SI100B Spring 2026 EE Part

Tutorial 0 : Introduction to Logisim

@Shihan Wang
wangshh2023@shanghaitech.edu.cn
Slides thanks to @Quanyu Chen

Contents

Contents

1. Before Start ->
2. 1. How to install Logisim ->
3. 2. Getting Start with Logisim ->
4. 3. How to Design a digital circuit ->

Before Start

Before Start

在开始之前，您可能需要了解以下内容：

- Logisim是一个带有GUI界面的程序，请在您的本地环境中运行它。
- 请使用我们提供的Logisim版本为(4.1.0)以避免兼容性问题。Windows、Ubuntu和macOS系统都可以运行它。
- Logicsim不会在进行过程中保存您的工作。记得在开始时保存，并在工作时经常保存。
- `Ctrl + S`（或者`Command + S`） 快捷键允许快速保存。

1. How to install Logisim

1.1. Windows Enviroment Setup

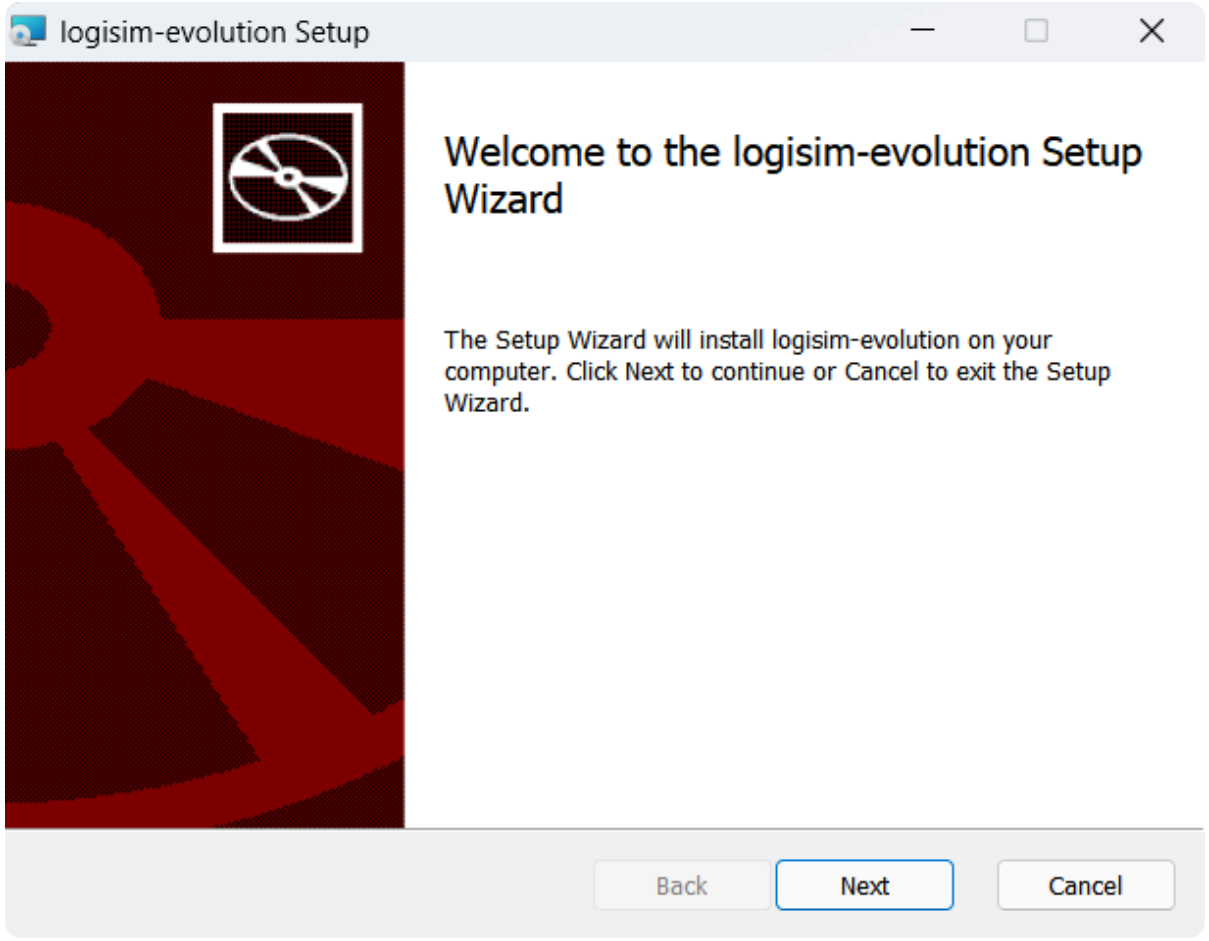
点击下载Logisim ->

如果无法打开链接，可以将下面的链接复制到浏览器中：

```
https://github.com/Heaticy/logisim-evolution/releases/download/v4.1.0/logisim-evolution-4.1.0-amd64.msi
```

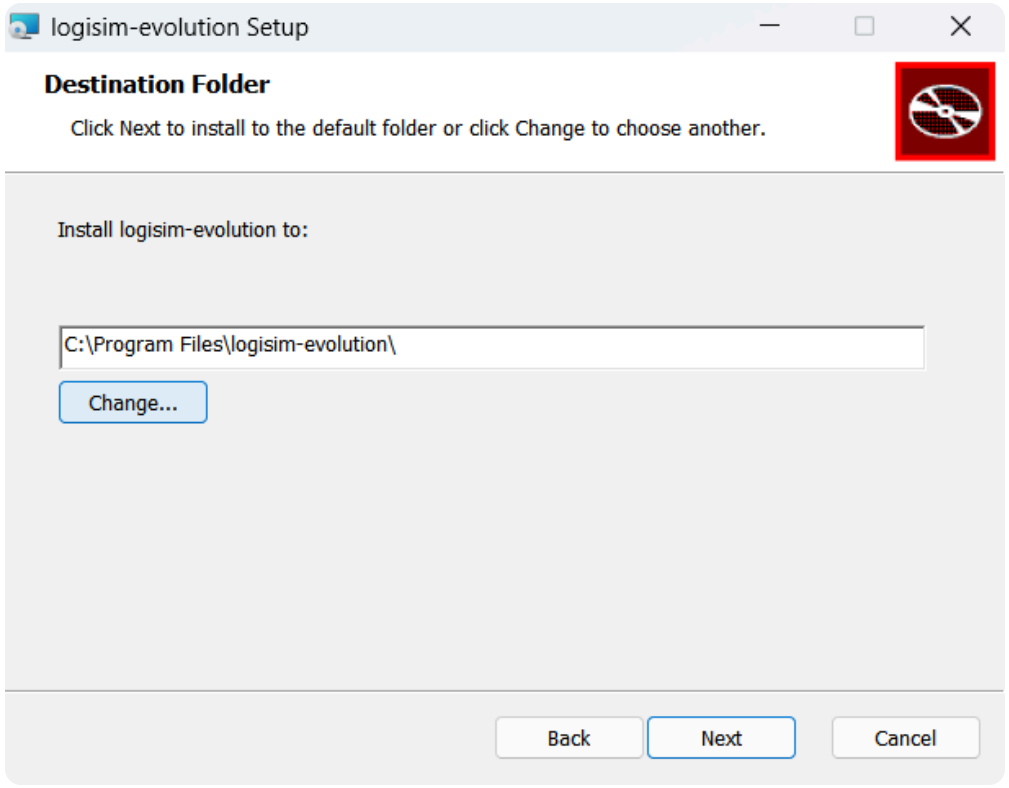
1.1. Windows Enviroment Setup

点击“next”:



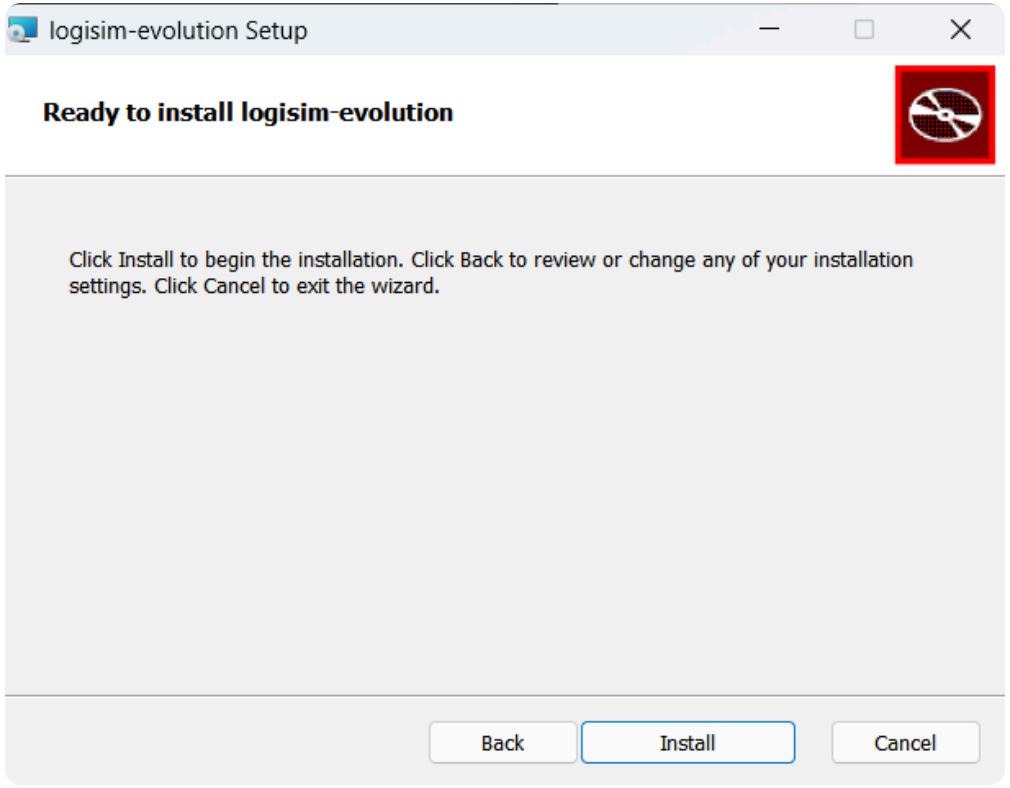
1.1. Windows Enviroment Setup

按需更改路径



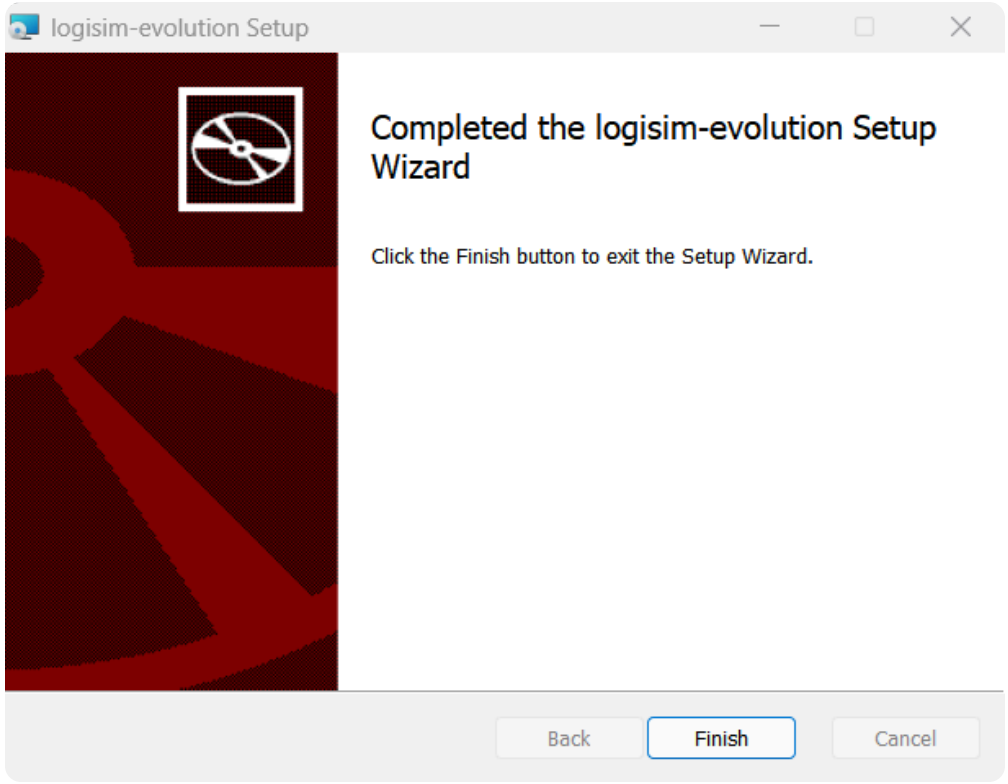
1.1. Windows Enviroment Setup

点击“Install”:



1.1. Windows Enviroment Setup

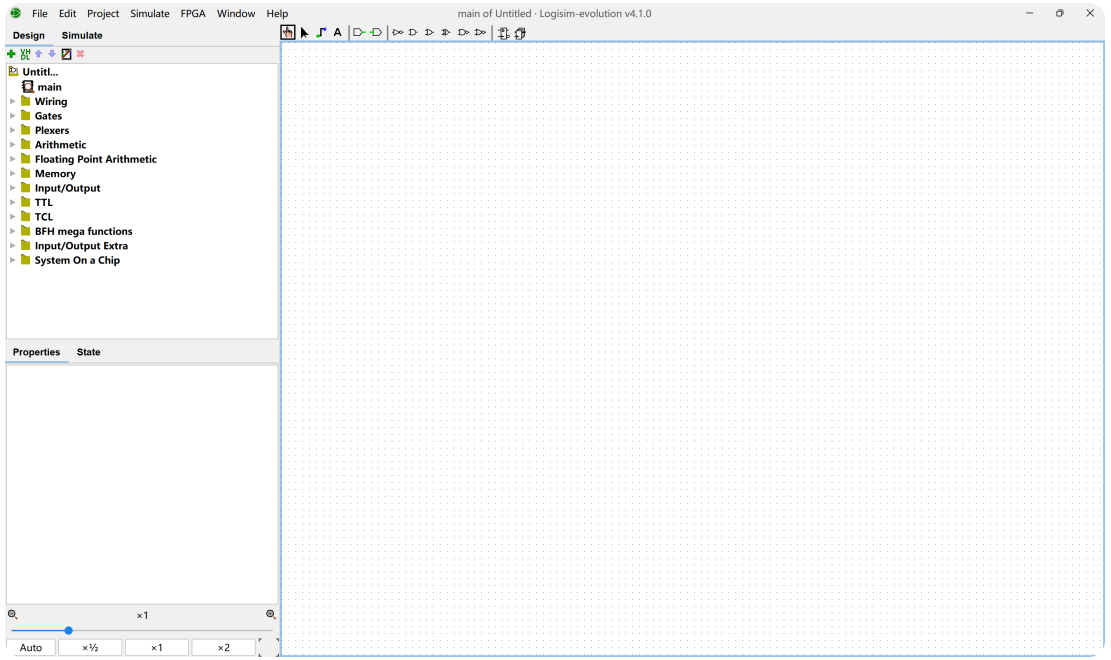
点击“Finish”，完成 Logisim 环境的安装：



1.1. Windows Enviroment Setup

双击打开桌面上  `logisim-evolution` 的图标，即可打开 Logisim 界面！

等待动画加载完成后，发现如下界面，则说明 logicsim 已经成功配置完毕！



1.2. MacOS Enviroment Setup

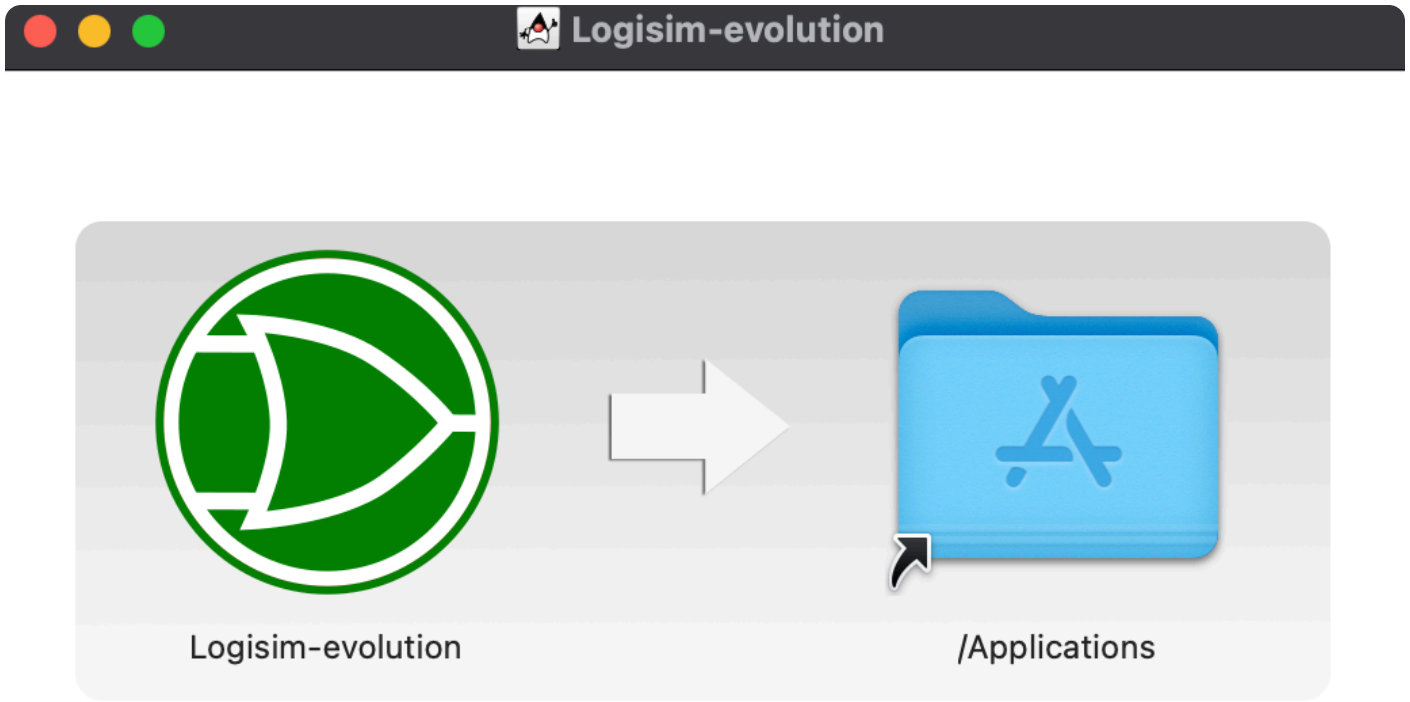
点击下载Logisim ->

如果无法打开链接，可以将下面的链接复制到浏览器中：

```
https://github.com/Heaticy/logisim-evolution/releases/download/v4.1.0/logisim-evolution-4.1.0-aarch64.pkg
```

1.2. MacOS Enviroment Setup

点击下载好的 `.dmg` 文件，弹出 MacOS 的安装界面。把 `logisim-evolution` 拖到 `Applications` 文件夹中：



1.2. MacOS Enviroment Setup

安装完成后，打开 `Applications` 文件夹，找到 `logisim-evolution` 的图标，点击打开，会发现报错：

☒ “无法打开 logisim-evolutoin"，因为“无法验证开发者”（或者类似报错）

此时，需要打开 `系统偏好设置`，找到 `安全性与隐私`，滑到底找到 `已阻止 Logisim-evolution 以保护 Mac` 的提示，点击 `仍要打开`：

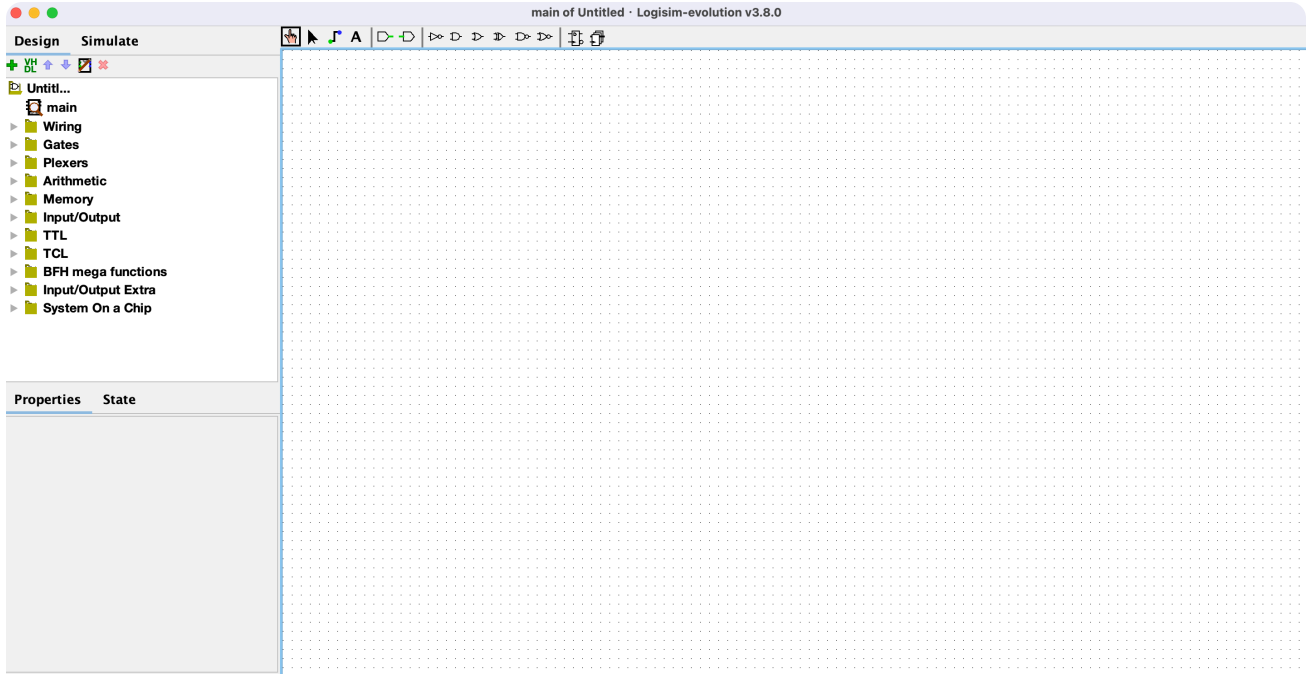


1.2. MacOS Enviroment Setup

此时需要**固执地**点击“仍要打开”（千万不要点击“移到废纸篓”！）

然后需要输入密码，输入密码之后安装完成。

点击应用即可打开界面！





2. Get Started With Logisim

2.1. Task 1: 创建AND门电路

我们将尝试创建一个 `AND` 门电路，以帮助我们学习如何使用 Logisim 。请注意程序左下角的缩放和网格功能，这将在我们构建大型电路设计中的时候更好的进行布线。

