注意 : この日本語版文書は参考資料としてご利用ください。最新情報は必ずオリジナルの英語版をご参照願います。

セクション 22. ダイレクト メモリアクセス (DMA)

ハイライト

本セクションには以下の主要項目を記載しています。

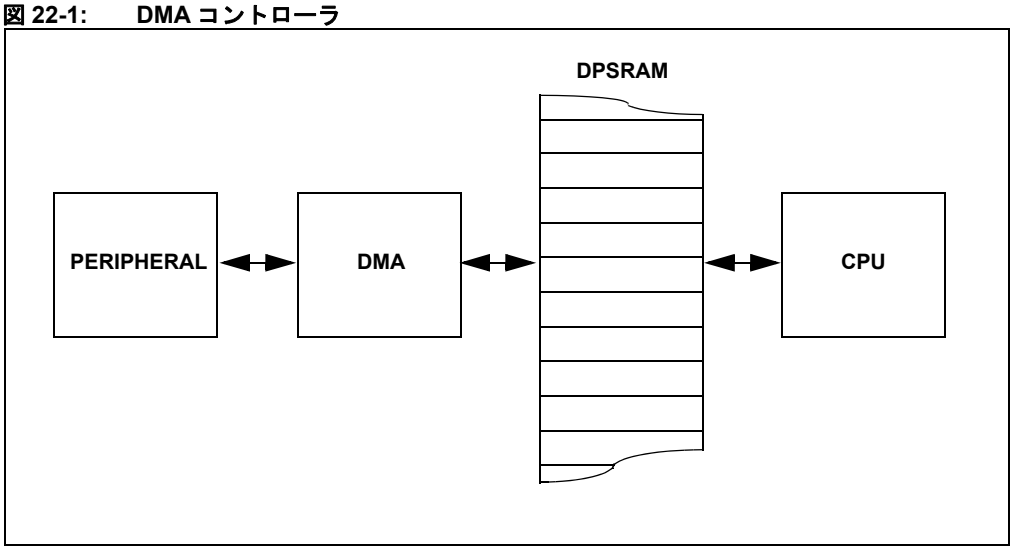
22.1 はじめに 22-2
22.2 DMA レジスタ 22-3
22.3 DMA のブロック図 22-12
22.4 DMA データ転送 22-13
22.5 DMA の設定 22-15
22.6 DMA 動作モード 22-21
22.7 DMA 転送の開始 22-46
22.8 DMA チャンネル アービトレーションとオーバーラン 22-48
22.9 デバッグサポート 22-49
22.10 データ書き込みコリジョン 22-50
22.11 省電力モード時の動作 22-51
22.12 設計のヒント 22-52
22.13 レジスタマップ 22-54
22.14 関連アプリケーション ノート 22-57
22.15 改訂履歴 22-58

22.1 はじめに

ダイレクト メモリ アクセス (DMA) コントローラは、マイクロチップ社の高性能 16 ビット デジタル シグナル コントローラ (DSC) ファミリにおける重要なサブシステムです。このサブシステムにより、CPU 時間を消費せずメモリと周辺モジュール間でデータ転送を行えます。dsPIC33F DMA コントローラは、決定論的機能とシステム レイテンシが重視される、高性能、リアルタイム、組み込みアプリケーション向けに最適化されています。

DMA コントローラは、周辺モジュール用データ レジスタとデータ領域 SRAM 間でデータを転送します。dsPIC33F DMA サブシステムはデュアルポート SRAM メモリ (DPSRAM) とレジスタ構造を採用し、CPU に負荷をかけずに独立したアドレスバスとデータバスで DMA を動作させる事が可能です。このアーキテクチャにより、優先度の高い DMA 転送が要求された際に CPU を停止させる事になるサイクル スチールが不要になります。CPU と DMA コントローラは共に、CPU ストール等の影響を受けずにデータ空間内のアドレスに対して読み書きできます。このため、リアルタイム性能を最大限に高められます。一方、DMA 動作と、メモリと周辺モジュール間でのデータ転送は CPU 処理の影響を受けません。例えば、実行時自己プログラミング (RTSP) 実行中は、RTSP が終了するまで CPU は一切の命令を実行しません。しかし、この状態でもメモリと周辺モジュール間のデータ転送に影響はありません。

図 22-1: DMA コントローラ



DMA コントローラは、8 つの独立したチャンネルをサポートしています。各チャンネルは、選択した周辺モジュールとの間でデータ転送するよう設定できます。DMA コントローラがサポートしている周辺モジュールは以下の通りです。

- ECAN™ テクノロジ
- データコンバータ インターフェイス (DCI)
- 10/12 ビット A/D コンバータ (ADC)
- SPI (シリアル ペリフェラル インターフェイス)
- UART
- 入力キャプチャ
- 出力コンペア

さらに、DMA 転送は外部割り込みに加えてタイマによって開始する事が可能です。各 DMA チャンネルは単方向です。1 つの周辺モジュールに対して読み書きを行うには、2 つの DMA チャンネルを割り当てる必要があります。データ転送の要求を受信するチャンネルが複数ある場合、どのチャンネルの転送を優先させ、どのチャンネルを待機状態にするかは、チャンネル番号に基づく簡単な固定優先度方式で決まります。各 DMA チャンネルは、最大 1024 のデータ要素からなるデータ ブロックを移動し、その後でこのブロックが処理準備完了している事を示すために CPU への割り込みを生成します。

DMA コントローラには以下の機能があります。

- 8 つの DMA チャンネル
- ポスト インクリメント アドレッシング モードを使用するレジスタ間接
- ポスト インクリメント アドレッシング モードを使用しないレジスタ間接
- 周辺モジュール間接アドレッシング モード(周辺モジュールがデスティネーション アドレスを生成)
- ハーフまたはフルブロック転送完了後の CPU 割り込み
- バイトまたはワード転送
- 固定優先度チャンネル調停
- 手動 (ソフトウェア) または自動 (周辺モジュール DMA 要求) 転送開始
- ワンショットまたは自動再送ブロック転送モード
- ピンポンモード (ブロック転送完了ごとに 2 つの DPSRAM 開始アドレスを自動切り換え)
- 各チャンネルに対する DMA 要求は割り込み要因から選択可能
- デバッグサポート機能

22.2 DMA レジスタ

各 DMA チャンネルは、6 つのステータス / 制御レジスタ セットを備えています。

- **DMAxCON: DMA チャンネル x 制御レジスタ**

  このレジスタでは、チャンネルを有効化 / 無効化し、データ転送サイズ、方向、ブロック割り込み方式を指定し、DMA チャンネル アドレッシング モード、動作モード、NULL データ書き込みモードを選択する事によって、対応する DMA チャンネルを設定します。

- **DMAxREQ: DMA チャンネル x IRQ 選択レジスタ**

  このレジスタでは、周辺モジュールの IRQ を DMA チャンネルに割り当てる事によって、DMA チャンネルと特定の DMA 対応周辺モジュールを関連付けます。

- **DMAxSTA: DMA チャンネル x DPSRAM 開始アドレス オフセット レジスタ A**

  このレジスタでは、DMA チャンネル x によって DPSRAM との間で送受信するデータブロックの、DMA DPSRAM ベースアドレスからのプライマリ開始アドレス オフセットを指定します。このレジスタを読み出すと、現在の DPSRAM 転送アドレス オフセット値が返されます。チャンネル x を有効 (アクティブな状態) にしたままこのレジスタに書き込むと、予期せぬ挙動をする可能性があるので避けてください。

- **DMAxSTB: DMA チャンネル x DPSRAM 開始アドレス オフセット レジスタ B**

  DMA チャンネル x によって DPSRAM との間で送受信するデータブロックの、DMA DPSRAM ベースアドレスからのセカンダリ開始アドレス オフセットを指定します。このレジスタを読み出すと、最新の DPSRAM 転送アドレス オフセット値が返されます。チャンネル x を有効 (アクティブな状態) にしたままこのレジスタに書き込むと、予期せぬ挙動をする可能性があるので避けてください。

- **DMAxPAD: DMA チャンネル x 周辺モジュール アドレス レジスタ**

  この読み書きレジスタには、周辺モジュール用データ レジスタの静的アドレスを格納します。対応する DMA チャンネルを有効 (アクティブな状態) にしたままこのレジスタに書き込むと、予期せぬ結果が生じる可能性があるので避けてください。

- **DMAxCNT: DMA チャンネル x 転送カウント レジスタ**

  このレジスタには転送カウントを格納します。DMAxCNT + 1 は、データブロック転送が完了したと見なす前にチャンネルで処理する必要がある DMA 要求の数を表します。つまり、DMAxCNT の値が「0」の場合、1 つの要素が転送されます。DMAxCNT レジスタの値は、転送データサイズ (DMAxCON レジスタの SIZE ビット) とは無関係です。対応する DMA チャンネルを有効 (アクティブな状態) にしたままこのレジスタに書き込むと、予期せぬ挙動をする可能性があるので避けてください。

個々の DMA チャンネル レジスタに加えて、DMA コントローラは 3 つの DMA ステータス レジスタを備えています。

- **DSADR: 最新の DMA DPSRAM アドレス レジスタ**

  この 16 ビット、読み出し専用のステータス レジスタは、全ての DMA チャンネルに共通です。最新の DPSRAM アクセス (読み出しまたは書き込み) のアドレスをキャプチャします。リセット時にクリアされ、DMA アクティビティの前に読み出すと「0x0000」という値を格納しています。このレジスタにはいつでもアクセス可能ですが、デバッグの支援が本来の目的です。

- **DMACS0: DMA コントローラ ステータス レジスタ 0**

  この 16 ビット、読み出し専用のステータス レジスタは、DPSRAM 書き込みコリジョンフラグ (XWCOLx) と周辺モジュール書き込みコリジョンフラグ (PWCOLx) を格納します。詳細は、**22.10「データ書き込みコリジョン」**を参照してください。

- **DMACS1: DMA コントローラ ステータス レジスタ 1**

  この 16 ビット、読み出し専用のステータス レジスタは、直近にアクティブであった DMA チャンネルを示し、どちらの DPSRAM 開始アドレス オフセット レジスタ (DMAxSTA または DMAxSTB) が選択されているか示す事によって各 DMA チャンネルのピンポンモード ステータスを示します。

レジスタ 22-1: DMAxCON: DMA チャンネル x 制御レジスタ

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	U-0	U-0	U-0
CHEN	SIZE	DIR	HALF	NULLW	—	—	—
bit 15							bit 8

U-0	U-0	R/W-0	R/W-0	U-0	U-0	R/W-0	R/W-0
—	—	AMODE<1:0>		—	—	MODE<1:0>	
bit 7							bit 0

凡例 :			
R = 読み出し可能ビット	W = 書き込み可能ビット	U = 未実装ビット、「0」として読み出し	
-n = POR 時の値	「1」= ビットをセット	「0」= ビットをクリア	x = ビットは未知

bit 15 **CHEN:** チャンネル イネーブルビット
1 = チャンネル有効
0 = チャンネル無効

bit 14 **SIZE:** データ転送サイズ ビット
1 = バイト
0 = ワード

bit 13 **DIR**: 転送方向ビット (ソース / デスティネーション バス選択)
1 = DPSRAM アドレスから読み出して、周辺モジュール アドレスに書き込む
0 = 周辺モジュール アドレスから読み出して、DPSRAM アドレスに書き込む

bit 12 **HALF:** ブロック転送割り込み選択ビット
1 = データの半分が移動した時点で割り込みを発生させる
0 = 全データが移動した時点で割り込みを発生させる

bit 11 **NULLW:** NULL データ周辺モジュール書き込みモード選択ビット
1 = DPSRAM 書き込みに加えて NULL データを周辺モジュールに書き込む (DIR ビットもクリアする必要がある)
0 = 通常動作

bit 10-6 **未実装 :** 「0」として読み出し

bit 5-4 **AMODE<1:0>:** DMA チャンネル アドレッシング モード選択ビット
11 = 予約
10 = 周辺モジュール間接アドレッシング モード
01 = ポスト インクリメントを使用しないレジスタ間接モード
00 = ポスト インクリメントを使用するレジスタ間接モード

bit 3-2 **未実装 :** 「0」として読み出し

bit 1-0 **MODE<1:0>:** DMA チャンネル動作モード選択ビット
11 = ワンショット モード、ピンポンモード有効 (各 DMA RAM バッファとの間の 1 ブロック転送)
10 = 連続モード、ピンポンモード有効
01 = ワンショット モード、ピンポンモード無効
00 = 連続モード、ピンポンモード無効

レジスタ 22-2: DMAxREQ: DMA チャンネル x IRQ 選択レジスタ

R/S-0	U-0	U-0	U-0	U-0	U-0	U-0	U-0
FORCE(1)	—	—	—	—	—	—	—
bit 15							bit 8

U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	IRQSEL<6:0>						
bit 7							bit 0

凡例 :			
R = 読み出し可能ビット	W = 書き込み可能ビット	U = 未実装ビット、「0」として読み出し	
-n = POR 時の値	「1」= ビットをセット	「0」= ビットをクリア	x = ビットは未知

bit 15 **FORCE:** 強制 DMA 転送ビット (1)
1 = 1 つの DMA 転送を強制的に実行する (手動モード)
0 = DMA 要求によって自動的に DMA 転送を開始する

bit 14-7 **未実装 :**「0」として読み出し

bit 6-0 **IRQSEL<6:0>:** DMA 周辺モジュール IRQ 番号選択ビット
0000000 = INT0 – 外部割り込み 0
0000001 = IC1 – 入力キャプチャ 1
0000010 = OC1 – 出力コンペア 1
0000101 = IC2 – 入力キャプチャ 2
0000110 = OC2 – 出力コンペア 2
0000111 = TMR2 – Timer2
0001000 = TMR3 – Timer3
0001010 = SPI1 – 転送完了
0001011 = UART1RX – UART1 レシーバ
0001100 = UART1TX – UART1 トランスミッタ
0001101 = ADC1 – ADC1 変換完了
0010101 = ADC2 – ADC2 変換完了
0011110 = UART2RX – UART2 レシーバ
0011111 = UART2TX – UART2 トランスミッタ
0100001 = SPI2 転送完了
0100010 = ECAN1 – RX データレディ
0110111 = ECAN2 – RX データレディ
0111100 = DCI – コーデック転送完了
1000110 = ECAN1 – TX データ要求
1000111 = ECAN2 – TX データ要求

Note 1: FORCE ビットはユーザからクリアできません。FORCE ビットは、強制 DMA 転送の完了時にハードウェアでクリアされます。

レジスタ 22-3: DMAxSTA: DMA チャンネル x DPSRAM 開始アドレス オフセット レジスタ A

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
STA<15:8>							
bit 15							bit 8

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
STA<7:0>							
bit 7							bit 0

凡例 :			
R = 読み出し可能ビット	W = 書き込み可能ビット	U = 未実装ビット、「0」として読み出し	
-n = POR 時の値	「1」= ビットをセット	「0」= ビットをクリア	x = ビットは未知

bit 15-0 **STA<15:0>:** プライマリ DPSRAM 開始アドレス オフセット ビット (ソースまたはデスティネーション)

レジスタ 22-4: DMAxSTB: DMA チャンネル x DPSRAM 開始アドレス オフセット レジスタ B

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
STB<15:8>							
bit 15							bit 8

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
STB<7:0>							
bit 7							bit 0

凡例 :			
R = 読み出し可能ビット	W = 書き込み可能ビット	U = 未実装ビット、「0」として読み出し	
-n = POR 時の値	「1」= ビットをセット	「0」= ビットをクリア	x = ビットは未知

bit 15-0 **STB<15:0>:** セカンダリ DPSRAM 開始アドレス オフセット ビット (ソースまたはデスティネーション)

レジスタ 22-5: DMAxPAD: DMA チャンネル x 周辺モジュール アドレス レジスタ

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
PAD<15:8>							
bit 15							bit 8

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
PAD<7:0>							
bit 7							bit 0

凡例 :			
R = 読み出し可能ビット	W = 書き込み可能ビット	U = 未実装ビット、「0」として読み出し	
-n = POR 時の値	「1」= ビットをセット	「0」= ビットをクリア	x = ビットは未知

bit 15-0 **PAD<15:0>:** 周辺モジュール アドレス レジスタ ビット

レジスタ 22-6: DMAxCNT: DMA チャンネル x 転送カウント レジスタ

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	—	—	—	—	CNT<9:8>	
bit 15							bit 8

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
CNT<7:0>							
bit 7							bit 0

凡例 :			
R = 読み出し可能ビット	W = 書き込み可能ビット	U = 未実装ビット、「0」として読み出し	
-n = POR 時の値	「1」= ビットをセット	「0」= ビットをクリア	x = ビットは未知

bit 15-10 **予約**

bit 9-0 **CNT<9:0>:** DMA 転送カウント レジスタ ビット

レジスタ 22-7: DSADR: 最新の DMA DPSRAM アドレス レジスタ

R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0
DSADR<15:8>							
bit 15							bit 8

R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0
DSADR<7:0>							
bit 7							bit 0

凡例 :			
R = 読み出し可能ビット	W = 書き込み可能ビット	U = 未実装ビット、「0」として読み出し	
-n = POR 時の値	「1」= ビットをセット	「0」= ビットをクリア	x = ビットは未知

bit 15-0 **DSADR<15:0>:** DMA によってアクセスされた最新の DMA DPSRAM アドレス ビット

レジスタ 22-8: DMACS0: DMA コントローラ ステータス レジスタ 0

R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0
PWCOL7	PWCOL6	PWCOL5	PWCOL4	PWCOL3	PWCOL2	PWCOL1	PWCOL0
bit 15							bit 8

R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0
XWCOL7	XWCOL6	XWCOL5	XWCOL4	XWCOL3	XWCOL2	XWCOL1	XWCOL0
bit 7							bit 0

凡例 :			
R = 読み出し可能ビット	W = 書き込み可能ビット	U = 未実装ビット、「0」として読み出し	
-n = POR 時の値	「1」= ビットをセット	「0」= ビットをクリア	x = ビットは未知

bit 15 **PWCOL7:** チャンネル 7 周辺モジュール書き込みコリジョンフラグ ビット
1 = 書き込みコリジョンを検出した
0 = 書き込みコリジョンを検出していない

bit 14 **PWCOL6:** チャンネル 6 周辺モジュール書き込みコリジョンフラグ ビット
1 = 書き込みコリジョンを検出した
0 = 書き込みコリジョンを検出していない

bit 13 **PWCOL5:** チャンネル 5 周辺モジュール書き込みコリジョンフラグ ビット
1 = 書き込みコリジョンを検出した
0 = 書き込みコリジョンを検出していない

bit 12 **PWCOL4:** チャンネル 4 周辺モジュール書き込みコリジョンフラグ ビット
1 = 書き込みコリジョンを検出した
0 = 書き込みコリジョンを検出していない

bit 11 **PWCOL3:** チャンネル 3 周辺モジュール書き込みコリジョンフラグ ビット
1 = 書き込みコリジョンを検出した
0 = 書き込みコリジョンを検出していない

bit 10 **PWCOL2:** チャンネル 2 周辺モジュール書き込みコリジョンフラグ ビット
1 = 書き込みコリジョンを検出した
0 = 書き込みコリジョンを検出していない

bit 9 **PWCOL1:** チャンネル 1 周辺モジュール書き込みコリジョンフラグ ビット
1 = 書き込みコリジョンを検出した
0 = 書き込みコリジョンを検出していない

bit 8 **PWCOL0:** チャンネル 0 周辺モジュール書き込みコリジョンフラグ ビット
1 = 書き込みコリジョンを検出した
0 = 書き込みコリジョンを検出していない

bit 7 **XWCOL7:** チャンネル 7 DPSRAM 書き込みコリジョンフラグ ビット
1 = 書き込みコリジョンを検出した
0 = 書き込みコリジョンを検出していない

bit 6 **XWCOL6:** チャンネル 6 DPSRAM 書き込みコリジョンフラグ ビット
1 = 書き込みコリジョンを検出した
0 = 書き込みコリジョンを検出していない

bit 5 **XWCOL5:** チャンネル 5 DPSRAM 書き込みコリジョンフラグ ビット
1 = 書き込みコリジョンを検出した
0 = 書き込みコリジョンを検出していない

bit 4 **XWCOL4:** チャンネル 4 DPSRAM 書き込みコリジョンフラグ ビット
1 = 書き込みコリジョンを検出した
0 = 書き込みコリジョンを検出していない

bit 3 **XWCOL3:** チャンネル 3 DPSRAM 書き込みコリジョンフラグ ビット
1 = 書き込みコリジョンを検出した
0 = 書き込みコリジョンを検出していない

レジスタ 22-8: DMACS0: DMA コントローラ ステータス レジスタ 0 (続き)

bit 2 **XWCOL2:** チャンネル 2 DPSRAM 書き込みコリジョンフラグ ビット
1 = 書き込みコリジョンを検出した
0 = 書き込みコリジョンを検出していない

bit 1 **XWCOL1:** チャンネル 1 DPSRAM 書き込みコリジョンフラグ ビット
1 = 書き込みコリジョンを検出した
0 = 書き込みコリジョンを検出していない

bit 0 **XWCOL0:** チャンネル 0 DPSRAM 書き込みコリジョンフラグ ビット
1 = 書き込みコリジョンを検出した
0 = 書き込みコリジョンを検出していない

レジスタ 22-9: DMACS1: DMA コントローラ ステータス レジスタ 1

U-0	U-0	U-0	U-0	R-1	R-1	R-1	R-1
—	—	—	—	LSTCH<3:0>			
bit 15							bit 8

R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0
PPST7	PPST6	PPST5	PPST4	PPST3	PPST2	PPST1	PPST0
bit 7							bit 0

