**计算机组成原理**

**--实验报告**

**专业班级： 人工智能2204**

**学 号： U202215150**

**姓 名： 王子旭**

**实验名称： 运算器组成实验**

**实验时间： 2024年5月16日**

**一、实验目的**

1. 掌握一位全加器和多位可控加减法电路的实现逻辑；

2. 掌握快速加法器中先行进位的原理并设计四位快速加法器；

3. 理解成组进位产生函数、传递函数的概念，设计多位快速加法器；

4. 掌握阵列乘法器的实现原理；

5. 掌握原码、补码一位乘法运算的基本原理；

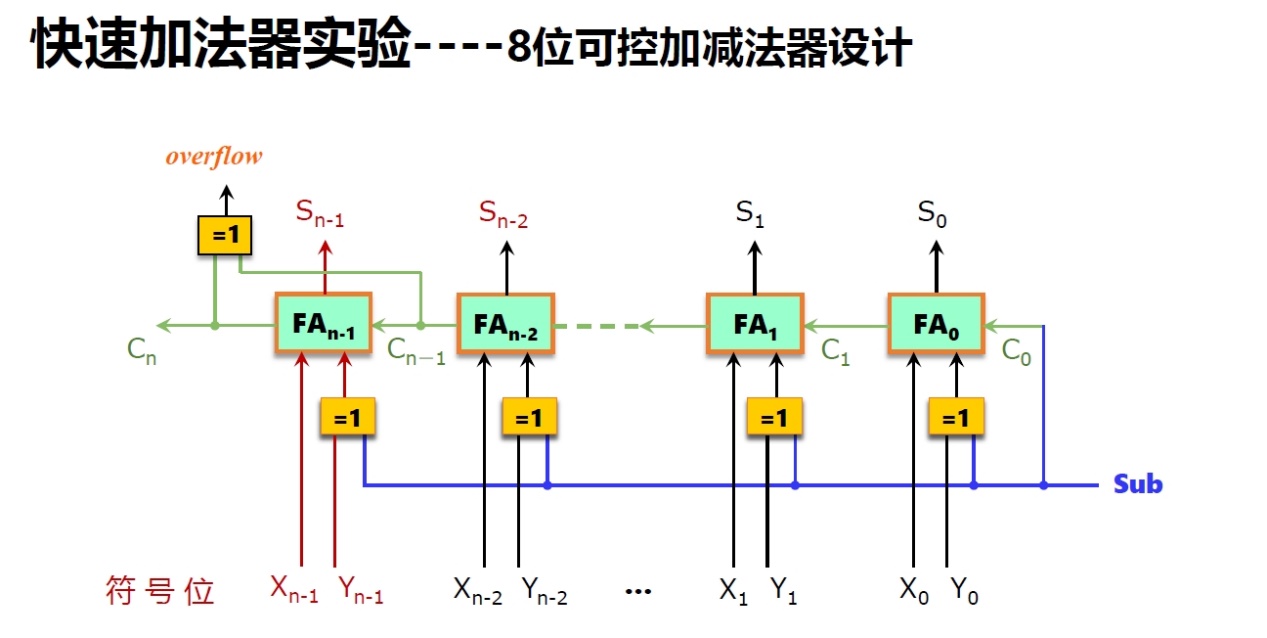
6. 理解算术逻辑单元ALU的基本构成；

1. **实验内容**

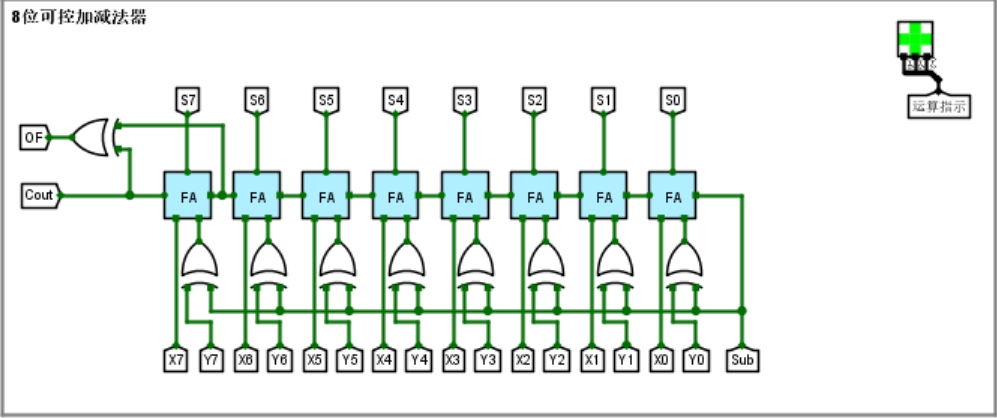
**加法器实验：**

**2.1、八位可控加减法器**

设计基本思想：其实多位可控加减法器的核心思想还是一位全加器，不同的是多个进位信号，如何对进位信号处理是重点。采用串行进位方法，前位的进位信号直接作为输入送给下一位。加法的实现很简单，减法的实现需要增加一个控制信号输入sub，通过异或门实现加减法。原理图如下：



