华东师范大学软件工程学院实践报告

课程名称：计算机组成与实践	**年级**：**2023 级**	**上机实践成绩**：
指导教师：谷守珍	**姓名**：张建夫	**上机实践日期**：4/22~4/29
实践编号： 实验1	**学号**：10235101477	**上机实践时间**：4 学时

一、 实验名称

快速加法器实验

二、 实验目的

- 验证串行加法器逻辑实现
  - 能设计8位可控加减法电路
- 掌握快速加法器逻辑实现
  - 能设计4位先行进位电路
  - 能设计4位快速加法器
- 理解组内先行，组间先行的基本原理
  - 利用4位快速加法器构建32位快速加法器
  - 能分析相关电路延迟

三、 实验内容

- 设计8位可控加减法设计
- 设计4位可级联先行进位电路CLA74182
- 设计4位快速加法器
- 利用4位快速加法器构建32位快速加法器

四、 实验原理

（1） 全加器原理：

首先设计一位的全加器，全加器有三个输入（操作数a，操作数b以及进位cin），两个输出（加法执行后该位的结果S，加法导致的进位cout），根据三个输入的所有可能，可以列出如下真值表：

A	B	C_{in}	C_{out}	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

可以看出cout和S关于三个输入的逻辑表达式为：

Cout = AB + AC + BC

S = ABC + A(非B)(非C) + (非A)B(非C) + (非A)(非B)C

化简后可以得到：

Cout = AB + AC + BC

S = A^B^C

门延迟分析：

假定每一个逻辑门的通过时间一定，每一个逻辑门的通过时间为1，下面所有的门延迟分析都基于这一点。

则根据上面的化简后式子可以知道，全加器的门延迟为2

第一个延迟：

A和B要同时通过一个与门和异或门（cout的AB，AC，BC和S的A^B）

第二个延迟：

最后AB，AC，BC要通过一个或门，A^B要与C异或得到cout和S。

（2） 先行进位加法器：

如果我们要构建32位加法器，按照全加器的构建方式，我们只要将32个全加器串联即可，但是这样32位的门延迟特别长，需要32 * 2 = 64个门延迟，因此，我们需要设计一种能够先行进位的加法器来减少延迟。

由于cout = AB + AC + BC， 化简可以得到AB+(A+B)C，此处我们定义进位生成因子（gi）：Ai * Bi，进位传递因子（pi）：Ai + Bi，其中i表示当前的位数，此时便可以写出进位的递推式：$C_{i+1} = A_iB_i + (A_i+B_i)C_i = g_i + p_iC_i$

由于最低为的cin为0，通过不断将C_i替换为关于C_{i-1}的式子，便能得到如下表达式：

$C_1 = g_0 + p_0C_0$

$C_2 = g_1 + p_1g_0 + p_1p_0C_0$

$C_3 = g_2 + p_2g_1 + p_2p_1g_0 + p_2p_1p_0C_0$

$C_4 = g_3 + p_3g_2 + p_3p_2g_1 + p_3p_2p_1g_0 + p_3p_2p_1p_0C_0$

……

此时我们就能通过这些逻辑式子构建超前进位来加快速度。

门延迟分析：

对于一个进位，如$C_4 = g_3 + p_3g_2 + p_3p_2g_1 + p_3p_2p_1g_0 + p_3p_2p_1p_0C_0$，可以看出门延迟为3

