中国科学院大学计算机组成原理实验课

实 验 报 告

学号：2019K8009915025　姓名：马月骁　专业：计算机科学与技术

实验序号：1　　实验名称：基本功能部件设计

注 1：请在实验项目个人本地仓库中创建顶层目录 doc。撰写此 Word 格式实验报告后以 PDF 格式保存在 doc 目录下。文件命名规则：学号-prjN.pdf，其中学号中的字母“K”为大写，“-”为英文连字符，“prj”和后缀名“pdf”为小写，“N”为 1 至 4 的阿拉伯数字。例如：2019K8009929000-prj1.pdf。PDF 文件大小应控制在 5MB 以内。此外，实验项目 5 包含多个选做内容，每个选做实验应提交各自的实验报告文件，文件命名规则：学号-prj5-projectname.pdf，例如：2019K8009929000-prj5-dma.pdf。具体要求详见实验项目 5 讲义。 注 2：使用 git add 及 git commit 命令将 doc 目录下的实验报告 PDF 文件添加到本地仓库 master 分支，并通过 git push 推送到 GitLab 远程仓库 master 分支（具体命令详见实验报告）。 注 3：实验报告模板下列条目仅供参考，可包含但不限定如下内容。实验报告中无需重复描述讲义中的实验流程。

一、 逻辑电路结构与仿真波形的截图及说明

1. Register File

整体代码如下：

```
`timescale 10 ns / 1 ns

`define DATA_WIDTH 32
`define ADDR_WIDTH 5

module reg_file(
        input clk,
        input rst,
        input [`ADDR_WIDTH - 1:0] waddr,
        input [`ADDR_WIDTH - 1:0] raddr1,
        input [`ADDR_WIDTH - 1:0] raddr2,
        input wen,
        input [`DATA_WIDTH - 1:0] wdata,
        output [`DATA_WIDTH - 1:0] rdata1,
        output [`DATA_WIDTH - 1:0] rdata2
);
```

```
        reg [`DATA_WIDTH - 1:0] r [`DATA_WIDTH - 1:0];

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

        assign rdata1 = {`DATA_WIDTH{|raddr1}} & r[raddr1];
        assign rdata2 = {`DATA_WIDTH{|raddr2}} & r[raddr2];

endmodule
```

整体电路原理图如下：

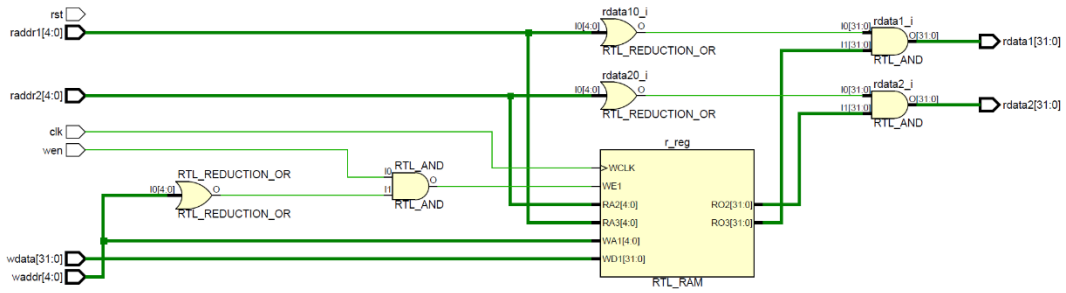

图 1 Register File 电路原理图

Register File 需满足要求：

（1）寄存器组：32 个 32-bit 寄存器；

（2）写使能信号 wen（高电平有效）：当且仅当写使能信号 wen=1 且 waddr 不等于 0 时，才可向以 waddr 为地址的寄存器中写入 wdata；

（3）0 号寄存器（$zero）：不能向 0 号寄存器写入 wdata；从 0 号寄存器中读出的值为常量 32’b0。

要求实现：

（1）寄存器组：

对应代码：

```
reg [`DATA_WIDTH - 1:0] r [`DATA_WIDTH - 1:0];
```

（2）写使能信号 wen（高电平有效）：

对应代码：

```
always @(posedge clk) begin
        if(wen & (|waddr)) begin
                r[waddr] <= wdata;
        end
end
```

