

品腾 8 位 MCU

编写高效 C 程序的注意事项

V1.4



目录

1. 需要了解遵从C规范的C编程和具体芯片汇编编程的区别	1
2. C编程下ROM中的常数数组存储密度只能是汇编编程的一半	2
3. PTCC编译器自动生成了“用户RAM空间清零”代码	2
4. C编程中使用嵌入汇编要特别注意的事项	2
5. 没有MOVC指令及对应寄存器的芯片C编程中使用ROM常数数组的限制	3
6. 快速开发项目的建议	5
7. 尽量避免多个代码量很少的.C源文件参与编译	6
8. 尽量用命名位域代替Bit变量	6
9. 避免使用指针	6
10. 合理使用数据类型	6
11. 编写C语言函数的注意事项	7
12. 优化RAM空间使用的注意事项	7
13. 合理使用位域	7
14. 灵活使用volatile修饰符	8
15. 芯片硬件支持乘除法指令时编程注意事项	8
16. 历史记录	9

1. 需要了解遵从C规范的C编程和具体芯片汇编编程的区别

一、C 语言编程、C 程序编译的基本过程

- 1) C 语言是高级语言，设计 C 语言的目的有两个：1) 提高开发效率（写程序快）；2) 便于移植。
- 2) C 语言作为高级语言，有数据类型、数据结构、程序结构、编译单元、链接目标代码库的规则等等规范定义，这方面知识参考 C 语言规范文本。
- 3) C 语言程序编译为真正的执行代码，通常需要如下处理过程（以 gcc 和 ptcc 为例）：
 - 1、C 编译器的 Driver 程序（比如 gcc 编译系统中是 gcc.exe，PTCC 编译系统中是 ptcc.exe）先调用预处理程序（gcc 编译系统中是 cpp.exe，PTCC 编译系统中是 ptcpp.exe）处理 C 程序中类似“#define、#include”等等伪指令，形成字符流。
 - 2、把预处理生成的 C 程序字符流编译为汇编程序，gcc 编译系统中是调用 cc1.exe 把 C 程序编译汇编程序（.asm 文本文件），PTCC 编译系统就是 ptcc.exe 程序本身做了这个工作。把 C 程序编译生成.asm 程序的过程就是狭义的编译。
 - 3、调用汇编器将上一步生成的.asm 程序编译为可重定位的目标文件（gcc 编译过程中生成的.o 文件，PTCC 编译过程中生成的.o 文件）；目标文件的格式在二进制工具软件中定义，gcc 通常用 GNU 的 BinUtils，PTCC 编译系统用的是 gputils 二进制工具包（里面包括汇编器、链接器、反汇编器、目标文件查看工具、调试信息查看工具、构建库的工具等等）；汇编器就是二进制工具软件中重要的一个程序。
 - 4、调用 BinUtils 中的链接器，将用户的若干个从.c 文件经预处理、编译生成的.o 可重定位目标文件链接生成一个目标文件，同时还要链接芯片相关的预先编译好的.o 文件、库文件.a 文件、crtstart.o 目标文件等。链接生成的目标文件在 PTCC 编译系统中是项目目录下的 obj 目录下的.cof 文件，这个文件用户不用关心。在 PTCC 编译系统调用链接器的过程中，PTCC 编译系统中的链接器就直接生成的 Hex 文件。Hex 文件的主体就是在芯片上执行的代码。用户通常只关心 Hex 文件，Hex 文件是后续烧录到芯片中执行的程序映像（Image）。

项目开发要深入理解一个 C 项目中的 C 源文件如何通过上面的过程最终生成 Hex 文件。另外，上述编译过程中会生成很多中间文件，比如 C 程序调试信息文件（.cdb 文件）、完成存储分配后的地址映射文件（.map 文件）、编译过程中不同阶段传递报错信息的文件，还有其他很多中间文件。用户通常不必关心这些编译过程中产生的中间文件。

二、PTCC 编译系统支持的汇编编程特点

PTCC 编译系统中的汇编器，支持汇编语言编程。支持的汇编编程模式是：单文件绝对地址模式。

虽然，在 PTIDE 下创建汇编工程项目，可以编写多个.asm 汇编文件，但是实际上汇编器会在预处理阶段就把多个.asm 文件装配成一个.asm 文件（通过处理#include 伪指令），然后汇编器再扫描最终的单个.asm 文件汇编生成最终的 Hex 文件。

