

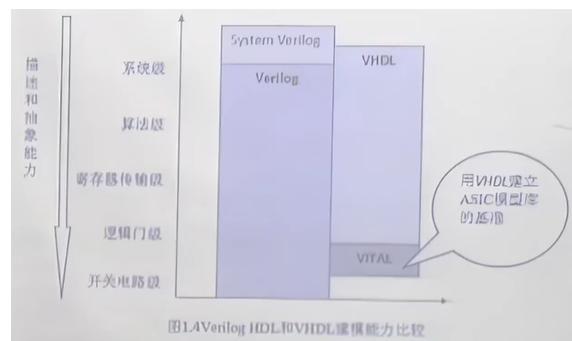
# CHAPTER I 导论

## 数字电路的发展与设计方法演变

### HDL语言的产生与发展

Verilog（语法宽松）与VHDL(思维更加严谨，语法更加严格，大规模应用更占优势)---->联合工作

1995, 2001标准应用较多！



```
module aand4(a,b,c);
    input [3:0]a,b;
    output [3:0]c;
    reg[3:0]c;
    always @(a or b)
        c = a&b;
endmodule
```

# CHAPTER II Verilog语言要素和数据类型

空白符： \b 代表空格符， \t 代表制表符，还有换行符和换页符，编译和综合时空白符被忽略！

注释和C相同！EDA工具一定要求英文！

标识符：字母，数字，\$和\_的组合。第一个字母必须是字母或者下划线

转义标识符:以\开头,以空白(或者制表符等)结尾

关键字

数值:基本数值状态: 1 0 X(未知) Z(高阻态)

整数及其表示:`+/-<size>'<base_format><number>`

二进制,八进制,十进制,十六进制分别用**b**,**o**,**d**,**h**来表示,也可以大写

数值以补码的形式出现!

### • 整数及其表示 `+/-<size>'<base_format><number>`

| 数制   | 基数符号 | 合法表示符                   |
|------|------|-------------------------|
| 二进制  | b或B  | 0、1、x、X、z、Z、?、_         |
| 八进制  | o或O  | 0~7、x、X、z、Z、?、_         |
| 十进制  | d或D  | 0~9、_                   |
| 十六进制 | h或H  | 0~9、a~f、A~F、x、X、z、Z、?、_ |

例2.1-6: 正确的表示

`8'b10001101` //位宽为8位的二进制数10001101

`8'ha6` //位宽为8位的十六进制数a6

`5'o35` //5位八进制数35

`4'd6` //4位十进制数6

`4'b1x_01` //4位二进制数1x01

例2.1-7: 错误的表示

`4'd-4` //数值不能为负,有负号应放最左边

`3'b001` //和基数b之间不允许出现空格

`(4+4)'b11` //位宽不能是表达式形式

表示浮点数与定点数

实数不能直接用于设计,小数点两边不能缺少值

数据类型

根据电流的强度来衡量!

物理数据类型:连线型、寄存器型和存储器型数据类型;

信号强度表示数字电路中不同强度的驱动源，用来解决不同驱动强度存在下的赋值冲突：

| 标记符    | 名称    | 类型 | 驱动强度 |
|--------|-------|----|------|
| supply | 电源级驱动 | 驱动 | 最强   |
| strong | 强驱动   | 驱动 | 强弱   |
| pull   | 上拉级驱动 | 驱动 |      |
| large  | 大容性   | 存储 |      |
| weak   | 弱驱动   | 驱动 |      |
| medium | 中性驱动  | 存储 |      |
| small  | 小容性   | 存储 |      |
| highz  | 高容性   | 高阻 |      |

驱动能力不同，则器件形式不同

## 1. 连线型

| 连线型数据类型      | 功能说明               |
|--------------|--------------------|
| wire, tri    | 标准连线（缺省为该类型）       |
| wor, trior   | 多重驱动时，具有线或特性的连线型   |
| wand, triand | 多重驱动时，具有线与特性的连线型   |
| trireg       | 具有电荷保持特性的连线型数据（特例） |
| tri1         | 上拉电阻               |
| tri0         | 下拉电阻               |
| supply1      | 电源线，用于对电源建模，为高电平1  |
| supply0      | 电源线，用于对“地”建模，为低电平0 |

