

# 目录

## A 一般使用

1. Digital .....	6
1.1. 简介 .....	6
1.2. 起步 .....	6
1.3. 导线 .....	13
1.4. 层次设计 .....	14
2. 仿真 .....	17
2.1. 传输延迟 .....	17
3. 分析和综合电路 .....	18
4. 硬件 .....	18
4.1. GAL16v8 和 GAL22v10 .....	18
4.2. ATF150xAS .....	18
4.3. 导出 VHDL 或 Verilog .....	18
5. 自定义外观 .....	19
6. 通用电路 .....	19
7. 使用脚本控制测试 .....	19
8. 常见问题 .....	20
9. 快捷键 .....	21

## B 设置

## C 命令行界面

## D 组件

1. 逻辑 .....	
1.1. 与门 .....	28
1.2. 与非门 .....	28
1.3. 或门 .....	29
1.4. 或非门 .....	30
1.5. 异或门 .....	30
1.6. 异或非门 .....	31
1.7. 非门 .....	32
1.8. 查找表 .....	32
2. 输入输出 .....	
2.1. 输出 .....	33
2.2. LED .....	33
2.3. 输入 .....	34
2.4. 时钟输入 .....	35
2.5. 按钮 .....	35
2.6. DIP开关 .....	36
2.7. 探测器 .....	36
2.8. 数据图 .....	37
2.9. 示波器 .....	37
3. 输入输出 - 显示 .....	
3.1. RGB-LED .....	37
3.2. 发光二极管 .....	38
3.3. 带有LED的按钮 .....	39
3.4. 7段数码管 .....	39

3.5. 7段数码管(十六进制输入)	40
3.6. 16段数码管	41
3.7. 灯泡	41
3.8. LED矩阵	42
4. 输入输出 - 机械	
4.1. 旋转编码器	42
4.2. 单极性步进电机	43
4.3. 双极性步进电机	43
5. 输入输出 - 外设	
5.1. 键盘	44
5.2. 终端	45
5.3. Telnet	45
5.4. VGA显示器	46
5.5. MIDI	47
6. 导线	
6.1. 地	48
6.2. 电源	48
6.3. 常量	48
6.4. 隧道	49
6.5. 分裂器/合并器	49
6.6. 驱动器	50
6.7. 驱动器(低电平有效)	51
6.8. 延迟	51
6.9. 上拉电阻	52
6.10. 下拉电阻	52
6.11. 无连接	52
7. 复用器	
7.1. 复用器	53
7.2. 多路分配器	53
7.3. 解码器	54
7.4. 位选择器	55
7.5. 优先级编码器	55
8. 触发器	
8.1. SR 触发器	56
8.2. SR 触发器(时钟控制)	57
8.3. JK 触发器	57
8.4. D 触发器	58
8.5. T 触发器	59
8.6. JK 触发器(异步)	60
8.7. D 触发器(异步)	61
8.8. 单稳态触发器	62
9. 存储器 - RAM	
9.1. RAM(独立端口)	63
9.2. Block-RAM(独立端口)	64
9.3. RAM(双向端口)	64
9.4. RAM(片选)	65
9.5. Register File	66
9.6. RAM(多端口)	67
9.7. 异步 RAM	68
9.8. 显存	69

10. 存储器 - EEPROM	
10.1. EEPROM	70
10.2. EEPROM(独立端口)	71
11. 存储器	
11.1. 寄存器	72
11.2. ROM	73
11.3. 多端口 ROM	74
11.4. 计数器	75
11.5. 计数器(可预设)	76
11.6. 随机数生成器	77
12. 运算器	
12.1. 加法器	77
12.2. 减法器	78
12.3. 乘法器	79
12.4. 除法器	79
12.5. 桶式移位器	80
12.6. 比较器	81
12.7. 补码器	81
12.8. 符号扩展器	82
12.9. 位计数器	82
13. 开关	
13.1. 开关	83
13.2. 双掷开关	83
13.3. 继电器	84
13.4. 双掷继电器	85
13.5. P 沟道场效应晶体管	85
13.6. N 沟道场效应晶体管	86
13.7. Fuse	87
13.8. 二极管(上拉)	87
13.9. 二极管(下拉)	87
13.10. P 沟道浮动门场效应晶体管	88
13.11. N 沟道浮动门场效应晶体管	89
13.12. 传输门	89
14. 其他	
14.1. 测试用例	90
15. 其他 - Decoration	
15.1. Text	90
15.2. 矩形	91
16. 其他 - Generic	
16.1. 通用电路初始化	91
16.2. Code	92
17. 其他 - VHDL/Verilog	
17.1. External	92
17.2. 外部文件	93
17.3. 双向管脚	93
18. 其他	
18.1. 电源连接器	94
18.2. 双向分裂器	94
18.3. 复位器	95
18.4. Break	95

18.5. 停止 ..... 96

18.6. 异步时序 ..... 96

**E** 库

## A 一般使用

### 1. Digital

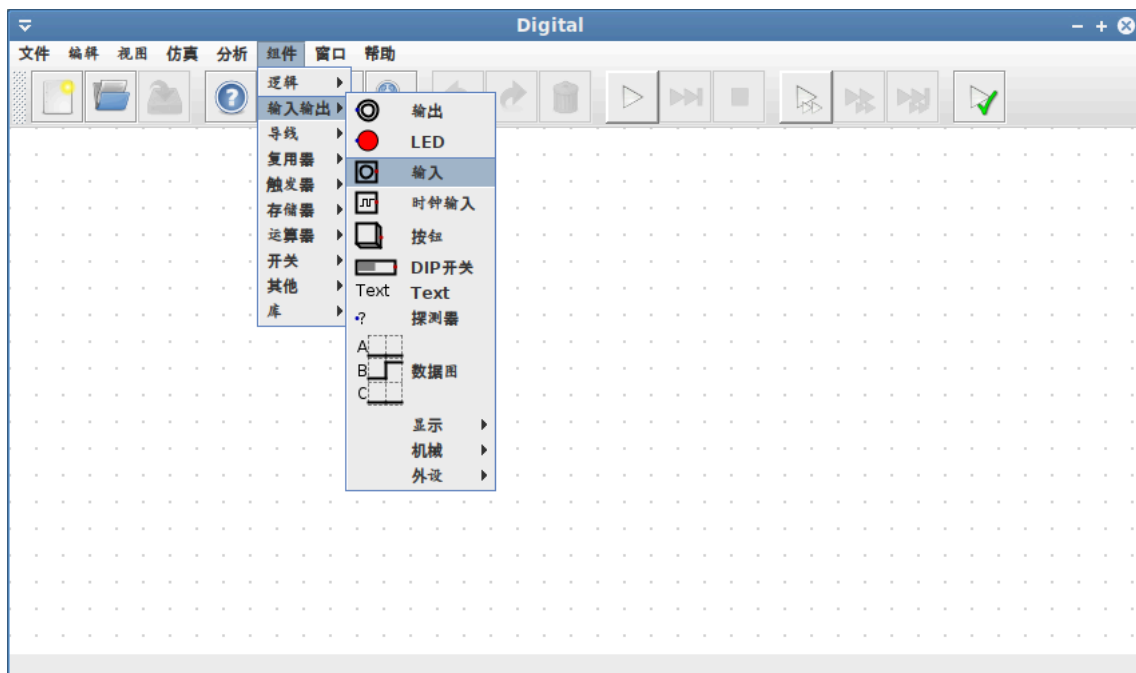
#### 1.1. 简介

**Digital** 是一个简单的数字电路仿真器。通过导线将各种逻辑门连接起来，可以仿真整个电路的行为。用户可以通过点击按钮或者设置输入值实现交互式仿真。

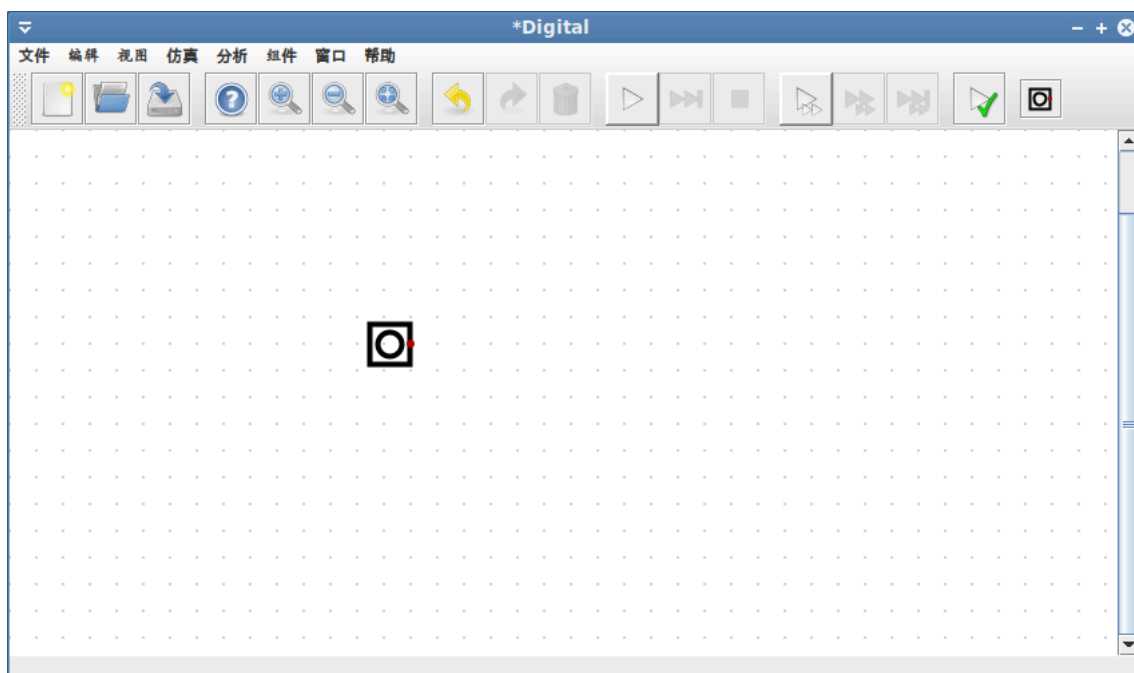
通过这种方式，多数数字电路中的基本电路可以被构建和仿真。在 "examples" 文件夹内，用户可以浏览各种示例，如 1 个 16 位的单周期哈佛架构处理器。

仿真器有两种操作模式：编辑和仿真模式。在编辑模式，不允许仿真，只允许修改电路。用户可以添加或连接各种组件。可以通过点按工具栏中的“启动仿真电路”按钮开启仿真模式。在开始仿真电路之前，将会先检测电路的一致性。如果有某些错误，将会显示对应的消息提示并且相关的组件或导线将会高亮。如果没有任何错误，将会开启仿真模式。这时，你可以和正在运行的仿真进行交互。

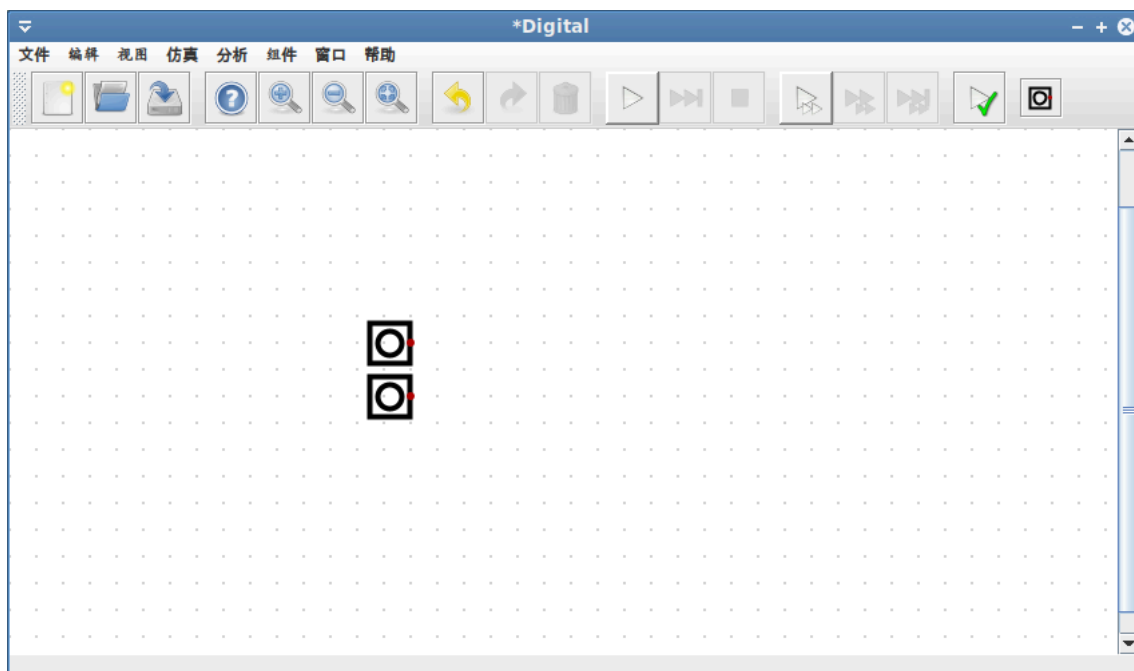
#### 1.2. 起步



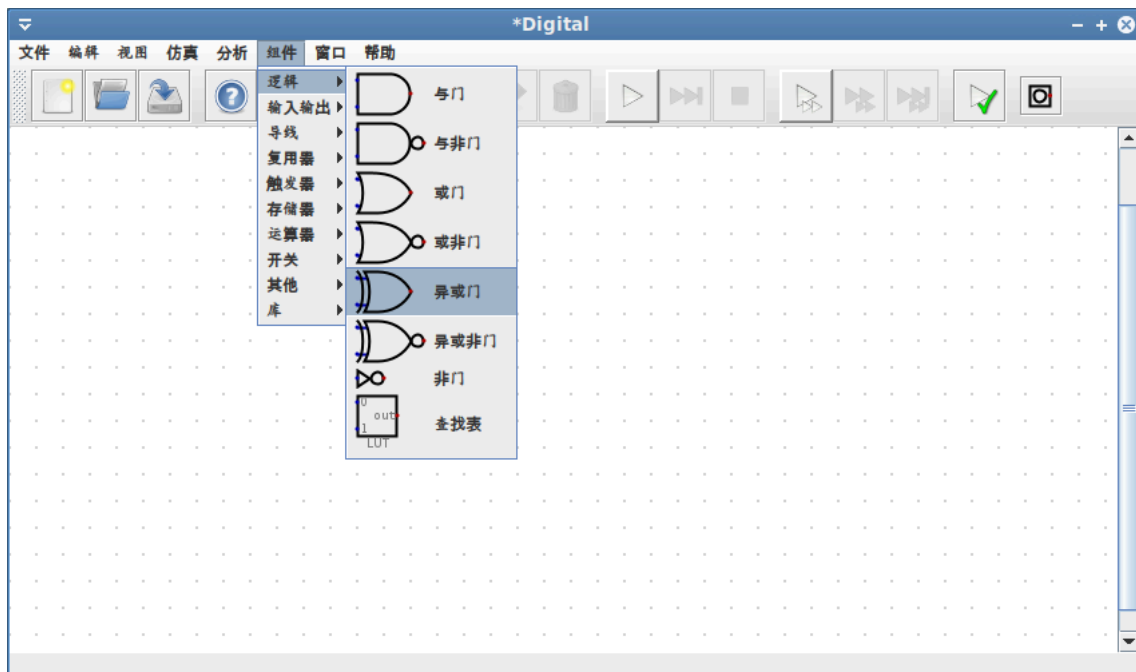
作为第一个示例，将会构建一个由异或门组成的电路。在主窗口，“组件”菜单允许我们选择各种组件。然后它们会被放置于绘图面板。在任何时候，可以通过 **ESC** 键取消操作。让我们开始选择一个输入组件，该组件在仿真时可以通过鼠标交互进行控制。



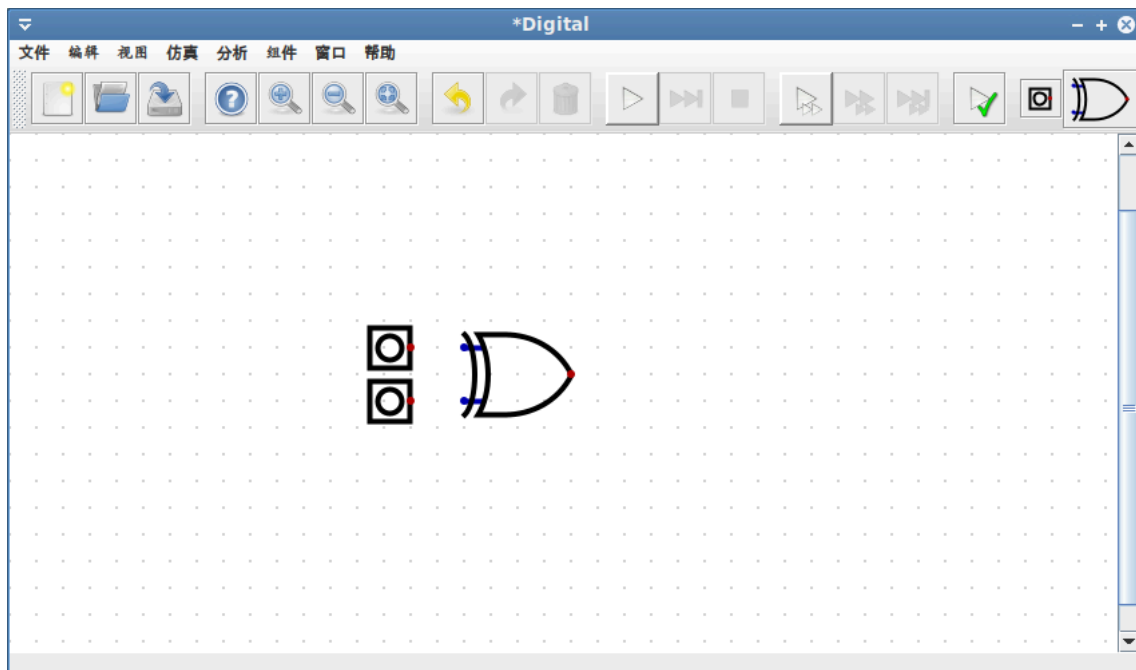
选择后，输入组件可以被放置在绘图面板。红色的点表示组件和导线之间的连接点，在之后会被连接。同时红色表示输出，意思为该端口定义了一个信号，该信号可以驱动导线。



