Lab3 算术逻辑部件实验

一、实验目的

1. 掌握先行进位加法器的设计方法和应用。
2. 掌握桶形移位器的设计方法和应用。
3. 掌握数值比较器的设计方法。

二、实验环境

1. Vivado 开发环境
2. Xilinx A7-100T 实验板

三、实验原理

全加器（FA）将加数、被加数和来自低位进位相加，生成和、进位两个输出位。将 n 个 1 位全加器相连可得 n 位加法器，称为行波进位加法器（CRA）。串行进位加法器所用元件较少，但进位传递时间较长，如图 3.1 所示。

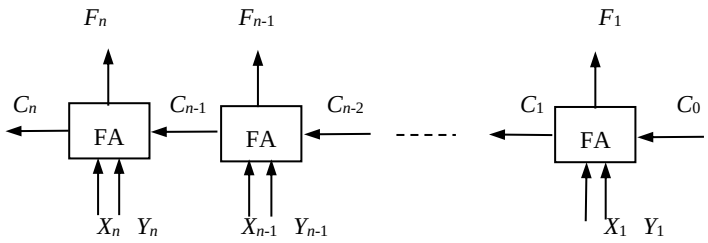


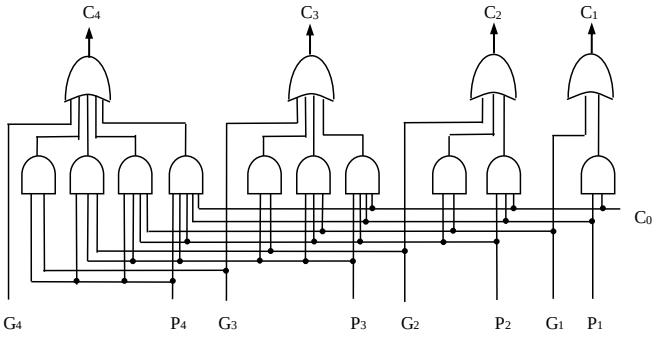
图 3.1 *n* 位串行进位加法器

根据串行加法器的原理图，利用结构化建模方式实现的 4 位串行加法器的模块代码如下：

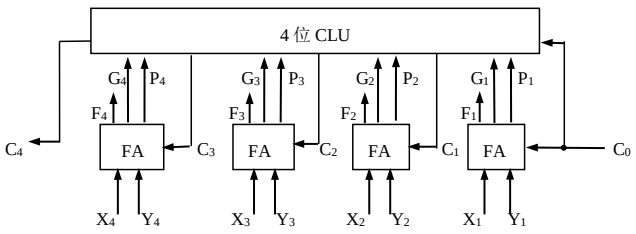
```
//全加器
module FA(
    output   f,cout,
    input    x,y,cin
 );
    assign   f= x ^ y ^ cin;
    assign   cout=( x & y) | (x & cin ) | ( y & cin);
```

```
endmodule
//四位串行加法器
module CRA (
        output [3:0] f,
        output cout,
        input [3:0] x, y,
        input cin
    );
        wire [4:0] c;
        assign c[0] = cin;
        FA fa0(f[0], c[1], x[0], y[0], c[0]);
        FA fa1(f[1], c[2], x[1], y[1], c[1]);
        FA fa2(f[2], c[3], x[2], y[2], c[2]);
        FA fa3(f[3], c[4], x[3], y[3], c[3]);
        assign cout = c[4];
endmodule
```

为了提高加法器的运算速度，必须尽量避免进位之间的依赖关系，可以通过设计先行进位部件（CLU）来实现。利用先行进位部件实现的加法器称为先行进位加法器（CLA）。因为各个数位的进位是并行产生的，所以是一种并行进位加法器。4 位先行进位部件和先行进位加法器原理图，如图 3.2 所示。



a）4 位 CLU



b） 4 位全先行进位加法器

图 3.2　4 位 CLU 和 4 位全先行进位加法器原理图

要实现先行进位部件，需要修改全加器电路，使其能够产生进位传递因子和进位生成因子。再利用先行进位部件，实现先行进位加法器。

输出进位因子和传递因子的全加器模块代码如下：

```
module FA_PG (
        output f, p, g,
        input x, y, cin
       );
          assign f = x ^ y ^ cin;
          assign p = x | y;
          assign g = x & y;
endmodule
```

