



《计算机组成与结构》 实验报告

班级： 21 软工创新

姓名： 李佳睿

学号： 2021204455

2022 年 秋 季学期

实验项目	4 位海明码编码器		日期	2022/10/28	
实验环境	logisim 2.16.1.0		成绩评定		
实验目的和要求	基本实验要求： 1. 实现 4 位海明码的编码器和解码器 2. 中间添加加扰器，观察实验结果（只考虑至多一位错） 进阶实验要求： 3. 实现 16 位海明码的编码器和解码器				
	序号	实验项目	实验要求	实验源程序文件	注意事项
	1（2学时）	4 位数据海明校验码的生成与纠错	1)4 位海明码的校验位（3 位）生成 2) 7 位数据在传输中加扰（只考虑至多 1 位错） 3)4 位数据的自动纠错	4 位海明码编码器.circ	加扰器借助于随机数生成器完成。
					完成 16 位数据码校验及纠错。

1. 海明码提出的背景:

海明码是 *Richard – Hanming* 于 1950 年提出的，具有一位的纠错能力。

由编码纠错理论得知，任何一种代码是否具有检错能力和纠错能力，都与编码的最小距离有关。所谓编码的最小距离，是指在一种编码系统当中，任何两组合法代码之间的最少二进制位数的差异，我们可以使用下面的公式进行表示：

$$L-1 = D + C, \text{ 且 } D \geq C$$

海明码就是基于该原理提出的具有一位纠错能力的代码。

2. 海明码的基本思想:

将有效信息按某种规律分成若干组，每组安排一个校验位，做奇偶测试，就能提供多位检错信息，以指出最大可能是哪位出错，从而将其纠正。实质上，海明校验是一种多重校验。

3. 海明码的主要特点:

它不仅具有检测错误的能力，同时还具有给出错误所在准确位置的能力，但是因为这种海明校验的方法只能检测和纠正一位出错的情况。所以如果有多个错误，就不能查出了。

4. 海明码检错和纠错的原理:

4.1 海明码纠错的基本步骤:

1 计算校验位数 2 确定校验码位置 3 确定校验码 4 实现校验和纠错

4.1.1 计算校验位数:

假设 N 表示添加了校验位后整个信息的二进制位数，用 K 表示有效信息位数， r 表示添加的校验码位，它们之间的关系应当满足：

$$N = K + r \leq 2^r - 1$$

我们可以举一个例子，假设有效信息的位数 K 为 6 位，我们可以得到添加的校验码位至少为 4 位。由此我们可以得出有效信息位数和校验码位数之间的关系：

表5-1 信息码位数与校验码位数之间的关系

信息码位数	1	2~4	5~11	12~26	27~57	58~120	121~247
校验码位数	2	3	4	5	6	7	8

4.1.2 确定校验码位置:

校验码的位置很容易确定的，那就是校验码必须是在 2^n 次方位置，如第 1、2、4、8、16、32……位（对应 20、21、22、23、24、25，……，是从最左边的位数起的），这样一来就知道了信息码的分布位置，也就是非 2^n 次方位置，如第 3、5、6、7、9、10、11、12、13，……位（是从最左边的位数起的）

4.1.3 确定校验码:

经过前面的两步，我们已经确定了所需的校验码位数和这些校验码的插入位置，我们还需要确定各个校验码值。这些校验码的值不是随意的，每个校验位的值代表了代码字中部分数据位的奇偶性（最终要根据是采用奇校验，还是偶校验来确定），其所在位置决定了要校验的比特位序列。总的原则是：第 i 位校验码从当前位开始，每次连续校验 i （这里是数值 i ，

不是第 i 位，下同)位后再跳过 i 位，然后再连续校验 i 位，再跳过 i 位，以此类推。最后根据所采用的是奇校验，还是偶校验即可得出第 i 位校验码的值。

4.1.4 实现校验和纠错:

5. 基础实验的具体操作:

5.1 4 位海明码的编码电路的实现:

通过 4 位输入，可以得到 7 位的输出

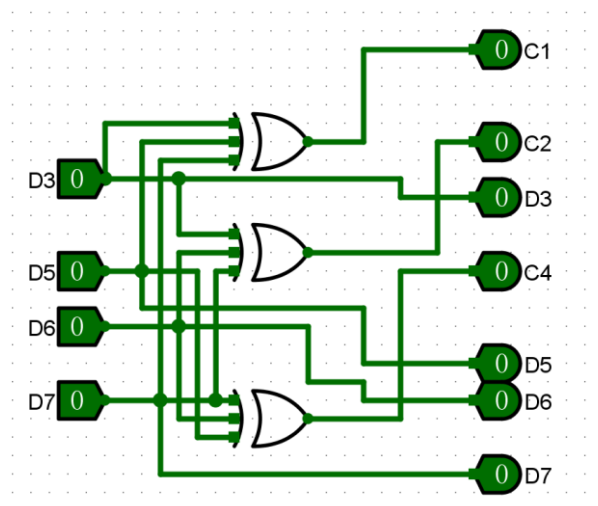


图 1 4 位海明码的编码电路

5.2 4 位海明码解码电路的实现:

通过 7 位输入，可以实现 4 位输出

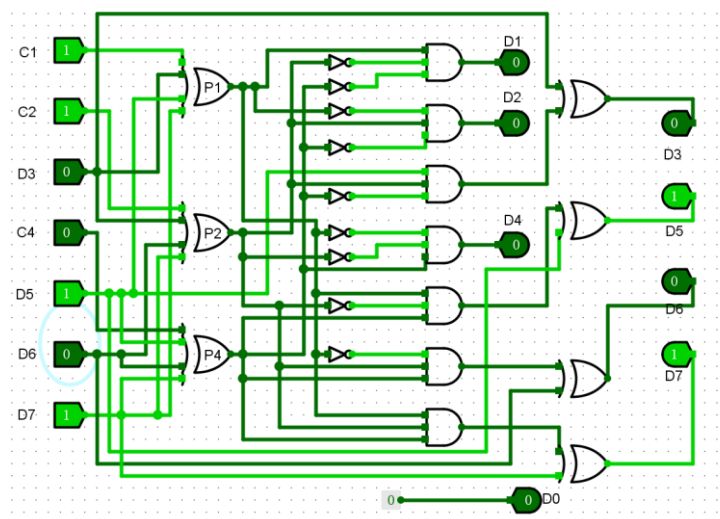


图 2 4 位海明码的解码电路

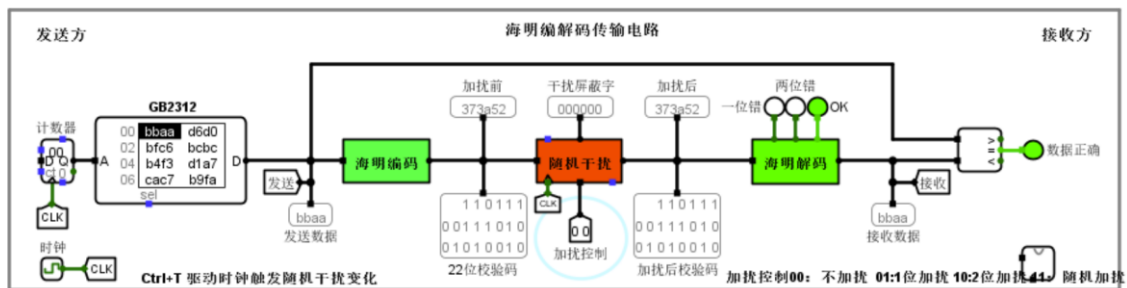


图 3 海明码编码和解码基本原理图

上面我们实现了 4 位海明码的编码和解码的基本电路，但是在中间还希望能够实现 1 位加扰，于是我实现了 7 位的随机加扰器。

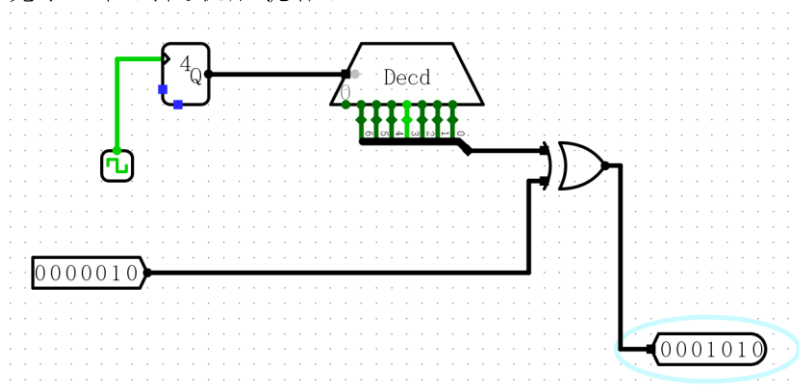


图 4 随机加扰器的基本实现电路

整体的电路实现如下：

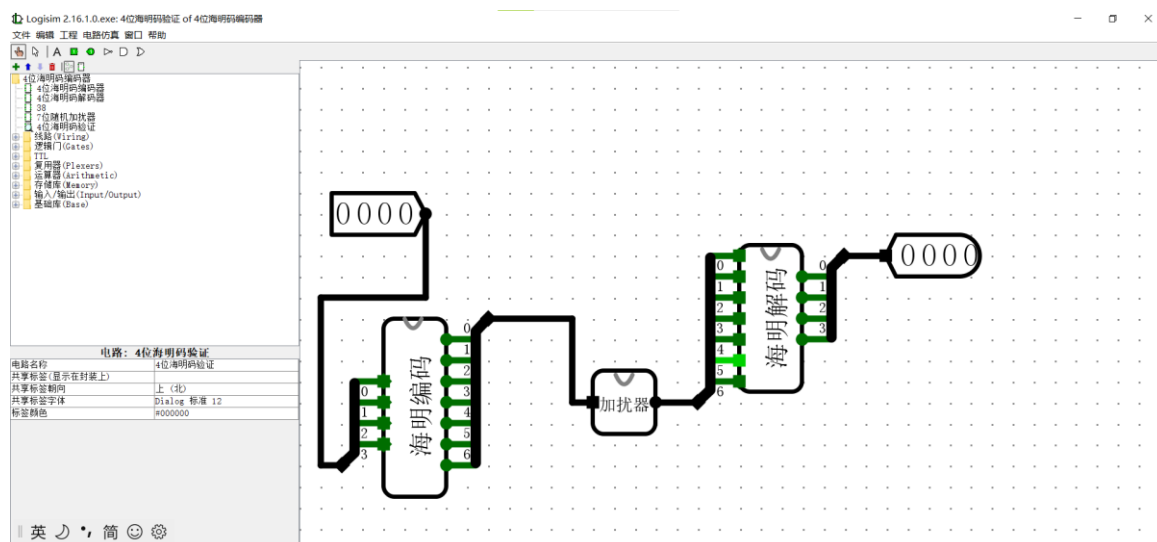


图 5 整体的电路实现

由于中间电路是由 38 译码器实现的，于是我又实现了 38 译码器的电路：

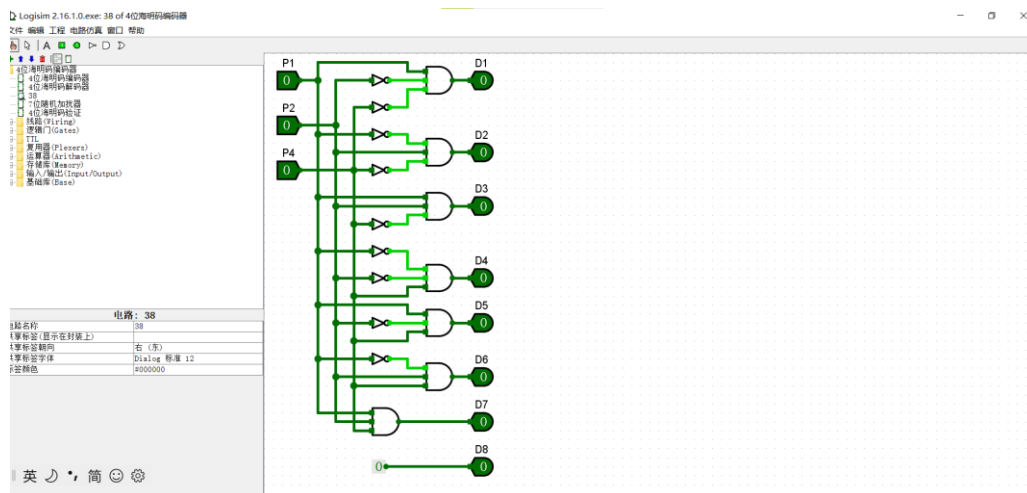


图 6 38 译码器的实现电路

最后我实现了海明码的解码器的基本电路：

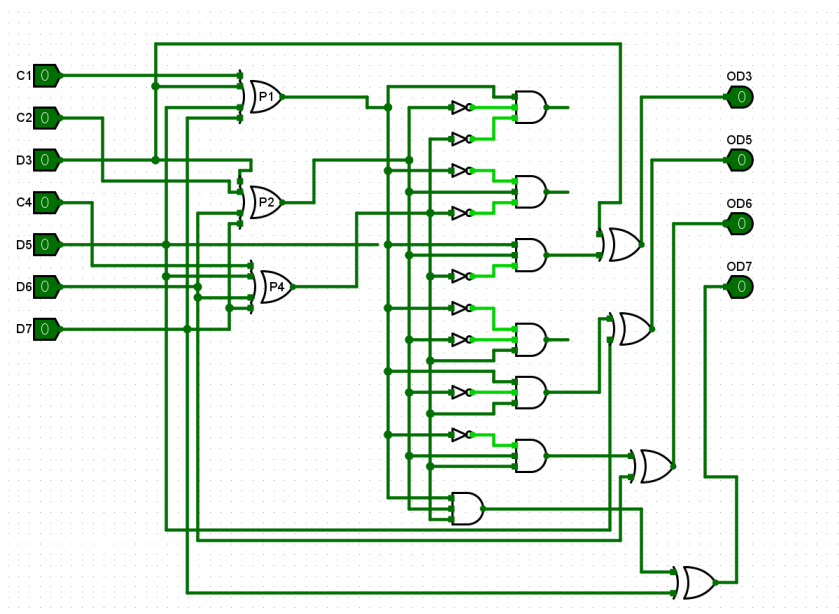


图 7 海明码的解码电路

6. 进阶实验的具体操作：

6.1 实验的基本要求：

该实验要求我们对 22 位海明编码进行解码成原来的 16 位，且判断是否出错，有一位错，两位错，还有无错。对于一位错，我们要进行纠正。

6.2 判断是否出错：

设置有六位指错字， $G_i (i=1,2,3,4,5,6)$ ，其中 $G_5G_4G_3G_2G_1$ 是进行判断是否数据出错， G_6 是总的奇偶校验位，可以用来检验数据是一位错，还是两位错。

