

本章节介绍了硬核处理器系统 (HPS) 中的直接存储器访问控制器 (DMAC)。DMA 控制器用于传递系统中的存储器与外设以及其它存储器位置之间的数据。DMA 控制器是 ARM® CoreLink™ DMA Controller (DMA-330) 的一个实例。

要了解关于 ARM 的 DMA-330 控制器的详细信息，请参考 ARM 网站 (infocenter.arm.com) 上的 *CoreLink DMA Controller DMA-330 Revision: r1p1 Technical Reference Manual*。

DMA 控制器的特性

HPS 提供一个 DMAC，用于处理存储器映射的外设与存储器之间的数据传递，从而不再需要微处理器 (MPU) 子系统来处理。DMAC 支持存储器到存储器，存储器到外设和外设到存储器的传递。DMAC 支持高达 8 个逻辑通道，用于各种等级的操作要求，并对外设硬件流程控制提供高达 31 个外设握手接口。

DMA 控制器包含一个指令处理模块，使 DMA 控制器能够处理用于控制 DMA 传递的程序代码。DMA 控制器也包含一个 ARM 高级微控制器总线体系结构 (AMBA®) 高级可扩展接口 (AXI™) 主接口单元，从系统存储器中提取程序代码并存储在它的指令高速缓存中。AXI 主接口也用于执行 DMA 数据传递。DMA 指令执行引擎执行其指令高速缓存中的程序代码，并通过读写 AXI 指令的各自指令序列来规划读或写 AXI 指令。它也包含一个 multi-FIFO (MFIFO) 数据传递，用于 DMA 传递过程中存储读取或写入的数据。

DMAC 提供 11 个中断输出来使能与 MPU 子系统的高效通信。外设请求接口支持具有 DMA 能力的外设连接，在没有处理器的干预下能够实现存储器到外设和外设到存储器的传递。由于 HPS 支持某些不符合 ARM DMA 外设接口协议的外设，因此通过添加适配器能够使这些外设与 DMAC 一起运行。下面的外设接口协议被支持：

- Synopsys 协议
 - 串行外设接口 (SPI)
 - 通用异步接收器 / 发送器 (UART)
 - 跨集成电路 (I²C)
 - FPGA
- ARM 协议
 - Quad SPI flash 控制器
 - 系统跟踪宏单元 (System Trace Microcell (STM™))
- Bosch 协议
 - 控制器区域网络 (CAN)

双从接口使 DMA 控制器的操作能够置于两种状态: Secure 状态和 Non-secure 状态。必须配置网络互联以确保只有安全传输能够访问安全接口。从接口能够访问状态寄存器,也能够直接执行 DMA 控制器中的指令。

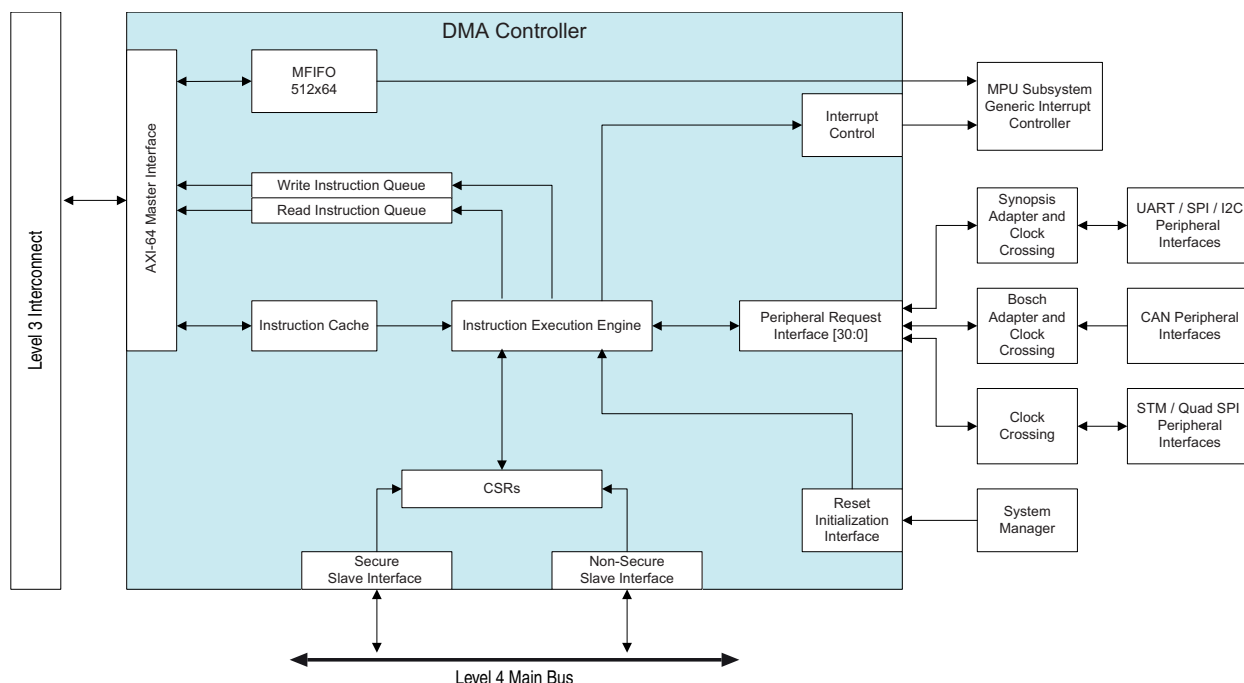
DMAC 具有下面特性:

- 一个小指令组, 提供一个灵活方法来指定 DMA 操作。与基于链表项 (LLI) 的 DMA 控制器的固定功能相比, 该体系结构具有更高的灵活性。
- 支持多种传输类型
 - 存储器到存储器
 - 存储器到外设
 - 外设到存储器
 - 分散收集 (scatter-gather)
- 支持高达 8 个 DMA 通道
- 支持高达 8 个未执行的 AXI 读和写传输
- 使软件能够规划高达 16 个未执行的读指令和写指令
- 支持 MPU 子系统内的 11 个中断行
 - 1 个用于 DMA 线程终止
 - 8 个用于事件
 - 2 个用于 MFIFO 缓存 ECC
- 支持 31 个外设请求接口
 - 4 个用于 FPGA
 - 4 个共享的用于 FPGA 或 CAN
 - 8 个用于 I²C
 - 8 个用于 SPI
 - 2 个用于 quad SPI
 - 1 个用于 System Trace Macrocell
 - 4 个用于 UART

DMA 控制器结构图和系统集成

图 16-1 显示 DMAC 的结构图以及如何与 HPS 系统的其余部分集成在一起。

图 16-1. DMA 控制器连接



DMA 控制器由 `l4_main_clk` 时钟驱动，用于控制器逻辑以及所有接口。DMA 控制器通过它的 64-bit AXI 主接口访问 level 3 (L3) 主开关。

DMA 控制器提供下面的从接口：

- 非安全从接口 (non-secure slave interface)
- 安全从接口 (secure slave interface)

您可以使用这些从接口访问用于控制 DMA 控制器功能的寄存器。DMA 控制器使用一个在 Secure 状态下运行的接口和一个在 Non-secure 状态下运行的接口来实现 TrustZone® 安全技术。

DMA 控制器的功能描述

本小节介绍了 DMAC 的主要接口和组件以及如何运行。

概述

DMAC 包含一个指令处理模块，此模块使 DMAC 能够处理用于 DMA 传输的程序代码。程序代码存储在系统存储器的一块区域中，DMAC 使用它的 AXI 主接口对该区域进行访问。DMAC 临时将指令存储在内部高速缓存中。



在图 16 - 2 中, DMAC 允许:

- 只有 DMA 通道线程能够使用显示为灰色的状态
- 无字母指定符的弧线表明 DMA 管理器与 DMA 通道线程的状态转换, 否则限制如下:
 - C— 仅 DMA 通道线程。
 - M— 仅 DMA 管理器线程。
- 虚线中的状态能够转变到 Faulting completing, Faulting 或者 Killing。

DMAC 从复位状态退出后, 将所有的 DMA 通道线程设置成 Stopped 状态, 并且 DMA 管理器线程转变到 Stopped 状态。

下面部分对各种状态进行了描述:

- “Stopped” (第 16 - 5 页)
- “Executing” (第 16 - 5 页)
- “Cache Miss” (第 16 - 6 页)
- “Updating PC” (第 16 - 6 页)
- “Waiting For Event” (第 16 - 6 页)
- “At Barrier” (第 16 - 6 页)
- “Waiting For Peripheral” (第 16 - 6 页)
- “Faulting Completing” (第 16 - 6 页)
- “Faulting” (第 16 - 6 页)
- “Killing” (第 16 - 7 页)
- “Completing” (第 16 - 7 页)

Stopped

线程有一个无效的 PC, 没有获取指令。根据不同的线程类型, 可以通过下面的操作将线程转换到 Executing 状态:

- DMA 管理器线程 — 通过从接口发出 DMAGO 指令。
- DMA 通道线程 — 编程 DMA 管理器线程, 对 Stopped 状态下的 DMA 通道线程执行 DMAGO。

Executing

线程有一个有效的 PC, 因此当 DMAC 仲裁时包括此线程。线程然后可以在下面情况下转换到下面其中一个状态:

- Stopped — 当 DMA 管理器线程执行 DMAEND 时。
- Cache miss — 当指令高速缓存不包含用于 DMA 管理器线程或者 DMA 通道线程的下一个指令时。
- Updating PC — 当 DMAC 计算高速缓存中下一个访问的地址时。
- Waiting for event — 当线程执行 DMAWFE 时。

- At barrier— 当 DMA 通道线程:
 - 执行 DMARMB, DMAWMB 或 DMAFLUSHP
 - 更新那些影响对齐的控制寄存器, 请参考第 16 - 22 页 “DMA 周期中更新 DMA 通道控制寄存器”。
 - Waiting for peripheral— 当 DMA 通道线程执行 DMAWFP 时。
 - Killing— 当 DMA 通道线程执行 DMAKILL 时。
 - Faulting completing— 对于 DMA 通道线程, 当:
 - 线程执行一个未定义或无效的指令时
 - 在指令提取或数据传输期间出现 AXI 错误
 - Faulting— 对于 DMA 管理器线程, 当:
 - 线程执行一个未定义或无效的指令时
 - 在指令提取期间出现 AXI 错误
- 对于 DMA 通道线程, 当出现看门狗超时终止时。
- Completing— 当 DMA 通道线程执行 DMAEND 时。

Cache Miss

线程被终止, DMAC 执行高速缓存线填充。DMAC 完成高速缓存线填充后, 线程返回到 Executing 状态。

Updating PC

DMAC 计算高速缓存中下一个访问的地址。计算出 PC 后, 线程返回到 Executing 状态。

Waiting For Event

线程被终止, 等待 DMAC 使用相应的事件编码执行 DMASEV。相应的事件出现后, 线程返回到 Executing 状态。

At Barrier

DMA 通道线程被终止, DMAC 等待 AXI 上的传输完成。AXI 传输完成后, 线程返回到 Executing 状态。

Waiting For Peripheral

DMA 通道线程被终止, DMAC 等待外设提供所请求的数据。外设提供数据后, 线程返回到 Executing 状态。

Faulting Completing

DMA 通道线程等待 AXI 主接口发送信号表明未完成的加载或存储传输已经完成。传输完成后, 线程转换到 Faulting 状态。

Faulting

线程被无限期终止。当您使用 DBGCMD 寄存器指示 DMAC 对线程执行 DMAKILL 时, 线程转换到 Stopped 状态。

Killing

DMA 通道线程等待 AXI 主接口发送信号表明未完成的加载或存储传输已经完成。传输完成后, 线程转换到 Stopped 状态。

Completing

DMA 通道线程等待 AXI 主接口发送信号表明未完成的加载或存储传输已经完成。传输完成后, 线程转换到 Stopped 状态。

初始化 DMAC

DMAC 提供几个存储器映射控制信号, 当 DMAC 从复位状态退出时初始化其操作状态。DMAC 被配置, 以便在它从复位状态退出时不会自动开始执行代码。三个存储器映射存储器信号由系统管理器控制。



关于更多信息, 请参考 *Cyclone® V 器件手册* 卷 3 中的 *System Manager* 章节。

如何设置 DMA 管理器的安全状态

boot_manager_ns 信号是设置 DMA 管理器的安全状态的唯一方法。

当 DMAC 从复位状态退出时, 它读取 boot_manager_ns 信号的状态并设置 DMA 管理器的安全。



安全状态设置后会保持不变, 直到 dma_rst_n 信号上的一个状态跳变复位 DMAC。

请参考第 16 - 17 页 “Secure 状态的 DMA 管理器线程” 和第 16 - 17 页 “Non-Secure 状态的 DMA 管理器线程” 来了解 DMA 管理器的安全状态如何影响 DMAC 如何操作。

如何设置中断输出的安全状态

DMAC 提供 boot_irq_ns[7:0] 信号, 使您能够分配给每个 irq[x] 信号一个安全状态。

boot_irq_ns[7:0] 信号连接到系统管理器。DMA 退出复位状态前, 您应该通过系统管理器编程 boot_irq_ns[7:0] 来控制哪些中断比特是安全的。



DMAC 从复位状态退出后立即采样这些比特, 然后忽略它们, 直到下一个复位。

每个 irq[x] 的安全状态设置后会保持不变, 直到 dma_rst_n 信号上的一个状态跳变复位 DMAC。

请参考第 16 - 17 页 “安全使用” 来了解 irq[x] 信号的安全状态如何影响 DMAC 如何执行 DMAWFE 和 DMASEV 指令。

如何对外设请求接口设置安全状态

DMAC 提供 boot_periph_ns[31:0] 信号, 使您能够分配给每个外设请求接口一个安全状态。

boot_periph_ns[31:0] 信号连接到系统管理器。DMA 退出复位状态前, 您应该通过系统管理器编程 boot_periph_ns[31:0] 来控制哪些外设接口是安全的。



DMAC 从复位状态退出后立即采样这些比特, 然后将它们忽略, 直到下一个复位。每个外设请求接口的安全状态设置后会保持不变, 直到 dma_rst_n 信号上的一个状态跳变复位 DMAC。

