计组实验四报告

孙启翔 241220098

April 2025

1 4 位先行进位加法器设计

1.1 实验整体方案设计

在 CLU4 中，第 i 位的进位标志 C_i 可以由前面的进位传递函数 P_i 和进位生成函数 G_i 以及初始传入的进位 C_0 计算得出。传入的信号 $P_0 \sim P_3$ 和 $G_0 \sim G_3$ 可以由下面将要实现的 FA 传入。输出的信号 $C_0 \sim C_3$ 代表每位的进位。为实现组间并行计算，输出 P_g 和 G_g 代表组间的进位传递函数和组间进位生成函数。

在 FA 中，Cin 是进位传入信号，输出信号 F,P,G 分表表示该位的输出结果，进位传递信号和进位生成信号。

在 main 电路中，四个 FA 的输出 F 组成最终的结果 Z，C_4 即为最后的进位信号，输出信号 P_g 和 G_g 代表组间的进位传递和进位生成。在下面 16 位先行进位加法器也会用到。

1.2 实验原理图及电路图

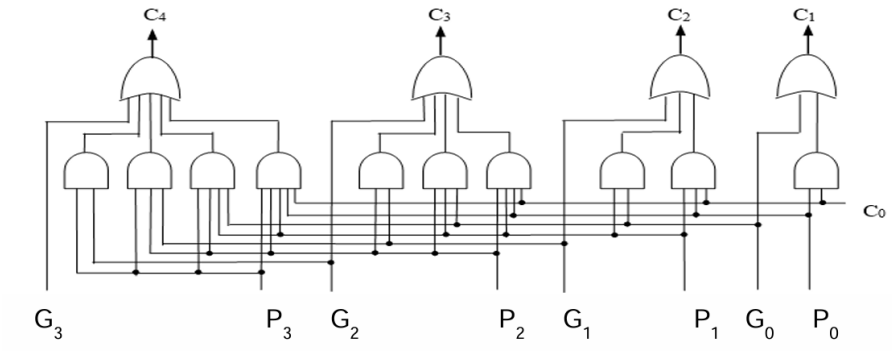

图 1: 原理图

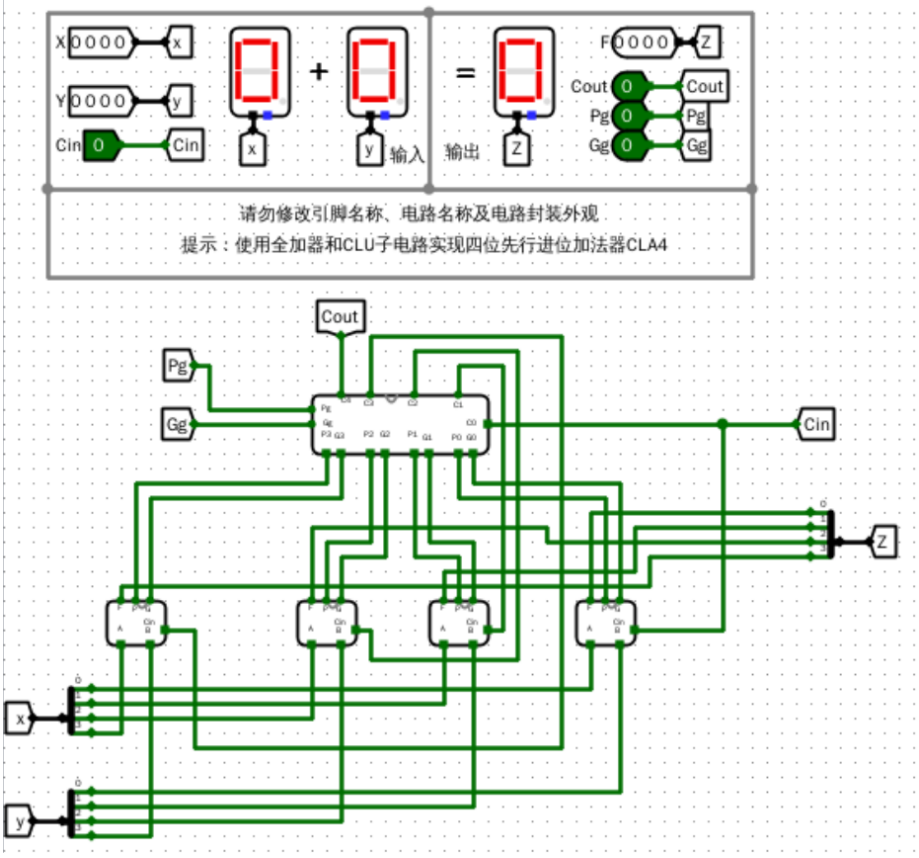
X 0000 x
Y 0000 y
Cin 0 Cin
x
y
输入
=
输出
Z
F 0000 Z
Cout 0 Cout
Pg 0 Pg
Gg 0 Gg
请勿修改引脚名称、电路名称及电路封装外观
提示：使用全加器和CLU子电路实现四位先行进位加法器CLA4
Cout
Pg
Gg
Cin
Z
x
y

图 2: main 电路图

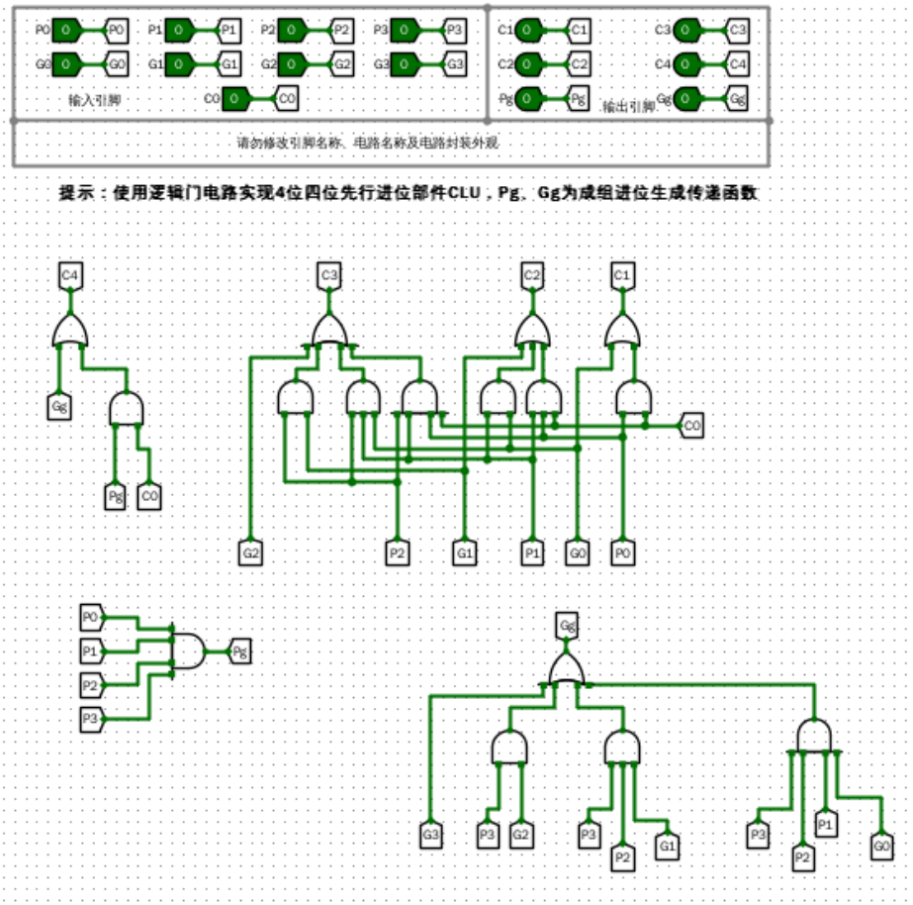
P0
P1
P2
P3
G0
G1
G2
G3
C0
输入引脚
C1
C2
C3
C4
Pg
Gg
输出引脚
请勿修改引脚名称、电路名称及电路封装外观
提示：使用逻辑门电路实现4位四位先行进位部件CLU，Pg、Gg为成组进位生成传递函数

