PMBus™ 协议栈
用户指南

请注意以下有关 Microchip 器件代码保护功能的要点：

- Microchip 的产品均达到 Microchip 数据手册中所述的技术指标。
- Microchip 确信：在正常使用的情况下， Microchip 系列产品是当今市场上同类产品中最安全的产品之一。
- 目前，仍存在着恶意、甚至是非法破坏代码保护功能的行为。就我们所知，所有这些行为都不是以 Microchip 数据手册中规定的操作规范来使用 Microchip 产品的。这样做的人极可能侵犯了知识产权。
- Microchip 愿与那些注重代码完整性的客户合作。
- Microchip 或任何其他半导体厂商均无法保证其代码的安全性。代码保护并不意味着我们保证产品是 “牢不可破”的。

代码保护功能处于持续发展中。 Microchip 承诺将不断改进产品的代码保护功能。任何试图破坏 Microchip 代码保护功能的行为均可视为违反了《数字器件千年版权法案 （Digital Millennium Copyright Act）》。如果这种行为导致他人在未经授权的情况下，能访问您的软件或其他受版权保护的成果，您有权依据该法案提起诉讼，从而制止这种行为。

QUALITY MANAGEMENT SYSTEM
CERTIFIED BY DNV
ISO/TS 16949:2002

Microchip 位于美国亚利桑那州 Chandler 和 Tempe 与位于俄勒冈州 Gresham 的全球总部、设计和晶圆生产厂及位于美国加利福尼亚州和印度的设计中心均通过了 ISO/TS-16949:2002 认证。公司在 PIC® MCU 与 dsPIC® DSC、KEELOQ® 跳码器件、串行 EEPROM、单片机外设、非易失性存储器和模拟产品方面的质量体系流程均符合 ISO/TS-16949:2002。此外，Microchip 在开发系统的设计和生产方面的质量体系也已通过了 ISO 9001:2000 认证。

目录

前言

客户须知

所有文档均会过时，本文档也不例外。 Microchip 的工具和文档将不断演变以满足客户的需求，因此实际使用中有些对话框和 / 或工具说明可能与本文档所述之内容有所不同。请访问我们的网站（www.microchip.com）获取最新文档。

文档均标记有 “DS”编号。该编号出现在每页底部的页码之前。 DS 编号的命名约定为 “DSXXXXXA”，其中 “XXXXX”为文档编号，“A”为文档版本。

欲了解开发工具的最新信息，请参考 MPLAB® IDE 在线帮助。从 Help （帮助）菜单选择 Topics （主题），打开现有在线帮助文件列表。

简介

本章包含使用 44 引脚演示板前需要了解的有用的一般信息。内容包括：

- 文档编排
- 本指南使用的约定
- 推荐读物
- Microchip 网站
- 开发系统变更通知客户服务
- 客户支持
- 文档版本历史

文档编排

本文档介绍了如何使用 44 引脚演示板开发工具在目标电路板上仿真和调试固件。本手册的内容编排如下：

- **第 1 章 “概述”** – 简要介绍了 PMBus 协议栈并概述了所需的单片机资源以及协议栈识别和处理的总线协议。
- **第 2 章 “协议栈限制”** – 说明了 PMBus 协议栈在故障管理、寻址、检测、控制线和分页方面的限制。
- **第 3 章 “PMBus 协议——协议栈如何对其进行处理”** – 详述了主机能在总线上发起的三类事务 （写、读和读 - 写）。
- **第 4 章 “数据输入 / 输出变量”** – 说明了如何通过 PMBus 命令在主机和从设备之间传递数据。

- **第 5 章 “协议栈功能”** – 说明了 I²C™ 模块能够识别和作为中断源处理的状态。
- **第 6 章 “器件故障管理”** – 提供有助于用户实现处理 PMBus 系统中警告和故障的统一机制的信息。
- **第 7 章 “实现协议栈功能所需的用户数据”** – 列出了一组用户定义的要求协议栈遵循的变量和命令。
- **附录 A “由用户自定义的标签列表”** – 列出了需要用户定义的标签和变量。
- **附录 B “协议栈测试”** – 说明了针对本实现方案如何测试协议栈。
- **附录 C “定义警告和故障响应函数”** – 列出了需要针对警告和故障定义的函数。
- **附录 D “PEC 字节计算”** – 说明了如何为每类事务计算 PEC 字节。
- **附录 E “命令表解析”** – 讨论结构数组应多大才能使协议栈在合理的时间内对其进行解析。
- **附录 F “缓冲器结构”** – 提供有关缓冲器包含和处理何种数据的信息。

本指南使用的约定

本手册采用以下文档约定：

文档约定

说明	涵义	示例
Arial 字体：		
斜体字	参考书目	*MPLAB® IDE User's Guide*
	需强调的文字	*... 仅有*的编译器 ...
首字母大写	窗口	Output 窗口
	对话框	Settings 对话框
	菜单选项	选择 Enable Programmer
引用	窗口或对话框中的字段名	“Save project before build”
带右尖括号且有下划线的斜体文字	菜单路径	*File>Save*
粗体字	对话框按钮	单击 **OK**
	选项卡	单击 **Power** 选项卡
‘b*nnnn*	二进制数， *n* 是一个数字	‘b00100, ‘b10
尖括号 < > 括起的文字	键盘上的按键	按 <Enter>，<F1>
Courier 字体：		
常规 Courier	源代码示例	`#define START`
	文件名	`autoexec.bat`
	文件路径	`c:\mcc18\h`
	关键字	`_asm, _endasm, static`
	命令行选项	`-Opa+, -Opa-`
	位值	`0, 1`
	常数	`0xFF, ‘A’`
斜体 Courier	可变参数	*file*.o，其中 *file* 可以是任一有效文件名
0x*nnnn*	十六进制数， *n* 是一个十六进制数字	0xFFFF，0x007A
方括号 **[ ]**	可选参数	`mcc18 [options] file [options]`
花括号和竖线：**{ | }**	选择互斥参数； “或”选择	`errorlevel {0|1}`
省略号 ...	代替重复文字	`var_name [, var_name...]`
	表示由用户提供的代码	`void main (void) { ... }`

推荐读物

本用户指南介绍了如何使用 PMBus 协议栈。下面列出了其他有用的文档。以下文档均已提供，并建议读者作为补充参考材料。

PMBus™ Power System Management Protocol Specification – Part I – Revision 1.1 （http://pmbus.org/）。

本文档说明了 PMBus 协议的一般要求以及传输和电子层。

PMBus™ Power System Management Protocol Specification – Part II – Revision 1.1 （http://pmbus.org/）。

本文档说明了 PMBus 协议的命令语言。

System Management Bus (SMBus™) Specification Version 2.0 （http://smbus.org/specs/）。

本文档是 SMBus 接口的官方规范。

Using the PIC® Devices' SSP and MSSP Modules for Slave I²C™ Communication（AN734）– http://www.microchip.com

本应用笔记总结了 SSP 模块在 I²C 通信中作为从机的用法。

MICROCHIP 网站

Microchip 网站 （www.microchip.com）为客户提供在线支持。客户可通过该网站方便地获取文件和信息。只要使用常用的因特网浏览器即可访问。网站提供以下信息：

- **产品支持**——数据手册和勘误表、应用笔记和样本程序、设计资源、用户指南以及硬件支持文档、最新的软件版本以及存档软件
- **一般技术支持**——常见问题 （FAQ）、技术支持请求、在线讨论组以及 Microchip 顾问计划成员名单
- **Microchip 业务**——产品选型和订购指南、最新 Microchip 新闻稿、研讨会和活动安排表、 Microchip 销售办事处、代理商以及工厂代表列表

开发系统变更通知客户服务

Microchip 的客户通知服务有助于客户了解 Microchip 产品的最新信息。注册客户可在他们感兴趣的某个产品系列或开发工具发生变更、更新、发布新版本或勘误表时，收到电子邮件通知。

欲注册，请登录 Microchip 网站 www.microchip.com，点击 “变更通知客户（Customer Change Notification）”服务并按照注册说明完成注册。

开发系统产品的分类如下：

- **编译器**——Microchip C 编译器及其他语言工具的最新信息，包括 MPLAB C18 和 MPLAB C30 C 编译器、 MPASM™ 和 MPLAB ASM30 汇编器、 MPLINK™ 和 MPLAB LINK30 目标链接器，以及 MPLIB™ 和 MPLAB LIB30 目标库管理器。
- **仿真器**——Microchip 在线仿真器的最新信息，包括 MPLAB ICE 2000 和 MPLAB ICE 4000。
- **在线调试器**——Microchip 在线调试器 MPLAB ICD 2 的最新信息。
- **MPLAB® IDE**——关于开发系统工具的Windows®集成开发环境Microchip MPLAB IDE 的最新信息，主要针对 MPLAB IDE、 MPLAB SIM 模拟器、 MPLAB IDE 项目管理器以及一般编辑和调试功能。
- **编程器**——Microchip 编程器的最新信息，包括 MPLAB PM3 和 PRO MATE® II 器件编程器以及 PICSTART® Plus 和 PICkit™ 1 开发编程器

客户支持

Microchip 产品的用户可通过以下渠道获得帮助：

- 代理商或代表
- 当地销售办事处
- 应用工程师 （FAE）
- 技术支持

客户应联系其代理商、代表或应用工程师 （FAE）寻求支持。当地销售办事处也可为客户提供帮助。本文档后附有销售办事处的联系方式。

也可通过 http://support.microchip.com 获得网上技术支持。

文档版本历史

版本 A （2008 年 12 月）

- 本文档的初始版本。

注：

第 1 章 概述

1.1 PMBus 概述

PMBus 是一种开放式标准协议，业界将其定义为一种与电源转换器和其他设备进行通信的方式，从而创立了世界上第一个有关对电源设备和系统进行数字控制方面的开放通信标准。

PMBus 是系统管理总线 （System Management Bus， SMBus™）的一个扩展集，是一个工业标准串行通信接口。它并未采用所有的 SMBus 协议，只是采用了其中的一部分，还增加了大量新的功能。

1.2 什么是协议栈？

Microchip的PMBus协议栈1.0版针对PIC16F88X系列中档PIC®单片机的传统I^2C™通信接口采用了 PMBus 协议。该协议栈于 2007 年 2 月 5 日发行了 PMBus 1.1 版。

从低层上说，协议栈实际上是在 PMBus 环境中充当从器件的单片机中 I^2C 中断的中断服务程序 （ISR）处理者。它表示一个代码块，意在执行协议中的传输链路层。

必要的单片机资源如下：

- 将 MSSP （SSP 模块）模块配置成在标准速度模式 （100 kHz）下运行的具有 7 位地址的从模式，并且能够在遇到启动 / 停止位时产生中断。
- 采用 T0CS 时钟源的 Timer0 在每个总线时钟脉冲的上升沿递增 1。因此，相应的 PIC 器件的 T0CKI 引脚 （例如， PIC16F886 器件的 RA4 引脚）应始终保持与 SCL 引脚 （例如， PIC16F886 器件的 RC3 引脚）在外部相连。
- 内部振荡器配置成 8 MHz。
- 协议栈和 42 条定义的 PMBus 命令共占用 123 字节的数据存储器和 1.5 KB 的程序存储器。

