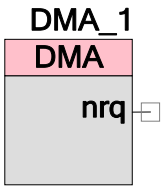
直接存储器访问 (Direct Memory Access, DMA)

1.50



特性

- 24 通道
- 八个优先级别
- 128 个数据操作描述符 (Transaction Descriptor, TD)
- 8 位、16 位、32 位数据传输
- 可配置源和目标地址
- 支持字节序兼容
- 数据传输完成后可生成中断
- 用于协助应用开发的 DMA 向导

概述

DMA 组件使数据能传输出入存储器、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，以在具有不同字节序的源和目标之间传输数据。能将 TD 串接在一起进行复杂操作。

可根据电平或上升沿信号触发 DMA。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。下面是几种基本的使用案例：

- 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 Wizard（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。

PSoC 3 Addresses（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 中的泛型指针*。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 Access（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。

Input/Output Connections（输入/输出连接）

本节介绍 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 可能隐藏在该符号中。

nrq – Output（输出）

nrq 终端可能连接至中断，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 – Input（输入）*

drq 终端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，当 drq 激活后，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 – Input（输入）*

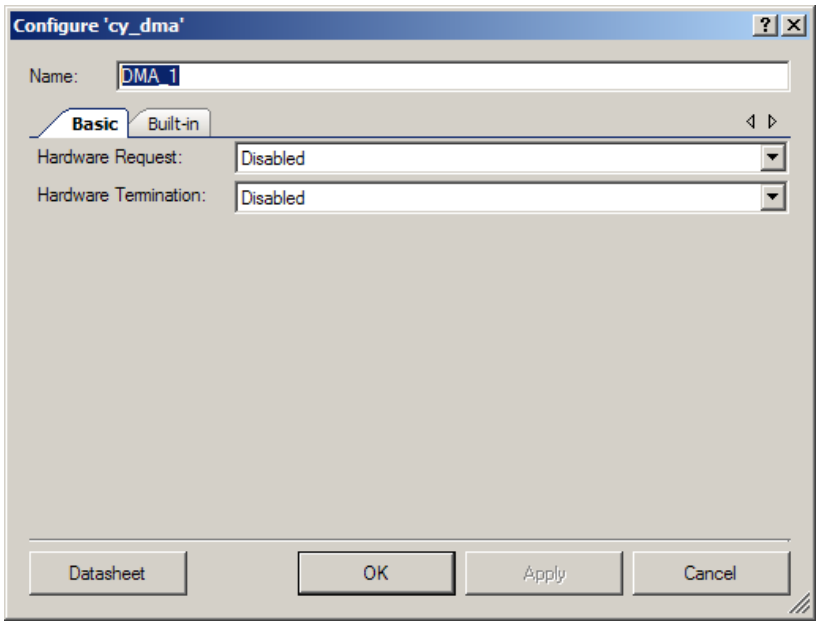
trq 终端连接至一个能够终止 DMA 操作的组件。当组件获知无数据可用时，可能需要提供 DMA 的数据。组件使用该信号终止操作。

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。

Component Parameters（组件参数）

将一个 DMA 组件拖放到您的设计上，并双击以打开**配置**对话框。



DMA 提供以下参数。

Hardware Request（硬件要求）

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

- **Disabled**（禁用）— 不显示 drq 终端。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。该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。
- **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。
- **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。

Hardware Termination（硬件终止）

该参数可设为**启用**或**禁用**。“启用”添加了 trq 终端，允许终止来自硬件的 DMA 请求。当终端被禁用时，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。

Resources（资源）

	Digital Blocks（数字模块）					API Memory (Bytes)（存储器（字节））		
DRQ	Datapaths（数据路径）	Macro cells（宏单元）	Status Registers（状态寄存器）	Control Registers（控制寄存器）	Counter7（计数器 7）	Flash（闪存）	RAM（随机访问内存）	Pins (per External I/O)（引脚（每个外部 I/O））
1	N/A（不可用）	N/A（不可用）	N/A（不可用）	N/A（不可用）	N/A（不可用）	2332	7	0

Application Programming Interface（应用程序编程接口）

应用程序编程接口 (API) 子程序允许您使用软件配置组件。下表列出了每个函数的接口，并进行了说明。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

Function（函数）	Description（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)