**2.1.1电路图**

****

**2.1.2、电路设计思路**

根据多位可控加减法器的原理图，设计8位可控加减法器。按照图示方法接入电路与sub控制信号做异或运算，sub为0时输入本身，实现加法运算；当sub为1时输入其反码，实现减法运算。OF输出的是减法的溢出检测。这样从右到左进位信号一次被送达，实现8位可控加减法电路。

**2.2、 四位先行进位与四位快速加法器**

**2.2.1、四位先行进位**

表达式：





4位先行进位逻辑表达式如下：



实验原理图如下：

图示

描述已自动生成

**2.2.1.1、 4位先行进位实验电路图**

**图示, 示意图

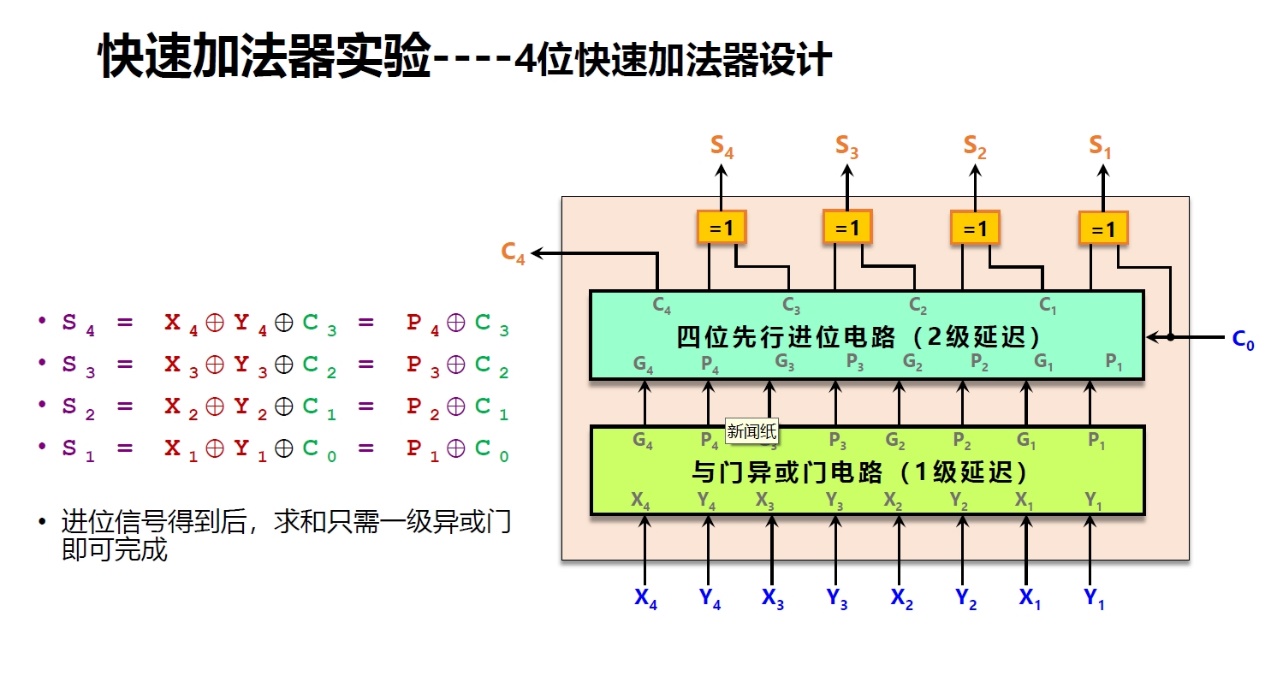
描述已自动生成**

**2.2.1.2、 电路图设计思路**

根据上述的逻辑表达式设计出的实现电路,的设计是为了下面的4位快速加法器,。

**2.2.2、四位快速加法器**

有了四位先行进位电路，实现四位快速加法器很简单。因为软件里面已经把四位快速加法器封装好了。所以在74182的基础上，我只要对输入和最后的输出进行逻辑变换即可。实现的电路原理图如下：



