第 22 章 直接存储器访问 （DMA）

目录

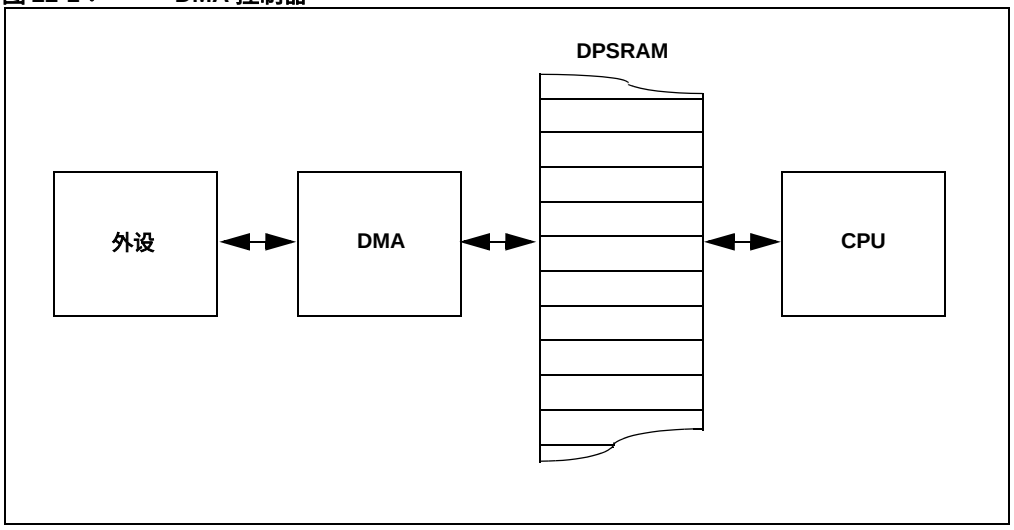
本章包括下列主题：

22.1 简介

直接存储器访问 （Direct Memory Access， DMA）控制器是 Microchip 的高性能 16 位数字信号控制器（Digital Signal Controller，DSC）系列中很重要的子系统。通过该子系统，无需 CPU 协助即可在 CPU 及其外设之间方便地传输数据。 dsPIC33F DMA 控制器针对高性能实时嵌入式应用进行了优化，这些应用要求优先考虑确定性和系统响应延时。

DMA 控制器在外设数据寄存器和数据空间 SRAM 之间传输数据。 dsPIC33F DMA 子系统使用双端口 SRAM 存储器 （DPSRAM）和寄存器结构，这使 DMA 可以通过自己独立的地址和数据总线进行操作，而不会影响 CPU 操作。这种架构无需进行周期挪用，在周期挪用情况下，当有优先级更高的 DMA 传输请求时会使 CPU 暂停。 CPU 和 DMA 控制器都可以读写数据空间中的地址，而不会造成干扰 （例如 CPU 暂停），这可以获得最大的实时性能。或者说， CPU 处理不会影响存储器与外设之间的 DMA 操作和数据传输。例如，执行运行时自编程 （Run-Time Self-Programming， RTSP）操作时，在 RTSP 完成之前， CPU 不会执行任何指令。但这种情况不会影响存储器和外设之间的数据传输。

图 22-1： DMA 控制器



DMA 控制器支持 8 个独立通道。每个通道都可以配置为向选定外设发送数据或从选定外设接收数据。 DMA 控制器支持的外设包括：

- ECAN™ 技术
- 数据转换器接口 （Data Converter Interface， DCI）
- 10 位 /12 位模数转换器 （Analog-to-Digital Converter， ADC）
- 串行外设接口 （Serial Peripheral Interface， SPI）
- UART
- 输入捕捉
- 输出比较

此外，DMA 传输可以通过定时器和外部中断进行触发。每个 DMA 通道都是单向的。要对某个外设进行读**和**写操作，必须分配两个 DMA 通道。如果有多个通道接收到数据传输请求，则基于通道编号的简单固定优先级机制会指定哪个通道完成传输，哪个或哪些通道保持等待状态。每个 DMA 通道会传送最多可包含 1024 个数据元素的数据块，之后向 CPU 发出中断，指示数据块已可进行处理。

DMA 控制器具有以下功能：

- 8 个 DMA 通道
- 带后递增的寄存器间接寻址模式
- 不带后递增的寄存器间接寻址模式
- 外设间接寻址模式 （外设生成目标地址）
- 在传输完一半或整个数据块后发出中断给 CPU
- 字节或字传输
- 固定优先级通道仲裁
- 手动 （软件）或自动 （外设 DMA 请求）启动传输
- 单数据块或自动重复数据块传输模式
- “ 乒乓 ”（Ping-Pong）模式 （每个数据块传输完成后，在两个 DPSRAM 起始地址之间进行自动切换）
- 每个通道的 DMA 请求可以从任何支持的中断源选择
- 调试支持功能

22.2 DMA 寄存器

每个 DMA 通道都具有一组共 6 个状态和控制寄存器。

- **DMAxCON：DMA 通道 x 控制寄存器**

  该寄存器用于配置相应的 DMA 通道：使能 / 禁止通道，指定数据传输大小、方向和数据块中断方法，以及选择 DMA 通道寻址模式、工作模式和空数据写模式。

- **DMAxREQ：DMA 通道 x IRQ 选择寄存器**

  该寄存器用于通过为 DMA 通道分配外设 IRQ，将 DMA 通道与支持 DMA 的特定外设关联。

- **DMAxSTA：DMA 通道 x DPSRAM 起始地址偏移寄存器 A**

  该寄存器用于指定主起始地址偏移，该偏移相对于要通过 DMA 通道 x 传输到 DPSRAM 或从 DPSRAM 传输的数据块的 DMA DPSRAM 基址。读该寄存器将返回最新的 DPSRAM 传输地址偏移的值。如果使能了通道 x （即通道处于工作状态），写入该寄存器可能导致不可预测的行为，应该避免。

- **DMAxSTB：DMA 通道 x DPSRAM 起始地址偏移寄存器 B**

  该寄存器用于指定辅助起始地址偏移，该偏移相对于要通过 DMA 通道 x 传输到 DPSRAM 或从 DPSRAM 传输的数据块的 DMA DPSRAM 基址。读该寄存器将返回最新的 DPSRAM 传输地址偏移的值。如果使能了通道 x （即通道处于工作状态），写入该寄存器可能导致不可预测的行为，应该避免。

- **DMAxPAD：DMA 通道 x 外设地址寄存器**

  该读 / 写寄存器包含外设数据寄存器的静态地址。如果使能了相应的 DMA 通道 （即通道处于工作状态），写入该寄存器可能导致不可预测的行为，应该避免。

- **DMAxCNT：DMA 通道 x 传输计数寄存器**

  该寄存器包含传输计数。DMAxCNT + 1 代表通道必须处理的 DMA 请求数，处理完该数量的请求之后，数据块传输才视为完成。即，DMAxCNT 值为 0 时，将传输一个数据元素。DMAxCNT 寄存器的值与传输数据大小（DMAxCON 寄存器中的 SIZE 位）无关。如果使能了相应的 DMA 通道 （即通道处于工作状态），写入该寄存器可能导致不可预测的行为，应该避免。

除了各个 DMA 通道的寄存器之外， DMA 控制器还有三个 DMA 状态寄存器。

- **DSADR：最近的 DMA DPSRAM 地址寄存器**

  该寄存器是 16 位只读状态寄存器，由所有 DMA 通道共用。它捕捉最近一次 DPSRAM 访问（读或写）的地址。它在复位时清零，因此，如果在任何 DMA 活动之前读取它，它包含的值为 0x0000。任意时候都可以访问该寄存器，但该寄存器主要用于协助调试。

- **DMACS0：DMA 控制器状态寄存器 0**

  该寄存器是 16 位只读状态寄存器，包含 DPSRAM 和外设写冲突标志 （分别为 XWCOLx 和 PWCOLx）。更多详细信息，请参见**第 22.10 节 “ 数据写冲突 ”**。

- **DMACS1：DMA 控制器状态寄存器 1**

  该寄存器是 16 位只读状态寄存器，它指示哪个 DMA 通道最近处于工作状态，并通过指示哪个 DPSRAM 起始地址偏移寄存器 （DMAxSTA 或 DMAxSTB）被选定来提供每个 DMA 通道的乒乓模式状态。

寄存器 22-1： DMAxCON：DMA 通道 x 控制寄存器

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	U-0	U-0	U-0
CHEN	SIZE	DIR	HALF	NULLW	—	—	—
bit 15							bit 8

U-0	U-0	R/W-0	R/W-0	U-0	U-0	R/W-0	R/W-0
—	—	AMODE<1:0>		—	—	MODE<1:0>	
bit 7							bit 0

图注：			
R = 可读位	W = 可写位	U = 未实现位，读为 0	
-n = POR 时的值	1 = 置 1	0 = 清零	x = 未知

bit 15 **CHEN：**通道使能位
1 = 使能通道
0 = 禁止通道

bit 14 **SIZE：**数据传输长度位
1 = 字节
0 = 字

bit 13 **DIR：**传输方向位 （源 / 目标总线选择）
1 = 从 DPSRAM 地址读取，写入外设地址
0 = 从外设地址读取，写入 DPSRAM 地址

bit 12 **HALF：**数据块传输中断选择位
1 = 当传送了一半数据时，发出中断
0 = 当传送了所有数据时，发出中断

bit 11 **NULLW：**空数据外设写模式选择位
1 = 除将外设地址中的数据写入 DPSRAM 外，还将空数据写入外设地址 （DIR 位也必须清零）
0 = 正常工作

bit 10-6 **未实现：**读为 0

bit 5-4 **AMODE<1:0>：**DMA 通道寻址模式选择位
11 = 保留
10 = 外设间接寻址模式
01 = 不带后递增的寄存器间接寻址模式
00 = 带后递增的寄存器间接寻址模式

bit 3-2 **未实现：**读为 0

bit 1-0 **MODE<1:0>：**DMA 通道工作模式选择位
11 = 使能单数据块乒乓模式 （与每个 DMA RAM 缓冲区之间传输一块数据）
10 = 使能连续数据块乒乓模式
01 = 禁止单数据块乒乓模式
00 = 禁止连续数据块乒乓模式

寄存器 22-2： DMAxREQ：DMA 通道 x IRQ 选择寄存器

R/S-0	U-0	U-0	U-0	U-0	U-0	U-0	U-0
FORCE(1)	—	—	—	—	—	—	—
bit 15							bit 8

U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	IRQSEL<6:0>						
bit 7							bit 0

图注：			
R = 可读位	W = 可写位	U = 未实现位，读为 0	
-n = POR 时的值	1 = 置 1	0 = 清零	x = 未知

bit 15 **FORCE：**强制 DMA 传输位 (1)
1 = 强制进行单次 DMA 传输 （手动模式）
0 = 自动按照 DMA 请求启动 DMA 传输

bit 14-7 **未实现：**读为 0

bit 6-0 **IRQSEL<6:0>：**DMA 外设 IRQ 编号选择位
0000000 = INT0——外部中断 0
0000001 = IC1——输入捕捉 1
0000010 = OC1——输出比较 1
0000101 = IC2——输入捕捉 2
0000110 = OC2——输出比较 2
0000111 = TMR2——Timer2
0001000 = TMR3——Timer3
0001010 = SPI1——传输完成
0001011 = UART1RX——UART1 接收器
0001100 = UART1TX——UART1 发送器
0001101 = ADC1——ADC1 转换完成
0010101 = ADC2——ADC2 转换完成
0011110 = UART2RX——UART2 接收器
0011111 = UART2TX——UART2 发送器
0100001 = SPI2——传输完成
0100010 = ECAN1——接收数据就绪
0110111 = ECAN2——接收数据就绪
0111100 = DCI——编解码器传输完成
1000110 = ECAN1——发送数据请求
1000111 = ECAN2——发送数据请求

注 1：FORCE 位不能被用户清零。当强制的 DMA 传输完成时， FORCE 位由硬件清零。

寄存器 22-3： DMAxSTA：DMA 通道 x DPSRAM 起始地址偏移寄存器 A

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
STA<15:8>							
bit 15							bit 8

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
STA<7:0>							
bit 7							bit 0

图注：			
R = 可读位	W = 可写位	U = 未实现位，读为 0	
-n = POR 时的值	1 = 置 1	0 = 清零	x = 未知

bit 15-0 **STA<15:0>：**主 DPSRAM 起始地址偏移位 （源地址或目标地址）

寄存器 22-4： DMAxSTB：DMA 通道 x DPSRAM 起始地址偏移寄存器 B

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
STB<15:8>							
bit 15							bit 8

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
STB<7:0>							
bit 7							bit 0

图注：			
R = 可读位	W = 可写位	U = 未实现位，读为 0	
-n = POR 时的值	1 = 置 1	0 = 清零	x = 未知

bit 15-0 **STB<15:0>：**辅助 DPSRAM 起始地址偏移位 （源地址或目标地址）

寄存器 22-5： DMAxPAD：DMA 通道 x 外设地址寄存器

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
PAD<15:8>							
bit 15							bit 8

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
PAD<7:0>							
bit 7							bit 0

图注：			
R = 可读位	W = 可写位	U = 未实现位，读为 0	
-n = POR 时的值	1 = 置 1	0 = 清零	x = 未知

bit 15-0 **PAD<15:0>：**外设地址寄存器位

寄存器 22-6： DMAxCNT：DMA 通道 x 传输计数寄存器

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	—	—	—	—	CNT<9:8>	
bit 15							bit 8

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
CNT<7:0>							
bit 7							bit 0

图注：			
R = 可读位	W = 可写位	U = 未实现位，读为 0	
-n = POR 时的值	1 = 置 1	0 = 清零	x = 未知

bit 15-10 保留

bit 9-0 **CNT<9:0>：**DMA 传输计数寄存器位

寄存器 22-7： DSADR：最近的 DMA DPSRAM 地址寄存器

R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0
DSADR<15:8>							
bit 15							bit 8

R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0
DSADR<7:0>							
bit 7							bit 0

图注：			
R = 可读位	W = 可写位	U = 未实现位，读为 0	
-n = POR 时的值	1 = 置 1	0 = 清零	x = 未知

bit 15-0 **DSADR<15:0>：**DMA 最近访问的 DMA DPSRAM 地址位

寄存器 22-8： DMACS0：DMA 控制器状态寄存器 0

R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0
PWCOL7	PWCOL6	PWCOL5	PWCOL4	PWCOL3	PWCOL2	PWCOL1	PWCOL0
bit 15							bit 8

R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0
XWCOL7	XWCOL6	XWCOL5	XWCOL4	XWCOL3	XWCOL2	XWCOL1	XWCOL0
bit 7							bit 0

图注：			
R = 可读位	W = 可写位	U = 未实现位，读为 0	
-n = POR 时的值	1 = 置 1	0 = 清零	x = 未知

bit 15 **PWCOL7：**通道 7 外设写冲突标志位
1 = 检测到写冲突
0 = 未检测到写冲突

bit 14 **PWCOL6：**通道 6 外设写冲突标志位
1 = 检测到写冲突
0 = 未检测到写冲突

bit 13 **PWCOL5：**通道 5 外设写冲突标志位
1 = 检测到写冲突
0 = 未检测到写冲突

bit 12 **PWCOL4：**通道 4 外设写冲突标志位
1 = 检测到写冲突
0 = 未检测到写冲突

bit 11 **PWCOL3：**通道 3 外设写冲突标志位
1 = 检测到写冲突
0 = 未检测到写冲突

bit 10 **PWCOL2：**通道 2 外设写冲突标志位
1 = 检测到写冲突
0 = 未检测到写冲突

bit 9 **PWCOL1：**通道 1 外设写冲突标志位
1 = 检测到写冲突
0 = 未检测到写冲突

bit 8 **PWCOL0：**通道 0 外设写冲突标志位
1 = 检测到写冲突
0 = 未检测到写冲突

bit 7 **XWCOL7：**通道 7 DPSRAM 写冲突标志位
1 = 检测到写冲突
0 = 未检测到写冲突

bit 6 **XWCOL6：**通道 6 DPSRAM 写冲突标志位
1 = 检测到写冲突
0 = 未检测到写冲突

bit 5 **XWCOL5：**通道 5 DPSRAM 写冲突标志位
1 = 检测到写冲突
0 = 未检测到写冲突

bit 4 **XWCOL4：**通道 4 DPSRAM 写冲突标志位
1 = 检测到写冲突
0 = 未检测到写冲突

bit 3 **XWCOL3：**通道 3 DPSRAM 写冲突标志位
1 = 检测到写冲突
0 = 未检测到写冲突

寄存器 22-8： DMACS0：DMA 控制器状态寄存器 0 （续）

bit 2 **XWCOL2：**通道 2 DPSRAM 写冲突标志位
1 = 检测到写冲突
0 = 未检测到写冲突

bit 1 **XWCOL1：**通道 1 DPSRAM 写冲突标志位
1 = 检测到写冲突
0 = 未检测到写冲突

bit 0 **XWCOL0：**通道 0 DPSRAM 写冲突标志位
1 = 检测到写冲突
0 = 未检测到写冲突

寄存器 22-9： DMACS1：DMA 控制器状态寄存器 1

U-0	U-0	U-0	U-0	R-1	R-1	R-1	R-1
—	—	—	—	LSTCH<3:0>			
bit 15							bit 8

R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0
PPST7	PPST6	PPST5	PPST4	PPST3	PPST2	PPST1	PPST0
bit 7							bit 0

