|  |
| --- |
| 一生一芯-answer |
|  |
|  |
| 作者：HateHanzo |
| 联系方式：HateHanzo@163.com |

版权所有 侵权必究

**修改记录**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 版本号 | 修改文件 | 描述 | 作者 | 时间 |
| v1.0 |  | 初稿 | HateHanzo | 20250728 |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

**缩略语与约定符号**

**缩略语**

|  |  |  |
| --- | --- | --- |
| **英文简写** | **英文全称** | **中文描述** |
| AHB | Advanced High-performance Bus | 先进高性能总线 |
| SRAM | Static Random-Access Memory | 静态随机存取存储器 |
| FSM | Finite State Machine | 有限状态机 |
| SR-Latch | Set-Reset-Latch | SR锁存器 |
| ICG | Integrated Clock Gating Cell | 门控时钟 |

**约定符号**

1、对于某信号名A，若无特殊声明，A\_d1表示该信号延时一拍，A\_d2表示延时两拍，以此类推。如ahb\_addr\_d1表示信号ahb\_addr延时一拍。

**目 录**

[**修改记录** I](#_Toc206518541)

[**缩略语与约定符号** II](#_Toc206518542)

[**目 录** 0](#_Toc206518543)

[**1 F阶段** 1](#_Toc206518544)

[**1.1** **F3** 1](#_Toc206518545)

[**1.1.1 分析门电路** 1](#_Toc206518546)

[**1.1.2 或门的晶体管结构** 2](#_Toc206518547)

[**1.1.3 对比两种实现的晶体管所需要的数量** 2](#_Toc206518548)

[**1.1.4 用其他门电路搭建异或门** 2](#_Toc206518549)

[**1.1.5  异或门的全定制电路** 3](#_Toc206518550)

[**1.1.6 设计同或门** 4](#_Toc206518551)

[**1.1.7 同或门的全定制电路** 5](#_Toc206518552)

[**1.1.8 八进制(octal)计数法** 6](#_Toc206518553)

[**1.1.9 搭建2-4译码器** 6](#_Toc206518554)

[**1.1.a 译码器的扩展** 7](#_Toc206518555)

[**1.1.b 搭建七段数码管译码器** 8](#_Toc206518556)

[**1.1.c 搭建七段数码管译码器(2)** 9](#_Toc206518557)

[**1.1.d 搭建编码器** 10](#_Toc206518558)

[**1.1.e 搭建4-2优先编码器** 11](#_Toc206518559)

[**1.1.f 优先编码器的扩展** 12](#_Toc206518560)

[**1.2.0 前导0和前导1的计数** 13](#_Toc206518561)

[**1.2.1 搭建1位2选1选择器** 14](#_Toc206518562)

[**1.2.2 搭建3位4选1选择器** 14](#_Toc206518563)

[**1.2.3 搭建可切换进位计数制的七段数码管** 15](#_Toc206518564)

[**1.2.4 搭建比较器** 15](#_Toc206518565)

[**1.2.5 搭建1位全加器** 15](#_Toc206518566)

[**1.2.6 搭建1位全加器(2)** 16](#_Toc206518567)

[**1.2.7 搭建4位加法器** 17](#_Toc206518568)

[**1.2.8 搭建4位减法器** 17](#_Toc206518569)

[**1.2.9 搭建4位原码加法器** 18](#_Toc206518570)

[**1.2.a 搭建4位反码加法器** 19](#_Toc206518571)

[**1.2.b 搭建4位反码加法器(2)** 19](#_Toc206518572)

[**1.2.c 检测补码加法是否发生溢出** 20](#_Toc206518573)

[**1.2.d 搭建SR锁存器** 21](#_Toc206518574)

[**1.2.e 用与非门搭建的SR锁存器** 21](#_Toc206518575)

[**1.2.f 分析D锁存器的行为** 22](#_Toc206518576)

[**1.3.0 搭建D锁存器** 22](#_Toc206518577)

[**1.3.1 搭建带复位功能的D锁存器** 22](#_Toc206518578)

[**1.3.2 用D锁存器实现位翻转功能** 23](#_Toc206518579)

[**1.3.3 搭建D触发器** 23](#_Toc206518580)

[**1.3.4 搭建带复位功能的D触发器** 23](#_Toc206518581)

[**1.3.5 用D触发器实现位翻转功能** 23](#_Toc206518582)

[**1.3.6 搭建下降沿触发的D触发器** 24](#_Toc206518583)

[**1.3.7 搭建带使能端的D触发器** 24](#_Toc206518584)

[**1.3.8 搭建4位寄存器** 24](#_Toc206518585)

[**1.3.9 搭建4位计数器** 24](#_Toc206518586)

[**1.3.a 设计数列求和电路** 24](#_Toc206518587)

[**1.3.b 实现电子时钟** 25](#_Toc206518588)

[**1.2** **F4** 25](#_Toc206518589)

[**1.2.1 继续执行上述指令** 25](#_Toc206518590)

[**1.2.2 计算10以内的奇数之和** 26](#_Toc206518591)

[**1.2.3 继续执行上述程序** 27](#_Toc206518592)

[**1.2.4 从状态机视角理解数列求和电路的工作过程** 28](#_Toc206518593)

[**1.3** **F5** 29](#_Toc206518594)

[1.3.1 实现完整的sCPU 29](#_Toc206518595)

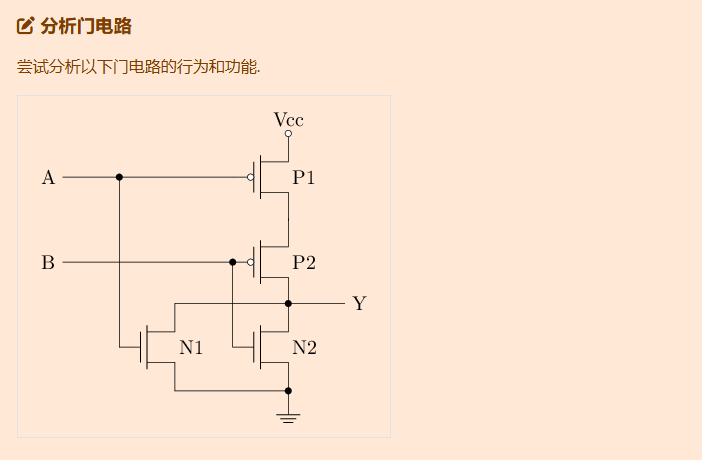
[**参考文献** 29](#_Toc206518596)

[**附 录** 30](#_Toc206518597)

**1 F阶段**

* 1. **F3**

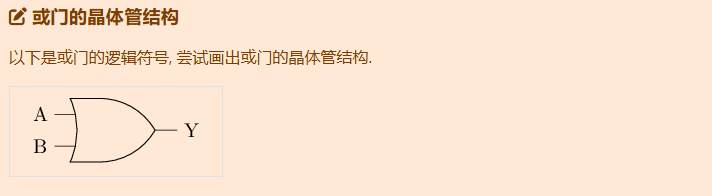
**1.1.1 分析门电路**



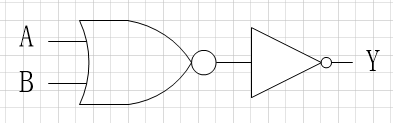
| **A** | **B** | **P1** | **P2** | **N1** | **N2** | **Y** |
| --- | --- | --- | --- | --- | --- | --- |
| 0 | 0 | 导通 | 导通 | 截止 | 截止 | 1 |
| 0 | 1 | 导通 | 截止 | 截止 | 导通 | 0 |
| 1 | 0 | 截止 | 导通 | 导通 | 截止 | 0 |
| 1 | 1 | 截止 | 截止 | 导通 | 导通 | 0 |

该电路是一个或非门

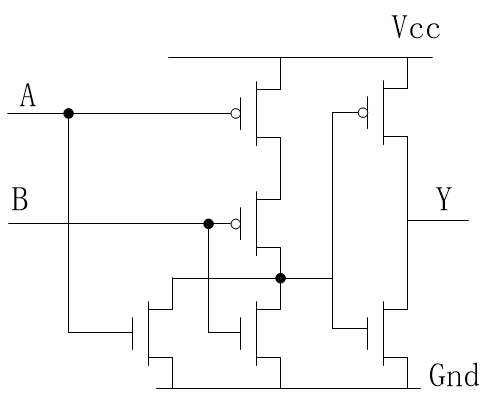
**1.1.2 或门的晶体管结构**



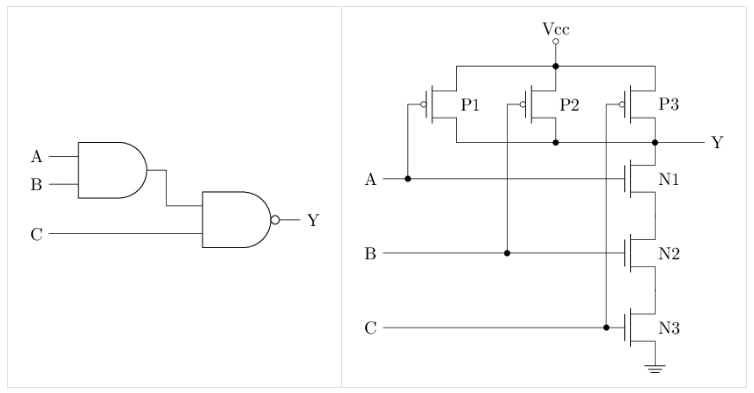
或门只需要或非门电路的输出和非门电路的输入连线即可得到，门电路结构可表示为：



晶体管结构为：

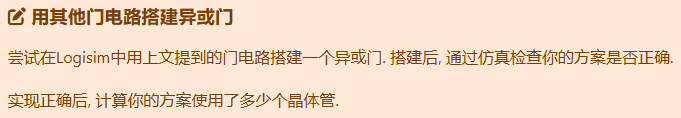


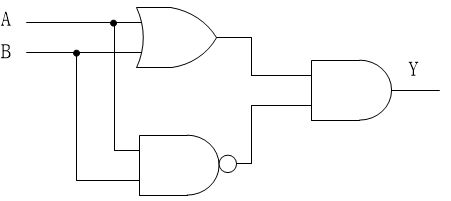
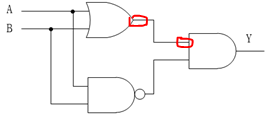
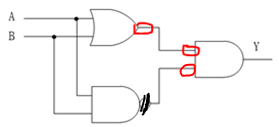
**1.1.3 对比两种实现的晶体管所需要的数量**

