杭州電子科技大学

计算机组成原理（甲）

实 验 报 告

学　　院	网络空间安全学院
专　　业	网络工程
班　　级	19272401
学　　号	19061440
学生姓名	F001
教师姓名	袁理峰
完成日期	2020-12-04
成　　绩	

实验四 寄存器堆设计实验

一、 实验目的

（1） 学习使用 Verilog HDL 进行时序电路的设计方法

（2） 掌握灵活运用 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

二、 实验原理

寄存器的相关原理，并以堆的形式存放。

CPU 内部通常包含若干个通用寄存器，以暂存参加运算的数据和中间结果。寄存器速度快，个数小，但是 RISC CPU 的设计强调设置大量的寄存器。

所谓寄存器堆，就是一个寄存器集合，为方便访问其中的寄存器，对寄存器堆中的寄存器进行统一编码，成为寄存器号或寄存器地址。每个寄存器均通过指定寄存器号进行访问。

本设宴设计了一个 32*32 位的寄存器对，即含有 32 个寄存器，每个寄存器 32 位。该寄存器堆有两个读端口、1 个写端口，即能够同时读出两个寄存器的值，写入 1 个寄存器，下图为其功能表：

输入信号					输出信号		操作
R_Addr_A	R_Addr_B	Write_Reg	W_Addr	W_Data	R_Data_A	R_Data_B	
寄存器号	——	——	——	——	A 口数据	——	读 A 口
——	寄存器号	——	——	——	——	B 口数据	读 B 口
——	——	1	寄存器号	写入数据	——	——	写操作

三、 主要操作步骤及实验结果记录

（对实验过程中的主要操作步骤进行描述，并随时记录实验过程中观察到的结果，必要时可辅助截图）

任务一：在 Xilinx ISE 中创建工程，编辑程序源代码，然后编译、综合，若编译出错，则修改程序代码，直至正确。

```
20  //////////////////////////////////////////////////////////////////////////////////
21  module ALU_REG(
22      input Clk,
23      input Reset,
24      input [4:0] R_Addr_A,
25      input [4:0] R_Addr_B,
26      input [4:0] W_Addr,
27      input Write_Reg,
28      input [2:0] ALU_OP,
29      output ZF,
30      output OF,
31      output [31:0] R_Data_A,
32      output [31:0] R_Data_B,
33      output [31:0] W_Data
34      );
35
36
37    assign R_Data_A = 32'h0000_0000;
38    assign R_Data_B = 32'h0000_0000;
39    assign W_Data = 32'h1010_0100;
40
41
42    MIPS_REG Registers(Clk,Reset,R_Addr_A,R_Addr_B,W_Addr,W_Data,Write_Reg,R_Data_A,R_Data_B);
43
44
45    ALU_8F MIPS_ALU(R_Data_A,R_Data_B,ALU_OP,W_Data,ZF,OF);
46
47
48
49  endmodule
50
```

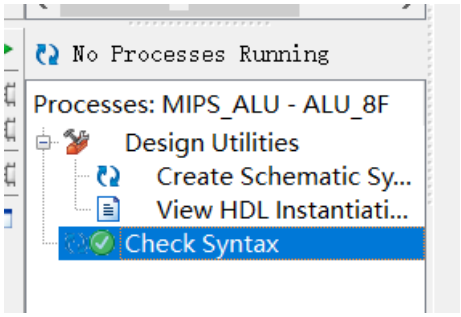
Design Summary　　ALU_REG.v

```
20  //////////////////////////////////////////////////////////////////////////////////
21  module MIPS_REG(
22      input Clk,
23      input Reset,
24      input [4:0] R_Addr_A,
25      input [4:0] R_Addr_B,
26      input [4:0] W_Addr,
27      input [31:0] W_Data,
28      input Write_Reg,
29      output [31:0] R_Data_A,
30      output [31:0] R_Data_B
31      );
32
33     reg [31:0] REG_Files[1:31];
34     integer i;  //初始化
35
36     assign R_Data_A = (R_Addr_A==5'b00000)? 32'h0000_0000 : REG_Files[R_Addr_A];
37     assign R_Data_B = (R_Addr_B==5'b00000)? 32'h0000_0000 : REG_Files[R_Addr_B];
38
39     always @(posedge Clk or posedge Reset)
40     begin
41        if(Reset)
42           begin
43              for(i=1;i<=31;i=i+1)
44                 REG_Files[i]<= 32'h0000_0000;
45           end
46        else
47           begin
48              if ((Write_Reg) &&(W_Addr != 5'b00000))
49                 REG_Files[W_Addr] <= W_Data;
50           end
51     end
52
53  endmodule
54
```

