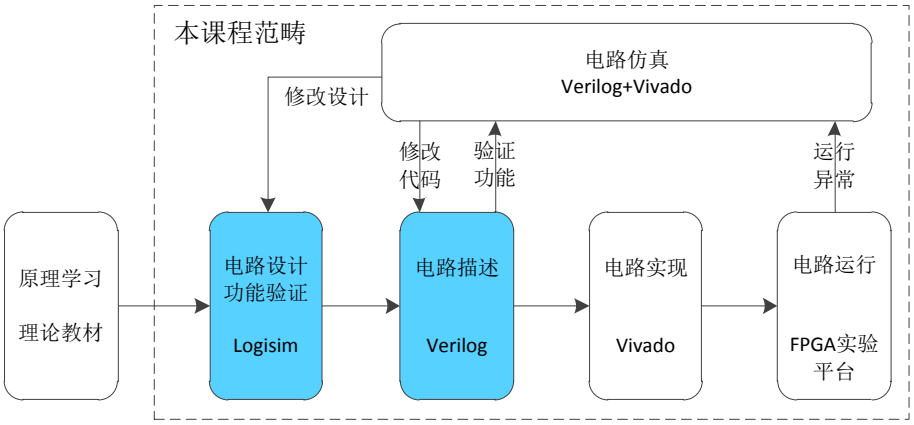
实验 02_简单组合逻辑电路

简介

下图以设计数字电路的一般流程为例，说明了电路设计过程中的关键步骤以及相关工具。



Logisim 可以用来搭建数字逻辑电路并对其功能进行仿真，有助于初学者直观的掌握电路原理及内部构造，以达到快速入门的目的，是一款非常实用的教学工具，但是它的功能仅限于仿真，是对实际电路行为的模拟，而不是真真正正的电路。为了将设计的电路切实的在硬件上运行，我们还需要借助硬件描述语言、综合工具、硬件平台的支持。本次实验中我们会介绍到一种叫做 Verilog HDL 的硬件描述语言，工具及平台我们会通过后续的课程逐步介绍。

本次实验我们将进一步使用 Logisim 设计组合逻辑电路，并使用 Verilog 语言对设计的电路进行描述。

实验目的

熟练掌握 Logisim 的基本用法

进一步熟悉 Logisim 更多功能

用 Logisim 设计组合逻辑电路并进行仿真

初步学习 Verilog 语法

实验环境

PC 一台,能流畅的连接校园网

Logisim 仿真工具

vlab.ustc.edu.cn（jre、Logisim 工具以及 Verilog 语法介绍都可在此网站获取）

实验步骤

Step1：用真值表自动生成电路

通过前面实验的学习，用户可以使用 Logisim 管理窗中的各种组件以及自己设计的子电路，设计出各种功能的组合逻辑电路，但是在搭建电路时大量的拖拽、布局、连线非常费时费力，这里介绍一种通过真值表生成电路的方法。以下面真值表为例，我们要设计该电路，一般的做法是：

