ADSP-BF533 Blackfin® プロセッサ・ハードウェア・リファレンス

改訂版 3.0、2004 年 9 月

製品番号
82-002005-01

Analog Devices, Inc.
One Technology Way
Norwood, Mass. 02062-9106

著作権情報

Printed in Japan.

免責条項

Analog Devices, Inc. は、予告なく、この製品を変更する権利を保有します。Analog Devices から提供する情報の正確性と信頼性には万全を尽くしています。しかし、Analog Devices は、その使用に対する責任を一切負いません。その使用によって第三者の特許権やその他の権利が侵害された場合でも、同様に責任を負いません。Analog Devices, Inc. の特許権のもとでは、暗示的にも明示的にも、いかなるライセンスも提供しません。

商標と登録商標の通知

Analog Devices のロゴ、Blackfin および Blackfin のロゴ、CROSSCORE、SHARC、TigerSHARC、VisualDSP++ は、Analog Devices, Inc. の登録商標です。

EZ-KIT Lite は、Analog Devices, Inc. の商標です。

その他すべての商標名と製品名は、それぞれの所有者の商標または登録商標です。

目次

まえがき

演算ユニット

動作モードと状態

プログラム・シーケンサ

チップ・バス階層

SPI 互換ポート・コントローラ

シリアル・ポート・コントローラ

UART ポート・コントローラ

プログラマブル・フラグ

システム設計

数値フォーマット

用語集

まえがき

アナログ・デバイセズのBlackfin®プロセッサでシステムを開発していただき、まことにありがとうございます。

マニュアルの目的

このマニュアルでは、Blackfinプロセッサのアーキテクチャ情報を提供します。アーキテクャ説明では、機能ブロック、バス、ポートに加えて、サポートされるすべての機能とプロセスも取り上げます。プログラミング情報については、『ADSP-BF53x Blackfin Processor Instruction Set Reference』を参照してください。タイミング仕様、電気仕様、パッケージ仕様については、『ADSP-BF531/ADSP-BF532/ADSP-BF533 Embedded Processor Data Sheet』を参照してください。

対象となる読者

このマニュアルは、アナログ・デバイセズのプロセッサに精通しているプログラマを対象にしており、該当するプロセッサ・アーキテクチャと命令セットに関する実用的な知識がある読者を想定しています。アナログ・デバイセズのプロセッサをよく知らないプログラマもこのマニュアルを使用できますが、その場合はターゲット・アーキテクチャについて解説している他の資料（該当するハードウェア・リファレンス・マニュアルやデータシートなど）も併用してください。

マニュアルの内容

このマニュアルは、以下の章で構成されています。

- 第1章「はじめに」

  プロセッサのハイレベルな概要を示します。アーキテクャ説明では、機能ブロック、バス、ポートに加えて、サポートされる機能とプロセスも取り上げます。

- 第2章「演算ユニット」

  算術論理演算ユニット (ALU)、積和演算ユニット (MAC)、シフタ、ビデオALU群について説明します。また、データ・フォーマット、データ型、レジスタ・ファイルについても説明します。

- 第3章「動作モードと状態」

  プロセッサの3つの動作モードである、エミュレーション・モード、スーパーバイザ・モード、ユーザ・モードについて説明します。また、アイドル状態とリセット状態についても説明します。

- 第4章「プログラム・シーケンサ」

  実行される次の命令のアドレスを提供してプログラム・フローを制御する、プログラム・シーケンサの動作について説明します。また、ループ、サブルーチン、ジャンプ、割込み、例外についても説明します。

- 第5章「データ・アドレス・ジェネレータ」

  データ・アドレス・ジェネレータ (DAG)、アドレッシング・モード、DAGレジスタとポインタ・レジスタの変更方法、メモリのアドレス・アライメント、DAG命令について説明します。

- 第6章「メモリ」

  L1メモリについて説明します。特に、そのメモリ・アーキテクチャ、メモリ・モデル、メモリ・トランザクション・モデル、メモリマップド・レジスタ (MMR) について詳述します。また、命令、データ、およびBlackfinプロセッサ・コアの一部であるスクラッチパッド・メモリについても説明します。

- 第7章「チップ・バス階層」

  システム内でのデータ転送も含めて、オンチップ・バスについて説明します。また、システム・メモリ・マップ、主要なシステム・コンポーネント、システム相互接続についても説明します。

- 第8章「ダイナミック・パワー・マネジメント」

  システムのリセット設定とパワーアップ設定、システムのクロッキングと制御、パワーマネジメントについて説明します。

- 第9章「ダイレクト・メモリ・アクセス」

  ペリフェラルDMAコントローラとメモリDMAコントローラについて説明します。ペリフェラルDMAの節では、DMAアクセスを備えたペリフェラルと内部／外部メモリ・スペースとの間の直接的なブロック・データ転送について説明します。

  メモリDMAの節では、プロセッサのメモリ・スペースとL1メモリ、外部の同期／非同期メモリの間でのメモリ間転送機能について説明します。

- 第10章「SPI互換ポート・コントローラ」

  SPI互換のさまざまなペリフェラル・デバイスにI/Oインターフェースを提供する、シリアル・ペリフェラル・インターフェース(SPI) ポートについて説明します。

- 第11章「パラレル・ペリフェラル・インターフェース」

  プロセッサのパラレル・ペリフェラル・インターフェース (PPI) について説明します。PPIは半二重の双方向ポートであり、最高16ビットのデータに対応し、デジタル・ビデオやデータ・コンバータのアプリケーションに使用されます。

- 第12章「シリアル・ポート・コントローラ」

  さまざまなシリアル・ペリフェラル・デバイスにI/Oインターフェースを提供する、2つの独立した同期シリアル・ポート・コントローラ（SPORT0とSPORT1）について説明します。

- 第13章「UARTポート・コントローラ」

  データ・フォーマットのシリアル／パラレル変換を行い、モデム制御と割込み処理のハードウェアを内蔵する、非同期シリアル・インターフェースLSI (UART) ポートについて説明します。UARTでは、モード対応機能として半二重のIrDA® SIRプロトコルをサポートします。

- 第14章「プログラマブル・フラグ」

  ピンを入／出力として設定する方法や割込みの生成方法も含めて、プロセッサ内のプログラマブル・フラグ、つまり汎用I/Oピンについて説明します。

- 第15章「タイマ」

  3つのモードに設定できる、3つの汎用タイマについて説明します。コア・タイマでは、さまざまなタイミング機能用に周期割込みを生成できます。ウォッチドッグ・タイマでは、Blackfinプロセッサ・コアへのイベント生成などのソフトウェア・ウォッチドッグ機能を実装できます。

- 第16章「リアルタイム・クロック（RTC）」

  時刻、アラーム、ストップウォッチ・カウントダウンなど、プロセッサの一連のデジタル・ウォッチ機能について説明します。

- 第17章「外部バス・インターフェース・ユニット」

  プロセッサの外部バス・インターフェース・ユニットについて説明します。また、非同期メモリ・インターフェース、SDRAMコントローラ (SDC)、関連するレジスタ、SDCの設定、コマンドについても説明します。

- 第18章「システム設計」

  プロセッサをシステム全体の一部として使用する方法について説明します。プロセッサを外部メモリ・チップにインターフェースする方法、バスのタイミング数とレイテンシ数、セマフォ、未使用ピンの取り扱いについても説明します。

- 第19章「Blackfinプロセッサのデバッグ」

  Blackfinプロセッサのデバッグ機能について説明します。この機能はソフトウェア・デバッギングに使用でき、オペレーティング・システムでよく採用されているサービスを補完します。

- 付録A「Blackfinプロセッサ・コアMMRの割当て」

  コア・メモリマップド・レジスタ、そのアドレス、本文へのクロスリファレンスを一覧にします。

- 付録B「システムMMRの割当て」

  システム・メモリマップド・レジスタ、そのアドレス、本文へのクロスリファレンスを一覧にします。

- 付録C「テスト機能」

  プロセッサのテスト機能について説明します。JTAG規格、バウンダリスキャン・アーキテクチャ、命令レジスタとバウンダリ・レジスタ、パブリック命令について説明します。

- 付録D「数値フォーマット」

  16ビット・データ・フォーマットのさまざまな側面について説明します。また、ソフトウェアでブロック浮動小数点フォーマットを実装する方法についても説明します。

- 付録G「用語集」

  略語を含め、この本で使用されている用語の定義があります。

マニュアルの最新情報

これは『ADSP-BF533 Blackfinプロセッサ・ハードウェア・リファレンス』の第3版です。この本の第2版からの改訂には、誤植や報告された正誤表の訂正が含まれます。

テクニカル／カスタマ・サポート

Blackfinプロセッサのカスタマ・サポートは、以下の方法でご利用になれます。

- Blackfinプロセッサのwebサイト：
  www.analog.com/jp/blackfin
- 電子メールでのお問い合わせ：marcom.japan@analog.com
- アナログ・デバイセズの正規販売代理店：
  www.analog.com/intl/japan/salesdir/index.html

対応するプロセッサ

以下は、VisualDSP++®に対応しているアナログ・デバイセズのプロセッサのリストです。

TigerSHARC®（ADSP-TSxxx）プロセッサ

TigerSHARCは、8/16/32ビットの浮動小数点／固定小数点のプロセッサです。VisualDSP++は、以下のTigerSHARCプロセッサで利用できます。

ADSP-TS101 、ADSP-TS201、ADSP-TS202、ADSP-TS203

SHARC®（ADSP-21xxx）プロセッサ

SHARCは、音声、サウンド、グラフィック、イメージング・アプリケーションに使用できる高性能、32ビット、浮動小数点のプロセッサです。VisualDSP++は、以下のSHARCプロセッサで利用できます。

ADSP-21020、ADSP-21060、ADSP-21061、ADSP-21062、ADSP-21065L、ADSP-21160、ADSP-21161、ADSP-21261、ADSP-21262、ADSP-21266、ADSP-21267、ADSP-21363、ADSP-21364、ADSP-21365

Blackfin®（ADSP-BFxxx）プロセッサ

Blackfinは、16ビットの組込みプロセッサです。
VisualDSP++は、以下のBlackfinプロセッサで利用できます。

ADSP-BF531、ADSP-BF532 （旧ADSP-21532）、ADSP-BF533、
ADSP-BF535（旧ADSP-21535）、ADSP-BF561、AD6532、AD90747

製品情報

製品情報については、アナログ・デバイセズWebサイト、製品のCD-ROM、または印刷刊行物（マニュアル）をご覧ください。

アナログ・デバイセズのWebサイトhttp://www.analog.com/jpでは、アナログ集積回路、アンプ、コンバータ、デジタル・シグナル・プロセッサなど、広範囲の製品情報を提供します。

■ MyAnalog.com

MyAnalog.comは、アナログ・デバイセズのWebサイトが提供する無料の機能です。関心のある製品の最新情報だけをWebページに表示するようにカスタマイズできます。また、ご自分の興味に一致するWebページの更新情報に関してお知らせメールを毎週受け取ることもできます。MyAnalog.comから書籍、アプリケーション・ノート、データシート、コード例などを入手することも可能です。

登録

www.myanalog.com でご登録ください。「Register」をクリックすれば、MyAnalog.com.をご利用になれます。登録には約5分かかります。登録することによって、欲しい情報を選択することができます。

すでに登録されているユーザは、ログインしてください。ご使用のユーザ名がメールのアドレスになります。

■ プロセッサ製品情報

組込みプロセッサとDSPの詳細については、当社のWebサイト http://www.analog.com/jp/dsp をご覧ください。技術刊行物、データシート、アプリケーション・ノート、製品概要、製品発表資料にアクセスできます。

■ 関連資料

ADSP-BF53xプロセッサ（および関連プロセッサ）に関する以下の資料は、アナログ・デバイセズの販売代理店にご注文ください。

- VisualDSP++ User's Guide for 16-Bit Processors
- VisualDSP++ C/C++ Compiler and Library Manual for Blackfin Processors
- VisualDSP++ Assembler and Preprocessor Manual for Blackfin Processors
- VisualDSP++ Linker and Utilities Manual for 16-Bit Processors
- VisualDSP++ Kernel (VDK) User's Guide for 16-Bit Processor
- VisualDSP++ Component Software Engineering User's Guide for 16-Bit Processors
- ADSP-BF531/ADSP-BF532/ADSP-BF533 Embedded Processor Data Sheet
- ADSP-BF535 Embedded Processor Data Sheet
- ADSP-BF53x Blackfin Processor Instruction Set Reference

すべてのプロセッサおよびツールのマニュアルとデータシートをご所望の場合は、テクニカル・ライブラリのWebサイトをご覧ください。

`http://www.analog.com/processors/Japan/resources/technicalLibrary`

■ オンライン技術資料

オンライン資料には、VisualDSP++Helpシステム、ソフトウェア・ツール・マニュアル、ハードウェア・ツール・マニュアル、プロセッサ・マニュアル、Dinkum Abridged C++ライブラリ、Flexible License Manager（FlexLM）ネットワーク・ライセンス・マネージャ・ソフトウェア・ドキュメンテーションがあります。関心のある問題について、VisualDSP++資料全体の中から簡単に検索できます。大部分のマニュアルを.PDFファイルで提供しています。

資料のファイルには、以下の種類があります。

ファイル	内容
.CHM	ヘルプ・システムのファイルおよびヘルプ形式のマニュアル。
.HTM または .HTML	Dinkum Abridged C++ ライブラリ、Flexible License Manager（FlexLM）ネットワーク・ライセンス・マネージャ・ソフトウェア・ドキュメンテーション。HTML ファイルでの表示や印刷には、Internet Explorer 4.0（またはそれ以上）のブラウザが必要です。
.PDF	ポータブル・ドキュメンテーション・フォーマット（PDF）の VisualDSP++ およびプロセッサのマニュアル。.PDF ファイルの表示や印刷には Adobe Acrobat Reader（4.0 以上）などの PDF リーダーが必要です。

ソフトウェアとともに資料をシステムにインストールしなかった場合は、VisualDSP++のCD-ROMからToolsのインストールを実行すればいつでもインストールできます。オンラインの資料は、VisualDSP++環境、Windows® Explorer、またはアナログ・デバイセズのWebサイトから入手できます。

▶ VisualDSP++ 環境からの資料の入手

以下の手順で、VisualDSP++環境から入手できます。

- ヘルプ・メニューの Contents、Search、Index から VisualDSP++ オンライン・ヘルプにアクセスします。
- コンテキスト・センシティブのユーザ・インターフェース項目からオンライン・ヘルプを開きます。

▶ Windows からの資料の入手

ユーザの設定したショートカットのほかに、VisualDSP++オンライン・ヘルプやWindowsから補足資料を開くにはさまざまな方法があります。

ヘルプ・システムのファイル（.CHM）はHelpフォルダの中に、.PDFファイルはVisualDSP++インストールCD-ROMのDocsフォルダの中にあります。Docsフォルダには、Dinkum Abridged C++ライブラリとFlexLMネットワーク・ライセンス・マネージャ・ソフトウェア・ドキュメンテーションも入っています。

Windows Explorerを使用する場合

- vdsp-help.chmファイル（マスターのヘルプ・システム）をダブルクリックして、すべての.CHMファイルにアクセスします。
- VisualDSP++ 資料セットの中の任意のファイルをダブルクリックします。

Windowsのスタート・ボタンを使用する場合

- **スタート**・ボタンから**プログラム**、Analog Devices、VisualDSP++、VisualDSP++ Documentationを選択して、VisualDSP++オンライン・ヘルプにアクセスします。
- **スタート**・ボタンから**プログラム**、Analog Devices、VisualDSP++、Documentation for Printing、書名を選択して、.PDFファイルを入手します。

▶ Web サイトからの資料の入手

以下のWebサイトからマニュアルをダウンロードします。
http://www.analog.com/processors/Japan/resources/technicalLibrary

プロセッサ・ファミリーや書名を選択します。マニュアルごとのアーカイブ（.ZIP）ファイルをダウンロードします。WinZipなど任意のアーカイブ管理ソフトウェアを使用して、ダウンロードしたファイルを解凍します。

■ 印刷物

資料のご請求に関しましては、**フリーダイアル** 0120-390769（**サンキューアナログ**）までお問い合わせください。

▶ VisualDSP++ マニュアル

VisualDSP++マニュアルは、アナログ・デバイセズのWebサイトよりダウンロードください。印刷物をご希望の場合は、併設のフリーダイアルまでお問い合わせください。

▶ ハードウェア・ツール・マニュアル

EX-KIT Lite™およびIn-Circuit Emulator（ICE）のマニュアルをご希望の場合は、併設のフリーダイアルまでお問い合わせください。

▶ プロセッサ・マニュアル

ハードウェア・リファレンス・マニュアルと命令セット・リファレンス・マニュアルは、アナログ・デバイセズのWebサイトよりダウンロードください。印刷物をご希望の場合は、併設のフリーダイアルまでお問い合わせください。

▶ データシート

すべてのデータシートは、アナログ・デバイセズのWebサイトよりダウンロードが可能です。印刷物をご希望の場合は、併設のフリーダイアルまでお問い合わせください。

▶ DSP 刊行物へのご意見

当社のマニュアルおよびオンライン・ヘルプについてのご意見・ご提案は、下記アドレスにお寄せください。

marcom.japan@analog.com

表記規則

このマニュアルで使用するテキスト表記規則は、以下のとおりです。

例	説明
SWRSTソフトウェア・リセット・レジスタ	レジスタ名は、**大文字**と特殊な字体で表記されます。レジスタの記述名は、大／小文字を混じえて通常字体で表記されます。
TMROE, $\overline{\text{RESET}}$	ピン名は、**大文字**と特殊な字体で表記されます。アクティブ・ロー信号は**上線**と共に表記されます。

例	説明
DRx, I[3:0] $\overline{\text{SMS[3:0]}}$	本文中のレジスタ名、ビット名、ピン名は、レジスタやピンのグループを表すことがあります。 レジスタ名 (DRx) の小文字 x は、一連のレジスタを示します (たとえば、DR2、DR1、DR0)。 大カッコ内で数値を区切るコロンは、一連のレジスタやピンを示します (たとえば、I[3:0] では I3、I2、I1、I0 を示し、$\overline{\text{SMS[3:0]}}$ では $\overline{\text{SMS3}}$、$\overline{\text{SMS2}}$、$\overline{\text{SMS1}}$、$\overline{\text{SMS0}}$ を示します)。
0xabcd, b#1111	接頭辞 0x は 16 進を示し、接頭辞 b# は 2 進を示します。
注：	注では、特に重要な情報や関連する問題を示します。
注意：	注意では、プロセッサの動作に影響を与える設計上またはプログラミング上の重要な問題について情報を示します。

本書の中では、特定の章の中でのみ、その他の表記規則を使用していることがあります。

■ レジスタ図の表記規則

レジスタ図では、以下の表記規則を使用します。

- レジスタの記述名が先頭に表記され、それに続くカッコ内に名前の短形式が表記されます。

表 P-1.　レジスタ名の短形式

パターン	説明	例
TIMERx_CONFIG	x は複数のペリフェラルを表します。	TIMER0_CONFIG TIMER1_CONFIG TIMER2_CONFIG
SIC_IARn	n は、同じペリフェラル内または同じコア・コンポーネント内の複数のレジスタを表します。	SIC_IAR2 ICPLB_DATA15

表 P-1.　レジスタ名の短形式

パターン	説明	例
SPORTx_TCRn	x と n の組合わせは、複数のペリフェラルと、同じペリフェラル内の複数のレジスタを示します。	SPORT0_TCR0 SPORT1_TCR1
MDMA_yy_CONFIG	yy は、デスティネーションまたはソースの、MemDMA ストリーム 0 または 1 を表します。	MDMA_D0_CONFIG MDMA_S0_CONFIG MDMA_D1_CONFIG MDMA_S1_CONFIG

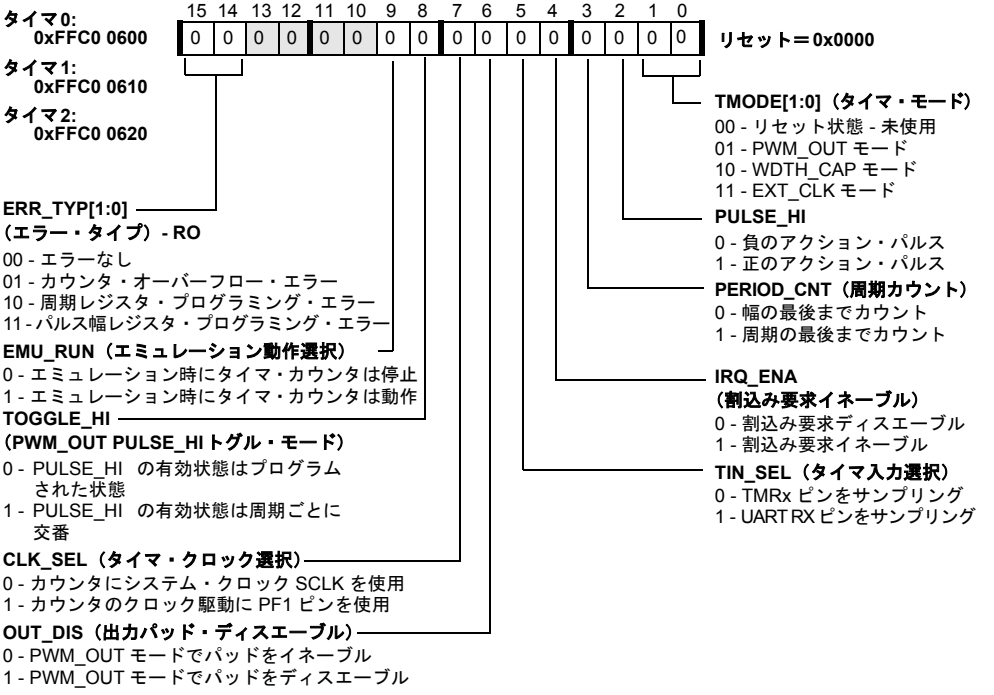
- レジスタがRead-Only (RO)、Write-1-to-Set (W1S)、または Write-1-to-Clear (W1C)である場合には、この情報は名前の下に表記されます。Read/Writeはデフォルトであり、特に表記されません。追加の説明文が続くこともあります。
- レジスタ内に全体的な読出し／書込み表記規則に従わないビットがある場合には、ビット名に続くビット説明にそのことが表記されます。
- ビットに短縮名がある場合には、短縮名がビット説明の最初に表記され、続いてカッコ内に正式名が表記されます。
- MMR割当ては、レジスタの左側に16進で表示します。複数のアドレスが関係する場合は、レジスタの下の表に示します。
- リセット値は個々のビットに2進で表記され、レジスタの右側に16進で表記されます。
- x とマークされたビットでは、リセット値が未知です。したがって、そのようなビットを含むレジスタのリセット値は、未定義であるか、またはリセット時のピン値に依存します。

- 網掛けされたビットは予約済みです。

将来のシステムとの上位互換性を保証するには、特に指定のない限り、レジスタ内の予約済みビットに対して、読み出した値は書き戻してください。

図P-1は、これらの表記規則の例を示します。

タイマ設定レジスタ (TIMERx_CONFIG)



コア・タイマ・カウント・レジスタ (TCOUNT)

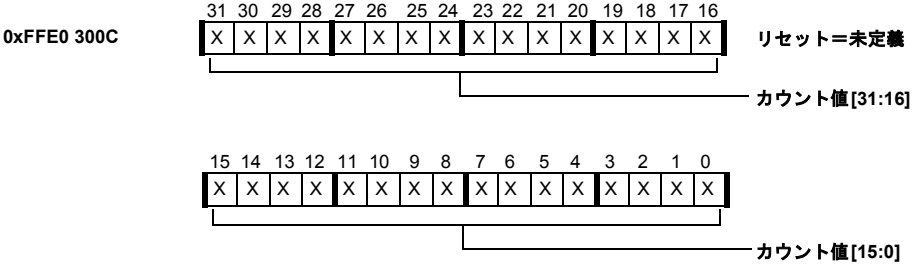


図 P-1. レジスタ図の例

第1章　はじめに

ADSP-BF533、ADSP-BF532、ADSP-BF531プロセッサは、Blackfinプロセッサ・ファミリーの強化されたメンバーであり、以前のBlackfinプロセッサに比べて性能と消費電力が大幅に改善されているだけでなく、その使いやすさとコード互換性の長所が継承されています。これら3つの新プロセッサは、完全にピン互換であり、性能とオンチップ・メモリだけが異なります。したがって、新しい製品開発に伴う多くの危険を軽減できます。

Blackfinプロセッサ・コア・アーキテクチャでは、デュアルMAC信号処理エンジン、RISCライクな直交性のあるマイクロプロセッサ命令セット、柔軟な単一命令複数データ (SIMD) 機能、およびマルチメディア機能を、単一の命令セット・アーキテクチャによって実現しています。

Blackfin製品はダイナミック・パワー・マネジメントを特長としています。この機能では、動作の電圧と周波数を変更することで、特定のタスクに合わせて消費電力プロファイルを最適化します。

ペリフェラル

プロセッサのシステム・ペリフェラルには、以下の要素が含まれます。

- パラレル・ペリフェラル・インターフェース (PPI)
- シリアル・ポート (SPORT)
- シリアル・ペリフェラル・インターフェース (SPI)
- 汎用タイマ

- 非同期シリアル・インターフェース (UART)
- リアルタイム・クロック (RTC)
- ウォッチドッグ・タイマ
- 汎用I/O（プログラマブル・フラグ）

図 1-1に示すように、これらのペリフェラルは、何本かの高帯域バスを介してコアに接続されます。

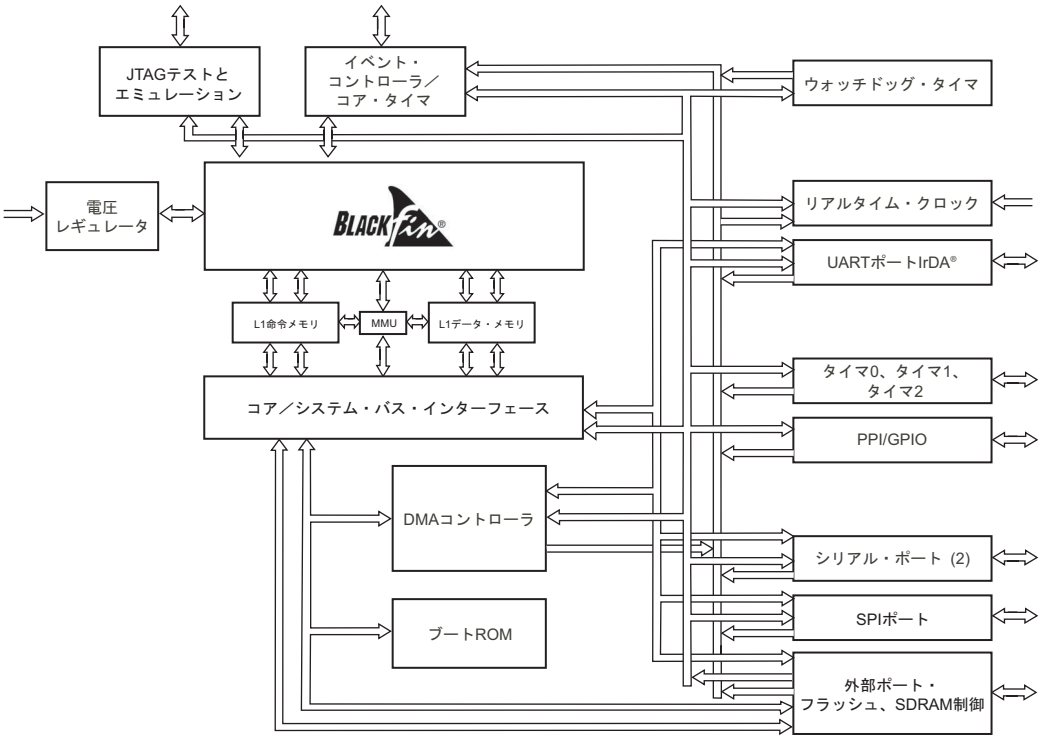


図 1-1. プロセッサのブロック図

汎用I/O、リアルタイム・クロック、タイマ以外のすべてのペリフェラルは、柔軟なDMA構造によってサポートされます。外付けSDRAMと非同期メモリを含めて、プロセッサのメモリ・スペース間でのデータ転送には、2つの独立した専用のメモリDMAチャンネルが用意されています。たとえ、すべてのオンチップ・ペリフェラルと外付けペリフェラルにアクティビティが存在する場合でも、複数のオンチップ・バスによってプロセッサ・コアの動作継続に十分な帯域幅が提供されます。

コア・アーキテクチャ

図 1-2に示すように、プロセッサ・コアには、2つの16ビット乗算器、2つの40ビット・アキュムレータ、2つの40ビット算術論理演算ユニット(ALU)、4つの8ビット・ビデオALU、および1つの40ビット・シフタが含まれます。演算ユニットは、レジスタ・ファイルからの8/16/32ビット・データを処理します。

演算レジスタ・ファイルには、8本の32ビット・レジスタが含まれます。16ビットのオペランド・データに対して演算動作を行うとき、このレジスタ・ファイルは、16本の独立した16ビット・レジスタとして動作します。演算動作用のすべてのオペランドは、マルチポート・レジスタ・ファイルと命令定数フィールドから得られます。

各MACでは、サイクルごとに16×16ビット乗算を実行して、40ビット結果に蓄積しておくことができます。符号付き／符号なしのフォーマット、丸め、飽和が可能です。

ALUでは、16ビットまたは32ビットのデータに対して、伝統的な算術／論理演算セットを実行します。さまざまな信号処理タスクを高速化するために、多くの特殊な命令が組み込まれています。これらの命令には、フィールド抽出やポピュレーション・カウントなどのビット操作、モジュロ2^{32}の乗算、除算プリミティブ、飽和と丸め処理、符号／指数部の検出などが含まれます。ビデオ命令セットには、バイト・アライメントとパッキング操作、クリッピング機能を持つ16ビットと8ビットの加算、8ビットの平

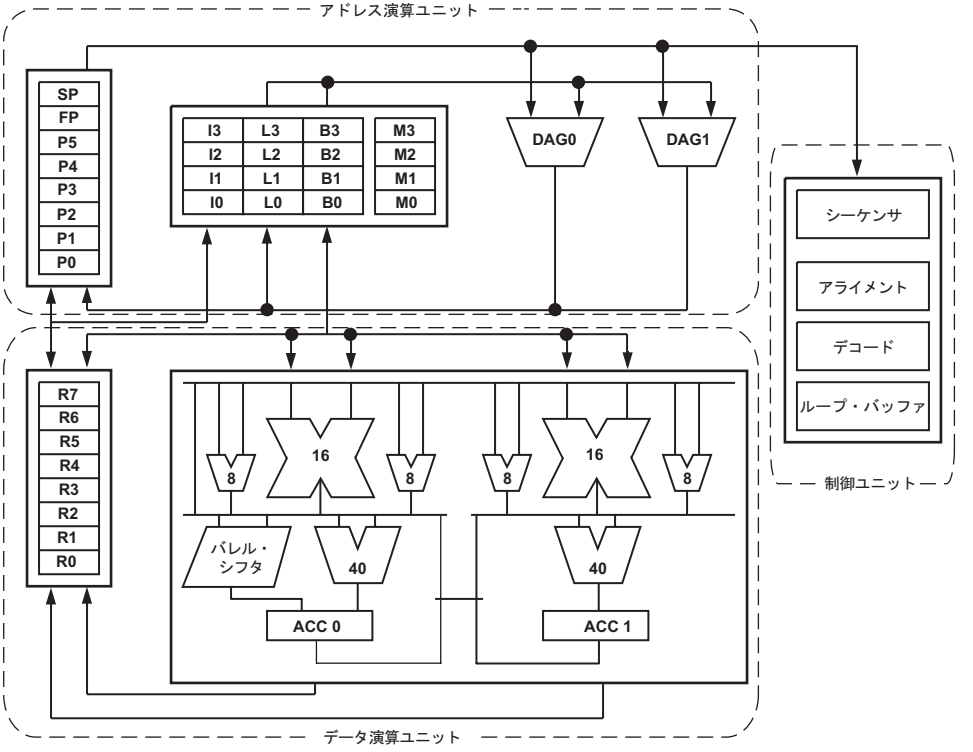


図 1-2. プロセッサ・コア・アーキテクチャ

均演算、および8ビットの減算／絶対値／アキュムレート (SAA) 演算が含まれます。また、比較／選択命令とベクトル検索命令も用意されています。命令によっては、レジスタ・ペア（演算レジスタの上位16ビットと下位16ビット）に対して2つの16ビットALU演算を同時に実行できます。また、2番目のALUを使用して、4つの16ビット演算も可能です。

40ビット・シフタでは、データを保管して、シフト、ローテート、正規化、および抽出の操作を実行できます。

プログラム・シーケンサでは、命令の整列とデコーディングを含めて、命令の実行フローを制御します。プログラム・フローの制御に関しては、シーケンサは、PC相対と間接の条件付きジャンプ（静的分岐予測付き）およびサブルーチン呼出しをサポートします。ゼロオーバーヘッド・ループ機能に対応するためのハードウェアも提供されています。アーキテクチャは完全にインターロックされるため、データ依存性を持つ命令の実行時に、パイプラインの影響を意識する必要はありません。

アドレス演算ユニットは、メモリからの同時デュアル・フェッチのために2つのアドレスを提供します。このユニットには、4セットの32ビット・インデックス・レジスタ、モディファイ・レジスタ、レングス・レジスタ、ベース・レジスタ（循環バッファリング用）と8本の追加32ビット・ポインタ・レジスタ（Cスタイルのインデックス付きスタック操作用）で構成される、マルチポート・レジスタ・ファイルが含まれています。

Blackfinプロセッサは、階層型メモリ構造と組み合わせた改良ハーバード・アーキテクチャを採用しています。一般に、レベル1 (L1) メモリは、ほとんどレイテンシなしでプロセッサの最大速度で動作します。L1レベルでは、命令メモリは命令だけを保持します。2つのデータ・メモリがデータを保持し、専用のスクラッチパッド・データ・メモリがスタックとローカル変数を格納します。

さらに、複数のL1メモリ・ブロックが提供され、SRAMとキャッシュを混在させたものとして構成できます。メモリ・マネジメント・ユニット (MMU) は、コア上で動作可能な個々のタスクに対してメモリ保護機能を提供して、システム・レジスタを想定外のアクセスから保護できます。

このアーキテクチャは、ユーザ・モード、スーパーバイザ・モード、エミュレーション・モードという3つの動作モードを提供します。ユーザ・モードでは、アクセスがシステム・リソースのサブセットに制限されるため、ソフトウェア環境を保護できます。スーパーバイザ・モードとエミュレーション・モードでは、システム・リソースとコア・リソースに無制限にアクセスできます。

ADSP-BF53x Blackfinプロセッサの命令セットは、最も頻繁に使用される命令を16ビットのオペコードで表すように最適化されています。複雑なDSP命令は、多機能命令として32ビットのオペコードにエンコードされます。Blackfin製品は、限定された多重発行機能を提供することで、32ビット命令を2つの16ビット命令と並行して発行できます。これによってプログラマは、コア・リソースの多くを単一の命令サイクルで使用できます。

ADSP-BF53x Blackfinプロセッサのアセンブリ言語では、代数構文を使用します。このアーキテクチャは、Cコンパイラでの使用に最適化されています。

メモリ・アーキテクチャ

Blackfinプロセッサのアーキテクチャは32ビット・アドレスを使用して、メモリを1つの統一された4Gバイトアドレス・スペースとして構成します。内部メモリ、外部メモリ、I/Oコントロール・レジスタを含めてすべてのリソースは、この共通アドレス・スペースの別個のセクションを占有します。このアドレス・スペースのメモリ部分は階層構造で整理されており、きわめて高速で低レイテンシのオンチップ・メモリをキャッシュやSRAMとし、大容量で低価格であるものの、低性能のオフチップ・メモリ・システムと併用することで優れたコスト／性能バランスを実現します。表1-1には、ADSP-BF531、ADSP-BF532、ADSP-BF533プロセッサのメモリ比較を示します。

表 1-1.　メモリ比較

メモリのタイプ	ADSP-BF531	ADSP-BF532	ADSP-BF533
命令 SRAM／キャッシュ	16K バイト	16K バイト	16K バイト
命令 SRAM	16K バイト	32K バイト	64K バイト
データ SRAM／キャッシュ	16K バイト	32K バイト	32K バイト
データ SRAM	–	–	32K バイト

表 1-1.　メモリ比較（続き）

メモリのタイプ	ADSP-BF531	ADSP-BF532	ADSP-BF533
スクラッチパッド	4K バイト	4K バイト	4K バイト
合計	84K バイト	116K バイト	148K バイト

L1メモリ・システムは、コアが使用できる最高性能のプライマリ・メモリです。外部バス・インターフェース・ユニット (EBIU) を通じてアクセスされるオフチップ・メモリ・システムは、SDRAM、フラッシュ・メモリ、SRAMによる拡張メモリを提供し、オプションでは132Mバイトまでの物理メモリにアクセスできます。

メモリDMAコントローラは、高帯域のデータ転送機能を提供し、内部メモリ・スペースと外部メモリ・スペースとの間でコードやデータのブロック転送を実行できます。

■ 内部メモリ

このプロセッサには、コアへの高帯域アクセスを提供する、3ブロックのオンチップ・メモリがあります。

- L1命令メモリ:SRAMと4ウェイ・セットアソシエイティブ・キャッシュから構成されます。このメモリは、プロセッサの最大速度でアクセスされます。
- L1データ・メモリ : SRAMまたは2ウェイ・セットアソシエイティブ・キャッシュ、あるいはその両方から構成されます。このメモリ・ブロックは、プロセッサの最大速度でアクセスされます。
- L1スクラッチパッドRAM : L1メモリと同じ速度で動作しますが、データSRAMとしてのみアクセス可能であり、キャッシュ・メモリとしては構成できません。

■ 外部メモリ

外部（オフチップ）メモリは、外部バス・インターフェース・ユニット (EBIU) を介してアクセスされます。この16ビット・インターフェースは、1バンクの同期DRAM （SDRAM）に加えて、フラッシュ・メモリ、EPROM、ROM、SRAM、メモリマップドI/Oデバイスなど、4バンクもの非同期メモリ・デバイスにグルーレスな接続を提供します。

PC133準拠のSDRAMコントローラは、最高128MバイトのSDRAMに接続できるようにプログラムできます。

非同期メモリ・コントローラは、最高4バンクのデバイスを制御するようにプログラムできます。各バンクは、使用されるデバイスのサイズとは無関係に1Mバイトのセグメントを占有するため、これらのバンクは、それぞれに1Mバイトのメモリがフル実装された場合にのみ連続になります。

■ I/O メモリ・スペース

Blackfinプロセッサは、独立したI/Oスペースを用いません。すべてのリソースは、フラットな32ビット・アドレス・スペースにマッピングされます。オンチップI/Oデバイスのコントロール・レジスタは、4Gバイト・アドレス・スペースの最上位近くのアドレスで、メモリマップド・レジスタ (MMR) にマッピングされます。これらのアドレスは、2つの小さなブロックに分割されます。一方のブロックには、すべてのコア機能に対するコントロールMMRが含まれています。他方のブロックには、コア外部のオンチップ・ペリフェラルのセットアップと制御に必要なレジスタが含まれています。MMRは、スーパーバイザ・モードでのみアクセスでき、オンチップ・ペリフェラルからは予約済みスペースのように見えます。

イベント処理

このプロセッサのイベント・コントローラは、プロセッサに対するすべての非同期／同期イベントを処理します。プロセッサのイベント処理では、ネスティングと優先順位付けが可能です。ネスティングによって、複数のイベント・サービス・ルーチンを同時にアクティブにできます。優先順位付けによって、優先順位の高いイベントの処理は、優先順位の低いイベントの処理よりも優先されます。コントローラは、5種類のイベントに対応します。

- エミュレーション：プロセッサはエミュレーション・モードに入り、JTAGインターフェースを介してプロセッサの指揮／制御が可能になります。
- リセット：プロセッサをリセットします。
- マスク不能割込み (NMI)：このイベントを生成するのは、プロセッサへのNMI入力信号またはソフトウェア・ウォッチドッグ・タイマです。NMIイベントは、システムの所定のシャットダウンを開始するためのパワーダウン・インジケータとしてよく使用されます。
- 例外：プログラム・フローに同期します。すなわち、命令が完了する前に、例外が実行されます。例外は、データ整列違反や未定義命令などの条件によって発生します。
- 割込み：プログラム・フローに同期しません。割込みは、入力ピン、タイマ、その他のペリフェラルによって発生します。

各イベントには、戻りアドレスを保持するための関連レジスタと、関連するイベント復帰命令があります。イベントがトリガされると、プロセッサの状態はスーパーバイザ・スタックに保存されます。

プロセッサ・イベント・コントローラは、コア・イベント・コントローラ (CEC) とシステム割込みコントローラ (SIC) の2段で構成されます。CECはSICと連携して動作し、すべてのシステム・イベントの優先順位付けと制御を行います。概念上、ペリフェラルからの割込みはSICに到着し、CECの汎用割込みに直接転送されます。

■ コア・イベント・コントローラ (CEC)

コア・イベント・コントローラは、専用の割込みイベントと例外イベントに加えて、9つの汎用割込み (IVG15-7) に対応します。これらの汎用割込みのうち、優先順位の最も低い2つの割込み (IVG15-14) をソフトウェア割込みハンドラ用に予約し、残り7つの割込み入力をペリフェラルのサポート用にすることをお勧めします。

■ システム割込みコントローラ (SIC)

システム割込みコントローラは、多数のペリフェラル割込み源からCECの優先順位付けされた汎用割込み入力まで、イベントのマッピングとルーティングを可能にします。プロセッサはデフォルト・マッピングを提供しますが、ユーザは割込み割当てレジスタ (IAR) に適切な値を書き込むことによって、割込みイベントのマッピングと優先順位を変更できます。

DMA サポート

このプロセッサには複数の独立したDMAコントローラがあり、コアへの最小のオーバーヘッドでデータの自動転送を実現します。DMA転送は、内部メモリとそのDMA対応のペリフェラルとの間で行うことができます。さらに、DMA対応のペリフェラルと、SDRAMコントローラや非同期メモリ・コントローラなど外部メモリ・インターフェースに接続された外部デバイスとの間でも、DMA転送を行うことができます。DMA対応の

ペリフェラルには、SPORT、SPIポート、UART、PPIが含まれます。個々のDMA対応のペリフェラルには、少なくとも1つの専用DMAチャンネルがあります。

DMAコントローラは、1次元 (1D) と2次元 (2D) のDMA転送をサポートします。DMA転送の初期化は、レジスタから行うか、あるいはディスクリプタ・ブロックと呼ばれる一連のパラメータから行います。

2D DMA機能は、64Kエレメント×64Kエレメントまでの任意の行／列サイズと、+/-32Kエレメントまでの任意の行／列ステップ・サイズに対応します。さらに、列ステップ・サイズは行ステップ・サイズよりも小さくできるため、インターリーブされたデータ・ストリームの実装が可能です。この機能は、実行時にデータをデインターリーブできるビデオ・アプリケーションで特に便利です。

利用可能なDMAの種類には、以下のようなものがあります。

- 完了時に停止する単一のリニア・バッファ
- バッファが満杯／断片的に満杯になるたびに割込みを発生する自動リフレッシュ循環バッファ
- ディスクリプタのリンク・リストを使用する1D/2D DMA
- 共通ページ内のベースDMAアドレスだけを指定する、ディスクリプタのアレイを使用する2D DMA

専用のペリフェラルDMAチャンネルのほかに、システムのさまざまなメモリ間の転送用に独立したメモリDMAチャンネルも提供されています。これによって、プロセッサの介入を最小限に抑えながら、外部SDRAM、ROM、SRAM、フラッシュ・メモリなど任意のメモリ間でデータのブロック転送が可能になります。メモリDMA転送は、非常に柔軟なディスクリプタ・ベースの手法または標準的なレジスタ・ベースのオート・バッファ・メカニズムによって制御できます。

外部バス・インターフェース・ユニット

このプロセッサの外部バス・インターフェース・ユニット (EBIU) は、業界標準の多種多様なメモリ・デバイスと接続できます。コントローラは、SDRAMコントローラと非同期メモリ・コントローラから構成されます。

■ PC133 SDRAM コントローラ

SDRAMコントローラは、シングル・バンクの業界標準のSDRAMデバイスまたはDIMMにインターフェースを提供します。このバンクは、PC133 SDRAM規格に完全に準拠し、16～128Mバイトのメモリを内蔵するように構成できます。

一連のプログラマブル・タイミング・パラメータを使用して、低速なメモリ・デバイスに対応するようにSDRAMバンクを設定できます。デバイス数を最小限に抑えてシステム・コストを下げるために、メモリ・バンクは16ビット幅になっています。

■ 非同期コントローラ

非同期メモリ・コントローラは、最高4つのメモリ・バンクまたはI/Oデバイスに対して設定可能なインターフェースを提供します。各バンクは、異なるタイミング・パラメータで独立にプログラムできます。これによって、SRAM、ROM、フラッシュ EPROMなど多種多様なメモリ・デバイスに加え、標準的なメモリの制御線を使用する各種のI/Oデバイスとも接続できます。各バンクはプロセッサのアドレス・スペース内で1Mバイトのウィンドウを占有しますが、フル実装されていない場合にも、これらのバンクはメモリ・コントローラによって連続にされません。一連のメモリやI/Oデバイスとインターフェースをとるためのこれらのバンクは16ビット幅です。

パラレル・ペリフェラル・インターフェース

このプロセッサが提供するパラレル・ペリフェラル・インターフェース(PPI) を使用すれば、パラレルA/DおよびD/Aコンバータ、ITU-R 601/656ビデオ・エンコーダ／デコーダ、およびその他の汎用ペリフェラルと直接に接続できます。PPIは、専用の入力クロック・ピン、最高3本のフレーム同期ピン、および最高16本のデータ・ピンから構成されます。入力クロックは、システム・クロック・レートの半分までのパラレル・データ・レートに対応します。

ITU-R 656モードでは、PPIは、8/10ビット・データ・エレメントのデータ・ストリームを受信して解析します。埋め込まれたプリアンブル制御情報や同期情報のオンチップ・デコードがサポートされています。

3つのITU-R 656モードが用意されています。

- アクティブ・ビデオ専用 : PPIは、プリアンブル・シンボルEnd of Active Video（EAV）とStart of Active Video (SAV) の間にあるデータや、垂直ブランキング期間中に存在するデータを読み込みません。このモードでは、制御バイト・シーケンスはメモリに格納されずに、PPIによって取り除かれます。
- 垂直ブランキング専用 : PPIは、垂直ブランキング期間 (VBI) ライン上の水平ブランキング情報と制御バイト・シーケンスに加えて、VBIデータだけを転送します。
- 全体フィールド : PPIを通じて着信ビットストリーム全体が読み出されます。これには、アクティブ・ビデオ、制御プリアンブル・シーケンス、水平／垂直ブランキング期間に埋め込まれている補助データが含まれます。

明示的に対応しているわけではありませんが、ITU-R 656の出力は、フレーム構造の全体（アクティブ・ビデオ、ブランキング、制御情報を含む）をメモリ内に設定し、フレーム無同期モードでデータをPPIにストリーム出力することによっても実現できます。プロセッサの2D DMA機

能を使用すれば、静的フレーム・バッファ（ブランキング・コードと制御コード）を一度メモリ内に置いて、フレーム単位でアクティブ・ビデオ情報を更新することで、この転送を容易にできます。

PPIの汎用モードは、多種多様なデータ・キャプチャ・アプリケーションと伝送アプリケーション向けに用意されています。これらのモードは4つのメイン・カテゴリに分類され、それぞれPPI_CLKサイクルごとに最高16ビットのデータ転送が可能です。

- 内部的に生成されたフレーム同期によるデータ受信
- 外部的に生成されたフレーム同期によるデータ受信
- 内部的に生成されたフレーム同期によるデータ送信
- 外部的に生成されたフレーム同期によるデータ送信

これらのモードでは、ハードウェア・シグナリングによるビデオ通信に加えて、ADC/DAC接続もサポートします。モードの多くは、複数レベルのフレーム同期をサポートします。必要ならば、フレーム同期のアサーションとデータの送／受信との間にプログラマブルな遅延を挿入できます。

シリアル・ポート (SPORT)

このプロセッサは、シリアル通信とマルチプロセッサ通信用に2つのデュアル・チャンネル同期シリアル・ポート（SPORT0とSPORT1）を内蔵しています。SPORTは、以下の機能をサポートします。

- I^2S対応の双方向動作

  各SPORTには2セットの独立した送／受信ピンがあり、8チャンネルのI^2Sステレオ・オーディオが可能になります。

- バッファ（深さ8）された送／受信ポート

  各ポートには、他のプロセッサ・コンポーネントとの間でデータ・ワードを転送するためのデータ・レジスタと、データ・レジスタとの間でデータをシフト・イン／アウトするためのシフト・レジスタがあります。

- クロッキング

  各送／受信ポートは外部シリアル・クロックを使用でき、また、広い範囲で独自に周波数を生成できます。

- ワード長

  各SPORTは、長さ3～32ビットのシリアル・データ・ワードを、MSBファースト／LSBファーストのフォーマットで転送できます。

- フレーミング

  各送／受信ポートは、データ・ワードごとにフレーム同期信号の有無とは無関係に動作できます。フレーム同期信号の生成は、内部／外部、アクティブ・ハイ／ロー、2つのうちいずれかのパルス幅、早期／遅延フレーム同期で行えます。

- ハードウェアでの圧縮伸張

  各SPORTでは、ITU勧告G.711に基づいて、A則またはμ則の圧縮伸張を実行できます。圧縮伸張は遅延を増やすことなく、SPORTの送信チャンネルまたは受信チャンネル、あるいはその両方で選択できます。

- 1サイクルのオーバーヘッドを持つDMA動作

  各SPORTは、複数バッファのメモリ・データを自動的に送／受信できます。プロセッサは、SPORTとメモリの間で一連のDMA転送をリンクまたはチェーン実行できます。

- 割込み

  各送／受信ポートでは、データ・ワードの転送完了時、またはDMAによってデータ・バッファ全体を転送した後で割込みを生成します。

- マルチチャンネル機能

  各SPORTは、1024チャンネル中128チャンネルのウィンドウに対応し、H.100、H.110、MVIP-90、HMVIPの各規格と互換性があります。

シリアル・ペリフェラル・インターフェース (SPI) ポート

このプロセッサは内蔵するSPI互換ポートによって、複数のSPI互換デバイスと通信できます。

SPIインターフェースは、データ転送に3本のピンを使用します。そのうちの2本はデータ・ピン、1本はクロック・ピンです。SPIチップ・セレクト入力ピンによって、他のSPIデバイスがプロセッサを選択します。7本のSPIチップ・セレクト出力ピンによって、プロセッサは他のSPIデバイスを選択します。SPIセレクト・ピンは再設定されたプログラマブル・フラグ・ピンです。これらのピンを使用すると、SPIポートはマスター／スレーブ・モードとマルチマスター環境に対応する、全二重の同期シリアル・インターフェースを提供します。

SPIポートのボーレートとクロック位相／極性はプログラマブルであり、送／受信いずれかのデータストリームに対応するように設定可能なDMAコントローラが内蔵されています。SPIのDMAコントローラは常に単方向アクセスだけを処理できます。

SPIポートは、転送時にその2本のシリアル・データ・ラインとの間でデータを連続的にシフト・イン／アウトすることによって同時に送／受信します。シリアル・クロック・ラインでは、2本のシリアル・データ・ライン上でデータのシフトとサンプリングの同期をとります。

タイマ

このプロセッサには、4つの汎用プログラマブル・タイマ・ユニットがあります。3つのタイマには外部ピンがあります。このピンは、パルス幅変調器 (PWM) やタイマの出力として、タイマをクロック駆動するための入力として、または外部イベントのパルス幅を測定するためのメカニズムとして設定できます。これらのタイマ・ユニットは、PF1ピンに接続された外部クロック入力、PPI_CLKピンへの外部クロック入力、または内部SCLKに同期することができます。

タイマ・ユニットをUARTと組み合わせて使用すれば、データストリーム内のパルスの幅を測定して、シリアル・チャンネルにオートボー検出機能を提供できます。

これらのタイマは、プロセッサ・コアへの割込みを生成して、プロセッサ・クロックまたは外部信号のカウントに対する同期用の周期イベントを提供できます。

3つの汎用プログラマブル・タイマに加えて、4番目のタイマも提供されています。このタイマは内部プロセッサ・クロックによって駆動され、一般にオペレーティング・システムの周期割込みを生成するためのシステム・チック・クロックとして使用されます。

UART ポート

このプロセッサはPC標準のUARTと完全互換である、半二重の非同期シリアル・インターフェース(UART) ポートを内蔵しています。このUARTポートは、他のペリフェラルやホストに対して簡略なUARTインターフェースを提供し、シリアル・データをDMA対応の半二重方式で非同期転送します。UARTポートは、5～8のデータ・ビット、1または2のストップ・ビット、偶数／奇数パリティまたはパリティなしに対応します。また、次の2つの動作モードを提供します。

- プログラムI/O

  このプロセッサは、I/OマップUARTレジスタに対して書込み／読出しを行うことにより、データを送／受信します。データは送信と受信でダブル・バッファされています。

- ダイレクト・メモリ・アクセス (DMA)

  DMAコントローラは送／受信データを転送します。これによって、メモリとのデータ転送に必要な割込みの回数と頻度が減少します。UARTには2つの専用DMAチャンネルがあり、一方は送信用、他方は受信用です。これらのDMAチャンネルは転送速度が相対的に低いため、大部分のDMAチャンネルよりも低い優先順位になっています。

以下の機能を提供するために、UARTポートのボーレート、シリアル・データ・フォーマット、エラー・コードの生成とステータス、割込みをプログラムできます。

- 広範囲のビットレート
- フレーム当たり7～12ビットのデータ・フォーマット
- 送／受信動作によるプロセッサへのマスク可能割込みの生成

汎用タイマ機能と組み合わせて、オートボー検出にも対応しています。

UARTの機能は、Infrared Data Association (IrDA®) のシリアル赤外線物理層リンク仕様 (SIR) プロトコルのサポートによって、さらに拡張されます。

リアルタイム・クロック

このプロセッサのリアルタイム・クロック (RTC) は、現在の時刻、ストップウォッチ、アラームなど、強力なデジタル・ウォッチ機能を提供します。RTCは、プロセッサに外付けされた32.768kHzの水晶発振器によって駆動されます。RTCペリフェラルには専用の電源ピンがあるため、残りのプロセッサ部分が低消費電力状態にあるときでもパワーアップ状態で駆動されます。RTCは、秒、分、時、日のクロック・チックによる割込み、プログラマブルなストップウォッチ・カウントダウンに基づく割込み、設定したアラーム時刻での割込みなど、プログラマブルな複数の割込みオプションを提供します。

32.768kHzの入力クロック周波数は、プリスケーラによって1Hz信号まで分周されます。タイマのカウンタ機能は、60秒カウンタ、60分カウンタ、24時間カウンタ、32768日カウンタという、4つのカウンタで構成されます。

アラーム機能がイネーブルにされると、タイマの出力がアラーム・コントロール・レジスタ内に設定された値と一致したとき割込みが生成されます。アラームには2種類あり、最初のアラームは時刻用、2番目のアラームは日時用です。

ストップウォッチ機能では、設定した値から分単位の分解能でカウントダウンします。ストップウォッチがイネーブルにされて、カウンタがアンダーフローすると、割込みが生成されます。

他のペリフェラルと同様に、RTCでは、RTCウェイクアップ・イベントの生成と同時に、プロセッサをスリープ・モードやディープ・スリープ・モードからウェイクアップさせることができます。RTCウェイクアップ・イベントでは、オンチップ内部電圧レギュレータをパワーダウン状態からウェイクアップさせることもできます。

ウォッチドッグ・タイマ

このプロセッサに内蔵される32ビット・タイマを使用すれば、ソフトウェア・ウォッチドッグ機能を実装できます。ソフトウェア・ウォッチドッグ機能は、タイマがソフトウェアによってリセットされる前に切れた場合に、ハードウェア・リセット、マスク不能割込み(NMI)、または汎用割込みを生成してプロセッサを既知の状態に強制設定することで、システムの可用性を向上させることができます。プログラマはタイマのカウント値を初期化し、該当する割込みをイネーブルにして、タイマをイネーブルにします。その後、ソフトウェアは、カウントが設定値からゼロに到達する前に、カウンタを再設定する必要があります。これによって、外部ノイズやソフトウェア・エラーに起因して、タイマをリセットするはずのソフトウェアが停止した場合に、システムが未知の状態にとどまることを防止できます。

ハードウェア・リセットを生成するように設定すると、ウォッチドッグ・タイマはCPUとペリフェラルの両方をリセットします。リセットの後、ソフトウェアは、ウォッチドッグ・コントロール・レジスタのステータス・ビットを調べることにより、ハードウェア・リセットの原因はウォッチドッグであったか否かを判断できます。

このタイマは、最大周波数f_{SCLK}のシステム・クロック(SCLK)によりクロック駆動されます。

プログラマブル・フラグ

このプロセッサには、16本の双方向プログラマブル・フラグ (PF) または汎用I/OピンPF[15:0]があります。各ピンは、フラグ・コントロール・レジスタ、ステータス・レジスタ、割込みレジスタを使用して、個別に設定できます。

- フラグ方向コントロール・レジスタ：各PFxピンの方向を、入力または出力として指定します。
- フラグ・コントロール・レジスタとステータス・レジスタ：このプロセッサは、“write-1-to-modify”方式を採用しています。この方式では、フラグの任意の組み合わせを1回の命令で変更し、かつ変更しないフラグのレベルに影響を与えないようにできます。4本のコントロール・レジスタが用意されており、フラグ値をセットするときに書き込むレジスタ、フラグ値をクリアするときに書き込むレジスタ、フラグ値をトグルするときに書き込むレジスタ、フラグ値を指定するときに書き込むレジスタがあります。フラグ・ステータス・レジスタを読み出すと、ソフトウェアはフラグの状態を調べることができます。
- フラグ割込みマスク・レジスタ：2本のフラグ割込みマスク・レジスタにより、各PFxピンがプロセッサへの割込みとして機能するように設定できます。各フラグ値のセットとクリアに使われる2本のフラグ・コントロール・レジスタと同様に、一方のフラグ割込みマスク・レジスタはビットをセットして割込み機能をイネーブルにし、他方のフラグ割込みマスク・レジスタはビットをクリアして割込み機能をディスエーブルにします。入力として定義されたPFxピンはハードウェア割込みを生成するように設定でき、出力PFxピンはソフトウェア割込みによりトリガできます。
- フラグ割込み検出レジスタ：2 本のフラグ割込み検出レジスタは、各PFxピンをレベル検出にするかまたはエッジ検出にするかを指定します。さらに、エッジ検出の場合には、信号の立上がりエッジだ

けを検出するか、あるいは立上がりと立下がりの両エッジを検出するかを指定します。一方のレジスタでは検出タイプを指定し、他方のレジスタではエッジ検出で有効とするエッジを指定します。

クロック信号

このプロセッサは、外部水晶発振器、正弦波入力、または外部クロック発振器から出力されるバッファおよび整形されたクロックによって駆動できます。

この外部クロックは、プロセッサのCLKINピンに接続されます。CLKIN入力は、通常動作時に停止／変更させたり、仕様周波数未満で動作させることはできません。このクロック信号はTTL互換の信号である必要があります。

コア・クロック (CCLK) とシステム・ペリフェラル・クロック (SCLK) は、入力クロック (CLKIN) 信号から取り出されます。オンチップ・フェーズ・ロック・ループ (PLL) は、ユーザ・プログラマブルな1～63倍の倍率（指定の最小／最大VCO周波数による制限）でCLKIN信号を逓倍できます。デフォルトの倍率は10倍ですが、ソフトウェア命令シーケンスにより変更できます。PLL_DIVレジスタに書込みを行うだけで、実行時に周波数を変更できます。

すべてのオンチップ・ペリフェラルは、システム・クロック (SCLK) によって駆動されます。システム・クロック周波数を設定するには、PLL_DIVレジスタのSSEL[3:0]ビットを使用します。

ダイナミック・パワー・マネジメント

このプロセッサには4つの動作モードがあり、その性能／消費電力特性は動作モードによって異なります。その上、ダイナミック・パワー・マネジメントの制御機能によって、プロセッサ・コアの電源電圧をダイナミックに変更して、さらに消費電力を減らすことができます。各ペリフェラルに対するクロックの制御によっても、消費電力を減らすことができます。

■ フル・オン・モード（最大性能）

フル・オン・モードでは、PLLがイネーブルにされ、しかもバイパスされないため、最大動作周波数で動作できます。これは通常の実行状態であり、最大性能が得られます。プロセッサ・コアとイネーブルにされた全ペリフェラルは最大速度で動作します。

■ アクティブ・モード（中程度の電力節減）

アクティブ・モードでは、PLLはイネーブルにされますが、バイパスされます。PLLがバイパスされるため、プロセッサのコア・クロック（CCLK）とシステム・クロック（SCLK）は入力クロック（CLKIN）周波数で動作します。このモードでは、CLKIN対VCOの逓倍率を変更できますが、その変更はフル・オン・モードになるまで有効になりません。適切に設定されたL1メモリに対してDMAアクセスを使用できます。

アクティブ・モードでは、PLLコントロール・レジスタ（PLL_CTL）を使ってPLLをディスエーブルにできます。PLLをディスエーブルにした場合、フル・オン・モードやスリープ・モードに入る前にPLLを再イネーブルにする必要があります。

■ スリープ・モード（高い電力節減）

スリープ・モードでは、プロセッサ・コアへのクロック（CCLK）をディスエーブルにして消費電力を削減します。ただし、PLLとシステム・クロック（SCLK）は動作を維持します。一般に、外部イベントまたはRTCの動作により、プロセッサがウェイクアップします。スリープ・モードでは、割込みがアサートされると、プロセッサはPLLコントロール・レジスタ（PLL_CTL）内のバイパス・ビット（BYPASS）の値を調べます。バイパスがディスエーブルにされている場合、プロセッサはフル・オン・モードになります。バイパスがイネーブルにされている場合、プロセッサはアクティブ・モードになります。

スリープ・モードは、L1メモリへのシステムDMAアクセスには対応しません。

■ ディープ・スリープ・モード（最高の電力節減）

ディープ・スリープ・モードでは、プロセッサ・コアと同期システム・クロック（CCLKとSCLK）をディスエーブルにすることにより、最高の電力節減を実現します。RTCのような非同期システムは動作を続けますが、内部リソースや外部メモリにはアクセスできません。このパワーダウン・モードを終了するには、リセット割込みのアサーション、またはRTCによって生成される非同期割込みが必要です。ディープ・スリープ・モードでは、RTC非同期割込みによってプロセッサはアクティブ・モードになります。ディープ・スリープ・モードで$\overline{\text{RESET}}$がアサートされると、プロセッサはフル・オン・モードになります。

■ 休止状態

最低の消費電力を実現するために、この状態では内部電源（V_{DDINT}）をオフにしたままI/O電源（V_{DDEXT}）をオンに保持することができます。これは、厳密には上述の4つのモードのような動作モードではありませんが、動作モードと見なすとわかりやすくなります。

電圧レギュレーション

このプロセッサは、2.25～3.6Vの外部電源から0.8～1.2Vの内部電圧レベルを生成できる電圧レギュレータを内蔵しています。図 1-3は、パワー・マネジメント・システムに必要な代表的な外付け部品を示します。このレギュレータは、内部ロジック電圧レベルを制御し、電圧レギュレータ・コントロール・レジスタ（VR_CTL）を使って50mV単位で設定できます。スタンバイ消費電力を減らすには、内部電圧レギュレータの設定によって供給されるI/O電源を保持しながら、プロセッサ・コアへの電力供給を切断できます。この状態では、V_{DDEXT}を依然として印加できるため、外付けバッファは不要です。このレギュレータは、ユーザ指定によってディスエーブルにしてバイパスすることもできます。

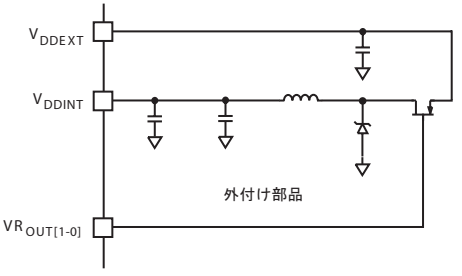


図 1-3.　電圧レギュレータの回路

ブート・モード

このプロセッサには、リセット後に内部L1命令メモリを自動ロードするための2つのメカニズムがあります。外部メモリから実行してブート・シーケンスをバイパスするように、3番目のモードも提供されています。

- 16ビット外部メモリからの実行：実行は、16ビット・パッキングのアドレス0x2000 0000から始まります。このモードでは、ブートROMはバイパスされます。すべての設定は、最低デバイス速度（3サイクルのホールド・タイム、15サイクルのR/Wアクセス・タイム、4サイクルのセットアップ）に設定されます。
- 8/16ビット外部フラッシュ・メモリからのブート：ブートROMメモリ・スペースに置かれているフラッシュ・ブート・ルーチンは、非同期メモリ・バンク0を使用して設定されます。すべての設定は、最低デバイス速度（3サイクルのホールド・タイム、15サイクルのR/Wアクセス・タイム、4サイクルのセットアップ）に設定されます。
- SPIシリアルEEPROM（8/16/24ビット・アドレス指定）からのブート：SPIは、PF2出力ピンを使って1つのSPI EEPROMデバイスを選択し、有効な8/16/24ビット・アドレス指定のEEPROMが検出されるまでアドレス0x00/0x0000/0x000000から連続した読出しコマンドを発行し、L1命令メモリの先頭からデータの書込みを開始します。

各ブート・モードでは、外部メモリ・デバイスから10バイトのヘッダが最初に読み込まれます[1]。このヘッダでは、転送バイト数とメモリ・デスティネーション・アドレスを指定します。どのブート・シーケンスでも複数のメモリ・ブロックをロードできます。すべてのブロックをロードした後、プログラム実行はL1命令SRAMの先頭から開始されます。

