CHAPTER

第3章 EDA设计工具

由于超大规模集成电路技术的发展和计算机的普遍应用，使电子电路的设计工作摆脱了“刀耕火种”的时代。利用可编程的集成电路器件CPLD/FPGA，再通过电子电路设计自动化软件EDA，人们不用去做器件和电路的搭建就可以实现自己的设计，并且可以通过EDA软件实现仿真检验。EDA软件有几个，这里以Altera公司的Quartus II 6.0为例，说明如何使用EDA进行设计。

3.1 建立工程项目

3.1.1 启动Quartus II 6.0

在系统桌面上打开Quartus II 6.0的快捷方式图标，也可以选择【开始】→【所有程序】→Altera→Quartus II 6.0→Quartus II 6.0。Quartus II 6.0启动成功后出现如图3-1所示的界面。

图3-1 Quartus II启动窗口

Quartus II 6.0 界面主要由菜单栏、工具栏 、工作区和多种辅助视窗构成。

3.1.2 建立项目

在使用 Quartus II 6.0 进行电路设计之前必须先建立项目，项目是所有设计工作的统一管理文件。建立项目的方法是选择菜单栏中的 File→New Project Wizard，系统弹出如图 3-2 所示对话框。

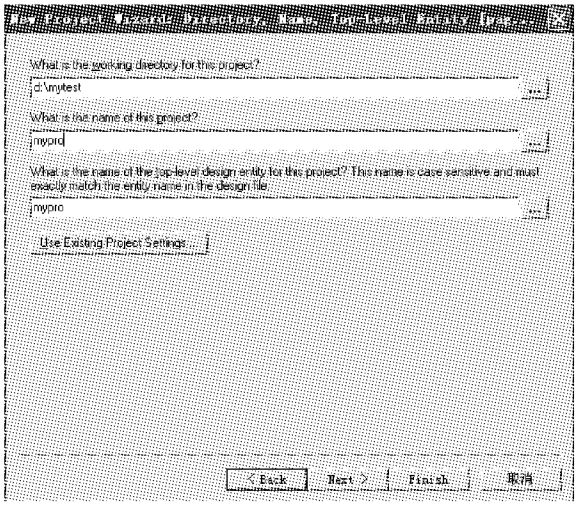

图 3-2 新建项目对话框

在第一栏中输入项目所在的文件夹路径(在此输入 d:\mytest)，第二栏中输入所建项目的名称(在此输入 mypro)，同时系统自动在第三栏中输入项目顶层文件的名称 mypro。

