第7章 ASIC设计

第7章 ASIC设计

- 7.1 ASIC的设计手段
- 7.2 GAL器件的编程及应用
- 7.3 CPLD/FPGA器件的编程及应用
- 7.4 SOPC简介

7.1 ASIC的设计手段

7.1.1 ASIC设计的发展历程

- 集成电路的设计方法和手段经历了几十年的发展演变，从最初的全手工设计发展到现在的可以全自动实现的过程。这也是近几十年来计算机技术、半导体技术和半导体集成电路技术等，尤其是电子信息技术发展的结果。从集成电路设计手段的发展过程划分，集成电路的设计手段经历了手工设计、计算机辅助设计（ICCAD）、电子设计自动化(EDA)、电子系统设计自动化(ESDA)以及用户现场可编程器(FPGA)等阶段。

7.1.2 ASIC设计方法

- 就ASIC设计方法而言，集成电路的设计方法可分为全定制、半定制和可编程ASIC设计三种方式。

1. 全定制设计

- 全定制ASIC是利用集成电路的最基本设计方法，对集成电路中所有的元器件进行精工细作（要考虑工艺条件，根据电路的复杂和难度决定器件工艺类型、布线层数、材料参数、工艺方法、极限参数、成品率等因素）的设计方法。全定制设计可以实现最小面积，最佳布线布局、最优功耗速度积，得到最好的电特性。该方法尤其适宜于模拟电路，数模混合电路以及对速度、功耗、管芯面积、其它器件特性（如线性度、对称性、电流容量、耐压等）有特殊要求的场合；或者在没有现成元件库的场合。

全定制设计的主要特点

- 全定制设计的主要特点如下：
  - 需要丰富的经验和特殊的技巧，需要掌握各种微电子电路的设计规则和方法,一般由专业微电子IC设计人员完成；
  - 常规设计可以借鉴以往的设计，部分器件需要根据电特性单独设计；
  - 布局、布线、排版组合等均需要反覆斟酌调整，按最佳尺寸、最合理布局、最短连线、最便捷引脚等设计原则设计版图。
  - 版图设计与工艺相关，要充分了解工艺规范，根据工艺参数和工艺要求合理设计版图和工艺；
  - 设计要求高、周期长，设计成本昂贵。

2. 半定制设计

- 半定制设计方法又分成基于标准单元的设计方法和基于门阵列的设计方法。半定制主要适合于开发周期短，低开发成本、投资、风险小的小批量数字电路设计。
- 基于标准单元的设计方法是：将预先设计好的称为标准单元的逻辑单元，如门电路，多路开关，触发器，时钟发生器等，按照某种特定的规则排列，与预先设计好的大型单元一起，根据电路功能和要求用掩膜版将所需的逻辑单元连接成所需的专用集成电路。

半定制设计

- 基于标准单元设计方法的主要特点
  - 用预先设计、预先测试、预定特性的标准单元库，省时、省钱、少风险；
  - 设计人员只需确定标准单元的布局以及单元的互连；
  - 所有掩膜层是定制的；
  - 制造周期较短，开发成本不是太高；
  - 需要花钱购买或自己设计标准单元库；
  - 要花较多的时间进行掩膜层的互连设计。

半定制设计

- 基于门阵列的设计方法是在预先制定的具有晶体管阵列的基片或母片上，根据电路功能和要求通过掩膜互连的方法完成专用集成电路设计。
- 用门阵列设计的ASIC中，只有上面几层用作晶体管互连的金属层由设计人员用全定制掩膜方法确定，这类门阵列称为掩膜式门阵列MGA（Masked Gate Array）。
- 门阵列中的逻辑单元称为宏单元，其中每个逻辑单元的基本单元版图相同，只有单元内以及单元之间的互连是定制的。客户设计人员可以从门阵列单元库中选择预先设计和预定特性逻辑单元或宏单元，进行定制的互连设计。

半定制设计

- 基于门阵列的设计方法的主要特点：
  - 适合于开发周期短，低开发成本的小批量数字电路设计；
  - 门阵列基本单元固定，不便于实现存储器之类的电路；
  - 在内嵌式门阵列中，留出一些IC区域专门用于实现特殊功能，如设计存储器模块或其它功能电路模块。

3. 可编程ASIC设计

- 可编程ASIC器件分为可编程逻辑器件（PLD）和现场可编程门阵列（FPGA）两类。
- 目前常用的可编程逻辑器件类型有：通用阵列逻辑（GAL）和复杂的可编程逻辑器件（CPLD）。
- 可编程逻辑器件的特点有：
  - 无定制掩膜层或逻辑单元；
  - 设计周期短；
  - 单独的大块可编程互连；
  - 具有可编程阵列逻辑，触发器或锁存器组成的逻辑宏单元矩阵。

可编程ASIC设计

- 现场可编程门阵列（FPGA）具有现场可编程特性，一般地讲，现场可编程门阵列比可编程逻辑器件规模更大、更复杂。
- 现场可编程门阵列的主要特点有：
  - 无定制掩膜层；
  - 基本逻辑单元和互连采用编程的方法实现；
  - 核心电路是规则的可编程基本逻辑单元阵列，可以实现组合逻辑和时序逻辑；
  - 设计周期很短。

7.2 GAL器件的编程及应用

- GAL逻辑器件具有如下特点：
  - 采用高速电可擦CMOS工艺制造的，CMOS的低功耗特性；
  - 可多次擦除和编程，适合于学习和样机研制；
  - 器件速度快，不低于任何其他TTL可编程逻辑芯片的速度。
- 在本书中，我们将介绍两种开发语言FASTMAP和ABEL，并使用这两种语言来编写GAL器件的代码。

7.2.1 FASTMAP语言及其应用举例

1. FASTMAP软件及语言规则

- 汇编软件FASTMAP的作用是将用户输入的布尔表达式翻译成标准JEDEC码，并产生列表文件和熔断图文件。
- FASTMAP支持如下布尔表达式：
  - （1）<SYMBOL>＝<逻辑表达式>，如Y＝A＋B。表示输出直接由输入表达式决定，与时钟无关，并且输出不允许高阻态（三态）。
  - （2）<SYMBOL>：＝<逻辑表达式>，如Y：＝A＋B。表示输出的状态只有在时钟的上升沿才改变状态，时钟输入脚不能作为信号输入脚。
  - （3）<SYMBOL>.OE=<逻辑表达式>，如Y.OE＝A·B。表示在逻辑表达式为真时，输出有效；而其它状态时，输出为高阻态。

FASTMAP软件及语言规则

- FASTMAP支持的逻辑关系符有：
- “*”——表示“与”的关系；
- “＋”——表示“或”的关系；
- “/”——表示“非”的关系。
- 符号“；”表示注释，在一行中，“；”后面的所有ASCII码仅仅写入列表文件。
- “DESCRIPTION”表示文件结束，“DESCRIPTION”后的所有内容，编译器都不予理会，仅仅写入列表文件中。

FASTMAP软件及语言规则

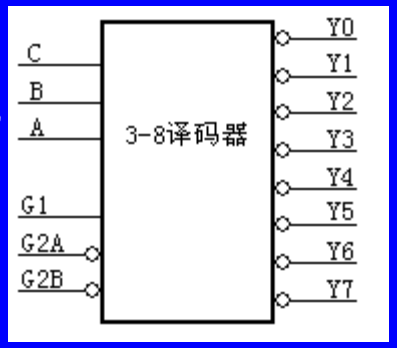
- 汇编的源文件是一个标准的ASCII码文件，任何文本编辑器，如记事本都可以编写，FASTMAP对于源文件书写格式有一定的要求。
- 源文件的第一行要求给出GAL器件型号，“GAL16V8”或“GAL20V8”；
- 第二行可给出GAL器件的逻辑名称，如“4 Bits Counder”、“Address Decoder”等；
- 第三行可列出版本信息和日期；
- 第四行可写明设计者的姓名。
- 这四行信息不能省略，否则不能通过编译。

FASTMAP软件及语言规则

- 接下来就是定义器件的管脚名称，必须从第五行开始。管脚名称可以使用不包含前面提到的逻辑关系符的任何符号或字符串，定义时只需将管脚的名称按管脚的序号依次排列即可，可以有多行。如果写在同一行里，两个管脚名之间应该用空格分开。
- 管脚定义完成之后就可以书写布尔表达式，布尔表达式中只能用FASTMAP所支持的三种关系符，第一个布尔表达式最多有8个“或”项（“＋”项），因此超过8个或项的逻辑关系需要用户手工化简，并且FASTMAP不支持括号，即“（）”。
- 当所有布尔表达式书写完后，应以一个关键字“DESCRIPTION”结束，并适当追加注释，以提高汇编代码的可读性。

2. FASTMAP应用举例

- 〔例1〕3-8译码器
  - 3-8译码器是一种很常用的译码器，一般用于地址译码，如I/O端口选择等。它有3个二进制码输入端，8个与输入二进制码相对应的输出端，以及控制端/G1、/G2A和/G2B，器件引脚如图7.2.1所示。由于只有6个输入和8个输出，因此用一片GAL16V8即可。

