I²C 硬件模块数据表 I2CHW V 1.5

Copyright © 2003-2010 Cypress Semiconductor Corporation. All Rights Reserved.

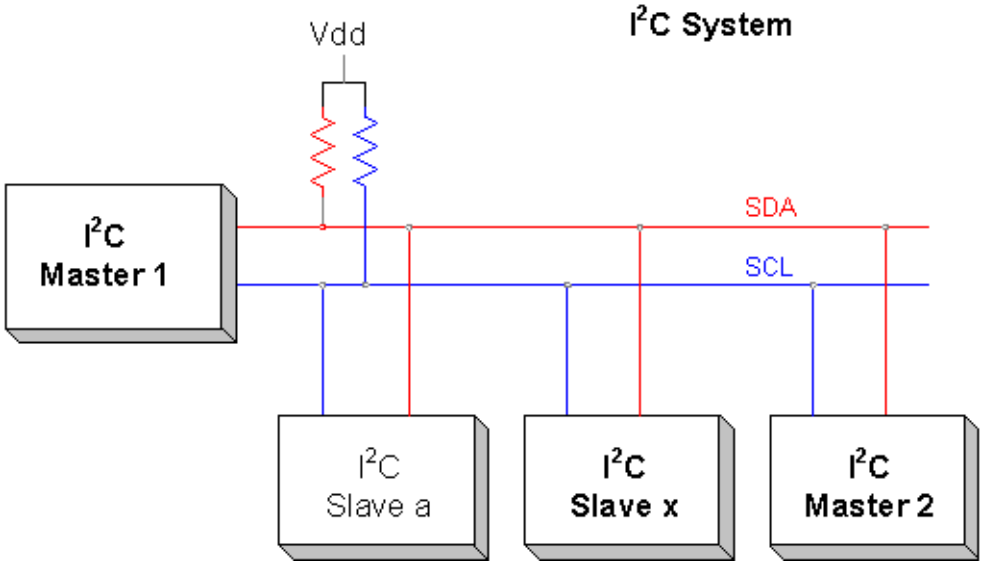
资源	PSoC® 模块			API 存储器 （字节）		引脚 （每个外部 I/O）
	CapSense®	I²C/SPI	定时器	闪存	RAM	
CY8C20x34、CY8C20x24 （仅从器件）		X		279-440	8	

特性与概述

- 行业标准 Philips I²C 总线兼容接口。
- 仅从器件操作，多主控可用。
- 只需两个引脚 （SDA 和 SCL）来连接 I²C 总线。
- 标准数据率为 100/400 kbits/s，也支持 50 kbits/s。
- 高层 API 需要最少用户编程。
- 7-bit 寻址模式，支持 10-bit 寻址。

I2C 硬件用户模块在固件中实现 I²C 从器件。 I²C 总线是 Philips® 开发的基于行业标准的两线硬件接口。 主控可启动 I²C 总线上的所有通信并为所有从器件提供时钟。 I2CHW 用户模块支持最高速度为 400 kbits/s 的标准模式。 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



功能说明

此用户模块支持 I^2C 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 kbits/s 的速度传输数据。 可以使用较慢的 CPU 时钟，但可能导致寻址或数据处理过程中总线有时停止。 I^2C 规范允许主控以从 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。 提供的 API 支持 7 位寻址模式，不过，如果用户扩展至 API 设置，还支持 10-bit 寻址。

I^2C 资源支持逐字节级别的数据传输。 在每个地址或数据发送 / 接收结束时，将报告状态或触发专用中断。 状态报告和中断生成取决于数据传输方向和硬件检测到的 ^2C 总线的条件。 可将中断配置为在字节完成、检测到总线错误和仲裁失败时发生。