P1	B1,B2,B4,B5,B7,B9,B11,B12,B14,B16
P2	B1,B3,B4,B6,B7,B10,B11,B13,B14
P3	B2,B3,B4,B8,B9,B10,B11,B15,B16
P4	B5,B6,B7,B8,B9,B10,B11
P5	B12,B13,B14,B15,B16
P6	全部数据的异或，即前面 21 位数据的异或

图 8 已知的条件

G1	P1,B1,B2,B4,B5,B7,B9,B11,B12,B14,B16
G2	P2,B1,B3,B4,B6,B7,B10,B11,B13,B14
G3	P3,B2,B3,B4,B8,B9,B10,B11,B15,B16
G4	P4,B5,B6,B7,B8,B9,B10,B11
G5	P5,B12,B13,B14,B15,B16
G6	全部数据的异或非

图 9 判断的数位

6.3 检错的原理：

当时， $G_5G_4G_3G_2G_1 = 0$ 表示数据无传输错误，否则，表示出错。

当发生一位数据错误时， $G_5G_4G_3G_2G_1$ 所指示的数据，表示那位数据出错（例如 $G_5G_4G_3G_2G_1 = 00101$ ，则表示第 5 位数据出错）。

当发生两位数据错误时， $G_5G_4G_3G_2G_1$ 仍不为 0，由于只能纠正 1 位错误，故该海明编码是尽努力去纠正。

因此，增加一位总的奇偶校验位 G_6 ，进行判断是一位错，还是两位错。

- ① 当 $G_5G_4G_3G_2G_1 = 0$ ， $G_6 = 0$ ，表示数据无出错；
- ② 当 $G_5G_4G_3G_2G_1 \neq 0$ ， $G_6 = 1$ ，表示发生一位数据出错；
- ③ 当 $G_5G_4G_3G_2G_1 = 0$ ， $G_6 = 1$ ，表示该奇偶校验位出错，即发生一位出错；
- ④ 当 $G_5G_4G_3G_2G_1 \neq 0$ ， $G_6 = 1$ ，表示发生两位数据出错。

6.4 纠正数据：

在纠正数据，得到原始 16 位数据这块设计上，可采用解码器对出错的位进行输出，再与原来的数据进行异或，即可得纠正后的数据。

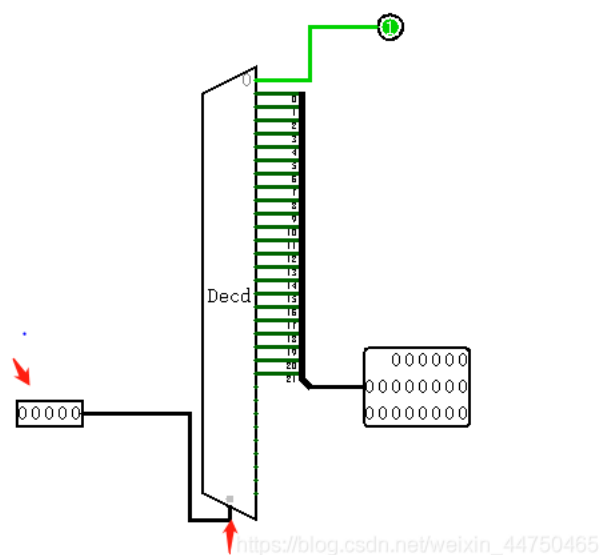


图 10 解码器功能实现电路图

注：左边箭头指向的为输入数据，是 5 位的二进制数据，第二个箭头是选择哪个输出为 1，此时由于它连接了 5 位二进制数据，则该 *Decd* 解码器的输出端有 2^5 个输出端。此时数据为 00000 对应的十进制为 0，则可从图中看出第 0 位输出是 1，其他输出为 0。

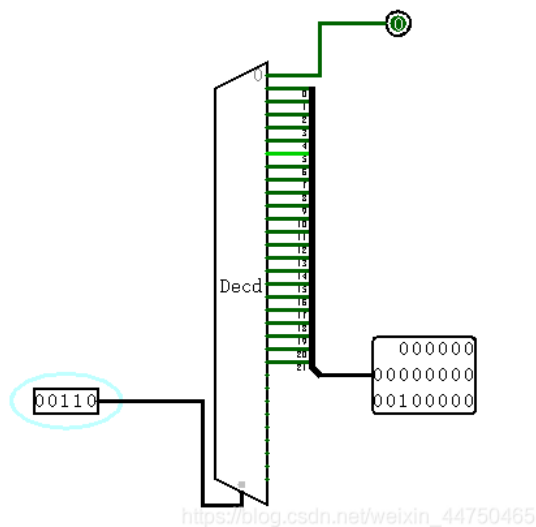


图 11 解码器功能实现电路图

注：此时输入数据为 00110 对应十进制为 6，则可从图中看出，第六位输出为 1，其他输出为 0。

因此如果输入数据为 0 的话，则第 0 位输出为 1，否则第 0 位输出为 0，因此我们可以将输入数据换成 $G_5G_4G_3G_2G_1$ ，用第 0 位输出判断 $G_5G_4G_3G_2G_1$ 是否为 0 ($G_5G_4G_3G_2G_1=0$ ，则第 0 为输出为 1)，或者不为 0 ($G_5G_4G_3G_2G_1 \neq 0$ ，则第 0 位输出为 0)。

纠正数据：将输入数据换成 $G_5G_4G_3G_2G_1$ ，将该解码器的第 1 位至 23 位用分离器连接起来，如上图的右边部分所示（此时，如果 $G_5G_4G_3G_2G_1$ 等于某个不为 0 的数据时，该数据将会输出为 1）。将该 22 位数据与原来的海明码进行按位异或，则可得到原来无出错的数据，即进行纠正。（提示： x 异或 $0 = x$ ， x 异或 $1 = x$ 的非）

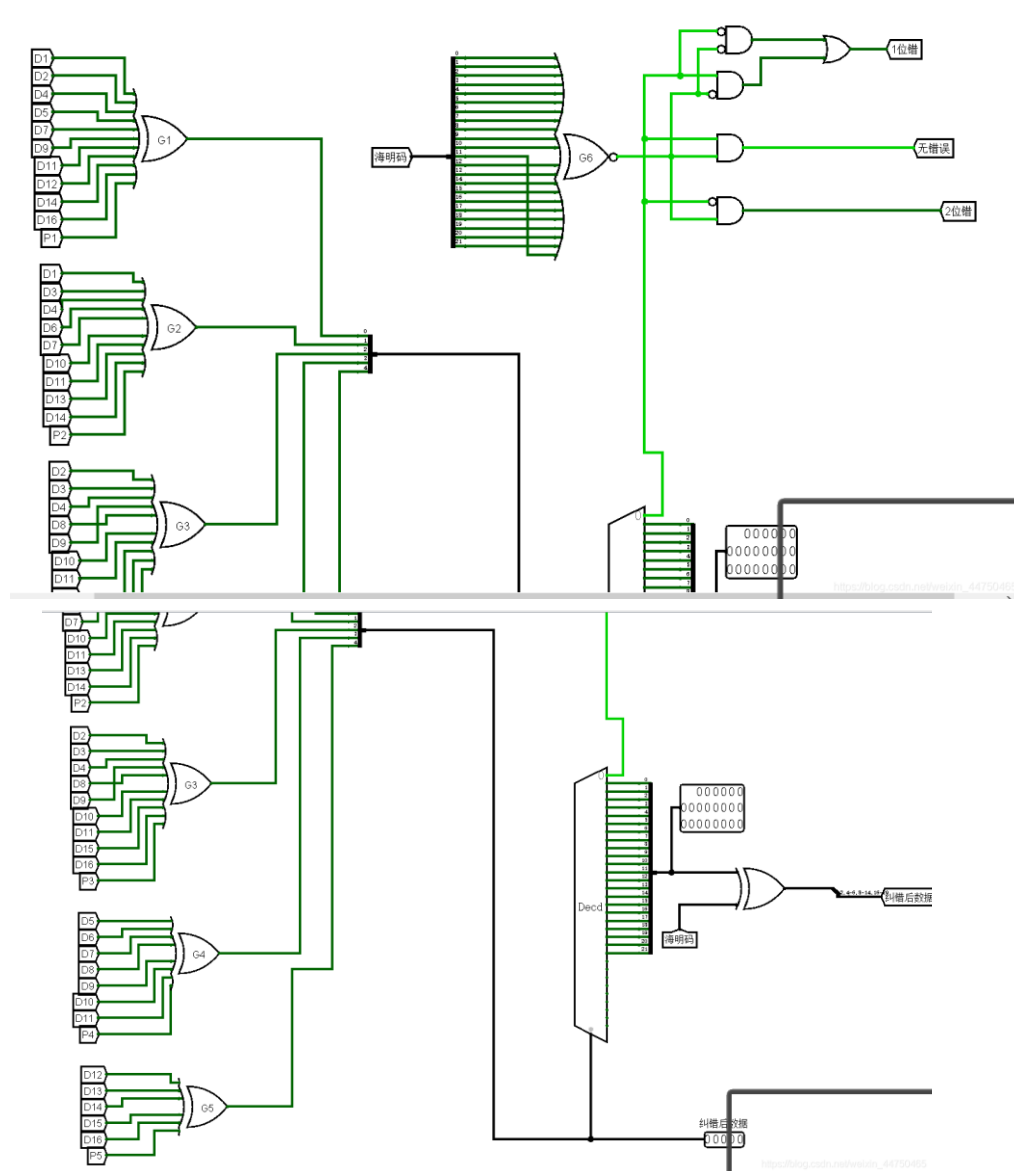


图 12 基本电路实现 1

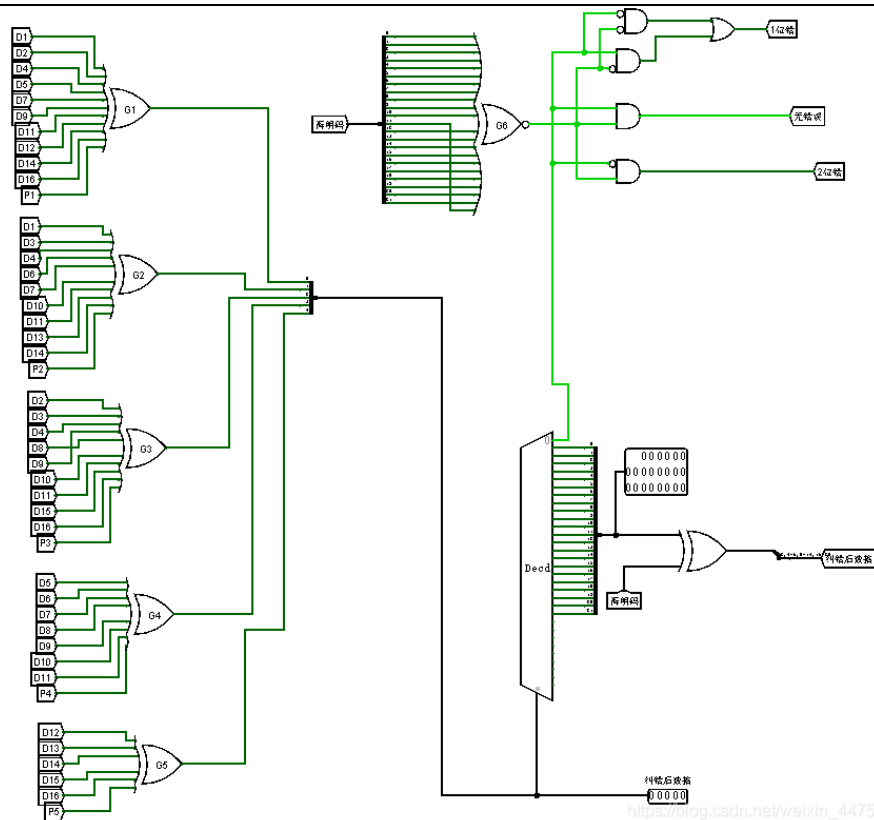


图 13 基本电路实现 2

由于要获得纠正后的数据，即 16 位数据，由于该数据是从 0 开始的，故其检验位分别位于 0, 1, 3, 7, 15, 21 上，因此要去掉这些位置的数据，即可得到原来的 16 位数据。因此采用分离器输出的时候，位宽仍然是 22 位，但是位 0, 位 1, 位 3, 位 7, 位 15, 位 21，是没有的。