4 位先行进位部件模块代码如下：

```
module CLU (
          output [4:1] c,
          input [4:1] p, g,
          input c0
);
          assign c[1] = g[1] | (p[1] & c0);
          assign c[2] = g[2] | (p[2] & g[1]) | (p[2] & p[1] & c0);
          //  以下两个表达式使用了位拼接运算和归约运算
          assign c[3] = g[3] | (p[3] & g[2]) | (&{p[3:2], g[1]}) | (&{p[3:1], c0});
          assign c[4] = g[4] | (p[4] & g[3]) | (&{p[4:3], g[2]}) | (&{p[4:2], g[1]}) | (&{p[4:1], c0});
endmodule
```

实现 4 位先行进位加法器的模块如下：

```
module CLA (
        output [3:0] f,
        output cout,
        input [3:0] x, y,
          input cin
);
          wire [4:0] c;
          wire [4:1] p, g;
          assign c[0] = cin;
          FA_PG fa0(f[0], p[1], g[1],x[0], y[0], c[0]);
          FA_PG fa1(f[1], p[2], g[2],x[1], y[1], c[1]);
          FA_PG fa2(f[2], p[3], g[3],x[2], y[2], c[2]);
          FA_PG fa3(f[3], p[4], g[4],x[3], y[3], c[3]);
          CLU clu(c[4:1],p, g, c[0]);
          assign cout = c[4];
endmodule
```

考虑到输入输出扇出系数的影响，更多位数的加法器可以通过 4 位先行进位加法器级联来实现。例如 16 位加法器可以分成 4 位一组，组内采用 4 位先行进位，组间也采用先行进位的方式来产生进位，这样组内和组件都采用先行进位方式，使得加法器的延迟时间和操作数的位数没有关系。

支持生成组件进位传递因子和进位因子的/4 位组间先行进位加法器模块代码如下：

```
module CLA_group (
        output [3:0] f,
        output pg,gg,
        input [3:0] x, y,
         input cin
);
         wire [4:0] c;
         wire [4:1] p, g;
         assign c[0] = cin;
         FA_PG fa0(f[0], p[1], g[1],x[0], y[0], c[0]);
         FA_PG fa1(f[1], p[2], g[2],x[1], y[1], c[1]);
         FA_PG fa2(f[2], p[3], g[3],x[2], y[2], c[2]);
         FA_PG fa3(f[3], p[4], g[4],x[3], y[3], c[3]);
         CLU clu(c[4:1],p, g, c[0]);
         assign pg=p[1] & p[2] & p[3] & p[4];
         assign gg= g[4] | (p[4] & g[3]) | (p[4] & p[3] & g[2]) |    (p[4] & p[3] & p[2] & g[1]);
endmodule
```

使用结构化建模方式实现的 16 位先行进位的模块代码如下：

```
module CLA_16(
          output wire [15:0] f,
          output wire    cout,
          input [15:0] x, y,
          input cin
   );
          wire [3:0] Pi,Gi;                // 4 位组间进位传递因子和生成因子
          wire [4:0] c;                    // 4 位组间进位和整体进位
          assign c[0] = cin;
         CLA_group cla0(f[3:0],Pi[0],Gi[0],x[3:0],y[3:0],c[0]);
         CLA_group cla1(f[7:4],Pi[1],Gi[1],x[7:4],y[7:4],c[1]);
         CLA_group cla2(f[11:8],Pi[2],Gi[2],x[11:8],y[11:8],c[2]);
         CLA_group cla3(f[15:12],Pi[3],Gi[3],x[15:12],y[15:12],c[3]);
         CLU clu(c[4:1],Pi,Gi, c[0]);
         assign cout = c[4];
 endmodule
```

