中国科学院大学计算机组成原理实验课

实 验 报 告

学号: 2015K8009929045 姓名: 张远航 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——ALU & Register File

注 1: 本实验报告请以 PDF 格式提交。文件命名规则: [学号]-PRJ[实验序号]-RPT.pdf, 其中文件名字母大写, 后缀名小写。例如: [2014K8009959088]-PRJ[1]-RPT.pdf
注 2: 实验报告模板以下部分的内容供参考, 可包含但不限定如下条目内容。

一、 逻辑电路结构与仿真波形的截图及说明 (比如关键 RTL 代码段 {包含注释} 及其对应的逻辑电路结构、相应信号的仿真波形和信号变化的说明等)

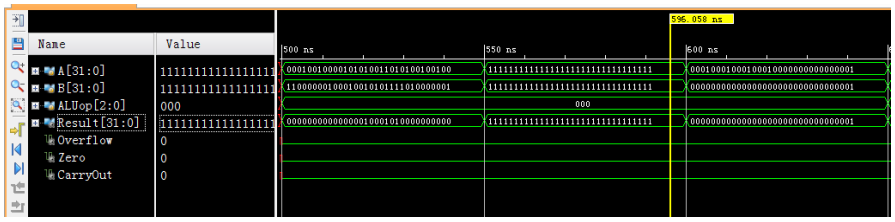
行为级实现的 ALU 代码如下, 逻辑电路图见图 1 :

```
1   ... (port definition, omitted)
2   // behavioral implementation
3   parameter   AND = 3'b000,   // bitwise and
4               OR  = 3'b001,   // bitwise or
5               ADD = 3'b010,   // arithmetic addition
6               SUB = 3'b110,   // arithmetic subtraction
7               SLT = 3'b111;   // set-on-less-than
8
9   // stores the last carry
10  reg LastCout;
11  reg [`DATA_WIDTH - 1:0] Result;
12
13  always @(A, B, ALUop) begin
14      case (ALUop)
15          AND: begin
16              Result = A & B; LastCout = `DATA_WIDTH'b0;
17          end
18           OR: begin
19              Result = A | B; LastCout = `DATA_WIDTH'b0;
20           end
21          ADD: {LastCout, Result} = A + B;
22          SUB: {LastCout, Result} = A - B;
23          // note: MIPS manual specifies the operands of SLT
                as signed
```

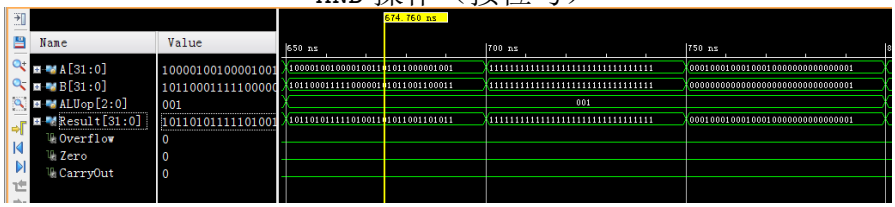
```
24              SLT: begin
25                  Result = ($signed(A) < $signed(B))? `DATA_WIDTH'
                        b1 : `DATA_WIDTH'b0; LastCout = `DATA_WIDTH'
                        b0;
26              end
27              // catches exceptions, should not occur
28              default: begin
29                  Result = `DATA_WIDTH'b0; LastCout = 0;
30              end
31          endcase
32      end
33
34      /*
35          CarryOut: defined for ADD and SUB, undefined otherwise (
                casts 0)
36          - CarryOut = LastCout XOR IS_SUB
37          - for A-B, we actually use A+~B+1 so LastCout should be
                inverted for SUB
38      */
39      assign CarryOut = (ALUop == ADD && LastCout)
40          || (ALUop == SUB && (~LastCout ^ 1));
41      /*
42          Overflow: defined for ADD and SUB, undefined otherwise (
                casts 0)
43          2's complement overflow happens:
44           - if a sum of two positive numbers results in a
                negative number
45           - if a sum of two negative numbers results in a
                positive number
46      */
47      assign Overflow = (ALUop == ADD && (A[`DATA_WIDTH - 1] == B[
            `DATA_WIDTH - 1]) && (LastCout ^ Result[`DATA_WIDTH - 1])
            )
48           || (ALUop == SUB && (A[`DATA_WIDTH - 1] == ~B[
                `DATA_WIDTH - 1]) && (~LastCout ^ Result[`DATA_WIDTH
                - 1]));
49      // Zero: defined for SUB, undefined otherwise (casts 0)
50      assign Zero = (ALUop == ADD || ALUop == SUB)
51                  && (!Result);
```

```
52  endmodule
```

