计算机系统结构实验指导书-LAB4

1. OVERVIEW

1.1 实验名称

简单的类 MIPS 单周期处理器存储部件的设计与实现（二）

1.2 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

1.3 实验报告与验收办法

需提交电子版实验报告和工程文件。本实验线上自查三个仿真结果并截图

1.4 注意事项

请按照实验指导书给定的单元部件图、控制线路、数据通路部件图的规范来完成实验

1. 5 实验预计时间

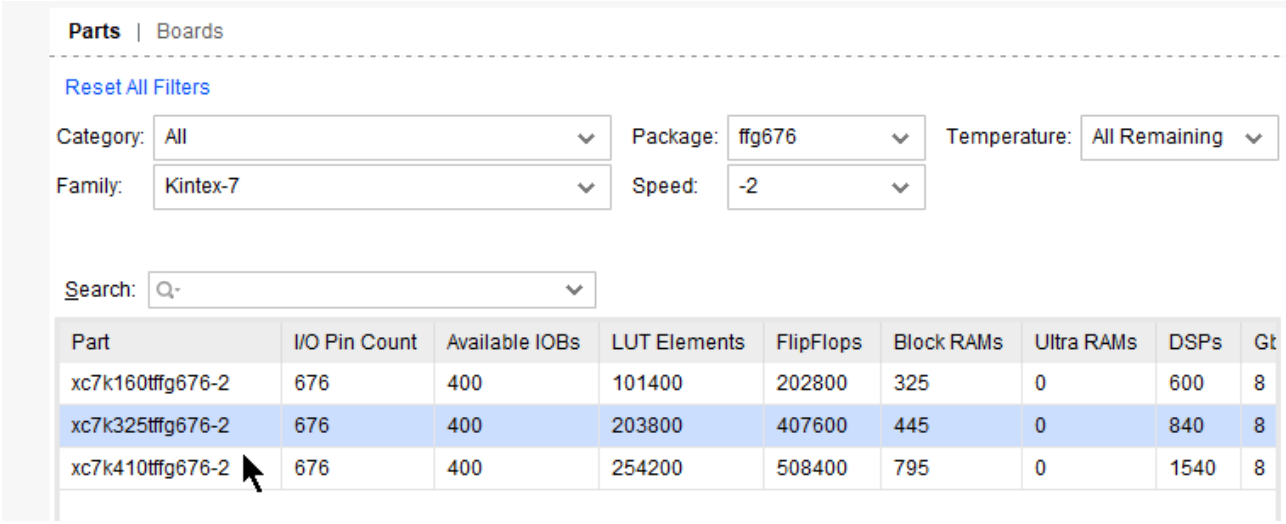
150 分钟

2. 新建工程

2.1 实验描述

2.1.1 新建工程

1. 可新建工程 lab04
2. 选择 FPGA 参数: xc7k325tffg676-2



3. 点击 Next
4. 点击 Finish

3. 寄存器组模块

3.1 实验描述

3.1.1 模块描述

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。

