Computer Science Hardware Design Excise Text A DMA Feb.27th. ,2017

CSHW 2017 Computer Science, Meiji University

CS_HWA_2017.PPT 41 Slides

Renji Mikami

ボトルネック

フォン・ノイマン・ボトルネック

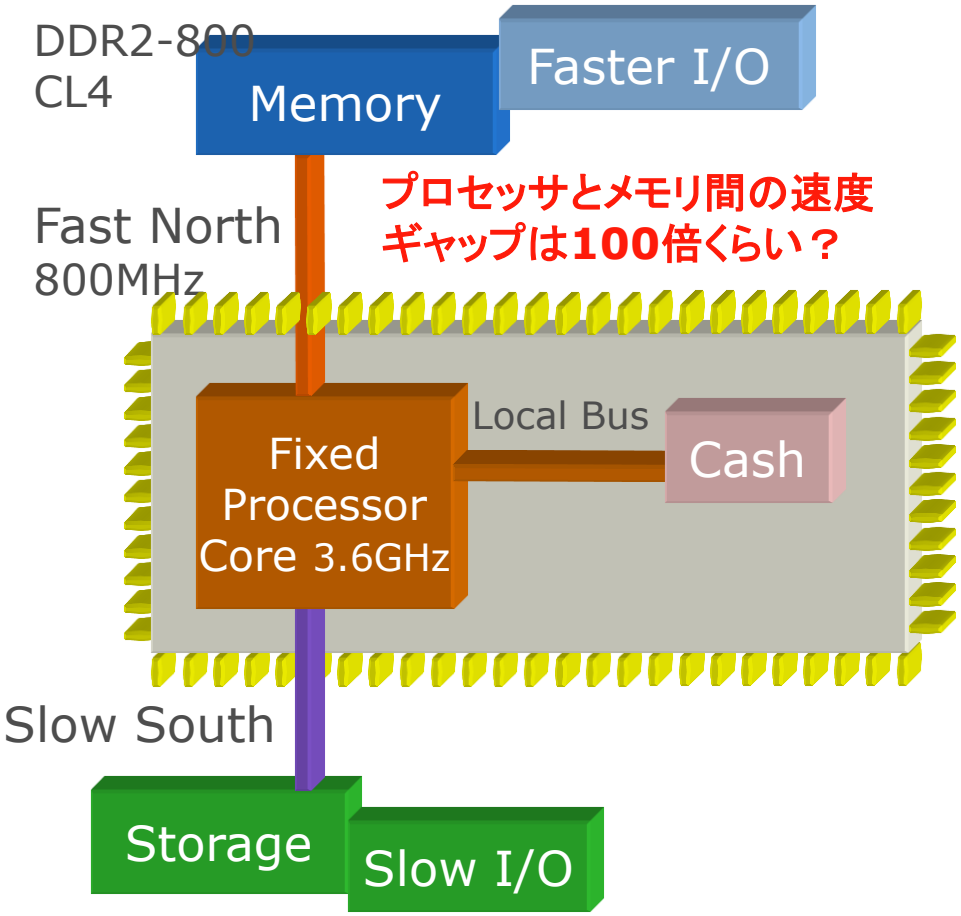
プロセッサの命令セットと実行時間を比較すると、

レジスタ間のデータ転送や即値命令は、1サイクルと非常に高速ですが、メモリの読み書き、特にメモリとメモリ間のデータの転送には、非常に時間がかかります。

また周辺のハードウェアのレジスタのデータの読み書きに関しては、そのハードウェアの処理の終了を待ってから行う必要があります。

処理の終了を知る方法に、CPUが定期的にレジスタの状態を見に行くポーリングと処理が終了した時点でCPUに割り込みをかけて割り込みサービス・ルーチンに飛ばす方法があります。