注意，这种情况下的汇编编程是“绝对地址模式”，和上一节提到的汇编器将一个.asm 文件汇编为可重定位的.o 目标文件完全不同，有些伪指令仅能支持其中一种汇编模式。那么，在此汇编编程

模式下，生成执行码的 Code Size 完全由用户编写的汇编指令决定，甚至用户可以通过非常精细的汇编编程将整个 ROM 用满。

2. C编程下ROM中的常数数组存储密度只能是汇编编程的一半

在绝对地址汇编编程下，一个 ROM 地址（对应一个 ROM 字）中的 2 个字节可以用 dw 伪指令填满数据，然后用 MOVC 指令将 ROM 字中的 2 个字节分别读取到 CPU 内核寄存器中。

但是，对于遵从 C 规范的 PTCC C 编程，用形如“const char romchar[16]; 或 const int romint[16];”在 C 编程下声明 ROM 中的常数数组，每个 ROM 字（芯片的 ROM 字含 2 个字节）中，只有低 8 位的一个字节可以存储有效数据。

这是遵从 C 编程规范的必然结果。

所有支持类似 PIC 架构的 8 位处理器的通用 C 编译器都如此。

3. PTCC编译器自动生成了“用户RAM空间清零”代码

PTCC 编译器在编译 C 程序的过程中，自动生“用户 RAM 空间清零”的代码，并且这段代码在项目的 main 函数之前就执行了。用户千万不要自己再写“用户 RAM 空间清零”的 C 代码或者“嵌入汇编段”，一方面完全没必要，另一方面程序调试的时候执行“用户 RAM 空间清零”代码会耗时较长。

4. C编程中使用嵌入汇编要特别注意的事项

1) 要特别注意内嵌汇编中对宏定义符号的使用

在 C 编程时 C 编程的头文件（如 pt8p3103.h）中通常对芯片的一些特殊的标志位或标志寄存器进行了宏定义（#define），这种宏定义在 C 程序编译过程的预处理阶段就进行了符号替换。那么对于内嵌汇编程序段，在 C 程序的预处理阶段也把这种内嵌汇编程序段当做 C 程序的字符流处理，也要进行宏定义的符号替换，这一点要特别注意。因此在 C 内嵌汇编的汇编程序段中，要用芯片对应的汇编编程头文件（如 pt8p3103.inc）中定义的内容部分。

比如，Z 标志位和 C 标志位，在 C 编程的头文件中，它们被定义为字节变量中的位域，在 pt8p3103.h 中定义如下：

```
#define C STATUSbits._C
#define DC STATUSbits._DC
#define Z STATUSbits._Z
```

而 STATUSbits 是 C 编程空间中的符号，在汇编编程空间中并不存在。

同样的，Z 标志位和 C 标志位，在汇编编程的头文件中，如 pt8p3103.inc 中定义如下：

```
#DEFINE Z STATUS,2
#DEFINE DC STATUS,1
#DEFINE C STATUS,0
```

那么在 C 编程的嵌入汇编的程序段中，就不能用 Z、C、DC 等符号。需要用它们被定义的内容。

例如下面的嵌入汇编程序段：

```
__asm
...
    BTSZ    Z
...
__endasm
```

应该写成::::

```
__asm
...
    BTSZ    STATUS,2
...
__endasm
```

在 C 编程和汇编编程两个不同的层次中，都可能会存在很多编程者为方便编程而编写的宏定义，需要用户了解编译处理的过程，编程者要仔细区分，这方面 C 编译器、汇编器都不可能提供完备的保证和帮助。

2) 特别注意 C 编程中内嵌汇编程序段中的 RET 指令

C 编程中的内嵌汇编程序片段，C 编译器就把此汇编段作为“字符流”直接输出到.asm 汇编程序中，当然要进行上面一小节所提到的“宏定义替换”。如果 C 程序的内嵌汇编片段中有 RET 指令，那么在此汇编段之后的 C 程序语句，可能根本就不会执行。调试的时候要特别注意这种情况，否则容易对程序的执行流程产生困惑。

3) C 编程中内嵌汇编程序段在调试模式下整体作为 C 程序的一步，汇编段中的符号在 C 程序调试中不可见

在 PTCC 的 C 编程中，用 __asm 和 __endasm 括起来的多行汇编，虽然在程序源代码上占据了多行，但是 C 编译器把这个汇编块当做 1 行 C 语句处理，里面的所有内容都在进行了 C 预处理阶段的宏定义替换后，作为一个字符串输出到 C 编译出来的.asm 汇编文件中。所有的 C 编译器处理 C 编程中内嵌汇编都是这样处理的。另外，嵌入汇编段中的符号，在 C 程序调试中是不可见的，因为 C 程序和汇编程序段处于不同的“符号名空间”。

