第三篇　VHDL 的应用

本篇内容

3.1 带你认识 VHDL
3.2 单项训练项目
3.3 综合实训项目——16-4 编码器的设计

3.1　带你认识 VHDL

VHDL 的英文全称是 Very-High-Speed Integrated Circuit Hardware Description Language，翻译成中文意思是超高速集成电路硬件描述语言。

VHDL 语言是 20 世纪 80 年代出现的，它是一种用于电路设计的高级计算机语言，用软件编程的方式来描述电子系统的逻辑功能、电路结构和连接形式。

VHDL 主要应用在数字电路系统的设计中，从最简单的门电路，到组合电路、时序电路以及较为复杂的数字电路系统都可以用 VHDL 来实现。

下面这段 VHDL 程序就是对与门电路的描述。

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
ENTITY AND_GATE IS
PORT(A:IN STD_LOGIC;
     B:IN STD_LOGIC;
     F:OUT STD_LOGIC);
END AND_GATE;

ARCHITECTURE BEHAVE OF AND_GATE IS
BEGIN
F<=A AND B;
END BEHAVE;
```

程序的前两行，即 LIBRARY IEEE;USE IEEE.STD_LOGIC_1164.ALL;，是对程序中所使用到的库和程序包进行声明，和 C 语言中的函数声明类似。

3～7 行定义了与门电路实体的外部端口特征，该电路具有两个输入端口 A 和 B、一个输出端口 F。

8～11 行描述了与门电路的功能，即输出 F 是两个输入 A 和 B 相与的结果：F<=A AND B。

如果读者看不懂这个程序，不要着急，在此只是做一个简单的介绍，详细的语法知识将在以后的单项训练项目中一一介绍。

下面来介绍几个和 VHDL 相关的名词。

1．PLD

PLD 是 Programmable Logic Device 的简称，翻译成中文就是可编程逻辑器件。

一般的集成电路芯片，功能已经设置好，是固定不变的。而可编程逻辑器件 PLD 的优点在于允许用户编程（使用硬件描述语言，如 VHDL）来实现所需要的逻辑功能。用户首先用硬件描述语言来表示所需要实现的逻辑功能，然后经过编译和仿真生成目标文件，再由编程器或下载电缆将设计文件配置到目标文件中，PLD 就变成了能满足用户要求的专用集成电路（ASIC）。

PLD 可以被重复编程，用户可以随时通过程序来修改器件的逻辑功能，而无须改变硬件电路。这个任务在实验室就可以完成，而不必去请芯片制造厂商设计和制作专用的集成电路芯片了。

目前常用的用于编译、仿真、配置的软件有 XILINX 公司的开发软件 FOUNDATION 和 ALTERA 公司的 MAX+PLUS II，后者使用更为简单。

PLD 电路早期代表产品有 XLINX 公司推出的门阵列，称为 FPGA（Field Programable Gate Array）；以及随后 ALTERA 公司推出的 CPLD（Complex Programable Logic Device）。每一片这样的 PLD 可以设计成单片机或者是 CPU 等，并且可以在外部接线完成后重新进行设计。

目前常见的 PLD 生产厂家有 XILINX、ALTERA、ACTEL、LATTIC、ATMEL、MICROCHIP 和 AMD 等，其中 XILINX 和 ALTERA 是两个主要生产厂，XILINX 的主要 PLD 产品为 FPGA，ALTERA 的主要 PLD 产品为 CPLD。

2．FPGA

FPGA 是 Field-Programmable Gate Array 的缩写，即现场可编程门阵列，是由美国的 Xilinx 公司率先推出的，后来有很多厂家也相继推出了相关的 FPGA 产品。

FPGA 是由存放在片内 RAM 中的程序来设置其工作状态的，因此，工作时需要对片内的 RAM 进行编程。用户可以根据不同的配置模式，采用不同的编程方式。

FPGA 的编程无需专用的 FPGA 编程器，只需要用通用的 EPROM、PROM 编程器即可。当需要修改 FPGA 功能时，只需要换一片 EPROM 即可。FPGA 能够反复使用。同一片 FPGA，不同的编程数据可以产生不同的电路功能。因此，FPGA 的使用非常灵活。

3．CPLD

CPLD 的英文全称为 Complex Programmable Logic Device，是 Complex PLD 的简称，中文意思是复杂可编程逻辑器件。

和 FPGA 类似，CPLD 也是一种用户根据各自需要而自行构造逻辑功能的数字集成电路。其基本设计方法是借助集成开发软件平台，用原理图、硬件描述语言等方法，生成相应的目标文件，通过下载电缆（“在系统”编程）将代码传送到目标芯片中，实现设计的数字系统。

CPLD 的主要优点有编程灵活、集成度高、设计开发周期短、适用范围宽、开发工具先进、设计制造成本低、对设计者的硬件经验要求低、标准产品无须测试、保密性强、价格大众化等，可实现较大规模的电路设计。

CPLD 几乎可以用在所有应用中小规模通用数字集成电路的场合，如仪器仪表、汽车电子、数控机床、航天测控设备等方面。

FPGA 和 CPLD 都是 PLD 器件，两者的功能基本相同，只是实现的原理有所区别，所以

有时可以忽略两者的区别，统称为可编程逻辑器件或 CPLD/FPGA。

CPLD 最早由 ALTERA 公司推出，即 MAX 系列，多为 Flash、E^2PROM 架构或乘积项架构的 PLD。FPGA 最早由 Xilinx 公司推出，多为 SRAM 架构，需要外接配置用 EPROM 下载。由于 ALTERA 的 FLEX/ACEX/APEX 系列 PLD 产品也为 SRAM 架构，所以常把 ALTERA 的这几个芯片也叫 FPGA。

使用 CPLD/FPGA，工程师可以通过传统的原理图输入法，或是硬件描述语言（HDL）自由地设计一个数字系统。设计完成后，可以通过软件仿真来验证设计的正确性，可以利用 CPLD/FPGA 的在线修改功能随时修改设计而不必改动硬件电路。

4. 常用的软件仿真工具——MAX+plus II

常用的 FPGA/CPLD 软件仿真工具有 MAX+plus II、Xilinx Foundation、Xilinx ISE、Quartus。其中 MAX+plus II 应用较为广泛。

MAX+plus II 开发工具是 ALTERA 自行设计的 EDA 软件，其界面友好、使用便捷、易学易用。

MAX+plus II 的主要功能包括原理图和文本（HDL 语言）设计，自带综合器、仿真器，支持波形输入、波形模拟、时间分析、编译及下载。

5. 硬件描述语言 HDL

HDL 是一种用形式化方法描述数字电路和系统的语言。利用这种语言，数字电路系统的设计者可以从上层到下层（从抽象到具体）逐层描述自己的设计思想，用一系列分层次的模块来表示极其复杂的数字系统。

目前，广泛使用的硬件描述语言 HDL 有 AHDL、Verilog HDL 和 VHDL。

AHDL（ALTERA 硬件描述语言）是由美国 ALTERA 公司开发出来的一种高级硬件描述语言。AHDL 具有 C 语言的风格，并且易学易用，在 20 世纪 90 年代使用很广泛，但是它的缺点是可移植性较差，只能在 ALTERA 公司的开发系统上使用，因此限制了它的使用范围。

Verilog HDL（Verilog 硬件描述语言）是 1984 年由 GDA 公司推出的，Verilog HDL 在语法结构上源于 C 语言，在某些方面比 C 语言更容易学习和使用。但是它的缺点是初学者容易犯一些设计上的错误，同时它对设计人员的硬件水平要求较高。

VHDL 是 20 世纪 80 年代由美国国防部和 IEEE 发起并资助开发出来的。它的主要特点有：

- 设计可按层次分解。
- 每个设计元素既有定义良好的界面（为了与其他元素连接），又有精确的行为描述（为了模拟）。
- 具有强大的描述能力。
- 具有共享和复用的能力。VHDL 采用基于库的设计方法。库中可以存放大量预先设计或者以前项目设计中曾经使用过的模块，这样设计人员在新项目的设计过程中可以直接使用这些功能模块，从而节约了开发成本。
- 具有良好的可移植能力，可以从一个仿真工具移植到另一个仿真工具，从一个操作平台移植到另一个操作平台。

以上 3 种 HDL 语言，各有其优点和缺点，后两者应用更加广泛，几乎占领了所有的逻辑综合市场，用户学好了一种语言后，很容易地就掌握了另一门语言。

本篇主要介绍的仿真工具是 MAX+plus II，介绍的语言是 VHDL。

3.2　单项训练项目

项目 1　三—八译码器电路的设计

最常用的 CPLD/FPGA 开发工具就是 Max+plus Ⅱ，该软件是 Altera 公司提供的 FPGA/CPLD 开发集成环境，Altera 是世界上最大的可编程逻辑器件供应商之一。Max+plus Ⅱ界面友好，使用便捷，被誉为业界最易用易学的 EDA 软件。

在 Max+plus Ⅱ上可以完成设计输入（原理图输入、波形输入、VHDL 输入等）、元件适配、时序仿真和功能仿真、编程下载整个流程，它提供了一种与结构无关的设计环境，使设计者能方便地进行设计输入、快速处理和器件编程。

下面通过实例来介绍如何在 Max+plus Ⅱ中设计一个三—八译码器电路。

图 3-1 所示是一个三—八译码器的电路图，要求在 MAX+plus II 中分别使用原理图输入、文本设计输入两种方法来实现该设计的输入。