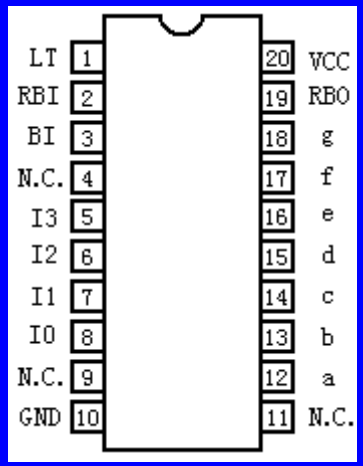


3-8译码器的FASTMAP代码

```
GAL16V8          ;DEVICE
Address Decoder
V1.0 2006.11.2
DESIGNED BY XXX
A  B  C  NC  G2A  G2B  G1  NC  NC  GND  ;PIN NAME
NC  Y7  Y6  Y5  Y4  Y3  Y2  Y1  Y0  VCC

Y0＝/A*/B*/C*G1*/G2A*/G2B
Y1＝A*/B*/C*G1*/G2A*/G2B
Y2＝/A*B*/C*G1*/G2A*/G2B
Y3＝A*B*/C*G1*/G2A*/G2B
Y4＝/A*/B*C*G1*/G2A*/G2B
Y5＝A*/B*C*G1*/G2A*/G2B
Y6＝/A*B*C*G1*/G2A*/G2B
Y7＝A*B*C*G1*/G2A*/G2B
DESCRIPTION
```

〔例2〕七段显示译码器

- 数字显示译码器是用来驱动数码管的中规模集成电路。它的功能是将BCD码变成十进制数字，并在数码管上显示出来。下面的以七段显示译码器为例讲解FASTMAP的使用。
- 七段显示译码器的输入由两部分组成，数码输入部分（I3、I2、I1、I0）和辅助输入部分（LT，RBI，BI），输出除了七位译码输出外，还有灭零输出信号RBO。



七段显示译码器

- LT为试灯信号输入端，用来检查数码管的七段是否都正常。由表中第一行可见，当LT=0，BI=1，即输入试灯信号时，不管I3，I2，I1，I0输入如何，数据管的七段全亮，说明各段显示正常，否则就不正常。BI为熄灭信号输入端，由表中第二行可见，当BI=0时，不管其它输入端状态如何，数码管熄灭。RBI为灭零信号输入端，用来熄灭不需要显示的0。由表中第三行可见，当LT=1，RBI=0，BI=1，且输入I3I2I1I0=0000时，数码管熄灭。这在多位数显系统中，可将有效数字前、后多余的0熄灭，既便于读数，又可减少功耗。当LT=1，RBI=0时输出端RBO=0。级联时，可将RBO送到另一片七段显示译码器的灭零信号输入端，可以使这

七段显示译码器的FASTMAP代码

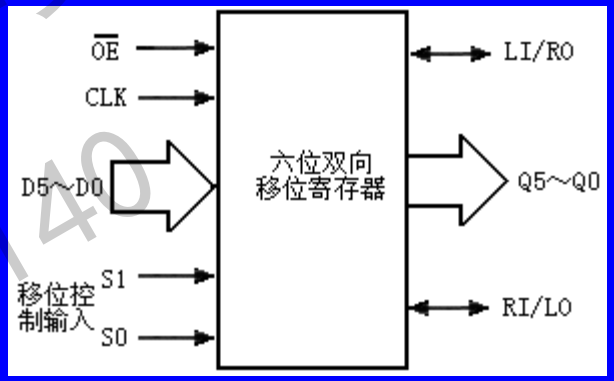
```
GAL16V8
DIGITAL  DISPLAY  DECODER
VER 1.0  2006-11.2
DESIGNED BY XXX
NULL  NULL  LT  RBI  BI  I3  I2  I1  I0
  GND
NULL  A  B  C  D  E  F  G  RBO  VCC

RBO=/LT+RBI+/BI+I3+I2+I1+I0
G=/BI+LT*BI*/I3*/I2*/I1*/I0+LT*BI*/I3*/I2*
  /I1*I0+LT*BI*/I3*I2*I1*I0
```

七段显示译码器的FASTMAP代码

```
F=/BI+LT*BI*/RBI*/I3*/I2*/I1*/I0+LT*BI*/I3*/I2*/I1*I0
   +LT*BI*/I3*/I2*I1 +LT*BI*/I3*I2*I1*I0

E=/BI+LT*BI   */RBI*/I3*/I2*/I1*/I0   +LT*BI   */I3*/I2*I0   +LT*BI
   */I3*I2*/I1 +LT*BI */I3*I2*I1*I0+LT*BI *I3*/I2*/I1*I0

D=/BI+LT*BI   */RBI*/I3*/I2*/I1*/I0+LT*BI   */I3*/I2*/I1*/I0   +LT*BI
   */I3*I2*/I1*/I0+LT*BI */I3*I2*I1*I0

C=/BI+LT*BI */RBI*/I3*/I2*/I1*/I0+LT*BI */I3*/I2*I1*/I0

B=/BI+LT*BI   */RBI*/I3*/I2*/I1*/I0   +LT*BI   */I3*I2*/I1*I0   +LT*BI
   */I3*I2*I1*/I0

A=/BI+LT*BI */RBI*/I3*/I2*/I1*/I0   +LT*BI */I3*/I2*/I1*I0 +LT*BI
   */I3*I2*/I1*/I0
DESCRIPTION
```

〔例3〕六位双向移位寄存器

- 移位寄存器的功能是在移位脉冲作用下，寄存器中的数据可以逐位左移或右移。双向移位寄存器可根据移位控制端的输入来选择工作方式，以完成左移或右移的功能。移位寄存器的应用很广，一般可用来对数码作串、并行变换，或在移位寄存器的基础上构成计数器和序列信号发生器等。



六位双向移位寄存器

- 图7.2.4所示的六位双向移位寄存器是采用GAL16V8设计的，可以将数据左右移位，可以串行或并行装入数据，用S0和S1两个输入端选择工作方式;所有串行口全部是双向的，因此引脚12（RI/LO）是右串行输入和左串行输出，而引脚19（LI/RO）是左串行输入和右串行输出 。

六位双向移位寄存器的FASTMAP代码

```
GAL16V8
6 BIT SHIFTER
VER1.0 2006-11-2
DESIGNED BY XXX
CLK   S1   S0   D5   D4   D3   D2   D1   D0   GND
OE   RILO   Q5   Q4   Q3   Q2   Q1   Q0   LIRO   VCC

Q0:=/S1*/S0*Q0+/S1*S0*Q1+S1*/S0*LIRO+S1*S0*D0
Q1:=/S1*/S0*Q1+/S1*S0*Q2+S1*/S0*Q2+S1*S0*D1
Q2:=/S1*/S0*Q2+/S1*S0*Q3+S1*/S0*Q1+S1*S0*D2
```

六位双向移位寄存器的FASTMAP代码

```
Q3:=/S1*/S0*Q3+/S1*S0*Q4+S1*/S0*Q2+S1*S0*D3
Q4:=/S1*/S0*Q4+/S1*S0*Q5+S1*/S0*Q3+S1*S0*D4

Q5:=/S1*/S0*A5+/S1*S0*RIL0+S1*/S0*Q4+S1*S0*D5

LIR0=Q0
LIR0.OE=S0*/S1

RIL0=Q5
RIL0.OE=/S0*S1
DESCRIPTION
```

〔例4〕四位同步可逆计数器

- 可逆计数器指的是既能做加法计数又能做减法计数的计数器。在实际应用中，计数不一定要从0开始，而是从某一指定数开始计数，因此要求计数器具有预置数的功能。
- 决定计数器工作方式的是两个控制输入CTL0和CTL1，其方式选择如表7.2.4所述。

CNTL1	CNTLO	功　能
0	0	清除
0	1	送数
1	0	减法计数
1	1	加法计数

四位同步可逆计数器的FASTMAP代码

```
GAL16V8
4 BIT ADD/SUB COUNTER
VER1.0 2006-11-2
DESIGNED BY XXX
CLK   I3   I2   I1   I0   NULL   NULL     CTL1   CTL0   GND
OE   NULL   NULL   NULL   NULL   Q3   Q2   Q1   Q0   VCC

Q3:=/CNTL1*CNTL0*I3+CNTL1*/CNTL0*/Q0*/Q1*/Q2*/Q3
   +CNTL1*/CNTL0*Q0*Q1*Q2*Q3+CNTL1*CNTL0*/Q0*/Q1*/Q2*Q3
    +CNTL1*CNTL0*Q0*Q1*Q2*/Q3+CNTL1 *Q0*/Q2*Q3
    +CNTL1 */Q1*Q2*Q3+CNTL1 */Q0*Q1*Q3
```

四位同步可逆计数器的FASTMAP代码

```
Q2：=/CNTL1*CNTL0*I2+CNTL1*/CNTL0*/Q0*/Q1*/Q2
    +CNTL1*/CNTL0*Q1*Q2+CNTL1*CNTL0*/Q1*Q2
    +CNTL1*CNTL0*Q0*Q1*/Q2+CNTL1*/ Q0*Q1*Q2
    +CNTL1*Q0*/Q1*Q2
Q1：=/CNTL1*CNTL0*I1+CNTL1*/CNTL0*/Q0*/Q1
     +CNTL1*/CNTL0*Q0*Q1+CNTL1*CNTL0*/Q0*Q1
     +CNTL1*CNTL0*Q0*/Q1

Q0：=/CNTL1*CNTL0*I0+CNTL1*/Q0
DESCRIPTION
```

7.2.2 ABEL语言及其应用举例

- 1. ABEL语言及其编程介绍
  - ABEL语言是由美国DATA I/O公司于1983～1988年推出的一种硬件描述语言(也称为ABEL-HDL)，是开发SPLD的一种高级代码设计语言。ABEL语言支持逻辑方程、真值表和状态图三种逻辑描述方式。

