ABEL PLD 编程笔记

一. ABEL 语言的组成元素要点

ABEL 源文件构成

模块开始（module 语句）

……

标志（flags 语句）　　说明段

标题（title 语句）

……

器件定义（device 语句）　　定义段

管脚、节点定义（pin，node 语句）

属性定义（istype 语句）

常量定义（constant 语句）

宏定义 （macro 语句）

……

逻辑方程式（equations 语句）　　描述段

真值表（truth_table 语句）

状态图（state_diagram 语句）

……

熔丝段定义（fuses 语句）　　熔丝段

测试向量（test_vectors 语句）　　测试段

……..

模块结束（end 语句）

[例]

module m6809a　　（模块语句）

title '6809 memory decode　　（标题语句）

Jean Designer　　Data I/O Corp Redmond WA　　24 Feb 1984'

U09a　　**device**　'P14L4';　（器件定义）

A15,A14,A13,A12,A11,A10 **pin** 1,2,3,4,5,6;　　（管脚定义）

ROM1,IO,ROM2,DRAM　　**pin** 14,15,16,17;

H,L,X　　= 1,0, .X.;　　（常量定义）

Address = [A15,A14,A13,A12, A11,A10,X,X, X,X,X,X, X,X,X,X];

equations　　（方程）

!DRAM　　= (Address <= ^hDFFF);

!IO　　= (Address >= ^hE000) & (Address <= ^hE7FF);

!ROM2　　= (Address >= ^hF000) & (Address <= ^hF7FF);

```
        !ROM1     = (Address >= ^hF800);

test_vectors (Address -> [ROM1,ROM2,IO,DRAM])      （测试向量）
                ^h0000   -> [   H,   H,   H,     L ];
                ^h4000   -> [   H,   H,   H,     L ];
                ^h8000   -> [   H,   H,   H,     L ];
                ^hC000   -> [   H,   H,   H,     L ];
                ^hE000   -> [   H,   H,   L,     H ];
                ^hE800   -> [   H,   H,   H,     H ];
                ^hF000   -> [   H,   L,   H,     H ];
                ^hF800   -> [   L,   H,   H,     H ];
end m6809a
```

ABEL 关键字

CASE	FUSES	PIN
DEVICE	GOTO	STATE
ELSE	IF	STATE_DIAGRAM
ENABLE	IN	TEST_VECTORS
END	ISTYPE	THEN
ENDCASE	LIBRARY	TITLE
ENDWITH	MACRO	TRUTH_TABLE
EQUATIONS	MODULE	WITH
FLAG	NODE	

注：关键字不分大小写

ABEL 标识符

标识符是用来标识器件、器件引脚、节点、集合、输入/输出信号等的合法的 ASCII 字符序列。标识符的限制规则为：

1. 标识符必须以字母或下划线开始；
2. 标识符最长不能超过 31 个字符；
3. 标识符可以包含大小写字母、数字及下划，但不允许出现空格；
4. 标识符区分大小写。

字符串

字符串是包含在双引号内的 ASCII 字符序列，通常用于标题语句、标志语句及管脚和节点说明语句中。如：

TITLE　‘1 to 8 line demultiplexer’；

DMI‘P16L8’；

注释

注释以双引号开始，以另一双引号或回车结束。如：

“declaration section”

module　Basic_logic；”gives the module a name（回车）

4 种基数表示方法

基数名称	基数	表示方法	举例	十进制值
二进制	2	^b	^b101	4
八进制	8	^o	^o77	63
十进制	10	^d	^d43 或 43	43
十六进制	16	^h	^h0F	15

特殊常量

.C.　　正脉冲时钟输入
.K.　　负脉冲时钟输入
.F.　　浮动输入或输出
.P.　　寄存器预装载
.X.　　任意态
.Z.　　高阻态测试输入或输出
.SVn.　　n = 1 - 9　驱动输入到超级电平 2 - 9

