本手册分两部分：

第一部分 HI-TECH PICC C 的使用说明. 这里我们只讲述了 PICC C 与标准 C 的不同,
它不是一本 C 语言的教程, 并且我们假定你有 C 语言的基础.

第二部分 在伟福集成环境下使用 PICC.
讲述在伟福集成环境如可设置 PICC, 简单的调试步骤. 更详细的说明请参阅
伟福仿真器使用手册.
关于如何在 MPLAB 下使用 PICC C 语言, 请参阅 Microchip 相应的手册.

该软件是以 Microhcip 公司提供的 PICC 限 7 版（智能编译 57 67 77 877）为准

相关资料获取网站：

M icrochip 网址: www.microchip.com

Hi-tech 公司网址: www.htsoft.com

第一部分

为了对 PIC 单片机有更好的支持，PICC 在标准 C 的基础上作了一些扩充：

- 定义 I/O 函数，以便在你的硬件系统中使用<stdio.h>中定义的函数。
- 用 C 语言编写中断服务程序
- 用 C 语言编写 I/O 操作程序
- C 语言与汇编语言间的接口

1-1 与标准 C 的不同

PICC 只在一处与标准 C 不同：函数的重入。

因为 PIC 单片机的寄存器及堆栈有限，所以 PICC 不支持可重入函数。

1-2 支持的 PIC 芯片

PICC 支持很多 PIC 单片机，支持 PIC 单片机的类型在 LIB 目录下的 picinfo.ini 文件中有定义。

1-3 PICC 包含一些标准库

1-4 PICC 编译器可以输出一些格式的目标文件，缺省设置为输出 Bytecraft 的'COD' 格式和 Intel 的'HEX'格式。你可以用表 1-1 中的命令来指定输出格式。

表 1 - 1

格式名称	描述	PICC 命令	文件类型
Motorola HEX	**S1/S9 type hex file**	**-MOT**	**.HEX**
Intel HEX	**Intel style hex records(缺省)**	**-INTEL**	**.HEX**
Binary	**Simple binary image**	**-BIN**	**.BIN**
UBROF	**Universal Binary Image Relocatable Format**	**-UBROF**	**.UBR**
Tektronix HEX	**Tektronix style hex records**	**-TEK**	**.HEX**
American Automation HEX	**Hex format with symbols for American Automation emulators**	**-AAHEX**	**.HEX**
Bytecraft .COD	**Bytecraft code format(缺省)**	**n/a(缺省)**	**.COD**
Library	**HI-TECH library file**	**n/a**	**.LIB**

1-5 符号文件

PICC -G 命令用于生成符号文件，有了符号文件，你就可以进行源程序调试.

命令格式为： PICC -16F877 -G test.c

在使用仿真器时必须使用-G 命令。

1-6 配置字

PIC 单片机的配置字可以用__CONFIG 命令来定义：

```
#include <pic.h>
__CONFIG(x)
```

其中 x 是配置字，头文件中定义了相应的配置说明符，如：

```
__CONFIG(WDTDIS & XT & UNPROTECT);
```

这将关闭看门狗，设置 XT 振方式，程序不加密。注意：不同的

配置符间用'&'相联，未定义的部分保留未编程状态。详细的情况请参考头文件及 PIC 数据手册。

1-7 ID 位置

有些 PIC 单片机在程序空间外还有 ID 空间，可用下面的方法来定义：

```
#include <pic.h>
__IDLOC(x)
```

其中 x 是 ID 标示，如：

```
__IDLOC(15F0);
```

将 ID 的四个单元定义为：1, 5, 15, 0. ID 的具体位置由所指定的 PIC 芯片自动设定。

1-8 EEPROM 数据

有些 PIC 单片机支持用外部编程器对内部的 EEPROM 进行编程。

__EEPROM_DATA()可以将用于初始化的数据放入 HEX 文件中，如：

```
__EEPROM_DATA(0, 1, 2, 3, 4, 5, 6, 7)
```

可将 0-7 八个数放入 HEX 文件中，在用外部的编程器进行编程时将这八个数写入 PIC 单片机中。

__EEPROM_DATA 不是用于运行时写入 EEPROM 数据的，在运行时请用 EEPROM_READ(), EEPROM_WRITE()。

1-9 位指令

只要有可能，PICC 总是采用位指令。如：

```
int foo;
foo |= 0x40;
```

的编译结果为：`bsf _foo, 6`

为了方便可以定义如下宏：

```
#define bitset(var, bitno) ((var) |= (1 << (bitno)))
#defien bitclr(var, bitno) ((var) &= (1 << (bitno)))
```