使用同样的方式，添加第二个输入组件，最好将其放置在第一个输入组件下面。

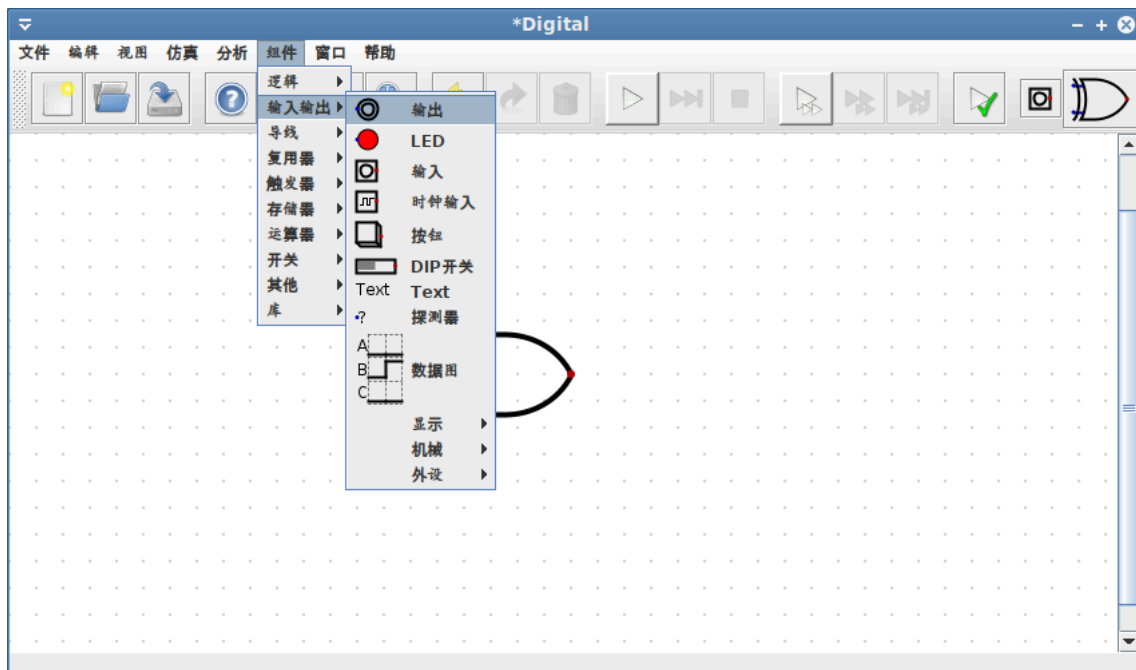


添加完输入组件，我们选中异或门。该门表示实际的逻辑功能。

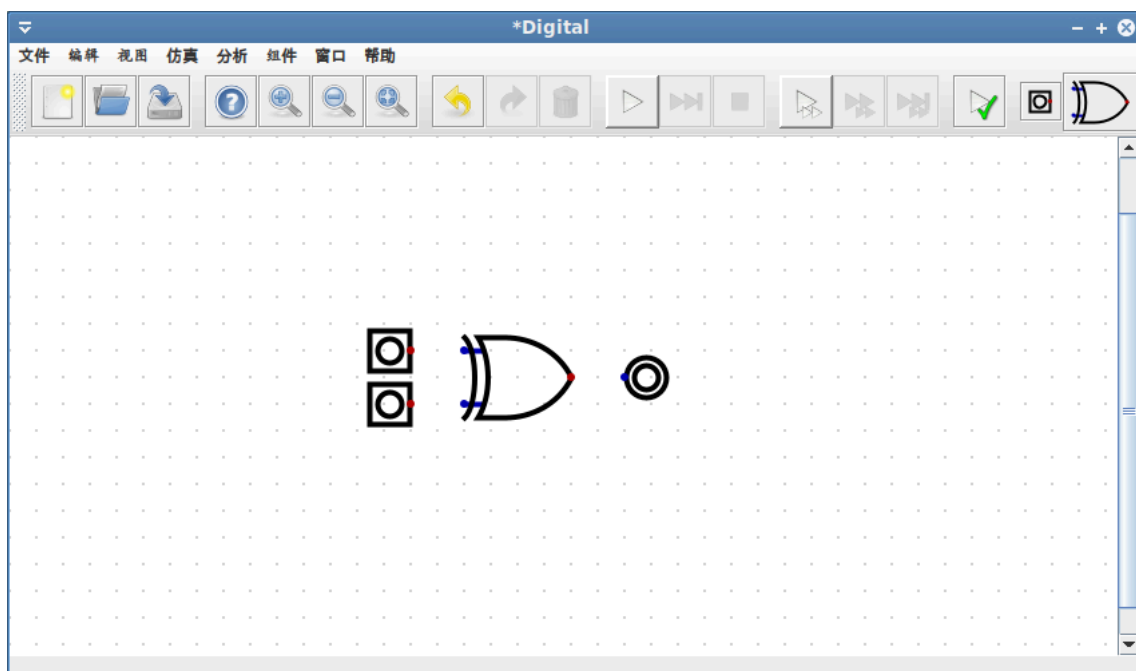


选中后我们可以将其添加到电路。最好将其放置在合适的位置使得后续的连线尽可能简单。蓝色的点表示门的输入端口。

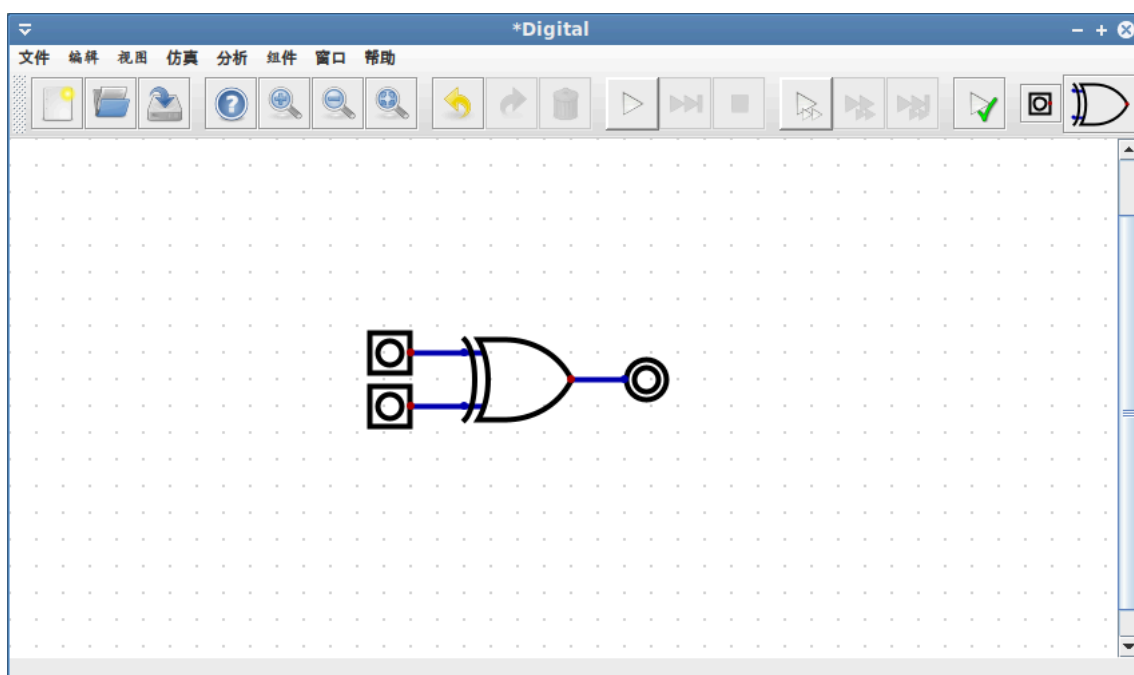




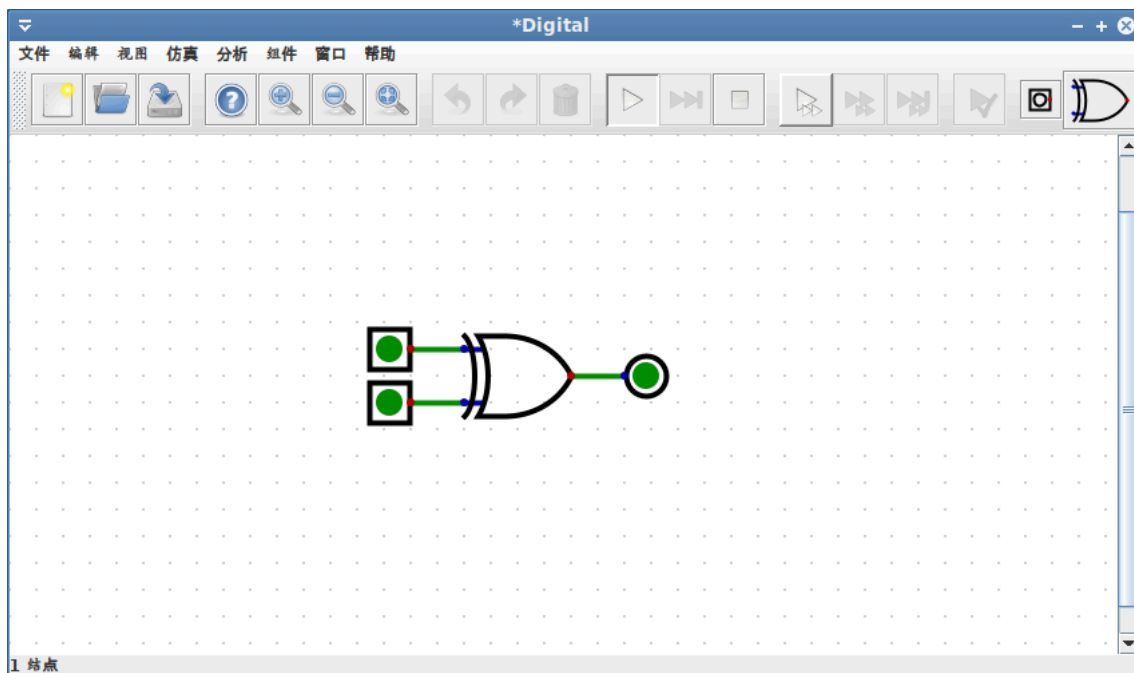
现在，选择一个输出组件，该组件用于显示一个信号状态或者传递信号给一个子电路。



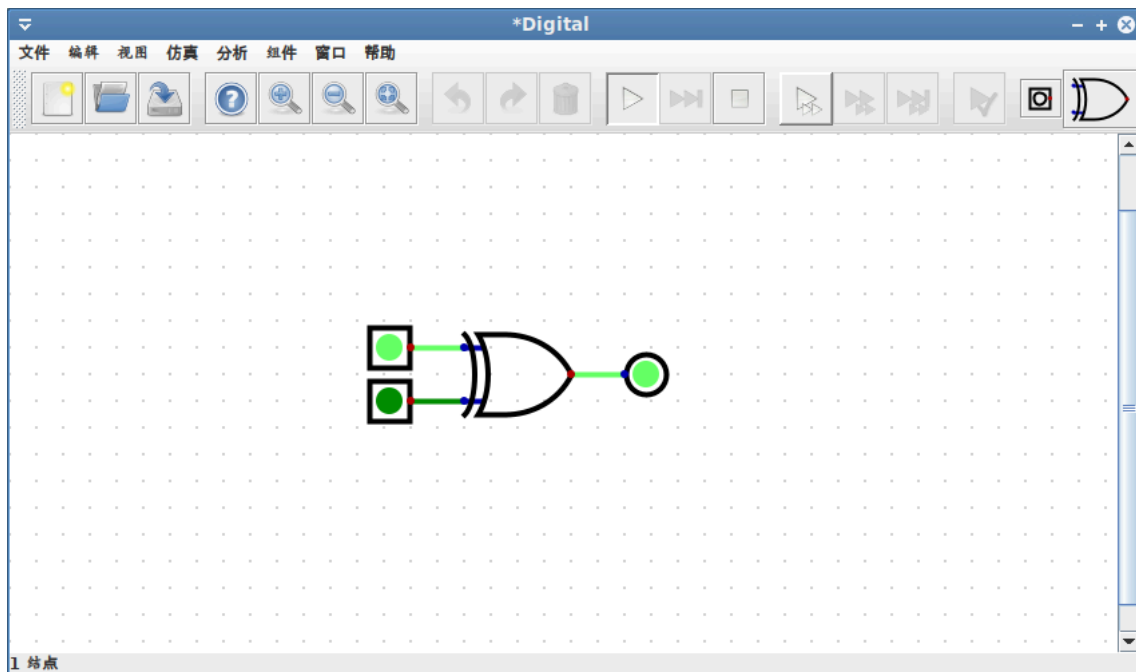
输出组件有一个蓝色的点，意思为输入端口，我们可以通过为其赋值以便之后使用。



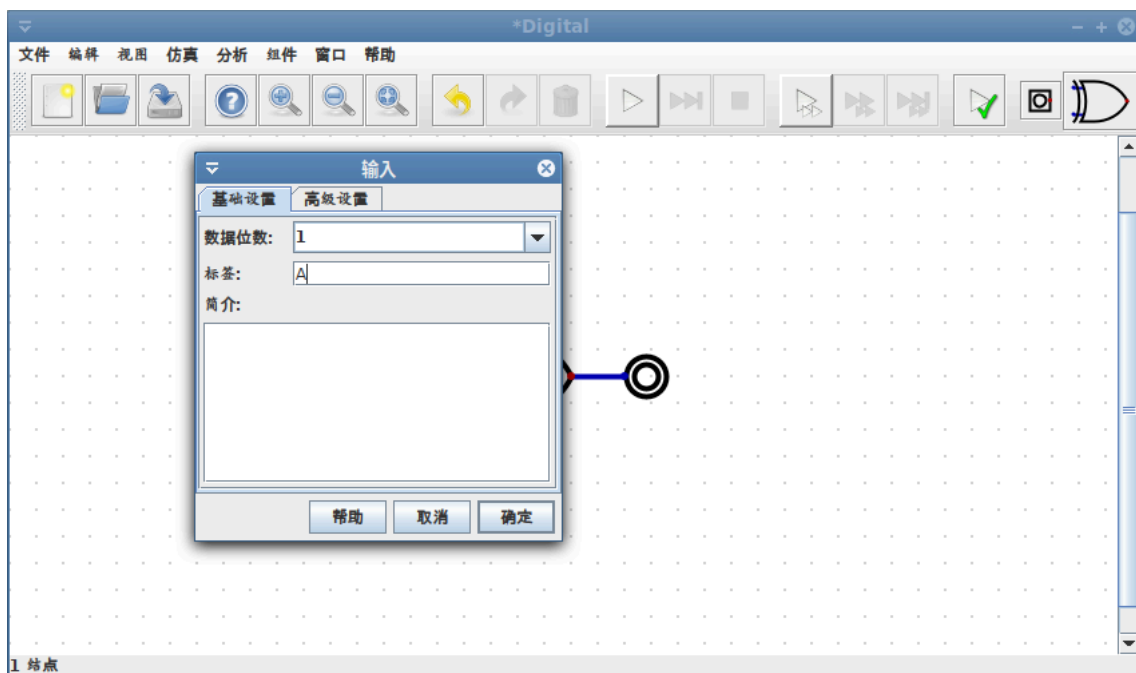
当所有组件放置好后，使用鼠标连接蓝色和红色的点。确保仅有一个红色的点被连接到任何蓝色的点。只有允许三态的输出可以打破该规则，允许连接到多个红色的点。当所有导线被绘制后，整个电路即完成。



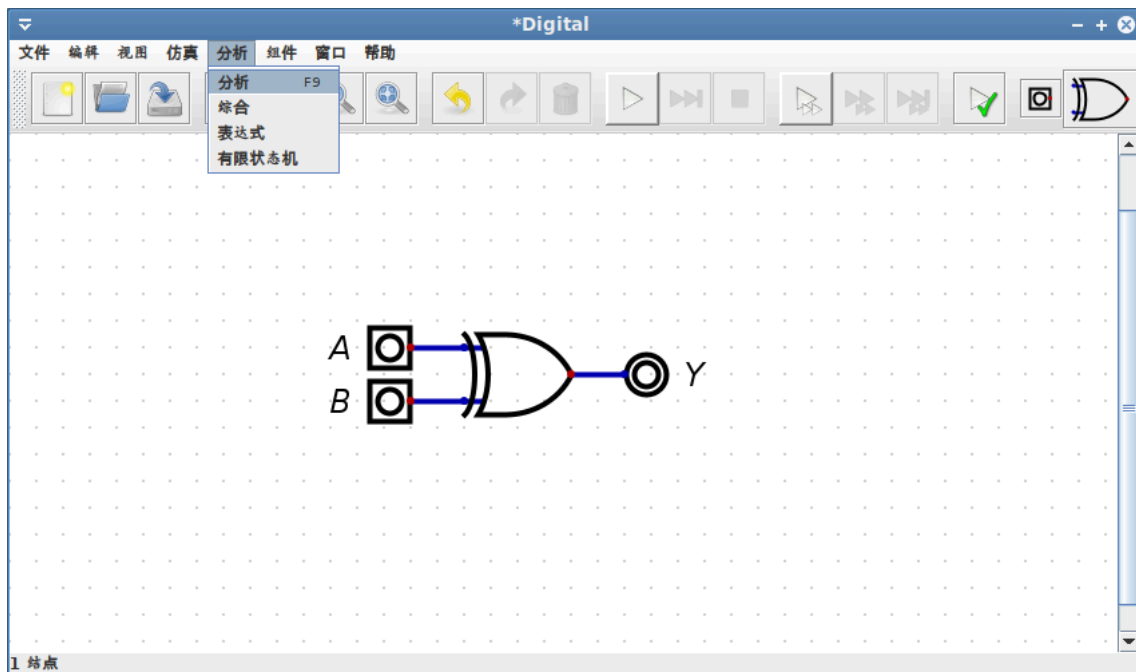
当仿真启动后，我们可以和电路进行交互。通过点击“启动仿真电路”按钮开始仿真。仿真开启后，导线的颜色改变并且输入、输出组件被填充。鲜绿色表示逻辑电平‘1’，深绿色表示逻辑电平‘0’。在上图中，所有的导线为‘0’。