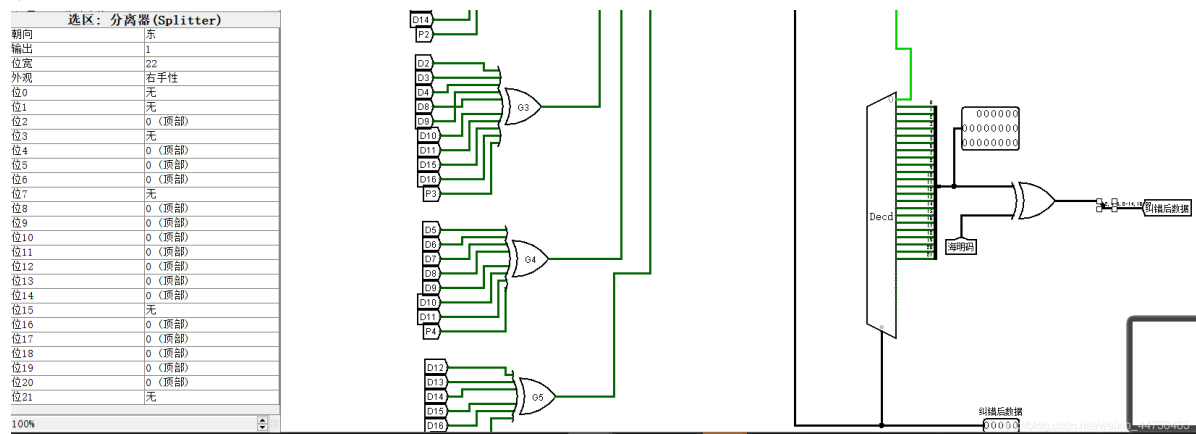


图 14 基本电路实现 3

思考 和 体会	<p>本次实验是本学期《计算机组成与结构》第一次的项目实验，第一次的实验是从课本上海明码出发的，要求实现海明码的检错和纠错电路，最初的任务目标是实现 4 位海明码的检错和纠错电路，通过一个随机加扰器进行一位偏差，再通过海明码的解码器进行纠错，实现了相关的功能操作，同时也加深了对课本上知识的理解。但是对于 16 位海明码的检错和纠错电路的实现，具有一定的难度，因此参考了 CSDN 上的博客，对相关的操作也加深了理解。</p>
备注	

实验项目	扩容器的扩展				日期	2022/11/03
实验环境	logisim 2.16.1.0				成绩评定	
实验目的和要求	用基本的门电路连接实现，达到存储容量扩展的目的 1. 实现存储容量扩展的字扩展 2. 实现存储容量扩展的位扩展 3. 实现存储容量扩展的字位扩展 4. 注：ROM 存储器的内容可以从（参考字库）中复制					
	2（2学时）	存储器的扩容	实现存储器的字扩展	storage-2020.circ（字库电路）	ROM 存储器的内容可以从（参考字库）中复制	将（字库测试）中的内容改为你的宿舍号+所有舍友的名字

1. 基本实现原理:

存储信息一般是存储在存储器（ROM、RAM）上的。在实际应用中，经常出现一片 ROM 或 RAM 芯片不能满足对存储器容量需求的情况，这就需要若干片 ROM 或 RAM 组合起来形成一个存储容量更大的存储器。而组合方式有字扩展和位扩展两种。

用多片位宽相同的存储器（ROM 或 RAM）芯片扩展包含更多存储器的过程。一般是在每个字的位数够而字的数目不够时使用。

2. 存储器的寻址方法:

2.1 线选法:

简单微机系统存储容量不大，存储器芯片数也不多，可用单根地址线作为片选信号，每个存储芯片或每个 I/O 端口只用一根地址线选通。

2.2 全译码片选法:

将低位地址总线直接连至各芯片的地址线，余下高位地址总线全部参加译码，译码输出作为各芯片片选信号。

2.3 局部译码片选法:

只对部分高位地址总线译码产生片选信号，剩余高位线或空或直接用做其他芯片片选信号，是介于全译码片选法和线选法间的寻址方法。

3. 各种扩展方法介绍:

3.1 字扩展:

数据位宽不变，通过增加存储单元个数来增加存储容量

3.2 位扩展:

存储单元个数不变，通过扩展存储单元芯片的位数来增加存储容量

3.3 字位扩展:

既增加存储单元的个数，又同时增加数据位宽，通过同时改变存储芯片的个数和位数来增加存储容量

4. 基本的实验操作:

4.1 字扩展的基本操作:

分别选择输入引脚和分离器的设置，位宽设置注意兼容性，通过解码器，分别连接四个芯片的片选端。

分离器设置如下图：选择两位输出端为片选信号，其余 5 位构成 32 位的片内地址信号

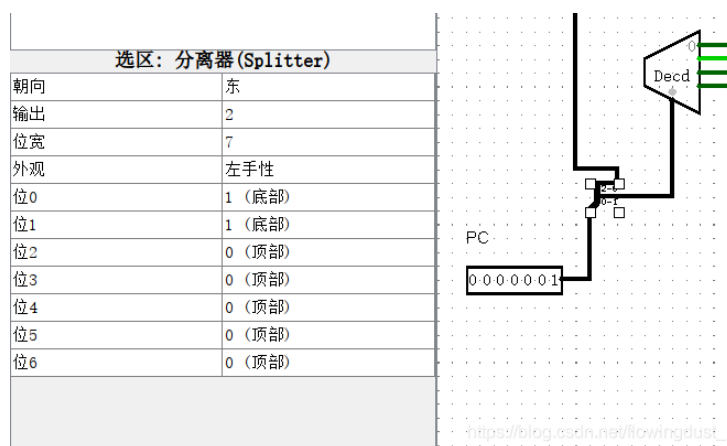


图 1 分离器设置电路图

分别使解码器的输出端连接四个 RAM 的片选端，使能端 (ld) 同时连接一个引脚，仿真时置为 1；复位端 (clr) 连接一个引脚，仿真时置为 0。

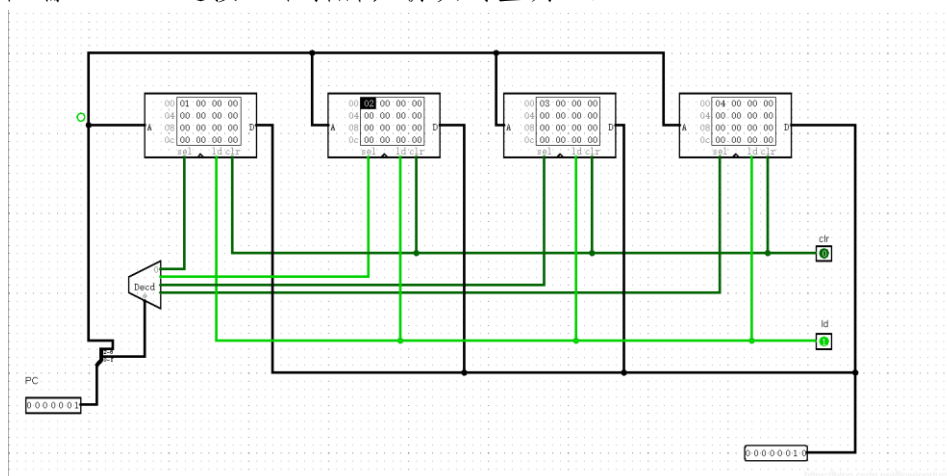


图 2 分离器实现整体电路图

4.2 位扩展的基本操作：

位扩展设计，我们需要对存储器单元的位数进行扩展，因此选要芯片输入端直接连接，只需要在输出端分别连接即可。

引脚分别并联后接至地址译码器的输出，而地址译码器的输入则由系统地址总线的高位来承担。当存储器工作时，系统根据高位地址的译码同时选中两个芯片，而地址码的低位也同时到达每一个芯片，从而选中它们的同一个单元。在读/写信号的作用下，两个芯片的数据同时读出，送上系统数据总线，产生一个字节的输出，或者同时将来自数据总线上的字节数据写入存储器。

