

I²C 控制器对电路板的集成电路之间的通信链路提供支持。它是一个包含串行数据线 (SDA) 和串行时钟 (SCL) 的简单的两线总线，可用于例如温度传感器、到 EEPROMs 的电压型转换器、A/D 和 D/A 转换器、CODECs 以及多种类型的微处理器应用程序。

硬核处理器系统 (HPS) 提供 4 个 I²C 控制器以使能系统软件和 I²C 总线进行串行地通信。每个 I²C 控制器均可在主器件或从器件模式中操作，并且支持高达每秒 100 kilobits(Kbps) 的标准模式或高达 400 Kbps 的快速模式。这些 I²C 控制器是 Synopsys® DesignWare® APB I²C (DW_apb_i2c) 控制器的实例。



每个 I²C 控制器必须被配置以便只在主器件模式或从器件模式中操作。不支持同时在主器件和从器件模式中操作。

I²C 控制器的功能

I²C 控制器具有以下功能：

- 高达 400 Kbps 的最高时钟速度
- 实现以下 I²C 操作的其中之一：
 - I²C 系统中的主器件并且仅编程为一个主器件
 - I²C 系统中的从器件并且仅编程为从器件
- 7- 或 10-bit 寻址
- 7-bit 和 10-bit 寻址模式中的混合读和写合并格式传输
- Bulk 发送模式
- 发送和接收缓冲器
- 处理所有总线速度的位和字节等待
- DMA 握手接口

© 2012 Altera Corporation. All rights reserved. ALTERA, ARRIA, CYCLONE, HARDCOPY, MAX, MEGACORE, NIOS, QUARTUS and STRATIX words and logos are trademarks of Altera Corporation and registered in the U.S. Patent and Trademark Office and in other countries. All other words and logos identified as trademarks or service marks are the property of their respective holders as described at www.altera.com/common/legal.html. Altera warrants performance of its semiconductor products to current specifications in accordance with Altera's standard warranty, but reserves the right to make changes to any products and services at any time without notice. Altera assumes no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to in writing by Altera. Altera customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.

Portions © 2011 Synopsys, Inc. Used with permission. All rights reserved. Synopsys & DesignWare are registered trademarks of Synopsys, Inc. All documentation is provided "as is" and without any warranty. Synopsys expressly disclaims any and all warranties, express, implied, or otherwise, including the implied warranties of merchantability, fitness for a particular purpose, and non-infringement, and any warranties arising out of a course of dealing or usage of trade.

†Paragraphs marked with the dagger (†) symbol are Synopsys Proprietary. Used with permission.



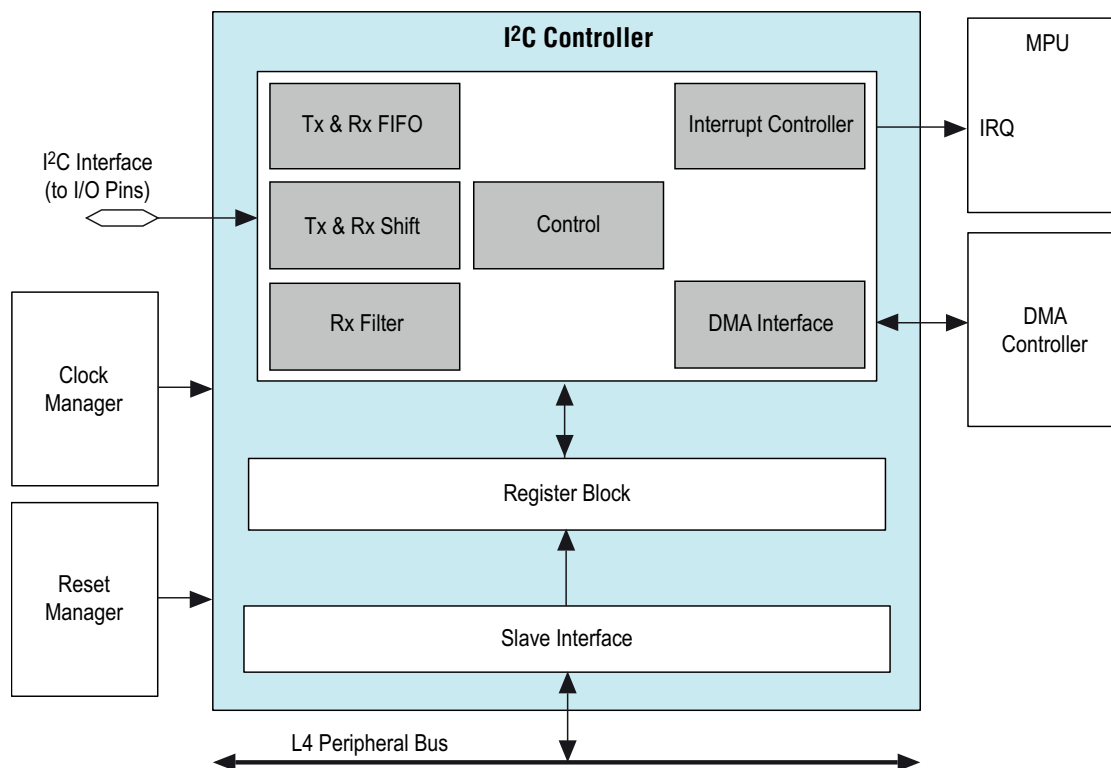
I²C 控制器结构图和系统集成

I²C 控制器包含一个从接口、I²C 接口以及两个接口之间的缓冲器数据的 FIFO 逻辑。

主机处理器通过 32-bit 从接口访问关于 I²C 控制器的数据、控制和状态信息。

图 20-1 显示了 I²C 控制器结构图。

图 20-1. I²C 控制器结构图



I²C 控制器包含以下模块和接口：

- 用于控制和状态寄存器 (CSR) 访问以及 DMA 传输的从接口，可支持主器件访问 CSR 和 DMA 以便直接读或写数据。
- 用于发送和接收数据的两个 FIFO 缓冲器，持有 Rx FIFO 和 Tx FIFO 缓冲寄存器 bank 和控制器，以及它们的状态水平。
- 并行到串行和串行到并行转换的移位逻辑
 - Rx 移位 - 将数据接收到设计并且以字节格式将其提取。
 - Tx 移位 - 呈现 CPU 提供的数据以便在 I²C 总线上传输。
- 负责实现 I²C 协议的控制逻辑
- DMA 接口，可生成到 DMA 控制器的握手信号以便在没有 CPU 干扰的情况下自动完成数据传输。
- 生成原始中断和中断标记的中断控制器，支持它们被设置和清除。
- 检测事件的接收滤波器，例如总线中的启动和停止情况；例如，开始、停止和失去仲裁。

I²C 控制器的功能说明

该部分介绍 I²C 控制器的功能说明。

功能使用

I²C 控制器可在标准模式（数据速率 0 到 100 Kbps）或快速模式（数据速率小于或等于 400 Kbps）中操作。除此之外，快速模式器件是向下兼容。例如，快速模式器件可以与 0 到 100 Kbps I²C 总线系统中的标准模式器件通信。然而，标准模式器件不是向上兼容的并且不应该内置在快速模式 I²C 总线系统中，因为它们不能够遵循较高传输速率，因此会出现不可预测状态。

您可将任何的 I²C 控制器连接到 I²C 总线并且每个器件都可以和任何主器件通信，来回地传递信息。总线上至少需要一个主器件（例如微控制器或 DSP），也可能为多个主器件，从而要求它们仲裁所有权。多个主器件和仲裁会在该章节的后面解释。

