

PSoC® 3 および PSoC 5LP - DMA 入門

著者: Anu M D、Lakshmi Natarajan

関連プロジェクト: あり

関連製品ファミリ: すべての PSoC® 3 および PSoC 5LP 製品

ソフトウェア バージョン: PSoC® Creator™ 3.0 SP1 以降

関連アプリケーション ノート: [AN61102](#)、[AN84810](#)

概要

AN52705 は PSoC® 3 と PSoC 5LP のダイレクト メモリ アクセス (DMA) の概要を説明します。PSoC DMA は CPU 介入なしで、オンチップ ペリフェラルとメモリ間でデータを転送することができます。アプリケーション ノートでは、サンプルプロジェクトを使用して、ペリフェラルとメモリ間、メモリとペリフェラル間、ペリフェラル間、メモリ間での簡単なデータ転送のために DMA をコンフィギュレーションする方法を示します。

目次

はじめに.....	1	著者について.....	18
DMA の基本的な概念	2	付録 A: DMA コンフィギュレーション手順.....	19
DMA コンフィギュレーション	4	その他の重要な DMA API 関数.....	21
チャンネル コンフィギュレーション.....	4	付録 B: DMA ウィザードによるコンフィギュレーション.....	21
TD コンフィギュレーション.....	4	付録 C: DMA チャンネルの優先順位の設定	23
DMA コンポーネントの概要.....	6	付録 D: サンプル プロジェクト – テスト セットアップ.....	24
DMA コンポーネントのハードウェア接続	6	例 1: ペリフェラル間の転送 – Eg1_ADC_DMA_DAC	24
DMA のファームウェア コンフィギュレーション.....	7	例 2: ペリフェラルからメモリへの転送 –	
例 1: ペリフェラル間の転送.....	8	Eg2_ADC_DMA_Mem	24
例 1 の DMA コンフィギュレーション.....	9	例 3: メモリからペリフェラルへの転送 –	
例 1 のプロジェクト ファイル.....	9	Eg3_Mem_DMA_DAC	25
例 1 の DMA のコンフィギュレーション コード.....	10	例 4: メモリ間の転送 – Eg4_Mem_DMA_Mem	25
例 2: ペリフェラルからメモリへの転送.....	11	例 5: TD チェーン化 – Eg5_TD_Chaining.....	26
例 2 の DMA のコンフィギュレーション.....	12	付録 E: よくある質問.....	27
例 2 のプロジェクト ファイル.....	12		
例 3: メモリからペリフェラルへの転送.....	13		
例 3 の DMA コンフィギュレーション.....	14		
例 3 のプロジェクト ファイル.....	14		
例 4: メモリ間の転送.....	15		
例 4 の DMA コンフィギュレーション.....	16		
例 4 のプロジェクト ファイル.....	16		
例 5: TD チェーン化.....	17		
例 5 の DMA コンフィギュレーション.....	18		
例 5 のプロジェクト ファイル.....	18		
まとめ.....	18		

はじめに

PSoC 3 と PSoC 5LP の DMA コントローラー (DMAC) は CPU 介入なしで転送元から転送先にデータを転送することができます。これにより、DMA がデータ転送を行っている間、CPU が他のタスクを処理することが可能になります。よって、「マルチプロセッシング」の環境を実現できます。

PSoC の DMA コントローラー (DMAC) は非常に柔軟です。ADC、DAC、フィルター、USB、UART、および SPI などのオンチップ ペリフェラルとメモリ間でデータをシームレスに転送することができます。24 の独立した DMA チャンネルがあります。

本アプリケーション ノートでは、単純なデータ転送のために DMA をコンフィギュレーションする方法について説明します。以下のように異なるタイプの DMA 転送を示すプロジェクトを含みます。

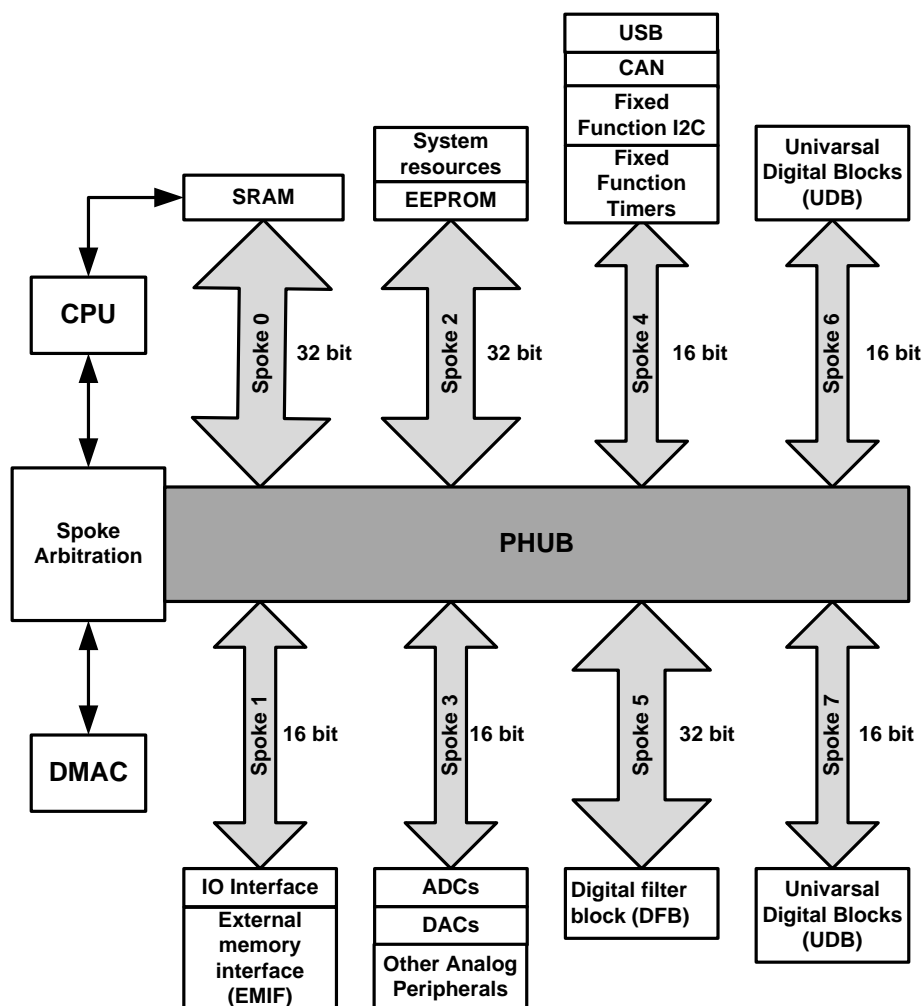
- ペリフェラルからメモリへ
- メモリからペリフェラルへ
- ペリフェラル間
- メモリ間

本アプリケーション ノートは、ユーザーが PSoC 3 または PSoC 5LP の PSoC Creator を使ったアプリケーションの開発に精通していることを前提としています。PSoC 3 または PSoC 5LP の初心者である場合、「[AN54181 - PSoC 3 入門](#)」および「[AN77759 - PSoC 5 入門](#)」を参照してください。DMA の高度なトピックは「[AN84810](#)」に記載されています。PSoC Creator の初心者である場合、「[PSoC Creator のホームページ](#)」を参照してください。

DMA の基本的な概念

PSoC 3 および PSoC 5LP の DMAC は図 1 に示すように、オンチップ ペリフェラル間の接続をするペリフェラル HUB (PHUB) と呼ばれる中央ハブの一部です。DMAC は 2 つの PHUB バス マスターの 1 つです。

図 1. ペリフェラル ハブ (PHUB)



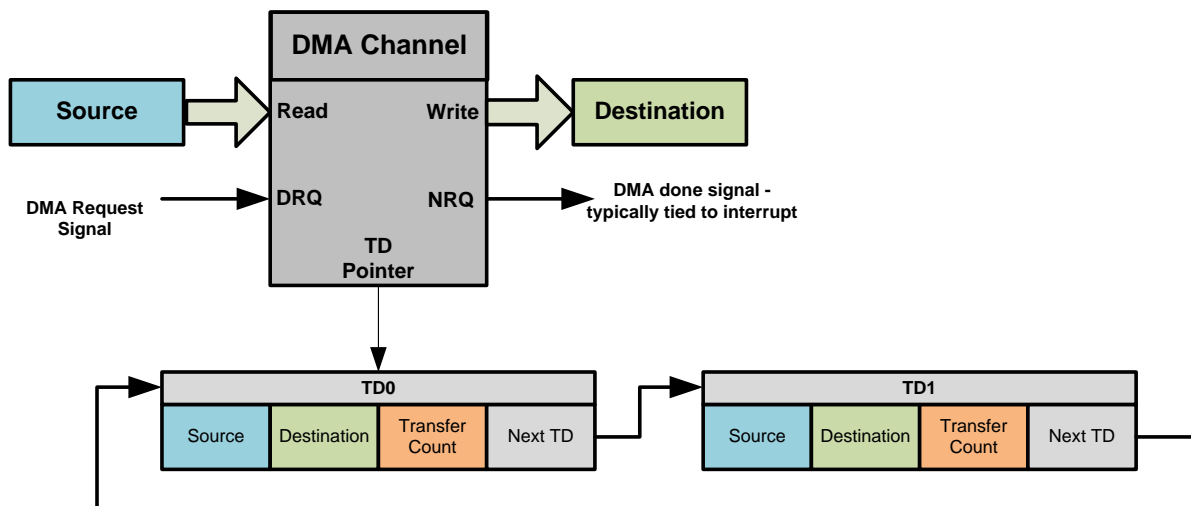
PHUB はスポークと呼ばれる 8 つのデータ バスを持っています。各スポークは、CPU と DMAC を 1 つまたは複数のペリフェラルに接続します。スポークの幅は、16 ビットまたは 32 ビットのどちらかです。スポークに接続するペリフェラルの幅は、8 ビット、16 ビット、または 32 ビットです。

通常、ペリフェラルのデータ幅は、ペリフェラルが接続しているスポークのデータ幅と等しい、またはそれ以下です。ペリフェラルのデータ幅が接続するスポークのデータ幅よりも大きい場合、PHUB はスポークの幅でペリフェラルとのトランザクションを行うことができます。

PHUB には CPU と DMAC という 2 つのバス マスターがあります。CPU と DMAC は、同時に異なる PHUB スポークにアクセスすることができます。CPU と DMAC が同時に同じスポークにアクセスしようすると、バス アービトレーションが発生します。詳細については、「[PSoC 3, PSoC 5LP テクニカル リファレンス マニュアル](#)」を参照してください。

24 の DMA チャンネルのそれぞれは、独立してデータを転送することができます。各チャンネルは、[図 2](#) に示すように、トランザクション ディスクリプタ (TD) のチェーンを持っています。TD には、ソース アドレス、宛先アドレス、転送回数、およびチェーン内の次の TD などの情報が含まれています。TD は最大 128 個あります。チャンネルと TD の組み合わせにより、完全な DMA 転送が定義されます。

図 2 DMA チャンネル



それぞれの DMA チャンネルは、トランザクションを開始する個別の DMA 要求入力を有しています。DMA 要求は、CPU またはペリフェラルのいずれかにより起動されます。DMA 要求が

受信されると、DMAC は、転送元と転送先に接続するスポークにアクセスし、チャンネルと関連 TD で構成された通りにデータを移動します。

DMA コンフィギュレーション

DMA 転送は、チャンネルと TD コンフィギュレーションレジスタを使用しコンフィギュレーションされます。図 3 は、チャンネルおよび TD コンフィギュレーション パラメータを示しています。

図 3. DMA コンフィギュレーション

Channel Configuration	TD Configuration
Source Address (Upper 16 bits)	Source Address (Lower 16 bits)
Destination Address (Upper 16 bits)	Destination Address (Lower 16 bits)
Burst Count (1-127)	Transfer Count (0 to 4095)
Request per Burst (TRUE or FALSE)	TD Property
First TD of Channel	Next TD
Preserve TD (TRUE or FALSE)	

チャンネル コンフィギュレーション

DMA チャンネル コンフィギュレーション パラメータを以下に説明します。

- **上位のソース アドレス (16 ビット)**
32 ビットのソース アドレスの上位 16 ビットは、チャンネル コンフィギュレーション レジスタでコンフィギュレーションされます。
- **上位の宛先アドレス (16 ビット)**
32 ビットの宛先アドレスの上位 16 ビットは、このチャンネル コンフィギュレーション レジスタでコンフィギュレーションされています。
- **バースト カウント (1~127)**
スポークを開放する前に、DMA チャンネルがソースから転送先に送信しなければならないバイト数を定義します。DMAC は、スポークを取得し、ソースから転送先へ指定されたバイト数を転送した後、スポークを開放します。次のバースト転送のためには、スポークを再び取得します。
内部スポークの DMA 転送のバースト カウントを 16 以下に制限します。

