SI100B Fall 2025 EE Part

Tutorial 1: Homework 1 Guide

@Quanyu Chen
chenqy2022@shanghaitech.edu.cn

写在前面

- 在 piazza 上提问的时候不要直接贴电路图片
- 认真阅读作业文档
- 善用纸笔，设计较复杂电路时一定先整理好思路再动手
- 学会正确的判断并描述自己的问题
- 模块化思想

Contents

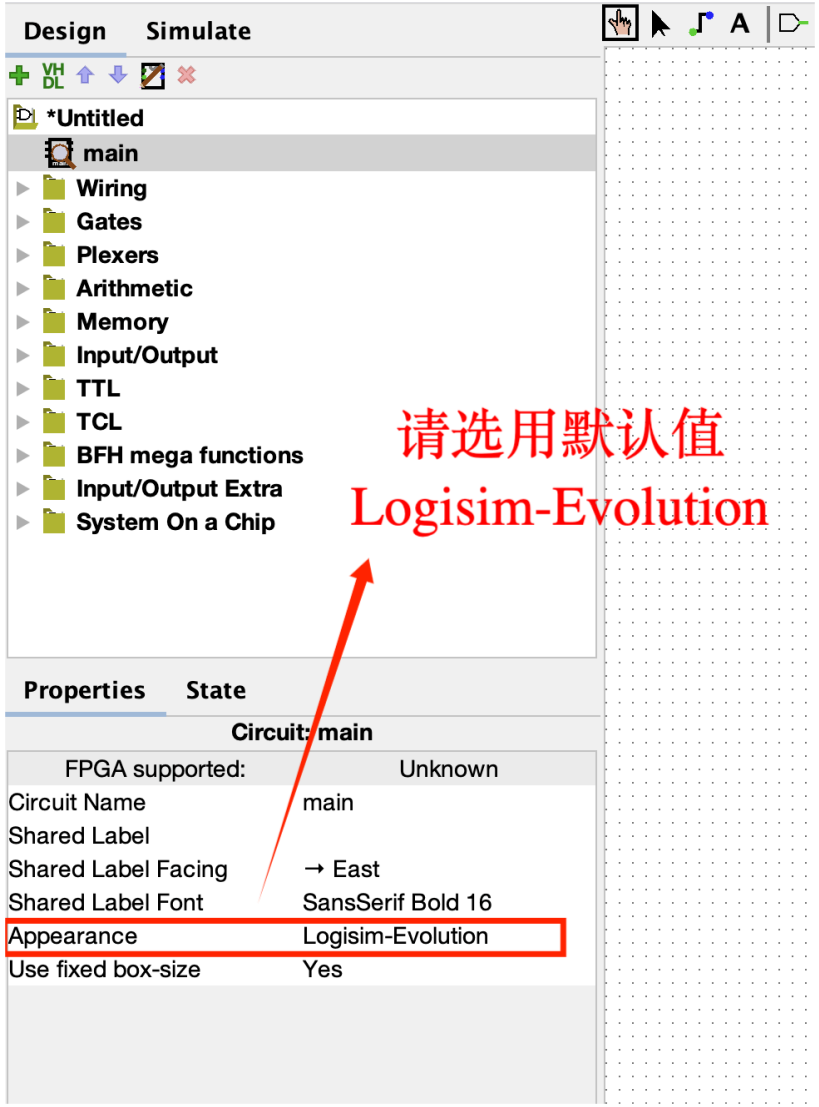
1. Logisim Tips ->
2. 元件的规范使用 ->
3. Exercise 1.1: Half-Adder ->
4. Exercise 1.2: Full-Adder ->
5. Exercise 2.1: 4-Bit Full Adder ->
6. Exercise 2.2: Subtracter ->
7. Exercise 3: 七段数码管的解码器 ->

Homework 1 Guide

Homework 1 Guide

Don't Modify the Appearance of the Circuits!!!

