# 中国科学技术大学计算机学院
# 《数字电路实验》报告


实验题目：简单组合逻辑电路

学生姓名：林宸昊

学生学号：PB20000034

完成日期：2021.10.21

计算机实验教学中心制

2020 年 09 月

【实验题目】简单逻辑组合电路

【实验目的】

1. 在第一次实验的基础上熟练掌握 Logisim 基本用法；
2. 进一步熟悉 Logisim 的更多功能；
3. 用 Logisim 设计组合逻辑电路而非直接搭建，并进行仿真；
4. 初步学习 Logisim 语法

【实验环境】

1. 个人 PC 一台
2. Logisim 仿真工具
3. vlab.ustc.edu.cn 提供的虚拟机

【实验过程】

Step 1：用真值表自动生成电路

Step1.1：新建电路图，放置引脚并标号

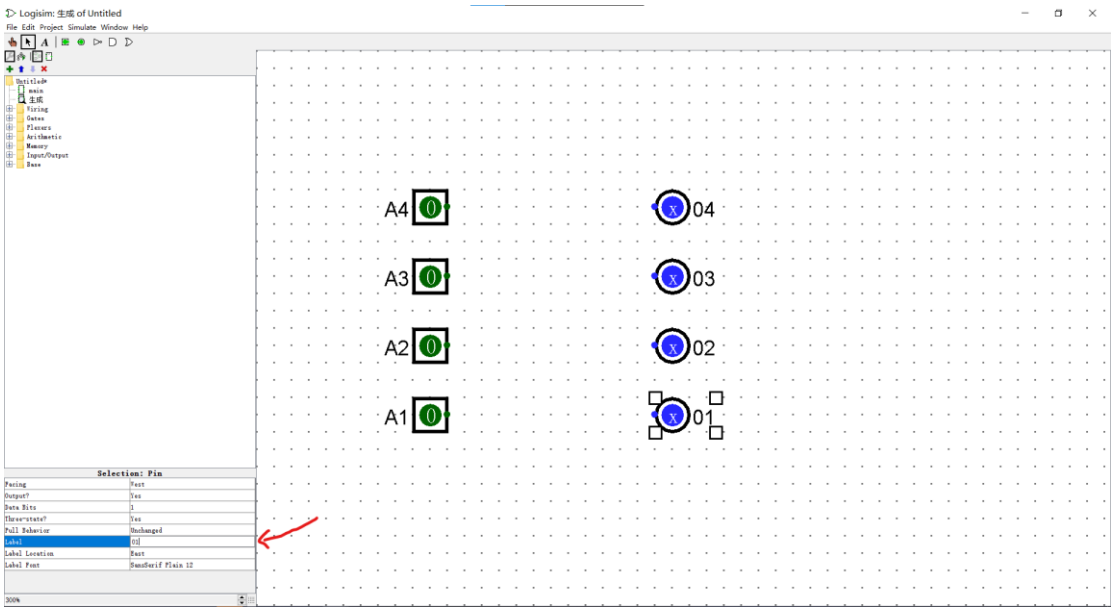


图 1.1.1：引脚标号

