**请注意赛普拉斯已正式并入英飞凌科技公司。**
此封面页之后的文件标注有“赛普拉斯”的文件即该产品为此公司最初开发的。请注意作为英飞凌产品组合的部分，英飞凌将继续为新的及现有客户提供该产品。

**文件内容的连续性**
事实是英飞凌提供如下产品作为英飞凌产品组合的部分不会带来对于此文件的任何变更。未来的变更将在恰当的时候发生，且任何变更将在历史页面记录。

**订购零件编号的连续性**
英飞凌继续支持现有零件编号的使用。下单时请继续使用数据表中的订购零件编号。

# I²C 硬件模块说明书 I2CHW V 1.90

Copyright © 2012 Cypress Semiconductor Corporation. All Rights Reserved.

| 资源 | PSoC® 模块 | | | API 存储器 （字节） | | 引脚 （每个外部 I/O） |
|---|---|---|---|---|---|---|
| | 数字 | 模拟 CT | 模拟 SC | Flash （闪存） | RAM | |
| CY8C29/27/24/22/21xxx, CY8C23x33, CY7C603xx, CY7C64215, CYWUSB6953, CY8CLED02/04/08/16, CY8CLED0xD, CY8CLED0xG, CY8CTST110, CY8CTMG110, CY8CTST120, CY8CTMG120, CY8CTMA120, CY8C21x45, CY8C22x45, CY8C28x45, CY8CPLC20, CY8CLED16P01, CY8C28xxx, CY8C21x12 | | | | | | |
| 从器件 | 0 | 0 | 0 | 374 - 591 | 6 - 11 | 2 |
| 主器件 | 0 | 0 | 0 | 1031 - 1085 | 7 - 11 | 2 |
| 多主从器件 | 0 | 0 | 0 | 1331 - 1944 | 14 - 22 | 2 |

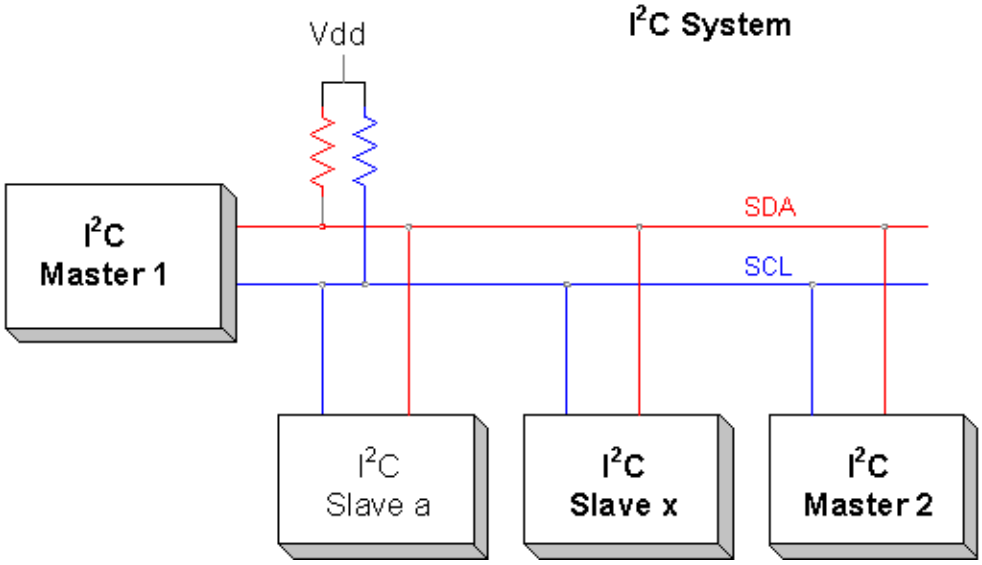
如需一个或多个使用此用户模块且完全配置的功能性示例工程，请登陆 www.cypress.com/psocexampleprojects.

## 功能概述

- 行业标准 Philips I²C 总线兼容接口
- 主器件和从器件操作模式，可实现多主控操作
- 只需要两个引脚 （SDA 和 SCL）与 I²C 总线连接
- 标准数据速率为 100/400 kbps，同时支持 50 kbps
- 高级 API 只需少量用户编程
- 7-bit 寻址模式

I²C 硬件用户模块在固件中实现 I²C 器件。 I²C 总线是 基于 Philips® 开发的行业标准的两线硬件接口。主器件在 I²C 总线上启动所有通信，并为所有从器件提供时钟。 I2CHW 用户模块支持标准模式，最大速度达 400 kbps 。 此模块不需使用任何数字或模拟用户模块。 I2CHW 用户模块与同一总线上的其他从器件兼容。

Figure 1. I²C 框图




## 功能描述

此用户模块支持 I²C 硬件资源。 当 CPU 时钟配置的运行频率为 12 MHz 时，它能够以 50/100/400 kbps 的速度传输数据。 可以使用更快或更慢的 CPU 时钟，但可能有时导致寻址或数据处理过程中总线停止。 I²C 规范允许主器件以从 100 kHz 到低至 DC 的时钟频率运行。 SDA 和 SCL 可通过两种方式直接访问硬件资源。 在提供的 API 中支持七位寻址模式。 该模块不需要任何数字或模拟 PSoC 模块即可传输数据。

I²C 资源支持逐字节级别的数据传输。 在每个地址或数据发送 / 接收末尾，将报告传输状态，或者可能触发专门的中断。 状态报告和中断生成取决于硬件检测到的数据传输方向和 I²C 总线条件。 可将中断配置为在字节完成、检测到总线错误和仲裁损耗时发生。

每个 I²C 数据操作由开始位、地址、读 / 写方向、数据和停止位组成。 此用户模块使用的 I²C 资源可以作为 I²C 主器件或 I²C 从器件进行操作。 无论主器件或从器件操作，该用户模块均提供了一种基于中断的、缓冲式的传输机制。 I2C 通信通过在前台调用函数启动。 在完成消息的每个字节传输时，将触发中断并停止 I²C 总线。提供的中断服务子程序 (ISR) 将根据用户执行的初始化，在总线上执行适当的操作，使通信继续。 无法确认地址的从器件在接收到下一个地址之前不会再被中断。 从器件必须以应答或非应答来响应每个地址。

如果忽略主器件与从器件之间的区别，则大致可以说存在两种器件，即接收器和发送器。 对于 I²C 接收器，中断出现在传入数据的第 8th 位之后。 在此位置点，接收器必须决定是应答 (ack) 还是不应答 (nak) 传入的字节 （无论它是地址还是数据）。 接收器然后将适当的控制位写入 I2C_SCR 寄存器，将 ack/nak 状态通知给 I²C 资源。 通过卸载总线、在总线上提供 ack/nak 状态和将下一个数据字节移入，从而对 I2C_SCR 寄存器的写入操作实现了对总线上的数据流的步进操作。 对于第二种发送器，中断出现在外部接收器件提供应答 ack 或非应答 nak 之后。 可以读取 I2C_SCR 以确定此位的状态。

对于发送器，数据加载到 I2C_DR 寄存器中，再次写入 I2C_SCR 寄存器以触发下一个数据的传输。

若使用缓冲读写子程序 （bWriteBytes(...)、bWriteCBytes(...)、fReadBytes(...)），则不需要使用任何缓冲初始化函数 （InitWrite(...)、InitRamRead(...)、InitFlashRead(...)）。 这些函数将作为启动缓冲读写子程序 （bWriteBytes(...)、bWriteCBytes(...)、fReadBytes(...)）的函数的一部分被调用。

除了支持缓冲式数据操作外，I²C 主器件用户模块还支持一字节一字节的轮询数据传输，而无需使用中断或提供的中断服务程序 ISR。 所有 API 都在本文档中进行了说明。

对 I²C 总线的详细说明及这里资源的使用例程，可参考网上可以下载的 I²C 完整规范，以及随 PSoC Designer 一起提供的器件的数据手册。

## I2C 睡眠处理

当配合进入睡眠状态的项目使用 I2C 时，需特别小心。 在项目进入睡眠状态之前，请遵循这下列步骤，以确保正确的进入睡眠和正确 I2C 处理：

1. 确保所有 I2C 通信已完成
2. 通过调用 Stop API 禁用 I2C。
3. 将 I2C 引脚配置为模拟 High-Z （高阻态）驱动模式。

在器件从睡眠中唤醒时遵循下列步骤：

1. 确保无活动的 I2C 通信。
2. 通过调用 Start API 启用 I2C。
3. 将 I2C 引脚配置为 “ 开漏 (Open-Drain) 驱动低电平 ” 驱动模式。
4. 使能中断。

## 多主从 （MultiMasterSlave ）操作

MultiMasterSlave 操作是主、从器件的扩展，综合了两种操作。 同样，实施过程还使用了大部分代码来实现 API 和 ISR。 MultiMasterSlave 增加的功能有：

1. 同一总线上可以有多个主控器件。 如果总线上有多个主器件均试图发出指令，则有一种机制使它们不冲突。 这种机制被称为仲裁。 在 Philips I²C 规范中对此有完整讨论。

2. 每一主器件还可以作为从器件进行操作，并有相应从器件地址。 另一主器件可以寻址任何其他主器件的 “ 从 ” 实例，就好像是寻址从器件。 如果主器件在某一地址上丢失仲裁，而该地址是其自身从地址，则在进行 I²C 数据操作的过程中，它能作为从器件正确进行响应。

## 多主从 （MultiMasterSltiMasterSlave）应用编程接口 APIs

用于 MultiMasterSlave 实现的函数调用和用于主从器件的那些函数几乎相同。 但是由于 MultiMasterSlave 使用主、从器件，因此对调用函数的名称进行了适当的调整。 比如：如果从器件 （与主器件）使用函数 “InitWrite(...)”，则 MultiMasterSlave 给出两个指定函数：“InitMasterWrite(...)” 与 “InitSlaveWrite(...)” 这两个初始化函数的使用方法，应同在单个主器件或单个从器件的情况下一样。 更具体来说，当使用 MultiMasterSlave 的主器件部分进行缓冲式读写时，应从 bWrite() 或 fRead() 函数中调用合适的初始化函数 （因此不能像通常那样由用户直接使用）。 另一方面，程序员 “ 会 ” 使用 InitSlave 函数组来建立一个存储器区域，便于将来外部主器件对其进行访问。

MultiMasterSlave 用户模块的主器件操作和从器件操作是分开启用的。 换言之， EnableMaster() 和 EnableSlave() 必须都调用，才能启用从器件和主器件功能。

## 设计注意事项

- 与 I²C 的软件实现不同，此实现的运行使用内部生成的时钟。 主振荡器可以设置为任意时钟频率。 数据吞吐量可能受处理器速度的限制，但逐字节数据传输在指定的 I²C 速度运行。

- 从器件保留剩余缓冲空间的内部数量，以便主器件可访问。 对于 MultiMasterSlave，计数变量的名称为 UMNAME_SlaveWrite_Count 和 UMNAME_SlaveRead_Count.。从器件 （只有从器件）变量名称为 UMNAME_Read_Count 和 UMNAME_Write_Count.。这些变量为全局变量，可通过用户 C 语言或汇编语言代码将其加入适当的 “extern” 声明进行访问。 用户可以通过从计数变量的初始值减去计数变量的当前值来确定主器件读取或写入的字节数。 在从器件 （仅作为从器件）用户模块中，通过使用函数 UMNAME_InitWrite、UMNAME_InitRamRead 和 UMNAME_InitFlashRead 来设置计数变量的初始大小。 在 MultiMasterSlave 用户模块中，用于设置可选从器件计数值的调用函数是 UMNAME_InitSlaveWrite、UMNAME_InitSlaveRamRead 和 UMNAME_InitSlaveFlashRead.
- 通过使用缓冲区来读写 $I^2C$ 从器件内的数据。 在启用 $I^2C$ 从器件之前，用户必须对相应的缓冲区进行初始化。 $I^2C$ 主器件初始化某个读或写操作后，在 I2CHW_Status 字节设置适当的状态位。 此时从器件中的前台进程能够对存入写缓冲区的数据或从读取缓冲区提取的数据执行操作。 从器件数据传输子程序 (ISR) 不允许要访问的缓冲区长度超过在进入 ISR 时所定义的长度。 对缓冲区的读写可以采用以下方式处理：

1. 如果 $I^2C$ 主器件试图读取多于缓冲区所包含的数据，将重复发送最后一个字节，直到 $I^2C$ 主器件停止读取。 （$I^2C$ 协议未定义使 $I^2C$ 从器件停止主器件读取操作的方法。）
2. 在 $I^2C$ 主器件向 $I^2C$ 从器件写一个或多个数据字节时，一旦收到最后一个有可用存储空间保存的字节，从器件即生成 NAK。 如果 $I^2C$ 主器件继续写入数据，从器件将继续生成 NAK。 生成第一个 NAK （数据存储在最后的可用位置）后，将不再存储后续数据。
3. 对于从器件读写缓冲区数据操作，在使用缓冲区的最后一个字节时，设置出现 “ 错误 ”。 原因是，由于从器件不能控制主器件读写字节的数量，一旦缓冲区最后一个字节被用完，无法确保 “ 不 ” 发生某种溢出的情况。 为此，为了避免使用 （从器件）读缓冲区溢出、以及 （从器件）写操作过程中对主器件生成 NAK，读写缓冲区应设置为比主器件预期要使用的最大容量长一个字节。 例如，一个一字节从器件写缓冲区在写入一个字节时，总是使从器件向主器件产生非应答 （NAK）。 向二字节缓冲区写一个字节，可产生对主器件的应答信号 ACK。
4. 如果缓冲区的长度定义为 0，则写入 $I^2C$ 从器件的数据将不被确认也不被存储。

启用直接从闪存读取数据功能，同时还可允许操作 RAM 或闪存缓冲区。 无论数据传输 ISR 使用位于闪存/ROM 还是 RAM 中的读取缓冲区，都可以使用提供的 API 对其进行配置。

### 动态重配置

我们不建议将 I2CHW 资源加入动态加载 / 卸载覆盖式使用。 将 I2CHW 资源仅作为基础配置的一部分放置。 可以根据操作要求修改 I2CHW 模块的操作，但是尝试 “ 删除 ” 作为动态重新配置一部分的资源会对外部 $I^2C$ 器件产生负面影响。

### $I^2C$ 寻址

$I^2C$ 地址包含在读取或写入数据操作的第一个字节的高 7-bits 位中。 该字节用于 $I^2C$ 主器件对从器件的寻址。 有效的地址选择为 0-127 （十进制）。 该字节的最低有效位 LSB 包含 R/~W 位。 如果该位是 0，则写入地址。 如果 LSB 是 1，则从地址从器件中读出数据。