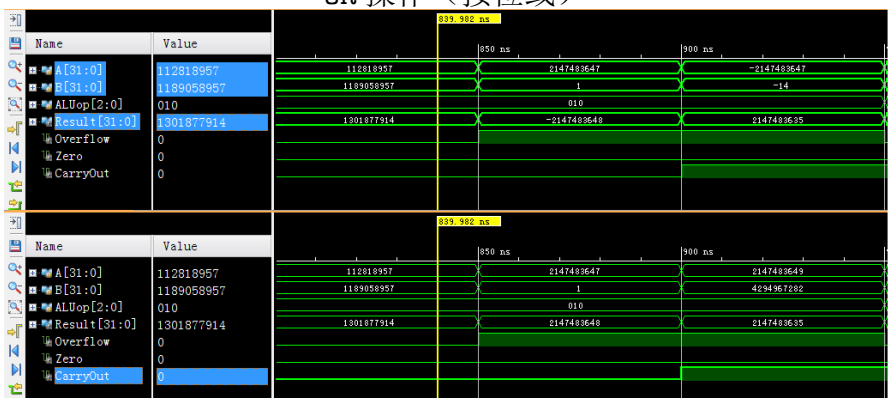
ALU 波形如下所示。对每个操作，有两个手工编写的样例和一个随机样例。可以直接看出，加减法操作正确产生了进位信号 (表示无符号数运算结果是否超出范围) 和溢出信号 (表示有符号数运算结果是否超出范围)；SLT 操作能够比较同号和异号的数。注意，CarryOut 有效表示无符号加法有进位、无符号减法无借位。此外，我们还加入两个专门用于检测 Zero 信号的测试点。



