实验 7-寄存器堆与 ALU 设计实验

一、 实验准备

1.1 装有 Vivado2015.4 的电脑一台：本实验不对 Vivado 环境有硬性要求，但涉及 Xilinx 库中 IP 的实验，在不同版本环境下无法兼容，且低版本 Vivado 无法运行高版本生成的项目，为方便实验检查，应当尽量使用实验要求版本。
1.2 熟悉 Vivado 的 IDE 环境，并能够使用其进行仿真、综合；（见第一次实验课件）
1.3 熟悉开发板；（见第一次实验课件）

二、 实验目的

2.1 了解算术逻辑单元 ALU 的原理；
2.2 熟悉并运用 Verilog 语言设计 ALU；
2.3 学习寄存器堆的数据传送与读写工作原理，掌握寄存器读写的设计方法；
2.4 熟悉并运用 Verilog 语言设计 ALU；
2.5 学习 Verilog 不同形式的编程方式，理解 assign 和 always 的区别；

三、 实验任务

2.1 ALU 设计实验
下图给出了一个具有 N 位输入和 N 位输出的算数逻辑单元的电路符号。算术逻辑单元接收说明执行哪个功能的控制信号 F，执行对应功能后输出 N 位结果。

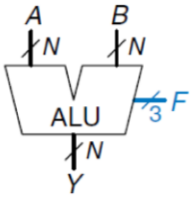


图 1

实验要求实现以下算术运算功能，其对应的指令码及功能如下：

表 1

$F_{2:0}$	功能	$F_{2:0}$	功能
000	A + B(Unsigned)	100	$\bar{A}$
001	A - B	***101***	***SLT***
010	A AND B	110	未使用
011	A OR B	111	未使用

本次实验将 ALU 输出结果通过板载七段数码管进行显示验证，原理图如下图所示：

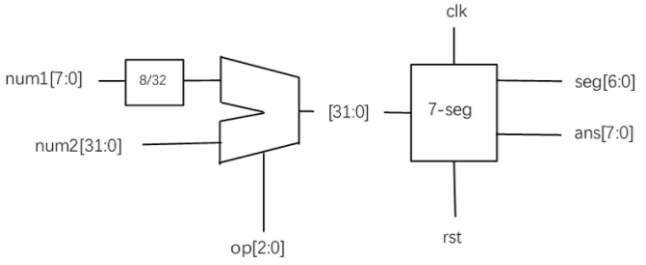


图 2

2.1.1 实验要求

1. 根据 ALU 原理图(图 1)，使用 Verilog 语言定义 ALU 模块，其中输入输出端口参考实验原理，运算指令码长度为[2:0]。
2. 内置一个 32 位 num2（值为 32h'01）作为输入到运算器端口 A;
3. 将 sw3~sw10 输入 num1，经过符号扩展至 32 位后，输入到运算器的端口 B;
4. 运算器支持"加、减、与、或"4 种运算，需要 3 位（8 个操作）。将 sw0~sw2 输入到 op 作为运算器的控制信号;
5.实现 SLT 功能。
6.将计算 32 位结果 s 显示到七段数码管(16 进制)。
7.验证表 1 中所有功能。
8.给出 RTL 源程序（.v 文件）

2.1.2 实验环境

实现 alu.v 文件（本次实验重点），seg.v (数码管显示以及拨码开关模块，在前面实验中已经完成，代码也已经提供)

2.2 寄存器堆设计实验

下图红色框里给出了一个寄存器堆电路符号，寄存器堆存有 32 个 registers，能够通过 3 个 5 位地址总线 Rw, Ra, Rb 寻址到寄存器，RW 是写信号，RW=1 时，将 busW 的数据存储到 Rw 的地址所对应的 register 里面。RW=0 时，根据地址 Ra, Rb 地址读取寄存器的值，放到 busA 和 busB 上。

