華中科技大學

计算机组成原理-实验报告

院　　系　软件学院

专业班级　软件 2003 班

姓　　名　侯皓斐

学　　号　U202010851

指导教师　黄浩

2022 年 12 月 20 日

目录

1 课程实验概述

1.1 实验目的

- 熟悉 Logisim 软件平台。
- 掌握运算器基本工作原理
- 掌握运算溢出检测的原理和实现方法；
- 理解有符号数和无符号数运算的区别；
- 理解基于补码的加/减运算实现原理；
- 熟悉运算器的数据传输通路。
- 熟悉 ROM、RAM 存储器的使用；
- 掌握存储器字扩展，位扩展的基本原理；
- 为 MIPS CPU 设计功能部件---寄存器文件；

1.2 实验环境

Logisim 是一款数字电路模拟的教育软件，用户都可以通过它来学习如何创建逻辑电路，方便简单。它是一款基于 Java 的应用程序，可运行在任何支持 JAVA 环境的平台，方便学生来学习设计和模仿数字逻辑电路。Logisim 中的主要组成部分之一就在于设计并以图示来显示 CPU。当然 Logisim 中还有其他多种组合分析模型来对你进行帮助，如转换电路，表达式，布尔型和真值表等等。同时还可以重新利用小规模的电路来作为大型电路的一部分。

http://www.cburch.com/logisim/docs.html

2 运算器组成实验

2.1 八位串行可控加减法器电路设计

利用已经封装好的全加器（封装 1）设计 8 位串行可控加减法电路，其引脚电路如下图所示。

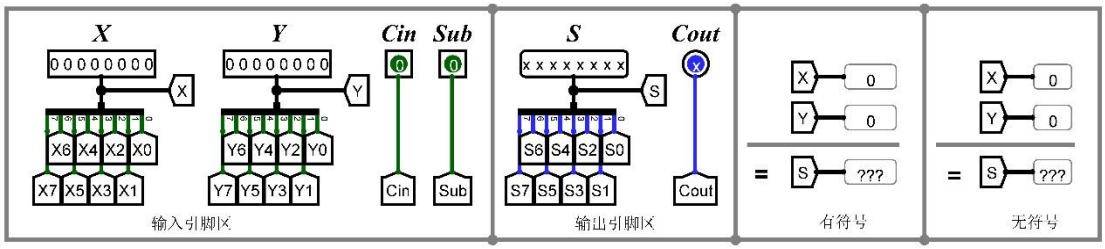
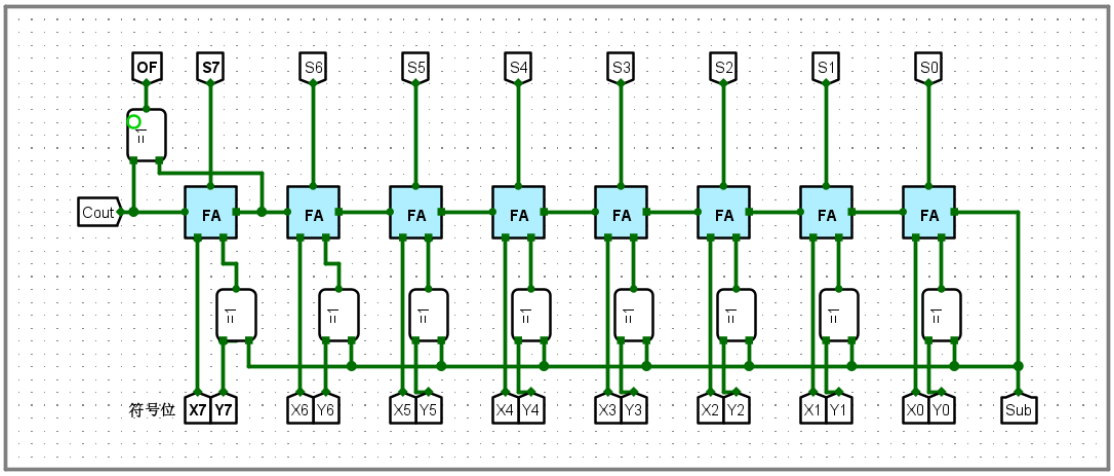


图 1 八位串行可控加减法电路引脚定义

设计结果如下：



1. 对于加减法的控制，你是采取何种方式实现的？

即将“减法”转换为“加法”，用加法电路实现减法运算，把-[y]补转换为+[-y]补来实现。 所以只要把减数 y 的负数也就是-y 的补码表示出来，就可以简单地利用“加法器”来实现减法运算。我们表示-y 的补码就需要将 y 按位取反再加 1. 当 Sub 为 1 的时候，输入一位全价器得数字就是 y 的取反，而一开始的进位确定了取反后加 1. 这样就实现了加减法的控制。

2. 你是怎样实现溢出检测的？其数学原理是什么？

运算结果超出数据类型所能表示的数据范围的现象称为溢出。我们此处根据运算过程中，最高数据位的进位与符号位的进位位是否一致进行检测。设运算时最高数据位产生的进位信号为 Cd，符号位产生的进位信号为 Cf，那么相应的检测逻辑表达式为：V= Cd xor Cf 这个表达式表明，运算过程中最高数据位的进位 Cd 与符号位的进位 Cf 不同步时，就表明有符号运算结果发生了溢出。从而使用 XOR gate 实现。

2.2 四位先行进位电路

根据下图定义的输入输出引脚完成 4 位先行进位电路。

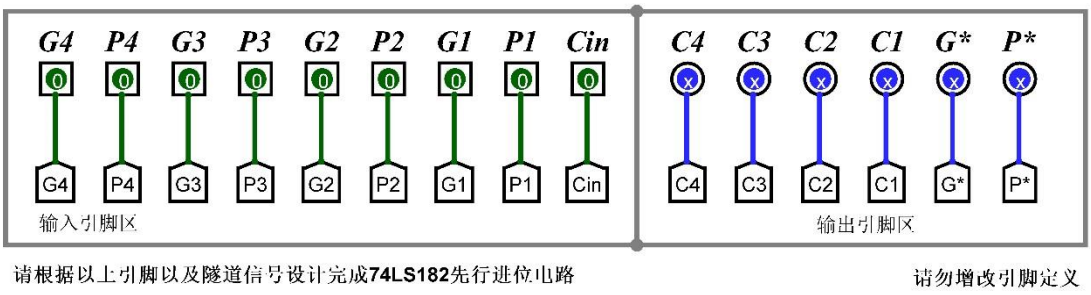
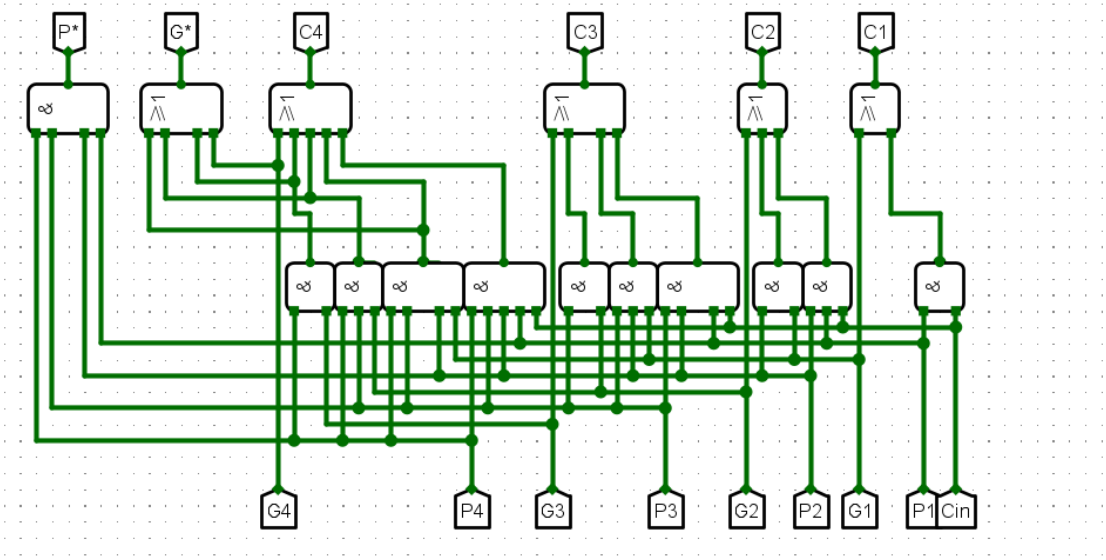