1. ABEL语言及其编程介绍

❖ABEL源文件的基本结构 ：

```
module  模块名;
[title  ’标题说明’ ]
Declarations
[器件名  device  ’器件的工业标号’ ;]
信号名, 信号名.....pin [引脚号, 引脚号]  [istype  ’属性’ ];
信号名, 信号名.....node [istype ’属性’ ];
[常量说明语句;]
[集合的定义;]
[宏定义语句;]

[equations 逻辑方程;]
[truth-table(输入变量 -> 输出变量) 真值表;]
[state-diagram(状态变量) 状态图描述;]
[test-vectors(输入变量 -> 输出变量) 测试向量表;]
end  模块名;
```

说明部分

逻辑关系描述部分
(三者取一或其任意组合)

ABEL语言及其编程介绍

- 源文件中的信号‘属性’可选择表7.2.5所述的属性字符串。

属性字符串	含义
‘com’	组合型输出
‘reg’	寄存器型输出
‘invert’	目标器件中的反相器
‘buffer’	目标器件中的缓冲器
‘neg’	未规定的逻辑为‘1’
‘pos’	未规定的逻辑为‘0’
‘keep’	不将此信号从方程式中简化掉
‘reg_d’	D型触发器
‘reg_jk’	JK型触发器
‘reg_sr’	SR型触发器
‘reg_t’	T型触发器

2. 应用举例

- 〔例1〕使用ABEL语言设计一个四位二进制数比较器

```
module  COMP4
declarations                    //变量说明段
    A3, A2, A1, A0     pin;
    B3, B2, B1, B0     pin;
    G, L, E    pin istype 'com' ;
    A = [A3..A0];  B = [B3..B0];
Equations                       //逻辑方程段
        G = ( A>B )   ;
        L = ( A<B )   ;
        E = ( A=B )   ;
test_vectors ( [ A, B ]->[ G, L, E ])   //测试向量段
                [ 0, 0 ] -> [ 1, 0, 0];
                [ 0, 1 ] -> [ 0, 1, 0];
                [ 15,10 ] -> [ 0, 0, 1];
end  COMP4
```

应用举例

- 该例子中的module，declarations，pin，istype，Equations，test_vectors，end是关键字。使用时不分大小写。不能用于器件名、引脚、节点、常量、组、宏和信号。ABEL语言中的关键字如下：
  - async_reset fuses state case goto state_diagram cycle if state_register declarations in sync_reset device interface test_vectors else istype then enable(obsolete) library title end macro trace endcase module truth_table endwith node wait equations options when external pin with flag(obsolete) property functional_block
- 注释有两种方法：一种是使用双引号“ ”，另一种是使用双斜杠//。

〔例2〕使用ABEL语言设计一个4位左移移位寄存器

- ABEL代码如下：

```
MODULE SHIFT4
TITLE 'SHIFT4'
DIN,CLK,CLR  PIN;
Q3..Q0  PIN ISTYPE 'REG';
OUT=[Q3..Q0];

EQUATIONS
OUT.CLK=CLK;
OUT.AR=CLR;
Q0 := DIN;
Q1 := Q0;
Q2 := Q1;
Q3 := Q2;
END
```

- 方程式中的赋值符为“＝”或“:＝”。“＝”为非时钟赋值，主要用于组合逻辑输出；
- “:＝”为时钟赋值，主要用于寄存器输出。

〔例3〕使用WHEN_THEN语句设计一个4选1数据选择器。

- ABEL代码如下：

```
MODULE      MUX4S1
TITLE    'MUX4S1'
S1,S0            PIN;
D3,D2,D1,D0      PIN;
Y    PIN   ISTYPE   'COM';
S=[S1,S0];
EQUATIONS
WHEN (S==0) THEN Y=D0;
WHEN (S==1) THEN Y=D1;
WHEN (S==2) THEN Y=D2;
WHEN (S==3) THEN Y=D3;
END
```

- 在方程中允许使用WHEN-THEN-ELSE语句。但不能使用IF-THEN-ELSE语句。其表达式如下：

[WHEN 条件 THEN][!]变量名＝表达式;

[ELSE 方程式;]

或者

[WHEN 条件 THEN]方程式;

[ELSE 方程式;]

〔例4〕使用WHEN_THEN_ELSE语句设计一位十进制加法计数器

- ABEL代码如下：

```
MODULE  count10
q3,q2,q1,q0      PIN ISTYPE 'reg';
clk,cd                  PIN;
count = [q3..q0];
EQUATIONS
count.clk = clk;
count.ar = cd;
WHEN (count==9) THEN count := 0;
          ELSE count := (count.fb+1);
END
```

ABEL代码说明

- 真值表语句用表格的形式将输出定义为输入的函数。以关键字TRUTH_TABLE为开始，后面跟表头向量和表格。其格式如下：

TRUTH_TABLE（输入 —> 输出）
输入 —> 输出；

或

TRUTH_TABLE（输入［ :> 寄存器型输出］ —> 输出；）
输入［ :> 寄存器型输出］ —> 输出；

〔例5〕用真值表实现一个4-2线编码器

- ABEL代码如下：

```
MODULE   CODER24                          //模块语句
TITLE   'CODER24'                         //标题语句
I3,I2,I1,I0   PIN;                        //定义段
O1,O0   PIN   ISTYPE   'COM';             //组合型输出
TRUTH_TABLE ([I3,I2,I1,I0]->[O1,O0])      //用真值表实现
            [0,0,0,1]->[0,0];
            [0,0,1,0]->[0,1];
            [0,1,0,0]->[1,0];
            [1,0,0,0]->[1,1];
END
```

〔例6〕用真值表实现一个2-4线译码器

- 2-4线译码器的输入输出关系如表7.2.6所示

输入		输出			
a	b	y0	y1	y2	y3
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1

2-4线译码器

- ABEL代码如下：

```
MODULE  decoder
DECLARATIONS
        a ,  b              pin;
        y0 , y1 , y2 , y3     pin                  istype
  'com' ;
TRUTH_TABLE    ( [ a, b] ->  [y0,y1,y2,y3] )
                [ 0,  0] -> [ 1 , 0 , 0 , 0];
                [ 0,  1] -> [ 0 , 1 , 0 , 0];
                [ 1,  0] -> [ 0 , 0 , 1 , 0];
                   [ 1,  1] -> [ 0 , 0 , 0 , 1];
END  decoder
```

〔例7〕设计一位十进制加法计数器及七段译码电路

- ABEL代码如下：

```
MODULE    count
clk           PIN;
q3..q0    PIN   ISTYPE ’REG’;
a,b,c,d,e,f,g   PIN   ISTYPE ’COM’;
H, L = 1,0;
count = [q3..q0];

EQUATIONS
[q3..q0].clk=clk;
```

一位十进制加法计数器及七段译码电路

```
TRUTH_TABLE
([count]:>[count]->[a, b, c, d, e, f, g])
[0]:>[1]->[1, 1, 1, 1, 1, 1, 0];
[1]:>[2]->[0, 1, 1, 0, 0, 0, 0];
[2]:>[3]->[1, 1, 0, 1, 1, 0, 1];
[3]:>[4]->[1, 1, 1, 1, 0, 0, 1];
[4]:>[5]->[0, 1, 1, 0, 0, 1, 1];
[5]:>[6]->[1, 0, 1, 1, 0, 1, 1];
[6]:>[7]->[0, 0, 1, 1, 1, 1, 1];
[7]:>[8]->[1, 1, 1, 0, 0, 0, 0];
[8]:>[9]->[1, 1, 1, 1, 1, 1, 1];
[9]:>[0]->[1, 1, 1, 0, 0, 1, 1];
END
```

ABEL代码说明

- 状态图语句的格式如下：

STATE_DIAGRAM 状态寄存器 [—>状态输出]
[STATE 状态表达式：[方程式]
[方程式]
状态转移描述]

- ABEL语言提供三种语句来描述状态转移 ：
  - 无条件转移语句（GOTO语句）；
  - 条件转移语句（IF-THEN-ELSE语句）；
  - 多路转移语句（CASE-ENDCASE）。
- 此外，状态机在发生状态转移的同时可能有相应的数据输出，所以还有条件输出语句（WITH语句）。

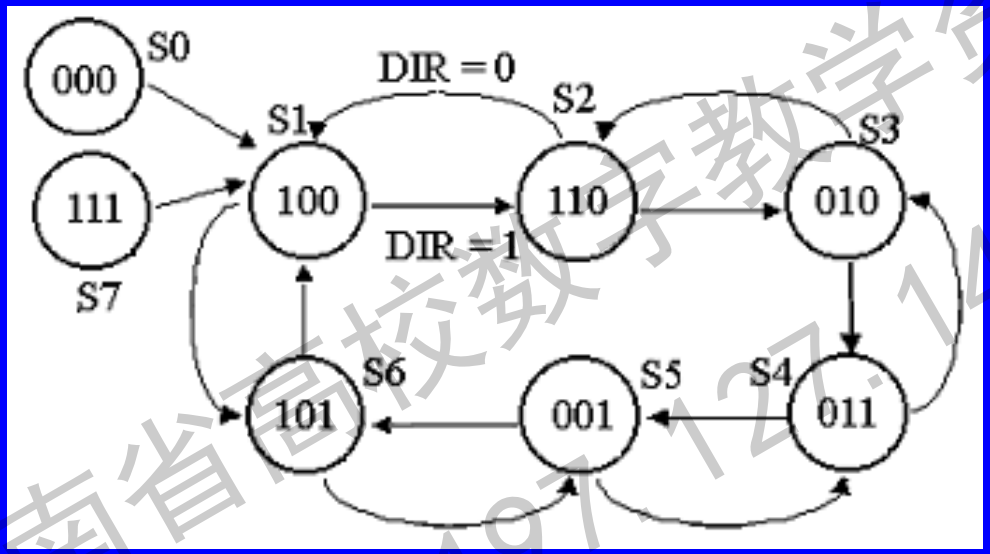
〔例8〕使用ABEL语言设计一个模为4的同步二进制递增计数器

- ABEL代码如下：

```
MODULE  cnt4
DECLARATIONS
        cp      pin;
        q1,q0   pin   istype 'reg' ;
         Co     pin   istype 'com' ;
         s0=^b00;        s2=^b10;
         s1=^b01;        s3=^b11;
EQUATIONS
        [q1,q0].clk = cp ;
STATE_DIAGRAM  [q1,q0]
            STATE  s0:  GOTO  s1  WITH  Co = 0;
            STATE  s1:  GOTO  s2  WITH  Co = 0;
            STATE  s2:  GOTO  s3  WITH  Co = 0;
            STATE  s3:  GOTO  s0  WITH  Co = 1;
END
```