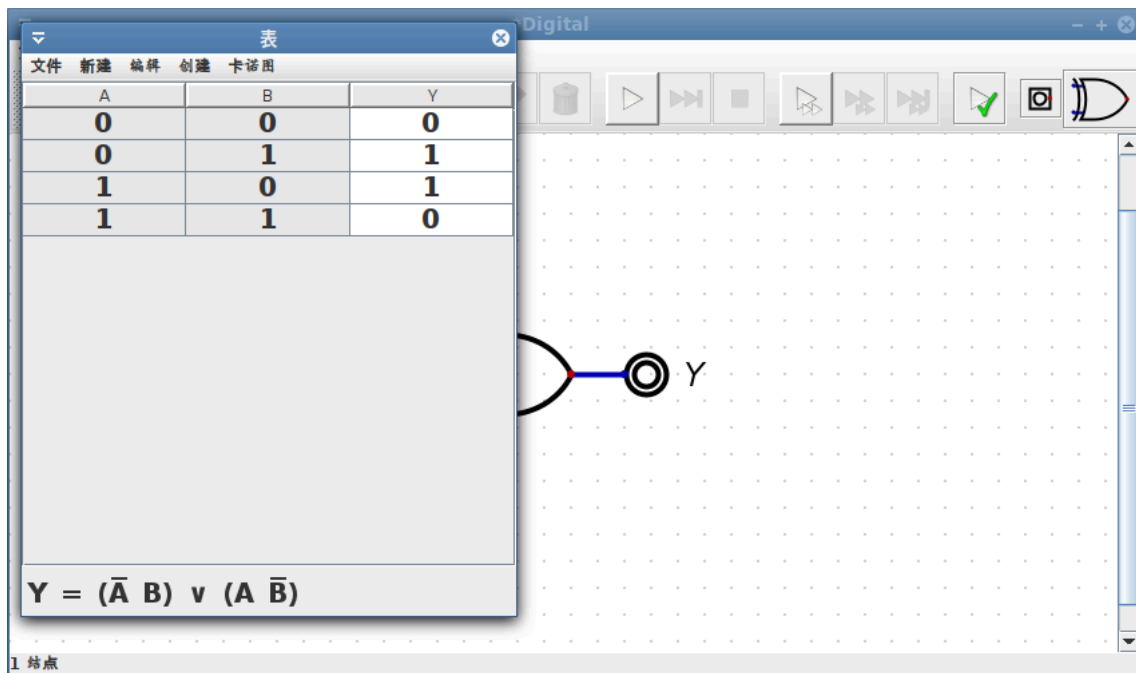
通过鼠标点击，可以切换输入组件的值。由于仿真已经运行，输出组件的值根据当前输入的状态而做出对应改变。电路的行为 和期望的异或门一样。



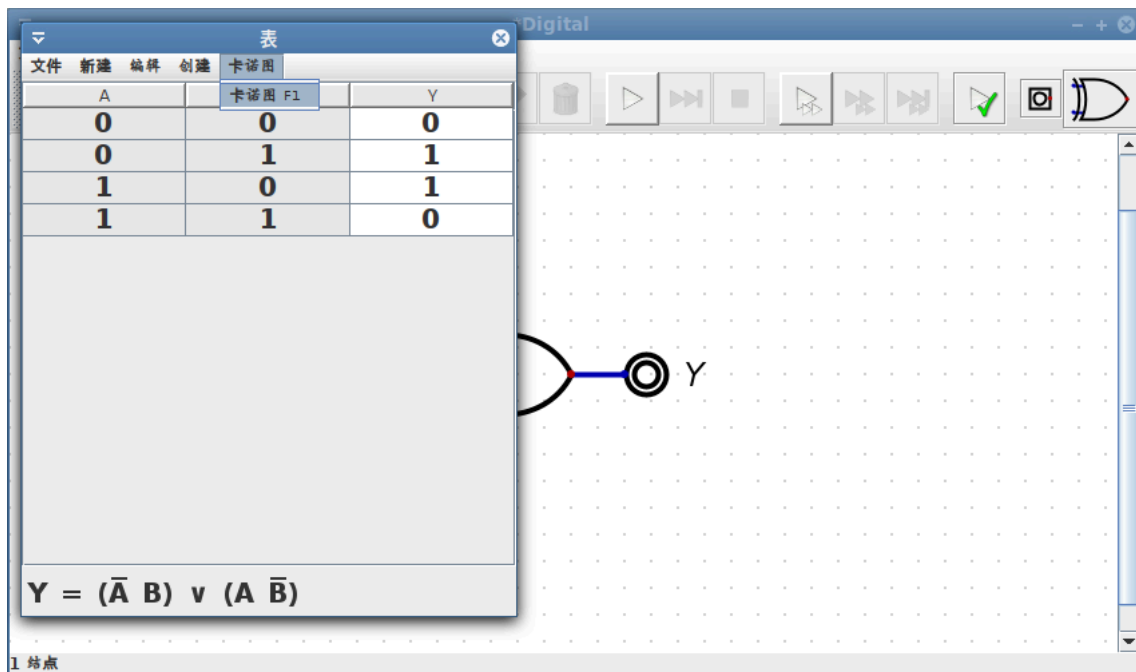
如果需要修改电路，必须先停止仿真。最简单的方式是通过点击工具栏的“停止”按钮。通过在组件上右击（MacOS 使用 **control**+左击）打开组件属性对话框。通过该对话框，我们定义第一个输入组件的标签为 ‘A’ 。



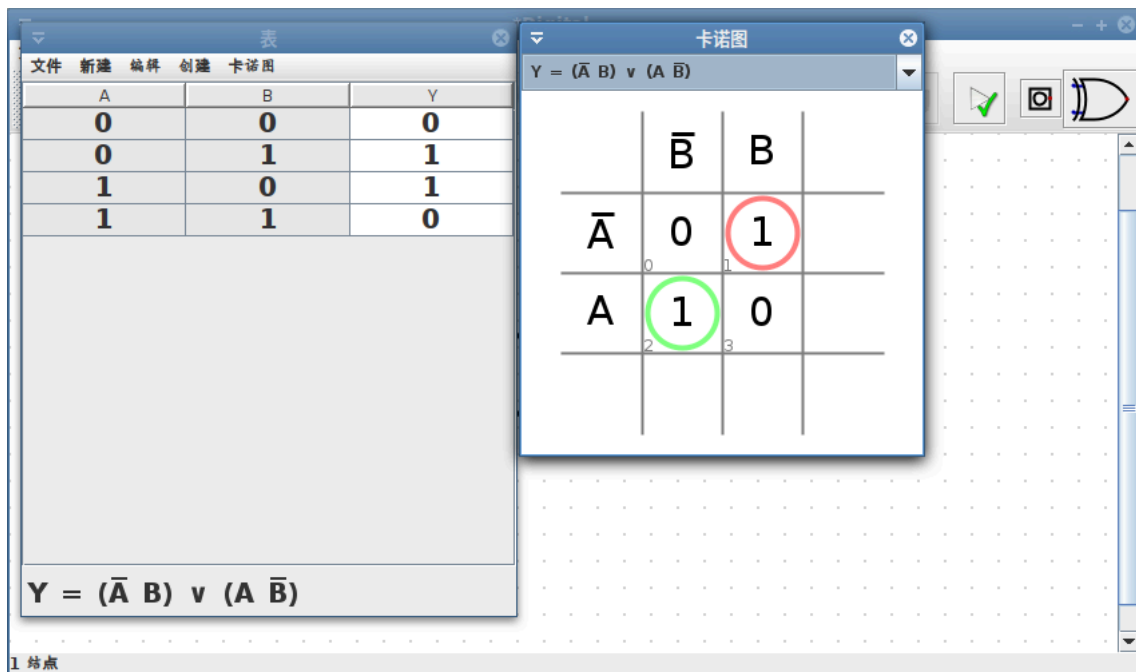
通过同样的方式，定义剩余组件的输入和输出端口标签。“分析”菜单包含一个同名的菜单项，该功能对当前电路进行分析。不过，需要先给所有的输入和输出端口添加标签才行。



仿真电路的真值表在一个新窗口中显示，在表的下面，我们可以找到一个和电路对应的布尔表达式。如果电路可能有多个布尔表达式，一个独立的窗口将会打开并显示所有可能的表达式。



真值表对话框有一个“卡诺图”菜单项，允许以卡诺图方式显示真值表。



在对话框的顶部，通过下拉列表允许选择期望的表达式。通过这种方式，我们可以评估多个等价的布尔表达式。然而，在该示例中，仅有一个最小表达式。点击卡诺图，可以实现对真值表的修改。

### 1.3. 导线

所有的组件必须通过导线连接起来。不可以通过将两个组件相邻放置实现连接。

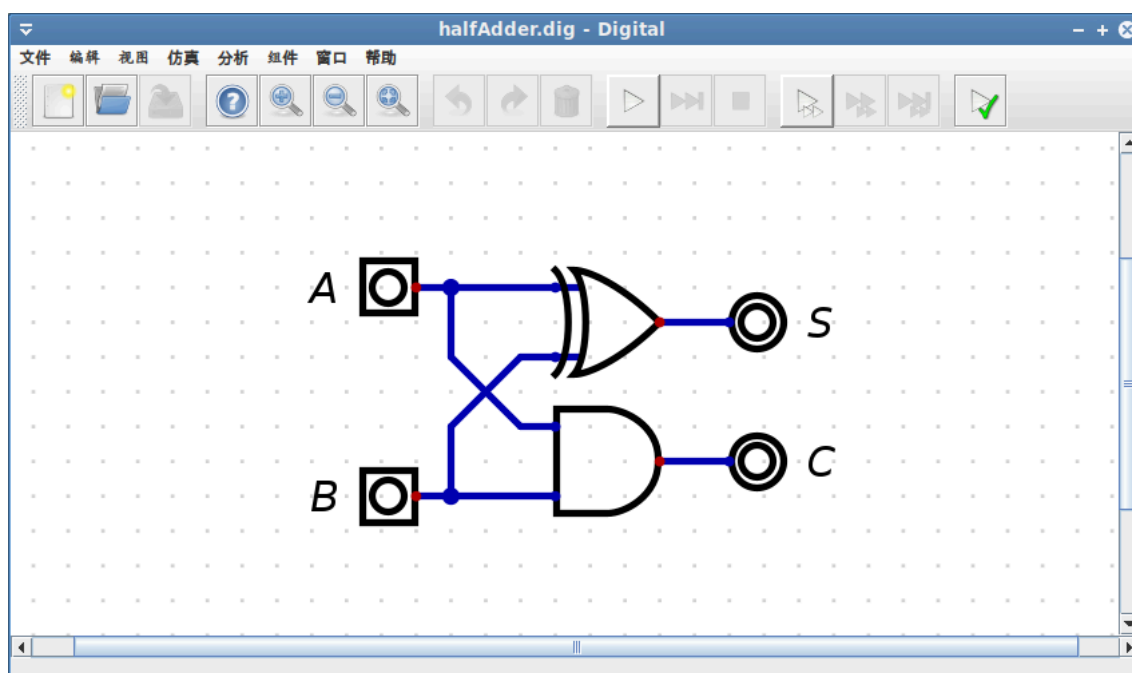
另外，仅可以通过导线的端点连接到组件，如果一个组件的管脚放置在一条导线中间，该组件和导线并不会实现连接。因此，导线必须在其所连接的管脚终止。即使使用隧道组件，也必须使用导线连接管脚和隧道组件。

如果需要移动组件，包括其连接的导线，则必须先框选该组件，如果不想同时移到其连接的导线，可以通过单击选择组件。

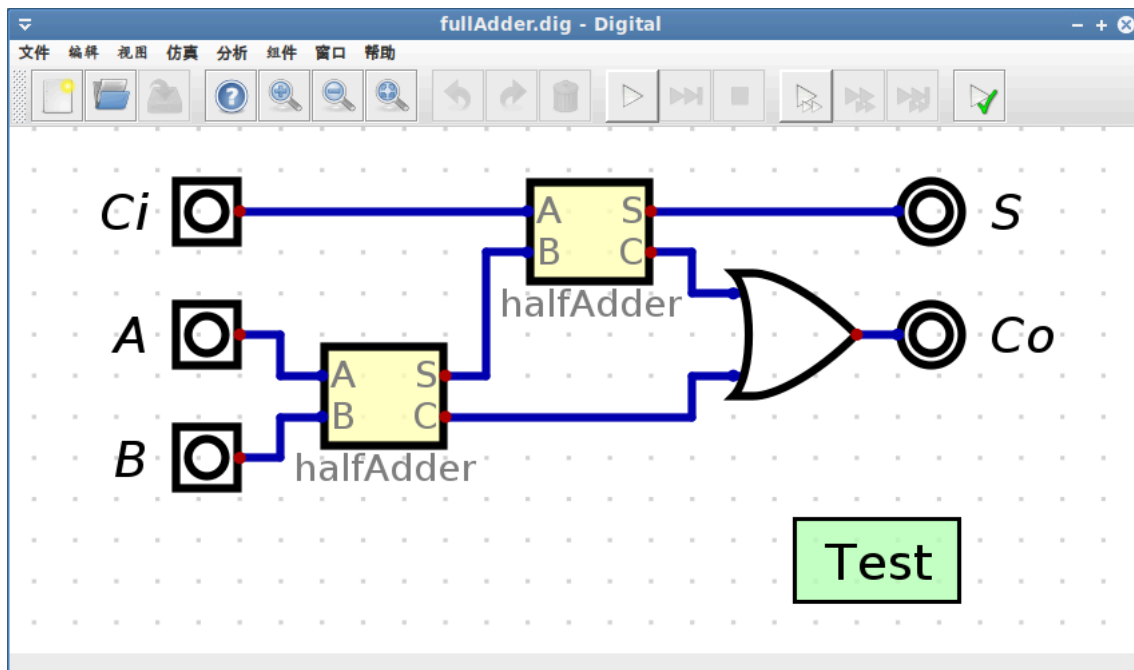
通过 **CTRL**-单击，可以选中且移动或删除一个单独的导线段。如果在画导线时按下 **D** 键，可以绘制斜线。键 **S** 可以将一条线段分割为两条。

#### 1.4. 层次设计

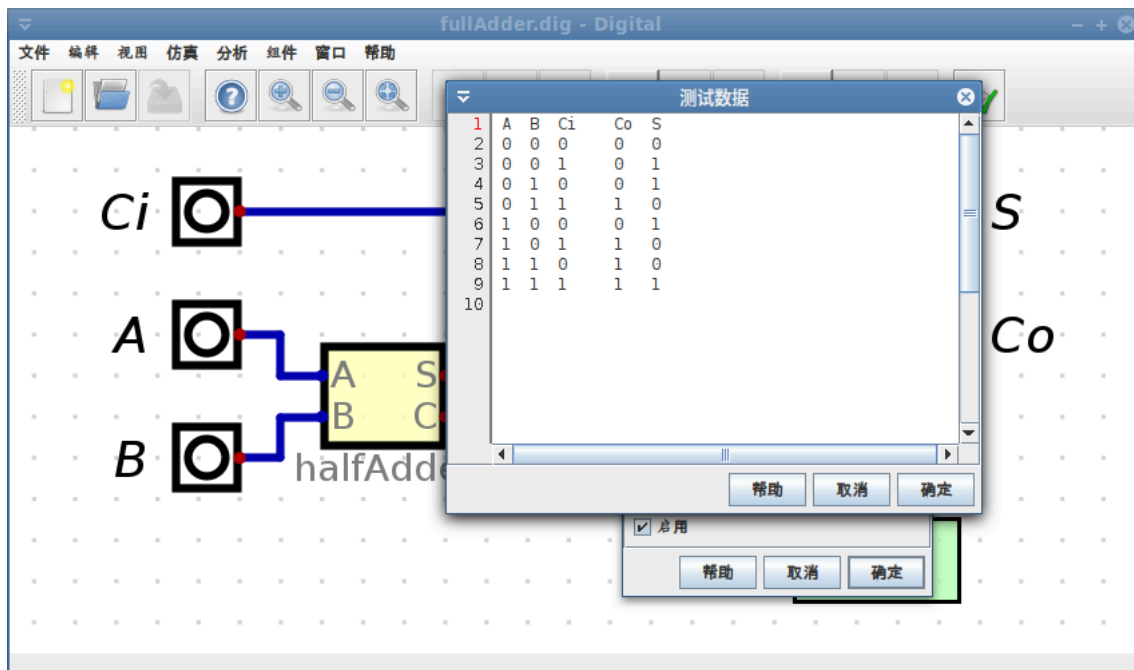
随着电路越来越复杂，电路将变得难以理解。为解决该问题，电路中不同的部分可以存放在不同的文件。该机制使得使用子电路变为可能。



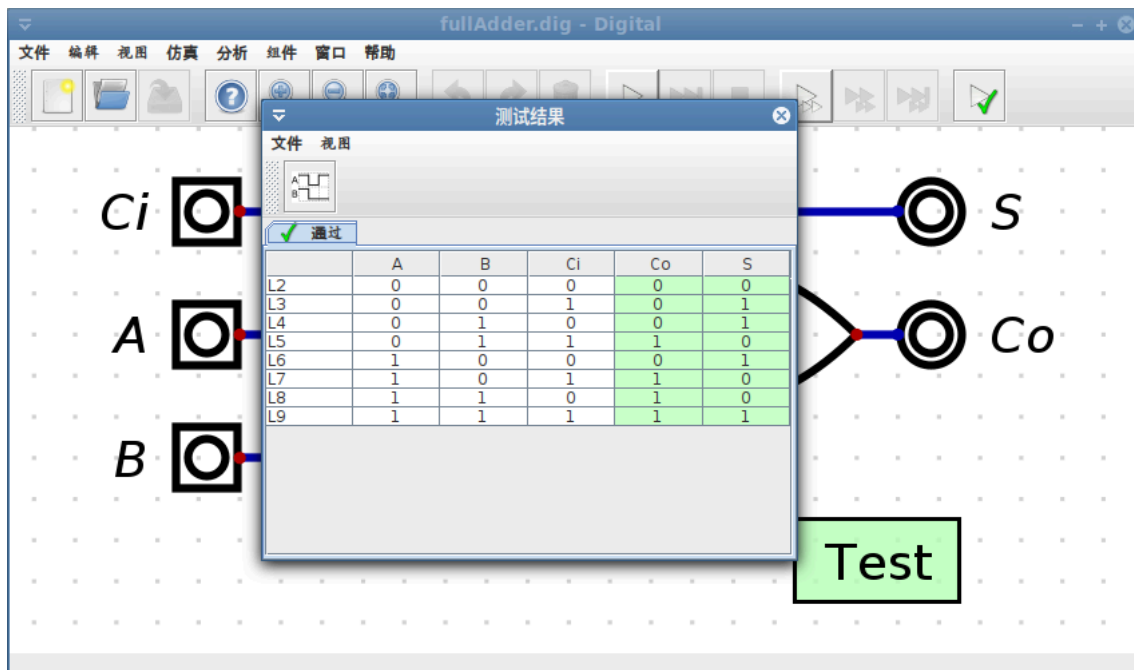
作为一个示例，我们考虑4位加法器：首先，我们构建一个简单的半加器，其包含一个异或门和一个与门。位 ‘A’ 和 ‘B’ 相加的结果通过 ‘S’ 和 ‘C’ 输出。该电路存储在文件 “halfAdder.dig”。



