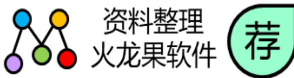
Verilog红宝书_基本语法

资料整理
火龙果软件
荐

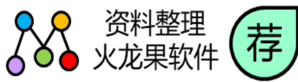
目 录

1. 前途+钱途
2. 什么是HDL?
3. 为什么使用HDL?
4. VHDL还是Verilog?
5. Verilog的历史
6. Verilog的用途
7. Verilog设计层次
8. Verilog一个具体例子



目 录

9. Verilog基本语法-标识符
10. Verilog基本语法-注释
11. Verilog基本语法-数字
12. Verilog基本语法-数据类型
13. Verilog基本语法-运算符&表达式
14. Verilog基本语法-条件语句
15. Verilog基本语法-case语句
16. Verilog建模方式



学习Verilog的目的：前途+钱途

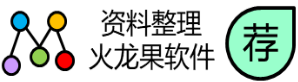
- 1、ASIC设计方向

  当前ASIC设计方向待遇都相对比较高，入职（硕士）时候8000+都很正常。

- nVidia 10000左右。
- 华为 8000左右。
- MTK 8000左右。

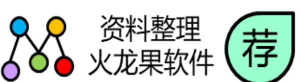
- 2、FPGA设计方向

@ZLG-周立功 V
做FPGA的应届研究生请注意：8000元月薪求人才，开发高端分析仪器，工作地点：广州，联系人：周立功，邮箱：zlg3@zlgmcu.com。
2月19日 10:59 来自iPhone客户端 (3) | 转发(188) | 评论(73)



什么是HDL（硬件描述语言）？

- 具有特殊结构能够对硬件逻辑电路的功能进行描述的一种高级编程语言
- 这种特殊结构能够：
  - 描述电路的连接
  - 描述电路的功能
  - 在不同抽象级上描述电路
  - 描述电路的时序
  - 表达具有并行性
- HDL主要有两种：Verilog和VHDL
  - Verilog起源于C语言，因此非常类似于C语言，容易掌握，但是思想和C语言完全不同。
  - VHDL起源于ADA语言，格式严谨，不易学习，VHDL出现较晚，但标准化早，IEEE 1076-1987标准。



为什么使用HDL？

- 使用HDL描述设计具有下列优点：
  - 设计在高层次进行，与具体实现无关
  - 设计开发更加容易
  - 早在设计期间就能发现问题
  - 能够自动的将高级描述映射到具体工艺实现
- HDL具有更大的灵活性
  - 可重用
  - 可以选择工具及生产厂
- HDL能够利用先进的软件（FPGA：Quartus/ISE/Synplify， ASIC:DC）
  - 更快的输入
  - 易于管理

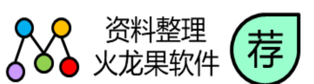
资料整理
火龙果软件
荐

VHDL还是Verilog？

- 很多初学者对于FPGA学习没有方向，之前一个初学者学习了VHDL一个月时间，然后才问我该学习什么VHDL还是Verilog。
- 现在基本所有的芯片和设计全部采用Verilog设计，我本身是6年的芯片设计工程师，设计了几款大型芯片，全部采用Verilog，包括ARM的芯片和IP都是全部采用Verilog，使用VHDL的人非常非常少，只有部分学校在教学，Verilog已经占据了完全主导的地位。

Verilog的历史

- Verilog HDL是在1983年由GDA(GateWay Design Automation)公司的Phil Moorby所创。Phi Moorby后来成为Verilog-XL的主要设计者和Cadence公司的第一个合伙人。
- 在1984~1985年间，Moorby设计出了第一个Verilog-XL的仿真器。
- 1986年，Moorby提出了用于快速门级仿真的XL算法。
- 1990年，Cadence公司收购了GDA公司。
- 1991年，Cadence公司公开发表Verilog语言，成立了OVI(Open Verilog International)组织来负责Verilog HDL语言的发展。
- 1995年制定了Verilog HDL的IEEE标准，即IEEE1364。
- 2001年制定了Verilog HDL的IEEE标准，即IEEE1364-2001。



Verilog的用途

- Verilog的主要应用包括：
  - ASIC和FPGA工程师编写可综合的RTL代码
  - 高抽象级系统仿真进行系统结构开发
  - 测试工程师用于编写各种层次的测试程序（主要是UT）
  - 用于ASIC和FPGA单元或更高层次的模块的模型开发