[1] Rev.0.3シリコンより、SPIホストからのスレーブ・ブートにも対応しています。

さらに、アプリケーション・コードによってリセット設定レジスタのビット4をセットすれば、ソフトウェア・リセット時に通常のブート・シーケンスをバイパスできます。この場合、プロセッサはL1命令メモリの先頭に直接ジャンプします。

命令セットの説明

ADSP-BF53xプロセッサ・ファミリーのアセンブリ言語命令セットでは、コーディングのしやすさと可読性を高めるために代数構文を採用しています。命令は、柔軟かつ高密度にエンコードされた命令セットを提供し、コンパイル後に最小のメモリ・サイズになるように特別に最適化されています。また、この命令セットでは、1つの命令で多くのプロセッサ・コア・リソースを使用可能にする、フル機能のマルチファンクション命令を提供します。また、マイクロコントローラで使用されている多くの機能との組み合わせにより、C/C++のソース・コードをコンパイルする際に非常に効率的です。さらに、このアーキテクチャでは、ユーザ動作モード（アルゴリズム／アプリケーション・コード）とスーパーバイザ動作モード(O/Sカーネル、デバイス・ドライバ、デバッガ、ISR)をサポートするため、コア・リソースに対する複数レベルのアクセスが可能です。

このアセンブリ言語は、プロセッサの独自のアーキテクチャを利用しており、以下の利点を提供します。

- シームレスに統合されたDSP/CPU機能は、8/16ビット動作に対して最適化されています。
- 多重発行ロード／ストア改良ハーバード・アーキテクチャによって、2回の16ビットMACまたは4回の8ビットALU＋2回のロード／ストア＋1サイクル当たり2回のポインタ更新に対応します。
- 全レジスタ、I/O、メモリが4Gバイトの統一されたメモリ・スペースにマップされているため、プログラミング・モデルが簡潔です。

- 任意のビットおよびビット・フィールドの操作、挿入、抽出などのマイクロコントローラ機能。8/16/32ビットのデータ型に対する整数演算。ユーザ・スタック・ポインタとスーパーバイザ・スタック・ポインタの分離。

コード密度の強化には、モード切替えやコード分離なしでの16/32ビット命令の混在が含まれます。使用頻度の高い命令は16ビットにエンコードされています。

開発ツール

プロセッサは、アナログ・デバイセズのエミュレータとVisualDSP++®開発環境を含む、CROSSCORE® ソフトウェアおよびハードウェア開発ツールの完全なセットによりサポートされています。他のアナログ・デバイセズ製品に対応している同じエミュレータ・ハードウェアが、ADSP-BF53xプロセッサ・ファミリーも完全にエミュレートします。

VisualDSP++プロジェクト管理環境によって、プログラマはアプリケーションの開発とデバッグが行えます。この環境には、代数構文に基づいた使いやすいアセンブラ、アーカイバ（ライブラリアン／ライブラリ・ビルダ）、リンカ、ローダー、システム・クロックに忠実な命令レベルのシミュレータ、C/C++コンパイラ、DSP関数と数学関数を含むC/C++ランタイム・ライブラリが含まれています。これらのツールのキー・ポイントは、C/C++コードの効率です。コンパイラは、C/C++コードをBlackfinプロセッサ・アセンブリに効率良く変換するように開発されています。Blackfinプロセッサには、コンパイルされたC/C++コードの効率を改善するアーキテクチャ上の機能があります。

VisualDSP++デバッガを使って、C/C++プログラムとアセンブリ・プログラムをデバッグすると、プログラマは次のことが可能になります。

- C/C++ とアセンブリの混在コードの表示（インターリーブされたソース／オブジェクト情報）

- ブレークポイントの挿入
- レジスタ、メモリ、スタックに対する条件付きブレークポイントの設定
- 命令実行のトレース
- プログラム実行の連続的または統計的なプロファイリングの実行
- メモリ内容のフィル、ダンプ、図式プロット
- ソース・レベル・デバッグの実行
- カスタム・デバッガ・ウィンドウの作成

VisualDSP++統合開発環境 (IDE) を使用すれば、ソフトウェア開発の定義と管理が可能になります。そのダイアログ・ボックスとプロパティ・ページによって、VisualDSP++エディタでのカラー構文強調表示など、すべての開発ツールの設定と管理ができます。これらの機能により、プログラマは次のことが可能になります。

- 開発ツールによる入力の処理方法と出力の生成方法の制御
- ツールのコマンドライン・スイッチとの1対1の対応の維持

VisualDSP++カーネル (VDK) は、DSPプログラミングのメモリ制約とタイミング制約に対処するように特別に作成されたスケジューリング管理とリソース管理を備えています。これらの機能を使うと、コードを効率的に開発できるようになり、新しいアプリケーション・コードを開発する際に、ゼロからスタートする必要がなくなります。VDKの機能には、スレッド領域、クリティカル領域、未スケジュール領域、セマフォ、イベント、デバイス・フラグが含まれています。また、VDKは優先順位ベースのプリエンプティブで協調動作的なタイムスライス・スケジューリング・アプローチにも対応しています。さらに、VDKはスケーラブルに設計されており、アプリケーションが特定の機能を使わない場合には、その機能に対するサポート・コードがターゲット・システムから除外されます。

VDKはライブラリであるため、VDKを使用するかどうかは開発者の判断によります。VDKは、VisualDSP++開発環境に統合されていますが、標準のコマンドライン・ツールでも使用できます。VDK開発環境は、システム・リソースの管理を支援し、VDKベースのさまざまなオブジェクトの生成を自動化し、アプリケーションのデバッグ時にシステム状態を視覚化します。

アナログ・デバイセズのエミュレータでは、プロセッサのIEEE 1149.1 JTAGテスト・アクセス・ポートを使って、エミュレーション時にターゲット・ボード・プロセッサの監視と制御を行います。このエミュレータではフルスピードのエミュレーションが可能なため、メモリ、レジスタ、プロセッサ・スタックの検査と変更が可能です。プロセッサのJTAGインターフェースを使用すると、エミュレータがターゲット・システムのローディングやタイミングに影響を与えない、非侵入型のインサーキット・エミュレーションが保証されます。

アナログ・デバイセズが提供するソフトウェア／ハードウェア開発ツールに加えて、サードパーティがBlackfinプロセッサ・ファミリーに対応する広範囲なツールを提供しています。ハードウェア・ツールには、ADSP-BF533 EZ-KIT Lite™スタンドアロン評価／開発カードなどがあります。サードパーティのソフトウェア・ツールには、DSPライブラリ、リアルタイム・オペレーティング・システム、ブロック図設計ツールなどがあります。

第2章　演算ユニット

プロセッサの演算ユニットは、DSPおよび一般の制御アルゴリズム用の数値処理を実行します。演算ユニットは、2つの算術論理演算ユニット(ALU)、2つの積和演算器（乗算器）ユニット、シフタ、一組のビデオALUで構成されます。これらのユニットは、データ・レジスタ・ファイルのレジスタからデータを取得します。これらのユニットの計算命令は固定小数点演算を提供し、各計算命令はどのサイクルでも実行できます。

演算ユニットでは、さまざまなタイプの演算を処理します。ALUでは、算術演算と論理演算を実行します。乗算器では乗算を実行し、乗算／加算および乗算／減算演算を実行します。シフタでは、論理シフトと算術シフトを実行し、ビットのパッキングと抽出を実行します。ビデオALUでは、特定の8ビット・データ・オペランドに対して単一命令複数データ(SIMD)の論理演算を実行します。

演算ユニットに出入りするデータは、それぞれ32ビット幅の8本のレジスタで構成されるデータ・レジスタ・ファイルを経由します。16ビット・オペランドを必要とする演算では、レジスタがペアにされて、16本の16ビット・レジスタが提供されます。

プロセッサのアセンブリ言語は、データ・レジスタ・ファイルへのアクセスを提供します。アセンブリ言語を使用すれば、これらのレジスタとの間でデータをやり取りすると同時に、計算のデータ・フォーマットを指定できます。

図 2-1は、この章で取り上げるその他のテーマを示します。各演算ユニットを調べれば、その動作の詳細に加えて、計算命令のまとめも参照できます。演算ユニット、レジスタ・ファイル、データ・バスの詳細を調べれば、計算のための適切なデータ・フローをよく理解できるようになります。次に、プロセッサの高度な並列処理の詳しい説明によって、多機能命令の利用方法が明らかになります。

図 2-1に、データ・レジスタ・ファイルと演算ユニット（乗算器、ALU、シフタ）との関連を示します。

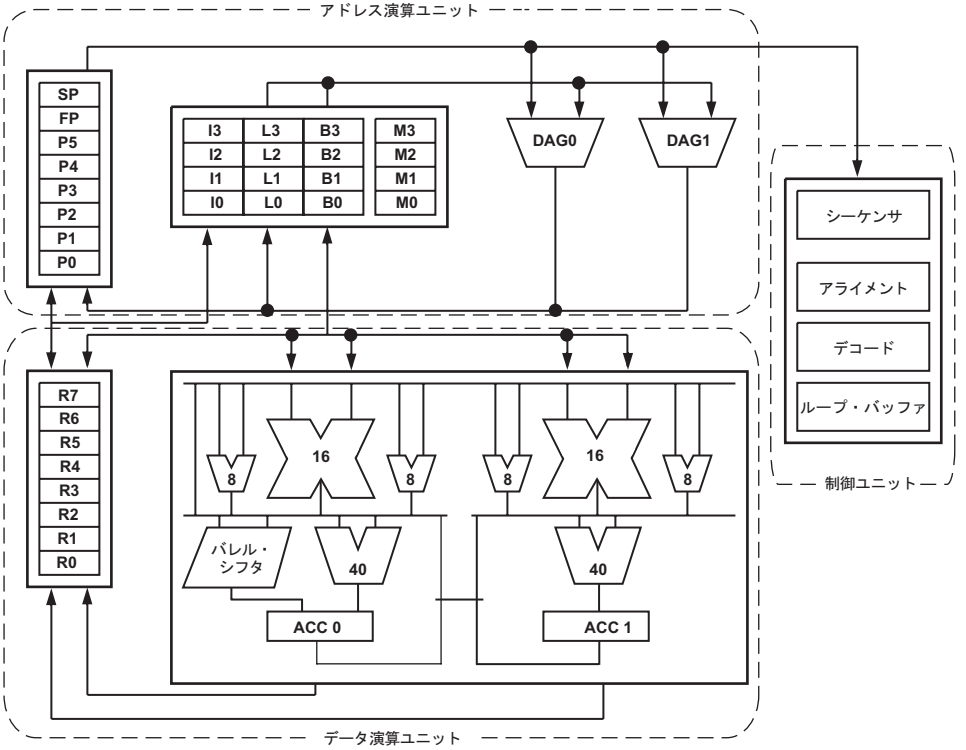


図 2-1.　プロセッサ・コア・アーキテクチャ

単機能乗算器命令、ALU命令、シフタ命令は、データ・レジスタ・ファイル内のデータ・レジスタに無制限のアクセスができます。多機能演算での制約については、それぞれの演算の節を参照してください。

2本の追加レジスタA0とA1は、40ビットの累算結果を提供します。これらのレジスタはALU専用であり、主に積和演算機能に使用されます。

小数と整数などの伝統的な算術演算モードは、命令で直接に指定されます。丸めモードはASTATレジスタから設定されます。このレジスタには、計算動作の結果のステータスと条件も記録されます。

データ・フォーマットの使い方

ADSP-BF53xプロセッサは、基本的に16ビットの固定小数点マシンです。大部分の演算では2の補数表現を想定していますが、符号なし数値や単純なビット列を想定する演算もあります。命令によっては32ビットの整数演算をサポートし、さらに特殊な機能によって8ビット演算とブロック浮動小数点をサポートすることもあります。各数値フォーマットの詳細については、付録D「数値フォーマット」を参照してください。

ADSP-BF53xプロセッサ・ファミリーの演算では、符号付き数値は、常に2の補数フォーマットで表現されます。これらのプロセッサでは、符号付き絶対値、1の補数、2進化10進数(BCD)、ゲタ履きフォーマットを使用しません。

■ ビット列

ビット列フォーマットは最も簡単な2進表記であり、16ビットが1つのビット・パターンとして扱われます。このフォーマットを使用する計算の例は、論理演算のNOT、AND、OR、XORです。これらのALU演算では、オペランドを符号ビットや2進小数点配置のないビット列として扱います。

■ 符号なし

符号なし2進数値は正であって、同じ長さの符号付き数値の絶対値のほぼ2倍の大きさを持つと考えることができます。プロセッサでは、多倍精度数値の下位ワードを符号なし数値として扱います。

■ 符号付き数値：2 の補数

ADSP-BF53xプロセッサの演算では、「**符号付き**」という語は2の補数値を表します。ADSP-BF53xプロセッサ・ファミリーの演算では、2の補数演算を想定しています。

■ 小数表現：1.15

ADSP-BF53xプロセッサの演算は、1.15（左に1ビット、右に15ビット）で表される小数バイナリ・フォーマットでの数値に最適化されています。1.15フォーマットでは、1つの符号ビット（最上位ビット(MSB)）と15個の小数ビットによって、－1～0.999969までの値を表します。

図 2-2は、1.15数値の例とそれに等価な10進値に加えて、1.15数値でのビットの重みを示します。

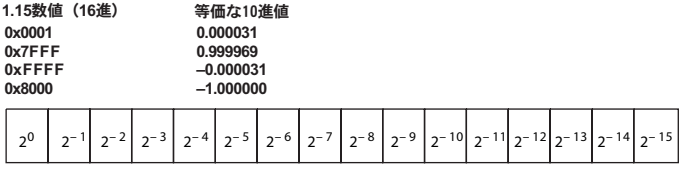


図 2-2.　1.15 数値でのビットの重み

レジスタ・ファイル

プロセッサの演算ユニットには、データ・レジスタ・ファイル、ポインタ・レジスタ・ファイル、データ・アドレス・ジェネレータ (DAG) レジスタ・セットという、3つのレジスタ・グループがあります。

- データ・レジスタ・ファイルは、演算ユニットのデータ・バスからオペランドを受け取り、計算結果を格納します。
- ポインタ・レジスタ・ファイルには、アドレッシング動作用のポインタがあります。
- DAG レジスタは、DSP 演算用のゼロオーバーヘッド循環バッファを管理する専用レジスタです。

詳細については、第5章「データ・アドレス・ジェネレータ」を参照してください。

プロセッサのレジスタ・ファイルを図 2-3に示します。

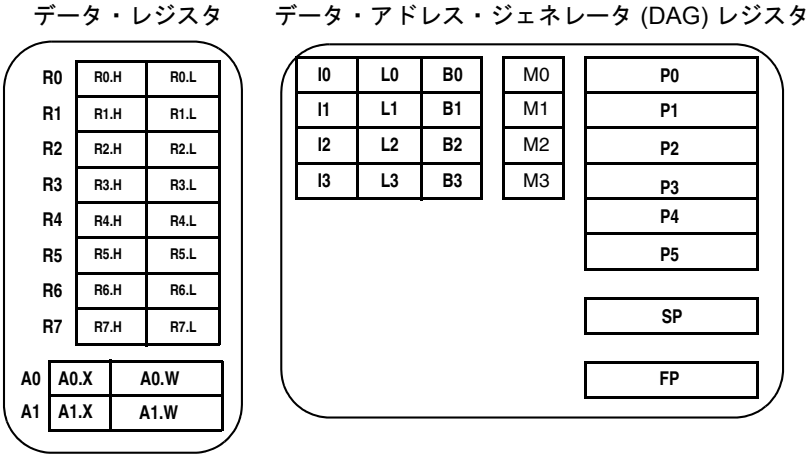


図 2-3.　レジスタ・ファイル

プロセッサでは、1ワードは32ビットの長さです。Hは32ビット・レジスタの上位16ビットを示し、Lは32ビット・レジスタの下位16ビットを示します。たとえば、A0.Wには40ビットA0レジスタの下位32ビットが含まれています。A0.LにはA0.Wの下位16ビットが含まれ、A0.HにはA0.Wの上位16ビットが含まれています。

■ データ・レジスタ・ファイル

データ・レジスタ・ファイルは、32ビット幅の8本のレジスタで構成されます。各レジスタは、独立した16ビット・レジスタのペアと見なすこともできます。それぞれは下位ハーフまたは上位ハーフとして表されます。したがって、32ビット・レジスタR0は、2つの独立したレジスタ・ハーフR0.LおよびR0.Hと見なすこともできます。

32ビット幅の3本のバス（2本は読出し、1本は書込み）によって、レジスタ・ファイルはL1データ・メモリに接続されます。データ・レジスタ・ファイルとデータ・メモリとの間の転送では、各サイクルで最高4つの16ビット・ワードの有効データを移動できます。

■ アキュムレータ・レジスタ

データ・レジスタ・ファイルに加えて、プロセッサには2本の専用の40ビット・アキュムレータ・レジスタがあります。それぞれ、その16ビット下位ハーフ (An.L) または上位ハーフ (An.H) と、その8ビット拡張部 (An.X) として表すことができます。さらには、下位32ビットで構成される32ビット・レジスタ (An.W)、または完全な40ビット結果レジスタ (An) として表すこともできます。

■ ポインタ・レジスタ・ファイル

汎用のアドレス・ポインタ・レジスタはPレジスタとも呼ばれ、以下の要素で構成されます。

- 6エントリのPレジスタ・ファイルP[5:0]
- 現在の手続きの駆動レコードのポイントに使用されるフレーム・ポインタ (FP)
- ランタイム・スタック上で最後に使用された位置のポイントに使用されるスタック・ポインタ (SP) レジスタ。第3章「動作モードと状態」のモード依存レジスタを参照。

Pレジスタは32ビット幅です。Pレジスタは主にアドレス計算に使用されますが、限定された算術演算セットによって一般の整数演算（たとえば、カウンタの保守）に使用することもできます。しかし、データ・レジスタとは異なり、Pレジスタの演算は、算術ステータス (ASTAT) レジスタのステータス・フラグに影響を与えません。

■ DAG レジスタ・セット

DSP命令では、主にアドレス指定用にデータ・アドレス・ジェネレータ (DAG) レジスタ・セットを使用します。DAGレジスタ・セットは以下のレジスタから構成されます。

- I[3:0]にはインデックス・アドレスが含まれています。
- M[3:0]には変更値が含まれています。
- B[3:0]にはベース・アドレスが含まれています。
- L[3:0]には長さの値が含まれています。

すべてのDAGレジスタは、32ビット幅です。

I（インデックス）レジスタとB（ベース）レジスタには、常にメモリ内の8ビット・バイトのアドレスが含まれています。インデックス・レジスタには実効アドレスが含まれています。M（モディファイ）レジスタには、インデックス・レジスタの1つに対して加算／減算されるオフセット値が含まれています。

BレジスタとL（レングス）レジスタでは、循環バッファを定義します。Bレジスタには、バッファの開始アドレスが含まれています。Lレジスタには、バイト単位での長さが含まれています。LレジスタとBレジスタの各ペアには、対応するIレジスタが関連付けられています。たとえば、L0とB0は常にI0に関連付けられています。しかし、任意のMレジスタを任意のIレジスタに関連付けることもできます。たとえば、I0をM3によって変更することもできます。詳細については、第5章「データ・アドレス・ジェネレータ」を参照してください。

■ レジスタ・ファイル命令のまとめ

表 2-1は、レジスタ・ファイル命令を示します。アセンブリ言語構文の詳細については、『ADSP-BF53x Blackfin Processor Instruction Set Reference』を参照してください。

表 2-1では、以下の記号の意味に注意してください。

- Allregは、R[7:0]、P[5:0]、SP、FP、I[3:0]、M[3:0]、B[3:0]、L[3:0]、A0.X、A0.W、A1.X、A1.W、ASTAT、RETS、RETI、RETX、RETN、RETE、LC[1:0]、LT[1:0]、LB[1:0]、USP、SEQSTAT、SYSCFG、CYCLES、CYCLES2を示します。
- A*n*は、ALU結果レジスタA0またはA1を示します。
- Dregは、データ・レジスタ・ファイルの任意のレジスタを示します。
- Sysreg は、システム・レジスタである ASTAT、SEQSTAT、SYSCFG、RETI、RETX、RETN、RETE、またはRETS、LC[1:0]、LT[1:0]、LB[1:0]、CYCLES、CYCLES2を示します。

- Pregは、任意のポインタ・レジスタ、FP、またはSPレジスタを示します。
- Dreg_evenは、R0、R2、R4、またはR6を示します。
- Dreg_oddは、R1、R3、R5、またはR7を示します。
- DPregは、データ・レジスタ・ファイルの任意のレジスタまたは任意のポインタ・レジスタ、FP、またはSPレジスタを示します。
- Dreg_loは、データ・レジスタ・ファイルの任意のレジスタの下位16ビットを示します。
- Dreg_hiは、データ・レジスタ・ファイルの任意のレジスタの上位16ビットを示します。
- A*n*.Lは、アキュムレータA0.WまたはA1.Wの下位16ビットを示します。
- A*n*.Hは、アキュムレータA0.WまたはA1.Wの上位16ビットを示します。
- Dreg_byteは、各データ・レジスタの下位8ビットを示します。
- Option (X) は、符号拡張を示します。
- Option (Z) は、ゼロ拡張を示します。
- *は、命令の結果に応じて、フラグがセットまたはクリアされることを示します。
- **は、フラグがクリアされることを示します。
- – は、影響がないことを示します。

表 2-1.　レジスタ・ファイル命令のまとめ

命令	ASTAT のステータス・フラグ						
	AZ	AN	AC0 AC0_COPY AC1	AV0 AVS	AV1 AV1S	CC	V V_COPY VS
allreg = allreg ; [1]	–	–	–	–	–	–	–
A*n* = A*n* ;	–	–	–	–	–	–	–
A*n* = Dreg ;	–	–	–	–	–	–	–
Dreg_even = A0 ;	*	*	–	–	–	–	*
Dreg_odd = A1 ;	*	*	–	–	–	–	*
Dreg_even = A0, Dreg_odd = A1 ;	*	*	–	–	–	–	*
Dreg_odd = A1, Dreg_even = A0 ;	*	*	–	–	–	–	*
IF CC DPreg = DPreg ;	–	–	–	–	–	–	–
IF ! CC DPreg = DPreg ;	–	–	–	–	–	–	–
Dreg = Dreg_lo (Z) ;	*	**	**	–	–	–	**/–
Dreg = Dreg_lo (X) ;	*	*	**	–	–	–	**/–
A*n*.X = Dreg_lo ;	–	–	–	–	–	–	–
Dreg_lo = A*n*.X ;	–	–	–	–	–	–	–
A*n*.L = Dreg_lo ;	–	–	–	–	–	–	–
A*n*.H = Dreg_hi ;	–	–	–	–	–	–	–
Dreg_lo = A0 ;	*	*	–	–	–	–	*
Dreg_hi = A1 ;	*	*	–	–	–	–	*
Dreg_hi = A1 ; Dreg_lo = A0 ;	*	*	–	–	–	–	*
Dreg_lo = A0 ; Dreg_hi = A1 ;	*	*	–	–	–	–	*
Dreg = Dreg_byte (Z) ;	*	**	**	–	–	–	**/–
Dreg = Dreg_byte (X) ;	*	*	**	–	–	–	**/–

1　警告：すべてのレジスタ組合わせが許されるわけではありません。詳細については、『ADSP-BF53x Blackfin Processor Instruction Set Reference』のレジスタ移動命令の機能説明を参照してください。

データ型

このプロセッサは、32ビット・ワード、16ビット・ハーフ・ワード、およびバイト型に対応します。32/16ビット・ワードは整数または小数とすることができますが、バイトは常に整数です。整数データ型は符号付きまたは符号なしとすることができますが、小数データ型は常に符号付きです。

表 2-3は、メモリ内、レジスタ・ファイル内、アキュムレータ内に存在するデータのフォーマットを示します。この表では、文字*d*は1ビットを表し、文字*s*は1つの符号付きビットを表します。

命令によっては、レジスタ内のデータを32ビットに符号拡張またはゼロ拡張することでデータを操作します。

- 符号なしデータをゼロ拡張する命令
- 符号付き16ビット・ハーフ・ワードと8ビット・バイトを符号拡張する命令

その他の命令では、データを32ビットの数値として操作します。さらに、2つの16ビット・ハーフ・ワードまたは4つの8ビット・バイトは、32ビット値として操作できます。詳細については、『ADSP-BF53x Blackfin Processor Instruction Set Reference』の命令を参照してください。

表2-2では、以下の記号の意味に注意してください。

- s＝符号ビット
- d＝データビット
- “.”＝慣習による小数点。ただし、実際には数値内に小数点は現われません。
- 斜体は、隣接するビット以外のソースからのデータを意味します。

表 2-2.　データ・フォーマット

フォーマット	メモリでの表現	32 ビット・レジスタでの表現
32.0 符号なしワード	dddd dddd dddd dddd dddd dddd dddd dddd	dddd dddd dddd dddd dddd dddd dddd dddd
32.0 符号付きワード	sddd dddd dddd dddd dddd dddd dddd dddd	sddd dddd dddd dddd dddd dddd dddd dddd
16.0 符号なしハーフ・ワード	dddd dddd dddd dddd	0000 0000 0000 0000 dddd dddd dddd dddd
16.0 符号付きハーフ・ワード	sddd dddd dddd dddd	ssss ssss ssss ssss sddd dddd dddd dddd
8.0 符号なしバイト	dddd dddd	0000 0000 0000 0000 0000 0000 dddd dddd
8.0 符号付きバイト	sddd dddd	ssss ssss ssss ssss ssss ssss sddd dddd
0.16 符号なし小数	.dddd dddd dddd dddd	0000 0000 0000 0000 .dddd dddd dddd dddd
1.15 符号付き小数	s.ddd dddd dddd dddd	ssss ssss ssss ssss s.ddd dddd dddd dddd
0.32 符号なし小数	.dddd dddd dddd dddd dddd dddd dddd dddd	.dddd dddd dddd dddd dddd dddd dddd dddd
1.31 符号付き小数	s.ddd dddd dddd dddd dddd dddd dddd dddd	s.ddd dddd dddd dddd dddd dddd dddd dddd
パック 8.0 符号なしバイト	dddd dddd *dddd dddd* dddd dddd *dddd dddd*	dddd dddd *dddd dddd* dddd dddd *dddd dddd*
パック 0.16 符号なし小数	.dddd dddd dddd dddd *.dddd dddd dddd dddd*	.dddd dddd dddd dddd *.dddd dddd dddd dddd*
パック 1.15 符号付き小数	s.ddd dddd dddd dddd *s.ddd dddd dddd dddd*	s.ddd dddd dddd dddd *s.ddd dddd dddd dddd*

■ エンディアン性

内部メモリと外部メモリは、いずれもリトル・エンディアンのバイト順でアクセスされます。詳細については、6-71ページの「メモリ・トランザクション・モデル」を参照してください。

■ ALUのデータ型

各ALUでの演算では、符号付き除算プリミティブ(DIVS)を除いて、オペランドと結果を16/32ビットのビット列として扱います。ALUの結果ステータス・ビットでは、結果を符号付きとして扱い、ステータスをオーバーフロー・フラグ(AV0、AV1)と負フラグ(AN)によって示します。各ALUには専用のスティッキー・オーバーフロー・フラグAV0SとAV1Sがあります。これらのビットはいったんセットされると、ASTATレジスタへの直接書込みによってクリアされるまではセットされたままです。追加のVフラグは、2つのアキュムレータからレジスタ・ファイルへの結果の転送に応じてセットまたはクリアされます。さらに、スティッキーVSビットはVビットによってセットされ、クリアされるまではセットされたままです。

オーバーフロー・ビット(V、VS、AV0、AV0S、AV1、AV1S)のロジックは、2の補数演算をベースにしています。最上位ビット(MSB)の変化がオペランドの符号と演算の性質によって予測できなかった場合には、1つのビットまたはビット群がセットされます。たとえば、2つの正数を加算すると正の結果が得られなければなりません。符号ビットの変化はオーバーフローを意味し、対応するオーバーフロー・フラグAVnがセットされます。負数と正数を加算すると、負または正の結果が得られますが、オーバーフローは発生しません。

キャリー・ビット(AC0、AC1)のロジックは、符号なし絶対値演算をベースにしています。ビット16(MSB)からキャリーが生成された場合には、ビットはセットされます。キャリー・ビット(AC0、AC1)は、マルチワード演算の下位ワード部分に対して最も役立ちます。

ALUの結果によって、ステータス情報が生成されます。ALUステータスの使い方の詳細については、2-30ページの「ALU命令のまとめ」を参照してください。

■ 乗算器のデータ型

各乗算器からは、結果がビット列で得られます。入力は命令から与えられる情報に基づいて解釈されます。つまり、符号付き×符号付き、符号なし×符号なし、その組合せ、および丸め演算などです。乗算器からの32ビットの結果は符号付きと見なされ、A0またはA1レジスタのフル40ビット幅にわたって符号拡張されます。

プロセッサは2つのフォーマット調整モードを提供します。つまり、小数オペランド（1つの符号ビットと15個の小数ビットを持つ1.15フォーマット）用の小数モードと、整数オペランド（16.0フォーマット）用の整数モードです。

プロセッサが2つの1.15オペランドを乗算すると、その結果は2.30（2つの符号ビットと30個の小数ビット）の数になります。小数モードの場合、乗算器は、積を1ビット左に自動的にシフトしてから、その結果を乗算結果レジスタ（A0、A1）に転送します。このように冗長な符号ビットをシフトすることで、乗算結果は1.31フォーマットになり、これは1.15フォーマットに丸めることができます。結果のフォーマットを2-18ページの図2-4に示します。

整数モードの場合、左シフトは行われません。たとえば、オペランドが16.0フォーマットである場合、32ビットの乗算結果は32.0フォーマットになります。左シフトは不要であり、もし行えば数値表現が変化します。結果のフォーマットを2-18ページの図2-5に示します。

乗算結果は、アキュムレータを更新するときに、またはレジスタ・ファイル内のデスティネーション・レジスタに転送されるときにステータス情報を生成します。詳細については、2-41ページの「乗算器命令のまとめ」を参照してください。

■ シフタのデータ型

シフタでの多くの演算は、符号付き（2の補数）または符号なしであると明示的に示されます。論理シフトでは、符号なし絶対値またはビット列値を想定します。算術シフトでは、2の補数値を想定します。

指数ロジックでは、2の補数値を想定します。指数ロジックでは2の補数小数をベースにした、ブロック浮動小数点にも対応します。

シフト結果によって、ステータス情報が生成されます。シフタ・ステータスの使い方の詳細については、2-56ページの「シフタ命令のまとめ」を参照してください。

■ 演算フォーマットのまとめ

表 2-3、表 2-4、表 2-5、表 2-6では、計算動作の演算特性をまとめています。

表 2-3.　ALU の演算フォーマット

演算	オペランド・フォーマット	結果フォーマット
加算	符号付きまたは符号なし	フラグを解釈
減算	符号付きまたは符号なし	フラグを解釈
論理	ビット列	オペランドと同じ
除算	明示的に符号付きまたは符号なし	オペランドと同じ

表 2-4.　乗算器の小数モード・フォーマット

演算	オペランド・フォーマット	結果フォーマット
乗算	1.15 の明示的な符号付きまたは符号なし	2.30 を 1.31 にシフト
乗算／加算	1.15 の明示的な符号付きまたは符号なし	2.30 を 1.31 にシフト
乗算／減算	1.15 の明示的な符号付きまたは符号なし	2.30 を 1.31 にシフト

表 2-5.　乗算器演算の整数モード・フォーマット

演算	オペランド・フォーマット	結果フォーマット
乗算	16.0 の明示的な符号付きまたは符号なし	32.0、シフトなし
乗算／加算	16.0 の明示的な符号付きまたは符号なし	32.0、シフトなし
乗算／減算	16.0 の明示的な符号付きまたは符号なし	32.0、シフトなし

表 2-6.　シフタ演算のフォーマット

演算	オペランド・フォーマット	結果フォーマット
論理シフト	符号なしバイナリ文字列	オペランドと同じ
算術シフト	符号付き	オペランドと同じ
指数検出	符号付き	オペランドと同じ

■ 乗算器の整数／小数フォーマットの使い方

積和演算機能の場合、プロセッサは小数値 (1.15) 用の小数演算と、整数 (16.0) 用の整数演算という、2つの選択肢を提供します。

小数演算の場合、32ビットの積出力はフォーマット調整（符号拡張および1ビット左シフト）されてから、アキュムレータA0またはA1に加算されます。たとえば、積のビット31はA0のビット32（A0.Xのビット0）と合致

し、積のビット0はA0のビット1（A0.Wのビット1）と合致します。最下位ビット (LSB) はゼロ詰めされます。乗算結果の小数フォーマットを図2-4に示します。

整数演算の場合、32ビットの積レジスタはシフトされずにA0またはA1に加算されます。図 2-5は、整数モードの結果配置を示します。

小数演算や整数演算では、乗算器の出力積は40ビットの加算器／減算器に送られ、そこでA0またはA1レジスタの現在の内容に対して加算／減算されて、最終の40ビットの結果が得られます。

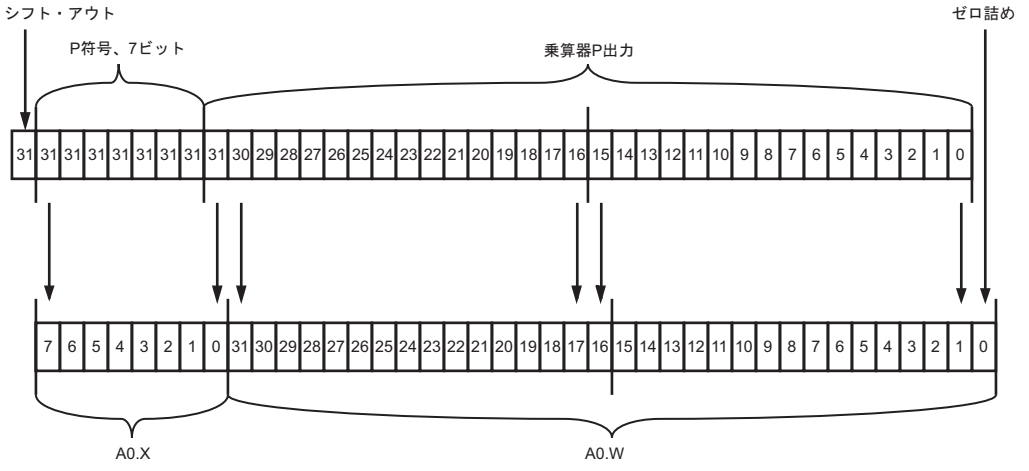


図 2-4.　乗算結果の小数フォーマット

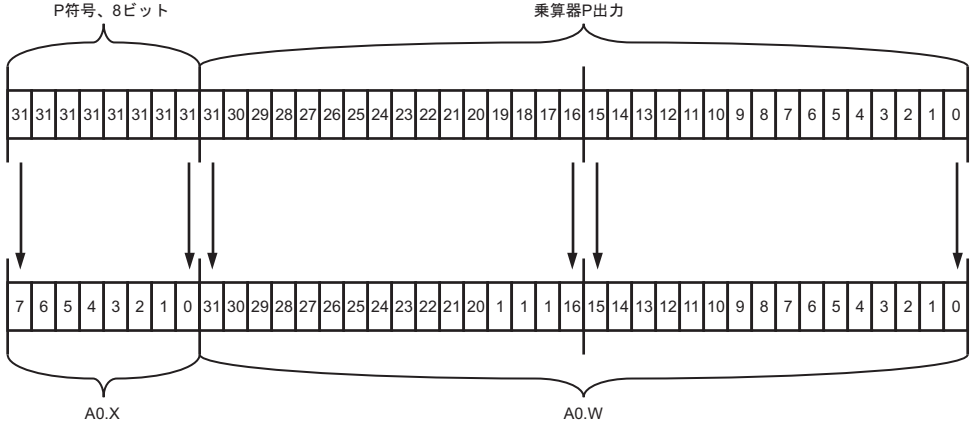


図 2-5.　乗算結果の整数フォーマット

■ 乗算器の丸め結果

多くの乗算器演算で、プロセッサは乗算結果の丸め（RNDオプション）に対応します。丸めは数値の精度を減らすための手段であり、数値の表現から下位レンジのビットを除去し、場合によっては、その前の値をより正確に表現するように数値の残り部分を変更することもあります。たとえば、元の数値はNビットの精度を持ちますが、新しい数値はMビットの精度しか持ちません（ここで、N＞M）。これにより、丸めのプロセスでは、数値からN－Mビットの精度が除去されます。

RNDオプションによってバイアスされた丸めが行われるか、バイアスのない丸めが行われるかは、ASTATレジスタのRND_MODビットによって決まります。**バイアスのない**丸めの場合、RND_MODビット＝0に設定します。**バイアスされた**丸めの場合、RND_MODビット＝1に設定します。

ほとんどのアルゴリズムでは、バイアスのない丸めが選択されます。

▶ バイアスのない丸め

収束丸め方式では、元の値に最も近い数値が返されます。元の数値が2つの数値のちょうど中間にある場合には、この方式では最も近い偶数値、つまり0のLSBを含む値が返されます。たとえば、3ビットの2の補数小数0.25（2進の0.01）を最も近い2ビットの2の補数小数に丸める場合、結果は0.0となります。なぜなら、それが0.5と0.0の偶数値選択だからです。この方式では、前後の値に基づいて切捨て／切上げが行われるため、**バイアスのない**丸めと呼ばれます。

バイアスのない丸めでは、ビット15とビット16との境界で40ビットの結果を丸めるというALUの機能を使用します。丸めは命令コードの一部として指定できます。丸めが選択されると、出力レジスタには、丸められた16ビットの結果が格納されます。アキュムレータでの丸めは行われません。

アキュムレータは、バイアスのない丸め方式を使用します。バイアスされた丸めという従来の方式では、加算器チェーンのビット位置15に1を加算します。この方法では、正の方向へのかたよりが生じます。なぜなら、A0.L/A1.L = 0x8000であるとき、中間値は常に切上げられるからです。

アキュムレータはこの中間点を検出すると、結果出力のビット16を0に設定することによってこのバイアスをなくします。ビット16を0に設定することは、奇数のA0.L/A1.L値を切り上げ、偶数値を切り下げる効果があるため、一様に分散した値を想定すると0という大きなサンプル・バイアスをもたらします。

以下の例は、*x*を使用して任意のビット・パターン（オール・ゼロ以外）を表します。図 2-6の例では、A0に対する代表的な丸め演算を示します。この例は、A1にも適用されます。

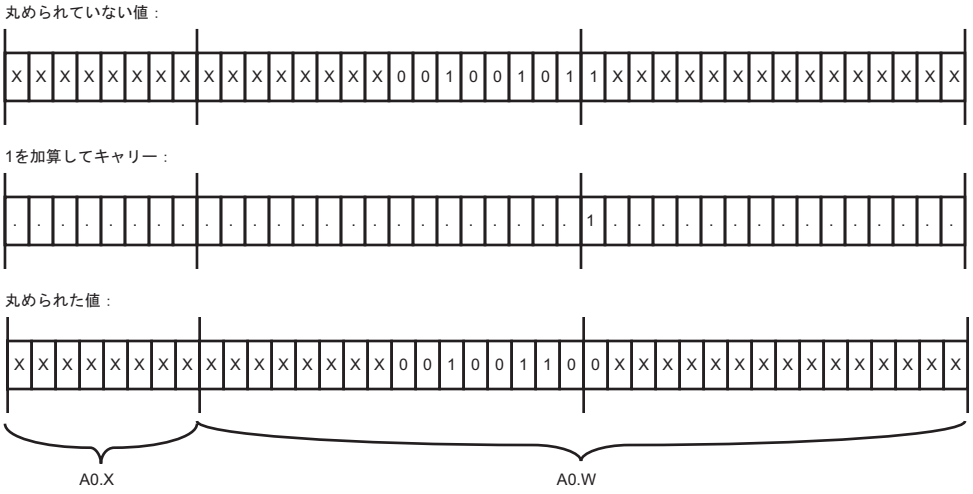


図 2-6.　代表的なバイアスのない乗算器丸め

図 2-7に示すように、下位15ビットがオール0であり、ビット15が1（中心値）であるとき、正へのかたよりを避けるための補償が明らかになります。

図 2-7では、A0のビット16が0に設定されます。このアルゴリズムはどの丸め演算でも採用されていますが、次の例の下位16ビットに示すビット・パターンが存在するときにだけ表面化します。

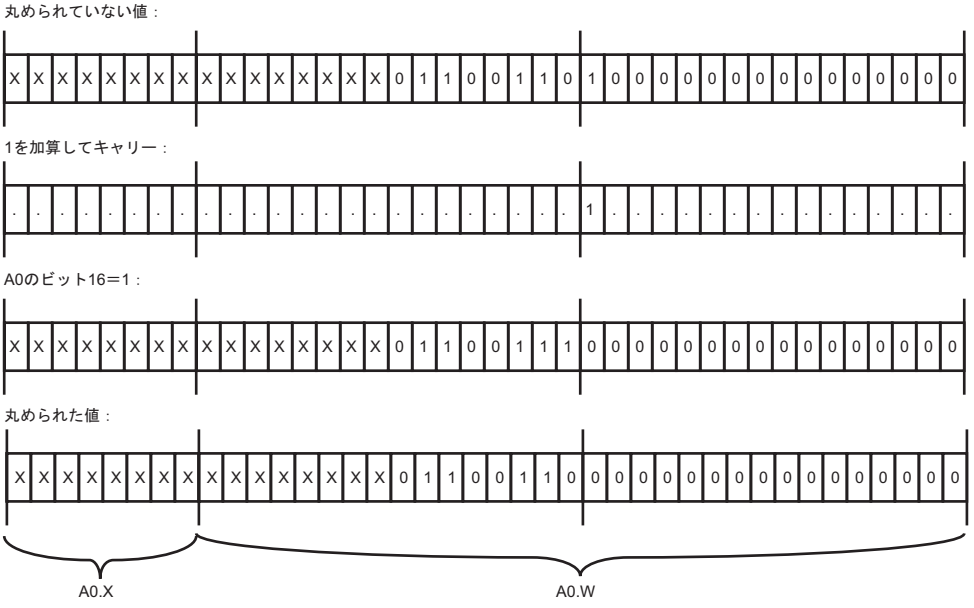


図 2-7.　バイアスのない乗算器丸めでの正へのかたよりの回避

▶ バイアスされた丸め

最近値への丸め方式でも、元の値に最も近い数値が返されます。しかし、元の数値が2つの数値のちょうど中間にある場合には、慣習的に常に2つのうちの大きな方へ切り上げられます。たとえば、3ビットの2の補数小数0.25（2進の0.01）を最も近い2ビットの2の補数小数に丸めるとき、この方式では0.5（2進の0.1）が返されます。元の小数は0.5と0.0（2進の0.0）のちょうど中間にあるため、この方式では切上げになります。常に切上げが行われるため、この方式は**バイアスされた**丸めと呼ばれます。

ASTAT レジスタの RND_MOD ビットは、バイアスされた丸めをイネーブルにします。RND_MOD ビットがクリアされると、2-19ページの「バイアスのない丸め」で説明したように、乗算器命令の RND オプションでは通常のバイアスのない丸め演算を使用します。

RND_MOD ビットがセット（＝1）されると、プロセッサはバイアスのない丸めの代わりに、バイアスされた丸めを使用します。バイアスされた丸めモードで動作するとき、A0.L/A1.L を 0x8000 に設定したすべての丸め演算では奇数値だけを切り上げるのではなく、常に切上げを行います。バイアスされた丸めの例については表 2-7 を参照してください。

表 2-7.　乗算器動作でのバイアスされた丸め

RND 前の A0/A1	バイアスされた RND 結果	バイアスのない RND 結果
0x00 0000 8000	0x00 0001 8000	0x00 0000 0000
0x00 0001 8000	0x00 0002 0000	0x00 0002 0000
0x00 0000 8001	0x00 0001 0001	0x00 0001 0001
0x00 0001 8001	0x00 0002 0001	0x00 0002 0001
0x00 0000 7FFF	0x00 0000 FFFF	0x00 0000 FFFF
0x00 0001 7FFF	0x00 0001 FFFF	0x00 0001 FFFF

バイアスされた丸めが結果に影響を与えるのは、A0.L/A1.L レジスタが 0x8000 を格納している場合だけです。その他の丸め演算はすべて普通に行われます。このモードでは、バイアスされた丸めを使用するビット指定のアルゴリズムを効率的に実装できます（たとえば、Global System for Mobile Communications (GSM) の音声圧縮ルーチン）。

▶ 切捨て

数値を表す有意なビットを減らす、もう1つの方法として一般的なのは、N－Mの下位ビットを単にマスクする方法です。このプロセスは**切捨て**と呼ばれ、比較的大きなバイアスが生じます。丸めに対応しない命令は切捨てを行います。ASTAT の RND_MOD ビットは切捨てに影響を与えません。

■ 特殊な丸め命令

前述のように、ALUはバイアスされた丸めまたはバイアスのない丸めによって、演算結果をデータ・レジスタに直接丸める機能を提供します。また、異なるビット境界で丸める機能も提供します。オプションRND12、RND、RND20では、それぞれビット12、ビット16、ビット20から16ビット値を抽出して、ASTATのRND_MODビットの状態とは無関係にバイアスされた丸めを実行します。

例：

```
R3.L = R4 (RND) ;
```

この列では、ビット16でバイアスされた丸めを実行し、結果をハーフ・ワードに保管します。

```
R3.L = R4 + R5 (RND12) ;
```

この列では、2つの32ビット数値を加算してビット12でバイアスされた丸めを実行し、結果をハーフ・ワードに保管します。

```
R3.L = R4 + R5 (RND20) ;
```

この列では、2つの32ビット数値を加算してビット20でバイアスされた丸めを実行し、結果をハーフ・ワードに保管します。

計算ステータスの使い方

乗算器、ALU、シフタは、プロセッサの算術ステータス (ASTAT) レジスタ内のオーバーフロー・フラグやその他のステータス・フラグを更新します。プログラム・シーケンス内の計算からステータス条件を使用するには、命令の実行後に条件付き命令を使用してASTATレジスタ内のCCフラグをテストします。この方法では、各命令の結果を監視できます。ASTATレジスタは32ビット・レジスタであり、いくつかのビットは予備です。将来のシステムとの互換性を保証するには、レジスタへの書込み時にこれらの予備ビットから読み出した値を書き戻す必要があります。

ASTAT レジスタ

図 2-8は算術ステータス (ASTAT) レジスタを示します。プロセッサは、最も新しいALU演算、乗算器演算、シフタ演算のステータスを示す、ASTAT のステータス・ビットを更新します。

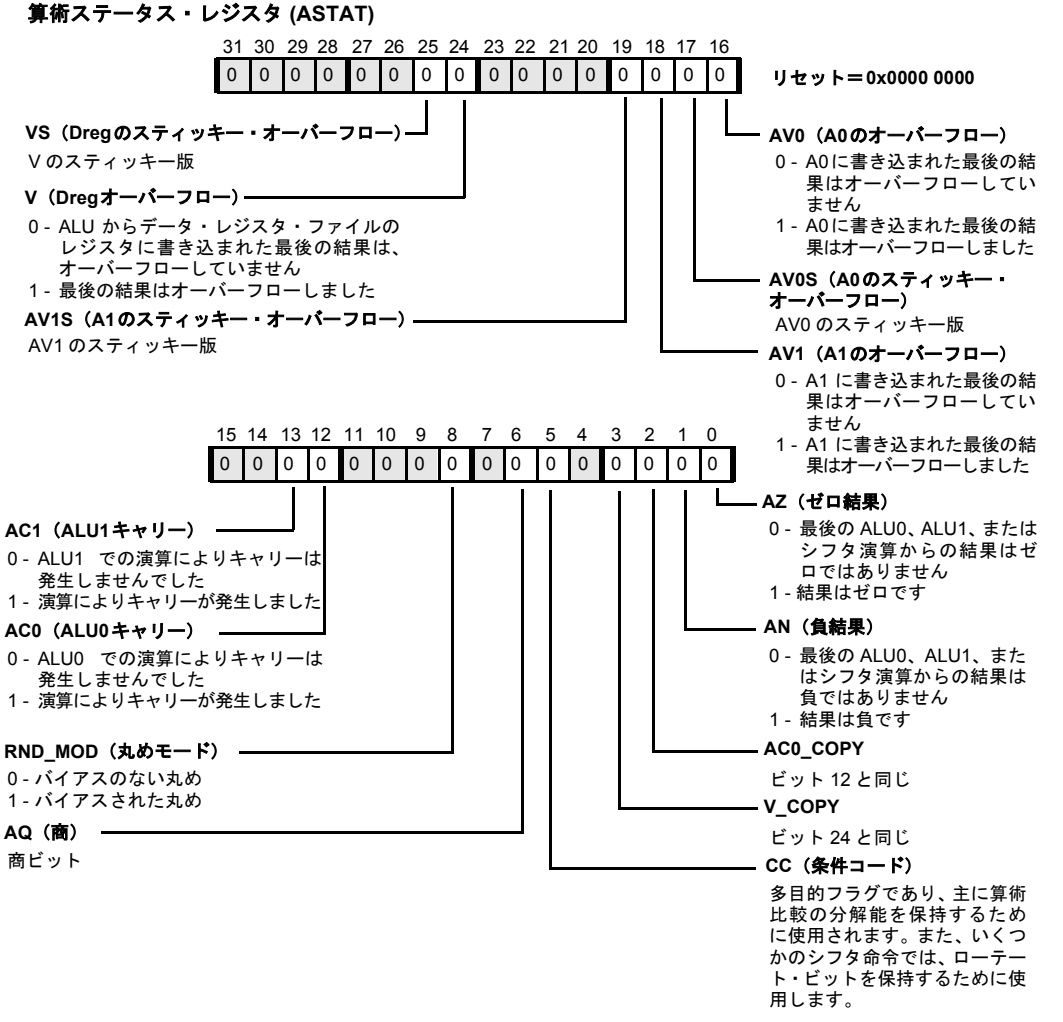


図 2-8.　算術ステータス・レジスタ

算術論理演算ユニット (ALU)

2つのALUは、固定小数点データに対して算術演算と論理演算を実行します。ALUの固定小数点命令は16/32/40ビットの固定小数点オペランドに作用し、16/32/40ビットの固定小数点結果を出力します。ALU命令には、以下の命令が含まれます。

- レジスタの固定小数点加算と減算
- 即値の加算と減算
- 乗算結果の累算と減算
- 論理AND、OR、NOT、XOR、ビット単位のXOR、ネゲート
- 機能：ABS、MAX、MIN、丸め、除算プリミティブ

■ ALU 演算

主なALU演算はALU0で行われますが、並列演算の場合には、ALU0演算のサブセットを実行するALU1も使います。

表 2-8は、各ALUで可能な入力と出力を示します。

表 2-8.　各 ALU での入力と出力

入力	出力
2 つまたは 4 つの 16 ビット・オペランド	1 つまたは 2 つの 16 ビットの結果
2 つの 32 ビット・オペランド	1 つの 32 ビットの結果
乗算器からの 32 ビットの結果	乗算器からの 32 ビットの結果と 40 ビットの累算結果との組合わせ

2つのALUでの演算を組合わせると、1つの命令で4つの16ビットの結果、2つの32ビットの結果、または2つの40ビットの結果が得られることもあります。

▶ シングル 16 ビット演算

シングル16ビット演算では、任意の2つの16ビット・レジスタ・ハーフをALUへの入力として使用できます。加算、減算、または論理演算によって生成される16ビットの結果は、任意のデスティネーション・レジスタ・ハーフに保管されます。ALU0がこの演算に使用されます。

例：

```
R3.H = R1.H + R2.L (NS) ;
```

この例では、R1.H（R1の上位ハーフ）の16ビットの内容をR2.L（R2の下位ハーフ）の内容に加算し、その結果を飽和なしでR3.H（R3の上位ハーフ）に保管します。

▶ デュアル 16 ビット演算

デュアル16ビット演算では、任意の2本の32ビット・レジスタを16ビット・オペランドのペアと見なして、ALUへの入力として使用できます。加算、減算、論理演算によって得られる2つの16ビットの結果は、任意の32ビット・デスティネーション・レジスタに保管されます。ALU0がこの演算に使用されます。

例：

```
R3 = R1 +|- R2 (S) ;
```

この例では、R2.H（R2の上位ハーフ）の16ビットの内容をR1.H（R1の上位ハーフ）の内容に加算し、結果を飽和付きでR3.H（R3の上位ハーフ）に保管します。

また、R1.L（R1の下位ハーフ）の内容からR2.L（R2の下位ハーフ）の16ビットの内容を減算し、結果を飽和付きでR3.L（R3の下位ハーフ）に保管します（2-36ページの図2-10を参照）。

▶ クワッド16ビット演算

クワッド16ビット演算では、任意の2本の32ビット・レジスタを16ビット・オペランドのペアと見なしてALU0とALU1への入力として使用できます。少数の加算演算や減算演算では、2本の任意の32ビット・デスティネーション・レジスタに保管される4つの16ビットの結果が得られます。ALU0とALU1の両方がこの演算に使用されます。データ・レジスタ・ファイルから演算ユニットまでは2本の32ビット・データ・パスしかないため、ALU0とALU1には同じ2ペアの16ビット入力が供給されます。命令構造はデュアル16ビット演算の場合と同じであり、2つのALUに対して入力オペランドは同じでなければなりません。

例：

```
R3 = R0 +|+ R1, R2 = R0 -|- R1 (S) ;
```

この例では、次の4つの演算を実行します。

- R1.H（R1の上位ハーフ）の16ビットの内容をR0.H（R0 の上位ハーフ）の16ビットの内容に加算し、その結果を飽和付きでR3.Hに保管します。
- R1.LをR0.Lに加算し、その結果を飽和付きでR3.Lに保管します。
- R0.H（R0 の上位ハーフ）の16ビットの内容からR1.H（R1の上位ハーフ）の16ビットの内容を減算し、その結果を飽和付きでR2.Hに保管します。
- R0.LからR1.Lを減算し、その結果を飽和付きでR2.Lに保管します。

以下の4命令は、上の命令と等価です。

```
R3.H = R0.H + R1.H (S) ;

R3.L = R0.L + R1.L (S) ;

R2.H = R0.H - R1.H (S) ;

R2.L = R0.L - R1.L (S) ;
```

▶ シングル 32 ビット演算

シングル32ビット演算では、任意の2本の32ビット・レジスタを32ビット・オペランドと見なして、ALUへの入力として使用できます。加算、減算、論理演算によって得られる32ビットの結果は、任意の32ビット・デスティネーション・レジスタに保管されます。ALU0がこの演算に使用されます。

データ・レジスタ・ファイルから得られる32ビット入力オペランドに加えて、8本のレジスタP[5:0]、SP、FPで構成されるポインタ・レジスタ・ファイルとの間でオペランドを入手／保管することができます。

命令では、ポインタ・レジスタとデータ・レジスタを混用できません。

例：

```
R3 = R1 + R2 (NS) ;
```

この例では、R2の32ビットの内容をR1の32ビットの内容に加算し、その結果を飽和なしでR3に保管します。

```
R3 = R1 + R2 (S) ;
```

この例では、R1 R1の32ビットの内容をR2の32ビットの内容に加算し、その結果を飽和付きでR3に保管します。

▶ デュアル 32 ビット演算

デュアル32ビット演算では、任意の2本の32ビット・レジスタを32ビット・オペランドのペアと見なして、ALU0とALU1への入力として使用できます。加算または減算によって得られる2つの32ビットの結果は、2本の32ビット・デスティネーション・レジスタに保管されます。ALU0とALU1はいずれもこの演算に使用されます。データ・レジスタ・ファイルから演算ユニットまでは2本の32ビット・データ・パスしかないため、ALU0とALU1には同じ2本の32ビット入力レジスタが供給されます。

例：

```
R3 = R1 + R2, R4 = R1 - R2 (NS) ;
```

この例では、R1の32ビットの内容にR2の32ビットの内容を加算し、その結果を飽和なしでR3に保管します。

また、R1の32ビットの内容からR2の32ビットの内容を減算し、その結果を飽和なしでR4に保管します。

この命令の特殊な形式では、入力オペランドとしてALUの40ビット結果レジスタを使用し、A0レジスタとA1レジスタの和と差を作成します。

例：

```
R3 = A0 + A1, R4 = A0 - A1 (S) ;
```

ALUレジスタの和と差の値を飽和付きの32ビットで結果レジスタに転送します。

■ ALU 命令のまとめ

表2-9はALU命令を示します。アセンブリ言語構文と、ALU命令がステータス・フラグに与える影響の詳細については、『ADSP-BF53x Blackfin Processor Instruction Set Reference』を参照してください。

表 2-9では、以下の記号の意味に注意してください。

- Dregは、データ・レジスタ・ファイルの任意のレジスタを示します。
- Pregは、任意のポインタ・レジスタ、FP、またはSPレジスタを示します。
- Dreg_lo_hiは、データ・レジスタ・ファイルの任意のレジスタ内の任意の16ビット・レジスタ・ハーフを示します。
- Dreg_loは、データ・レジスタ・ファイルの任意のレジスタの下位16ビットを示します。
- imm7は、7ビット幅の符号付き即値を示します。
- A*n*は、ALU結果レジスタのA0またはA1を示します。
- DIVSは、除算の符号プリミティブを示します。
- DIVQは、除算の商プリミティブを示します。
- MAXは、ソース・レジスタの最大（つまり正の無限大に最も近い）値を示します。
- MINは、ソース・レジスタの最小値を示します。
- ABSは、シングル32ビット・レジスタの上位／下位ハーフの絶対値を示します。
- RNDは、ハーフ・ワードの丸めを示します。
- RND12は、加算／減算の結果の飽和とビット12での結果の丸めを示します。
- RND20は、加算／減算の結果の飽和とビット20での結果の丸めを示します。
- SIGNBITSは、数値内の符号ビットの数－1を示します。

- EXPADJは、数値内の符号ビットの数－1とスレッショールド値のうち、小さい方を示します。
- *は、命令の結果に応じて、フラグがセット／クリアされることを示します。
- **は、フラグがクリアされることを示します。
- – は、影響がないことを示します。
- *d*は、AQに被除数のMSBと除数のMSBとの排他的論理和が含まれていることを示します。

表 2-9.　ALU 命令のまとめ

命令	ASTAT のステータス・フラグ						
	AZ	AN	AC0 AC0_COPY AC1	AV0 AV0S	AV1 AV1S	V V_COPY VS	AQ
Preg = Preg + Preg ;	–	–	–	–	–	–	–
Preg += Preg ;	–	–	–	–	–	–	–
Preg –= Preg ;	–	–	–	–	–	–	–
Dreg = Dreg + Dreg ;	*	*	*	–	–	*	–
Dreg = Dreg – Dreg (S) ;	*	*	*	–	–	*	–
Dreg = Dreg + Dreg, Dreg = Dreg – Dreg ;	*	*	*	–	–	*	–
Dreg_lo_hi = Dreg_lo_hi + Dreg_lo_hi ;	*	*	*	–	–	*	–
Dreg_lo_hi = Dreg_lo_hi – Dreg_lo_hi (S) ;	*	*	*	–	–	*	–
Dreg = Dreg +| + Dreg ;	*	*	*	–	–	*	–
Dreg = Dreg +| – Dreg ;	*	*	*	–	–	*	–
Dreg = Dreg –| + Dreg ;	*	*	*	–	–	*	–
Dreg = Dreg –| – Dreg ;	*	*	*	–	–	*	–

表 2-9.　ALU 命令のまとめ（続き）

命令	ASTAT のステータス・フラグ						
	AZ	AN	AC0 AC0_COPY AC1	AV0 AV0S	AV1 AV1S	V V_COPY VS	AQ
Dreg = Dreg + | +Dreg, Dreg = Dreg – | – Dreg ;	*	*	–	–	–	*	–
Dreg = Dreg + | – Dreg, Dreg = Dreg – | + Dreg ;	*	*	–	–	–	*	–
Dreg = A*n* + A*n*, Dreg = A*n* – A*n* ;	*	*	*	–	–	*	–
Dreg += imm7 ;	*	*	*	–	–	*	–
Preg += imm7 ;	–	–	–	–	–	–	–
Dreg = (A0 += A1) ;	*	*	*	*	–	*	–
Dreg_lo_hi = (A0 += A1) ;	*	*	*	*	–	*	–
A0 += A1 ;	*	*	*	*	–	–	–
A0 –= A1 ;	*	*	*	*	–	–	–
DIVS (Dreg, Dreg) ;	*	*	*	*	–	–	d
DIVQ (Dreg, Dreg) ;	*	*	*	*	–	–	d
Dreg = MAX (Dreg, Dreg) (V) ;	*	*	–	–	–	**/–	–
Dreg = MIN (Dreg, Dreg) (V) ;	*	*	–	–	–	**/–	–
Dreg = ABS Dreg (V) ;	*	**	–	–	–	*	–
A*n* = ABS A*n* ;	*	**	–	*	*	*	–
A*n* = ABS A*n*, A*n* = ABS A*n* ;	*	**	–	*	*	*	–
A*n* = –A*n* ;	*	*	*	*	*	*	–
A*n* = –A*n*, A*n* =– A*n* ;	*	*	*	*	*	*	–
A*n* = A*n* (S) ;	*	*	–	*	*	–	–
A*n* = A*n* (S), A*n* = A*n* (S) ;	*	*	–	*	*	–	–

表 2-9.　ALU 命令のまとめ（続き）

命令	ASTAT のステータス・フラグ						
	AZ	AN	AC0 AC0_COPY AC1	AV0 AV0S	AV1 AV1S	V V_COPY VS	AQ
Dreg_lo_hi = Dreg (RND) ;	*	*	–	–	–	*	–
Dreg_lo_hi = Dreg + Dreg (RND12) ;	*	*	–	–	–	*	–
Dreg_lo_hi = Dreg – Dreg (RND12) ;	*	*	–	–	–	*	–
Dreg_lo_hi = Dreg + Dreg (RND20) ;	*	*	–	–	–	*	–
Dreg_lo_hi = Dreg – Dreg (RND20) ;	*	*	–	–	–	*	–
Dreg_lo = SIGNBITS Dreg ;	–	–	–	–	–	–	–
Dreg_lo = SIGNBITS Dreg_lo_hi ;	–	–	–	–	–	–	–
Dreg_lo = SIGNBITS An ;	–	–	–	–	–	–	–
Dreg_lo = EXPADJ (Dreg, Dreg_lo) (V) ;	–	–	–	–	–	–	–
Dreg_lo = EXPADJ (Dreg_lo_hi, Dreg_lo);	–	–	–	–	–	–	–
Dreg = Dreg & Dreg ;	*	*	**	–	–	**/–	–
Dreg = ~ Dreg ;	*	*	**	–	–	**/–	–
Dreg = Dreg | Dreg ;	*	*	**	–	–	**/–	–
Dreg = Dreg ^ Dreg ;	*	*	**	–	–	**/–	–
Dreg =– Dreg ;	*	*	*	–	–	*	–

■ ALU のデータ・フローの詳細

図 2-9は、2-2ページの図2-1に示す演算ユニットとデータ・レジスタ・ファイルの詳細を示します。

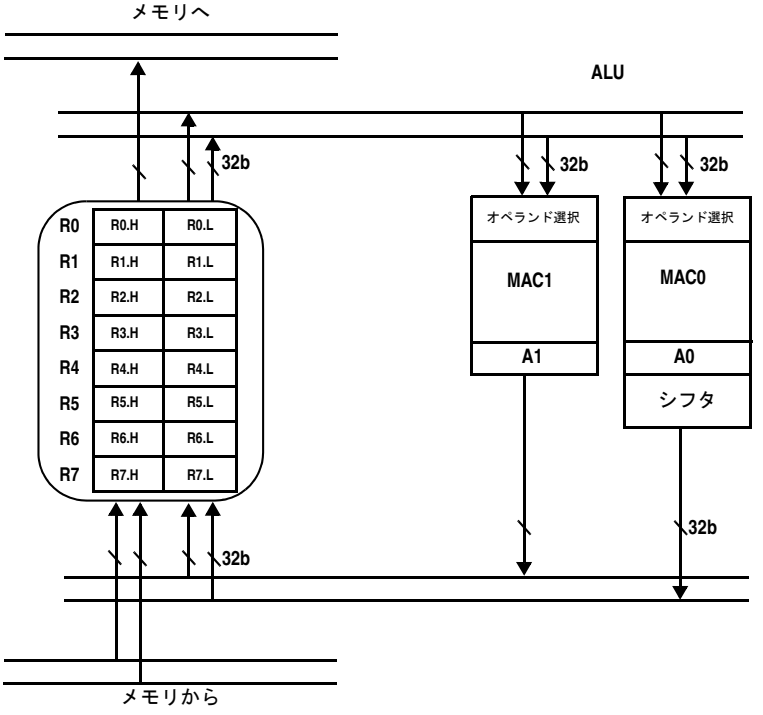


図 2-9.　レジスタ・ファイルと ALU

ここではALU0について説明します。ALU1は、ALU0のサブセットであり、非常に似ています。

各ALUでは、32ビットとデュアル16ビットの演算だけでなく、乗算結果の累算のための40ビット加算も実行します。各ALUには、16ビット・オペランドのペアまたは1つの32ビット・オペランドと見なせる32ビット

入力ポートが2つあります。シングル16ビット演算の場合には、4つの16ビット・オペランドのどれでもALUへの入力に供給される他の16ビット・オペランドと一緒に使用できます。

デュアル16ビット演算は図 2-10に示すように、上位ハーフと下位ハーフがペアにされ、加算と減算の4つの可能な組合わせを提供します。

```
(A) H + H, L + L     (B) H + H, L - L

(C) H - H, L + L     (D) H - H, L - L
```