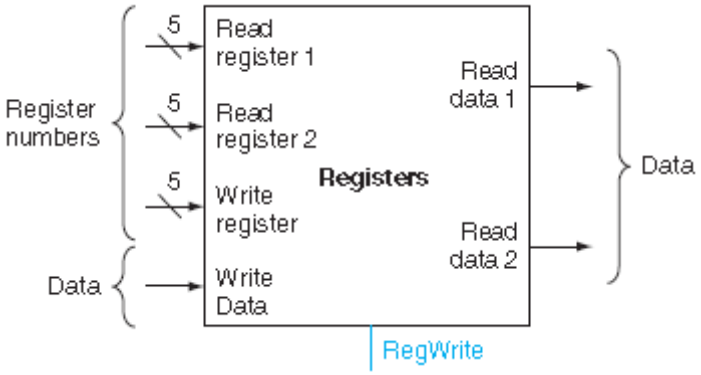
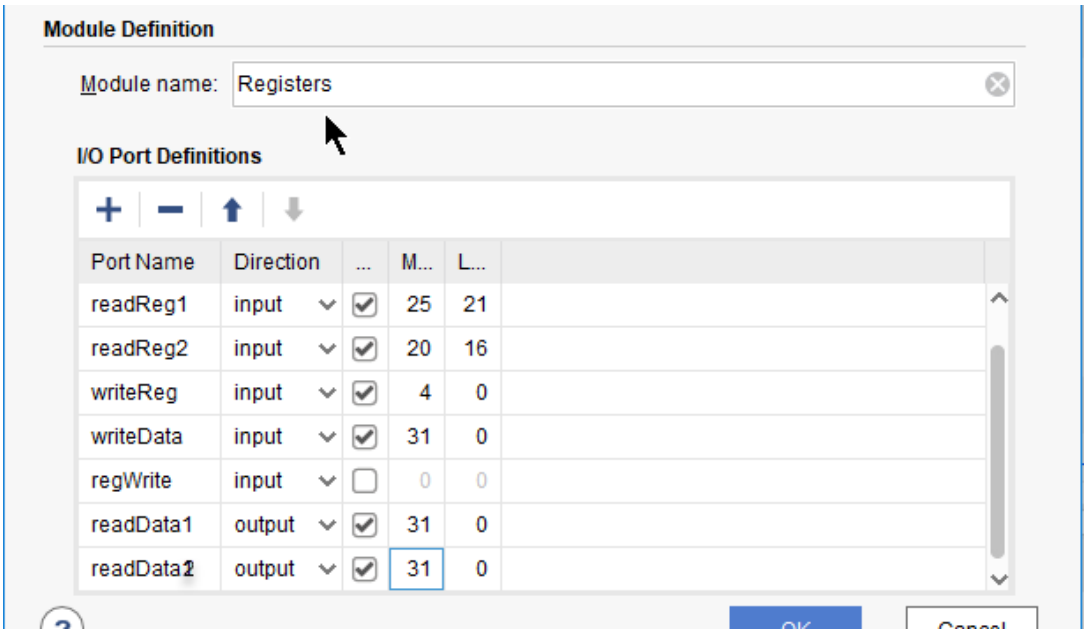


图 1. 寄存器模块的 IO 定义

3.1.2 新建模块源文件

新建文件 Registers。I/O 端口定义，这里加入了时钟信号 Clk



3.1.3 编写功能

由于不确定 WriteReg, WriteData, RegWrite 信号的先后次序，我们采用时钟（样例里命名为 Clk）的下降沿作为写操作的同步信号，防止发生错误。

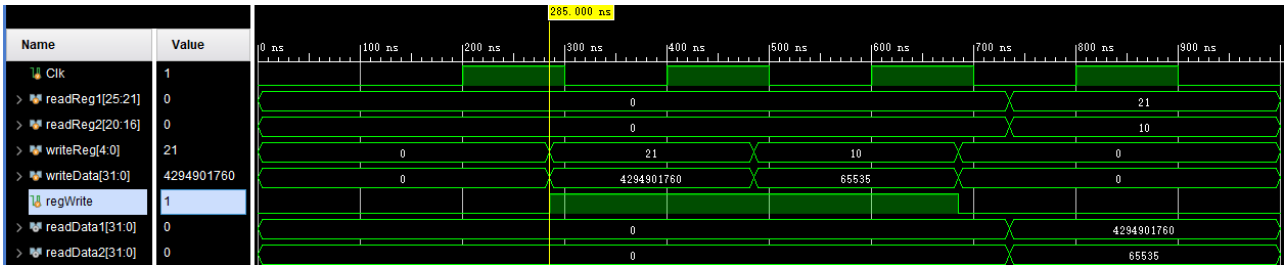
```
……
);

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


always @(readReg1 or readReg2 or writeReg)
    begin
        // ToDo
    end

always @ (negedge Clk)
    begin
        // ToDo
    end