凡例 :			
R = 読み出し可能ビット	W = 書き込み可能ビット	U = 未実装ビット、「0」として読み出し	
-n = POR 時の値	「1」= ビットをセット	「0」= ビットをクリア	x = ビットは未知

bit 15-12 **未実装 :** 「0」として読み出し

bit 11-8 **LSTCH<3:0>:** 直近の DMAC チャンネル アクティブ ビット
1111 = システム リセット以降、DMA 転送は発生していない
1110-1000 = 予約
0111 = 直近のデータ転送はチャンネル 7 で実行された
0110 = 直近のデータ転送はチャンネル 6 で実行された
0101 = 直近のデータ転送はチャンネル 5 で実行された
0100 = 直近のデータ転送はチャンネル 4 で実行された
0011 = 直近のデータ転送はチャンネル 3 で実行された
0010 = 直近のデータ転送はチャンネル 2 で実行された
0001 = 直近のデータ転送はチャンネル 1 で実行された
0000 = 直近のデータ転送はチャンネル 0 で実行された
リセット時には「1111」に設定されます。このフィールドにはいつでもアクセス可能ですが、デバッグの支援が本来の目的です。

bit 7 **PPST7:** チャンネル 7「ピンポン」モード ステータスフラグ
1 = DMA7STB レジスタが選択されている
0 = DMA7STA レジスタが選択されている

bit 6 **PPST6:** チャンネル 6「ピンポン」モード ステータスフラグ
1 = DMA6STB レジスタが選択されている
0 = DMA6STA レジスタが選択されている

bit 5 **PPST5:** チャンネル 5「ピンポン」モード ステータスフラグ
1 = DMA5STB レジスタが選択されている
0 = DMA5STA レジスタが選択されている

bit 4 **PPST4:** チャンネル 4「ピンポン」モード ステータスフラグ
1 = DMA4STB レジスタが選択されている
0 = DMA4STA レジスタが選択されている

bit 3 **PPST3:** チャンネル 3「ピンポン」モード ステータスフラグ
1 = DMA3STB レジスタが選択されている
0 = DMA3STA レジスタが選択されている

bit 2 **PPST2:** チャンネル 2「ピンポン」モード ステータスフラグ
1 = DMA2STB レジスタが選択されている
0 = DMA2STA レジスタが選択されている

bit 1 **PPST1:** チャンネル 1「ピンポン」モード ステータスフラグ
1 = DMA1STB レジスタが選択されている
0 = DMA1STA レジスタが選択されている

bit 0 **PPST0:** チャンネル 0「ピンポン」モード ステータスフラグ
1 = DMA0STB レジスタが選択されている
0 = DMA0STA レジスタが選択されている

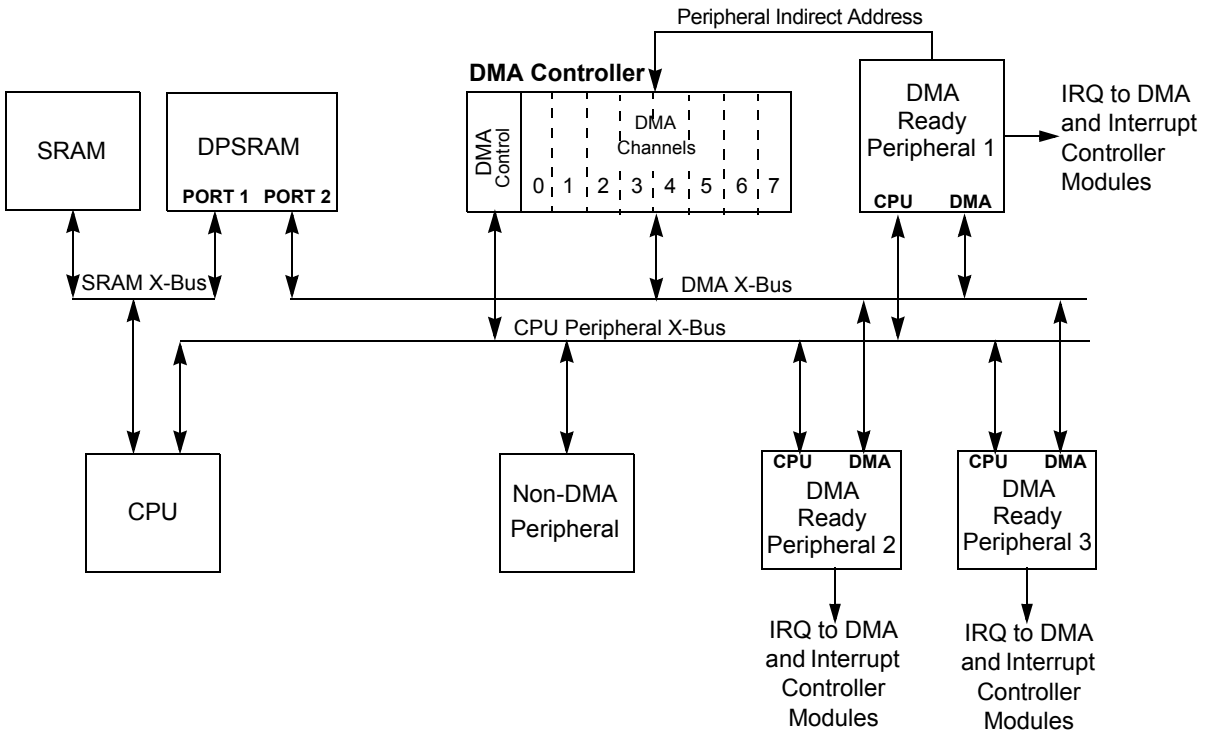
Note: このレジスタは読み出し専用です。

22.3 DMA のブロック図

図 22-2 は、dsPIC33F の内部アーキテクチャに DMA がどのように組み込まれているのかを示すブロック図です。CPU は従来の SRAM と X バスを介して通信します。また、同じ X バスを介してデュアルポート SRAM (DPSRAM) ブロックのポート 1 とも通信します。CPU は、別の周辺 X バスを介して周辺モジュールと通信します。この X バスも X データ空間内にあります。

各 DMA チャンネルは、DMA 専用バスを介して、DPSRAM のポート 2 と各 DMA 対応周辺モジュールの DMA ポートと通信します。

図 22-2: DMA コントローラのブロック図



Note: わかりやすくするために、CPU および DMA のアドレスバスは表示していません。

他のアーキテクチャとは異なり、dsPIC33F CPU は CPU バスサイクルごとに読み書きアクセスが可能です。同様に、DMA は専用バスを介して、バスサイクルごとに 1 バイトまたは 1 ワードの転送を完了させます。これにより、全ての DMA 転送で割り込みが発生しません。つまり、他のチャンネル アクティビティとは関係なく、転送を開始すると同一サイクル内で転送を完了させます。

ユーザ アプリケーションは任意の DMA 対応周辺モジュール割り込みを DMA 要求として指定できます。これは DMA に向けられた IRQ です。もちろん、特定の割り込みを DMA 要求として応答するように DMA チャンネルを設定している場合、対応する CPU 割り込みは無効化しておきます。そうでないと CPU 割り込みも要求されてしまいます。

各 DMA チャンネルは、ソフトウェアから手動でトリガする事もできます。DMAxCON レジスタの FORCE ビットをセットすると、割り込みベースの全ての DMA 要求と同じ調停の対象となる手動 DMA 要求が開始されます (**22.8「DMA チャンネル アービトレーションとオーバーラン」**参照)。

22.4 DMA データ転送

図 22-3 に、周辺モジュールとデュアルポート SRAM 間のデータ転送を示します。

A. この例では、DMA チャンネル 5 は DMA 対応周辺モジュール 1 と共に動作するように設定されています。

B. 周辺モジュールからデータを転送する準備が整うと、周辺モジュールが DMA 要求を発行します。この DMA 要求と同時に発生した他の要求があれば、調停が行われます。このチャンネルが最も高い優先度を持つ場合、転送は次のサイクル中に完了します。そうでない場合、最も優先度が高くなるまで DMA 要求は保留されます。

C. DMA チャンネルは、指定された周辺モジュール アドレスからデータの読み出しを実行します。アドレスは、ユーザ アプリケーションがアクティブ チャンネル内で定義します。

D. DMA チャンネルは、指定された DPSRAM アドレスにデータを書き込みます。

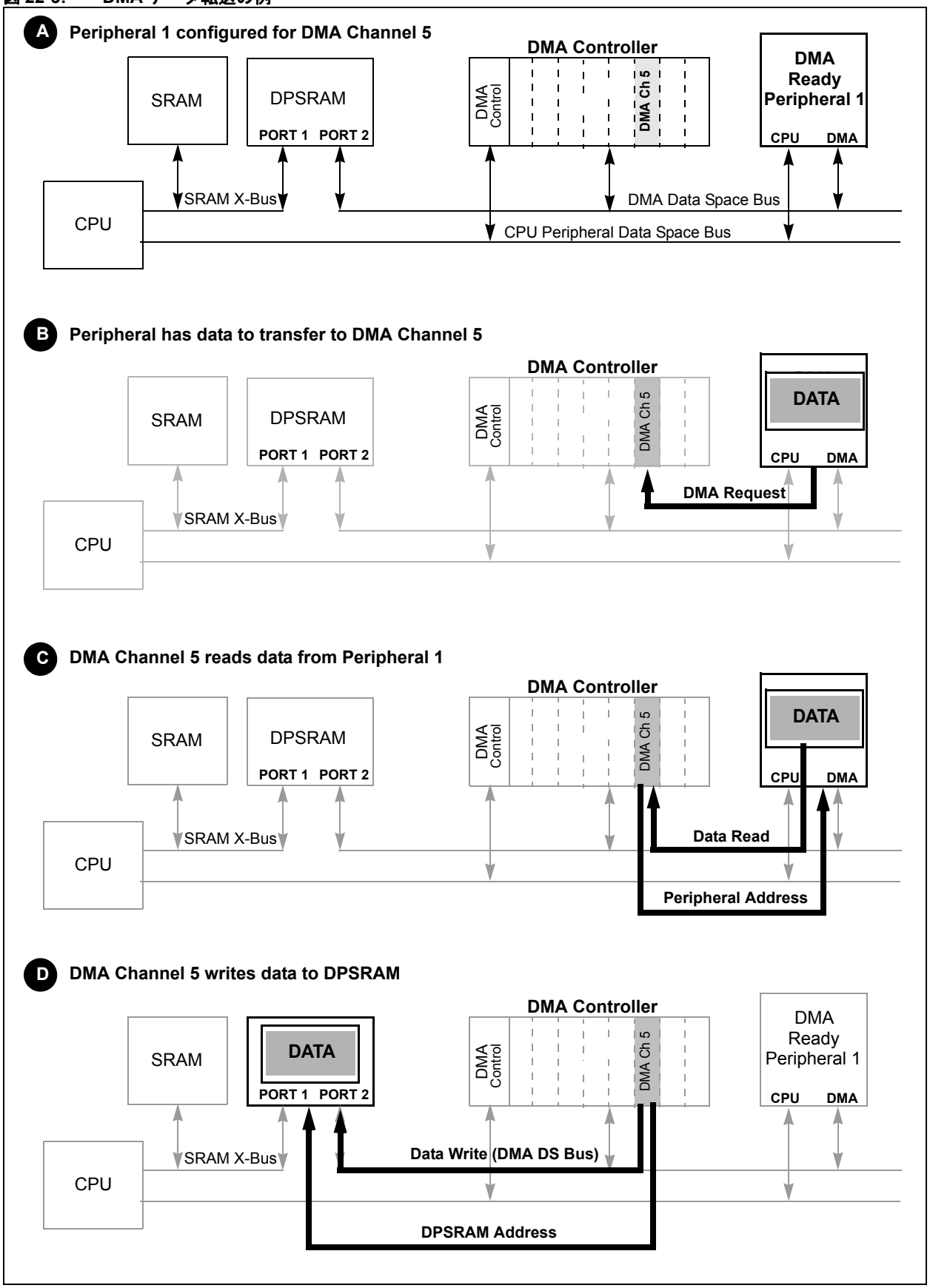
この例はレジスタ間接モードを表しています。DPSRAM アドレスは、DMA ステータス レジスタ (DMAxSTA または DMAxSTB) によって DMA チャンネル内で指定されます。周辺モジュール間接モードでは、DPSRAM アドレスはアクティブ チャンネルではなく周辺モジュールが指定します。詳細は、**22.6.6「周辺モジュール間接アドレッシング モード」**を参照してください。

DMA の読み書き転送動作全体は、1 つの命令サイクル内で中断される事なく実行されます。このプロセス全体を通じて、データ転送が完了するまで DMA 要求は DMA チャンネルでラッチされた状態を保ちます。

DMA チャンネルは併行して転送カウンタ レジスタ (DMA5CNT) を監視します。転送カウントがユーザ アプリケーションの指定する限界値に達すると、データ転送は完了と見なされ、CPU 割り込みが生成され、新たに受信したデータを処理するよう CPU に通知します。

データ転送サイクルの間、DMA コントローラは、保留中および後続の DMA 要求の調停も継続し、スループットを最大限にします。

図 22-3: DMA データ転送の例



22.5 DMA の設定

DMA データ転送を正常に機能させるには、DMA チャンネルと周辺モジュールを以下のように適切に設定する必要があります。

- DMA チャンネルを周辺モジュールと関連付ける (**22.5.1「DMA チャンネルと周辺モジュールの関連付けの設定」**参照)
- 周辺モジュールを適切に設定する (**22.5.2「周辺モジュールの設定」**参照)
- DPSRAM データ開始アドレスを初期化する (**22.5.3「メモリアドレスの初期化」**参照)
- DMA 転送カウントを初期化する (**22.5.4「DMA 転送カウントの設定」**参照)
- 適切なアドレッシング モードと動作モードを選択する (**22.6「DMA 動作モード」**参照)

22.5.1 DMA チャンネルと周辺モジュールの関連付けの設定

DMA チャンネルは、読み書きの対象となる周辺モジュール アドレスとそのタイミングを把握する必要があります。この情報は、DMA チャンネル x 周辺モジュール アドレス レジスタ (DMAxPAD) と DMA チャンネル x IRQ 選択レジスタ (DMAxREQ) で設定します。

表 22-1 に、周辺モジュールと DMA チャンネルを関連付けるためにレジスタに書き込む値を示します。

表 22-1: DMA チャンネルと周辺モジュールの関連付け

周辺モジュールと DMA の関連付け	DMAxREQ レジスタ IRQSEL<6:0> ビット	周辺モジュールから読み出す DMAxPAD レジスタの値	周辺モジュールに書き込む DMAxPAD レジスタの値
INT0 – 外部割り込み 0	0000000	—	—
IC1 – 入力キャプチャ 1	0000001	0x0140 (IC1BUF)	—
IC2 – 入力キャプチャ 2	0000101	0x0144 (IC2BUF)	—
OC1 – 出力コンペア 1 データ	0000010	—	0x0182 (OC1R)
OC1 – 出力コンペア 1 セカンダリデータ	0000010	—	0x0180 (OC1RS)
OC2 – 出力コンペア 2 データ	0000110	—	0x0188 (OC2R)
OC2 – 出力コンペア 2 セカンダリデータ	0000110	—	0x0186 (OC2RS)
TMR2 – Timer2	0000111	—	—
TMR3 – Timer3	0001000	—	—
SPI1 – 転送完了	0001010	0x0248 (SPI1BUF)	0x0248 (SPI1BUF)
SPI2 – 転送完了	0100001	0x0268 (SPI2BUF)	0x0268 (SPI2BUF)
UART1RX – UART1 レシーバ	0001011	0x0226 (U1RXREG)	—
UART1TX – UART1 トランスミッタ	0001100	—	0x0224 (U1TXREG)
UART2RX – UART2 レシーバ	0011110	0x0236 (U2RXREG)	—
UART2TX – UART2 トランスミッタ	0011111	—	0x0234 (U2TXREG)
ECAN1 – RX データレディ	0100010	0x0440 (C1RXD)	—
ECAN1 – TX データ要求	1000110	—	0x0442 (C1TXD)
ECAN2 – RX データレディ	0110111	0x0540 (C2RXD)	—
ECAN2 – TX データ要求	1000111	—	0x0542 (C2TXD)
DCI – コーデック転送完了	0111100	0x0290 (RXBUF0)	0x0298 (TXBUF0)
ADC1 – ADC1 変換完了	0001101	0x0300 (ADC1BUF0)	—
ADC2 – ADC2 変換完了	0010101	0x0340 (ADC2BUF0)	—

2 つの DMA チャンネルが DMA 要求のソースとして同じ周辺モジュールを選択した場合、両チャンネルが同時に DMA 要求を受信します。しかし、最も高い優先度を持つチャンネルが最初に転送を実行し、他のチャンネルは保留状態となります。1 回の DMA 要求で周辺モジュールに対してデータの送信と受信を行う場合 (SPI 等)、このような状況がよく発生します。DMA チャンネルは 2 つ使用されます。1 つは周辺モジュールの読み出し用、もう 1 つは周辺モジュールのデータ書き込み用に割り当てられます。どちらのチャンネルも同じ DMA 要求を使用します。

DMAxPAD レジスタが表 22-1 に記載されている値以外に初期化されている場合、この周辺モジュール アドレスに対する DMA チャンネルの書き込みは無視されます。このアドレスからの DMA チャンネルの読み出しは、「0」として読み出されます。

22.5.2 周辺モジュールの設定

DMA 設定プロセスの第 2 段階は、周辺モジュールを DMA 動作用に正しく設定する事です。表 22-2 に、DMA 対応周辺モジュールの設定要件の概要を示します。

表 22-2: DMA 対応周辺モジュール設定の注意事項

DMA 対応周辺モジュール	設定時の注意事項
ECAN	ECAN バッファは DMA RAM 内に割り当てられます。DMA RAM 内の CAN バッファ領域と FIFO の全体サイズはユーザが指定し、ECAN FIFO 制御レジスタ (C1FCTRL) 内の DMA バッファサイズ ビット DMABS<2:0> で定義する必要があります。例 22-9 に、そのサンプルコードを示します。
データコンバータ インターフェイス (DCI)	DCI は、データワードがバッファリングされるたびに割り込みを生成するように、DCI 制御 2 レジスタ (DCICON2) 内のバッファ長制御ビット (BLEN<1:0>) を「00」にセットします。RX と TX のデータ転送をサポートするには、2 つの DMA チャンネルに対する要求と同じ DCI 割り込みを使用する必要があります。DCI モジュールがマスタとして動作中でデータ受信のみ行っている場合、2 番目の DMA チャンネルを使用してダミーの送信データを送出する必要があります。例 22-11 に、そのサンプルコードを示します。
10/12 ビットの A/D コンバータ (ADC)	ADC を周辺モジュール間接モードとして DMA と使用する場合、ADCx 制御 2 レジスタ (ADCxCON2) 内の DMA アドレス インクリメント レート ビット (SMPI<3:0>) と、ADCx 制御 4 レジスタ (ADCxCON4) 内のアナログ入力ごとの DMA バッファ位置ビット (DMABL<2:0>) の数を正しく設定する必要があります。また、ADCx 制御 1 レジスタ (ADxCON1) 内の DMA バッファ構築モード ビット (ADDMABM) を ADC アドレス生成用に正しくセットします。詳細は、**22.6.6.1「DMA アドレス生成に対する ADC のサポート」**を参照してください。例 22-5 と例 22-7 に、そのサンプルコードを示します。
SPI (シリアル ペリフェラル インターフェイス)	SPI モジュールがマスタとして動作中でデータ受信のみ行っている場合、2 番目の DMA チャンネルを割り当て、ダミーの送信データを送出する必要があります。あるいは、1 つの DMA チャンネルを NULL データ書き込みモードで使用する事も可能です。詳細は、**22.6.11「NULL データ書き込みモード」**を参照してください。例 22-12 に、そのサンプルコードを示します。
UART	UART では、文字を送受信するたびに割り込みを生成するように設定する必要があります。UART レシーバが 1 キャラクタ受信するたびに RX 割り込みを生成するように、ステータス / 制御レジスタ (UxSTA) 内の受信割り込みモード選択ビット (URXISEL<1:0>) を「00」または「01」にセットする必要があります。 UART トランスミッタが 1 キャラクタ送信するたびに TX 割り込みを生成するように、ステータス / 制御レジスタ (UxSTA) 内の送信割り込みモード選択ビット (UTXISEL0 と UTXISEL1) を「0」にセットする必要があります。例 22-10 に、そのサンプルコードを示します。
入力キャプチャ	入力キャプチャ制御レジスタ (ICxCON) 内の割り込み発生キャプチャ回数選択ビット (ICI<1:0>) を「00'」にセットして、キャプチャ イベントごとに割り込みを生成するよう入力キャプチャ モジュールを設定する必要があります。例 22-4 に、そのサンプルコードを示します。
出力コンペア	出力コンペア モジュールでは、DMA を使用するにあたり特殊な設定は不要です。しかし、通常タイマを使用して DMA 要求を供給するため、タイマを正しく設定する必要があります。例 22-3 に、そのサンプルコードを示します。

表 22-2: DMA 対応周辺モジュール設定の注意事項 (続き)