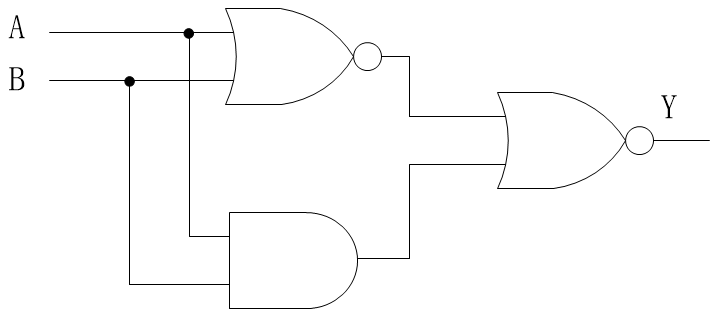
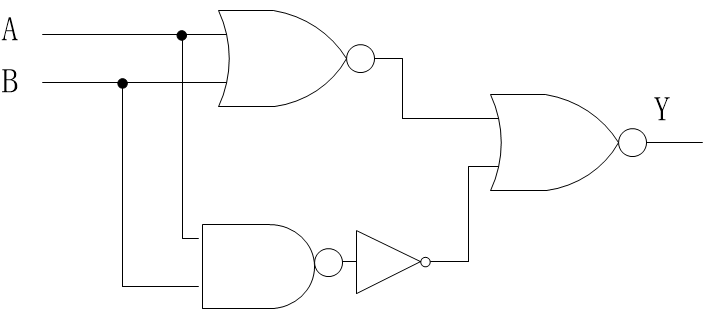


