第3章 FPGA的硬件描述语言VHDL

3.1 VHDL 程序的特点

FPGA 的硬件描述语言 VHDL(Very High Speed Integrated Circuit Hardware Description Language,超高速集成电路硬件描述语言)符合美国电气和电子工程师协会标准(IEEE 1076—2008),利用一种和数字电路基本知识结合较密切的语言来描述数字电路和设计数字电路系统。利用 VHDL 进行分块单元电路设计和整个系统设计,并结合一些先进的 EDA 工具软件(如 Quartus Ⅱ),通过计算机下载到硬件芯片上,实现电路功能,可以极大地缩短产品的设计周期,加快产品进入市场的步伐,在当今高速发展的信息时代,可以更好地把握商机。硬件芯片 FPGA 如图 3.1 所示。

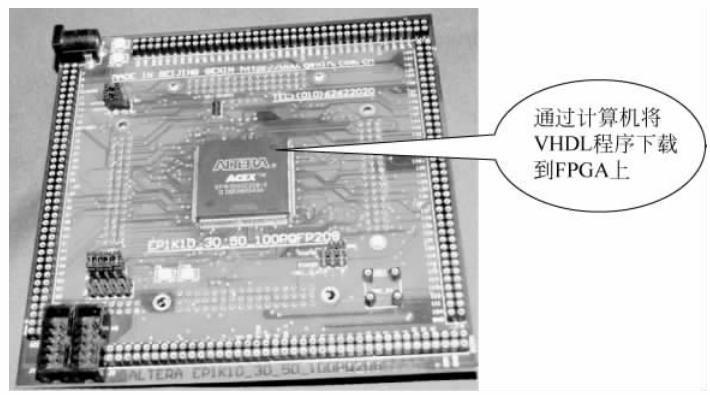

图 3.1 硬件芯片 FPGA

为适应实际数字电路的工作方式,VHDL 以并行和顺序的多种语句方式来描述在同一时刻中所有可能发生的事件。因此 VHDL 程序执行方式与其他语言不同,它不是按顺序一条一条执行每一条语句,而是有并行执行的语句同时也有按顺序执行的语句;要求数字电路设计人员摆脱一维的思维模式,以多维并发的思路完成 VHDL 的程序设计。VHDL 程序的特点如图 3.2 所示,VHDL 程序由一组并行语句构成,并行语句里有顺序语句。

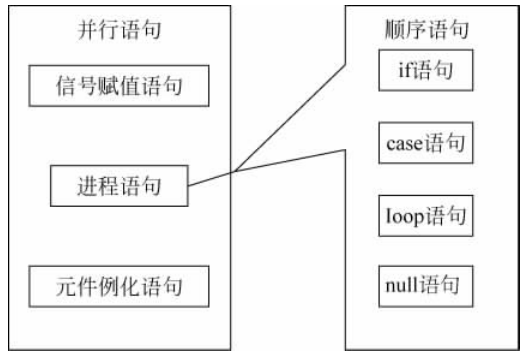

图 3.2　VHDL 程序的特点

为完成 VHDL 的程序设计，实现数字电路的功能，要求工程设计人员要懂得每一种 VHDL 语句格式、内容和各种 VHDL 语句执行的顺序，以及 VHDL 语句中的各种变量、数据类型和表达式。一个成功的 VHDL 程序设计，不能只完成数字电路功能，还要求程序设计得简捷，以便器件工作速度快、占用资源少。

3.2　VHDL 程序的基本结构

先看一个关于 D 触发器设计的完整程序。

【程序 3.1】

```
library ieee;
use ieee.std_logic_1164.all;                    -- 库说明
entity dff1 is
    port(clk,d:in std_logic;
         q:out std_logic);
end dff1;                                       -- 实体说明
architecture rtl of dff1 is
begin
    process(clk)
    begin
        if(clk'event and clk = '1')then
          q <= d;
        end if;
    end process;
end rtl;                                        -- 结构体说明
```