总之，要理解 C 编程中内嵌汇编的逻辑，在 C 编程中灵活高效使用内嵌汇编。

5. 没有 MOVC 指令及对应寄存器的芯片 C 编程中使用 ROM 常数数组的限制

对于没有访问 ROM 中常数数据的 MOVC 指令（也没有相应的 EADRH、EADRL、EDATH 寄存器）的芯片，如 PT8P1101、PT8P2104，如果要用形如“const char romchar[16]; 或 const int romint[16];”在 ROM 中存放数据，那么就只能按照芯片规格书中说明的用“ADDR A, PCL”指令跳转到数据在 ROM 中的地址，该地址是一条“RETK data”指令，此指令将 ROM 中的数据 data 读取到寄存器 A 中并返回。

对于没有 MOVC 指令的芯片，如 PT8P1101、PT8P2104，C 编程时候在 ROM 中存储常数数组有下面的限制：

1) const 数组的声明必须在全局变量空间，不能在函数内部声明 const 数组。

通常，都在全局变量空间声明常数数据，PTCC 编译器将常数数据存放在 ROM 里，这些常数数

据用于程序控制逻辑中使用的常量。这里提醒一下的原因是，在自由格式的 C 编程情况下，把常数数组声明在用户函数内也是符合 C 规范的，但是 **PTCC 编译器要求 ROM 中的常数数组必须声明在全局变量空间。**

下面用一段程序，说明 ROM 中常数数组声明和编译出来的汇编程序的情况。

```
volatile unsigned char romitem;
const unsigned char rom_data[4] = {1, 2, 3, 4};
void PWM_Test()
{
    romitem = rom_data[2];
    romitem = rom_data[3];
    PWM_INIT();
    PWM1_DUTY(romitem);
    while(1)
    {

    }
}
```

上面程序中的 **const** 数组，编译后的汇编程序段如下：

```
;Now output GSINIT (CODE) segment
G@rom_data@0@0:
ptc_rom_data      code
ptc_rom_data:
    addr A, PCL
    db 0x01 ; 1
    db 0x02 ; 2
    db 0x03 ; 3
    db 0x04 ; 4
;   area      CABS      (ABS,CODE)
```

对于上面的汇编程序段，进一步说明如下：

- 1) **const** 数组声明的时候没有指定地址，那么 **ptc_rom_data** 的地址就由 C 编译系统中的**链接器**最终分配地址。
- 2) 汇编段中的 **db** 伪指令将数据存放到 ROM 区，C 编译系统中的**汇编器**会根据芯片型号判断出芯片的类型（无 **MOVC** 指令及对应的内核寄存器），对此种情况，将“**db 0xkk**”伪指令构造成一个和汇编指令“**RETK 0xkk**”相同的指令码，存储到 ROM 中。

访问数组元素的 C 程序段编译为如下的汇编段：

```
; -----  
; function PWM_Test  
; -----  
ptc_PWM_Test code  
ptc_PWM_Test:  
G@PWM_Test@0@0:  
C@APP.c@13@0@7:  
C@APP.c@16@1@7:  
movk 0x2  
call ptc_rom_data  
mov ptc_romitem,A  
C@APP.c@17@1@7:  
movk 0x3  
call ptc_rom_data  
mov ptc_romitem,A  
C@APP.c@18@1@7:  
call ptc_PWM_INIT  
C@APP.c@19@1@7:  
mov A,ptc_romitem  
mov ptc_PWM1_DUTY_PARM_1,A  
call ptc_PWM1_DUTY  
t_00102:  
C@APP.c@21@1@7:  
jmp t_00102  
t_00104:  
C@APP.c@25@1@7:  
XG@PWM_Test@0@0:  
ret
```

从上述编译出的汇编段，用户可以了解在没有 MOVC 指令的芯片上的 C 编程，对 ROM 中常数数组数据的访问，是编译生成了 call 指令，call 指令的地址就是数组在 ROM 区中的地址，此 ROM 区中的第一项是“ADDR A, PCL”指令的二进制码。当然，访问 ROM 数组数据之前，需要预先把数组索引值放到 A 寄存器中。

6. 快速开发项目的建议

PTCC 编译系统支持用户使用 C 或汇编进行项目开发。

- 1) 用 C 开发，如果编译后报如下图所示的错误时，要注意观察此项目需要的 ROM 大小，如果超过芯片的 ROM 空间很多，那么建议立即选择用汇编语言开发。


```
10:42:45 编译结果:  
This project need 2029 words ROM!!!  
error: No target ROM memory available!!!
```

```
10:42:45 错误: 编译失败, 未生成HEX文件!
```

