§5.5 VHDL语言程序基本结构

现代电子设计方法EDA技术

Electronic Design Automation

现代电路的设计方法：硬件设计+编程设计

掌握描述方式：硬件描述语言 （VHDL）

熟悉设计工具：集成化开发系统

（ Altera公司： QuartusII、MuxplusII

Xilinx公司： ISE

Lattice公司： ispDesignEXPERT ）

了解实现的载体：PLD （Programmable Logic Device）

学习方法：重视上机实践

硬件描述语言：描述数字系统的结构、行为、功能和接口

ABEL

AHDL

Verilog HDL

VHDL

} **IEEE标准**

Very high speed integrated Circuit Hardware Description Language (VHDL)

设计开始之前都应当有一个硬件框图。

标识符

合法的标识符：

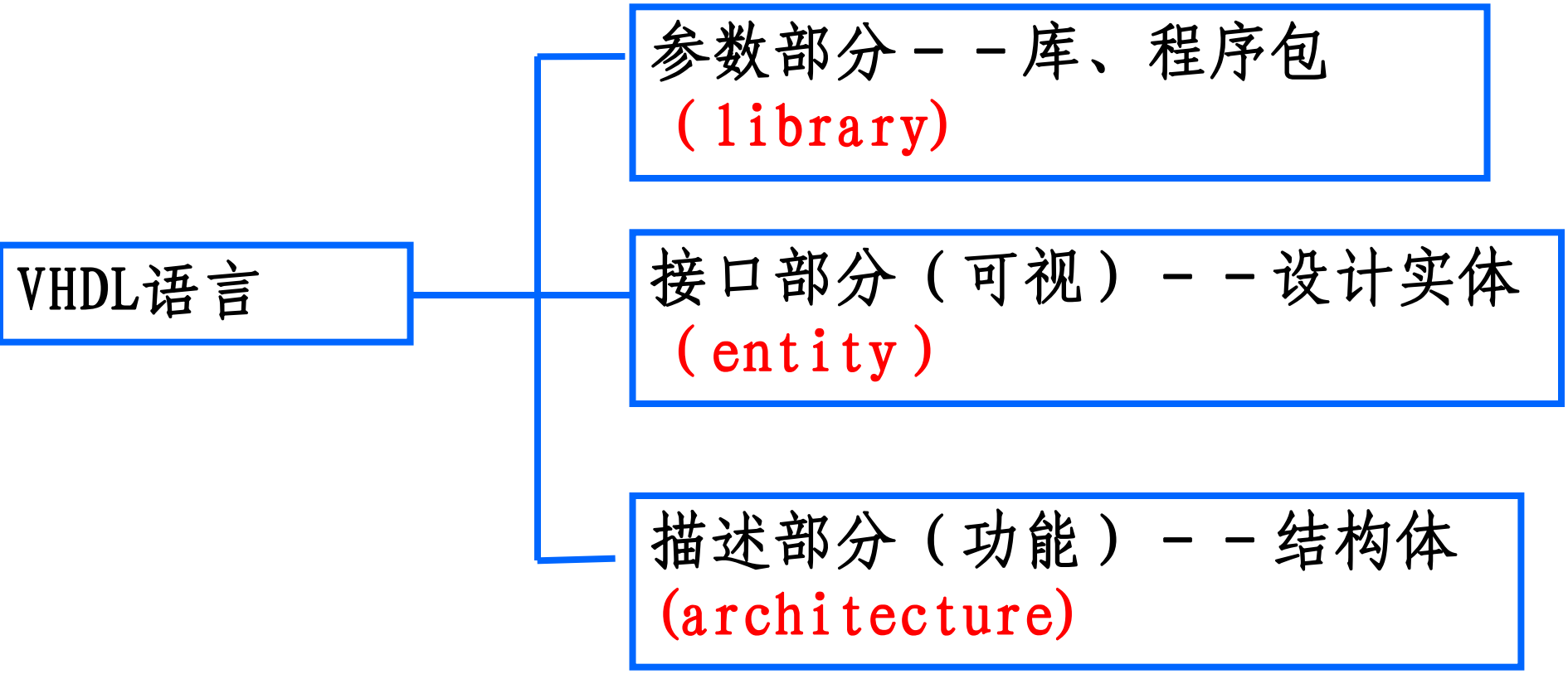
A，fft，and_4，　max2uc

非法的标识符：

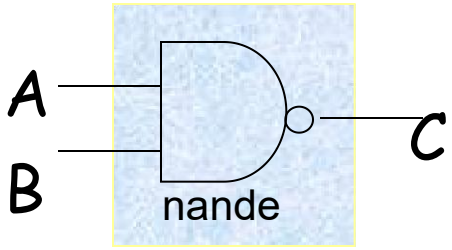
21A，_fft，and_ _4，　max#2uc，a-b，ab_

- 第一个字符必须是字母；
- 下画线不能开头与结尾，不允许连续2个下画线；
- 最长32个字符，不区分大小写(‘ ’内的字符、“ ”内的字符串除外)，不能和VHDL的保留字相同；
- 字符‘Z’、字符串“ZZZZ”的大写有特殊意义。
- 存盘文件名应与设计的实体名相同。
- 各完整语句均以“;”结尾，以“- -”开始的语句为注释语句，不参与编译。

5.5.1 VHDL模型的基本结构



例：与非门




```
LIBRARY  IEEE;
USE IEEE.STD_LOGIC_1164.all;
```

```
ENTITY nande IS
