TRAVEO™ T2G ファミリの Direct Memory Access (DMA) コントローラの使用方法

本書について

適用範囲と目的

このアプリケーションノートは、TRAVEO™ T2G ファミリ MCU で DMA コントローラ (P-DMA および M-DMA) を使用する方法について説明します。DMA コントローラは、CPU の介在なしにデータを転送できます。このアプリケーションノートでは、ペリフェラルからメモリ, メモリからペリフェラル, およびメモリからメモリへのデータ転送のための DMA の設定方法を示します。

関連製品ファミリ

TRAVEO™ T2G ファミリ CYT2/CYT3/CYT4 シリーズ。

対象者

本書は、TRAVEO™ T2G ファミリ CYT2/3/4 シリーズを利用されている方を対象とします。

目次

目次

1 はじめに

このアプリケーションノートでは、TRAVEO™ T2G ファミリ MCU の Direct Memory Access (DMA)コントローラの使用方法について説明します。

DMA コントローラは、メモリと内蔵ペリフェラル間、またはメモリ間で CPU の介在なしでデータをシームレスに転送できます。これにより、DMA コントローラがデータを転送している間に CPU が他のタスクを処理できます。

P-DMA と M-DMA の両方には、複数の独立した DMA チャネルがあります。各 DMA チャネルは、トランザクションを開始する独立した DMA 要求入力を持ち、独立してデータを転送できます。各デバイスで使用可能な DMA チャネルの数については、デバイスデータシートを参照してください。

本シリーズは、ペリフェラル DMA (P-DMA) とメモリ DMA (M-DMA) の 2 種類の DMA コントローラをサポートします。P-DMA は、多数のチャネルに対してペリフェラルからメモリへ, メモリからペリフェラルへ, の低遅延なデータ転送に使用されます。MDMA は、少数のチャネルに対して、メモリからメモリへの高い帯域幅のデータ転送に使用されます。

これらの DMA コントローラは転送動作を指定するディスクリプタを持ち、様々なアプリケーションに柔軟に対応します。ディスクリプタは連結および循環リストを構成できます。

このアプリケーションノートでは、DMA コントローラの機能, 初期設定, およびユースケースによるデータ転送動作について説明します。

このアプリケーションノートで使用する機能と用語については、Architecture technical reference manual (TRM)の「Direct Memory Access」の章を参照してください。

1.1 特長

表 1 では P-DMA と M-DMA を比較します。これらは類似のレジスタとディスクリプタの構造を持ちます。

表 1 P-DMA/M-DMA の特長

特長	P-DMA	M-DMA
機能の焦点	低レイテンシ	高メモリ帯域幅
有用な機能	ペリフェラルとメモリ間の転送	メモリ間の転送
転送エンジン	すべてのチャネルで共有	各チャネル専用
転送サイズ	8 ビット, 16 ビット, 32 ビット	8 ビット, 16 ビット, 32 ビット
チャネル優先度	• 4 レベル • プリエンプタブル	4 レベル
ディスクリプタタイプ	• Single transfer • 1D / 2D transfer • CRC transfer	• Single transfer • 1D / 2D transfer • Memory copy • Scatter
ディスクリプタ構成	• Source/Destination address • Transfer size • Descriptor type • Trigger-in type (4 タイプ) • Trigger-out type (4 タイプ) • Interrupt type (4 タイプ) • ディスクリプタチェイン	• Source/Destination address • Transfer size • Descriptor type • Trigger-in type (4 タイプ) • Trigger-out type (4 タイプ) • Interrupt type (4 タイプ) • ディスクリプタチェイン

(続く)

表 1 (続き) P-DMA/M-DMA の特長

特長	P-DMA	M-DMA
トリガ入力	• Hardware trigger • Software trigger • Trigger output (tr_out)	• Software trigger • Trigger output (tr_out)

P-DMA はメモリ間の転送にも使用できますが、転送帯域幅は M-DMA と比較して十分ではありません。M-DMA はメモリとペリフェラル間の転送にも使用できますが、P-DMA と比較して転送レイテンシは低くない場合があります。

P-DMA では、Single transfer の間により高い優先度の保留中のチャネルを転送できます (プリエンプタブル機能)。M-DMA は、メモリ帯域幅全体を低下させるため、プリエンプタブル機能はありません。

ディスクリプタは、DMA 転送の仕様を決定します。ディスクリプタタイプによって、DMA 転送動作のタイプが決まります。どちらの DMA もディスクリプタタイプとして、Single transfer, 1D transfer, および 2D transfer をサポートします。さらに、P-DMA は CRC transfer をサポートし、M-DMA は Memory copy および Scatter をサポートします。各ディスクリプタタイプの詳細については、ディスクリプタタイプを参照してください。ディスクリプタは、現在のディスクリプタに次のディスクリプタのポインタを格納することで連結できます (ディスクリプタチェイン)。ディスクリプタチェインは、ディスクリプタリストとも呼ばれます。

Hardware trigger, Software trigger, Trigger output (tr_out)などのトリガ入力は、DMA 外の周辺機能である Trigger Multiplexer を介して入力されます。Trigger Multiplexer は、トリガを潜在的なソースからデスティネーションにルーティングします。詳細は、Architecture TRM の「Trigger Multiplexer」の章を参照してください。

P-DMA は Hardware trigger, Software trigger, Trigger output (tr_out) をトリガ入力としてサポートし、M-DMA は Software trigger と Trigger output (tr_out) のみをサポートします。利用可能な Hardware trigger については、デバイスデータシートを参照してください。Software trigger は、Trigger Multiplexer の機能によって実現されます。どちらの DMA も、Trigger output を独自のトリガ入力として使用できます。各トリガ機能については トリガ機能を参照してください。

1.2 ブロックダイヤグラム

図 1 に P-DMA のブロックダイヤグラムを示します。

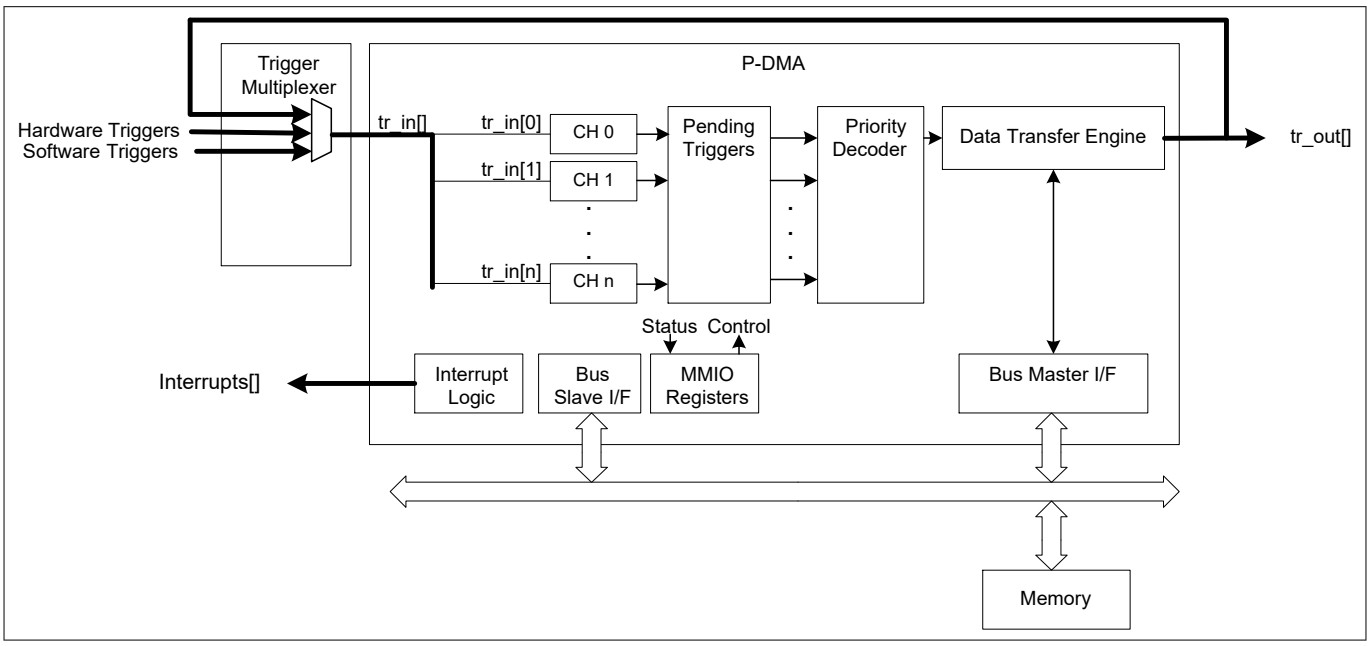


図 1 P-DMA ブロックダイヤグラム M-DMA ブロックダイヤグラム

1 はじめに

P-DMA は、チャネル (CH0 – CHn), Pending Trigger ブロック, Priority Decoder, Data Transfer Engine, および Interrupt logic で構成されます。P-DMA の Transfer Engine は、すべてのチャネルで共有されます。各ブロックの詳細については、Architecture TRM を参照してください。

先に述べたように、P-DMA のトリガ入力は、Hardware trigger, Software trigger, または Trigger output (tr_out) です。これらのトリガは、Trigger Multiplexer を介して入力されます。

Trigger output (tr_out) は、自身のトリガ入力や、他のチャネルの異なる転送のためのトリガ入力として使用できます。

ディスクリプタを格納するために使用するメモリは、DMA ブロックの外にあります。Transfer Engine が次の保留中のチャネルをアクティブにする時、Transfer Engine はそのチャネルに対応するディスクリプタをメモリから読み出し、転送を開始します。

図 2 に M-DMA のブロックダイヤグラムを示します。

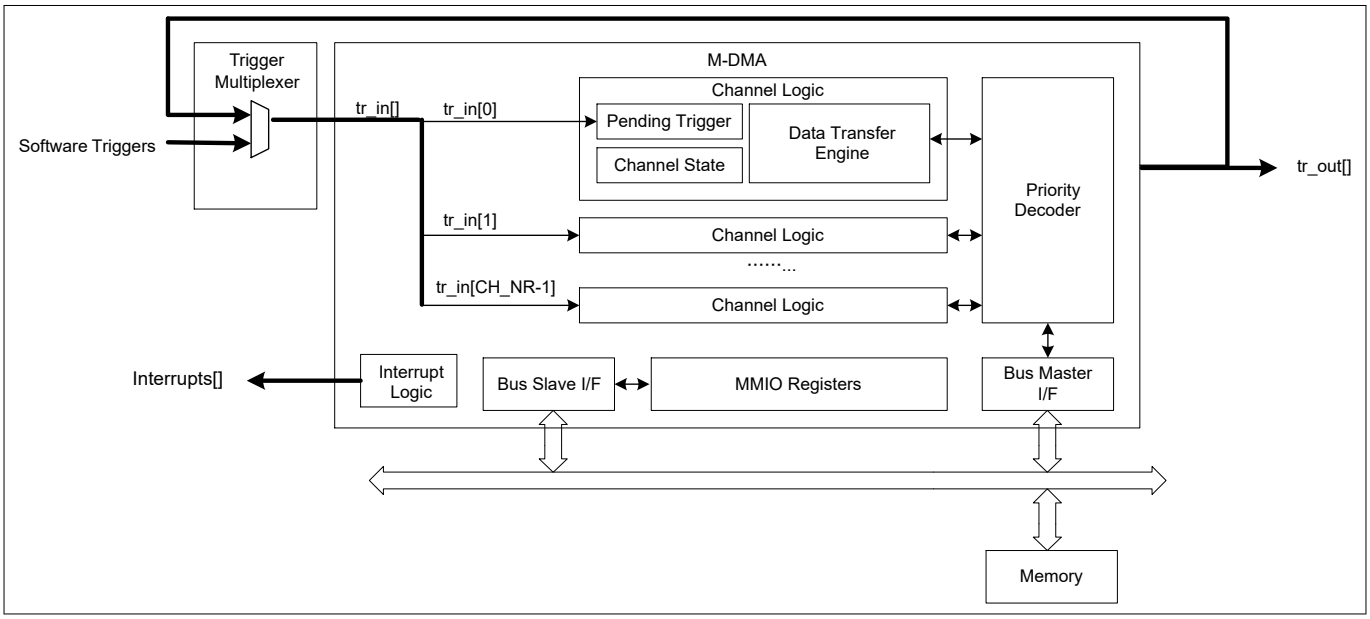


図 2　　M-DMA ブロックダイヤグラム

M-DMA ブロックは、Channel Logic, Priority Decoder, および Register で構成されます。Channel Logic 自体は Pending Trigger を記憶し、現在のチャネル状態および Data Transfer Engine をホストします。M-DMA には、各チャネル専用の Transfer Engine があります。各ブロックの詳細については、Architecture TRM を参照してください。

トリガ入力として、M-DMA は Software trigger と、それ自身の Trigger output (tr_out) をサポートします。これらのトリガは、Trigger Multiplexer を介して入力されます。P-DMA と異なり、M-DMA はハードウェアトリガをサポートしないことに注意してください。

2 操作概要

図 3 に、P-DMA と M-DMA を設定する方法を示します。

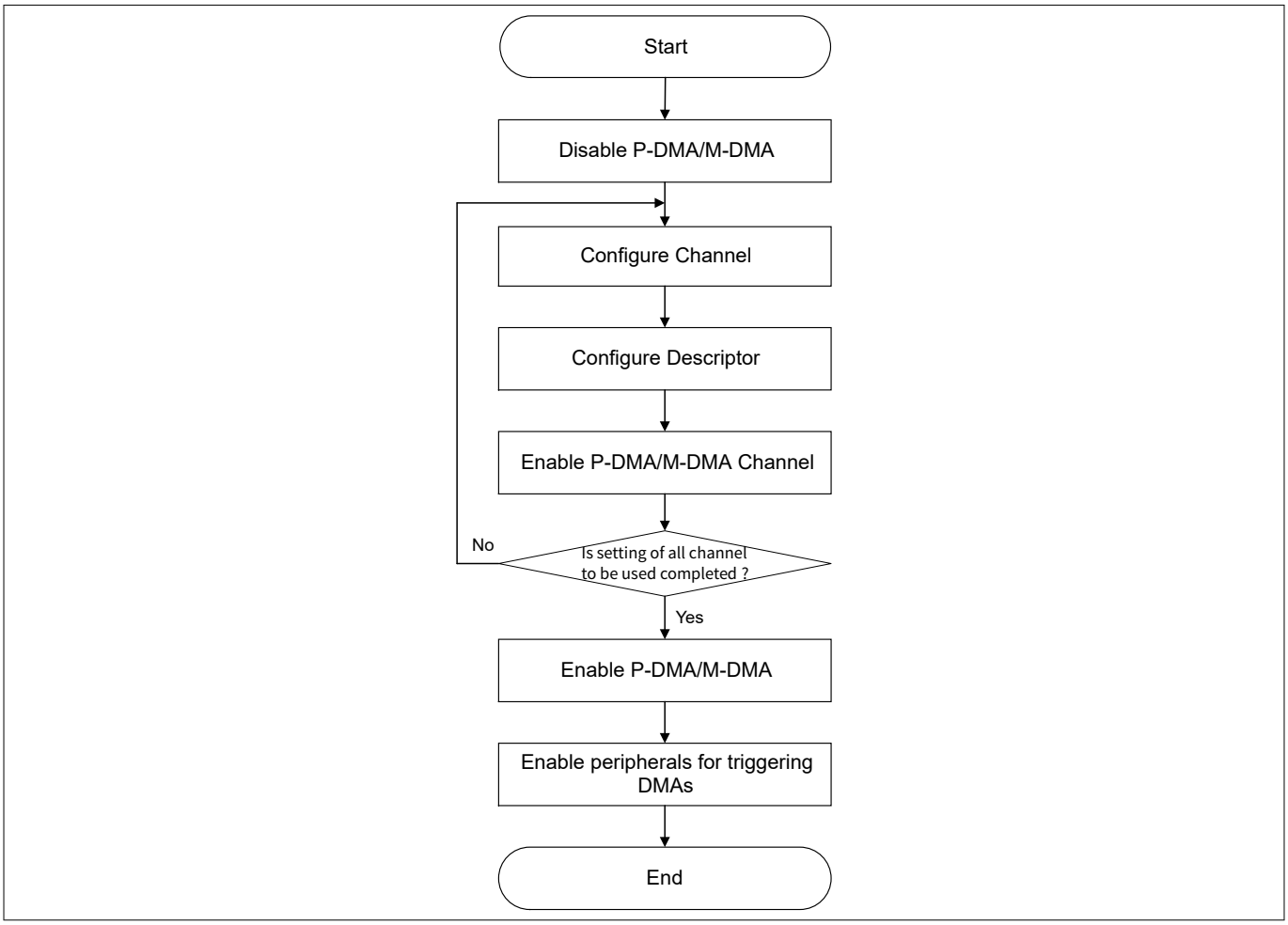


図 3 P-DMA/M-DMA の一般的な設定

この例では、チャネル, ディスクリプタ, およびチャネルイネーブルが各チャネルに対して設定されます。また、各ステップで、使用するすべてのチャネルの設定も可能です。

対応する DMA チャネルの設定後、DMA は周辺からのトリガを待ちます。

2.1 P-DMA/M-DMA のディセーブルとイネーブル

P-DMA と M-DMA は表 2 に示すようにそれぞれのビットを使用してイネーブル/ディセーブルにできます。リセット後のデフォルト設定は‘0’ (ディセーブル)です。

表 2 P-DMA/M-DMA ディセーブル/イネーブル

DMA タイプ	レジスタ (bit)	説明
P-DMA	DW_CTL.ENABLED (bit31)	0: ディセーブル, 1: イネーブル
M-DMA	DMAC_CTL.ENABLED (bit31)	0: ディセーブル, 1: イネーブル

2.2 チャネルの設定

このステップでは、チャネルに対応するディスクリプタのチャネル優先度やポインタアドレスなどの P-DMA/M-DMA チャネルの設定を行います。

さらに P-DMA では必要に応じて、プリエンプタブル機能と CRC transfer のための CRC 計算モードを設定します。表 3 および表 4 に、それぞれ P-DMA および M-DMA のチャネルを設定するために使用するレジスタを示します。チャネル番号に対応するレジスタを設定します。詳細は Architecture TRM および Registers technical reference manual (TRM)を参照してください。

表 3　　チャネルの設定 (P-DMA)

レジスタ (bit)	説明
DW_CH_STRUCT_CH_CURR_PTR	チャネルの現在のディスクリプタポインタを設定します。ソフトウェアはこのレジスタを初期化する必要があります。
DW_CH_STRUCT_CH_CTL.PREEMPTABLE (bit11)	チャネルがプリエンプタブルかどうかを指定します。
DW_CH_STRUCT_CH_CTL.PRIO (bit9:8)	チャネルの優先順位を設定します。
DW_CH_STRUCT_CH_IDX.X_IDX (bit7:0)	チャネルの X インデックスを現在のディスクリプタに設定します。ソフトウェアはこのレジスタを初期化する必要があります。
DW_CH_STRUCT_CH_IDX_Y_IDX (bit15:0)	チャネルの Y インデックスを現在のディスクリプタに設定します。ソフトウェアはこのレジスタを初期化する必要があります。
CRC transfer にのみ設定が必要なレジスタ	
DW_CRC_CTL.DATA_REVERSE (bit0)	データバイトが処理されるビット順 (MSb または LSb ファースト) を指定します。
DW_CRC_CTL.REM_REVERSE (bit8)	剰余をビット反転するかどうかを指定します。
DW_CRC_DATA_CTL.DATA_XOR (bit7:0)	各データバイトが XOR されるバイトマスクを設定します。この 8 ビットの値はランダムに選択できます。
DW_CRC_POL_CTL.POLYNOMIAL	CRC 多項式を設定します。
DW_CRC_LFSR_CTL.LFSR32	CRC 計算のシード値を設定します。
DW_CRC_REM_CTL.REM_XOR	CRC_LFSR_CTL.LFSR32 レジスタが XOR されるマスクを設定します。

表 4　　チャネルの設定 (M-DMA)

レジスタ (bit)	説明
DMAC_CH_CH_CURR_PTR	チャネルの現在のディスクリプタポインタを設定します。ソフトウェアはこのレジスタを初期化する必要があります。
DMAC_CH_CH_CTL.PRIO (bit9:8)	チャネルの優先順位を設定します。

2.3　　ディスクリプタの設定

このステップでは、ディスクリプタを設定します。ディスクリプタは、DMA チャネルの転送の詳細を指定します。ディスクリプタは、DMA の外部のメモリに格納され、Transfer Engine によって読み出されます。Transfer Engine は、ディスクリプタに従ってデータを転送します。各チャネルのディスクリプタポインタの位置は、ディスクリプタポインタレジスタに格納します(チャネルの設定を参照してください)。

図 4 に、P-DMA と M-DMA のディスクリプタ構造を示します。P-DMA のディスクリプタは 6 つの 32 ビット ワードで構成され、M-DMA のディスクリプタは 8 つの 32 ビット ワードで構成されます。しかしながら、両方の DMA のディスクリプタは同様の機能を持ちます。

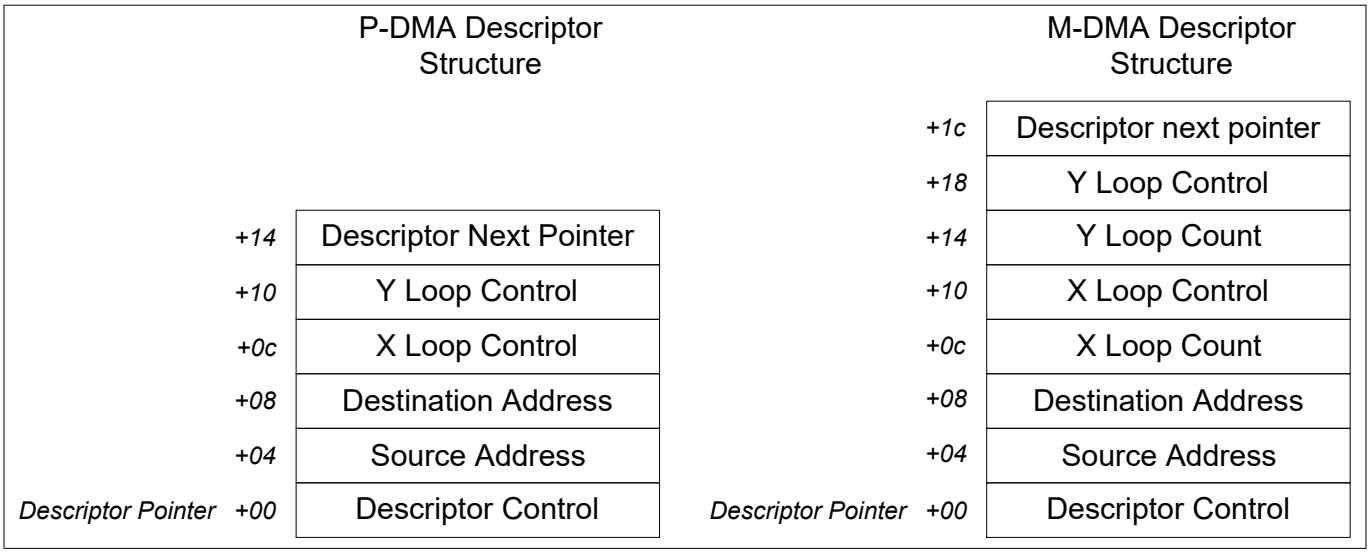


図 4 P-DMA/M-DMA ディスクリプタ構造

ディスクリプタパラメータの設定

- **Descriptor control**: Descriptor type, Transfer size, Trigger-in/out, および Interrupt setting などの DMA パラメータを記述するワードです。
- **Source address, destination address**: 転送元および転送先領域のベースアドレスを指定するワードです。
- **X loop control, X loop count**: 1D transfer のループまたは 2D transfer の内部ループを制御するワードです。X loop control は、各 X ループ反復の転送元アドレスと転送先アドレスの増分を指定します。X loop count は、X ループの反復回数を指定します。
- **Y loop control, Y loop count**: 2D transfer の外側ループを制御するワードです。Y loop control は、各 Y ループ反復の転送元アドレスと転送先アドレスの増分を指定します。Y loop count は、Y ループの反復回数を指定します。
- **Descriptor next pointer**: 次のディスクリプタのアドレスを指定するワードです。ディスクリプタは、現在のディスクリプタに次のディスクリプタのポインタを格納し連結できます。ディスクリプタリストの最後のディスクリプタは、このワードに '0' (NULL) があります。

ディスクリプタの詳細については、Architecture TRM および registers TRM を参照してください。

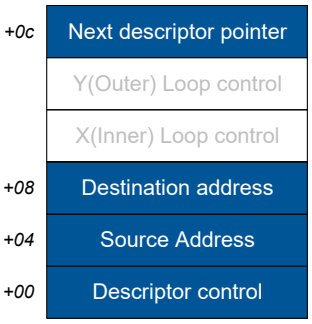
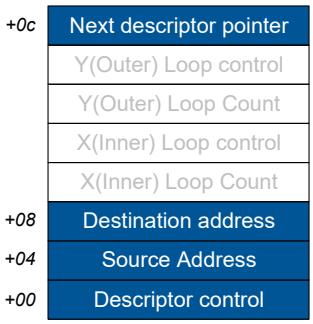
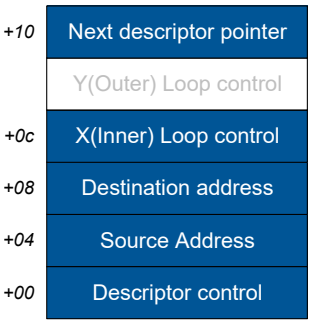
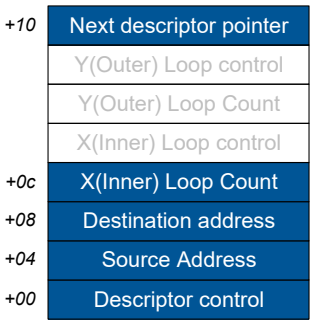
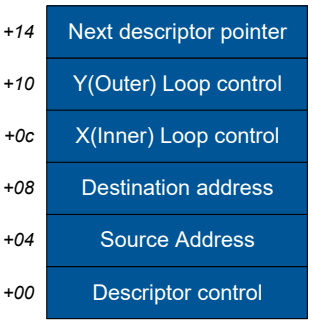
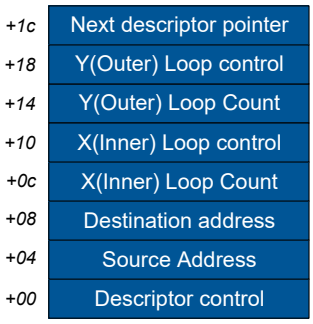
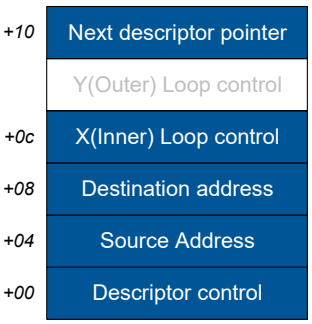
使用されるディスクリプタのワードの数は、ディスクリプタの種類によって異なります。使用されないディスクリプタのワードがある場合、ワードのアドレスは前詰めされます。