■ バーストごとの要求 (0 または 1)

DMA データ転送を完了するために複数のバースト転送が必要な場合、このビットはバーストの特性を決定します。

0: 最初のバーストに後続するすべてのバーストは個別の DMA 要求なしで自動的に転送される。最初のバースト転送だけは、DMA 要求が必要。

1: すべてのバースト転送は、個別の DMA 要求が必要。

■ 最初の TD

チャンネルと関連付けられている最初の TD を定義します。最初の TD へのポインタは、チャンネル コンフィギュレーション メモリに格納されます。後続の TD ポインタが連結リストと同じように TD コンフィギュレーション メモリに格納されます。

■ TD の保存 (0 または 1)

次の DMA 転送に再利用するために元の TD コンフィギュレーションを保存するかどうかを定義します。通常、TD コンフィギュレーションが保持されます。

0: TD コンフィギュレーションを保持しない。

1: TD コンフィギュレーションを保持する。

TD が保存された場合、チャンネルは (チャンネル番号に対応する) 個別の TD メモリを使用して進行中のトランザクションを追跡します。別の方法として、元の TD コンフィギュレーション レジスタが、進行中のトランザクションを追跡するためにアクティブなレジスタとして使用されます。

TD コンフィギュレーション

TD コンフィギュレーション パラメータを以下に説明します。

■ 下位のソース アドレス (16 ビット)

32 ビットのソース アドレスの上位 16 ビットは、TD コンフィギュレーション レジスタでコンフィギュレーションされています。

■ 下位の宛先アドレス (16 ビット)

32 ビットの宛先アドレスの下位 16 ビットです。

■ 転送カウント (0~4095 バイト)

転送元から転送先に転送する総バイト数です。

転送カウントは、バースト カウント パラメータと共に使用されます。例えば、16 ビットペリフェラルからメモリ バッファに 2 バイトのデータ ワードを 50 個移動する場合、バースト カウントは 2 に設定され、転送カウントは 100 に設定されます。

■ 次の TD

連結リストと同様な次の TD です。

■ TD のプロパティ

表 1 は、TD プロパティ コンフィギュレーション レジスタのビット フィールドで定義された TD プロパティを示します。

表 1. TD のプロパティ

プロパティ	説明
Increment Source Address (ソースアドレスのインクリメント)	このビットが設定された場合、ソースアドレスは、DMA トランザクションが進行するにつれてインクリメントされる
Increment Destination Address (宛先アドレスのインクリメント)	このビットが設定された場合、宛先アドレスは、DMA トランザクションが進行するにつれてインクリメントされる
Swap Enable (スワップ イネーブル)	設定された場合、DMA は転送元から転送先にデータを転送する間、データ バイトをスワップする
Swap Size (スワップ サイズ)	スワップ イネーブルが設定された場合、実行されるスワップのサイズを定義する 0: DMA 転送中に 2 バイトごとにエンディアン変換される 1: DMA 転送中に 4 バイトごとにエンディアン変換される
Auto Execute Next TD (次の TD の自動実行)	0: チェーン内の次の TD は、次の DMA 要求が発行された後にのみ実行される 1: チェーン内の次の TD は、現在の TD 転送が完了すると自動的に実行される
DMA Completion Event (DMA 完了イベント)	設定された場合、データ転送が完了した後に DMA「完了信号」を生成する。これは通常、転送が完了した後に割り込みを生成するために使用される
Enable TD termination (TD 中止のイネーブル)	設定された場合、ハードウェアの信号を使用して進行中のトランザクションを中止することができる

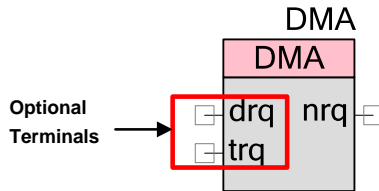
PSoC 3 Keil コンパイラは、16 ビットおよび 32 ビットの変数を格納するためにビッグ エンディアン形式を使用しています。しかし PSoC 3 ペリフェラルのレジスタはリトル エンディアン形式を使用しています。このため、DMA は、PSoC 3 内のペリフェラル レジスタとメモリ間でマルチバイトのデータを転送する時にバイトをスワップするように設定される必要があります。PSoC 5LP の場合、ペリフェラルとメモリの両方が同じエンディアン形式を使用しているため、バイトスワップの設定は不要です。

PSoC Creator を使用して DMA をコンフィギュレーションする方法を見てください。

DMA コンポーネントの概要

図 4 は、PSoC Creator 内の DMA チャンネルのコンポーネントを示しています。このコンポーネントは、Component Catalog 内の「Systems」タブの下にあります。

図 4. DMA チャンネル コンポーネント

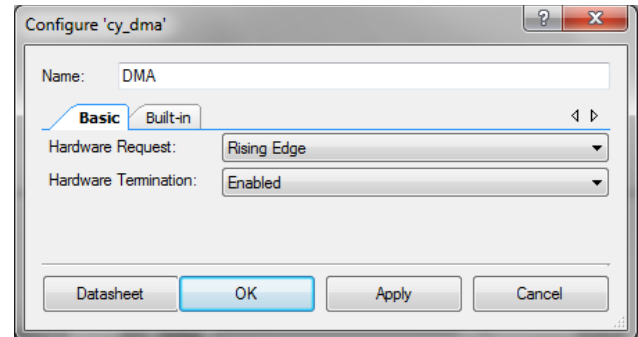


DMA チャンネル コンポーネントと関連する API は、データ転送を目的として DMA をコンフィギュレーションするために使用されます。

DMA コンポーネントのハードウェア接続

図 5 に示すように、コンポーネントのコンフィギュレーション ウィンドウで、次のパラメータを設定することができます。

図 5. DMA コンポーネント コンフィギュレーション



Hardware Request (ハードウェア リクエスト) (drq) : この設定は、DMA チャンネルをトリガーするために使用される信号の種類 (立ち上がりエッジ/レベル) を定義します。「Disabled」を除き、このパラメータを選択すると、drq 端子がコンポーネントに加えられます。drq は、DMA チャンネルをトリガーするために、任意のハードウェア信号に接続することができます。

drq 端子がない場合、DMA トランザクションをトリガーできるのは CPU のみとなります。

このパラメータは、「derived」に設定された場合、DMA トリガーの種類 (エッジ/レベル) は、DMA のトリガー ソースから判定されます。詳細情報については、「[DMA コンポーネント データシート](#)」を参照してください。

Hardware Termination (ハードウェア ターミネーション) (trq): このオプションを「true」に設定した場合、コンポーネントではもう 1 つの入力端子 (trq) が表示されます。TD の中止を有効にした場合、この端子上の立ち上がりエッジは、進行中の DMA トランザクションを停止します。trq は、進行中の DMA バースト トランザクションがある場合にのみ、TD チェーンを中止することに注意してください。詳細については、コンポーネントのデータシートを参照してください。

Transfer complete (転送完了) (nrq): DMA 転送が完了したことを示すために、転送が終了する DMA チャンネルの NRQ 端子で 2 バス クロックのパルス幅を生成するように TD をコンフィギュレーションすることができます。その後の動作のために、nrq 端子を割り込み、または別のコンポーネントに接続することができます。

nrq 端子で信号を生成するかどうかを、そして trq を使用して TD の中止を有効にするかどうかを定義するために、TD プロパティを設定します。

DMA のファームウェア コンフィギュレーション

DMA コンポーネントは、プロジェクトのビルド プロセス中に各 DMA インスタンスのソース ファイルと対応するヘッダ ファイルを生成します。デザインに DMA_1 という名前を持つ DMA コンポーネント インスタンスが存在している場合、「DMA_1_dma.c」と「DMA_1_dma.h」というファイルがビルド プロセス中に作成されます。これらのファイルは、DMA チャンネルを初期化するために使用する「DmaInitialize」API を含みます。他のチャンネルおよび TD コンフィギュレーション機能は、「Generated Source」フォルダ内の「CyDmac.c」と「CyDmac.h」ファイルに含まれています。

DMA のファームウェア コンフィギュレーションの手順は次の通りです。

1. DMA チャンネルを開始

```
Channel_Handle = DMA_DmaInitialize(DMA_BYTES_PER_BURST, DMA_REQUEST_PER_BURST,  
                                   HI16(Source Address), HI16(Destination Address))
```

2. TD のインスタンスを作成

```
TD_Handle = CyDmaTdAllocate();
```

3. TD コンフィギュレーションを設定

```
CyDmaTdSetConfiguration(TD_Handle, Transfer_Count, Next_TD, TD_Property);
```

4. TD アドレスを設定

```
CyDmaTdSetAddress(TD_Handle, LO16(Source Address), LO16(Destination Address))
```

5. チャンネルの初期 TD を設定

```
CyDmaChSetInitialTd(Channel_Handle, TD_Handle)
```

6. DMA チャンネルを有効化

```
CyDmaChEnable(Channel_Handle, preserve_TD)
```

上記のファームウェアの手順は、[19 ページ](#)の「[付録 A: DMA コンフィギュレーション手順](#)」で詳しく説明されます。DMA ウィザードを使用して DMA チャンネルをコンフィギュレーションするコードを自動的に生成することができます。詳細については、[21 ページ](#)の「[付録 B: DMA ウィザードによるコンフィギュレーション](#)」を参照してください。

DMA ウィザードは PSoC の限られたペリフェラルでの DMA トランザクションのみをサポートしていることに注意してください。DMA ウィザードがペリフェラルをサポートしない場合、付録 A で詳しく説明する機能を使用して、手動で DMA チャンネルをコンフィギュレーションする必要があります。

以下は、メモリとペリフェラル間の DMA 転送を行う方法を詳細に説明する 4 つの例です。5 番目の例では、複数 TD のチェーンを作成する方法を説明します。

例 1: ペリフェラル間の転送

この例では、DMA を使用してペリフェラル間の転送、つまり図 6 のような ADC データ出力レジスタから DAC データ入力レジスタへの転送を行う方法を示しています。

図 6. ペリフェラル間の転送のブロック図

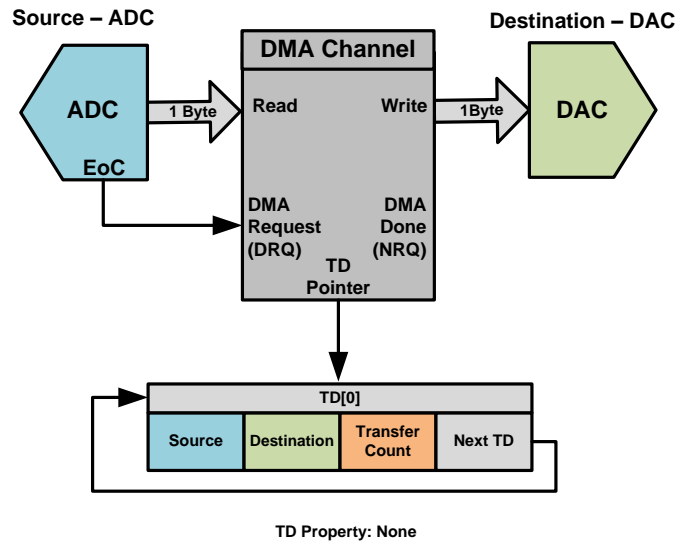
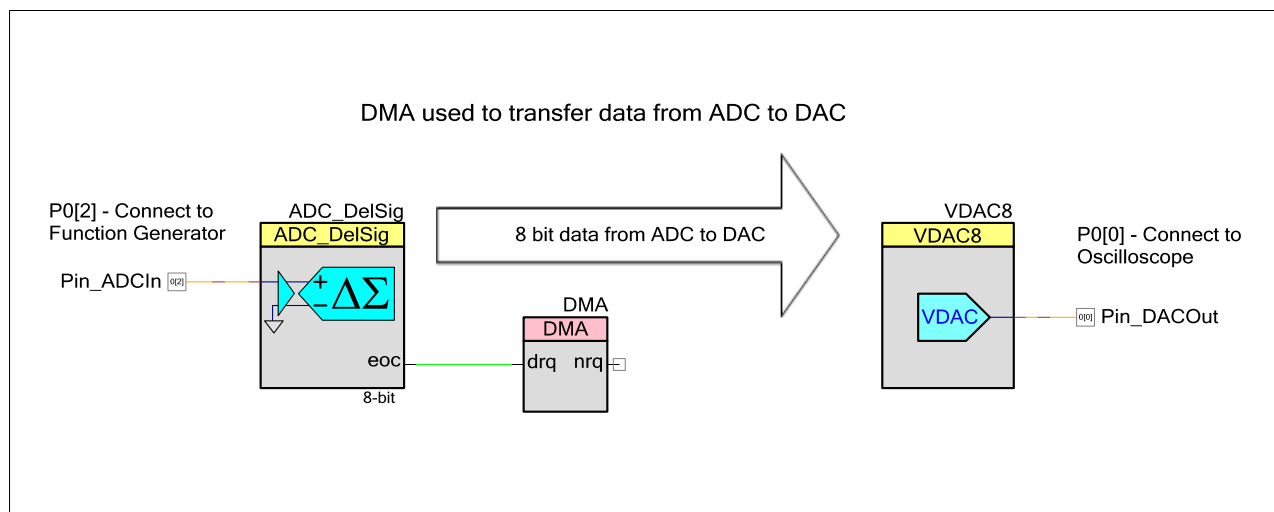


