计算机系统结构试验

Lab04: 简单的类 MIPS 单周期处理器功能部件的设计与实现（二）

姓名：N/A

摘要

在 Lab04 中，我进行了有关 register、data memory 和有符号数位扩展的实验，包括寄存器和内存的设计实现、解码指令、读写寄存器等，根据输入指令类型和操作，设置适当的输出。通过本次实验，我进一步加深了对 Verilog 语言的理解和运用，掌握 MIPS 处理器的 Register File 组成部分的设计和实现方法，并且能够使用仿真工具进行验证和调试，给我带来宝贵的经验和收获。

目录

1. 实验目的

（1）理解寄存器、数据存储器、有符号扩展单元的 IO 定义；
（2）Registers 的设计实现；
（3）Data Memory 的设计实现；
（4）有符号扩展部件的实现；
（5）对功能模块进行仿真。

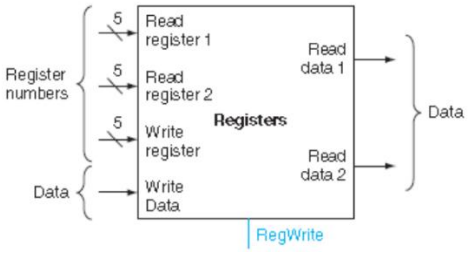
2. 原理分析

2.1 Vivado 工程的基本组成

（1）Registers.v 文件
（2）dataMemory.v 文件
（3）signext.v 文件
（4）Registers_tb.v 激励文件
（5）dataMemory _tb.v 激励文件
（6）signext _tb.v 激励文件

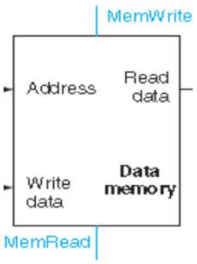
2.2 Registers 模块的原理

寄存器是指令操作的主要对象，MIPS 中一共有 32 个 32 位的寄存器，用作数据的缓存。寄存器模块结构如下：



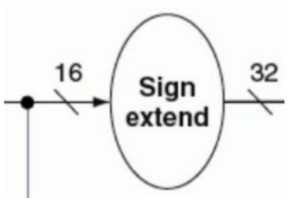
2.3 dataMemory 模块的原理

Data memory 是用来存储运行完成的数据，或者初始化的数据。内存模块的编写与 register 类似，由于写数据也要考虑信号同步，因此也需要时钟。Memory 模块的结构如下：



2.4 signext 模块的原理

Signext 模块将 16 位有符号数扩展为 32 位有符号数。模块结构如下：



3. 功能实现

3.1 Registers 模块的实现

由于不确定 WriteReg，WriteData，RegWrite 信号的先后次序，采用时钟的下降沿作为写操作的同步信号，防止发生错误。因此模块逻辑为：当 readReg1，readReg2，writeReg 其中一个发生变化时，将寄存器内值读到输出；当遇到 Clk 下降沿时，若 regWrite 使能，则将输入写入寄存器。代码如下：

```
reg [31:0] regFile[31:0];

always @(readReg1, readReg2, writeReg) begin
  readData1 = regFile[readReg1];
  readData2 = regFile[readReg2];
end

always @(negedge Clk) begin
  if (regWrite) begin
    regFile[writeReg] = writeData;
  end
end
```

3.2 dataMemory 模块的实现

dataMemory 的实现与 register 基本一致，但只有一个地址输入和一个数据输出。为了防止对同一个地址的读写竞争，设置为当 memRead 和 memWrite 同时使能时，memWrite 被禁用。代码如下：

```
reg [31:0] memory[0:63];

always @(*) begin
  if (memRead) begin
    readData = memory[address];
  end else begin
    readData = 0;
  end
end

always @(negedge Clk) begin
  if (memWrite & ~memRead) begin
    memory[address] = writeData;
  end
end
```

3.3 signext 模块的实现

Signext 模块将 16 位有符号数扩展为 32 位有符号数。因此当输入数值为正数（第 15 位为 0）时位扩展 16 位 0，当输入数值为负数（第 15 位为 1）时位扩展 16 位 1。代码如下：

```
always @(*) begin
  if (inst[15] == 1) begin
    data = {16'b1111111111111111, inst};
  end else begin
    data = {16'b0000000000000000, inst};
  end
end
```