图 2　四位先行进位电路引脚定义

设计结果如下：



1.　CLA74182 的作用是什么？

之前实现的八位加法器以及八位加减法器都是以“串行”方式实现的，因为高位运算需要等待低位运算，因此串行进位加法器的速度较慢。CLA74182 通过事先推导进位公式，使用逻辑门直接得到进位信息。

2. 快速加法器与普通的串行加法器的区别是什么？他是通过什么方法来实现“快速”的？

快速加法器并不依赖上一位全加器计算的进位结果来计算本位的结果。高位运算需要等待低位运算，因此串行进位加法器的速度较慢。CLA74182 通过事先推导进位公式，使用逻辑门直接得到进位信息。从而减少了计算的时钟周期，实现了快速。

2.34 位快速加法器设计

利用已经前一步设计好的四位先行进位电路构造四位快速加法器，其引脚定义如下图。

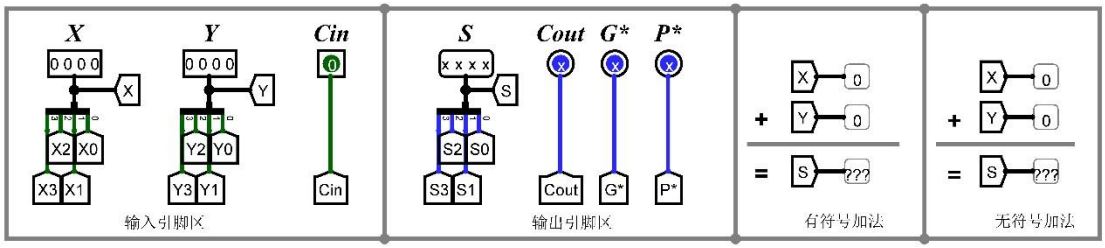
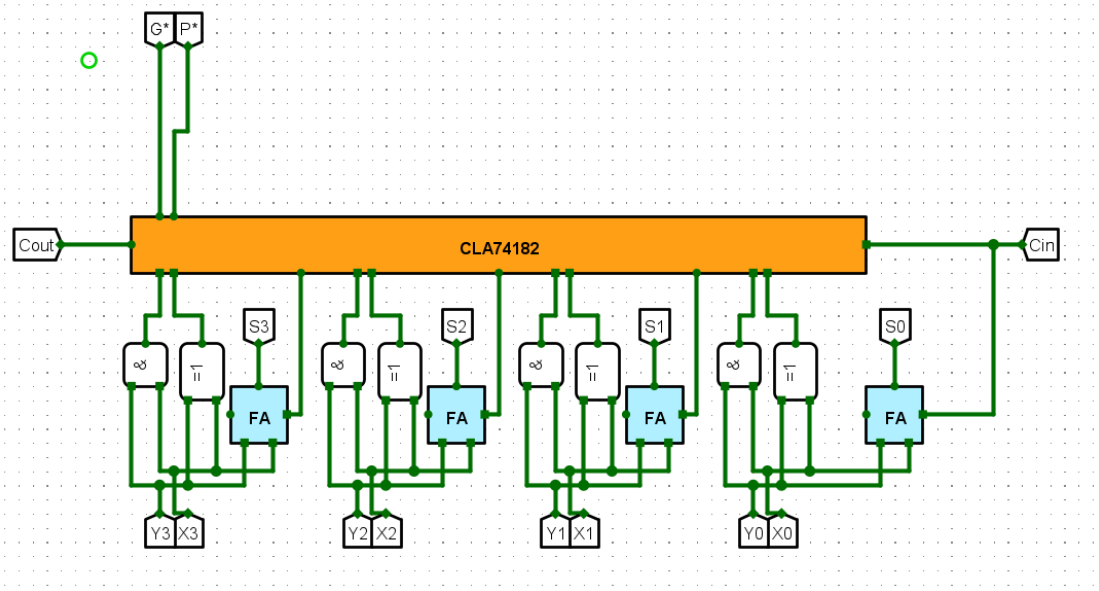


图 3 四位快速加法器引脚定义

设计如下：



2.416 位快速加法器设计

利用四位先行进位电路和四位快速加法器构造十六位组间先行进位，组内先行进位快速加法器，其引脚定义如下图。

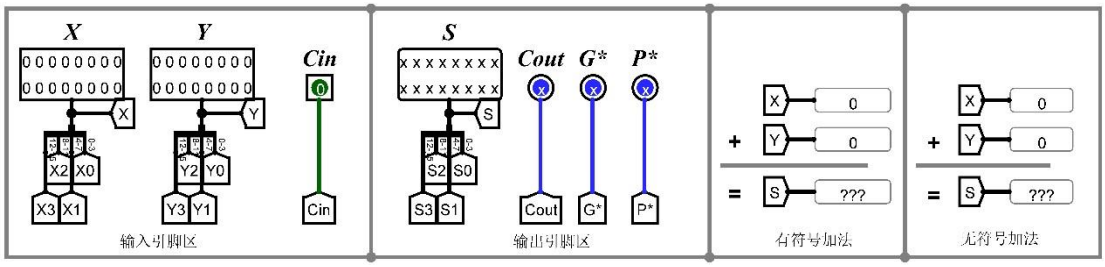
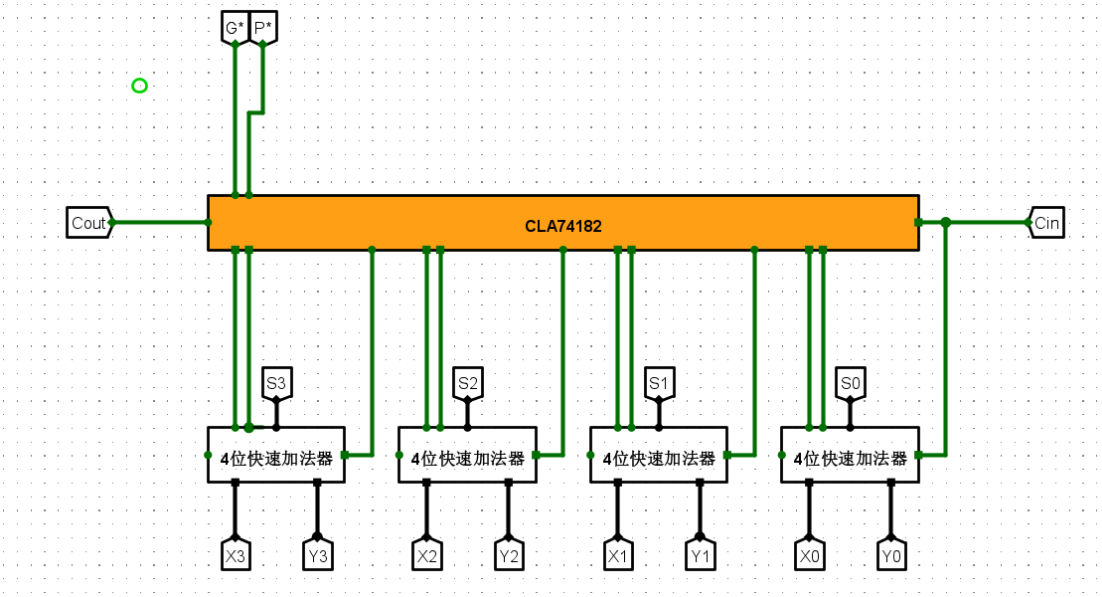