资料整理
火龙果软件
荐

Verilog设计层次

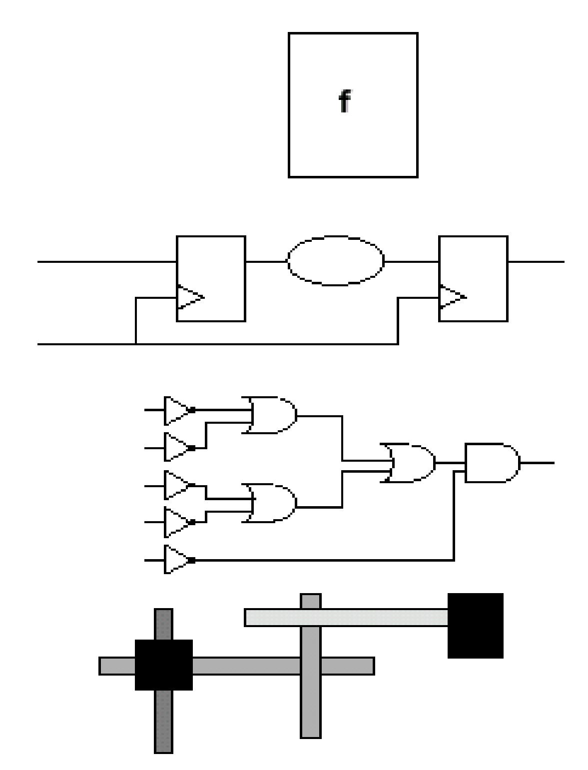
- Verilog既是一种行为描述的语言也是一种结构描述语言。Verilog模型可以是实际电路的不同级别的抽象。这些抽象的级别包括：

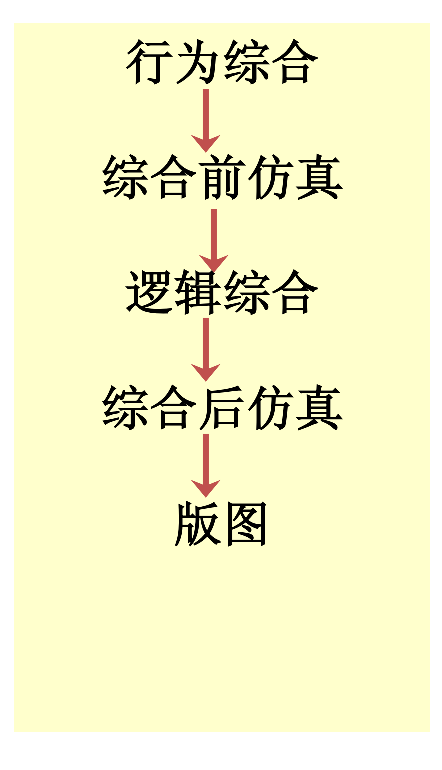
系统说明
-需求/设计文档/算法描述

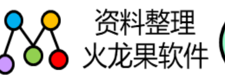
RTL/功能级
-Verilog

门级/结构级
-Verilog

版图/物理级
-几何图形







资料整理
火龙果软件
荐

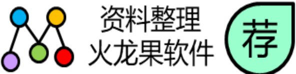
Verilog设计层次

- 在抽象级上需要进行折衷

输入/仿真速度
高
↓
低

系统说明 -需求/设计文档/算法描述
RTL/功能级 -Verilog
门级/结构级 -Verilog
版图/物理级 -几何图形

详细程度
低
↓
高



Verilog一个实际例子

```
module LED (
                  //input
input             sys_clk          ,    //system clock;
input             sys_rst_n        ,    //system reset, low is active;

                  //output
output reg [7:0]  LED
               );


//Parameter define
parameter WIDTH = 8                          ;
parameter SIZE  = 8                          ;
parameter WIDTH2 = 18                        ;
parameter Para = 100000                      ;

//Reg define
reg    [SIZE-1:0]        counter             ;
reg    [WIDTH2-1:0]      count               ;

//Wire define

//**************************************************************************
//**                             Main Program
//**
//**************************************************************************

// count for add counter
always @(posedge sys_clk or negedge sys_rst_n) begin
        if (sys_rst_n ==1'b0)
            count <= 18'b0;
        else
            count <= count + 18'b1;
end
```

```
// counter for delay time to LED display
always @(posedge sys_clk or negedge sys_rst_n) begin
        if (sys_rst_n ==1'b0)
            counter <= 8'b0;
        else if ( count == Para)
            counter  <= counter + 8'b1;
        else ;
end

// ctrl LED pipeline display when counter is equal 10 or 20 .
always @(posedge sys_clk or negedge sys_rst_n) begin
        if (sys_rst_n ==1'b0)
            LED <= 8'b0;
        else begin
            case (counter)
                8'd10     : LED <= 8'b10000000   ;
                8'd20     : LED <= 8'b01000000   ;
                8'd30     : LED <= 8'b00100000   ;
                8'd40     : LED <= 8'b00010000   ;
                8'd50     : LED <= 8'b00001000   ;
                8'd60     : LED <= 8'b00000100   ;
                8'd70     : LED <= 8'b00000010   ;
                8'd80     : LED <= 8'b00000001   ;
                default   : LED <= 8'b00000000   ;
            endcase
        end
end

endmodule
//end of RTL code
```

要点：

1、module名字

2、输入输出

3、信号定义

4、功能逻辑

Verilog基本语法-标识符

- 标识符(identifier）用于定义模块名、端口名、信号名等。
- Verilog HDL 中的标识符(identifier)可以是任意一组字母、数字、$符号和_(下划线)符号的组合，但标识符的第一个字符必须是字母或者下划线。另外，标识符是区分大小写的。
- 以下是标识符的几个例子：

