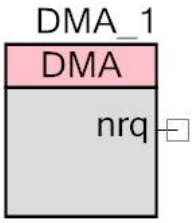
直接储存器访问（DMA）

1.70



特性

- **支持24 个通道**
- 八个优先级别
- 128 个数据操作描述符（TD）
- 8 位、16 位、32 位数据传输
- 可配置来源和目标地址
- 兼容大端和小端存储
- 数据传输完成时可以生成中断
- DMA 向导可帮助应用开发

概述

DMA 组件使数据能传出和传入存储器、组件和寄存器。控制器支持 8 位宽、16 位宽和 32 位宽的数据传输，并且可以进行配置，以在具有不同字节存储顺序（大端/小端存储）的源和目标之间传输数据。TD 可以链接在一起进行复杂操作。

DMA 支持电平触发和上升沿触发。更多详细信息，请参考**硬件要求**参数选择。

何时使用 DMA 组件

当您要释放 CPU 的传输数据任务负担，或者当需要通过可预先设置的可预测方式传输数据时，DMA 组件是很有用的。基本用例如下：

- 存储器到存储器
- 存储器到外设
- 外设到存储器
- 外设到外设

TD 可以单独执行或者链接在一起以执行复杂的传输。

DMA 向导

PSoC Creator 提供了 DMA 向导，以帮助快速、精确地开发使用 DMA 的应用。向导会全程引导您定义 TD 并生成必需的 C 代码，您可以将该代码复制并粘贴到您的应用中。

通过 PSoC Creator 的 **Tools**（工具）菜单中启动向导。相关详细信息，请参见《PSoC Creator 帮助》部分的内容。

PSoC 3 地址

在 PSoC 3 中，除闪存外，DMA 传输中涉及的所有位置都在存储器的前 64 K 空间中。对于除闪存之外的所有位置，地址的高 16 位的值必须为 0。Keil 编译器无法识别前 64 K 空间以外的地址，而是用高 16 位储存其他信息，导致高位字节不为 0。因此，不能直接使用指向该位置指针的高 16 位。更多详细信息，请参见 *Keil 中的通用指针*。对于闪存而言，地址的高 16 位的使用正确值为：

```
HI16(CYDEV_FLS_BASE)
```

这是由编译器完成的具体处理。要创建在 PSoC 3 和 PSoC 5 中都能够正确运行的代码，可以使用以下代码格式。假设“src”为闪存中的变量，“dst”为 SRAM 中的变量：

```
#if (defined(__C51__))
    /* PSoC 3 - Source is Flash */
    dmaChan = DMA_1_DmaInitialize(1, 0, HI16(CYDEV_FLS_BASE), 0);
#else
    /* PSoC 5 */
    dmaChan = DMA_1_DmaInitialize(1, 0, HI16(src), HI16(dst));
#endif
```

PSoC 5 SRAM 访问

在 PSoC 5 中，DMAC 无法从 0x1FFF8000 到 0x1FFFFFFF 访问 SRAM，但是可以访问同一存储器的 0x20008000 到 0x2000FFFF。

CPU 访问：

```
0x1FFF8000 - 0x1FFFFFFF C-BUS 32KB
0x20000000 - 0x20007FFF S-BUS 32KB
```

DMA 访问：

```
0x20000000 - 0x20007FFF S-BUS 32KB
0x20008000 - 0x2000FFFF C-BUS 32KB
```

重新映射由设置 DMA 的 API 自动处理。传递至 API 的参数应为由 DMA API 自动处理的 CPU 原生地址的高 16 位和低 16 位。请注意，当 DMA 引擎增加地址时，仅增加低 16 位。因此，紧跟

0x2000FFFF 的地址为 0x20000000，这使得存储器空间仍可用作存储器的连续 64 KB 字节存储器块。

输入/输出接口

本节介绍了 DMA 的各种输入和输出连接。I/O 列表中的星号（*）表示，在 I/O 说明中列出的情况下，该 I/O 可能不可见。

nrq — 输出

nrq 终止信号可能连接至中断，或连接至组件，以将 DMA 传输完成的信息传送至组件。当 DMA 传输完成时，DMA 在 nrq 上生成一个宽度为 2 总线时钟的脉冲。

drq — 输入*

drq 终止信号连接至一个能够请求 DMA 操作的组件。

drq 输入对电平或边沿敏感。如果 drq 为电平敏感，那么当 drq 激活时，DMA 请求将持续发生。如果 drq 为边沿敏感，DMA 请求的宽度至少应为一个总线时钟周期。