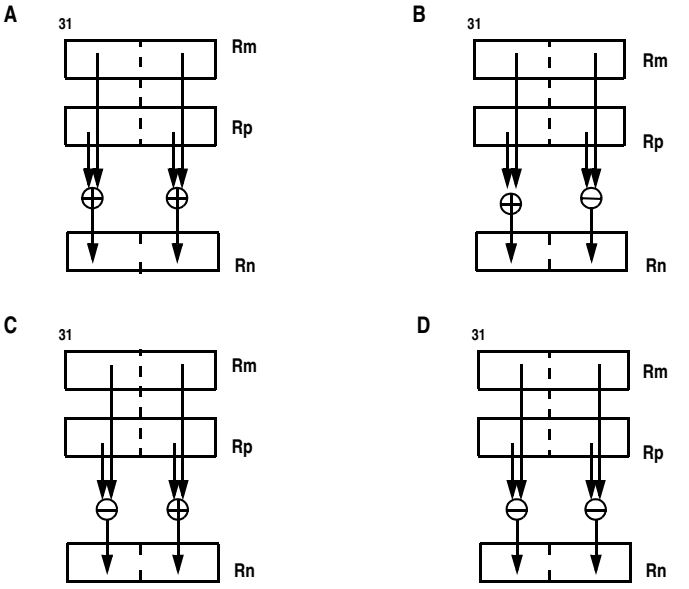


図 2-10. デュアル 16 ビット ALU 演算

▶ デュアル 16 ビットのクロス・オプション

デュアル16ビット演算の場合、結果がクロスされることもあります。「結果をクロスさせる」ことで、計算結果用の結果レジスタでの位置が変化します。通常、上位の計算からの結果は結果レジスタの上位ハーフに置かれ、下位の計算からの結果は結果レジスタの下位ハーフに置かれます。クロス・オプションを使用すれば、上位の結果はデスティネーション・レジスタの下位ハーフに置かれ、下位の結果はデスティネーション・レジスタの上位ハーフに置かれます（図 2-11を参照）。これが特に役立つのは、複雑な数値演算や高速フーリエ変換 (FFT) の一部を扱うときです。クロス・オプションはALU0にのみ適用されます。

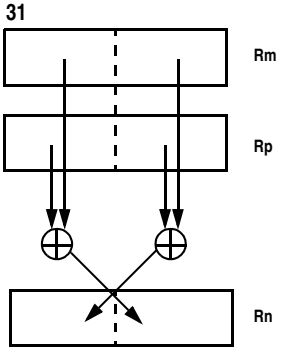


図 2-11. デュアル 16 ビット ALU 演算用のクロス・オプション

▶ ALU のステータス信号

各ALUは、ゼロ (AZ) ステータス、ネガティブ (AN) ステータス、キャリー (ACn) ステータス、スティッキー・オーバーフロー (AVnS) ステータス、イミディエイト・オーバーフロー (AVn) ステータス、商 (AQ) ステータスという、6つのステータス信号を生成します。すべての算術ステータス信号は、

サイクルの最後で算術ステータス・レジスタ (ASTAT) にラッチされます。ALU命令がステータス・フラグに与える影響については、2-32ページの表2-9を参照してください。

命令によっては、入力はデータ・レジスタ・ファイル、ポインタ・レジスタ・ファイル、または演算結果レジスタから得られます。32ビット・オペランドでの演算は、ALUでの高精度演算に直接対応します。

■ ALU の除算対応機能

ALUは、2つの特殊な除算プリミティブによる除算に対応します。これらの命令 (DIVS、DIVQ) を使用すると、プログラムは非回復型で条件付き（エラー・チェック）の加算／減算／除算アルゴリズムを実装できます。

除算は、符号付きまたは符号なしとすることができます。しかし、被除数と除数は、両方とも同じ型であることが必要です。除算の使い方の詳細とプログラミング例については『ADSP-BF53x Blackfin Processor Instruction Set Reference』を参照してください。

■ 特殊な SIMD ビデオ ALU 演算

4つの8ビット・ビデオALUによって、プロセッサはビデオ情報を効率よく処理できます。各ビデオALU命令では1～4ペアの8ビット入力を受け取り、1～4の8ビットの結果を返すことができます。入力は、データ・レジスタ・ファイルから2つの32ビット・ワードでビデオALUに供給されます。可能な動作には、以下のものが含まれます。

- クワッド8ビットの加算または減算
- クワッド8ビットの平均
- クワッド8ビットのパックまたはアンパック
- クワッド8ビットの減算－絶対－累算

- バイト整列

これらの命令の動作の詳細については『ADSP-BF53x Blackfin Processor Instruction Set Reference』を参照してください。

積和演算器（乗算器）

2つの乗算器（MAC0とMAC1）は、固定小数点の乗算と積和演算を実行します。積和演算は、累積加算または累積減算で使用できます。

乗算器の固定小数点命令は、16ビットの固定小数点データに用い、32ビットの積を40ビットのアキュムレータに対して加算または減算できます。

入力は、小数または整数、符号なしまたは2の補数として扱われます。乗算器命令には、以下の命令が含まれます。

- 乗算
- 加算による積和演算、丸めオプション
- 減算による積和演算、丸めオプション
- 上記のデュアル・バージョン

■ 乗算器の動作

各乗算器には2つの32ビット入力があり、そこから2つの16ビット・オペランドが得られます。シングル積和演算命令の場合、これらのオペランドはデータ・レジスタ・ファイル内の任意のデータ・レジスタとすることができます。各乗算器は、そのアキュムレータ・レジスタ`A1`または`A0`内に結果を累算できます。累算結果は、32ビットまたは40ビットに飽和させられます。乗算結果は丸めのオプション付きで、16/32ビットのデスティネーション・レジスタに直接書き込むこともできます。

両方の入力が整数フォーマットであるか小数フォーマットであるかは、各乗算器命令によって決まります。結果のフォーマットは、入力のフォーマットと一致します。MAC0では、両方の入力が符号付きまたは符号なしとして扱われます。MAC1には、混合モード・オプションがあります。

両方の入力が符号付き小数である場合には、冗長な符号ビットを除去するために乗算器は結果を自動的に1ビット左へシフトします。符号なし小数、整数、および混合モードでは、符号ビット訂正のためのシフトは行われません。乗算器命令のオプションでは、入力のデータ・フォーマットを指定します。詳細については、2-43ページの「乗算器命令のオプション」を参照してください。

▶ 積和演算器レジスタへの乗算結果の格納

2-35ページの図2-9に示すように、各乗算器には専用のアキュムレータA0またはA1があります。各アキュムレータ・レジスタは、A0.L/A1.L （ビット15:0）、A0.H/A1.H（ビット31:16）、A0.X/A1.X（ビット39:32）という、3つのセクションに分けられます。

乗算器がそのアキュムレータ結果レジスタに書込みを行うと、32ビットの結果は結合されたアキュムレータ・レジスタの下位ビットに保管され、MSBはレジスタの上位8ビット（A0.X/A1.X）に符号拡張されます。

乗算器出力は、A0レジスタやA1レジスタでだけではなく、データ・レジスタ・ファイル内のさまざまな16/32ビット・データ・レジスタにも保管できます。

▶ 乗算結果の丸めと飽和

積和演算では、アキュムレータ・データを飽和させることができ、オプションとしてレジスタやレジスタ・ハーフへの抽出のために丸めることもできます。乗算結果をレジスタやレジスタ・ハーフにだけ保管する場合、飽和と丸めも同様に機能します。丸めと飽和の演算は、次のように機能します。

- 丸めは小数結果にだけ適用されます。ただし、IHオプションの場合には、整数結果への上位ハーフ抽出と丸めに適用されます。

  IHオプションの場合、丸めた結果を取得するには、アキュムレータ（MACの場合）または乗算結果（乗算の場合）に0x8000を加算してから32ビットに飽和させます。詳細については、2-19ページの「乗算器の丸め結果」を参照してください。

- オーバーフローやアンダーフローが発生した場合には、飽和演算によって指定の結果レジスタには正または負の最大値が設定されます。詳細については、次の節を参照してください。

■ オーバーフロー時の乗算結果の飽和

ASTAT内の以下のビットは、乗算器のオーバーフロー・ステータスを示します。

- ビット16 (AV0) とビット18 (AV1) は、それぞれA0アキュムレータとA1アキュムレータのオーバーフロー条件（結果が32ビットをオーバーフローしたかどうか）を記録します。

  ビットがクリアされている (＝0) 場合には、オーバーフローやアンダーフローは発生していません。ビットがセットされている (＝1) 場合には、オーバーフローまたはアンダーフローが発生しました。AV0SビットとAV1Sビットはスティッキー・ビットです。

- 累算結果をレジスタに抽出する際にオーバーフローが発生した場合には、ビット24 (V) とビット25 (VS) がセットされます。

■ 乗算器命令のまとめ

表 2-10は乗算器命令を示します。アセンブリ言語構文と乗算器命令がステータス・フラグに与える影響の詳細については、『ADSP-BF53x Blackfin Processor Instruction Set Reference』を参照してください。

表 2-10では、以下の記号の意味に注意してください。

- Dregは、データ・レジスタ・ファイルの任意のレジスタを示します。
- Dreg_lo_hiは、データ・レジスタ・ファイルの任意のレジスタ内の任意の16ビット・レジスタ・ハーフを示します。
- Dreg_loは、データ・レジスタ・ファイルの任意のレジスタの下位16ビットを示します。
- Dreg_hiは、データ・レジスタ・ファイルの任意のレジスタの上位16ビットを示します。
- A*n*は、MACアキュムレータ・レジスタA0またはA1を示します。
- *は、命令の結果に応じて、フラグがセットまたはクリアされることを示します。
- – は、影響がないことを示します。

乗算器命令のオプションを2-43ページで説明します。

表 2-10. 乗算器命令のまとめ

命令	ASTAT のステータス・フラグ		
	AV0 AV0S	AV1 AV1S	V V_COPY VS
Dreg_lo = Dreg_lo_hi * Dreg_lo_hi ;	–	–	*
Dreg_hi = Dreg_lo_hi * Dreg_lo_hi ;	–	–	*
Dreg = Dreg_lo_hi * Dreg_lo_hi ;	–	–	*
A*n* = Dreg_lo_hi * Dreg_lo_hi ;	*	*	–
A*n* += Dreg_lo_hi * Dreg_lo_hi ;	*	*	–
An –= Dreg_lo_hi * Dreg_lo_hi ;	*	*	–
Dreg_lo = (A0 = Dreg_lo_hi * Dreg_lo_hi) ;	*	*	*

表 2-10. 乗算器命令のまとめ（続き）

命令	ASTAT のステータス・フラグ		
	AV0 AV0S	AV1 AV1S	V V_COPY VS
Dreg_lo = (A0 += Dreg_lo_hi * Dreg_lo_hi) ;	*	*	*
Dreg_lo = (A0 –= Dreg_lo_hi * Dreg_lo_hi) ;	*	*	*
Dreg_hi = (A1 = Dreg_lo_hi * Dreg_lo_hi) ;	*	*	*
Dreg_hi = (A1 += Dreg_lo_hi * Dreg_lo_hi) ;	*	*	*
Dreg_hi = (A1 –= Dreg_lo_hi * Dreg_lo_hi) ;	*	*	*
Dreg = (A*n* = Dreg_lo_hi * Dreg_lo_hi) ;	*	*	*
Dreg = (A*n* += Dreg_lo_hi * Dreg_lo_hi) ;	*	*	*
Dreg = (A*n* –= Dreg_lo_hi * Dreg_lo_hi) ;	*	*	*
Dreg *= Dreg ;	–	–	–

▶ 乗算器命令のオプション

以下に、乗算器命令のオプションの概要を示します。すべての命令ですべてのオプションを使用できるわけではありません。各命令でのこれらのオプションの使い方については、『ADSP-BF53x Blackfin Processor Instruction Set Reference』を参照してください。

無指定時 オプションなし。入力データは符号付き小数です。

(IS) 入力データ・オペランドは符号付き整数です。シフト補正は行われません。

(FU) 入力データ・オペランドは符号なし小数です。シフト補正は行われません。

(IU) 入力データ・オペランドは符号なし整数です。シフト補正は行われません。

(T)　入力データ・オペランドは符号付き小数です。デスティネーション・ハーフ・レジスタにコピーするとき、アキュムレータの内容の下位16ビットを切り捨てます。

(TFU)　入力データ・オペランドは符号なし小数です。デスティネーション・ハーフ・レジスタにコピーするとき、アキュムレータの内容の下位16ビットを切り捨てます。

(ISS2)　レジスタへの積和演算の場合：

入力データ・オペランドは符号付き整数です。デスティネーション・レジスタにコピーするとき、アキュムレータの内容はスケーリングされます（1桁の左シフトによりx2乗算）。スケーリングによって32ビットを超える符号付き値が得られた場合、数値はその正または負の最大値に飽和します。

ハーフ・レジスタへの積和演算の場合：

下位16ビットをデスティネーション・ハーフ・レジスタにコピーするとき、アキュムレータの内容はスケーリングされます。スケーリングによって16ビットを超える符号付き値が得られた場合、数値はその正または負の最大値に飽和します。

(IH)　このオプションは、上位ハーフ・ワード抽出による整数乗算を示します。アキュムレータは32ビットで飽和し、アキュムレータのビット[31:16]が丸められてから、デスティネーション・ハーフ・レジスタにコピーされます。

(W32)　入力データ・オペランドは、32ビットでアキュムレータ内に拡張ビットを持たない符号付き小数です。必要に応じて、積の左シフト補正が実行されます。このオプションは、32ビット・アキュムレータ用に記述された旧来のGSM音声ボコーダ・アルゴリズムに使用されます。このオプションの場合にのみ、特殊なケースとして0x8000 x 0x8000 = 0x7FFFが適用されます。

(M)　演算には混合乗算モードが使用されます。MAC1バージョンの命令にだけ有効です。左シフト補正なしで、符号付き小数に符号なし小数オペランドを乗算します。オペランド1は符号付き、オペランド2は符号なしです。MAC0は、デフォルトで符号付き小数の純粋乗算を実行するか、指定された別のフォーマットで実行します。つまり、MAC0は指定された符号付き／符号付きの乗算または符号なし／符号なしの乗算を実行します。(M)オプションは単独で使用したり、他の1つのフォーマット・オプションと組み合わせて使用できます。

■ 乗算器のデータ・フローの詳細

図 2-12は、レジスタ・ファイルとALU、さらに積和演算器を示します。

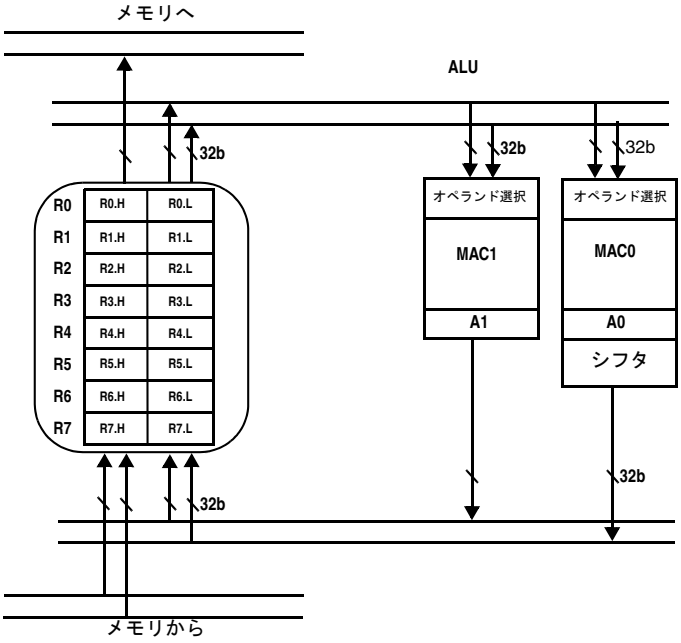


図 2-12. レジスタ・ファイルと ALU

各乗算器は2つの16ビット入力を持ち、16ビット乗算を実行して、結果を40ビット・アキュムレータに格納したり、16/32ビットのレジスタに抽出します。MAC入力では2つの32ビット・ワードを使用でき、選択可能な4つの16ビット・オペランドが提供されます。

オペランドの1つは、1つの32ビット・ワードの下位ハーフまたは上位ハーフから選択する必要があります。もう一方のオペランドは、もう一方の32ビット・ワードの下位ハーフまたは上位ハーフから選択する必要があります。したがって、各MACには4つの可能な入力オペランドの組合わせが供給されます。2つの32ビット・ワードは同じレジスタ情報を含むことができ、オプションによって、同じレジスタの上位ハーフと下位ハーフを2乗および乗算することができます。図 2-13は、これらの可能な組合わせを示します。

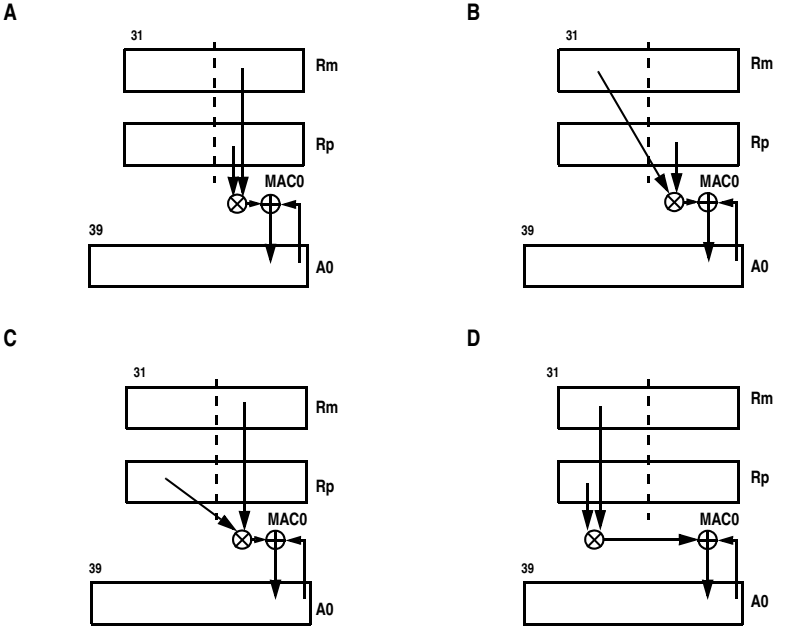


図 2-13. MAC 演算の 4 つの可能な組合わせ

32ビットの積は40ビットの加算器／減算器に渡されます。そこでは、新しい積をアキュムレータ結果レジスタの内容に対して加算／減算したり、または新しい積をデータ・レジスタ・ファイルの結果レジスタに直接渡したりできます。結果に関しては、A0レジスタとA1レジスタは40ビット幅です。これらの各レジスタは、さらに小さな32/8ビット・レジスタ（A0.W、A1.W、A0.X,、A1.X）から構成されます。

命令の例：

```
A0 = R3.L * R4.H ;
```

この命令は、MAC0積和演算器は乗算を実行して、その結果をアキュムレータ・レジスタに格納します。

```
A1 += R3.H * R4.H ;
```

この命令は、MAC1積和演算器は乗算を実行して、その結果をA1アキュムレータ内の以前の結果に累算します。

■ 累算なしの乗算

乗算器は、累算機能なしでも動作できます。累算が使用されない場合には、結果はデータ・レジスタ・ファイルのレジスタやアキュムレータ・レジスタに直接格納できます。デスティネーション・レジスタは、16ビットまたは32ビットとすることができます。16ビットのデスティネーション・レジスタが下位ハーフである場合には、MAC0が使用されます。これが上位ハーフである場合には、MAC1が使用されます。32ビットのデスティネーション・レジスタの場合には、MAC0またはMAC1が使用されます。

デスティネーション・レジスタが16ビットである場合には、乗算器から抽出されるワードは入力のデータ型に依存します。

- 乗算で小数オペランドまたはIHオプションを使用する場合には、結果の上位ハーフが抽出され、16ビットのデスティネーション・レジスタに格納されます（図 2-14を参照）。

- 乗算で整数オペランドを使用する場合には、結果の下位ハーフが抽出され、16ビットのデスティネーション・レジスタに格納されます。これらの抽出では、選択されたデータ型に対して結果として得られる16ビット・ワードで最も有益な情報を提供します（図 2-15を参照）。

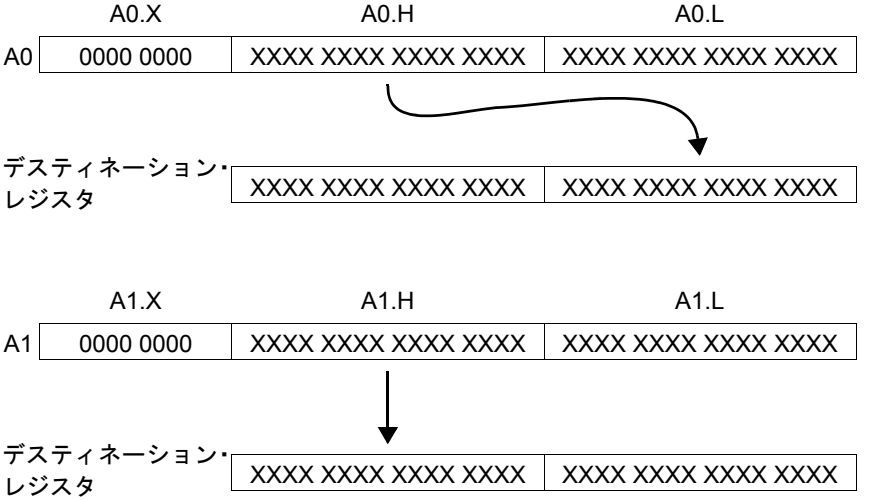


図 2-14. 小数オペランドの乗算

たとえば、次の命令では、小数の符号なしオペランドを使用します。

```
R0.L = R1.L * R2.L (FU) ;
```

この命令では、MAC0を使用して、乗算結果の上位16ビットを丸めと飽和付きでR0の下位ハーフに保管します。次の命令では、符号なし整数オペランドを使用します。

```
R0.H = R2.H * R3.H (IU) ;
```

この命令では、MAC1を使用して、乗算結果の下位16ビットを必要な飽和付きでR0の上位ハーフに保管します。

```
R0 = R1.L * R2.L ;
```

前の演算では、MAC0を使用して、オペランド・タイプとは無関係に乗算結果の32ビットを飽和付きでR0に保管します。

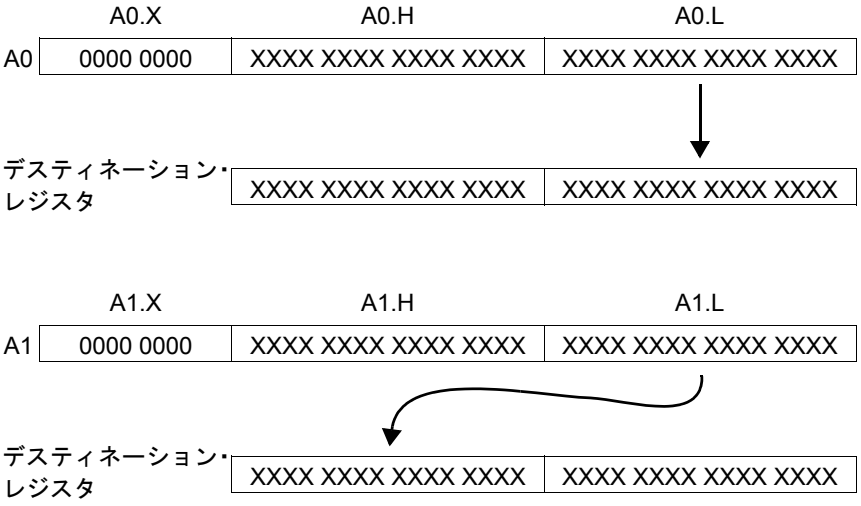


図 2-15. 整数オペランドの乗算

■ 特殊な 32 ビット整数 MAC 命令

このプロセッサは、32ビットのマルチサイクルMAC命令に対応します。

```
Dreg *= Dreg
```

この単一命令では、2つの32ビット整数オペランドを乗算して32ビットの整数結果を提供し、入力オペランドの1つを破壊します。

この命令の実行には複数のサイクルが必要です。また、命令の正確な動作については、製品データシートと『ADSP-BF53x Blackfin Processor Instruction Set Reference』を参照してください。このマクロ機能は割込み可能であり、アキュムレータ・レジスタA0およびA1のデータを変更しません。

■ デュアル MAC 演算

このプロセッサには2つの16ビットMACがあります。両方のMACを同じ演算で使用して、MACスループットを2倍にすることができます。各MACユニットには2本の同じ32ビット入力レジスタが提供されるため、それぞれに16ビット入力オペランドの4つの組合わせが提供されます。デュアルMAC演算は、ベクトル演算と呼ばれることがあります。その理由は、プログラムがサンプルのベクトルを4つの入力オペランドに格納して、ベクトル計算を実行できるからです。

デュアル積和演算命令の例は、次のとおりです。

```
A1 += R1.H * R2.L, A0 += R1.L * R2.H ;
```

この命令は、2つの積和演算を表します。

- 1つ目の演算 (MAC1) では、R1の上位ハーフにR2の下位ハーフが乗ぜられ、A1アキュムレータの内容に加算されます。
- 2番目の演算 (MAC0) では、R1の下位ハーフにR2の上位ハーフが乗ぜられ、A0の内容に加算されます。

MAC演算の結果は、複数の方法でレジスタに書込みできます。つまり、16ビット・ハーフのペアとして、32ビット・レジスタのペアとして、または独立した16ビット・ハーフ・レジスタまたは32ビット・レジスタとして書込みできます。

例：

```
R3.H = (A1 += R1.H * R2.L), R3.L = (A0 += R1.L * R2.L) ;
```

この命令では、40ビット・アキュムレータが16ビット・ハーフ・レジスタにパックされます。MAC1からの結果は、デスティネーション・レジスタの上位ハーフに転送する必要があります。MAC0からの結果は、同じデスティネーション・レジスタの下位ハーフに転送する必要があります。

アキュムレータから抽出して16ビット・デスティネーション・レジスタに保管する適正なビットは、オペランド・タイプによって決まります。2-47ページの「累算なしの乗算」を参照してください。

```
R3 = (A1 += R1.H * R2.L), R2 = (A0 += R1.L * R2.L) ;
```

この命令では、40ビット・アキュムレータは2本の32ビット・レジスタにパックされます。これらのレジスタは、レジスタ・ペア（`R[1:0]`、`R[3:2]`、`R[5:4]`、`R[7:6]`）であることが必要です。

```
R3.H = (A1 += R1.H * R2.L), A0 += R1.L * R2.L ;
```

この命令は、いずれか一方のアキュムレータだけがレジスタに転送される例です。デスティネーション・レジスタとしては、16ビットまたは32ビットのレジスタを指定できます。

バレル・シフタ（シフタ）

シフタは、16/32/40ビットの入力にビット単位のシフト機能を提供して、16/32/40ビットの出力をもたらします。これらの機能には、算術シフト、論理シフト、ローテート、さまざまなビット・テスト、セット、パック、

アンパック、および指数検出の機能が含まれます。これらのシフト機能を組み合わせれば、完全な浮動小数点形式などの数値フォーマット制御を実装できます。

■ シフタ演算

シフタ命令（>>>、>>、<<、ASHIFT、LSHIFT、ROT）は、ベースとなる演算条件に応じてさまざまな方法で使用できます。ASHIFT命令と>>>命令は、算術シフトを表します。LSHIFT命令、<<命令、>>命令は、論理シフトを表します。

算術シフト演算と論理シフト演算は、さらにサブセクションに分けられます。多くのDSPアルゴリズムで見られるような単独または対になった16ビット数値に作用するように意図された命令では、命令ASHIFTとLSHIFTを使用できます。一般に、これらは3オペランド命令です。

コンパイラでよく見られるような32ビットのレジスタ値に作用し、2つのオペランドを使用するように意図された命令では、>>>命令と>>命令を使用できます。

算術シフト命令、論理シフト命令、ローテート命令では、シフト引数をレジスタから取得したり、命令内の即値から直接に取得したりできます。シフタ関連の命令の詳細については、2-56ページの「シフタ命令のまとめ」を参照してください。

▶ 2オペランド・シフト

2オペランド・シフト命令は入力レジスタをシフトして、その結果を同じレジスタに保管します。

即値シフト

即値シフト命令は、入力ビット・パターンを指定のビット数だけ右（ダウン）または左（アップ）にシフトします。即値シフト命令は、命令自身に含まれるデータ値を使用して、シフト動作の量と方向を制御します。

次の例は、入力値がダウンシフトされる場合を示します。

```
R0 contains 0000 B6A3 ;
R0 >>= 0x04 ;
```

結果は次のようになります。

```
R0 contains 0000 0B6A ;
```

次の例は、入力値がアップシフトされる場合を示します。

```
R0 contains 0000 B6A3 ;
R0 <<= 0x04 ;
```

結果は次のようになります。

```
R0 contains 000B 6A30 ;
```

レジスタ・シフト

レジスタベースのシフトは、レジスタを使用してシフト値を保持します。シフト値を得るために32ビット・レジスタの全体が使用されます。シフトの大きさが32以上であるときは、結果は0または－1となります。

次の例は、入力値がアップシフトされる場合を示します。

```
R0 contains 0000 B6A3 ;
R2 contains 0000 0004 ;
R0 <<= R2 ;
```

結果は次のようになります。

```
R0 contains 000B 6A30 ;
```

▶ 3オペランド・シフト

3オペランド・シフタ命令は入力レジスタをシフトし、その結果をデスティネーション・レジスタに保管します。

即値シフト

即値シフト命令は、命令自身に含まれるデータ値を使用して、シフト動作の量と方向を制御します。

次の例は、入力値がダウンシフトされる場合を示します。

```
R0 contains 0000 B6A3 ;
R1 = R0 >> 0x04 ;
```

結果は次のようになります。

```
R1 contains 0000 0B6A ;
```

次の例は、入力値がアップシフトされる場合を示します。

```
R0.L contains B6A3 ;
R1.H = R0.L << 0x04 ;
```

結果は次のようになります。

```
R1.H contains 6A30 ;
```

レジスタ・シフト

レジスタベースのシフトは、レジスタを使用してシフト値を保持します。ASHIFT、LSHIFT、またはROTのシフト値を保持するためにレジスタが使用されるとき、シフト値は常にレジスタ(Rn.L)の下位ハーフにあります。Rn.Lの最下位6ビットはマスクされ、シフト値として使用されます。

次の例は、入力値がアップシフトされる場合を示します。

```
R0 contains 0000 B6A3 ;
R2.L contains 0004 ;
R1 = R0 ASHIFT by R2.L ;
```

結果は次のようになります。

```
R1 contains 000B 6A30 ;
```

次の例は、入力値がローテートされる場合を示します。条件コード(CC)ビットは0に設定されていると想定します。CCの詳細については、4-13ページの「条件コード・フラグ」を参照してください。

```
R0 contains ABCD EF12 ;
R2.L contains 0004 ;
R1 = R0 ROT by R2.L ;
```

結果は次のようになります。

```
R1 contains BCDE F125 ;
```

なお、CCビットは、結果のビット3に組み込まれています。

▶ ビットのテスト、セット、クリア、トグル

シフタは、データ・レジスタの特定ビットをテスト、セット、クリア、トグルする場合にも使われます。すべての命令には、ソース・レジスタとビット・フィールド値という2つの引数があります。テスト命令では、ソース・レジスタは変更されません。テスト命令の結果はCCビットにあります。

次の例は、さまざまな演算を示します。

```
ビットCLR ( R0, 6 ) ;
ビットSET ( R2, 9 ) ;
ビットTGL ( R3, 2 ) ;
CC = ビットTST ( R3, 0 ) ;
```

▶ フィールド抽出とフィールド置換

シフタが使用される場合、ソース・フィールドは、32ビットのデスティネーション・フィールドのどこにでも保管できます。ソース・フィールドの長さは1～16ビットとすることができます。さらに、32ビットのソース・フィールド内のどこからでも1～16ビット・フィールドを抽出できます。

これらの機能のために2つのレジスタ引数が使用されます。その一方には、32ビットのデスティネーションまたは32ビットのソースが保持されます。もう一方には、抽出／置換値、その長さ、ソース内でのその位置が保持されます。

■ シフタ命令のまとめ

表 2-11は、シフタ命令を示します。アセンブリ言語構文とシフタ命令がステータス・フラグに与える影響の詳細については『ADSP-BF53x Blackfin Processor Instruction Set Reference』を参照してください。

表 2-11では、以下の記号の意味に注意してください。

- Dregは、データ・レジスタ・ファイルの任意のレジスタを示します。
- Dreg_loは、データ・レジスタ・ファイルの任意のレジスタの下位16ビットを示します。
- Dreg_hiは、データ・レジスタ・ファイルの任意のレジスタの上位16ビットを示します。

- * は、命令の結果に応じて、フラグがセットまたはクリアされることを示します。
- * 0は、アキュムレータAV0に結果を送信してA0をセットまたはクリアする命令のバージョンを示します。
- * 1は、アキュムレータA1に結果を送信してAV1をセットまたはクリアする命令のバージョンを示します。
- ** は、フラグがクリアされることを示します。
- *** は、CCにはシフトインされた最新の値が含まれていることを示します。
- – は、影響がないことを示します。

表 2-11. シフタ命令のまとめ

命令	ASTAT のステータス・フラグ						
	AZ	AN	AC0 AC0_COPY AC1	AV0 AV0S	AV1 AV1S	CC	V V_COPY VS
BITCLR (Dreg, uimm5) ;	*	*	**	–	–	–	**/–
BITSET (Dreg, uimm5) ;	**	*	**	–	–	–	**/–
BITTGL (Dreg, uimm5) ;	*	*	**	–	–	–	**/–
CC = BITTST (Dreg, uimm5) ;	–	–	–	–	–	*	–
CC = !BITTST (Dreg, uimm5) ;	–	–	–	–	–	*	–
Dreg = DEPOSIT (Dreg, Dreg) ;	*	*	**	–	–	–	**/–
Dreg = EXTRACT (Dreg, Dreg) ;	*	*	**	–	–	–	**/–
BITMUX (Dreg, Dreg, A0) ;	–	–	–	–	–	–	–
Dreg_lo = ONES Dreg ;	–	–	–	–	–	–	–

表 2-11. シフタ命令のまとめ（続き）

命令	ASTAT のステータス・フラグ						
	AZ	AN	AC0 AC0_COPY AC1	AV0 AV0S	AV1 AV1S	CC	V V_COPY VS
Dreg = PACK (Dreg_lo_hi, Dreg_lo_hi);	–	–	–	–	–	–	–
Dreg >>>= uimm5 ;	*	*	–	–	–	–	**/–
Dreg >>= uimm5 ;	*	*	–	–	–	–	**/–
Dreg <<= uimm5 ;	*	*	–	–	–	–	**/–
Dreg = Dreg >>> uimm5 ;	*	*	–	–	–	–	**/–
Dreg = Dreg >> uimm5 ;	*	*	–	–	–	–	**/–
Dreg = Dreg << uimm5 ;	*	*	–	–	–	–	*
Dreg = Dreg >>> uimm4 (V) ;	*	*	–	–	–	–	**/–
Dreg = Dreg >> uimm4 (V) ;	*	*	–	–	–	–	**/–
Dreg = Dreg << uimm4 (V) ;	*	*	–	–	–	–	*
A*n* = A*n* >>> uimm5 ;	*	*	–	** 0/–	** 1/–	–	–
A*n* = A*n* >> uimm5 ;	*	*	–	** 0/–	** 1/–	–	–
A*n* = A*n* << uimm5 ;	*	*	–	* 0	* 1	–	–
Dreg_lo_hi = Dreg_lo_hi >>> uimm4 ;	*	*	–	–	–	–	**/–
Dreg_lo_hi = Dreg_lo_hi >> uimm4 ;	*	*	–	–	–	–	**/–
Dreg_lo_hi = Dreg_lo_hi << uimm4 ;	*	*	–	–	–	–	*
Dreg >>>= Dreg ;	*	*	–	–	–	–	**/–
Dreg >>= Dreg ;	*	*	–	–	–	–	**/–
Dreg <<= Dreg ;	*	*	–	–	–	–	**/–
Dreg = ASHIFT Dreg BY Dreg_lo ;	*	*	–	–	–	–	*

表 2-11. シフタ命令のまとめ（続き）

命令	ASTAT のステータス・フラグ						
	AZ	AN	AC0 AC0_COPY AC1	AV0 AV0S	AV1 AV1S	CC	V V_COPY VS
Dreg = LSHIFT Dreg BY Dreg_lo ;	*	*	–	–	–	–	**/–
Dreg = ROT Dreg BY imm6 ;	–	–	–	–	–	***	–
Dreg = ASHIFT Dreg BY Dreg_lo (V) ;	*	*	–	–	–	–	*
Dreg = LSHIFT Dreg BY Dreg_lo (V) ;	*	*	–	–	–	–	**/–
Dreg_lo_hi = ASHIFT Dreg_lo_hi BY Dreg_lo ;	*	*	–	–	–	–	*
Dreg_lo_hi = LSHIFT Dreg_lo_hi BY Dreg_lo ;	*	*	–	–	–	–	**/–
A*n* = A*n* ASHIFT BY Dreg _lo ;	*	*	–	* 0	* 1	–	–
A*n* = A*n* ROT BY imm6 ;	_	–	–	–	–	***	–
Preg = Preg >> 1 ;	–	–	–	–	–	–	–
Preg = Preg >> 2 ;	–	–	–	–	–	–	–
Preg = Preg << 1 ;	–	–	–	–	–	–	–
Preg = Preg << 2 ;	–	–	–	–	–	–	–
Dreg = (Dreg + Dreg) << 1 ;	*	*	*	–	–	–	*
Dreg = (Dreg + Dreg) << 2 ;	*	*	*	–	–	–	*
Preg = (Preg + Preg) << 1 ;	–	–	–	–	–	–	–
Preg = (Preg + Preg) << 2 ;	–	–	–	–	–	–	–
Preg = Preg + (Preg << 1) ;	–	–	–	–	–	–	–
Preg = Preg + (Preg << 2) ;	–	–	–	–	–	–	–

第3章　動作モードと状態

このプロセッサは、次の3つのプロセッサ・モードを提供します。

- ユーザ・モード
- スーパーバイザ・モード
- エミュレーション・モード

エミュレーション・モードとスーパーバイザ・モードは、コア・リソースに無制限にアクセスできます。ユーザ・モードは特定のシステム・リソースへのアクセスが制限されるため、ソフトウェア環境を保護することができます。

ユーザ・モードはアプリケーション・プログラム用の領域と見なされます。通常、スーパーバイザ・モードとエミュレーション・モードは、オペレーティング・システムのカーネル・コード用になっています。

プロセッサ・モードはイベント・コントローラによって決定されます。割込み、マスク不能割込み（NMI）、または例外を処理するとき、プロセッサはスーパーバイザ・モードです。エミュレーション・イベントを処理するとき、プロセッサはエミュレーション・モードです。イベントを処理しないときには、プロセッサはユーザ・モードです。

現在のプロセッサ・モードを識別するには、表3-1に示すように、IPENDメモリマップド・レジスタ（MMR）を調べます。

プロセッサがユーザ・モードにある間、MMRの読出しはできません。

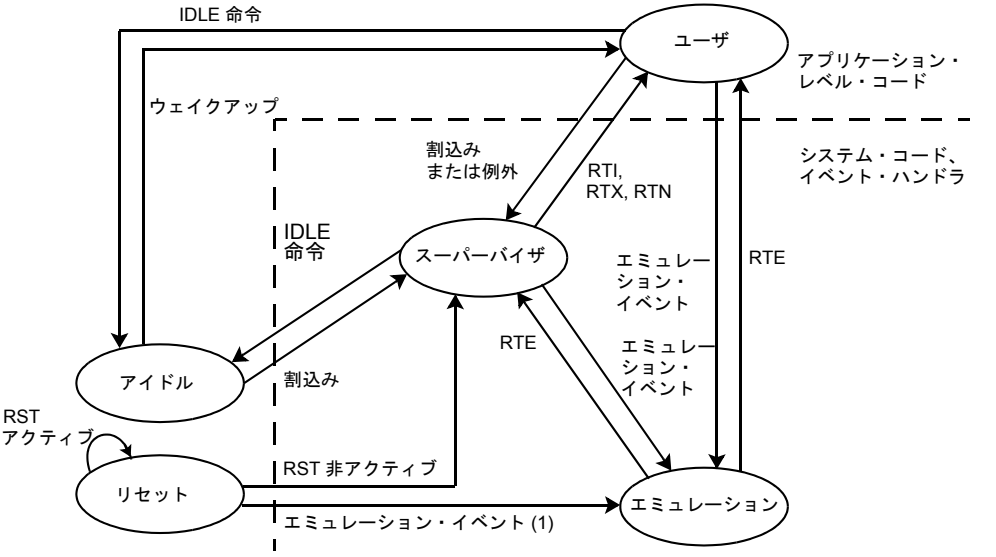
表 3-1.　現在のプロセッサ・モードの識別

イベント	モード	IPEND
割込み	スーパーバイザ	≥ 0x10 ただし、IPEND[0]、IPEND[1]、IPEND[2]、IPEND[3] = 0
例外	スーパーバイザ	≥ 0x08 IPEND[0] = 0、IPEND[1] = 0、IPEND[2] = 0、IPEND[3] = 1、かつ IPEND[15:4] がオール 0 またはオール 1 の場合、コアは例外イベントを処理しています。
NMI	スーパーバイザ	≥ 0x04 IPEND[0] = 0、IPEND[1] = 0、IPEND[2] = 1、かつ IPEND[15:2] がオール 0 またはオール 1 の場合、コアは NMI イベントを処理しています。
リセット	スーパーバイザ	= 0x02 リセット状態が終了すると、IPEND には 0x02 が設定され、リセット・ベクトルはスーパーバイザ・モードで動作します。
エミュレーション	エミュレータ	= 0x01 IPEND[0] = 1 の場合、残りのビット IPEND[15:1] の状態とは無関係に、プロセッサはエミュレーション・モードです。
なし	ユーザ	= 0x00

さらに、このプロセッサは、次の2つの処理用ではない状態も提供しています。

- アイドル状態
- リセット状態

図 3-1は、プロセッサのモードと状態に加えて、それらの間の遷移条件も示しています。



(1) 通常、リセットを終了するとスーパーバイザ・モードになります。しかし、エミュレーション・ハードウェアがリセットを開始した可能性もあります。その場合には、リセットを終了するとエミュレーション・モードになります。

図 3-1. プロセッサのモードと状態

ユーザ・モード

このプロセッサは、リセット状態やアイドル状態でないとき、および割込み、NMI、例外、またはエミュレーション・イベントを処理していないときにユーザ・モードにあります。ユーザ・モードは、システム・レジスタへの明示的アクセスを必要としないアプリケーション・レベル・コードの処理に使用されます。制限されたシステム・レジスタにアクセスを試みると例外イベントが発生します。表 3-2 は、ユーザ・モードでアクセスできるレジスタを示します。

表 3-2.　ユーザ・モードでアクセス可能なレジスタ

プロセッサ・レジスタ	レジスタ名
データ・レジスタ	R[7:0], A[1:0]
ポインタ・レジスタ	P[5:0], SP, FP, I[3:0], M[3:0], L[3:0], B[3:0]
シーケンサとステータス・レジスタ	RETS, LC[1:0], LT[1:0], LB[1:0], ASTAT, CYCLES, CYCLES2

■ 保護されたリソースと命令

システム・リソースはプロセッサ・レジスタのサブセット、すべてのMMR、保護された命令のサブセットから構成されます。これらのシステムMMRとコアMMRは、アドレス0xFFC0 0000で始まる位置に置かれています。このメモリ領域はユーザ・モードのアクセスから保護されます。ユーザ・モードでMMR空間にアクセスを試みると例外が発生します。

保護された命令の一覧を表 3-3に示します。ユーザ・モードから保護された命令の発行を試みると例外イベントが発生します。

表 3-3.　保護された命令

命令	説明
RTI	割込みからの復帰
RTX	例外からの復帰
RTN	NMI からの復帰
CLI	割込みをディスエーブル
STI	割込みをイネーブル
RAISE	割込み／リセットの強制
RTE	エミュレーションからの復帰 エミュレーション・モード以外で実行された場合にのみ、例外を発生

■ 保護されたメモリ

追加のメモリ位置はユーザ・モードのアクセスから保護できます。キャッシュ可能性保護索引バッファ（CPLB）のエントリを作成してイネーブルにできます。詳細については、6-51ページの「メモリ・マネジメント・ユニット」を参照してください。

■ ユーザ・モードへの入り方

リセット状態を抜け出すと、プロセッサはリセット・イベントを処理しているためにスーパーバイザ・モードになります。リセット状態からユーザ・モードに入るには、2つのステップを実行する必要があります。まず、RETI レジスタに復帰アドレスをロードする必要があります。次に、RTI を発行する必要があります。次のコード例は、リセット時にユーザ・モードに入る方法を示します。

▶ リセット時にユーザ・モードに入るためのコード例

リスト3-1は、リセットからユーザ・モードに入るためのコードを示します。

リスト 3-1. リセットからユーザ・モードに入る

```
P1.L = START ;    /* ユーザ・コードの先頭をポイント */
P1.H = START ;
RETI = P1 ;
RTI ;    /* リセット・イベントから復帰 */


START :    /* ここにユーザ・コードを記述 */
```

▶ ユーザ・モードへの復帰命令

表 3-4は、さまざまなプロセッサ・イベント・サービス・ルーチンからのユーザ・モードの呼出しに使用できる、復帰命令のまとめを示します。これらの命令がサービス・ルーチンで使用されるとき、最初に復帰アドレスの値を適切なイベントRETxレジスタに格納する必要があります。割込みルーチンの場合、サービス・ルーチンが割込み可能であれば復帰アドレスはスタックに格納されます。この場合、アドレスを得るには、スタックからRETIに値をポップします。RETIがロードされたら、RTI命令を発行できます。

なお、スタック・ポップはオプションです。RETI レジスタがプッシュ／ポップされない場合には、復帰アドレスがスタックに保存されないため、割込みサービス・ルーチンは割込み不可能になります。

このプロセッサは、以下のいずれかのイベントが発生するまで、ユーザ・モードにとどまります。

- 割込み、NMI、または例外イベントによるスーパーバイザ・モードの呼出し。
- エミュレーション・イベントによるエミュレーション・モードの呼出し。
- リセット・イベントによるリセット状態の呼出し。

表 3-4.　ユーザ・モードへの復帰命令

現在のプロセス動作	使用する復帰命令	実行再開のアドレスを持つレジスタ
割込みサービス・ルーチン	RTI	RETI
例外サービス・ルーチン	RTX	RETX
マスク不能割込みサービス・ルーチン	RTN	RETN
エミュレーション・サービス・ルーチン	RTE	RETE

スーパーバイザ・モード

このプロセッサは、スーパーバイザ・モードですべての割込み、NMI、例外イベントを処理します。

CPLBが設定されてイネーブルにされていない限り、スーパーバイザ・モードではエミュレーション・リソースを含めて、すべてのプロセッサ・システム・リソースに無制限にアクセスできます。詳細については、6-51ページの「メモリ・マネジメント・ユニット」を参照してください。レジスタ・エイリアスUSPはユーザ・スタック・ポインタを参照するものであり、スーパーバイザ・モードでのみ使用できます。このレジスタ・エイリアスが必要な理由は、スーパーバイザ・モードでのSPはユーザ・スタック・ポインタではなく、カーネル・スタック・ポインタを参照するからです。

リセット状態からはスーパーバイザ・モードで通常の処理が始まります。RESET信号をアサート解除すると、プロセッサはリセット状態からスーパーバイザ・モードに切り替わり、エミュレーション・イベントや復帰命令によってモードが変更されるまでスーパーバイザ・モードにとどまります。復帰命令が発行される前に、RETIレジスタに有効な復帰アドレスをロードする必要があります。

■ 非 OS 環境

非OS環境の場合には、アプリケーション・コードはすべてのコア・リソースとシステム・リソースにアクセスできるようにスーパーバイザ・モードにとどまる必要があります。RESETがアサート解除されると、プロセッサはリセット・イベントを処理することにより動作を開始します。エミュレーションはこの動作を乗っとることができる唯一のイベントです。したがって、優先順位の低いイベントは処理できません。

プロセッサをスーパーバイザ・モードに維持しながら、しかも優先順位の低いイベントの処理を可能にする方法の一つは、優先順位の最も低い割込み（IVG15）を設定および強制する方法です。イベントと割込みの詳細に

ついては、4-19ページの「イベントとシーケンス」を参照してください。RAISE 15命令を使用して優先順位の低い割込みを強制した後、RETIにはIVG15が発行されるまで実行できるユーザ・コードを指す復帰アドレスをロードできます。RETIがロードされた後、RTI命令を発行してリセット・イベントから復帰できます。

IVG15用の割込みハンドラは、アプリケーション・コードの開始アドレスにジャンプするように設定できます。追加のRTIは必要ありません。結果として、IPEND[15]がセットされたままであるため、プロセッサはスーパーバイザ・モードにとどまります。この時点では、プロセッサは優先順位の最も低い割込みを処理しています。これによって、優先順位の高い割込みの処理が保証されます。

▶ リセットを抜け出るスーパーバイザ・モードのコード例

リセット状態から抜け出したときにスーパーバイザ・モードにとどまるには、リスト 3-2に示すコードを使用します。

リスト 3-2. リセットを抜け出してスーパーバイザ・モードにとどまる

```
P0.L = LO(EVT15) ;    /* イベント・ベクトル・テーブル（EVT）のIVG15をポイント  */
P0.H = HI(EVT15) ;
P1.L = START ;    /* ユーザ・コードの先頭をポイント */


P1.H = START ;
[P0] = P1 ;    /* EVTのIVG15にスタート・コードのアドレスを置く */


P0.L = LO(IMASK) ;


R0 = [P0] ;
```

```
R1.L = EVT_IVG15 & 0xFFFF ;


R0 = R0 | R1 ;
[P0] = R0 ;   /* 割込みマスク・レジスタのIVG15ビットをセット(イネーブル) */


RAISE 15 ;   /* IVG15割込みの呼出し */
P0.L = WAIT_HERE ;
P0.H = WAIT_HERE ;
RETI = P0 ;   /* RETIに復帰アドレスをロード */
RTI ;   /* リセット・イベントからの復帰 */
WAIT_HERE :   /* IVG15割込みが処理されるまで、ここで待機 */


JUMP WAIT_HERE ;


START:   /* IVG15ベクトルをここに記述 */
[--SP] = RETI ;   /* 割込みをイネーブルにし、復帰アドレスをスタックに保存
*/
```

エミュレーション・モード

エミュレーション・モードがイネーブルにされており、次のいずれかの条件が満たされる場合には、プロセッサはエミュレーション・モードに入ります。

- 外部エミュレーション・イベントが発生した
- EMUEXCPT命令が発行された

このプロセッサは、エミュレーション・サービス・ルーチンがRTE命令を実行するまでエミュレーション・モードにとどまります。RTE命令の実行時にペンディング中の割込みがない場合には、プロセッサはユーザ・モードに切り替わります。そうでない場合は、プロセッサはスーパーバイザ・モードに切り替わって割込みを処理します。

エミュレーション・モードは、優先順位の最も高いモードです。そして、プロセッサは、すべてのシステム・リソースに無制限にアクセスできます。

アイドル状態

アイドル状態では、すべてのプロセッサ動作が停止します。この状態は、通常は動作が小康状態にあるときの消費電力を節約するためにユーザの判断で使用します。アイドル状態では処理は行われません。アイドル状態を呼び出すには、IDLEを含む一連の命令を使用します。IDLE命令は、アイドル状態が要求されたことをプロセッサ・ハードウェアに知らせます。SSYNC命令は、コアと外部システムにおける投機的な状態と過渡的な状態をすべてパージします。

プロセッサがアイドル状態を抜け出すのは、SPORTやリアルタイム・クロック（RTC）などのペリフェラルや外部デバイスが処理を必要とする割込みを生成したときです。

リスト 3-3では、コア割込みがディスエーブルにされ、IDLE命令が実行されます。ペンディング中のプロセスがすべて完了すると、コアはそのクロックをディスエーブルにします。割込みがディスエーブルにされるため、アイドル状態を終了させるにはWAKEUP信号をアサートするしか方法がありません。詳細については、4-27ページの「SIC_IWR レジスタ」を参照してください。（必須ではありませんが、割込みはWAKEUP信号と一緒にイネーブルにすることもできます。）

WAKEUP信号がアサートされると、プロセッサがウェイクアップし、STI命令が割込みを再びイネーブルにします。

■ アイドル状態に遷移するためのコード例

アイドル状態に遷移するには、リスト 3-3に示すコードを使用します。

リスト 3-3. アイドル状態への遷移

```
CLI R0 ;    /* 割込みをディスエーブル */
IDLE ;    /* パイプラインを空にし、コアをIDLE状態にする */
STI R0 ;    /* ウェイクアップ後に割込みを再びイネーブルにする */
```

リセット状態

リセット状態では、プロセッサ・ロジックを初期化します。リセット状態では、アプリケーション・プログラムとオペレーティング・システムは実行されません。また、クロックも停止します。

このプロセッサは、外付けロジックが外部$\overline{\text{RESET}}$信号をアサートする限り、リセット状態にとどまります。アサートが解除されると、プロセッサはリセット・シーケンスを完了し、スーパーバイザ・モードに切り替わって、リセット・イベント・ベクトル内のコードを実行します。

スーパーバイザ・モードまたはエミュレーション・モードのソフトウェアは、外部$\overline{\text{RESET}}$信号を必要とすることなく、リセット状態を呼び出すことができます。そのためには、RAISE命令のリセット版を発行します。

ユーザ・モードのアプリケーション・プログラムでは、オペレーティング・システム・カーネルによって提供されるシステム・コールを使用しない限り、リセット状態を呼び出すことができません。表 3-5に、リセット時のプロセッサの状態をまとめます。

表 3-5.　リセット時のプロセッサ状態

項目	リセット状態の説明
コア	
動作モード	リセット・イベントでのスーパーバイザ・モード、クロックは停止
丸めモード	バイアスのない丸め
サイクル・カウンタ	ディスエーブル、ゼロ
DAG レジスタ（I、L、B、M）	ランダム値（初期化時にクリアが必要）
データ・レジスタとアドレス・レジスタ	ランダム値（初期化時にクリアが必要）
IPEND、IMASK、ILAT	クリア、割込みはIPENDビット4でグローバルにディスエーブル
CPLB	ディスエーブル
L1 命令メモリ	SRAM（キャッシュ・ディスエーブル）
L1 データ・メモリ	SRAM（キャッシュ・ディスエーブル）
キャッシュ有効性ビット	無効
システム	
ブート方式	リセット時の BMODE ピンの値によって決定
MSEL クロック周波数	リセット値＝ 10
PLL バイパス・モード	ディスエーブル
VCO ／コア・クロック比	リセット値＝ 1
VCO ／システム・クロック比	リセット値＝ 5
ペリフェラル・クロック	ディスエーブル

システム・リセットとパワーアップ

表 3-6では、5種類のリセットについて説明します。なお、システム・ソフトウェア以外のすべてのリセットでコアをリセットします。

表 3-6.　リセット

リセット	ソース	結果
ハードウェア・リセット	RESET ピンはハードウェア・リセットを発生させます。	ダイナミック・パワー・マネジメント・コントローラ（DPMC）を含めて、コアとペリフェラルの両方をリセットします。 SYSCR のノー・ブート・オン・ソフトウェア・リセット・ビットをリセットします。詳細については、3-15 ページの「SYSCR レジスタ」を参照してください。
システム・ソフトウェア・リセット	アドレス 0xFFC0 0100 にあるシステム MMR SWRST のビット [2:0] に b#111 を書き込むと、システム・ソフトウェア・リセットが発生します。	RTC（リアルタイム・クロック）ブロックと大部分の DPMC を除いて、ペリフェラルだけをリセットします。DPMC は、SYSCR のノー・ブート・オン・ソフトウェア・リセット・ビットだけをリセットします。コアをリセットしません。ブート・シーケンスを開始しません。
ウォッチドッグ・タイマ・リセット	ウォッチドッグ・タイマを適切にプログラムすると、ウォッチドッグ・タイマ・リセットが発生します。	RTC ブロックと大部分の DPMC を除いて、コアとペリフェラルの両方をリセットします。 リセット・ソースがウォッチドッグ・タイマであったかどうかは、ソフトウェア・リセット・レジスタ（SWRST）の読出しによって判定できます。

表 3-6.　リセット（続き）

リセット	ソース	結果
コア・ダブル・フォルト・リセット	コアがダブル・フォルト状態に入っている場合、システム割込みコントローラ割込みマスク・レジスタ（SIC_IMASK）のコア・ダブル・フォルト・リセット・マスク・ビットをマスク解除することで、リセットを発生できます。	RTC ブロックと大部分の DPMC を除いて、コアとペリフェラルの両方をリセットします。 リセット・ソースがコア・ダブル・フォルトであったかどうかは、SWRST レジスタの読出しによって判定できます。
コアオンリー・ソフトウェア・リセット	このリセットを発生させるには、RAISE1 命令を実行するか、または JTAG ポートを通じてエミュレーション・ソフトウェアによってコアのデバッグ・コントロール・レジスタ（DBGCTL）のソフトウェア・リセット（SYSRST）ビットを設定します。DBGCTL レジスタは、メモリ・マップから見えません。	コアだけをリセットします。 ペリフェラルは、このリセットを認識しません。

■ ハードウェア・リセット

このプロセッサのチップ・リセットは、非同期のリセット・イベントです。ハードウェア・リセットを実行するには、$\overline{\text{RESET}}$ 入力ピンをアサート解除する必要があります。詳細については、『ADSP-BF531/ADSP-BF532/ADSP-BF533 Embedded Processor Data Sheet』を参照してください。

ハードウェアによるリセットでは、コアとペリフェラルの両方を含むシステム全体のリセットが行われます。$\overline{\text{RESET}}$ ピンがアサート解除された後、プロセッサはすべての非同期ペリフェラルがリセットを認識して完了したことを保証します。リセットの後、プロセッサは BMODE 状態によって設定されたブート・モード・シーケンスに移行します。

BMODE[1:0]ピンは専用のモード制御ピンです。これらのピンは、他の機能を共有しておらず、V_{DD}またはV_{SS}に直接接続することによって永続的に接続できます。ハードウェア・リセットやシステム・ソフトウェア・リセットの後で採用されるブート・モードは、これらのピンとSYSCRの対応するビットによって設定されます。詳細については、4-41ページの「リセット」と4-44ページの表4-11「例外の原因となるイベント」を参照してください。

■ SYSCR レジスタ

BMODE[1:0]ピンから検出された値は、$\overline{\text{RESET}}$ピンのアサート解除時にシステム・リセット設定レジスタ（SYSCR）にラッチされます。これらの値は、ハードウェア・リセット・シーケンスの後で、ソフトウェアによるアクセスと変更が可能になります。ソフトウェアでは、ノー・ブート・オン・ソフトウェア・リセット・ビットだけを変更できます。

SYSCRからは、さまざまな設定パラメータが適切なデスティネーションに配布されます（図3-2を参照）。

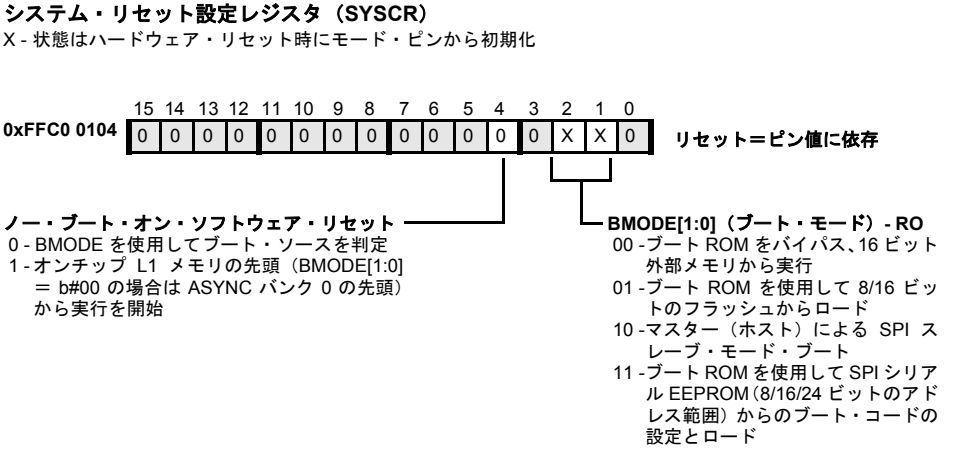


図 3-2.　システム・リセット設定レジスタ

■ ソフトウェア・リセットとウォッチドッグ・タイマ

ソフトウェア・リセットを開始するには、3つの方法があります。

- ウォッチドッグ・タイマによる方法（適切に設定された場合）
- ソフトウェア・リセット・レジスタのシステム・ソフトウェア・リセット・フィールドの設定による方法（3-18ページの図3-3を参照）
- RAISE 1命令による方法

ウォッチドッグ・タイマは、コアとペリフェラルの両方をリセットします。システム・ソフトウェア・リセットでは、コアのリセットやブート・シーケンスの実行なしにペリフェラルのリセットが行われます。

システム・ソフトウェア・リセットは、レベル1メモリ（キャッシュまたはSRAM）からの実行中に行う必要があります。

L1命令メモリがキャッシュとして設定されている場合には、システム・ソフトウェア・リセット・シーケンスがキャッシュに読み込まれていることを確認してください。

ウォッチドッグまたはシステム・ソフトウェア・リセットが開始された後、プロセッサはすべての非同期ペリフェラルがリセットを認識および完了していることを確認します。

ウォッチドッグ・タイマによって生成されたリセットの場合、プロセッサはブート・モード・シーケンスに移行します。ブート・モードは、BMODEとノー・ブート・オン・ソフトウェア・リセット制御ビットの状態によって設定されます。

SYSCRのノー・ブート・オン・ソフトウェア・リセット・ビットがクリアされている場合には、リセット・シーケンスはBMODE[1:0]制御ビットによって決定されます。

■ SWRST レジスタ

ソフトウェア・リセットを実行するには、ソフトウェア・リセット・レジスタ（SWRST）のシステム・ソフトウェア・リセット・フィールドを設定します。ビット15は、SWRSTが最後に読み出されて以降、ソフトウェア・リセットが発生したかどうかを示します。ビット14とビット13は、ソフトウェア・ウォッチドッグ・タイマまたはコア・ダブル・フォルトがソフトウェア・リセットを生成したかどうかを示します。ビット[15:13]は読出し専用であり、レジスタの読出し時にクリアされます。ビット[3:0]は読出し／書込みです。

BMODEピンがb#00に設定されず、SYSCRのノー・ブート・オン・ソフトウェア・リセット・ビットがセットされた場合、プロセッサはオンチップL1メモリの先頭から実行を開始します。この設定では、コアはオンチップL1メモリの先頭から命令の取出しを始めます。

BMODEピンがb#00に設定されている場合、コアはアドレス0x2000 0000（ASYNCバンク0の先頭）から命令の取出しを始めます。

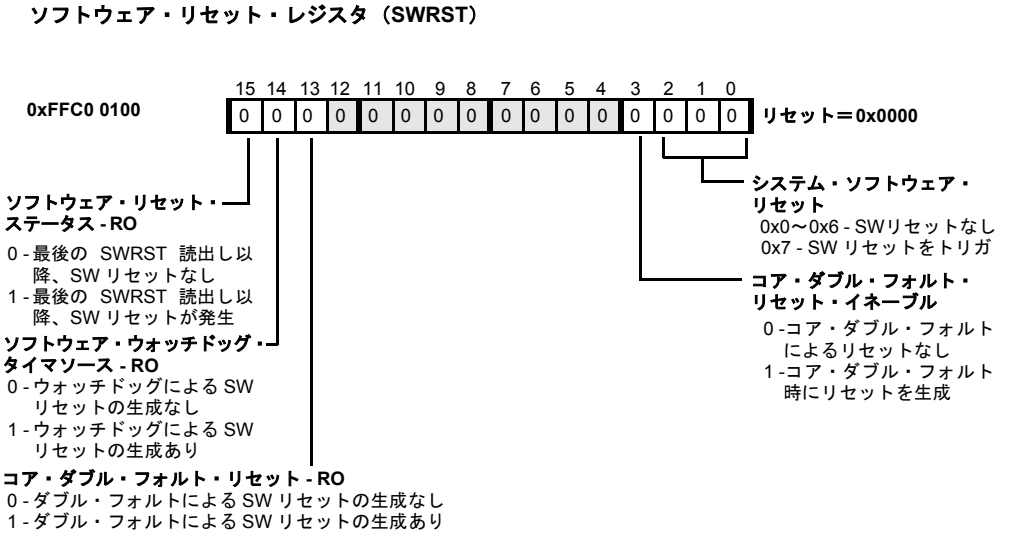


図 3-3.　ソフトウェア・リセット・レジスタ

■ コアオンリー・ソフトウェア・リセット

コアオンリー・ソフトウェア・リセットを実行するには、RAISE 1命令を実行するか、JTAGポートを通じてエミュレーション・ソフトウェアによりコアのデバッグ・コントロール・レジスタ（DBGCTL）のソフトウェア・リセット（SYSRST）ビットをセットします。（DBGCTLはメモリ・マップから見えません）。

コアオンリー・ソフトウェア・リセットは、コアの状態にだけ影響を与えます。なお、リセット期間中のシステム動作によっては、システム・リソースが未確定または不安定な状態になることもあります。

■ コアとシステムのリセット

システムとコアのリセットを行うには、リスト 3-4に示すコード・シーケンスを使用します。

リスト 3-4. コアとシステムのリセット

```
/* ソフト・リセットを発行 */
P0.L = LO(SWRST) ;
P0.H = HI(SWRST) ;
R0.L = 0x0007 ;
W[P0] = R0 ;
SSYNC ;

/* ソフト・リセットをクリア */
P0.L = LO(SWRST) ;
P0.H = HI(SWRST) ;
R0.L = 0x0000 ;
W[P0] = R0 ;
SSYNC ;

/* コア・リセット - 強制的なリブート */

RAISE 1 ;
```