图注：			
R = 可读位	W = 可写位	U = 未实现位，读为 0	
-n = POR 时的值	1 = 置 1	0 = 清零	x = 未知

bit 15-12 **未实现：**读为 0

bit 11-8 **LSTCH<3:0>：**上一次工作的 DMAC 通道位
1111 = 自系统复位以来没有发生 DMA 传输
1110-1000 = 保留
0111 = 上次数据传输是通过通道 7 进行的
0110 = 上次数据传输是通过通道 6 进行的
0101 = 上次数据传输是通过通道 5 进行的
0100 = 上次数据传输是通过通道 4 进行的
0011 = 上次数据传输是通过通道 3 进行的
0010 = 上次数据传输是通过通道 2 进行的
0001 = 上次数据传输是通过通道 1 进行的
0000 = 上次数据传输是通过通道 0 进行的
复位时设置为 1111。任意时候都可以访问该字段，但它主要用于协助调试。

bit 7 **PPST7：**通道 7 乒乓模式状态标志位
1 = 选择 DMA7STB 寄存器
0 = 选择 DMA7STA 寄存器

bit 6 **PPST6：**通道 6 乒乓模式状态标志位
1 = 选择 DMA6STB 寄存器
0 = 选择 DMA6STA 寄存器

bit 5 **PPST5：**通道 5 乒乓模式状态标志位
1 = 选择 DMA5STB 寄存器
0 = 选择 DMA5STA 寄存器

bit 4 **PPST4：**通道 4 乒乓模式状态标志位
1 = 选择 DMA4STB 寄存器
0 = 选择 DMA4STA 寄存器

bit 3 **PPST3：**通道 3 乒乓模式状态标志位
1 = 选择 DMA3STB 寄存器
0 = 选择 DMA3STA 寄存器

bit 2 **PPST2：**通道 2 乒乓模式状态标志位
1 = 选择 DMA2STB 寄存器
0 = 选择 DMA2STA 寄存器

bit 1 **PPST1：**通道 1 乒乓模式状态标志位
1 = 选择 DMA1STB 寄存器
0 = 选择 DMA1STA 寄存器

bit 0 **PPST0：**通道 0 乒乓模式状态标志位
1 = 选择 DMA0STB 寄存器
0 = 选择 DMA0STA 寄存器

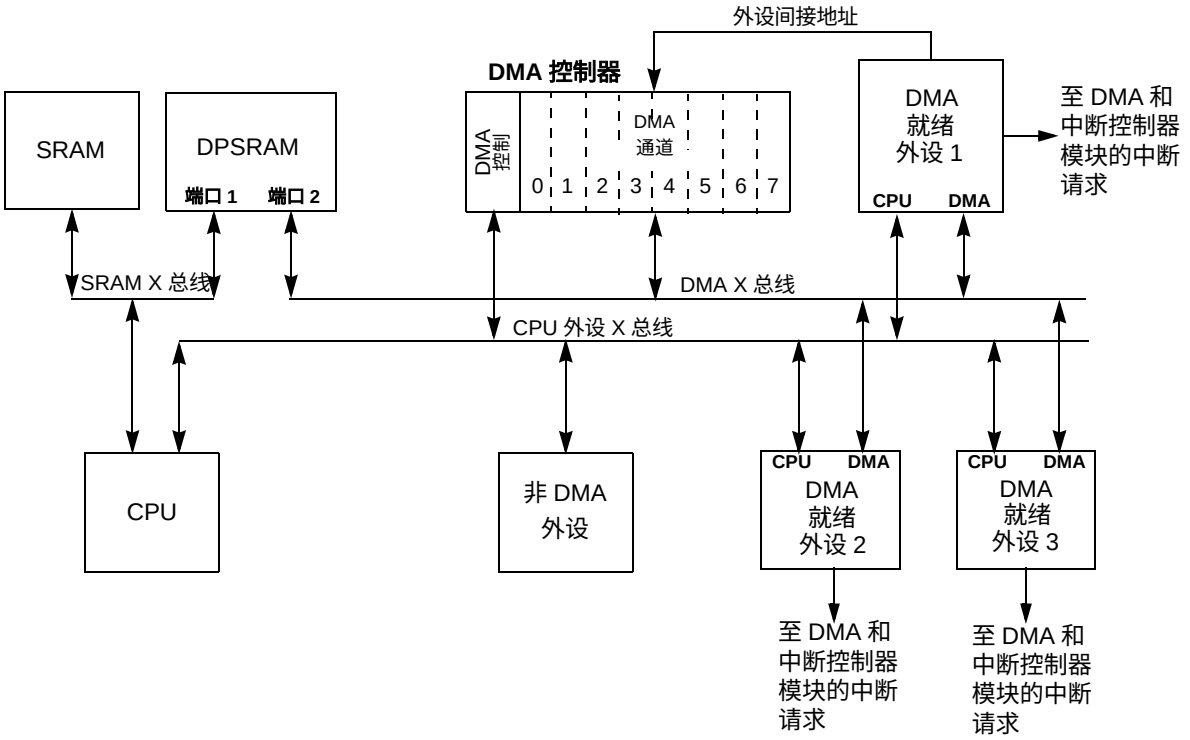
注： 该寄存器是只读的。

22.3 DMA 框图

图 22-2 的框图显示了如何将 DMA 集成到 dsPIC33F 内部架构中。 CPU 通过 X 总线与常规的 SRAM 通信。它还通过相同的 X 总线与双端口 SRAM（DPSRAM）模块的端口 1 通信。CPU 通过独立的外设 X 总线 （也位于 X 数据空间内）与外设通信。

DMA 通道通过专用的 DMA 总线与 DPSRAM 的端口 2 和每个 DMA 就绪外设的 DMA 端口通信。

图 22-2： DMA 控制器框图



注： 为清晰起见，没有显示 CPU 和 DMA 地址总线。

不同于其他架构，dsPIC33F CPU 可以在每个 CPU 总线周期中进行读和写访问。类似地，DMA 可以在每个总线周期通过其专用总线完成一个字节或一个字的传输。这也确保所有 DMA 传输都不会被中断。即，一旦开始传输，则无论其他通道的活动如何，都将在同一周期内完成传输。

用户应用程序可以指定任意 DMA 就绪外设中断作为 DMA 请求，“ DMA 请求 ”这个术语指针对 DMA 的中断请求 （IRQ）。当然，假设当某个 DMA 通道配置为将特定中断作为 DMA 请求进行响应时，相应的 CPU 中断已被禁止，否则也会请求产生 CPU 中断。

每个 DMA 通道也可以通过软件手动触发。将 DMAxCON 寄存器中的 FORCE 位置 1 会启动手动 DMA 请求，该请求与所有基于中断的 DMA 请求一样，服从相同的仲裁规则（见**第 22.8 节“ DMA 通道仲裁和溢出 ”**）。

22.4 DMA 数据传输

图 22-3 给出了外设和双端口 SRAM 之间的数据传输的图示。

A. 在该示例中， DMA 通道 5 配置为使用 DMA 就绪外设 1 工作。

B. 当数据准备好从外设传输时，外设会发出 DMA 请求。DMA 请求与所有其他同时产生的请求一起接受仲裁。如果该通道的优先级最高，则会在下一个周期完成传输。否则，DMA 请求保持等待处理状态，直到它的优先级变为最高。

C. DMA 通道对指定的外设地址执行读数据操作，该外设地址由用户应用程序在工作通道内定义。

D. DMA 通道将数据写入指定的 DPSRAM 地址。

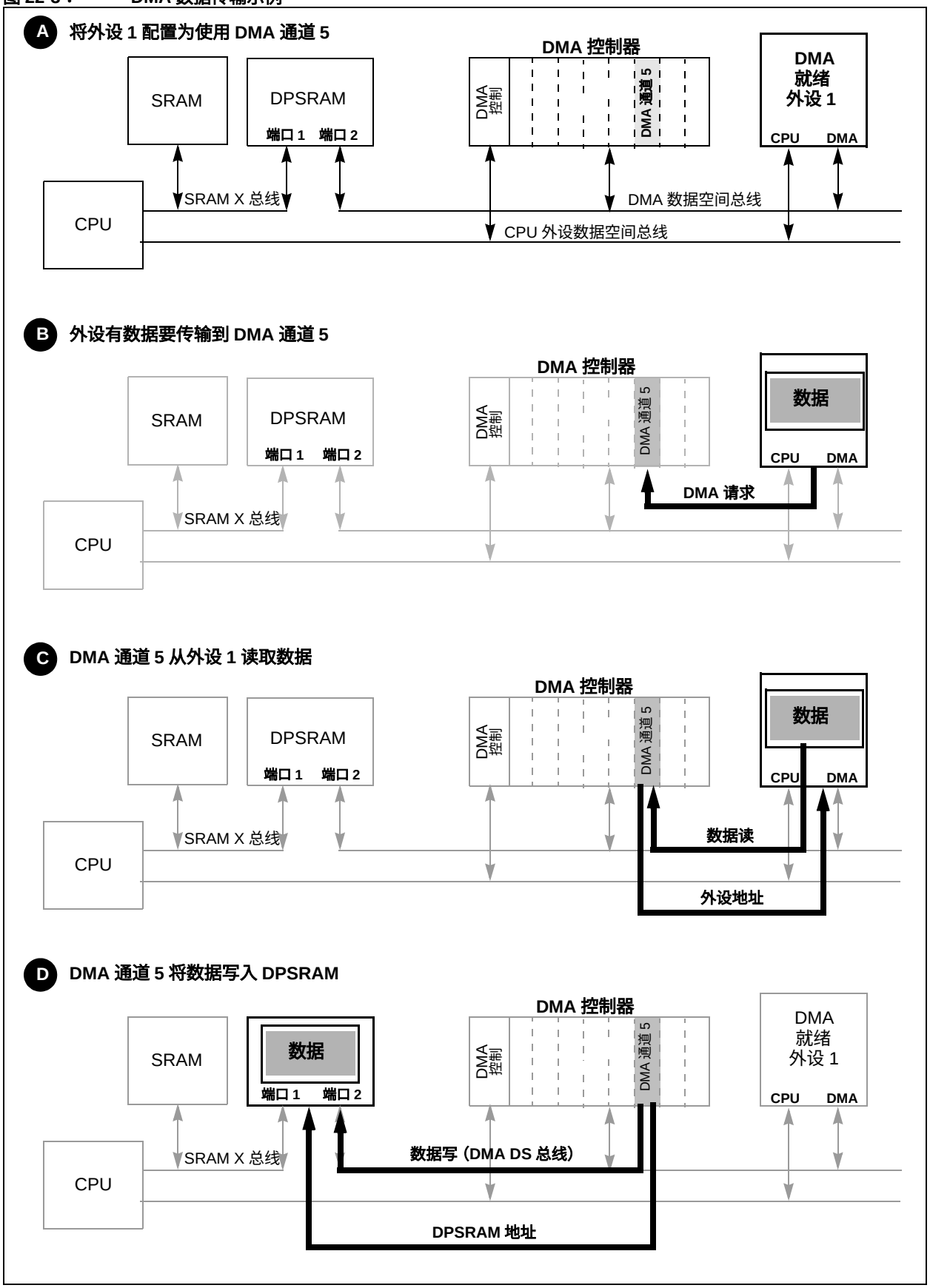
该示例代表寄存器间接寻址模式，在该模式下， DPSRAM 地址在 DMA 通道内通过 DMA 状态寄存器 （DMAxSTA 或 DMAxSTB）指定。在外设间接寻址模式下， DPSRAM 地址将从外设而不是工作通道获得。**第 22.6.6 节 “ 外设间接寻址模式 ”** 中介绍了关于该主题的更多信息。

整个 DMA 读和写传输操作在单个指令周期内无中断地完成。在这整个过程中，DMA 请求一直锁存在 DMA 通道中，直到数据传输完成。

DMA 通道会同时监视传输计数器寄存器（DMA5CNT）。当传输计数达到用户应用程序指定的限制时，数据传输即视为完成，并发出 CPU 中断，提醒 CPU 处理新接收到的数据。

在数据传输周期中，DMA 控制器还会继续对待处理或后续的 DMA 请求进行仲裁，以最大程度提高吞吐量。

图 22-3： DMA 数据传输示例



22.5 DMA 设置

为了 DMA 数据传输能够正常进行，必须适当地配置 DMA 通道和外设：

- 必须将 DMA 通道与外设关联 （见**第 22.5.1 节 “ DMA 通道与外设关联设置 ”**）
- 必须正确地配置外设 （见**第 22.5.2 节 “ 外设配置设置 ”**）
- 必须初始化 DPSRAM 数据起始地址 （见**第 22.5.3 节 “ 存储器地址初始化 ”**）
- 必须初始化 DMA 传输计数 （见**第 22.5.4 节 “ DMA 传输计数设置 ”**）
- 必须选择适当的寻址和工作模式 （见**第 22.6 节 “ DMA 工作模式 ”**）

22.5.1 DMA 通道与外设关联设置

DMA 通道需要知道要对哪个外设目标地址进行读操作或写操作，以及何时执行该操作。该信息分别在 DMA 通道 x 外设地址寄存器 （DMAxPAD）和 DMA 通道 x IRQ 选择寄存器 （DMAxREQ）中配置。

表 22-1 显示了要将特定外设与给定 DMA 通道关联，应向这些寄存器中写入哪些值。

表 22-1： DMA 通道与外设关联

外设与 DMA 关联	DMAxREQ 寄存器 IRQSEL<6:0> 位	要从外设读取的 DMAxPAD 寄存器值	要向外设写入的 DMAxPAD 寄存器值
INT0——外部中断 0	0000000	—	—
IC1——输入捕捉 1	0000001	0x0140 （IC1BUF）	—
IC2——输入捕捉 2	0000101	0x0144 （IC2BUF）	—
OC1——输出比较 1 数据	0000010	—	0x0182 （OC1R）
OC1——输出比较 1 辅助数据	0000010	—	0x0180 （OC1RS）
OC2——输出比较 2 数据	0000110	—	0x0188 （OC2R）
OC2——输出比较 2 辅助数据	0000110	—	0x0186 （OC2RS）
TMR2——Timer2	0000111	—	—
TMR3——Timer3	0001000	—	—
SPI1——传输完成	0001010	0x0248 （SPI1BUF）	0x0248 （SPI1BUF）
SPI2——传输完成	0100001	0x0268 （SPI2BUF）	0x0268 （SPI2BUF）
UART1RX——UART1 接收器	0001011	0x0226 （U1RXREG）	—
UART1TX——UART1 发送器	0001100	—	0x0224 （U1TXREG）
UART2RX——UART2 接收器	0011110	0x0236 （U2RXREG）	—
UART2TX——UART2 发送器	0011111	—	0x0234 （U2TXREG）
ECAN1——接收数据就绪	0100010	0x0440 （C1RXD）	—
ECAN1——发送数据请求	1000110	—	0x0442 （C1TXD）
ECAN2——接收数据就绪	0110111	0x0540 （C2RXD）	—
ECAN2——发送数据请求	1000111	—	0x0542 （C2TXD）
DCI——编解码器传输完成	0111100	0x0290 （RXBUF0）	0x0298 （TXBUF0）
ADC1——ADC1 转换完成	0001101	0x0300 （ADC1BUF0）	—
ADC2——ADC2 转换完成	0010101	0x0340 （ADC2BUF0）	—

如果两个 DMA 通道选择同一外设作为它们的 DMA 请求源，则两个通道会同时接收到 DMA 请求。但优先级最高的通道先执行它的传输，另一个通道保持等待状态。在使用单个 DMA 请求来从外设和向外设 （例如 SPI）传送数据时，也会发生同样的情况。需要使用两个 DMA 通道。一个分配为用于外设读操作，另一个分配为用于外设数据写操作。两个通道使用相同的 DMA 请求。

如果将 DMAxPAD 寄存器初始化为表 22-1 中未列出的值，则对该外设地址的 DMA 通道写操作会被忽略。对该地址执行 DMA 通道读操作将返回 0。

22.5.2 外设配置设置

DMA 设置过程中的第二步是为 DMA 操作正确配置 DMA 就绪外设。表 22-2 列出了对于 DMA 就绪外设的配置要求。

表 22-2： DMA 就绪外设的配置注意事项

DMA 就绪外设	配置注意事项
ECAN	ECAN 缓冲区在 DMA RAM 中分配。 DMA RAM 中的 CAN 缓冲区和 FIFO 的总大小由用户指定，必须通过 ECAN FIFO 控制 （C1FCTRL）寄存器中的 DMA 缓冲区大小位 DMABS<2:0> 定义。例 22-9 给出了示例代码。
数据转换器接口 （DCI）	DCI 必须配置为对于每个缓冲的数据字产生中断，方法是将 DCI 控制 2（DCICON2）寄存器中的缓冲区长度控制位 （BLEN<1:0>）设置为 00。必须将同一 DCI 中断用作两个 DMA 通道的请求，以支持接收和发送数据。如果 DCI 模块作为主器件工作且仅接收数据，则必须使用第二个 DMA 通道来发送无效发送数据。例 22-11 给出了示例代码。
10 位 /12 位模数转换器 （ADC）	在外设间接寻址模式下将 ADC 与 DMA 配合使用时，必须正确设置 ADCx 控制 2 （ADCxCON2）寄存器中的 DMA 地址递增速率位 （SMPI<3:0>）和 ADCx 控制 4（ADCxCON4）寄存器中的每个模拟输入的 DMA 缓冲单元数位（DMABL<2:0>）。此外，还必须正确设置 ADCx 控制 1 （ADxCON1）寄存器中的 DMA 缓冲区构建模式位 （ADDMABM），以进行 ADC 地址生成。详细信息，请参见**第 22.6.6.1 节 “ ADC 对于 DMA 地址生成的支持 ”**。例 22-5 和例 22-7 给出了示例代码。
串行外设接口 （SPI™）	如果SPI模块作为主器件工作且仅接收数据，则必须分配并使用第二个DMA通道来发送无效发送数据。或者，可以在空数据写模式下使用单个 DMA 通道。详细信息，请参见**第 22.6.11 节 “ 空数据写模式 ”**。例 22-12 给出了示例代码。
UART	UART 必须配置为对于接收或发送的每个字符产生中断。为使 UART 接收器对于接收的每个字符产生接收中断，必须将状态和控制寄存器（UxSTA）中的接收中断模式选择位 （URXISEL<1:0>）设置为 00 或 01。 为使 UART 发送器对于发送的每个字符产生发送中断，必须将状态和控制（UxSTA）寄存器中的发送中断模式选择位 UTXISEL0 和 UTXISEL1 设置为 0。例 22-10 给出了示例代码。
输入捕捉	输入捕捉模块必须配置为对于每个捕捉事件产生中断，方法是将输入捕捉控制（ICxCON）寄存器中的每个中断的捕捉数位（ICI<1:0>）设置为 00。例 22-4 给出了示例代码。
输出比较	输出比较模块无需特别配置即可使用 DMA。但是，通常会使用定时器来产生 DMA 请求，所以需要正确配置定时器。例 22-3 给出了示例代码。

表 22-2： DMA 就绪外设的配置注意事项 （续）

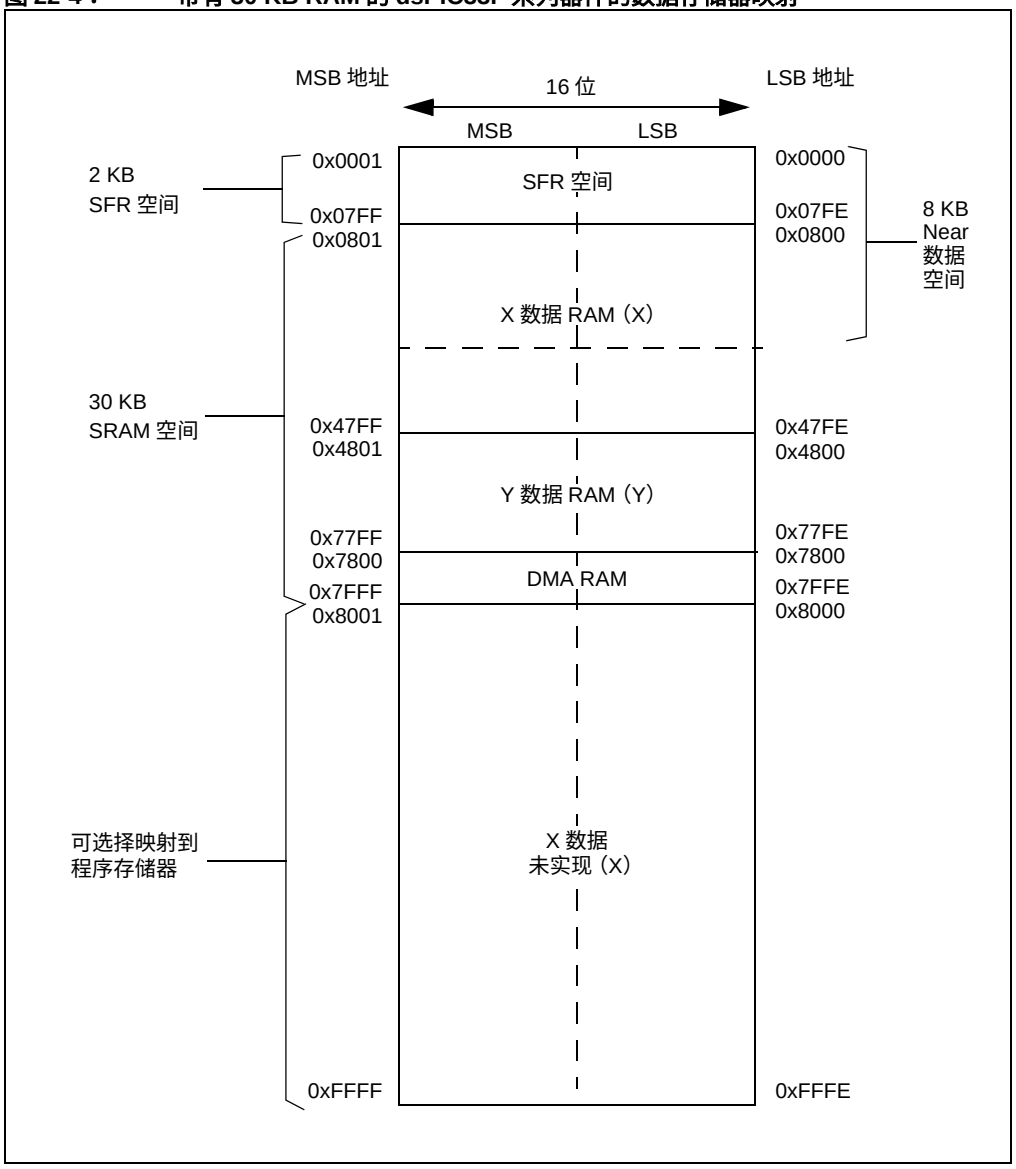
DMA 就绪外设	配置注意事项
外部中断和定时器	只有外部中断 0、Timer2 和 Timer3 可以选择用于产生 DMA 请求。虽然这些外设本身不支持DMA传输，但它们可用于触发其他支持DMA的外设的DMA传输。例如， Timer2 能触发处于 PWM 模式的输出比较外设的 DMA 事务。例 22-3 给出了示例代码。