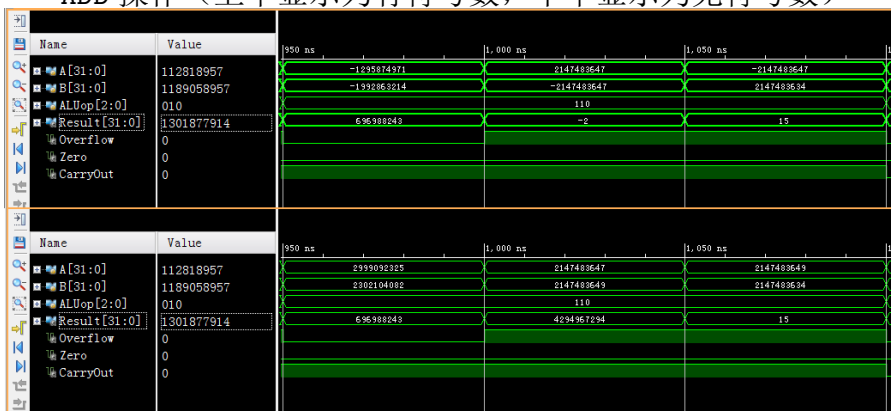
AND 操作（按位与）



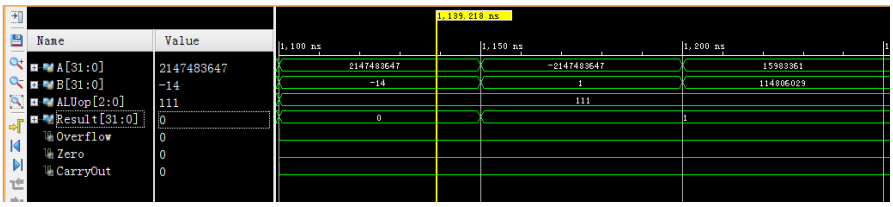
OR 操作（按位或）



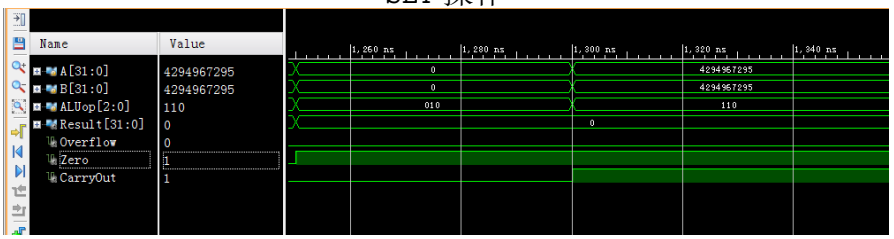
ADD 操作（上半显示为有符号数，下半显示为无符号数）



SUB 操作（上半显示为有符号数，下半显示为无符号数）



SLT 操作



检查 Zero 信号

寄存器堆代码如下，逻辑电路图见图 2：

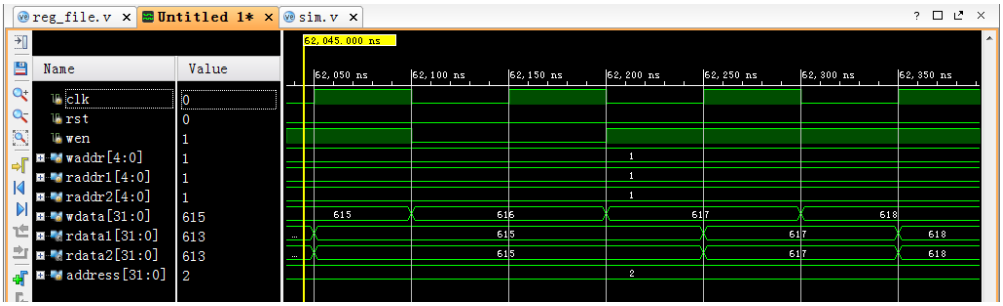
```
    ... (port definition, omitted)
    // calculate number of storage units with logical shift
    reg [1 << `ADDR_WIDTH - 1:0] r[`DATA_WIDTH - 1:0];
    integer i;

    always @(posedge clk) begin
        if (rst) begin
            // on reset, wipe memory
            for (i = 0; i < (1 << `ADDR_WIDTH); i = i + 1)
                r[i] <= `DATA_WIDTH'b0;
        end
        // zero register ($zero) always gives a value of zero
           and should not be overwritten
        else r[waddr] <= (wen && waddr)? wdata : r[waddr];
    end

    assign rdata1 = r[raddr1];
    assign rdata2 = r[raddr2];

endmodule
```

仿真波形局部如下图所示。62050ns 时向 2 号寄存器写入了数据 615，在 62150ns 时写使能信号无效，读出数据仍为 615；在 62200ns 时写使能开始有效，但读出数据仍为 615，直到 62250ns 下一上升沿到来时才写入新数据 617，读出的数据也跟着发生变化。



二、 实验过程中遇到的问题、对问题的思考过程及解决方法（比如 RTL 代码中出现的逻辑 bug，仿真及上板调试过程中的难点等）

(a) ALU 中，赋值不完整 导致综合产生锁存器（latch）。

```
case (ALUop)
    AND: Result = A & B;
    OR: Result = A | B;
    ADD: {LastCout, Result} = A + B;
    ...
    default: Result = 0;
    // wrong: LastCout is not properly assigned
endcase
```

对加减法以外的情况补充赋值 `LastCout = 0` 前后的 RTL 图对比见图 3 。

(b) 要表示 32 位全为 1，正确的写法应该是 `32'hFFFFFFFF` 或 `32'b111...1` 而非 `32'bFFFFFFFF`。

(c) case 语句中，一个情况下如有多条语句要有 `begin` 和 `end`。

(d) 括号嵌套错误。

```
assign CarryOut = LastCout ^ (ALUop == SUB) ? 1 : 0;
// wrong
assign CarryOut = (LastCout ^ (ALUop == SUB))? 1 : 0;
// correct
```