  Count
  COUNT //与Count 不同。
  R56_68
  FIVE$

- 虽然标识符写法很多，但是推荐写法如下：

  count
  fifo_wr

说明：

1、Verilog HDL定义了一系列保留字，叫做关键词，这些关键字不能用于标识符。
2、信号命名最好体现信号的含义，简洁、清晰、易懂。
3、不建议大小写混合使用，普通内部信号建议全部小写，输入输出PAD建议大写。

Verilog基本语法-注释

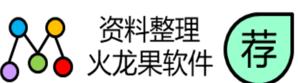
- Verilog HDL中有两种注释的方式，一种是以“/*”符号开始，“*/” 结束，在两个符号之间的语句都是注释语句，因此可扩展到多行。

  如：
  /* statement1 ，
  statement2，
  .. ...
  statementn */
  以上n个语句都是注释语句。

- 另一种是以//开头的语句，它表示以//开始到本行结束都属于注释语句。

建议写法：
1、使用//作为注释。
2、建议全部使用英文注释。

```
// 注释的例子
// counter for gen a clk_50k : need count to 1000, for 50M/1000 = 50K hz
always @(posedge sys_clk or negedge sys_rst_n) begin
        if (sys_rst_n ==1'b0)
            counter_div <= 10'b0;
        else if (counter_div >= 10'd999)
            counter_div <= 10'b0;
        else
            counter_div <= counter_div + 10'b1;
end
```



Verilog基本语法-数字

- 数字包括值集合、常量（整型、实型、字符型）和变量等。

- 值集合

Verilog HDL中规定了四种基本的值类型：
0：逻辑0或“假”；
1：逻辑1或“真”；
X：未知值；
Z：高阻。

```
// 一个TCAM匹配使用"X"的例子
always@(*) begin
    case (hit_vld )
    32'b1xxx_xxxx_xxxx_xxxx_xxxx_xxxx_xxxx_xxxx: addr = 8'd0 ;
    32'b01xx_xxxx_xxxx_xxxx_xxxx_xxxx_xxxx_xxxx: addr = 8'd1 ;
    32'b001x_xxxx_xxxx_xxxx_xxxx_xxxx_xxxx_xxxx: addr = 8'd2 ;
    32'b0001_xxxx_xxxx_xxxx_xxxx_xxxx_xxxx_xxxx: addr = 8'd3 ;
    32'b0000_1xxx_xxxx_xxxx_xxxx_xxxx_xxxx_xxxx: addr = 8'd4 ;
    32'b0000_01xx_xxxx_xxxx_xxxx_xxxx_xxxx_xxxx: addr = 8'd5 ;
    32'b0000_001x_xxxx_xxxx_xxxx_xxxx_xxxx_xxxx: addr = 8'd6 ;
    32'b0000_0001_xxxx_xxxx_xxxx_xxxx_xxxx_xxxx: addr = 8'd7 ;
    default                                    : addr = 8'd255 ;
  endcase
end
```

注意这四种值的解释都内置于语言中。如一个为z的值总是意味着高阻抗，一个为0 的值通常是指逻辑0 。

在门的输入或一个表达式中的为“z ”的值通常解释成“x ”。

此外，x 值和z 值都是不分大小写的，也就是说，值0x1z与值0X1Z 相同。

Verilog HDL 中的常量是由以上这四类基本值组成的。

```
// 一个I2C接口使用"Z"的例子
// output sda data when sda enable is 1, else output Z state
assign sda_port = ( enable == 1'b1 )?  sda : 1'bz;
```

Verilog基本语法-数字

- 常量

Verilog HDL中有三种常量：
整型、实型、字符串型。
下划线符号“_”可以随意用在整数或实数中，它们就数量本身没有意义。
它们能用来提高易读性；唯一的限制是下划线符号不能用作为首字符。

下面主要介绍整型和字符串型。
1. 整型
整型数可以按如下两种方式书写：
1) 简单的十进制数格式
2) 基数格式
A. 简单的十进制格式
这种形式的整数定义为带有一个可选的“+”（一元）或“—”（一元）
操作符的数字序列。
下面是这种简易十进制形式整数的例子。
32 十进制数32
—15 十进制数—15

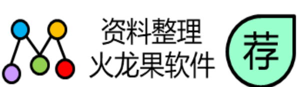
实际项目中Paramter参数一般使用十进制。

```
//Parameter define
parameter WIDTH = 8                  ;
parameter SIZE  = 8                  ;
```

Verilog基本语法-数字

B. 基数表示法
这种形式的整数格式为：
[size] 'base value
size 定义以位计的常量的位长；
base 为o或O（表示八进制），b或B（表示二进制），d或D（表示十进制），h 或H （表示十六进制）之一；
value 是基于base 的值的数字序列。值x和z以及十六进制中的a 到f 不区分大小写。
下面是一些具体实例：
5 'o37 5位八进制数（二进制11111）
4 'd2 4 位十进制数（二进制0011）
4 'b1x_01 4 位二进制数
7 'hx 7位x(扩展的x), 即xxxxxxx
4 'hz 4 位z(扩展的z) , 即zzzz
4 'd-4 非法：数值不能为负
3' b 001 非法：和基数b之间不允许出现空格
(2+3)'b10 非法：位长不能够为表达式