図 7 に示すように、ADC は、シングルエンドの 8 ビット電圧 DAC である VDAC のデータ形式に一致するように、シングルエンドで機能する 8 ビット構成のものとしてコンフィギュレーションされます。ADC の結果が出ると、ADC がデータ転送の要求を発行することができるように、DMA チャンネルのハードウェ

ア要求 (DRQ) が有効化され、ADC の EoC 信号に接続されます。

この要求を受信した後、DMA チャンネルは ADC の出力レジスタから 1 データ バイトを読み出して DAC データ レジスタに書き込みます。

図 7. ペリフェラル間の転送のトップ デザイン



例 1 の DMA コンフィギュレーション

このプロジェクトの DMA チャンネルおよび TD のコンフィギュレーションは表 2 および表 4 に示します。

表 2. チャンネル コンフィギュレーション設定

パラメーター	プロジェクト設定
ソース アドレスの上位	HI16(CYDEV_PERIPH_BASE)
宛先アドレスの上位	HI16(CYDEV_PERIPH_BASE)
バースト カウント	1 (1 バイト)
バーストごとに要求	1 (True)
最初の TD	TD[0]
TD の保存	1 (True)

チャンネル コンフィギュレーションは、ソース アドレスと宛先アドレスの両方の 32 ビット アドレスには、上位 16 ビットがあります。

PSoC Creator の自動生成されたファイル *cydevice.h* に定義された CYDEV_PERIPH_BASE は、ADC と DAC を含むすべての PSoC ペリフェラルのベース アドレスを定義します。

HI16 は 32 ビット値の上位 16 ビットを返す PSoC Creator のマクロです。このマクロは、ソース アドレスと宛先アドレスの上位 16 ビットを取得するために使用されます。

別の方法として、表 3 に示すように、コンポーネントのレジスタに対応するソース アドレスと宛先アドレスの上位アドレスを割り当てることができます。アドレスの定義はそれぞれ、「ADC_DelSig.h」と「VDAC8.h」のコンポーネント ファイルにあります。

表 3. 代替の上位アドレス

パラメーター	プロジェクト設定
ソース アドレスの上位	HI16(ADC_DelSig_DEC_OUTSAMP_PTR)
宛先アドレスの上位	HI16(VDAC8_DATA_PTR)

TD は、連鎖された TD のアレイとして考えることができます。この例の場合、1 つの単一要素のアレイ TD[0]のみが必要です。

表 4. TD[0] コンフィギュレーション設定

パラメーター	プロジェクト設定
ソースアドレスの下位	LO16(ADC_DelSig_DEC_OUTSAMP_PTR)
宛先アドレスの下位	LO16(VDAC8_DATA_PTR)
転送カウント	1 (1 バイト)
TD プロパティ	なし(0)
次の TD	TD[0] (同じ TD にループバック)

LO16 マクロは、32 ビット値の下位 16 ビットを返します。

DMA チャンネルは各 DMA 要求に対して 1 バイトを移動する必要があるため、バースト カウントが 1 バイトに設定され、バーストごとの要求が「True」に設定されます。

次に実行される TD が (ループされた) 同じ TD に設定されるため、同じトランザクションが各 DMA 要求に応じて繰り返されます。TD の保存パラメーターが「True」に設定されます。

例 1 のプロジェクト ファイル

本アプリケーション ノートに添付する AN52705.zip ファイル内の Eg1_ADC_DMA_DAC プロジェクトはこの例をデモします。このプロジェクトのテスト方法の詳細については、「付録 D: サンプル プロジェクト - テスト セットアップ」を参照してください。

この例の DMA コンフィギュレーション コードを以下に記載します。DMA ウィザードを使用してこのコンフィギュレーション コードを生成する方法の詳細については、「付録 B: DMA ウィザードによるコンフィギュレーション」を参照してください。

例 1 の DMA のコンフィギュレーション コード

```
/* Define for DMA Configuration */
#define DMA_BYTES_PER_BURST 1
#define DMA_REQUEST_PER_BURST 1
#define DMA_SRC_BASE (CYDEV_PERIPH_BASE)
#define DMA_DST_BASE (CYDEV_PERIPH_BASE)

/* Variable declarations for the DMA channel.
 * DMA_Ch is used to store the DMA channel */
uint8 DMA_Ch;
/* DMA_TD array is used to store all of the TDs associated with the channel
 * Since there is only one TD in this example, DMA_TD array contains only one element */
uint8 DMA_TD[1];

/* DMA Configuration steps */

/* Step 1 */
/* DMA Initializations done for both the DMA Channels
 * Burst count = 1, (8 bit data transferred to VDAC one at a time)
 * Request per burst = 1 (transfer burst only on new request)
 * High byte of source address = Upper 16 bits of ADC data register
 * High byte of destination address = Upper bytes of the VDAC8 data register
 * DMA_Ch holds the channel handle returned by the 'DmaInitialize' function. This is
 * used for all further references of the channel */
DMA_Ch = DMA_DmaInitialize(DMA_BYTES_PER_BURST, DMA_REQUEST_PER_BURST,
                          HI16(DMA_SRC_BASE), HI16(DMA_DST_BASE));

/* Step 2 */
/* Allocate TD for DMA Channel
 * DMA_TD[0] is a variable that holds the TD handle returned by the TD allocate function.
 * This is used for all further references of the TD */
DMA_TD[0] = CyDmaTdAllocate();

/* Step 3 */
/* Configure TD[0]
 * Transfer count = 1 (total number of bytes to transfer from the ADC to DAC)
 * Next Td = DMA_TD[0]. The same td has to repeat itself for every ADC EoC.
 * Configuration = No special TD configurations required */
CyDmaTdSetConfiguration(DMA_TD[0], 1, DMA_TD[0], 0);

/* Step 4 */
/* Configure the td address
 * Source address = Lower 16 bits of ADC data register
 * Destination address = Lower 16 bits of VDAC8 data register */
CyDmaTdSetAddress(DMA_TD[0], LO16((uint32)ADC_DelSig_DEC_SAMP_PTR),
                  LO16((uint32)VDAC8_Data_PTR));

/* Step 5 */
/* Map the TD to the DMA Channel */
CyDmaChSetInitialTd(DMA_Ch, DMA_TD[0]);

/* Step 6 */
/* Enable the channel
 * The Channel is enabled with Preserve TD parameter set to 1. This preserves the
 * original TD configuration and reload it after the transfer is complete so that the TD
 * can be repeated */
CyDmaChEnable(DMA_Ch, 1);
```

例 2: ペリフェラルからメモリへの転送

この例では、図 8 に示すように、ADC データ出力レジスタから 16 ビットのメモリ アレイへのペリフェラル-メモリ間の転送を実行する方法を示します。

図 8. ペリフェラルからメモリへの転送のブロック図

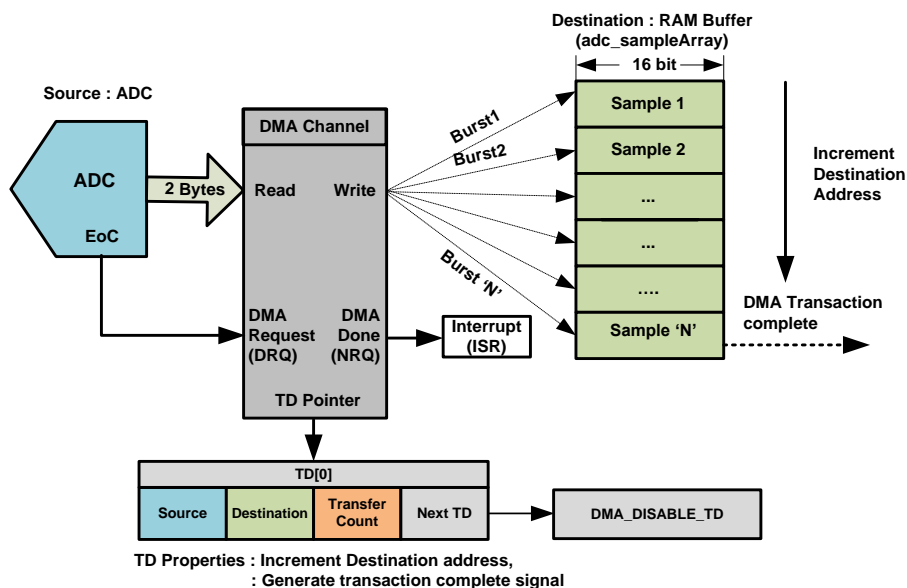


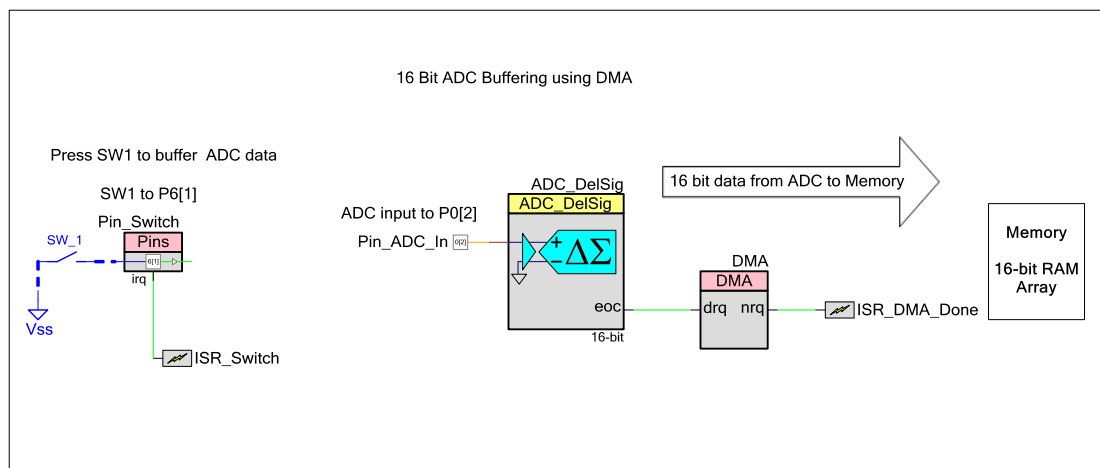
図 9 は、プロジェクトのトップ デザインを示します。Pin_Switch を押す度に、ISR_Switch がトリガーされ、DMA チャンネルを有効にするためにフラグが isr に設定されます。DMA チャンネルが有効になると、ADC からの EoC 信号が DMA チャンネル要求をアクティブにします。

各 DMA 要求に応じて、DMA は、転送元 (ADC 出力レジスタ) から 2 バイトを読み出し、転送先の RAM バッファに書き込み、宛先アドレスを 2 ずつインクリメントします。転送カウントは、各バースト転送後 2 ずつデクリメントされます。これは、転送カ

ウントが 0 になるまで繰り返します。転送カウンタが 0 になると、ISR_DMA_Done 割り込みをアクティブにする、DMA コンポーネントの NRQ 端子でのトランザクション完了信号が生成されます。

割込みサービス ルーチンでは、トランザクションが完了したことを示すためにフラグが設定されます。トランザクションが完了すると DMA チャンネルは無効になり、スイッチが再び押された時に有効になります。

