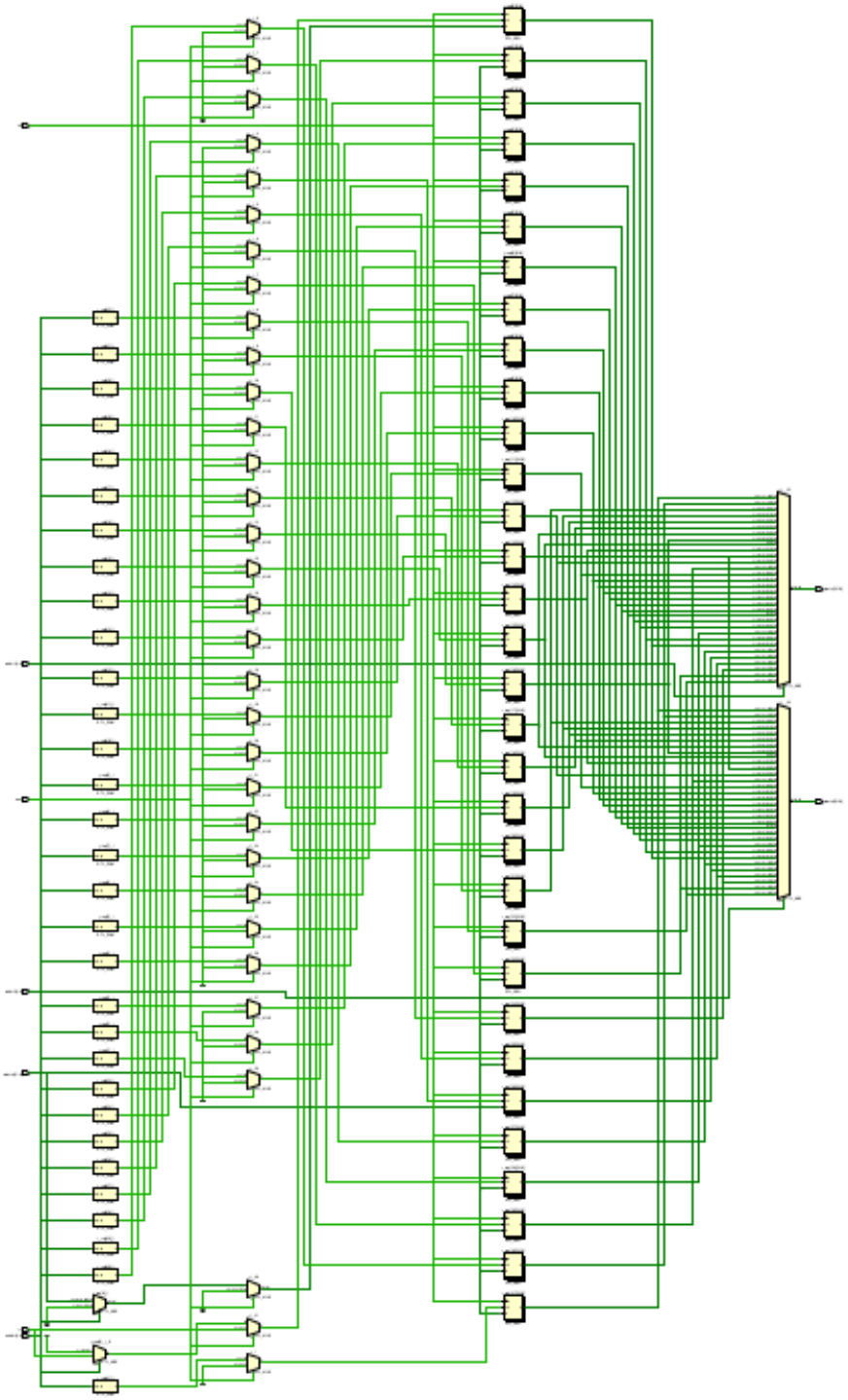
Reg_file 寄存器堆

实验结构图



实验内容

寄存器堆(register file)是 CPU 中多个寄存器组成的阵列，通常由快速的静态随机读写存储器(SRAM)实现。这种 RAM 具有专门的读端口与写端口，可以多路并发访问不同的寄存器。

CPU 的指令集架构总是定义了一批寄存器,用于在内存与 CPU 运算部件之间暂存数据。在更为简化的 CPU，这些架构寄存器（architectural registers）与 CPU 内的物理存在的寄存器一一对应。在更为复杂的 CPU，使用寄存器重命名技术，使得执行期间哪个架构寄存器对应于哪个寄存器堆的物理存储条目（physical entry stores）是动态改变的。寄存器堆是指令集架构的一部分，程序可以访问，