4. 结果验证

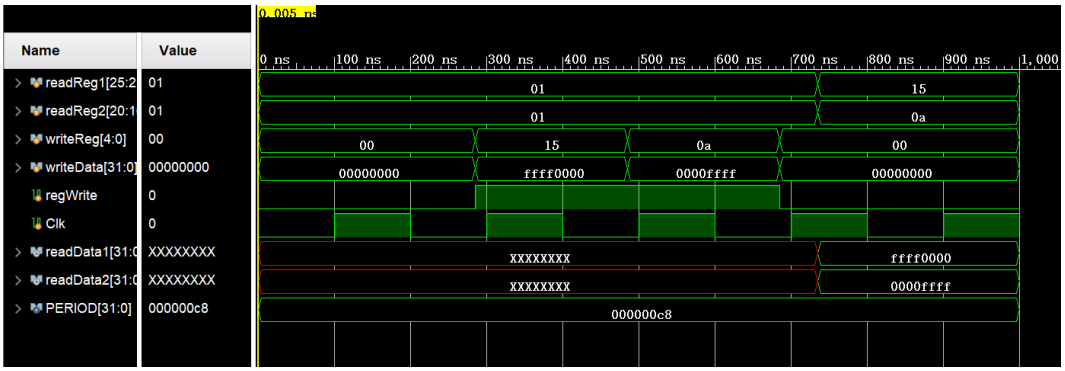
4.1 Registers 模块的测试

编写激励文件设置各输入初值，代码如下：

```
parameter PERIOD = 200;
always #(PERIOD / 2) Clk = !Clk;

initial begin
  Clk = 1'b0;
  readReg1 = 5'd0;
  readReg2 = 5'd0;
  writeReg = 5'd0;
  writeData = 32'd0;
  regWrite = 1'b0;
  #285;  // current time: 285
  writeReg  = 5'd21;
  writeData = 32'd4294901760;
  regWrite  = 1'b1;
  #200;  // current time: 485
  writeReg  = 5'd10;
  writeData = 32'd65535;
  #200;  // current time: 685
  writeReg  = 5'd0;
  writeData = 32'd0;
  regWrite  = 1'b0;
  #50;  // current time: 735
  readReg1 = 5'd21;
  readReg2 = 5'd10;
end
```

测试结果如图所示：



因为初始状态下模块内寄存器未被初始化，所以 Registers 模块在开始时输出为未定值；在写入 reg15 和 reg0a 后，读取值与写入值一致。

4.2 dataMemory 模块的测试

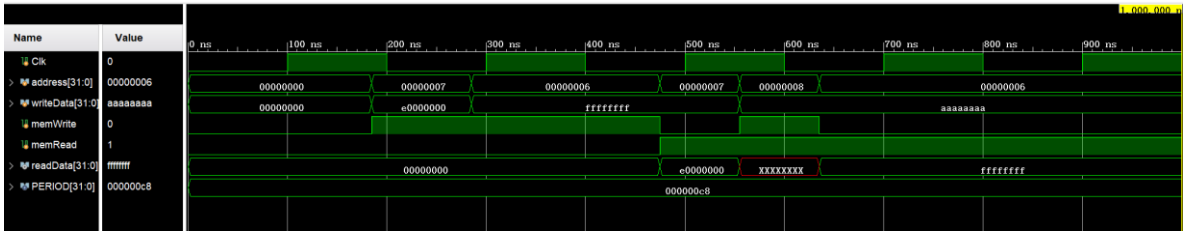
编写激励文件设置各输入初值，代码如下：

```
parameter PERIOD = 200;
always #(PERIOD / 2) Clk = !Clk;

initial begin
  Clk = 1'b0;
  address = 32'h0;
  writeData = 32'h0;
  memWrite = 1'b0;
  memRead = 1'b0;
  #185;  // current time: 185
  address   = 32'h7;
  writeData = 32'he0000000;
  memWrite  = 1'b1;
  #100;  // current time: 285
  address   = 32'h6;
  writeData = 32'hffffffff;
  #190;  // current time: 475
  address   = 32'h7;
  memWrite  = 1'b0;
  memRead   = 1'b1;
  #80;  // current time: 555
  address = 32'h8;
  writeData = 32'haaaaaaaa;
  memWrite = 1'b1;
  #80;  // current time: 635
  address = 32'h6;
  memWrite = 1'b0;

end
```

测试结果如图所示：



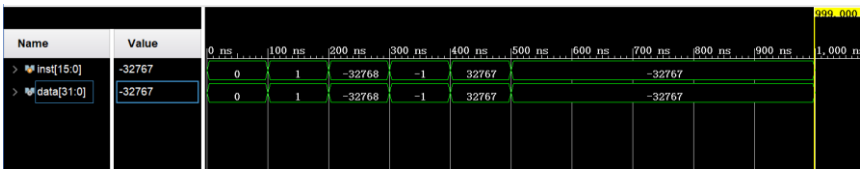
475ns-555ns 时，读取到了 mem[7]在 185ns-285ns 时写入的值 0xe0000000；475ns-555ns 时，发生读写竞争，写信号被禁用；635ns-1000ns 时，读取到了 mem[6]在 285ns-475ns 时写入的值 0xffffffff。

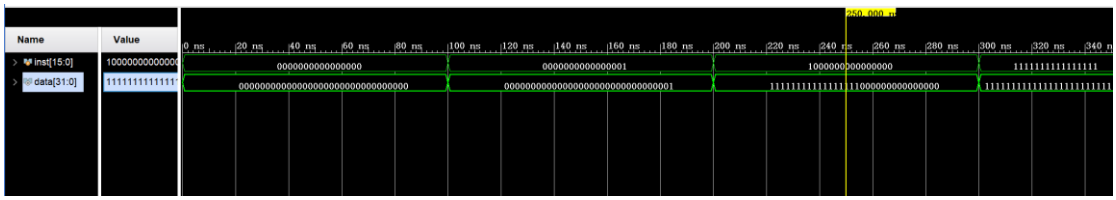
4.3 signext 模块的测试

编写激励文件设置各输入初值，代码如下：

```
initial begin
  inst = 16'h0000;
  #100;
  inst = 16'h0001;
  #100;
  inst = 16'h8000;
  #100;
  inst = 16'hffff;
  #100;
  inst = 16'h7fff;
  #100;
  inst = 16'h8001;
  #100;
end
```

测试结果如图所示：





输出与预期一致。

5. 总结与反思

在 Lab04 中，在本次实验中，我主要关注了 register、data memory 和符号数位扩展部件的设计和实现。通过使用 Vivado 开发环境，我能够更好地理解 Verilog HDL 的基本语法和编程技巧。通过这次实验，我掌握了使用 if-else 块编写分支逻辑的方法，并学习了存储部件的内部结构实现。

我要感谢课程组为我们提供的详细指导书，它为我提供了清晰的实验步骤，使我能够更好地理解和实践所学的知识。通过这次实验，我不仅巩固了 Verilog 的基础知识，还为的学习和设计打下了坚实的基础。