図 9. ペリフェラルからメモリへの転送のトップ デザイン



例 2 の DMA のコンフィギュレーション

このプロジェクトの DMA チャンネルおよび TD のコンフィギュレーションは、表 5 および表 6 に示します。

表 5. チャンネル コンフィギュレーション設定

パラメーター	プロジェクト設定
ソース アドレスの上位	HI16(CYDEV_PERIPH_BASE)
宛先アドレスの上位	HI16(CYDEV_SRAM_BASE)
バースト カウント	2 (2 バイト)
バーストごとに要求	1 (True)
最初の TD	TD[0]
TD の保存	1 (True)

チャンネル コンフィギュレーションは、ソース アドレスと宛先アドレスの両方の 32 ビット アドレスには、上位 16 ビットがあります。ソース アドレスは、例 1 と同様です。

PSoC Creator の自動生成ファイル *cydevice.h* に定義された CYDEV_PERIPH_BASE は、SRAM のベース アドレスを定義します。これは、宛先アドレスの上位 16 ビットを指定するために使用されます。

別の方法として、RAM アレイ ポインタを HI16 マクロと併用して PSoC 5LP のソース アドレスの上位 16 ビットを指定することができます。この方法は PSoC 3 では使用できません。これは、PSoC 3 では、RAM 変数の上位 16 アドレス ビットがゼロですが、Keil コンパイラは、Keil 固有の情報を変数アドレスの上位 16 ビットに格納するためです。このため、PSoC3 の Keil コンパイラが併用される場合、HI16 (&adc_sampleArray) は不正なアドレスを返します。

この例では、各 DMA 要求に応じて 2 バイトの ADC 結果を ADC から RAM に転送する必要があるため、バースト カウントは 2 に設定され、バーストごとの要求は「true」に設定されます。

「TD の保存」は、元の TD 設定、すなわちソース アドレス、宛先アドレスおよび転送カウントが保存され、トランザクションを繰り返すことができるように、1 (TRUE) に設定されます。

表 6 に示すように、ソース アドレスと宛先アドレスの下位 16 ビットは、トランザクション ディスクリプタ (TD[0]) のコンフィギュレーションで指定されます。

表 6. TD[0]コンフィギュレーション設定

パラメーター	プロジェクト設定
ソースアドレスの下位	LO16(ADC_Delsig_DEC_OUTSAMP_PTR)
宛先アドレスの下位	LO16(&adc_sampleArray)
転送カウント	200 x 2 (サンプル数 x サンプル当たりのバイト数)
TD プロパティ	Increment Destination Address (転送先アドレスのインクリメント)、 Generate DMA done event (DMA 完了イベントの生成)、 「スワップ イネーブル」は PSoC 3 の場合にのみ必要 (TD_INC_DST_PTR DMA__TD_TERMOUT_EN TD_SWAP_EN)
次の TD	DMA_DISABLE_TD

転送カウントは、「バッファリングするサンプル数 x サンプル当たりのバイト数」に等しい「400」に設定されます。

TD プロパティは、各バースト転送後に宛先アドレスをインクリメントし、指定されたサンプル数がバッファリングされると、トランザクション完了信号を生成するように設定されます。PSoC 3 プロジェクトでは、TD は、TD プロパティ セクションで説明したように、ADC からメモリにデータを転送中にバイトをスワップするように構成されます。TD プロパティのそれぞれに対応するビットは、PSoC Creator により自動生成された CyDmac.h ファイルで定義されます。TD プロパティを設定するために、必要なプロパティビットフィールドは論理和されます。

指定されたサンプル数をバッファリングした後に DMA 転送を停止するために、TD[0]は、「DMA_DISABLE_TD」と連鎖され、よって DMA チャンネルは無効になります。

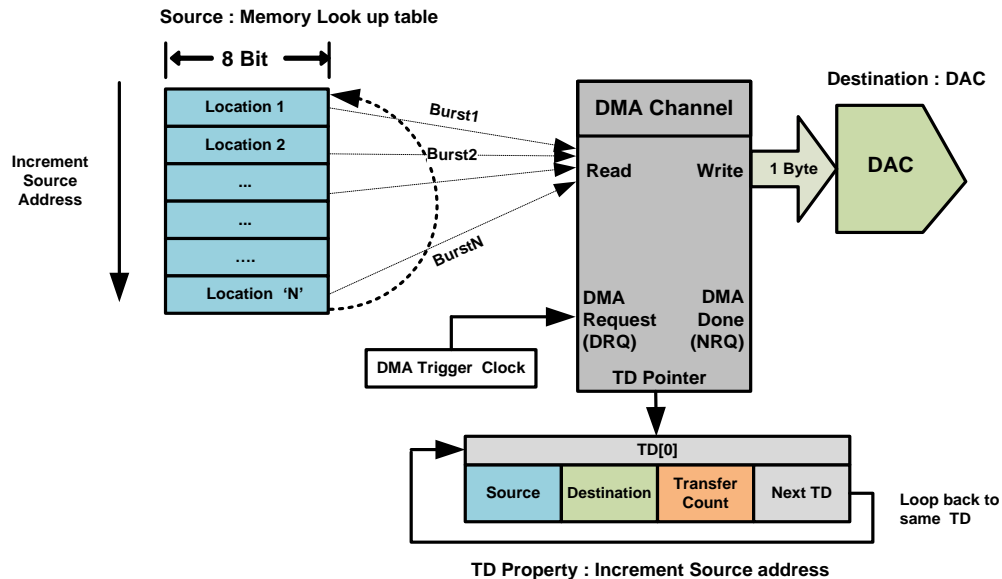
例 2 のプロジェクト ファイル

本アプリケーション ノートに添付する「AN52705.zip」ファイル内の Eg2_ADC_DMA_Mem プロジェクトはこの例をデモします。この例の DMA コンフィギュレーション コードは例 1 と同じです。関数に渡される引数は、上記のチャンネルおよび TD コンフィギュレーション表に記載されています。このプロジェクトのテスト方法の詳細については、「付録 D: サンプル プロジェクト - テスト セットアップ」を参照してください。

例 3: メモリからペリフェラルへの転送

この例では、DMA を使用してメモリからペリフェラルへのデータ転送を行う方法を示します。この例では、図 10 に示すように、DAC を使用して、波を発生させる方法をデモしています。

図 10. メモリからペリフェラルへの転送のブロック図



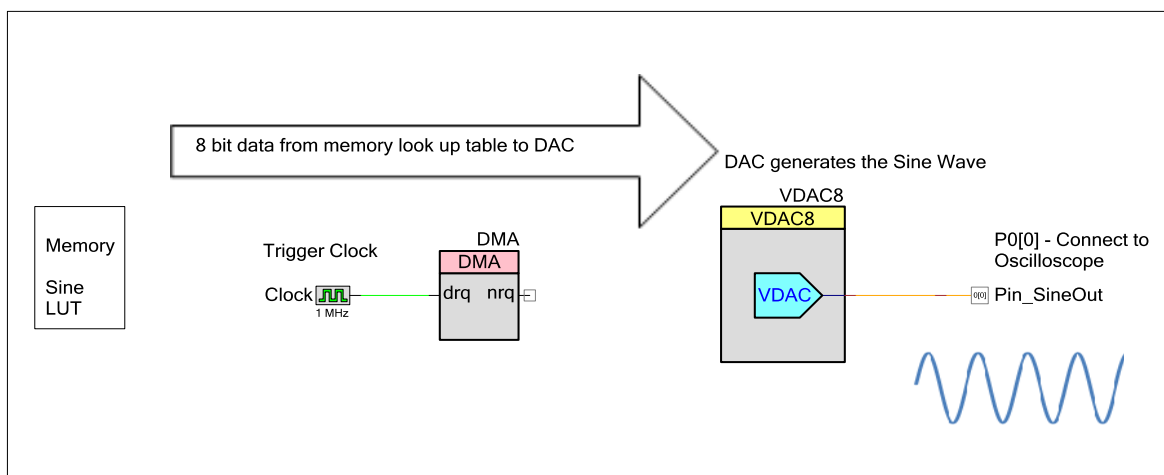
128 点の正弦ルックアップ テーブルがフラッシュ メモリに格納されます。正弦波を生成するために、これらの値は DMA を使用して順次に DAC に送信されます。図 11 は、プロジェクトのトップ デザインを示します。

定期的に DMA 要求 (drq) を生成するためにクロックコンポーネントを使用します。この要求を受信した後、DMA チャンネルはルックアップ テーブルから 1 データ バイトを取得し、DAC データレジスタに書き込みます。宛先アドレスは 1 つずつインクリメ

ントされ、転送カウントは各バースト転送後に 1 つずつデクリメントされます。これはテーブル内のすべての値が DAC に送信されることが完了するまで続きます。

TD コンフィギュレーションが保存され、連続的な正弦波を生成するように転送の終了時に再びロードされます。正弦波の周波数は、ルックアップ テーブル内のポイント数で分周される DMA トリガー クロック周波数に等しいです。

図 11. メモリからペリフェラルへの転送のトップ デザイン



例 3 の DMA コンフィギュレーション

このプロジェクトの DMA チャンネルおよび TD のコンフィギュレーションは表 7 と表 8 に示します。

表 7. チャンネル コンフィギュレーション

パラメーター	プロジェクト設定
ソース アドレスの上位	PSoC 3 の場合は HI16(CYDEV_FLS_BASE) PSoC 5LP の場合は HI16 (&sineTable)
宛先アドレスの上位	HI16(CYDEV_PERIPH_BASE)
バースト カウント	1 (1 バイト)
バーストごとに要求	1 (True)
最初の TD	TD[0]
TD の保存	1 (True)

DMA 転送元は、フラッシュ メモリに保存された正弦テーブル アレイです。PSoC5LP では HI16(&sineTable) がソース アドレスの上位 16 ビットを設定するのにに対し、PSoC 3 では HI16(CYDEV_FLS_BASE) が、前の例で述べた理由のため、ソース アドレスの上位 16 ビットを識別するために使用されます。

DMA チャンネルは、各 DMA 要求に際してルックアップ テーブルのアレイから DAC に 1 バイトを転送する必要があります。そのため、バースト カウントは 1 バイトに設定され、バーストごとの要求は「true」に設定されます。

元の TD コンフィギュレーションは、再利用のために保存されます。

表 8 に示すように、ソース アドレスと宛先アドレスの下位 16 ビットは LO16 マクロを使用して設定されます。

表 8. TD[0]コンフィギュレーション

パラメーター	プロジェクト設定
ソースアドレスの下位	LO16 (&sineTable)
宛先アドレスの下位	LO16(VDAC8_DATA_PTR)
転送カウント	128 (正弦ルックアップ テーブル内のバイト数)
TD プロパティ	ソース アドレスのインクリメント (TD_INC_SRC_ADR)
次の TD	TD[0] - 同じ TD に再びループバック

転送カウントは、正弦ルックアップ テーブル内のバイト総数に設定されます。

TD は、各バースト転送後にソース アドレス、すなわちルックアップ テーブル ポインタをインクリメントするようにコンフィギュレーションされます。

転送完了時に、正弦波の 1 つの完了サイクルが DAC 出力で生成されます。TD は保存され、連続波を生成するように自己にループバックされます。

例 3 のプロジェクト ファイル

本アプリケーション ノートに添付する AN52705.zip ファイル内の Eg3_Mem_DMA_DAC プロジェクトはこの例をデモします。この例の DMA コンフィギュレーション コードは例 1 と同じです。関数に渡される引数は、上記のチャンネルおよび TD コンフィギュレーション表に記載されています。このプロジェクトのテスト方法の詳細については、「付録 D: サンプル プロジェクト – テスト セットアップ」を参照してください。

例 4: メモリ間の転送

この例では、DMA を使用してメモリ間のデータ転送を行う方法を示します。また、CPU を使用して DMA チャンネルをトリガーする方法も示します。この例では、図 12 に示すように、8 バイトのフラッシュ アレイが、CPU 要求に応じて 8 バイトの RAM アレイにコピーされます。