在内部，用户模块获取输入地址，移位并将其与读 / 写位组合成为完整的地址字节。

### 示例

地址 0x48 作为参数传递或定义为从器件地址。 包含读 / 写信息的参数将单独传递。 $I^2C$ 主器件发送字节 (8-bits) 0x90 以向从器件写入数据，发送字节 0x91 以从从器件读取数据。

因为从器件模块的地址参数接受十进制数值输入，所以 7-bit 地址也可以采用十进制键入 （十进制数 72）。

## 直流和交流电气特性

如框图显示，$I^2C$ 总线需要外部上拉电阻。 上拉电阻 ($R_P$) 取决于供电电压、时钟频率和总线电容。 输出阶段的任何器件 （主器件或从器件）的最小灌电流应不小于 3 mA （在 $V_{OLmax}$ = 0.4V 的条件下）。 这将 5-volt 系统最小上拉电阻值限制在大约 1.5 KΩ。 $R_P$ 的最大值取决于总线电容和时钟频率。 对于总线电容为 150pF 的 5V 系统，上拉电阻不应超过 6 kΩ。 有关 “$I^2C$ 总线规范 ” 的更多信息，请参见 Philips 网站 www.philips.com。

I2C 的一个常见考虑事项是上拉电阻的大小。 对于大多数设计，上拉电阻介于 1.5k 到 6k。 应根据通信频率和总线电容选择电阻的大小。 总线电容和上拉电阻值较大时，会延长时钟或数据线路的上升时间。 I2C 规范指明了最大上升时间。 I2C 如果总线上升时间超出了此最大值，I2C 通信会无法正常进行。 上拉电阻必须大小合适，以防出现过大的上升时间。 I2C 规范中提供了图形，用以确定上拉电阻的大小。 更多信息，请参考 I2C 规范。

**Note** 从赛普拉斯或其获得分许可的其中一个联营公司处购买 $I^2C$ 组件，即可根据 Philips $I^2C$ 专利权获得一份许可，以便在 $I^2C$ 系统中使用这些组件，但前提是该系统符合 Philips 定义的 $I^2C$ 标准规范。

## 放置

I2CHW 用户模块的 SCL 和 SDA 有两种选择，即 P1[5]/P1[7] 或 P1[0]/P1[1]，并且不需要任何数字或模拟 PSoC 模块。 不存在放置限制。 无法放置多个 $I^2C$ 模块，因为 $I^2C$ 模块使用专用 PSoC 资源模块和中断。

## 参数和资源

所有缓存名称的定义均体现了其对 $I^2C$ 主器件的用途。 例如，I2Cs_pRead_Buf 是指包含了将由 $I^2C$ 主器件读取的数据的 RAM 位置 。

**Slave_Addr**

这是一个从器件和多主从器件 (Slave and MultiMasterSlave) 参数。 此参数选择一个 7-bit 的从器件地址，$I^2C$ 主器件将采用此地址作为从器件或处于从器件模式下的多主从器件的地址。 有效的地址选择范围为 0-127（十进制）。

**Auto_Addr_Check**

选择是禁用还是启用硬件地址功能。 如果设置为禁用 （disable），则硬件地址比较功能不可用。 将此选项设置为启用，则 I2C 模块不支持特殊系统地址定义。

此参数是仅针对 CY8C28045 器件用户可配置的参数。

**I2C_Clock**

具体指定了运行 $I^2C$ 接口所需的时钟频率。 存在三种可用的时钟速率：

- 50 K 标准
- 100 K 标准
- 400 K 快速

**Note** 以上 I2C 时钟基于 24MHz 的 SysClk。 如果 SysClk 小于 24 MHz，I2C 时钟将按比例减小。 例如，如果 SysClk 是 6 MHz，则可能的时钟频率为 12.5K、25K、和 100K。SysClk 与 CPU 时钟是分开的。

I2C_Pin

从端口 1 选择用于 $I^2C$ 信号的引脚。 无需为这些引脚选择适当的驱动模式，PSoC Designer 会自动完成这项选择。

**注意** 改变默认驱动模式，可使引脚以非正常方式驱动，会引起总线上不可预期的操作。

Read_Buffer_Types

选择数据读取所支持的缓冲区的类型。 有 2 种选择项可用： “ 仅 RAM”(RAM ONLY) 或 “RAM 或闪存 ”(RAM OR FLASH)。 选择 “ 仅 RAM”(RAM ONLY) ，会删除支持直接闪存 ROM 读取所需的代码和变量。 选择 “RAM 或闪存 ”(RAM OR FLASH)，能够提供用于读取 RAM 缓冲区或闪存 ROM 缓冲区以便将数据发送到主器件的代码和变量的支持。 如果选择了 “RAM 或闪存 ”，用 API 调用选择使用 RAM 缓冲区还是闪存读取缓冲区。

Communication_Service_Type

此项参数让用户能够在基于中断的数据处理策略或轮询策略之间做出选择。 在基于中断的策略中，数据传输针对预先定义的缓冲区而启动。 数据随后在后台以最快速度移入或移出缓冲区。 其中还包含一个用于处理数据移动的中断服务子程序 (ISR)。 选择轮询数据处理策略时，用户可以控制何时执行数据移动操作。 为了实现轮询策略，用户必须定期调用函数 I2CHW_Poll()（具体实例名称请参见 I2C.h 文件）。 每次轮询函数被调用时，将有 1 个单字节进行传输。 其他 $I^2C$ 函数的使用完全相同。 在中断延迟极为严重的情况下，可使用轮询通信策略 （异步通信中断也可能导致问题）。 另一种使用情况是在用户要求对何时进行数据传输拥有绝对控制权的情况下。 轮询的一个缺点是，当 $I^2C$ 状态机启用时，总线将会在每个字节传输后自动停顿，直到轮询函数被调用时为止。 轮询函数只在 $I^2C$ 的从器件和多主从器件实现方式下可用。 “ 单一主器件 (Single Master)” 实现方式提供了支持字节式数据传输的 API 函数。

### 中断生成控制

当在 PSoC Designer 中选中 **启用中断生成控制** 复选框时，会有一个附加参数变为可用。 可以在以下菜单下找到此复选框： **项目 （Project） > 设置 （Settings） > 芯片编辑器 （Chip Editor）**. 当多个外覆层（overlays）和多个用户模块跨外覆层所共享的中断一起使用时，中断生成控制非常重要：

IntDispatchMode

IntDispatchMode 参数用于指定中断请求的处理方式，这些中断是由同一模块而在不同外覆层中的多个用户模块所共享。 选择 **ActiveStatus** 会导致固件在处理共享的中断请求之前测试哪一个外覆层正处于活动状态。 每次共享的中断发出请求时，都会进行此测试。 这会增加延迟，还会产生为共享中断请求提供服务的不确定过程，但是不需要任何 RAM。 选择 **OffsetPreCalc** 参数时，会导致固件仅当初始加载一个外覆层时，计算共享中断请求的来源。 这种计算可减少中断延迟，并产生为共享中断请求提供服务的确定过程，但会占用一个字节的 RAM 空间。

## 应用程序编程接口

应用程序编程接口 (API) 固件提供了高级命令，支持多字节传输的发送和接收操作。 可以在 RAM 存储器或闪存中设置读取缓冲区。 写入缓冲区只能设置在 RAM 存储器之内。

Note

在这里，如同所有用户模块 API 中的一样，A 和 X 寄存器的值可能通过调用 API 函数发生更改。 如果在调用后需要 A 和 X 的值，则调用函数负责在调用前保留 A 和 X 的值。 选择此 “ 寄存器易失 ” 策略旨在提高效率，自 PSoC Designer 1.0 版起已强制使用此策略。 C 编译器自动遵循此要求。 汇编语言程序员也必须确保其代码遵守这一策略。 虽然一些用户模块 API 函数可以保留 A 和 X 不变，但是无法保证它们将来也会如此。

对于大型储存器模块驱动， 保存 CUR_PP、IDX_PP、MVR_PP 以及 MVW_PP 寄存器中的所有值也是调用程序的职责。 尽管部分寄存器现在可能不可修改，但是无法保证在将来的版本中也会如此。

## 常用函数

下面几个函数对主、从器件、及 I2CHW 用户模块的 MultiMasterSlave 是共用的：

### *I2CHW_Start*

**说明：**

不做任何操作。 只为保证接口一致性。

**C 原型：**

```
void  I2CHW_Start(void);
```

**汇编程序：**

```
lcall  I2CHW_Start
```

**参数：**

None

**返回值：**

None

**副作用：**

A 和 X 寄存器可能会因此函数的本次执行或以后执行而被修改 对于大内存模式下 (CY8C29xxx) 的所有 RAM 页面指针寄存器也是如此。 如果需要，调用函数负责通过调用 fastcall16 函数保存寄存器的值。

### *I2CHW_Stop*

**说明：**

通过禁用 $I^2C$ 中断，禁用 $I^2CHW$。

**C 原型：**

```
void   I2CHW_Stop(void);
```

**汇编程序：**

```
lcall  I2CHW_Stop
```

**参数：**

None

**返回值：**

None

**副作用：**

A 和 X 寄存器可能会因此函数的本次执行或以后执行而被修改。 对于大内存模式下 (CY8C29xxx) 的所有 RAM 页面指针寄存器也是如此。 如果需要，调用函数负责通过调用 fastcall16 函数保存寄存器的值。

### *I2CHW_EnableInt*

**说明：**

启用 $I^2C$ 中断以便实现启动条件检测。 请牢记要通过使用以下这个宏来调用全局中断使能函数：M8C_EnableGInt.

**C 原型：**

```
void   I2CHW_EnableInt(void);
```

**汇编程序：**

```
lcall I2CHW_EnableInt
```

**参数：**

None

**返回值：**

None

**副作用：**

A 和 X 寄存器可能会因此函数的本次执行或以后执行而被修改。 对于大内存模式下 (CY8C29xxx) 的所有 RAM 页面指针寄存器也是如此。 如果需要，调用函数负责通过调用 fastcall16 函数保存寄存器的值。

### *I2CHW_DisableInt*

**说明：**

通过禁用 SDA 中断来禁用 $I^2C$ 的从器件。 执行与 I2Cs_Stop. 相同的操作

**C 原型：**

```
void   I2CHW_DisableInt(void);
```

**汇编程序：**

```
lcall I2CHW_DisableInt
```

**参数：**

None

**返回值：**

None

**副作用：**

A 和 X 寄存器可能会因此函数的本次执行或以后执行而被修改。 对于大内存模式下 (CY8C29xxx) 的所有 RAM 页面指针寄存器也是如此。 如果需要，调用函数负责通过调用 fastcall16 函数保存寄存器的值。

### *I2CHW_Poll（适用于从器件及 MultiMasterSlave）*

**说明：**

在 Communication_ Service_Type 参数设置为 “Polled（轮询）” 时使用。 此函数提供了一个由用户控制的进入 I/O 处理例程的入口。 如果将 Communication_Service_Type 参数设置为 “ 中断 ”，则此函数不做任何操作。

注意： 调用 I2CHW_Poll 释放 I2C 总线。 这使得在从器件固件完成处理先前的请求之前，允许主器件可以读取或写入数据。

**C 原型：**

```
void   I2CHW_Poll(void);
```

**汇编程序：**

```
lcall I2CHW_Poll
```

**参数：**

None

**返回值：**

None

**副作用：**

每次调用该例程时，将处理一个 $I^2C$ 事件，以及状态变量将得到更新。 事件可以构成一个故障状况、一个 I/O 字节，或在某些特定情况下，一个停止状况。 调用该例程可能有三种结果：

1. 如果没有可用数据，则不执行任何操作。
2. 如果有一个可用地址或数据字节，则对此地址或数据进行接收或发送。
3. 如果外部主器件已完成其写入操作，则处理停止事件。 在写入操作结束阶段处理停止状态时，仅更新状态变量。 如果在处理停止状态时， $I^2C$ 字节处于待处理状态，则必须再次调用 I2CHW_Poll 函数对其进行处理。

I2CHW_Poll() 函数在 Communication_Service_Type 设置为 “ 中断 ”(Interrupt) 时不会产生任何作用。 在总线上检测到启动 / 重启条件和地址时，总线将处于停顿状态，直至 I2CHW_Poll() 函数被调用。 如果地址未被应答，则该数据操作发送的后续字节将会被忽略，直到检测到另一个启动 / 重启和地址，否则，$I^2C$ 总线上的每个数据字节都将处于停顿状态，直至调用 I2CHW_Poll() 函数。 在 Communication_Service_Type 设置为 “ 轮询 ”(Polled) ，并调用此函数之前， $I^2C$ 硬件控制 $I^2C$ 数据停顿。 对于已接收的数据，在字节末尾和生成 ACK/NAK （通过将 SCL （时钟）线保持在低位）之前，总线将停顿。 对于已发送的数据，在 ACK/NAK 位在生成之后，总线将立即停顿。

### *I2CHW_ResumeInt*

**说明：**

重新启用 $I^2C$ 中断以实现启动条件检测。 该 API 启用 INT_MSK3 寄存器中的 $I^2C$ 中断，且不清空 INT_CLR3 寄存器中的 $I^2C$ 中断。 请牢记要通过以下这个宏来调用这个全局中断使能函数：M8C_EnableGInt.

**C 原型：**

```
void I2CHW_ResumeInt(void);
```

**汇编程序：**

```
lcall I2CHW_ResumeInt
```

**参数：**

None

**返回值：**

None

**副作用：**

A 和 X 寄存器可能会因此函数的本次执行或以后执行而被修改。 对于大内存模式下 (CY8C29xxx) 的所有 RAM 页面指针寄存器也是如此。 如果需要，调用函数负责通过调用 fastcall16 函数保存寄存器的值。

### *I2CHW_ClearInt*

**说明：**

清空 INT_CLR3 寄存器中的 $I^2C$ 中断。 请牢记要通过以下这个宏来调用这个全局中断使能函数：M8C_EnableGInt.