2.1. Task 1: 创建AND门电路

1. 首先单击 `AND` 门按钮。这将使 `AND` 门的影像跟随光标移动。在主原窗口中单击一下，就可以放置一个 `AND` 门。

2. 单击 `Input Pin` 按钮。现在，在 `AND` 门左侧的某个位置放置两个输入引脚。

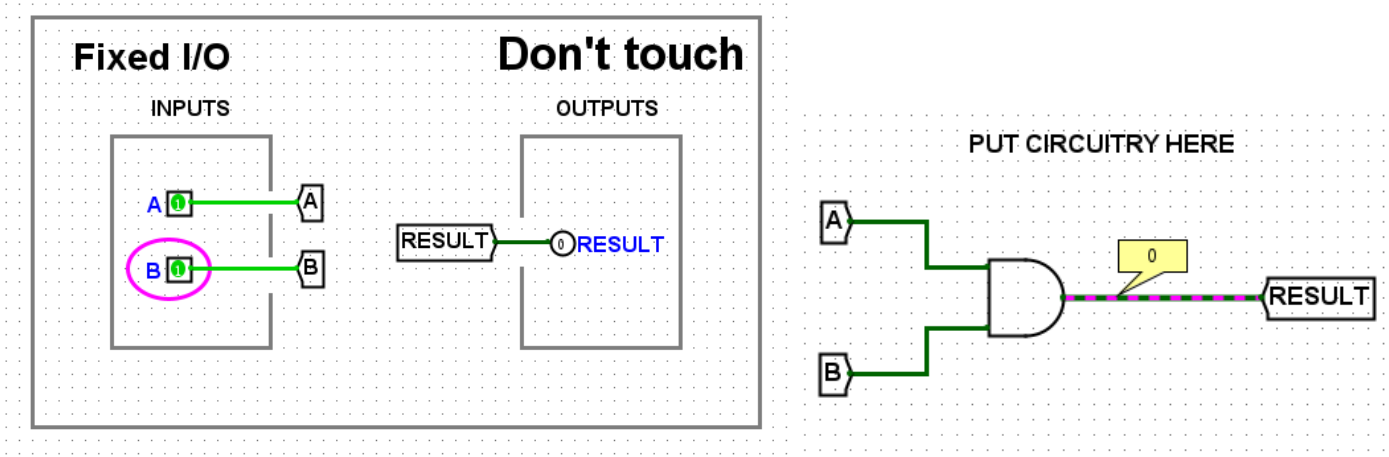
3. 单击 `Output Pin` 按钮。然后在 `AND` 门的右侧放置一个输出引脚。

2.1. Task 1: 创建AND门电路

4. 单击 `Select` 工具按钮。单击并拖动，将输入引脚连接到 `AND` 门的左侧。这需要几个步骤，因为**您只能绘制垂直和水平导线**。只需水平绘制导线，松开鼠标键，然后从导线末端开始单击并拖动，继续垂直绘制。您可以将导线连接到左侧 `AND` 门上的任意引脚。重复同样的步骤，将 `AND` 门（右侧）的输出连接到输出引脚。双击这三个引脚，为其命名标签。

2.1. Task 1: 创建AND门电路

5. 单击 `Poke` 工具并尝试单击 Schematic 中的输入引脚。观察会发生什么。



Properties	State
Pin "B"	
FPGA supported:	Supported
Facing	→ East
Output?	No
Data Bits	1
Three-state?	No
Pull Behavior	Unchanged
Label	B
Label Font	SansSerif Bold 16
Radix	Binary
Reset value:	0x0
Appearance	Classic Logisim

> poke流在lol英雄联盟里非常的流行，所谓poke就是利用英雄的技能优势,一般是射程远伤害高的技能，在和对方团战之前就打残对方的一种战术，简单的说一下poke的优劣势以及那些英雄可以poke。



2.2. Exercise 1: Pin

针脚是电路的输出还是输入，取决于它的输出值属性（即 Selection 中的 “Output?” ）。

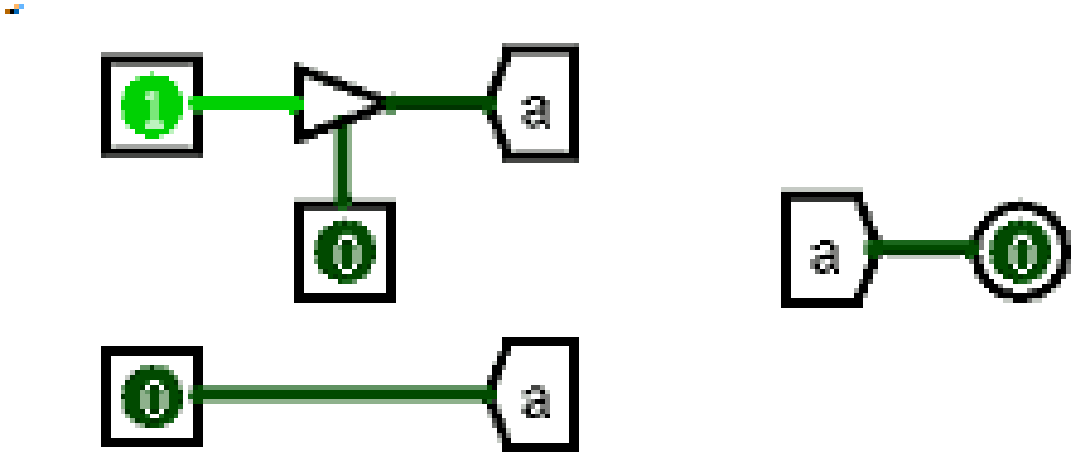
圆形或圆角矩形表示输出针脚，正方形或矩形表示输入针脚。

输入或输出值都会在组件上显示(打印视图除外)。

2.2. Exercise 2: Tunnel

标签通道的作用类似于导线，但与导线不同的是，连接不是明确绘制的。当你需要连接电路中相隔很远的点时，用标签通道来替代就很有用。下面的插图说明了它是如何工作的：这三个隧道都有相同的标签a，这三个隧道相当于连接的点。(如果其中一条隧道被标记为其他东西，比如b，那么它将是另一组隧道的一部分)

其主要参数是 “Label（标签）”，这是一个特别重要的属性，如果两个标签通道的标签名称一样，那么它们相当于之间有导线连接，是连通的。



2.2. Exercise 3: Constant

Constant 输出在 Value（值）属性中指定的值。它只有一个引脚，输出对应位宽属性的值。

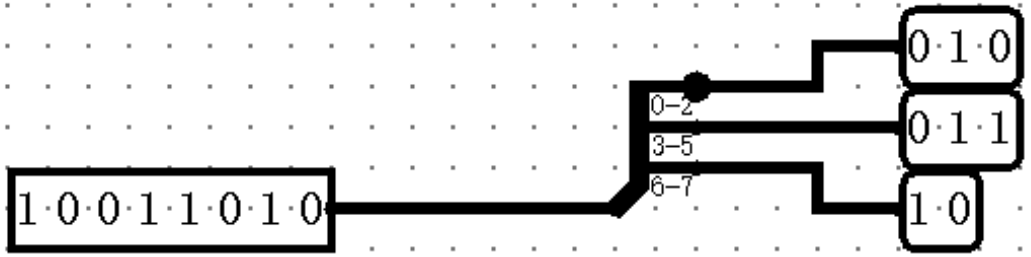
其属性包括:

- Data Bits（数据位宽）：指定输出数据的位宽，一个 n 位的常量有 2^n 个可能的值（范围为 $[0, 2^{n-1}]$ ）
- Value（值）：指定常量的值，注意 `0x` 表示的是十六进制

2.2. Exercise 4: Splitter

分线器可以把一个多位的输入拆分为若干位，也可反过来把若干个若干位的输入合并为一个输出，可设置具体拆分方式。
其参数包括:

- Facing（朝向）：控制元件的朝向
- Fan Out（分流端口数）：设置输出端口的数量
- Bit Width In（输入位宽）：设置输入位宽
- Bit X （第X位）：设置第X位数据在哪个口输出



Selection: Splitter	
Facing	East
Fan Out	3
Bit Width In	8
Appearance	Left-handed
Bit 0	0 (Top)
Bit 1	0 (Top)
Bit 2	0 (Top)
Bit 3	1
Bit 4	1
Bit 5	1
Bit 6	2 (Bottom)
Bit 7	2 (Bottom)

CSDN @Hi_KER

2.2. Exercise 5: 构建子电路

正如 Python 程序可以包含函数一样， Schematic 也可以包含子电路。在这部分实验中，我们将创建几一些子电路来演示它们的用途。

☒ 重要提示

Logisim Evolution 规定，不能用关键字（如 `NAND` ）命名子电路，也不能在名称的第一个字符使用数字。

2.2. Exercise 5: 构建子电路

请按照以下步骤操作

1. 打开练习 1 的 Schematic (`File -> Open -> ex1.circ`)。

2. 双击左侧菜单电路选择器中的 `NAND1` 打开 `NAND1` 空子电路。（注意末尾的 `1` ； 因为有一个名为 `NAND` 的组件，所以我们不能将其称为 `NAND` ）。

2.2. Exercise 5: 构建子电路

3. 在新的 Schematic 窗口中，创建一个简单的 `NAND` 电路，左侧为 2 个输入引脚，右侧为输出引脚。**请不要使用 Gates 文件夹中的内置 `NAND` 门**（即只使用选择工具图标旁边提供的 `AND` 、 `OR` 和 `NOT` 门）。您可以使用选择工具选择输入/输出，并更改窗口左下角的属性 `Label` 来更改输入和输出的标签。

4. 重复上述步骤，再创建几个子电路：

   - `NOR`
   - `XOR`
   - `2-to-1 MUX`
   - `4-to-1 MUX`

2.2. Exercise 5: 构建子电路

以4-1选择器为例：总计六位输入，分别是**四路数据输入A、B、C、D**和**两位地址选择sel1、sel2**。我们可以朴素地感知，这个电路通过地址选择将一路输入接通至输出。

由此可知运行逻辑：
两位地址选择，采用二进制编码，将被译码为**四位输出（0001或0010或0100或1000）**。通过这四位输出分别和数据输入取逻辑与（AND）并输出，仅有一位数据输入被带出，另外三位输出必为0。最终将取逻辑与之后的四位输出通过四输入或门输出即可。

可推知输出 `Y=Asel1'sel2'+Bsel1'sel2+Csel1sel2'+Dsel1sel2`

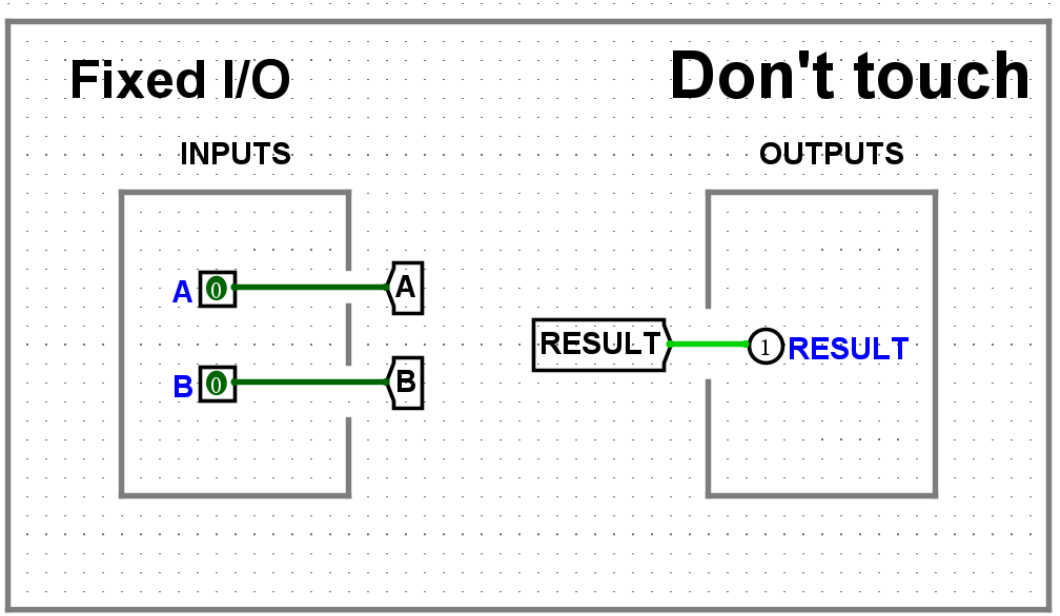
2.2. Exercise 5: 构建子电路

1. 注意事项
   - 请不要更改子电路的名称或创建新的子电路
   - 请在**分别命名**的电路中工作，否则自动跟踪器将无法正常工作。
   - 请勿使用除 `AND` 、 `OR` 和 `NOT` 以外的任何内置门电路。
   - 建立子电路后，您可以（而且我们鼓励您）用它来建立其他电路。您可以像放置其他元件一样，点击并放置您创建的子电路。

2.2. Exercise 5: 构建子电路

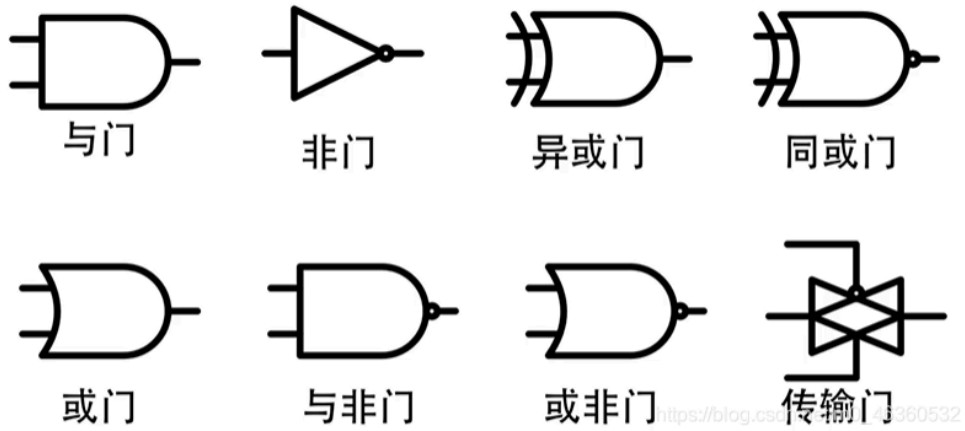
在作业中，我们有一些标注了 Don't Touch 的电路，这部分是固定位置的输入输出，用于进行作业的结果检测。

- 你可以**复制**它们，但请不要以任何形式**移动**它们。
- 如果因为移动 IO 部分导致测试不通过，TA 有权利不进行补测事宜。
  - 当然，如果你及时发现了问题，可以选择重新下载模板文件。



Appendix

异或 XOR; 同或 XNOR; 与非 NAND; 或非 NOR; 传输门 TG.