从程序 3.1 的描述层次上可以看出，一个完整的 VHDL 程序通常包括库说明、实体说明、结构体说明 3 个部分，下面具体介绍这 3 个部分的语法。

3.2.1　库说明

在程序 3.1 中，库说明语句为

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

用到了 ieee 库以及 ieee 库中的 std_logic_1164 程序包的全部资源。

库说明语句一般语法如下：

```
library 库名;
use 库名.程序包名.项目名;
```

库是用 VHDL 编写的源程序及其通过编译的数据的集合，库由各种程序包组成，程序包提供了各种数据类型以及各种类型转换函数及运算等，以供给设计者使用。VHDL 提供了 IEEE 库和其他的库。

IEEE 库是按 IEEE 组织制定的工业标准进行编写的标准资源库，库的内容很丰富，是最常用的资源库，其中常用的程序包有：

(1) std_logic_1164 程序包：常用数据类型(其中有 std_logic 及 std_logic_vector 数据类型)和各种数据类型转换函数及其算术、逻辑运算。

(2) std_logic_arith 程序包：它在 std_logic_1164 程序包的基础上定义了无符号数 unsigned、有符号数 signed 数据类型并为其定义了相应的算术运算、比较，无符号数 unsigned、有符号数 signed 及整数 integer 之间转换函数。

(3) std_logic_unsigned 和 std_logic_signed 程序包：这些程序包定义了可用于 integer 数据类型和 std_logic 及 std_logic_vector 数据类型混合运算的运算符，并定义了由无符号数 unsigned、有符号数 signed 与 std_logic_vector 数据类型之间的转换函数，std_logic_vector 数据类型与 integer 数据类型之间的转换函数。其中，std_logic_signed 中定义的运算符是有符号数运算符。

用于一般的 FPGA/CPLD 的开发，IEEE 库中的 4 个程序包(std_logic_1164、std_logic_arith、std_logic_signed 和 std_logic_unsigned)已足够使用。

库说明语句作用范围是从一个实体说明开始到它所属的结构体为止。

3.2.2 实体说明

实体的电路意义相当于器件，在电路原理图上相当于元器件符号(或元器件引脚图或电路图)，程序 3.1 实体的电路图如图 3.3 所示。实体说明了器件的输入、输出引脚。它是一个完整的、独立的语言结构，也称为模块，实体给出了设计模块和外部的接口。

实体说明语句语法如下：

```
entity  实体名  is
    port(端口名称 1: 端口方式 1 端口类型 1;
         端口名称 2: 端口方式 2 端口类型 2;…);
end  实体名;
```

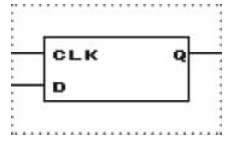

图 3.3　程序 3.1 实体的电路图

在程序 3.1 中实体说明语句为

```
entity dff1 is
    port(clk,d:in std_logic;
             q:out std_logic);
end dff1;
```

实体名为 DFF1。在程序设计过程中，要求实体名与存储的文件名一致。实体名为 DFF1，则存储的文件名为 DFF1.VHD。其端口名称是这个设计模块与外部接口的桥梁，共有 3 个端口，其名称分别为 D、CLK、Q。端口方式 D 和 CLK 是输入，Q 是输出。端口类型 D、CLK、Q 都是 std_logic 数据类型。

特别提示：触发器电路符号 DFF1，输入引脚 CLK，D；输出引脚 Q。实体名为 DFF1，则存储的文件名为 DFF1.VHD。

端口方式有 5 种，如表 3.1 所示。

表 3.1 端口方式

端口方式	类 型	说 明
in	输入型	信号从该端口进入实体
out	输出型	信号从实体内部经该端口输出
inout	输入输出型	信号从该端口既可输入也可输出
buffer	缓冲型	与 out 类似但在结构体内部可作反馈
linkage		无指定方向，可与任何方向的信号连接