以上工作完成后单击 next 按钮，如果输入的文件夹并不存在，系统会提示是否建立，单击 Yes 按钮之后，即可建立相应文件夹和设定名称的项目。

随后系统弹出如图 3-3 所示的添加已经设计好的文件对话框。如果以前已经进行过

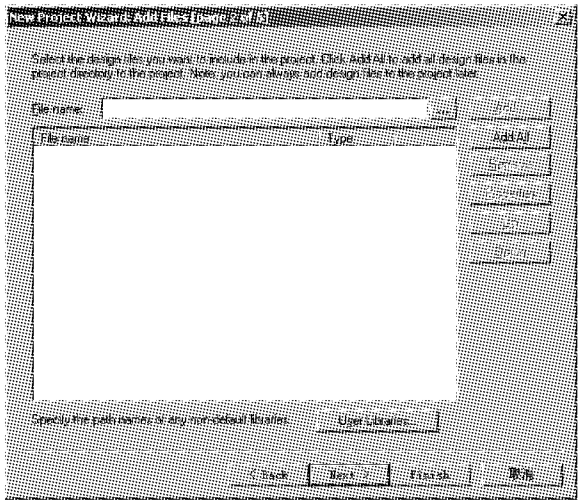

图 3-3 添加项目文件对话框

相关的设计，那么在此就可以将设计的文件添加到这个项目当中。

添加文件的工作可以在设计过程中随时进行，在此直接单击 Next 按钮即可。

随后系统将弹出如图 3-4 所示的选择目标器件及参数对话框，目标器件的参数包括器件封装型号、引脚数和速度级别等。此处按照图进行选择，然后单击 Next 按钮。

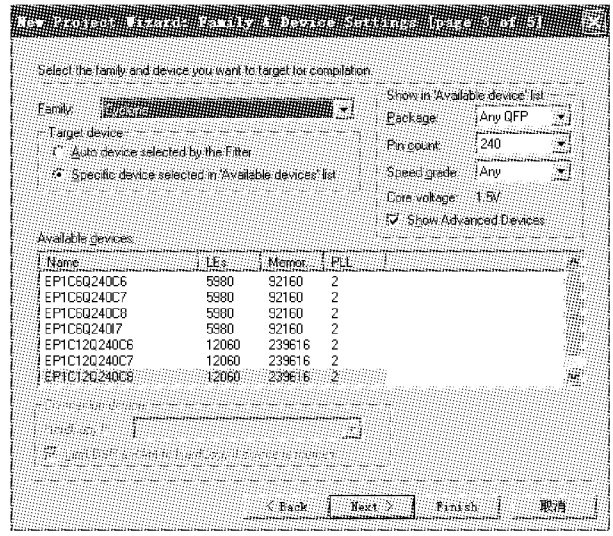

图 3-4　选择目标器件及参数对话框

单击 Next 按钮之后，系统弹出如图 3-5 所示窗口，要求用户选择 EDA 工具。可以不选，直接单击 Next 按钮。

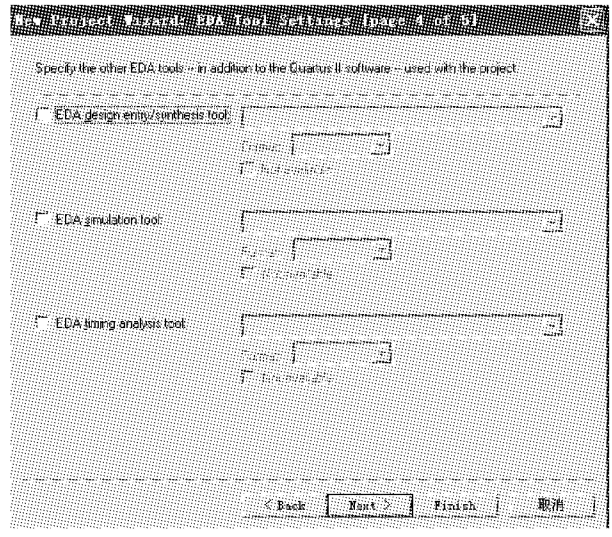

图 3-5　EDA 工具设置对话框

单击 Next 按钮之后，将弹出如图 3-6 所示的项目设置信息总结信息窗口，该窗口对之前所作的设置进行了汇总。单击 Finish 按钮，至此就完成了一个项目的建立工作。

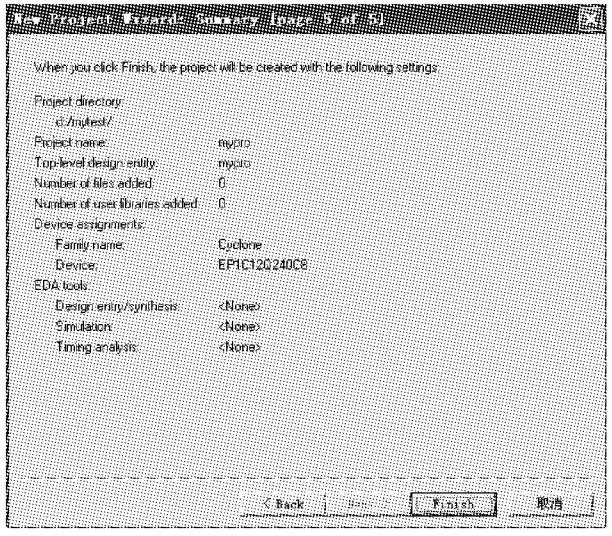

图 3-6 项目设置信息

3.2 设计文件

项目文件是一个工程管理文件，进行电子电路的设计，需要在工程项目文件的管理下建立具体的工作文件。Quartus II 根据需要为用户设立多种文件，并用菜单和对话框的方式，让用户能够十分方便地建立和使用它们。

3.2.1 原理图设计

原理图是电子电路设计最基本的一种形式，使用原理图进行电路设计直观、容易理解。建立原理图文件，可以从菜单栏中选择 File→New，将弹出如图 3-7 所示的新建文件对话框 New。

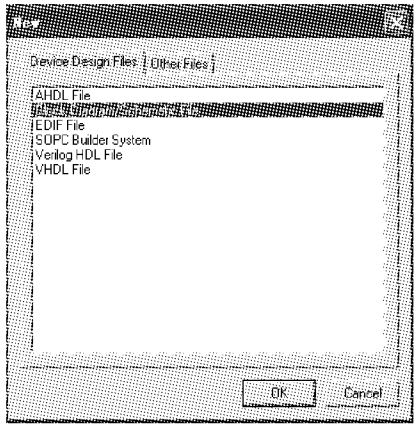

图 3-7 建立原理图文件对话框

New 对话框由两个选项卡组成。Device Design Files 选项卡中有原理图设计文件 Block Diagram/Schematic File，选择它，然后单击 OK 按钮，就可以进入原理图设计方式。