请参考第 16 - 17 页 “安全使用” 来了解外围请求接口的安全状态如何影响 DMA 通道线程如何执行 DMAWFP, DMALDP, DMASTP 或 DMAFLUSHP 指令。

使用从接口

从接口连接 DMAC 与 level 4 (L4) 主总线, 使微处理器能够访问寄存器。通过使用这些寄存器, 一个微处理器能够:

- 访问 DMA 管理器线程的状态
- 访问 DMA 通道线程的状态
- 使能或清除中断
- 使能事件
- 通过编程下面的调试寄存器, 发出一个 DMAC 执行的指令:
 - DBGCMD 寄存器
 - DBGINST0 寄存器
 - DBGINST1 寄存器

使用从接口发出指令到 DMAC

当 DMAC 运行时, 您只能发出指令的以下限制子集:

- DMAGO— 使用指定的 DMA 通道开始一个 DMA 传输。
- DMASEV— 使用指定的事件编号发出一个事件或中断的并发的信号。
- DMAKILL— 终止一个线程。

你要根据 boot_manager_ns 信号初始化 DMAC 运行的安全状态, 确保使用正确的从接口。例如, 如果 DMAC 处于 Secure 状态, 那么您必须使用安全从接口发出指令, 否则 DMAC 会忽略该指令。当 DMAC 处于 Non-secure 状态时, 您可以使用安全或非安全的从接口来开始或重新开始 DMA 通道。



在您能够使用调试指令寄存器或者 DBGCMD 寄存器发出指令前, 您必须读取 DBGSTATUS 寄存器来确保调试是空闲的, 否则 DMAC 会忽略这些指令。

当 DMAC 接收到来自从接口的指令时, 在它能够处理该指令前需要几个时钟周期, 例如, 如果流水线正在处理另一个指令。



发出 DMAGO 前, 您必须确保系统存储器包含一个合适的程序, 使 DMAC 能够运行, 开始于 DMAGO 指定的地址。

使用 DMAGO 和调试指令寄存器

此实例显示了使用调试指令寄存器开始 DMA 通道线程的必要步骤。

1. 对 DMA 通道创建一个程序。
2. 将此程序存储在系统存储器中一块区域中。

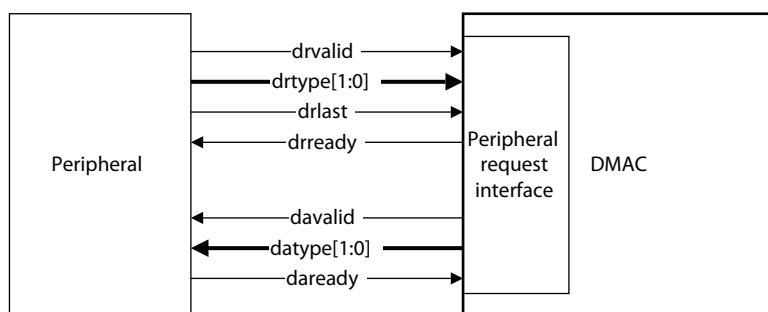
3. 使用 DMAC 上的一个从接口来编程 DMAGO 指令:
 - a. 轮询 DBGSTATUS 寄存器来确保调试是空闲的, 也就是 dbgstatus 比特为 0。
 - b. 写入 DBGINST0 寄存器, 并输入:
 - DMAGO 的指令类型 0 编码。
 - DMAGO 的指令类型 1 编码。
 - 调试线程比特为 0, 以选择 DMA 管理器线程。
 - c. 使用 DMAGO 指令类型 [5:2] 数据写入 DBGINST1 寄存器。您必须将这四个字节设置成程序中第一个指令的地址, 该指令是在步骤 b 中写入到系统存储器中的。
4. 通过写入 0 到 DBGCMD 寄存器, 指导 DMAC 执行调试指令寄存器中包含的指令。DMAC 开始 DMA 通道线程并将 dbgstatus 比特设成 1。DMAC 完成执行指令后对 dbgstatus 比特清零。

外设请求接口

图 16-3 显示了外设请求接口包含一个外围请求总线和一个 DMAC 接收确认总线, 这两条总线分别使用下面的前缀:

- dr— 外设请求总线
- da—DMAC 接收确认总线

图 16-3. 外设请求接口上的请求和接收确认总线



外设的请求总线上发出下面信号:

- 请求单一传递
- 请求突发传递
- 确认接收刷新 (flush) 请求

当外设发出 DMA 传递序列的最后一个请求时, 发送信号到 DMAC。

DMAC 能够在接收确认总线上发出下面信号:

- 当它完成请求的单一传递时发出信号
- 当它完成请求的突发传递时发出信号
- 发出刷新 (flush) 请求

DMAC 使您能够对所有 DMA 通道分配外设请求接口。当 DMA 通道线程执行 DMAWFP 时，peripheral [4:0] 域中设置的值指定了与 DMA 通道相关联的外设。关于更多信息，请参考第 16 - 38 页 “DMAWFP”。

DMAC 支持 31 个外设请求握手。每个请求握手最多可接收 4 个未完成请求，并分配一个指定的外设器件 ID。表 16 - 1 列出了外设器件 ID 分配。

表 16 - 1. 外设请求接口映射

外设	请求接口 ID	协议	外设	请求接口 ID	协议
FPGA 0	0	Synopsys	SPI Master 0 TX	16	Synopsys
FPGA 1	1	Synopsys	SPI Master 0 RX	17	Synopsys
FPGA 2	2	Synopsys	SPI Slave 0 TX	18	Synopsys
FPGA 3	3	Synopsys	SPI Slave 0 RX	19	Synopsys
FPGA 4 / CAN 0 interface 1	4	Synopsys / Bosch	SPI Master 1 TX	20	Synopsys
FPGA 5 / CAN 0 interface 2	5	Synopsys / Bosch	SPI Master 1 RX	21	Synopsys
FPGA 6 / CAN 1 interface 1	6	Synopsys / Bosch	SPI Slave 1 TX	22	Synopsys
FPGA 7 / CAN 1 interface 2	7	Synopsys / Bosch	SPI Slave 1 RX	23	Synopsys
I ² C 0 TX	8	Synopsys	Quad SPI Flash TX	24	ARM
I ² C 0 RX	9	Synopsys	Quad SPI Flash RX	25	ARM
I ² C 1 TX	10	Synopsys	STM	26	ARM
I ² C 1 RX	11	Synopsys	Reserved	27	—
I ² C 2 TX (EMAC)	12	Synopsys	UART 0 TX	28	Synopsys
I ² C 2 RX (EMAC)	13	Synopsys	UART 0 RX	29	Synopsys
I ² C 3 TX (EMAC)	14	Synopsys	UART 1 TX	30	Synopsys
I ² C 3 RX (EMAC)	15	Synopsys	UART 1 RX	31	Synopsys

请求接口编号 4-7 在 CAN 控制器与 FPGA 架构中实现的软核逻辑之间复用。CAN 控制器与 FPGA 接口之间的切换由系统管理器控制。



关于控制请求接口 4-7 的详细信息，请参考 *Cyclone V 器件手册* 卷 3 中的 *System Manager* 章节。

请求接受能力

对于每个外设请求接口，DMAC 能够接受一个活动请求。活动请求是 DMAC 还没有开始请求的 AXI 数据传递的地方。


外设长度管理

外设请求接口使外设能够控制一个 DMA 周期中包含的数据量，而无需 DMAC 知道它包含多少个数据传递。外设以下面的一种方式控制 DMA 周期：

- 选择单一传递
- 选择突发传递
- 当它开始当前系列中的最后请求时通知 DMAC

当 DMAC 执行 DMAWFP 外设指令时, 它停止执行线程, 并等待外设发送请求。当外设发送请求时, DMAC 根据以下信号的状态来设置请求标志位的状态:

- `drtype_<x>[1:0]`—DMAC 设置 `request_type` 标志位的状态:
 - `drtype_<x>[1:0]=b00`—`request_type<x> = Single`。
 - `drtype_<x>[1:0]=b01`—`request_type<x> = Burst`。
- `drlast_<x>`—DMAC 设置 `request_last` 标志位的状态:
 - `drlast_<x>=0`—`request_last<x> = 0`。
 - `drlast_<x>=1`—`request_last<x> = 1`。

 如果 DMAC 执行一个 DMAWFP 单一或者 DMAWFP 突发指令, 那么 DMAC 设置:

- `request_type<x>` 标志位为 Single 或者 Burst
- `request_last<x>` 标志位为 0

DMALPFE 是一个汇编指令, 强制相关的 DMALPEND 指令将其 `nf` 比特设成 0。这会创建一个程序循环 (program loop), 该程序循环不使用循环计数器来终止循环。

当 `request_last` 标志位设为 1 时, DMAC 退出循环。

DMAC 根据 `request_type` 和 `request_last` 标志位的状态, 有条件地执行下面的命令:

- DMALD, DMAST, DMALPEND

当这些指令使用可选的 B|S 后缀时, 如果 `request_type` 标志位不匹配, 那么 DMAC 执行 DMANOP。
- DMALDP<B|S>, DMASTP<B|S>

如果 `request_type<x>` 标志位不匹配 B|S, 那么 DMAC 执行 DMANOP。
- DMALPEND

当 `nf` 比特为 0 时, 如果 `request_last` 标志位被设置, 那么 DMAC 执行 DMANOP。

当 DMAC 接收到一个突发请求时, 如果需要 DMAC 发出一个突发传递, 那么使用 DMALDB, DMALDPB, DMASTB 和 DMASTPB 指令。CCRn 寄存器中的值控制 DMAC 传递的数据量。


当 DMAC 接收到一个单一请求时, 如果需要 DMAC 发出一个单一传递, 那么使用 DMALDS, DMALDPS, DMASTS 和 DMASTPS 指令。DMAC 忽略 CCRn 寄存器中 `src_burst_len` 和 `dst_burst_len` 域中的值, 并将 `arlen[3:0]` 或者 `awlen[3:0]` 总线设成 0x0。

DMAC 长度管理

DMAC 长度管理是 DMAC 何时控制要传递的数据总量。当外设需要 DMAC 传递数据到外设, 或从外设传递数据时, 外设使用外设请求接口来通知 DMAC。DMA 通道线程控制 DMAC 如何响应外设请求。

下面约束应用到 DMAC 传递管理:

- 来自外设的所有单一请求的数据总量必须小于来自该外设的突发请求的数据量。

 CCRn 寄存器控制对突发请求和单一请求传递的数据量。Altera 建议在通道 *n* 上进行数据传递时不要更新 CCRn 寄存器。

- 当外设发送一个突发请求时，直到 DMAC 确认了突发请求已经完成，外设才能发送一个单一请求。

当您需要程序线程终止执行，直到外设请求接口接收到任意请求类型时，使用 DMAWFP 单一指令。如果请求 FIFO 缓存中的头入口 (head entry) 是下面的类型：

- Single—DMAC 从 FIFO 缓存中获取入口，并继续程序执行。
- Burst—DMAC 保持 FIFO 缓存中的入口，并继续程序执行。



突发请求入口保持在请求 FIFO 缓存中，直到 DMAC 执行 DMAWFP 突发指令或者 DMAFLUSHP 指令。

当您需要程序线程终止执行，直到外设请求接口接收到突发请求时，使用 DMAWFP 突发指令。如果请求 FIFO 缓存中的头入口 (head entry) 是下面的类型：

- Single—DMAC 移除 FIFO 缓存中的入口，并保持程序运行。
- Burst—DMAC 从 FIFO 缓存中获取入口，并继续程序执行。

当 DMAC 完成 AXI 读传递时，如果需要 DMAC 发送一个应答 (acknowledgement) 到外设，那么使用 DMALDP 指令。类似地，当 DMAC 完成 AXI 写传递时，如果需要 DMAC 发送一个应答 (acknowledgement) 到外设，那么使用 DMASTP 指令。DMAC 使用确认接收总线来发送一个传递应答信号到外设 <x>。



当 rvalid 和 rlast 信号为高电平时，DMAC 发送一个读传输的 acknowledgement。当 bvalid 信号为高电平时，DMAC 发送一个写传输的 acknowledgement。对目的地的写数据传递期间，DMAC 可能发送一个 acknowledgement 到外设。

使用 DMAFLUSHP 指令复位外设请求接口的请求 FIFO 缓存。DMAC 执行 DMAFLUSHP 后，它会忽略外设请求，直到外设确认接收刷新 (flush) 请求。这使 DMAC 和外设能够彼此同步。

限制

与 DMA 外设请求接口连接的外设使用下面其中一个协议：

- ARM
- Synopsys
- Bosch

对于使用 ARM 协议的外设，DMA 与外设之间的唯一逻辑是时钟交叉逻辑。对于其它协议，时钟交叉和协议适应逻辑位于 DMA 与外设之间。下面小节讨论了使用 Synopsys 或者 Bosch 协议的外设的外设握手接口的限制。

仅突发请求

CAN 控制器使用的 Bosch 协议仅支持突发请求。因此，任何时候 CAN 控制器发出外设突发请求，DMA 控制器接收突发和单一请求。

不支持 Flush

对于使用 Bosch 或 Synopsys 外设请求协议的外设，flush 命令是不支持的。发出一个 flush 命令到任何支持 Bosch 或 Synopsys 协议的接口都会在到达外设前被终止。

无应答类型

对于 Bosch 外设请求接口协议, 没有可用的应答。发出一个应答到任何 CAN 控制器都会在到达外设前被终止。

对于 Synopsys 外设请求接口协议, 没有可用的应答类型。因此使用 Synopsys 协议的外设不能区分突发与单一传递的应答。

使用时间和中断

DMAC 能够支持 8 个事件和中断。INTEN 寄存器用于控制每个事件中断 (event-interrupt) 资源是一个事件还是一个中断。

当 DMAC 执行 DMASEV 指令时, 它会修改您指定的 event-interrupt 资源。INTEN 寄存器设置 event-interrupt 资源来用作:

- **Event**—DMAC 对指定的 event-interrupt 资源生成一个事件。当 DMAC 对同一 event-interrupt 资源执行 DMAWFE 指令时, 它会清除该事件。
- **Interrupt**—DMAC 设置 `irq<event_num>` 信号高电平, 其中的 `<event_num>` 是指定的事件资源数量。您必须写入 INTCLR 寄存器来清除中断。

使用一个事件来重新开始 DMA 通道

当您编程 INTEN 寄存器来生成一个事件时, 您可以使用 DMASEV 和 DMAWFE 指令来重新开始一个或多个 DMA 通道。

DMAC 在 DMASEV 之前执行 DMAWFE

要重新开始一个单一 DMA 通道:

1. 第一个 DMA 通道执行 DMAWFE, 然后在等待事件出现期间停止。
2. 另一个 DMA 通道使用同一事件编号执行 DMASEV。这会生成一个事件, 第一个 DMA 通道重新开始。DMAC 在执行 DMASEV 后的一个周期清除事件。

您可以编程多个通道来等待同一事件。例如, 如果四个 DMA 通道具有事件 2 的全部已执行的 DMAWFE, 那么当另一个 DMA 通道对事件 2 执行 DMASEV 时, 四个 DMA 通道全部同时重新开始。DMAC 在执行 DMASEV 后的一个周期清除事件。

DMAC 在 DMAWFE 之前执行 DMASEV

如果 DMAC 在另一个通道执行 DMAWFE 之前执行 DMASEV, 那么事件暂挂, 直到 DMAC 执行 DMAWFE。当 DMAC 执行 DMAWFE 时, 它停止执行一个时钟周期, 清除事件, 然后继续执行通道线程。

例如, 如果 DMAC 执行 DMASEV 6, 并且没有其它线程已经执行 DMAWFE 6, 那么事件暂挂。如果 DMAC 对通道 4 执行 DMAWFE 6 指令, 然后对通道 3 执行 DMAWFE 6 指令, 那么:

1. DMAC 停止执行通道 4 线程一个 aclk 时钟。
2. DMAC 清除事件 6。
3. DMAC 继续执行通道 4 线程。
4. DMAC 停止执行通道 3 线程, 并且当 DMAC 等待下一个事件 6 出现时, 线程停止。

中断 MPU 子系统

DMAC 提供 `irq[x]` 信号, 用作 MPU 子系统的高有效电平敏感中断。当编程 `INTEN` 寄存器生成中断时, DMAC 执行 `DMASEV` 后, 它将设置相应的 `irq[x]` 信号高电平。

MPU 子系统能够通过写入 `INTCLR` 寄存器来清除中断。



执行 `DMAWFE` 不会清除中断。

当 DMAC 完成 `DMALD` 或 `DMAST` 指令时, 如果使用 `DMASEV` 指令通知微处理器, 那么 Altera 建议您在 `DMASEV` 之前插入一个存储器屏障指令。否则, 在 AXI 传递完成前 DMAC 可能会发出一个中断信号。请参考例 16-1。

例 16-1. 存储器屏障指令

```
DMALD
DMAST
# Issue a write memory barrier
# Wait for the AXI write transfer to complete before the DMAC
# can send an interrupt
DMAWMB
# The DMAC sends the interrupt
DMASEV
```

终止 (Aborts)

本部分描述:

- 终止类型 (abort types)
- 终止源 (Abort Sources)
- Watchdog Abort
- 终止处理

终止类型 (abort types)

根据发生终止时 DMAC 是否提供给 DMAC 终止处理程序一个精确状态的 DMAC, 一个终止 (abort) 可以归类为精确的 (precise) 或不精确的 (imprecise)。如果一个终止是:

- **Precise** DMAC 使用创建 abort 的指令地址更新 PC 寄存器。
- **Imprecise** PC 寄存器可能包括没有导致出现 abort 的指令地址。

终止源 (Abort Sources)

DMAC 在下面条件下发出精确的 abort 信号:

- Non-secure 状态中的 DMA 通道线程试图编程其 `CCRn` 寄存器并生成一个安全的 AXI 传输。
- Non-secure 状态中的 DMA 通道线程对设为安全的事件执行 `DMAWFE` 或者 `DMASEV`。
`boot_irq_ns` 存储器映射控制信号初始化事件的安全状态。



对于每个事件, `INTEN` 寄存器控制 DMAC 生成一个事件还是发出一个中断信号。

- DMA 通道线程试图执行 DMAST, 但 DMAC 计算它最终何时执行存储, MFIFO 缓存没有包含足够的数据使其完成存储。
- Non-secure 状态的 DMA 通道线程对设置为 secure 的外设请求接口执行 DMAWFP, DMALDP, DMASTP 或者 DMAFLUSHP。boot_periph_ns 存储器映射控制信号初始化外设请求接口的安全状态。
- Non-secure 状态的 DMA 管理器线程执行 DMAGO, 试图开始一个安全的 DMA 通道线程。
- 当 DMAC 执行取指令时, 它在 AXI 主接口上接收到的 ERROR 响应。
- 线程执行未定义的指令。
- 线程通过一个对 DMAC 配置无效的操作数来执行指令。



当 DMAC 信号是一个精确终止 (precise abort) 时, DMAC 不执行触发 abort 的指令, 而执行 DMANOP。

在下面情况下, DMAC 信号是一个不精确终止 (imprecise abort):

- 当 DMAC 执行数据加载时, 它在 AXI 主接口上接收到 ERROR 响应
- 当 DMAC 执行数据存储时, 它在 AXI 主接口上接收到 ERROR 响应
- DMA 通道线程执行 DMALD 或者 DMAST, MFIFO 缓存太小而不能存储所需数据量
- DMA 通道线程执行 DMAST, 但线程还没有执行足够的 DMALD 指令
- DMA 通道会因为资源缺乏而锁定, 从而导致内部看门狗计时器超时。

Watchdog Abort

如果一个或多个 DMA 通道程序运行和 MFIFO 缓存太小而不能满足 DMA 程序的存储要求, 那么 DMAC 能够锁定。

DMAC 包含逻辑, 防止保持在一种不能完成 DMA 传递的状态。

当下面全部情况出现时, DMAC 会检测到一个锁定 (lockup):

- 加载队列为空
- 存储队列为空
- 由于 MFIFO 缓存没有足够的可用空间, 或者另一个通道有加载锁 (load-lock), 因而防止所有运行通道执行 DMALD 指令。

当 DMAC 检测到 lockup 时, 它发出一个中断信号, 并能够终止运行的通道。DMAC 行为取决于 WD 寄存器中 wd_irq_only 比特的状态。

如果:

- wd_irq_only=0—DMAC 终止所有运行的 DMA 通道, 并设置 irq_abort 信号高电平。
- wd_irq_only=1—DMAC 设置 irq_abort 信号高电平。

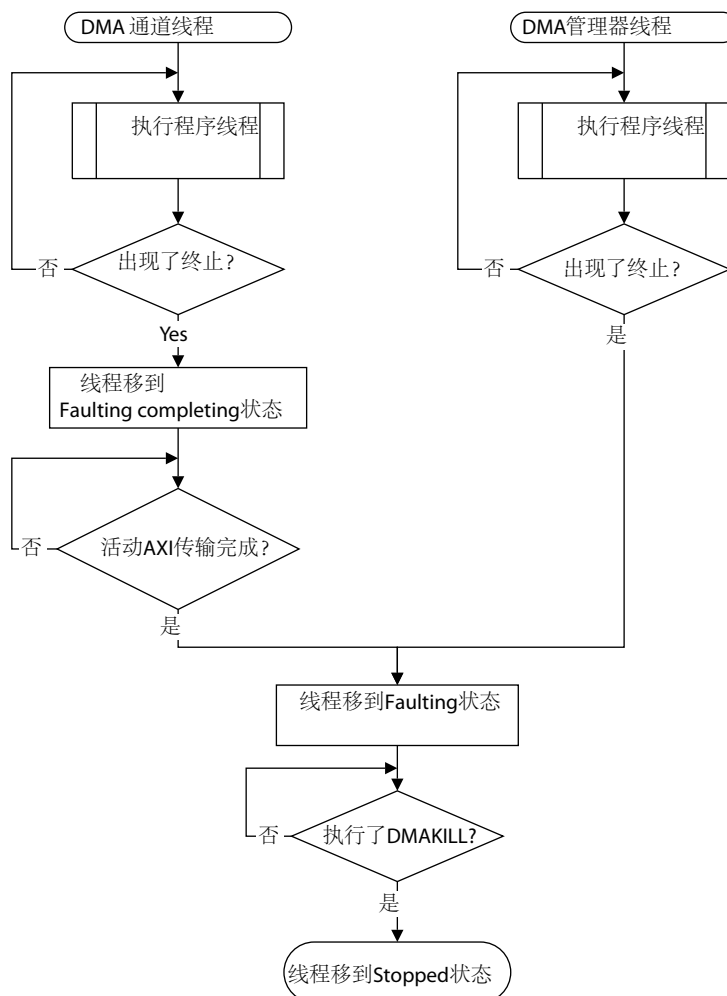
关于详细信息, 请参考第 16 - 23 页 “DMA 通道之间的资源共享”。

终止处理

DMAC 的体系结构没有设计成从 abort 恢复, 因此当 abort 发生时您必须使用外部器件 (例如微处理器) 来终止一个线程。


图 16-4 显示了 abort 出现后的 DMA 通道和 DMA 管理器线程的操作状态。

图 16-4. 终止流程



abort 出现后, DMAC 根据不同线程类型采取相应操作:

- DMA 通道线程 — 线程立即移到 Faulting 完成状态。在此状态中, DMAC:
 - 设置 irq_abort 信号高电平
 - 停止执行 DMA 通道的指令
 - 使 DMA 通道的全部高速缓存入口无效, 更新 CPCn 寄存器以包含终止的指令地址 (如果终止是精确的)
 - 对读写队列中剩余的指令不生成 AXI 访问
 - 允许完成当前的有效 AXI 传输

 DMA 通道完成传输后, 线程移到 Faulting 状态。


- DMA 管理器线程 — 线程立即移到 Faulting 状态, DMAC 设置 `irq_abort` 信号高电平。

外部器件可以通过下面操作响应 `irq_abort` 信号的置位:

- 读取 `FSRD` 寄存器的状态来确定 DMA 管理器是否是 Faulting。在 Faulting 状态中, `FSRD` 寄存器终止的原因。
- 读取 `FSRC` 寄存器的状态来确定 DMA 通道是否是 Faulting。在 Faulting 状态中, `FSRC` 寄存器提供终止的原因。

要使 Faulting 状态的线程移到 Stopped 状态, 外部器件必须:

- 使用 `DMAKILL` 指令的编码编程 `DBGINST0` 寄存器。
- 写入 `DBGCMD` 寄存器。

 如果终止的线程是安全的, 您必须使用安全从接口来更新这些

Faulting 状态的线程执行 `DMAKILL` 后移到 Stopped 状态。

安全使用

当 DMAC 从复位状态退出时, 配置信号的状态用于:

- DMA 管理器线程 — `DSR` 寄存器中的 `DNS` 比特返回 DMA 管理器线程的安全状态。
- 事件和中断 — `CR3` 寄存器中的 `INS` 比特返回事件中断资源的安全状态。
- 外设请求接口 — `CR4` 寄存器中的 `PNS` 比特返回这些接口的安全状态。

此外, 每个 DMA 通道线程包含一个动态 non-secure 比特 `CNS`, 当通道不在 Stopped 状态时有效。

Secure 状态的 DMA 管理器线程

如果 `DNS` 比特为 0, 那么 DMA 管理器线程运行在 Secure 状态, 并且只执行安全的取指令。当一个 Secure 状态的 DMA 管理器线程处理:

- `DMAGO`—DMAC 使用 `ns` 比特的状态, 通过写入 DMA 通道的 `CNS` 比特来设置该 DMA 通道线程的安全状态。
- `DMAWFE`—DMAC 停止线程的执行, 直到事件出现。当事件出现时, DMAC 继续执行线程, 不考虑相应 `INS` 比特的安全状态。
- `DMASEV`—DMAC 设置 `INT_EVENT_RIS` 寄存器中的相应比特, 不考虑相应 `INS` 比特的安全状态。

Non-Secure 状态的 DMA 管理器线程

如果 `DNS` 比特为 1, 那么 DMA 管理器线程运行在 Non-secure 状态, 并且只执行非安全的取指令。当一个 Non-secure 状态的 DMA 管理器线程处理:

- DMAGO—DMAC 使用 ns 比特的状态, 控制是否开始一个 DMA 通道线程。如果:
 - ns = 0—DMAC 不会开始 DMA 通道线程, 而是:
 - 执行 NOP。
 - 设置 FSRD 寄存器。
 - 设置 FTRD 寄存器中的 dmago_err 比特。
 - 将 DMA 管理器移到 Faulting 状态。
 - ns = 1—DMAC 开始一个 Non-secure 状态的 DMA 通道线程, 并编程 CNS 比特成 non-secure。
- DMAWFE—DMAC 使用 CR3 寄存器中的相应 INS 比特的状态, 控制是否等待事件。如果:
 - INS = 0—事件在 Secure 状态。DMAC:
 - 执行 NOP。
 - 设置 FSRD 寄存器。
 - 设置 FTRD 寄存器中的 mgr_evt_err 比特。
 - 将 DMA 管理器移到 Faulting 状态。
 - INS = 1—事件在 Non-secure 状态。DMAC 停止线程的执行, 等待事件出现。
- DMASEV—DMAC 使用 CR3 寄存器中的相应 INS 比特的状态, 控制是否创建事件中断。如果:
 - INS = 0—事件中断资源处于 Secure 状态。DMAC:
 - 执行 NOP。
 - 设置 FSRD 寄存器。
 - 设置 FTRD 寄存器中的 mgr_evt_err 比特。
 - 将 DMA 管理器移到 Faulting 状态。
 - INS = 1—事件中断资源处于 Non-secure 状态。DMAC 创建事件中断。

Secure 状态的 DMA 通道线程

当 CNS 比特为 0 时, DMA 通道线程被编程以运行在 Secure 状态, 它仅执行安全的取指令。

当 Secure 状态的 DMA 通道线程执行下面指令时:

- DMAWFE—DMAC 停止线程的执行, 直到事件出现。当事件出现时, DMAC 继续执行线程, 不考虑 CR3 寄存器中相应 INS 比特的安全状态。
- DMASEV—DMAC 创建事件中断, 不考虑 CR3 寄存器中相应 INS 比特的安全状态。
- DMAWFP—DMAC 停止线程的执行, 直到外设发出一个 DMA 请求信号。外设发出一个 DMA 请求信号后, DMAC 继续执行线程, 不考虑 CR4 寄存器中相应 PNS 比特的安全状态。
- DMALDP 和 DMASTP—DMAC 发送一条消息到外设, 告知数据传递完成, 不考虑 CR4 寄存器中相应 PNS 比特的安全状态。
- DMAFLUSHP—DMAC 清除外设的状态, 并发送消息到外设, 重发送它的电平状态, 不考虑 CR4 寄存器中相应 PNS 比特的安全状态。

当 DMA 通道线程处于 Secure 状态时, 它使 DMAC 能够执行安全的和非安全的 AXI 访问。

Non-Secure 状态的 DMA 通道线程

当 CNS 比特为 1 时, DMA 通道线程被编程以运行在 Non-secure 状态, 它仅执行非安全的取指令。

当 Non-secure 状态的 DMA 通道线程执行下面指令时:

- DMAWFE—DMAC 使用 CR3 寄存器中的相应 INS 比特的状态, 控制是否等待事件。如果
 - INS = 0—事件处于 Secure 状态。DMAC:
 - 执行 NOP。
 - 设置 FSRC 寄存器中对应于 DMA 通道编号的相应比特。
 - 设置 FTRn 寄存器中的 ch_evnt_err 比特。
 - 将 DMA 通道移到 Faulting 完成状态。
 - INS = 1—事件处于 Non-secure 状态。DMAC 停止线程的执行, 并等待事件出现。
- DMASEV—DMAC 使用 CR3 寄存器中的相应 INS 比特的状态, 控制是否创建事件。如果
 - INS = 0—事件中中断资源处于 Secure 状态。DMAC:
 - 执行 NOP。
 - 设置 FSRC 寄存器中对应于 DMA 通道编号的相应比特。
 - 设置 FTRn 寄存器中的 ch_evnt_err 比特。
 - 将 DMA 通道移到 Faulting 完成状态。
 - INS = 1—事件中中断资源处于 Non-secure 状态。DMAC 创建事件中断。
- DMAWFP—DMAC 使用 CR4 寄存器中的相应 PNS 比特的状态, 控制是否等待外设发出请求信号。如果:
 - PNS = 0—外设处于 Secure 状态。DMAC:
 - 执行 NOP。
 - 设置 FSRC 寄存器中对应于 DMA 通道编号的相应比特。
 - 设置 FTRn 寄存器中的 ch_periph_err 比特。
 - 将 DMA 通道移到 Faulting 完成状态。
 - PNS = 1—外设处于 Non-secure 状态。DMAC 停止线程的执行, 并等待外设发出一个请求信号。
- DMALDP 和 DMASTP—DMAC 使用 CR4 寄存器中的相应 PNS 比特的状态来控制是否发送一个 acknowledgement 到外设。如果:
 - PNS = 0—外设处于 Secure 状态。DMAC:
 - 执行 NOP。
 - 设置 FSRC 寄存器中对应于 DMA 通道编号的相应比特。
 - 设置 FTRn 寄存器中的 ch_periph_err 比特。
 - 将 DMA 通道移到 Faulting 完成状态。
 - PNS = 1—外设处于 Non-secure 状态。当数据传递完成时, DMAC 发送消息到外设。

- DMAFLUSHP—DMAC 使用 CR4 寄存器中的相应 PNS 比特的状态, 控制是否发出一个 flush 请求到外设。如果:
 - PNS = 0— 外设处于 Secure 状态。DMAC:
 - 执行 NOP。
 - 设置 FSRC 寄存器中对应于 DMA 通道编号的相应比特。
 - 设置 FTRn 寄存器中的 ch_periph_err 比特。
 - 将 DMA 通道移到 Faulting 完成状态。
 - PNS = 1— 外设处于 Non-secure 状态。DMAC 清除外设的状态, 并发送消息到外设来重新发送它的电平状态。

当 DMA 通道线程处于 Non-secure 状态, 并且 DMAMOV CCR 指令尝试编程通道来执行一个安全的 AXI 传输时, DMAC:

1. 执行 DMANOP。
2. 设置 FSRC 寄存器中对应于 DMA 通道编号的相应比特。
3. 设置 FTRn 寄存器中的 ch_rdwr_err 比特。
4. 将 DMA 通道移到 Faulting 完成状态。

约束和使用限制

本部分介绍了约束和使用限制。

DMA 通道仲裁

DMAC 使用一个循环 (round-robin) 方案来运行活动 DMA 通道。要确保 DMAC 继续运行 DMA 管理器, 它要在运行下一个 DMA 通道前运行 DMA 管理器。

DMAC 的仲裁过程是不可能改变的。

DMA 通道优先权

DMAC 使用相同的优先权响应所有的活动 DMA 通道, 就某一个 DMA 通道提高其优先权是不可能的。

指令高速缓存延迟

当出现高速缓存缺失 (cache miss) 时, 实现请求的延迟主要取决于包含 DMA 代码的存储器的读延迟。DMAC 添加的延迟是最小延迟。

AXI 数据传递大小

DMAC 能够执行高达 64 比特宽的数据访问。如果您将 src_burst_size 或 dst_burst_size 域编程为更大的值, 那么 DMAC 会发出一个精确的终止信号。关于详细信息, 请参考第 16 - 14 页 “终止源 (Abort Sources)”。

跨越 4 KB 边界的 AXI 突发

AXI 规范不允许 AXI 突发跨越 4 KB 地址边界。如果您使用可能导致单一突发跨越 4 KB 地址边界的突发起始地址, 大小和长度的组合编程 DMAC, 那么 DMAC 将使用指定组的合长度生成一对突发。此操作对于 DMAC 通道线程是透明的, 因此 DMAC 通过生成相应的 AXI 读突发对来响应单一的 DMALD 指令。

AXI 突发类型

您可以通过编程 DMAC 仅生成数据访问的固定地址或增量地址突发类型。它不对数据访问或者指令获取生成包装地址。

AXI 写地址

DMAC 能够发出高达 8 个未处理 (outstanding) 写地址。DMAC 直到读取被要求用于实现写传输的全部数据字节时, 才发出写地址。

AXI 写数据交错

DMAC 不生成交错的写数据。一个写传输的所有写数据差拍 (beat) 都是下一个写传输的任何写数据差拍前的输出。

编程限制

下面部分介绍了编程 DMAC 时应用的限制。

固定未对齐突发

DMAC 不支持固定的未对齐突发。如果编程下面条件, 那么 DMAC 会把这当作一个编程错误:

- 未对齐读
 - CCRn 寄存器中的 src_inc 域为 0。
 - SARn 寄存器包含一个地址, 该地址与 src_burst_size 域中包含的数据大小不对齐。
- 未对齐写
 - CCRn 寄存器中的 dst_inc 为 0。
 - DARN 寄存器包含一个地址, 该地址与 dst_burst_size 域中包含的数据大小不对齐。

Endian 交换大小限制

如果您编程 CCRn 寄存器中的 endian_swap_size 域, 使 DMA 通道能够执行 Endian 交换, 那么您必须在执行 DMALD 或 DMAST 指令前设置相应的 SARn 寄存器和 DARN 寄存器, 以包含一个地址, 该地址对齐于 endian_swap_size 域指定的大小。



如果使用 DMAADDH SAR 指令更新 endian_swap_size, SARn 或者 DARN, 那么您必须确保在执行任何其它 DMALD 或 DMASTSARn 指令前, SARn 和 DARN 寄存器包含一个对齐于 endian_swap_size 域指定的大小的地址。

如果您通过编程 CCRn 寄存器中的 src_inc 域来使用一个固定地址, 那么您必须编程 src_burst_size 域来选择一个大于或等于 endian_swap_size 域指定的值的突发大小。类似地, 如果您通过编程 dst_inc 域来选择一个固定目的地址, 那么您必须编程 dst_burst_size 域来选择一个大于或等于 endian_swap_size 域指定的值的突发大小。

如果您通过编程 CCRn 寄存器中的 dst_inc 域来使用一个增量地址, 那么您必须编程 CCRn 寄存器, 因而 dst_burst_len 与 dst_burst_size 的乘积是 endian_swap_size 的倍数。例如, 如果 endian_swap_size = 2, 32-bit, 和 dst_burst_size = 1, 2 bytes 每个差拍 (per beat), 那么您能编程 dst_burst_len = 1, 3, 5, ..., 15, 也就是 2, 4, 6, ..., 16 个传递。

DMA 周期中更新 DMA 通道控制寄存器

DMAC 执行一序列 DMALD 和 DMAST 指令前, CCRn, SARn 和 DARN 寄存器中配置的值控制 DMA 从源地址到目的地址传递数据时执行的数据字节通道操作。

您可以在 DMA 周期中更新这些寄存器, 但是如果修改某些寄存器域, 那么它能够导致 DMAC 丢弃数据。下面部分介绍了那些对数据传递有不良影响的寄存器域。

影响目的地址的更新

如果您使用 DMAMOV 指令通过 DMA 周期更新 DARN 寄存器或者 CCRn 寄存器, 那么这可能会导致目的数据流中的中断。

如果进行以下修改, 那么会出现中断:

- endian_swap_size 域
- dst_inc 比特
- 当 dst_inc = 0, dst_burst_size 域, 也就是, 固定地址突发。
- DARN 寄存器, 以便修改目的字节通道对齐。由于总线宽度是 64 比特, 因此要修改 DARN 寄存器中的比特 [2:0]。

当目的数据流中出现中断时, DMAC:

1. 停止 DMA 通道线程的执行。
2. 完成所有未执行的通道读写操作, 就好像 DMAC 执行 DMARMB 和 DMAWMB。
3. 丢弃所有剩余的 MFIFO 缓存数据。
4. 继续 DMA 通道线程的执行。

影响源地址的更新

如果您使用 DMAMOV 指令通过 DMA 周期更新 SARn 寄存器或者 CCRn 寄存器, 那么这可能会导致源数据流中的中断。

如果以下被修改, 那么会出现中断:

- src_inc 比特
- src_burst_size 域
- SARn 寄存器, 以便修改源字节通道对齐。由于总线宽度是 64 比特, 因此要修改 SARn 寄存器中的比特 [2:0]。

当源数据流中出现中断时, DMAC:

1. 停止 DMA 通道线程的执行。
2. 完成所有未执行的通道读写操作, 就好像 DMAC 执行 DMARMB。
3. 继续 DMA 通道线程的执行。MFIFO 缓存中的数据没有丢弃。

DMA 通道之间的资源共享

DMA 通道程序共享 MFIFO 缓存数据存储资源。一定不要使用大于 512 的资源要求 (MFIFO 缓存大小) 并发运行一组 DMA 通道程序。如果超过这一限制, 那么 DMAC 可能锁定, 并生成 Watchdog 终止, 请参考第 16 - 15 页 “Watchdog Abort”。

DMAC 包含一个称作 *load-lock* 的机制, 确保正确使用共享的 MFIFO 缓存。加载锁 (load-lock) 属于一个通道, 或者独立。具有加载锁的通道能够成功执行 DMALD 指令。不具有加载锁的通道在 DMALD 指令上暂停, 直到具有加载锁。

在下面两种情况下, 通道要求拥有加载锁:

- 通道执行 DMALD 或者 DMALDP 指令
- 当前没有其它通道拥有加载锁

当出现下面情况时, 通道释放加载锁的拥有权:

- 通道执行 DMAST, DMASTP 或者 DMASTZ
- 通道达到一个屏障, 也就是, 它执行 DMARMB 或者 DMAWMB
- 通道等待, 也就是, 它执行 DMAWFP 或者 DMAWFE
- 通道正常终止, 也就是, 它执行 DMAEND
- 由于某种原因通道终止, 包括 DMAKILL。

在 MFIFO 缓存入口测量 DMA 通道程序的 MFIFO 缓存资源使用, 随着程序的进行升高或降低。使用被加载锁机制影响的 *static requirement* 和 *dynamic requirement* 描述 DMA 通道程序的 MFIFO 缓存资源要求。

在通道执行下面其中一个操作前, 静态要求 (static requirement) 被定义成此通道当前使用的最大数量的 MFIFO 缓存入口:

- 执行 WFP 或 WFE 指令
- 要求加载锁的所有权

动态要求被定义成通道程序随时使用的静态要求与最大数量 MFIFO 缓存入口之间的差异。

要计算总 MFIFO 缓存要求, 需要将最大动态要求与所有静态要求和相加。

要避免 DMAC 锁定, 通道程序组的总 MFIFO 缓存要求必须小于等于 512 (MFIFO 缓存深度)。

关于详细信息, 请参考第 16 - 42 页 “MFIFO 缓存使用概况”。

DMA 控制器编程模型

这一部分介绍了 DMAC 的指令集。

指令语法定义

以下约定用于汇编语法原型行及其子域:

- < >

< > 中的各项是必需的。随后的文本描述了各项及如何在指令中编码。

- []
[] 中的各项是可选的。随后的文本描述了各项及如何在指令中对其出现或缺失进行编码。
- 空格
为了清晰，使用单空格分离各项。在汇编语法中，如果空格是必需的，那么使用两个或多个连续空格。

指令集汇总

- DMAC 指令：
- 使用 DMA 前缀，提供唯一的命名空间
 - 具有 8-bit 操作码，可能使用 0, 8, 16 或 32 bits 的可变数据净荷
 - 使用一致的后缀
- 表 16 - 2 为指令语法的汇总。