四、实验内容

1、带标志位的加减运算部件

要实现带符号整数的加/减运算，就需要在无符号数加法器的基础上增加相应的门电路。利用加法器实现减法运算：把减数 Y 取反，低位进位 Cin 置 1。因此，只要在原加法器的 Y 输入端，加 n 个反相器以实现取反的功能，再用一个 2 选 1 多路选择器来选择加法器的输入数据，用一个控制端 Sub 来控制选择将 Y（sub=0）还是 Y 反码（sub=1）输入到加法器的输入端，并将控制端 Sub 作为低位进位 Cin 送到加法器。输出不仅包含加减法运算结果，还要能够生成相应的标志位信息，用来分析运算的结果。常用的标志位有溢出标志 OF、符号位标志 SF、进位/借位标志 CF 和结果为 0 标志位 ZF，如图 3.3 所示。

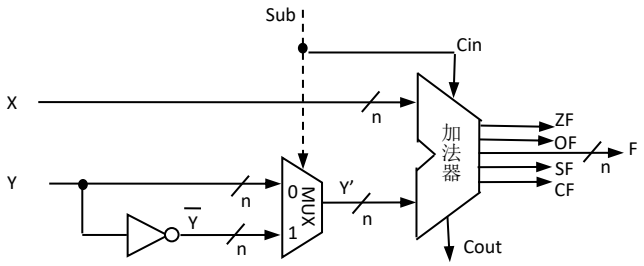


图 3.3 带标志位的加减运算部件逻辑图

溢出标志 OF 的逻辑表达式为 OF=Cn⊕Cn-1 或者 OF=$\overline{X_{n-1}} \cdot \overline{Y_{n-1}} \cdot F_{n-1} + X_{n-1} \cdot Y_{n-1} \cdot \overline{F_{n-1}}$；符号位标志 SF 就是运算结果 F 的最高位，即 SF=Fn-1。进位/借位标志 CF=Cout⊕Cin，即当 Cin=0 时，表示执行加法运算，CF 表示进位；当 Cin=1 时，表示执行减法运算，CF 表示借位。零标志 ZF=1 当且仅当 F=0。

不使用“+/-”运算操作符号，利用上一节 16 位先行进位加法器 CLA_16 模块级联实现 32 位加法器。使用结构化建模方式实现带有标志位的加减法器的模块端口定义如下：

```
module Adder32 (
   output [31:0] f,
   output OF, SF, ZF, CF,
   output cout,
   input [31:0] x, y,
   input sub
);
    //add your code here
endmodule
```

请根据上述描述，按照下列步骤完成实验。

1、 使用Vivado创建一个新工程。

2、 点击添加设计源码文件，加入lab3.zip里的Adder32.v文件。

3、 点击添仿真测试文件，加入lab3.zip里的Adder32_tb.v文件。

4、 根据实验要求，完成源码文件的设计。

5、 对工程进行仿真测试，分析输入输出时序波形和控制台信息。

2、桶形移位器

移位寄存器作为时序电路部件其每个时钟周期只能移动一位，移动效率比较低下，而桶形移位器采用组合逻辑的方式来实现移动功能，能在一个时钟周期内完成移动多位，具有很高的效率，常被用在 ALU 中来实现移位运算。它具有 n 位数据输入和 n 位数据输出，以及指定移动方向、移动类型（算术/逻辑/循环）和移动位数等。

8 位桶形移位器的输入输出引脚图，如图 3.4 所示。其中输入数据 din 和输出数据 dout 均为 8 位，移位位数 shamt 为 3 位。选择端 L/R 表示左移和右移，置为 0 为右移，置为 1 为左移。选择端 A/L 为算术/逻辑选择，置为 1 为算术右移，置为 0 为逻辑右移。