## Step1.2：通过”Table”设置这真值表并选择“Build Circuit”自动生成电路

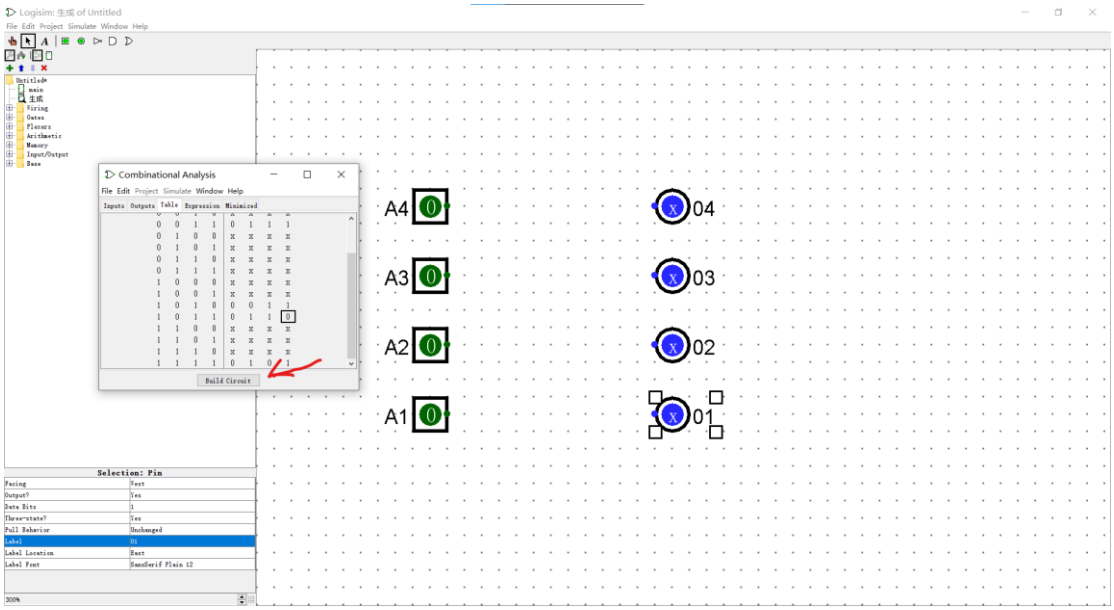



图 1.2.1：设置真值表

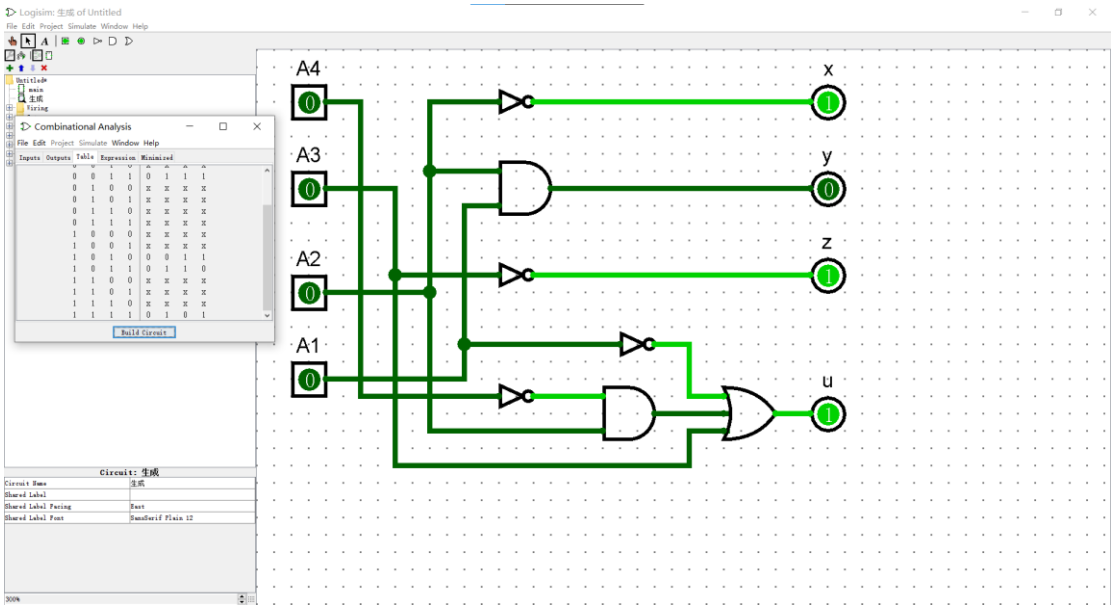



图 1.2.2：最终生成电路图

## Step 2：用表达式生成电路

### Step2.1：首先设置引脚

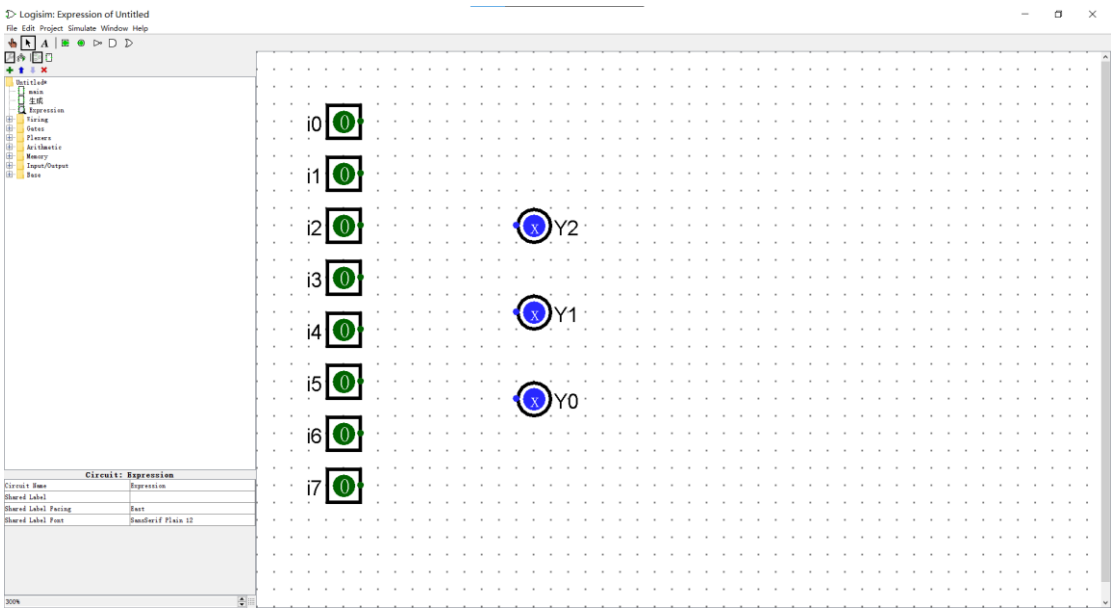



图 2.1.1：设置引脚并标号

## Step2.2：选择表达式窗口“Expression”

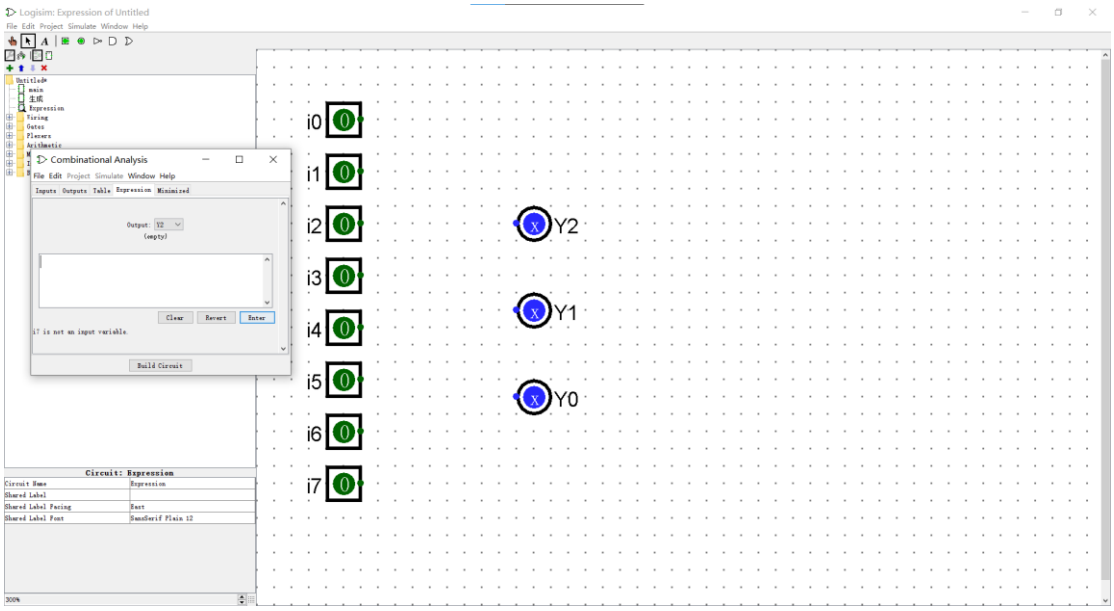



图 2.1.2：选择“Expression”窗口输入表达式

## Step2.3：对每个输出输入对应表达式

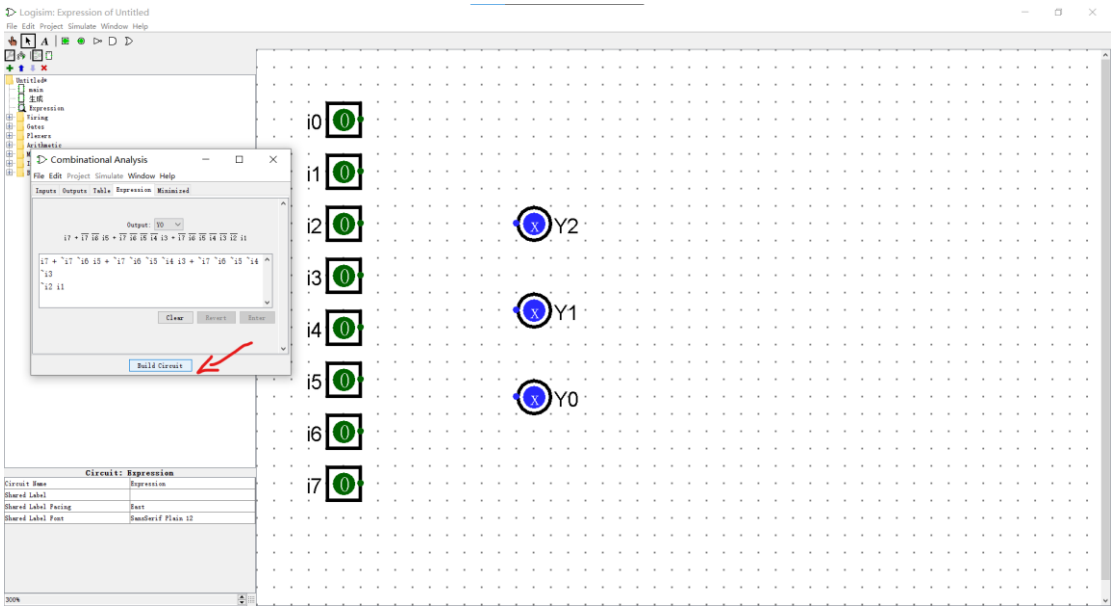