本次实验需要设计 MIPS 处理器中使用的 32-bit 通用寄存器堆（Register File），包含 32 个 32-bit 通用寄存器的寄存器堆

a) 支持 2 读 1 写端口

b) 理解“同步写、异步读”的基本原理

为后续完整 MIPS CPU 设计实验准备好可用组件。

实验说明

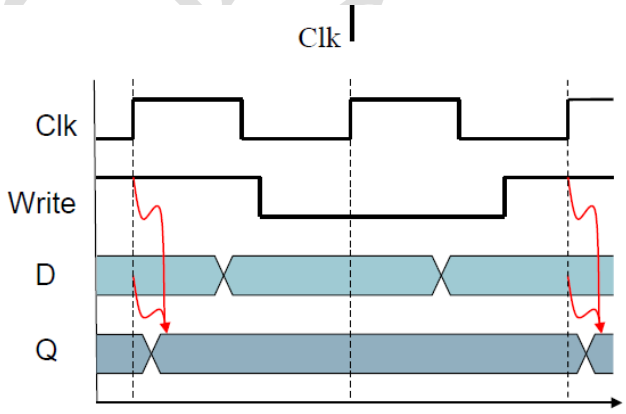
1. 支持 32 个 32-bit 寄存器堆输入输出端口定义

信号名	I/O	说明
clk	Input	时钟
rst	Input	高电平同步复位（仅对0号寄存器有效）
waddr[4:0]	Input	写口地址
raddr1[4:0]	Input	读口地址1
raddr2[4:0]	Input	读口地址2
wen	Input	写使能
wdata[31:0]	Input	写数据
rdata1[31:0]	Output	读口1数据
rdata2[31:0]	Output	读口2数据

2. 写使能信号 wen：

Write Enable:

- =0: Data Out will not change
- =1: Data Out will become Data In (on the clock edge)
  - Only updates on clock edge when write control input is 1
  - Used when stored value is required later



3. 一个 1 位寄存器示例：

```
wire clk;

reg r;

wire data;

always @(posedge clk)

begin

  r <= data;

end
```

4. 寄存器组：

- ❑ 多个存储长度相同的寄存器的组合
- ❑ 不能整体访问
- ❑ 地址 = 寄存器的编号
  - 读出5号寄存器的内容
  - 把数据写入17号寄存器

```
wire clk;
reg r [7:0];
wire rdata;
wire wdata;
wire [2:0] raddr;
wire [2:0] waddr;

always @(posedge clk) begin
    r[waddr] <= wdata;
end

assign rdata = r[raddr];
```

- ❑ 由8个1位寄存器组成的寄存器组
  - 通过二维寄存器引入"地址"的概念
  - 带有1个读口和1个写口
    - 异步读 = 读操作不受时钟约束 = 组合逻辑
    - 同步写 = 写操作受时钟约束 = 时序逻辑
  - 读写地址独立访问