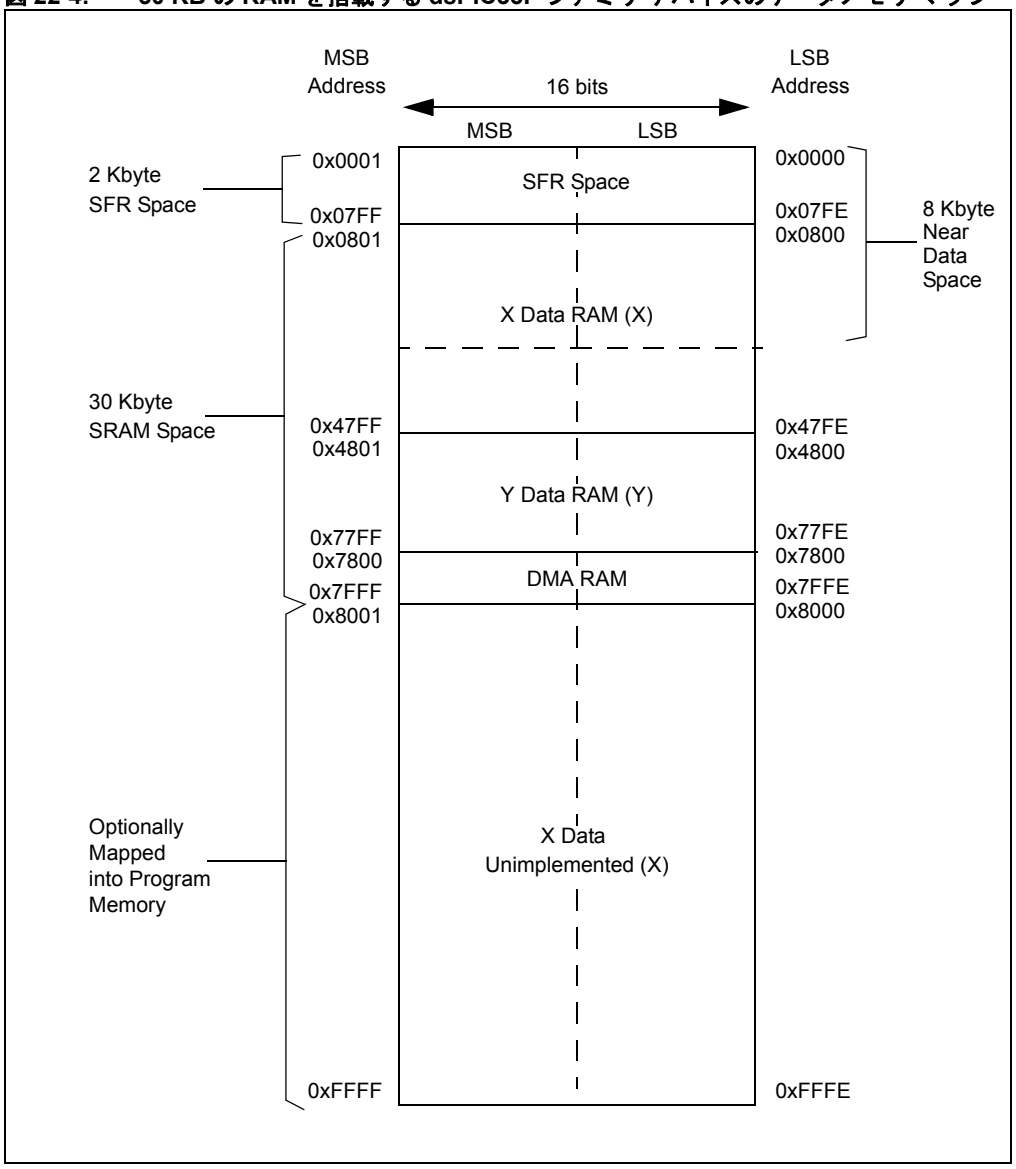
DMA 対応周辺モジュール	設定時の注意事項
外部割り込みとタイマ	DMA 要求として選択できるのは、外部割り込み 0、Timer2、Timer3 のみです。これらの周辺モジュール自体は DMA 転送をサポートしていませんが、他の DMA 対応周辺モジュールの DMA 転送をトリガするために使用できます。例えば、Timer2 は PWM モードの出力コンペア周辺モジュールに対する DMA トランザクションをトリガできます。例 22-3 に、そのサンプルコードを示します。

通常、DMA 対応周辺モジュール内でエラー条件が発生すると、ステータスフラグがセットされ、割り込みが生成されます (ユーザ アプリケーションで割り込みが有効な場合)。周辺モジュールが CPU によって使用されている場合、データ割り込みハンドラはエラーフラグをチェックし、必要に応じて適切な措置をとる事が要求されます。しかし、周辺モジュールが DMA チャンネルによって使用されている場合、DMA はデータ転送要求のみに応答し、後続のエラー条件は認識しません。DMA 対応周辺モジュール内で発生するエラー条件は全て、関連する割り込みが有効化されている必要があり、このような割り込みが周辺モジュール内に存在する場合にユーザ定義割り込みサービスルーチン (ISR) によって使用される必要があります。

22.5.3 メモリアドレスの初期化

第 3 の DMA 設定要件は、DMA アクセス用に特定のメモリ領域にメモリバッファを割り当てる事です。このメモリ領域の位置とサイズは、dsPIC33F デバイスによって異なります (デバイスのデータシート参照)。図 22-4 に、30 KB の RAM を搭載する dsPIC33F における 2 KB の DMA メモリ領域を示します。

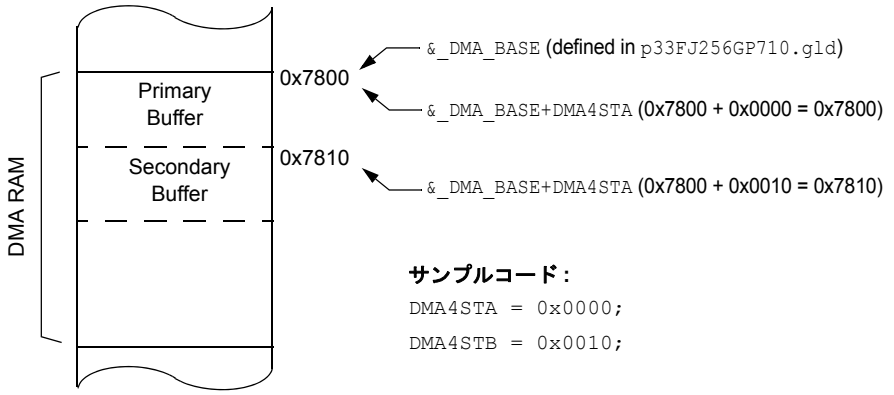
図 22-4: 30 KB の RAM を搭載する dsPIC33F ファミリ デバイスのデータメモリ マップ



適切に動作させるため、読み書きする DPSRAM アドレスを DMA メモリの開始位置からのオフセットとして指定する必要があります。この情報は、DMA チャンネル x DPSRAM 開始アドレス オフセット A (DMAxSTA) レジスタと DMA チャンネル x DPSRAM 開始アドレス開始アドレス オフセット B (DMAxSTB) レジスタで設定します。

図 22-5 に、dsPIC33FJ256GP710 上でプライマリ / セカンダリ DMA チャンネル 4 バッファをそれぞれ 0x7800 と 0x7810 のアドレスに設定する例を示します。

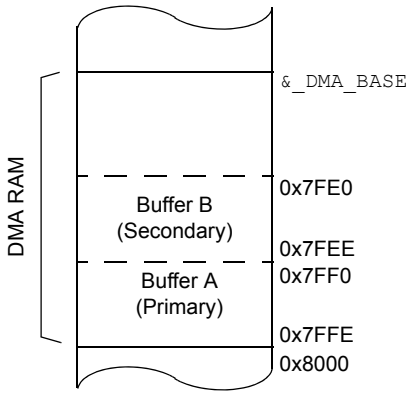
図 22-5: DMA メモリにおけるプライマリ / セカンダリバッファの割り当て



この例では、この情報をアプリケーション内にハードコードするにはデバイスのメモリ レイアウトに精通している必要があります。また、DMA 転送の完了後これらのバッファにアクセスするにはポインタ演算を利用する必要があります。そのため、この実装は別のプロセッサに移植するのが困難です。

MPLAB® C30 コンパイラはこの問題に対処するため、ビルトイン C 言語プリミティブで DMA バッファの初期化とアクセスを簡単にしています。例えば、図 22-6 のコードは DMA メモリ内にバッファを 2 つ割り当て、2 つのバッファを指すように DMA チャンネルを初期化します。

図 22-6: MPLAB® IDE を使用するプライマリ / セカンダリ DMA バッファの割り当て



サンプルコード :

```
unsigned int BufferA[8] __attribute__((space(dma)));
unsigned int BufferB[8] __attribute__((space(dma)));

DMA0STA = __builtin_dmaoffset(BufferA);
DMA0STB = __builtin_dmaoffset(BufferB);
```

Note: MPLAB LINK30 リンカは、DMA メモリ領域の一番下から逆の順番でプライマリとセカンダリバッファを割り当てます。

DMAxSTA (または DMAxSTB) レジスタ初期化により DMA チャンネルが DMA RAM 領域外の RAM アドレスを読み書きする結果となる場合、このメモリアドレスへの DMA チャンネルの書き込みは無視されます。このメモリアドレスからの DMA チャンネルの読み出しは、「0」として読み出されます。

22.5.4 DMA 転送カウントの設定

DMA 設定プロセスの第 4 段階では、データブロック転送が完了したと見なす前に、N + 1 の数の要求に対応するよう DMA チャンネルをプログラムする必要があります。「N」の値は DMA チャンネル x 転送カウント (DMAxCNT) レジスタを使って指定します。つまり、DMAxCNT の値「0」で 1 つの要素が転送されます。

DMAxCNT レジスタの値は転送データ サイズ (バイトまたはワード) とは無関係です。後者は DMAxCON レジスタの SIZE ビットで指定します。

DMAxCNT レジスタ初期化により DMA チャンネルが DMA RAM 領域外の RAM アドレスを読み書きする結果となる場合、このメモリアドレスへの DMA チャンネルの書き込みは無視されます。このメモリアドレスからの DMA チャンネルの読み出しは、「0」として読み出されます。

22.5.5 動作モードの設定

DMA 設定プロセスの最後となる第 5 段階では、DMA チャンネル x 制御 (DMAxCON) レジスタを設定して各 DMA チャンネルのモードを指定します。設定の詳細は、**22.6「DMA 動作モード」**を参照してください。

22.6 DMA 動作モード

DMA チャンネルは以下の動作モードをサポートします。

- ワードまたはバイトデータ転送
- 転送方向 (周辺モジュール→ DPSRAM または DPSRAM →周辺モジュール)
- CPU へのフルまたはハーフ転送割り込み
- ポスト インクリメントまたは静的 DPSRAM アドレッシング
- 周辺モジュール間接アドレッシング
- ワンショットまたは連続ブロック転送
- 各転送の完了後の、2 つの開始アドレス オフセット (DMAxSTA または DMAxSTB) 間での自動切り換え (ピンポンモード)
- NULL データ書き込みモード

さらに、DMA は手動モードをサポートしており、1 つの DMA 転送を強制的に実行します。

22.6.1 ワードまたはバイトデータ転送

ワードまたはバイトごとにデータを転送するよう各 DMA チャンネルを設定できます。ワードデータはアライメントされた (偶数) アドレス間でのみ移動できます。一方、バイトデータは任意の (正規) アドレス間で移動できます。

SIZE ビット (DMAxCON<14>) がクリアされている場合、ワードサイズのデータが転送されます。ポスト インクリメント アドレッシングを使用するレジスタ間接モードが有効な場合、ワード転送ごとにアドレスが 2 ずつポストインクリメントされます (**22.6.5「ポスト インクリメント アドレッシングを使用しないレジスタ間接モード」**参照)。

SIZE ビットがセットされている場合、バイトサイズのデータが転送されます。ポスト インクリメント アドレッシングを使用するレジスタ間接モードが有効な場合、バイト転送ごとにアドレスが 1 ずつインクリメントされます。

22.6.2 転送方向

各 DMA チャンネルは、周辺モジュールから DPSRAM へ、または DPSRAM から周辺モジュールへデータを転送するよう設定できます。

DMAxCON 内の転送方向 (DIR) ビットがクリアされている場合、データは周辺モジュールから読み出され (DMAxPAD が提供する周辺モジュール アドレスを使用)、DPSRAM DMA メモリアドレス オフセットに書き込まれます (DMAxSTA または DMAxSTB を使用)。

DIR ビットがセットされている場合、データは DPSRAM DMA メモリアドレス オフセットから読み出され (DMAxSTA または DMAxSTB を使用)、周辺モジュールに書き込まれます (DMAxPAD が提供する周辺モジュール アドレスを使用)。

一度設定すると、各チャンネルは単方向のデータコンジットになります。つまり、周辺モジュールが DMA モジュールを使用してデータを読み書きする必要がある場合、読み出し用と書き込み用の 2 つのチャンネルを割り当てる必要があります。

22.6.3 フルブロックまたはハーフブロック転送割り込み

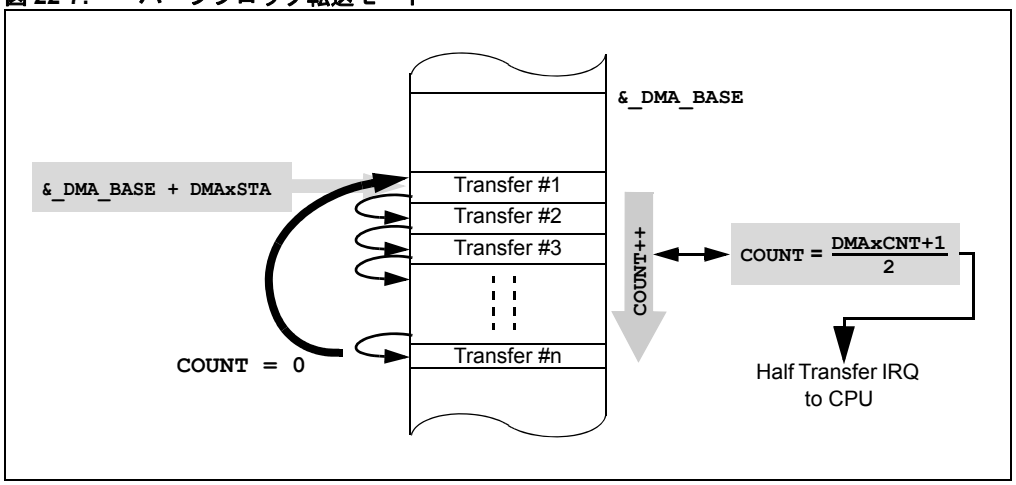
ブロック データ転送が完了または半分完了した時点で、各 DMA チャンネルは割り込みコントローラに割り込みを生成します。このモードを指定するには、DMA チャンネル x 制御 (DMAxCON) レジスタの HALF ビットをクリアまたはセットします。

HALF = 0 (データが全て移動した時点で割り込みを生成)

HALF = 1 (データの半分が移動した時点で割り込みを生成)

DMA 連続モードを使用する場合、CPU は DMA がデータを移動するのと同等以上の速度で入出力データを処理する必要があります。ハーフ転送割り込みは、データの半分のみが転送された時点で割り込みを生成する事によってこの問題を軽減します。例えば、ADC が DMA コントローラで連続的に読み出されている場合、ハーフ転送割り込みを使用すれば CPU はバッファが完全にフルになる前に処理できます。DMA 書き込みよりも先に進むことはないという条件で、この方式を利用すると CPU の応答時間に対する要求を緩和できます。図 22-7 にこのプロセスを示します。

図 22-7:　ハーフブロック転送モード



全てのモードにおいて、HALF ビットをセットすると、DMA はバッファ A または B の前半が転送された時点でのみ割り込みを生成します。バッファ A または B が完全に転送された時点では割り込みは生成されません。つまり、割り込みは DMA が (DMAxCNT + 1)/2 転送を完了した時点でのみ生成されます。(DMAxCNT + 1) が奇数の場合、割り込みは (DMAxCNT + 2)/2 転送の完了後に生成されます。

例えば、DMA3 をワンショット、ピンポンバッファ (MODE<1:0> = 11)、DMA3CNT = 7 に設定する場合、2 つの DMA3 割り込みが生成されます。1 つはバッファ A から 4 つの要素を転送した後、もう 1 つはバッファ B から 4 つの要素を転送した後に生成されます (詳細は、**22.6.7「ワンショットモード」**と **22.6.9「ピンポンモード」**参照)。

DMA チャンネルは**ハーフブロックまたはフルブロック**転送で割り込みを生成するのですが、ユーザ アプリケーションを使って DMA チャンネルを「あざむいて」、DMA 割り込み中に HALF ビットの値を切り換えて、**ハーフブロックとフルブロック**転送の両方で割り込みを生成させる事ができます。例えば、DMA チャンネルの設定で HALF ビットを「1」にセットすると、割り込みはハーフブロック転送ごとに生成されます。割り込みの実行中にユーザ アプリケーションが HALF ビットを「0」にリセットすると、DMA はフルブロック転送完了時に再度割り込みを生成します。

これらの割り込みを有効にするには、表 22-3 で示すように、割り込みコントローラ モジュールの割り込みイネーブル制御 (IECx) レジスタで対応する DMA 割り込みイネーブルビット (DMAxIE) をセットする必要があります。

表 22-3:　DMA 割り込みを有効化 / 無効化するための割り込みコントローラの設定

DMA チャンネル	割り込みコントローラ レジスタ名とビット番号	対応するレジスタ ビット名	C 構造体アクセスコード
0	IEC0<4>	DMA0IE	IEC0bits.DMA0IE
1	IEC0<14>	DMA1IE	IEC1bits.DMA1IE
2	IEC1<8>	DMA2IE	IEC1bits.DMA2IE
3	IEC2<4>	DMA3IE	IEC2bits.DMA3IE
4	IEC2<14>	DMA4IE	IEC2bits.DMA4IE
5	IEC3<13>	DMA5IE	IEC3bits.DMA5IE
6	IEC4<4>	DMA6IE	IEC4bits.DMA6IE
7	IEC4<5>	DAM7IE	IEC4bits.DMA7IE

例 22-1 に、DMA チャンネル 0 割り込みを有効にする方法を示します。

例 22-1:　DMA チャンネル 0 割り込みを有効にするためのコード

```
IEC0bits.DMA0IE = 1;
```

各 DMA チャンネル転送割り込みは、割り込みコントローラ内の対応するステータスフラグをセットし、割り込みサービスルーチン (ISR) をトリガします。この時ユーザ アプリケーションは、転送完了 ISR が再実行されるのを防ぐために、そのステータスフラグをクリアする必要があります。

表 22-4 に、割り込みコントローラ モジュール内の割り込みフラグ ステータス (IFSx) レジスタと対応するビット名 (DMAxIF) を示します。また、フラグをクリアする C 構造体アクセス コードも併せて示します。

表 22-4: DMA 割り込みステータスフラグをクリアするための割り込みコントローラの設定

DMA チャンネル	割り込みコントローラ レジスタ名とビット番号	対応するレジスタ ビット名	C 構造体アクセス コード
0	IFS0<4>	DMA0IF	IFS0bits.DMA0IE
1	IFS0<14>	DMA1IF	IFS0bits.DMA1IE
2	IFS1<8>	DMA2IF	IFS1bits.DMA2IE
3	IFS2<4>	DMA3IF	IFS2bits.DMA3IE
4	IFS2<14>	DMA4IF	IFS2bits.DMA4IE
5	IFS3<13>	DMA5IF	IFS3bits.DMA5IE
6	IFS4<4>	DMA6IF	IFS4bits.DMA6IE
7	IFS4<5>	DMA7IF	IFS4bits.DMA7IE

例として、DMA チャンネル 0 割り込みが有効で、DMA チャンネル 0 転送が終了し、関連する割り込みが割り込みコントローラに対して生成されたと仮定します。ステータスフラグをクリアし、割り込みが保留になるのを防ぐために、以下のコードが DMA チャンネル 0 の ISR 内に必要です。

例 22-2: DMA チャンネル 0 割り込みをクリアするためのコード

```
void __attribute__((__interrupt__)) _DMA0Interrupt(void)
{
    . . .

IFS0bits.DMA0IF = 0;
}
```

22.6.4 ポスト インクリメント アドレッシングを使用するレジスタ間接モード

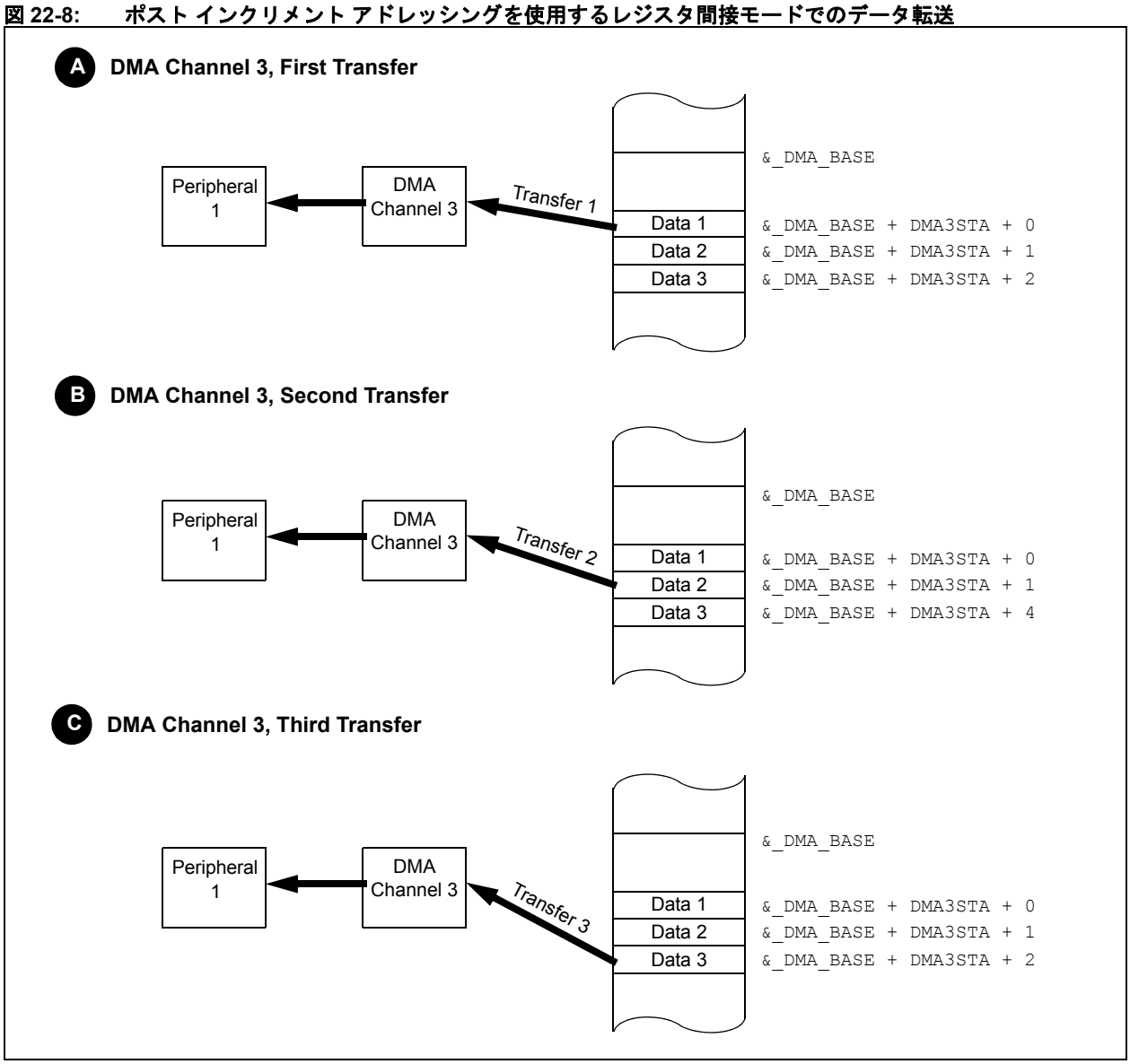
ポスト インクリメント アドレッシングを使用するレジスタ間接モードでは、転送ごとに DPSRAM アドレスをインクリメントしてデータブロックを移動します。