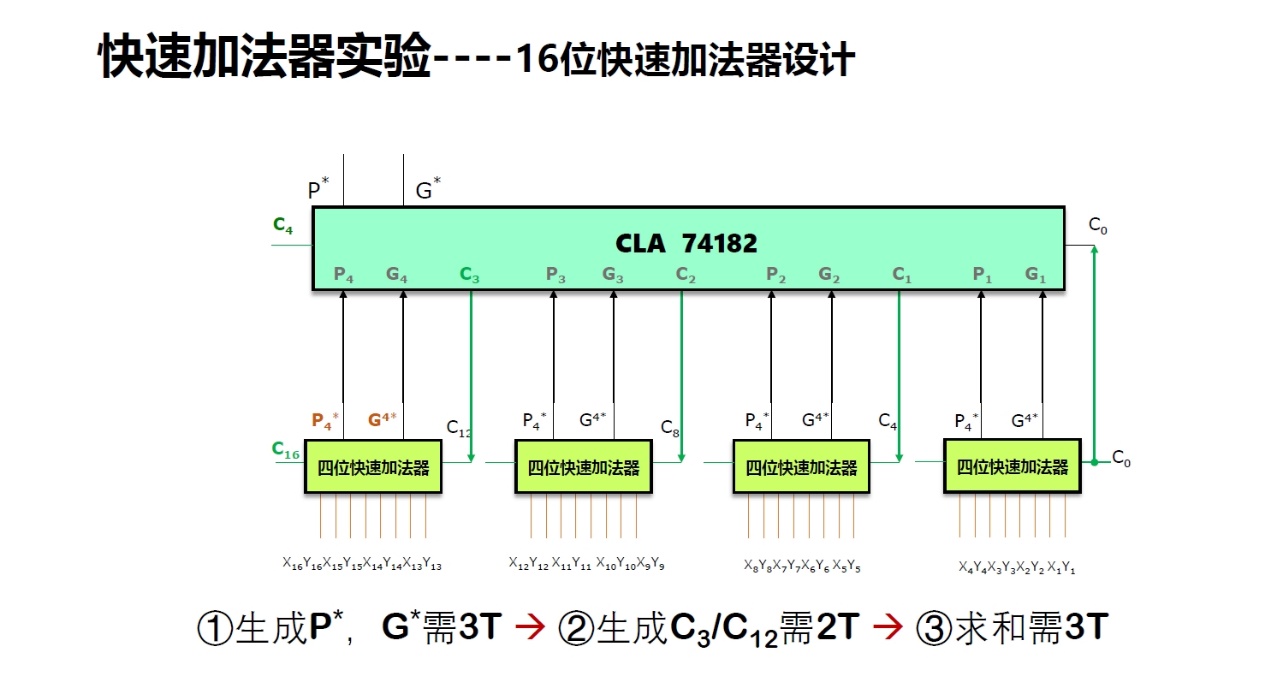
**2.2.2.1电路实现**

图片包含 图示

描述已自动生成

**2.3、 多位快速加法器**

本实验中的16位和32位快速加法器都是在前面的4位快速加法器基础上做的，把前面的4为快速加法器封装为了模块，做16位快速加法器需要用4个4位模块，做32位快速加法器需要用8个4位模块。多位快速加法器是组内并行、组间并行，4位快速加法器输出的作为上层先行进位电路的输入，并由上层进位电路产生下层需要的组间进位信号。实验原理图如下：