Description: （说明）： 分配一个 DMA 通道，并将其初始化，以供调用方使用。

Parameters: （参数）： uint8 burstCount。指定该 TD 需分割为多大的突发（1 至 127）。突发大小应为 spoke 大小的倍数。

如果该值为零，则整个传输在一次突发中完成。在这种情况下，TD 的传输计数参数决定一次突发中传输的字节数。

(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突发：

Value（值）	Action（操作）
0	将自动请求并执行首个突发后的所有后续突发
1	必须单独请求首个突发后的所有后续突发。

uint16 upperSrcAddress: 源地址的高 16 位。

uint16 upperDestAddress: 目标地址的高 16 位。

Return Value: （返回值）： uint8：DMA 活动的调用方能使用的通道。如果无通道剩余，则返回 DMA_INVALID_CHANNEL (0xFF)。

Side Effects: （副作用）： None（无）

void DMA_DmaRelease(void)

Description:| （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。

Parameters: （参数）： None（无）

Return Value: （返回值）： None（无）

Side Effects: （副作用）： None（无）

DMA 库 API（由所有 DMA 实例共享）

DMA Controller Functions（DMA 控制器函数）

Function（函数）	Description（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)

Description:
（说明）： 为所有要分配的 TD 创建链接列表。该函数由启动代码调用，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。

Parameters:
（参数）： None（无）

Return Value:
（返回值）： None（无）

Side Effects:
（副作用）： None（无）

uint8 CyDmacError(void)

Description:
（说明）： 返回 DMA_ERROR 类型的值，其中包含前一次失败的 DMA 操作的错误类型。

Parameters:
（参数）： None（无）

Return Value:
（返回值）： 从 DMA_ERROR 类型返回错误数据（4 位）。

Bit（位）	Define（定义）	Description（说明）
Bit 3（位 3）	DMAC_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。
Bit 2（位 2）	DMAC_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。
Bit 1（位 1）	DMAC_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。

Side Effects:
（副作用）： None（无）

void CyDmacClearError(uint8 error)

Description:
（说明）： 清除 DMAC 错误寄存器中的错误位。

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

Bit（位）	Define（定义）	Description（说明）
Bit 3（位 3）	DMAC_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。
Bit 2（位 2）	DMAC_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。
Bit 1（位 1）	DMAC_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。

Return Value:
（返回值）： None（无）

Side Effects:
（副作用）： None（无）

uint32 CyDmacErrorAddress(void)

Description:
（说明）： 当 BUS_TIMEOUT、UNPOP_ACC 以及 PERIPH_ERR 发生时，错误地址会被写入错误地址寄存器并可由该函数读取。如果存在多个错误，则仅保存首个错误的地址。

Parameters:
（参数）： None（无）

Return Value:
（返回值）： 引起错误的地址。

Side Effects:
（副作用）： None（无）

Channel Specific Functions（通道特定函数）

Function（函数）	Description（说明）
CyDmaChAlloc()	分配一个 DMA 通道以供调用方使用。

Function（函数）	Description（说明）
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。

uint8 CyDmaChAlloc(void)

Description:
（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。

Parameters:
（参数）： None（无）

Return Value:
（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 DMA_INVALID_CHANNEL。

Side Effects:
（副作用）： None（无）

cystatus CyDmaChFree(uint8 chHandle)

Description:
（说明）： 释放由 CyDmaChAlloc() 分配的通道句柄。

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

Return Value:
（返回值）： 如果成功，则返回 CYRET_SUCCESS。

如果 **chHandle** 无效，则返回 CYRET_BAD_PARAM。

Side Effects:
（副作用）： None（无）

cystatus CyDmaChEnable(uint8 chHandle, uint8 preserveTds)

Description:
（说明）：

启用 DMA 通道。软件或硬件请求仍必须在通道被执行前发出。

Parameters:
（参数）：

uint8 chHandle：CyDmaChAlloc() 或 DMA_DmaInitalize() 先前所返回的句柄。

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。

Value（值）	Action（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，编号与通道编号相等的 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** 被设为 0，请勿链接回已完成 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。

Return Value:
（返回值）：

如果成功，则返回 CYRET_SUCCESS。

如果 **chHandle** 无效，则返回 CYRET_BAD_PARAM。

Side Effects:
（副作用）：

None（无）

cystatus CyDmaChDisable(uint8 chHandle)