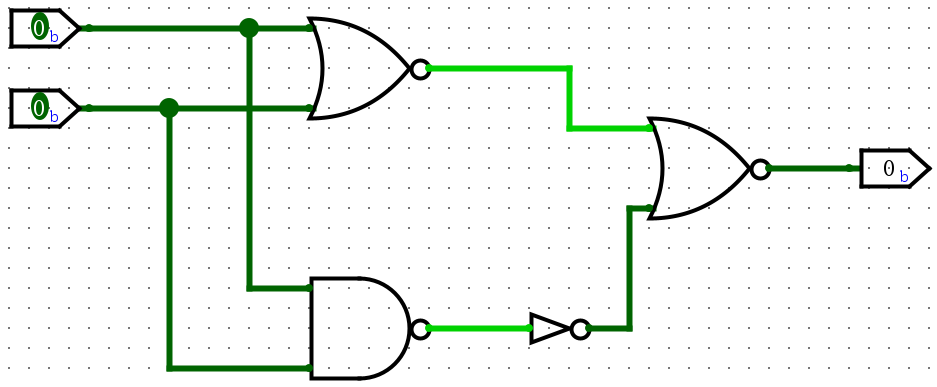
左图为与门+与非门，6+4=10；右图为6。

**1.1.4 用其他门电路搭建异或门**



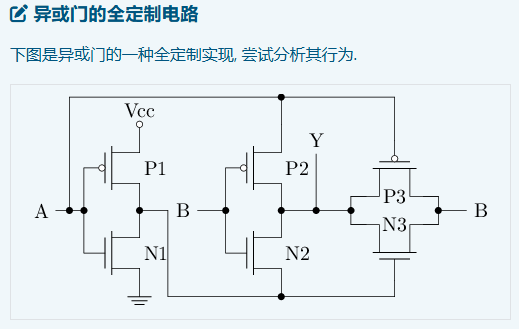
 ->  ->  ->

 -> 

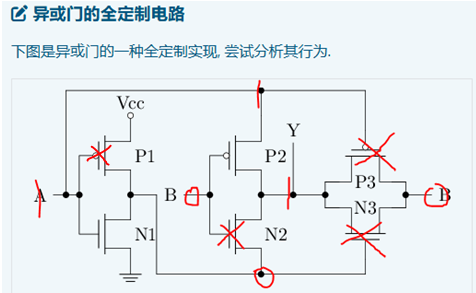


4+4+4+2 = 14

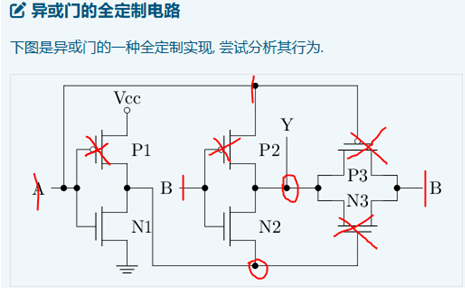
**1.1.5  异或门的全定制电路**



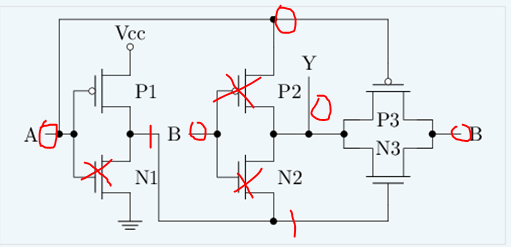
A=1,B=0，Y=1



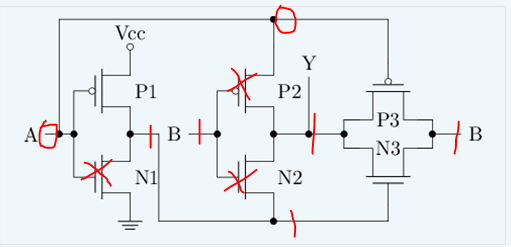
A=1,B=1，Y=0



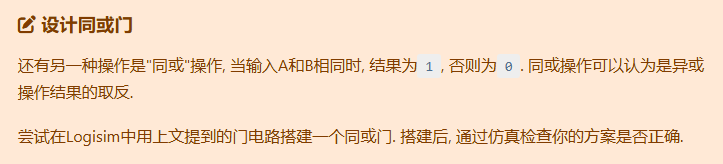
A=0,B=0，Y=0

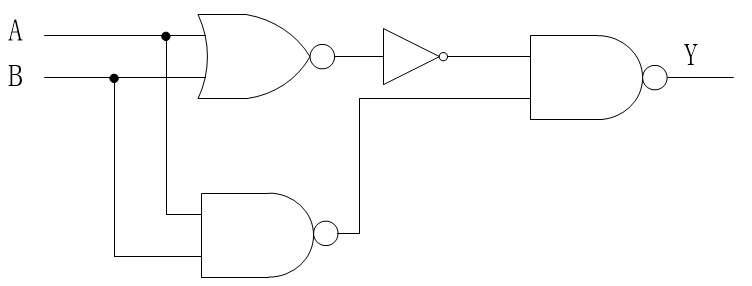


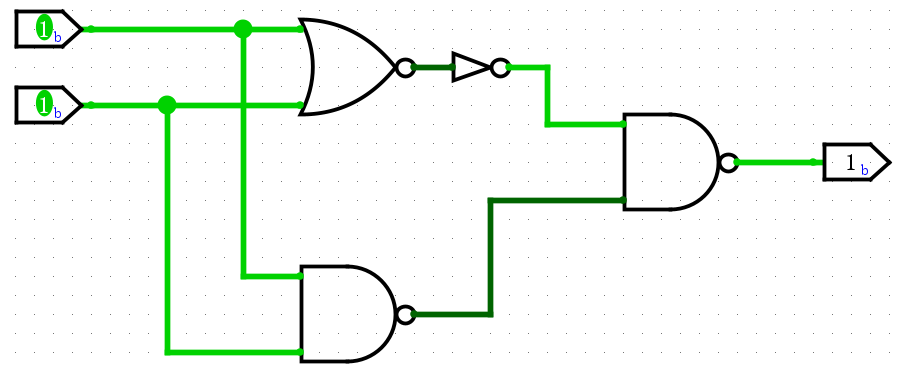
A=0,B=1，Y=1



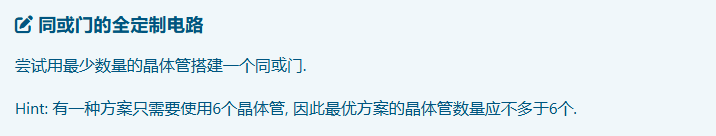
**1.1.6 设计同或门**

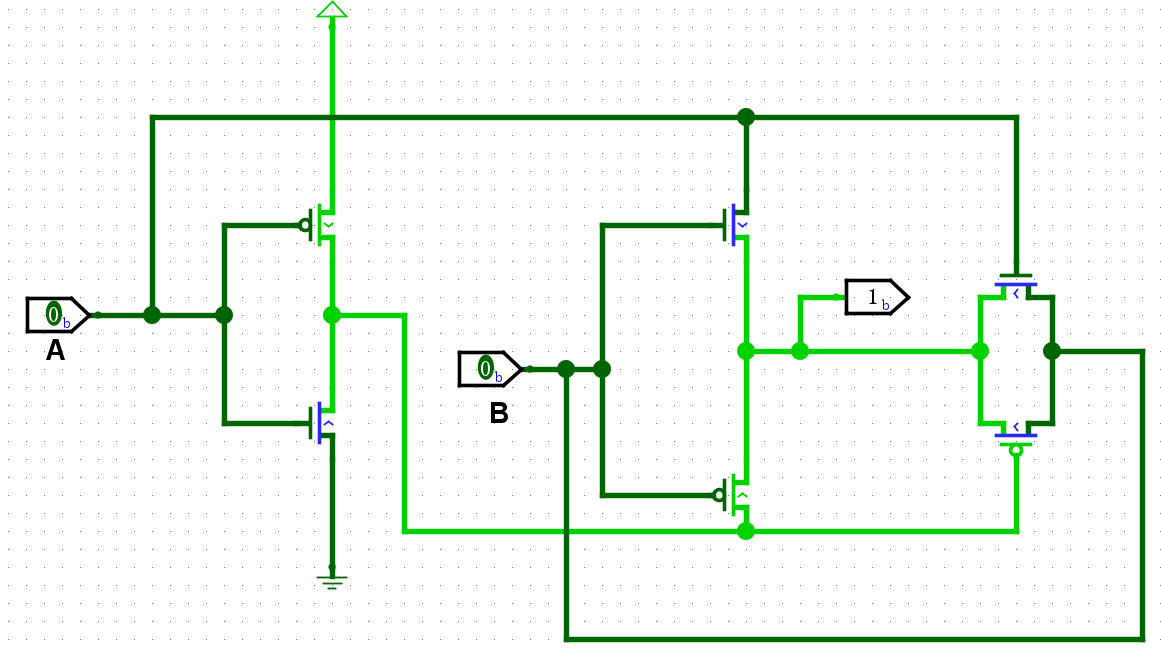






**1.1.7 同或门的全定制电路**





**1.1.8 八进制(octal)计数法**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 2进制 | 100 | 111 | 1000 | 1111 |
| 8进制 | 4 | 7 | 10 | 17 |
| 10进制 | 4 | 7 | 8 | 15 |

2进制转8进制：从最右为起点，每3个bit为一组建立和8进制的一一对应关系，比如000-0；001-1；。。。110-6；111-7，不足3位的用0补齐即可实现

8进制转2进制，每一位按照对应关系转换为2进制即可，比如17，7对应111，1对应001，那么17对应的2进制就是001111，通常把高位的0舍去，变成1111

8进制转10进制就是和2进制转10进制一样，加权求和，比如17转换为10进制就是1\*8^1 + 7\*8^0 = 15

10进制转8进制就是不停的除以8直到商为0，记录下余数，最后把余数倒序排列即可，比如十进制255转成8进制就是：

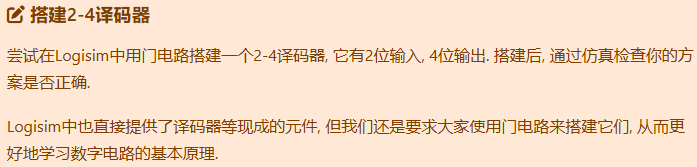
255/8=31…7

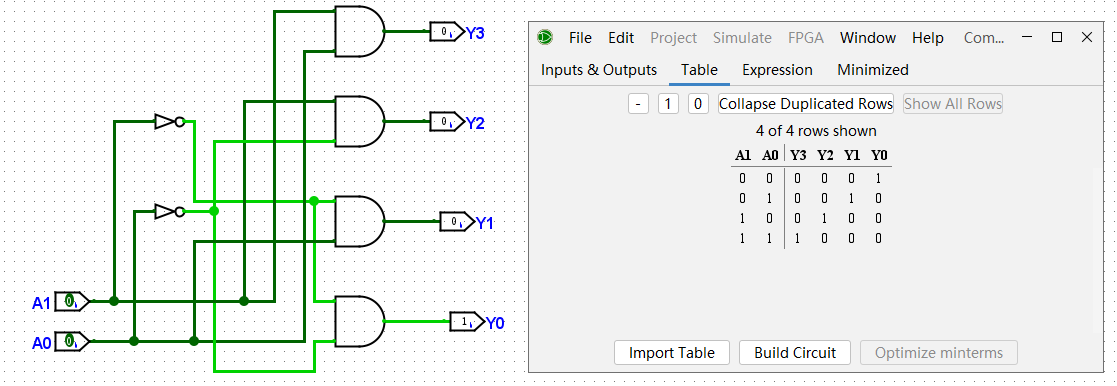
31/8=3…7

3/8=0…3

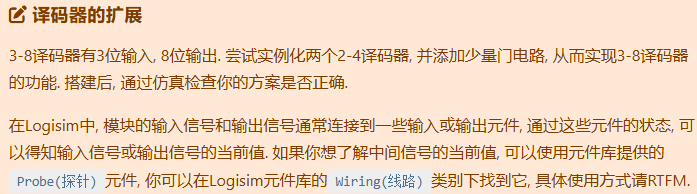
所以255对应的8进制就是377

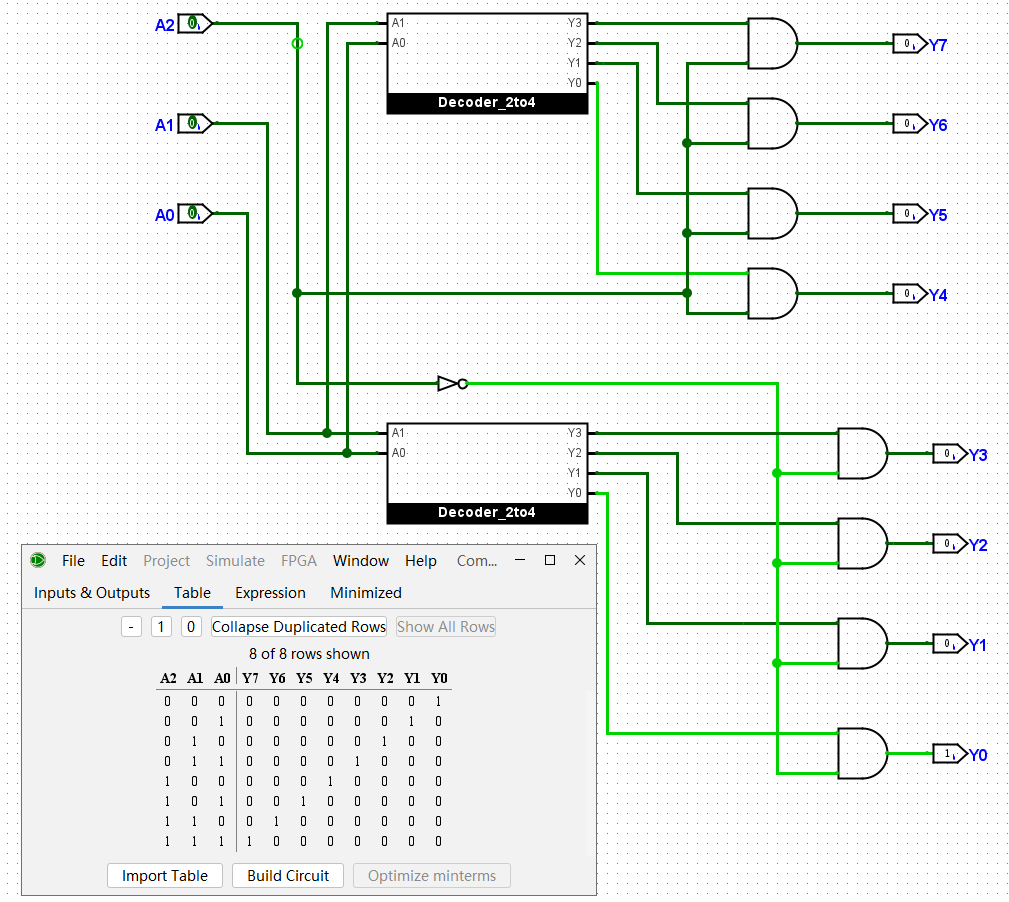
**1.1.9 搭建2-4译码器**



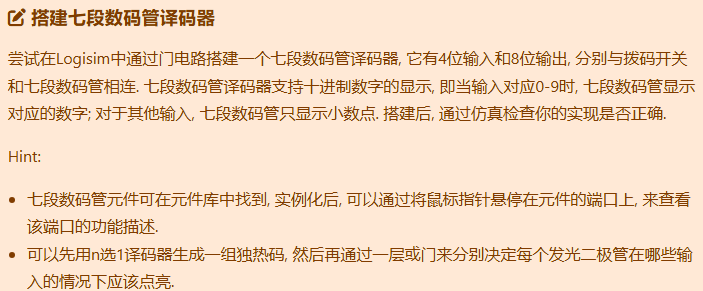


**1.1.a 译码器的扩展**





**1.1.b 搭建七段数码管译码器**





|  |  |  |  |
| --- | --- | --- | --- |
| 拨码开关 | 4-16译码器输出 | 数码管显示数字 | 需点亮的数码管 |
| 0000 | 0000\_0000\_0000\_0001 | 0 | a/b/c/d/e/f |
| 0001 | 0000\_0000\_0000\_0010 | 1 | b/c |
| 0010 | 0000\_0000\_0000\_0100 | 2 | a/b/d/e/g |
| 0011 | 0000\_0000\_0000\_1000 | 3 | a/b/c/d/g |
| 0100 | 0000\_0000\_0001\_0000 | 4 | b/c/f/g |
| 0101 | 0000\_0000\_0010\_0000 | 5 | a/c/d/f/g |
| 0110 | 0000\_0000\_0100\_0000 | 6 | a/c/d/e/f/g |
| 0111 | 0000\_0000\_1000\_0000 | 7 | a/b/c |
| 1000 | 0000\_0001\_0000\_0000 | 8 | a/b/c/d/e/f/g |
| 1001 | 0000\_0010\_0000\_0000 | 9 | a/b/c/d/f/g |
| 其他 | … | 小数点 | h |

a = b0|b2|b3|b5|b6|b7|b8|b9

b = b0|b1|b2|b3|b4|b7|b8|b9

c = b0|b1|b3|b4|b5|b6|b7|b8|b9

d = b0|b2|b3|b5|b6|b8|b9

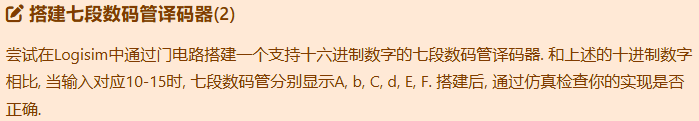
e = b0|b2|b6|b8

f = b0 |b4|b5|b6|b8|b9

g= b2|b3|b4|b4|b6|b8|b9

h= |b[15:10]