**C 原型：**

```
void I2CHW_ClearInt(void);
```

**汇编程序：**

```
lcall I2CHW_ ClearInt
```

**参数：**

None

**返回值：**

None

**副作用：**

A 和 X 寄存器可能会因此函数的本次执行或以后执行而被修改。 对于大内存模式下 (CY8C29xxx) 的所有 RAM 页面指针寄存器也是如此。 如果需要，调用函数负责通过调用 fastcall16 函数保存寄存器的值。

## *API 低层通信函数 （主器件及 MultiMasterSlave 中可用）*

对于大多数应用，低层函数并非必需。 低层函数给专业应用提供了更大灵活性。 它们不使用 ISR。 大多数情况下，如果启用了 I2C 中断，则调用任一低层子程序就可确切地禁用它。

### I2CHW_fSendStart

**说明：**

生成 $I^2C$ 总线开始条件，发送地址和 R/W 位，然后返回 ACK 结果。 R/W 位由 fRW 参数决定。

**C 原型：**

```
BYTE I2CHW_fSendStart( BYTE bSlaveAddr, BYTE fRW );
```

**汇编程序：**

```
mov   A,0x68                  ; Load slave address
mov   X,I2CHW_WRITE           ; Prepare for a write sequence
lcall  I2CHW_fSendStart        ; Return value in A
```

**参数：**

bSlaveAddr：7-bit 从地址。 fRW： 如果设为 I2CHW_READ，则发起读序列。 如果设为 I2CHW_WRITE，则发起写序列。

**返回值：**

如果返回值非零，则从器件已确认地址。 如果返回值为零，则从器件不确认地址。

**副作用：**

A 和 X 寄存器可能会因此函数的本次执行或以后执行而被修改。 对于大内存模式下 (CY8C29xxx) 的所有 RAM 页面指针寄存器也是如此。 如果需要，调用函数负责通过调用 fastcall16 函数保存寄存器的值。 当前，仅修改 CUR_PP 页面指针寄存器。

如果前面已启用 I2CHW 中断，则该中断被禁用。

| 常量 | 值 | 说明 |
|---|---|---|
| I2CHW_WRITE | 0x00 | 启动 $I^2C$ 写序列 |
| I2CHW_READ | 0x01 | 启动 $I^2C$ 读序列 |
| I2CHW_ACKslave | 0x01 | 读一个字节时确认从器件 |
| I2CHW_NAKslave | 0x00 | 写一个字节时非应答从器件 |

### I2CHW_fSendRepeatStart

**说明：**

生成 $I^2C$ 总线重复开始条件，发送地址和 R/W 位，然后返回 ACK 结果。 R/W 位由 fRW 参数决定。

**C 原型：**

```
BYTE I2CHW_fSendRepeatStart( BYTE bSlaveAddr, BYTE fRW );
```

**汇编程序：**

```
mov   A,0x68                      ; Load address
mov   X,I2CHW_READ                 ; Prepare for a read sequence
lcall  I2CHW_fSendRepeatStart      ; Return value in A
```

**参数：**

bSlaveAddr：7-Bit 从地址。 fRW： 如果设为 I2CHW_READ，则启动读序列。 如果设为 I2CHW_WRITE，则启动写序列。

**返回值：**

如果返回值非零，则从器件已确认地址。 如果返回值为零，则从器件非应答地址。

**副作用：**

A 和 X 寄存器可能会因此函数的本次执行或以后执行而被修改。 对于大内存模式下 (CY8C29xxx) 的所有 RAM 页面指针寄存器也是如此。 如果需要，调用函数负责通过调用 fastcall16 函数保存寄存器的值。 当前，仅修改 CUR_PP 页面指针寄存器。 如果前面已启用 I2CHW 中断，则中断被禁用。

在 bStatus 字节中设置 RepStart 标志。 这样能防止在轮询界面使用时，轮询子程序被挂起。 注意避免混淆缓冲命令和低层命令。 通过轮询界面选项，特别是如果进入该子程序而在 I2C_SCR 寄存器中未设置 “ 字节完成 ” 标志，则该标志不存在。

### I2CHW_SendStop

**说明：**

生成 $I^2C$ 停止条件。

**C 原型：**

```
void I2CHW_SendStop( void );
```

**汇编程序：**

```
lcall I2CHW_SendStop        ; Generate I2C stop condition
```

**参数：**

None

**返回值：**

None

**副作用：**

A 和 X 寄存器可能会因此函数的本次执行或以后执行而被修改。 对于大内存模式下 (CY8C29xxx) 的所有 RAM 页面指针寄存器也是如此。 如果需要，调用函数负责通过调用 fastcall16 函数保存寄存器的值。

### I2CHW_fWrite

**说明：**

发送单字节 $I^2C$ 总线写操作和 ACK。 此函数不生成开始或停止条件。 只有在 I2C 总线上生成了上一个开始和地址时，“ 才 ” 应调用该子程序。 只有当在 I2C_BYTE_COMPL 寄存器中设置了 I2C_SCR 时，才能使用它。

**C 原型：**

```
BYTE I2CHW_fWrite( BYTE bData );
```

**汇编程序：**

```
mov   A,[bRamData]             ; Load data to send to slave
lcall  I2CHW_fWrite             ; Initiate I2C write
```

**参数：**

bData： 要发送给从器件的字节。

**返回值：**

如果从器件应答主器件，则返回值为非零。 如果从器件未应答主器件，返回值为零。 如果从器件未确认主器件，则 bStatus 的值为 0xff。

**副作用：**

A 和 X 寄存器可能会因此函数的本次执行或以后执行而被修改。 对于大内存模式下 (CY8C29xxx) 的所有 RAM 页面指针寄存器也是如此。 如果需要，调用函数负责通过调用 fastcall16 函数保存寄存器的值。 当前，仅修改 CUR_PP 页面指针寄存器。 如果前面已启用 I2CHW 中断，则中断被禁用。

### I2CHW_bRead

**说明：**

启动单字节 $I^2C$ 总线读取操作和 ACK 阶段。 此函数不生成开始或停止条件。 fACK 标志确认在收到数据时是否应答从器件。 只有在 I2C 总线上生成了上一个开始和地址时，“ 才 ” 应调用该子程序。 只有当在 I2C_BYTE_COMPL 寄存器中设置了 I2C_SCR 时，才能使用它。 如果设置了 fACK，则其后面应跟下一个 I2CHW_bRead 调用。 要完成读数据操作，主器件应 调用带 I2CHW_NAKslave 参数的该函数。

**C 原型：**

```
BYTE I2CHW_bRead( BYTE fACK );
```

**汇编程序：**

```
mov   A,I2CHW_ACKslave        ; Set flag to ACK slave
lcall  I2CHW_bRead             ; Read single byte from slave
                              ; Return data is in reg A
```

**参数：**

fACK： 如果主器件收到数据后应答从器件，则设置为 I2CHW_ACKslave ; 否则，标志应被设置为 I2CHW_NAKslave.。 通常，来自主器件的 ACK 表示要从从器件获得下一数据字节。 如果被设置为 I2CHW_ACKslave，则在接收到当前数据字节并确认后，主器件将立即记录来自从器件的下一字节。 在下次调用 I2CHW_bRead() 时，将返回该下一字节。 如果被设置为 I2CHW_NAKslave，主器件将不应答当前字节，且不会记录数据的下一字节。

**返回值：**

从从器件接收的字节。

**副作用：**

A 和 X 寄存器可能会因此函数的本次执行或以后执行而被修改。 对于大内存模式下 (CY8C29xxx) 的所有 RAM 页面指针寄存器也是如此。 如果需要，调用函数负责通过调用 fastcall16 函数保存寄存器的值。 当前，仅修改 CUR_PP 页面指针寄存器。 如果前面已启用 I2CHW 中断，则中断被禁用。

## 从器件函数

下面的函数仅针对 I2CHW 用户模块的从器件版本。

### *I2CHW_EnableSlave*

**说明：**

通过设置 I2C_CFG 寄存器中的 “Enable Slave”（启用从器件）位，为 $I^2C$ HW 模块启用 $I^2C$ Slave 函数。

**C 原型：**

```
void   I2CHW_EnableSlave(void);
```

**汇编程序：**

```
lcall  I2CHW_EnableSlave
```

**参数：**

None

**返回值：**

None

**副作用：**

A 和 X 寄存器可能会因此函数的本次执行或以后执行而被修改。 对于大内存模式下 (CY8C29xxx) 的所有 RAM 页面指针寄存器也是如此。 如果需要，调用函数负责通过调用 fastcall16 函数保存寄存器的值。 如 CY8C27xxxA 数据手册中所述，$I^2C$ 配置寄存器应以 6 MHz 的 CPU 时钟速率写入。 此处所实施的子程序会进行该项操作。 如果直接写入配置寄存器，需确保正确执行操作，否则可能发生 $I^2C$ 通信故障。

### *I2CHW_DisableSlave*

**说明：**

通过清除 I2C_CFG 寄存器中的 “Enable Slave” （使能从器件）位，禁用 $I^2C$ Slave 函数。

**C 原型：**

```
void   I2CHW_DisableSlave(void);
```

**汇编程序：**

```
lcall  I2CHW_DisableSlave
```

**参数：**

None

**返回值：**

None

**副作用：**

A 和 X 寄存器可能会因此函数的本次执行或以后执行而被修改。 对于大内存模式下 (CY8C29xxx) 的所有 RAM 页面指针寄存器也是如此。 如果需要，调用函数负责通过调用 fastcall16 函数保存寄存器的值。 如 CY8C27xxxA 数据手册中所述，$I^2C$ 配置寄存器应以 6 MHz 的 CPU 时钟速率写入。 此处所实施的子程序会进行该项操作。 如果直接写入配置寄存器，需确保正确执行操作，否则可能发生 $I^2C$ 通信故障。

### *I2CHW_bReadI2CStatus*

**说明：**

返回控制 / 状态寄存器的状态位。

**C 原型：**

```
BYTE  I2CHW_bReadI2CStatus(void);
```

**汇编程序：**

```
lcall  I2CHW_bReadI2CStatus ; Accumulator contains status
```

**参数：**

None

**返回值：**

```
BYTE  I2CHW_RsrcStatus
```

**副作用：**

A 和 X 寄存器可能会因此函数的本次执行或以后执行而被修改。 对于大内存模式下 (CY8C29xxx) 的所有 RAM 页面指针寄存器也是如此。 如果需要，调用函数负责通过调用 fastcall16 函数保存寄存器的值。 当前，仅修改 CUR_PP 页面指针寄存器。

| 常量 | 值 | 说明 |
|---|---|---|
| I2CHW_RD_NOERR | 01h | 主器件读取数据，正常 ISR 退出 |
| I2CHW_RD_OVERFLOW | 02h | 主器件所读取的数据字节超出了可用字节数 |
| I2CHW_RD_COMPLETE | 04h | 读取操作已经开始且完成 |
| I2CHW_READFLASH | 08h | 下一个读取操作来自闪存位置 |
| I2CHW_WR_NOERR | 10h | 主器件成功写入了数据 |
| I2CHW_WR_OVERFLOW | 20h | 主器件向写入缓冲区写入了过多字节 |
| I2CHW_WR_COMPLETE | 40h | 主器件写入操作因新地址或停止位而结束 |
| I2CHW_ISR_ACTIVE | 80h | $I^2C$ ISR 尚未退出并处于活动状态 |

### *I2CHW_ClrRdStatus*

**说明：**

清除 I2CHW_RsrcStatus 寄存器中的读取状态位。 不影响其他位。

**C 原型：**

```
void  I2CHW_ClrRdStatus (void);
```

**汇编程序：**

```
lcall  I2CHW_ClrRdStatus
```

**参数：**

None

**返回值：**

None

**副作用：**

A 和 X 寄存器可能会因此函数的本次执行或以后执行而被修改。 对于大内存模式下 (CY8C29xxx) 的所有 RAM 页面指针寄存器也是如此。 如果需要，调用函数负责通过调用 fastcall16 函数保存寄存器的值。 当前，仅修改 CUR_PP 页面指针寄存器。

### *I2CHW_ClrWrStatus*

**说明：**

清除 I2CHW_RsrcStatus 寄存器中的写入状态位。 不影响其他位。

**C 原型：**

```
void  I2CHW_ClrWrStatus (void);
```

**汇编程序：**

```
lcall  I2CHW_ClrWrStatus
```

**参数：**

None

**返回值：**

None

**副作用：**

A 和 X 寄存器可能会因此函数的本次执行或以后执行而被修改。 对于大内存模式下 (CY8C29xxx) 的所有 RAM 页面指针寄存器也是如此。 如果需要，调用函数负责通过调用 fastcall16 函数保存寄存器的值。 当前，仅修改 CUR_PP 页面指针寄存器。

### *I2CHW_InitWrite*

**说明：**

初始化从器件的数据缓冲区指针用于存放数据，并初始化该缓冲区的字节计数值。 将计数值初始化至提供的最大缓冲区长度。 在下一个主器件写入的实例中，将数据放置在由此函数定义的地址。

**C 原型：**

```
void   I2CHW_InitWrite(BYTE * pWriteBuf, BYTE bBufLen);
```

**汇编程序：**

```
AREA    bss (RAM, REL)
abWriteBuf    blk 10h

AREA    text (ROM,REL)
    push X                          ; save registers
    push A
    add  SP, 3
    mov  X, SP
    dec  X                          ; X points at data SP points at next
                                    ; empty stack location
    mov  [X], <abWriteBuf           ; place the buffer address
    mov  [X-1], >abWriteBuf         ; (page 0)on the stack at [X]
    mov  [X-2], 10                  ; place the count at [x-2]
                                    ; don't care what [X-1] is
                                    ; the compiler would assign 0 as the
                                    ; MSB of the Ramtbl addr
    lcall  I2CHW_InitWrite
    add   SP, -3                    ; restore the stack
    pop   A                         ; restore registers
    pop   X
```

**参数：**

pWriteBuf： 指向 RAM 缓冲区位置的指针。 buf_len： 写缓冲区的长度。

**返回值：**

None

**副作用：**

A 和 X 寄存器可能会因此函数的本次执行或以后执行而被修改。 对于大内存模式下 (CY8C29xxx) 的所有 RAM 页面指针寄存器也是如此。 如果需要，调用函数负责通过调用 fastcall16 函数保存寄存器的值。 当前，仅修改 CUR_PP 页面指针寄存器。

### *I2CHW_InitRamRead*

**说明：**

初始化闪存数据缓冲区指针 —— 从器件从该指针读取数据 —— 并初始化相同缓冲区的计数字节值。清除 I2CHW_SlaveStatus 标志 I2CHW_READFLASH 为 0，会使从 RAM 中*先前*设置的缓冲区位置进行下一次读取操作。

**C 原型：**

```
void   I2CHW_InitRamRead(BYTE * pReadBuf, BYTE bBufLen);
```

**汇编程序：**

```
AREA bss (RAM,REL)
abReadBuf:    blk 10h

AREA text (ROM,REL)
    push  X                         ; save registers
    push  A
add   SP, 3
    mov   X, SP
    dec   X                         ; X points at data SP points at next
                                    ; empty stack location
    mov   [X], <abReadBuf           ; place the read buffer address
    mov   [X-1], >abReadBuf         ; (page0)on the stack at [X]
    mov   [X-2], 10                 ; place the count at [x-2]
                                    ; don't care what [X-1] is
                                    ; the compiler would assign 0 as
                                    ; the MSB of the Ramtbl addr
    lcall  I2CHW_InitRamRead
    add   SP, -3                    ; back up the stack (subtract 3)
    pop   A                         ; restore registers
    pop   X
```