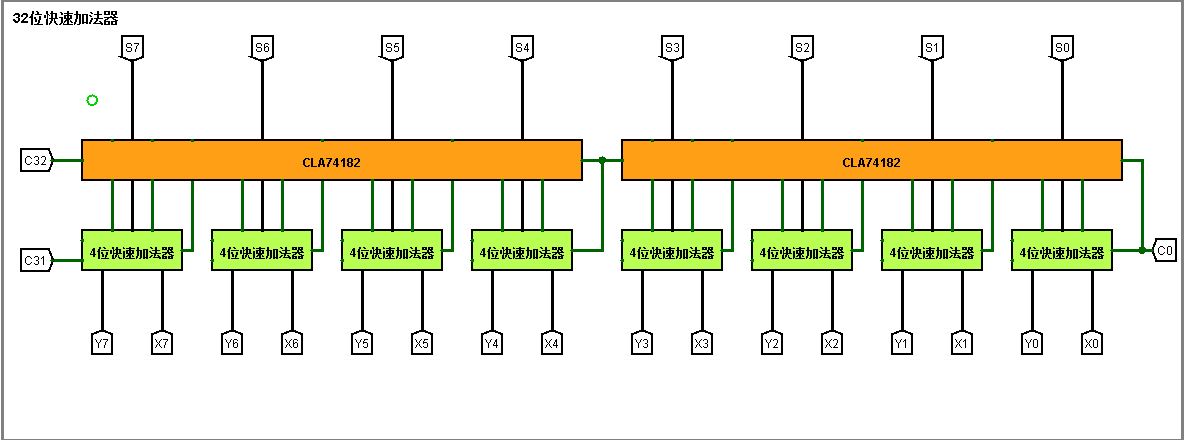


**2.3.1、实验电路图**

16位快速加法器电路设计：

图示

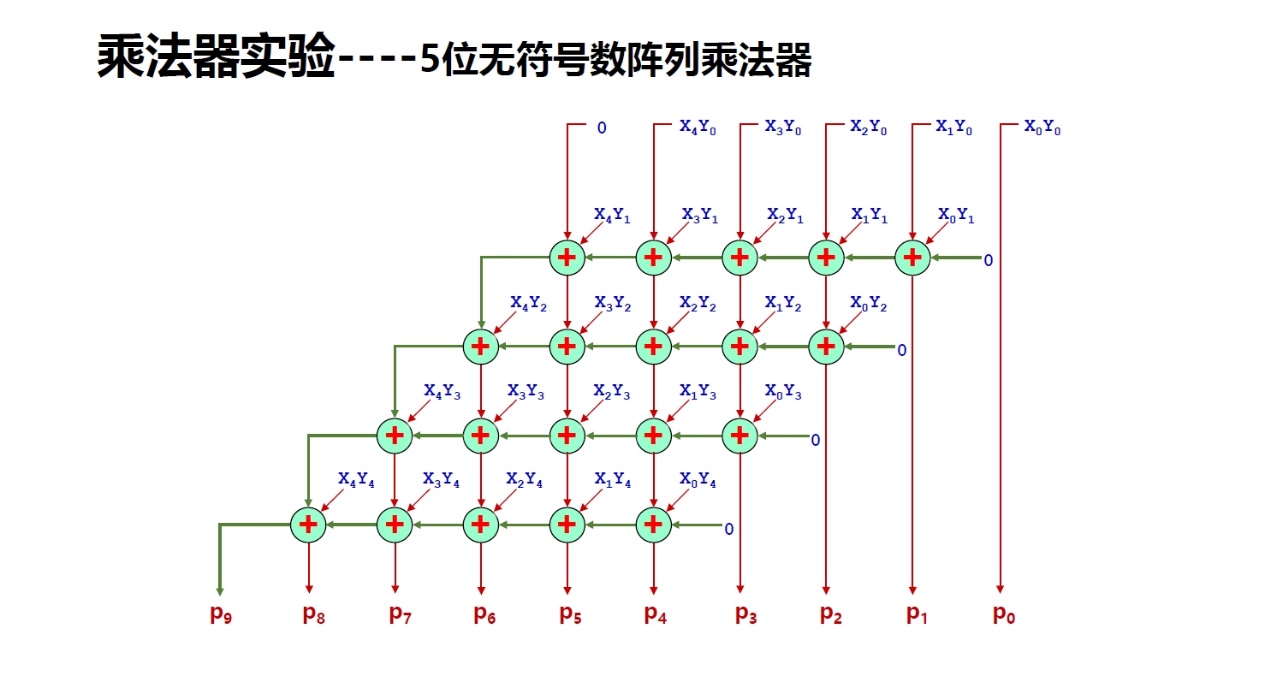
描述已自动生成32位快速加法器电路设计：



**阵列乘法器实验**

**2.4、五位阵列乘法器**

两个5位无符号数计算乘积，首先计算25个位乘积项，二进制一位乘法可以用与门逻辑实现，所以需要25个与门。这里我用的斜向进位阵列乘法器，用全加器相连后构成，每行4个全加器，共5行。斜向进位消除了行中的进位依赖而且可以并发运行；但是最后一行的全加器是横向传递的，所以最后一行是串行运算。实验原理图如下所示：



**2.4.1、5位无符号斜向进位阵列乘法器电路图**

**图示, 日程表

描述已自动生成**

**2.4.2、时间延迟分析**

5位无符号斜向进位阵列乘法器的延时：

**2.5、6位补码阵列乘法器**

对于补码的运算来说，关键是将符号位与数值位区分开。对于正数，直接用数值位进行相应的运算，然后前面在加上符号位即可；对于负数，需要特殊处理这个数值位，然后根据相应的逻辑再转化即可。实验原理图如下：