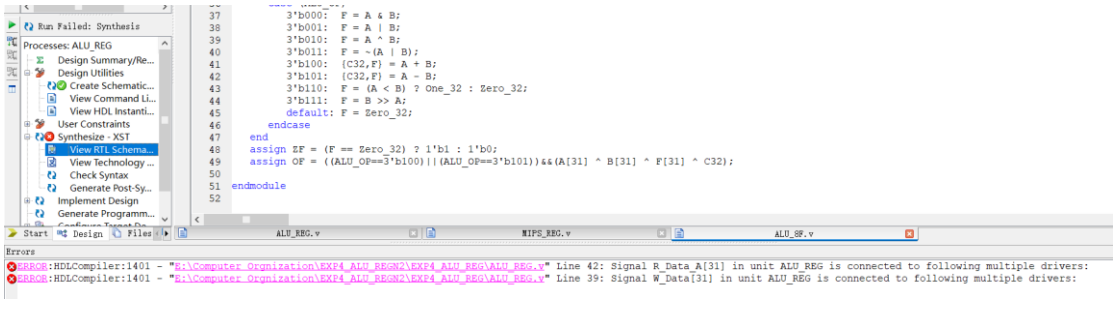
Design Summary | ALU_REG.v | MIPS_REG.v

```
18  // Additional Comments:
19  //
20  //////////////////////////////////////////////////////////////////////////////////
21  module ALU_8F(
22      input [31:0] A,
23      input [31:0] B,
24      input [2:0] ALU_OP,
25      output reg [31:0] F,
26      output ZF,
27      output OF
28      );
29
30     parameter Zero_32 = 32'h0000_0000, One_32 = 32'h0000_0001;
31     reg C32;
32
33     always @(*)
34     begin
35        C32 = 1'b0;
36        case (ALU_OP)
37           3'b000:  F = A & B;
38           3'b001:  F = A | B;
39           3'b010:  F = A ^ B;
40           3'b011:  F = ~(A | B);
41           3'b100:  {C32,F} = A + B;
42           3'b101:  {C32,F} = A - B;
43           3'b110:  F = (A < B) ? One_32 : Zero_32;
44           3'b111:  F = B >> A;
45           default: F = Zero_32;
46        endcase
47     end
48     assign ZF = (F == Zero_32) ? 1'b1 : 1'b0;
49     assign OF = ((ALU_OP==3'b100)||(ALU_OP==3'b101))&&(A[31] ^ B[31] ^ F[31] ^ C32);
50
51  endmodule
52
```

ALU_REG.v | MIPS_REG.v | ALU_8F.v



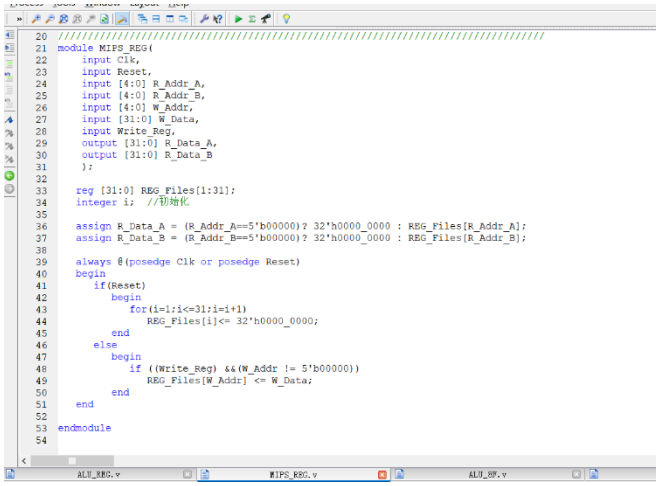
尽管通过了语法检查，但是在查看电路的过程中，出现了相关错误



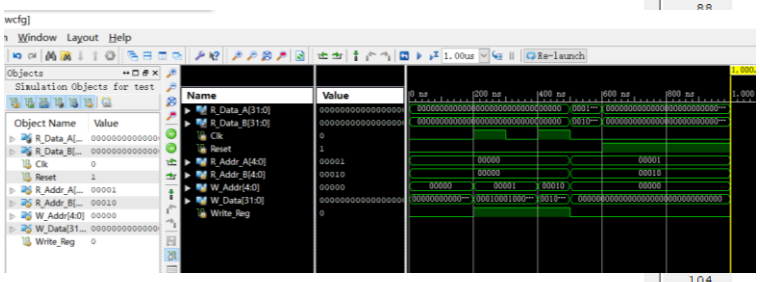
上网查询后得知，该错误是由于在不同的子程序内调修改了相同变量的值，导致最终结果不知道依据哪个程序进行，导致程序出现错误。