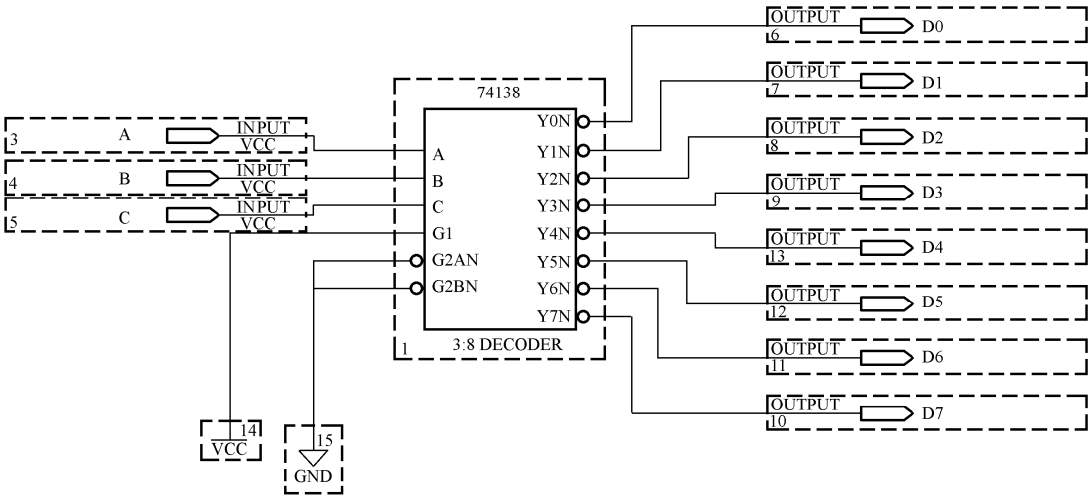

图 3-1　三—八译码器电路图

- 熟悉 MAX+plus II 的界面构成。

- 了解 MAX+plus II 常用菜单的作用。
- 掌握 MAX+plus II 中的设计开发步骤。
- 掌握原理图输入和文本输入两种方法。

执行步骤

方法一：图形输入方式

步骤 1：建立新项目

打开 Max+PlusII，单击 File→Project→Name 命令，弹出 Project Name 对话框，如图 3-2 所示。

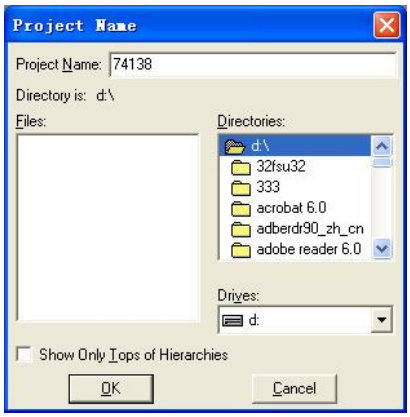

图 3-2　Project Name 对话框

在 Project Name 文本框中输入项目的名字，本项目保存的名字为 74138，在 Drives 下拉列表框中选择保存的根目录，本项目选择 D 盘为保存的目录。在 Directories 列表框中选择保存的详细路径，用户可以根据需要选择，然后单击 OK 按钮。这样在 D 盘就建立了一个名为 74138.acf 的项目文件。

在图 3-2 所示的对话框中，Files 列表框中列出的是所选择路径中包含的 MAX+plus II 文件。

步骤 2：新建图形编辑文件

单击 File→New 命令，弹出如图 3-3 所示的对话框。

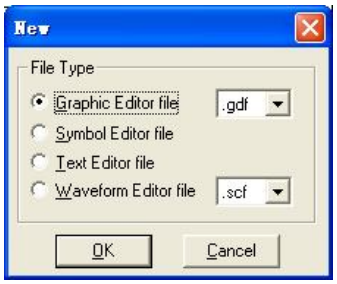

图 3-3　New 对话框

该对话框中有 4 种类型的文件供选择，分别是图形编辑文件（Graphic Editor file）、符号编辑

文件（Symbol Editor file）、文本编辑文件（Text Editor file）、波形编辑文件（Waveform Editor file）。

本项目中需要建立的是图形编辑文件，选择 Graphic Editor file 单选项，单击 OK 按钮。

建立了一个图形编辑文件，如图 3-4 所示。

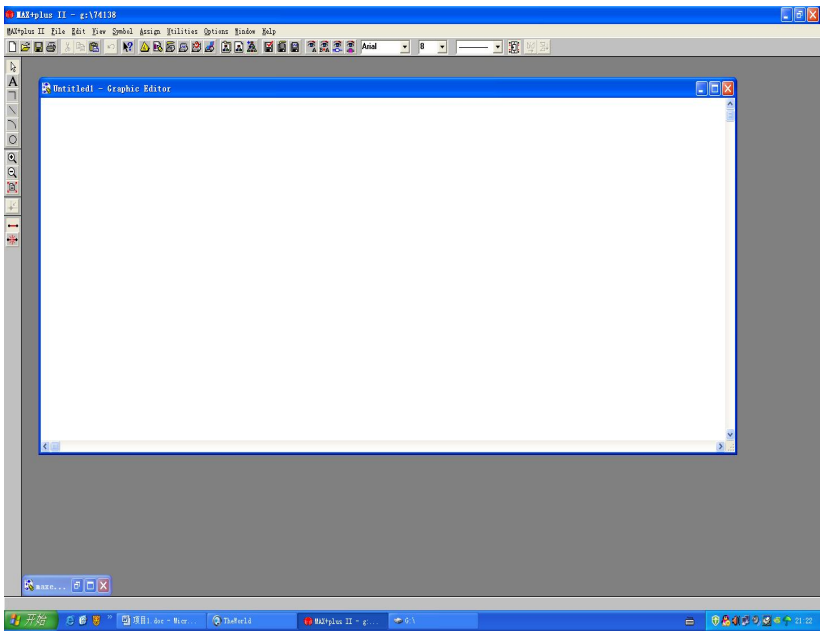

图 3-4　图形编辑窗口

步骤 3：保存文件

单击 File→Save 命令，弹出如图 3-5 所示的对话框。

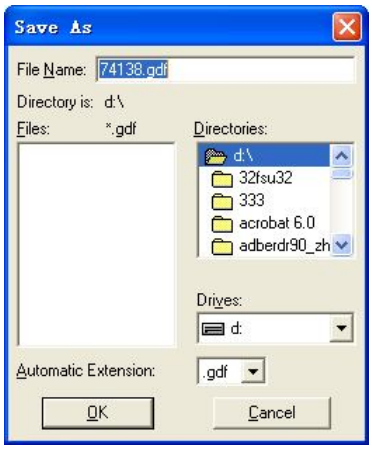

图 3-5　文件保存对话框

在 File Name 文本框中输入保存的文件名，本项目保存名为 74138.gdf。

在 Directories 下拉列表框中选择保存文件的路径，本项目选择 D 盘作为保存的位置。

在 Automatic Extension 下拉列表框中选择保存的类型为.gdf。

注意：原理图的后缀名为.gdf，原理图的文件名可以用设计者认为合适的任何英文名，但

是名称最好和其功能相联系。

步骤 4：在图形编辑窗口中放置图元

在图形编辑窗口中双击，弹出如图 3-6 所示的对话框。

在 Symbol Name 文本框中输入要放置的元件的名字，本项目中先放置元件 74138，所以在该文本框中输入 74138，然后单击 OK 按钮，所选的图元就出现在图形编辑窗口中了，如图 3-7 所示。

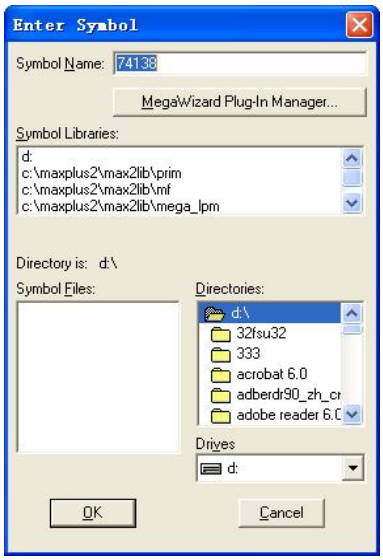

图 3-6　放置图元对话框

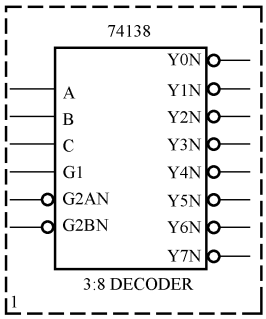

图 3-7　图元 74138

按照上述方法，依次放置输入端口 INPUT、输出端口 OUTPUT、电源 VCC 和接地 GND。

如果需要将图元翻转，则可以在放置好的图元上右击，在弹出的快捷菜单中选择 Rotate，在其级联菜单中选择需要的角度，如图 3-8 所示。

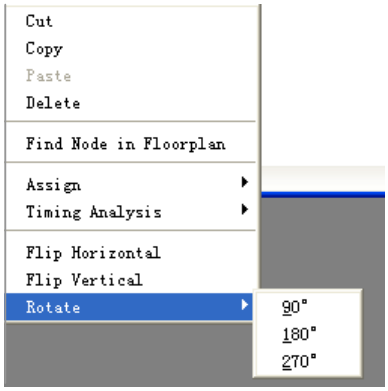

图 3-8　翻转菜单

放置好的图元如图 3-9 所示。

注意：单击对象可以将其选中，如果需要移动某个对象，单击选中对象后，按住左键拖动即可。如果需要删除对象，单击选中后按 Delete 键。

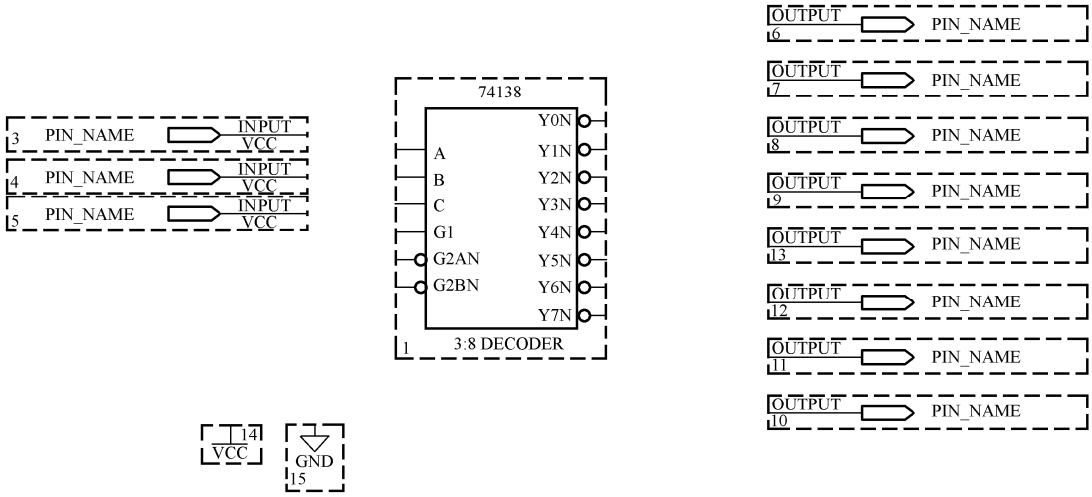

图 3-9　放置好的图元

步骤 5：添加连线

将鼠标指针移到需要连接的引脚附近，等到鼠标指针变为十字形时，按住鼠标左键拖动即可画出直线，松开鼠标可完成直线的绘制。

按照上述方法，参照图 3-1 将图 3-9 中的各图元引脚之间连接起来。

步骤 6：改变输入/输出端口的属性

双击输入端口的名字 PIN_NAME，当其变成黑色时，即可输入端口的标记符，输入完成后按 Enter 键即可。

本项目中，3 个输入端口的标识符分别为 A、B、C，8 个输出端口分别为 D0、D1、D2、D3、D4、D5、D6、D7。完成后的图形如图 3-10 所示。

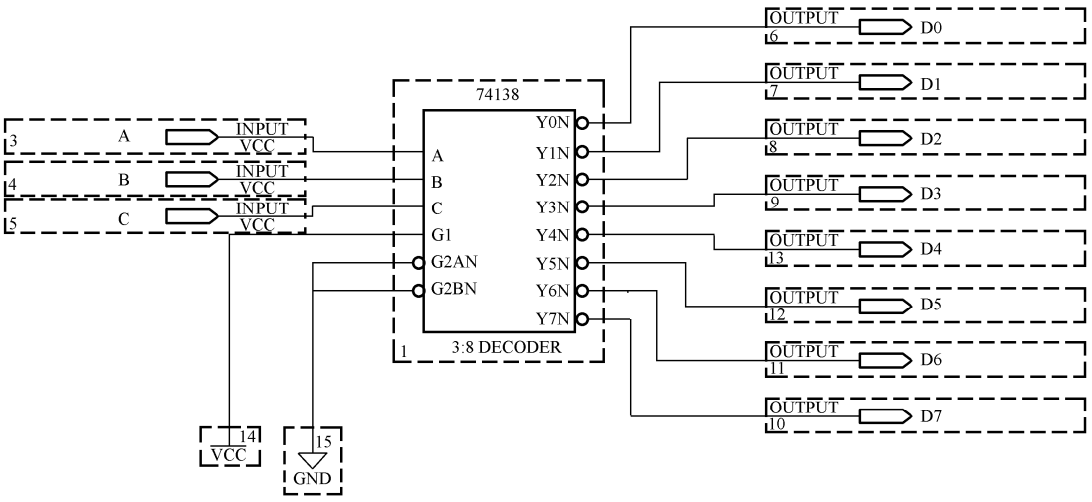

图 3-10　完成的三—八译码器的电路图

单击“保存”按钮，将完成的原理图保存。

知识链接： Max+Plus II 中的元件库及元件的使用

在 Max+Plus II 的安装目录 Maxplus 2 下有一个文件夹 max2lib，在该文件夹中分门别类地存放了 Max+Plus 的所有元件。

在图形编辑窗口中，当需要使用某一个元件时，可以在窗口中双击，会弹出如图 3-6 所示的对话框。

用户在 Directories 列表框中找到安装路径，如 C:\maxplus 2\max2lib\，然后双击其中的某个文件夹，如 Prim，在 Symbol Files 列表框中会列出该文件夹中所存在的所有元件，如图 3-11 所示。

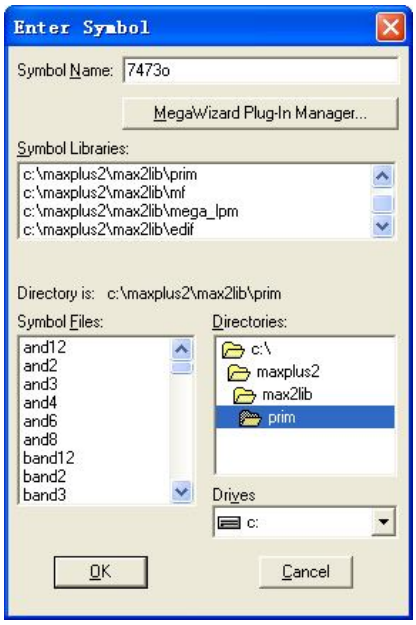

图 3-11　放置图元对话框

用户可以单击选中某个需要的元件，然后单击 OK 按钮将元件放置在图形编辑窗口中。

方法二：文本设计输入方式

下面介绍用 VHDL 语言来描述三—八译码器电路图的过程。关于 VHDL 语言的具体语法规则和使用方法参照后面的项目，本项目只是简单介绍使用 Max+Plus II 来编译仿真 VHDL 的过程。

步骤 1：新建一文本设计文件

单击 File→New 命令，在弹出的 New 对话框中选择 Text Editor file 单选项，新建一文本设计文件。

在文本文件中输入如图 3-12 所示的能够实现 3-8 译码功能的 VHDL 程序代码。

步骤 2：保存文件

单击 File→Save 命令，弹出如图 3-13 所示的对话框。

在 File Name 文本框中输入文件保存的名字 decoder。

在 Directories 列表框中选择文件的保存路径为 C:\max2work。该文件夹在 MAX+plus II 的安装目录下。

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
ENTITY decoder IS
Port(A,B,C,G1,G2A,G2B:IN STD_LOGIC;
     Y0,Y1,Y2,Y3,Y4,Y5,Y6,Y7:OUT STD_LOGIC);
END decoder;

ARCHITECTURE decoder3_8 of decoder IS
BEGIN
process(A,B,C,G1,G2A,G2B)
VARIABLE G:STD_LOGIC;
VARIABLE Y:STD_LOGIC_VECTOR(7 DOWNTO 0);
BEGIN
IF(G1='1'AND G2A='0'AND G2B='0') THEN G:='1';
ELSE G:='0';
END IF;
IF(G='1') THEN
 IF(C='0' AND B='0' AND A='0') THEN Y:="11111110";
 ELSIF(C='0'AND B='0' AND A='1') THEN Y:="11111101";
  ELSIF(C='0'AND B='1'AND A='0') THEN Y:="11111011";
   ELSIF(C='0'AND B='1' AND A='1') THEN Y:="11110111";
    ELSIF(C='1'AND B='0'AND A='0') THEN Y:="11101111";
     ELSIF(C='1'AND B='0'AND A='1') THEN Y:="11011111";
      ELSIF(C='1'AND B='1' AND A='0')THEN Y:="10111111";
       ELSIF(C='1'AND B='1' AND A='1') THEN Y:="01111111";
 END IF;
ELSE Y:="11111111";
END IF;
Y0<=Y(0);Y1<=Y(1);Y2<=Y(2);Y3<=Y(3);Y4<=Y(4);Y5<=Y(5);Y6<=Y(6);Y7<=Y(7);
END PROCESS;
END decoder3_8;
```

图 3-12　VHDL 程序代码

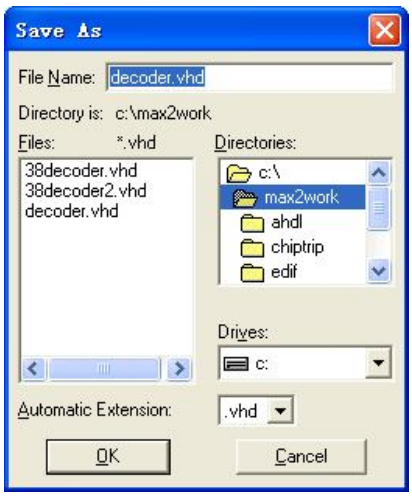

图 3-13　保存对话框

请注意保存文件的名字一定要和实体的名字一致。如果使用到 WORK 库中的内容，则必须将文件保存到 max2work 这个文件夹中；如果没有使用到 WORK 库中的内容，则无此指定要求。