```

3.1.4 仿真测试

1. 测试文件名可取 Registers_tb
2. 添加如下激励信号。使用 Clk 作为时钟输入，仿真周期自定，时钟周期可设为 200ns。

```
    initial begin
        // Initialize Inputs
        //......

        //Current Time: 285ns
        #285;
        regWrite = 1'b1;
        writeReg     = 5'b10101;
        writeData = 32'b11111111111111111000000000000000;

        //Current Time: 485ns
        #200;
        writeReg = 5'b01010;
        writeData = 32'b00000000000000001111111111111111;

        #200;
        regWrite = 1'b0;
        writeReg = 5'b00000;
        writeData = 32'b00000000000000000000000000000000;

        //Current Time: 735ns
        #50;
        readReg1 = 5'b10101;
        readReg2 = 5'b01010;

    end
```

3. 下面给出一个简单仿真样例：



观察波形是否符合预期，如果有错，检查代码，重新仿真

4. 数据存储器模块

4.1 实验描述

4.1.1 模块描述

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。

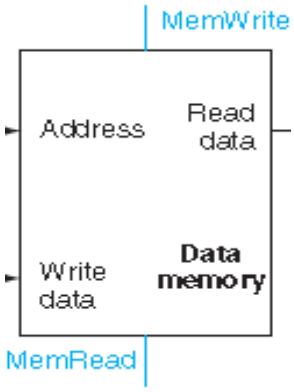
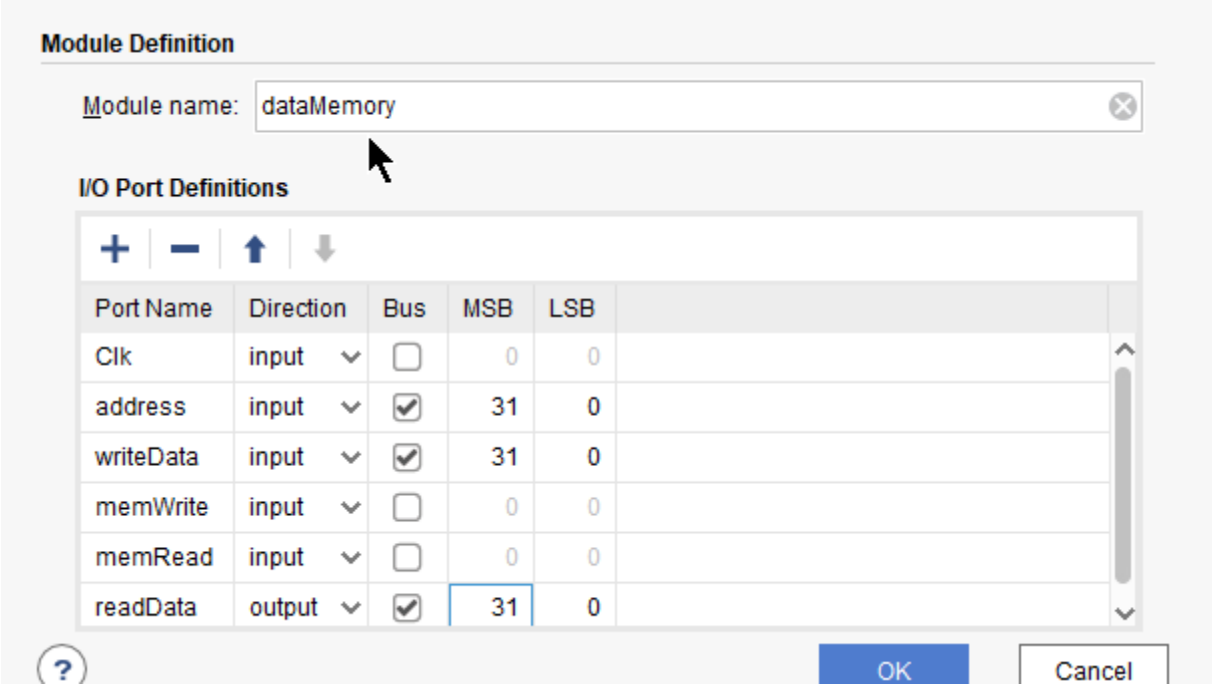