行为

可以通过将软件，设置 I²C 控制器工作在下面两种模式：

- 仅为 I²C 主器件模式，与其它 I²C 从器件通信；或
- 仅为 I²C 从器件模式，与一个或多个 I²C 主器件通信

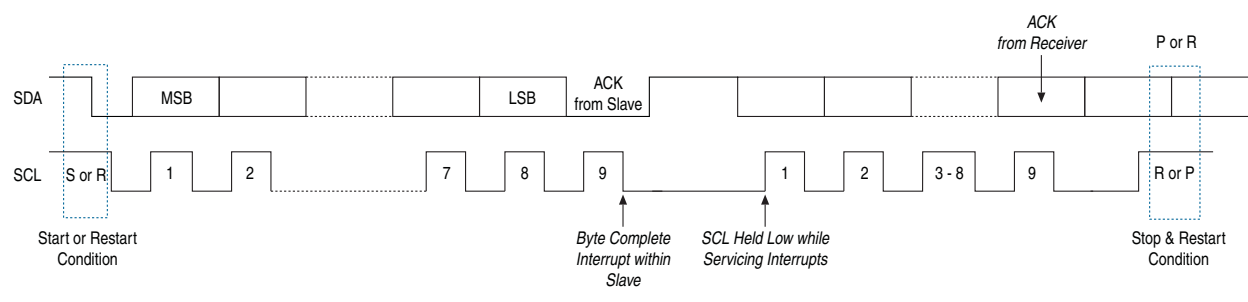
主器件负责生成时钟和控制数据传输。从器件负责发送数据到主器件或从主器件接收数据。数据的确认信号由接收数据的器件发送，可以为一个主器件或从器件。如同之前所提到的，I²C 协议也支持多个主器件位于 I²C 总线并且使用一个仲裁程序决定总线所有权。

每个从器件都有一个系统设计者决定的独特地址。当一个主器件想要和从器件通信时，主器件发送一个 START/RESTART 条件，然后一个从器件地址和控制位 (R/W) 以决定主器件想要发送数据到从器件或接收从器件数据。然后从器件在地址后发送一个确认 (ACK) 脉冲。

如果主器件（主 - 发送器）写入从器件（从 - 接收器），那么接收器接收到一个字节的的数据。该传输继续进行直到主器件使用一个 STOP 条件终止传输。如果主器件从从器件（主 - 接收器）读取，那么从器件发送（从 - 发送器）一个字节的的数据到主器件，然后主器件使用一个 ACK 脉冲确认传输。该传输继续进行直到主器件通过在接收到最后字节后不确认 (NACK) 传输而终止传输，然后主器件发出一个 RESTART 条件，随后发出一个 STOP 条件或对另一个从器件寻址。

图 20 - 2 显示了 I²C 总线上的数据传输行为。

图 20 - 2. I²C 总线的数据传输†



I²C 控制器是一个同步串行接口。SDA 线是一个双向信号并且只有在 SCL 线低电平时才更改 (除了 STOP、START 和 RESTART 条件以外)。输出驱动器为开漏或集电极开路，以便执行总线的 wire-AND 功能。总线上的器件的最高数只由 400 pF 的最高电容规范而限制。数据使用字节数据包发送。

START 和 STOP 生成

作为主器件操作时，将数据放入发送 FIFO 会导致 I²C 控制器在 I²C 总线上生成一个 START 条件。使发送 FIFO 变空会导致 I²C 控制器在 I²C 总线上生成一个 STOP 条件。

作为从器件操作时，根据协议，I²C 控制器不生成 START 和 STOP 条件。然而，如果对 I²C 控制器做出读请求，那么它保持 SCL 线为低电平直到已经对它提供了读数据。这会中止 I²C 总线直到读数据被提供到从器件 I²C 控制器，或 I²C 控制器从器件通过将 0 写入 IC_ENABLE 寄存器而被禁用。

合并格式

I²C 控制器支持 7-bit 和 10-bit 寻址模式中的混合读和写合并格式传输。

I²C 控制器不支持混合的地址和混合的地址格式 — 也就是，一个 7 位地址传输，紧接是一个 10 位地址传输或反之亦然 — 合并的格式传输。

要启动合并格式传输，IC_CON 寄存器中的 IC_RESTART_EN 位应该被设置为 1。通过设置该值并且作为主器件操作，当 I²C 控制器完成 I²C 传输时，它检查发送 FIFO 并且执行下一个传输。如果该传输的方向不同于之前的传输，那么合并格式用于发出传输。当前 I²C 传输完成时如果发送 FIFO 为空，那么 STOP 被发出并且下一个传输紧跟 START 条件被发出。

协议详细信息

该部分介绍 I²C 控制器协议。

START 和 STOP 条件

当总线处于闲置状态，scl 和 sda 信号通过总线上的上拉电阻而拉高。当主器件想要开始在总线传输时，主器件发出一个 START 条件。这被定义为 SCL 为 1 时的 SDA 信号从高到低的跳变。当主器件想要终止传输时，主器件发出一个 STOP 条件。这被定义为 SCL 为 1 时的 SDA 线从低到高的跳变。

图 20-3 显示了 START 和 STOP 条件的时序。当数据在总线上发送时，SCL 为 1 时 SDA 线必须稳定。

图 20-3. START 和 STOP 条件的结构图

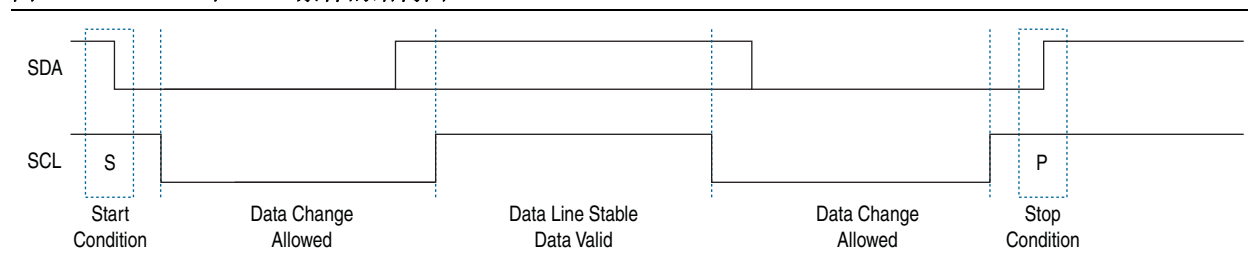


图 20-3 所示的 START 或 STOP 条件的信号跳变，反映了在驱动 I²C 总线的主器件的输出信号所观察的信号。当在从器件的输入信号观察 SDA 或 SCL 信号时应该谨慎，因为不相等的线路延迟会导致一个错误的 SDA 或 SCL 时序关系。

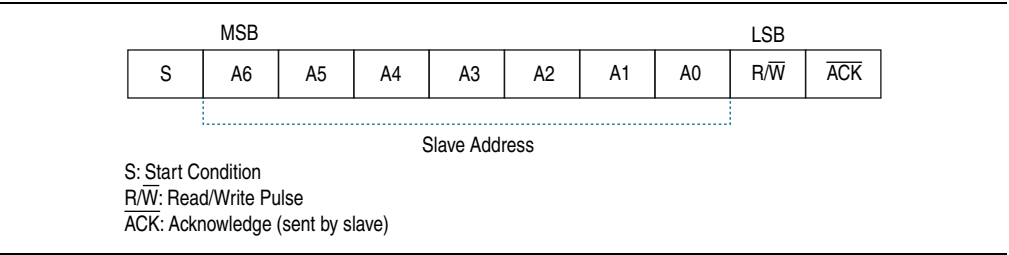
对从器件协议寻址

有两种寻址格式：7-bit 寻址格式和 10-bit 寻址格式。

7-Bit 寻址格式

7-bit 寻址格式中，第一个字节的前 7 个位（位 7:1）设置从器件地址并且 LSB 位（位 0）是图 20 - 4 所示的 R/W 位。当位 0（R/W）被设置为 0 时，主器件写入从器件。当位 0（R/W）被设置为 1 时，主器件从从器件读取。

图 20 - 4. 7-bit 地址格式



10-Bit 寻址格式

10-bit 寻址格式中，两个字节被传输以设置 10-bit 地址。第一个字节的传输包含以下位定义。前 5 个位（位 7:3）提示从器件这是一个下两个位（位 2:1）紧跟其后的 10-bit 传输，从而将从器件地址位设置为 9:8，并且 LSB 位（位 0）是 R/W 位。第 2 个传输的字节设置从器件地址的位 7:0。图 20 - 5 显示了 10-bit 寻址格式。