表 16 - 2. 指令语法汇总

助记符	指令	DMA 管理器 使用	DMA 通道 使用	说明
DMAADDH	Add Halfword	No	Yes	请参考第 16 - 25 页 “DMAADDH”。
DMAADNH	Add Negative Halfword	No	Yes	请参考第 16 - 25 页 “DMAADNH”。
DMAEND	End	Yes	Yes	请参考第 16 - 26 页 “DMAEND”。
DMAFLUSHP	Flush and Notify Peripheral	No	Yes	请参考第 16 - 26 页 “DMAFLUSHP”。
DMAGO	Go	Yes	No	请参考第 16 - 27 页 “DMAGO”。
DMAKILL	Kill	Yes	Yes	请参考第 16 - 28 页 “DMAKILL”。
DMALD	Load	No	Yes	请参考第 16 - 29 页 “DMALD[S B]”。
DMALDP	Load and Notify Peripheral	No	Yes	请参考第 16 - 30 页 “DMALDP<S B>”。
DMALP	Loop	No	Yes	请参考第 16 - 31 页 “DMALP”。
DMALPEND	Loop End	No	Yes	请参考第 16 - 31 页 “DMALPEND[S B]”。
DMALPFE	Loop Forever	No	Yes	请参考第 16 - 33 页 “DMALPFE”。
DMAMOV	Move	No	Yes	请参考第 16 - 33 页 “DMAMOV”。
DMANOP	No Operation	Yes	Yes	请参考第 16 - 34 页 “DMANOP”。
DMARMB	Read Memory Barrier	No	Yes	请参考第 16 - 35 页 “DMARMB”。
DMASEV	Send Event	Yes	Yes	请参考第 16 - 35 页 “DMASEV”。
DMAST	Store	No	Yes	请参考第 16 - 36 页 “DMAST[S B]”。
DMASTP	Store and Notify Peripheral	No	Yes	请参考第 16 - 37 页 “DMASTP<S B>”。
DMASTZ	Store Zero	No	Yes	请参考第 16 - 37 页 “DMASTZ”。
DMAWFE	Wait For Event	Yes	Yes	请参考第 16 - 38 页 “DMAWFE”。
DMAWFP	Wait For Peripheral	No	Yes	请参考第 16 - 38 页 “DMAWFP”。
DMAWMB	Write Memory Barrier	No	Yes	请参考第 16 - 39 页 “DMAWMB”。

指令

接下来的部分介绍了 DMAC 能够执行的指令。

DMAADDH

对于 DMA 通道线程，Add Halfword 添加一个 16-bit 立即值到 SARn 寄存器或者 DARn 寄存器，这使 DMAC 能够支持 2D DMA 操作。


 在 DMAC 使用 32-bit 加法将 16-bit 无符号立即值与地址相加之前，该 16-bit 无符号立即值是一个补零值 (zero-extended)。DMAC 丢弃进位比特 (carry bit)，从而地址范围是 0xFFFFFFFF 到 0x00000000。

图 16 - 5 为指定编码。

图 16 - 5. DMAADDH 编码

23	16	15	8	7	6	5	4	3	2	1	0
imm[15:8]				imm[7:0]				0	1	0	1
								0	1	ra	0

汇编语法

DMAADDH <address_register>, <16-bit immediate>

其中：

<address_register>— 选择要使用的地址寄存器，必须是：

SAR SARn 寄存器，并将 ra 设置为 0

或者， DAR DARn 寄存器，并将 ra 设置为 1

<16-bit immediate>— 与 <address_register> 相加的立即值。

操作

您只能在 DMA 通道线程中使用这一指令。

DMAADNH

对于 DMA 通道线程，Add Negative Halfword 将一个 16-bit 立即负值加到 SARn 寄存器或者 DARn 寄存器，这使 DMAC 能够支持 2D DMA 操作，或者以不同的顺序读取或者写入存储器中的一块区域到自然增加地址。


 16-bit 无符号立即值是一个扩展到 32 比特的值 (one-extended)，创建一个 -65536 到 -1 之间的二进制补码表示法，然后 DMAC 使用 32-bit 加法将此值与地址相加。DMAC 丢弃进位比特 (carry bit)，从而地址范围是 0xFFFFFFFF 到 0x00000000。净影响是从源或目的地址寄存器的当前值减去 65536 和 1 之间的值。

图 16 - 6 为指令编码。

图 16 - 6. DMAADNH 编码

23	16	15	8	7	6	5	4	3	2	1	0
imm[15:8]				imm[7:0]				0	1	0	1
								0	1	ra	0

汇编语法

DMAADNH <address_register>, <16-bit immediate>

其中：

<address_register>— 选择要使用的地址寄存器。必须是：

SAR SARn 寄存器，并将 ra 设置为 0

或者， DAR DARn 寄存器，并将 ra 设置为 1

<16-bit immediate>— 与 <address_register> 相加的立即值。



您应该将 16-bit 立即值指定为一个在指令编码中表示的数。例如， DMAADNH DAR, 0xFFFF0 导致 0xFFFFFFF0 与 Destination Address 寄存器的当前值相加，有效地从 DAR 中减去 16。

操作

您只能在 DMA 通道线程中使用这一指令。

DMAEND

DMAC 的结束信号，表示 DMA 序列完成。DMA 通道的全部 DMA 传递完成后，DMAC 将通道移到 Stopped 状态，并刷新 MFIFO 缓存中的数据，使线程的所有高速缓存入口无效。

图 16 - 7 为指令编码。

图 16 - 7. DMAEND 编码

7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0

汇编语法

DMAEND

操作

此指令可用于 DMA 管理器线程和 DMA 通道线程。

DMAFLUSHP

Flush Peripheral 清除 DMAC 中的状态，此状态描述外设的内容，并发送消息到外设，要求外设重新发送它的电平状态。

图 16 - 8 为指令编码。

图 16 - 8. DMAFLUSHP 编码

15	11	10	9	8	7	6	5	4	3	2	1	0
periph[4:0]				0	0	0	0	0	1	1	0	1

汇编语法：
DMAFLUSHP <peripheral>
其中：
<peripheral> 5-bit 立即值 0-31
操作
您只能在 DMA 通道线程中使用这一指令。

DMAGO

当 DMA 管理器对 Stopped 状态的 DMA 通道执行 Go 时，它在 DMA 通道上执行下面的步骤：

- 1. 将 32-bit 立即值移到程序计数器
- 2. 设置它的安全状态
- 3. 将它更新到 Executing 状态


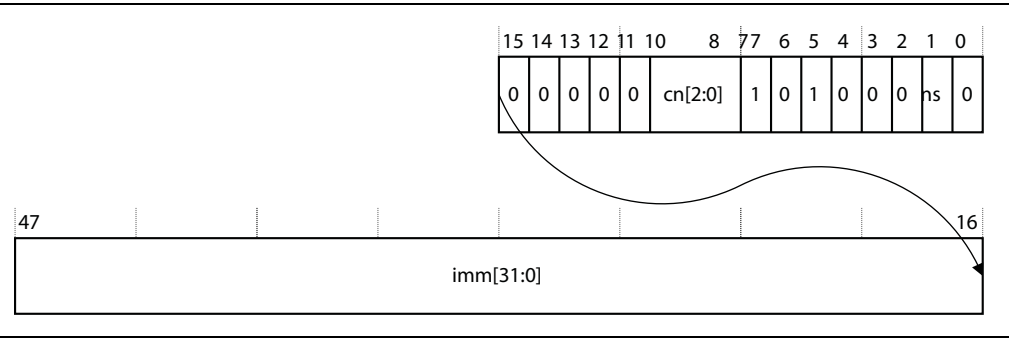
 当 DMA 管理器执行 DMAGO 时，如果 DMA 通道没有处于 Stopped 状态，那么 DMAC 不执行 DMAGO，而执行 DMANOP。

图 16 - 9 为指令编码。

图 16 - 9. DMAGO 编码




汇编语法

```
DMAGO <channel_number>, <32-bit_immediate> [, ns]
```

其中:

<channel_number>— 选择一个 DMA 通道, 必须是下面其中的一个 DMA 通道:

```
C0 DMA channel 0
C1 DMA channel 1
C2 DMA channel 2
C3 DMA channel 3
C4 DMA channel 4
C5 DMA channel 5
C6 DMA channel 6
C7 DMA channel 7
```

 如果提供的通道编号不适用于您的 DMAC 配置, 那么 DMA 管理器线程终止。

<32-bit_immediate>— 对所选的 <channel_number> 写入到 CPCn 寄存器的立即值。

[ns]

- 如果呈现 ns, 那么 DMA 通道运行在 Non-secure 状态。

- 否则, 指令的执行取决于 DMA 管理器的安全状态:

DMA manager is in the Secure state—DMA 通道运行在 Secure 状态。

DMA manager is in the Non-secure state—DMAC 终止。

操作

您只能在 DMA 管理器线程中使用这一指令。

DMAKILL

Kill 指示 DMAC 立即终止线程的执行。根据线程类型, DMAC 执行下面步骤:

DMA 管理器线程

1. 对于 DMA 管理器, 使所有高速缓存入口无效。
2. 将 DMA 管理器移到 Stopped 状态。

DMA 通道线程

1. 将 DMA 通道移到 Killing 状态。
2. 等待 AXI 传输 (具有一个等于 DMA 通道编号的 ID) 完成。
3. 对于 DMA 通道, 使所有高速缓存入口无效。
4. 对于 DMA 通道, 移除 MFIFO 缓存中的所有入口。
5. 对于 DMA 通道, 移除读缓存队列和写缓存队列中的所有入口。
6. 将 DMA 通道移到 Stopped 状态。

图 16 - 10 为指令编码。

图 16 - 10. DMAKILL 编码


7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	1

汇编语法

DMAKILL

操作

该指令可用于 DMA 管理器线程和 DMA 通道线程。

 一定不要在 DMA 通道程序中使用 DMAKILL 指令。使用 DBGINST0 寄存器，发出 DMAKILL 指令。

DMALD[S | B]

Load 指示 DMAC 执行 DMA 加载，使用源地址寄存器和通道控制寄存器指定的 AXI 传输。它将读数据放置在 MFIFO 缓存中，并附上相应的通道编号。DMALD 是一个无条件指令，但 DMALDS 和 DMALDB 在 request_type 标志位状态上是有条件指令。如果通道控制寄存器中的 src_inc 比特被设置成递增 (incrementing)，那么 DMAC 执行 DMALD[S|B] 后更新源地址寄存器。


 当 DMAC 执行 DMAWFP 指令时，它设置 request_type 值。请参考第 16 - 38 页 “DMAWFP”。

图 16 - 11 为指令编码。

图 16 - 11. DMALD[S|B] 编码

7	6	5	4	3	2	1	0
0	0	0	0	0	1	bs	x

汇编语法

DMALD[S|B]

其中：

[S] 如果呈现 S，汇编程序将 bs 设成 0，x 设成 1。该指令在 request_type 标志位状态上是有条件的：

- request_type = Single
DMAC 执行 DMALD 指令，并设置 arlen[3:0]=0x0，这样 AXI 读传输长度为 1。DMAC 忽略通道控制寄存器中 src_burst_len 域中的值。

- request_type = Burst
DMAC 执行 DMANOP 指令。DMAC 递增通道 PC 到下一个指令。没有出现状态变化。

[B] 如果呈现 B，汇编程序将 bs 设成 1，x 设成 1。该指令在 request_type 标志位状态上是有条件的：

- request_type = Single
DMAC 执行 DMANOP 指令。DMAC 递增通道 PC 到下一个指令。没有出现状态变化。
- request_type = Burst
DMAC 执行 DMALD。

如果不指定 S 或 B 操作数，汇编程序将 bs 设成 0， x 设成 0， DMAC 始终执行 DMA 加载。

操作
您只能在 DMA 通道线程中使用这一指令。如果指定 S 或 B 操作数，指令的执行在 request_type 状态（匹配该指令的状态）上是有条件的。

DMALDP<S | B>

Load and notify Peripheral 指示 DMAC 执行 DMA 加载，使用源地址寄存器和通道控制寄存器指定的 AXI 传输。DMA 将读数据放置在 MFIFO 缓存中，并附上相应的通道编号，DMA 收到最后一个数据项后，发送一个数据传递完成的应答 (acknowledgement) 到外设。如果通道控制寄存器中的 src_inc 比特被设置成递增 (incrementing)，那么 DMAC 执行 DMALD[S|B] 后更新源地址寄存器。

图 16 - 12 为指令编码。

图 16 - 12. DMALDP<S|B> 编码



汇编语法

DMALDP<S|B> <peripheral>

其中：

<S> 当 S 呈现时，汇编程序将 bs 设成 0。该指令在 request_type 标志位状态上是有条件的：


- request_type = Single
DMAC 执行 DMALDP 指令，并设置 arlen[3:0]=0x0，这样 AXI 读传输长度为 1。DMAC 忽略通道控制寄存器中 src_burst_len 域中的值。

- request_type = Burst
DMAC 执行 DMANOP。

 当 B 呈现时，汇编程序将 bs 设成 1。该指令在 request_type 标志位状态上是有条件的：

- request_type = Single
DMAC 执行 DMANOP。
- request_type = Burst
DMAC 使用突发 DMA 传递执行加载。

<peripheral> 5-bit 立即值 0-31。

 DMAC 执行 DMAWFP 指令时设置 request_type 标志位的值。请参考第 16 - 38 页“DMAWFP”。

操作
您只能在 DMA 通道线程中使用此命令。指令的执行在 request_type 状态（匹配该指令的状态）上是有条件的。

DMALP

Loop 指示 DMAC 将一个 8-bit 值加载到指定的循环计数寄存器 (loop counter register)。

该指令表明指令部分的开始和使用 DMALPEND 指令设置指令部分的结束。请参考第 16 - 31 页“DMALPEND[S | B]”。DMAC 重复在 DMALP 与 DMALPEND 之间插入的指令集，直到循环计数寄存器的值达到 0。