第一个门延迟：

A_iB_i和A_i+B_i算出每一个pi和gi

第二个门延迟：

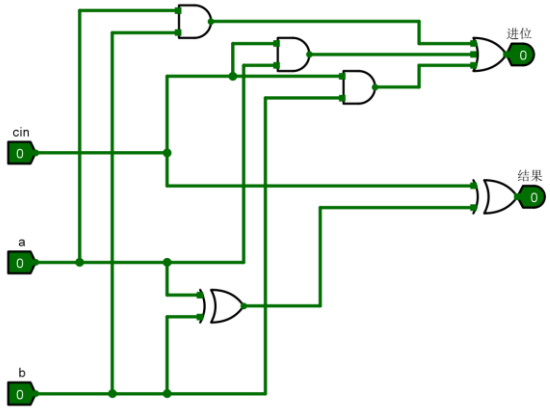
每一个‘+’之间式子的计算

第三个门延迟：

整体通过一个或门得到结果

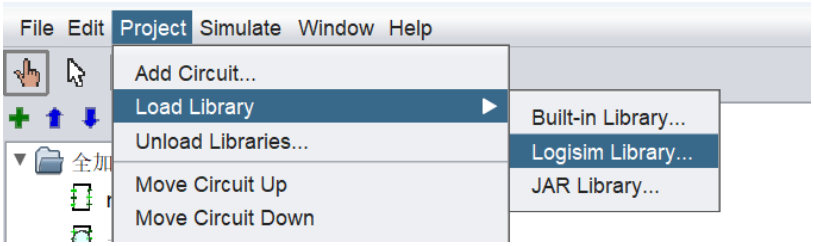
五、 实验过程

在正式开始实验之前，由于我是第一次使用logism这个软件，我花了大约一个小时时间熟悉该软件的操作，并根据一位全加器的原理和ppt上的讲解自己设计了一个全加器：



在下面的实验中，所有加法器的实现都是根据我自己的全加器，并没有使用模板里面的全加器，因此，所有的门延迟分析都是基于我自己的全加器。

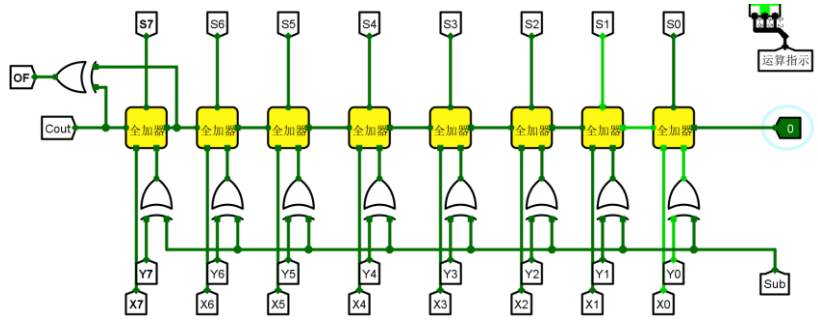
同时为了在模板中使用自己设计的全加器，我学会了从logism中导入自定义库：



然后选择本地的circ文件即可。

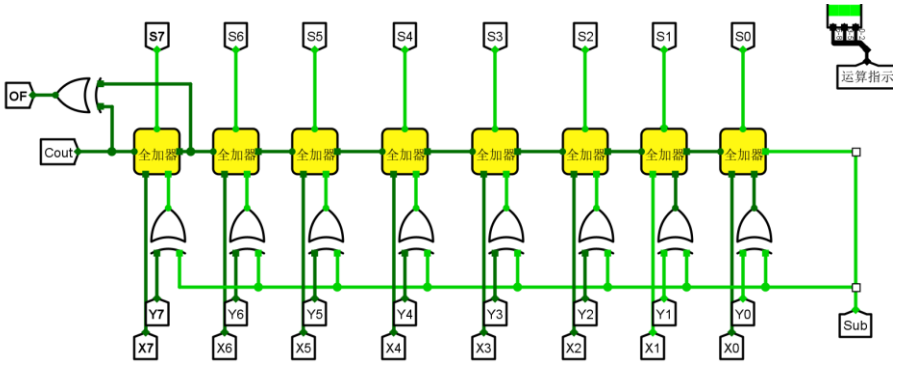
（1） 8位可控加减法设计

该加法器的要求是根据sub位是否设置决定加法器的行为是加法还是减法，我最先的思路是，将sub的值和第二个操作数进行异或（这样，如果sub为1，第二个操作数就会取反，实现减法，否则不会有任何影响），然后就按照正常的加法器串联实现，便有了如下的初级版本：



其中OF表示溢出，是最高位进位和次高位进位的异或（这个部分我查了资料才做出来，因为最高位是符号位，要是溢出了，符号会改变，根据进位结果的真值表可以发现是最高位进位和次高位进位异或的结果）

但是这个结果是有问题的，它在计算减法时得到的结果会比实际结果小一，因为正数变为其相反数是反码加一，发现这个问题后我将sub和cin连起来，才解决：



实验结果和门延迟分析见后面一部分。

（2） 4位先行进位74182：

先行进位的本质是将复杂的高位进位展开成由初始状态能得到的逻辑式子，再通过门电路相连，在实验原理中，我已经将四位先行加法进位的表达式给出：

$C_1 = g_0 + p_0 C_0$

$C_2 = g_1 + p_1 g_0 + p_1 p_0 C_0$

$C_3 = g_2 + p_2 g_1 + p_2 p_1 g_0 + p_2 p_1 p_0 C_0$

$C_4 = g_3 + p_3 g_2 + p_3 p_2 g_1 + p_3 p_2 p_1 g_0 + p_3 p_2 p_1 p_0 C_0$

其中进位生成因子（gi）= Ai * Bi，进位传递因子（pi）= Ai + Bi

而该先行进位模块就是求出四个位分别的进位，以及整个模块产生的进位传递因子P和进位生成因子G（这样就可以通过4位快速加法器构造更高位的快速加法器，而不是4位快速加法器串联在一起）

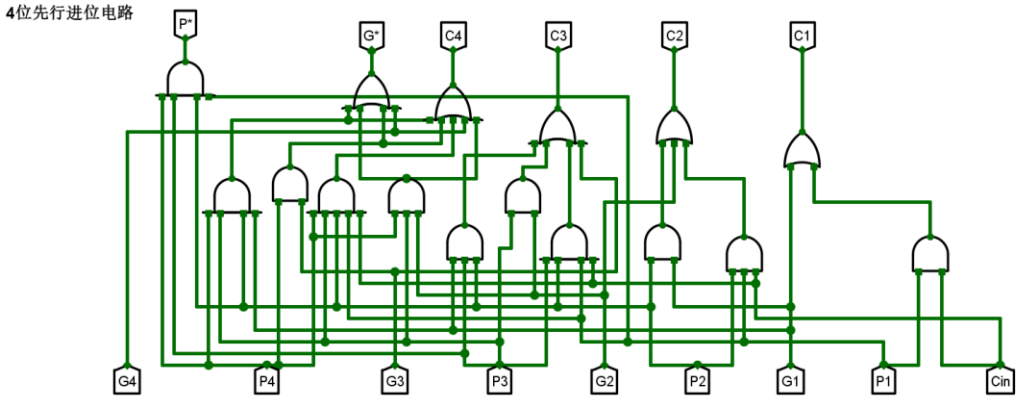
对于整个模块产生的进位传递因子P和进位生成因子G，他们的逻辑表达式如下：

$P_0 = p_3 p_2 p_1 p_0$

$G_0 = g_3 + (p_3 g_2) + (p_3 p_2 g_1) + (p_3 p_2 p_1 g_0)$

对于进位生成因子G，实际上就是模块是否有与上一个模块进位无关的进位（即去掉C_4里C_0的部分），而进位传递因子P则是表示模块中每个进位传递因子都有效（这样才能说明整个模块是能够传递这个ai + bi的）。