## 1) wire和tri

| wire/tri | 0 | 1 | x | z |
|----------|---|---|---|---|
| 0        | 0 | x | x | 0 |
| 1        | x | 1 | x | 1 |
| x        | x | x | x | x |
| z        | 0 | 1 | x | z |

## 2) wor和trior

| wor/trior | 0 | 1 | x | z |
|-----------|---|---|---|---|
| 0         | 0 | 1 | x | 0 |
| 1         | 1 | 1 | 1 | 1 |
| x         | x | 1 | x | x |
| z         | 0 | 1 | x | z |

注意比较两者的差别！

reg型是数据储存单元的抽象类型，其对应的硬件电路元件具有状态保持作用，能够存储数据，如触发器、锁存器等。

reg型变量常用于行为级描述，由过程赋值语句对其进行赋值。

reg型变量简单例子：

```
reg a;          //定义一个一位的名为a的reg变量  
reg [3:0] b;    //定义一个4位的名为b的reg型变量  
reg[8:1]c,d,e; //定义了三个名称分别为c、d、e的8位的reg型变量
```

reg型变量一般为无符号数，若将一个负数赋给reg型变量，则自动转换成其二进制补码形式。例如：

```
reg signed[3:0] rega;  
rega=-2;      //rega的值为1110 (14)，是2的补码
```

## 连线型数据类型的声明

```
<net_declaration><drive_strength><range><delay>[<list_of_variables>];
```

- net\_declaration包括 wire、tri、tri0、tri1、wand、\_triand、trior、wor中的任意一种。
- range: 用来指定数据为标量或矢量。若该项默认，表示数据类型为1位的标量，超过1位则为矢量形式。
- delay: 指定仿真延迟时间。
- list\_of\_variables: 变量名称，一次可定义多个名称，之间用逗号分开。
- drive\_strength: 表示连线变量的驱动强度。

- 寄存器型数据类型的声明

```
reg<range><list_of_register_variables>;
```

- range为可选项，它指定了reg型变量的位宽，缺省时为1位。
- <list\_of\_register\_variables>为变量名称列表，一次可以定义多个名称，之间用逗号分开。

存储器型变量可以描述RAM型、ROM型存储器以及reg文件。

存储器变量的一般声明格式为：

```
reg <range1><name_of_register><range2>;
```

- range1和range2都是可选项，缺省都为1。
- <range1>: 表示存储器当中寄存器的位宽，格式为[msb:lsb]。
- <range2>: 表示寄存器的个数，格式为[msb:lsb]，即有msb-lsb+1个。
- <name\_of\_register>为变量名称列表，一次可以定义多个名称，之间用逗号分开。

- reg[7:0] mem1[255:0]; //定义了一个有256个8位寄存器的存储器mem  
//地址范围是0到255。
- reg [15:0]mem2[127:0],reg1,reg2;  
//定义了一个具有128个16位寄存器的存储器mem2  
//和两个16位的寄存器reg1和reg2

#### 例2.2-4:

- reg[n-1:0] a; //表示一个n位的寄存器a
- reg mem1[n-1:0]; //表示一个由n个1位寄存器构成的存储器mem1

- 抽象数据类型主要包括整型（integer）、时间型（time）  
、实型（real）及参数型（parameter）。
- 整型  
`integer<list_of_register_variables>;`
- 例2.2-6：
  - integer index; //简单的32位有符号整数
  - integer i[31: 0] //定义了整型数组，它有32个元素