5. MIPS 中的通用寄存器

a) 由 32 个 32 位寄存器组成的寄存器组

b) 两个读口+ 一个写口

c) 写使能信号 wen

i. 并不是所有指令都会修改通用寄存器，如跳转指令

ii. 寄存器堆需要提供写使能信号，有效才写

d) 0 号寄存器（$zero）

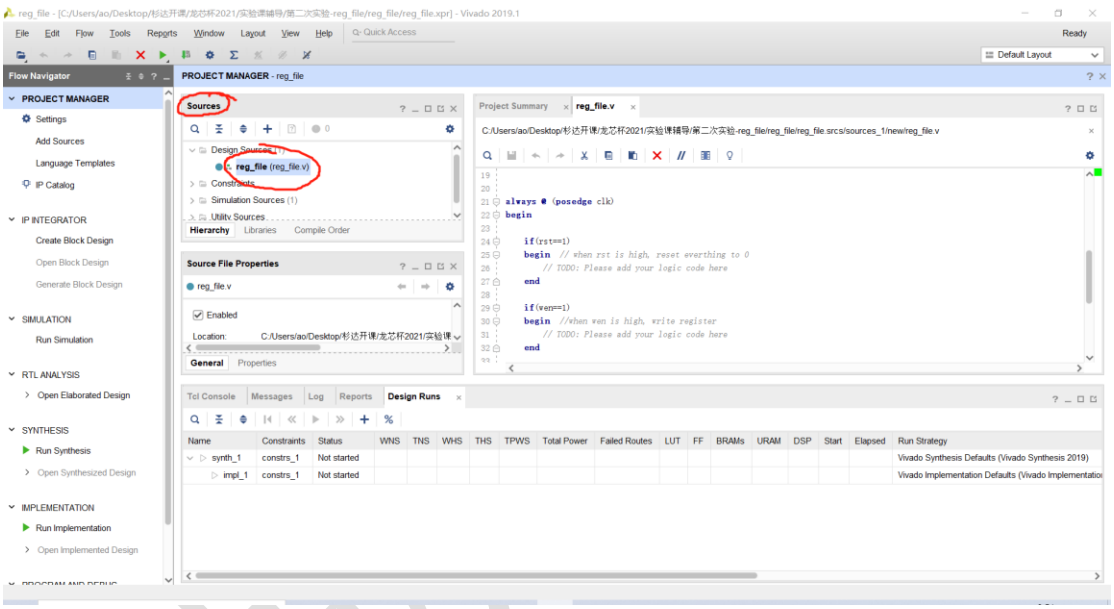
i. 从 0 号地址读出的值总是常量 32’b0

Hint：在实现时，若 waddr==0，则即使 wen 信号为高，也不能将数据写入 0 号寄存器

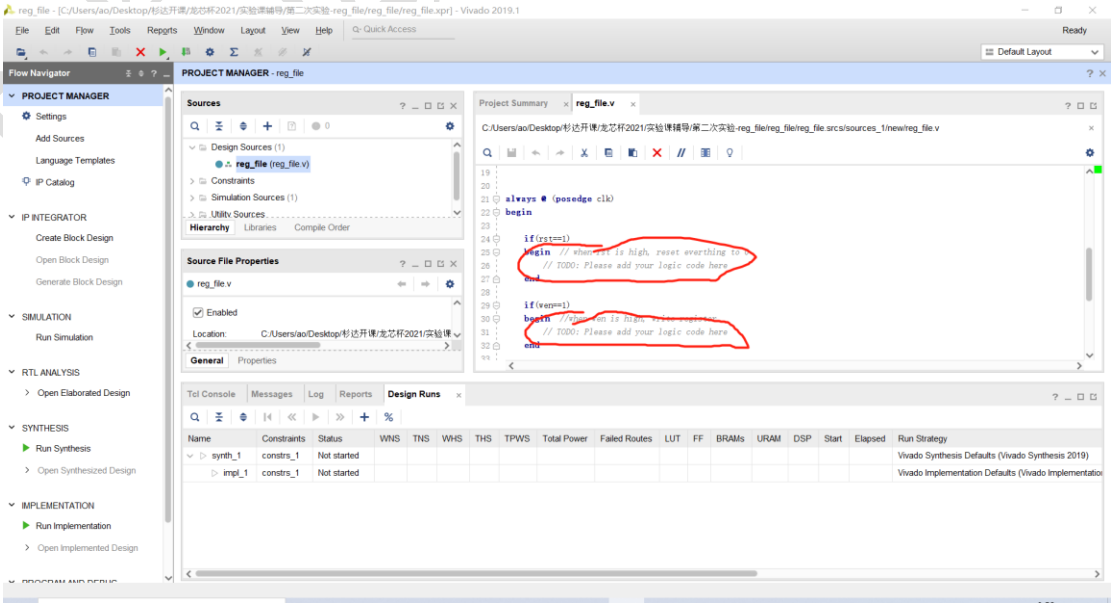
6. 为后续完整 MIPS CPU 设计实验准备好可用组件

实验步骤

1. 下载实验项目包“reg_file”，解压后用 Vivado 软件打开“reg_file.xpr”进入项目。
2. 打开项目后点击 Sources 并找到其中的源文件