图 2.3.1：输入表达式

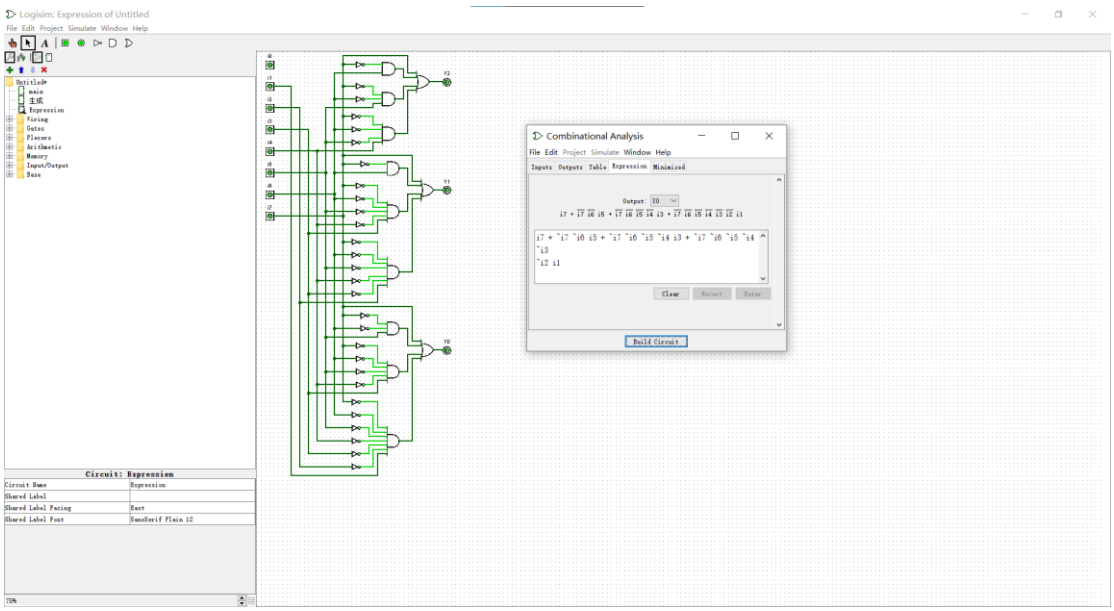



图 2.3.2：最终生成电路图

## Step2.4：使用“minimize”对电路做适当的优化

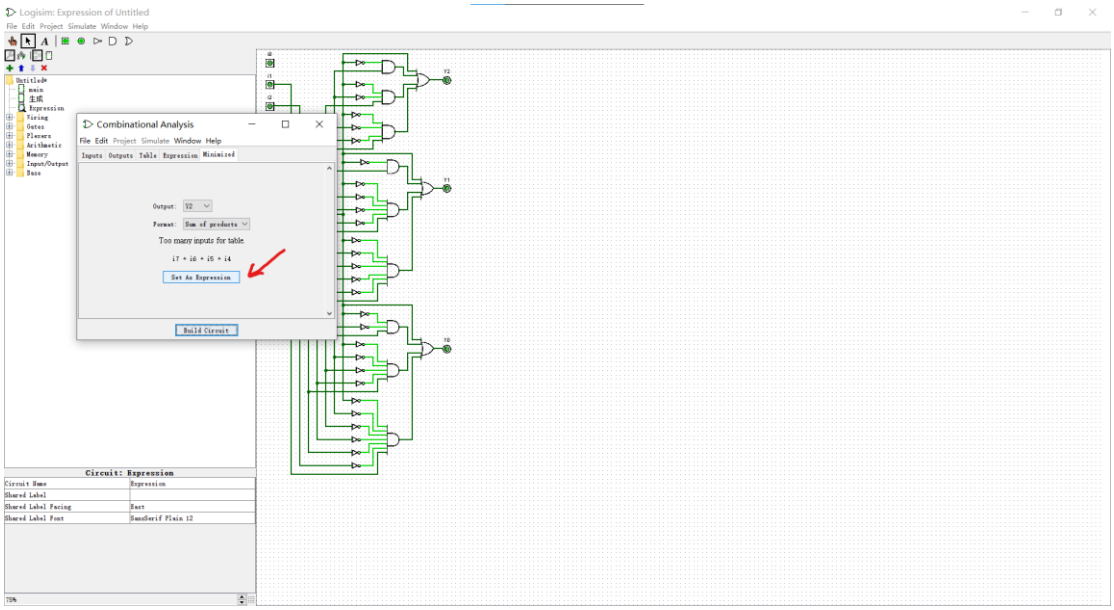



图 2.4.1：将简化后的表达式选作最终表达式

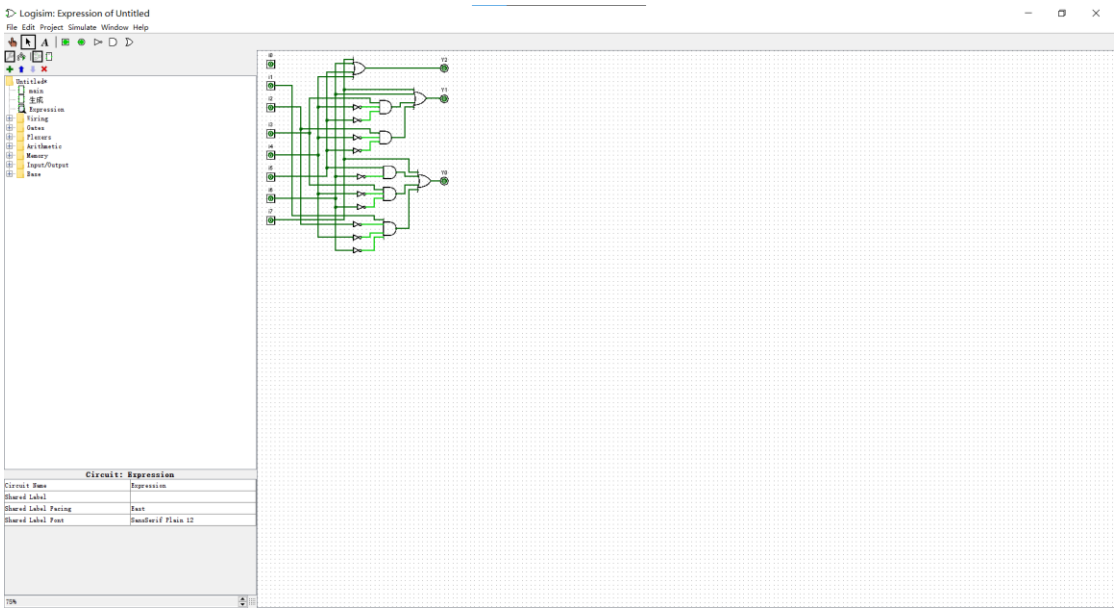



图 2.4.2：简化后的电路图

可见，简化表达式后生成的电路简单很多。

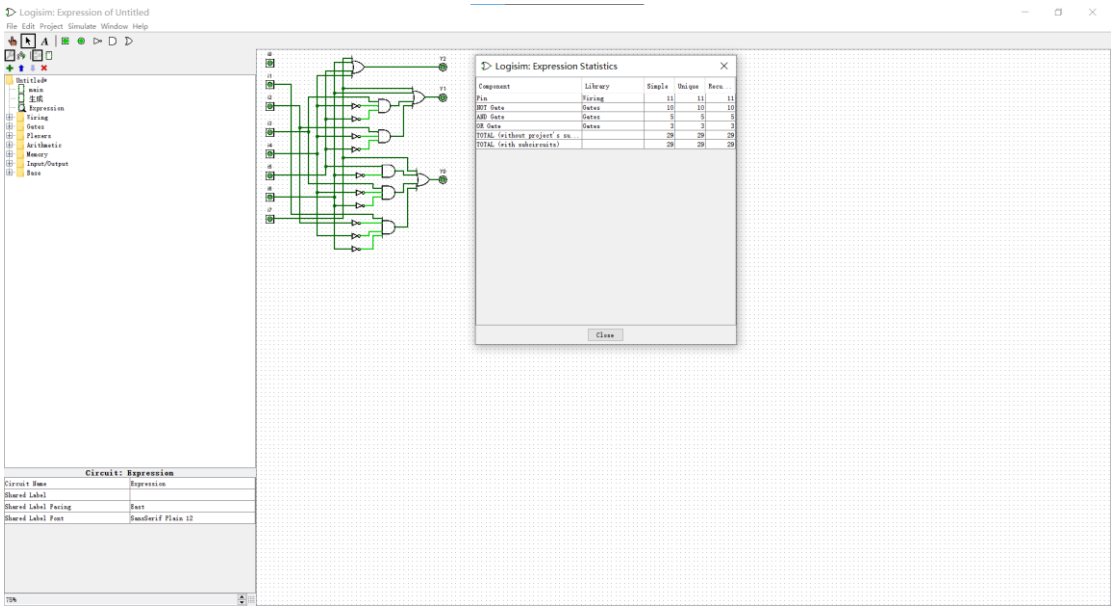
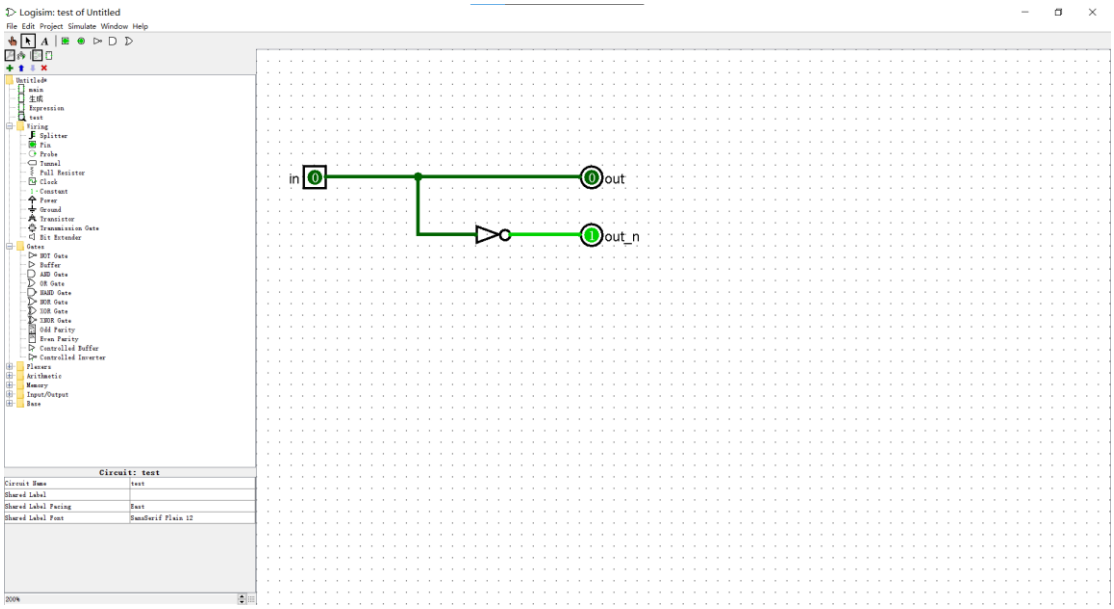
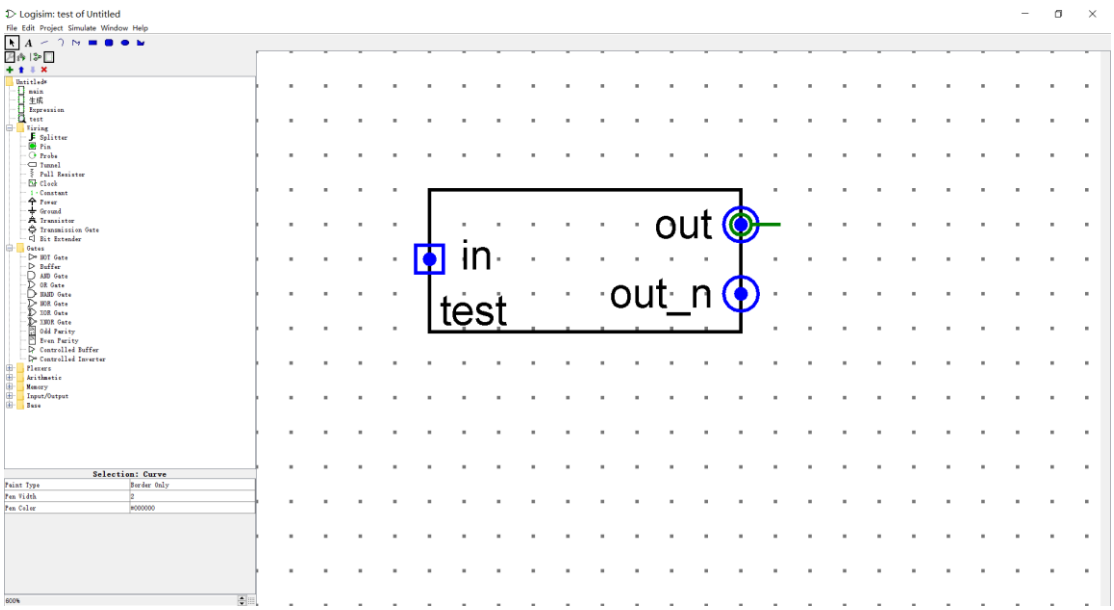