端口类型是预先定义好的数据类型，这部分内容在 3.3 节有详细介绍。

3.2.3 结构体说明

结构体是整个 VHDL 语言中至关重要的一个组成部分，这个部分会给出模块的具体实现，指定输入与输出之间的行为。结构体说明语句语法如下：

```
architecture  结构体名称  of  实体名 is
    结构体说明部分;
begin
    结构体并行语句部分;
end  结构体名称;
```

结构体名称：本结构体的命名。通常根据结构体描述方式自己命名。

实体名：实体的命名，也就是所存储文件的命名。

结构体说明部分：对结构体内部所使用的信号、常数、数据类型和函数进行定义。

结构体并行语句部分：具体地确定各个输入、输出之间的关系，描述了结构体的行为，是一组并行处理语句，也就是说结构体中的语句的执行是不以书写语句顺序为准的，结构体内是一组并行语句，并行执行。

特别提示：结构体内是一组并行语句。

在程序 3.1 中结构体名 rtl，实体名 dff1，结构体并行语句部分从 process 开始，到 end process 为止。结构体语句部分只有一个进程语句，进程语句描述了当时钟 clk 上升沿到时，将输入 d 信号赋给输出 q，否则 q 不变。程序 3.1 产生了如图 3.3 所示 D 触发器电路（或元器件）符号。

3.3 VHDL 的数据

VHDL 和其他软件编程语言一样，也有严格的标识符、数据对象、数据类型定义，准确、熟练掌握基本的数据定义，对初学者是非常必要的。

3.3.1 基本标志符

基本标志符有 A～Z、a～z、0～9 以及下画线“_”。VHDL 不区分大小写。标志符必须以字母开头，不能以下画线为结尾，不能出现连续的两个或多个下画线。以下是一些有效的基本标志符：DRIVE_BUS、addr_bus、decoder_38、RAM18。

3.3.2 数据对象

数据对象也可认为是数值的载体，共有 3 种形式的数据对象：常量(constant)、变量(variable)、信号(signal)。

1. 常量

常量是设计者给某一常数名赋予固定值的量，一旦赋值就不会发生变化。一般格式为

```
constant 常数名: 数据类型 := 表达式;
```

对常量赋值用“:=”常量声明的例子如下：

```
constant width:integer := 8;                    -- 常数名 width,数据类型 integer 整数,赋值 8
constant VCC:real := 3.3;                       -- 常数名 VCC,数据类型 real 实数,赋值 3.3
```

2. 变量

变量是可以改变值的量，可以在进程和子程序中说明，可以是任意数据类型。变量的赋值是立即生效的。一般格式为

```
variable 变量名: 数据类型 := 初始值或表达式;
```

对变量赋值用“:=”表示。变量声明的示例如下：

```
variable temp:std_iogic := '0';     -- 变量名 temp,数据类型 std_iogic 标准逻辑位赋初始值 0
variable a,b:bit_vector(0 to 7);    -- 变量名 a,b,数据类型 bit_vector(0 to 7) 是位矢量
b := "10101010";                    -- 位矢量赋值
a(3 to 6) := ('1','1','0','1');     -- 段赋值
a(0 to 5) := b(2 to 7);
a(7) := '0';                        -- 位赋值
```

3. 信号

信号是电子电路内部硬件连接的抽象，可以将结构体中分离的并行语句连接起来，并且能通过端口与其他的模块连接。可以随着时间改变值，不像变量赋值立即生效，允许产生延时。信号通常在实体、结构体和程序包中说明，但不能在进程中说明，只能在进程中使用。对信号赋值使用<= 表示，允许产生延时，这和实际元件的传输延时特性吻合。

一般格式为：

```
signal 信号名: 数据类型 := 初始值;
```

信号声明的示例如下：

signal a：bit :='0'；--信号名 a，数据类型是位，赋初值为 0。赋值符 :=给信号赋初值时会立即生效，而不产生延迟。但信号一般用代入符 <= 赋值，会产生一定延迟才生效。信号可以赋初值也可以不赋初值，没有赋初值，那么取默认值，即指定数据类型的最左值或最小值。

```
signal INIT:bit_vector(7 downto 0);          -- 定义信号 INIT,是位矢量
signal c:integer range 0 to 15;              -- 定义信号 c,数据类型是整数,整数范围 0 到 15
signal y,x: real;                            -- 定义信号 y,x,数据类型是实数
        y<= x;                               -- 经过 δ 延时后将 x 值赋给 y
```