図 12. メモリ間の転送時のブロック図

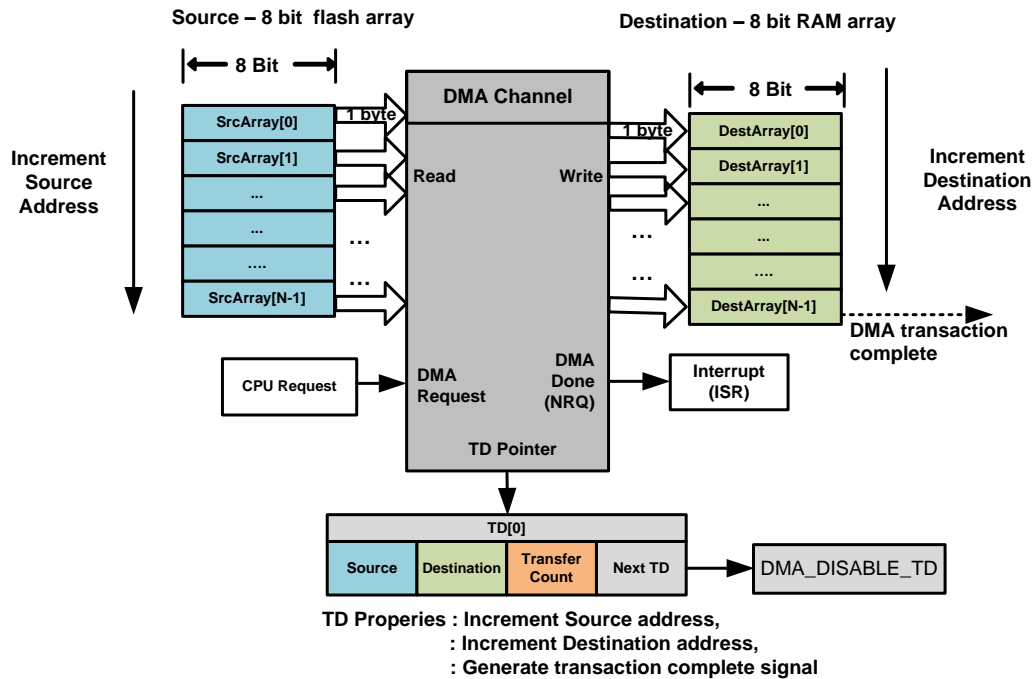


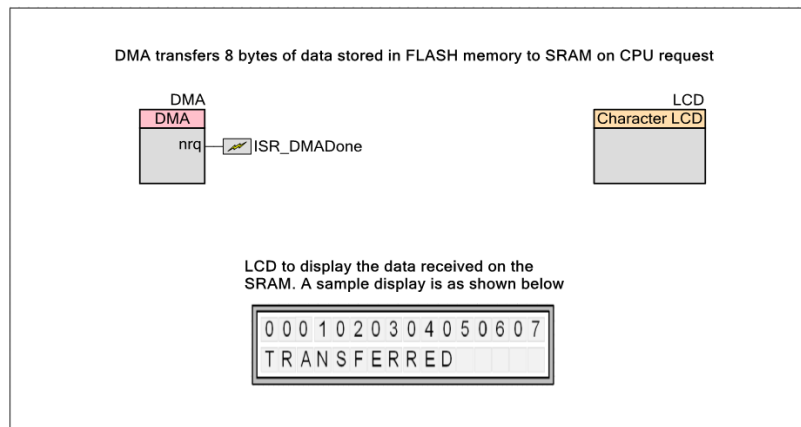
図 13 は、プロジェクトのトップ デザインを示します。CyDmaChSetRequest 関数はデバイスへ電源投入した約 1 秒後 DMA 転送をアクティブにするために使用されます。

CPU から要求を受信すると、DMA はチャンネルと TD コンフィギュレーション レジスタでコンフィギュレーションされたようにフラッシュ アレイから RAM アレイに 8 バイトを転送します。TD の

ソース アドレスと宛先アドレスは、転送の進行と共にインクリメントされます。

転送が完了すると、パルスが DMA の nrq 信号端子で生成されます。これは、転送が完了したことを示すフラグを設定する ISR_DMADone 割り込みをアクティブにします。RAM の新規内容は、その後 LCD に表示されます。

図 13. メモリ間の転送のトップ デザイン



例 4 の DMA コンフィギュレーション

このプロジェクトの DMA チャンネルおよび TD のコンフィギュレーションは表 9 と表 10 に示します。

表 9. チャンネル コンフィギュレーション

パラメーター	プロジェクト設定
ソース アドレスの上位	PSoC 3 の場合は、 HI16(CYDEV_FLS_BASE) PSoC 5LP の場合は、 HI16(&sourceArray)
宛先アドレスの上位	HI16(CYDEV_SRAM_BASE)
バースト カウント	1 (1 バイト)
バーストごとに要求	0 (False)
最初の TD	TD[0]
TD の保存	0 (False)

DMA 転送元は、フラッシュ メモリで定義された「sourceArray」です。宛先は RAM 内の「destinationArray」です。前の例で説明したようにフラッシュの宛先アドレスの上位 16 ビットは、PSoC 5LP では HI16(&sourceArray) で、PSoC 3 では HI16(CYDEV_FLS_BASE) で設定されます。同様に、SRAM 内の転送先アドレスの上位 16 ビットは、マクロ HI16(CYDEV_SRAM_BASE) を使用して設定されます。

DMA がフラッシュから 1 バイトごとに読み出し、RAM アレイに書き込むように、バースト カウントが 1 バイトに設定されます。データ転送を加速するためにバースト カウントを 8 バイトに設定することもできます。ただし、通常、他の DMA チャンネルもスプークを共用することが可能になるように、バースト カウントを低い値に設定する必要があります。

各バースト転送に個別の要求が必要ないように、「バーストごとの要求」パラメーターは「false」に設定されます。

表 10. TD[0] コンフィギュレーション

パラメーター	プロジェクト設定
ソース アドレス	LO16(&sourceArray)
宛先アドレス	LO16(&destinationArray)
転送カウント	8 (バイト)
TD プロパティ	Increment Source Address (ソース アドレスのインクリメント) Increment Destination Address (宛先アドレスのインクリメント) Generate DMA done signal (DMA 完了信号の生成) (TD_INC_SRC_ADR TD_INC_DST_ADR DMA_TDMOUT_EN)
次の TD	DMA_DISABLE_TD (0xFE)

ソース アドレスの下位 16 ビットと TD コンフィギュレーション用の宛先アドレスは、LO16 マクロによって識別されます。

総数 8 バイトがソース アドレスから宛先アドレスへ転送されるように、転送回数が「8」に設定されます。

TD は、各バースト転送後に、ソース アドレス、すなわちフラッシュ アレイ ポインタ、および宛先アドレス、すなわち RAM のアレイ ポインタをインクリメントするようにコンフィギュレーションされます。8 バイト全てがフラッシュから RAM アレイに転送された後、nrq ラインで TERMOUT パルスを送信するように TD がコンフィギュレーションされます。このパルスは、転送が完了したことを示すための ISR をトリガーするために使用されます。次の TD は、転送が終了した後に DMA チャンネルを無効にするために DMA_DISABLE_TD (0xFE) に設定されます。

トランザクションが一度のみ起こるため、TD コンフィギュレーションが保存される必要はありません。

例 4 のプロジェクト ファイル

本アプリケーション ノートに添付する AN52705.zip ファイル内の Eg4_Mem_DMA_Mem プロジェクトはこの例をデモします。この例の DMA コンフィギュレーション コードは例 1 と同じです。関数に渡される引数は、上記のチャンネルおよび TD コンフィギュレーション表に記載されています。このプロジェクトのテスト方法の詳細については、「付録 D: サンプル プロジェクト - テスト セットアップ」を参照してください。

例 5 の DMA コンフィギュレーション

このプロジェクトの DMA チャンネルおよび TD のコンフィギュレーションは、表 11、表 12、および表 13 に示します。

表 11. チャンネル コンフィギュレーション

パラメーター	設定
ソース アドレスの上位	HI16(CYDEV_PERIPH_BASE)
宛先アドレスの上位	HI16(CYDEV_SRAM_BASE)
バースト カウント	2 (2 バイト)
バーストごとに要求	1 (True)
最初の TD	TD[0]
TD の保存	1 (True)

チャンネルおよび TD のコンフィギュレーションは例 2 と同様です。トランザクションを連鎖するために TD[0]の「次の TD」パラメーターを TD[1]に設定します。その逆も同じです。

表 12. TD[0] コンフィギュレーション

パラメーター	プロジェクト設定
ソースアドレスの下位	LO16(ADC_Delsig_DEC_OUTSAMP_PTR)
宛先アドレスの下位	LO16(adc_sampleArray1)
転送カウント	N1x2 (サンプル数 x サンプル当たりのバイト数)
TD プロパティ	宛先アドレスのインクリメント: TD_INC_DST_ADR DMA 完了イベントの生成: DMA__TD_TERMOUT_EN PSoC 3に必要なスワップ イネーブル: TD_SWAP_EN
次の TD	TD[1]

表 13. TD[1] コンフィギュレーション

パラメーター	プロジェクト設定
ソースアドレスの下位	LO16(ADC_Delsig_DEC_OUTSAMP_PTR)
宛先アドレスの下位	LO16(adc_sampleArray2)
転送カウント	N2x2 (サンプル数 x サンプル当たりのバイト数)
TD プロパティ	宛先アドレスのインクリメント: TD_INC_DST_ADR DMA 完了イベントの生成: DMA__TD_TERMOUT_EN PSoC 3に必要なスワップ イネーブル: TD_SWAP_EN
次の TD	TD[0]

例 5 のプロジェクト ファイル

このアプリケーション ノートに添付する AN52705.zip ファイル内の Eg5_TD_Chaining は、TD チェーン化の例を示しています。この例の DMA コンフィギュレーション コードは例 1 と同じです。関数に渡される引数は、上記のチャンネルおよび TD コンフィギュレーション表に記載されています。このプロジェクトのテスト方法の詳細については、24 ページの「付録 D: サンプルプロジェクト - テスト セットアップ」を参照してください。

まとめ

このアプリケーション ノートでは、PSoC 3 と PSoC 5LP の DMA コントローラーについて説明しました。アプリケーション ノートでは、単純な PSoC Creator のサンプル プロジェクトを使用して、各種のデータ転送用 DMA をコンフィギュレーションする方法を示しました。詳細情報については、「PSoC 3 と PSoC 5LP テクニカル リファレンス マニュアル」と「PSoC Creator DMA コンポーネント データシート」を参照してください。

著者について

氏名: Anu MD

肩書き: シニアアプリケーション エンジニア

経歴: コーチンのモデル工科大学の電子通信工学学士。

連絡先: anmd@cyperess.com

付録 A: DMA コンフィギュレーション手順

ステップ 1: DMA チャンネルの初期化

```
Channel_Handle = DMA_DmaInitialize(
    DMA_BYTES_PER_BURST,
    DMA_REQUEST_PER_BURST,
    HI16(Source Address),
    HI16(Destination Address))
```

API 関数の `DmaInitialize()` は、次のようにいくつかの DMA チャンネル パラメーターをコンフィギュレーションします。

- **DMA_BYTES_PER_BURST:** 1 つのバーストで DMA チャンネルにより読み書きされるバイト数
例えば、8 ビットの ADC データを収集するために DMA を定義する場合、DMA チャンネルが各要求に応じて転送元から転送先へ 1 バイトを転送する必要があるため、このパラメーターを「1」に設定します。16 ビット ADC データを収集したい場合は、このパラメーターを「2」に設定します。
- **DMA_REQUEST_PER_BURST:** 各バーストに対して個別の要求が必要かどうか判定するパラメーター
「1」に設定すれば、各バースト転送を個別に要求する必要があります。0 に設定すれば、最初のバースト転送に後続するバースト転送は個別の DMA 要求なしで自動的に行われます。(最初のバースト転送だけは DMA 要求が必要です。)
- **HI16(Source Address):** ソース アドレスの上位 16 ビット。HI16 は 32 ビット値またはアドレスの上位 16 ビットを指定するために PSoC Creator により作成されるマクロです。
- **HI16(DestinationAddress):** 宛先アドレスの上位 16 ビット。前述の表に記載されているマクロを使用して、PSoC 3 のソース アドレスと宛先アドレスの上位 16 ビットを識別します。

PSoC 3 の Keil コンパイラは Keil 固有の情報を変数アドレスの上位 16 ビットに格納します。このため、表 14 に示す以下の定数を使用します。これらの定数は、特に DMA 転送の転送元または転送先が RAM またはフラッシュメモリの場合、PSoC 3 のソース アドレスと宛先アドレスの上位 16 ビットをコンフィギュレーションするために、HI16 マクロと共に `CyDevice.h` で定義されています。