trq — 输入*

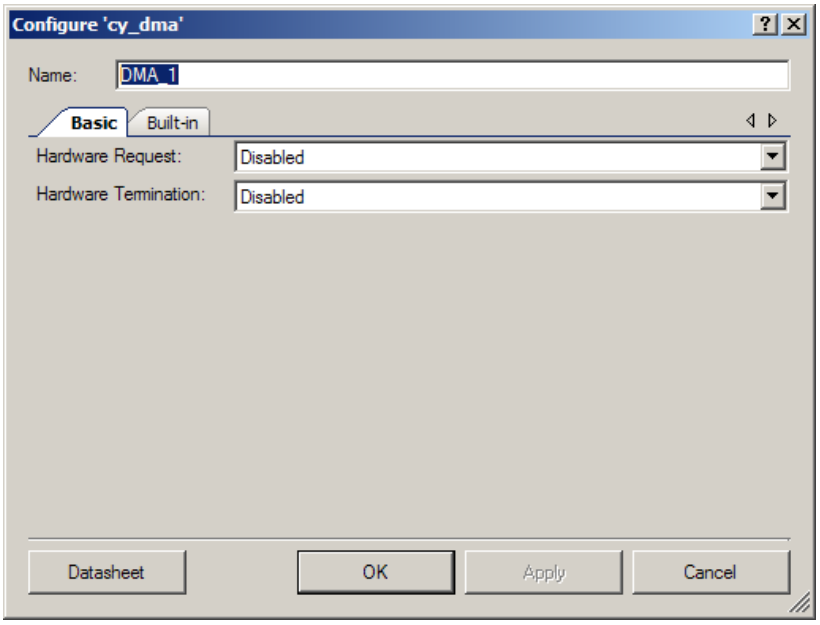
trq 终止信号连接至一个能够终止 DMA 操作的组件。当组件获知无数据可用时，可能需要提供 DMA 的数据。组件使用该信号终止数据操作。

当任务链中的当前 TD 被终止时，它也就完成，如同传输计数完成一样。因此，是否终止操作，取决于链中是否有其他 TD 以及定义的数据操作是何种类型（例如，乒乓、循环、自动重复等）。

该信号仅在通道尝试传输数据时使用。会忽略其他时间内该线路的上升沿。

组件参数

将一个 DMA 组件拖放到您的设计上，并双击以打开 **Configure** 对话框。



DMA 提供以下参数。

硬件要求

该参数配置可能处理 DMA 触发的波形类型。除 **Disabled**（禁用）之外的所有选项均添加了 drq 终止信号，该终止信号使硬件能够发出 DMA 请求。可用的选项有：

- **Disabled**（禁用） — 不显示 drq 终止信号。在这种情况下，只能通过 CPU 触发 DMA。
- **Derived**（派生） — 连接到固定功能块 （I^2C、USB、CAN 等）时，检查 drq 的驱动，派生出它连接时所依据的 DMA 类型。自动分配根据器件数据手册中的信息进行。如果未连接至固定功能块，则使用**上升沿选项**。
- **Rising Edge**（上升沿） — 在源信号的上升沿上触发 DMA。DMA 需要根据事件而触发时，选择该选项。例如，会定期发生的 DMA 将在上升沿模式中进行配置，并且“DRQ”信号可以被连接至速率设置适当的时钟信号。
- **Level**（电平） — 将连接至 DMA 的源选为电平敏感请求。在只要满足特定的条件就需要持续触发 DMA 时，选择该选项。这通常针对视 FIFO 填充电平而触发的 DMA。这是使用通信组件（例如，I2S）的典型配置。

硬件终止

该参数可设为 **Enabled**（启用）或 **Disabled**（禁用）。**Enabled** 添加了 trq 终止信号，允许终止来自硬件的 DMA 请求。当终止信号被禁用时，只能通过 CPU 请求终止 DMA 传输，或者等 DMA 完成数据传输时才能终止。

应用程序接口

通过应用编程接口（API）函数，您可以使用软件对组件进行配置。下表列出并说明了每个函数的接口。以下各节将更详细地介绍每个函数。

默认情况下，PSoC Creator 将实例名称“DMA_1”分配给指定设计中组件的第一个实例。您可以将该实例重命名为符合标识符语法规则的任意唯一值。实例名称会成为每个全局函数名称、变量和常量符号的前缀。出于可读性考虑，下表中使用的实例名称为“DMA”。

每个 DMA 实例的 API

函数	说明
DMA_DmaInitialize()	分配一个DMA通道，并将其初始化，以供调用方使用。
DMA_DmaRelease()	释放并禁用与该组件实例相关联的DMA通道。

uint8 DMA_DmaInitialize(uint8 burstCount, uint8 requestPerBurst, uint16 upperSrcAddress, uint16 upperDestAddress)

说明： 分配一个DMA通道，并将其初始化，以供调用方使用。

参数： uint8 burstCount：指定该通道上的TD需分割为多少个 Bursts（1至127）。Burst的长度应为spoke大小的倍数。

如果该值为零，则每个传输在一次Burst中完成。在这种情况下，每个TD的传输计数参数确定一次Burst中传输的字节数。

如果将通道配置为内部spoke传输，burstCount限定为16，并不能将它设置为0。

uint8 requestPerBurst：如果完成操作处理要求如此的话，数据整体可分为多个Bursts：

值	操作
0	将自动请求并执行首个Burst后的所有后续Bursts。
1	必须单独请求首个Burst后的所有后续Bursts。

uint16 upperSrcAddress：源地址的高16位。

uint16 upperDestAddress：目标地址的高16位。

返回值： uint8：可供DMA活动的调用方使用的通道。如果无通道剩余，则返回CY_DMA_INVALID_CHANNEL (0xFF)。

其他影响: 无

void DMA_DmaRelease(void)

说明： 释放与该组件实例相关联的通道。除非再次调用DMA_DmaInitialize()，否则无法再次使用该通道。

参数： 无

返回值： 无

其他影响： 无

DMA 库 API（由所有 DMA 实例共享）

DMA 控制器函数

函数	说明
CyDmacConfigure()	使用默认值设置DMAC配置寄存器。
CyDmacError()	从DMAC获取错误位。
CyDmacClearError()	清除DMAC错误寄存器中的错误位。
CyDmacErrorAddress()	获取前一次DMAC错误发生的地址。

void CyDmacConfigure(void)