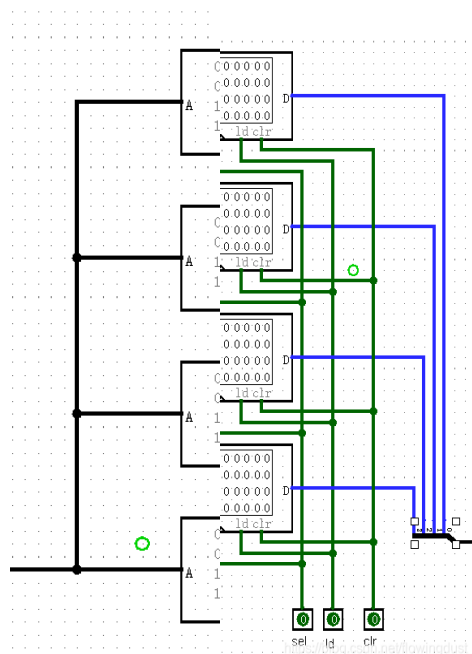


图 3 位扩展的基本实现电路

5. 实现效果：

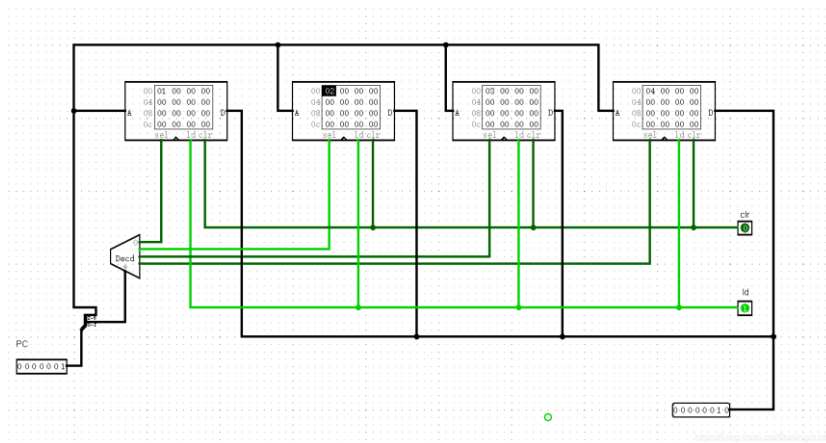


图 4 字扩展的基本实现电路

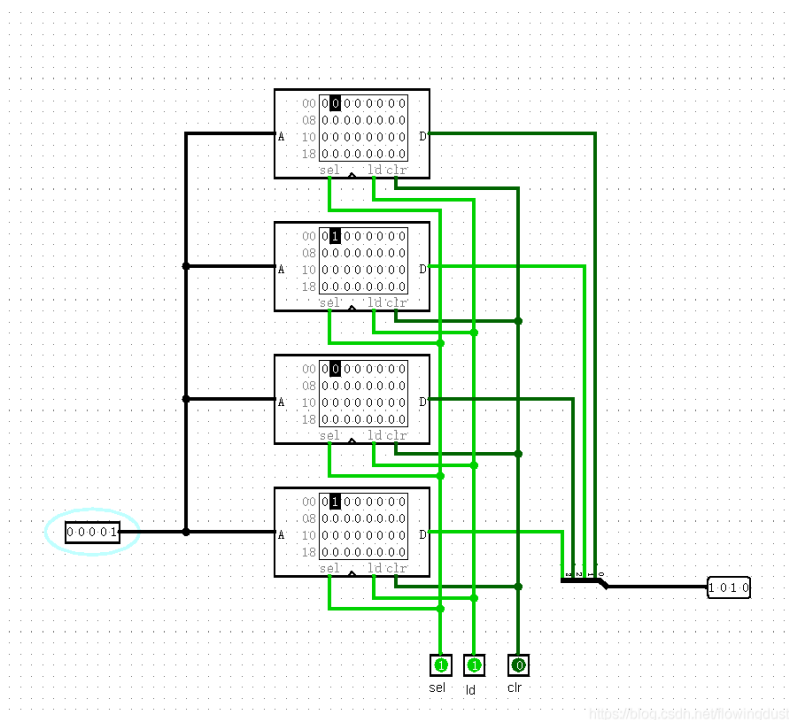


图 5 位扩展的基本实现电路

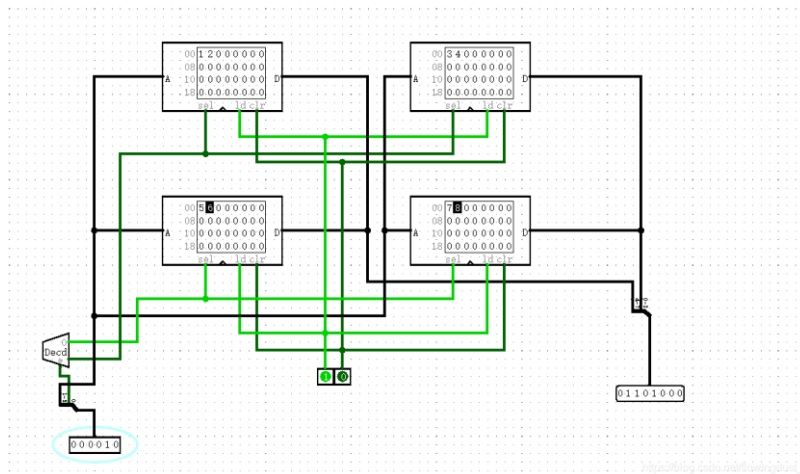


图 6 字位扩展的基本实现电路

思考 和 体会	<p>本次实验是扩容器的扩展操作，通过相关的实验操作加深了对相关原理的理解。</p>
备注	

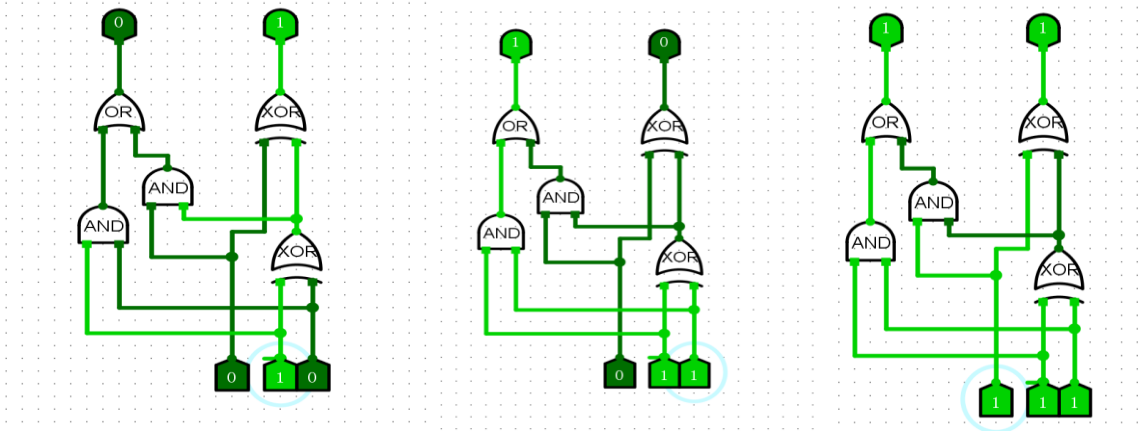
实验项目	8 位加减法器	日期	2022/11/11
实验环境	logisim 2.16.1.0	成绩评定	
实验目的和要求	1. 基本的实验要求：实现 8 位加减法器 2. 可参考 8 位串行加法器的实现的实验，对于 8 位可控加法器使用多路选择器 3. 注意溢出的判断 4. 进阶的实验要求：实现 16 位加减法器		
实验内容和结果	<p>1. 实现 8 位加减法器： 基本实现思路：</p> <p>设计思路：输入 X_i 和 Y_i 各 8 位数，选通端 Sub 为 0 时做加法，为 1 时做减法，检测运算结果是否溢出后再进行输出 首先将 8 位运算器进行封装，实现封装的子电路</p> <p>1.1 实现一位全加器 FA 封装</p>  <p style="text-align: center;">图 1 实现一位全加器 FA 封装</p> <p>注：一位全加器 FA 实现的基本公式：</p> $Cout = X_i Y_i + Cin(X_i \oplus Y_i)$ $S_i = X_i \oplus Y_i \oplus Cin$ <p>其中 X_i 和 Y_i 表示的是输入，Cin 表示的是低位的进位</p>		



图 2 运算器的封装

1.2 溢出判断的原理:

符号位进位 \oplus 最高位进位

1.3 一位加法逻辑电路:

$$S = X \oplus Y$$

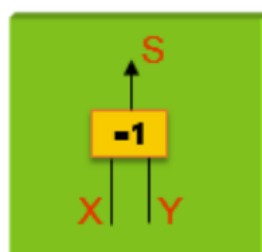


图 3 一位加法逻辑电路的实现

1.4 N 位加法器:

N 位加法器包括 N 个全加器，将 N 个一位全加器串联，低位进位输出连接到低位进位输入

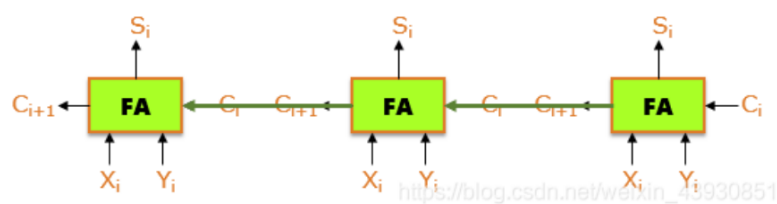


图 3 N 位全加器的电路实现

1.5 可控加减法电路:

各位逐位相加，从右向左传递

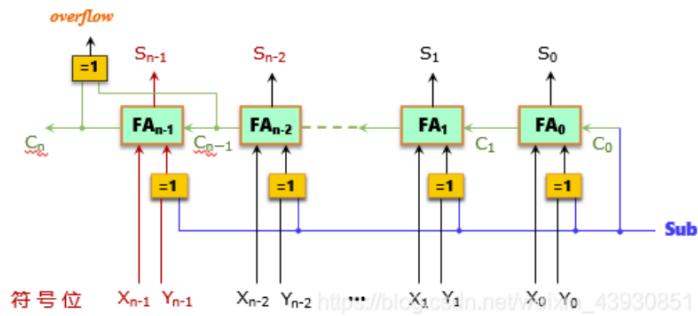


图4 可控加减法电路实现

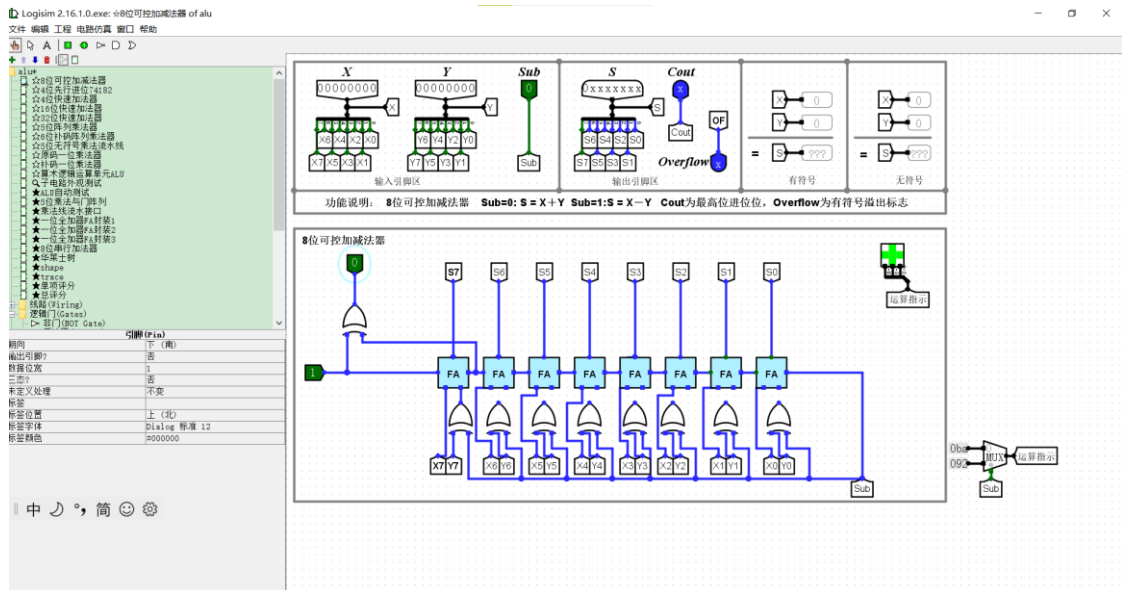
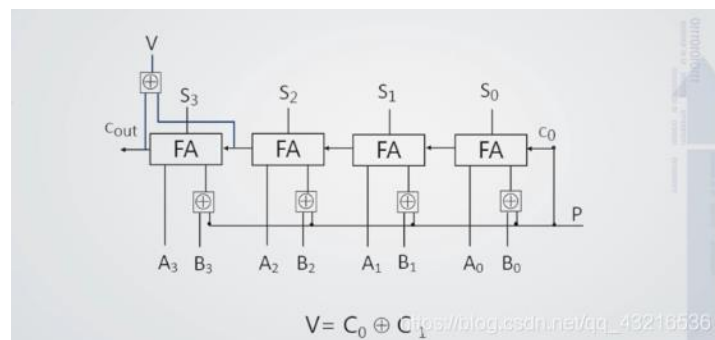
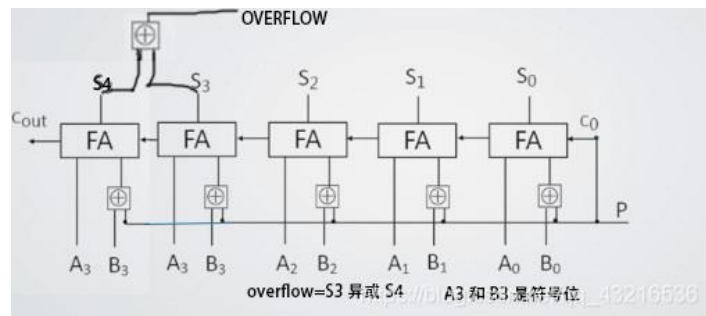


图5 实验电路的实现

2.1 溢出的判断:

2.1.1 有符号数溢出的判断:



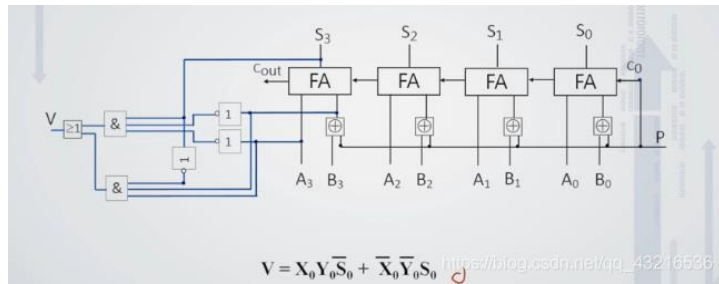


图 6 有符号数溢出的判断

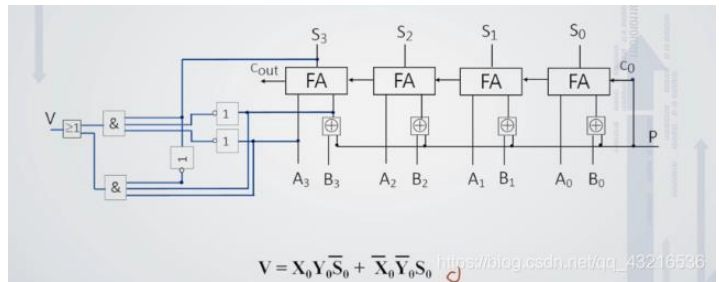


图 7 无符号数溢出的判断

3.1 16 位快速加法器的实现:

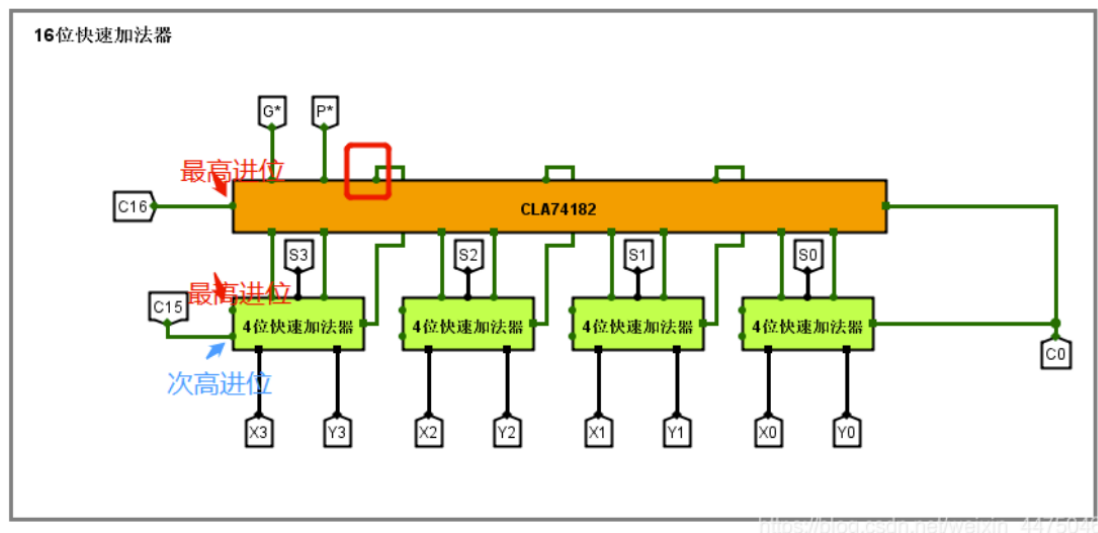
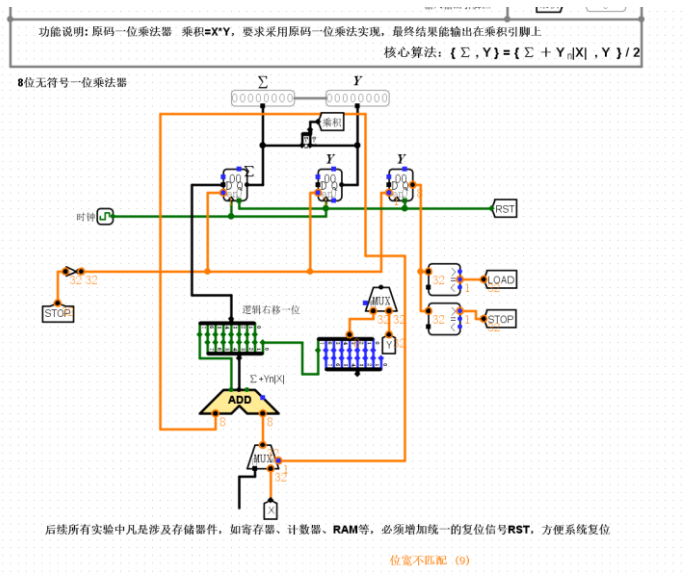


图 8 16 位快速加法器实现原理图

思考 和 体会	<p>本次实验主要实现的是运算器相关的操作，实验的操作环境是 <i>Logisim-ITA</i>。首先实现的功能是 8 位可控加减法器，这个电路实现的是较为成功的。但是对于后面的使用多路选择器进行控制的操作，没有很好地实现。后面又实现了溢出的判断，基本的实现思路是通过最高位的进位和符号位的进位进行异或操作实现的。最后实现了 16 位的快速加法器，但是遗憾的是，没有在 8 位可控加减法器的基础上对 16 位可控加减法器进行实现。</p>
备注	

实验项目	原码一位乘	日期	2022/11/18
实验环境	logisim2.16.1.0	成绩评定	
实验目的和要求	1. 实现 8 位无符号一位乘法器原码一位乘的操作 2. 实现 8 位无符号一位乘法器补码一位乘的操作		
实验内容和结果	<p>1. 实验设计要求：</p> <p>在电路文件当中的原码一位乘子电路中增加控制电路和数据通路使得该电路能够自动完成 8 位无符号数的一位乘法运算，设置引脚初始值，然后驱动时钟自动仿真，电路即可自动完成运算，运算结果传输到输出引脚，电路的实现即可完成。</p> <p>2. 基本方案设计：</p> <p>无符号原码一位乘的运算：维护一个部分积，初始为 0，每次取乘数 b 的最低位。如果最低位为 1，部分积=部分积+乘数；如果最低位为 0，部分积=部分积+0。然后将部分积右移一位，同时将乘数 b 右移一位，第 i 次对部分积的右移操作即可得到对应的第 i 位低位。8 次结束后的部分积即为最终结果的高 8 位。</p> <p>结合我们上面的叙述，我们可以得到该方案的本质：乘数 a 和乘数 b 的部分积各位相乘后再相加，即可得到最终的结果。</p> <p>但是在实验的过程当中遇到了一些问题，以下是出现问题的电路图：</p>  <p>图 1 出现问题的电路图</p>		

主要存在的问题是电路的位宽不合适，因此在电路图当中对于相关的位宽进行了调整，得到了正确的电路图。

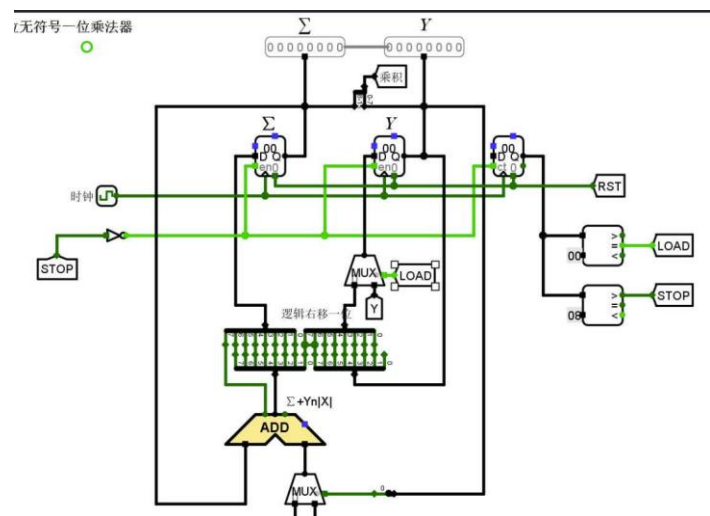


图 2 正确的电路图示

下面我们对于电路图当中的一些比较关键的地方进行分析，加深对于实验原理的理解。

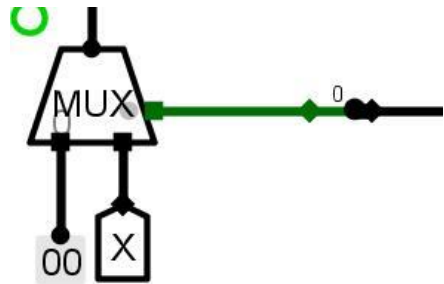


图 3 多路选择器部分电路

对部分积作加运算时对加数 0 或乘数 a 的选择：用多路选择器实现，选择依据为乘数 b 的当前最低位，最低位的选择用了分线器来实现。

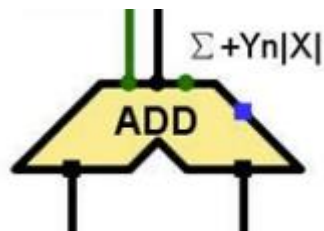


图 4 加法器部分电路

对所维护的部分积作加运算：用 8 位串行加法器实现。

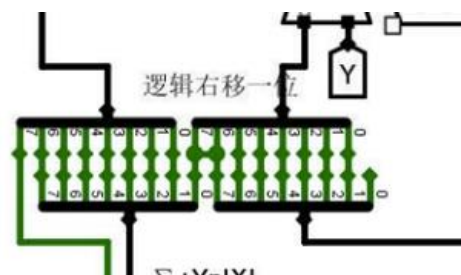


图 5 分线器部分电路

对部分积的右移和对乘数 b 的右移：用四组八位宽的分线器实现。其中新部分积的最高位由串行加法器的进位得到，乘数 b 的最高位由部分积右移出来的那一位得到，这样部分积的结果即为乘积的低八位。

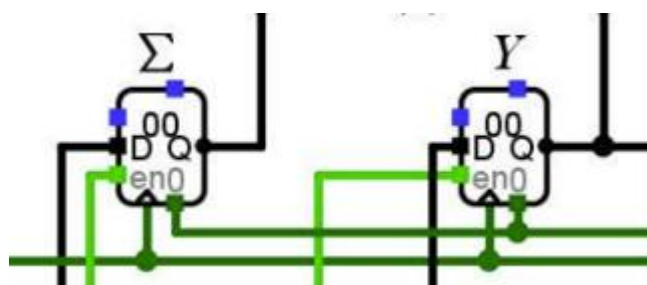


图 6 两个寄存器的电路图

我们使用了 2 个寄存器用来存储部分积和乘数 b ，如上图所示。

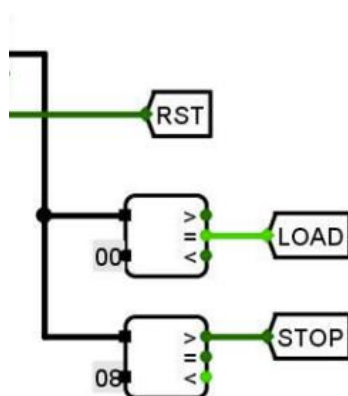


图 7 边界情况的处理电路图

对于边界情况的特殊处理，我们引入了一个计数器用来统计脉冲的次数，进而对边界的情况进行处理。

3. 补码一位乘电路设计：

我们首先对补码一位乘的流程图和硬件逻辑的实现进行大致的把握，有助于增加我们对电路设计的理解。

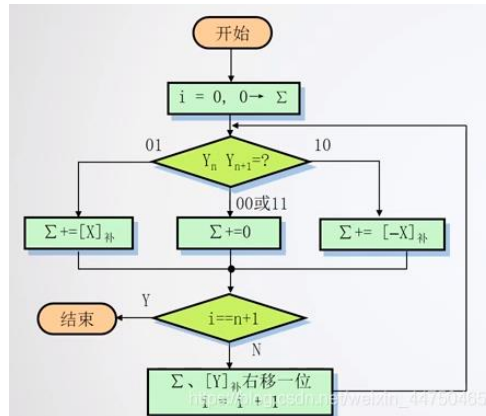


图 8 补码一位乘的流程图设计

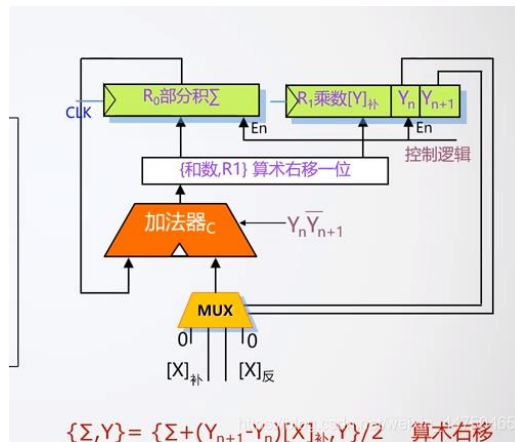


图 9 补码一位乘的硬件逻辑实现

对于补码一位乘的电路设计我们也主要分为以下的 4 个步骤，分别是数据加载、移位控制、停机逻辑、 Y_{n+1} 和 Y_n 的获取

数据加载：

我们需要对 $Y_{n+1} Y_n$ 数值进行判断，，从而判断在所求和的基础上加 0、 $[x_{补}]$ 还是 $[-x_{补}]$

当 $Y_{n+1} Y_n$ 的数值为 00 的时候，加 0

当 $Y_{n+1} Y_n$ 的数值为 01 的时候，加 $[-x_{补}]$

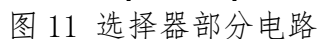
当 $Y_{n+1} Y_n$ 的数值为 10 的时候，加 $[x_{补}]$

当 $Y_{n+1} Y_n$ 的数值为 11 的时候，加 0



对于这一部分的功能，我们可用选择器进行实现。

入对应的是 $[x_{\text{反}}]$,在 ADD 这个加法器中当 $Y_{n+1}Y_n=01$ 时,进位输入为1,此时实现加 $[-x_{\text{补}}]$,在选择器的第2位输入的是 x 。



首次输入 Y 寄存器的数据：八位的原始输入数据 y

我们采用选择器进行控制，若此时是初始数据（即还未进行计算，可用计数器和比较器的电路设计实现判断）即该选择端为 1，则将 y 的 8 位数据输入到 Y 寄存器中，若不是初始数据（即已经开始计算了），则选择端为 0，则将右移动后的积和 y 输入 Y 寄存器中

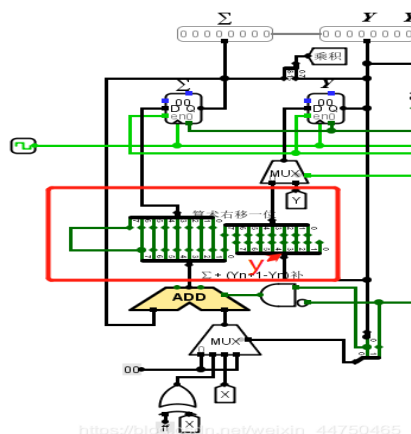


图 12 分线器部分电路

我们将得到的和算数右移，去掉最低位，最高位复制原来的最高位，采用分线器实现。右移后的 16 位数据，左边 8 位为部分积，存入部分积寄存器中；右边 8 位为右移过来的低位积+y 的几位高位数据（低位数据随着时钟驱动后的每次计算，被移出去），存入 Y 寄存器中。

