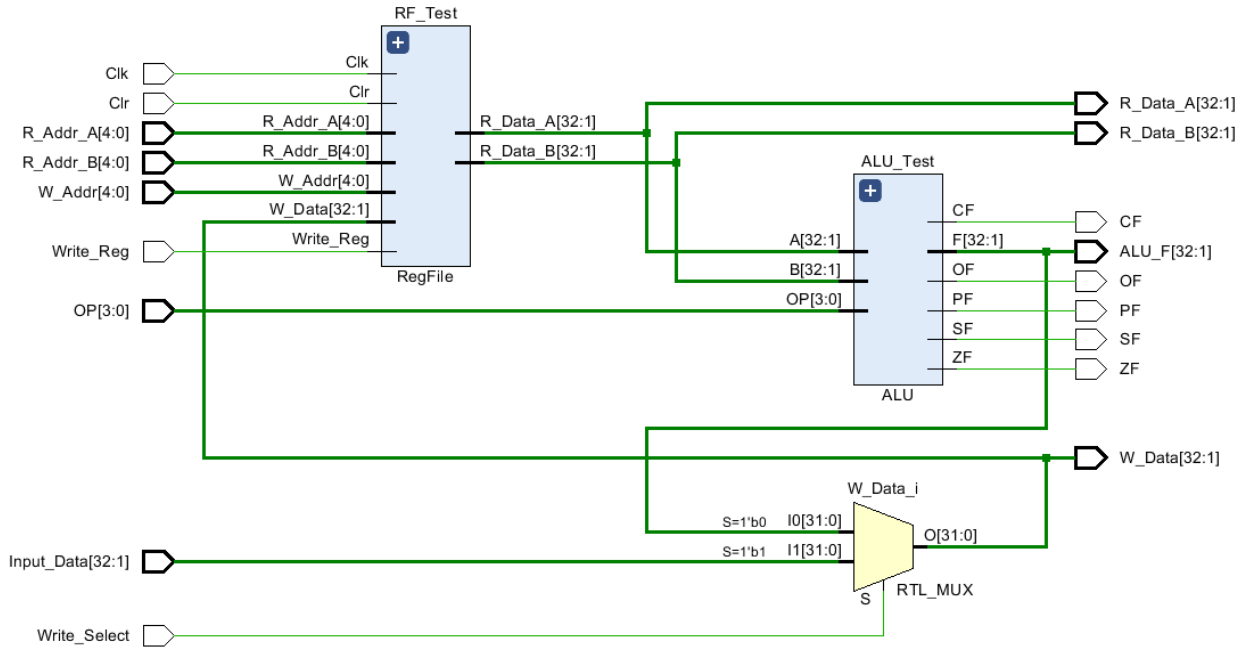

图 4-顶层模块模块 RTL 图

第四部分：仿真模块的实现：

由于老师给的仿真文件中，有两段 initial 代码段，由于 verilog 语言中，各个 initial 代码段是在开始的阶段是同时执行的，所以有多段 initial 代码段的情况下，较难分清语句的执行顺序，于是打算重写了 test 模块。

为了尽可能测试寄存器堆的功能，设计以下几个操作来测试

操作序号	寄存器操作	ALU 操作，地址	写入地址
1	直接写入 7	X	1
2	直接写入 4	X	2
3	从 ALU 写入	加，1，2	3
4	从 ALU 写入	减，1，2	4
5	从 ALU 写入	或，1，2	5
6	从 ALU 写入	异或，1，2	6
7	从 ALU 写入	左移，1，2	7