该协议栈识别和处理以下 PMBus 协议：SEND_BYTE、READ_BYTE、WRITE_BYTE、READ_WORD、 WRITE_WORD、 READ_BLOCK、 WRITE_BLOCK 和以下扩展协议：BLOCK_WRITE_BLOCK_READ_PROCESS_CALL 和 GROUP_COMMAND_PROTOCOL。对于所有这些协议，协议栈认为 PEC 字节是内隐的。 PEC 是一个 CRC-8 的错误校验字节，是根据所有的报文字节 （包括地址和读 / 写位）计算得到的。 PEC 由提供最新数据字节的器件添加到报文中。

信息以非常简捷的方式传递给主应用程序或从主应用程序中传出。

任何发送到器件中的命令都会采用上面提到的一种 PMBus 协议。协议栈处理这种事务，将信息传递给主应用程序或从主应用程序中取回。从而，我们得出这样的结论，协议栈与主应用程序之间紧密合作，它会向主应用程序传送较新数据，或要求应用程序为主机准备数据。

注：

第 2 章 协议栈限制

2.1 故障管理

在发生警告或故障事件之后， PMBus 器件可以通过设置状态寄存器中的警告 / 故障状态位，来向主机通报其状态。而其他直接向主机通报的方法，如通过 SMBAlert 线或“主机通报协议”的方法，在该版本中没有实现。

此外，协议栈无法检测到由下列情况引起的故障：

- 数据超出范围
- 无效或不受支持的数据

主应用程序负责修改相应的状态寄存器以便主机能够察觉这种故障。本手册以及本协议栈随附的演示程序中都包含了有关这方面的代码实例。

2.2 寻址

由于受 PIC16F886/887 中 I^2C™ 寻址能力的限制，本协议栈仅允许器件对在总线上发送的 7 位从地址进行响应。

2.3 超时检测

本协议栈不支持超时检测。

2.4 控制线

这一额外的 PMBus 线的功能受用户程序支持。

2.5 分页

本协议栈不支持 PAGE 命令。

注：

第 3 章 PMBus 协议——协议栈如何对其进行处理

3.1 事务类型

主机能够在总线上发起三种类型的事务：写、读和联合写读协议。由于协议栈所支持的协议具有很多相同的处理方式，因此，我们将逐一介绍这三类事务，并说明各类事务所对应的支持协议。

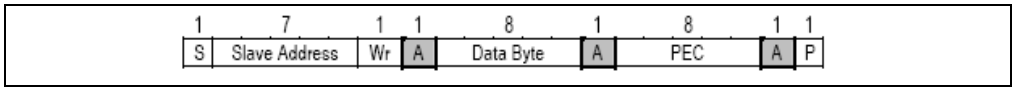
3.2 写事务

此类事务是指主机可以发送所有的数据，而从器件可以应答 / 不应答所接收的数据包。所有的写协议都始于正要发送给从器件的地址，其含有一个数值为 0 的 RW 位（bit[0]）。如果硬件已经应答了具有所定义地址的从器件，那么就会进入主机写地址（Master Writes Address， MWA）状态，然后操作继续执行。含有命令代码的主机写数据 （MWD）状态必须紧随其后。

所有的写协议都始于正要发送给从器件的地址，其含有一个数值为 0 的 RW 位（bit[0]），使协议栈进入 MWA 状态。接下来仅由主机发送的字节是命令代码、要发送的字节的数目 （如适用）、数据字节本身 （如适用）和可选的 PEC 字节。协议栈对每一个字节进行应答，但要检查通信的有效性或数据内容。

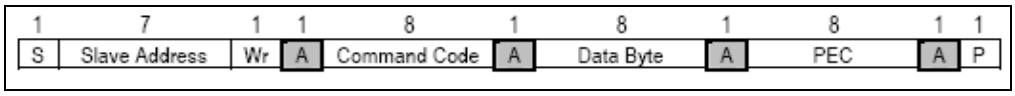
每一个受支持的写协议的格式如下 （注意 PEC 字节是可选的）：

3.2.1 发送字节



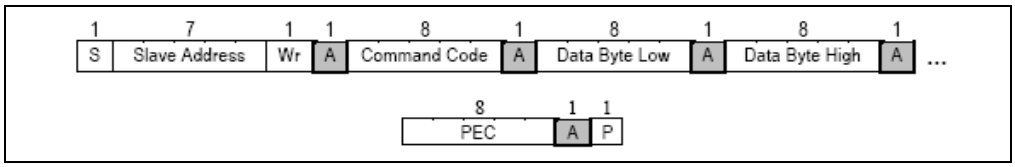
只可以发送从器件的地址和一个数据字节，即命令代码。命令示例采用这种协议：CLEAR_FAULTS。

3.2.2 按字节写



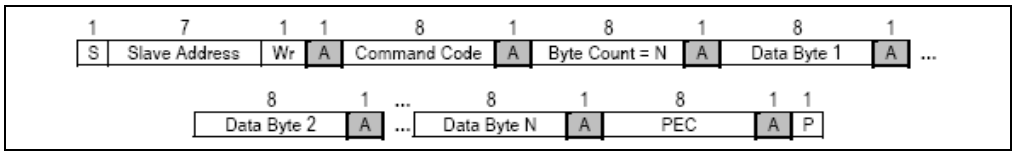
除地址和命令代码外，主机还可以发送一个数据字节。命令示例采用这种协议：VOUT_MODE。

3.2.3 按字写



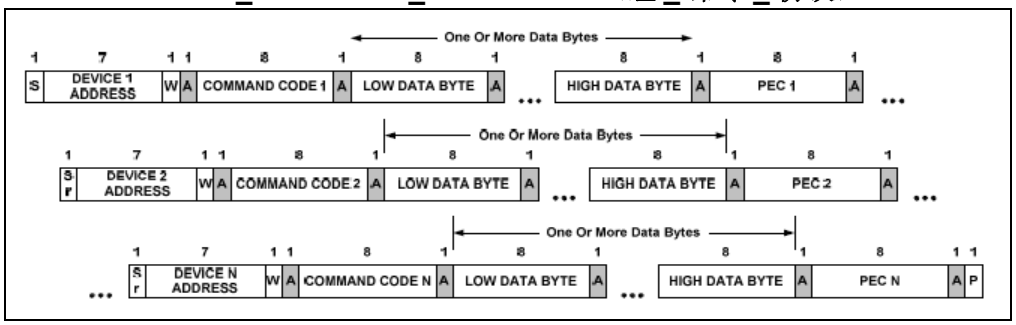
在这种协议下，主机可以发送两个数据字节。命令示例采用这种协议：VOUT_COMMAND。

3.2.4 块写



除地址和命令代码外，主机负责发送一个字节，其表示将要发送的数据字节的实际数目和数据字节本身。示例命令采用这样的协议：MFR_ID。

3.2.5 GROUP_COMMAND_PROTOCOL（组 _ 命令 _ 协议）

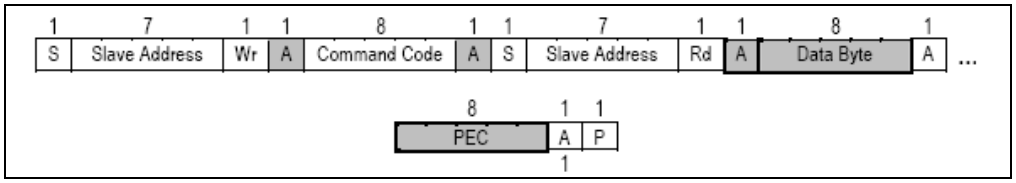


主机可以通过采用本扩展的 PMBus 协议来向多个 PMBus 器件发送命令。这些命令以连续传输的方式发送。当各器件检测到用于结束发送命令的 “停止” 条件时，它们都开始执行所接收到的命令。从协议栈的角度来看，这只不过是一个简单的 WRITE_BYTE/WORD/BLOCK（写 _ 字节 / 字 / 块）协议。唯一的差别是，主机队列中的所有器件均写入之后才开始接收 “停止” 条件；因此，仅在那时才调用应用函数来处理数据。

3.3 读事务

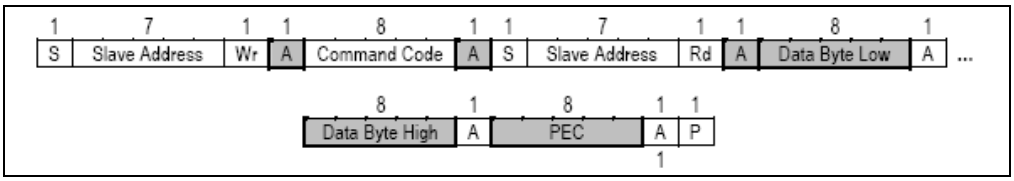
在这种 PMBus 处理中，主机和从器件都在总线上放置数据，都可以对所接收到的字节进行应答 / 不应答。读事务的第一部分与写事务类似，即主机必须发送从器件地址（含有一个数值为零的 RW 位），之后再发出命令代码。为了使事务变成一个读事务，主机必须产生一个 “重新启动” 条件，再发送从器件地址 （RW 位置 1）。从而协议栈识别出主机想从从器件中读取数据，因此，将数据字节放置在总线上，同时计算整个事务的 PEC 字节，再将该字节添加到传输列的末尾。

3.3.1 字节读操作



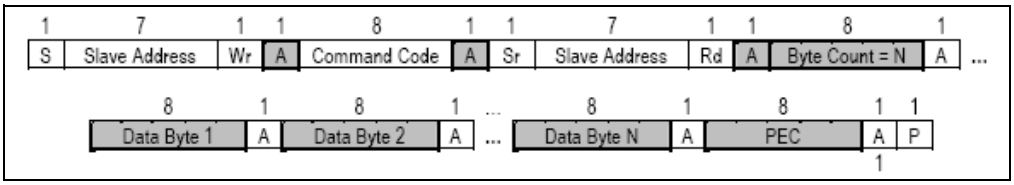
主机采用该协议来取回一个数据字节。命令示例采用这样的协议：STATUS_VOUT 和 STATUS_IOUT。

3.3.2 字读操作



主机采用该协议来取回两个数据字节。命令示例采用这样的协议：STATUS_WORD。

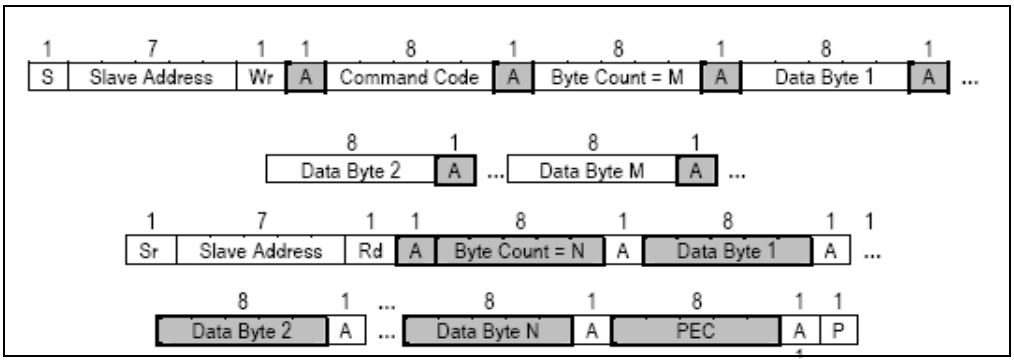
3.3.3 块读操作



主机采用该协议来取回一个属于命令的数据块。在将数据字节放到总线上之前，协议栈必须先发送字节的数目。命令示例采用的协议为：MFR_MODEL。

3.4 写 / 读事务

在 1.1 版中提到的 PMBus 协议，引入了一个扩展协议，即 process call（过程调用）。该扩展协议名为 Block Write/Block Read Process（块写 / 块读过程调用）。



