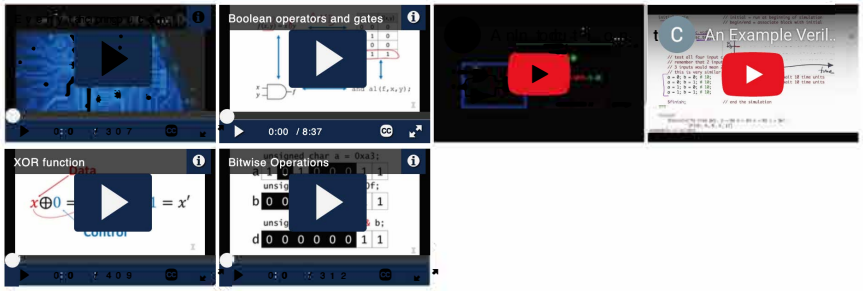


视频来源：Geoffrey Herman 和 Craig Zilles，文字来源：Geoffrey Herm 一个

视频和文本从略微不同的角度涵盖了相同的内容，因此请选择更能帮助你的 或者
使用其中一个来帮助您澄清来自另一个的问题。

除了“高级概述”外，所有文本和视频也都包含在以下相关问题中。如果您想要 吨，
你可以在后面的问题中分成小部分阅读和观看视频，而不是阅读/观看所有内容 第一
然后开始回答问题。这个问题主要是为了方便搜索而将所有内容聚合到一个地方 和 克
参考。

高层概述 观点

计算机硬件的核心组织概念是我们存储状态（例如，在 RAM 中），然后操纵该状态（即。 例如，
定期执行计算。我们将状态信息存储为 1 和 0，这需要我们使用布尔
函数来执行我们的状态操作。在本课程中，您需要能够流畅地在布尔函数的四种表示形式之间进行转换：布尔代数表达式、真值表、
电路图和 Verilog 硬件
描述语言。这四种表示在数学上是等价的，但可以帮助我们更轻松地执行不同的任务 伊莉。

在 C 等编程语言中，您将看到布尔运算符以两种不同的方式使用：按位逻辑运算 和逻辑运
算。位逻辑运算符 {&（按位与）、I（按位或）、 （按位非）、^按位异或）执行逻辑 艾尔
对变量中的位对进行运算（例如，char x, y, z; z = x & y;）。相比之下，逻辑运算符 {&& (AND)、11 (OR ），
！（NOT）对整个位串执行逻辑运算，将 O 视为 False，将非 O 视为 True（例如，if (x = =
0 && y = = 1))。按位逻辑运算是基础计算，我们可以使用它来构造更复杂的 d
加法和减法等计算。逻辑运算通常用于程序控制结构，如 if 语句或 for 循环。注意：当我们说“逻辑运算符”时，我们不包括按位逻辑
运算符，就像加法和减法等算术运算符不属于逻辑运算符集一样。见下文
佛 r
一些关于如何/何时使用这些运算符以及它们如何工作的示例。