〔例9〕使用ABEL语言步进电机三相六拍脉冲分配器电路

- 其三状态转换图如图7.2.7所示。



三相六拍脉冲分配器状态转换

- ABEL代码如下：

```
module MOTOR
declarations
        CP, DIR    pin;
        C, B, A    pin  istype 'reg';
        CK,X,Z = .C. , .X. , .Z.;

       Q= [C,B,A];
        S0=^b000;        S4=^b011;
        S1=^b100;        S5=^b001;
        S2=^b110;        S6=^b101;
        S3=^b010;        S7=^b111;
```

三相六拍脉冲分配器状态转换

```
Mode = DIR;
        Up   = 1  ;
        Down = 0  ;
EQUATIONS
          Q.CLK=CP;
STATE_DIAGRAM  Q
          State S0:  goto S1;
          State S1:  case (Mode = = Up)    : S2;
                          (Mode = = Down)  : S6;
                           endcase;
          State S2:  case (Mode = = Up)    : S3;
                          (Mode == Down)   : S1;
                            endcase;
```

三相六拍脉冲分配器状态转换

```
State S3:  case (Mode = = Up)    : S4;
                                 (Mode = = Down) : S2;
                                  endcase;
          State S4:  case (Mode = = Up)    : S5;
                                 (Mode = = Down) : S3;
endcase;
State S5:   case (Mode = = Up)    : S6;
                 (Mode = = Down) : S4;
                  endcase;
 State S6:  case (Mode = = Up)    : S1;
                 (Mode = = Down) : S5;
                  endcase;
State S7:   goto  S1;
end MOTOR
```

7.3 CPLD/FPGA器件的编程及应用

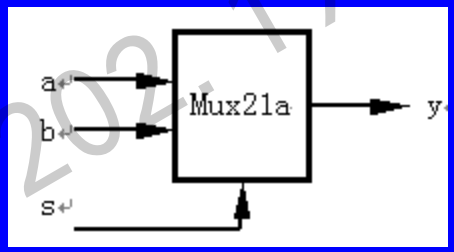
7.3.1 VHDL语言介绍

- 〔例1〕2选1多路选择器的VHDL语言描述

```
LIBRARY IEEE ;
USE IEEE.STD_LOGIC_1164.ALL ;
ENTITY mux21a IS
PORT ( a, b : IN  STD_LOGIC;
           s : IN  STD_LOGIC;
           y : OUT STD_LOGIC );
END ENTITY mux21a;
ARCHITECTURE one OF mux21a IS
  BEGIN
 y <= a  WHEN  s='0'  ELSE
           b  ;
END ARCHITECTURE one ;
```

2选1多路选择器的VHDL语言描述

- 该代码是2选1多路选择器的VHDL完整描述，即可以直接综合出实现相应功能的逻辑电路及其功能器件。
- 图7.3.1是此描述对应的逻辑图，图中a和b分别是两个数据输入端的端口名，s为通道选择控制信号输入端的端口名，y为输出端的端口名。“mux21a”是此器件的名称，这类似于“74HC138”、“CD4051”等器件的名称。



VHDL语言介绍

- 这里将对上述代码中出现的语言现象作出说明和归纳。
- 一般地，一个可综合的，完整的VHDL代码有比较固定的结构。该结构一般首先出现的是各类库及其代码包的使用声明，包括未以显式表达的工作库WORK库的使用声明，接着是实体描述，然后是结构体描述，而在结构体中可以含有不同的逻辑表达语句结构。

VHDL语言介绍

- 1. 设计库和标准代码包
  - 定义数据类型STD_LOGIC的函数包含于标准库IEEE的STD_LOGIC_1164标准代码包中。一般地，为了使用STD_LOGIC数据类型，应该在如例7.3.1的代码前加入如下两句说明语句：

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

  - 第一句中的LIBRARY是关键词，LIBRARY IEEE表示打开IEEE库；第二句的USE和ALL是关键词，USE IEEE.STD_LOGIC_1164.ALL表示允许使用IEEE库中STD_LOGIC_1164代码包中的所有内容，如类型定义、函数、过程、常量等。
  - 使用库和代码包的一般定义表式是：

```
LIBRARY  <设计库名> ;
USE <设计库名>.<代码包名>.ALL ;
```

VHDL语言介绍

- 2. 实体表达
  - VHDL完整的、可综合的代码结构，必须完整地表达出一片专用集成电路ASIC器件的端口结构和电路功能，无论是一片简单的门电路还是一片较复杂的CPU，都必须包含实体和结构体两个最基本的语言结构，这里将含有完整代码结构（包含实体和结构体）的VHDL表述称为设计实体。如前所述，实体描述的是电路器件的端口构成和信号属性，它的最简表式如下：

```
ENTITY e_name IS
        PORT ( p_name :  port_m   data_type;
                          . . .
               p_namei : port_mi   data_type );
END e_name;
```

  - 上式中ENTITY、IS、PORT和END ENTITY都是描述实体的关键词，在实体描述中必须包含这些关键词。关键词不分大写和小写。

VHDL语言介绍

- 3. 实体名
  - 实体表达式中的e_name是实体名，具体取名由设计者自定。由于实体名实际上表达的是该设计电路的器件名，所以最好根据相应电路的功能来确定，如4位二进制计数器，实体名可取为Counter4b；8位二进制加法器，实体名可取为Adder8b等等。需要特别注意的是，一般不用数字或中文定义实体名，不用与EDA工具库中已定义好的元件名作为实体名，如or2、latch等，也不用数字带头的实体名，如74LSX。

VHDL语言介绍

- 4. PORT语句和端口信号名
  - 描述电路的端口及其端口信号，必须用端口语句PORT（）引导，并在语句结尾处加分号“;”。实体表达式中的p_name是端口信号名，也由设计者自己确定，如例7.3.1中的端口信号名分别是a、b、s和y。
- 5. 端口模式
  - 实体表达式中的port_m表示端口模式，可综合的端口模式有4种，它们分别是“IN”、“OUT”、“INOUT”和“BUFFER”，用于定义端口上数据的流动方向和方式。

VHDL语言介绍

- 5. 端口模式
  - IN ：定义的通道为单向只读模式，规定数据只能通过此端口被读入实体中。
  - OUT ：定义的通道为单向输出模式，规定数据只能通过此端口从实体向外流出，或者说可以将实体中的数据向此端口赋值。
  - INOUT ：定义的通道确定为输入输出双向端口，即从端口的内部看，可以对此端口进行赋值，也可以通过此端口读入外部的数据信息；而从端口的外部看，信号既可以从此端口流出，也可以向此端口输入信号，如RAM的数据端口，单片机的I/O口。
  - BUFFER ：BUFFER的功能与INOUT类似，区别在于当需要输入数据时，只允许内部回读输出的信号，即反馈。如计数器的设计，可将计数器输出的计数信号回读，以作下一计数值的初值。与INOUT模式相比，BUFFER回读（输入）的信号不是由外部输入的，而是由内部产生，向外输出的信号。

VHDL语言介绍

- 6. 标准逻辑位数据类型STD_LOGIC
  - 例7.3.1中，2选1多路选择器的4个信号端口a、b、s和y的数据类型都被定义为STD_LOGIC。就数字系统设计来说，类型STD_LOGIC比BIT包含的内容丰富和完整的多。STD_LOGIC和BIT两种数据类型的代码包定义式如下（其中TYPE是数据类型定义语句）：
  - BIT数据类型定义：TYPE BIT IS（'0'，'1'）
  - STD_LOGIC数据类型定义：TYPE STD_LOGIC IS（'U'，'X'，'0'，'1'，'Z'，'W'，'L'，'H'，'-'）；

VHDL语言介绍

6. 标准逻辑位数据类型STD_LOGIC

- 以上定义的9种数据的含义是：‘U’——未初始化的；‘X’——强未知的；‘0’——强逻辑0；‘1’—— 强逻辑1；‘Z’——高阻态；‘W’——弱未知的；‘L’——弱逻辑0；‘H’——弱逻辑1；‘-’——忽略。它们较完整地概括了数字系统中所有可能的数据表现形式。因此STD_LOGIC比BIT具有更宽的取值范围，从而用STD_LOGIC描述的实际电路有更好的适应性。
- 在仿真和综合中，将信号或其它数据对象定义为STD_LOGIC数据类型是非常重要的，它可以使设计者精确地模拟一些未知的和具有高阻态的线路情况。对于综合器，高阻态’Z’和’-’忽略态（有的综合器对’X’）可用于三态的描述。但就目前的综合器而言，STD_LOGIC型数据能够在数字器件中实现的只有其中的四种值，即’X’(或’—’)、’0’、’1’和’Z’。