图 3: CLU4 电路图

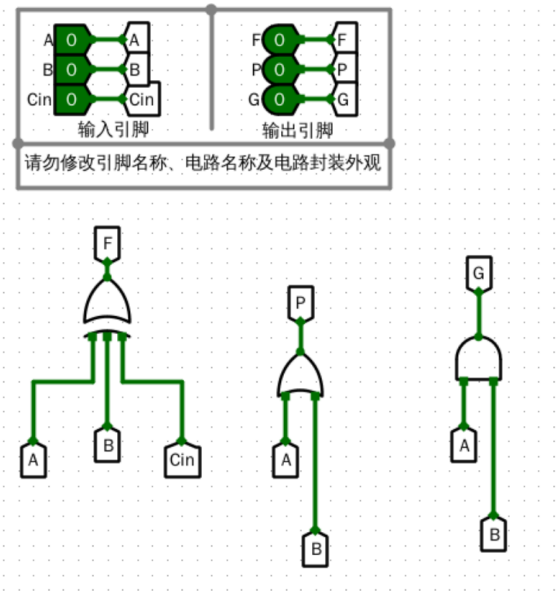

图 4: FA 电路图

1.3 实验数据仿真测试图

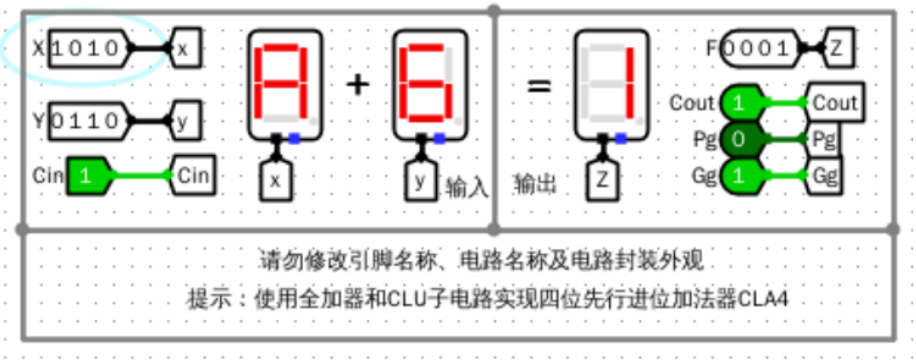

图 5: 仿真测试 1

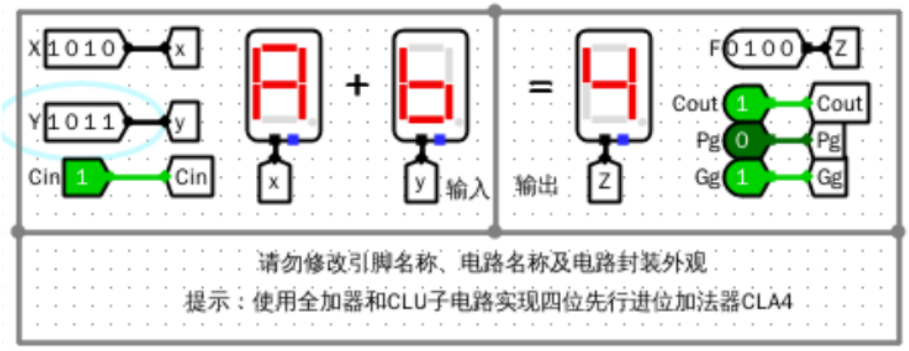

图 6: 仿真测试 2

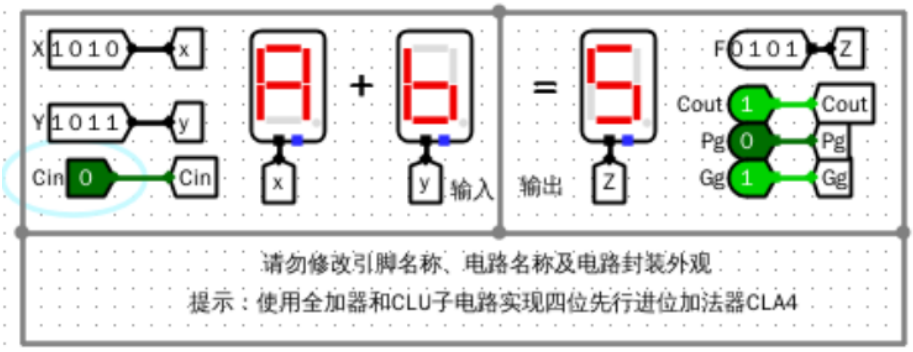

图 7: 仿真测试 3

1.4 错误现象及分析

一遍过了，但在 main 电路中需要注意电路连线的排版和规划，要不然会显着很乱

2 16 位先行进位加法器设计

2.1 实验整体方案设计

本题其实是上一个题 4 位先行进位加法器进行一个组间连接并构成一个整体，即 16 位加法器。这里的输出 C_{16} 代表这个整体向上传递的进位。这里的 CLU4 由上一道题计算单个位变成计算 4 位为一组的进位函数，但总体的逻辑并没有太大的变化。