这是一个联合的块写 / 块读协议。主机和从器件都需要参与事务，并把数据放在总线上。当主机需要得到从器件的信息，但该信息必须根据主机发送的输入数据字节来处理时，要用到这种协议。

对于采用这种协议发出的命令，等效传递函数可以定义为 Y=F(X) 的形式，其中 X 表示由主机发送的数据， Y 表示用各个命令的函数 F 计算出的从器件输出数据。

注意，在协议的读部分，从器件地址和命令代码丢失了。命令示例采用协议：QUERY。该命令用于询问 PMBus 器件是否支持所给命令，如果支持，命令的数据格式是什么。该命令使用上述的 Block Write-Block Read Process Call。对于过程调用的写部分，数据字节是一个无符号的二进制整数，其值等于所询问命令的命令代码。对于过程调用的读部分，数据字节是一个无符号的二进制整数，其对从器件应答主机的询

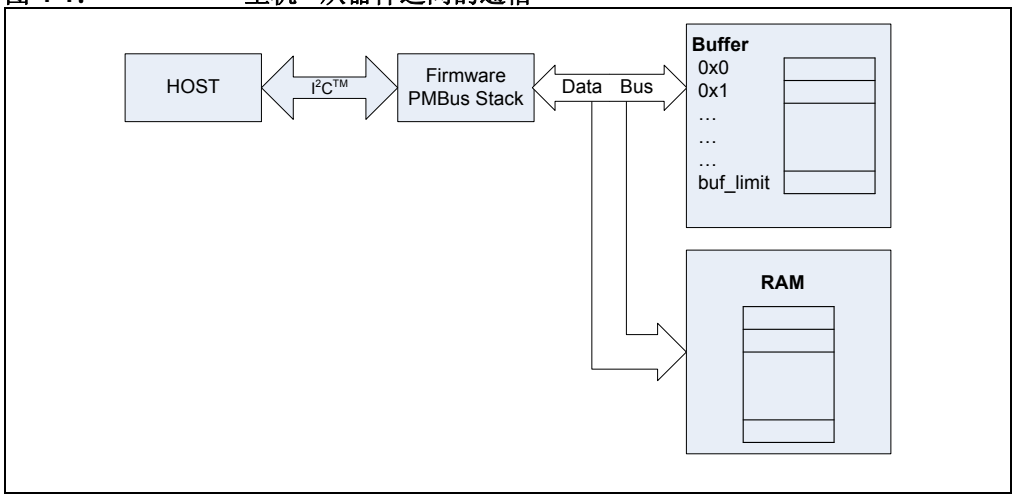
问进行加密。注意在写操作过程中主机发送的字节数目可能与读操作过程中从器件发送的字节数目不同。这一差异与命令有关， COEFFICIENTS 是一个很好的例子，其中主机发送两个数据字节，并期望从器件返回 5 个字节。

第 4 章 数据输入 / 输出变量

PMBus 命令可以在主机和从器件之间交换数据。协议栈是主机和从器件之间的固件接口，其能够处理通信以及将命令的数据字节传递到从器件 RAM 的相应单元，或从 RAM 中取出数据，然后发送到总线上。

主机 – 从器件之间的通信框图如图 4-1 所示：

图 4-1： 主机 - 从器件之间的通信



上面的框图中提到了两种存储空间：缓冲区和普通 RAM 空间。为了完成与主机之间的通信，协议栈对这些存储区都有需求。简单地说，几乎所有这些由应用支持的、组成语言集的命令都需要数据空间。使用 PMBus SEND_BYTE 协议的命令属于例外。

为了更好地过滤那些导致主机发送错误数据的故障，在器件的 RAM 数据存储区中构建了缓冲区。任何写命令都会将其数据字节暂存在缓冲区中。当接收到一个 “停止” 条件时，协议栈就会调用应用函数来处理这些数据字节，之后再将其复制到命令的 RAM GPR 单元。

在 PMBus 的命令语言中，几乎所有命令的数据字节都适用于写或读相应的信息。用户在决定如何设置应用所支持的命令时，需检查器件上可用的数据存储空间。应用程序必须预留一定的存储单元，其数据等于命令所写或读的数据字节的个数。

每个命令都必须以数组的形式预留存储空间，而数组的维数等于该命令的字节个数。例如，对于 STATUS_WORD 命令，应这样定义一个数组：unsigned char STATUS_WORD[2]。注意，数组的维数应等于该命令的字节个数。有关如何对所支持的命令预留存储空间的方法可在**附录 A “由用户自定义的标签列表”**中找到。

4.1 对器件执行写操作

协议栈对事务的有效性进行依次检查。这意味着除非发生数据内容 / 通信故障，否则认为每个事务都是有效的。

在写事务中，协议栈接收的每一个数据字节都会传输到缓冲区中，并存储在那里。当接收到一个 “停止” 条件时，如果没有发生 “内容 / 通信” 故障，协议栈就会调用与命令有关的应用函数。用户如需获取那些与命令有关的应用函数的信息，请查阅**第 7.3 节 “协议栈调用的应用程序函数”。**

由协议栈调用的应用函数可以用来处理缓冲区中的命令的数据字节，以及决定是否应该通过置位 / 复位 `ready_to_copy` 标志，来将这些字节转移到相应的 RAM 单元。该标志是结构体 `global_flags` 的成员。在程序的主循环中，不断地检查标志 `ready_to_copy` 的值，如果为 1，就调用 `copy_buf_to_ram()` 函数。该函数使用缓冲区中的开销数据信息，将数据字节转移到命令的各个 RAM 区中。

在这一点上，应用程序将命令的数据以数组的形式从主机发送到指定的存储区中。主应用程序可以使用那些它认为合适的数据。协议栈无需知道对命令的数据所作的任何更改。

从器件中读

读进行中的命令的数据字节是非常容易的。协议栈会从 RAM 存储空间中取回命令的数据字节。在这一流程中，只有 `Block Write/Block Read Process Call` 命令会发生中断。主应用程序用来处理写操作中那些由主机发送的数据字节，因此在将数据放到总线上之前，协议栈先调用一个应用函数，该应用函数会将数据放到总线上，并提供将由主机从相应 RAM 单元中读取的数据字节。

第 5 章 协议栈功能

以下是一些由 I²C™ 模块作为中断源来识别和处理的状态。

由于中档 PIC16F886 单片机没有字节应答功能，因此，主机接收到的所有经从器件应答的字节都会触发 PIR1 寄存器中的位 SSPIF 置 1，从而产生 I²C 中断。

5.1 状态 1– 启动

每当总线上出现 “启动”或 “重复启动”条件时，就进入该状态。检查该状态下的 i2c_busy 标志，以判断 PMBus 传输是否正在进行。此外，检查 TMR0 寄存器的内容，以确保这不是一个已将 “发送 / 接收”字节中断了的 “启动”。如果主机已经发送了一个已将 “发送 / 接收”字节中断了的 “启动 / 停止”条件，那么 TMR0 寄存器中的内容会将其显示出来。如果 TMR0 寄存器中的数值在 1 到 9 之间，那么这就意味着一个错误已经发生，且 “状态”寄存器中的相应标志 （STATUS_BYTE 和 STATUS_CML）置 1。

5.2 状态 2–MWA （主机写地址）

当从器件的硬件检测到主机发送到总线上的地址与内部 SSPADD 寄存器相匹配时，就进入该状态。这是任何一个 PMBus 事务中的第一个状态。这里，所有的标志和计数器都清零，且从器件将其自身设置为接收或请求数据。

对这一 PMBus 事务， CRC-8 的计算是从发送到 basic_crc 函数的从地址开始的，然后该函数计算当前的 PEC 字节。

5.3 状态 3–MWD （主机写数据）

对该状态的处理是与 MWD_counter 的值严格一致的， MWD_counter 用于对当前 PMBus 事务进入该状态的次数进行编码。该状态包含若干个分支，每个都可用于处理 MWD_counter 计数器的值。计数器的 0 值表示，刚刚接收到命令代码，可以通过查表来检查该命令代码的有效性。如果是真，则返回结构数组中的命令指针，通过该指针可获取与该命令有关的属性。如果是假，则认为该命令不受支持，由总线读或写的数据被丢弃，相应的状态寄存器中的标志置 1。

计数器的以下各值表示由主机发送的数据字节的数目。每次进入一种状态分支，就有两种验证途径：

- 网络链接协议的有效性

  仅处理事务当前时间标记的有效协议，否则检测到故障。

- 由主机发送的字节个数的有效性

  对主机发送的字节进行处理，其个数要与所定义的各命令的字节最大数目相等，否则检测到故障。

5.4 状态 4–MRA （主机读地址）

每当从器件应答了总线上的地址 （置位 R\W 表示主机期望从从器件读取数据字节）时，就进入该状态。

为了能够继续如期运行，协议栈将检查以下情况：

- 从器件已经应答了地址 （RW 复位），且已经接收到命令代码。
- 当前命令是有效的、受支持的。

5.5 状态 5–MRD （主机读数据）

每当从器件发送了数据字节，且已经接到主机的响应时，就进入该状态。

如果到那时还没有出现通信错误，且 SDA 线仍然为低，这意味着主机仍然期望数据字节，那么协议栈会根据当前事务中使用的协议和允许发送的字节数目，来准备那些向主机发送的数据。

5.6 状态 6– 停止

每次主机在总线上放置 “停止”条件时，协议栈都会校验这是否发生在事务当中，且该条件是否会意外中断字节的发送 / 接收。

协议栈必须对命令是否已经通过写事务进行无误的发送作最后的检查。如果该事务成功完成，且命令指出其必须要通过用户自定义函数来处理 （在这种情况下相应的结构成员 `func_support` 必须等于命令的代码），那么协议将调用该自定义函数。