**参数：**

_ReadBuf： 指向 RAM 缓冲区位置的指针。bBufLen： 读取缓冲区的长度。

**返回值：**

None

**副作用：**

A 和 X 寄存器可能会因此函数的本次执行或以后执行而被修改 。对于大内存模式下 (CY8C29xxx) 的所有 RAM 页面指针寄存器也是如此。 如果需要，调用函数负责通过调用 fastcall16 函数保存寄存器的值。 当前，仅修改 CUR_PP 页面指针寄存器。

### *I2CHW_InitFlashRead*

**说明：**

初始化闪存数据缓冲区指针，用于读取数据。 设置 I2CHW_SlaveStatus 标志 I2CHW_READFLASH 为 1，致使从闪存中*前面*设置的缓冲区位置尝试进行下一个读取操作。

**C 原型：**

```
void   I2CHW_InitFlashRead(const BYTE * pFlashBuf, WORD wBufLen);
```

**汇编程序：**

```
area table(ROM,ABS)
org 0x1015

abFlashBuf:

      db 0x11, 0x22, 0x33, 0x44, 0x55, 0x66, 0x77, 0x88
      db 0x99, 0xaa, 0xbb, 0xcc, 0xdd, 0xee, 0xff


area text(ROM,REL)


    push  X                          ; save registers
    push  A
    add   SP, 4
    mov   X, SP
    dec   X                          ; X points at data SP points at next
                            ; empty stack location
    mov   [X], <abFlashBuf           ; place the LSB of rom
                            ; address on the stack at [X]
    mov   [X-1], >abFlashBuf         ; place the MSB of the rom address
                            ; at [x-1] variable
    mov   [X-2], 0x0F                ; place the LSB of length
                                     ; at [x-2]
    mov   [X-3], 0x00                ; place the MSB of length
                            ; at [x-3]
    lcall  I2CHW_InitFlashRead
    add   SP, -4                     ; adjust the stack (subtract 4)
    pop   A                          ; restore registers
    pop   X
```

**参数：**

pFlashBuf： 指向闪存 /ROM 缓冲区位置的指针。wBufLen： 缓冲区长度。

**返回值：**

None

**副作用：**

A 和 X 寄存器可能会因此函数的本次执行或以后执行而被修改。 对于大内存模式下 (CY8C29xxx) 的所有 RAM 页面指针寄存器也是如此。 如果需要，调用函数负责通过调用 fastcall16 函数保存寄存器的值。 当前，仅修改 CUR_PP 页面指针寄存器。

## 主器件函数

下面的函数仅针对 I2CHW 用户模块的主器件版本。

**Note** 在给定 API 内，支持两种不同的传递 API 参数的方法。 以前的参数传递方法版本在后来的 C 编译器版本上已废弃不用。 只有在小内存模型器件上使用汇编代码和老 API 调用结构的用户，在他们升级应用程序到大内存模型器件时，才会感觉到这一变化的影响。 通过对汇编调用语句添加下划线，新的应用程序可在汇编语言实现中使用本节描述的调用结构 （新型参数传递）。 对以 C 语言编写的应用程序无任何影响。 这仅适用于 I2CHW 用户模块的主器件版本。 符合本说明的子程序有: I2CHW_fReadBytes、 I2CHW_bWriteBytes、及 I2CHW_bWriteCBytes.

### *I2CHW_EnableMstr*

**说明：**

通过设置 I2C_CFG 寄存器中的 Enable Master 位，将 $I^2C$ HW 模块作为 Master 启用。

**C 原型：**

```
void   I2CHW_EnableMstr(void);
```

**汇编程序：**

```
lcall I2CHW_EnableMstr
```

**参数：**

None

**返回值：**

None

**副作用：**

A 和 X 寄存器可能会因此函数的本次执行或以后执行而被修改。 对于大内存模式下 (CY8C29xxx) 的所有 RAM 页面指针寄存器也是如此。 如果需要，调用函数负责通过调用 fastcall16 函数保存寄存器的值。

如 CY8C27xxxA 数据手册中所述，$I^2C$ 配置寄存器应以 6 MHz 的 CPU 时钟速率写入。 此处所实施的子程序会进行该项操作。 如果直接写入配置寄存器，需确保正确执行操作，否则可能发生 $I^2C$ 通信故障。

### *I2CHW_DisableMstr*

**说明：**

通过清除 I2C_CFG 寄存器中的 “Enable Slave” （使能从器件）位，禁用 $I^2C$ Master 函数。

**C 原型：**

```
void   I2CHW_DisableMstr(void);
```

**汇编程序：**

```
lcall  I2CHW_DisableMstr
```

**参数：**

None

**返回值：**

None

**副作用：**

A 和 X 寄存器可能会因此函数的本次执行或以后执行而被修改。 对于大内存模式下 (CY8C29xxx) 的所有 RAM 页面指针寄存器也是如此。 如果需要，调用函数负责通过调用 fastcall16 函数保存寄存器的值。 如 CY8C27xxxA 数据手册中所述，$I^2C$ 配置寄存器应以 6 MHz 的 CPU 时钟速率写入。 此处所实施的子程序会进行该项操作。 如果直接写入配置寄存器，需确保正确执行操作，否则可能发生 $I^2C$ 通信故障。

### *I2CHW_bReadI2CStatus*

**说明：**

返回控制 / 状态寄存器的状态位。

**C 原型：**

```
BYTE  I2CHW_bReadI2CStatus(void);
```

**汇编程序：**

```
lcall  I2CHW_bReadI2CStatus ; Accumulator contains status
```

**参数：**

None

**返回值：**

```
BYTE  I2CHW_RsrcStatus
```

参见表格。

**副作用：**

A 和 X 寄存器可能会因此函数的本次执行或以后执行而被修改。 对于大内存模式下 (CY8C29xxx) 的所有 RAM 页面指针寄存器也是如此。 如果需要，调用函数负责通过调用 fastcall16 函数保存寄存器的值。 当前，仅修改 CUR_PP 页面指针寄存器。

| 常量 | 值 | 说明 |
|---|---|---|
| I2CHW_RD_NOERR | 01h | 主器件读取数据，正常 ISR 退出 |
| I2CHW_RD_ OVERFLOW | 02h | 主器件所读取的数据字节超出了可用字节数 |
| I2CHW_RD_ COMPLETE | 04h | 启动了读操作，但尚未完成。 |
| I2CHW_READFLASH | 08h | 下一个读取操作来自闪存位置 |
| I2CHW_WR_NOERR | 10h | 主器件成功写入了数据 |
| I2CHW_WR_OVERFLOW | 20h | 主器件向写入缓冲区写入了过多字节 |
| I2CHW_WR_COMPLETE | 40h | 主器件写入操作因新地址或停止位而结束 |
| I2CHW_ISR_ACTIVE | 80h | $I^2C$ ISR 尚未退出并处于活动状态 |

### *I2CHW_ClrRdStatus*

**说明：**

清除 I2CHW_RsrcStatus 寄存器中的读取状态位。 不影响其他位。

**C 原型：**

```
void  I2CHW_ClrRdStatus (void);
```

**汇编程序：**

```
lcall  I2CHW_ClrRdStatus
```

**参数：**

None

**返回值：**

None

**副作用：**

A 和 X 寄存器可能会因此函数的本次执行或以后执行而被修改。 对于大内存模式下 (CY8C29xxx) 的所有 RAM 页面指针寄存器也是如此。 如果需要，调用函数负责通过调用 fastcall16 函数保存寄存器的值。 当前，仅修改 CUR_PP 页面指针寄存器。

### *I2CHW_ClrWrStatus*

**说明：**

清除 I2CHW_RsrcStatus 寄存器中的写入状态位。 不影响其他位。

**C 原型：**

```
void  I2CHW_ClrWrStatus (void);
```

**汇编程序：**

```
lcall  I2CHW_ClrWrStatus
```

**参数：**

None

**返回值：**

None

**副作用：**

A 和 X 寄存器可能会因此函数的本次执行或以后执行而被修改。 对于大内存模式下 (CY8C29xxx) 的所有 RAM 页面指针寄存器也是如此。 如果需要，调用函数负责通过调用 fastcall16 函数保存寄存器的值。 当前，仅修改 CUR_PP 页面指针寄存器。

### *I2CHW_fReadBytes*

**说明：**

这是一个 Master 函数。 它启动与可寻址的从器件进行的 Read 读数据操作。 从 $I^2C$ 从器件读取一个或多个字节（bCnt），并向 pbXferData 所指向的数组写入该数据。 一旦调用了该子程序，内含的 ISR 即处理更多数据。

**C 原型：**

```
void I2CHW_fReadBytes(BYTE bSlaveAddr, BYTE * pbXferData, BYTE bCnt, BYTE bMode);
```

**汇编程序：**

```
mov   A,I2CHW_CompleteXfer   ; Pass complete transfer flag
push  A
mov   A,0x09                 ; Pass the byte count
push  A
mov   A,>sData               ; Load the MSB of the sData pointer
push  A
mov   A,<sData               ; Load the LSB of the sData pointer
push  A

mov   A,0x68                 ; Pass slave address 0x68
push  A
```

```
lcall  _I2CHW_fReadBytes      ; lcall function to read data from slave
                              ; Reg A contains return value.
add   sp,-5 ; Restore the stack
```

**参数：**

bSlaveAddr：7-bit 从器件地址。pbXferData： 指向 RAM 中数据数组的指针， bCnt ： 要读取的数据个数。bMode： 操作模式： 如果模式被设置为 I2CHW_CompleteXfer，则执行完整传输。 如果模式被设置为 I2CHW_RepStart、或被设置为 I2CHW_NoStop，则不生成停止位。 这可实现向从器件发送 $I^2C$ 总线的合并传输。 通过重复开始，可发起后续传输。 见本节末表格。

**返回值：**

None。

**副作用：**

A 和 X 寄存器可能会因此函数的本次执行或以后执行而被修改。 对于大内存模式下 (CY8C29xxx) 的所有 RAM 页面指针寄存器也是如此。 如果需要，调用函数负责通过调用 fastcall16 函数保存寄存器的值。 当前，仅修改 CUR_PP 页面指针寄存器。

### *I2CHW_bWriteBytes*

**说明：**

这是一个 Master 函数。 它启动与可寻址从器件进行的 Write 写数据操作。 将 pbXferData 所指向的 RAM 数组的一个或多个字节 (bCnt) 写向 (bSlaveAddr) 编址的从器件中。 一旦调用了该子程序，内含的 ISR 即处理更多数据。

**C 原型：**

```
void I2CHW_bWriteBytes(BYTE bSlaveAddr, BYTE * pbXferData, BYTE bCnt, BYTE bMode);
```

**汇编程序：**

```
mov   A,I2CHW_CompleteXfer   ; Pass complete transfer flag
push  A
mov   A,0x09                 ; Pass the byte count
push  A
mov   A,>sData               ; Load the MSB of the sData pointer
push  A
mov   A,<sData               ; Load the LSB of the sData pointer
push  A
mov   A,0x68                 ; Pass slave address 0x68
push  A
lcall  _I2CHW_bWriteBytes     ; lcall function to write data to slave
                              ; Reg A contains return value.
add   sp,-5 ; Restore the stack
```

**参数：**

bSlaveAddr：7-bit 从器件地址。pbXferData： 指向 RAM 中数据数组的指针。bCnt： 要写入的数据的个数。bMode： 操作模式： 如果模式被设置为 I2CHW_CompleteXfer，则执行完整传输。 如果模式被设置为 I2CHW_RepStart、或被设置为 I2CHW_NoStop，则不生成停止位。 这可实现向从器件发送 $I^2C$ 总线的合并传输。 通过重复开始，可启动后续传输。 见本节末表格。

**返回值：**

None

**副作用：**

A 和 X 寄存器可能会因此函数的本次执行或以后执行而被修改。 对于大内存模式下 (CY8C29xxx) 的所有 RAM 页面指针寄存器也是如此。 如果需要，调用函数负责通过调用 fastcall16 函数保存寄存器的值。 当前，仅修改 CUR_PP 页面指针寄存器。

### *I2CHW_bWriteCBytes*

**说明：**

这是一个 Master 函数。 它启动与可寻址从器件进行的 Write 写入数据操作。 将 指针 (pbXferData) 指向的常量 flash 数组的 一个或多个字节 (bCnt) 写入到 bSlaveAddr 编址的从器件中。 一旦此子函数启动数据传输，包含的 ISR 处理进一步的数据传输。

**C 原型：**

```
void I2CHW_bWriteCBytes(BYTE bSlaveAddr, const BYTE * pbXferData, BYTE bCnt, BYTE
bMode);
```

**汇编程序：**

```
mov   A,I2CHW_CompleteXfer    ; Pass complete transfer flag
push  A
mov   A,0x09                  ; Pass the byte count
push  A
mov   A,>sData                ; Load the MSB of the sData pointer
push  A
mov   A,<sData                ; Load the LSB of the sData pointer
push  A
mov   A,0x68                  ; Pass slave address 0x68
push  A
lcall  _I2CHW_bWriteCBytes    ; lcall function to write data to slave
                              ; A contains return value.
add   sp,-5 ; Restore the stack
```

**参数：**

bSlaveAddr：7-bit 从器件地址。pbXferData： 指向闪存 “const” 数据数组的指针。bCnt： 要写入的数据的个数。bMode： 操作模式： 如果模式被设置为 I2CHW_CompleteXfer，则执行完整传输。 如果模式被设置为 I2CHW_RepStart、或被设置为 I2CHW_NoStop，则不生成停止位。 这可实现向从器件发送 $I^2C$ 总线的合并传输。 通过重复开始，可启动后续数据传输。 见本节末表格。

**返回值：**

None。

**副作用：**

A 和 X 寄存器可能会因此函数的本次执行或以后执行而被修改。 对于大内存模式下 (CY8C29xxx) 的所有 RAM 页面指针寄存器也是如此。 如果需要，调用函数负责通过调用 fastcall16 函数保存寄存器的值。 当前，仅修改 CUR_PP 页面指针寄存器。

**Note** 可以使用 bMode 参数来执行 $I^2C$ 总线合并格式传输。 要执行合并传输，需先执行 I2CHW_bWriteBytes 或 I2CHW_bWriteCBytes 命令，同时将 bMode 参数设置为 I2CHW_NoStop (0x02)。 这可执行写操作而无需停止位。 接下来，执行 I2CHW_fReadBytes 命令，并将参数设置为 I2CHW_RepStart (0x01)。

| 常量 | 值 | 说明 |
|---|---|---|
| I2CHW_CompleteXfer | 0x00 | 从头至尾执行完整的传输。 |
| I2CHW_RepStart | 0x01 | 发送重复启动 (Repeat Start) 而不是单个启动 (Start)。 |
| I2CHW_NoStop | 0x02 | 执行传输，而没有停止位。 |