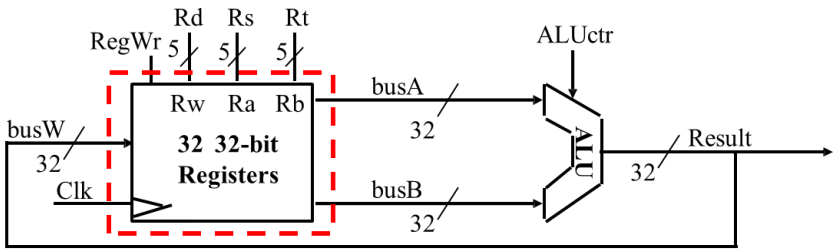


图 3

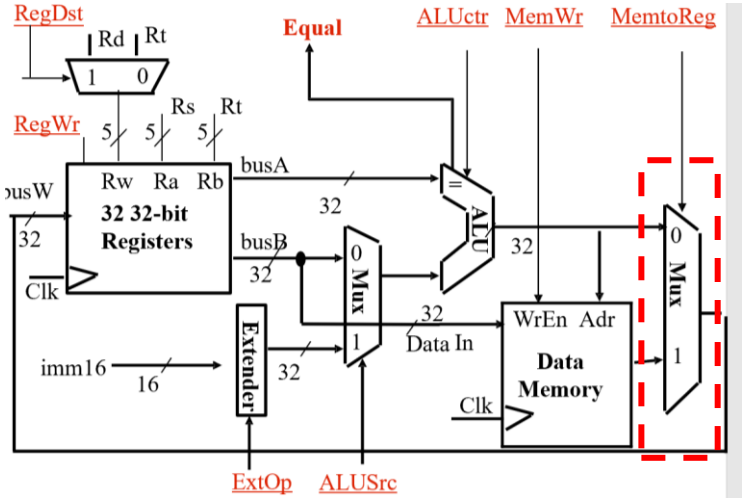
2.2.1 实验要求

1. 根据寄存器堆（register file）原理图(图 3)，使用 Verilog 语言定义 register file 模块，其实现方式可以参照实验给出的参照设计;
2. 将所涉及的 ALU 与 register file 模块连起来，形成图 3 所示的系统;
3. 图 4 在图 3 的基础上加入了多路选择器的部件，用来选择寄存器的写入数据是根据 ALU 还是根据存储器来进行写入。根据图 4 中红色框框里面的原理，完成一次模拟存储器写入，通过 MemtoReg 控制信号来确定 busW 信号是来之存储器的常数还是 ALU 输出的结果，通过下列语句可以实现:

busW=(MemtoReg==1)? 32'b1: result;

其中假设存储器恒定输出 32'b1（也可以固定成其他值），result 是 ALU 的输出；往$t1, $t2 里面写入 32'b1（也可以固定成其他值），通过仿真验证写入过程的正确性;
4. 在$t1, $t2 初始值的基础上，模拟指令 add $t0, $t1, $t2 的过程，通过仿真验证 add 计算过程的正确性。

5.给出 RTL 源程序（.v 文件）



2.2.2 实验环境

register file 实验给出了逻辑实现方式：

```
module regfile(
    input          clk,
    // READ PORT 1
    input  [ 4:0] raddr1,
    output [31:0] rdata1,
    // READ PORT 2
    input  [ 4:0] raddr2,
    output [31:0] rdata2,
    // WRITE PORT
    input          we,       //write enable, HIGH valid
    input  [ 4:0] waddr,
    input  [31:0] wdata
);
reg [31:0] rf[31:0];

//WRITE
always @(posedge clk) begin
    if (we) rf[waddr]<= wdata;
end

//READ OUT 1
assign rdata1 = (raddr1==5'b0) ? 32'b0 : rf[raddr1];

//READ OUT 2
assign rdata2 = (raddr2==5'b0) ? 32'b0 : rf[raddr2];

endmodule
```