使用LaTeX能够轻松打出异或符号和同或符号：在线 LaTeX 公式编辑器 ->

- XOR → `\oplus` → $\oplus$
- XNOR → `\odot` → $\odot$

4、 与非门：所有输入为高时，才会有输出低。逻辑函数表示为F=(A*B)'。

输入A	输入B	输出Y
0	0	1
0	1	1
1	0	1
1	1	0

5、 或非门：所有输入为低时，才会有输出高。逻辑函数表示为F=(A + B)'。

输入A	输入B	输出Y
0	0	1
0	1	0
1	0	0
1	1	0

6、 异或门：输入相同时输出为低，否则为高。逻辑函数表示为F=A'*B* + *A*B'。

输入A	输入B	输出Y
0	0	0
0	1	1
1	0	1
1	1	0

7、 同或门：与异或门相反。输入相同时输出为高，否则为低。逻辑函数表示为F= A*B + A'*B'。

输入A	输入B	输出Y
0	0	1
0	1	0
1	0	0
1	1	1

3. How to Design a digital circuit

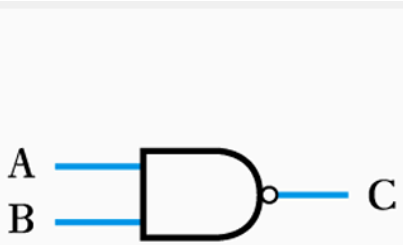
3. How to design a digital circuit

组合逻辑数字电路是电子工程中的基本概念之一，它在数字系统设计中扮演着极其重要的角色。本部分将从基础知识讲解到具体实施步骤，一步一步带你学习如何基于真值表绘制组合逻辑数字电路。

3.1. 理解真值表

3.1.1 理解真值表：定义

真值表是组合逻辑电路设计的起点，它罗列了所有可能的输入组合及其对应的输出结果。



A	B	C
0	0	1
0	1	1
1	0	1
1	1	0

$$C = \overline{A \cdot B}$$

(a) Circuit symbol (b) Truth table (c) Boolean expression

- (a) Circuit Symbol: 电路符号，使用表示门电路的图形符号来表示逻辑操作。
- (b) Truth Table: 真值表，列出了所有可能的输入组合及其对应的输出结果。
- (c) Boolean Expression: 布尔表达式，使用逻辑运算符号来表示逻辑操作。

3.1. 理解真值表

3.1.2. 理解真值表-以全加器为例

以一个三输入的全加器为例，全加器的作用是将两个 1 位（1-Bit）的二进制数相加，并考虑到可能的进位。

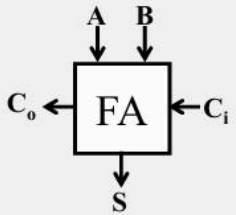
- 输入：设输入为A、B、C
  - A 和 B 为加数
  - Cin 为进位
- 输出：设输出为 S（和，Sum）和 Cout（新的进位）

> 关于 Full Adder 更详细的讲解，你可以在 Homework 1 的题目中看到！

3.1. 理解真值表

3.1.2. 理解真值表-以全加器为例

真值表列出所有输入变量的可能组合。
对于 n 个输入变量，真值表包含 2^n 行，每一行输入唯一，计算对应的输出值。



A	B	C_i	C_o	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

$$
\begin{aligned}
S &= A\cdot B\cdot C_i + A\cdot \overline{B}\cdot \overline{C_i} + \overline{A}\cdot B\cdot \overline{C_i} + \overline{A}\cdot \overline{B}\cdot C_i \\
&= (A\cdot B + \overline{A}\cdot \overline{B})\cdot C_i + (A\cdot \overline{B} + \overline{A}\cdot B)\cdot \overline{C_i} \\
&= \overline{A\oplus B}\cdot C_i + A\oplus B\cdot \overline{C_i} \\
&= A\oplus B\oplus C_i
\end{aligned}
$$

$$
C_o = A\cdot B + A\cdot \overline{B}\cdot C_i + \overline{A}\cdot B\cdot C_i = A\cdot B + (A\cdot \overline{B} + \overline{A}\cdot B)\cdot C_i = A\cdot B + (A\oplus B)\cdot C_i
$$

3.2. SOP（Sum Of Product）

SOP形式是用“与”（Product，或者称 And）项（称为 Product Terms）的“相加”（Add，或者称 Or）来表达逻辑函数，这些 Product Term 中的每个都能导致逻辑函数输出为 1 。

3.2. SOP（Sum Of Product）

3.2.1. Standard SOP（最小项）

标准SOP形式中的每个product term都应该包含逻辑函数中的所有变量，无论是原始变量还是其反转形式。

例如，对于一个三输入函数 $f(A,B,C)$，一个标准SOP项可能是 $A\overline{B}C$ 表示当 A 为 1、B 为 0、C 为 1 时函数输出为 1。

3.2. SOP（Sum Of Product）

A	B	C_i	C_o	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

3.2.2. 通过真值表得到 Standard SOP

仍以全加器为例，观察真值表中 Sum(S) 和 Carry(C_o) 为1的行，可以将这些行对应的输入组合转换成逻辑表达式中的 Product terms。

- 对于输出 S，真值表中 $S = 1$ 的行对应的表达式为：

$$S = \overline{A}\,\overline{B}C + \overline{A}B\overline{C} + A\overline{B}\,\overline{C} + ABC$$

- 对于输出 Cout（新的进位），$C_o = 1$ 的行对应的表达式可以分为：

$$C_o = \overline{A}BC + A\overline{B}C + AB\overline{C} + ABC$$

3.3. 布尔代数与逻辑函数化简

布尔代数提供了一系列规则，用于逻辑函数的化简，使之可以用更少的逻辑门电路实现相同的功能。

3.3.1.常见布尔代数规则

Basic rules of Boolean algebra.

1. $A + 0 = A$	**7.** $A \cdot A = A$
2. $A + 1 = 1$	**8.** $A \cdot \overline{A} = 0$
3. $A \cdot 0 = 0$	**9.** $\overline{\overline{A}} = A$
4. $A \cdot 1 = A$	**10.** $A + AB = A$
5. $A + A = A$	**11.** $A + \overline{A}B = A + B$ *
6. $A + \overline{A} = 1$	**12.** $(A + B)(A + C) = A + BC$ *

A, B, or C can represent a single variable or a combination of variables.