### *I2CHW_bReadBusStatus*

**说明：**

读 I2CHW_bStatus 字节的当前值。 这是 I2CHW 用户模块使用的 API 函数及 ISR 的内部状态字节。 没有给出任何工具来更改 theI2CHW_bStatus 字节中的数据。

**C 原型：**

```
BYTE I2CHW_bReadBusStatus (void);
```

**汇编程序：**

```
lcall  I2CHW_bReadBusStatus
```

**参数：**

None

**返回值：**

```
BYTE I2CHW_bStatus
```

详细说明见下表。 这些定义报告的是状态变量，而非状态 / 控制位。

| 常量 | 值 | 说明 |
|---|---|---|
| RepStart, NoStop | 01h, 02h | 预留用于传输选项 CompleteXfer/RepStart/NoStop |
| NAKnextWr | 04h | 标志，告诉从器件不需应答来自主器件的下一字节 |

**副作用：**

A 和 X 寄存器可能会由此函数的本次执行或以后执行而被修改。 对于大内存模式下 (CY8C29xxx) 的所有 RAM 页面指针寄存器也是如此。 如果需要，调用函数负责通过调用 fastcall16 函数保存寄存器的值。

## 多主控从器件函数

下面各函数仅针对 I2CHW 用户模块的 MultiMasterSlave 版本。

### *I2CHW_EnableSlave*

**说明：**

通过设置 I2C_CFG 寄存器中的 “Enable Slave”（启用从器件）位，为 $I^2C$ HW 模块启用 $I^2C$ Slave 函数。

**C 原型：**

```
void   I2CHW_EnableSlave(void);
```

**汇编程序：**

```
lcall  I2CHW_EnableSlave
```

**参数：**

None

**返回值：**

None

**副作用：**

A 和 X 寄存器可能会因此函数的本次执行或以后执行而被修改。 对于大内存模式下 (CY8C29xxx) 的所有 RAM 页面指针寄存器也是如此。 如果需要，调用函数负责通过调用 fastcall16 函数保存寄存器的值。 如 CY8C27xxxA 数据手册中所述，$I^2C$ 配置寄存器应以 6 MHz 的 CPU 时钟速率写入。 此处所实施的子程序会进行该项操作。 如果直接写入配置寄存器，需确保正确执行操作，否则可能发生 $I^2C$ 通信故障。

### *I2CHW_DisableSlave*

**说明：**

通过清除 I2C_CFG 寄存器中的 “Enable Slave” （使能从器件）位，禁用 $I^2C$ Slave 函数。

**C 原型：**

```
void   I2CHW_DisableSlave(void);
```

**汇编程序：**

```
lcall  I2CHW_DisableSlave
```

**参数：**

None

**返回值：**

None

**副作用：**

A 和 X 寄存器可能会因此函数的本次执行或以后执行而被修改。 对于大内存模式下 (CY8C29xxx) 的所有 RAM 页面指针寄存器也是如此。 如果需要，调用函数负责通过调用 fastcall16 函数保存寄存器的值。 如 CY8C27xxxA 数据手册中所述，$I^2C$ 配置寄存器应以 6 MHz 的 CPU 时钟速率写入。 此处所实施的子程序会进行该项操作。 如果直接写入配置寄存器，需确保正确执行操作，否则可能发生 $I^2C$ 通信故障。

### *I2CHW_EnableMstr*

**说明：**

通过设置 I2C_CFG 寄存器中的 Enable Master 位，将 $I^2C$ HW 模块作为 Master 启用。

**C 原型：**

```
void   I2CHW_EnableMstr(void);
```

**汇编程序：**

```
lcall I2CHW_EnableMstr
```

**参数：**

None

**返回值：**

None

**副作用：**

A 和 X 寄存器可能会因此函数的本次执行或以后执行而被修改。 对于大内存模式下 (CY8C29xxx) 的所有 RAM 页面指针寄存器也是如此。 如果需要，调用函数负责通过调用 fastcall16 函数保存寄存器的值。 如 CY8C27xxxA 数据手册中所述，$I^2C$ 配置寄存器应以 6 MHz 的 CPU 时钟速率写入。 此处所实施的子程序会进行该项操作。 如果直接写入配置寄存器，需确保正确执行操作，否则可能发生 $I^2C$ 通信故障。

### *I2CHW_DisableMstr*

**说明：**

通过清除 I2C_CFG 寄存器中的 “Enable Slave” （使能从器件）位，禁用 $I^2C$ Master 函数。

**C 原型：**

```
void   I2CHW_DisableMstr(void);
```

**汇编程序：**

```
lcall  I2CHW_DisableMstr
```

**参数：**

None

**返回值：**

None

**副作用：**

A 和 X 寄存器可能会因此函数的本次执行或以后执行而被修改。 对于大内存模式下 (CY8C29xxx) 的所有 RAM 页面指针寄存器也是如此。 如果需要，调用函数负责通过调用 fastcall16 函数保存寄存器的值。 如 CY8C27xxxA 数据手册中所述，$I^2C$ 配置寄存器应以 6 MHz 的 CPU 时钟速率写入。 此处所实施的子程序会进行该项操作。 如果直接写入配置寄存器，需确保正确执行操作，否则可能发生 $I^2C$ 通信故障。

### *I2CHW_InitSlaveWrite*

**说明：**

初始化数据缓冲区指针，在从器件模式下，供 MultiMasterSlave 存放数据，并初始化该缓冲区的字节计数值。 将计数值初始化为提供的最大缓冲区长度。 在下一个主器件写入的实例中，将数据放置在由此函数定义的地址。

**C 原型：**

```
void   I2CHW_InitSlaveWrite(BYTE * pWriteBuf, BYTE bBufLen);
```

**汇编程序：**

```
AREA    bss (RAM, REL)
```

```
abWriteBuf    blk 10h

AREA    text (ROM,REL)
    push X                          ; save registers
    push A
    add  SP, 3
    mov  X, SP
    dec  X                          ; X points at data SP points at next
                        ; empty stack location
    mov  [X], <abWriteBuf           ; place the buffer address
    mov  [X-1], >abWriteBuf         ; (page 0)on the stack at [X]
    mov  [X-2], 10                  ; place the count at [x-2]
                                    ; don't care what [X-1] is
                                    ; the compiler would assign 0 as the
                                ; MSB of the Ramtbl addr
    lcall  I2CHW_InitSlaveWrite
    add   SP, -3                    ; restore the stack
    pop   A                         ; restore registers
    pop   X
```

**参数：**

pWriteBuf： 指向 RAM 缓冲区位置的指针。 buf_len： 写入缓冲区的长度。

**返回值：**

None

**副作用：**

A 和 X 寄存器可能因此函数的本次执行或以后执行而被修改。 对于大内存模式下 (CY8C29xxx) 的所有 RAM 页面指针寄存器也是如此。 如果需要，调用函数负责通过调用 fastcall16 函数保存寄存器的值。 当前，仅修改 CUR_PP 页面指针寄存器。

### *I2CHW_InitSlaveRamRead*

**说明：**

初始化 RAM 数据缓冲区指针，用于处于从器件模式的 MultiMasterSlave 读取数据，并初始化相同缓冲区字节的计数值。 将 I2CHW_SlaveStatus 标记 I2CHW_READFLASH 清除为零 （仅限读取状态位），从 RAM 中*先前*设置的缓冲区位置尝试进行下一次读取操作。

**C 原型：**

```
void   I2CHW_InitSlaveRamRead(BYTE * pReadBuf, BYTE bBufLen);
```

**汇编程序：**

```
AREA bss (RAM,REL)
abReadBuf:    blk 10h

AREA text (ROM,REL)
    push  X                         ; save registers
    push  A
add   SP, 3
    mov   X, SP
    dec   X                         ; X points at data SP points at next
                                    ; empty stack location
    mov   [X], <abReadBuf           ; place the read buffer address
    mov   [X-1], >abReadBuf         ; (page0)on the stack at [X]
```

```
    mov   [X-2], 10               ; place the count at [x-2]
                                  ; don't care what [X-1] is
                                  ; the compiler would assign 0 as
                                  ; the MSB of the Ramtbl addr
    lcall  I2CHW_InitRamRead
    add   SP, -3                  ; back up the stack (subtract 3)
    pop   A                       ; restore registers
    pop   X
```

**参数：**

_ReadBuf： 指向 RAM 缓冲区位置的指针。bBufLen： 读取缓冲区的长度。

**返回值：**

None

**副作用：**

A 和 X 寄存器可能会因此函数的本次执行或以后执行而被修改。 对于大内存模式下 (CY8C29xxx) 的所有 RAM 页面指针寄存器也是如此。 如果需要，调用函数负责通过调用 fastcall16 函数保存寄存器的值。 当前，仅修改 CUR_PP 页面指针寄存器。

### *I2CHW_InitSlaveFlashRead*

**说明：**

初始化闪存数据缓冲区指针，用于读取数据。 设置 I2CHW_SlaveStatus 标志 I2CHW_READFLASH 为 1，将会从闪存中*前面*设置的缓冲区位置尝试进行下一个读取操作。

**C 原型：**

```
void   I2Cs_InitSlaveFlashRead(const BYTE * pFlashBuf, WORD wBufLen);
```

**汇编程序：**

```
area table(ROM,ABS)
org 0x1015

abFlashBuf:

      db 0x11, 0x22, 0x33, 0x44, 0x55, 0x66, 0x77, 0x88
      db 0x99, 0xaa, 0xbb, 0xcc, 0xdd, 0xee, 0xff


area text(ROM,REL)


    push  X                       ; save registers
    push  A
    add   SP, 4
    mov   X, SP
    dec   X                       ; X points at data SP points at next
                          ; empty stack location
    mov   [X], <abFlashBuf        ; place the LSB of rom
                          ; address on the stack at [X]
    mov   [X-1], >abFlashBuf      ; place the MSB of the rom address
                          ; at [x-1] variable
    mov   [X-2], 0x0F             ; place the LSB of length
                                  ; at [x-2]
```

```
    mov   [X-3], 0x00                ; place the MSB of length
                           ; at [x-3]
    lcall  I2CHW_InitSlaveFlashRead
    add   SP, -4                     ; adjust the stack (subtract 4)
    pop   A                          ; restore registers
    pop   X
```

**参数：**

pFlashBuf： 指向闪存 /ROM 缓冲区位置的指针。wBufLen： 缓冲区长度。

**返回值：**

None

**副作用：**

A 和 X 寄存器可能会因此函数的本次执行或以后执行而被修改。 对于大内存模式下 (CY8C29xxx) 的所有 RAM 页面指针寄存器也是如此。 如果需要，调用函数负责通过调用 fastcall16 函数保存寄存器的值。 当前，仅修改 CUR_PP 页面指针寄存器。

### *I2CHW_fReadBytes*

**说明：**

该函数启动与可寻址的从器件进行的 Read 读数据操作。 从 $I^2C$ 从器件读取一个或多个字节 (bCnt)，并向 pbXferData 所指向的数组写入该数据。 一旦调用了该子程序，内含的 ISR 即处理更多数据。

**C 原型：**

```
void I2CHW_fReadBytes(BYTE bSlaveAddr, BYTE * pbXferData, BYTE bCnt, BYTE bMode);
```

**汇编程序：**

```
mov   A,I2CHW_CompleteXfer   ; Pass complete transfer flag
push  A
mov   A,0x09                 ; Pass the byte count
push  A
mov   A,>sData               ; Load the MSB of the sData pointer
push  A
mov   A,<sData               ; Load the LSB of the sData pointer
push  A
mov   A,0x68                 ; Pass slave address 0x68
push  A
lcall  I2CHW_fReadBytes       ; lcall function to read data from slave
                             ; Reg A contains return value.
add   sp,-5 ; Restore the stack
```

**参数：**

bSlaveAddr：7-bit 从器件地址。pbXferData： 指向 RAM 中数据数组的指针。bCnt ： 要读取的数据的个数。bMode： 操作模式： 如果模式被设置为 I2CHW_CompleteXfer，则执行完整传输。 如果模式被设置为 I2CHW_RepStart、或被设置为 I2CHW_NoStop，则不生成停止位。 这可实现向从器件发送 $I^2C$ 总线的合并传输。 通过重复开始，可发起后续传输。 见本节末表格。

**返回值：**

None。

**副作用：**

A 和 X 寄存器可能因此函数的本次执行或以后执行而被修改。 对于大内存模式下 (CY8C29xxx) 的所有 RAM 页面指针寄存器也是如此。 如果需要，调用函数负责通过调用 fastcall16 函数保存寄存器的值。 当前，仅修改 CUR_PP 页面指针寄存器。

### *I2CHW_bWriteBytes*

**说明：**

该函数启动与可寻址的从器件进行的 Write 写入数据操作。 将 pbXferData 所指向的 RAM 数组的一个或多个字节 (bCnt) 写入 (bSlaveAddr) 编址的从器件中。 一旦调用了该子程序，内部的 ISR 将处理更多数据。

**C 原型：**

```
void I2CHW_bWriteBytes(BYTE bSlaveAddr, BYTE * pbXferData, BYTE bCnt, BYTE bMode);
```

**汇编程序：**

```
mov   A,I2CHW_CompleteXfer   ; Pass complete transfer flag
push  A
mov   A,0x09                 ; Pass the byte count
push  A
mov   A,>sData               ; Load the MSB of the sData pointer
push  A
mov   A,<sData               ; Load the LSB of the sData pointer
push  A
mov   A,0x68                 ; Pass slave address 0x68
push  A
lcall  I2CHW_bWriteBytes      ; lcall function to write data to slave
                             ; Reg A contains return value.
add   sp,-5 ; Restore the stack
```

**参数：**

bSlaveAddr：7-bit 从器件地址。pbXferData： 指向 RAM 中数据数组的指针。bCnt： 要写入的数据的个数。bMode： 操作模式： 如果模式被设置为 I2CHW_CompleteXfer，则执行完整传输。 如果模式被设置为 I2CHW_RepStart、或被设置为 I2CHW_NoStop，则不生成停止位。 这可实现向从器件发送 $I^2C$ 总线的合并传输。 通过重复开始，可启动后续传输。 见本节末表格。

**返回值：**

None

**副作用：**

A 和 X 寄存器可能因此函数的本次执行或以后执行而被修改。 对于大内存模式下 (CY8C29xxx) 的所有 RAM 页面指针寄存器也是如此。 如果需要，调用函数负责通过调用 fastcall16 函数保存寄存器的值。 当前，仅修改 CUR_PP 页面指针寄存器。

### *I2CHW_bWriteCBytes*

**说明：**

该函数启动与可寻址的从器件进行的 Write 写入数据操作。 将 指针 (pbXferData) 指向的常量 flash 数组中的一个或多个字节 (bCnt) 写入 bSlaveAddr 编址的从器件中。 一旦此子函数启动数据传输，包含的 ISR 将进一步处理的数据传输。