      port ( a, b: IN std_logic;
                   c: OUT std_logic );
END nande;
```

```
ARCHITECTURE art1 OF nande  IS
BEGIN
       c <= not (a and b);
END  art1;
```

一、库的调用、说明

库（Library）：存放公用数据类型，共5大类。

- 语法：

```
LIBRARY 库名;
USE 库名.程序包名.项目名（all）;
```

最常用IEEE库：

```
LIBRARY IEEE;    --库名
USE IEEE.STD_LOGIC_1164.ALL;  --定义标准逻辑数据类型及其逻辑运算函数
USE IEEE.STD_LOGIC_UNSIGNED.ALL;  --调用无符号数、标准逻辑与整数之间的算术、比较函数
USE IEEE.STD_LOGIC_ARITH.ALL;  --定义标准逻辑的算术运算函数
```

二、实体说明

实体：描述系统的输入、输出端口。

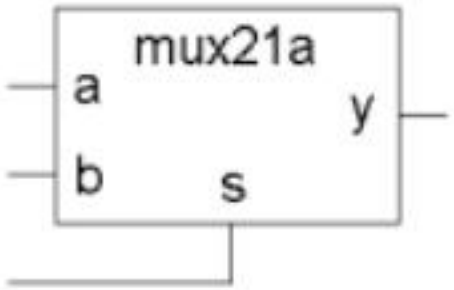
- 语法：

```
ENTITY      实体名      IS
        PORT      （端口表说明）;
END   实体名;
```

  - 端口声明：

```
PORT  (
      端口名称{, 端口名称}: 端口模式     数据类型;
      …
      端口名称{, 端口名称}: 端口模式     数据类型
            );
```

```
ENTITY mux21a IS

   PORT( a, b ： IN  BIT ；
                s ： IN  BIT;
                y ： OUT BIT  ) ;

END mux21a ；
```