- 2) 如果超过芯片 ROM 不是很多, 编程者要根据高效 C 编程的建议改写程序, 也可以在影响 ROM 使用率的程序部分使用内嵌汇编。各种芯片各种编译系统的 C 编程都支持嵌入汇编, 有些性能关键、Code Size 关键的编程片段, 使用内嵌汇编会得到很大的收益。
- 3) 如果超过芯片 ROM 很多, 用汇编编程来开发项目, 也需要编程者精心排布指令, 减少 ROM 使用。
- 4) 如果用汇编编程最终 Code Size 仍超过 ROM 大小, 那么只能换芯片。

7. 尽量避免多个代码量很少的.C源文件参与编译

使用 PT-IDE2 创建面向品腾 8 位处理器工程的时候, IDE 会缺省创建一个 main.c 文件, 该文件中提供了 main 函数和中断处理函数的框架, 用户可以基于此框架编写自己的工程源文件。用户可以向工程中添加其他 C 源程序文件。但是, 用户应该尽量避免使用多个代码量很少的 C 源文件, 这样可以减少编译时间, 也可以给编译优化提供更多的机会。

8. 尽量用命名位域代替Bit变量

目前, 用 bit 变量编译器的优化效果不高, 编译器目前不能做到将定义在多个文件中的 Bit 变量合并到一个字节变量中来节省 RAM 空间的使用。对于单片机编程中常用到的作为标志位的位变量, 建议按《C 语言程序设计指南》中的位域变量定义和使用方法来实现。

9. 避免使用指针

品腾 8 位 MCU 的 C 编程中, 一个指针变量要占用 3 个字节 RAM 空间, 请尽量避免使用指针, 可以用数组实现同样的功能。

10. 合理使用数据类型

PTCC 编译系统仅面向品腾 8 位 MCU 芯片。此类 8 位芯片的工作寄存器只有 8 位。PTCC 支持 bit(1 位)、char (8 位)、int (16 位) 三种基本数据类型。

对于 bit 类型, 都是无符号数。

对于 char 类型, 缺省为无符号的 char 类型和声明为“unsigned char”相同。若要使用有符号的 char 类型, 需要声明为“signed char”。

对于 int 类型, 缺省为有符号的 int 类型和声明为“signed int”相同。若要使用无符号的 int 类型, 需要声明为“unsigned int”。

例如:

```
const char aa;
```

```
#define aa 1u; //这个 1u 是无符号数;
```


特别注意：尽量避免使用 `int` 类型。

11. 编写C语言函数的注意事项

品腾 8 位 MCU，ROM 和 RAM 空间都很有限，硬件支持函数间互相调用的硬件堆栈的级数也非常有限。PTCC 不支持函数的递归调用，而且要特别注意：**C 函数都是不可重入的**。

（1）功能简单的操作不要使用函数

函数调用至少会额外生成调用和返回指令，函数调用也会带来编译优化的困难。因此，尽量避免写大量代码量很小的小函数；并且，函数代码量大能为编译优化提供更大的上下文范围。

（2）中断函数的编写注意事项

中断函数尽可能不要再调用其他函数，如果要调用其他函数，要保证只有中断函数调用该函数。由于函数不可重入，如果中断函数和主函数都调用了相同的函数，那么会引发正确性问题，难以调试定位。

（3）限制函数的形参数量和形参类型

对于必须使用形参传递数据的函数，函数形参的数量也要尽可能限制。注意：**位域类型不能作为函数形参**。

（4）使用 `inline` 关键字修饰函数

对于在整个工程中仅被调用 1 次的函数，可以使用 `inline` 关键字修饰函数，编译器会将调用函数的函数体内嵌到调用此函数的函数中，可减少函数调用的开销。但是，要注意的一点是：被 `inline` 的函数，其中不能再设置调试断点了。

12. 优化RAM空间使用的注意事项

品腾 8 位 MCU 的 RAM 容量非常有限，并且这些 8 位 MCU 不能运行操作系统，不支持程序运行过程中 RAM 空间的释放和再分配。因此，用户编写程序，必须对 RAM 的使用精打细算。