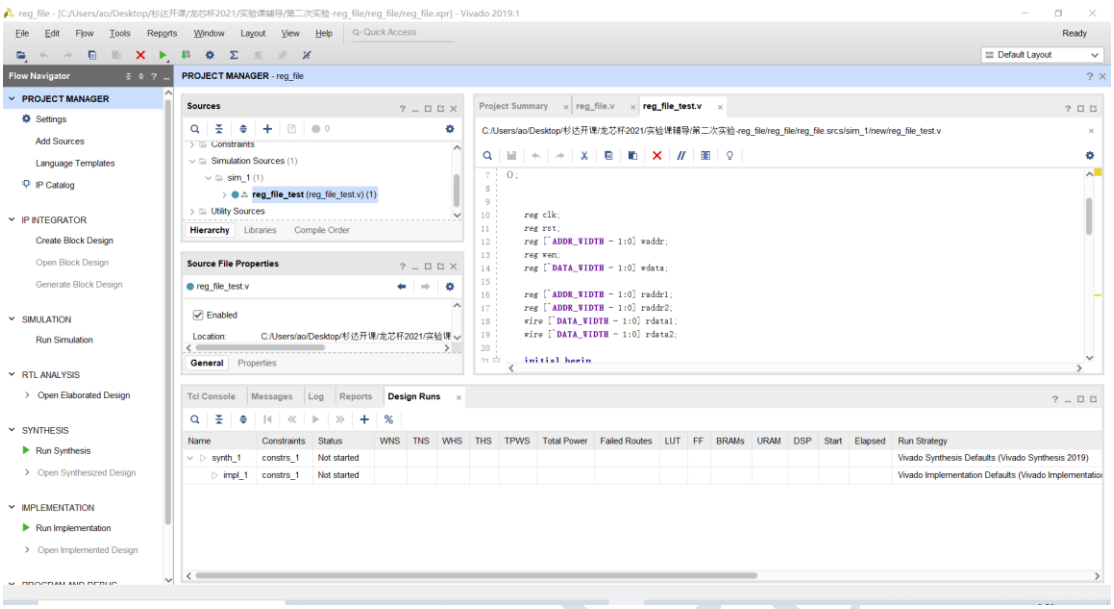


3. 找到代码中标识的“TODO”部分，并将其补充完整



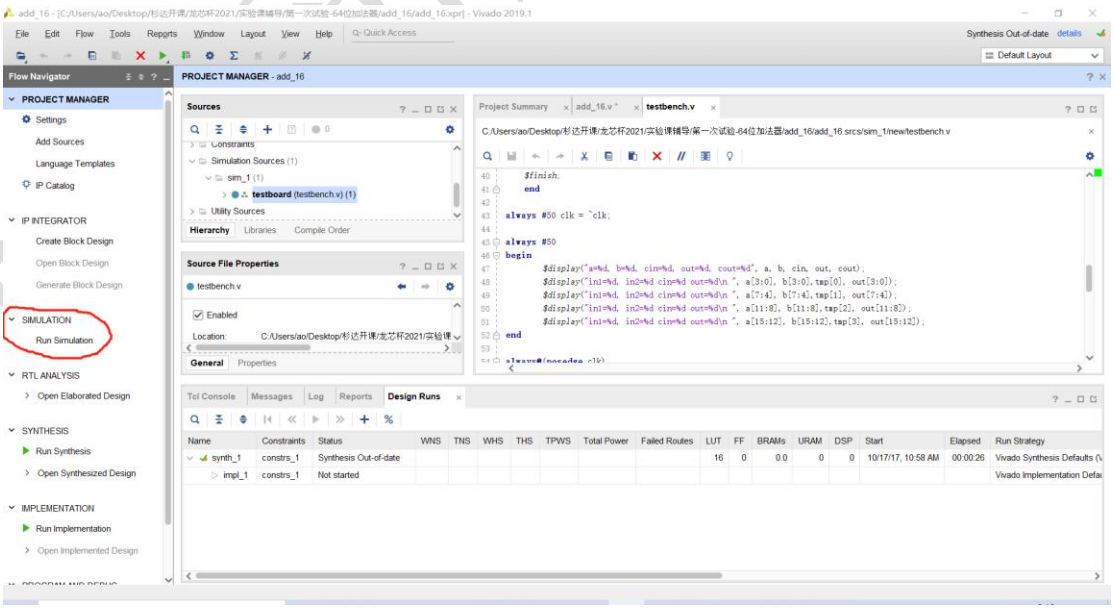
Hint：TODO 上方的注释标明了这段代码的用法。

4. 点击 Simulation Sources，编写 testbench 文件

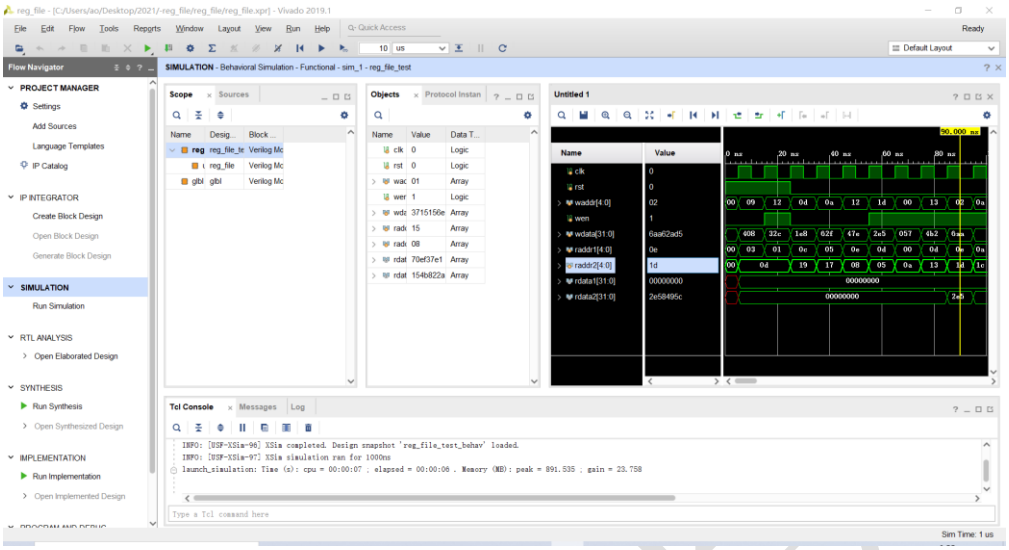


注：reg_file 的测试文件已经写好，无需修改可以直接进行仿真。

5. 点击 Run Simulation 进行仿真



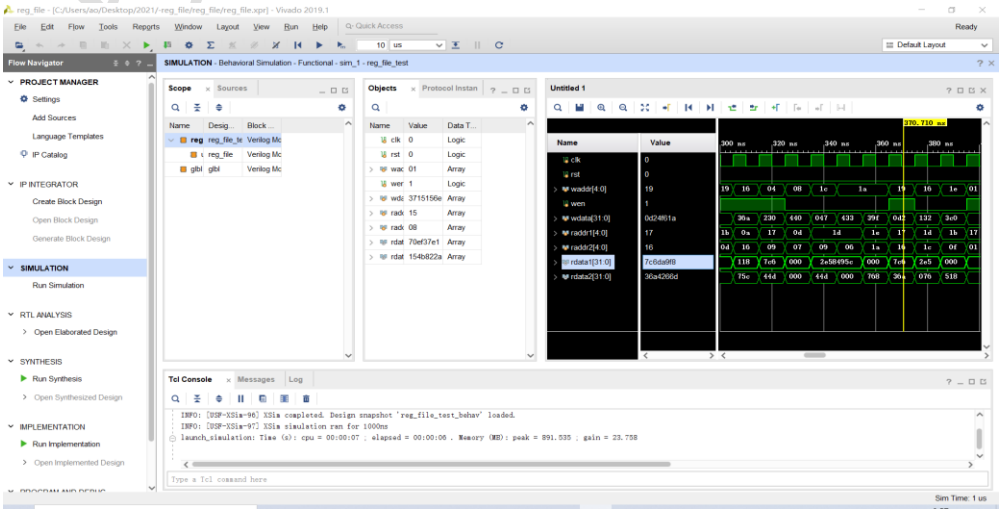
6. 对仿真波形进行检查，并根据其中的问题对代码进行修改，



Hint 1：点击波形上方的“+”和“-”可以对波形进行放大和缩小，一开始看不到波形可能是因为波形过大，多点几下“-”即可。

Hint 2：波形绿色为正常，若出现红色、黄色、蓝色等颜色的波形，则说明代码有问题，需要返回修改。其中红色代表未赋值，蓝色代表位宽错误。