图 2. 内存模块的 IO 定义

4.1.2 新建模块源文件



4.1.3 编写功能

端口声明好了，可如下编写两个 Verilog 代码的 always 语句块：

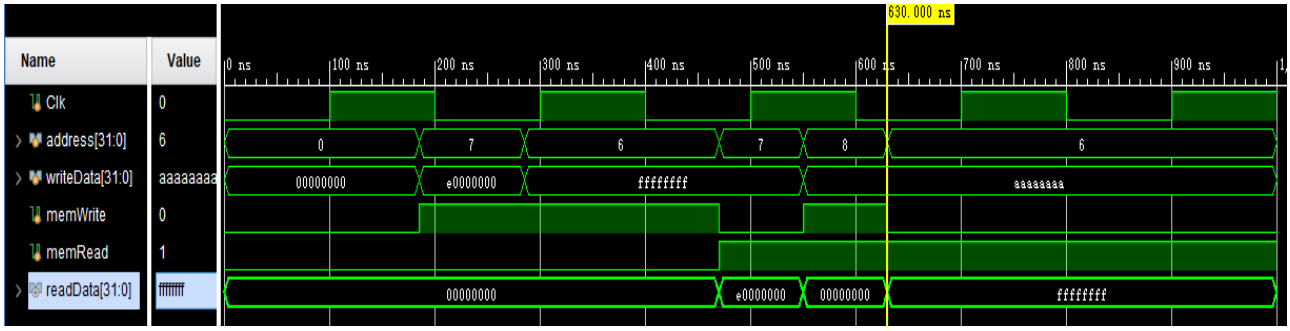
```
13
14      reg [31:0] memFile[0:63];
15
16      always@ ( /* conditions */ )
17        begin
18          // ToDo
19        end
20
21      always@ (/* which edg */)
22        begin
23          // ToDo
24        end
```

4.1.4 功能仿真

1. 文件名可取 dataMemory_tb
2. 添加激励信号如下，设定不同的输入，覆盖所有的情形以保证逻辑的正确

```
20
21          //------------Current Time: 185ns
22          #185;
23          memWrite = 1'b1;
24          address =  32'b00000000000000000000000000000111;
25          writeData = 32'b11100000000000000000000000000000;
26
27          #100;
28          memWrite = 1'b1;
29          writeData = 32'hffffffff;
30          address = 32'b00000000000000000000000000000110;
31
32          #185;
33          memRead = 1'b1;
34          memWrite = 1'b0;
35          //----------
36
37          #80;
38          memWrite = 1;
39          address  = 8;
40          writeData = 32'haaaaaaaa;
41
42          #80;
43          memWrite = 0;
44          memRead = 1;
45          //----------
```