图 20 - 5. 10-Bit 寻址格式

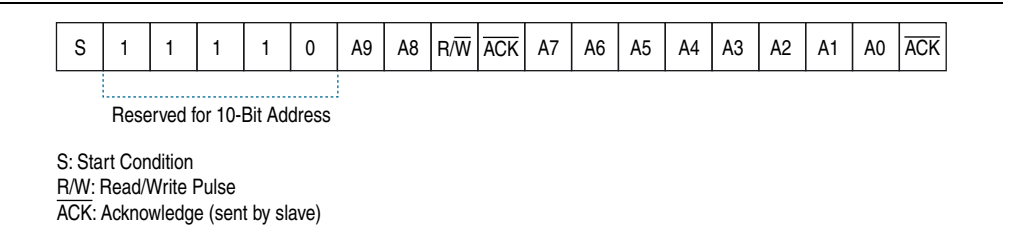


表 20 - 1 定义特殊目的和所保留的第一个字节地址。

表 20 - 1. I²C 定义第一个字节中的位

从器件地址	R/W 位	说明
0000 000	0	一般调用地址。I ² C 控制器将数据放入接收缓冲器并且发出一个一般调用中断。
0000 000	1	START 字节。要了解更多信息，请参考第 20 - 7 页的“START BYTE 传输协议”。
0000 001	X	CBUS 地址。I ² C 控制器忽略这些访问。
0000 010	X	保留
0000 011	X	保留
0000 1XX	X	未使用
1111 1XX	X	保留

表 20 - 1. I²C 定义第一个字节中的位

从器件地址	R/W 位	说明
1111 0XX	X	10-bit 从器件寻址。
表 20 - 1 注释： (1) ‘X’ 表示忽略。		

发送和接收协议

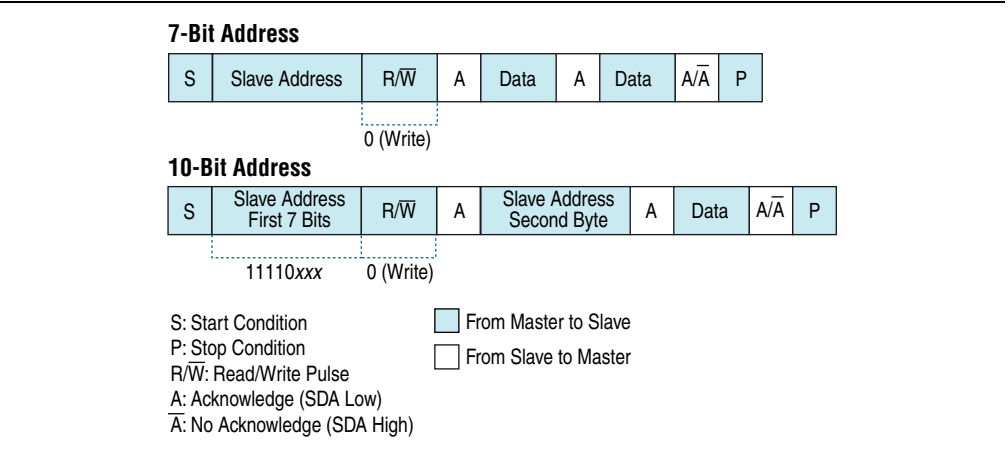
主器件可以发起数据传输到总线或者从总线接收数据，从而作为主器件 - 发送器或主器件 - 接收器。一个从器件通过发送数据到总线或从总线接收数据来响应主器件的请求，从而分别作为一个从器件发送器或从器件接收器。

主器件 - 发送器和从器件 - 接收器

所有数据都使用字节格式发送，每数据传输的字节数没有限制。主器件发送地址和 R/W 位或主器件发送一个字节的的数据到从器件后，从器件 - 接收器必须使用确认信号 (ACK) 做出响应。当从器件 - 接收器没有使用 ACK 脉冲做出响应时，主器件通过发出一个 STOP 条件中断传输。从器件必须使 SDA 线为高电平以便主器件可以中断传输。

如果主器件 - 发送器如图 20 - 6 所示正在发送数据，那么从器件-接收器在接收到数据的每个字节后使用 ACK 脉冲响应主器件 - 发送器。

图 20 - 6. 主器件 - 发送器协议



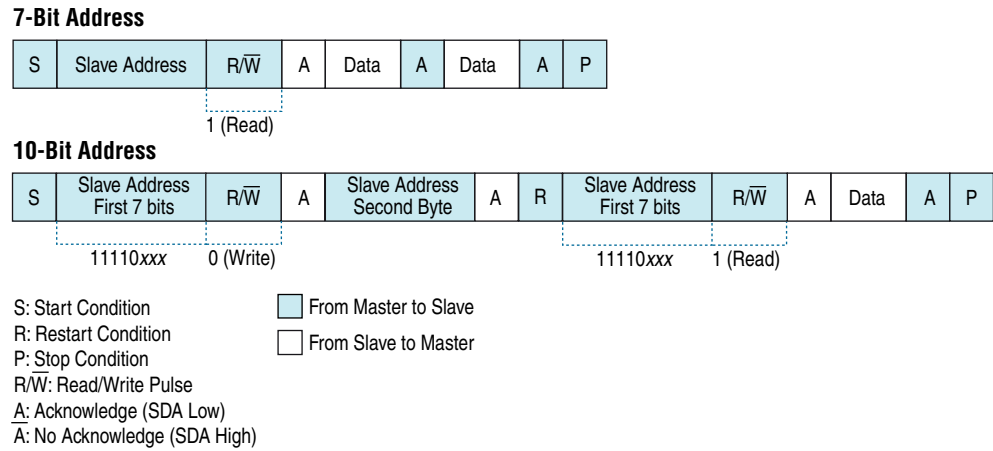
主器件 - 接收器和从器件 - 发送器

如果主器件如图 20 - 7 所示正在接收数据，那么主器件接收到一个字节的的数据后 (除了最后字节) 使用 ACK 脉冲响应从器件 - 发送器。主器件 - 接收器通过这种方式指示从器件 - 发送器这是最后字节。从器件 - 发送器检测到无确认信号 (No Acknowledge, NACK) 位之后丢弃 SDA 线，以便主器件可以发出一个 STOP 条件。

当主器件不想要通过 STOP 条件释放总线时，主器件会发出一个 RESTART 条件。这个等同于 START 条件 (除了它在 ACK 脉冲之后出现以外)。在主器件模式中操作时，I²C 控制器可以通过使用不同方向的传输与相同的从器件通信。要了解关于 I²C 控制器支持的合并格式传输的详细说明，请参考第 20 - 4 页的“合并格式”。

在目标从器件地址寄存器 IC_TAR 可以被重新配置之前，I²C 控制器的串行端口必须处于未激活状态 (I2C_DYNAMIC_TAR_UPDATE = 1)。

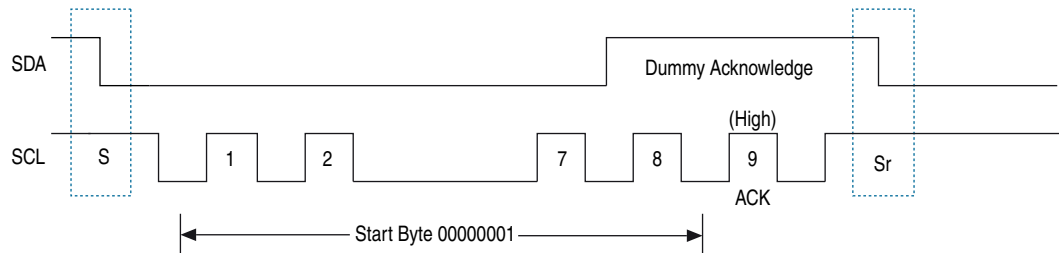
图 20 - 7. 主器件 - 接收器协议



START BYTE 传输协议

START BYTE 传输协议针对没有板上专用 I²C 硬件模块的系统而设置。当 I²C 控制器被设置为从器件时，它总是在所支持的最高速度采样 I²C 总线以便它从不要求一个 START BYTE 传输。然而，当 I²C 控制器被设置为主器件时，每次传输开始，它都支持 START BYTE 传输的生成，以防从器件需要它。该协议包括发送 7 个零然后 1 个一，如图 20 - 8 所示。这样支持轮询总线的处理器可以 (欠采样) 地址相位直到微控制器检测到一个 0。一旦微控制器检测到 0，它从欠采样速率转换到主器件的正确速率。