说明： 为所有要分配的TD创建链接列表。只有当DMA组件被置于设计原理图上时，才通过启动代码调用该函数。用户通常无需调用它。如果所有DMA通道都处于非活动状态，用户可以调用该函数。

参数： 无

返回值： 无

其他影响： 无

uint8 CyDmacError(void)

说明： 返回前一次失败的DMA操作的错误。

参数： 无

返回值： 前一次失败的DMA操作的错误。

位	定义	说明
位3	CY_DMA_PERIPH_ERR	当外设对总线数据操作做出错误响应时，设置为1。通过写入1进行清除。
位2	CY_DMA_UNPOP_ACC	当尝试访问无效地址时，设置为1。通过写入1进行清除。
位1	CY_DMA_BUS_TIMEOUT	当总线发生超时时，设置为1。通过写入1进行清除。超时值由PHUBCFG寄存器中的BUS_TIMEOUT字段决定。

其他影响： 无

void CyDmacClearError(uint8 error)

说明： 清除DMAC错误寄存器中的错误位。

参数： uint8 error：DMA_ERROR类型中需清除的错误位的位掩码。

位	定义	说明
位3	CY_DMA_PERIPH_ERR	当外设对总线数据操作做出错误响应时，设置为1。通过写入1进行清除。
位2	CY_DMA_UNPOP_ACC	当尝试访问无效地址时，设置为1。通过写入1进行清除。
位1	CY_DMA_BUS_TIMEOUT	当总线发生超时时，设置为1。通过写入1进行清除。超时值由PHUBCFG寄存器中的BUS_TIMEOUT字段决定。

返回值： 无

其他影响： 无

uint32 CyDmacErrorAddress(void)

说明： 当 CY_DMA_BUS_TIMEOUT 、 CY_DMA_UNPOP_ACC 和 CY_DMA_PERIPH_ERR 发 生时，错误地址会被写入到错误地址寄存器中，并可由该函数读取。如果存在多个错误，则仅保存首个错误的地址。

参数： 无

返回值： 引起错误的地址。

其他影响: 无

通道特定函数

函数	说明
CyDmaChAlloc()	分配一个DMA通道以供调用方使用。
CyDmaChFree()	释放由CyDmaChAlloc()分配的通道。
CyDmaChEnable()	为执行启用DMA通道。
CyDmaChDisable()	禁用DMA通道。
CyDmaClearPendingDrq()	清除一个待处理的DMA数据请求。
CyDmaChPriority()	设置DMA通道的优先级。
CyDmaChSetExtendedAddress()	设置源和目标地址的高16位。
CyDmaChSetInitialTd()	设置通道的初始TD。
CyDmaChSetRequest()	终止一个TD链、一个TD或启动DMA的请求。
CyDmaChGetRequest()	检查是否满足CyDmaChSetRequest()请求。
CyDmaChStatus()	确定当前TD的状态。
CyDmaChSetConfiguration()	设置通道的配置信息。
CyDmaChRoundRobin()	使能/禁用循环调度算法

uint8 CyDmaChAlloc(void)

说明： 从DMAC中分配一个通道以供所有需要通道句柄的函数使用。

参数： 无

返回值： 所分配通道的编号。零是有效的通道编号。无通道可用时，则返回 CY_DMA_INVALID_CHANNEL。

其他影响: 无

cystatus CyDmaChFree(uint8 chHandle)

说明： 释放由CyDmaChAlloc()分配的通道句柄。

参数： uint8 chHandle：CyDmaChAlloc()或DMA_DmaInitalize()先前返回的句柄。

返回值： 如果成功，则返回CYRET_SUCCESS。

如果**chHandle**无效，则返回CYRET_BAD_PARAM。

其他影响： 无

cystatus CyDmaChEnable(uint8 chHandle, uint8 preserveTds)

说明： 使能DMA通道。软件或硬件请求仍必须在通道被执行前发出。

参数： uint8 chHandle：CyDmaChAlloc()或DMA_DmaInitalize()先前返回的句柄。

uint8 preserveTds：TD完成后，保留原始TD状态。该参数适用于通道中的所有TD。

值	操作
0	TD完成后，DMAC不会将TD配置值恢复到原始状态，而是保持当前状态。
1	TD完成后，DMAC将恢复TD的原始配置值。

设置了**preserveTds**时，与通道编号相等的TD槽变为RESERVED状态，即工作寄存器所在之处。例如，如果您正使用CH06并设置了**preserveTds**，TD槽6将不可使用。DMA引擎将其收回，留作私用。

注意：如果通道的**preserveTds**设置为0，请不要链回到已完成的TD。如果TD已完成被设置为0的通道**preserveTds**，则传输计数将为0。如果启动了一个传输计数为0的TD，则该TD将传输不确定量的数据。

如果**preserveTds**被设为0，那么在使用硬件请求（DRQ）选项时请多加注意，因为您可能正在请求错误数据。

返回值： 如果成功，则返回CYRET_SUCCESS。

如果**chHandle**无效，则返回CYRET_BAD_PARAM。

其他影响: 使能通道前会寄存待处理的DMA请求。如果不想发生这种情况，可以在调用该API前，先调用CyDmaClearPendingDrq()，以清除DMA请求。

cystatus CyDmaChDisable(uint8 chHandle)