时序电路实现，当且仅当写使能信号 wen=1 且 waddr 不等于 0 时，才可向以 waddr 为地址的寄存器中写入 wdata。其中（|waddr=0）当且仅当 waddr=32’b0

对应逻辑电路结构：

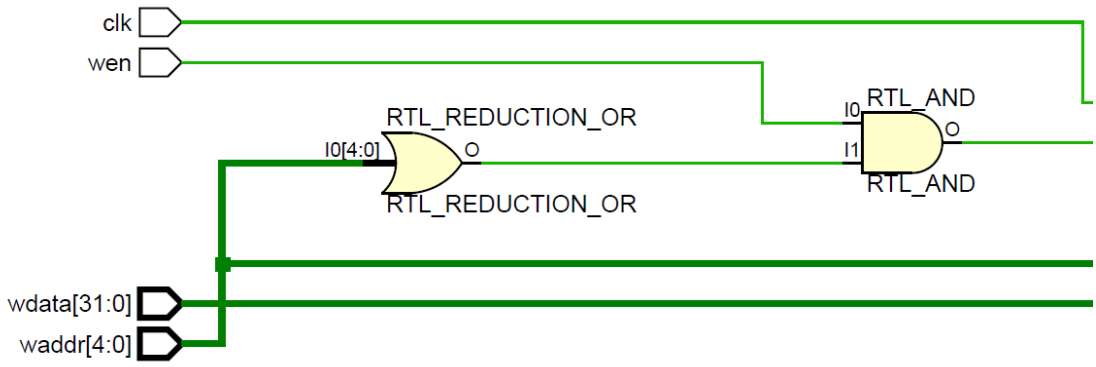

图 2 写使能信号 wen 对应逻辑电路结构

（3）0 号寄存器（$zero）：

对应代码：

```
assign rdata1 = {`DATA_WIDTH{|raddr1}} & r[raddr1];
```

```
assign rdata2 = {`DATA_WIDTH{|raddr2}} & r[raddr2];
```

“不能向 0 号寄存器写入 wdata”已在写使能信号部分实现；

{`DATA_WIDTH{|raddr}}将读信号按位或并算数拓展为 32 位，保证了当 raddr 为 32’b0 时读出数值为 32’b0，非 0 时正常读出寄存器中的值。

对应仿真波形：

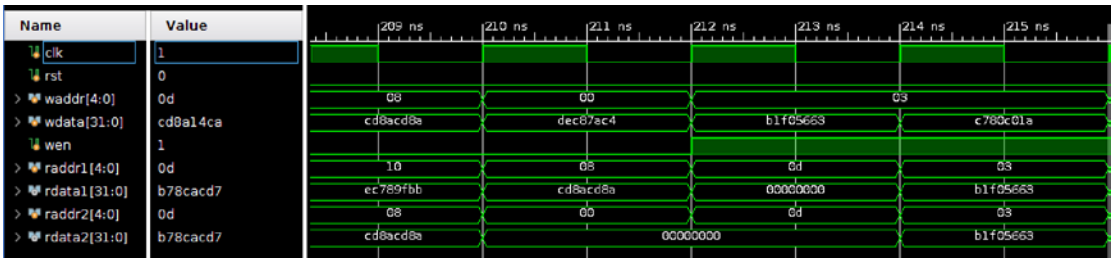

图 3 仿真波形

在 210ns 处，waddr=00，但由于 wen=0 且|waddr=0，故 wdata 不写入寄存器；在 212ns 处 waddr=03，同时 wen=1 且|waddr=1，故 wdata（cd8a14ca）写入 03 寄存器。

在 210ns 处，radd2=00，对应 0 号寄存器，则 rdata2=32’b0；在 214ns 处，raddr=03，正常读出数据 rdata1（cd8a14ca）。