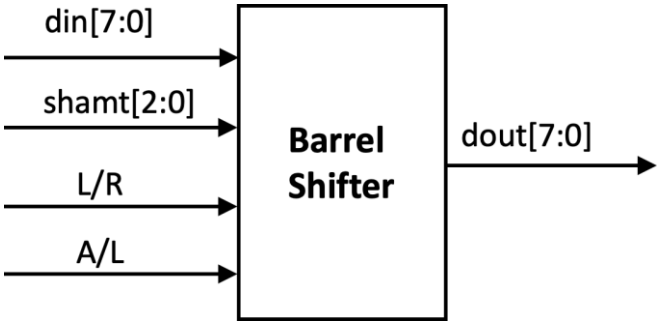


图 3.4　8 位桶形移位器逻辑符号图

8 位桶形移位器的具体实现使用了三级四路 8 位选择器来实现，原理图如图 3.5 所示。第一级利用 shamt[0]来控制是否需要移动一位，第二级在第一级的移动结果上用 shamt[1] 来控制是否要移动两位，第三级在第二级的基础上用 shamt[2]判断是否要移动四位。每个四路选择器有两位控制端，控制端低位 S0 对应 shamt[i]输入信号，判断是否需要移动和移动的位数。控制端高位 S1 对应 L/R 输入信号，判断移位的方向。对于算术和逻辑右移的选择，是通过控制右移时移入位是 0 还是 din[7]来实现的。