实际项目中使用十进制、二进制和十六进制较多，八进制使用较少。



Verilog基本语法-数字

注意：

x（或z）在十六进制值中代表4位x（或z），在八进制中代表3位x（或z ），在二进制中代表1位x（或z）。

基数格式计数形式的数通常为无符号数。这种形式的整型数的长度定义是可选的。如果没有定义一个整数型的长度，数的长度为相应值中定义的位数。下面是两个例子：

'o 7219 位八进制数

'h AF8 位十六进制数

```
// 位宽的例子
// counter for gen a clk_50k : need count to 1000, for 50M/1000 = 50K hz
always @(posedge sys_clk or negedge sys_rst_n) begin
        if (sys_rst_n ==1'b0)
            counter_div <= 10'b0;
        else if (counter_div >= 10'd999)
            counter_div <= 10'b0;
        else
            counter_div <= counter_div + 10'b1;
end
```

规范建议：

不建议这种写法，会导致nLint等代码检查工具报位宽不匹配告警。

如果定义的长度比为常量指定的长度长，通常在左边填0补位。但是如果数最左边一位为x 或z，就相应地用x 或z 在左边补位。例如：

10'b10左边添0占位,0000000010

10'bx0x1左边添x占位, xxxxxxx0x1

如果长度定义得更小，那么最左边的位相应地被截断。例如：

3 ' b1001_0011与3'b011相等

5'H0FFF与5'H1F相等

Verilog基本语法-数字

2. 字符串型

字符串是双引号内的字符序列。字符串不能分成多行书写。

例如：

"INTERNAL ERROR"

" REACHED－>HERE "

用8 位ASCII值表示的字符可看作是无符号整数。因此字符串是8 位ASCII值的序列。为存储字符串“INTERNAL ERROR ”，变量需要8*14位。

```
reg [1:8*14] Message;
. . .
Message = "INTERNAL ERROR“
```

说明：

这种一般使用较少，本人经历的项目里面还没有使用过这种字符串型。

资料整理
火龙果软件
荐

Verilog基本语法-数据类型

Verilog HDL 主要包括两种数据类型:

线网类型(net type) 和 寄存器类型（reg type）。

线网类型

1. wire 和 tri 定义

线网类型主要有wire 和tri 两种。线网类型用于对结构化器件之间的物理连线的建模。如器件的管脚，内部器件如与门的输出等。以上面的加法器为例，输入信号A，B是由外部器件所驱动，异或门X1的输出S1是与异或门X2输入脚相连的物理连接线，它由异或门X1所驱动。

由于线网类型代表的是物理连接线，因此它不存贮逻辑值。必须由器件所驱动。通常由assign进行赋值。如 assign A = B ^ C;

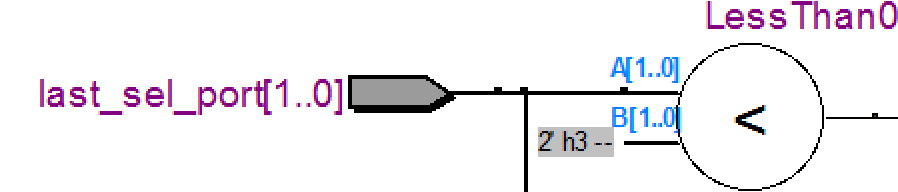
当一个wire 类型的信号没有被驱动时，缺省值为Z（高阻）。

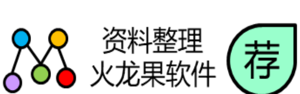
信号没有定义数据类型时，缺省为 wire 类型。

如上面一位全加器的端口信号 A，B，SUM等，没有定义类型，故缺省为wire 线网类型。

2. 两者区别

tri主要用于定义三态的线网。





Verilog基本语法-数据类型

寄存器类型

1. 定义

reg 是最常用的寄存器类型，寄存器类型通常用于对存储单元的描述，如D型触发器、ROM等。存储器类型的信号当在某种触发机制下分配了一个值，在分配下一个值之时保留原值。

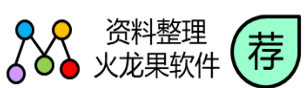
reg 类型定义语法如下：

```
reg [msb: lsb] reg1, reg2, . . . reg N;
```

msb 和lsb 定义了范围，并且均为常数值表达式。范围定义是可选的；如果没有定义范围，缺省值为1 位寄存器。例如：

```
reg [3:0] Sat; // Sat 为4 位寄存器。
reg Cnt; //1 位寄存器。
reg [1:32] Kisp, Pisp, Lisp ;
```

寄存器类型的值可取负数，但若该变量用于表达式的运算中，则按无符号类型处理。



Verilog基本语法-数据类型

如：

```
reg A ;
.....
A = -1;
....
```

则A的二进制为1111，在运算中，A总按 无符号数15 来看待。

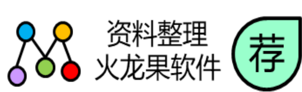
2. 寄存器类型的存储单元建模举例

用寄存器类型来构建两位的D触发器如下：

```
reg [1： 0] dout ;
.....
always@(posedge clk)
    dout <= din;
....
```

用寄存器数组类型来建立存储器的模型，

实际项目中经常使用寄存器搭建小的RAM使用。



Verilog基本语法-数据类型

如对2个8位的RAM建模如下：

reg [7：0] mem[0：1] ；

对存储单元的赋值必须一个个的赋值，如上2个8位的RAM的赋值必须用两条赋值语句：

.....

mem[0] = ’h 55;

mem[1] = ’haa;

....