2.2 实验原理图及电路图

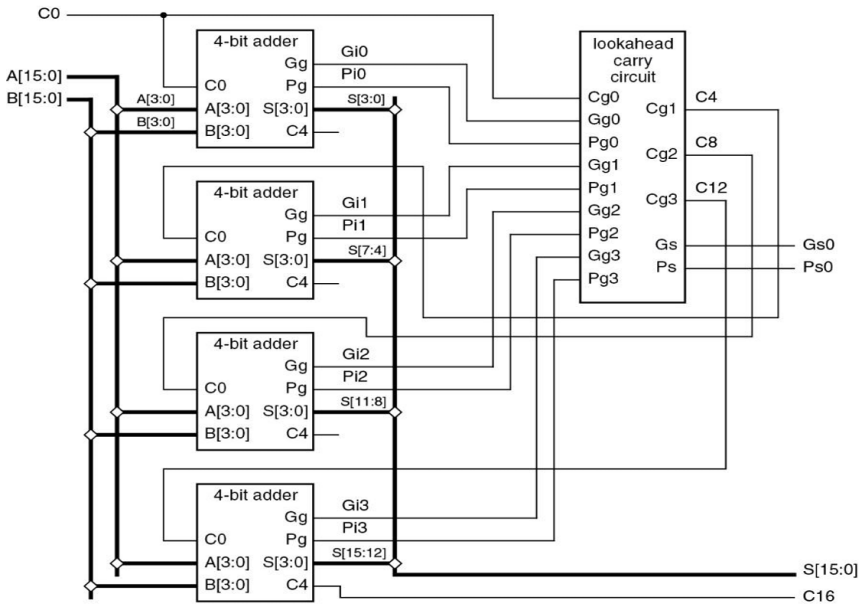

图 8: 原理图

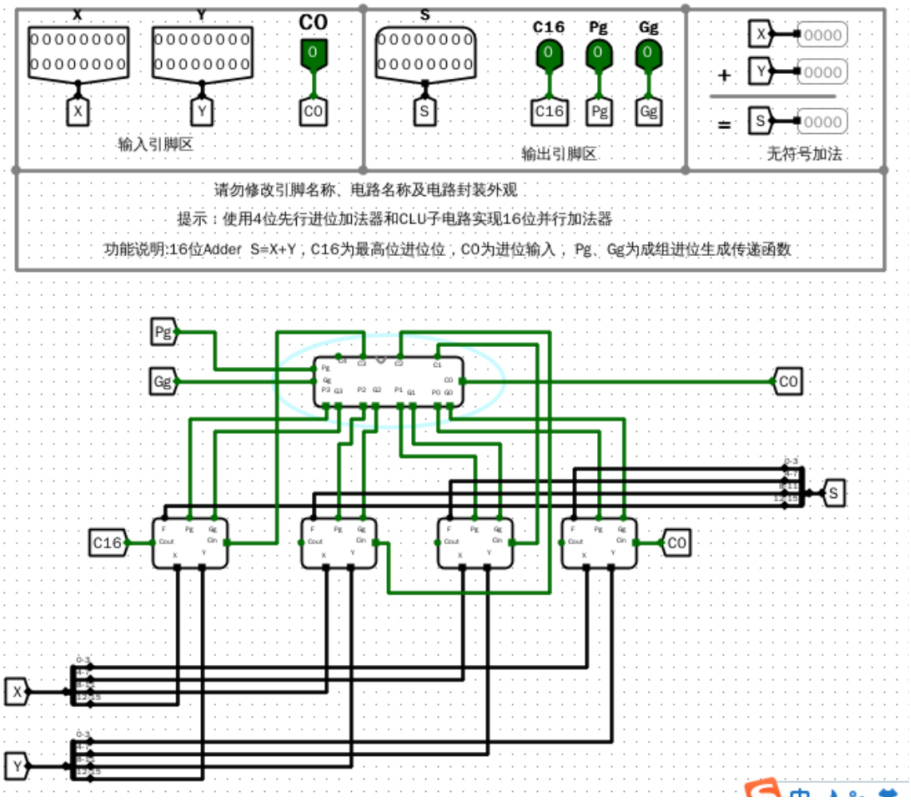

图 9: 电路图

2.3 实验数据仿真测试图

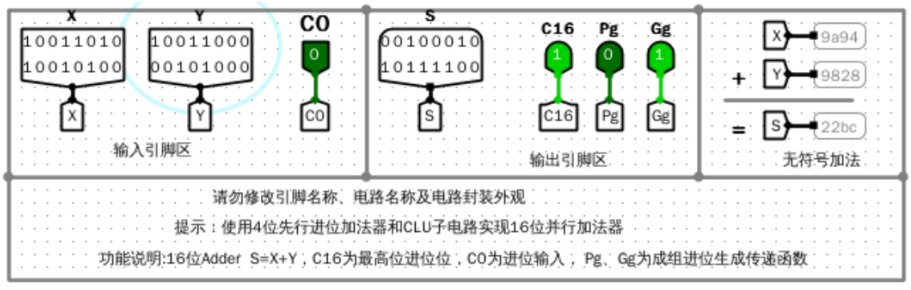

图 10: 仿真测试 1

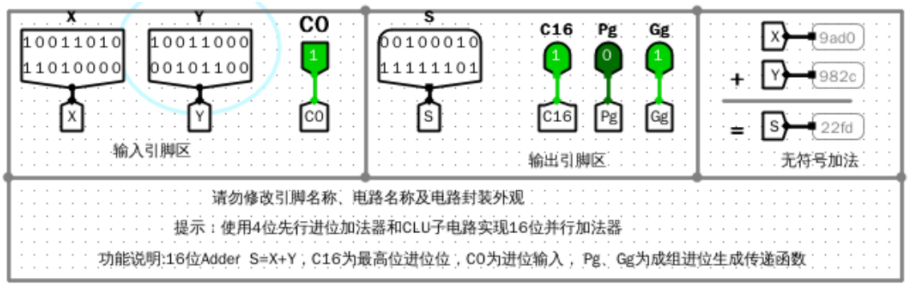

图 11: 仿真测试 2

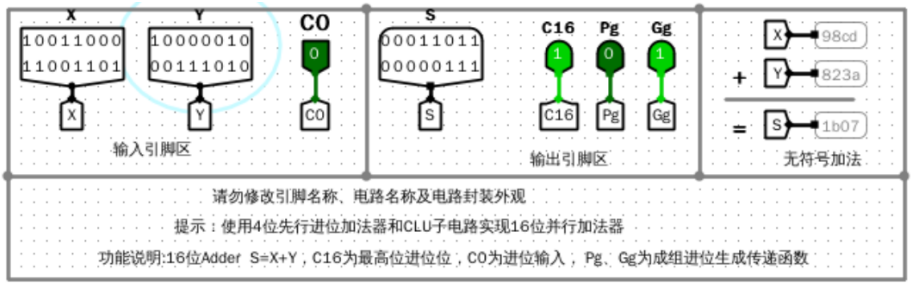

图 12: 仿真测试 3

2.4 错误现象及分析

也是一遍过了，这道题的整体思路和上一道题几乎没有差别，如果有问题可能是因为上一道题的逻辑没理解清楚

3 32 位快速加法器设计

3.1 实验整体方案设计

题目要求 32 位由两个 16 位相连实现，即低位的加法器的输出 C_{16} 作为高位加法器的输入 C_0，下面来讨论本题输出的各种标志位的含义以及实现方式：

	含义	实现方式
CF	进位标识	$Cout \otimes Cin$
OF	溢出标识	$\overline{A_{n-1}} \cdot \overline{B_{n-1}} \cdot F_{n-1} + A_{n-1} \cdot B_{n-1} \cdot \overline{F_{n-1}}$
SF	符号标识	F_{n-1}
ZF	零标识	$\overline{F_{n-1} + F_{n-1} + \cdots + F_0}$