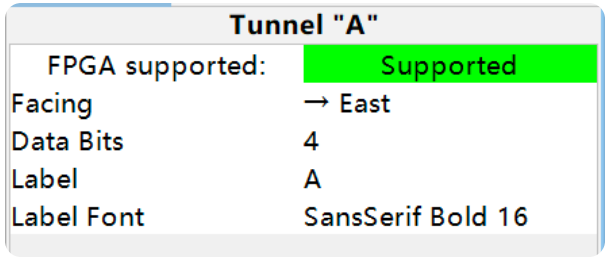
- **随意修改电路的外观可能导致测试不通过**!
- 虽然不是一定，但请不要以任何形式修改外观（以及任何看起来像是 Appearance 的东西）。
- 这个问题并不是很常见，因为我们没有任何关于电路外观的要求，一般都是直接使用默认的外观。



Homework 1 Guide

关于『红线』、『橙线』、『蓝线』

1. 红线说明连接出现**致命错误**。请不要抱有任何侥幸，仔细检查电路。
2. 橙线通常说明**位宽不匹配**，橙线通常会在两端显示每一端的位宽。
   - 位宽是什么？在每一个Tunnel或者Pin原件你都会在properties栏看到Data Bits这一项，这就是位宽。举例：

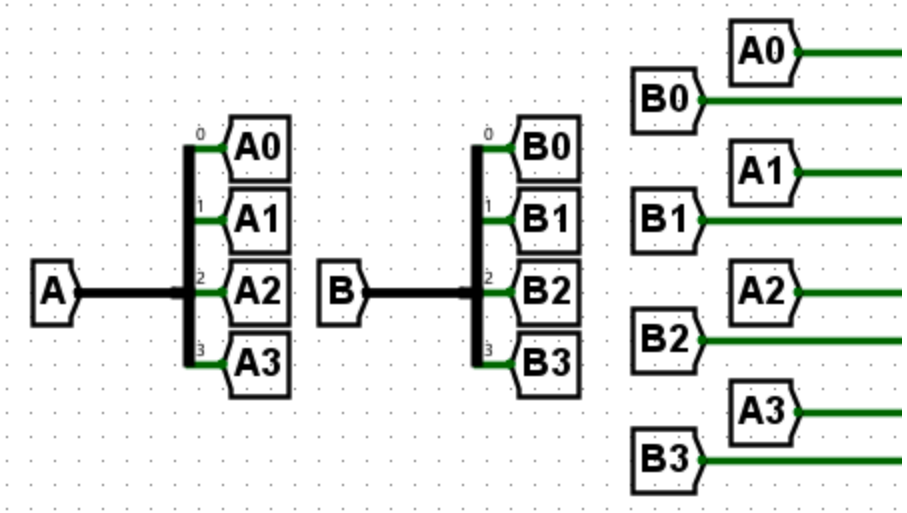


     - 数字信号输入`0`和`1`的位宽都是1。
     - 数字信号输入`101`和`010`的位宽都是3，`010`本身的十进制值是2，在二进制下为`10`，是两位的，但`010`这个信号的位宽不会因为自身的值而发生改变，仍然是3。
3. 蓝线说明这条线尚未有信号通过，是常见形态，输入信号后即呈现绿色。