DMA コントローラのリセット後、DMA チャンネルの既定値はこのモードです。このモードを選択するには、DMA チャンネル制御 (DMAxCON) レジスタのアドレッシング モード選択ビット (AMODE<1:0>) を「00」にプログラムします。このモードでは、DPSRAM 開始アドレス オフセット (DMAxSTA または DMAxSTB) レジスタが DPSRAM バッファの開始アドレスを指定します。

ユーザ アプリケーションは DPSRAM 開始アドレス オフセット レジスタを読み出して、最新の DPSRAM 転送アドレス オフセットを判定します。しかし、このレジスタの内容は DMA コントローラから修正できません。

図 22-8 に、このモードでのデータ転送を示します。

図 22-8: ポスト インクリメント アドレッシングを使用するレジスタ間接モードでのデータ転送



例 22-3: 出力コンペアと、ポスト インクリメントを使用するレジスタ間接モードの DMA のコード

出力コンペア 1 モジュールを PWM モードに設定 :

```
OC1CON = 0;                         // Reset OC module
OC1R = 0x60;                        // Initialize PWM Duty Cycle
OC1RS = 0x60;                       // Initialize PWM Duty Cycle Buffer

OC1CONbits.OCM = 6;                 // Configure OC for the PWM mode
```

Timer2 を要求ソースに指定して DMA チャンネル 3 をポスト インクリメント モードに設定 :

```
unsigned int BufferA[32] __attribute__((space(dma)));
/* Insert code here to initialize BufferA with desired Duty Cycle values */

DMA3CONbits.AMODE = 0;              // Configure DMA for Register indirect mode
                                    // with post-increment
DMA3CONbits.MODE = 0;               // Configure DMA for Continuous mode
DMA3CONbits.DIR   = 1;              // RAM-to-Peripheral data transfers
DMA3PAD = (volatile unsigned int)&OC1RS; // Point DMA to OC1RS
DMA3CNT = 31;                       // 32 DMA request
DMA3REQ = 7;                        // Select Timer2 as DMA Request source

DMA3STA = __builtin_dmaoffset(BufferA);

IFS2bits.DMA3IF = 0;                // Clear the DMA interrupt flag bit
IEC2bits.DMA3IE = 1;                // Set the DMA interrupt enable bit

DMA3CONbits.CHEN = 1;               // Enable DMA
```

Timer2 を出力コンペア PWM モードに設定 :

```
PR2 = 0xBF;                         // Initialize PWM period
T2CONbits.TON = 1;                  // Start Timer2
```

DMA チャンネル 3 割り込みハンドラの設定 :

```
void __attribute__((__interrupt__)) _DMA3Interrupt(void)
{
    /* Update BufferA with new Duty Cycle values if desired here*/

    IFS2bits.DMA3IF = 0;            //Clear the DMA3 Interrupt Flag
}
```

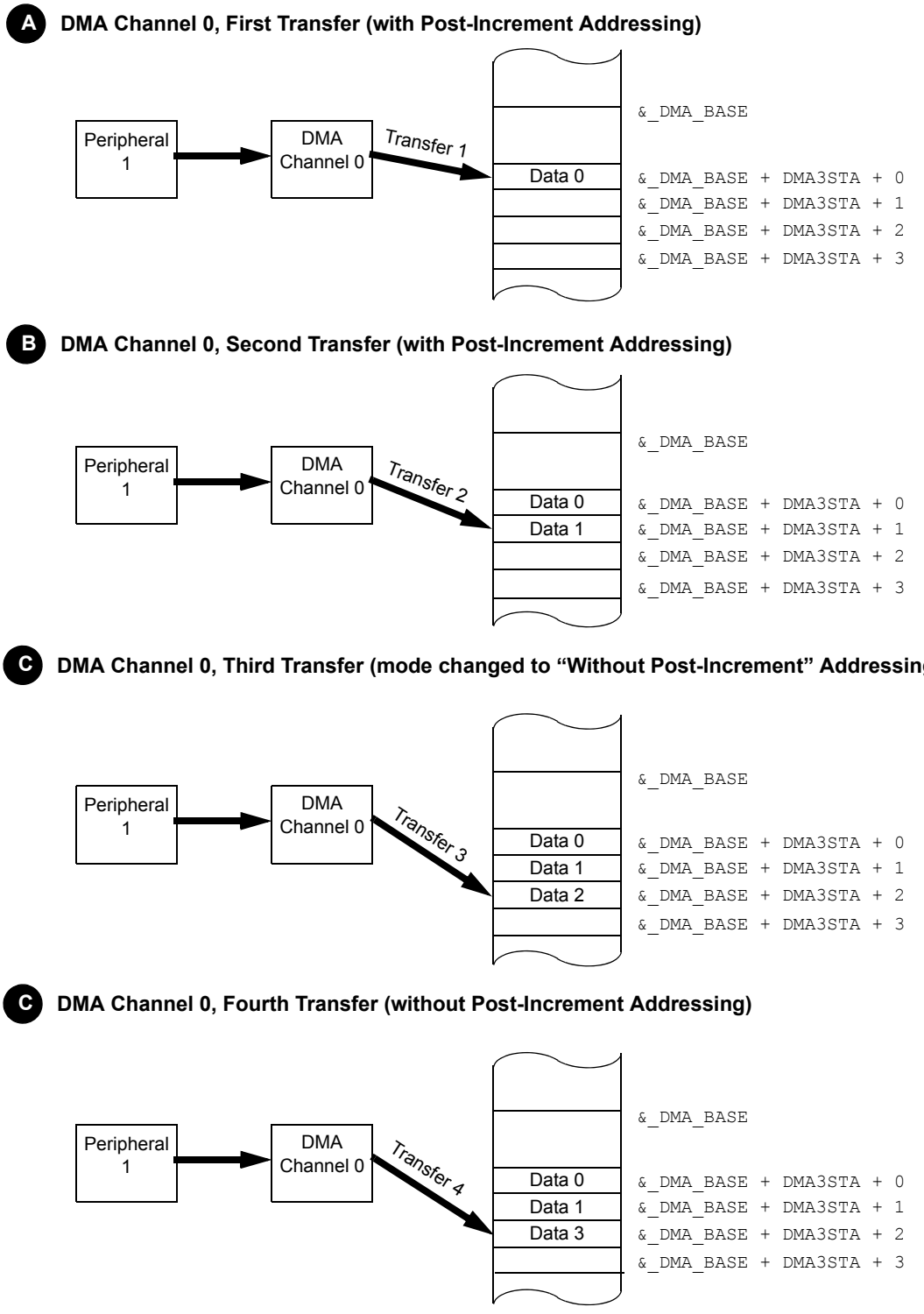
22.6.5 ポスト インクリメント アドレッシングを使用しないレジスタ間接モード

ポスト インクリメント アドレッシングを使用しないレジスタ間接モードでは、転送ごとにデータ バッファの開始アドレスをインクリメントせずにデータ ブロックを移動します。このモードでは、DPSRAM 開始アドレス オフセット (DMAxSTA または DMAxSTB) レジスタで DPSRAM バッファの開始アドレスに対するオフセットを指定します。DMA データ転送実行時、DPSRAM アドレスは次の位置にインクリメントされません。このため、次の DMA データ転送は同じ DPSRAM アドレスで開始します。

このモードを選択するには、DMA チャンネル制御 (DMAxCON) レジスタのアドレッシング モード選択ビット (AMODE<1:0>) を「01」にプログラムします。

DMA チャンネルがアクティブな状態で (つまり、DMA 転送が何回か発生した後)、アドレッシング モードをポスト インクリメント アドレッシングを使用しないレジスタ間接モードに変更する場合、DMA DPSRAM アドレスは現行の DPSRAM バッファ位置 (DMAxSTA または DMAxSTB の内容ではない、これらはこの時点で現行の DPSRAM バッファ位置とは異なる可能性がある) を指します。図 22-9 に、周辺モジュールから DMA DPSRAM へのデータ転送を、ポスト インクリメント アドレッシングを使用する場合と使用しない場合を対比させる形で示します。

図 22-9: ポスト インクリメント アドレッシングを使用する場合と使用しない場合のデータ転送の対比



例 22-4: 入力キャプチャと、ポスト インクリメント アドレッシングを使用しないレジスタ間接モードの DMA のコード

入力キャプチャ 1 モジュールを DMA 動作に設定 :

```
IC1CON = 0;                    // Reset IC module
IC1CONbits.ICTMR = 1;          // Select Timer2 contents for capture
IC1CONbits.ICM   = 2;          // Capture every falling edge
IC1CONbits.ICI   = 0;          // Generate DMA request on every capture event
```

入力キャプチャ モジュールが Timer2 を使用するように設定 :

```
PR2 = 0xBF;                    // Initialize count value
T2CONbits.TON = 1;             // Start timer
```

DMA チャンネル 0 をポストインクリメントを使用しないモードに設定 :

```
unsigned int CaptureValue __attribute__((space(dma)));

DMA0CONbits.AMODE = 1;         // Configure DMA for Register indirect
                               // without post-increment
DMA0CONbits.MODE = 0;          // Configure DMA for Continuous mode
DMA0PAD = (volatile unsigned int)&IC1BUF; // Point DMA to IC1BUF
DMA0CNT = 0;                   // Interrupt after each transfer
DMA0REQ = 1;                   // Select Input Capture module as DMA Request source

DMA3STA = __builtin_dmaoffset(&CaptureValue);

IFS0bits.DMA0IF = 0;           // Clear the DMA interrupt flag bit
IEC0bits.DMA0IE = 1;           // Set the DMA interrupt enable bit

DMA0CONbits.CHEN = 1;          // Enable DMA
```

DMA チャンネル 0 割り込みハンドラの設定 :

```
void __attribute__((__interrupt__)) _DMA3Interrupt(void)
{
    /* Process CaptureValue variable here*/

    IFS0bits.DMA0IF = 0;       //Clear the DMA3 Interrupt Flag
}
```

22.6.6 周辺モジュール間接アドレッシング モード

周辺モジュール間接アドレッシング モードは、DMA チャンネルではなく周辺モジュールが DPSRAM アドレスの可変部分を提供するという特殊なアドレッシング モードです。つまり、DMA チャンネルが固定のバッファ ベースアドレスを提供する一方で、周辺モジュールが DPSRAM アドレスの下位ビット (LSb) を生成します。しかし、DMA チャンネルは引き続き実際のデータ転送を調整し、転送カウントを記録し、対応する CPU 割り込みを生成します。

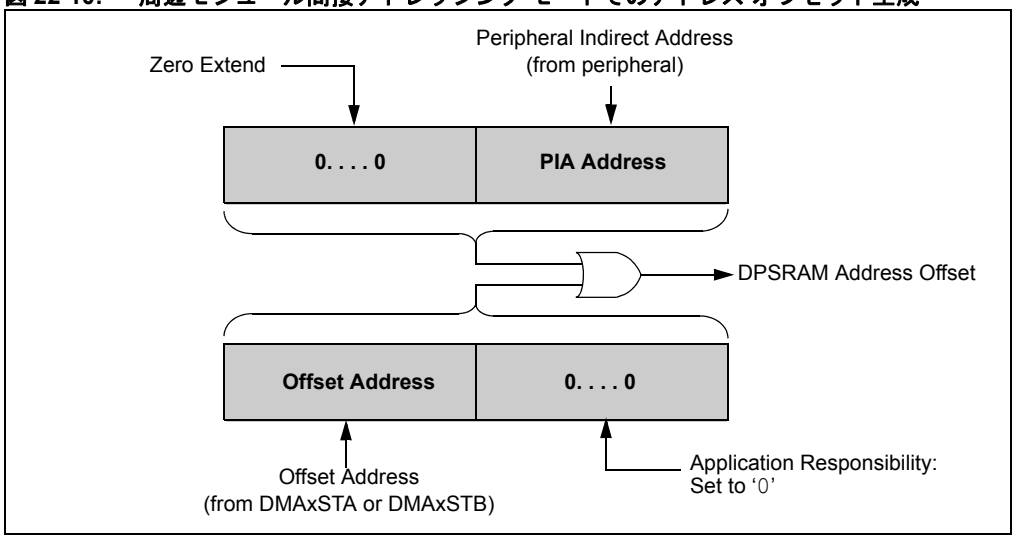
周辺モジュール間接アドレッシング モードでは、周辺モジュールの必要に応じて双方向で動作可能であるため、ターゲットとなる周辺モジュールの読み書きをサポートするために DMA チャンネルも適切に設定する必要があります。

周辺モジュール間接アドレッシング モードを選択するには、DMA チャンネル制御 (DMAxCON) レジスタのアドレッシング モード選択ビット (AMODE<1:0>) を「1x」にプログラムします。

周辺モジュール間接アドレッシング モードの DMA 機能は、その機能をサポートする各周辺モジュールでの必要に応じて個別に調整する事が可能です。周辺モジュールは、DPSRAM 内のデータにアクセスするためのアドレス シーケンスを定義し、例えば、入力 ADC データを複数のバッファに並べ換えて CPU のタスクを軽減する事ができます。

周辺モジュールが周辺モジュール間接アドレッシング モードをサポートしている場合、その周辺モジュールからの DMA 要求割り込みには、DMA チャンネルに渡すアドレスが付与されます。要求に応答する DMA チャンネルも周辺モジュール間接アドレッシングが有効な場合、バッファ ベースアドレスと、入力される周辺モジュール間接アドレスをゼロ拡張した値を論理 OR 演算して、実際の DPSRAM オフセット アドレスを生成します (図 22-10 参照)。

図 22-10:　周辺モジュール間接アドレッシング モードでのアドレス オフセット生成



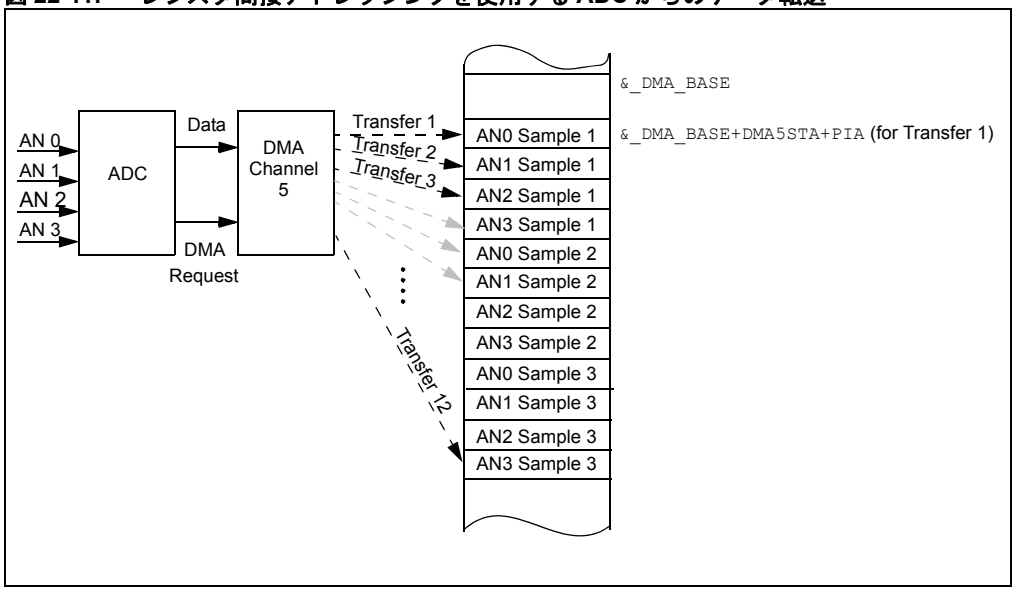
周辺モジュールによって、制御する下位アドレス ビット数が異なります。アプリケーション プログラムは DPSRAM 内のバッファ用にベースアドレスを選択し、そのアドレス オフセットの対応する数の下位ビットを確実にゼロにする必要があります。他のモード同様に、DPSRAM 開始アドレス オフセット レジスタを読み出すと、最新の DPSRAM 転送アドレス オフセットの値が返されます。この中には、前述のアドレス オフセット計算結果が含まれます。DMA チャンネルを周辺モジュール間接アドレッシングに設定していない場合、入力アドレスは無視されデータ転送は通常モードで実行されます。

周辺モジュール間接アドレッシング モードは、他の全ての動作モードと互換性があり、現在、ADC モジュールと ECAN モジュールでサポートされています。

22.6.6.1　DMA アドレス生成に対する ADC のサポート

周辺モジュール間接アドレッシング モードでは、周辺モジュールが周辺モジュールの機能に合うようにアドレス シーケンスを定義します。例えば、入力 0 ～ 3 を順番に連続して変換するように ADC を設定し (例 : 0, 1, 2, 3, 0, 1...)、ポスト インクリメントを使用するレジスタ間接アドレッシング モードに設定した DMA チャンネルに関連付けている場合、DMA 転送はこのデータをシーケンシャル バッファに移動します (図 22-11 参照)。例 22-5 にこの設定のコードを示します。

図 22-11:　レジスタ間接アドレッシングを使用する ADC からのデータ転送



例 22-5: レジスタ間接アドレッシングを使用する ADC からのデータ転送用のコード

ADC1 をチャンネル 0 ～ 3 のサンプリングに設定 :

```
AD1CON1bits.FORM = 3;          // Data Output Format: Signed Fraction (Q15 format)
AD1CON1bits.SSRC = 2;          // Sample Clock Source: GP Timer starts conversion
AD1CON1bits.ASAM = 1;          // Sampling begins immediately after conversion
AD1CON1bits.AD12B = 0;         // 10-bit ADC operation
AD1CON1bits.SIMSAM = 0;        // Samples individual channels sequentially

AD1CON2bits.BUFM = 0;
AD1CON2bits.CSCNA = 1;         // Scan CH0+ Input Selections during Sample A bit
AD1CON2bits.CHPS = 0;          // Converts CH0

AD1CON3bits.ADRC = 0;          // ADC Clock is derived from Systems Clock
AD1CON3bits.ADCS = 63;         // ADC Conversion Clock

//AD1CHS0: A/D Input Select Register
AD1CHS0bits.CH0SA = 0;         // MUXA +ve input selection (AIN0) for CH0
AD1CHS0bits.CH0NA = 0;         // MUXA -ve input selection (VREF-) for CH0

//AD1CHS123: A/D Input Select Register
AD1CHS123bits.CH123SA = 0;     // MUXA +ve input selection (AIN0) for CH1
AD1CHS123bits.CH123NA = 0;     // MUXA -ve input selection (VREF-) for CH1

//AD1CSSH/AD1CSSL: A/D Input Scan Selection Register
AD1CSSH = 0x0000;
AD1CSSL = 0x000F;              // Scan AIN0, AIN1, AIN2, AIN3 inputs
```

ADC1 変換をトリガするよう Timer3 を設定 :

```
TMR3 = 0x0000;
PR3 = 4999;                    // Trigger ADC1 every 125 µs @ 40 MIPS
IFS0bits.T3IF = 0;             // Clear Timer3 interrupt
IEC0bits.T3IE = 0;             // Disable Timer3 interrupt

T3CONbits.TON = 1;             //Start Timer3
```

DMA チャンネル 5 をポスト インクリメント アドレッシングを使用するレジスタ間接モードに設定 :

```
unsigned int BufferA[32] __attribute__((space(dma)));
unsigned int BufferB[32] __attribute__((space(dma)));

DMA5CONbits.AMODE = 0;         // Configure DMA for Register indirect mode
                               // with post-increment
DMA5CONbits.MODE = 2;          // Configure DMA for Continuous Ping-Pong mode
DMA5PAD = (volatile unsigned int)&ADC1BUF0; // Point DMA to ADC1BUF0
DMA5CNT = 31;                  // 32 DMA request
DMA5REQ = 13;                  // Select ADC1 as DMA Request source

DMA5STA = __builtin_dmaoffset(BufferA);
DMA5STB = __builtin_dmaoffset(BufferB);

IFS3bits.DMA5IF = 0;           //Clear the DMA interrupt flag bit
IEC3bits.DMA5IE = 1;           //Set the DMA interrupt enable bit

DMA5CONbits.CHEN=1;            // Enable DMA
```