这里 SF 和 ZF 的实现非常好理解，这里就不详细讨论。下面主要讲解 CF 和 OF 的实现：

先看 CF，其表示无符号数运算中的进位或借位状态，其中 $Cout$ 为 32 位加法器的最高位进位输入，Cin 为加法器的进位输入。若为加法，此时 $Cin = 0, CF = Cout$。若为减法，此时 $Cin = 1$，若最高位产生进位，实际表示无借位，所以此时 $CF = \overline{Cout}$。综上，$CF = Cout \otimes Cin$

再看 OF，其表示有符号数的溢出状态，此时如果两个正数相加得到负数 ($A_{n-1} = 0, B_{n-1} = 0, F_{n-1} = 1$) 或两个负数相加得到正数 ($A_{n-1} = 1, B_{n-1} = 1, F_{n-1} = 0$) 会发生溢出，则有 $OF = \overline{A_{n-1}} \cdot \overline{B_{n-1}} \cdot F_{n-1} + A_{n-1} \cdot B_{n-1} \cdot \overline{F_{n-1}}$

3.2 实验原理图及电路图

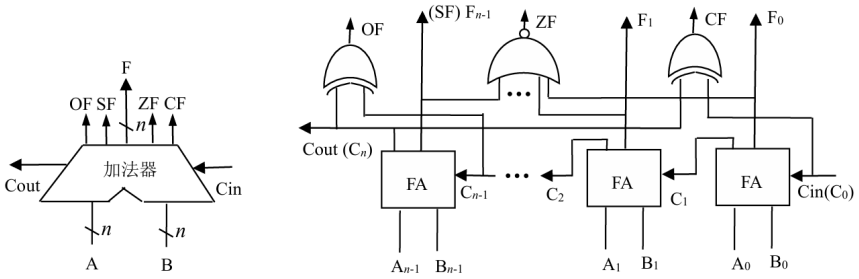

图 13: 原理图

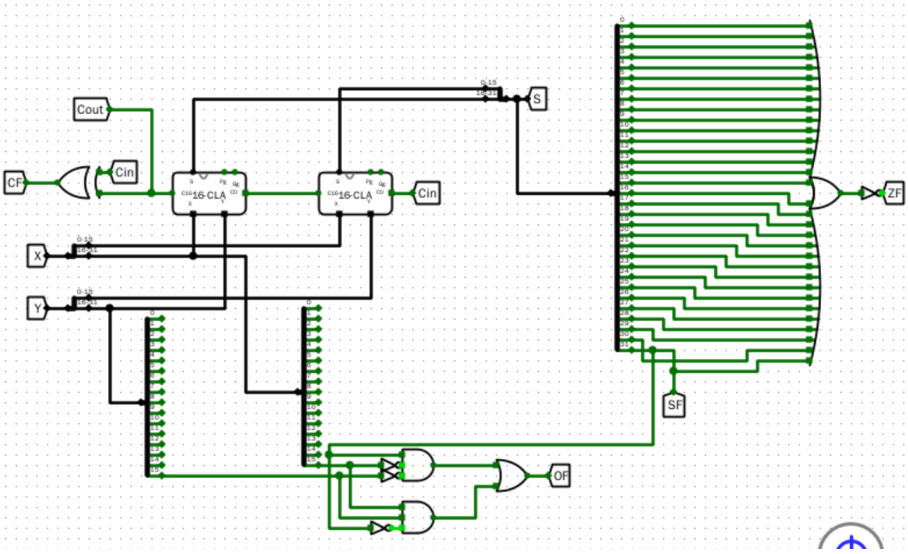

图 14: 电路图

3.3 实验数据仿真测试图

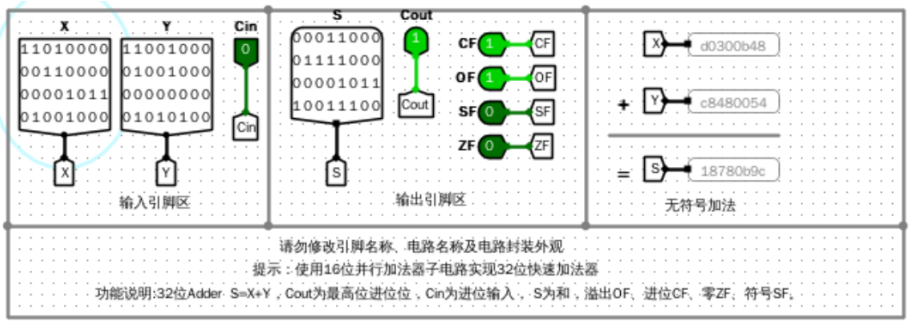

图 15: 仿真测试 1

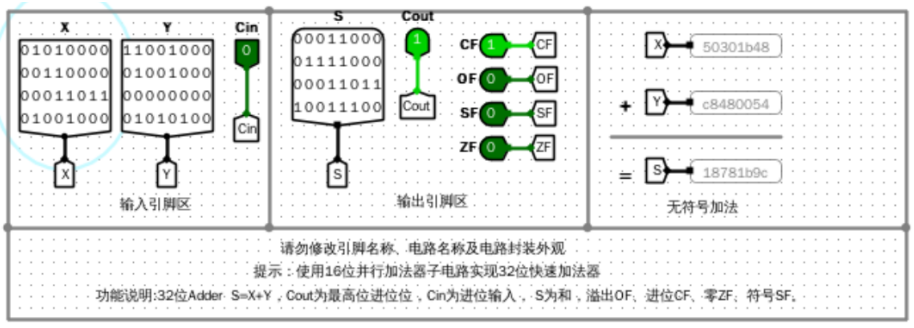

图 16: 仿真测试 2

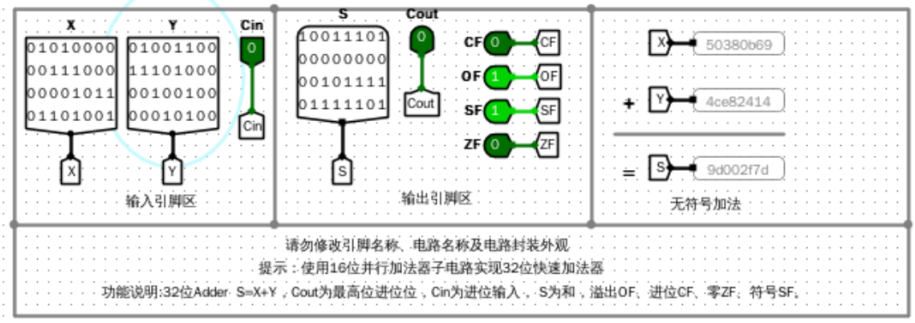

图 17: 仿真测试 3

3.4 错误现象及分析

这里其实 ZF 的信号是要求扇入系数不太过于大的，但本人做实验的时候发现 logisim 或门支持的最大输入数正好是 32，所以并没有采用并联的方式，但实际上这种做法是错误的