VHDL语言介绍

7. 数据类型BIT

- 实体表达式中的data_type是数据类型名。例7.3.1中，端口信号a、b、s和y的数据类型也可以定义为BIT。只是取值范围仅限于‘0’和‘1’两种。
- VHDL作为一种强类型语言，任何一种数据对象（信号、变量、常数）必须严格限定其取值范围，即对其传输或存储的数据类型作明确的界定。这对于大规模电路描述的排错是十分有益的。在VHDL中，预先定义好的数据类型有多种，如整数数据类型INTEGER、布尔数据类型BOOLEAN、标准逻辑位数据类型STD_LOGIC和位数据类型BIT等。

VHDL语言介绍

- 7. 数据类型BIT
  - BIT数据类型的信号规定的取值范围是逻辑位'1'和'0'。在VHDL中，逻辑位0和1的表达必须加单引号''，否则VHDL综合器将0和1解释为整数数据类型INTEGER。
  - BIT数据类型可以参与逻辑运算或算术运算，其结果仍是位的数据类型。VHDL综合器用一个二进制位表示BIT。BIT数据类型的定义或者说是解释包含在VHDL标准代码包STANDARD中，而代码包STANDARD包含于VHDL标准库STD中。标准库STD在VHDL代码中是默认打开的。所以一般不用如下语句去打开标准库STD和代码包STANDARD。

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

VHDL语言介绍

8. 结构体表述

结构体的一般表述如下：

```
ARCHITECTURE arch_name OF e_name IS
        （说明语句）
   BEGIN
（功能描述语句）
END arch_name ;
```

- 其中ARCHITECTURE、OF、IS、BEGIN和END都是描述结构体的关键词，在描述中必须包含，arch_name是结构体名。

VHDL语言介绍

- 8. 结构体表述
  - （说明语句）包括在结构体中需要说明和定义的数据对象、数据类型、元件调用声明等等。（说明语句）并非是必须的，（功能描述语句）则不同，结构体中必须给出相应的电路功能描述语句，可以是并行语句，顺序语句或它们的混合。

VHDL语言介绍

- 9. 信号传输（赋值）符号和数据比较符号
  - 例7.3.1中的表达式y<=a表示输入端口a的数据向输出端口y传输；但也可以解释为信号a向信号y赋值。在VHDL仿真中赋值操作y<=a并非立即发生的，而是要经历一个模拟器的最小分辨时间 δ 后，才将a的值赋予y 。在此不妨将 δ 看成是实际电路存在的固有延时量。VHDL要求赋值符“<=”两边的信号的数据类型必须一致。

VHDL语言介绍

- 9.信号传输（赋值）符号和数据比较符号
  - 例7.3.1中，条件判断语句WHEN_ELSE通过测定表式s＝’0’的比较结果，以确定由哪一端口向y赋值。式中的等号“=”没有赋值的含义，只是一种数据比较符号。其输出结果的数据类型是布尔数据类型BOOLEAN，它的取值分别是：true（真）和false（伪）。即当s为高电平’1’时，表式“s=’0’”输出“false”； 当s为低电平’0’时，表式“s =’0’”输出“true”。在VHDL综合器或仿真器中分别用’1’和’0’表达true和false。
  - 布尔数据不是数值，只能用于逻辑操作或条件判断。
  - 用于条件语句的判断表式可以是一个值，也可以是更复杂的逻辑或运算表达式，如：

```
IF a  THEN ..  -- 注意，a的数据类型必须是boolean
IF （s1=’0’）AND（s2=’1’）OR(c<b+1) THEN ..
```

VHDL语言介绍

- 10. WHEN_ELSE条件信号赋值语句
  - 例7.3.1中出现的是条件信号赋值语句，这是一种并行赋值语句，其表达方式如下：

赋值目标 <= 表达式 WHEN 赋值条件 ELSE
表达式 WHEN 赋值条件 ELSE
...
表达式 ;

VHDL语言介绍

- 10. WHEN_ELSE条件信号赋值语句
  - 在结构体中的条件信号赋值语句的功能与在进程（后面将详细介绍）中的IF 语句相同，在执行条件信号语句时，每一“赋值条件”是按书写的先后关系逐项测定的，一旦发现（赋值条件＝TRUE），立即将“表达式”的值赋给“赋值目标”信号。另外应注意，由于条件测试的顺序性，条件信号赋值语句中的第一子句具有最高赋值优先级，第二句其次，如此类推。例如在以下代码中，如果当p1和p2同时为’1’时，z获得的赋值是a而不可能是b 。

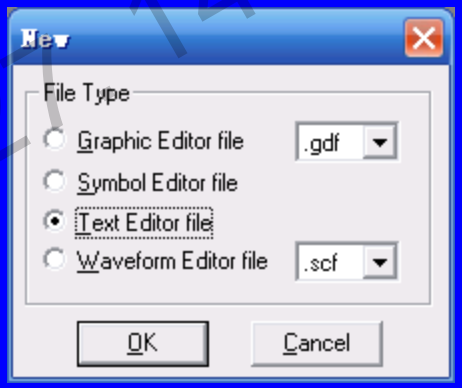
```
z  <= a WHEN p1 = ’1’ ELSE
      b WHEN p2 = ’1’ ELSE
      c  ;
```

VHDL语言介绍

- 11. 文件取名和存盘
  - 在文件存盘前，任一VHDL设计代码（代码）都必须给予一正确的文件名。一般地，文件名可以由设计者任意给定，但具体取名最好与文件实体名相同；文件后缀扩展名必须是“.VHD”，如 ADDER.VHD 。但考虑到某些EDA软件的限制和VHDL代码的特点，要求元件名与文件名必须是等同的，因此建议，VHDL代码的文件名最好与该代码的实体名一致。

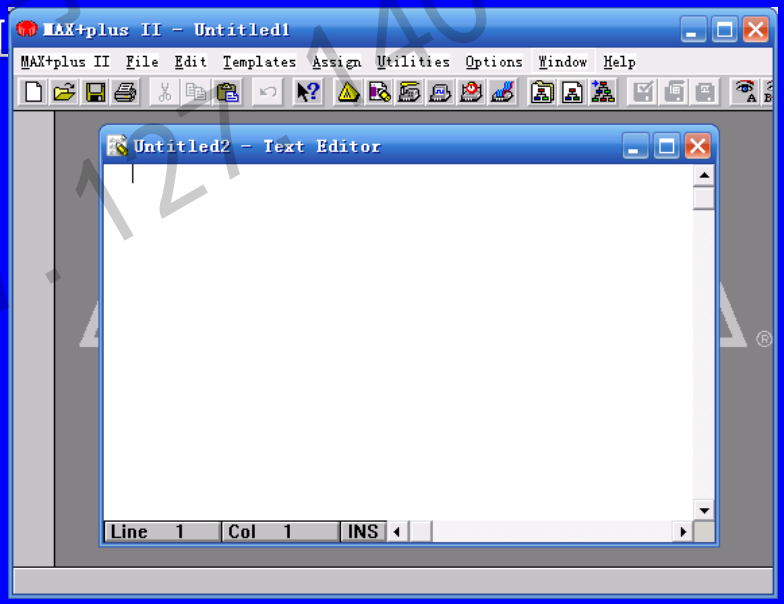
7.3.2 VHDL文本输入设计步骤

- 1. 文本编辑输入并存盘VHDL源文件
  - 打开MAX+plusII， 选择菜单“File-New...”，出现如图7.3.1所示的对话框，在框中选中“Text Editor file”，按“OK”按钮，即选中了文本编辑方式。



VHDL文本输入设计步骤

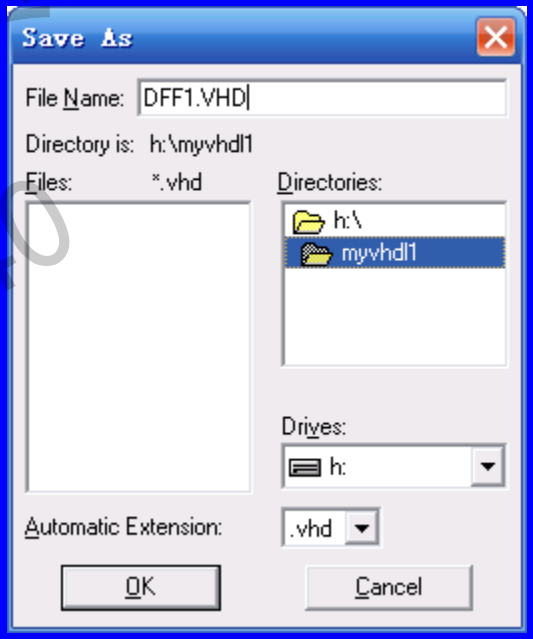
- 在出现的“Untitled2 - Text Editor”文本编辑窗(图7.3.2)中键入如例1所示的VHDL代码（D触发器），输入完毕后，选择菜单“File—Save”，即出现如图7.3.3所示的“Save As”对话框。首先在“Directories”目录框中选择自己已建立好的存放本文件的目录H:\MYVHDL1（用鼠标双击此目录，使其打开），然后在“File Name”框中键入文件名DFF1.VHD，按“OK”按钮，即把输入的文件放在目录H:\MYVHDL