mux21a 实体

分号在外

端口名

端口模式

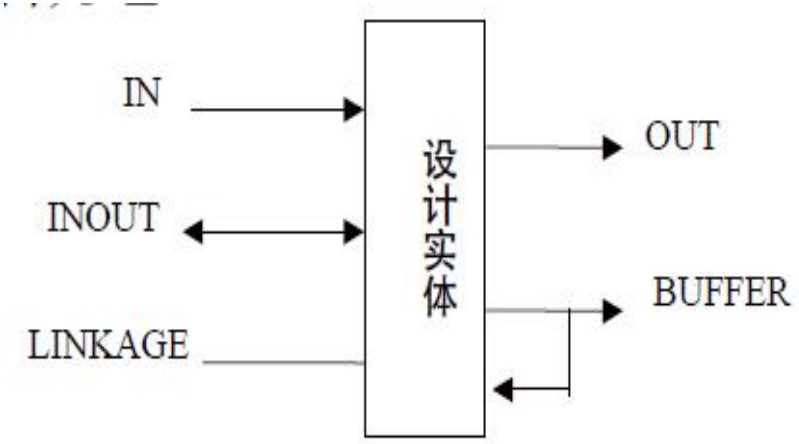
数据类型

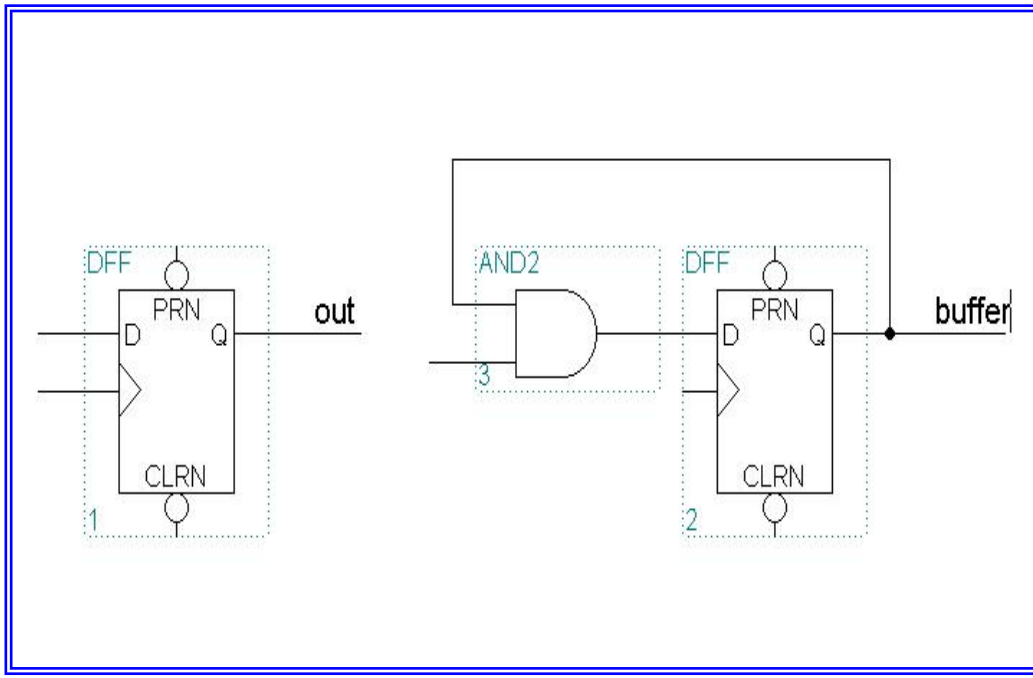
注意：

- 实体名必须与文件名相同，否则无法编译。
- 实体名不能用中文，也不能用数字、下划线开头。

- 端口模式:

方向定义	含义
IN	输入
OUT	输出
INOUT	双向
BUFFER	输出





- **数据类型：**

指端口上流动的数据的表达格式。应为预先定义好数据类型。用户可自己定义。**VHDL是强数据类型语言。**

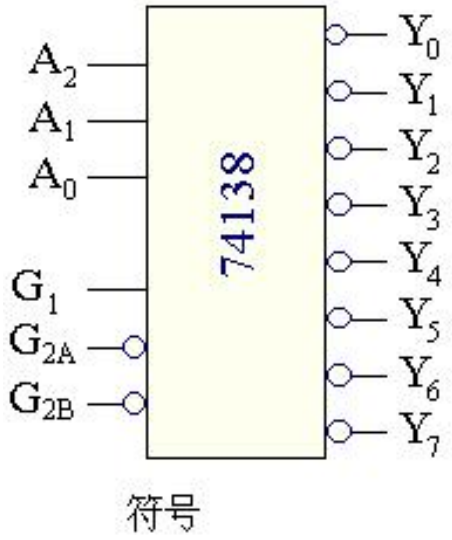
bit：位逻辑型（只能取‘0’或‘1’）。
bit_vector：位逻辑矢量型（“0011”）。