(e) 使用三目运算符时，`(A == B)? [variable] : 0` 总写成 `(A == B)? 0 : [variable]`。

(f) 实验初期对 `CarryOut` 和 `OverFlow` 信号的理解不透彻，做减法时未对 `Cout[31]` 取反导致结果出错。

(g) 实现寄存器堆时，局限于本次试验数据位宽和存储器单元数恰好相等的情形。

```
reg [`DATA_WIDTH - 1:0] r[`DATA_WIDTH - 1:0];
// wrong
reg [1 << `ADDR_WIDTH - 1:0] r[`DATA_WIDTH - 1:0];
// correct
```

(h) 误认为 Verilog 中的比较操作是对有符号数进行的。要对有符号数进行比较，可以在变量名前加 Verilog 中定义的函数 `$signed`（小技巧）。

三、 对讲义中思考题的理解和回答

问题 1 (对比位宽与地址长度)

`reg [7:0] r` 定义一个 8 位的寄存器，而 `reg r [7:0]` 定义了一个由 8 个 1 位寄存器组成的寄存器组，或者说，一个有 8 个存储单元，每个存储单元位宽为 1 的存储器。Verilog 语言中没有多维数组，所以采用寄存器组来表示存储器，用来对 ROM、RAM 或寄存器堆建模。

问题 2 (推导无符号减法有借位 $\Rightarrow$ `Cout[31]` 无效)

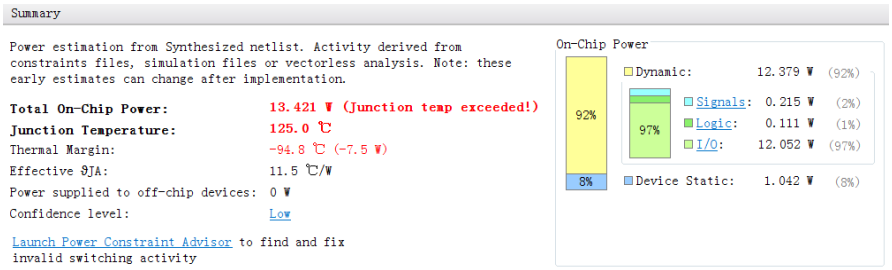
- 减法运算 (A-B=C) 有借位
- $\Leftrightarrow$ 不够减 [“不够减”的数学定义]
- $\Leftrightarrow$ C 的真值比 A 大 [“不够减”在加法器中的直观含义]
- $\Leftrightarrow$ 加法器运算 (A + (-B) = C) 无进位 [加法器的进位定义]
- $\Leftrightarrow$ `Cout[31]` 无效

四、 对于此次实验的心得、感受和建议（比如实验是否过于简单或复杂，是否缺少了某些你认为重要的信息或参考资料，以及其他想与任课老师交流的内容等）

- ALU 的实现难度主要取决于所用的描述方式。采用行为级描述代码相对比较简单而且易于调试，相比之下手工编写超前进位加法器比较容易出错，电路会变得更复杂，而且寄存器位数改变的话又要重新考虑拼接方式。但是都做的好处在于二者可以并行执行相互对照结果。

另一方面，串行的效率太低，不考虑了……如果要实现的话就是按 Patterson 书里给的那种实现来就好了（用 `genvar` 语句可以减少一点复制粘贴实例化的工作量）。

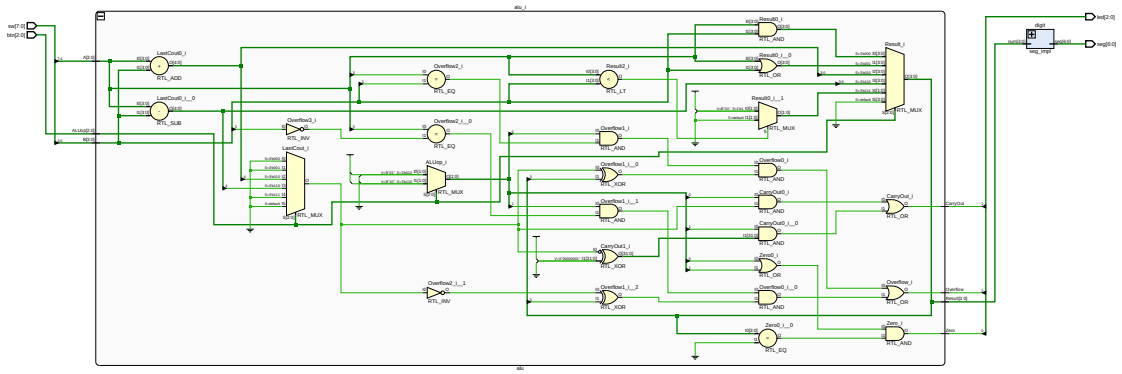
- 寄存器堆（RF）实现起来更加简单，但是作为时序电路也更容易出现不易觉察和理解的错误。
- 良好的代码书写和注释习惯可以节省大量的 debug 时间。
- 给的约束文件中没加 timing，于是跑出来 implementation 之后 Vivado 估计的功率变得很大（junction temperature exceeded）？[1]而且大部分耗能在数码管上?? 尽管提示置信度是低，看着还是很不爽啊……