2.3.1 ディスクリプタタイプ

ここでは、DMA 転送の種類を決定するディスクリプタタイプについて説明します。

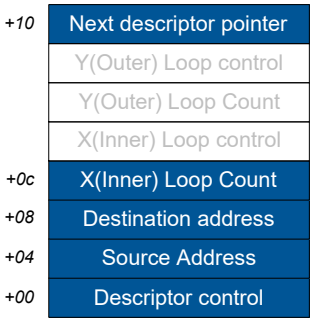
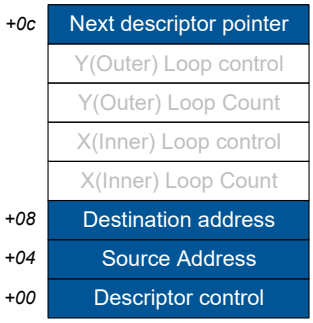
P-DMA には 4 つのディスクリプタタイプがあり、M-DMA には 5 つのディスクリプタタイプがあります。使用されるディスクリプタのワードの数は、ディスクリプタの種類によって異なります。表 5 に、各ディスクリプタタイプを示します。表 5 では、“転送例”の列に各ディスクリプタタイプの概要と疑似コードを示します。“使用するディスクリプタ”の列には、各 DMA のディスクリプタタイプで使用されるディスクリプタのワードを示します。未使用のディスクリプタのワードがある場合、ワードのアドレスは前詰めされることに注意してください。

表 5　各ディスクリプタタイプの転送例と使用するディスクリプタ (P-DMA/M-DMA)

ディスクリプタタイプ	転送例	使用するディスクリプタ	
		P-DMA	M-DMA
Single transfer	**単一のデータ要素を転送**: `DST_ADDR = (DATA_SIZE) SRC_ADDR`		
1D transfer	**1 次元の「for ループ」転送**: `for (X_IDX =0; X_IDX <= COUNT; X_IDX++) { DST_ADDR[DST_INCR] = (DATA_SIZE) SRC_ADDR[SRC_INCR] } 。 *DST_INCR/SRC_INCR は X_INCR に依存する DST_ADDR = (DATA_SIZE) SRC_ADDR`		
2D transfer	**2 次元の「for ループ」転送**: `for (Y_IDX =0; Y_IDX <= Y_COUNT; Y_IDX++) { for (X_IDX =0; X_IDX <= X_COUNT; X_IDX++) { DST_ADDR[DST_INCR] = (DATA_SIZE) SRC_ADDR[SRC_INCR] } } DST_ADDR[DST_INCR] = (DATA_SIZE) SRC_ADDR[SRC_INCR] } } DST_ADDR[DST_INCR] = (DATA_SIZE) SRC_ADDR[SRC_INCR} *DST_INCR/SRC_INCR` は `X/Y_INCR` に依存する。		
CRC transfer	**指定された領域の CRC を計算します。** CRC 転送の場合、CRC はメモリマップ I/O (MMIO) レジスタで設定する必要があることに注意してください。		サポートされません

(続く)

表 5 (続き) 各ディスクリプタタイプの転送例と使用するディスクリプタ (P-DMA/M-DMA)

ディスクリプタタイプ	転送例	使用するディスクリプタ	
		P-DMA	M-DMA
Memory copy	**1 次元の「for ループ」転送**: `for (X_IDX =0; X_IDX <= X_COUNT; X_IDX+) { DST_ADDR[IDX] = SRC_ADDR[IDX] }.`	サポートされません	
Scatter	**アドレスが散在する 32 ビットのデータ要素のセットを書く 1 次元の「for ループ」転送**: `for (X_IDX =0; X_IDX <= X_COUNT; X_IDX +=2) { address = SRC_ADDR[IDX] data = SRC_ADDR[IDX+1] *address = data }.`	サポートされません	

2.3.2 トリガ機能

ここでは、トリガ機能について説明します。Trigger input, Trigger output (tr_out), および Interrupt は、ディスクリプタによって制御されます。

Trigger input は DMA チャネル転送をアクティブにします。転送が完了すると、Trigger output (tr_out), Iinterrupt が出力されます。トリガ操作は、ディスクリプタの TR_IN_TYPE, TR_OUT_TYPE, および INTR_TYPE によって指定されます。Trigger input, Trigger output (tr_out), Interrupt は、チャネルごとに独立して設定できます。ディスクリプタの詳細については、Registers TRM を参照してください。

- 4 種類のトリガ入力操作があります。
  - Type 0: トリガは、Single transfer を実行します。
  - Type 1: トリガは、Single 1D transfer" を実行します。
  - Type 2: トリガは、現在のディスクリプタを実行します。
  - Type 3: トリガは、ディスクリプタリストを実行します。
- 4 種類のトリガ出力と割込みタイミングがあります。
  - Type 0: Output trigger または Interrupt は、single transfer 後に生成されます。
  - Type 1: Output trigger または Interrupt は、Single 1D transfer 後に生成されます。
  - Type 2: Output trigger または Interrupt は、現在のディスクリプタ実行後に生成されます。
  - Type 3: Output trigger または Interrupt は、ディスクリプタリストの実行後に生成されます。

2.3.2.1 トリガ機能の例

ここでは、トリガ機能の例を示します。以下の例では、ディスクリプタリストは 2D transfer で 2 つのディスクリプタ (Descriptor 0 と Descriptor 1) を連結して構成されます。

例 1:

この例では、Type 0 トリガの動作について説明します。図 5 に Type 0 トリガの Trigger input と Trigger output または Interrupt の動作を示します。

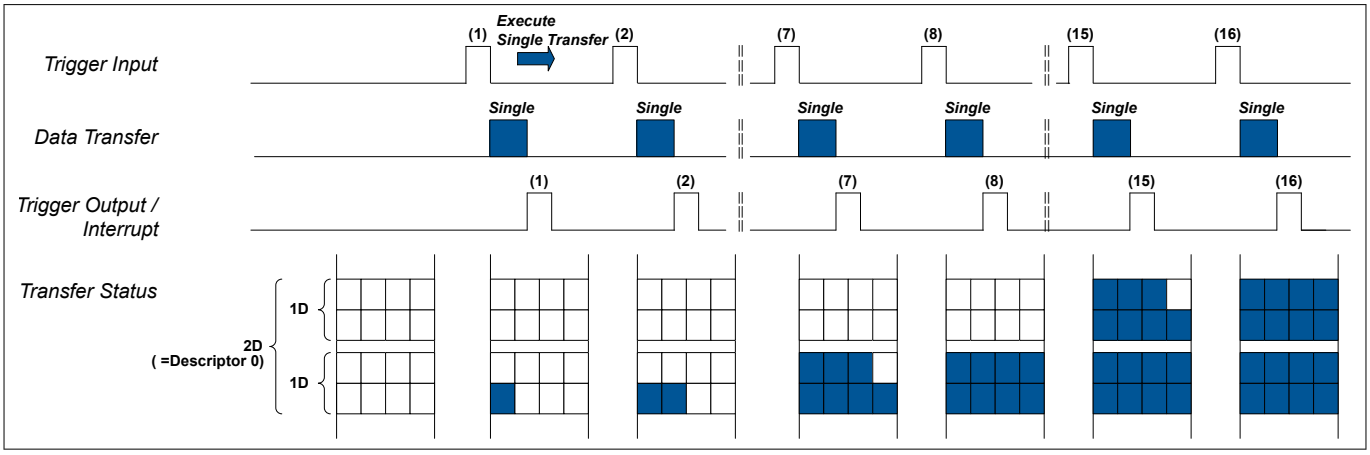


図 5　　トリガ動作例 1

Type 0 トリガでトリガ入力が設定されると、P-DMA/M-DMA はトリガ入力ごとに 1 回の Single transfer を実行します。したがって、Descriptor 0 を完了するためには 16 個のトリガ入力が必要です。

Type 0 トリガでトリガ出力と割込みを設定すると、1 回の転送が完了するたびに Trigger output, Interrupt, またはその両方を出力します。したがって、Descriptor 0 の完了によって、Trigger output, Interrupt, またはその両方を 16 回出力します。

例 2:

この例では、Type 1 トリガの動作について説明します。図 6 に Type 1 トリガの Trigger input と Trigger output または Interrupt の動作を示します。

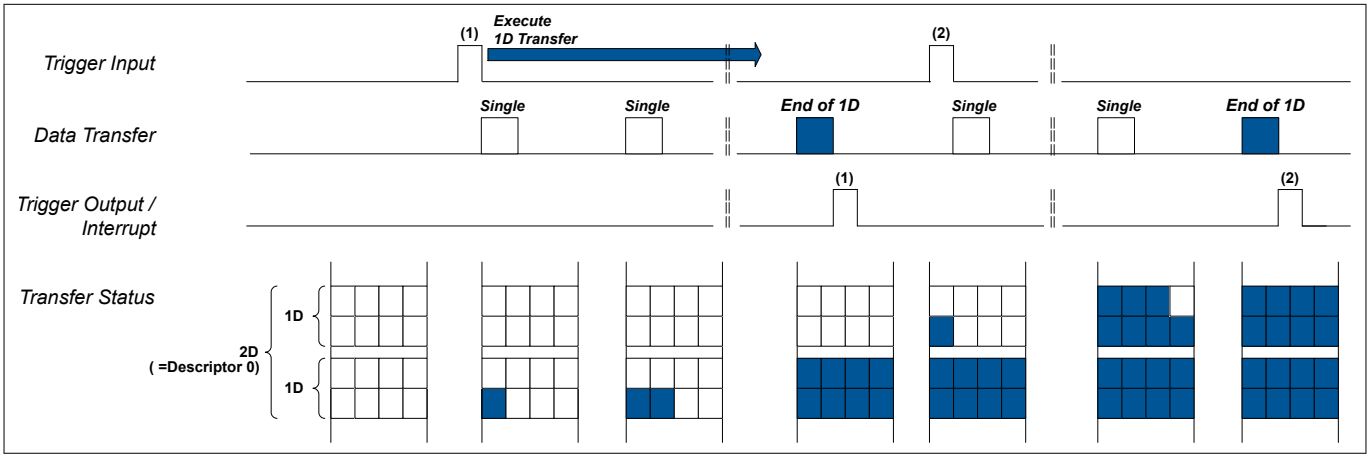


図 6　　トリガ動作例 2

Type 1 トリガでトリガ入力を設定すると、P-DMA/M-DMA はトリガ入力で 1D transfer を行います。次のトリガが再び発生すると、P-DMA/M-DMA は 1D transfer を実行します。したがって、Descriptor 0 を完了するためには 2 つのトリガ入力が必要です。

Type 1 トリガでトリガ出力と割込みを設定すると、1D transfer の転送が完了するたびに Trigger output, Interrupt, またはその両方を出力します。したがって、Descriptor 0 の完了によって、Trigger output, Interrupt, またはその両方を 2 回出力します。

例 3:

この例では、Type 2 トリガの動作について説明します。図 7 に Type 2 トリガの Trigger input と Trigger output または Interrupt の動作を示します。

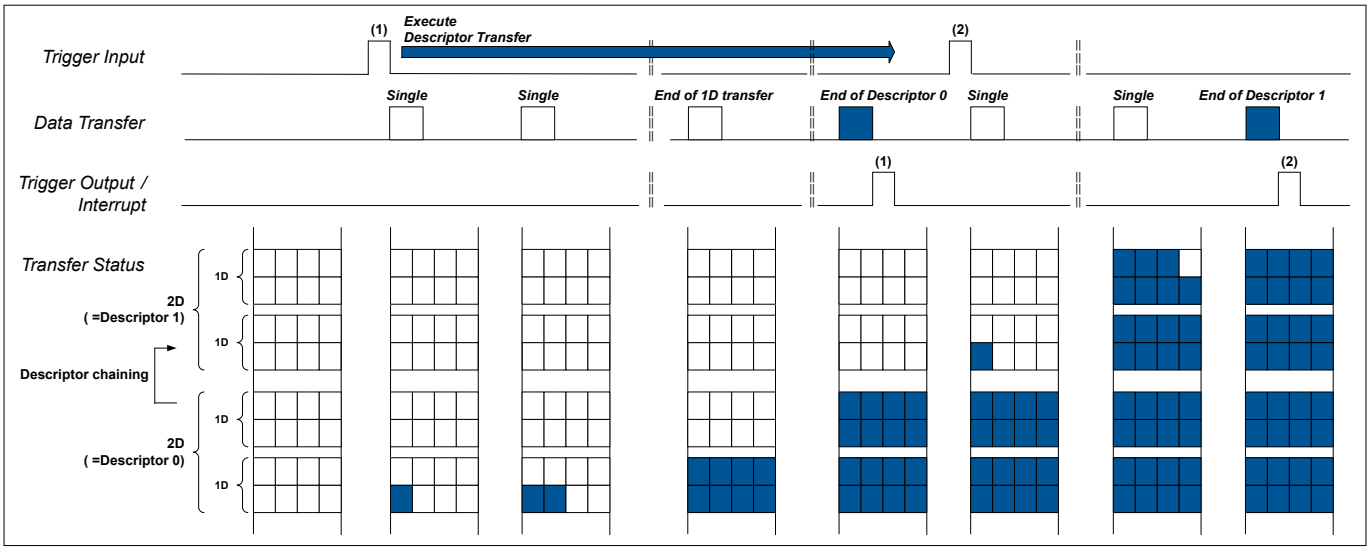


図 7　　トリガ動作例 3

Type 2 トリガでトリガ入力を設定すると、P-DMA/M-DMA はトリガ入力で現在のディスクリプタ (ここでは Descriptor 0)を実行します。次のトリガ入力が発生すると、P-DMA/M-DMA は次のディスクリプタ (ここでは Descriptor 1) を実行します。したがって、ディスクリプタリストを完了させるには 2 つのトリガ入力が必要です。

Type 2 トリガでトリガ出力と割込みを設定した場合、現在のディスクリプタの転送が完了するたびに、Trigger output, Interrupt, またはその両方を出力します。したがって、ディスクリプタリストの完了によって、Trigger output, Interrupt, またはその両方を 2 回出力します。

例 4:

この例では、Type 3 トリガの動作について説明します。図 8 に Type 3 トリガの Trigger input と Trigger output または Interrupt の動作を示します。

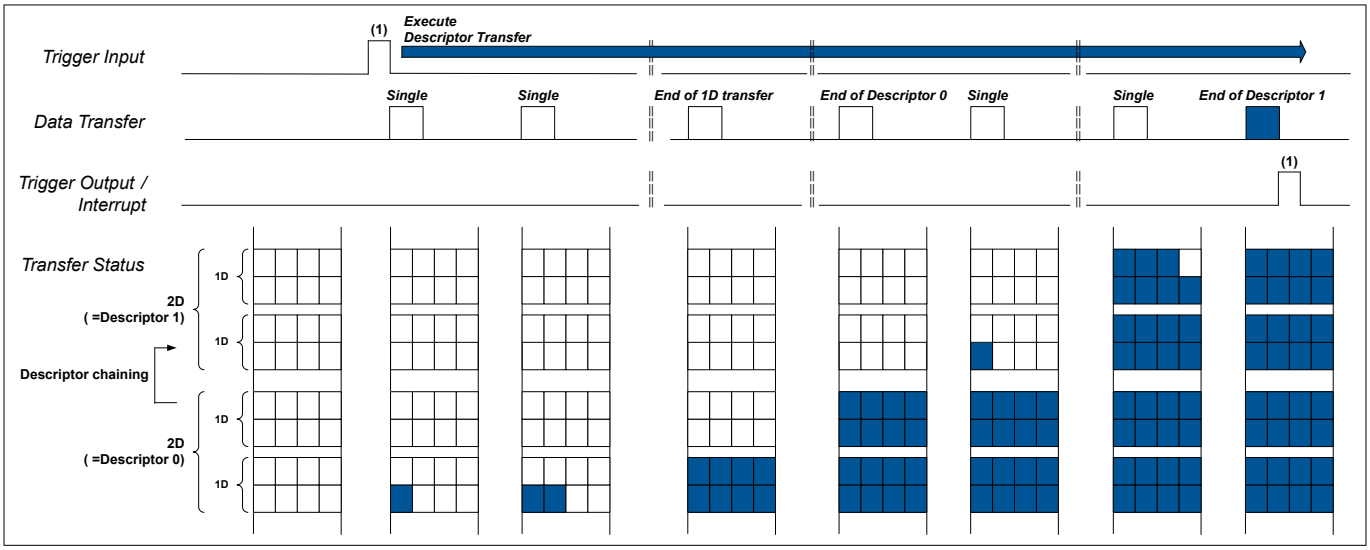


図 8　　トリガ動作例 4

Type 3 トリガでトリガ入力を設定すると、P-DMA/M-DMA はトリガ入力ごとに完全なディスクリプタリストを実行します。したがって、ディスクリプタリストを完了させるためには 1 つのトリガ入力が必要です。

Type 3 トリガでトリガ出力と割込みを設定した場合、ディスクリプタリストの転送完了後に Trigger output, Interrupt, またはその両方を出力します。したがって、ディスクリプタリストの完了によって、Trigger output または Interrupt を 1 回出力します。

2.4 P-DMA/M-DMA チャネルのディセーブルとイネーブル

P-DMA および M-DMA は、複数の独立したデータ転送を実行するように設定/プログラムできます。各データ転送はチャネルによって管理されます。

DMA チャネルの設定では、設定中は DMA チャネルをディセーブルにし、チャネルを設定した後で DMA チャネルをイネーブルにしてください。表 6 に、DMA チャネルをイネーブルまたはディセーブルにするために使用されるレジスタを示します。

チャネルの数は、製品型格によって異なります。使用可能なチャネルの数については、デバイスデータシートを参照してください。

表 6 P-DMA/M-DMA チャネルのディセーブルとイネーブル

DMA タイプ	レジスタ (bit)	説明
P-DMA	DW_CH_STRUCT_CH_CTL.ENABLED (bit31)	0: ディセーブル, 1: イネーブル
M-DMA	DMAC_CH_CH_CTL.ENABLED (bit31)	0: ディセーブル, 1: イネーブル

3 P-DMA ユースケース

ここでは、サンプルドライバライブラリ (SDL) を使用した Smart I/O の使用方法について説明します。このアプリケーションノートに記載されているプログラムコードは、SDL に含まれるものです。SDL については、その他の参考資料を参照してください。

SDL には設定部とドライバ部があります。設定部では、主に目的の動作をさせるためのパラメータ値を設定します。ドライバ部は設定部のパラメータ値に基づいて各レジスタを設定します。設定部は、お客様のシステムに合わせて設定できます。

この例では、CYT2B7 シリーズを使用しています。

3.1 1D Transfer (メモリからペリフェラル)

3.1.1 概要

ここでは、SCB UART モードを使用時に、DMA によって送信データをメモリから TX-FIFO に転送する例を説明します。この例では、UART は 8 ビットのデータフレームを使用し、データが FIFO に書き込まれると送信を開始します。送信するバイト数は 16 です。TX FIFO の設定は 8 ビットのデータ要素で、128 要素です。詳細については、Architecture TRM の「Serial Communications Block (SCB)」の章を参照してください。

P-DMA は、CPU からのソフトウェアトリガで転送を開始し、すべてのデータを送信 FIFO に転送した後に CPU への割込みを生成します。

図 9 にデータ転送動作を示します。

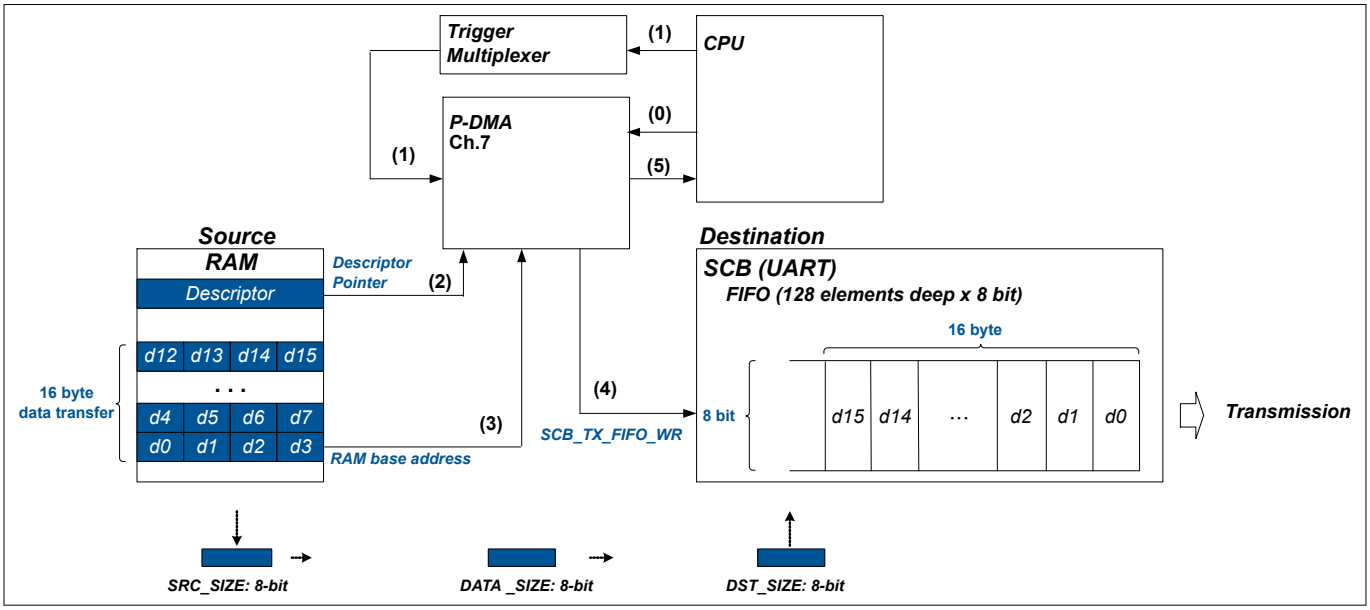


図 9 SCB (UART) を使用したメモリからペリフェラル (1D Transfer) のユースケース

(0) 第 3.1.2 章 チャネルレジスタの初期設定項に従って P-DMA を設定し、SCB と Trigger Multiplexer を設定してください。

(1) CPU は、Software trigger を Trigger multiplexer を介して P-DMA に通知します。

(2) P-DMA は、次の保留中のチャネルをアクティブ時に指定された領域 (ディスクリプタポインタ) からディスクリプタを読み出します。

(3) P-DMA は、転送元 (RAM ベースアドレス) からデータ (d0) を読み出します。

(4) P-DMA はリードデータ (d0) を転送先アドレス (SCB_TX_FIFO_WR) に書き込みます。その後、P-DMA は転送元アドレスを 0x01 だけインクリメントしますが、転送先アドレスのインクリメントはありません。次に、P-DMA は、転

送元アドレス (RAM ベースアドレス+ 0x01) からデータ (d1) を読み出し、再び転送先アドレス (SCB_TX_FIFO_WR) に書き込みます。P-DMA は d15 のデータが転送されるまで (3) から (4) までを繰り返します。

(5) P-DMA は、すべてのデータの転送が完了すると CPU に割込みを通知します。

チャネルレジスタを初期化し、次のようにディスクリプタを設定します。

3.1.2 チャネルレジスタの初期設定

ここでは、このユースケースの P-DMA チャネルとディスクリプタの初期化について説明します。

図 10 は P-DMA の設定手順を示します。

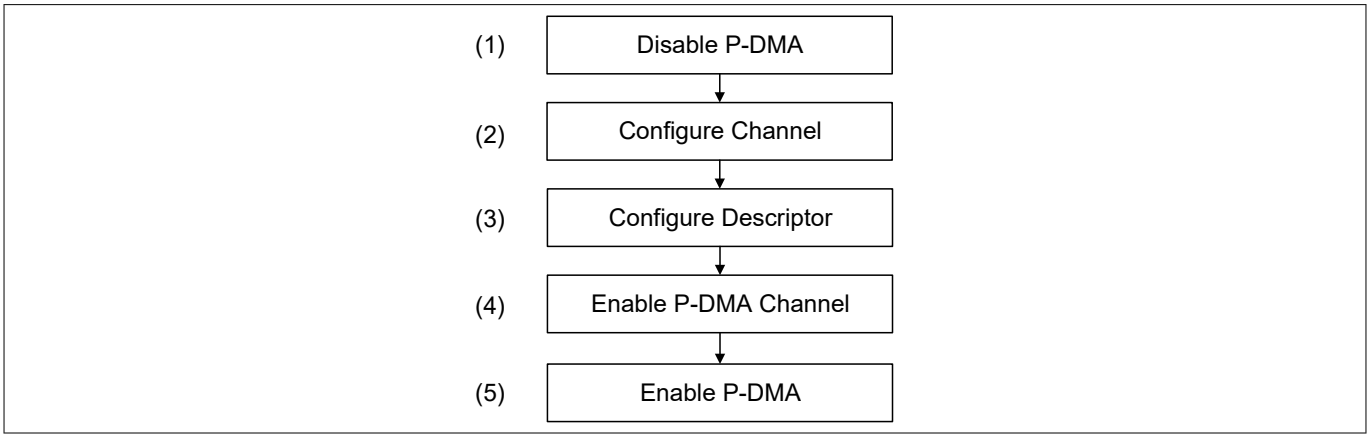


図 10 P-DMA の設定手順

3.1.3 設定とサンプルコード

表 7 は P-DMA 用の SDL におけるパラメータの一覧、表 8 は設定部の機能の一覧です。

表 7 DMA 設定パラメータの一覧

パラメータ	説明	値
DW_CHANNEL	P-DMA チャネルの定義	7ul
DW_CH_INTR	P-DMA チャネル割込みの定義	cpuss_interrupts_dw1_7_IRQn
DW_SW_TRIG	P-DMA SW トリガの定義	TRIG_OUT_MUX_1_PDMA1_TR_IN7
.PDMA_Descriptor	P-DMA の現在のディスクリプタポインタ	&dwUartTxDescr
.preemptable	チャネル プリエンプタブル	0ul
.priority	チャネル優先度	0ul
.enable	チャネル イネーブル	1ul
.deact	DESCR_CTL WAIT_FOR_DEACT	0ul
.intrType	DESCR_CTL INTR_TYPE	CY_PDMA_INTR_DESCR_CMPLT
.trigoutType	DESCR_CTL TR_OUT_TYPE	CY_PDMA_TRIGOUT_DESCR_CMPLT
.chStateAtCmplt	DESCR_CTL CH_DISABLE	CY_PDMA_CH_DISABLED
.triginType	DESCR_CTL TR_IN_TYPE	CY_PDMA_TRIGIN_DESCR
.dataSize	DESCR_CTL DATA_SIZE	CY_PDMA_BYTE

(続く)

表 7 (続き) DMA 設定パラメータの一覧

パラメータ	説明	値
.srcTxfrSize	DESCR_CTL SRC_TRANSFER_SIZE	0ul
.destTxfrSize	DESCR_CTL DST_TRANSFER_SIZE	1ul
.descrType	DESCR_CTL DESCR_TYPE	CY_PDMA_1D_TRANSFER
.srcAddr	DESCR_SRC	(void *)g_uart_tx_buffer
.destAddr	DESCR_DST	(void *)&CY_USB_SCB_TYPE->unTX_FIFO_WR.u32Register
.srcXincr	DESCR_X_CTL SRC_X_INCR	1ul
.destXincr	DESCR_X_CTL DST_X_INCR	0ul
.xCount	DESCR_X_CTL X_COUNT	UART_TX_BUFFER_SIZE
.descrNext	DESCR_NEXT_PTR	NULL