根据逻辑表达式，就可以画出如下连接图：

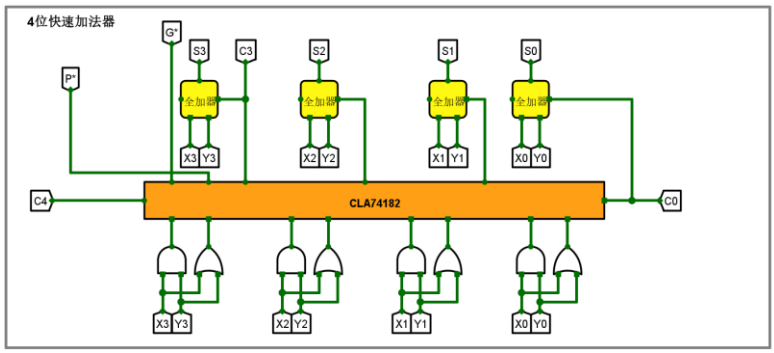


实验结果和门延迟分析见后面一部分。

（3） 4位快速加法器：

由于我们已经构建了4位先行进位部件74182，所有的进位信号只要由74182产生即可，

其他就是4个全加器分别计算：



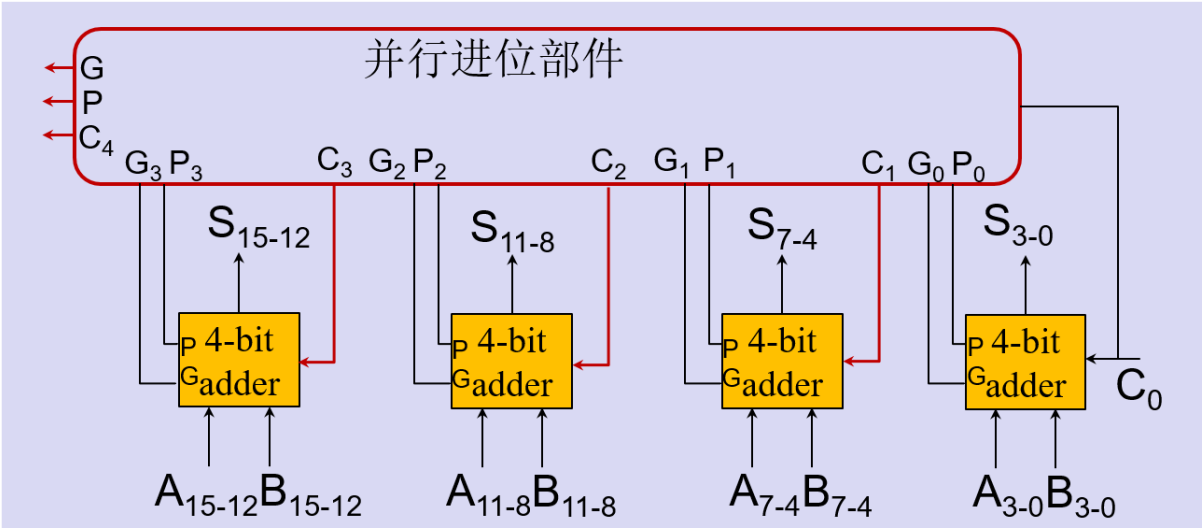
该处下方的连接点都是pi（xi + yi）和gi（xi * yi），就分别设置了一个门，上方就是实际计算的部分，74182产生的进位直接连到全加器的进位部分来加速计算，避免串联。

实验结果和门延迟分析见后面一部分。

（4） 32位快速加法器：

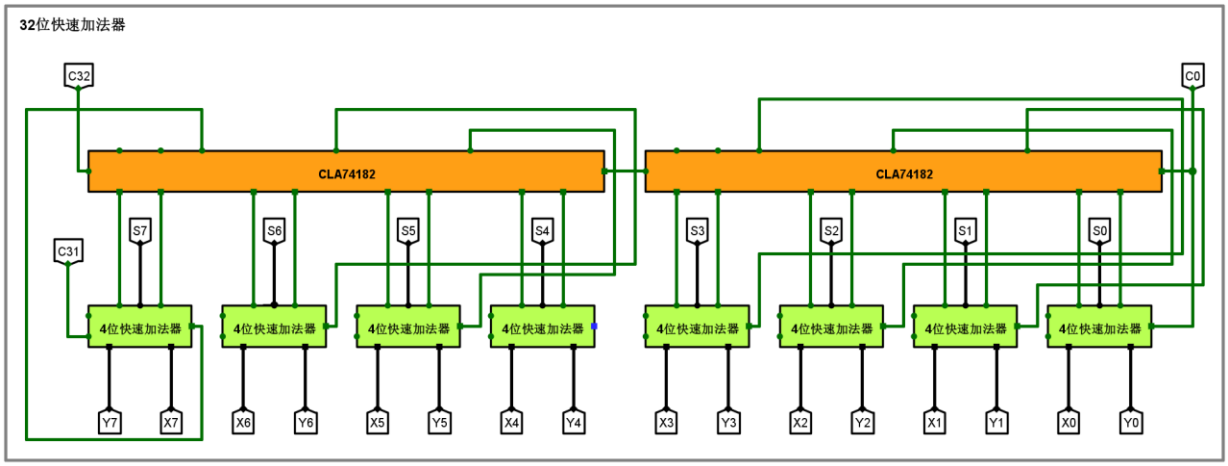
最开始做这个实验的时候，我仔细想了一遍，由于74182只提供计算进位的功能，因此计算结果的功能一定是上一个实验完成的4位加法器来做，每个4位加法器又会产生相应的G和P和进位C4，由于是快速加法器，不可能是4位加法器之间串联，想到这里的时候，我产生了疑惑，按照上面的分析，C4是没有用的，要不要连？之后，在我回顾构建4位快速加法器的时候，我意识到实际上并没有用到一位全加器的进位，32位快速加法器应该也是同理。