2.ALU

整体代码如下：

```
`timescale 10 ns / 1 ns

`define DATA_WIDTH 32

module alu(
        input [`DATA_WIDTH - 1:0] A,
        input [`DATA_WIDTH - 1:0] B,
        input [2:0] ALUop,
        output Overflow,
        output CarryOut,
```

```
        output Zero,
        output [`DATA_WIDTH - 1:0] Result
);

//creat wire variables for intermediate results
           wire [`DATA_WIDTH - 1:0] R_AND;
           wire [`DATA_WIDTH - 1:0] R_OR;
           wire [`DATA_WIDTH - 1:0] R_AND_OR;
        wire [`DATA_WIDTH - 1:0] R_ADD_SUB;
        wire [`DATA_WIDTH - 1:0] R_SLT;
        wire COUT;
        wire CIN;
        wire Cin;
        wire [`DATA_WIDTH - 1:0] B_ADD_SUB;
        wire Compare;
//ALUop operators
        parameter AND = 3'b000,
                  OR = 3'b001,
                  ADD = 3'b010,
                  SUB = 3'b110,
                  SLT = 3'b111;

//determine CIN and B(inverse or not) according to ALOup(SUB && SLT)
        assign B_ADD_SUB = ALUop[2] ? ~B : B;
        assign CIN = ALUop[2];
//intermediate result of ADD and OR operations
        assign R_AND = A & B;
        assign R_OR = A | B;
//instance of 32 bits carry lookahead adders
        advance_add_32 instance_0(
                .A(A),
                .B(B_ADD_SUB),
                .CIN(CIN),
                .S(R_ADD_SUB),
                .COUT(COUT),
                .Cin(Cin)
        );
//result of intermediate result of SLT opperations
        assign Compare =  ~Overflow & R_ADD_SUB[`DATA_WIDTH - 1] |
Overflow & ~R_ADD_SUB[`DATA_WIDTH - 1];
//ADD or SUB or SLT operations
        assign R_SLT = {31'b0,Compare};

//output
```

```
//final result according to ALUop
        assign Result = ({`DATA_WIDTH{ALUop == AND}} & R_AND) |
                        ({`DATA_WIDTH{ALUop == OR}} & R_OR) |
                        ({`DATA_WIDTH{ALUop == ADD || ALUop == SUB}} &
R_ADD_SUB) |
                        ({`DATA_WIDTH{ALUop == SLT}} & R_SLT);
//signed numbers add & sub operations whether overflow
        assign Overflow = ~Cin & COUT | Cin & ~COUT;
//insigned numbers add & sub operations whether carry or borrow number
from sign bit
        assign CarryOut =  ~ALUop[2] & COUT | ALUop[2] & ~COUT;
//whether the result is 32'b0
        assign Zero = ~|Result;

endmodule

module advance_add_32(
        input [`DATA_WIDTH - 1:0] A,
        input [`DATA_WIDTH - 1:0] B,
        input CIN,
        output Cin,
        output COUT,
        output [`DATA_WIDTH - 1:0] S
);
        wire [`DATA_WIDTH:0] cout;

        assign cout[0] = CIN;

        genvar i;
        generate
            for(i = 0; i < `DATA_WIDTH; i = i + 1)
                begin : CLA
                    wire p, g;
                    assign p = ~A[i] & B[i] | A[i] & ~B[i];
                    assign g = A[i] & B[i];
                    assign S[i] = ~p & cout[i] | p & ~cout[i];
                    assign cout[i + 1] = g | p & cout[i];
                end
        endgenerate

        assign COUT = cout[`DATA_WIDTH];
        assign Cin = cout[`DATA_WIDTH - 1];
endmodule
```

整体电路原理图如下：

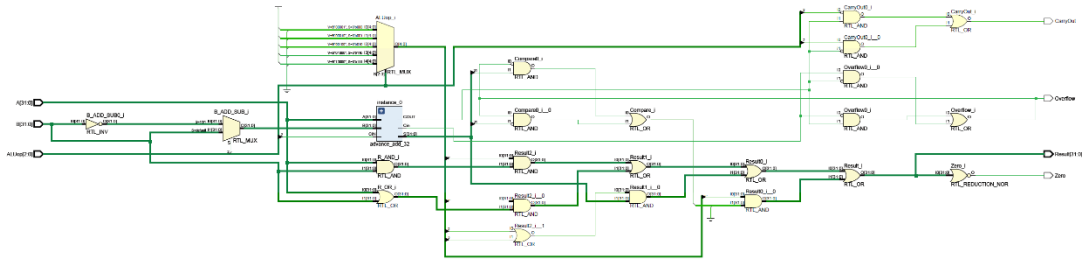

图 4 ALU 整体电路原理图

ALU 需满足要求：

（1） 五种基本功能：

ALUop	功能操作
000	逻辑按位与
001	逻辑按位或
010	算术加法
110	算数减法
111	有符号数整数比较（A<B, Result=1；反之 Result=0）