4 32 位桶形移位器设计

4.1 实验整体方案设计

本题的实验思路与实验二的第五题十分类似（可以说一模一样），本题也就是实验二思考题的第 4 题，通过指定移动方向、移动类型（算术/逻辑/循环）和移动位数来对 n 位数据输入进行调整，而后输出 n 位调整后的数据。

输入信号 DataIn 和输出信号 DataOut 均为 32 位，移位位数 shamt 为 5 位；选择端 L/R 表示左移和右移控制，置为 1 时表示左移，置为 0 时表示右移；选择端 A/L 表示算术（循环）

移位和逻辑控制移位，置为 1 时表示算术（循环）移位，置为 0 时表示逻辑移位。

移位方向有 3 种不同的情形：左移、右移和不移位。右移时，移入位有 2 种情形：A/L 为 1 时表示算术右移，移入符号位，A/L 为 0 时表示逻辑右移，移入 0。左移时，A/L 为 1 时表示为循环左移，将移出位移入；A/L 为 0 时表示为逻辑左移，移入 0。循环右移可以使用循环左移通过改变移位位数来实现。

4.2 实验原理图及电路图

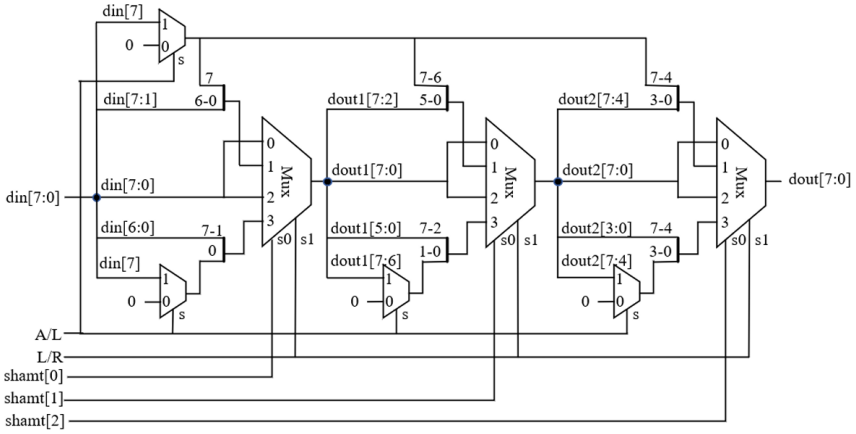

图 18: 原理图

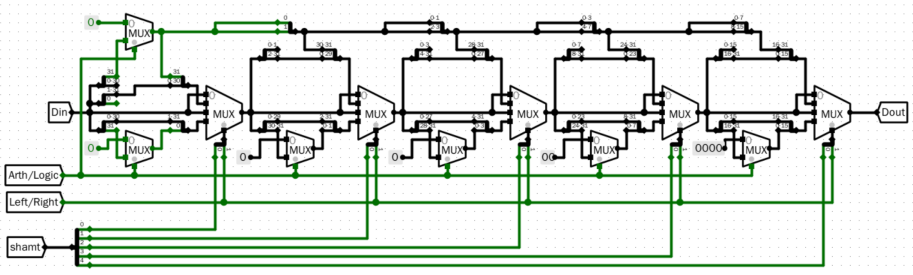

图 19: 电路图

4.3 实验数据仿真测试图

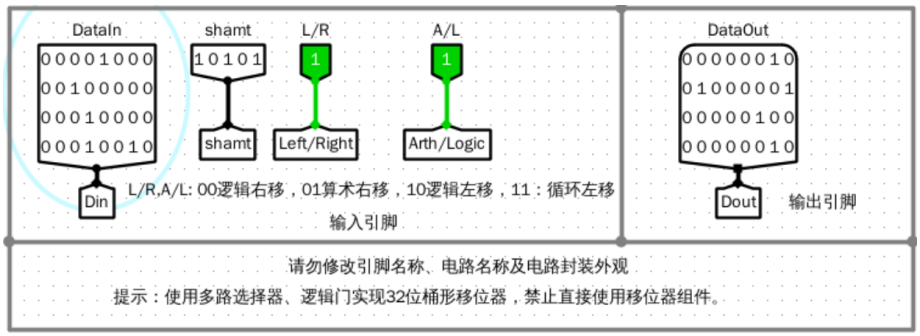

图 20: 仿真测试 1

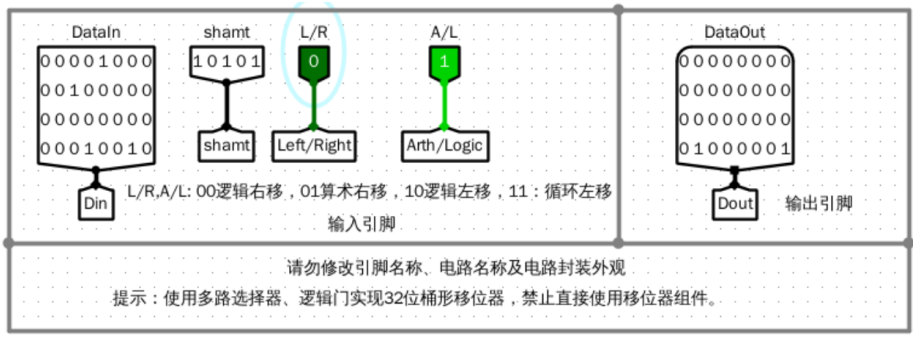

图 21: 仿真测试 2

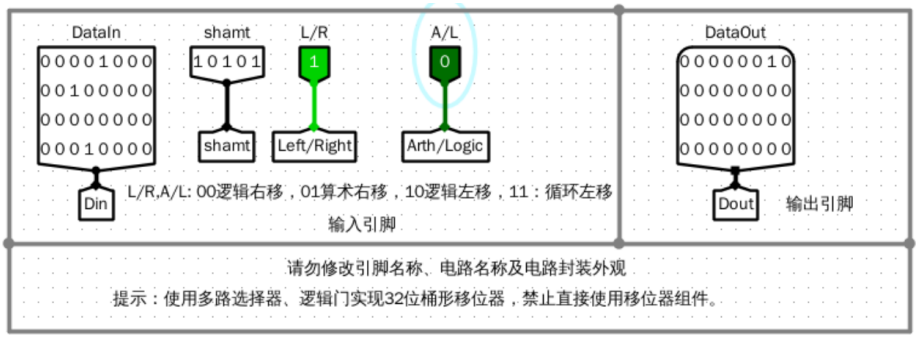

图 22: 仿真测试 3

4.4 错误现象及分析

这道题的连线十分复杂繁琐，可以根据 OJ 中错误的数据针对性地分析，检查每一块电路的输出是否符合预期，不断 debug 直到 OJ 通过

5 32 位 ALU 设计

5.1 实验整体方案设计

本题实际上要求真正掌握前面四道题我们做出来了什么，就是需要应用前面四道题已经实现好的全部封装，并组装成一个完整的 ALU 部件。本题实际要求完成的部件是 ALU 和 ALUctrl，所以这里我们也只探讨这两个的实现思路。