说明： 禁用DMA通道一旦调用该函数，则可调用CyDmaChStatus()，以确定何时禁用通道以及正在执行哪些TD。

如果当前正在执行DMA通道，它将自然完成当前的Burst。

参数： uint8 chHandle：CyDmaChAlloc()或DMA_DmaInitalize()先前返回的句柄。

返回值： 如果成功，则返回CYRET_SUCCESS。

如果**chHandle**无效，则返回CYRET_BAD_PARAM。

其他影响： 无

cystatus CyDmaClearPendingDrq(uint8 chHandle)

说明： 清除待处理的DMA数据请求。

参数： chHandle：dma通道的句柄。

返回值： 如果成功，则返回CYRET_SUCCESS。

如果**chHandle**无效，则返回CYRET_BAD_PARAM。

其他影响： 无

cystatus CyDmaChPriority(uint8 chHandle, uint8 priority)

说明： 设置DMA通道的优先级。当用户想在运行中更改优先级时，可使用该函数。如果DMA通道的优先级仍未改变，用户可在“.cydwr”文件中配置优先级。

参数： uint8 chHandle：CyDmaChAlloc()或DMA_DmaInitalize()先前返回的句柄。

uint8 priority：要设置的通道优先级，0至7（其中0表示优先级最高，7表示优先级最低）。

返回值： 如果成功，则返回CYRET_SUCCESS。

如果**chHandle**无效，则返回CYRET_BAD_PARAM。

其他影响： 无

cystatus CyDmaChSetExtendedAddress(uint8 chHandle, uint16 source, uint16 destination)

说明： 为DMA通道设置源和目标地址的高16位（适用于链中的所有TD）。

参数： uint8 chHandle：CyDmaChAlloc()或DMA_DmaInitalize()先前返回的句柄。

uint16 source：DMA传输源的高16位地址。

uint16 destination：DMA传输目标的高16位地址。

返回值： 如果成功，则返回CYRET_SUCCESS。

如果**chHandle**无效，则返回CYRET_BAD_PARAM。

其他影响： 无

cystatus CyDmaChSetInitialTd(uint8 chHandle, uint8 startTd)

说明： 调用CyDmaChEnable()函数时，为通道设置要执行的初始TD。

参数： uint8 chHandle：CyDmaChAlloc()或DMA_DmaInitialize()先前返回的句柄。

uint8 startTd：设置为与通道相关联的首个TD的TD索引。零是有效的TD索引。

返回值： 如果成功，则返回CYRET_SUCCESS。

如果**chHandle**无效，则返回CYRET_BAD_PARAM。

其他影响： 无

cystatus CyDmaChSetRequest(uint8 chHandle, uint8 request)

说明： 允许调用方终止一个TD链，终止单个TD，或创建一个直接请求以启动DMA通道。

参数： uint8 chHandle：CyDmaChAlloc()或DMA_DmaInitalize()先前返回的句柄。

uint8 request：以下常量之一。各常量均为3位的值。

请求值	说明
CY_DMA_CPU_REQ	创建一个直接请求以启动DMA通道
CY_DMA_CPU_TERM_TD	终止一个TD，但不会禁用通道
CY_DMA_CPU_TERM_CHAIN	终止一个TD链，并禁用通道

返回值： 如果成功，则返回CYRET_SUCCESS。

如果**chHandle**无效，则返回CYRET_BAD_PARAM。

其他影响： 无

cystatus CyDmaChGetRequest(uint8 chHandle)

说明： 该函数使CyDmaChSetRequest()的调用方能够确定请求是否已完成。

参数： uint8 chHandle：CyDmaChAlloc()或DMA_DmaInitalize()先前返回的句柄。

返回值： 返回一个3位字段，该字段对应于请求中描述先前已发布请求状态的3位。如果值为零，则请求已完成。

如果句柄无效，则返回CY_DMA_INVALID_CHANNEL。

其他影响: 无

cystatus CyDmaChStatus(uint8 chHandle, uint8 * currentTd, uint8 * state)

说明： 确定DMA通道的状态。

参数： uint8 chHandle：CyDmaChAlloc()或DMA_DmaInitalize()先前返回的句柄。

uint8 * currentTd：存储当前TD索引的地址。如果不需要该值，则可以为NULL（空）。

uint8 * state：存储通道状态的地址。如果不需要该值，则可以为NULL（空）。

位 1	CY_DMA_STATUS_TD_ACTIVE	0：通道当前不处于DMAC的处理中
		1：通道当前正处于DMAC的处理中
位 0	CY_DMA_STATUS_CHAIN_ACTIVE	0：TD链处于非活动状态；没有DMA请求触发新链、或者先前的链已完成。
		1：DMA请求触发了TD链

返回值： 如果成功，则返回CYRET_SUCCESS。

如果**chHandle**无效，则返回CYRET_BAD_PARAM。

其他影响： 无

cystatus CyDmaChSetConfiguration(uint8 chHandle, uint8 burstCount, uint8 requestPerBurst, uint8 tdDone0, uint8 tdDone1, uint8 tdStop)