表 2-仿真测试操作表

结根据上述操作表，仿真测试代码如下：

```
module Test();
reg  Clk, Clr, Write_Reg;
reg Write_Select;
reg [4:0] R_Addr_A ,R_Addr_B, W_Addr;
reg [31:0]  Input_Data;
reg [3:0] OP;
wire [31:0] R_Data_A, R_Data_B, ALU_F;
wire ZF,CF,OF,SF,PF;
wire [31:0]  W_Data;

initial begin
    $timeformat(-9, 0, " ns");
    Clk=0;#50;//清零
    Clk=1;#50; //寄存器清零
    Clr=0;
    Clk=0;#50;
    Clr=1;
    Clk=1;#50;
    Clr=0;
    Write_Reg=1;Write_Select=1'b1;Input_Data=32'h00000007; W_Addr=5'b00001;//寄存器 1 写入 7
    Clk=0;#50;
    Clk=1;#50;
    Write_Select=1'b1;Input_Data=32'h00000004; W_Addr=5'b00010;//寄存器 1 写入 4
    Clk=0;#50;
```

```
    Clk=1;#50;
    Write_Select=1'b0;R_Addr_A=5'b00001;R_Addr_B=5'b00010;OP=4'b0100;W_Addr=5'b00011;//加法
    Clk=0;#50;
    Clk=1;#50;
    $display ("%0t R_Data_A=%1d R_Data_B=%1d OP=+ ALU_F=%3d ZF=%1b CF=%1b OF=%1b SF=%1b PF=%1b",
    $time, R_Data_A,R_Data_B,ALU_F,ZF,CF,OF,SF,PF);

    Write_Select=1'b0;R_Addr_A=5'b00001;R_Addr_B=5'b00010;OP=4'b0101;W_Addr=5'b00100;//减法
    Clk=0;#50;
    Clk=1;#50;
    $display ("%0t R_Data_A=%1d R_Data_B=%1d OP=+ ALU_F=%3d ZF=%1b CF=%1b OF=%1b SF=%1b PF=%1b",
    $time, R_Data_A,R_Data_B,ALU_F,ZF,CF,OF,SF,PF);

    Write_Select=1'b0;R_Addr_A=5'b00001;R_Addr_B=5'b00010;OP=4'b0001;W_Addr=5'b00101;//或
    Clk=0;#50;
    Clk=1;#50;
    $display ("%0t R_Data_A=%1d R_Data_B=%1d OP=+ ALU_F=%3d ZF=%1b CF=%1b OF=%1b SF=%1b PF=%1b",
    $time, R_Data_A,R_Data_B,ALU_F,ZF,CF,OF,SF,PF);

    Write_Select=1'b0;R_Addr_A=5'b00001;R_Addr_B=5'b00010;OP=4'b0010;W_Addr=5'b00110;//异或
    Clk=0;#50;
    Clk=1;#50;
    $display ("%0t R_Data_A=%1d R_Data_B=%1d OP=+ ALU_F=%3d ZF=%1b CF=%1b OF=%1b SF=%1b PF=%1b",
    $time, R_Data_A,R_Data_B,ALU_F,ZF,CF,OF,SF,PF);

    Write_Select=1'b0;R_Addr_A=5'b00001;R_Addr_B=5'b00010;OP=4'b0111;W_Addr=5'b00111;//左移
    Clk=0;#50;
    Clk=1;#50;
    $display ("%0t R_Data_A=%1d R_Data_B=%1d OP=+ ALU_F=%3d ZF=%1b CF=%1b OF=%1b SF=%1b PF=%1b",
    $time, R_Data_A,R_Data_B,ALU_F,ZF,CF,OF,SF,PF);
    Clk=0;#50;
    Clk=1;#50;

end         //实例化寄存器堆模块
RF_ALU RF_Test(
            .Clk(Clk),
            .Clr(Clr),
            .Write_Reg(Write_Reg),
            .Write_Select(Write_Select),
            .R_Addr_A(R_Addr_A),
            .R_Addr_B(R_Addr_B),
            .W_Addr(W_Addr),
            .Input_Data(Input_Data),
            .R_Data_A(R_Data_A),
            .R_Data_B(R_Data_B),
            .OP(OP),
```

```
            .ZF(ZF),
            .CF(CF),
            .OF(OF),
            .SF(SF),
            .PF(PF),
            .ALU_F(ALU_F),
            .W_Data(W_Data)
        );

endmodule
```