图 4 十六位快速加法器引脚定义

设计如下：



2.532 位快速加法器设计

利用前面构建的部件完成 32 位快速加法器，并分析其时间延迟，其引脚定义如下。

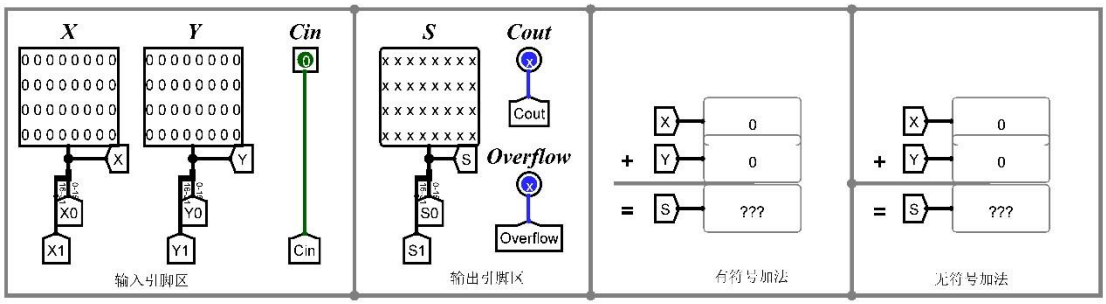
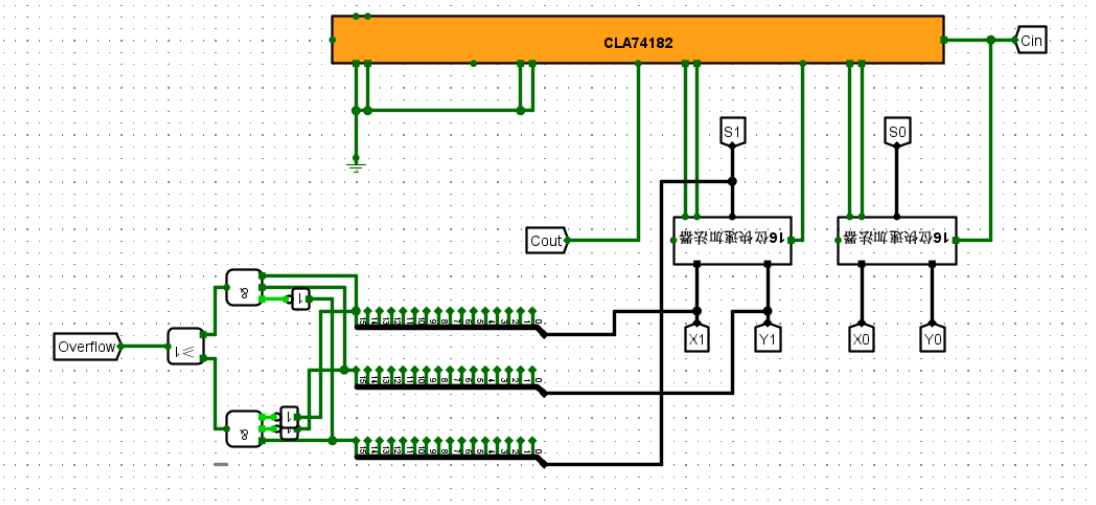


图 5 32 位快速加法器引脚定义

设计方案如下：



1. 你的溢出检测是如何实现的？这里是否可以选择其他的溢出检测方法?

运算结果超出数据类型所能表示的数据范围的现象称为溢出。我们此处根据根据操作数和运算结果的符号位是否一致来进行检测。在有符号运算加法中，只有两个符号相同的数相加时才有可能产生溢出，因此，可以根据操作数与运算结果的符号位是否一致来进行检测。

$$V = X_f Y_f \overline{S_f} + \overline{X_f}\,\overline{Y_f} S_f$$

除了这种方法还有两种方法，一种是利用最高数据位产生的进位信号为 Cd，符号位产生的进位信号为 Cf 进行比较，此处我们由于使用快速加法器无法获得最高数据位产生的进位信号。另一种方法是使用两位符号信息，我们采用补码编制计算，此处事先也不合理，所以只能选择这一种方法。

2. 实现此电路有哪些值得注意的地方？

应注意事先 4 位并行加法器时需要先行利用们器件获得 P 与 G，而在后续的并行加法器设计服用了 4 位并行加法器，故无需再次生成 P 或 G 的信息，直接输入数位信息即可。

2.632 位 MIPS 运算器设计

构建 **32** 位运算器。利用封装好的 32 位加法器以及 logisim 平台中现有运算部件（禁用系统自带的加法器，减法器）构建一个 32 位运算器，可支持算术加、减、乘、除，逻辑与、或、非、异或运算、逻辑左移、逻辑右移，算术右移运算，支持常用程序状态标志（有符号溢出 OF、无符号溢出 CF，结果相等 Equal），运算器功能以及输入输出引脚见下表，在主电路中详细测试自己封装的运算器，在报告中分析该运算器的优缺点。

表 **1.** 芯片引脚与功能描述

引脚	输入/输出	位宽	功能描述
X	输入	32	操作数 X
Y	输入	32	操作数 Y
ALU_OP	输入	4	运算器功能码，具体功能见下表
Result	输出	32	ALU 运算结果
Result2	输出	32	ALU 结果第二部分，用于乘法指令结果高位或除法指令的余数位，其他操作为零
OF	输出	1	有符号加减溢出标记，其他操作为零
UOF	输出	1	无符号加减溢出标记，其他操作为零 溢出条件（加法和小于加数，减法差大于被减数）
Equal	输出	1	Equal=(x==y)?1:0, 对所有操作有效

表 **2.** 运算符功能

ALU OP	十进制	运算功能

0000	0	Result = X << Y　逻辑左移（Y 取低五位） Result2=0
0001	1	Result = X >>>Y　算术右移（Y 取低五位） Result2=0
0010	2	Result = X >> Y　逻辑右移（Y 取低五位） Result2=0
0011	3	Result = (X * Y)[31:0];　Result2 = (X * Y)[63:32]　有符号
0100	4	Result = X/Y;　Result2 = X%Y　无符号
0101	5	Result = X + Y　(Set OF/UOF)
0110	6	Result = X - Y　(Set OF/UOF)
0111	7	Result = X & Y　按位与
1000	8	Result = X | Y　按位或
1001	9	Result = X⊕Y　按位异或
1010	10	Result = ~(X |Y)　按位或非
1011	11	Result = (X < Y) ? 1 : 0　符号比较
1100	12	Result = (X < Y) ? 1 : 0　无符号比较