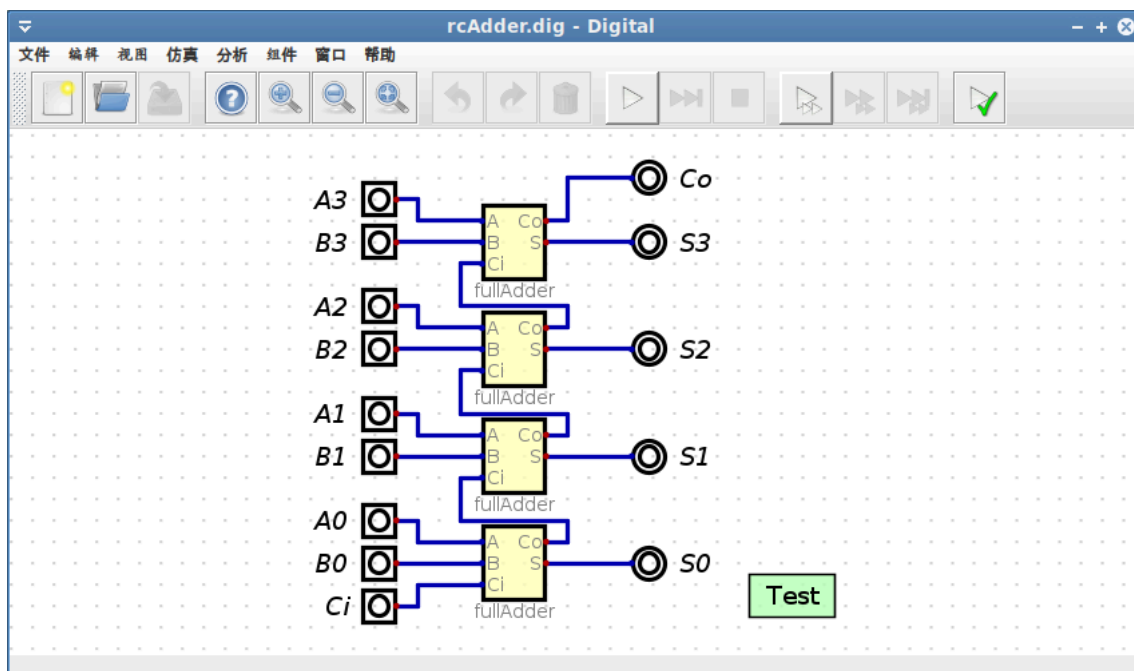
使用两个半加器可以构建一个全加器。创建一个空文件并保存为“fullAdder.dig”，将其放在和半加器相同的路径下。这时，可以通过菜单“组件”→“自定义”将半加器添加进电路。半加器的管脚顺序可以在半加器电路中通过菜单“编辑”→“排序输入信号”或者“编辑”→“排序输出信号”重新排序。全加器实现将位‘A’，‘B’，‘Ci’相加，结果给到输出‘S’和‘Co’。



为了检验全加器正常工作，我们需要添加一个测试用例。在测试用例中存储满足电路功能的真值表，通过这种方式，可以自动测试电路功能是否正确。

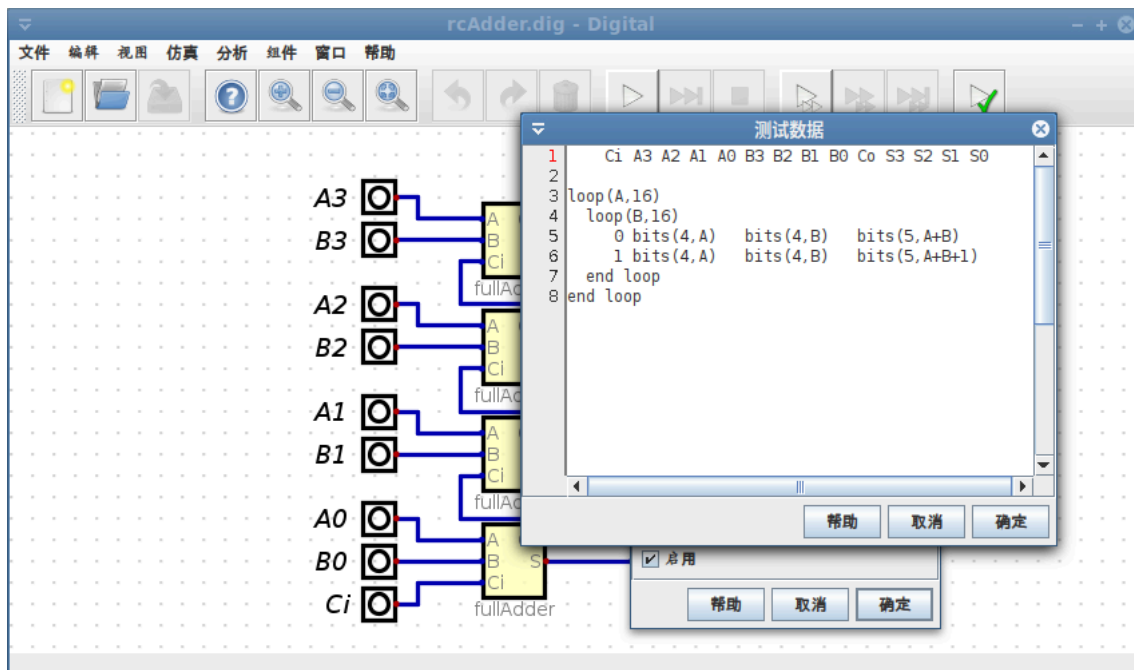


可以通过测试用例编辑器或工具栏中的测试按钮执行测试。绿色高亮的单元格表示电路的输出和测试用例中的真值表相匹配。



现在可以通过全加器进一步构建逐位（行波）进位加法器；。在该示例中，前一个加法器的进位输出作为下一级加法器的进位输入，就像使用纸和笔进行加法一样。为了测试该 4 位加法器的功能，我们添加了一个测试用例。





该测试用例执行 100% 测试，这对于相对简单的电路是可能的。在这里，所有的512种输入组合被应用于电路，同时检测其输出是否正确。第一行列出输入和输出信号，下面的内容应用输入值并检测其对应的输出，像真值表一样。在该示例中，需要512行。然而这样输入即繁琐又容易出错，如果能够自动生成所需要的行，则更容易和可靠。为此，变量 "A" 和 "B" 分别被分别从0到15进行遍历。对应的值被赋值给 'A[n]' 和 'B[n]'。然后检测电路输出 是否为 "A+B"。然后再次检测当设置进位时，即 "A+B+1" 的对应结果。详细的测试语法可以通过帮助对话框查看。

如果一个电路被嵌入另外一个电路，仅仅子电路的名字被保存在父电路而不是子电路本身。因此，在仿真时，必须可以通过文件系统找到对应的子电路。为了避免复杂的导入路径管理，使用以下导入策略。

仅仅子电路的文件名而不是完整的文件路径被存储。如果需要打开某个子电路，需要在所有的子文件夹下寻找。该过程仅依赖文件名称而不是完整路径，当在不同路径下有多个同名文件时，将会给出错误信息。

一个合理的项目结构看起来应当是这样的：根电路放在独立的文件夹下，所有导入的子电路必须放在相同和子文件夹下。所有的电路必须不能重名，即使位于不同的文件夹内。

## 2. 仿真

### 2.1. 传输延迟

在仿真时，所有的逻辑门均有一个传输延迟。所有的组件具有相同的传输延迟而与其复杂度无关。因此，与门和乘法器具有相同的传输延迟。特例是二极管，开关和分裂器，这些组件没有任何延迟。

如果需要仿真一个组件如乘法器更长的传输延迟，必须在乘法器的输出右侧添加一个延迟组件。

如果一个电路被包含进另一个父电路，被包含的电路保持其时序行为。被包含的电路同将其所有组件直接插入父电路行为一致。

### 3. 分析和综合电路

可以通过“分析”菜单来分析电路。对于纯组合电路，会生成对应的真值表。可以编辑该真值表，然后从编辑后的真值表生成新的电路。

分析和生成时序电路也是可能的。此时，会生成状态转换表而不是简单的真值表。每个触发器在状态转换表的输入和输出侧都会显示。在该表中，右侧为下一个状态，其会在下一个时钟信号产生。下一个状态依赖左侧触发器的当前状态。为了能够分析电路，必须为触发器命名。

右侧的下一个状态通过小写的‘n+1’表示，对应的当前状态通过附加‘n’表示。如果状态变量为‘A’，‘An’表示当前状态，‘An+1’表示下一个状态。在一个真值表中，如果信号的名称以此方式命名，则被认为时状态转换表，会生成时序电路而不是组合电路。

需要注意的是，需要分析的电路仅可包含纯组合电路和内建的 D、JK 触发器。如果一个触发器由用户自己构建，如使用或非门组成，则该电路不被认为触发器，因此不可以分析这种电路。

### 4. 硬件

#### 4.1. GAL16v8 和 GAL22v10

在真值表对话框电路生成菜单中，有生成 JEDEC 文件的功能。这是一种描述 PLD 配置图（fuse map）的特殊文件格式。该 JEDEC 文件可以通过特殊的编程器烧写进对应的 PLD 设备。目前，支持 "GAL16v8" 和 "GAL22v10" 或其兼容的设备。

#### 4.2. ATF150xAS

"ATF150x" 系列芯片是简单的 CPLDs，其最多有128个宏单元。具有 PLCC 封装。这样，如果一个 IC 在实验中被毁坏，那么其可以被简单的替换。另外，通过 "ATDH1150USB" 廉价和易用的编程器，可以使用 JTAG 接口编程 "ATF150x" 芯片。一个合适的评估开发板为 ("ATF15XX-DK3-U")，编程所需要的软件 "ATMISP"，通过 ATMEL/Microchip 网站提供。

#### 4.3. 导出 VHDL 或 Verilog

电路可以导出为 VHDL 或 Verilog，导出文件包含电路的完整描述。生成的 VHDL 代码通过了 Xilinx Vivado 和开源 VHDL 仿真器 ghdl 的测试。Verilog 代码通过了仿真器 Icarus Verilog 的测试。

如果电路包含测试用例，测试数据用于生成对应的 HDL 测试平台（test bench），可用于在 HDL 仿真时检测电路功能的正确性。

对于一些开发板，会为其生成额外所需的文件。目前，支持 BASYS3 和 Mimas 开发板 Mimas 和 Mimas V2。包含管脚分配的约束文件将会被创建。可以在开发板的数据手册中查找其管脚描述，并为输入和输出组件分配管脚编号。

对于 BASYS3 板子，如果电路的时钟频率比较低，在生成的 HDL 代码中将包含一个分频器，从而合理的对板载的时钟进行分频。如果电路的时钟频率超过 4.7MHz，则使用 Artix-7 的 MMCM 单元用于生成时钟。

为了创建 HDL 需要的约束文件，必须在设置中配置对应的开发板。在字段“工具链配置”中，可以设置对应的 XML 文件。在 "examples/hdl" 目录下，可以找到以 ".config" 为后缀的可用配置文件。如果配置文件正确，会在主菜单右侧显示对应的功能菜单。

## 5. 自定义外观

尽管 **Digital** 有些内建的选项可以设置子电路的外观，但在某些时候，需要使用更特殊的形状来显示子电路。如当表示处理器中的 **ALU** 时，本节解释如何为电路定义一个特殊的形状。

**Digital** 没有提供一个用例创建特殊形状的编辑器。为了创建电路形状，需要一些特殊步骤：首先，打开需要创建特殊外观形状的电路，然后为该电路创建一个 **SVG** 模板，在该模板中，电路表示为一个简单矩形，另外还包含电路中所有的管脚，其中输入使用蓝色圆圈表示，输出使用红色圆圈表示。可以通过查看圆圈的 **ID** 对象属性了解该圆圈属于哪个管脚。ID 格式为 "pin:[name]" 或 "pin+:[name]"。对于后者，当导入 **Digital** 时，管脚将被包含一个标签。

**SVG** 文件可以被编辑，如使用最常用的开源编辑器 **Inkscape**。管脚可以自由移动，但在重新导入 **Digital** 时，其被替换为最近的栅格坐标位置。

当导入 **Digital** 后，所有的信息将被提取和保存在电路文件中，**SVG** 文件不再需要。最后提示：**SVG** 是一种非常强大和灵活的文件格式。可用于描述非常复杂的图形。**Digital** 并不能导入所有可能的 **SVG** 文件。如果一个文件不能被导入，可能会出现异常。

## 6. 通用电路

有这样一种情形，一个子电路用于多种变形。如需要一个不同位宽的计数器，分别用于创建一个 4,5,6 位宽的电路，这样，将来维护时会非常困难。因为我们必须维护多个子电路，然而这些子电路除了一个位宽参数，其它都相同。

为解决该问题，我们可以创建一个通用参数化的部分电路。为此，电路设置中的“通用电路”复选框必须被选中，然后每个组件的属性对话框包含一个额外的字段“通用参数”。在该字段中，可以输入程序代码用来改变组件的参数。每个参数须有一个名字，可以作为字段“this”的属性被修改。可用参数的名称可以通过组件的帮助对话框查看。如果我们需要改变加法器的位宽，可以通过 "this.Bits=int(1);" 实现，这里的常量 1 始终是 "long" 类型，但位宽是 "int"，因此必须使用类型转换。

此时，电路仍然没有被参数化。这种情形是必要的，即当电路实际被使用时给出参数。这可以通过 "args" 字段来实现。如果我们想在外部设置位宽，代码可以这样 "this.Bits=int(args.bitWidth);"。参数的名称-这里为 "bitWidth" 是任意的。当实际使用该电路时，需要设置参数 "bitWidth"：。

当使用电路时，打开子电路的属性对话框，其也有一个“通用参数”字段。这里可以通过输入 "bitWidth:=5;" 来实现设置位宽。

"examples/generic" 文件夹包含一个格雷码计数器的例子，其位宽可以被配置。

## 7. 使用脚本控制测试

如果学生通过 **Digital** 来完成测验，能够自动测试学生提交的电路将会非常有帮助。为实现这种测试，可以通过命令行启动 **Digital**：

```
java -cp Digital.jar CLI test [file to test] [-tests [optional file with test cases]]
```

如果仅指定一个文件，则该文件中的测试用例将被执行。

如果指定了第二个文件，则使用第二个文件中的测试用例和第一个文件中的电路进行测试。第二个文件中的电路将被忽略。

此时，待测电路和测试用例中的电路输入输出信号名称必须相同。

## 8. 常见问题

如何移动导线？

选择其中一个端点或通过 **CTRL**+单击选择导线，移动鼠标。

如何删除导线？

选择其中一个端点或通过 **CTRL**+单击选择导线，然后按 "DEL" 键。

如何移动组件，包括其所连接的导线？

框选整个组件

当组件管脚放在导线上时没有连接

仅当导线的端点位于管脚时才会连接

如果管脚的名字比较长，当作为子电路时，管脚名字无法阅读

通过菜单 "编辑 → 设置当前电路" 修改组件的宽度

修改子电路管脚的顺序

菜单 "编辑 → 排序输入信号" 或 "编辑 → 排序输出信号"

当开始仿真时，导线颜色变为灰色

鲜绿色表示高电平，深绿色表示低电平，灰色表示高阻。

我有一个真值表，如何计算最小布尔表达式？

通过菜单 "分析" 选择 "综合"，然后输入真值表。在窗口底部，你会发现匹配的布尔表达式。

我输入真值表后，显示多个布尔表达式，哪个是正确的？

化简一个布尔表达式可能出现多个结果，它们描述同样的功能

我有一个真值表，如何创建对应的电路？

通过菜单 "分析" 选择 "综合"，然后输入真值表。使用菜单 "创建" "电路" 可以创建对应的电路。

如何编辑真值表中信号的名称？

在表头中右击信号名称

如何根据表达式创建电路？

通过菜单 "分析" "表达式"，输入表达式。

如何通过表达式创建真值表？

菜单 "分析" "表达式" 输入表达式，创建电路。然后通过菜单 "分析" "分析" 创建真值表。

我创建了一个电路，想把它用于很多电路，如何才能不重复的复制到不同的文件夹下？

放在 "lib" 文件夹内

## 9. 快捷键

<b>Space</b>	开启或停止仿真
<b>F6</b>	打开测量表对话框
<b>F7</b>	运行至中断
<b>F8</b>	执行测试用例
<b>C</b>	时钟步进（仅在仿真模式且只有一个时钟组件时可用）
<b>V</b>	单门步进
<b>B</b>	执行所有单门步骤，直至中断或完成
<b>F9</b>	分析电路
<b>CTRL-A</b>	选择所有
<b>CTRL-X</b>	剪切
<b>CTRL-C</b>	复制
<b>CTRL-V</b>	从剪贴板插入
<b>CTRL-D</b>	复制当前选中而不更改剪贴板
<b>R</b>	旋转组件
<b>L</b>	插入最近一次插入的组件
<b>T</b>	插入隧道组件
<b>CTRL-N</b>	新建电路
<b>CTRL-O</b>	打开电路
<b>CTRL-S</b>	保存电路
<b>CTRL-Z</b>	撤销上次修改
<b>CTRL-Y</b>	重做上次撤销操作
<b>P</b>	对二极管或浮动栅场效应管编程
<b>D</b>	画导线时使用对角线模式
<b>F</b>	画线时翻转方向
<b>S</b>	分割一条导线为两条
<b>ESC</b>	放弃当前操作
<b>Del</b>	删除
<b>Backspace</b>	删除
<b>+</b>	加1
<b>-</b>	减1
<b>CTRL +</b>	放大
<b>CTRL -</b>	缩小
<b>F1</b>	适合窗口

**F5**

显示或隐藏组件树

## B 设置

以下内容为可用的仿真器设置

### 设置

全局设置，包括仿真器、界面语言、符号、外部工具等。

### 属性

使用 **IEEE 91-1984** 外观

使用 **IEEE 91-1984** 形状代替矩形

语言

图形界面的语言，需要重启才能生效

格式

表达式格式

颜色方案

颜色方案

自定义颜色

自定义颜色

应用启动时显示组件树

如果选中，应用启动时会在左侧显示组件树

显示栅格

在主窗口中显示栅格

在总线上显示导线数。

注意：值仅在仿真启动后更新。

导线提示

如果选中，当鼠标划过时导线会高亮

使用等号按键

使用等号键而不是加号键。如果加号字符不是主键，而是等号字符的第二个赋值，这总是有用的，例如 用于美式或法式键盘布局。

**Increase numbers in identifiers when copying.**

If set, in components whose label ends with a number, this number is incremented when copying.