## 运算符和表达式

| Verilog 运算符                                | 功能                      | 运算符的优先级别 |
|--|-------------------------|----------|
| $\neg$ 、 $\sim$                            | 反逻辑、位反相                 |          |
| $*$ 、 $/$ 、 $\%$                           | 乘、除、取模                  |          |
| $+$ 、 $-$                                  | 加、减                     |          |
| $<<$ 、 $>>$                                | 左移、右移                   |          |
| $<$ 、 $\leq$ 、<br>$>$ 、 $\geq$             | 小于、小于等于、大于、大于等于         |          |
| $==$ 、 $\neq$<br>$\$signed$ 、 $\$unsigned$ | 等、不等、全等、非全等             | 高优先级别    |
| $\&$                                       | 按位与                     |          |
| $\wedge$ 、 $\wedge\sim$                    | 按位逻辑异或和同或               |          |
| $\mid$                                     | 按位逻辑或                   |          |
| $\&\&$                                     | 逻辑与                     |          |
| $\ \ $                                     | 逻辑或                     |          |
| $?$  | 条件选择符，唯一的三目运算符，等同于 C 语句 | 低优先级别    |

## 算术运算符

结果长度由最长的操作数来决定!(除了赋值)

加法 (+) ; 减法 (-) ; 乘法 (\*) ; 除法 (/) ; 取模 (%)。

- (1) 算术操作结果的位宽

- 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左端目标长度决定。

- 例2.3-1:

```
reg[3:0]A,B,C;
reg[5:0]D;
A=B+C;           //4位
D=B+C;           //6位
```

使用例:

```
module arith_tb;
```

```

reg[3:0]a;
reg[2:0]b;
initial
begin
    a=4'b1111;//15
    b=3'b011;//3
    $display("%b",a*b);//乘法运算, 结果为4'b1101. 高位被舍去, 等于45的
低四位
    $display("%b".a/b);//除法运算, 结果为4'b0101
    $display("%b".a+b);//加法运算, 结果为4'b0010
    $display("%b",a-b);//4'b1100
    $display("%b" a%b);//取模运算, 结果为4'b0000
end
endmodule

```

## 关系操作符

- 大于“>”、 小于“<”、 大于等于“>=”和小于等于“<=”。

例2.3-3：

```

module rela_tb;
reg[3:0]a,b,c,d;
initial
begin
    a=3;  b=6;  c=1;  d=4'hx;
    $display(a<b);    //结果为真1
    $display(a>b);    //结果为假0
    $display(a<=c);   //结果为假0
    $display(d<=a);   //结果为未知数x
end
endmodule

```

## 相等关系操作符

- 等于“==”、不等于“!=”、全等“== ==”、非全等“! ==”
- 比较的结果有三种，即真“1”、假“0”和不定值“x”

表2.3-2 (a) “==”运算符的真值表

| == | 0 | 1 | x | z |
|----|---|---|---|---|
| 0  | 1 | 0 | x | x |
| 1  | 0 | 1 | x | x |
| x  | x | x | x | x |
| z  | x | x | x | x |

表2.3-2 (b) “=====”运算符的真值表

| ===== | 0 | 1 | x | z |
|-------|---|---|---|---|
| 0     | 1 | 0 | 0 | 0 |
| 1     | 0 | 1 | 0 | 0 |
| x     | 0 | 0 | 1 | 0 |
| z     | 0 | 0 | 0 | 1 |

例2.3-4:

```
module equal_tb;
reg[3:0]a,b,c,d;
initial
begin
    a=4'b0xx1;
    b=4'b0xx1;
    c=4'b0011;
    d=2'b11;
    $display(a==b); //结果为不定值x
    $display(c==d); //结果为真1
    $display(a==b); //结果为真1
    $display(c=====d); //结果为假0
end
endmodule
```

相等与全等操作符,全等操作符对所有的都有效

利用全等符阻断不定状态的传递!