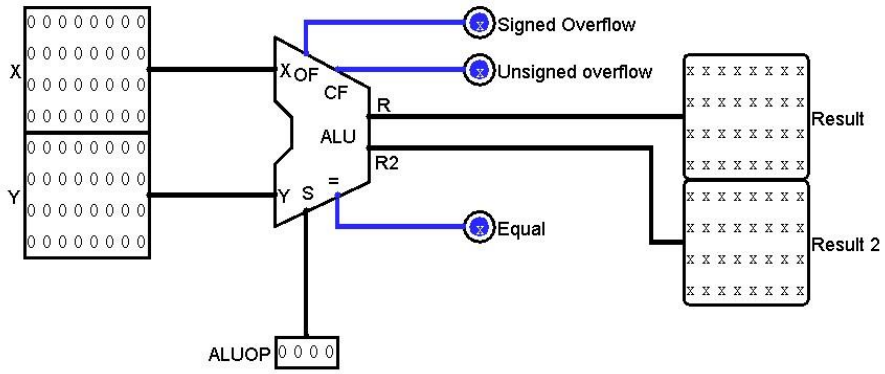
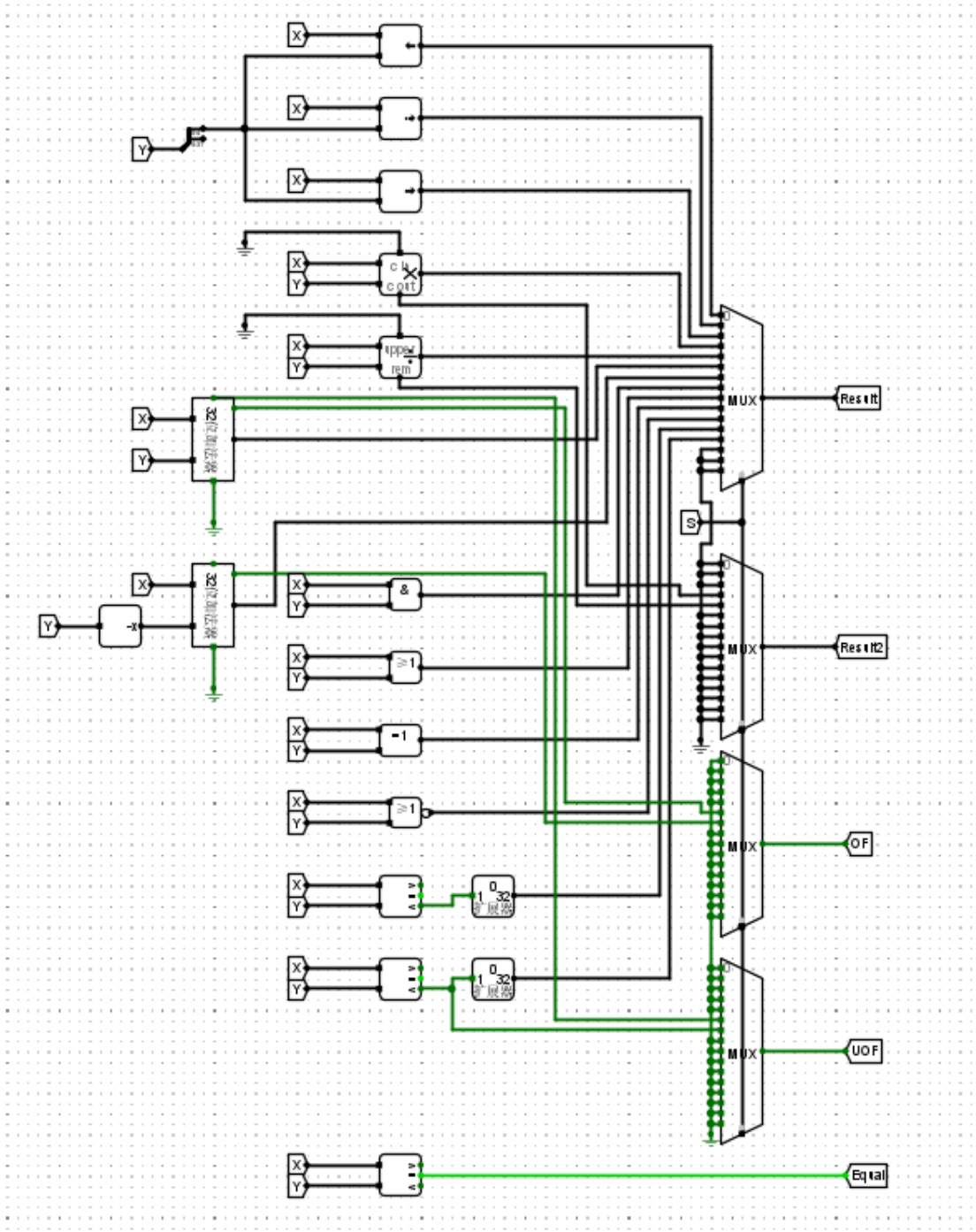


图 6 运算器封装示意图

设计如下：



1. 怎样实现功能选择？

运算器将所有结果算出后，使用多路选择器将结果选择性的输出到 Result1，Result2，OF，UOF，EQUAL。

2. 依次介绍各个功能的实现方式

ALU OP	十进制	运算功能事先方式
0000	0	Result = X << Y　逻辑左移，使用现有移位运算器件 Result2=0
0001	1	Result = X >>>Y　算术右移，使用现有移位运算器件 Result2=0

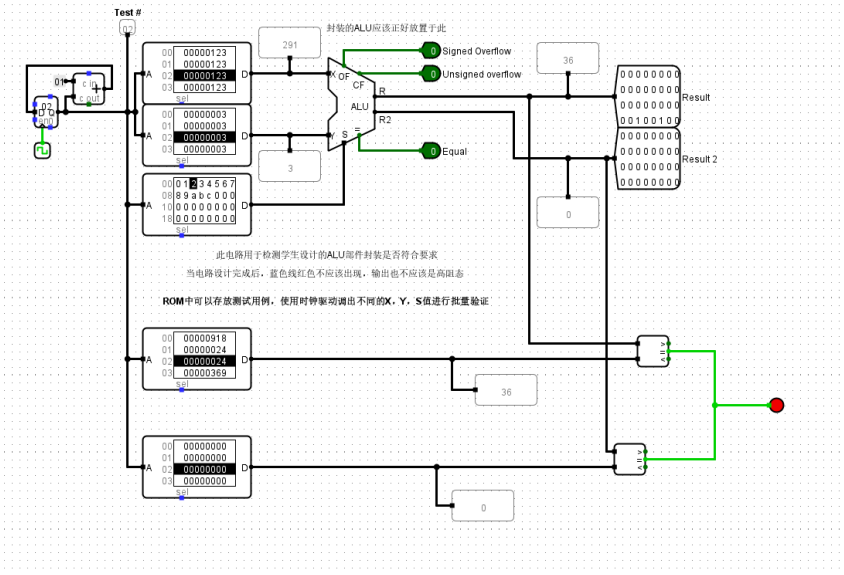
0010	2	Result = X >> Y　逻辑右移，使用现有移位运算器件 Result2=0
0011	3	Result = (X * Y)[31:0];　Result2 = (X * Y)[63:32]　有符号 使用现有乘法器
0100	4	Result = X/Y;　Result2 = X%Y　无符号 使用现有除法器
0101	5	Result = X + Y　(Set OF/UOF) 使用 32 位快速加法器
0110	6	Result = X - Y　(Set OF/UOF) 使用求补器与 32 位快速加法器
0111	7	Result = X & Y　按位与，使用现有逻辑器件
1000	8	Result = X | Y　按位或，使用现有逻辑器件
1001	9	Result = X⊕Y　按位异或，使用现有逻辑器件
1010	10	Result = ~(X |Y)　按位或非，使用现有逻辑器件
1011	11	Result = (X < Y) ? 1 : 0　符号比较，使用现有逻辑器件
1100	12	Result = (X < Y) ? 1 : 0 无符号比较，使用现有逻辑器件

3. 重点介绍 OF，UOF，加法，减法

OF，仅在加减法中使用。我们直接将加法和减法对应的 32 位快速加法器的 OF 位接到对应的多路选择器的对应位即可。
UOF 仅在加减法中使用。若是加法，其无符号数加减是否溢出，看符号位进位即可。无符号减法减法只需判断减数大于被减数即可。