图 20 - 8. START BYTE 传输



START BYTE 具有以下步骤：

1. 主器件生成一个 START 条件。
2. 主器件发送 START 字节 (0000 0001)。
3. 主器件发送 ACK 时钟脉冲。(只为与总线上使用的字节处理格式保持一致)
4. 没有从器件将 ACK 信号设置为 0。
5. 主器件生成一个 RESTART (R) 条件。

硬件接收器不会响应 START BYTE，因为它是一个保留地址并且在生成 RESTART 条件之后复位。

多个主器件仲裁

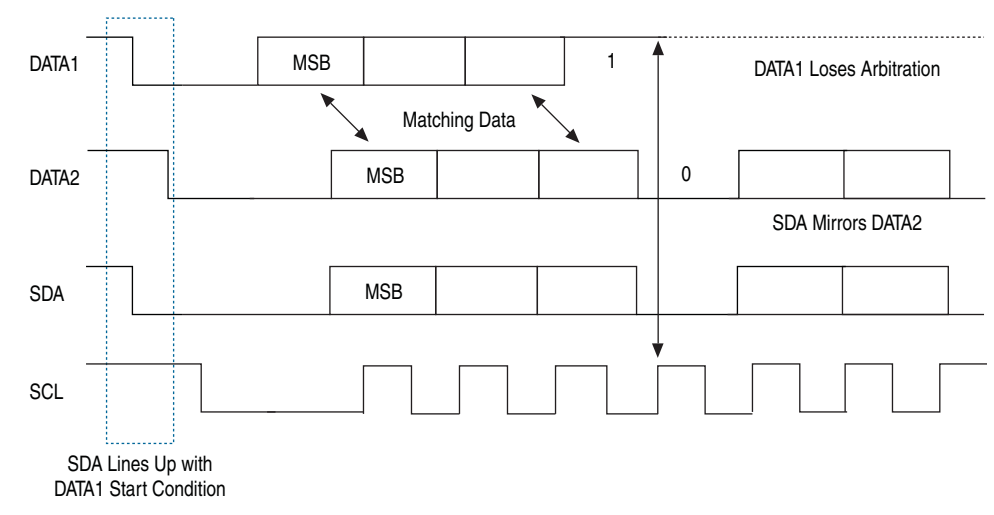
I²C 控制器总线协议支持多个主器件位于相同的总线上。如果相同的 I²C 总线上具有两个主器件，那么会有一个仲裁程序（如果这两个主器件通过同时生成一个 START 条件来试图同时控制总线）。一旦一个主器件（例如，一个微控制器）已经控制了总线，其它的主器件不可以控制总线直到第一个主器件发送 STOP 条件并且使总线处于闲置状态。

当 SCL 线是 1 时，仲裁在 SDA 线上发生。另一个主器件发送 0 时，发送 1 的主器件会失去仲裁并且关闭其数据输出阶段。失去仲裁的主器件会继续生成时钟直到字节传输结束。如果两个主器件对相同的从器件寻址，那么仲裁会进入数据阶段。

一旦检测到它对于另一个主器件已经失去仲裁，那么 I²C 控制器停止生成 SCL。

图 20-9 显示了两个主器件对总线仲裁时的时序。

图 20-9. 多个主器件仲裁



总线控制由地址或主器件代码决定并且通过主器件之间的竞争数据被发送，所以总线上没有中央主器件也没有任何优先权顺序。

仲裁在以下的条件中不被支持：

- RESTART 条件和一个数据位
- STOP 条件和一个数据位
- RESTART 条件和 STOP 条件

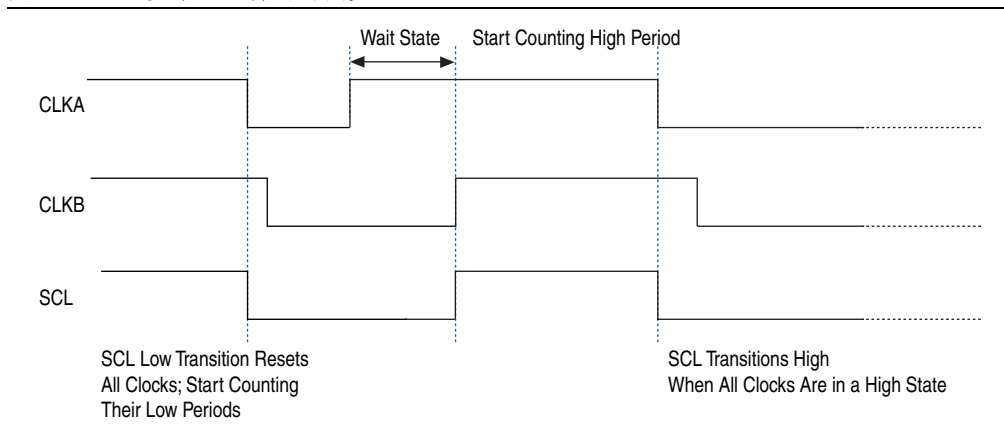
仲裁程序中不涉及从器件。

时钟同步

当两个或多个主器件试图同时在总线上传输信息时，它们必须仲裁并且同步 SCL 时钟。所有主器件生成本身的时钟来传输消息。只有在 SCL 时钟的高电平期间数据才有效。通过将 wired-AND 连接到 SCL 信号，时钟同步被执行。当主器件使 SCL 时钟跳变为 0 时，主器件开始计数 SCL 时钟的低时间并且在下一个时钟周期开始时使 SCL 时钟信号跳变为 1。然而，如果另一个主器件保持 SCL 线为 0，那么主器件进入 HIGH 等待状态直到 SCL 时钟线跳变为 1。

然后所有主器件计数其高时间，具有最短高时间的主器件使 SCL 线跳变为 0。然后主器件计数其低时间并且具有最长低时间的主器件强制另一个主器件进入 HIGH 等待状态。因此，如图 20-10 所示，一个同步的 SCL 时钟被生成。另外，从器件可以保持 SCL 线低电平以便减慢 I²C 总线的时序。

图 20-10. 多个主器件时钟同步



时钟频率配置

当将 I²C 控制器配置成主器件时，必须在所有 I²C 总线传输发生之前设置 SCL 计数寄存器以便确保正确的 I/O 时序。有 4 个 SCL 计数寄存器：

- 标准速度 I²C 时钟 SCL 高计数，IC_SS_SCL_HCNT
- 标准速度 I²C 时钟 SCL 低计数，IC_SS_SCL_LCNT
- 快速 I²C 时钟 SCL 高计数，IC_FS_SCL_HCNT
- 快速 I²C 时钟 SCL 低计数，IC_FS_SCL_LCNT



如果 I²C 控制器被使能而仅作为一个 I²C 从器件进行操作，那么没有必要编程所有 SCL 计数寄存器，因为这些寄存器仅用于决定作为一个 I²C 主器件进行操作的 SCL 时序要求。

最低高和低计数值

当 I²C 控制器在发送和接收传输中作为 I²C 主器件进行操作时，SCL 低计数寄存器中编程的最小值是 8 而 SCL 高计数寄存器中允许的最低值是 6。

低计数寄存器的最低值 8 是由于 I²C 控制器在 SCL 的负边沿之后驱动 SDA 所需时间所导致。高计数寄存器的最低值 6 是由于 I²C 控制器在 SCL 的高电平期间采样 SDA 所需时间所导致。