プロセッサ・メモリ・ボトルネック

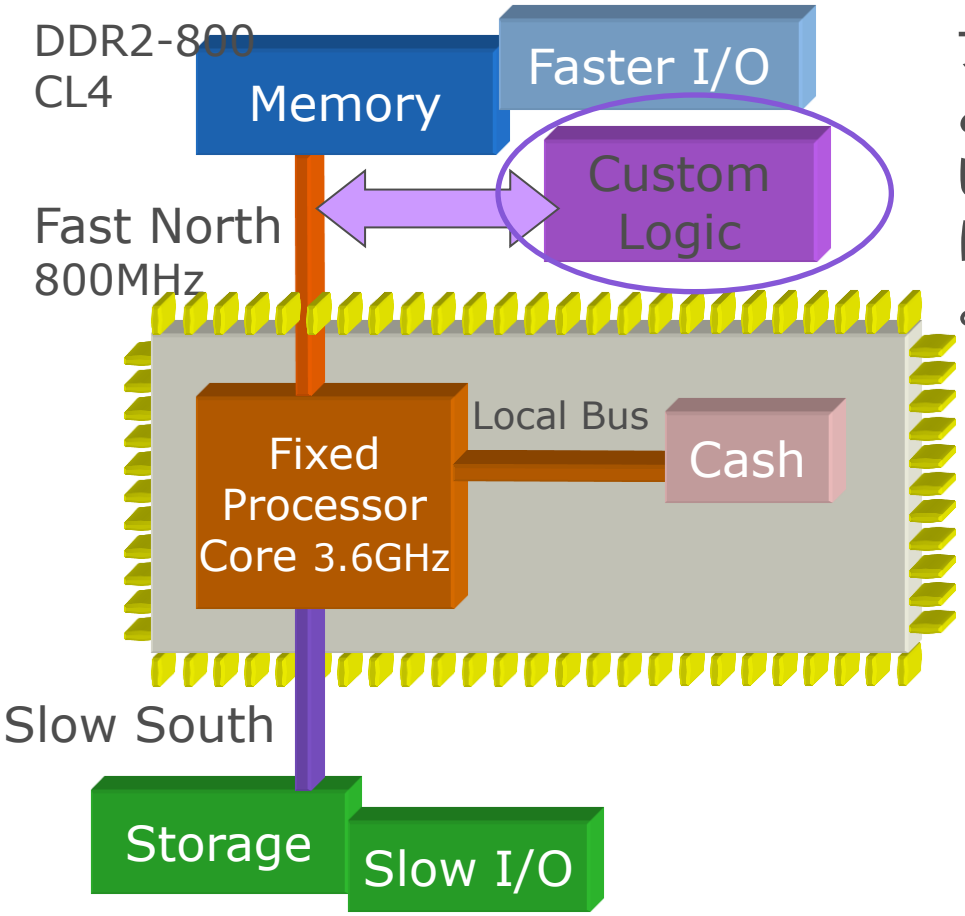


プロセッサ性能は飛躍的に高くなりましたがコンピュータシステム全体で見るとMPU性能ほどにはパフォーマンスは上がっていません

シングル・プロセッサの動作クロックが3.6GHzあっても外部高速インターフェースバス速度は1/5に低下します

高速インターフェースバスに接続されたメモリ・レイテンシでさらに1/4に低下します

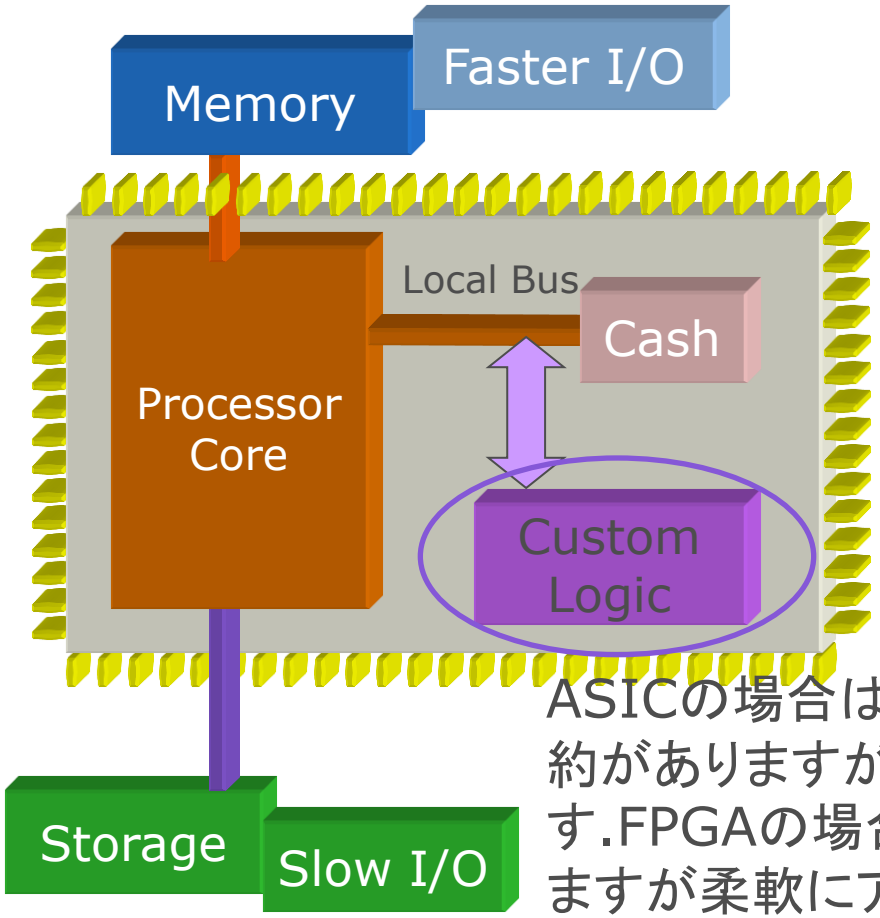
専用回路付加による一般的方法



プロセッサの仕様(ハードウェアと命令セット)は固定されていますから、性能を上げるためには外部にカスタムロジックによる専用ハードウェアを付加します。

このハードウェアが高速で動作可能であってもメモリとのRead/WriteでMPUコアとのデータのやりとりをしたのでは、膨大なオーバーヘッドが発生します。

コア内に専用回路を追加したカスタム化

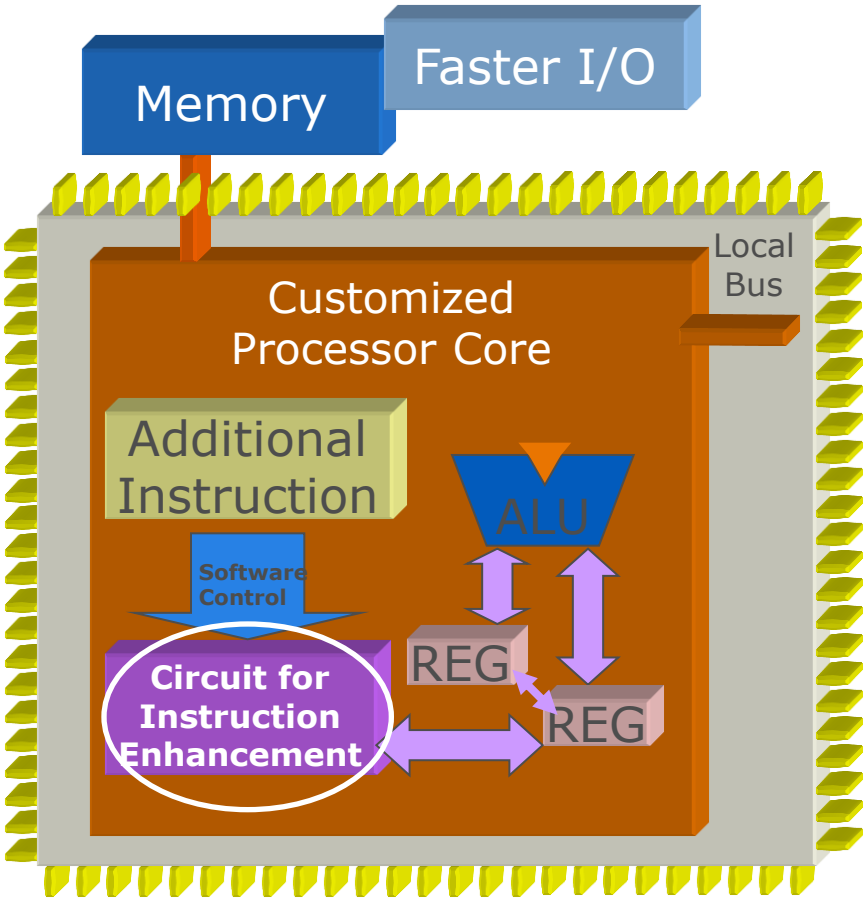


さらに専用ハードウェアをIPとしてダイの中に集積すれば高集積化,高速化,ローパワー化ができます。

このハードウェアの制御とプロセッサ間のデータのやりとりは,ダイ内部のインターフェースロジックで決定します。

ASICの場合はProcessor Coreは固定機能IPの制約がありますが性能と消費電力メリットがあります.FPGAの場合は消費電力とダイサイズが大きくなりますが柔軟にアーキテクチャ拡張が可能となります。

プロセッサの命令拡張による方法

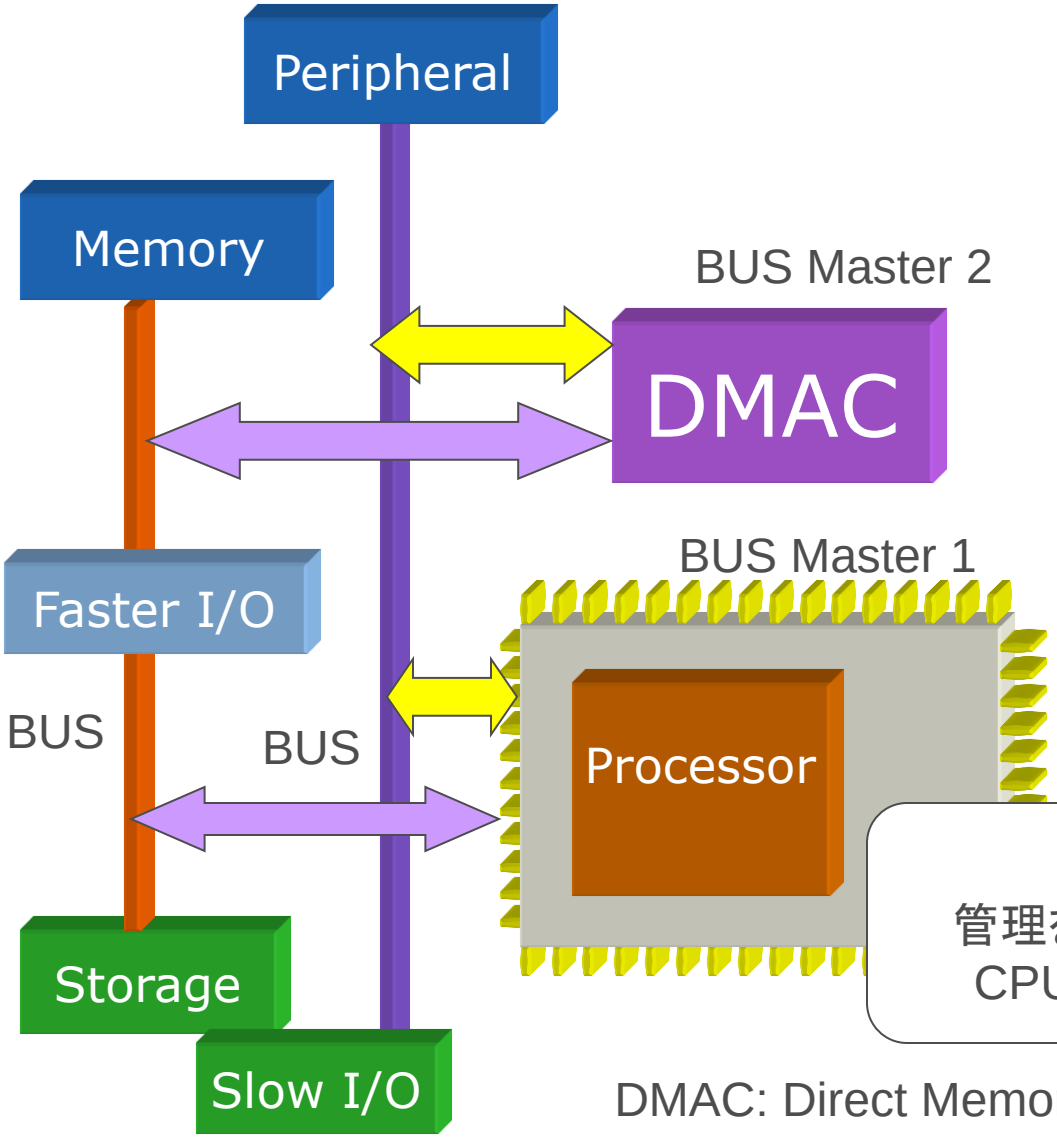


専用ハードウェア処理をプロセッサに拡張命令として実装すれば,専用回路がMPU内部アーキテクチャに実装され,ソフトウェアで制御できます。

データのやりとりは,MPUのレジスタ間となりますのでオーバーヘッドがなくなり高速,低消費電力,ダイサイズも小型化します。

FPGAでプロトタイプ化ができASIC化で性能をさらに向上できます

DMAによる方法



周辺デバイスとメモリ間のデータ転送量が多い場合CPUのプログラム制御では、オーバーヘッドが大きくなります。
その場合、DMACを使用して周辺デバイス(間)とメモリの直接データ転送を行います。
CPUは、この処理に関与せず、他の処理を行うことができます。

CPUとDMAC双方がバス管理を行うバスマスタになりますが、CPUがより高い優先度を持ちます

DMAC: Direct Memory Access Controller

PSoC3/5のDMA

PSoC3(8051-8bit)/PSoC5(ARMCortexM3-32bit)のDMAは多機能で,PSoC3ではKeil,PSoC5ではGCCコンパイラを使用しています.コンパイラによってエンディアンと上位16ビットの扱いが異なります.

これによってDMAの使い方はやや複雑になります.

まずは,Point to Point のデータ転送のタイプで,転送元と転送先のアドレスの設定の仕方を理解してください。

続いてチャネルのコンフィギュレーションの方法とTD（Transaction Descripter）のコンフィギュレーションの方法を理解するようにしてから各種のデータ転送パターンに進んでください.

これらのソースコードは、DMAウィザードツールで自動生成できますから、仕様から生成されたコードをもとに記述や理解を進めることができます。

AN52705に詳しい解説があります。DMAコンポーネントデータシートを参照してください。

PsoC3/5 のPHUB構造とDMA

周辺のH/WはPeripheral HUBを経由して相互にデータ交換

DMAはCPUを使用せずトランザクションを制御

DMAのふるまいは、
24の各チャネル
ごとに最大128
ステップのTD
(TransactionDescripter)
で記述する

HUBはBUS構造で、
CPUとDMACが
バスマスタになる
(優先度はCPUが高い)