原理图设计方式的界面如图 3-8 所示，窗口中间空白的区域是工作区，在工作区中可以直接放入器件，并能够用导线进行器件的连接，构造用户所需要的电路。

选择原理图设计文件时，系统会给出文件的默认名称 Block1.bdf，如果想自己设定文件名称，可以在存储文件时更改。原理图设计窗口的左侧是系统自动打开的绘图工具栏。单击工

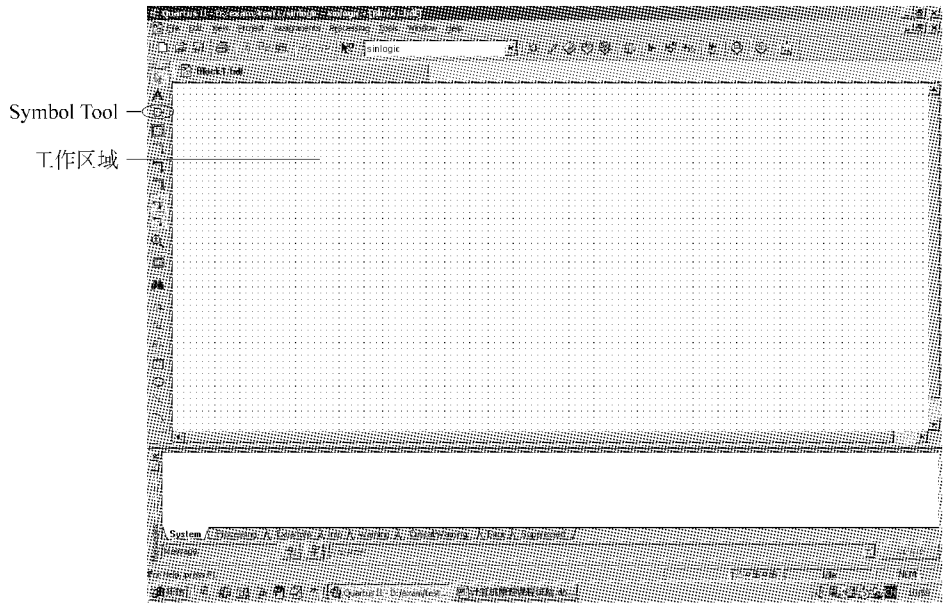

图 3-8　图形文件编辑窗口

具栏中按钮，可以打开系统给出的元件库（见图 3-9）。

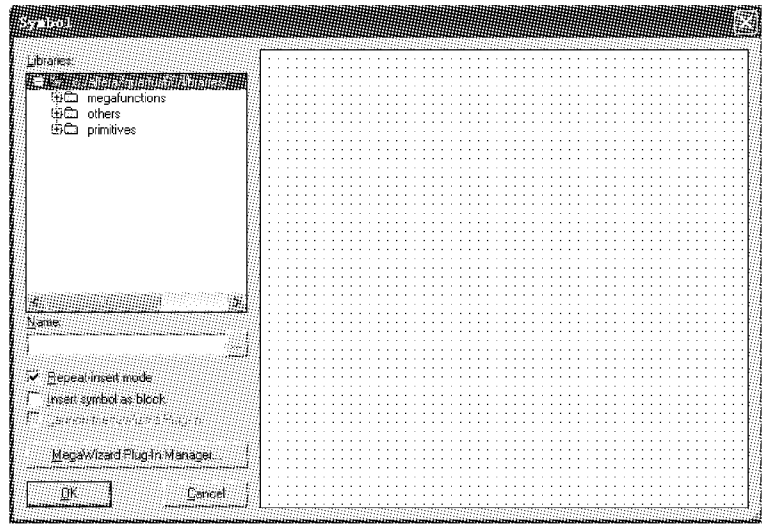

图 3-9　系统元件库

Quartus II 给出的元件库有三部分。第一部分叫 megafunctions，其中有 Arithmetic、Gates、IO 和 Storage 这 4 个文件夹。Arithmetic 中主要放置一些算术运算器件，Gates 中放置的是一些多位的门电路器件，IO 中放置了一些已经设计成功的输入输出器件，Storage 中放置了一些常用的存储器设备。