第 6 章 器件故障管理

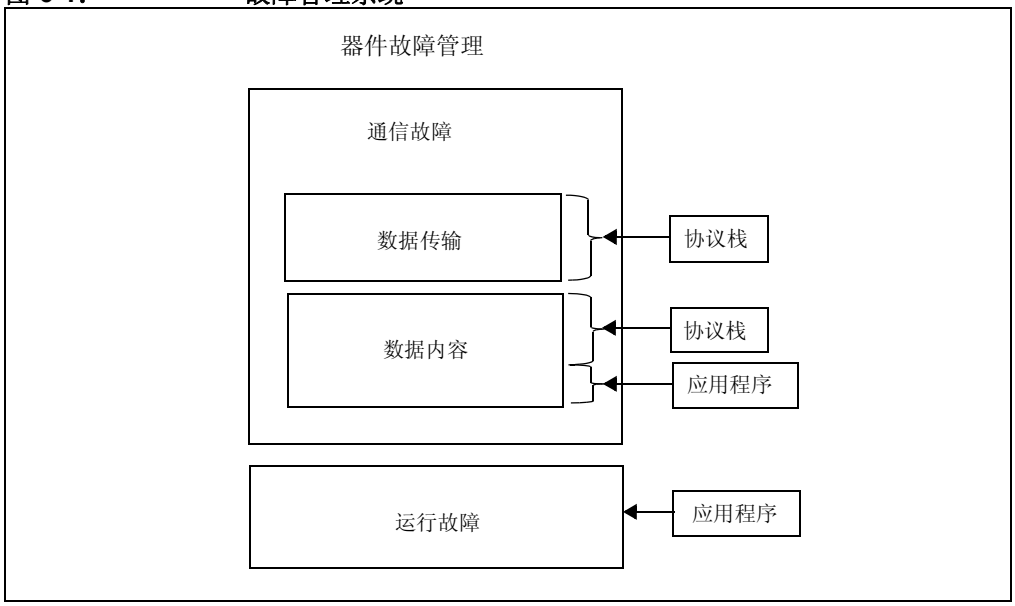
这一节提供的信息，意在帮助用户建立一个有关处理 PMBus 系统中的警告和故障方面的统一机制。然而，这只是一些说明性的信息，并不能代替 PMBus 规范 1.1 版。

PMBus 协议为用户提供了一整套监测运行和管理 PMBus 器件中的故障或警告的工具。不同的警告或故障有不同的来源。它可以是一个温度警告，一个电压故障或仅仅是一个关于通信或数据内容的故障。主机可以使用 READ 命令来查询一个 PMBus 器件的当前状态。此外，对于每一个运行参数，例如输出电压，器件温度或输出电流，都对应一个 READ 命令。 PMBus 协议还具有为电源转换器件的各个重要方面编写故障或警告等级的能力。

由于协议栈完成主机与从器件之间的通信，因此，它负责处理传输故障和一些内容故障。后一类，即内容故障，与事务中的字节内容有关，字节的内容确保数据从主机到从器件的传递：地址和命令代码。然而，协议栈无法对命令数据字节的内容作出推断。这些操作只能由主应用程序完成。

图 6-1 显示了协议栈和主应用程序是如何处理故障管理系统的：

图 6-1： 故障管理系统



几乎对于任何事件， PMBus 协议都支持其设置两种类型的阈值：

- **A.** 作为次要警报的警告阈值

  警告条件是器件端发生问题的一个指示器，其不影响器件的正常运行。

- **B.** 作为主要警报的故障阈值

  故障是一个程度比警告严重的事件，其可能导致器件禁用输出级，以及停止向输出传递能量。

  这里需要提到的一个重要问题是， PMBus 协议提供了一些命令，能够对每一种故障状况进行自动的响应。这些命令都包含一个数据字节，其指示了器件是如何响应故障的。每一种故障响应命令都需要用户从器件响应故障状况的三种方式中选择一个。

在警告或故障事件发生之后， PMBus 器件可以通过设置状态寄存器中的警告 / 故障状况位，等待主机轮询，来向主机通报其状态。

6.1 状态寄存器的树状结构

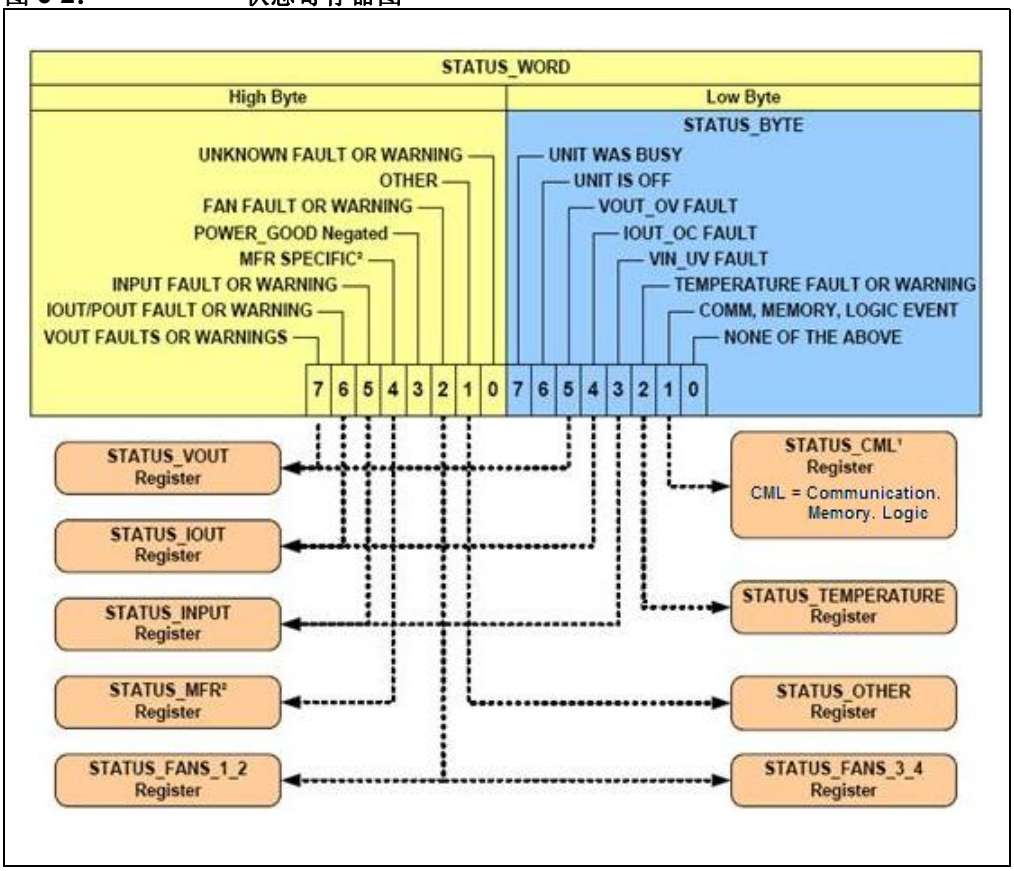
PMBus 协议提供了三种等级的状态寄存器。这使得主机能够在快速事务中获取有关器件的重要信息。基于这一信息，主机能够操作或请求更多的详细信息。下面的图 6-2 给出了 STATUS_BYTE 寄存器、 STATUS_WORD 寄存器和内容较详细的 STATUS 寄存器之间的关系图。从中可见， STATUS_BYTE 寄存器中包含了最重要的故障和警告。这要求大多数基本的 PMBus 器件能够提供有关它们当前状态的最关键信息。STATUS_WORD 中包含了 STATUS_BYTE 作为其低位字节。改变 STATUS_BYTE 中的一个位，会引起 STATUS_WORD 寄存器发生相应的变化。这就是为什么协议栈仅在 STATUS_WORD 中进行操作，而不是在 STATUS_BYTE 和 STATUS_WORD 中同时操作的原因。

在 STATUS_WORD 的高位字节中，有一些额外的位能够提供有关 PMBus 器件的更多状态信息。

高级的 PMBus 器件实现了 7 个额外的寄存器，其含有部件状态方面的较详细信息。主机或电源系统管理器知道可以通过设置 STATUS_BYTE 或 STATUS_WORD 中的哪些位来读取相应的寄存器。

当 STATUS 寄存器清零时，所有寄存器中的所有位就会同时清零。关于 STATUS 寄存器的更多详细信息可在 PMBus 规范 1.1 版中找到。

图 6-2：　　状态寄存器图



6.2　协议栈能够处理哪些警告？

协议栈只是 PMBus 从器件和主机之间的一个通信接口。因此，只有在发送 / 接收过程中发生的警告和故障才能被协议栈检测到并进行处理。由于将所有与数据传输和数据内容相关的错误都归到故障一类，因此，我们认为协议栈能够处理与通信过程相关的故障。

6.2.1　数据传输故障

6.2.1.1　受损数据

该故障是由从主机接收到的 PEC 字节与由协议栈计算出的 PEC 字节不匹配而产生的。产生该故障后会：

- 忽略所接收命令的数据字节。
- 不调用任何与此命令有关的应用函数 （该命令不能由主机识别，因此无法向主应用程序通报）。
- 将 STATUS_BYTE 中的位 CML 置 1。
- 将 STATUS_CML 寄存器中的位 Packet Error Check Failed（数据包错误校验故障）置 1。

因此，在写事务中，当主机试图向从器件发送一条命令，而协议末端的 PEC 字节与协议栈计算出的 PEC 字节不匹配时，上述操作就会发生。请注意从这个角度来说，由主机通过写协议发送的命令 （不包含末尾的 PEC 字节）仍被认为有效。该故障会影响 STATUS_BYTE 和 STATUS_CML 寄存器的内容，各自的标志进行逻辑或运算。

6.2.1.2 发送 / 读取位过少

如果主机在向 PMBus 器件中写数据或从 PMBus 器件中读数据的过程中，还没有发送 / 读取完一个完整的字节就被一个 “启动”或 “停止”条件中断，那么协议栈会：

- 忽略所接收的命令代码和数据。
- 不调用任何与该命令有关的应用函数。
- 将 STATUS_BYTE 中的位 CML 置 1。
- 将 STATUS_CML 寄存器中的 bit 1 （`Other fault`，其他故障）置 1。

6.2.1.3 主机发送 / 读取字节过少

如果主机在完成 PMBus 器件所期望接收 / 发送的所有字节 （包括 PEC 字节）的写 / 读操作之前，就通过发出 “停止”条件终止了 PMBus 数据包的传输，那么协议栈将不把它当作错误处理，而是继续处理命令。它假设主机知道其所作所为，并有意结束事务。

6.2.1.4 主机发送字节过多

当向 PMBus 器件中写数据时，如果主机发送的字节比器件期望的多，那么协议栈会将其作为故障处理，并将：

- 忽略所接收的命令代码和数据。
- 将 STATUS_BYTE 中的位 CML 置 1，以及将 STATUS_CML 寄存器中的 Invalid Or Unsupported Command Received （无效或接收到不支持的数据）位置 1。

6.2.1.5 读过多字节

当从器件中读数据时，如果主机试图读取的字节比器件期望发送的多，那么就会出现以下情况：

- 只要主机保持提供时钟并应答，协议栈就会发送全 1 （FFh）。
- 将 STATUS_BYTE 中的位 CML 置 1。
- 将 STATUS_CML 寄存器中的 bit 1 （`Other fault`）置 1。

6.2.1.6 器件忙

如果器件太忙以致无法响应总线上的通信，那么就会发生数据传输故障。用户可以自行处理位变量 `device_busy`，如果器件太忙以致无法处理事务，就可以将该变量置 1。当协议栈检测到该位置 1，且发生 PMBus 事务时，就会进行以下操作：

- 应答从地址，这是必须的。
- 应答命令和数据字节，但可以忽略它们。
- 只要主机保持提供时钟并应答，协议栈就会发送全 1 （FFh）。
- 将 STATUS_BYTE 中的 BUSY 标志置 1。