3.3. 布尔代数与逻辑函数化简

3.3.2 使用布尔代数化简SOP

虽然布尔代数可以用于化简逻辑函数，但是这种方法的局限性在于，可能无法找到逻辑函数的最简形式。尤其是对于包含多个变量的函数，手动化简可能既复杂又容易出错。

3.4. 利用卡诺图化简

卡诺图是真值表的图形化表示。一个逻辑函数的卡诺图就是**将此函数的最小项表达式中的各项最小项相应地填入一个特定的方格图内**，此方格图为卡诺图。

L \ CD / AB	00	01	11	10
00	0	1	3	2
01	4	5	7	6
11	12	13	15	14
10	8	9	11	10

图 2.4.6　图 2.4.5 的简化表示法

L \ CD / AB	00	01	11	10
00	0	1	1	1
01	1	1	1	0
11	1	0	0	1
10	1	1	1	0

图 2.4.8　例 2.4.1 的卡诺图

左图为最小项的序号排列，右图为表达式对应的最小项

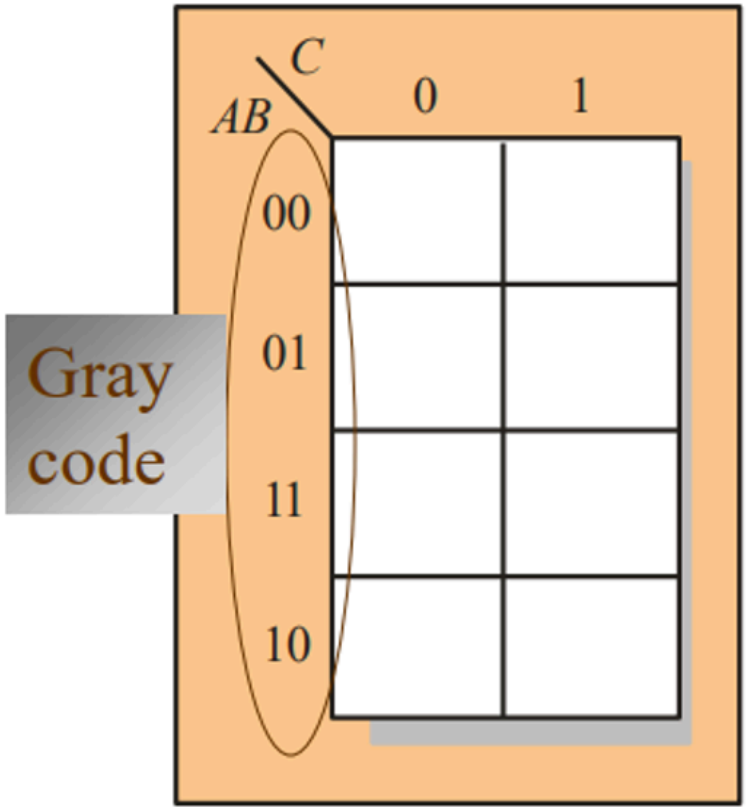
3.4. 卡诺图

3.4.1. 利用卡诺图化简

为什么两侧的数值不采用二进制数从小到大进行排列（事实上，这种排列形式叫“格雷码”）？

- 为了保证图中几何位置相邻的最小项在逻辑上也具有相邻性，两个相邻的最小项仅有一个变量是不同的。

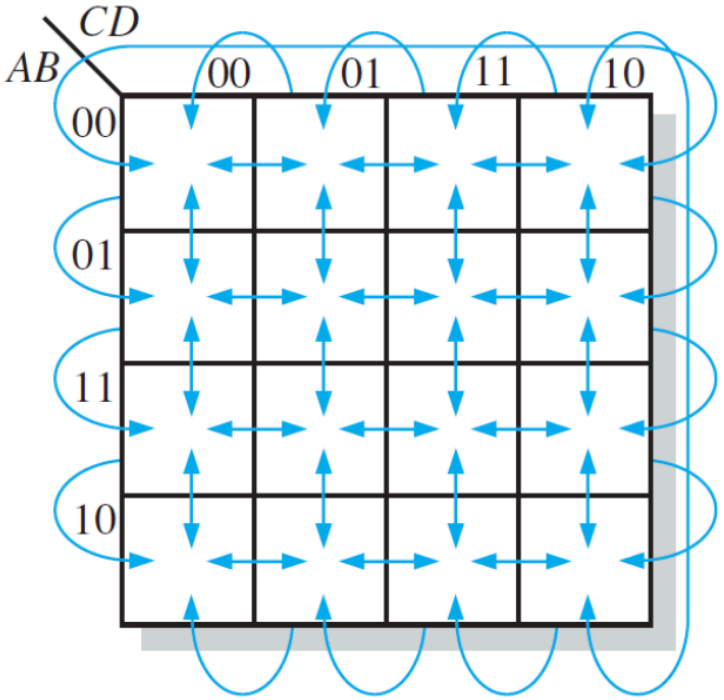
局限性：只能化简小于等于四个变量，（五个变量也可以，但一般不用）



3.4. 卡诺图

3.4.2 卡诺图化简规则

- 卡诺图的边缘是相连的，即：最左边的列与最右边的列相邻，最上面的行与最下面的行相邻。
- 将所有输出为1的方格圈起来，每个圈应该包含 2^N 个方格。
- 尽量使用大的圈来减少表达式中变量的数量。
- 确保卡诺图中的每个1至少被一个圈覆盖。
- 圈的数量应该尽可能少，这样可以确保最终的表达式尽可能简单。



3.4. 卡诺图

3.4.2 卡诺图化简规则：例子

这是一些不同大小的卡诺图。

- 我们要尽量把相邻的“1”合并
- 注意，跨过边缘的“1”们也是相邻的!

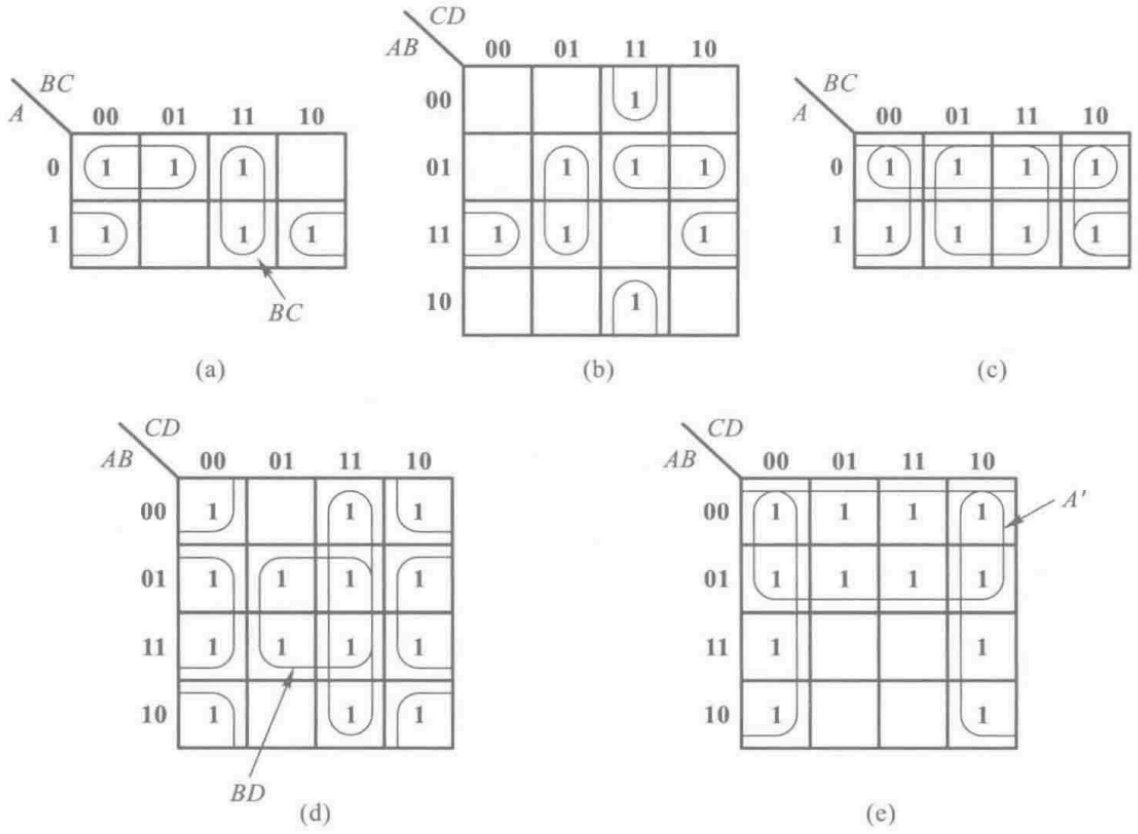


图 2.6.4　最小项相邻的几种情况

(a)、(b) 两个最小项相邻　(c)、(d) 四个最小项相邻　(e) 八个最小项相邻

3.5. 绘制组合逻辑电路

基于化简后的逻辑表达式，可以选择合适的逻辑门来实现相应的逻辑功能。

- **与门**（AND Gate）用于实现逻辑乘（AND）操作。
- **或门**（OR Gate）用于实现逻辑加（OR）操作。
- **非门**（NOT Gate）用于实现逻辑反转（NOT）操作。