第二部分叫 others，其中有 maxplus2 和 opencore_plus 两个文件夹，maxplus2 主要

是继承 Quartus Ⅱ的前版本 maxplus2 的库内容而设立的，opencore_plus 是放置系统给出的开放核。

第三部分叫 primtives，是基本的元件库。primtives 中有 buffer、logic、other、pin 和 storage 五部分，这几部分在原理图设计中都很常用。

下面以逻辑表达式 $Y=AB+A'BC'+AC$ 的电路设计来说明原理图设计的基本方法。在图 3-8 工作区域内采用在系统元件库中选择元件来绘制该逻辑电路。具体步骤如下：

(1) 在绘图工具栏内单击 Symbol Tool 按钮，弹出如图 3-10 所示的 Symbol 对话框。

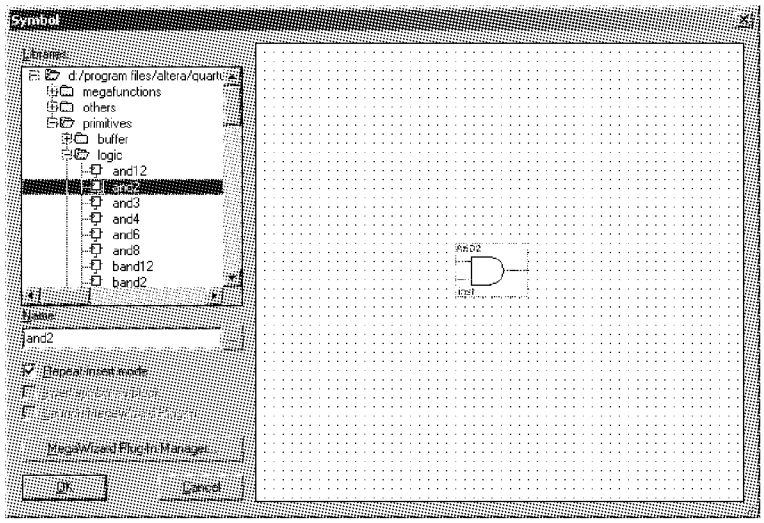

图 3-10 有两个输入端的与门 and2

(2) 在 Libraries 栏中选择 primitives 子目录。

(3) 在 primitives 子目录下选择 logic 目录。

logic 目录中包括一些最基本逻辑元件，如或门、与门、非门、异或门等。

如图 3-10 所示，在打开的 logic 文件夹中，选择有两个输入端的与门 and2，这时 Symbol 对话框右侧会显示元件图形。

(4) 单击 OK 按钮后系统会返回工作区，这时鼠标会带着与门图形移动，在工作区适当位置单击左键即可绘制出一个与门，再将鼠标移到别处单击，还可以得到相同的图形，选择工具栏中的图标，可以去掉单击鼠标绘制图形的功能。

(5) 照此方法在 logic 目录中找到非门 not、有三个输入端的或门 or3 和有三个输入端的与门 and3，分别放到如图 3-11 所示的工作区中的适当位置。

(6) 在工具栏中图标和都凹陷时，按住鼠标拖动，可以直接在元件间连线。

从一个元件的输入输出端点按住鼠标，拖到另一个元件的端点松开，就可以做出一条连线。两条连线对接时，接线处会自动出现实点，这表示这两条连线的导线连通。交叉的两线通过时一般不会出现实点，如果连通，可先对接，出现实点后接着画其余的线。