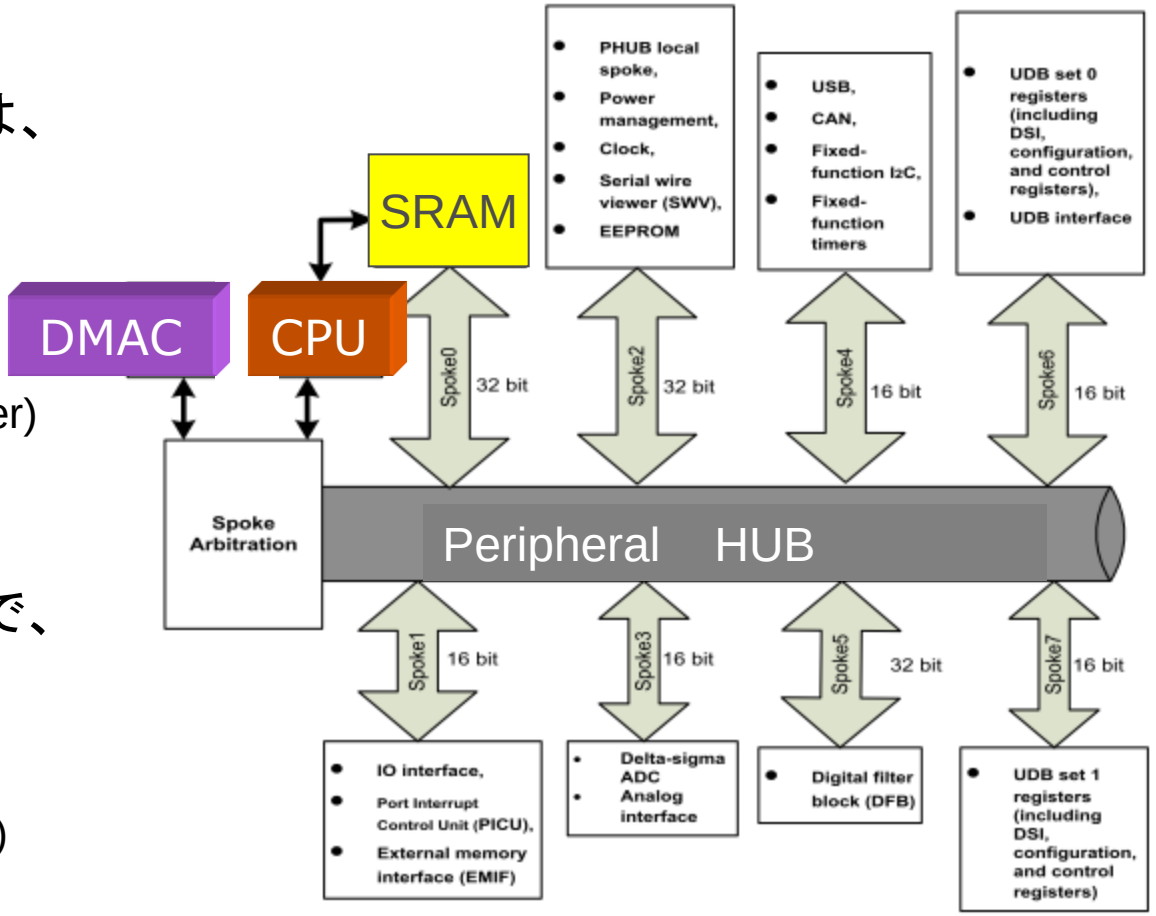


Figure 1. Peripheral HUB

Document No. 001-52705 Rev. *D

DMAデータ転送

DMAによる(CPUを介さない)4つのデータ転送パターン

メモリ－＞メモリ

例：データテーブルのコピー

メモリ－＞ペリフェラル

例：サイン波値
テーブルを
DACに転送

ペリフェラル－＞メモリ

例：マスタからの
SPIデータ

ペリフェラル－＞ペリフェラル

例：ADCのデータを
デジタルフィルタに転送

各ハードウェアはスポーク経由でPHUB接続

PHUB Spokes	Peripherals
0	SRAM
1	IOs, PICU, EMIF
2	PHUB local configuration, Power manager, Clocks, IC, SWV, EEPROM, Flash programming interface
3	Analog interface and trim, Decimator
4	USB, CAN, I^2C, Timers, Counters, and PWMs
5	DFB
6	UDBs group 1
7	UDBs group 2

DMAC

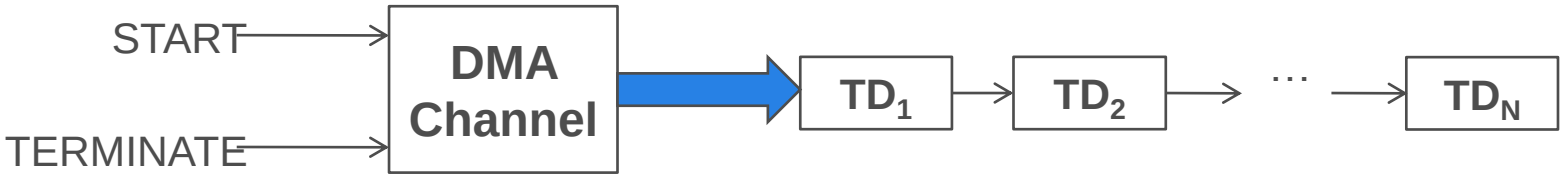
24のDMAチャネルがあり、各チャネルに8レベルのプライオリティーを設定、プライオリティーごとに処理を割当るバンド幅が定まる

(DMAチャネルとは機能概念的なもの)

各チャネルごとにその動作をTD(Transaction Descripter, DMACの制御プログラムのようなもの)で記述する。

TDの最大は128(ステップ)

トランザクションのサイズは最大64KB



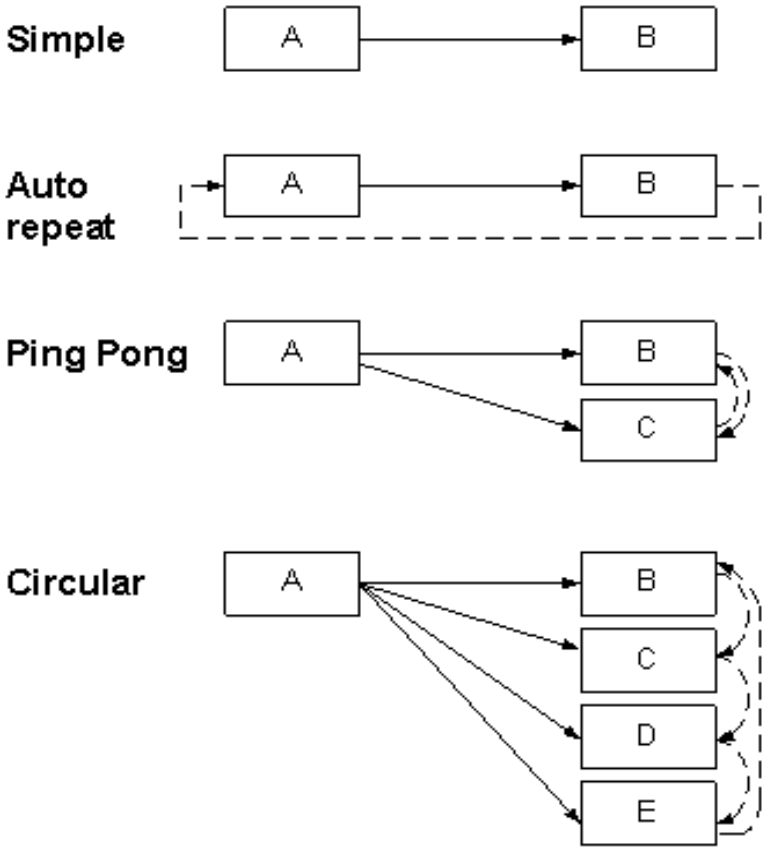
DMAプライオリティーとバンド幅割当

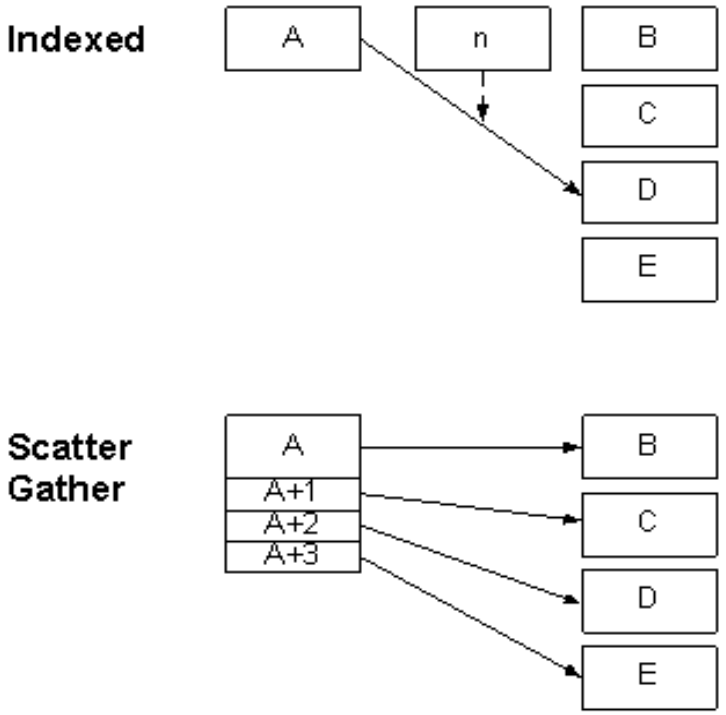
各DMAチャネルにはプライオリティーを設定します。
このプライオリティーによってバスの使用率(バンド幅)が設定されます
Priority 0と1は、どちらもBUSを100%占有しますが、
Priority 0が最優先
ですから,Priprity 1
のバス占有時でも
Prioriyty 0が
入ってきた場合には
バスのバンドはすべて
Priority 0が占有します。

Priority Level	% Bus Bandwidth
0	100.0
1	100.0
2	50.0
3	25.0
4	12.5
5	6.2
6	3.1
7	1.5

CPUとDMAは、同時に異なるスポーク経由でハードウェアにアクセスできますが、CPUのほうがDMACより高い優先度を持ちます

Trasactionモデルの例

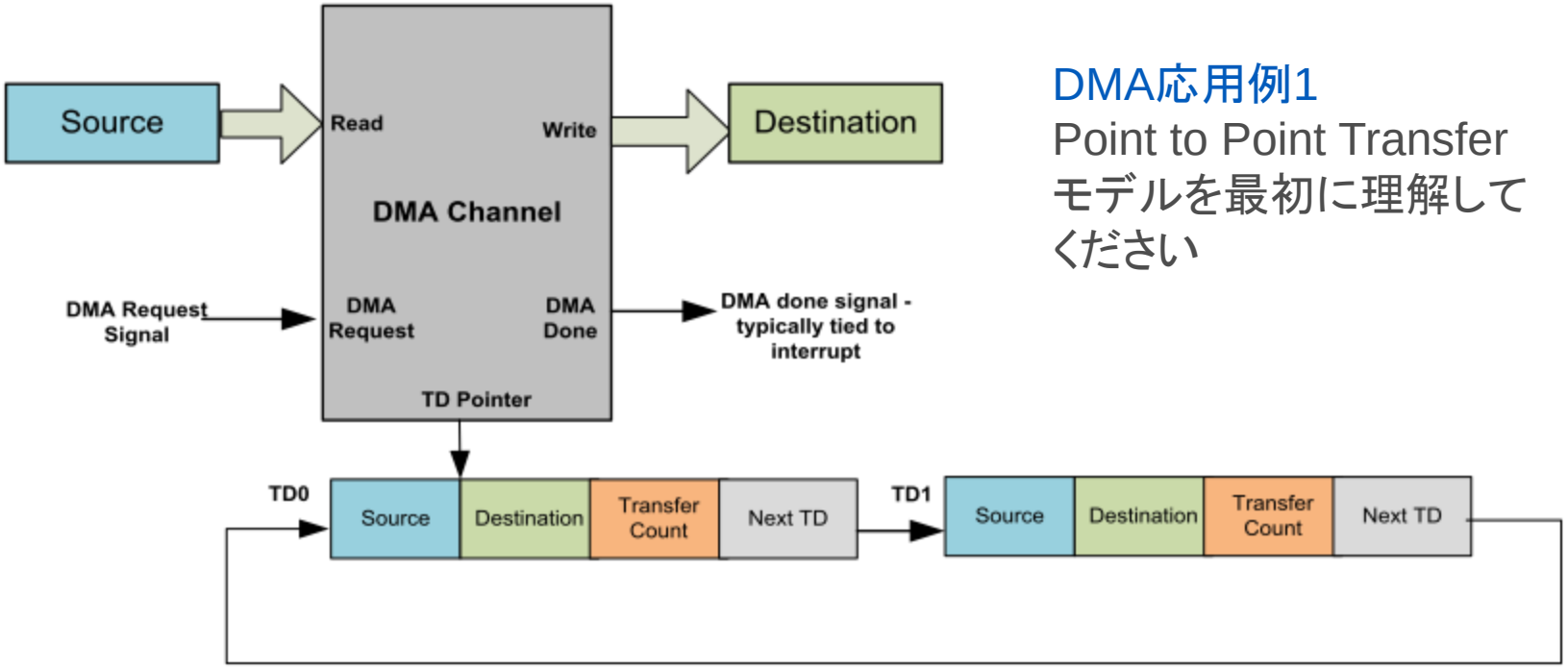




DMAの開始と終了(DMA応用例1)