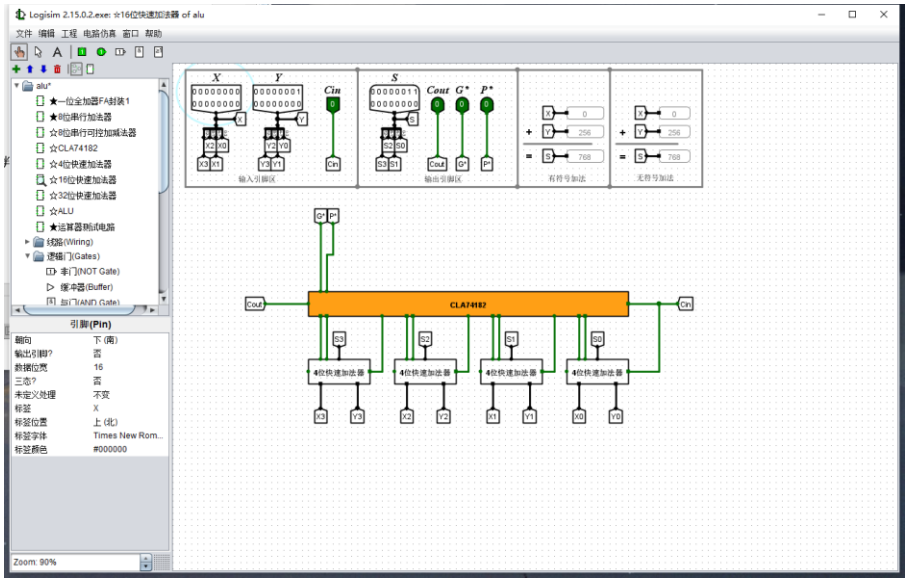
4. 运行运算器测试电路检测

我们进行电路仿真，始终连续运行。我们进行检查，可以得到结果全部为绿线。测试证明其功能已经完备。

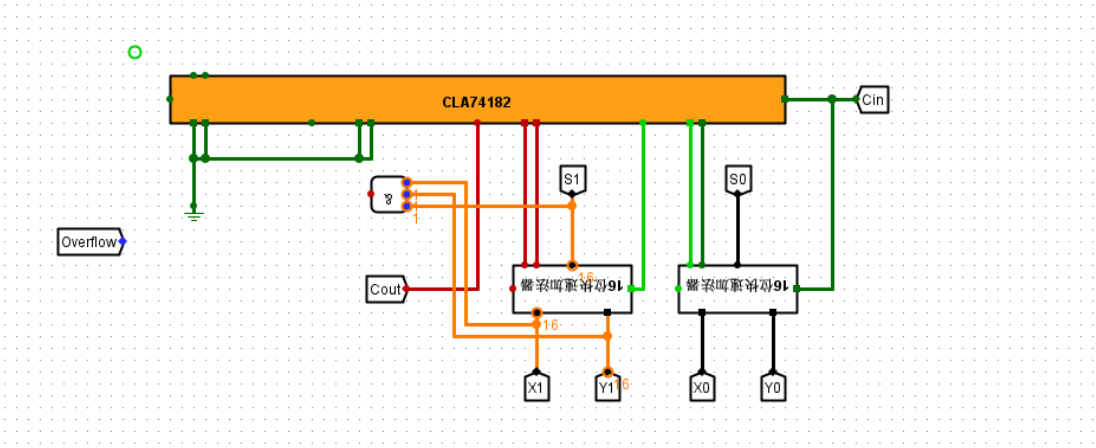


2.7 实验中遇到的问题与收获

我在一开始设计 16 位快速加法器时发现，其中有两段红线，仔细检查后发现是 CLA74182 芯片挡住了两个直接连接在一起的短路线，我们删除后红线就全部变成了绿线。



实验中莫名其妙出现的橙线，其代表着线路位宽对应不对，遇到这类问题，我学会使用分线器选出合适的位信息来。



通过这次实验我充分熟悉了熟悉 Logisim 软件平台。并复习并掌握运算器基本工作原理，运算溢出检测的原理和实现方法，有符号数和无符号数运算的区别；理解基于补码的加/减运算实现原理；并在数字逻辑电路的理解之上，使用 Logisim 软件平台对运算器设计了串行加减法器与并行进位加法器。我分析了运算器的数据传输通路。并利用多路选择器和分线器和现有的组件设计了简单的 ALU。加深了自己对于 CPU 工作原理的理解，更好的认识到了数字电路的设计。

3 存储系统综合实验

3.1 存储扩展实验

实验目的：掌握存储扩展基本原理。

实验内容：设计字库文件，利用指定规格存储器进行存储器字扩展。

实验要求：现有如下 ROM 部件，2 个 4K*16 位 ROM，3 个 4K*32 位 ROM，7 个 16K*32 位 ROM，请构建 GB2312 16*16 点阵字库存储器电路，电路输入为汉字区号和位号，由于 16*16 点阵的字模码需要 256 位点阵信息才能显示一个汉字，所以电路输出为 8*32 位（256 位点阵信息），实验电路输入输出引脚如下图：

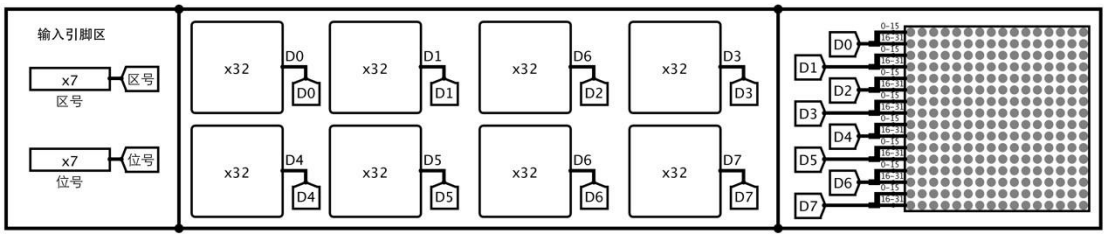


图 1 16*16 点阵字库电路输入输出引脚

本实验的主要目的是进行存储器字扩展（容量扩展，地址总线扩展），故实验工程文件中已经提供了一个参考实现，完成实验所需的点阵信息均可以通过该电路直接导出后载入，也可直接复制拷贝，区位码转存储器地址的电路也可一并参考使用。

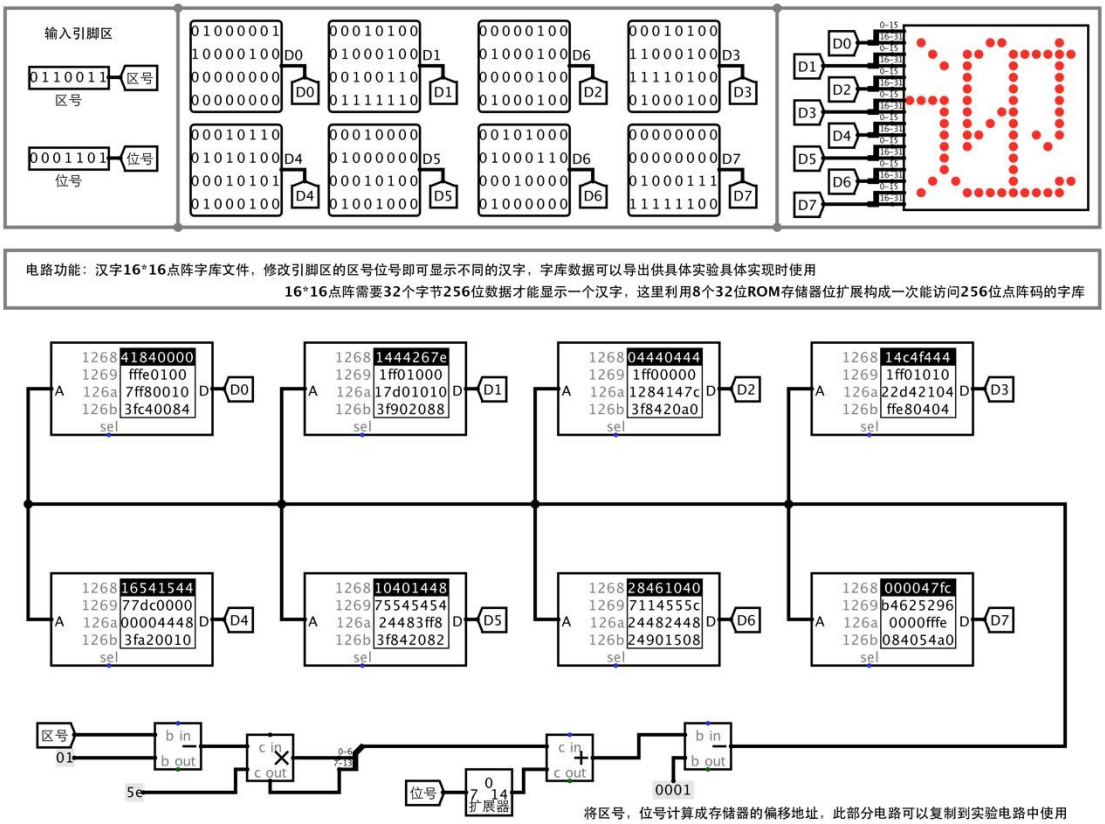


图 2 6*16 点阵字库参考实现

功能测试：

设计实现待测字库后，可以在如下字库测试电路进行功能测试，测试时按下 ctrl+T(command+T　MAC)键启动时钟自动仿真即可，通过对比上下两个显示区显示内容是否一致即可验证字库功能正确性。