**C 原型：**

```
void I2CHW_bWriteCBytes(BYTE bSlaveAddr, const BYTE * pbXferData, BYTE bCnt, BYTE
bMode);
```

**汇编程序：**

```
mov   A,I2CHW_CompleteXfer    ; Pass complete transfer flag
push  A
mov   A,0x09                  ; Pass the byte count
push  A
mov   A,>sData                ; Load the MSB of the sData pointer
push  A
mov   A,<sData                ; Load the LSB of the sData pointer
push  A
mov   A,0x68                  ; Pass slave address 0x68
push  A
lcall  I2CHW_bWriteCBytes     ; lcall function to write data to slave
                              ; A contains return value.
add   sp,-5 ; Restore the stack
```

**参数：**

bSlaveAddr：7-bit 从器件地址。pbXferData： 指向闪存 “const” 数据数组的指针。bCnt： 要写入的数据的个数。bMode： 操作模式： 如果模式被设置为 I2CHW_CompleteXfer，则执行完整传输。 如果模式被设置为 I2CHW_RepStart、或被设置为 I2CHW_NoStop，则不生成停止位。 这可实现向从器件发送 $I^2C$ 总线的合并传输。 通过重复开始，可发起后续传输。 见本节末表格。

**返回值：**

None。

**副作用：**

A 和 X 寄存器可能因此函数的本次执行或以后执行而被修改。 对于大内存模式下 (CY8C29xxx) 的所有 RAM 页面指针寄存器也是如此。 如果需要，调用函数负责通过调用 fastcall16 函数保存寄存器的值。 当前，仅修改 CUR_PP 页面指针寄存器。

**Note** 可以使用 bMode 参数来执行 $I^2C$ 总线合并格式传输。 要执行合并传输，需先执行 I2CHW_bWriteBytes 或 I2CHW_bWriteCBytes 命令，同时将 bMode 参数设置为 I2CHW_NoStop (0x02)。 这可执行写操作而无需停止位。 接下来，执行 I2CHW_fReadBytes 命令，并将参数设置为 I2CHW_RepStart (0x01)。

| 常量 | 值 | 说明 |
|---|---|---|
| I2CHW_CompleteXfer | 0x00 | 从开始至结束执行完整的传输。 |
| I2CHW_RepStart | 0x01 | 发送重复启动 (Repeat Start) 而不是单个启动 (Start)。 |
| I2CHW_NoStop | 0x02 | 执行没有停止位的传输。 |

### *I2CHW_bReadSlaveStatus*

**说明：**

返回控制 / 状态寄存器的状态位。

**C 原型：**

```
BYTE  I2CHW_bReadSlaveStatus(void);
```

**汇编程序：**

```
lcall  I2CHW_bReadSlaveStatus ; Accumulator contains status
```

**参数：**

None

**返回值：**

```
BYTE  I2CHW_SlaveStatus
```

参见表格。

**副作用：**

A 和 X 寄存器可能因此函数的本次执行或以后执行而被修改。 对于大内存模式下 (CY8C29xxx) 的所有 RAM 页面指针寄存器也是如此。 如果需要，调用函数负责通过调用 fastcall16 函数保存寄存器的值。

| 常量 | 值 | 说明 |
|---|---|---|
| I2CHW_RD_NOERR | 01h | 主器件读取数据，正常 ISR 退出 |
| I2CHW_RD_ OVERFLOW | 02h | 主器件所读取的数据字节超出了可用字节数 |
| I2CHW_RD_ COMPLETE | 04h | 读取操作启动并完成 |
| I2CHW_READFLASH | 08h | 下一个读取操作来自闪存位置 |
| I2CHW_WR_NOERR | 10h | 主器件成功写入了数据 |
| I2CHW_WR_OVERFLOW | 20h | 主器件向写入缓冲区写入了过多字节 |
| I2CHW_WR_COMPLETE | 40h | 主器件写入操作因新地址或停止位而结束 |
| I2CHW_ISR_ACTIVE | 80h | $I^2C$ ISR 尚未退出并处于活动状态 |

## *I2CHW_ ClrSlaveRdStatus*

**说明：**

清除 I2CHW_SlaveStatus 寄存器中的读取状态位。 不影响其他位。

**C 原型：**

```
void  I2CHW_ClrSlaveRdStatus (void);
```

**汇编程序：**

```
lcall  I2CHW_ClrSlaveRdStatus
```

**参数：**

None

**返回值：**

None

**副作用：**

A 和 X 寄存器可能会因此函数的本次执行或以后执行而被修改。 对于大内存模式下 (CY8C29xxx) 的所有 RAM 页面指针寄存器也是如此。 如果需要，调用函数负责通过调用 fastcall16 函数保存寄存器的值。

### *I2CHW_ ClrSlaveWrStatus*

**说明：**

清除 I2CHW_SlaveStatus 寄存器中的写入状态位。 不影响其他位。

**C 原型：**

```
void  I2CHW_ClrSlaveWrStatus (void);
```

**汇编程序：**

```
lcall  I2CHW_ClrSlaveWStatus
```

**参数：**

None

**返回值：**

None

**副作用：**

A 和 X 寄存器可能会因此函数的本次执行或以后执行而被修改。 对于大内存模式下 (CY8C29xxx) 的所有 RAM 页面指针寄存器也是如此。 如果需要，调用函数负责通过调用 fastcall16 函数保存寄存器的值。

### *I2CHW_bReadMasterStatus*

**说明：**

返回控制 / 状态寄存器的状态位。

**C 原型：**

```
BYTE  I2CHW_bReadMasterStatus(void);
```

**汇编程序：**

```
lcall  I2CHW_bReadMasterStatus ; Accumulator contains status
```

**参数：**

None

**返回值：**

```
BYTE  I2CHW_MasterStatus
```

参见表格。

**副作用：**

A 和 X 寄存器可能会因此函数的本次执行或以后执行而被修改。 对于大内存模式下 (CY8C29xxx) 的所有 RAM 页面指针寄存器也是如此。 如果需要，调用函数负责通过调用 fastcall16 函数保存寄存器的值。

| 常量 | 值 | 说明 |
|---|---|---|
| I2CHW_RD_NOERR | 01h | 主器件读取数据，正常 ISR 退出 |
| I2CHW_RD_ OVERFLOW | 02h | 主器件所读取的数据字节超出了可用字节数 |
| I2CHW_RD_ COMPLETE | 04h | 读取操作已启动并完成 |
| I2CHW_READFLASH | 08h | 下一个读取操作来自闪存位置 |
| I2CHW_WR_NOERR | 10h | 主器件成功写入了数据 |
| I2CHW_WR_OVERFLOW | 20h | 主器件向写入缓冲区写入了过多字节 |
| I2CHW_WR_COMPLETE | 40h | 主器件写入操作因新地址或停止位而结束 |
| I2CHW_ISR_ACTIVE | 80h | $I^2C$ ISR 尚未退出并处于活动状态 |

### *I2CHW_ ClrMasterRdStatus*

**说明：**

清除 I2CHW_MasterStatus 寄存器中的读取状态位。 不影响其他位。

**C 原型：**

```
void  I2CHW_ClrMasterRdStatus (void);
```

**汇编程序：**

```
lcall  I2CHW_ClrMasterRdStatus
```

**参数：**

None

**返回值：**

None

**副作用：**

A 和 X 寄存器可能会因此函数的本次执行或以后执行而被修改。 对于大内存模式下 (CY8C29xxx) 的所有 RAM 页面指针寄存器也是如此。 如果需要，调用函数负责通过调用 fastcall16 函数保存寄存器的值。

### *I2CHW_ ClrMasterWrStatus*

**说明：**

清除 I2CHW_MasterStatus 寄存器中的写入状态位。 不影响其他位。

**C 原型：**

```
void  I2CHW_ClrMasterWrStatus (void);
```

**汇编程序：**

```
lcall  I2CHW_ClrMasterWStatus
```

**参数：**

None

**返回值：**

None

**副作用：**

A 和 X 寄存器可能会因此函数的本次执行或以后执行而被修改。 对于大内存模式下 (CY8C29xxx) 的所有 RAM 页面指针寄存器也是如此。 如果需要，调用函数负责通过调用 fastcall16 函数保存寄存器的值。

### *I2CHW_bReadBusStatus*

**说明：**

读 I2CHW_bStatus 字节的当前值。 这是 I2CHW 用户模块使用的 API 函数及 ISR 的内部状态字节。 没有给出任何工具来更改 theI2CHW_bStatus 字节中的数据。 I2CHW 用户模块的所有三个版本都可使用该字节，但仅给 MultiMasterSlave 版本提供信息，这是因为在 MultiMasterSlave 环境下操作给出了额外的状态信息。

**C 原型：**

```
BYTE I2CHW_bReadBusStatus (void);
```

**汇编程序：**

```
lcall  I2CHW_bReadBusStatus
```

**参数：**

None

**返回值：**

```
BYTE I2CHW_bStatus
```

详细说明见下表。 这些定义报告的是状态变量，而非状态 / 控制位。

| 常量 | 值 | 说明 |
|---|---|---|
| RepStart, NoStop | 01h, 02h | 预留用于传输选项 CompleteXfer/RepStart/NoStop |
| BUS_BUSY | 04h | 总线忙 |
| LOST_ARB | 08h | 主器件丢失仲裁 |
| BUS_ERROR | 10h | 发生了总线错误 |
| SLAVE_NAK | 20h | 从器件未响应 |
| ERROR | 40h | 请求的操作失败 |
| ISR_ACTIVE | 80h | ISR 处于活动状态，是 I2C 操作 |

**副作用：**

A 和 X 寄存器可能会因此函数的本次执行或以后执行而被修改。 对于大内存模式下 (CY8C29xxx) 的所有 RAM 页面指针寄存器也是如此。 如果需要，调用函数负责通过调用 fastcall16 函数保存寄存器的值。

### *I2CHW_fReadBytesNoStall*

**说明：**

该函数启动与可寻址从器件进行的 Read 读数据操作。 从从 I2C 器件读取一个或多个字节 (bCnt)，并将该数据写入 pbXferData 所指向的数组中。 调用该子程序后，只要总线不在服务于另一主器件，包含的 ISR 即处理进一步的数据。

**C 原型：**

```
BYTE I2CHW_fReadBytesNoStall(BYTE bSlaveAddr, BYTE * pbXferData, BYTE bCnt, BYTE
bMode);
```

**汇编程序：**

```
mov A,I2CHW_CompleteXfer ; Pass complete transfer flag
push A
mov A,0x09 ; Pass the byte count
push A
mov A,>sData ; Load the MSB of the sData pointer
push A
mov A,<sData ; Load the LSB of the sData pointer
push A
mov A,0x68 ; Pass slave address 0x68
push A
lcall I2CHW_fReadBytesNoStall ; lcall function to read data from slave
; Reg A contains return value.
add sp,-5 ; Restore the stack
```

**参数：**

bSlaveAddr：7-bit 从地址。

pbXferData： 指向 RAM 中数据数组的指针：

bCnt： 要读入的数据的个数。

bMode： 操作模式： 如果模式被设置为 I2CHW_CompleteXfer，则执行完整传输。 如果模式被设置为 I2CHW_RepStart、或被设置为 I2CHW_NoStop，则不生成停止位。 这可实现向从器件发送 II2C2C 总线的合并传输。 通过重复开始，可发起后续传输。 见本节末表格。

**返回值：**

如果总线不忙，返回字节 BYTE I2CHW_MasterStatus 或如果总线忙，返回 0xFF

**副作用：**

A 和 X 寄存器可能会因此函数的本次执行或以后执行而被修改。 对于大内存模式下 (CY8C29xxx) 的所有 RAM 页面指针寄存器也是如此。 如果需要，调用函数负责通过调用 fastcall16 函数保存寄存器的值。 当前，仅修改 CUR_PP 页面指针寄存器。

### *I2CHW_bWriteBytesNoStall*

**说明：**

该函数启动向可寻址的从器件进行的 Write 写入数据操作。 将 pbXferData 所指向的 RAM 数组的一个或多个字节 (bCnt) 写入向 (bSlaveAddr) 编址的从器件中。 调用该子程序后，只要总线不在服务于另一主器件，包含的 ISR 即处理进一步的数据。

**C 原型：**

```
BYTE I2CHW_bWriteBytesNoStall(BYTE bSlaveAddr, BYTE * pbXferData, BYTE bCnt, BYTE
bMode);
```

**汇编程序：**

```
mov A,I2CHW_CompleteXfer ; Pass complete transfer flag
push A
mov A,0x09 ; Pass the byte count
push A
mov A,>sData ; Load the MSB of the sData pointer
push A
mov A,<sData ; Load the LSB of the sData pointer
push A
mov A,0x68 ; Pass slave address 0x68
push A
lcall I2CHW_bWriteBytesNoStall ; lcall function to write data to slave
; Reg A contains return value.
add sp,-5 ; Restore the stack
```

**参数：**

bSlaveAddr：7-bit 从地址。

pbXferData： 指向 RAM 中数据数组的指针：

bCnt： 要写的数据的计数。

bMode： 操作模式： 如果模式被设置为 I2CHW_CompleteXfer，则执行完整传输。 如果模式被设置为 I2CHW_RepStart、或被设置为 I2CHW_NoStop，则不生成停止位。 这可实现向从器件发送 II2C2C 总线的合并传输。 通过重复开始，可发起后续传输。 见本节末表格。

**返回值：**

如果总线不忙，返回字节 BYTE I2CHW_MasterStatus 或如果总线忙，返回 0xFF

**副作用：**

A 和 X 寄存器可能会因此函数的本次执行或以后执行而被修改。 对于大内存模式下 (CY8C29xxx) 的所有 RAM 页面指针寄存器也是如此。 如果需要，调用函数负责通过调用 fastcall16 函数保存寄存器的值。 当前，仅修改 CUR_PP 页面指针寄存器。

### *I2CHW_bWriteCBytesNoStall*

**说明：**

该函数启动与可寻址的从器件进行的 Write 写入数据操作。 将指针 (pbXferData) 指向的常量 flash 数组的 写一个或多个字节 (bCnt) 写入 bSlaveAddr 编址的从器件中。 一旦此子函数发起数据传输，只要总线不在服务于另一主器件，包含的 ISR 即处理进一步的数据传输。

**C 原型：**

```
BYTE I2CHW_bWriteCBytesNoStall(BYTE bSlaveAddr, const BYTE * pbXferData, BYTE bCnt,
BYTE bMode);
```

**汇编程序：**

```
mov A,I2CHW_CompleteXfer ; Pass complete transfer flag
push A
mov A,0x09 ; Pass the byte count
push A
mov A,>sData ; Load the MSB of the sData pointer
push A
mov A,<sData ; Load the LSB of the sData pointer
mov A,0x68 ; Pass slave address 0x68
push A
```

```
lcall I2CHW_ bWriteCBytesNoStall ; lcall function to write data to slave
; A contains return value.
add sp,-5 ; Restore the stack
```