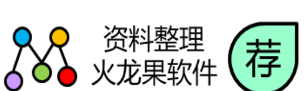
3. 书写规范建议

对数组类型，请按降序方式，如[7：0] ；

说明：

reg的类型不一定是寄存器，只有带有时钟的always块才能是寄存器，不带时钟的always块是组合逻辑。

```
// always 综合出来的不一定是寄存器
always@(*) begin
   case (hit_vld )
   32'b1xxx_xxxx_xxxx_xxxx_xxxx_xxxx_xxxx_xxxx: addr = 8'd0 ;
   32'b01xx_xxxx_xxxx_xxxx_xxxx_xxxx_xxxx_xxxx: addr = 8'd1 ;
   default                                    : addr = 8'd255 ;
  endcase
end
```



Verilog基本语法-运算符

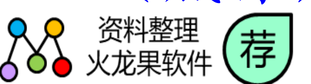
Verilog HDL中的操作符可以分为下述类型：

1) 算术操作符
2) 关系操作符
3) 相等操作符
4) 逻辑操作符
5) 按位操作符
6) 归约操作符
7) 移位操作符
8) 条件操作符
9) 连接和复制操作符

+	一元加	>>	右移
-	一元减	<	小于
!	一元逻辑非	< =	小于等于
~	一元按位求反	>	大于
&	归约与	>=	大于等于
~&	归约与非	==	逻辑相等
^	归约异或	!=	逻辑不等
^~ 或 ~^	归约异或非	===	全等
|	归约或	!==	非全等
~|	归约或非	&	按位与
*	乘	^	按位异或
/	除	^~ or ~^	按位异或非
%	取模	|	按位或
+	二元加	&&	逻辑与
-	二元减	||	逻辑或
<<	左移	?:	条件操作符

说明：

操作符从最高优先级（顶行）到最低优先级（底行）排列。同一行中的操作符优先级相同。



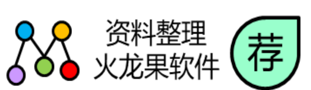
Verilog基本语法-运算符

算术运算符

在常用的算术运算符主要是 ：
加法（二元运算符）： “+” ;
减法（二元运算符）： “-” ;
乘法（二元运算符）： “*” ;
在算术运算符的使用中，注意如下两个问题：
1. 算术操作结果的位数长度
算术表达式结果的长度由最长的操作数决定。在赋值语句下，算术操作结果的长度由操作符左端目标长度决定。考虑如下实例：

```
reg [3:0] arc, bar, crt;
reg [5:0] frx;
...
arc = bar + crt;
frx = bar + crt;
```

实际项目中经常“*”乘法，使用时候可以直接写A*B，ASIC综合器会综合为专用乘法器，FPGA会综合为硬件乘法器。



Verilog基本语法-运算符

第一个加的结果长度由bar， crt 和arc 长度决定，长度为4 位。

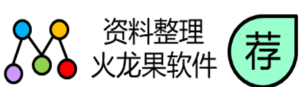
第二个加法操作的长度同样由frx 的长度决定（frx 、bat 和crt 中的最长长度），长度为6位。

在第一个赋值中，加法操作的溢出部分被丢弃；而在第二个赋值中，任何溢出的位存储在结果位frx [4]中。

在较大的表达式中，中间结果的长度如何确定？在Verilog HDL 中定义了如下规则：表达式中的所有中间结果应取最大操作数的长度（赋值时，此规则也包括左端目标）。

考虑另一个实例：

```
wire [4:1] box, drt;
wire [5:1] cfg;
wire [6:1] peg;
wire [8:1] adt;
. . .
assign adt = (box + cfg) + (drt + peg) ;
```

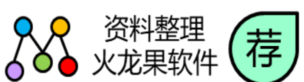


Verilog基本语法-运算符

表达式右端的操作数最长为6，但是将左端包含在内时，最大长度为8。所以所有的加操作使用8位进行。例如：box 和cfg 相加的结果长度为8位。

2. 有符号数和无符号数在设计中，请先使用无符号数。

算法类项目会使用有符号数，控制类项目一般不使用有符号数。

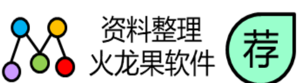


Verilog基本语法-运算符

关系运算符

关系运算符有：
>（大于）
<（小于）
>=（不小于）
<=（不大于）
= = （逻辑相等）
！ = （逻辑不等）
关系操作符的结果为真（1）或假（0）。如果操作数中有一位为X或Z ，那么结果为X 。
例：
23 > 45 结果为假（0），
而：
52 < 8'hxFF 结果为x 。