**1.1.c 搭建七段数码管译码器(2)**





|  |  |  |  |
| --- | --- | --- | --- |
| 拨码开关 | 4-16译码器输出 | 数码管显示数字 | 需点亮的数码管 |
| 0000 | 0000\_0000\_0000\_0001 | 0 | a/b/c/d/e/f |
| 0001 | 0000\_0000\_0000\_0010 | 1 | b/c |
| 0010 | 0000\_0000\_0000\_0100 | 2 | a/b/d/e/g |
| 0011 | 0000\_0000\_0000\_1000 | 3 | a/b/c/d/g |
| 0100 | 0000\_0000\_0001\_0000 | 4 | b/c/f/g |
| 0101 | 0000\_0000\_0010\_0000 | 5 | a/c/d/f/g |
| 0110 | 0000\_0000\_0100\_0000 | 6 | a/c/d/e/f/g |
| 0111 | 0000\_0000\_1000\_0000 | 7 | a/b/c |
| 1000 | 0000\_0001\_0000\_0000 | 8 | a/b/c/d/e/f/g |
| 1001 | 0000\_0010\_0000\_0000 | 9 | a/b/c/d/f/g |
| 1010 | 0000\_0100\_0000\_0000 | A (b10) | a/b/c/e/f/g |
| 1011 | 0000\_1000\_0000\_0000 | b (b11) | c/d/e/f/g |
| 1100 | 0001\_0000\_0000\_0000 | C (b12) | a/d/e/f |
| 1101 | 0010\_0000\_0000\_0000 | d (b13) | b/c/d/e/g |
| 1110 | 0100\_0000\_0000\_0000 | E (b14) | a/d/e/f/g |
| 1111 | 1000\_0000\_0000\_0000 | F (b15) | a/e/f/g |

a = b0|b2|b3|b5|b6|b7|b8|b9|b10|b12|b14|b15

b = b0|b1|b2|b3|b4|b7|b8|b9|b10|b13

c = b0|b1|b3|b4|b5|b6|b7|b8|b9|b10|b11|b13

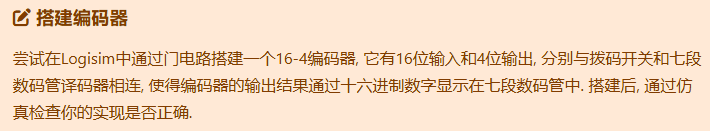
d = b0|b2|b3|b5|b6|b8|b9|b11|b12|b13|b14

e = b0|b2|b6|b8|b10|b11|b12|b13|b14|b15

f = b0 |b4|b5|b6|b8|b9|b10|b11|b12|b14|b15

g= b2|b3|b4|b4|b6|b8|b9|b10|b11|b13|b14|b15

**1.1.d 搭建编码器**



|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| a15 | a14 | a13 | a12 | a11 | a10 | a9 | a8 | a7 | a6 | a5 | a4 | a3 | a2 | a1 | a0 |  | y3 | y2 | y1 | y0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |  | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |  | 0 | 0 | 0 | 1 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |  | 0 | 0 | 1 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |  | 0 | 0 | 1 | 1 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |  | 0 | 1 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |  | 0 | 1 | 0 | 1 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |  | 0 | 1 | 1 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |  | 0 | 1 | 1 | 1 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |  | 1 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |  | 1 | 0 | 0 | 1 |
| 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |  | 1 | 0 | 1 | 0 |
| 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |  | 1 | 0 | 1 | 1 |
| 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |  | 1 | 1 | 0 | 0 |
| 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |  | 1 | 1 | 0 | 1 |
| 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |  | 1 | 1 | 1 | 0 |
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |  | 1 | 1 | 1 | 1 |
| 其他情况 | | | | | | | | | | | | | | | |  | X | X | X | X |