先看 ALUctrl，这个封装实际上是一个控制端的封装，输出一些控制信号传入到主电路 ALU 中，这里给出一个表格来表示各个控制信号的含义及最小项表示方式：

将 SUBctr 符号拓展到 32 位，是为了构造一个减法操作数 B'，满足下面这个公式：

$$A - B = A + \overline{B} + 1$$

在硬件里，我们通常这么构造这个表达式：

$$A + (B \oplus SUBctr) + SUBctr$$

	含义	最小项表示
SUBctr	加减法控制	$\Sigma m(2,3,8)$
SIGctr	比较控制是否带有符号	m_2
ALctr	移位控制	m_{13}
LRctr	移位方向	m_1
Opctr	选择哪种运算结果	Opctr[2]=$\Sigma m(1,2,3,5,13,15)$ Opctr[1]=$\Sigma m(2,3,4,6)$ Opctr[0]=$\Sigma m(4,7,15)$

至于 32 位 ALU 我觉得没有什么需要说的，只需要按照不同的计算方式分别连接控制信号就可以了。

序号	A	B	ALUctr[3:0]	Result	Zero
1	0x80000000	0x7fffffff	0000	0xffffffff	0
2	0x00000001	0x7fffffff	0001	0x80000000	0

3	0x80000000	0x7fffffff	0010	0x00000001	0
4	0x80000000	0x7fffffff	0011	0x00000000	0
5	0x80000000	0x7fffffff	0100	0xffffffff	0
6	0x80000000	0x7fffffff	0101	0x00000001	0
7	0x80000000	0x7fffffff	0110	0xffffffff	0
8	0x80000000	0x7fffffff	0111	0x00000000	0
9	0x80000000	0x7fffffff	1000	0x00000001	0
10	0x80000000	0x7fffffff	1101	0xffffffff	0
11	0x80000000	0x7fffffff	1111	0x7fffffff	0
12	0x80000000	0x80000000	0000	0x00000000	1
13	0x80000000	0x80000000	0010	0x00000000	1
14	0x80000000	0x80000000	0011	0x00000000	1
15	0x80000000	0x80000000	0100	0x00000000	1
16	0x80000000	0x80000000	0101	0x80000000	1
17	0x80000000	0x80000000	0110	0x80000000	1
18	0x80000000	0x80000000	0111	0x80000000	1
19	0x80000000	0x80000000	1000	0x00000000	1
20	0x80000000	0x80000000	1101	0x80000000	1
21	0x80000000	0x80000000	1111	0x80000000	1
22	0x5a5a5a5a	0xa5a5a5a5	0000	0xffffffff	0
23	0x5a5a5a5a	0xa5a5a5a5	0001	0x4b4b4b40	0
24	0x5a5a5a5a	0xa5a5a5a5	0010	0x00000000	0
25	0x5a5a5a5a	0xa5a5a5a5	0011	0x00000001	0
26	0x5a5a5a5a	0xa5a5a5a5	0100	0xffffffff	0
27	0x5a5a5a5a	0xa5a5a5a5	0101	0x02d2d2d2	0
28	0x5a5a5a5a	0xa5a5a5a5	0110	0xffffffff	0
29	0x5a5a5a5a	0xa5a5a5a5	0111	0x00000000	0
30	0x5a5a5a5a	0xa5a5a5a5	1000	0xa4a4a4a5	0
31	0x5a5a5a5a	0xa5a5a5a5	1101	0x02d2d2d2	0
32	0x5a5a5a5a	0xa5a5a5a5	1111	0xa5a5a5a5	0