（1）尽可能重用变量

如果不同的函数使用存储在 RAM 中的变量保存中间计算结果，那么要尽量重用这些保存中间结果的变量。当程序编译、链接过程中报 RAM 空间不足的时候，要重新分析整个程序的编写过程，优化 RAM 空间的使用。编译和链接过程中会进行 RAM 空间使用的优化，但是难以做到最优。

（2）尽可能使用直接寻址

对几乎所有的处理器而言，直接寻址肯定比间接寻址更高效。因此，用户编程应尽可能使用直接寻址方式，避免使用间接寻址。

（3）只读数据用 ROM 空间存储

对于程序中值不变的变量，建议加上“`const`”修饰符，PTCC 编译器会将 `const` 变量分配到 ROM 中，可以减少 RAM 资源的使用。当然，芯片的 ROM 空间也很有限，需要用户在程序编写时进行权衡。

13. 合理使用位域

品腾 8 位 MCU，PTCC 对于位域的宽度限制是 1~8。用户使用位域时应注意以下几点：

（1）默认使用 `unsigned char` 类型

在使用位域时，用户指定的类型将被忽略，由编译器默认为 `unsigned char` 类型。例如，`int a:3;` 这里的 `int` 将被忽略。

（2）避免使用宽度为 8 的位域

宽度为 8 的位域是没有实际意义的，可以直接声明为 `unsigned char` 类型。

（3）合理分布位域，避免空洞

在 PTCC 编译器中，位域是按声明的先后顺序依次分布的，并且一个位域是不跨字节存储的。因此，如果当前字节所剩余的位不足以容纳待分配的位域时，编译器将跳过当前字节，而从下一字节第 0 位开始分配。在这种情况下，就出现了位域的空洞。

14. 灵活使用 `volatile` 修饰符

若非必要，建议用户尽可能避免用“`volatile`”和“`static`”修饰符，这类修饰符不利于编译优化实施。如果用户希望运用调试工具监视某一局部变量时，建议在该局部变量声明处加上“`volatile`”修饰符，否则，用户可能无法实时监视局部变量的变化情况。建议用户在程序的正确性调试阶段使用 `volatile` 限定符，有利于追踪变量值的变化情况；调试正确后，删掉 `volatile` 限定符，由编译器进行更多的程序优化。

15. 芯片硬件支持乘除法指令时编程注意事项

（1）硬件是否支持乘除法指令调用的库不同。

5) 对于芯片硬件仅支持加减移位指令的 MCU，变量的乘法、除法、取模等运算，通过调用编译系统提供的整数库 `libptcc.a` 实现。

6) 对于芯片硬件支持 8 位数据的乘/除法指令的 MCU，那么 C 程序中单字节的数据和变量的乘/除运算，C 编译器会直接生成乘/除法汇编指令，对于超过 8 位的变量的乘除法等运算，通过调用编译系统提供的整数库 `libptcc_muldiv.a` 实现。

这些库都在集成开发环境 PT-IDE 软件包中，使用 IDE 时，在编译过程中根据芯片型号自动链接对应的库，不用用户手动设置。

（2）用户要特别注意除数可能为 0 的情况

支持乘除法指令的 8 位 MCU，为了防止由于除以 0 造成溢出，MCU 在处理器状态标志寄存器中定义了 OV 位，当除法运算中除数为 0 则 OV 位为 1，反之则 OV 位为 0。

编程使用除法指令时，无论是 C 程序中编写除法表达式还是汇编程序中直接编写汇编除法指令，都要特别注意除数是否为 0。如果用户能确定在调用除法运算的时候除数肯定不是 0，那么不必做特殊处理。但是，如果用户不能确定调用除法时除数是否为 0，用户编程要在调用除法后立即判断 OV 位的值来决定后续的处理，如果 OV 为 1 进入死循环状态，否则进行用户程序的正常逻辑。这样可以防止出现由于除法结果溢出造成程序错误难以定位的情况。

16. 历史记录

版本号	修改记录	发布日期
V1.0	初版	2024-03-18
V1.1	重新修订整个文档	2024-05-15
V1.2	重新修订整个文档	2024-09-26
V1.3	增加硬件支持乘除法指令时编程注意事项；修改格式。	2025-03-03
V1.4	增加如下小节： 1、需要了解遵从C规范的C编程和具体芯片汇编编程的区别 2、C编程下ROM中的常数数组存储密度只能是汇编编程的一半 3、PTCC编译器自动生成了“用户RAM空间清零”代码 4、C编程中使用嵌入汇编要特别注意的事项 5、没有MOVC指令及对应寄存器的芯片C编程中使用ROM常数数组的限制 6、快速开发项目的建议	2025-05-16