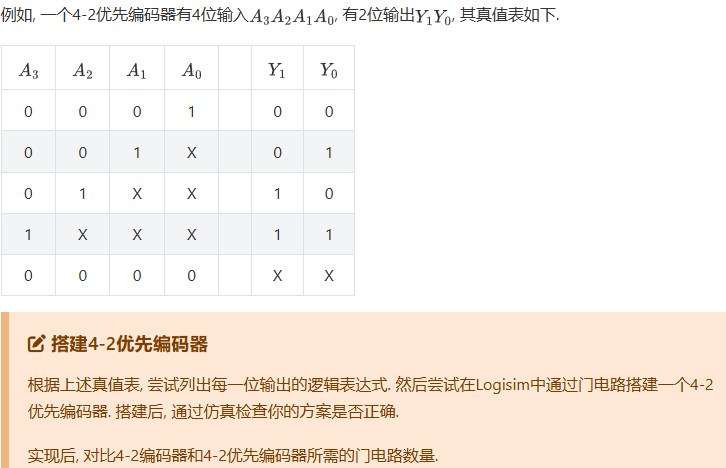
y3 = a8|a9|a10|a11|a12|a13|a14|a15

y2 = a4|a5|a6|a7|a12|a13|a14|a15

y1 = a2|a3|a6|a7|a10|a11|a14|a15

y0 = a1|a3|a5|a7|a9|a11|a13|a15

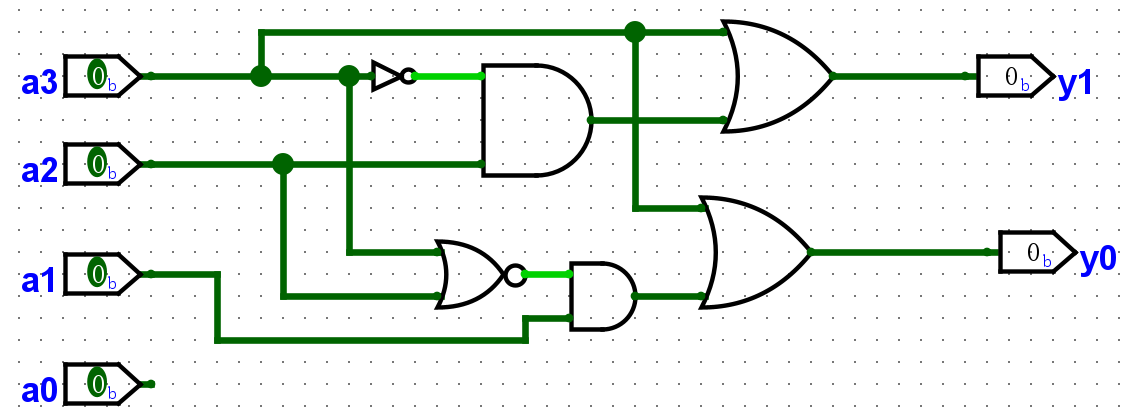
**1.1.e 搭建4-2优先编码器**



y1 = ～a3&a2 | a3

y0 = ～a3&～a2&a1 | a3

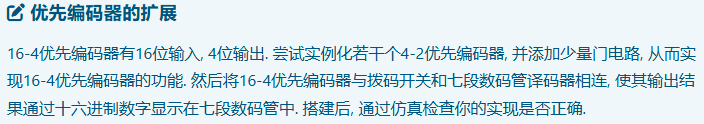
= (～(a3|a2))&a1 | a3



4-2编码器需要2个或门

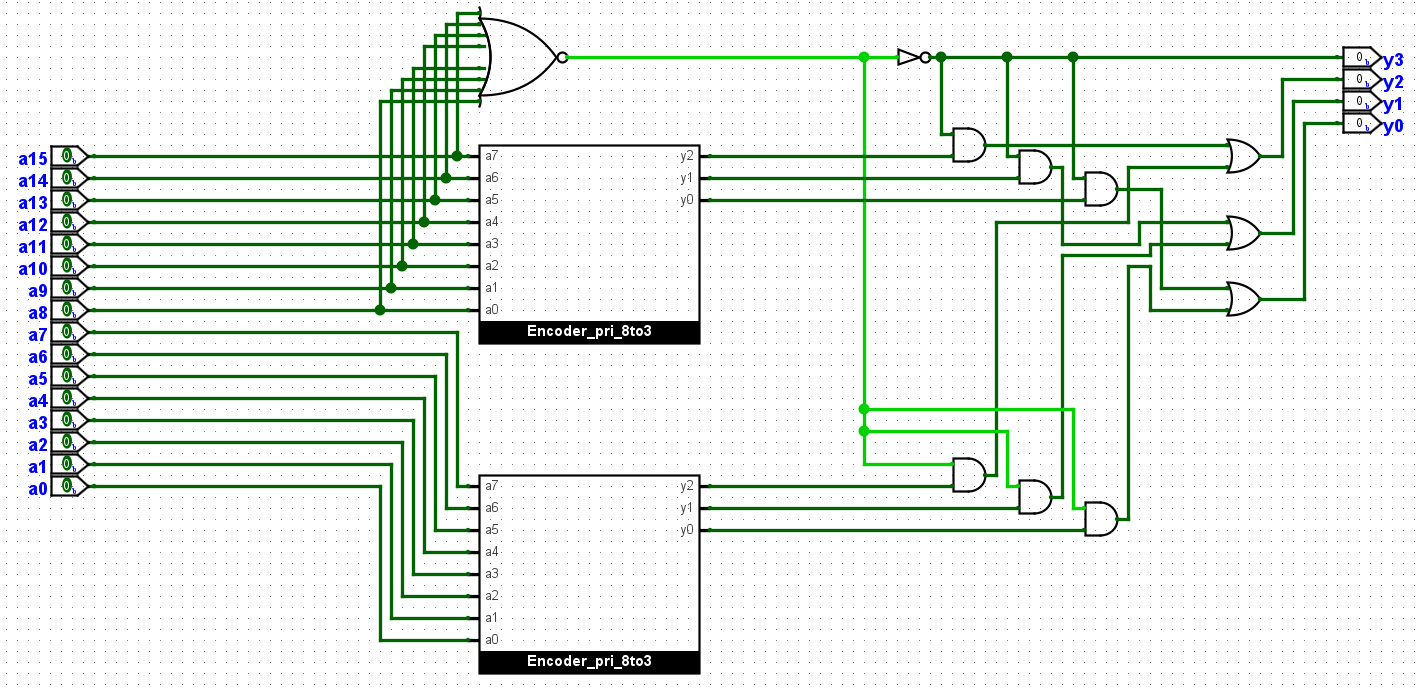
4-2优先编码区需要2个或门+2个与门+1个非门+1个或非门 = 6

**1.1.f 优先编码器的扩展**

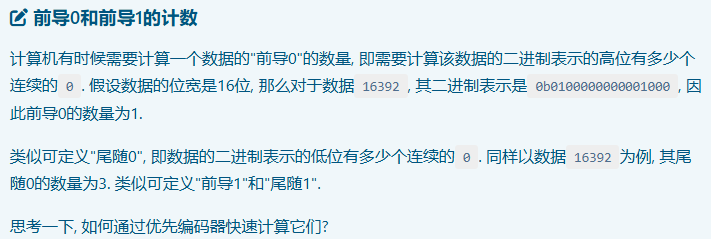


16-4优先编码器真值表

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| a15 | a14 | a13 | a12 | a11 | a10 | a9 | a8 | a7 | a6 | a5 | a4 | a3 | a2 | a1 | a0 |  | y3 | y2 | y1 | y0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |  | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | x |  | 0 | 0 | 0 | 1 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | x | X |  | 0 | 0 | 1 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | x | x | x |  | 0 | 0 | 1 | 1 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | x | x | x | x |  | 0 | 1 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | x | x | x | x | x |  | 0 | 1 | 0 | 1 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | x | x | x | x | x | x |  | 0 | 1 | 1 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | x | x | x | x | x | x | x |  | 0 | 1 | 1 | 1 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | x | x | x | x | x | x | x | x |  | 1 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 1 | x | x | x | x | x | x | x | x | x |  | 1 | 0 | 0 | 1 |
| 0 | 0 | 0 | 0 | 0 | 1 | x | x | x | x | x | x | x | x | x | x |  | 1 | 0 | 1 | 0 |
| 0 | 0 | 0 | 0 | 1 | x | x | x | x | x | x | x | x | x | x | x |  | 1 | 0 | 1 | 1 |
| 0 | 0 | 0 | 1 | x | x | x | x | x | x | x | x | x | x | x | x |  | 1 | 1 | 0 | 0 |
| 0 | 0 | 1 | x | x | x | x | x | x | x | x | x | x | x | x | x |  | 1 | 1 | 0 | 1 |
| 0 | 1 | x | x | x | x | x | x | x | x | x | x | x | x | x | x |  | 1 | 1 | 1 | 0 |
| 1 | x | x | x | x | x | x | x | x | x | x | x | x | x | x | x |  | 1 | 1 | 1 | 1 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |  | x | x | x | x |



**1.2.0 前导0和前导1的计数**



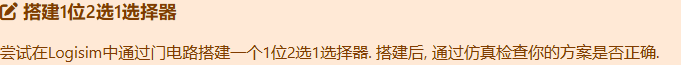
计算前导0：将数据输入对应位宽的优先编码器，假设优先编码器的输出值为x，那么前导0的数量为（位宽-1）-x，比如16392对应的数据位宽是16，其二进制值为0b0100\_0000\_0000\_1000，那么将其输入优先编码器其输出为14，前导0的数量为16-1-14=1

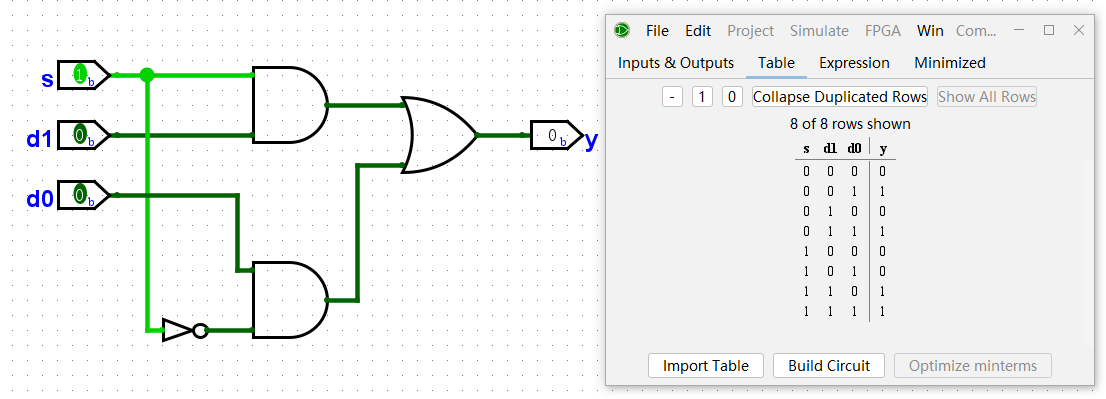
计算尾随0，将数据先倒序排列，再输入优先编码器计算，比如16392的二进制值为0b0100\_0000\_0000\_1000，先将其倒叙排列0b\_0001\_0000\_0000\_0010，输入优先编码器，得到12，16-1-12=3

计算前导1，先将数据按位取反，然后再输入优先编码器，比如数据61442，对应的二进制为0b1111\_0000\_0000\_0010，按位取反为0b0000\_1111\_1111\_1101，输入优先编码器，得到输出为11，16-1-11=4，所以前导1数量为4。

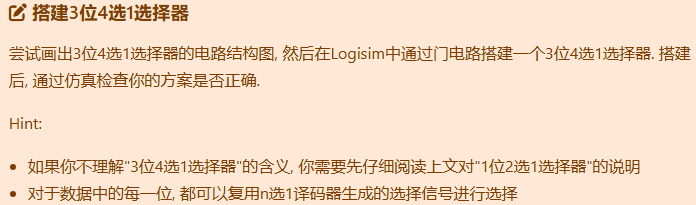
计算尾随1，参考计算尾随0，先将数据倒叙排列，再按位取反，再输入优先编码器，最后计算尾随1的数量。