Homework 1 Guide

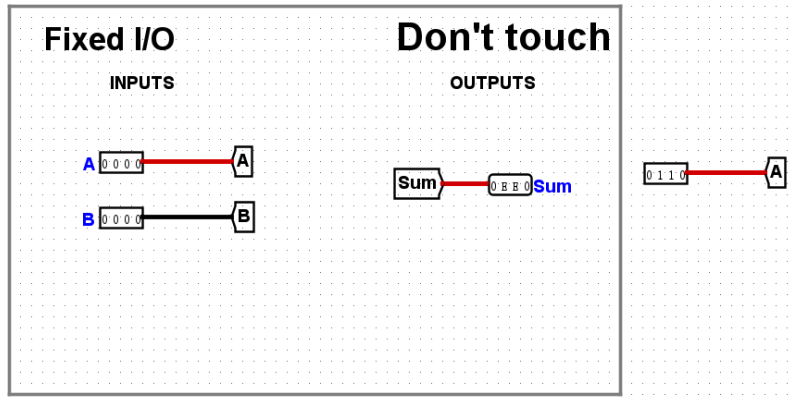
元件的规范使用（一）Tunnels

Tunnel的规范使用如下，它会简化导线连接在电路图上的呈现。



元件的规范使用（二）Pins & Constants

Pin 能够创造一个数字信号输入电路，但是请不要对着一个Tunnel输入两组数字信号！

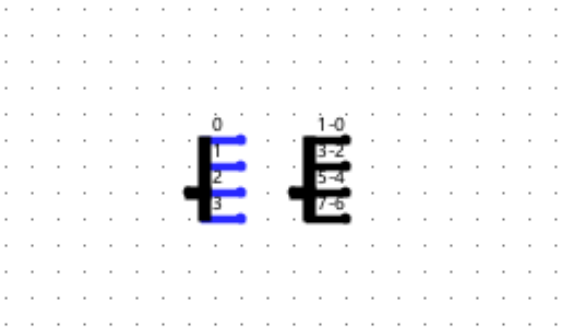


- Constant能够创造一个不会更改的数字信号输入电路（例如`0`、`1`、`010`），它的使用方式和Tunnel和Pin一样。

Homework 1 Guide

元件的规范使用（三）Splitter

Splitter 能够将一路多位数字信号输入拆分为多路一位（或多位）数字信号输入。下面展示了一个将一路八位信号输入拆分为四路两位信号输出的Splitter：



- `Bit Width In` 是输入的数字信号的位宽。
- `Fan Out` 是输出数字信号的路数，也就是个数。
- 最后八行：
  - 左侧从上到下分别是输入信号从最低位0到最高位7的每一位。
  - 右侧从上到下分别表示这一位在四路输出信号中的哪一路。
    例如，输入信号10010101，输出信号从低到高四路为：
    - 输出第一路（0-1位）：10 | 输出第二路（2-3位）：10
    - 输出第一路（4-5位）：10 | 输出第一路（6-7位）：01

Splitter (120,830)	
FPGA supported:	Supported
Facing	→ East
Fan Out	4
Bit Width In	8
Appearance	Centered
Spacing	1
Bit 0	0 (↑ Top)
Bit 1	0 (↑ Top)
Bit 2	1
Bit 3	1
Bit 4	2
Bit 5	2
Bit 6	3 (↓ Bottom)
Bit 7	3 (↓ Bottom)

Homework 1 Guide

Exercise 1.1: Half-Adder（半加器）

对于一个 1 Bit 的二进制（ `0` or `1` ）来说，它的加法被定义为：

- $0 + 0 = 0 \mid 1 + 0 = 1$
- $0 + 1 = 1 \mid 1 + 1 = (1)0$

其中，在 1+1 中出现的 (1) 被称之为“进位”标志。因此，在硬件上，我们定义半加器有两个输入 `a` 和 `b` 和两个输出 `sum`（表示 1 Bit 加法的结果） 和 `Cout`（表示进位），其公式为：

$$\text{sum} = a \oplus b$$
$$\text{Cout} = a \ \& \ b$$

半加器只能执行 1 Bit 的加法运算，这显然不够“给力”。我们需要进一步分析加法的逻辑，来构建更加有用的加法单元：全加器。

Homework 1 Guide

Exercise 1.2：Full Adder（全加器）

全加器的核心是考虑了进位（被记为 `Cin`）。对于某一位来说，我们除了考虑两个加数的对应位数 `a` 和 `b`，还要考虑是否有前一位传来的进位信息。因此，我们的输入就有三个数：`a`，`b`，`Cin`。我们需要构建这三个数到输出 `sum` 和 `Cout` 的逻辑运算。

运行逻辑

- `Cout`： `a`、`b`、`Cin` 之中至少两个为高电平时，输出高电平。此时我们对输入两两取逻辑与，如果 `a`、`b`、`Cin` 之中至少有两个高电平，那么三个逻辑与的输出也至少有一个为高电平，对逻辑与的三个输出取逻辑或即可。
- `sum`： `a`、`b`、`Cin` 之中仅有一个或全部为高电平时，输出高电平。将 `a`、`b`、`Cin` 输入一个三输入异或门即可。（注意！Logisim中的“三输入异或门”有不同的定义方式）