std_logic：标准逻辑型（取‘0’、‘1’、‘Z’等9值逻辑）
std_logic_vector：标准矢量型（“ZZZZ”）
integer：要对具体的整数作出范围限定，否则无法综合成硬件电路。不能使用逻辑操作符。

如：*signal s : integer range 15 downto 0;*

整数在表达式中不加引号，而逻辑数必须加引号，一位的加单引号，一位以上的加双引号。

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

ENTITY decode38 IS
      PORT( a2,a1,a0,g1,g2a,g2b:IN STD_LOGIC;
            y: OUT STD_LOGIC_VECTOR (7 DOWNTO 0));
END decode38;
```



符号

三、结构体

作用：定义系统（或模块）的行为，描述其逻辑功能。

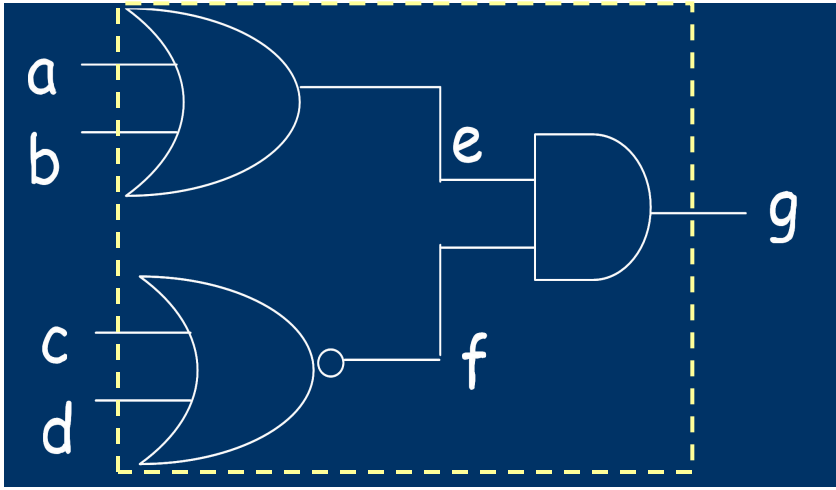
两部分组成：

- 结构体说明；
- 结构体描述；

结构体的语法：

```
ARCHITECTURE    结构体名   OF   实体名      IS
      [说明语句]   内部信号、常数、元件、数据类型等的说明;
BEGIN
      [并行处理语句];    （行为描述、数据流描述、结构化描述）
END   结构体名;
```

例:




```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY simp IS
   PORT(a, b, c, d : IN std_Logic;
                     g : OUT std_Logic);
END simp;

ARCHITECTURE act OF simp IS
     SIGNAL e，f : std_Logic;
BEGIN
     g <= e and f;
     e <= a or b;
     f <= not (c or d);
END act;
```

信号要有数据类型说明。不需要方向说明。

并行行为

5.5.2 VHDL的数据对象、操作符

一、数据对象

三种对象：

（1）常量（CONSTANT）：固定值（电源、地等）。

（2）变量（VARIABLE）：暂时数据的局部存储（算法描述）。

（3）信号（SIGNAL）：代表实际连线。

对象的说明格式为：

对象类型　标识符: 数据类型[: = 初值];