赋值运算符

=　　非时钟赋值（组合逻辑输出）
:=　　时钟赋值（寄存器输出）

运算符及其优先级

类别	运算符	优先级	说明
算术运算符	-	1	取补
	<<	2	左移
	>>	2	右移
	*	2	乘法
	/	2	无符号除法
	%	2	取模
	+	3	加法
	-	3	减法
逻辑运算符	!	1	非
	&	2	与
	#	3	或
	$	3	异或
	!$	3	同或
关系运算符	==	4	等于
	!=	4	不等于
	<	4	小于
	<=	4	小于或等于
	>	4	大于
	>=	4	大于或等于

注：优先级最高为 1，最低为 4。

布尔方程语法

[WHEN **条件** THEN] [！] …… [ENABLE] **元素 = 表达式；**[ELSE 方程]

或者，

[WHEN **条件** THEN] [！] …… [ENABLE] **元素 ：= 表达式；**[ELSE 方程]

式中：

条件：任意合法表达式；

元素：给一个或一组信号命名的标识符，或要被表达式赋值的集合；

表达式：由标识符和运算符号组成的式子，求值后得出一个结果。

[例]

WHEN B THEN A = B；ELSE A = C；

X = A & B；

Y ：= A & B；

ENABLE Y = C # D；若 C # D 为真，Y 被使能（并不是 Y 输出为 Y = C # D）

集合

集合是一组可作为整体进行运算的信号和常量。表示方法是用方括号将这些信号和常量括起来，信号和常量之间需用逗号或排列号（..）隔开。

例如：

MULTOUT = $[B_0, B_1, B_2, B_3, B_4, B_5, B_6, B_7]$；

SELECT = $[S_0, S_1, S_2]$；

或用排列符定义：

MULTOUT = $[B_0..B_7]$；

SELECT = $[S_0..S_2]$；

用排列符定义时，排列符两边的标识符名必须一致，即带数字下标的字母要相同。也可按下标递减排列。

对集合的运算是对应集合中的每个元素进行的。对两个以上集合进行运算，集合中的元素个数必须相等。

[例 1]

用集合运算实现布尔方程

ChipSel = $A_{15}\&!A_{14}\&A_{13}$；

首先，定义一个包含 A_{15}, A_{14}, A_{13} 的常量集合

Addr = $[A_{15}, A_{14}, A_{13}]$；

于是方程

ChipSel = Addr = = [1，0，1];

的逻辑功能等效于 ChipSel = $A_{15}\&!A_{14}\&A_{13}$. 因为如果 Addr = = [1，0，1]，即 $A_{15}=1, A_{14}=0, A_{13}=1$，那么 ChipSel 为真。上述集合方程也可写成：

ChipSel = Addr = = 5；

本例定义了一个包含 16 位地址线中高 3 位的一个集合，并用于集合运算中，也可用其它方法对全部地址进行效果相同的操作。请看下例。

[例 2]
如果在定义段中定义了常量：
Addr = $[A_{15}..A_0]$；
X = .X.；
那么，下例两种方法等效于地址线集合中只使用地址高三位的方法：
方法 1：ChipSel = Addr = = [1, 0, 1, X, X, X, X, X, X, X, X, X, X, X, X, X];
方法 2：ChipSel = (Addr >= ^hA000) & (Addr <= ^hBFFF);

[例 3]
集合赋值
Sigsel = [1，1，0] & [0，1，0];
结果是
Sigsel = [0，1，0];
也可写成
Sigsel =6 & 3;

[a，b] = c & d;
等效于下面两个赋值
a = c & d;
b = c & d;
用于集合赋值、比较的数要转换成二进制，并遵从以下规则：
1. 若该二进制数的有效位数多于集合中元素的个数，要从左边截去多余位；
2. 若该二进制数的有效位数少于集合中元素的个数，要从左边用 0 补齐。

块
块是在大括号内的 ASCII 文本段。块用于宏定义和指示字中，块中的文本可以为一行或多行。块可以嵌套。
[例]
{This is a block}
再如
{
This is also a block，and it
spans more than one line
}
再如
{
A = B # C;
D = [0，1] & [1，0];
}