但是，在实际接线的时候，我还是对整个构建的方式不清晰，后面打开ppt查看，发现了这张图：

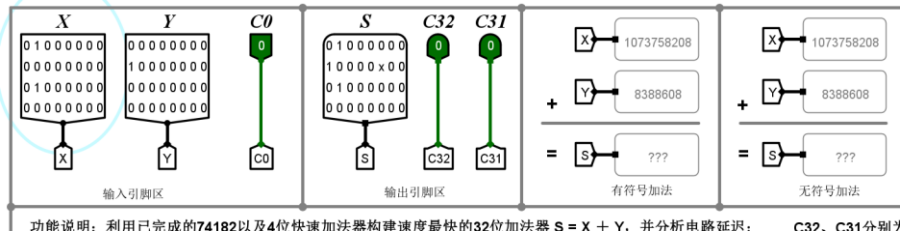


此时我立马明白了多位快速加法器的构建方式，不过ppt上的是16位的，我们在构建的时候还需要一个74182和四个4位快速加法器来实现32位快速加法器。

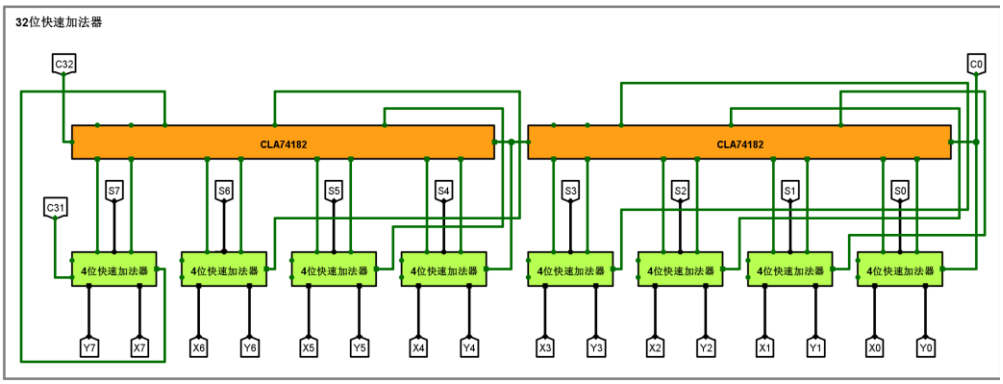
在按照ppt上的图连线之后，有了做出来的第一个版本：



不过这个版本是失败的，将它们分成左右两边可以发现，都是和ppt上的图一致，但是两个16位快速加法器之间的连接我并没有想好，我只是简单地将右边低位的C4进位传递给左边的Cin，而没有传递给下一个4位快速加法器，也就是说，该加法器只有低16位计算正确。我是在测试如下图的式子时发现这个问题的：