停机控制：

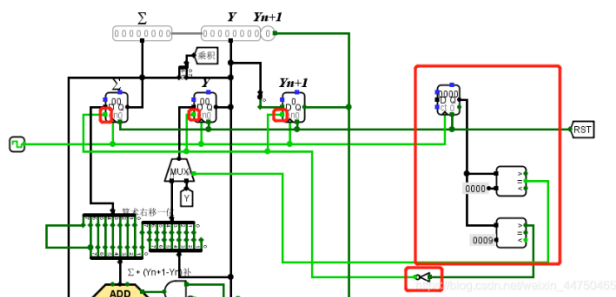


图 13 停机控制电路实现

Y_{n+1} 和 Y_n 的获取：

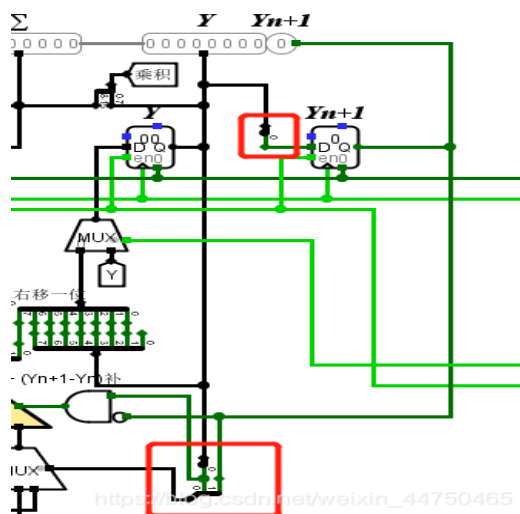
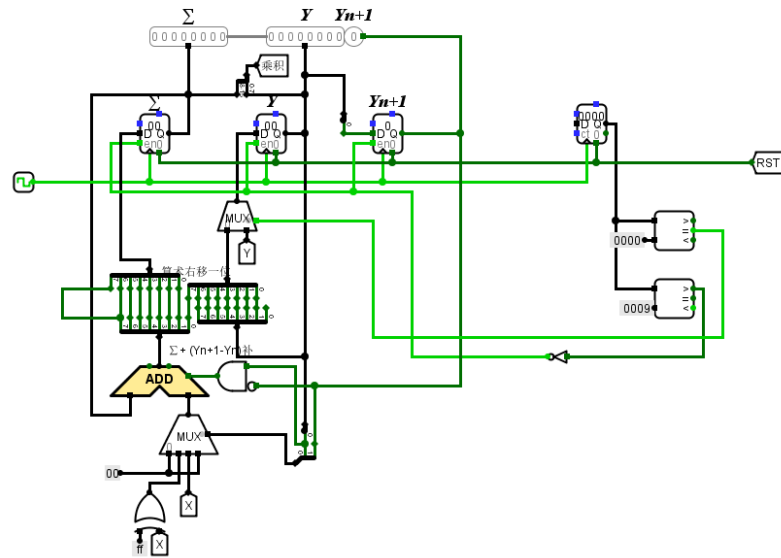


图 14 Y_{n+1} 和 Y_n 的获取 电路图

由于在开始计算时，应该将 8 位数据 y 的后面添加一位 0 作为 Y_{n+1} ，此后将 8 位数据 y 的最低位取为 Y_{n+1} ，次低位取位 Y_n ，因此取 y 的第 0 位（最低位）作为 Y_{n+1} 寄存器的输入，驱动时钟时，该数据将会存入到 Y_{n+1} 寄存器中，而 Y_n 是右移后的第 0 位即是原来的次低位，从而实现获取 Y_{n+1} 与 Y_n 。

1位补码 Booth一位乘法器

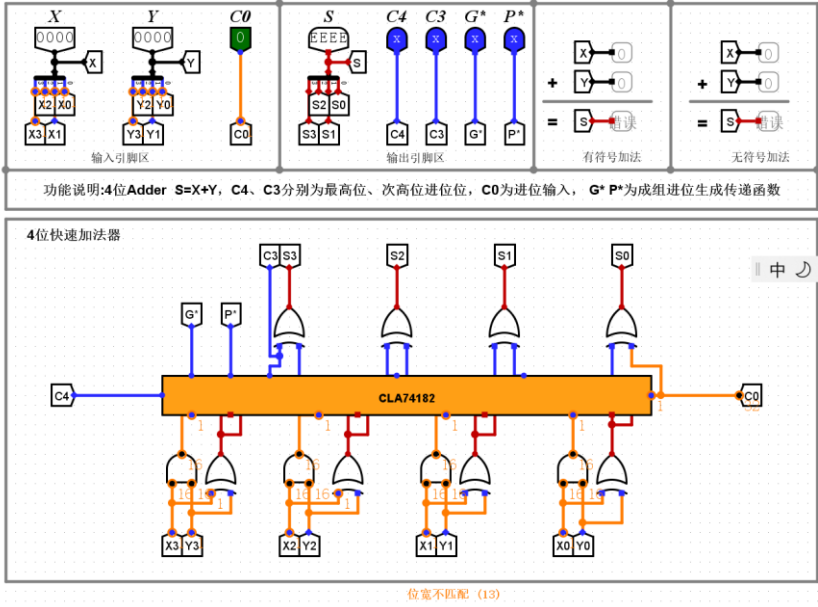


$$\Sigma' = (Y_{n+1} - Y_n)[X] \text{ 补}/2$$

https://blog.csdn.net/weixin_44750465

图 15 整体实现的电路图

思考 和 体会	<p>本次实验是设计 8 位无符号原码一位乘和补码一位乘的乘法器，其中补码一位乘电路的实现是在完成原码一位乘的基础上再进行完善的。在实验的过程当中，也遇到了一些问题，比如在链接电路的时候，出现了位宽不一致的情况，因此需要对相关电路元件的位宽进行了调整。另外，存在的一个问题是对于实验的原理仍然不是很清晰，因此对于自己而言，自己课下还是需要花更多的时间加深对于实验原理的理解，这样才能够取得更好的实验效果。</p>
备注	

实验项目	4 位快速加法器	日期	2022/11/25
实验环境	logisim 2.16.1.0	成绩评定	
实验目的和要求	1. 实现 4 位快速加法器 2. 实现 16 位快速加法器 3. 实现 32 位快速加法器		
实验内容和结果	<p>对于 4 位快速进位器，我们需要实现 2 个功能，一个是 4 位快速进位器电路的实现，另一个是对于该电路当中的 CLA74182 的设计实现，也就是我们所说的 4 位先行进位。</p> <p>1. 4 位快速加法器的设计实现：</p> <p>对于这一部分的设计实现，我们首先需要理解其原理，对于 C1 和 C2 的值，我们可以发现最后都可以递归到 G1,P1 和 C0 的实现当中，因此，我们可以类似于“套娃”的思想，一步步按照其原理，实现其电路图。</p>  <p>功能说明:4位Adder $S=X+Y$, C4、C3分别为最高位、次高位进位, C0为进位输入, $G^* P^*$为成组进位生成传递函数</p> <p>4位快速加法器</p> <p>位宽不匹配 (13)</p> <p>图 1 四位进位加法器的设计电路</p> <p>如上图是对于四位进位加法器的设计电路，但是因为位宽不匹配的问题，导致电路出现了一定的问题，因此我们应该首先明确各位的位宽，然后在此基础上进行电路的设计。</p>		

电路引脚

信号	输入/输出	位宽	说明
X	输入	4 位	加数
Y	输入	4 位	加数
C0	输入	1 位	进位输入
S	输出	4 位	运算和
C4	输出	1 位	最高位进位位
C3	输出	1 位	第 3 位进位位
G*	输出	1 位	成组生成函数
P*	输出	1 位	成组传递函数

图 2 电路引脚当中各信号位宽的设计说明

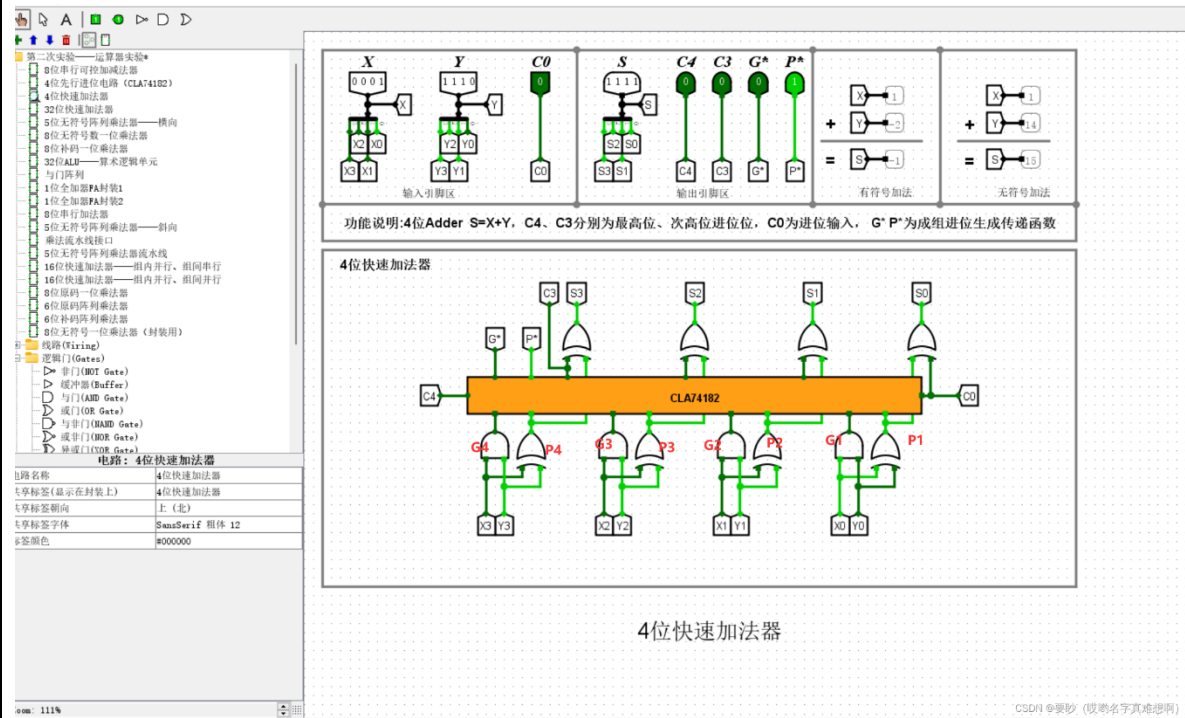


图 3 四位进位加法器正确的电路设计图

完成了四位进位加法器以后，我们还需要对中间的 CLA74182 部件进行相关的设计，即我们所说的 4 位先行加法器。因此我们又实现了该电路的设计。

在 CLA74182 的电路设计当中，也是出现了位宽不一致的情形，而且由于时间的原因，一开始也没有能够将该电路设计的很好。

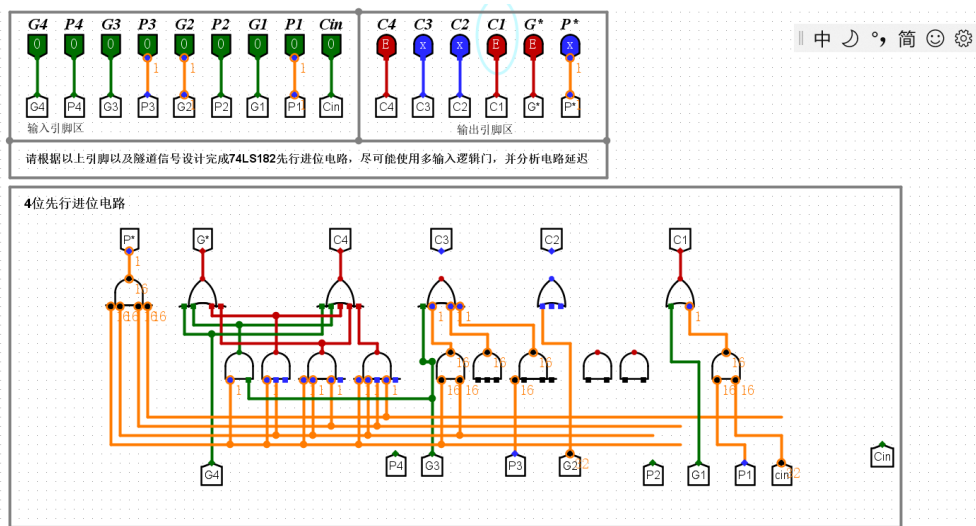
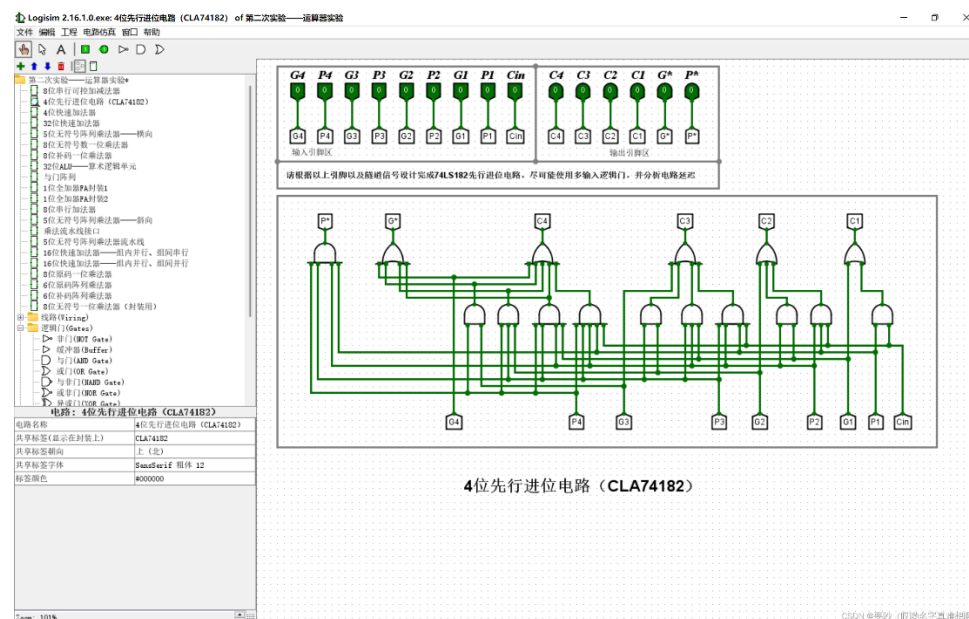