（2） 标志信号：

信号名	说明
Overflow	溢出标志信号，指示有符号数的加减法运算结果是否溢出，非此操作时信号未定义
CarryOut	进借位标志信号，指示无符号数的加减法运算结果是否产生进位/借位，非此操作时信号未定义
Zero	零标志，指示 ALU 运算结果是否为 0

要求实现：

（1） 五种基本功能：

对应代码：

逻辑按位与（AND）、逻辑按位或（OR）：

```
//intermediate result of ADD and OR operations
        assign R_AND = A & B;
```

```
        assign R_OR = A | B;
```

算数加法（ADD)、算数减法（SUB)：(加法、减法、比较使用同一加法器)

```
//determine CIN and B(inverse or not) according to ALOup(SUB && SLT)
        assign B_ADD_SUB = ALUop[2] ? ~B : B;
        assign CIN = ALUop[2];
//instance of 32 bits carry lookahead adders
        advance_add_32 instance_0(
                .A(A),
                .B(B_ADD_SUB),
                .CIN(CIN),
                .S(R_ADD_SUB),
                .COUT(COUT),
                .Cin(Cin)
        );
```

当为加法时，进位数 CIN 设为 0；当为减法或选择操作时 (ALUop[2]=1)，需要使用减法。此时需将 B 按位取反，进位数 CIN 设为 1。

advance_add_32 instance_0 为单独设计的 32 位 CLA，具体代码实现细节位于上文中 module advance_add_32。其中端口定义如下：

端口名	定义
A、B (input)	被加数与加数
CIN (input)	进位信号
S (output)	和
COUT (output)	加法运算产生的进位
Cin (output)	加法运算中 31 位向 32 位的进位

有符号数整数比较 (SLT)：

```
//result of intermediate result of SLT opperations
        assign Compare =  ~Overflow & R_ADD_SUB[`DATA_WIDTH - 1] |
Overflow & ~R_ADD_SUB[`DATA_WIDTH - 1];
//ADD or SUB or SLT operations
        assign R_SLT = {31'b0,Compare};
```

Compare 信号表示 A、B 大小。当有符号数减法运算无溢出时 (Overflow=0)，结果符号位 R_ADD_SUB[`DATA_WIDTH - 1]=1，则 A<B；反之，当有符号数减法运算结果存在溢出时 (Overflow=1)，结果符号位 R_ADD_SUB[`DATA_WIDTH - 1]=0，则 A<B。

结果选择：

```
//ALUop operators
        parameter AND = 3'b000,
                  OR = 3'b001,
                  ADD = 3'b010,
                  SUB = 3'b110,
                  SLT = 3'b111;
//final result according to ALUop
        assign Result = ({`DATA_WIDTH{ALUop == AND}} & R_AND) |
                        ({`DATA_WIDTH{ALUop == OR}} & R_OR) |
                        ({`DATA_WIDTH{ALUop == ADD || ALUop == SUB}} &
R_ADD_SUB) |
                        ({`DATA_WIDTH{ALUop == SLT}} & R_SLT);
```