在支持 DMA 的外设中产生错误条件时，通常会将状态标志置 1 并产生中断 （如果用户应用程序允许中断）。在由 CPU 为外设服务时，需要数据中断处理程序检查错误标志，并根据需要执行相应的操作。但在由 DMA 通道为外设服务时，DMA 只能响应数据传输请求，它不会知道任何后续的错误条件。DMA 兼容外设中的所有错误条件都必须允许关联的中断，并在外设中出现对应中断时，由用户定义的中断服务程序 （Interrupt Service Routine， ISR）进行处理。

22.5.3 存储器地址初始化

第三个 DMA 设置要求是在特定存储区中为 DMA 访问分配存储器缓冲区。该存储区的位置和大小取决于具体的 dsPIC33F 器件 （具体信息，请参见器件数据手册）。图 22-4 显示了带有 30 KB RAM 的 dsPIC33F 器件的 2 KB 大小 DMA 存储区。

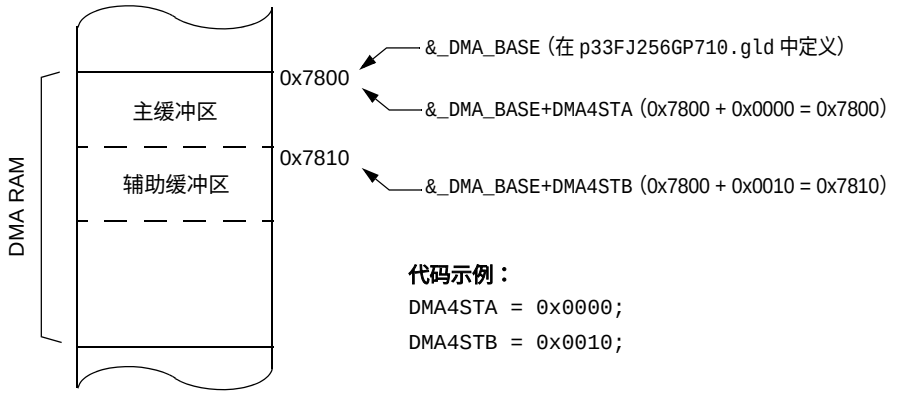
图 22-4： 带有 30 KB RAM 的 dsPIC33F 系列器件的数据存储器映射



为了正确工作，DMA 需要知道要读取或写入的 DPSRAM 地址，地址表示为相对于 DMA 存储器起始地址的偏移。该信息在 DMA 通道 x DPSRAM 起始地址偏移 A（DMAxSTA）寄存器和 DMA 通道 x DPSRAM 起始地址偏移 B （DMAxSTB）寄存器中配置。

图 22-5 给出了一个示例，说明如何在 dsPIC33FJ256GP710 器件上，将 DMA 通道 4 主缓冲区和辅助缓冲区分别设置在地址 0x7800 和 0x7810 处。

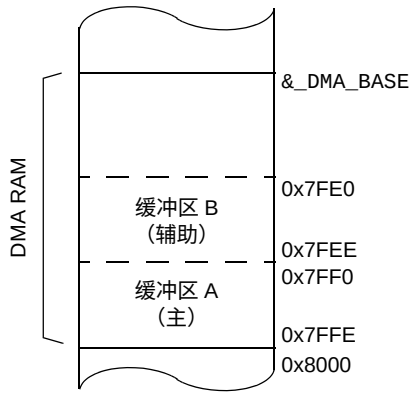
图 22-5： DMA 存储器中的主缓冲区和辅助缓冲区分配



在该示例中，您必须熟悉器件的存储器布局，以便将该信息硬编码到应用程序中。此外，在 DMA 传输完成之后，还必须使用指针算法来访问这些缓冲区。所以，这种实现难以从一种处理器移植到另一种处理器。

MPLAB® C30 编译器提供了用于该用途的内建 C 语言原语，可以简化 DMA 缓冲区的初始化和访问。例如，图 22-6 中的代码会在 DMA 存储器中分配两个缓冲区，并初始化 DMA 通道，使之指向它们。

图 22-6： 使用 MPLAB® IDE 分配 DMA 主缓冲区和辅助缓冲区



代码示例：

```
unsigned int BufferA[8] __attribute__((space(dma)));
unsigned int BufferB[8] __attribute__((space(dma)));

DMA0STA = __builtin_dmaoffset(BufferA);
DMA0STB = __builtin_dmaoffset(BufferB);
```

注： MPLAB LINK30 链接器会从 DMA 存储空间底部开始，按相反顺序分配主缓冲区和辅助缓冲区。

如果将 DMAxSTA（和 / 或 DMAxSTB）寄存器初始化为某个值，导致 DMA 通道读取或写入 DMA RAM 空间之外的 RAM 地址，则对该存储器地址的 DMA 通道写操作会被忽略。对该存储器地址执行 DMA 通道读操作将返回 0。

22.5.4 DMA 传输计数设置

在 DMA 设置过程的第四步中，必须将每个 DMA 通道设定为处理 N + 1 个请求，处理完该数量的请求之后，数据块传输才视为完成。“ N ”的值通过设置 DMA 通道 x 传输计数 （DMAxCNT）寄存器指定。即， DMAxCNT 值为 0 时，将传输一个数据元素。

DMAxCNT 寄存器的值与传输数据长度（字节或字）无关，后者在 DMAxCON 寄存器的 SIZE 位中指定。

如果将 DMAxCNT 寄存器初始化为某个值，导致 DMA 通道读取或写入 DMA RAM 空间之外的 RAM 地址，则对该存储器地址的 DMA 通道写操作会被忽略。对该存储器地址执行 DMA 通道读操作将返回 0。

22.5.5 工作模式设置

DMA 设置过程的第五步 （也是最后一步）是指定每个 DMA 通道的工作模式，方法是配置 DMA 通道 x 控制 （DMAxCON）寄存器。具体的设置信息，请参见**第 22.6 节 “ DMA 工作模式 ”**。

22.6 DMA 工作模式

DMA 通道支持以下工作模式：

- 字或字节数据传输
- 传输方向 （外设至 DPSRAM，或 DPSRAM 至外设）
- 在传输完整个或一半数据块后发出中断给 CPU
- 后递增或静态 DPSRAM 寻址
- 外设间接寻址
- 单数据块或连续数据块传输
- 每个数据块传输完成后，在两个起始地址 （DMAxSTA 或 DMAxSTB）之间进行自动切换 （乒乓模式）
- 空数据写模式

此外， DMA 还支持手动模式，该模式会强制执行单次 DMA 传输。

22.6.1 字或字节数据传输

每个 DMA 通道都可以配置为按字或按字节传输数据。字数据只能传送到对齐 （偶）地址或从对齐 （偶）地址传送出。另一方面，字节数据可以传送到任意 （合法）地址或从任意 （合法）地址传送出。

如果 SIZE 位 （DMAxCON<14>）清零，则传输字大小的数据。如果使能了带后递增的寄存器间接寻址模式，则在传输每个字之后地址将递增 2 （见**第 22.6.5 节 “ 不带后递增的寄存器间接寻址模式 ”**）。

如果 SIZE 位置 1，则传输字节长度的数据。如果使能了带后递增的寄存器间接寻址模式，则在传输每个字节之后地址将递增 1。

22.6.2 传输方向

每个 DMA 通道都可以配置为从外设向 DPSRAM 传输数据或从 DPSRAM 向外设传输数据。

如果 DMAxCON 中的传输方向 （DIR）位清零，则从外设读取数据 （使用 DMAxPAD 提供的外设地址），并对 DPSRAM DMA 存储器地址偏移 （使用 DMAxSTA 或 DMAxSTB）位置执行目标写操作。

如果 DIR 位置 1，则从 DPSRAM DMA 存储器地址偏移 （使用 DMAxSTA 或 DMAxSTB）位置读取数据，并对外设执行目标写操作 （使用 DMAxPAD 提供的外设地址）。

进行配置之后，每个通道都是单向数据通道。即，如果某个外设需要使用 DMA 模块读写数据，则必须分配两个通道——一个用于读操作，一个用于写操作。

22.6.3 全数据块或半数据块传输中断

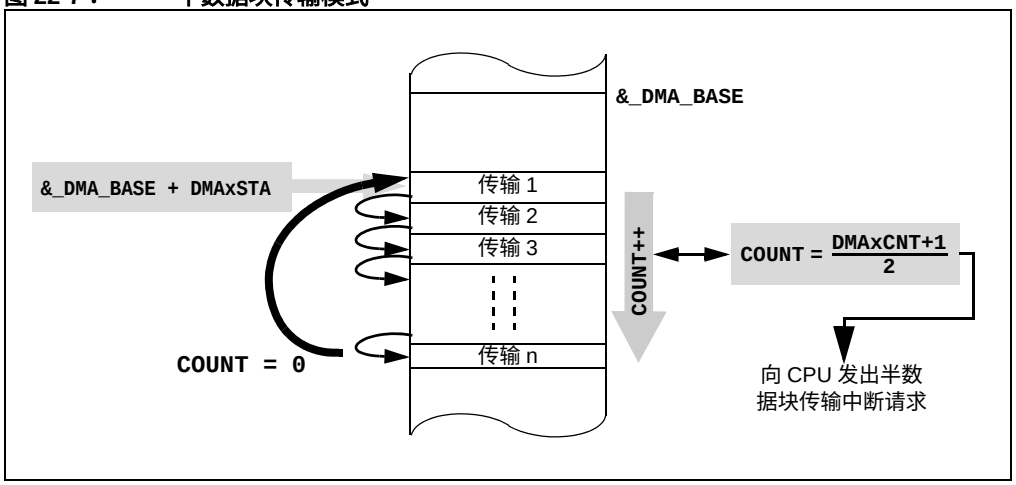
在数据块数据传输全部完成或完成一半时，每个 DMA 通道会向中断控制器发出中断。该模式通过将 DMA 通道 x 控制 （DMAxCON）寄存器中的 HALF 位清零或置 1 来指定：

HALF = 0 （当传送了所有数据时，发出中断）

HALF = 1 （当传送了一半数据时，发出中断）

使用 DMA 连续模式时， CPU 处理传入或外发数据的速度必须至少达到 DMA 传送数据的速度。半数据块传输中断可以在仅传输一半数据时产生中断，帮助减轻该问题。例如，如果通过 DMA 控制器连续读取 ADC，半数据块传输中断使 CPU 可以在缓冲区全满之前处理缓冲区数据。只要它不超过 DMA 写操作的位置，就可以使用这种方案来缓解对于 CPU 响应速度的要求。图 22-7 给出了该过程的图示。

图 22-7： 半数据块传输模式



在所有模式下，如果 HALF 位置 1，则 DMA 只会在传输完缓冲区 A 和 / 或 B 的前一半数据时发出中断。缓冲区 A 和 / 或 B 传输完成时不会发出中断。也就是说，只有在 DMA 完成 (DMAxCNT + 1)/2 次数据传输时，才会发出中断。如果 (DMAxCNT + 1) 等于奇数，则在 (DMAxCNT + 2)/2 次数据传输之后发出中断。

例如，如果为 DMA3 配置了单数据块乒乓缓冲区 （MODE<1:0> = 11），且 DMA3CNT = 7，则会发出两次 DMA3 中断，从缓冲区 A 传输 4 个数据元素后发出一次，从缓冲区 B 传输 4 个数据元素后发出一次。（更多信息，请参见**第 22.6.7 节“ 单数据块模式 ”**和**第 22.6.9 节“ 乒乓模式 ”**。）

虽然 DMA 通道只对于半数据块**或**全数据块传输发出一次中断，可以在用户应用程序中使用技巧使DMA通道对于半数据块**和**全数据块传输都发出一次中断，方法是在每次DMA中断中翻转HALF位的值。例如，如果 DMA 通道设置为 HALF 位置 1，则会在每次完成半数据块传输时发出中断。如果用户应用程序在处理中断的过程中将 HALF 位复位为 0，则在完成全数据块传输时，DMA 会发出另一次中断。

要允许这些中断，必须将中断控制器模块的中断允许控制 （IECx）寄存器中相应的 DMA 中断允许位 （DMAxIE）置 1，如表 22-3 所示。

表 22-3： 用于允许 / 禁止 DMA 中断的中断控制器设置

DMA 通道	中断控制器寄存器名称和位编号	对应的寄存器位名称	C 语言结构的访问代码
0	IEC0<4>	DMA0IE	IEC0bits.DMA0IE
1	IEC0<14>	DMA1IE	IEC1bits.DMA1IE
2	IEC1<8>	DMA2IE	IEC1bits.DMA2IE
3	IEC2<4>	DMA3IE	IEC2bits.DMA3IE
4	IEC2<14>	DMA4IE	IEC2bits.DMA4IE
5	IEC3<13>	DMA5IE	IEC3bits.DMA5IE
6	IEC4<4>	DMA6IE	IEC4bits.DMA6IE
7	IEC4<5>	DAM7IE	IEC4bits.DMA7IE

例 22-1 说明了如何允许 DMA 通道 0 中断：

例 22-1： 用于允许 DMA 通道 0 中断的代码

```
IEC0bits.DMA0IE = 1;
```

每个 DMA 通道传输中断会将中断控制器中的相应状态标志置 1，这会触发中断服务程序（ISR）。然后，用户应用程序必须清零该状态标志，以防止重新执行传输完成 ISR。

表 22-4 列出了中断控制器模块中的中断标志状态 （IFSx）寄存器和对应的位名称 （DMAxIF）。它还给出了将标志清零的 C 语言结构访问代码。

表 22-4： 用于清除 DMA 中断状态标志的中断控制器设置

DMA 通道	中断控制器寄存器名称和位编号	对应的寄存器位名称	C 语言结构的访问代码
0	IFS0<4>	DMA0IF	IFS0bits.DMA0IF
1	IFS0<14>	DMA1IF	IFS0bits.DMA1IF
2	IFS1<8>	DMA2IF	IFS1bits.DMA2IF
3	IFS2<4>	DMA3IF	IFS2bits.DMA3IF
4	IFS2<14>	DMA4IF	IFS2bits.DMA4IF
5	IFS3<13>	DMA5IF	IFS3bits.DMA5IF
6	IFS4<4>	DMA6IF	IFS4bits.DMA6IF
7	IFS4<5>	DMA7IF	IFS4bits.DMA7IF

举例来说，假设允许了 DMA 通道 0 中断，DMA 通道 0 传输已完成，并向中断控制器发出了关联的中断。 DMA 通道 0 ISR 中必须出现以下代码，用于将状态标志清零，防止中断挂起。

例 22-2： 用于清除 DMA 通道 0 中断的代码

```
void __attribute__((__interrupt__)) _DMA0Interrupt(void)
{
    . . .

IFS0bits.DMA0IF = 0;
}
```

22.6.4 带后递增的寄存器间接寻址模式

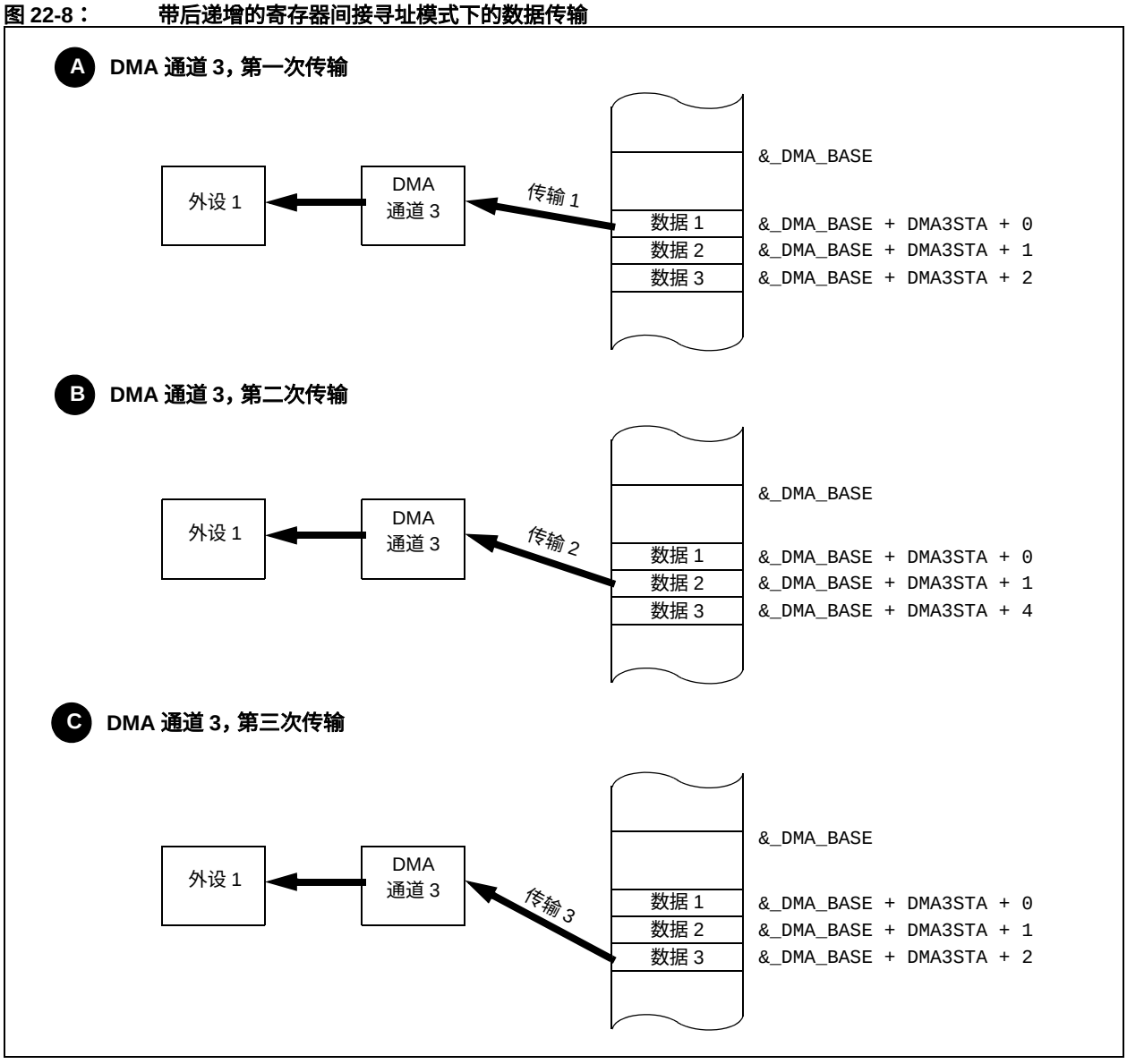
带后递增的寄存器间接寻址用于通过在每次传输后递增 DPSRAM 地址来传送数据块。

在 DMA 控制器复位之后， DMA 通道默认设置为该模式。该模式通过将 DMA 通道控制（DMAxCON）寄存器中的寻址模式选择位 （AMODE<1:0>）设置为 00 来进行选择。在该模式下，DPSRAM起始地址偏移（DMAxSTA或DMAxSTB）寄存器提供DPSRAM缓冲区的起始地址。

用户应用程序通过读取DPSRAM起始地址偏移寄存器来确定最新的DPSRAM传输地址偏移。但是， DMA 控制器不会修改该寄存器的内容。

图 22-8 给出了该模式下数据传输的图示。

图 22-8： 带后递增的寄存器间接寻址模式下的数据传输



例 22-3： 用于带后递增的寄存器间接寻址模式下的输出比较和 DMA 的代码

将输出比较 1 模块设置为 PWM 模式：

```
OC1CON = 0;                         // Reset OC module
OC1R = 0x60;                        // Initialize PWM Duty Cycle
OC1RS = 0x60;                       // Initialize PWM Duty Cycle Buffer

OC1CONbits.OCM = 6;                 // Configure OC for the PWM mode
```