5.2 实验原理图及电路图

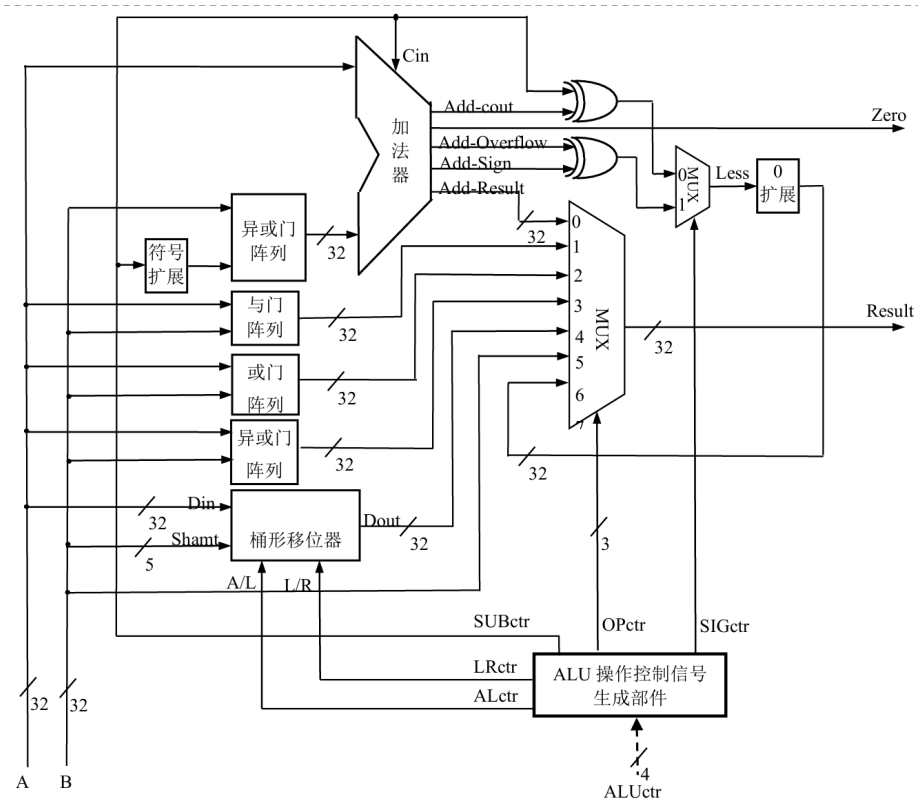

图 23: 原理图

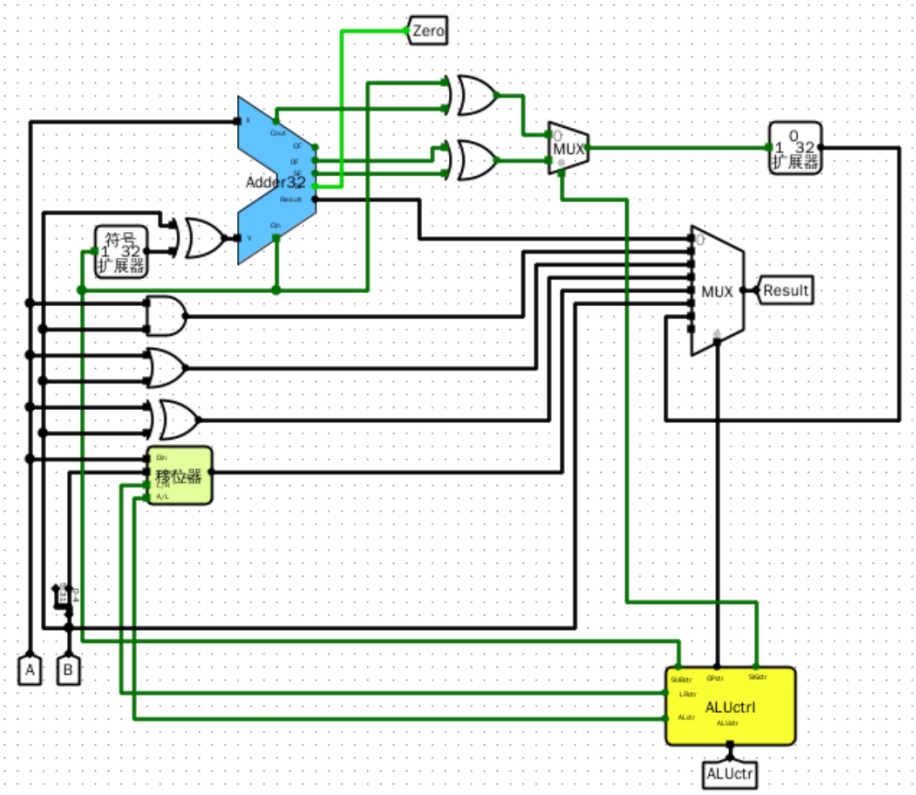
Zero
Adder32
MUX
0
1 32
扩展器
符号
1 32
扩展器
MUX
Result
移位器
A
B
ALUctrl
ALUctr