说明： 设置通道的配置信息。

参数： uint8 chHandle：CyDmaChAlloc()或DMA_DmaInitialize()先前返回的句柄。

uint8 burstCount：指定数据传输需分割为多少个Bursts（1至127）。如果该值为零，则整个传输在一次Burst中完成。

如果将通道配置为内部spoke传输，burstCount限定为16，并不能将它设置为0。

uint8 requestPerBurst：如果完成操作处理要求如此的话，数据整体可分为多个Burst：

值	操作
0	将自动请求并执行首个Burst后的所有后续Bursts。
1	必须单独请求首个Burst后的所有后续Bursts。

uint8 tdDone0：选择一个TERMOUT0中断线以发送完成信号。该线与nrq终止信号相连，将确定TERMOUT0_SEL定义，并且应按照*cyfitter.h*所提供的那样进行使用

uint8 tdDone1：选择一个TERMOUT1中断线以发送完成信号。该线与nrq终止信号相连，将确定TERMOUT1_SEL定义，并且应按照*cyfitter.h*所提供的那样进行使用

uint8 tdStop：选择一个TERMIN中断线以向DMAC发送终止TD的信号。与trq终止信号相连的信号将确定使用哪一个TERMIN（终止请求）。

返回值： 如果成功，则返回CYRET_SUCCESS。

如果**chHandle**无效，则返回CYRET_BAD_PARAM。

其他影响： 无

cystatus CyDmaChRoundRobin(uint8 chHandle, uint8 enableRR)

说明： 启用/禁用循环调度算法在一个优先级内，强制循环公平算法。默认配置中，循环调度算法被禁用。

参数： uint8 chHandle：CyDmaChAlloc()或DMA_DmaInitialize()先前返回的句柄。

uint8 enableRR

值	操作
0	禁用循环公平算法
1	使能循环公平算法

返回值： 如果成功，则返回CYRET_SUCCESS。

如果**chHandle**无效，则返回CYRET_BAD_PARAM。

其他影响： 无

数据操作描述符函数

函数	说明
CyDmaTdAllocate()	从空闲列表中选取一个TD以分配进行使用。
CyDmaTdFree()	将一个TD返回至空闲列表中。
CyDmaTdFreeCount()	获得可用的空闲TD的数量。
CyDmaTdSetConfiguration()	设置TD的配置。
CyDmaTdGetConfiguration()	获得TD的配置。
CyDmaTdSetAddress()	设置源和目标地址的低16位。
CyDmaTdGetAddress()	获得源和目标地址的低16位。

uint8 CyDmaTdAllocate(void)

说明： 分配一个TD以与分配的DMA通道配合使用。

参数： 无

返回值： 供调用方使用且从零开始的TD索引。因为总共有128个TD，减去预留TD（0到23），因此返回值的范围是24到127，而不是24到128。

如果无可用的空闲TD，则返回CY_DMA_INVALID_TD。

其他影响: 无

void CyDmaTdFree(uint8 tdHandle)

说明： 将一个TD返回至空闲列表中。

参数： uint8 tdHandle：由CyDmaTdAllocate() API返回的TD句柄

返回值： 无

其他影响： 无

uint8 CyDmaTdFreeCount(void)

说明： 返回可分配的空闲TD的数量。

参数： 无

返回值： 空闲TD的数量。

其他影响： 无

cystatus CyDmaTdSetConfiguration(uint8 tdHandle, uint16 transferCount, uint8 nextTd, uint8 configuration)

说明： 对TD进行配置。

参数： uint8 tdHandle：CyDmaTdAlloc()先前所返回的句柄。

uint16 transferCount：该TD数据传输的大小（单位为字节）。如果大小为零，则传输将无限期地进行下去。该参数被限制为4095字节；如果传递了更大的值，则TD根本不会得到初始化。

uint8 nextTd：TD链中下一个传输描述符的索引从零开始。零是指向下一个TD的有效指针；该TD完成后，CY_DMA_DISABLE_TD会禁用通道。

uint8 configuration：存储配置位的位字段。

配置选项	说明
CY_DMA_TD_SWAP_EN	执行字节序交换
CY_DMA_TD_SWAP_SIZE4	交换大小 = 4字节
CY_DMA_TD_AUTO_EXEC_NEXT	当前TD完成后，链中的下一个TD将会自动触发。
CY_DMA_TD_TERMIN_EN	如果trq输入线路发生上升沿，则终止该TD。在一次Burst过程中必须发生上升沿。DMAC仅在此时对其进行侦听。
DMA__TD_TERMOUT_EN	该TD完成时，TERMOUT信号将生成一个脉冲。请注意，该选项名称取决于特定实例的名称：实例名称后加两个下划线。本例中，实例名称为DMA。
CY_DMA_TD_INC_DST_ADR	根据Burst中每个数据操作的大小而定的增量DST_ADR。如果下一个传输是在另一个地址（与先前的传输不同）上进行的，那么应该使用该配置选项。另外它还使用于大于spoke宽度的Burst。
CY_DMA_TD_INC_SRC_ADR	根据Burst中每个数据操作的大小而定的增量SRC_ADR。从其他地址（与先前的传输不同）进行下一个传输时，应使用该配置选项。另外它还使用于大于spoke宽度的Burst。

返回值： 如果成功，则返回CYRET_SUCCESS。

如果**tdHandle**或**transferCount**无效，则返回CYRET_BAD_PARAM。

其他影响： 无

cystatus CyDmaTdGetConfiguration(uint8 tdHandle, uint16 * transferCount, uint8 * nextTd, uint8 * configuration)