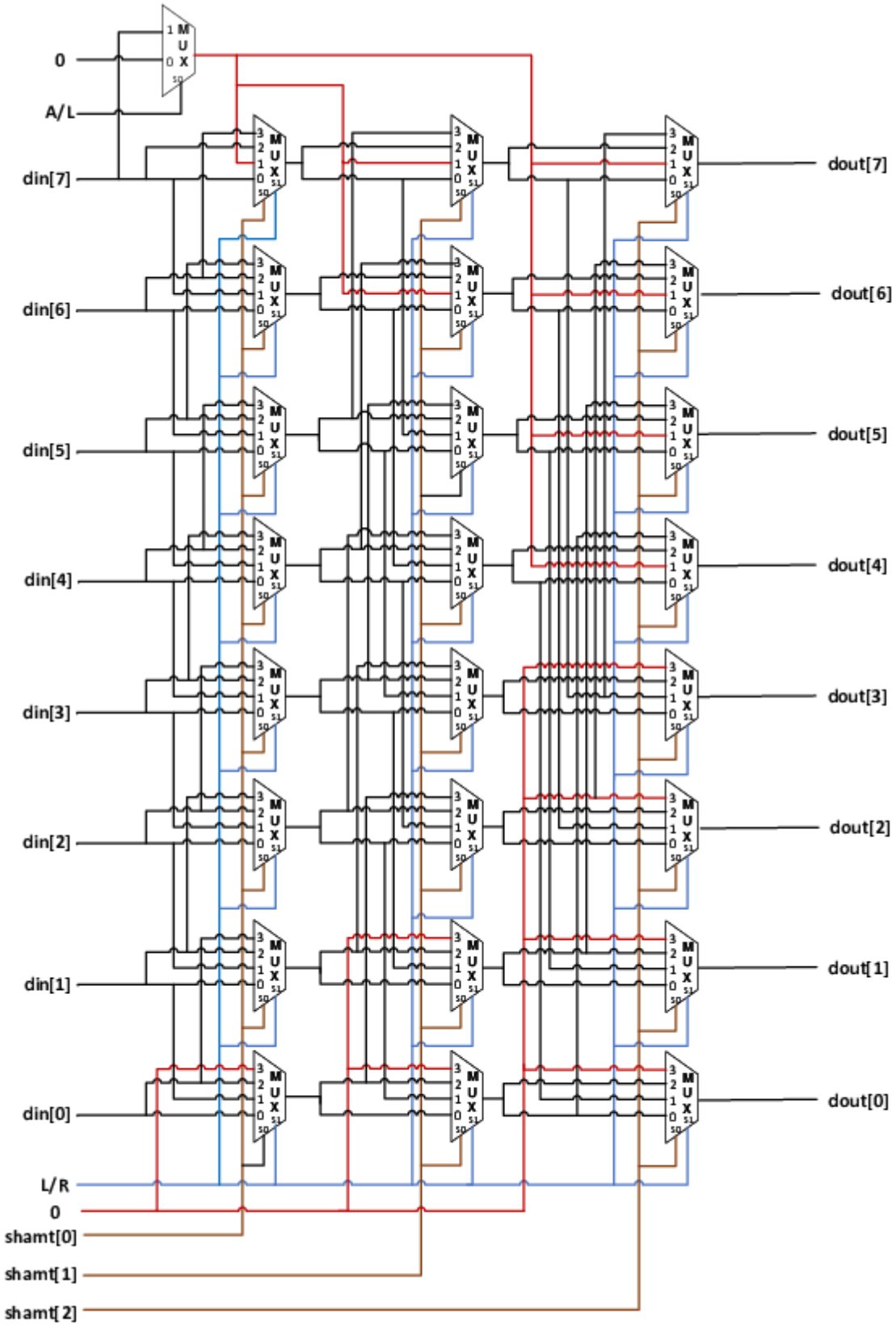


图 3.5 8 位桶形移位器原理图

根据 8 位桶形移位器结构实现 32 位的桶形移位器，需要将多路选择器的级联扩展到 5 级，数据位扩展到 32 位。采用数据流建模方式实现的 32 位桶形移位器 barrelsft32 的模块端口定义如下：

```
module barrelsft32 (
   output [31:0] dout,
   input [31:0] din,
   input [4:0] shamt,        //移动位数
   input LR,                 // LR=1 时左移，LR=0 时右移
   input AL                  // AL=1 时算术右移，AR=0 时逻辑右移
);
    //add your code here
endmodule
```

请根据上述描述，按照下列步骤完成实验。

1、 使用Vivado创建一个新工程。

2、 点击添加设计源码文件，加入lab3.zip里的barrelsft32.v文件。

3、 点击添仿真测试文件，加入lab3.zip里的barrelsft32_tb.v文件。

4、 根据实验要求，完成源码文件的设计。

5、 对工程进行仿真测试，分析输入输出时序波形和控制台信息。

3、32 位 ALU

ALU 是 CPU 中的核心数据通路部件之一，它主要完成 CPU 中需要进行的算术逻辑运算。RV32I 的 ALU 在操作控制信号 ALUctr 的指示下，执行相应运算，Result 作为 ALU 运算的结果被输出，零标志 Zero 被作为 ALU 的结果标志信息输出。

通过对 RV32I 的 37 条整数运算指令分析后发现，这些指令主要完成算术运算（加、减）、移位运算（左右移位、算术和逻辑移位）、逻辑运算（与、或、异或）、小于比较置数运算（带符号数和无符号数）、读取操作数 B 等操作。最终结果 Result 输出是通过一个八选一选择器选择不同运算部件的结果，选择端用 OPctr 信号来控制。ALU 的具体操作通过 4 位控制信号 ALUctr 生成具体控制信号来确定，这些控制信号包括：

SUBctr 信号：当 SUBctr=1 时，做减法运算，当 SUBctr=0 时，做加法运算；

SIGctr 信号：当 SIGctr=1，则执行“带符号整数比较小于置 1”，当 SIGctr=0，则执行“无符号数比较小于置 1”；

SFTctr 信号用来控制移位方向，当 SFTctr=0，则执行右移操作，当 SFTctr=1，则执行左移操作；

ALctr 信号用来设置算术移位还是逻辑移位，当 ALctr=0，则执行逻辑移位，当 ALctr=1，则执行算术移位；

OPctr[2:0]用来选择运算的结果 Result 输出，ALU 有加法器和、按位或、按位与、按位异或、移位器输出、操作数 B 输出、小于置 1 等 7 种运算结果，OPctr 信号需要 3 位二进制数来表示。图 3.6 给出一个实现 RV32I 指令中运算的 ALU 原理图。