每个 I^2C 数据操作都包括 “ 开始 ”(Start)、“ 地址 ”(Address)、“ 读 / 写方向 ”(R/W Direction)、“ 数据 ”(Data) 和 “ 终止 ”(Termination)。 此用户模块使用的 I^2C 资源可以仅作为 I^2C 从器件进行操作。 此用户模块提供基于中断的缓冲传输机制。 通信通过前台函数调用启动。 完成消息的每个字节时，将触发中断并停止 I^2C 总线。用户执行初始化后，提供的中断服务子程序 (ISR) 将在总线上执行适当操作以使通信继续。 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。 从器件必须以确认或否认来响应每个地址。

如果忽略主控与从器件之间的区别，则大致可以说存在两种器件，即接收器和发射器。 对于 I^2C 接收器，中断出现在传入数据的第 8th 位之后。 此时接收器件必须决定确认 (ack) 或否认 (nak) 传入字节为地址或数据。 然后接收器件向 I2C_SCR 寄存器写入相应的控制位，告知 I^2C 资源 ack/nak 状态。 通过解除总线锁止、将 ack/nak 状态传入总线并移入下一位数据字节，向 I2C_SCR 寄存器的写入操作得以在总线上传输数据流。 对于第二种器件传输器，中断出现在外部接收器件提供 ack 或 nak 之后。 可读取 I2C_SCR 以决定此位的状态。 对于传输器，将数据加载到 I2C_DR 寄存器，并再写入到 I2C_SCR 寄存器以触发传输的下一部分。

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(...)）的函数调用的一部分被调用。

可在 Internet 上获取 I^2C 总线和资源实现的详细描述，或参考 PSoC Designer 器件数据表。

设计注意事项

与 I^2C 的软件实现不同，此实现使用内部生成的时钟运行。 将主振荡器设置为任意时钟频率。 数据吞吐量受处理器速度影响，但逐字节数据传输以指定的 I^2C 速度运行。

从器件保留主控可访问的剩余缓冲空间的内部计数。 变量的名称为 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加入适当的 “extern” 声明进行访问。 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控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函数调用是 UMNAME_InitSlaveWrite、UMNAME_InitSlaveRamRead 和 UMNAME_InitSlaveFlashRead.

在 I^2C 从器件内部，使用缓冲区来读写数据。 在启动 I^2C 从器件之前，您必须对相应的缓冲区进行初始化。 I^2C 主控初始化某个读或写操作后，将在 I2CHW_Status 字节设置适当的状态位。 从器件中的前台进程将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) 限制了对缓冲区的访问，长度不允许超过在输入 ISR 时所定义的长度。 对缓冲区的读取和写入以这种方式处理。

1. 如果 I^2C 主控试图读取多于缓冲区所包含的数据，将重新发送最后一个字节，直到 I^2C 主控停止读取。 （I^2C 协议未定义使 I^2C 从器件停止主控读取操作的方法。）

[+] Feedback

2. 若 I²C 主控收到数据并写入到 I²C 从器件时，确定没有更多可用存储，则从器件将在接收最后一个字节时生成 NAK。 如果 I²C 主控继续写入数据，从器件将继续生成 NAK。 一旦生成第一个 NAK （数据存储在最后的可用位置），将不再存储后续数据。
3. 如果缓冲区的长度定义为 0， 则写入 I²C 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还可允许使用 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，都使用提供的 API 进行配置。

动态重新配置

我们建议将 I2CHW 资源加入动态加载 / 卸载程序层。 将 I2CHW 资源仅作为基础配置的一部分放置。 根据运行要求所述内容修改 I2CHW 模块操作，但试图移除作为动态重新配置一部分的资源可能造成对外部 I²C 器件的不利影响。

I²C 寻址

I²C 地址包含在读取或写入数据操作的第一个字节的前 7 位中。 该字节由 I²C 主控用于对从器件寻址。 有效的选择范围为 0-127(dec)。 该字节的 LSB 包含 R/~W 位。 如果该位为 0，则写入地址；如果 LSB 为 1，则已寻址的从器件从该位读取数据。

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。 传递单独的参数以包含读 / 写信息。 I²C 主控发送字节 (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。