详细信息和示例 莱斯

基本布尔运算 托尔斯

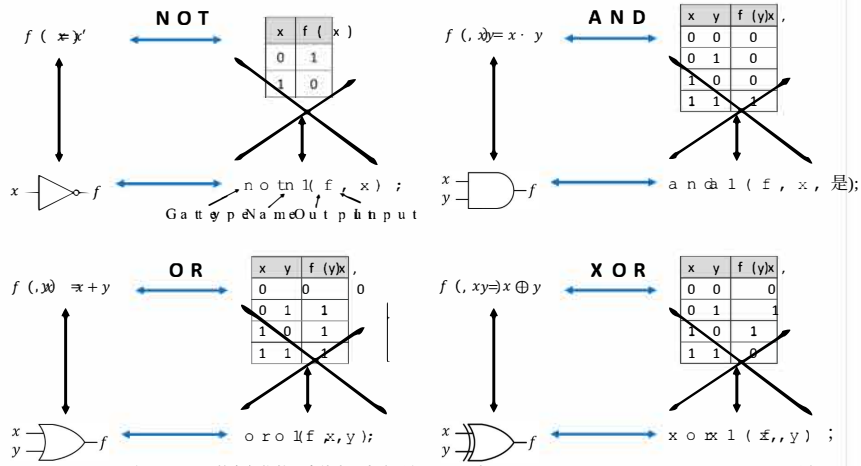


图 1：NOT、AND、OR 和 XOR Bool 的布尔代数、真值表、电路图和 Verilog 表示 平均
运算符

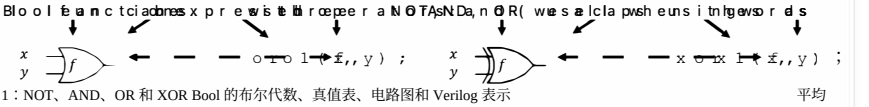


图 1：NOT、AND、OR 和 XOR Bool 的布尔代数、真值表、电路图和 Verilog 表示 平均
运算符

所有布尔函数都可以用三个运算符来表示：NOT、AND 和 OR（我们在使用这些词时全部大写 作为
布尔运算符）。布尔运算符总结在上图 1 和下表中。

- 不是
  - o 真值表：当输入为 True 或 1 时，输出为 False 或 0。当输入为 False 或 0 时，输出为 True 或 1 。
  - o 代数：我们在变量（或子表达式）后使用 ' 运算符来表示非。o 电路：一个三角形，左侧是
    输入，右侧是圆圈，即输出侧。o Verilog：not name(out, in); 注意：只能有一个输入变量
- 和
  - o 真值表：当两个输入均为 True 或 1 时，输出为 True 或 1，否则输出为 False 或 0。o 代数：我们将使用乘法符号，如 *
    或没有运算符的相邻变量。o 电路：在电路图中，它表示为“ D ”形门，输入在左侧，输出在装置上
    高度。
  - o Verilog：和名称（out，in0，inl， . . .); 注意：必须至少有两个输入变量，可以有更多
- 或者
  - o 真值表：当任何输入为 True 或 1 时，输出为 True 或 1，否则输出为 False 或 0。o 代数：我们将使用加法符号（即 + ）。o
    电路：它在电路图中表示为“箭头”形门，左侧为输入，右侧为输出
    埃
    对。o Verilog：或名称（out，in0，inl，
    。 . .); 注意：必须至少有两个输入变量，可以有更多

XOR

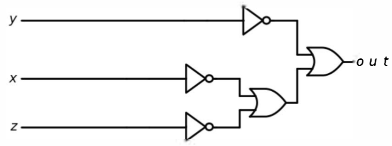
伊希+ 米伊夫赫

XOR 是一个有用的附加布尔函数，用作奇偶检测电路（当输入的 乌特
位为奇数，偶数为 O）或可控反相电路（对于 z = x XOR y，z = x when y = 0 and z = x when y = 1）We ca n
从基本布尔运算符 6:) = 构建一个 XOR 函数 · + · y')。

- 异或
  - o 真值表：当奇数个输入为真或 1 时，输出为真或 1。当偶数个输入为真或 1 时，输出为 True 或 1，输出为 False 或 0。o 代数：我们使用 6:) 运算符进行 XOR。o 电路：一个三角形，左侧是输入，右侧是圆圈，即输出侧。o Verilog：xor name(out, inl, in2, . . .); 注意：必须至少有两个输入变量，可以有更多 应收账款

将电路图转换为布尔表达式 锡安

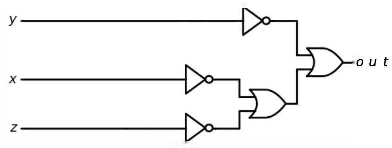
要导出布尔表达式，请从左侧的输入开始，并跟踪它们的值在向右侧传播时的情况。一旦所有 升
已经对门的输入进行了评估，则将该门添加到布尔表达式中。



1. 我们有所有非门的输入，因此我们得到 x', y', 和 z' 2. 我们有底部或门的所有输入，所以我们得到 x 1+ 2 1 3. 我们已经将所有输入都输入到了顶部的或门，因此我们得到了 = x1+ y + 2

1. 我们有底部与门和非门的所有输入 门，因此我们分别得到 z ' 和 yz 2. 我们有或门的所有输入，因此我们得到 z '+ yz 3. 我们有顶部与门的所有输入，因此我们得到 = (z'+ yz)x 。