——根据真值表画出各输出项的卡诺图

——通过卡诺图写出各输出项的逻辑表达式

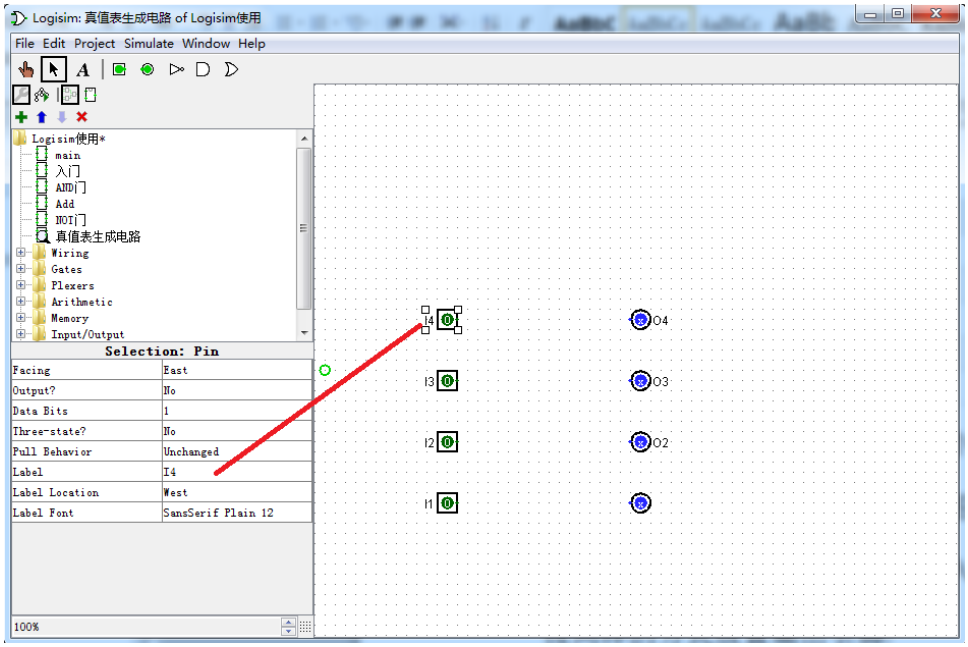
——根据逻辑表达式画出电路图，完成电路设计

输入	输出
0001	1010
0011	0111
1010	0011
1011	0110
1111	0101

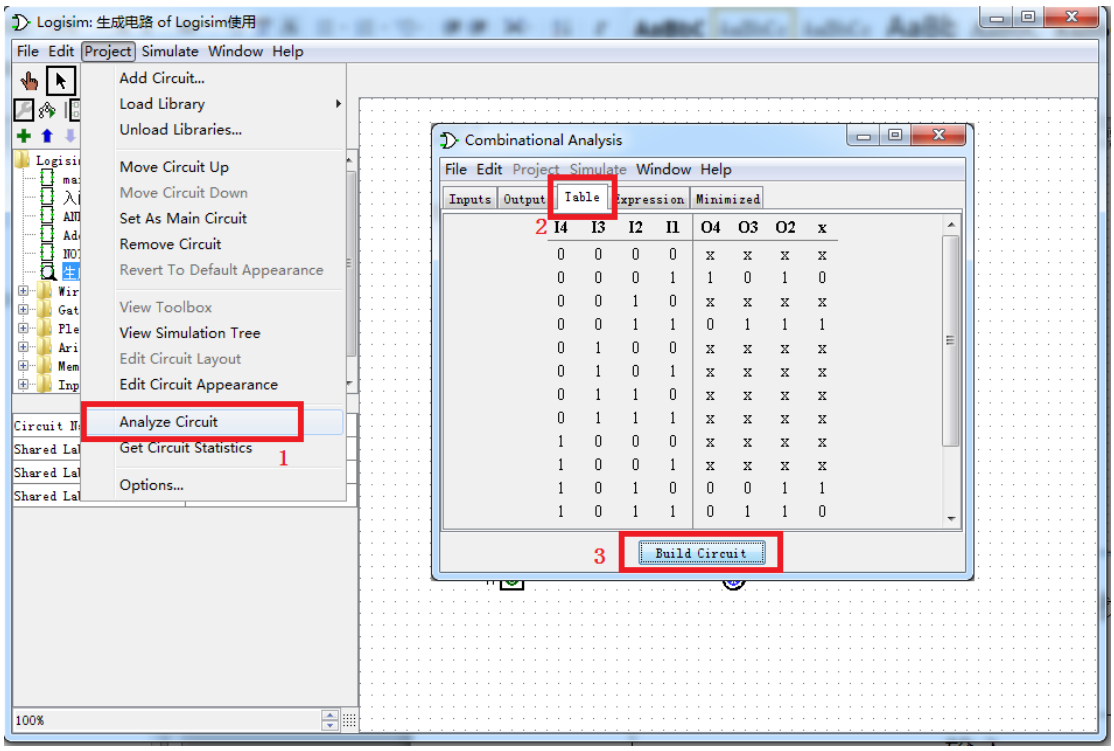
Logisim 能够帮我们完成上述步骤中大部分的工作

首先在 Logisim 中新建一个电路图，命名为“生成电路”（电路

名可以任取），然后再电路图中放置输入引脚，有几个输入就放几个引脚，按同样的方式放置输出引脚。放置完毕后，给所有引脚标上标号，并按高低位顺序排列。



在菜单栏的“Project”选项卡中找到“Analyze Circuit”选项，并选中。在弹出的窗口中选择“Table”选项，按照前面的真值表修改输出值（鼠标点击输出信号对应的叉号就可修改），最后点击“Build Circuit”便可生成电路（弹出的对话框都选择“是”）。