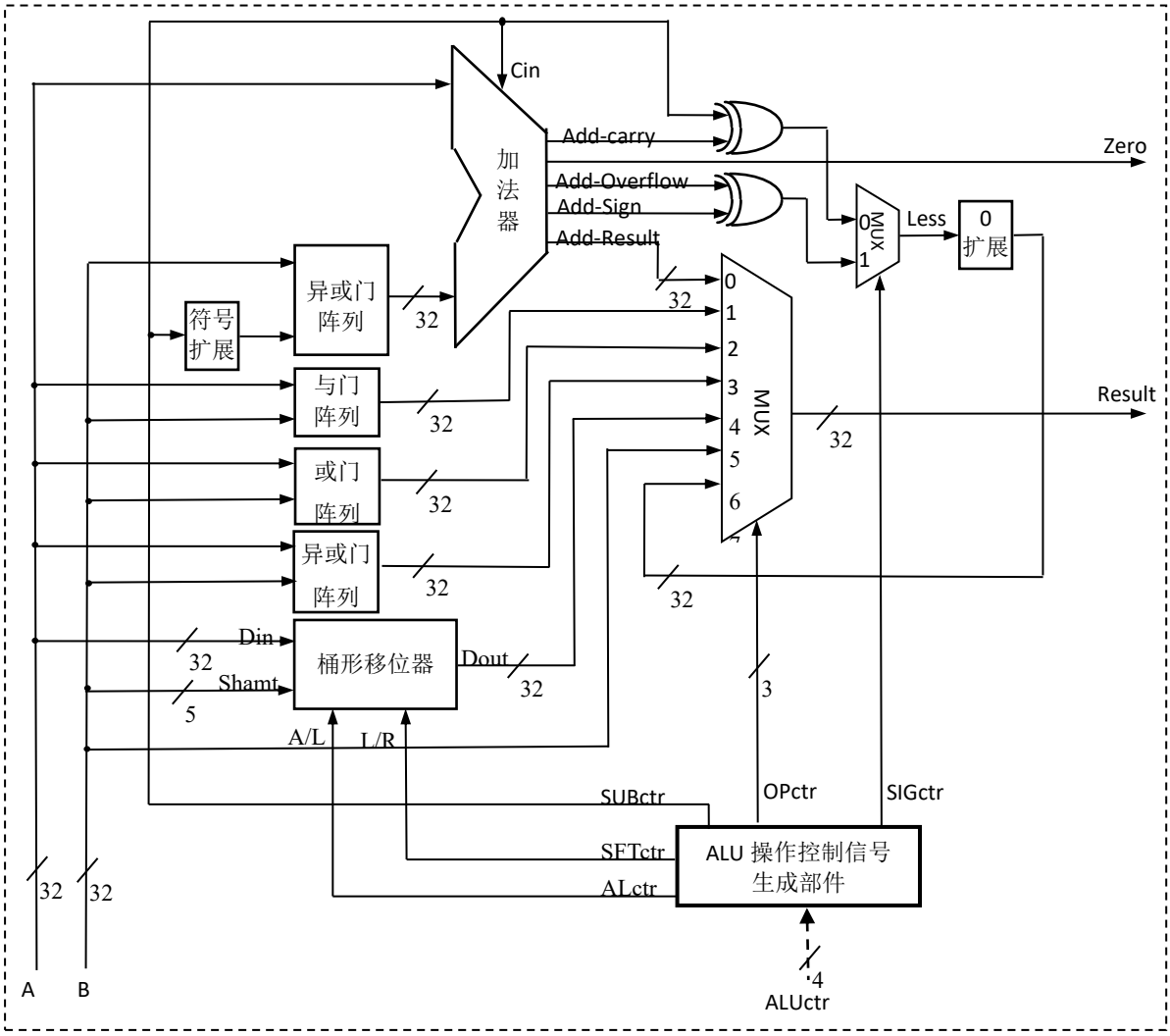


图 3.6　ALU 原理图

ALU 的操作控制与 RISC-V 指令中的 funct3 字段存在较强关系，因此，在对 ALUctr 进行编码时，可以根据 funct3 字段进行优化，使得后续控制器的设计变得更简单。表 3.1 给出了 ALUctr 的一种四位编码方案。