上一条语句可写为：`bitset(foo, 6);`

1-10 支持的数据类型

PICC 支持 1,2,4 字节的基本类型。 所有的多字节类型都采用低有效位在前的格式，表 1-2 列出了所有数据类型及它们所占空间大小。

表 1 - 2

类型	大小(位)	数字类型	值
bit	**1**	**逻辑类型**	**0 或 1**
signed char	**8**	**有符号字符**	**-128..+127**
unsigned char	**8**	**无符号字符**	**0..255**
signed short	**16**	**有符号整数**	**-32768..+32767**
unsigned short	**16**	**无符号整数**	**0..65535**
signed int	**16**	**有符号整数**	**-32768..+32767**
unsigned int	**16**	**无符号整数**	**0..65535**
signed long	**32**	**有符号整数**	**-2147483648..+2147483647**
unsigned long	**32**	**无符号整数**	**0..4294967295**
float	**24**	**浮点**	
double	**24 or 32**	**浮点**	**由-D24, -D32 决定**

1-10-1 常量及进制表示

PICC 支持标准 C 的进制表示方法。

l 或 L 后缀表明常量为 long 类型， u 或 U 后缀表示常量为 unsinged 类型。

浮点数为 double 类型，可以用 f 或 F 指定浮点数为 float 类型。

字符型由单引号括起，如'a'.

字符串由双引号括起，如"Hello world".

1-10-2 位数据类型

PICC 支持一位的变量，用 bit 来定义。如：

```
static bit init_flag;
```

变量必须是全局的或静态的，它不能是自动变量或一个函数的参数，但可以作为一个函数的返回类型。

位变量很象 unsigned char，但它只有 0 或 1 两个值，位变量占用空间少，且运算速度快. 所有的位变量在 startup 是被清 0，请在程序开始处初始化它们。

如果将一个整型数赋给位变量，只是将最低位赋给位变量，如果你是想要将一个整型变量是否为 0 赋值给一个位变量，请用: bitvar = other_var != 0;

如要你使用了 PICC 的-STRICT 命令，bit 将被视为非法命令。

1-10-2-1 使用可位寻址的寄存器

位变量的定义可以与绝对地址的定义结合起来使用。如：

为了访问 STATUS 中 Power Down 位，先定义 STATUS 的绝对地址为 3，然后再定义一位变量绝对地址为 27

```
static unsigned char STATUS @ 0x03;
static bit PD @ (unsigned)&STATUS*8+3;
```

注意: 头文件中已定义所有的特殊功能寄存器及相应的位寄存器.

1-10-3

PICC 浮点数使用 IEEE754 32 位格式和 IEEE754(截断)24 位格式.
float 类型使用 24 位格式，double 使用 24 位或 32 位格式，由 PICC 命令控制,
-D24 使用 24 位格式， -D32 使用 32 位格式.

1-11 绝对地址变量

一个全局的或静态的变量可以定位绝对地址，使用如下格式:

```
unsigned char Portvar @ 0x06;
```

这里定义了一个名为'Portvar'的变量，地址为 06h，注意，编译器并不保留任何单元，仅仅是将一个变量分配在 06h 单元. 它等价于汇编语言: _Portvar EQU 06h
编译器及连接器都不作任何检查，完全由程序员保证分配不会发生冲突.

1-12 结构与联合

PICC 支持 struct 及 union，它们可以作为函数的参数及返回值，也可以作为指针指向的目标.

1-12-1 结构限定

PICC 支持在结构上使用限定符，如果在一个结构上使用限定符，那么，这个结构的所有成员都被限定. 如:

```
bank1 struct {
       int number;
       int *ptr;
}
```

在这个结构里，number, ptr 都被放在 bank1 寄存器内.

1-12-2 结构中定义位成员

PICC 支持在结构中定义位成员.
位成员按最低有效位在前的方式存储，位成员总是按 8 位字节方式存放，当当前字节放满后再放下一个字节，位成员不会跨字节存放. 如:

```
struct {
  unsigned  hi: 1;
  unsigned  dummy: 6;
  unsigned  lo: 1;
} foo @ 0x10;
```

结构 foo 占用 10h 单元, hi 为 10h 单元的第 0 位, lo 为 10h 单元的第 7 位, dummy 为 10h 单元的 2-6 位,(第 6 位为最高有效位)

不使用的位可用未命名的位成员来定义，如果我们不使用 dummy，就可定义为:

```
struct {
  unsigned  hi: 1;
  unsigned    : 6;
  unsigned  lo: 1;
} foo @ 0x10;
```

1-13 在 ROM 及 RAM 存放字符串.

一个未说明的字符串总是存放在 ROM 中，并且只能通过常量指针为访问.

```
#define HELLO "Hello word"
SendBuff(HELLO);
```

一个非常量的数组被一个字符串初始化，如:

```
char fred[] = 'Hello world";
```

将在 RAM 中保留一个数组，在 startup 时，用存放在 ROM 中的"Hello world"来初始化.