DMAチャネルは、DMA要求元のHWからの信号がDMACに入力されて開始され、DMACがDMA終了信号を出力して終了する。DMA処理は一連の処理手順で実行される

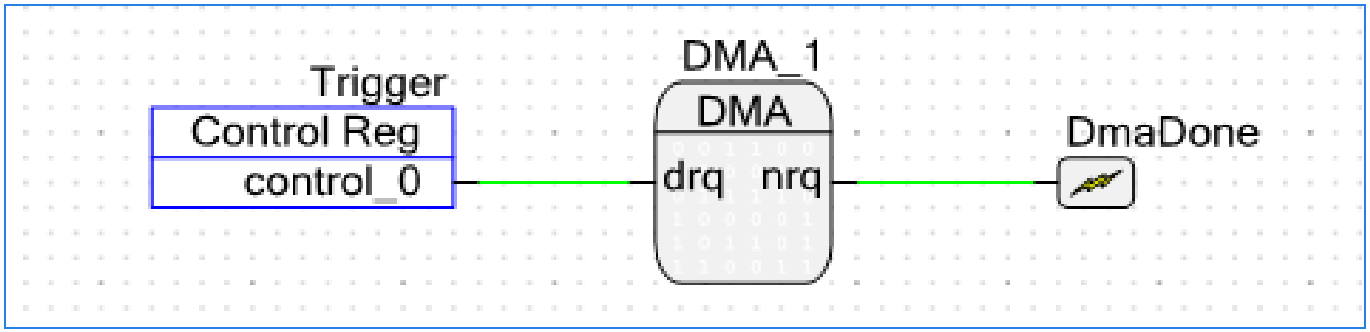
Figure 2. DMA Channel



DMA応用例1
Point to Point Transfer モデルを最初に理解してください

ハードウェアのDMA開始と終了信号

1.DMAコンポーネントを配置し,DMAスタート信号の入力結線を行う、続いてDMA終了の出力信号を出力先のコンポーネントに接続する、続いてDMAコンポーネントのプロパティーを開いてコンフィギュレーションを行う



例：DMA要求元が、コントロールレジスタ0(F/Fの出力)で、この信号がDMACのdrqをアサートし、DMA転送処理が開始される。DMA転送終了時にDMACは、nrqでCPUの割り込み信号をアサートする。CPUは、現在のレジスタ状態をスタックにPUSH（書き出）し割り込み処理プログラムを実行するためにISR(Interrupt Service Routine)にジャンプする。DMA転送の間はCPUは平行して作業中の処理を継続する。

DMA処理の手順 (Cで記述)

1. DMAチャネルのイニシャライズ
2. TDのアロケーション
3. TDのコンフィギュレーション
   TD_Handle, 転送バイト数の設定, 記述, 次のTD, Configuration Flagの設定
4. TDへのアドレスアサイン
   転送元の下位16ビットアドレス, 転送先の下位16ビットアドレス指定
5. イニシャルTDの選択
6. DMA チャネルのイネーブル
7. DMA転送要求のセット

DMAウィザード(ツール)でコードが自動生成できる

注意：PSoC3と5では、コンパイラ(3:Keil/5:GCC)が異なるので上位16ビットアドレスの取得方法が異なる。エンディアンKeil:B/GCC:L)も異なる。PSoC3用のコードはそのままでは、PSoC5では動かないケースがある。(互換の記法はある)

Channel と TDのコンフィギュレーション

Channel Configuration
Source Address(Upper 16) 転送*元*上位16ビットアドレス
Destination Address(Upper 16) 転送*先*上位16ビットアドレス
Burst Count(1 to 255) 1バーストあたり転送バイト
Request Per Burst(0 or 1) 1バースト毎に転送要求=1
First TD of channel 最初のTD
Preserve TD (0 or 1) TD Config.REGへの状態記録=0

TDの設定-2:コンフィギュレーションフラグを参照

TD Configuration
Source Address(Lower 16) 転送*元*下位16ビットアドレス
Destination Address(Lower 16) 転送*先*下位16ビットアドレス
Transfer Count 転送バイト数(0~4095)
TD Property アドレスのインクリメント,スワップ指定
Next TD 次のTDへのポイント

DMA記述例：SRAM値をDACに転送

メモリからペリフェラルに1バイトずつ合計100バイト転送

```
uint8 wave[100] = {112, 百個のデータ 120};
uint8 DMA_1_Chan;
uint8 DMA_1_TD[1];

#define DMA_1_BYTES_PER_BURST 1          //1バーストあたり1バイトの転送
#define DMA_1_REQUEST_PER_BURST 1     //1バーストごとに転送リクエスト
#define DMA_1_SRC_BASE (wave)              //転送元ソースのアドレス
#define DMA_1_DST_BASE (CYDEV_PERIPH_BASE) //転送先ディスティネーションアドレス

/*① DMA Configuration for DMA_1 */
DMA_1_Chan = DMA_1_DmaInitialize(DMA_1_BYTES_PER_BURST,
      DMA_1_REQUEST_PER_BURST,
    HI16(DMA_1_SRC_BASE), HI16(DMA_1_DST_BASE));  //( 1, 1, 0, 1)

②DMA_1_TD[0] = CyDmaTdAllocate();
②CyDmaTdSetConfiguration(DMA_1_TD[0], sizeof(wave), DMA_1_TD[0],
      TD_INC_SRC_ADR);
②CyDmaTdSetAddress(DMA_1_TD[0], LO16((uint32)wave),
      LO16((uint32)VDAC8_1_Data_PTR));
③CyDmaChSetInitialTd(DMA_1_Chan, DMA_1_TD[0]);
③CyDmaChEnable(DMA_1_Chan, 1);
```

PSoC5/GCCの場合

①DMAコンフィギュレーション

```
#define DMA_1_BYTES_PER_BURST 1          //1バーストあたり1バイトの転送
#define DMA_1_REQUEST_PER_BURST 1        //1バーストごとに転送リクエスト
#define DMA_1_SRC_BASE (wave)            //転送元ソースのアドレス
#define DMA_1_DST_BASE (CYDEV_PERIPH_BASE) //転送先ディスティネーションアドレス
```

```
/* ①DMA Configuration for DMA_1 */
DMA_1_Chan =
    DMA_1_DmaInitialize(DMA_1_BYTES_PER_BURST,
    DMA_1_REQUEST_PER_BURST,
    HI16(DMA_1_SRC_BASE),
    HI16(DMA_1_DST_BASE));  //( 1, 1, 0, 1)
```

バーストあたり転送バイト、1バーストごとの転送リクエストかどうか(YESなら1)、転送元の上位16ビットアドレス、転送先の上位16ビットアドレスを指定(今回の値は1,1,0,1になる)

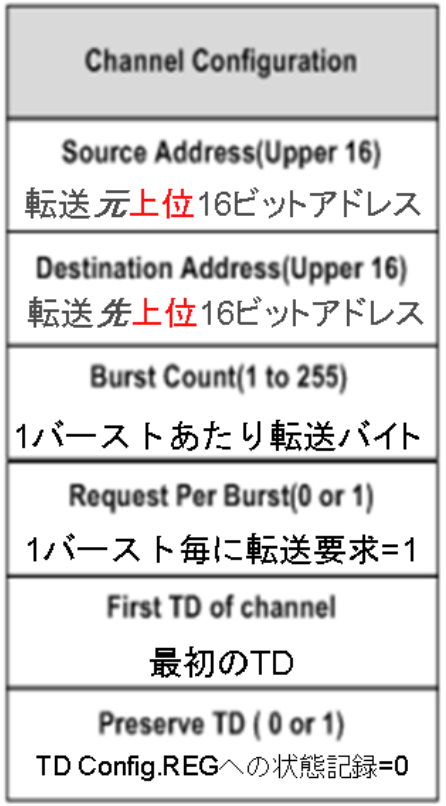
PSoC5/GCCの場合

①DMAコンフィギュレーション

```
#define DMA_1_BYTES_PER_BURST 1          //1バーストあたり1バイトの転送
#define DMA_1_REQUEST_PER_BURST 1        //1バーストごとに転送リクエスト
#define DMA_1_SRC_BASE (wave)            //転送元ソースのアドレス
#define DMA_1_DST_BASE (CYDEV_PERIPH_BASE) //転送先ディスティネーションアドレス
```

```
/* ①DMA Configuration for DMA_1 */
DMA_1_Chan = DMA_1_DmaInitialize
(
DMA_1_BYTES_PER_BURST,
DMA_1_REQUEST_PER_BURST,
    HI16(DMA_1_SRC_BASE),
HI16(DMA_1_DST_BASE)
);
//( 1, 1, 0, 1)
```

バーストあたり転送バイト、1バーストごとの転送
リクエストかどうか(YESなら1)、転送元の上位16ビットアドレ
転送先の上位16ビットアドレスを指定
(今回の値は1,1,0,1になる)



PSoC5/GCCの場合

②TDの設定-1

②DMA_1_TD[0] = CyDmaTdAllocate();

//TDのアロケーション

②CyDmaTdSetConfiguration(DMA_1_TD[0],
sizeof(wave), DMA_1_TD[0], TD_INC_SRC_ADR);

//TDのコンフィギュレーション(引き数)

//CyDmaTdSetConfiguration(td,転送バイト数,次のTD,
コンフィギュレーションフラグ)

②CyDmaTdSetAddress(DMA_1_TD[0],
LO16((uint32)wave),
LO16((uint32)VDAC8_1_Data_PTR));