显然，由于有一个4位加法器的进位输入未知，因此结果也未知，在发现问题后，我便将右侧74182的C4进位也接到下一个的4位加法器上：



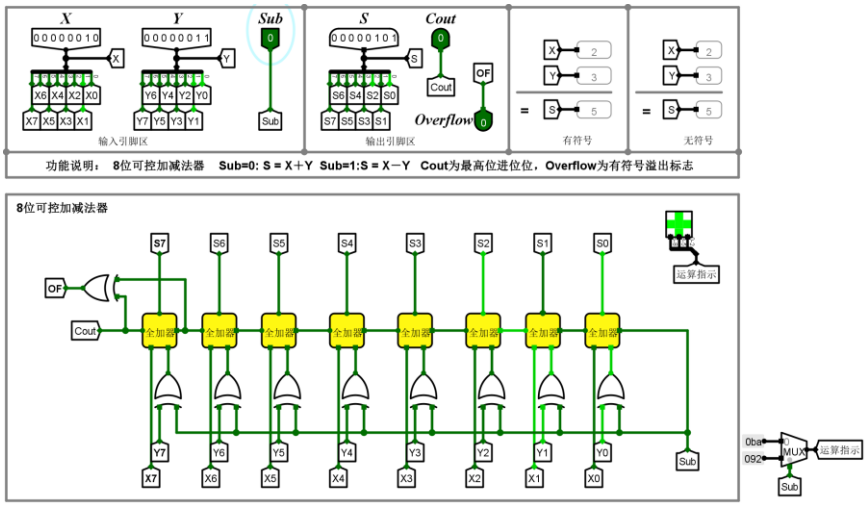
此时计算结果就是正确的了。

实验结果和门延迟分析见后面一部分。

六、 实验结果及分析

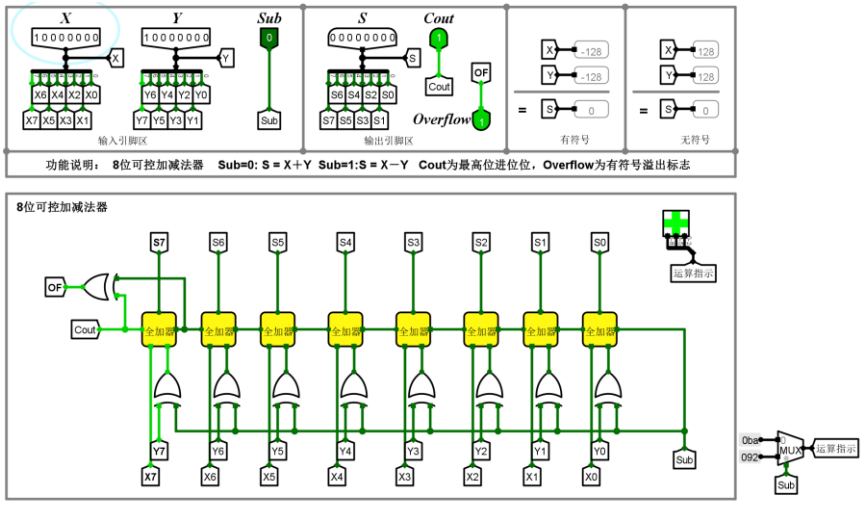
（1） 8位可控加减法设计：

普通加法：



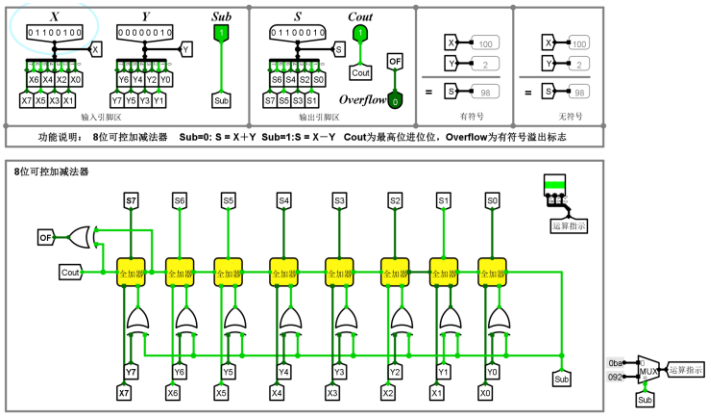
2 + 3 = 5，正确

加法溢出：



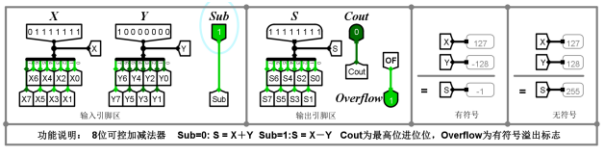
−128 + （−128） 根据溢出，值应该为0，故正确。

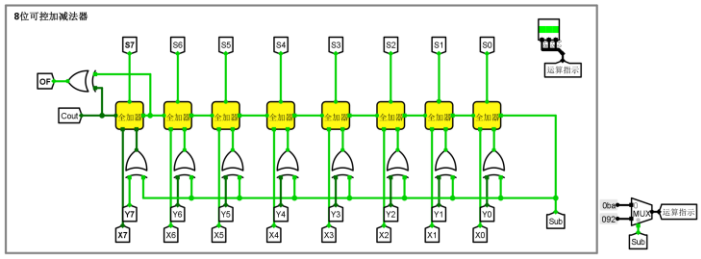
普通减法：



100 − 2 = 98，正确

减法溢出：





此处需要注意的是，y是要先求相反数再和x相加，因此右上角结果正确却溢出，当时我检查的时候以为我错了，为什么结果正确却溢出了，发现原来是我没完全理解补码加法。

门延迟分析：

每个全加器有2个门延迟，而第一个全加器在得到结果之前要先等待Y0的结果，故门延迟计算如下：

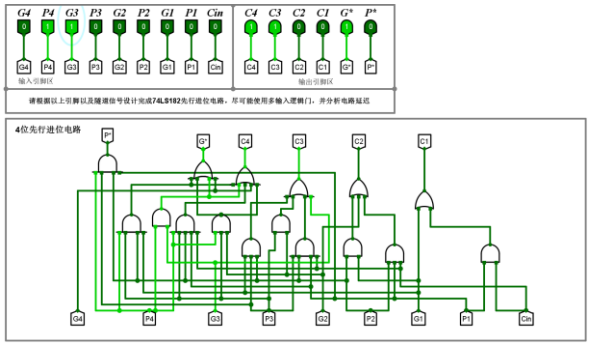
结果的门延迟：3 + （2 * 7）= 17个门延迟

Cout的门延迟：与结果门延迟一样，为17

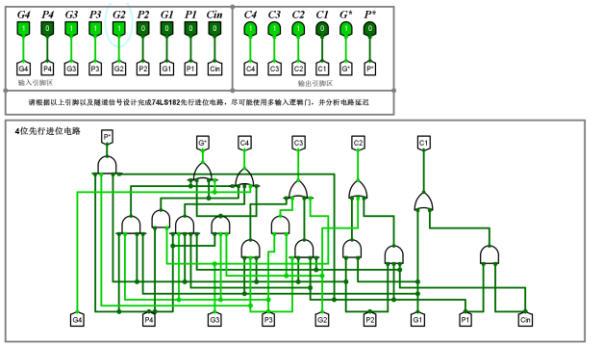
OF的门延迟：OF要额外等待一个门延迟，故为17 + 1 = 18个门延迟。