例 22-5: レジスタ間接アドレッシングを使用するADCからのデータ転送用のコード(続き)

DMA チャンネル 5 割り込みハンドラの設定 :

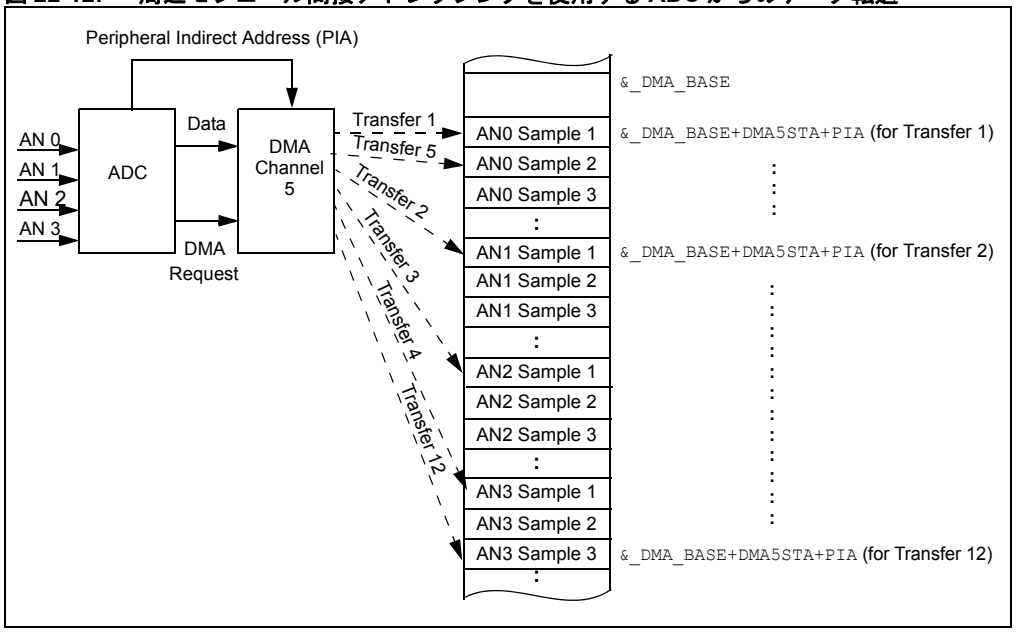
```
unsigned int DmaBuffer = 0;

void __attribute__((__interrupt__)) _DMA5Interrupt(void)
{
    // Switch between Primary and Secondary Ping-Pong buffers
if(DmaBuffer == 0)
    {
        ProcessADCSamples(BufferA);
    }
    else
    {
        ProcessADCSamples(BufferB);
    }

    DmaBuffer ^= 1;

    IFS3bits.DMA5IF = 0;       //Clear the DMA5 Interrupt Flag
}
```

ADC1 を DMA 動作に設定 :

```
AD1CON1bits.ADDMABM = 0;       // Don't Care: ADC address generation is
                               // Ignored by DMA
AD1CON2bits.SMPI    = 3;       // Don't Care
AD1CON4bits.DMABL   = 3;       // Don't Care

IFS0bits.AD1IF      = 0;       // Clear the A/D interrupt flag bit
IEC0bits.AD1IE      = 0;       // Do Not Enable A/D interrupt
AD1CON1bits.ADON    = 1;       // Turn on the A/D converter
```

標準的なアルゴリズムは ADC データ チャンネル ベースで動作する事から、転送データを並べ換えるか不要データを飛び越すようインデックスを付ける必要があります。いずれの方法も、多くのコードと多くの実行時間を必要とします。ADC 周辺モジュール間接アドレッシング モードでは、各 ADC チャンネルのデータを固有のバッファに格納する、特殊なアドレッシング方式を定義します。上記の例では、DMA チャンネルを周辺モジュール間接アドレッシング モードに設定する場合、DMA 転送は ADC データを個別のバッファに移動します (図 22-12 参照)。

図 22-12: 周辺モジュール間接アドレッシングを使用する ADC からのデータ転送



この ADC アドレッシングを有効にするには、ADCx 制御 1 (ADxCON1) レジスタの DMA バッファ構築モード (ADDMABM) ビットをクリアする必要があります。このビットをセットする場合、ADC は変換順 (ポスト インクリメントを使用する DMA レジスタ間接モードと同様) にアドレスを生成します。

前述のように、DPSRAM 開始アドレス オフセット レジスタ (DMAxSTA と DMAxSTB) をユーザ アプリケーションが初期化する場合、周辺モジュール用に予約されている下位ビット数に特に注意が必要です。ADC の場合、ビット数は ADC バッファのサイズと数によって決まります。

ADC のバッファ数は、ADCx 制御 2 (ADxCON2) レジスタの DMA アドレスのインクリメント レート ビット (SMPI<3:0>) で初期化します。各 ADC バッファのサイズは、ADCx 制御 4 (ADCxCON4) レジスタのアナログ入力ごとの DMA バッファ位置ビット (DMABL<2:0>) の数で初期化します。例えば、SMPI<3:0> を 3 に、DMABL<2:0> を 3 に初期化する場合、ADC バッファは 4 つ (SMPI<3:0> + 1) で、それぞれに 8 つのワード ($2^{DMABL<2:0>}$) があり、合計では 32 ワード (64 バイト) となります。つまり、DMAxSTA と DMAxSTB に書き込まれるアドレス オフセットは、下位 6 ビット ($2^{6\ ビット}$ = 64 バイト) をゼロにセットする必要があります。

MPLAB® C30 コンパイラで DMAxSTA と DMAxSTAB レジスタを初期化する場合、データ属性で適切なデータ配置を指定する必要があります。上記の条件で、例 22-6 に示すコードは DMAxSTA と DMAxSTB レジスタを正しく初期化します。

例 22-6: MPLAB® C30 での DMA バッファ配置

```
int  BufferA[4][8] __attribute__((space(dma),aligned(64)));
int  BufferB[4][8] __attribute__((space(dma),aligned(64)));

DMA0STA = __builtin_dmaoffset(BufferA);
DMA0STB = __builtin_dmaoffset(BufferB);
```

例 22-7 にこの設定のコードを示します。

例 22-7:　ADC と周辺モジュール間接アドレッシングを使用する DMA のコード

ADC1 をチャンネル 0 ～ 3 のサンプリングに設定 :

```
AD1CON1bits.FORM = 3;           // Data Output Format: Signed Fraction (Q15 format)
AD1CON1bits.SSRC = 2;           // Sample Clock Source: GP Timer starts conversion
AD1CON1bits.ASAM = 1;           // Sampling begins immediately after conversion
AD1CON1bits.AD12B = 0;          // 10-bit ADC operation
AD1CON1bits.SIMSAM = 0;         // Samples multiple channels sequentially

AD1CON2bits.BUFM = 0;
AD1CON2bits.CSCNA = 1;          // Scan CH0+ Input Selections during Sample A bit
AD1CON2bits.CHPS = 0;           // Converts CH0

AD1CON3bits.ADRC = 0;           // ADC Clock is derived from Systems Clock
AD1CON3bits.ADCS = 63;          // ADC Conversion Clock

//AD1CHS0: A/D Input Select Register
AD1CHS0bits.CH0SA = 0;          // MUXA +ve input selection (AIN0) for CH0
AD1CHS0bits.CH0NA = 0;          // MUXA -ve input selection (VREF-) for CH0

//AD1CHS123: A/D Input Select Register
AD1CHS123bits.CH123SA = 0;      // MUXA +ve input selection (AIN0) for CH1
AD1CHS123bits.CH123NA = 0;      // MUXA -ve input selection (VREF-) for CH1

//AD1CSSH/AD1CSSL: A/D Input Scan Selection Register
AD1CSSH = 0x0000;
AD1CSSL = 0x000F;               // Scan AIN0, AIN1, AIN2, AIN3 inputs
```

ADC1 変換をトリガするよう Timer3 を設定 :

```
TMR3 = 0x0000;
PR3 = 4999; // Trigger ADC1 every 125usec
IFS0bits.T3IF = 0;              // Clear Timer3 interrupt
IEC0bits.T3IE = 0;              // Disable Timer3 interrupt

T3CONbits.TON = 1;              //Start Timer3
```

DMA チャンネル 5 を周辺モジュール間接アドレッシングに設定 :

```
struct
{
    unsigned int Adc1Ch0[8];
    unsigned int Adc1Ch1[8];
    unsigned int Adc1Ch2[8];
    unsigned int Adc1Ch3[8];
} BufferA __attribute__((space(dma)));

struct
{
    unsigned int Adc1Ch0[8];
    unsigned int Adc1Ch1[8];
    unsigned int Adc1Ch2[8];
    unsigned int Adc1Ch3[8];
} BufferB __attribute__((space(dma)));

int BufferA[4][8] __attribute__((space(dma), aligned(64)));
BufferB[4][8] __attribute__((space(dma), aligned(64)));

DMA5CONbits.AMODE = 2;          // Configure DMA for Peripheral indirect mode
DMA5CONbits.MODE = 2;           // Configure DMA for Continuous Ping-Pong mode
DMA5PAD = (volatile unsigned int)&ADC1BUF0; // Point DMA to ADC1BUF0
DMA5CNT = 31;                   // 32 DMA request (4 buffers, each with 8 words)
DMA5REQ = 13;                   // Select ADC1 as DMA Request source
DMA5STA = __builtin_dmaoffset(BufferA);
DMA5STB = __builtin_dmaoffset(BufferB);

IFS3bits.DMA5IF = 0;            //Clear the DMA interrupt flag bit
IEC3bits.DMA5IE = 1;            //Set the DMA interrupt enable bit

DMA5CONbits.CHEN=1;             // Enable DMA
```

例 22-7:　ADC と周辺モジュール間接アドレッシングを使用する DMA のコード (続き)

DMA チャンネル 5 割り込みハンドラの設定 :

```
unsigned int DmaBuffer = 0;

void __attribute__((__interrupt__)) _DMA5Interrupt(void)
{
    // Switch between Primary and Secondary Ping-Pong buffers
if(DmaBuffer == 0)
    {
        ProcessADCSamples(BufferA.Adc1Ch0);
        ProcessADCSamples(BufferA.Adc1Ch1);
        ProcessADCSamples(BufferA.Adc1Ch2);
        ProcessADCSamples(BufferA.Adc1Ch3);
    }
    else
    {
        ProcessADCSamples(BufferB.Adc1Ch0);
        ProcessADCSamples(BufferB.Adc1Ch1);
        ProcessADCSamples(BufferB.Adc1Ch2);
        ProcessADCSamples(BufferB.Adc1Ch3);
    }

    DmaBuffer ^= 1;

    IFS3bits.DMA5IF = 0;        //Clear the DMA5 Interrupt Flag
}
```

ADC1 を DMA 動作に設定 :

```
AD1CON1bits.ADDMABM = 0;        // DMA buffers are built in scatter/gather mode
AD1CON2bits.SMPI    = 3;        // 4 ADC buffers
AD1CON4bits.DMABL   = 3;        // Each buffer contains 8 words

IFS0bits.AD1IF      = 0;        // Clear the A/D interrupt flag bit
IEC0bits.AD1IE      = 0;        // Do Not Enable A/D interrupt
AD1CON1bits.ADON    = 1;        // Turn on the A/D converter
```

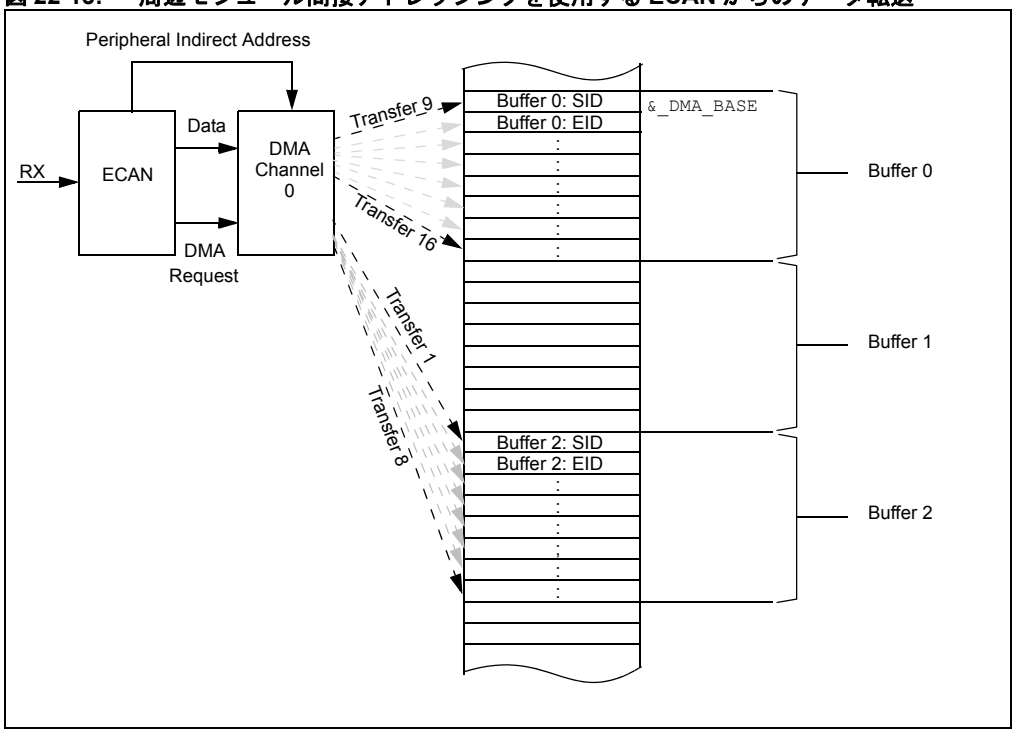
22.6.6.2　DMA アドレス生成に対する ECAN のサポート

周辺モジュール間接アドレッシングを ECAN モジュールと共に使用して、ECAN でさらに個別性の高いアドレッシング機能を定義する事も可能です。dsPIC33F が CAN バス経由のメッセージをフィルタ処理して受信する場合、メッセージを以下の 2 つのグループに分類できます。

- 処理する必要がある受信メッセージ
- 未処理のまま他の CAN ノードに転送する必要がある受信メッセージ

前者では、ユーザ アプリケーションで処理する前に、受信メッセージを 8 ワードずつのバッファに再構築する必要があります。DMA RAM 内に複数の ECAN バッファが配置されている場合、ECAN 周辺モジュールに入力 (または出力) データ用の RAM アドレスを生成させる方が簡単です (図 22-13 参照)。この例では、バッファ 2 を最初に受信し、バッファ 0 を次に受信します。ECAN モジュールはデスティネーション アドレスを生成し、DMA RAM (周辺モジュール間接アドレッシング) 内にデータを適切に配置します。

図 22-13:　周辺モジュール間接アドレッシングを使用する ECAN からのデータ転送



前述のように、DPSRAM 開始アドレス オフセット レジスタ (DMAxSTA と DMAxSTB) をユーザ アプリケーションが初期化し、DMA が周辺モジュール間接アドレッシング モードで動作する場合、周辺モジュール用に予約されている下位ビット数に特に注意が必要です。ECAN の場合、ビット数は ECAN FIFO 制御レジスタ (CiFCTRL) 内の DMA バッファサイズ ビット (DMABS<2:0>) で定義する ECAN バッファ数によって決まります。

例えば、ECAN モジュールが DMABS<2:0> ビットを「3」にセットして 12 のバッファを確保する場合、各 8 ワードの 12 のバッファで、合計 96 ワードです (192 バイト)。つまり、DMAxSTA と DMAxSTB レジスタに書き込まれるアドレス オフセットは、下位 8 ビット ($2^{8\text{ ビット}}$ = 256 バイト) を「0」にセットする必要があります。MPLAB C30 コンパイラで DMAxSTA レジスタを初期化する場合、データ属性で適切なデータ配置を指定する必要があります。上記の例では、例 22-8 のコードで DMAxSTA レジスタを適切に初期化します。

例 22-8:　MPLAB® C30 での DMA バッファ配置

```
int  BufferA[12][8] __attribute__((space(dma),aligned(256)));

DMA0STA = __builtin_dmaoffset(&BufferA[0][0]);
```

例 22-9 にこの設定のコードを示します。

しかし、入力メッセージ処理は常に必要なわけではありません。例えば、車載アプリケーションの中には、受信メッセージを CPU で処理するのではなく、単に別のノードに転送するだけで済むものもあります。この場合、受信バッファをメモリ内で並び換える必要はなく、使用可能になった時点で転送できます。

このデータ転送モードは、ポスト インクリメントを使用するレジスタ間接モードの DMA で実行可能です。図 22-14 に、このシナリオを示します。

例 22-9:　ECAN と周辺モジュール間接アドレッシングを使用する DMA のコード

ECAN1 に 2 つのフィルタを設定 :

```
/* Initialize ECAN clock first. See ECAN section for example code */


C1CTRL1bits.WIN = 1;          // Enable filter window
C1FEN1bits.FLTEN0 = 1;        // Filter 0 is enabled
C1FEN1bits.FLTEN1 = 1;        // Filter 1 is enabled
C1BUFPNT1bits.F0BP = 0;       // Filter 0 points to Buffer0
C1BUFPNT1bits.F1BP = 2;       // Filter 1 points to Buffer2

C1RXF0SID = 0xFFEA;           // Filter 0 configuration
C1RXF0EID = 0xFFFF;

C1RXF1SID = 0xFFEB;           // Filter 1 configuration
C1RXF1EID = 0xFFFF;

C1FMSKSEL1bits.F0MSK = 0;     // Mask 0 used for both filters
C1FMSKSEL1bits.F1MSK = 0;     // Mask 0 used for both filters
C1RXM0SID = 0xFFEB;
C1RXM0EID = 0xFFFF;

C1FCTRLbits.DMABS = 3;        // 12 buffers in DMA RAM
C1FCTRLbits.FSA   = 3;        // FIFO starts from TX/RX Buffer 3

C1CTRL1bits.WIN     = 0;
C1TR01CONbits.TXEN0 = 0;      // Buffer 0 is a receive buffer
C1TR23CONbits.TXEN2 = 0;      // Buffer 2 is a receive buffer

C1TR01CONbits.TX0PRI = 0b11;  //High Priority
C1TR01CONbits.TX1PRI = 0b10;  //Intermediate High Priority

C1CTRL1bits.REQOP = 0;// Enable Normal Operation Mode
```

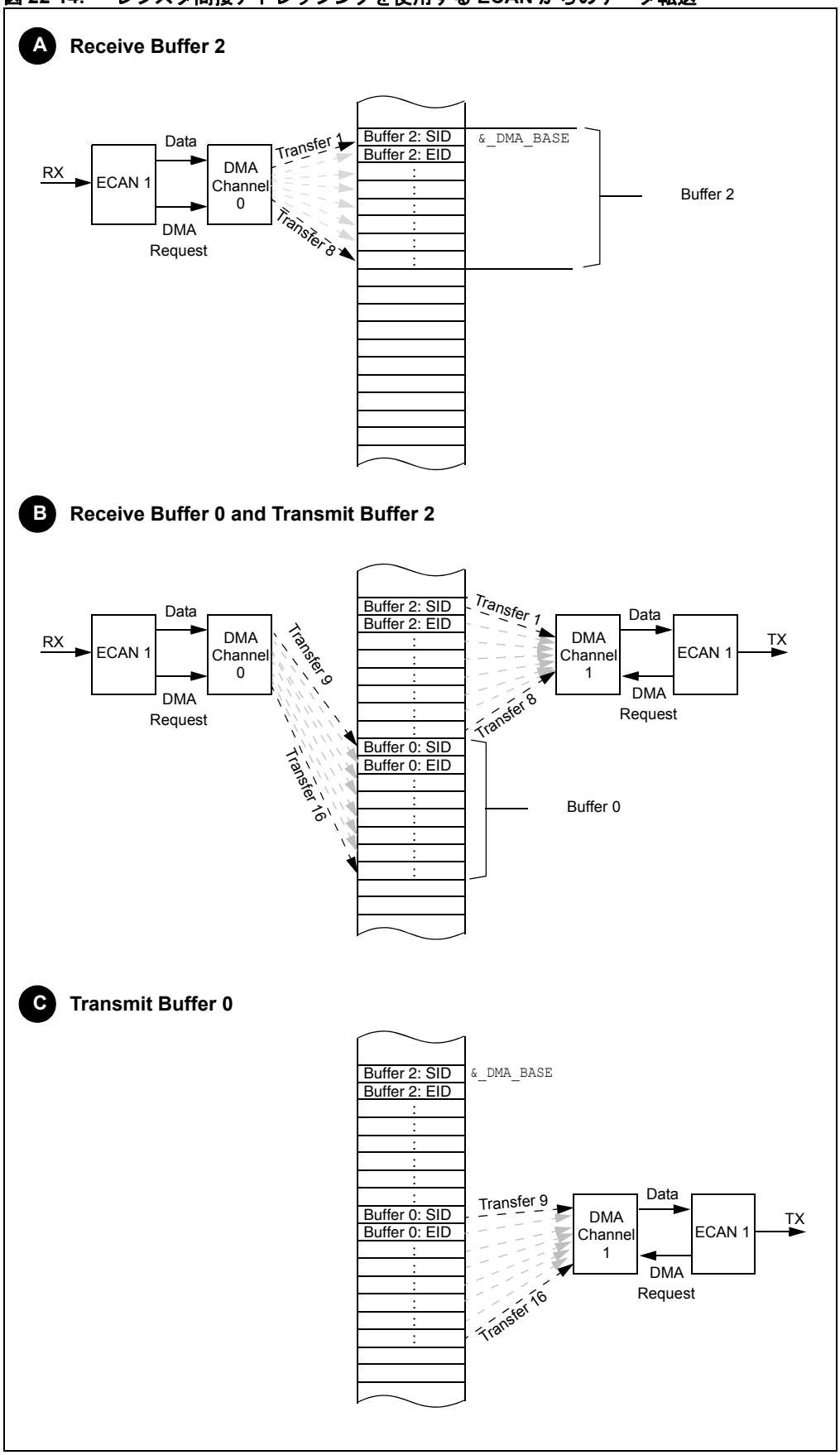
DMA チャンネル 0 を周辺モジュール間接アドレッシングに設定 :

```
unsigned int Ecan1Rx[12][8] __attribute__((space(dma)));// 12 buffers, 8
words each

DMA0CONbits.AMODE = 2;        // Continuous mode, single buffer
DMA0CONbits.MODE = 0;         // Peripheral Indirect Addressing

DMA0PAD = (volatile unsigned int) &C1RXD;    // Point to ECAN1 RX register
DMA0STA = __builtin_dmaoffset(Ecan1Rx);      // Point DMA to ECAN1 buffers

DMA0CNT = 7;                  // 8 DMA request (1 buffer, each with 8 words)
DMA0REQ = 0x22;               // Select ECAN1 RX as DMA Request source

IEC0bits.DMA0IE = 1;          // Enable DMA Channel 0 interrupt
DMA0CONbits.CHEN = 1;         // Enable DMA Channel 0
```