实际项目中除了芯片/FPGA接口，模块内部禁止带有“Z”。



Verilog基本语法-运算符

如果操作数长度不同，长度较短的操作数在最重要的位方向（左方）添0补齐。例如：

'b1000 > = 'b01110

等价于：

'b01000 > = 'b01110

结果为假（0）。

在逻辑相等与不等的比较中，只要一个操作数含有x 或z，比较结果为未知（x），如：

假定：

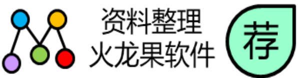
Data = 'b11x0;

Addr = 'b11x0;

那么：

Data == Addr 比较结果不定，也就是说值为x 。

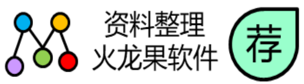
实际项目中除了比较禁止使用 "X" 。



Verilog基本语法-运算符

逻辑运算符

逻辑运算符有：
&&（逻辑与）
||（逻辑或）
!（逻辑非）
用法为：（表达式1） 逻辑运算符 （表达式2） ....
这些运算符在逻辑值0（假） 或1（真） 上操作。逻辑运算的结果为0 或1 。例如，假定：
crd = 'b0; //0 为假
dgs = 'b1; //1 为真
那么：
crd && dgs 结果为0（假）
crd || dgs 结果为1（真）
! dgs 结果为0（假）



Verilog基本语法-运算符

逻辑与（&&)的真值表如下：

表1 逻辑与真值表

&&	0（假）	1（真）	X/Z（不定）
0(假）	0	0	x
1（真）	0	1	x
X/Z（不定）	x	x	x

逻辑或的真值表如下：

表1 逻辑与真值表

||	0（假）	1（真）	x/z（不定）
0	0	1	x
1	1	1	1
x/z（不定）	x	1	x

资料整理
火龙果软件
荐

Verilog基本语法-运算符

按位逻辑运算符

按位运算符有：

~（一元非）：（相当于非门运算）

&（二元与）：（相当于与门运算）

|（二元或）：（相当于或门运算）

^（二元异或）：（相当于异或门运算）

~^，^~（二元异或非即同或）：（相当于同或门运算）

这些操作符在输入操作数的对应位上按位操作，并产生向量结果。

下表显示对于不同按位逻辑运算符按位操作的结果：

& 与	0	1	x	z
0	0	0	0	0
1	0	1	x	x
x	0	x	x	x
z	0	x	x	x

| 或	0	1	x	z
0	0	1	x	x
1	1	1	1	1
x	x	1	x	x
z	x	1	x	x

^ 异或	0	1	x	z
0	0	1	x	x
1	1	0	x	x
x	x	x	x	x
z	x	x	x	x

^~异或非	0	1	x	z
0	1	0	x	x
1	0	1	x	x
x	x	x	x	x
z	x	x	x	x

~ 非	0	1	x	z
	1	0	x	x

实际项目中非、与、或、异或使用较多，非常普遍。

Verilog基本语法-运算符

例如，假定,

A = 'b0110;

B = 'b0100;

那么：

A | B 结果为0110

A & B 结果为0100

如果操作数长度不相等, 长度较小的操作数在最左侧添0补位。

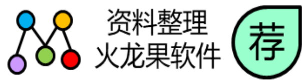
例如,

'b0110 ^ 'b10000

与如下式的操作相同：

'b00110 ^ 'b10000

结果为'b10110。



Verilog基本语法-运算符

条件运算符

条件操作符根据条件表达式的值选择表达式，形式如下：
cond_expr ? expr1 : expr2
如果cond_expr 为真(即值为1)，选择expr1 ；如果cond_expr 为假(值为0)，选择expr2 。如果cond_expr 为x 或z ，结果将是按以下逻辑expr1 和expr2 按位操作的值： 0 与0 得0 ，1 与1 得1 ，其余情况为x 。
如下所示:
wire [2:0] student = marks > 18 ? grade_a : grade_c;
计算表达式marks > 18; 如果真,grade_a 赋值为student; 如果marks < =18, grade_c 赋值为student 。

```
// 一个使用条件运算符的例子
// output sda data when sda enable is 1, else output Z state
assign sda_port = ( enable == 1'b1 )?  sda : 1'bz;
```

不建议使用条件预算符嵌套条件运算符，会导致代码非常难读和维护，F= (A?B:C)?D:E。

Verilog基本语法-运算符

连接运算符

连接操作是将小表达式合并形成大表达式的操作。形式如下：

{expr1, expr2, . . ., exprN}

实例如下所示：

```
wire [7:0] Dbus;
assign Dbus [7:4] = {Dbus [0], Dbus [1], Dbus[2],
Dbus[ 3 ]};
//以反转的顺序将低端4 位赋给高端4 位。
assign Dbus = {Dbus [3:0], Dbus [7:4]};
//高4 位与低4 位交换。
```

由于非定长常数的长度未知，不允许连接非定长常数。例如，下列式子非法：

{Dbus,5} //不允许连接操作非定长常数。

```
// 连接符例子
assign LED = { 6'b0, port_win } ;
```

Verilog基本语法-条件语句

if 语句的语法如下：

if(condition_1)

procedural_statement_1

{else if(condition_2)

procedural_statement_2}

{else

procedural_statement_3}

以下是一个例子。

```
if (sum < 60) begin
    grade = c;
    total_c = total _c + 1;
end
else if(sum < 75) begin
    grade = b;
    total_b = total_b + 1;
end
else begin
    grade = a;
    total_a = total_a + 1;
end
```