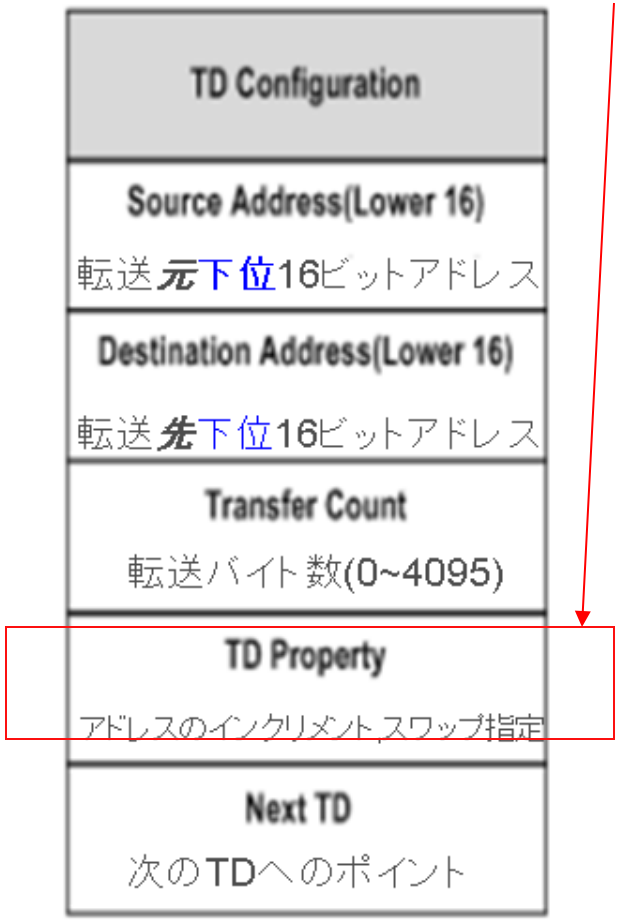
//CyDmaTdSetAddress(td,転送元下位16ビットアドレス.
転送先下位16ビットアドレス)

PSoC5/GCCの場合

②TDの設定-1

②DMA_1_TD[0] = CyDmaTdAllocate();
//TDのアロケーション
②CyDmaTdSetConfiguration(
DMA_1_TD[0],
sizeof(wave),
DMA_1_TD[0],
TD_INC_SRC_ADR);
//TDのコンフィギュレーション(引き数)
//CyDmaTdSetConfiguration(
td,転送バイト数,次のTD,
コンフィギュレーションフラグ)
②CyDmaTdSetAddress(
DMA_1_TD[0], LO16((uint32)wave),
LO16((uint32)VDAC8_1_Data_PTR));
//CyDmaTdSetAddress(
td,転送元下位16ビットアドレス,
転送先下位16ビットアドレス)

TDの設定-2:コンフィギュレーションフラグを参照



PSoC5/GCCの場合

②TDの設定-2：コンフィギュレーションフラグ

.....sizeof(wave), DMA_1_TD[0], TD_INC_SRC_ADR);

REF:DMAコンポーネントデータシート

TD_SWAP_EN	エンディアン変更を実行します。
TD_SWAP_SIZE4	スワップサイズ = 4 バイト
TD_AUTO_EXEC_NEXT	現在のTDが完了すると、自動的にチェーンの次のTDがトリガされます。
TD_TERMIN_EN	「trq」入力ラインでポジティブエッジが発生すると、このTDを終了します。ポジティブエッジはバースト中に発生しなければなりません。その場合だけ、DMACが検知します。
DMA__TD_TERMOUT_EN	このTDが完了すると、TERMOUT信号がパルスを生成します。注：このオプションはインスタンスに特有で、インスタンス名のつぎに下線が2文字分続きます。この例では、インスタンス名はDMAです。
TD_INC_DST_ADR	バーストの各データトランザクションのサイズに応じて、DST_ADRが増分します。
TD_INC_SRC_ADR	バーストの各データトランザクションのサイズに応じて、SRC_ADRが増分します。

② TDの設定-3 下位16ビットアドレス

②DMA_1_TD[0] = CyDmaTdAllocate();

②CyDmaTdSetConfiguration(DMA_1_TD[0], sizeof(wave), DMA_1_TD[0], TD_INC_SRC_ADR);

②CyDmaTdSetAddress(DMA_1_TD[0], LO16((uint32)wave), LO16((uint32)*VDAC8_1*_Data_PTR));

// CyDmaTdSetAddress(td,転送元下位16ビットアドレス. 転送先下位16ビットアドレス)

*VDAC8_1*_Data_PTR

*コンポーネント・インスタンス*に_Data_PTR をつける

PSoC5/GCCの場合

③ チャネルのイネーブル

```
③CyDmaChSetInitialTd(DMA_1_Chan,
    DMA_1_TD[0]);
//DMAチャネルとTDのイニシャライズ
③CyDmaChEnable(DMA_1_Chan, 1);
// DMAチャネルのイネーブル化
```

PSoC5/GCCの場合

DMAのソースとディスティネーション1-5

REF:DMAコンポーネントデータシート

RamからRamへ

```
uint8 DMA_1_Chan;
uint8 DMA_1_TD[1];
/* DMA Configuration for DMA_1 */
#define DMA_1_BYTES_PER_BURST 1
#define DMA_1_REQUEST_PER_BURST 1
#define DMA_1_SRC_BASE (buffer1)
#define DMA_1_DST_BASE (buffer2)
DMA_1_Chan = DMA_1_DmaInitialize(DMA_1_BYTES_PER_BURST,
    DMA_1_REQUEST_PER_BURST,
HI16(DMA_1_SRC_BASE), HI16(DMA_1_DST_BASE));
DMA_1_TD[0] = CyDmaTdAllocate();
CyDmaTdSetConfiguration(DMA_1_TD[0], 128, DMA_INVALID_TD,
    DMA_1__TD_TERMOUT_EN | TD_INC_SRC_ADR | TD_INC_DST_ADR);
CyDmaTdSetAddress(DMA_1_TD[0], LO16((uint32)buffer1),
    LO16((uint32)buffer2));
CyDmaChSetInitialTd(DMA_1_Chan, DMA_1_TD[0]);
CyDmaChEnable(DMA_1_Chan, 1)
```

PSoC5/GCCの場合

DMAのソースとディスティネーション1-3

REF:DMAコンポーネントデータシート

RamからRamへ

```
uint8 DMA_1_Chan;
uint8 DMA_1_TD[1];
/* DMA Configuration for DMA_1 */
#define DMA_1_BYTES_PER_BURST 16
#define DMA_1_REQUEST_PER_BURST 1
#define DMA_1_SRC_BASE (CYDEV_SRAM_BASE)
#define DMA_1_DST_BASE (CYDEV_SRAM_BASE)
DMA_1_Chan = DMA_1_DmaInitialize(DMA_1_BYTES_PER_BURST,
    DMA_1_REQUEST_PER_BURST,
HI16(DMA_1_SRC_BASE), HI16(DMA_1_DST_BASE));
DMA_1_TD[0] = CyDmaTdAllocate();
CyDmaTdSetConfiguration(DMA_1_TD[0], 128, DMA_INVALID_TD,
    DMA_1__TD_TERMOUT_EN | TD_INC_SRC_ADR | TD_INC_DST_ADR);
CyDmaTdSetAddress(DMA_1_TD[0], LO16((uint32)memory1),
    LO16((uint32)memory2));
CyDmaChSetInitialTd(DMA_1_Chan, DMA_1_TD[0]);
CyDmaChEnable(DMA_1_Chan, 1);
```

PSoC3/Keilの場合

DMAのソースとディスティネーション2-5

REF:DMAコンポーネントデータシート

フラッシュからDACへ:

```
uint8 DMA_1_Chan;
uint8 DMA_1_TD[1];
/* DMA Configuration for DMA_1 */
#define DMA_1_BYTES_PER_BURST 1
#define DMA_1_REQUEST_PER_BURST 1
#define DMA_1_SRC_BASE (FlashMem)
#define DMA_1_DST_BASE (CYDEV_PERIPH_BASE)
DMA_1_Chan = DMA_1_DmaInitialize(DMA_1_BYTES_PER_BURST,
    DMA_1_REQUEST_PER_BURST,
HI16(DMA_1_SRC_BASE), HI16(DMA_1_DST_BASE));
DMA_1_TD[0] = CyDmaTdAllocate();
CyDmaTdSetConfiguration(DMA_1_TD[0], 64, DMA_INVALID_TD,
    DMA_1__TD_TERMOUT_EN | TD_INC_SRC_ADR);
CyDmaTdSetAddress(DMA_1_TD[0], LO16((uint32)FlashMem),
    LO16((uint32)VDAC8_1_Data_PTR));
CyDmaChSetInitialTd(DMA_1_Chan, DMA_1_TD[0]);
CyDmaChEnable(DMA_1_Chan, 1);
```

PSoC5/GCCの場合

DMAのソースとディスティネーション1-3

REF:DMAコンポーネントデータシート

フラッシュからDACへ

```
uint8 DMA_1_Chan;
uint8 DMA_1_TD[1];
/* DMA Configuration for DMA_1 */
#define DMA_1_BYTES_PER_BURST 1
#define DMA_1_REQUEST_PER_BURST 1
#define DMA_1_SRC_BASE (CYDEV_FLS_BASE)
#define DMA_1_DST_BASE (CYDEV_PERIPH_BASE)
DMA_1_Chan = DMA_1_DmaInitialize(DMA_1_BYTES_PER_BURST,
    DMA_1_REQUEST_PER_BURST,
HI16(DMA_1_SRC_BASE), HI16(DMA_1_DST_BASE));
DMA_1_TD[0] = CyDmaTdAllocate();
CyDmaTdSetConfiguration(DMA_1_TD[0], 64, DMA_INVALID_TD,
    TD_INC_SRC_ADR);
CyDmaTdSetAddress(DMA_1_TD[0], LO16((uint32)fFashMem),
    LO16((uint32)VDAC8_1_Data_PTR));
CyDmaChSetInitialTd(DMA_1_Chan, DMA_1_TD[0]);
CyDmaChEnable(DMA_1_Chan, 1);
```

PSoC3/Keilの場合

DMAのソースとディスティネーション3-5

REF:DMAコンポーネントデータシート

境界の条件

単一のDMAチャネルは64Kの境界を越えることができません。従って、DMAが64Kの境界を越えた場合、動作は警告なしに失敗します。これはPSoC 5にとってのみ重要です。その理由は、PSoC 3デバイスのメモリ空間はフラッシュの64KやRamの8Kを超えることがないからです。

このデータ構成を使用する場合、64 Kの境界を越えないように注意してください。それには次のキーワードを利用することができます。

__attribute__((section()))

__attribute__((aligned()))

これらのキーワードはいずれも、GCCの「Specifying Attributes of Variables（変数の属性指定）」セクションのヘルプで説明されています。変数を特定のセクションと位置に表示したい場合は、sectionのキーワードを使用します。コンパイラを使って変数を特定の境界に置くには、alignedのキーワードを使用します。

PSoC5/GCCの場合

DMAのソースとディスティネーション3-3

REF:DMAコンポーネントデータシート

フラッシュからRamへ

```
uint8 DMA_1_Chan;
uint8 DMA_1_TD[1];
/* DMA Configuration for DMA_1 */
#define DMA_1_BYTES_PER_BURST 1
#define DMA_1_REQUEST_PER_BURST 1
#define DMA_1_SRC_BASE (CYDEV_FLS_BASE)
#define DMA_1_DST_BASE (CYDEV_SRAM_BASE)
DMA_1_Chan = DMA_1_DmaInitialize(DMA_1_BYTES_PER_BURST,
    DMA_1_REQUEST_PER_BURST,
HI16(DMA_1_SRC_BASE), HI16(DMA_1_DST_BASE));
DMA_1_TD[0] = CyDmaTdAllocate();
CyDmaTdSetConfiguration(DMA_1_TD[0], 128, DMA_INVALID_TD,
    DMA_1__TD_TERMOUT_EN | TD_INC_SRC_ADR | TD_INC_DST_ADR);
CyDmaTdSetAddress(DMA_1_TD[0], LO16((uint32)buf1), LO16((uint32)buf2));
CyDmaChSetInitialTd(DMA_1_Chan, DMA_1_TD[0]);
CyDmaChEnable(DMA_1_Chan, 1);
```