图 4 CLA74182 的电路设计

上面是对于 CLA74182 的电路的设计，没有能够完成的很完善，在整体的电路布局上面还是存在一定的问题，因此做出了如下的修正。



参考网络上的电路设计，我们得到了 CLA74182 的电路设计图，感觉总体的逻辑还是较为复杂，连线也是比较多的。

这样我们就实现了 4 位快速进位器的实现。后面我们进行 16 位和 32 位快速进位器的电路的实现。

参考博客：

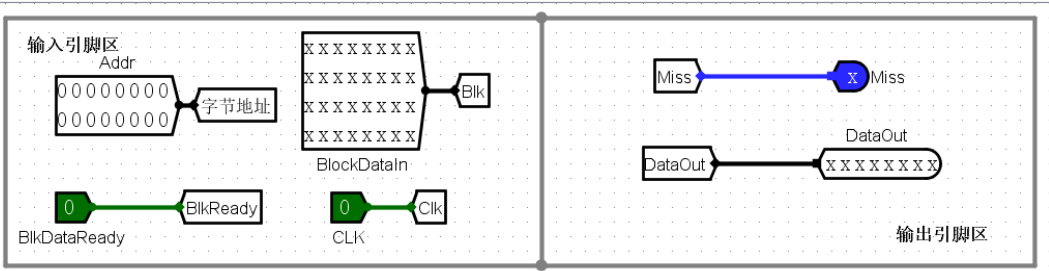

https://blog.csdn.net/qq_52641289/article/details/124064319

https://blog.csdn.net/qq_46545864/article/details/115386163

思考和
体会

本次实验实现了快速进位链的设计实现，主要实现了 4 位快速进位器的设计实现，但是后面的 16 位和 32 位的设计并没有完成，这也是本次实验有所遗憾的地方，也是后面需要进一步完善的地方。

备注

实验项目	cache	日期	2022/12/02																												
实验环境	log isim 2.16.1.0	成绩评定																													
实验目的和要求	1. 实现直接相联映射 cache 电路设计 2. 实现全相联映射电路设计 3. 实现 2 路组相联映射电路设计 4. 实现 4 路组相联映射电路设计																														
实验内容和结果	<p>1. 直接相联映射电路的实现：</p> <p>为了实现直接相联映射的电路，我们需要重点掌握数据查找、地址映射、替换算法的使用，同时我们需要掌握译码器、多路选择器、寄存器的使用，能够实现直接相联映射 cache 电路的基本实现。</p>  <p>图 1 cache 模块电路框架</p>  <table border="1"> <thead> <tr> <th>引脚</th><th>类型</th><th>位数</th><th>功能说明</th></tr> </thead> <tbody> <tr> <td>Addr</td><td>输入</td><td>16</td><td>主存地址</td></tr> <tr> <td>BlkDIn</td><td>输入</td><td>32</td><td>块数据输入</td></tr> <tr> <td>BlkReady</td><td>输入</td><td>1</td><td>块数据准备就绪</td></tr> <tr> <td>Clk</td><td>输入</td><td>1</td><td>时钟输入</td></tr> <tr> <td>Miss</td><td>输出</td><td>1</td><td>1 表示数据缺失；0 表示数据命中</td></tr> <tr> <td>Data</td><td>输出</td><td>8</td><td>数据输出</td></tr> </tbody> </table> <p>图 2 cache 电路封装和电路简述</p> <p>映射机制：Blkready 信号用于判断数据块是否准备好，当 Blkready 信号有效并且时钟到来时，Cache 将块数据从 BlkDin 端口一次性载入对应的 Cache 缓冲区中，此时 Cache 数据命中，直接输出请求数据，解锁计数器使能端，继续访问下一个地址。</p>			引脚	类型	位数	功能说明	Addr	输入	16	主存地址	BlkDIn	输入	32	块数据输入	BlkReady	输入	1	块数据准备就绪	Clk	输入	1	时钟输入	Miss	输出	1	1 表示数据缺失；0 表示数据命中	Data	输出	8	数据输出
引脚	类型	位数	功能说明																												
Addr	输入	16	主存地址																												
BlkDIn	输入	32	块数据输入																												
BlkReady	输入	1	块数据准备就绪																												
Clk	输入	1	时钟输入																												
Miss	输出	1	1 表示数据缺失；0 表示数据命中																												
Data	输出	8	数据输出																												