表 8 DMA 設定機能の一覧

機能	説明	備考
DW1_Ch_IntHandler()	P-DMA1 割込み用ハンドラ	Code Listing 3 を参照してください
Cy_PDMA_Disable()	P-DMA ディセーブルの設定	P-DMA_CTL_ENABLED ビットに書き込みます。 Code Listing 6 を参照してください
Cy_PDMA_Chnl_DeInit()	P-DMA をデフォルト値にリセット	チャネルに対応するレジスタの内容をすべてクリアします。 Code Listing 7 を参照してください
Cy_PDMA_Descr_Init()	P-DMA ディスクリプタ初期化の設定	Code Listing 8 を参照してください
Cy_PDMA_Chnl_Init()	P-DMA チャネル初期化の設定	Code Listing 9 を参照してください
Cy_PDMA_Chnl_Enable()	P-DMA チャネル イネーブルの設定	Code Listing 10 を参照してください
Cy_PDMA_Chnl_SetInterruptMask()	P-DMA チャネル割込みマスクの設定	Code Listing 11 を参照してください
Cy_PDMA_Enable()	P-DMA イネーブルの設定	P-DMA_CTL_ENABLED ビットに書き込みます。 Code Listing 12 を参照してください
Cy_SysInt_InitIRQ()	割込みの設定	-
Cy_SysInt_SetSystemIrqVector()	割込みベクタの設定	-
NVIC_EnableIRQ()	NVIC イネーブル割込み	-
Cy_TrigMux_SwTrigger()	ソフトウェアトリガの生成	-
Cy_PDMA_Chnl_GetInterruptStatusMasked()	1 回のロードで、対応する INTR と INTR_MASK フィールドの論理和を返します。	Code Listing 4 を参照してください
Cy_PDMA_Chnl_ClearInterrupt()	P-DMA チャネルが割込みステータスをクリアするように設定	Code Listing 5 を参照してください

Code Listing 1 は、1D Transfer (メモリからペリフェラル) へのプログラム例を示します。GPIO, UART, およびクロックの設定については、Architecture TRM およびアプリケーションノートを参照してください。

以下は、SDL のドライバ部のレジスタ表記を説明します。

- base は DMA レジスタのベースアドレスへのポインタを意味します。
- **base-> CH_STRUCT[idx].unCH_CTL.u32Register** は Registers TRM に記載されている DWx_CH_STRUCT[idx] レジスタです。その他のレジスタについても、同様の意味です。“x”はポートサフィックス番号、“idx”はレジスタインデックス番号を表します。
- レジスタ設定のパフォーマンスを向上させるため、SDL は完全な 32 ビットデータをレジスタに書き込みます。各ビットフィールドが生成され、最終的に 32 ビットデータとしてレジスタに書き込まれます。

```
pstcPDMA->CH_STRUCT[chNum].unCH_CTL.u32Register;
pstcPDMA->CH_STRUCT[chNum].unCH_IDX.u32Register;
pstcPDMA->CH_STRUCT[chNum].unCH_CURR_PTR.u32Register;
pstcPDMA->CH_STRUCT[chNum].unINTR_MASK.u32Register;
pstcPDMA->CH_STRUCT[chNum].unINTR_SET.u32Register;
```

レジスタ表記の結合と構造の詳細については、`hdr/rev_x/ip` の `cyip_dw_v2.h` を参照してください。

Code Listing 1 1D Transfer (メモリからペリフェラル) の例

```
/* Define DW channel
Define DW channel interrupt
Define DW SW Trigger  */
#define DW_CHANNEL              7ul
#define DW_CH_INTR              cpuss_interrupts_dw1_7_IRQn
#define DW_SW_TRIG              TRIG_OUT_MUX_1_PDMA1_TR_IN7

void DW1_Ch_IntHandler(void);     /* See Code Listing 3. */

int main(void)
{
    :

/*  (1) Disable P-DMA. See Code Listing 6.
(2) Resets P-DMA to default values. See Code Listing 7.
(3) Configures P-DMA descriptor initialize.  See Code Listing 8.
(4) Enable P-DMA channel. See Code Listing 10.
(5) Enable P-DMA. See Code Listing 12   */
    /* Initialize & Enable DMA */
    Cy_PDMA_Disable(DW1);
    Cy_PDMA_Chnl_DeInit(DW1, DW_CHANNEL);
    Cy_PDMA_Descr_Init(&dwUartTxDescr, &dw1ChUartTxConfig);
    Cy_PDMA_Chnl_Init(DW1, DW_CHANNEL, &dwUartTxConfig);
    Cy_PDMA_Chnl_Enable(DW1, DW_CHANNEL);
    Cy_PDMA_Chnl_SetInterruptMask(DW1, DW_CHANNEL);
    Cy_PDMA_Enable(DW1);

    /* Prepare source buffer data */
    for(uint32_t i = 0ul; i < UART_TX_BUFFER_SIZE; i++)
    {
        g_uart_tx_buffer[i] = (i + 48ul);    /* ASCII Code '0','1','2' ... */
    }

    /* Interrupt Initialization */
    Cy_SysInt_InitIRQ(&stc_sysint_irq_cfg);
    /*  Configures interrupt   */
    Cy_SysInt_SetSystemIrqVector(stc_sysint_irq_cfg.sysIntSrc, DW1_Ch_IntHandler);
    NVIC_EnableIRQ(stc_sysint_irq_cfg.intIdx);

    /* SW Trigger */
    /*  Generates a software trigger    */
    Cy_TrigMux_SwTrigger(DW_SW_TRIG, TRIGGER_TYPE_EDGE, 1ul);

    for(;;);

}
```

Code Listing 2 P-DMA チャネル、ディスクリプタの設定

```
/**
 * \var uint8_t g_uart_tx_buffer
 * \brief UART TX buffer
 */
uint8_t  g_uart_tx_buffer[UART_TX_BUFFER_SIZE] = { 0ul };   /*  Variable for source address
TX_buffer  */


/**
 * \var cy_stc_pdma_descr_t dwUartTxDescr   /*  Variable for P-DMA descriptor   */
 * \brief PDMA descriptor
 */
static  cy_stc_pdma_descr_t         dwUartTxDescr;

/**
 * \var cy_stc_pdma_chnl_config_t dwUartTxConfig
 * \brief PDMA configuration */
     /*  Configure P-DMA channel  */
const   cy_stc_pdma_chnl_config_t   dwUartTxConfig =
{
    .PDMA_Descriptor = &dwUartTxDescr,
    .preemptable     = 0ul,
    .priority        = 0ul,
    .enable          = 1ul,
};

/**
 * \var cy_stc_pdma_descr_config_t dw1ChUartTxConfig
 * \brief PDMA descriptor configuration */
static  cy_stc_pdma_descr_config_t  dw1ChUartTxConfig =
{
    .deact          =   0ul,   /*  Configure P-DMA descriptor  */
    .intrType       =   CY_PDMA_INTR_DESCR_CMPLT,   /*  Configure P-DMA descriptor  */
    .trigoutType    =   CY_PDMA_TRIGOUT_DESCR_CMPLT,   /*  Configure P-DMA descriptor  */
    .chStateAtCmplt =   CY_PDMA_CH_DISABLED,   /*  Configure P-DMA descriptor  */
    .triginType     =   CY_PDMA_TRIGIN_DESCR,   /*  Configure P-DMA descriptor  */
    .dataSize       =   CY_PDMA_BYTE,   /*  Configure P-DMA descriptor  */
    .srcTxfrSize    =   0ul, // same as "dataSize"   /*  Configure P-DMA descriptor  */
    .destTxfrSize   =   1ul, // 32bit width   /*  Configure P-DMA descriptor  */
    .descrType      =   CY_PDMA_1D_TRANSFER,   /*  Configure P-DMA descriptor  */
    .srcAddr        =   (void *)g_uart_tx_buffer,   /*  Configure P-DMA descriptor  */
    .destAddr       =   (void *)&CY_USB_SCB_TYPE->unTX_FIFO_WR.u32Register,   /*  Configure P-
DMA descriptor  */
    .srcXincr       =   1ul,   /*  Configure P-DMA descriptor  */
    .destXincr      =   0ul,   /*  Configure P-DMA descriptor  */
    .xCount         =   UART_TX_BUFFER_SIZE,   /*  Configure P-DMA descriptor  */
    .descrNext      =   NULL   /*  Configure P-DMA descriptor  */
};

/**
 * \var cy_stc_sysint_irq_t stc_sysint_irq_cfg
```

```
 * \brief IRQ DMA configuration */
   /*  Configure interrupt  */
static const cy_stc_sysint_irq_t stc_sysint_irq_cfg =
{
    .sysIntSrc = DW_CH_INTR,
    .intIdx    = CPUIntIdx2_IRQn,
    .isEnabled = true,
};
```

Code Listing 3 DW1_Ch_IntHandler()

```
void DW1_Ch_IntHandler(void)
{
    uint32_t masked;

    masked = Cy_PDMA_Chnl_GetInterruptStatusMasked(DW1, DW_CHANNEL);   /*  See code Listing 4.
*/
    if ((masked & CY_PDMA_INTRCAUSE_COMPLETION) != 0ul)
    {
        /* Clear Complete DMA interrupt flag */
        Cy_PDMA_Chnl_ClearInterrupt(DW1, DW_CHANNEL);    /*  See Code Listing 5.  */
    }
    else
    {
        CY_ASSERT(false);
    }
}
```

Code Listing 4 Cy_PDMA_Chnl_GetInterruptStatusMasked()

```
__STATIC_INLINE uint32_t Cy_PDMA_Chnl_GetInterruptStatusMasked(const volatile stc_DW_t
*pstcPDMA, uint32_t chNum)
{
    return (pstcPDMA->CH_STRUCT[chNum].unINTR_MASKED.u32Register);
```

Code Listing 5 Cy_PDMA_Chnl_ClearInterrupt()

```
void Cy_PDMA_Chnl_ClearInterrupt(volatile stc_DW_t *pstcPDMA, uint32_t chNum)
{
    pstcPDMA->CH_STRUCT[chNum].unINTR.u32Register = CY_PDMA_INTR_BIT_MASK;

    /* Readback of the register is required by hardware. */
    (void) pstcPDMA->CH_STRUCT[chNum].unINTR.u32Register;
}
```

Code Listing 6 Cy_PDMA_Disable()

```
void Cy_PDMA_Disable(volatile stc_DW_t *pstcPDMA)
{
    pstcPDMA->unCTL.stcField.u1ENABLED  =       0ul;   /*  Write to P-DMA_CTL_ENABLED bit  */
}
```

Code Listing 7 Cy_PDMA_Chnl_DeInit()

```
void Cy_PDMA_Chnl_DeInit(volatile stc_DW_t *pstcPDMA, uint32_t chNum)
   /*  Resets P-DMA to default value  */
{
    pstcPDMA->CH_STRUCT[chNum].unCH_CTL.u32Register            = 0ul;
    pstcPDMA->CH_STRUCT[chNum].unCH_IDX.u32Register            = 0ul;
    pstcPDMA->CH_STRUCT[chNum].unCH_CURR_PTR.u32Register       = 0ul;
    pstcPDMA->CH_STRUCT[chNum].unINTR_MASK.u32Register         = 0ul;
    pstcPDMA->CH_STRUCT[chNum].unINTR_SET.u32Register          = 0ul;
}
```

Code Listing 8 Cy_PDMA_Descr_Init()

```
cy_en_pdma_status_t Cy_PDMA_Descr_Init(cy_stc_pdma_descr_t* descriptor, const
cy_stc_pdma_descr_config_t* config)
{
    cy_en_pdma_status_t retVal = CY_PDMA_ERR_UNC;

    if ((descriptor != NULL) && (config != NULL))
    {
        descriptor->unPDMA_DESCR_CTL.stcField.u2WAIT_FOR_DEACT          =       config->deact;
        descriptor->unPDMA_DESCR_CTL.stcField.u2INTR_TYPE               =       config-
>intrType;
        descriptor->unPDMA_DESCR_CTL.stcField.u2TR_OUT_TYPE             =       config-
>trigoutType;
        descriptor->unPDMA_DESCR_CTL.stcField.u2TR_IN_TYPE              =       config-
>triginType;
        descriptor->unPDMA_DESCR_CTL.stcField.u1SRC_TRANSFER_SIZE       =       config-
>srcTxfrSize;
        descriptor->unPDMA_DESCR_CTL.stcField.u1DST_TRANSFER_SIZE       =       config-
>destTxfrSize;
        descriptor->unPDMA_DESCR_CTL.stcField.u1CH_DISABLE              =       config-
>chStateAtCmplt;
        descriptor->unPDMA_DESCR_CTL.stcField.u2DATA_SIZE               =       config-
>dataSize;
        descriptor->unPDMA_DESCR_CTL.stcField.u2DESCR_TYPE              =       config-
>descrType;

        descriptor->u32PDMA_DESCR_SRC                                   =
(uint32_t)config->srcAddr;
        descriptor->u32PDMA_DESCR_DST                                   =
(uint32_t)config->destAddr;

        switch(config->descrType)
        {
        case (uint32_t)CY_PDMA_SINGLE_TRANSFER:
        {
            descriptor->unPDMA_DESCR_X_CTL.u32Register                  =       (uint32_t)config-
>descrNext;
            break;
        }
        case (uint32_t)CY_PDMA_1D_TRANSFER:
        {
            descriptor->unPDMA_DESCR_X_CTL.stcField.u12SRC_X_INCR       =       (uint32_t)config-
>srcXincr;
            descriptor->unPDMA_DESCR_X_CTL.stcField.u12DST_X_INCR       =       (uint32_t)config-
>destXincr;
            descriptor->unPDMA_DESCR_X_CTL.stcField.u8X_COUNT           =       (uint32_t)((config-
>xCount) - 1ul);
            descriptor->unPDMA_DESCR_Y_CTL.u32Register                  =       (uint32_t)config-
>descrNext;
            break;
        }
        case (uint32_t)CY_PDMA_CRC_TRANSFER:
```

```
        {
            descriptor->unPDMA_DESCR_X_CTL.stcField.u12SRC_X_INCR   =       (uint32_t)config-
>srcXincr;
            descriptor->unPDMA_DESCR_X_CTL.stcField.u12DST_X_INCR   =       0ul;
            descriptor->unPDMA_DESCR_X_CTL.stcField.u8X_COUNT       =       (uint32_t)((config-
>xCount) - 1ul);
            descriptor->unPDMA_DESCR_Y_CTL.u32Register              =       (uint32_t)config-
>descrNext;
            break;
        }
        case (uint32_t)CY_PDMA_2D_TRANSFER:
        {
            descriptor->unPDMA_DESCR_X_CTL.stcField.u12SRC_X_INCR   =       (uint32_t)config-
>srcXincr;
            descriptor->unPDMA_DESCR_X_CTL.stcField.u12DST_X_INCR   =       (uint32_t)config-
>destXincr;
            descriptor->unPDMA_DESCR_X_CTL.stcField.u8X_COUNT       =       (uint32_t)((config-
>xCount) - 1ul);

            descriptor->unPDMA_DESCR_Y_CTL.stcField.u12SRC_Y_INCR   =       (uint32_t)config-
>srcYincr;
            descriptor->unPDMA_DESCR_Y_CTL.stcField.u12DST_Y_INCR   =       (uint32_t)config-
>destYincr;
            descriptor->unPDMA_DESCR_Y_CTL.stcField.u8Y_COUNT       =       (uint32_t)((config-
>yCount) - 1ul);

            descriptor->u32PDMA_DESCR_NEXT_PTR                      =       (uint32_t)config-
>descrNext;
            break;
        }
        default:
        {
            /* Unsupported type of descriptor */
            break;
        }
        }

        retVal = CY_PDMA_SUCCESS;
    }
    else
    {
        retVal = CY_PDMA_INVALID_INPUT_PARAMETERS;
    }

    return retVal;
}
```

Code Listing 9 Cy_PDMA_Chnl_Init()

```
cy_en_pdma_status_t Cy_PDMA_Chnl_Init(volatile stc_DW_t *pstcPDMA, uint32_t chNum, const
cy_stc_pdma_chnl_config_t* chnlConfig)
{
    cy_en_pdma_status_t retVal = CY_PDMA_ERR_UNC;

    if ((pstcPDMA != NULL) && (chnlConfig != NULL))
    {
        /* Set current descriptor */
        pstcPDMA->CH_STRUCT[chNum].unCH_CURR_PTR.u32Register           =  (uint32_t)chnlConfig-
>PDMA_Descriptor;

        /* Set if the channel is preemtable */
        pstcPDMA->CH_STRUCT[chNum].unCH_CTL.stcField.u1PREEMPTABLE     =  chnlConfig-
>preemptable;

        /* Set channel priority */
        pstcPDMA->CH_STRUCT[chNum].unCH_CTL.stcField.u2PRIO            =  chnlConfig->priority;

        /* Set enabled status */
        pstcPDMA->CH_STRUCT[chNum].unCH_CTL.stcField.u1ENABLED         =  chnlConfig->enable;

        retVal = CY_PDMA_SUCCESS;
    }
    else
    {
        retVal = CY_PDMA_INVALID_INPUT_PARAMETERS;
    }

    return (retVal);
}
```

Code Listing 10 Cy_PDMA_Chnl_Enable()

```
__STATIC_INLINE void Cy_PDMA_Chnl_Enable(volatile stc_DW_t *pstcPDMA, uint32_t chNum)
{
    pstcPDMA->CH_STRUCT[chNum].unCH_CTL.stcField.u1ENABLED = 1ul;
}
```

Code Listing 11 Cy_PDMA_Chnl_SetInterruptMask()

```
__STATIC_INLINE void Cy_PDMA_Chnl_SetInterruptMask(volatile stc_DW_t *pstcPDMA, uint32_t chNum)
{
    pstcPDMA->CH_STRUCT[chNum].unINTR_MASK.u32Register = CY_PDMA_INTR_BIT_MASK;
}
```

Code Listing 12 Cy_PDMA_Enable()

```
void Cy_PDMA_Enable(volatile stc_DW_t *pstcPDMA)
{
pstcPDMA->unCTL.stcField.u1ENABLED  =       1ul;     // Write to P-DMA_CTL_ENABLED bit //
}
```

3.2 1D Transfer (ペリフェラルからメモリ)

3.2.1 概要

ここでは、複数のポートの入力状態をメモリ (RAM) に周期的なトリガで転送する例を説明します。TCPWM は、周期的なトリガを生成するために使用されます。詳細は、Architecture TRM の「Timer, Counter, and PWM」の章を参照してください。

ポートの入力状態は、TCPWM に設定された周期的なタイミングでメモリに格納されます。その結果、CPU はポートレジスタにアクセスしなくてもメモリの読出しにより、指定したタイミングでポートの状態を確認できます。これは、低電力モードに移行する前にポートの状態を保存する場合に便利です。

この例では、P-DMA は GPIO_PRT_IN レジスタの値を RAM に転送します。転送するレジスタの数は 23 です (ポート 0 からポート 22 まで)。GPIO_PRT_IN レジスタは 0x80 の間隔で配置されています。P-DMA は、すべてのレジスタからデータを転送した後に割込みを生成します。図 11 にデータ転送動作を示します。

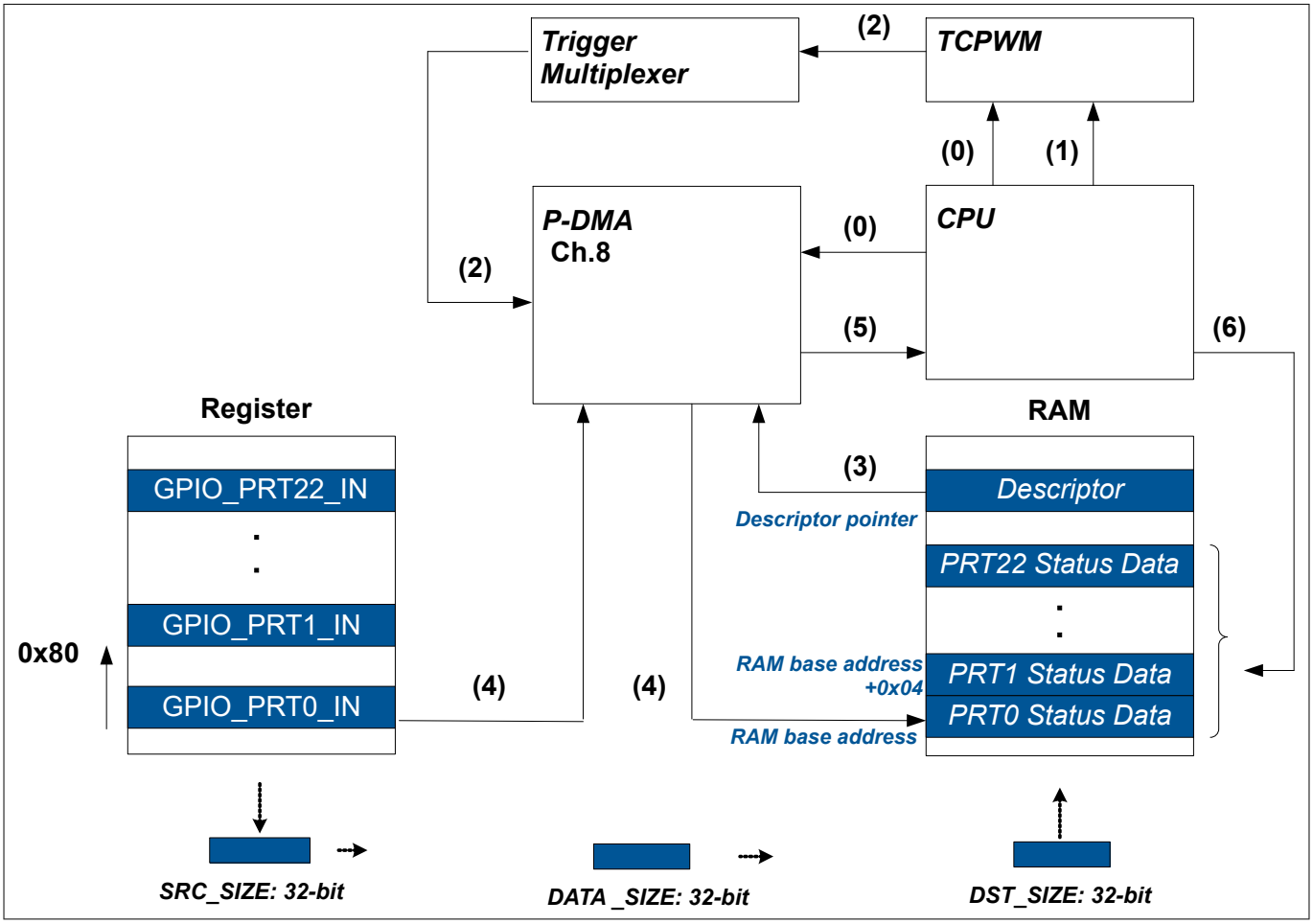


図 11 1D Transfer (ペリフェラルからメモリ) の動作

(0) P-DMA の初期設定に従って P-DMA を設定してください。さらに、TCPWM とトリガマルチプレクサを設定してください。

(1) CPU が TCPWM タイマを起動します。

(2) TCPWM は、指定したカウント値 (オーバフロー) に達すると、Trigger Multiplexer を介して P-DMA にトリガを出力します。

(3) P-DMA は、転送を受け付け時に、指定された領域 (ディスクリプタポインタ) からディスクリプタを読み出します。

(4) P-DMA は、転送元アドレス (GPIO_PRT0_IN) からデータを読み出し、読み出したデータを転送先アドレス (RAM ベースアドレス) に書き込みます。その後、P-DMA は転送元アドレスを 0x80 だけインクリメントし、転送先アドレスは 0x04 だけインクリメントします。次に、P-DMA は、転送元アドレス (GPIO_PRT1_IN) からデータを読み出し、転送先アドレス (RAM ベースアドレス+ 0x04) に再度書き込みます。

(5) すべての転送が完了すると P-DMA は、CPU に割込みを通知します。

(6) CPU は割込みを受け付け、RAM にあるポート状態を読み出します。

TCPWM が再度トリガを出力すると、手順 (3) から (6) を繰り返します。

3.2.2 P-DMA の初期設定

ここでは、このユースケースの P-DMA チャネルとディスクリプタの初期化について説明します。

図 12 に P-DMA の設定手順を示します。

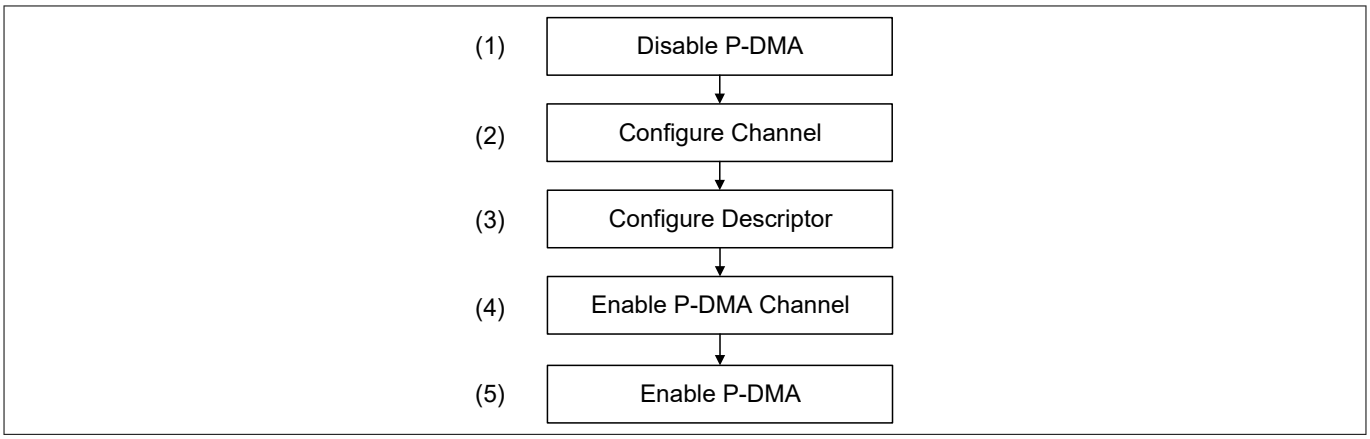


図 12 P-DMA の設定手順

3.2.3 設定とサンプルコード

表 9 は DMA 用の SDL におけるパラメータの一覧、表 10 は設定部の機能の一覧です。

表 9 DMA 設定パラメータの一覧

パラメータ	説明	値
TCPWMx_GRPx_CNTx_COUNTER	カウンタを使用した TCPWM の定義	TCPWM0_GRP0_CNT0
DW_CHANNEL	P-DMA チャネルの定義	8ul
TCPWM_TO_MUX_TRIG	マルチプレクサへのトリガ TCPWM の定義	TRIG_IN_MUX_3_TCPWM_16_TR_OUT00
MUX_TO_PDMA_TRIG	P-DMA に対するトリガマルチプレクサの定義	TRIG_OUT_MUX_3_PDMA0_TR_IN8