图 2.4.3：通过“Get Circuit Statistics”获得电路基本信息

## Step 3：Verilog HDL 语法入门

### 例1. 编写图示电路的 verilog 代码






如图，该电路包含一个输入取名为 in，两个输出，其中 out 输出与输入直连，out_n 输出为输入信号取反，命名为 test。对此创建 verilog 代码

```
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);
//声明模块后进行逻辑描述
  assign out = in;
  assign out_n = ~in;
//逻辑描述结束
endmodule    //模块结束关键词
```

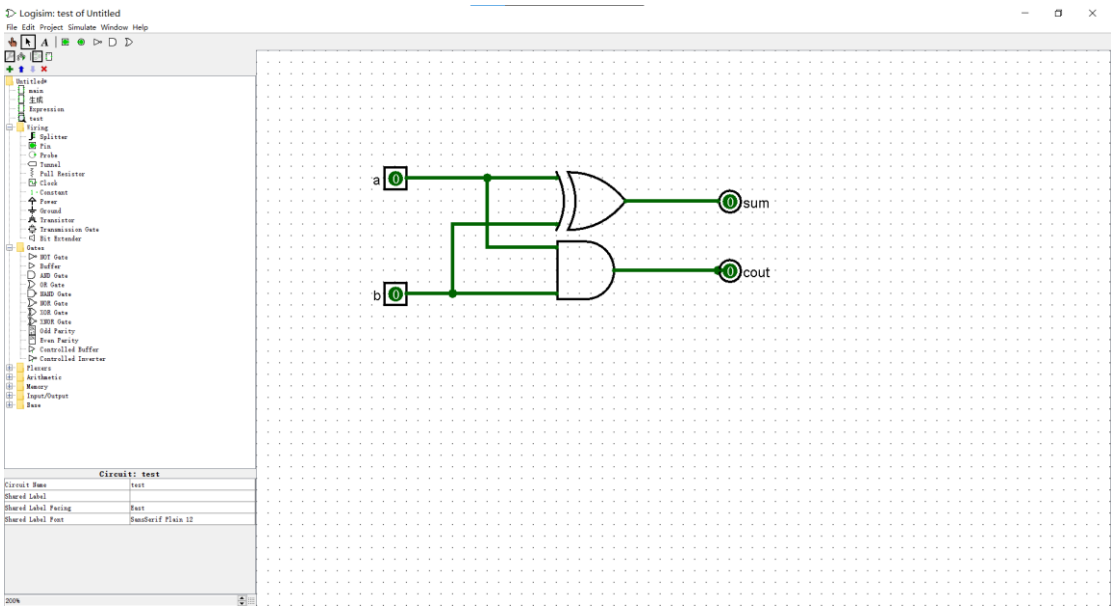
Verilog 模块的最基本架构如上已经显示出来了

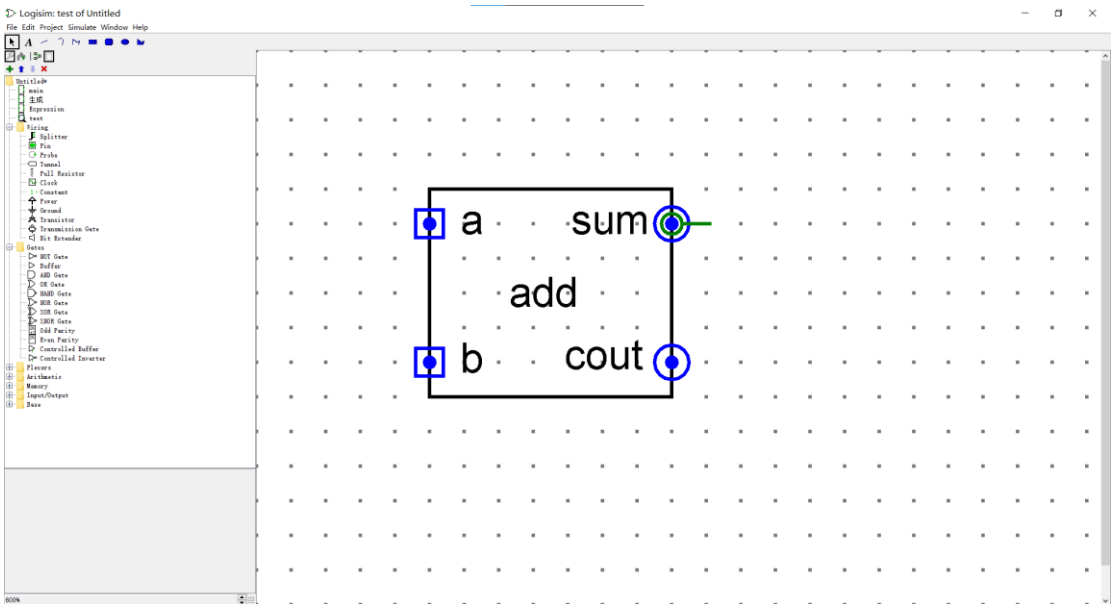
```
module 模块名(
  输入端口声明,
  输出端口声明);
  内部信号声明<可选>;

  逻辑描述(模块主体)
endmodule
```