## 逻辑运算符

- 逻辑与运算符“&&”、逻辑或运算符“||”、逻辑非运算符“!”

| a | b | !a | !b | a&&b | a  b |
|---|---|----|----|------|------|
| 1 | 1 | 0  | 0  | 1    | 1    |
| 1 | 0 | 0  | 1  | 0    | 1    |
| 0 | 1 | 1  | 0  | 0    | 1    |
| 0 | 0 | 1  | 1  | 0    | 0    |

例如：寄存器变量a, b的初值分别为4'b1110和4'b0000，则：  
!a=0, !b=1, a&&b=0; a||b=1。

例如：a的初值分别为4'b1100, b的初值分别为4'b01x0，则!a=0, !b=x, a&&b=x, a||b=x。操作数中存在不定态x，则逻辑运算的结果也是不定态

## 按位操作符

- 按位取反“~”、按位与“&”、按位或“|”、按位异或“^”、按位同或“^~”

按位与真值表

| & | 0 | 1 | x |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 1 | 0 | 1 | x |
| x | 0 | x | x |

例2.3-5:

```
module bit_tb;
reg[4:0]a;
reg[4:0]b;
initial
begin
a=5'b101; //运算的时候a自动变为5'b00101
b=5'b11101;
$display("%b",~a); //结果为5'b11010
$display("%b",~b); //结果为5'b00010
$display("%b",a&b); //结果为5'b00101
$display("%b",a|b); //结果为5'b11101
$display("%b",a^b); //结果为5'b11000
end
endmodule
```

## 归约操作符

与“&”、或“|”、异或“^”，以及相应的非操作“~&”、“~|”、“~^”、“^~”

例2.3-6:

```
module cut_tb;
reg[5:0]a;
initial
begin
a=6'b101011;
$display("%b",&a); //结果为1'b0
$display("%b",|a); //结果为1'b1
$display("%b",^a); //结果为1'b0
end
endmodule
```

## 移位运算符

左移位运算符“`<<`”、右移位运算符“`>>`”。

运算过程是将左边（右边）的操作数向左（右）移，所移动的位数由右边的操作数来决定，然后用0来填补移出的空位。

```
例2.3-7:  
module shift_tb;  
reg[5:0]a,b,c,d;  
reg[7:0]e;  
initial  
begin  
a=6'b101101;  
b=a<<2;  
c=a>>3;  
d=a<<7;  
e=a<<2;  
$display("%b",b); //结果为6'b110100  
$display("%b",c); //结果为6'b000101  
$display("%b",d); //结果为6'b000000  
$display("%b",e); //结果为8'b10110100  
end  
endmodule
```

## 条件运算符

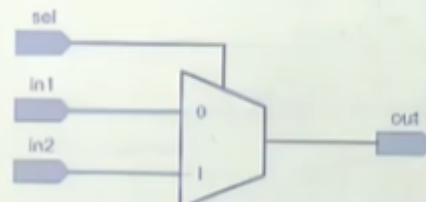
表达形式如下：

`<条件表达式>?<表达式1>:<表达式2>`

- 条件表达式的计算结果有真“1”、假“0”和未知态“x”三种，当条件表达式的结果为真时，执行表达式1，当条件表达式的结果为假时，执行表达式2。

例2.3-8：

```
module mux2(in1,in2,sel,out);  
input [3:0]in1,in2;  
input sel;  
output [3:0]out;  
reg [3:0]out;  
assign out=(!sel)?in1:in2;  
//sel为0时out等于in1,反之out等于in2  
endmodule
```



2选1数据选择器

## 连接和复制运算符 \*

连接运算符“( )”和复制运算符“{ }”

- 连接操作符

(信号1的某几位，信号2的某几位，...，信号n的某几位)

- 重复操作符{ }将一个表达式放入双重花括号中，复制因子放在第一层括号中。

例2.3-9：

```
module con_rep_tb;
reg [2:0]a;
reg [3:0]b;
reg [7:0]c;
reg [4:0]d;
reg [5:0]e;
initial
begin
a=3'b101;
b=4'b1110;
c=(a,b);           //连接操作
d=(a[2:1],b[2:0]);//连接操作
e={2(a)};          //复制操作符
$display("%b",c); //结果8'b01011110
$display("%b",d); //结果5'b10110
$display("%b",e); //结果6'b101101
end
endmodule
```