在 Automatic Extension 下拉列表框中选择文件的保存类型为.vhd。

单击 OK 按钮。

注意：保存的类型、名字、位置如果出错，将可能导致编译失败。

步骤 3：项目的编译

单击 File→Project→Set Project to Current File 命令，将这个项目设置为当前的项目文件。

单击 MAX+plus II→Compile 命令，打开如图 3-14 所示的窗口。

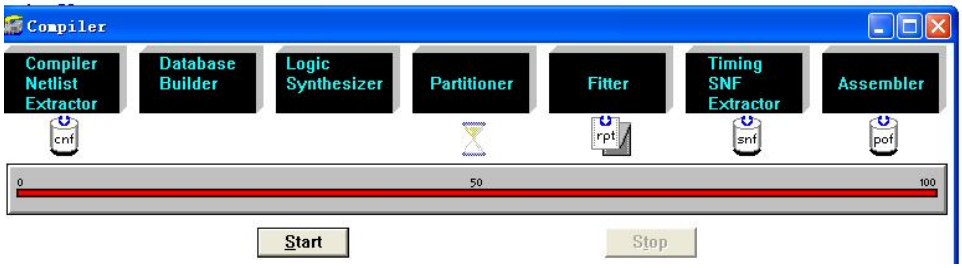

图 3-14　编译窗口

单击 Start 按钮，开始编译。如果有错误，修改后再次进行编译。

步骤 4：生成图元符号

单击 File→Edit Symbol 命令，即可将当前设计的 VHDL 文件创建成一个默认的逻辑符号，本设计中对应的符号名称是 decoder.sym，如图 3-15 所示，该符号在本设计项目中可以供其他文件调用。它所具有的功能和系统所提供的元件是一样的。

步骤 5：新建一图形编辑文件，调用 decoder.sym 完成原理图的设计

按照图 3-3 所示新建一图形编辑文件。

双击界面空白区，将弹出一如图 3-16 所示的对话框。在 Symbol Files 区域中，找到前一步骤中生成的 DECODER 符号，单击 OK 按钮，即可将其放置在图形编辑界面中。然后按照前面所讲的方法依次放置端口、电源、接地，并连线，完成整个电路图的设计。

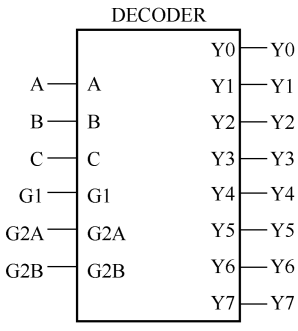

图 3-15　decoder 图元符号

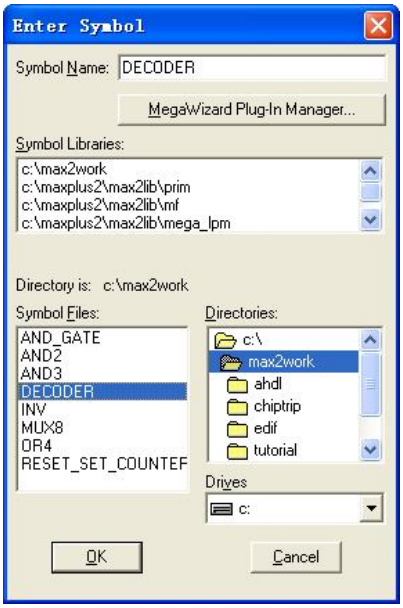

图 3-16　Enter Symbol 对话框

知识链接：使用 Max+Plus II 实现可编程逻辑器件的设计流程

可编程逻辑器件的设计是指利用开发软件和编程工具对器件进行开发的过程，主要步骤如下：

（1）设计准备。在设计开始前，设计者根据任务要求，将所设计的任务按照功能划分为若干个模块，并画出功能框图，确定输入和输出引脚。

（2）设计输入。设计输入有如下几种方式：原理图输入方式、硬件描述语言输入方式、原理图和硬件描述语言的混合方式、波形输入方式。用户可以根据实际情况选择一种方式输入设计。

（3）设计处理。设计处理包括语法检查、设计规则检查、适配和分割等。

（4）设计校验。设计校验过程包括功能仿真和时序仿真。

（5）器件编程。器件编程是指将编程数据下载到可编程逻辑器件中去。

如果读者没有相关的硬件配置，可以在软件中执行前 4 个步骤。

下面通过一个简单的或门电路实例介绍可编程逻辑器件的设计流程(只介绍前 4 个步骤)。

例：二输入或门电路的设计

步骤 1：设计准备——绘制逻辑框图，确定输入/输出引脚

或门电路非常简单，只有一个功能，包含两个输入引脚和一个输出引脚，逻辑框图如图 3-17 所示。

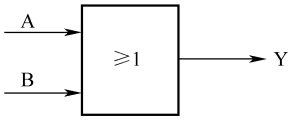

图 3-17　逻辑框图

步骤 2：设计输入

本例采用硬件描述语言输入方式，采用 VHDL 语言描述，如下：

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
ENTITY OR_GATE IS
PORT(A:IN STD_LOGIC;
B:IN STD_LOGIC;
F:OUT STD_LOGIC);
END OR_GATE;
ARCHITECTURE BEHAVE OF OR_GATE IS
BEGIN
F<=A OR B;
END BEHAVE;
```

保存文件，将其保存在安装目录下的 max2work 文件夹中，名为 or_gate，类型为.vhd。

步骤 3：设计处理

（1）对 VHDL 设计进行编译（编译包括语法规则检查）。

单击 File→Project→Set project to current file 命令，将设计文件设置为当前项目。

单击 Max+Plus II→Compile 命令，对设计进行编译，如无错误，将弹出如图 3-18 所示的对话框。

单击“确定”按钮，并关闭掉 Compile 窗口。

图 3-18　编译成功对话框

（2）选定器件。

单击 Assign→Device 命令，将弹出如图 3-19 所示的对话框，用户在其中根据需要选择合适的器件。

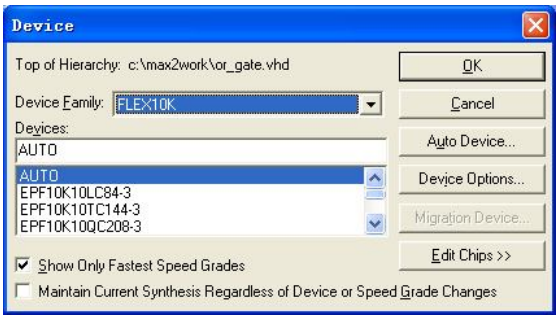

图 3-19　Device 对话框

（3）指定下载方式和保留引脚。

单击 Assign→Global Project Device Option 命令，将弹出一对话框。该对话框中的下载方式和所用器件的配置方式有关，若只仿真而不下载配置数据则可以不设置该对话框。

（4）设置逻辑综合方式。

单击 Assign→Global Logic Synthesis 命令，在弹出的对话框中设置逻辑综合的方式。

（5）将设计电路的管脚和所选芯片的管脚对应起来。

单击 MAX+plusII→floorplan Editor→Layout/Device 命令，在弹出的对话框中进行芯片 I/O 管脚的配置。

步骤 4：设计校验

下面主要介绍时序仿真的步骤，如下：

（1）新建仿真波形文件。

单击 File→New 命令，在弹出的对话框中选择 Waveform Editor file 单选项，如图 3-20 所示。

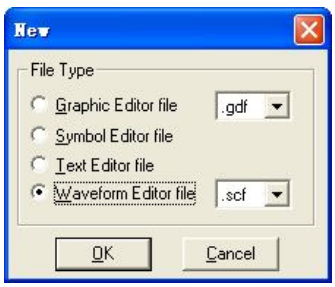

图 3-20　New 对话框

（2）保存仿真波形文件。

单击 File→Save 命令，将波形以名 full_adder、类型.scf、路径 C:/Max2work2 保存起来。新建的仿真波形文件如图 3-21 所示。

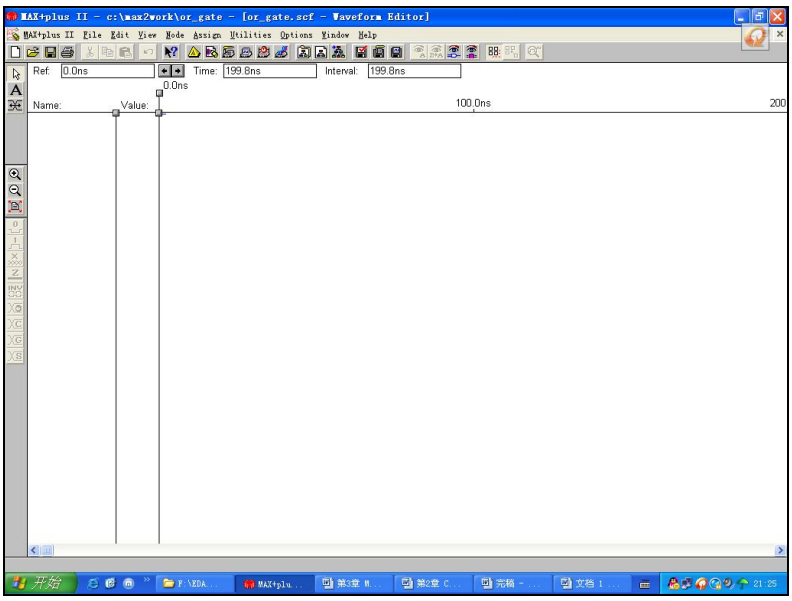

图 3-21　新建的仿真波形文件

（3）添加仿真所需的管脚。

在波形文件空白处右击，在弹出的快捷菜单中选择 Enter Nodes from SNF 选项，弹出如图 3-22 所示的对话框。

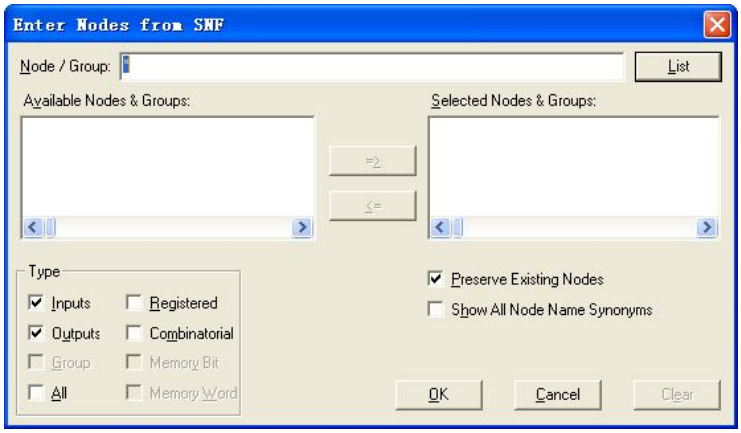

图 3-22　Enter Nodes from SNF 对话框

单击对话框中的 List 按钮，在 Available Nodes & Groups 列表框中列出了所有可以添加的管脚。

单击 => 按钮，将所有的管脚都选中。

单击 OK 按钮。

添加完引脚的波形文件如图 3-23 所示。

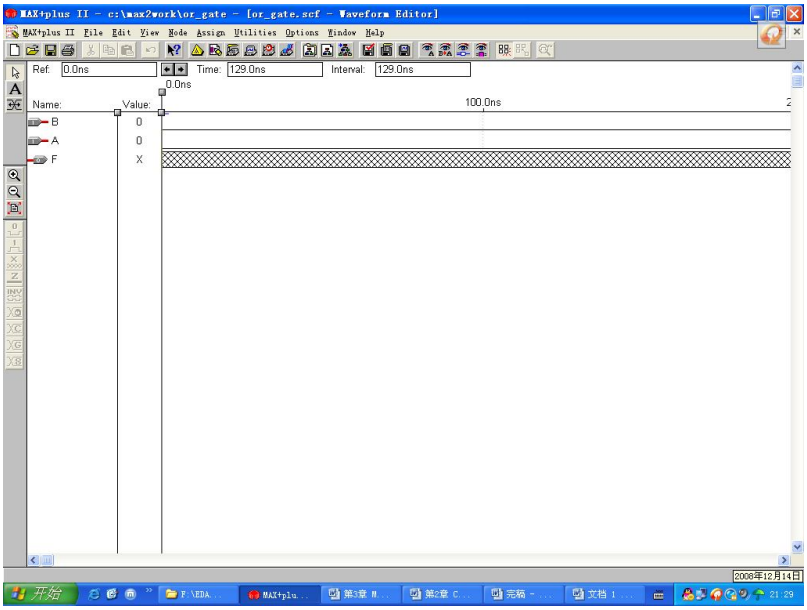

图 3-23　添加完引脚的波形文件

如果需要调整引脚的编号顺序，单击界面左侧的引脚标号（Name），然后按住鼠标左键拖动到合适的位置即可。

（4）添加激励波形。

可以根据需要为两个输入信号 A、B 添加激励信号。

单击输入端口 A，然后单击窗口左侧的时钟信号源图标，将弹出如图 3-24 所示的对话框。

图 3-24　Overwrite Clock 对话框

将初始电平（Starting Value）设置为 0，将时钟周期倍数（Multiplied By）设置为 1。单击 OK 按钮确认。这样就为端口 A 添加了激励信号，单击“全屏显示”按钮。

按照上述方法将 B 的初始电平设置为 1。

单击 MAX+plusII→Simulator 命令仿真，将得到如图 3-25 所示的仿真波形。

用户可以按照需要随意地设置输入信号，仿真器根据输入信号的值计算出输出结果，并将其显示在波形文件中。

（5）延时时间分析。

由图 3-25 可以看出输出信号和输入信号之间存在一定的延时，可以对延时时间进行分析。

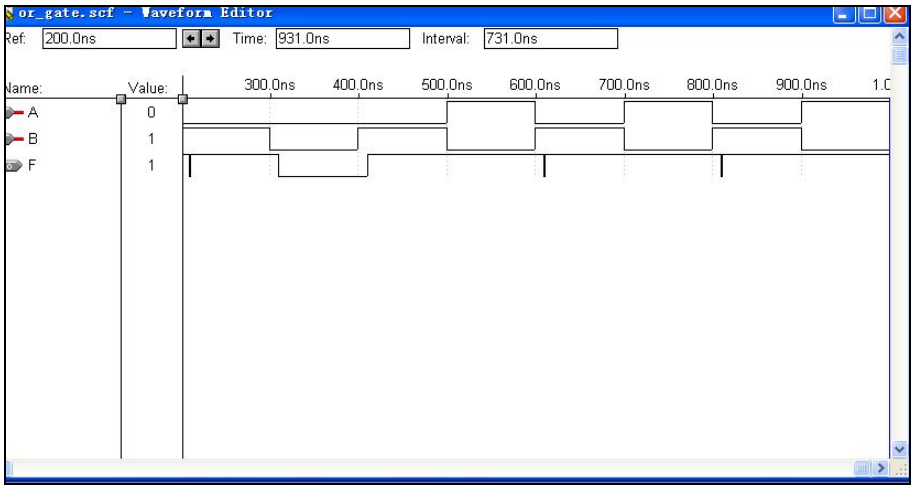

图 3-25　仿真波形

单击 MAX+plus II→Timing Analyzer 命令，将打开 Timing Analyzer 对话框，单击 Start 按钮即可进行电路的延时分析，分析结果如图 3-26 所示。

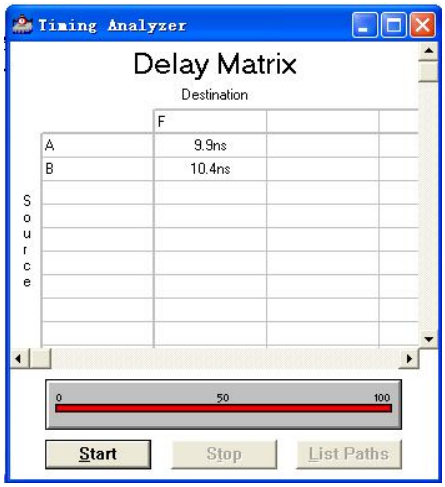

图 3-26　分析结果

Max+Plus II 的输入方法有多种，主要包括：原理图输入、硬件描述语言输入、波形设计输入、层次输入等。设计者可以根据实际情况选择合适的方法。这里主要介绍的是原理图输入方法、文本设计输入法。

如果读者不具有硬件描述语言 HDL 的相关知识，则可以选择原理图输入方式。原理图输入方式直观易懂、容易实现，但是需要具备相关的电路知识，否则如何设计电路图呢？

如果读者具备相关的 HDL 知识，则可以用 HDL 描述所需要实现的逻辑功能，然后用软件编译、仿真、调试，同样也可以达到和原理图输入相同的电路设计目的。

以原理图输入方式绘制如图 3-27 所示的图形。

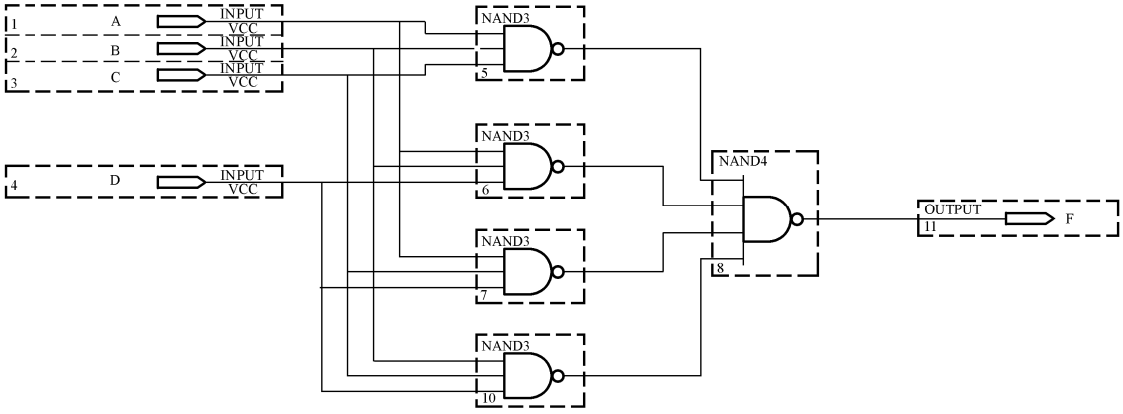

图 3-27　实训电路图

项目 2　用 VHDL 语言设计全加器电路

VHDL 的主要功能就是用来描述数字系统的逻辑功能、电路结构和连接形式。本项目以数字电路中的全加器为例，介绍如何实现用 VHDL 来描述。

表 3-2 所示是一个全加器的真值表，要求用 VHDL 语言来实现对全加器电路的描述。

表 3-2　全加器的真值表

a	b	cin	s	count
0	0	0	0	0
0	1	0	1	0
1	0	0	1	0
1	1	0	0	1
0	0	1	1	0
0	1	1	0	1
1	0	1	0	1
1	1	1	1	1

- 掌握 VHDL 代码的结构。
- 掌握简单 VHDL 代码的编写。
- 理解 VHDL 代码中数据类型、运算符的概念，掌握常见数据类型和运算符的使用。

步骤 1：绘制功能框图，确定全加器电路的输入/输出端口

如果用 VHDL 语言来描述电路，首先需要描述电路具有几个输入端口、几个输出端口，以及这些端口的数据类型。

根据真值表可知，全加器电路具有 3 个输入端口：a、b、cin，两个输出端口 s、count，这些端口都只有两种逻辑状态，即要么为 0，要么为 1。

框图如图 3-28 所示。

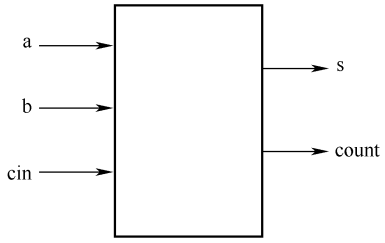

图 3-28　功能框图

步骤 2：实体描述

在 VHDL 代码中，用来描述电路输入/输出端口特征的部分称为实体（ENTITY）描述。

实体（ENTITY）描述部分代码如下：