表 14. 上位 16 ビットのアドレス マクロ

ソース	DMA_SRC_BASE
ペリフェラル	CYDEV_PERIPH_BASE
RAM	CYDEV_SRAM_BASE
フラッシュ	CYDEV_FLS_BASE

ステップ 2: TD の割り当て

```
TD_Handle = CyDmaTdAllocate();
```

API 関数の `CyDmaTdAllocate()` は、TD のインスタンスを作成し、その TD へハンドルを返します。他の API は、TD ハンドルを TD のコンフィギュレーションに使用します。複数の TD を作成するには、関数を何回も呼び出します。

ステップ 3: TD コンフィギュレーション

```
Channel_Handle = DMA_DmaInitialize(
    DMA_BYTES_PER_BURST,
    DMA_REQUEST_PER_BURST,
    HI16(Source Address),
    HI16(Destination Address));
```

API 関数の `CyDmaTdSetConfiguration()` は、次のパラメーターを使用し TD をコンフィギュレーションします。

- **TD_Handle:** `CyDmaTdAllocate()` 関数により返されるハンドル。
- **Transfer_Count:** 転送元から転送先に転送する総バイト数です。
- **Next_TD:** TD チェーン内の次の TD のインデックスです。この TD がチェーンの最後になる場合は、`DMA_DISABLE_TD (0xFE)`のマクロを使用し、TD の転送が完了した後に DMA チャンネルを無効にします。

TD_Property: DMA トランザクションのプロパティを設定するために、20 ページの表 15 に示した TD コンフィギュレーションレジスタのフラグを使用します。TD プロパティをコンフィギュレーションするために、すべてのフラグを論理和します。例えば、データ転送中に 4 バイトをスワップするために TD を設定するには、以下を使用します。

```
(TD_SWAP_EN | TD_SWAP_SIZE4)
```

表 15. TD プロパティ

コンフィギュレーション フラグ	機能
TD_SWAP_EN	エンディアン スワップを行う。転送元から転送先ヘデータを転送しながらバイトをスワップ
TD_SWAP_SIZE4	スワップ サイズを 4 バイトに設定。デフォルトのスワップ サイズは 2 バイト
TD_AUTO_EXEC_NEXT	現在の TD が完了すると、チェーンの次の TD が自動的にアクティブになる
TD_TERMIN_EN	trq 入力ラインでポジティブ エッジが発生すると、この TD を中止。そのポジティブ エッジはバースト中に発生しなければならない。DMAC がポジティブ エッジを待機するのはこの時だけ
DMA__TD_TERMOUT_EN	このフラグを使用すれば、TD 転送が完了するとパルスが nrq ライン上で発生する。このフラグは、DMA コンポーネント インスタンスに固有のもので、コンポーネントのインスタンス ヘッダ ファイルで定義される。例えば、トップ デザインでは DMA コンポーネント インスタンス名が DMA_1 の場合、インスタンスの TERMOUT マクロは、「DMA_1_dma.h」に含まれる「DMA_1__TD_TERMOUT_EN」
TD_INC_DST_ADR	各バースト データトランザクションのサイズに応じて宛先アドレスをインクリメント
TD_INC_SRC_ADR	各バースト データトランザクションのサイズに応じてソース アドレスをインクリメント

ステップ 4:

TD の転送元と転送先のコンフィギュレーション

```
CyDmaTdSetAddress (TD_Handle,
                    LO16(source),
                    LO16(destination))
```

API 関数の CyDmaTdSetAddress() は、次のパラメーターを使用して、TD のソース アドレスと宛先アドレスを設定します。

- **TD_Handle:** CyDmaTdAllocate() 関数により返されたハンドル
- **LO16(source):** ソース アドレスの下位 16 ビット
- **LO16(destination):** 宛先アドレスの下位 16 ビット

PSoC は、高度にプログラム可能です。多くのコンポーネントは、プログラム可能なデジタルとアナログ ブロックから作成され、ペリフェラルの物理位置はデザインに応じて変更されます。従って、すべてのソース アドレスと宛先アドレスをリストする従来のレジスタ マップを作ることは不可能です。

その代わりに、各コンポーネントのレジスタは、ビルドのプロセス中に PSoC Creator により生成されたコンポーネント API ヘッダ ファイルにて定義されています。コンポーネントのレジスタ アドレスを識別するために、これらのヘッダ ファイルを確認する必要があります。

ステップ 5: チャンネルに TD を接続

```
CyDmaChSetInitialTd (Channel_Handle,
                     TD_Handle)
```

API 関数の CyDmaChSetInitialTD() は DMA チャンネルの最初の TD を設定します。

- **Channel_Handle:** DMA_DmaInitialize() 関数により返される DMA インスタンスのハンドル
- **TD_Handle:** CyDmaTdAllocate() 関数により返されたハンドル

ステップ 6: DMA チャンネルの有効化

```
CyDmaChEnable (Channel_Handle,
               Preserve_TD)
```

API 関数の CyDmaChEnable() は DMA チャンネルを有効にします。

- **Channel_Handle:** DMA_DmaInitialize()関数により返される DMA インスタンスのハンドル
- **Preserve_TD:** TRUE の場合、DMA チャンネルは TD を繰り返すことができるように、TD コンフィギュレーション (転送元、転送先および転送カウント) を保存

その他の重要な DMA API 関数

CPU からの要求で DMA チャンネルを有効にするためには、以下の関数を使用します。

```
CyDmaChSetRequest (Channel_Handle, CPU_REQ);
```

DMA チャンネルを無効にするには、以下の関数を使用します。

```
CyDmaChDisable (Channel_Handle);
```

付録 B: DMA ウィザードによるコンフィギュレーション

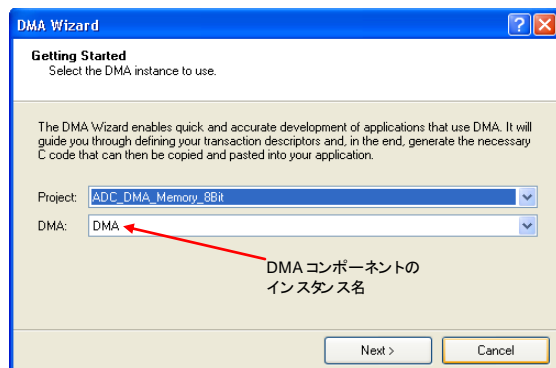
付録 A で説明した手順の代わりに、DMA ウィザードにより、簡単に DMA チャンネルと TD ファームウェアのコンフィギュレーションを定義することができます。ただし、ウィザードは、DMA 転送元または転送先として機能しているペリフェラルを少数のみサポートしています。ペリフェラルがサポートされない場合、付録 A で説明されているコンフィギュレーション手順に従ってください。

DMA ウィザードを起動するには、**PSoC Creator > Tools > DMA Wizard** をクリックしてください。

ステップ 1: DMA チャンネル (DMA コンポーネント インスタンス) を選択

図 15 に示すようにコンフィギュレーションする DMA チャンネルを選択します。

図 15. DMA チャンネルの選択



次のようにダイアログのパラメーターを選択します。

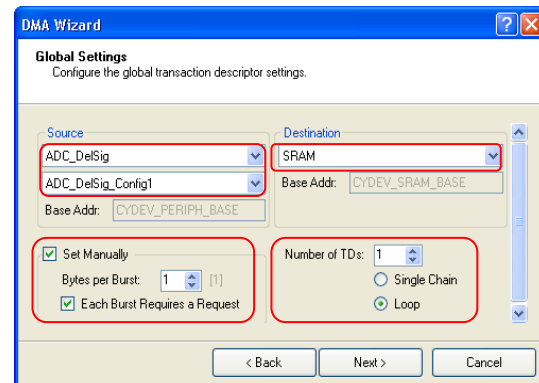
- **Project (プロジェクト):** PSoC Creator プロジェクトの名前
- **DMA:** プロジェクト内の DMA コンポーネント インスタンスの名前

完了したら、「Next」をクリックします。

ステップ 2: グローバル設定を選択

図 16 に示すように、DMA 転送グローバル設定を選択します。

図 16. グローバル設定



このダイアログを使用して DMA チャンネル コンフィギュレーションパラメーターを選択します。

Source (転送元) と Destination (転送先): ソースアドレスと宛先アドレスの上位 16 ビットです。

Bytes per Burst (バースト当たりのバイト数): 単一のバーストで転送するバイト数を決めます。

Each Burst Requires a Request (各バーストごとに要求が必要): 各バーストには個別の要求が必要かどうか決めます。

Number of TDs (TD 数): DMA チャンネルに関連するトランザクション ディスクリプタの数 (1~128) を選択します。

Single Chain or Loop (シングル チェーンまたはループ): どの「次の TD」がチェーンの最後の TD になるかを決めます。シングル チェーンの場合、次の TD は DMA_DISABLE_TD (0xFE) です。ループの場合、次の TD は最初の TD です。

完了したら、「Next」をクリックします。

ステップ 3: チャンネル用のトランザクション ディスクリプタの定義

図 17 に示すように、DMA 転送グローバル設定を選択します。表 16 は、各 TD コンフィギュレーション パラメーターについて説明します。

図 17. トランザクション ディスクリプタの追加

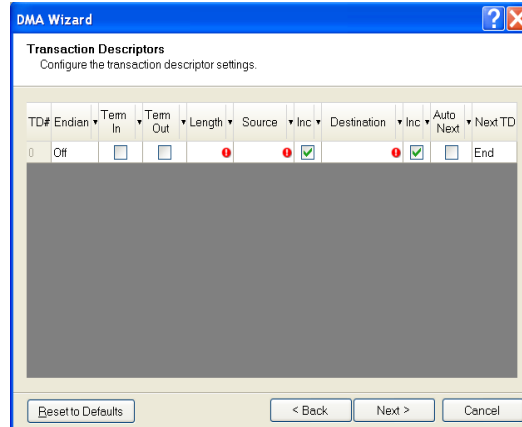


表 16. TD コンフィギュレーションの詳細

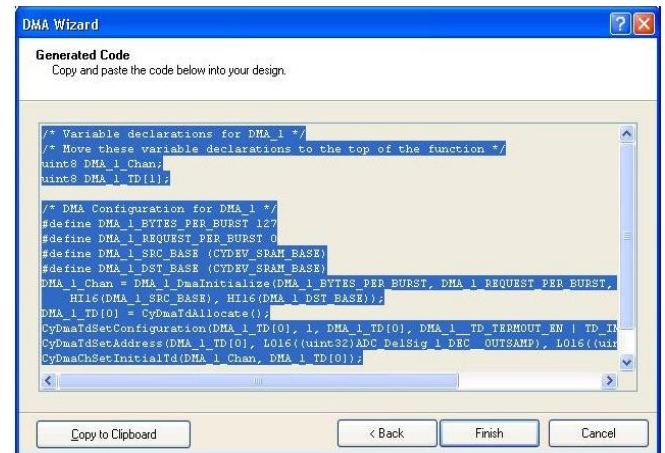
フィールド	説明
TD#	トランザクション ディスクリプタの論理的番号を表示
Endian	2 または 4 バイトエンディアンの バイト スワッピングを有効にする。これにより、データが転送元から転送先へ移動中にバイトをスワップすることが可能になる。 Bytes per Burst (バースト当たりのバイト数) の設定は選択したエンディアンの倍数に設定されていなければならない。これは通常、エンディアンの違いのため PSoC 3 のメモリとペリフェラル間の DMA 転送に使用される
Term In	TERMIN (trq) 信号の立ち上がりエッジで TD トランザクションを中止することを可能にする
Term Out	TD が終了すると TERMOUT (nrq) 信号を生成することを可能にする
Length	TD の転送カウントを バイト単位 (0~4095) で指定する。これは、トランザクションを完了するために DMA が転送すべき総バイト数
Source	DMA 転送用ソース アドレスの下位 16 ビット。選択した転送元が (メモリではなく) コンポーネントの場合、ソース アドレスのドロップダウン リストが DMA ウィザードで提供される。また、手動でソース アドレスを変更したり入力することも可能
Inc (Source)	DMA がトランザクションを行っている間にソース アドレスをインクリメントすることを可能にする。これが有効になっている場合、DMA が転送元からデータを読み出すたびに、ソース アドレスは DMA が読み出したバイト数でインクリメントされる。トランザクション全体 (転送カウント) が終了するまで、DMA はソース アドレスをインクリメントする
Destination	DMA 転送の転送先アドレスの下位 16 ビット。選択した転送先が (メモリではなく) コンポーネントの場合、宛先アドレスのドロップダウンリストが DMA ウィザードで提供される。また、手動で宛先アドレスを変更したり入力することも可能
Inc(Destination)	DMA がトランザクションを行っている間に宛先アドレスをインクリメントすることを可能にする。トランザクション全体 (転送カウント) が終了するまで、DMA は宛先アドレスをインクリメントする
Auto Next	追加の DMA 要求なしで、自動的に次の TD を実行する
Next TD	TD チェーン内の次の論理的 TD。このチェーンがこの TD で終わる場合、END (終了) に設定