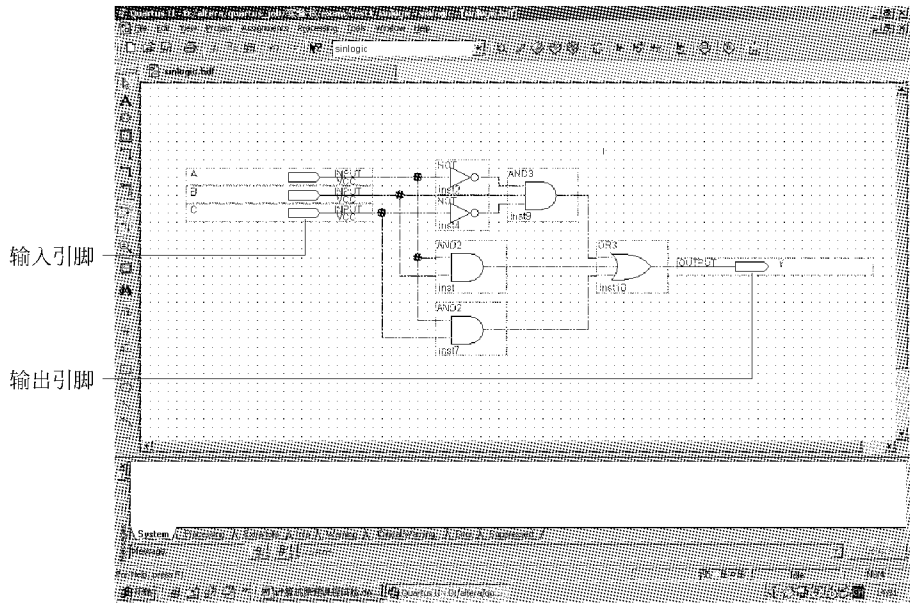

图 3-11　完成的逻辑电路图

将放入工作区的这些元件按照逻辑关系连接到一起，最后要加入逻辑电路图的输入引脚和输出引脚来表示信息的流向。具体做法是在 primitives 目录下的 pin 目录中选择引脚。输入输出引脚共有三种，其中输入引脚是 input，输出引脚是 output，输入输出引脚是 bidir。

为了便于观察，一般情况下输入引脚要放在电路图的最左侧，输出引脚放在电路图的右侧。用鼠标双击放好的引脚，就可对其重新命名。将三个输入引脚分别命名为 A、B、C，输出引脚命名为 Y。完成后的电路如图 3-11 所示。

完成原理图的设计后要将其保存。方法是选择菜单栏中的 File → Save，弹出保存文件对话框，输入文件名 sinlogic. bdf，然后单击【保存】按钮。注意此文件必须是顶层文件，不然不能进行编译。

3.2.2　Verilog HDL 语言设计

用程序设计语言对硬件电路进行描述是电子电路设计极其重要的变革，这种计算机辅助设计的方法依靠可编程逻辑器件的支持，使人们在电子电路的设计阶段，完全可以摆脱实际的电路和电器元件，从而大量地节省了人力、物力和财力。

比较流行的硬件电路描述语言有 VHDL 和 Verilog HDL。VHDL 语言出现较早，Verilog HDL 语言虽然出现较晚一些，但该语言具有 C 程序设计语言风格，很容易被学习过 C 程序设计的人掌握，故普及速度很快。本书用 Verilog HDL 进行电路描述，只介绍一些 Verilog HDL 的一些常用的基本知识，详细内容可参考专门介绍 Verilog HDL 语言的书籍。

Verilog HDL 是硬件描述语言，最初是于 1983 年由 Gateway Design Automation 公

司为其模拟器产品开发的硬件描述语言。那时它只是一种专用语言。由于该公司模拟、仿真器产品的广泛使用，Verilog HDL 作为一种便于使用且实用的语言逐渐为众多设计者所接受。在一次努力增加语言普及性的活动中，Verilog HDL 语言于 1990 年被推向公众领域。Open Verilog International(OVI)是促进 Verilog 发展的国际性组织。1992 年，OVI 决定致力于推广 Verilog OV 标准成为 IEEE 标准。这一努力最后获得成功，Verilog 语言于 1995 年成为 IEEE 标准，称为 IEEE Std1364—1995。

1. Verilog HDL 语言的常量

Verilog HDL 语言中有下列 4 种基本的值。

(1) 0：代表逻辑 0 或“假”

(2) 1：代表逻辑 1 或“真”

(3) x：表示未知

(4) z：表示高阻

注意这 4 种值的解释都内置于 Verilog HDL 语言中。如一个值为 z，则意味着该电路处于高阻抗状态，一个为值 0，通常是指逻辑 0。在门电路的输入或一个表达式中，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 0、x、1、z 与值 0、X、1、Z 相同。Verilog HDL 中的电路常量是由以上这 4 类基本值组成的。

Verilog HDL 程序设计中还使用整型、实数型和字符串型三类常量，整型数常用基数表示法给出。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。

2. Verilog HDL 的数据类型

Verilog HDL 有网线 wire 和寄存器 reg 两大类数据类型。

网线类型表示 Verilog HDL 结构化元件间的物理连线，它的值由驱动元件的值决定，如果没有驱动元件连接到网线，网线的默认值为高阻 z。

寄存器类型表示一个抽象的数据存储单元，它只能在 always 语句和 initial 语句中被赋值，并且寄存器变量赋值之后，值会一直被保存下来。寄存器类型的变量具有 x 的默认值。