例2. 编写半加器的 verilog 代码

Step 1：首先绘制出相应电路图




Step 2：编写 Verilog 代码

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

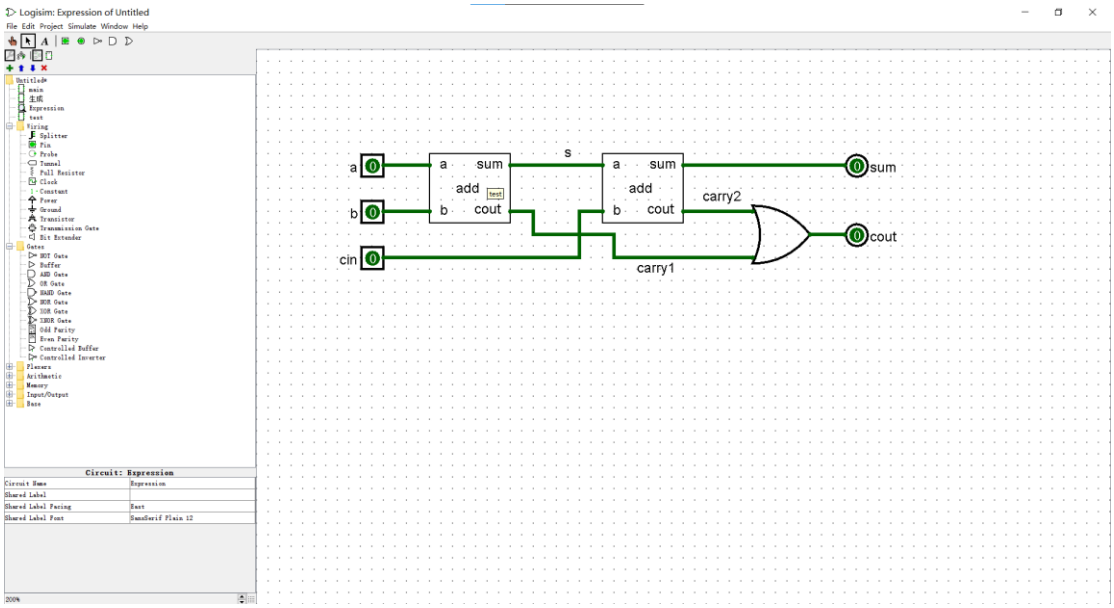
在这里我们使用了位拼接功能通过使用{ }符号将两个

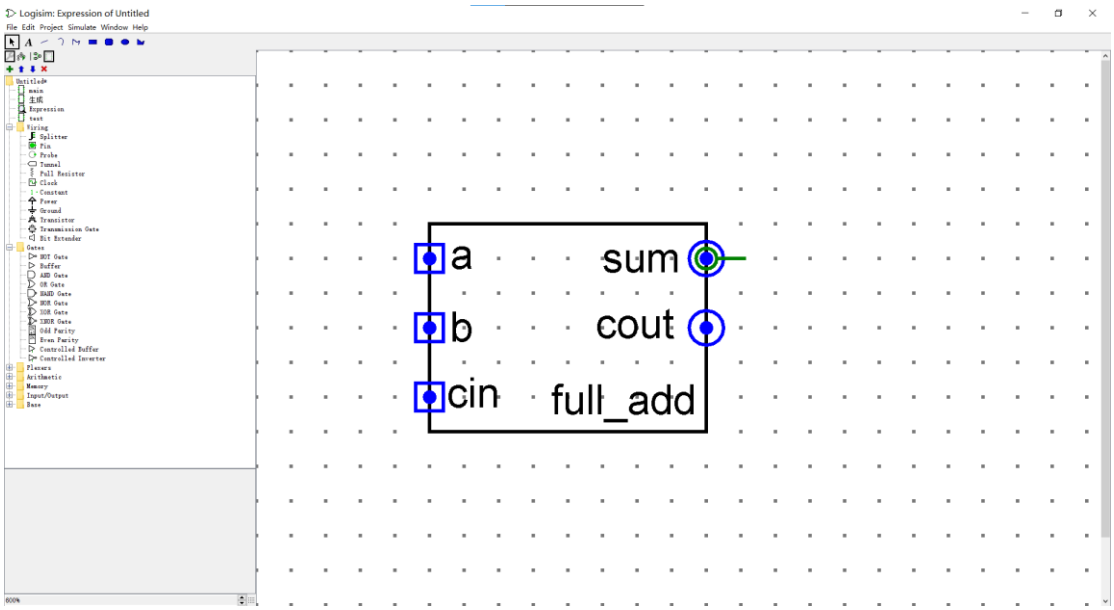
单 bit 信号拼接称为 2bit 信号。事实上它的效果等同于下列代码

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

例3. 编写全加器的 Verilog 代码

Step 1：首先绘制电路图






Step 2：编写对应 Verilog 代码

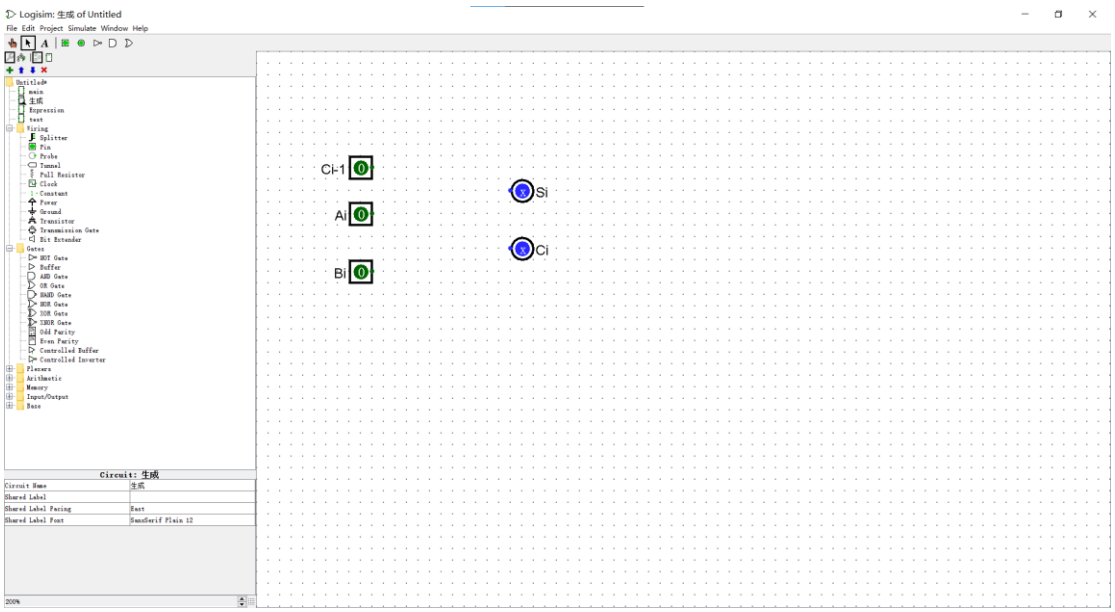
```
module full_add(
  input a, b, cin,
  output sum, cout);
  wire s, carry1, carry2;
  add add_inst1(
    .a (a),
    .b (b),
    .sum (s),
    .cout (carry1));
  add add_inst2(
    .a (a),
    .b (b),
    .sum (s),
    .cout (carry2));
  assign cout = carry1 | carry2;
endmodule
```