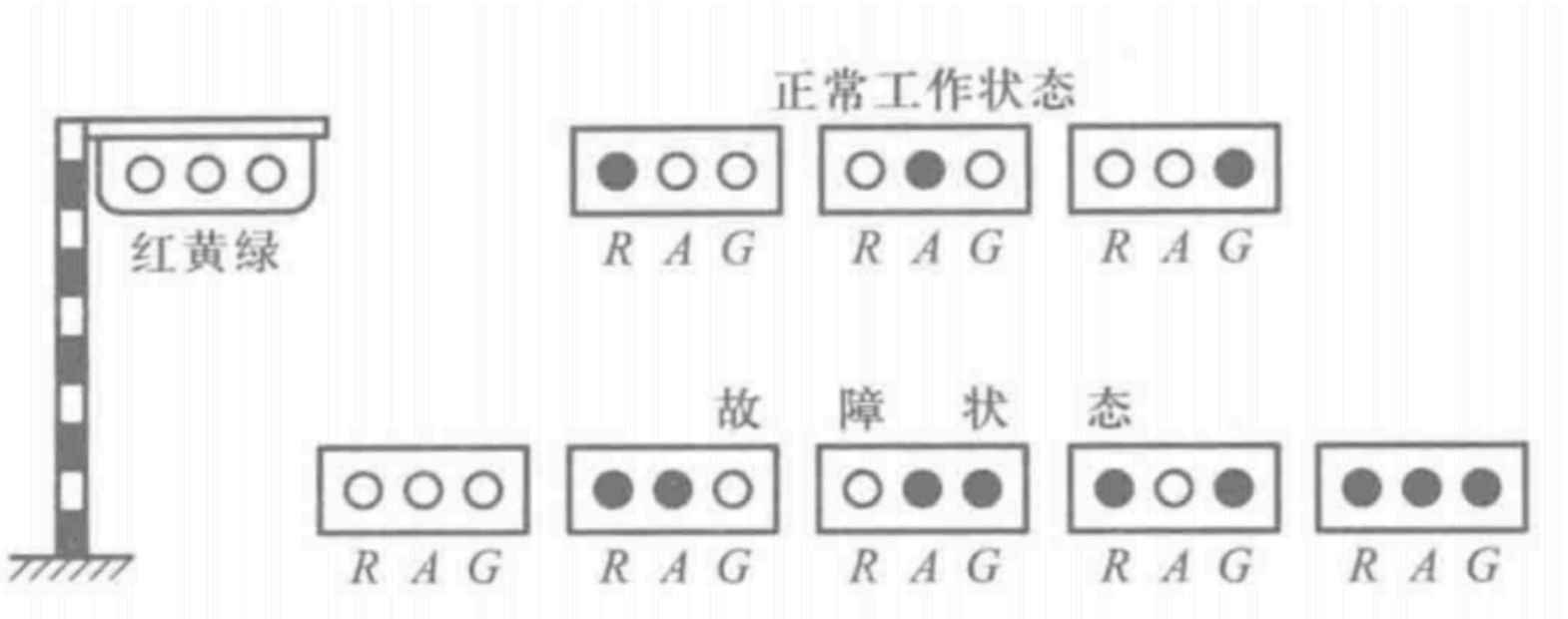
例如，对于前述全加器中S的逻辑表达式$\bar{A}\bar{B}C + \bar{A}B\bar{C} + A\bar{B}\bar{C} + ABC$，可以通过以下逻辑门来实现：

1. 使用四个非门分别处理所有变量的反转。
2. 使用与门处理每个 Product term。
3. 最后使用或门将所有 Product terms 的结果合并输出。

3.6. Example

根据上面所学到的步骤，尝试搭建一个红绿灯故障检测的电路：

使用逻辑门电路设计一个监视交通信号灯工作状态的逻辑电路。每一组信号灯均由红、黄、绿三盏灯组成，如下图所示。正常工作情况下，任何时刻必有一盏灯点亮，而且只允许有一盏灯点亮。而当出现其他五种点亮状态时，电路发生故障，这时要求发出故障信号，以提醒维护人员前去修理。



3.6. Example

Step 0：进行逻辑抽象

取红、黄、绿三盏灯的状态为输入变量，分别用 **R**、**A**、**G** 表示，并规定灯亮时为 1，不亮时为 0。

取故障信号为输出变量，以 **Z** 表示之，并规定正常工作状态下 **Z** 为 0，发生故障时 **Z** 为 1。

3.6. Example

Step 1：画真值表

R	A	G	Z
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

Step 2：最小项

找 `Z=1` 时对应的 R、A、G 的值

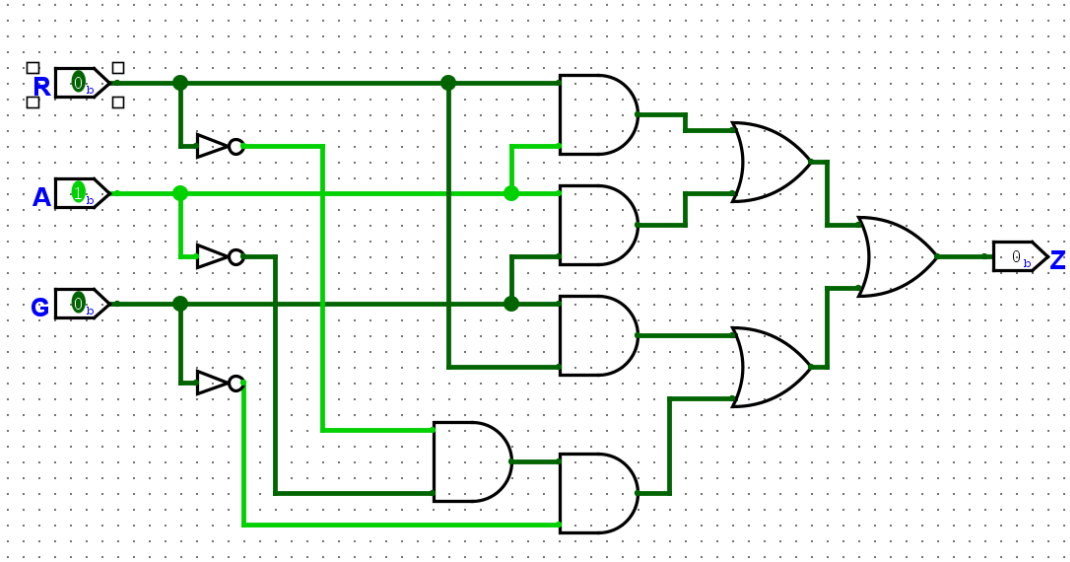
$$Z = R'A'G' + R'AG + RA'G' + RAG' + RAG$$

3.6. Example

Step 3：利用卡诺图进行化简

R \ AG	00	01	11	10
0	1	0	1	0
1	0	1	1	1

Step 4：画电路图



化简后的逻辑表达式为：

$$Z = R'A'G' + RA + RG + AG$$

Thanks!