由于从器件模块的地址参数接受基于十进制的数值输入，应以十进制 （十进制 72）输入 7-bit 地址。

外部电气连接

如框图显示，I²C 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。 对于输出级，当 V_{OLmax} = 0.4V 时，所有器件（主控或从器件）的最小灌电流不小于 3 mA。 这将 5-volt 系统最小上拉电阻值限制在大约 1.5 千欧。 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 150 pF 的 5-volt 系统，上拉电阻不大于 6 千欧。 有关 I²C 总线规范的更多信息，请参阅 Philips 网站。网址为 www.philips.com。

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 I²C 组件，即可根据赛普拉斯 I²C 专利权获得一份许可，以便在 I²C 系统中使用这些组件，但前提是该系统符合赛普拉斯定义的 I²C 标准规范。

直流和交流电气特性

请参见器件数据表 “ 直流和交流电气特性 ” 部分的器件特性数据。

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]。没有放置限制。 不存在 I²C 模块的多个放置，因为 I²C 模块使用专用 PSoC 资源模块和中断。

[+] Feedback

参数和资源

所有缓冲区名称的定义均描述其对 I^2C 主控的用途。 例如，I2Cs_pRead_Buf 是指在 RAM 中的位置，该位置包含供 I^2C 主控读取的数据。

Slave_Addr

Slave_addr 选择 7-bit 从器件地址，该地址由 I^2C 主控用于对从器件寻址。 有效的选择范围为 0-127(dec)。

I2C_Clock

指定 I^2C 接口运行所需的时钟频率。 有三种可能的时钟频率。

I2C_Clock 频率
50 K 标准
100 K 标准
400 K 快速

I2C_Pin

从端口 1 选择用于 I^2C 信号的引脚。 不需要为这些引脚选择合适的驱动模式，因为 PSoC Designer 会自动选择。

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” 或 “RAM 或闪存 ”。 “ 仅 RAM” 选项会删除支持直接闪存 ROM 读取所需的代码和变量。 “RAM 或闪存 ” 选项为读取 RAM 缓冲区或闪存 ROM 缓冲区提供代码和变量支持，以便将数据传送至主控。 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。 然后，数据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，用户可以控制数据移动出现的时间。 为了实现轮询策略，用户必须定期调用函数 I2CHW_Poll()（准确的实例名称请参见 I2CHWslave.h 文件）。 每次调用轮询函数时传输单个字节。 其他 I^2C 函数的使用完全相同。 在中断延迟具有关键重要性 （而且异步通信中断可能导致问题）的情况下使用轮询通信策略。 另一种使用方式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后自动停顿，直到调用轮询函数时为止。

中断生成控制

下列参数仅在选中了 PSoC Designer 中 “ 启用中断生成控制 ”(Enable interrupt generation control) 复选框时可用。该复选框位于 “ 项目 ” > “ 设置 ” > “ 芯片编辑器 ” 之下。 **“ 启用中断生成控制 ”** 复选框时，有两个附加参数将变为可用。 此复选框位于 **“ 项目 ” > “ 设置 ” > “ 芯片编辑器 ” 之下。**.

IntDispatchMode

IntDispatchMode 参数用于指定如何处理中断请求 （当处于同一模块的多个用户模块在不同的程序层共享该中断时）。 选择 **ActiveStatus** 会导致固件在为共享的中断请求提供服务之前测试哪一个程序层正处于活动状态。 这项测试发生在每次请求共享中断的时候。 这样会导致延迟增加并且会产生一

[+] Feedback

个服务于共享中断请求的非确定性的程序，但并不要求任何 RAM 资源。 选择 **OffsetPreCalc** 参数会导致固件只在最初已经有一个程序层运行时计算共享中断请求的来源。 这项计算减少了中断延迟会产生一个服务于共享中断请求的确定性程序，但其代价是 1 个字节的 RAM 资源。