其中调用了半加器模块 add，则最终电路中就会出现两个半加器，行为特性完全一致，可以视作是封装后的电路直接引用。

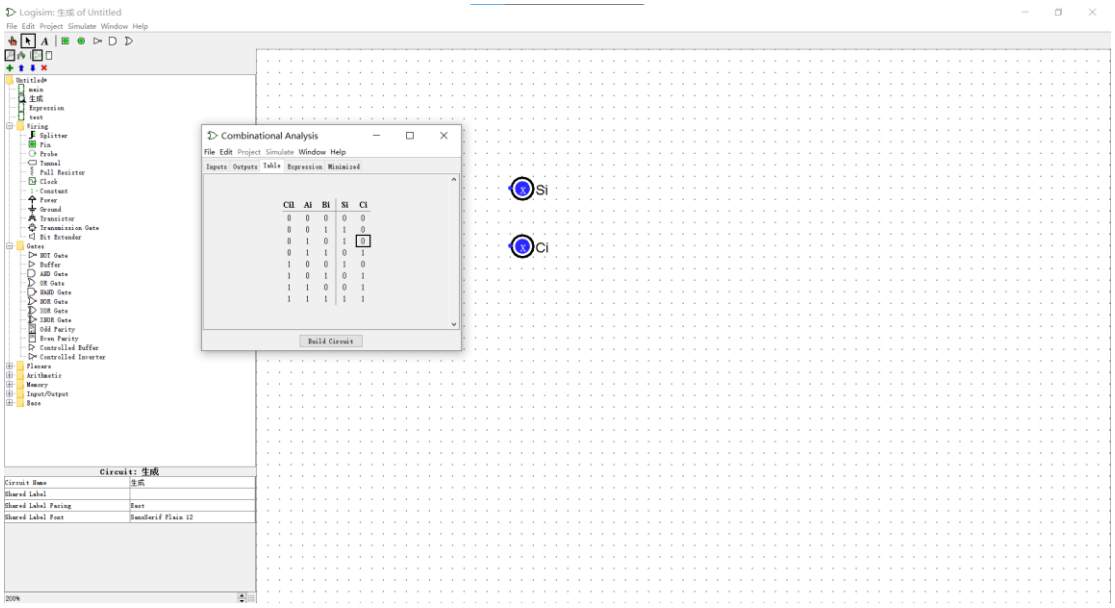
【实验练习】

题目 1：

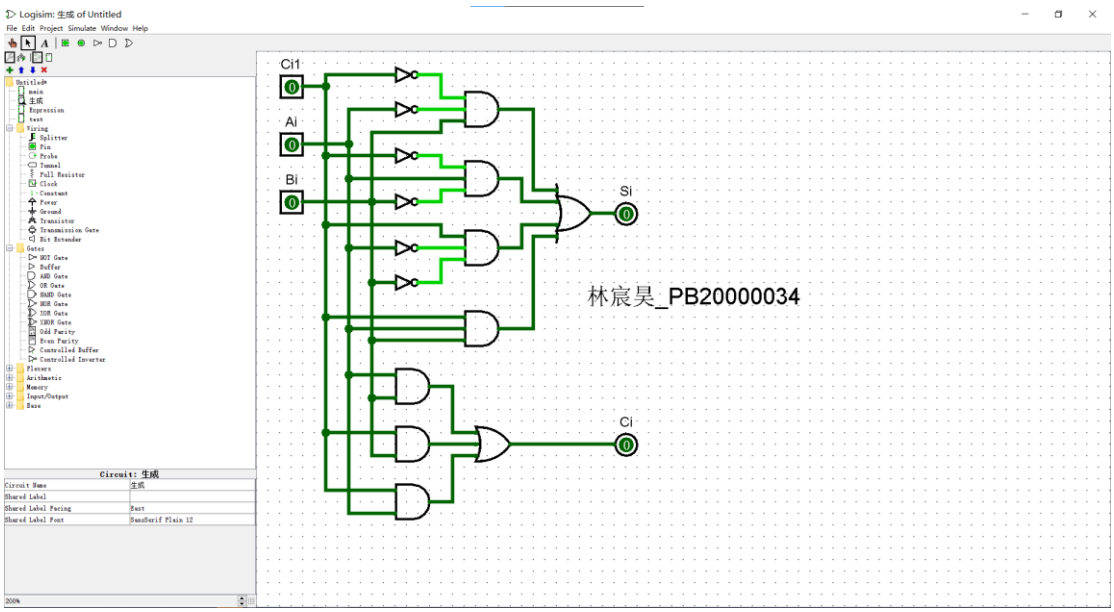
Step 1.1：设置对应引脚




Step1.2：修改真值表

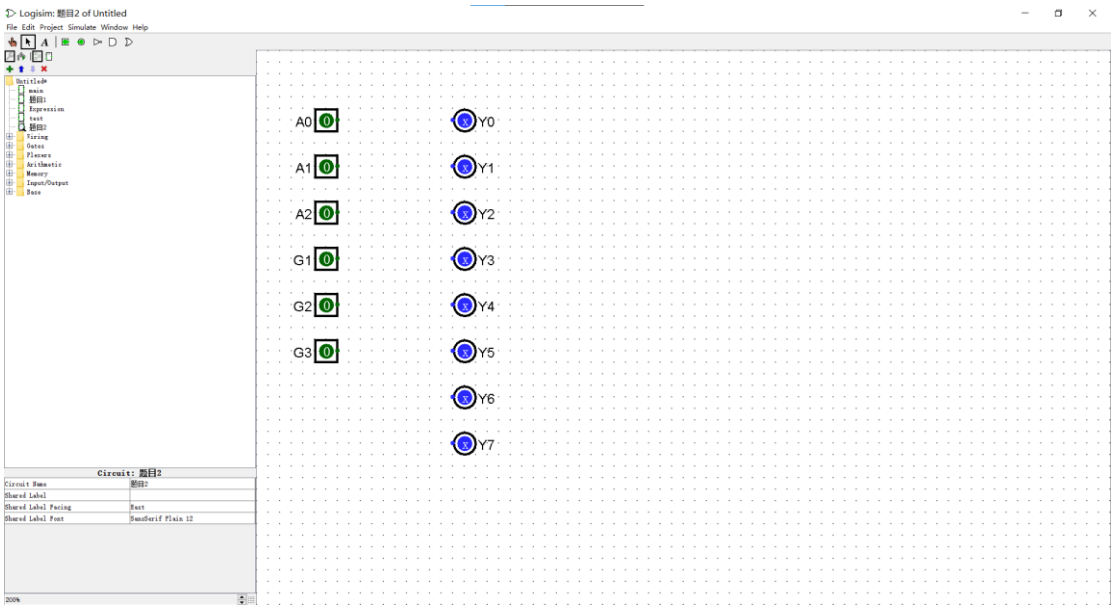



Step1.3：生成电路




题目 2：

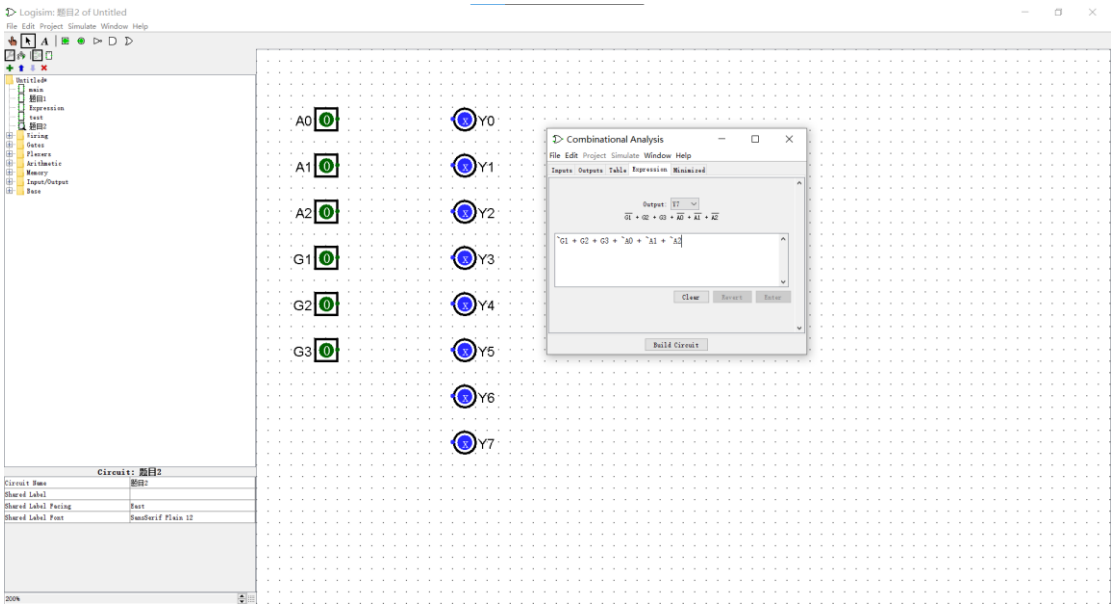
Step1：设置对应引脚

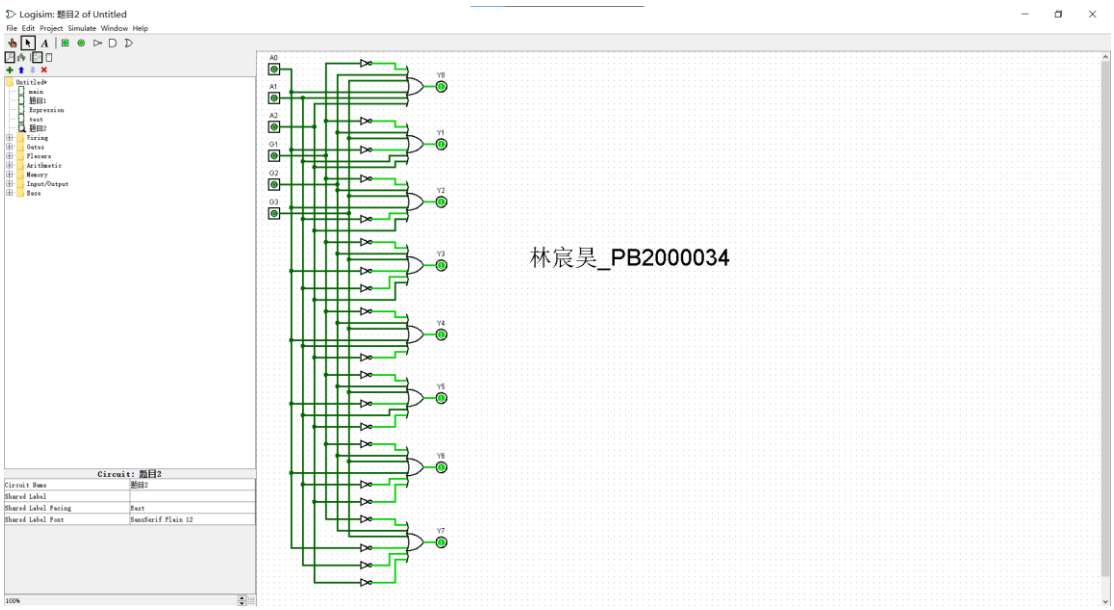



Step2：由真值表给出对应逻辑表达式

```
Y0 = ~G1 + G2 + G3 + A0 + A1 + A2
Y1 = ~G1 + G2 + G3 + ~A0 + A1 + A2
Y2 = ~G1 + G2 + G3 + A0 + ~A1 + A2
Y3 = ~G1 + G2 + G3 + ~A0 + ~A1 + A2
Y4 = ~G1 + G2 + G3 + A0 + A1 + ~A2
Y5 = ~G1 + G2 + G3 + ~A0 + A1 + ~A2
Y6 = ~G1 + G2 + G3 + A0 + ~A1 + ~A2
Y7 = ~G1 + G2 + G3 + ~A0 + ~A1 + ~A2
```

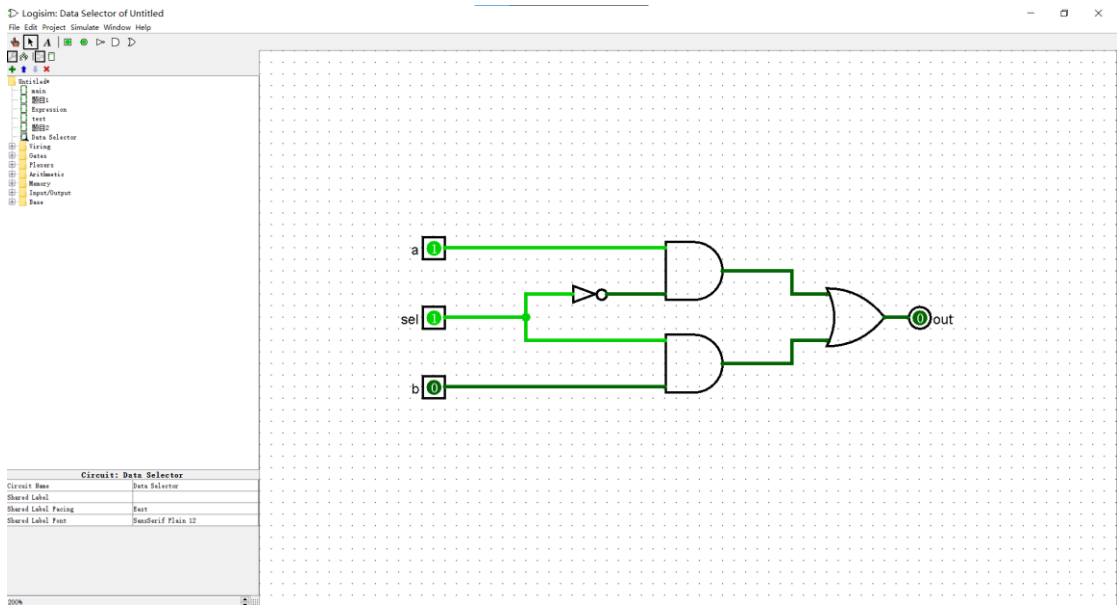