变量及变量代换
哑变量：在宏定义、模块或指示字中可被真实变量替代的标识符；
真实变量：用于宏定义、模块或指示字中的变量，可替代哑变量。
在需要用真实变量取代哑变量的地方，哑变量前要加一个问号“？”，以与其它标识

符区分开来。如下面的宏定义

OR_EM　MACRO（a，b，c） {?a　#　?b　#　?c}；

中，a，b，c 就是哑变量。下列方程是对 OR_EM 宏的宏引用

D = OR_EM（X，Y，Z）；

其中 X，Y，Z 是真实变量。

二. ABEL 的语言结构

ABEL 源文件有若干成为模块的部分组成。各模块相互独立，每个模块都包含多个完整的逻辑设计，结构如下：

……

（1st module Start 第一模块开始）

模块开始（module 语句）

标志（flags 语句） 说明段

标题（title 语句）

器件定义（device 语句） 定义段

管脚、节点定义（pin，node 语句）

属性定义（istype 语句）

常量定义（constant 语句）

宏定义 （macro 语句）

逻辑方程式（equations 语句） 描述段

真值表（truth_table 语句）

状态图（state_diagram 语句）

熔丝段定义（fuses 语句） 熔丝段

测试向量（test_vectors 语句） 测试段

模块结束（end 语句）

（1st module End 第一模块结束）

……

（2nd module Start 第二模块的开始）

……

（2nd module End 第二模块结束）

……..

（3th module Start 第三模块的开始）

……

（3th module End 第三模块结束）

……..

等等…

模块结构必须遵从以下原则：

1. 模块开头为 module 语句，结束必须用相应的 end 语句；
2. 若使用 flags 语句，必须为 module 语句后的第一条语句；
3. 若使用 title 语句，必须为 flags 语句后的第一条语句。若没有 title 语句，则必须为 module 语句后的第一条语句；
4. 一个模块至少有一个定义段，模块中可按需要以任意次序使用多个定义段。

模块语句和结束语句

语法：module 模块名 [（哑变量名 [，哑变量名]…）]

模块内容…

end 模块名 [；]

模块名——一个代表模块名称的合法标识符；

哑变量名——哑变量名称。

若模块选用哑变量，则在使用语言处理程序处理模块时，可将真实变量传给模块。哑变量可为模块引用。模块中，凡带有“？”号的哑变量，语法分析程序都用真实变量将其取代。

[例]

```
module    my_example    (A，B)
        ……
        C = ?B + ?A;
        ……
end    my_example
```

标志语句

标志语句为定义命令参数提供了又一方法。通常这些参数是由批处理命令行提供的，或单独由某处理程序的命令行来提供。标志语句则可使源语句直接指定处理参数。由命令行输入的参数可取代 FLAG 标志语句指定的参数。

[例]

```
flag    ‘-r2’, ‘-t3’
```

标题语句

标题语句赋予模块一个标题，该标题将作为生成的编程器下载文件及设计编制文件的题头。但如果标题语句的字符串中含*号，则编程器下载文件不再使用该标题作为题头。

[例]

```
title '6809 memory decode
Jean Designer        Data I/O Corp Redmond WA      24 Feb 1984'
```