显示自动重命名隧道组件对话框

如果选中，当一个隧道组件被重命名后，将会显示为同名隧道组件自动重命名的对话框。

库

包含预定义的子电路，还可以在当前路径下添加自定义电路。

**Java 库**

使用java实现的额外组件jar文件。

**ATF15xx Fitter**

包含Microchip(ATMEL)fit15xx.exe 文件的路径

**ATMISP**

可执行文件ATMISP.exe的路径。设置后，ATMISP 软件可以被自动启动！

**GHDL**

可执行文件ghdl路径。仅当需要使用 ghdl 仿真 vhdl 组件时设置。

**IVerilog**

Icarus verilog 安装路径

## 工具链配置

用于配置集成工具链，如启动外部工具，对 **FPGA** 进行编程

## 菜单字体大小(百分比)

菜单字体大小，相对于默认大小的百分比

## 使用 MacOS 鼠标单击

使用 **CTRL**+左键 代替右键单击。

## 允许远程连接

-如果开启，会打开一个 **TCP** 端口，可以通过该端口控制仿真器。

## 端口号

远程控制服务的端口

## 设置当前电路

该电路设置影响当前打开电路的行为，例如当电路作为子电路被其他电路使用时的形状，这些设置和电路文件保存在一起。

## 属性

### 标签

该组件的名称

### 宽度

当电路作为子电路时，电路符号的宽度

### 背景色

当嵌入其它电路时的电路背景色

### 简介

关于该组件和其使用的简短描述

### 电路被锁定

电路被锁定

### 形状

当电路作为子电路时的形状。“Simple”模式：输入管脚显示在一个简单矩形左侧，输出管脚显示在一个简单矩形右侧。“Layout”模式：管脚的位置由实际电路中输入、输出组件的位置和方向决定。此时，管脚可以位于顶部或底部。“DIL-Chip”模式：使用双列直插封装外形显示电路，输入、输出组件的管脚编号决定管脚的位置。

### 自定义形状

导入 **SVG** 文件

### 高度

当电路作为子电路时，电路符号的高度

### 管脚数

管脚数，0表示自动确定管脚数

### ROM 内容

所有使用的 **ROM** 中的内容

### 在仿真开始时显示测量值

当仿真开始时，包含测量值的表将被显示。

### 仿真启动时显示测量图

当仿真开始时，包含测量值的图形将被显示。

### 以单步模式显示测量图

当仿真启动时，以单步模式显示测量图。

### 最大显示步数

存储值的最大数，如果达到最大数，旧的值将被忽略。

### 在启动时预加载程序到存储器

当仿真一个使用 **RAM** 作为程序存储器的处理器时，因为 **RAM** 内容被初始化为零，因此很难启动处理器。该设置允许在处理器启动时加载数据到程序存储器。



**程序文件**

在仿真开始时被自动加载进存储器的程序文件

**导入时使用大端对齐**

导入时使用大端对齐

**导出 Verilog/VHDL 时忽略**

当导出 Verilog/VHDL 时，忽略内部实现，保留对电路的引用，以实现手动编写电路实现。

**通用电路**

创建一个通用电路

**振荡检测**

如果电路尚未稳定，则检测到振荡的门传播次数。

**In case of oscillations, continue with random values.**

**If this option is set, the behavior of the simulation becomes unpredictable and extremely slow if there is an oscillation!**

## C 命令行界面

```
java -cp Digital.jar CLI
```

```
test -circ [String] [-tests [String]] [-allowMissingInputs] [-verbose]:
```

第一个文件名指定将要被测的电路。如果指定第二个文件名，则执行该文件中的测试用例。

选项

```
-circ [String(def: )]
```

待测文件的文件名

```
[-tests [String(def: )]]
```

测试用例文件的文件名

```
[-allowMissingInputs(def: false)]
```

允许电路中缺少测试用例中定义的输入。如果有几种可能的解决方案可能取决于不同的输入，这将很有用。

```
[-verbose(def: false)]
```

如果设置，当发生错误时，输出相关值的表格。

```
svg -dig [String] [-svg [String]] [-ieee] [-LaTeX] [-pinsInMathMode] [-hideTest] [-noShapeFilling] [-smallIO] [-noPinMarker] [-thinnerLines] [-highContrast] [-monochrome]:
```

根据电路创建 SVG 文件

选项

```
-dig [String(def: )]
```

电路文件名

```
[-svg [String(def: )]]
```

SVG 文件名

```
[-ieee(def: false)]
```

使用 IEEE 符号

```
[-LaTeX(def: false)]
```

使用 LaTeX 格式表示文本

```
[-pinsInMathMode(def: false)]
```

即使不包含索引，仍然使用数学模式命名管脚标签

```
[-hideTest(def: false)]
```

隐藏测试用例

```
[-noShapeFilling(def: false)]
```

多边形不会被填充

```
[-smallIO(def: false)]
```

输入和输出表示为小圆圈。

`[-noPinMarker(def: false)]`

符号中蓝色和红色的管脚标注会被忽略。

`[-thinnerLines(def: false)]`

如果选中，绘制的线会比较细

`[-highContrast(def: false)]`

导线和管脚的文本显示为黑色。

`[-monochrome(def: false)]`

仅使用灰色。

`stats -dig [String] [-csv [String]]:`

创建包含电路统计信息的 CSV 文件。

选项

`-dig [String(def: )]`

电路文件名

`[-csv [String(def: )]]`

CSV 文件名，如果为空，将写入到标准输出。

`run -dig [String]:`

Runs a circuit headless.

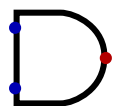
选项

`-dig [String(def: )]`

File name of the circuit.

## D 组件

### 1. 逻辑



#### 1.1. 与门

二进制与门，仅当所有输入为 **1** 时输出为 **1**。其输入和输出也可以使用多位总线，此时将执行按位与运算。可以导出为 VHDL/Verilog。

输入

`In_1`

逻辑运算输入 1

`In_2`

逻辑运算输入 2

输出

`out`

逻辑运算结果

属性

数据位数

数据线位数

输入端口数

输入端口个数，所有的输入端口都必须被连接。

翻转输入

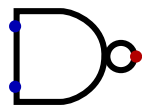
选择需要翻转的输入信号

旋转

组件在电路中的旋转角度

宽形

使用较宽的形状显示门



#### 1.2. 与非门

与和非的组合。仅当所有输入为 **1** 时，输出为 **0**。其输入也可以使用多位总线，此时将先执行按位与运算，然后执行逻辑非运算。可以导出为 VHDL/Verilog。

输入

`In_1`

逻辑运算输入 1

`In_2`

逻辑运算输入 2

输出

out

逻辑运算结果

属性

数据位数

数据线位数

输入端口数

输入端口个数，所有的输入端口都必须被连接。

翻转输入

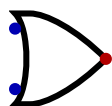
选择需要翻转的输入信号

旋转

组件在电路中的旋转角度

宽形

使用较宽的形状显示门



### 1.3. 或门

二进制或门，如果其中一个输入为 1 则输出 1。其输入和输出也可以使用多位总线，此时将执行按位或运算。可以导出为 VHDL/Verilog。

输入

In\_1

逻辑运算输入 1

In\_2

逻辑运算输入 2

输出

out

逻辑运算结果

属性

数据位数

数据线位数

输入端口数

输入端口个数，所有的输入端口都必须被连接。

翻转输入

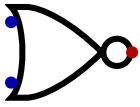
选择需要翻转的输入信号

旋转

组件在电路中的旋转角度

宽形

使用较宽的形状显示门



#### 1.4. 或非门

或门和非门的组合。如果其中一个输入为 1 则输出为 0，如果所有输入为 0，输出为 1。其输入也可以使用多位总线，此时将先执行按位或运算，然后执行逻辑非运算。可以导出为 VHDL/Verilog。

输入

`In_1`

逻辑运算输入 1

`In_2`

逻辑运算输入 2

输出

`out`

逻辑运算结果

属性

数据位数

数据线位数

输入端口数

输入端口个数，所有的输入端口都必须被连接。

翻转输入

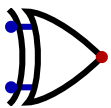
选择需要翻转的输入信号

旋转

组件在电路中的旋转角度

宽形

使用较宽的形状显示门



#### 1.5. 异或门

如果两个输入被使用，当两个输入相等时输出为 0，否则为 1。如果多于两个输入，其行为类似于级联的异或门(A 异或 B 异或 C = (A 异或 B) 异或 C)。其输入也可以使用多位总线，此时将执行按位异或运算。可以导出为 VHDL/Verilog。

输入

`In_1`

逻辑运算输入 1

`In_2`

逻辑运算输入 2

输出

out

逻辑运算结果

属性

数据位数

数据线位数

输入端口数

输入端口个数，所有的输入端口都必须被连接。

翻转输入

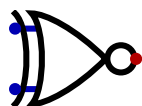
选择需要翻转的输入信号

旋转

组件在电路中的旋转角度

宽形

使用较宽的形状显示门



## 1.6. 异或非门

异或门和非门的组合。可以导出为 VHDL/Verilog。

输入

In\_1

逻辑运算输入 1

In\_2

逻辑运算输入 2

输出

out

逻辑运算结果

属性

数据位数

数据线位数

输入端口数

输入端口个数，所有的输入端口都必须被连接。

翻转输入

选择需要翻转的输入信号

旋转

组件在电路中的旋转角度

宽形

使用较宽的形状显示门



### 1.7. 非门

翻转输入，1 变为 0,0变为 1。其输入和输出也可以使用多位总线，此时将执行按位非运算。可以导出为 VHDL/Verilog。

输入

in

非门的输入

输出

out

翻转后的输入值

属性

数据位数

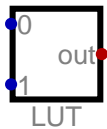
数据线位数

旋转

组件在电路中的旋转角度

宽形

使用较宽的形状显示门



### 1.8. 查找表

从一张存储表中获取输出，该组件可以模拟任何组合电路。可以导出为 VHDL/Verilog。

输入

0

输入 0，和其他输入共同定义将要返回的所存储内容地址。

1

输入 1，和其他输入共同定义将要返回的所存储内容地址。

输出

out

根据指定的输入组合，返回对应的存储值。

属性

数据位数

数据线位数

输入端口数

输入端口个数，所有的输入端口都必须被连接。

标签

该组件的名称



数据

存储在该组件中的值

旋转

组件在电路中的旋转角度

## 2. 输入输出



### 2.1. 输出

用于在电路中显示输出信号。也用于连接子电路。当用于生成 CPLD 或 FPGA 代码时，用于为其赋值管脚编号。可以导出为 VHDL/Verilog。

输入

in

用于连接输出

属性

数据位数

数据线位数

标签

该组件的名称

简介

关于该组件和其使用的简短描述

旋转

组件在电路中的旋转角度

数字格式

显示数字的格式

定点数

小数位数

管脚编号

管脚的编号，用于表示电路的 DIL 封装和生成 CPLD 代码时分配管脚。如果具有多位，所有的管脚编号可以表示为逗号分割列表。

在测量图中显示

在测量图中显示值

较小符号

如果选中，将使用较小的符号形状。



### 2.2. LED

用于可视化输出信号，仅接受 1 位值，当输入为 1 时灯被点亮。

输入

in

LED 输入，仅接受 1 位值，当输入为 1 时灯被点亮。

## 属性

## 标签

该组件的名称

## 颜色

组件的颜色

## 旋转

组件在电路中的旋转角度

## 大小

形状大小

## 在测量图中显示

在测量图中显示值



## 2.3. 输入

可以通过鼠标交互操作输入信号。也用于连接子电路。当用于生成 CPLD 或 FPGA 代码时，用于为其赋值管脚编号。可以导出为 VHDL/Verilog。

## 输出

## out

给出连接到该输入的值

## 属性

## 数据位数

数据线位数

## 标签

该组件的名称

## 简介

关于该组件和其使用的简短描述

## 旋转

组件在电路中的旋转角度

## 默认值

电路仿真启动时的默认值，“Z”表示高阻状态。

## 允许三态输入

如果勾选，则输入信号允许高阻状态，其控制输入组件的默认值是否可以设为 "Z"

## 非零输出

避免输出0。用于设置延迟电路。仅当允许高阻态输出时可用。

## 数字格式

显示数字的格式

## 定点数

小数位数

## 管脚编号

管脚的编号，用于表示电路的 DIL 封装和生成 CPLD 代码时分配管脚。如果具有多位，所有的管脚编号可以表示为逗号分割列表。

## 在测量图中显示

在测量图中显示值

## 较小符号

如果选中，将使用较小的符号形状。



## 2.4. 时钟输入

时钟信号，可以通过实时时钟进行控制。如果频率大于50赫兹，导线的颜色将不会更新。如果没有使用实时时钟，可以通过鼠标单击控制时钟信号。当生成 CPLD 或 FPGA 代码时，用于为其赋值管脚编号。可以导出为 VHDL/Verilog。

输出

C

根据设置的时钟频率在0和1间切换

属性

标签

该组件的名称

使用实时时钟

如果选中，当电路开始模拟时将使用实时时钟

频率/赫兹

实时时钟频率

旋转

组件在电路中的旋转角度

管脚编号

管脚的编号，用于表示电路的 DIL 封装和生成 CPLD 代码时分配管脚。如果具有多位，所有的管脚编号可以表示为逗号分割列表。

较小符号

如果选中，将使用较小的符号形状。



## 2.5. 按钮

一个简单按键，当释放时回到其初始的状态

输出

out

按钮输出信号

属性

标签

该组件的名称

低电平有效

如果选中该选项，则该组件处于活动状态时输出为低电平。

映射到键盘

按钮被映射的键盘。使用 UP, DOWN, LEFT 或者 RIGHT 作为标签以使用方向键。

旋转

组件在电路中的旋转角度

在测量图中显示

在测量图中显示值



## 2.6. DIP开关

简单的DIP开关，可以输出高或低电平  
输出

out

开关的输出值

属性

标签

该组件的名称

简介

关于该组件和其使用的简短描述

旋转

组件在电路中的旋转角度

输出为高电平

默认输出



## 2.7. 探测器

测量值可以通过数据图或测量表显示，该组件可用于观察子电路中的值。

输入

in

需要测量的值

属性

标签

该组件的名称

显示模式

定义是否显示值或者计数器。

旋转

组件在电路中的旋转角度

数字格式

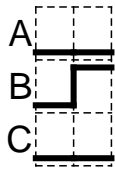
显示数字的格式

定点数

小数位数

在测量图中显示

在测量图中显示值



## 2.8. 数据图

在电路面板显示数据绘图，可以绘制完整的时钟周期或单个门改变，该组件不影响仿真。

属性

显示单步

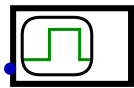
在图形中显示所有单步

最大显示步数

存储值的最大数，如果达到最大数，旧的值将被忽略。

对齐到栅格

如果选中，组件将会和栅格对齐。



## 2.9. 示波器

显示可测量值的变化图，仅当电路稳定且该组件的输入信号变化时才会对测量数据采样。

输入

T

当输入的信号发生变化时对测量数据采样

属性

标签

该组件的名称

触发

触发条件

最大显示步数

存储值的最大数，如果达到最大数，旧的值将被忽略。

## 3. 输入输出 - 显示



### 3.1. RGB-LED

RGB LED，其颜色通过3个输入信号控制，每个输入表示一个颜色通道。

输入

- R  
红色通道
- G  
绿色通道
- B  
蓝色通道

属性

- 数据位数
- 数据线位数
- 标签  
该组件的名称
- 旋转  
组件在电路中的旋转角度
- 大小  
形状大小



3.2. 发光二极管

当阳极连接到高电平且阴极连接到低电平时，LED点亮。该LED不能用作下拉电阻。它仅充当显示元素。图中所示电阻仅表示需要串联一个电阻来限流。

输入

- A  
阳极
- C  
阴极

属性

- 标签  
该组件的名称
- 颜色  
组件的颜色
- 旋转  
组件在电路中的旋转角度
- 在测量图中显示  
在测量图中显示值



### 3.3. 带有LED的按钮

一个简单按键，当释放时回到其初始的状态。按键有一个LED，可以通过输入信号进行开关。

输入

in

控制LED的输入信号

输出

out

按键输出信号

属性

标签

该组件的名称

低电平有效

如果选中该选项，则该组件处于活动状态时输出为低电平。

映射到键盘

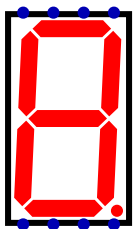
按钮被映射的键盘。使用 UP, DOWN, LEFT 或者 RIGHT 作为标签以使用方向键。

颜色

组件的颜色

旋转

组件在电路中的旋转角度



### 3.4. 7段数码管

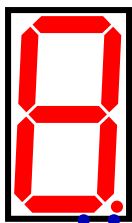
每段都有自己的控制输入

## 输入

- a 该输入控制顶部，水平线
- b 该输入控制顶部，右侧，垂直线
- c 该输入控制底部，右侧，垂直线
- d 该输入控制底部，水平线
- e 该输入控制底部，左侧，垂直线
- f 该输入控制顶部，左侧，垂直线
- g 该输入控制中间，水平线
- dp 该输入控制小数点