将 DMA 通道 3 设置为后递增模式，并使用 Timer2 作为请求源：

```
unsigned int BufferA[32] __attribute__((space(dma)));
/* Insert code here to initialize BufferA with desired Duty Cycle values */

DMA3CONbits.AMODE = 0;              // Configure DMA for Register indirect mode
                                    // with post-increment
DMA3CONbits.MODE = 0;               // Configure DMA for Continuous mode
DMA3CONbits.DIR   = 1;              // RAM-to-Peripheral data transfers
DMA3PAD = (volatile unsigned int)&OC1RS; // Point DMA to OC1RS
DMA3CNT = 31;                       // 32 DMA request
DMA3REQ = 7;                        // Select Timer2 as DMA Request source

DMA3STA = __builtin_dmaoffset(BufferA);

IFS2bits.DMA3IF = 0;                // Clear the DMA interrupt flag bit
IEC2bits.DMA3IE = 1;                // Set the DMA interrupt enable bit

DMA3CONbits.CHEN = 1;               // Enable DMA
```

为输出比较 PWM 模式设置 Timer2：

```
PR2 = 0xBF;                         // Initialize PWM period
T2CONbits.TON = 1;                  // Start Timer2
```

设置 DMA 通道 3 中断处理程序：

```
void __attribute__((__interrupt__)) _DMA3Interrupt(void)
{
    /* Update BufferA with new Duty Cycle values if desired here*/

    IFS2bits.DMA3IF = 0;            //Clear the DMA3 Interrupt Flag
}
```

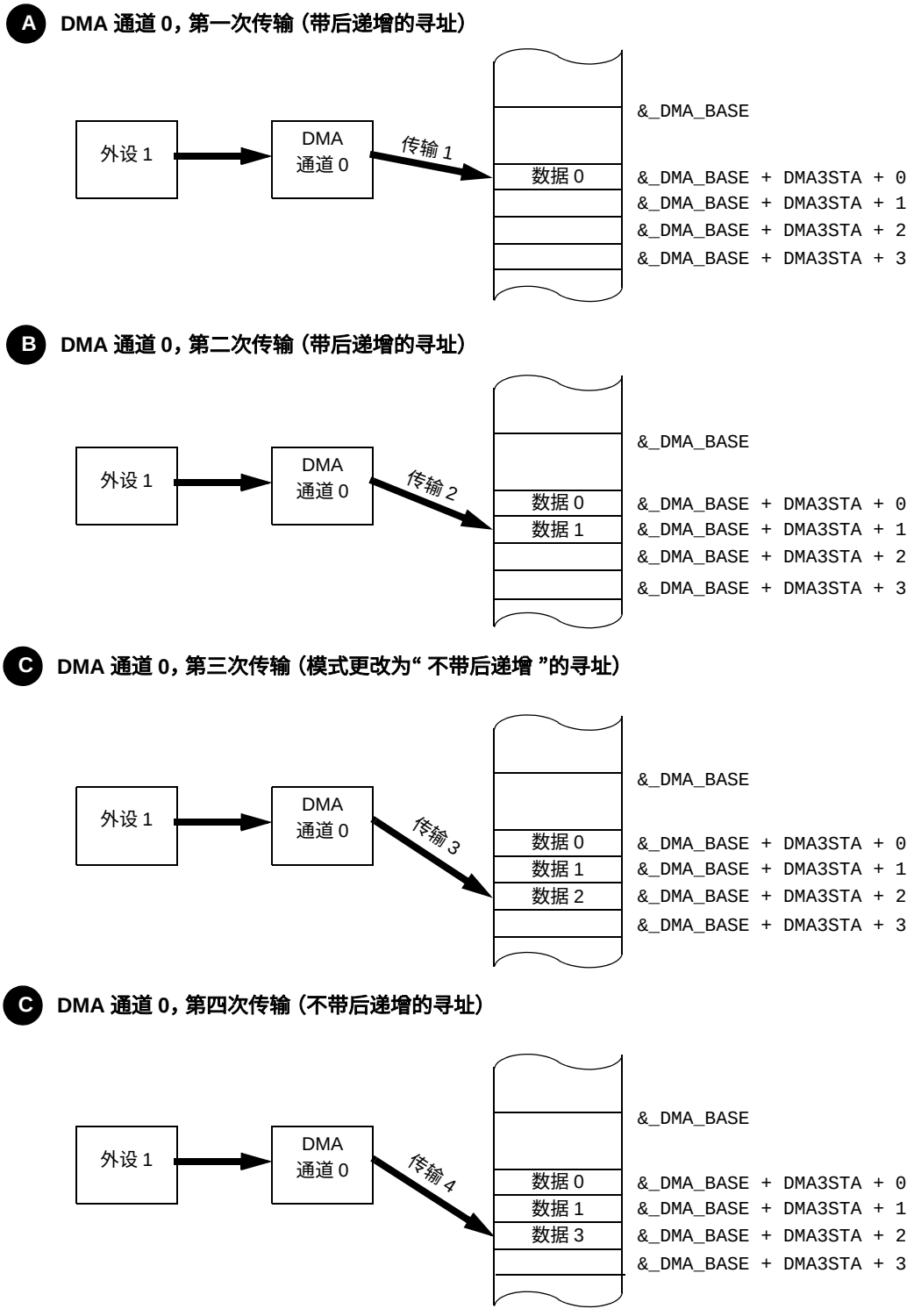
22.6.5 不带后递增的寄存器间接寻址模式

不带后递增的寄存器间接寻址用于在每次传输后不递增数据缓冲区起始地址的情况下传送数据块。在该模式下，DPSRAM起始地址偏移（DMAxSTA或DMAxSTB）寄存器提供相对于DPSRAM缓冲区起始地址的偏移。当发生 DMA 数据传输时， DPSRAM 地址不会递增到下一单元。所以，下一次 DMA 数据传输在同一 DPSRAM 地址发生。

该模式通过将 DMA 通道控制 （DMAxCON）寄存器中的寻址模式选择位 （AMODE<1:0>）设置为 01 来进行选择。

如果在 DMA 通道处于工作状态 （即，在发生几次 DMA 传输之后）时将寻址模式更改为不带后递增的寄存器间接寻址模式， DMA DPSRAM 地址将指向当前的 DPSRAM 缓冲区位置 （即，不是 DMAxSTA 或 DMAxSTB 的内容，此时它可能不同于当前的 DPSRAM 缓冲区位置）。图 22-9 给出了从外设到 DMA DPSRAM 的数据传输的图示，比较了使用带后递增寻址模式和不带后递增寻址模式的情况。

图 22-9： 带后递增的寻址和不带后递增的寻址的比较



例 22-4： 用于不带后递增的寄存器间接寻址模式下的输入捕捉和 DMA 的代码

为 DMA 操作设置输入捕捉 1 模块：

```
IC1CON = 0;                    // Reset IC module
IC1CONbits.ICTMR = 1;          // Select Timer2 contents for capture
IC1CONbits.ICM   = 2;          // Capture every falling edge
IC1CONbits.ICI   = 0;          // Generate DMA request on every capture event
```

设置要由输入捕捉模块使用的 Timer2：

```
PR2 = 0xBF;                    // Initialize count value
T2CONbits.TON = 1;             // Start timer
```

将 DMA 通道 0 设置为不带后递增的寻址模式：

```
unsigned int CaptureValue __attribute__((space(dma)));

DMA0CONbits.AMODE = 1;         // Configure DMA for Register indirect
                               // without post-increment
DMA0CONbits.MODE = 0;          // Configure DMA for Continuous mode
DMA0PAD = (volatile unsigned int)&IC1BUF; // Point DMA to IC1BUF
DMA0CNT = 0;                   // Interrupt after each transfer
DMA0REQ = 1;                   // Select Input Capture module as DMA Request source

DMA3STA = __builtin_dmaoffset(&CaptureValue);

IFS0bits.DMA0IF = 0;           // Clear the DMA interrupt flag bit
IEC0bits.DMA0IE = 1;           // Set the DMA interrupt enable bit

DMA0CONbits.CHEN = 1;          // Enable DMA
```

设置 DMA 通道 0 中断处理程序：

```
void __attribute__((__interrupt__)) _DMA3Interrupt(void)
{
    /* Process CaptureValue variable here*/

    IFS0bits.DMA0IF = 0;       //Clear the DMA3 Interrupt Flag
}
```

22.6.6 外设间接寻址模式

外设间接寻址模式是一种特殊的寻址模式，在该模式下，由外设而不是 DMA 通道提供 DPSRAM 地址的可变部分。即，外设生成 DPSRAM 地址的最低有效位 （Least Significant bit，LSb），而 DMA 通道提供固定的缓冲区基址。但是，DMA 通道会继续协调实际的数据传输，跟踪传输计数，并产生相应的 CPU 中断。

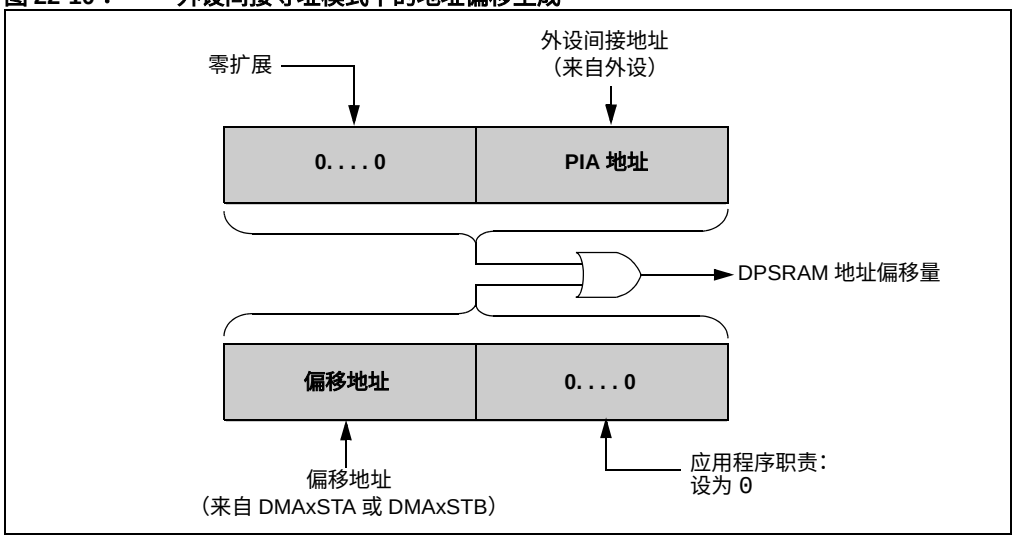
外设间接寻址模式可以根据外设需要双向工作，所以仍然需要适当地配置 DMA 通道，以支持目标外设读或写操作。

外设间接寻址模式通过将DMA通道控制 （DMAxCON） 寄存器中的寻址模式选择位 （AMODE<1:0>） 设置为 1x 来进行选择。

外设间接寻址模式下的 DMA 功能可以进行特别定制，以满足支持它的每个外设的需求。外设定义用于访问 DPSRAM 中数据的地址序列，举例来说，这使它可以将传入的 ADC 数据分选到多个缓冲区中，减轻 CPU 的任务处理负担。

如果某个外设支持外设间接寻址模式，则来自该外设的 DMA 请求中断会伴随一个地址，该地址会被送到 DMA 通道。如果响应该请求的 DMA 通道也使能了外设间接寻址模式，则它会将缓冲区基址与经过零扩展的传入外设间接地址进行逻辑或运算，产生实际的 DPSRAM 偏移地址，如图 22-10 所示。

图 22-10： 外设间接寻址模式下的地址偏移生成



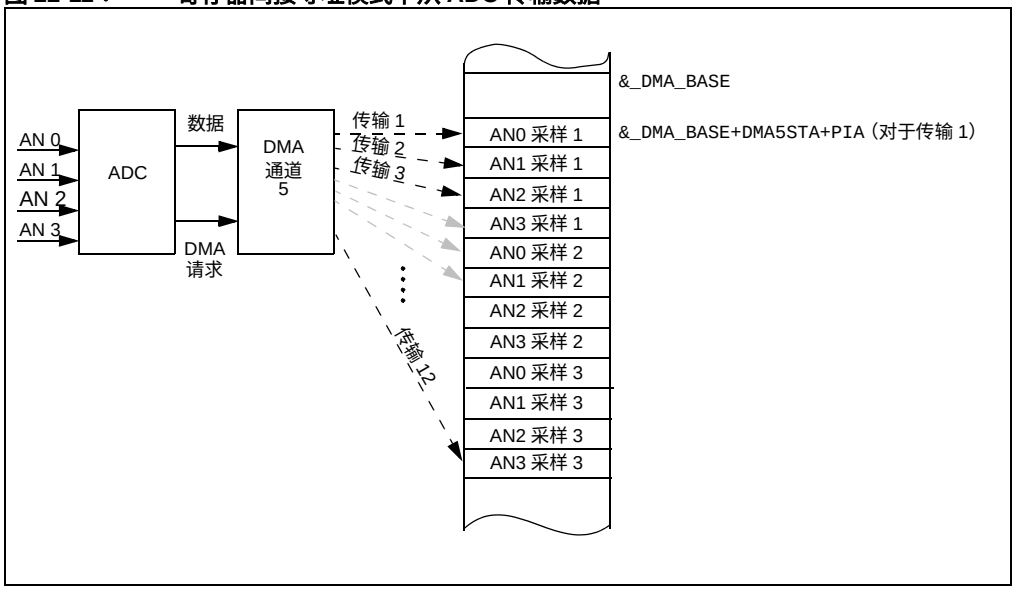
外设决定它将控制多少个最低有效地址位。应用程序必须选择缓冲区在 DPSRAM 中的基址，并确保该地址偏移中相应数量的最低有效位为 0。对于其他模式，读 DPSRAM 起始地址偏移寄存器时，它会返回最新的 DPSRAM 传输地址偏移的值，其中包含了上述的地址偏移计算。如果 DMA 通道未配置为外设间接寻址模式，则传入的地址会被忽略，数据传输正常进行。

外设间接寻址模式与所有其他工作模式兼容， ADC 和 ECAN 模块当前都支持它。

22.6.6.1 ADC 对于 DMA 地址生成的支持

在外设间接寻址模式下，由外设定义寻址序列，这可以更好地适应外设功能。例如，如果 ADC 配置为按顺序（例如，0、1、2、3、0、1，并依此类推）连续转换输入 0 至 3，并且它与配置为带后递增的寄存器间接寻址模式的 DMA 通道关联，则 DMA 传输会将该数据传送到连续缓冲区中，如图 22-11 所示。例 22-5 给出了该配置的代码示例。

图 22-11： 寄存器间接寻址模式下从 ADC 传输数据



例 22-5： 用于寄存器间接寻址模式下从 ADC 传输数据的代码

设置 ADC1 进行通道 0-3 采样：

```
AD1CON1bits.FORM = 3;         // Data Output Format:Signed Fraction (Q15 format)
AD1CON1bits.SSRC = 2;         // Sample Clock Source:GP Timer starts conversion
AD1CON1bits.ASAM = 1;         // Sampling begins immediately after conversion
AD1CON1bits.AD12B = 0;        // 10-bit ADC operation
AD1CON1bits.SIMSAM = 0;       // Samples individual channels sequentially

AD1CON2bits.BUFM = 0;
AD1CON2bits.CSCNA = 1;        // Scan CH0+ Input Selections during Sample A bit
AD1CON2bits.CHPS = 0;         // Converts CH0

AD1CON3bits.ADRC = 0;         // ADC Clock is derived from Systems Clock
AD1CON3bits.ADCS = 63;        // ADC Conversion Clock

//AD1CHS0:A/D Input Select Register
AD1CHS0bits.CH0SA = 0;        // MUXA +ve input selection (AIN0) for CH0
AD1CHS0bits.CH0NA = 0;        // MUXA -ve input selection (VREF-) for CH0

//AD1CHS123:A/D Input Select Register
AD1CHS123bits.CH123SA = 0;    // MUXA +ve input selection (AIN0) for CH1
AD1CHS123bits.CH123NA = 0;    // MUXA -ve input selection (VREF-) for CH1

//AD1CSSH/AD1CSSL:A/D Input Scan Selection Register
AD1CSSH = 0x0000;
AD1CSSL = 0x000F;             // Scan AIN0, AIN1, AIN2, AIN3 inputs
```

设置 Timer3 来触发 ADC1 转换：

```
TMR3 = 0x0000;
PR3 = 4999;                   // Trigger ADC1 every 125 μs @ 40 MIPS
IFS0bits.T3IF = 0;            // Clear Timer3 interrupt
IEC0bits.T3IE = 0;            // Disable Timer3 interrupt

T3CONbits.TON = 1;            //Start Timer3
```

将 DMA 通道 5 设置为带后递增的寄存器间接寻址：

```
unsigned int BufferA[32] __attribute__((space(dma)));
unsigned int BufferB[32] __attribute__((space(dma)));

DMA5CONbits.AMODE = 0;        // Configure DMA for Register indirect mode
                              // with post-increment
DMA5CONbits.MODE = 2;         // Configure DMA for Continuous Ping-Pong mode
DMA5PAD = (volatile unsigned int)&ADC1BUF0; // Point DMA to ADC1BUF0
DMA5CNT = 31;                 // 32 DMA request
DMA5REQ = 13;                 // Select ADC1 as DMA Request source

DMA5STA = __builtin_dmaoffset(BufferA);
DMA5STB = __builtin_dmaoffset(BufferB);

IFS3bits.DMA5IF = 0;          //Clear the DMA interrupt flag bit
IEC3bits.DMA5IE = 1;          //Set the DMA interrupt enable bit

DMA5CONbits.CHEN=1;           // Enable DMA
```

例 22-5： 用于寄存器间接寻址模式下从 ADC 传输数据的代码 （续）

设置 DMA 通道 5 中断处理程序：

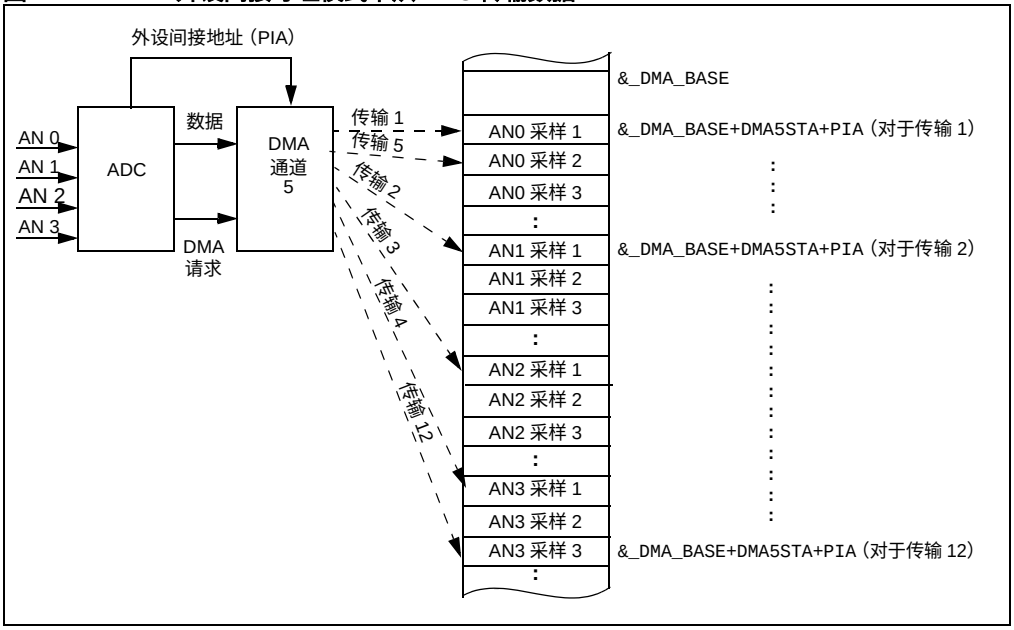
```
unsigned int DmaBuffer = 0;

void __attribute__((__interrupt__)) _DMA5Interrupt(void)
{
    // Switch between Primary and Secondary Ping-Pong buffers
if(DmaBuffer == 0)
    {
        ProcessADCSamples(BufferA);
    }
    else
    {
        ProcessADCSamples(BufferB);
    }

    DmaBuffer ^= 1;

    IFS3bits.DMA5IF = 0;      //Clear the DMA5 Interrupt Flag
}
```