定义段

在定义段中，定义语句共有 6 种：

device	器件定义
pin	管脚定义
node	节点定义
constant	常量定义
macro	宏定义
attribute	属性定义

1. 器件定义

语法：器件名 [，器件名]… device 实际器件；

器件名——模块中说明一个器件所用的标识符；

实际器件——实际器件的工业型号，用字符串表示。

生成的编程器下载文件名将为本处定义的器件名加.JED 扩展名。

[例]

```
U14，U15    device    ‘F159’
```

2. 管脚定义

语法：管脚名 [，管脚名]… pin [in 器件名] 管脚号[= ‘属性[，属性]… ’][，管脚号[= ‘属性[，属性]… ’]]…；

管脚名——管脚标识符；

器件名——已定义的器件名标识符；

管脚号——实际器件的管脚号；
属性 ——在管脚可编程器件中，表示管脚属性的字符串，其中有

pos	正极性	feed-or	由或门反馈
neg	负极性	reg-d	D 型寄存器
reg	寄存器信号	reg-g	G 型（时钟使能）寄存器
com	组合信号	reg-jk	JK 型寄存器
latch	锁存输入脚	reg-rs	RS 型寄存器
feed-pin	由管脚反馈	reg-t	T 型寄存器
feed-reg	由寄存器反馈	reg-jkd	JK / D 可控寄存器

[例]
!Clock, Rest, SI pin in U12 , 12, 15, 3；

这行代码将器件 U12 的管脚 12, 15, 3 分别用标识符 Clock, Rest, SI 来代表。若模块中不止定义一个器件，必须用语句中的 in 项来指明器件。

[例]
管脚的属性可由语句中的属性项或 ISTYPE 语句定义，比如

FO pin 13 = 'neg, reg'；

3. 节点定义

类似管脚的定义，略。

4. 常量定义

语法：常量名 [，常量名]… = 表达式 [，表达式]…；
常量名——表示一个常量的标识符；
表达式——定义常量值的表达式。
[例]

A， B，C = 5，[1，0]，6；
G = [1，2] + [3，4]；

5. 宏定义和宏代换

语法：宏名 MACRO [（哑变量 [，哑变量]…）] 块；

宏名——宏的标识符。

宏在源文件中用来定义可多次引用的重复代码。一个宏定义后，便可在整个模块中使用，且只能在其定义的模块中使用。
[例]
若定义了

NAND3 MACRO（A，B，C）{！（?A & ?B & ?C）}；

则引用

D = NAND3（Clock，Hello，Busy）；

相当于

D = ！（Clock & Hello & Busy）；

6. ISTYPE 属性定义

语法：信号名 [，信号名]… ISTYPE [IN 器件名] ‘属性 [，属性]…’；

信号名——管脚或节点标识符；

器件名——已定义的、管脚或节点所在的器件名标识符。

属性 ——在管脚可编程器件中，表示管脚属性的字符串（见管脚定义）。

[例]

FO，A istype ‘neg，latch’；

将 FO 和 A 定义为负极性锁存信号。

7. LIBRARY 库语句

语法：LIBRARY ‘名称’；

名称——定义文件名（不包含扩展名）的字符串。

库语句使被指明的文件内容插入 ABEL 源文件中。插入从库语句开始。

文件扩展名 .inc 是定义文件的默认扩展名，这样的文件将被查找，如果找不到，将试图在库文件 abel3lib . inc 中找该文件。

方程语句

语法：equations [in 器件名]

器件名——已定义过的器件标识符，表示与方程有关的器件。

方程语句表示与某个器件有关的一组方程的开始，后面跟针对所指器件的布尔方程。若模块中只有一个器件，器件名这一项可以任选。

[例]

```
equations  in  IC13
    A = B & C # A;
    [W,  Y] = 3;
    WHEN  B  THEN  A = B;
    ELSE  A = C;
```

真值表

1. 表头向量

语法：truth_table [in 器件名] (输入向量 -> 输出向量)

或

语法：truth_table [in 器件名] (输入向量 ：> 输出向量)

或

语法：truth_table [in 器件名] (输入向量 ：> 寄存器输出 -> 输出向量)

器件名——已定义过的器件标识符，表示与真值表有关的器件；

输入向量——逻辑关系中的输入部分；

输出向量——逻辑关系中的输出部分；

寄存器输出——信号寄存（时钟同步）后输出；

-> —— 表示输入输出为组合型；

：> —— 表示输入输出为寄存器型；

若模块中只有一个器件，器件名这一项可以任选。

2. 真值表格式

格式由真值表的表头向量定义，真值表本身即为按格式定义格式排列的一组输入输出信号。真值表中使用的所有信号必须为常量。

[例 1]