```
ENTITY   full_adder   IS
PORT(a,b,cin: IN   BIT;
        s, count:OUT BIT);
END full_adder;
```

ENTITY 和 IS 是实体描述的关键字，full_adder 是该实体的名称。ENTITY 和 IS 是固定不变的格式。

PORT 是关键字，其后的()内定义的是端口及其类型。本项目中定义 a、b、cin 为输入端口（IN），占一位；s、count 为输出端口（OUT），占一位。

END 是关键字，和前面的 ENTITY 相对应，表示实体描述结束。

实体描述的一般格式是：

```
ENTITY    entity_name IS
PORT(
port_name: signal_mode    signal_type;
port_name: signal_mode    signal_type;
…
port_name: signal_mode    signal_type);
END entity_name;
```

entity_name 表示实体名，设计者可以根据需要给所设计的实体起相应的名字。

port_name 表示端口名，可以根据需要改变。当需要定义多个模式相同的端口时，可以用逗号分开。

signal_mode 表示端口信号模式，有 4 种类型：IN、OUT、INOUT、BUFFER。

IN（输入模式）表示数据或者信号从外部通过该端口向实体内起作用，如时钟信号、使能信号、复位信号或地址信号等。

OUT（输出模式）表示数据或者信号从该端口输出，向实体外作用。

INOUT（双向模式）表示数据或者信号既可以从实体内部向外输出，对外起作用，又可以由外部流入实体，向实体内部起作用。

BUFFER（缓冲模式）表示数据或者信号既可以从该端口向外作用，也可以将从该端口流出的数据或信号引回设计实体，用来实现内部反馈。常用于时序电路中。

signal_type 表示端口信号类型，常见的类型有 BIT、STD_LOGIC、INTEGER 等。

知识链接：VHDL 中的标识符和关键字

1. 标识符

标识符是用来为常量、变量、信号、端口、实体、结构体、子程序命名的。标识符的命名遵循以下规则：

- 由字母、数字和下划线构成。
- 第一个字符必须是字母。
- 大小写不区分。
- 最后一个字符不能是下划线，而且下划线不能连续出现两个。
- 关键字不能用作标识符。
- 标识符的长度不能超过 32 个字符。

需要提示的是，在 VHDL 中，大小写不区分。

2. 关键字

关键字就是 VHDL 中具有特别意义的单词，只能用作固定的用途，不能用作标识符。在用 Max+Plus 编译 VHDL 程序时，关键字的颜色和其他字符不一样，系统能够自动识别。

实体描述相当于将电路的轮廓勾勒出来了，但是电路还没有功能。那么如何使电路具有指定的功能呢？

步骤 3：描述电路的行为和功能

在 VHDL 代码中，用构造体（ARCHITECTURE）来描述电路的行为和功能。

根据表 3-2 所示的真值表可以得知，当 3 个输入引脚中有奇数个 1 时，s 的值为 1。所以 s 可以用如下表达式表示：s<=a XOR b XOR cin。

而 3 个输入引脚中有两个或两个以上为 1 时，count 的值为 1。可以用逻辑表达式表示为：s<=(a AND b) OR (a AND cin) OR (b AND cin)。

结构体（ARCHITECTURE）描述代码如下：

```
ARCHITECTURE data OF full_adder IS
BEGIN
    s<=a XOR b XOR cin;
    count<=(a AND b) OR (a AND cin) OR (b AND cin);
END data;
```

ARCHITECTURE 和 OF、IS 是结构体部分的关键字。

data 是结构体的名字，用户可以根据自己的需要对结构体命名。

full_adder 是步骤 1 中所描述的实体的名字，这里表示是对该实体进行的结构体的描述。

BEGIN 和 END 是关键字，两者之间是结构体中的功能描述部分，用来具体描述电路的功能（行为）。

结构体部分的一般形式是：

```
ARCHITECTURE architecture_name OF    entity name IS
[declarations]
BEGIN
(code)
END architecture_name;
```

architecture_name 表示结构体名，可以改变。

entity name 是前面定义好的实体的名字。

[declarations] 用来对信号和常量进行声明，该部分可以没有。本例所实现的功能较为简单，没有信号和变量的描述。

[code]是代码部分，用来实现电路的功能。

当 VHDL 代码设计完毕后，用户可以在 Max+Plus II 中调试、编译、仿真，参照项目 1，在此不再详细介绍。

知识链接：VHDL 中的库和包

一段 VHDL 代码一般由 5 个部分组成：库（LIBRARY）声明、包（Package）声明、实体（ENTITY）、结构体（ARCHITECTURE）、配置说明（Configuration）。

实体用来描述电路的输入/输出端口的特征。

结构体用来描述如何运算从而实现电路所要实现的功能。

前面已经讲了实体和结构体的含义和使用，下面介绍库和包的含义和使用。

库是经过编译后的数据的集合，它主要用来存放已经编译过的实体说明、结构体、程序包和配置说明等，以便在其他的 VHDL 设计中可以随时引用这些信息。

在设计单元内的语句可以使用库中的结果，所以库的好处就是设计者可以共享已经编译的设计结果，在 VHDL 中有很多库，但它们相互独立。

VHDL 中常见的库主要包括以下 3 类：

（1）IEEE 库：是使用最为广泛的资源库。IEEE 库主要包括的程序包有 std_logic_1164、numeric_bit、numeric_std，其中 std_logic_1164 是设计人员最常使用和最重要的程序包，该包

主要定义了一些常用的数据类型和函数，如 std_logic、std_ulogic、std_logic_vector、std_ulogic_vector 等类型。

IEEE 库的声明是:

LIBRARY IEEE;

（2）WORK 库：WORK 库可以用来临时保存以前编译过的单元和模块，用户自己设计的模块一般可以存放在该库中。所以，如果需要引用以前编译过的单元和模块，设计人员只需要引用该库即可。

WORK 库的声明:

LIBRARY WORK;

（3）STD 库：STD 库是 VHDL 的标准库，该库中包含了 STANDARD 包和 TEXTIO 包。程序包 STANDARD 中定义了位（BIT）、位矢量（BIT_VECTOR）、字符、时间等数据类型，这是一个使用非常广泛的包。

STD 库的声明:

LIBRARY STD;

库的声明一般形式是:

LIBRARY 库名;

如果要使用库中的包，则还要进行包声明，包声明的一般形式是:

USE LIBRARY 库名.包名.逻辑体名

例如:

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

该例说明要使用 IEEE 库中的 1164 包中的所有项目。

库的作用范围从一个实体说明开始到它所属的结构体、配置为止，当有两个实体时，第二个实体前要另加库和包的说明。

有的库和程序包已经默认包含，所以可以省略库和包的声明语句，如 STD 库中的 STANDARD 包、WORK 库。

在 VHDL 中，一个非常完整的 VHDL 程序是由库、程序包、实体描述、结构体描述和配置说明 5 个部分组成的。通常，一个基本的 VHDL 程序只包括两个基本组成部分：实体（ENTITY）、结构体（ARCHITECTURE）。库、程序包和配置并不是每个程序都必需的，只是在需要的情况下才出现。

库的功能是用来存放已经编译的实体说明、结构体、程序包和配置。

程序包的功能是用来存放各种设计模块都共享的类型、常量和子程序等。

实体说明是用来描述设计所包含的输入/输出端口及其特征。

结构体用来描述设计的行为和结构，即功能是如何实现的。

配置的功能是用来描述设计实体和元件或者结构体之间的连接关系。

用 VHDL 语言来描述一个四输入与门的功能。

四输入与门的逻辑表达式为 F=ABCD。

项目 3　四位与门电路的设计

设计一个与门电路，可以实现两个 4 位二进制的逻辑与运算，如 1011 和 0101 相与的结果是 0001。

- 进一步掌握使用 VHDL 描述数字电路系统的步骤。
- 掌握数据类型和运算符的使用。
- 了解 VHDL 中常用的数据对象。

步骤 1：绘制功能框图，确定输入/输出端口

根据设计要求可以知道该设计有两个输入端口和一个输出端口，分别占 4 位的宽度，命名为 a、b 和 x，如图 3-29 所示。

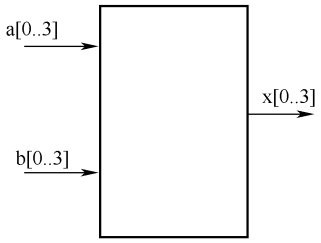

图 3-29　功能框图

步骤 2：实体描述

在 VHDL 中，BIT 表示一个比特（0 或 1）的宽度。如果现在希望输入端口 a 和 b 是 4 位 BIT 的宽度，如 0000、0001、0010 等，那么输入端 a 和 b 该定义为何种类型呢？

显然，不能定义为 BIT 类型。

此时可以将 a 和 b 定义为位矢量类型（BIT_VECTOR）。

所谓位矢量类型，就是一组二进位。BIT 类型只能取值 0 和 1，而 BIT_VECTOR 可以取一组二进位。如 a: IN　BIT_VECTOR(0 TO 3)，表示定义 a 是 BIT VECTOR 类型，长度是 4 位，从 a(0)到 a(3)。

实体描述部分代码如下：