如果对condition_1 求值的结果为个非零值， 那么procedural_statement_1 被执， 如果condition_1 的值为0 、x 或z ， 那么procedural_statement_1 不执行。如果存在一个else分支，那么这个分支被执行。

资料整理 火龙果软件 荐

Verilog基本语法-运算符

注意条件表达式必须总是被括起来，如果使用if - if - else 格式，那么可能会有二义性，如下例所示：

```
if (clk == 1)
if (reset== 1)
q = 0;
else
q = d;
```

问题是最后一个else 属于哪一个if？它是属于第一个if 的条件(clk)还是属于第二个if的条件(reset)？这在Verilog HDL中已通过将else 与最近的没有else 的if 相关联来解决。在这个例子中，else 与内层if 语句相关联。

资料整理
火龙果软件
荐

Verilog基本语法-运算符

规范建议：

1、条件表达式需用括号括起来。

2、若为if－if 语句，请使用块语句 begin --- end ：

```
if(clk == 1) begin
    if (reset== 1)
      q = 0;
    else
      q = d;
end
```

以上两点建议是为了使代码更加清晰，防止出错。

3、对if 语句，除了在时序逻辑中，if 语句都需要有else语句。若没有缺省语句，设计将产生一个锁存器，锁存器在ASIC设计中有诸多的弊端（无法进行DFT测试、无法进行STA分析）。

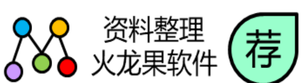
如下一例：

if （t == 1）

q = d;

没有else 语句，当t为1（真）时，d 被赋值给q，当t为0（假）时，因为没有else 语句，电路保持 q 以前的值，这就形成一个锁存器。

说明：if/else语句有优先级的概念，是和case语句最大的差别。



荐

Verilog基本语法-运算符

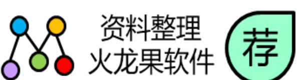
case 语句

case 语句是一个多路条件分支形式，其语法如下：

```
case(case_expr)
case_item_expr{ ,case_item_expr} :procedural_statement
. . .
. . .
[default:procedural_statement]
endcase
```

case 语句首先对条件表达式case_expr 求值，然后依次对各分支项求值并进行比较，第一个与条件表达式值相匹配的分支中的语句被执行。可以在1 个分支中定义多个分支项；这些值不需要互斥。缺省分支覆盖所有没有被分支表达式覆盖的其他分支。

说明： case语句的分支没有优先级的概念，是和if语句最大的差别。



Verilog基本语法-运算符

例：

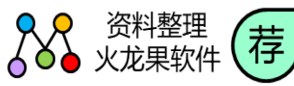
```
case (HEX)
4'b0001 : led = 7'b1111001; // 1
4'b0010 : led = 7'b0100100; // 2
4'b0011 : led = 7'b0110000; // 3
4'b0100 : led = 7'b0011001; // 4
4'b0101 : led = 7'b0010010; // 5
4'b0110 : led = 7'b0000010; // 6
4'b0111 : led = 7'b1111000; // 7
4'b1000 : led = 7'b0000000; // 8
4'b1001 : led = 7'b0010000; // 9
4'b1010 : led = 7'b0001000; // a
4'b1011 : led = 7'b0000011; // b
4'b1100 : led = 7'b1000110; // c
4'b1101 : led = 7'b0100001; // d
4'b1110 : led = 7'b0000110; // e
4'b1111 : led = 7'b0001110; // f
default  : led = 7'b1000000; // 0
```

规范建议：

case 的缺省项必须写，防止产生锁存器。

说明：

只有组合逻辑才可能产生锁存器，时序逻辑不会产生锁存器。



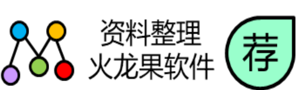
总　结

1、Verilog基本语法和C语言有点类似，但是思想却完全不同。C语言编译的结果是指令，而Verilog编译的结果是电路，需要使用电路设计的思想去写Verilog。

2、大家要多看波形，建立时序概念，熟练画时序图进行时序设计。

3、现在绝对主流是Verilog进行数字设计。

4、Verilog最重要的是方案设计，掌握基本语法即可。



Verilog建模方式

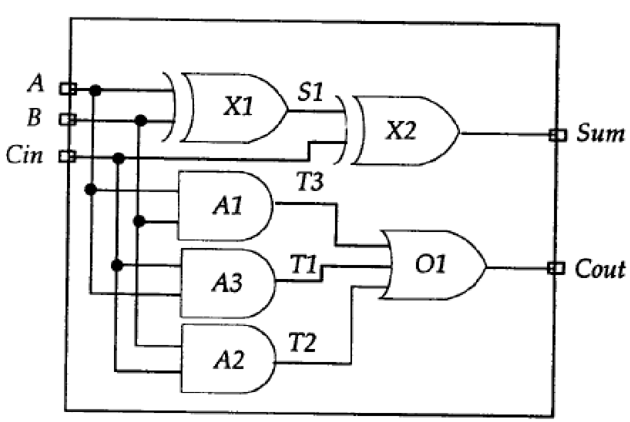
- 结构级描述方式
  - 用基本单元(primitive)或低层元件(component)的连接来描述系统以得到更高的精确性，特别是时序方面。
  - 在综合时用特定工艺和低层元件将RTL描述映射到门级网表
- 数据流描述方式
  - 通过对数据流在设计中的具体行为的描述的来建模
  - 一般采用连续赋值语句
- 行为级描述方式
  - 采用对信号行为级的描述的方法来建模
  - 一般是always 块语句和assign块语句描述的归为行为建模方式