应用程序编程接口

应用程序编程接口 (API) 固件提供支持发送和接收多字节传输的高级命令。 在 RAM 或者闪存中创建读取缓冲区。 只能在 RAM 存储器中设置写入缓冲区。

Note 在这里，如同所有用户模块 API 中的一样，A 和 X 寄存器的值通过调用 API 函数来更改。 发起调用 API 的函数有责任在调用之前保留 A 和 X 的值 （如果调用后需要再次用到它们）。 选择这种 “ 寄存器易变 ” 策略是为了提高效率，并且自从 PsoC Designer 的 1.0 版本起使用。 C 语言编译器自动处理此项要求。 汇编语言程序员也必须确保其代码遵守这一策略。 虽然有些用户模块的 API 函数可能保留 A 和 X 不变，但并不保证在未来也将如此。

I2CHW_Start

说明：

不做任何操作。 只为保证接口一致性。

C 语言原型：

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

汇编程序：

```
lcall  I2CHW_Start
```

参数：

无

返回值：

无

副作用：

寄存器 A 和寄存器 X 有可能在此函数的当前版本或后续版本中被修改。 在大型存储器模型 (CY8C29xxx) 中，所有 RAM 页指针寄存器也会出现这种情况。 在必要时，调用函数有责任在调用 fastcall16 函数之前保存这些值。

I2CHW_Stop

说明：

通过禁用 I^2C 中断禁用 I^2CHW。

C 语言原型：

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

汇编程序：

```
lcall  I2CHW_Stop
```

参数：

无

返回值：

无

[+] Feedback

副作用：

寄存器 A 和寄存器 X 有可能在此函数的当前版本或后续版本中被修改。 在大型存储器模型 (CY8C29xxx) 中，所有 RAM 页指针寄存器也会出现这种情况。 在必要时，调用函数有责任在调用 fastcall16 函数之前保存这些值。

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宏来调用全局中断启用函数：M8C_EnableGInt.

C 语言原型：

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

汇编程序：

```
lcall  I2CHW_EnableInt
```

参数：

无

返回值：

无

副作用：

寄存器 A 和寄存器 X 有可能在此函数的当前版本或后续版本中被修改。 在大型存储器模型 (CY8C29xxx) 中，所有 RAM 页指针寄存器也会出现这种情况。 在必要时，调用函数有责任在调用 fastcall16 函数之前保存这些值。

I2CHW_DisableInt

说明：

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

C 语言原型：

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

汇编程序：

```
lcall I2CHW_DisableInt
```

参数：

无

返回值：

无

寄存器 A 和寄存器 X 有可能在此函数的当前版本或后续版本中被修改。 在大型存储器模型 (CY8C29xxx) 中，所有 RAM 页指针寄存器也会出现这种情况。 在必要时，调用函数有责任在调用 fastcall16 函数之前保存这些值。

[+] Feedback

I2CHW_Poll

说明：

在 Communication_ Service_Type 参数设置为 “ 轮询 ”(Polled) 时使用。 此函数提供进入 I/O 处理子程序的由用户控制的入口。 如果将 Communication_Service_Type 参数设置为 “ 中断 ”(Interrupt)，则该函数不执行任何操作。

注： 调用 I2CHW_Poll 释放 I2C 总线，这允许主控在从器件固件处理完上一个请求之前读取或写入数据。

C 语言原型：

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

汇编程序：

```
lcall I2CHW_Poll
```

参数：

无

返回值：

无

副作用：

每次调用该子程序时，将处理一个 I^2C 事件，而状态变量将得到更新。 事件的构成可以是一个故障状况、一个 I/O 字节，或在某些特定情况下，一个停止状况。 调用该子程序可能有三种结果：

1. 如果没有数据，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. 当外部主控完成写入操作时处理停止事件。

在写入操作结束阶段处理停止状态时仅更新状态变量。 当处理停止状态时，如果 I^2C 字节正等待解决，再次调用 I2CHW_Poll 函数来处理。 如果设置 Communication_Service_Type 为 “ 中断 ”(Interrupt)，则 I2CHW_Poll() 函数没有影响。 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，直至调用 I2CHW_Poll() 函数。 如果地址被否认，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。