虽然用户有时可以在传输过程中将 `device_busy` 变量复位，即意味着现在准备响应通信，但协议栈会继续忽略正在进行的命令，而仅在接收到 “停止”条件之后才将该状态复位。

6.2.2 数据内容故障

6.2.2.1 地址字节的不恰当置位

跟在“启动”条件（而非重复启动条件）之后的器件地址字节的 bit [0] 必须为零（标志着一个写事务）。如果协议栈在“启动”条件（而非重复启动条件）之后，直接接收到了器件的地址（其 bit [0] 等于 1），那么它将：

- 像所有的 PMBus 器件必须应答其地址一样地应答地址字节。
- 只要主机保持提供时钟并应答，协议栈就会发送全 1（FFh）。
- 将 STATUS_BYTE 中的位 CML 置 1。
- 将 STATUS_CML 寄存器中的 bit 1（`Other fault`）置 1。

6.2.2.2 不受支持的命令代码

如果器件接收了一个不受支持的命令，包括那些预留的命令代码，那么协议栈会做出如下响应：

- 忽略所接收的命令代码和数据。
- 将 STATUS_BYTE 中的位 CML 置 1。
- 将STATUS_CML寄存器中的位Invalid Or Unsupported Command Received置1。

6.2.3 由主应用程序处理的数据内容故障

如上所述，协议栈无法检测到事务中的命令的数据字节内容。这就是为什么下面的*数据内容故障*被留给主应用程序处理的原因。

6.2.3.1 无效或不受支持的数据

如果器件接收到一些不受支持的数据，正如 PMBus 文献中所提到的，那么器件的首选响应方法就是调用 `Content_Fault2()` 函数，其将：

- 将 STATUS_BYTE 中的位 CML 置 1。
- 将STATUS_CML寄存器中的位Invalid Or Unsupported Command Received置1。

已对该函数进行了声明和定义，可供用户使用。注意协议栈无法推测命令的数据字节的内容，因此，它会处理那个引起这种故障的命令，而将检查数据字节的任务留给用户。

6.2.3.2 数据超出范围故障

PMBus 文献规定，器件可以选择是否检测尝试将参数设置为器件无法识别的范围这一操作。器件如何知道数据超出范围是留给器件制造商来判断的。如果器件支持检测那些超出器件范围的数据，那么它可以调用 `Content_Fault2()` 函数，其将：

- 将 STATUS_BYTE 中的位 CML 置 1。
- 将STATUS_CML寄存器中的位Invalid Or Unsupported Command Received置1。

已对该函数进行了声明和定义，可供用户使用。注意协议栈无法推测命令的数据字节的内容，因此，它会处理那个引起这种故障的命令，而将检查数据字节的任务留给用户。

6.2.4 主应用程序所能报告的警告和故障

对于如何检查和报告那些与器件运行有关的警告和故障，可按以下两步进行：

- 设置故障和警告阈值。
- 设置所检测故障条件的响应

因此，从器件可以采用一套已定义了阈值的命令来检测警告或故障，并根据其他一些从主机接收到的、已对这些警告 / 故障定义了自动响应的命令来采取相应的措施。

在此处值得一提的有趣内容是：针对几乎所有的重要操作参数， PMBus 定义了三个命令。它们分别为：

- 警告阈值以及应采取的措施
- 故障阈值 （仅仅有阈值，而不采取任何措施）
- 故障条件的响应

这里给出一个有关 VIN_UV 参数的例子。针对该参数，协议定义了以下三条命令：

- VIN_UV_WARN_LIMIT
- VIN_UV_FAULT_LIMIT
- VIN_UV_FAULT_RESPONSE

由于那些与操作参数有关的警告和故障是由主应用程序特殊处理的，因此，主应用程序一次只处理一个警告或故障。

6.2.5 设置警告阈值 – 对警告作出的反应

当检测到一个与器件操作有关的警告条件时，器件必须作出相应的反应，即必须根据相应的参数来采取措施并通知主机。作为主机的通信接口， PMBus 协议栈仅允许采用轮询方法，且器件必须将状态寄存器中的相应位置 1。

这意味着通过轮询方法通知主机，是通过调用函数或程序 （具有修改状态寄存器的功能）来实现的。在**附录 D “PEC 字节计算”**中，有一些关于如何编写这些函数的例子，其由用户定义并从主应用程序中调用。

下面将对上文提到的一个操作参数 VIN_UV 进行说明。这是一个与输入电压的欠电压阈值相关的量。正如我们所见， PMBus 协议为该参数定义了三条命令。在这三条命令中，只有一个是与警告条件紧密相关的，从而也与欠电压参数的警告阈值相关。这就是 VIN_UV_WARN_LIMIT 命令。该 PMBus 命令设置了一个会引起输入电压发出低警告的输入电压值。通常，该值要大于那个由命令 VIN_UV_FAULT_LIMIT 设置的输入欠电压故障阈值。

为了对即将超过 VIN_UV_WARN_LIMIT 阈值进行响应，器件将执行以下操作：

- 将 STATUS_BYTE 中的位 OTHER 置 1。
- 将 STATUS_WORD 的高字节中的位 INPUT 置 1。
- 将 STATUS_INPUT 寄存器中的欠压警告位 VIN 置 1。
- 通知主机。

由于告知主机这一过程采用轮询机理，操作 4 隐含在前三步操作中，即将相应状态寄存器中的位置 1，这样即可实现通过轮询的方法向主机报告故障。

命令 VIN_UV_WARN_LIMIT 定义了输入低电压阈值，因此当超过该阈值时，主应用程序必须调用一个能够执行上述操作的用户自定义函数。

从这个有关输入电压低参数的例子可以看出，协议已经定义了一条命令，其能够设置阈值和提及超出该阈值后将要执行的操作。有关如何编写那些用于处理操作参数的警告条件的函数的例子可在**附录 D “PEC 字节计算”**中找到。

6.2.6 设置警告阈值 – 对故障作出的反应

如前文所述， PMBus 为每个参数都打包了三种命令。在该命令包中，有两种命令用于处理与各自参数相关的故障状况。

以上文的输入电压低参数为例， PMBus 协议与两个包含故障状态的函数同时作用以应对输入电压欠压的状况。

- VIN_UV_FAULT_LIMIT
- VIN_UV_FAULT_RESPONSE

首先，需要用户为这些命令编写代码函数。**附录 D “PEC 字节计算”**中含有相关实例。

第一种命令的意思实际上是显而易见的。它允许主机告知从器件输入电压低故障的阈值信息。主应用程序得到了这个经由协议栈传递的值，并设置阈值。

第二种命令在响应与运行参数相关的故障状况方面，引入了一种 PMBus 协议。PMBus 定义了电压、电流、温度和 ton_max 方面的故障的自动响应。这些自动响应也是通过命令设置的。有关这些命令的例子如下：VIN_UV_FAULT_RESPONSE、IIN_OC_FAULT_RESPONSE、 POUT_OP_FAULT_RESPONSE 等等。这些命令的完整清单可在 PMBus 规范的第 15 章中找到。

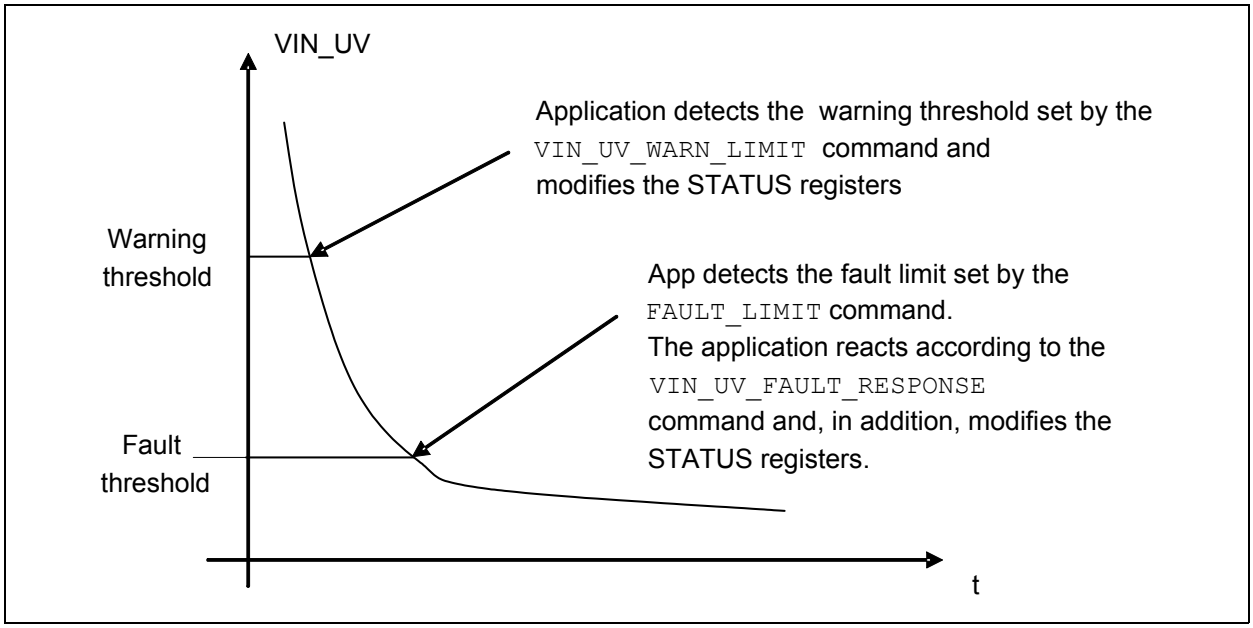
正如我们所见，第二种命令 VIN_UV_FAULT_RESPONSE 是为这些故障设置响应的命令集中的一种。主机可以使用该命令来指示从器件，如果检测到故障，应该进行哪些操作。从器件的固件必须能够对那些由主机通过该命令指示器件所作的操作进行解码。包含有规定了电压、电流、温度和 ton_max 故障响应的数据字节的表可在 PMBus 规范的第 10.5.1 章中找到。

用户必须编写一个能够对从主机接收到的数据字节进行解码的函数 / 程序，以便主应用程序能够知道在超过输入电压低故障阈值的情况下要进行的操作。

PMBus 还规定了一些从器件必须执行的额外操作，类似于那些当超出警告阈值时所应执行的操作。这些操作就是更改状态寄存器中的一些位，并告知主机。对于警告条件，因为协议栈只允许使用 “轮询”方法通知主机，因此该过程通过修改状态寄存器来实现。为了应对与输入电压低参数有关的故障，协议中规定了一些操作，这里给出了一个相关例子，可在 PMBus 规范的第 15.28 章中找到。

总之，应用程序处理输入电压低参数的方式如图 6-3 所示。

图 6-3： VIN_UV 阈值响应



第 7 章 实现协议栈功能所需的用户数据

协议栈工作时要与主用户程序严格协调。由于受主应用程序支持的命令集之间差别很大，且有些命令需要特殊的处理，因此用户必须预先定义一套准则来让协议栈遵循。这些准则中的大多数是由用户定义的标签表示的，但还有一些其他的协议栈功能模式需要用户进行更多的操作。