（2） 4位先行进位74182：

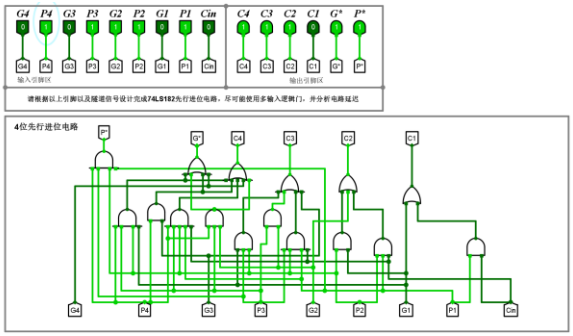
随机选取5个组合进行验证：



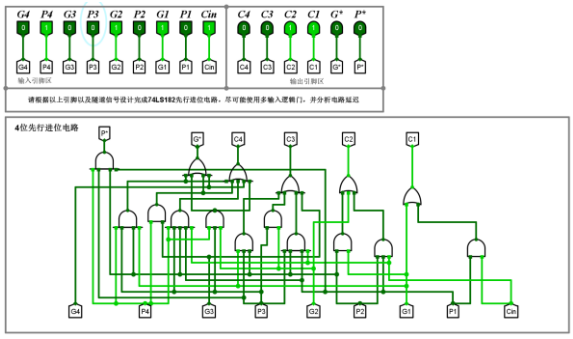
G3使能，故C3置位，P4使能，加上G3的进位，故C4置位，大G也置位。



G4使能，故大G，C4置位，G3使能，C3置位，G2使能，C2置位。



P1，P2，P3，P4使能，故大P置位，G2使能，C2置位，C2加上P3，使C3置位，C3加上P4使C4置位，同理，由于是内部进位，大G也置位。



G1使能，C1置位，G2使能，C2置位，此处cin并没有发挥作用，可以看到右下角p1的线没有亮。

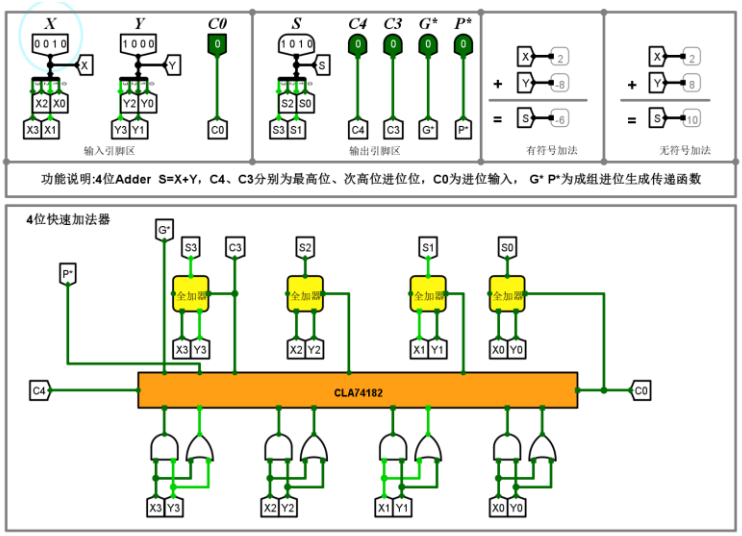
其他的测试实际包含在后面的实验里了，如果这个做错了，后面肯定也做错。

门延迟分析：

假设此时我们已经得到pi和gi，则根据图中的门电路可以看出，对于几乎所有输出变量，门延迟为2，只有大P（P1 * P2 * P3 * P4）只有一个门延迟，这为后续两个实验的门延迟分析奠定基础。

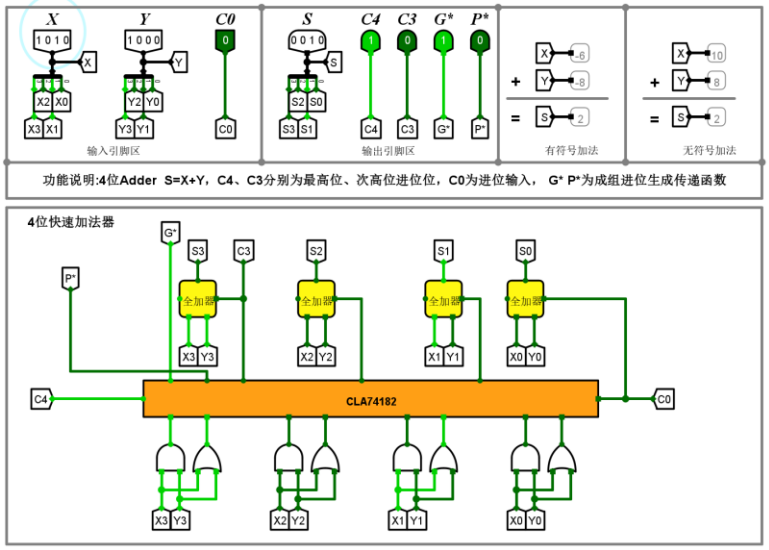
（3） 4位快速加法器：

普通加法：



可以看到右上角有符号和无符号的结果正确。

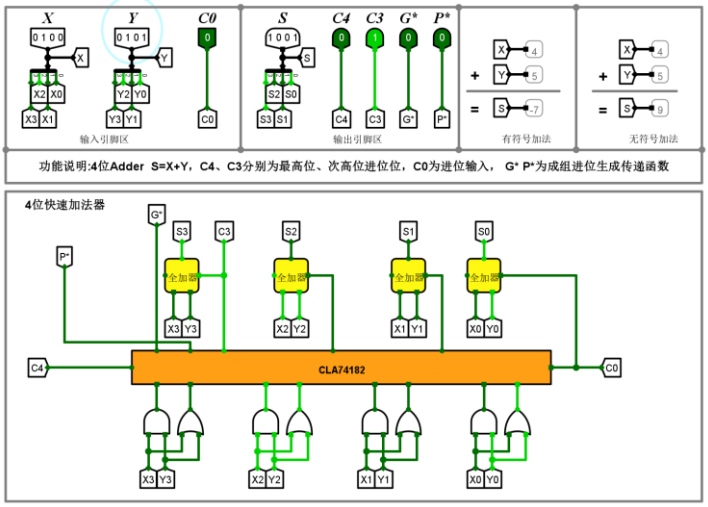
加法溢出：



根据补码加法，-6 + （-8） = -8 + （-6）= 7（此时溢出了） + （-5）= 2

无符号加法同理。

随机测试一组：



4 + 5 = 9，仅仅溢出一位，故 -8 + 1 = -7.