```
ENTITY AND2 IS
PORT(a,b:IN BIT_VECTOR(0 TO 3);
        x:OUT BIT_VECTOR(0 TO 3));
END AND2;
```

知识链接：数据类型

VHDL 中包含了很多的数据类型，这些数据类型是由系统的库所提供的，所以一般在使用之前，需要声明你所用的数据类型是哪个库的哪个包里的。

下面对常用的数据类型进行介绍。

（1）BIT：占一位，有两种取值：0 和 1。

赋值的方式：x<='1';

注意：1 要用单引号括起来。

（2）BIT_VECTOR：位宽可以指定，由若干个 0 和 1 构成。

赋值的方式：x<="0100"

注意，这里是双引号。

（3）BOOLEAN：布尔类型，只有“真”（true）或“假”（false）两种状态。

（4）INTEGER：整型，32 位的整数。和 C 语言里的 int 型差不多。

（5）REAL：实型。

以上 5 种类型包括在 std 库的 standard 程序包中。

所以在使用这 5 个数据类型前，需要进行如下所示的库声明：

```
LIBRARY STD;
USE STD.STANDARD.ALL;
```

不过需要指出的是，在使用 STD 库和 WORK 库时，一般声明可以省略，系统默认。

（6）STD_LOGIC：该种数据类型有 8 个不同的取值，分别为：X、0、1、Z、W、L、H、_。

（7）STD_LOGIC_VECTOR：位矢量。

赋值方式：x:="0001"

或

x:="01XZ"

这两个数据类型包括在 IEEE 库的 STD_LOGIC_1164 包中。

所以在使用之前需要做如下库和包声明：

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

（8）SIGNED 和 UNSIGNED：无符号数只能表示大于 0 的数，而有符号数可以表示整数和负数。

如 1101 是一个无符号数的话，那么它所对应的十进制是 13，如果它是一个有符号数，则它所对应的数是-3。

这两个数据类型包括在 IEEE 库的 STD_LOGIC_ARITH 中，所以在使用之前需要进行如下声明：

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_ARITH.ALL;
```

（9）数组类型。

另一个非常重要的数据类型是数组类型（Array Type）。所谓数组，就是指相同类型元素的有序集合，每个元素都由数组下标（Array Index）来选择。

数组定义的一般形式是：

```
TYPE  数据类型名 IS  ARRAY  范围 OF 原数据类型名;
```

例如：

```
TYPE word IS ARRAY (1 TO 8) OF STD_LOGIC;
```

表示定义了一个名为 word 的数组，包含了从 word(1)到 word(8)的 8 个数组元素，数组元素类型都为 STD_LOGIC。

```
TYPE X IS ARRAY (7 DOWNTO 0) OF INTEGER;
```

表示定义了一个名为 X 的数组，包含了从 word(7)到 word(0)的 8 个数组元素，数组类型都为 INTEGER。

除了上述常用类型外，VHDL 还有一些其他类型，如字符串类型、时间类型，读者如果感兴趣可以参阅相关的参考书。

步骤 3：结构体描述

该设计的功能非常简单，就是对两个输入端口的数进行“与”运算，代码如下：

```
ARCHITECTURE A OF AND2 IS
BEGIN
x<=a AND b;
END A;
```

知识链接：VHDL 中的数据对象和运算符

1．VHDL 中的数据对象

VHDL 中的数据对象有 4 种类型：常量、变量、信号和文件。VHDL 中的数据对象和 C 语言中一样，必须先说明，后使用。

下面主要对常量、变量、信号进行说明。

（1）常量（Constants）。

常量是固定不变的量。对于常量，定义后只能赋值一次。

例如：

```
CONSTANT b: INTEGER:=7;
```

表示将常量 b 定义为整型（INTEGER）常量，并且给其赋值 7。

常量声明的一般格式是：

```
CONSTANT 常量名[,常量名]:数据类型[:=设置值];
```

注意，这里是用“:=”赋值。

（2）变量（Variables）。

变量用来存储中间数据，以便实现存储的算法。

例如：

VARIABLE y: STD_LOGIC;

y:='1';

表示定义了一个变量 y，是 STD_LOGIC 类型，并赋值 7。

注意，这里是用 “:=” 赋值。

变量声明的一般格式是：

VARAABLE 变量名[,变量名]:数据类型[:=设置值];

（3）信号（Signals）。

信号的作用不在于存储数据，它相当于连接线。像端口就是用来外接的，相当于信号。

信号定义的一般形式是：

SIGNAL 信号名:数据类型[<=设置值];

在程序中，信号值输入时采用代入符 “<=”，而不是赋值符 “:=”，同时信号可以附加延时。

信号与变量的区别：信号赋值可以有延迟时间，变量赋值无时间延迟；信号除当前值外还有许多相关值，如历史信息等，变量只有当前值；进程对信号敏感，对变量不敏感；信号可以是多个进程的全局信号，但变量只在定义它之后的顺序域可见；信号可以看作硬件的一根连线，但变量无此对应关系。

2. VHDL 中的运算符

VHDL 中的运算符主要包括算术运算符、关系运算符、逻辑运算符、赋值运算符和关联运算符等。

（1）算术运算符。

+（加）、-（减）、+（取正）、-（取负）、*（乘）、/（除）：用于对整数、实数或物理类型进行运算，和普通的数值运算相同。

MOD（取模）、REM（取余）：用于对整数类型进行运算。

ABS（取绝对值）：用于对任意类型取绝对值。

**（乘方）：左边的操作数，即底数可以是整数或实数，右边的操作数，即指数，必须是整数。

（2）关系运算符。

关系运算符要求运算符两边的数据对象类型必须相同，关系运算符的结果是 BOOLEAN（布尔）类型。

常用的关系运算符有 6 种：

=（等于）、/=（不等于）：可以对所有的数据类型进行操作。

<(小于)、<=(小于或等于)、>(大于)、>=（大于或等于）：用于对整数、实数、BIT、BIT_VECTOR、STD_ULOGIC、STD_ULOGIC_VECTOR 等进行运算。

（3）逻辑运算符。

VHDL 有 7 种逻辑运算符：NOT（逻辑非、取反）、AND（逻辑与）、NAND（逻辑与非）、OR（逻辑或）、NOR（逻辑或非）、XOR（逻辑异或）、NXOR（逻辑异或非）。

逻辑运算符可以应用的数据类型是 BOOLEAN、BIT、BIT_VECTOR、STD_LOGIC、

STD_LOGIC VECTOR。

除了逻辑非以外，其他运算符都是二元运算符，逻辑运算符两侧对象的数据类型必须相同。

NOT 的优先级最高，其他几个逻辑运算符相同。在使用逻辑运算符时，应该将先运算的表达式用括号括起来，以免出现运算错误。如：Y<=(a AND b) OR (c AND d)。

（4）赋值运算符。

VHDL 中，给不同的数据对象赋值时，赋值运算符有所区别。主要有两种类型的赋值运算符：<=和:=。

<=：主要用于对信号赋值；符号和<=（小于或等于）符号一样，但是用在不同的地方，意义不一样。

:=：主要用于对变量赋值。

（5）连接运算符。

&：连接，将两个对象或矢量连接成维数更大的矢量。

VHDL 中的主要数据类型有：BIT、BIT_VECOTR、BOOLEAN、INTEGER、REAL、STD_LOGIC、STD_LOGIC_VECTOR、SIGNED 和 UNSIGNED、数组类型。当在程序中使用到某一数据类型时，需要对该类型所在的库和包进行声明。一般情况下，WORK 库和 STD 库的声明可以省略。

VHDL 中的数据对象主要包括：常量、变量和信号。如果在结构体描述部分需要使用到某一对象，应该在结构体中的声明部分对数据对象进行声明，然后才能使用。

VHDL 中的运算符非常丰富，包括算术运算符、关系运算符、逻辑运算符、赋值运算符和关联运算符等。

在用 VHDL 描述电路时，要注意将不同的对象定义为合适的类型，另外还要注意对不同类型的对象进行运算时要使用合适的运算符。

用 VHDL 分别描述一个四位的或门电路、与非门电路、或非门电路。

项目 4　八选一数据选择器的设计

用 VHDL 语言来设计一个 8 选 1 电路。该电路有 3 个选择输入端和 8 个数据输入端，一个输出端。

该 8 选 1 电路实现的功能是：

当 A2 A1 A0 取值为 000 时，Y=D0。

当 A2 A1 A0 取值为 001 时，Y=D1。
当 A2 A1 A0 取值为 010 时，Y=D2。
当 A2 A1 A0 取值为 011 时，Y=D3。
当 A2 A1 A0 取值为 100 时，Y=D4。
当 A2 A1 A0 取值为 101 时，Y=D5。
当 A2 A1 A0 取值为 110 时，Y=D6。
当 A2 A1 A0 取值为 111 时，Y=D7。

- 理解顺序语句的概念。
- 掌握 IF 语句、CASE 语句的使用。
- 理解 LOOP 语句、NEXT 语句、EXIT 语句、WAIT 语句的含义及用法。

步骤 1：绘制功能框图，确定输入/输出引脚

根据要求可知，该电路有 11 个输入引脚和一个输出引脚，如图 3-30 所示。

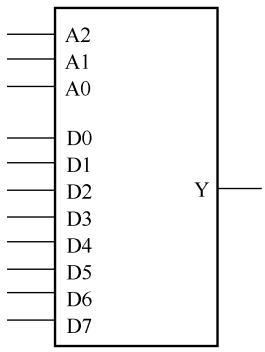

图 3-30　功能框图

步骤 2：库和包声明

将该设计中的输入输出端口类型定义为 STD_LOGIC_VECTOR 类型，该类型包含在 IEEE 库的 STD_LOGIC_1164 中。

所以在使用之前需要进行如下库和包声明：

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

步骤 3：实体描述

实体描述比较简单，如下：

```
ENTITY mux8 IS
PORT(A:IN STD_LOGIC_VECTOR(0 TO 2);
     D:IN STD_LOGIC_VECTOR(0 TO 7);
     Y:OUT STD_LOGIC);
END mux8;
```

步骤 4：结构体描述

根据任务要求可知，该任务是对 A2 A1 A0 的值进行判断，由判断的结果决定执行什么样的操作，是条件选择结构。VHDL 中实现条件选择主要有两种语句：IF 语句和 CASE 语句。

（1）IF 语句。

VHDL 中的 IF 语句就是根据指定的条件来确定执行哪些语句的。

IF 语句有如下 3 种类型：

- 只有一个条件分支的结构：

```
IF(分支)  THEN
    顺序处理语句;
END IF;
```

- 有两个条件分支的结构

```
IF(条件)  THEN
    顺序处理语句 1;
ELSE
    顺序处理语句 2;
END IF;
```

- 有多个条件分支的结构

```
IF  条件 1  THEN
    顺序处理语句 1;
ELSIF  条件 2  THEN
    顺序处理语句 2;
    …
ELSIF  条件 N-1  THEN
    顺序处理语句 N-1;
ELSE
    顺序处理语句 N;
END IF;
```

本任务是多个条件分支结构，可以用 IF 语句的第 3 种结构实现如下：

```
ARCHITECTURE aa OF mux8 IS
BEGIN
PROCESS(A,D)
BEGIN
   IF (A="000") THEN Y<=D(0);
   ELSIF (A="001") THEN Y<=D(1);
```

```
    ELSIF (A="010") THEN Y<=D(2);
    ELSIF (A="011") THEN Y<=D(3);
    ELSIF (A="100") THEN Y<=D(4);
    ELSIF (A="101") THEN Y<=D(5);
    ELSIF (A="110") THEN Y<=D(6);
    ELSE Y<=D(7);
    END IF;
END PROCESS;
END aa;
```

（2）CASE 语句。

CASE 语句是 VHDL 提供的另一种形式的条件选择语句，和 IF 语句相比较，CASE 语句更适合于多分支条件选择结构。

CASE 语句的一般格式是：

```
CASE 表达式 IS
    WHEN 条件表达式 1=>顺序处理语句 1;
    WHEN 条件表达式 2=>顺序处理语句 2;
    WHEN OTHERS=>顺序处理语句 3;
END CASE;
```

该任务用 CASE 语句可以实现如下：

```
ARCHITECTURE aa OF mux88 IS
BEGIN
PROCESS(A,D)
BEGIN
    CASE A IS
    WHEN "000"=>Y<=D(0);
    WHEN "001"=>Y<=D(1);
    WHEN "010"=>Y<=D(2);
    WHEN "011"=>Y<=D(3);
    WHEN "100"=>Y<=D(4);
    WHEN "101"=>Y<=D(5);
    WHEN "110"=>Y<=D(6);
    WHEN OTHERS =>Y<=D(7);
    END CASE;
END PROCESS;
END aa;
```

执行的过程是：当 CASE 和 IS 之间的表达式的取值和某个 WHEN 后面的条件表达式的值一致时，程序就执行该 WHEN 后面的=>后的顺序处理语句。

CASE 分支的个数没有限制，但是 CASE 语句中的选择（即条件表达式）必须是唯一的，不能重复，除了 OTHERS 以外，其他 CASE 分支的顺序可以任意排列。