## 属性

- 颜色
  - 组件的颜色
- 共极性
  - 如果设置，将会仿真共阴极或共阳极
- 共
  - 共极性
- 视觉暂留
  - 设置余辉的持续时间。数值越大，余辉持续时间越长。



### 3.5. 7段数码管(十六进制输入)

#### 4位十六进制输入7段数码管

## 输入

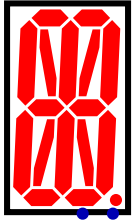
- d 将被显示的输入值
- dp 该输入控制小数点

## 属性

- 颜色
  - 组件的颜色



大小  
形状大小



### 3.6. 16段数码管

通过16位输入控制各段，第二个输入控制小数点  
输入

led  
16位驱动总线  
dp  
该输入控制小数点

属性

颜色  
组件的颜色  
大小  
形状大小



### 3.7. 灯泡

灯泡具有两个连接点，当电流经过时灯泡变亮。不关心电流的方向。当输入为不同的值时点亮，行为类似异或门。

输入

A  
连接点  
B  
连接点

属性

标签  
该组件的名称  
颜色  
组件的颜色  
旋转  
组件在电路中的旋转角度



### 3.8. LED矩阵

LED矩阵，在独立窗口中显示。通过一个数据字段控制一列LED的开关，另外一个输入为当前列的地址。

输入

**r-data**

当前列对应的数据位字，自底向上

**c-addr**

当前列地址

属性

标签

该组件的名称

行数

行数

列地址位数

单个列的地址，3位表示8列。

颜色

组件的颜色

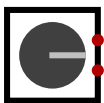
避免闪烁

不能将频率增加得太多以至于使闪烁消失。尽管如此，为了抑制闪烁，可以使用此选项为LED开启“余辉”。当启用时，即使某个引脚变为高阻，LED仍保持点亮状态。这模拟了高于临界闪烁融合频率的频率。

旋转

组件在电路中的旋转角度

## 4. 输入输出 - 机械



### 4.1. 旋转编码器

带旋转编码器的旋钮，用于检测旋转运动。

输出

**A**

信号A

**B**

信号B

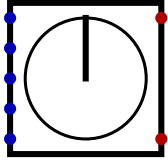
属性

标签

该组件的名称

旋转

组件在电路中的旋转角度



## 4.2. 单极性步进电机

具有两个限位开关的单极步进电机。支持全步驱动，半步驱动和细分(微步)驱动。

输入

P0

相数 0

P1

相数 1

P2

相数 2

P3

相数 3

com

共中心连接

输出

S0

限位开关0, 当电机角度为  $0^\circ$  时变为高电平

S1

限位开关1, 当电机角度为  $180^\circ$  时变为高电平

属性

标签

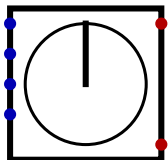
该组件的名称

翻转输出

如果选中，输出将被翻转。

旋转

组件在电路中的旋转角度



## 4.3. 双极性步进电机

具有两个限位开关的双极步进电机。支持全步驱动，半步驱动和细分(微步)驱动。

## 输入

A+

线圈 A, 正极

A-

线圈 A, 负极

B+

线圈 B, 正极

B-

线圈 B, 负极

## 输出

S0

限位开关0, 当电机角度为  $0^\circ$  时变为高电平

S1

限位开关1, 当电机角度为  $180^\circ$  时变为高电平

## 属性

标签

该组件的名称

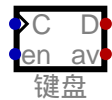
翻转输出

如果选中, 输出将被翻转。

旋转

组件在电路中的旋转角度

## 5. 输入输出 - 外设



### 5.1. 键盘

可以用于输入文本的键盘。该组件缓存输入，然后读取输入作为输出。

## 输入

C

时钟, 上升沿时从缓冲中移除旧的字符

en

如果为高电平, 则输出 **D** 有效, 输出一个字符。同时使能时钟输入。

## 输出

D

最后输入的字符或0(如果没有有效字符), 输出为**16位Java**字符值

av

该输出信号表示字符有效, 用于触发中断。

## 属性

标签

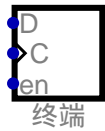
该组件的名称

翻转输入

选择需要翻转的输入信号

旋转

组件在电路中的旋转角度



## 5.2. 终端

可以向终端写入ASCII字符，该终端打开新的窗口并显示输出。

输入

D

将要写入终端的数据

C

时钟，在上升沿将输入信号的值写入终端

en

高电平使能

属性

每行字符数

每行字符数

行数

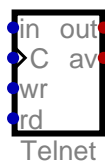
显示的行数

标签

该组件的名称

旋转

组件在电路中的旋转角度



## 5.3. Telnet

允许通过 Telnet 协议连接到电路，可以通过 Telnet 发送或接收字符。

## 输入

in

待发送数据

C

时钟输入

wr

如果置位，发送输入字节数据。

rd

如果置位，输出接收到的字节数据。

## 输出

out

数据输出

av

如果数据存在则输出 1

## 属性

标签

该组件的名称

Telnet 模式

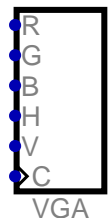
如果置位，Telnet 控制命令将会被执行。另外，服务器会发送 **SGA** 和 **ECHO** 命令。  
如果禁用该选项，服务器工作为简单的 **TCP** 服务器。

端口

服务器端口

旋转

组件在电路中的旋转角度



## 5.4. VGA显示器

分析输入视频信号并显示对应图形。因为仿真不能实时运行，除了视频信号，还需要像素时钟。

输入

- R  
红色组件
- G  
绿色组件
- B  
蓝色组件
- H  
水平同步信号
- V  
垂直同步信号
- C  
像素时钟

属性

- 标签  
该组件的名称
- 旋转  
组件在电路中的旋转角度



5.5. MIDI

使用 MIDI 系统播放记录

输入

- N  
记录
- V  
音量
- OnOff  
为1相当于按下按键，为0相当于释放按键。
- en  
使能
- C  
时钟

属性

- 标签  
该组件的名称
- MIDI通道  
选择需要使用的 MIDI通道
- MIDI 设备  
选择需要使用的 MIDI 设备

允许改变程序

增加一个新的输入端口 (PC)，如果该输入为高电平，输入端口 **N** 的值用于改变程序 (设备)

旋转

组件在电路中的旋转角度

## 6. 导线



### 6.1. 地

用于接地，输出始终为 0 可以导出为 VHDL/Verilog。

输出

out

输出始终为 0

属性

数据位数

数据线位数

标签

该组件的名称

旋转

组件在电路中的旋转角度



### 6.2. 电源

用于连接电源，输出始终为 1 可以导出为 VHDL/Verilog。

输出

out

输出始终为 1

属性

数据位数

数据线位数

标签

该组件的名称

旋转

组件在电路中的旋转角度

### 1•

### 6.3. 常量

常量，值在属性对话框设置 可以导出为 VHDL/Verilog。



输出

out

返回常量设置值

属性

数据位数

数据线位数

值

常量的值

旋转

组件在电路中的旋转角度

数字格式

显示数字的格式

定点数

小数位数



## 6.4. 隧道

不使用导线实现组件连接。所有具有相同名称的隧道网络被认为连接一起。 仅在当前电路有效，不能用于不同电路。没有名称的隧道组件被忽略。 可以导出为 VHDL/Verilog。

输入

in

连接点

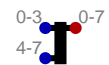
属性

网络名称

所有具有相同名称的网络连在一起。

旋转

组件在电路中的旋转角度



## 6.5. 分裂器/合并器

分裂信号或创建总线。 可以导出为 VHDL/Verilog。

输入

0-3

输入位 0-3

4-7

输入位 4-7

输出

0-7

输出位 0-7

## 属性

## 输入分割

假设 4 位, 2 位, 2 位作为输入, 可表示为 "4, 2, 2"。还可以使用 \* 号, 如 16 位表示为 "1\*16"([Bits]\*[Number])。还可以直接指定使用的位而不用关心顺序, 如 "4-7, 0-3"。输入位必须明确且完整的指定。

## 输出分割

假设 4 位, 2 位, 2 位作为输出, 可表示为 "4, 2, 2"。还可以使用 \* 号, 如 16 位表示为 "1\*16"([Bits]\*[Number])。还可以直接指定使用的位而不用关心顺序, 如 "4-7, 0-3"。输出位可以使用多次, 如 "0-7, 1-6, 4-7"。

## 旋转

组件在电路中的旋转角度

## 镜像

镜像组件

## 间距

设置输入和输出端口间距



## 6.6. 驱动器

用于将信号连接到其它导线。如果 **sel** 输入为低电平, 输出为高阻状态, 如果 **sel** 为高电平, 输出等于输入。可以导出为 VHDL/Verilog。

## 输入

**in**

输入

**sel**

控制位

## 输出

**out**

输出

## 属性

## 数据位数

数据线位数

## 翻转输出

如果选中, 输出将被翻转。

## 翻转选择管脚位置

将选择管脚移到对面的位置

## 旋转

组件在电路中的旋转角度



### 6.7. 驱动器 (低电平有效)

用于将信号连接到其它导线。如果 **sel** 输入为高电平，输出为高阻状态，如果 **sel** 为低电平，输出等于输入。可以导出为 VHDL/Verilog。

输入

**in**

输入

**sel**

控制位

输出

**out**

输出

属性

数据位数

数据线位数

翻转输出

如果选中，输出将被翻转。

翻转选择管脚位置

将选择管脚移到对面的位置

旋转

组件在电路中的旋转角度



### 6.8. 延迟

延迟信号传输。Digital 中的所有组件具有相同的传输延迟单位，该组件用于实现必要的传输延迟。

输入

**in**

将要延迟的输入信号

输出

**out**

经过延迟的输入信号 (默认一个延迟单位)

属性

数据位数

数据线位数

时长

延迟时间，单位为常见门的传输延迟。

旋转  
组件在电路中的旋转角度



### 6.9. 上拉电阻

如果一个网络为高阻态，该电阻将网络上拉到高电平，其它时候该组件无效。

输出

out  
弱高电平

属性

数据位数  
数据线位数  
旋转  
组件在电路中的旋转角度



### 6.10. 下拉电阻

如果一个网络为高阻态，该电阻将网络下拉到低电平，其它时候该组件无效。

输出

out  
弱低电平

属性

数据位数  
数据线位数  
旋转  
组件在电路中的旋转角度



### 6.11. 无连接

该组件可用于设置导线为高阻态。如果逻辑门的输入被设置为高阻态，则输出随机。请注意，实际中，如果数字输入未设置为高电平或低电平，而是保持未连接状态，则可能会产生过多的电流消耗甚至损坏。

输出

out  
输出高阻

属性

数据位数  
数据线位数

## 7. 复用器



### 7.1. 复用器

根据 **sel** 端口选择哪个输入可以通过 可以导出为 VHDL/Verilog。

输入

**sel**  
选择端口  
**in\_0**  
输入端口 0  
**in\_1**  
输入端口 1

输出

**out**  
输出

属性

数据位数  
数据线位数  
选择位  
选择位位数  
翻转选择管脚位置  
将选择管脚移到对面的位置  
旋转  
组件在电路中的旋转角度



### 7.2. 多路分配器

根据 **sel** 将输入值给到某个输出端口 可以导出为 VHDL/Verilog。

输入

**sel**  
选择端口  
**in**  
输入端口

输出

out\_0  
输出端口 0  
out\_1  
输出端口 1

属性

数据位数  
数据线位数  
选择位  
选择位位数  
翻转选择管脚位置  
将选择管脚移到对面的位置  
旋转  
组件在电路中的旋转角度  
默认  
电路仿真开始时的默认值。



### 7.3. 解码器

仅有一个选中的输出端口为高电平，其它输出为低电平。可以导出为 VHDL/Verilog。

输入

sel  
选择端口

输出

out\_0  
输出端口 0  
out\_1  
输出端口 1

属性

选择位  
选择位位数  
翻转选择管脚位置  
将选择管脚移到对面的位置  
旋转  
组件在电路中的旋转角度



#### 7.4. 位选择器

在数据总线中选择某一位 可以导出为 VHDL/Verilog。

输入

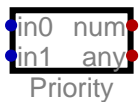
**in**  
输入总线  
**sel**  
选择端口

输出

**out**  
输出端口

属性

选择位  
选择位位数  
翻转选择管脚位置  
将选择管脚移到对面的位置  
旋转  
组件在电路中的旋转角度



#### 7.5. 优先级编码器

当输入只有一个为高电平时，其对应的值被输出。当多个输入为高电平时，输出对应最大的值。可以导出为 VHDL/Verilog。

输入

**in0**  
输入信号 0  
**in1**  
输入信号 1

输出

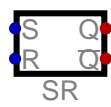
**num**  
输入信号对应的值  
**any**  
如果该端口为高电平，则至少有一个输入为高电平

属性

标签  
该组件的名称

- 选择位
  - 选择位位数
- 旋转
  - 组件在电路中的旋转角度

8. 触发器



8.1. SR 触发器

用于存储 1 位数据。通过置位和复位来设置存储的数据。 如果输入同时为高电平，输出全为低电平。如果输入同时为低电平，输出状态随机。

输入

- S
  - 置位
- R
  - 复位

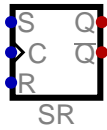
输出

- Q
  - 返回存储的值
- $\neg Q$ 
  - 返回翻转后的存储值

属性

- 标签
  - 该组件的名称
- 翻转输入
  - 选择需要翻转的输入信号
- 旋转
  - 组件在电路中的旋转角度
- 镜像
  - 镜像组件
- 默认
  - 电路仿真开始时的默认值。
- 作为测量值
  - 如果选中，该值将作为测量值在测量图和数据表中显示。此时，必须指定标签作为值的标识。





### 8.2. SR 触发器 (时钟控制)

用于存储 1 位数据。通过置位和复位来设置存储的数据。如果输入在时钟上升沿同时为低电平，输出状态随机。

输入

S

置位

C

时钟输入，在上升沿状态转换

R

复位

输出

Q

返回存储的值

$\neg Q$

返回翻转后的存储值

属性

标签

该组件的名称

翻转输入

选择需要翻转的输入信号

旋转

组件在电路中的旋转角度

镜像

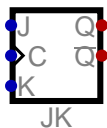
镜像组件

默认

电路仿真开始时的默认值。

作为测量值

如果选中，该值将作为测量值在测量图和数据表中显示。此时，必须指定标签作为值的标识。



### 8.3. JK 触发器

可以存储( $J=K=0$ )，置位( $J=1, K=0$ )，复位( $J=0, K=1$ )或翻转( $J=K=1$ )存储的内容。状态转换仅在时钟上升沿发生。可以导出为 VHDL/Verilog。

## 输入

J

置位

C

时钟输入，在上升沿状态转换

K

复位

## 输出

Q

返回存储的值

 $\neg Q$ 

返回翻转后的存储值

## 属性

标签

该组件的名称

翻转输入

选择需要翻转的输入信号

旋转

组件在电路中的旋转角度

镜像

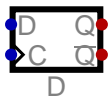
镜像组件

默认

电路仿真开始时的默认值。

作为测量值

如果选中，该值将作为测量值在测量图和数据表中显示。此时，必须指定标签作为值的标识。

**8.4. D 触发器**

管脚 **D** 的值在时钟上升沿被存储。可以设置位宽以允许存储多位数据。可以导出为 VHDL/Verilog。

## 输入

D

输入

C

时钟

## 输出

Q

返回存储的值

 $\neg Q$ 

返回翻转后的存储值

## 属性

数据位数

数据线位数

标签

该组件的名称

翻转输入

选择需要翻转的输入信号

旋转

组件在电路中的旋转角度

镜像

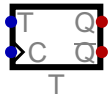
镜像组件

默认

电路仿真开始时的默认值。

作为测量值

如果选中，该值将作为测量值在测量图和数据表中显示。此时，必须指定标签作为值的标识。

**8.5. T 触发器**

存储一位，在时钟上升沿切换状态。

## 输入

T

使能状态切换

C

时钟输入

## 输出

Q

返回存储的值

 $\neg Q$ 

返回翻转后的存储值

## 属性

标签

该组件的名称

允许输入

如果设置，将使能输入端口 T

翻转输入

选择需要翻转的输入信号

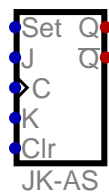
旋转

组件在电路中的旋转角度

镜像

镜像组件

- 默认
  - 电路仿真开始时的默认值。
- 作为测量值
  - 如果选中，该值将作为测量值在测量图和数据表中显示。此时，必须指定标签作为值的标识。



8.6. JK 触发器(异步)

能够存储(J=K=0), 置位(J=1, K=0), 复位(J=0, K=1)或翻转(J=K=1)存储的内容。 状态仅能够在时钟上升沿发生改变。 另有两个额外的输入可以立即改变状态而不需要时钟信号。 可以导出为 VHDL/Verilog。

输入

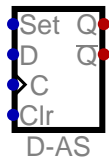
- Set
  - 异步置位，输入高电平将置位触发器
- J
  - 置位
- C
  - 时钟输入
- K
  - 复位
- Clr
  - 异步复位，输入高电平将复位触发器

输出

- Q
  - 返回存储的值
- ¬Q
  - 返回翻转后的存储值

属性

- 标签
  - 该组件的名称
- 翻转输入
  - 选择需要翻转的输入信号
- 旋转
  - 组件在电路中的旋转角度
- 镜像
  - 镜像组件
- 默认
  - 电路仿真开始时的默认值。
- 作为测量值
  - 如果选中，该值将作为测量值在测量图和数据表中显示。此时，必须指定标签作为值的标识。



### 8.7. D 触发器(异步)

管脚 **D** 的值在时钟上升沿被存储。可以设置位宽以允许存储多位数据。 另有两个额外的输入可以立即改变状态而不需要时钟信号。 可以导出为 **VHDL/Verilog**。

输入

**Set**

异步置位，输入高电平将置位触发器

**D**

输入

**C**

时钟，上升沿触发

**Clr**

异步复位，输入高电平将复位触发器

输出

**Q**

返回存储的值

**¬Q**

返回翻转后的存储值

属性

数据位数

数据线位数

标签

该组件的名称

翻转输入

选择需要翻转的输入信号

旋转

组件在电路中的旋转角度

镜像

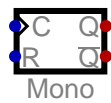
镜像组件

默认

电路仿真开始时的默认值。

作为测量值

如果选中，该值将作为测量值在测量图和数据表中显示。此时，必须指定标签作为值的标识。



8.8. 单稳态触发器

单稳态触发器(monoflop)，在时钟上升沿置位。在经过一个可配置的延迟后，自动复位。该单稳态触发器可重复触发。电路中必须存在唯一的一个时钟组件， 时钟组件作为测量时间延迟的基准。