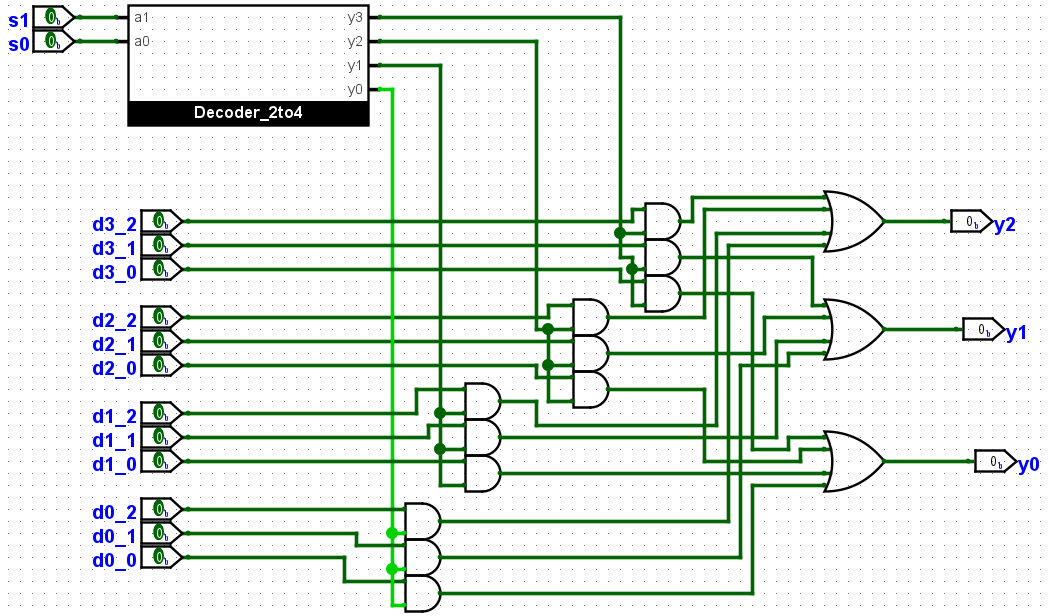
**1.2.1 搭建1位2选1选择器**



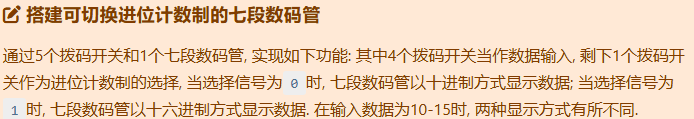


**1.2.2 搭建3位4选1选择器**



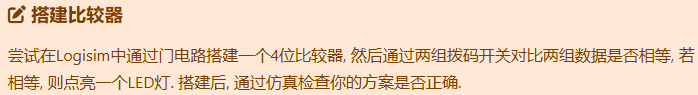


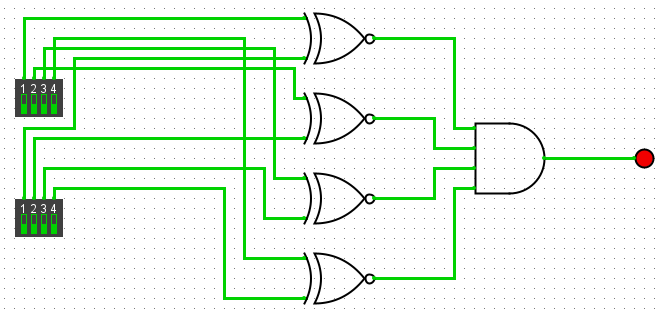
**1.2.3 搭建可切换进位计数制的七段数码管**



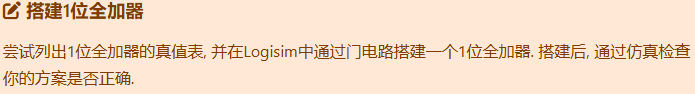
见 1.2.3\_7-Segment\_ctl3.circ

**1.2.4 搭建比较器**

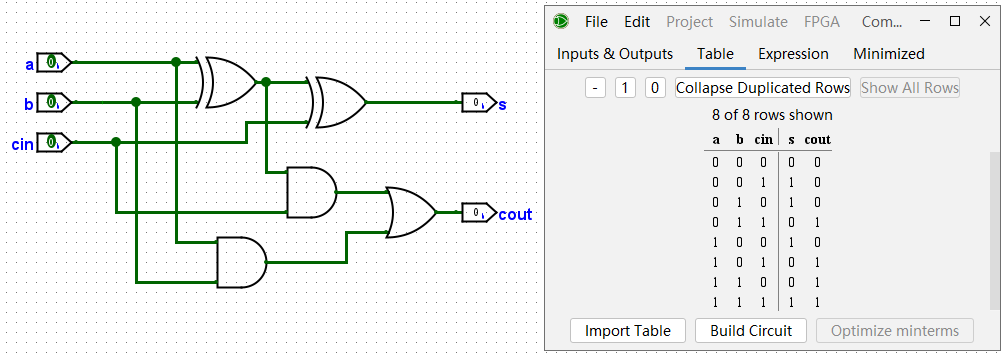




**1.2.5 搭建1位全加器**



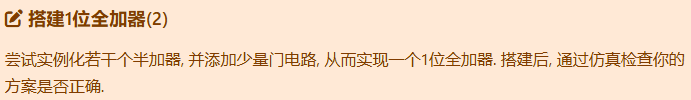
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| a | b | cin |  | s | cout |
| 0 | 0 | 0 |  | 0 | 0 |
| 0 | 0 | 1 |  | 1 | 0 |
| 0 | 1 | 0 |  | 1 | 0 |
| 0 | 1 | 1 |  | 0 | 1 |
| 1 | 0 | 0 |  | 1 | 0 |
| 1 | 0 | 1 |  | 0 | 1 |
| 1 | 1 | 0 |  | 0 | 1 |
| 1 | 1 | 1 |  | 1 | 1 |

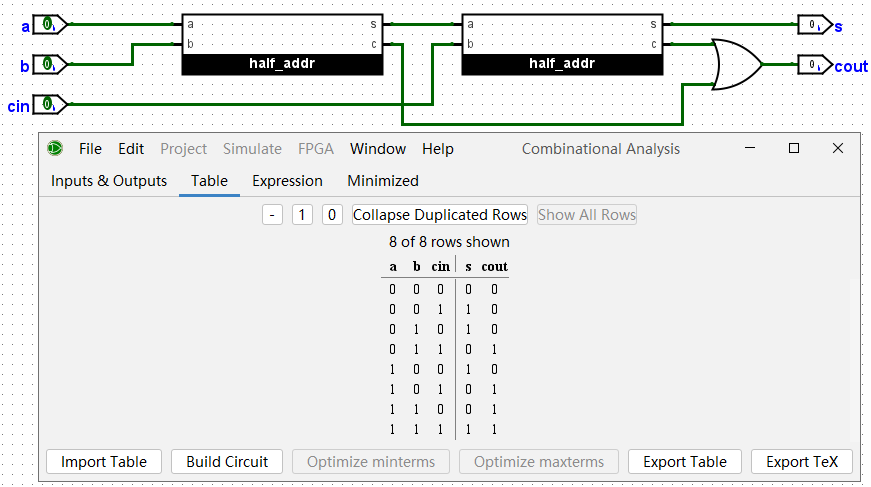


s = cin ^ (a ^ b)

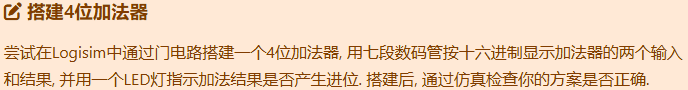
cout = (cin & (a ^ b)) | (a & b)

**1.2.6 搭建1位全加器(2)**



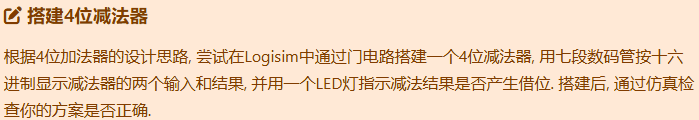


**1.2.7 搭建4位加法器**



见 1.2.7\_full\_adder\_4bit.circ

**1.2.8 搭建4位减法器**



1位全减器的真值表

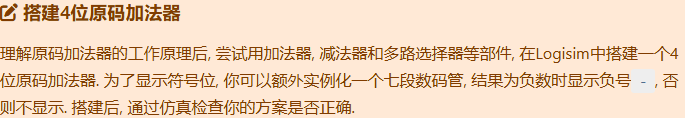
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| a(被减数) | b(减数) | cin  (低位是否向本位借位) |  | d | cout  (本位是否向高位借位) |
| 0 | 0 | 0 |  | 0 | 0 |
| 0 | 0 | 1 |  | 1 | 1 |
| 0 | 1 | 0 |  | 1 | 1 |
| 0 | 1 | 1 |  | 0 | 1 |
| 1 | 0 | 0 |  | 1 | 0 |
| 1 | 0 | 1 |  | 0 | 0 |
| 1 | 1 | 0 |  | 0 | 0 |
| 1 | 1 | 1 |  | 1 | 1 |

d = (a ^ b) ^ cin

cout = ((a ^ b) & cin) | ((～a) & b)

见 1.2.8\_full\_sub\_4bit.circ

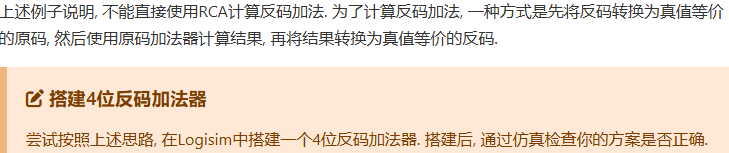
**1.2.9 搭建4位原码加法器**



|  |  |  |
| --- | --- | --- |
| 判断条件 | 数据处理方式 | 结果符号 |
| a3==0，b3==0（两数皆正数） | a[3:0] + b[3:0] | 正 |
| a3==1，b3==1（两数皆负数） | a[3:0] + b[3:0] | 负 |
| a3==1，b3==0，a[2:0]的绝对值大于b[2:0]  a[3:0]=-5,b[3:0]=4 | a[2:0] - b[2:0] | 负 |
| a3==1，b3==0，a[2:0]的绝对值不大于b[2:0]  a[3:0]=-4,b[3:0]=4  a[3:0]=-3,b[3:0]=4 | b[2:0] - a[2:0] | 正 |
| a3==0，b3==1，a[2:0]的绝对值大于b[2:0]  a[3:0]=5,b[3:0]=-4 | a[2:0] - b[2:0] | 正 |
| a3==0，b3==1，a[2:0]的绝对值不大于b[2:0]  a[3:0]=4,b[3:0]=-4  a[3:0]=3,b[3:0]=-4 | b[2:0] - a[2:0] | 负  （如果a[2:0]的绝对值等于b[2:0]则为正） |

见 1.2.9\_adder\_4bit\_sign-and-magnitude.circ

**1.2.a 搭建4位反码加法器**



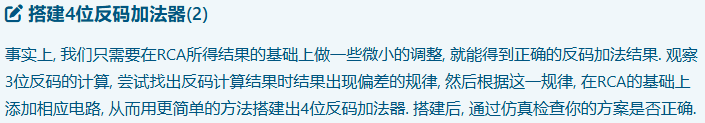
先设计一个反码转原码的电路，然后在将转换后的数据送到原码加法器中即可

**反码转原码逻辑设计表**

|  |  |
| --- | --- |
| 判断条件 | 数据处理方式 |
| a3==0 | y3=0,y[2:0] = a[2:0] |
| a3==1且a[2:0]==111 | y3=0,y[2:0] = ～a[2:0] |
| a3==1且a[2:0]有任意一个bit为0 | y3=1，y[2:0] = ～a[2:0] |

见 1.2.a\_adder\_4bit\_one's complement.circ

**1.2.b 搭建4位反码加法器(2)**



分析采4bit反码进行RCA加法时的结果规律，有以下结论

|  |  |  |
| --- | --- | --- |
| 判断条件 | 数据处理方式 | 结果符号 |
| a3==0，b3==0（两数皆正数） | a[3:0] + b[3:0] （00）（y0） | 正 |
| a3==1，b3==1（两数皆负数） | ～（a[3:0] + b[3:0] + 1）（11）（y3） | 负（若结果为0则不显示7段数码管的负号） |
| a3==1，b3==0，且a[2:0]真值的绝对值大于b[2:0]真值的绝对值 | ～（a[3:0] + b[3:0]）（01）(y1) | 负 |
| a3==1，b3==0，且a[2:0]真值的绝对值小于等于b[2:0]真值的绝对值 | （a[3:0] + b[3:0] + 1）（10）(y2) | 正 |
| a3==0，b3==1，且a[2:0]真值的绝对值大于等于b[2:0]真值的绝对值 | （a[3:0] + b[3:0] + 1）（10）(y2) | 正 |
| a3==0，b3==1，且a[2:0]真值的绝对值小于b[2:0]真值的绝对值 | ～（a[3:0] + b[3:0]）（01）(y1) | 负 |