Step2：用表达式生成电路图

通过真值表生成电路确实能为我们减少工作，但是也存在不足之处，我们知道，真值表条目数与输入项个数呈指数相关，当输入信号数量较多时，编辑真值表也是一项非常繁重的工作，以下表为例，这是一个 8 位的优先编码器电路，其完整的真值表有 256 项之多。

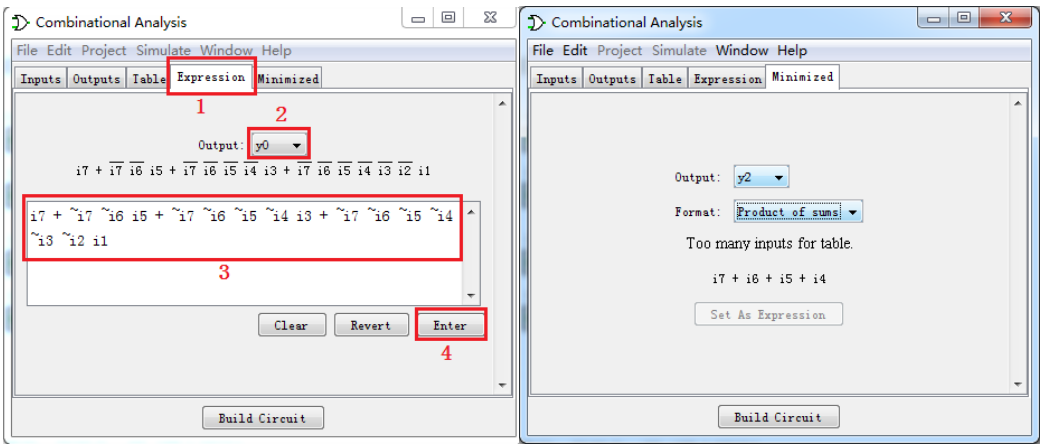
输入								输出		
i7	i6	i5	i4	i3	i2	i1	i0	y2	y1	y0
1	x	x	x	x	x	x	x	1	1	1
0	1	x	x	x	x	x	x	1	1	0
0	0	1	x	x	x	x	x	1	0	1
0	0	0	1	x	x	x	x	1	0	0
0	0	0	0	1	x	x	x	0	1	1
0	0	0	0	0	1	x	x	0	1	0
0	0	0	0	0	0	1	x	0	0	1
0	0	0	0	0	0	0	1	0	0	0

根据真值表，我们可以很快的写出各输出信号的表达式：

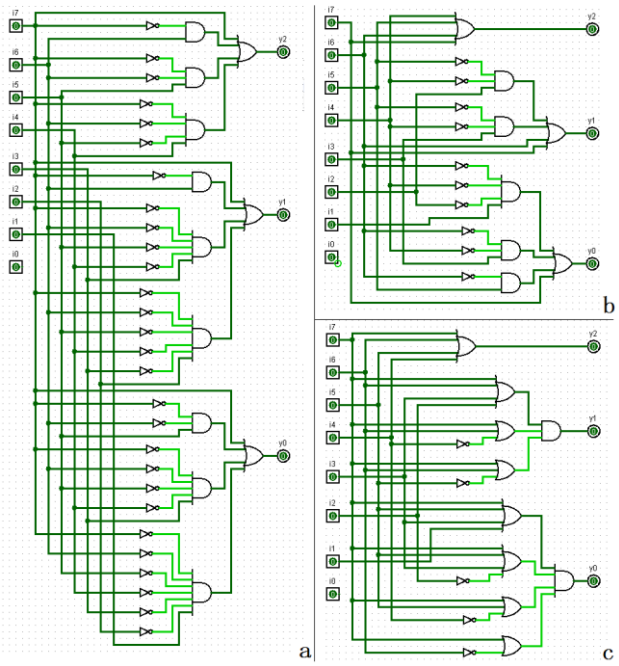
y2 = i7 + ~i7 i6 + ~i7 ~i6 i5 + ~i7 ~i6 ~i5 i4