## Step3：输入“Expression”并生成电路



题目 3:

Step 1：绘制电路图

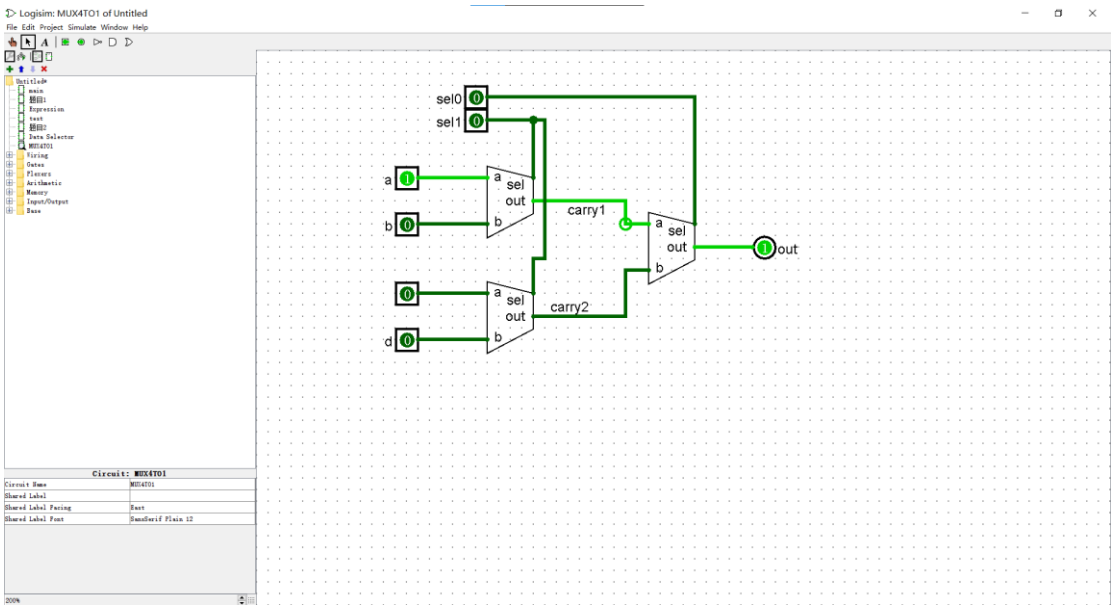



Step 2：编写 Verilog 代码

```verilog
module mux2to1(
  input a, b, sel,
  output out);
  assign out = (~sel & a) | (sel & b);
endmodule
```

题目 4:

Step 1：画出电路图




Step 2: 通过例化上题的 2 选 1 数据选择器编写 verilog 代码

```verilog
module mux4to1(
  input a, b, c, d, sel1, sel0,
  output out);
  wire carry1, carry2;
  mux2to1 mux1(a, b, sel1, carry1);
  mux2to1 mux2(c, d, sel1, carry2);
  mux2to1 mux3(carry1, carry2, sel0, out);
endmodule
```

题目 5:

Step 1: 根据真值表得到相应逻辑表达式

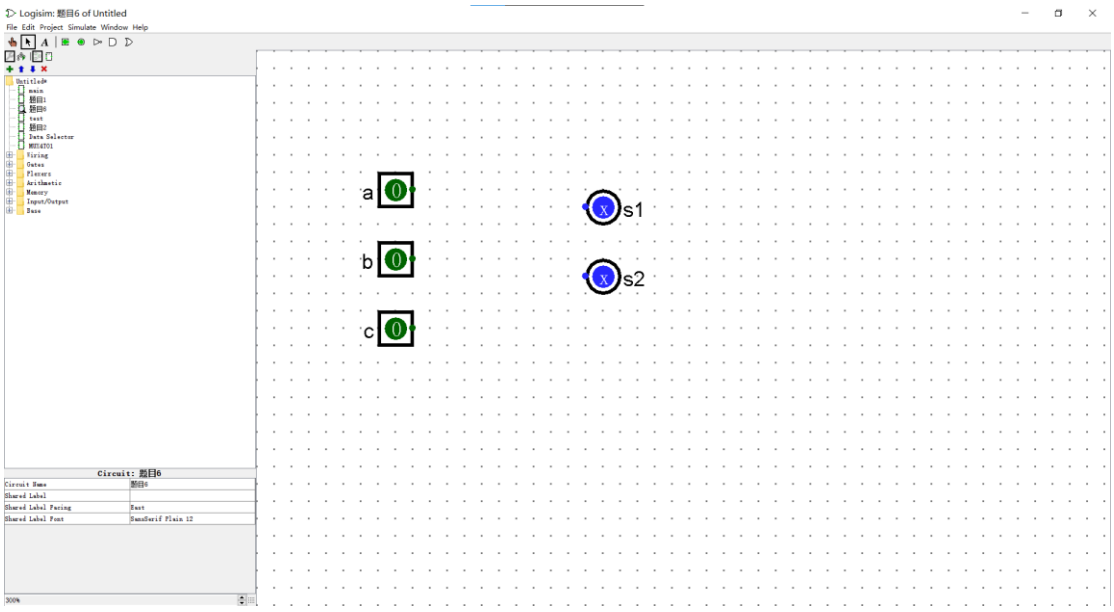
```
y0 = i7 + i5 ~i6 + i3 ~i4 ~i6 + i1 ~i2 ~i4 ~i6
y1 = i7 + i6 + i3 ~i4 ~i5 + i2 ~i4 ~i5
y2 = i7 + i6 + i5 + i4
```