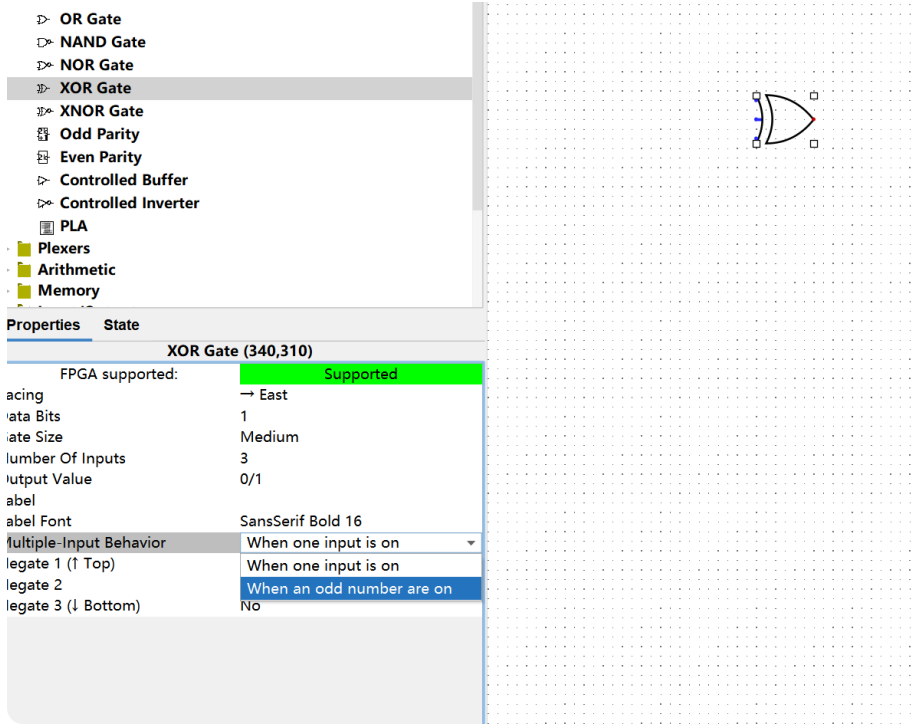
Homework 1 Guide

Exercise 1.2: Full Adder（全加器）

关于“三输入的异或门”的两种定义方式：

1. “When One Input is On"：只有一个输入为高电平时，输出高电平。
2. “Odd Number of Inputs On"：奇数个输入为高电平时，输出高电平。

你可以仔细思考一下这两种定义方式的异同，然后想一想，我们的全加器应该使用哪种定义方式。



Homework 1 Guide

Exercise 2.1: 4-Bit Full Adder

刚才，我们设计了一位的全加器。这种全加器引入了 Cin 和 Cout 的概念，可以很方便的扩展到多位全加器，从而实现我们常规认知意义上的加法运算。

现在，我们尝试将 1-Bit 的全加器扩展到四位：对于每一位我们都需要构建包含三输入 A、B、Cin，两输出 Sum 和 Cout 的全加器。然后将四个全加器连接起来，使得每一个全加器的 Cout 都连接到下一个全加器的 Cin 上。

Homework 1 Guide

Exercise 2.1: 4-Bit Full Adder

在全加器的实现中，我们忽略可能出现的“多”的一位（称之为“溢出”位），这个值体现为全加器模块 Cout 的值，例如：

```
0b1111(15) + 0b1111(15) = 0b11110(30)
```

- 这里我们的全加器输出应该是 `(Cout)1 (其他部分)1110`
- 我们只需要输出 `1110` 部分即可。

Homework 1 Guide

Exercise 2.2: Subtracter

所谓“补码”，是指在二进制数的表示中，用一种特殊的方式表示负数：

- 正数直接由二进制表示。
  - 例如：5 被表示为 `0101`。
- 负数的表示是将对应正数的二进制表示的每一位取反，然后加1。
  - 例如：-5 对应的正数是 5，其二进制表示是 `0101`，取反得到 `1010`，加1得到 `1011`。