 DMAC 保存 DMALP 后面指令的 PC 值。DMAC 执行 DMALPEND 后，并且循环计数寄存器不是 0，这使 DMAC 能够执行循环中的第一个指令。

图 16 - 13 为指令编码。

图 16 - 13. DMALP 编码

15	8	7	6	5	4	3	2	1	0
iter[7:0]		0	0	1	0	0	0	lc	0

汇编语法

DMALP <loop_iterations>

其中：

<loop_iterations>

指定要执行的循环数，范围是 1-256。

- 汇编程序决定要使用的循环计数寄存器：
 - 将 lc 设成 0，DMAC 将 loop_iterations 减 1 的结果写入到循环计数器 0 寄存器
 - 或者，将 lc 设成 1，DMAC 将 loop_iterations 减 1 的结果写入到循环计数器 1 寄存器。

操作
您只能在 DMA 通道线程中使用此命令。

DMALPEND[S | B]

Loop End 表明程序循环中的最后指令，但是 DMAC 的行为取决于是否 DMALP 还是 DMALPFE 启动循环。如果一个循环开始于：

- DMALP，那么循环有一个定义的循环数，DMALPEND[S|B] 指示 DMAC 读取循环计数寄存器的值。如果循环计数寄存器返回：
 - Zero—DMAC 执行 DMANOP，从而退出循环。
 - Nonzero—DMAC 递减循环计数寄存器中的值，并更新线程 PC 以获得程序循环中的第一个指令的地址，也就是 DMALP 后面的指令。
- DMALPFE，那么循环有一个未定义的循环数，并且 DMAC 使用 request_last 标志位状态来控制何时退出循环。如果 request_last 标志位是：
 - 0—DMAC 更新线程 PC 以获得程序循环中的第一个指令的地址，也就是 DMALP 后面的指令。
 - 1—DMAC 执行 DMANOP，从而退出循环。

图 16 - 14 为指令编号。

图 16 - 14. DMALPEND[S|B] 编码

15	8	7	6	5	4	3	2	1	0
backwards_jump[7:0]		0	0	1	nf	1	lc	bs	x

汇编语法

DMALPEND[S|B]

其中：


[S] 如果呈现 S 并且循环开始于 DMALP，那么汇编程序将 bs 设成 0，x 设成 1。该指令在 request_type 标志位状态上是有条件的：

- request_type = Single
 - DMAC 执行 DMALPEND。
- request_type = Burst
 - DMAC 执行 DMANOP，从而退出循环。

[B] 如果呈现 B 并且循环开始于 DMALP，那么汇编程序将 bs 设成 1，x 设成 0。该指令在 request_type 标志位状态上是有条件的：

- request_type = Single
 - DMAC 执行 DMANOP，从而退出循环。
- request_type = Burst
 - DMAC 执行 DMALPEND。

如果不指定 S 或 B 操作数，那么汇编程序将 bs 设成 0，x 设成 0，并且 DMAC 始终执行 DMALPEND。

 当循环开始于 DMALPFE 时，一定不要指定 S 或 B 操作数，否则汇编程序会发出一条警告信息，并将 bs 设成 0，x 设成 0，nf 设成 1。同 DMALPFE 一样，DMAC 使用 request_last 标志位状态来控制何时退出循环。

 DMAC 设置以下值：

- 当 DMAC 执行 DMAWFP 指令时, 设置 request_type 标志位。请参考第 16 - 38 页 “DMAWFP”。
- 当相应外设通过外设请求接口发出最后一个请求命令时, 将 request_last 标志位设为 1。关于详细信息, 请参考第 16 - 10 页 “外设长度管理”。

要正确分配图 16 - 14 中所示的 DMALPEND 指令中的附加比特 (additional bits), 汇编程序要确定以下值:

backwards_jump[7:0] 设置程序循环中第一个指令的相关位置。汇编程序通过从 DMALPEND 的地址减去循环中第一个指令的地址, 计算出 backwards_jump[7:0] 的值。

- nf 将它设置为:
 - 0, 如果 DMALPFE 开始程序循环
 - 1, 如果 DMALP 开始程序循环。
- lc 将它设置为:
 - 0, 如果循环计数器 0 寄存器包含循环计数器值
 - 1, 如果循环计数器 1 寄存器包含循环计数器值
 - 1, 如果 DMALPFE 开始程序循环。

操作

您只能在 DMA 通道线程中使用此指令。如果指定 S 或 B 操作数, 那么指令的执行在 request_type 状态 (匹配该指令的状态) 上是有条件的。

DMALPFE

汇编程序使用 Loop Forever 来配置 DMALPEND 中的某些比特。请参考第 16 - 31 页 “DMALPEND[S | B]”。



当汇编程序遇到 DMALPFE 时, 它不对 DMAC 创建指令, 而是修改 DMALPEND 的行为。程序代码中 DMALPFE 的插入识别循环的开始。

汇编语法

DMALPFE

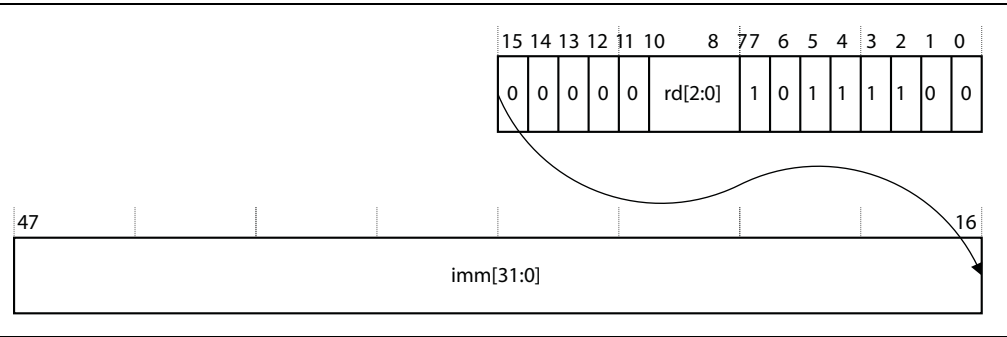
DMAMOV

Move 指示 DMAC 将 32-bit 立即值移到下面寄存器中:

- 源地址寄存器
- 目的地址寄存器
- 通道控制寄存器

图 16 - 15 显示指令编码。

图 16 - 15. DMAMOV 编码



汇编语法

DMAMOV <destination_register>, <32-bit_immediate>

其中：

<destination_register>

有效寄存器是：

- SAR— 选择源地址寄存器，并将 rd 设成 b000
- CCR— 选择通道控制寄存器，并将 rd 设成 b001
- DAR— 选择目的地址寄存器，并将 rd 设成 b010

<32-bit_immediate>

写入到指定目的寄存器中的 32-bit 值。

关于使用汇编程序编程通道控制寄存器各个域的详细信息，请参考第 16 - 41 页 “DMAMOV CCR”。

操作

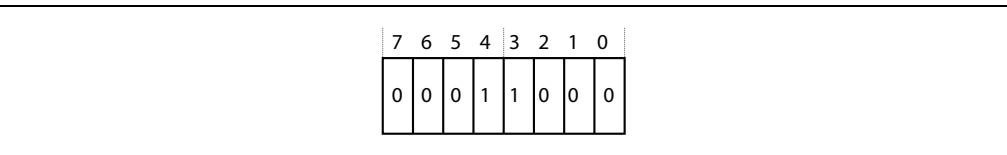
您只能在 DMA 通道线程中使用此命令。

DMANOP

此命令可用于代码对齐。

图 16 - 16 为指令编码。

图 16 - 16. DMANOP 编码



汇编语法

DMANOP

操作

此命令可用于 DMA 管理器线程和 DMA 通道线程。

DMARMB

Read Memory Barrier 强制 DMA 通道等待，直到此通道的全部执行的 DMALD 指令在 AXI 主接口上被发出并完成。

这安全地使能了对同一地址位置的读后写 (write-after-read) 序列。

图 16 - 17 为指令编码。

图 16 - 17. DMARMB 编码

7	6	5	4	3	2	1	0
0	0	0	1	0	0	1	0

汇编语法

DMARMB


操作

您只能在 DMA 通道线程中使用此命令。

DMASEV

Send Event 指示 DMAC 修改事件中断资源。根据您的如何编程中断使能寄存器，这会：

- 生成事件 `<event_num>`

 通常情况下使用 DMAWFE 停止线程，然后另一个线程执行 DMASEV，使用相应的事件编号恢复运行等待的线程。请参考第 16 - 13 页 “使用一个事件来重新开始 DMA 通道”。

- 使用 `irq<event_num>` 发送一个中断信号。

图 16 - 18 为指令编码。

图 16 - 18. DMASEV 编码


15	11	10	9	8	7	6	5	4	3	2	1	0
event_num[4:0]	0	0	0	0	0	1	1	0	1	0	0	

汇编语法

DMASEV `<event_num>`

其中：

`<event_num>` 5-bit 立即值 0-31

 如果选择一个无效的事件编码，DMAC 将终止线程。

操作

此命令可用于 DMA 管理器线程和 DMA 通道线程。关于详细信息，请参考第 16 - 13 页 “使用时间和中断”。

DMAST[S | B]

Store 指示 DMAC 使用 DA 寄存器和通道控制寄存器指定的 AXI 传输，从 FIFO 缓存到目的地址寄存器指定的位置传递数据。如果通道控制寄存器中的 dst_inc 比特设置成递增 (incrementing)，那么 DMAC 执行 DMAST[S|B] 之后更新目的地址寄存器。

图 16 - 19 为指令编码。

图 16 - 19. DMAST[S|B] 编码

7	6	5	4	3	2	1	0
0	0	0	0	1	0	bs	x

汇编语法

DMAST[S|B]

其中：

[S] 如果呈现 S，汇编程序将 bs 设成 0，x 设成 1。该指令在 request_type 标志位状态上是有条件的：

- request_type = Single
 - DMAC 执行 DMAST 指令，并设置 awlen[3:0]=0x0，这样 AXI 写传输长度为 1。DMAC 忽略通道控制寄存器中 dst_burst_len 域的值。
- request_type = Burst
 - DMAC 执行 DMANOP 指令。DMAC 递增通道 PC 到下一个指令。没有出现状态变化。

[B] 如果呈现 B，汇编程序将 bs 设成 1，x 设成 1。该指令在 request_type 标志位状态上是有条件的：

- request_type = Single
 - DMAC 执行 DMANOP 指令。DMAC 递增通道 PC 到下一个指令。没有出现状态变化。
- request_type = Burst
 - DMAC 执行 DMAST。
 - 如果不指定 S 或 B 操作数，那么汇编程序将 bs 设成 0，x 设成 0，并且 DMAC 始终执行 DMA 存储。



当 DMAC 执行 DMAWFP 指令时，它设置 request_type 标志位的值。请参考第 16 - 38 页 “DMAWFP”。

操作

您只能在 DMA 通道线程中使用此命令。如果指定 S 或 B 操作数，那么指令的执行在 request_type 状态（匹配该指令的状态）上是有条件的。

仅当 MFIFO 缓存包含用于完成突发传递的所有必要数据时，DMAC 才开始执行突发。

DMASTP<S | B>

Store and notify Peripheral 指示 DMAC 使用 DA 寄存器和通道控制寄存器指定的 AXI 传输，从 FIFO 缓存到目的地址寄存器指定的位置传递数据。它使用 DMA 通道编号来访问 FIFO 缓存中的相应位置。DMA 存储完成后，并且 DMAC 已经接收到缓存的写响应，DMA 发出一个数据传递完成的应答 (acknowledgement) 到外设。如果通道控制寄存器中的 `dst_inc` 比特设置成递增 (incrementing)，那么 DMAC 执行 `DMAST[S|B]` 之后更新目的地址寄存器。

图 16 - 20 为指令编码。

图 16 - 20. DMASTP<S|B> 编码

15	11	10	9	8	7	6	5	4	3	2	1	0
periph[4:0]					0	0	0	0	1	0	1	bs


汇编语法

DMASTP<S|B> <peripheral>

其中：

<S> 将 `bs` 设成 0，指示 DMAC 执行：

- 单一 DMA 存储操作，如果 `request_type` 被编程为 Single


 DMAC 忽略通道控制寄存器中 `dst_burst_len` 域的状态，并始终执行突发长度为 1 的 AXI 传递。

- `DMANOP`，如果 `request_type` 被编程为 Burst

 将 `bs` 设为 1，指示 DMAC 执行：

- DMA 存储，如果 `request_type` 被编程为 Burst
- `DMANOP`，如果 `request_type` 被编程为 Single。

<peripheral> 5-bit 立即值 0-31。

 当 DMAC 执行 `DMAWFP` 指令时，它设置 `request_type` 的值。请参考第 16 - 38 页“`DMAWFP`”。

操作

您只能在 DMA 通道线程中使用此命令。

仅当 MFIFO 缓存包含用于完成突发传递的所有必要数据时，DMAC 才开始执行突发。

DMASTZ

Store Zero 指示 DMAC 使用目的地址寄存器和通道控制寄存器指定的 AXI 传输来存储零。如果通道控制寄存器中的 `dst_inc` 比特设置成递增 (incrementing)，那么 DMAC 执行 `DMASTZ` 之后更新目的地址寄存器。

图 16 - 21 为指令编码。

图 16 - 21. DMASTZ 编码

7	6	5	4	3	2	1	0
0	0	0	0	1	1	0	0

汇编语法

DMASTZ

操作

您只能在 DMA 通道线程中使用此命令。

DMAWFE

Wait For Event 指示 DMAC 停止线程的执行，直到 *<event_num>* 指定的事件出现。事件出现时，线程移到 Executing 状态，DMAC 清除事件。请参考第 16 - 13 页 “使用时间和中断”。

图 16 - 22 为指令编码。

图 16 - 22. DMAWFE 编码

15	11	10	9	8	7	6	5	4	3	2	1	0
event_num[4:0]	0	i	0	0	0	1	1	0	1	1	0	


汇编语法

DMAWFE *<event_num>* [, invalid]

其中：

<event_num> 5-bit 立即值 0-31

[invalid] 将 i 设成 1。如果呈现 invalid，DMAC 使当前 DMA 线程的指令高速缓存无效。如果没有呈现 invalid，那么汇编程序将 i 设成 0，DMAC 不使当前 DMA 线程的指令高速缓存无效。