图示

描述已自动生成

在6位补码阵列乘法器中利用5位无符号乘法器以及求反器实现补码阵列乘法器。这里用到了求反器，如图所示：

**2.5.1、6位补码阵列乘法器电路图**

图示

描述已自动生成

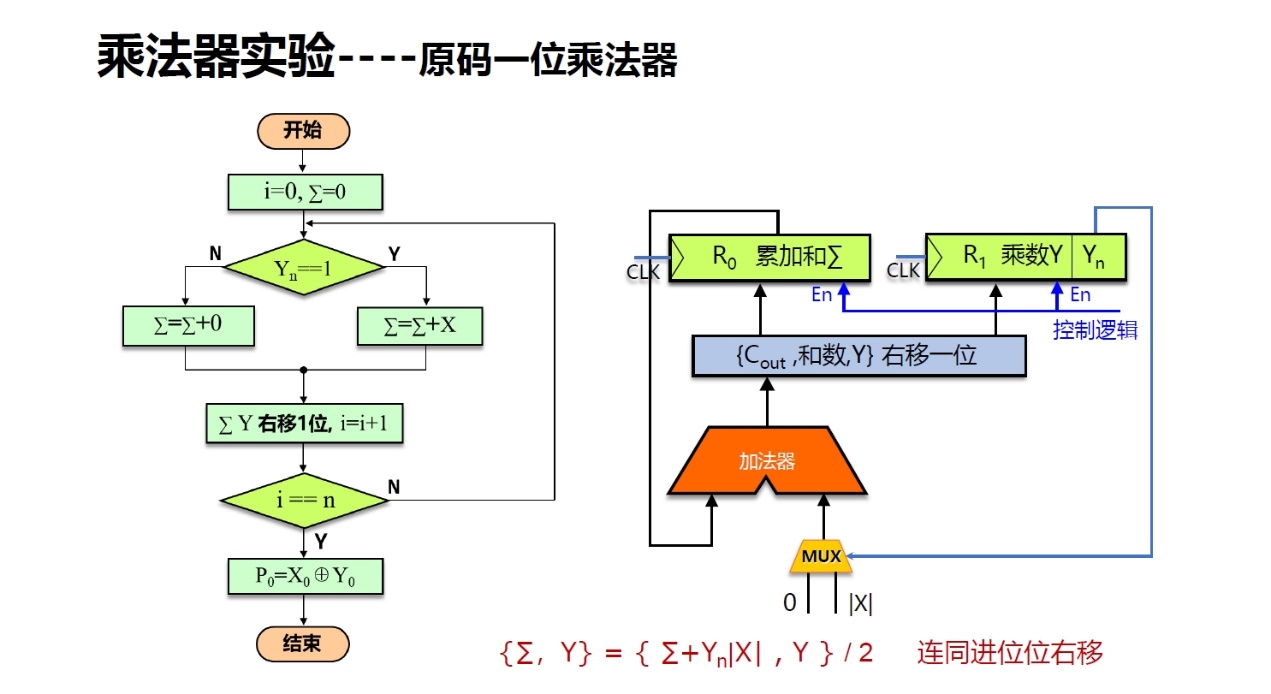
**2.5.2、设计思路**

电路分为两个部分，上面的补码转化电路、下面的乘积电路。上面的左右两个部分很明显是对输入做求补运算的电路，用到了选择器和求反器，其原理就是补码怎么转换的原理。下面的乘法电路首先对经过求补运算的进行5位阵列乘法器运算，随后线分两路，一路维持本身值、另一路经过求反器送给多路转接器，并且其选择端是符号位经过异或门的信号(异号时选择1，同号时选择0)，并且该信号作为输出的最高位。这样就根据阵列乘法器和高位异或的输出组合得到了6位补码阵列乘法器的结果。