在数字电路中，如果我们使用补码实现数字的表示，则“对一个数取负数”的操作是非常简单的。

Homework 1 Guide

Exercise 2.2: Subtracter

使用了 `Cin` 和 `Cout` 结构的全加器可以很方便的实现减法：

- 把最低位的 `Cin` 接到常数 `1` 上（表示 +1）
- 把输入 `B` 取反

我们就得到了一个减法器。事实上，这个减法器所计算的结果就是 $A - B$。

- 注意，减法器也不需要考虑最高位 `Cout` 的输出。

Homework 1 Guide

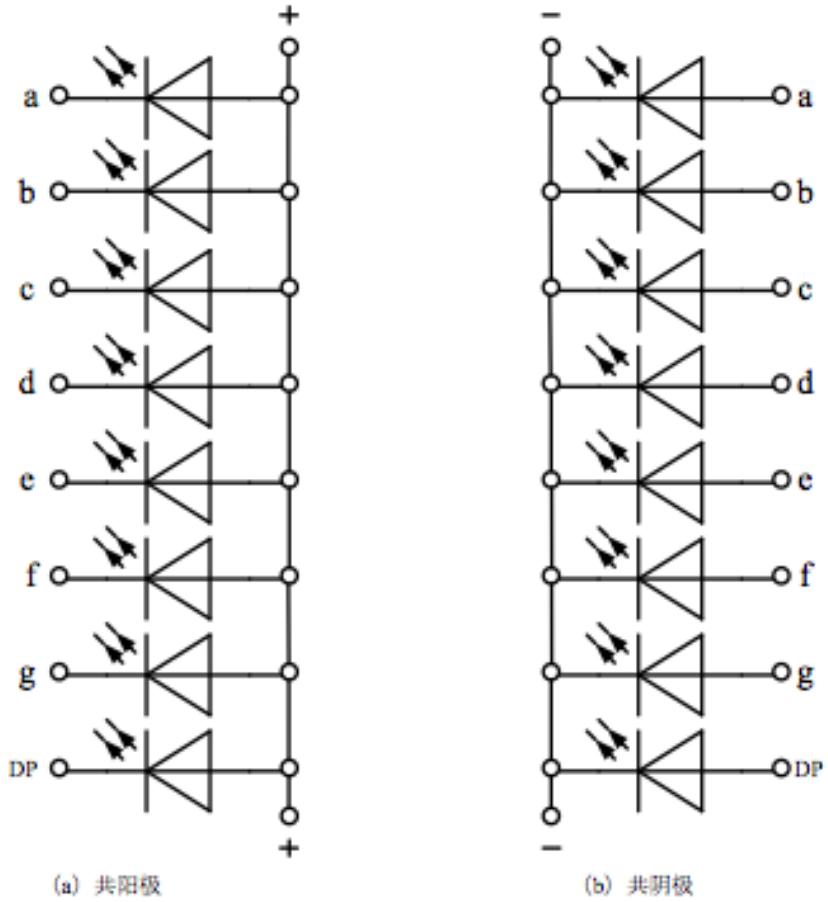
Exercise 3: 七段数码管的解码器

七段 LED 数码管是一种常用的显示元件，常应用于手表、计算器等仪器中，用于显示数值。

数码管分为共阴极和共阳极两种类型：

- 共阴极就是将七个 LED 的阴极连在一起，让其接低电平。这样给任何一个 LED 的另一端高电平，它便能点亮。
- 而共阳极就是将七个 LED 的阳极连在一起，让其接高电平。这样，给任何一个 LED 的另一端低电平，它就能点亮。