图 24: ALU 电路图

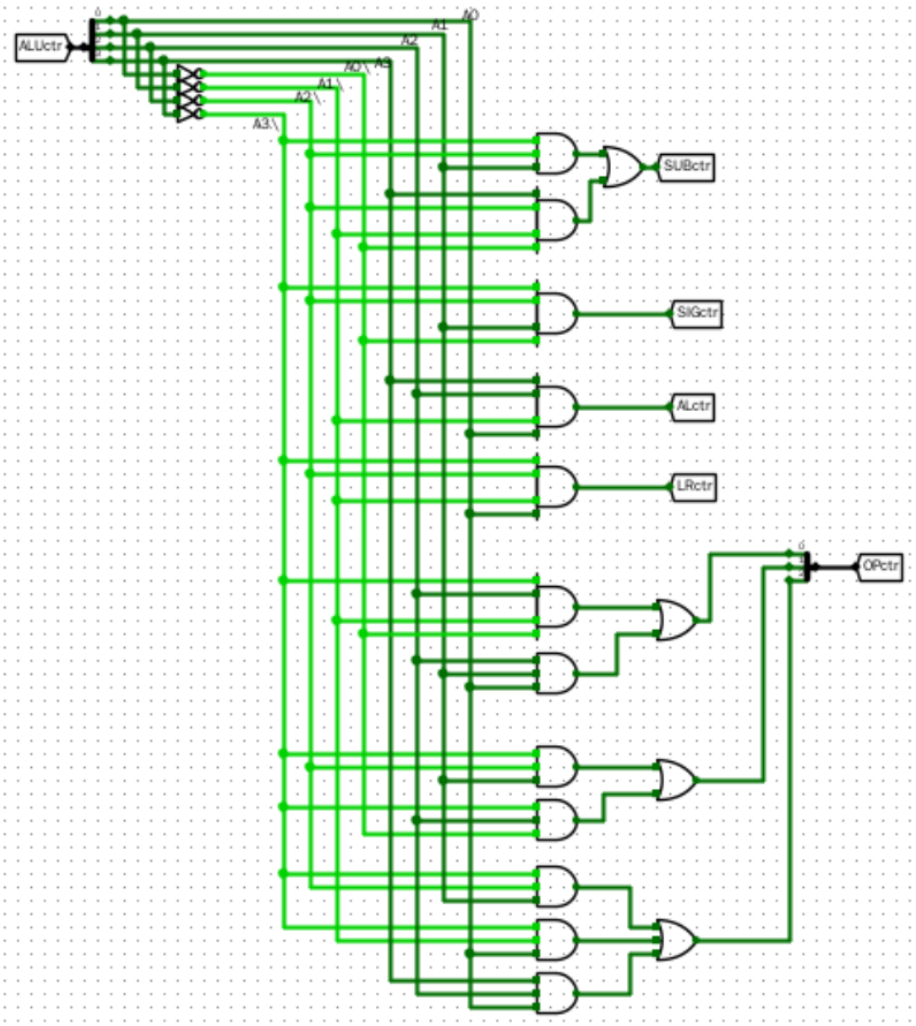

图 25: ALUctrl 电路图

5.3 实验数据仿真测试图

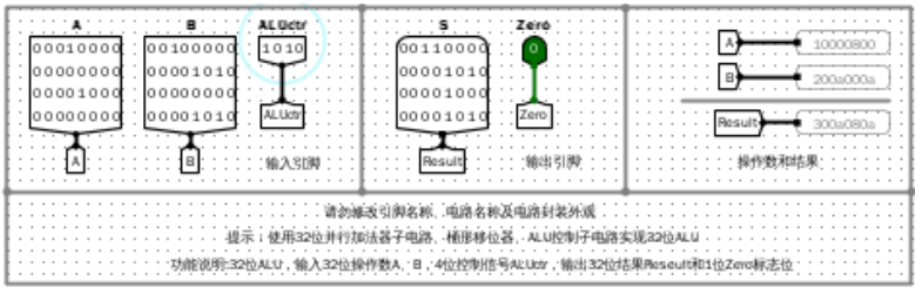

图 26: 仿真测试 1

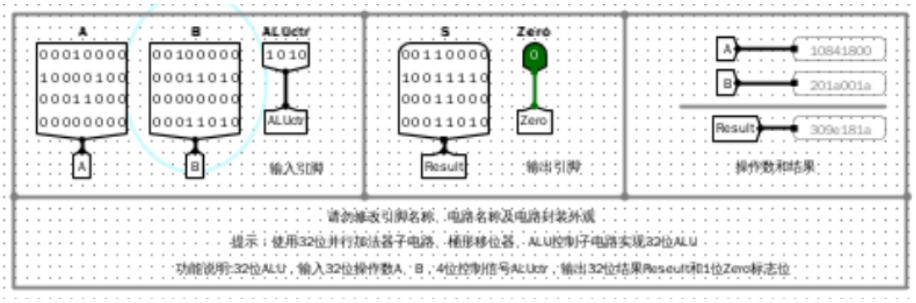

图 27: 仿真测试 2

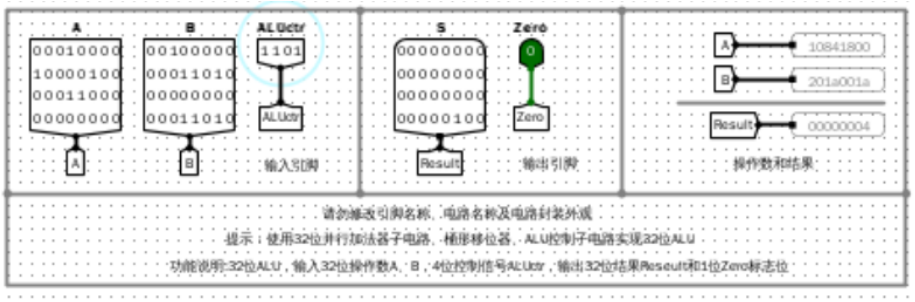

图 28: 仿真测试 3

5.4 错误现象及分析

利用卡诺图化简最小项时容易算错，需要注意。

6 思考题

6.1 通过查找资料，实现一种 64 位二进制快速加法器的设计。

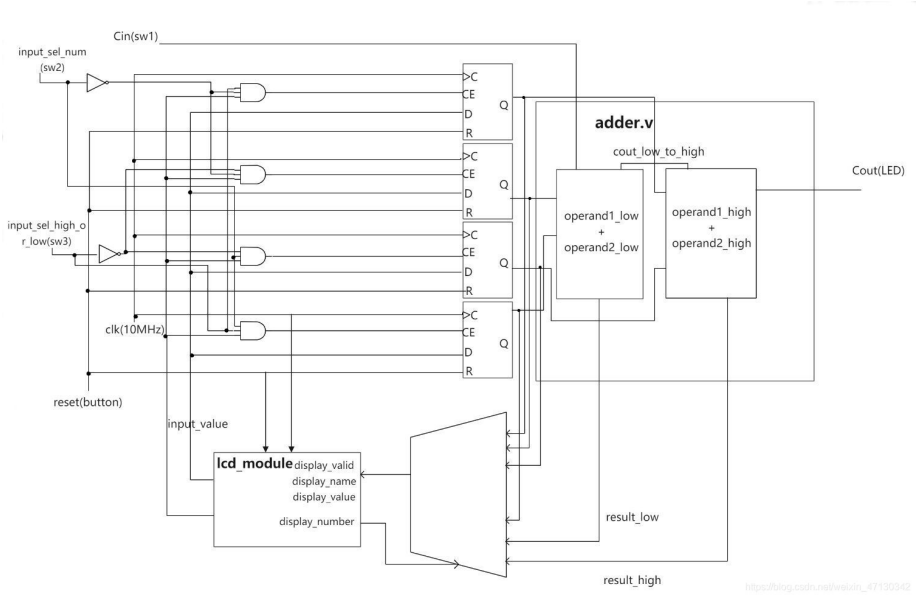

图 29: 64 位加法器

本电路实现了一个带有输入控制、时钟同步、进位传递和结果显示的 **64 位加法器系统**，其核心由两个 32 位加法器组成，分别负责高 32 位和低 32 位的加法操作。该电路可通过开关控制输入的数值段，使用 D 触发器实现输入保持，并通过 LED 和 LCD 显示模块输出结果。

输入控制部分 电路提供两个控制信号：

- **input_sel_num (sw2)**：用于选择操作数编号（操作数 A 或 B）。
- **input_sel_high_or_low (sw3)**：用于选择输入的是高 32 位还是低 32 位。

这两个信号经由若干 **反相器（NOT）和与门（AND）组合**，生成 4 组互斥的时钟使能（CE）信号，分别控制以下 4 个寄存器：

- operand1_low
- operand1_high

- operand2_low
- operand2_high

所有寄存器都使用 **D 触发器**，只有在 CE 信号有效的时钟上升沿到来时才更新数据。按钮 **reset** 控制所有寄存器复位为 0。

加法器逻辑部分 加法器模块分为两个级联的子模块：

- **低位加法器（adder_low）**：计算 operand1_low + operand2_low + Cin，输出结果为 `result_low`，并产生向高位传递的进位信号 `cout_low_to_high`。
- **高位加法器(adder_high)**:计算 operand1_high+operand2_high+cout_low_to_high，输出结果为 `result_high`。

最终组合的 64 位加法结果为：

$$\text{result} = \text{result\_high} \parallel \text{result\_low}$$

最高位的进位输出（Cout）连接至 LED 指示器，用于观察是否发生了无符号溢出。

显示模块（LCD） `lcd_module` 接收以下输入：

- **input_sel_num, input_sel_high_or_low**：用于判断当前正在输入的是哪一部分寄存器。
- **input_value**：当前用户通过开关输入的数据。

LCD 会根据这些信息，实时显示：

- 当前正在输入的寄存器名称（如 A 高位、B 低位等）
- 当前输入的数据值
- 当前输入编号（辅助定位）

显示由 `display_valid` 信号控制，确保显示内容只在有效输入时更新。

设计亮点　该设计在基本的 64 位加法基础上增加了许多工程实践中的细节：

- **使用时钟控制的数据加载机制**：利用 D 触发器和时钟使能，确保输入稳定。
- **将低位加法的进位传给高位**，符合典型 64 位加法器结构。
- **LCD 模块可视化输入状态**：增强用户交互性，方便调试。
- **LED 实时显示进位状态**：快速判定是否发生溢出。

6.2 将实验 3 中的快速乘法器设计电路扩展到 32 位无符号数相乘，并探讨如何将该乘法器融合到实验中的 ALU 电路来实现乘法运算。

实验 3 中实现的是 4 位无符号数乘法器，要拓展到 32 位的话，ChatGpt 给出的解决方案是利用并行方式一次性生成 32 个部分积，而后用大量加法器压缩求和，最后再用加法器加和得到结果。

其实 ChatGpt 给出的思路可以看成把两个数的乘法进行拆分，两个数都类似地分为高位和低位，并两两相乘得到 4 个部分积，然后再对这些部分积进行适当地移位并加和得到整体的结果。例如，两个高位相乘就需要左移 2×16 位。当然我们已经实现的是 4 位，所以操作会更加繁琐，但大致的思路不变。

接下来看看如何将乘法器融合到 ALU 电路中，其实这也十分地容易，只需要在 ALUctrl 中的 Opctrl 信号多给一位用于乘法就可以了。并在左侧内置一个快速乘法器实现好的封装即可。

6.3 假设在 RV32I 中新增一条指令，导致在 ALU 中增加一个新运算操作，试修改 ALU 设计电路，并通过测试数据进行验证。

因为新增了一条指令，所以需要修改 ALU 控制信号的编码，假设新加入指令 JNTM 用来获取“加法后取高 32 位”，那么只需要为其添加一个新的编码比如“1000”并产生一个控制信号，利用已有的加法器来求其高 32 位并添加到最后的多路选择器中。

6.4 分析比较运算使用独立的比较器和使用减法运算通过标志位来实现两种方法的特性。

独立比较器的思路一般是从高位开始，一旦有一位不相等立刻输出结果，所以其速度非常快，但其需要独立设计一整套的电路封装。而对于减法运算标志位，其可以直接复用已经实现好的加法器封装，对整体电路的复杂度十分友好，其增加的复杂度也只是添加了一个标志位。