**一位乘法器实验**

**2.7原码一位乘法器**

首先是原码1位乘法的计算方法，原码数值部分和真值相同，可单独计算符号位，具体的流程图如下图所示：

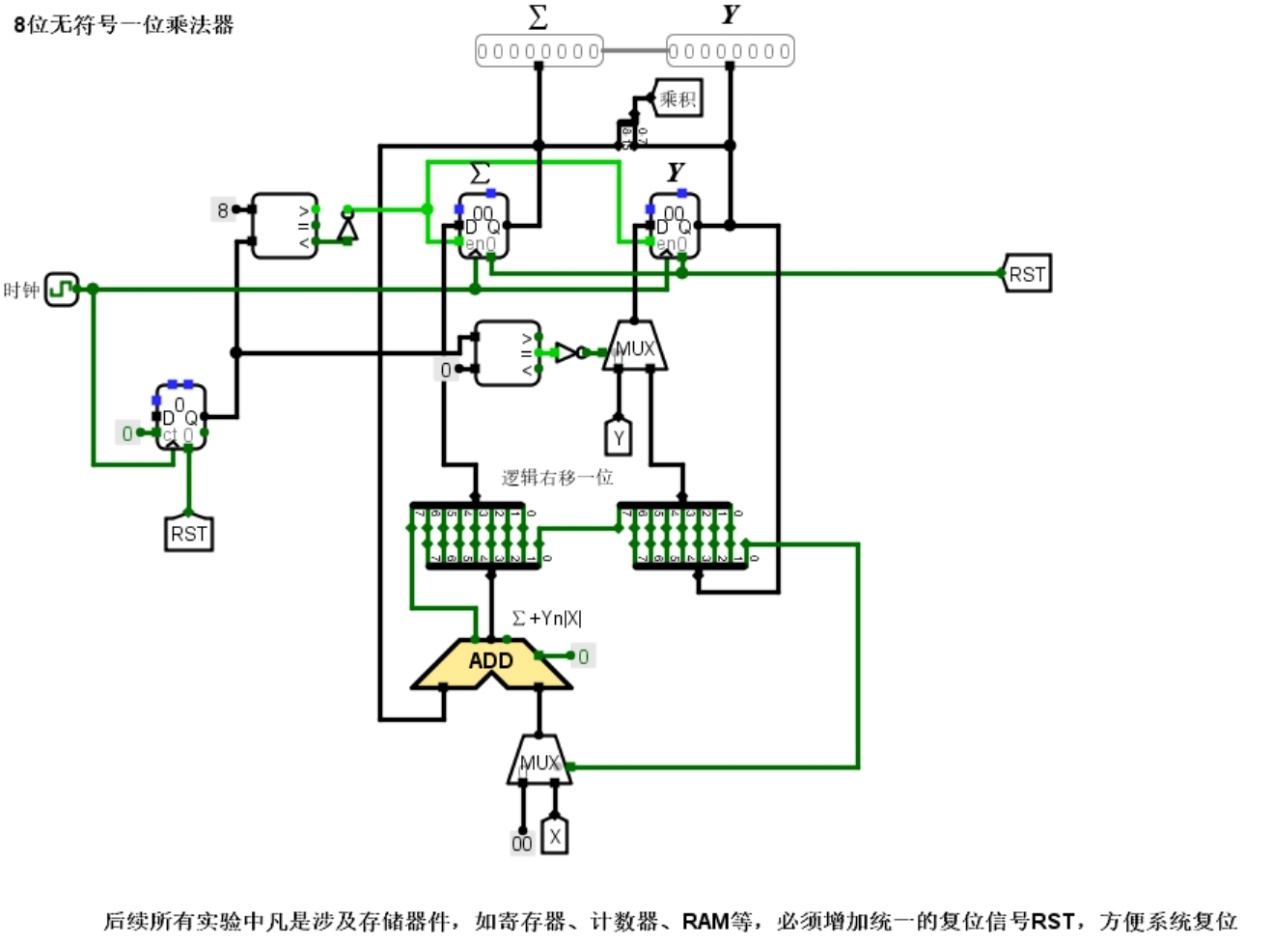


由原码一位乘法可知，乘数依次从最低位乘以被乘数，然后和部分积进行累加。乘数可以右移一位，将最低位丢掉。在进行累加时，因为每次做完乘法都是高一位的，所以把部分积右移一位，这样就可以实现对齐。另外，丢掉的那一位可以存在乘数寄存器的最高位，这样每次右移丢掉的一位都在最前列。在运行8次后，第一次丢掉的刚好到达最低位。

乘法部分：如果 = 0，则结果为0，如果 = 1，则结果为。可知乘法可以用数据选择器实现。对运算次数进行控制，在第0次运算时，把Y载入乘数寄存器中。在第8次运算后停止运算。

硬件逻辑结构如下图所示，寄存器存放部分积，寄存器存放乘数，并且最低位为判断位；寄存器存放被乘数。，中的乘数随时钟驱动右移出寄存器。原理逻辑图见上图右

**2.7.1、8位无符号一位乘法器电路图**



**2.7.2、电路设计思路**

设计思路主要是参考实验指导书上的原理图。初始化寄存器的值为0，首先把送给对应的寄存器，用电路实现部分积。然后把加法器运算结果逻辑右移1位送出，将移位后的数据载入寄存器进行下次运算，最后根据时钟计数器的值判断运算的结束与否。