VHDL文本输入设计步骤

- 〔例1〕 D触发器的VHDL描述。

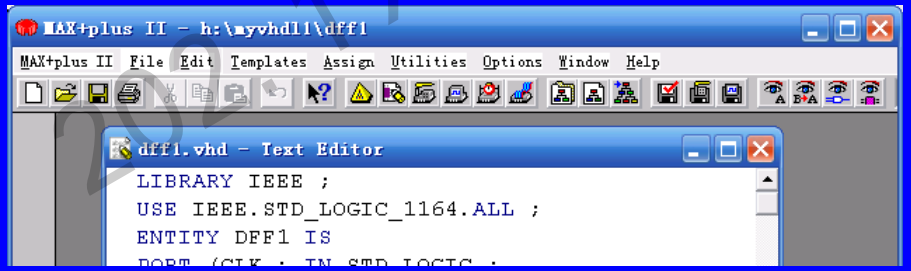
```
LIBRARY IEEE ;
USE IEEE.STD_LOGIC_1164.ALL ;
ENTITY DFF1 IS
PORT (CLK : IN STD_LOGIC ;
           D : IN STD_LOGIC ;
           Q : OUT STD_LOGIC );
 END ;
ARCHITECTURE bhv OF DFF1 IS
BEGIN
 PROCESS (CLK)
    BEGIN
       IF  CLK'EVENT AND CLK = '1'
            THEN  Q <= D ;
         END IF;
 END PROCESS ;
END bhv;
```



VHDL文本输入设计步骤

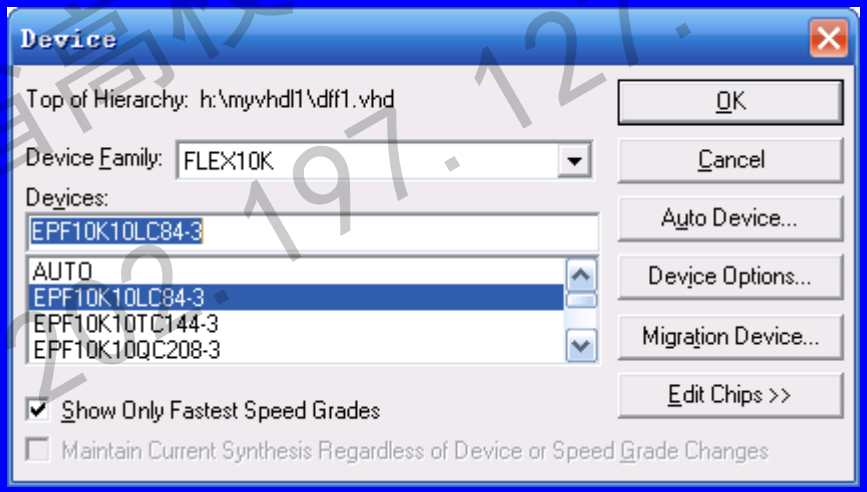
2. 将当前设计设定为工程

- 需要特别注意的是，在编译/综合DFF1.VHD之前，需要设置此文件为工程文件（Project）
- 选择菜单“File—Project—Set Project to Current File”，当前的设计工程即被指定为DFF1 。也可以通过选“File—Project — Name”，在跳出的“Project Name”窗中指定H:\MYVHDL1下的DFF1.VHD为当前的工程。设定后可以看见MAX+plusII主窗左上方的工程项目路径指向为：“h:\myvhdl1\dff1”。如图7.3.4所示。



VHDL文本输入设计步骤

- 在设定工程文件后，应该选择用于编程的目标器件：选择菜单“Assign－Device...”，在弹出的对话框中的“Device Family”下拉栏中，例如选择FLEX10K，然后在Devices列表框中选择芯片型号“EPF10K10LC84-3”，按OK。
- 在设定工程文件后，应该选择用于编程的目标芯片：选择菜单“Assign－Device...”，在弹出的对话框中的“Device Family”下拉栏中选择器件系列，例如选择FLEX10K，然后在“Devices”列表框中选择芯片型号“EPF10K10LC84-3”，按OK。如图7.3.5所示。

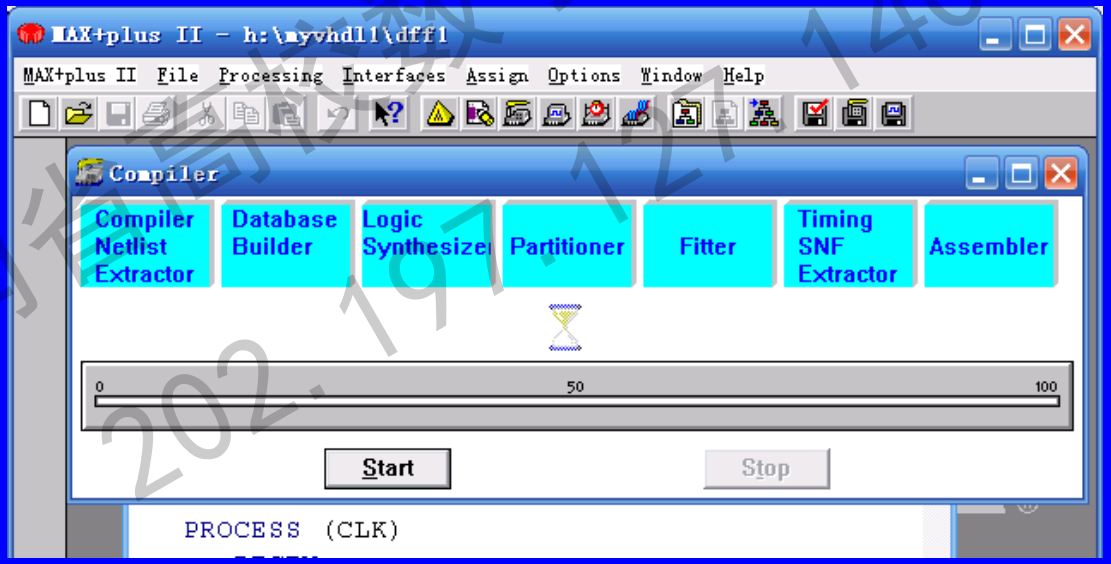


VHDL文本输入设计步骤

- 在设计中，设定某项VHDL设计为工程应该注意以下3方面的问题：
- （1）如果设计项目由多个VHDL文件组成，应先对各底层文件分别进行编辑、设置成工程、编译、综合、乃至仿真测试并存盘后以备后用。
- （2）最后将顶层文件（存在同一目录中）设置为工程，统一处理，这时顶层文件能根据例化语句自动调用底层设计文件。
- （3）在设定顶层文件为工程后，底层设计文件原来设定的元件型号和引脚锁定信息自动失效。元件型号的选定和引脚锁定情况始终以工程文件（顶层文件）的设定为准。同样，仿真结果也是针对工程文件的。所以在对最后的顶层文件处理时，仍然应该对它重新设定元件型号和引脚锁定（引脚锁定只有在最后硬件测试时才是必须的）。如果需要对特定的底层文件（元件）进行仿真，只能将某底层文件（元件）暂时设定为工程，进行功能测试或时序仿真。

VHDL文本输入设计步骤

- 3. 选择VHDL文本编译版本号和排错
  - 选菜单“MAX+plus II－Compiler”菜单，出现编译窗（图7.3.6）后，需要根据自己输入的VHDL文本格式选择VHDL文本编译版本号。



VHDL文本输入设计步骤

- 选择如图7.3.6所示界面上方的“Interfaces－VHDL Netlist Reader Settings”，在弹出的窗口（如图7.3.7所示）中选“VHDL ’1987”或“VHDL ’1993”。这样，编译器将支持87或93版本的VHDL语言。由于综合器的VHDL’1993版本兼容VHDL’1987版本的表述，所以如果设计文件含有VHDL’1987或混合表述，都应该选择“VHDL’1993”项。选择完后，回到编译窗口，按“START”键，运行编译。
- 如果编译有错误，将会出现相应的提示。有时尽管只有1、2个小错，但却会出现大量的出错信息，确定错误所在的最好办法是找到最上一排错误信息指示，用鼠标点成黑色，然后点击如图7.3.8所示窗口左下方的“Locate”错误定位钮，就能发现在出现文本编译窗中闪动的光标附近找到错误所在。纠正后再次编译，直至排除所有错误。闪动的光标指示错误所在是相对的，有的错误比较复杂，很难用此定位。

VHDL文本输入设计步骤

- VHDL文本编辑中可能出现许多错误，如：
  - （1）错将设计文件存入了根目录，并将其设定成工程，由于没有了工作库，报错信息如下：

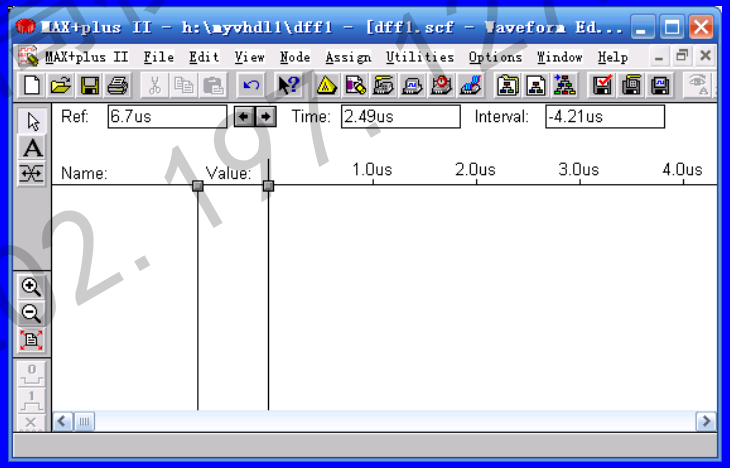
    Error ：Can't open VHDL "WORK"
  - （2）错将设计文件的后缀写成.tdf而非.vhd，在设定工程后编译时，报错信息如下：

    Error ：Line1，File h:\myvhdl1\dff1.tdf: TDF syntax error: ...
  - （3）未将设计文件名存为其实体名，如错写为muxa.vhd，设定工程编译时，报错信息如下：

    Error ： Line1 ， ...VHDL Design File "muxa.vhd" must contain ...