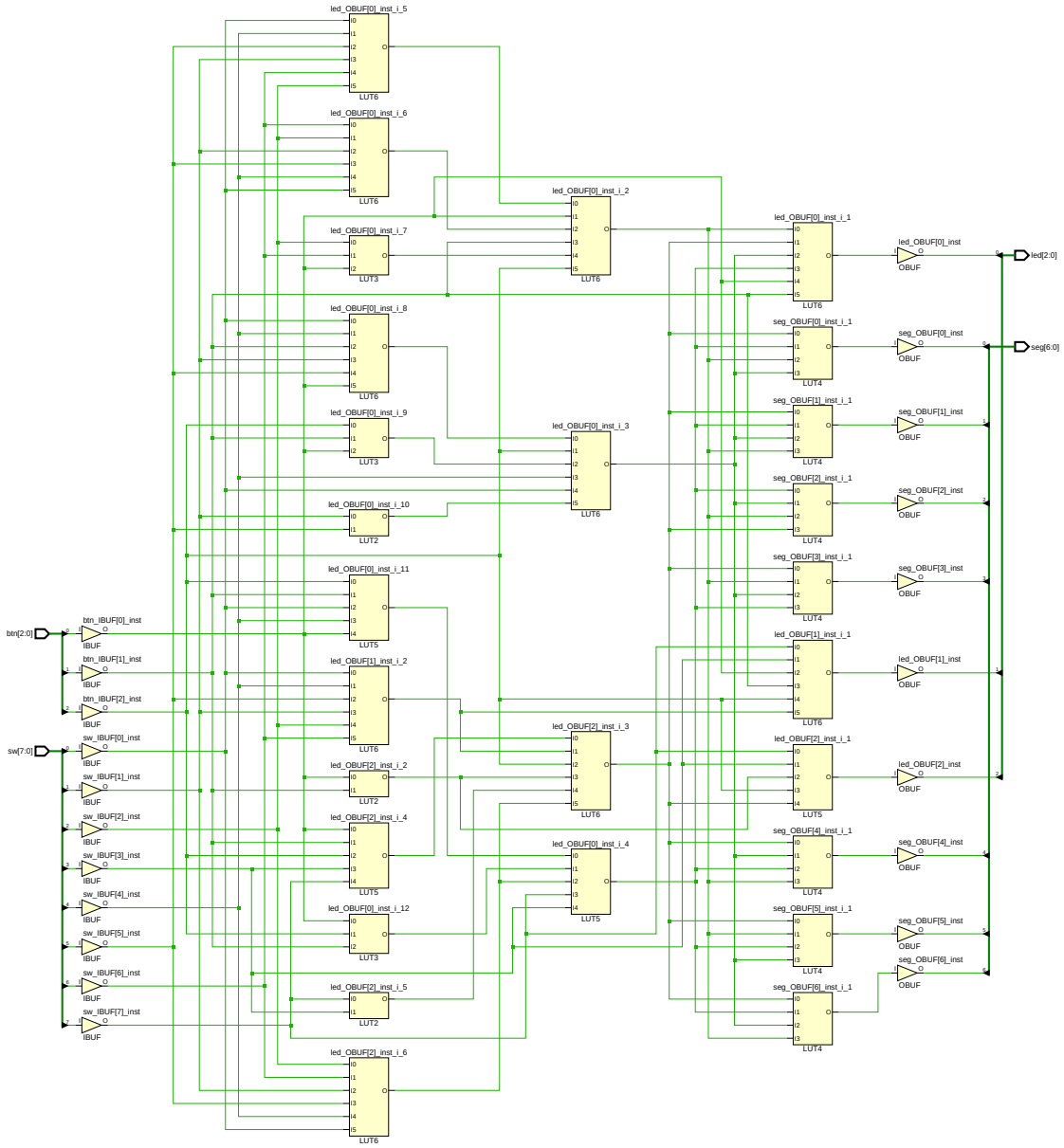
- 感谢认真检查了我的代码的张科老师，今后我会更加严谨，努力做到完美。
- 为实验报告重制了个 LaTeX 版的模板，这样就可以加引用和插入 Vivado 生成的矢量 RTL 图了！