说明： 检索TD的配置。如果将空指针作为参数传递，则将跳过该参数。用户可能仅请求他们感兴趣的值。

参数： uint8 tdHandle：CyDmaTdAlloc()先前所返回的句柄。

uint16 * transferCount：存储该TD数据传输大小（单位为字节）的地址。如果大小为零，则可能表示TD已完成其传输，也可能是TD正在进行无限期传输。

uint8 * nextTd：存储TD链中下一个TD索引的地址。

uint8 * configuration：存储配置位的位字段的地址。请查阅CyDmaTdSetConfiguration()函数说明。

返回值： 如果成功，则返回CYRET_SUCCESS。

如果**tdHandle**无效，则返回 CYRET_BAD_PARAM。

其他影响： 如果TD传输计数为N且已执行，则该传输计数会变为0。如果重新执行该TD，则传输计数零将解释为无限传输。在请求传输计数为零的TD时要注意。

cystatus CyDmaTdSetAddress(uint8 tdHandle, uint16 source, uint16 destination)

说明： 仅为该TD设置源和目标地址的低16位。

参数： uint8 tdHandle：CyDmaTdAlloc()先前所返回的句柄。

uint16 source：数据传输源的低16位地址。

uint16 destination：数据传输目标的低16位地址。

返回值： 如果成功，则返回CYRET_SUCCESS。

如果**tdHandle**无效，则返回 CYRET_BAD_PARAM。

其他影响： 无

cystatus CyDmaTdGetAddress(uint8 tdHandle, uint16 * source, uint16 * destination)

说明： 仅为该TD检索源和/或目标地址的低16位。如果NULL（空）被当做指针参数，则将跳过该值。用户可能仅需要他们感兴趣的值。

参数： uint8 tdHandle：CyDmaTdAlloc()先前所返回的句柄。

uint16 * source：存储数据传输源的低16位地址的地址。

uint16 * destination：存储数据传输目标的低16位地址的地址。

返回值： 如果成功，则返回CYRET_SUCCESS。

如果**tdHandle**无效，则返回 CYRET_BAD_PARAM。

其他影响： 无

MISRA 合□性

本节介绍了MISRA-C:2004合规性和本组件的偏差情况。定义了下面两种类型的偏差：

- 项目偏差 — 适用于所有 PSoC Creator 组件的偏差
- 特定偏差 — 仅适用于该组件的偏差

本节提供了有关组件特定偏差的信息。*系统参考指南*的“MISRA 合规性”章节中介绍了项目偏差以及有关 MISRA 合规性验证环境的信息。

DMA 组件具有以下特定偏差：

MISRA-C:2004 规则	规则类别（必须 (R) / 建议 (A)）	规则说明	偏差说明
10.1	R	在下面情况下，整数类型表达式的数值不应隐式转换为不同的底层类型。 a)没有转换到相同符号的更宽的整数类型，或 b)表达式是复合的，或 c)表达式不是一个常量，而是函数参数，或 d)表达式不是一个常量，而是返回表达式。	DMA组件提供了缺少无符号修改的整数类型定义（例如，0x01，而不是0x01u）。
13.2	A	对一个非零值进行显性测试，除非操作数是有效的布尔（Boolean）值。	直接将整数值作为有条件运算符的第一个操作数使用。
17.4	R	阵列索引是唯一允许的指针运算形式。	使用数组下标访问DMA寄存器中的结构域。

示例固件源代□

在“Find Example Project”对话框中，PSoC Creator 提供了大量的示例项目，包括原理图和示例代码。要获取组件特定的示例，请打开器件目录中的对话框或原理图中的组件实例。要查看通用示例，请打开“Start Page”或 **File** 菜单中的对话框。根据要求，可以通过使用对话框中的 **Filter Options** 选项来限定可选的项目列表。

更多有关信息，请参见《PSoC Creator 帮助》中的“Find Example Project”（查找示例项目）主题。

DMA 地址可移植性

传递到 DMA_DmaInitialize()函数或 CyDmaChSetExtendedAddress()函数中的高位地址意义对于 PSoC 3 和 PSoC 5 器件而言并不相同。PSoC 3 和 PSoC 5 器件的 DMA 地址不能互相移植。高位地址需设置为物理源和目标地址的高 16 位。

以下各节提供了示例代码。

PSoC 3 器件

对于 PSoC 3 器件，由于高位地址值不对应于指针的高位字节，所以需要手动设置高位地址值。Keil 编译器（与 PSoC 3 器件配合使用）用指针的高位字节代表存储器类型，而不是物理地址。

要读取闪存，PSoC 3 器件源的高位地址定义必须为：`CYDEV_FLS_BASE`。

从 RAM 到 RAM

```
uint8 DMA_1_Chan;
uint8 DMA_1_TD[1];

/* DMA Configuration for DMA_1 */
#define DMA_1_BYTES_PER_BURST 16
#define DMA_1_REQUEST_PER_BURST 1
#define DMA_1_SRC_BASE (CYDEV_SRAM_BASE)
#define DMA_1_DST_BASE (CYDEV_SRAM_BASE)
DMA_1_Chan = DMA_1_DmaInitialize(DMA_1_BYTES_PER_BURST, DMA_1_REQUEST_PER_BURST,
    HI16(DMA_1_SRC_BASE), HI16(DMA_1_DST_BASE));
DMA_1_TD[0] = CyDmaTdAllocate();
CyDmaTdSetConfiguration(DMA_1_TD[0], 128, CY_DMA_DISABLE_TD, DMA_1__TD_TERMOUT_EN
| CY_DMA_TD_INC_SRC_ADR | CY_DMA_TD_INC_DST_ADR);
CyDmaTdSetAddress(DMA_1_TD[0], LO16((uint32)memory1), LO16((uint32)memory2));
CyDmaChSetInitialTd(DMA_1_Chan, DMA_1_TD[0]);
CyDmaChEnable(DMA_1_Chan, 1);
```