或者，你可以从右边开始，向后追溯每个门的输入 ts。

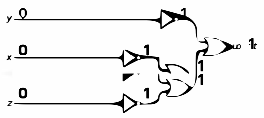


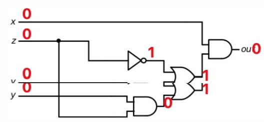
1. 从最右边的或门开始 ( ) + ( )
2. 添加或门的输入 (( ) ')+ (( ) + ( ))
3. 再返回一级 ((y) ')+ ((() ')+ (() '))
4. 再返回一级 ((y) ')+ (((x) ')+ ((z) '))
5. 删除多余的括号 = y' + x' + z'

1. 从最右边的与门开始 () *())
2. 添加与门的输入 (x)*(( )+ ( ))
3. 再返回一级 (x)*((() ')+ (( ) *()))
4. 再返回一级 (x)*(((z) ') + ((y) *((z))
5. 删除多余的括号 out = x (z '+ yz)

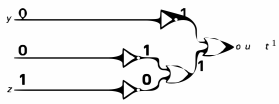
将电路图转化为事实 标签 乐

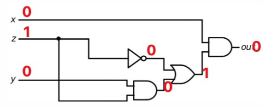
要导出真值表，请从表格顶行的值开始（例如，x= 0、y= 0、z= 0），然后将其代入循环 伊特
图。然后追踪 1 和 0 在电路中传播时的值。





对所有行重复此过程。我们再做一次 (x= 0, y= 0, z= l) 。





将布尔表达式转换为真值 标签 乐

要导出真值表，请从表格顶行的值开始（例如，x= 0、y= 0、z= 0），然后将其代入布尔值 一个
表达式。然后按照优先级顺序评估每个运算符：括号 ()、NOT '、AND*，然后是 OR+ 。

f(x,y,z) = y'+ x' + z' f (X, y, Z) = X (Z ' + Y * Z)

- f (0, 0, 0) = 0 ' + 0 ' + ' 0
- = 升+ 升+ 升
- = 1

- f(0,0,0) = 0(0'+ 0 *0)
- = 0(1+ 0)
- = 0(1)
- = 0

对所有行重复此过程。我们只做一次 e.

- f (0, 0, 1) = 0' + 0' + 1'
- = 1+ 1+ 0
- = 1

- f(0,0, 1) = 0(1 '+ 0* 1)
- = 0(0+ 0)
- = 0(0)
- = 0

转换布尔表达式 致 Veri 日志

Verilog 是一种描述语言，它仅描述电路（称为模块）具有哪些组件以及它们如何连接。我们建议在转换为 Verilog 之前先将布尔表达式转换为电路图。按照惯例，布尔函数的输出应首先列在 Verilog 模块声明中（即模块电路（输出，

. . 。））。布尔表达式的每个输出都需要声明为输出类型。类似地，输入 的 (編號)
布尔函数应该列在 Verilog 模块声明中的输出之后（即模块电路（输出 1、输出 2、
... , inl, in2, ...)。每个输入都需要声明为输入类型。电路图中的每条导线都需要声明为导线类型。布尔函数中的每个运算符或电路图中的门都需要在 Verilog 模块中有一个对应的门。您可以将以下原语用于逻辑门。

```
线路 < id> , < id> , 输入 < id
> , < id> , 输出 < id> , < id
> ...;
// N - i n p u t A N D g a t e
和 < id> ( out, inl, in2, inN) ; 或 < id> ( out, inl, in
// N - i n p u t O R g a t e
2, inN) ; xor < id> ( out, inl, in2, inN) ; nand < id
// N - i n p u t X O R g a t e
( out, inl, in2, inN) ; nor < id> ( out, inl, in2, .
// N - i n p u t N A N D g a t e ( N O T A N D )
// N - i n p u t N O R g a t e ( N O T O R )
// N - i n p u t X N O R g a t e ( N O T X O R ) inN) ; xn
or < id> ( out, inl, in2, . . . , inN) ; not < id> ( out, in) ; buf < id> ( out, in) ;
// 1 - i n p u t N O T g a t e
// 1 - i n p u t b u f f e r : c i r c u i t u s e d t o c o n n e c t i n p u t o r t o a n o u t
```