如果将 Communication_Service_Type 设置为 “ 轮询 ”(Polled)，I^2C 硬件将使 I^2C 数据停顿，直到调用该函数。 对于已接收的数据，总线将在字节末尾并在生成 ACK/NAK 之前通过将 SCL （时钟）线保持在低位而停顿。 对于已发送的数据，总线将在 ACK/NAK 位在外部生成之后立即停顿。

I2CHW_EnableSlave

说明：

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

C 语言原型：

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

汇编程序：

```
lcall  I2CHW_EnableSlave
```


[+] Feedback

参数：

无

返回值：

无

副作用：

寄存器 A 和寄存器 X 有可能在此函数的当前版本或后续版本中被修改。 在大型存储器模型 (CY8C29xxx) 中，所有 RAM 页指针寄存器也会出现这种情况。 在必要时，调用函数有责任在调用 fastcall16 函数之前保存这些值。

正如在 CY8C27xxxA 数据表中指定的，I^2C 以 6MHz CPU 时钟频率写入配置寄存器。 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，否则会发生 I^2C 通信故障。

I2CHW_DisableSlave

说明：

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

C 语言原型：

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

汇编程序：

```
lcall  I2CHW_DisableSlave
```

参数：

无

返回值：

无

副作用：

寄存器 A 和寄存器 X 有可能在此函数的当前版本或后续版本中被修改。 在大型存储器模型 (CY8C29xxx) 中，所有 RAM 页指针寄存器也会出现这种情况。 在必要时，调用函数有责任在调用 fastcall16 函数之前保存这些值。

正如在 CY8C27xxxA 数据表中指定的，以 6MHz CPU 时钟频率写入 I^2C 配置寄存器。 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，否则会发生 I^2C 通信故障。

I2CHW_bReadI2CStatus

说明：

返回控制 / 状态寄存器的状态位。

C 语言原型：

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

汇编程序：

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

参数：

无

[+] Feedback

返回值：

```
BYTE  I2CHW_RsrcStatus
```

参见表格。

副作用：

寄存器 A 和寄存器 X 有可能在此函数的当前版本或后续版本中被修改。 在大型存储器模型 (CY8C29xxx) 中，所有 RAM 页指针寄存器也会出现这种情况。 在必要时，调用函数有责任在调用 fastcall16 函数之前保存这些值。 目前，只修改了 CUR_PP 页面指针寄存器。

常量	值	说明
I2CHW_RD_NOERR	01h	主控读取数据，正常 ISR 退出
I2CHW_RD_ OVERFLOW	02h	主控所读取的数据字节超出了可用字节数
I2CHW_RD_ COMPLETE	04h	已开始读取操作且尚未完成
I2CHW_READFLASH	08h	下一个读取操作来自闪存位置
I2CHW_WR_NOERR	10h	主控成功写入了数据
I2CHW_WR_OVERFLOW	20h	主控向写入缓冲区写入了过多字节
I2CHW_WR_COMPLETE	40h	主控写入操作因新地址或停止而结束
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 语言原型：

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

汇编程序：

```
lcall  I2CHW_ClrRdStatus
```

参数：

无

返回值：

无

副作用：

寄存器 A 和寄存器 X 有可能在此函数的当前版本或后续版本中被修改。 在大型存储器模型 (CY8C29xxx) 中，所有 RAM 页指针寄存器也会出现这种情况。 在必要时，调用函数有责任在调用 fastcall16 函数之前保存这些值。 目前，只修改了 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。

[+] Feedback

C 语言原型：

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

汇编程序：

```
lcall  I2CHW_ClrWrStatus
```

参数：

无

返回值：

无

副作用：

寄存器 A 和寄存器 X 有可能在此函数的当前版本或后续版本中被修改。 在大型存储器模型 (CY8C29xxx) 中，所有 RAM 页指针寄存器也会出现这种情况。 在必要时，调用函数有责任在调用 fastcall16 函数之前保存这些值。 目前，只修改了 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，并初始化相同缓冲区的计数字节值。 将计数初始化至提供的最大缓冲区长度。 在下一个主控写入的实例中，将数据置于由此函数定义的地址。