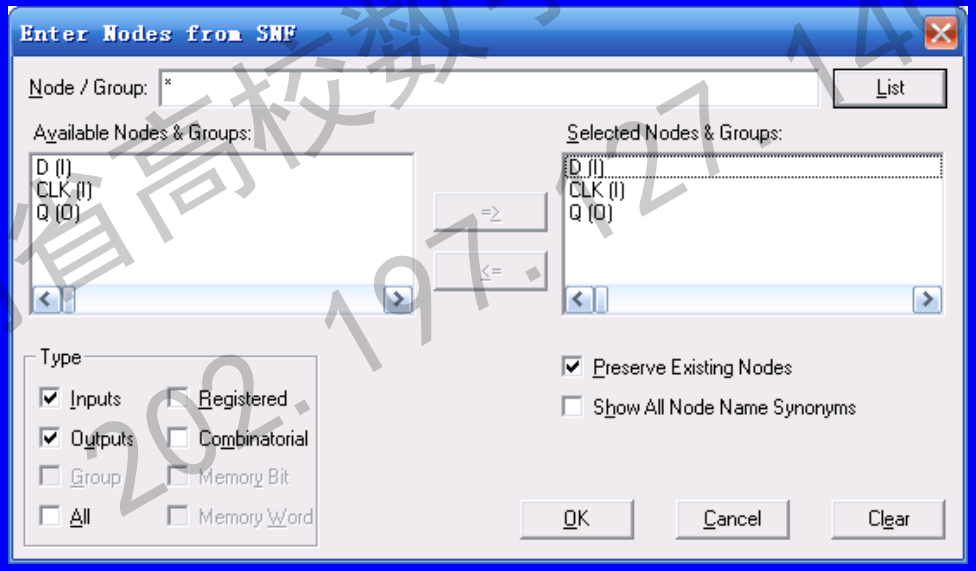
VHDL文本输入设计步骤

- 4. 时序仿真
- 首先选择菜单“File－New...”，打开图7.3.1所示的对话框，选择“Waveform Editor”，按“OK”按钮后进入仿真波形编辑窗。如图7.3.9所示。



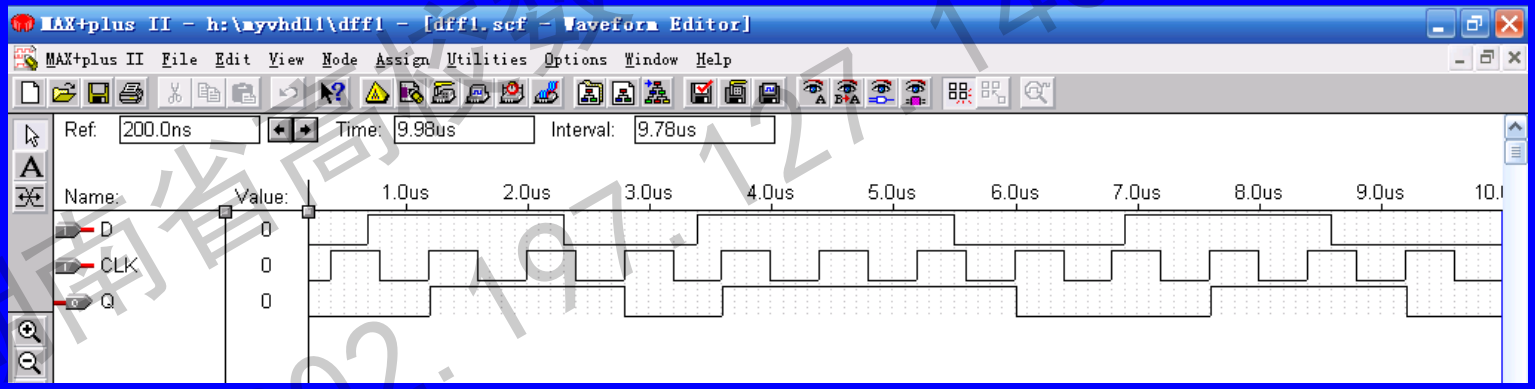
VHDL文本输入设计步骤

- 接下去选择菜单“Node－Enter Nodes from SNF”，进入仿真文件信号接点输入窗，按右上角“List”键后，将测试信号D(I)、CLK(I)和Q(O)输入仿真波形编辑窗。如图7.3.10所示。



VHDL文本输入设计步骤

- 选择Options项，将Snap to Grid的勾去掉；选择“File-End Time”，设定仿真时间区域，如设10us。给出输入信号后，选择MAX+plusII菜单Simulator进行仿真运算，波形如图7.3.11所示。



VHDL文本输入设计步骤

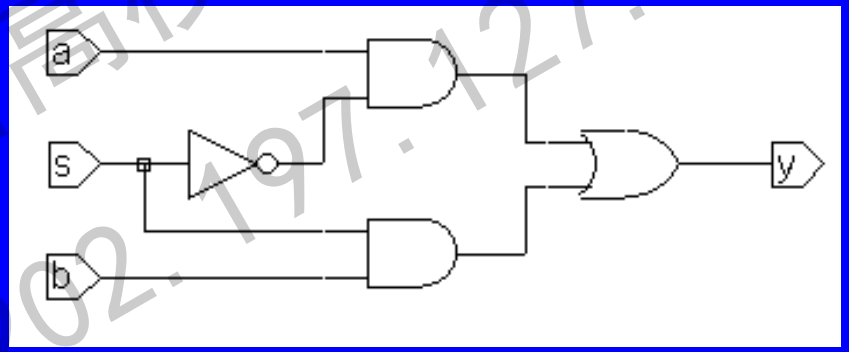
- 5. 硬件测试
  - 首先通过选择“MAX+plus II－Compiler”菜单，进入编辑窗，然后在“Assign”项中选“Pin / Location / Chip”选项，在跳出的窗口中的Node Name项中输入引脚clk，这时“Pin Type”项会出现“Input”指示字，表明 clk的引脚性质是输入，否则将不出现此字。此时在“PIN”项内输入引脚名为83脚，再点击右下方的Add项，此引脚即设定好了；以同样方法分别设引脚 d和q的引脚名，再点击上方的OK。关闭“Pin / Location / Chip”窗后，应点击编辑窗的“Start”，将引脚信息编辑进去。

VHDL文本输入设计步骤

- 接着编程下载。选“MAX+plus II”项中的“Programmer”项，跳出Programmer窗后，选Options项中的硬件设置项“Hardware Setup”，在此窗的下拉窗中选“ByteBlaster (MV)”项，点击OK即可。将实验板连接好，接好电源，点“Configure”，即可进行编程下载。然后就可以进行硬件测试。

7.3.3 VHDL文本输入设计举例

- 1.2选1选择器的VHDL语言的多种描述
  - 2选1选择器的实体如图7.3.1所示。在7.3.1小节我们使用“WHEN ELSE”语句实现了该器件的描述。其实每个电路或器件都可以使用多种描述方式得到相同的综合结果。这里我们将用另外的描述语句来描述2选1选择器。



2选1选择器的VHDL语言的多种描述

- 使用“IF THEN ELSE”顺序语句的VHDL代码如下：

```
ENTITY mux21a IS
PORT ( a, b : IN  BIT;
           s : IN  BIT;
           y : OUT BIT  );
END mux21a;
ARCHITECTURE one OF mux21a IS
 BEGIN
    PROCESS (a,b,s)
       BEGIN
        IF s = '0'  THEN    y <= a ;
                    ELSE    y <= b ;
       END IF;
     END PROCESS;
END one ;
```

2选1选择器的VHDL语言的多种描述

- 用结构体图（如图7.3.12所示）来描述2选1选择器的代码如下：

```
ENTITY mux21a IS
  PORT ( a, b : IN  BIT;
                s : IN  BIT;
               y : OUT BIT  );
 END mux21a;
 ARCHITECTURE one OF mux21a IS
 SIGNAL d,e :  BIT;
    BEGIN
       d <= a AND (NOT S) ;
       e <= b AND s ;
        y <= d OR e  ;
 END one ;
```

VHDL语言现象

- 这里将对上述代码中出现的语言现象作出说明和归纳。
  - （1）逻辑操作符AND、OR、NOT
    - AND、OR和NOT是逻辑操作符号。VHDL共有七种基本逻辑操作符，它们是AND(与)、OR(或)、NAND(与非)、NOR(或非)、XOR(异或)、XNOR（同或）和NOT(取反)。信号在这些操作符的作用下，可构成组合电路。逻辑操作符所要求的操作数(操作对象)的数据类型有三种，即BIT、BOOLEAN和STD_LOGIC。
    - 与其它硬件描述语言用符号表达逻辑操作符不同，VHDL中直接用对应的英语文字表达逻辑操作符号，这更明确显示了VHDL作为硬件行为描述语言的特征。