I²C 控制器添加 1 个周期到低计数寄存器值以便生成 SCL 时钟的低电平周期。

I²C 控制器添加 7 个周期到高计数寄存器值以便生成 SCL 时钟的高电平周期。这是由于以下因素导致：

- 应用到 SCL 线的数字滤波出现 4 个 14_sp_clk 周期的延迟。该滤波包含亚稳移除和 SDA 和 SCL 边沿的 2/3 的 majority vote process。
- 只要 I²C 控制器驱动 SCL 从 1 到 0——也就是，完成 SCL 高时间——那么会导致 3 个 14_sp_clk 周期的内部逻辑延迟。

结果，I²C 控制器能够接受的最小 SCL 低时间是 9 个 (9) l4_sp_clk 周期 (8+1)，而最小 SCL 高时间是 13 个 (13) l4_sp_clk 周期 (6+1+3+3)。

计算高和低计数值

以下的计算显示了如何对 I²C 控制器中的每个速度模式计算 SCL 高和低计数值的实例。设置正确 SCL 时钟高和低时间所需的正确 l4_sp_clk 时钟脉冲数的计算公式如下所示：

公式 20 - 1.

$$IC_HCNT = \text{ceil}(\text{MIN_SCL_HIGHTime} * \text{OSCFREQ})$$

$$IC_LCNT = \text{ceil}(\text{MIN_SCL_LOWtime} * \text{OSCFREQ})$$

$$\text{MIN_SCL_HIGHTime} = \text{minimum high period}$$

$$\text{MIN_SCL_HIGHTime} =$$

4000 ns for 100 kbps

600 ns for 400 kbps

60 ns for 3.4 Mbs, bus loading = 100pF

160 ns for 3.4 Mbs, bus loading = 400pF

$$\text{MIN_SCL_LOWtime} = \text{minimum low period}$$

$$\text{MIN_SCL_LOWtime} =$$

4700 ns for 100 kbps

1300 ns for 400 kbps

120 ns for 3.4Mbs, bus loading = 100pF

320 ns for 3.4Mbs, bus loading = 400pF

$$\text{OSCFREQ} = \text{l4_sp_clk clock frequency (Hz)}$$

例如：

$$\text{OSCFREQ} = 100 \text{ MHz}$$

$$\text{I2Cmode} = \text{fast, 400 kbps}$$

$$\text{MIN_SCL_HIGHTime} = 600 \text{ ns}$$

$$\text{MIN_SCL_LOWtime} = 1300 \text{ ns}$$

$$IC_HCNT = \text{ceil}(600 \text{ ns} * 100 \text{ MHz}) \quad IC_HCNTSCL \text{ PERIOD} = 60$$

$$IC_LCNT = \text{ceil}(1300 \text{ ns} * 100 \text{ MHz}) \quad IC_LCNTSCL \text{ PERIOD} = 130$$

$$\text{Actual MIN_SCL_HIGHTime} = 60 * (1/100 \text{ MHz}) = 600 \text{ ns}$$

$$\text{Actual MIN_SCL_LOWtime} = 130 * (1/100 \text{ MHz}) = 1300 \text{ ns} \dagger$$


SDA 保持时间

I²C 协议规范要求标准和快速模式中的 SDA 信号的保持时间为 300 ns。SCL 和 SDA 信号的电路板延迟意味着保持时间要求在 I²C 主器件满足，而不是在 I²C 从器件（或反之亦然）。因为每个应用遇到不同的电路板延迟，所以 I²C 控制器包含一个软件可编程寄存器 IC_SDA_HOLD，以便使能 SDA 保持时间的动态调整。

DMA 控制器接收


I²C 控制器支持 DMA 发出信号以指示数据什么时候可读或发送 FIFO 什么时候需要数据。该支持需要 2 个 DMA 通道，一个用于发送数据，另一个用于接收数据。I²C 控制器支持单一和突发 DMA 传输。系统软件可以通过将相应值编程到阈值寄存器来选择 DMA 突发模式。所建议的 FIFO 阈值寄存器值的设置为半满。

要使能 I²C 控制器的 DMA 控制器接口，必须写入 DMA 控制寄存器 (DMACR) 位。将 1 写入 DMACR 寄存器的 TDMAE 位域会使能 I²C 控制器发送握手接口。将 1 写入 DMACR 寄存器的 RDMAE 位域会使能 I²C 控制器接收握手接口。

 要了解关于 DMA 控制器的详细信息，请参考 *Cyclone V 器件手册* 第 3 卷的 [DMA Controller](#) 章节。


时钟

每个 I²C 控制器都被连接到 l4_sp_clk 时钟，这个时钟在标准和快速模式间切换。时钟输入由时钟管理器驱动。

 要了解更多信息，请参考 *Cyclone V 器件手册* 第 3 卷的 [Clock Manager](#) 章节。

复位

每个 I²C 控制器都有一个单独的复位信号。复位管理器在冷或热复位时驱动信号。

 要了解更多信息，请参考 *Cyclone V 器件手册* 第 3 卷的 [Reset Manager](#) 章节。

接口管脚

所有 I²C 控制器的实例通过管脚多路复用器连接到外部管脚。管脚复用支持所有实例同时和单独运行。管脚必须连接到上拉电阻并且 I²C 总线电容不可以超过 400 pF。

表 20-2 显示了 I²C 控制器接口的 I/O 管脚使用。

表 20-2. I²C 控制器接口管脚

管脚名称	信号宽度	方向	说明
SCL	1 bit	双向	串行时钟
SDA	1 bit	双向	串行数据

I²C 控制器编程模型

该部分介绍基于两个主器件和从器件操作模式的 I²C 控制器的编程模型。



每个 I²C 控制器应该被单独地设置为作为 I²C 主器件或 I²C 从器件进行操作，禁止同时设置为两者。要确保 IC_CON 寄存器的位 6 (IC_SLAVE_DISABLE) 和 0 (IC_MASTER_MODE) 不被分别地设置为 0 和 1。

从器件模式操作

该部分讨论从器件模式操作程序。

初始配置

要将 I²C 控制器作为从器件，请执行以下步骤：

1. 通过将 0 写入 IC_ENABLE 寄存器的位 0 来禁用 I²C 控制器。
2. 写入 IC_SAR 寄存器（位 9:0）来设置从器件地址。这是 I²C 控制器做出响应的地址。



I²C 控制器从器件地址的复位值是 0x55。如果将 0x55 用作从器件地址，那么可以安全地跳过该步骤。

3. 写入 IC_CON 寄存器以指定哪种类型的寻址被支持（通过设置位 3 的 7- 或 10-bit）。通过将 0 写入位 6 (IC_SLAVE_DISABLE) 以及将 0 写入位 0 (MASTER_MODE) 来使能仅作为从器件模式中的 I²C 控制器。



从器件和主器件不需要配置成相同类型的寻址 (7- 或 10-bit) 而被编程。例如，一个从器件可以使用 7-bit 寻址编程并且一个主器件使用 10-bit 寻址编程，反之亦然。

4. 通过将 1 写入 IC_ENABLE 寄存器的位 0 来使能 I²C 控制器。

单字节的从器件 - 发送器操作

当总线的另一个 I²C 主器件对 I²C 控制器寻址并且请求数据时，I²C 控制器作为一个从器件 - 发送器并且会执行以下步骤：

1. 另一个 I²C 主器件使用一个地址（匹配 I²C 控制器的 IC_SAR 寄存器中的从器件地址）发起 I²C 传输。
2. I²C 控制器确认发送地址并且识别传输的方向以便表示它作为从器件 - 发送器。
3. I²C 控制器置位 RD_REQ 中断 (IC_RAW_INTR_STAT 寄存器的位 5) 并且等待软件响应。


如果 IC_INTR_MASK 寄存器的位 5 (M_RD_REQ 位域) 被设置为 0 而导致 RD_REQ 中断被屏蔽，那么建议您指示 CPU 对 IC_RAW_INTR_STAT 寄存器执行定期读取。