为实现协议栈功能，用户需要声明一套标签和定义一套变量。下面的章节将介绍一些与用户提供的信息直接相关的协议栈功能模式。附录 A 包含了需要用户知道和可能修改的整套变量和标签。

协议栈与一套预定义的命令集和适当的标签一起提供，用户可将此作为一个着手点。用户所需要做的所有工作就是扩展这套命令集以及将代码添加到应用程序函数中。

为创建实现协议栈功能所需的数据，以下列出了用户需要执行的步骤：

- 定义 / 修改**附录 A“由用户自定义的标签列表”**中列出的标签和变量清单。
- 建立命令表
- 定义可由协议栈调用的函数清单

接下来，我们将依次介绍这些步骤。

7.1 由用户定义的协议栈标签列表和变量

该列表包含的信息如下：从地址、缓冲区最大容量和 PMBus 通信协议等。完整的清单可在**附录 A“由用户自定义的标签列表”**中找到。请注意，协议栈是与一套预定义的标签和变量相连的，用户可以根据自身需要对其进行更改。

7.2 建立命令表

这一节介绍如何在应用中建立命令集以及对其进行编码。

协议栈是主机和从机之间的通信接口，因此，它必须知道从机所支持的那套命令集。每个命令都具有 5 个属性。将这些属性组合在一起成为结构体的元素。因此，每个命令都具有一个与其相关的结构体，且所有来自受支持命令集的结构体组合在一个数组中。该常数结构数组的大小等于应用程序所支持的命令的最大数目。

例 7-1 显示了如何定义一个命令的结构。

例 7-1：　　一个命令的结构

```
struct command{
    char protocol;                    //PMBus protocol
    unsigned char nr_bytes_wr;        //max nr of bytes written
    unsigned char nr_bytes_rd;        //max nr of bytes read
    unsigned char * ptrCommandData; //pointer to command's data
    unsigned char func_support;       //application function support
} ;
```

该例建立了一种新的数据类型，且列表中的每个命令都属于这种类型。没有必要对所支持的每个命令的结构体进行声明和初始化，因为所有的命令都是在结构数组中定义的。

例 7-2 显示了如何对数组中的 VOUT_MAX 命令的结构体进行初始化。

例 7-2：　　VOUT_MAX 命令初始化

```
{RW_Word,0x2,0x2, VOUT_MAX, VOUT_MAX_COMMAND }
```

协议栈将使用该结构体中元素的数值来检查与该命令有关的 PMBus 事务。这里给出了事务中的协议栈是如何使用每个元素的。

- 任何与该命令相关的 PMBus 事务都必须遵循 RW_Word PMBUS 协议，否则协议栈会宣布发生了一个故障。
- 如果主机向设备发送的字节个数超过最大给定值，在该例中是 2 个字节，那么协议栈将宣布发生了一个故障。
- 如果主机从设备读取的字节个数超过最大给定值，在该例中是 2 个字节，那么协议栈将宣布发生了一个故障。
- *ptrCommandData 是命令数据 （其他数组）的一个指针。
- func_support：主机检查该常数变量，看它是否应该调用一个可以处理该命令的应用程序函数。如果是主应用程序处理该命令，那么用户必须用命令代码的值对变量进行初始化，如果不是，就用值 0x0 对变量进行初始化。在该例中， VOUT_MAX 命令受协议栈支持，因此，这个值是 VOUT_MAX_COMMAND （与该命令代码相关的标签）。

以上就是如何在与命令集相关的大结构数组中，对一个命令进行初始化。请注意，数组具有用户想要的任何大小，这取决于应用所支持的命令个数。正如我们接下来所看到的，从协议栈的角度来看，这一详情仅与数组的访问有关。

有关如何建立命令表的一些实例和解说可在**附录 A“由用户自定义的标签列表”**中找到。

7.3 协议栈调用的应用程序函数

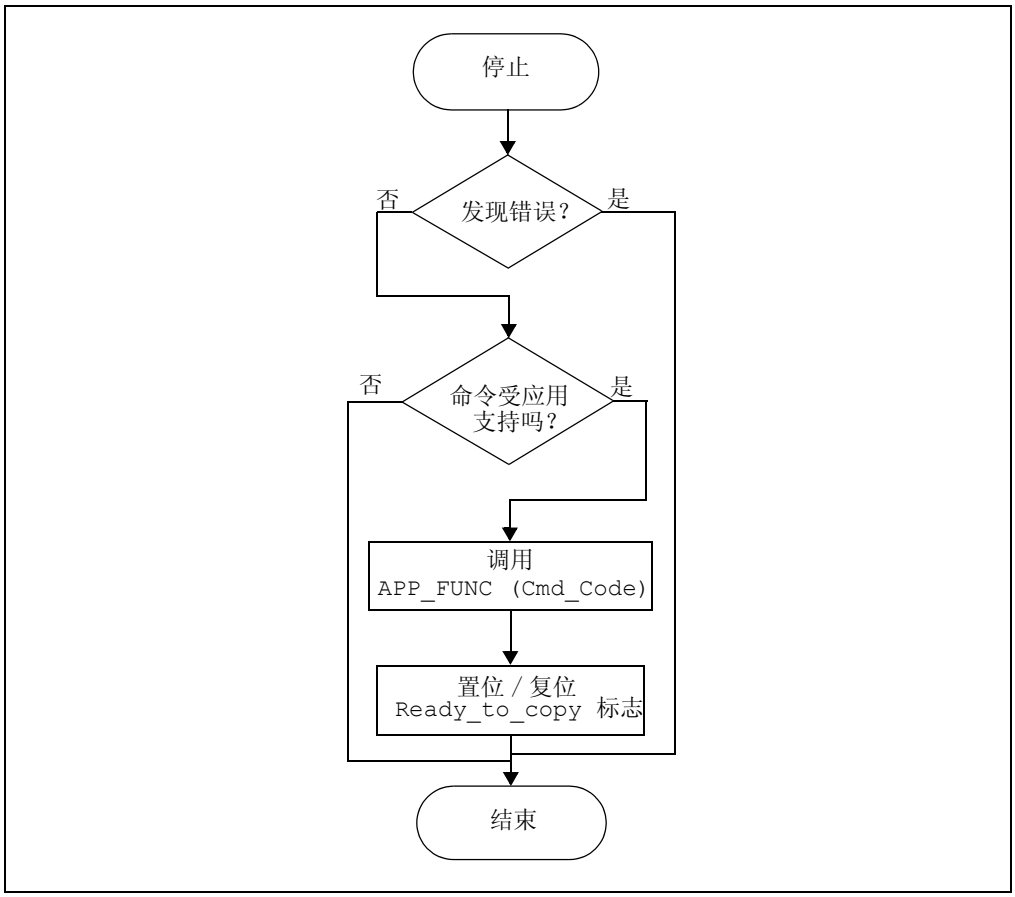
在与主机传递数据的过程中，协议栈必须与主应用程序互相配合。当接收到 “停止” 条件和采用 Block Write/Block Read Process Call （“块写 / 块读过程调用”）协议时，这种交互伴随着写事务一起发生。

写协议执行到 “停止” 条件，且在事务中没有发现错误时，协议栈对结构数组中的 func_support 元素的值进行检查。如果该元素的值等于当前命令代码，那么协议栈认为这是一个主应用程序选择处理的命令。因此，协议栈调用一个用户自定义的应用程序 APP_FUNC，而命令代码作为它的参数。

接下来，该应用程序函数将通过特定的操作，或处理缓冲器中的数据字节来处理命令。如果这些数据字节是正确的，且其内容不会触发故障，否则 APP_FUNC 会将协议栈标志 ready_to_copy 置位。例如，如果检测到 ***invalid or unsupported data***（无效或不受支持的数据）故障，那么 APP_FUNC 就会将协议栈标志 ready_to_copy 复位。为什么这个如此重要？因为在程序的主循环中，有一个 “if” 条件会检查该协议栈标志以及调用或不调用一个可以将缓冲器的内容复制到命令的 RAM 存储单元的函数。

下面是一个简化的流程图，其显示了当出现 “停止” 条件时，协议栈是如何调用或不调用 APP_FUNC 的。

图 7-1： 停止状态下的流程图



例 7-3 显示了在接收到 CLEAR_FAULTS 命令且应用程序已决定处理该命令的条件下，主应用程序可以调用的一个应用程序函数。

例 7-3：　　清除故障命令

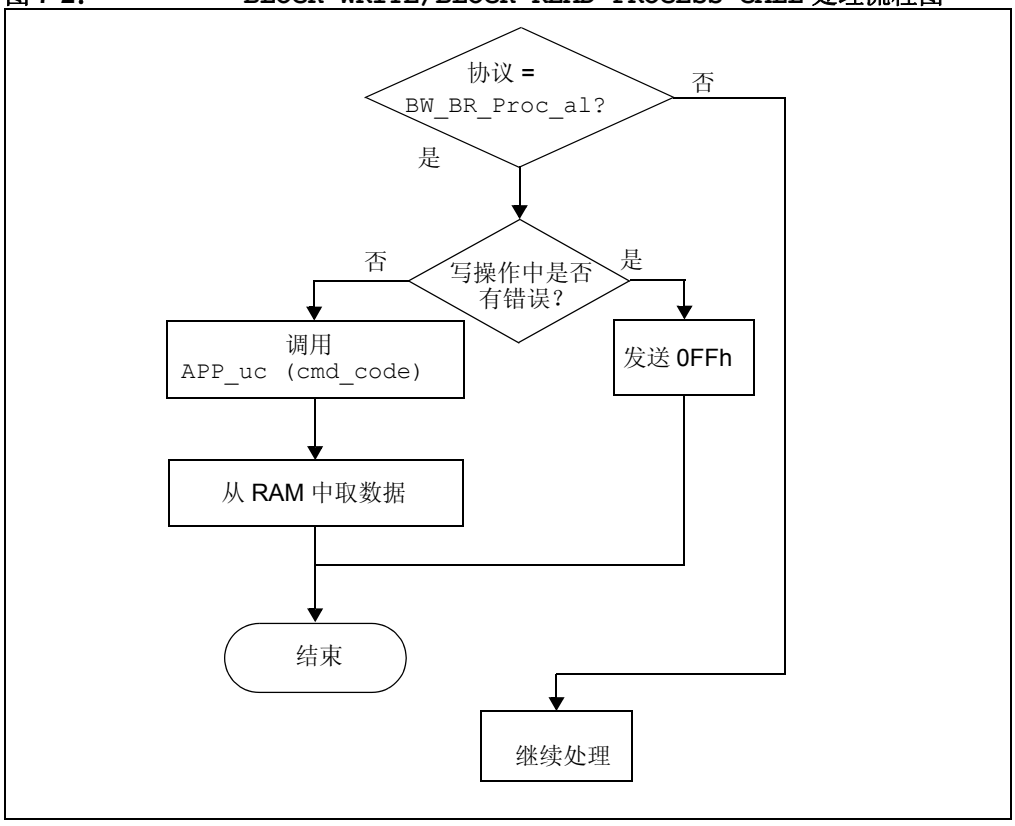
```
void Clear_faults(void){

     STATUS_WORD[0]=0;//STATUS_BYTE is here
     STATUS_WORD[1]=0;
     STATUS_VOUT[0]=0;
     STATUS_IOUT[0]=0;
     STATUS_INPUT[0]=0;
     STATUS_TEMP[0]=0;
     STATUS_CML[0]=0;
     STATUS_OTHER[0]=0;
     STATUS_FAN_1_2[0]=0;
     STATUS_FAN_3_4[0]=0;

}
```