y1 = i7 + ~i7 i6 + ~i7 ~i6 ~i5 ~i4 i3 + ~i7 ~i6 ~i5 ~i4 ~i3 i2

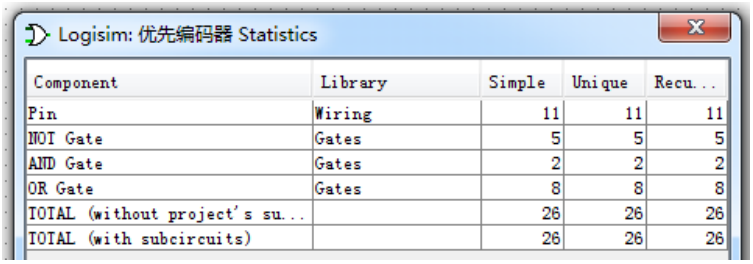
y0 = i7 + ~i7 ~i6 i5 + ~i7 ~i6 ~i5 ~i4 i3 + ~i7 ~i6 ~i5 ~i4 ~i3 ~i2 i1



我们可以在 Logisim 中直接输入表达式生成电路，在“Project” --> “Analyze Circuit”的弹出窗口中选择“Expression”选项，填入每个输出信号的表达式。最后点击“Build Circuit”生成电路。有时候手动输入的表达式并不是最简形式，最终生成的电路也会占用较多的逻辑门，我们可以借助“Minimized”选项卡对表达式进行简化，进而减少电路使用的逻辑门数量，电路输入信号不多的情况下，该窗口还能显示卡诺图。下图对比了同一功能电路未化简的电路结构和化简后的与或式和或与式结构，可以看出占用逻辑门数量有明显差异。



我们还可以通过“Project”--> “Get Circuit Statistics”选项统计电路的基本信息。



Component	Library	Simple	Unique	Recu...
Pin	Wiring	11	11	11
NOT Gate	Gates	5	5	5
AND Gate	Gates	2	2	2
OR Gate	Gates	8	8	8
TOTAL (without project's su...		26	26	26
TOTAL (with subcircuits)		26	26	26

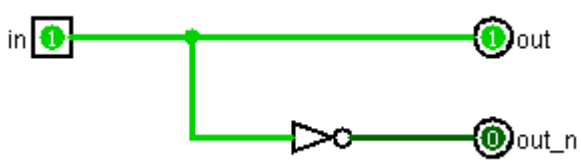
Logisim 的自动生成电路功能，能为用户带来便利，节省大量时间，但也有一点小小的不足，其输入输出信号必须是单 bit 位宽，对于多 bit 位宽的输入信号并不支持，需要将其拆分成多个单 bit 信号才可以。

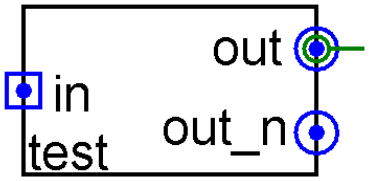
Step3：Verilog HDL 语法入门

下面我们通过对照 Logisim 中设计的简单电路来学习 Verilog 语法。

例 1. 如下图所示，左侧为电路结构，右侧为电路封装图，可以看到，该电路包含一个输入，取名为 in 信号（信号可自行命名），两个输

出，out 信号与输入直连，out_n 信号为输入信号取反，该电路模块我们命名为 test





下面以此为例介绍 Verilog 的写法。编写 Verilog 代码前首先需要新建一个文本文件，修改后缀并重命名为 test.v，该文件可以使用各类文本编辑器进行编辑。代码内容为：

```
module test(  //模块名称
input in,     //输入信号声明
output out,   //输出信号声明
output out_n);
//如需要，可在此处声明内部变量
/*******以下为逻辑描述部分******/
    assign out = in;
    assign out_n = ~in;
/*******逻辑描述部分结束******/
endmodule //模块名结束关键词
```