(続く)

表 9 (続き) DMA 設定パラメータの一覧

パラメータ	説明	値
DW_CH_INTR	P-DMA チャネル割込みの定義	cpuss_interrupts_dw0_8_IRQn
DST_BUFFER_SIZE	転送先バッファ サイズの定義	23ul
.PDMA_Descriptor	P-DMA の現在のディスクリプタポインタ	&dwTcpwmDescr
.preemptable	チャネル プリエンプタブル	0ul
.priority	チャネル優先度	0ul
.enable	チャネル イネーブル	1ul
.deact	DESCR_CTL WAIT_FOR_DEACT	0ul
.intrType	DESCR_CTL INTR_TYPE	CY_PDMA_INTR_X_LOOP_CMPLT
.trigoutType	DESCR_CTL TR_OUT_TYPE	CY_PDMA_TRIGOUT_X_LOOP_CMPLT
.chStateAtCmplt	DESCR_CTL CH_DISABLE	CY_PDMA_CH_ENABLED
.triginType	DESCR_CTL TR_IN_TYPE	CY_PDMA_TRIGIN_DESCR
.dataSize	DESCR_CTL DATA_SIZE	CY_PDMA_WORD
.srcTxfrSize	DESCR_CTL SRC_TRANSFER_SIZE	0ul
.destTxfrSize	DESCR_CTL DST_TRANSFER_SIZE	0ul
.descrType	DESCR_TYPE	CY_PDMA_1D_TRANSFER
.srcAddr	DESCR_SRC	(void *)&GPIO_PRT0->unIN.u32Register
.destAddr	DESCR_DST	(void *)g_DestBuffer
.srcXincr	DESCR_X_CTL SRC_X_INCR	32ul
destXincr	DESCR_X_CTL DST_X_INCR	1ul
.xCount	DESCR_X_CTL X_COUNT	DST_BUFFER_SIZE
.descrNext	DESCR_NEXT_PTR	&dwTcpwmDescr

表 10 DMA 設定機能の一覧

機能	説明	備考
DW0_Ch_IntHandler()	P-DMA0 割込み用ハンドラ	Code Listing 15 を参照してください
Counter_Handler()	TCPWM カウンタ割込み用ハンドラ	Code Listing 18 を参照してください
Cy_SysInt_InitIRQ()	割込みの設定	-
Cy_SysInt_SetSystemIrqVector()	割込みベクタの設定	-
NVIC_SetPriority()	NVIC セット プライオリティ	-
NVIC_EnableIRQ()	NVIC イネーブル割込み	-
Cy_PDMA_Disable()	P-DMA ディセーブルの設定	P-DMA_CTL_ENABLED ビットに書き込みます。 Code Listing 19 を参照してください

(続く)

表 10 (続き) DMA 設定機能の一覧

機能	説明	備考
Cy_PDMA_Chnl_DeInit()	P-DMA をデフォルト値にリセット	チャネルに対応するレジスタの内容をすべてクリアします。 Code Listing 20 を参照してください
Cy_PDMA_Descr_Init()	P-DMA ディスクリプタ初期化の設定	Code Listing 21 を参照してください
Cy_PDMA_Chnl_Init()	P-DMA チャネル初期化の設定	Code Listing 22 を参照してください
Cy_PDMA_Chnl_Enable()	P-DMA チャネル イネーブルの設定	Code Listing 23 を参照してください
Cy_PDMA_Chnl_SetInterruptMask()	P-DMA チャネル割込みマスクの設定	Code Listing 24 を参照してください
Cy_PDMA_Enable()	P-DMA イネーブルの設定	DW_CTL_ENABLED ビットに書き込みます。 Code Listing 25 を参照してください
Cy_TrigMux_Connect()	トリガ マルチプレクサの初期化	-
Cy_Tcpwm_TriggerStart()	TCPWM 開始	-
Cy_PDMA_Chnl_GetInterruptStatusMasked()	1 回のロードで、対応する INTR と INTR_MASK フィールドの論理和を返す	Code Listing 16 を参照してください
Cy_PDMA_Chnl_ClearInterrupt()	P-DMA チャネルが割込みステータスをクリアするように設定	Code Listing 17 を参照してください
Cy_Tcpwm_Counter_ClearCC0_Intr()	TCPWM CC0 割り込みフラグをクリア	-

Code Listing 13 は、1D Transfer (ペリフェラル→メモリ) するプログラム例を示します。GPIO, TCPWM, およびクロックの設定については、Architecture TRM およびアプリケーションノートを参照してください。

Code Listing 13 1D Transfer (ペリフェラルからメモリ)例

```
#define TCPWMx_GRPx_CNTx_COUNTER            TCPWM0_GRP0_CNT0  /* Write to P-DMA_CTL_ENABLED bit
*/


/*  Define P-DMA channel
Define Trigger TCPWM to MUX
Define Trigger MUX to P-DMA
Define P-DMA channel interrupt
Define DST buffer size   */
#define DW_CHANNEL                          8ul
#define TCPWM_TO_MUX_TRIG                   TRIG_IN_MUX_3_TCPWM_16_TR_OUT00
#define MUX_TO_PDMA_TRIG                    TRIG_OUT_MUX_3_PDMA0_TR_IN8
#define DW_CH_INTR                          cpuss_interrupts_dw0_8_IRQn
#define DST_BUFFER_SIZE                     23ul


void DW0_Ch_IntHandler(void);   /*  See Code Listing 15   */
void Counter_Handler(void);    /*  See Code Listing 18   */

int main(void)
{

:

    /*  (1) Disable P-DMA. See Code Listing 19.
(2) Resets P-DMA to default values. See Code Listing 20.
(3) Configures P-DMA descriptor initialize.  See Code Listing 21.
(4) Enable P-DMA channel. See Code Listing 23.
(5) Enable P-DMA. See Code Listing 25.  */
    /* Initialize & Enable DMA */
    Cy_PDMA_Disable(DW0);
    Cy_PDMA_Chnl_DeInit(DW0, DW_CHANNEL);
    Cy_PDMA_Descr_Init(&dwTcpwmDescr, &dw0ChTcpwmConfig);
    Cy_PDMA_Chnl_Init(DW0, DW_CHANNEL, &dwTcpwmConfig);
    Cy_PDMA_Chnl_Enable(DW0, DW_CHANNEL);
    Cy_PDMA_Chnl_SetInterruptMask(DW0, DW_CHANNEL);
    Cy_PDMA_Enable(DW0);

    /* Trigger Multiplexer Initialization */
    /* TCPWM To DW */
    Cy_TrigMux_Connect(TCPWM_TO_MUX_TRIG, MUX_TO_PDMA_TRIG, CY_TR_MUX_TR_INV_DISABLE,
TRIGGER_TYPE_LEVEL, 0ul);

    /* Interrupt Initialization */
    Cy_SysInt_InitIRQ(&stc_sysint_irq_cfg);
    Cy_SysInt_SetSystemIrqVector(stc_sysint_irq_cfg.sysIntSrc, DW0_Ch_IntHandler);     /*  See
Configures interrupt   */
    NVIC_SetPriority(stc_sysint_irq_cfg.intIdx, 0ul);
    NVIC_EnableIRQ(stc_sysint_irq_cfg.intIdx);

    /* TCPWM Start */
```

```
    Cy_Tcpwm_TriggerStart(TCPWMx_GRPx_CNTx_COUNTER);    /*  See TCPWM start  */

    for(;;);
}
```

Code Listing 14 P-DMA チャネル、ディスクリプタの設定

```
/**
 * \var cy_stc_pdma_descr_t dwTcpwmDescr
 * \brief PDMA descriptor */
static  cy_stc_pdma_descr_t         dwTcpwmDescr;  /*  Variable for source address TX_buffer */

/**
 * \var cy_stc_pdma_chnl_config_t dwTcpwmConfig
 * \brief PDMA configuration */
    /*  Configure P-DMA channel  */
const   cy_stc_pdma_chnl_config_t   dwTcpwmConfig =
{
    .PDMA_Descriptor = &dwTcpwmDescr,
    .preemptable     = 0ul,
    .priority        = 0ul,
    .enable          = 1ul,
};

/**
 * \var cy_stc_pdma_descr_config_t dw0ChTcpwmConfig
 * \brief PDMA descriptor configuration */
static  cy_stc_pdma_descr_config_t  dw0ChTcpwmConfig =
    /*  Configure P-DMA descriptor  */
{
    .deact          =   0ul,
    .intrType       =   CY_PDMA_INTR_X_LOOP_CMPLT,
    .trigoutType    =   CY_PDMA_TRIGOUT_X_LOOP_CMPLT,
    .chStateAtCmplt =   CY_PDMA_CH_ENABLED,
    .triginType     =   CY_PDMA_TRIGIN_DESCR,
    .dataSize       =   CY_PDMA_WORD,
    .srcTxfrSize    =   0ul, /* Same as dataSize */
    .destTxfrSize   =   0ul, /* Same as dataSize */
    .descrType      =   CY_PDMA_1D_TRANSFER,
    .srcAddr        =   (void *)&GPIO_PRT0->unIN.u32Register,
    .destAddr       =   (void *)g_DestBuffer,
    .srcXincr       =   32ul, /* address +80h */
    .destXincr      =   1ul,  /* address +04h */
    .xCount         =   DST_BUFFER_SIZE,
    .descrNext      =   &dwTcpwmDescr
};

/**
 * \var cy_stc_sysint_irq_t stc_sysint_irq_cfg
 * \brief IRQ DMA configuration */
/*  Configure interrupt  */
static const cy_stc_sysint_irq_t stc_sysint_irq_cfg =
{
    .sysIntSrc = DW_CH_INTR,
    .intIdx    = CPUIntIdx2_IRQn,
    .isEnabled = true,
};
```

Code Listing 15 DW0_Ch_IntHandler()

```
void DW0_Ch_IntHandler(void)
{
    uint32_t masked;

    masked = Cy_PDMA_Chnl_GetInterruptStatusMasked(DW0, DW_CHANNEL);   /*  See Code Listing
16.  */
    if ((masked & CY_PDMA_INTRCAUSE_COMPLETION) != 0ul)
    {
        /* Clear Complete DMA interrupt flag */
        Cy_PDMA_Chnl_ClearInterrupt(DW0, DW_CHANNEL);  /*See Code Listing 17.*/

        /* Read the port status */
        for(uint8_t portcnt = 0; portcnt < DST_BUFFER_SIZE; portcnt++)
        {
            g_portStatus[portcnt] = g_DestBuffer[portcnt];
        }
    }
    else
    {
        CY_ASSERT(false);
    }
}
```

Code Listing 16 Cy_SysInt_SetVector()

```
__STATIC_INLINE uint32_t Cy_PDMA_Chnl_GetInterruptStatusMasked(const volatile stc_DW_t
*pstcPDMA, uint32_t chNum)
{
    return (pstcPDMA->CH_STRUCT[chNum].unINTR_MASKED.u32Register);
}
```

Code Listing 17 Cy_PDMA_Chnl_ClearInterrupt()

```
void Cy_PDMA_Chnl_ClearInterrupt(volatile stc_DW_t *pstcPDMA, uint32_t chNum)
{
    pstcPDMA->CH_STRUCT[chNum].unINTR.u32Register = CY_PDMA_INTR_BIT_MASK;

    /* Readback of the register is required by hardware. */
    (void) pstcPDMA->CH_STRUCT[chNum].unINTR.u32Register;
}
```

Code Listing 18 Cy_Prot_ConfigPpuFixedSlaveStruct()

```
void Counter_Handler(void)
{
    if(Cy_Tcpwm_Counter_GetCC0_IntrMasked(TCPWMx_GRPx_CNTx_COUNTER))
    {
        /* Clear TCPWM CC0 interrupt flag Group#0 Counter#0 */
        Cy_Tcpwm_Counter_ClearCC0_Intr(TCPWMx_GRPx_CNTx_COUNTER);

        /* user program here.. */
    }
}
```

Code Listing 19 Cy_PDMA_Disable()

```
void Cy_PDMA_Disable(volatile stc_DW_t *pstcPDMA)
{
    pstcPDMA->unCTL.stcField.u1ENABLED  =       0ul;    /*  Write to P-DMA_CTL_ENABLED bit  */
}
```

Code Listing 20 Cy_PDMA_Chnl_DeInit()

```
void Cy_PDMA_Chnl_DeInit(volatile stc_DW_t *pstcPDMA, uint32_t chNum)
   /*  Resets P-DMA to default value  */
{
    pstcPDMA->CH_STRUCT[chNum].unCH_CTL.u32Register            = 0ul;
    pstcPDMA->CH_STRUCT[chNum].unCH_IDX.u32Register            = 0ul;
    pstcPDMA->CH_STRUCT[chNum].unCH_CURR_PTR.u32Register       = 0ul;
    pstcPDMA->CH_STRUCT[chNum].unINTR_MASK.u32Register         = 0ul;
    pstcPDMA->CH_STRUCT[chNum].unINTR_SET.u32Register          = 0ul;
}
```

Code Listing 21 Cy_PDMA_Descr_Init()

```
cy_en_pdma_status_t Cy_PDMA_Descr_Init(cy_stc_pdma_descr_t* descriptor, const
cy_stc_pdma_descr_config_t* config)
{
    cy_en_pdma_status_t retVal = CY_PDMA_ERR_UNC;

    if ((descriptor != NULL) && (config != NULL))
    {
        descriptor->unPDMA_DESCR_CTL.stcField.u2WAIT_FOR_DEACT          =       config->deact;
        descriptor->unPDMA_DESCR_CTL.stcField.u2INTR_TYPE               =       config-
>intrType;
        descriptor->unPDMA_DESCR_CTL.stcField.u2TR_OUT_TYPE             =       config-
>trigoutType;
        descriptor->unPDMA_DESCR_CTL.stcField.u2TR_IN_TYPE              =       config-
>triginType;
        descriptor->unPDMA_DESCR_CTL.stcField.u1SRC_TRANSFER_SIZE       =       config-
>srcTxfrSize;
        descriptor->unPDMA_DESCR_CTL.stcField.u1DST_TRANSFER_SIZE       =       config-
>destTxfrSize;
        descriptor->unPDMA_DESCR_CTL.stcField.u1CH_DISABLE              =       config-
>chStateAtCmplt;
        descriptor->unPDMA_DESCR_CTL.stcField.u2DATA_SIZE               =       config-
>dataSize;
        descriptor->unPDMA_DESCR_CTL.stcField.u2DESCR_TYPE              =       config-
>descrType;

        descriptor->u32PDMA_DESCR_SRC                                   =
(uint32_t)config->srcAddr;
        descriptor->u32PDMA_DESCR_DST                                   =
(uint32_t)config->destAddr;

        switch(config->descrType)
        {
        case (uint32_t)CY_PDMA_SINGLE_TRANSFER:
        {
            descriptor->unPDMA_DESCR_X_CTL.u32Register                  =       (uint32_t)config-
>descrNext;
            break;
        }
        case (uint32_t)CY_PDMA_1D_TRANSFER:
        {
            descriptor->unPDMA_DESCR_X_CTL.stcField.u12SRC_X_INCR       =       (uint32_t)config-
>srcXincr;
            descriptor->unPDMA_DESCR_X_CTL.stcField.u12DST_X_INCR       =       (uint32_t)config-
>destXincr;
            descriptor->unPDMA_DESCR_X_CTL.stcField.u8X_COUNT           =       (uint32_t)((config-
>xCount) - 1ul);
            descriptor->unPDMA_DESCR_Y_CTL.u32Register                  =       (uint32_t)config-
>descrNext;
            break;
        }
        case (uint32_t)CY_PDMA_CRC_TRANSFER:
```

```
        {
            descriptor->unPDMA_DESCR_X_CTL.stcField.u12SRC_X_INCR   =       (uint32_t)config-
>srcXincr;
            descriptor->unPDMA_DESCR_X_CTL.stcField.u12DST_X_INCR   =       0ul;
            descriptor->unPDMA_DESCR_X_CTL.stcField.u8X_COUNT       =       (uint32_t)((config-
>xCount) - 1ul);
            descriptor->unPDMA_DESCR_Y_CTL.u32Register              =       (uint32_t)config-
>descrNext;
            break;
        }
        case (uint32_t)CY_PDMA_2D_TRANSFER:
        {
            descriptor->unPDMA_DESCR_X_CTL.stcField.u12SRC_X_INCR   =       (uint32_t)config-
>srcXincr;
            descriptor->unPDMA_DESCR_X_CTL.stcField.u12DST_X_INCR   =       (uint32_t)config-
>destXincr;
            descriptor->unPDMA_DESCR_X_CTL.stcField.u8X_COUNT       =       (uint32_t)((config-
>xCount) - 1ul);

            descriptor->unPDMA_DESCR_Y_CTL.stcField.u12SRC_Y_INCR   =       (uint32_t)config-
>srcYincr;
            descriptor->unPDMA_DESCR_Y_CTL.stcField.u12DST_Y_INCR   =       (uint32_t)config-
>destYincr;
            descriptor->unPDMA_DESCR_Y_CTL.stcField.u8Y_COUNT       =       (uint32_t)((config-
>yCount) - 1ul);

            descriptor->u32PDMA_DESCR_NEXT_PTR                      =       (uint32_t)config-
>descrNext;
            break;
        }
        default:
        {
            /* Unsupported type of descriptor */
            break;
        }
        }

        retVal = CY_PDMA_SUCCESS;
    }
    else
    {
        retVal = CY_PDMA_INVALID_INPUT_PARAMETERS;
    }

    return retVal;
}
```

Code Listing 22 Cy_PDMA_Chnl_Init()

```
cy_en_pdma_status_t Cy_PDMA_Chnl_Init(volatile stc_DW_t *pstcPDMA, uint32_t chNum, const
cy_stc_pdma_chnl_config_t* chnlConfig)
{
    cy_en_pdma_status_t retVal = CY_PDMA_ERR_UNC;

    if ((pstcPDMA != NULL) && (chnlConfig != NULL))
    {
        /* Set current descriptor */
        pstcPDMA->CH_STRUCT[chNum].unCH_CURR_PTR.u32Register           =  (uint32_t)chnlConfig-
>PDMA_Descriptor;

        /* Set if the channel is preemtable */
        pstcPDMA->CH_STRUCT[chNum].unCH_CTL.stcField.u1PREEMPTABLE     =  chnlConfig-
>preemptable;

        /* Set channel priority */
        pstcPDMA->CH_STRUCT[chNum].unCH_CTL.stcField.u2PRIO            =  chnlConfig->priority;

        /* Set enabled status */
        pstcPDMA->CH_STRUCT[chNum].unCH_CTL.stcField.u1ENABLED         =  chnlConfig->enable;

        retVal = CY_PDMA_SUCCESS;
    }
    else
    {
        retVal = CY_PDMA_INVALID_INPUT_PARAMETERS;
    }

    return (retVal);
}
```

Code Listing 23 Cy_PDMA_Chnl_Enable()

```
__STATIC_INLINE void Cy_PDMA_Chnl_Enable(volatile stc_DW_t *pstcPDMA, uint32_t chNum)
{
    pstcPDMA->CH_STRUCT[chNum].unCH_CTL.stcField.u1ENABLED = 1ul;
}
```

Code Listing 24 Cy_PDMA_Chnl_SetInterruptMask()

```
__STATIC_INLINE void Cy_PDMA_Chnl_SetInterruptMask(volatile stc_DW_t *pstcPDMA, uint32_t chNum)
{
    pstcPDMA->CH_STRUCT[chNum].unINTR_MASK.u32Register = CY_PDMA_INTR_BIT_MASK;
}
```

Code Listing 25 Cy_PDMA_Enable()

```
void Cy_PDMA_Enable(volatile stc_DW_t *pstcPDMA)
{
    pstcPDMA->unCTL.stcField.u1ENABLED  =       1ul;    /*  Write to P-DMA_CTL_ENABLED bit  */
}
```

3.3 ディスクリプタチェイン

3.3.1 概要

ここでは、現在のディスクリプタに次のディスクリプタのポインタ (DESCR_NEXT_PTR) を格納しディスクリプタを連結させる例を説明します。

この例では、ディスクリプタ 0 とディスクリプタ 1 という 2 つのディスクリプタを使用します。両方のディスクリプタは、TCPWM からの定期的なトリガを使用して複数のポート入力状態をメモリ (RAM) に転送する 1D Transfer 用です。ただし、ディスクリプタ 0 は転送先アドレスとして RAM ベースアドレス 0 を持ち、ディスクリプタ 1 は転送先アドレスとして RAM ベースアドレス 1 を持ちます。

P-DMA は、TCPWM からの定期的なトリガを使用して、複数のポート入力状態をメモリ (RAM) に転送します。P-DMA は、転送が完了時に CPU に割込みを通知します。この転送のディスクリプタ (ディスクリプタ 0) の次のディスクリプタへのポインタ (DESCR_NEXT_PTR) は、別のディスクリプタ (ディスクリプタ 1) へのポインタに設定されます。結果として P-DMA は、ディスクリプタ 0 の転送が完了後、ディスクリプタ 1 の転送を開始できます。ディスクリプタ 0 とディスクリプタ 1 は異なる転送先アドレスを持ちます。

P-DMA では、同じポートアドレスのデータを異なる RAM アドレスに転送できます。言い換えれば、ダブルバッファを構成できます。この場合、ディスクリプタ 0 とディスクリプタ 1 は互いに連結され、ディスクリプタ 0 とディスクリプタ 1 は循環リストを持ちます。

図 13 に、ディスクリプタチェインを使用した 1D Transfer を示します。

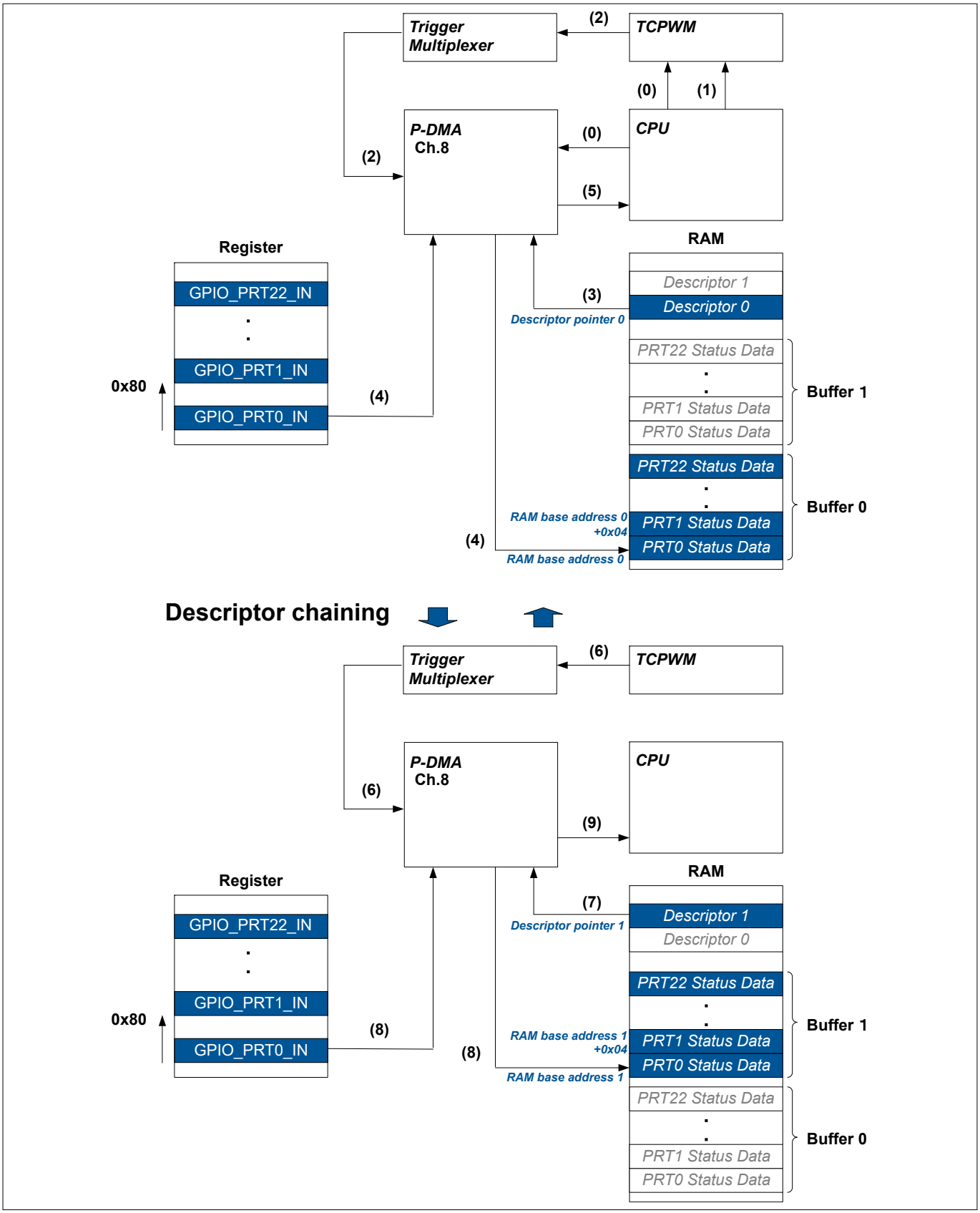


図 13　　ディスクリプタチェインを使用した 1D Transfer

(0) ユースケースに従って P-DMA を設定してください。さらに、TCPWM と Trigger multiplexer を設定してください。

(1) CPU が TCPWM タイマを起動します。

(2) TCPWM は、指定したカウント値 (オーバフロー) に達すると、Trigger multiplexer を介して P-DMA にトリガを出力します。

(3) 次の保留された転送を起動時に、P-DMA は指定された領域 (ディスクリプタポインタ 0) からディスクリプタを読み出します。

(4)P-DMA は、転送元アドレス (GPIO_PRT0_IN) からデータを読み出し、読み出したデータを転送先アドレス (RAM ベースアドレス 0) に書き込みます。その後、P-DMA は転送元アドレスを 0x80 だけインクリメントし、転送先アドレスは 0x04 だけインクリメントします。次に P-DMA は、転送元アドレス (GPIO_PRT1_IN) からデータを読み出し、転送先アドレス (RAM ベースアドレス 0 + 0x04) に再度書き込みます。

(5) すべての転送が完了すると P-DMA は、CPU に割込みを通知します。

(6) TCPWM は指定したカウント値 (オーバフロー) に再度到達すると、P-DMA にトリガを出力します。

(7) 転送を受け付け時、P-DMA はディスクリプタ (ディスクリプタポインタ 1) 内の次のディスクリプタポインタによって設定されたポインタからディスクリプタを読み出します。

(8) P-DMA は転送元アドレス (GPIO_PRT0_IN) からデータを読み出し、読み出したデータを転送先アドレス (RAM ベースアドレス 1) に書き込みます。その後、P-DMA は転送元アドレスを 0x80 だけインクリメントし、転送先アドレスは 0x04 だけインクリメントします。次に、P-DMA は、転送元アドレス (GPIO_PRT1_IN) からデータを読み出し、転送先アドレス (RAM ベースアドレス 1 + 0x04) に再度書き込みます。

(9) P-DMA は、すべての転送を完了後に割込みを CPU に通知します。ディスクリプタ 1 はディスクリプタ 0 に連結されます。したがって、TCPWM が再びトリガを出力すると、(3) からの手順を繰り返します。