定义带使能的异或门

```
truth_table   in IC6      （[EN，A，B] -> C）
[0，  .X.，  .X.]  -> .X.;
[1，    0，    0]   -> 0;
[1，    0，    1]   -> 1;
[1，    1，    0]   -> 1;
[1，    1，    1]   -> 0;
```

[例 2]

定义一个简单的状态机。当前状态用放在一个集合里的信号 A，B 表示，下一状态用寄存器输出信号 C，D 表示，它们也放在一个集合中。输出信号为一简单的组合型信号。

该状态机工作时，在各状态之间循环。当 A = 0，B=1 时，将 E 置为低电平。

```
truth_table   in IC17   ([A，B]：> [C，D] -> E)
[0，0]：> [0，1] -> 1;
[0，1]：> [1，0] -> 0;
[1，0]：> [1，1] -> 1;
[1，1]：> [0，0] -> 1;
```

也可写为

```
truth_table   in IC17   ([A，B]：> [C，D] -> E)
0：> 1 -> 1;
1：> 2 -> 0;
2：> 3 -> 1;
3：> 0 -> 1;
```

状态图

状态图设计需用 state_diagram 结构，该结构表示所设计为一状态机，其工作过程由结构中的 IF - THEN - ELSE 语句、CASE 语句和 GOTO 语句来定义。

1. 状态图定义

语法：state_diagram [in 器件名] 当前状态名 [-> 下一状态名]

```
        state 状态表达式：      [方程];
                                [方程];
                                ……
                                转移语句;
```

器件名——已定义过的器件标识符，表示与状态图有关的器件；

当前状态名—— 一个或一组标识符，表示决定状态机当前状态的信号；

下一状态名—— 一个或一组标识符，表示状态机的下一个状态的信号。用于带外部寄存器的设计；

状态表达式—— 表示当前状态的表达式；

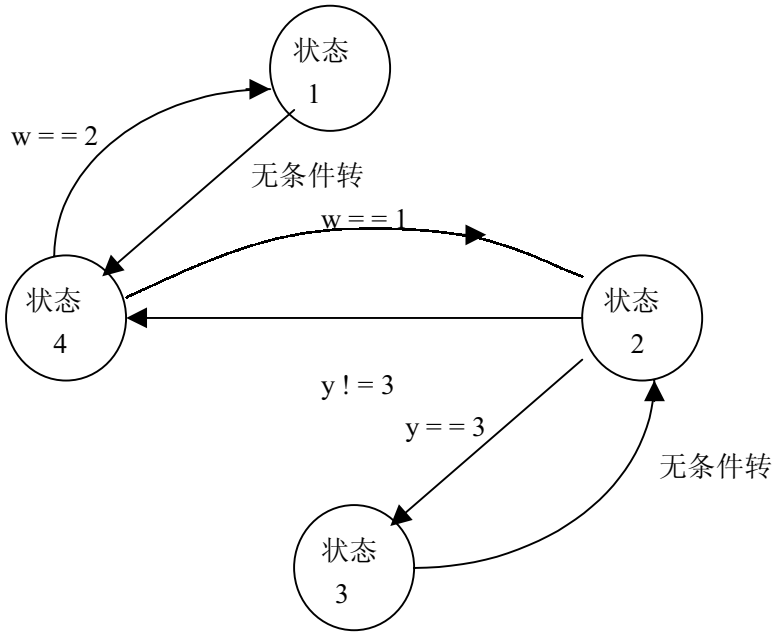
方程—— 定义状态机输出信号的合法方程；

转移语句—— 可为 IF - THEN - ELSE 语句、CASE 语句和 GOTO 语句，后边可跟 WITH - ENDWITH 语句。

[例]

```
current_state = [a，b]；常量定义
state_diagram   in U15 current_state
    state 1：  w = 1；
               y = 1；
               goto 4；
    state 2：  if y = = 3 then 3 else 4；
    state 3：  w = 2；
               y = w；
               goto 2；
    state 4：  y = 3；
               case w = = 1：2；
                    w = = 2：1；
               endcase；
```