为 DMA 操作设置 ADC1：

```
AD1CON1bits.ADDMABM = 0;      // Don't Care:ADC address generation is
                              // Ignored by DMA
AD1CON2bits.SMPI    = 3;      // Don't Care
AD1CON4bits.DMABL   = 3;      // Don't Care

IFS0bits.AD1IF      = 0;      // Clear the A/D interrupt flag bit
IEC0bits.AD1IE      = 0;      // Do Not Enable A/D interrupt
AD1CON1bits.ADON    = 1;      // Turn on the A/D converter
```

典型的算法按照 ADC 数据通道来进行操作，这要求它或者对传输的数据进行分选，或者通过跳过不需要的数据来以索引方式访问数据。这两种方法都需要更多的代码，需要消耗更多的执行时间。 ADC 外设间接寻址模式定义了一种特殊的寻址技术，在该技术中，每个 ADC 通道的数据会被放入它自己的缓冲区中。对于以上示例，如果 DMA 通道配置为外设间接寻址模式，DMA 传输会将 ADC 数据传送到独立的缓冲区中，如图 22-12 所示。

图 22-12： 外设间接寻址模式下从 ADC 传输数据



要使能这种 ADC 寻址，必须将 ADCx 控制 1 （ADxCON1）寄存器中的 DMA 缓冲区构建模式（ADDMABM）位清零。如果该位置 1， ADC 会按照转换顺序生成地址 （等同于 DMA 的带后递增的寄存器间接寻址模式）。

如前面所提到，在用户应用程序初始化DPSRAM起始地址偏移寄存器（DMAxSTA和DMAxSTB）时，必须特别注意为外设保留的最低有效位的位数。对于 ADC，位数取决于 ADC 缓冲区的大小和数量。

ADC 缓冲区的数量使用 ADCx 控制 2 （ADxCON2）寄存器中的 DMA 地址递增速率位 SMPI<3:0> 初始化。每个 ADC 缓冲区的大小使用 ADCx 控制 4 （ADCxCON4）寄存器中的每个模拟输入 DMA 缓冲单元数位 DMABL<2:0> 初始化。例如，如果 SMPI<3:0> 初始化为 3，DMABL<2:0> 初始化为 3，则有 4 个 ADC 缓冲区 （SMPI<3:0> + 1），每个缓冲区有 8 个字（$2^{DMABL<2:0>}$），总共 32 个字 （64 字节）。这意味着写入 DMAxSTA 和 DMAxSTB 的地址偏移必须将 6 个 （$2^{6\ 位}$ = 64 字节）最低有效位设置为 0。

如果使用 MPLAB® C30 编译器来初始化 DMAxSTA 和 DMAxSTAB 寄存器，则必须通过数据属性来指定正确的数据对齐。对于以上条件，例 22-6 中所示代码可以正确地初始化 DMAxSTA 和 DMAxSTB 寄存器。

例 22-6： 使用 MPLAB® C30 时的 DMA 缓冲区对齐

```
int  BufferA[4][8] __attribute__((space(dma),aligned(64)));
int  BufferB[4][8] __attribute__((space(dma),aligned(64)));

DMA0STA = __builtin_dmaoffset(BufferA);
DMA0STB = __builtin_dmaoffset(BufferB);
```

例 22-7 给出了该配置的代码示例。

例 22-7： 用于外设间接寻址模式下的 ADC 和 DMA 的代码

设置 ADC1 进行通道 0-3 采样：

```
AD1CON1bits.FORM = 3;           // Data Output Format:Signed Fraction (Q15 format)
AD1CON1bits.SSRC = 2;           // Sample Clock Source:GP Timer starts conversion
AD1CON1bits.ASAM = 1;           // Sampling begins immediately after conversion
AD1CON1bits.AD12B = 0;          // 10-bit ADC operation
AD1CON1bits.SIMSAM = 0;         // Samples multiple channels sequentially

AD1CON2bits.BUFM = 0;
AD1CON2bits.CSCNA = 1;          // Scan CH0+ Input Selections during Sample A bit
AD1CON2bits.CHPS = 0;           // Converts CH0

AD1CON3bits.ADRC = 0;           // ADC Clock is derived from Systems Clock
AD1CON3bits.ADCS = 63;          // ADC Conversion Clock

//AD1CHS0:A/D Input Select Register
AD1CHS0bits.CH0SA = 0;          // MUXA +ve input selection (AIN0) for CH0
AD1CHS0bits.CH0NA = 0;          // MUXA -ve input selection (VREF-) for CH0

//AD1CHS123:A/D Input Select Register
AD1CHS123bits.CH123SA = 0;      // MUXA +ve input selection (AIN0) for CH1
AD1CHS123bits.CH123NA = 0;      // MUXA -ve input selection (VREF-) for CH1

//AD1CSSH/AD1CSSL:A/D Input Scan Selection Register
AD1CSSH = 0x0000;
AD1CSSL = 0x000F;               // Scan AIN0, AIN1, AIN2, AIN3 inputs
```

设置 Timer3 来触发 ADC1 转换：

```
TMR3 = 0x0000;
PR3 = 4999; // Trigger ADC1 every 125usec
IFS0bits.T3IF = 0;              // Clear Timer3 interrupt
IEC0bits.T3IE = 0;              // Disable Timer3 interrupt

T3CONbits.TON = 1;              //Start Timer3
```

将 DMA 通道 5 设置为外设间接寻址：

```
struct
{
    unsigned int Adc1Ch0[8];
    unsigned int Adc1Ch1[8];
    unsigned int Adc1Ch2[8];
    unsigned int Adc1Ch3[8];
} BufferA __attribute__((space(dma)));

struct
{
    unsigned int Adc1Ch0[8];
    unsigned int Adc1Ch1[8];
    unsigned int Adc1Ch2[8];
    unsigned int Adc1Ch3[8];
} BufferB __attribute__((space(dma)));

int BufferA[4][8] __attribute__((space(dma), aligned(64)));
BufferB[4][8] __attribute__((space(dma), aligned(64)));

DMA5CONbits.AMODE = 2;          // Configure DMA for Peripheral indirect mode
DMA5CONbits.MODE = 2;           // Configure DMA for Continuous Ping-Pong mode
DMA5PAD = (volatile unsigned int)&ADC1BUF0; // Point DMA to ADC1BUF0
DMA5CNT = 31;                   // 32 DMA request (4 buffers, each with 8 words)
DMA5REQ = 13;                   // Select ADC1 as DMA Request source
DMA5STA = __builtin_dmaoffset(BufferA);
DMA5STB = __builtin_dmaoffset(BufferB);

IFS3bits.DMA5IF = 0;            //Clear the DMA interrupt flag bit
IEC3bits.DMA5IE = 1;            //Set the DMA interrupt enable bit

DMA5CONbits.CHEN=1;             // Enable DMA
```

例 22-7： 用于外设间接寻址模式下的 ADC 和 DMA 的代码 （续）

设置 DMA 通道 5 中断处理程序：

```
unsigned int DmaBuffer = 0;

void __attribute__((__interrupt__)) _DMA5Interrupt(void)
{
    // Switch between Primary and Secondary Ping-Pong buffers
if(DmaBuffer == 0)
    {
        ProcessADCSamples(BufferA.Adc1Ch0);
        ProcessADCSamples(BufferA.Adc1Ch1);
        ProcessADCSamples(BufferA.Adc1Ch2);
        ProcessADCSamples(BufferA.Adc1Ch3);
    }
    else
    {
        ProcessADCSamples(BufferB.Adc1Ch0);
        ProcessADCSamples(BufferB.Adc1Ch1);
        ProcessADCSamples(BufferB.Adc1Ch2);
        ProcessADCSamples(BufferB.Adc1Ch3);
    }

    DmaBuffer ^= 1;

    IFS3bits.DMA5IF = 0;        //Clear the DMA5 Interrupt Flag
}
```

为 DMA 操作设置 ADC1：

```
AD1CON1bits.ADDMABM = 0;        // DMA buffers are built in scatter/gather mode
AD1CON2bits.SMPI    = 3;        // 4 ADC buffers
AD1CON4bits.DMABL   = 3;        // Each buffer contains 8 words

IFS0bits.AD1IF      = 0;        // Clear the A/D interrupt flag bit
IEC0bits.AD1IE      = 0;        // Do Not Enable A/D interrupt
AD1CON1bits.ADON    = 1;        // Turn on the A/D converter
```

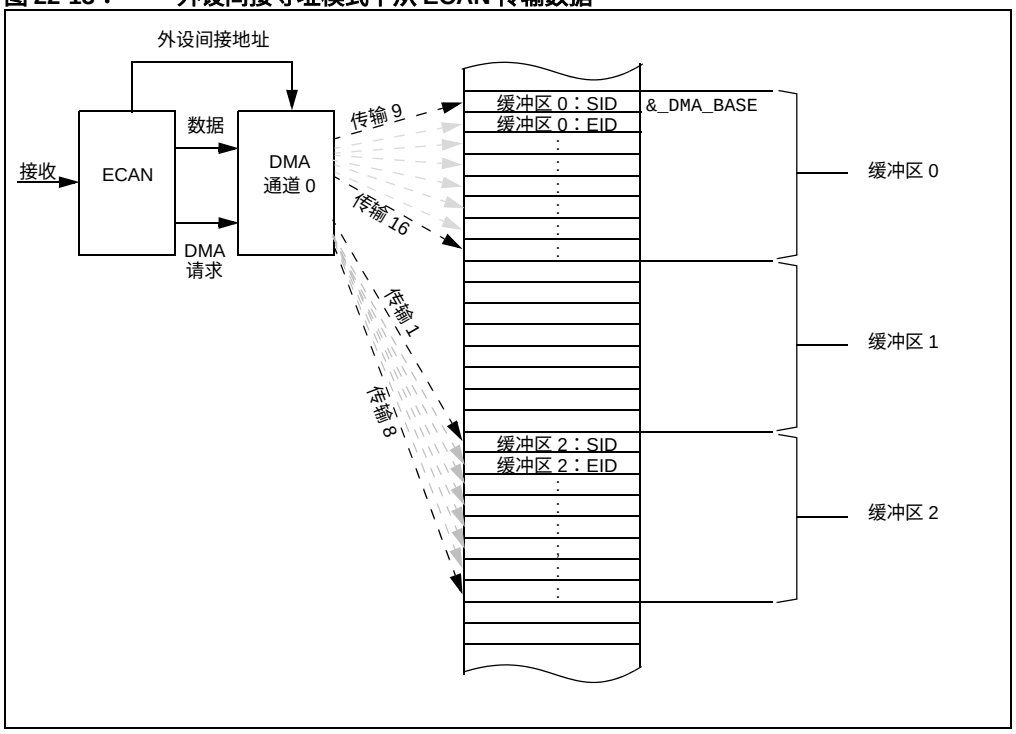
22.6.6.2 ECAN 对于 DMA 地址生成的支持

外设间接寻址也可用于 ECAN 模块，让 ECAN 定义更具体的寻址功能。当 dsPIC33F 器件通过 CAN 总线过滤并接收报文时，报文可分为两类：

- 必须处理的接收报文
- 不进行处理而必须转发给其他 CAN 节点的接收报文

对于第一种情况，必须重新构造接收到的报文，分配到各为 8 个字的缓冲区中，然后由用户应用程序进行处理。由于 DMA RAM 中有多个 ECAN 缓冲区，让 ECAN 外设为传入 （或外发）数据生成 RAM 地址会比较简单，如图 22-13 所示。在该示例中，缓冲区 2 先接收数据，然后是缓冲区 0。 ECAN 模块会生成目标地址，以将数据正确放入 DMA RAM （外设间接寻址）。

图 22-13： 外设间接寻址模式下从 ECAN 传输数据



如前面所提到，在由用户应用程序初始化 DPSRAM 起始地址偏移寄存器 （DMAxSTA 和 DMAxSTB），且 DMA 以外设间接寻址模式工作时，必须特别注意为外设保留的最低有效位的位数。对于 ECAN，位数取决于 ECAN FIFO 控制寄存器 （CiFCTRL）的 DMA 缓冲区大小位（DMABS<2:0>）定义的 ECAN 缓冲区数量。

例如，如果 ECAN 模块通过将 DMABS<2:0> 位设置为 3 而保留 12 个缓冲区，将有 12 个各为 8 个字的缓冲区，总共 96 个字 （192 字节）。这意味着写入 DMAxSTA 和 DMAxSTB 寄存器的地址偏移必须将 8 个（2^8 位 = 256 字节）最低有效位设置为 0。如果使用 MPLAB C30 编译器来初始化 DMAxSTA 寄存器，则必须通过数据属性来指定正确的数据对齐。对于以上示例，例 22-8 中的代码可以正确初始化 DMAxSTA 寄存器。

例 22-8： 使用 MPLAB® C30 时的 DMA 缓冲区对齐

```
int  BufferA[12][8] __attribute__((space(dma),aligned(256)));

DMA0STA = __builtin_dmaoffset(&BufferA[0][0]);
```

例 22-9 给出了该配置的代码示例。

但是，可能并不总是需要处理传入的报文。例如，在某些汽车应用中，可以简单地将接收的报文转发到另一个节点，而不是由 CPU 进行处理。这种情况下，不需要在存储器中对接收的缓冲区进行数据分选，可以在它们变为可用时进行转发。

这种数据传输模式可以使用配置为带后递增的寄存器间接寻址模式的 DMA 实现。图 22-14 给出了这种情况的图示。

例 22-9： 用于外设间接寻址模式下的 ECAN 和 DMA 的代码

为 ECAN1 设置两个过滤器：

```
/* Initialize ECAN clock first.See ECAN section for example code */


C1CTRL1bits.WIN = 1;          // Enable filter window
C1FEN1bits.FLTEN0 = 1;        // Filter 0 is enabled
C1FEN1bits.FLTEN1 = 1;        // Filter 1 is enabled
C1BUFPNT1bits.F0BP = 0;       // Filter 0 points to Buffer0
C1BUFPNT1bits.F1BP = 2;       // Filter 1 points to Buffer2

C1RXF0SID = 0xFFEA;           // Filter 0 configuration
C1RXF0EID = 0xFFFF;

C1RXF1SID = 0xFFEB;           // Filter 1 configuration
C1RXF1EID = 0xFFFF;

C1FMSKSEL1bits.F0MSK = 0;     // Mask 0 used for both filters
C1FMSKSEL1bits.F1MSK = 0;     // Mask 0 used for both filters
C1RXM0SID = 0xFFEB;
C1RXM0EID = 0xFFFF;

C1FCTRLbits.DMABS = 3;        // 12 buffers in DMA RAM
C1FCTRLbits.FSA   = 3;        // FIFO starts from TX/RX Buffer 3

C1CTRL1bits.WIN     = 0;
C1TR01CONbits.TXEN0 = 0;      // Buffer 0 is a receive buffer
C1TR23CONbits.TXEN2 = 0;      // Buffer 2 is a receive buffer

C1TR01CONbits.TX0PRI = 0b11;  //High Priority
C1TR01CONbits.TX1PRI = 0b10;  //Intermediate High Priority

C1CTRL1bits.REQOP = 0;// Enable Normal Operation Mode
```

将 DMA 通道 0 设置为外设间接寻址：

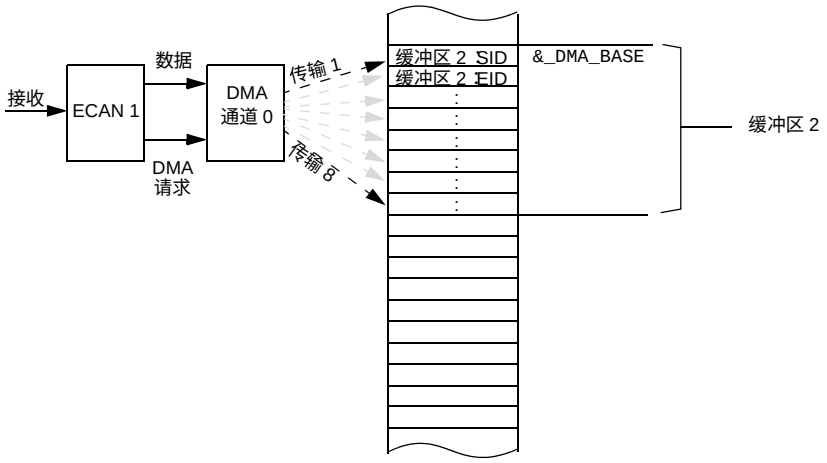
```
unsigned int Ecan1Rx[12][8] __attribute__((space(dma)));// 12 buffers, 8
words each

DMA0CONbits.AMODE = 2;        // Continuous mode, single buffer
DMA0CONbits.MODE = 0;         // Peripheral Indirect Addressing

DMA0PAD = (volatile unsigned int) &C1RXD;    // Point to ECAN1 RX register
DMA0STA = __builtin_dmaoffset(Ecan1Rx);      // Point DMA to ECAN1 buffers

DMA0CNT = 7;                  // 8 DMA request (1 buffer, each with 8 words)
DMA0REQ = 0x22;               // Select ECAN1 RX as DMA Request source

IEC0bits.DMA0IE = 1;          // Enable DMA Channel 0 interrupt
DMA0CONbits.CHEN = 1;         // Enable DMA Channel 0
```

设置 DMA 中断处理程序：

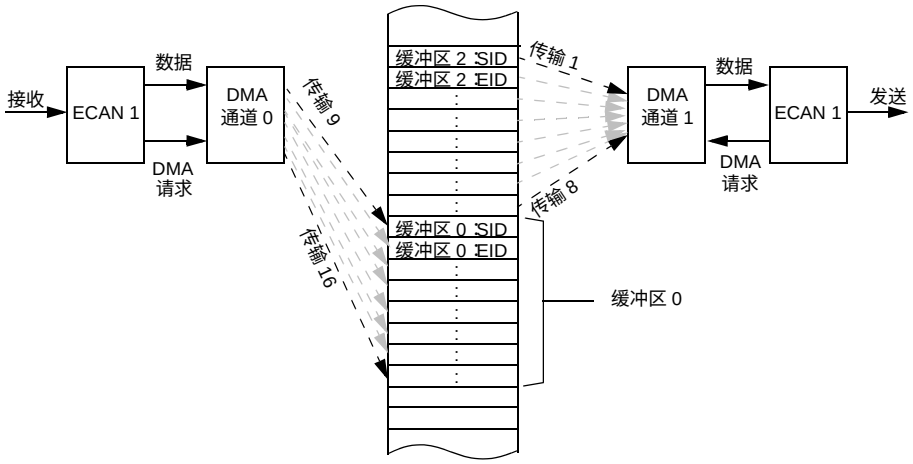
```
void __attribute__((__interrupt__)) _DMA0Interrupt(void)
{
   ProcessData(Ecan1Rx[C1VECbits.ICODE]);    // Process received buffer;

   IFS0bits.DMA0IF = 0;       // Clear the DMA0 Interrupt Flag;
}
```