DMA 割り込みハンドラの設定 :

```
void __attribute__((__interrupt__)) _DMA0Interrupt(void)
{
   ProcessData(Ecan1Rx[C1VECbits.ICODE]);    // Process received buffer;

   IFS0bits.DMA0IF = 0;       // Clear the DMA0 Interrupt Flag;
}
```

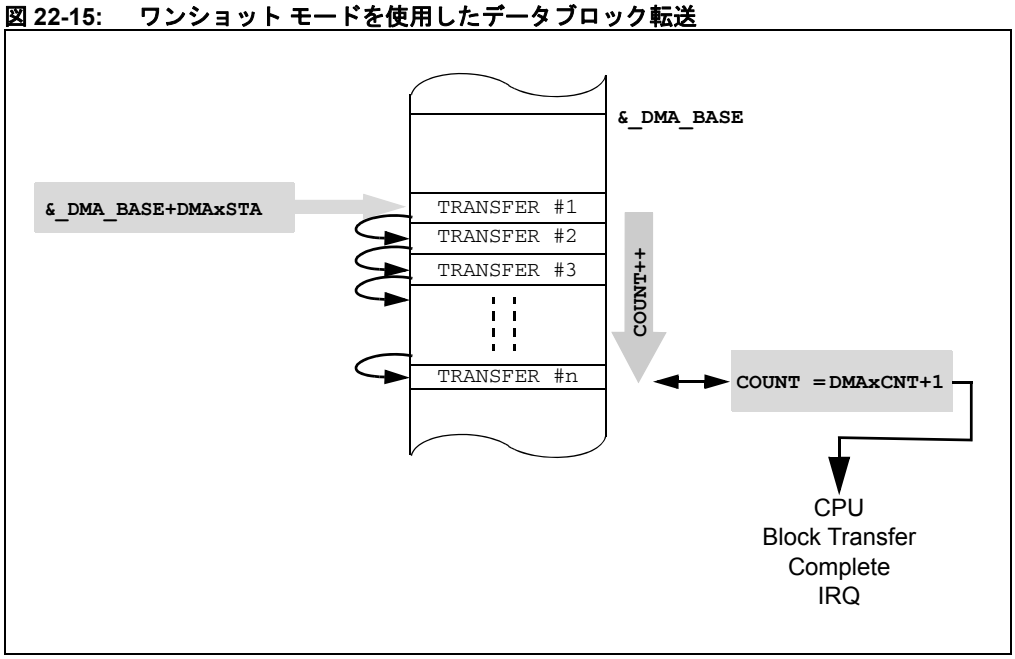
図 22-14:　レジスタ間接アドレッシングを使用する ECAN からのデータ転送



22.6.7 ワンショットモード

ワンショット モードは、繰り返しのデータ転送が必要とされない場合に使用します。ワンショット モードを選択するには、DMA チャンネル制御 (DMAxCON) レジスタの動作モード選択ビット (MODE<1:0>) を「x1」にプログラムします。このモードでは、データブロック全体を移動すると (ブロック長は DMAxCNT によって定義)、データブロックの終わりが検出され、チャンネルが自動的に無効になります (DMA チャンネル制御レジスタ (DMAxCON) 内の CHEN ビットはハードウェアによってクリアされます)。図 22-15 にワンショット モードを示します。

図 22-15: ワンショット モードを使用したデータブロック転送



DMA チャンネル制御 (DMAxCON) レジスタの HALF ビットをセットした場合、データブロック転送の半分が完了した時点で DMAxIF ビットがセットされ (アプリケーション プログラムによって有効にされている場合は DMA 割り込みも生成されます)、チャンネルは有効な状態を維持します。フルブロック転送が完了すると、割り込みフラグはセットされずにチャンネルは自動的に無効になります。ハーフブロック転送とフルブロック転送の両方で割り込みを発生させるための DMA チャンネルの設定方法の詳細は、**22.6.3「フルブロックまたはハーフブロック転送割り込み」**を参照してください。

DMAxCON 内の CHEN ビットを「1」にセットしてチャンネルを再度有効にすると、DPSRAM 開始アドレス オフセット (DMAxSTA と DMAxSTB) レジスタで指定されている開始アドレスからブロック転送を実行します。例 22-10 にワンショット モード動作のコードを示します。

例 22-10: UART とワンショット モードを使用する DMA のコード

UART を RX と TX に設定 :

```
#define FCY      40000000
#define BAUDRATE 9600
#define BRGVAL   ((FCY/BAUDRATE)/16)-1

U2MODEbits.STSEL = 0;      // 1-stop bit
U2MODEbits.PDSEL = 0;      // No Parity, 8-data bits
U2MODEbits.ABAUD = 0;      // Autobaud Disabled

U2BRG = BRGVAL;// BAUD Rate Setting for 9600

U2STAbits.UTXISEL0 = 0;    // Interrupt after one TX character is transmitted
U2STAbits.UTXISEL1 = 0;
U2STAbits.URXISEL = 0;     // Interrupt after one RX character is received

U2MODEbits.UARTEN   = 1;   // Enable UART
U2STAbits.UTXEN     = 1;   // Enable UART TX
```

例 22-10:　UART とワンショット モードを使用する DMA のコード (続き)

DMA チャンネル 0 をワンショット、シングル バッファ モードで送信するように設定 :

```
unsigned int BufferA[8] __attribute__((space(dma)));
unsigned int BufferB[8] __attribute__((space(dma)));


DMA0CON = 0x2001;             // One-Shot, Post-Increment, RAM-to-Peripheral
DMA0CNT = 7;                  // 8 DMA requests
DMA0REQ = 0x001F;             // Select UART2 Transmitter

DMA0PAD = (volatile unsigned int) &U2TXREG;
DMA0STA = __builtin_dmaoffset(BufferA);

IFS0bits.DMA0IF = 0;          // Clear DMA Interrupt Flag
IEC0bits.DMA0IE = 1;          // Enable DMA interrupt
```

DMA チャンネル 1 を連続ピンポンモードの受信用に設定 :

```
DMA1CON = 0x0002;             // Continuous, Ping-Pong, Post-Inc., Periph-RAM
DMA1CNT = 7;                  // 8 DMA requests
DMA1REQ = 0x001E;             // Select UART2 Receiver

DMA1PAD = (volatile unsigned int) &U2RXREG;
DMA1STA = __builtin_dmaoffset(BufferA);
DMA1STB = __builtin_dmaoffset(BufferB);


IFS0bits.DMA1IF = 0;          // Clear DMA interrupt
IEC0bits.DMA1IE = 1;          // Enable DMA interrupt
DMA1CONbits.CHEN = 1;         // Enable DMA Channel
```

DMA 割り込みハンドラの設定 :

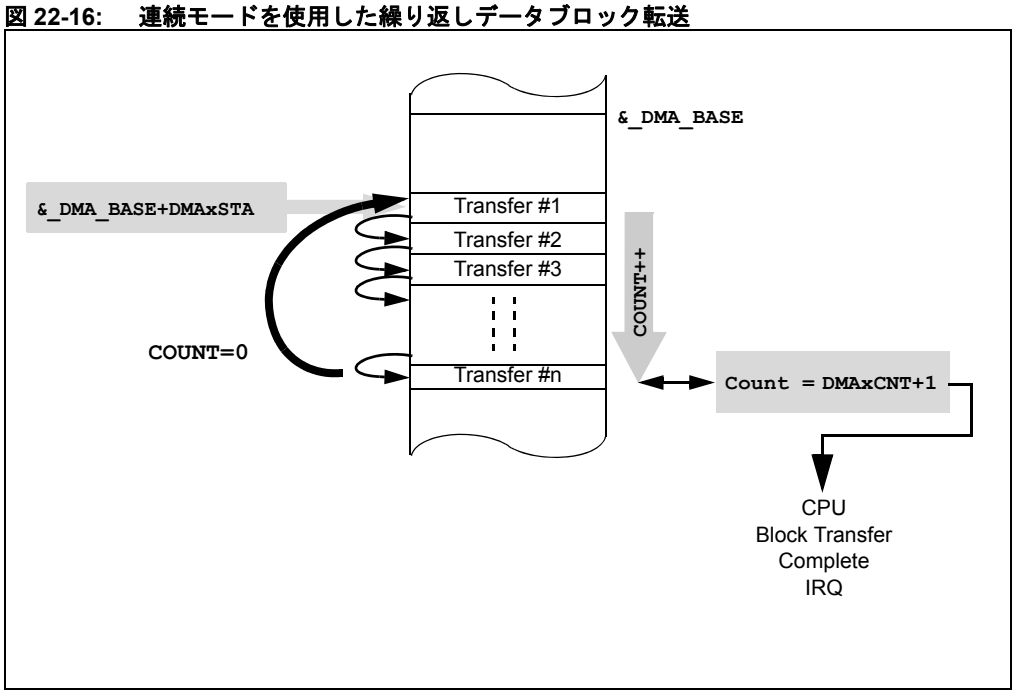
```
void __attribute__((__interrupt__)) _DMA0Interrupt(void)
{
   IFS0bits.DMA0IF = 0;       // Clear the DMA0 Interrupt Flag;
}


void __attribute__((__interrupt__)) _DMA1Interrupt(void)
{
   static unsigned int BufferCount = 0;      // Keep record of which buffer
                                             // contains RX Data

   if(BufferCount == 0)
   {
    DMA0STA = __builtin_dmaoffset(BufferA);  // Point DMA 0 to data
                                             // to be transmitted
   }
   else
 {
    DMA0STA = __builtin_dmaoffset(BufferB);  // Point DMA 0 to data
                                             // to be transmitted
   }

   DMA0CONbits.CHEN = 1;      // Enable DMA0 Channel
   DMA0REQbits.FORCE = 1;     // Manual mode: Kick-start the 1st transfer

   BufferCount ^= 1;
   IFS0bits.DMA1IF = 0;       // Clear the DMA1 Interrupt Flag
}
```

22.6.8 連続モード

連続モードは、プログラム実行中常時繰り返しデータ転送が必要な場合に使用します。

このモードを選択するには、DMA チャンネル制御 (DMAxCON) レジスタの動作モード選択ビット (MODE<1:0>) を「x0」にプログラムします。このモードでは、データブロック全体を移動すると (ブロック長は DMAxCNT によって定義)、データブロックの終わりが検出され、チャンネルは有効な状態を維持します。最後のデータ転送時に、DMA DPSRAM アドレスは (プライマリ)DPSRAM 開始アドレス オフセット A (DMAxSTA) レジスタの値にリセットされます。図 22-16 に連続モードを示します。

図 22-16: 連続モードを使用した繰り返しデータブロック転送



DMA チャンネル制御 (DMAxCON) レジスタの HALF ビットをセットした場合、データブロック転送の半分が完了した時点で DMAxIF ビットがセットされます (有効にされている場合は DMA 割り込みも生成されます)。チャンネルは有効な状態を維持します。フルブロック転送が完了すると、割り込みフラグはセットされずにチャンネルは有効な状態を維持します。ハーフブロック転送とフルブロック転送の両方で割り込みを生成するための DMA チャンネルの設定方法の詳細は、**22.6.3「フルブロックまたはハーフブロック転送割り込み」**を参照してください。

22.6.9 ピンポンモード

ピンポンモードを使用すると、CPU が 1 つのバッファを処理する間に、DMA チャンネルで 2 番目のバッファを使用できます。その結果、CPU は DMA ブロック転送時間全体を利用して、DMA チャンネルが現在使用していないバッファを処理します。当然この転送モードでは、所定のバッファサイズに必要な DPSRAM の量は 2 倍となります。

DMA チャンネルが有効になると、全ての DMA 動作モードで (プライマリ) DMA チャンネル x DPSRAM 開始アドレス オフセット A (DMAxSTA) レジスタが既定値で選択され、DPSRAM 実効アドレス初期値を生成します。ブロック転送が完了し DMA チャンネルが再初期化されるたびに、同じ DMAxSTA レジスタからバッファ開始アドレスを取得します。

ピンポンモードでは、バッファ開始アドレスは以下の 2 つのレジスタから取得します。

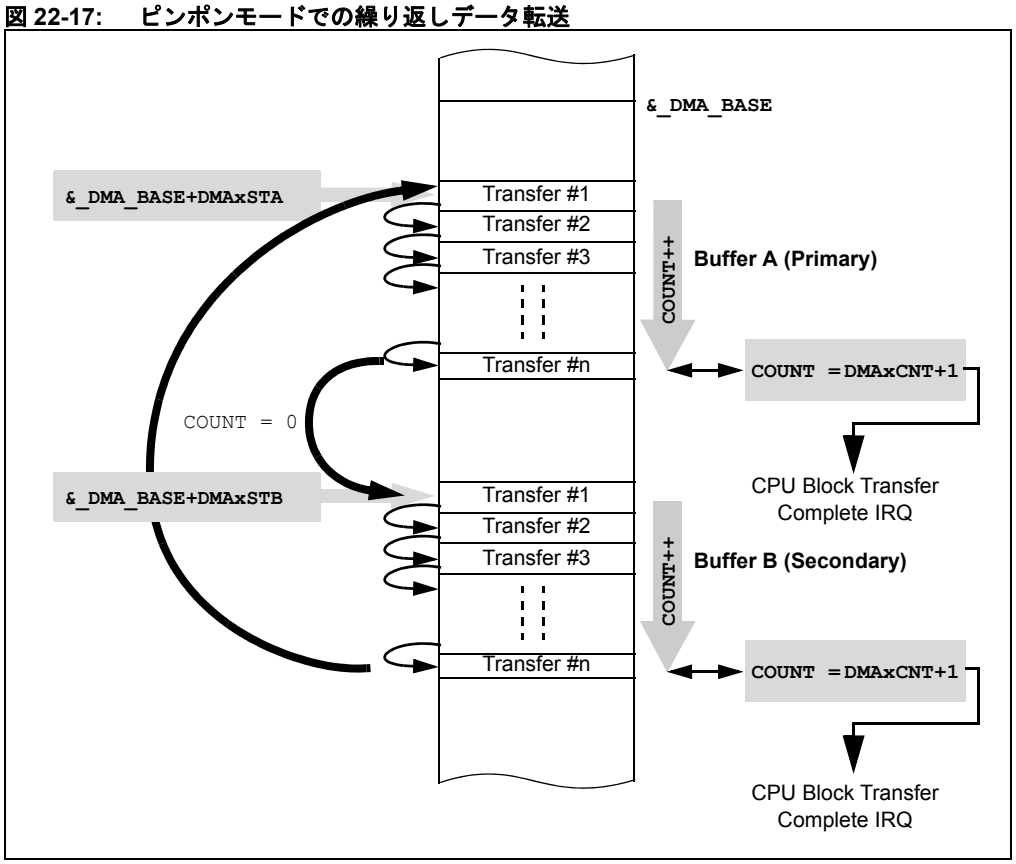
- プライマリ : DMA チャンネル xDPSRAM 開始アドレス オフセット A (DMAxSTA) レジスタ
- セカンダリ : DMA チャンネル xDPSRAM 開始アドレス オフセット B (DMAxSTB) レジスタ

DMA は、代替ブロック転送にセカンダリ バッファを使用します。ブロック転送が完了し DMA チャンネルが再初期化されるたびに、バッファ開始アドレスは代替レジスタから取得します。

ピンポンモードを選択するには、DMA チャンネル制御 (DMAxCON) レジスタの動作モード選択ビット (MODE<1:0>) を「1x」にプログラムします。

DMA がピンポンモードで動作中に連続モードを選択した場合、DMA はプライマリ バッファを転送した後でセカンダリ バッファを指すように再初期化し、続いてセカンダリ バッファを転送します。その後のブロック転送はプライマリ バッファとセカンダリ バッファを交互に使用します。各バッファ転送が終了するたびに、割り込みが生成されます (アプリケーション プログラムによって有効化されている場合)。図 22-17 に、連続動作でのピンポンモードを示します。例 22-11に、DCIモジュールを使用するピンポン動作で使用するサンプルコードを示します。

図 22-17: ピンポンモードでの繰り返しデータ転送



例 22-11:　DCI と連続ピンポン動作を使用する DMA のコード

DCI を RX と TX に設定 :

```
#define FCY    40000000
#define FS     48000
#define FCSCK  64*FS
#define BCGVAL (FCY/(2*FS))-1

DCICON1bits.CSCKD = 0; // Serial Bit Clock (CSCK pin) is output
DCICON1bits.CSCKE = 0; // Data sampled on falling edge of CSCK
DCICON1bits.COFSD = 0; // Frame Sync Signal is output
DCICON1bits.UNFM = 0;  // Transmit '0's on a transmit underflow
DCICON1bits.CSDOM = 0; // CSDO pin drives '0's during disabled TX time slots
DCICON1bits.DJST = 0;  // TX/RX starts 1 serial clock cycle after frame sync pulse
DCICON1bits.COFSM = 1; // Frame Sync Signal set up for I2S mode

DCICON2bits.BLEN = 0;  // One data word will be buffered between interrupts
DCICON2bits.COFSG = 1; // Data frame has 2 words: LEFT & RIGHT samples
DCICON2bits.WS = 15;   // Data word size is 16 bits

DCICON3 = BCG_VAL;// Set up CSCK Bit Clock Frequency

TSCONbits.TSE0 = 1;    // Transmit on Time Slot 0
TSCONbits.TSE1 = 1;    // Transmit on Time Slot 1
RSCONbits.RSE0 = 1;    // Receive on Time Slot 0
RSCONbits.RSE1 = 1;    // Receive on Time Slot 1
```

DMA チャンネル 0 を連続ピンポンモードの送信用に設定 :

```
unsigned int TxBufferA[16] __attribute__((space(dma)));
unsigned int TxBufferB[16] __attribute__((space(dma)));

DMA0CON = 0x2002;      // Ping-Pong, Continous, Post-Increment, RAM-to-Peripheral
DMA0CNT = 15;          // 15 DMA requests
DMA0REQ = 0x003C;      // Select DCI as DMA Request source

DMA0PAD = (volatile unsigned int) &TXBUF0;
DMA0STA = __builtin_dmaoffset(TxBufferA);
DMA0STB = __builtin_dmaoffset(TxBufferB);

IFS0bits.DMA0IF = 0;   // Clear DMA Interrupt Flag
IEC0bits.DMA0IE = 1;   // Enable DMA interrupt
DMA0CONbits.CHEN = 1;  // Enable DMA Channel
```

DMA チャンネル 1 を連続ピンポンモードの受信用に設定 :

```
unsigned int RxBufferA[16] __attribute__((space(dma)));
unsigned int RxBufferB[16] __attribute__((space(dma)));


DMA1CON = 0x0002;      // Continuous, Ping-Pong, Post-Inc., Periph-RAM
DMA1CNT = 15;          // 16 DMA requests
DMA1REQ = 0x003C;      // Select DCI as DMA Request source

DMA1PAD = (volatile unsigned int) &RXBUF0;
DMA1STA = __builtin_dmaoffset(RxBufferA);
DMA1STB = __builtin_dmaoffset(RxBufferB);

IFS0bits.DMA1IF = 0;// Clear DMA interrupt
IEC0bits.DMA1IE = 1;// Enable DMA interrupt
DMA1CONbits.CHEN = 1;// Enable DMA Channel
```

例 22-11: DCI と連続ピンポン動作を使用する DMA のコード (続き)

DMA 割り込みハンドラの設定 :