这里的 LED 是共阴极 LED。

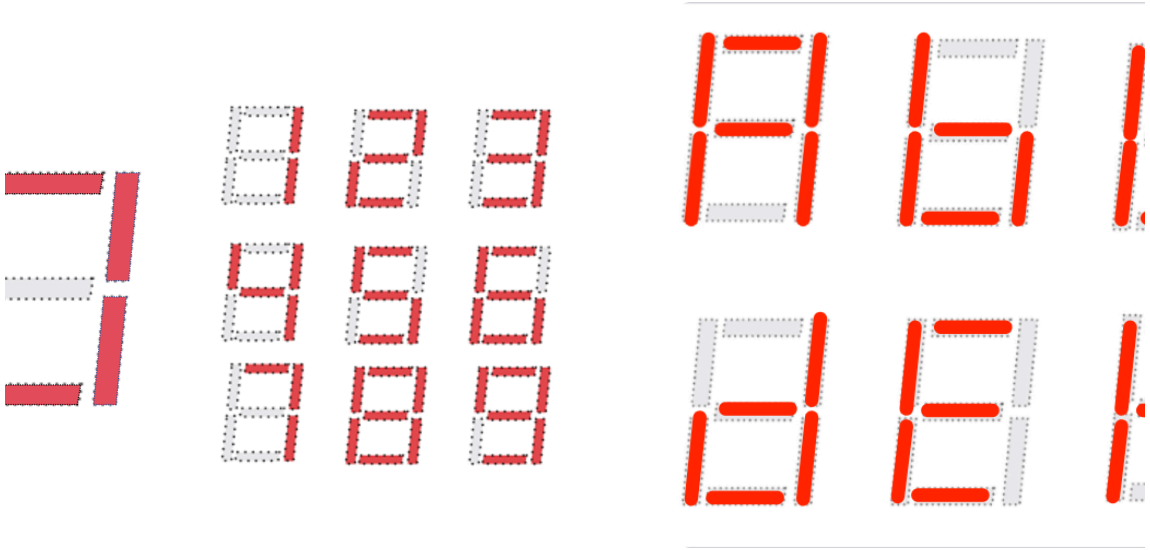


Homework 1 Guide

Exercise 3: 七段数码管的解码器

右图为数码管显示的数字 0~9 及十六进制的 a~f 的显示方式。

可以看到，每一个数字或字母都可以用七个二进制位表示，每一个二进制位对应数码管的一个引脚。我们所要做的就是将输入的数字或字母转换为七个二进制位，然后输出到数码管的引脚上。