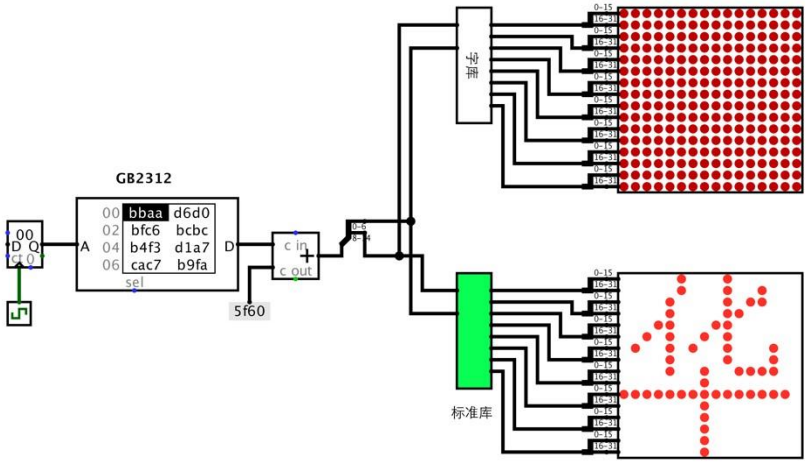
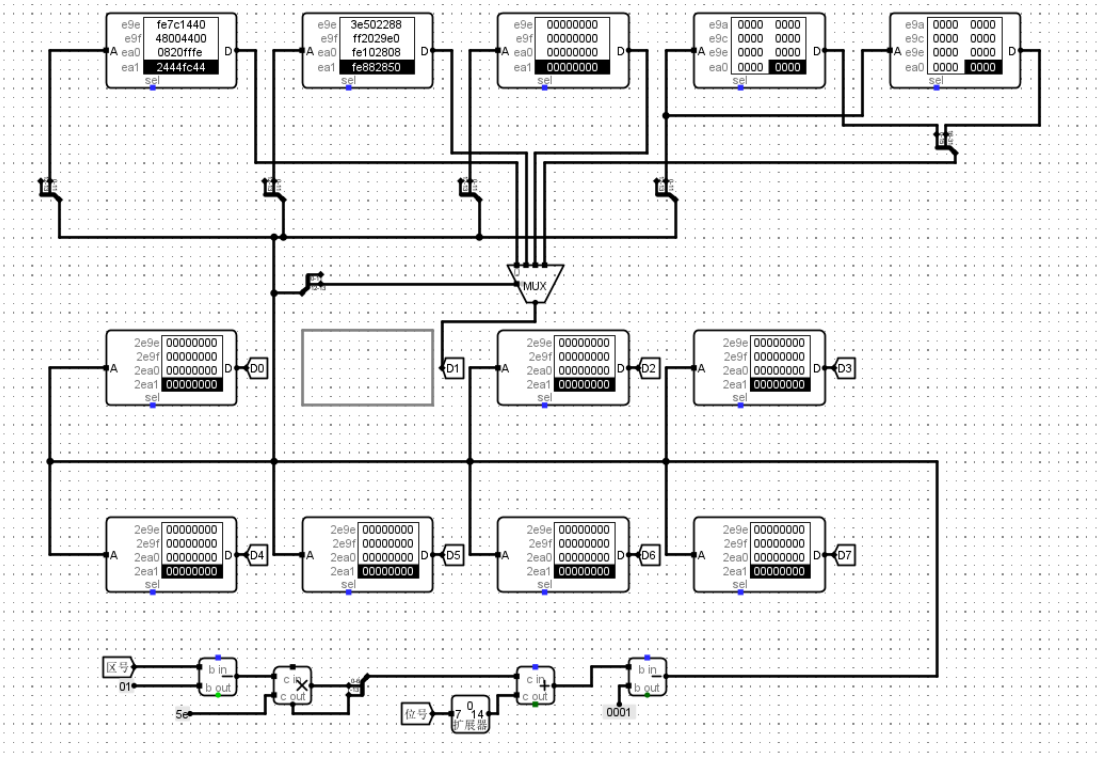


图 3 16*16 点阵字库功能测试电路

我们的存储电路设计如下：



1. 该电路的作用，以及各个引脚的含义

这个电路就是事先在存储器中存储点阵码。对区号和位号经过一系列处理之后计算出存储器的偏移地址，每个偏移地址对应一个 16*16 汉字点阵码，从而获取点阵码后可以进行显示。

2. 为什么该实验需要进行存储扩展？

参考字库电路使用 8 个 16K*32 位 ROM 进行位扩展，实现了一次获取 16*16 点阵屏需要的 256 位点阵码，现在要求使用 2 个 4K*16 位 ROM，3 个 4K*32 位 ROM，7 个 16K*32 位 ROM 来实现相同功能的电路。

与参考电路相比，我们缺少一个 16K*32 位 ROM，多了 2 个 4K16 位 ROM 和 3 个 4K*32 位 ROM。所以利用字扩展（容量扩展，地址线扩展）和位扩展，将现有的 2 个 4K16 位 ROM 和 3 个 4K*32 位 ROM 组织成一个 16K*32 位 ROM

3. 实验中存储器数据和 LED 点阵码是如何对应的？

对区号和位号经过一系列处理之后计算出存储器的偏移地址，每个偏移地址对应一个 16*16 汉字点阵码，点阵码经历了位扩展，我们取出所有存储器中对应位置的点阵码即可，从而获取点阵码后可以进行显示。

4. 实验中存储器扩展的基本思路以及具体实现

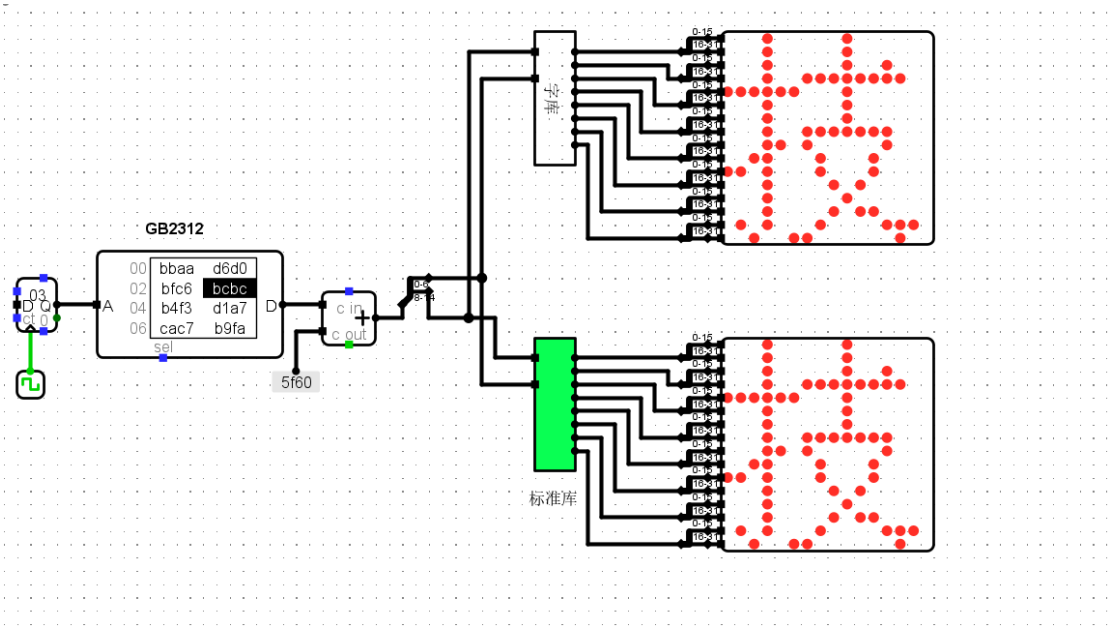
利用位扩展，将现有的 2 个 4K*16 位 ROM 组织成 1 个 4K*32 位 ROM