**2.8、补码一位乘法器**

Booth补码1位乘法的流程图如下所示：

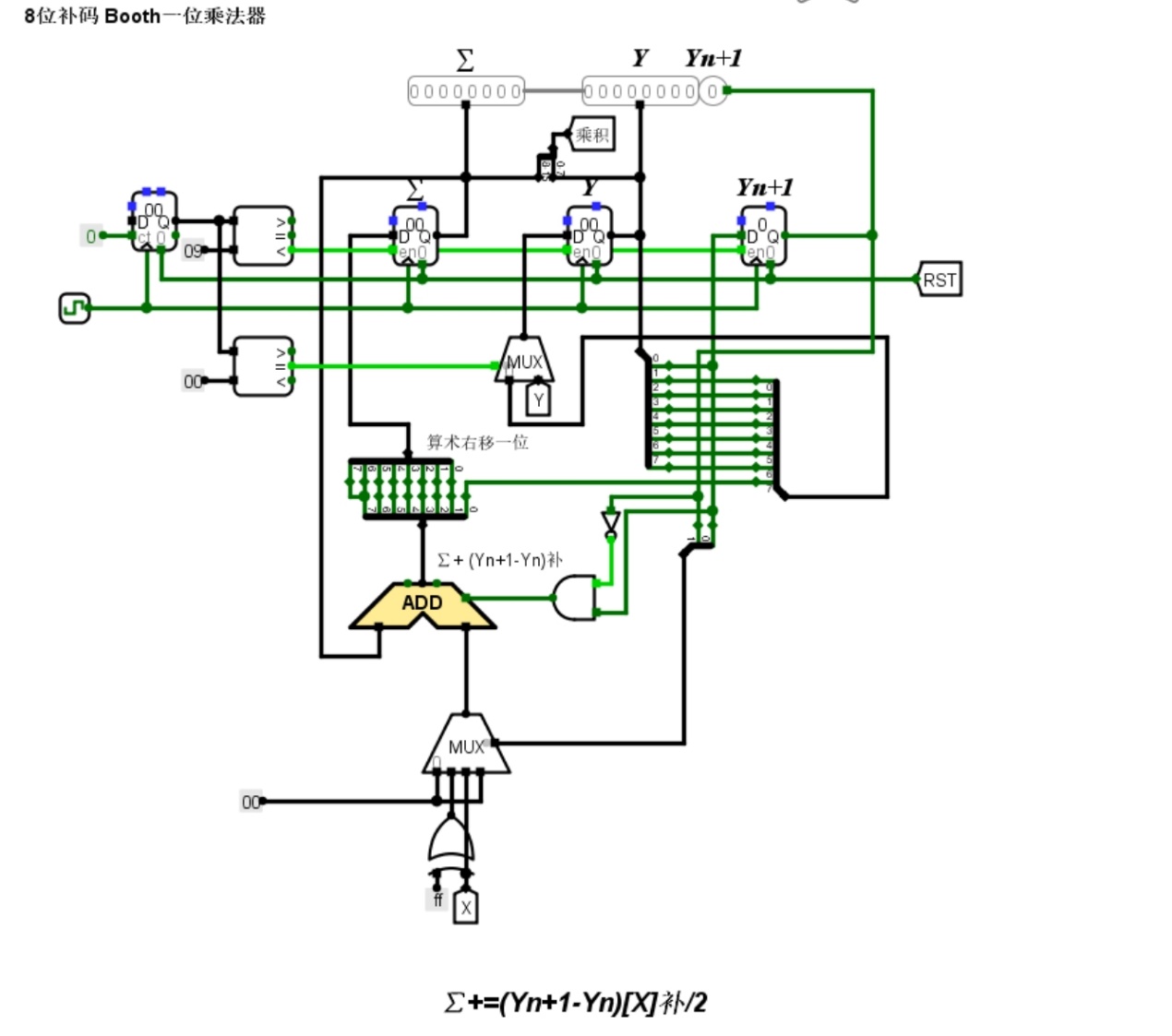
图示

描述已自动生成

与原码1位乘法不同,booth1位乘法中乘数采用双符号位参加运算,符号位也参与运算。利用∑存放部分积，i为循环计数器,初始值为零,部分积累加公式为,根据与的差值决定累加运算的参数是0还是或者是。运算完毕后,先判断循环次数是否达到,如未达到则部分积∑右移1位,右移1位,然后继续循环累加,当乘数符号位参与运算后,运算结束,得到的乘积存放在∑和中,无须单独计算符号位。如果数值部分为n位,需要进行n+1次加法运算和n次移位操作。

补码1位乘法的硬件逻辑结构如上图所示,图中寄存器存放部分积∑,寄存器存放乘数以及扩展位,为判断位;寄存器存放被乘数的补码;加法器实现部分积的累加。控制电路负责移位控制和循环计数，受时钟驱动,每运算一次,加法器运算结果与寄存器的值一起算术右移1位后产生的新值载入和寄存器中,当运算结束时,乘积的高n位数据在中,低n位在中,中原来的乘数在右移过程中逐位移出寄存器。