***注**: TCPWM のカウント時間は、すべてのデータを転送するために DMA が必要とする時間よりも長くなければいけません。*

3.3.2 初期設定

ここでは、このユースケースの P-DMA チャネルとディスクリプタの初期化について説明します。

図 14 に P-DMA の設定手順を示します。

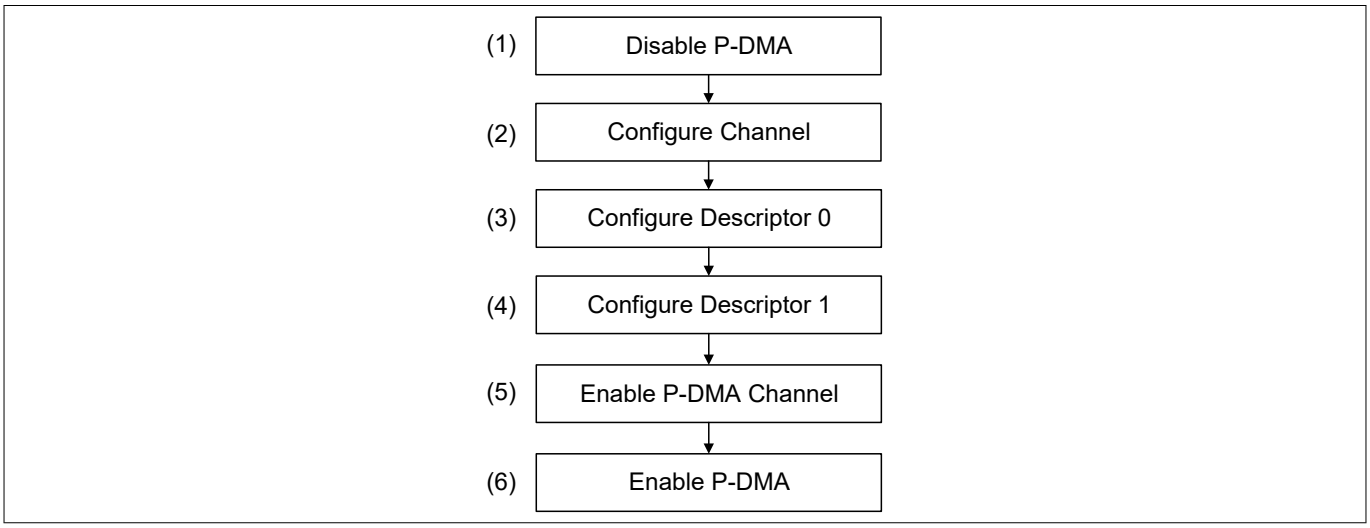


図 14 P-DMA の設定手順

3.3.3 設定とサンプルコード

表 11 は DMA 用の SDL におけるパラメータの一覧、表 12 は設定部の機能の一覧です。

表 11 DMA 設定パラメータの一覧

パラメータ	説明	値
TCPWMx_GRPx_CNTx_COUNTER	カウンタを使用した TCPWM の定義	TCPWM0_GRP0_CNT0
DW_CHANNEL	P-DMA チャネルの定義	8ul
TCPWM_TO_MUX_TRIG	TCPWM からマルチプレクサへのトリガの定義	TRIG_IN_MUX_3_TCPWM_16_TR_OUT00
MUX_TO_PDMA_TRIG	P-DMA に対するトリガマルチプレクサの定義	TRIG_OUT_MUX_3_PDMA0_TR_IN8
DW_CH_INTR	P-DMA チャネル割込みの定義	cpuss_interrupts_dw0_8_IRQn
DST_BUFFER_SIZE	転送先バッファ サイズの定義	23ul
DST_BUFFER_NUMBER	転送先バッファ番号の定義	2ul
.PDMA_Descriptor	P-DMA の現在のディスクリプタポインタ	dwTcpwmDescr
.preemptable	チャネル プリエンプタブル	0ul
.priority	チャネル優先度	0ul
.enable	チャネル イネーブル	1ul
.deact	DESCR_CTL WAIT_FOR_DEACT	0ul
.intrType	DESCR_CTL INTR_TYPE	CY_PDMA_INTR_X_LOOP_CMPLT
.trigoutType	DESCR_CTL TR_OUT_TYPE	CY_PDMA_TRIGOUT_X_LOOP_CMPLT
.chStateAtCmplt	DESCR_CTL CH_DISABLE	CY_PDMA_CH_ENABLED
.triginType	DESCR_CTL TR_IN_TYPE	CY_PDMA_TRIGIN_DESCR
.dataSize	DESCR_CTL DATA_SIZE	CY_PDMA_WORD
.srcTxfrSize	DESCR_CTL SRC_TRANSFER_SIZE	0ul
.destTxfrSize	DESCR_CTL DST_TRANSFER_SIZE	0ul
.descrType	DESCR_CTL DESCR_TYPE	CY_PDMA_1D_TRANSFER
.srcAddr	DESCR_SRC	(void *)&GPIO_PRT0->unIN.u32Register
.destAddr	DESCR_DST	(void *)g_DestBuffer0 .srcXincr
.srcXincr	DESCR_X_CTL SRC_X_INCR	32ul
.destXincr	DESCR_X_CTL DST_X_INCR	1ul
.xCount	DESCR_X_CTL X_COUNT	DST_BUFFER_SIZE
.descrNext	DESCR_NEXT_PTR	NULL

表 12 DMA 設定機能の一覧

機能	説明	備考
DW0_Ch_IntHandler()	P-DMA0 割込み用ハンドラ	Code Listing 28 を参照してください

(続く)

表 12 (続き) DMA 設定機能の一覧

機能	説明	備考
Counter_Handler()	TCPWM カウンタ割込み用ハンドラ	
Cy_SysInt_InitIRQ()	割込みの設定	-
Cy_SysInt_SetSystemIrqVector()	割込みベクタの設定	-
NVIC_SetPriority()	NVIC セット優先度	-
NVIC_EnableIRQ()	NVIC イネーブル割込み	-
Cy_PDMA_Disable()	P-DMA ディセーブルの設定	P-DMA_CTL_ENABLED ビットに書き込みます。 Code Listing 32 を参照してください
Cy_PDMA_Chnl_DeInit()	P-DMA をデフォルト値にリセット	チャネルに対応するレジスタの内容をすべてクリアします。 Code Listing 33 を参照してください
Cy_PDMA_Descr_Init()	P-DMA ディスクリプタ初期化の設定	Code Listing 34 を参照してください
Cy_PDMA_Chnl_Init()	P-DMA チャネル初期化の設定	Code Listing 35 を参照してください
Cy_PDMA_Chnl_Enable()	P-DMA チャネル イネーブルの設定	Code Listing 36 を参照してください
Cy_PDMA_Chnl_SetInterruptMask()	P-DMA チャネル割込みマスクの設定	Code Listing 37 を参照してください
Cy_PDMA_Enable()	P-DMA イネーブルの設定	P-DMA_CTL_ENABLED ビットに書き込みます。 Code Listing 38 を参照してください
Cy_TrigMux_Connect()	トリガ マルチプレクサの初期化	-
Cy_Tcpwm_TriggerStart()	TCPWM 開始	--
Cy_PDMA_Chnl_GetInterruptStatusMasked()	1 回のロードで、対応する INTR と INTR_MASK フィールドの論理和を返します。	Code Listing 29 を参照してください
Cy_PDMA_Chnl_ClearInterrupt()	P-DMA チャネルが割込みステータスをクリアするように設定	Code Listing 30 を参照してください
Cy_Tcpwm_Counter_ClearCC0_Intr()	TCPWM CC0 割込みフラグをクリア	-

Code Listing 26 は、ディスクリプタチェインのサンプルプログラムを示します。GPIO, TCPWM, およびクロックの設定については、Architecture TRM、およびアプリケーションノートを参照してください。

Code Listing 26 ディスクリプタチェインの例

```
#define TCPWMx_GRPx_CNTx_COUNTER            TCPWM0_GRP0_CNT0   /*Define TCPWM using counter*/

#define DW_CHANNEL                          8ul
#define TCPWM_TO_MUX_TRIG                   TRIG_IN_MUX_3_TCPWM_16_TR_OUT00
   /*  Define P-DMA channel, Trigger, P-DMA channel interrupt   */
#define MUX_TO_PDMA_TRIG                    TRIG_OUT_MUX_3_PDMA0_TR_IN8
#define DW_CH_INTR                          cpuss_interrupts_dw0_8_IRQn
   /*  Define DST buffer size, DST buffer number  */
#define DST_BUFFER_SIZE                     23ul
#define DST_BUFFER_NUMBER                   2ul

void DW0_Ch_IntHandler(void);    /*  See Code Listing 28  */
void Counter_Handler(void);    /*  See Code Listing 31  */

int main(void)
{
:

    /* Initialize & Enable DMA */
    Cy_PDMA_Disable(DW0);
    /*  (1) Disable P-DMA. See Code Listing 32.
(2) Resets P-DMA to default values. See Code Listing 33.  */
    Cy_PDMA_Chnl_DeInit(DW0, DW_CHANNEL);

    for(uint32_t i = 0ul; i < DST_BUFFER_NUMBER; i++)
    {
        if((i + 1ul) == DST_BUFFER_NUMBER)
        {
            /* Last descriptor */
             /*  (4) Configures P-DMA descriptor 1. See Code Listing 34.  */
            dw0ChTcpwmConfig.chStateAtCmplt = CY_PDMA_CH_ENABLED;
            dw0ChTcpwmConfig.destAddr       = &g_DestBuffer1[0];
            dw0ChTcpwmConfig.descrNext      = &dwTcpwmDescr[i - (DST_BUFFER_NUMBER-1)];
        }
        else
        {
            /*  (3) Configures P-DMA descriptor 0. See Code Listing 34.  */
            dw0ChTcpwmConfig.chStateAtCmplt = CY_PDMA_CH_ENABLED;
            dw0ChTcpwmConfig.destAddr       = &g_DestBuffer0[0];
            dw0ChTcpwmConfig.descrNext      = &dwTcpwmDescr[i + 1ul];
        }
        Cy_PDMA_Descr_Init(&dwTcpwmDescr[i], &dw0ChTcpwmConfig);
    }
    Cy_PDMA_Chnl_Init(DW0, DW_CHANNEL, &dwTcpwmConfig);
    Cy_PDMA_Chnl_Enable(DW0, DW_CHANNEL);
   /*  (5) Enable P-DMA channel. See Code Listing 36. (6) Enable P-DMA. Code Listing 38.  */
    Cy_PDMA_Chnl_SetInterruptMask(DW0, DW_CHANNEL);
    Cy_PDMA_Enable(DW0);

    /* Trigger Multiplexer Initialization */
    /* TCPWM To DW */
```

```
    Cy_TrigMux_Connect(TCPWM_TO_MUX_TRIG, MUX_TO_PDMA_TRIG, CY_TR_MUX_TR_INV_DISABLE,
TRIGGER_TYPE_LEVEL, 0ul);

    /* Interrupt Initialization */
    Cy_SysInt_InitIRQ(&stc_sysint_irq_cfg);
    /*  Configures Interrupt   */
    Cy_SysInt_SetSystemIrqVector(stc_sysint_irq_cfg.sysIntSrc, DW0_Ch_IntHandler);
    NVIC_SetPriority(stc_sysint_irq_cfg.intIdx, 0ul);
    NVIC_EnableIRQ(stc_sysint_irq_cfg.intIdx);

    /* TCPWM Start */
    Cy_Tcpwm_TriggerStart(TCPWMx_GRPx_CNTx_COUNTER);    /*  TCPWM start  */

    for(;;);
}
```

Code Listing 27 P-DMA チャネル、ディスクリプタの設定

```
/**
 * \var cy_stc_pdma_descr_t dwTcpwmDescr
 * \brief PDMA descriptor
 */
static  cy_stc_pdma_descr_t         dwTcpwmDescr[DST_BUFFER_NUMBER];

/**
 * \var cy_stc_pdma_chnl_config_t dwTcpwmConfig
 * \brief PDMA configuration */
   /*  Configure P-DMA channel  */
const   cy_stc_pdma_chnl_config_t   dwTcpwmConfig =
{
    .PDMA_Descriptor = dwTcpwmDescr,
    .preemptable     = 0ul,
    .priority        = 0ul,
    .enable          = 1ul,
};

/**
 * \var cy_stc_pdma_descr_config_t dw0ChTcpwmConfig
 * \brief PDMA descriptor configuration */
static  cy_stc_pdma_descr_config_t  dw0ChTcpwmConfig =
   /*  Configure P-DMA descriptor  */
{
    .deact          =   0ul,
    .intrType       =   CY_PDMA_INTR_X_LOOP_CMPLT,
    .trigoutType    =   CY_PDMA_TRIGOUT_X_LOOP_CMPLT,
    .chStateAtCmplt =   CY_PDMA_CH_ENABLED,
    .triginType     =   CY_PDMA_TRIGIN_DESCR,
    .dataSize       =   CY_PDMA_WORD,
    .srcTxfrSize    =   0ul, /* Same as dataSize */
    .destTxfrSize   =   0ul, /* Same as dataSize */
    .descrType      =   CY_PDMA_1D_TRANSFER,
    .srcAddr        =   (void *)&GPIO_PRT0->unIN.u32Register,
    .destAddr       =   (void *)g_DestBuffer0,
    .srcXincr       =   32ul, /* address +80h */
    .destXincr      =   1ul,  /* address +04h */
    .xCount         =   DST_BUFFER_SIZE,
    .descrNext      =   NULL /* will be updated in run time */
};

/**
 * \var cy_stc_sysint_irq_t stc_sysint_irq_cfg
 * \brief IRQ DMA configuration */
static const cy_stc_sysint_irq_t stc_sysint_irq_cfg =
   /*  Configure Interrupt  */
{
    .sysIntSrc = DW_CH_INTR,
    .intIdx    = CPUIntIdx2_IRQn,
```

```
    .isEnabled = true,
};
```

Code Listing 28 DW0_Ch_IntHandler()

```
void DW0_Ch_IntHandler(void)
{
    uint32_t masked;

    masked = Cy_PDMA_Chnl_GetInterruptStatusMasked(DW0, DW_CHANNEL); /*See Code Listing 29.*/
    if ((masked & CY_PDMA_INTRCAUSE_COMPLETION) != 0ul)
    {
        /* Clear Complete DMA interrupt flag */
        Cy_PDMA_Chnl_ClearInterrupt(DW0, DW_CHANNEL);   /*  See Code Listing 30.  */

        /* Read the port status */
        for(uint8_t portcnt = 0; portcnt < DST_BUFFER_SIZE; portcnt++)
        {
            g_portStatus[portcnt] = g_DestBuffer[portcnt];
        }
    }
    else
    {
        CY_ASSERT(false);
    }
}
```

Code Listing 29 Cy_PDMA_Chnl_GetInterruptStatusMasked()

```
__STATIC_INLINE uint32_t Cy_PDMA_Chnl_GetInterruptStatusMasked(const volatile stc_DW_t
*pstcPDMA, uint32_t chNum)
{
    return (pstcPDMA->CH_STRUCT[chNum].unINTR_MASKED.u32Register);
}
```

Code Listing 30 Cy_PDMA_Chnl_ClearInterrupt()

```
void Cy_PDMA_Chnl_ClearInterrupt(volatile stc_DW_t *pstcPDMA, uint32_t chNum)
{
    pstcPDMA->CH_STRUCT[chNum].unINTR.u32Register = CY_PDMA_INTR_BIT_MASK;

    /* Readback of the register is required by hardware. */
    (void) pstcPDMA->CH_STRUCT[chNum].unINTR.u32Register;
}
```

Code Listing 31 Counter_Handler()

```
void Counter_Handler(void)
{
    if(Cy_Tcpwm_Counter_GetCC0_IntrMasked(TCPWMx_GRPx_CNTx_COUNTER))
    {
        /* Clear TCPWM CC0 interrupt flag Group#0 Counter#0 */
        Cy_Tcpwm_Counter_ClearCC0_Intr(TCPWMx_GRPx_CNTx_COUNTER);

        /* user program here.. */
    }
}
```

Code Listing 32 Cy_PDMA_Disable()

```
void Cy_PDMA_Disable(volatile stc_DW_t *pstcPDMA)
{
    pstcPDMA->unCTL.stcField.u1ENABLED  =       0ul;    /*  Write to P-DMA_CTL_ENABLED bit  */
}
```

Code Listing 33 Cy_PDMA_Chnl_DeInit()

```
void Cy_PDMA_Chnl_DeInit(volatile stc_DW_t *pstcPDMA, uint32_t chNum)
   /*  Resets PDMA to default value  */
{
    pstcPDMA->CH_STRUCT[chNum].unCH_CTL.u32Register            = 0ul;
    pstcPDMA->CH_STRUCT[chNum].unCH_IDX.u32Register           = 0ul;
    pstcPDMA->CH_STRUCT[chNum].unCH_CURR_PTR.u32Register      = 0ul;
    pstcPDMA->CH_STRUCT[chNum].unINTR_MASK.u32Register        = 0ul;
    pstcPDMA->CH_STRUCT[chNum].unINTR_SET.u32Register         = 0ul;
```

Code Listing 34 Cy_PDMA_Descr_Init()

```
cy_en_pdma_status_t Cy_PDMA_Descr_Init(cy_stc_pdma_descr_t* descriptor, const
cy_stc_pdma_descr_config_t* config)
{
    cy_en_pdma_status_t retVal = CY_PDMA_ERR_UNC;

    if ((descriptor != NULL) && (config != NULL))
    {
        descriptor->unPDMA_DESCR_CTL.stcField.u2WAIT_FOR_DEACT          =       config->deact;
        descriptor->unPDMA_DESCR_CTL.stcField.u2INTR_TYPE               =       config-
>intrType;
        descriptor->unPDMA_DESCR_CTL.stcField.u2TR_OUT_TYPE             =       config-
>trigoutType;
        descriptor->unPDMA_DESCR_CTL.stcField.u2TR_IN_TYPE              =       config-
>triginType;
        descriptor->unPDMA_DESCR_CTL.stcField.u1SRC_TRANSFER_SIZE       =       config-
>srcTxfrSize;
        descriptor->unPDMA_DESCR_CTL.stcField.u1DST_TRANSFER_SIZE       =       config-
>destTxfrSize;
        descriptor->unPDMA_DESCR_CTL.stcField.u1CH_DISABLE              =       config-
>chStateAtCmplt;
        descriptor->unPDMA_DESCR_CTL.stcField.u2DATA_SIZE               =       config-
>dataSize;
        descriptor->unPDMA_DESCR_CTL.stcField.u2DESCR_TYPE              =       config-
>descrType;

        descriptor->u32PDMA_DESCR_SRC                                   =
(uint32_t)config->srcAddr;
        descriptor->u32PDMA_DESCR_DST                                   =
(uint32_t)config->destAddr;

        switch(config->descrType)
        {
        case (uint32_t)CY_PDMA_SINGLE_TRANSFER:
        {
            descriptor->unPDMA_DESCR_X_CTL.u32Register                  =       (uint32_t)config-
>descrNext;
            break;
        }
        case (uint32_t)CY_PDMA_1D_TRANSFER:
        {
            descriptor->unPDMA_DESCR_X_CTL.stcField.u12SRC_X_INCR       =       (uint32_t)config-
>srcXincr;
            descriptor->unPDMA_DESCR_X_CTL.stcField.u12DST_X_INCR       =       (uint32_t)config-
>destXincr;
            descriptor->unPDMA_DESCR_X_CTL.stcField.u8X_COUNT           =       (uint32_t)((config-
>xCount) - 1ul);
            descriptor->unPDMA_DESCR_Y_CTL.u32Register                  =       (uint32_t)config-
>descrNext;
            break;
        }
        case (uint32_t)CY_PDMA_CRC_TRANSFER:
```

```
        {
            descriptor->unPDMA_DESCR_X_CTL.stcField.u12SRC_X_INCR   =       (uint32_t)config-
>srcXincr;
            descriptor->unPDMA_DESCR_X_CTL.stcField.u12DST_X_INCR   =       0ul;
            descriptor->unPDMA_DESCR_X_CTL.stcField.u8X_COUNT       =       (uint32_t)((config-
>xCount) - 1ul);
            descriptor->unPDMA_DESCR_Y_CTL.u32Register              =       (uint32_t)config-
>descrNext;
            break;
        }
        case (uint32_t)CY_PDMA_2D_TRANSFER:
        {
            descriptor->unPDMA_DESCR_X_CTL.stcField.u12SRC_X_INCR   =       (uint32_t)config-
>srcXincr;
            descriptor->unPDMA_DESCR_X_CTL.stcField.u12DST_X_INCR   =       (uint32_t)config-
>destXincr;
            descriptor->unPDMA_DESCR_X_CTL.stcField.u8X_COUNT       =       (uint32_t)((config-
>xCount) - 1ul);

            descriptor->unPDMA_DESCR_Y_CTL.stcField.u12SRC_Y_INCR   =       (uint32_t)config-
>srcYincr;
            descriptor->unPDMA_DESCR_Y_CTL.stcField.u12DST_Y_INCR   =       (uint32_t)config-
>destYincr;
            descriptor->unPDMA_DESCR_Y_CTL.stcField.u8Y_COUNT       =       (uint32_t)((config-
>yCount) - 1ul);

            descriptor->u32PDMA_DESCR_NEXT_PTR                      =       (uint32_t)config-
>descrNext;
            break;
        }
        default:
        {
            /* Unsupported type of descriptor */
            break;
        }
        }

        retVal = CY_PDMA_SUCCESS;
    }
    else
    {
        retVal = CY_PDMA_INVALID_INPUT_PARAMETERS;
    }

    return retVal;
}
```

Code Listing 35 Cy_PDMA_Chnl_Init()

```
cy_en_pdma_status_t Cy_PDMA_Chnl_Init(volatile stc_DW_t *pstcPDMA, uint32_t chNum, const
cy_stc_pdma_chnl_config_t* chnlConfig)
{
    cy_en_pdma_status_t retVal = CY_PDMA_ERR_UNC;

    if ((pstcPDMA != NULL) && (chnlConfig != NULL))
    {
        /* Set current descriptor */
        pstcPDMA->CH_STRUCT[chNum].unCH_CURR_PTR.u32Register           =  (uint32_t)chnlConfig-
>PDMA_Descriptor;

        /* Set if the channel is preemtable */
        pstcPDMA->CH_STRUCT[chNum].unCH_CTL.stcField.u1PREEMPTABLE     =  chnlConfig-
>preemptable;

        /* Set channel priority */
        pstcPDMA->CH_STRUCT[chNum].unCH_CTL.stcField.u2PRIO            =  chnlConfig->priority;

        /* Set enabled status */
        pstcPDMA->CH_STRUCT[chNum].unCH_CTL.stcField.u1ENABLED         =  chnlConfig->enable;

        retVal = CY_PDMA_SUCCESS;
    }
    else
    {
        retVal = CY_PDMA_INVALID_INPUT_PARAMETERS;
    }

    return (retVal);
}
```

Code Listing 36 Cy_PDMA_Chnl_Enable()

```
__STATIC_INLINE void Cy_PDMA_Chnl_Enable(volatile stc_DW_t *pstcPDMA, uint32_t chNum)
{
    pstcPDMA->CH_STRUCT[chNum].unCH_CTL.stcField.u1ENABLED = 1ul;
}
```

Code Listing 37 Cy_PDMA_Chnl_SetInterruptMask()

```
__STATIC_INLINE void Cy_PDMA_Chnl_SetInterruptMask(volatile stc_DW_t *pstcPDMA, uint32_t chNum)
{
    pstcPDMA->CH_STRUCT[chNum].unINTR_MASK.u32Register = CY_PDMA_INTR_BIT_MASK;
}
```

Code Listing 38 Cy_PDMA_Enable()

```
void Cy_PDMA_Enable(volatile stc_DW_t *pstcPDMA)
{
    pstcPDMA->unCTL.stcField.u1ENABLED  =       1ul;    /*  Write to P-DMA_CTL_ENABLED bit  */
}
```

3.4 2D Transfer (ペリフェラルからメモリ)

3.4.1 概要

この例は、ADC グループ変換の結果がどのようにメモリに格納されるかを示します。詳細は Architecture TRM の「SAR ADC」の章を参照してください。

変換結果はチャネルごとにグループ化され、メモリに格納されます。この例では、ADC は指定した周期で 3 つのチャネル CH_0, CH_1, および CH_2 をグループ変換で変換します。P-DMA は、ADC 変換完了トリガを使用して、3 つのグループ変換結果を各チャネルのメモリに格納します。

図 15 に、2D transfer の転送動作を示します。

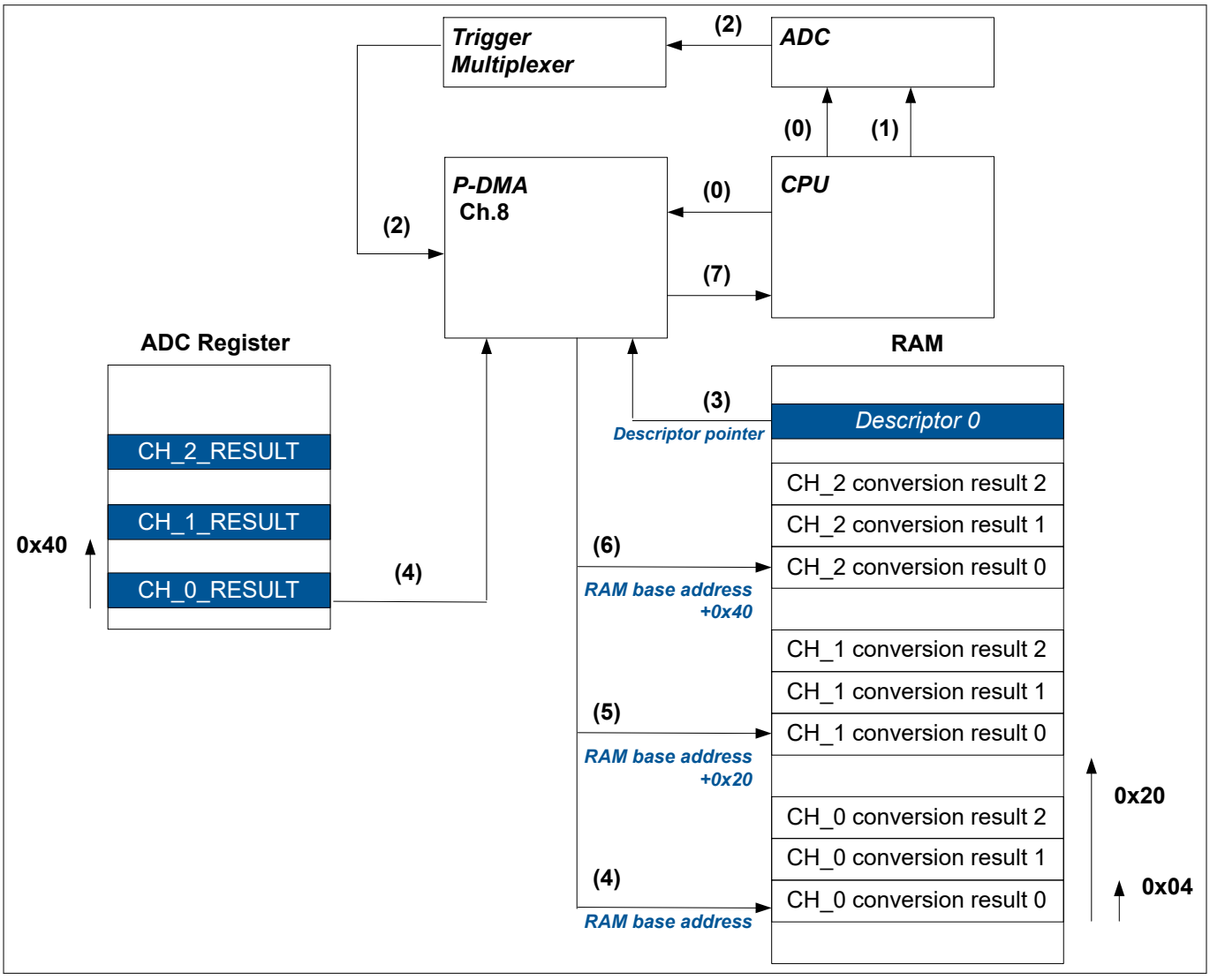


図 15 2D Transfer (ペリフェラルからメモリ)