ブート方式

内部ブートROMに含まれる小さなブート・カーネルは、バイパスしたり、外部メモリ・デバイスからのユーザ・コードのロードに使用できます。詳細については、4-41ページの表4-10「リセット・ベクトル・アドレス」を参照してください。ブート・カーネルはリセット時にBMODE[1:0]ピンの状態を読み出して、ダウンロード・ソースを識別します（4-25ページ

の表4-7を参照）。ブート・モード0では、プロセッサはアドレス0x2000 0000（ASYNCバンク0）にある16ビット幅の外部メモリから実行するように設定されています。

ユーザ・コードを外部メモリ・デバイスやホスト・デバイス（SPIスレーブ・モード・ブーティングのような場合）からロードするには、いくつかのブート方式があります。これらのモードでは、ブート・カーネルは`BMODE[1:0]`ピンの設定値に基づいて、選択されたペリフェラルを設定します。

メモリ・デバイスから読み出されたユーザ・コードは、ブート・モードごとにL1メモリの開始位置に置かれます。これ以外のセクションは、ローダー・ファイルのヘッダでの指定により、内部メモリに読み出されます。ブート・カーネルは、L1命令メモリ空間の先頭へのジャンプによってブート・プロセスを終了します。その後、プロセッサはこのアドレスから実行を開始します。

シリアル・ペリフェラル・インターフェース（SPI）からブートする場合には、汎用フラグ・ピン2がSPIチップ・セレクトとして使用されます。適正な動作のためには、このラインが接続される必要があります。

コアオンリー・ソフトウェア・リセットでは、コアからブートROMに制御が移されます。コアオンリー・ソフトウェア・リセットでは、コアだけがリセットされます。このリセットは、残りのシステムに影響を与えません。ブートROMカーネルは、ダウンロードの実行を回避するために、`SYSCR`のノー・ブート・オン・ソフトウェア・リセット条件を検出します。ソフトウェア・リセットでこのビットがセットされた場合には、プロセッサは通常のブート・シーケンスをスキップしてL1メモリの先頭にジャンプし、実行を開始します。

ブート・カーネルでは、フラッシュ・ブート・モード（`BMODE＝01`）に対して以下の条件を想定します。

- 非同期メモリ・バンク（AMB）0イネーブル

- AMB 0イネーブル用の16ビット・パッキング
- バンク0 RDYはアクティブ・ハイに設定
- バンク0のホールド・タイム（読出し／書込みのアサート解除～$\overline{\text{AOE}}$のアサート解除）＝3サイクル
- バンク0の読出し／書込みアクセス・タイム＝15サイクル

SPIマスタ・モード・ブート（BMODE＝11）の場合、ブート・カーネルでは、SPIボーレートは500kHzであると想定します。8/16/24ビット・アドレス指定のSPIシリアルEEPROMに対応します。SPIは、PF2出力ピンを使用して1つのSPI EEPROMデバイスを選択します。SPIコントローラは、8/16/24ビット・アドレス指定の有効なEEPROMが検出されるまで、アドレス0x00、0x0000、0x000000から連続した読出しコマンドを発行します。その後、L1命令メモリの先頭へのデータ・クロック入力を開始します。

SPIマスタ・モードのブーティング（BMODE＝11）には、MISOピンをハイレベルにプルする必要があります。

ブート・モードごとに、最初に外部メモリ・デバイスから10バイトのヘッダが読み出されます。ヘッダでは、転送されるバイト数とメモリ・デスティネーション・アドレスを指定します。すべてのブロックがロードされたら、L1命令SRAMの先頭からプログラム実行が始まります。

SPI スレーブ・モード・ブート（BMODE＝10）の場合には、図 3-4 に示すハードウェア設定が想定されます。

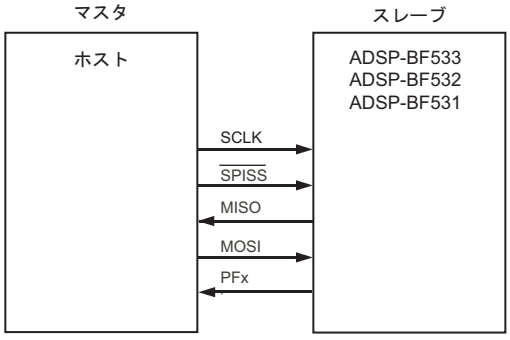


図 3-4.　SPI スレーブ・ブート・モード

ユーザ定義プログラマブル・フラグ PFx は、Blackfin プロセッサでの出力、およびホスト・デバイスでの入力です。このフラグによって、プロセッサはブート・プロセスの特定セクション中にホスト・デバイスによるデータ送信を阻止できます。このフラグがアサート解除されると、ホストはプロセッサへのバイト送信を続行できます。

第4章　プログラム・シーケンサ

このプロセッサでは、プログラム・シーケンサがプログラム・フローを制御することにより、プロセッサの他の部分で実行される次の命令のアドレスが絶えず提供されていきます。チップ内でのプログラム・フローは、ほとんどの場合にリニアであり、プロセッサはプログラム命令を連続的に実行しています。

プログラムが図 4-1に示すように非シーケンシャルなプログラム構造を使用する場合、命令の流れが変化します。非シーケンシャルな構造では、プロセッサは次の順次アドレスにない命令を実行するように指示されます。このような構造には、以下の要素が含まれます。

- **ループ**：1つの命令シーケンスが、ゼロ・オーバーヘッドで複数回実行されます。
- **サブルーチン**：　プロセッサは別のメモリ部分にある命令を実行するために、シーケンシャル・フローを一時的に中断します。
- **ジャンプ**：プログラム・フローは、別のメモリ部分に永久的に移されます。
- **割込みと例外**：ランタイム・イベントまたは命令が、サブルーチンの実行をトリガします。
- **アイドル**：命令によってプロセッサは動作を停止し、割込みが発生するまでその現状を保持します。その後、プロセッサは割込みを処理して、通常の実行を継続します。

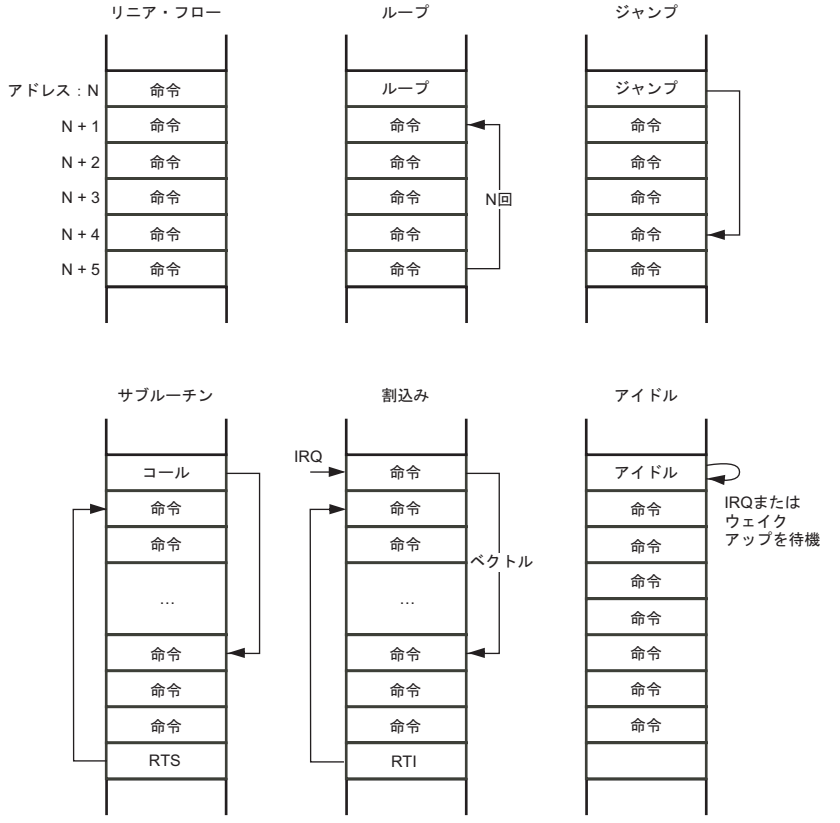


図 4-1. プログラム・フローの種類

シーケンサは、次に実行する命令のアドレスを選択することで、これらのプログラム構造の実行を管理します。

フェッチされたアドレスは命令パイプラインに入り、プログラム・カウンタ（PC）で終わります。パイプラインには、現在のフェッチ、デコード、および実行されている命令の32ビットアドレスが含まれています。PCは、

復帰アドレスを格納するRETnレジスタとペアになります。シーケンサによって生成されるすべてのアドレスは、32ビットのメモリ命令アドレスです。

イベントを管理するため、シーケンサのイベント・コントローラは割込みとイベントの処理を扱い、割込みがマスクされているかどうかを判定し、適切なイベント・ベクトル・アドレスを生成します。

データ・アドレス・ジェネレータ（DAG）は、データ・アドレスの提供に加えて、シーケンサの間接分岐用の命令アドレスも提供できます。

シーケンサは、条件付き命令とループ終了条件を評価します。ループ・レジスタは、ネストされたループを提供します。メモリマップド・レジスタ（MMR）には、割込みサービス・ルーチンの実装に使用される情報が格納されます。

シーケンサ関連のレジスタ

表 4-1は、シーケンサに関連するプロセッサ内のレジスタを示します。PCレジスタとSEQSTATレジスタを除いて、シーケンサ関連のすべてのレジスタは直接に読出し／書込みが可能です。スタックとの間でレジスタを手動でプッシュ／ポップするには、次の命令を明示的に使用します。

- `[--SP] = Rn`（プッシュ用）
- `Rn = [SP++]`（ポップ用）

表 4-1.　シーケンサ関連のレジスタ

レジスタ名	説明
SEQSTAT	シーケンサ・ステータス・レジスタ
RETX RETN RETI RETE RETS	復帰アドレス・レジスタ：4-19 ページの「イベントとシーケンス」を参照 例外復帰 NMI 復帰 割込み復帰 エミュレーション復帰 サブルーチン復帰
LC0, LC1 LT0, LT1 LB0, LB1	ゼロオーバーヘッド・ループ・レジスタ： ループ・カウンタ ループ先頭 ループ末尾
FP, SP	フレーム・ポインタとスタック・ポインタ：5-5 ページの「フレーム・ポインタとスタック・ポインタ」を参照
SYSCFG	システム設定レジスタ
CYCLES, CYCLES2	サイクル・カウンタ：19-28 ページの「CYCLES および CYCLES2 レジスタ」を参照
PC	プログラム・カウンタ

■ SEQSTAT レジスタ

シーケンサ・ステータス・レジスタ（SEQSTAT）には、シーケンサの現在の状態についての情報だけでなく、前回のイベントからの診断情報も含まれています。SEQSTATは読出し専用レジスタであり、スーパーバイザ・モードでのみアクセスできます。

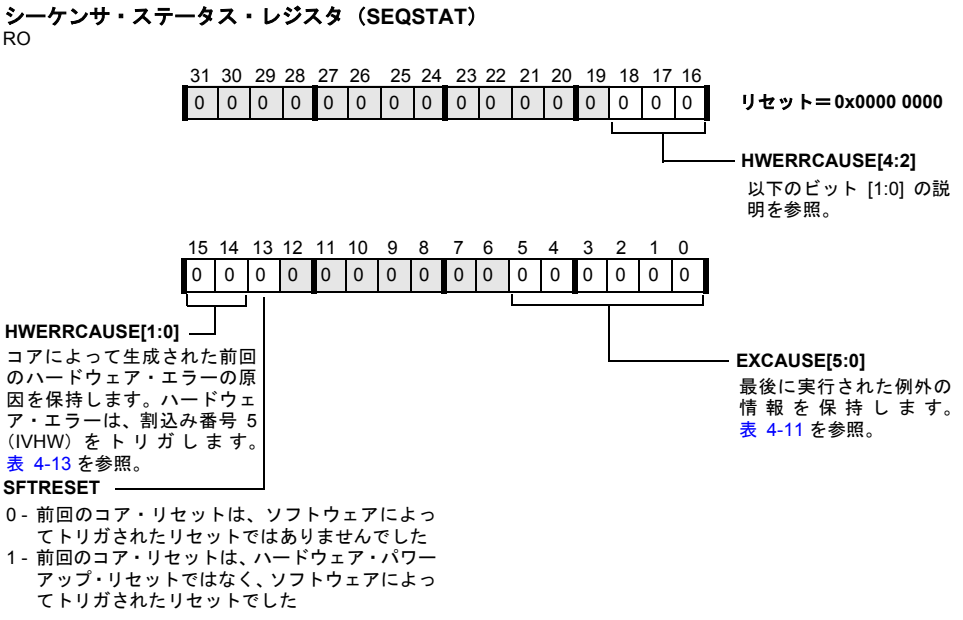


図 4-2.　シーケンサ・ステータス・レジスタ

■ ゼロオーバーヘッド・ループ・レジスタ（LC、LT、LB）

ループを実装するには、ソフトウェア命令の代わりに、2組のゼロオーバーヘッド・ループ・レジスタでハードウェア・カウンタを使用してループ条件を評価します。評価後に、処理は新しいターゲット・アドレスに分岐します。2組のレジスタには、ループ・カウンタ（LC）、ループ・トップ（LT）、ループ・ボトム（LB）の各レジスタが組み込まれています。

表 4-2には、32 ビットのループ・レジスタ・セットを示します。

表 4-2.　ループ・レジスタ

レジスタ	説明	機能
LC0, LC1	ループ・カウンタ	ループの残りの繰返し回数を維持します
LT0, LT1	ループ・トップ	ループ内の最初の命令のアドレスを保持します
LB0, LB1	ループ・ボトム	ループの最後の命令のアドレスを保持します

■ SYSCFG レジスタ

システム設定レジスタ（SYSCFG）は、プロセッサの設定を制御します。このレジスタは、スーパーバイザ・モードからのみアクセスできます。

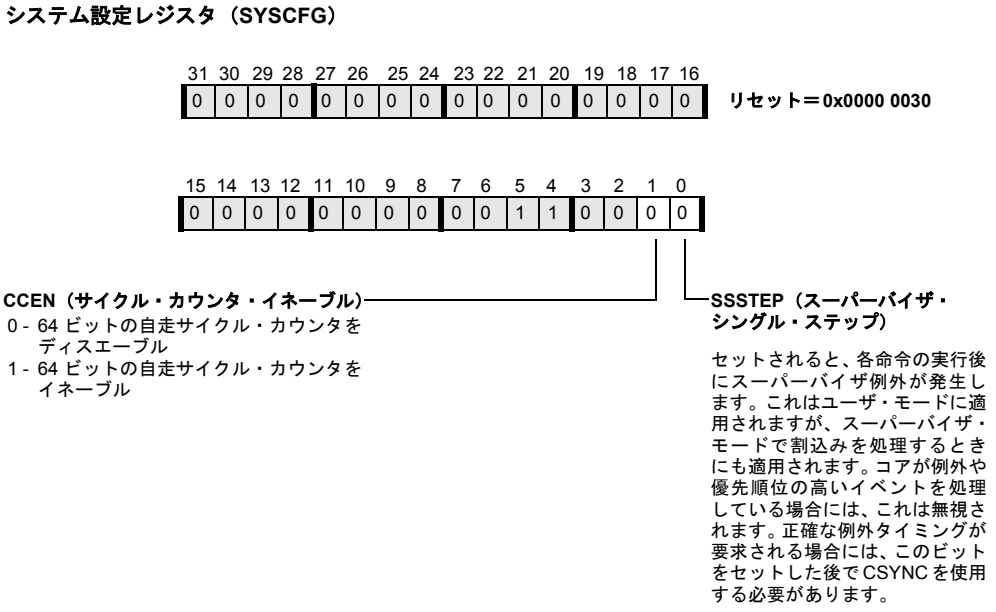


図 4-3.　システム設定レジスタ

命令パイプライン

プログラム・シーケンサは、実行中の現在の命令とプロセッサの現在の状態を調べることで次の命令アドレスを決定します。特殊な条件がない限り、プロセッサは先読みアドレスをインクリメントすることで、メモリから順番に命令を実行します。

表 4-3に示すように、プロセッサには10段の命令パイプラインがあります。

表 4-3.　命令パイプラインの段

パイプライン段	説明
命令フェッチ 1（IF1）	命令メモリのアクセスを開始
命令フェッチ 2（IF2）	中間メモリ・パイプライン
命令フェッチ 3（IF3）	L1 命令メモリのアクセスを完了
命令デコード（DEC）	命令を整列、命令デコードを開始し、ポインタ・レジスタ・ファイルにアクセス
アドレス計算（AC）	データ・アドレスと分岐ターゲット・アドレスを計算
実行 1（EX1）	データ・メモリのアクセスを開始
実行 2（EX2）	レジスタ・ファイルの読出し
実行 3（EX3）	データ・メモリのアクセスを完了し、デュアルサイクル命令の実行を開始
実行 4（EX4）	1 サイクル命令を実行
ライト・バック（WB）	データ・レジスタ・ファイルとポインタ・レジスタ・ファイルに状態を書き込み、イベントを処理

図 4-4はパイプラインの図を示します。

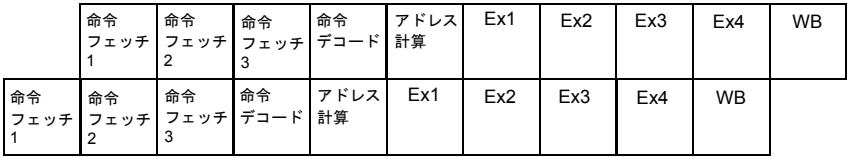


図 4-4. プロセッサのパイプライン

シーケンサは演算をデコードし、命令メモリ・ユニットと命令整列ユニットに配布します。また、パイプライン内での命令のストールと無効化も制御します。シーケンサは、パイプラインが完全にインターロックされることを保証します。したがって、プログラマはパイプラインを管理する必要がありません。

命令のフェッチ／分岐ロジックによって、命令メモリ・ユニット用の32ビット・フェッチ・アドレスが生成されます。命令整列ユニットでは、命令デコード段の先頭で命令とその幅情報を返します。

命令タイプ（16/32/64ビット）ごとに、命令整列ユニットでは、整列バッファに有効な命令が十分にあってすべてのサイクルで命令を提供できることを保証します。命令は16/32/64ビット幅とすることができるため、命令整列ユニットでは、すべてのサイクルでキャッシュから命令をフェッチする必要はありません。たとえば、一連の16ビット命令の場合、命令整列ユニットでは、4サイクルに1回だけ命令メモリ・ユニットから命令を取得します。整列ロジックでは、整列バッファのステータスに基づいて次の命令アドレスを要求します。フローの変更がない場合には、シーケンサは次のサイクルで次のフェッチ・アドレスを生成することで応答します。

シーケンサは、整列ロジックから要求を受け取ったり、フローの変更が行われたりするまではフェッチ・アドレスを保持します。シーケンサは、常に前のフェッチ・アドレスを8（次の8バイト）だけインクリメントしま

す。分岐や割込みなど、フローの変更が行われた場合には、シーケンサはその変更を命令メモリ・ユニットに伝達します。これによって、命令整列ユニット内のデータが無効にされます。

実行ユニットには、2つの16ビット乗算器、2つの40ビットALU、2つの40ビット・アキュムレータ、1つの40ビット・シフタ、ビデオ・ユニット(8ビットのALUサポートを追加)、8エントリの32ビット・データ・レジスタ・ファイルが含まれています。

レジスタ・ファイルの読出しは、EX2パイプライン段で行われます（オペランドの場合)。書込みはWB段で行われます（ストアの場合)。乗算器とビデオ・ユニットはEX3段でアクティブであり、ALUとシフタはEX4段でアクティブです。アキュムレータは、EX4段の最後で書き込まれます。

非シーケンシャルなプログラム・フローがあると、プロセッサの命令スループットは減少する可能性があります。非シーケンシャルなプログラム動作には、以下の動作が含まれます。

- ジャンプ
- サブルーチン・コールと復帰
- 割込みと復帰
- ループ

分岐とシーケンス

分岐とは、シーケンサが提供する非シーケンシャルなプログラム・フローの1つです。分岐が発生するのは、次の順次アドレス以外の新しい位置で、JUMP命令やCALL命令が実行を始めるときです。JUMP命令とCALL命令の使い方については、『ADSP-BF53x Blackfin Processor Instruction Set Reference』を参照してください。以下に簡単に説明します。

- JUMPまたはCALL命令は、プログラム・フローを別のメモリ位置に移します。JUMPとCALLとの違いとして、CALLでは、復帰アドレスをRETSレジスタに自動的にロードします。復帰アドレスは、CALL命令の後にある次の順次アドレスです。このプッシュによって、このアドレスはCALL命令の対応する復帰命令に使用できるようになり、サブルーチンからの容易な復帰を可能にします。

- 復帰命令によって、シーケンサはRETSレジスタに格納されている復帰アドレスから命令をフェッチします（サブルーチン復帰の場合）。復帰命令には、サブルーチンからの復帰（RTS）、割込みからの復帰（RTI）、例外からの復帰（RTX）、エミュレーションからの復帰（RTE）、マスク不能割込みからの復帰（RTN）のタイプが含まれます。各復帰タイプには、復帰アドレスを保持するための専用のレジスタがあります。

- JUMP命令は、ASTATレジスタのCCビットのステータスに応じて、条件付きにすることができます。これらの命令は即値命令であり、遅延がないこともあります。プログラム・シーケンサは、CCステータス・ビットを評価して、分岐を実行するかどうかを決定できます。条件が指定されていない場合には、分岐は常に実行されます。

- 条件付きJUMP命令では、パイプラインの長さに起因する分岐遅延を減らすために、静的分岐予測を使用します。

分岐は、直接または間接とすることができます。直接分岐アドレスは、命令ワードによってのみ決定されます（たとえば、JUMP 0x30）。一方、間接分岐では、DAGレジスタの内容からそのアドレスを取得します（たとえば、JUMP(P3)）。

あらゆるタイプのJUMPとCALLは、PC相対とすることができます。間接JUMPとCALLは、絶対またはPC相対とすることができます。

■ 直接のショート・ジャンプとロング・ジャンプ

シーケンサは、ショート・ジャンプとロング・ジャンプの両方を提供します。分岐のターゲットは、命令の位置からのPC相対アドレス＋オフセットです。ショート・ジャンプのPC相対オフセットは13ビットの即値であり、2の倍数（ビット0＝0）であることが必要です。この13ビット値は、－4096～＋4094バイトという実効ダイナミック・レンジを提供します。

ロング・ジャンプのPC相対オフセットは25ビットの即値であり、これも2の倍数（ビット0＝0）であることが必要です。この25ビット値は、－16,777,216～＋16,777,214バイトという実効ダイナミック・レンジを提供します。

プログラムの作成時に、デスティネーションは現在のPC値から13ビットより小さなオフセットであることが判明している場合には、JUMP.S 0xnnnn 命令を使用できます。デスティネーションが13ビットより大きなオフセットを必要とする場合には、JUMP.L 0xnnnnnnn命令を使用する必要があります。デスティネーションのオフセットが未知であり、開発ツールがオフセットを見積もる必要がある場合には、JUMP 0xnnnnnnn命令を使用してください。逆アセンブル時には、この命令は適切なJUMP.S命令やJUMP.L命令によって置き換えられます。

■ ダイレクト・コール

CALL命令は分岐命令の1つであり、CALL命令が実行されなかった場合に次に実行するはずの命令のアドレスをRETSレジスタにコピーします。直接CALL命令には25ビットのPC相対オフセットがあり、2の倍数（ビット0＝0）であることが必要です。この25ビット値は、－16,777,216～＋16,777,214バイトの実効ダイナミック・レンジを提供します。

■ 間接分岐とコール

間接JUMP命令とCALL命令は、デスティネーション・アドレスをデータ・アドレス・ジェネレータ（DAG）のPレジスタから取得します。CALL命令の場合、RETSレジスタには、CALL命令がない場合に次に実行するはずの命令のアドレスがロードされます。

例：

```
JUMP (P3) ;
CALL (P0) ;
```

■ PC 相対の間接分岐とコール

PC相対の間接JUMP命令とCALL命令は、分岐ターゲットへのオフセットとしてPレジスタの内容を使用します。CALL命令の場合、RETSレジスタには、CALL命令が実行されなかった場合に次に実行するはずの命令のアドレスがロードされます。

例：

```
JUMP (PC + P3) ;
CALL (PC + P0) ;
```

■ 条件コード・フラグ

このプロセッサには、条件コード（CC）フラグ・ビットがあります。このビットは、分岐の方向を解決するために使用されます。このフラグには、8つの方法でアクセスできます。

- 条件付き分岐はCC内の値によって解決されます。
- データ・レジスタ値はCCにコピーでき、CC内の値はデータ・レジスタにコピーできます。
- ビットTST命令ではCCフラグにアクセスします。
- ステータス・フラグはCCにコピーでき、CC内の値はステータス・フラグにコピーできます。
- CCフラグ・ビットには、ポインタ・レジスタ比較の結果を設定できます。
- CCフラグ・ビットには、データ・レジスタ比較の結果を設定できます。
- いくつかのシフタ命令（ローテートやBXOR）では、シフト・オペランド／結果の一部としてCCを使用します。
- テスト・アンド・セット命令では、CCビットのセット／クリアができます。

プログラム・フローの制御には、これらの8つの方法でCCビットにアクセスします。分岐は、演算フラグを設定する命令とは明示的に分離されています。CCの値の解釈を指定する命令エンコーディングには1つのビットが存在します。この解釈は「真で分岐」または「偽で分岐」です。

比較演算の書式はCC = exprです。ここで*expr*は、同じタイプのレジスタのペアを必要とします（たとえば、データ・レジスタやポインタ・レジスタ、または単一レジスタと小さな即値定数）。この小さな即値定数は、符号付き比較では3ビット（－4～3）の符号付き数値であり、符号なし比較では3ビット（0～7）の符号なし数値です。

CCの値は、等しい（==）、小さい（<）、小さいか等しい（<=）といった演算子によって決まります。また、32ビットRレジスタ内のビットがセットされているかどうかをテストする、ビット・テスト演算もあります。

▶ 条件付き分岐

シーケンサは条件付き分岐に対応しています。条件付き分岐はJUMP命令であり、その実行は、CCビットの値に応じて分岐したり、直線的に続行されます。分岐のターゲットは、命令の位置からのPC相対アドレス＋オフセットです。このPC相対オフセットは11ビットの即値であり、2の倍数（ビット0＝0）であることが必要です。この11ビット値は、－1024～＋1022バイトの実効ダイナミック・レンジを提供します。

たとえば、次の命令ではCCフラグをテストし、正である場合には、ラベルdest_addressによって識別される位置にジャンプします。

```
IF CC JUMP dest_address ;
```

▶ 条件付きレジスタ間移動

レジスタ間移動は、CCフラグの値が真であるか偽であるか（1または0）に応じて実行できます。場合によっては、分岐の代わりにこの命令を使用すると、分岐に起因するサイクル損失を解消できます。このような条件付き移動は、SPとFPを含めて任意のRレジスタ間またはPレジスタ間で可能です。

コード例：

```
IF CC R0 = P0 ;
```

■ 分岐予測

シーケンサは条件付き分岐の実行速度を上げるために、静的分岐予測を行います。これらの分岐は、CCビットの状態に基づいて実行されます。

EX4段では、シーケンサは実際のCCビット値を予測値と比較します。値の予測に誤りがあった場合には分岐が修正され、パイプラインのWB段では正しいアドレスを使用できます。

条件付き分岐の分岐遅延は次のとおりです。

- 予測が「分岐しない」であり、実際に分岐しなかった場合：0 CCLKサイクル。
- 予測が「分岐しない」であり、実際に分岐した場合：8 CCLKサイクル。
- 予測が「分岐する」であり、実際に分岐した場合：4 CCLKサイクル。
- 予測が「分岐する」であり、実際に分岐しなかった場合：8 CCLKサイクル。

すべての無条件分岐で、パイプラインのAC段で計算された分岐ターゲット・アドレスは、EX1段の先頭で命令フェッチ・アドレス・バスに送られます。すべての無条件分岐には、4 CCLKサイクルの遅延があります。

表 4-4の例を参照してください。

表 4-4.　分岐予測

命令	説明
If CC JUMP dest (bp)	この命令では CC フラグをテストし、フラグがセットされている場合には、ラベル dest によって識別される位置にジャンプします。 CC フラグがセットされている場合には、分岐は正しく予測され、分岐遅延は減少します。そうでない場合には、分岐の予測に誤りがあり、分岐遅延は増大します。

ループとシーケンス

シーケンサには、ゼロオーバーヘッド・ループのメカニズムがあります。シーケンサに内蔵される2つのループ・ユニットには、それぞれ3本のレジスタが含まれています。各ループ・ユニットには、ループ・トップ・レジスタ（LT0、LT1）、ループ・ボトム・レジスタ（LB0、LB1）、ループ・カウント・レジスタ（LC0、LC1）があります。

アドレスxにある命令が実行され、xがLB0の内容と一致する場合には、次に実行される命令はLT0内のアドレスからとなります。つまり、PC == LB0の場合、LT0への暗黙的ジャンプが実行されます。

ループバックが発生するのは、カウントが2以上であるときだけです。カウントが非ゼロである場合には、カウントは1だけデクリメントされます。たとえば、2つの繰返しを持つループを想定します。最初に、カウントは2です。最初のループ終端に到達すると、カウントは1にデクリメントされ、2回目の実行のためにプログラム・フローはループの先頭に戻ります。再びループ終端に到達すると、カウントは0にデクリメントされますが、ループの本体はすでに2回実行されたので、ループバックは発生しません。

2つのループ・ユニットがあるため、ループ・ユニット1には、ネストされたループ構造において内側ループとして使用できるように高い優先順位が割り当てられます。つまり、特定の命令（PC == LB1, LC1 >= 2）でのループ・ユニット1によるループバックは、たとえアドレスが一致した場合でも、同じ命令でのループ・ユニット0のループバックを防止します。ループ・ユニット0がループ・バックを許可されるのは、ループ・カウント1が空になった後だけです。

LSETUP命令を使用すれば、ループ・ユニットの3本のレジスタを一度にすべてロードできます。各ループ・レジスタは、レジスタ転送によって個々にロードすることもできます。しかし、転送の時点でループ・カウントがゼロ以外（ループがアクティブ）である場合には、かなりのオーバーヘッドが発生します。

次のコード例に示すループは2つの命令を含み、32回の繰返しを行います。

リスト 4-1. ループ例

```
P5 = 0x20 ;
LSETUP ( lp_start, lp_end ) LC0 = P5 ;
lp_start:
R5 = R0 + R1(ns) || R2 = [P2++] || R3 = [I1++] ;


lp_end:  R5 = R5 + R2 ;
```

2つのネストされたループを管理するために、2組のループ・レジスタが使用されます。

- LC[1:0] – ループ・カウント・レジスタ
- LT[1:0] – ループ・トップ・アドレス・レジスタ
- LB[1:0] – ループ・ボトム・アドレス・レジスタ

LSETUP命令を実行するとき、プログラム・シーケンサはループの最後の命令のアドレスをLBxにロードし、ループの最初の命令のアドレスをLTxにロードします。ループの先頭アドレスと末尾アドレスは、LSETUP命令からのPC相対アドレス＋オフセットとして計算されます。いずれの場合にも、LSETUP命令の位置にオフセット値が加算されます。

LC0レジスタとLC1レジスタは符号なし32ビット・レジスタであり、それぞれループによる$2^{32}-1$回の繰返しが可能です。

LCx = 0の場合、ループは無効であり、シングル・パスのコードが実行されます。

表 4-5.　ループ・レジスタ

ループの先頭／最終アドレス	ループ開始アドレスの計算に使用する PC 相対オフセット	ループ開始命令の有効範囲
トップ／先頭	5 ビットの符号付き即値、2 の倍数であることが必要	LSETUP 命令から 0 ～ 30 バイトの距離
ボトム／最終	11 ビットの符号付き即値、2 の倍数であることが必要	LSETUP 命令から 0 ～ 2046 バイトの距離（定義されるループ長は 2046 バイトとすることができる）

このプロセッサは4位置の命令ループ・バッファに対応しており、ループ内での命令フェッチを減らします。ループ・コードに含まれる命令の数が4つ以下である場合には、命令がローカルに格納されるため、ループの繰返し数とは無関係に命令メモリへのフェッチは不要です。ループ・バッファは、ループ・バッファ内の命令の実行中にフェッチを可能にすることで、5つ以上の命令を持つループでの命令フェッチ時間を効果的に解消します。

LSETUPが非ゼロの開始オフセット（lp_start）を指定すると、最初のループバックで4サイクルの遅延が発生します。したがって、ゼロ開始オフセットが好まれます。

ループ終端位置で実行される命令についても、プロセッサは何の制約も加えません。その位置でも分岐とコールを使用できます。

イベントとシーケンス

プロセッサのイベント・コントローラは、5種類の動作やイベントを管理します。

- エミュレーション
- リセット
- マスク不能割込み（NMI）
- 例外
- 割込み

なお、「**イベント**」という語は、5種類すべての動作を表します。イベント・コントローラは、全部で15種類のイベントを管理します。つまり、エミュレーション、リセット、NMI、例外、および11種類の割込みです。

割込みとは、プロセッサの通常の命令フローを変更するイベントであり、プログラム・フローに非同期です。その一方で、例外とは、ソフトウェアによって開始されるイベントであり、その作用はプログラム・フローに同期します。

イベント・システムはネストされ、優先順位付けされています。したがって、任意の時点で複数のサービス・ルーチンがアクティブになったり、優先順位の低いイベントが優先順位の高いイベントによって先取りされたりすることがあります。

このプロセッサは、2レベルのイベント制御メカニズムを採用しています。プロセッサのシステム割込みコントローラ（SIC）は、コア・イベント・コントローラ（CEC）と連携して、すべてのシステム割込みの優先順位付

けと制御を行います。SICは、多くのペリフェラル割込みソースとコアの優先順位付けされた汎用割込み入力とのマッピングを提供します。このマッピングはプログラマブルであり、個々の割込みソースはSICでマスクできます。

CECでは、表4-6に示す専用の割込みイベントと例外イベントに加えて、9つの汎用割込み（IVG7 – IVG15）が可能です。優先順位の最も低い2つの割込み（IVG14とIVG15）をソフトウェア割込みハンドラ用に確保し、7つの優先順位付けされた割込み入力（IVG7 – IVG13）でシステムに対応することをお勧めします。表 4-6を参照してください。

表 4-6.　システムとコア・イベントのマッピング

	イベント・ソース	コア・イベント名
コア・イベント	エミュレーション（最高の優先順位）	EMU
	リセット	RST
	NMI	NMI
	例外	EVX
	予備	–
	ハードウェア・エラー	IVHW
	コア・タイマ	IVTMR

表 4-6.　システムとコア・イベントのマッピング（続き）

	イベント・ソース	コア・イベント名
システム割込み	PLL ウェイクアップ割込み DMA エラー（汎用） PPI エラー割込み SPORT0 エラー割込み SPORT1 エラー割込み SPI エラー割込み UART エラー割込み	IVG7
	リアルタイム・クロック割込み DMA0 割込み（PPI）	IVG8
	DMA1 割込み（SPORT0 RX） DMA2 割込み（SPORT0 TX） DMA3 割込み（SPORT1 RX） DMA4 割込み（SPORT1 TX）	IVG9
	DMA5 割込み（SPI） DMA6 割込み（UART RX） DMA7 割込み（UART TX）	IVG10
	Timer0、Timer1、Timer2 割込み	IVG11
	プログラマブル・フラグ割込み A/B	IVG12
	DMA8/9割込み(メモリ DMAストリーム0) DMA10/11 割込み（メモリ DMA ストリーム 1） ソフトウェア・ウォッチドッグ・タイマ	IVG13
	ソフトウェア割込み 1	IVG14
	ソフトウェア割込み 2（最低の優先順位）	IVG15

なお、ここに示すシステム割込みとコア・イベントのマッピングはリセット時のデフォルト値であり、ソフトウェアで変更できます。

■ システム割込み処理

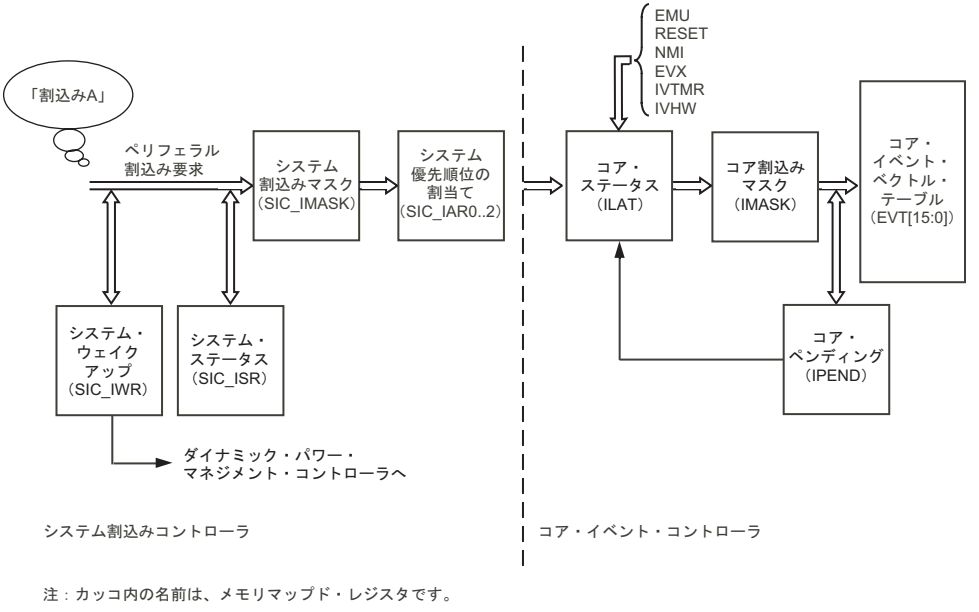
4-24ページの図4-5では、割込み（割込みA）は割込み対応のペリフェラルによって生成されるものとします。

1. SIC_ISR は要求を記録し、アサート済みであってもまだ処理されていない（つまり、割込みサービス・ルーチンがまだ割込みをクリアしていない）システム割込みを監視します。
2. SIC_IWR は、この割込み要求に基づいて、コアをアイドル状態からウェイクアップさせるべきかどうかを確認します。
3. SIC_IMASK は、システム・レベルでペリフェラルからの割込みをマスクまたはイネーブルにします。割込み A がマスクされていない場合には、要求はステップ 4 に進みます。
4. SIC_IARx レジスタは、ペリフェラル割込みを小さな汎用コア割込み（IVG7 - IVG15）群にマッピングし、割込み A のコア優先順位を決定します。
5. ILAT は割込み A を、コアによってラッチ済みであってもまだアクティブに処理されていない割込みのログに追加します。
6. IMASK は、さまざまなコア優先順位のイベントをマスクまたはイネーブルにします。割込み A に対応する IVGx イベントがマスクされていない場合には、プロセスはステップ 7 に進みます。
7. 割込み A の割込みサービス・ルーチン（ISR）に対して適切なベクトルを検索するために、イベント・ベクトル・テーブル（EVT）がアクセスされます。
8. 割込み A のイベント・ベクトルがコア・パイプラインに入ると、適切な IPEND ビットがセットされて、それぞれの ILAT ビットがクリアされます。このように、IPEND では現在処理中の割込みに加えて、すべてのペンディング割込みも追跡します。

9. 割込み A の割込みサービス・ルーチン（ISR）が実行された場合、RTI 命令は適切な IPEND ビットをクリアします。しかし関連する SIC_ISR ビットは、割込みサービス・ルーチンが割込み A を生成したメカニズムをクリアしない場合や、割込みの処理プロセスでこのビットをクリアしない場合にはクリアされません。

エミュレーション・イベント、リセット・イベント、NMIイベント、例外イベントに加えて、ハードウェア・エラー（IVHW）とコア・タイマ（IVTMR）の割込み要求は、ILAT レベルで割込み処理チェーンに入り、システムレベルの割込みレジスタ（SIC_IWR、SIC_ISR、SIC_IMASK、SIC_IARx）による影響を受けないことに注意してください。

複数の割込みソースが1つのコア割込みを共有する場合には、割込みサービス・ルーチン（ISR）は割込みを生成したペリフェラルを識別する必要があります。その後、ISRはペリフェラルに問い合わせて必要な措置を講じなければならないこともあります。



注：カッコ内の名前は、メモリマップド・レジスタです。

図 4-5.　割込み処理のブロック図

■ システム・ペリフェラル割込み

プロセッサ・システムには多数のペリフェラルがあるため、多くのサポート割込みも必要になります。これを表4-7に示します。

- ペリフェラル割込みソース
- システム割込み割当てレジスタ（SIC_IARx）で使用されるペリフェラル割込みID（4-32ページの「システム割込み割当てレジスタ（SIC_IARx）」を参照してください。）

- リセット時に割込みのマッピング先となるコアの汎用割込み
- システム割込み割当てレジスタ（SIC_IARx）で使用されるコア割込みID（4-32ページの「システム割込み割当てレジスタ（SIC_IARx）」を参照してください。）

表 4-7.　ペリフェラル割込みソースのリセット状態

ペリフェラル割込みソース	ペリフェラル割込み ID	汎用割込み（リセット時の割当て）	コア割込み ID
PLL ウェイクアップ割込み	0	IVG7	0
DMA エラー（汎用）	1	IVG7	0
PPI エラー割込み	2	IVG7	0
SPORT0 エラー割込み	3	IVG7	0
SPORT1 エラー割込み	4	IVG7	0
SPI エラー割込み	5	IVG7	0
UART エラー割込み	6	IVG7	0
リアルタイム・クロック割込み（アラーム、秒、分、時、カウントダウン）	7	IVG8	1
DMA0 割込み（PPI）	8	IVG8	1
DMA1 割込み（SPORT0 RX）	9	IVG9	2
DMA2 割込み（SPORT0 TX）	10	IVG9	2
DMA3 割込み（SPORT1 RX）	11	IVG9	2
DMA4 割込み（SPORT1 TX）	12	IVG9	2
DMA5 割込み（SPI）	13	IVG10	3
DMA6 割込み（UART RX）	14	IVG10	3
DMA7 割込み（UART TX）	15	IVG10	3
タイマ 0 割込み	16	IVG11	4
タイマ 1 割込み	17	IVG11	4
タイマ 2 割込み	18	IVG11	4

表 4-7.　ペリフェラル割込みソースのリセット状態（続き）

ペリフェラル割込みソース	ペリフェラル割込み ID	汎用割込み（リセット時の割当て）	コア割込み ID
PF 割込み A	19	IVG12	5
PF 割込み B	20	IVG12	5
DMA 8/9 割込み（メモリ DMA ストリーム 0）	21	IVG13	6
DMA 10/11 割込み（メモリ DMA ストリーム 1）	22	IVG13	6
ソフトウェア・ウォッチドッグ・タイマ割込み	23	IVG13	6
予備	24-31	–	–

プロセッサのペリフェラル割込み構造は柔軟です。リセット時のデフォルトは表 4-7に示すように、複数のペリフェラル割込みがコア内の1つの汎用割込みを共有します。

複数の割込みソースに対応する割込みサービス・ルーチンは、適切なシステム・メモリ・マップド・レジスタ（MMR）に問い合わせて割込みを生成したペリフェラルを判定する必要があります。

表 4-7に示すデフォルト割当てが許容される場合には、割込みの初期化には以下の動作だけが必要です。

- コアのイベント・ベクトル・テーブル（EVT）のベクトル・アドレス・エントリの初期化
- IMASK レジスタの初期化
- システムが必要とする、SIC_IMASK 内の特定ペリフェラル割込みのマスク解除

■ SIC_IWR レジスタ

システム割込みウェイクアップイネーブル・レジスタ（SIC_IWR）は、ペリフェラル割込みソースとダイナミック・パワー・マネジメント・コントローラ（DPMC）との間のマッピングを提供します。どのペリフェラルでも、アイドル状態のコアをウェイクアップさせて割込みを処理するように設定するには、システム割込みウェイクアップイネーブル・レジスタ（SIC_IWR、4-28ページの図4-6を参照）の適切なビットをイネーブルにします。ペリフェラル割込みソースがSIC_IWRでイネーブルにされ、コアがアイドル状態である場合には、割込みによってDPMCはコア・ウェイクアップ・シーケンスを実行して割込みを処理します。なお、パワー制御状態によっては、この動作モードによって割込み処理の遅延が増大することもあります。コアのアイドル状態とパワー・モードの詳細については、第8章「ダイナミック・パワー・マネジメント」を参照してください。

デフォルトでは、すべての割込みはコアへのウェイクアップ要求を生成します。しかし、アプリケーションによっては、SPORTx送信割込みなどいくつかのペリフェラルに対して、この機能をディスエーブルにすることが望ましい場合もあります。

コアがアイドル状態でない限り、SIC_IWRレジスタは効果がありません。このレジスタのビットは、システム割込みマスク（SIC_IMASK）レジスタと割込みステータス（SIC_ISR）レジスタのビットに対応します。

リセット後に、このレジスタのすべての有効ビットに1が設定され、マスクされていないすべての割込みに対するウェイクアップ機能がイネーブルにされます。割込みをイネーブルにする前に、このレジスタをリセット初期化シーケンスで設定します。

SIC_IWR レジスタは、いつの時点でも読出し／書込みが可能です。スプリアス割込みや割込みの欠落動作を防止するため、このレジスタへの書込みは、すべてのペリフェラル割込みがディスエーブルにされているときに限定してください。

なお、ウェイクアップ機能は、割込みマスク機能から独立しています。割込みソースが SIC_IWR でイネーブルにされていても、SIC_IMASK でマスクされている場合には、アイドル状態のコアはウェイクアップしますが、割込みは生成しません。

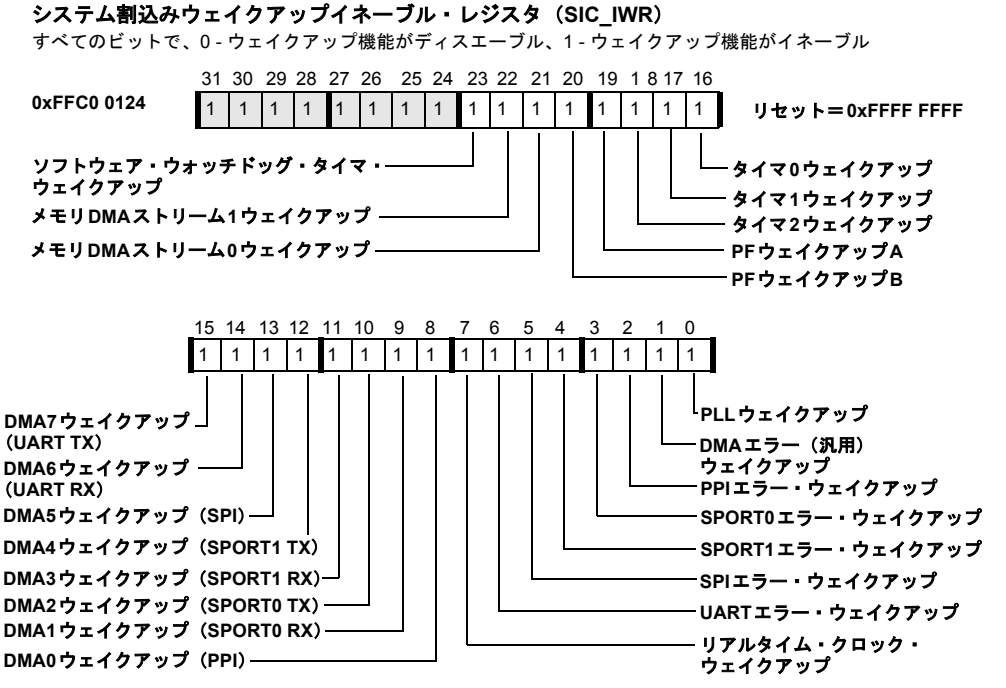


図 4-6.　システム割込みウェイクアップイネーブル・レジスタ

■ SIC_ISR レジスタ

システム割込みコントローラ（SIC）は、図 4-7に示すように、読出し専用のステータス・レジスタであるシステム割込みステータス・レジスタ（SIC_ISR）を内蔵しています。このレジスタ内の有効な各ビットは、ペリフェラル割込みソースの1つに対応します。このビットは、SICが割込みのアサートを検出するとセットされ、SICがペリフェラル割込み入力のアサート解除を検出するとクリアされます。なお、プログラマブル・フラグ非同期入力割込みなどいくつかのペリフェラルでは、割込みサービス・ルーチンが（通常はシステムMMRへの書込みによって）割込みのクリアを開始してからSICが割込みのアサート解除を感知するまでに、相当なサイクル数の遅延が経過することもあります。

割込みソースがコアの汎用割込み入力にどうマッピングされるかによりますが、割込みサービス・ルーチンは、複数の割込みステータス・ビットに問い合わせて、割込みのソースを確認しなければならないこともあります。割込みサービス・ルーチンで実行される最初の命令の1つでは、SIC_ISRを読み出して、入力を共有している複数のペリフェラルがその割込み出力をアサートしたかどうか確認してください。サービス・ルーチンでは、ペンディング中のすべての共有割込みを完全に処理してからRTIを実行してください。RTIでは、その割込み入力でのさらなる割込み生成をイネーブルにします。

割込みのサービス・ルーチンが完了すると、RTI 命令では、IPEND レジスタの適切なビットをクリアします。しかし、サービス・ルーチンが割込みを生成したメカニズムをクリアしない限り、関連する SIC_ISR ビットはクリアされません。

多くのシステムが必要とする割込み対応のペリフェラルは比較的少ないため、各ペリフェラルは、重複しないコア優先順位レベルにマッピングできます。このような設計では、SIC_ISRに問い合わせる必要はほとんどありません。

SIC_ISR レジスタは、システム割込みマスク・レジスタ（SIC_IMASK）の状態による影響を受けず、いつでも読出しできます。SIC_ISR レジスタへの書込みは、その内容に影響を与えません。

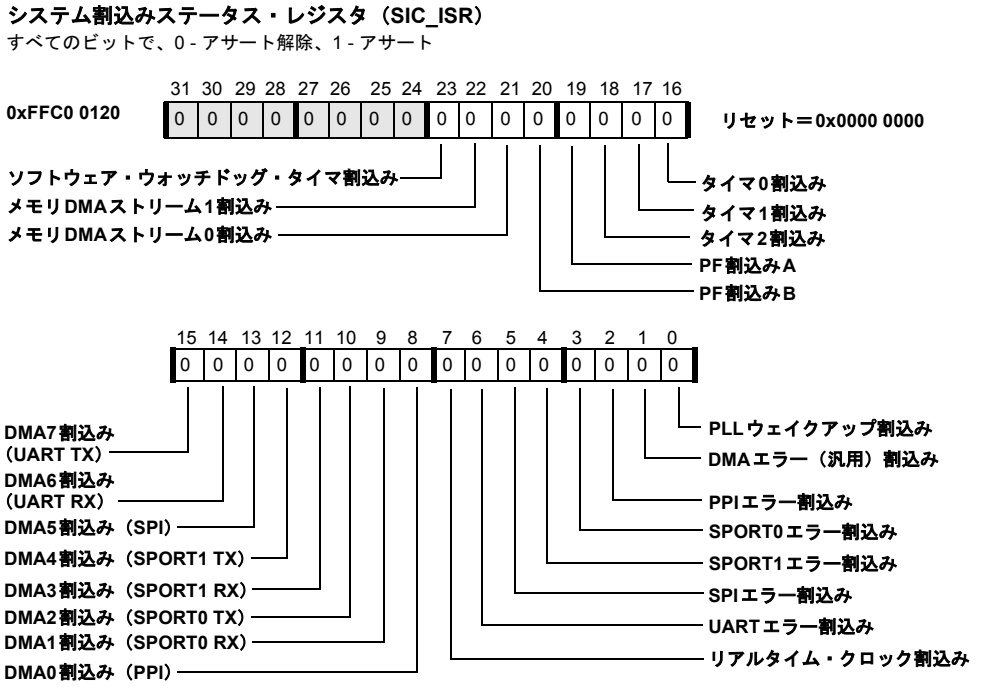


図 4-7.　システム割込みステータス・レジスタ

■ SIC_IMASK レジスタ

システム割込みマスク・レジスタ（図 4-8 に示す SIC_IMASK）を使用すれば、ペリフェラル自身でイネーブルにされているかどうかとは無関係に、システム割込みコントローラ（SIC）で任意のペリフェラル割込みソースをマスクできます。

リセットによって、SIC_IMASKの内容にはオール0が設定され、すべてのペリフェラル割込みがマスクされます。ビット位置に1を書き込むとマスクがオフになり、割込みがイネーブルにされます。

このレジスタは（スーパーバイザ・モードで）いつでも読出し／書込みが可能ですが、リセット初期化シーケンスで設定してから割込みをイネーブルにしてください。

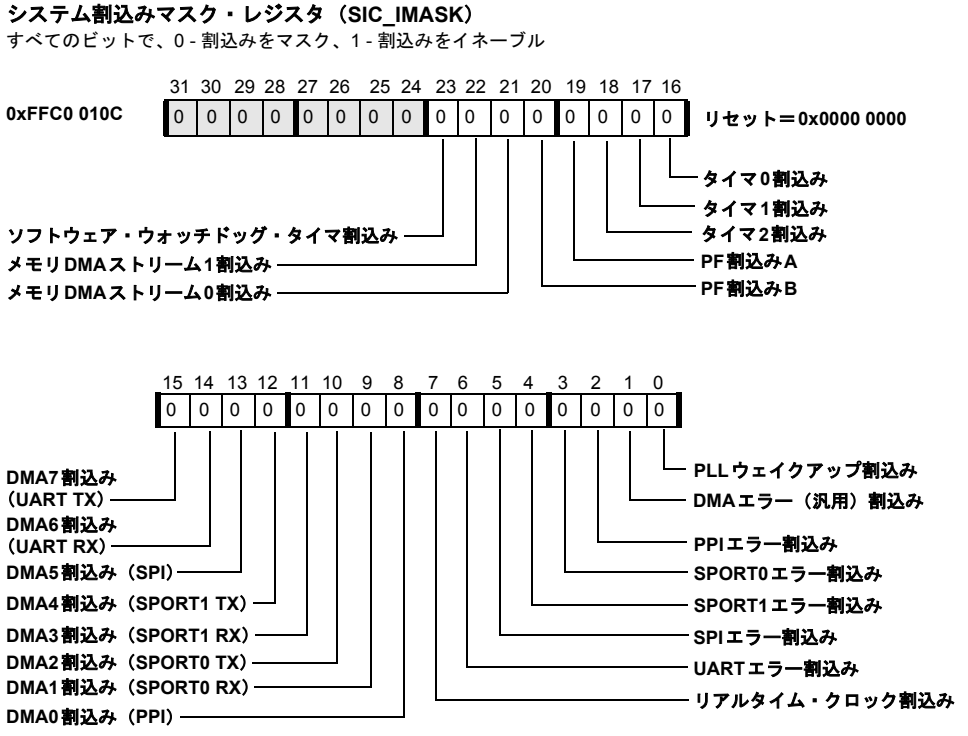


図 4-8.　システム割込みマスク・レジスタ

■ システム割込み割当てレジスタ（SIC_IARx）

ペリフェラル割込みの相対的な優先順位を設定するには、ペリフェラル割込みをコア内の適切な汎用割込みレベルにマッピングします。図 4-9、図 4-10、図 4-11に示すように、このマッピングは、システム割込み割当てレジスタの設定値によって制御されます。同じ割込みに対して複数の割込みソースがマッピングされた場合には、それらの割込みソースはハードウェアの優先順位付けなしで論理的にORがとられます。ソフトウェアでは、特定のシステム・アプリケーションの要求に応じて、割込み処理に優先順位を付けることができます。

複数のペリフェラル割込みが割り当てられた汎用割込みの場合、その入力を共有しているペンディング中のすべての割込みをソフトウェアが正しく処理できるように特に注意してください。共有される割込みの優先順位付けについては、ソフトウェアの責任となります。

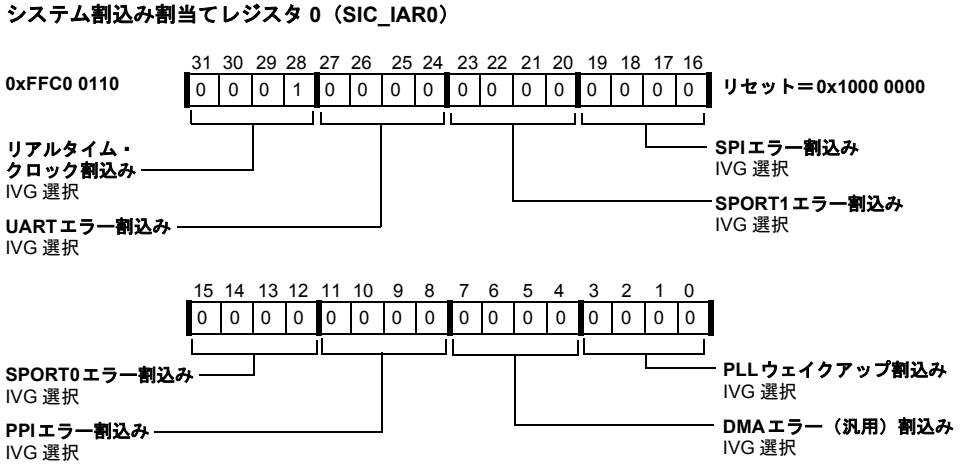


図 4-9.　システム割込み割当てレジスタ 0

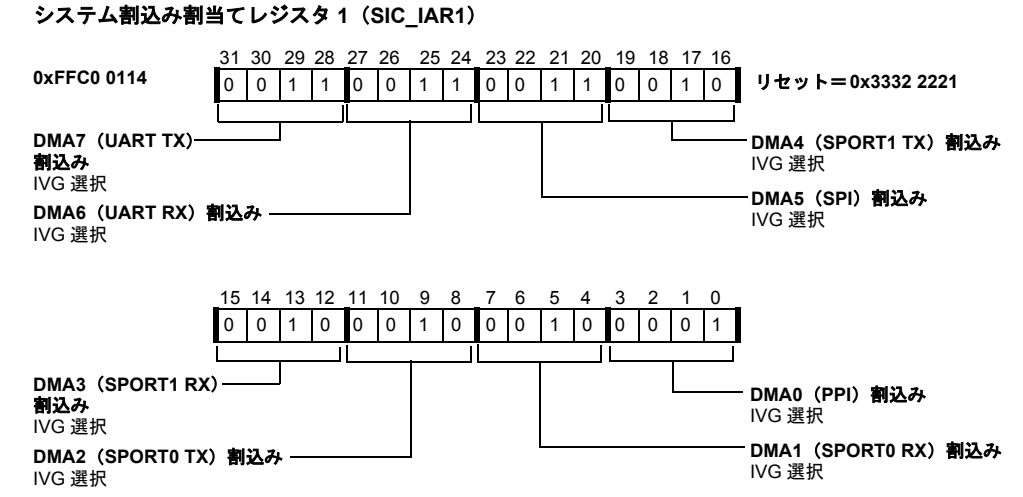


図 4-10. システム割込み割当てレジスタ 1

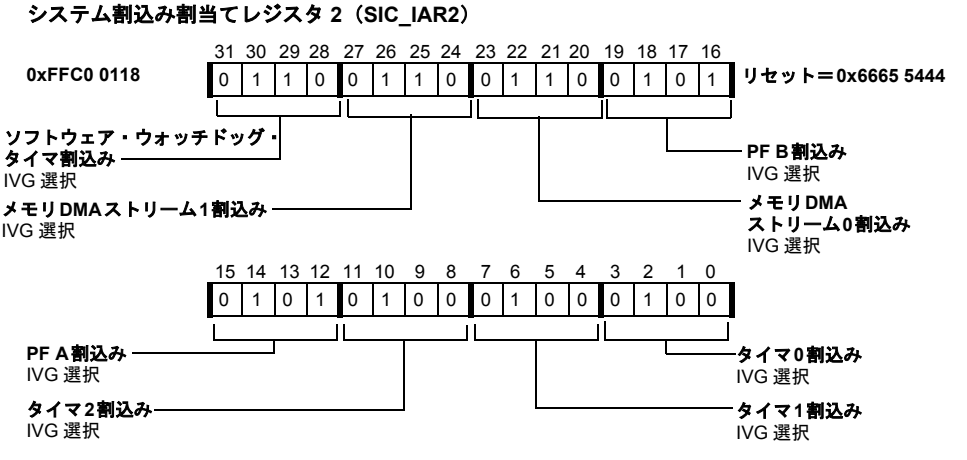


図 4-11. システム割込み割当てレジスタ 2

これらのレジスタは、スーパーバイザ・モードでいつでも読出し／書込みが可能です。しかし、リセット割込みサービス・ルーチンでこれらのレジスタを設定してから、割込みをイネーブルにすることをお勧めします。スプリアス割込みや割込みの欠落動作を防止するため、これらのレジスタへの書込みは、すべてのペリフェラル割込みがディスエーブルにされているときに限定してください。

表 4-8 は、ペリフェラルを特定の IVG 優先順位に設定するために、SIC_IARx に書き込む値を定義します。

表 4-8.　IVG 選択の定義

汎用割込み	SIC_IAR での値
IVG7	0
IVG8	1
IVG9	2
IVG10	3
IVG11	4
IVG12	5
IVG13	6
IVG14	7
IVG15	8

コア・イベント・コントローラ・レジスタ

イベント・コントローラは3つのMMRを使用して、ペンディング中のイベント要求を調整します。各MMRでは、下位16ビットが16種類のイベント・レベルに対応します（たとえば、ビット0は「エミュレータ・モード」に対応します)。レジスタは次のとおりです。

- IMASK ― 割込みマスク
- ILAT ― 割込みラッチ
- IPEND ― ペンディング中の割込み

これら3本のレジスタには、スーパーバイザ・モードでのみアクセスできます。

■ IMASK レジスタ

コア割込みマスク・レジスタ（IMASK）は、発生可能な割込みレベルを示します。IMASKレジスタは、スーパーバイザ・モードで読出し／書込みが可能です。ビット[15:5]には有効数字があります。ビット[4:0]は1にハードコードされており、これらのレベルのイベントは常にイネーブルにされています。IMASK[N] == 1でILAT[N] == 1の場合、高い優先順位がまだ認識されていなければ割込みNが発生します。IMASK[N] == 0で、割込みNによってILAT[N]がセットされた場合には、割込みは発生せずに、ILAT[N]はセットされたままです。

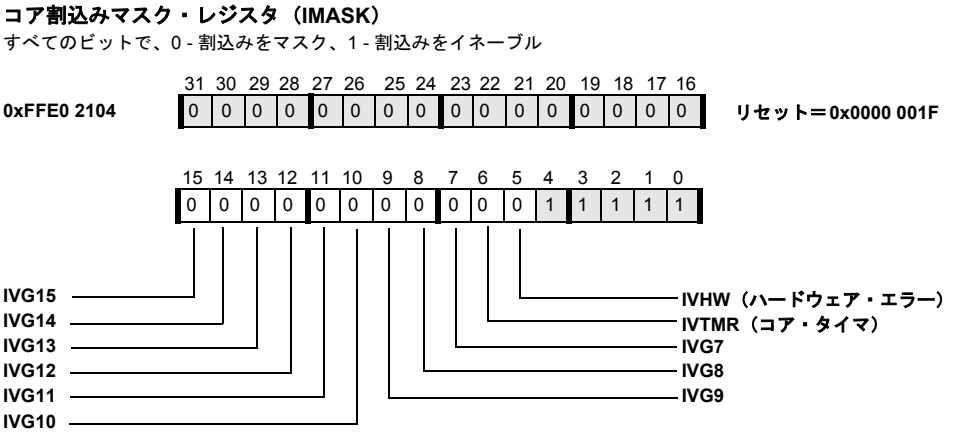


図 4-12. コア割込みマスク・レジスタ

■ ILAT レジスタ

コア割込みラッチ・レジスタ（ILAT）の各ビットは、対応するイベントがラッチされていてもプロセッサにはまだ受け付けられていないことを示します（図 4-13を参照）。このビットがリセットされてから、対応するISR の最初の命令が実行されます。割込みが受け付けられた時点でILAT[N]がクリアされ、同時にIPEND[N]がセットされます。ILAT レジスタは、スーパーバイザ・モードで読出し可能です。ILAT への書込みは、(スーパーバイザ・モードで) ビットをクリアするためにだけ使用されます。ILATからビットNをクリアするには、まずIMASK[N] == 0であることを確認してから、ILAT[N] = 1を書き込みます。ILAT へのこの書込み機能は、ラッチされた割込み要求を処理せずにクリア（キャンセル）する必要がある場合に備えて提供されています。

RAISE 命令を使用すれば、ILAT[15]～ILAT[5]およびILAT[2]またはILAT[1]をセットできます。

ILAT[0]をクリアできるのは、JTAG TRST ピンだけです。

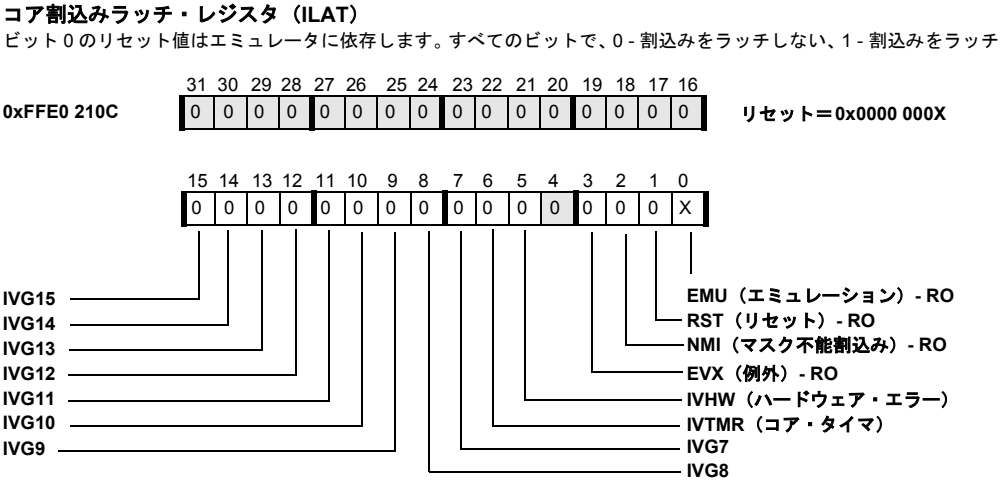


図 4-13. コア割込みラッチ・レジスタ

■ IPEND レジスタ

コア割込みペンディング・レジスタ（IPEND）は、現在ネストされているすべての割込みを監視します（図 4-14 を参照）。IPEND の各ビットは、対応する割込みが現在アクティブであるか、いずれかのレベルでネストされていることを示します。このビットは、スーパーバイザ・モードで読出し可能ですが、書込みはできません。イベント・コントローラでは、IPEND[4] ビットを使用して、エントリ時に割込みを一時的にディスエーブルにして割込みサービス・ルーチンに抜け出ることができます。