见 1.2.b\_adder\_4bit\_one's complement2.circ

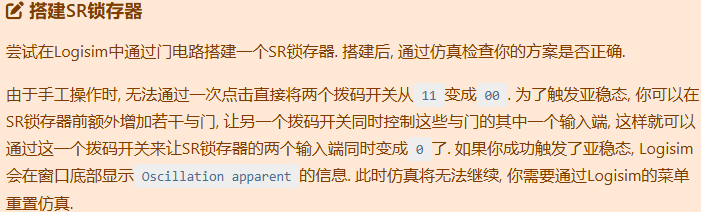
**1.2.c 检测补码加法是否发生溢出**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| An-1 | Bn-1 | Cn-1 |  | Cn | Sn-1 | 溢出 |
| 0 | 0 | 0 |  | 0 | 0 | 否 |
| 0 | 0 | 1 |  | 0 | 1 | 是 |
| 0 | 1 | 0 |  | 0 | 1 | 否 |
| 0 | 1 | 1 |  | 1 | 0 | 否 |
| 1 | 0 | 0 |  | 0 | 1 | 否 |
| 1 | 0 | 1 |  | 1 | 0 | 否 |
| 1 | 1 | 0 |  | 1 | 0 | 是 |
| 1 | 1 | 1 |  | 1 | 1 | 否 |

assign over\_flow = (～An-1) & (～Bn-1) & Sn-1 | (An-1 & Bn-1 & ～Sn-1)

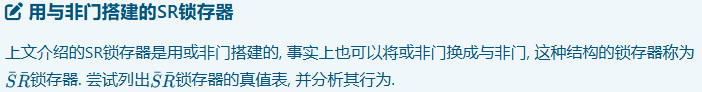
见 1.2.c\_adder\_4bit\_two's complement.circ

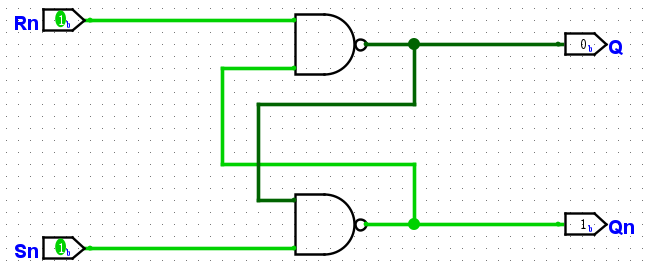
**1.2.d 搭建SR锁存器**



见 1.2.d\_sr-latch.circ

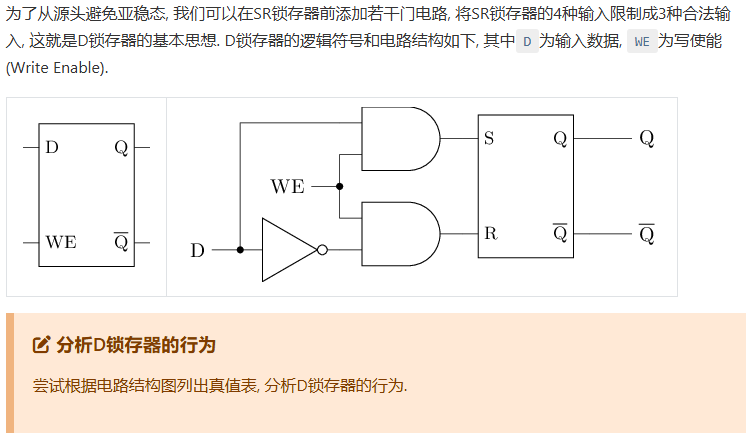
**1.2.e 用与非门搭建的SR锁存器**





|  |  |  |  |
| --- | --- | --- | --- |
| Rn | Sn |  | Q |
| 0 | 0 |  | 禁止 |
| 0 | 1 |  | 1 |
| 1 | 0 |  | 0 |
| 1 | 1 |  | 保持 |

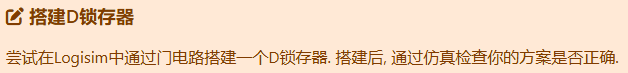
**1.2.f 分析D锁存器的行为**



|  |  |  |  |
| --- | --- | --- | --- |
| D | WE |  | Q |
| 0 | 0 |  | 保持 |
| 0 | 1 |  | 0 |
| 1 | 0 |  | 保持 |
| 1 | 1 |  | 1 |

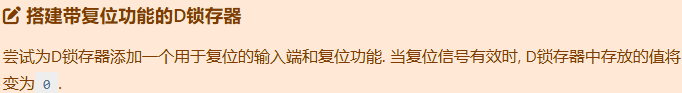
D锁存器在WE使能时，将输入端D的值传递给Q

**1.3.0 搭建D锁存器**

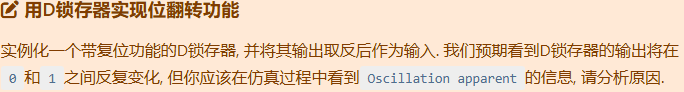


见 1.3.0\_d-latch.circ

**1.3.1 搭建带复位功能的D锁存器**

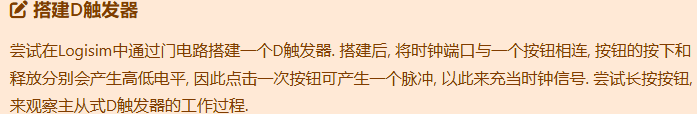


**1.3.2 用D锁存器实现位翻转功能**



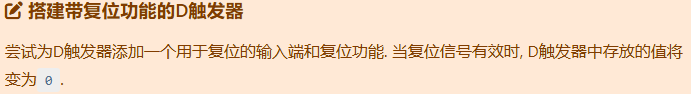
因为D锁存器WE为“1”，复位释放后，Q=0，取反后为“1”送到D端，对于内部的RS锁存器而言，相当于R端和S端同时接“1”，在之前的章节已经讨论过，该组合会导致RS锁存器出现亚稳态，所以会看到Oscillation apparent的出现。

**1.3.3 搭建D触发器**



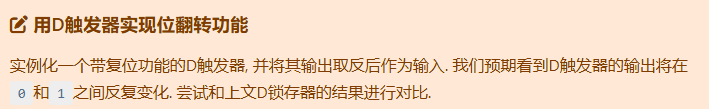
见 1.3.3\_dff.circ

**1.3.4 搭建带复位功能的D触发器**



见1.3.4\_dff\_rstn.circ

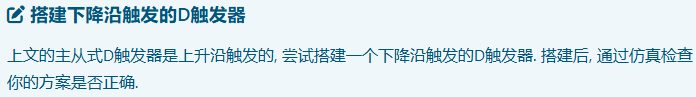
**1.3.5 用D触发器实现位翻转功能**



见1.3.5\_dff\_rstn\_inv.circ

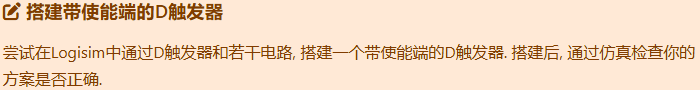
和D锁存器相比DFF没有组合逻辑环路，不会让RS锁存器出现亚稳态从而导致仿真停止