协议栈可以调用应用程序函数的另一种情况是当 Write/Block Read Process Call 协议执行到状态 4，即协议栈流程中的主机读地址 （Master Reads Address，MRA）的情况。因为在这种特殊情况下，协议栈需要将数据放到总线上，它必须首先调用相同的用户自定义的 command_handler 函数。后者将处理那些来自缓冲器的数据字节，并将它们放到当前命令的 RAM 存储单元中。

该例的流程图与前面一个类似，除了协议栈不需再检查是否要为该命令调用一个应用程序函数。这是由于如果假定命令处理到了这一点，那么主应用程序必须提供支持来处理那些目前为止所接收到的数据字节以及准备那些主机需要的字节。

图 7-2 给出了一个有关协议栈如何处理 MRA 状态中的 Write/Block Read Process Call 协议的简化流程图。该协议由类似 COEFFICIENTS 和 QUERY 这样的命令来使用。

图 7-2：　　BLOCK WRITE/BLOCK READ PROCESS CALL 处理流程图



注意在上述流程图中，协议栈首先检查在该 PMBus 协议的写部分中是否有错误。如果有，那么读部分将非常简单，协议栈仅需将值为 0FFh 的字节送回主机。

然而，如果在该事务之前或当中没有遇到错误，那么协议栈会调用一个可以对目前为止接收到的字节进行处理的 APP_FUNC，并将新数据送回命令的 RAM 中。同样，命令代码是所调用函数的参数。

有一个常识性的问题是，在以上介绍的两种情况下，协议栈是否可能恰巧调用了相同的应用程序函数 （当接收到一个 “停止”条件，且主应用程序支持处理那个刚刚结束了事务的命令，或当 Block Write/Block Read Process Call PMBus 协议执行到需要主应用程序操作的状态）。答案取决于用户所选择执行的应用程序。该应用程序可以使用相同的函数或两种不同的函数。唯一需要修改的是在协议栈执行过程中协议栈所调用函数的名称。在前面描述的例子中，默认情况下协议栈仅调用一个函数 APP_FUNC 来触发主应用程序进行操作。

注：

附录 A 由用户自定义的标签列表

请注意，本信息已经在协议栈随附的命令集实例中定义过了。用户可以根据应用程序来更改这些值。请记住，标签和变量的名称对于协议栈来说是特定的，其任何改变都会影响到协议栈。

A.1 标签

A.1.1 通用标签

SLAVE_ADDR – 从机唯一的 I²C™ 地址

SLAVE_ADDR_WRITE – 用于写协议的从机 I²C™ 地址

SLAVE_ADDR_READ – 用于读协议的从机 I²C™ 地址

NR_COMMANDS – 表中命令的最大数目

UNSUPPORTED_CMD_CODE – 不受支持的命令代码

MAX_BYTES – 缓冲区的字节最大数目

MSSP_SSP – 选择供协议栈使用的 PIC 器件的模块类型。值 1 表示 MSSP，值 0 表示 SSP。而其他任何值都表示编译时发生了一个故障。

A.1.2 PMBus 协议标签

SEND_BYTE

READ_BYTE

WRITE_BYTE

RW_BYTE

READ_WORD

RW_WORD

RW_BLOCK

BW_BR_PROC_CALL

A.1.3 命令代码标签

OPERATION_COMMAND

ON_OFF_CONFIG_COMMAND

CLEAR_FAULTS_COMMAND

等

该标签列表与一套受支持的命令集相对应。

A.2 变量

A.2.1 通用标签

unsigned char buffer[MAX_BYTES + 1]; – 缓冲区声明。使用标签 MAX_BYTES。

A.2.2 与命令相关的数据声明 （给出两个命令实例）

一个命令的数据由一个数组构成，该数组的维数等于该命令所读出或写入字节的最大数目。

bank1 unsigned char OPERATION[1];– 为 OPERATION 命令定义的数组，其位于存储器的 Bank 1。

bank1 unsigned char ON_OFF_CONFIG[1]; – 为 ON_OFF_CONFIG 命令定义的数组，其位于存储器的 Bank 1。

以上两个例子是两个内含字节最大数目为 1 的命令。例 A-1 给出了一个使用 BW_BR_PROC_CALL 协议的命令。

例 A-1： 系数命令

```
bank1 unsigned char COEFFICIENTS[5];
```

这一命令有 2 个字节写入和 5 个字节读出。请注意，如果所读出字节的个数大于所写入字节的个数，那么数组大小就等于所读出字节的个数。

列表中包含了一套由用户决定是否执行的命令集。没有必要为采用 SEND_BYTE 协议的命令声明一个数组，因为除了命令代码外，没有字节发送。

A.3 结构数组（给出一个内含 5 个元素的结构数组实例）

例 A-2： 定义一个 5 元素结构数组

```
const command matrix[5] = {
{RW_BYTE,           0x1,   0x1,  OPERATION,       OPERATION_COMMAND},
{RW_BYTE,           0x1,   0x1,  ON_OFF_CONFIG,   ON_OFF_CONFIG_COMMAND},
{SEND_BYTE,         0x0,   0x0,  0x0,             CLEAR_FAULTS_COMMAND},
{BW_BR_PROC_CALL,   0x1,   0x1,  QUERRY,          QUERRY_COMMAND},
{RW_BYTE,           0x1,   0x1,  VOUT_MODE,       VOUT_MODE_COMMAND}
};
```

该结构数组可以使用 const 限定符，因为其数据不会被修改。请注意，对于使用 SEND_BYTE 协议的 CLEAR_FAULTS 命令，因为它不含数据字节，所以没有为其预留数据空间 （即与其相关的数组）。

结构数组可以通过扩展来包含那些用户需要的整套命令集。

A.4 指针数组

如果命令集不够大，即命令个数小于或等于 35，那么协议栈将使用一个函数，在命令数组中对刚接收到的代码进行检查。

然而，如果命令个数大于 35，那么还需要另外一个数组。这个数组包含了这个大结构数组中的命令的指针。它的大小为 255，这意味着在没有分页的情况下，它可以供 255 个命令用。

例 A-3 显示了该指针数组的一部分。

例 A-3：　　该指针数组的一部分

```
const unsigned char ArrayOfIndices[255] = {
      UNSUPPORTED_CMD_CODE,//0x0 cmd code
      0, //OPERATION
      1, //ON_OFF_CONFIG
      2, //CLEAR_FAULTS
      UNSUPPORTED_CMD_CODE,//0x4 cmd code
      UNSUPPORTED_CMD_CODE,//0x5 cmd code
};
```

数组中的每一个元素都有一个指针，从 0 到 255 （数组的最大尺寸）。每个元素与一个受支持或不受支持的 PMBus 命令相对应。数组元素的指针是那些受支持或不受支持的命令代码，而元素的值表示该命令的结构数组的指针。

例如，命令代码 0x0 （PAGE 命令）是不受支持的。这意味着指针数组中的相关元素的值等于 UNSUPPORTED_CMD_CODE 标签。当协议栈接收到该代码 0x0 时，它将检查该指针数组，并发现命令不受支持。

与命令代码相关的元素可以在位置 1 （等于 OPERATION 命令的命令代码）处的指针数组中找到。元素的值是 0，其为 OPERATION 命令处的结构数组中的指针。

当协议栈接收到命令代码 1 时，它将检查指针数组，并获取大结构数组中的指针，对于 OPERATION 命令来说，该指针为 0。

请注意，仅当命令个数小于或等于 35 时，才在编译时间添加该数组。然后一个具有自动检查功能的代码提供给用户。 NR_COMMANDS 标签的声明是非常有用的。如果不声明，将在编译时产生一个错误信息。

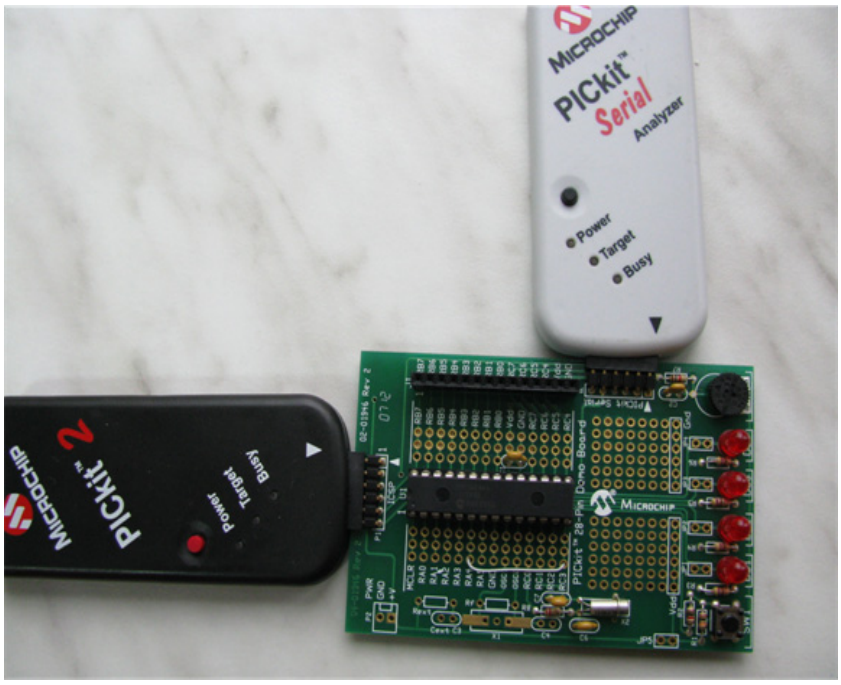
注：

附录 B 协议栈测试

为测试协议栈的功能，需要 PMBus 主从配合。为实现这一功能，在测试协议栈时需要使用 Microchip 的 28 引脚的演示板，其上含有一个单片机 PIC16F886，一个作为主器件的 PKSA （PICkit™ 串行分析器）和一个板上电源 PICkit 2。

板子安装时需要将引脚 RA4 （T0CKI）连接到引脚 RC3 （I²C™ SCK 线）上。图 B-1 显示了系统的外观。

图 B-1： 协议栈的测试系统



为了熟悉主器件 PKSA 的图形用户界面，用户需要参考文档 《PICkit™ 串行分析器用户指南》（DS51647C_CN）”。该文档可从以下网址获取：

http://ww1.microchip.com/downloads/en/DeviceDoc/51647c_cn.pdf

注

附录 C 定义警告和故障响应函数

C.1 警告

当超过警告阈值时，用户必须定义一些可从主应用程序调用的函数，以便修改状态寄存器，从而通知主机。

```
/*****************************************************************
* Function:       VOUT_UV_WARN_LIMIT
*
* PreCondition: A warning generated by an undervoltage condition
* for the output voltage
*
* Input:        None
*
* Output:       None
*
* Side Effects: None
*
* Overview:     This function is called by the main application
* to modify the
* corresponding STATUS in case of the precondition
* mentioned above
* Note:         None
*****************************************************************/
void VOUT_UV_WARN_LIMIT (void){
STATUS_WORD[0] |= 0x1; //set the “OTHER” bit. Be careful, in   //the
PMBus
//literature, this bit is mentioned as //“NONE OF THE ABOVE”
STATUS_WORD[1] |= 0x80; // set the “VOUT” bit
STATUS_VOUT[0] |=  0x20; //set the “VOUT Undervoltage Warning bit”
}
```