以下是本例的状态图：



若初态为 1（a = 0，b = 1），则经历的状态顺序为：1，4，2，3，2，4，1…

2. IF - THEN - THEN 语句

语法：if 表达式 then 状态表达式 [else 状态表达式]；

[例]

if X - Y then J else K； 若 X - Y 不等于零，转入 J 态，否则转入 K 态。

3. 链式 IF - THEN - THEN 语句

语法：if 表达式 then 状态表达式 [else 状态表达式]；

else

```
        if 表达式  then 状态表达式
      else
        if 表达式  then 状态表达式
      else  状态表达式;
```

可链接的数目没有限制，但最后一个语句必须以分号结尾。

[例]

```
if  a  then  1
  else
  if  b  then  2
  else
  if  c  then  3
  else  0;
```

4. CASE - ENDCASE 语句

语法：case [表达式 ： 状态表达式];

```
          [表达式 ： 状态表达式];
          [表达式 ： 状态表达式];
          ……
        endcase:
```

[例]

```
case  a = = 0：1;
      a = = 1：2;
      a = = 2：3;
      a = = 3：0;
endcase;
```

5. GOTO 语句

语法：goto 状态表达式;

[例]

```
goto  x + y;
```

6. WITH - ENDWITH 语句

语法：转移语句 状态表达式 with 方程;

```
  [方程];
  ……
  endwith;
```

转移语句—— 即 IF、ELSE 或 CASE 语句;

状态表达式——表示下一状态;

方程—— 表示状态机的输出信号。

WITH - ENDWITH 语句与 IF -THEN 或 CASE 语句连用时，可使你根据转移状态写出相应的输出方程。

[例]

```
case  5：if  a = = 1  then  1  with  X：= 1;
             Y：=0;
             endwith;
```

```
else  2  with  X：= 0；
  Y：=1；
  endwith；
```

FUSES 熔丝图定义段

源文件中的 FUSES 熔丝图定义段可以用来定义指定器件中任意熔丝的状态。（略）

测试向量表

语法：test_vectors [in 器件名] [注释] (输入向量 -> 输出向量)

[输入信号值 -> 输出信号值]；

[输入信号值 -> 输出信号值]；

……

如果有注释，它也会出现在下载文件中。

[例]

```
test_vectors（[A，B] -> [C，D]）
              [0，  0] -> [1，1]；
              [0，1] -> [1，0]；
              [1，0] -> [0，1]；
              [1，1] -> [0，0]；
```

三. ABEL 3.0 命令行

1。PARSE 语法分析程序

PARES ［-I 输入文件名］［-O 输出文件名］［-L 列表文件名］［-E］［-P］
［-A 变量名］...［-H 路径名］［-Y 路径名］

-I 输入文件名： 输入文件名即为 ABEL 源文件名
-O 输出文件名：本程序产生的输出文件名（.TM1）
-L 列表文件名：生成的列表文件名，内容为经语法分析后的源代码，如有错，还包含错误信息。如果使用了-E 参数，列表中还要有扩展了的宏，及由指示字引用的代码。 如果使用的为-P 参数，列表中还要给出引用代码的指示字。
-E： 使列表文件给出宏扩展及指示字引用后的扩展文本
-P： 使列表文件给出宏扩展及指示字引用后的扩展文本，并给出代码的指示字引用
-A 变量名： -A 后的变量名松入源文件中，进行变量替代。
-H 路径名： 指出源文件中由指示字 @INCLUDE 所引用文件的路径名
-Y 路径名： 指出器件说明文件所在路径名