输入

- C  
时钟输入
- R  
复位输入，高电平有效

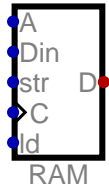
输出

- Q  
输出
- ¬Q  
翻转后的输出

属性

- 标签  
该组件的名称
- 脉冲宽度  
用时钟周期表示的脉冲宽度
- 翻转输入  
选择需要翻转的输入信号
- 旋转  
组件在电路中的旋转角度
- 镜像  
镜像组件
- 默认  
电路仿真开始时的默认值。
- 作为测量值  
如果选中，该值将作为测量值在测量图和数据表中显示。此时，必须指定标签作为值的标识。

## 9. 存储器 - RAM



### 9.1. RAM(独立端口)

拥有独立的数据输入存储端口和数据输出读取端口的RAM。 可以导出为 VHDL/Verilog。

输入

A

输入和输出共用地址位

Din

输入数据

str

输入使能，高电平有效

C

时钟输入

Id

输出使能，高电平有效，当为低电平时，输出呈高阻态

输出

D

输出数据

属性

数据位数

数据线位数

地址位数

地址线位数

标签

该组件的名称

旋转

组件在电路中的旋转角度

数字格式

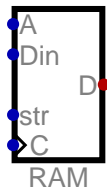
显示数字的格式

定点数

小数位数

可编程存储器

设置 ROM 为可编程存储器，这样就可以通过外部 IDE 访问。



## 9.2. Block-RAM(独立端口)

该 RAM 输出仅在时钟上升沿更新，用于 FPGA 的块内存。可以导出为 VHDL/Verilog。

输入

A

输入和输出共用地址位

Din

输入数据

str

输入使能，高电平有效

C

时钟输入

输出

D

输出数据

属性

数据位数

数据线位数

地址位数

地址线位数

标签

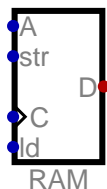
该组件的名称

旋转

组件在电路中的旋转角度

可编程存储器

设置 ROM 为可编程存储器，这样就可以通过外部 IDE 访问。



## 9.3. RAM(双向端口)

拥有一个既可用作数据输入，又可用作数据输出端口的RAM。



## 输入

A

输入和输出共用地址位

str

输入使能，高电平有效

C

时钟输入

ld

输出使能，高电平有效

## 输出

D

双向数据端口

## 属性

数据位数

数据线位数

地址位数

地址线位数

标签

该组件的名称

旋转

组件在电路中的旋转角度

数字格式

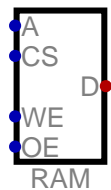
显示数字的格式

定点数

小数位数

可编程存储器

设置 ROM 为可编程存储器，这样就可以通过外部 IDE 访问。



#### 9.4. RAM(片选)

拥有一个既可用作数据输入，又可用作数据输出端口的RAM。如果片选输入(CS)为低电平，组件被禁止。用于通过使用较小容量的RAM和地址解码器构造较大容量的RAM。

## 输入

A

输入和输出共用地址位

CS

片选使能，输入为高电平时，RAM可用，否则输出为高阻态。

WE

输入使能，高电平有效

OE

输出使能，高电平有效

## 输出

D

双向数据端口

## 属性

数据位数

数据线位数

地址位数

地址线位数

标签

该组件的名称

翻转输入

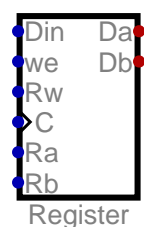
选择需要翻转的输入信号

旋转

组件在电路中的旋转角度

可编程存储器

设置 ROM 为可编程存储器，这样就可以通过外部 IDE 访问。



## 9.5. Register File

1个写端口，两个读端口的存储器，用于实现处理器寄存器。可以导出为 VHDL/Verilog。

## 输入

Din

输入数据

we

写使能，高电平有效

Rw

写寄存器

C

时钟

Ra

输出到端口 a 的寄存器

Rb

输出到端口 b 的寄存器

## 输出

Da

输出端口 a

Db

输出端口 b

## 属性

数据位数

数据线位数

地址位数

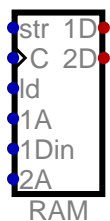
地址线位数

标签

该组件的名称

旋转

组件在电路中的旋转角度



## 9.6. RAM(多端口)

一个端口可以读写，另一个端口只读的RAM。第二个端口可用于图形逻辑访问。此时，处理器写数据到RAM，图形逻辑同时读取数据。可以导出为 VHDL/Verilog。

## 输入

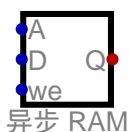
- str**  
输入使能，高电平有效
- C**  
时钟输入
- ld**  
输出使能，高电平有效，用于输出端口 1
- 1A**  
用于端口 1 读取和写入数据的地址位
- 1Din**  
输入数据
- 2A**  
用于端口 2 读取和写入数据的地址位

## 输出

- 1D**  
输出端口 1
- 2D**  
输出端口 2

## 属性

- 数据位数  
数据线位数
- 地址位数  
地址线位数
- 标签  
该组件的名称
- 旋转  
组件在电路中的旋转角度
- 可编程存储器  
设置 **ROM** 为可编程存储器，这样就可以通过外部 **IDE** 访问。



## 9.7. 异步 RAM

当 **we** 为高电平时，每当地址或数据改变时，都会存储数据 **D** 到地址 **A** 可以导出为 VHDL/Verilog。

## 输入

- A**  
读写地址
- D**  
待存储数据
- we**  
写使能

输出

Q

输出地址 A 存储的数据

属性

数据位数

数据线位数

地址位数

地址线位数

翻转输入

选择需要翻转的输入信号

标签

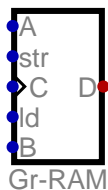
该组件的名称

旋转

组件在电路中的旋转角度

可编程存储器

设置 ROM 为可编程存储器，这样就可以通过外部 IDE 访问。



## 9.8. 显存

用于显示位图图形。每个像素都由一个存储器地址表示。存储的值使用固定的调色板定义像素的颜色。实现了两个屏幕缓冲区以支持页翻转。输入B选择显示哪个缓冲区。总存储器大小为  $dx * dy * 2$ 。所使用的调色板的结构如下：索引0-9对应于白色，黑色，红色，绿色，蓝色，黄色，青色，品红色，橙色和粉红色。索引32-63映射灰度值，索引64-127表示64个颜色值，其中每个颜色通道有两位。这样产生了一个简单的调色板，该调色板只能用7位寻址。如果支持16位索引，从0x8000开始，则可以使用每个颜色通道5位的高色模式，从而启用32768种颜色。

输入

A

输入和输出共用地址位

str

输入使能，高电平有效

C

时钟输入

ld

输出使能，高电平有效

B

选择需要显示的屏幕缓冲区

输出

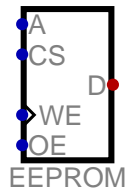
D

双向数据端口

## 属性

- 数据位数
  - 数据线位数
- 标签
  - 该组件的名称
- 宽度(像素)
  - 使用像素描述的屏幕宽度
- 高度(像素)
  - 使用像素描述的屏幕高度
- 旋转
  - 组件在电路中的旋转角度

## 10. 存储器 - EEPROM



### 10.1. EEPROM

拥有一个既可用作数据输入，又可用作数据输出端口的EEPROM。如果片选输入(**CS**)为低电平，组件被禁止。

## 输入

- A**
  - 输入和输出共用地址位
- CS**
  - 片选使能，输入为高电平时，EEPROM可用，否则输出为高阻态。
- WE**
  - 输入使能，高电平有效
- OE**
  - 输出使能，高电平有效

## 输出

- D**
  - 双向数据端口

## 属性

- 数据位数
  - 数据线位数
- 地址位数
  - 地址线位数
- 标签
  - 该组件的名称
- 翻转输入
  - 选择需要翻转的输入信号

数据

存储在该组件中的值

旋转

组件在电路中的旋转角度

数字格式

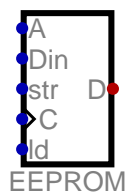
显示数字的格式

定点数

小数位数

可编程存储器

设置 ROM 为可编程存储器，这样就可以通过外部 IDE 访问。



## 10.2. EEPROM(独立端口)

拥有独立的数据输入存储端口和数据输出读取端口的EEPROM。

输入

A

输入和输出共用地址位

Din

输入数据

str

输入使能，高电平有效

C

时钟输入

Id

输出使能，高电平有效

输出

D

输出数据

属性

数据位数

数据线位数

地址位数

地址线位数

标签

该组件的名称

数据

存储在该组件中的值

旋转

组件在电路中的旋转角度

数字格式

显示数字的格式

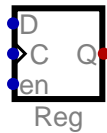
定点数

小数位数

可编程存储器

设置 ROM 为可编程存储器，这样就可以通过外部 IDE 访问。

## 11. 存储器



### 11.1. 寄存器

可以设置数据位宽度，和D触发器不同，该寄存器具有使能输入。可以导出为 VHDL/Verilog。

输入

D

输入

C

时钟输入，上升沿触发

en

使能输入，高电平有效

输出

Q

返回存储内容

属性

数据位数

数据线位数

标签

该组件的名称

翻转输入

选择需要翻转的输入信号

旋转

组件在电路中的旋转角度

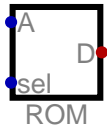
程序计数器

将该寄存器作为程序计数器。寄存器的值返回给外部汇编器，用来表示当前调试代码行。

作为测量值

如果选中，该值将作为测量值在测量图和数据表中显示。此时，必须指定标签作为值的标识。





## 11.2. ROM

非易失存储器。可以通过属性对话框编辑存储内容。可以导出为 VHDL/Verilog。

输入

A

地址位

sel

如果 sel 信号为高电平，输出为地址A所存储的数据，如果 sel 信号为低电平，输出为高阻态。

输出

D

输出

属性

数据位数

数据线位数

地址位数

地址线位数

标签

该组件的名称

数据

存储在该组件中的值

旋转

组件在电路中的旋转角度

数字格式

显示数字的格式

定点数

小数位数

可编程存储器

设置 ROM 为可编程存储器，这样就可以通过外部 IDE 访问。

模型启动时重新载入

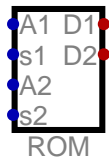
每次模型启动时重新载入十六进制文件

文件

载入 ROM 数据的文件

导入时使用大端对齐

导入时使用大端对齐



### 11.3. 多端口 ROM

非易失性存储器，可以在属性对话框编辑存储数据。

输入

A1

D1输出端口的地址

s1

如果输入为高电平，D1 输出使能，如果输入为低电平，D1 输出高阻。

A2

D2输出端口的地址

s2

如果输入为高电平，D2 输出使能，如果输入为低电平，D2 输出高阻。

输出

D1

当 s1 为高电平时，输出 A1 地址的数据

D2

当 s2 为高电平时，输出 A2 地址的数据

属性

数据位数

数据线位数

地址位数

地址线位数

标签

该组件的名称

数据

存储在该组件中的值

旋转

组件在电路中的旋转角度

数字格式

显示数字的格式

定点数

小数位数

可编程存储器

设置 ROM 为可编程存储器，这样就可以通过外部 IDE 访问。

模型启动时重新载入

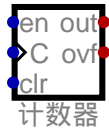
每次模型启动时重新载入十六进制文件

文件

载入 ROM 数据的文件

导入时使用大端对齐

导入时使用大端对齐



#### 11.4. 计数器

简单的计数器。时钟上升沿增加计数，通过 **clr** 输入重置为 0。可通过属性对话框设置计数器位数。可以导出为 VHDL/Verilog。

输入

**en**

使能位，高电平有效

**C**

时钟输入

**clr**

同步复位，高电平有效

输出

**out**

计数输出

**ovf**

溢出位

属性

数据位数

数据线位数

翻转输入

选择需要翻转的输入信号

标签

该组件的名称

旋转

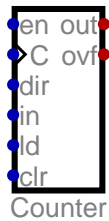
组件在电路中的旋转角度

作为测量值

如果选中，该值将作为测量值在测量图和数据表中显示。此时，必须指定标签作为值的标识。

程序计数器

将该寄存器作为程序计数器。寄存器的值返回给外部汇编器，用来表示当前调试代码行。



### 11.5. 计数器(可预设)

可预置位的计数器，可以指定最大值和计数方向。可以导出为 VHDL/Verilog。

输入

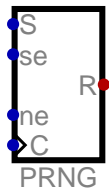
- en  
使能位，高电平有效
- C  
时钟输入
- dir  
计数方向，0 表示递增
- in  
数据输入
- ld  
输入使能，当为 1 时，输入数据在下一个时钟信号存入计数器
- clr  
同步复位

输出

- out  
计数输出
- ovf  
溢出位，当使能时，如果计数到达最大或递减到 0，则输出 1。

属性

- 数据位数  
数据线位数
- 最大值  
如果为 0，则使用可能的最大值
- 翻转输入  
选择需要翻转的输入信号
- 标签  
该组件的名称
- 旋转  
组件在电路中的旋转角度
- 作为测量值  
如果选中，该值将作为测量值在测量图和数据表中显示。此时，必须指定标签作为值的标识。
- 程序计数器  
将该寄存器作为程序计数器。寄存器的值返回给外部汇编器，用来表示当前调试代码行。



## 11.6. 随机数生成器

用于生成随机数。

输入

**S**

随机数种子

**se**

如果为高电平，生成器在下次时钟上升沿初始化。

**ne**

如果为高电平，下次时钟上升沿输出一个新的随机数。

**C**

时钟输入

输出

**R**

随机数输出

属性

数据位数

数据线位数

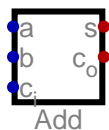
标签

该组件的名称

旋转

组件在电路中的旋转角度

## 12. 运算器



### 12.1. 加法器

计算输入 **a** 和输入 **b** 的和，如果设置进位，则结果加 1。可以导出为 VHDL/Verilog。

## 输入

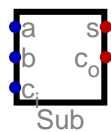
- a**  
第一个输入
- b**  
第二个输入
- c\_i**  
进位输入，如果为高电平，则结果加 1。

## 输出

- s**  
加法运算结果
- c\_o**  
输出进位

## 属性

- 标签  
该组件的名称
- 数据位数  
数据线位数
- 旋转  
组件在电路中的旋转角度



## 12.2. 减法器

将二进制输入 **a** 和输入 **b** 相减，如果进位输入为高电平，则结果减 1。可以导出为 VHDL/Verilog。

## 输入

- a**  
输入 **a**
- b**  
输入 **b**
- c\_i**  
进位输入，如果为高电平，则结果减 1。

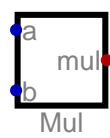
## 输出

- s**  
减法运算结果
- c\_o**  
如果出现溢出，则输出为 1。

## 属性

- 标签  
该组件的名称

数据位数  
    数据线位数  
旋转  
    组件在电路中的旋转角度



12.3. 乘法器

将整数输入 **a** 和 **b** 相乘 可以导出为 VHDL/Verilog。

输入

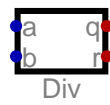
**a**  
    输入 **a**  
**b**  
    输入 **b**

输出

**mul**  
    乘法运算结果

属性

标签  
    该组件的名称  
补码运算  
    使用补码  
数据位数  
    数据线位数  
旋转  
    组件在电路中的旋转角度



12.4. 除法器

将整数 **a** 除以 整数 **b**。如果除数为零，则除以1。在有符号除法中，余数始终为正。

输入

**a**  
    被除数  
**b**  
    除数

输出

q

商

r

余数

属性

标签

该组件的名称

数据位数

数据线位数

补码运算

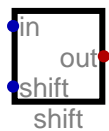
使用补码

余数始终为正

如果为高电平，符号除法的余数始终为正。

旋转

组件在电路中的旋转角度



## 12.5. 桶式移位器

根据移位输入将数据移位

输入

in

输入数据

shift

移位宽度

输出

out

移位后的输出

属性

标签

该组件的名称

数据位数

数据线位数

补码输入

输入数据为补码格式

方向

设置方向

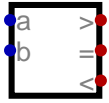
模式

桶式移位器模式

旋转

组件在电路中的旋转角度





## 12.6. 比较器

比较二进制输入 **a** 和 **b** 可以导出为 VHDL/Verilog。

输入

**a**  
输入 **a**

**b**  
输入 **b**

输出

**>**  
如果 **a** 大于 **b**, 输出 1

**=**  
如果 **a** 等于 **b**, 输出 1

**<**  
如果 **a** 小于 **b**, 输出 1

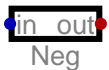
属性

标签  
该组件的名称

数据位数  
数据线位数

补码运算  
使用补码

旋转  
组件在电路中的旋转角度



## 12.7. 补码器

补码运算 可以导出为 VHDL/Verilog。

输入

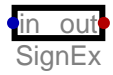
**in**  
输入

输出

**out**  
补码输出

属性

数据位数  
 数据线位数  
 旋转  
 组件在电路中的旋转角度



## 12.8. 符号扩展器

增加符号数的位宽同时保持值不变 可以导出为 VHDL/Verilog。

输入

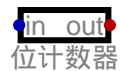
in  
 输入值，位宽必须小于输出

输出

out  
 扩展后的输入

属性

标签  
 该组件的名称  
 输入位宽  
 输出位宽必须大于输入位宽  
 输出位宽  
 输出位宽必须大于输入位宽  
 旋转  
 组件在电路中的旋转角度