**参数：**

bSlaveAddr：7-bit 从地址。

pbXferData： 指向闪存 “const” 数据数组的指针。

bCnt： 要写的数据的个数。

bMode： 操作模式： 如果模式被设置为 I2CHW_CompleteXfer，则执行完整传输。 如果模式被设置为 I2CHW_RepStart、或被设置为 I2CHW_NoStop，则不生成停止位。 这可实现向从器件发送 II2C2C 总线的合并传输。 通过重复开始，可发起后续传输。 见本节末表格。

**返回值：**

如果总线不忙，返回字节 BYTE I2CHW_MasterStatus 或如果总线忙，返回 0xFF

**副作用：**

A 和 X 寄存器可能会因此函数的本次执行或以后执行而被修改。 对于大内存模式下 (CY8C29xxx) 的所有 RAM 页面指针寄存器也是如此。 如果需要，调用函数负责通过调用 fastcall16 函数保存寄存器的值。 当前，仅修改 CUR_PP 页面指针寄存器。

**注意** 可以使用 bMode 参数来执行 I2C 总线合并格式传输。 要执行合并传输，需先执行 I2CHW_bWriteBytes 或 I2CHW_bWriteCBytes 命令，同时将 bMode 参数设置为 I2CHW_NoStop (0x02)。 这可执行写操作而无需停止位。 接下来，如果总线上有其他主器件，则执行 I2CHW_fReadBytes、或 I2CHW_fReadBytesNoStall 命令，bMode 参数设置为 I2CHW_RepStart (0x01)。

| 常量 | 值 | 说明 |
|---|---|---|
| I2CHW_RepStart | 01h | 内部状态位，用于控制主器件的停止 / 重复启动 |
| I2CHW_NoStop | 02h | 内部状态，用于控制主器件的停止 / 重复启动 |
| I2CHW_BUS_BUSY | 04h | 指示 I2C 总线正在由该设备使用的标志 |
| I2CHW_LOST_ARB | 08h | 状态标志，用于指示主器件丢失了对另一主器件的仲裁，且 “ 未 ” 获得对 I2C 总线的控制。 |
| I2CHW_BUS_ERROR | 10h | I2C 总线上发现了非法条件。 |
| I2CHW_SLAVE_NAK | 20h | 由该主器件所编址的外部从器件未响应其地址 |
| I2CHW_ERROR | 40h | 该主器件尝试了一个操作，但未成功 |
| I2CHW_ISR_ACTIVE | 80h | 正在对该设备进行某种主器件或从器件活动。 |

## 固件源代码示例

以下是一个被配置为从器件的 I2CHW 用户模块的实现：

```
/*****************************************************************************/
/* Sample assembly code to commuinication with:                              */
/*  C Master sample code.                                                    */
/*  ASM Master sample code.                                                  */
/*                                                                           */
/* This code writes and reads back echoed data from the slave.               */
```

```
/*                                                                    */
/* NOTE: 1. I2CHW does not depend on the CPU clock.                   */
/*       2. The instance name of the I2CHWs User Module is assumed to */
/*          be I2CHW.                                                 */
/*       3. Device is assumed to be Large Memory Model.               */
/**********************************************************************/
#include <m8c.h>        // part specific constants and macros
#include "PSoCAPI.h"    // PSoC API definitions for all User Modules

/* Define buffer size */
#define BUFFERSIZE 8
/* Setup a 8 byte buffer */
BYTE txRxBuffer[BUFFERSIZE];
BYTE status;

void I2C_poll(void)
{
    status = I2CHW_bReadI2CStatus();
        /* Wait to read data from the master */
        if( status & I2CHW_WR_COMPLETE )
        {
            /* Data received - clear the Write status */
            I2CHW_ClrWrStatus();
            /* Reset the pointer for the next read data */
            I2CHW_InitWrite(txRxBuffer,BUFFERSIZE);
        }
        /* wait until data is echoed */
        if( status & I2CHW_RD_COMPLETE )
        {
            /* Data echoed - clear the read status */
            I2CHW_ClrRdStatus();
            /* Reset the pointer for the next data to echo */
            I2CHW_InitRamRead(txRxBuffer,BUFFERSIZE);
        }
}
void main(void)
{
    /* Start the slave and wait for the master */
    I2CHW_Start();
    I2CHW_EnableSlave();
    /* Enable the global and local interrupts */
    M8C_EnableGInt;
    I2CHW_EnableInt();
    /* Setup the Read and Write Buffer - set to the same buffer */
    I2CHW_InitRamRead(txRxBuffer,BUFFERSIZE);
    I2CHW_InitWrite(txRxBuffer,BUFFERSIZE);
    /* Echo forever */
    while( 1 )
    {
        I2C_poll();

        // Place user code here to update and read structure data.
        // Please note that the I2C_poll() should be called often enough to properly
        //serve I2C transactions.
```

```
        // Thus if user code is large call the I2C_poll() multiple times during main
        //loop execution.
    }
} //end of main
```

被配置为主器件的 I2CHW 用户模块的实现：

```
/*****************************************************************************/
/* Sample assembly code to commuinication with:                              */
/*  C Slave sample code.                                                     */
/*  ASM Slave sample code.                                                   */
/*                                                                           */
/* This sample code will transmit data to a slave and then will read         */
/* back from the slave.                                                      */
/*                                                                           */
/* NOTE: 1. I2CHW does not depend on the CPU clock.                          */
/*       2. The instance name of the I2CHWm User Module is assumed to        */
/*          be I2CHW.                                                        */
/*       3. Device is assumed to be Large Memory Model.                      */
/*****************************************************************************/
#include <m8c.h>        // part specific constants and macros
#include "PSoCAPI.h"    // PSoC API definitions for all User Modules

/* Define slave address */
#define SLAVE_ADDRESS 0x55
/* Define buffer size */
#define BUFFER_SIZE 0x08

/* Setup buffers */
BYTE txBuffer[BUFFER_SIZE];
BYTE rxBuffer[BUFFER_SIZE];
BYTE status;

void main(void)
{
    /* Start the master */
    I2CHW_Start();
    I2CHW_EnableMstr();
    /* Enable the global and local interrupts */
    M8C_EnableGInt;
    I2CHW_EnableInt();

/* Send and Receive forever*/
while( 1 )
    {
        /* Send the contents of the data in txBuffer */;
        I2CHW_bWriteBytes(SLAVE_ADDRESS, txBuffer, BUFFER_SIZE, I2CHW_CompleteXfer);
        /* Wait until the data is transferred */
        while(!(I2CHW_bReadI2CStatus() & I2CHW_WR_COMPLETE));
        /* Clear Write Complete Status bit */
        I2CHW_ClrWrStatus();

/* Read from the slave and place in rxBuffer */;
        I2CHW_fReadBytes(SLAVE_ADDRESS, rxBuffer, BUFFER_SIZE, I2CHW_CompleteXfer);
        /* Wait until the data is read */
```

```
        while(!(I2CHW_bReadI2CStatus() & I2CHW_RD_COMPLETE));
        /* Clear Read Complete Status bit */
        I2CHW_ClrRdStatus();

/* Increment 1st byte in txBuffer so the data changes each loop */
        txBuffer[0]++;
    }
} //end of main
```

被配置为 MultiMasterSlave 的 I2CHW 用户模块的实现：

```
/*****************************************************************************/
/* This sample code will receive data from Muster and                        */
/* transmit data to a Slave.                                                 */
/*                                                                           */
/* The instance name of the I2CHWs User Module is assumed                    */
/* to be I2CHW.                                                              */
/*                                                                           */
/* NOTE: 1. I2CHW does not depend on the CPU clock.                          */
/*       2. The instance name of the I2CHWm User Module is assumed to        */
/*          be I2CHW.                                                        */
/*       3. Device is assumed to be Large Memory Model.                      */
/*****************************************************************************/
#include <m8c.h>        // part specific constants and macros
#include "PSoCAPI.h"    // PSoC API definitions for all User Modules

/* Define Slave address */
#define SLAVEADDRESS 0x7

/* Define buffer size */
#define BUFFERSIZE 0x10

void main(void)
{
    /* Setup buffer */
    BYTE TxRxBuffer[BUFFERSIZE];

/* Start the slave and master */
    I2CHW_Start();
    I2CHW_EnableSlave();
    I2CHW_EnableMstr();

/* Setup the read and write buffers */
    I2CHW_InitSlaveRamRead(TxRxBuffer,BUFFERSIZE);
    I2CHW_InitSlaveWrite(TxRxBuffer,BUFFERSIZE);

/* Enable the local and global interrupts */
    I2CHW_EnableInt();
    M8C_EnableGInt;

/* Loop forever */
    while(1)
    {
        /* Look for data write from master */
        if (I2CHW_bReadSlaveStatus() & I2CHW_WR_COMPLETE)
```

```
        {
            /* Data received - clear the Write status */
            I2CHW_ClrSlaveWrStatus();

            /* Reset the pointer for the next write data */
            I2CHW_InitSlaveWrite(TxRxBuffer,BUFFERSIZE);

            /* Master sends the content of the data in Buffer to the Slave */
          I2CHW_bWriteBytes(SLAVEADDRESS, TxRxBuffer, BUFFERSIZE, I2CHW_CompleteXfer);

            /* Wait until the data is transferred */
            while(!(I2CHW_bReadMasterStatus() & I2CHW_WR_COMPLETE));

            /* Clear Write Complete Status bit */
            I2CHW_ClrMasterWrStatus();
        }

        /* If any Master has read data, it is necessary to reset the read pointer
        and clear the Read Complete Status bit for the correct next read of data */
        if (I2CHW_bReadSlaveStatus() & I2CHW_RD_COMPLETE)
        {
            /* Clear Read Complete Status bit */
            I2CHW_ClrSlaveRdStatus();

  /* Reset the Read pointer */
            I2CHW_InitSlaveRamRead(TxRxBuffer,BUFFERSIZE);
        }

/* uncomment this code for Errors handling */
        // if (I2CHW_SlaveStatus & (I2CHW_WR_OVERFLOW | I2CHW_RD_OVERFLOW))
        //{
        //    I2CHW_ClrSlaveWrStatus();
         //   I2CHW_ClrSlaveRdStatus();
        //    I2CHW_InitSlaveRamRead(buf_wr,2);
        //    I2CHW_InitSlaveWrite(buf_wr,2);
        //}
    }
}
```

被配置为从器件的 I2CHW 用户模块的汇编代码编写实现：

```
;*****************************************************************
;* Sample assembly code to commuinication with:                  *
;*  C Master sample code.                                        *
;*  ASM Master sample code.                                      *
;*                                                               *
;* This code writes and reads back echoed data from the slave.   *
;*                                                               *
;* NOTE: 1. I2CHW does not depend on the CPU clock.              *
;*       2. The instance name of the I2CHWm User Module is assumed to  *
;*          be I2CHW.                                            *
;*       3. Device is assumed to be Large Memory Model.          *
;*****************************************************************
include "m8c.inc"       ; part specific constants and macros
include "memory.inc"    ; Constants & macros for SMM/LMM and Compiler
```

```
include "PSoCAPI.inc"   ; PSoC API definitions for all User Modules

BUFFERSIZE: equ 8

export txRxBuffer

area bss(RAM)

txRxBuffer: blk BUFFERSIZE

area text(ROM, REL)

export _main

_main:
    ; Initialize I2CHW
    lcall I2CHW_Start
    lcall I2CHW_EnableSlave
    ; Enable the global and local interrupts
    M8C_EnableGInt
    ; Enable the I2CHW as an ISR based process
    lcall I2CHW_EnableInt

; Set the Read Buffer
    mov A, BUFFERSIZE ; Pass the byte count
   push A
    mov A,>txRxBuffer ; Load the MSB of the rxBuffer pointer
    push A
    mov A,<txRxBuffer ; Load the LSB of the rxBuffer pointer
    push A
    lcall I2CHW_InitRamRead
    add SP, -3 ; Restore the stack

; Set the Write Buffer
    mov A, BUFFERSIZE ; Pass the byte count
    push A
    mov A,>txRxBuffer ; Load the MSB of the rxBuffer pointer
    push A
    mov A,<txRxBuffer ; Load the LSB of the rxBuffer pointer
    push A
    lcall I2CHW_InitWrite
    add SP,-3 ; Restore the stack

    ; Echo forever
    ; Wait until some data is transferred
    CheckI2CStatus:
        lcall I2CHW_bReadI2CStatus ; Accumulator contains status
        push A ; Preserve a copy of A for RD comparison

; Look for data read from master
        and A,I2CHW_RD_COMPLETE
        jnz ReadHappened
        ; Look for data write from master
pop A ; Retrieve preserved copy of A
        and A,I2CHW_WR_COMPLETE
```

```
        jnz WriteHappened

    jmp CheckI2CStatus

    ReadHappened:
        ; Clears the read status bits in the I2CHW_RsrcStatus register
        lcall I2CHW_ClrRdStatus
        ;Reset the pointer for the next data to echo
        mov A,BUFFERSIZE ; Pass the byte count
        push A
        mov A,>txRxBuffer ; Load the MSB of the rxBuffer pointer
        push A
        mov A,<txRxBuffer ; Load the LSB of the rxBuffer pointer
        push A
        lcall I2CHW_InitRamRead
        add SP,-4 ; Restore the stack
jmp CheckI2CStatus

    WriteHappened:
        ; Clears the write status bits in the I2CHW_RsrcStatus register
        lcall I2CHW_ClrWrStatus
        ;Reset the pointer for the next data write
        mov A,BUFFERSIZE ; Pass the byte count
        push A
        mov A,>txRxBuffer ; Load the MSB of the rxBuffer pointer
        push A
        mov A,<txRxBuffer ; Load the LSB of the rxBuffer pointer
        push A
        lcall I2CHW_InitWrite
        add SP,-3 ; Restore the stack
        jmp CheckI2CStatus
;end _main
```