**1.3.6 搭建下降沿触发的D触发器**



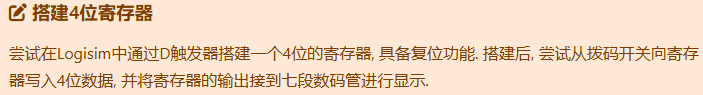
见1.3.6\_dffn\_rstn.circ

**1.3.7 搭建带使能端的D触发器**



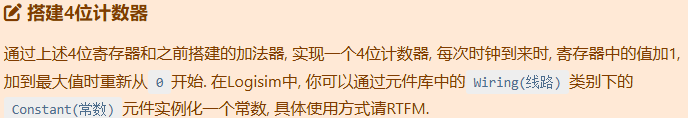
见1.3.7\_dffe\_rstn.circ

**1.3.8 搭建4位寄存器**



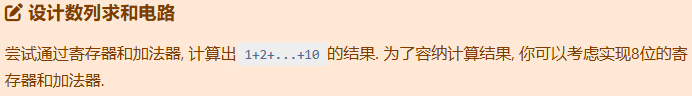
见 1.3.8\_dff\_rstn\_4bit.circ

**1.3.9 搭建4位计数器**



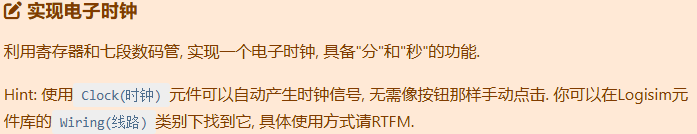
见 1.3.9\_cnt\_4bit.circ

**1.3.a 设计数列求和电路**



见 1.3.a\_sum\_1to10.circ

**1.3.b 实现电子时钟**

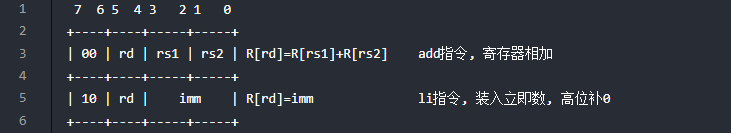


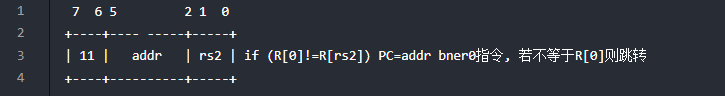
1.3.b\_elec\_clock.circ

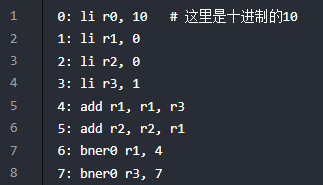
* 1. **F4**

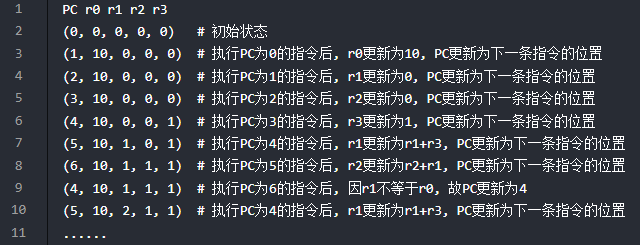
**1.2.1 继续执行上述指令**

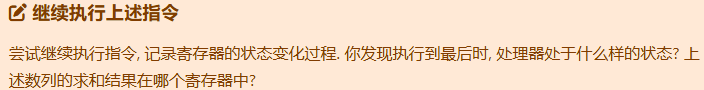
用指令来计算1+2+...+10这一数列的和。0

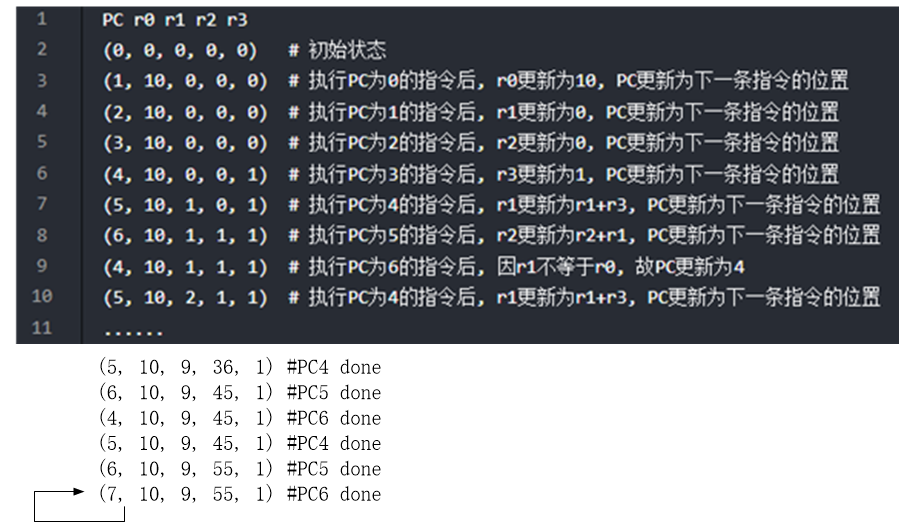






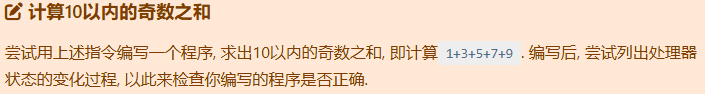






最后处理器会一直执行benr0 r3 7指令，求和结果存放在r2寄存器中

**1.2.2 计算10以内的奇数之和**



0: li r0, 9 # 这里是十进制的9

1: li r1, 1

2: li r2, 0

3: li r3, 2

4: add r2, r2, r1

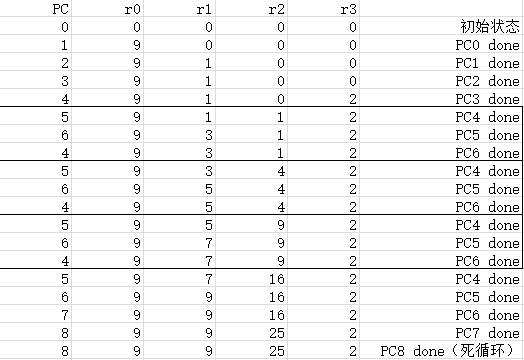
5: add r1, r1, r3

6: bner0 r1, 4

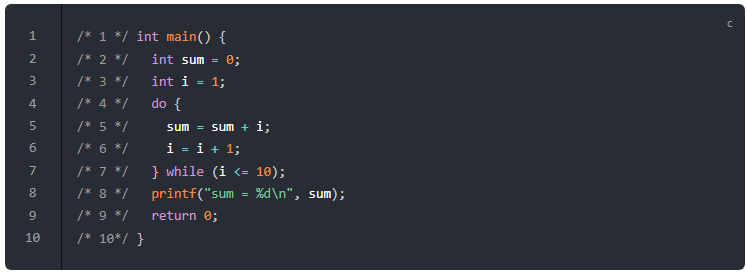
7: add r2, r2, r1

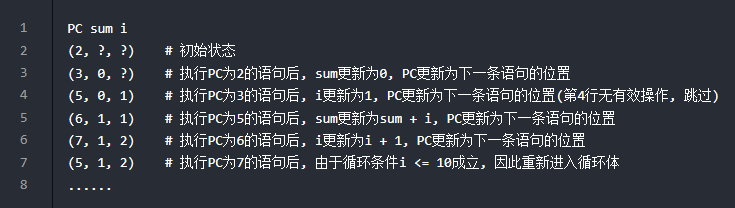
8: bner0 r3, 8

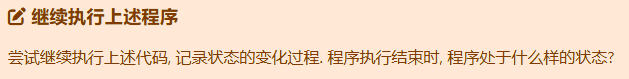
最后所求的和存放在r2寄存器中



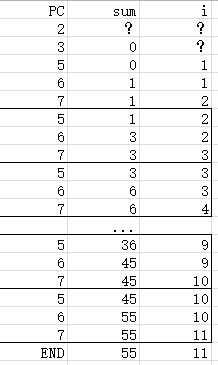
**1.2.3 继续执行上述程序**



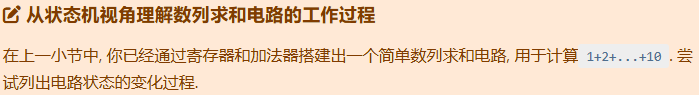


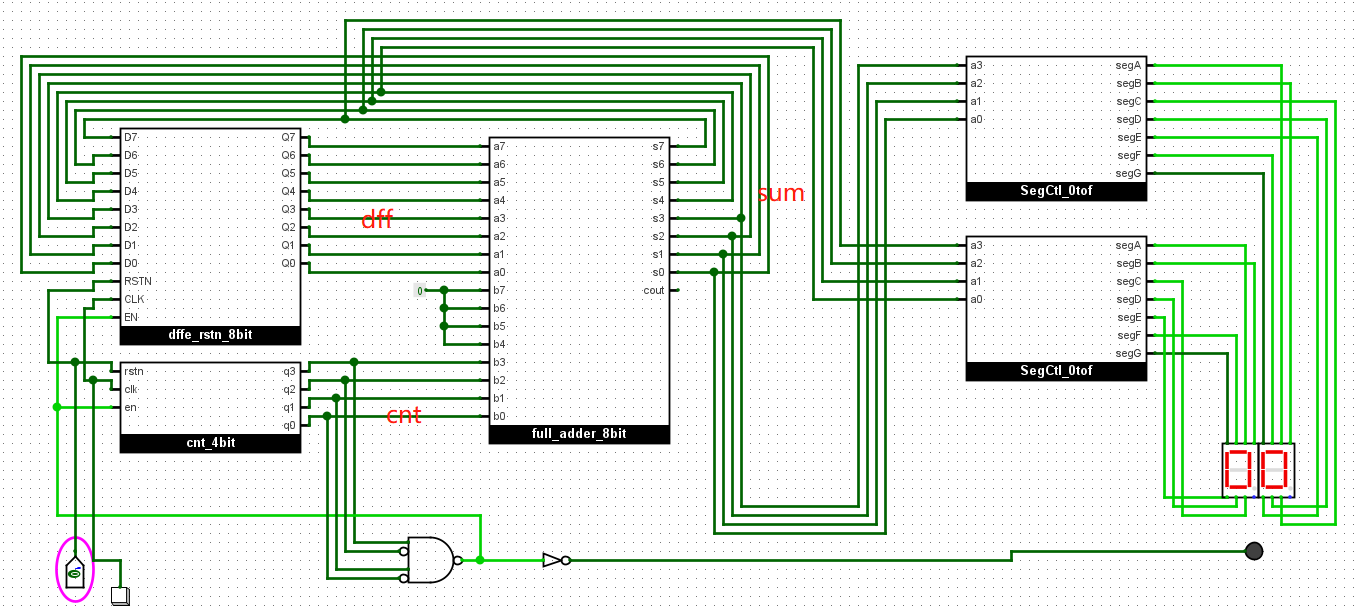


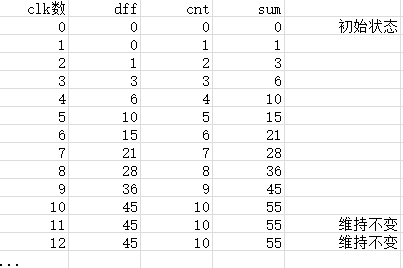
程序结束时，sum的值为55，i的值为11



**1.2.4 从状态机视角理解数列求和电路的工作过程**







* 1. **F5**

## 1.3.1 实现完整的sCPU

key draf：

选pc+1的判断条件为：

1. 不为benr0指令
2. 为benr0指令且R[0]==R[rs2]

见 F5\_1.3.1\_sCPU.circ

**参考文献**

1. https://ysyx.oscc.cc/docs/2407/f/2.html

**附 录**