[1]Just a wild guess, but at least it's what happened to this guy.

插图

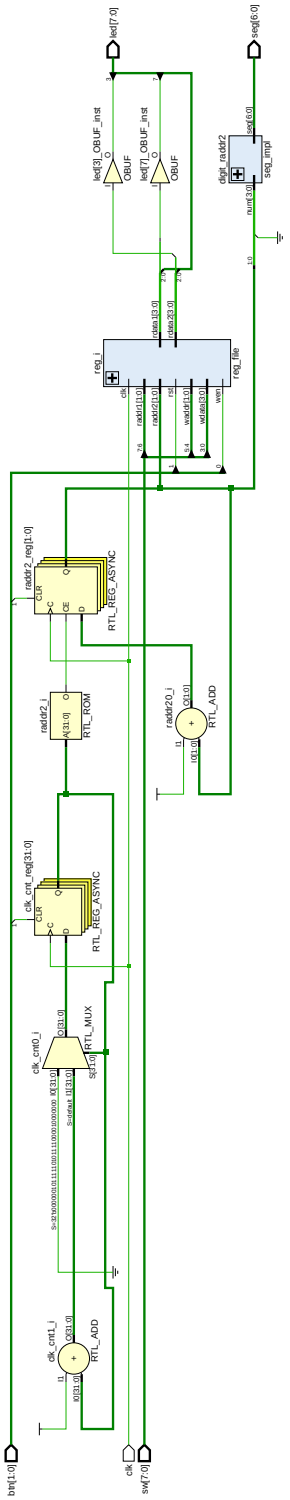


(a) RTL 图

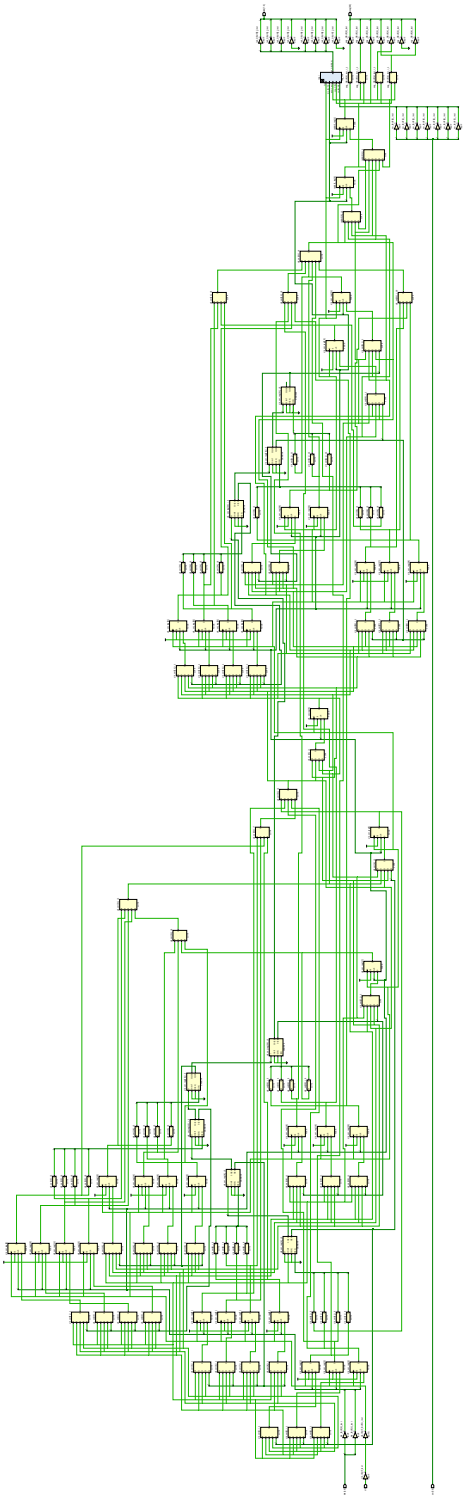


(b) 综合后