到这里所有的模块设计完成，接下来进行仿真测试。

四、实验结果

仿真波形图结果如下：

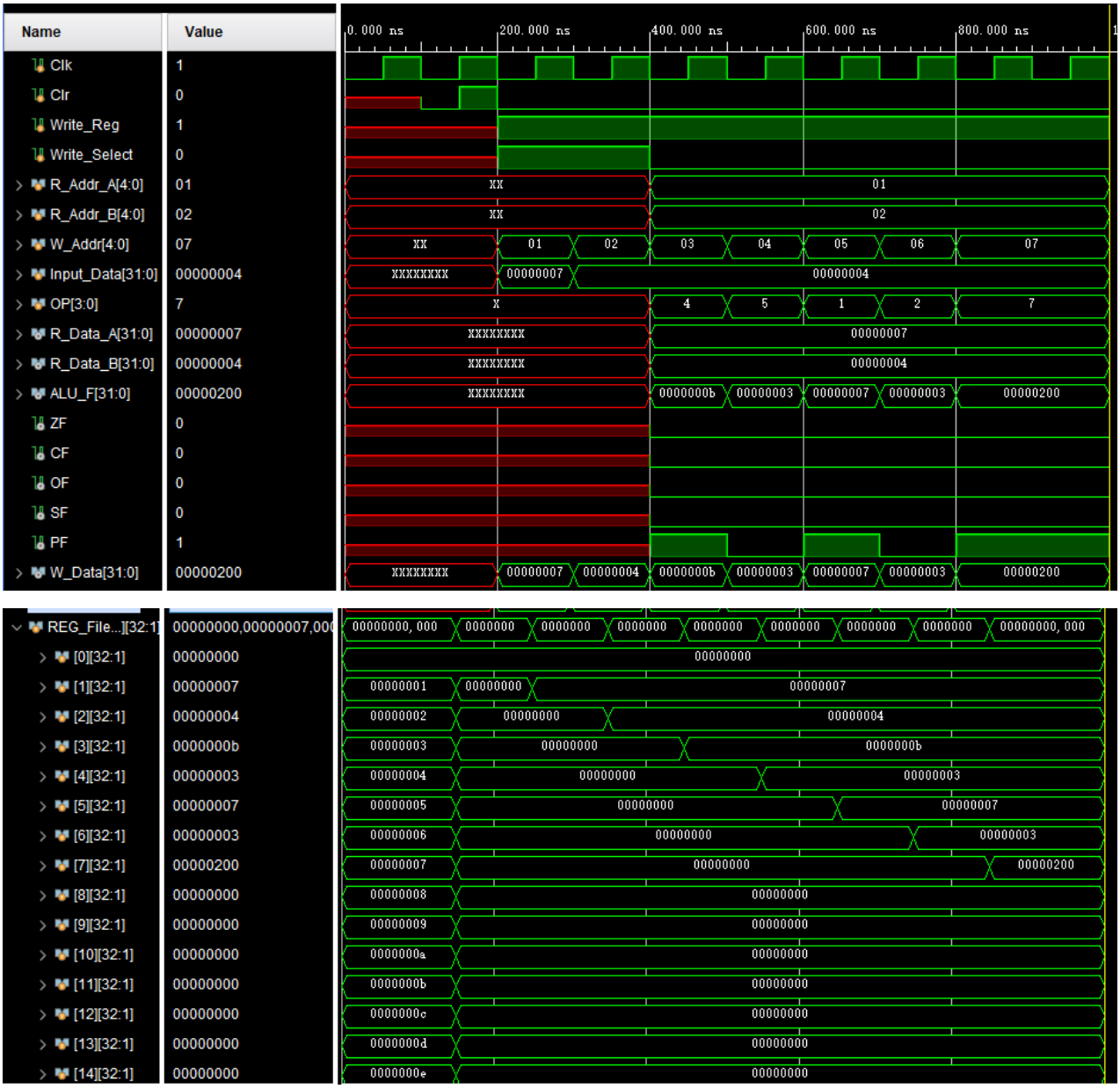

图 5 图 6-仿真测试结果波形图

Vivado 输出如下：

```
# run 1000ns
500 ns R_Data_A=7 R_Data_B=4 OP=+ ALU_F= 11 ZF=0 CF=0 OF=0 SF=0 PF=1
600 ns R_Data_A=7 R_Data_B=4 OP=- ALU_F=  3 ZF=0 CF=0 OF=0 SF=0 PF=0
700 ns R_Data_A=7 R_Data_B=4 OP=or ALU_F=  7 ZF=0 CF=0 OF=0 SF=0 PF=1
800 ns R_Data_A=7 R_Data_B=4 OP=xor ALU_F=  3 ZF=0 CF=0 OF=0 SF=0 PF=0
900 ns R_Data_A=7 R_Data_B=4 OP=<< ALU_F=512 ZF=0 CF=0 OF=0 SF=0 PF=1
```

图 7-仿真测试输出结果

分析：

操作 1：第二个时钟周期时，清零信号上升沿，所有寄存器清零。

操作 2：第三个时钟周期上升沿时，向寄存器 1 写入数字 7，由图可见写入成功。

操作 3：第四个时钟周期上升沿时，向寄存器 2 写入数字 4，由图可见写入成功。

操作 4：第五个时钟周期上升沿时，设置 alu 操作为加法，操作数分别为寄存器 1 和寄存器 2 对应的 7 和 4，并且将结果写入寄存器 3，由图寄存器 3 的值为 4+7=0xb，结果正确。