イベントが処理されると、IPEND の対応するビットがセットされます。IPEND の最下位ビットが現在セットされていれば、これは現在処理中の割込みを示します。いつの時点でも、IPEND はネストされたすべてのイベントの現在のステータスを保持します。

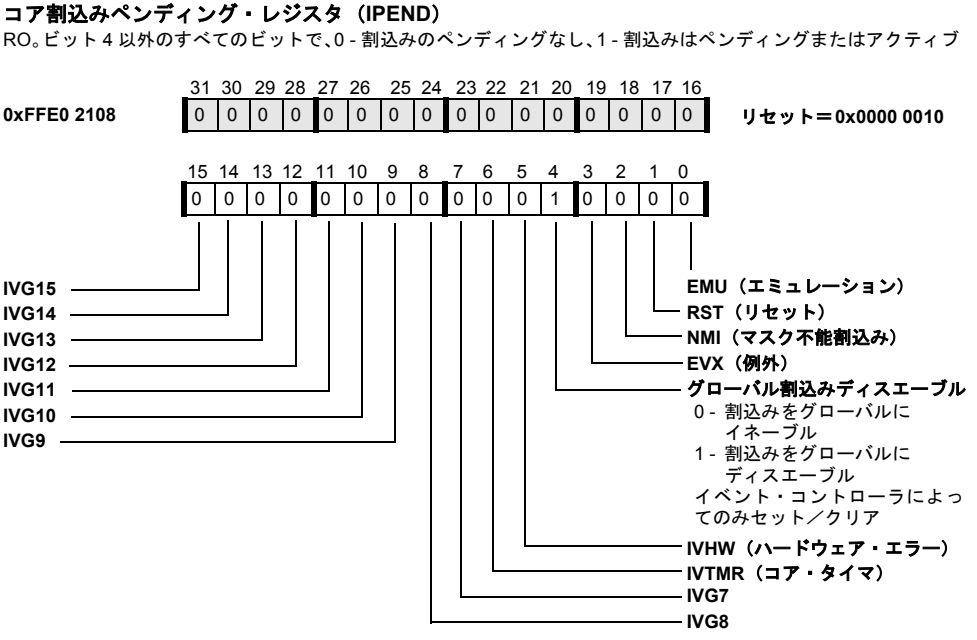


図 4-14. コア割込みペンディング・レジスタ

割込みのグローバルなイネーブル／ディスエーブル

汎用割込みは、CLI Dreg命令でグローバルにディスエーブルにでき、STI Dreg命令で再びイネーブルにできます。ただし、いずれの命令もスーパーバイザ・モードでのみ使用できます。リセット・イベント、NMIイベント、エミュレーション・イベント、例外イベントは、グローバルにディスエーブルにできません。割込みをグローバルにディスエーブルにすると、IMASK の現在の状態を保存した後で IMASK[15:5] がクリアされます。『ADSP-BF53x Blackfin Processor Instruction Set Reference』の「External Event Management (外部イベント管理)」の章にある「Enable Interrupts（割込みイネーブル)」と「Disable Interrupts（割込みディスエーブル)」を参照してください。

プログラム・コードの時間条件が厳しすぎて、割込みによる遅延を許容できない場合には、汎用割込みをディスエーブルにしてください。ただし、コード・シーケンスの終了時点で、再びイネーブルにすることを忘れないでください。

イベント・ベクトル・テーブル

イベント・ベクトル・テーブル（EVT）は16個のエントリを持つハードウェア・テーブルであり、それぞれのエントリは32ビット幅です。EVTには、可能なコア・イベントごとに1つのエントリがあります。エントリはMMRとしてアクセスされ、各エントリは、リセット時に割込みサービス・ルーチンの対応するベクトル・アドレスによってプログラムできます。イベントが発生すると、そのイベントに対するEVTエントリにあるアドレス位置から命令フェッチが開始されます。

プロセッサ・アーキテクチャによって、各割込みベクトルに重複しないアドレスをプログラムできます。つまり、割込みベクトルは、割込みベクトル・テーブルのベース・アドレスからの固定オフセットによって決まるものではありません。この方式では、ベクトル・テーブルから実際のISRコードまでのロング・ジャンプを必要としないため、遅延が最小限に抑えられます。

表 4-9に、イベントを優先順位別に示します。各イベントには、イベント状態レジスタILAT、IMASK、IPEND内に対応するビットがあります。

表 4-9.　コア・イベント・ベクトル・テーブル

イベント番号	イベント・クラス	名前	MMR 位置	注
EVT0	エミュレーション	EMU	0xFFE0 2000	最高優先順位。ベクトル・アドレスは JTAG によって提供されます。
EVT1	リセット	RST	0xFFE0 2004	
EVT2	NMI	NMI	0xFFE0 2008	

表 4-9.　コア・イベント・ベクトル・テーブル（続き）

イベント番号	イベント・クラス	名前	MMR 位置	注
EVT3	例外	EVX	0xFFE0 200C	
EVT4	予備	予備	0xFFE0 2010	予備ベクトル
EVT5	ハードウェア・エラー	IVHW	0xFFE0 2014	
EVT6	コア・タイマ	IVTMR	0xFFE0 2018	
EVT7	割込み 7	IVG7	0xFFE0 201C	
EVT8	割込み 8	IVG8	0xFFE0 2020	
EVT9	割込み 9	IVG9	0xFFE0 2024	
EVT10	割込み 10	IVG10	0xFFE0 2028	
EVT11	割込み 11	IVG11	0xFFE0 202C	
EVT12	割込み 12	IVG12	0xFFE0 2030	
EVT13	割込み 13	IVG13	0xFFE0 2034	
EVT14	割込み 14	IVG14	0xFFE0 2038	
EVT15	割込み 15	IVG15	0xFFE0 203C	最低優先順位

■ エミュレーション

エミュレーション・イベントによってプロセッサはエミュレーション・モードに入り、命令はJTAGインターフェースから読み出されます。これは、コアへの最高優先順位の割込みです。

エミュレーションの詳細については、第19章「Blackfinプロセッサのデバッグ」を参照してください。

■ リセット

リセット割込み（RST）を開始するには、RESETピンを使用したり、ウォッチドッグ・タイマの満了を利用します。この位置は、その内容が読出し専用であるという点で他の割込みの位置とは異なります。このアドレスへの書込みはレジスタを変更しますが、リセット時にプロセッサが制御を移す先は変化しません。リセット時には、プロセッサは常にリセット・ベクトル・アドレスに制御を移します。詳細については、3-11ページの「リセット状態」と3-19ページの「ブート方式」を参照してください。

コアにはダブル・フォルトの発生を示す出力があります。これは回復不能な状態です。システムは（SWRSTレジスタを利用し）、ダブル・フォルト条件が検出された場合にリセット要求を送信するようにプログラムできます。その後、リセット要求によってコアとペリフェラルのシステム・リセットが行われます。

リセット・ベクトルはプロセッサ・システムによって決定され、BMODE[1:0]ピンの状態に応じて、オンチップ・ブートROMの先頭、または外部非同期メモリの先頭をポイントします。表 4-10を参照してください。

表 4-10. リセット・ベクトル・アドレス

ブート・ソース	BMODE[1:0]	実行開始アドレス
ブート ROM をバイパス。16 ビット幅の外部メモリ（非同期バンク 0）から実行。	00	0x2000 0000
ブート ROM を使用して 8/16 ビット・フラッシュからブート。	01	0xEF00 0000
予備。	10	0xEF00 0000
ブート ROM を使用して、SPI シリアル EEPROM（8/16/24 ビットのアドレス範囲）からブート・コードを設定およびロード[1]。	11	0xEF00 0000

1 Rev.0.3 シリコンより。BMODE[1:0]=10 は、SPI ホストからのスレーブ・ブートとして使われています。

BMODE[1:0]ピンが、フラッシュまたはシリアルEEPROMからのブーティングを示す場合には、リセット・ベクトルは、小さなブートストラップ・カーネルが存在する内部ブートROMの先頭をポイントします。ブートストラップ・コードは、システム・リセット設定レジスタ（SYSCR）を読み出してBMODE[1:0]ピンの値を決定することで、適切なブート・シーケンスを決定します。ブートROMの詳細については、3-19ページの「ブート方式」を参照してください。

BMODE[1:0]ピンがブートROMのバイパスを示す場合には、リセット・ベクトルは外部非同期メモリ領域の先頭をポイントします。このモードでは、内部ブートROMは使用されません。このメモリ領域からの読出しをサポートするため、外部バス・インターフェース・ユニット（EBIU）は、ハードウェア・リセットの結果として生じるデフォルトの外部メモリ設定を使用します。

■ NMI（マスク不能割込み）

NMIエントリは、マスク不能割込み用に予約されています。この割込みは、プロセッサへのNMI入力信号やウォッチドッグ・タイマによって生成できます。即座のプロセッサ・アテンションを必要とするため、NMIとして適切なイベントの例はパワーダウン警告です。

例外、NMI、リセット、またはエミュレーション・イベントをすでに処理しているイベント・ハンドラで例外が発生した場合には、ダブル・フォルト条件がトリガされ、例外を引き起こした命令のアドレスがRETXに書き込まれます。

■ 例外

例外は命令ストリームに同期します。つまり、特定の命令が実行完了時に例外を引き起こします。例外ハンドラが有効になるまでは、原因となる命令の後の命令は実行されません。

例外の多くはメモリ関連です。たとえば、非整列アクセスが試みられたり、キャッシュ可能性保護索引バッファ（CPLB）ミスや保護違反が発生したときに例外が発生します。また、不正な命令やレジスタの不正な組合わせが実行されたときにも例外が発生します。

例外を引き起こした命令は、その例外がサービス・タイプであるかエラー・タイプであるかに応じて、例外イベントが発生する前にコミットする場合とコミットしない場合があります。

サービス・タイプのイベントを引き起こした命令はコミットし、例外を引き起こした命令の後の次の命令はRETXレジスタに書き込まれたアドレスとなります。サービス・タイプの例外の一例はシングル・ステップです。

エラー・タイプのイベントを引き起こした命令はコミットできないため、原因となる命令のアドレスはRETXレジスタに書き込まれたアドレスとなります。エラー・タイプのイベントの一例はCPLBミスです。

通常、RETXレジスタには復帰先の正しいアドレスが格納されています。例外を引き起こした命令を飛ばすには、次のアドレスが単に次の順次アドレスでない場合に注意してください。これが起きるのは、例外を引き起こした命令がループ終端であるときです。その場合には、次の適切なアドレスはループ先頭となります。

シーケンサ・ステータス・レジスタ（SEQSTAT）のEXCAUSE[5:0]フィールドは例外が発生するたびに書き込まれ、発生した例外のタイプを例外ハンドラに知らせます。例外の原因となるイベントの一覧については、表 4-11を参照してください。

例外、NMI、リセット、またはエミュレーション・イベントをすでに処理しているイベント・ハンドラで例外が発生した場合には、ダブル・フォルト条件がトリガされ、例外を引き起こした命令のアドレスがRETXに書き込まれます。

表 4-11. 例外の原因となるイベント

例外	EXCAUSE [5:0]	タイプ：（E）エラー（S）サービス 注 1 を参照	注／例
4 ビットの m フィールドを持つ例外命令 EXCPT を強制	m フィールド	S	命令は4ビットのEXCAUSEを提供します。
シングル・ステップ	0x10	S	プロセッサがシングル・ステップ・モードにあるとき、どの命令も例外を生成します。主にデバッギングに使用されます。
トレース・バッファのフル条件によって引き起こされた例外	0x11	S	プロセッサがこの例外を受け取るのは、トレース・バッファがオーバーフローしたときです（トレース・ユニット・コントロール・レジスタによってイネーブルにされたときのみ）。
未定義命令	0x21	E	特定のプロセッサ・システムに対して定義されていない命令のエミュレートに使用できます。
不正な命令組合わせ	0x22	E	『ADSP-BF53x Blackfin Processor Instruction Set Reference』の多重発行ルールの節を参照してください。

表 4-11. 例外の原因となるイベント（続き）

例外	EXCAUSE [5:0]	タイプ：（E）エラー（S）サービス 注 1 を参照	注／例
データ・アクセスの CPLB 保護違反	0x23	E	スーパーバイザ・リソースへの読出し／書込み、または不正なデータ・メモリ・アクセス。スーパーバイザ・リソースとは、スーパーバイザ用に予約されているレジスタと命令であり、スーパーバイザ専用レジスタ、すべての MMR、スーパーバイザ専用命令があります。（データ・アドレス・ジェネレータを使用して2つのMMRに同時にデュアル・アクセスすると、このタイプの例外が発生します。）さらに、このエントリは、禁止されたメモリ・アクセスによって引き起こされた保護違反の通知に使用され、メモリ・マネジメント・ユニット（MMU）のキャッシュ可能性保護索引バッファ（CPLB）によって定義されます。
データ・アクセスの非整列アドレス違反	0x24	E	非整列データ・メモリやデータ・キャッシュのアクセス。
回復不能なイベント	0x25	E	たとえば、前の例外の処理中に生成された例外。
データ・アクセスの CPLB ミス	0x26	E	データ・アクセスでの CPLB ミスを通知するために MMU によって使用されます。
データ・アクセスの複数 CPLB ヒット	0x27	E	複数の CPLB エントリがデータ・フェッチ・アドレスに一致します。
エミュレーション・ウォッチポイントの一致によって引き起こされた例外	0x28	E	ウォッチポイントの一致があり、ウォッチポイント命令アドレス・コントロール・レジスタ（WPIACTL）の EMUSW ビットの 1 つがセットされます。

表 4-11. 例外の原因となるイベント（続き）

例外	EXCAUSE [5:0]	タイプ：（E）エラー（S）サービス 注 1 を参照	注／例
命令フェッチの非整列アドレス違反	0x2A	E	非整列な命令のキャッシュ・フェッチ。非整列な命令のフェッチ例外では、RETX で提供される復帰アドレスは原因となる命令のアドレスではなく、非整列なデスティネーション・アドレスです。たとえば、P0 に保持されている非整列アドレスへの間接分岐が行われた場合、RETX 内の復帰アドレスは分岐命令のアドレスではなく、P0 に等しくなります。（なお、この例外は、PC 相対分岐から生成されることはなく、間接分岐からだけ生成されます）。
命令フェッチの CPLB 保護違反	0x2B	E	不正な命令フェッチ・アクセス（メモリ保護違反）。
命令フェッチの CPLB ミス	0x2C	E	命令フェッチでの CPLB ミス。
命令フェッチの複数 CPLB ヒット	0x2D	E	複数の CPLB エントリが命令フェッチ・アドレスと一致します。
スーパーバイザ・リソースの不正使用	0x2E	E	ユーザ・モードからスーパーバイザのレジスタや命令を使用しました。スーパーバイザ・リソースは、スーパーバイザ用に予約されているレジスタや命令であり、スーパーバイザ専用レジスタ、すべての MMR、スーパーバイザ専用命令が該当します。

注1：サービス（S）の場合、復帰アドレスは、例外に続く命令のアドレスです。エラー（E）の場合、復帰アドレスは、例外を引き起こした命令のアドレスです。

1つの命令が多重例外を引き起こした場合には、優先順位の最も高い例外だけが発生します。次の表では、例外を優先順位の降順に分類しています。

表 4-12. 優先順位の降順に示した例外

優先順位	例外	EXCAUSE
1	回復不能なイベント	0x25
2	命令フェッチで複数の CPLB ヒット	0x2D
3	命令フェッチで非整列アクセス	0x2A
4	命令フェッチで保護違反	0x2B
5	命令フェッチで CPLB ミス	0x2C
6	命令フェッチでアクセス例外	0x29
7	ウォッチポイントの一致	0x28
8	未定義命令	0x21
9	不正な組合わせ	0x22
10	保護リソースの不正使用	0x2E
11	DAG0 で複数の CPLB ヒット	0x27
12	DAG0 で非整列アクセス	0x24
13	DAG0 で保護違反	0x23
14	DAG0 で CPLB ミス	0x26
15	DAG1 で非整列アクセス	0x27
16	DAG1 で保護違反	0x24
17	DAG1 Protection Violation	0x23
18	DAG1 で CPLB ミス	0x26
19	EXCPT 命令	m フィールド
20	シングル・ステップ	0x10
21	トレース・バッファ	0x11

■ 例外ハンドラの実行中の例外

例外ハンドラの実行中には、別の例外を生成する命令の発行を避けてください。例外ハンドラ、NMIハンドラ、リセット・ベクトル、またはエミュレータ・モードでのコードの実行中に例外が引き起こされた場合：

- 例外を引き起こした命令はコミットされません。その命令からのすべてのライトバックは禁止されます。
- 生成された例外は受け取られません。
- SEQSTAT の EXCAUSE フィールドは、回復不能なイベント・コードで更新されます。
- 原因となる命令のアドレスがRETXに保存されます。なお、プロセッサが、たとえばNMIハンドラを実行していた場合には、RETN レジスタは更新されていません。例外を引き起こした命令のアドレスは、常にRETXに格納されます。

例外ハンドラの実行中に例外が発生したかどうかを判定するには、例外ハンドラの最後にあるSEQSTATを調べて、「回復不能なイベント」（EXCAUSE = 0x25）を示すコードがあるかどうかをチェックします。回復不能なイベントが発生した場合、レジスタRETXには例外を引き起こした最も新しい命令のアドレスが保持されています。このメカニズムは、回復ではなく検出を意図したものです。

ハードウェア・エラー割込み

ハードウェア・エラー割込みは、ハードウェア・エラーまたはシステムの誤動作を示します。ハードウェア・エラーが発生するのは、メモリ・バス・コントローラなど、コアの外部にあるロジックがデータ転送（読出し／書込み）を完了することができずに、コアのエラー入力信号をアサートしたときです。このようなハードウェア・エラーは、ハードウェア・エラー割込み（イベント・ベクトル・テーブル（EVT）、ILAT レジスタ、IMASK

レジスタ、IPENDレジスタの割込みIVHW）を呼び出します。その後、ハードウェア・エラー割込みサービス・ルーチンは、シーケンサ・ステータス・レジスタ（SEQSTAT）内に現れる5ビットのHWERRCAUSEフィールドからエラーの原因を読み出し、それに応じて応答できます。

ハードウェア・エラー割込みの生成原因を次に示します。

- バス・パリティ・エラー
- パフォーマンス・モニタのオーバーフローなど、コア内での内部エラー条件
- ペリフェラル・エラー
- バス・タイムアウト・エラー

表4-13に、対応するハードウェア条件を、関連するHWERRCAUSEコードとともに示します。最も新しいエラーのビット・コードは、HWERRCAUSEフィールドに現われます。複数のハードウェア・エラーが同時に発生した場合には、最後のハードウェア・エラーだけが認識および処理できます。コアは、複数のエラー・コードの優先順位付け、パイプライン処理、キュー登録には対応していません。ハードウェア・エラー割込みは、いずれかのエラー条件がアクティブである限り、アクティブ状態を保ちます。

表 4-13. ハードウェア・エラー割込みを引き起こすハードウェア条件

ハードウェア条件	HWERRCAUSE（2 進）	HWERRCAUSE（16 進）	注／例
システム MMR エラー	0b00010	0x02	エラーが発生するのは、無効なシステム MMR 位置がアクセスされた場合、32 ビット・レジスタが 16 ビット命令でアクセスされた場合、または 16 ビット・レジスタが 32 ビット命令でアクセスされた場合です。
外部メモリ・アドレッシング・エラー	0b00011	0x03	
パフォーマンス・モニタのオーバーフロー	0b10010	0x12	
RAISE 5 命令	0b11000	0b18	ソフトウェアが RAISE 5 命令を発行してハードウェア・エラー割込み（IVHW）を呼び出しました。
予備	その他すべてのビット組合わせ	その他すべての値	

■ コア・タイマ

コア・タイム割込み(IVTMR)は、コア・タイマ値がゼロに達したときにトリガされます。第15章「タイマ」を参照してください。

■ 汎用割込み（IVG7 - IVG15)

汎用割込みは、プロセッサによる介在を必要とするイベントに使用されます。たとえば、DMA コントローラがデータ伝送の終了を通知するために使用したり、シリアル通信デバイスが伝送エラーを通知するために使用できます。

ソフトウェアでも、RAISE命令を使用して汎用割込みをトリガできます。RAISE命令は、割込みIVG15-IVG7、IVTMR、IVHW、NMI、RST用のイベントを強制しますが、例外とエミュレーション（それぞれ、EVXとEMU）用には強制しません。

優先順位の最も低い2つの割込み（IVG15とIVG14）は、ソフトウェア割込みハンドラ用に確保することをお勧めします。

割込みの処理

コア・イベント・コントローラ（CEC）には、イベントごとに1つの割込みキュー・エレメント（ILATレジスタ内のビット）があります。割込みの立上がりエッジが検出されると、適切なILATビットがセットされます（2つのコア・クロック・サイクルが必要）。このビットは、それぞれのIPENDレジスタ・ビットがセットされるとクリアされます。IPENDビットは、イベント・ベクトルがコア・パイプラインに入ったことを示します。この時点でCECは、対応する割込み入力での次の立上がりエッジ・イベントを認識してキューに登録します。汎用割込みの立上がりエッジ遷移からIPEND出力のアサートまでの最小遅延は、3つのコア・クロック・サイクルです。しかし、コアの動作レベルと状態によっては、遅延が増大することもあります。

割込みの処理タイミングを決定するため、コントローラは、ILAT、IMASK、現在のプロセッサ優先順位レベル内の3つの量の論理的なANDをとります。

優先順位の最も高い割込みの処理には、以下のアクションを伴います。

1. イベント・ベクトル・テーブル（EVT）の割込みベクトルは、次のフェッチ・アドレスになります。

   割込み時には、現在パイプライン内にある大部分の命令がアボートされます。サービス例外時には、例外を引き起こした命令の後の命令はすべてアボートされます。エラー例外時には、例外を引き起こした命令とそれ以降のすべての命令がアボートされます。

2. 復帰アドレスは、適切なリターン・レジスタに保存されます。

   リターン・レジスタは、割込みの場合は RETI、例外の場合は RETX、NMI の場合は RETN、デバッグ・エミュレーションの場合は RETE となります。復帰アドレスとは、通常のプログラム・フローから実行された最後の命令の後の命令のアドレスです。

3. プロセッサ・モードには、受け取られたイベントのレベルが設定されます。

   イベントが NMI、例外、または割込みである場合には、プロセッサ・モードはスーパーバイザです。イベントがエミュレーション例外である場合には、プロセッサ・モードはエミュレーションです。

4. 最初の命令が実行を開始する前に ILAT 内の対応する割込みビットがクリアされ、IPEND 内の対応するビットがセットされます。

   RETI 内の復帰アドレスが保存されるまでは、ビット IPEND[4] もすべての割込みをディスエーブルにするように設定されます。

割込みのネスティング

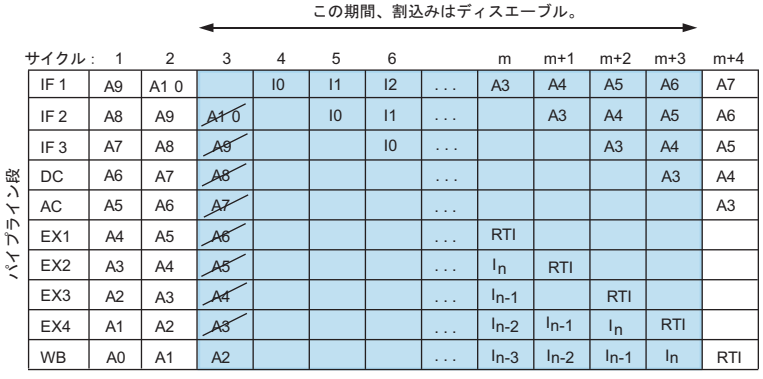
割込みは、ネスティング付きまたはネスティングなしで処理されます。

■ ネストされていない割込み

割込みがネスティングを必要としない場合には、割込みサービス・ルーチンの実行中にはすべての割込みがディスエーブルにされます。ただし、システムは、エミュレーション、NMI、例外を依然として受け付けます。

システムがネストされた割込みに対応する必要がない場合は、RETIに保持されている復帰アドレスを格納する必要はありません。割込みサービス・ルーチンで使用されるマシン状態の一部だけをスーパーバイザ・スタックに保存する必要があります。ネストされていない割込みサービス・ルーチンから復帰するには、復帰アドレスはすでにRETIレジスタに保持されているため、RTI命令だけを実行する必要があります。

図 4-15は、割込みサービス・ルーチンの全体に対して割込みがグローバルにディスエーブルにされている割込み処理の一例を示します。



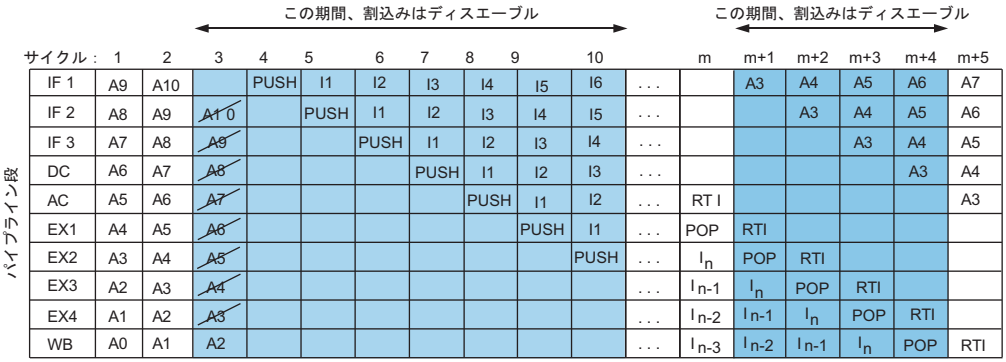
サイクル1：割込みはラッチされます。すべての可能な割込みソースが決定されます。
サイクル2：割込みは優先順位付けされます。
サイクル3：A2より上のすべての命令がキルされます。A2は、RTIまたはCLI命令である場合にはキルされます。ISRの開始アドレスのルックアップが行われます。
サイクル4：I0（ISRの先頭にある命令）がパイプラインに入ります。
サイクルM：RTI命令がEX1段に到達すると、割込みからの復帰に備えて、命令A3がフェッチされます。
サイクルM+4：RTIがWB段に到達し、割込みを再度イネーブルにします。

図 4-15. ネストされていない割込みの処理

■ ネストされた割込み

割込みがネスティングを必要とする場合には、元の割込みサービス・ルーチン内の割り込まれたポイントへの復帰アドレスを明示的に保存し、その後、ネストされた割込みサービス・ルーチンの実行が完了したときに復元する必要があります。ネスティングをサポートする割込みサービス・ルーチン内の最初の命令では、現在RETIに保持されている復帰アドレスをスーパーバイザ・スタックにプッシュして保存する必要があります（[--SP] = RETI）。これによってグローバル割込みディスエーブル・ビットIPEND[4]がクリアされ、割込みがイネーブルになります。次に、割込みサービス・ルーチンによって変更されるすべてのレジスタが、スーパーバイザ・スタックに保存されます。プロセッサ状態は、ユーザ・スタックではなく、スーパーバイザ・スタックに格納されます。したがって、RETIをプッシュする命令（[--SP] = RETI）とRETIにポップする命令（RETI = [SP++]）では、スーパーバイザ・スタックを使用します。

図 4-16では、上記の動作を、RETIをスタックにプッシュすることで説明します。割込みは、割込みサービス・ルーチンの実行中に再びイネーブルにできるため、割込みが短期間だけグローバルにディスエーブルにされます。



サイクル1：割込みはラッチされます。すべての可能な割込みソースが決定されます。
サイクル2：割込みは優先順位付けされます。
サイクル3：A2より上のすべての命令がキルされます。A2は、RTIまたはCLI命令である場合にはキルされます。
ISRの開始アドレスのルックアップが行われます。
サイクル4：I0（ISRの先頭にある命令）がパイプラインに入ります。
これは（ネスティングをイネーブルにする）PUS HRETI命令であると想定します。
サイクル10：プッシュがEX2段に到達すると、割込みは再びイネーブルにされます。
サイクルM+1：POP RETI命令がEX2段に到達すると、割込みはディスエーブルにされます。
サイクルM+5：RTIがWB段に到達すると、割込みは再びイネーブルにされます。

図 4-16. ネストされた割込みの処理

▶ ネストされた割込みサービス・ルーチンのプロローグ・コード例

リスト 4-2. ネストされた ISR のプロローグ・コード

```
/* ネストされた割込みサービス・ルーチンのプロローグ・コード。
RETI内の復帰アドレスをスーパーバイザ・スタックにプッシュして、割込みが再開されるようにします。今まで、割込みは中断されていました。*/
ISR:
[--SP] = RETI ; /* 割込みをイネーブルにし、復帰アドレスをスタックに保存します */
[--SP] = ASTAT ;
[--SP] = FP ;
[-- SP] = (R7:0, P5:0) ;
/* サービス・ルーチンの本体。なお、いずれのプロセッサ・リソース（アキュムレータ、DAG、ループ・カウンタ、ループ範囲）も保存されていません。この割込みサービス・ルーチンでは、プロセッサ・リソースを使用しないと想定します。*/
```

▶ ネストされた割込みサービス・ルーチンのエピローグ・コード例

リスト 4-3. ネストされた ISR のエピローグ・コード

```
/* ネストされた割込みサービス・ルーチンのエピローグ・コード。
ASTATレジスタ、データ・レジスタ、ポインタ・レジスタを復元します。スーパーバイザ・スタックからRETIにポップすることで、復帰アドレスのロードとRTIとの間で割込みが中断されます。*/
(R7:0, P5:0) = [SP++] ;
FP    = [SP++] ;
ASTAT = [SP++] ;
RETI  = [SP++] ;
/* RTIを実行して復帰アドレスにジャンプし、割込みを再びイネーブルにし、ユーザ・モードに切り替えます（これが処理中に最後にネストされた割込みである場合）。*/
RTI;
```

RTI命令によって、割込みからの復帰が行われます。復帰アドレスはスタックからRETIレジスタにポップされます。このアクションによって、RETIが復元された時点からRTIが実行を完了するまで割込みは中断されます。割込みの保留によって、それ以降の割込みによるRETIレジスタの破壊を防止します。

次に、RTI命令では、現在IPENDでセットされている優先順位の最も高いビットをクリアします。その後、プロセッサはRETIレジスタの値でポイントされているアドレスにジャンプし、IPEND[4]をクリアして割込みを再びイネーブルにします。

▶ ネストされた割込み要求のロギング

システム割込みコントローラ（SIC）は、ペリフェラルからのレベル・センシティブな割込み要求を検出します。コア・イベント・コントローラ（CEC）は、その汎用割込み（IVG7-IVG15）にエッジ・センシティブな検出を提供します。したがって、SICはCECへの同期割込みパルスを生成した後、CECからの割込み確認通知を待機します。割込みが（適切なIPEND出力のアサーションによって）コアから確認通知されると、ペリフェラル割込みがまだアサートされている場合には、SICはCECへの別の同期割込みパルスを生成します。このように、システムは別の割込みの処理中に発生するペリフェラル割込み要求を紛失しません。

複数の割込みソースを1つのコア・プロセッサの汎用割込みにマッピングできます。このため、SICからの複数のパルス・アサーションは、この割込み入力ですでに検出されている割込みイベントに対する割込みの処理中または処理前に同時に行うことができます。共有割込みの場合、上述のIPEND割込みアクノレッジ・メカニズムは、すべての共有割込みを再びイネーブルにします。まだアサートされている共有割込みソースがある場合には、SICによって少なくとも1つのパルスが再び生成されます。割込みステータス・レジスタでは、共有割込みソースの現在の状態を示します。

■ 例外処理

割込みと例外では、パイプライン内の命令の処理が異なります。

- 割込みが発生すると、パイプライン内のすべての命令がアボートされます。
- 例外が発生すると、パイプライン内の例外を引き起こした命令より後のすべての命令がアボートされます。エラー例外の場合には、例外を引き起こした命令もアボートされます。

例外、NMI、エミュレーション・イベントには専用のリターン・レジスタがあるため、復帰アドレスの保護はオプションです。したがって、例外、NMI、エミュレーション・イベントに対するPUSH命令とPOP命令は、割込みシステムに影響を与えません。

しかし、例外（RTX、RTN、RTE）に対する復帰命令は、IPENDで現在セットされている最下位のビットをクリアします。

▶ 例外処理の遅延

例外ハンドラは一般に、長いルーチンとなります。なぜなら、複数の例外原因を見分けて、それに応じた是正処置をとる必要があるためです。ルーチンの長さが増大することで、割込みシステムが事実上中断されることもあります。

割込みの長期の保留を回避するには、例外原因を識別する例外ハンドラを作成し、その処理を優先順位の低い割込みに任せます。優先順位の低い割込みハンドラを設定するには、Force Interrupt / Reset（RAISE）命令を使用します。

例外の処理を優先順位の低い割込みIVGxに任せるとき、システムはIVGxに入ってから、例外を発行したアプリケーション・レベル・コードに復帰することを保証しなければなりません。IVGxより優先順位の高いペンディング割込みが発生した場合には、IVGxより前に優先順位の高い割込みに入ってもかまいません。

▶ 例外ハンドラのコード例

次に、保留処理付きの例外ルーチン・ハンドラのコードを示します。

リスト 4-4. 保留処理付きの例外ルーチン・ハンドラ

```
/* SEQSTATのEXCAUSEフィールドを検査して例外原因を確認（最初にR0、P0、P1、
ASTATの内容をスーパーバイザSPに保存）*/
[--SP] = R0 ;
[--SP] = P0 ;
[--SP] = P1 ;
[--SP] = ASTAT ;
R0 = SEQSTAT ;
/* SEQSTATの内容をマスクし、EXCAUSEだけをR0内に残す */
R0 <<= 26 ;
R0 >>= 26 ;
/* ジャンプ・テーブルEVTABLEを使用して、R0によってポイントされるイベントに
ジャンプ */
P0 = R0 ;
P1 = _EVTABLE ;
P0 = P1 + ( P0 << 1 ) ;
R0 = W [ P0 ] (Z) ;
```

```
P1 = R0 ;
JUMP (PC + P1) ;
/* イベントのエントリ・ポイントは次のとおりです。ここで、処理は優先順位の低い
割込みIVG15に任せられます。また、パラメータの受け渡しも一般にここで行われます。
*/
_EVENT1:
RAISE 15 ;
JUMP.S _EXIT ;
/* IVG14でのイベントのエントリ */
_EVENT2:
RAISE 14 ;
JUMP.S _EXIT ;
/* 他のイベントのコメント */
/* ハンドラの最後で、R0、P0、P1、ASTATを復元し、復帰する。*/
_EXIT:
ASTAT = [SP++] ;
P1 = [SP++] ;
P0 = [SP++] ;
R0 = [SP++] ;
RTX ;
_EVTABLE:
.バイト2 addr_event1;
.バイト2 addr_event2;
...
.バイト2 addr_eventN;
/* ジャンプ・テーブルEVTABLEには、イベントごとの16ビット・アドレス・オフセッ
トが保持されます。オフセットによって、このコードは位置独立であり、テーブルは小
さくなります。
+--------------+
| addr_event1  | _EVTABLE
+--------------+
| addr_event2  | _EVTABLE + 2
```

```
+--------------+
|     . . .    |
+--------------+
| addr_eventN  | _EVTABLE + 2N
+--------------+
*/
```

▶ 例外ルーチンのコード例

次のコードは、上述のような例外ハンドラとの間でジャンプする割込みルーチンのフレームワーク例を提供します。

リスト 4-5. 例外処理用の割込みルーチン

```
[--SP] = RETI ;    /* 復帰アドレスをスタックにプッシュする。 */

/* ここにルーチンの本体を置く。*/

RETI = [SP++] ;    /* 復帰するには、復帰アドレスをポップしてジャンプする。 */

RTI ;    /* 割込みから復帰する。 */
```

▶ ISR でハードウェア・ループを使用するためのコード例

次のコードでは、割込みサービス・ルーチンでハードウェア・ループを使用する際の、保存と復元の最適な方式を示します。

リスト 4-6. ハードウェア・ループによる保存と復元

```
lhandler:
<ここに他のレジスタを保存>
[--SP] = LC0;  /* ループ0を保存 */
[--SP] = LB0;
```

```
[--SP] = LT0;

<ここにハンドラ・コード>

/* ハンドラがループ0を使用する場合には、最後にLC0＝0にしておくことをお勧めします。通常、ループが完全に実行されると自然にこうなります。LC0 == 0の場合、LT0とLB0の復元は、サイクル数の増加を招きません。LC0 != 0の場合、以下のポップが発生すると、ポップのたびに10サイクルの「再実行」損失を招きます。ポップやLC0の書込みは、常に損失を招きます。 */

LT0 = [SP++];
LB0 = [SP++];
LC0 = [SP++];    /* これは「再生」、つまり10サイクルの再フェッチを引き起こします。 */

<ここに他のレジスタを復元>

RTI;
```

■ その他のユーザビリティ問題

以下の節では、その他のユーザビリティ問題について説明します。

▶ 優先順位の低いイベントでの RTX、RTN、または RTE の実行

命令RTX、RTN、RTEは、それぞれ例外、NMI、またはエミュレータ・イベントから復帰するように設計されています。これらの命令は優先順位の低いイベントからの復帰に使用しないでください。割込みからの復帰にはRTI命令を使用します。正しい命令を使用しないと、思いがけない結果を引き起こすことがあります。

RTXの場合には、ビットIPEND[3]がクリアされます。RTIの場合には、IPENDでの優先順位の最も高い割込みのビットがクリアされます。

▶ システム・スタックの割当て

例外処理用のソフトウェア・スタック・モデルによれば、例外ハンドラがその状態を保存している間は、スーパーバイザ・スタックは例外を生成してはいけません。しかし、スーパーバイザ・スタックがCPLBエントリやSRAMブロックを越えて成長した場合には、実際には例外を生成することもあります。

スーパーバイザ・スタックが例外を生成しない（つまり、例外ハンドラの実行中に、CPLBエントリやSRAMブロックを越えてオーバーフローしない）ことを保証するには、すべての割込みサービス・ルーチンと例外ハンドラがアクティブ時に占有する最大空間を計算して、その量のSRAMメモリを割り当てます。

■ イベント処理における遅延

プロセッサ・アーキテクチャによっては、命令が外部メモリから実行され、命令フェッチ動作の進行中に割込みが発生した場合は、現在のフェッチ動作が完了するまで割込みの処理は先送りにされます。プロセッサが300MHzで動作し、100nsのアクセス・タイムで外部メモリからのコードを実行しているとします。命令フェッチ動作での割込みの発生タイミングにもよりますが、割込みサービス・ルーチンはおよそ30命令クロック・サイクルだけ先送りにされることもあります。キャッシュ・ライン・フィル動作を考慮に入れると、割込みサービス・ルーチンの先送りは数百サイクルに及ぶ場合もあります。

優先順位の高い割込みを最小の遅延で処理するためには、プロセッサは高遅延のフィル動作をシステム・レベルで完了できるようにし、割込みサービス・ルーチンはL1メモリから実行するようにします。図 4-17を参照してください。

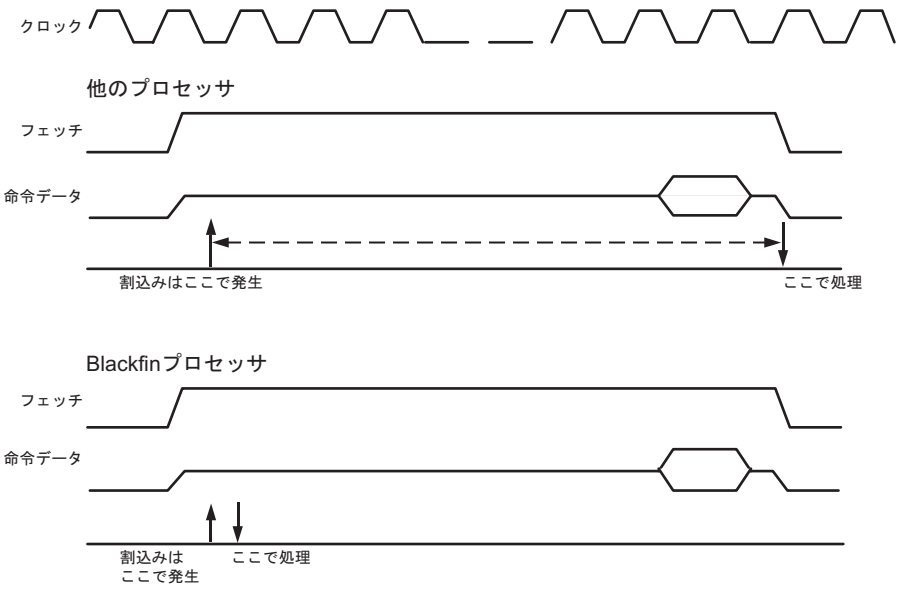


図 4-17. 割込みサービス・ルーチンの処理における遅延の最小化

命令のロード動作でL1命令キャッシュをミスして、高遅延のライン・フィル動作を生成した場合には、割込みが発生しても、割込みはフィルが完了するまで先送りされません。代わりに、プロセッサは割込みサービス・ルーチンをその新しいコンテキストで実行します。また、キャッシュ・フィル動作はバックグラウンドで完了します。

なお、プロセッサは元のキャッシュ・ライン・フィル動作の実行で、すでにビジー状態です。したがって、割込みサービス・ルーチンは、L1キャッシュまたはSRAMメモリ内に存在する必要があり、キャッシュ・ミス、L2メモリ・アクセス、ペリフェラル・アクセスを引き起こしてはいけません。これらのアクセスの1つを必要とする割込みサービス・ルーチンで

ロードまたはストア動作が実行される場合には、元の外部アクセスが実行される間は、割込みサービス・ルーチンが先送りされてから新しいロードまたはストアが開始されます。

ロード動作が完了する前に割込みサービス・ルーチンが実行を完了した場合には、プロセッサはストールを継続して、フィルの完了を待機します。

これと同じ動作は、低速なデータ・メモリやペリフェラルの読出しを伴うストールの場合にも出現します。

一般に、この動作は低速メモリへの書込みによっては生じません。なぜなら、書込みは単一サイクルと見なされ、それ以降の実行のためにすぐに書込みバッファに転送されるからです。

キャッシュとメモリ構造の詳細については、第6章「メモリ」を参照してください。

第5章　データ・アドレス・ジェネレータ

データ・アドレス・ジェネレータ（DAG）は、メモリとの間でデータ移動するためのアドレスを生成します。DAGが生成するアドレスによって、プログラムは絶対アドレスの代わりにDAGレジスタを使用して、アドレスを間接的に参照できます。

5-3ページの図5-1に示すDAGアーキテクチャは、データ・アクセス・ルーチンでのオーバーヘッドを最小限に抑えるいくつかの機能をサポートします。これらの機能には以下のものが含まれます。

- アドレス提供：データ・アクセス時にアドレスを提供します。
- アドレス提供と後更新：データ移動時にアドレスを提供し、格納されたアドレスを次の移動のためにオートインクリメント／オートデクリメントします。
- オフセット付きでのアドレス提供：元のアドレス・ポインタをインクリメントせずに、オフセット付きでベースからのアドレスを提供します。
- アドレス更新：データ移動を行わずに、格納されたアドレスをインクリメント／デクリメントします。
- ビット反転キャリー・アドレス：格納されたアドレスを反転せずに、データ移動時にビット反転キャリー・アドレスを提供します。

DAGサブシステムは、2つのDAG演算ユニット、9本のポインタ・レジスタ、4本のインデックス・レジスタ、および関連する4組のモディファイ・レジスタとベース・レジスタとレングス・レジスタで構成されます。これらのレジスタは、DAGがアドレスの生成に使用する値を保持しています。レジスタのタイプは次のとおりです。

- インデックス・レジスタ I[3:0]。32 ビットの符号なしインデックス・レジスタは、メモリへのアドレス・ポインタを保持します。たとえば、命令R3 = [I0] では、レジスタ I0によってポイントされるメモリ位置にあるデータ値をロードします。インデックス・レジスタは、16/32ビットのメモリ・アクセスに使用できます。

- モディファイ・レジスタ M[3:0]。32 ビットの符号付きモディファイ・レジスタは、レジスタ間移動中にインデックス・レジスタが後更新されるインクリメントまたはステップのサイズを提供します。たとえば、命令R0 = [I0 ++ M1] では、DAGに対して次のように指示します。

  – アドレスをレジスタ I0に出力
  – I0によってポイントされるメモリ位置の内容をR0にロード
  – I0の内容をM1 レジスタに含まれる値によって変更

- ベース・レジスタ B[3:0] とレングス・レジスタ L[3:0]。32 ビットの符号なしベース・レジスタとレングス・レジスタは、循環バッファのアドレス範囲と開始アドレスを設定します。B、Lの各ペアは、たとえばI3、B3、L3のように、常に対応するIレジスタと結合しています。循環バッファの詳細については、5-6ページの「循環バッファのアドレッシング」を参照してください。

- ポインタ・レジスタ P[5:0]、FP、USP、SP。32 ビットのポインタ・レジスタは、メモリへのアドレス・ポインタを保持します。P[5:0] フィールド、FP（フレーム・ポインタ）、SP/USP（スタック・ポインタ／ユーザ・スタック・ポインタ）は、さまざまな命令で操作および使用することができます。たとえば、命令R3 = [P0] では、レ

ジスタP0によってポイントされるメモリ位置にあるデータ値をレジスタR3にロードします。ポインタ・レジスタは循環バッファのアドレッシングに影響を与えません。ポインタ・レジスタは8/16/32ビットのメモリ・アクセスに使用できます。モード保護を強化するため、USPにはユーザ・モードでアクセスできますが、SPにはスーパーバイザ・モードでのみアクセスできます。

Lレジスタについては、リニア・アドレッシング用に自動的にゼロに初期化されると想定しないでください。I、M、L、Bの各レジスタには、リセット後にランダムな値が格納されます。使用するIレジスタごとに対応するLレジスタを、リニア・アドレッシングの場合はゼロに、循環バッファ・アドレッシングの場合はバッファ長に、それぞれプログラムで初期化しなければなりません。

すべてのDAGレジスタは、個別に初期化する必要があります。Bレジスタを初期化しても、Iレジスタは自動的に初期化されません。

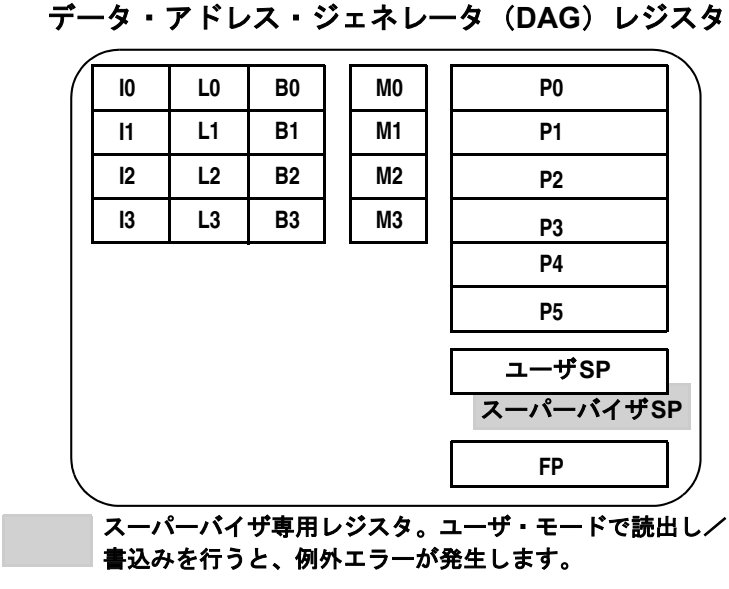


図 5-1. プロセッサの DAG レジスタ

DAG によるアドレッシング

DAGは、値またはレジスタによってインクリメントされるアドレスを生成できます。後更新アドレッシングでは、DAGはIレジスタの値を変更なしで出力します。その後、DAGはMレジスタまたは即値をIレジスタに加算します。

インデックス・アドレッシングでは、DAGはPレジスタ内の値に小さなオフセットを加算しますが、この新しい値でPレジスタを更新することはありません。このように、その特定のメモリ・アクセス用のオフセットを提供します。

プロセッサはバイト・アドレス方式です。すべてのデータ・アクセスはデータ・サイズに整列させる必要があります。つまり、32ビットのフェッチは32ビットに合わせる必要がありますが、8ビットのストアは任意のバイトに合わせることができます。使用するデータのタイプに応じて、DAGレジスタへのインクリメントとデクリメントは、8/16/32ビットのアクセスに合わせて1、2、または4とすることができます。

たとえば、次の命令を考えてみます。

```
R0 = [ P3++ ];
```

この命令は、P3の値によってポイントされる32ビット・ワードをフェッチし、それをR0に格納します。その後、P3を*4*だけポストインクリメントし、32ビット・アクセスとの整合性を維持します。

```
R0.L = W [ I3++ ];
```

この命令は、I3の値によって指し示される16ビット・ワードをフェッチし、それをデスティネーション・レジスタの下位ハーフR0.Lに格納します。その後、I3を*2*だけポストインクリメントし、16ビット・アクセスとの整合性を維持します。

```
R0 = B [ P3++ ] (Z) ;
```

この命令では、P3の値によって指し示される8ビット・ワードをフェッチし、それをデスティネーション・レジスタR0に格納します。その後、P3を*1*だけポストインクリメントして、8ビット・アクセスとの整合性を維持します。バイト値については、ゼロ拡張したり、32ビットのデータ・レジスタに符号拡張したりできます。

インデックス・レジスタを使用する命令では、修飾子としてMレジスタまたは小さな即値（+/-2または4）を使用します。ポインタ・レジスタを使用する命令では、修飾子として小さな即値または別のPレジスタを使用します。詳細については、5-18ページの表5-3「DAG命令のまとめ」を参照してください。

■ フレーム・ポインタとスタック・ポインタ

フレーム・ポインタ・レジスタとスタック・ポインタ・レジスタは、多くの点で他のPレジスタP[5:0]と同じように動作します。これらのレジスタは、たとえばR1 = B[SP] (Z)のように、任意のロード／ストア命令で一般のポインタとして機能できます。しかし、FPとSPにはこれ以外の機能もあります。

スタック・ポインタ・レジスタには、次のレジスタが含まれます。

- ユーザ・スタック・ポインタ（スーパーバイザ・モードではUSP 、ユーザ・モードではSP）
- スーパーバイザ・スタック・ポインタ（スーパーバイザ・モードではSP）

ユーザ・スタック・ポインタ・レジスタとスーパーバイザ・スタック・ポインタ・レジスタは、レジスタ・エイリアスSPを使用してアクセスされます。その時のプロセッサ動作モードに応じて、これらのレジスタの1つだけがアクティブとなり、SPとしてアクセスできます。

- ユーザ・モードは、SPへの参照（たとえば、スタック・ポップ R0 = [SP++] ;）では、有効アドレスとして暗黙的にUSPを使用します。
- スーパーバイザ・モードは、SPへの同じ参照（たとえば、R0 = [SP++] ;）では、有効アドレスとして暗黙的にスーパーバイザ・スタック・ポインタを使用します。

  スーパーバイザ・モードで動作するコードを対象としてユーザ・スタック・ポインタを操作するには、レジスタ・エイリアスUSPを使用します。スーパーバイザ・モードでは、USPからのレジスタ間移動（たとえば、R0 = USP ;）によって、現在のユーザ・スタック・ポインタはR0に転送されます。レジスタ・エイリアスUSPは、スーパーバイザ・モードでのみ使用できます。

いくつかのロード／ストア命令は、暗黙的にFPとSP を使用します。

- FP インデックス付きのロード／ストア：アドレッシング範囲を 16 ビット・エンコードされたロード／ストア用に拡張します。
- スタックのプッシュ／ポップ命令：複数のレジスタをプッシュ／ポップする命令を含みます。
- リンク／アンリンク命令：スタック・フレーム空間を制御し、その空間用にフレーム・ポインタ・レジスタ（FP）を管理します。

■ 循環バッファのアドレッシング

DAGは循環バッファのアドレッシングに対応しています。循環バッファとは、DAGが繰り返し段階的に実行するデータを含んでいる一連のアドレスであり、折り返しによって同じアドレス範囲を循環的に繰り返し、段階的な実行ができます。

DAGは、循環バッファのアドレッシング用に4種類のDAGレジスタを使用します。循環バッファリングの場合、レジスタは次のように動作します。

- インデックス（I）レジスタには、DAGがアドレス・バスに出力する値が含まれています。
- モディファイ（M）レジスタには、各メモリ・アクセスの最後にDAGがIレジスタに加算する後更新量（正または負）が含まれています。

  MレジスタとIレジスタは自由に組合わせて使用できます。更新値は、Mレジスタの代わりに即値とすることもできます。更新値のサイズは、循環バッファの長さ（Lレジスタ）以下であることが必要です。
- レングス（L）レジスタは、循環バッファのサイズと、DAGがIレジスタを巡回させるアドレス範囲を設定します。

  Lは正であり、$2^{32}-1$を超える値を持つことはできません。Lレジスタの値がゼロである場合には、その循環バッファ動作はディスエーブルにされます。
- ベース（B）レジスタ（またはBレジスタ）＋Lレジスタは、各アクセスの後でDAGが更新されたIレジスタ値と比較する値です。

循環バッファのアドレス指定を行うには、DAGはバッファ値の範囲内でインデックス・ポインタ（Iレジスタ）を段階的に変化させ、アクセスのたびにMレジスタからの正または負の更新値でインデックスを後更新します。

インデックス・ポインタがバッファ範囲を外れた場合には、DAGは値からバッファの長さ（Lレジスタ）を引くか、値にバッファの長さを足して、インデックス・ポインタをバッファ内に折り返します。

DAGが折り返すこの開始アドレスは、バッファのベース・アドレス（Bレジスタ）と呼ばれます。8ビット・データを含む循環バッファのベース・アドレスの値には何の制約もありません。16/32ビット・データを含む循環バッファは、それぞれ16/32ビットに整列する必要があります。例外は、

ビデオ動作用に作成できます。詳細については、5-14ページの「メモリのアドレス整列」を参照してください。循環バッファリングでは、後更新アドレッシングを使用します。

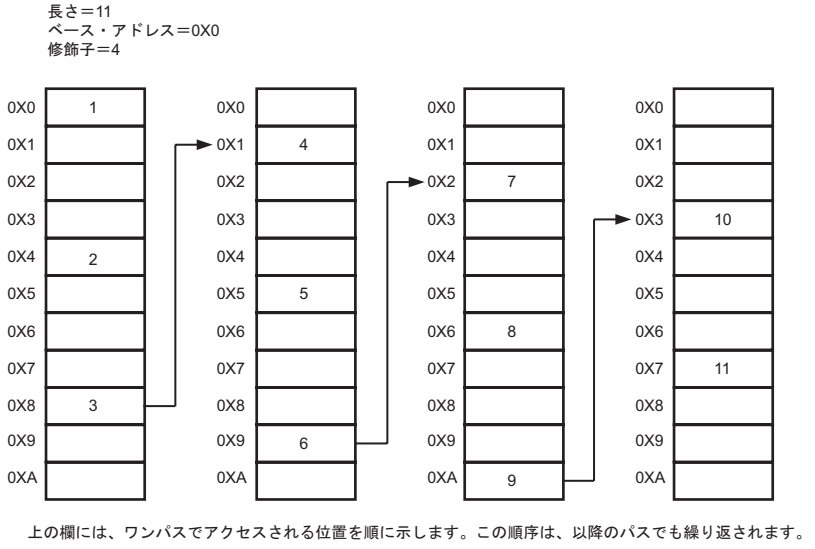


図 5-2.　循環バッファ

図 5-2に示すように、バッファへの最初の後更新アクセスで、DAGはIレジスタ値をアドレス・バス上に出力してから、更新値を加算してアドレスを変更します。

- 更新されたインデックス値がバッファ長の範囲内である場合には、DAGはその値をIレジスタに書き込みます。
- 更新されたインデックス値がバッファ長を超える場合には、DAGはLレジスタ値を引く（正の更新値の場合）または足してから（負の更新値の場合）、更新されたインデックス値をIレジスタに書き込みます。

方程式の形では、これらの後更新動作とラップアラウンド動作は、「I+M」演算では次のようになります。

- Mが正：

  $I_{new} = I_{old} + M$
  $I_{old} + M <$ バッファ・ベース + 長さ（バッファの末尾）の場合

  $I_{new} = I_{old} + M - L$
  $I_{old} + M \geq$ バッファ・ベース + 長さ（バッファの末尾）の場合

- Mが負：

  $I_{new} = I_{old} + M$
  $I_{old} + M \geq$ バッファ・ベース（バッファの先頭）の場合

  $I_{new} = I_{old} + M + L$
  $I_{old} + M <$ バッファ・ベース（バッファの先頭）の場合

■ ビット反転アドレスによるアドレッシング

いくつかのアルゴリズム（特に高速フーリエ変換（FFT）計算）で正順の結果を得るには、プログラムにビット反転キャリー・アドレッシングが必要になります。これらのアルゴリズムの条件を満たすため、DAGのビット反転アドレッシング機能では、データ・シーケンスを繰り返し細分割して、このデータをビット反転して格納します。ビット反転アドレッシングの詳細については、『ADSP-BF53x Blackfin Processor Instruction Set Reference』のModify-Increment命令を参照してください。

インデックス・レジスタとポインタ・レジスタによるインデックス・アドレッシング

インデックス・アドレッシングは、インデックス・レジスタまたはポインタ・レジスタの値を有効アドレスとして使用します。この命令では、16/32ビット値をロード／ストアできます。デフォルトは32ビット転送です。16ビット転送が必要な場合には、ロード／ストアの前に*W*指示子を付けます。

例：

```
R0 = [ I2 ];
```

I2によって指し示されたアドレスから32ビット値をロードし、それをデスティネーション・レジスタR0にストアします。

```
R0.H = W [ I2 ];
```

I2によって指し示されたアドレスから16ビット値をロードし、それを16ビットのデスティネーション・レジスタR0.H.にストアします。

```
[ P1 ] = R0 ;
```

32ビットのストア操作の一例です。

ポインタ・レジスタは、8ビットのロード／ストアに使用できます。

例：

```
B [ P1++] = R0 ;
```

R0レジスタからの8ビット値をP1レジスタによって指し示されたアドレスにストアして、P1レジスタをインクリメントします。

■ オートインクリメントとオートデクリメントのアドレッシング

オートインクリメント・アドレッシングは、アクセスの後でポインタ・レジスタとインデックス・レジスタを更新します。インクリメントの量はワード・サイズに依存します。32ビット・ワードのアクセスはポインタが4だけ更新されます。16ビット・ワードのアクセスはポインタが2だけ更新され、8ビット・ワードのアクセスはポインタが1だけ更新されます。8/16ビットの読出し動作は、内容がデスティネーション・レジスタに符号拡張またはゼロ拡張されることがあります。ポインタ・レジスタは8/16/32ビットのアクセスに使用できますが、インデックス・レジスタは16/32ビットのアクセスにだけ使用できます。

例：

```
R0 = W [ P1++ ] (Z)；
```

P1ポインタ・レジスタによって指し示されたアドレスから、32ビットのデスティネーション・レジスタに16ビット・ワードをロードします。その後、ポインタは2だけインクリメントされ、ワードはゼロ拡張されて32ビットのデスティネーション・レジスタに詰め込まれます。

オートデクリメントの場合も、アクセスの後でアドレスをデクリメントすることで同様に機能します。

例：

```
R0 = [ I2-- ] ;
```

32ビット値をデスティネーション・レジスタにロードし、インデックス・レジスタを4だけデクリメントします。

■ スタック・ポインタ・アドレッシングの前更新

プロセッサでの唯一の前更新命令は、スタック・ポインタ・レジスタSPを使用します。SP内のアドレスは4だけデクリメントされてから、ストア用の有効アドレスとして使用されます。命令[--SP] = R0;はスタックのプッシュ動作に使用され、32ビットのワード転送だけに対応します。

■ 即値オフセットによるインデックス・アドレッシング

インデックス・アドレッシングを使用すれば、データ・テーブルのベースを基準にしてプログラムがデータ・テーブルから値を取得できます。ポインタ・レジスタは、即値フィールドによって変更されてから有効アドレスとして使用されます。ポインタ・レジスタの値は更新されません。

最終アドレスが非整列である場合、整列例外がトリガされます。たとえば、P1 = 0x13である場合には、[P1 + 0x11]はすべてのアクセスに対して整列されている[0x24]と実質的に等しくなります。

■ 後更新アドレッシング

後更新アドレッシングでは、インデックス・レジスタやポインタ・レジスタ内の値を有効アドレスとして使用してから、別のレジスタの内容によってそれを変更します。ポインタ・レジスタは他のポインタ・レジスタによって変更されます。インデックス・レジスタはモディファイ・レジスタによって変更されます。後更新アドレッシングでは、デスティネーションとしてのポインタ・レジスタを使えません。また、バイト・アドレッシングにも対応していません。

例：

```
R5 = [ P1++P2 ] ;
```

P1レジスタによって指し示されるメモリ位置にある32ビット値を、R5レジスタにロードします。

その後、P2 レジスタの値が P1 レジスタの値に加算されます。

例：

```
R2 = W [ P4++P5 ] (Z) ;
```

16 ビット・ワードをデスティネーション・レジスタ R2 の下位ハーフにロードし、それを 32 ビットにゼロ拡張します。ポインタ P4 の値は、ポインタ P5 の値によってインクリメントされます。

例：

```
R2 = [ I2++M1 ];
```

32 ビット・ワードをデスティネーション・レジスタ R2 にロードします。インデックス・レジスタ I2 の値は、モディファイ・レジスタ M1 の値によって更新されます。

DAG レジスタとポインタ・レジスタの変更

DAG は、アドレスを出力せずにインデックス・レジスタのアドレス値を変更する動作が可能です。アドレス変更と呼ばれるこの動作は、ポインタの維持に役立ちます。

アドレス変更動作は、メモリにアクセスせずに DAG インデックスとポインタ・レジスタ（I[3:0]、P[5:0]、FP、SP）内のアドレスを変更します。インデックス・レジスタの対応する B レジスタと L レジスタが循環バッファリング用に設定されている場合には、アドレス変更動作では指定のバッファ・ラップアラウンドを実行します（必要な場合）。

この構文は、後更新アドレッシング（インデックス＋＝修飾子）に似ています。インデックス・レジスタの場合、M レジスタは修飾子として使用されます。ポインタ・レジスタの場合、別の P レジスタが修飾子として使用されます。

I1 += M2;という例を考えてみます。

この命令では、M2をI1に加算し、I1を新しい値で更新します。

メモリのアドレス整列

プロセッサでは、アクセスされるデータ・サイズに対して適切なメモリ整列の維持が要求されます。例外がディスエーブルにされていない限り、メモリ整列の違反によって整列例外が発生します。いくつかの命令（たとえばビデオALU命令の多く）では、データはメモリへの格納時に適切に整列されていないことがあるため、整列例外を自動的にディスエーブルにします。整列例外をディスエーブルにするには、ロード／ストア動作と並列にDISALGNEXPT命令を発行します。

通常、メモリ・システムは2つのアドレス整列を必要とします。

- 32ビット・ワードのロード／ストアは4バイト境界でアクセスされるため、アドレスの最下位2ビットはb#00です。
- 16ビット・ワードのロード／ストアは2バイト境界でアクセスされるため、アドレスの最下位ビットはb#0でなければいけません。

表 5-1に、アドレッシング・モードで利用可能な転送のタイプと転送サイズを要約します。

DISALGNEXPT命令の使用に際しては注意が必要です。なぜなら、この命令は、メモリ整列エラーの自動検出を無効にするからです。DISALGNEXPT命令は、Iレジスタの間接アドレッシングを使用する非整列ロードにのみ影響を与えます。Pレジスタ・アドレッシングを使用する非整列ロードは、依然として例外を引き起こします。

表 5-1.　利用可能な転送のタイプと転送サイズ

アドレッシング・モード	利用可能な転送のタイプ	転送サイズ
オート インクリメント オート デクリメント 間接 インデックス	データ・ レジスタとの やり取り	ロード： 32 ビット・ワード 16 ビット、ゼロ拡張されたハーフ・ワード 16 ビット、符号拡張されたハーフ・ワード 8 ビット、ゼロ拡張されたバイト 8 ビット、符号拡張されたバイト ストア： 32 ビット・ワード 16 ビット・ハーフ・ワード 8 ビット・バイト
	ポインタ・ レジスタとの やり取り	ロード： 32 ビット・ワード ストア： 32 ビット・ワード
ポスト インクリメント	データ・ レジスタとの やり取り	ロード： 32 ビット・ワード データ・レジスタ上位ハーフへの 16 ビット・ハーフ・ワード データ・レジスタ下位ハーフへの 16 ビット・ハーフ・ワード 16 ビット、ゼロ拡張されたハーフ・ワード 16 ビット、符号拡張されたハーフ・ワード ストア： 32 ビット・ワード データ・レジスタ上位ハーフからの 16 ビット・ハーフ・ワード データ・レジスタ下位ハーフからの 16 ビット・ハーフ・ワード