知识链接：顺序语句

顺序语句是按出现的次序执行的。

顺序语句主要有信号赋值语句、变量赋值语句、IF 语句、CASE 语句、LOOP 语句、NEXT 语句、EXIT 语句、NULL 语句和 WAIT 语句。

（1）信号赋值语句。例如：

```
a<=b;
```

（2）变量赋值语句。例如：

```
c:=a+d;
```

（3）IF 语句。

前面已经详细介绍了。

（4）CASE 语句。已详细介绍。

（5）LOOP 语句。LOOP 语句能够使程序有规则地循环。LOOP 语句的格式有如下两种：

1）FOR 循环语句。FOR 循环语句的一般格式：

```
[循环标号:] FOR  循环变量  IN  范围  LOOP
顺序处理语句;
END LOOP [循环标号];
```

范围表示的是循环变量在循环过程中依次取值的范围。

例如：

```
ASUM: FOR i IN 1 TO 9 LOOP
    sum=i+sum;
END LOOP ASUM;
```

i 的值依次从 1 取值到 9，和 sum 累积相加。

2）WHILE 循环语句。

WHILE 语句的一般格式：

```
[循环标号:] WHILE(条件) LOOP
顺序处理语句;
END LOOP [循环标号];
```

如果条件为真，则循环；如果条件为假，则结束循环。

例如：

```
sum:=0;I:=1;
abcd: WHILE (I<10) LOOP
    sum:=I+sum;
    I:=I+1;
END LOOP abcd;
```

表示在 I 小于 10 的条件下，不断地实现 sum 与 I 相加和 I 增 1，直到 I>=10 终止。

（6）NEXT 语句。

NEXT 语句主要用于 LOOP 语句的内部循环控制，有条件或无条件地跳出本次循环。

一般格式：

```
NEXT [循环标号][WHEN  条件];
```

循环标号表示下一次循环的起始位置。

WHEN 条件表示 NEXT 语句执行的条件，条件为真，则退出本次循环，同时转入下一次循环；条件为假，则不执行 NEXT 语句。如果既无“循环标号”又无“WHEN 条件”，则只要执行 NEXT 语句就立即无条件跳出本次循环，并从 LOOP 语句的起始位置进入下一次循环。

NEXT 语句用于控制内循环的结束。

（7）EXIT 语句。

EXIT 语句用于结束 LOOP 循环状态。

一般格式:

EXIT [循环标号] [WHEN 条件]

当 EXIT 语句后没有循环标号和 WHEN 条件时，则程序执行到该处则无条件跳出 LOOP 语句，接着执行 LOOP 语句后面的语句；当后面有循环标号时，则执行 EXIT 语句，程序将无条件地从循环标号所指明的循环中跳出；当后面有 WHEN 条件时，则执行 EXIT 语句，只有在所给所有条件为真时，才跳出 LOOP 语句，执行下一条语句。

（8）NULL 语句。

空操作语句，只占位置。

（9）WAIT 语句。

表明将程序挂起，并视条件决定再次执行。

一般格式有如下几种:

- WAIT;: 表示无限等待。
- WAIT ON 信号表;: 敏感信号量变化，激活运行程序。
- WAIT UNTIL 条件表达式;: 条件为真，激活运行程序。
- WAIT FOR 时间表达式;: 时间到，运行程序继续执行。

小结

在顺序语句中，所有的语句都是顺序执行的，因此语句出现的顺序很重要。VHDL 中的顺序执行语句主要有信号赋值语句、变量赋值语句、IF 语句、CASE 语句、LOOP 语句、NEXT 语句、EXIT 语句、NULL 语句和 WAIT 语句。

实训

编写一个 3-8 译码器，用 IF 语句实现，包括 3 个输入端 A0、A1、A2 和 8 个输出端 D0～D7。

3-8 译码器的功能是：

当 A0 A1 A2 分别为 000 时，D0=1，其他位为 0。

当 A0 A1 A2 分别为 001 时，D1=1，其他位为 0。

当 A0 A 1A2 分别为 010 时，D2=1，其他位为 0。

当 A0 A1 A2 分别为 011 时，D3=1，其他位为 0。

当 A0 A1 A2 分别为 100 时，D4=1，其他位为 0。

当 A0 A1 A2 分别为 101 时，D5=1，其他位为 0。

当 A0 A1 A2 分别为 110 时，D6=1，其他位为 0。
当 A0 A1 A2 分别为 111 时，D7=1，其他位为 0。

项目 5　异步置位/复位边沿 D 触发器的设计

设计一个边沿 D 触发器，带异步置位端和复位端。

- 理解并行语句的概念。
- 掌握常用的并行语句的使用。

步骤 1：绘制功能框图，确定输入/输出端口

带异步置位端口和复位端口的 D 触发器的功能框图如图 3-31 所示。

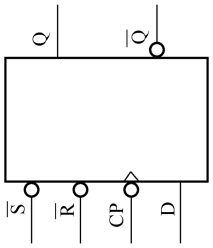

图 3-31　功能框图

步骤 2：库和包的声明

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

步骤 3：实体描述

```
ENTITY    DFF IS
PORT(D: IN    STD_LOGIC;
CP: IN    STD_LOGIC;
R:    IN    STD_LOGIC;
S:    IN    STD_LOGIC;
Q:    OUT STD_LOGIC;
```

```
NQ: OUT STD_LOGIC);
END DFF;
```

步骤 4：结构体描述

CP 是时钟脉冲信号；D 是数据输入端；R 是复位端，低电平有效；S 是触发器的置位端，低电平有效；Q 和 NQ 是两个输出端，NQ 是 Q 信号的反向，两个输出端互补。

边沿触发器的工作原理是：当 S 端为 0 时，为有效信号，无论其他输入端口是什么，输出端 Q 为 1，NQ 为 0；当 R 端为 0 时，为有效信号，无论其他输入端口是什么，输出端 Q 为 0，NQ 为 1；当 S=R=1 时，都为无效信号，此时如果 CP 是下降沿，则 Q<=D;，NQ<=NOT Q;。

那么该如何表示 CP 下降沿的条件呢？

CP’event AND CP='0'表示检测时钟下降沿有效。

CP’event AND CP='1'表示检测时钟上升沿有效。

知识链接：信号属性

信号属性用来表示信号的某种特性，例如：

signal A:std_logic_vector(3 downto 12);表示定义 A 为一个矢量信号，包含从 A（3）到 A（12）共 10 位；A’left 表示信号 A 的左边界值，A 的左边界值在此处为 3；A’right 表示信号 A 的右边界值，A 的右边界值在此处为 12；A’high 表示 A 的最高端边界值，为 12；A’low 表示 A 的最低端边界值，为 3；A’length 表示 A 的长度，为 10；A’range 表示 A 的范围，为 3 to 12。

event 属性表示信号内涵改变，if(clk’event and clk='1')表示信号 clk 发生变化而且 clk 的值为 1，即 clk 上升沿触发，也可以用如下语句表示：

```
if(rising_edge(clk))
```

结构体描述部分的代码如下：

```
ARCHITECTURE RT1 OF DFF1 IS
BEGIN
PROCESS(CP,S,R)
BEGIN
IF(S='0'AND R='1') THEN
Q<='1';NQ<='0';
ELSIF(S='1'AND R='0')THEN
Q<='0';NQ<='1';
ELSIF(CP'event AND CP='0') THEN
Q<=D;
NQ<=NOT D;
END IF;
END PROCESS;
END RT1;
```

知识链接：并行语句的概念

VHDL 中的代码按照执行顺序可以分为并发代码和顺序代码两大类。

所谓顺序代码，就是按照语句的顺序逐行执行的。

所谓并发代码，语句是并发执行的，和语句的先后顺序无关。

VHDL 中的并发描述语句有进程语句（PROCESS）、并行信号赋值语句、条件信号赋值语句、选择信号赋值语句、并行过程调用语句、块语句、元件例化语句和生成语句等。

下面着重介绍 PROCESS 语句、并行信号赋值语句、条件信号赋值语句和选择信号赋值语句，其他并行语句在下一个项目中有详细描述。

1. PROCESS 语句

在 VHDL 中，进程语句是使用最为频繁、应用最为广泛的一种语句。一般来说，一个结构体可以包含一个或者多个进程语句，结构体的各个进程语句之间是一组并发行为，也就是各进程语句是并行执行的；但在每一个进程语句中，组成进程的各个语句则是顺序执行的。

进程语句的一般形式：

```
[进程名称:]PROCESS[(敏感信号表)]
        [说明部分;]
        BEGIN
         [顺序语句;]
         END PROCESS[进程名称];
```

进程名称可有可无，敏感信号表列出进程对其敏感的所有信号，如上例的 D 触发器中，因为 R、S、CP 只要有一个或多个发生变化，进程就会被启动，所以 R、S、CP 被列为敏感信号。

说明部分用于说明进程内部使用的数据的类型、子程序或者变量；顺序语句用于描述一个数字电子硬件电路的工作过程，可以反复执行。

2. 并行信号赋值语句

在编写 VHDL 程序的过程中，设计人员经常会采用两种类型的信号赋值语句：一种是应用于进程和子程序内部的信号赋值语句，这时它是一种顺序语句，又称顺序信号赋值语句；另外一种是应用于进程和子程序外部的信号赋值语句，这时称为并行语句，又称并行信号赋值语句。

例如：

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
ENTITY AND_GATE IS
PORT(A,B:IN STD_LOGIC;
Y: OUT STD_LOGIC);
END AND_GATE;
ARCHITECTURE RT OF AND_GATE IS
BEGIN
Y<=A AND B;
END RT;
```

本例中的“Y<=A AND B;”就是一条并行信号赋值语句。

通常情况下，一条并行信号赋值语句等价于一个进程语句，因此也可以将一条并行信号赋值语句改写成相应的进程语句。上例的与门改写如下：

```
ARCHITECTURE RT OF AND_GATE IS
```

```
PROCESS (A,B);
BEGIN
Y<=A AND B;
END PROCESS;
END RT;
```

3. 条件信号赋值语句

条件信号赋值语句指的是根据不同条件将不同的表达式赋给目标信号的一种并行信号赋值语句，它是一种应用较为广泛的信号赋值语句。

条件信号赋值语句的一般形式如下:

```
信号量<=表达式 1      WHEN    条件 1      ELSE
        表达式 2      WHEN    条件 2      ELSE
        …
        表达式 N-1    WHEN    条件 N-1    ELSE
        表达式 N;
```

判断过程是: 当满足条件 1 时，将表达式 1 的值赋给信号量; 否则
当满足条件 2 时，将表达式 2 的值赋给信号量; 否则
…
当满足条件 N-1 时，将表达式 N-1 的值赋给信号量; 否则
将表达式 N 的值赋给信号量。

例如，项目 4 中的八选一数据选择器可以用条件信号赋值语句实现如下:

```
ARCHITECTURE aa OF mux8 IS
BEGIN
Y<=D(0) WHEN A="000" ELSE
   D(1) WHEN A="001" ELSE
   D(2) WHEN A="010" ELSE
   D(3) WHEN A="011" ELSE
   D(4) WHEN A="100" ELSE
   D(5) WHEN A="101" ELSE
   D(6) WHEN A="110" ELSE
   D(7) WHEN A="111" ELSE
   'Z';
END aa;
```

4. 选择信号赋值语句

选择信号赋值语句是指根据选择条件表达式的值将不同的表达式赋值给目标信号。

选择信号赋值语句的一般形式是:

```
WITH   选择表达式   SELECT
信号量<=表达式 1   WHEN   选择值 1,
        表达式 2   WHEN   选择值 2,
        …
```

表达式 N　WHEN　选择值 N;

判断过程是：判断选择表达式的值，如果和选择值 1 相同，将表达式 1 的值赋给信号量；如果和选择值 2 的值相同，将表达式 2 的值赋给信号量；……如果和选择值 N 的值相同，则将表达式 N 的值赋给信号量。

项目 4 中的八选一数据选择器可以用条件信号赋值语句实现如下:

```
ARCHITECTURE aa OF mux8 IS
BEGIN
WITH A SELECT
Y<=D(0) WHEN "000",
    D(1) WHEN "001",
    D(2) WHEN "010",
    D(3) WHEN "011",
    D(4) WHEN "100",
    D(5) WHEN "101",
    D(6) WHEN "110",
    D(7) WHEN "111",
    'Z'   WHEN OTHERS;
END aa;
```

在实际的电子系统中，几乎所有的操作都是并发执行的，这些操作之间没有具体的顺序之分，一旦得到事件触发，它们就会开始并行工作。VHDL 所提供的并行语句即可实现这种并行操作性。

进程语句（PROCESS）、并行信号赋值语句、条件信号赋值语句、选择信号赋值语句是几种常见的并行语句。在用户使用 VHDL 的时候，可以根据需要选择其中的一种来使用。

分别用进程语句、条件信号赋值语句、选择信号赋值语句来实现四选一数据选择器。

该选择器包括两个地址输入端 A0、A1 和 4 个数据输入端 D0、D1、D2、D3，一个输出端 Y，还有一个使能端口 EN。

工作原理：如果 EN=1，则电路开始工作。

当 A1 A0 为 00 时，输出 Y 为 D0。

当 A1 A0 为 01 时，输出 Y 为 D1。

当 A1 A0 为 10 时，输出 Y 为 D2。

当 A1 A0 为 11 时，输出 Y 为 D3。

项目 6　四位移位寄存器的设计

训练任务

图 3-32 所示是四位移位寄存器，由 4 个相同的 D 触发器构成（不带置位端和复位端）。要求用 VHDL 实现该设计。

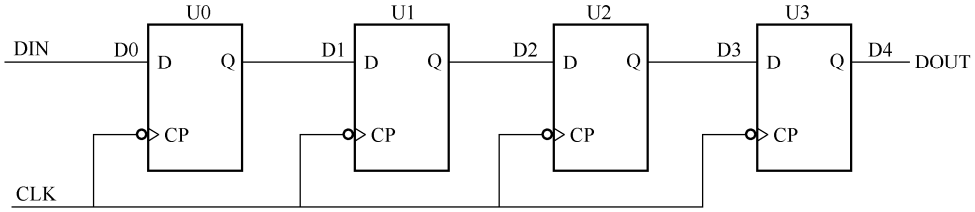

图 3-32　四位移位寄存器

学习目标

- 了解层次化设计思想，掌握层次化设计方法。
- 掌握元件例化语句的使用。
- 了解参数传递语句、生成语句的作用和用法。

执行步骤

知识链接：层次化设计思想

图 3-32 所示的四位移位寄存器是由 4 个 D 触发器构成的。

对于这样一个功能较为复杂的电路系统而言，可以采用层次化的设计思想。所谓层次化设计思想，就是将经常使用到的或者使用比较多的单元模块定义好并编译完成后，将其存放在常用库的程序包中。当设计人员需要使用这些常用的功能模块时，可以对其“调用”。

在 VHDL 中，库中单元模块的调用是通过例化元件说明语句和例化元件映射语句实现的，它们常常被分别称为 COMPONENT 语句和 PORT MAP 语句。

步骤 1：定义单元模块，并将其存放在库中

D 触发器的 VHDL 代码如下：

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
ENTITY D_FF IS
```

```
PORT(D,CP:IN STD_LOGIC;
     Q:OUT STD_LOGIC);
END D_FF;
ARCHITECTURE DFF OF D_FF IS
BEGIN
PROCESS(CP)
BEGIN
IF(CP'EVENT AND CP='0') THEN
Q<=D;
END IF;
END PROCESS;
END DFF;
```