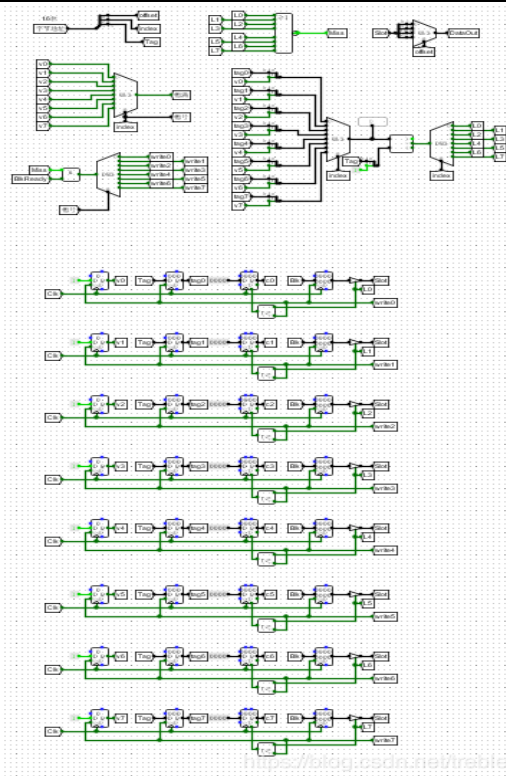


图 3 直接相联映射电路的实现

此电路的设计相对较为麻烦，但是行是可以复用的，因此我们只需实现其中一部分的设计实现。

根据谭志虎老师的实验用书的叙述，我们先将行实现，然后再进行复用。

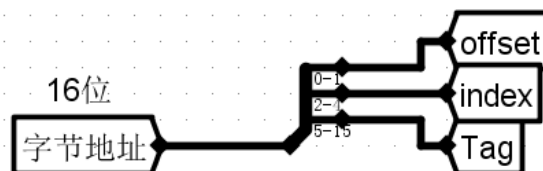


图 4 字节分配电路图

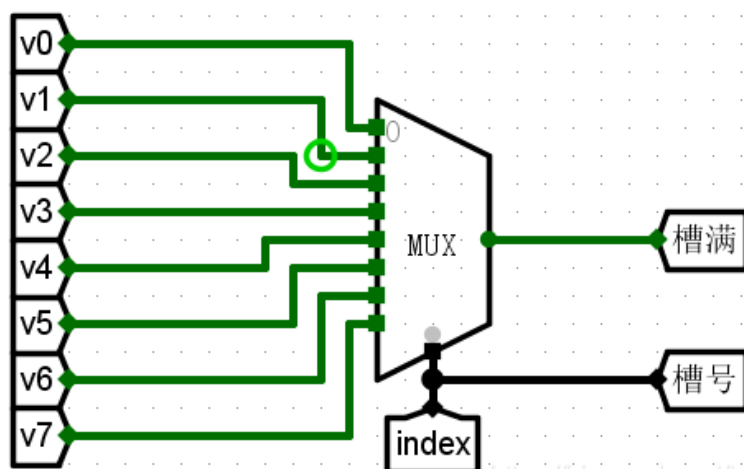


图 5 多路选择器部分电路

对于多路选择器部分的电路图，我们需要做出如下几点说明，其中 index 是用来表示槽号的，而判断位是用来判断是否将 cache 行写入多路选择器。

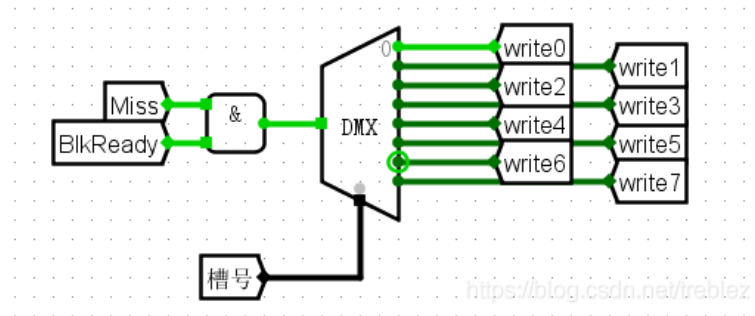


图 6 miss 部分电路图

判断是否需要写入解复用器，则需要再数据块已经准备完毕，同时 miss 没有写入的情况下是需要写入解复用器的。

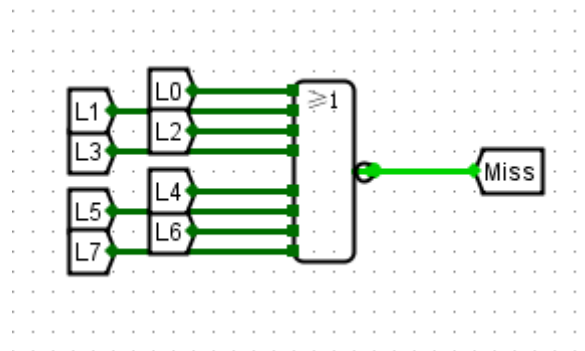


图 7 与非门电路

与非门电路是用来判断是否写入 miss 当中的。

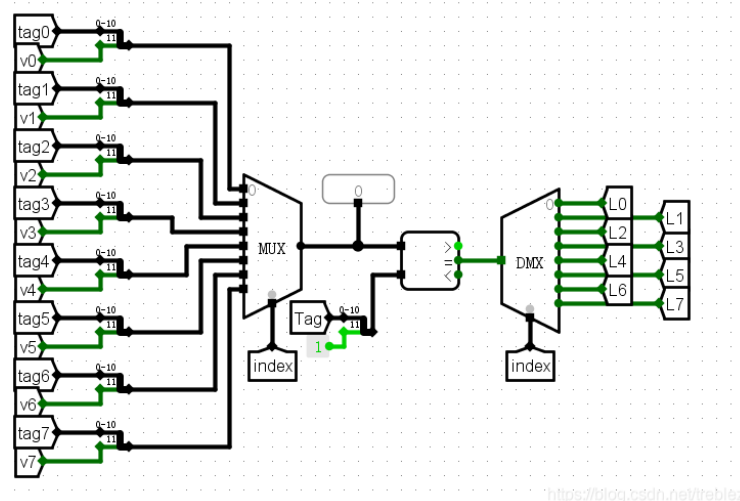


图 8 找到相应行的电路图

我们使用解复用器和比较器可以找到相应的 cache 行

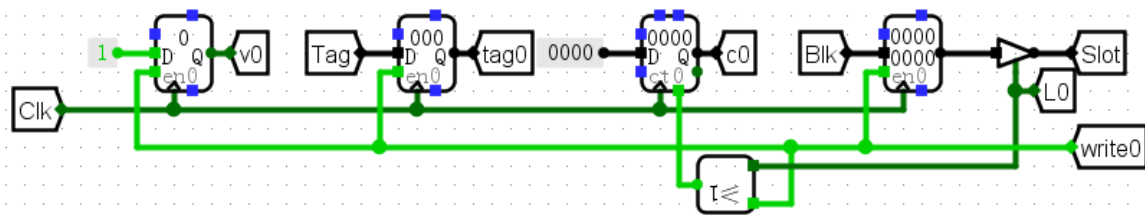


图 9 数据读入写出的判断电路

在 cache 的行当中有三个寄存器和一个计时器，当使能端为 1 的情况下进行数据的写入。当数据已经读入，且 L0 为 1 的情况下，进行数据的读出。

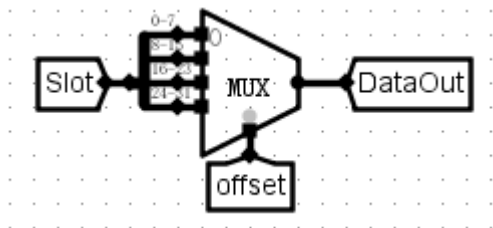


图 10 cache 行偏移电路图

Cache 行进行偏移，以此得到最终的电路图。

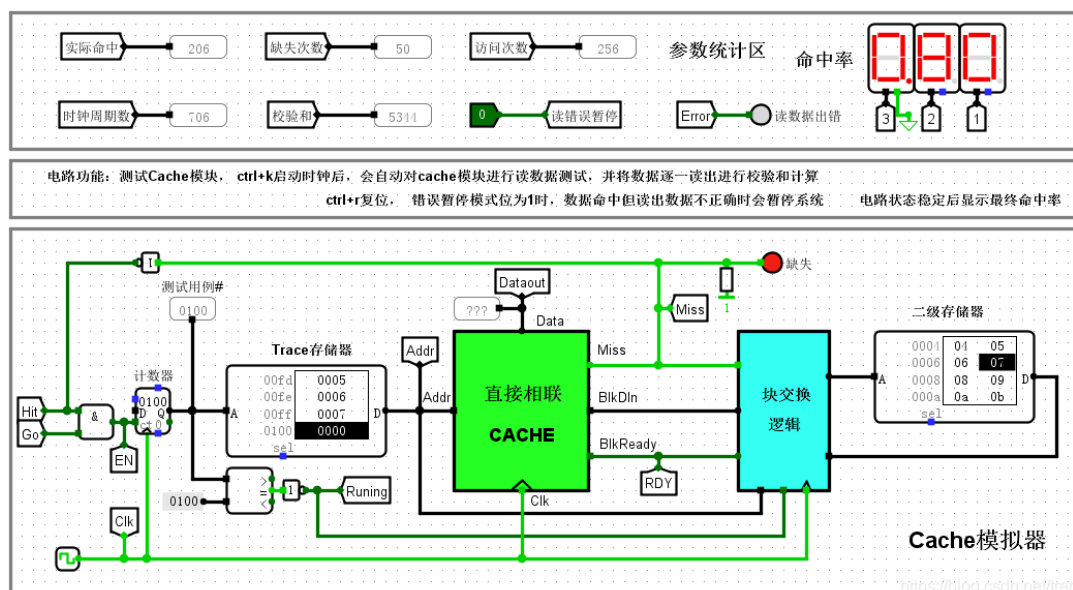


图 11 最终实现的电路图

2. 全相联映射电路设计：

该电路的设计难度是介于直接相联映射和组相联映射之间的，刚刚我们实现了较为简单的直接相联映射的电路的设计，现在我们开始全相联映射的电路的设计。

全相联映射的电路考察的技术主要包括以下几个方面：字节地址和 cache 槽的设计、写入和写出设计、比较设计、淘汰算法设计，其实和直接相联映射的设计方法是基本相同的，但是在具体的细节实现方面，难度会更大一些。



图 12 字节地址设计的电路图

在全相联映射当中是主存地址随机存储在任意的 cache 行，只要是空行就可以进行任意的存储，因此不需要行地址。只需要 Tag 标记位和 Offset 字内偏移地址。由于每个 cache 副本中只包含 4 个字节，因此 Offset 只需要 2 位，所以 Tag 标记位为字节地址 16 位-Offset 2 位=14 位。

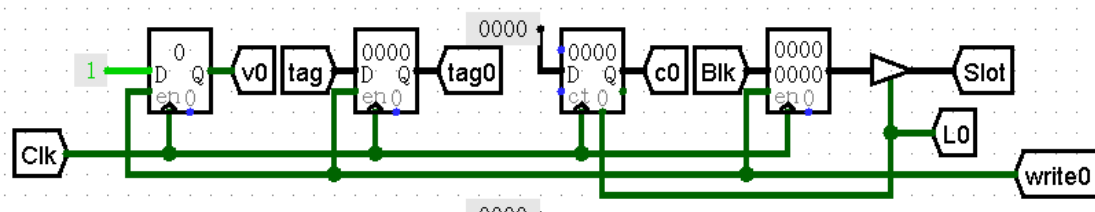


图 13 cache 槽设计的电路图

cache 主要包含四部分：**valid 标志**、**主存标记位**、**淘汰技术标记**、**数据副本**。而我们需要考虑的主要是前三部分，并且这三部分和之后的设计息息相关。当中 valid 位有效位，判断该 cache 槽是否被命中过，存入了数据；Tag 位是存入的主存标记位；淘汰计数是一个计数器，初值为 0，若行命中标志 Li 有效时，读入数据 0，达到清零的效果，若 Li 无效，则随着时钟频率一直进行计数。而数据副本通过三态门缓冲器连接到总线上，这里三态门缓冲器的作用是当 Li 行有效时，将数据副本中输出的数据进行缓存，也就是使 8 个 cache 槽中的数据都可以缓存到 1 个 Slot，实现了总线的作用。

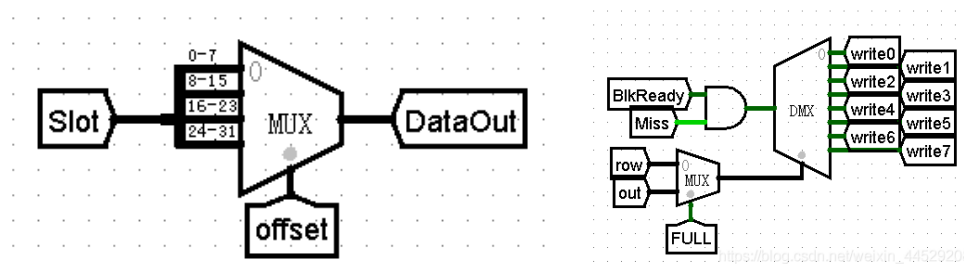


图 14 写入写出电路图

写出设计即通过字内偏移地址 Offset 进行选择总线 Slot 上的某一字节进行输出。

写入设计则要当 BlkReady 数据准备完成时（测试电路中的数据准备）选择具体的 cache 行进行写入，而写入的前提是该行为空，即 Miss 信号有效，才能写入数据。

写入行的判断：若存在空行，即 FULL 信号无效，则选择相应的空行进行写入；若 FULL 信号有效，则选择淘汰的行进行写入。两种情况都为之后的淘汰算法中选出的行号。

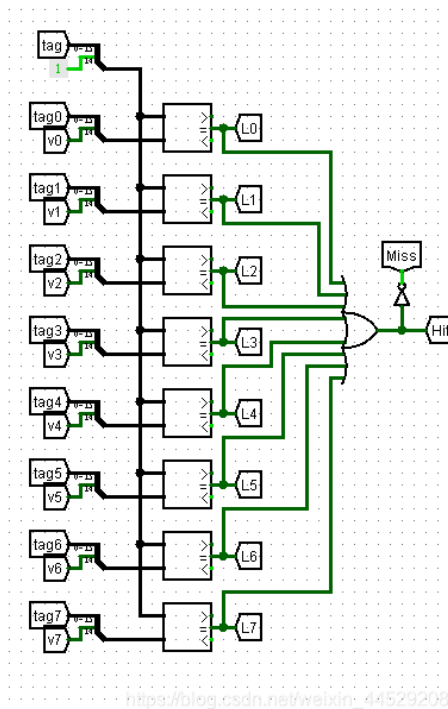


图 15 比较设计电路图

全相联因为是随机选取的 cache 行进行写入，因此没有行地址（索引），因此直接进行 8 个 cache 槽的并发比较，得到命中的 cache 行 L_i ，并给出 Miss 和 Hit 信号。

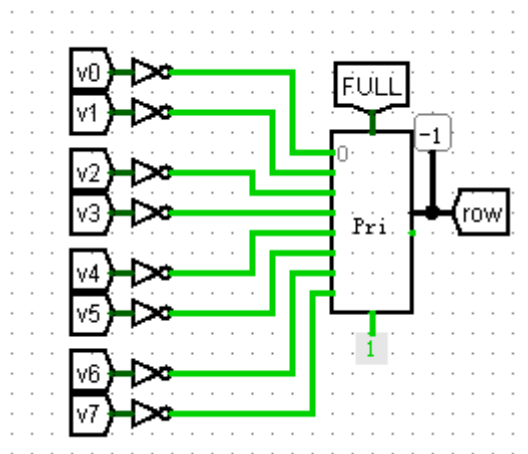


图 16 淘汰设计电路图

在空行的选择中，通过优先比较器来实现，优先比较器通过比较，输出的是索引较大的非 0 行，因此先对所有行的标志位取反，取反后，若为 1，则表示该行为空；若为 0，则表示该行已经存在数据。若所有行都存在数据则 FULL 行满信号有效。

由于我们在 cache 槽设计中的计数器是当行选中时清零，行为选中时随时钟频率进行计数，因此这里计数的最大值即为最少使用的 cache 槽。而我们用文件中附带的归并算法，这里的归并算法是输出两个值中较大的一位的数据和索引，最后得到相应的淘汰行。

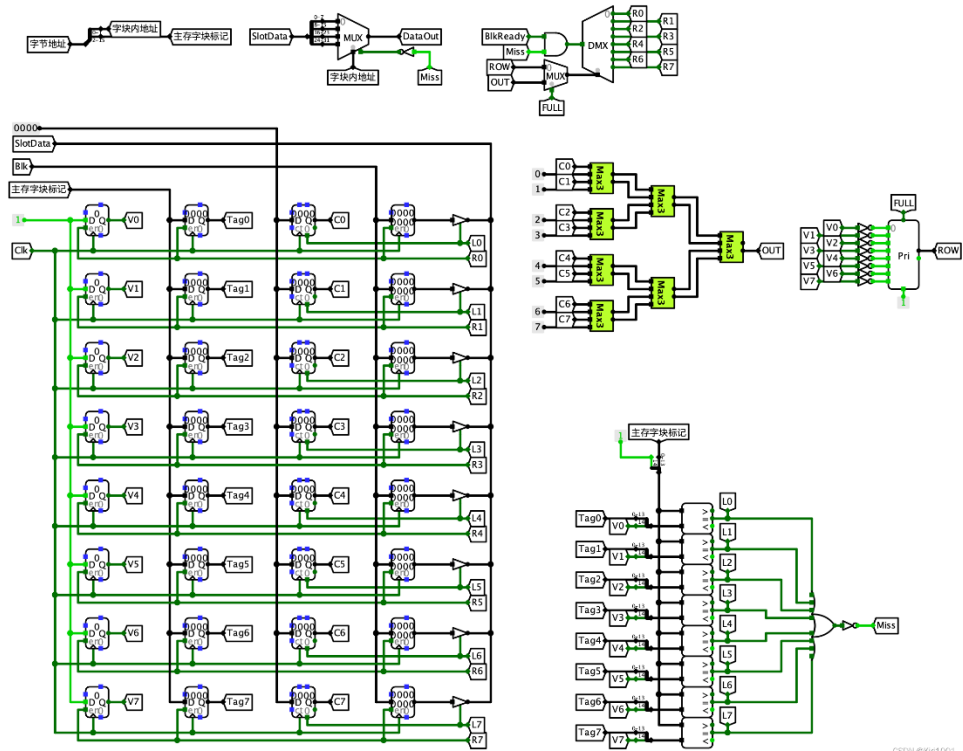


图 17 全相联映射实现电路图

参考博客：

[\(97 条消息\) 【计算机硬件系统设计（华科）——存储器设计（Logisim 实现）】 Kiril001 的博客-CSDN 博客 logisim 存储器实验](#)

[\(97 条消息\) 华科计算机组成原理 存储系统实验 汉字字库 MIPS Cache 存储 \(Logisim&Educoder\) JAMESHUANG 的博客-CSDN 博客 logisim 字库在哪](#)

思考
和
体会

本次实验是 cache 电路的设计实现,此次实验是计算机组成与结构本学期的最后的一次实验,同时本次实验较之前的实验相比,难度有了较大的提升。我主要参考了 CSDN 上的一些电路原理的叙述,逐渐明晰了电路原理图的设计思路,实现了直接相联映射和全相联映射的电路图,但是最后的组相联映射当中的 2 路组相联映射电路图和 4 路组相联映射电路图并没有实现,这也是本次实验有些遗憾的地方所在。

备注