## 12.9. 位计数器

返回输入值中位为 1 的位数

输入

in  
 输入

输出

out  
 包含 1 的位数

属性

数据位数  
 数据线位数  
 旋转  
 组件在电路中的旋转角度

## 13. 开关



### 13.1. 开关

简单的开关，没有门延迟

输出

A1

连接点

B1

连接点

属性

数据位数

数据线位数

标签

该组件的名称

极数

可用极数

关闭

设置开关的初始状态

旋转

组件在电路中的旋转角度

镜像

镜像组件

开关行为输入化

开关行为类似输入，开对应 0，关对应 1。



### 13.2. 双掷开关

双掷开关，没有门延迟

输出

A1

连接点

B1

连接点

C1

连接点

属性

数据位数

数据线位数

标签

该组件的名称

极数

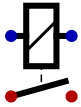
可用极数

旋转

组件在电路中的旋转角度

镜像

镜像组件



### 13.3. 继电器

继电器是一种可以通过线圈控制的开关，如果电流流过线圈，则开关会闭合或断开。不关心电流方向。

输入

in1

控制输入

in2

控制输入

输出

A1

连接点

B1

连接点

属性

数据位数

数据线位数

标签

该组件的名称

极数

可用极数

闭合寄存器

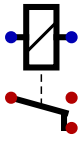
输入输入为低电平，继电器闭合

旋转

组件在电路中的旋转角度

镜像

镜像组件



### 13.4. 双掷继电器

继电器是一种可以通过线圈控制的开关，如果电流流过线圈，则开关会闭合或断开。不关心电流方向。

输入

in1

控制输入

in2

控制输入

输出

A1

连接点

B1

连接点

C1

连接点

属性

数据位数

数据线位数

标签

该组件的名称

极数

可用极数

旋转

组件在电路中的旋转角度

镜像

镜像组件



### 13.5. P 沟道场效应晶体管

P 沟道场效应晶体管

输入

G

栅极

输出

S

源极

D

漏极

属性

数据位数

数据线位数

单向

单向晶体管仅将信号从源极传播到漏极。仿真时，比双向晶体管快很多。由于没有从漏极到源极的反馈，因此在这种模式下，晶体管导通时无法使连接的导线短路。此模式对于模拟某些CMOS电路是必需的。

标签

该组件的名称

旋转

组件在电路中的旋转角度

镜像

镜像组件



### 13.6. N 沟道场效应晶体管

N 沟道场效应晶体管

输入

G

栅极

输出

D

漏极

S

源极

属性

数据位数

数据线位数

单向

单向晶体管仅将信号从源极传播到漏极。仿真时，比双向晶体管快很多。由于没有从漏极到源极的反馈，因此在这种模式下，晶体管导通时无法使连接的导线短路。此模式对于模拟某些CMOS电路是必需的。

标签

该组件的名称

旋转

组件在电路中的旋转角度

镜像  
镜像组件



### 13.7. Fuse

用于构建一次性可编程存储器

输出

out1

连接点

out2

连接点

属性

已编程

如果选中，二极管(熔丝)为已烧写，浮动门场效应晶体管为已充电。可以通过快捷键‘p’修改设置。

旋转

组件在电路中的旋转角度



### 13.8. 二极管(上拉)

用于将导线上拉至VDD，必须将下拉电阻连接到二极管输出。

输入

in

如果输入为高电平，输出则为高电平，其它情况输出为高阻。

输出

out

如果输入为高电平，输出则为高电平，其它情况输出为高阻。

属性

已编程

如果选中，二极管(熔丝)为已烧写，浮动门场效应晶体管为已充电。可以通过快捷键‘p’修改设置。

旋转

组件在电路中的旋转角度



### 13.9. 二极管(下拉)

用于将导线下拉至地，必须将上拉电阻连接到二极管输出。

输入

in

如果输入为低电平，输出则为低电平，其它情况输出为高阻。

输出

out

如果输入为低电平，输出则为低电平，其它情况输出为高阻。

属性

已编程

如果选中，二极管(熔丝)为已烧写，浮动门场效应晶体管为已充电。可以通过快捷键‘p’修改设置。

旋转

组件在电路中的旋转角度



### 13.10. P 沟道浮动门场效应晶体管

P 沟道浮动门场效应，如果浮栅中存储有电荷，即使栅极为低电平时晶体管也不导通

输入

G

栅极

输出

S

源极

D

漏极

属性

数据位数

数据线位数

标签

该组件的名称

已编程

如果选中，二极管(熔丝)为已烧写，浮动门场效应晶体管为已充电。可以通过快捷键‘p’修改设置。

旋转

组件在电路中的旋转角度

镜像

镜像组件





### 13.11. N 沟道浮动门场效应晶体管

N 沟道浮动门场效应晶体管，如果浮栅中存储有电荷，即使栅极为高电平时晶体管也不导通

输入  
G  
栅极

输出  
D  
漏极  
S  
源极

属性

数据位数  
数据线位数

标签  
该组件的名称

已编程  
如果选中，二极管(熔丝)为已烧写，浮动门场效应晶体管为已充电。可以通过快捷键 'p' 修改设置。

旋转  
组件在电路中的旋转角度

镜像  
镜像组件



### 13.12. 传输门

由两个晶体管构建的传输门

输入  
S  
控制输入  
 $\neg S$   
翻转的控制输入

输出  
A  
输入 A  
B  
输入 B

属性

数据位数

数据线位数

旋转

组件在电路中的旋转角度

## 14. 其他

### Test

#### 14.1. 测试用例

用于定义测试用例，可用于自动检测电路的行为是否符合定义。可以导出为 VHDL/Verilog。

属性

标签

该组件的名称

测试数据

测试用例描述，详细的语法可以在测试数据编辑器帮助对话框查看。

启用

启用或禁用该组件

## 15. 其他 - Decoration

Text

#### 15.1. Text

显示一段文本，其对电路仿真没有影响，可以通过其属性对话框修改文本内容。

属性

简介

关于该组件和其使用的简短描述

字体大小

设置文本字体大小

旋转

组件在电路中的旋转角度

文本对齐

文本对齐方式

对齐到栅格

如果选中，组件将会和栅格对齐。

Text



## 15.2. 矩形

显示一个矩形，其对电路仿真没有影响。 如果使用 - 作为标题，将不会显示标题。

属性

标签

该组件的名称

宽

以栅格为单位的宽度

高

以栅格为单位的高度

字体大小

设置文本字体大小

内置文本

将文本放在矩形内部

底部文本

将文本放在矩形底部

右侧文本

将文本放在矩形右侧

对齐到栅格

如果选中，组件将会和栅格对齐。

## 16. 其他 - Generic

### 16.1. 通用电路初始化

用于启动通用电路时执行的代码。如果直接启动通用电路，必须设置。 可以导出为 VHDL/Verilog。

属性

标签

该组件的名称

启用

启用或禁用该组件

通用参数

通用化电路的代码

## Code

### 16.2. Code

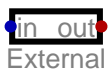
创建具体电路时执行的代码，例如向电路添加导线或组件。可以导出为 VHDL/Verilog。

属性

通用参数

通用化电路的代码

## 17. 其他 - VHDL/Verilog



### 17.1. External

通过执行外部程序计算逻辑值。用于通过 VHDL 或 Verilog 定义组件行为。实际的仿真行为由外部仿真器完成。目前支持 VHDL 仿真器 ghdl 和 verilog 仿真器 Icarus Verilog。组件的名称必须和模块名称一致！可以导出为 VHDL/Verilog。

输入

in

输出

out

属性

标签

该组件的名称

宽度

当电路作为子电路时，电路符号的宽度

输入

外部进程的输入端口。格式为逗号分割的信号名称，对于每个信号，可通过冒号指定位数。如8位加法器的输入可表示为 "a:8, b:8, c\_in"。

输出

外部进程的输出端口。格式为逗号分割的信号名称，对于每个信号，可通过冒号指定位数。如8位加法器的输出可表示为 "s:8, c\_out"。

程序代码

被外部应用执行的程序代码

应用

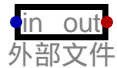
定义使用哪个应用

GHDL 选项

用于处理 GHDL 的选项

IVerilog 选项

IVerilog 选项



## 17.2. 外部文件

通过执行外部程序计算逻辑值。用于通过 VHDL 或 Verilog 定义组件行为。实际的仿真行为由外部仿真器完成。目前支持 VHDL 仿真器 ghdl 和 verilog 仿真器 Icarus Verilog。组件的名称必须和模块名称一致！可以导出为 VHDL/Verilog。

输入

in

输出

out

属性

标签

该组件的名称

宽度

当电路作为子电路时，电路符号的宽度

输入

外部进程的输入端口。格式为逗号分割的信号名称，对于每个信号，可通过冒号指定位数。如8位加法器的输入可表示为 "a:8, b:8, c\_in"。

输出

外部进程的输出端口。格式为逗号分割的信号名称，对于每个信号，可通过冒号指定位数。如8位加法器的输出可表示为 "s:8, c\_out"。

程序代码

通过外部应用执行的代码文件。

应用

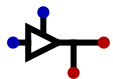
定义使用哪个应用

GHDL 选项

用于处理 GHDL 的选项

IVerilog 选项

IVerilog 选项



## 17.3. 双向管脚

该组件仅在生成 VHDL 或 Verilog 时有效，用于创建双向端口。该组件只能在顶级电路使用。可以导出为 VHDL/Verilog。

输入

wr

待输出数据

oe

使能输出

输出

rd

待读取数据

pin

实际管脚连接点，只有一个输出可以连接到该端口

属性

数据位数

数据线位数

旋转

组件在电路中的旋转角度

镜像

镜像组件

## 18. 其他



### 18.1. 电源连接器

用于确保电源和地被连接

输入

VDD

必须连接到电源

GND

必须连接到地

属性

标签

该组件的名称

旋转

组件在电路中的旋转角度



### 18.2. 双向分裂器

可用于数据总线特别是存储器模块的构建

输入

OE

如果置位，位于管脚 D 的值分别输出到 D[i]，否则，D[i] 汇聚输出到 D

## 输出

D

集合连接

D0

数据位 0

## 属性

数据位数

数据线位数

旋转

组件在电路中的旋转角度

镜像

镜像组件

间距

设置输入和输出端口间距



## 18.3. 复位器

在电路初始化期间，该组件的输出保持高电平。电路稳定后，输出变为低电平。如果输出反相，则其行为相反。可以导出为 VHDL/Verilog。

## 输出

Reset

复位输出

## 属性

标签

该组件的名称

翻转输出

如果选中，输出将被翻转。

旋转

组件在电路中的旋转角度



## 18.4. Break

如果电路中使用到该组件，则 运行至中断 按钮可用。仅在禁用实时时钟时才能使用此功能！

## 输入

brk

如果检测到上升沿则停止仿真

## 属性

标签

该组件的名称

启用

启用或禁用该组件

超时周期

如果设置的周期数完成而没有中断信号则会报错。

旋转

组件在电路中的旋转角度



## 18.5. 停止

上升沿时停止仿真，效果同工具栏中的停止按钮

输入

stop

升沿时停止仿真

属性

标签

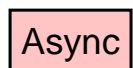
该组件的名称

翻转输入

选择需要翻转的输入信号

旋转

组件在电路中的旋转角度



## 18.6. 异步时序

允许设置异步时序电路如Muller流水线的时序。

属性

使用实时时钟

如果选中，当电路开始模拟时将使用实时时钟

频率/赫兹

实时时钟频率



## E 库

- 27c801:** 8 Mbit (1Mb x 8) UV EPROM
- 28c010:** 1-Megabit (128K x 8) Paged Parallel EEPROM; DATA Polling for End of Write Detection not implemented!
- 28c16:** 16K (2K x 8) Parallel EEPROM; DATA Polling for End of Write Detection not implemented!
- 28c64:** 64K (8K x 8) Parallel EEPROM; DATA Polling for End of Write Detection not implemented!
- 28c256:** 256K (32K x 8) Paged Parallel EEPROM; DATA Polling for End of Write Detection not implemented!
- 28c512:** 512K-Bit (64K x 8) CMOS Parallel EEPROM; DATA Polling for End of Write Detection not implemented!
- 7400:** quad 2-input NAND gate
- 7401:** quad 2-input NAND gate with open-collector outputs
- 7402:** quad 2-input NOR gate
- 7403:** quad 2-input NAND gate with open-collector outputs, different pinout than 7401
- 7404:** hex inverter
- 7405:** hex inverter, open-collector output
- 7406:** hex inverter buffer, open-collector output
- 7407:** hex buffer, open-collector output
- 7408:** quad 2-input AND gate
- 7409:** quad 2-input AND gate with open-collector outputs
- 7410:** triple 3-input NAND gate
- 7411:** triple 3-input AND gate
- 7412:** triple 3-input NAND gate with open-collector outputs
- 7413:** dual 4-input NAND gate, Schmitt trigger
- 7414:** hex inverter, Schmitt trigger
- 7415:** triple 3-input AND gate with open-collector outputs
- 7416:** hex inverter buffer, open-collector output, same as 7406
- 7417:** hex buffer, open-collector output, same as 7407
- 7420:** dual 4-input NAND gate
- 7421:** dual 4-input AND gate
- 7425:** dual 4-input NOR gate with strobe
- 7427:** triple 3-input NOR gate
- 7428:** quad 2-input NOR buffer
- 7430:** 8-input NAND gate
- 7432:** quad 2-input OR gate
- 7434:** hex buffer
- 7440:** dual 4-input NAND buffer
- 7442:** 4-line BCD to 10-line decimal decoder
- 7447:** BCD to 7-segment decoder, active low
- 7448:** BCD to 7-segment decoder, active high
- 7451:** 2-input/3-input AND-NOR gate
- 7454:** 2-3-2-3-line AND NOR gate
- 7455:** 2 wide 4-input AND-NOR gate
- 7458:** dual AND OR gate
- 7474:** dual D-flip-flop
- 7476:** dual J-K flip-flops with preset and clear
- 7480:** Gated Full Adder with Complementary Inputs and Complementary Sum Outputs

**7482:** 2-bit binary full adder  
**7483:** 4-bit binary full adder  
**7483Real:** 4-bit binary full adder, real gates  
**7485:** 4-bit comparator  
**7486:** quad 2-input XOR gate  
**7489:** 64-bit RAM  
**7490:** asynchronous two - five - decimal addition counter  
**7493:** 4-bit Binary Counter. Connect QA to CKB and clock on CKA for full 4-bit counter.  
Connect QB to R1 and QD to R2 for a BCD counter.  
**74107:** dual J-K flip-flops with clear  
**74109:** Dual J-NOT-K flip-flop with set and reset; positive-edge-trigger  
**74112:** Dual J-K negative-edge-triggered flip-flop, clear and preset  
**74116:** dual 4-bit D-type latches  
**74125:** Quadruple bus buffer gates with 3-state outputs (active low output enable)  
**74126:** Quadruple bus buffer gates with 3-state outputs (active high output enable)  
**74133:** 13-input NAND gate  
**74138:** 3-line to 8-line decoder/demultiplexer, inverted out  
**74139:** dual 2-line to 4-line decoder/demultiplexer  
**74147:** 10-line to 4-line priority encoder  
**74148:** 8-line to 3-Line priority encoder  
**74150:** 4-line to 16-line data selectors/multiplexers  
**74151:** 3-line to 8-line data selectors/multiplexers  
**74153:** dual 4-line to 1-line data selectors/multiplexers  
**74154:** 4-line to 16-line decoders/demultiplexers  
**74157:** quad 2-line to 1-line data selectors/multiplexers  
**74160:** decimal synchronous counter, async clear  
**74161:** hex synchronous counter, async clear  
**74162:** decimal synchronous counter  
**74162Real:** decimal synchronous counter, real gates  
**74163:** hex synchronous counter  
**74164:** 8-bit parallel-out serial shift register, asynchronous clear  
**74165:** parallel-load 8-bit shift register  
**74166:** 8-Bit Parallel-In/Serial-Out Shift Register  
**74173:** quad 3-state D flip-flop with common clock and reset  
**74174:** hex D-flip-flop  
**74175:** quad D-flip-flop  
**74181:** 4-bit arithmetic logic unit  
**74182:** look-ahead carry generator  
**74189:** 64-Bit Random Access Memory with 3-STATE Outputs  
**74190:** Presettable synchronous 4-bit bcd up/down counter  
**74191:** Presettable synchronous 4-bit binary up/down counter  
**74193:** Synchronous 4-Bit Up/Down Binary Counter with Dual Clock  
**74194:** 4-Bit Bidirectional Universal Shift Register  
**74194real:** 4-Bit Bidirectional Universal Shift Register, Databook implementation.  
**74198:** 8-bit shift register  
**74238:** 3-line to 8-line decoder/demultiplexer  
**74244:** octal 3-state buffer/line driver/line receiver  
**74245:** octal bus transceivers with 3-state outputs  
**74247:** BCD to 7-segment decoder, active low, tails on 6 and 9  
**74248:** BCD to 7-segment decoder, active high, tails on 6 and 9

**74253:** dual tri state 4-line to 1-line data selectors/multiplexers  
**74257:** quad 2-line to 1-line data selectors/multiplexers (3-state output)  
**74260:** dual 5-input NOR gate  
**74266:** quad 2-input XNOR gate with open collector outputs  
**74273:** octal D-type flip-flop with clear  
**74280:** 9 bit Odd-Even Parity Generator-Checker  
**74283:** 4-bit binary full adder, alternative pinning  
**74299:** 8-Input Universal Shift/Storage Register with Common Parallel I/O Pins  
**74373:** octal transparent latches  
**74374:** octal positive-edge-triggered flip-flops  
**74377:** Octal D Flip-Flop with enable  
**74381:** 4-Bit Arithmetic Logic Unit with high-speed expansion  
**74382:** 4-Bit Arithmetic Logic Unit with ripple carry output  
**74540:** octal buffer/line driver, inverted  
**74541:** octal buffer/line driver  
**74573:** octal transparent latches, different pinout compared to 74373  
**74574:** octal positive-edge-triggered flip-flops, different pinout compared to 74374  
**74590:** 8-bit binary counter with tri-state output registers  
**74595:** 8-Bit Shift Registers with 3-State Output Registers  
**74670:** 3-state 4-by-4 Register File  
**74682:** 8-bit digital comparator  
**74688:** 8-bit identity comparator  
**74779:** 8-Bit Bidirectional Binary Counter with 3-STATE Outputs  
**74804:** hex 2-input NAND gate <https://www.ti.com/lit/ds/symlink/sn74as804b.pdf>  
**74805:** hex 2-input NOR gate <http://www.ti.com/lit/ds/symlink/sn54as805b.pdf>  
**74808:** hex 2-input AND gate <http://www.ti.com/lit/ds/symlink/sn54as808b.pdf>  
**74832:** hex 2-input OR gate <http://www.ti.com/lit/ds/symlink/sn54as832b.pdf>  
**744002:** dual 4-input NOR gate  
**744017:** Johnson decade counter with 10 decoded outputs  
**744075:** triple 3-input OR gate  
**747266:** quad 2-input XNOR gate  
**7440105:** 4-Bit x 16-Word FIFO Register  
**A623308A:** 8K X 8 BIT CMOS SRAM  
**RAM32Bit:** A 32-bit memory that allows byte access and can handle non-aligned memory addresses.