VHDL语言现象

- （2）IF_THEN条件语句
  - 利用IF_THEN_ELSE表达的VHDL顺序语句的方式，描述了同一多路选择器的电路行为。结构体中的IF语句的执行顺序类似于软件语言，首先判断如果s为低电平，则执行y<=a语句，否则（当s为高电平），则执行语句y<=b。由此可见VHDL的顺序语句同样能描述并行运行的组合电路。另外，IF语句必须以语句“END IF;”结束。

VHDL语言现象

- （3）PROCESS进程语句和顺序语句
  - 顺序语句“IF_THEN_ELSE_END IF;”是放在由“PROCESS”和“END PROCESS”引导的语句中的，由PROCESS引导的语句称为进程语句。在VHDL中，所有合法的顺序描述的语句都必须放在进程语句中（并非所有语句都能放在进程语句中）。
  - PROCESS旁的（a，b，s）称为进程的敏感信号表，通常要求将进程中所有的输入信号都放在敏感信号表中。例如，例中的输入信号是a，b和s，所以将它们全部列入敏感信号表中。由于PROCESS语句的执行依赖于敏感信号的变化，当某一敏感信号（如a）从原来的’1’跳变到’0’，或者从原来的’0’跳变到’1’时，就将启动此进程语句，而在执行一遍整个进程的顺序语句后，便进入等待状态，直到下一次敏感信号表中某一信号的跳变才再次进入“启动-运行”状态。在一个结构体中可以包含任意个进程语句，所有的进程语句都是并行语句，而由任一进程PROCESS引导的语句结构属于顺序语句。

2. 七段数码显示译码器设计

- 七段数码显示译码器是纯组合电路，通常的小规模专用IC，如74系列或4000系列的器件只能作十进制BCD码译码，然而数字系统中的数据处理和运算都是2进制的，所以输出表达都是16进制的，为了满足16进制数的译码显示，最方便的方法就是利用VHDL译码代码在FPGA或CPLD中实现。
- 作为七段BCD码译码器的设计，输出信号LED7S的7位分别接数码管的7个段，高位在左，低位在右。例如当LED7S输出为”1101101”时，数码管的7个段：gfedcba分别接1100110，接有高电平的段发亮，于是数码管显示“4”。

七段数码显示译码器设计

- 七段数码显示译码器的VHDL代码如下：

```
LIBRARY IEEE ;
USE IEEE.STD_LOGIC_1164.ALL ;
  ENTITY DecL7S IS
    PORT ( din  : IN  STD_LOGIC_VECTOR(3 DOWNTO 0) ;
           LED7S : OUT STD_LOGIC_VECTOR(6 DOWNTO 0)) ;
 END ;
ARCHITECTURE one OF DecL7S IS
 BEGIN
    PROCESS(din )
    BEGIN
        CASE  din(3 DOWNTO 0)  IS
```

七段数码显示译码器设计

```
        WHEN "0000" =>  LED7S <= "0111111" ; -- X"3F"→0
        WHEN "0001" =>  LED7S <= "0000110" ; -- X"06"→1
        WHEN "0010" =>  LED7S <= "1011011" ; -- X"5B"→2
        WHEN "0011" =>  LED7S <= "1001111" ; -- X"4F"→3
        WHEN "0100" =>  LED7S <= "1100110" ; -- X"66"→4
        WHEN "0101" =>  LED7S <= "1101101" ; -- X"6D"→5
        WHEN "0110" =>  LED7S <= "1111101" ; -- X"7D"→6
        WHEN "0111" =>  LED7S <= "0000111" ; -- X"07"→7
        WHEN "1000" =>  LED7S <= "1111111" ; -- X"7F"→8
        WHEN "1001" =>  LED7S <= "1101111" ; -- X"6F"→9
        WHEN "1010" =>  LED7S <= "1110111" ; -- X"77"→10
        WHEN "1011" =>  LED7S <= "1111100" ; -- X"7C"→11
        WHEN "1100" =>  LED7S <= "0111001" ; -- X"39"→12
        WHEN "1101" =>  LED7S <= "1011110" ; -- X"5E"→13
        WHEN "1110" =>  LED7S <= "1111001" ; -- X"79"→14
        WHEN "1111" =>  LED7S <= "1110001" ; -- X"71"→15
        WHEN OTHERS =>  NULL ;
    END CASE ;
END PROCESS ;

END ;
```

3. 数控分频器的设计

- 3. 数控分频器的设计
  - 数控分频器的功能是当在输入端给定不同输入数据时，将对输入的时钟信号有不同的分频比。本设计中的数控分频器是用计数值可并行预置的加法计数器设计完成的，方法是将计数溢出位与预置数加载输入信号相接即可。

数控分频器的设计

数控分频器的VHDL代码如下：

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;
ENTITY PULSE IS
    PORT ( CLK  : IN STD_LOGIC;
           D    : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
           FOUT : OUT STD_LOGIC  );
END;
ARCHITECTURE one OF PULSE IS
    SIGNAL FULL : STD_LOGIC;
BEGIN
  P_REG: PROCESS(CLK)
   VARIABLE CNT8 : STD_LOGIC_VECTOR(7 DOWNTO 0);
```

数控分频器的设计

```
BEGIN
   IF CLK'EVENT AND CLK = '1' THEN
         IF CNT8 = "11111111" THEN
           CNT8 := D;--当 CNT8 计数计满时，
                     --输入数据 D 被同步预置给计数器 CNT8
           FULL <= '1'; --同时使溢出标志信号 FULL 输出为高电平
             ELSE   CNT8 := CNT8 + 1;  --否则继续作加 1 计数
                      FULL <= '0'; --且输出溢出标志信号 FULL 为低电平
         END IF;
    END IF;
  END PROCESS P_REG ;
P_DIV: PROCESS(FULL)
  VARIABLE CNT2 : STD_LOGIC;
BEGIN
IF FULL'EVENT AND FULL = '1'
    THEN  CNT2 := NOT CNT2;--如果溢出标志信号 FULL 为高电平
                           --D 触发器输出取反
       IF CNT2 = '1' THEN  FOUT <= '1';
         ELSE   FOUT <= '0';
      END IF;
END IF;
  END PROCESS P_DIV ;
END;
```

7.4 SOPC简介

- 微电子技术的发展为SOC（片上系统）的实现提供了多种途径。对于经过验证而又具有批量的系统芯片，可以做成专用集成电路（ASIC）而大量生产。而对于一些仅为小批量应用或处于开发阶段的SOC，若马上投入流片生产，需要投入较多的资金和承担较大的试制风险。最近几年发展起来的SOPC（，System On Programmable Chip，即可编程的片上系统，或者说是基于大规模FPGA的单片系统）技术则提供了一种有效的解决方案，即用大规模可编程器件（如FPGA）来实现SOC的功能。

SOPC简介

- 可编程逻辑器件产生于20世纪70年代。其最初目的是为了用较少的PLD品种替代种类繁多的各式中小规模逻辑电路。在30多年的发展过程中，PLD的结构、工艺、功耗、逻辑规模和工作速度等都得到了重大的改进。尤其是在20世纪90年代出现的FPGA，单片的集成度由原来的数千门，发展到数十万甚至数百万门。芯片的I/O口也由数十个发展至上千个。有的FPGA制造商还推出了可植入CPU或DSP等IP Core的FPGA。因此，完全可能将一个电子系统集成到一片FPGA中，即SOPC，为SOC的实现提供了一种简单易行而又成本低廉的手段，极大地促进了SOC的发展。

SOPC简介

- SOPC技术是美国Altrea公司于2000年最早提出的，并同时推出了相应的开发软件Quartus II。SOPC是基于FPGA解决方案的SOC，与ASIC的SOC解决方案相比，SOPC系统及其开发技术具有更多的特色。

SOPC简介

- 构成SOPC的方案也有如下多种途径：
  - 1．基于FPGA嵌入IP硬核的SOPC系统

  即在FPGA中预先植入嵌入式系统处理器。
  - 2．基于FPGA嵌入IP软核的SOPC系统
  - 3．基于HardCopy技术的SOPC系统

小 结

- 1.现代ASIC设计技术是电子信息技术、计算机技术、半导体集成电路技术发展的结果。
- 2.ASIC设计方法可分为全定制、半定制和可编程ASIC设计三种方式。
- 3.GAL逻辑器件在中小规模电路设计中，仍然占有很重要的地位。本章分别介绍了FASTMAP和ABEL-HDL的使用。列举了几个使用FASTMAP设计典型例子：3-8译码器、七段显示译码器、六位双向移位寄存器和四位同步可逆计数器。列举了几个使用ABEL语言设计典型例子：四位二进制数比较器、4位左移移位寄存器、4选1数据选择器、十进制加法计数器、4-2线编码器、2-4线译码器、十进制加法计数器及七段译码电路、二进制递增计数器、步进电机三相六拍脉冲分配器电路。

小 结

- 4. 以2选1选择器为例介绍了VHDL代码的各要素。
- 5. 介绍了利用MAX＋plusII进行VHDL文本输入设计的基本方法和流程。
- 6. 介绍了使用VHDL语言描述3个不同电路的实例。
- 7. 简单介绍了SOPC知识。

设计练习

- 1. VHDL代码有哪些基本端口模式？
- 2. 简述端口模式INOUT与BUFFER的不同。
- 3. 简述利用MAXplus II进行VHDL文本输入设计的一般流程。
- 4. 请分别使用VHDL代码中“IF　THEN”语句、“WHEN　ELSE”语句和“CASE”语句设计一个四选一的选择器。
- 5. 简述SOPC技术。