尝试为你的门命名有意义的名字，例如 nx，表示非 x 的门。同样，为你的线路命名有意义的名字，例如 al_o 乌特
用于第一个 AND 门的输出。以下 Verilog 代码为布尔表达式 w = x' yz 提供了一个 Verilog 模块

```
module example(w,x,y,z);        //new circuit definition
  output w;                     // output of new circuit
  input x,y,z;                  //input of new circuit
  wire a1_out,a2_out,not_x;     // connecting wires

  not  nx(not_x,x);             //each circuit instance needs a unique name
  and  a1(a1_out,not_x,z);
  and  a2(a2_out,x,z);          // gate output first, inputs after
  or   o1(w,a1_out,a2_out);
endmodule //example one
```

如果模块包含多个布尔表达式，则可以有多个输出。例如，你可以制作一个循环 伊特
由 f = x' yz + xz 和 g = x 'y + xy' + z 组成

```
module example(f,g,x,y,z);      //new circuit definition
  output f,g;                   //outputs of new circuit
  input x,y,z;                  //inputs of new circuit
  wire a1_out,a2_out,a3_out,a4_out,not_x,not_y;

  // complement variables
  not  nx(not_x,x);
  not  ny(not_y,y);

  // f output
  and  a1(a1_out,not_x,z);
  and  a2(a2_out,x,z);
  or   o1(f,a1_out,a2_out);

  // g output
  and  a3(a3_out,not_x,y);
  and  a4(a4_out,x,not_y);

endmodule //example one
```

将 Verilog 转换为布尔表达式 锡安

要将 Verilog 代码转换为布尔表达式，请从产生模块输出值的门开始，然后回溯 钾
朝向输入。以下示例使用替换方法，使用线路名称作为占位符，然后替换 尤丁
线路名称以及产生线路值的门操作。

```
模块示例（w, x, y, z）; 输出 w ; 输入 x, y, z ; 连接 al_out, not_x ; 非 nl（not_x, x）;
                      和 al（al_out, not_x, y）;
//circuit name(out,in1,in2);   或 ol（w, al_out, z）;
//each circuit instance needs a unique name   结束模块 // example one
```

门 ol 创建输出 w：w = z + (al_o 呃

Gateal 为 wireal_out 创建信号：w = z + (not_x * y)

门 n 1 为线 not_x : w = z 创建信号 y +

因为 Verilog 是一种描述语言，而不是编程语言，所以你可以重新排列门的顺序，而不需要把一切都搞砸了。在硬件中，一切都是并行运行的，所以所有的门总是根据 伊尔
电流连续输入。例如，下面的 Verilog 会产生相同的行为。注意声明 不需要 纳秒
首先发生，以便编译器知道每个变量的类型。

```
模块示例（w, x, y, z）; 输出 w ; 输入 x, y, z ; 连接 al_out, not_x ;
//new circuit definition   或 ol（w, al_out, z）;
//each circuit instance needs a unique name   和 al（al_out, not_x, y）;
//circuit name(out,in1,in2);   非 nl（not_x, x）; 结束模块 // example one
// connecting wires
```

这是另一个例子 乐

```
module example2(w,x,y,z);       // new circuit definition
  output w;                     //output of new circuit
  input x,y,z;                  // input of new circuit
  wire o1_out,o2_out,not_x,not_y; // connecting wires

  not  nx(not_x,x);
  not  ny(not_y,y);
  or   o1(o1_out,not_x,z);      //each circuit instance needs a unique name
  or   o2(o2_out,not_y,z);      //primitive gates can have 2 or more inputs
  and  a1(w,o1_out,o2_out);
endmodule //example one
```