**2.8.1、8位booth一位乘法器电路图**



**ALU实验**

**2.9算术逻辑单元ALU**

ALU是根据OP字段来决定对应的功能，芯片引脚与对应关系和如下：



表格

描述已自动生成

**2.9.1、算术逻辑单元ALU电路**

图示, 示意图

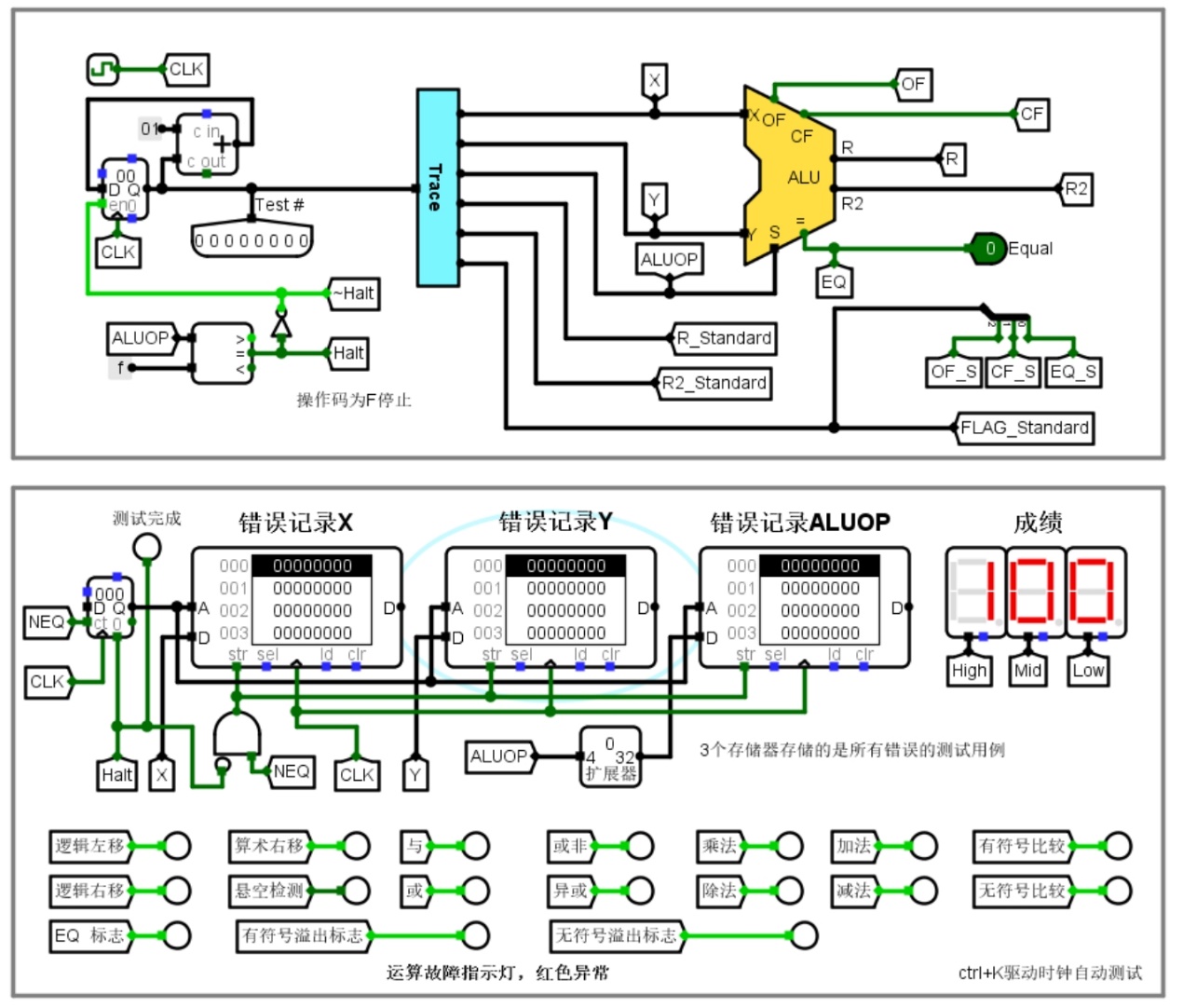
描述已自动生成

**2.9.2、电路设计思路**

ALU算法单元并行进行各种算法单元运算，以算术操作符做为选择信号，选择最终的输出形式，所以，只需分别设计各操作并打好标签，用MUX控制输出即可完成ALU算法单元。这里分成了多个模块。

1. **实验结果**

在ALU自动测试中按下ctrl+k运行结果如下：



每个时钟周期显示数字都是100，说明前面电路设计正确。

1. **实验中的bug和解决方案**

1、在设计原码、补码一位乘法器过程中，第一个周期应该是将Y送入寄存器，一开始我没有想到合适的方法，后来发现可以根据计数器的状态来判断是否处于第1个周期，借助一个数据选择器即可实现初态的设置

2、在最后ALU自动测试实验一开始发现分数并不是100，根据下方的故障指示灯，发现有几项故障，到电路中排查电路逻辑并非发现问题，反复检查之后发现隧道标签与实际上的不一致，导致数据不能正确送入测试电路，才产生了故障。