- a. 表示 IC_RAW_INTR_STAT 寄存器 (R_RD_REQ 位域) 的位 5 被设置为 1 的读取必须作为等同于 RD_REQ 中断被置位。
- b. 然后软件必须执行以满足 I²C 传输。
- c. 所用的时间间隔应该大约为 I²C 控制器可以处理的最快 SCL 时钟周期的 10 倍。例如，对于 400 Kbps，时间间隔是 25 us。



10 在此被建议是因为这是单字节的数据在 I²C 总线上传输所需的大概时间。

4. 如果接收读请求之前，TX FIFO 中有剩余的数据，那么 I²C 控制器置位一个 TX_ABORT 中断 (IC_RAW_INTR_STAT 寄存器的位 6) 以便从 TX FIFO 中刷新旧数据。

 因为只要 TX_ABORT 事件出现，I²C 控制器的 TX FIFO 就会被强制进入刷新/复位状态，所以试图写入 TX FIFO 之前，软件有必要通过读取 IC_CLR_TX_ABORT 寄存器将 I²C 控制器从该状态释放。要了解更多信息，请参考寄存器映射中的 C_RAW_INTR_STAT 寄存器说明。

如果 IC_INTR_MASK[6] 寄存器的位 6 (M_TX_ABORT 位域) 被设置为 0 而导致 TX_ABORT 中断被屏蔽，那么建议 CPU 对 IC_RAW_INTR_STAT 寄存器执行定期读取。

- a. 表示位 6 (R_TX_ABORT) 被设置为 1 的读取必须作为等同于 TX_ABORT 中断被置位。
 - b. 软件不需要进一步进行操作。
 - c. 所用时序间隔应该和之前的步骤中对 IC_RAW_INTR_STAT[5] 寄存器的描述相似。
5. 软件使用要写入的数据写入 IC_DATA_CMD 寄存器的 DAT 位并且将 0 写入位 8。
 6. 软件在继续操作之前必须清除 IC_RAW_INTR_STAT 寄存器的 RD_REQ 和 TX_ABORT 中断（分别为位 5 和位 6）。


当 R_RD_REQ 或 R_TX_ABORT 位读取为 1 时，如果 RD_REQ 和 / 或 TX_ABORT 中断被屏蔽，那么 IC_RAW_INTR_STAT 寄存器的清除应该已经被执行。

7. I²C 控制器发送字节。
8. 主器件可能通过发出 RESTART 条件保持 I²C 总线或通过发出 STOP 条件释放总线。

单字节的从器件 - 接收器操作

当总线的另一个 I²C 主器件对 I²C 控制器寻址并且发送数据时，I²C 控制器作为一个从器件 - 接收器并且会执行以下步骤：

1. 另一个 I²C 主器件使用一个地址（匹配 IC_SAR 寄存器中的 I²C 控制器的从地址）发起 I²C 传输。
2. I²C 控制器确认发送的地址并且识别传输的方向以表明 I²C 控制器作为一个从器件 - 接收器。
3. I²C 控制器接收到发送的字节并且将其放入接收缓冲器。

 当字节被发送时如果 RX FIFO 已满，那么 FIFO 溢出并且 I²C 控制器继续接下来的 I²C 传输。因为 NACK 没有被生成，所以软件必须在 I²C 控制器指示时（通过 IC_INTR_STAT 寄存器中的 R_RX_OVER 位）识别 FIFO 溢出并且采取相应的措施以恢复丢失的数据。因而，软件有一个实时约束以在后面的上溢出现之前执行 RX FIFO，因为不可以将压力重新应用到远程发送主器件。

4. I²C 控制器置位 RX_FULL 中断 (IC_RAW_INTR_STAT[2] 寄存器)。

如果设置 IC_INTR_MASK[2] 寄存器为 0 或设置 IC_TX_TL 为一个大于 0 的值而导致 RX_FULL 中断被屏蔽，那么建议 CPU 进行 IC_STATUS 寄存器的周期性读取。IC_STATUS 寄存器的位 3 (RFNE) 设置为 1 的读取，必须由软件作为等同于 RX_FULL 中断被置位。

5. 软件会从 IC_DATA_CMD 寄存器（位 7:0）读取字节。
6. 另一个主器件可能通过发出 RESTART 条件保持 I²C 总线或通过发出 STOP 条件释放总线。

Bulk 传输的从器件 – 传输操作

标准 I²C 协议中，所有传输都是单字节传输并且编程器通过将一个字节写入从器件的 TX FIFO 响应远程主器件读请求。当远程主器件（主器件 – 接收器）对从器件（从器件 – 发送器）发起读请求（RD_REQ）时，应该至少有一个读请求被放入从器件 – 发送器的 TX FIFO。I²C 控制器的设计是为了处理 TX FIFO 中的更多数据，以便接下来的读请求可以接收到该数据而无需产生一个中断来请求更多数据。最终，这会移除每次对于数据产生中断时严重延迟发生的可能性（如果存在只有一个读请求放入 TX FIFO 的限制时）。

只有当 I²C 控制器作为从器件 – 发送器时，该模式才出现。如果远程主器件确认从器件 – 发送器发送的数据并且从器件的 TX FIFO 中没有数据，那么 I²C 控制器产生读请求中断（RD_REQ）并且在数据被发送到远程主器件之前等待数据写入 TX FIFO。

如果 IC_INTR_STAT 寄存器的位 5 (M_RD_REQ) 设置为 0 导致 RD_REQ 中断被屏蔽，那么建议 CPU 进行 IC_RAW_INTR_STAT 寄存器的周期性读取。读取 IC_RAW_INTR_STAT 而得到位 5 (R_RD_REQ) 设置为 1 必须作为等同于该部分中所指的 RD_REQ 中断。

读请求一发出，RD_REQ 中断就会产生，并且如同其它中断一样，当退出处理中断服务例程（ISR）时必须被清除。ISR 使您能够写入 1 个字节或多于 1 个字节到 TX FIFO。发送这些字节到主器件期间，如果主器件确认最后字节，那么从器件必须再次产生 RD_REQ，因为主器件正在请求更多数据。

如果编程器提前获知远程主器件请求一个数据包的 n 个字节，那么当另一个主器件对 I²C 控制器寻址并且请求数据时，可以使用 n 个数的字节写入 TX FIFO 并且远程主器件将其作为连续的数据流而接收。例如，只要远程主器件正在确认发送的数据并且 TX FIFO 中具有可用数据，那么 I²C 控制器从器件继续将数据发送到远程主器件。无需再次发出 RD_REQ。

如果远程主器件要从 I²C 控制器接收 n 个字节但是编程器将大于 n 的字节数写入 TX FIFO，那么当从器件完成发送所需的 n 个字节时，它会清除 TX FIFO 并且忽略多余字节。

I²C 控制器生成一个发送中断（TX_ABORT）事件以指示该实例中 TX FIFO 的清除。期待 ACK/NACK 时，如果接收到 NACK，那么远程主器件会有它想要的所有数据。此时，从器件的状态机内生成一个标记以清除 TX FIFO 中的剩余数据。该标记被传输到 FIFO 存在的处理器总线时钟域并且 TX FIFO 的内容在此时被清除。

主器件模式操作

该部分讨论主器件模式操作程序。

初始配置

对于主器件模式操作，目标地址和地址格式可以被动态地更改而不必禁用 I²C 控制器。该功能只有当 I²C 控制器作为主器件时才应用，因为对地址进行任何更改之前，从器件要求组件被禁用。要将 I²C 控制器用作主器件，请执行以下步骤：

1. 通过将 0 写入 IC_ENABLE 寄存器而禁用 I²C 控制器。
2. 写入 IC_CON 寄存器以便设置从器件操作（位 2:1）的最大速度模式并指定当器件是从器件时（位 3），I²C 控制器是否在 7/10 位寻址模式中开始传输。
3. 将 I²C 器件被寻址的地址写入 IC_TAR 寄存器。它也会表明 General Call 或 START BYTE 命令是否将会由 I²C 执行。I²C 控制器主器件发起的传输的所需速度，7-bit 或 10-bit 寻址，由 IC_10BITADDR_MASTER 位域（位 12）控制。
4. 通过将 1 写入 IC_ENABLE 寄存器使能 I²C 控制器。