利用字扩展，将现有的 3 个 4K*32 位 ROM 和上一步得到的 4K*32 位 ROM 组织成一个 16K*32 位 ROM

5. 对于该实验，在进行字位扩展过程中需要注意什么？（提示 :地址空间）

对于电路图中的 3 个 4K*32 位 ROM 和 2 个 4K*16 位 ROM，由于数据已经导入到 ROM 中，所以顺序固定为从左到右地址增加，即在进行字扩展时，注意地址区间的划分

6. 我们对实验进行测试，上下两个输出均一致，输出了华科的一段描述性文字。任意时刻的截屏如下：



3.2 MIPS 寄存器文件设计

实验目的：为 MIPS CPU 构造核心功能部件，进一步熟悉多路选择器，译码器，解复用器等 Logisim 部件的使用。

实验内容：设计完成满足如下规格要求的 MIPS 通用寄存器组。

实验电路输入输出引脚如下图：

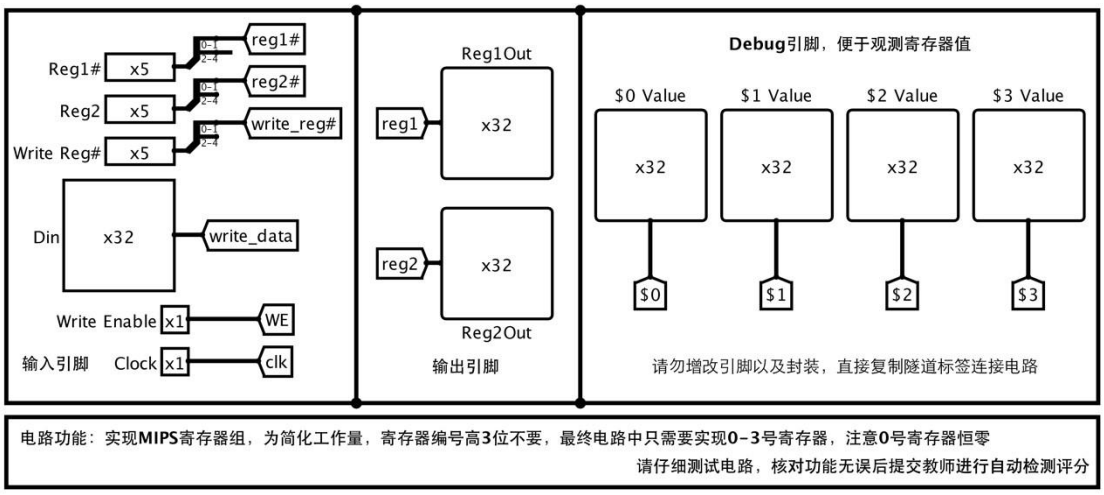


图 3 MIPS Regfile 输入输出引脚

2) 为减少实验中画图工作量，实验工程文件中对 5 位寄存器地址进行了简化，具体见引脚示意图，最终只需实现 4 个寄存器，0 号寄存器功能仍然是恒零。后续实验中如需要使用 32 个寄存器的 MIPS 寄存器文件组，将提供标准组件。

3) 注意时钟信号和电平信号不要混连，时钟仅仅触发状态改变。

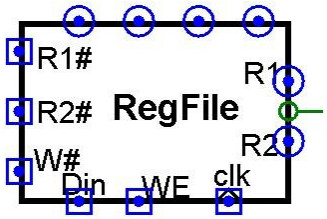
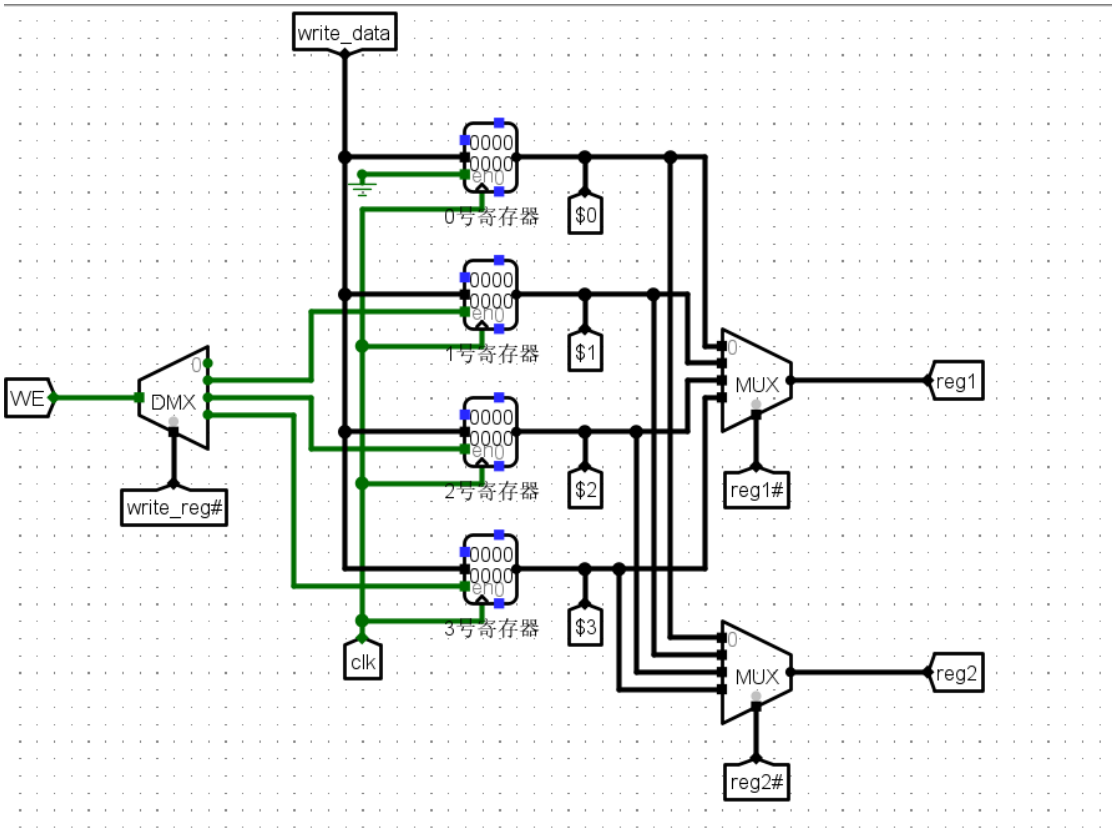


图 4.

MIPS Regfile 封装形式

设计结果如下：



1. 该电路的作用，以及各个引脚的含义

利用 logisim 平台构建一个 MIPS 寄存器组，内部包含 32 个 32 位寄存器，其具体功能如下，具体封装文件为 regfile.circ.

引脚	输入/输出	位宽	功能描述
R1#	输入	5	读寄存器 1 编号
R2#	输入	5	读寄存器 2 编号
W#	输入	5	写入寄存器编号
Din	输入	32	写入数据

WE	输入	1	写入使能信号，为 1 时，CLK 上跳沿将 Din 数据写入 W#寄存器
CLK	输入	1	时钟信号，上跳沿有效
R1	输出	32	R1#寄存器的值
R2	输出	32	R2#寄存器的值
$s0	输出	32	编号为 16 的寄存器的值
$s1	输出	32	编号为 17 的寄存器的值
$s2	输出	32	编号为 18 的寄存器的值
$ra	输出	32	编号为 31 的寄存器的值

2. 如何实现寄存器选择输出？

利用多路复用器（multiplexer），多路复用器可以通过 select bits 选择多个输入中的一个进行输出

因此，我们可以将四个寄存器的输出引脚同时接到多路复用器的输入引脚上，将 reg1#和 reg2#作为多路复用器的 select bits，从而选择指定输入进行输出。