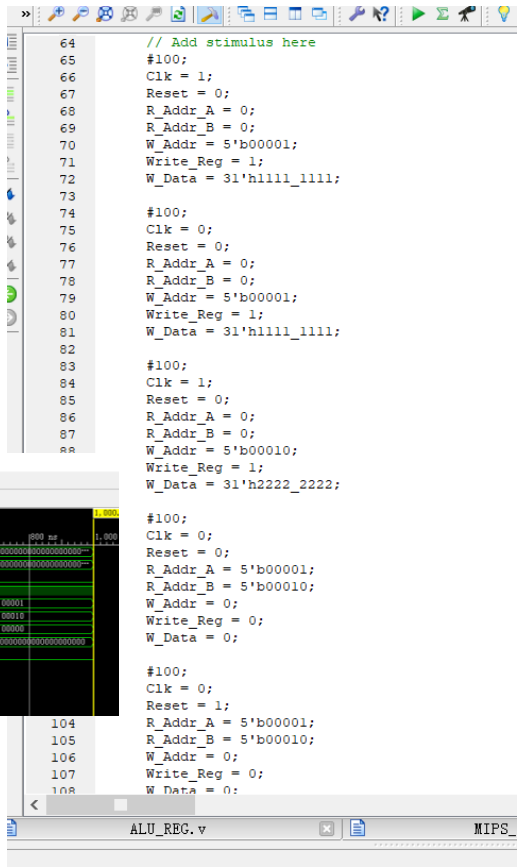
但是奇怪的是，该问题似乎难以解决。相同的代码，在不同的终端上运行，结果不同（老师电脑没问题我的电脑有问题，机房电脑有问题，且都是相同的问题），尽管了解了问题的原因，但是经过无数次修改之后，最后还是有问题，难以观察其电路图。

任务二：编写激励代码，观察仿真波形。如果验证逻辑有误，则修改代码，重新编译、仿真，直至正确。



```
20  //////////////////////////////////////////////////////////////////////////////////
21  module MIPS_REG(
22      input Clk,
23      input Reset,
24      input [4:0] R_Addr_A,
25      input [4:0] R_Addr_B,
26      input [4:0] W_Addr,
27      input [31:0] W_Data,
28      input Write_Reg,
29      output [31:0] R_Data_A,
30      output [31:0] R_Data_B
31      );
32
33      reg [31:0] REG_Files[1:31];
34      integer i;  //初始化
35
36      assign R_Data_A = (R_Addr_A==5'b00000)? 32'h0000_0000 : REG_Files[R_Addr_A];
37      assign R_Data_B = (R_Addr_B==5'b00000)? 32'h0000_0000 : REG_Files[R_Addr_B];
38
39      always @(posedge Clk or posedge Reset)
40      begin
41          if(Reset)
42              begin
43                  for(i=1;i<=31;i=i+1)
44                      REG_Files[i]<= 32'h0000_0000;
45              end
46          else
47              begin
48                  if ((Write_Reg) &&(W_Addr != 5'b00000))
49                      REG_Files[W_Addr] <= W_Data;
50              end
51      end
52
53  endmodule
54
```