Description:
（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。

Parameters:
（参数）： uint8 chHandle：CyDmaChAlloc() 或 DMA_DmaInitalize() 先前所返回的句柄。

Return Value:
（返回值）： 如果成功，则返回 CYRET_SUCCESS。

如果 **chHandle** 无效，则返回 CYRET_BAD_PARAM。

Side Effects:
（副作用）： None（无）

cystatus CyDmaClearPendingDrq(uint8 chHandle)

Description:
（说明）： 清除待处理的 DMA 数据请求。

Parameters:
（参数）： chHandle：dma 通道的句柄。

Return Value:
（返回值）： 如果成功，则返回 CYRET_SUCCESS 或 CYRET_BAD_PARAM。

Side Effects:
（副作用）： None（无）

cystatus CyDmaChPriority(uint8 chHandle, uint8 priority)

Description:
（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。

Parameters:
（参数）： uint8 chHandle：CyDmaChAlloc() 或 DMA_DmaInitalize() 先前所返回的句柄。

uint8 priority: 要设置的通道优先级，0 - 7。

Return Value:
（返回值）： 如果成功，则返回 CYRET_SUCCESS。

如果 **chHandle** 无效，则返回 CYRET_BAD_PARAM。

Side Effects:
（副作用）： None（无）

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

Description:
（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）。

Parameters:
（参数）：
uint8 chHandle：CyDmaChAlloc() 或 DMA_DmaInitalize() 先前所返回的句柄。

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。

Return Value:
（返回值）：
如果成功，则返回 CYRET_SUCCESS。

如果 **chHandle** 无效，则返回 CYRET_BAD_PARAM。

Side Effects:
（副作用）：
None（无）

cystatus CyDmaChSetInitialTd(uint8 chHandle, uint8 startTd)

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

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

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。

Return Value:
（返回值）：
如果成功，则返回 CYRET_SUCCESS。

如果 **chHandle** 无效，则返回 CYRET_BAD_PARAM。

Side Effects:|
（副作用）：
None（无）

cystatus CyDmaChSetRequest(uint8 chHandle, uint8 request)

Description:
（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。

Parameters:
（参数）：
uint8 chHandle：CyDmaChAlloc() 或 DMA_DmaInitalize() 先前所返回的句柄。

uint8 request: 以下常量之一。各常量均为 3 位的值。

Request Values（请求值）	Description（说明）
CPU_REQ	创建一个直接请求以启动 DMA 通道
CPU_TERM_TD	终止一个 TD
CPU_TERM_CHAIN	终止一个 TD 链

Return Value:
（返回值）：
如果成功，则返回 CYRET_SUCCESS。

如果 **chHandle** 无效，则返回 CYRET_BAD_PARAM。

Side Effects:
（副作用）：
None（无）

cystatus CyDmaChGetRequest(uint8 chHandle)

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

Parameters:
（参数）：
uint8 chHandle：CyDmaChAlloc() 或 DMA_DmaInitalize() 先前所返回的句柄。

Return Value:
（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 DMA_INVALID_CHANNEL。

Side Effects:
（副作用）：
None（无）

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

Description:
（说明）：

确定 DMA 通道的状态。

Parameters:
（参数）：

uint8 chHandle：CyDmaChAlloc() 或 DMA_DmaInitalize() 先前所返回的句柄。

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）。

Bit 1（位 1）	STATUS_TD_ACTIVE	0：通道当前未处于 DMAC 的处理中
		1：通道当前正处于 DMAC 的处理中
Bit 0（位 0）	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

Return Value:
（返回值）：

如果成功，则返回 CYRET_SUCCESS。

如果 **chHandle** 无效，则返回 CYRET_BAD_PARAM。

Side Effects:
（副作用）：

None（无）

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

Description:
（说明）：
设置通道的配置信息。

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

uint8 burstCount。指定数据传输需分割为多大的突发（1 至 127）。如果该值为零，则整个传输在一次突发中完成。

(uint8) requestPerBurst：数据整体可分为多个突发，如果完成事务处理必须如此的话：

Value（值）	Action（操作）
0	将自动请求并执行首个突发后的所有后续突发
1	必须单独请求首个突发后的所有后续突发。

uint8 tdDone0: 选择一个 TERMOUT0 中断线以发送完成信号。该线与 nrq 终端相连，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 终端相连，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 终端相连的信号将确定使用哪一个 TERMIN（终止请求）。

Return Value:
（返回值）：
如果成功，则返回 CYRET_SUCCESS。

如果 **chHandle** 无效，则返回 CYRET_BAD_PARAM。

Side Effects:
（副作用）：
None（无）

数据操作描述符函数

Function（函数）	Description（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)