从闪存到 RAM

```
uint8 DMA_1_Chan;
```

```
uint8 DMA_1_TD[1];

/* DMA Configuration for DMA_1 */
#define DMA_1_BYTES_PER_BURST 1
#define DMA_1_REQUEST_PER_BURST 1
#define DMA_1_SRC_BASE (CYDEV_FLS_BASE)
#define DMA_1_DST_BASE (CYDEV_SRAM_BASE)

DMA_1_Chan = DMA_1_DmaInitialize(DMA_1_BYTES_PER_BURST, DMA_1_REQUEST_PER_BURST,
    HI16(DMA_1_SRC_BASE), HI16(DMA_1_DST_BASE));

DMA_1_TD[0] = CyDmaTdAllocate();
CyDmaTdSetConfiguration(DMA_1_TD[0], 128, CY_DMA_DISABLE_TD, DMA_1__TD_TERMOUT_EN
| CY_DMA_TD_INC_SRC_ADR | CY_DMA_TD_INC_DST_ADR);
CyDmaTdSetAddress(DMA_1_TD[0], LO16((uint32)buf1), LO16((uint32)buf2));
CyDmaChSetInitialTd(DMA_1_Chan, DMA_1_TD[0]);
CyDmaChEnable(DMA_1_Chan, 1);
```

从闪存到 DAC

```
uint8 DMA_1_Chan;
uint8 DMA_1_TD[1];

/* DMA Configuration for DMA_1 */
#define DMA_1_BYTES_PER_BURST 1
#define DMA_1_REQUEST_PER_BURST 1
#define DMA_1_SRC_BASE (CYDEV_FLS_BASE)
#define DMA_1_DST_BASE (CYDEV_PERIPH_BASE)
DMA_1_Chan = DMA_1_DmaInitialize(DMA_1_BYTES_PER_BURST, DMA_1_REQUEST_PER_BURST,
    HI16(DMA_1_SRC_BASE), HI16(DMA_1_DST_BASE));
DMA_1_TD[0] = CyDmaTdAllocate();
CyDmaTdSetConfiguration(DMA_1_TD[0], 64, CY_DMA_DISABLE_TD,
CY_DMA_TD_INC_SRC_ADR);
CyDmaTdSetAddress(DMA_1_TD[0], LO16((uint32)fFashMem),
LO16((uint32)VDAC8_1_Data_PTR));
CyDmaChSetInitialTd(DMA_1_Chan, DMA_1_TD[0]);
CyDmaChEnable(DMA_1_Chan, 1);
```

PSoC 5 器件

对于 PSoC 5 器件而言，高位地址可以设置为源和目标地址的高 16 位。

从 RAM 到 RAM

```
uint8 DMA_1_Chan;
uint8 DMA_1_TD[1];

/* DMA Configuration for DMA_1 */
#define DMA_1_BYTES_PER_BURST 1
#define DMA_1_REQUEST_PER_BURST 1
```

```
#define DMA_1_SRC_BASE (buffer1)
#define DMA_1_DST_BASE (buffer2)
DMA_1_Chan = DMA_1_DmaInitialize(DMA_1_BYTES_PER_BURST, DMA_1_REQUEST_PER_BURST,
    HI16(DMA_1_SRC_BASE), HI16(DMA_1_DST_BASE));
DMA_1_TD[0] = CyDmaTdAllocate();
CyDmaTdSetConfiguration(DMA_1_TD[0], 128, CY_DMA_DISABLE_TD, DMA_1__TD_TERMOUT_EN
| CY_DMA_TD_INC_SRC_ADR |CY_DMA_TD_INC_DST_ADR);
CyDmaTdSetAddress(DMA_1_TD[0], LO16((uint32)buffer1), LO16((uint32)buffer2));
CyDmaChSetInitialTd(DMA_1_Chan, DMA_1_TD[0]);
CyDmaChEnable(DMA_1_Chan, 1);
```

从闪存到 DAC：

```
uint8 DMA_1_Chan;
uint8 DMA_1_TD[1];

/* DMA Configuration for DMA_1 */
#define DMA_1_BYTES_PER_BURST 1
#define DMA_1_REQUEST_PER_BURST 1
#define DMA_1_SRC_BASE (FlashMem)
#define DMA_1_DST_BASE (CYDEV_PERIPH_BASE)
DMA_1_Chan = DMA_1_DmaInitialize(DMA_1_BYTES_PER_BURST, DMA_1_REQUEST_PER_BURST,
    HI16(DMA_1_SRC_BASE), HI16(DMA_1_DST_BASE));
DMA_1_TD[0] = CyDmaTdAllocate();
CyDmaTdSetConfiguration(DMA_1_TD[0], 64, CY_DMA_DISABLE_TD, DMA_1__TD_TERMOUT_EN |
CY_DMA_TD_INC_SRC_ADR);
CyDmaTdSetAddress(DMA_1_TD[0], LO16((uint32)FlashMem),
LO16((uint32)VDAC8_1_Data_PTR));
CyDmaChSetInitialTd(DMA_1_Chan, DMA_1_TD[0]);
CyDmaChEnable(DMA_1_Chan, 1);
```