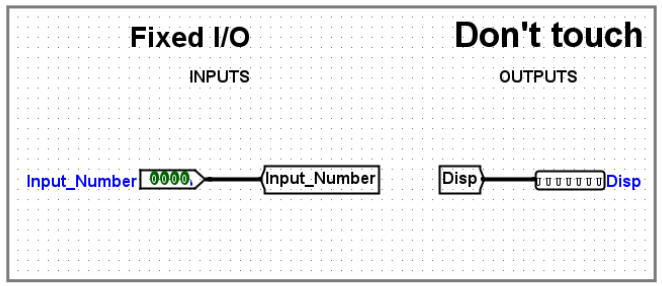
Homework 1 Guide

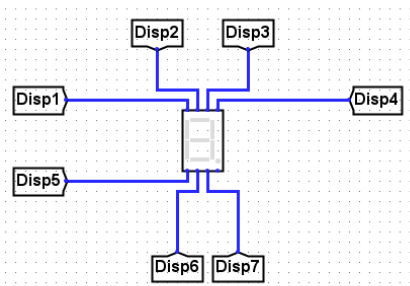
Exercise 3: 七段数码管的解码器

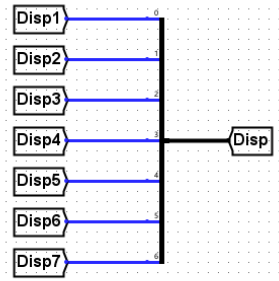
我们将以1号管为例讲解。**观察.circ文件：**

- **Input_Number**: 是一个四位二进制输入→用以表示一位十六进制
- **Disp**: 是一个七位二进制输出→用以控制七根LED管开关

既然只研究1号管，我们只需要关心： **四位输入是如何决定Disp1的高低电平的?**



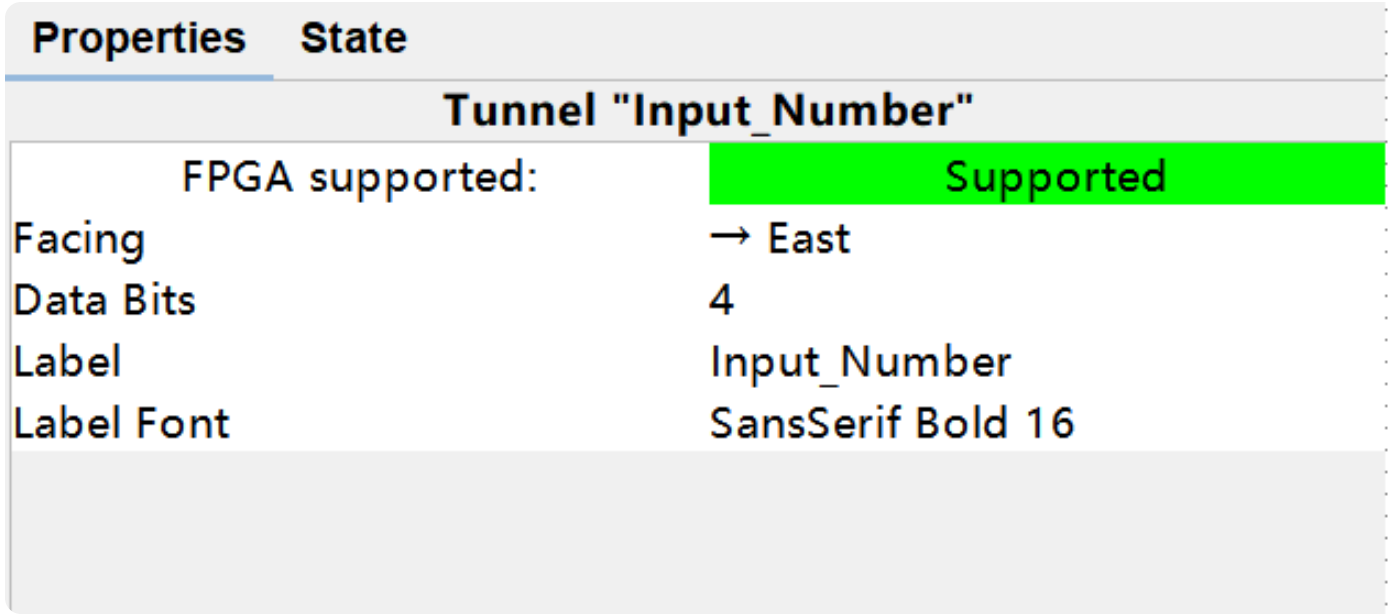


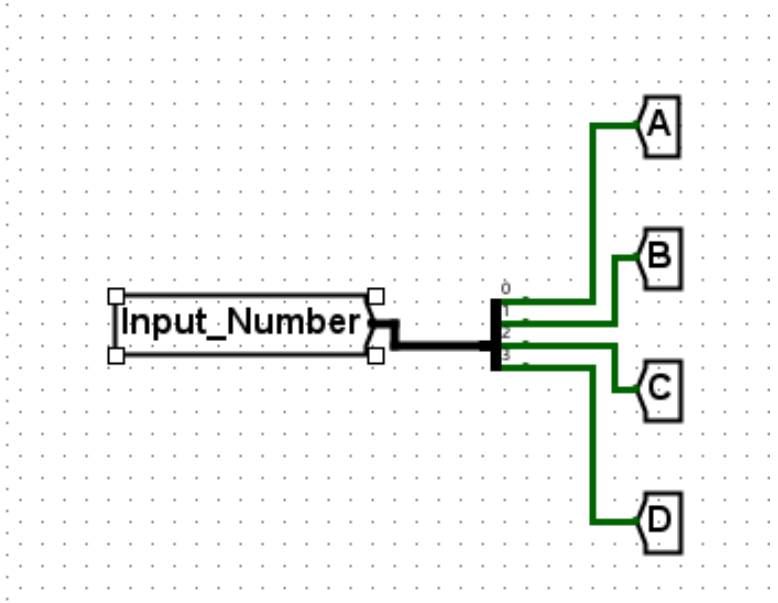


Homework 1 Guide

Exercise 3: 七段数码管的解码器

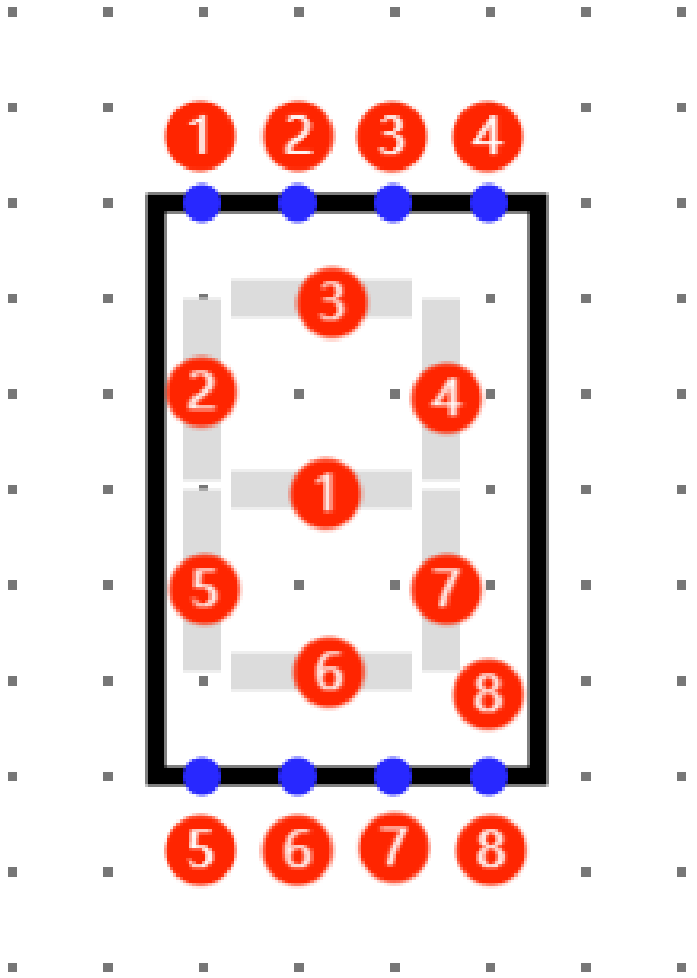
在 4-1 选择器中，我们已经熟悉了两位二进制输入表示一位四进制的想法，这里我们用分线器将输入拆分为独立的四个二进制输入 A、B、C、D。





Homework 1 Guide

Exercise 3: 七段数码管的解码器



你的工作流程应该是:。