对应逻辑电路结构：

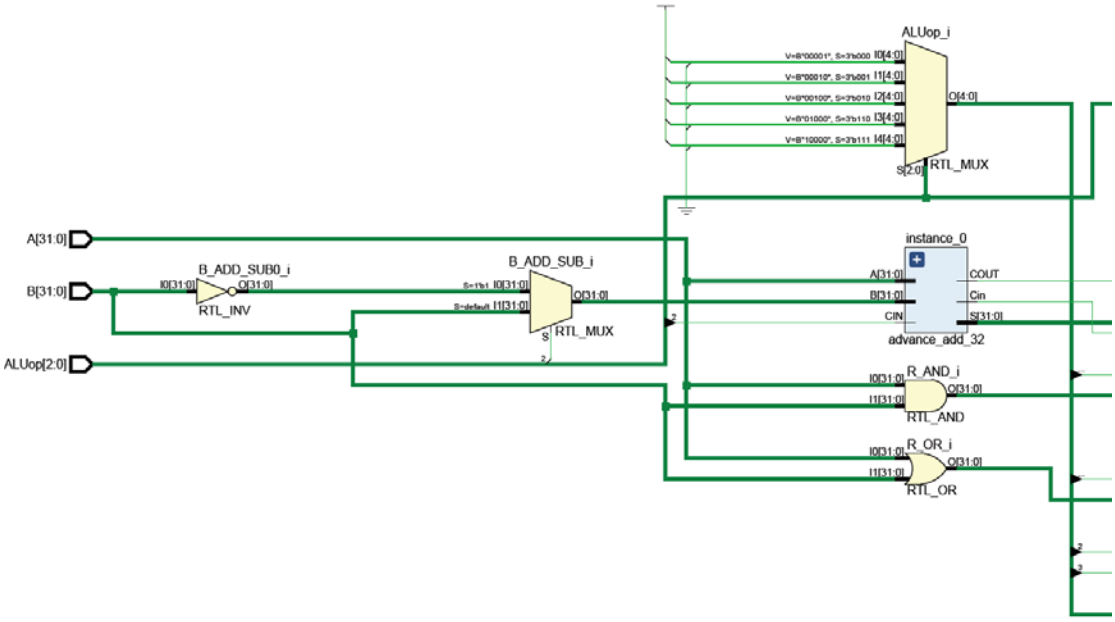

图 5.1 五种基本功能对应逻辑电路图

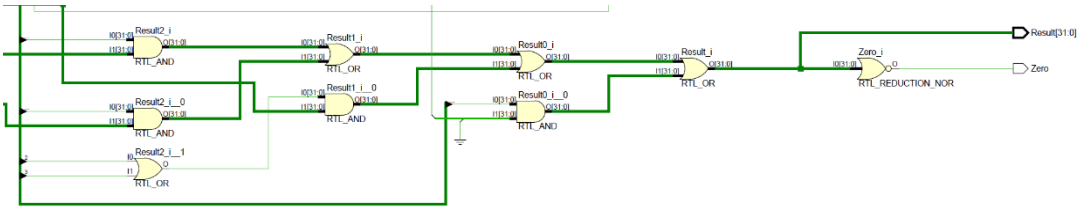

图 5.2 五种基本功能对应逻辑电路图

（2） 标志信号：

对应代码：

```
//signed numbers add & sub operations whether overflow
        assign Overflow = ~Cin & COUT | Cin & ~COUT;
//insigned numbers add & sub operations whether carry or borrow number
from sign bit
        assign CarryOut =  ~ALUop[2] & COUT | ALUop[2] & ~COUT;
//whether the result is 32'b0
        assign Zero = ~|Result;
```

Overflow 信号为加法器中 31 位向 32 位的进位信号与结果进位信号的异或。

当无符号数加法产生进位（~ALUop[2] & COUT），或减法产生借位，即对应补码加法没有产生进位（ALUop[2] & ~COUT）时，CarryOut=1。

当且仅当 Result=32‘b0 时，|Result=0，此时 Zero=1。

对应逻辑电路结构：

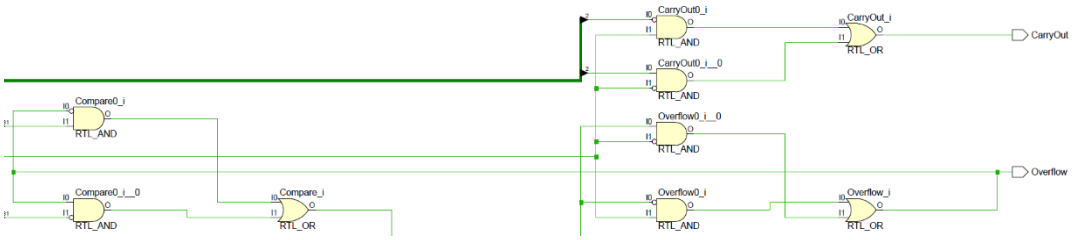

图 6 标志信号对应逻辑电路结构

对应仿真波形:

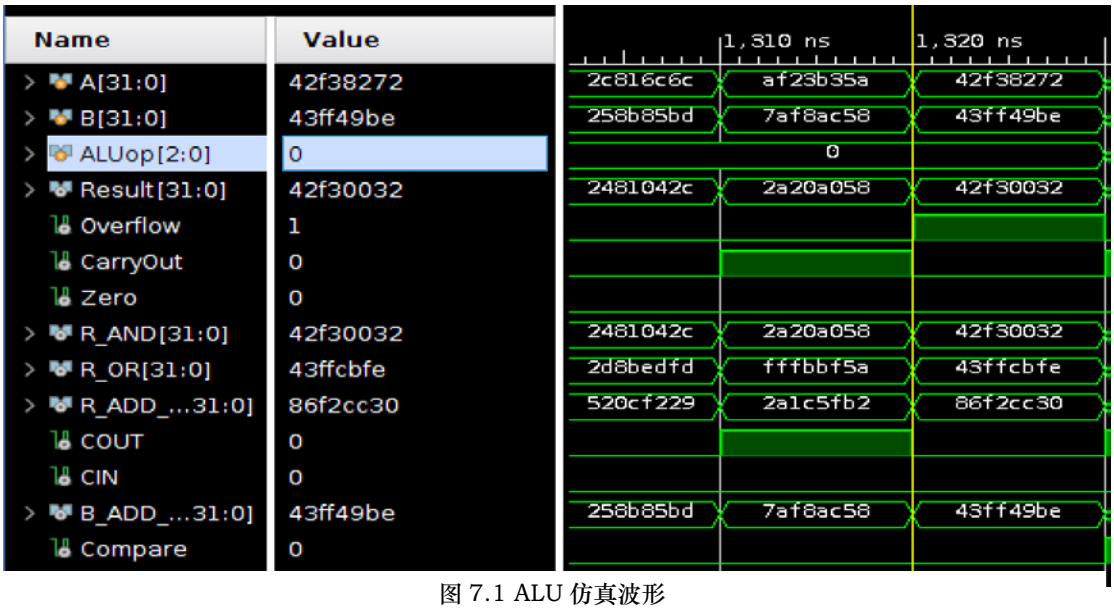

图 7.1 ALU 仿真波形

1330ns 前，ALUop=0，执行逻辑按位与操作。

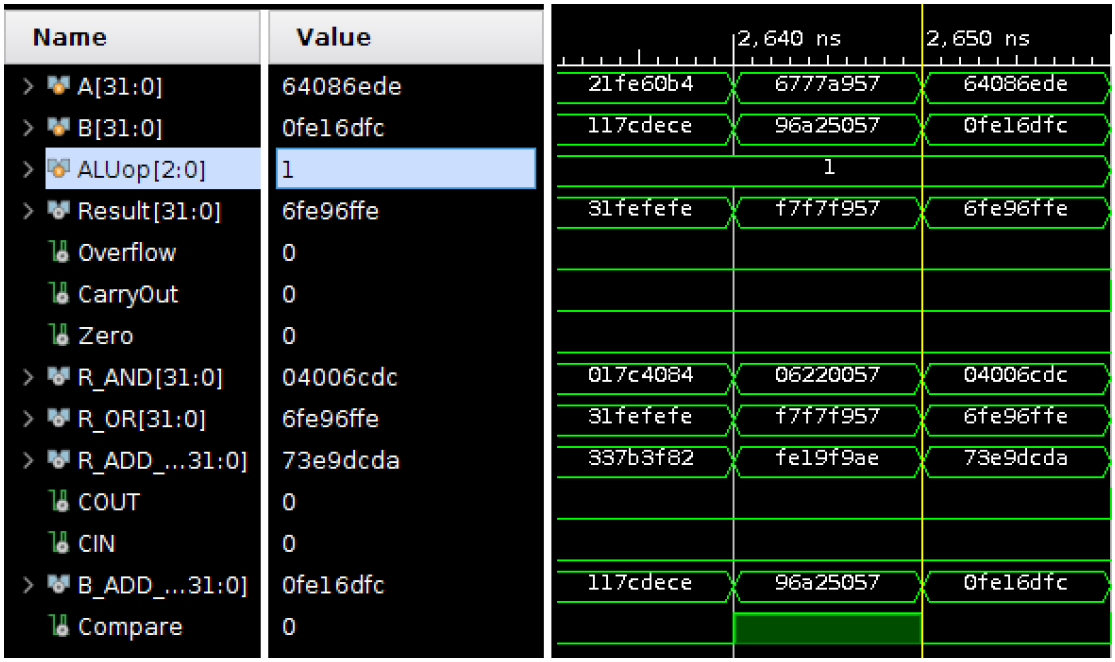

图 7.2 ALU 仿真波形

2660ns 前，ALUop=1，执行逻辑按位或操作。

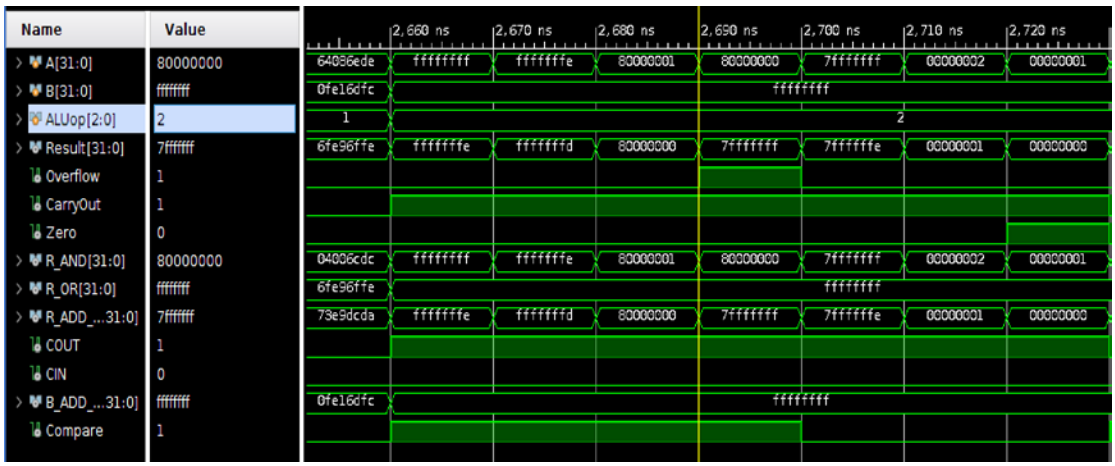

图 7.3 ALU 仿真波形

ALUop=2，执行加法操作，且在 2690ns 处产生有符号数加法溢出/无符号数加法进位。

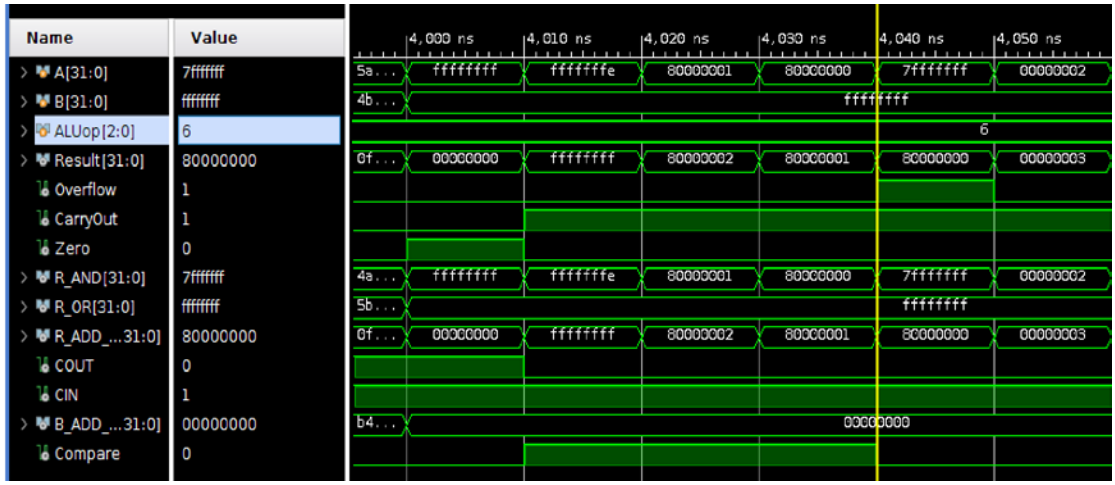

图 7.4 ALU 仿真波形

ALUop=6，执行减法操作，且在 4040ns 处产生有符号数减法溢出/无符号数减法借位。在 4000ns 处结果为 32’b0，则 Zero=1。

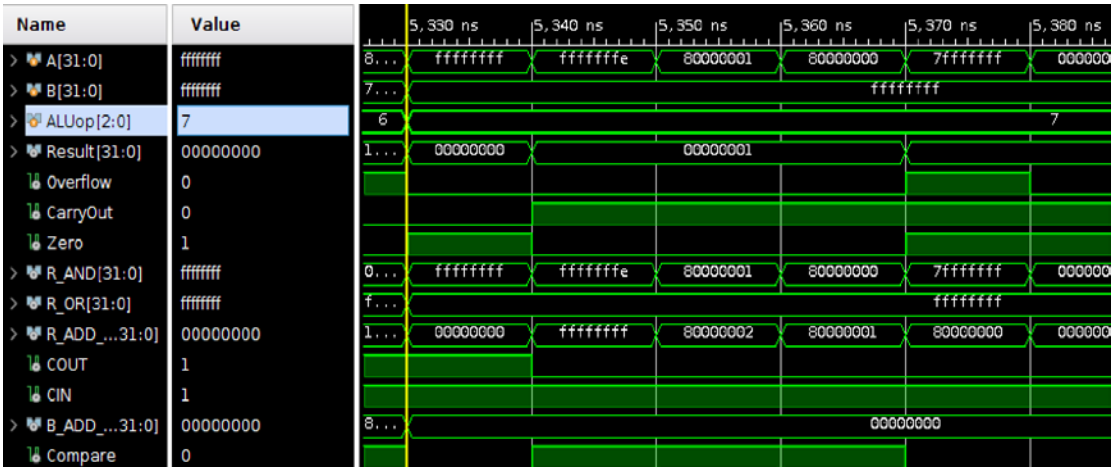

图 7.5 ALU 仿真波形

5330ns 起，ALUop=7，执行有符号数比较操作。