门延迟分析：

由于xi和yi都要得到对应的进位传递因子和进位生成因子，这里有一个门延迟，而74182有两个门延迟（上一节分析了），此时总共3个门延迟，而一个全加器的门延迟为2，故得到结果的延迟为5，而进位实际上是由74182得到的，故整个模块的进位门延迟为3，大P的门延迟根据上一节的分析，此处为2。

总结如下：

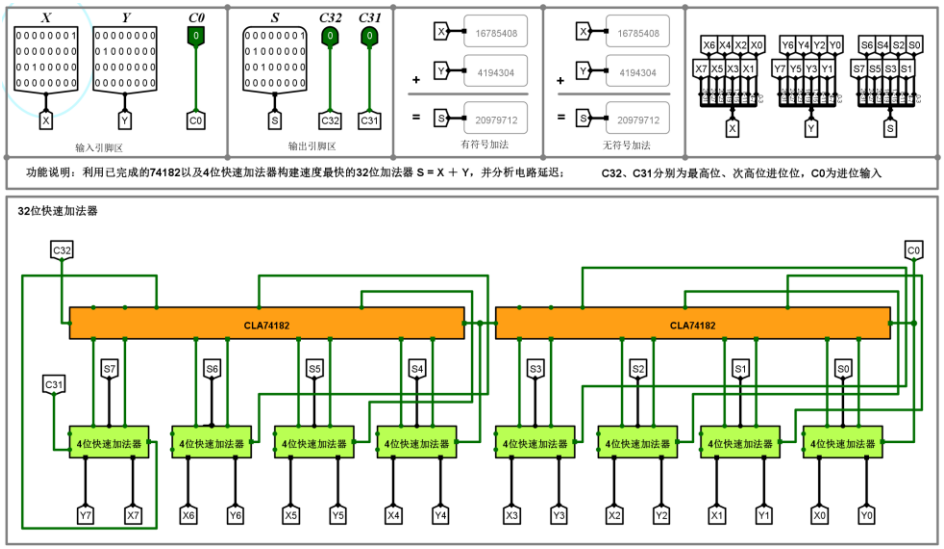
运算结果的门延迟：5个

进位（C4）和大G的门延迟：3个

大P的门延迟：2个

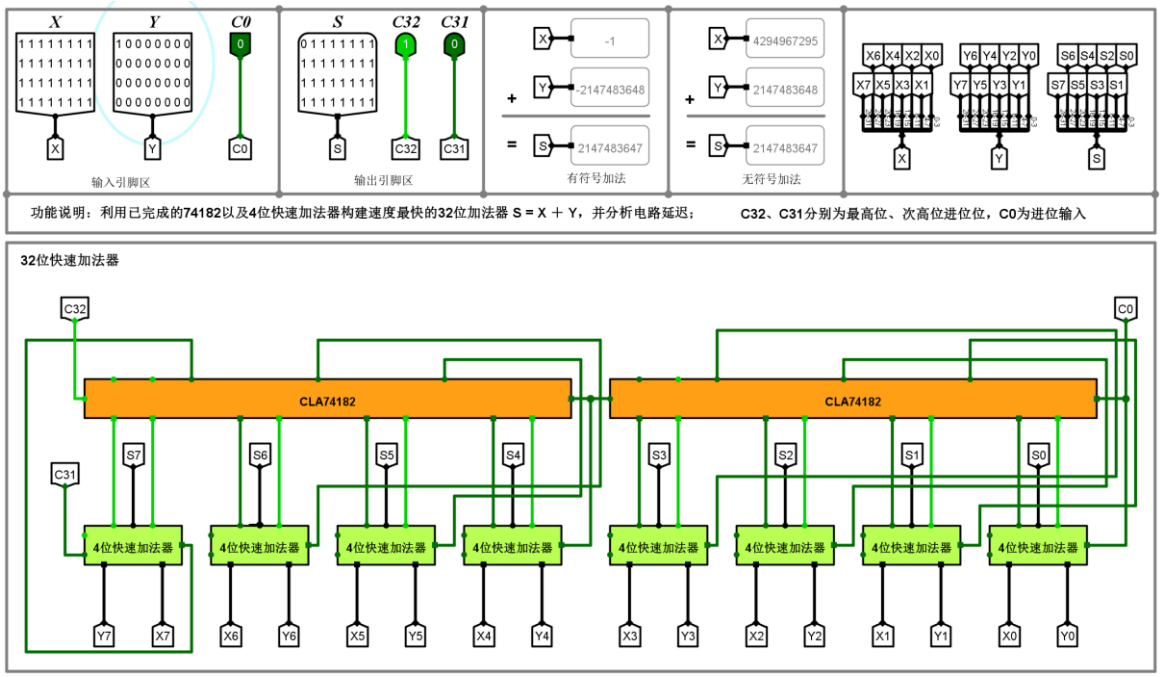
（4） 32位快速加法器：

普通加法：



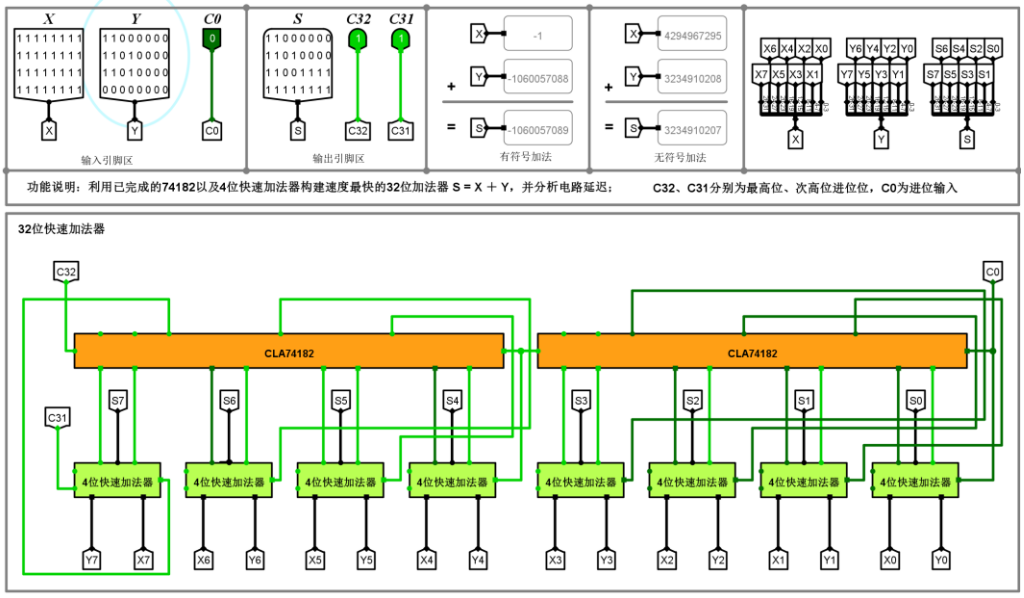
16785408 + 4194304 = 20979712，计算正确

加法溢出：



可以看到有符号int最小值为-2147483648，减一后数值下溢，变成最大值2147483647，对于无符号加法也是同理。

随机测试一组：



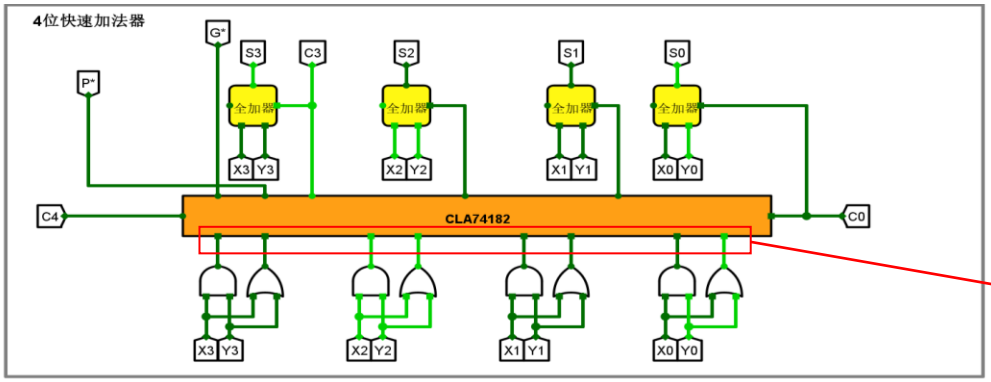
可以看到此处有符号加法并未溢出，由于在其表示范围内，而无符号加法产生了溢出，但是结果仍然正确。

门延迟分析：

对于每一个4位快速加法器，产生大G和大P的时间为3个门延迟，这些大G和大P通过右侧的74182产生进位的门延迟为2，此时左侧的714182才能正确计算进位，因此又需要2个门延迟产生进位（左侧74182的进位），进位传递给高16位的快速加法器后，经过4个