图 22-14： 寄存器间接寻址模式下从 ECAN 传输数据

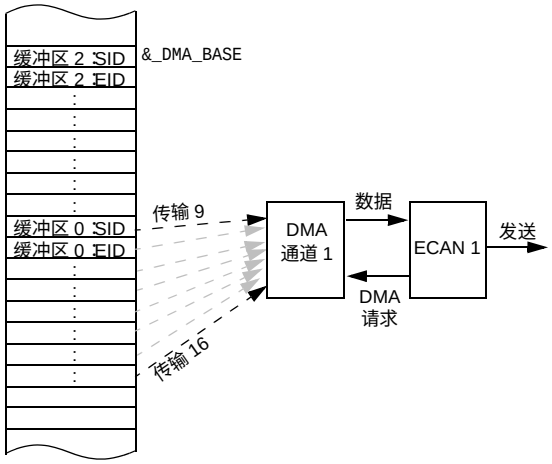
A 接收缓冲区 2



B 接收缓冲区 0 和发送缓冲区 2



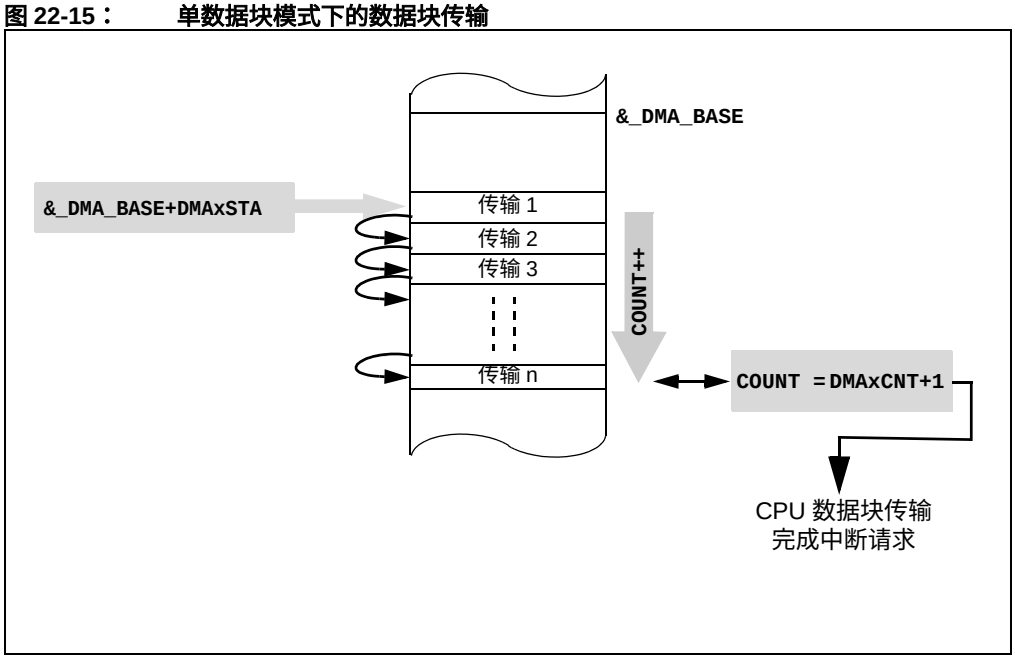
C 发送缓冲区 0



22.6.7 单数据块模式

在不需要进行重复数据传输时，应用程序使用单数据块模式。单数据块模式通过将 DMA 通道控制 （DMAxCON）寄存器中的工作模式选择位 （MODE<1:0>）设置为 x1 来进行选择。在该模式下，当整个数据块被传送时 （数据块长度由 DMAxCNT 定义），将会检测到数据块结束，并自动禁止通道 （即， DMA 通道控制 （DMAxCON）寄存器中的 CHEN 位由硬件清零）。图 22-15 给出了单数据块模式的图示。

图 22-15： 单数据块模式下的数据块传输



如果 DMA 通道控制 （DMAxCON）寄存器中的 HALF 位置 1，则当数据块传输完成一半时，DMAxIF 位会置 1（如果应用程序允许 DMA 中断，则产生中断），并且通道保持使能。当整个数据块传输完成时，不会置 1 中断标志，通道会被自动禁止。关于如何设置 DMA 通道对于半数据块传输和全数据块传输产生中断的信息，请参见**第 22.6.3 节 “ 全数据块或半数据块传输中断 ”**。

如果通过将 DMAxCON 中的 CHEN 置 1 而重新使能通道，则数据块传输从起始地址开始进行，起始地址由 DPSRAM 起始地址偏移 （DMAxSTA 和 DMAxSTB）寄存器提供。例 22-10 给出了单数据块操作的代码示例。

例 22-10： 用于单数据块模式下的 UART 和 DMA 的代码

设置 UART 进行接收和发送：

```
#define FCY      40000000
#define BAUDRATE 9600
#define BRGVAL   ((FCY/BAUDRATE)/16)-1

U2MODEbits.STSEL = 0;      // 1-stop bit
U2MODEbits.PDSEL = 0;      // No Parity, 8-data bits
U2MODEbits.ABAUD = 0;      // Autobaud Disabled

U2BRG = BRGVAL;// BAUD Rate Setting for 9600

U2STAbits.UTXISEL0 = 0;    // Interrupt after one TX character is transmitted
U2STAbits.UTXISEL1 = 0;
U2STAbits.URXISEL = 0;     // Interrupt after one RX character is received

U2MODEbits.UARTEN   = 1;  // Enable UART
U2STAbits.UTXEN     = 1;  // Enable UART TX
```

例 22-10： 用于单数据块模式下的 UART 和 DMA 的代码 （续）

设置 DMA 通道 0 在单数据块、单缓冲区模式下发送数据：

```
unsigned int BufferA[8] __attribute__((space(dma)));
unsigned int BufferB[8] __attribute__((space(dma)));


DMA0CON = 0x2001;             // One-Shot, Post-Increment, RAM-to-Peripheral
DMA0CNT = 7;                  // 8 DMA requests
DMA0REQ = 0x001F;             // Select UART2 Transmitter

DMA0PAD = (volatile unsigned int) &U2TXREG;
DMA0STA = __builtin_dmaoffset(BufferA);

IFS0bits.DMA0IF = 0;          // Clear DMA Interrupt Flag
IEC0bits.DMA0IE = 1;          // Enable DMA interrupt
```

设置 DMA 通道 1 在连续乒乓模式下接收数据：

```
DMA1CON = 0x0002;             // Continuous, Ping-Pong, Post-Inc., Periph-RAM
DMA1CNT = 7;                  // 8 DMA requests
DMA1REQ = 0x001E;             // Select UART2 Receiver

DMA1PAD = (volatile unsigned int) &U2RXREG;
DMA1STA = __builtin_dmaoffset(BufferA);
DMA1STB = __builtin_dmaoffset(BufferB);


IFS0bits.DMA1IF = 0;          // Clear DMA interrupt
IEC0bits.DMA1IE = 1;          // Enable DMA interrupt
DMA1CONbits.CHEN = 1;         // Enable DMA Channel
```

设置 DMA 中断处理程序：

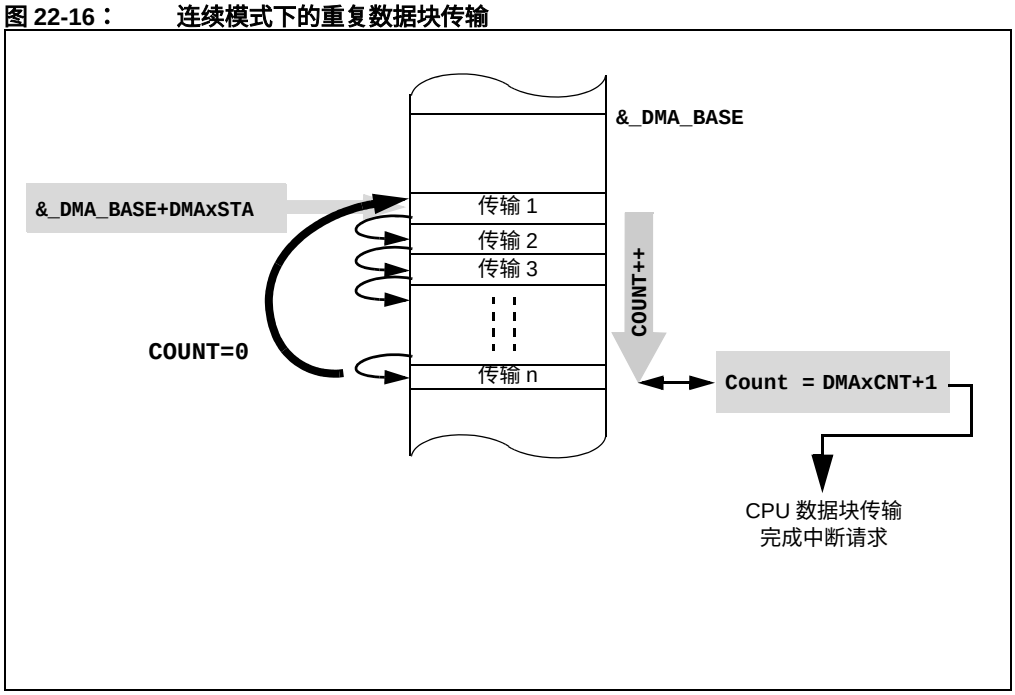
```
void __attribute__((__interrupt__)) _DMA0Interrupt(void)
{
   IFS0bits.DMA0IF = 0;       // Clear the DMA0 Interrupt Flag;
}


void __attribute__((__interrupt__)) _DMA1Interrupt(void)
{
   static unsigned int BufferCount = 0;      // Keep record of which buffer
                                             // contains RX Data

   if(BufferCount == 0)
   {
    DMA0STA = __builtin_dmaoffset(BufferA);  // Point DMA 0 to data
                                             // to be transmitted
   }
   else
 {
    DMA0STA = __builtin_dmaoffset(BufferB);  // Point DMA 0 to data
                                             // to be transmitted
   }

   DMA0CONbits.CHEN = 1;      // Enable DMA0 Channel
   DMA0REQbits.FORCE = 1;     // Manual mode:Kick-start the 1st transfer

   BufferCount ^= 1;
   IFS0bits.DMA1IF = 0;       // Clear the DMA1 Interrupt Flag
}
```

22.6.8 连续模式

需要在程序的整个工作周期中进行重复数据传输时，应用程序使用连续模式。

该模式通过将 DMA 通道控制 （DMAxCON）寄存器中的工作模式选择位 （MODE<1:0>）设置为 x0 来进行选择。在该模式下，当整个数据块被传送时 （数据块长度由 DMAxCNT 定义），会检测到数据块结束，而通道会保持使能。在最后一次数据传输中，DMA DPSRAM 地址会复位为（主） DPSRAM 起始地址偏移 A （DMAxSTA）寄存器的值。图 22-16 给出了连续模式的图示。

图 22-16： 连续模式下的重复数据块传输



如果 DMA 通道控制 （DMAxCON）寄存器中的 HALF 位置 1，则当数据块传输完成一半时，DMAxIF 位会置 1（如果允许 DMA 中断，则产生中断）。通道保持使能。当整个数据块传输完成时，不会置 1 中断标志，并且通道会保持使能。关于如何设置 DMA 通道对于半数据块传输和全数据块传输产生中断的信息，请参见**第 22.6.3 节 “ 全数据块或半数据块传输中断 ”**。

22.6.9 乒乓模式

乒乓模式使 CPU 可以处理一个缓冲区，同时第二个缓冲区供 DMA 通道进行操作。产生的效果就是 CPU 可以在整个 DMA 数据块传输时间中，处理 DMA 通道当前不使用的缓冲区。当然，对于给定大小的缓冲区，这种传输模式使所需的 DPSRAM 空间量加倍。

在所有 DMA 工作模式下，当使能 DMA 通道时，默认情况下会选择（主）DMA 通道 x DPSRAM 起始地址偏移 A （DMAxSTA）寄存器来生成初始的 DPSRAM 有效地址。当每次数据块传输完成， DMA 通道被重新初始化时，将从相同的 DMAxSTA 寄存器获取缓冲区起始地址。

在乒乓模式下，缓冲区起始地址从两个寄存器获取：

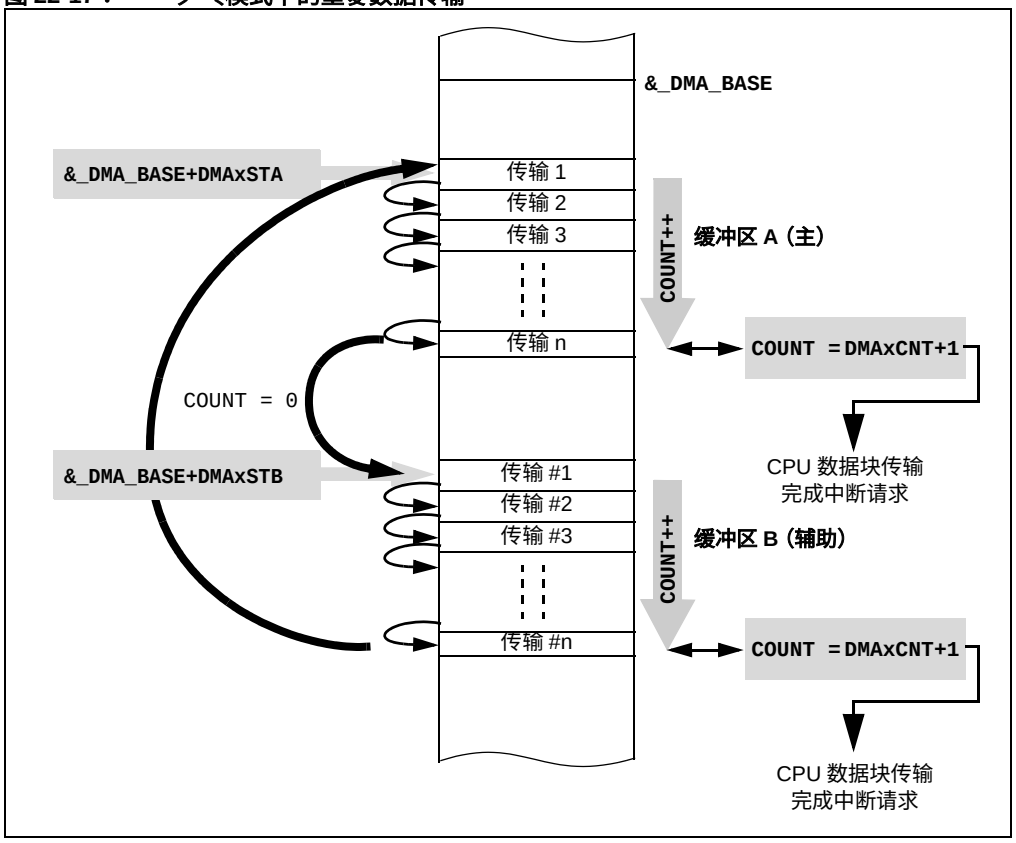
- 主缓冲区：DMA 通道 x DPSRAM 起始地址偏移 A （DMAxSTA）寄存器
- 辅助缓冲区：DMA 通道 x DPSRAM 起始地址偏移 B （DMAxSTB）寄存器

DMA 使用辅助缓冲区来交替进行数据块传输。当每次数据块传输完成， DMA 通道被重新初始化时，将从交替使用的寄存器获取缓冲区起始地址。

乒乓模式通过将 DMA 通道控制 （DMAxCON）寄存器中的工作模式选择位 （MODE<1:0>）设置为 1x 来进行选择。

如果在 DMA 工作于乒乓模式时选择了连续模式，则 DMA 会在传输完主缓冲区之后重新初始化为指向辅助缓冲区，然后传输辅助缓冲区。后续的数据块传输将在主缓冲区和辅助缓冲区之间交替进行。在传输完每个缓冲区之后会产生中断 （如果应用程序允许中断）。图 22-17 给出了乒乓模式下的连续操作的图示。例 22-11 以 DCI 模块作为示例，给出了用于乒乓操作的代码示例。

图 22-17： 乒乓模式下的重复数据传输



例 22-11： 用于连续乒乓操作时 DCI 和 DMA 的代码

设置 DCI 进行接收和发送：

```
#define FCY    40000000
#define FS     48000
#define FCSCK  64*FS
#define BCGVAL (FCY/(2*FS))-1

DCICON1bits.CSCKD = 0; // Serial Bit Clock (CSCK pin) is output
DCICON1bits.CSCKE = 0; // Data sampled on falling edge of CSCK
DCICON1bits.COFSD = 0; // Frame Sync Signal is output
DCICON1bits.UNFM = 0;  // Transmit '0's on a transmit underflow
DCICON1bits.CSDOM = 0; // CSDO pin drives '0's during disabled TX time slots
DCICON1bits.DJST = 0;  // TX/RX starts 1 serial clock cycle after frame sync pulse
DCICON1bits.COFSM = 1; // Frame Sync Signal set up for I2S mode

DCICON2bits.BLEN = 0;  // One data word will be buffered between interrupts
DCICON2bits.COFSG = 1; // Data frame has 2 words:LEFT & RIGHT samples
DCICON2bits.WS = 15;   // Data word size is 16 bits

DCICON3 = BCG_VAL;// Set up CSCK Bit Clock Frequency

TSCONbits.TSE0 = 1;    // Transmit on Time Slot 0
TSCONbits.TSE1 = 1;    // Transmit on Time Slot 1
RSCONbits.RSE0 = 1;    // Receive on Time Slot 0
RSCONbits.RSE1 = 1;    // Receive on Time Slot 1
```

设置 DMA 通道 0 在连续乒乓模式下发送数据：

```
unsigned int TxBufferA[16] __attribute__((space(dma)));
unsigned int TxBufferB[16] __attribute__((space(dma)));

DMA0CON = 0x2002;      // Ping-Pong, Continous, Post-Increment, RAM-to-Peripheral
DMA0CNT = 15;          // 15 DMA requests
DMA0REQ = 0x003C;      // Select DCI as DMA Request source

DMA0PAD = (volatile unsigned int) &TXBUF0;
DMA0STA = __builtin_dmaoffset(TxBufferA);
DMA0STB = __builtin_dmaoffset(TxBufferB);

IFS0bits.DMA0IF = 0;  // Clear DMA Interrupt Flag
IEC0bits.DMA0IE = 1;  // Enable DMA interrupt
DMA0CONbits.CHEN = 1;  // Enable DMA Channel
```

设置 DMA 通道 1 在连续乒乓模式下接收数据：

```
unsigned int RxBufferA[16] __attribute__((space(dma)));
unsigned int RxBufferB[16] __attribute__((space(dma)));


DMA1CON = 0x0002;      // Continuous, Ping-Pong, Post-Inc., Periph-RAM
DMA1CNT = 15;          // 16 DMA requests
DMA1REQ = 0x003C;      // Select DCI as DMA Request source

DMA1PAD = (volatile unsigned int) &RXBUF0;
DMA1STA = __builtin_dmaoffset(RxBufferA);
DMA1STB = __builtin_dmaoffset(RxBufferB);

IFS0bits.DMA1IF = 0;// Clear DMA interrupt
IEC0bits.DMA1IE = 1;// Enable DMA interrupt
DMA1CONbits.CHEN = 1;// Enable DMA Channel
```