操作 5：第六个时钟周期上升沿时，设置 alu 操作为减法，操作数分别为寄存器 1 和寄存器 2 对应的 7 和 4，并且将结果写入寄存器 4，由图寄存器 4 的值为 7-4=0x3，结果正确。

操作 6：第六个时钟周期上升沿时，设置 alu 操作为按位或，操作数分别为寄存器 1 和寄存器 2 对应的 7 和 4，并且将结果写入寄存器 5，由图寄存器 5 的值为 4&7=0x7，结果正确。

操作 7：第七个时钟周期上升沿时，设置 alu 操作为按位异或，操作数分别为寄存器 1 和寄存器 2 对应的 7 和 4，并且将结果写入寄存器 6，由图寄存器 6 的值为 4|7=0x3，结果正确。

操作 8：第八个时钟周期上升沿时，设置 alu 操作为左移，操作数分别为寄存器 1 和寄存器 2 对应的 7 和 4，并且将结果写入寄存器 7，由图寄存器 7 的值为 4<<7=0x200=512，结果正确。

经过多次实验，其他的操作与运算结果均无误。

综上，该实验完成了通过控制信号与 alu 交互并能从外部读取和写入的寄存器堆。

五、实验感想

本次实验完成了另一个 CPU 设计中的基础部件，寄存器堆，距离完成 CPU 的设计又更近了一步。

本次实验中，在仿真的时候一开始不能在显示波形的界面中查看 32 个寄存器的值的变化，为 debug 造成了一定的困扰，后来经过查阅资料，发现是因为寄存区是子 module 中的临时变量，默认不会出现在波形图中，需要手动点击子模块添加进入波形窗口后再重新仿执行仿真即可查看寄存器堆内的值的变化。