(0) ユースケースに従って P-DMA を設定してください。さらに、ADC とトリガマルチプレクサを設定してください。

(1) CPU が ADC グループ変換を起動します。

(2) グループ変換完了後、ADC は Trigger multiplexer を介して P-DMA にトリガを出力します。

(3) P-DMA は、転送を受け付け時に、指定された領域 (ディスクリプタポインタ) からディスクリプタを読み出します。

(4) P-DMA は、転送元アドレス (CH_0_RESULT) からデータを読み出し、読み出したデータを転送先アドレス (RAM ベースアドレス) に書き込みます。

(5) P-DMA は転送元アドレスを 0x40、転送先アドレスを 0x20 でインクリメントします。次に、P-DMA は、転送元アドレス (CH_1_RESULT) からデータを読み取り、それを転送先アドレス (RAM ベースアドレス 0 + 0x20) に書き込みます。

(6) P-DMA は転送元アドレスを 0x40、転送先アドレスを 0x20 でインクリメントします。次に P-DMA は、転送元アドレス (CH_2_RESULT) からデータを読み出し、読出しデータを転送先アドレス (RAM ベースアドレス 0 + 0x40) に書き込みます。次のトリガが発生すると、P-DMA は RAM ベースアドレスを 0x04 だけインクリメントし、(4) からの手順を繰り返します。

(7) P-DMA は、CH_2 変換結果 2 を転送後、CPU に割込みを通知します。

この場合、ディスクリプタを完成させるには 3 つのトリガが必要です。

3.4.2 初期設定

ここでは、このユースケースの P-DMA チャネルとディスクリプタの初期化について説明します。

図 16 に P-DMA の設定手順を示します。

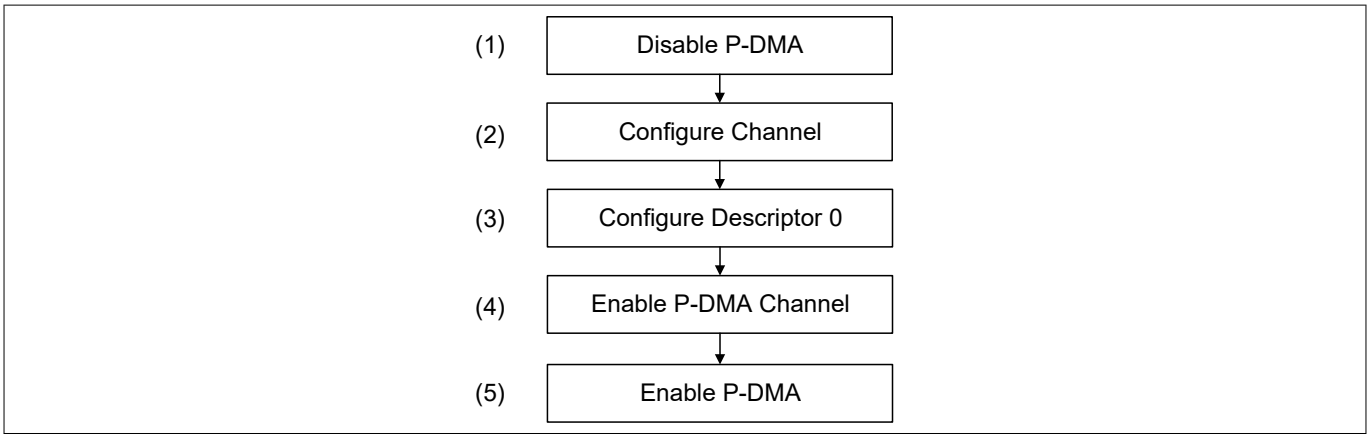


図 16 P-DMA の設定手順

3.4.3 設定とサンプルコード

表 13 は DMA 用の SDL におけるパラメータの一覧、表 14 は設定部の機能の一覧です。

表 13 DMA 設定パラメータの一覧

パラメータ	説明	値
ADC_MACRO	ADC マクロの定義	PASS0_SAR0
ADC_GROUP_NUMBER_OF_CHANNELS	ADC グループ番号の定義	3ul
ADC_LOGICAL_CHANNEL	ADC 論理チャネルの定義	0ul

(続く)

表 13 (続き) DMA 設定パラメータの一覧

パラメータ	説明	値
ADC_GROUP_FIRST_CHANNEL	ADC グループ先頭チャネルの定義	ADC_LOGICAL_CHANNEL
ADC_GROUP_LAST_CHANNEL	ADC グループ最終チャネルの定義	(ADC_GROUP_FIRST_CHANNEL + ADC_GROUP_NUMBER_OF_CHANNELS - 1ul)
ADC_CH_INTR_BASE	ADC チャネル割込みの定義	pass_0_interrupts_sar_0_IRQn
BUFFER_SIZE_IN_WORD	バッファ サイズの定義	25ul
BUFFER_SIZE_IN_BYTE	バッファ サイズの定義	(BUFFER_SIZE_IN_WORD * 4ul)
DW_CHANNEL	P-DMA チャネルの定義	27ul
DW_CH_INTR	P-DMA チャネル割込みの定義	cpuss_interrupts_dw0_27_IRQn
ADC_TO_DMA_TRIG	ADC から DMA へのトリガの定義	TRIG_OUT_1TO1_2_PASS_CH_DONE_TO_PDMA02
.PDMA_Descriptor	P-DMA の現在のディスクリプタポインタ	&
.preemptable	チャネル プリエンプタブル	0ul
.priority	チャネル優先度	0ul
.enable	チャネル イネーブル	1ul
.deact	DESCR_CTL WAIT_FOR_DEACT	0ul
.intrType	DESCR_CTL INTR_TYPE	CY_PDMA_INTR_DESCR_CMPLT
.trigoutType	DESCR_CTL TR_OUT_TYPE	CY_PDMA_TRIGOUT_DESCR_CMPLT
.chStateAtCmplt	DESCR_CTL CH_DISABLE	CY_PDMA_CH_ENABLED
.triginType	DESCR_CTL TR_IN_TYPE	CY_PDMA_TRIGIN_XLOOP
.dataSize	DESCR_CTL DATA_SIZE	CY_PDMA_WORD
.srcTxfrSize	DESCR_CTL SRC_TRANSFER_SIZE	0ul, /* as specified in DATA_SIZE */
.destTxfrSize	DESCR_CTL DST_TRANSFER_SIZE	0ul, /* as specified in DATA_SIZE */
.descrType	DESCR_CTL DESCR_TYPE	CY_PDMA_2D_TRANSFER
.srcAddr	DESCR_SRC	(void *)&ADC_MACRO->CH[ADC_GROUP_FIRST_CHANNEL].unRESULT.u32Register
.destAddr	DESCR_DST	(void *)g_DestBuffer
.srcXincr	DESCR_X_CTL SRC_X_INCR	16ul
.destXincr	DESCR_X_CTL DST_X_INCR	8ul
.xCount	DESCR_X_CTL X_COUNT	ADC_GROUP_NUMBER_OF_CHANNELS
.srcYincr	DESCR_Y_CTL SRC_Y_INCR	0ul
.destYincr	DESCR_Y_CTL DST_Y_INCR	1ul

(続く)

表 13 (続き) DMA 設定パラメータの一覧

パラメータ	説明	値
.yCount	DESCR_Y_CTL Y_COUNT	3ul
.descrNext	DESCR_NEXT_PTR	&stcDescr

表 14 DMA 設定機能の一覧

機能	説明	備考
DW0_Ch_IntHandler()	P-DMA0 割込み用ハンドラ	Code Listing 41 を参照してください
Cy_SysInt_InitIRQ()	割込みの設定	-
Cy_SysInt_SetSystemIrqVector()	割込みベクタの設定	-
NVIC_SetPriority()	NVIC セット プライオリティ	-
NVIC_EnableIRQ()	NVIC イネーブル割込み	-
Cy_PDMA_Disable()	P-DMA ディセーブルの設定	P-DMA_CTL_ENABLED ビットに書き込みます。 Code Listing 44 を参照してください
Cy_PDMA_Chnl_DeInit()	P-DMA をデフォルト値にリセット	Code Listing 45 を参照してください
Cy_PDMA_Descr_Init()	P-DMA ディスクリプタ初期化の設定	Code Listing 46 を参照してください
Cy_PDMA_Chnl_Init()	P-DMA チャネル初期化の設定	Code Listing 47 を参照してください
Cy_PDMA_Chnl_Enable()	P-DMA チャネル イネーブルの設定	Code Listing 48 を参照してください
Cy_PDMA_Chnl_SetInterruptMask()	P-DMA チャネル割込みマスクの設定	Code Listing 49 を参照してください
Cy_PDMA_Enable()	P-DMA イネーブルの設定	P-DMA_CTL_ENABLED ビットに書き込みます。 Code Listing 50 を参照してください
Cy_TrigMux_Connect1To1()	トリガ マルチプレクサの初期化	-
Cy_PDMA_Chnl_GetInterruptStatusMasked()	1 回のロードで、対応する INTR と INTR_MASK フィールドの論理和を返します。	Code Listing 42 を参照してください
Cy_PDMA_Chnl_ClearInterrupt()	P-DMA チャネルが割込みステータスをクリアするように設定	Code Listing 43 を参照してください
Cy_Adc_Channel_SoftwareTrigger()	ADC ソフトウェア開始トリガ	-

Code Listing 39 は、2D Transfer (ペリフェラルからメモリ) するプログラム例を示します。GPIO, ADC, およびクロックの設定については、Architecture TRM およびアプリケーションノートを参照してください。

Code Listing 39 2D Transfer (ペリフェラルからメモリ) 例

```
#define ADC_MACRO                   PASS0_SAR0

#define ADC_GROUP_NUMBER_OF_CHANNELS (3ul)    /*  Define ADC macro, ADC group number   */



/* ADC logical channel to be used */
#define ADC_LOGICAL_CHANNEL         0ul
/*  Define ADC logical channel, ADC group first channel, ADC group last channel,  ADC channel
interrupt   */
#define ADC_GROUP_FIRST_CHANNEL     ADC_LOGICAL_CHANNEL
#define ADC_GROUP_LAST_CHANNEL      (ADC_GROUP_FIRST_CHANNEL + ADC_GROUP_NUMBER_OF_CHANNELS -
1ul)
#define ADC_CH_INTR_BASE            pass_0_interrupts_sar_0_IRQn

#define BUFFER_SIZE_IN_WORD         25ul
/*  Define buffer size, P-DMA channel, P-DMA channel interrupt, Trigger  */
#define BUFFER_SIZE_IN_BYTE         (BUFFER_SIZE_IN_WORD * 4ul)
#define DW_CHANNEL                  27ul
#define DW_CH_INTR                  cpuss_interrupts_dw0_27_IRQn
#define ADC_TO_DMA_TRIG             TRIG_OUT_1TO1_2_PASS_CH_DONE_TO_PDMA02

void DW0_Ch_IntHandler(void);   /*  See Code Listing 41.  */


int main(void)
{
:
    /*  (1) Disable P-DMA. See Code Listing 44.
(2) Resets P-DMA to default values. See Code Listing 45.
(3) Configures P-DMA descriptor initialize.  See Code Listing 46.
(4) Enable P-DMA channel. See Code Listing 47.
(5) Enable P-DMA. See Code Listing 50.  */
    /* Initialize & Enable DW */
    Cy_PDMA_Disable(DW0);
    Cy_PDMA_Chnl_DeInit(DW0, DW_CHANNEL);
    Cy_PDMA_Descr_Init(&stcDescr,&stcDmaDescrConfig);
    Cy_PDMA_Chnl_Init(DW0, DW_CHANNEL, &chnlConfig);
    Cy_PDMA_Chnl_Enable(DW0, DW_CHANNEL);
    Cy_PDMA_Chnl_SetInterruptMask(DW0, DW_CHANNEL);
    Cy_PDMA_Enable(DW0);

    /* Trigger MUX */
    Cy_TrigMux_Connect1To1( ADC_TO_DMA_TRIG,
                            CY_TR_MUX_TR_INV_DISABLE,
                            TRIGGER_TYPE_EDGE,
                            0ul);

    /* Interrupt Initialization */
    Cy_SysInt_InitIRQ(&stc_sysint_irq_cfg);
    Cy_SysInt_SetSystemIrqVector(stc_sysint_irq_cfg.sysIntSrc, DW0_Ch_IntHandler);
```

```
    /*  Configures interrupt   */
     NVIC_SetPriority(stc_sysint_irq_cfg.intIdx, 0ul);
     NVIC_EnableIRQ(stc_sysint_irq_cfg.intIdx);

     /* Issue SW trigger */
    /*  Generates a software trigger   */
     Cy_Adc_Channel_SoftwareTrigger(&ADC_MACRO->CH[ADC_GROUP_FIRST_CHANNEL]);

     for(;;);
}
```

Code Listing 40 P-DMA チャネル、ディスクリプタの設定

```
/**
 * \var uint32_t g_DestBuffer
 * \brief DMA Destination buffer
 */
static uint32_t g_DestBuffer[BUFFER_SIZE_IN_BYTE] = {0ul};

/**
 * \var cy_stc_pdma_descr_t stcDescr
 * \brief PDMA descriptor
 */
static cy_stc_pdma_descr_t stcDescr;

/**
 * \var cy_stc_pdma_chnl_config_t chnlConfig
 * \brief PDMA configuration
 */
static const cy_stc_pdma_chnl_config_t   chnlConfig =
   /*  Configure P-DMA channel  */
{
    .PDMA_Descriptor = &stcDescr,
    .preemptable     = 0ul,
    .priority        = 0ul,
    .enable          = 1ul,
};

/**
 * \var cy_stc_pdma_descr_config_t stcDmaDescrConfig
 * \brief PDMA descriptor configuration
 */
static const cy_stc_pdma_descr_config_t  stcDmaDescrConfig =
   /*  Configure P-DMA descriptor  */
{
    .deact          = 0ul,
    .intrType       = CY_PDMA_INTR_DESCR_CMPLT,
    .trigoutType    = CY_PDMA_TRIGOUT_DESCR_CMPLT,
    .chStateAtCmplt = CY_PDMA_CH_ENABLED,
    .triginType     = CY_PDMA_TRIGIN_XLOOP,
    .dataSize       = CY_PDMA_WORD,
    .srcTxfrSize    = 0ul,  /* as specified in DATA_SIZE */
    .destTxfrSize   = 0ul,  /* as specified in DATA_SIZE */
    .descrType      = CY_PDMA_2D_TRANSFER,
    .srcAddr        = (void *)&ADC_MACRO->CH[ADC_GROUP_FIRST_CHANNEL].unRESULT.u32Register,
    .destAddr       = (void *)g_DestBuffer,
    .srcXincr       = 16ul,  /* address +40h */
    .destXincr      = 8ul,   /* address +20h */
    .xCount         = ADC_GROUP_NUMBER_OF_CHANNELS,
    .srcYincr       = 0ul,   /* Not increments */
    .destYincr      = 1ul,   /* Destination address +4h */
    .yCount         = 3ul,   /* Store results for three times */
    .descrNext      = &stcDescr
};
```

```
/**
 * \var cy_stc_sysint_irq_t stc_sysint_irq_cfg
 * \brief IRQ DMA configuration
 */
static const cy_stc_sysint_irq_t stc_sysint_irq_cfg =
   /*  Configure interrupt  */
{
    .sysIntSrc = DW_CH_INTR,
    .intIdx    = CPUIntIdx2_IRQn,
    .isEnabled = true,
};
```

Code Listing 41 DW0_Ch_IntHandler()

```
void DW0_Ch_IntHandler(void)
{
    uint32_t masked;

    masked = Cy_PDMA_Chnl_GetInterruptStatusMasked(DW0, DW_CHANNEL);    /*  See Code Listing
42.  */
    if ((masked & CY_PDMA_INTRCAUSE_COMPLETION) != 0ul)
    {
        /* Clear Complete DMA interrupt flag */
        Cy_PDMA_Chnl_ClearInterrupt(DW0, DW_CHANNEL);   /*See Code Listing 43.*/

        /* Read the port status */
        for(uint8_t portcnt = 0; portcnt < DST_BUFFER_SIZE; portcnt++)
        {
            g_portStatus[portcnt] = g_DestBuffer[portcnt];
        }
    }
    else
    {
        CY_ASSERT(false);
    }
}
```

Code Listing 42 Cy_Prot_ConfigPpuFixedSlaveStruct()

```
__STATIC_INLINE uint32_t Cy_PDMA_Chnl_GetInterruptStatusMasked(const volatile stc_DW_t
*pstcPDMA, uint32_t chNum)
{
    return (pstcPDMA->CH_STRUCT[chNum].unINTR_MASKED.u32Register);
}
```

Code Listing 43 Cy_PDMA_Chnl_ClearInterrupt()

```
void Cy_PDMA_Chnl_ClearInterrupt(volatile stc_DW_t *pstcPDMA, uint32_t chNum)
{
    pstcPDMA->CH_STRUCT[chNum].unINTR.u32Register = CY_PDMA_INTR_BIT_MASK;

    /* Readback of the register is required by hardware. */
    (void) pstcPDMA->CH_STRUCT[chNum].unINTR.u32Register;
}
```

Code Listing 44 Cy_PDMA_Disable()

```
void Cy_PDMA_Disable(volatile stc_DW_t *pstcPDMA)
{
    pstcPDMA->unCTL.stcField.u1ENABLED  =       0ul;   /*  Write to P-DMA_CTL_ENABLED bit  */
}
```

Code Listing 45 Cy_PDMA_Chnl_DeInit()

```
void Cy_PDMA_Chnl_DeInit(volatile stc_DW_t *pstcPDMA, uint32_t chNum)
   /*  Resets P-DMA to default value  */
{
    pstcPDMA->CH_STRUCT[chNum].unCH_CTL.u32Register            = 0ul;
    pstcPDMA->CH_STRUCT[chNum].unCH_IDX.u32Register            = 0ul;
    pstcPDMA->CH_STRUCT[chNum].unCH_CURR_PTR.u32Register       = 0ul;
    pstcPDMA->CH_STRUCT[chNum].unINTR_MASK.u32Register         = 0ul;
    pstcPDMA->CH_STRUCT[chNum].unINTR_SET.u32Register          = 0ul;
}
```

Code Listing 46 Cy_PDMA_Descr_Init()

```
cy_en_pdma_status_t Cy_PDMA_Descr_Init(cy_stc_pdma_descr_t* descriptor, const
cy_stc_pdma_descr_config_t* config)
{
    cy_en_pdma_status_t retVal = CY_PDMA_ERR_UNC;

    if ((descriptor != NULL) && (config != NULL))
    {
        descriptor->unPDMA_DESCR_CTL.stcField.u2WAIT_FOR_DEACT          =       config->deact;
        descriptor->unPDMA_DESCR_CTL.stcField.u2INTR_TYPE               =       config-
>intrType;
        descriptor->unPDMA_DESCR_CTL.stcField.u2TR_OUT_TYPE             =       config-
>trigoutType;
        descriptor->unPDMA_DESCR_CTL.stcField.u2TR_IN_TYPE              =       config-
>triginType;
        descriptor->unPDMA_DESCR_CTL.stcField.u1SRC_TRANSFER_SIZE       =       config-
>srcTxfrSize;
        descriptor->unPDMA_DESCR_CTL.stcField.u1DST_TRANSFER_SIZE       =       config-
>destTxfrSize;
        descriptor->unPDMA_DESCR_CTL.stcField.u1CH_DISABLE              =       config-
>chStateAtCmplt;
        descriptor->unPDMA_DESCR_CTL.stcField.u2DATA_SIZE               =       config-
>dataSize;
        descriptor->unPDMA_DESCR_CTL.stcField.u2DESCR_TYPE              =       config-
>descrType;

        descriptor->u32PDMA_DESCR_SRC                                   =
(uint32_t)config->srcAddr;
        descriptor->u32PDMA_DESCR_DST                                   =
(uint32_t)config->destAddr;

        switch(config->descrType)
        {
        case (uint32_t)CY_PDMA_SINGLE_TRANSFER:
        {
            descriptor->unPDMA_DESCR_X_CTL.u32Register                  =       (uint32_t)config-
>descrNext;
            break;
        }
        case (uint32_t)CY_PDMA_1D_TRANSFER:
        {
            descriptor->unPDMA_DESCR_X_CTL.stcField.u12SRC_X_INCR       =       (uint32_t)config-
>srcXincr;
            descriptor->unPDMA_DESCR_X_CTL.stcField.u12DST_X_INCR       =       (uint32_t)config-
>destXincr;
            descriptor->unPDMA_DESCR_X_CTL.stcField.u8X_COUNT           =       (uint32_t)((config-
>xCount) - 1ul);
            descriptor->unPDMA_DESCR_Y_CTL.u32Register                  =       (uint32_t)config-
>descrNext;
            break;
        }
        case (uint32_t)CY_PDMA_CRC_TRANSFER:
```

```
        {
            descriptor->unPDMA_DESCR_X_CTL.stcField.u12SRC_X_INCR   =       (uint32_t)config->srcXincr;
            descriptor->unPDMA_DESCR_X_CTL.stcField.u12DST_X_INCR   =       0ul;
            descriptor->unPDMA_DESCR_X_CTL.stcField.u8X_COUNT       =       (uint32_t)((config->xCount) - 1ul);
            descriptor->unPDMA_DESCR_Y_CTL.u32Register              =       (uint32_t)config->descrNext;
            break;
        }
        case (uint32_t)CY_PDMA_2D_TRANSFER:
        {
            descriptor->unPDMA_DESCR_X_CTL.stcField.u12SRC_X_INCR   =       (uint32_t)config->srcXincr;
            descriptor->unPDMA_DESCR_X_CTL.stcField.u12DST_X_INCR   =       (uint32_t)config->destXincr;
            descriptor->unPDMA_DESCR_X_CTL.stcField.u8X_COUNT       =       (uint32_t)((config->xCount) - 1ul);

            descriptor->unPDMA_DESCR_Y_CTL.stcField.u12SRC_Y_INCR   =       (uint32_t)config->srcYincr;
            descriptor->unPDMA_DESCR_Y_CTL.stcField.u12DST_Y_INCR   =       (uint32_t)config->destYincr;
            descriptor->unPDMA_DESCR_Y_CTL.stcField.u8Y_COUNT       =       (uint32_t)((config->yCount) - 1ul);

            descriptor->u32PDMA_DESCR_NEXT_PTR                      =       (uint32_t)config->descrNext;
            break;
        }
        default:
        {
            /* Unsupported type of descriptor */
            break;
        }
        }

        retVal = CY_PDMA_SUCCESS;
    }
    else
    {
        retVal = CY_PDMA_INVALID_INPUT_PARAMETERS;
    }

    return retVal;
}
```

Code Listing 47 Cy_PDMA_Chnl_Init()

```
cy_en_pdma_status_t Cy_PDMA_Chnl_Init(volatile stc_DW_t *pstcPDMA, uint32_t chNum, const
cy_stc_pdma_chnl_config_t* chnlConfig)
{
    cy_en_pdma_status_t retVal = CY_PDMA_ERR_UNC;

    if ((pstcPDMA != NULL) && (chnlConfig != NULL))
    {
        /* Set current descriptor */
        pstcPDMA->CH_STRUCT[chNum].unCH_CURR_PTR.u32Register            =  (uint32_t)chnlConfig-
>PDMA_Descriptor;

        /* Set if the channel is preemtable */
        pstcPDMA->CH_STRUCT[chNum].unCH_CTL.stcField.u1PREEMPTABLE      =  chnlConfig-
>preemptable;

        /* Set channel priority */
        pstcPDMA->CH_STRUCT[chNum].unCH_CTL.stcField.u2PRIO             =  chnlConfig->priority;

        /* Set enabled status */
        pstcPDMA->CH_STRUCT[chNum].unCH_CTL.stcField.u1ENABLED          =  chnlConfig->enable;

        retVal = CY_PDMA_SUCCESS;
    }
    else
    {
        retVal = CY_PDMA_INVALID_INPUT_PARAMETERS;
    }

    return (retVal);
}
```

Code Listing 48 Cy_PDMA_Chnl_Enable()

```
__STATIC_INLINE void Cy_PDMA_Chnl_Enable(volatile stc_DW_t *pstcPDMA, uint32_t chNum)

    pstcPDMA->CH_STRUCT[chNum].unCH_CTL.stcField.u1ENABLED = 1ul;
}
```

Code Listing 49 Cy_PDMA_Chnl_SetInterruptMask()

```
__STATIC_INLINE void Cy_PDMA_Chnl_SetInterruptMask(volatile stc_DW_t *pstcPDMA, uint32_t chNum)
{
    pstcPDMA->CH_STRUCT[chNum].unINTR_MASK.u32Register = CY_PDMA_INTR_BIT_MASK;
}
```

Code Listing 50 Cy_PDMA_Enable()

```
void Cy_PDMA_Enable(volatile stc_DW_t *pstcPDMA)
{
    pstcPDMA->unCTL.stcField.u1ENABLED  =       1ul;    /*  Write to P-DMA_CTL_ENABLED bit  */
}
```

3.5 CRC transfer

3.5.1 概要

ここでは、CRC transfer の例について説明します。CRC transfer は P-DMA 特有のディスクリプタタイプです。CRC transfer は、転送元アドレスとサイズで指定した領域の CRC を計算し、その結果を転送先アドレスに転送します。

これは、CRC transfer によるフラッシュでのプログラムコード検証の例です。CPU は、プログラム実行前に P-DMA でプログラムコード領域の CRC を計算します。CRC 計算は CRC32 を使用して実行されます。CRC 計算結果が期待値と一致すると、CPU はプログラムの実行を開始します。この例では、プログラムコードの期待値を準備する必要があることに注意してください。