特别提示：变量只能在进程语句中说明和使用；信号、常量可以在结构体和进程语句中使用。变量、常量用 := 赋值，无延时；信号用 <= 赋值，有延时。

3.3.3 数据类型

1. 标准数据类型

标准数据类型共有 10 种，见表 3.2。

表 3.2 标准数据类型

数据类型	含　义	备　注	例　子
整数	$-(2^{31}-1)\sim+(2^{31}-1)$	integer	+136，-457，3e4(3×10^4)
实数	$-10^{38}\sim+10^{38}$	real 一定有小数点	-1.0，+2.5e23(+2.5×10^{23})
位	逻辑 0 或 1	bit 单引号括起来	'1'，'0'
位矢量	位矢量	bit_vector 双引号括起来的一组数	"00101"
布尔量	逻辑假或真	boolean 只有真(true)和假(false)	
字符	ASCII 字符	character 用单引号括起来	'a'，'b'，'1'
时间	整数和时间单位	time fs，ps，ns，us，ms，sec，min，hr	20μs、32ns
错误等级	VHDL 程序在编译、仿真、综合过程的工作状态	severitylevel note，warning，error，failure	note，warning 可以忽略，error，failure 不可以忽略
自然数，正整数	整数的子集	natural，positive	
字符串	字符矢量	string 双引号括起来的字符序列	"START"

特别提示：常用的数据类型包括整数、实数、位、位矢量。整数可以用十进制、二进制、八进制、十六进制表示，例如，十进制 125，用二进制写为 2#11111111#，八进制写为 8#377#，十六进制写为 16#FF#。布尔量用于关系运算，如 a>b 成立，结果为真 true。

2. 用户自定义的数据类型

VHDL 允许用户自定义数据类型,一般书写格式为:

```
type 数据类型名 is 数据类型定义;
```

VHDL 常用的用户自定义类型包括枚举类型、整数类型、数组类型、子类型等。

1) 枚举类型

把数据类型中的各个元素都列举出来,方便、直观,提高了程序可阅读性。书写格式为:

```
type 数据类型名称 is(元素 1,元素 2,…);
```

其中,数据类型名称和元素都是标志符。例如:

```
type color is (blue,green,yellow,red);
```

数据类型名称是 color,(元素 1,元素 2,…)是(blue,green,yellow,red)。枚举类型中所列举的元素在程序编译过程通常是自动编码,编码顺序是默认的,左边第一个元素编码为 0,以后的依次加 1。编码过程中自动将每一个元素转变成位矢量,位矢量的长度将由所列举元素的个数决定。如上例 4 个元素,位矢量的长度为 2,编码默认值为 blue="00"; green="01"; yellow="10"; red="11"。在信号定义时就可以使用这种数据类型,例如:

```
type color is (blue,green,yellow,red);
signal p: color;                              -- 信号 p 是 color 数据类型
```

2) 整数类型、实数类型

自定义的整数类型、实数类型是标准数据类型的整数、实数的子类型,是根据特殊需要自定义的数据类型,以便编译过程降低逻辑电路的复杂性和提高芯片资源的利用率。书写格式为:

```
type 数据类型名称 is integer range 整数范围;
type 数据类型名称 is real range 实数范围;
```

例如:

```
type percent is integer range - 100 to 100;
```

percent 是数据类型名称,它的数据类型是 integer 整数,range 整数范围是-100～100。

```
type current is real range - 1.5 to 3.0;
```

current 是数据类型名称,它的数据类型是 real 实数,range 实数范围是-1.5～3.0。

3) 数组(array)类型

数组是将相同类型的数据即数组元素集合在一起所形成的一个新的数据类型。数组类型分限定数组和非限定数组两种,书写格式为:

```
type 数组类型名 is array 范围 of 数组元素的数据类型;
type 数组类型名 is array ( range <>) of 数组元素的数据类型;
```