将该段程序代码存放在 WORK 库的 example 程序包中。

步骤 2：绘制功能框图，确定输入/输出端口

观察图 3-32，可以看出该图具有 3 个端口，功能结构框图如图 3-33 所示。

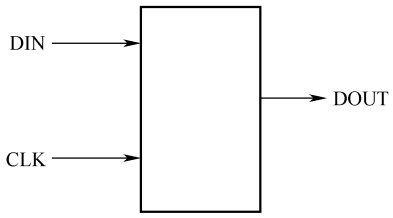

图 3-33　功能框图

步骤 3：库声明和实体描述

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

ENTITY SHIFTER4 IS
PORT(DIN,CLK:IN STD_LOGIC;
     DOUT:OUT STD_LOGIC);
END SHIFTER4;
```

步骤 4：结构体描述

```
ARCHITECTURE SS OF SHIFTER4 IS
COMPONENT D_FF
PORT(D,CP:IN STD_LOGIC;
     Q:OUT STD_LOGIC);
END COMPONENT D_FF;
SIGNAL D:STD_LOGIC_VECTOR(0 TO 4);
```

```
BEGIN
D(0)<=DIN;
U0:D_FF PORT MAP(D(0),CLK,D(1));
U1:D_FF PORT MAP(D(1),CLK,D(2));
U2:D_FF PORT MAP(D(2),CLK,D(3));
U3:D_FF PORT MAP(D(3),CLK,D(4));
DOUT<=D(4);
END SS;
```

下面对结构体描述部分的代码进行解释。

（1）例化元件说明语句 COMPONENT。

```
COMPONENT D_FF
PORT(D,CP:IN STD_LOGIC;
     Q:OUT STD_LOGIC);
END COMPONENT D_FF;
```

这一段是例化元件说明语句，又称为 COMPONENT 语句，一般放在结构体的说明部分，作用是说明结构体中所要调用的元件或者单元模块。本例中声明的是在步骤 1 中编写好的元件 D 触发器 D_FF。比较可以发现，COMPONENT 语句和步骤 1 中的实体描述语句一样，列出了每个端口的名字、模式和类型。需要注意的是，COMPONENT 中的元件名和元件声明中的名字必须一致，端口名称、模式、数据类型也必须相同。

COMPONENT 语句的语法结构如下所示：

```
COMPONENT <引用元件名>
[GENERIC<参数说明>;]
PORT<端口说明>;
END COMPONENT;
```

（2）信号定义语句。

```
SIGNAL D:STD_LOGIC_VECTOR(0 TO 4);
```

该语句定义了一个信号类型，由图 3-32 可以看到，D0、D1、D2、D3、D4 是电路中实际存在的中间连线，所以在此要定义为信号。变量不具有这样的特点，变量只是短暂的表示，没有相应的实际物理信号。

（3）例化元件映射语句 PORT MAP。

```
U0:D_FF PORT MAP(D(0),CLK,D(1));
U1:D_FF PORT MAP(D(1),CLK,D(2));
U2:D_FF PORT MAP(D(2),CLK,D(3));
U3:D_FF PORT MAP(D(3),CLK,D(4));
```

该段语句的作用是为了把调用的元件或者单元模块的端口信号与结构体中的相应端口信号进行正确连接，又称为映射语句。

PORT MAP 语句的一般形式是：

```
<标号:><元件名>[GENERIC MAP(参数映射)]
        PORT MAP(端口映射);
```

标号是用来表征该语句的标识符，它在结构体描述中应该是唯一的；元件名应该与 COMPONENT 语句中的引用元件名保持一致，也就是要与存放在库中的被调用元件或单元模块的 VHDL 程序的实体名一致。

映射语句的功能就是用来将调用的元件或者单元模块的端口信号与结构体中的相应端口信号进行正确的连接，从而达到引用元件的目的。可以看出，映射语句中的信号一般为当前结构体中的实际信号。

COMPONENT 语句和 PORT 语句是一种应用广泛的并行描述语句，主要用在层次化的设计中。采用这两种语句可以直接使用库中已经存在的元件或单元模块，避免了大量重复 VHDL 程序的书写工作，从而节省了大量的时间。

知识链接：并行语句之块语句和生成语句

块语句和生成语句也是两种常见的并行语句。

1. 块语句（BLOCK）

当一个电路较复杂时，可以考虑使用块语句将它划分为几个模块。

例：用块语句描述如图 3-34 所示的电路。

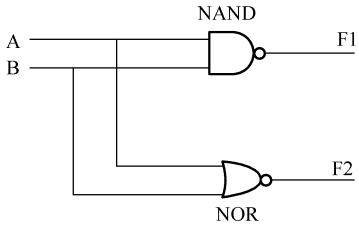

图 3-34　示例电路

代码如下：

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
ENTITY SS IS
PORT(A,B:IN STD_LOGIC;
     F1,F2:OUT STD_LOGIC);
END SS;
ARCHITECTURE FF OF SS IS
BEGIN
B1:BLOCK
BEGIN
F1<=A NAND B;
END BLOCK B1;
B2:BLOCK
BEGIN
F2<=A NOR B;
END BLOCK B2;
END FF;
```

块语句 B1：

```
B1:BLOCK
BEGIN
F1<=A NAND B;
END BLOCK B1;
```

该块语句名称为 B1，没有说明部分，实现的功能很简单，就是与非（NAND）功能。

块语句 B2：

```
B2:BLOCK
BEGIN
F2<=A NOR B;
END BLOCK B2;
```

该块语句名称为 B2，实现的是或非（NOR）功能。

由上可知，块语句定义的一般形式是：

```
块名称:BLOCK
[接口说明;]
[说明部分;]
BEGIN
并行处理语句;
END BLOCK  块名称;
```

2. 生成语句

生成语句用来描述电路中有规则和重复性的结构。

本项目中的四位移位寄存器用生成语句实现如下（只列出结构体部分，实体部分一样）：

```
ARCHITECTURE SS OF SHIFTER4 IS
COMPONENT D_FF
PORT(D,CP:IN STD_LOGIC;
     Q:OUT STD_LOGIC);
END COMPONENT D_FF;
SIGNAL D:STD_LOGIC_VECTOR(0 TO 4);
BEGIN
D(0)<=DIN;
G:FOR i IN 0 TO 3 GENERATE
U:D_FF PORT MAP(D(i),CLK,D(i+1));
END GENERATE G;
DOUT<=D(4);
END SS;
```

其中，

```
G:FOR i IN 0 TO 3 GENERATE
    U:D_FF PORT MAP(D(i),CLK,D(i+1));
    END GENERATE G;
```

是生成语句，生成的是 4 个相同的 D 触发器，从 D(0) 到 D(4)。

生成语句的常见形式是：

```
[标号:] FOR 循环变量 IN 取值范围  GENERATE
[类属说明;]
并行处理语句;
END GENERATE [标号];
```

在编写 VHDL 程序的过程中，通常可以按照语句的执行顺序将其分为并行描述语句和顺序描述语句。

并行描述语句是指在语句的执行过程中，语句的执行顺序和语句的书写顺序无关，所有语句都是并发执行的。常见的并行描述语句有：进程语句、并行信号赋值语句、条件信号赋值语句、选择信号赋值语句、块语句、元件例化语句和生成语句。

顺序描述语句是指在语句的执行过程中，语句的执行顺序是按照语句的书写顺序依次执行的。常见的顺序描述语句有：信号赋值语句、IF 语句、CASE 语句、LOOP 语句、NEXT 语句、EXIT 语句、NULL 语句、WAIT 语句。

结构体中的各个模块之间是并行执行的，因此应该采用并行语句来进行描述；而各个模块内部的语句则需要根据描述方式来决定，即模块内部既可以采用并行描述语句，同时也可以采用顺序描述语句。

采用层次化设计方法实现如图 3-35 所示的电路，可以采用元件例化语句或元件生成语句实现。

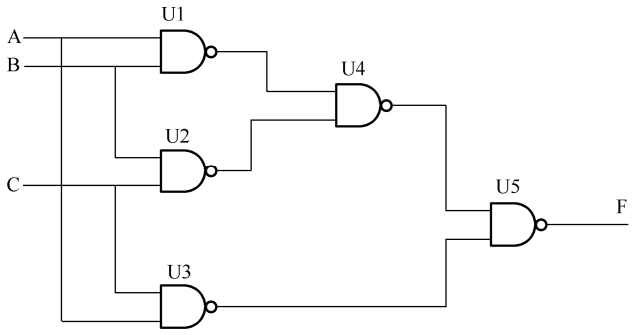

图 3-35　实训电路

3.3　综合实训项目——16–4 编码器的设计

任务 1：用 VHDL 描述一个带使能端 E1 和输出状态端 E0 和 GS 的 8 线-3 线优先编码器，所有信号都是低电平有效。当 E1=1 时，电路停止工作，输出全为 1；当 E1=0 时，电路进行

正常的编码工作；如果电路正常工作，且有信号输入，则 GS=0，其他状态下 GS=1；如果电路正常工作，但无信号输入，E0=0，其他状态下 E0=1。

任务 2：用任务 1 中已经设计好的 8 线-3 线编码器作为单元模块，设计一个 16-4 编码器。要求带一个使能端和两个输出状态端，作用和任务 1 中相同。

以上两个任务都要求画出逻辑框图，编写出 VHDL 代码，并且用 MAX+Plus II 仿真出其波形。

综合运用 VHDL 知识设计一个较为复杂的电路系统。

任务 1：

步骤 1：绘制功能框图，确定输入/输出端口

根据任务要求，确定有几个输入端口和几个输出端口，并绘制出功能框图，参照图 3-36 所示。

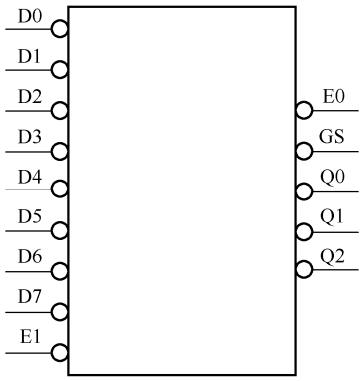

图 3-36　功能框图

步骤 2：库和包声明

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

步骤 3：实体描述

```
ENTITY encoder_priority IS
PORT(D:IN STD_LOGIC_VECTOR(7 DOWNTO 0);
     E1:IN STD_LOGIC;
     Q:OUT STD_LOGIC_VECTOR(2 DOWNTO 0);
```

```
      GS:OUT STD_LOGIC;
      E0:OUT STD_LOGIC);
END encoder_priority;
```

步骤 4：结构体描述

```
ARCHITECTURE RT OF encoder_priority IS
BEGIN
PROCESS(E1,D)
BEGIN
    IF(E1='1') THEN
        Q<="111";
        GS<='1';
        E0<='1';
    ELSIF(D="11111111" AND E1='0') THEN
        Q<="111";
        GS<='1';
        E0<='0';
    ELSIF(D(7)='0' AND E1='0') THEN
        Q<="000";
        GS<='0';
        E0<='1';
    ELSIF(D(6)='0' AND E1='0') THEN
        Q<="001";
        GS<='0';
        E0<='1';
    ELSIF(D(5)='0' AND E1='0') THEN
        Q<="010";
        GS<='0';
        E0<='1';
    ELSIF(D(4)='0' AND E1='0') THEN
        Q<="011";
        GS<='0';
        E0<='1';
    ELSIF(D(3)='0' AND E1='0') THEN
        Q<="100";
        GS<='0';
        E0<='1';
    ELSIF(D(2)='0' AND E1='0') THEN
        Q<="101";
        GS<='0';
```

```
        E0<='1';
    ELSIF(D(1)='0' AND E1='0') THEN
        Q<="110";
        GS<='0';
        E0<='1';
    ELSIF(D(0)='0' AND E1='0') THEN
        Q<="111";
        GS<='0';
        E0<='1';
    ELSE
        Q<="111";
        GS<='1';
        E0<='0';
    END IF;
END PROCESS;
END RT;
```