真值表→SOP形式→ 化简（卡诺图）（可选）→逻辑表达式→构建AOI电路

- SOP：Sum Of Product
- 卡诺图：用于化简 SOP 逻辑表达式的一种工具
- AOI：And Or Inverter (反相器，用于实现 Not 逻辑) 电路，只有and or not门的电路

我们将以1号管为例讲解。

Homework 1 Guide

Exercise 3: 七段数码管的解码器

- 需要点亮一号管的数是:
  `2 3 4 5 6 8 9 A B d E F`
- 因此，我们需要构建的电路只在这些数的时候输出高电平。
- 最简单的方法是：**直接构造对应的 SOP 表达式**。但这样的工作量非常巨大。
- 因此需要**对这个表达式进行化简**，这里我们可以使用卡诺图。
- 接下来，按照我们上面给出的 Practice Example 的方法，构建电路即可。

二进制	十进制	十六进制
0000 0000	0	0
0000 0001	1	1
0000 0010	2	2
0000 0011	3	3
0000 0100	4	4
0000 0101	5	5
0000 0110	6	6
0000 0111	7	7
0000 1000	8	8
0000 1001	9	9
0000 1010	10	a
0000 1011	11	b
0000 1100	12	c
0000 1101	13	d
0000 1110	14	e
0000 1111	15	f