表 3.1　ALUctr 的四位编码及其对应的控制信号

ALUctr <3:0>	指令操作类型	SUBctr	SIGctr	ALctr	SFTctr	OPctr <2:0>	OPctr 的含义
0 0 0 0	auipc,addi,add,jal,jalr, lb,lh,lw,lbu,lhu,sb,sh,sw：加法	0	×	×	×	000	选择加法器的结果输出
0 0 0 1	sll,slli：逻辑左移	×	×	0	1	100	选择移位器结果输出
0 0 1 0	slt,slti,beq,bne,blt,bge：带符号数小于比较	1	1	×	×	110	选择小于置位结果输出

0 0 1 1	sltu,sltui,bltu,bgeu：无符号数小于比较	1	0	×	×	110	选择小于置位结果输出
0 1 0 0	xor,xori：异或	×	×			011	选择“按位异或”结果输出
0 1 0 1	srl,srli：逻辑右移	×	×	0	0	100	选择移位器结果输出
0 1 1 0	or,ori：或运算	×	×	×	×	010	选择“按位或”结果输出
0 1 1 1	and,andi：与运算	×	×	×	×	001	选择“按位与”结果输出
1 0 0 0	sub：减法	1	×	×	×	000	选择加法器的结果输出
1 1 0 1	sra,srai：算术右移	×	×	1	0	100	选择移位器结果输出
其余	（未用）						
1 1 1 1	lui：取操作数 B	×	×	×	×	101	选择操作数 B 直接输出

实例化上述 32 位加减法器模块 Adder32 和 32 位桶形移位器模块 barrelsft32，比较运算使用减法，逻辑运算使用位运算，假设 32 位算术逻辑部件 ALU32 的模块端口定义如下：

```
module ALU32(
    output   reg [31:0] result,        //32 位运算结果
    output   reg zero,                 //结果为 0 标志位
    input    [31:0] dataa,             //32 位数据输入，送到 ALU 端口 A
    input    [31:0] datab,             //32 位数据输入，送到 ALU 端口 B
    input    [3:0] aluctr              //4 位 ALU 操作控制信号
);
   //add your code here
endmodule
```

为了能够在实验板进行验证，分别使用 4 位输入开关表示输入端 a 和 b，使用 4 位输入开关表示 4 位 ALU 控制码，32 位运算结果现在 8 个七段数码管上，运算结果的低 16 位输出到 16 个 led 指示灯，结果为 0 的标志位输出到 3 色 led 的蓝色指示灯，建立 ALU32 模块的父级模块 ALU32_top，其模块端口定义如下：

```
module ALU32_top(
    output [6:0] segs,                 //七段数码管字形输出
    output [7:0] AN,                   //七段数码管显示 32 位运算结果
    output   [31:0] result,            //32 位运算结果
    output   zero,                     //结果为 0 标志位
    input    [3:0] data_a,             //4 位数据输入，重复 8 次后送到 ALU 端口 A
    input    [3:0] data_b,             //4 位数据输入，重复 8 次后送到 ALU 端口 B
    input    [3:0] aluctr,             //4 位 ALU 操作控制信号
    input    clk                       //时钟信号
);
   //add your code here
endmodule
```

请根据上述描述，按照下列步骤完成实验。

1、 使用Vivado创建一个新工程。

2、 点击添加设计源码文件，加入lab3.zip里的ALU32.v、ALU32_top.v文件。

3、 点击添仿真测试文件，加入lab3.zip里的ALU32_tb.v文件。

4、 点击添加约束文件，加入lab3.zip里的ALU32_top.xdc文件。

5、 根据实验要求，完成源码文件的设计。

6、 对工程进行仿真测试，分析输入输出时序波形和控制台信息。

7、 仿真通过后，进行综合、实现并生成比特流文件。

8、 生成比特流文件后，加载到实验开发板，进行调试验证，并记录验证过程。

五、思考题

1、分析 32 位 ALU 的资源占用情况。

2、如果比较运算直接使用组合电路比较器来实现，则 32 位 ALU 电路原理图需要做哪些修改？

3、在 32 位 ALU 的基础上如何实现 64 位的 ALU？

4、查找资料说明还有哪些并行加法器的设计方法，并详细介绍其中一种方法。