 如果选择的事件编码不适用于您的 DMAC 配置，那么 DMAC 将终止线程。

要确保高速缓存的一致性，当处理器对 DMA 通道写入指令流时必须使用 invalid。

操作

此命令可用于 DMA 管理器线程和 DMA 通道线程。

DMAWFP

Wait For Peripheral 指示 DMAC 停止线程的执行，直到指定的外设对该 DMA 通道发送一个 DMA 请求信号。

图 16 - 23 为指令编码。

图 16 - 23. DMAWFP 编码


15	11	10	9	8	7	6	5	4	3	2	1	0
peripheral[4:0]		0	0	0	0	0	1	1	0	0	bs	p

汇编语法

DMAWFP <peripheral>, <single|burst|periph>


其中：

<peripheral> 5-bit 立即值 0-31

 如果选择的外设编号不可用，DMAC 将终止线程。

<single>将bs设成0，p设成0，指示DMAC在接收到单一或者突发DMA请求后继续执行DMA通道线程。对于该 DMA 通道，DMAC 将 request_type 设成 Single。

<burst>将 bs 设成 1，p 设成 0，指示 DMAC 在接收到突发 DMA 请求后继续执行 DMA 通道线程。DMAC 将 request_type 设成 Burst。

 DMAC 忽略单一突发 DMA 请求。

<periph>将bs设成0，p设成0，指示DMAC在接收到单一或者突发DMA请求后继续执行DMA通道线程。DMAC 将 request_type 设成：

Single，当 DMA 接收到单一 DMA 请求时。

Burst，当 DMA 接收到突发 DMA 请求时。

操作

您只能在 DMA 通道线程中使用该指令。

DMAWMB

Write Memory Barrier 强制 DMA 通道等待，直到此通道的全部执行的 DMALD 指令在 AXI 主接口上被发出并完成。

这安全地使能了对同一地址位置的读后写 (write-after-read) 序列。

图 16 - 24 为指令编码。

图 16 - 24. DMAWMB 编码

7	6	5	4	3	2	1	0
0	0	0	1	0	0	1	1

汇编语法

DMAWMB

操作

您只能在 DMA 通道线程中使用此命令。

汇编指令

汇编程序提供以下额外命令：

- DCD
- DCB
- DMALP
- DMALPFE
- DMAMOV CCR

DCD

将一个 32-bit 立即值放置在指令流中的汇编指令。

语法

```
DCD imm32
```

DCB

将一个 8-bit 立即值放置在指令流中的汇编指令。

语法

```
DCB imm8
```

DMALP

插入一个迭代循环的汇编指令。

语法

```
DMALP [<LC0>|<LC1>] <loop_iterations>
```

其中：

<loop_iterations>

一个 8-bit 值，用于指定要执行的循环数。



为清晰起见，在写入汇编指令时，8-bit 值是要执行的实际循环迭代数。汇编程序逐一递减该值来生成 DMAC 使用的实际值 0-255。

[LC0] 如果呈现 LC0，那么 DMAC 将 <loop_iterations> 存储在循环计数器 0 寄存器中。

[LC1] 如果出现 LC1，那么 DMAC 将 <loop_iterations> 存储在循环计数器 1 寄存器中。



如果 LC0 或 LC1 没有呈现，那么汇编程序决定要使用的循环计数寄存器。

DMALPFE

插入重复循环的汇编指令。

语法

DMALPFE

使汇编程序能够清除呈现在 DMALPEND 中的 nf 比特。请参考第 16 - 31 页“DMALPEND[S | B]”。

DMAMOV CCR

汇编指令，使您能够使用指定的格式编程通道控制寄存器。

语法

DMAMOV CCR,
[SB<1-16>] [SS<8|16|32|64|128>] [SA<I|F>]
[SP<imm3>] [SC<imm4>]
[DB<1-16>] [DS<8|16|32|64|128>] [DA<I|F>]
[DP<imm3>] [DC<imm4>]
[ES<8|16|32|64|128>]

表 16 - 3 为自变量描述及默认值。

表 16 - 3. DMAMOV CCR 自变量描述及默认值

语法	描述	选项	默认
SA	源地址增量。设置 arburst[0] 的值。	I = Increment F = Fixed	I
SS	源突发大小 (bits)。设置 arsize[2:0] 的值。	8, 16, 32, or 64	8
SB	源突发长度。设置 arlen[3:0] 的值。	1 to 16	1
SP	源保护	0 to 7 (1)	0
SC	源高速缓存	0 to 15 (1) (2)	0
DA	目的地址增量。设置 awburst[0] 的值。	I = Increment F = Fixed	I
DS	目的突发大小 (bits)。设置 awsize[2:0] 的值。	8, 16, 32, or 64	8
DB	目的突发长度。设置 awlen[3:0] 的值。	1 to 16	1
DP	目的保护	0 to 7 (1)	0
DC	目的高速缓存	0 to 15 (1) (3)	0
ES	Endian 交换大小 (bits)	8, 16, 32, or 64	8
表 16 - 3 注释： (1) 当编程该立即值时，必须使用十进制值。 (2) 由于 DMAC 连接 ARCACHE[3] LOW，因此汇编程序始终将 bit 3 设成 0，并且将您选择的 bits [2:0] 值用于 SC。 (3) 由于 DMAC 连接 AWCACHE[2] LOW，因此汇编程序始终将 bit 2 设成 0，并且将您选择的 bit [3] 和 bits [1:0] 值用于 DC。			

MFIFO 缓存使用概况

本部分介绍了某些 DMA 通道程序的 MFIFO 缓存使用情况。

关于 MFIFO 缓存使用概况

MFIFO 缓存是一个共享资源,当前所有活动通道基于先来先得的原则使用该共享资源。对于一个程序而言, MFIFO 缓存是一组深度可调的并行 FIFO 缓存,每个通道一个,所有的 Fifies 总深度不能超过 512 的缓存深度。AXI 主接口的宽度与 MFIFO 缓存宽度相同。

DMAC 能够重新对齐源到目的的数据。例如,当 DMAC 从地址 0x103 读取一个字并写入到地址 0x205 时,DMAC 以两个字节移位数据。当数据进入 MFIFO 缓存时,作为 DMALD 指令的 AXI 读取操作的结果,会出现全部字节操作。因此,作为 DMAST 指令的 AXI 写操作的结果,当 DMAC 从 MFIFO 缓存中移除数据时,它不需要操作这些数据。从而 MFIFO 缓存中的数据存储和封装由目的地址和传输特性来决定。

当程序指定增量式传输将被执行到目的地,DMAC 将数据封装进 MFIFO 缓存,以最小化 MFIFO 缓存入口的使用。例如,当 DMAC 有一个 64-bit AXI 数据总线,并且程序使用一个 0x100 的源地址和 0x200 的目的地址时,DMAC 将两个 32-bit 字封装进 MFIFO 缓存中的一个入口。

在某些情况下,用于存储从一个源加载的数据所需要的入口数并不是 MFIFO 缓存宽度除以源数据量的简单计算。当出现下面任何情况,所需入口数的计算都不是简单的:

- 源地址没有对齐到 AXI 总线宽度。
- 目的地址没有对齐到 AXI 总线宽度。
- 传输到固定目的地,也就是非递增地址 (non-incrementing address)。

DMALD 和 DMAST 指令指定要执行的 AXI 传输。AXI 传输传递的数据量取决于编程到 CCRn 寄存器中的值以及传输地址。



关于未对齐传递的信息,请参考 ARM 网站 (infocenter.arm.com) 上的 *AMBA AXI Protocol Specification v1.0*。

下面部分提供了几个 DMAC 程序实例以及 MFIFO 缓存使用的演示。



此部分以下面方式显示了 MFIFO 缓存使用:

- MFIFO 缓存入口数量与时间关系图表
- 当数据进入 MFIFO 缓存时,DMAC 执行的字节通道操作图。



MFIFO 缓存图中的编号 0 和 7 表示 MFIFO 缓存中的字节通道。

对齐的传递

接下来的部分显示对齐的传递实例。

简单的对齐程序

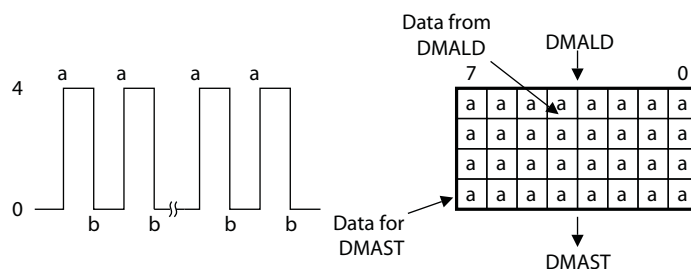
在此程序中，源地址与目的地址对齐于 AXI 数据总线宽度。

例 16 - 2. 简单对齐程序

```
DMAMOV CCR, SB4 SS64 DB4 DS64
DMAMOV SAR, 0x1000
DMAMOV DAR, 0x4000
DMALP 16
DMALD ; 如图 16 - 25 中 a 所示
DMAST ; 如图 16 - 25 中 b 所示
DMALPEND
DMAEND
```

图 16 - 25 显示了我程序中的 MFIFO 缓存使用。

图 16 - 25. 简单对齐程序



在图 16 - 25 中，每个 DMALD 需要四个入口，每个 DMAST 删除四个入口。

此实例有一个零 MFIFO 缓存入口的静态要求和四个 MFIFO 缓存入口的动态要求。

多个加载的对齐非对称程序

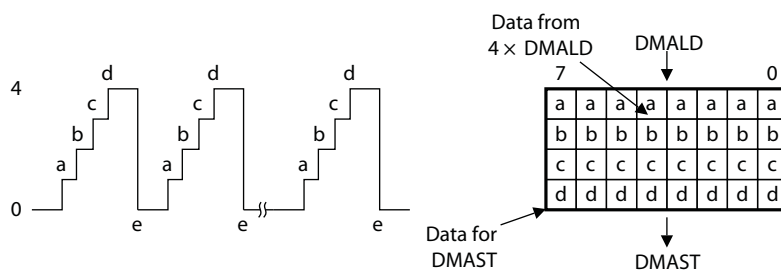
下面程序对每次存储执行四个加载，源地址和目的地址与 AXI 数据总线宽度对齐。

例 16 - 3. 多个加载的对齐非对称程序

```
DMAMOV CCR, SB1 SS64 DB4 DS64
DMAMOV SAR, 0x1000
DMAMOV DAR, 0x4000
DMALP 16
DMALD ; 如图 16 - 26 中的 a 所示
DMALD ; 如图 16 - 26 中的 b 所示
DMALD ; 如图 16 - 26 中的 c 所示
DMALD ; 如图 16 - 26 中的 d 所示
DMAST ; 如图 16 - 26 中的 e 所示
DMALPEND
DMAEND
```

图 16-26 显示了这个程序中的 MFIFO 缓存使用。

图 16-26. 多个加载的对齐非对称程序



在图 16-26 中, 每个 DMALD 需要一个入口, 每个 DMAST 删除四个入口。

此实例有一个零 MFIFO 缓存入口的静态要求和四个 MFIFO 缓存入口的动态要求。

多个存储的对齐非对称程序

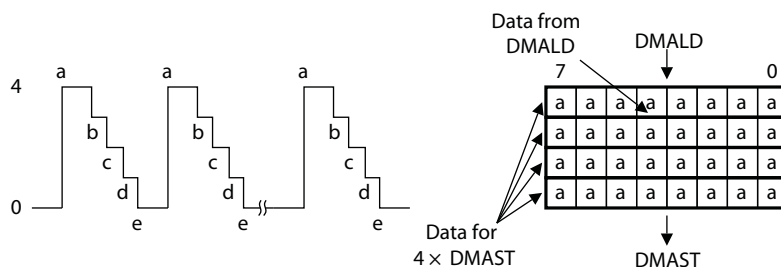
下面程序对每次加载执行四个存储, 源地址和目的地址与 AXI 数据总线宽度对齐。

例 16-4. 多个存储的对齐非对称程序

```
DMAMOV CCR, SB4 SS64 DB1 DS64
DMAMOV SAR, 0x1000
DMAMOV DAR, 0x4000
DMALP 16
DMALD ; 如图 16-27 中的 a 所示
DMAST ; 如图 16-27 中的 b 所示
DMAST ; 如图 16-27 中的 c 所示
DMAST ; 如图 16-27 中的 d 所示
DMAST ; 如图 16-27 中的 e 所示
DMALPEND
DMAEND
```

图 16-27 显示了这个程序中的 MFIFO 缓存使用。

图 16-27. 多个存储的对齐非对称程序



在图 16-27 中, 每个 DMALD 需要四个入口, 每个 DMAST 删除一个入口。

此实例有一个零 MFIFO 缓存入口的静态要求和四个 MFIFO 缓存入口的动态要求。

未对齐的传递

下面部分显示了未对齐传递的实例。

对齐的源地址到未对齐的目的地址

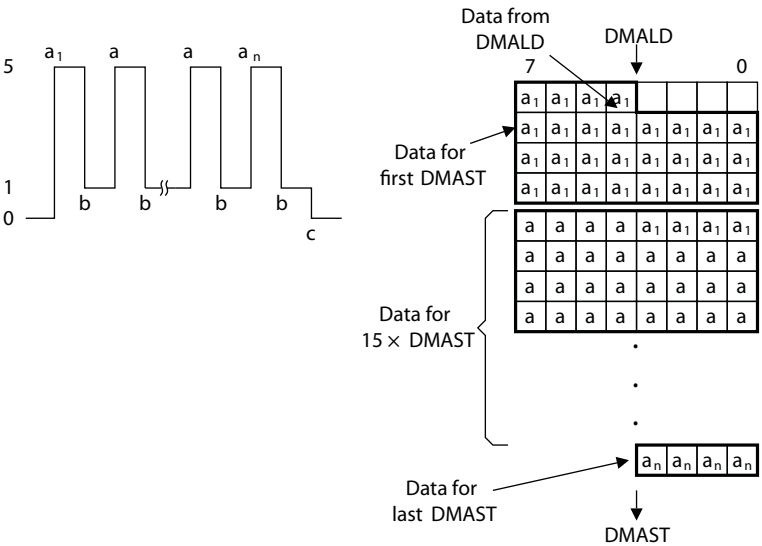
在此程序中，源地址对齐于 AXI 数据总线宽度，但目的地址是未对齐的。目的地址没有对齐于目的突发大小，所以第一个 DMAST 指令移除的数据少于第一个 DMALD 指令读取的数据。因此，单一字的最后 DMAST 被需要用于清除 MFIFO 缓存中的数据。