完了したら、「Next」をクリックします。必要なコードが生成されます。

ステップ 4: DMA ウィザードで作成されたコードをコピー

DMA チャンネルと TD のコンフィギュレーションが完了したら、ウィザードが DMA チャンネルのためにコードを生成します。このコードは、DMA チャンネルと TD のコンフィギュレーションを含み、[図 18](#) に示すように、DMA ウィザード ダイアログ内のウィンドウで生成されます。コードを使用するには、ウィンドウ内のすべてのコードを選択し、コピーし、*main.c* に貼り付けます。

図 18. 生成コード

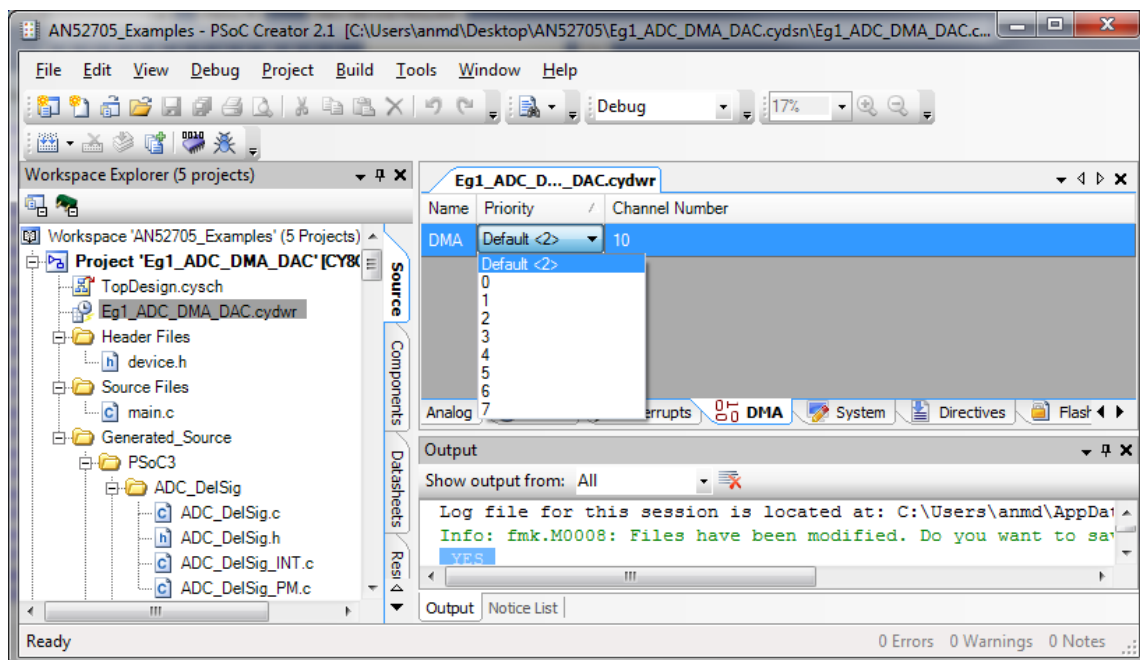


ウィザードの詳細については、「PSoC Creator Help」ファイルを参照してください。

付録 C: DMA チャンネルの優先順位の設定

複数の DMA チャンネル要求がアクティブになった場合、DMA チャンネルは、チャンネル優先順位設定に応じて、DMAC により処理されます。各 DMA チャンネルには、8 つの異なる優先レベルの中から 1 つを与えることができます。[図 19](#) に示すように、**Design Wide Resources (*.cydwr)** > **DMA** タブをクリックして PSoC Creator で DMA チャンネルの優先順位を設定することができます。

図 19. DMA チャンネルの優先順位の設定



CPU と DMAC の要求が同時に PHUB 上の同じスプークにアクセスした場合、CPU はデフォルトで優先されます。PHUB は、DMA と CPU 間、および DMA チャンネル間のアービトレーションを管理します。詳細については、「**PSoC[®] 3、PSoC[®] 5LP アーキテクチャ TRM**」を参照してください。

付録 D: サンプル プロジェクト – テスト セットアップ

例 1: ペリフェラル間の転送 – Eg1_ADC_DMA_DAC

このサンプル プロジェクトでは、ADC のサンプリング周波数 (fs) は 384kHz です。デルタ シグマ ADC は 0.22 fs で-3dB 降下のローパス特性を持っているため、入力周波数が 84kHz 以下の場合、出力が最適に再構成されます。テスト セットアップを次に示します。

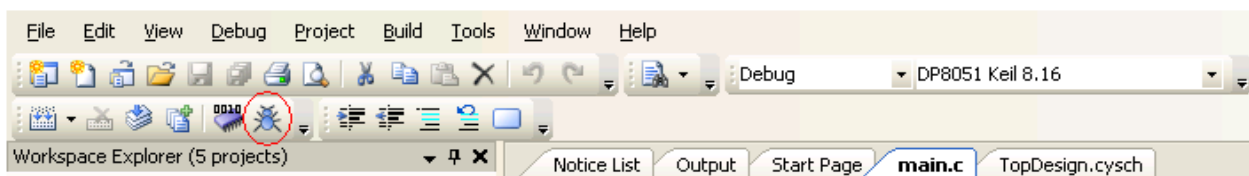
1. ファンクションジェネレータを P0[2]ピン (ADC の入力) に接続します。
2. 100Hz の正弦波を生成するようにファンクションジェネレータを設定します。
3. オシロスコープ プローブを P0[0]ピン (VDAC の出力) に接続します。
4. プロジェクトをビルドし、デバイスをプログラムします。
5. オシロスコープ上の P0[0]ピンからの出力を見てください。入力と同様に、周波数 100Hz の正弦波が生成されているはずです。

例 2: ペリフェラルからメモリへの転送 – Eg2_ADC_DMA_Mem

テスト セットアップを次に示します。

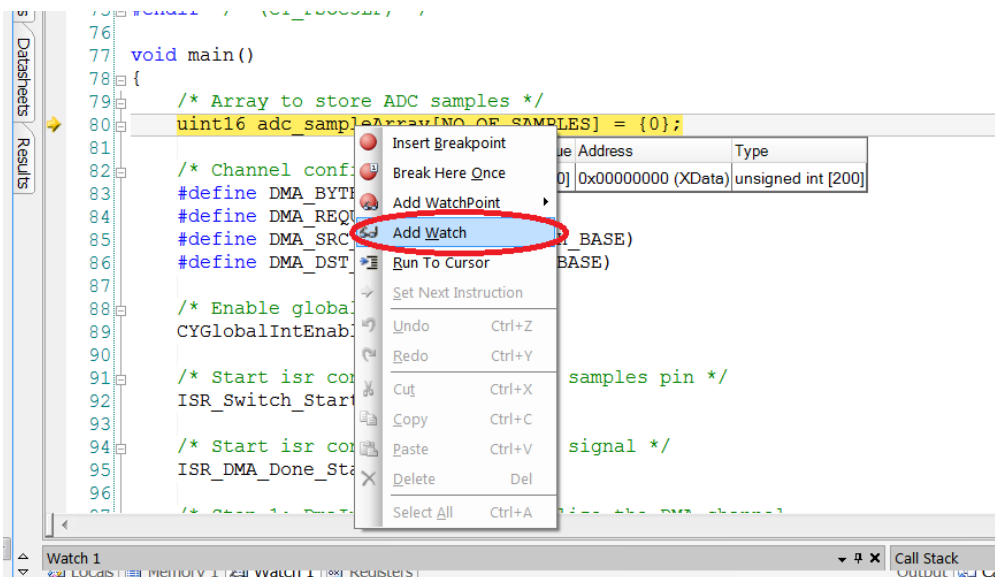
1. 入力信号を P0[2]ピン (ADC の入力) に接続します。入力が ADC の範囲内 ($V_{SSA} \sim 2.048$ V) であることを確認してください。
2. DVK 上のスイッチ (SW1) を P6[1]に接続します。
3. プロジェクトをビルドします。
4. 図 20 に示すように、プログラムをダウンロードし、デバッグを開始するために、F5 キーを押す、またはデバッグのアイコンをクリックします。

図 20. デバッグ ボタン



5. 図 21 に示すように、ウォッチ変数として adc_sampleArray を追加します。

図 21. ウォッチ変数



6. 図 22 に示すように、if (DMA_Done_flag) ループ内にブレークポイントを設定します。

図 22. ブレークポイントの追加

```

154 |
155 |     /* If statement ends here */
156 |
157 |     /* DMA_Done_flag is set inside ISR_DMA_Done after the
158 |      * of ADC samples are buffered */
159 |     if (DMA_Done_flag)
160 |     {
161 |         /* Put a break point here to view the data in the
162 |          * DMA_Done_flag = 0;
163 |     }
164 |     /* If statement ends here */
165 |
166 | } /* for loop ends here */
167 |
168 | /* Place your application code here. */
169 | }

```

7. F5 キーを押してプログラムを実行します。P6[1]に接続されたスイッチ (SW1) を押して、DMA を有効にし、ADC サンプルのバッファリングを開始します。DMA が指定されたサンプル数を ADC からメモリに転送した後、実行はブレークポイントで停止します。結果は、図 23 に示すように、ウォッチ ウィンドウで adc_sampleArray を監視することにより確認することができます。

図 23. ウォッチ ウィンドウでの ADC サンプル

Watch 1				
Name	Value	Address	Type	Radix
adc_sampleArray	[200]	0x00000000 (XData)	unsigned int [200]	Default
0	0x7FFE	0x00000000 (XData)	unsigned int	Default
1	0x7FFD	0x00000002 (XData)	unsigned int	Default
2	0x7FFE	0x00000004 (XData)	unsigned int	Default
3	0x7FFF	0x00000006 (XData)	unsigned int	Default
4	0x7FFC	0x00000008 (XData)	unsigned int	Default
5	0x7FFD	0x0000000A (XData)	unsigned int	Default
6	0x7FFE	0x0000000C (XData)	unsigned int	Default

例 3: メモリからペリフェラルへの転送 – Eg3_Mem_DMA_DAC

テスト セットアップを次に示します。

- オシロスコープ プローブを P0[0]ピン (VDAC の出力) に接続します。
- プロジェクトをビルドして、デバイスをプログラムします。
- オシロスコープで周波数 7.8kHz の正弦波を観察します。

例 4: メモリ間の転送 – Eg4_Mem_DMA_Mem

テスト セットアップを次に示します。

- キャラクタ LCD モジュールを CY8CKIT-001 PSoC の開発キットの P18 ヘッダ (LCD モジュールのポート 2) に接続します。
- J12 ジャンパが、LCD に電源を供給するために ON の位置にあることを確認します。
- プロジェクトをビルドして、デバイスをプログラムします。
- LCD ディスプレイを見ます。最初の行は、宛先アレイの内容を表示します。最初はすべての値がゼロです。1 秒の遅延の後、最初の行は、00 ~ 07 を表示し、DMA がフラッシュから RAM にデータを正常に転送したことを示します。2 行目には、「TRANSFERRED」(転送完了) メッセージが表示されます。図 24 は LCD ディスプレイの例を示します。