2。TRANSFOR 变换程序

TRANSFOR ［-I 输入文件名］［-O 输出文件名］

-I 输入文件名： 输入文件名为 PARSE 生成的输出文件名
-O 输出文件名：本程序生成的输出文件名（.TM2）。

3。REDUCE 逻辑化简程序

REDUCE ［-I 输入文件名］［-O 输出文件名］［-Rn］

-I 输入文件名： 输入文件名为 TRANSFOR 生成的输出文件名
-O 输出文件名：本程序生成的输出文件名（.TM3）。
-Rn： n=0,1,2,3 说明化简使用的级别（**默认值为 1**）
n=0 不进行化简
n=1 进行简单化简
n=2 进行简单及 PRESTO 化简
n=3 逐个管脚进行简单及 PRESTO 化简

4。FUSEMAP 熔丝图生成程序

FUSEMAP［-I 输入文件名］［-O 输出文件名］［-J 路径名］［-Cn］［-Dn］［-Kq］

-I 输入文件名： 输入文件名为 REDUCE 生成的输出文件名
-O 输出文件名：本程序生成的输出文件名（.OUT）。
-J 路径名： 说明编程器下载文件要存放的路径名
-Cn： 校验和，n 可取以下值
n=0 编程器下载文件不使用 STX、ETX 及传输校验和等参数
n=1 编程器下载文件中使用 STX、ETX 及哑传输校验和等参数

n=0 编程器下载文件不使用 STX、ETX 及传输校验和等参数（默认态）

-Dn：　编程器下载文件的格式说明，n=0 表示取标准的 JEDED 格式（默认态），n 为其它值表示取某种微处理器格式

-Kq：　q 值决定 FPLA 或阵列中未用熔丝状态为通还是断，q=Y 表示未用熔丝接通，q=N 表示未用熔丝断开

5。SIMULATE 仿真程序

SIMULATE［-I 输入文件名］［-O 输出文件名］［-Tn］［-N 器件型号］
［-Bn1，n2［，n3］］［-Xn］［-Zn］［-Wn, n....n ］

-I 输入文件名：　输入文件名可为 FUSEMAP 生成的输出文件名或 JEDEC 格式的编程器下载　文件名

-O 输出文件名：本程序生成的输出文件名（.SIM）。

-Tn：　n 为仿真的跟踪级别，可取下列值
n=0 只给出仿真错误（**默认态**）
n=1 给出测试向量
n=2 给出输出及测试向量
n=3 给出完整的器件信号
n=4 给出指定管脚的仿真波形
n=5 给出指定管脚的逻辑值

-N 器件型号：　若仿真输入的为下载文件，需给出所有器件的型号

-Bn1,n2 [,n3]：　在测试向量 n1 与 n2 间设置断点，断点间（含向量 n1 与 n2）的仿真级别由　n3 定义

-Xn：　给下载文件测试向量中的“.X.”赋值，n 可为 1, 0, H, L，默认值为 0

-Zn：　给下载文件测试向量中的“.Z.”赋值，n 可为 1, 0, H, L，默认值为 1

-Wn, n....n：　指定在跟踪级为 3，4 和 5 的仿真过程中哪些器件管脚要“观测”

6。DOCUMENT 设计文件编制程序

DOCUMENT［-I 输入文件名］［-O 输出文件名］［-V］［-Fn］［-G］［-S］［-Qxyz］

-I 输入文件名：　输入文件名为 FUSEMAP 生成的输出文件名

-O 输出文件名：本程序生成的输出文件名（.DOC）。

-V：　列出测试向量

-Fn：　列出熔丝图的 and/or 项
n=0 列出熔丝图及器件利用信息
n=1 只列出器件利用信息
n=2 给出简明熔丝图

-G：　列出器件芯片图

-S：　列出符号表

-Qxyz：　选择要列的方程，x, y, z 为下列值的任意组合
2：列出化简后的方程
1：列出变换后的方程
0：列出原有方程

附录 1　D: \ DATAIO \ WORK \ ABL.BAT　批处理文件

```
ECHO OFF
rem    input *.abl file
rem    output n*.jed file
copy %1.abl d:\dataio
if exist %1.vec copy %1.vec d:\dataio
cd\dataio
abel %1
```