P-DMA によって設定できる CRC パラメータの詳細については、Architecture TRM を参照してください。

図 17 に CRC transfer のユースケースを示します。

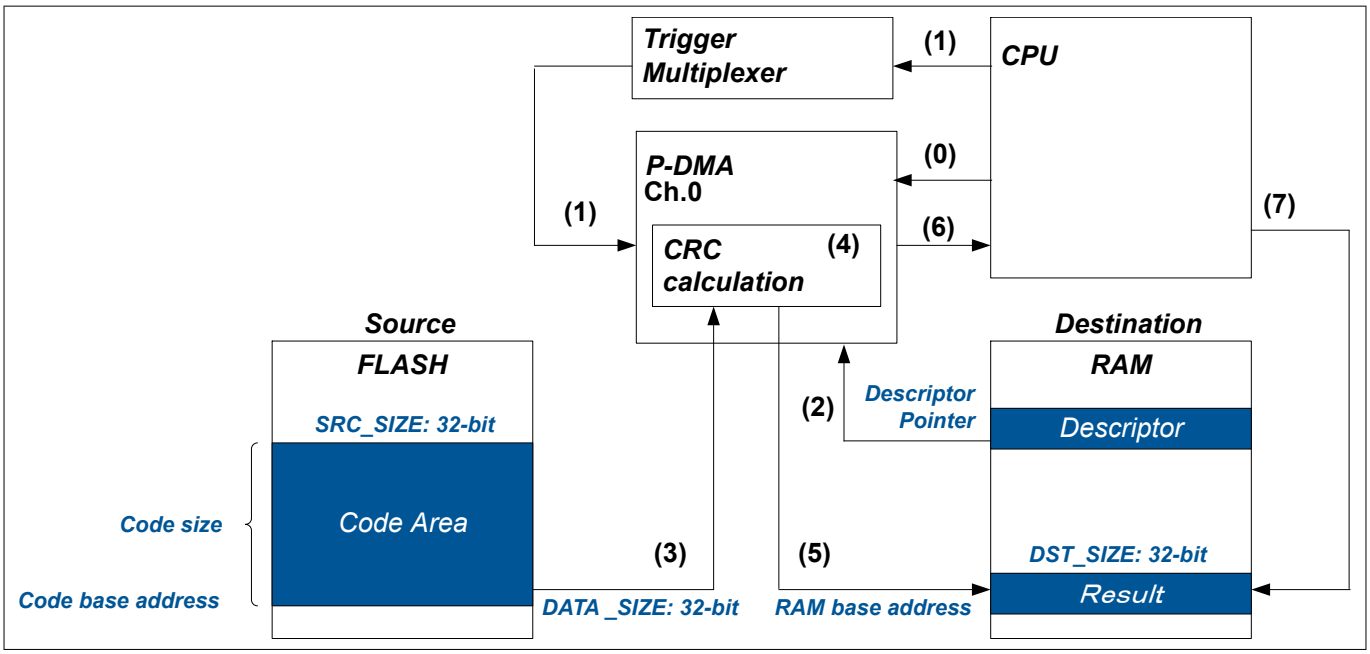


図 17 CRC Transfer のユースケース

(0) ユースケースに従って P-DMA を設定してください。

(1) CPU は、Software trigger を Trigger multiplexer を介して P-DMA に通知します。

(2) P-DMA は、転送を受け付け時に、指定された領域 (ディスクリプタポインタ) からディスクリプタを読み出します。

(3) P-DMA は、転送元アドレス (コードベースアドレス) からデータを読み込みます。

(4) P-DMA は、読み出したデータを CRC 演算器に入力し、アドレスをインクリメントした後、再びデータを読み出します。P-DMA は転送サイズ (コードサイズ) で指定された領域に達するまで (4) を繰り返します。

(5) 指定エリアの CRC 計算が終了すると、その結果を転送先アドレス (RAM ベースアドレス) に転送します。

(6) P-DMA は、割込みを CPU に通知します。
CPU が割込みを受け付けると、その結果を期待値と比較します。一致すると、CPU はプログラムの実行を開始します。一致しない場合、CPU は安全動作モードに移行します。

3.5.2 初期設定

ここでは、このユースケースの P-DMA チャネルとディスクリプタの初期化について説明します。
図 18 に P-DMA の設定手順を示します。

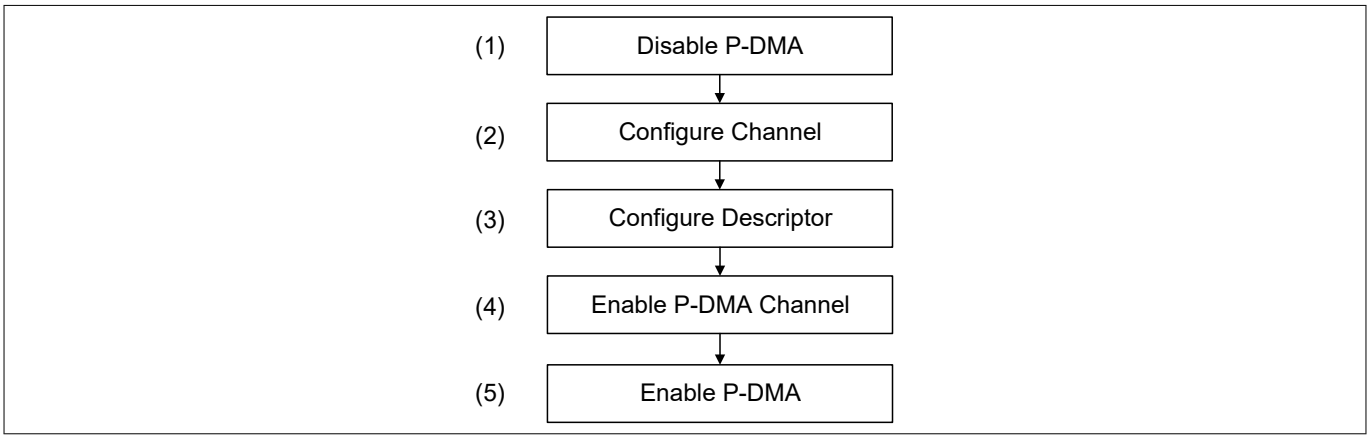


図 18 P-DMA の設定手順

3.5.3 設定とサンプルコード

表 15 は DMA 用の SDL におけるパラメータの一覧、表 16 は設定部の機能の一覧です。

表 15 DMA 設定パラメータの一覧

パラメータ	説明	値
BUFFER_SIZE	バッファ サイズの定義	5
DW_CHANNEL	DW チャネルの定義	0
EXPECTED_RESULT_VALUE	結果値の定義	(0x3C4687AFUL)
.data_reverse	最下位ビット (ビット 0) を優先	1
.rem_reverse	残りはビット反転	1
.data_xor	各データバイトは 00h と XOR されます。	0
.polynomial	CRC32: POLYNOMIAL	0x04c11db7
.lfsr32	シード値	0xFFFFFFFF
.rem_xor	CRC_LFSR_CTL.LFSR32 register is XORed with FFFFh	0xFFFFFFFF
.PDMA_Descriptor	P-DMA の現在のディスクリプタポインタ	&stcDescr
.preemptable	チャネル プリエンプタブル	0
.priority	チャネル優先度	0
.enable	チャネル イネーブル	1

(続く)

表 15 (続き) DMA 設定パラメータの一覧

パラメータ	説明	値
.deact	DESCR_CTL WAIT_FOR_DEACT	0
.intrType	DESCR_CTL INTR_TYPE	CY_PDMA_INTR_1ELEMENT_CMPLT
.trigoutType	DESCR_CTL TR_OUT_TYPE	CY_PDMA_TRIGOUT_1ELEMENT_CMPLT
.chStateAtCmplt	DESCR_CTL CH_DISABLE	CY_PDMA_CH_DISABLED
.triginType	DESCR_CTL TR_IN_TYPE	CY_PDMA_TRIGIN_DESCR
.dataSize	DESCR_CTL DATA_SIZE	CY_PDMA_BYTE
.srcTxfrSize	DESCR_CTL SRC_TRANSFER_SIZE	0
.destTxfrSize	DESCR_CTL DST_TRANSFER_SIZE	0
.descrType	DESCR_CTL DESCR_TYPE	CY_PDMA_CRC_TRANSFER
.srcAddr	DESCR_SRC	(void*) au8SrcBuffer
.destAddr	DESCR_DST	0
.srcXincr	DESCR_X_CTL SRC_X_INCR	1
.destXincr	DESCR_X_CTL DST_X_INCR	1
.xCount	DESCR_X_CTL X_COUNT	BUFFER_SIZE
.srcYincr	DESCR_Y_CTL SRC_Y_INCR	0
.destYincr	DESCR_Y_CTL DST_Y_INCR	0
.yCount	DESCR_Y_CTL Y_COUNT	0

表 16 DMA 設定機能の一覧

機能	説明	備考
Cy_GPIO_Pin_Init()	GPIO ピンの設定	-
Cy_PDMA_Disable()	P-DMA ディセーブルの設定	P-DMA_CTL_ENABLED ビットに書き込みます。 Code Listing 53 を参照してください
Cy_PDMA_CRC_Config()	P-DMA CRC の設定	Code Listing 54 を参照してください
Cy_PDMA_Descr_Init()	P-DMA ディスクリプタ初期化の設定	Code Listing 55 を参照してください
Cy_PDMA_Chnl_Init()	P-DMA チャネル初期化の設定	Code Listing 56 を参照してください
Cy_PDMA_Enable()	P-DMA イネーブルの設定	P-DMA_CTL_ENABLED ビットに書き込みます。 Code Listing 57 を参照してください
Cy_TrigMux_SwTrigger()	ソフトウェアトリガの生成	-

Code Listing 51 は、2D Transfer (ペリフェラルからメモリ) へのプログラム例を示します。GPIO およびクロックの設定については、Architecture TRM およびアプリケーションノートを参照してください。

Code Listing 51 CRC 転送の例

```
/*  Define buffer size
Define P-DMA channel
Define result value   */
#define BUFFER_SIZE         5
#define DW_CHANNEL          0

#define EXPECTED_RESULT_VALUE (0x3C4687AFUL)

int main(void)
{
:
    /* Place your initialization/startup code here (e.g. MyInst_Start()) */
    Cy_GPIO_Pin_Init(CY_LED8_PORT, CY_LED8_PIN, &user_led8_port_pin_cfg);

    Cy_PDMA_Disable(pstcDW);

    Cy_PDMA_CRC_Config( pstcDW,
                        &stcCrcConfig);

   /*  (1) Disable P-DMA. See Code Listing 53.
(2) Configures P-DMA CRC. See Code Listing 54.
(3) Configures P-DMA descriptor initialize. See Code Listing 55.
(4) Enable P-DMA channel. See Code Listing 56.
(5) Enable P-DMA. See Code Listing 57  */
    stcDmaDescrConfig.destAddr = (void *)&pstcDW->unCRC_LFSR_CTL.u32Register;
    Cy_PDMA_Descr_Init(&stcDescr,&stcDmaDescrConfig);
    Cy_PDMA_Chnl_Init(  pstcDW,
                        DW_CHANNEL,
                        (const cy_stc_pdma_chnl_config_t*) &chnlConfig);


    Cy_PDMA_Enable(pstcDW);

   /*  (1) Disable P-DMA. See Code Listing 53.
(2) Configures P-DMA CRC. See Code Listing 54.
(3) Configures P-DMA descriptor initialize. See Code Listing 55.
(4) Enable P-DMA channel. See Code Listing 56.
(5) Enable P-DMA. See Code Listing 57  */
    /*Trigger DMA by SW*/
    Cy_TrigMux_SwTrigger(TRIG_OUT_MUX_0_PDMA0_TR_IN0,
                         TRIGGER_TYPE_CPUSS_DW0_TR_IN__EDGE,
                         1);                            /*output*/

    //Wait for completion
    while(pstcDW->CH_STRUCT[DW_CHANNEL].unCH_CTL.stcField.u1ENABLED)
    {
        u32Time++;
    }

    u32CrcResult                            =   Cy_PDMA_GetCrcRemainderResult(pstcDW);
```

```
    CY_ASSERT(u32CrcResult == EXPECTED_RESULT_VALUE);

}
```

Code Listing 52 P-DMA チャネル、ディスクリプタの設定

```
static  volatile stc_DW_t*          pstcDW          =   DW0;
static  cy_stc_pdma_descr_t         stcDescr;
const   uint8_t                     au8SrcBuffer[]   =   {0x12,0x34,0x56,0x78,0x9a};

const   cy_stc_pdma_crc_config_t    stcCrcConfig    =   {
                                                        .data_reverse = 1,
                                                        .rem_reverse  = 1,
                                                        .data_xor     = 0,
                                                        .polynomial   = 0x04c11db7,
                                                        .lfsr32       =
0xFFFFFFFF,
                                                        .rem_xor      =
0xFFFFFFFF,
                                                        };
const   cy_stc_pdma_chnl_config_t   chnlConfig      =   {
                                                        .PDMA_Descriptor=   &stcDescr,
                                                        .preemptable    =   0,
                                                        .priority       =   0,
                                                        .enable         =   1,  /*enabled after
initialization*/
                                                        };
static  uint32_t                    u32Time         =   0;
static  cy_stc_pdma_descr_config_t  stcDmaDescrConfig=  {
                                                        .deact          =   0,  /*Do not wait
for trigger de-activation*/
                                                        .intrType       =
CY_PDMA_INTR_1ELEMENT_CMPLT,
                                                        .trigoutType    =
CY_PDMA_TRIGOUT_1ELEMENT_CMPLT,
                                                        .chStateAtCmplt =
CY_PDMA_CH_DISABLED,
                                                        .triginType     =
CY_PDMA_TRIGIN_DESCR,
                                                        .dataSize       =   CY_PDMA_BYTE,
                                                        .srcTxfrSize    =   0,  /*= dataSize*/
                                                        .destTxfrSize   =   0,  /*= dataSize*/
                                                        .descrType      =
CY_PDMA_CRC_TRANSFER,
                                                        .srcAddr        =   (void*)
au8SrcBuffer,
                                                        .destAddr       =   0,          //below
initialized
                                                        .srcXincr       =   1,
                                                        .destXincr      =   1,
                                                        .xCount         =   BUFFER_SIZE,
                                                        .srcYincr       =   0,
                                                        .destYincr      =   0,
                                                        .yCount         =   0,
                                                        };
```

Code Listing 53 Cy_PDMA_Disable()

```
void Cy_PDMA_Disable(volatile stc_DW_t *pstcPDMA)
{
    pstcPDMA->unCTL.stcField.u1ENABLED  =       0ul;   /*  Write to P-DMA_CTL_ENABLED bit  */
}
```

Code Listing 54 Cy_PDMA_CRC_Config()

```
void Cy_PDMA_CRC_Config ( volatile stc_DW_t * pstcPDMA,
                          const cy_stc_pdma_crc_config_t*   pstcCrcConfig)
{
   /*  Configures P-DMA CRC   */
    pstcPDMA->unCRC_CTL.stcField.u1DATA_REVERSE     = pstcCrcConfig->data_reverse;
    pstcPDMA->unCRC_CTL.stcField.u1REM_REVERSE      = pstcCrcConfig->rem_reverse;
    pstcPDMA->unCRC_DATA_CTL.stcField.u8DATA_XOR    = pstcCrcConfig->data_xor;
    pstcPDMA->unCRC_LFSR_CTL.stcField.u32LFSR32     = pstcCrcConfig->lfsr32;
    pstcPDMA->unCRC_POL_CTL.stcField.u32POLYNOMIAL  = pstcCrcConfig->polynomial;
    pstcPDMA->unCRC_REM_CTL.stcField.u32REM_XOR     = pstcCrcConfig->rem_xor;
}
```

Code Listing 55 Cy_PDMA_Descr_Init()

```
cy_en_pdma_status_t Cy_PDMA_Descr_Init(cy_stc_pdma_descr_t* descriptor, const
cy_stc_pdma_descr_config_t* config)
{
    cy_en_pdma_status_t retVal = CY_PDMA_ERR_UNC;

    if ((descriptor != NULL) && (config != NULL))
    {
        descriptor->unPDMA_DESCR_CTL.stcField.u2WAIT_FOR_DEACT          =       config->deact;
        descriptor->unPDMA_DESCR_CTL.stcField.u2INTR_TYPE               =       config-
>intrType;
        descriptor->unPDMA_DESCR_CTL.stcField.u2TR_OUT_TYPE             =       config-
>trigoutType;
        descriptor->unPDMA_DESCR_CTL.stcField.u2TR_IN_TYPE              =       config-
>triginType;
        descriptor->unPDMA_DESCR_CTL.stcField.u1SRC_TRANSFER_SIZE       =       config-
>srcTxfrSize;
        descriptor->unPDMA_DESCR_CTL.stcField.u1DST_TRANSFER_SIZE       =       config-
>destTxfrSize;
        descriptor->unPDMA_DESCR_CTL.stcField.u1CH_DISABLE              =       config-
>chStateAtCmplt;
        descriptor->unPDMA_DESCR_CTL.stcField.u2DATA_SIZE               =       config-
>dataSize;
        descriptor->unPDMA_DESCR_CTL.stcField.u2DESCR_TYPE              =       config-
>descrType;

        descriptor->u32PDMA_DESCR_SRC                                   =
(uint32_t)config->srcAddr;
        descriptor->u32PDMA_DESCR_DST                                   =
(uint32_t)config->destAddr;

        switch(config->descrType)
        {
        case (uint32_t)CY_PDMA_SINGLE_TRANSFER:
        {
            descriptor->unPDMA_DESCR_X_CTL.u32Register                  =       (uint32_t)config-
>descrNext;
            break;
        }
        case (uint32_t)CY_PDMA_1D_TRANSFER:
        {
            descriptor->unPDMA_DESCR_X_CTL.stcField.u12SRC_X_INCR       =       (uint32_t)config-
>srcXincr;
            descriptor->unPDMA_DESCR_X_CTL.stcField.u12DST_X_INCR       =       (uint32_t)config-
>destXincr;
            descriptor->unPDMA_DESCR_X_CTL.stcField.u8X_COUNT           =       (uint32_t)((config-
>xCount) - 1ul);
            descriptor->unPDMA_DESCR_Y_CTL.u32Register                  =       (uint32_t)config-
>descrNext;
            break;
        }
        case (uint32_t)CY_PDMA_CRC_TRANSFER:
```

```
        {
            descriptor->unPDMA_DESCR_X_CTL.stcField.u12SRC_X_INCR   =       (uint32_t)config-
>srcXincr;
            descriptor->unPDMA_DESCR_X_CTL.stcField.u12DST_X_INCR   =       0ul;
            descriptor->unPDMA_DESCR_X_CTL.stcField.u8X_COUNT       =       (uint32_t)((config-
>xCount) - 1ul);
            descriptor->unPDMA_DESCR_Y_CTL.u32Register              =       (uint32_t)config-
>descrNext;
            break;
        }
        case (uint32_t)CY_PDMA_2D_TRANSFER:
        {
            descriptor->unPDMA_DESCR_X_CTL.stcField.u12SRC_X_INCR   =       (uint32_t)config-
>srcXincr;
            descriptor->unPDMA_DESCR_X_CTL.stcField.u12DST_X_INCR   =       (uint32_t)config-
>destXincr;
            descriptor->unPDMA_DESCR_X_CTL.stcField.u8X_COUNT       =       (uint32_t)((config-
>xCount) - 1ul);

            descriptor->unPDMA_DESCR_Y_CTL.stcField.u12SRC_Y_INCR   =       (uint32_t)config-
>srcYincr;
            descriptor->unPDMA_DESCR_Y_CTL.stcField.u12DST_Y_INCR   =       (uint32_t)config-
>destYincr;
            descriptor->unPDMA_DESCR_Y_CTL.stcField.u8Y_COUNT       =       (uint32_t)((config-
>yCount) - 1ul);

            descriptor->u32PDMA_DESCR_NEXT_PTR                      =       (uint32_t)config-
>descrNext;
            break;
        }
        default:
        {
            /* Unsupported type of descriptor */
            break;
        }
        }

        retVal = CY_PDMA_SUCCESS;
    }
    else
    {
        retVal = CY_PDMA_INVALID_INPUT_PARAMETERS;
    }

    return retVal;
}
```

Code Listing 56 Cy_PDMA_Chnl_Init()

```
cy_en_pdma_status_t Cy_PDMA_Chnl_Init(volatile stc_DW_t *pstcPDMA, uint32_t chNum, const
cy_stc_pdma_chnl_config_t* chnlConfig)
{
    cy_en_pdma_status_t retVal = CY_PDMA_ERR_UNC;

    if ((pstcPDMA != NULL) && (chnlConfig != NULL))
    {
        /* Set current descriptor */
        pstcPDMA->CH_STRUCT[chNum].unCH_CURR_PTR.u32Register            =  (uint32_t)chnlConfig-
>PDMA_Descriptor;

        /* Set if the channel is preemtable */
        pstcPDMA->CH_STRUCT[chNum].unCH_CTL.stcField.u1PREEMPTABLE      =  chnlConfig-
>preemptable;

        /* Set channel priority */
        pstcPDMA->CH_STRUCT[chNum].unCH_CTL.stcField.u2PRIO             =  chnlConfig->priority;

        /* Set enabled status */
        pstcPDMA->CH_STRUCT[chNum].unCH_CTL.stcField.u1ENABLED          =  chnlConfig->enable;

        retVal = CY_PDMA_SUCCESS;
    }
    else
    {
        retVal = CY_PDMA_INVALID_INPUT_PARAMETERS;
    }

    return (retVal);
}
```

Code Listing 57 Cy_PDMA_Enable()

```
void Cy_PDMA_Enable(volatile stc_DW_t *pstcPDMA)
{
    pstcPDMA->unCTL.stcField.u1ENABLED  =       1ul;   /*  Write to P-DMA_CTL_ENABLED bit  */
}
```

4 M-DMA ユースケース

ここでは、サンプルドライバライブラリ (SDL) を使用した Smart I/O の使用方法について説明します。このアプリケーションノートに記載されているプログラムコードは、SDL に含まれるものです。SDL については、その他の参考資料を参照してください。

SDL には設定部とドライバ部があります。設定部では、主に目的の動作をさせるためのパラメータ値を設定します。ドライバ部は設定部のパラメータ値に基づいて各レジスタを設定します。設定部は、お客様のシステムに合わせて設定できます。

この例では、CYT2B7 シリーズを使用しています。

4.1 メモリからメモリ (Memory Copy)

ここでは、Memory copy の例について説明します。Memory copy は、M-DMA 固有のディスクリプタタイプです。これは特別な 1D transfer です。Memory copy は、転送元アドレスとサイズで指定した領域から転送先アドレスにデータを転送します。

これはフラッシュから RAM へのデータ転送の例です。このディスクリプタタイプは、RAM の実行およびベクタテーブルのコピー用のプログラムコードのコピーに役立ちます。この例では、連続するフラッシュメモリ領域を RAM に転送します。

図 19 に、memory copy を使用したメモリ間転送のユースケースを示します。

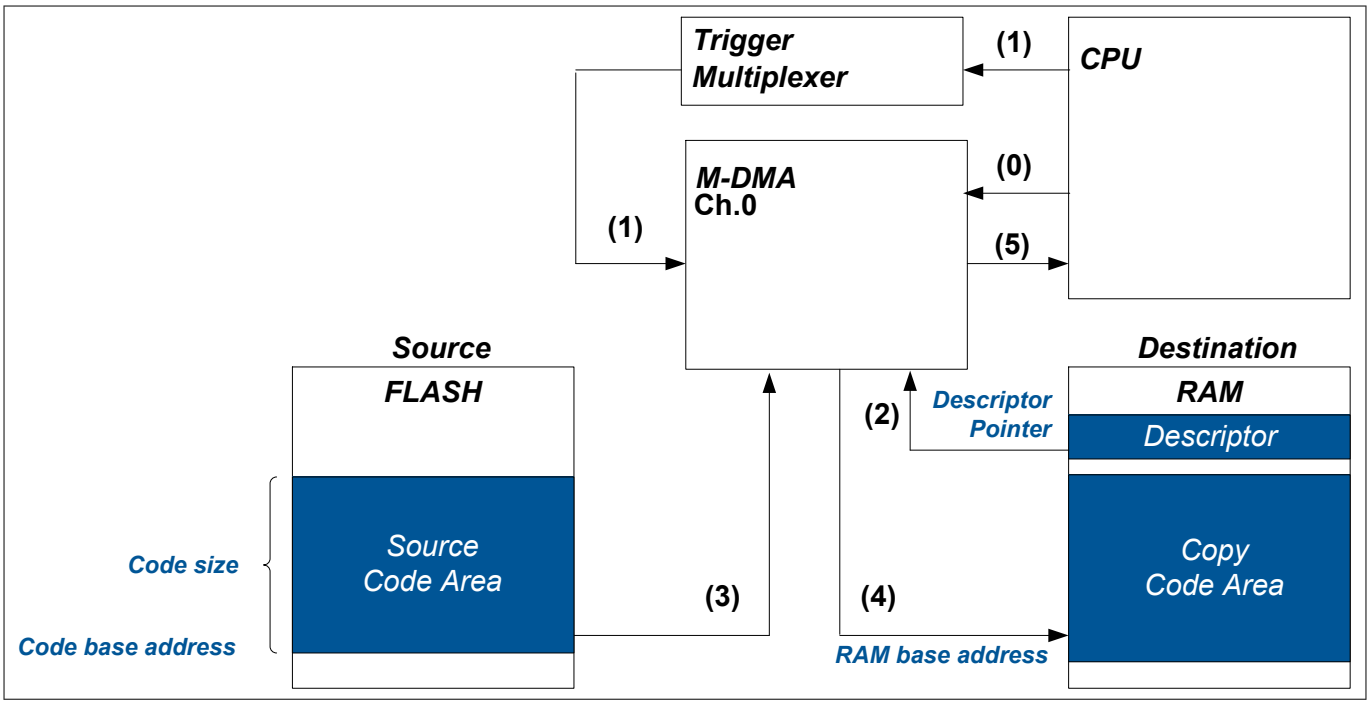


図 19 Memory copy を使用したメモリ間転送のユースケース

(0) ユースケースの設定に従って M-DMA を設定してください。

(1) CPU は、Software trigger を Trigger multiplexer を介して M-DMA に通知します。

(2) M-DMA は、転送を受け付けると、指定された領域 (ディスクリプタポインタ) からディスクリプタを読み出します。

(3) M-DMA は、転送元アドレス (コードベースアドレス) からデータを読み出します。

(4) M-DMA は読み出したデータを転送先アドレス (RAM ベースアドレス) に書き込みます。その後、転送元アドレスと転送先アドレスをインクリメントします。M-DMA は転送サイズ (コードサイズ) で指定した領域に達するまで (3) と (4) を繰り返します。