表 5-2に、アドレッシング・モードを要約します。この表では、プロセッサがアドレッシング・モードに対応していることをアスタリスク（*）で示しています。

表 5-2.　アドレッシング・モード

	32 ビット・ワード	16 ビット・ハーフ・ワード	8 ビット・バイト	符号／ゼロ拡張	データ・レジスタ	ポインタ・レジスタ	データ・レジスタ・ハーフ
P オートインク [P0++]	*	*	*	*	*	*	
P オートデクリ [P0--]	*	*	*	*	*	*	
P 間接 [P0]	*	*	*	*	*	*	*
P インデックス [P0+im]	*	*	*	*	*	*	
FP インデックス [FP+im]	*				*	*	
P ポストインク [P0++P1]	*	*		*	*		*
I オートインク [I0++]	*	*			*		*
I オートデクリ [I0--]	*	*			*		*
I 間接 [I0]	*	*			*		*
I ポストインク [I0++M0]	*				*		

DAG 命令のまとめ

表 5-3にDAG命令を示します。アセンブリ言語構文の詳細については、『ADSP-BF53x Blackfin Processor Instruction Set Reference』を参照してください。表 5-3では、以下の記号の意味に注意してください。

- Dregはデータ・レジスタ・ファイルの任意のレジスタを示します。
- Dreg_loはデータ・レジスタ・ファイルの任意のレジスタの下位16ビットを示します。
- Dreg_hiはデータ・レジスタ・ファイルの任意のレジスタの上位16ビットを示します。
- Pregは任意のポインタ・レジスタ、FP、またはSPレジスタを示します。
- Iregは任意のDAGインデックス・レジスタを示します。
- Mregは任意のDAGモディファイ・レジスタを示します。
- Wは16ビット幅の値を示します。
- Bは8ビット幅の値を示します。
- immAは符号付き、Aビット幅の即値を示します。
- uimmAmBは符号なし、Aビット幅で、Bの偶数倍である即値を示します。
- Zはゼロ拡張修飾子を示します。
- Xは符号拡張修飾子を示します。
- BREVはビット逆転修飾子を示します。

これらの命令に適用されるオプションや即値フィールドのサイズについては、『ADSP-BF53x Blackfin Processor Instruction Set Reference』を参照してください。

DAG命令は、ASTATステータス・フラグに影響を与えません。

表 5-3.　DAG 命令のまとめ

命令
Preg = [Preg] ;
Preg = [Preg ++] ;
Preg = [Preg --] ;
Preg = [Preg + uimm6m4] ;
Preg = [Preg + uimm17m4] ;
Preg = [Preg – uimm17m4] ;
Preg = [FP – uimm7m4] ;
Dreg = [Preg] ;
Dreg = [Preg ++] ;
Dreg = [Preg --] ;
Dreg = [Preg + uimm6m4] ;
Dreg = [Preg + uimm17m4] ;
Dreg = [Preg – uimm17m4] ;
Dreg = [Preg ++ Preg] ;
Dreg = [FP – uimm7m4] ;
Dreg = [Ireg] ;
Dreg = [Ireg ++] ;
Dreg = [Ireg --] ;
Dreg = [Ireg ++ Mreg] ;
Dreg =W [Preg] (Z) ;
Dreg =W [Preg ++] (Z) ;

表 5-3.　DAG 命令のまとめ（続き）

命令
Dreg =W [Preg --] (Z) ;
Dreg =W [Preg + uimm5m2] (Z) ;
Dreg =W [Preg + uimm16m2] (Z) ;
Dreg =W [Preg – uimm16m2] (Z) ;
Dreg =W [Preg ++ Preg] (Z) ;
Dreg = W [Preg] (X) ;
Dreg = W [Preg ++] (X) ;
Dreg = W [Preg --] (X) ;
Dreg =W [Preg + uimm5m2] (X) ;
Dreg =W [Preg + uimm16m2] (X) ;
Dreg =W [Preg – uimm16m2] (X) ;
Dreg =W [Preg ++ Preg] (X) ;
Dreg_hi = W [Ireg] ;
Dreg_hi = W [Ireg ++] ;
Dreg_hi = W [Ireg --] ;
Dreg_hi = W [Preg] ;
Dreg_hi = W [Preg ++ Preg] ;
Dreg_lo = W [Ireg] ;
Dreg_lo = W [Ireg ++] ;
Dreg_lo = W [Ireg --] ;
Dreg_lo = W [Preg] ;
Dreg_lo = W [Preg ++ Preg] ;
Dreg = B [Preg] (Z) ;
Dreg = B [Preg ++] (Z) ;
Dreg = B [Preg --] (Z) ;
Dreg = B [Preg + uimm15] (Z) ;

表 5-3.　DAG 命令のまとめ（続き）

命令
Dreg = B [Preg – uimm15] (Z) ;
Dreg = B [Preg] (X) ;
Dreg = B [Preg ++] (X) ;
Dreg = B [Preg --] (X) ;
Dreg = B [Preg + uimm15] (X) ;
Dreg = B [Preg – uimm15] (X) ;
[Preg] = Preg ;
[Preg ++] = Preg ;
[Preg --] = Preg ;
[Preg + uimm6m4] = Preg ;
[Preg + uimm17m4] = Preg ;
[Preg – uimm17m4] = Preg ;
[FP – uimm7m4] = Preg ;
[Preg] = Dreg ;
[Preg ++] = Dreg ;
[Preg --] = Dreg ;
[Preg + uimm6m4] = Dreg ;
[Preg + uimm17m4] = Dreg ;
[Preg – uimm17m4] = Dreg ;
[Preg ++ Preg] = Dreg ;
[FP – uimm7m4] = Dreg ;
[Ireg] = Dreg ;
[Ireg ++] = Dreg ;
[Ireg --] = Dreg ;
[Ireg ++ Mreg] = Dreg ;
W [Ireg] = Dreg_hi ;

表 5-3.　DAG 命令のまとめ（続き）

命令
W [Ireg ++] = Dreg_hi ;
W [Ireg --] = Dreg_hi ;
W [Preg] = Dreg_hi ;
W [Preg ++ Preg] = Dreg_hi ;
W [Ireg] = Dreg_lo ;
W [Ireg ++] = Dreg_lo ;
W [Ireg --] = Dreg_lo ;
W [Preg] = Dreg_lo ;
W [Preg] = Dreg ;
W [Preg ++] = Dreg ;
W [Preg --] = Dreg ;
W [Preg + uimm5m2] = Dreg ;
W [Preg + uimm16m2] = Dreg ;
W [Preg – uimm16m2] = Dreg ;
W [Preg ++ Preg] = Dreg_lo ;
B [Preg] = Dreg ;
B [Preg ++] = Dreg ;
B [Preg --] = Dreg ;
B [Preg + uimm15] = Dreg ;
B [Preg – uimm15] = Dreg ;
Preg = imm7 (X) ;
Preg = imm16 (X) ;
Preg += Preg (BREV) ;
Ireg += Mreg (BREV) ;
Preg = Preg << 2 ;
Preg = Preg >> 2 ;

表 5-3.　DAG 命令のまとめ（続き）

命令
Preg = Preg >> 1 ;
Preg = Preg + Preg << 1 ;
Preg = Preg + Preg << 2 ;
Preg –= Preg ;
Ireg –= Mreg ;

第6章　メモリ

このプロセッサは、階層構造内のメモリ位置に応じてさまざまな性能パラメータとサイズ・パラメータを持つ階層型メモリ・モデルに対応しています。レベル1（L1）メモリはチップ上にあり、レベル2（L2）メモリ・システムより高速です。レベル2（L2）メモリはオフチップであり、アクセス遅延が大きくなります。高速なL1メモリは一般に小さなスクラッチパッド・メモリまたはキャッシュ・メモリであり、コア自身の中にあります。

メモリ・アーキテクチャ

このプロセッサには、オンチップ／オフチップ・メモリとメモリマップドI/Oリソースの組合わせにまたがる4Gバイトの統一されたアドレス範囲があります。この範囲のうち、アドレス空間の一部は内部のオンチップ・リソース専用です。プロセッサは、この内部メモリ空間に以下のメモリを実装します。

- L1スタティック・ランダム・アクセス・メモリ（SRAM）
- 一連のメモリマップド・レジスタ（MMR）
- ブート読出し専用メモリ（ROM）

内部L1 SRAMの一部は、キャッシュとして動作するようにも設定できます。プロセッサは非同期メモリ空間とシンクロナスDRAM（SDRAM）空間を含む、外部メモリ空間もサポートします。これらのメモリ領域とそれをサポートするコントローラの詳細については、第17章「外部バス・インターフェース・ユニット」を参照してください。

6-3ページの図6-1は、ADSP-BF533プロセッサのシステム・メモリ・マップの概要を示しています。6-4ページの図6-2は、ADSP-BF532プロセッサの情報を示し、6-5ページの図6-3は、ADSP-BF531プロセッサの情報を示します。なお、このアーキテクチャでは、別個のI/O空間を定義しません。すべてのリソースは、フラットな32ビットのアドレス空間を通じてマッピングされます。メモリはバイト・アドレス可能です。

表 6-1に示すように、ADSP-BF533、ADSP-BF532、ADSP-BF531の各プロセッサでは、さまざまな命令設定とデータ・メモリ設定が可能です。

表 6-1.　メモリ設定

メモリのタイプ	ADSP-BF531	ADSP-BF532	ADSP-BF533
命令 SRAM ／キャッシュ、ウェイまたはラインによってロック可能	16K バイト	16K バイト	16K バイト
命令 SRAM	16K バイト	32K バイト	64K バイト
データ SRAM ／キャッシュ	16K バイト	32K バイト	32K バイト
データ SRAM	–	–	32K バイト
データ・スクラッチパッド SRAM	4K バイト	4K バイト	4K バイト
合計	**84K バイト**	**116K バイト**	**148K バイト**

内部メモリ空間の上位部分は、コアとシステムMMRに割り当てられます。この領域へのアクセスが許可されるのは、プロセッサがスーパーバイザ・モードまたはエミュレーション・モードにあるときだけです（第3章「動作モードと状態」を参照）。

内部メモリ空間の最下位1Kバイトはブート ROMによって占有されます。選択したブート・オプションに応じて、プロセッサのリセット時にメモリ空間から適切なブート・プログラムが実行されます（3-19ページの「ブート方式」を参照）。

外部メモリ・マップでは、4バンクの非同期メモリ空間と1バンクのSDRAMメモリを使用できます。各非同期バンクは1Mバイトであり、SDRAMバンクは128Mバイトまでです。

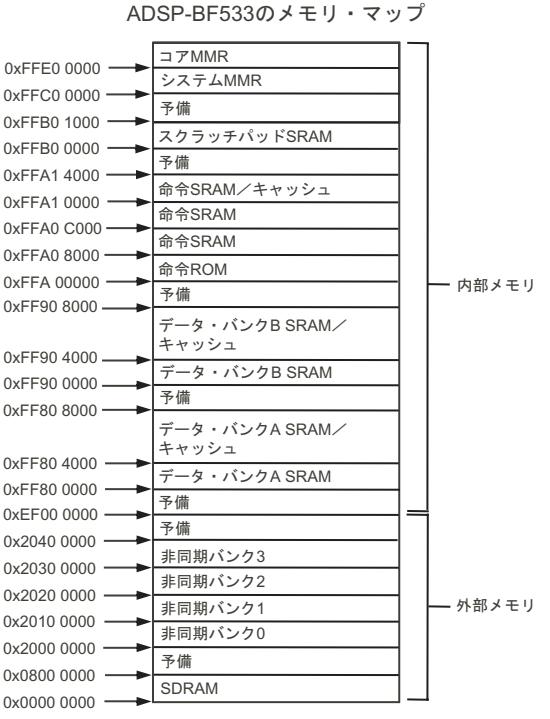


図 6-1.　ADSP-BF533 のメモリ・マップ

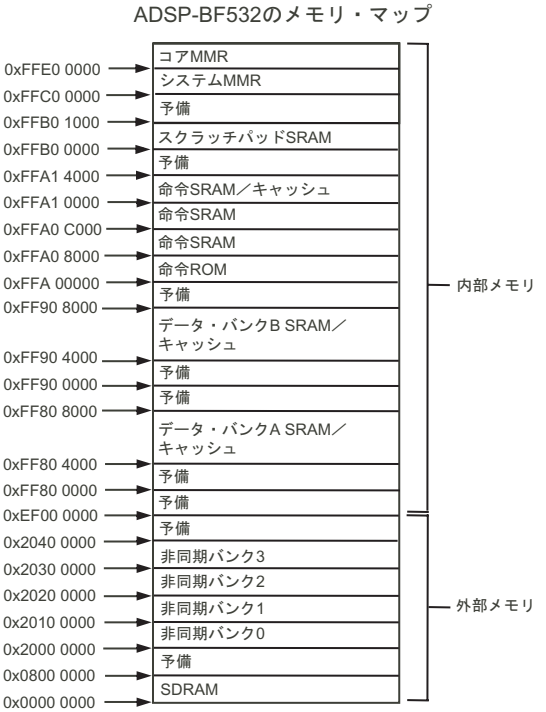


図 6-2.　ADSP-BF532 のメモリ・マップ

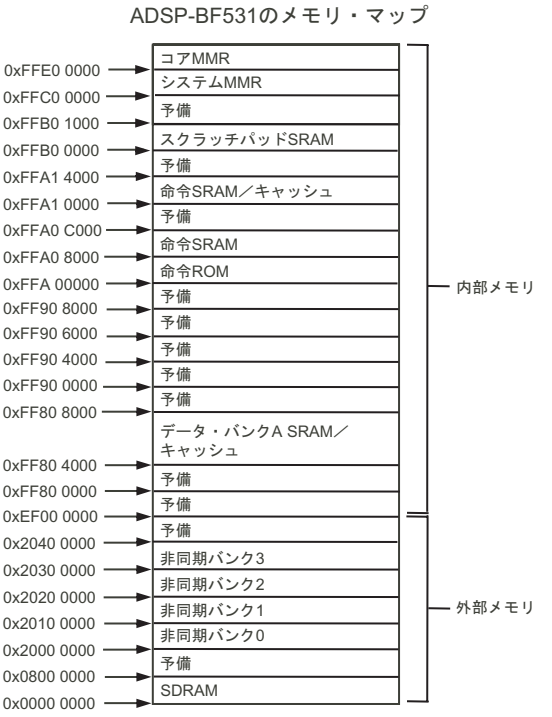


図 6-3.　ADSP-BF531 のメモリ・マップ

■ 内部メモリの概要

L1メモリ・システムの性能は高帯域と低遅延をもたらします。SRAMは決定論的なアクセス・タイムときわめて高いスループットを提供するため、DSPシステムでは高速なSRAMをオンチップで提供することによって、伝統的に性能を向上させてきました。

命令キャッシュとデータ・キャッシュ（キャッシュ制御ハードウェア付きのSRAM）の追加によって、高性能と簡単なプログラミング・モデルが得られます。キャッシュによって、L1メモリとの間でのデータ転送を明示

的に管理する必要がなくなります。メモリ構造に合わせて性能を最適化することなく、短時間でコードをプロセッサに移植したり、プロセッサ向けに開発したりできます。

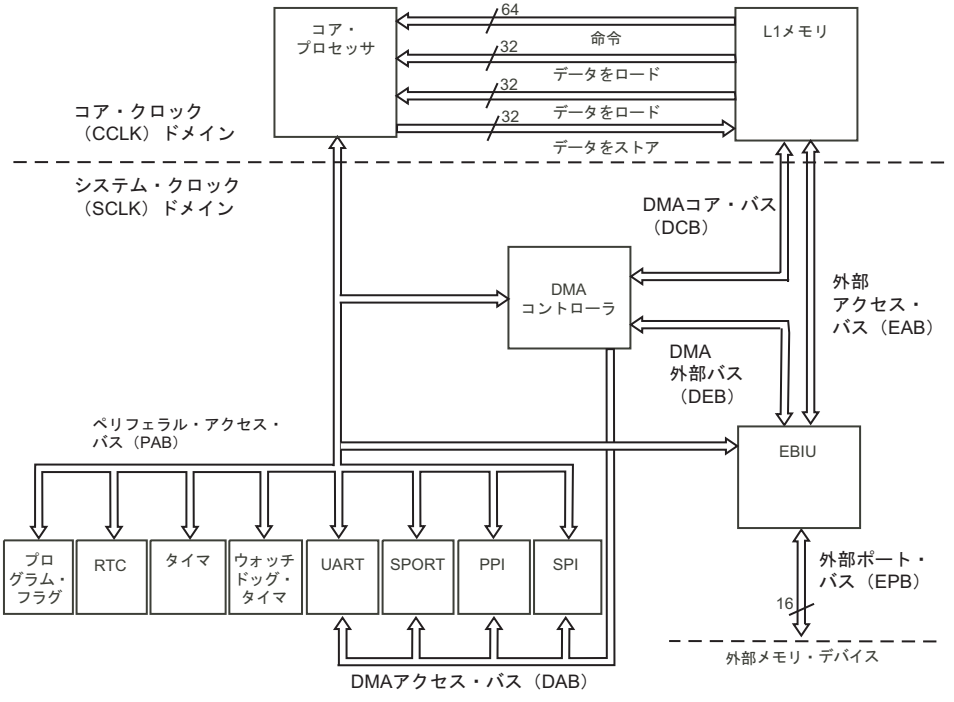


図 6-4.　プロセッサのメモリ・アーキテクチャ

L1メモリは以下のメリットを提供します。

- 改良ハーバード・アーキテクチャにより、クロック・サイクル当たり最高4つのコア・メモリ・アクセスが可能（1つの64ビット命令フェッチ、2つの32ビット・データ・ロード、1つのパイプライン型32ビット・データ・ストア）
- システムDMA、キャッシュ・メンテナンス、コア・アクセスが同時に可能
- クリティカルなDSPアルゴリズムと高速なコンテキスト切り替えを実現する、プロセッサ・クロック・レート（CCLK）でのSRAMアクセス
- マイクロコントローラ・コード用の命令オプションとデータ・キャッシュ・オプション、優れた高水準言語（HLL）サポート、PREFETCHやFLUSHなどのプログラミングしやすいキャッシュ制御命令
- メモリ保護

L1メモリは、コア・クロック周波数（CCLK）で動作します。

■ スクラッチパッド・データ SRAM の概要

このプロセッサは、スクラッチパッド・データSRAMという専用の4Kバイト・バンクを提供します。スクラッチパッドは他のL1メモリ・バンクの設定と独立しており、キャッシュとして設定したり、DMAのターゲットにしたりすることはできません。一般的なアプリケーションでは、速度が重要視される場合にスクラッチパッド・データ・メモリを使用します。

たとえば、割込み処理中に最速のコンテキスト切替えを実現するには、ユーザ・スタックとスーパーバイザ・スタックをスクラッチパッド・メモリにマッピングしてください。

L1メモリは、コア・クロック周波数（CCLK）で動作します。

スクラッチパッド・データSRAMは、DMAコントローラによってはアクセスできません。

L1 命令メモリ

L1命令メモリは、専用SRAMと、SRAMまたはキャッシュとして設定できるバンクとの組合わせで構成されています。キャッシュまたはSRAMとすることのできる16Kバイト・バンクの場合、IMEM_CONTROLレジスタの制御ビットを使用して、L1命令メモリの4つのサブバンクすべてを次のように構成できます。

- 単純なSRAM
- 4ウェイのセット・アソシエイティブ命令キャッシュ
- 4つのロックされたウェイを持つキャッシュ

L1命令メモリは、命令の格納にだけ使用できます。

■ IMEM_CONTROL レジスタ

命令メモリ・コントロール（IMEM_CONTROL）レジスタには、L1命令メモリの制御ビットが含まれています。リセット後のデフォルトでは、キャッシュおよびキャッシュ可能性保護Lookasideバッファ（CPLB）のアドレス・チェックがディスエーブルにされます（6-13ページの「L1命令キャッシュ」を参照）。

LRUPRIORSTビットが1に設定されると、すべてのCPLB_LRUPRIOビットのキャッシュ状態（6-61ページの「ICPLB_DATAxレジスタ」を参照）がクリアされます。これによって、キャッシュされたすべてのラインが同時に同じ（低い）重要度に設定されます。キャッシュ置換ポリシーは、まずCPLB_LRUPRIOビットのキャッシュ状態によって示されるライン重要度をベースにし、次にLRU（最長時間未使用）をベースにします。詳細については、6-20ページの「ラインによる命令キャッシュのロック」を参照してください。新しいラインがキャッシュされたときにCPLB_LRUPRIOビットの状態を格納するには、このビットは0であることが必要です。

ILOC[3:0]ビットは、コードが手動でキャッシュにロードされた後でだけ有効な機能を提供します。6-21ページの「ウェイによる命令キャッシュのロック」を参照してください。これらのビットは、キャッシュ置換ポリシーから除外するウェイを指定します。これには、キャッシュ置換ポリシーに参加していないウェイに存在するコードをロックする効果があります。IFLUSH命令を使用すれば、参加していないウェイ内のコードを依然としてキャッシュから除去できます。ILOC[3:0]ビットが0である場合には、対応するウェイはロックされず、そのウェイはキャッシュ置換ポリシーに参加します。ILOC[3:0]ビットが1である場合には、対応するウェイはロックされ、キャッシュ置換ポリシーに参加しません。

IMCビットは、L1命令SRAMの一部を確保してキャッシュとして機能させます。なお、キャッシュとして機能するメモリを確保するだけでは、L2メモリ・アクセスをキャッシュすることはできません。EN_ICPLBビットを使用してCPLBをイネーブルにし、CPLBディスクリプタ（ICPLB_DATAxレジスタとICPLB_ADDRxレジスタ）で希望するメモリ・ページをキャッシュ使用可能として指定することも必要です。

リセット後のデフォルトでは、命令CPLBはディスエーブルにされています。ディスエーブルにされている場合、L1メモリ・インターフェースでは最小限のアドレス・チェックしか行いません。この最小限のチェックでは、以下の領域から命令をフェッチするたびにプロセッサへの例外が生成されます。

- 予備（非実装）の L1 命令メモリ空間
- L1 データ・メモリ空間
- MMR 空間

このビットを使用して CPLB をディスエーブルにしてから、CPLB のディスクリプタ（レジスタ DCPLB_DATAx と DCPLB_ADDRx）を更新することが必要です。なお、ロード／ストアの順序付けは弱いため（6-73 ページの「ロードとストアの順序付け」を参照）、CSYNC 命令を実行してから CPLB をディスエーブルにしてください。

キャッシュや CPLB をイネーブル／ディスエーブルにする際には、IMEM_CONTROL への書込みの直後に SSYNC を実行して正常な動作を保証します。

正常な動作と将来の互換性を保証するため、このレジスタ内のすべての予備ビットは、このレジスタが書き込まれるたびに 0 に設定する必要があります。

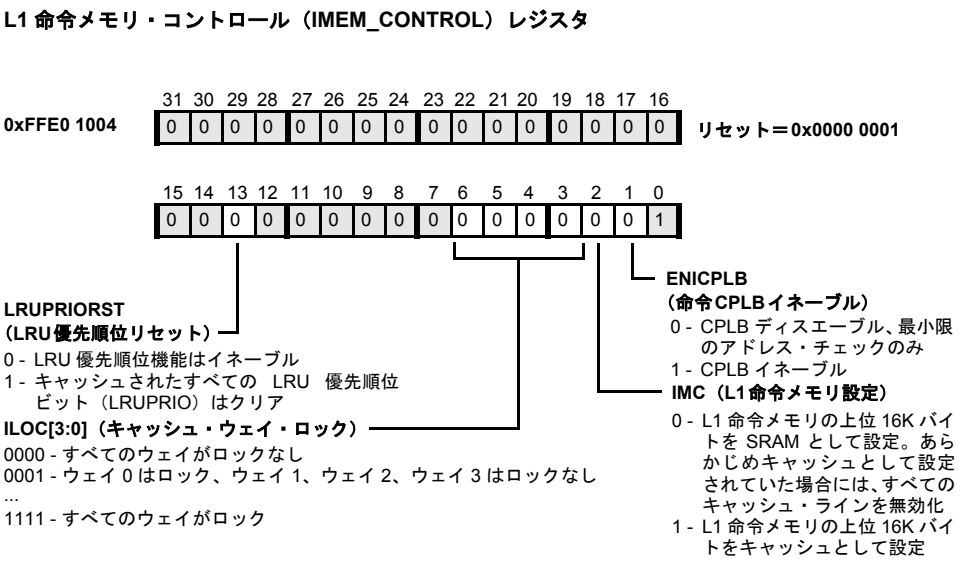


図 6-5. L1 命令メモリ・コントロール・レジスタ

■ L1 命令 SRAM

プロセッサ・コアは、64ビット幅の命令フェッチ・バスを通じて命令メモリを読み出します。このバスからのすべてのアドレスは、64ビットに整列されています。各命令フェッチでは、16/32/64ビット命令の任意の組合わせを返すことができます（たとえば、4つの16ビット命令、2つの16ビット命令と1つの32ビット命令、または1つの64ビット命令）。

第5章で説明したDAGは、L1命令メモリに直接アクセスできません。命令メモリSRAM空間へのDAGリファレンスは、例外を生成します（4-42ページの「例外」を参照）。

L1命令SRAMメモリへの書込みアクセスは、64ビット幅のシステムDMAポートを通じて行う必要があります。SRAMはシングル・ポート型サブバンクの集合として実装されるため、命令メモリは実質的にデュアル・ポート型となります。

表 6-2は、L1命令メモリ・サブバンクのメモリ開始位置を示します。

表 6-2.　L1 命令メモリ・サブバンク

メモリ・サブバンク	メモリ開始位置 ADSP-BF533	メモリ開始位置 ADSP-BF532	メモリ開始位置 ADSP-BF531
0	0xFFA0 0000	0xFFA0 8000	0xFFA0 8000
1	0xFFA0 1000	0xFFA0 9000	0xFFA0 9000
2	0xFFA0 2000	0xFFA0 A000	0xFFA0 A000
3	0xFFA0 3000	0xFFA0 B000	0xFFA0 B000
4	0xFFA0 4000	0xFFA0 C000	
5	0xFFA0 5000	0xFFA0 D000	
6	0xFFA0 6000	0xFFA0 E000	
7	0xFFA0 7000	0xFFA0 F000	
8	0xFFA0 8000		
9	0xFFA0 9000		
10	0xFFA0 A000		
11	0xFFA0 B000		
12	0xFFA0 C000		
13	0xFFA0 D000		

表 6-2.　L1 命令メモリ・サブバンク（続き）

メモリ・サブバンク	メモリ開始位置 ADSP-BF533	メモリ開始位置 ADSP-BF532	メモリ開始位置 ADSP-BF531
14	0xFFA0 E000		
15	0xFFA0 F000		

6-14ページの図6-6は、L1命令メモリのバンク・アーキテクチャを示します。この図に示すように、各16Kバイト・バンクは4つの4Kバイト・サブバンクで構成されます。

■ L1 命令キャッシュ

キャッシュ用語の詳細については、6-80ページの「用語集」を参照してください。

L1命令メモリは、16Kバイトの4ウェイ・セット・アソシエイティブ命令キャッシュを含むようにも構成できます。クリティカルなコード・セクションに対する平均アクセス遅延を改善するため、キャッシュの各ウェイまたはラインは、独立してロックできます。メモリがキャッシュとして設定されている場合、メモリには直接アクセスできません。

キャッシュがイネーブルであるとき、CPLBによってさらにキャッシュ可能として指定されたメモリ・ページだけがキャッシュされます。CPLBがイネーブルであるとき、アクセスされるメモリ位置では、関連するページ定義が使用可能でなければいけません。さもなければ、CPLB例外が生成されます。CPLBについては、6-51ページの「メモリ保護とプロパティ」を参照してください。

6-16ページの図6-7は、Blackfinプロセッサの全体的な命令キャッシュ構造を示します。

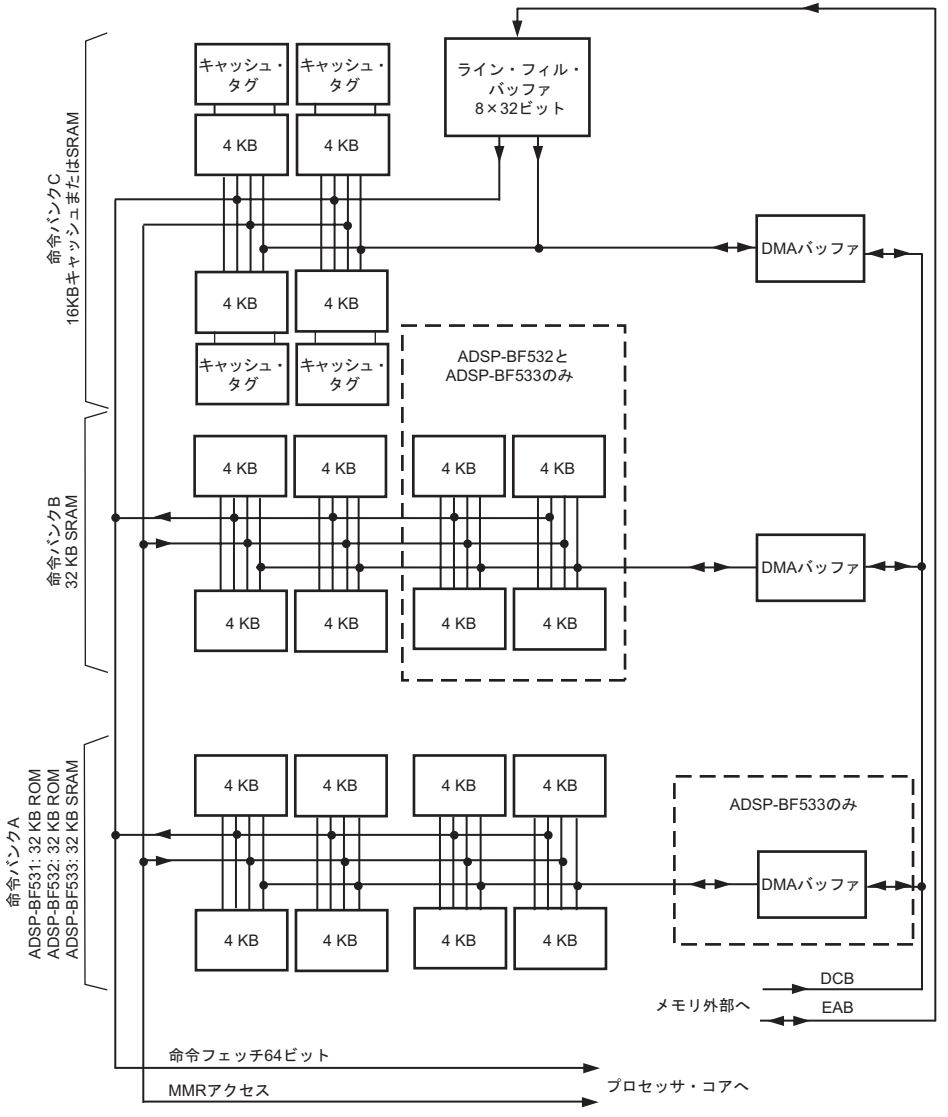


図 6-6. L1 命令メモリのバンク・アーキテクチャ

▶ キャッシュ・ライン

図 6-7に示すように、キャッシュは、キャッシュ・ラインの集合から構成されます。各キャッシュ・ラインは、**タグ**・コンポーネントと**データ**・コンポーネントで構成されます。

- タグ・コンポーネントは、20 ビットのアドレス・タグ、最長時間未使用（LRU）ビット、有効ビット、ライン・ロック・ビットを内蔵しています。
- データ・コンポーネントは、命令データの4つの64ビット・ワードで構成されます。

キャッシュ・ラインのタグ・コンポーネントとデータ・コンポーネントは、それぞれタグ・メモリ・アレイとデータ・メモリ・アレイに格納されています。

アドレス・タグは物理アドレスの上位18ビットに加えて、ビット11と10から構成されます。物理アドレスのビット12と13は、アドレス・タグの一部ではありません。これらのビットは、アクセスのターゲットとなる4Kバイトのメモリ・サブバンクの識別に使用されます。

LRUビットはLRUアルゴリズムの一部であり、キャッシュ・ミスが発生した場合に置換されるキャッシュ・ラインの決定に使用されます。

有効ビットは、キャッシュ・ラインの状態を示します。キャッシュ・ラインは常に有効または無効です。

- 無効なキャッシュ・ラインでは有効ビットがクリアされて、アドレス—タグの比較演算でそのラインが無視されることを示します。
- 有効なキャッシュ・ラインでは有効ビットがセットされて、そのラインにはソース・メモリと整合性のある有効な命令／データが含まれていることを示します。

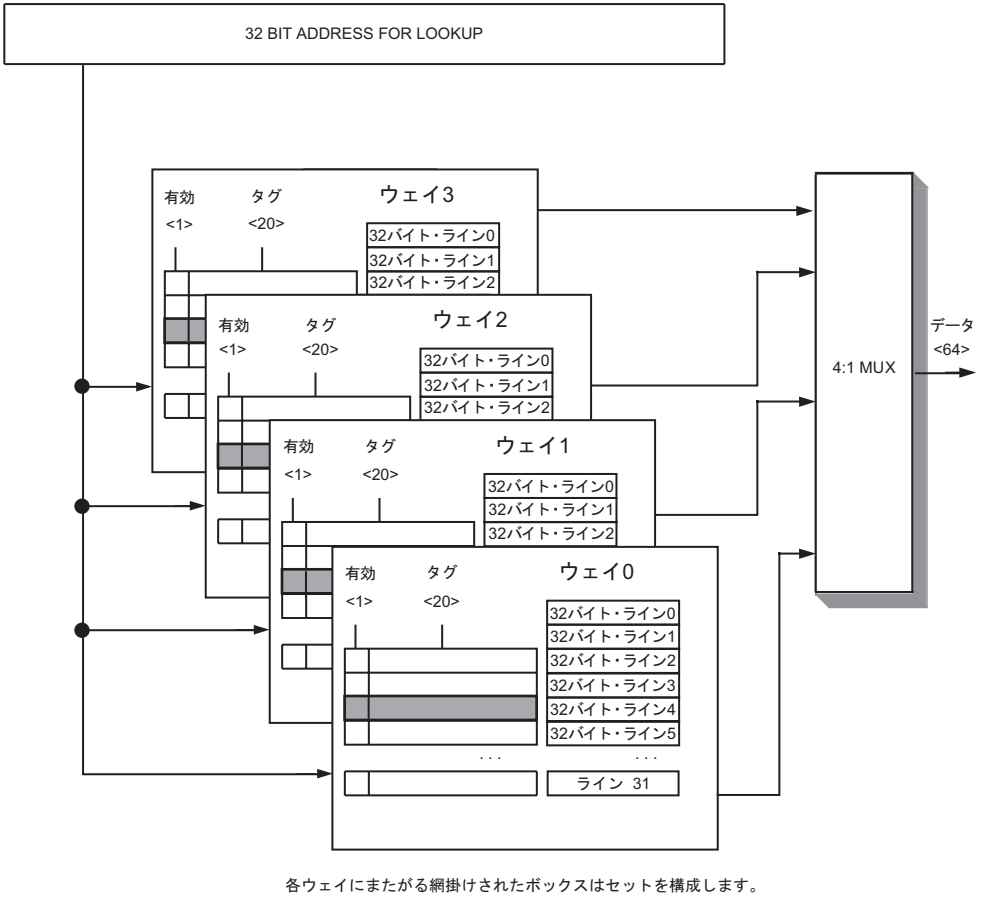


図 6-7.　Blackfin プロセッサの命令キャッシュ構造

図 6-8は、キャッシュ・ラインのタグ・コンポーネントとデータ・コンポーネントを示します。

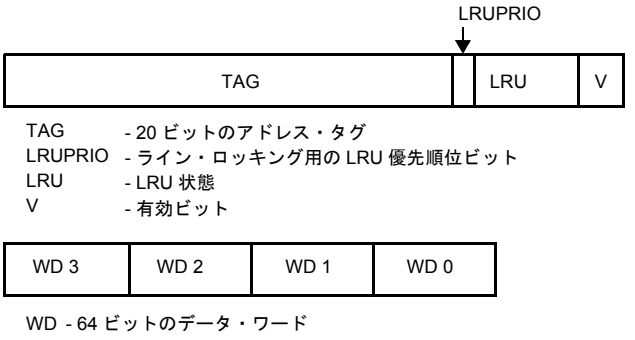


図 6-8.　キャッシュ・ライン — タグ部分とデータ部分

キャッシュ・ヒットとキャッシュ・ミス

キャッシュ・ヒットが発生するのは、コアからの命令フェッチ要求のアドレスがキャッシュ内の有効エントリに一致したときです。具体的に、キャッシュ・ヒットの判定には、命令フェッチ・アドレスの上位18ビットとビット11および10をキャッシュ・セット内に現在格納されている有効ラインのアドレス・タグと比較します。キャッシュ・セットの選択には、命令フェッチ・アドレスのビット9から5が使用されます。アドレス—タグ比較演算の結果が一致した場合には、キャッシュ・ヒットが発生します。アドレス—タグ比較演算の結果が一致しない場合には、キャッシュ・ミスが発生します。

キャッシュ・ミスが発生すると、命令メモリ・ユニットはキャッシュ・ライン・フィル・アクセスを生成して、コアの外部にあるメモリから不足しているキャッシュ・ラインを取り出します。外部メモリ・アクセスのアドレスは、ターゲット命令ワードのアドレスです。キャッシュ・ミスが発生すると、コアは外部メモリからターゲット命令ワードが返されるまで停止します。

キャッシュ・ライン・フィル

キャッシュ・ライン・フィルは、メモリから32バイトのデータを取り出します。この動作が開始されるのは、命令メモリ・ユニットが外部読出しデータ・ポートでライン読出しデータ転送（4つの64ビット・ワードのデータのバースト）を要求したときです。読出し転送のアドレスは、ターゲット命令ワードのアドレスです。外部メモリは、命令メモリ・ユニットからのライン読出し要求に応答するとき、最初にターゲット命令ワードを返します。ターゲット命令ワードが返されてから、次の3ワードが順次アドレス順にフェッチされます。表 6-3に示すように、必要ならば、このフェッチは折り返されます。

表 6-3.　キャッシュ・ライン・ワードのフェッチ順

ターゲット・ワード	次の 3 ワードのフェッチ順序
WD0	WD0, WD1, WD2, WD3
WD1	WD1, WD2, WD3, WD0
WD2	WD2, WD3, WD0, WD1
WD3	WD3, WD0, WD1, WD2

ライン・フィル・バッファ

新しいキャッシュ・ラインが外部メモリから取り出されると、各64ビット・ワードは4エントリのライン・フィル・バッファにバッファリングされてから、L1メモリ内の4Kバイトのメモリ・バンクに書き込まれます。ライン・フィル・バッファを使用すれば、コアはラインがキャッシュに書き込まれるまで待つことなく、ラインが外部メモリから取り出されると同時に新しいキャッシュ・ラインからのデータにアクセスできます。

キャッシュ・ライン置換

命令メモリ・ユニットがキャッシュとして設定されている場合、命令フェッチ・アドレスのビット9から5は、タグ―アドレス比較演算用のキャッシュ・セットを選択するインデックスとして使用されます。タグ―

アドレス比較演算の結果がキャッシュ・ミスになった場合には、キャッシュ・ライン置換ユニットは選択されたセットの有効ビットとLRUビットを調べて、新しいキャッシュ・ラインに使用するエントリ（つまり、ウェイ0、ウェイ1、ウェイ2、またはウェイ3の使用）を決定します。6-16ページの図6-7「Blackfinプロセッサの命令キャッシュ構造」を参照してください。

キャッシュ・ライン置換ユニットは、最初に無効なエントリ（つまり、有効ビットがクリアされているエントリ）をチェックします。無効なエントリが1つだけ見つかった場合には、そのエントリが新しいキャッシュ・ラインに選択されます。無効なエントリが複数件見つかった場合には、新しいキャッシュ・ライン用の置換エントリは、次の優先順位に基づいて選択されます。

- 最初にウェイ0
- 次にウェイ1
- 次にウェイ2
- 最後にウェイ3

例：

- ウェイ3が無効で、ウェイ0、1、2が有効である場合には、ウェイ3が新しいキャッシュ・ラインに選択されます。
- ウェイ0と1が無効で、ウェイ2と3が有効である場合には、ウェイ0が新しいキャッシュ・ラインに選択されます。
- ウェイ2と3が無効で、ウェイ0と1が有効である場合には、ウェイ2が新しいキャッシュ・ラインに選択されます。

無効なエントリが見つからなかった場合には、キャッシュ置換ロジックは、LRUアルゴリズムを使用します。

▶ 命令キャッシュの管理

システムDMAコントローラとコアDAGは、命令キャッシュに直接アクセスできません。命令の組合わせとコアMMRの使用によって、命令タグとデータ・アレイを間接的に初期化し、命令キャッシュのテスト、初期化、およびデバッグ用のメカニズムを提供できます。

命令キャッシュのコヒーレンシは、明示的に管理する必要があります。これを実現し、命令キャッシュが最新バージョンの修正命令空間をフェッチできるようにするには、必要に応じて命令キャッシュ・ラインのエントリを無効化します。

6-22ページの「命令キャッシュの無効化」を参照してください。

ラインによる命令キャッシュのロック

ICPLB_DATAxレジスタ（6-51ページの「メモリ保護とプロパティ」を参照）のCPLB_LRUPRIOビットは、コードを命令キャッシュ内に常駐させて制御を強化するために使用されます。キャッシュ・ラインがフィルされると、このビットの状態はラインのタグとともに格納されます。その後、新しいキャッシュ可能ラインのフェッチ時にすべてのキャッシュ・ウェイが占有された場合に犠牲になるウェイを決定するために、このビットの状態はLRU（最長時間未使用）ポリシーと組み合わせて使用されます。このビットは、ラインの重要度が「低」であるか「高」であるかを示します。改良LRUポリシーでは、「高」は「低」に取って代われますが、「低」が「高」に取って代わることはできません。すべてのウェイが「高」によって占有された場合、その他の点ではキャッシュ可能な「低」は依然としてコア用にフェッチされますが、キャッシュはされません。フェッチされた「高」は、まず占有されていないウェイでの置換を試み、次に最長時間未使用の「低」で試み、最後にLRUポリシーを使用して他の「高」で試みます。「低」は、占有されていないウェイまたは他の「低」だけを置換でき、その際にLRUポリシーを使用します。あらかじめキャッシュされていた**すべて**の

「高」で、その重要度が低下した場合には、IMEM_CONTROL レジスタ（6-8 ページを参照）の LRUPRIRST ビットに書き込むことによって、これらを同時に「低」に変換することもできます。

ウェイによる命令キャッシュのロック

命令キャッシュには、4つの独立したロック・ビット（ILOC[3:0]）があり、それぞれが命令キャッシュの4つのウェイを制御します。キャッシュがイネーブルにされると、L1命令メモリでは4つのウェイが使用可能になります。特定のウェイのロック・ビットをセットすると、そのウェイはLRU置換ポリシーに参加できなくなります。したがって、キャッシュされた命令のウェイがロックされている場合、その命令を除去するには、IFLUSH命令を使用するか、またはタグ・アレイの「裏口」MMR支援操作を行うしか方法がありません。

次のシーケンス例は、ウェイ0のロック方法を示します。

- 対象となるコードがすでに命令キャッシュ内に存在する可能性がある場合には、最初にキャッシュ全体を無効化します（一例として、6-22ページの「命令キャッシュの無効化」を参照）。
- 必要ならば、割込みをディスエーブルにして、割込みサービス・ルーチン（ISR）がロックされたキャッシュを破壊しないようにします。
- ILOC[3:1]をセットして、キャッシュの他のウェイに対してロックを設定します。これで、命令キャッシュのウェイ0だけが新しいコードで置換できます。
- 対象となるコードを実行します。このコード実行によって走査された終了コードなどのキャッシュ可能なその他の命令も、命令キャッシュに組み込まれます。

- クリティカルなコードを終了した時点で ILOC[3:1] をクリアし、ILOC[0]をセットします。これで、クリティカルなコード(とILOC[0]をセットした命令）は、ウェイ0に組み込まれます。
- 必要ならば、割込みを再びイネーブルにします。

キャッシュの4つのウェイがすべてロックされている場合には、キャッシュへのそれ以上の割当てはできません。

命令キャッシュの無効化

命令キャッシュは、アドレス単位、キャッシュ・ライン単位、またはキャッシュ全体で無効化できます。IFLUSH命令は、ライン・アドレスに基づいてキャッシュ・ラインを明示的に無効化できます。命令のターゲット・アドレスはPレジスタから生成されます。命令キャッシュは変更された（ダーティな）データを含むべきではないため、キャッシュ・ラインはそのまま無効化されます。

次の例では、P2レジスタには、有効なメモリ位置のアドレスが含まれています。このアドレスがキャッシュに入れられた場合には、対応するキャッシュ・ラインはこの命令の実行後に無効化されます。

ICACHE命令の例：

```
iflush [ p2 ] ;    /* P2がポイントするアドレスを含んでいるキャッシュ・ラインを無効化 */
```

IFLUSH命令はメモリ・マップ内の特定アドレスを無効化するために使用されるので、この命令を使用してキャッシュのウェイまたはバンク全体を無効化することは現実的ではありません。キャッシュの大きな部分を直接無効化するには、2番目の手法を使用できます。この2番目の手法では、各キャッシュ・ラインの無効ビットに無効状態を設定することによって、有効ビットを直接無効化します。この手法を実現するため、別のMMR

（ITEST_COMMAND と ITEST_DATA[1:0]）を使用すれば、すべてのキャッシュ・エントリに対して直接の読出し／書込みが可能になります。この方法については、次の節で説明します。

命令キャッシュを全面的に無効化するには、3番目の方法を使用します。IMEM_CONTROL レジスタ（6-11ページの図6-5「L1命令メモリ・コントロール・レジスタ」を参照）の IMC ビットをクリアすれば、命令キャッシュ内のすべての有効ビットが無効状態に設定されます。IMEM_CONTROL レジスタへの2回目の書込みで IMC ビットをセットすれば、命令メモリは再びキャッシュとして設定されます。なお、SSYNC 命令を実行してからキャッシュを無効化してください。また、これらの動作が終了するたびに、CSYNC 命令を挿入してください。

命令テスト・レジスタ

命令テスト・レジスタを使用すれば、すべてのL1キャッシュ・エントリに対して直接の読出し／書込みが可能になります。これらのレジスタによって、命令タグ・アレイとデータ・アレイの初期化が可能になり、命令キャッシュをテスト、初期化、およびデバッグするためのメカニズムが提供されます。

命令テスト・コマンド・レジスタ（ITEST_COMMAND）を使用すると、L1キャッシュのデータ・アレイやタグ・アレイがアクセスされ、データは命令テスト・データ・レジスタ（ITEST_DATA[1:0]）を通じて転送されます。ITEST_DATAx レジスタには、アクセスによって書き込む64ビット・データ、またはアクセス中に読み出された64ビット・データが含まれています。下位 32 ビットは ITEST_DATA[0] レジスタに格納され、上位 32 ビットは ITEST_DATA[1] レジスタに格納されます。タグ・アレイがアクセスされるときには ITEST_DATA[0] が使用されます。ITEST レジスタについては、6-25ページの図6-9から説明が始まります。

ITEST レジスタの説明は、以下のようになります。

- 6-25ページの図6-9「命令テスト・コマンド・レジスタ」
- 6-26ページの図6-10「命令テスト・データ1レジスタ」
- 6-27ページの図6-11「命令テスト・データ0レジスタ」

これらのレジスタへのアクセスは、スーパーバイザ・モードまたはエミュレーション・モードでのみ可能です。ITEST レジスタへの書込みに際しては、最初にITEST_DATAx レジスタに書き込んでから、ITEST_COMMAND レジスタに書き込みます。ITEST レジスタからの読出しに際しては順番が逆になります。つまり、最初にITEST_COMMAND レジスタを読み出してから、ITEST_DATAx レジスタを読み出します。

■ ITEST_COMMAND レジスタ

命令テスト・コマンド・レジスタ（ITEST_COMMAND）への書込みに際しては、L1キャッシュのデータ・アレイまたはタグ・アレイがアクセスされ、データは命令テスト・データ・レジスタ（ITEST_DATA[1:0]）を通じて転送されます。

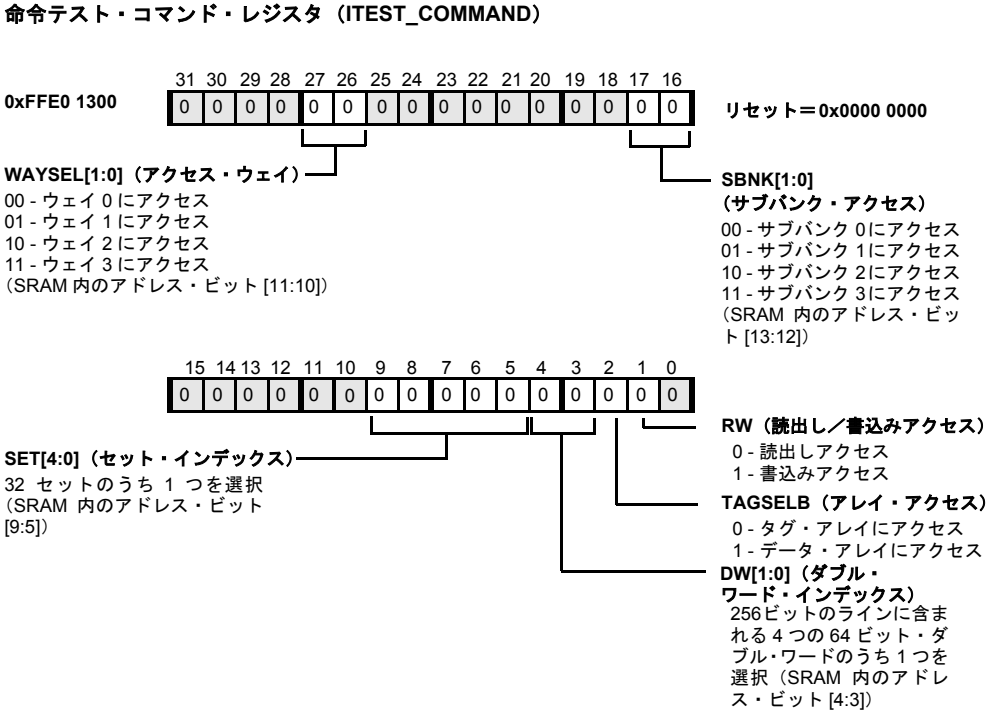


図 6-9.　命令テスト・コマンド・レジスタ

■ ITEST_DATA1 レジスタ

命令テスト・データ・レジスタ（ITEST_DATA[1:0]）は、L1キャッシュのデータ・アレイへのアクセスに使用されます。これらのレジスタには、アクセスによって書き込む64ビット・データ、またはアクセスによって読み出す64ビット・データが含まれています。命令テスト・データ1レジスタ（ITEST_DATA1）には、上位32ビットが格納されています。

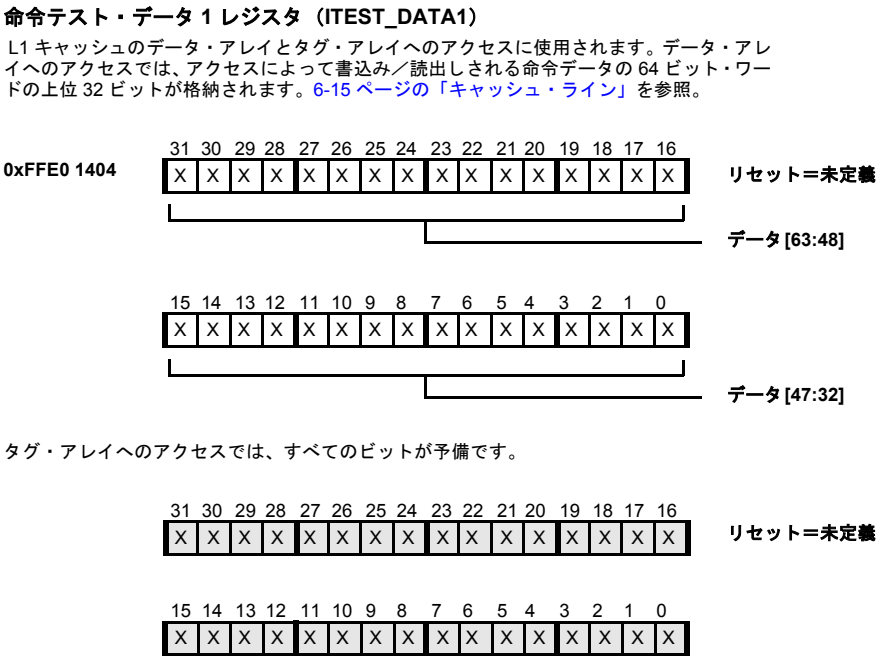


図 6-10. 命令テスト・データ 1 レジスタ

■ ITEST_DATA0 レジスタ

命令テスト・データ0レジスタ（ITEST_DATA0）には、アクセスによって書込み／読出しされる64ビット・データの下位32ビットが格納されます。ITEST_DATA0レジスタは、タグ・アレイへのアクセスにも使用されます。このレジスタには、キャッシュ・ラインの状態を示す、有効ビットとダーティ・ビットも含まれています。

命令テスト・データ 0 レジスタ（ITEST_DATA0）

L1 キャッシュのデータ・アレイとタグ・アレイへのアクセスに使用されます。データ・アレイへのアクセスでは、アクセスによって書込み／読出しされる命令データの 64 ビット・ワードの下位 32 ビットが格納されます。6-15 ページの「キャッシュ・ライン」を参照。

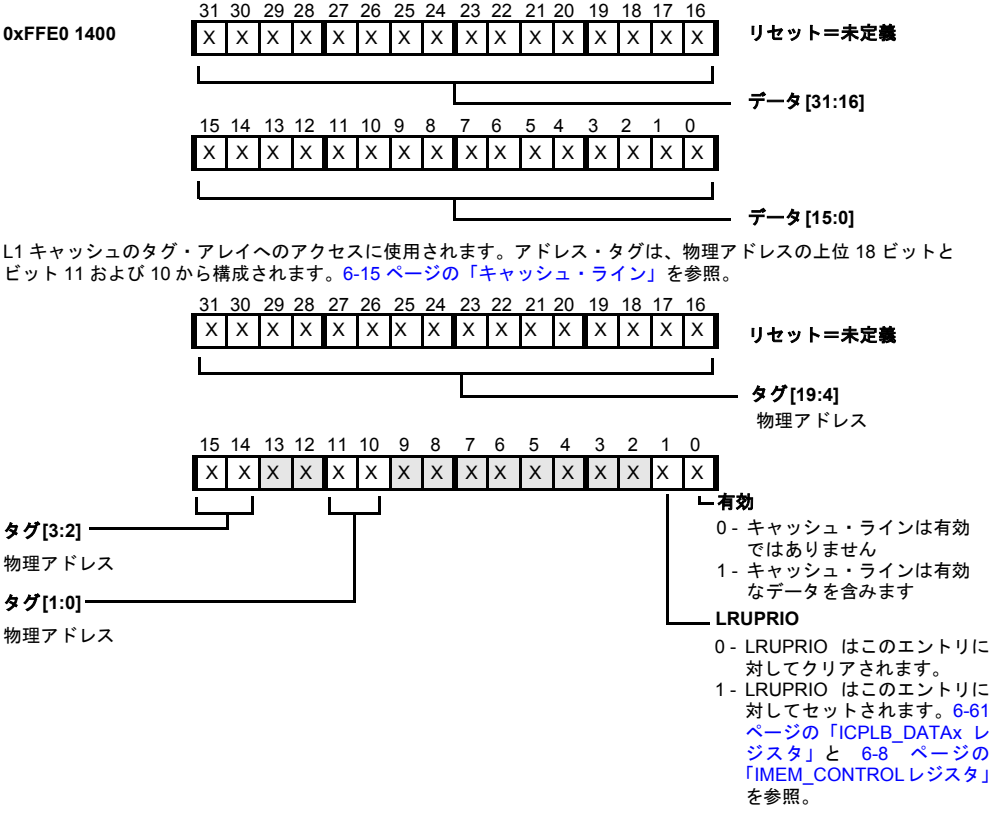


図 6-11. 命令テスト・データ 0 レジスタ

L1 データ・メモリ

L1データSRAM／キャッシュは、シングル・ポート型サブセクションから構築されますが、アクセス衝突の可能性を減らすように構成されています。この構成によって、見掛け上のマルチポート型動作が行われます。衝突がない場合には、以下のL1データ・トラフィックがシングル・コア・クロック・サイクルで発生することがあります。

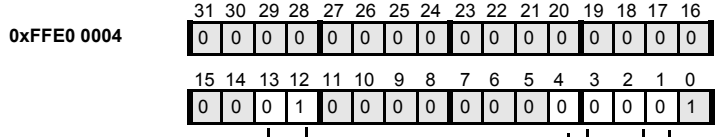
- 2つの32ビットDAGロード
- 1つのパイプライン化された32ビットDAGストア
- 1つの64ビットDMA IO
- 1つの64ビット・キャッシュ・フィル／ビクティム・アクセス

L1データ・メモリは、データの格納にだけ使用できます。

■ DMEM_CONTROL レジスタ

データ・メモリ・コントロール・レジスタ（DMEM_CONTROL）には、L1データ・メモリの制御ビットが含まれています。

データ・メモリ・コントロール・レジスタ（DMEM_CONTROL）



PORT_PREF1（DAG1ポート・プリファレンス）

0 - DAG1キャッシュ不可能フェッチでポート A を使用
1 - DAG1キャッシュ不可能フェッチでポート B を使用

PORT_PREF0（DAG0ポート・プリファレンス）

0 - DAG0キャッシュ不可能フェッチでポート A を使用
1 - DAG0キャッシュ不可能フェッチでポート B を使用

DCBS（L1データ・キャッシュ・バンク選択）

ADSP-BF532 と ADSP-BF533 では、DMC[1:0] = 11 の場合にのみ有効。L1 データ・キャッシュ・バンクの選択にアドレス・ビット A[14] または A[23] が使用されるかどうかを決定。

0 - アドレス・ビット 14 を使用して、キャッシュ・アクセス用にバンク A または B を選択。アドレスのビット 14 が 1 の場合、L1 データ・メモリのデータ・バンク A を選択。アドレスのビット 14 が 0 の場合、L1 データ・メモリのデータ・バンク B を選択。
1 - アドレス・ビット 23 を使用して、キャッシュ・アクセス用にバンク A または B を選択。アドレスのビット 23 が 1 の場合、L1 データ・メモリのデータ・バンク A を選択。アドレスのビット 23 が 0 の場合、L1 データ・メモリのデータ・バンク B を選択。

6-36 ページの「キャッシュ可能アドレス空間のマッピング例」を参照。

ENDCPLB（データ・キャッシュ可能性保護索引バッファ・イネーブル）

0 - CPLB はディスエーブル。最小限のアドレス・チェックのみ。
1 - CPLB はイネーブル

DMC[1:0]（L1データ・メモリ設定）

ADSP-BF533 の場合：
00 - 両方のデータ・バンクは SRAM。あらかじめキャッシュとして設定されている場合には、すべてのキャッシュ・ラインを無効化。
01 - 予備
10 - データ・バンク A は、下位 16K バイトの SRAM と上位 16K バイト・キャッシュ。データ・バンク B は SRAM。
11 - 両方のデータ・バンクは、下位 16K バイトの SRAM と上位 16K バイト・キャッシュ。

ADSP-BF532 の場合：
00 - 両方のデータ・バンクは SRAM。あらかじめキャッシュとして設定されている場合には、すべてのキャッシュ・ラインを無効化。
01 - 予備
10 - データ・バンク A はキャッシュ、データ・バンク B は SRAM
11 - 両方のデータ・バンクはキャッシュ

ADSP-BF531 の場合：
00 - データ・バンク A は SRAM。あらかじめキャッシュとして設定されている場合には、すべてのキャッシュ・ラインを無効化。
01 - 予備
10 - データ・バンク A はキャッシュ
11 - 予備

図 6-12. L1 データ・メモリ・コントロール・レジスタ

PORT_PREF1 ビットは、DAG1キャッシュ不可能L2フェッチの処理に使用されるデータ・ポートを選択します。キャッシュ可能フェッチは、ターゲットとなるキャッシュ・メモリに物理的に関連付けられたデータ・ポートによって常に処理されます。DAG0、DAG1、キャッシュ・トラフィックをさまざまなポートへ向け、L2メモリへのキューを満杯に保持することで性能を最適化します。

PORT_PREF0 ビットは、DAG0キャッシュ不可能L2フェッチの処理に使用されるデータ・ポートを選択します。キャッシュ可能フェッチは、ターゲットとなるキャッシュ・メモリに物理的に関連付けられたデータ・ポートによって常に処理されます。DAG0、DAG1、キャッシュ・トラフィックをさまざまなポートへ向け、L2メモリへのキューを満杯に保持することで性能を最適化します。

デュアルDAG読出しによる最適性能を得るには、DAG0とDAG1は異なるポートに設定してください。たとえば、PORT_PREF0が1に設定されている場合には、PORT_PREF1は0に設定してください。

DCBS ビットは、どのアドレスが同じセットにエイリアスされるかをいくらか制御できます。このビットを使用すれば、繰り返し使用されるセットの犠牲を回避することで、キャッシュ内に常駐しがちなアドレスに影響を与えることができます。ただし、データ・バンクAとデータ・バンクBの両方がキャッシュとして機能している必要があります（このレジスタのビットDMC[1:0]を11に設定）。

ENDCPLB ビットは、データ用に使用される16個のキャッシュ可能性保護索引バッファ（CPLB）をイネーブル／ディスエーブルにするために使用されます（6-35ページの「L1データ・キャッシュ」を参照）。データCPLBは、リセット後にデフォルトでディスエーブルにされます。ディスエーブルにされると、L1メモリ・インターフェースによって最小限のアドレス・チェックだけが実行されます。この最小限のチェックは、プロセッサが以下の動作を行ったときに例外を生成します。

- 存在しない（予備の）L1メモリ空間をアドレス指定

- 非整列のメモリ・アクセスを実行
- DAG1やユーザ・モードを利用してMMR空間にアクセス

このビットを使用してCPLBをディスエーブルにしてから、CPLBのディスクリプタ（レジスタDCPLB_DATAxとDCPLB_ADDRx）を更新することが必要です。なお、ロード／ストアの順序付けは弱いため（6-73ページの「ロードとストアの順序付け」を参照）、CSYNC命令を実行してからCPLBをディスエーブルにしてください。

キャッシュやCPLBをイネーブル／ディスエーブルにする際には、DMEM_CONTROLへの書込みの直後にSSYNCを実行して正常な動作を保証します。

リセット後のデフォルトによって、すべてのL1データ・メモリはSRAMとして機能します。DMC[1:0]ビットを使用すれば、このメモリの一部をキャッシュとして機能するように確保することができます。メモリをキャッシュとして機能するように確保しても、L2メモリ・アクセスはキャッシュされません。そのためには、CPLBもイネーブルにし（ENDCPLBビットを使用）、選択したメモリ・ページをCPLBディスクリプタ（レジスタDCPLB_DATAxとDCPLB_ADDRx）でキャッシュ・イネーブルとして指定する必要があります。

リセット後のデフォルトによって、キャッシュとCPLBのアドレス・チェックはディスエーブルにされます。

正常な動作と将来の互換性を保証するため、このレジスタ内のすべての予備ビットは、このレジスタが書き込まれるたびに0に設定する必要があります。

■ L1 データ SRAM

SRAMへのアクセスは衝突しません。ただし、そのアクセスが同じ32ビット・ワード極性（アドレス・ビット2が一致）、同じ4Kバイト・サブバンク（アドレス・ビット13と12が一致）、同じ16Kバイト・ハーフ・バンク

（アドレス・ビット16が一致）、同じバンク（アドレス・ビット21と20が一致）に対するものである場合を除きます。アドレス衝突が検出されると、アクセスは最初にDAGに対して許可され、次にストア・バッファに対して許可され、最後にDMAとキャッシュ・フィル／ビクティム・トラフィックに対して許可されます。十分なDMA帯域幅を保証するため、DMAが16を超える連続したコア・クロック・サイクルの間ブロックされていた場合、または2番目のDMA I/Oがキュー登録されてから最初のDMA I/Oが処理された場合、DMAには最高の優先順位が与えられます。

表 6-4は、サブバンク構造がメモリにマッピングされる様子を示します。

表 6-4.　L1 データ・メモリ SRAM サブバンクの開始アドレス

メモリ・バンクとサブバンク	ADSP-BF533	ADSP-BF532	ADSP-BF531
データ・バンク A、サブバンク 0	0xFF80 0000	–	–
データ・バンク A、サブバンク 1	0xFF80 1000	–	–
データ・バンク A、サブバンク 2	0xFF80 2000	–	–
データ・バンク A、サブバンク 3	0xFF80 3000	–	–
データ・バンク A、サブバンク 4	0xFF80 4000	0xFF80 4000	0xFF80 4000
データ・バンク A、サブバンク 5	0xFF80 5000	0xFF80 5000	0xFF80 5000
データ・バンク A、サブバンク 6	0xFF80 6000	0xFF80 6000	0xFF80 6000
データ・バンク A、サブバンク 7	0xFF80 7000	0xFF80 7000	0xFF80 7000
データ・バンク B、サブバンク 0	0xFF90 0000	–	–

表 6-4.　L1 データ・メモリ SRAM サブバンクの開始アドレス（続き）

メモリ・バンクとサブバンク	ADSP-BF533	ADSP-BF532	ADSP-BF531
データ・バンク B、サブバンク 1	0xFF90 1000	–	–
データ・バンク B、サブバンク 2	0xFF90 2000	–	–
データ・バンク B、サブバンク 3	0xFF90 3000	–	–
データ・バンク B、サブバンク 4	0xFF90 4000	0xFF90 4000	–
データ・バンク B、サブバンク 5	0xFF90 5000	0xFF90 5000	–
データ・バンク B、サブバンク 6	0xFF90 6000	0xFF90 6000	–
データ・バンク B、サブバンク 7	0xFF90 7000	0xFF90 7000	–