门延迟，算出结果（低16位的加法器在右侧74182算出进位后就提前得到低16位加法结果了），故总的门延迟为3 + 2 + 2 + 4 = 11个门延迟。

此处解释为什么4位快速加法器是4个门延迟算出结果（上一节分析的实际上是5），这是由于pi和gi已经算出：



故延迟仅为74182和全加器的延迟。

因此门延迟为11。

七、 问题讨论

在连线的时候比较奇怪这些人是怎么想到这些方法的，以及为什么要以4位快速加法器作为基础构建32位快速加法，为什么不用8位快速加法器，理论上更快。后面上网查了资料，发现大部分实现都是通过4位快速加法器，也有使用8位快速加法器的，但是数量不多，结论更偏向于是一种约定俗成的使用方式。

八、 心得体会

本次实验十分有意思，只要在电脑上连连线就可以看到自己的结果，当我实际做出32位加法器的时候还是蛮激动的（因为里面的全加器是我自己做的，没有使用给定的封装实现），如果是使用实验箱（数字逻辑那个祖传实验箱是真lj），估计很难得出结果。

另外一个特别的点就是门延迟的计算，尤其是计算32位快速加法器的门延迟时，发现还是很绕的，而且对于不同的实现，门延迟还不一样，有时候发现计算的过程中自己把自己搞晕了，又必须重新计算一遍。

整体来讲，整个实验过程是比较顺利的，对于alu加法器的构建有了一个较为全面的理解，也自己实现了一遍，里面一些细小的点也都注意到了，我相信这对后面的构建单时钟周期的cpu会有很大帮助。