例 16 - 5. 对齐源地址到未对齐目的地址

```
DMAMOV CCR, SB4 SS64 DB4 DS64
DMAMOV SAR, 0x1000
DMAMOV DAR, 0x4004
DMALP 16
DMALD ; 图 16 - 28 中显示的 a1, ... a, an
DMAST ; 图 16 - 28 中显示的 b
DMALPEND
DMAMOV CCR, SB4 SS64 DB1 DS32
DMAST ; 图 16 - 28 中显示的 c
DMAEND
```

图 16 - 28 显示了此程序中的 MFIFO 缓存使用。

图 16 - 28. 对齐到未对齐程序



第一个 DMALD 指令加载 4 个双字，但由于目的地址是未对齐的，因此 DMAC 将它们移位 4 个字节，所以它在 MFIFO 缓存中使用 5 个入口。

每个 DMAST 只需要数据的 4 个入口，因此其余入口用于编程期间，直到被最后一个 DMAST 清空。

此实例有一个 MFIFO 缓存入口的静态要求和四个 MFIFO 缓存入口的动态要求。

过量初始加载的未对齐源地址到对齐的目的地址

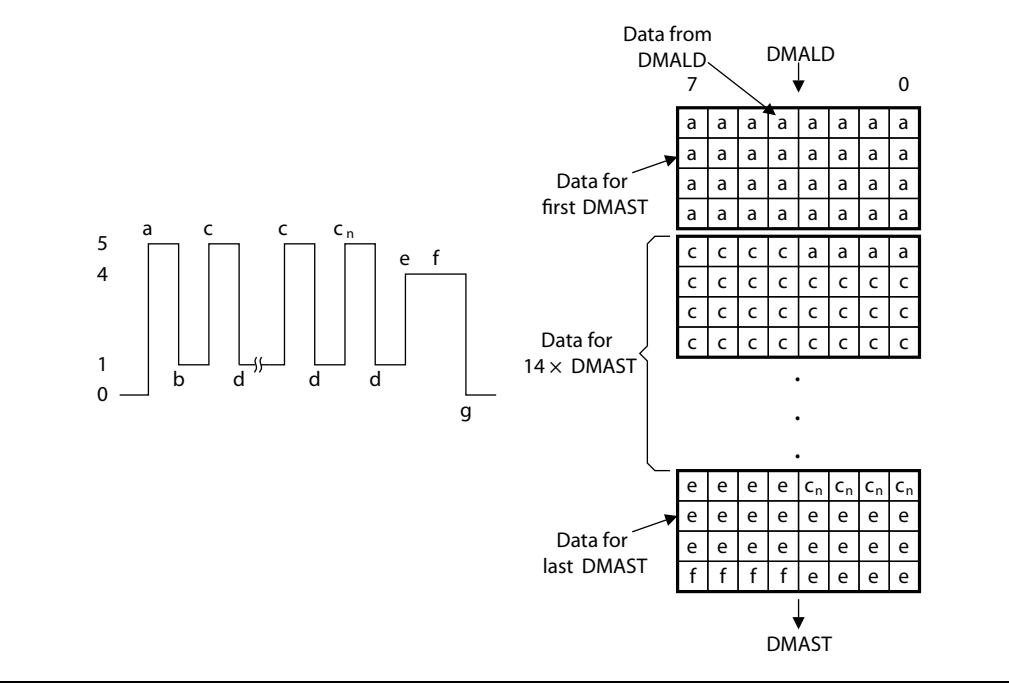
此程序是第 16 - 46 页 “未对齐的源地址和对齐的目的地址” 中所描述的程序的替代程序。此程序使用不同序列的源突发，可能不是很高效，但需要更少的 MFIFO 缓存入口。


例 16 - 7. 过量初始加载的未对齐源地址到对齐的目的地址

```
DMAMOV CCR, SB5 SS64 DB4 DS64
DMALD SAR, 0x1004
DMAMOV DAR, 0x4000
DMALD ; 图 16 - 30 中 a 所示
DMAST ; 图 16 - 30 中 b 所示
DMAMOV CCR, SB4 SS64 DB4 DS64
DMALP 14
DMALD ; 图 16 - 30 中 c 和 cn 所示
DMAST ; 图 16 - 30 中 d 所示
DMAEND
```

图 16 - 30 为此程序的 MFIFO 缓存使用。

图 16 - 30. 过量初始加载的未对齐到对齐



 显示为 **f** 的 DMALD 不增加 MFIFO 缓存使用，因为它将 DMAC 已经分配给此通道的 4 个字节加载到 MFIFO 缓存入口。

第一个 DMALD 指令加载 5 个差拍 (beats) 的数据，使 DMAC 能够执行第一个 DMAST。

第一个 DMALD 之后，接下来的 DMALDs 没有与源突发大小对齐，例如地址 0x1028 上的第二个 DMALD 读操作。循环后，最后两个 DMALDs 读取用于满足最后 DMAST 所需要的数据。

此实例有 1 个 MFIFO 缓存入口的静态要求和 4 个 MFIFO 缓存入口的动态要求。

对齐的突发大小未对齐的 MFIFO 缓存

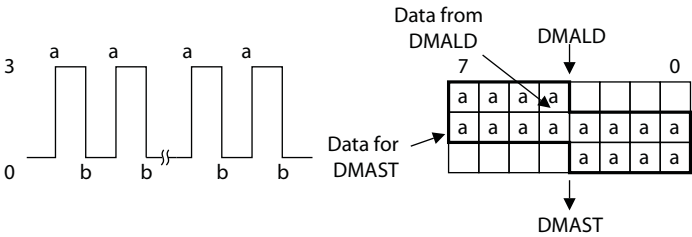
在此程序中，目的地址（窄于 MFIFO 缓存宽度）对齐于突发大小，但不对齐于 MFIFO 缓存宽度。

例 16 - 8. 对齐的突发大小未对齐的 MFIFO 缓存

```
DMAMOV CCR, SB4 SS32 DB4 DS32
DMAMOV SAR, 0x1000
DMAMOV DAR, 0x4004
DMALP 16
DMALD ; 图 16 - 31 中 a 所示
DMAST ; 图 16 - 31 中 b 所示
DMALPEND
DMAEND
```

图 16 - 31 为此程序的 MFIFO 缓存使用。

图 16 - 31. 带有未对齐 MFIFO 缓存宽度的对齐突发



在此实例中，目的地址不是 64-bit 对齐的，它需要 3 个 MFIFO 缓存入口，而不是预期的 2 个 MFIFO 缓存入口。

此实例有零 MFIFO 缓存入口的静态要求和 3 个 MFIFO 缓存入口的动态要求。

固定传递

下面部分显示了一个具有对齐地址的固定目的地的实例。

具有对齐地址的固定目的地

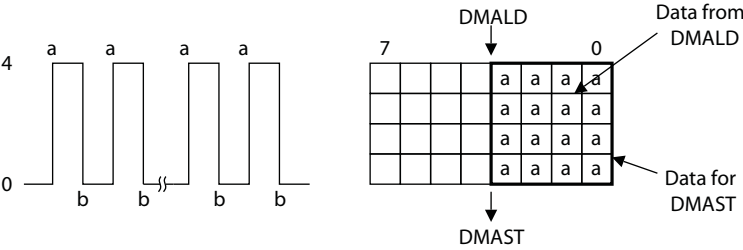
在此程序中，源地址和目的地址与 AXI 数据执行宽度对齐，目的地址是固定的。

例 16 - 9. 具有对齐地址的固定目的地

```
DMAMOV CCR, SB2 SS64 DB4 DS32 DAF
DMAMOV SAR, 0x1000
DMAMOV DAR, 0x4000
DMALP 16
DMALD ; 图 16 - 32 中 a 所示
DMAST ; 图 16 - 32 中 b 所示
DMALPEND
DMAEND
```

图 16 - 32 为此程序的 MFIFO 缓存使用。

图 16 - 32. 具有对齐地址的固定目的地

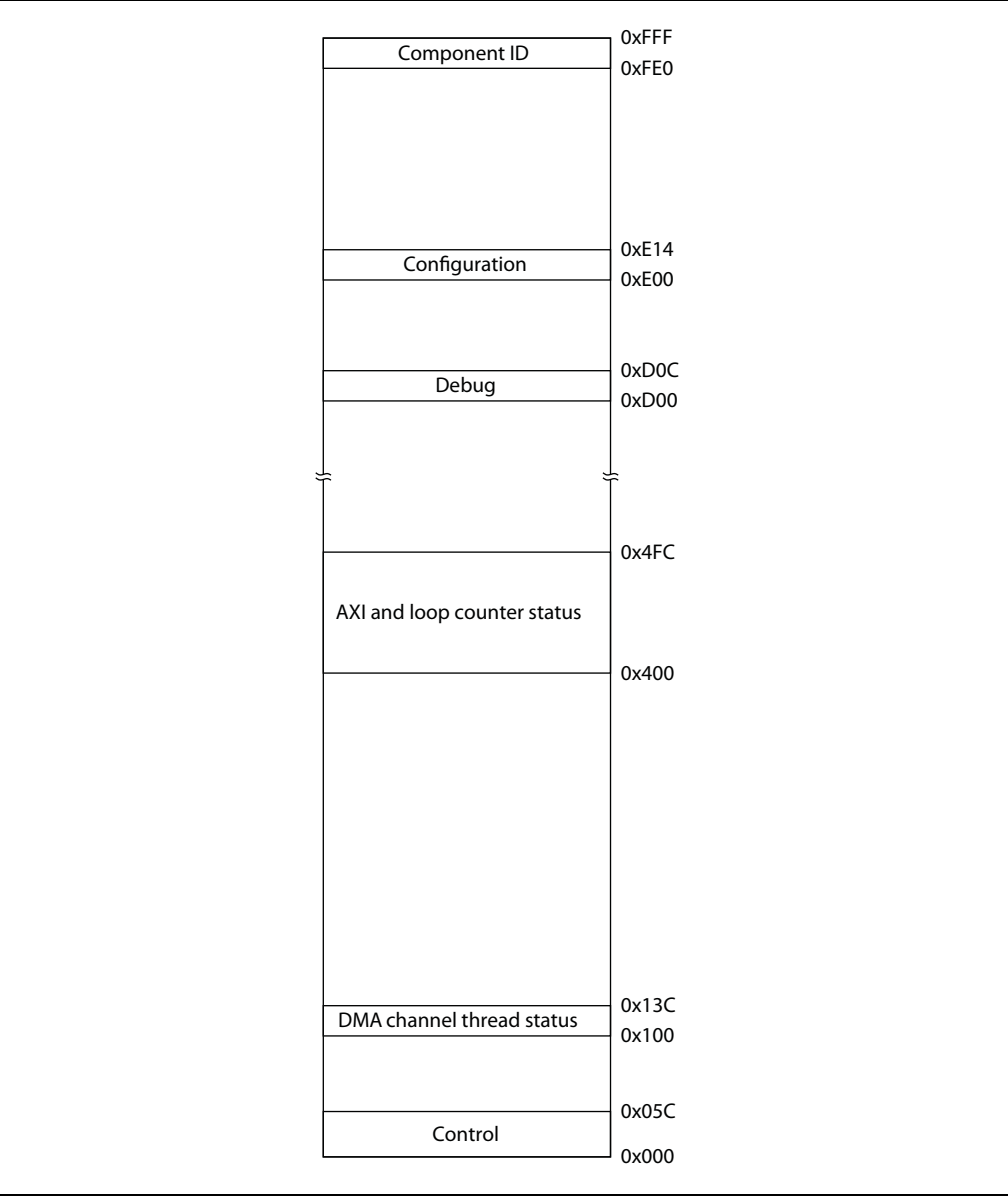


程序中的每个 DMALD 加载 2 个 64-bit 数据传递到 MFIFO 缓存。由于目的地址是一个 32-bit 固定地址，所以 DMAC 跨 MFIFO 缓存中的两个入口来分离每个 64-bit 数据项。此实例有零 MFIFO 缓存入口的静态要求和 4 个 MFIFO 缓存入口的动态要求。

DMA 控制寄存器

图 16 - 33 显示了跨越 4 KB 区域的 DMAC 的寄存器映射。

图 16 - 33. DMAC 寄存器映射




在图 16 - 33 中，寄存器映射包含下面部分。

- 控制寄存器 — 用于控制 DMAC。
- DMA 通道线程状态寄存器 — 提供 DMA 通道线程的状态。
- AXI 和循环计数器状态寄存器 — 提供每个 DMA 通道线程的 AXI 传递状态和循环接收器状态。

- 调试寄存器 — 使能以下功能：
 - 当调试程序代码时，支持发送指令到线程。
 - 支持系统固件发送指令到 DMA 管理器线程，如第 16 – 8 页 “使用从接口发出指令到 DMAC” 所描述。
- 配置寄存器 — 使系统固件能够发现 DMAC 的配置并控制看门狗的行为。
- 组件 ID 寄存器 — 使系统固件能够识别外设。不要尝试访问保留的或未使用的地址位置，否则将导致不可预测的行为。


地址映射与寄存器定义

 地址映射和寄存器定义位于本卷中的 [hps.html](#) 文件中。点击下面链接打开文件。

要查看模块定义和基地址，点击下面模块实例的链接：

- `dmanonsecure`
- `dmasecure`

然后，点击寄存器名拉查看寄存器和域描述。寄存器地址是相对于每个模块实例基地址的偏移。

 所有模块的基地址也列于 *Cyclone V 器件手册* 卷 3 中的 *Introduction to the Hard Processor System* 章节中。

文档修订历史

表 16 – 4 显示了本文档的修订历史。

表 16 – 4. 文档修订历史

日期	版本	修订内容
2012 年 11 月	1.1	次要更新。
2012 年 1 月	1.0	首次发布。