3. 下面给出参考样例：



观察波形是否满足逻辑，如果有误，检查代码，重新仿真

5. 带符号扩展模块

5.1 实验描述

5.1.1 模块描述

将 16 位有符号数扩展为 32 位有符号数。

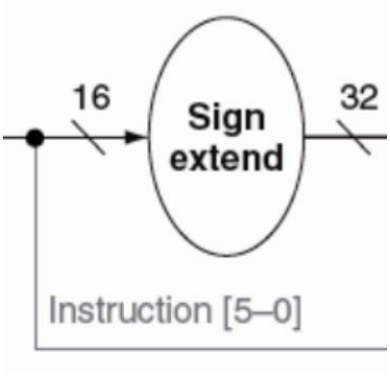


图 3. 带符合扩展单元

补码：

（1）正数的补码：与原码相同。

+9 的补码是 00001001。

（2）负数的补码：符号位为 1，其余位为该数绝对值的原码按位取反；然后整个数加 1。

求-7 的补码。

因为给定数是负数，则符号位为“1”。

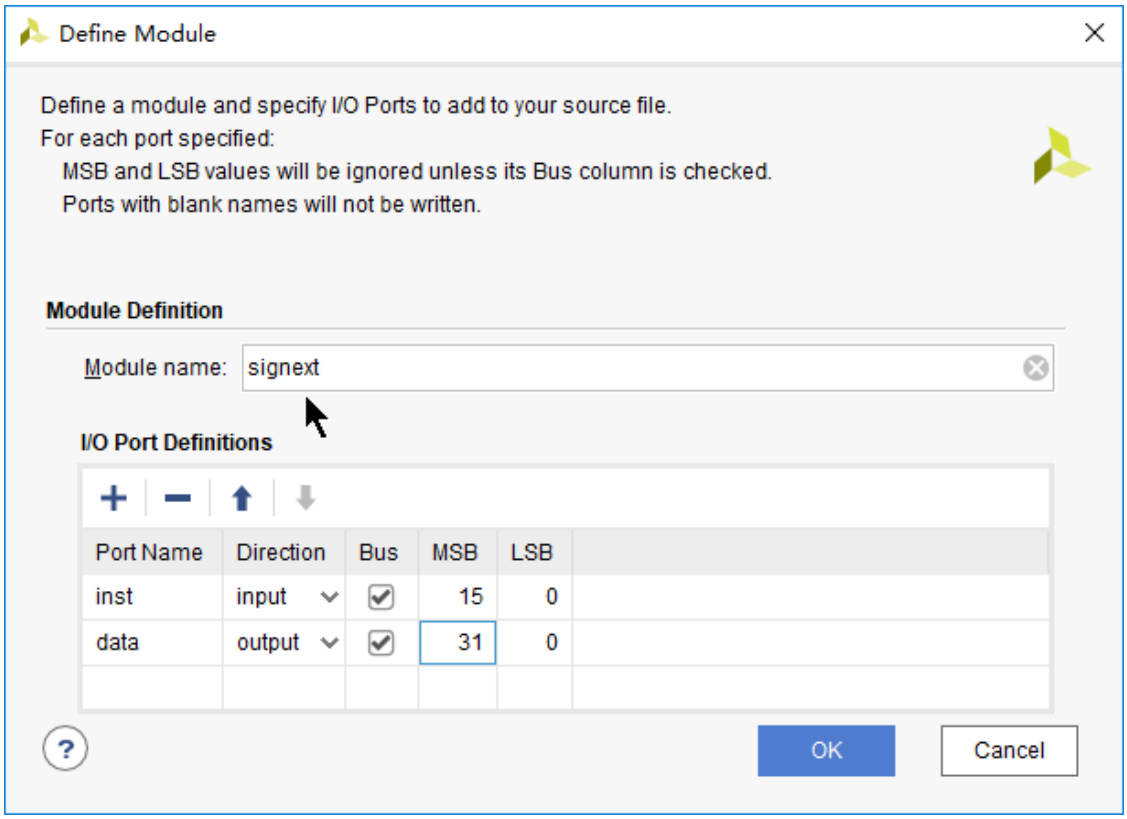
后七位：+7 的原码（0000111）→按位取反（1111000）→加 1（1111001）

所以-7 的补码是 11111001。

注意：带符号扩展只需要在前面补足符号即可

5.1.2 新建模块源文件

文件名可取 signext



5.1.3 实现功能

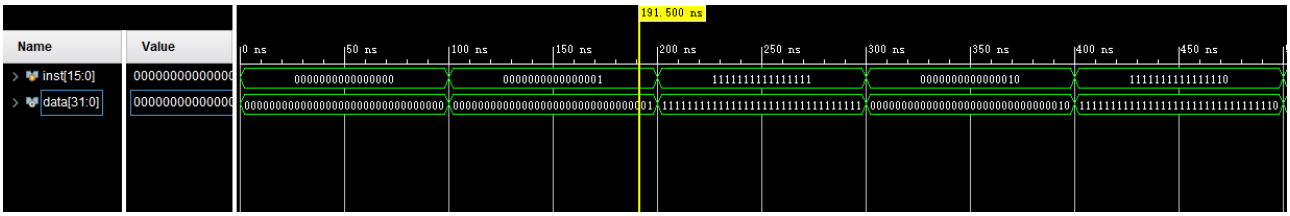
有多种方法将符号补齐

```
module signext(
    input [15:0] inst,
    output [31:0] data
    );

      assign data=  //How to;

endmodule
```

5.1.4 仿真测试

1. 添加激励信号
2. 观察波形是否满足设计逻辑

3. 参考波形如下：



PS：至少需要一个正数和一个负数的观察