其中蓝色字体部分是 Verilog 的关键字(如 C 语言中也有关键字)，这些关键字都有特定含义和专门的用途，不可以用来作为信号名或模块名。上述代码虽然简单，但包含了 Verilog 模块的最基本结构。

```
module  模块名(
    输入端口声明,
    输出端口声明);
    内部信号声明<可选>;

    逻辑描述（模块主体）
endmodule
```

每个模块都是以关键字 module 开头，以 endmodule 结束。module 后面是模块名，括号内是输入输出信号的声明（任何一个有实际功能的电路都应该有输入输出，但也存在例外，比如后续讲到对电路进行

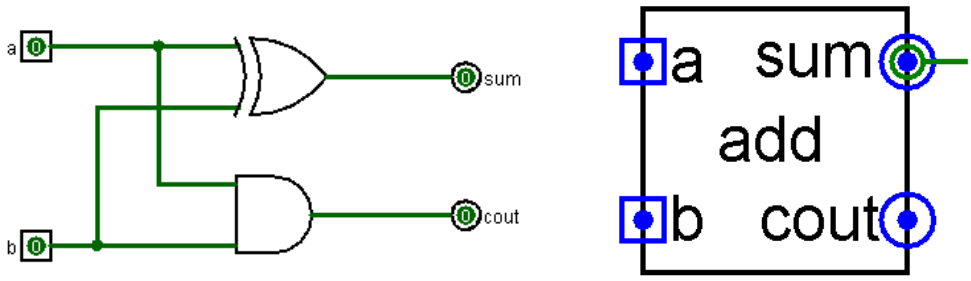
仿真时，其仿真激励文件一般就没有输入输出信号）。

如果模块功能较复杂的话，可能会用到一些中间信号，那就要在模块内部声明，此例中没有用到，所以没有声明。

逻辑描述部分是每个模块的主体，用于描述该电路的行为特性，其语法还算简单，相信读者可以很容易就能看懂。这里用到了一个非常重要的关键字“assign”，该关键字放在逻辑表达式之前，用于表明后面是一条连续赋值语句，一般来说，对组合逻辑的赋值都可以使用该关键字实现。

此外，同很多编程语言一样，Verilog 中也可以有注释，单行注释以“//”开始，多行注释则使用“/* 注释内容 */”，注释内容仅仅是为了增加代码可读性，不会对代码功能产生影响。

例 2. 下图左侧为半加器电路结构图，右侧为电路封装图



其 Verilog 代码为：

```
module add(
input a,b,
output sum,cout);
assign {cout,sum} = a + b;
endmodule
```

此例中使用了位拼接功能，我们知道输入 a,b 都是单 bit 信号，但他们相加的结果需要一个两 bit 位宽的信号才能保存，而 cout 和 sum 都是单 bit 的信号，通过使用{ }（位拼接符号）将两个单 bit

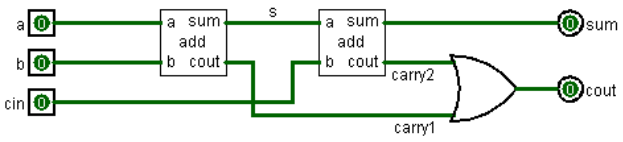
信号拼接成了一个 2bit 信号，用于接收相加的结果。

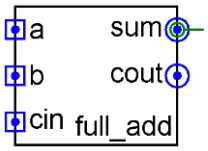
此外此电路从行为级上进行了描述，直接描述了两个输入信号相加这一行为，实际上下面这段代码与前面所描述的电路是完全一样的，因为它们的行为特性一样。

```
module add(
input a,b,
output sum,cout);
assign cout = a & b;
assign sum  = a ^ b;
endmodule
```

在上面这段代码中有两条连续赋值语句，它们的顺序交换并不会对电路产生影响，换句话说，他们是位置无关的，并不是前面的先执行后面的后执行，这点通过对比代码和前面的电路图可以很容易理解，但之前学过 C 语言等软件编程语言的读者反而容易产生困扰，需要特别注意。