例 22-11： 用于连续乒乓操作时 DCI 和 DMA 的代码 （续）

设置 DMA 中断处理程序：

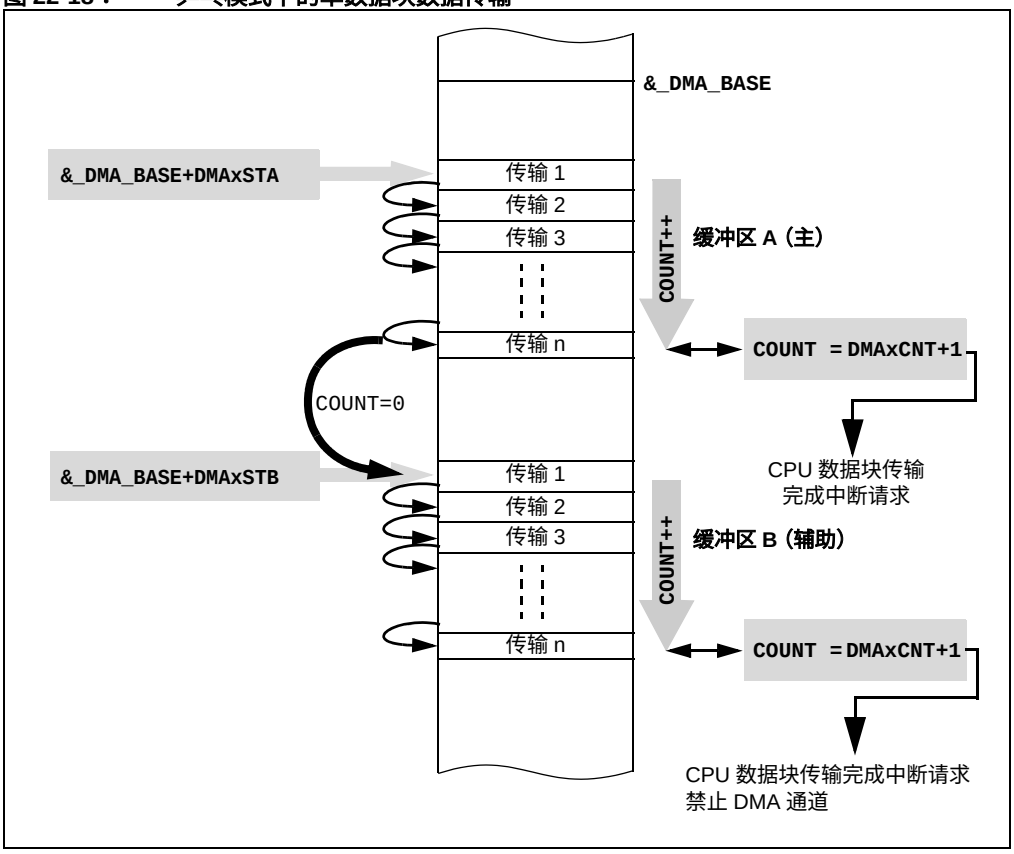
```
void __attribute__((__interrupt__)) _DMA0Interrupt(void)
{
   static unsigned int TxBufferCount = 0;// Keep record of which buffer
                                         // has RX Data

   if(BufferCount == 0)
   {
    /* Notify application that TxBufferA has been transmitted */
   }
   else
   {
    /* Notify application that TxBufferB has been transmitted */
   }

   BufferCount ^= 1;
   IFS0bits.DMA0IF = 0;                  // Clear the DMA0 Interrupt Flag;
}


void __attribute__((__interrupt__)) _DMA1Interrupt(void)
{
   static unsigned int RxBufferCount = 0; // Keep record of which buffer
                                         // has RX Data

   if(BufferCount == 0)
   {
    /* Notify application that RxBufferA has been received */
   }
   else
   {
    /* Notify application that RxBufferB has been received */   }

   BufferCount ^= 1;
   IFS0bits.DMA1IF = 0;                  // Clear the DMA1 Interrupt Flag
}
```

使能 DCI：

```
/* Force First two words to fill-in TX buffer/shift register */
DMA0REQbits.FORCE = 1;
while(DMA0REQbits.FORCE == 1);

DMA0REQbits.FORCE = 1;
while(DMA0REQbits.FORCE == 1);

DCICON1bits.DCIEN = 1;                   // Enable DCI
```

如果在 DMA 工作于乒乓模式时选择了单数据块模式，则 DMA 会在传输完主缓冲区之后重新初始化为指向辅助缓冲区，然后传输辅助缓冲区。但是，后续的数据块传输将不会发生，因为 DMA 通道会禁止它自身。图 22-18 给出了乒乓模式下的单数据块数据传输的图示。

图 22-18： 乒乓模式下的单数据块数据传输



22.6.10 手动传输模式

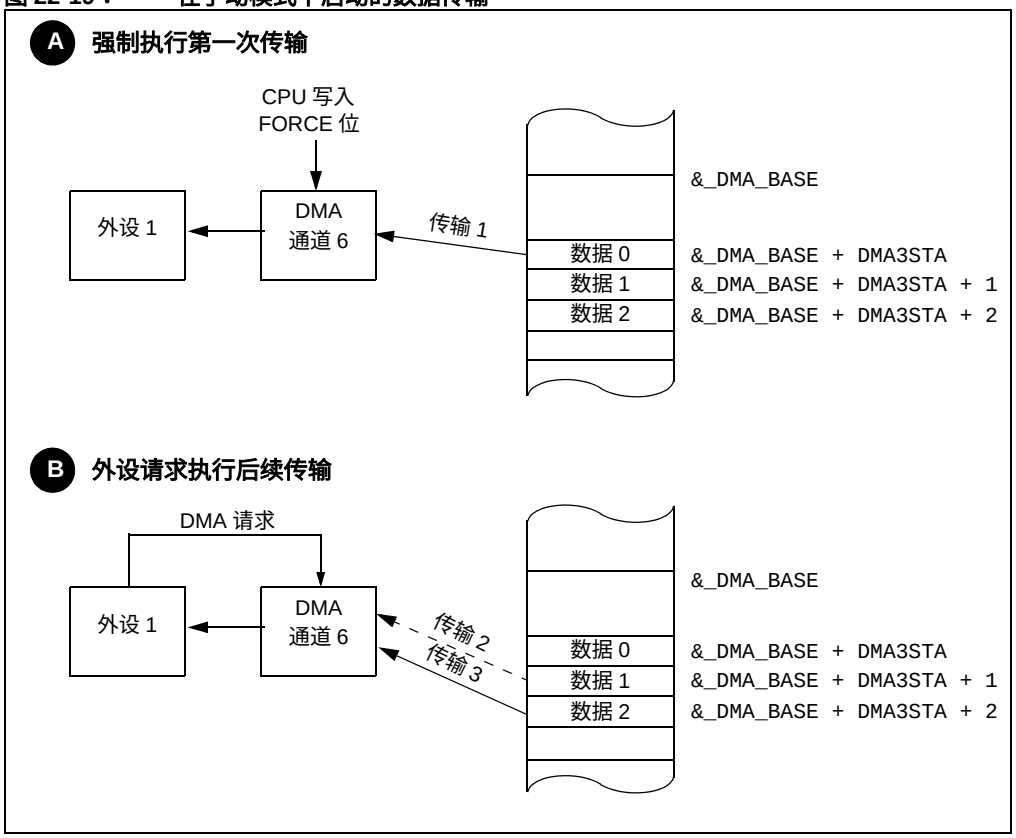
对于使用 DMA 控制器向 DPSRAM 发送数据的外设， DMA 数据传输在 DMA 通道和外设初始化之后自动开始。当外设准备好向 DPSRAM 传送数据时，它会发出 DMA 请求。如果此时还需要将数据发送到外设，可以使用相同的 DMA 请求来激活另一个通道，使之从 DPSRAM 读取数据并将数据写入外设。

另一方面，如果应用程序只需要将数据发送到外设 （从 DPSRAM 缓冲区），则可能需要执行初始 （手动）操作来将数据装入外设，以启动传输过程 （见**第 22.7 节 “ 启动 DMA 传输 ”**）。该过程可以使用传统软件启动。但是，更方便的方式是通过将选定 DMA 通道中的某个位置 1 来简单地模拟通道 DMA 请求。DMA 通道会像任何其他请求一样处理强制请求，并传输第一个数据元素来启动序列。当外设准备好接收下一个数据时，它会发送常规 DMA 请求，然后 DMA 会发送下一个数据元素。图 22-19 中给出了该过程的图示。

手动 DMA 请求可以通过将 DMA 通道 x IRQ 选择 （DMAxREQ）寄存器中的 FORCE 位置 1 来产生。置 1 之后， FORCE 位不能由用户应用程序清零。当强制的 DMA 传输完成时，它必须由硬件清零。根据 FORCE 位置 1 的时间，适用以下特殊条件：

- 在 DMA 传输正在进行时将 FORCE 位置 1 没有任何影响，将被忽略。
- 在配置通道 x 时将 FORCE 位置 1 （即，在配置 DMA 通道的同一写操作期间将 FORCE 位置 1）会导致不可预测的行为，应当避免。
- 如果在某个外设中断请求正在等待处理时试图将 FORCE 位置 1 （对于该通道），则会优先考虑基于中断的请求，放弃该尝试。但是，这会产生错误条件，即会将 DMA 控制器状态 0 （DMACS0）寄存器中的 DMA RAM 写冲突标志位 （XWCOLx）和外设写冲突标志位 （PWCOLx）置 1。更多详细信息，请参见**第 22.10 节 “ 数据写冲突 ”**。

图 22-19： 在手动模式下启动的数据传输



22.6.11 空数据写模式

在需要连续接收数据而无需发送任何数据的应用中 （例如 SPI），空数据写模式最为有用。

SPI 本质上是一个简单的移位寄存器，它在每个时钟周期移入并移出一个数据位。但是，当 SPI 配置为主模式 （即， SPI 要作为时钟源），但只对接收的数据感兴趣时，会出现一种例外情况。在这种情况下，必须向 SPI 数据寄存器中写入内容以启动 SPI 数据时钟并接收外部数据。

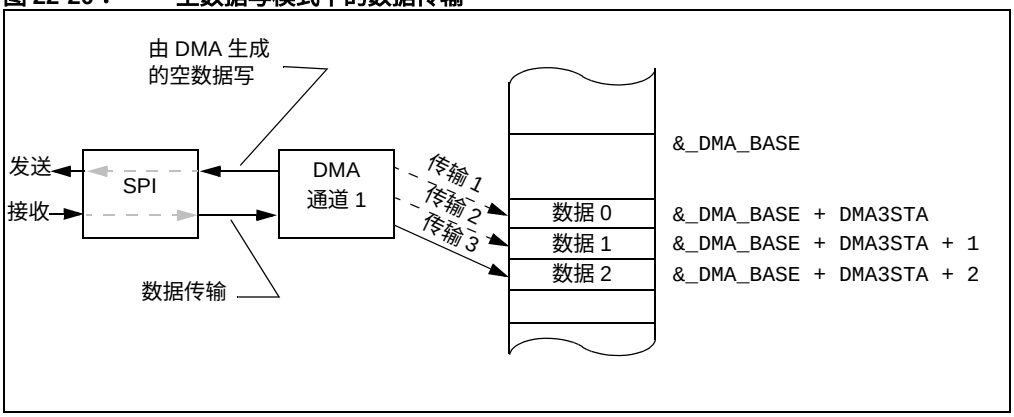
可以分配两个 DMA 通道，一个用于接收数据，另一个只是向 SPI 中输入空数据或零数据。但是，更高效的解决方案是使用 DMA 空数据写模式，该模式会在接收到每个数据元素并通过为外设数据读操作配置的 DMA 通道传输之后，自动向 SPI 数据寄存器中写入空值。

如果将 DMA 通道 x 控制 （DMAxCON）寄存器中的空数据外设写模式选择位 （NULLW）置 1，并将 DMA 通道配置为从外设读数据，则 DMA 通道会在外设数据读操作的同一周期中对外设地址执行空（全 0）写操作。该写操作通过外设总线与对 DPSRAM 的（数据）写操作（通过 DPSRAM 总线）同时发生。图 22-20 给出了该模式的图示。

在该模式下的正常操作期间，只有为了响应外设 DMA 请求 （即，在接收到数据，并可用于传输之后）才会发生空数据写操作。第一个字的接收需要通过对外设执行初始 CPU 写操作来启动，之后 DMA 会处理所有后续的外设（空）数据写操作。即，CPU 空写操作会启动 SPI（主器件）发送 / 接收数据，这最终会产生一个 DMA 请求来传送新接收的数据。

或者，也可以使用强制 DMA 传输来启动该过程。但是，这需要另外执行一次冗余的外设读操作（数据无效），并调整关联的 DPSRAM 指针，这是必须要考虑的。

图 22-20： 空数据写模式下的数据传输



例 22-12： 空数据写模式下的 SPI 和 DMA

设置 SPI 为主模式：

```
SPI1CON1bits.MODE16 = 1;      // Communication is word-wide (16 bits)
SPI1CON1bits.MSTEN = 1;       // Master Mode Enabled
SPI1STATbits.SPIEN = 1;       // Enable SPI Module
```

将 DMA 通道 1 设置为空数据写模式：

```
unsigned int BufferA[16] __attribute__((space(dma)));
unsigned int BufferB[16] __attribute__((space(dma)));

DMA1CON = 0x0802;             // Null Write, Continuous, Ping-Pong,
                              // Post-Increment, Periph-to-RAM
DMA1CNT = 15;                 // Transfer 16 words at a time
DMA1REQ = 0x000A;             // Select SPI1 as DMA request source

DMA1STA = __builtin_dmaoffset(BufferA);
DMA1STB = __builtin_dmaoffset(BufferB);
DMA1PAD = (volatile unsigned int) &SPI1BUF;

IFS0bits.DMA1IF = 0;
IEC0bits.DMA1IE = 1;          // Enable DMA interrupt
DMA1CONbits.CHEN = 1;         // Enable DMA Channel

DMA1REQbits.FORCE = 1;        // Force First word after Enabling SPI
```

设置 DMA 中断处理程序：

```
void __attribute__((__interrupt__)) _DMA1Interrupt(void)
{
   static unsigned int BufferCount = 0;  // Keep record of which buffer
                                         // contains RX Data

   if(BufferCount == 0)
   {
    ProcessRxData(BufferA);              // Process received SPI data in
                                         // DMA RAM Primary buffer
   }
   else
   {
    ProcessRxData(BufferB);              // Process received SPI data in
                                         // DMA RAM Secondary buffer
   }

   BufferCount ^= 1;
   IFS0bits.DMA1IF = 0;                  // Clear the DMA1 Interrupt Flag
}
```

22.7 启动 DMA 传输

在 DMA 传输开始之前，必须通过将 DMAxCON 寄存器中的 CHEN 位置 1 来使能 DMA 通道。在 DMA 通道处于工作状态时，可以通过禁止该通道（CHEN = 0），然后重新使能它（CHEN = 1）来重新初始化。该过程会将 DMA 传输计数复位为 0，并将工作的 DMA 缓冲区设置为主缓冲区。

正确初始化 DMA 通道和外设后，DMA 传输会在外设准备好传送数据并发出 DMA 请求时立即开始。但是，一些外设只有在特定条件下才会发出 DMA 请求（因而不会启动 DMA 传输）。在这些情况下，可能需要应用不同 DMA 模式和过程的组合，以启动 DMA 传输。

22.7.1 使用串行外设接口 （SPI）启动 DMA

与 SPI 外设之间的 DMA 传输的启动取决于 SPI 数据方向以及从模式或主模式：

- **在主模式下仅发送数据——**在该配置中，只有发送第一个 SPI 数据块之后，才会发出 DMA 请求。要启动 DMA 传输，用户应用程序必须先使用 DMA 手动传输模式发送数据，或者必须不依赖于 DMA 先向 SPI 缓冲区 （SPIxBUF）中写入数据。
- **在主模式下仅接收数据——**在该配置中，只有接收到第一个 SPI 数据块之后，才会发出 DMA 请求。但是，在主模式下，只有 SPI 先发送数据之后才会接收到数据。要启动 DMA 传输，用户应用程序必须使用 DMA 空数据写模式，并启动 DMA 手动传输模式。
- **在主模式下接收和发送数据——**在该配置中，只有接收到第一个 SPI 数据块之后，才会发出 DMA 请求。但是，在主模式下，只有 SPI 发送数据之后才会接收到数据。要启动 DMA 传输，用户应用程序必须先使用 DMA 手动传输模式发送数据，或者必须不依赖于 DMA 先向 SPI 缓冲区 （SPIxBUF）中写入数据。
- **在从模式下仅发送数据——**在该配置中，只有接收到第一个 SPI 数据块之后，才会发出 DMA 请求。要启动 DMA 传输，用户应用程序必须先使用 DMA 手动传输模式发送数据，或者必须不依赖于 DMA 先向 SPI 缓冲区 （SPIxBUF）中写入数据。
- **在从模式下仅接收数据——**该配置会在第一个 SPI 数据到达之后立即产生 DMA 请求，所以用户不需要执行任何特殊步骤来启动 DMA 传输。
- **在从模式下接收和发送数据——**在该配置中，只有接收到第一个 SPI 数据块之后，才会发出 DMA 请求。要启动 DMA 传输，用户应用程序必须先使用 DMA 手动传输模式发送数据，或者必须不依赖于 DMA 先向 SPI 缓冲区 （SPIxBUF）中写入数据。

22.7.2 使用数据转换器接口 （DCI）启动 DMA

不同于其他串行外设， DCI 会在使能之后立即开始传输 （假设它是主器件）。它会持续地向它所连接的外部编解码器发送同步数据帧。使能 DCI 之前，必须：

- 按照**第 22.5.2 节 “ 外设配置设置 ”** 中所述配置 DCI。
- 如果连接到立体声编解码器，则使用 DMA 手动传输模式来启动前两次数据传输。
  - 将 DMAxREQ 寄存器中的 FORCE 位置 1，以传输 DCI 左通道采样。
  - 再次将 FORCE 位置 1，以传输 DCI 右通道采样。

完成这些步骤之后，使能 DCI 外设 （见例 22-11）。

22.7.3 使用 UART 启动 DMA

UART 接收器会在接收到数据之后立即发出 DMA 请求。用户应用程序不需要执行任何特殊步骤来启动 DMA 传输。 UART 发送器会在 UART 和发送器使能时立即发出 DMA 请求。这意味着必须在 UART 和发送器之前先初始化并使能 DMA 通道和缓冲区。

请确保按**第 22.5.2 节 “ 外设配置设置 ”**（表 22-2）中所述配置 UART。

或者，可以在使能 DMA 通道之前先使能 UART 和 UART 发送器。这种情况下，UART 发送器的 DMA 请求将丢失，用户应用程序必须通过将 DMAxREQ 寄存器中的 FORCE 位置 1 来发出 DMA 请求，以启动 DMA 传输。

22.8 DMA 通道仲裁和溢出

每个 DMA 通道都具有固定的优先级。通道 0 优先级最高，通道 7 优先级最低。当某个源请求 DMA 传输时，关联的 DMA 通道会锁存请求。 DMA 控制器用作仲裁器。如果没有其他传输正在进行或正在等待，控制器会将总线资源授予发出请求的 DMA 通道。DMA 控制器会确保只有当前 DMA 通道完成其操作时，才会授予其他 DMA 通道任何资源。

如果有多个 DMA 请求到达或正在等待，DMA 控制器内的优先级逻辑会将资源授予优先级最高的 DMA 通道，让它完成操作。所有其他 DMA 请求保持等待处理状态，直到选定 DMA 传输完成。如果在当前 DMA 传输正在进行时，有另一个 DMA 请求到达，则也会将它与所有等待的 DMA 请求一起比较优先级，确保在当前 DMA 传输完成之后始终处理优先级最高的请求。

由于 DMA 通道需要比较优先级，所以可能会有 DMA 请求不会立即被处理，进入等待处理状态。该请求将保持等待处理状态，直到所有优先级更高的通道都得到处理。如果在 DMA 控制器清除原始 DMA 请求之前，有另一个中断到达，并且中断的类型与正在等待的中断的类型相同，则发生数据溢出。

数据溢出的定义为：在 DMA 传送先前数据之前，新数据进入外设数据缓冲区。一些 DMA 就绪外设可以检测数据溢出并发出 CPU 中断 （如果允许相应的外设错误中断），如表 22-5 所示。

表 22-5： DMA 就绪外设的溢出处理

DMA 就绪外设	数据溢出处理
串行外设接口 （SPI）	等待通过 DMA 通道传送的数据不会被其他传入数据覆盖。后续的传入数据会丢失， SPI 状态 （SPIxSTAT）寄存器中的 SPI 接收溢出 （SPIROV）位会置 1。如果将中断控制器的中断允许控制 （IECx）寄存器中的 SPI 错误中断允许 （SPIxEIE）位置 1，则还会产生 SPIx 故障中断。
UART	等待通过 DMA 通道传送的数据不会被其他传入数据覆盖。后续的传入数据会丢失， UART 状态 （UxSTA）寄存器中的溢出错误 （OERR）位会置 1。如果将中断控制器的中断允许控制 （IECx）寄存器中的 UART 错误中断允许 （UxEIE）位置 1，则还会产生 UARTx 错误中断。
数据转换器接口 （DCI）	等待通过 DMA 通道传送的数据会被其他传入数据覆盖，并且 DCI 状态 （DCISTAT）寄存器中的接收溢出（ROV）位会置 1。如果将中断控制器的中断允许控制（IEC0）寄存器中的 DCI 错误中断允许 （DCIEIE）位置 1，则还会产生 DCI 故障中断。
10 位 /12 位模数转换器 （ADC）	等待通过 DMA 通道传送的数据会被其他传入数据覆盖。 ADC 不会检测溢出条件。
其他 DMA 就绪外设	不会发生数据溢出。