不论是网线型数据还是寄存器型数据都可以表述成向量的形式。例如定义，

```
wire [7:0] x;
reg [63:0] jcq;
```

x 是有 8 条线的多股线，jcq 是一个 64 位的寄存器。

向量描述的数据类型还能够选择其中的元素，具体的选择方法是用下标的形式。例如，x[5]表示选择 x 的第 5 条编号线，jcq[3]表示选择的是寄存器 jcq 的第 3 位数。还可以用这种表达方式选择其中的部分。如，x[5:1]表示选择 x 的第 5～1 条编号线，jcq[8:3]表示选择的是 jcq 寄存器的第 8～3 位数。

3. Verilog HDL 的操作符

同其他程序设计语言一样，EDA 硬件描述语言要定义运算和操作符号。其实不论是哪一种运算或操作符都会有实际的电路与之对应，运算或操作的过程最终还是通过

实际的电路来完成的。各种 EDA 硬件描述语言事先已经将各种运算符或操作符电路设计好，在程序设计的时候用符号代替，在编译的时候在将预先设计好的电路添加进去。

Verilog HDL 中的操作符可以分为算术操作符、关系操作符、相等操作符、逻辑操作符、按位操作符、归约操作符、移位操作符、条件操作符、连接和复制操作符等，具体如表 3-1 所示。

表 3-1　Verilog HDL 的操作符

操作符	功　能	操作符	功　能
+	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	>	小于
&	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	^~或~^	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

此表中操作符的优先级排列顺序是，列从上到下，最高优先级（顶行）到最低优先级（底行）排列，处在同一级别操作符优先级相同，在表达式中先完成前面的操作。例如 a−b+c 是先“−”后“+”。

4. Verilog HDL 描述方式

Verilog HDL 语言的描述主要有数据流描述和行为描述两种方式。数据流描述方式主要用 assign 语句，行为描述主要用 always 结构。每个描述语句以“;”结束。

1）assign 连接语句

Verilog HDL 中线路的连接使用 assign 语句描述。assign 语句的格式如下：

```
assign  网线变量=表达式;
```

连接语句只要在右端表达式的操作数上有事件（事件为值的变化）发生时，表达式即被计算，如果结果值有变化，新结果就赋给左边的网线变量。格式右面的表达式常为逻辑运算或条件操作表达式。例如：

```
assign W3=~A & B & ~C;
assign y=x? a:b;
```

条件操作表达式表示的连接连接语句，具有条件变换连接的功能，当 x 的值为 1 时，y 与 a 连接，而当 x 的值为 0 时，y 与 b 连接。

assign 连接语句是并行独立执行的。

2）always 结构

行为描述结构主要有 initial 结构和 always 结构。

一个程序中可以包含任意多个 initial 或 always 结构。这些结构都是相互并行执行的，即这些结构的执行顺序与其书写的顺序无关。一个 initial 结构或 always 结构的执行，产生一个单独的控制流，所有的 initial 和 always 结构都在 0 时刻开始并行执行。

由于本书基本不涉及 initial 结构，故在此不予介绍，重点介绍 always 结构。

always 结构是不断重复执行的，内部语句顺序执行的结构。下面举例说明 always 结构的形式和应用。

例如：

```
always Clk=~Clk;     //将无限循环
```

一行中出现的“//”其后面是注释部分，如果要用多行进行注释，那么注释部分前端要用“/*”，注释的结束处要使用“*/”。

此 always 结构有一个过程性赋值。因为 always 结构重复执行，并且在此例中没有延时控制，过程语句将在 0 时刻无限循环。因此，这种 always 结构语句的执行必须带有某种延时控制。语句的延时控制是由“#”加正整数构成的，如上面的 always 结构，加上延时控制表示如下。

```
always #5 Clk=~Clk;     //产生时钟周期为 10 个单位时间长度的波形
```

always 结构有事件触发的表达方式，形式是在 always 的后面添加“@”字符，并用括号指出事件，对于多个操作在事件发生时产生，要用 begin ... end 的方式将动作语句包含在其中。例如：

```
reg Q,Qbar;
always @(t)
    begin
      #5 Q=1'b1;
      #1 Qbar=~Q;
    end
```

各 always 结构是并行执行的，always 内部的语句都顺序执行的。因而这里描述的是：如果事件 t 发生，经过 5 个单位时间将二进制数 1 送到 Q，之后再经过 1 个单位时间将 Q 的反码送给 Qbar。

这里用“=”连接的赋值方式是要消耗一定时间的，叫阻塞赋值。在顺序结构中阻塞