5. 现在将要发送的传输方向和数据写入 IC_DATA_CMD 寄存器。如果 IC_DATA_CMD 寄存器在使能 I²C 控制器之前被写入，那么使能 I²C 控制器时，由于缓冲器被清除，数据和命令丢失。



对于多个 I²C 传输，执行额外的写入到 TX FIFO 以便 TX FIFO 在 I²C 传输期间不会变空。如果 TX FIFO 在所有阶段完全为空，那么进一步写入到 TX FIFO 会导致单独的 I²C 传输。

动态 IC_TAR 或 IC_10BITADDR_MASTER 更新

I²C 控制器支持 IC_TAR 寄存器的 IC_TAR (位 9:0) 和 IC_10BITADDR_MASTER (位 12) 位域的动态更新。只要以下条件满足，您可以动态地写入 IC_TAR 寄存器：

- I²C 控制器不被使能 (IC_ENABLE=0)；
- I²C 控制器被使能 (IC_ENABLE=1)；以及 I²C 控制器没有参与任何的主器件 (TX、RX) 操作 (IC_STATUS[5]=0)；以及 I²C 控制器被使能以在主器件模式中操作 (IC_CON[0]=1)；以及 TX FIFO 中没有入口 (IC_STATUS[2]=1)。

主器件发送和主器件接收

I²C 控制器支持动态地读和写之间的来回切换。要发送数据，将要写的数据写入 I²C Rx/Tx 数据缓冲器和命令寄存器 (IC_DATA_CMD) 的较低字节。对于 I²C 写操作，CMD 位 [8] 应该被置 0。随后，读命令可能通过将 “don’t cares” 写入 IC_DATA_CMD 寄存器的较低字节而被发出，并且 CMD 位应该置 1。只要发送 FIFO 中存在命令，主器件模式中的 I²C 控制器就会继续发起传输。如果发送 FIFO 变空，那么 I²C 控制器完成当前传输后插入 STOP 条件。

禁用 I²C 控制器

通过 IC_ENABLE_STATUS 寄存器，软件可以决定当 IC_ENABLE 从 1 设置为 0 后，什么时候完全关闭硬件。

1. 定义计时器间隔 (ti2c_poll) 等同于最高 I²C 传输速度 (系统中使用和 I²C 控制器支持) 的信号周期的 10 倍。例如，如果最高 I²C 传输模式是 400 Kbps，那么 ti2c_poll 是 25 us。
2. 定义一个最高超时参数 MAX_T_POLL_COUNT，以便如果任何重复的轮询操作超出该最高值，那么错误被报告。
3. 执行一个阻塞线程 / 进程 / 函数以防止软件开始进一步的 I²C 主器件传输，但是允许完成所有待定传输。



如果 I²C 控制器被编程为仅作为 I²C 从器件操作，那么该步骤可以被忽略。

4. 变量 POLL_COUNT 被初始化为零。
5. 将 IC_ENABLE 设置为 0。
6. 读取 IC_ENABLE_STATUS 寄存器并且测试 IC_EN 位 (位 0)。对 POLL_COUNT 增加 1。如果 POLL_COUNT >= MAX_T_POLL_COUNT，会出现相关的错误代码。
7. 如果 IC_ENABLE_STATUS[0] 为 1，然后休眠 ti2c_poll 并且继续之前的操作步骤。否则，会出现一个相关成功代码。

DMA 控制器操作

要使能 I²C 控制器的 DMA 控制器接口，必须写入 DMA 控制寄存器 (IC_DMA_CR)。将 1 写入 IC_DMA_CR 寄存器的 TDMAE 位域会使能 I²C 控制器发送握手接口。将 1 写入 IC_DMA_CR 寄存器的 RDMAE 位域会使能 I²C 控制器接收握手接口。



要了解关于 DMA 控制器的详细信息，请参考 *Cyclone V 器件手册* 第 3 卷的 *DMA Controller* 章节。

I²C 控制器中 RX 和 TX 缓冲器的 FIFO 缓冲器深度 (FIFO_DEPTH) 是 64 个入口。

发送 FIFO 下溢

I²C 串行传输期间，只要发送 FIFO 中的入口数小于或等于 DMA 发送数据水平寄存器 (IC_DMA_TDLR) 中的值 (也称为水位值)，就会对 DMA 控制器进行发送 FIFO 请求。DMA 控制器通过将突发数据写入发送 FIFO 缓冲器而进行响应，突发长度被指定为 DMA 突发长度。



要了解关于 DMA 突发长度微代码设置的详细信息，请参考 *Cyclone V 器件手册* 第 3 卷的 *DMA Controller* 章节。

应该从 DMA 经常获取数据，以便发送 FIFO 缓冲器可以连续地执行串行传输，也就是，当 FIFO 缓冲器开始为空时，另一个 DMA 请求应该被触发。否则，FIFO 会运行完数据 (下溢)，从而导致 STOP 被插入到 I²C 总线。要防止该情况发生，必须正确地设置水位值。

发送水位值

请考虑以下假设的实例：

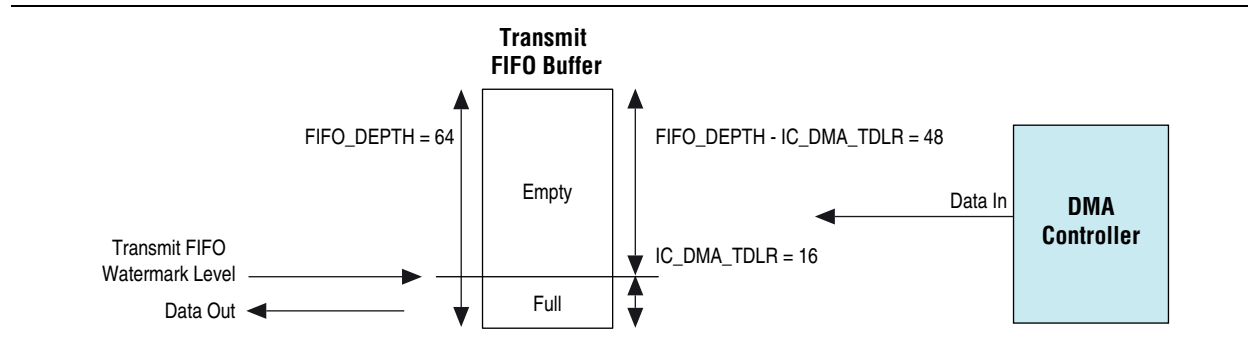
$$\text{DMA 突发长度} = \text{FIFO_DEPTH} - \text{IC_DMA_TDLR}$$

此实例中，DMA 突发中传输的数据项数等于发送 FIFO 缓冲器中的空区域。请考虑下列两个不同的水位值设置：

- 用例 1: IC_DMA_TDLR = 16:
 - 发送 FIFO 水位值 = IC_DMA_TDLR = 16
 - DMA 突发长度 = FIFO_DEPTH - IC_DMA_TDLR = 48
 - I²C 发送 FIFO_DEPTH = 64
 - 模块传输容量 = 240