二、 实验过程中遇到的问题、对问题的思考过程及解决方法

问题 1：RTL 语法错误

解决方法：逐行梳理代码/在 vivado 中打开代码，根据自动纠错修改/根据仿真时的 log 文件定位错误位置。

问题 2：RTL 逻辑 bug（CarryOut，Overflow，SLT）

解决方法：由于云平台调试 bhv_sim 阶段报错中止，仅有一组错误数据。首先根据报错定位错误位置，再自己写 testbench 根据波形确定 bug 并加以修复。

注：RTL 逻辑简化

思考过程：先考虑所有情况，随后利用卡诺图对简单部分加以简化，最后分析各种情况对应的逻辑，对复杂表达部分用等价的简单信号加以替换。

问题 3：云平台调试时因操作不当报错

解决方法：根据讲义以及群内问题寻找是否已有解决方案/询问助教。

三、 对讲义中思考题（如有）的理解和回答

此次实验无思考题。

四、 在课后，我花费了大约 7 小时完成此次实验。

五、 对于此次实验的心得、感受和建议

1．实验心得:

在本次 project1 实验中，我对通用寄存器堆 Register File 和 ALU 的工作原理以及工作方式、特点有了进一步的认识。

在 ALU 代码编写过程中，我对 Verilog 代码的规范性有了进一步的认识，如：尽量避免直接使用门电路语言来进行赋值，应使用“|、&”等。

同时，在编写代码的时候需要考量“简洁”与“通用”之间的取舍。例如：在本次 ALU 实验中，对于最后输出的结果选择，可以使用上述代码中的多路选择器的实现方式，该种方式也便于后续 ALUop 信号的更改与添加；也可以根据信号的特点进行特殊化的处理，用更少的语言和门电路实现，但该种方式不利于后续 ALUop 信号的更改与添加。

调试代码时，可以自己写 testbench 进行测试。

2．实验感受

云平台调试时若产生结果错误会终止，仅输出一种错误情况、一组错误结果，而并不生成波形，一次只能调试一部分，不利于整体调试。希望能够一次输出所

有错误结果，或者给出错误的波形。

ALU 可有多种方式实现同一功能，寻找相对较优的实现方式会花费一定的时间。

十分感谢王嵩岳助教在我使用云平台出错时的帮助。

3. 实验建议

希望实验课可以拓展讲授完整 ALU 的功能以及其与其他部件交互的方式；或是以补充资料的方式附在课程讲义最后。

希望后续课程中可以选择尝试直接使用 FPGA 进行实验。