C 语言原型：

```
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
                                    ; (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: 写入缓冲区的长度。

[+] Feedback

返回值：

无

副作用：

寄存器 A 和寄存器 X 有可能在此函数的当前版本或后续版本中被修改。 在大型存储器模型 (CY8C29xxx) 中，所有 RAM 页指针寄存器也会出现这种情况。 在必要时，调用函数有责任在调用 fastcall16 函数之前保存这些值。 目前，只修改了 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，从器件从该指针检索数据，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，导致下一次读取操作将尝试从 RAM 中*先前*设置的缓冲区位置读取数据。

C 语言原型：

```
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
                                    ; (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：

无

副作用：

寄存器 A 和寄存器 X 有可能在此函数的当前版本或后续版本中被修改。 在大型存储器模型 (CY8C29xxx) 中，所有 RAM 页指针寄存器也会出现这种情况。 在必要时，调用函数有责任在调用 fastcall16 函数之前保存这些值。 目前，只修改了 CUR_PP 页面指针寄存器。

[+] Feedback

I2CHW_InitFlashRead

说明：

初始化闪存数据缓冲器指针用于检索数据。 设置 I2CHW_SlaveStatus 标志 I2CHW_READFLASH 至 1，导致下一次读取操作将尝试从闪存中*先前*设置的缓冲区位置读取数据。

C 语言原型：

```
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：

无

副作用：

寄存器 A 和寄存器 X 有可能在此函数的当前版本或后续版本中被修改。 在大型存储器模型 (CY8C29xxx) 中，所有 RAM 页指针寄存器也会出现这种情况。 在必要时，调用函数有责任在调用 fastcall16 函数之前保存这些值。 目前，只修改了 CUR_PP 页面指针寄存器。

[+] Feedback

固件源代码示例

以下为 I2CHW 从器件用户模块的一个实现示例：

```
/*************************************************************/
/* This sample code echoes bytes received from a master */
/* The master sends and requests up to 64 bytes.          */
/*                                                          */
/* The instance name of the I2CHWs User Module is defined */
/* as I2CHW.                                               */
/*                                                          */
/*  NOTE I2CHW does not depend upon the CPU clock         */
/*************************************************************/
#include <m8c.h>
#include <i2CHWCommon.h>

/* setup a 64 byte buffer */
BYTE    abBuffer[64];
BYTE    status;

void EchoData()
{
/* 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(abBuffer,64);
    I2CHW_InitWrite(abBuffer,64);


    /* Echo forever */
    while( 1 )
    {
        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(abBuffer,64);
        }
        /* wait until data is echoed */
        /* want to know if RD_NOERR is SET AND RD_COMPLETE is SET */
        if( status & I2CHW_RD_COMPLETE )
        {
        /* Data echoed - clear the read status */
        I2CHW_ClrRdStatus();
        /* Reset the pointer for the next data to echo */
        I2CHW_InitRamRead(abBuffer,64);
        }
    }
```


[+] Feedback

```
}
void main(void)
{
    EchoData();
}
```

以下是用汇编代码编写的配置为从器件的 I2CHW 用户模块实现：

```
;-------------------------------------------------------------
; this assembly assumes small memory model
;-------------------------------------------------------------
include "m8c.inc"
include "I2CHW_1Common.inc"
export rec_cnt
export i2c_addr
;export master_state

INT_MSK3:  equ 0xDE
BUFFERSIZE: equ 64

export abBuffer
export rec_cnt
export status

area bss(RAM)

rec_cnt: blk 1
i2c_addr: blk 1
abBuffer: blk BUFFERSIZE
status:   blk 1
;master_state: blk 1


area text(ROM, REL)

export _main

_main:
    ;enable the I2C slave as an ISR based process

    call I2CHW_1_EnableSlave
call I2CHW_1_EnableInt
M8C_EnableGInt

Init:
mov A, BUFFERSIZE
push A
push A
mov A, <abBuffer
push A
mov X, sp
dec X
call I2CHW_1_InitWrite
;
;
;everything should still be initialized on the stack to call initRamRead
```


[+] Feedback