Gate al 创建输出 章：吨 w = (ol_out} * (o2_out}* 十

门 o2 为线路 ol_out 创建信号：w = (not_y + z }*(o2_o }*X

门 ny 为线 not_y 创建信号：w = (y' + z}*(o2_o }*X

Gateol 为 wireo2_out 创建信号：w = (y' + z}*(not_x + y + z}* 十

门 nx 为线 not_x 创建信号：w = (y' + z} * (x' + y + *十 }

按位逻辑示例 莱斯

按位逻辑运算符（又称按位运算符）将变量视为单个位。输出的每个单独位都是一个函数 化
同一比特位置的两个比特的和。例如，89 和 103 的两个最低有效位均为 1，因此输出的最低有效位为 1 & 1 = 1。相反，89 和 103 的下一对最低有效位分别为 0 和 1 积极地，
所以 0 & 1 = 0。

我 o 0 1 1 0 0 1 我 o 0 0

&

我 o 1 0 0 0 1 0 1 0 1 1 我

=

1

例2，89和103的两个最低有效位均为1，因此输出的最低有效位为1 11= 1。下一个
89 和 103 的最低有效位对分别为 0 和 1，因此 0 11 = 1。最高有效位对为 0 和 0，因此 0 我
0 = 0.

0	1	0	1	1		1

|

0	1	1	0000	1	1	1

=

0	1	1	1	1	1	1	1	我

逻辑示例 乐

逻辑运算将变量整体视为布尔值（即单个真或假）。任何非 O 的变量都是 True。例如，数字 89 和 103 都被视为 True，因为它们不是 0 。如果逻辑表达式的计算结果为 False， 则 吨
输出 O（即 00000000）。如果逻辑表达式计算结果为 True，则输出 1（即 00000001）。在下面的示例中，运算符实际上是在执行 True && True，输出 True（00000001）。

我 o 0 1 1 0 0 1 我 o
& &
0 0 1 1 0 我 o 1
=
0 0 0 0 0 0 1 我

cod 中的按位逻辑 埃

利用信息的编码方式，可以使用按位逻辑运算来更快地进行许多计算 在
位。例如，假设我们将数字的状态（即声明一个变量）存储为 C 中的 char 或 int。正如您将在本章中学习到的 s
当然，所有奇数 int 或 char 的最低有效位都以 1 结尾，所有偶数 int 或 char 的最低有效位都以 O 结尾。简单来说，我们可以使用模数运算和条件语句（即 % 2 = = 1）来阻止 矿
int 或 char 是奇数还是偶数。然后您可以编写以下 C 代码来计算 int 或 char 中 1 的数量。

```
unsigned pop_count(unsigned in) {
  unsigned count = 0;
  for( ; in != 0; in = in >> 1) {
    if(in % 2 == 1) {
      count++;
    }
  }
  return count;
}
```

不幸的是，正如你稍后将在课程中了解到的，条件语句可能很慢，模数运算也很慢 此代码的更快版本将使用如下所示的按位逻辑运算。&1 只是一个一位掩码，当 in 的最低有效位为 1 时，它加 1，否则加 O。如果您不完全理解为什么，请尝试按位 1110 和 1111 与 0001 和

然后将 1100 1101 与 0001 相加，看看会发生什么。如果你还没有完全理解这个想法，不要担心，我们将重新讨论 bitwis 埃
随着我们学习到更多知识，我们稍后会再次讨论逻辑运算符。

```
无符号 pop_count（无符号 in）{无符号计数
= 0; for（; in != 0; in = in » 1）n,'f8 Atc 0
iii [tn & l):                {

}
```

这是另一个例子，说明如何创造性地对信息进行编码，并使用按位逻辑运算来实现疯狂的 鳕鱼 埃
性能改进（从这里开始了解精彩内容，但我们建议观看整个过程）

逻辑运算 鳕鱼 s埃

我们几乎只使用逻辑运算符来创建复合条件语句，例如 if (x = = 0 && y = = 1)。
是啊...就是这样！

Verilog 参考 桂 德

当我们要求您编写 Verilog 代码时，我们将为您提供 Verilog 参考指南和 Verilog 样式 gui 德
在适当的时候。如果你愿意，你现在就可以阅读，不过在需要的时候再参考这些也是完全没问题的。

► Verilog 参考 桂 德

► Verilog 样式指南 埃

补充可选阅读：Mano & Kime 第 4 版，2。 1

标记为真实 dI

" 我