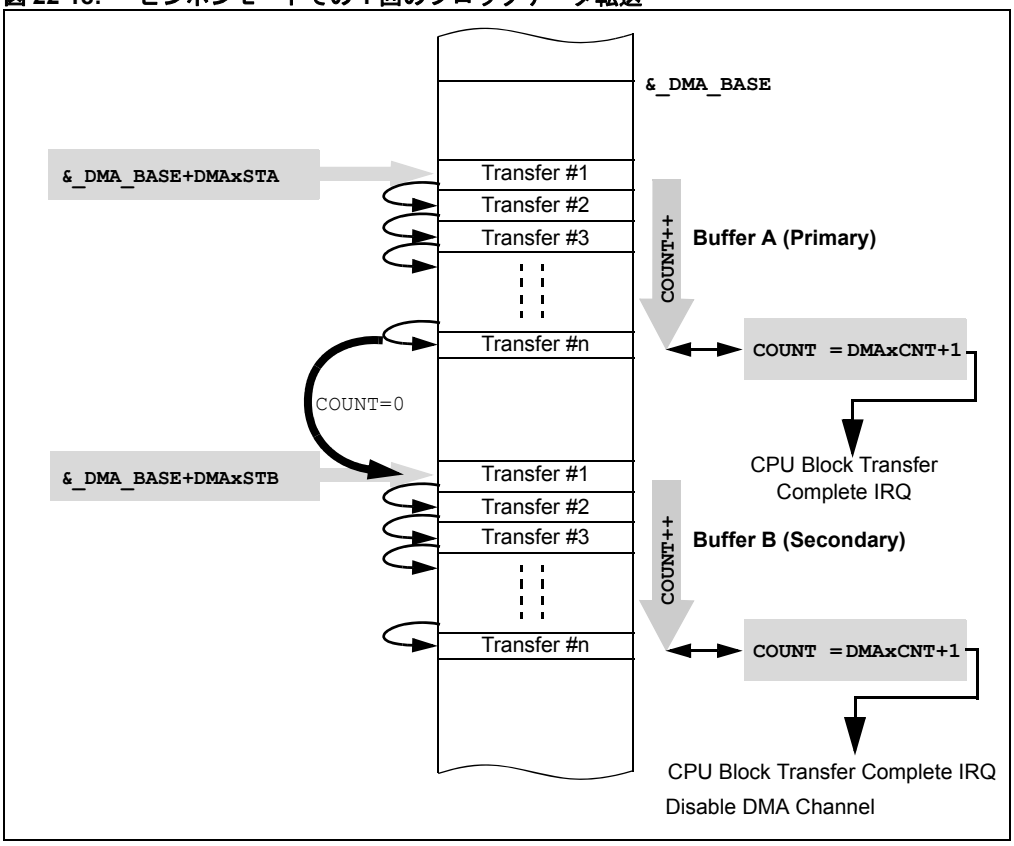
```
void __attribute__((__interrupt__)) _DMA0Interrupt(void)
{
   static unsigned int TxBufferCount = 0;// Keep record of which buffer
                                         // has RX Data

   if(BufferCount == 0)
   {
    /* Notify application that TxBufferA has been transmitted */
   }
   else
   {
    /* Notify application that TxBufferB has been transmitted */
   }

   BufferCount ^= 1;
   IFS0bits.DMA0IF = 0;                  // Clear the DMA0 Interrupt Flag;
}


void __attribute__((__interrupt__)) _DMA1Interrupt(void)
{
   static unsigned int RxBufferCount = 0; // Keep record of which buffer
                                         // has RX Data

   if(BufferCount == 0)
   {
    /* Notify application that RxBufferA has been received */
   }
   else
   {
    /* Notify application that RxBufferB has been received */   }

   BufferCount ^= 1;
   IFS0bits.DMA1IF = 0;                  // Clear the DMA1 Interrupt Flag
}
```

Enable DCI:

```
/* Force First two words to fill-in TX buffer/shift register */
DMA0REQbits.FORCE = 1;
while(DMA0REQbits.FORCE == 1);

DMA0REQbits.FORCE = 1;
while(DMA0REQbits.FORCE == 1);

DCICON1bits.DCIEN = 1;                   // Enable DCI
```

DMA がピンポンモードで動作中にワンショット モードを選択した場合、DMA はプライマリ バッファを転送した後でセカンダリ バッファを指すように再初期化し、続いてセカンダリ バッファを転送します。DMA チャンネルは自動的に無効化されるため、後続のブロック転送は行われません。図 22-18 に、ピンポンモードでのワンショット データ転送を示します。

図 22-18:　ピンポンモードでの 1 回のブロックデータ転送



22.6.10　手動転送モード

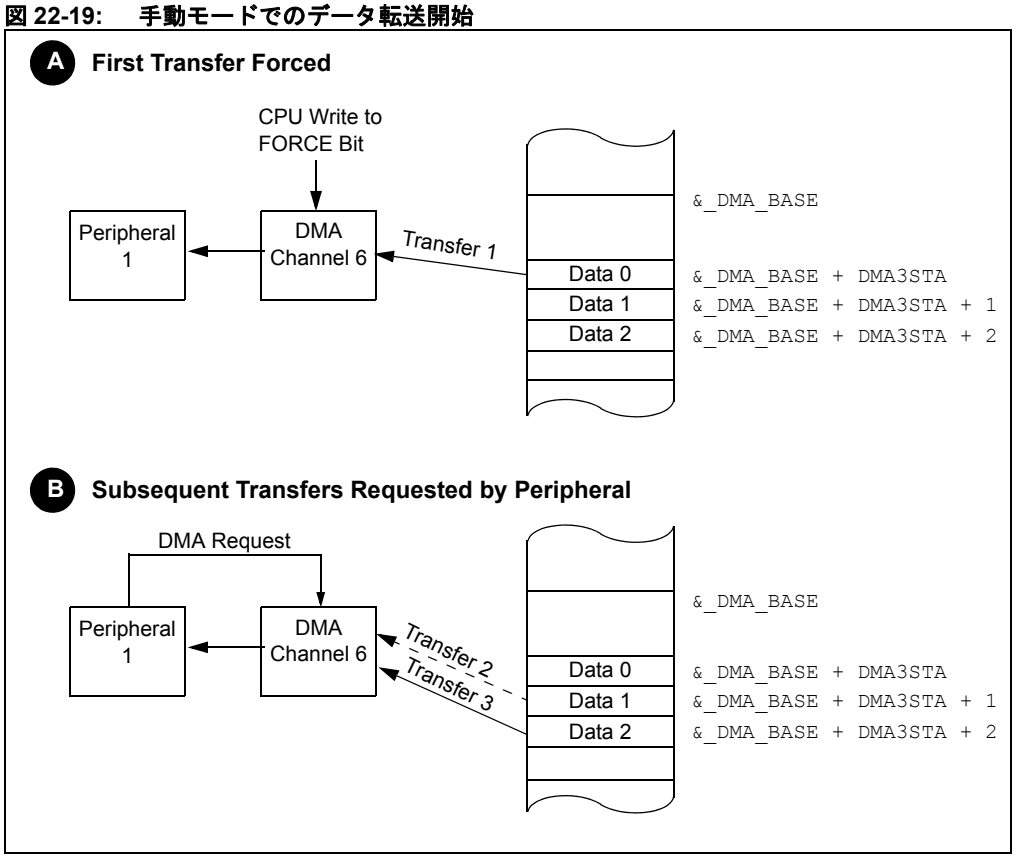
DMA コントローラを使用してデータを DPSRAM に送信する周辺モジュールの場合、DMA データ転送は DMA チャンネルと周辺モジュールの初期化後、自動的に開始します。周辺モジュールはデータを DPSRAM に移動する準備が整うと、DMA 要求を発行します。この時点で周辺モジュールにデータを送信する必要もある場合、同じ DMA 要求を利用して別のチャンネルをアクティブにし、DPSRAM からデータを読み出して周辺モジュールに書き込む事ができます。

一方、アプリケーションが (DPSRAM バッファから) 周辺モジュールへのデータ送信のみ必要である場合、処理を開始するために周辺モジュールに対する最初の (手動) データ読み込みが必要となる事があります (**22.7「DMA 転送の開始」**参照)。この処理はソフトウェアでも実行可能です。しかし、より便利な方法は、選択した DMA チャンネル内のビットをセットして、チャンネルの DMA 要求を模倣する事です。DMA チャンネルはこの要求を他の要求と同様に処理し、最初のデータ要素を転送してシーケンスを開始します。周辺モジュールは次のデータの準備が整うと通常の DMA 要求を送信し、DMA は次のデータ要素を送信します。図 22-19 にこのプロセスを示します。

手動 DMA 要求は、DMA チャンネル x IRQ 選択 (DMAxREQ) レジスタの FORCE ビットをセットする事によって生成できます。セットすると、FORCE ビットはユーザ アプリケーションからはクリアできません。このビットは、強制 DMA 転送の完了後にハードウェアによってクリアする必要があります。FORCE ビットをセットするタイミングによって、以下のいずれかの条件が適用されます。

- DMA 転送実行中に FORCE ビットをセットすると、何も実行されず変化も生じません。
- チャンネル x の設定中に FORCE ビットをセットする (DMA チャンネルを設定する書き込みサイクル中に FORCE ビットをセットする) と、予期せぬ挙動を示す可能性があるため、避ける必要があります。
- (当該チャンネルに対する) 周辺モジュール割り込み要求が保留中に FORCE ビットのセットを試みても、割り込みベースの要求が優先されるため、破棄されます。しかし、DMA コントローラ ステータス 0 (DMACS0) レジスタの DMA RAM 書き込みコリジョンフラグ ビット (XWCOLx) と周辺モジュール書き込みコリジョンフラグ ビット (PWCOLx) の両方をセットするとエラー条件が発生します。詳細は **22.10「データ書き込みコリジョン」**を参照してください。

図 22-19:　手動モードでのデータ転送開始



22.6.11　NULL データ書き込みモード

NULL データ書き込みモードは、SPI のようにデータ送信を行わず連続してデータを受信する必要があるアプリケーションに最適です。

SPI は実質的に、クロック周期ごとにビットデータをクロック入出力する単純なシフトレジスタです。しかし、SPI をマスタモードに設定し (SPI がクロックのソース) て受信データのみを処理すると、異常な状況が発生します。この場合、SPI データクロックを開始し外部データを受信するには、SPI データレジスタに何か書き込む必要があります。

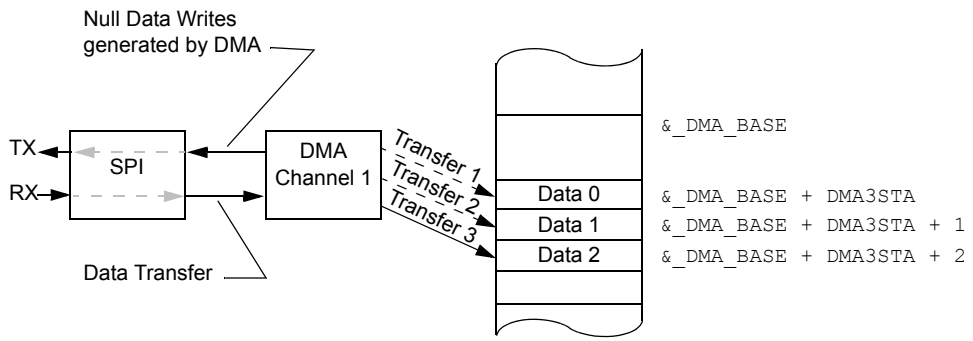
2 つの DMA チャンネルを割り当て、1 つをデータ受信用にし、もう 1 つは SPI への NULL (ゼロ) データ供給用とする事が可能です。しかし、さらに効率的な解決策は DMA NULL データ書き込みモードを使用する事です。このモードでは、周辺モジュールのデータ読み出し用に設定されている DMA チャンネルがデータ要素の送受信を行うたびに、SPI データレジスタに自動的に NULL 値を書き込みます。

DMA チャンネル x 制御 (DMAxCON) レジスタの NULL データ周辺モジュール書き込みモード選択ビット (NULLW) をセットし、DMA チャンネルを周辺モジュールからの読み出し用に設定した場合、DMA チャンネルは周辺モジュールのデータ読み出しと同じサイクル内で周辺モジュール アドレスへの NULL (全てゼロ) 書き込みも実行します。この書き込みは、DPSRAM への (データ) 書き込み (DPSRAM バス経由) と併行して周辺モジュールバス経由で行います。図 22-20 に、このモードを示します。

このモードでの通常動作中、NULL データ書き込みは周辺モジュール DMA 要求に応答する場合 (データ受信が完了して転送準備が整っている状態) にのみ発生します。最初のワード受信を開始するには、最初に周辺モジュールに対する CPU 書き込みが必要です。その後は DMA が周辺モジュールへの (NULL) データ書き込みを処理します。つまり、CPU NULL 書き込みが SPI (マスタ) データ送受信を開始し、その結果 DMA 要求が生成されて新たに受信したデータを移動します。

あるいは、強制 DMA 転送を使用してプロセスを「キックスタート」する事もできます。しかし、これには冗長な周辺モジュール読み出し (無効データ) と、これに関連した DPSRAM ポインタの調整も含まれるため、それらを考慮する必要があります。

図 22-20: NULL データ書き込みモードでのデータ転送



例 22-12: SPI と NULL データ書き込みモードでの DMA

SPI をマスタモードに設定 :

```
SPI1CON1bits.MODE16 = 1;      // Communication is word-wide (16 bits)
SPI1CON1bits.MSTEN = 1;       // Master Mode Enabled
SPI1STATbits.SPIEN = 1;       // Enable SPI Module
```

DMA チャンネル 1 を NULL データ書き込みモードに設定 :

```
unsigned int BufferA[16] __attribute__((space(dma)));
unsigned int BufferB[16] __attribute__((space(dma)));

DMA1CON = 0x0802;             // Null Write, Continuous, Ping-Pong,
                              // Post-Increment, Periph-to-RAM
DMA1CNT = 15;                 // Transfer 16 words at a time
DMA1REQ = 0x000A;             // Select SPI1 as DMA request source

DMA1STA = __builtin_dmaoffset(BufferA);
DMA1STB = __builtin_dmaoffset(BufferB);
DMA1PAD = (volatile unsigned int) &SPI1BUF;

IFS0bits.DMA1IF = 0;
IEC0bits.DMA1IE = 1;          // Enable DMA interrupt
DMA1CONbits.CHEN = 1;         // Enable DMA Channel

DMA1REQbits.FORCE = 1;        // Force First word after Enabling SPI
```

DMA 割り込みハンドラの設定 :

```
void __attribute__((__interrupt__)) _DMA1Interrupt(void)
{
   static unsigned int BufferCount = 0;  // Keep record of which buffer
                                         // contains RX Data

   if(BufferCount == 0)
   {
    ProcessRxData(BufferA);              // Process received SPI data in
                                         // DMA RAM Primary buffer
   }
   else
   {
    ProcessRxData(BufferB);              // Process received SPI data in
                                         // DMA RAM Secondary buffer
   }

   BufferCount ^= 1;
   IFS0bits.DMA1IF = 0;                  // Clear the DMA1 Interrupt Flag
}
```

22.7 DMA 転送の開始

DMA 転送を開始する前に、DMAxCON レジスタの CHEN ビットを「1」にセットして DMA チャンネルを有効にする必要があります。DMA チャンネルがアクティブな状態で、このチャンネルを再度有効にして (CHEN = 1) から無効にする (CHEN = 0) と、再初期化できます。これにより DMA 転送カウントはゼロにリセットされ、アクティブな DMA バッファはプライマリバッファに設定されます。

DMA チャンネルと周辺モジュールが正しく初期化され、周辺モジュールがデータを移動する準備が整うと DMA 要求を発行し、DMA 転送が開始します。しかし、一部の周辺モジュールは、一定の条件が揃うまで DMA 要求を発行しません (従って DMA 転送は開始されません)。このような場合、DMA 転送を開始するには各種の DMA モードと手順を組み合わせて適用する必要があります。

22.7.1 シリアル ペリフェラル インターフェイス (SPI) を使用した DMA の開始

SPI 周辺モジュールとの間で DMA 転送を開始する際には、SPI のデータ方向とスレーブ / マスタモードによって以下の設定があります。

- **マスタモードで TX のみ –** この設定の場合、最初の SPI データブロックを送信するまで DMA 要求は発行されません。DMA 転送を開始するには、ユーザ アプリケーションが DMA 手動転送モードを使用して最初にデータを送信するか、DMA とは別に SPI バッファ (SPIxBUF) に最初にデータを書き込む必要があります。
- **マスタモードで RX のみ –** この設定の場合、最初の SPI データブロックを受信するまで DMA 要求は発行されません。しかし、マスタモードでは SPI が最初に送信するまでデータは受信されません。DMA 転送を開始するには、ユーザ アプリケーションが DMA NULL データ書き込みモードを使用して DMA 手動転送モードを開始する必要があります。
- **マスタモードで RX および TX –** この設定の場合、最初の SPI データブロックを受信するまで DMA 要求は発行されません。しかし、マスタモードでは SPI が送信するまでデータは受信されません。DMA 転送を開始するには、ユーザ アプリケーションが DMA 手動転送モードを使用して最初にデータを送信するか、DMA とは別に SPI バッファ (SPIxBUF) に最初にデータを書き込む必要があります。
- **スレーブ モードで TX のみ –** この設定の場合、最初の SPI データブロックを受信するまで DMA 要求は発行されません。DMA 転送を開始するには、ユーザ アプリケーションが DMA 手動転送モードを使用して最初にデータを送信するか、DMA とは別に SPI バッファ (SPIxBUF) に最初にデータを書き込む必要があります。
- **スレーブ モードで RX のみ –** この設定の場合、最初の SPI データを受信するとすぐに DMA 要求を生成するため、DMA 転送を開始するためにユーザが特別な手順を実施する必要はありません。
- **スレーブモードで RX および TX –** この設定の場合、最初の SPI データブロックを受信するまで DMA 要求は発行されません。DMA 転送を開始するには、ユーザ アプリケーションが DMA 手動転送モードを使用して最初にデータを送信するか、DMA とは別に SPI バッファ (SPIxBUF) に最初にデータを書き込む必要があります。

22.7.2 データコンバータ インターフェイス (DCI) を使用する DMA の開始

他のシリアル ペリフェラルとは異なり、DCI は有効になるとすぐに転送を開始します (DCI がマスタの場合)。DCI は接続されている外部コーデックに同期データフレームを継続的に供給します。DCI を有効にする前に以下を実行する必要があります。

- **22.5.2「周辺モジュールの設定」**に示す手順で DCI を設定する。
- ステレオ コーデックに接続されている場合、DMA 手動転送モードを使用して最初の 2 つのデータ転送を開始する。
  - DMAxREQ レジスタの FORCE ビットをセットして DCI 左チャンネル サンプルを転送する。
  - 再度 FORCE ビットをセットして DCI 右チャンネル サンプルを転送する。

これらの手順が完了した後、DCI 周辺モジュールを有効にします (例 22-11 参照)。

22.7.3 UART を使用する DMA の開始

UART レシーバはデータを受信するとすぐに DMA 要求を発行します。DMA 転送を開始するためにユーザ アプリケーションで特別な手順を実施する必要はありません。UART トランスミッタは UART とトランスミッタが有効になるとすぐに DMA 要求を発行します。つまり、DMA チャンネルとバッファはUARTとトランスミッタより前に初期化され有効になっている必要があります。

UART は、**22.5.2「周辺モジュールの設定」**(表 22-2) に示すように設定する必要があります。

あるいは、UART と UART トランスミッタを DMA チャンネルが有効になる前に有効にする事もできます。この場合、UART トランスミッタの DMA 要求は失われます。ユーザ アプリケーションは DMAxREQ レジスタの FORCE ビットをセットする事によって、DMA 転送を開始する DMA 要求を発行する必要があります。

22.8 DMA チャンネル アービトレーションとオーバーラン

各 DMA チャンネルには固定の優先度があります。チャンネル 0 の優先度は最高で、チャンネル 7 の優先度は最低です。ソースが DMA 転送を要求すると、対応する DMA チャンネルで要求がラッチされます。DMA コントローラはアービトレーターとして機能します。他に実行中または保留状態の転送がない場合、コントローラは要求中の DMA チャンネルにバス リソースを許可します。DMA コントローラは、現在の DMA チャンネルが動作を完了するまで、他の DMA チャンネルにリソースを許可しないようにします。

複数の DMA 要求が到達するか保留状態にある場合、DMA コントローラ内の優先度ロジックは、最も優先度の高い DMA チャンネルが動作を完了できるようリソースを許可します。他の全ての DMA 要求は、選択した DMA 転送が完了するまで保留状態を維持します。DMA 転送中に別の DMA 要求が到達すると、保留中の DMA 要求と共に優先順序が付けられ、DMA 転送完了後に最も優先度が高い要求が実行されます。

DMA チャンネルが優先付けされているため、DMA 要求がただちに実行されず保留状態になる可能性があります。その要求は、より優先度が高いチャンネルの要求が実行され、完了するまで保留のままです。DMA コントローラが元の DMA 要求をクリアする前に別の割り込みを受信し、その割り込みが保留中の割り込みと同じタイプの場合、データ オーバーランが発生します。

データ オーバーランは、DMA が前のデータを移動する前に周辺モジュールのデータバッファに新しいデータが到達した状態と定義されます。DMA 対応周辺モジュールの中には、データ オーバーランを検出して CPU 割り込みを生成できるものもあります (対応する周辺モジュール エラー割り込みが有効な場合) (表 22-5 参照)。

表 22-5: DMA 対応周辺モジュールによるオーバーラン処理

DMA 対応周辺モジュール	データ オーバーランの処理
SPI (シリアル ペリフェラル インターフェイス)	DMA チャンネルによる移動待機中のデータは、追加の入力データによって上書きされません。後続の入力データは失われ、SPI ステータス (SPIxSTAT) レジスタの SPI 受信オーバーフロー (SPIROV) ビットがセットされます。また、割り込みコントローラで割り込みイネーブル制御 (IECx) レジスタの SPI エラー割り込みイネーブル (SPIxEIE) ビットがセットされている場合、SPIx フォルト割り込みが生成されます。
UART	DMA チャンネルによる移動待機中のデータは、追加の入力データによって上書きされません。後続の入力データは失われ、UART ステータス (UxSTA) レジスタのオーバーフロー エラー (OERR) ビットがセットされます。また、割り込みコントローラで割り込みイネーブル制御 (IECx) レジスタの UART エラー割り込みイネーブル (UxEIE) ビットがセットされている場合、UARTx エラー割り込みが生成されます。
データコンバータ インターフェイス (DCI)	DMA チャンネルによる移動待機中のデータは、追加の入力データによって上書きされ、DCI ステータス (DCISTAT) レジスタの受信オーバーフロー (ROV) ビットがセットされます。また、割り込みコントローラで割り込みイネーブル制御 (IEC0) レジスタの DCI エラー割り込みイネーブル (DCIEIE) ビットがセットされている場合、DCI フォルト割り込みが生成されます。
10/12 ビットの A/D コンバータ (ADC)	DMA チャンネルによる移動待機中のデータは、追加の入力データによって上書きされます。オーバーラン状態は ADC で検出されません。
他の DMA 対応周辺モジュール	データオーバーランは発生しません。