```
constant num :  integer := 6;
```

注意！常数定义的同时进行赋初值。

```
signal   s4：STD_LOGIC_VECTOR( 7 DOWNTO 0)
signal   s1：STD_LOGIC;
variable   b,c : INTEGER range 0 to 10;
```

constant data : bit_vector(3 downto 0):="1010"

三种数据对象的特点及说明场合:

- 信号: 全局量， 在architecture说明区中声明。
- 变量: 局部量。只能在process说明区中声明，只能用于process中。
- 常量: 全局量，可用于上面两种场合。

数据对象的赋值:

- Constant ：在程序中不能被赋值

- Variable ：“ ：＝ ”，赋值后立即变化为新值，无延迟。

- Signal：“ <＝ ”，先改变驱动值，但不立即更新赋值，有延迟。

赋值原则：相同位宽、相同数据类型

例：进程中信号与变量的使用

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

entity  ex  is
     port (…..);
end  ex;

architecture  arch_ex  of  ex  is
     signal  a, b : std_logic;
begin
          ……
     process (a, b)
          variable  c, d : std_logic;
     begin
          c := a and b;
          d := a or b;
             ……
     end  process;
end  arch_ex;
```

三、 VHDL操作符

四类：逻辑、算术、关系、重载操作符。

1. 逻辑操作符：（应用于 bit，std_logic 类型）

6种：and、 or、 nand、 nor、 xor、 not

要求：

（1) 操作数类型、宽度必须相同，可为

bit, bit_vector, std_logic, std_logic_vector。

（2) 无优先级，要用括号。

```
X <= ( a and b ) or ( not c and d );
```

2. 关系操作符: （=、/=用于任何数据类型，其他用于整数、枚举类型、逻辑矢量）

6种：=、 /=、 <、 <=、 >、 >=

用于相同类型的两个操作数；返回 **boolean** 值。

```
if( en= '0' ) then
        dout <= din ;
else
        dout <= 'Z';
End  if ;
```

与赋值语句相同

3. 算术操作符：（除&外，都用于integer类型）

十、 一、 *、/、&

并置操作符 &：连接多个操作数，形成一个新的逻辑矢量。

```
signal a : std_logic_vector (5 downto 0);
signal b : std_logic_vector (2 downto 0);
          a <= “100” & b;
```

```
signal a,b,c,d : std_logic ;
Signal co : std_logic_vector(2 downto 0) ;
signal q : std_logic_vector (3 downto 0);
         q <=  a&b&c&d ;
         co <= q(2 downto 0);
```

类　型	操作符	功　能	操作数数据类型
算术操作符	+	加	整数
	—	减	整数
	&	并置	一维数组
	*	乘	整数和实数(包括浮点数)
	/	除	整数和实数(包括浮点数)
	MOD	取模	整数
	REM	取余	整数
	SLL	逻辑左移	BIT 或布尔型一维数组
	SRL	逻辑右移	BIT 或布尔型一维数组
	SLA	算术左移	BIT 或布尔型一维数组
	SRA	算术右移	BIT 或布尔型一维数组
	ROL	逻辑循环左移	BIT 或布尔型一维数组
	ROR	逻辑循环右移	BIT 或布尔型一维数组
	**	乘方	整数
	ABS	取绝对值	整数

关系操作符	=	等于	任何数据类型
	/=	不等于	任何数据类型
	<	小于	枚举与整数类型，及对应的一维数组
	>	大于	枚举与整数类型，及对应的一维数组
	<=	小于等于	枚举与整数类型，及对应的一维数组
	>=	大于等于	枚举与整数类型，及对应的一维数组
逻辑操作符	AND	与	BIT，BOOLEAN，STD_LOGIC
	OR	或	BIT，BOOLEAN，STD_LOGIC
	NAND	与非	BIT，BOOLEAN，STD_LOGIC
	NOR	或非	BIT，BOOLEAN，STD_LOGIC
	XOR	异或	BIT，BOOLEAN，STD_LOGIC
	XNOR	异或非	BIT，BOOLEAN，STD_LOGIC
	NOT	非	BIT，BOOLEAN，STD_LOGIC
符号操作符	+	正	整数
	—	负	整数