图 20 - 11 显示了水位值等于 16 时的发送 FIFO。

图 20 - 11. 发送 FIFO 水位值 = 16



所需的突发传输数等于模块容量除以每突发的数据项数：

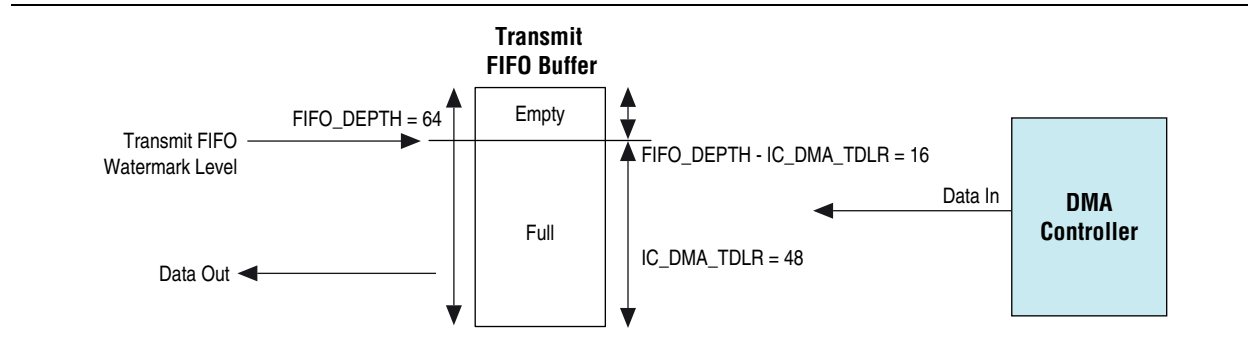
模块传输容量 /DMA 突发长度 = 240/48 = 5

DMA 模块传输中的突发传输数是 5。但是水位值 (IC_DMA_TDLR) 很低。因此，I²C 串行发送线需要发送数据的发送下溢的可能性很高，但是在发送 FIFO 缓冲器中不会留下数据。这种情况发生是因为 DMA 在 FIFO 缓冲器变空之前没有时间执行 DMA 请求。

- 用例 2: IC_DMA_TDLR = 48
 - 发送 FIFO 水位值 = IC_DMA_TDLR = 48
 - DMA 突发长度 = FIFO_DEPTH - IC_DMA_TDLR = 16
 - I²C 发送 FIFO_DEPTH = 64
 - 模块传输容量 = 240

图 20 - 12 显示了水位值等于 48 时的发送 FIFO。

图 20 - 12. 发送 FIFO 水位值 = 48



模块的突发传输数：

模块传输容量 /DMA 突发长度 = 240/16 = 15

对于该模块传输中，在 DMA 模块传输中具有 15 个目的突发传输。但是水位值，IC_DMA_TDLR 很高。因此，I²C 发送下溢的可能性很低，因为 I²C 发送 FIFO 变空之前，DMA 控制器有足够的时间执行目的突发传输请求。

因此，通过每模块的较多突发传输，第 2 个用例具有较低的下溢可能性。跟第 1 个用例相比，这提供了一个潜在地每模块的较高量突发和较差总线利用率。

因此，选择水准值的目标是最小化每模块的传输数，同时保持下溢情况的可能性到可接受的水平。在实践中，这是一个速率比函数，在这个速率比上，I²C 数据发送为 DMA 可以响应的目的突发请求速率。

发送 FIFO 上溢

当发送 FIFO 中没有足够的空间执行目的突发请求时，设置 DMA 突发长度为一个大于水准值（触发 DMA 请求）的值会导致上溢。因此，必须遵循以下的公式来避免上溢：

$$\text{DMA 突发长度} \leq \text{FIFO_DEPTH} - \text{IC_DMA_TDLR}$$

用例 2 中：IC_DMA_TDLR = 48，突发请求进行时的发送 FIFO 中的空间量等于 DMA 突发长度。因此，突发传输完成时，发送 FIFO 缓冲器可能为满，但是不上溢。

因此，要实现最佳操作，DMA 突发长度应设置在触发一个发送 DMA 请求的 FIFO 缓冲器水平；也就是：

$$\text{DMA 突发长度} = \text{FIFO_DEPTH} - \text{IC_DMA_TDLR}$$

遵循这一公式会减少模块传输所需的 DMA 突发数，反而会提高总线利用率。



传输期间，如果 I²C 控制器在 I²C 串行发送线上已经成功地发送了一个或多个数据项，那么发送 FIFO 缓冲器在 DMA 突发传输结束时不会满。

接收 FIFO 上溢

I²C 串行传输期间，无论接收 FIFO 中的入口数在 DMA 接收数据水平寄存器还是在其之上 (IC_DMA_RDLR + 1)，都会对 DMA 进行接收 FIFO 请求。这被称为水准值。DMA 通过从接收 FIFO 缓冲器获取一个突发的数据做出响应。

数据应该由 DMA 经常获取，以便接收 FIFO 缓冲器连续地接受串行传输，也就是，当 FIFO 开始填充时，另一个 DMA 传输被请求。否则，FIFO 将会被填充数据（上溢）。要防止这一情况，用户必须正确地设置水准值。

接收水准值

与之前介绍的选择发送水准值相似，接收水准值，IC_DMA_RDLR + 1，应被设置以最小化上溢的可能性，如图 20-13 所示。它是每模块所需的 DMA 突发传输数和上溢发生的可能性之间的权衡。

接收 FIFO 下溢

将源传输突发长度设置为大于水印层会导致下溢，其中没有足够数据执行源突发请求。因此，必须遵循以下公式来避免下溢：

$$\text{DMA 突发长度} = \text{IC_DMA_RDLR} + 1$$

进行了突发请求时，如果接收 FIFO 缓冲器中的数据项数等于源突发长度，那么突发传输完成时，接收 FIFO 可能为空，但是不下溢。为了实现最佳操作，DMA 突发长度应该设置在水准值，IC_DMA_RDLR + 1。

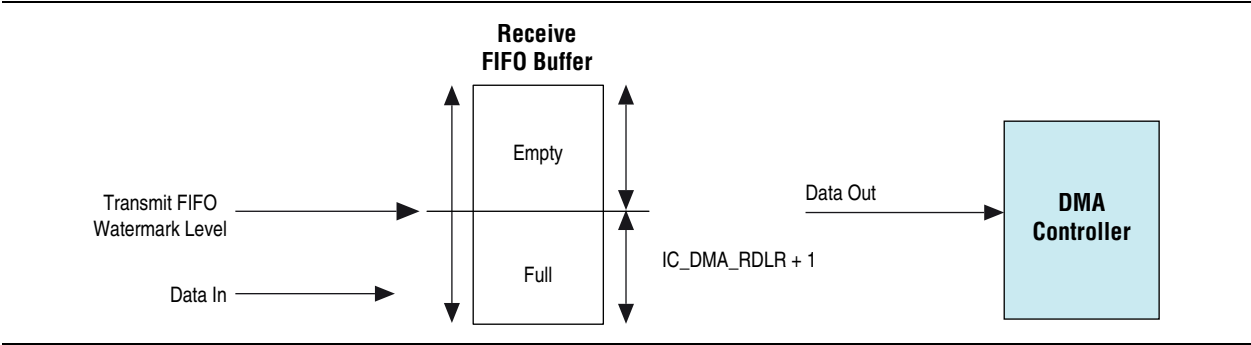
遵循该公式会减少模块传输中 DMA 突发数，从而可以避免下溢和提高总线利用率。




突发期间，如果 I²C 控制器在 I²C 串行接收线上已经成功地接收了一个或多个数据项，那么源突发传输结束时接收 FIFO 不会为空。

图 20 - 13 显示了接收 FIFO 缓冲器。

图 20 - 13. 接收 FIFO 缓冲器




I²C 控制器地址映射和寄存器定义

 地址映射和寄存器定义位于该手册卷附带的 [hps.html](#) 文件中。点击链接以打开文件。

要查看模块说明和基地址，找到并且点击以下任何模块实例的链接：

- [i2c0](#)
- [i2c1](#)
- [i2c2](#)
- [i2c3](#)

然后要查看寄存器和域说明，找到并且点击寄存器名称。寄存器地址是相对于每个模块实例的基地址的偏移。

 所有模块的基地址也在 *Cyclone V 器件手册* 第 3 卷的 *Introduction to the Hard Processor System* 章节中列出。

文档修订历史

表 20 - 3 显示了该文档的修订历史。

表 20 - 3. 文档修订历史

日期	版本	修订内容
2012 年 11 月	1.2	少量文本编辑。
2012 年 5 月	1.1	添加了编程模型、地址映射和寄存器定义、时钟、复位和接口管脚部分。
2012 年 1 月	1.0	首次发布。