Step 2：根据逻辑表达式编写 Verilog 代码

```
module (input i0, i1, i2, i3, i4, i5, i6, i7, output y0, y1, y2);
  assign y0 = i7 | i5 & ~i6 | i3 & ~i4 & ~i6 | i1 & ~i2 & ~i4 ~i6;
  assign y1 = i7 | i6 | i3 & ~i4 & ~i5 | i2 & ~i4 & ~i5;
  assign y2 = i7 | i6 | i5 | i4;
endmodule
```

题目 6:

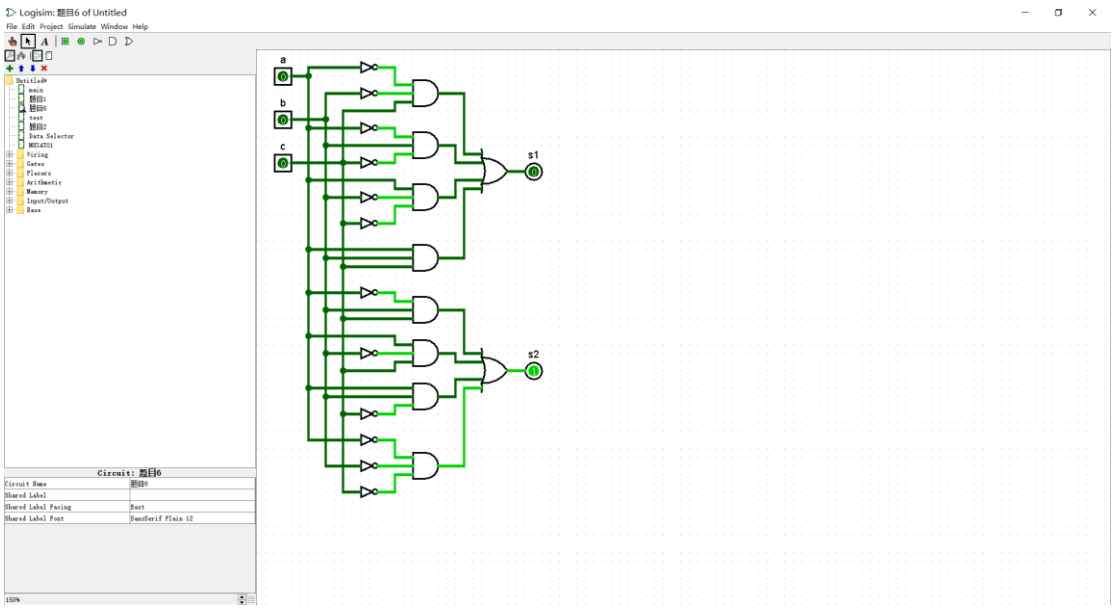
Step 1：根据模块输入输出设置引脚



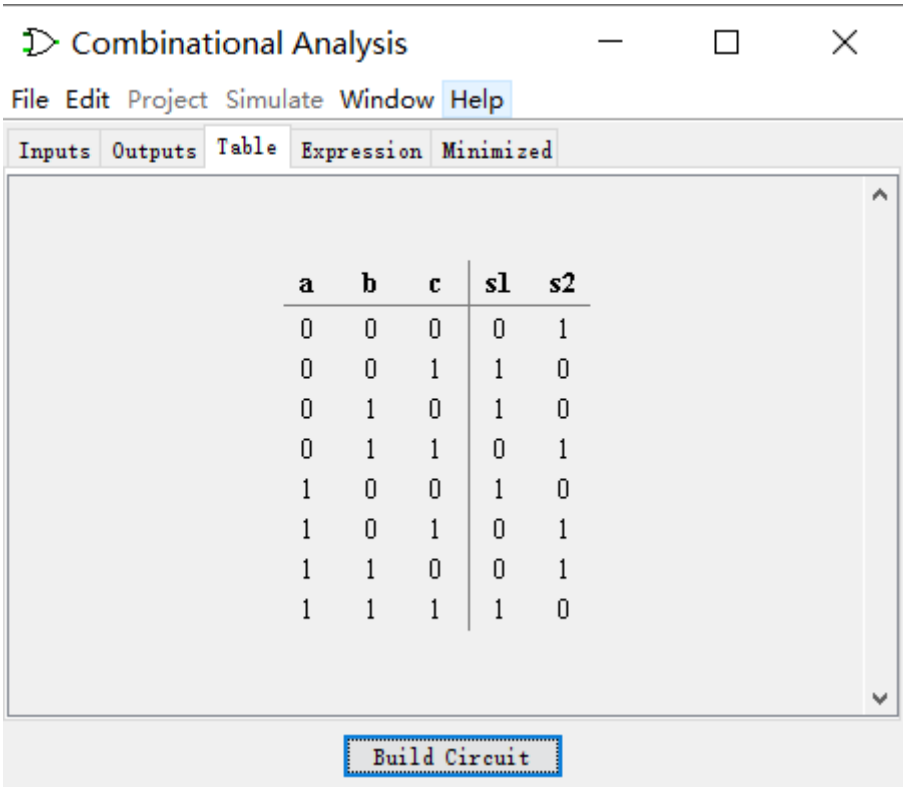

Step 2：根据 Verilog 代码给出对应逻辑表达式

```
s1 = ~a ~b c + ~a b ~c + a ~b ~c + a b c
s2 = ~a b c + a ~b c + a b ~c + ~a ~b ~c
```

Step 3：利用“Expression”得到相应电路图




Step 4：利用“table”得到对应真值表




| a | b | c | s1 | s2 |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 1 |
| 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 | 0 |

观察发现：

1. s1 s2 的值是互斥的，说明所代表的结果应该是互斥事件；
2. s1 s2 出现 1 的次数相同，可以猜想是否 2 者代表的结果是对称的，换言之，输入是对称的。

在此基础上可以猜想该电路用于计算输入中“1”的个数——若为奇数，s1 输出 1，s2 输出 0；若为偶数，s1 输出 0，s2 输出 1。由此实现计数功能。

【总结与思考】

1. 实验收获

其一，学会使用修改真值表以及输入表达式方式来生成逻辑电路，在输入输出并不多时能更快更方便的生成电路。同时能够根据已搭建的电路方便地获得真值表以及

相应表达式。除此之外，还能使用 minimize 功能便捷的优化逻辑表达式。

其二，学习了 verilog 基础语法以及相应的一些基础代码规范，能够根据表达式编写代码或者根据代码写出表达式；

其三，学会使用例化功能进行较复杂电路的代码编写。

2. 实验难易

   较易。尤其是学会工具的使用后免去很多手绘电路图的不便之处。

3. 实验任务量

   中等。虽然较易但是算上实验步骤在内任务量应该说不会太少，大概一个下午的水准（加上实验报告）。

4. 实验建议

   也许在 verilog 代码相关题中可以有这样的出法：“请用至少两种 verilog 代码写法实现对应逻辑电路”。正常情况下都是照抄电路可以直接得到相应的 verilog 表达式，但这种也是最省力最不费脑子的方式。也许通过这种硬性要求可以稍稍提升一些难度。

*吐槽：vivado 下了我一下午没下完，这校园网属实绷不住了。*