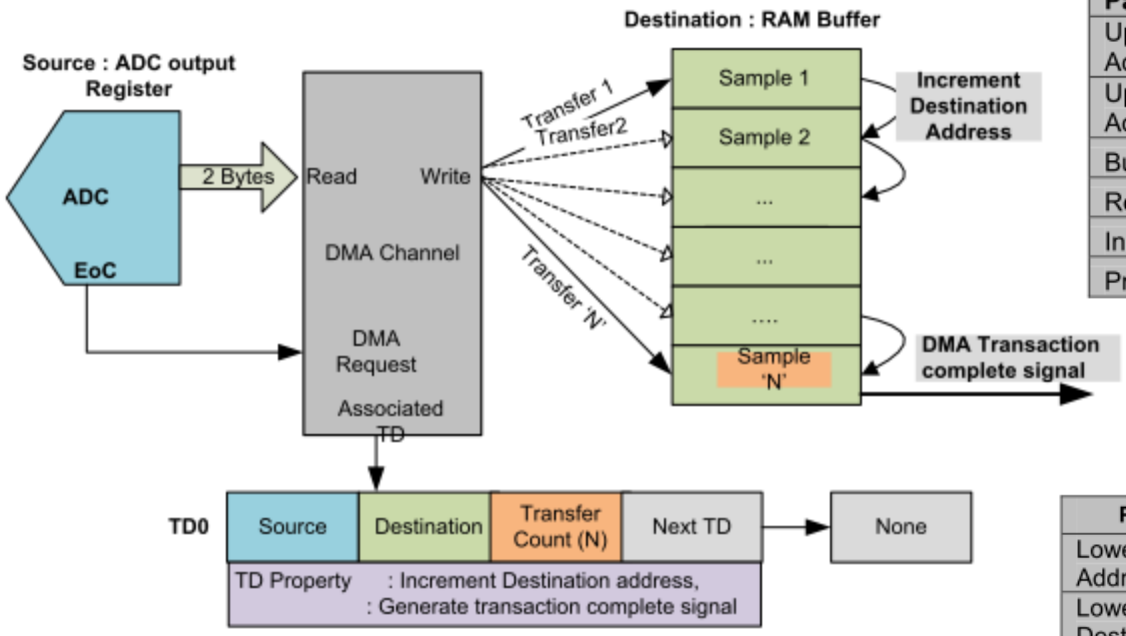
PSoC3/Keilの場合

DMAの応用例2

Example 2: Point-to-Array Transfer

This 16-bit ADC data buffering example shows how use DMA to do a point-to-array transfer.

Figure 6. Point to Array Transfer Block Diagram



Channel Configuration

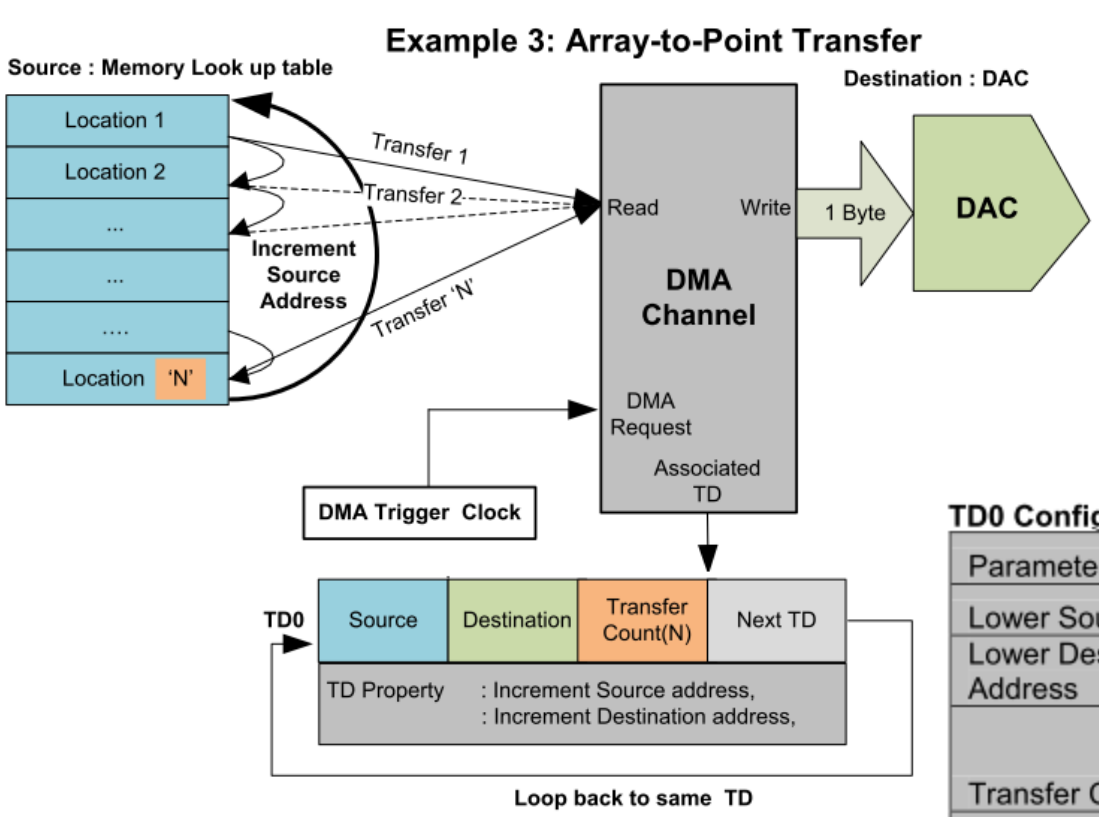
Parameter	Project Setting
Upper Source Address	HI16(CYDEV_PERIPH_BASE)/ HI16(ADC_DEC_OUTSAMP_PTR)
Upper Destination Address	HI16(CYDEV_SRAM_BASE)
Burst Count	2 Bytes
Request Per Burst	True (1)
Initial TD	TD0
Preserve TD	Yes(1)

TD0 Configuration

Parameter	Project Setting
Lower Source Address	LO16 (ADC_DEC_OUTSAMP_PTR)
Lower Destination Address	LO16 (adc_sampleArray)
Transfer Count	N×2 (No. of samples × Bytes per sample)
TD property	▪ Increment Destination Address ▪ Generate DMA done event ▪ Swap Enable required for PSoC3
Next TD	None/repeat to same TD

REF:AN52705D

DMAの応用例3



Channel Configuration

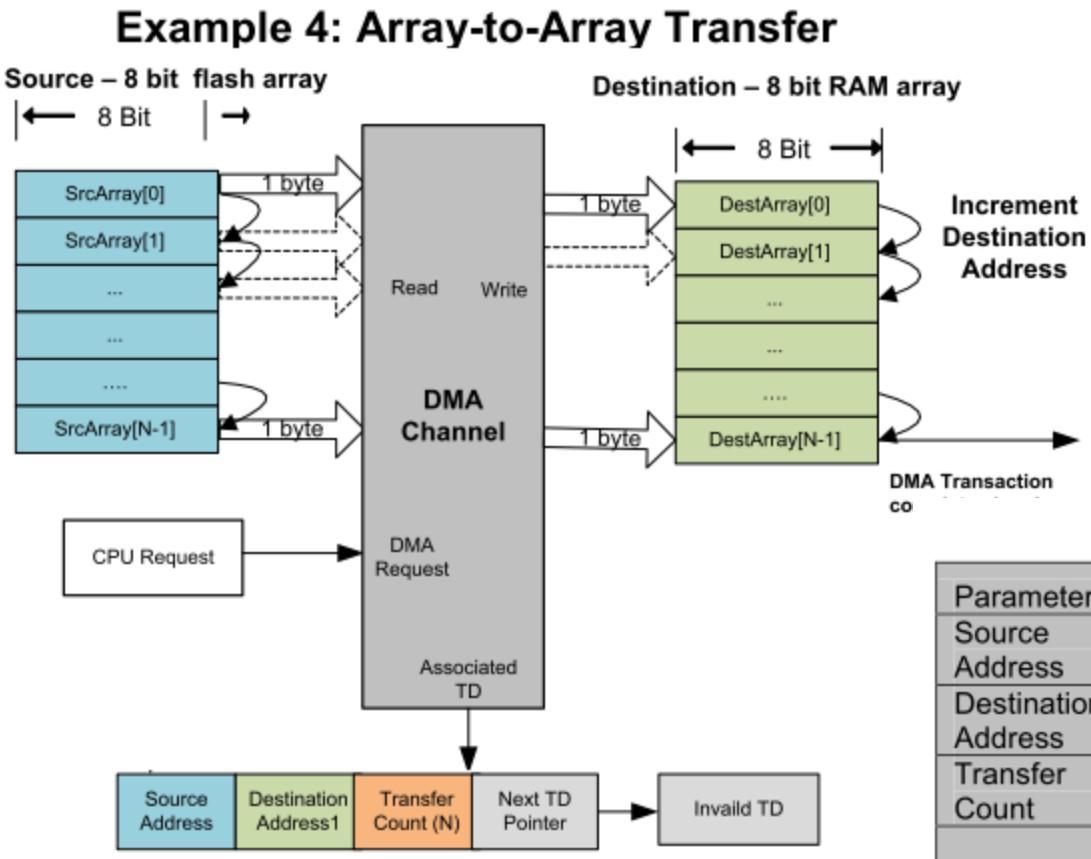
Parameter	Project Setting
Upper Source Address	-HI16(CYDEV_FLS_BASE) for PSoC 3 - HI16 (&sineTable) for PSoC 5
Upper Destination Address	HI16(CYDEV_PERIPH_BASE)/ HI16(VDAC8_DATA_PTR)
Burst Count	1 Byte
Request Per Burst	True (1)
Initial TD	TD0
Preserve TD	Yes(1)

TD0 Configuration

Parameter	Project Setting
Lower Source Address	LO16(&sineTable)
Lower Destination Address	LO16(VDAC8_DATA_PTR)
Transfer Count	N (No. of entries in the sine look up table x no. of bytes per entry)
TD property	▪ Increment Source Address
Next TD	Loop back to the same TD again