资料整理
火龙果软件
荐

结构级描述

- 结构级**Verilog**适合开发小规模元件，如**ASIC**和**FPGA**的单元
  - **Verilog**内部带有描述基本逻辑功能的基本单元**(primitive)**，如**and**门。
  - 综合产生的结果网表通常是结构级的。
- 下面是一位全加器的结构级描述，采用**Verilog**基本单元**(**门**)**描述。

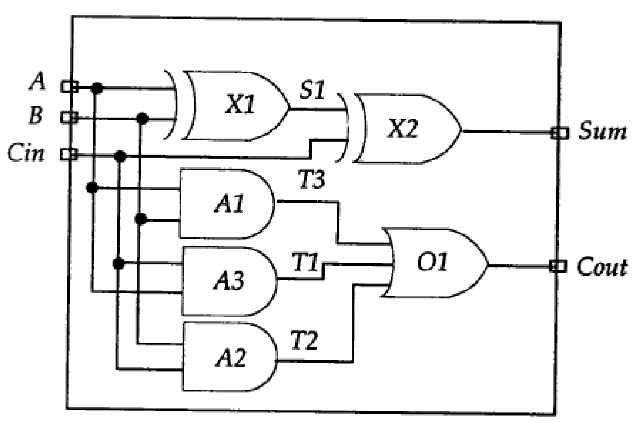
```
module FA_struct (
input A,
input B,
input Cin,
output Sum,
output Count
);
wire S1, T1, T2, T3;

 xor x1 (S1, A, B);
xor x2 (Sum, S1, Cin);
and A1 (T3, A, B );
and A2 (T2, B, Cin);
and A3 (T1, A, Cin);
or O1 (Cout, T1, T2, T3 );

 endmodule
```



数据流描述

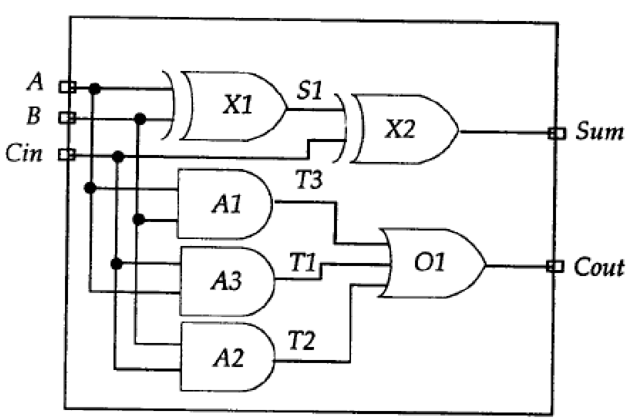
- 数据流的建模方式就是通过对数据流在设计中的具体行为的描述的来建模。最基本的机制就是用连续赋值语句。
- 在连续赋值语句中，某个值被赋给某个线网变量（信号），语法如下：assign [delay] net_name = expression; 如：assign #2 A = B;
- 在数据流描述方式中，还必须借助于HDL提供的一些运算符，如按位逻辑运算符 ：逻辑与（&)，逻辑或（|）等。
- **下面是一位全加器的数据流级描述。**

```
module FA_flow(
input A,
input B,
input Cin,
output wire Sum,
output wire Count
);
wire S1,T1,T2,T3;
assign # 2 S1 = A ^ B;
assign # 2 Sum = S1 ^ Cin;
assign #2 T3 = A & B;
assign #2 T1 = A & Cin;
assign #2 T2 = B & Cin ;
endmodule
```



行为级别描述

- 行为方式的建模是指采用对信号行为级的描述的方法来建模。
- 在表示方面，类似数据流的建模方式，但一般是always 块语句和assign块语句描述的归为行为建模方式。
- 行为建模方式通常需要借助一些行为级的运算符如加法运算符（+），减法运算符（-）等。
- **下面是一位全加器的行为级描述。**




```
module FA_BEHAV1(
input a,
input b,
input cin,
output sum,
output cout
);
reg sum, cout;
reg t1,t2,t3;

always@ ( a or b or cin )
begin
sum = (a ^ b) ^ cin ;
t1 = a & cin;
t2 = b & cin ;
t3 = a & b;
cout = (t1| t2) | t3;
end
```


资料整理
火龙果软件
荐

建模方式总结

- 当前数字逻辑规模越来越大，使用结构化描述已经越来越不现实，也没有必要，当前行为描述方式已经占据绝对的主导地位，大家学习的时候知道有这两种方式即可，重点学习行为描述。
- 我们经历了很多个芯片项目，全部使用的是行为模式方式。

资料整理
火龙果软件
荐