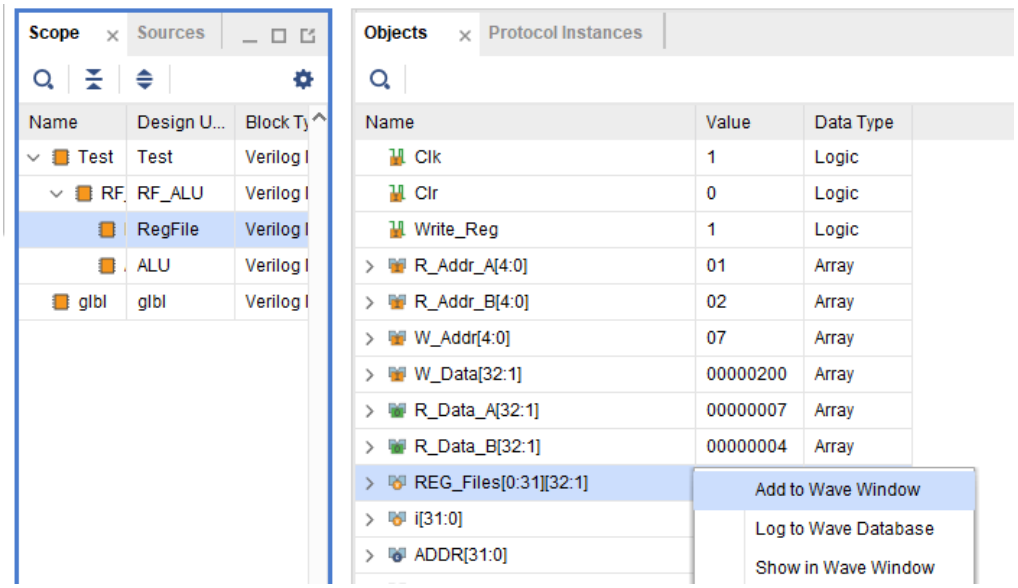

图 8-手动添加到波形窗口中实时查看寄存器堆的值

同时，对于之前认为的 assign 只能用于与，或，非等基本的逻辑运算不同，本次实验中出现了 assign a=b[c]这样通过寻址的方式 assign 的语句，一开始并不理解其中的原理，通过查看综合生成的 RTL 图，得知这部分实际上对应了在数字逻辑电路课程中所实现过的数据选择器。

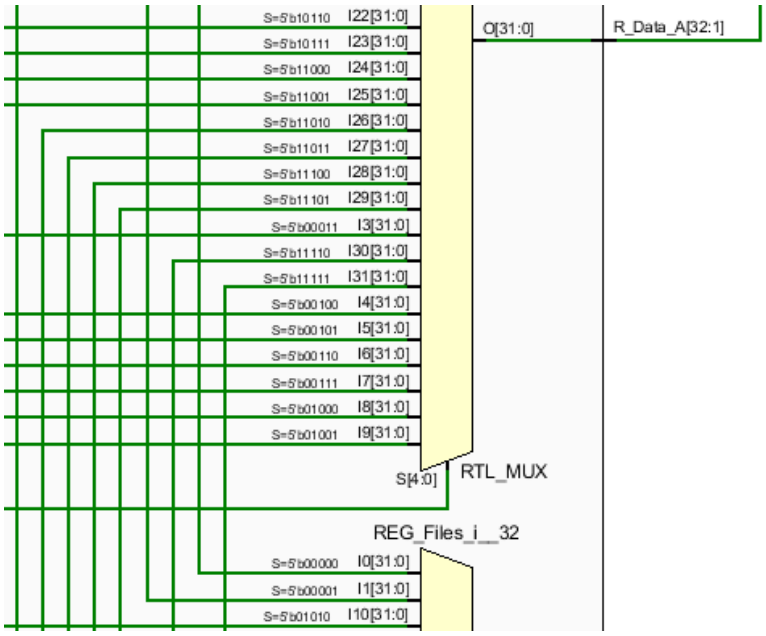

图 9-assign 语句对应的数据选择器模型

附录（流程图，注释过的代码）：

模块关系图：

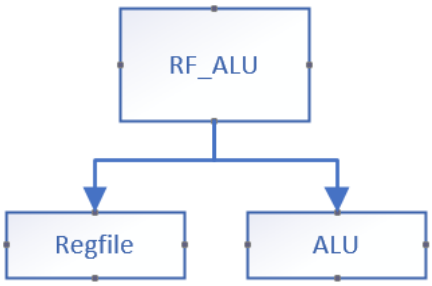

图 10-模块关系图

本次实验代码均已在上文中出现，此处不再重复。