図 24. DMA 転送の LCD ディスプレイ

0	0	0	1	0	2	0	3	0	4	0	5	0	6	0	7
T	R	A	N	S	F	E	R	R	E	D					

例 5: TD のチェーン化 – Eg5_TD_Chaining

このテスト セットアップは、例2のテスト セットアップと同様です。テストのセットアップ手順は次の通りです。

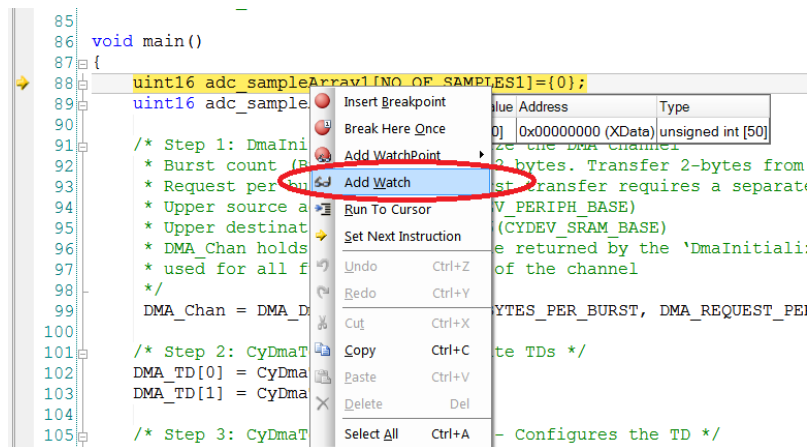
1. 入力信号を P0[2]ピン (ADC の入力)に接続します。入力が ADC の範囲内 ($V_{SSA} \sim 2.048V$) であることを確認してください。
2. DVK 上のスイッチ (SW1) を P6[1]に接続します。
3. プロジェクトをビルドします。
4. 図 25 に示すように、プログラムをダウンロードし、デバッグを開始するために、F5 キーを押す、またはデバッグのアイコンをクリックします。

図 25. デバッグ ボタン



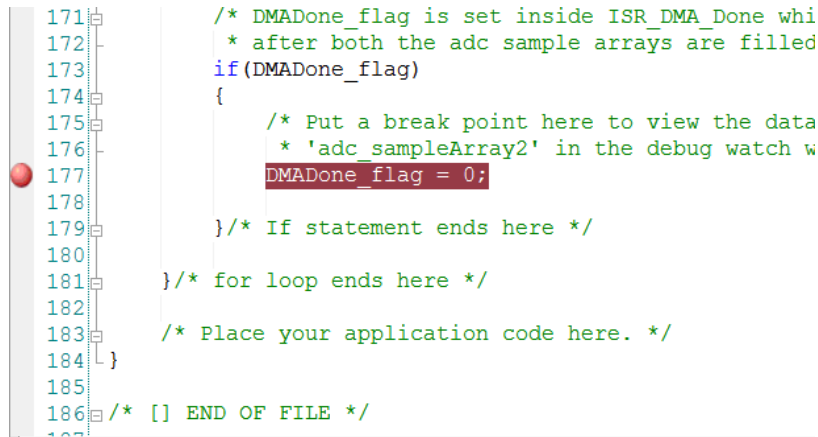
5. 図 26 に示すように、ウォッチ変数として `adc_samplearray1` と `adc_samplearray2` を追加します。

図 26. ウォッチ変数



6. 図 27 に示すように、`if (DMADone_flag)` ループ内にブレークポイントを設定します。

図 27. ブレークポイントの追加



7. F5 キーを押してプログラムを実行します。P6[1]に接続されたスイッチ (SW1) を押して、DMA を有効にし、ADC サンプルのバッファリングを開始します。DMA が指定されたサンプル数を ADC からメモリに転送した後、実行はブレークポイントで停止します。結果を検証するには、図 28 に示すようにウォッチ ウィンドウで adc_sampleArray1 と adc_sampleArray2 を監視します。

図 28. ウォッチ ウィンドウでの ADC サンプル

Watch 1			Watch 1		
Unavailable, target is running			Name	Value	
adc_sampleArray1 [50]			adc_sampleArray2 [20]		
0		0x7FEF	0		0x7FF5
1		0x7FF8	1		0x7FF5
2		0x7FF3	2		0x7FF6
3		0x7FF5	3		0x7FF6
4		0x7FF6	4		0x7FF9
5		0x7FF6	5		0x7FF7
6		0x7FF6	6		0x7FF5
7		0x7FF7	7		0x7FF7
8		0x7FF6	8		0x7FF6
9		0x7FF8	9		0x7FF7
10		0x7FF9	10		0x7FF7
11		0x7FF7	11		0x7FF7
12		0x7FF8	12		0x7FF5
13		0x7FF6	13		0x7FF4
14		0x7FF5	14		0x7FF6
15		0x7FF6	15		0x7FF8
16		0x7FF6	16		0x7FF6

付録 E: よくある質問

1. DMA を使用してどのように、4095 バイト以上をバッファリングできますか？
TD の最大転送カウントは 4095 バイトに制限されます。1 つの DMA チャンネルを使って 4095 バイト以上を転送する必要がある場合、例 5 に示すように、複数の TD を使用し、それらを連鎖します。
2. どのようにして、DMA データ転送のためのペリフェラルのソース アドレスと宛先アドレスを確定できますか？
PSoC は、高度にプログラム可能です。多くのコンポーネントは、プログラム可能なデジタルとアナログ ブロックから作成され、ペリフェラルの物理位置はデザインに応じて変更されます。従って、すべてのソース アドレスと宛先アドレスをリストする従来のレジスタマップを作ることは不可能です。
その代わりに、各コンポーネントのレジスタは、ビルドのプロセス中に PSoC Creator により生成されたコンポーネント API ヘッダ ファイルにて定義されています。コンポーネントのレジスタ アドレスを識別するために、これらのヘッダ ファイルを確認する必要があります。
3. どのようにして、DMA を UART、SPI などの通信プロトコルと併用しますか？
UART や SPI などの通信プロトコルを DMA と共に使用する場合は、データ転送のために内部割り込みがトリガーされないように、バッファ サイズを 4 以下に設定します。ハードウェアの FIFO ポインタを DMA の読み出しと書き込みデータ アドレスとして使用し、FIFO レベルのステータスを割り込みとしてコンフィギュレーションして DMA をトリガーします。DMA チャンネルのハードウェア要求を、FIFO レベルで使用できるように、レベルトリガーに設定します。
4. DMA 転送のタイミングの詳細を教えてください。
DMA 転送のタイミングの詳細は「[PSoC 3, PSoC 5LP テクニカル リファレンス マニュアル](#)」を参照してください。DMA のタイミングに関する詳細説明は、このアプリケーション ノートの範囲外です。

改訂履歴

文書名: AN52705 – PSoC® 3 および PSoC 5LP - DMA 入門

文書番号: 001-75652

版	ECN 番号	変更者	発行日	変更内容
**	3497899	NITA	01/20/2012	これは英語版 001-52705 を翻訳した日本語版 001-75652 Rev. **です。
*A	4716723	HZEN	04/02/2015	これは英語版 001-52705 Rev. *I を翻訳した日本語版 001-75652 Rev. *A です。
*B	5815462	AESATMP9	07/19/2017	ロゴと著作権を更新しました。

ワールドワイドな販売と設計サポート

サイプレスは、事業所、ソリューション センター、メーカー代理店および販売代理店の世界的なネットワークを保持しています。お客様の最寄りのオフィスについては、[サイプレスのロケーション](#) ページをご覧ください。

製品

ARM® Cortex® Microcontrollers	cypress.com/arm
車載用	cypress.com/automotive
クロック&バッファ	cypress.com/clocks
インターフェース	cypress.com/interface
IoT (モノのインターネット)	cypress.com/iot
メモリ	cypress.com/memory
マイクロコントローラ	cypress.com/mcu
PSoC	cypress.com/psoc
電源用 IC	cypress.com/pm/c
タッチ センシング	cypress.com/touch
USB コントローラ	cypress.com/usb
ワイヤレス	cypress.com/wireless

PSoC®ソリューション

[PSoC 1](#) | [PSoC 3](#) | [PSoC 5LP](#) | [PSoC 6](#)

サイプレス開発者コミュニティ

[フォーラム](#) | [WICED IOT Forums](#) | [Projects](#) | [ビデオ](#) | [ブログ](#) | [トレーニング](#) | [Components](#)

テクニカル サポート

cypress.com/go/support

本書で言及するその他すべての商標または登録商標は、それぞれの所有者に帰属します。



Cypress Semiconductor
198 Champion Court
San Jose, CA 95134-1709

©Cypress Semiconductor Corporation, 2009-2017. 本書面は、Cypress Semiconductor Corporation 及び Spansion LLC を含むその子会社（以下「Cypress」という。）に帰属する財産である。本書面（本書面に含まれ又は言及されているあらゆるソフトウェア若しくはファームウェア（以下「本ソフトウェア」という。）を含む）は、アメリカ合衆国及び世界のその他の国における知的財産法令及び条約に基づき Cypress が所有する。Cypress はこれらの法令及び条約に基づく全ての権利を留保し、本段落で特に記載されているものを除き、その特許権、著作権、商標権又はその他の知的財産権のライセンスを一切許諾しない。本ソフトウェアにライセンス契約書が伴っておらず、かつ Cypress との間で別途本ソフトウェアの使用方法を定める書面による合意がない場合、Cypress は、(1) 本ソフトウェアの著作権に基づき、(a) ソースコード形式で提供されている本ソフトウェアについて、Cypress ハードウェア製品と共に用いるためにのみ、かつ組織内部でのみ、本ソフトウェアの修正及び複製を行うこと、並びに (b) Cypress のハードウェア製品ユニットに用いるためにのみ、（直接又は再販売者及び販売代理店を介して間接のいずれかで）本ソフトウェアをバイナリコード形式で外部エンドユーザーに配布すること、並びに (2) 本ソフトウェア（Cypress により提供され、修正がなされていないもの）が抵触する Cypress の特許権のクレームに基づき、Cypress ハードウェア製品と共に用いるためにのみ、本ソフトウェアの作成、利用、配布及び輸入を行うことについての非独占的で譲渡不能な一身専属的ライセンス（サブライセンスの権利を除く）を付与する。本ソフトウェアのその他の使用、複製、修正、変換又はコンパイルを禁止する。

適用される法律により許される範囲内で、Cypress は、本書面又はいかなる本ソフトウェア若しくはこれに伴うハードウェアに関しても、明示又は黙示をとわず、いかなる保証（商品性及び特定の目的への適合性の黙示の保証を含むがこれらに限られない）も行わない。いかなるコンピューティングデバイスも絶対に安全ということはない。従って、Cypress のハードウェアまたはソフトウェア製品に講じられたセキュリティ対策にもかかわらず、Cypress は、Cypress 製品への権限のないアクセスまたは使用といったセキュリティ違反から生じる一切の責任を負わない。加えて、本書面に記載された製品には、エラーと呼ばれる設計上の欠陥またはエラーが含まれている可能性があり、公表された仕様とは異なる動作をする場合がある。適用される法律により許される範囲内で、Cypress は、別途通知することなく、本書面を変更する権利を留保する。Cypress は、本書面に記載のあるいかなる製品若しくは回路の適用又は使用から生じる一切の責任を負わない。本書面で提供されたあらゆる情報（あらゆるサンプルデザイン情報又はプログラムコードを含む）は、参照目的のためのみに提供されたものである。この情報で構成するあらゆるアプリケーション及びその結果としてのあらゆる製品の機能性及び安全性を適切に設計、プログラム、かつテストすることは、本書面のユーザーの責任において行われるものとする。Cypress 製品は、兵器、兵器システム、原子力施設、生命維持装置若しくは生命維持システム、蘇生用の設備及び外科的移植を含むその他の医療機器若しくは医療システム、汚染管理若しくは有害物質管理の運用のために設計され若しくは意図されたシステムの重要な構成部分としての使用、又は装置若しくはシステムの不具合が人身傷害、死亡若しくは物的損害を生じさせるようなその他の使用（以下「本目的外使用」という。）のために設計、意図又は承認されていない。重要な構成部分とは、その不具合が装置若しくはシステムの不具合を生じさせるか又はその安全性若しくは実効性に影響すると合理的に予想できるような装置若しくはシステムのあらゆる構成部分という。Cypress 製品のあらゆる本目的外使用から生じ、若しくは本目的外使用に関連するいかなる請求、損害又はその他の責任についても、Cypress はその全部又は一部をとわず一切の責任を負わず、かつ Cypress はそれら一切から本書により免除される。Cypress は Cypress 製品の目的外使用から生じ又は本目的外使用に関連するあらゆる請求、費用、損害及びその他の責任（人身傷害又は死亡に基づく請求を含む）から免責補償される。

Cypress, Cypress のロゴ, Spansion, Spansion のロゴ及びこれらの組み合わせ, WICED, PSoC, Capsense, EZ-USB, F-RAM, 及び Traveo は、米国及びその他の国における Cypress の商標又は登録商標である。Cypress のより完全な商標のリストは、cypress.com を参照すること。その他の名称及びブランドは、それぞれの権利者の財産として権利主張がなされている可能性がある。