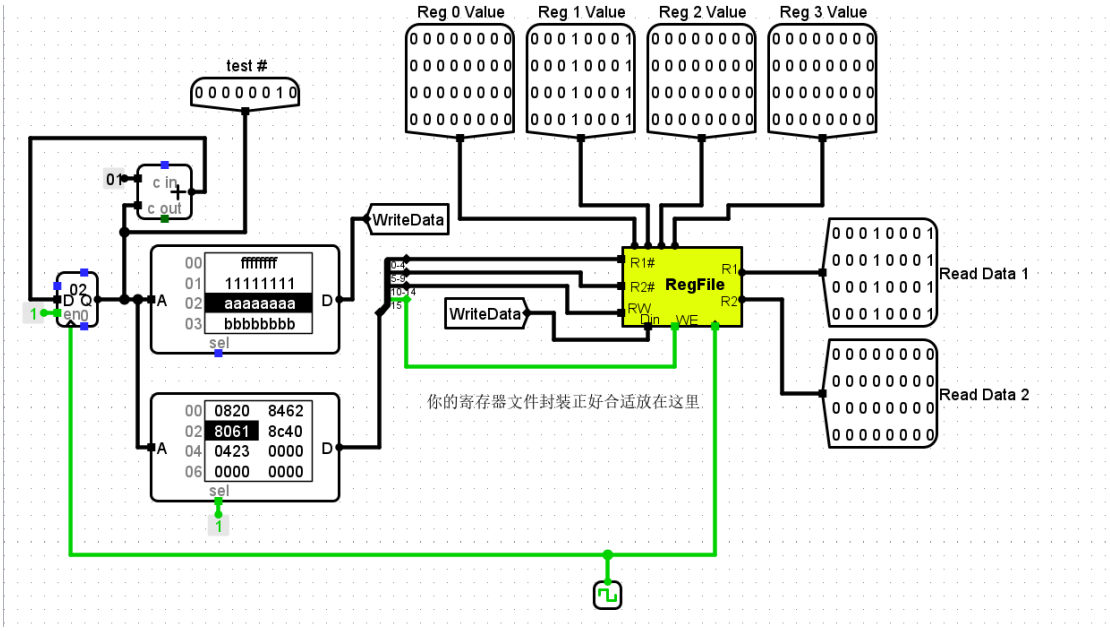
3. 如何实现寄存器选择写入？

利用解复用器（demultiplexer），解复用器可以将输入只通过 select bits 指定的线路输出。

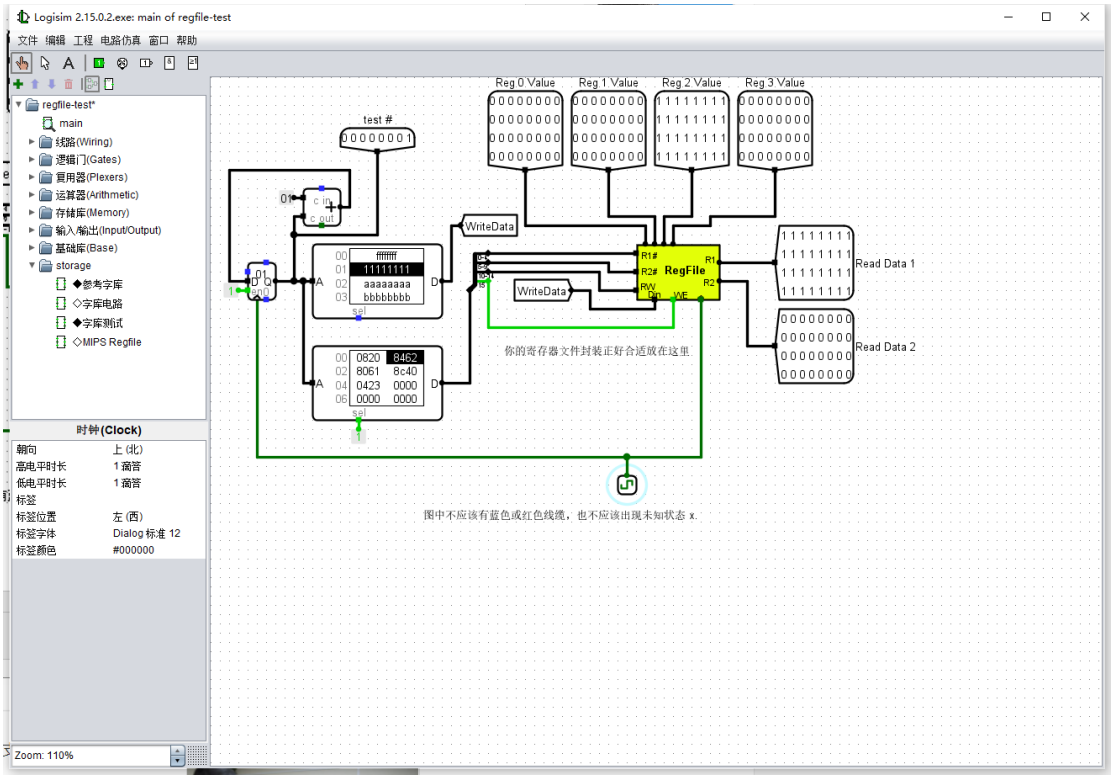
因此，我们可以将 write_data 同时连接到四个寄存器的 data 引脚（由于 0 号寄存器恒为 0，因此可以忽略 data 引脚，或者将 en 置 0），将 we 引脚作为解复用器的输入，将 write_reg#作为解复用器的 select bits，通过解复用器将要写入数据的寄存器的 enable 引脚置 1，下个时钟信号到来时，只有该写入寄存器的 enable 引脚为 1，因此 wrtie_data 数据只会写入到该寄存器中（其余寄存器 enable 引脚为 0，时钟信号不会触发寄存器写入），从而实现寄存器选择写入。

4. 我们利用现有的程序对寄存器组进行测试。

组装好的 ALU 可以将其放入“运算器测试电路”中根据时钟信号测试其功能是否完备。电路运行结果与理论结果一致。我们选取随便一时刻截图如下：


Reg 0 Value
Reg 1 Value
Reg 2 Value
Reg 3 Value
test #
c in
c out
WriteData
RegFile
R1#
R2#
RW
Din
WE
R1
R2
Read Data 1
Read Data 2
00 ffffffff
01 11111111
02 aaaaaaaa
03 bbbbbbbb
sel
00 0820 8462
02 8061 8c40
04 0423 0000
06 0000 0000
你的寄存器文件封装正好合适放在这里

3.3 实验中遇到的问题与收获



我们在一开始测试时发生了比较奇怪的错误，出现了错误的写入，通过调试发现，是因为我们将 DEMUX 的输入设为了 1，应该将 we 引脚作为解复用器的输入，否则就会出现在不应该写入时，发生写入的问题。

经过本次实验我熟悉 ROM、RAM 存储器的使用；对计算机组成原理更加深入理解，对存储系统的理解也更为深刻。掌握存储器字扩展，位扩展的基本原理；并使用 Logisim 模拟了存储器字扩展，位扩展，实现了一个蛮有意思的字码输出小应用。我也更加深入的理解了寄存器的各项功能，为 MIPS CPU 设计功能部件---寄存器文件；

4 心得体会

通过这次实验一我充分熟悉了熟悉 Logisim 软件平台。并复习并掌握运算器基本工作原理,运算溢出检测的原理和实现方法,有符号数和无符号数运算的区别;理解基于补码的加/减运算实现原理；并在数字逻辑电路的理解之上，使用 Logisim 软件平台对运算器设计了串行加减法器与并行进位加法器。我分析了运算器的数据传输通路。并利用多路选择器和分线器和现有的组件设计了简单的 ALU。加深了自己对于 CPU 工作原理的理解，更好的认识到了数字电路的设计。

经过本次实验二我熟悉 ROM、RAM 存储器的使用；对计算机组成原理更加深入理解，对存储系统的理解也更为深刻。掌握存储器字扩展，位扩展的基本原理；并使用 Logisim 模拟了存储器字扩展，位扩展，实现了一个蛮有意思的字码输出小应用。我也更加深入的理解了寄存器的各项功能，为 MIPS CPU 设计功能部件---寄存器文件；

我更加深入的理解了数字电路的设计与分析的基本方法（结合数字逻辑），学会如何从硬件的角度看待 CPU 是如何实现的各种功能，系统的提升了自己的硬件思维，科学思维，实现思维。明白了一个小小的 ALU 也是非常复杂的。