其中范围是用整数指明数组的上下界,是一个限定数组,例如:

```
type stb is array (7 downto 0) of bit;
variable addend: stb;                          -- 变量 addend 定义为 stb 数组
```

stb 是数组类型名。(7 downto 0)是数组的上下界,数组有 8 个元素,数组的下标排序是 7、6、5、4、3、2、1、0,各元素的排序是 stb(7)、stb(6)、stb(5)、stb(4)、stb(3)、stb(2)、stb(1)、stb(0),每一个元素的数据类型是 bit。范围由"range <>"指定,这是一个没有范围限制的数组,是一个非限定数组。在这种情况下,具体范围由信号说明语句来确定。例如:

```
type bit_vector is array(integer range <>)of bit;
```

bit_vector 是一个非限定数组,每一个元素的数据类型是 bit。

```
variable my_vector:bit_vector(5 downt to -5);
```

变量 my_vector 定义为 bit_vector 数组,数组的下标排序是 5、4、3、2、1、0、−1、−2、−3、−4、−5。

4) 子类型

子类型说明语句就是对已经存在的基本数据类型作一些范围限制,便形成了一种新的数据类型。子类型 subtype 的语句格式如下所示:

```
subtype 子类型名 is 基本数据类型 range 约束范围
```

用子类型说明语句的好处在于编译过程中可以根据子类型的约束范围,有效地推知参与的寄存器的最合适的数目,节省芯片资源。例如:

```
Type nat is integer range 0 to 999;          -- 自定义整数类型 nat 是整数,范围是 0-999
subtype a_nat is nat ragne 0 to 255;         -- 子类型 a_nat 是 nat 类型,范围是 0-255
```

3. IEEE 标准数据类型 std-logic 和 std-logic-vector

在 IEEE 库的程序包 std_logic_1164 中定义了两个非常重要的数据类型,即标准逻辑位 std-logic 数据类型和标准逻辑矢量 std-logic-vector 数据类型。

std-logic 定义了 9 种不同的值,增加了不定状态'X'、高阻状态'Z'。不定状态方便了系统仿真,高阻状态方便了双向总线的描述。

'U'——初始值
'X'——不定,未知
'0'——0
'1'——1
'Z'——高阻
'W'——弱信号不定,未知
'L'——弱信号 0
'H'——弱信号 1
'—'——不可能情况

3.4 VHDL 的表达式

在 VHDL 中,表达式是通过不同的操作符连接多个操作数完成算术或逻辑计算的式子。VHDL 的操作符包括逻辑运算符、关系运算符、算术运算符、并置运算符。VHDL

的数据对象(即操作数)有不同类型,逻辑类型的变量要用逻辑操作符、整数、实数类型的变量要用算术操作符。对于运算操作符和变量类型不匹配的情况,EDA 工具在编译时不予通过。

3.4.1 逻辑运算符

在 VHDL 中,逻辑运算符有 6 种,分别为:

- NOT——取反;
- AND——与;
- OR——或;
- NAND——与非;
- NOR——或非;
- XOR——异或。

逻辑运算符适用的变量为 std-logic,bit,std-logic-vector 这 3 种数据类型,进行逻辑运算时,逻辑运算符的左边、右边以及代入的信号的数据类型必须相同。对位矢量进行逻辑运算时是按位操作,其结果还是位矢量。

在一个 VHDL 语句中存在两个或两个以上逻辑表达式时,左右没有优先级差别。一个逻辑式中,先做括号里的运算,再做括号外的运算。

逻辑运算符规范书写方法如下:

```
(1) a<=b AND c AND d AND e;          — 用 VHDL 程序规范书写的语句
    a=b×c×d×e                        — 用等效的逻辑代数书写的逻辑方程

(2) a<=b OR c OR d OR e              — 用 VHDL 程序规范书写的语句
    a =b + c + d + e                 — 用等效的逻辑代数书写的逻辑方程

(3) a<=(b AND c )OR (d AND e );      — 用 VHDL 程序规范书写的语句
    a=(b×c) + (d×e)                  — 用等效的逻辑代数书写的逻辑方程
```