请记住， STATUS_BYTE 与 STATUS_WORD 低字节的内容和存储位置是相同的，因此，为该命令再声明其他数组是没有意义的。然而，因为该命令是在结构数组中声明和定义的，因此主机读该命令就像读其他状态命令一样。

C.2 故障

如果操作参数失去控制，超过故障阈值，那么从机必须对这种情况进行响应。而之前主机已通过一个 PARAMETER_NAME_FAULT_RESPONSE，就像实例 IOUT_OC_FAULT_RESPONSE 一样，通知了从机。但除了物理响应外，从机还必须修改状态寄存器。

这里给出一个主应用程序必须使用的函数来响应与输出电流有关的过流故障。

例 C-1： 应用程序函数实例

```
/*****************************************************************
* Function:       IOUT_OC_FAULT_RESPONSE
*
* PreCondition: A fault generated by an overcurrent condition
* for the output current
*
* Input:        None
*
* Output:       None
*
* Side Effects: None
*
* Overview:     This function is called by the main application to
* modify the corresponding STATUS in case of the
* precondition mentioned above
* Note:        None
*****************************************************************/

void IOUT_OC_FAULT_RESPONSE (void){
STATUS_WORD[0] |= 0x1; //set the “OTHER” bit. Be careful, in the PMBus
//literature, this bit is mentioned as “NONE OF        //THE ABOVE”
STATUS_WORD[1] |= 0x80; // set the “VOUT” bit
STATUS_VOUT[0] |=  0x20; //set the “VOUT Undervoltage Warning bit”
}
```

请记住， STATUS_BYTE 与 STATUS_WORD 低字节的内容和存储位置是相同的，因此，为该命令再声明其他数组是没有意义的。然而，因为该命令是在结构数组中声明和定义的，因此主机读该命令就像读其他状态命令一样。

附录 D PEC 字节计算

PEC 字节的计算取决于正在发送的命令，因而也取决于所采用协议的类型。按照经验法则，将最新数据字节放置在总线上的器件也负责计算并将 PEC 字节添加到信息中。针对每一种类型的事务，下文都给出了这种字节的计算方法。

D.1 写事务

正如我们之前所介绍的，在取得了写事务资格的协议中，主机负责发送所有字节。这意味着主机也必须计算和发送 PEC 字节。在从机一侧，协议栈为当前协议计算其自身的 PEC 字节。在事务结束时，如果它从主机接收到 PEC 字节，那么协议栈将检查到不匹配，即一个触发数据传输故障的事件。请注意，在一个写事务中不存在 PEC 字节，也就是 “主机发送字节个数太少”，并不表示发生了故障。因此，认为由从机接收到的不含 PEC 字节的数据是有效的。

D.2 读事务

在遵循读事务的协议中，主机和从机都必须将数据放置在总线上。但只有从机必须计算整个事务的 PEC 字节，并在事务结束时将其放置在总线上。请注意，在不读 PEC 字节时，主机可能选择终止传输。虽然这意味着 “主机读出字节太少”，但在主机知道它在做什么的情况下，这种情况不被视为故障，

D.3 写 / 读事务

在这种联合协议中，从机需要计算整个信息的 PEC 字节。正如上面所提到的，主机可能不选择读 CRC-8 错误校验字节，且协议栈不将其视为故障。

注：

附录 E 命令表解析

每接收到一个新的命令代码，协议栈就必须准备获取与命令表中当前命令相关的信息。为达到这一目的，协议栈必须对结构数组进行解析，找出与数组中命令相关的结构数组的指针。

因为实际上协议栈要从大的结构数组中查询元素，这严重消耗了单片机的执行时间，可能会使从机丢失主机发送的下一个数据包。

通常最有效的方法是引入第二个数组。这是一个大结构数组中的受支持命令的指针数组。该数组的大小等于受支持的命令代码中的最大值。

这里给出一个如何构建指针数组的例子。在该例中， 0xD0 是受支持的命令代码中的最大值，因此它表示了数组的大小。

第一列是命令代码清单，而第二列表示数组的实际定义，该数组中装有一些与受支持的命令代码相对应的结构数组指针 （左边一列），或一个已指明不受支持的标签。

用户负责定义指针数组和标签 unsupported_cmd_code。如果左边一列中的命令代码不受支持，那么该标签将代替结构数组的指针。

表 E-1： 构建指针数组

命令代码 （受支持命令列表）	结构数组的指针 （数组元素的值）	Array[N]
0x0	0x0	Array[0]
0x1	Label unsupported_cmd_code	Array[1]
0x2	0x1	Array[2]
0x3	0x2	Array[3]
...	...	...
...	...	...
0x0	Label unsupported_cmd_code	Array[0x9F]
...	...	...
0x0	0x9A	Array[0xD0]

正如我们所观察到的，代码为 0xD0 的命令可在大结构数组的 0x9A 指针处找到。

注：

附录 F 缓冲器结构

缓冲器包括那些正在被处理的写命令的数据字节，和一些用于将数据字节复制到相应的 RAM 单元的开销数据。

表 F-1： 缓冲器结构

数据字节编号
数据字节 1
数据字节 2
…
…
…
数据字节 N

任何一个写事务，甚至是组命令协议，从开始到停止，都会由主机向设备发送一个命令。因此，缓冲器每次都会包含那些单个命令数据。每开始一个新事务，缓冲器被重写一次。

当前事务可能会因为一个故障而中途中断。缓冲器将包含设备的不相关数据，但协议栈会忽略该信息，且不向主应用程序通报。

全球销售及服务网点

美洲

公司总部 Corporate Office
2355 West Chandler Blvd.
Chandler, AZ 85224-6199
Tel: 1-480-792-7200
Fax: 1-480-792-7277
技术支持：
http://support.microchip.com
网址：www.microchip.com

亚特兰大 Atlanta
Duluth, GA
Tel: 678-957-9614
Fax: 678-957-1455

波士顿 Boston
Westborough, MA
Tel: 1-774-760-0087
Fax: 1-774-760-0088

芝加哥 Chicago
Itasca, IL
Tel: 1-630-285-0071
Fax: 1-630-285-0075

克里夫兰 Cleveland
Independence, OH
Tel: 216-447-0464

Fax: 216-447-0643

达拉斯 Dallas
Addison, TX
Tel: 1-972-818-7423
Fax: 1-972-818-2924

底特律 Detroit
Farmington Hills, MI
Tel: 1-248-538-2250
Fax: 1-248-538-2260

科科莫 Kokomo
Kokomo, IN
Tel: 1-765-864-8360
Fax: 1-765-864-8387

洛杉矶 Los Angeles
Mission Viejo, CA
Tel: 1-949-462-9523
Fax: 1-949-462-9608

圣克拉拉 Santa Clara
Santa Clara, CA
Tel: 408-961-6444
Fax: 408-961-6445

加拿大多伦多 Toronto
Mississauga, Ontario,
Canada
Tel: 1-905-673-0699
Fax: 1-905-673-6509

亚太地区

亚太总部 Asia Pacific Office
Suites 3707-14, 37th Floor
Tower 6, The Gateway
Harbour City, Kowloon
Hong Kong
Tel: 852-2401-1200

Fax: 852-2401-3431

中国 – 北京
Tel: 86-10-8528-2100
Fax: 86-10-8528-2104

中国 – 成都
Tel: 86-28-8665-5511
Fax: 86-28-8665-7889

中国 – 香港特别行政区
Tel: 852-2401-1200
Fax: 852-2401-3431

中国 – 南京
Tel: 86-25-8473-2460
Fax: 86-25-8473-2470

中国 – 青岛
Tel: 86-532-8502-7355
Fax: 86-532-8502-7205

中国 – 上海
Tel: 86-21-5407-5533
Fax: 86-21-5407-5066

中国 – 沈阳
Tel: 86-24-2334-2829
Fax: 86-24-2334-2393

中国 – 深圳
Tel: 86-755-8203-2660
Fax: 86-755-8203-1760

中国 – 武汉
Tel: 86-27-5980-5300
Fax: 86-27-5980-5118

中国 – 厦门
Tel: 86-592-238-8138

Fax: 86-592-238-8130

中国 – 西安
Tel: 86-29-8833-7252
Fax: 86-29-8833-7256

中国 – 珠海
Tel: 86-756-321-0040
Fax: 86-756-321-0049

台湾地区 – 高雄
Tel: 886-7-536-4818
Fax: 886-7-536-4803

台湾地区 – 台北
Tel: 886-2-2500-6610
Fax: 886-2-2508-0102

台湾地区 – 新竹
Tel: 886-3-6578-300
Fax: 886-3-6578-370

亚太地区

澳大利亚 Australia - Sydney
Tel: 61-2-9868-6733
Fax: 61-2-9868-6755

印度 India - Bangalore
Tel: 91-80-3090-4444
Fax: 91-80-3090-4080

印度 India - New Delhi
Tel: 91-11-4160-8631
Fax: 91-11-4160-8632

印度 India - Pune
Tel: 91-20-2566-1512
Fax: 91-20-2566-1513

日本 Japan - Yokohama
Tel: 81-45-471- 6166
Fax: 81-45-471-6122

韩国 Korea - Daegu
Tel: 82-53-744-4301
Fax: 82-53-744-4302

韩国 Korea - Seoul
Tel: 82-2-554-7200
Fax: 82-2-558-5932 或
82-2-558-5934

马来西亚 Malaysia - Kuala Lumpur
Tel: 60-3-6201-9857
Fax: 60-3-6201-9859

马来西亚 Malaysia - Penang
Tel: 60-4-227-8870
Fax: 60-4-227-4068

菲律宾 Philippines - Manila
Tel: 63-2-634-9065
Fax: 63-2-634-9069

新加坡 Singapore
Tel: 65-6334-8870
Fax: 65-6334-8850

泰国 Thailand - Bangkok
Tel: 66-2-694-1351
Fax: 66-2-694-1350

欧洲

奥地利 Austria - Wels
Tel: 43-7242-2244-39
Fax: 43-7242-2244-393

丹麦 Denmark-Copenhagen
Tel: 45-4450-2828
Fax: 45-4485-2829

法国 France - Paris
Tel: 33-1-69-53-63-20
Fax: 33-1-69-30-90-79

德国 Germany - Munich
Tel: 49-89-627-144-0
Fax: 49-89-627-144-44

意大利 Italy - Milan
Tel: 39-0331-742611
Fax: 39-0331-466781

荷兰 Netherlands - Drunen
Tel: 31-416-690399
Fax: 31-416-690340

西班牙 Spain - Madrid
Tel: 34-91-708-08-90
Fax: 34-91-708-08-91

英国 UK - Wokingham
Tel: 44-118-921-5869
Fax: 44-118-921-5820

03/26/09