Hint 4：在开始阶段 rdata1, rdata2 会出现一小段红色，那是在初始化之前会出现的未赋值，属于正常情况，忽略即可。



Hint5：开始阶段，由于是随机写寄存器，因此 rdata 读到的值大概率是 0，没有关系，随着时间推移，大部分寄存器都被写过之后，就能读出正常的值了。

Hint6：本次实验的正确性检验较为复杂，首先随机找一个读到的数据 rdata 以及对应 raddr，然后根据 raddr 找到上一次 wen 写使能为高且写这个寄存器的 waddr 匹配时，写入的数据 wdata 与读到的 radata 是否吻合。相同则说明结果正确。

Hint7：注意 r[0] 寄存器的值永远为 0，注意检查一下 raddr 为 0 时读到的值是否都为 0。

提交方式

请同学们与本周日（6 月 14 日）24:00 前将工程文件项目打包好，命名为“姓名_reg_file.zip”的格式，并附上实验报告，发送至邮箱 aoyunbiao@purlytech.com

尤其是本次实验的仿真验证较为复杂，仿真结果的截图至少要包括波形中的以下部分：

1. 写使能信号 wen 为高时对同一个寄存器的读写
2. 写使能信号 wen 为低时对同一个寄存器的读写
3. 对 0 号寄存器的读写