边界条件

单一 DMA 通道无法超出 64 KB 边界。因此，如果一个 DMA 偶然超出了 64 KB 边界，操作就会失败。这点仅对 PSoC 5 器件重要，因为 PSoC 3 器件的存储器空间不会超过 64 KB 闪存或 8 KB 的 RAM。

使用数据结构时，必须确保其不超过 64 KB 边界。可以使用以下关键词进行确认：

```
__attribute__((section()))
__attribute__((aligned()))
```

对这两个关键词的描述，可参考“指定变量属性”一节中的 GCC 帮助。如果您需要使变量出现在特定的部分和位置上，则可以使用节关键词。使用对齐的关键词以使编译器将变量分配到特定的边界。

地址对齐

对于使用大小超过了一个字节的Burst的DMA传输而言，源地址和目标地址的对齐方式会影响所传输的数据以及传输速度。2字节Burst的对齐地址是一个偶地址。4字节Burst的对齐地址是一个4的倍数的地址。如果地址未对齐，并被设置为不递增，那么将不会传送正确数据。如果地址未对齐，但被设置为递增，则仍然传输正确的数据，并且需要使用额外周期对该传输中未对齐的部分进行读取和写入操作。

如果可能的话，请将源地址和目标地址位配置为对齐地址。对于使用了PSoC 5/5LP的SRAM或闪存地址，16位变量（比如，uint16）为16位对齐的，32位变量为32位对齐的。不管它的大小如何，PSoC 3变量一般不对齐。为了确保变量的特定对齐，必须将该变量置于对齐的绝对位置。编译器将变量中的剩余部分存放在可用的剩余空间内。下面的示例介绍的是将一个16位阵列放置在一个对齐的位置：

```
uint16 buf[10] _at_ 0x20;
```

资源

DMA组件使用器件的一个DMA通道。

API 储存器的使用情况

根据编译器、器件、所使用的API数量以及组件的配置不同，组件对存储资源的占用也不一样。下表提供了在某种器件配置中所有API占用存储器的大小。
数据是在将编译器设置为Release模式并将优化等级设置为Size的情况下测得的。不测量DMA库API。有关特定的设计，可分析编译器生成的映射文件以确定存储器使用情况。

配置	PSoC 3（Keil_PK51）		PSoC 5（GCC）		PSoC 5LP（GCC）	
	闪存 字节	SRAM 字节	闪存 字节	SRAM 字节	闪存 字节	SRAM 字节
默认值	84	1	68	1	68	1

口件更改

本节列出了该组件各版本中的主要更改内容。

版本	更改内容	更改原因/影响
1.70.c	Minor datasheet edit.	
1.70.b	对数据手册进行少量更新。	搞清楚新API的说明。
1.70.a	修正定义过时的参考内容。	数据手册的先前版本包括了缺少“CY_DMA”前缀的局部定义参考。这些定义都过时了。应该使用带有“CY_DMA”前缀的定义。
	说明CY_DMA_DISABLE_TD的使用	CY_DMA_DISABLE_TD应作为“下一个TD”使用，以便在某个链的末端禁用通道。
1.70	添加了MISRA合规性章节。	该组件具有三项特定偏差。
	添加了内部spoke DMA的BURST_CNT限制说明。	内部spoke DMA的BURST_CNT的值不能超过16。
	改进了SetRequest API的请求参数说明。	说明
1.60	添加了对CyDmaChRoundRobin()函数的说明。	执行该函数，但在数据手册中不进行说明。
	添加了**地址对齐**部分。	说明了非对齐传输陷阱以及性能影响。
	更新了对CyDmacConfigure()函数的说明。	更新了说明中的描述内容：当DMA组件被置于设计原理图中时，将通过启动代码调用该函数。
	对数据手册进行了少量编辑和更新	
1.50.b	对数据手册进行了少量的编辑和更新	
1.50.a	添加了CyDmaClearPendingDrq API	
	对数据手册进行了少量编辑和更新	
1.50	添加了选取DRQ类型的功能。	原有功能（新版中的“派生”）有时不能正确地确定DRQ类型，因此我们添加了手动确定的功能。 这包括将“Configure”对话框更改为不再使用默认编辑器。
	删除了num_tds参数。	TD的使用数量取决于写入的代码。允许等同于注释前确定该数量，并且不能确保反映出实际的使用数量。该数字不再显示在DWR编辑器中。
	添加了一个符号摘要。	提供组件的概述。
	将组件的边角形状改为方形。	更新以符合企业风格指南。
	更新了CyDmaTdSetConfiguration()函数。	将NRQ信号路由/未路由到原理图内时，API自动处理termout信号。

版本	更改内容	更改原因/影响
	在头文件中添加了#define DMA__TD_TERMOUT_EN。	该值可以与其他TD标志组合，以使能组件使用的termout。
	从uint8 DMA_DmaInitialize()中删除了一项激活	由于之前对代码进行了更改，该激活已过期。