REF:AN52705D

DMAの応用例4



Channel Configuration

Parameter	Project Setting
Upper Source Address	-HI16(CYDEV_FLS_BASE) , for PSoC 3 -HI16(&sourceArray) for PSoC 5
Upper Destination Address	HI16(CYDEV_SRAM_BASE)
Burst Count	1 Byte
Request Per Burst	False
Initial TD	TD0
Preserve TD	0

TD0 Configuration

Parameter	Project Setting
Source Address	LO16(&sourceArray)
Destination Address	LO16(&destinationArray)
Transfer Count	N (8 bytes)
TD property	▪ TD_INC_SRC_ADR ▪ TD_INC_DST_ADR DMA__TD_TERMOUT_EN
Next TD	None (Terminate TD – 0xFE)

REF:AN52705D

DMAの応用例5

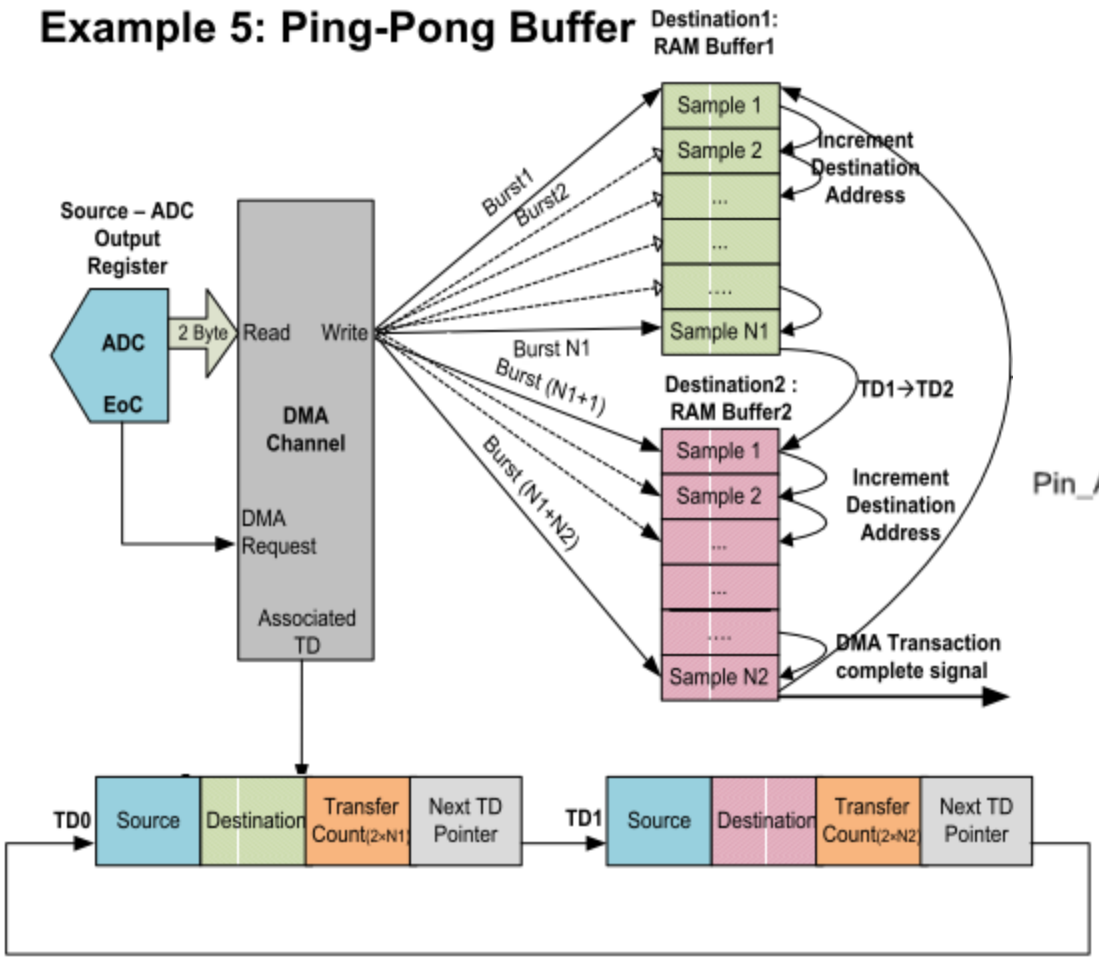
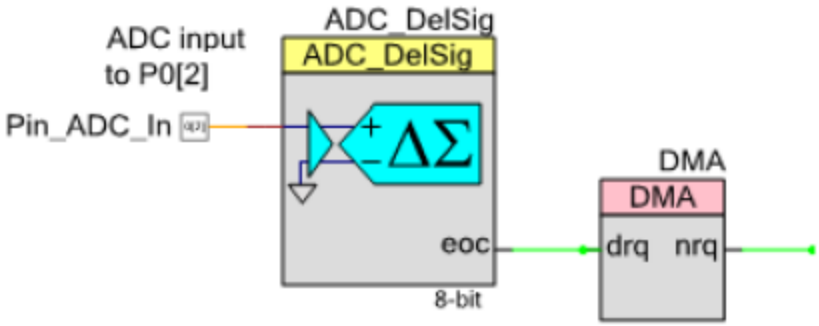


Figure 14. ADC-DMA Memory



REF:AN52705D

DMAの応用例5

Channel Configuration

The following table shows the channel configuration, which is similar to that in Example 2.

Parameter	Setting
Upper Source Address	HI16(ADC_DEC_OUTSAMP_PTR)
Upper Destination Address	HI16(CYDEV_SRAM_BASE)
Burst Count	2 byte
Request Per Burst	True (1)
Initial TD	TD0
Preserve TD	Yes(1)

Transaction 1 uses TD0, and Transaction 2 uses TD1. To attach the transactions to each other, the **Next TD** parameter of TD0 is set to TD1 and vice versa.

TD0 Configuration

Parameter	Project Setting
Lower Source Address	LO16(ADC_DEC_OUTSAMP_PTR)
Lower Destination Address	LO16(adc_sampleArray1)
Transfer Count	N1×2 (No. of samples × Bytes per sample)
TD property	▪ Increment Destination Address ▪ Generate DMA done event ▪ Swap Enable required for PSoC 3
Next TD	TD1

TD1 Configuration

Parameter	Project Setting
Lower Source Address	LO16(ADC_DEC_OUTSAMP_PTR)
Lower Destination Address	LO16(adc_sampleArray2)
Transfer Count	N2×2 (No. of samples × Bytes per sample)
TD property	▪ Increment Destination Address ▪ Generate DMA done event ▪ Swap Enable required for PSoC 3
Next TD	TD0

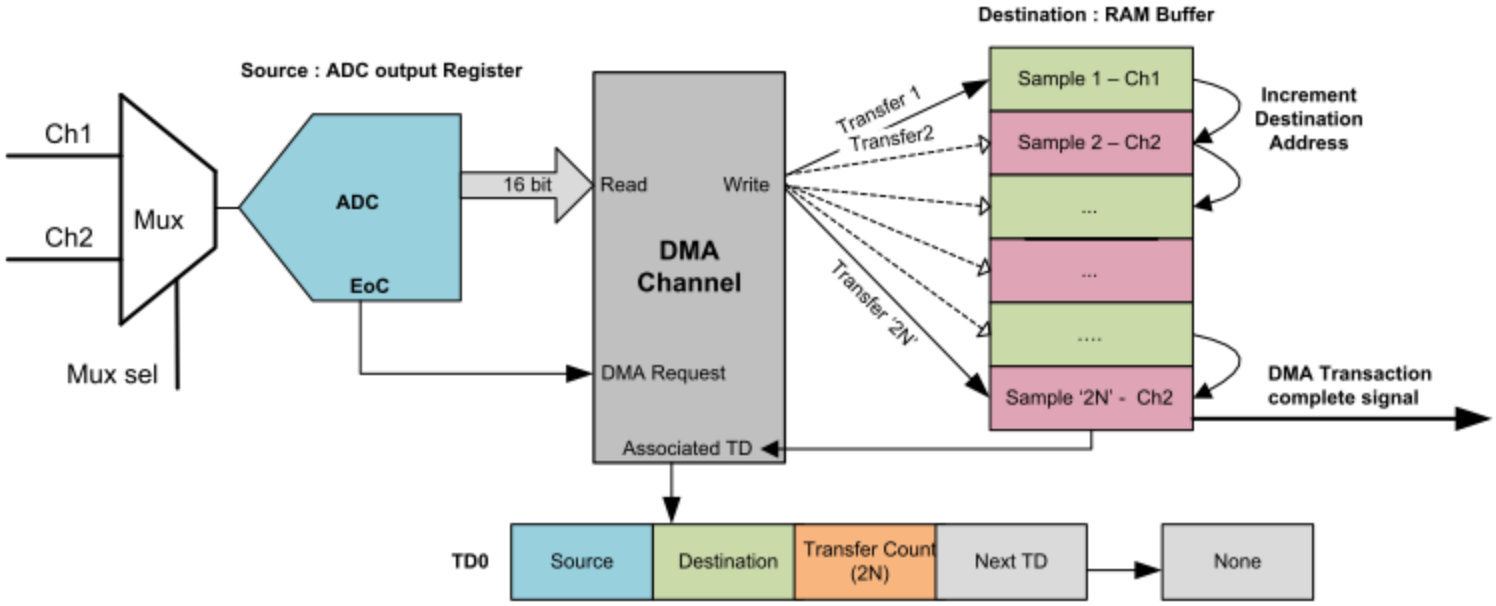
REF:AN52705D

DMAの応用例6

Example 6: Multiplexed Data Buffering

This example shows how to collect ADC data when the input to ADC is multiplexed. If a single DMA channel is used to collect multiplexed ADC data, the buffered data will be as shown in Figure 155 with channel 1 and channel 2 data combined. But you can also keep the channel 1 and channel 2 data separate.