图 1: ALU 行为级实现的逻辑电路结构

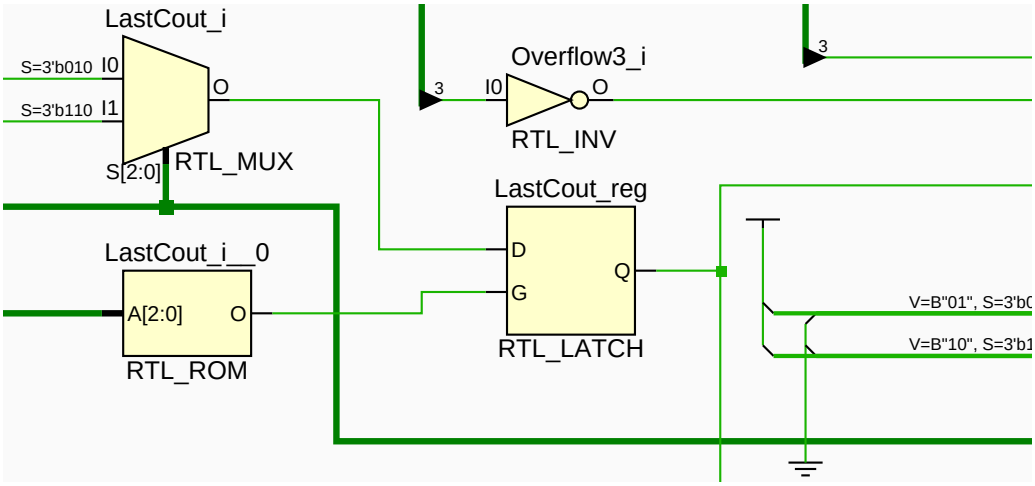


(a) RTL 图

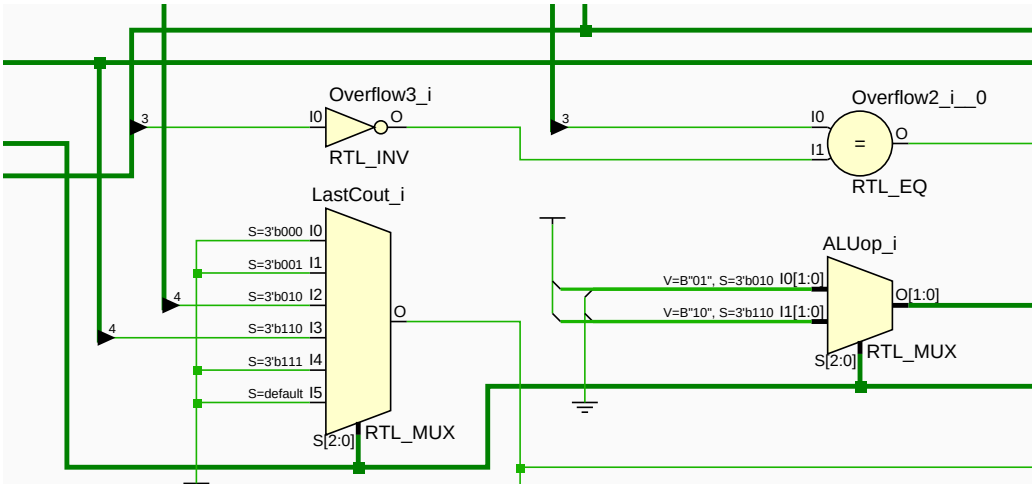


(b) 综合后

图 2: 寄存器堆的逻辑电路结构



(a) 错误结果（出现了 latch）



(b) 正确结果

图 3: 赋值不完整导致错误