3.4.2 算术运算符

VHDL 语言有 10 种算数运算符,仅有 4 种可以被 EDA 工具编译为逻辑电路。VHDL 算数运算符的列表如下:

+——加运算
−——减运算
*——乘运算
/——除运算
MOD——求模运算
REM——取余运算
+——正
−——负

** ——指数运算

ABS——取绝对值

算术运算符的使用规则如下：

(1) 正负操作的操作数可以是整数、实数、物理量。

(2) 加减运算符的使用范围可以是整数、实数，而且对于加、减运算的两个操作数必须类型相同。

(3) 乘除法的操作数可以同为整数、实数，物理量乘以或除以整数的结果仍为物理量，物理量除以相同的物理量后，商为整数或实数。

(4) 求模、取余运算的操作数必须是同一整数类型的数据。

(5) 加、减、乘、除能编译为逻辑电路，其余运算编译成逻辑电路很困难，有时是完全不可能的。

3.4.3 关系运算符

两个对象在比较运算时，将两个操作数比较的结果表示出来则需要使用关系运算符。这些关系运算符运算的结果为布尔数据类型(boolean)，即为真(true)或为假(false)。

VHDL的关系运算符如下：

=——等于；

/=——不等于；

<——小于；

<=——小于等于；

>——大于；

=>—— 大于等于。

关系运算符，在VHDL程序设计中有如下规则：

(1) 两个对象进行比较时，数据类型一定要相同。

(2) =(等于)和/=(不等于)适用于所有数据类型的对象之间的比较。

(3) 大于、小于、大于等于、小于等于适用于整数、实数、位、位矢量及数组类型的比较。

(4) <=符号有两种含义：代入符和小于等于符，要根据上下文判断。

(5) 两个位矢量类型的对象比较时，自左至右，按位比较。

位矢量std-logic-vector在程序包std_logic_unsigned中，对std-logic-vector的关系运算作为专门定义，在位矢量比较之前，必须说明调用该包集合。这时位矢量还可以和整数进行关系运算。

3.4.4 并置运算符

在VHDL程序设计中，并置运算符号“&”用于位的连接。并置运算符的使用规则如下：

(1) 并置运算符可用于位的连接,形成位矢量。

(2) 并置运算符可用于两个位矢量的连接,构成更大的位矢量。

例如:

```
signal a, b, c : bit;
signal x, y : bit_vector(2 downto 0);
z <= a&b&c ;                                    -- z 是 3 位的位矢量
w <= x&y ;                                      -- w 是 6 位的位矢量
```

3.4.5 操作符的运算优先级

在 VHDL 程序设计中,逻辑运算、关系运算、算术运算、并置运算优先级是各不相同的,操作符的运算优先级如表 3.3 所示。操作符优先级为由高到低。not 优先级最高,xor 异或最低。

表 3.3 VHDL 中的操作符运算优先级

运算操作符类型	操 作 符	含 义
乘除运算	not	取反
	abs	取绝对值
	**	取幂
	*	乘
	/	除
	mod	取模
	rem	取余
一元正负运算	+	正
	−	负
加、减、合并运算	+	加
	−	减
	&	合并
关系运算	=	相等
	/=	不等
	<	小于
	<=	小于等于
	>	大于
	>=	大于等于
逻辑运算	and	逻辑与
	or	逻辑或
	nand	逻辑与非
	nor	逻辑或非
	xor	逻辑异或

特别提示:在 VHDL 程序中,有并行语句和顺序语句,写在不同程序行的硬件描述语句有的同时并行工作,也有的按书写顺序执行。操作符的运算优先顺序仅在同一行的情况下有顺序、有优先,不同行的操作符的运算有可能是同时的。