附录 2　D: \ DATAIO \ ABEL.BAT　批处理文件

```
: ABEL(tm) Version-3.00a    Copyright(C) 1983-1988 FutureNet Div, Data I/O Corp.
echo off
if X%1 == X goto help
if NOT X%1 == X-help goto begin
:help
echo :    NAME:    abel = batch script to execute all phases of ABEL (COMMAND.COM)
echo :    USAGE: abel    name [args]...
echo :           name = design file name -- without suffix [.abl]
echo :           args = command line options
echo :    FILES:      name.abl -- source file
echo :           name.lst, name.sim, name.doc, name.out, device.jed -- results
echo :           name.tm1, name.tm2, name.tm3 -- intermediate files
echo :    Expects device files to be in the directory specified by ABEL3DEV
goto exit

:begin
echo + abel %1
PARSE    -i%1.abl -o%1.tm1 -l%1.lst    %2 %3 %4 %5 %6 %7 %8 %9      语法分析，生成.tm1
if errorlevel 1 goto parserr
TRANSFOR -i%1.tm1 -o%1.tm2    %2 %3 %4 %5 %6 %7 %8 %9          生成.tm2 文件
if errorlevel 1 goto tranerr
if exist %1.tm1 del %1.tm1
REDUCE -i%1.tm2 -o%1.tm3 %2 %3 %4 %5 %6 %7 %8 %9          逻辑化简，生成.tm3 文件
if errorlevel 2 goto fuse
if errorlevel 1 goto rederr
:fuse
if exist %1.tm2 del %1.tm2
FUSEMAP -i%1.tm3 -o%1.out %2 %3 %4 %5 %6 %7 %8 %9          生成.jed 编程器下载文件
if errorlevel 1 goto fuserr
if exist %1.tm3 del %1.tm3
SIMULATE -i%1.out -o%1.sim %2 %3 %4 %5 %6 %7 %8 %9        模拟仿真，生成.sim 文件
if errorlevel 1 goto simerr
DOCUMENT -i%1.out -o%1.doc -f0 -g -q2 %2 %3 %4 %5 %6 %7 %8 %9          生成.doc 文件
```

```
if errorlevel 1 goto docerr
goto exit
:parserr
        echo ? abel: Error in parsing source              (see PARSE)
        goto exit
:tranerr
        DOCUMENT -i%1.tm1 -o%1.doc    -g -q0 %2 %3 %4 %5 %6 %7 %8 %9
        echo ? abel: Error in transforming terms       (see TRANSFOR)
        goto exit
:rederr
        DOCUMENT -i%1.tm2 -o%1.doc -g -q1 %2 %3 %4 %5 %6 %7 %8 %9
        echo ? abel: Error in reducing factors          (see REDUCE)
        goto exit
:fuserr
        DOCUMENT -i%1.tm3 -o%1.doc -g -v -q2 %2 %3 %4 %5 %6 %7 %8 %9
        echo ? abel: Error in generating fusemap       (see FUSEMAP)
        goto exit
:simerr
        DOCUMENT -i%1.out -o%1.doc -f -g -v -q2 %2 %3 %4 %5 %6 %7 %8 %9
        echo ? abel: Error in simulating logic/vectors          (see SIMULATE)
        goto exit
:docerr
        echo ? abel: Error producing documents          (see DOCUMENT)
:exit
if exist *.jed copy *.jed \dataio\work
if exist *.lst copy *.lst \dataio\work
if exist *.doc copy *.doc \dataio\work
if exist *.sim copy *.sim \dataio\work
if exist *.out copy *.out \dataio\work
rem delete abel work files
if exist *.tm? del *.tm?
if exist *.jed del *.jed
if exist *.sim del *.sim
if exist *.lst del *.lst
if exist *.doc del *.doc
if exist *.bak del *.bak
if exist *.out del *.out
if exist *.p8? del *.p8?
del *.abl
if exist temp del temp
if exist *.vec del *.vec
cd\dataio\work
```

晓风　treehouse@sina.com