Figure 15. Multiplexed Data Buffering Using Single Channel

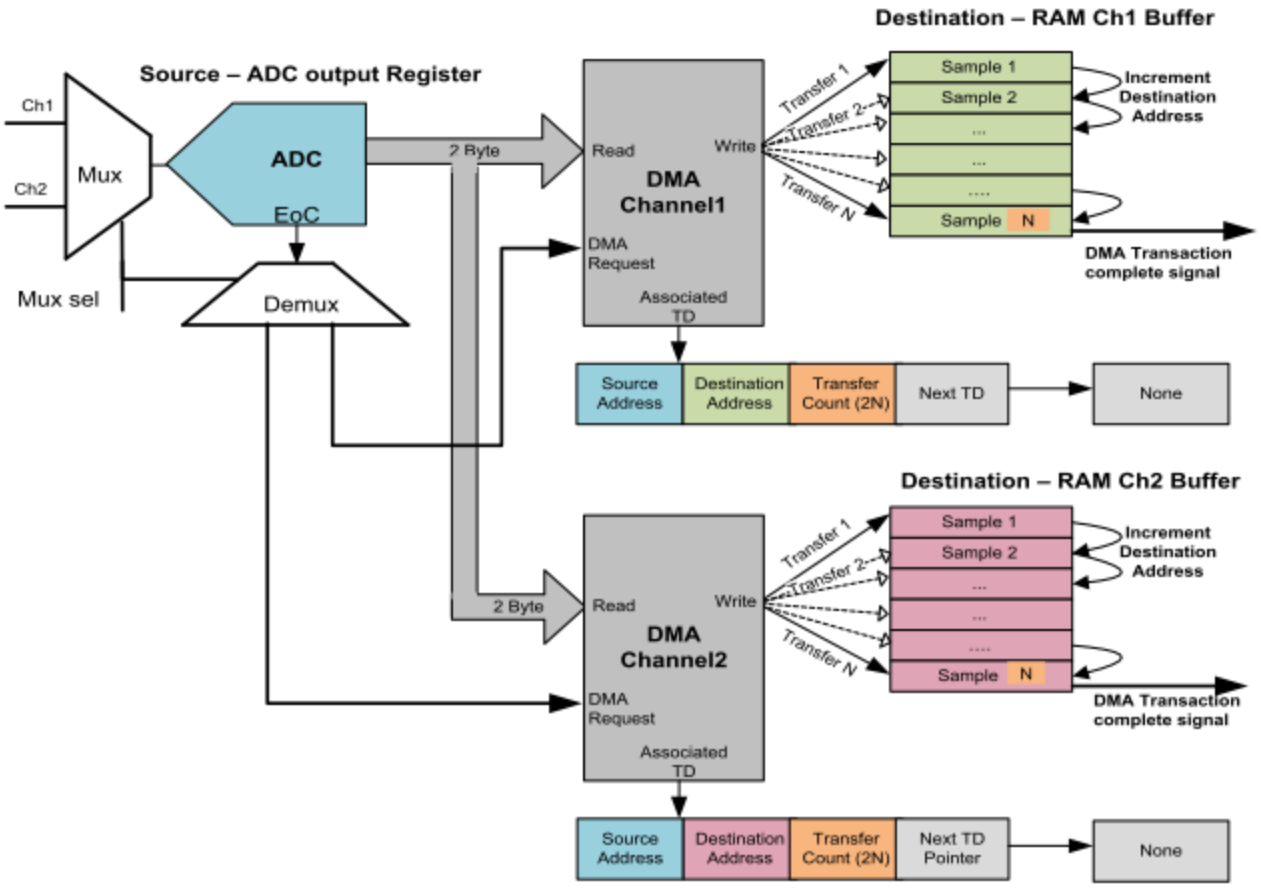


REF:AN52705D

DMAの応用例6

DMA channels cannot change between two transactions until the entire transfer count is finished. For this reason, multiple TDs will not work as well.. Figure 16 shows how you can use multiple DMA channels and multiplex them to move the multiplexed channel data into separate buffers.

Figure 16. Multiple DMA Channels for Buffering Multiplexed ADC Data



REF:AN52705D

4.4.2 DMA Features

- 24 DMA channels
- Each channel has one or more transaction descriptors (TD) to configure channel behavior. Up to 128 total TDs can be defined
- TDs can be dynamically updated
- Eight levels of priority per channel
- Any digitally routable signal, the CPU, or another DMA channel, can trigger a transaction
- Each channel can generate up to two interrupts per transfer
- Transactions can be stalled or canceled
- Supports transaction size of infinite or 1 to 64 KB
- TDs may be nested and/or chained for complex transactions

4.4.3 Priority Levels

The CPU always has higher priority than the DMA controller when their accesses require the same bus resources. Due to the system architecture, the CPU can never starve the DMA. DMA channels of higher priority (lower priority number) may interrupt current DMA transfers. In the case of an interrupt, the current transfer is allowed to complete its current transaction. To ensure latency limits when multiple DMA accesses are requested simultaneously, a fairness algorithm guarantees an interleaved minimum percentage of bus bandwidth for priority levels 2 through 7. Priority levels 0 and 1 do not take part in the fairness algorithm and may use 100 percent of the bus bandwidth. If a tie occurs on two DMA requests of the same priority level, a simple round robin method is used to evenly share the allocated bandwidth. The round robin allocation can be disabled for each DMA channel, allowing it to always be at the head of the line.

Priority levels 2 to 7 are guaranteed the minimum bus bandwidth shown in Table 4-7 after the CPU and DMA priority levels 0 and 1 have satisfied their requirements.

Table 4-7. Priority Levels

Priority Level	% Bus Bandwidth
0	100.0
1	100.0
2	50.0
3	25.0
4	12.5
5	6.2
6	3.1
7	1.5

When the fairness algorithm is disabled, DMA access is granted based solely on the priority level; no bus bandwidth guarantees are made.

4.4.4 Transaction Modes Supported

The flexible configuration of each DMA channel and the ability to chain multiple channels allow the creation of both simple and complex use cases. General use cases include, but are not limited to:

4.4.4.1 Simple DMA

In a simple DMA case, a single TD transfers data between a source and sink (peripherals or memory location). The basic timing diagrams of DMA read and write cycles are shown in Figure 4-1. For more description on other transfer modes, refer to the Technical Reference Manual.

Figure 4-1. DMA Timing Diagram

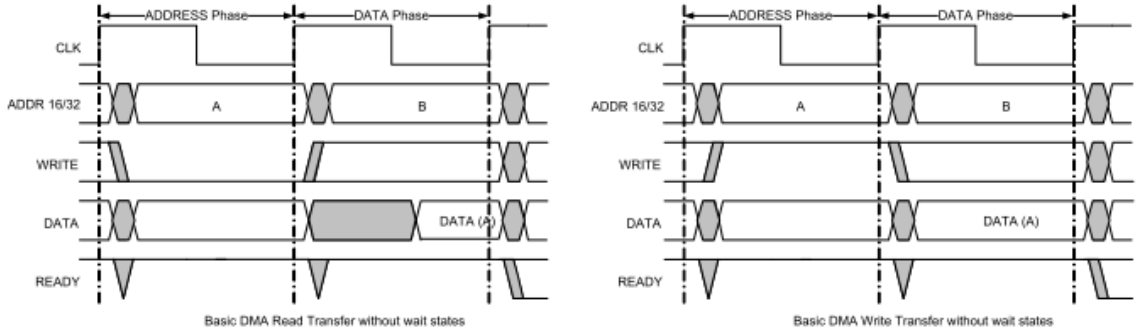
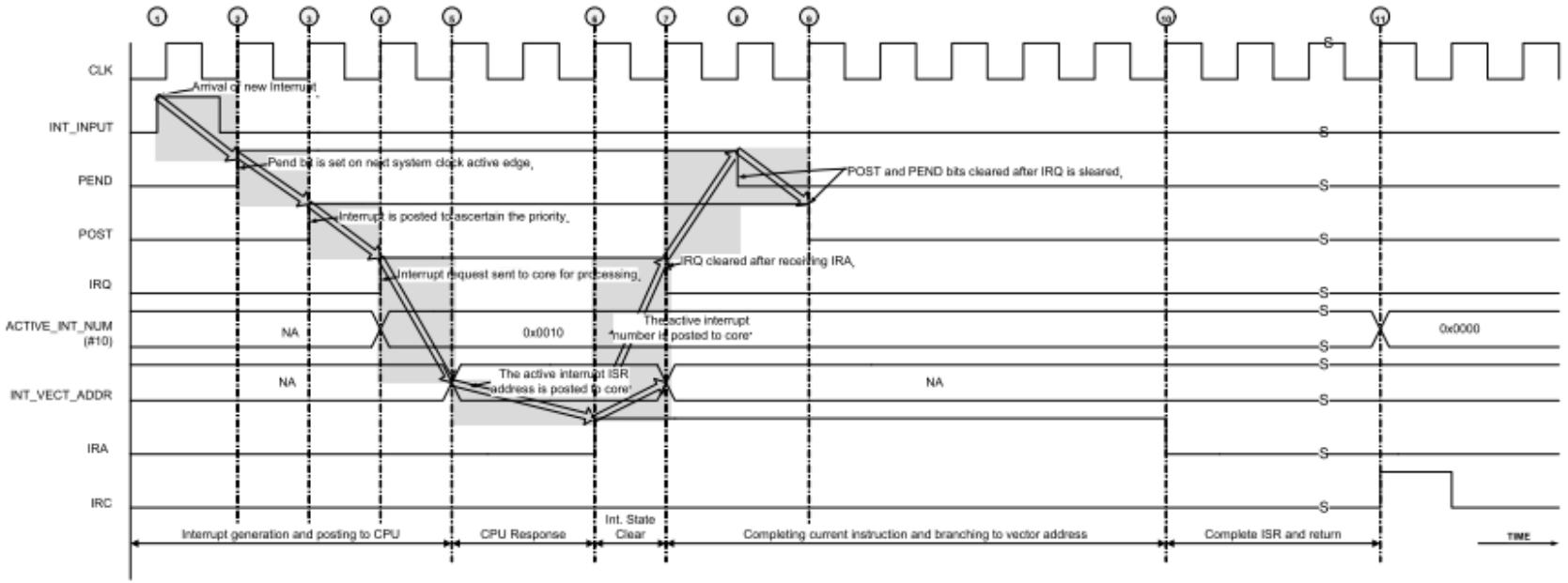


Figure 4-2. Interrupt Processing Timing Diagram



Notes

- 1: Interrupt triggered asynchronous to the clock
- 2: The PEND bit is set on next active clock edge to indicate the interrupt arrival
- 3: POST bit is set following the PEND bit
- 4: Interrupt request and the interrupt number sent to CPU core after evaluation priority (Takes 3 clocks)
- 5: ISR address is posted to CPU core for branching
- 6: CPU acknowledges the interrupt request
- 7: ISR address is read by CPU for branching
- 8, 9: PEND and POST bits are cleared respectively after receiving the IRA from core
- 10: IRA bit is cleared after completing the current instruction and starting the instruction execution from ISR location (Takes 7 cycles)
- 11: IRC is set to indicate the completion of ISR, Active int. status is restored with previous status

The total interrupt latency (ISR execution)
= POST + PEND + IRQ + IRA + Completing current instruction and branching
= 1+1+1+2+7 cycles
= 12 cycles

Memo

フォローアップURL

http://mikami.a.la9.jp/meiji/meiji.htm

担当講師

ミカミ設計コンサルティング

三上廉司(みかみれんじ)

Renji_Mikami(at_mark)nifty.com