步骤 5：在 MAX+plus II 中编译、仿真

编译仿真波形如图 3-37 所示。

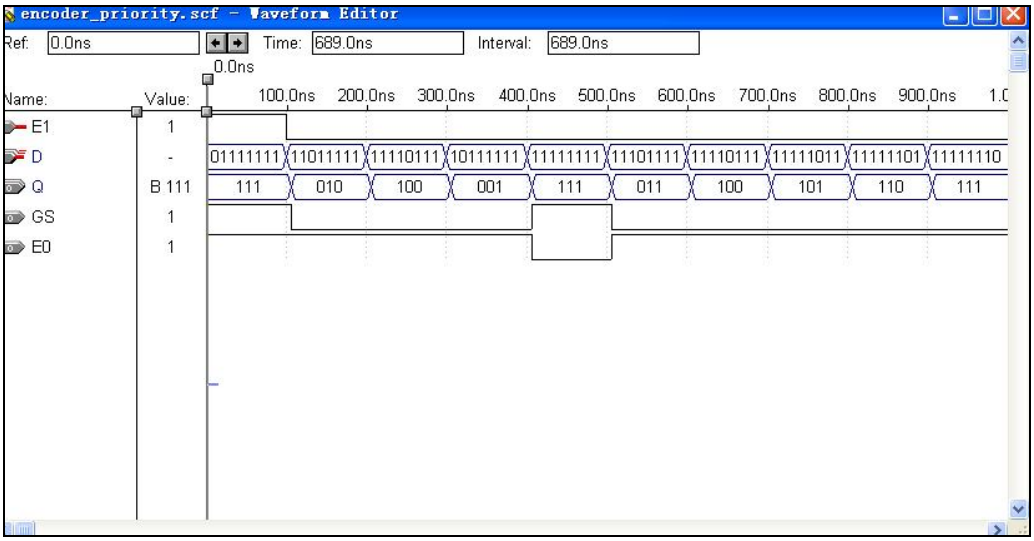

图 3-37　编译仿真波形

任务 2：

步骤 1：绘制功能框图、总体结构框图，确定输入输出端口

功能框图如图 3-38 所示。

16 线-4 线可以由两个 8 线-3 线编码器构成，总体框图如图 3-39 所示。

根据总体框图可以看出，系统包括两个 8 线-3 线编码器模块 U1 和 U2、3 个与非门模块 U3～U5、1 个非门模块 U6。请读者仔细观察图 3-39，明白各模块之间的信号流向。

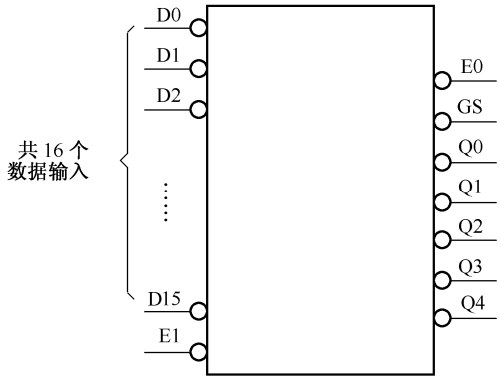

图 3-38　功能框图

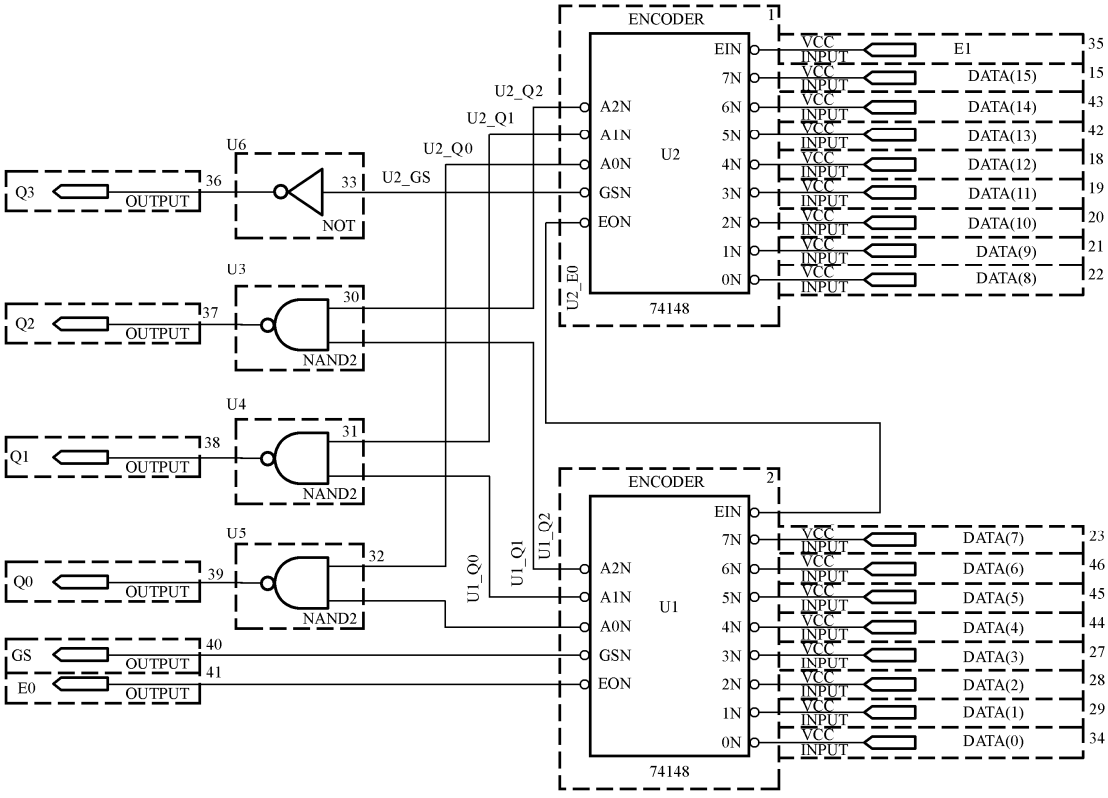

图 3-39　总体框图

高位（左侧）的 8 线-3 线编码器模块 U2 包含的接口如下：

- 数据端 D15～D8：输入信号。
- 使能端 E1：输入信号。
- 数据输出端 U2_Q2、U2_Q1、U2_Q0：输出信号。
- 状态输出端 U2_GS、U2_E0：输出信号。

低位（右侧）的 8 线-3 线编码器模块 U1 包含的接口如下：

- 数据端 D7～D0：输入信号。

- 使能端 U2_E0：输入信号。
- 数据输出端 U1_Q2、U1_Q1、U1_Q0：输出信号。
- 状态输出端 GS、E0：输出信号。

与非门 U3 包含的接口如下：

- 数据输入端 U2_Q2、U1_Q2：输入信号。
- 输出端 Q2：输出信号。

与非门 U4 包含的接口如下：

- 数据输入端 U2_Q1、U1_Q1：输入信号。
- 输出端 Q1：输出信号。

与非门 U5 包含的接口如下：

- 数据输入端 U2_Q0、U1_Q0：输入信号。
- 输出端 Q0：输出信号。

非门 U6 包含的接口如下：

- 数据输入端 U2_GS：输入信号。
- 输出端 Q3：输出信号。

步骤 2：各功能模块设计

（1）8 线-3 线编码器 encoder8_3。

参照任务 1。

（2）与非门 ANDX。

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
ENTITY ANDx IS
PORT(I0,I1:IN STD_LOGIC;
     O1:OUT STD_LOGIC);
END ANDx;
ARCHITECTURE SS OF ANDx IS
BEGIN
O1<=I0 AND I1;
END SS;
```

（3）非门 INV。

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
USE IEEE.std_logic_arith.ALL;
USE IEEE.std_logic_unsigned.ALL;

ENTITY INV IS
PORT(I:IN STD_LOGIC;
     O:OUT STD_LOGIC);
END INV;
```

```
ARCHITECTURE RT1 OF INV IS
BEGIN
O<=NOT I;
END RT1;
```

步骤 3：16 线-4 线编码器的顶层 VHDL 设计

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

ENTITY encoder16_4 IS
PORT(DATA:IN STD_LOGIC_VECTOR(15 DOWNTO 0);
     E1:IN STD_LOGIC;
     Q:OUT STD_LOGIC_VECTOR(3 DOWNTO 0);
     GS:OUT STD_LOGIC;
     E0:OUT STD_LOGIC);
END encoder16_4;

ARCHITECTURE ST OF encoder16_4 IS
COMPONENT ANDX
PORT(I0,I1:IN STD_LOGIC;
     O1:OUT STD_LOGIC);
END COMPONENT;

COMPONENT INV
PORT(I:IN STD_LOGIC;
     O:OUT STD_LOGIC);
END COMPONENT;

COMPONENT encoder_priority
PORT(D:IN STD_LOGIC_VECTOR(7 DOWNTO 0);
     E1:IN STD_LOGIC;
     Q:OUT STD_LOGIC_VECTOR(2 DOWNTO 0);
     GS:OUT STD_LOGIC;
     E0:OUT STD_LOGIC);
END COMPONENT;

SIGNAL U2_Q:STD_LOGIC_VECTOR(2 DOWNTO 0);
SIGNAL U2_GS:STD_LOGIC;
SIGNAL U2_E0:STD_LOGIC;
SIGNAL U1_Q:STD_LOGIC_VECTOR(2 DOWNTO 0);
```

```
SIGNAL DATA_1:STD_LOGIC_VECTOR(7 DOWNTO 0);
SIGNAL DATA_2:STD_LOGIC_VECTOR(7 DOWNTO 0);
BEGIN
DATA_2(7)<=DATA(15);
DATA_2(6)<=DATA(14);
DATA_2(5)<=DATA(13);
DATA_2(4)<=DATA(12);
DATA_2(3)<=DATA(11);
DATA_2(2)<=DATA(10);
DATA_2(1)<=DATA(9);
DATA_2(0)<=DATA(8);
DATA_1(7)<=DATA(7);
DATA_1(6)<=DATA(6);
DATA_1(5)<=DATA(5);
DATA_1(4)<=DATA(4);
DATA_1(3)<=DATA(3);
DATA_1(2)<=DATA(2);
DATA_1(1)<=DATA(1);
DATA_1(0)<=DATA(0);
U0:encoder_priority PORT MAP(DATA_2,E1,U2_Q,U2_GS,U2_E0);
U1:encoder_priority PORT MAP(DATA_1,U2_E0,U1_Q,GS,E0);
U2:ANDX PORT MAP(U2_Q(2),U1_Q(2),Q(2));
U3:ANDX PORT MAP(U2_Q(1),U1_Q(1),Q(1));
U4:ANDX PORT MAP(U2_Q(0),U1_Q(0),Q(0));
U5:INV PORT MAP(U2_GS,Q(3));
END ST;
```

步骤 4：在 MAX+plus II 中编译、仿真

仿真波形如图 3-40 所示。

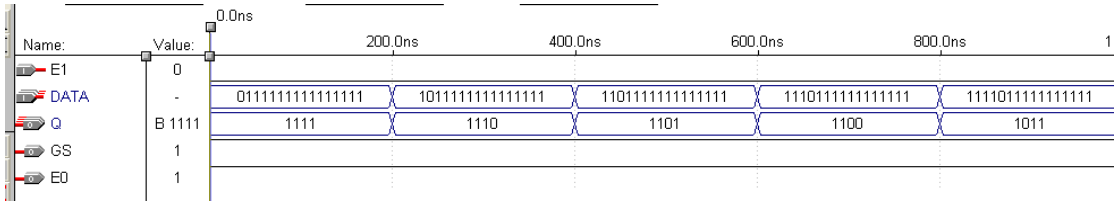

图 3-40　仿真波形

对于一个较为复杂的电路而言，可以采用层次化的设计方法，使系统设计变得简洁和方

便。层次化设计是分层次、分模块进行设计描述的。描述器件总功能的模块放在最上层，称为顶层设计；描述器件某一部分功能的模块放在下层，称为底层设计。用户既可以采用自顶向下的描述方式，也可以采用自底向上的描述方式。

当采用层次化设计方式时，在使用 MAX+plus II 编译仿真时用户需要将设计好的单元模块（底层设计）和顶层设计文件存放在同一个文件夹中，否则可能会出现无法调用的情况。

参考文献

[1] 赵景波，王劲松．Protel 2004 电路设计从基础到实践．北京：电子工业出版社，2007．

[2] 黎文模，段晓峰．Protel DXP 电路设计与实例精解．北京：人民邮电出版社，2006．

[3] 林庭双，柯常志．Protel DXP 电子电路设计精彩范例．北京：机械工业出版社，2005．

[4] 谭会生，瞿遂春．EDA 技术综合应用实例与分析．西安：西安电子科技大学出版社，2007．

[5] 杨晓慧，许红梅．电子技术 EDA 实践教程．北京：国防工业出版社，2005．

[6] 马淑华，高原．电子设计自动化．北京：北京邮电大学出版社，2006．

[7] 赵建领编著．Protel 电路设计与制板宝典．北京：电子工业出版社，2007．

[8] 郭勇，董志刚编著．Protel 99 SE 印制电路板设计教程．北京：机械工业出版社，2006．

[9] 程昱，刘建伟编著．Protel DXP 电路设计白金教学．北京：科学出版社，北京科海电子出版社，2004．

[10] 林庭双，柯常志等编著．Protel DXP 电子电路设计精彩范例．北京：机械工业出版社，2005．

[11] 蒋卓勤，邓玉元．Multisim 2001 及其在电子设计中的应用．西安：西安电子科技大学出版社，2003．

[12] 郑步生，吴渭．Multisim 2001 电路设计及仿真入门与应用．北京：电子工业出版社，2002．

[13] 聂典．Multisim 9 计算机仿真在电子电路设计中的应用．北京：电子工业出版社，2007．

[14] 宋嘉玉，孙丽霞．EDA 实用技术．北京：人民邮电出版社，2006．

[15] John F.Wakerly 著．数字设计原理与实践（第四版）．林生，葛红，金京林译．北京：机械工业出版社，2007．

[16] 赵鑫，蒋亮，齐兆群，李晓凯．VHDL 与数字电路设计．北京：机械工业出版社，2005．

[17] 北京理工大学 ASIC 研究所．VHDL 语言 100 例详解．北京：清华大学出版社，1999．

[18] 姜雪松，吴钰淳，王鹰．VHDL 设计实例与仿真．北京：机械工业出版社，2007．