被配置为主器件的 I2CHW 用户模块的汇编代码编写实现：

```
;*****************************************************************************
;* Sample assembly code to commuinication with:                              *
;*  C Slave sample code.                                                     *
;*  ASM Slave sample code.                                                   *
;*                                                                           *
;* This sample code will transmit data to a slave and then will read         *
;* back from the slave.                                                      *
;*                                                                           *
;* NOTE: 1. I2CHW does not depend on the CPU clock.                          *
;*       2. The instance name of the I2CHWm User Module is assumed to        *
;*          be I2CHW.                                                        *
;*       3. Device is assumed to be Large Memory Model.                      *
;*****************************************************************************
include "m8c.inc"       ; part specific constants and macros
include "memory.inc"    ; Constants & macros for SMM/LMM and Compiler
include "PSoCAPI.inc"   ; PSoC API definitions for all User Modules

BUFFERSIZE: equ 0x08 ; Define buffer size
SLAVEADDRESS: equ 0x55 ; Define slave address
```

```
export txBuffer
export rxBuffer

area bss(RAM)

txBuffer: blk BUFFERSIZE
rxBuffer: blk BUFFERSIZE

area text(ROM, REL)

export _main

_main:
    ; Initialize I2CHW
    lcall I2CHW_Start
    lcall I2CHW_EnableMstr
    ; Enable the global and local interrupts
    M8C_EnableGInt
    ; Enable the I2CHW as an ISR based process
    lcall I2CHW_EnableInt

    ; Send the contents of the data in txBuffer
    WriteTXBuffer:
        mov A,I2CHW_CompleteXfer ; Pass complete transfer flag
        push A
        mov A,BUFFERSIZE ; Pass the byte count
        push A
        mov A,>txBuffer ; Load the MSB of the txBuffer pointer
        push A
        mov A,<txBuffer ; Load the LSB of the txBuffer pointer
        push A
        mov A,SLAVEADDRESS ; Pass slave address SLAVEADDRESS
        push A
        ; Call function to write data to slave
        lcall _I2CHW_bWriteBytes
        ; Reg A contains return value.
        add SP,-5 ; Restore the stack
        jmp CheckI2CStatus

    ; Read from the slave and place in rxBuffer
    ReadRXBuffer:
        mov A,I2CHW_CompleteXfer ; Pass complete transfer flag
        push A
        mov A,BUFFERSIZE ; Pass the byte count
        push A
        mov A,>rxBuffer ; Load the MSB of the rxBuffer pointer
        push A
        mov A,<rxBuffer ; Load the LSB of the rxBuffer pointer
        push A
        mov A,SLAVEADDRESS ; Pass slave address SLAVEADDRESS
        push A
        ; Call function to read data from slave
        lcall _I2CHW_fReadBytes
        ; Reg A contains return value.
```

```
        add SP,-5 ; Restore the stack
        jmp CheckI2CStatus ; Poll I2C status

    ; Wait until the data is transferred
    CheckI2CStatus:
        lcall I2CHW_bReadI2CStatus ; Accumulator contains status
        push A ; Preserve a copy of A for RD comparison
        and A, I2CHW_WR_COMPLETE
       jnz WriteHappened
        pop A ; Retrieve preserved copy of A
        and A, I2CHW_RD_COMPLETE
        jnz ReadHappened
        jmp CheckI2CStatus

    ReadHappened:
        ; Clears the read status bits in the I2CHW_RsrcStatus register
        lcall I2CHW_ClrRdStatus
        ; Do something after read
       ; Increment 1st byte in txBuffer so the data changes each loop
        inc [txBuffer+0]
        jmp WriteTXBuffer

    WriteHappened:
        ; Clears the write status bits in the I2CHW_RsrcStatus register
        lcall I2CHW_ClrWrStatus
        ; Do something with the data
        jmp ReadRXBuffer

;end _main
```

被配置为 MultiMasterSlave 的 I2CHW 用户模块的汇编代码编写实现：

```
;*****************************************************************************
;* This sample code will receive data from Muster and                        *
;*  transmit data to a slave.                                                *
;*                                                                           *
;* NOTE: 1. I2CHW does not depend on the CPU clock.                          *
;*       2. The instance name of the I2CHWm User Module is assumed to        *
;*          be I2CHW.                                                        *
;*       3. Device is assumed to be Large Memory Model.                      *
;*****************************************************************************
include "m8c.inc"       ; part specific constants and macros
include "memory.inc"    ; Constants & macros for SMM/LMM and Compiler
include "PSoCAPI.inc"   ; PSoC API definitions for all User Modules

; Define slave address
SLAVEADDRESS:  equ 0x7

; Define buffer size
BUFFERSIZE:    equ 0x10

area bss(RAM)

; Setup buffer
TxRxBuffer:     blk BUFFERSIZE
```

```
area text(ROM, REL)

export _main

_main:
    ; Start the slave and master
    lcall I2CHW_Start
    lcall I2CHW_EnableSlave
    lcall I2CHW_EnableMstr

    ; Setup the read and write buffers of the Slave
    ; Set the read buffer
    mov A, BUFFERSIZE ; Pass the byte count
    push A
    mov A, >TxRxBuffer ; Load the MSB of the TxRxBuffer pointer
    push A
    mov A, <TxRxBuffer ; Load the LSB of the TxRxBuffer pointer
    push A
    lcall I2CHW_InitSlaveRamRead
    add SP, -3 ; Restore the stack

    ; Set the write buffer
    mov A, BUFFERSIZE ; Pass the byte count
    push A
    mov A,>TxRxBuffer ; Load the MSB of the TxRxBuffer pointer
    push A
    mov A,<TxRxBuffer ; Load the LSB of the TxRxBuffer pointer
    push A
    lcall I2CHW_InitSlaveWrite
    add SP,-3 ; Restore the stack

    ; Enable the local and global interrupts
    lcall I2CHW_EnableInt
    M8C_EnableGInt

; loop forever
loop:

    ; Look for data write from master
    lcall I2CHW_bReadSlaveStatus ; Accumulator contains status
    push A ; Preserve a copy of A for RD comparison
    and A,  I2CHW_WR_COMPLETE
    jz ReadHappened

; Data received - clear the write status
lcall I2CHW_ClrSlaveWrStatus

; Reset the write pointer for the next write data
mov A, BUFFERSIZE ; Pass the byte count
    push A
    mov A,>TxRxBuffer ; Load the MSB of the TxRxBuffer pointer
    push A
    mov A, <TxRxBuffer ; Load the LSB of the TxRxBuffer pointer
    push A
    lcall I2CHW_InitSlaveWrite
```

```
    add SP,-3 ; Restore the stack

; Master sends the content of the data in Buffer to the Slave
mov A, I2CHW_CompleteXfer ; Pass complete transfer flag
    push A
    mov A,BUFFERSIZE ; Pass the byte count
    push A
    mov A,>TxRxBuffer ; Load the MSB of the TxRxBuffer pointer
    push A
    mov A, <TxRxBuffer ; Load the LSB of the TxRxBuffer pointer
    push A
    mov A,SLAVEADDRESS ; Pass slave address SLAVEADDRESS
    push A
    lcall _I2CHW_bWriteBytes; Reg A contains return value
    add SP,-5 ; Restore the stack

; Wait until the data is transferred
CheckI2CStatus:
    lcall I2CHW_bReadMasterStatus ; Accumulator contains status
    and A, I2CHW_WR_COMPLETE
    jz CheckI2CStatus

    ; Clear Write Complete Status bit
lcall I2CHW_ClrMasterWrStatus

; If any Master has read data, it is necessary to reset the read pointer
; and clear the Read Complete Status bit for the correct next read of data
ReadHappened:
    pop A ; Retrieve preserved copy of A
    and A,I2CHW_RD_COMPLETE
    jz ErrorHappened

; Clear Read Complete Status bit
lcall I2CHW_ClrSlaveRdStatus

; Reset the Read pointer
    mov A, BUFFERSIZE ; Pass the byte count
    push A
    mov A, >TxRxBuffer ; Load the MSB of the TxRxBuffer pointer
    push A
    mov A, <TxRxBuffer ; Load the LSB of the TxRxBuffer pointer
    push A
    lcall I2CHW_InitSlaveRamRead
    add SP, -3 ; Restore the stack

ErrorHappened:
    ;insert here code for Errors handling

jmp loop
```

## 配置寄存器

本节介绍由 $I^2C$ HW 用户模块使用或者修改的 PSoC 资源寄存器。

Table 1. 资源 I2C_CFG: Bank 0 reg[D6] 配置寄存器

| 位 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| 值 | 保留 | PinSelect | Bus Error IE | Stop IE | Clock Rate[1] | Clock Rate[0] | Enable Master | Enable Slave |

Pin Select: 选择 P1[5]/P1[7] 或 P1[0]/P1[1] 作为 SCL 和 SDA。

Bus Error Interrupt Enable: 在总线发生错误时允许 $I^2C$ 中断的产生。

Stop Error Interrupt Enable: 在 $I^2C$ 停止条件下允许 $I^2C$ 中断。

Clock Rate[1,0]: 从三个有效时钟频率 50- 100- 400 kbps 中选择。

Enable Master: 启用 $I^2C$ HW 模块作为总线主器件。

Enable Slave: 启用 $I^2C$ HW 模块作为总线 Slave。

Table 2. 资源 I2C_SCR: Bank 0 reg[D7] 状态控制寄存器

| 位 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| 值 | Bus Error | Lost Arb | Stop Status | ACK out | 寻址 | 发送 | Last Recd Bit (LRB) | Byte Complete |

Bus Error: 表示检测到总线错误条件。

Lost Arbitration: 在多主控模式下，表示此器件仲裁失败，（器件没有能够控制总线）。

Stop Status: 检测到 $I^2C$ 停止条件。

ACK out: 指示 $I^2C$ 模块对所收到的字节进行应答 (1) 或非应答 (0)。

Address: 所接收或所发送的字节是一个地址。

发送: 发送位由固件进行设置，以确定字节传输的方向。 检测到或写入启动 (Start) 或重启 (Restart) 位的任何启动，当以主器件模式运行时将清空此位。 0-Receive 模式，1-Transmit 模式。

Last Received Bit (LRB): 在发送序列中最后收到的一个位 （第 9 位）的值，表示目标器件的应答 / 非应答的状态。

Byte Complete: 接收到了 8 个数据位。 对于接收模式，总线在等待应答 / 非应答时停顿。 在发送模式下，已经接收了应答 / 非应答 （参见 LRB），总线会停顿以等待下一项操作的执行。

Table 3. 资源 I2C_DR: Bank 0 reg[D8] 数据寄存器

| 位 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| 值 | 数据 | | | | | | | |

所接收或所发送的数据。 若要发送数据，该寄存器必须在写入到 I2C_SCR 寄存器之前已经加载。 所接收的数据从此寄存器中读取。 寄存器中可能包含地址或数据。

Table 4. 资源 I2C_MSCR: Bank 0 reg[D9] 主器件状态控制寄存器

| 位 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| 值 | 保留 | 保留 | 保留 | 保留 | Bus Busy | Master Mode | Restart Gen | Start Gen |

Bus Busy: 仅用于主器件，在检测到总线 “ 启动 ” （Start）条件时置位，在检测到 “ 停止 ”(Stop) 条件时清除。

Master Mode: 表示此器件当前作为总线主器件运行。

Restart Gen: 仅用于主器件，可以进行设置，生成 $I^2C$ 总线的重复启动。

Start Gen: 仅用于主器件，在总线闲置时，生成 $I^2C$ 总线启动，并使用数据寄存器 (I2C_DR) 内的数据发送 $I^2C$ 地址

## 版本历史记录

| 版本 | 创作者 | 说明 |
|---|---|---|
| 1.6 | DHA | 添加了版本历史。 |
| 1.7 | DHA | 以下是对 “ 启动 ”(Start) 函数所做的更改:<br>1. 已将用户模块引脚的初始的漏极开路低电平驱动模式更改为 HI-Z 模拟驱动模式。<br>2. 已启用 $I^2C$ 模块。<br>3. 给定延时 5 秒的 NOP 指令。<br>4. 已恢复初始 $I^2C$ 引脚驱动模式。 |
| 1.80 | DHA | 为主器件配置添加了 I2CHW_bReadBusStatus() 函数。<br><br>将头文件中的 API 函数定义与数据手册中的介绍进行了统一。<br><br>重新组织了 .inc 与 .asm 文件里的预编译程序指令。 |
| 1.80.b | DHA | 注释格式从 “:” 更改为 “//”。 |
| 1.90 | DHA | 更新变量名 “@INSTANCE_NAME`_DoBufferRepeatStart”，使其在 I2C 设备上支持两种 CY8C28X45 用户模块实例。 |
| 1.90.b | DHA | 1. 向该用户模块基本介绍的 I2C Clock 参数描述中添加一条关于 Clock Dependency on SYSCLK （时钟对 SYSCLK 的依存关系）的注释。<br><br>2. 更新本用户模块数据手册中的直流与交流电气特性一节的内容。 |
| 1.90.c | DHA | 更新本用户模块基本介绍中从器件模式的固件代码示例。 |

**Note** PSoC Designer 5.1 在所有用户模块基本介绍中都引入了 “ 版本历史 ”。 本数据表详细介绍了当前和先前用户模块版本之间的区别。

Copyright © 2012 Cypress Semiconductor Corporation. The information contained herein is subject to change without notice. Cypress Semiconductor Corporation assumes no responsibility for the use of any circuitry other than circuitry embodied in a Cypress product. Nor does it convey or imply any license under patent or other rights. Cypress products are not warranted nor intended to be used for medical, life support, life saving, critical control or safety applications, unless pursuant to an express written agreement with Cypress. Furthermore, Cypress does not authorize its products for use as critical components in life-support systems where a malfunction or failure may reasonably be expected to result in significant injury to the user. The inclusion of Cypress products in life-support systems application implies that the manufacturer assumes all risk of such use and in doing so indemnifies Cypress against all charges.

PSoC Designer™ and Programmable System-on-Chip™ are trademarks and PSoC® is a registered trademark of Cypress Semiconductor Corp. All other trademarks or registered trademarks referenced herein are property of the respective corporations.

Any Source Code (software and/or firmware) is owned by Cypress Semiconductor Corporation (Cypress) and is protected by and subject to worldwide patent protection (United States and foreign), United States copyright laws and international treaty provisions. Cypress hereby grants to licensee a personal, non-exclusive, non-transferable license to copy, use, modify, create derivative works of, and compile the Cypress Source Code and derivative works for the sole purpose of creating custom software and or firmware in support of licensee product to be used only in conjunction with a Cypress integrated circuit as specified in the applicable agreement. Any reproduction, modification, translation, compilation, or representation of this Source Code except as specified above is prohibited without the express written permission of Cypress.

Disclaimer: CYPRESS MAKES NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS MATERIAL, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. Cypress reserves the right to make changes without further notice to the materials described herein. Cypress does not assume any liability arising out of the application or use of any product or circuit described herein. Cypress does not authorize its products for use as critical components in life-support systems where a malfunction or failure may reasonably be expected to result in significant injury to the user. The inclusion of Cypress' product in a life-support systems application implies that the manufacturer assumes all risk of such use and in doing so indemnifies Cypress against all charges.

Use may be limited by and subject to the applicable Cypress software license agreement.