如果要将一个常数字符串作为函数参数或将它赋给一个指针，必须定义一个常数指针.

如: `void SendBuff(const char * ptr)`

1-14 const, volatile 类型限定符.

PICC 支持标准 C 的 const, volatile 类型限定符

const 类型限定符通知编译器一个目标含有的常量并且不会改变. 一个常量被放在 ROM 中，显然一个常量是不能被赋值的. 如:

```
const int version = 3;
```

volatile 类型限定符通知编译器，一个目标不能保证在连续的访问中不被改变. 这将禁止编译器对该目标的优化. 所有的 I/O 口及在中断中使用的变量必须有 volatile 类型限定符. 如:

```
volatile unsigned char P_A @ 0x05;
```

对 volatile 目标的访问与对 non-volatile 的访问是不同的，如对 volatile 目标置 1 是先将该目标清 0 后加 1，而对 non-volatile 目标置 1 是先将 1 放在 W 中后再将 W 赋值到目标中.

1-14 特别的类型限定符.

PICC 支持一些特别的类型限定符: persistent, bank1, bank2 及 bank3, 这些限定符不可用于自动变量.

1-14-1 persistent

按 C 的标准，所有的 C 变量在 startup 时被清为 0. 但是在有些情况下，我们希望在复位后仍保持一些变量的值. persistent 类型限定符使被其限定的变量在 startup 时不被清 0，而保留原有的值.

1-14-2 bank1, bank2 及 bank3 类型限定符.

bank1, bank2 及 bank3 类型限定符用于指定变量所在的寄存器页. 如:

```
static bank3 unsigned char fred;
bank3 unsigned char *ptrfred;
```

缺省页是 bank0

1-15 C 语言中的中断处理

在 PICC C 语言中可以用"interrupt"限定符来编写中断服务程序.
一个中断服务函数必须用 interrupt void 来定义, 不能有参数, 并且不能被 C 语言直接调用. 如:

```
long tick_count;
void interrupt tc_int(void)
{
  ++tick_count;
}
```

1-15-1 在中断中保存环境

PIC 单片机硬件只保存 PC, PICC 编译器自动地保存其它可能用到的变量. 但是编译器无法确定 inline 中的汇编语言段使用变量的情况, 你必须自已保护它们.

1-15-2 开启中断

在 PIC.H 中定义了所有的中断位, di()关闭所有中断, ei()打开所有中断. 如:

```
ADIE = 1;  // A/D interrupt will be used
PEIE = 1;  // all peripheral interrupts are enabled
ei();      // enable all interrupts
di();      // disable all interrupts
```

1-16 在 C 中使用汇编语言 #asm, #endasm, asm()

可以在 C 语言中直接使用汇编语言. #asm, #endasm 用来加入一段汇编语言, 而 asm()用来加入一条汇编语言. 如:

```
#include <stdio.h>
unsinged char var;
void main(void)
{
  var = 1;
#asm
  rlf  _var, 1
  rlf  _var, 1
#endasm
  asm("rlf _var, 1");
}
```

1-17 函数调用变换

由于 PIC5X 只有两级堆栈, PICC 编译器使用转移指令来调用函数, 这样被套调用层次增加, 但调用速度下降, 请在需要快速调用的函数前加 fastcall 来指定编译器

直接使用调用指令调用函数.
对于 14 位的 PIC 单片机, 将永远使用调用指令调用.

1-18 MPLAB 使用的调试控制项

-FACKLOCAL 命令用于在 MPLAB 下观察函数内的局部变量

-MPLAB_ICD 命令用于使用 ICD 调试 C 语言.

第二部分

2-1 安装 PICC

将 CD-ROM 装入光驱，自动运行程序将自动启动，如果你已禁止自动运行功能，可以直接运行: cd_drive:\compiler\install.exe

安装程序将指导你完成 PICC 的安装.

2-2 设置伟福集成环境

在伟福集成环境中，将编译器路径指向 PICC 所在目录

将C命令行设置为: -16F877 –G –O –Zg -c

将连接命令行设置为: -16F877 –G –O -Zg

其中: -16F877 为芯片型号

–G –O -c 为源程序调试设置项，不可修改

–Zg 为打开优化

你可以在命令行中加入其它控制项

2-3 调试C语言

在 WAVE\SAMPLES 目录下有一个 PIC C 语言的例子程序: PIC_C.PRJ.

1．打开 PIC_C 项目.

2．编译该项目(F9)

3．用 F7,F8 单步调试例子程序

4．打开观察窗口观察变量