```
call I2CHW_1_InitRamRead
add SP, -3

checkI2CStatus:
    call I2CHW_1_bReadI2CStatus
    and A, I2CHW_WR_NOERR
    jnz writeHappened
    and A, I2CHW_RD_NOERR
    jnz readHappened

    jmp checkI2CStatus



readHappened:
    ;call I2CHW_3_ClrRdStatus
    ;calculate bytes read
    mov A, BUFFERSIZE
    sub A, [I2CHW_1_Read_Count]
    inc A
    cmp A, BUFFERSIZE
    jnz checkI2CStatus
    jmp _main

writeHappened:
    ;call I2CHW_3_ClrWrStatus
    ;calculate bytes read
    mov A, BUFFERSIZE
    sub A, [I2CHW_1_Write_Count]
    inc A
    cmp A, BUFFERSIZE
    jnz checkI2CStatus
    jmp Init


;end _main
```

配置寄存器

本节介绍由 I2CHW 用户模块使用或者修改的 PSoC 资源寄存器。

Table 1. 资源 I2C_CFG：组 0 reg[D6] 配置寄存器

位	7	6	5	4	3	2	1	0
值	保留	PinSelect	保留	停止 IE	时钟 Rate[1]	时钟 Rate[0]	保留	使能

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

停止错误中断允许：在 I^2C 停止条件下允许 I^2C 中断。

时钟频率 [1,0]：在 3 个有效时钟频率中选择 50- 100- 400kbps。

启用：启用 I^2C HW 模块。

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

位	7	6	5	4	3	2	1	0
值	总线错误	保留	停止状态	确认状态	地址	发送	最后接收位 (LRB)	字节完成

总线错误：表示检测到总线错误条件。

停止状态：表示检测到 I^2C 停止条件。

确认：指示 I^2C 模块对所收到的字节进行确认 (1) 或否认 (0)。

地址：所接收或所发送的字节是一个地址。

发送：此位设置后续字节传输的移位器方向。移位器总是在来自 I^2C 总线的数据中移位，但是写入“1”将启用移位器的输出来驱动 SDA 输出行。由于对此寄存器的写入操作即启动下一个传输，因此必须在写入此位之前将数据写入到数据寄存器。在接收模式中 (0)，在本次写入前，之前接收的数据必须已从数据寄存器中读取。固件从已接收从器件地址中的 RW 位中衍生出此方向。此方向控制仅对数据传输有效。硬件决定地址字节的方向。

最后接收位 (LRB)：发送序列中最后一个接收位 （第 9 位）的值，来自目标器件的确认 / 否认状态。

字节完成：已收到 8 个数据位。对于接收模式，总线处于停顿状态，以等待确认 / 否认应答。对于发送模式，且已经收到确认 / 否认应答 （参见 LRB），总线处于停顿状态以等待执行下一个操作。

Table 3. 资源 I2C_DR：组 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。要传送数据，必须在写入 I2C_SCR 寄存器之前加载此寄存器。所接收的数据从此寄存器中读取。其中可能包含地址或数据。

[+] Feedback

版本历史记录

版本	创作者	说明
1.5	DHA	以下是对 “ 启动 ”(Start) 函数所做的更改： 1. 将用户模块引脚的初始开漏低电平驱动模式更改为 HI-Z 模拟。 2. 启用 I^2C 模块。 3. 延迟 5 个正常操作程序指令。 4. 恢复初始 I^2C 引脚驱动模式。

Note PSoC Designer 5.1 在所有用户模块数据表中提供版本历史记录。 本部分记录了当前和先前用户模块版本之间区别的高级描述。

Copyright © 2003-2010 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.