データ オーバーランは、DMA が周辺モジュールから DPSRAM にデータを移動する場合のみハードウェアで検出可能です。DPSRAM から周辺モジュールへの DMA データ転送 (例えばバッファ エンプティ割り込みに基づく) は、常に実行されます。重大な DPSRAM データ オーバーランは、ソフトウェアで検出する必要があります。重複する DMA 要求は無視され、保留中の要求は保留状態を維持します。通常通り、DMA チャンネルは転送完了時に DMA 要求をクリアします。CPU が途中で介入しない場合、転送されたデータが最新の (オーバーラン) データとなり、それ以前のデータは失われます。

ユーザ アプリケーションは、データ ソースの性質に応じて複数の方法でオーバーラン エラーを処理できます。DMAC のデータ リカバリとそのデータ ソース / シンクとの同期再確立は、アプリケーションに大きく依存するタスクです。ストリーミング データ (DCI 周辺モジュール経由のコーデックからのデータ等) の場合、アプリケーションは失われたデータを無視できます。問題の原因を修正した後 (可能な場合)、DMA 割り込みハンドラはデータが再度正しくバッファされるように DMAC と DCI の同期再確立を試みます。ユーザ アプリケーションは、さらにオーバーランが発生するのを防ぐため迅速に対応する必要があります。

周辺モジュール オーバーラン割り込みに移行するまでに、保留状態の DMA 要求は、失われたデータが移動されるはずであったアドレスにオーバーラン データ値の移動を完了しています。そのデータを正しいアドレスに移動し、欠損データ スロットに NULL データ値を挿入する事ができます。その後、チャンネルの DPSRAM アドレスを適切に調整できます。エラーのあったチャンネルに対する後続の DMA 要求は、修正済み DPSRAM アドレスに対して通常通り転送を開始します。データ損失を許容できないアプリケーションの場合、周辺モジュール オーバーラン割り込みによってデータが失われる前に現在のブロック転送を中止し、DMA チャンネルを再初期化し、データの再送を要求する必要があります。

22.9 デバッグサポート

DMA 動作に関するデバッグが容易なように、DMA コントローラは複数のステータス レジスタを備えています。この情報には、最後に実行された DMA チャンネル (DMACS1 レジスタの LSTCH<3:0> ビット)、アクセスしていた DPSRAM アドレス オフセット (DSADR レジスタの DSADR<15:0> ビット)、アクセスを開始したバッファ (DMACS1 レジスタの PPSTx ビット) が含まれます。

22.10 データ書き込みコリジョン

CPU と DMA チャンネルは、任意の DPSRAM または DMA 対応周辺モジュール データレジスタに対して、同時に読み出しまたは読み書きする事があります。唯一の制約事項は、CPU と DMA チャンネルは同時に同じアドレスに書き込む事ができないという事です。通常の状況では、このような状態が発生する事はありません。しかし、何らかの理由で発生した場合は、検出してフラグを立て、DMA フォルトトラップを起動します。CPU 書き込みを優先させる事もできますが、これは主に予測可能な動作を行うためであり、それ以外にほとんど実用性はありません。

同じバスサイクルで、CPU が読み出し中の位置に DMA チャンネルが書き込む事も許容されます。その逆も同様です。しかし、読み出されるデータは古いデータであり、そのバスサイクル中に書き込まれたデータではないので注意が必要です。また、この状況は通常動作と見なされるため、特別な処置が取られる事は一切ありません。

CPU と DMA チャンネルによる同一 DPSRAM アドレスへの同時書き込みが発生すると、DMA コントローラ ステータス 0 (DMACS0) レジスタの XWCOLx ビットがセットされます。CPU と DMA チャンネルによる同一周辺モジュール アドレスへの同時書き込みが発生すると、DMA コントローラ ステータス 0 (DMACS0) レジスタの PWCOLx ビットがセットされます。全てのコリジョン ステータスフラグは論理 OR で結合され、共通の DMAC フォルト トラップを生成します。ユーザ アプリケーションが割り込みコントローラ (INTCON1) レジスタの DMAC エラー ステータスビット (DMACERR) をクリアすると、XWCOLx フラグと PWCOLx フラグは自動的にクリアされます。

書き込みコリジョンエラーのあるチャンネルへの後続の DMA 要求は、XWCOLx または PWCOLx がセットされている間は無視されます。

書き込みコリジョン条件下で、XWCOLx と PWCOLx のいずれかは書き込みコリジョンによってセットできますが、両方セットする事はできません。両方のフラグをセットするのは、ステータスビットを追加せずにまれな手動トリガ イベントエラーを通知するための特殊な手段として使用します (**22.6.10「手動転送モード」**参照)。

例 22-13 に、DPSRAM から周辺モジュール (UART) にデータを転送する DMA チャンネル 0 と、周辺モジュール (ADC) から DPSRAM にデータを転送する DMA チャンネル 1 を使用する、DMA コントローラのトラップ処理を示します。

例 22-13: DMA コントローラのトラップ処理 :

```
void __attribute__((__interrupt__)) _DMACError(void)
{
    static unsigned int ErrorLocation;

    // Peripheral Write Collision Error Location
    if(DMACS0 & 0x0100)
    {
        ErrorLocation = DMA0STA;
    }

    // DMA RAM Write Collision Error Location
    if(DMACS0 & 0x0002)
    {
        ErrorLocation = DMA1STA;
    }

    DMACS0 = 0;                          //Clear Write Collision Flag
    INTCON1bits.DMACERR = 0;             //Clear Trap Flag
}
```

22.11 省電力モード時の動作

22.11.1 スリープモード

スリープモード時、DMA は無効になります。スリープモードに移行する前に、全ての DMA チャンネルに現在実行中のブロック転送を完了させるか、無効にする事を推奨します (必須ではありません)。

22.11.2 アイドルモード

DMA はシステム内の第 2 のバスマスタである事から、CPU が省電力アイドルモードに移行した後もデータ転送を継続できます。DMA チャンネルを使う周辺モジュールがアイドルモードでも動作するよう設定されている場合、周辺モジュールと DPSRAM の間でデータ転送が可能です。ブロック転送が完了すると、DMA チャンネルは割り込みを生成し (有効な場合)、CPU をウェイクアップします。CPU は続いて割り込みサービス ハンドラを実行します。

周辺モジュールごとにアイドル時停止制御ビットがあります。この制御ビットをセットすると、CPU がアイドルモードにある間、周辺モジュールを無効にします。DMAC を使用して周辺モジュールとの間でデータを転送する場合、その周辺モジュールのアイドル時停止機能を使用すると、実質的にその周辺モジュールに関連付けられた DMA チャンネルも無効になります。

22.12 設計のヒント

22.12.1 DMA と DCI のインターフェイス

DCI でフレームごとに複数のオーディオ チャンネルがある場合 (例 : ステレオ コーデックにおける左右のチャンネル)、全てのサンプルは順番にインターリーブされ DMA チャンネルで転送されます (図 22-21 参照)。

しかし、一般的にユーザ アプリケーションの多くは、チャンネル単位でデータを処理します。すなわち、転送データを並べ換えるか不要データを飛び越すようインデックスを付ける、追加アルゴリズムが必要です。いずれの方法でも追加のコードが必要で、より長い実行時間を要します。

DCI は、周辺モジュール間接アドレッシングをサポートしていません。しかし、2 つの DMA チャンネルを組み合わせた特殊な DCI 設定のオーディオ チャンネルでデータを収集する事は可能です。DCI 制御 2 (DCICON2) レジスタのバッファ長制御ビット (BLEN<1:0>) を (「00」ではなく)「01」にセットし、2 つの DMA チャンネルを使用して DCI から DPSRAM に受信データを転送すると、受信したオーディオ データはチャンネルごとに並び換えられます。この場合、DCI が DMA 要求を生成すると、2 つのワードがバッファされるたびに両方の DMA チャンネルに DMA 要求が送信されます (1 つは右サンプル、もう 1 つは左サンプル)。DMA 要求が発生すると、1 つの DMA チャンネルが DCI RX バッファ 0 (RXBUF0) レジスタからデータを転送し、もう 1 つの DMA チャンネルが DCI RX バッファ 1 (RXBUF1) レジスタからデータを転送します。転送データは実質的にオーディオ チャンネルごとに並び換えられます(図 22-22参照)。

この例が機能するように、DMA チャンネル 1 は DMA1PAD レジスタを (**22.5.1「DMA チャンネルと周辺モジュールの関連付けの設定」**に記載された RXBUF0 アドレスではなく) RXBUF1 アドレスに初期化します。

図 22-21: 代表的な DCI のデータ転送

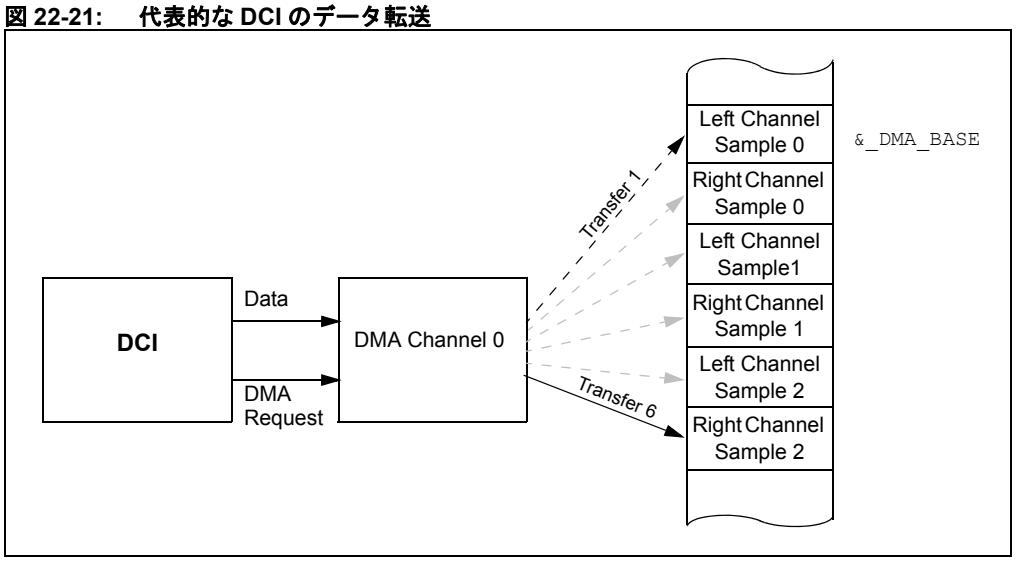
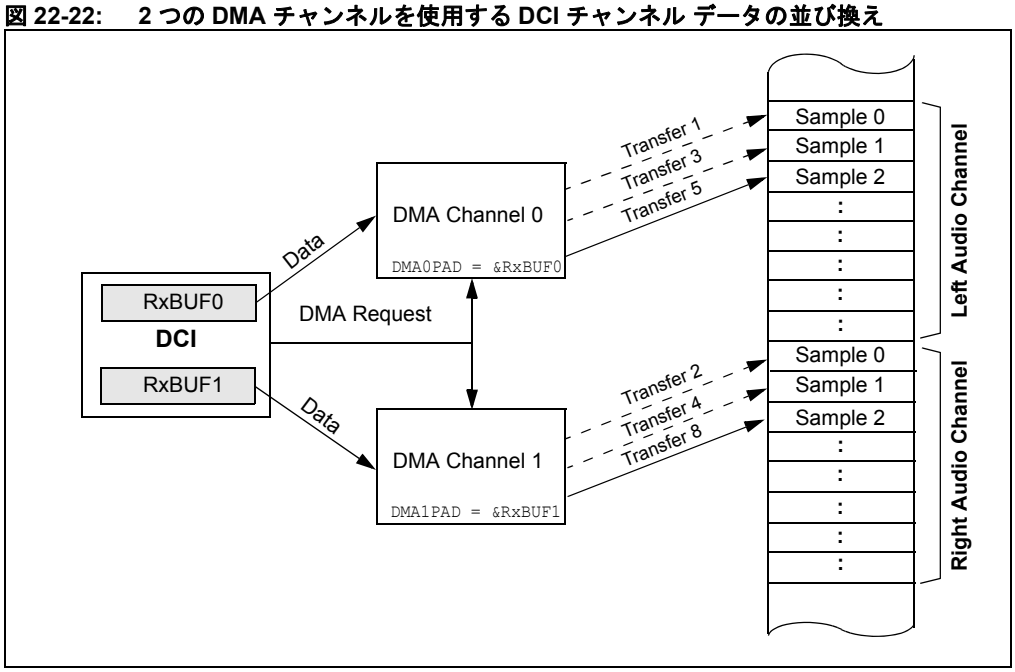


図 22-22:　2 つの DMA チャンネルを使用する DCI チャンネル データの並び換え



22.13 レジスタマップ

表 22-6: DMA レジスタマップ

レジスタ名	アドレス	Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	全リセット
DMA0CON	0380	CHEN	SIZE	DIR	HALF	NULLW	—	—	—	—	—	AMODE<1:0>		—	—	MODE<1:0>		0000
DMA0REQ	0382	FORCE	—	—	—	—	—	—	—	—	IRQSEL<6:0>							0000
DMA0STA	0384	STA<15:0>																0000
DMA0STB	0386	STB<15:0>																0000
DMA0PAD	0388	PAD<15:0>																0000
DMA0CNT	038A	—	—	—	—	—	—	CNT<9:0>										0000
DMA1CON	038C	CHEN	SIZE	DIR	HALF	NULLW	—	—	—	—	—	AMODE<1:0>		—	—	MODE<1:0>		0000
DMA1REQ	038E	FORCE	—	—	—	—	—	—	—	—	IRQSEL<6:0>							0000
DMA1STA	0390	STA<15:0>																0000
DMA1STB	0392	STB<15:0>																0000
DMA1PAD	0394	PAD<15:0>																0000
DMA1CNT	0396	—	—	—	—	—	—	CNT<9:0>										0000
DMA2CON	0398	CHEN	SIZE	DIR	HALF	NULLW	—	—	—	—	—	AMODE<1:0>		—	—	MODE<1:0>		0000
DMA2REQ	039A	FORCE	—	—	—	—	—	—	—	—	IRQSEL<6:0>							0000
DMA2STA	039C	STA<15:0>																0000
DMA2STB	039E	STB<15:0>																0000
DMA2PAD	03A0	PAD<15:0>																0000
DMA2CNT	03A2	—	—	—	—	—	—	CNT<9:0>										0000
DMA3CON	03A4	CHEN	SIZE	DIR	HALF	NULLW	—	—	—	—	—	AMODE<1:0>		—	—	MODE<1:0>		0000
DMA3REQ	03A6	FORCE	—	—	—	—	—	—	—	—	IRQSEL<6:0>							0000
DMA3STA	03A8	STA<15:0>																0000
DMA3STB	03AA	STB<15:0>																0000
DMA3PAD	03AC	PAD<15:0>																0000
DMA3CNT	03AE	—	—	—	—	—	—	CNT<9:0>										0000
DMA4CON	03B0	CHEN	SIZE	DIR	HALF	NULLW	—	—	—	—	—	AMODE<1:0>		—	—	MODE<1:0>		0000
DMA4REQ	03B2	FORCE	—	—	—	—	—	—	—	—	IRQSEL<6:0>							0000
DMA4STA	03B4	STA<15:0>																0000
DMA4STB	03B6	STB<15:0>																0000
DMA4PAD	03B8	PAD<15:0>																0000
DMA4CNT	03BA	—	—	—	—	—	—	CNT<9:0>										0000
DMA5CON	03BC	CHEN	SIZE	DIR	HALF	NULLW	—	—	—	—	—	AMODE<1:0>		—	—	MODE<1:0>		0000
DMA5REQ	03BE	FORCE	—	—	—	—	—	—	—	—	IRQSEL<6:0>							0000

凡例： — = 未実装、「0」として読み出し、リセット時の値を 16 進数で表示

表 22-6: DMA レジスタマップ (続き)

レジスタ名	アドレス	Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	全リセット
DMA5STA	03C0	STA<15:0>																0000
DMA5STB	03C2	STB<15:0>																0000
DMA5PAD	03C4	PAD<15:0>																0000
DMA5CNT	03C6	—	—	—	—	—	—	CNT<9:0>										0000
DMA6CON	03C8	CHEN	SIZE	DIR	HALF	NULLW	—	—	—	—	—	AMODE<1:0>		—	—	MODE<1:0>		0000
DMA6REQ	03CA	FORCE	—	—	—	—	—	—	—	—	IRQSEL<6:0>							0000
DMA6STA	03CC	STA<15:0>																0000
DMA6STB	03CE	STB<15:0>																0000
DMA6PAD	03D0	PAD<15:0>																0000
DMA6CNT	03D2	—	—	—	—	—	—	CNT<9:0>										0000
DMA7CON	03D4	CHEN	SIZE	DIR	HALF	NULLW	—	—	—	—	—	AMODE<1:0>		—	—	MODE<1:0>		0000
DMA7REQ	03D6	FORCE	—	—	—	—	—	—	—	—	IRQSEL<6:0>							0000
DMA7STA	03D8	STA<15:0>																0000
DMA7STB	03DA	STB<15:0>																0000
DMA7PAD	03DC	PAD<15:0>																0000
DMA7CNT	03DE	—	—	—	—	—	—	CNT<9:0>										0000
DMACS0	03E0	PWCOL7	PWCOL6	PWCOL5	PWCOL4	PWCOL3	PWCOL2	PWCOL1	PWCOL0	XWCOL7	XWCOL6	XWCOL5	XWCOL4	XWCOL3	XWCOL2	XWCOL1	XWCOL0	0000
DMACS1	03E2	—	—	—	—	LSTCH<3:0>				PPST7	PPST6	PPST5	PPST4	PPST3	PPST2	PPST1	PPST0	0000
DSADR	03E4	DSADR<15:0>																0000
INTCON1	0080	NSTDIS	—	—	—	—	—	—	—	—	—	DMACERR	—	—	—	—	—	0000
IFS0	0084	—	DMA1IF	—	—	—	—	—	—	—	—	—	DMA0IF	—	—	—	—	0000
IFS1	0086	—	—	—	—	—	—	—	DMA2IF	—	—	—	—	—	—	—	—	0000
IFS2	0088	—	DMA4IF	—	—	—	—	—	—	—	—	—	DMA3IF	—	—	—	—	0000
IFS3	008A	—	—	DMA5IF	—	—	—	—	—	—	—	—	—	—	—	—	—	0000
IFS4	008C	—	—	—	—	—	—	—	—	—	—	DMA7IF	DMA6IF	—	—	—	—	0000
IEC0	0094	—	DMA1IE	—	—	—	—	—	—	—	—	—	DMA0IE	—	—	—	—	0000
IEC1	0096	—	—	—	—	—	—	—	DMA2IE	—	—	—	—	—	—	—	—	0000
IEC2	0098	—	DMA4IE	—	—	—	—	—	—	—	—	—	DMA3IE	—	—	—	—	0000
IEC3	009A	—	—	DMA5IE	—	—	—	—	—	—	—	—	—	—	—	—	—	0000
IEC4	009C	—	—	—	—	—	—	—	—	—	—	DMA7IE	DMA6IE	—	—	—	—	0000
IPC1	00A6	—	—	—	—	—	—	—	—	—	—	—	—	—	DMA0IP<2:0>			4444
IPC3	00AA	—	—	—	—	—	DMA1IP<2:0>			—	—	—	—	—	—	—	—	4444
IPC6	00B0	—	—	—	—	—	—	—	—	—	—	—	—	—	DMA2IP<2:0>			4444
IPC9	00B6	—	—	—	—	—	—	—	—	—	—	—	—	—	DMA3IP<2:0>			4444

凡例 : — = 未実装、「0」として読み出し、リセット時の値を 16 進数で表示

表 22-6: DMA レジスタマップ (続き)

レジスタ名	アドレス	Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	全リセット
IPC11	00BA	—	—	—	—	—	DMA4IP<2:0>			—	—	—	—	—	—	—	—	4444
IPC15	00C2	—	—	—	—	—	—	—	—	—	DMA5IP<2:0>			—	—	—	—	4444
IPC17	00C6	—	—	—	—	—	—	—	—	—	DMA7IP<2:0>			—	DMA6IP<2:0>			4444

凡例 : — = 未実装、「0」として読み出し、リセット時の値を 16 進数で表示

22.14 関連アプリケーション ノート

ダイレクト メモリ アクセス (DMA) の使用に関連するアプリケーション ノートの一覧を以下に示します。一部のアプリケーション ノートは dsPIC33F 製品ファミリ向けではありません。ただし概念は共通しており、変更が必要であったり制限事項が存在するものの利用が可能です。DMA モジュールに関連する現在のアプリケーション ノートは以下の通りです。

タイトル	アプリケーション ノート番号
現在関連するアプリケーション ノートはありません。	

Note: PIC33F ファミリ関連のアプリケーション ノートとサンプルコードは弊社ウェブサイト (www.microchip.com) をご覧ください。

22.15 改訂履歴

リビジョン A (2006 年 12 月)

本書の初版

リビジョン B (2008 年 7 月)

改訂内容 :

- 図 :
  - レジスタ間接アドレッシングを使用する ADC からのデータ転送 (図 22-11 参照)
  - 周辺モジュール間接アドレッシングを使用する ADC からのデータ転送 (図 22-12 参照)
- 上記に加えて、表現および体裁の変更等、本書全体の細部を修正

ISBN: 978-1-60932-871-9