(5) 指定領域のメモリコピー完了後、M-DMA は割込みを CPU に通知します。

4.1.1 初期設定

ここでは、このユースケースの M-DMA チャネルとディスクリプタの初期化について説明します。
図 20 に M-DMA の設定手順を示します。

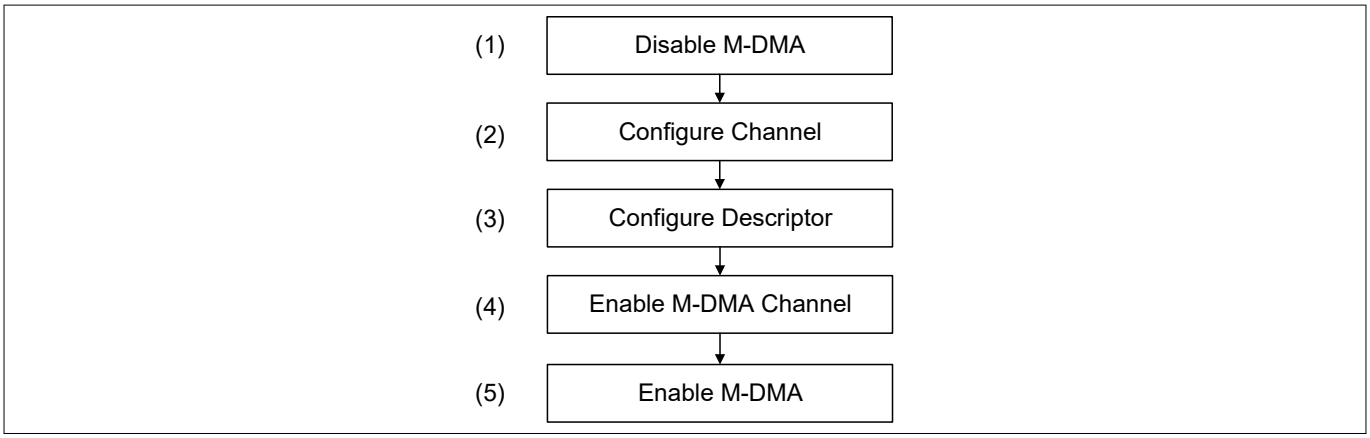


図 20 M-DMA の設定手順

4.1.2 設定とサンプルコード

表 17 は M-DMA 用の SDL におけるパラメータの一覧、表 18 は設定部の機能の一覧です。

表 17 DMA 設定パラメータの一覧

パラメータ	説明	値
BUFFER_SIZE	バッファ サイズの定義	36
DMAC_CHANNEL	M-DMA チャネルの定義	0
.MDMA_Descriptor	M-DMA の現在のディスクリプタポインタ	&stcDescr
.preemptable	チャネル プリエンプタブル	0
.priority	チャネル優先度	0
.enable	チャネル イネーブル	1
.deact	DESCR_CTL WAIT_FOR_DEACT	0
.intrType	DESCR_CTL INTR_TYPE	CY_PDMA_INTR_1ELEMENT_CMPLT
.trigoutType	DESCR_CTL TR_OUT_TYPE	CY_PDMA_TRIGOUT_1ELEMENT_CMPLT
.chStateAtCmplt	DESCR_CTL CH_DISABLE	CY_MDMA_CH_DISABLED
.triginType	DESCR_CTL TR_IN_TYPE	CY_PDMA_TRIGIN_DESCR
.dataSize	DESCR_CTL DATA_SIZE	CY_MDMA_BYTE
.srcTxfrSize	DESCR_CTL SRC_TRANSFER_SIZE	0
.destTxfrSize	DESCR_CTL DST_TRANSFER_SIZE	0
.descrType	DESCR_CTL DESCR_TYPE	CY_MDMA_1D_TRANSFER
.srcAddr	DESCR_SRC	au8SrcBuffer
.destAddr	DESCR_DST	au8DestBuffer

(続く)

表 17 (続き) DMA 設定パラメータの一覧

パラメータ	説明	値
.srcXincr	DESCR_X_INCR SRC_X_INCR	1
.destXincr	DESCR_X_INCR DST_X_INCR	1
.xCount	DESCR_X_SIZE X_COUNT	BUFFER_SIZE
.srcYincr	DESCR_Y_INCR SRC_Y_INCR	0
.destYincr	DESCR_Y_INCR DST_Y_INCR	0
.yCount	DESCR_Y_SIZE Y_COUNT	0
.descrNext	DESCR_NEXT_PTR	0

表 18 DMA 設定機能の一覧

機能	説明	備考
Cy_MDMA_Disable()	M-DMA ディセーブルの設定	M-DMA_CTL_ENABLED ビットに書き込みます。 Code Listing 60 を参照してください
Cy_MDMA_Descr_Init()	M-DMA ディスクリプタ初期化の設定	Code Listing 61 を参照してください
Cy_MDMA_Chnl_Init()	M-DMA チャネル初期化の設定	Code Listing 62 を参照してください
Cy_MDMA_Chnl_Enable()	M-DMA チャネルイネーブル化の設定	Code Listing 63 を参照してください
Cy_MDMA_Enable()	M-DMA イネーブルの設定	M-DMA_CTL_ENABLED ビットに書き込みます。 Code Listing 64 を参照してください
Cy_TrigMux_SwTrigger()	ソフトウェアトリガの生成	-

Code Listing 58 に、2D Transfer (ペリフェラルからメモリ) へのプログラム例を示します。GPIO、クロックの設定については、Architecture TRM およびアプリケーションノートを参照してください。

Code Listing 58 M-DMA 転送の例

```
   /*  Define buffer size
Define DMAC channel   */
#define BUFFER_SIZE             36
#define DMAC_CHANNEL            0

int main(void)
{
:
    uint32_t  i;
    uint8_t   *p_src;

    __enable_irq(); /* Enable global interrupts. */

    /*******/
    /* DMA */
    /*******/

    /* Initialie & Enable DMA */
   /*  (1) Disable M-DMA. See Code Listing 60
(2) Configures M-DMA channel initialize. See Code Listing 62
(3) Configures M-DMA descriptor initialize. See Code Listing 61
(4) Enable M-DMA channel. See Code Listing 63
(5) Enable M-DMA. See Code Listing 64  */
    Cy_MDMA_Disable(DMAC);
    Cy_MDMA_Descr_Init(&stcDescr,&stcDmaDescrConfig);
    Cy_MDMA_Chnl_Init( DMAC,
                       DMAC_CHANNEL,
                       (const cy_stc_mdma_chnl_config_t*) &chnlConfig);
    Cy_MDMA_Chnl_Enable( DMAC, DMAC_CHANNEL);
    Cy_MDMA_Enable(DMAC);

    /* for test */
    p_src = &au8SrcBuffer[0];
    for (i=0; i<BUFFER_SIZE; i++) {
        *p_src++ = i;
        au8DestBuffer[i] = 0;
    }

    /***************/
    /* Trigger MUX */
    /***************/
    Cy_TrigMux_SwTrigger( TRIG_OUT_MUX_2_MDMA_TR_IN0,
                          TRIGGER_TYPE_CPUSS_DMAC_TR_IN__EDGE,
                          1u);

    for(;;)
    {
    }
}
```

Code Listing 59 M-DMA チャネル、ディスクリプタの設定

```
static  uint8_t                     au8DestBuffer[BUFFER_SIZE] = {0};
static  uint8_t                     au8SrcBuffer[BUFFER_SIZE];
static  cy_stc_mdma_descr_t         stcDescr;
   /*  Configure M-DMA channel  */
const   cy_stc_mdma_chnl_config_t   chnlConfig      =   {
        /* CURR_PTR                         */     .MDMA_Descriptor =  &stcDescr,
                                                   .preemptable    =   0,
                                                   .priority       =   0,
                                                   .enable         =   1,
                                               };
   /*  Configure M-DMA descriptor  */
static  cy_stc_mdma_descr_config_t  stcDmaDescrConfig = {
        /* DESCR_CTL       WAIT_FOR_DEACT    */     .deact          =   0,
        /* DESCR_CTL       INTR_TYPE         */     .intrType       =
CY_PDMA_INTR_1ELEMENT_CMPLT,
        /* DESCR_CTL       TR_OUT_TYPE       */     .trigoutType    =
CY_PDMA_TRIGOUT_1ELEMENT_CMPLT,
        /* DESCR_CTL       CH_DISABLE        */     .chStateAtCmplt =   CY_MDMA_CH_DISABLED,
        /* DESCR_CTL       TR_IN_TYPE        */     .triginType     =   CY_PDMA_TRIGIN_DESCR,
        /* DESCR_CTL       DATA_SIZE         */     .dataSize       =   CY_MDMA_BYTE,
        /* DESCR_CTL       SRC_TRANSFER_SIZE */     .srcTxfrSize    =   0,
        /* DESCR_CTL       DST_TRANSFER_SIZE */     .destTxfrSize   =   0,
        /* DESCR_CTL       DESCR_TYPE        */     .descrType      =   CY_MDMA_1D_TRANSFER,
        /* DESCR_SRC                         */     .srcAddr        =   au8SrcBuffer,
        /* DESCR_DST                         */     .destAddr       =   au8DestBuffer,
        /* DESCR_X_INCR    SRC_X_INCR        */     .srcXincr       =   1,
        /* DESCR_X_INCR    DST_X_INCR        */     .destXincr      =   1,
        /* DESCR_X_SIZE    X_COUNT           */     .xCount         =   BUFFER_SIZE,
        /* DESCR_Y_INCR    SRC_Y_INCR        */     .srcYincr       =   0,
        /* DESCR_Y_INCR    DST_Y_INCR        */     .destYincr      =   0,
        /* DESCR_Y_SIZE    Y_COUNT           */     .yCount         =   0,
        /* DESCR_NEXT_PTR                    */     .descrNext      =   0
                                               };
```

Code Listing 60 Cy_MDMA_Disable()

```
void Cy_MDMA_Disable(volatile stc_DMAC_t *pstcMDMA)
{
    pstcMDMA->unCTL.stcField.u1ENABLED  =       0ul;   /*  Write to M-DMA_CTL_ENABLED bit  */
}
```

Code Listing 61 Cy_MDMA_Descr_Init()

```
cy_en_mdma_status_t Cy_MDMA_Descr_Init(cy_stc_mdma_descr_t* descriptor, const
cy_stc_mdma_descr_config_t* config)
{
    cy_en_mdma_status_t retVal = CY_MDMA_ERR_UNC;

    if ((descriptor != NULL) && (config != NULL))
    {
        /* Descriptor[0] */
        descriptor->unMDMA_DESCR_CTL.stcField.u2WAIT_FOR_DEACT     =       config->deact;
        descriptor->unMDMA_DESCR_CTL.stcField.u2INTR_TYPE          =       config->intrType;
        descriptor->unMDMA_DESCR_CTL.stcField.u2TR_OUT_TYPE        =       config->trigoutType;
        descriptor->unMDMA_DESCR_CTL.stcField.u2TR_IN_TYPE         =       config->triginType;
        descriptor->unMDMA_DESCR_CTL.stcField.u1DATA_PREFETCH      =       config->dataPrefetch;
        descriptor->unMDMA_DESCR_CTL.stcField.u2DATA_SIZE          =       config->dataSize;
        descriptor->unMDMA_DESCR_CTL.stcField.u1CH_DISABLE         =       config-
>chStateAtCmplt;
        descriptor->unMDMA_DESCR_CTL.stcField.u1SRC_TRANSFER_SIZE  =       config->srcTxfrSize;
        descriptor->unMDMA_DESCR_CTL.stcField.u1DST_TRANSFER_SIZE  =       config->destTxfrSize;
        descriptor->unMDMA_DESCR_CTL.stcField.u3DESCR_TYPE         =       config->descrType;

        /* Descriptor[1] */
        descriptor->u32MDMA_DESCR_SRC                              =       (uint32_t)config-
>srcAddr;

        /* after 3rd word of descriptor depends on descriptor type */

        switch(config->descrType)
        {
        case (uint32_t)CY_MDMA_SINGLE_TRANSFER:
        {
            /* Descriptor[2] */
            descriptor->u32MDMA_DESCR_DST                          =       (uint32_t)config-
>destAddr;

            /* Descriptor[3] -> NEXT_PTR */
            descriptor->unMDMA_DESCR_X_SIZE.u32Register            =       (uint32_t)config-
>descrNext;
            break;
        }
        case (uint32_t)CY_MDMA_1D_TRANSFER:
        {
            /* Descriptor[2] */
            descriptor->u32MDMA_DESCR_DST                          =       (uint32_t)config-
>destAddr;

            /* Descriptor[3] */
            descriptor->unMDMA_DESCR_X_SIZE.stcField.u16X_COUNT    =       (uint32_t)((config-
>xCount) - 1ul);

            /* Descriptor[4] */
            descriptor->unMDMA_DESCR_X_INCR.stcField.u16SRC_X_INCR =       (uint32_t)config-
```

```
>srcXincr;
            descriptor->unMDMA_DESCR_X_INCR.stcField.u16DST_X_INCR  =       (uint32_t)config-
>destXincr;

            /* Descriptor[5] -> NEXT_PTR */
            descriptor->unMDMA_DESCR_Y_SIZE.u32Register             =       (uint32_t)config-
>descrNext;
            break;
        }
        case (uint32_t)CY_MDMA_2D_TRANSFER:
        {
            /* Descriptor[2] */
            descriptor->u32MDMA_DESCR_DST                           =       (uint32_t)config-
>destAddr;

            /* Descriptor[3] */
            descriptor->unMDMA_DESCR_X_SIZE.stcField.u16X_COUNT     =       (uint32_t)((config-
>xCount) - 1ul);

            /* Descriptor[4] */
            descriptor->unMDMA_DESCR_X_INCR.stcField.u16SRC_X_INCR  =       (uint32_t)config-
>srcXincr;
            descriptor->unMDMA_DESCR_X_INCR.stcField.u16DST_X_INCR  =       (uint32_t)config-
>destXincr;

            /* Descriptor[5] */
            descriptor->unMDMA_DESCR_Y_SIZE.stcField.u16Y_COUNT     =       (uint32_t)((config-
>yCount) - 1ul);

            /* Descriptor[6] */
            descriptor->unMDMA_DESCR_Y_INCR.stcField.u16SRC_Y_INCR  =       (uint32_t)config-
>srcYincr;
            descriptor->unMDMA_DESCR_Y_INCR.stcField.u16DST_Y_INCR  =       (uint32_t)config-
>destYincr;

            /* Descriptor[7] */
            descriptor->u32MDMA_DESCR_NEXT_PTR                      =       (uint32_t)config-
>descrNext;
            break;
        }
        case (uint32_t)CY_MDMA_MEMORY_COPY_TRANSFER:
        {
            /* Descriptor[2] */
            descriptor->u32MDMA_DESCR_DST                           =       (uint32_t)config-
>destAddr;

            /* Descriptor[3] */
            descriptor->unMDMA_DESCR_X_SIZE.stcField.u16X_COUNT     =       (uint32_t)((config-
>xCount) - 1ul);

            /* Descriptor[4] -> NEXT_PTR */
            descriptor->unMDMA_DESCR_X_INCR.u32Register             =       (uint32_t)config-
>descrNext;
```

```
            break;
        }
        case (uint32_t)CY_MDMA_SCATTER_TRANSFER:
        {
            /* Descriptor[2] -> X_SIZE */
            descriptor->u32MDMA_DESCR_DST                           =       (uint32_t)((config-
>xCount) - 1ul);

            /* Descriptor[3] -> NEXT_PTR */
            descriptor->unMDMA_DESCR_X_SIZE.u32Register             =       (uint32_t)config-
>descrNext;
            break;
        }
        default:
        {
            /* Unsupported type of descriptor */
            break;
        }
        }

        retVal = CY_MDMA_SUCCESS;
    }
    else
    {
        retVal = CY_MDMA_INVALID_INPUT_PARAMETERS;
    }

    return retVal;
}
```

Code Listing 62 Cy_MDMA_Chnl_Init()

```
cy_en_mdma_status_t Cy_MDMA_Chnl_Init(volatile stc_DMAC_t *pstcMDMA, uint32_t chNum, const
cy_stc_mdma_chnl_config_t* chnlConfig)
{
    cy_en_mdma_status_t retVal = CY_MDMA_ERR_UNC;

    if ((pstcMDMA != NULL) && (chnlConfig != NULL))
    {
        /* Set current descriptor */
        pstcMDMA->CH[chNum].unCURR.u32Register   =  (uint32_t)chnlConfig->MDMA_Descriptor;

        /* Set if the channel is preemptable  */
        /* There is no the parameter in MDMA */

        /* Set channel priority */
        pstcMDMA->CH[chNum].unCTL.stcField.u2PRIO    =   chnlConfig->priority;

        /* Set enabled status */
        pstcMDMA->CH[chNum].unCTL.stcField.u1ENABLED =   chnlConfig->enable;

        retVal = CY_MDMA_SUCCESS;
    }
    else
    {
        retVal = CY_MDMA_INVALID_INPUT_PARAMETERS;
    }

    return (retVal);
}
```

Code Listing 63 Cy_MDMA_Chnl_ Enable()

```
__STATIC_INLINE void Cy_MDMA_Chnl_Enable(volatile stc_DMAC_t *pstcMDMA, uint32_t chNum)
{
    pstcMDMA->CH[chNum].unCTL.stcField.u1ENABLED = 1ul;
}
```

Code Listing 64 Cy_MDMA_ Enable()

```
void Cy_MDMA_Enable(volatile stc_DMAC_t *pstcMDMA)
{
    pstcMDMA->unCTL.stcField.u1ENABLED  =       1ul;    /*  Write to M-DMA_CTL_ENABLED bit  */
}
```

5 用語集

表 19 用語集

用語	説明
DMA controller	Direct Memory Access controller
P-DMA	Peripheral DMA
M-DMA	Memory DMA
Single transfer	単一のデータ要素 (8 ビット, 16 ビット, または 32 ビット) を転送します。詳細については、Architecture TRM の「Direct Memory Access」章の「Descriptors」セクションを参照してください。
1D transfer	1 次元の "for ループ"を実行します。詳細については、Architecture TRM の「Direct Memory Access」章の「Descriptors」セクションを参照してください。
2D transfer	2 次元の "for ループ"を実行します。詳細については、Architecture TRM の「Direct Memory Access」章の「Descriptors」セクションを参照してください。
CRC transfer	1D transfer と同様に 1 次元の "for ループ"を実行します。ただし、転送元データは転送先に転送されません。CRC は、ソースデータにわたって計算されます。P-DMA のみサポートします。詳細については、Architecture TRM の「Direct Memory Access」章の「Descriptors」セクションを参照してください。
Memory copy	1D transfer の特別なケースです。M-DMA のみサポートします。詳細については、Architecture TRM の「Direct Memory Access」章の「Descriptors」セクションを参照してください。
Scatter	このディスクリプタタイプは、アドレス空間にアドレスが「Scattered (散在)」する 32 ビットのデータ要素のセットを書き込むことを目的とします。M-DMA のみサポートします。詳細については、Architecture TRM の「Direct Memory Access」章の「Descriptors」セクションを参照してください。
Descriptor	DMA チャネルのデータ転送の詳細を指定します。詳細については、Architecture TRM の「Direct Memory Access」章の「Descriptors」セクションを参照してください。
Descriptor chain	DMA チャネルは、ディスクリプタの実行が完了すると、現在のディスクリプタに指定されている次のディスクリプタを実行します。詳細については、Architecture TRM の「Direct Memory Access」章の「Descriptors」セクションを参照してください。
Descriptor list	descriptor chain と同じです。
Descriptor pointer	ディスクリプタが格納されているメモリの開始アドレス。詳細については、Architecture TRM の「Direct Memory Access」章の「Descriptors」セクションを参照してください。
Descriptor type	DMA によって実行される転送動作タイプ。詳細については、Architecture TRM の「Direct Memory Access」章の「Descriptors」セクションを参照してください。
Descriptor word	ディスクリプタの構成要素。ディスクリプタ制御, ソース/宛先アドレス, X/Y ループ制御, およびディスクリプタの次のポインタがあります。詳細については、Architecture TRM の「Direct Memory Access」章の「P-DMA ディスクリプタ構造」と「M-DMA ディスクリプタ構造」セクションを参照してください。

(続く)

表 19 (続き) 用語集

用語	説明
Preemptable	プリエンプタブルは、P-DMA 特有の機能です。詳細については、Architecture TRM の「Direct Memory Access」章の「Channels」セクションを参照してください。
MMIO	Memory Mapped I/O
ADC	Analog-to-digital converter。詳細については、Architecture TRM の「SAR ADC」章を参照してください。
SCB	Serial Communications Block。詳細については、Architecture TRM の「Serial Communications Block (SCB)」章を参照してください。
TCPWM	Timer, Counter, and Pulse Width Modulator。詳細については、Architecture TRM の「Timer, Counter, and PWM」章を参照してください。
Trigger multiplexer	トリガマルチプレクサは、ソースペリフェラルからデスティネーションにトリガをルーティングします。詳細については、Architecture TRM の「Trigger Multiplexer」章を参照してください。

6 参考資料

以下は、TRAVEO™ T2G ファミリのデータシートとテクニカルリファレンスマニュアルです。これらの資料を入手するにはテクニカルサポートに連絡してください。

[1] デバイスデータシート

- CYT2B7 datasheet 32-bit Arm® Cortex®-M4F microcontroller TRAVEO™ T2G family
- CYT2B9 datasheet 32-bit Arm® Cortex®-M4F microcontroller TRAVEO™ T2G family
- CYT4BF datasheet 32-bit Arm® Cortex®-M7 microcontroller TRAVEO™ T2G family
- CYT4DN datasheet 32-bit Arm® Cortex®-M7 microcontroller TRAVEO™ T2G family (Doc No. 002-24601)
- CYT3BB/4BB datasheet 32-bit Arm® Cortex®-M7 microcontroller TRAVEO™ T2G family
- CYT3DL datasheet 32-bit Arm® Cortex®-M7 microcontroller TRAVEO™ T2G family (Doc No. 002-27763)

[2] テクニカル リファレンス マニュアル

- Body Controller Entry ファミリ
  - TRAVEO™ T2G automotive body controller entry family architecture technical reference manual (TRM)
  - TRAVEO™ T2G automotive body controller entry registers technical reference manual (TRM) for CYT2B7
  - TRAVEO™ T2G automotive body controller entry registers technical reference manual (TRM) for CYT2B9
- Body Controller High ファミリ
  - TRAVEO™ T2G automotive body controller high family architecture technical reference manual (TRM)
  - TRAVEO™ T2G automotive body controller high registers technical reference manual (TRM) for CYT4BF
  - TRAVEO™ T2G automotive body controller high registers technical reference manual (TRM) for CYT3BB/4BB
- Cluster 2D ファミリ
  - TRAVEO™ T2G automotive cluster 2D family architecture technical reference manual (TRM) (Doc No. 002-25800)
  - TRAVEO™ T2G automotive cluster 2D registers technical reference manual (TRM) for CYT4DN (Doc No. 002-25923)
  - TRAVEO™ T2G automotive cluster 2D registers technical reference manual (TRM) for CYT 3DL (Doc No. 002-29854)

[3] アプリケーションノート

- AN220193 - TRAVEO™ T2G ファミリ GPIO の使用方法
- AN225401 - TRAVEO™ T2G ファミリのシリアル通信ブロック (SCB) 使用方法
- AN220224 – How to use timer, counter, and PWM (TCPWM) in TRAVEO™ T2G family
- AN219755 - TRAVEO™ T2G ファミリの SAR ADC 使用方法

7 その他の参考資料

各種周辺機器にアクセスするためのサンプルソフトウェアとして、スタートアップを含むサンプルドライバライブラリ (SDL) を提供します。SDL は、AUTOSAR の公式製品ではカバーされていないドライバのリファレンスとしての役割も担っています。SDL は自動車用規格に適合していないため、量産用としては使用できません。このアプリケーションノートに記載されているプログラムコードは、SDL に含まれるものです。SDL を入手するには、テクニカルサポートに連絡してください。

改訂履歴

版数	発行日	変更内容
英語版**	2018-06-09	英語版の改訂内容: New application note.
**	2019-01-30	これは英語版 002-20191 Rev. *A を翻訳した日本語版 002-25978 Rev. ** です。 英語版の改訂内容: Updated associated part family as “TRAVEO™ T2G family CYT2B series”. Changed target part numbers from “CYT2B5/B7 series” to “CYT2B series” in all instances across the document.
*A	2019-04-24	これは英語版 002-20191 Rev. *B を翻訳した日本語版 002-25978 Rev. *A です。英語版の改訂内容: Updated associated part family as “TRAVEO™ T2G family CYT2B/CYT4B series”. Added target part numbers “CYT4B series” related information in all instances across the document. Updated operation overview: Updated configure channel: Updated “Table 3. channel configuration for P-DMA”. Updated P-DMA use cases: Updated CRC transfer: Updated initial configuration: Updated description (correction of errors in CRC register name).
*B	2019-11-18	これは英語版 002-20191 Rev. *C を翻訳した日本語版 002-25978 Rev. *B です。英語版の改訂内容: Updated associated part family as “TRAVEO™ T2G family CYT2B/CYT4B/CYT4D series”. Added target part numbers “CYT4D series” related information in all instances across the document. Added the setting flow of DW_CH_STRUCT_CH_IDX into channel configuration for P-DMA in all instances across the document. Updated operation overview: Updated configure channel: Updated “Table 3. channel configuration for P-DMA”. Updated “Table 4. channel configuration for M-DMA”.
*C	2020-06-12	これは英語版 002-20191 Rev. *D を翻訳した日本語版 002-25978 Rev. *C です。英語版の改訂内容: Updated associated part family as “TRAVEO™ T2G family CYT2/CYT3/CYT4 series”. Changed target part numbers from “CYT2B/CYT4B/CYT4D series” to “CYT2/ CYT4 series” in all instances across the document. Added target part numbers “CYT3 series” related information in all instances across the document.
*D	2021-05-19	これは英語版 002-20191 Rev. *E を翻訳した日本語版 002-25978 Rev. *D です。英語版の改訂内容: Updated to Infineon template.
*E	2022-05-19	これは英語版 002-20191 Rev. *F を翻訳した日本語版 002-25978 Rev. *E です。英語版の改訂内容: Added example of SDL code and description in all instances.
*F	2024-05-22	これは英語版 002-20191 Rev. *G を翻訳した日本語版 002-25978 Rev. *F です。英語版の改訂内容: Template update; no content update

Trademarks

All referenced product or service names and trademarks are the property of their respective owners.

Edition 2024-05-22

Published by

Infineon Technologies AG

81726 Munich, Germany

Do you have a question about any aspect of this document?

Email: erratum@infineon.com

Document reference
IFX-sja1682326000729

重要事項

本手引書に記載された情報は、本製品の使用に関する手引きとして提供されるものであり、いかなる場合も、本製品における特定の機能性能や品質について保証するものではありません。本製品の使用の前に、当該手引書の受領者は実際の使用環境の下であらゆる本製品の機能及びその他本手引書に記された一切の技術的情報について確認する義務が有ります。インフィニオンテクノロジーズはここに当該手引書内で記される情報につき、第三者の知的所有権の不侵害の保証を含むがこれに限らず、あらゆる種類の一切の保証および責任を否定いたします。

本文書に含まれるデータは、技術的訓練を受けた従業員のみを対象としています。本製品の対象用途への適合性、およびこれら用途に関連して本文書に記載された製品情報の完全性についての評価は、お客様の技術部門の責任にて実施してください。

警告事項

技術的要件に伴い、製品には危険物質が含まれる可能性があります。当該種別の詳細については、インフィニオンの最寄りの営業所までお問い合わせください。

インフィニオンの正式代表者が署名した書面を通じ、インフィニオンによる明示の承認が存在する場合を除き、インフィニオンの製品は、当該製品の障害またはその使用に関する一切の結果が、合理的に人的傷害を招く恐れのある一切の用途に使用することはできないこと予めご了承ください。