図 6-13は、L1データ・メモリのアーキテクチャを示します。

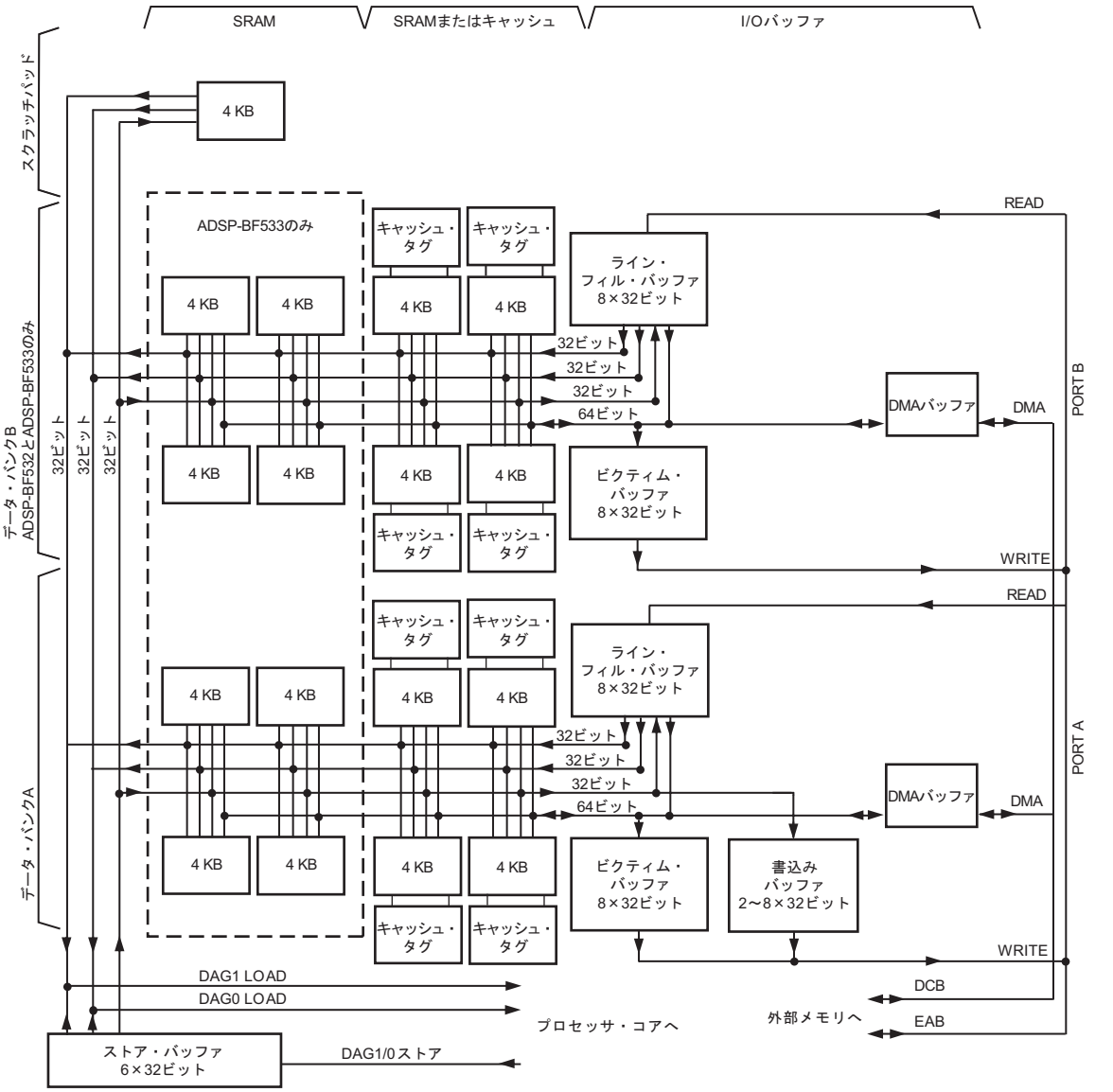


図 6-13. L1 データ・メモリのアーキテクチャ

■ L1 データ・キャッシュ

キャッシュ用語の定義については、6-80ページの「用語集」を参照してください。

データ・キャッシュがイネーブルにされる（DMEM_CONTROLレジスタのビットDMC[1:0]によって制御される）と、16Kバイトのデータ・バンクA、または16Kバイトのデータ・バンクAとデータ・バンクBの両方をキャッシュとして機能するように設定できます。ADSP-BF533では、上位16Kバイトが使用されます。ADSP-BF531では、データ・バンクAだけが使用できます。4ウェイ・セット・アソシエイティブな命令キャッシュとは異なり、データ・キャッシュは2ウェイ・セット・アソシエイティブです。2つのバンクが使用可能であり、キャッシュとしてイネーブルにされると、ウェイではなく、新たなセットが作成されます。データ・バンクAとデータ・バンクBの両方にキャッシュとして機能するメモリがある場合、DMEM_CONTROLレジスタのDCBSビットを使用すれば、キャッシュ・メモリのどのバンクですべてのアドレス空間のどのハーフを処理するかを制御できます。DCBSビットでは、アドレス・ビット14または23を選択して、キャッシュ・バンク間でのトラフィックを誘導します。これによって、アドレスが同じセットにエイリアスされることをいくらか制御できます。したがって、繰り返し使用されるセットの犠牲を回避することで、キャッシュ内に常駐しがちなアドレスに影響を与えることができます。

キャッシュへのアクセスは、そのアクセスが同じ4Kバイト・サブバンク、同じハーフ・バンク、同じバンクに対するものである場合を除いて衝突しません。キャッシュの場合、タグの維持に必要なオーバーヘッドのために、マルチポート型動作はSRAMほど明白ではありません。キャッシュ・アドレスが衝突すると、アクセスは、最初にDTESTレジスタ・アクセスに許可され、次にストア・バッファに許可され、最後にキャッシュ・フィル／ビクティム・トラフィックに許可されます。

3つのキャッシュ・モードを使用できます。

- 読出し専用で、キャッシュ・ライン割当て付きのライトスルー

- 読出し／書込みで、キャッシュ・ライン割当て付きのライトスルー
- 読出し／書込みで、キャッシュ・ラインを割り当てるライトバック

キャッシュ・モードは、DCPLBディスクリプタによって選択されます（6-51ページの「メモリ保護とプロパティ」を参照）。キャッシュ・モードはメモリ・ページごとに独立して選択できるため、これらのキャッシュ・モードの任意の組合わせを同時に使用できます。

キャッシュがイネーブルにされている（DMEM_CONTROLレジスタのビットDMC[1:0]によって制御されている）場合には、データCPLBもイネーブルにしてください（DMEM_CONTROLレジスタのENDCPLBビットによって制御される状態）。キャッシュされるのは、データCPLBによってキャッシュ可能と指定されたメモリ・ページだけです。データCPLBがディスエーブルにされた場合のデフォルト動作は、何もキャッシュされません。

MMR 空間がデータ CPLB によってキャッシュ可能と設定されたり、L1 SRAM として機能しているデータ・バンクがデータ CPLB によってキャッシュ可能と設定されたりした場合には、異常な動作につながることがあります。

▶ キャッシュ可能アドレス空間のマッピング例

次に、キャッシュ可能アドレス空間を2つのデータ・バンクにマッピングする例を示します。

2つのバンクがADSP-BF533またはADSP-BF532でキャッシュとして設定されているとき、これらのバンクはBlackfinプロセッサのアドレス空間に独立してマッピングできる、2つの独立した16Kバイトの2ウェイ・セット・アソシエイティブ・キャッシュとして動作します。

2つのデータ・バンクがキャッシュとして設定されている場合には、DMEM_CONTROLレジスタのDCBSビットは、アドレス・ビットA[14]またはA[23]をキャッシュ・セレクタとして指定します。アドレス・ビットA[14]またはA[23]は、データ・バンクAによって実装されたキャッシュ、またはデータ・バンクBによって実装されたキャッシュを選択します。

- DCBS = 0の場合、A[14]はアドレス・インデックスの一部であり、A[14] = 0であるすべてのアドレスではデータ・バンクBを使用します。A[14] = 1であるすべてのアドレスでは、データ・バンクAを使用します。

  この場合、A[23]は、タグ付きでキャッシュに格納されたアドレス内の単なる別ビットとして扱われ、キャッシュによるヒット／ミス処理で比較されます。

- DCBS = 1の場合、A[23]はアドレス・インデックスの一部であり、A[23] = 0であるすべてのアドレスではデータ・バンクBを使用します。A[23] = 1であるすべてのアドレスでは、データ・バンクAを使用します。

  この場合、A[14]はタグ付きでキャッシュに格納されたアドレス内の単なる別ビットとして扱われ、キャッシュによるヒット／ミス処理で比較されます。

DCBS = 0またはDCBS = 1を選択した結果は、次のとおりです。

- DCBS = 0の場合、A[14]はデータ・バンクBの代わりにデータ・バンクAを選択します。

交互に繰り返す16Kバイト・ページのメモリは、2つのデータ・バンクによって実装された2つの16Kバイト・キャッシュにそれぞれマッピングされます。したがって、次のことがいえます。

最初の 16K バイトのメモリにあるデータは、データ・バンク B にだけ格納できます。

次のアドレス範囲（16 ～ 32K バイト）－ 1 にあるデータは、データ・バンク A にだけ格納できます。

次の範囲（32 ～ 48K バイト）－ 1 にあるデータは、データ・バンク B に格納されます。

交互マッピングが続きます。

その結果、このキャッシュは、あたかも単一で連続した2ウェイ・セット・アソシエイティブな32Kバイト・キャッシュであるかのように動作します。各ウェイは16Kバイト長であり、同じ14ビットの先頭アドレスを持つすべてのデータ・エレメントは、2つまでのエレメントを（各ウェイに1つ）格納できる重複しないセットにインデックス付けされます。

- DCBS = 1の場合、A[23]はデータ・バンクBの代わりにデータ・バンクAを選択します。

  DCBS = 1では、システムは2つの独立したキャッシュ（それぞれが2ウェイ・セット・アソシエイティブな16Kバイト・キャッシュ）のように機能します。各バンクは、8Mバイト・ブロックのメモリの交互セットとして機能します。

  たとえば、データ・バンクBでは、最初の8Mバイトのメモリ・アドレス範囲に対するすべてのデータ・アクセスをキャッシュします。つまり、どの8Mバイト範囲も（16Kバイトの繰返しではなく）

2つのライン・エントリを得ようと競います。同様に、データ・バンクAでは、8Mバイトより上で16Mバイトより下にあるデータをキャッシュします。

たとえば、アプリケーションが最初の 8M バイトのメモリに完全に収まる 1M バイト長のデータ・セットから動作している場合には、そのアプリケーションは実質的にキャッシュの半分、つまりデータ・バンク B（2 ウェイ・セット・アソシエイティブな 16K バイト・キャッシュ）によって処理されます。この場合、アプリケーションはデータ・バンク A からは何のメリットも得られません。

多くのアプリケーションでは、`DCBS = 0` での動作が最適です。

しかし、アプリケーションが少なくとも8Mバイト離れた2つのメモリ空間にある2つのデータ・セットから動作している場合には、キャッシュとデータのマッピング関係をより綿密に制御できます。たとえば、プログラムが一連のデュアルMAC演算を実行しており、2つのDAGがどのサイクルでもデータにアクセスしている場合には、DAG0のデータ・セットを1つのブロックのメモリに置き、DAG1のデータ・セットをもう一方のブロックのメモリに置くことで、システムは以下のことを保証できます。

- DAG0は、そのすべてのアクセスでデータ・バンクAからそのデータを取得します。
- DAG1は、データ・バンクBからそのデータを取得します。

この配置では、コアはキャッシュ・ライン転送に両方のデータ・バスを使用し、キャッシュとコアの間で最大のデータ帯域幅が得られます。

図 6-14は、DCBS = 1の場合のマッピング例を示します。

DCBS選択は動的に変更できます。しかし、データ紛失を防止するには、最初にキャッシュ全体をフラッシュして無効化してください。

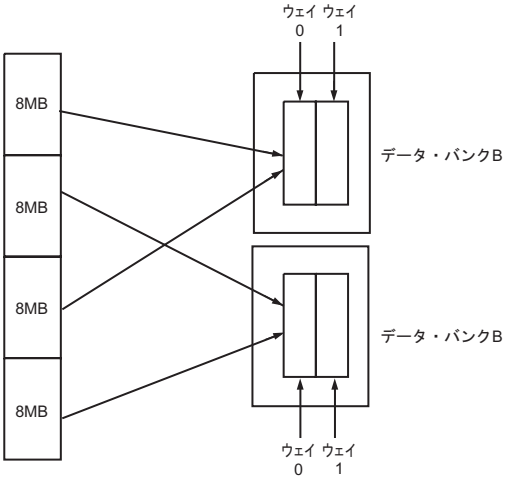


図 6-14. DCBS = 1 の場合のデータ・キャッシュ・マッピング

▶ データ・キャッシュ・アクセス

キャッシュ・コントローラは、DAGからのアドレスをタグ・ビットに基づいてテストします。論理アドレスがL1キャッシュに存在する場合にはキャッシュ・ヒットが発生し、データはL1でアクセスされます。論理アドレスが存在しない場合にはキャッシュ・ミスが発生し、メモリ・トランザクションはシステム・インターフェースを介して次のレベルのメモリに渡されます。外部メモリから戻ってくるデータに割り当てられるキャッシュ・タグとデータ空間は、キャッシュ・コントローラの置換ポリシーとライン・インデックスによって決まります。

データ・キャッシュ・ラインの状態は、無効、排他（有効でクリーン）、更新（有効でダーティ）のいずれかです。割り当てられたラインを有効なデータがすでに占有し、キャッシュがライトバック・ストレージ用に設定されている場合には、コントローラはキャッシュ・ラインの状態をチェックし、それに応じて処置します。

- ラインの状態が排他（クリーン）である場合には、新しいタグとデータは古いラインに上書きされます。
- ラインの状態が更新（ダーティ）である場合には、キャッシュにはデータの有効なコピーだけが含まれています。

  ラインがダーティである場合には、キャッシュの現在の内容が外部メモリにコピーバックされてから、キャッシュに新しいデータが書き込まれます。

プロセッサは、ビクティム・バッファとライン・フィル・バッファを提供します。キャッシュ・ロード・ミスによって、置換の必要なビクティム・キャッシュ・ラインが生成された場合には、これらのバッファが使用されます。ライン・フィル動作は、外部メモリにまで達します。データ・キャッシュは、システムへのライン・フィル要求をクリティカル（または要求された）ワード・ファーストとして実行し、そのデータをキャッシュ・ラインの更新時に待機中のDAGに転送します。つまり、このキャッシュは、クリティカル・ワード転送を行います。

データ・キャッシュは、ヒットアンダー・ストア・ミスとヒットアンダー・プリフェッチ・ミスに対応しています。つまり、書込みミスが発生したり、キャッシュをミスする（キャッシュ可能領域への）PREFETCH命令が実行されると、命令パイプラインは、少なくとも4サイクルのストールを引き起こします。さらに、ライン・フィルが完了する間に、それ以降のロード／ストア命令がL1キャッシュでヒットすることがあります。

十分な優先順位（現在のコンテキストから見て）を持つ割込みは、ストールされたロード命令をキャンセルします。したがって、ロード動作がL1データ・メモリ・キャッシュをミスし、システム・インターフェースに高遅延のライン・フィル動作を生成した場合には、コアに割込みをかけて、別のコンテキストの処理を開始させることが可能です。キャッシュ・ラインをフィルするためのシステム・アクセスはキャンセルされず、データ・キャッシュが新しいデータで更新されてから、それぞれのデータ・バンクに対するその後のキャッシュ・ミス動作が処理されます。詳細については、4-42ページの「例外」を参照してください。

▶ キャッシュ書込み方式

キャッシュ書込みメモリ動作は、ライトスルー方式またはライトバック方式を使用して実現できます。

- ストア動作ごとに、ライトスルー・キャッシュは、キャッシュへの書込み直後に外部メモリへの書込みを開始します。

  キャッシュ・ラインがソフトウェアによって明示的にフラッシュされたか置換された場合には、キャッシュ・ラインの内容は、外部メモリに書き戻されずに無効化されます。

- ライトバック・キャッシュは、ラインがそのラインを必要とするロード動作によって置換されるまでは、外部メモリに書き込みません。

L1データ・メモリは、各データ・バンク上にフル・キャッシュ・ライン幅のコピーバック・バッファを採用しています。さらに、L1データ・メモリ内の2エントリ書込みバッファでは、キャッシュ禁止またはストアスルー保護付きのすべてのストアを受け付けます。SSYNC命令は、書込みバッファをフラッシュします。

▶ IPRIO レジスタと書込みバッファの深さ

割込み優先順位レジスタ（IPRIO）を使用すれば、ポートAでの書込みバッファのサイズを制御できます（6-34ページの「L1データ・メモリのアーキテクチャ」を参照してください）。

IPRIO[3:0]ビットは、低優先順位の割込みのウォーターマークを反映するようにプログラムできます。割込みが発生して、プロセッサが低優先順位の割込みサービス・ルーチンから高優先順位の割込みサービス・ルーチンに制御を移すと、書込みバッファのサイズは2つの32ビット・ワード深さから8つの32ビット・ワード深さまで増大します。これによって、書込みバッファが低優先順位の割込みルーチンですでにフィルされていた場合に、割込みサービス・ルーチンは初期ストールなしで書込みを実行およびポストできます。これが最も役立つのは、ポストされた書込みが低速な外部メモリ・デバイスへのものである場合です。高優先順位の割込みサービス・ルーチンから、低優先順位の割込みサービス・ルーチンまたはユー

ザ・モードに戻ると、コアは書込みバッファが必要な書込みを完了して深さ2の状態に戻るまでストールします。デフォルトでは、書込みバッファは固定された深さ2のFIFOです。

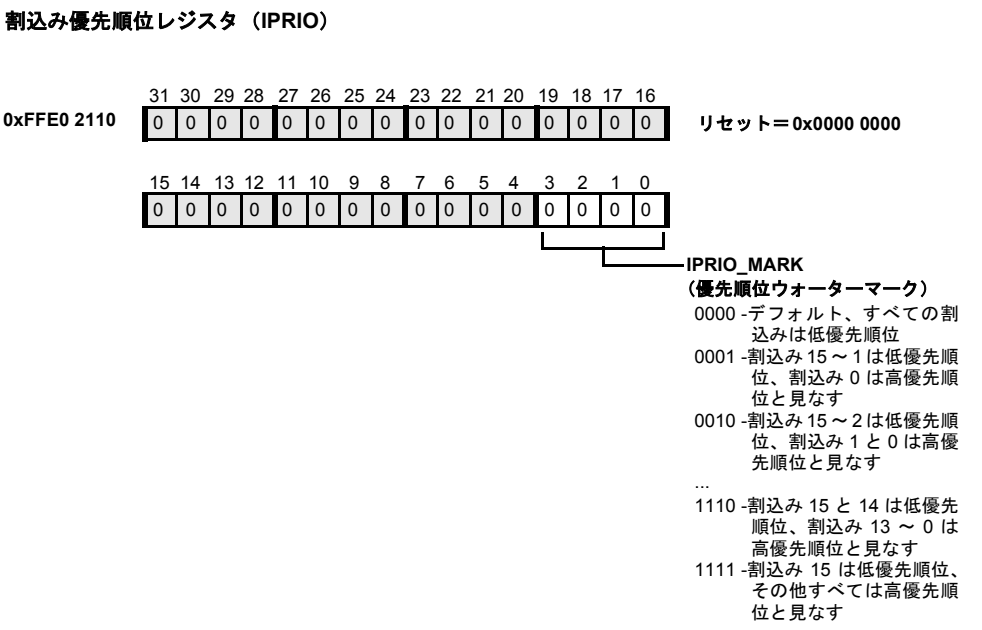


図 6-15. 割込み優先順位レジスタ

▶ データ・キャッシュ制御命令

このプロセッサは、ユーザ・モードとスーパーバイザ・モードでアクセスできる3つのデータ・キャッシュ制御命令を定めています。つまり、PREFETCH、FLUSH、FLUSHINVの各命令です。

- PREFETCH（データ・キャッシュ・プリフェッチ）では、L1 キャッシュ内にラインの割当てを試みます。プリフェッチがキャッシュでヒットした場合、または例外を生成するか、キャッシュ禁止領域をアドレス指定した場合には、PREFETCHはNOPのように機能します。

- FLUSH（データ・キャッシュ・フラッシュ）では、データ・キャッシュは指定のキャッシュ・ラインと外部メモリを同期させます。キャッシュされたデータ・ラインがダーティである場合には、この命令はラインを書き出して、データ・キャッシュでそのラインをクリーンとマークします。指定のデータ・キャッシュ・ラインがすでにクリーンであるか存在しない場合には、FLUSHはNOPのように機能します。

- FLUSHINV（データ・キャッシュ・ラインのフラッシュと無効化）では、データ・キャッシュはFLUSH命令と同じ機能を実行してから、キャッシュ内の指定のラインを無効化します。ラインがキャッシュ内にあってダーティである場合には、キャッシュ・ラインは外部メモリに書き出されます。その後、キャッシュ・ライン内のバリッド・ビットがクリアされます。ラインがキャッシュにない場合には、FLUSHINVはNOPのように機能します。

ソフトウェアがシステム・ハードウェアとの同期を必要とする場合には、FLUSH動作が完了したことを確認するために、FLUSH命令の後にSSYNC命令を置きます。これまでのストアがすべてのキューを通じてプッシュされたことを保証するために順序付けが望まれる場合には、FLUSHの前にSSYNC命令を置きます。

▶ データ・キャッシュの無効化

前の節で説明したFLUSHINV命令に加えて、フラッシングが必要ない場合には、2つの方法でデータ・キャッシュを無効化できます。最初の方法では、各キャッシュ・ラインのインバリッド・ビットを無効状態に設定することでバリッド・ビットを直接に無効化します。この方法を実現するため、新たなMMR（DTEST_COMMANDとDTEST_DATA[1:0]）によって、すべてのキャッシュ・エントリを直接に読出し／書込みできるようにします。この方法については、次の節で説明します。

データ・キャッシュをまるごと無効化するには、2番目の方法を使用します。DMEM_CONTROLレジスタ（6-29ページの図6-12「L1データ・メモリ・コントロール・レジスタ」を参照）のDMC[1:0]ビットをクリアすると、データ・キャッシュ内のすべてのバリッド・ビットは無効状態に設定されます。DMEM_CONTROL レジスタへの2回目の書込みによって、DMC[1:0]ビットを以前の状態に設定してから、データ・メモリを以前のキャッシュ／SRAM設定に戻します。SSYNC命令を実行してからキャッシュを無効化します。また、これらの動作が終了するたびに、CSYNC命令を挿入してください。

データ・テスト・レジスタ

L1命令メモリと同様に、L1データ・メモリにも新たなMMRが含まれており、すべてのキャッシュ・エントリを直接に読出し／書込みできます。これらのレジスタは、データ・キャッシュをテスト、初期化、デバッグするためのメカニズムを提供します。

データ・テスト・コマンド・レジスタ（DTEST_COMMAND）への書込みの際には、L1キャッシュのデータ・アレイまたはタグ・アレイがアクセスされ、データはデータ・テスト・データ・レジスタ（DTEST_DATA[1:0]）を通じて転送されます。DTEST_DATA[1:0]レジスタには、書き込まれる64ビット・データ、または64ビット・データ読出し用のデスティネーショ

ンが含まれています。下位32ビットはDTEST_DATA[0]レジスタに格納され、上位32ビットはDTEST_DATA[1]レジスタに格納されます。タグ・アレイのアクセス時には、DTEST_DATA[0]レジスタが使用されます。

DTEST_COMMANDの書込み後には、CSYNC命令が必要です。

DTESTレジスタについては、以下の図で説明します。

- 6-48ページの図6-16「データ・テスト・コマンド・レジスタ」
- 6-49ページの図6-17「データ・テスト・データ1レジスタ」
- 6-50ページの図6-18「データ・テスト・データ0レジスタ」

これらのレジスタへのアクセスは、スーパーバイザ・モードまたはエミュレーション・モードでのみ可能です。DTESTレジスタへの書込みに際しては、最初にDTEST_DATAレジスタに書き込んでから、次にDTEST_COMMANDレジスタに書き込みます。

■ DTEST_COMMAND レジスタ

データ・テスト・コマンド・レジスタ（DTEST_COMMAND）への書込みの際には、L1キャッシュのデータ・アレイまたはタグ・アレイがアクセスされ、データはデータ・テスト・データ・レジスタ（DTEST DATA[1:0]）を通じて転送されます。

データ／命令アクセス・ビットによって、DTEST_COMMAND MMRによるL1命令SRAMへの直接アクセスが可能です。

データ・テスト・コマンド・レジスタ（DTEST_COMMAND）

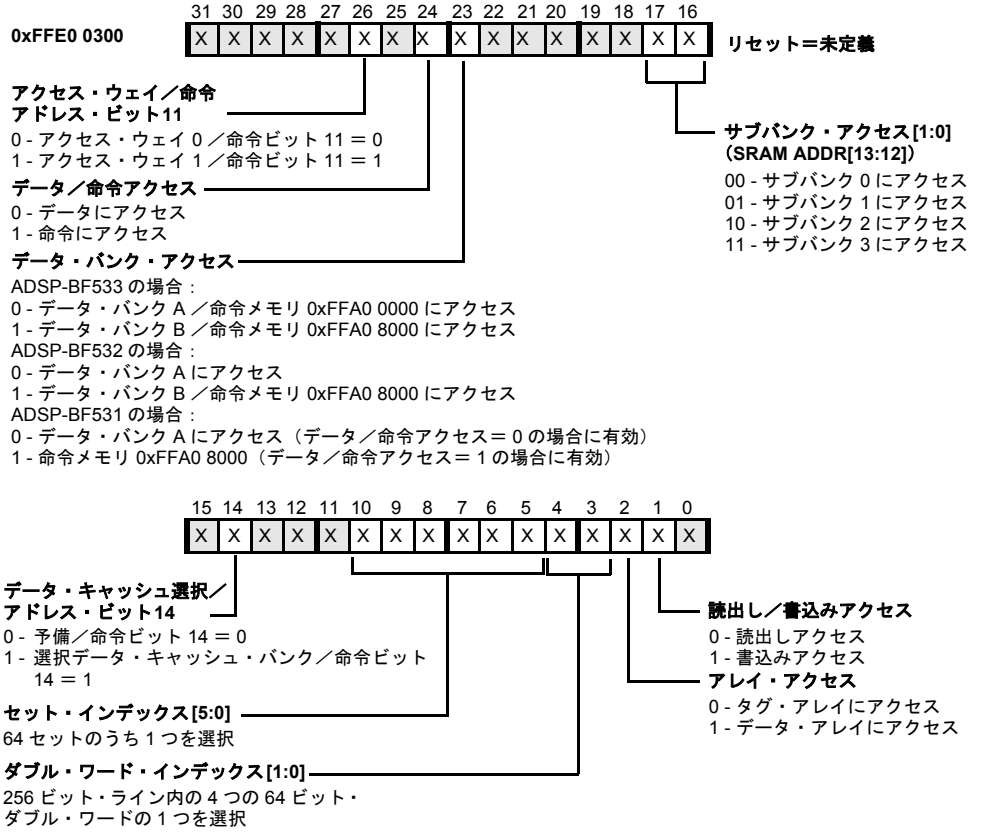


図 6-16. データ・テスト・コマンド・レジスタ

■ DTEST_DATA1 レジスタ

データ・テスト・データ・レジスタ（DTEST_DATA[1:0]）には、書き込まれる64ビット・データ、または64ビット・データ読出しのデスティネーションが含まれています。データ・テスト・データ1レジスタ（DTEST_DATA1）には上位32ビットを格納します。

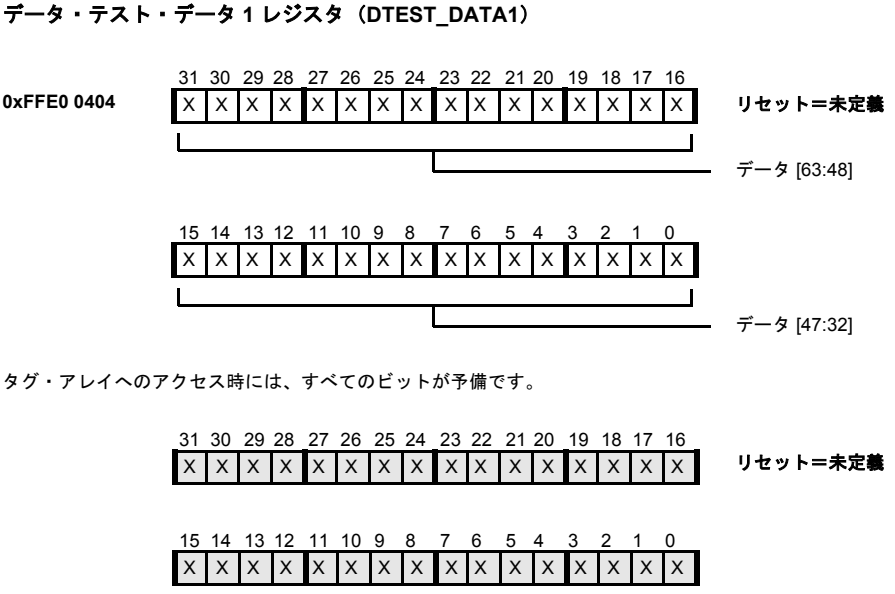


図 6-17. データ・テスト・データ 1 レジスタ

■ DTEST_DATA0 レジスタ

データ・テスト・データ0レジスタ（DTEST_DATA0）には、書き込まれる64ビット・データの下位32ビット、または64ビット・データ読出しのデスティネーションの下位32ビットが含まれています。DTEST_DATA0レジスタは、タグ・アレイへのアクセスにも使用され、キャッシュ・ラインの状態を示すバリッド・ビットとダーティ・ビットを含みます。

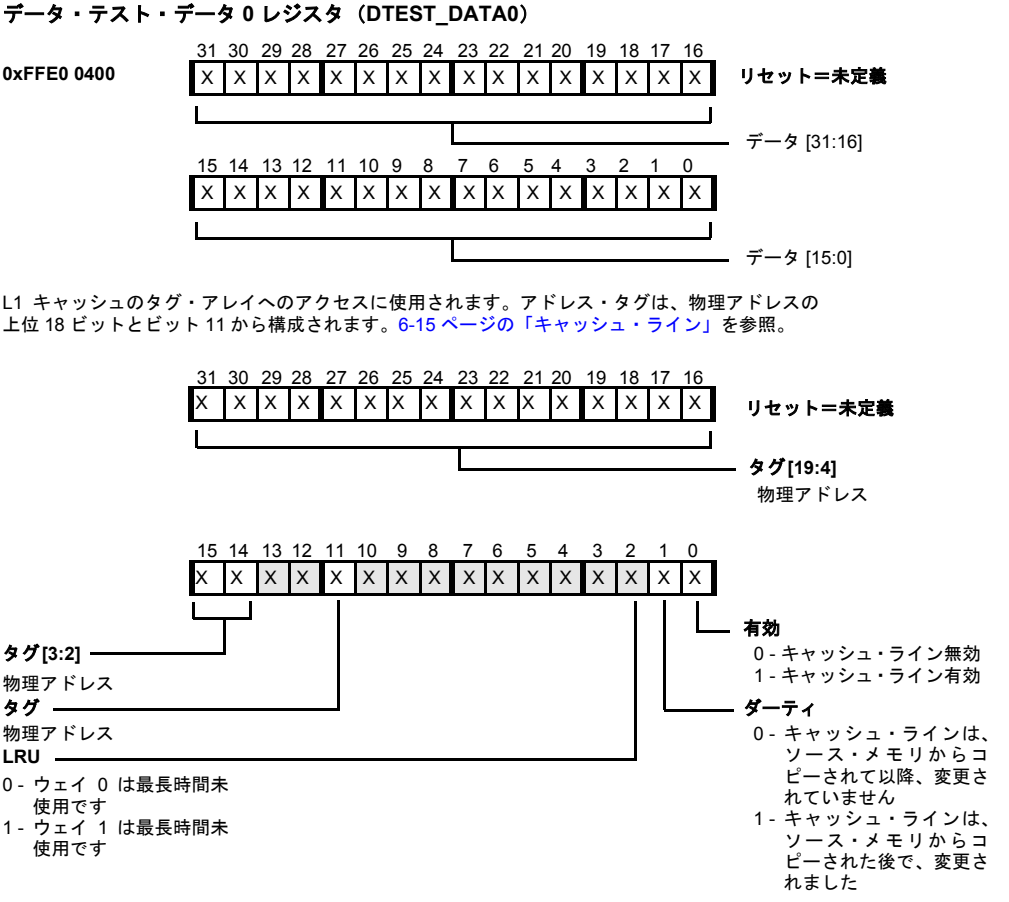


図 6-18. データ・テスト・データ 0 レジスタ

外部メモリ

外部メモリ空間を6-3ページの図6-1に示します。メモリ領域の1つはSDRAMサポート専用です。SDRAMバンクのサイズは16～128Mバイトの範囲でプログラマブルです。バンクの開始アドレスは0x0000 0000です。

次の4つのバンクはそれぞれ1Mバイトを含み、非同期メモリのサポート専用です。非同期メモリ・バンクの開始アドレスは0x2000 0000です。

メモリ保護とプロパティ

この節では、メモリ・マネジメント・ユニット（MMU）、メモリ・ページ、CPLB管理、MMU管理、CPLBレジスタについて説明します。

■ メモリ・マネジメント・ユニット

Blackfinプロセッサには、ページ・ベースのメモリ・マネジメント・ユニット（MMU）が内蔵されています。このメカニズムでは、メモリ範囲のキャッシュ可能性を制御できるだけでなく、ページ・レベルでの保護特性も管理できます。MMUでは、アクセス権とキャッシュ動作を完全に制御して、タスク間でメモリ・リソースとI/Oリソースを極めて柔軟に割り当てることができます。

MMUは、16エントリの2つの連想メモリ（CAM）ブロックとして実装されます。各エントリは、キャッシュ可能性保護索引バッファ（CPLB）ディスクリプタと呼ばれます。イネーブルにされた場合、MMU内のすべての有効エントリがフェッチ、ロード、またはストア動作のたびに検査され、要求されたアドレスとCPLBエントリによって記述されたページとの間に一致があるかどうかを判定されます。一致した場合には、ディスクリプタ内に含まれるキャッシュ可能性特性と保護特性がメモリ・トランザクション用に使用されます（命令の実行サイクルは増えません）。

L1メモリは命令メモリとデータ・メモリに分割されるため、CPLBエントリも命令CPLBとデータCPLBの間で分割されます。命令フェッチ要求に対しては16個のCPLBエントリが使用され、これらは*ICPLB*と呼ばれます。データ・トランザクションにはさらに16個のCPLBエントリが使用され、これらは*DCPLB*と呼ばれます。ICPLBとDCPLBをイネーブルにするには、それぞれL1命令メモリ制御（IMEM_CONTROL）レジスタとL1データ・メモリ制御（DMEM_CONTROL）レジスタの適切なビットをセットします。これらのレジスタを、それぞれ6-11ページの図6-5と6-29ページの図6-12に示します。

各CPLBエントリは、32ビット値のペアで構成されます。命令フェッチの場合：

- ICPLB_ADDR[n] では、CPLB ディスクリプタによって記述されたページの開始アドレスを定義します。
- ICPLB_DATA[n] では、CPLB ディスクリプタによって記述されたページのプロパティを定義します。

データ操作の場合：

- DCPLB_ADDR[m] では、CPLB ディスクリプタによって記述されたページの開始アドレスを定義します。
- DCPLB_DATA[m] では、CPLB ディスクリプタによって記述されたページのプロパティを定義します。

スクラッチパッド・データ・メモリへのデータ・アクセスと、システムおよびコアMMR空間へのデータ・アクセスには、2つのデフォルトCPLBディスクリプタがあります。これらのデフォルト・ディスクリプタでは上記の空間をキャッシュ不可能と定義するため、これらのメモリ領域に対して新たなCPLBを設定する必要はありません。

この空間に有効な CPLB が設定されている場合には、デフォルトCPLBは無視されます。

■ メモリ・ページ

プロセッサの4Gバイトのアドレス空間は、より小さな範囲のメモリ、つまりメモリ・ページと呼ばれるI/Oに分割できます。ページ内のすべてのアドレスは、そのページに定義された特性を共有します。このアーキテクチャでは、4つの異なるページ・サイズに対応します。

- 1Kバイト
- 4Kバイト
- 1Mバイト
- 4Mバイト

さまざまなページ・サイズによって、さまざまな種類のメモリとI/Oに合わせて特性のマッピングを整合させる柔軟なメカニズムが提供されます。

▶ メモリ・ページ特性

各ページは、アドレス・ディスクリプタ・ワード`xCPLB_ADDR[n]`とプロパティ・ディスクリプタ・ワード`xCPLB_DATA[n]`から構成される2ワード・ディスクリプタによって定義されます。アドレス・ディスクリプタ・ワードは、メモリ内のページのベース・アドレスを提供します。ページは、ページ・サイズの整数倍であるページ境界に合わせる必要があります。たとえば、4Mバイトのページは、4Mバイトで割り切れるアドレスから始める必要があります。一方、1Kバイトのページは、どの1Kバイト境界でも始めることができます。ディスクリプタ内の2番目のワードは、ページのその他のプロパティや特性を指定します。これらのプロパティには、以下のものが含まれます。

- ページ・サイズ

  1Kバイト、4Kバイト、1Mバイト、4Mバイト

- キャッシュ可能／キャッシュ不可能

  このページへのアクセスは、L1キャッシュを使用するか、キャッシュをバイパスします。

- キャッシュ可能な場合：ライトスルー／ライトバック

  データ書込みは、メモリに直接伝搬するか、キャッシュ・ラインが再割当てされるまで延期されます。ライトスルーの場合、読出し専用、または読出し／書込みで割り当てます。

- ダーティ／変更済み

  メモリ内のこのページのデータは、CPLBが最後にロードされて以降に変更されました。

- スーパーバイザの書込みアクセス権

  – スーパーバイザ・モードでは、このページへの書込みをイネーブル／ディスエーブルにします。
  – データ・ページのみ。

- ユーザの書込みアクセス権

  – ユーザ・モードでは、このページへの書込みをイネーブル／ディスエーブルにします。
  – データ・ページのみ。

- ユーザの読出しアクセス権

  ユーザ・モードでは、このページからの読出しをイネーブル／ディスエーブルにします。

- バリッド

  有効なCPLBデータであるかどうかを判定するには、このビットをチェックします。

- ロック

  このエントリはMMR内に保持します。CPLB置換ポリシーに参加させないでください。

■ ページ・ディスクリプタ・テーブル

CPLBが命令アクセス、データ・アクセス、またはその両方に対してイネーブルにされているとき、メモリ・アクセスでキャッシュを利用するには、MMRペアで有効なCPLBエントリを使用できることが必要です。CPLBエントリ用のMMRの保管場所は、命令フェッチの場合は16個のディスクリプタ、データのロード／ストア動作の場合も16個のディスクリプタに限定されています。

小さなメモリ・モデルや簡単なメモリ・モデルでは、これらの32個のエントリに収まる一連のCPLBディスクリプタを定義して、アドレス可能な全空間をカバーし、置換を不要にすることも可能です。この種の定義は、**静的**メモリ・マネジメント・モデルと呼ばれます。

しかし、一般のオペレーティング環境では、アドレス可能なメモリ空間とI/O空間をカバーするために定義すべきCPLBディスクリプタの数は、使用可能なオンチップCPLB MMRに収まりません。このような場合、ページ・ディスクリプタ・テーブルと呼ばれる、メモリベースのデータ構造が使用されます。その中には、潜在的に要求されるすべてのCPLBディスクリプタを格納できます。ページ・ディスクリプタ・テーブルの具体的なフォーマットは、Blackfinプロセッサ・アーキテクチャの一部としては定義されていません。メモリ・マネジメント・モデルはオペレーティング・システムによって異なります。したがって、OSごとにその条件に合致す

るページ・ディスクリプタ・テーブル構造を実装できます。このため、提供される保護のレベルとメモリ管理サポート・ルーチンの性能特性との間で調整を行うことが可能です。

■ CPLB 管理

Blackfinプロセッサが発行したメモリ操作に対して、MMRページ内に有効なCPLB（キャッシュ可能性保護索引バッファ）ディスクリプタが存在しない場合には、例外が発生して、プロセッサはスーパーバイザ・モードになり、MMU例外ハンドラに制御が移ります（詳細については、4-42ページの「例外」を参照）。一般に、ハンドラはCPLB置換ポリシーを実行するオペレーティング・システム（OS）カーネルの一部です。

CPLBがイネーブルにされる前に、ページ・ディスクリプタ・テーブルとMMU例外ハンドラの両方に対して有効なCPLBディスクリプタが所定の位置にある必要があります。これらのCPLBディスクリプタについては、ソフトウェアで誤って置換されないように一般に**ロック・ビット**がセットされています。

オンチップCPLBレジスタ・ペアの1つをロードすべき正しいCPLBディスクリプタ・データを見つけるために、ハンドラは障害を起こしたアドレスを使用して、ページ・ディスクリプタ・テーブル構造にインデックス付けします。すべてのオンチップ・レジスタが有効なCPLBエントリを含んでいる場合には、ハンドラは置換すべきディスクリプタの1つを選択し、新しいディスクリプタ情報がロードされます。新しいディスクリプタ・データをCPLBにロードする前に、以下のビットを使用して、対応するグループの16個のCPLBをディスエーブルにする必要があります。

- データ・ディスクリプタの場合は、DMEM_CONTROLレジスタのイネーブルDCPLB（ENDCPLB）ビット
- 命令ディスクリプタの場合は、IMEM_CONTROLレジスタのイネーブルICPLB（ENICPLB）ビット

使用するアルゴリズムとCPLB置換ポリシーは、システムMMU例外ハンドラの責任になります。このポリシーは、オペレーティング・システムの特性によって規定され、通常は修正LRU（最長時間未使用）ポリシー、ラウンド・ロビン・スケジューリング方式、または擬似ランダム置換を実装します。

新しいCPLBディスクリプタがロードされた後、例外ハンドラが復帰し、障害を起こしたメモリ操作が再開されます。この操作では、今度は要求されたアドレスに対して有効なCPLBディスクリプタが見つかり、正常に続行されるはずです。

1つの命令が、1回の命令フェッチに加えて、1つまたは2つのデータ・アクセスも引き起こすことがあります。このような複数のメモリ操作で、有効なCPLBディスクリプタがMMRペア内に存在しないデータを参照する可能性があります。この場合、例外は次の順番に優先順位付けおよび処理されます。

- 命令のページ・ミス
- DAG0でのページ・ミス
- DAG1でのページ・ミス

■ MMU アプリケーション

メモリ・マネジメントは、Blackfinプロセッサ・アーキテクチャでのオプション機能です。その使用は、特定アプリケーションのシステム条件に基づいています。リセットした時点で、すべてのCPLBがディスエーブルにされ、メモリ・マネジメント・ユニット（MMU）は使用されません。

すべてのL1メモリがSRAMとして設定された場合には、タスク間またはユーザ・モードとスーパーバイザ・モードの間でのメモリ空間の保護に対するアプリケーションのニーズに応じて、データと命令のMMU機能はオプションになります。タスク間でメモリを保護するため、オペレーティング・システムは各タスクで使用できる命令やデータのメモリ・ページの個

別のテーブルを維持して、関連するタスクの実行時にだけそれらのページをアクセス可能にできます。タスク切替えが発生すると、オペレーティング・システムは、新しいタスクでは使用すべきでないCPLBディスクリプタのオンチップでの無効化を保証できます。また、新しいタスクに適切なディスクリプタをプリロードすることも可能です。

多くのオペレーティング・システムでは、アプリケーション・プログラムはユーザ・モードで実行され、オペレーティング・システムとそのサービスはスーパーバイザ・モードで実行されます。オペレーティング・システムによって使用されるコードとデータ構造については、実行中のユーザ・モード・アプリケーションによる不注意な変更から保護すべきです。この保護を実現するには、スーパーバイザ・モードだけでの書込みアクセスを許容する、保護されたメモリ範囲用にCPLBディスクリプタを定義します。保護されたメモリ領域への書込みがユーザ・モードで試みられた場合には、メモリが変更される前に例外が生成されます。オプションでは、ユーザ・モード・アプリケーションには、アプリケーションにとって有益なデータ構造に対する読出しアクセスが認められることがあります。変更が予定されていないコードを含むメモリ・ページについては、スーパーバイザ・モードの機能でさえ書込みを阻止されることがあります。CPLBエントリはスーパーバイザ・モードでだけ書込みできるMMRであるため、ユーザ・プログラムは、このようにして保護されたリソースにはアクセスできません。

L1命令メモリまたはL1データ・メモリの一部または全部がキャッシュとして設定されている場合には、対応するCPLBをイネーブルにすることが必要です。命令がメモリ要求を生成し、キャッシュがイネーブルにされると、プロセッサは最初にICPLBを調べて、要求されたアドレスがキャッシュ可能なアドレス範囲に含まれるかどうかを判定します。MMRペア内に要求されたアドレスに対応する有効なICPLBエントリが存在しない場合には、メモリがキャッシュ可能かどうかを判定する有効なICPLBディスクリプタを取得するためにMMU例外が生成されます。その結果、L1命令メモリがキャッシュとしてイネーブルにされている場合には、命令を格納するメモリ領域に有効なICPLBディスクリプタが定義されている必要

があります。これらのディスクリプタは、常時MMR内に存在するか、MMU例外ハンドラによって管理されるメモリベースのページ・ディスクリプタ・テーブルに存在することが必要です。同様に、L1データ・バンクの一方または両方がキャッシュとして設定されている場合には、潜在的なすべてのデータ・メモリ範囲は、DCPLBディスクリプタによってサポートされる必要があります。

キャッシュがイネーブルにされる前に、MMU　とそれに対応するデータ構造を設定してイネーブルにする必要があります。

■ 保護されたメモリ領域の例

図 6-19では、命令CPLBとデータCPLB用の基本的なCPLB割当てのために開始点が提供されています。なお、いくつかのICPLBとDCPLBには、同じアドレス空間に対して共通のディスクリプタがあります。

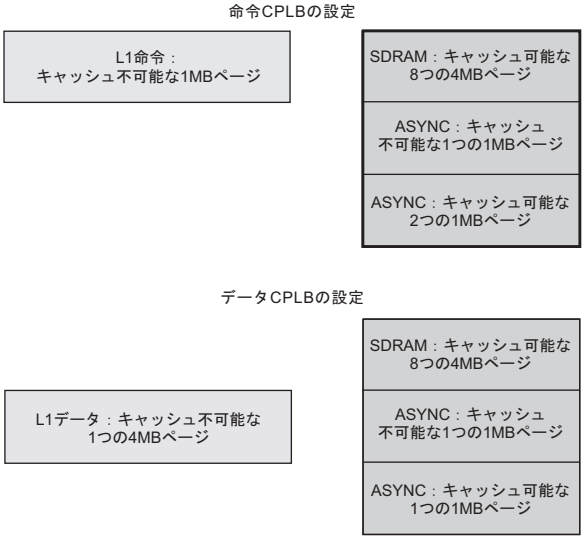


図 6-19. 保護されたメモリ領域の例

■ ICPLB_DATAx レジスタ

図 6-20は、ICPLBデータ・レジスタ（ICPLB_DATAx）を示します。

正常な動作と将来の互換性を保証するため、このレジスタ内のすべての予備ビットは、このレジスタが書き込まれるたびに0に設定する必要があります。

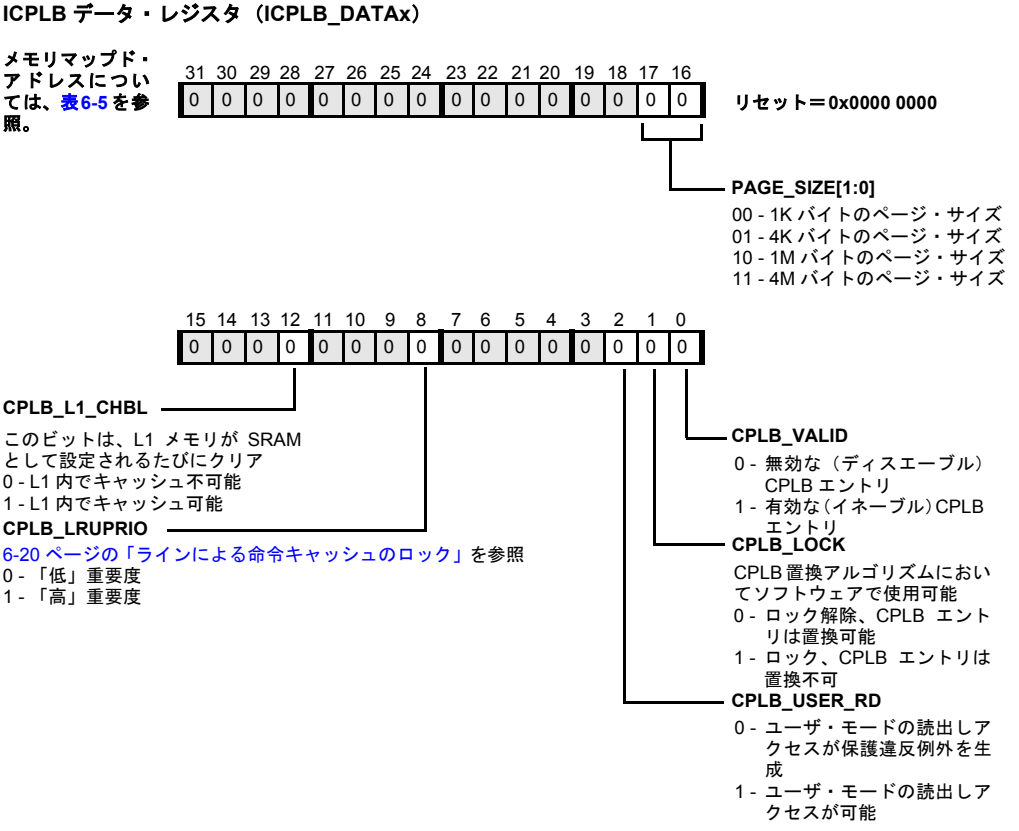


図 6-20. ICPLB データ・レジスタ

表 6-5.　ICPLB データ・レジスタのメモリマップド・アドレス

レジスタ名	メモリマップド・アドレス
ICPLB_DATA0	0xFFE0 1200
ICPLB_DATA1	0xFFE0 1204
ICPLB_DATA2	0xFFE0 1208
ICPLB_DATA3	0xFFE0 120C
ICPLB_DATA4	0xFFE0 1210
ICPLB_DATA5	0xFFE0 1214
ICPLB_DATA6	0xFFE0 1218
ICPLB_DATA7	0xFFE0 121C
ICPLB_DATA8	0xFFE0 1220
ICPLB_DATA9	0xFFE0 1224
ICPLB_DATA10	0xFFE0 1228
ICPLB_DATA11	0xFFE0 122C
ICPLB_DATA12	0xFFE0 1230
ICPLB_DATA13	0xFFE0 1234
ICPLB_DATA14	0xFFE0 1238
ICPLB_DATA15	0xFFE0 123C

■ DCPLB_DATAx レジスタ

図 6-21は、DCPLBデータ・レジスタ（DCPLB_DATAx）を示します。

正常な動作と将来の互換性を保証するため、このレジスタ内のすべての予備ビットは、このレジスタが書き込まれるたびに0に設定する必要があります。

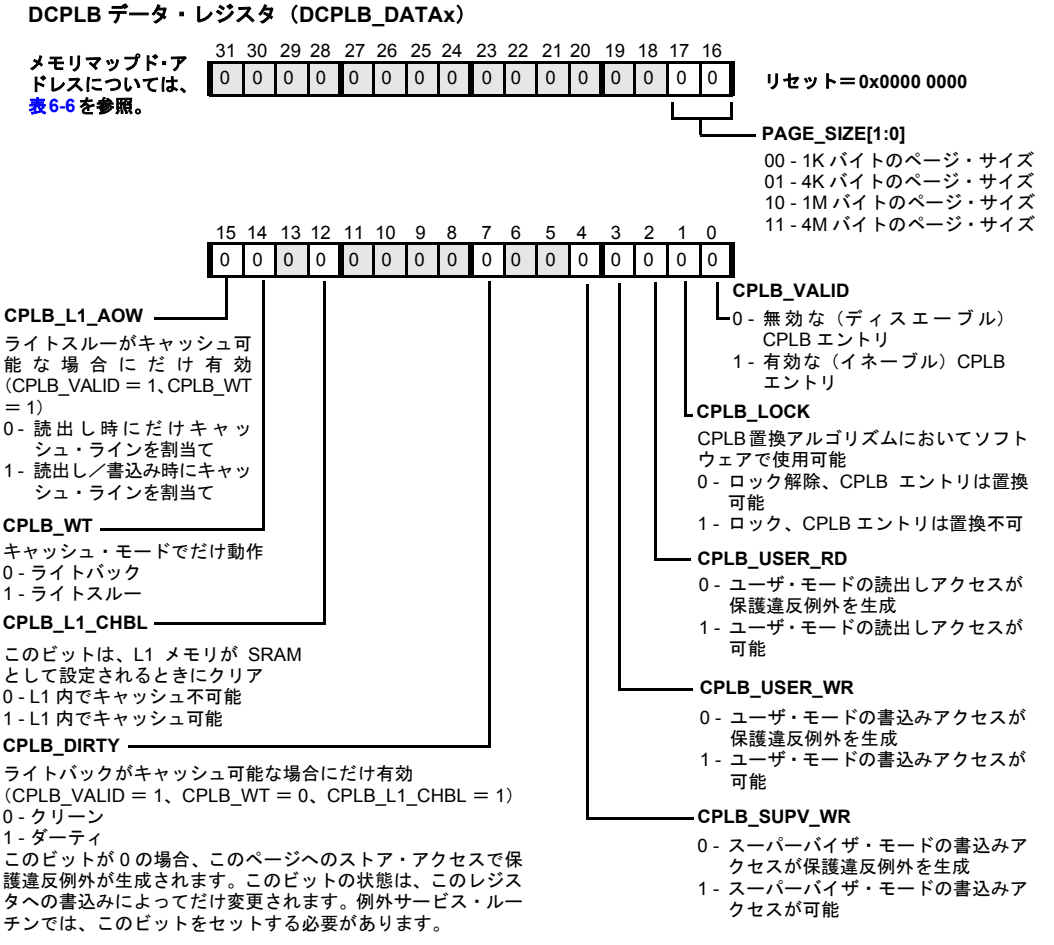


図 6-21. DCPLB データ・レジスタ

表 6-6.　DCPLB データ・レジスタのメモリマップド・アドレス

レジスタ名	メモリマップド・アドレス
DCPLB_DATA0	0xFFE0 0200
DCPLB_DATA1	0xFFE0 0204
DCPLB_DATA2	0xFFE0 0208
DCPLB_DATA3	0xFFE0 020C
DCPLB_DATA4	0xFFE0 0210
DCPLB_DATA5	0xFFE0 0214
DCPLB_DATA6	0xFFE0 0218
DCPLB_DATA7	0xFFE0 021C
DCPLB_DATA8	0xFFE0 0220
DCPLB_DATA9	0xFFE0 0224
DCPLB_DATA10	0xFFE0 0228
DCPLB_DATA11	0xFFE0 022C
DCPLB_DATA12	0xFFE0 0230
DCPLB_DATA13	0xFFE0 0234
DCPLB_DATA14	0xFFE0 0238
DCPLB_DATA15	0xFFE0 023C

■ DCPLB_ADDRx レジスタ

図 6-22は、DCPLBアドレス・レジスタ（DCPLB_ADDRx）を示します。

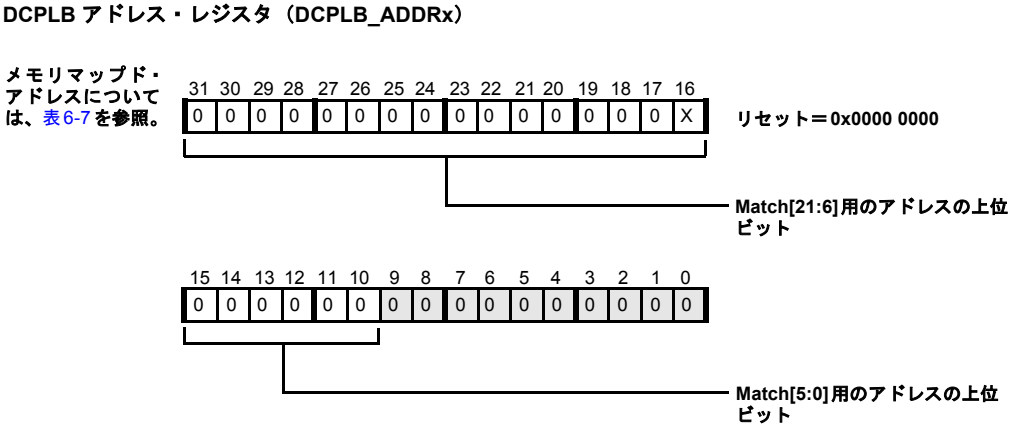


図 6-22. DCPLB アドレス・レジスタ

表 6-7. DCPLB アドレス・レジスタのメモリマップド・アドレス

レジスタ名	メモリマップド・アドレス
DCPLB_ADDR0	0xFFE0 0100
DCPLB_ADDR1	0xFFE0 0104
DCPLB_ADDR2	0xFFE0 0108
DCPLB_ADDR3	0xFFE0 010C
DCPLB_ADDR4	0xFFE0 0110
DCPLB_ADDR5	0xFFE0 0114
DCPLB_ADDR6	0xFFE0 0118
DCPLB_ADDR7	0xFFE0 011C
DCPLB_ADDR8	0xFFE0 0120
DCPLB_ADDR9	0xFFE0 0124
DCPLB_ADDR10	0xFFE0 0128

表 6-7.　DCPLB アドレス・レジスタのメモリマップド・アドレス（続き）

レジスタ名	メモリマップド・アドレス
DCPLB_ADDR11	0xFFE0 012C
DCPLB_ADDR12	0xFFE0 0130
DCPLB_ADDR13	0xFFE0 0134
DCPLB_ADDR14	0xFFE0 0138
DCPLB_ADDR15	0xFFE0 013C

■ ICPLB_ADDRx レジスタ

図 6-23は、ICPLBアドレス・レジスタ（ICPLB_ADDRx）を示します。

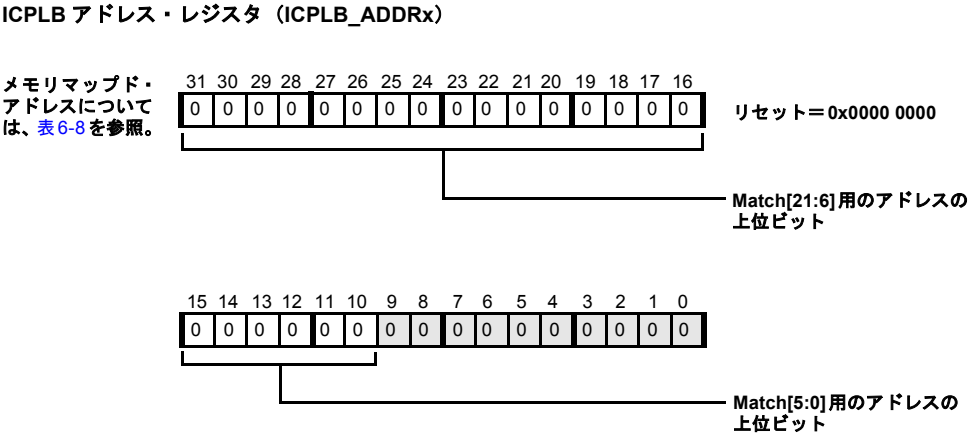


図 6-23. ICPLB アドレス・レジスタ

表 6-8.　ICPLB アドレス・レジスタのメモリマップド・アドレス

レジスタ名	メモリマップド・アドレス
ICPLB_ADDR0	0xFFE0 1100
ICPLB_ADDR1	0xFFE0 1104
ICPLB_ADDR2	0xFFE0 1108
ICPLB_ADDR3	0xFFE0 110C
ICPLB_ADDR4	0xFFE0 1110
ICPLB_ADDR5	0xFFE0 1114
ICPLB_ADDR6	0xFFE0 1118
ICPLB_ADDR7	0xFFE0 111C
ICPLB_ADDR8	0xFFE0 1120
ICPLB_ADDR9	0xFFE0 1124
ICPLB_ADDR10	0xFFE0 1128
ICPLB_ADDR11	0xFFE0 112C
ICPLB_ADDR12	0xFFE0 1130
ICPLB_ADDR13	0xFFE0 1134
ICPLB_ADDR14	0xFFE0 1138
ICPLB_ADDR15	0xFFE0 113C

■ DCPLB_STATUS レジスタと ICPLB_STATUS レジスタ

DCPLBステータス・レジスタ（DCPLB_STATUS）とICPLBステータス・レジスタ（ICPLB_STATUS）内のビットでは、CPLB関連の例外をトリガしたCPLBエントリを識別します。例外サービス・ルーチンでは、CPLBエントリを検査することで、障害の原因を推測できます。

DCPLB_STATUS レジスタと ICPLB_STATUS レジスタは、障害を起こした例外サービス・ルーチンでだけ有効です。

DCPLBステータス・レジスタ（DCPLB_STATUS）内のビットFAULT_DAG、FAULT_USERSUPV、FAULT_RWは、CPLB関連の例外をトリガしたCPLBエントリの識別に使用されます（図 6-24を参照）。

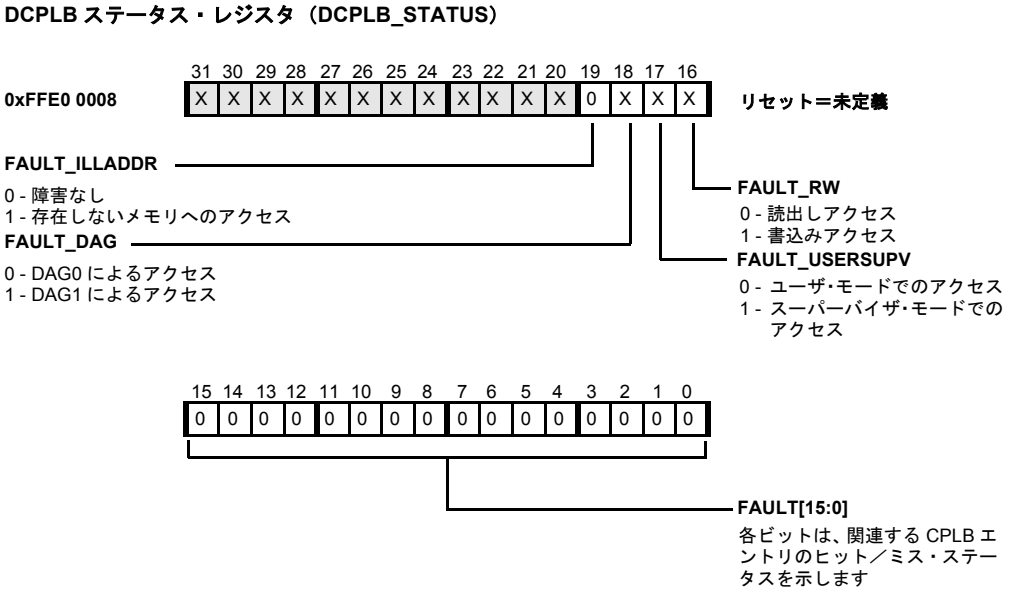


図 6-24. DCPLB ステータス・レジスタ

ICPLBステータス・レジスタ（ICPLB_STATUS）のビットFAULT_USERSUPVは、CPLB関連の例外をトリガしたCPLBエントリの識別に使用されます（図 6-25を参照）。

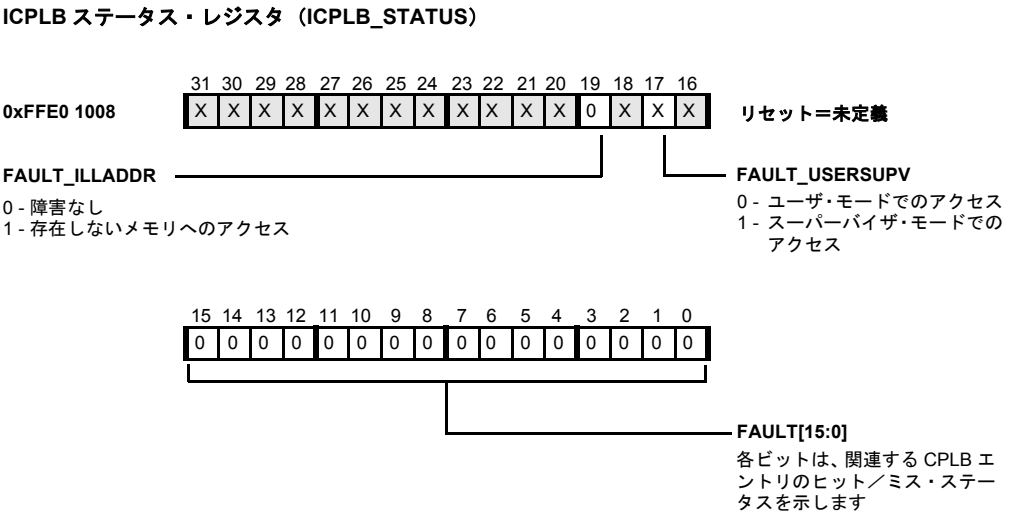


図 6-25. ICPLB ステータス・レジスタ

■ DCPLB_FAULT_ADDR レジスタと ICPLB_FAULT_ADDR レジスタ

DCPLBアドレス・レジスタ（DCPLB_FAULT_ADDR）とICPLBフォールト・アドレス・レジスタ（ICPLB_FAULT_ADDR）は、それぞれL1データ・メモリまたはL1命令メモリに障害を引き起こしたアドレスを保持します。図 6-26と図 6-27を参照。