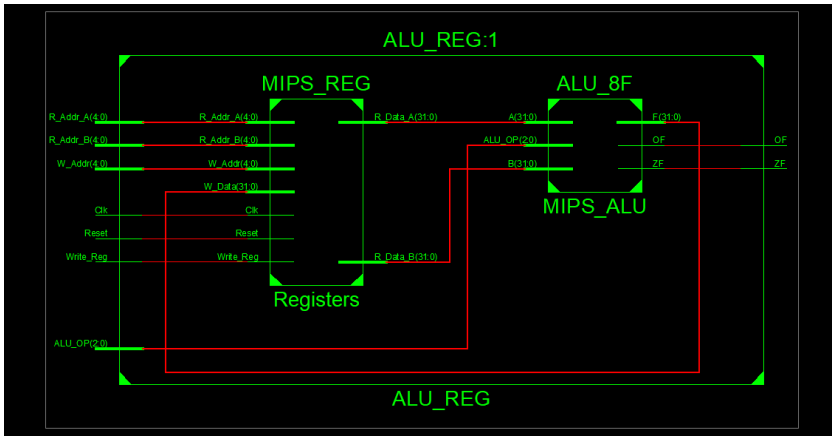




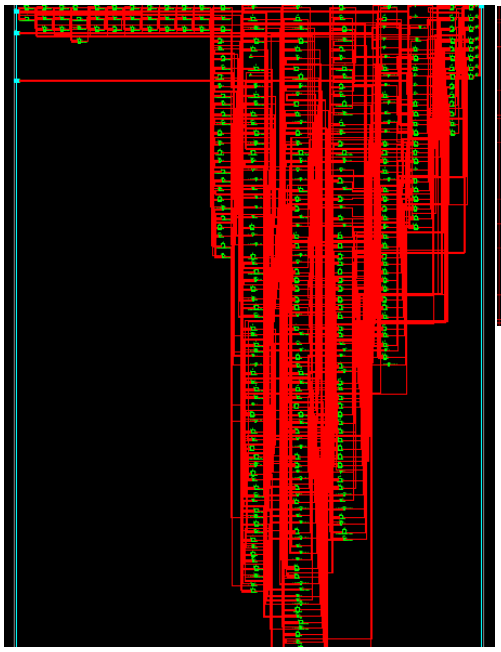
```
64          // Add stimulus here
65          #100;
66          Clk = 1;
67          Reset = 0;
68          R_Addr_A = 0;
69          R_Addr_B = 0;
70          W_Addr = 5'b00001;
71          Write_Reg = 1;
72          W_Data = 31'h1111_1111;
73
74          #100;
75          Clk = 0;
76          Reset = 0;
77          R_Addr_A = 0;
78          R_Addr_B = 0;
79          W_Addr = 5'b00001;
80          Write_Reg = 1;
81          W_Data = 31'h1111_1111;
82
83          #100;
84          Clk = 1;
85          Reset = 0;
86          R_Addr_A = 0;
87          R_Addr_B = 0;
88          W_Addr = 5'b00010;
89          Write_Reg = 1;
90          W_Data = 31'h2222_2222;
91
92          #100;
93          Clk = 0;
94          Reset = 0;
95          R_Addr_A = 5'b00001;
96          R_Addr_B = 5'b00010;
97          W_Addr = 0;
98          Write_Reg = 0;
99          W_Data = 0;
100
101         #100;
102         Clk = 0;
103         Reset = 1;
104         R_Addr_A = 5'b00001;
105         R_Addr_B = 5'b00010;
106         W_Addr = 0;
107         Write_Reg = 0;
108         W_Data = 0;
```

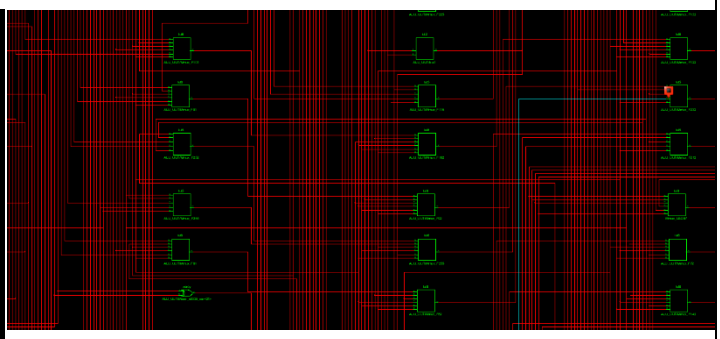
画出仿真波形后，发现程序内部好像是正常执行的，但是总是难以查看电路图。

与其他同学交流后，我了解到其内部的原理如下图所示



内部具体结构更为复杂：





四、 实验分析总结及心得

（结合所学知识对实验过程中观察到的实验结果进行分析总结，以便加深对知识的理解，并总结通过实验学到的知识或技术）

上次遇到问题的时间已经很久了，宝贵的问题终于久别重逢。

此次实验中，尽管所有同学使用的代码相同，但是一大部分人出现了上述的错误。经查询，该问题是由于“在不同的进程里面同时对同一个寄存器进行了赋值当在不同进程里面对同一个寄存器赋值的时候 编译器无法判断寄存器的值到底该被哪个进程赋值”这确实是个问题，但是尽管尝试了更改变量名，调整函数内部语句等多种操作，该错误仍然会出现。但是若将出错行注释掉，内容又相应正常。

更为奇怪的是，该段代码在不同终端运行时的结果不尽相同。有相同错误的，有一切正常的，我进行的猜测是可能是操作系统的环境，以及配置 ISE 时的相关环境上的区别，导致的这个现象发生。另一种猜测是，由于 ISE 可能涉及相关硬件的操作，这是由于硬件本身的复杂性导致的，出现的某种“玄学错误”。例如接触不良、引脚松动等，当然，这些只是硬件本身的问题，并不是引用在该问题上，仅是一种启发与思考。

在这个实验中，32*32 位的寄存器堆内部详细结构确实十分复杂，现实中真正使用的 CPU 结构必然更为复杂，不由得对相关工程师心生敬意。