例 3. 利用前面例子中所设计的半加器，构造一个全加器，其电路结构图和电路封装图如下所示





其 Verilog 代码如下：

```
module full_add(
input  a,b,cin,
output sum,cout);
wire s,carry1,carry2;
add   add_inst1(
.a    (a   ),
.b    (b   ),
.sum  (s   ),
.cout (carry1));
add add_inst2(
```

```
.a    (s   ),
.b    (cin ),
.sum  (sum ),
.cout (carry2));
assign cout = carry1 | carry2;
endmodule
```

本例中用到了内部信号声明，关键字 wire 表明声明的信号为线网类型，对于这种信号类型，可以简单的理解为电路中的导线，可以通过 assign 关键字进行赋值的信号都是这种类型，wire 类型是 verilog 中的默认类型，凡是没有明确声明类型的信号，都被当作 wire 类型处理。

模块调用在 Verilog 中也非常重要，在电路较复杂时，我们需要将其分解成若干个子电路，最后再将子电路整合，或者复用第三方以及自己之前设计的功能模块时，都需要用到模块调用。在本例中，我们调用了两个半加器，以实现全加器的功能，其中 add 为被调用模块的模块名，此名称不可随意改动，必须与被调用模块的名称完全一致，add_inst1、add_inst2 为实例化名称，该名称可自行指定。add 模块被实例化了两次，那在最终的电路中就会实实在在的出现两个半加器，它们的行为特性完全一样，只不过各自的输入输出信号不同，工作时也相互独立，互不影响。

实验练习

题目 1：依据如下真值表，通过 Logisim 编辑真值表功能，完成电路设计。电路下方需标注姓名学号。

输入			输出	
Ci-1	Ai	Bi	Si	Ci
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

题目 2：根据下列真值表，通过 Logisim 的编辑表达式功能完成电路设计，电路下方需标注姓名学号。

输入						输出							
G1	G2	G3	A2	A1	A0	Y7	Y6	Y5	Y4	Y3	Y2	Y1	Y0
X	1	X	X	X	X	1	1	1	1	1	1	1	1
X	X	1	X	X	X	1	1	1	1	1	1	1	1
0	X	X	X	X	X	1	1	1	1	1	1	1	1
1	0	0	0	0	0	1	1	1	1	1	1	1	0
1	0	0	0	0	1	1	1	1	1	1	1	0	1
1	0	0	0	1	0	1	1	1	1	1	0	1	1
1	0	0	0	1	1	1	1	1	1	0	1	1	1
1	0	0	1	0	0	1	1	1	0	1	1	1	1
1	0	0	1	0	1	1	1	0	1	1	1	1	1
1	0	0	1	1	0	1	0	1	1	1	1	1	1
1	0	0	1	1	1	0	1	1	1	1	1	1	1

题目 3：使用 Logisim 绘制 1bit 位宽的二选一选择器电路图，并根据生成的电路图编写 Verilog 代码。输入信号为 a, b, sel，输出信号为 out, sel 为 0 时选通 a 信号。

题目 4：通过例化题目 3 中的二选一选择器，用 Verilog 实现一个四选一选择器，并画出对应的电路图。输入信号为

a, b, c, d, sel1, sel0, out， sel1 和 sel0 都为 0 时选中 a 信号。

题目 5:根据前面用到的八位优先编码器真值表,编写 verilog 代码。

输入								输出		
i7	i6	i5	i4	i3	i2	i1	i0	y2	y1	y0
1	x	x	x	x	x	x	x	1	1	1
0	1	x	x	x	x	x	x	1	1	0
0	0	1	x	x	x	x	x	1	0	1
0	0	0	1	x	x	x	x	1	0	0
0	0	0	0	1	x	x	x	0	1	1
0	0	0	0	0	1	x	x	0	1	0
0	0	0	0	0	0	1	x	0	0	1
0	0	0	0	0	0	0	1	0	0	0

题目 6：阅读如下 Verilog 代码，描述其功能，并画出其对应的电路图。

```
module test(
input a,b,c,
output s1,s2);
assign s1= ~a &~b & c | ~a & b &~c | a &~b &~c | a & b & c;
assign s2= ~a & b & c | a &~b & c | a & b &~c | ~a &~b &~c;
endmodule
```

总结与思考

1.请总结本次实验的收获

2.请评价本次实验的难易程度

3.请评价本次实验的任务量

4.请为本次实验提供改进建议