4、重载操作符

如何实现不同数据类型之间的运算？

重载操作符：
对已存在的操作符重新定义，使其能进行不同类型操作数之间的运算。通过调用一系列重载函数实现。

重载操作符的定义在 **IEEE** 库的程序包中：

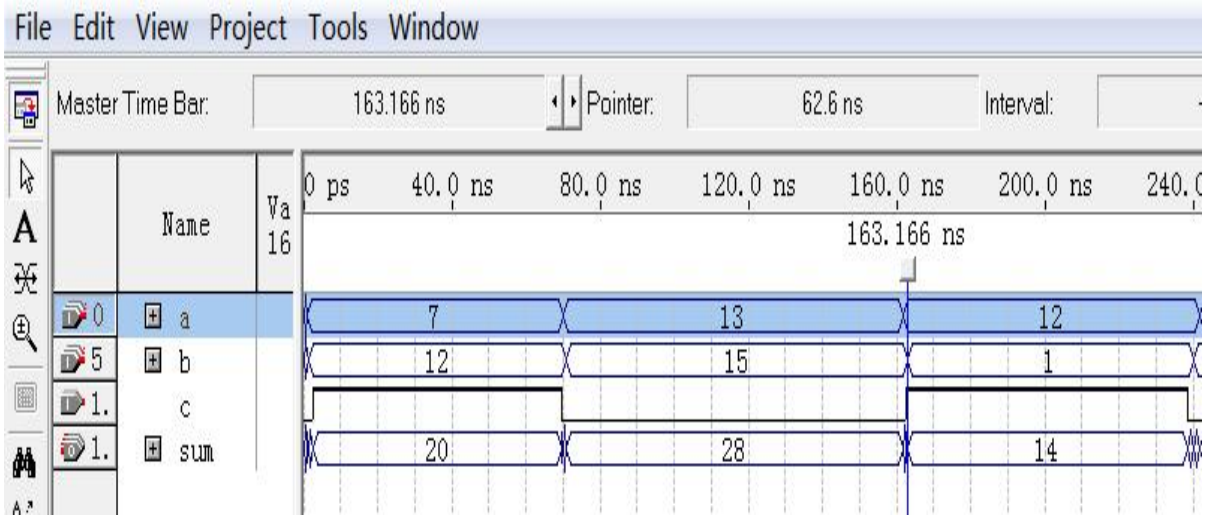
std_logic_unsigned、
std_logic_arith、
std_logic_signed

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_unsigned.all;

ENTITY overload IS
   PORT ( a : IN STD_LOGIC_VECTOR (3 downto 0);
          b : IN STD_LOGIC_VECTOR (3 downto 0);
          c : IN integer range 0 to 15;
          sum1 : OUT STD_LOGIC_VECTOR (4 downto 0);
          sum2 : OUT STD_LOGIC_VECTOR (4 downto 0));
END overload;