Description:
（说明）： 分配一个 TD 以与分配的 DMA 通道配合使用。

Parameters:
（参数）： None（无）

Return Value:
（返回值）： TD 索引从零开始，将供调用方使用。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 DMA_INVALID_TD。

Side Effects:
（副作用）： None（无）

void CyDmaTdFree(uint8 tdHandle)

Description:
（说明）： 将一个 TD 返回至空闲列表中。

Parameters:
（参数）： uint8 tdHandle: TD 句柄由 CyDmaTdAllocate() API 返回

Return Value:
（返回值）： None（无）

Side Effects:
（副作用）： None（无）

uint8 CyDmaTdFreeCount(void)

Description:
（说明）： 返回可分配的空闲 TD 的数量。

Parameters:
（参数）： None（无）

Return Value:
（返回值）： 空闲 TD 的数量。

Side Effects:
（副作用）： None（无）

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

Description:
（说明）：

对 TD 进行配置。

Parameters:
（参数）：

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

uint16 transferCount: 该 TD 数据传输的大小（按字节计）。如果大小为零，则传输将无限期地进行下去。该参数被限制在 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 TD 的有效指针；DMA_END_CHAIN_TD 是链的末端。

uint8 配置: 存储配置位的位字段。

Configuration Options（配置选项）	Description（说明）
TD_SWAP_EN	执行字节序交换
TD_SWAP_SIZE4	交换大小 = 4 字节
TD_AUTO_EXEC_NEXT	当前 TD 完成后，链中的下一个 TD 将会自动触发.
TD_TERMIN_EN	如果 trq 输入线路发生上升沿，则终止该 TD。在一次突发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，该选项名称视实例而定：实例名称后加两个下划线。本例中，实例名称为 DMA。
TD_INC_DST_ADR	根据突发中每个数据操作的大小而定的增量 DST_ADR。
TD_INC_SRC_ADR	根据突发中每个数据操作的大小而定的增量 SRC_ADR。

Return Value:
（返回值）：

如果成功，则返回 CYRET_SUCCESS。

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

Side Effects:
（副作用）：

None（无）

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

Description:
（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。

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

uint16 * transferCount: 存储该 TD 数据传输的大小（按字节计）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: 存储配置位位字段的地址。见 CyDmaTdSetConfiguration()。

Return Value:
（返回值）：
如果成功，则返回 CYRET_SUCCESS。

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

Side Effects:
（副作用）：
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)

Description:
（说明）：
仅为该 TD 设置源和目标地址的低 16 位。

Parameters:
（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。

Return Value:
（返回值）：
如果成功，则返回 CYRET_SUCCESS。

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

Side Effects:
（副作用）：
None（无）

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

Description: **（说明）：**	仅为该 TD 设置源和目标地址的低 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。
Parameters: **（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。
Return Value: **（返回值）：**	如果成功，则返回 CYRET_SUCCESS。 如果 **tdHandle** 无效，则返回 CYRET_BAD_PARAM。
Side Effects: **（副作用）：**	None（无）

Sample Firmware Source Code（固件源代码示例）

PSoC Creator 在“查找示例项目”对话框中提供了大量包括原理图和代码示例的示例项目。要获取组件特定的示例，请打开组件目录中的对话框或原理图中的组件实例。要获取通用的示例，请打开开始页或 **File（文件）**菜单中的对话框。根据需要，使用对话框中的**滤波器选项**可缩小可选项目的列表。

更多信息，请参考 PSoC Creator 帮助中的“查找示例项目”主题。

DMA Address Portability（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 设备而言并不相同。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 Devices（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, DMA_INVALID_TD, DMA_1__TD_TERMOUT_EN |
TD_INC_SRC_ADR | 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);
```

From Flash to RAM（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, DMA_INVALID_TD, DMA_1__TD_TERMOUT_EN |
TD_INC_SRC_ADR | 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);
```

From Flash to DAC（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, DMA_INVALID_TD, 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 Devices（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, DMA_INVALID_TD, DMA_1__TD_TERMOUT_EN |
TD_INC_SRC_ADR | 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, DMA_INVALID_TD, DMA_1__TD_TERMOUT_EN |
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);
```

Boundary Conditions（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，操作就会失败且无任何通知。这点仅对 PSoC 5 器件很重要，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。