只有 DMA 从外设向 DPSRAM 传送数据时，才能在硬件中检测到数据溢出。从 DPSRAM 到外设的 DMA 数据传输 （例如，基于缓冲区空中断）始终都会执行。产生的任何 DPSRAM 数据溢出都必须使用软件进行检测。重复的 DMA 请求会被忽略，等待处理的请求会保持等待状态。与通常情况一样， DMA 通道会在传输最终完成时清除 DMA 请求。如果此时 CPU 不进行干预，则传输的数据将是最新的 （溢出）数据，先前的数据将丢失。

用户应用程序可以根据数据源的性质，使用不同方式来处理溢出错误。DMAC 与数据源 / 阱之间的数据恢复和重新同步是一项高度依赖于应用的任务。对于流数据（例如通过 DCI 外设来自编解码器的数据），应用程序可以忽略丢失的数据。在修复问题来源之后（如果可能），DMA 中断处理程序应尝试将 DMAC 和 DCI 重新同步，使数据可以重新正确缓冲。用户应用程序应尽快作出反应，以防止发生任何进一步的溢出。

在进入外设溢出中断之前，等待处理的 DMA 请求已经将溢出数据值传送到丢失数据的正确目标地址。可以将该数据传送到它的正确地址处，并在丢失数据的位置中插入空数据值。然后，可以相应地调整通道的 DPSRAM 地址。对于故障通道的后续 DMA 请求可以像正常情况一样将数据传输到修正后的 DPSRAM 地址。对于不能丢失数据的应用，外设溢出中断需要中止当前的数据块传输，重新初始化 DMA 通道，并请求重新发送丢失前的数据。

22.9 调试支持

为了提高调试期间用户对于 DMA 操作的可视性，DMA 控制器包含了几个状态寄存器来提供以下信息：哪个 DMA 通道执行了最近一次数据传输（DMACS1 寄存器中的 LSTCH<3:0> 位），它先前访问的是哪个 DPSRAM 地址偏移 （DSADR 寄存器中的 DSADR<15:0> 位），以及来自哪个缓冲区 （DMACS1 寄存器中的 PPSTx 位）。

22.10 数据写冲突

CPU 和 DMA 通道可以同时读或读 / 写任意 DPSRAM 或 DMA 就绪外设数据寄存器。唯一的约束是 CPU 和 DMA 通道不应同时对同一地址执行写操作。正常情况下，绝不应当产生这种情况。但是，如果由于某些原因确实产生了这种情况，则会检测并标记该情况，并启动 DMA 故障陷阱。此外，还允许优先执行 CPU 写操作，虽然这主要是为了产生可预测的行为，在其他情况下几乎没有实际的作用。

在 CPU 读取某个单元时，还允许 DMA 通道在同一总线周期中对同一单元进行写操作，反之亦然。但应当注意，读操作返回的是旧数据，而不是在该总线周期中写入的数据。还需要注意，这种情况被视为正常操作，不会导致执行任何特殊操作。

在 CPU 和 DMA 通道同时对同一 DPSRAM 地址执行写操作时，DMA 控制器状态 0（DMACS0）寄存器中的 XWCOLx 位会置 1。在 CPU 和 DMA 通道同时对同一外设地址执行写操作时，DMA 控制器状态 0 （DMACS0）寄存器中的 PWCOLx 位会置 1。所有冲突状态标志会进行逻辑或操作，产生共用的 DMAC 故障陷阱。当用户应用程序清零中断控制器 （INTCON1）寄存器中的 DMAC 错误状态位 （DMACERR）时， XWCOLx 和 PWCOLx 标志会被自动清零。

当 XWCOLx 或 PWCOLx 保持置 1 时，对存在写冲突错误的通道的后续 DMA 请求会被忽略。

在写冲突条件下， XWCOLx 或 PWCOLx 会由于写冲突而置 1，但不会同时置 1。两个标志同时置 1 用作一种独特的方式来标记很少发生的手动触发事件错误，而不需要增加另外的状态位（见 **第 22.6.10 节 “ 手动传输模式 ”**）。

例22-13给出了DMA控制器陷阱处理的示例，其中使用DMA通道0从DPSRAM向外设（UART）传输数据，使用 DMA 通道 1 从外设 （ADC）向 DPSRAM 传输数据。

例 22-13： DMA 控制器陷阱处理

```
void __attribute__((__interrupt__)) _DMACError(void)
{
    static unsigned int ErrorLocation;

    // Peripheral Write Collision Error Location
    if(DMACS0 & 0x0100)
    {
        ErrorLocation = DMA0STA;
    }

    // DMA RAM Write Collision Error Location
    if(DMACS0 & 0x0002)
    {
        ErrorLocation = DMA1STA;
    }

    DMACS0 = 0;                          //Clear Write Collision Flag
    INTCON1bits.DMACERR = 0;             //Clear Trap Flag
}
```

22.11 节能模式下的操作

22.11.1 休眠模式

在休眠节能模式下，DMA 会被禁止。在进入休眠模式之前，建议（虽然不是必需）允许所有 DMA 通道完成当前正在进行的数据块传输，或者禁止它们。

22.11.2 空闲模式

DMA 是系统中的第二个总线主器件，因此可以在 CPU 进入空闲节能模式时继续传输数据。只要由 DMA 通道服务的外设配置为在空闲模式期间继续工作，就可以在外设和 DPSRAM 之间传输数据。当数据块传输完成时，DMA 通道会发出中断（如果允许）并唤醒 CPU。然后，CPU 会运行中断服务处理程序。

每个外设都包含空闲模式停止控制位。在置 1 时，该控制位会在 CPU 处于空闲模式时禁止外设。如果正在使用 DMAC 从外设或向外设传输数据，则在外设中启用空闲停止功能实际上还会禁止与外设关联的 DMA 通道。

22.12 设计技巧

22.12.1 DMA 与 DCI 接口

当 DCI 每帧具有多个音频通道时 （例如，立体声编解码器的左右通道），所有采样会按顺序交叉，并通过 DMA 通道传输，如图 22-21 所示。

但是，用户应用程序通常希望按照通道来对数据进行处理，这意味着程序必须提供额外的算法来对传输的数据进行分选，或者通过跳过不需要的数据来以索引方式访问数据。两种方式都需要额外的代码，导致更多的执行时间。

DCI 不支持外设间接寻址。但是，仍然可以使用结合两个 DMA 通道的特殊 DCI 配置按音频通道来汇集数据。当 DCI 控制 2 （DCICON2）寄存器中的缓冲区长度控制位 （BLEN<1:0>）设置为 01（而不是 00），并且使用两个 DMA 通道将接收到的数据从 DCI 传输到 DPSRAM 时，接收到的音频数据将按照通道进行分选。这种情况下，当 DCI 产生 DMA 请求时，请求将在每次缓冲两个字 （一个右采样，一个左采样）后送到两个 DMA 通道。当产生 DMA 请求时，一个 DMA 通道会传输来自 DCI 接收缓冲 0 （RXBUF0）寄存器的数据，而另一个 DMA 通道会传输来自 DCI 接收缓冲 1 （RXBUF1）寄存器的数据。实际上，传输的数据将按照音频通道进行分选，如图 22-22 所示。

为使该示例工作， DMA 通道 1 将 DMA1PAD 寄存器初始化为 RXBUF1 地址 （而不是**第 22.5.1 节 “ DMA 通道与外设关联设置 ”** 中所述的 RXBUF0 地址）。

图 22-21： DCI 的典型数据传输

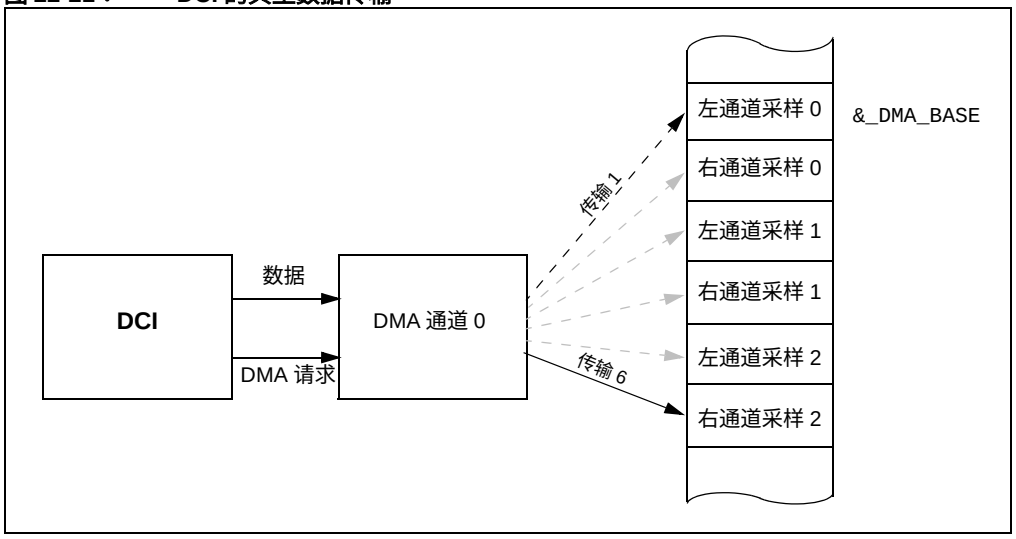
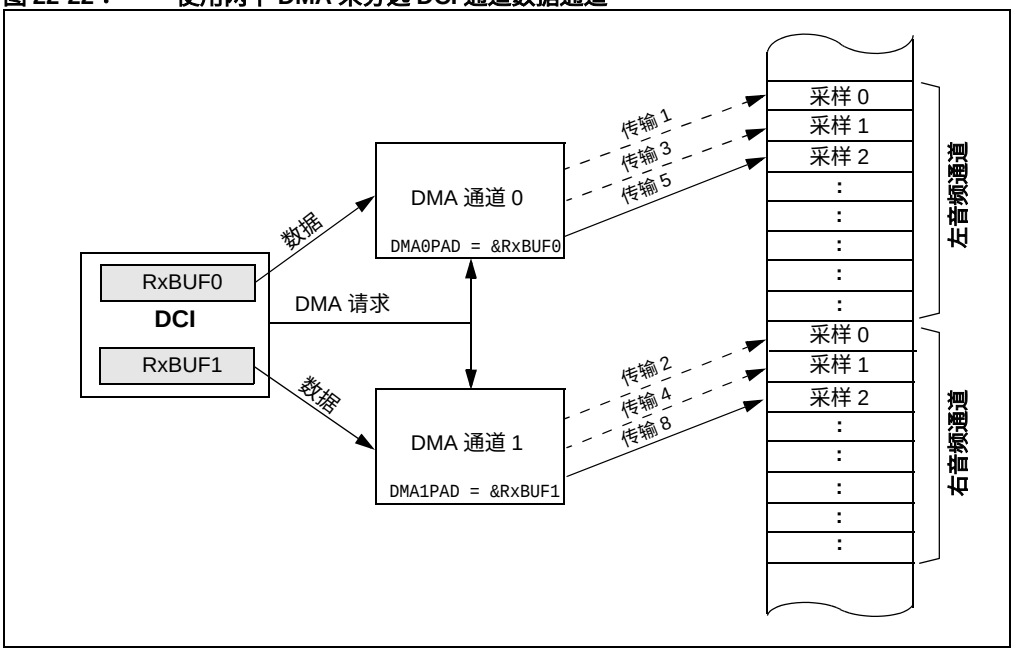


图 22-22： 使用两个 DMA 来分选 DCI 通道数据通道



22.13 寄存器映射

表 22-6： DMA 寄存器映射

寄存器名称	地址	Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	所有复位时的状态
DMA0CON	0380	CHEN	SIZE	DIR	HALF	NULLW	—	—	—	—	—	AMODE<1:0>		—	—	MODE<1:0>		0000
DMA0REQ	0382	FORCE	—	—	—	—	—	—	—	—	IRQSEL<6:0>							0000
DMA0STA	0384	STA<15:0>																0000
DMA0STB	0386	STB<15:0>																0000
DMA0PAD	0388	PAD<15:0>																0000
DMA0CNT	038A	—	—	—	—	—	—	CNT<9:0>										0000
DMA1CON	038C	CHEN	SIZE	DIR	HALF	NULLW	—	—	—	—	—	AMODE<1:0>		—	—	MODE<1:0>		0000
DMA1REQ	038E	FORCE	—	—	—	—	—	—	—	—	IRQSEL<6:0>							0000
DMA1STA	0390	STA<15:0>																0000
DMA1STB	0392	STB<15:0>																0000
DMA1PAD	0394	PAD<15:0>																0000
DMA1CNT	0396	—	—	—	—	—	—	CNT<9:0>										0000
DMA2CON	0398	CHEN	SIZE	DIR	HALF	NULLW	—	—	—	—	—	AMODE<1:0>		—	—	MODE<1:0>		0000
DMA2REQ	039A	FORCE	—	—	—	—	—	—	—	—	IRQSEL<6:0>							0000
DMA2STA	039C	STA<15:0>																0000
DMA2STB	039E	STB<15:0>																0000
DMA2PAD	03A0	PAD<15:0>																0000
DMA2CNT	03A2	—	—	—	—	—	—	CNT<9:0>										0000
DMA3CON	03A4	CHEN	SIZE	DIR	HALF	NULLW	—	—	—	—	—	AMODE<1:0>		—	—	MODE<1:0>		0000
DMA3REQ	03A6	FORCE	—	—	—	—	—	—	—	—	IRQSEL<6:0>							0000
DMA3STA	03A8	STA<15:0>																0000
DMA3STB	03AA	STB<15:0>																0000
DMA3PAD	03AC	PAD<15:0>																0000
DMA3CNT	03AE	—	—	—	—	—	—	CNT<9:0>										0000
DMA4CON	03B0	CHEN	SIZE	DIR	HALF	NULLW	—	—	—	—	—	AMODE<1:0>		—	—	MODE<1:0>		0000
DMA4REQ	03B2	FORCE	—	—	—	—	—	—	—	—	IRQSEL<6:0>							0000
DMA4STA	03B4	STA<15:0>																0000
DMA4STB	03B6	STB<15:0>																0000
DMA4PAD	03B8	PAD<15:0>																0000
DMA4CNT	03BA	—	—	—	—	—	—	CNT<9:0>										0000
DMA5CON	03BC	CHEN	SIZE	DIR	HALF	NULLW	—	—	—	—	—	AMODE<1:0>		—	—	MODE<1:0>		0000
DMA5REQ	03BE	FORCE	—	—	—	—	—	—	—	—	IRQSEL<6:0>							0000

图注： — = 未实现，读为 0。复位值以十六进制显示。

表 22-6： DMA 寄存器映射 （续）

寄存器名称	地址	Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	所有复位时的状态
DMA5STA	03C0	STA<15:0>																0000
DMA5STB	03C2	STB<15:0>																0000
DMA5PAD	03C4	PAD<15:0>																0000
DMA5CNT	03C6	—	—	—	—	—	—	CNT<9:0>										0000
DMA6CON	03C8	CHEN	SIZE	DIR	HALF	NULLW	—	—	—	—	—	AMODE<1:0>		—	—	MODE<1:0>		0000
DMA6REQ	03CA	FORCE	—	—	—	—	—	—	—	—	IRQSEL<6:0>							0000
DMA6STA	03CC	STA<15:0>																0000
DMA6STB	03CE	STB<15:0>																0000
DMA6PAD	03D0	PAD<15:0>																0000
DMA6CNT	03D2	—	—	—	—	—	—	CNT<9:0>										0000
DMA7CON	03D4	CHEN	SIZE	DIR	HALF	NULLW	—	—	—	—	—	AMODE<1:0>		—	—	MODE<1:0>		0000
DMA7REQ	03D6	FORCE	—	—	—	—	—	—	—	—	IRQSEL<6:0>							0000
DMA7STA	03D8	STA<15:0>																0000
DMA7STB	03DA	STB<15:0>																0000
DMA7PAD	03DC	PAD<15:0>																0000
DMA7CNT	03DE	—	—	—	—	—	—	CNT<9:0>										0000
DMACS0	03E0	PWCOL7	PWCOL6	PWCOL5	PWCOL4	PWCOL3	PWCOL2	PWCOL1	PWCOL0	XWCOL7	XWCOL6	XWCOL5	XWCOL4	XWCOL3	XWCOL2	XWCOL1	XWCOL0	0000
DMACS1	03E2	—	—	—	—	LSTCH<3:0>				PPST7	PPST6	PPST5	PPST4	PPST3	PPST2	PPST1	PPST0	0000
DSADR	03E4	DSADR<15:0>																0000
INTCON1	0080	NSTDIS	—	—	—	—	—	—	—	—	—	DMACERR	—	—	—	—	—	0000
IFS0	0084	—	DMA1IF	—	—	—	—	—	—	—	—	—	DMA0IF	—	—	—	—	0000
IFS1	0086	—	—	—	—	—	—	—	DMA2IF	—	—	—	—	—	—	—	—	0000
IFS2	0088	—	DMA4IF	—	—	—	—	—	—	—	—	—	DMA3IF	—	—	—	—	0000
IFS3	008A	—	—	DMA5IF	—	—	—	—	—	—	—	—	—	—	—	—	—	0000
IFS4	008C	—	—	—	—	—	—	—	—	—	—	DMA7IF	DMA6IF	—	—	—	—	0000
IEC0	0094	—	DMA1IE	—	—	—	—	—	—	—	—	—	DMA0IE	—	—	—	—	0000
IEC1	0096	—	—	—	—	—	—	—	DMA2IE	—	—	—	—	—	—	—	—	0000
IEC2	0098	—	DMA4IE	—	—	—	—	—	—	—	—	—	DMA3IE	—	—	—	—	0000
IEC3	009A	—	—	DMA5IE	—	—	—	—	—	—	—	—	—	—	—	—	—	0000
IEC4	009C	—	—	—	—	—	—	—	—	—	—	DMA7IE	DMA6IE	—	—	—	—	0000
IPC1	00A6	—	—	—	—	—	—	—	—	—	—	—	—	—	DMA0IP<2:0>			4444
IPC3	00AA	—	—	—	—	—	DMA1IP<2:0>			—	—	—	—	—	—	—	—	4444
IPC6	00B0	—	—	—	—	—	—	—	—	—	—	—	—	—	DMA2IP<2:0>			4444
IPC9	00B6	—	—	—	—	—	—	—	—	—	—	—	—	—	DMA3IP<2:0>			4444

图注： — = 未实现，读为 0。复位值以十六进制显示。

表 22-6： DMA 寄存器映射 （续）

寄存器名称	地址	Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	所有复位时的状态
IPC11	00BA	—	—	—	—	—	DMA4IP<2:0>			—	—	—	—	—	—	—	—	4444
IPC15	00C2	—	—	—	—	—	—	—	—	—	DMA5IP<2:0>			—	—	—	—	4444
IPC17	00C6	—	—	—	—	—	—	—	—	—	DMA7IP<2:0>			—	DMA6IP<2:0>			4444

图注： — = 未实现，读为 0。复位值以十六进制显示。

22.14 相关应用笔记

本节列出了与直接存储器访问的使用相关的应用笔记。这些应用笔记可能并不是专为 dsPIC33F 产品系列而编写的，但其概念是相近的，通过适当修改并受到一定限制即可使用。当前与 DMA 模块相关的应用笔记有：

标题	应用笔记编号

目前没有相关的应用笔记。

注： 如需获取更多 dsPIC33F 系列器件的应用笔记和代码示例，请访问 Microchip 网站（www.microchip.com）。

22.15 版本历史

版本 A （2006 年 12 月）

这是本文档的初始版本。

版本 B （2008 年 7 月）

该版本包括以下内容更新：

- 图片：
  - 寄存器间接寻址模式下从 ADC 传输数据 （见图 22-11）。
  - 外设间接寻址模式下从 ADC 传输数据 （见图 22-12）。
- 对整篇文档进行了其他少量修正，如语言和格式的更新。