ARCHITECTURE example OF overload IS
BEGIN
  sum1 <= ('0'&a)+b ;
  sum2 <= ('0'&a)+c ;
END example;
```

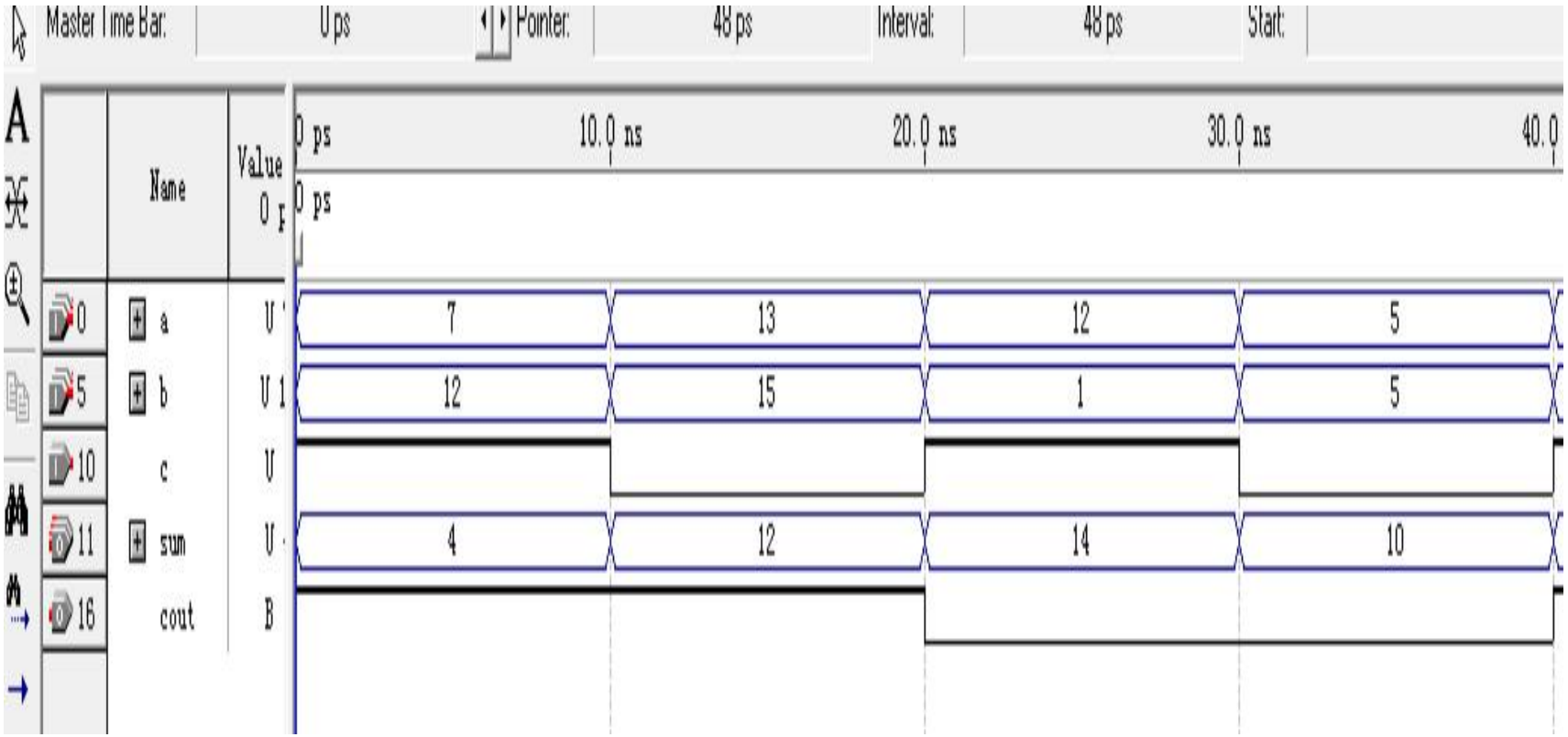
定义了基本算术、比较运算的重载运算符

实现 四位二进制全加器（和为五位）。

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity add is
  port (
        a,b : in std_logic_vector(3 downto 0);
        c : in std_logic;
        sum : out std_logic_vector (4 downto 0)
        );
end add;

architecture arc of add is
  begin
      sum <= ('0'&a)+b+c;
end arc;
```



```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity add is
  port (
        a,b : in std_logic_vector(3 downto 0);
        c : in std_logic;
        sum : out std_logic_vector (3 downto 0);
        cout: out std_logic
        );
end add;

architecture arc of add is
    signal sumt : std_logic_vector(4 downto 0);
  begin
      sumt <= ('0'&a)+b+c;
      cout <= sumt(4);
      sum <= sumt(3 downto 0);
end arc;
```


Master Time Bar:
0 ps
Pointer:
48 ps
Interval:
48 ps
Start:
Name
Value
0 ps
10.0 ns
20.0 ns
30.0 ns
40.0
a
b
c
sum
cout
7
13
12
5
12
15
1
5
4
12
14
10