DCPLB_FAULT_ADDR レジスタと ICPLB_FAULT_ADDR レジスタは、障害を起こした例外サービス・ルーチンでだけ有効です。

DCPLB アドレス・レジスタ（DCPLB_FAULT_ADDR）

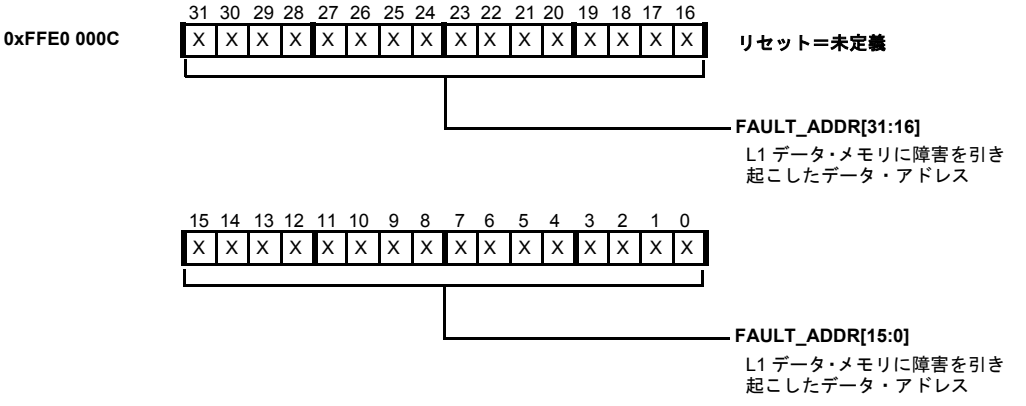


図 6-26. DCPLB アドレス・レジスタ

ICPLB フォールト・アドレス・レジスタ（ICPLB_FAULT_ADDR）

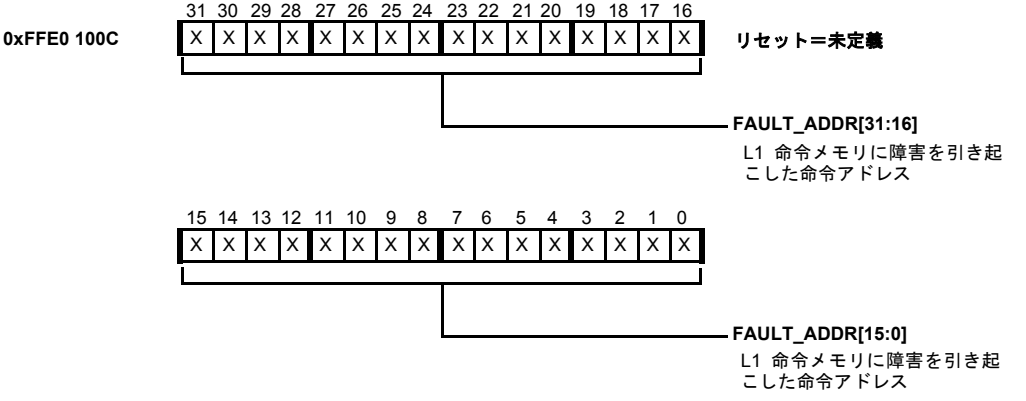


図 6-27. ICPLB フォールト・アドレス・レジスタ

メモリ・トランザクション・モデル

内部と外部の記憶領域は、リトル・エンディアンのバイト順にアクセスされます。図 6-28は、レジスタR0に格納されたデータ・ワードと、アドレス位置*addr*のメモリに格納されたデータ・ワードを示します。B0は、32ビット・ワードの最下位バイトを表します。

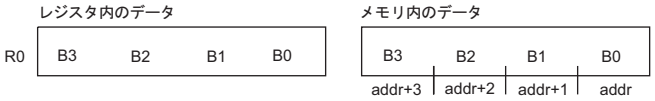


図 6-28. リトル・エンディアン順に格納されたデータ

図 6-29は、メモリに格納された16ビット命令と32ビット命令を示します。左側の図は、メモリに格納された16ビット命令を示します。命令の最上位バイトは上位アドレスに格納され（*addr+1*のバイトB1）、最下位バイトは下位アドレスに格納されます（*addr*のバイトB0）。

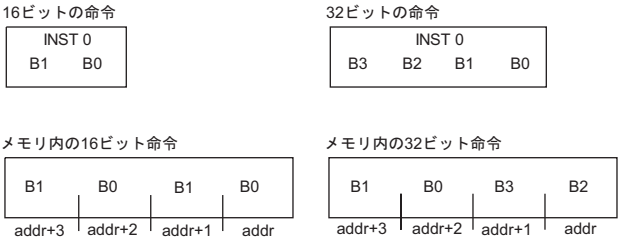


図 6-29. リトル・エンディアン順に格納された命令

右側の図は、メモリに格納された32ビット命令を示します。なお、命令の最上位16ビット・ハーフ・ワード（バイトB3とB2）は下位アドレス（*addr+1*と*addr*）に格納され、最下位ハーフ・ワード（バイトB1とB0）は上位アドレス（*addr+3*と*addr+2*）に格納されます。

ロード／ストア動作

Blackfinプロセッサ・アーキテクチャは、ロード／ストア・マシンのRISC概念に対応しています。このマシンはRISCアーキテクチャの特性であり、メモリ操作（ロードとストア）は、メモリ操作のターゲットを使用する算術関数から意図的に分離されています。この分離が行われた理由は、メモリ操作、特にオフチップ・メモリやI/Oデバイスにアクセスする命令では、完了するまでに複数のサイクルを要することがあり、通常はプロセッサを停止させて、1サイクル当たり1命令という命令実行レートを妨げるからです。

ロード動作をそれに関連する算術関数から分離すると、コンパイラやアセンブリ言語プログラマは、ロード命令とその従属する命令との間に関連のない命令を置くことができます。プロセッサがメモリ・システムからのデータ返却を待つ間に、関連のない命令は並行して実行されます。従属する動作がパイプラインの実行段に到達する前に値が返却された場合には、その動作は1サイクルで完了します。

書込み動作の場合、たとえデータが外部メモリやI/O位置に実際に書き込まれるまでに多数のサイクルが実行されとしても、ストア命令は実行されるとすぐに完了したと見なされます。この仕組みによって、プロセッサはクロック・サイクルごとに1つの命令を実行できます。このことは、書込みが完了するタイミングと、それ以降の命令が実行されるタイミングとの同期が保証されないことを意味します。さらに、大部分のメモリ操作という観点からは、この同期は重要でないと見なされます。

■ インターロック機能付きパイプライン

命令の実行に際して、Blackfinプロセッサ・アーキテクチャは、インターロック機能付きパイプラインを実装します。ロード命令が実行されるとき、読出し動作のターゲット・レジスタはメモリ・システムから値が返されるまでビジーとマークされます。それ以降の命令が新しい値が供給される前にこのレジスタにアクセスした場合には、メモリ操作が完了するまで

パイプラインはストールします。このストールによって、たとえ命令はメモリ読出しの完了前に実行を開始できる場合でも、ロードによって得られるデータを使用する必要のある命令は、レジスタ内の以前のデータや無効なデータを使用しないことが保証されます。

このメカニズムを利用すれば、プログラマやコンパイラはメモリ読出し動作の完了に実際に必要なサイクル数を知らなくても、ロードと読出しターゲットを使用する命令との間にある独立した命令を実行できます。ロードの直後にある命令が同じレジスタを使用する場合には、その命令は値が返されるまでストールします。したがって、プログラマの期待どおりに動作します。しかし、ロードの後で、同じレジスタを使用する命令よりも前に他の4つの命令が置かれた場合には、それらはすべて実行されて、プロセッサの全体的なスループットが向上します。

■ ロードとストアの順序付け

メモリ・アクセス命令とその前後の命令との同期の緩和は、ロードとストアの弱い順序付けと呼ばれます。弱い順序付けとは、メモリ操作の実際の完了のタイミングが、これらのイベントが発生する順序も含めてプログラムのソース・コードに現われる順序とは一致しない可能性があることを意味します。保証されている内容は次のとおりです。

- ロード動作は、それ以降の命令によって戻りデータが使用される前に完了します。
- 前もって書き込まれたデータを使用するロード動作では、更新された値を使用します。
- ストア動作は、最終的には最終目的地に伝搬します。

弱い順序付けによって、メモリ・システムは読出しに対して書込みよりも高い優先順位を付けることができます。この場合、パイプライン内のどこかにキュー登録され、まだ完了していない書込みは、それ以降の読出し動作によって保留されることがあります。そして、読出しは、書込みより前

に完了することができます。読出し動作にはその完了を待つ従属する動作があるため、読出しには書込みよりも高い優先順位が付けられます。一方、値をメモリに伝搬させるのに多くのサイクルが必要な場合には、プロセッサは書込み動作が完了したものと見なし、書込みはパイプラインをストールさせません。この動作によって、プログラムのソース・コードではプログラム・フロー的に書込みの後で出現する読出しが、実際には書込みの完了より前にその値を返すことがあります。この順序付けは、多くのメモリ命令の動作において性能上の大きな利点を提供します。しかし、副作用として、プログラマには不適切なシステム運用を回避するという負担が生じます。

I/Oデバイス・レジスタなどの非メモリ位置からの書込み／読出しに際しては、読出し／書込みの動作が完了する順序が重要になることがあります。たとえば、ステータス・レジスタの読出しは、コントロール・レジスタへの書込みに依存することがあります。アドレスが同じである場合には、読出しでは、実際のI/Oデバイス・レジスタではなく、書込みバッファから値を返すことになり、レジスタにおける読出し／書込みの順序が反転されることがあります。このような効果は、プログラムとペリフェラルの意図した動作に望ましくない副作用をもたらすことがあります。ロード／ストア動作の正確な（強い）順序付けを必要とするコードにおいて、このような効果の発生を防止するには、同期命令（CSYNCまたはSSYNC）を使用してください。

■ 同期命令

I/Oデバイスへの設定や制御を目的とする順次書込みのように、ロードとストアの強い順序付けが必要な場合には、コアまたはシステムの同期命令（それぞれCSYNCまたはSSYNC）を使用します。

CSYNC命令では、すべてのペンディング・コア動作が完了し、コア・バッファ（プロセッサ・コアとL1メモリとの間）がフラッシュされてから、次の命令に進むことが保証されます。ペンディング・コア動作には、保留中の割込み、投機的状態（分岐予測など）、または例外を含むことがあります。

次のコード・シーケンス例で考えてみます。

```
IF CC JUMP away_from_here
csync;
r0 = [p0];
away_from_here:
```

上記のコード例では、CSYNC命令によって以下のことが保証されます。

- 条件付き分岐（IF CC JUMP away_from_here）が解決されることで、条件が解決されてプロセッサ・ストア・バッファ内のエントリがフラッシュされるまでは、実行パイプラインにストールが発生します。
- すべての保留中の割込みや例外が処理されてから、CSYNCが完了します。
- ロードは、メモリから投機的にフェッチされません。

SSYNC命令によって、以前の動作のすべての副作用がL1メモリと残りのチップとの間のインターフェースを通じて伝搬されることが保証されます。SSYNC命令では、CSYNCのコア同期機能を実行するだけでなく、L1メモリとシステム領域との間の書込みバッファをフラッシュし、SSYNCが完了する前にアクノレッジを必要とするシステムに対して同期要求を生成します。

■ 投機的ロード実行

メモリからのロード動作は、メモリ値の状態を変更しません。したがって、それ以降のロード命令に対して投機的なメモリ読出し操作を発行しても、望ましくない副作用は生じないのが普通です。条件付き分岐命令の後にロードが続くといった、いくつかのコード・シーケンスでは、条件付き分岐が解決される前にメモリ・システムへの読出し要求を投機的に発行することで性能が向上する場合もあります。次に例を示します。

```
            IF CC JUMP away_from_here
            R0 = [P2];
            …
away_from_here:
```

分岐が行われる場合には、ロードはパイプラインからフラッシュされ、返却のプロセスにある結果は無視できます。逆に分岐が行われない場合には、メモリからは、分岐条件が解決されるまで動作がストールされた場合に比べて早く正しい値が返されます。

しかし、I/Oデバイスの場合には、要求された読出しの数に基づいて値を変更するレジスタやFIFOから順次データを返すペリフェラルに対しては、望ましくない副作用を生じることがあります。この効果を回避するには、同期命令（CSYNCまたはSSYNC）を使用して読出し動作間の正しい動作を保証します。

ストア動作では、投機的にメモリをアクセスすることはありません。なぜなら、これによって命令を実行すべきであったかどうかを判定する前に、メモリ値が変更される可能性があるからです。

■ 条件付きロード動作

同期命令では、すべての投機的状態が解決されてから、ロード命令がメモリ参照を開始します。しかし、ロード命令は割込み可能であるため、ロード命令自身が複数のメモリ読出し動作を生成することもあります。同期命令の完了とロード命令の完了との間に、十分な優先順位を持つ割込みが発生した場合には、シーケンサはロード命令をキャンセルします。割込みの実行後、割込まれたロードは再実行されます。この方法は、割込み遅延を最小限に抑えます。しかし、メモリ読出しサイクルが開始されてからロードがキャンセルされ、その後にロードが再実行された後の2番目の読出し動作が続く可能性もあります。多くのメモリ・アクセスでは、同じメモリ・アドレスの複数の読出しには副作用がありません。しかし、ペリフェ

ラル・データFIFOなど、いくつかのメモリマップド・デバイスでは読出しは破壊的です。デバイスが読み出されるたびにFIFOが前進するため、データの回復と再読出しはできません。

特定のアドレス位置での読出し／書込み動作の数に状態依存するメモリマップド・デバイスにアクセスする場合には、割込みをディスエーブルにしてから、ロード／ストア動作を実行します。

メモリの操作

ここでは、メモリ内でのデータの整列と、タスク間でのセマフォをサポートするメモリ操作について説明します。また、MMRレジスタとコアMMRのプログラミング例についても簡単に説明します。

■ 整列

非整列のメモリ操作には、直接対応していません。非整列のメモリ参照は、非整列アクセス例外イベントを生成します（4-42ページの「例外」を参照）。しかし、一部のデータストリーム（8ビットのビデオ・データなど）では、メモリ内で正常に非整列であり得ます。整列例外をディスエーブルにするには、DISALGNEXCPT命令を使用します。さらに、クワッド8ビット・グループ内のいくつかの命令では、整列例外を自動的にディスエーブルにします。

■ キャッシュ・コヒーレンシ

共有データの場合、ソフトウェアは、必要に応じてキャッシュ・コヒーレンシに対応する必要があります。そのためには、FLUSH命令（6-45ページの「データ・キャッシュ制御命令」を参照）またはコアMMRを通じての明示的なライン無効化（6-46ページの「データ・テスト・レジスタ」を参照）、あるいはその両方を使用します。

■ アトミック操作

このプロセッサは、単一のアトミック操作TESTSETを提供します。アトミック操作は、タスク間でセマフォをサポートする過程で割込み不可能なメモリ操作を提供するために使用されます。TESTSET命令では、間接的にアドレス指定されたメモリ・ハーフ・ワードをロードし、下位バイトがゼロかどうかをテストしてから、他のビットに影響を与えることなく、下位メモリ・バイトの最上位ビット（MSB）をセットします。そのバイトがもともとゼロである場合には、CCビットがセットされます。バイトがもともと非ゼロである場合には、CCビットがクリアされます。このメモリ・トランザクションのシーケンスはアトミックです。つまり、ハードウェア・バス・ロッキングによって、この命令のテスト部分とセット部分との間には他のメモリ操作が発生しないことが保証されます。TESTSET命令はコアによって割込みできます。この場合には、割込みから復帰した時点でTESTSET命令が再び実行されます。

TESTSET命令では、4Gバイトのメモリ空間全体をアドレス指定できます。しかし、オンコア・メモリ（L1またはMMR空間）をターゲットにしないでください。このメモリへのアトミック・アクセスには対応していません。

メモリ・アーキテクチャでは、たとえアドレスに対するCPLBディスクリプタがキャッシュ有効アクセスを示す場合でも、アトミック操作を常にキャッシュ禁止アクセスとして扱います。しかし、メモリのキャッシュ可能領域でTESTSET 命令を実行することはお勧めできません。これは、TESTSET命令が実行されたとき、メモリのキャッシュ可能位置がコヒーレントであることを保証できないためです。

■ メモリマップド・レジスタ

MMRの予備空間は、メモリ空間の最上位に位置しています（0xFFC0 0000）。この領域はキャッシュ不可能と定義され、システムMMR（0xFFC0 0000～0xFFE0 0000）とコアMMR（0xFFE0 0000～0xFFFF FFFF）に分割されます。

強い順序付けが必要な場合には、MMRへのストアの後に同期命令を置いてください。詳細については、6-72ページの「ロード／ストア動作」を参照してください。

すべてのMMRは、スーパーバイザ・モードでだけアクセスできます。ユーザ・モードでMMRにアクセスすると、保護違反例外が発生します。

すべてのコアMMRは、32ビットの整列アクセスを使用して読出し／書込みされます。しかし、いくつかのMMRでは、32ビット未満でしか定義されていません。この場合、未使用のビットは予備です。システムMMRは16ビットの場合があります。

存在しないMMRへのアクセスは、不正アクセス例外を生成します。システムは、読出し専用MMRへの書込みを無視します。

付録Aに、すべてのコアMMRのまとめを示します。付録Bには、すべてのシステムMMRのまとめを示します。

■ コア MMR のプログラミング・コード例

コアMMRは、整列された32ビット・ワードとしてだけアクセスできます。MMRへの非整列アクセスは例外イベントを生成します。リスト6-1には、一般的なコアMMRの操作に必要な命令を示します。

リスト 6-1. コア MMR のプログラミング

```
CLI R0;    /*  割込みを停止させ、IMASKを保存*/
```

```
P0 = MMR_BASE;    /*  MMRのベースをロードする32ビット命令 */
R1 = [P0 + TIMER_CONTROL_REG];    /*  制御レジスタの値を取得 */
BITSET R1, #N;    /*  ビットNをセット */
[P0 + TIMER_CONTROL_REG] = R1;    /*  制御レジスタを復元 */
CSYNC;    /*  制御レジスタの書込みを保証 */
STI R0;    /* 割込みをイネーブル */
```

CLI命令では、IMASKレジスタの内容を保存し、IMASKをクリアして割込みをディスエーブルにします。STI命令では、IMASKレジスタの内容を復元して割込みをイネーブルにします。CLIとSTIの間にある命令は、割込み不可能です。

用語集

以下の用語は、メモリの説明に使用されます。

インデックス：アレイ・エレメント（たとえば、ライン・インデックス）の選択に使用されるアドレス部分。

ウェイ：ウェイ：*N*ウェイ・キャッシュ内のライン記憶素子のアレイです（6-16ページの図6-7を参照）。

キャッシュ・ヒット：キャッシュ内の現在有効なエントリに適合するメモリ・アクセスです。

キャッシュ・ブロック：キャッシュ・ミスの結果として、メモリの次の階層とキャッシュとの間で転送されるメモリの最小単位です。

キャッシュ・ミス：キャッシュ内のどの有効エントリにも一致しないメモリ・アクセスです。

キャッシュ・ライン：キャッシュ・ブロックと同じです。この章では、キャッシュ・ブロックの代わりにキャッシュ・ラインを使用します。

最長時間未使用（LRU）アルゴリズム：キャッシュによって使用される置換アルゴリズムであり、最も長期間にわたって使用されていないラインを最初に置換します。

セット：*N*ウェイ・キャッシュのウェイに含まれる*N*ライン保管場所のグループであり、アドレスの**インデックス**・フィールドによって選択されます（6-16ページの図6-7を参照）。

セット・アソシエイティブ：ライン配置のセット（またはウェイ）数を制限するキャッシュ・アーキテクチャです。

ダーティ／ モディファイ：タグとともに格納される状態ビットであり、データ・キャッシュ・ライン内のデータがソース・メモリからコピーされて以降に変更されたかどうか（したがって、そのソース・メモリ内で更新する必要があるかどうか）を示します。

タグ：キャッシュされたラインが表すメモリ内の特定アドレス・ソースを識別するために、キャッシュされたデータ・ラインと一緒に格納される上位アドレス・ビットです。

置換ポリシー：キャッシュ・ミス時に置換すべきラインを判断するために、プロセッサによって使用される機能です。通常は、LRUアルゴリズムを使用します。

直接マップ：各ラインがキャッシュ内の出現場所を1つだけ持つというキャッシュ・アーキテクチャであり、1ウェイ・アソシエイティブとも呼ばれます。

排他的、クリーン：データ・キャッシュ・ラインの状態であり、ラインが有効であって、ラインに含まれるデータがソース・メモリ内のデータと一致することを示します。クリーンなキャッシュ・ライン内のデータは、置換される前にソース・メモリに書き込む必要はありません。

ビクティム：メモリに書き込まなければ、キャッシュ・ライン割当て用に空き空間に置換できないダーティなキャッシュ・ライン。

フル・アソシエイティブ：各ラインをキャッシュ内のどこにでも置けるキャッシュ・アーキテクチャです。

無効：キャッシュ・ラインの状態の説明です。キャッシュ・ラインが無効な場合、キャッシュ・ラインの一致は発生しません。

有効：タグとともに格納される状態ビットであり、対応するタグとデータが最新かつ正確であり、メモリ・アクセス要求を満たすために使用できることを示します。

ライトスルー：キャッシュ書込みポリシーの1つであり、ストア・スルーとも呼ばれます。書込みデータは、キャッシュ・ラインとソース・メモリの両方に書き込まれます。変更されたキャッシュ・ラインは、置換時にソース・メモリに**書き込まれません**。キャッシュ・ラインは、読出し時に割り当てる必要があり、モードによっては書込み時に割り当てることもできます。

ライトバック：キャッシュ書込みポリシーの1つであり、**コピーバック**とも呼ばれます。書込みデータは、キャッシュ・ラインにだけ書き込まれます。変更されたキャッシュ・ラインは、キャッシュ・ラインの置換時にだけソース・メモリに書き込まれます。キャッシュ・ラインは、読出しと書込みの両方で割り当てられます。

リトル・エンディアン：Blackfinプロセッサの固有のデータ格納フォーマットです。ワードとハーフ・ワードは、データ保管場所の最下位バイト・アドレスに最下位バイトを置き、最上位バイト・アドレスに最上位バイトを置いてメモリ（とレジスタ）に格納されます。

レベル1（L1）メモリ：メモリとコアとの間にメモリ・サブシステムを介在させることなく、コアによって直接アクセスされるメモリです。

第7章　チップ・バス階層

この章では、オンチップ・バス、システム内のデータ移動、およびシステム構造を決定する要因について説明します。また、システム内部チップ・インターフェース、システム相互接続、および関連するシステム・バスについても説明します。

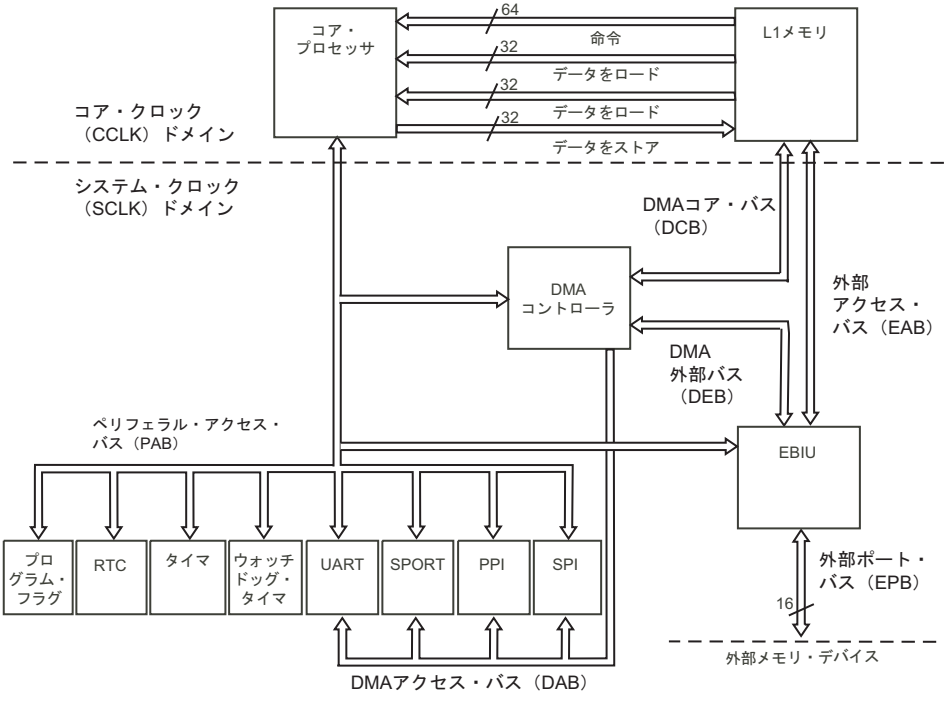


図 7-1.　プロセッサ・バス階層

内部インターフェース

図 7-1は、コア・プロセッサとシステム境界に加えて、その間のインターフェースを示します。

内部クロック

コア・プロセッサ・クロック（CCLK）周波数は、CLKINを基にして自由に設定可能です。CCLK周波数はフェーズ・ロック・ループ（PLL）出力周波数から分周されます。この分周器比率を設定するには、PLL分周レジスタのCSELパラメータを使用します。

ペリフェラル・アクセス・バス（PAB）、DMAアクセス・バス（DAB）、外部アクセス・バス（EAB）、DMAコア・バス（DCB）、DMA外部バス（DEB）、外部ポート・バス（EPB）、外部バス・インターフェース・ユニット（EBIU）は、システム・クロック周波数（SCLKドメイン）で動作します。この分周器比率は、PLL分周レジスタのSSELパラメータを使用して設定し、しかもこれらのバスが、コア・クロック周波数以下の速度で『ADSP-BF531/ADSP-BF532/ADSP-BF533 Embedded Processor Data Sheet』の指定に基づいて動作するように設定しなければいけません。

これらのバスは、消費電力を減らしたり、コア・プロセッサが最適周波数で動作できるようにしたりするために、プログラマブルな周波数の範囲で変更させることもできます。なお、すべての同期ペリフェラルは、そのタイミングをSCLKから取得します。たとえば、UARTクロック・レートを決定するには、このクロック周波数をさらに分周します。

コアの概要

説明のために、レベル1メモリ（L1）はコアの説明に含めてあります。L1メモリには、64ビット命令バスと2本の32ビット・データ・バスによって、プロセッサ・コアからフル帯域幅でアクセスできます。

図 7-2は、コア・プロセッサと、ペリフェラルや外部メモリ・リソースへのインターフェースを示します。

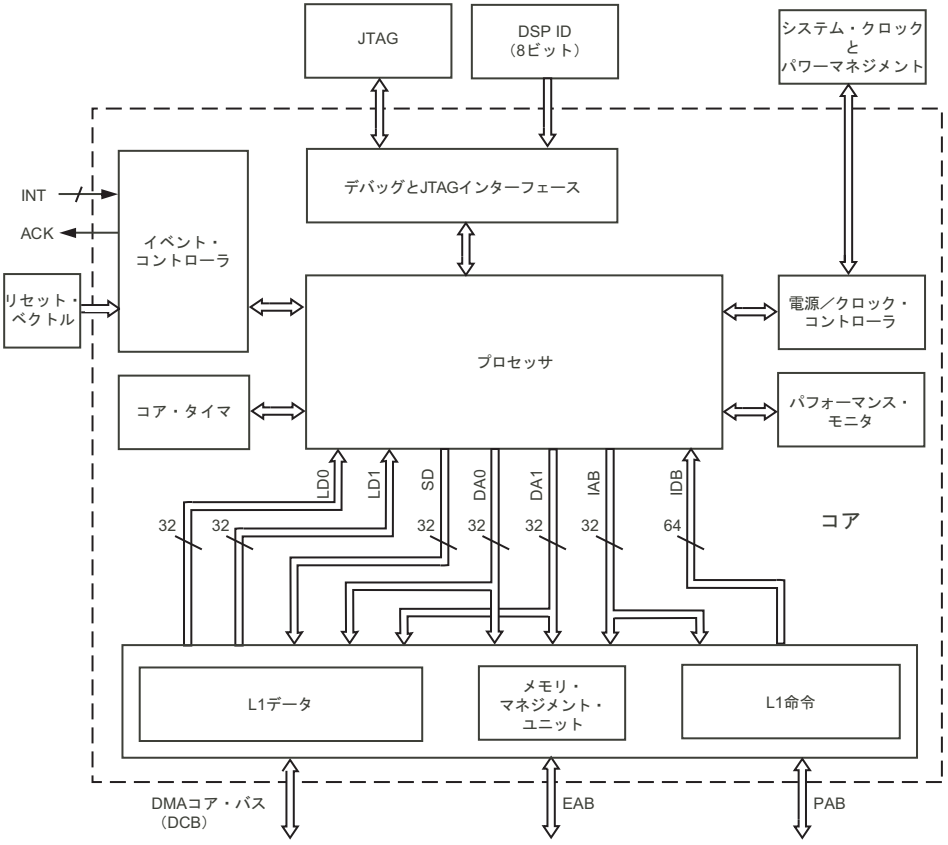


図 7-2.　コアのブロック図

コアは、サイクル当たり3つまでの同時オフコア・アクセスを生成できます。

プロセッサとL1メモリとの間のコア・バス構造は、フル・コア周波数で動作し、64ビットまでのデータ・パスを備えています。

命令要求がフィルされるとき、64ビット読出しには1つの64ビット命令、または16/32/64ビット（部分）命令の任意の組合わせを含むことができます。

キャッシュがイネーブルにされると、32バイトのライン・フィル・バースト動作に対応するために、4つの64ビット読出し要求が発行されます。これらの要求は、2番目以降の各転送が1つの連続したサイクルでフィルされるようにパイプライン化されます。

システムの概要

システムには、システム割込み、テスト／エミュレーション、クロックおよびパワーマネジメント用のコントローラが内蔵されています。コアとシステムとの間のクロック・ドメイン・トランザクションに対応するために、同期クロック・ドメイン変換が提供されています。

システム・インターフェース

プロセッサ・システムには、以下の要素が内蔵されています。

- ペリフェラル・セット（タイマ、リアルタイム・クロック、プログラマブル・フラグ、UART、SPORT、PPI、ウォッチドッグ・タイマ、SPI）
- 外部メモリ・コントローラ（EBIU）
- DMAコントローラ

- 上記の要素、システム、オプションの外付け（オフチップ）リソースを結ぶインターフェース

7-3ページの図7-2を参照してください。

以下の節では、システムとペリフェラルとの間のオンチップ・インターフェースについて説明します。

- ペリフェラル・アクセス・バス（PAB）
- DMAアクセス・バス（DAB）
- DMAコア・バス（DCB）
- DMA外部バス（DEB）
- 外部アクセス・バス（EAB）

外部バス・インターフェース・ユニット（EBIU）は、チップ・ピンのプライマリ・バスです。EBIUについては、第17章「外部バス・インターフェース・ユニット」を参照してください。

■ ペリフェラル・アクセス・バス（PAB）

プロセッサには専用のペリフェラル・バスがあります。低遅延のペリフェラル・バスはコア・ストールを最小限に維持し、速度が重視されるペリフェラルに対して管理しやすい割込み遅延を可能にします。PABを通じてアクセスされるすべてのペリフェラル・リソースは、プロセッサ・メモリ・マップのシステムMMR空間にマッピングされます。コアは、PABバスを通じてシステムMMR空間にアクセスできます。

コア・プロセッサにはバイト・アドレス指定機能がありますが、プログラミング・モデルは、システムMMRへの32ビット（整列）アクセスだけに限定されます。この領域へのバイト・アクセスはできません。

▶ PAB 調停

コアは、このバスでの唯一のマスターです。調停は不要です。

▶ PAB 性能

PABの場合、主要な性能基準はスループットではなく遅延です。PABでの読出し／書込み転送に対する転送遅延は2 SCLKサイクルです。

たとえば、コアはPABスレーブに対してアクセス当たり最高32ビットで転送できます。コア・クロックをシステム・クロックの周波数の2倍で動作させると、システムMMRの最初とそれに続く読出し／書込みアクセスでは4コア・クロック（CCLK）の遅延が発生します。

PABは、SCLKの最大周波数を備えています。

▶ PAB エージェント（マスター、スレーブ）

プロセッサ・コアは、PABでのバス動作を管理できます。すべてのペリフェラルにはペリフェラル・バス・スレーブ・インターフェースがあるため、コアは制御状態とステータス状態にアクセスできます。これらのレジスタは、メモリ・マップのシステムMMR空間にマッピングされます。付録Bにシステム MMRアドレスを示します。

PABバス上のスレーブは、次のとおりです。

- イベント・コントローラ
- クロックおよびパワーマネジメント・コントローラ
- ウォッチドッグ・タイマ
- リアルタイム・クロック（RTC）
- Timer0、1、2
- SPORT0

- SPORT1
- SPI
- プログラマブル・フラグ
- UART
- PPI
- 非同期メモリ・コントローラ（AMC）
- SDRAMコントローラ（SDC）
- DMAコントローラ

■ DMA アクセス・バス（DAB）、DMA コア・バス（DCB）、DMA 外部バス（DEB）

DMA対応のペリフェラルは、DABバス、DCBバス、DEBバスを使用して、メモリへのコア帯域幅をほとんど低下させることなく、オンチップ／オフチップ・メモリにアクセスできます。

▶ DAB 調停

プロセッサ・システムには、メモリDMAコントローラを含めて6つのDMA対応ペリフェラルがあります。これらのデバイスは、12本のDMAチャンネルとバス・マスターを利用できます。ペリフェラルDMAコントローラは、ペリフェラルと内部／外部メモリとの間でデータを転送できます。メモリDMAコントローラの読出し／書込みチャンネルは、DABを通じてディスクリプタ・リストにアクセスします。

SRAMとして設定されたL1への調停について、DCBはコア・プロセッサよりも高い優先順位を持っています。オフチップ・メモリの場合、EPBへのアクセスについては、コアは（デフォルトで）DEBよりも高い優先順位を持っています。DABについては、プロセッサにはプログラマブルな

優先順位調停ポリシーがあります。表 7-1に、デフォルトの調停優先順位を示します。さらに、EBIU_AMGCTLレジスタのCDPRIOビットの設定によって、EPBへのすべてのDEBトランザクションに外部メモリへのコア・アクセスよりも高い優先順位を持たせることができます。このビットの使用はアプリケーションに依存します。たとえば、長いアクセス・タイムを持つ非同期メモリにマッピングされたペリフェラルをポーリングしている場合、デフォルトでは、コアはDMA要求に「勝り」ます。CDPRIOビットの設定によって、コアはDMA要求が処理されるまで先送りされます。

表 7-1.　DAB、DCB、DEB の調停優先順位

DAB、DCB、DEB マスター	デフォルトの調停優先順位
PPI	0 - 最高
SPORT0 RCV DMA コントローラ	1
SPORT1 RCV DMA コントローラ	3
SPORT0 XMT DMA コントローラ	2
SPORT1 XMT DMA コントローラ	4
SPI DMA コントローラ	5
UART RCV コントローラ	6
UART XMT コントローラ	7
メモリ DMA0（デスティネーション）コントローラ	8
メモリ DMA0（ソース）コントローラ	9
メモリ DMA1（デスティネーション）コントローラ	10
メモリ DMA1（ソース）コントローラ	11 - 最低

▶ DAB、DCB、DEB の性能

プロセッサのDABでは、16ビットまたは32ビットのデータ転送をサポートします。データ・バスは16ビット幅であり、その最大周波数は『ADSP-BF531/ADSP-BF532/ADSP-BF533 Embedded Processor Data Sheet』で指定されています。

DABには、L1メモリへの専用ポートがあります。コア・アクセスとDMAアクセスが同じメモリ・バンク（L1では4Kバイト・サイズ）に対するものでない限り、ストールは発生しません。衝突がある場合には、DMAが優先順位の最も高いリクエスタであり、コアがそれに続きます。

なお、コア・プロセッサによるロック転送（たとえば、TESTSET命令の実行）では、メモリ・ロックがアサート解除されるまで、アドレス指定されたメモリ・バンクやリソースに対する調停は実質的にディスエーブルにされます。DMAコントローラでは、ロック転送を実行できません。

L1メモリへのDMAアクセスは、別のDMAチャンネルからすでに開始されているアクセスによってのみストールされることがあります。このようなストールに起因する遅延量は、調停遅延に付加されます。

コア・プロセッサとDABは、EBIUを通じての外部メモリへのアクセスを調停する必要があります。オフチップ・メモリ・デバイスの読出しに必要な遅延に加えて、このような調停遅延が生じるため、DABスループットが著しく低下して、ペリフェラル・データ・バッファのアンダーフロー／オーバーフローにつながることもあります。メモリDMAコントローラ以外のDMAペリフェラルを使用し、DMAアクセスのターゲットを外部メモリとする場合には、具体的なトラフィック・パターンを慎重に解析する必要があります。内部メモリをターゲットとするアイソクロナス・ペリフェラルには、十分な帯域幅と適切な最大調停遅延が割り当てられていることを確認してください。

▶ DAB バス・エージェント（マスター）

7-9ページの表7-1に示すように、DMAアクセスのソースになれるすべてのペリフェラルは、このバスのマスターです。1つのアービタが、DABにアクセスするためのプログラマブルな優先順位調停ポリシーを提供します。

複数のDMAマスター・チャンネルがDABをアクティブに要求している場合、DABのパイプライン化された設計によってバス使用率はかなり高くなります。バス調停サイクルは、先行するDMAアクセスのデータ・サイクルと同時に発生します。

■ 外部アクセス・バス（EAB）

EABによって、プロセッサ・コアは、オフチップ・メモリに直接アクセスできます。

■ 外部バスの調停

外部ポート・バスの潜在的なマスター間には競合の可能性があるため、このバスのインターフェース・リソースの使用を調停する必要があります。固定優先順位の調停方式が使用されます。つまり、EABを介するコア・アクセスでは、DMA外部バス（DEB）からのアクセスよりも優先順位が高くなります。

■ DEB/EAB 性能

DEBとEABは、8ビットまたは16ビットのデータ型のシングル・ワード・アクセスに対応します。DEBとEABは『ADSP-BF531/ADSP-BF532/ADSP-BF533 Embedded Processor Data Sheet』で指定された最大SCLK周波数まで、PABやDABと同じ周波数で動作します。

メモリDMA転送では、同じメモリ位置に対する繰返しアクセスが生じることがあります。メモリDMAコントローラでは、オンチップ／オフチップ・メモリに同時アクセスする可能性があるため、かなりのスループットを達成できます。オンチップ／オフチップ・メモリ・アクセスのスループット・レートは、2つのアクセスのうち、遅い方によって制限されます。

オンチップ・メモリ間またはオフチップ・メモリ間の転送では、バースト・アクセスは同時に発生しません。したがって、転送速度を計算するには、各転送時間と各転送間での追加サイクルを加算します。

表 7-2は、多くのタイプの16ビット・メモリDMA転送を示します。この表では、他のDMA動作は進行中の動作と衝突していないものと想定します。表の値は理論値です。実際のハードウェアで測定すると、EBIUに接続されているデバイスに関するさまざまな理由によって、これらの値は高くなることがあります。

非DMAアクセス（たとえば、EABを介するコア・アクセス）では、SDRAMへの32ビット・アクセス（書式は`R0 = [P0]`であり、P0はSDRAM内のアドレスをポイント）は、2つの16ビット・アクセス（書式は`R0 = W[P0++]`であり、P0はSDRAM内のアドレスをポイント）を実行する場合よりも常に効率的です。この例では、32ビットのSDRAM読出しには10 SCLKサイクルが必要ですが、2つの16ビット読出しにはそれぞれ9 SCLKサイクルが必要です。

表 7-2.　外部メモリへの DMA アクセスの性能

ソース	デスティネーション	n ワードに対する概略の SCLK（DMA の開始から最後の割込みまで）
16 ビット SDRAM	L1 データ・メモリ	n + 14
L1 データ・メモリ	16 ビット SDRAM	n + 11
16 ビット非同期メモリ	L1 データ・メモリ	xn + 12。ここで x は、待ち状態の数＋セットアップ／ホールドSCLKサイクル（最小の x = 2）
L1 データ・メモリ	16 ビット非同期メモリ	xn + 9。ここで x は、待ち状態の数＋セットアップ／ホールドSCLKサイクル（最小の x = 2）
16 ビット SDRAM	16 ビット SDRAM	10 + (17n/7)
16 ビット非同期メモリ	16 ビット非同期メモリ	10 + 2xn。ここで x は、待ち状態の数＋セットアップ／ホールドSCLKサイクル（最小の x = 2）
L1 データ・メモリ	L1 データ・メモリ	2n + 12

第8章　ダイナミック・パワー・マネジメント

この章では、プロセッサのダイナミック・パワー・マネジメント機能について説明します。これには以下の機能が含まれます。

- クロッキング
- フェーズ・ロック・ループ（PLL）
- ダイナミック・パワー・マネジメント・コントローラ
- 動作モード
- 電圧制御

クロッキング

プロセッサへの入力クロックであるCLKINは、オンチップ・フェーズ・ロック・ループ（PLL）モジュールによる正確な内部クロック逓倍を可能にするため、必要なクロック周波数、デューティ・サイクル、安定性を提供します。通常の動作時には、ユーザはCLKINの増倍率でPLLをプログラムします。結果として得る逓倍された信号は、電圧制御発信器（VCO）クロックです。その後、ユーザ・プログラマブルな値でVCOクロック信号を分周してコア・クロック（CCLK）を生成します。

VCO信号をユーザ・プログラマブルな値で分周してシステム・クロック（SCLK）を生成します。このSCLK信号は、ペリフェラル・アクセス・バス（PAB）、DMAバス（DAB）、外部アドレス・バス（EAB）、外部バス・インターフェース・ユニット（EBIU）をクロック駆動します。

これらのバスは、PLL周波数を1～15（SCLKドメイン）で分周した周波数で動作します。PLL分周レジスタのSSELパラメータを使用して、これらのバスが『ADSP-BF531/ADSP-BF532/ADSP-BF533 Embedded Processor Data Sheet』で指定される最大SCLKレート以下で動作できる分周器値を選択します。

性能と消費電力を最適化するため、プロセッサでは、コア周波数とシステム・クロック周波数を粗調整で動的に変更できます。微調整では、PLLクロック周波数も変更できます。

■ フェーズ・ロック・ループとクロック制御

コアとシステムのクロック生成を行うため、プロセッサでは、プログラマブルなステート・マシン制御を備えたアナログPLLを使用します。

PLL設計では、広範なアプリケーションに対応し、性能、柔軟性、消費電力の制御が主要な特長である低価格の汎用プロセッサ、組込みアプリケーション、ポータブル・アプリケーションにウェイトを置いています。このようにアプリケーション範囲が広いため、クロック生成回路には広範囲の周波数が要求されます。入力クロックには、水晶、水晶発振器、または外付けシステム・クロック発振器から出力されるバッファされ整形されたクロックを使用できます。

PLLでは、ダイナミック・パワー・マネジメント・コントローラ（DPMC）ブロックと対話処理を行って、プロセッサにパワーマネジメント機能を提供します。DPMCの詳細については、8-12ページの「ダイナミック・パワー・マネジメント・コントローラ」を参照してください。

▶ PLL の概要

PLLは最大VCO周波数を生成するために、広範囲な逓倍率を提供して入力クロックCLKINの逓倍を行います。この広範な逓倍範囲を達成するため、プロセッサはPLLフィードバック回路のプログラマブル分周器と出力設定ブロックを組み合わせて使用します。

図 8-1は、PLL回路の概念モデル、設定入力、結果として得られる出力を示します。この図では、VCOは中間クロックであり、そこからコア・クロック（CCLK）とシステム・クロック（SCLK）が得られます。

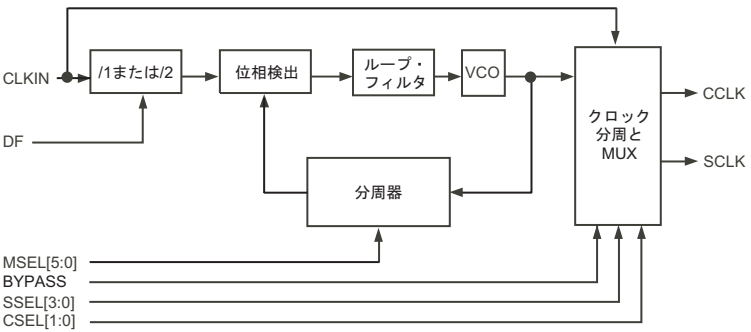


図 8-1.　PLL のブロック図

■ PLL クロック逓倍率

PLL コントロール・レジスタ（PLL_CTL）はPLLの動作を管理します。PLL_CTL レジスタの詳細については、8-8ページの「PLL_CTL レジスタ」を参照してください。

周波数分周（DF）ビットと乗算器選択（MSEL[5:0]）フィールドでは、さまざまなPLLクロック分周器を設定します。

- DFは入力分周器をイネーブルにします
- MSEL[5:0]は帰還分周器を制御します

  MSELのリセット値は0xAです。この値は、ブート・コードで起動時に再設定できます。

表 8-1は、さまざまなMSEL設定とDF設定に対するVCO増倍率を示します。

この表に示すように、MSEL[5:0]とDFのさまざまな組合わせによって、同じVCO周波数を生成できます。特定のアプリケーションでは、1つの組合わせで低消費電力を提供したり、VCO最大周波数を満たします。通常の条件では、一般にDFを1に設定すると、低消費電力が得られます。CLKIN、CCLK、VCOの最大／最小周波数については、『ADSP-BF531/ADSP-BF532/ADSP-BF533 Embedded Processor Data Sheet』を参照してください。

表 8-1.　MSEL のエンコーディング

信号名 MSEL[5:0]	VCO 周波数 DF = 0	DF = 1
0	64x	32x
1	1x	0.5x
2	2x	1x
N = 3–62	Nx	0.5Nx
63	63x	31.5x

▶ コア・クロック／システム・クロックの比率制御

8-5ページの表8-2は、VCO周波数とコア・クロックとのプログラマブルな関係を示します。8-6ページの表8-3には、VCO周波数とシステム・クロックとの関係を示します。なお、分周器比率を選択して、SCLKを『ADSP-BF531/ADSP-BF532/ADSP-BF533 Embedded Processor Data Sheet』で指定された周波数に制限する必要があります。SCLKは、すべての同期型システム・レベル・ロジックを駆動します。

分周器比率制御ビットCSELとSSELは、PLL分周レジスタ（PLL_DIV）にあります。このレジスタについての詳細は、8-7ページの「PLL_DIVレジスタ」を参照してください。付録Bにはレジスタ・アドレスを示します。

CSEL[1:0]のリセット値は0x0 (/1)であり、SSEL[3:0]のリセット値は0x5です。これらの値は、ブート・コードによって起動時に再度プログラムできます。

CSELとSSELの値を動的に変更するには、PLL_DIVに適切な値を書き込みます。なお、コア・クロックの分周器比率は、システム・クロックの分周器比率よりも大きくできません。PLL_DIVレジスタに不正な値がプログラムされた場合には、SCLK分周器は自動的にコア・クロック分周器以上の値に増やされます。

PLL_DIVレジスタはいつでもプログラム可能であり、アイドル状態に入ることなく、CCLKとSCLKの分周値を変更できます。

表 8-2.　コア・クロック比率

信号名 CSEL[1:0]	分周器比率 VCO/CCLK	周波数比の例（MHz） VCO	 CCLK
00	1	300	300
01	2	600	300
10	4	600	150
11	8	400	50

PLLコントロール・レジスタ（PLL_CTL）内の制御ビットMSELとDFが一定である限り、PLLはロックされています。

表 8-3.　システム・クロック比率

信号名 SSEL[3:0]	分周器比率 VCO/SCLK	周波数比の例（MHz） VCO	 SCLK
0000	予備	N/A	N/A
0001	1:1	100	100
0010	2:1	200	100
0011	3:1	400	133
0100	4:1	500	125
0101	5:1	600	120
0110	6:1	600	100
N = 7–15	N:1	600	600/N

PLL_DIVに新しいSSEL値を書き込んでクロック比率を変更する場合には、SCLK周波数の変化によってイネーブル状態のペリフェラルにデータ損失が発生しないように注意してください。

■ PLL レジスタ

PLLへのユーザ・インターフェースには、4本のメモリマップド・レジスタ（MMR）を使用します。

- PLL分周レジスタ（PLL_DIV）
- PLLコントロール・レジスタ（PLL_CTL）
- PLLステータス・レジスタ（PLL_STAT）
- PLLロック・カウント・レジスタ（PLL_LOCKCNT）

これら4本のレジスタはすべて16ビットMMRであり、整列された16ビットの読出し／書込みでアクセスする必要があります。

▶ PLL_DIV レジスタ

PLL分周レジスタ（PLL_DIV）では、PLL出力クロックを分周してプロセッサのコア・クロック（CCLK）とシステム・クロック（SCLK）を生成します。これらの値は、処理中に独立に変更して、PLL状態を変更せずに消費電力を減らすことができます。唯一の制約として、得られるCCLK周波数はSCLK周波数以上であり、SCLK は『ADSP-BF531/ADSP-BF532/ADSP-BF533 Embedded Processor Data Sheet』で指定される許容範囲内に収まることが必要です。CCLKとSCLKの分周値の設定が誤っている場合には、SCLK値は自動的にコア・クロック以下の値に調整されます。図 8-2は、PLL_DIV レジスタのビットを示します。

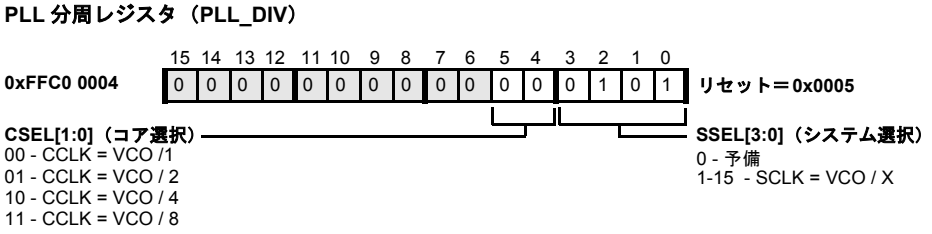


図 8-2.　PLL 分周レジスタ

▶ PLL_CTL レジスタ

PLLコントロール・レジスタ（PLL_CTL）では、PLLの動作を制御します（図 8-3を参照）。なお、PLL_CTLレジスタへの変更は、すぐに有効にはなりません。一般に、変更を実現するには、まずPLL_CTLレジスタに新しい値を設定し、次に特定のPLLプログラミング・シーケンスを実行する必要があります。8-20ページの「PLLプログラミング・シーケンス」を参照してください。

PLL コントロール・レジスタ（PLL_CTL）

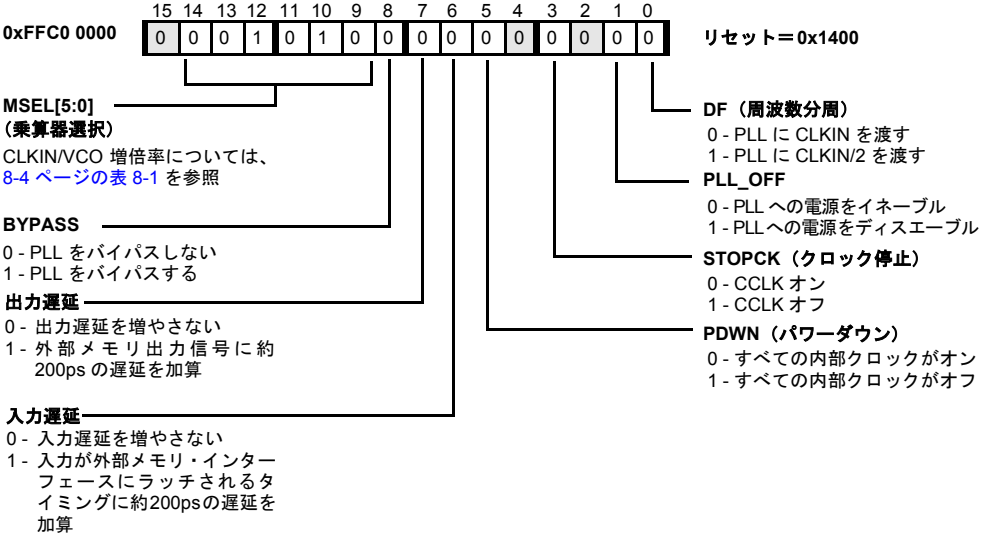


図 8-3.　PLL コントロール・レジスタ

PLL_CTL レジスタの以下のフィールドは、PLLの制御に使用されます。

- MSEL[5:0] — 乗算器選択（MSEL）フィールドでは、入力クロックからVCOクロック（CLKINからVCO）への乗算器を定義します。
- BYPASS — このビットはPLLのバイパスに使用されます。BYPASSがセットされると、コア・クロックとペリフェラル・クロックにCLKINが直接渡されます。
- 出力遅延 — このビットは、外部メモリ出力信号に約200psの遅延を加算するために使用されます。このビットをセットすべきかどうか判断するには、『ADSP-BF531/ADSP-BF532/ADSP-BF533 Embedded Processor Data Sheet』を参照してください。
- 入力遅延 — このビットは、入力が外部メモリ・インターフェースにラッチされるタイミングに約200psの遅延を加算するために使用されます。このビットをセットすべきかどうか判断するには、『ADSP-BF531/ADSP-BF532/ADSP-BF533 Embedded Processor Data Sheet』を参照してください。
- PDWN — パワーダウン（PDWN）ビットは、プロセッサをディープ・スリープ動作モードにするために使用されます。

  動作モードの詳細については、8-13ページの「動作モード」を参照してください。
- STOPCK — クロック停止（STOPCK）ビットは、コア・クロックCCLKをイネーブル／ディスエーブルにするために使用されます。
- PLL_OFF — このビットは、PLLへの電源をイネーブル／ディスエーブルにするために使用されます。
- DF — 周波数分周（DF）ビットは、PLLに対してCLKINを直接渡すか、それともCLKIN/2を渡すかを決定します。

▶ PLL_STAT レジスタ

PLL ステータス・レジスタ（PLL_STAT）では、PLL とプロセッサの動作モードを示します（図 8-4 を参照）。動作モードの詳細については、8-13 ページの「動作モード」を参照してください。

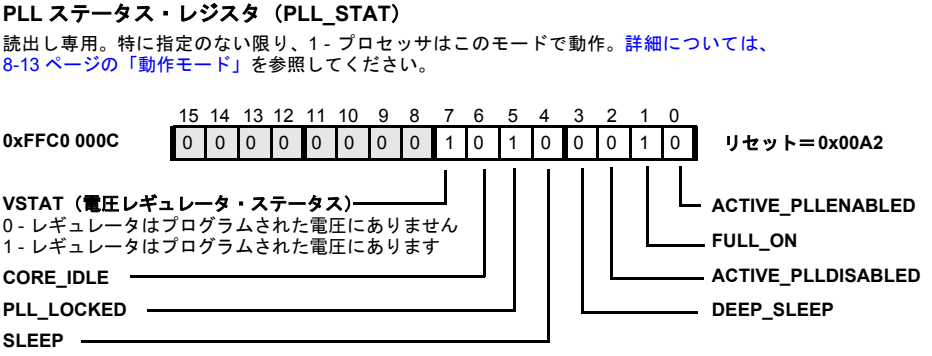


図 8-4.　PLL ステータス・レジスタ

以下のフィールドは、PLL_STAT レジスタで使用されます。

- VSTAT（電圧レギュレータ・ステータス）— このビットは、電圧レギュレータがプログラムされた電圧に達したかどうかを示します。

  電圧レベルを変更するとき、PLL を新しい電圧レベルでロックできるようにコアをアイドル動作状態にしておく必要があります。8-20 ページの「PLL プログラミング・シーケンス」を参照してください。

- CORE_IDLE — このビットは、Blackfin プロセッサ・コアがアイドル状態にされると 1 に設定されます。つまり、IDLE 命令が実行され、コアはウェイクアップ信号を待機します。

- PLL_LOCKED — 内部PLLロック・カウンタがPLLロック・カウント・レジスタ（PLL_LOCKCNT）で設定された値までインクリメントされると、このフィールドは1に設定されます。詳細については、8-11ページの「PLL_LOCKCNTレジスタ」を参照してください。
- SLEEP — プロセッサがスリープ動作モードにあるとき、このフィールドは1に設定されます。
- DEEP_SLEEP — プロセッサがディープ・スリープ動作モードにあるとき、このフィールドは1に設定されます。
- ACTIVE_PLLDISABLED — PLLをパワーダウンしてプロセッサがアクティブ動作モードにあるとき、このフィールドは1に設定されます。
- FULL_ON — プロセッサがフル・オン動作モードにあるとき、このフィールドは1に設定されます。
- ACTIVE_PLLENABLED — PLLをパワーアップしてプロセッサがアクティブ動作モードにあるとき、このフィールドは1に設定されます。

▶ PLL_LOCKCNT レジスタ

PLLのクロック周波数を変更するとき、PLLが安定して新しい周波数にロックされるには時間が必要です。

PLLロック・カウント・レジスタ（PLL_LOCKCNT）は、プロセッサがPLL_STATレジスタのPLL_LOCKEDビットを設定するまでに発生するSCLKサイクル数を定義します。PLLプログラミング・シーケンスを実行するとき、IDLE命令の実行と同時に内部PLLロック・カウンタはインクリメントを開始します。ロック・カウンタは、SCLKサイクルごとに1だけインクリメントします。ロック・カウンタがPLL_LOCKCNTレジスタで定義された値までインクリメントすると、PLL_LOCKEDビットが設定されます。

PLLの安定化時間とこのレジスタの設定値については、『ADSP-BF531/ADSP-BF532/ADSP-BF533 Embedded Processor Data Sheet』を参照してください。動作モードについては、8-13ページの「動作モード」を参照してください。PLLプログラミング・シーケンスの詳細については、8-20ページの「PLLプログラミング・シーケンス」を参照してください。

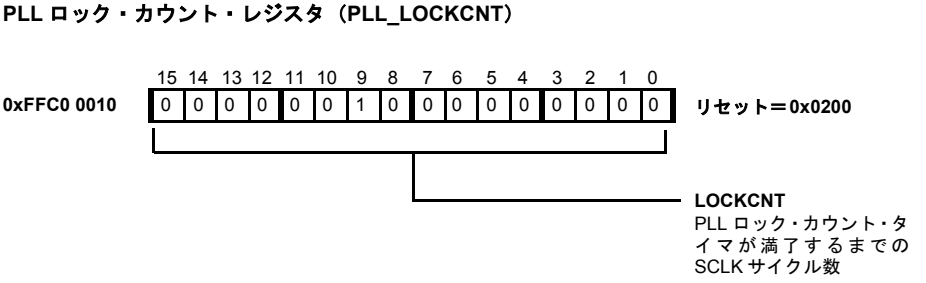


図 8-5.　PLL ロック・カウント・レジスタ

ダイナミック・パワー・マネジメント・コントローラ

ダイナミック・パワー・マネジメント・コントローラ（DPMC）はPLLと連携して動作するため、ユーザはプロセッサの性能特性と消費電力を動的に制御できます。DPMCによって提供される以下の機能を使用して、ユーザは性能と電源を制御できます。

- 複数の動作モード — プロセッサは4つの動作モードで動作し、性能特性と消費電力プロファイルは動作モードによって異なります。8-13ページの「動作モード」を参照してください。
- ペリフェラル・クロック — ペリフェラルがディスエーブルにされると、各ペリフェラルへのクロックは自動的にディスエーブルにされます。

- 電圧制御 — プロセッサに内蔵のスイッチング・レギュレータ・コントローラは、いくつかの外付け部品を利用して、外部 Vdd（V_{DDEXT}）電源から内部電圧レベルを生成できます。

  システムのニーズに応じて、電圧レベルを下げて節電を図ることができます。8-27ページの「VR_CTL レジスタ」を参照してください。

■ 動作モード

プロセッサは4つの動作モードで動作し、性能と節電の特性は各動作モードによって異なります。表 8-4に、各モードの動作特性を要約します。

表 8-4.　動作特性

動作モード	節電	PLL ステータス	PLL バイパス	CCLK	SCLK	可能な DMA アクセス
フル・オン	なし	イネーブル	いいえ	イネーブル	イネーブル	L1
アクティブ	中	イネーブル[1]	はい	イネーブル	イネーブル	L1
スリープ	高	イネーブル	いいえ	ディスエーブル	イネーブル	–
ディープ・スリープ	最高	ディスエーブル	–	ディスエーブル	ディスエーブル	–

1　このモードでは PLL もディスエーブルにできます。

■ ダイナミック・パワー・マネジメント・コントローラの状態

パワーマネジメント状態は、PLL制御状態と同義です。DPMC/PLLの状態を判定するには、PLLステータス・レジスタを読出します（8-10ページの「PLL_STAT レジスタ」を参照）。スリープとディープ・スリープ以外の全モードで、コアは命令を実行したり、アイドル・コア状態にあることができます。コアがアイドル状態にある場合には、ウェイクアップさせることができます。

アクティブ以外の全モードで、SCLK周波数はVCOに対するSSELで指定した比率によって決まります。スリープ・モードでは、コア・クロックはディスエーブルにされますが、SCLKは指定のSSEL比率で動作し続けます。

以下の節では、パワーマネジメント・コントローラの機能に関連するDPMC/PLL状態の詳細を説明します。

▶ フル・オン・モード

フル・オン・モードは、最高性能が得られるモードです。このモードでは、PLLはイネーブルにされており、バイパスされません。フル・オン・モードはプロセッサの通常の実行状態であり、プロセッサとイネーブルにされたすべてのペリフェラルは最高速度で動作します。L1メモリにはDMAアクセスが可能です。フル・オン・モードからは、8-17ページの図8-6「動作モード遷移」に示すように、プロセッサはアクティブ・モード、スリープ・モード、ディープ・スリープ・モードに直接遷移できます。

▶ アクティブ・モード

アクティブ・モードではPLLはイネーブルにされていますが、バイパスされます。PLLがバイパスされるため、プロセッサのコア・クロック（CCLK）とシステム・クロック（SCLK）は入力クロック（CLKIN）周波数で動作します。適切に設定されたL1メモリにはDMAアクセスが可能です。

アクティブ・モードでは、PLLをバイパスするだけでなく、ディスエーブルにすることも可能です。ディスエーブルにされた場合には、フル・オン・モードやスリープ・モードに遷移する前に、PLLを再びイネーブルにする必要があります。

アクティブ・モードからは、プロセッサはフル・オン・モード、スリープ・モード、ディープ・スリープ・モードに直接遷移できます。

▶ スリープ・モード

スリープ・モードでは、コア・プロセッサをアイドル状態にすることで消費電力を大幅に低下します。このモードではCCLKはディスエーブルにされます。しかしSCLKは、MSELとSSELのビット設定に基づく速度で動作を続行します。CCLKはディスエーブルにされるため、DMAアクセスはスリープ・モードで外部メモリに対してだけ可能です。スリープ・モードからは、ウェイクアップ・イベントによって、プロセッサは次のいずれかのモードに遷移します。

- アクティブ・モード：PLL_CTLレジスタのBYPASSビットがセットされている場合
- フル・オン・モード：BYPASSビットがクリアされている場合

▶ ディープ・スリープ・モード

ディープ・スリープ・モードでは、PLL、CCLK、SCLKをディスエーブルにして最大の節電を図ります。このモードでは、プロセッサ・コアとすべてのペリフェラル（リアルタイム・クロック（RTC）を除く）がディスエーブルにされます。このモードではDMAはサポートされません。

ディープ・スリープ・モードは、RTC割込みまたはハードウェア・リセット・イベントによってのみ終了できます。RTC割込みでは、プロセッサはアクティブ・モードに遷移します。ハードウェア・リセットでは、ハードウェア・リセット・シーケンスが開始されます。ハードウェア・リセットの詳細については、3-14ページの「ハードウェア・リセット」を参照してください。

なお、ディープ・スリープ・モードでのRTC割込みによって、PLLコントロール・レジスタ（PLL_CTL）のいくつかのフィールドは自動的にリセットされます。表 8-5を参照してください。

ディープ・スリープ動作モードでは、SDRAMへのクロッキングはオフにされます。ディープ・スリープ・モードに入る前に、ソフトウェアによって、SDRAM内の重要な情報を不揮発性メモリに保存してください。

表 8-5.　RTC ウェイクアップ割込み後のコントロール・レジスタ値

フィールド	値
PLL_OFF	0
STOPCK	0
PDWN	0
BYPASS	1

▶ 休止状態

最低の消費電力を実現するために、この状態では内部電源（V_{DDINT}）をオフにしたままI/O電源（V_{DDEXT}）をオンに保持することができます。これは厳密には、上述の4つのモードのような動作モードではありませんが、動作モードと見なせば図 8-6での説明に役立ちます。この機能は内蔵のスイッチング・レギュレータ・コントローラとペアになっているため、8-31ページの「コアのパワーダウン（休止状態）」で詳しく説明します。

■ 動作モード遷移

8-17ページの図8-6「動作モード遷移」は、動作モードと遷移を示します。この図では、楕円は動作モードを表します。楕円間の矢印は、各モード間で可能な遷移を示します。

各遷移矢印の隣にあるテキストは、遷移が生じるために変更する必要のあるPLLコントロール・レジスタ（PLL_CTL）内のフィールドを示します。たとえば、フル・オン・モードからスリープ・モードへの遷移では、STOPCK

ビットを1に設定し、PDWNビットを0に設定する必要があることを示します。モード遷移を生じさせる方法については、8-20ページの「動作モード遷移のプログラミング」を参照してください。

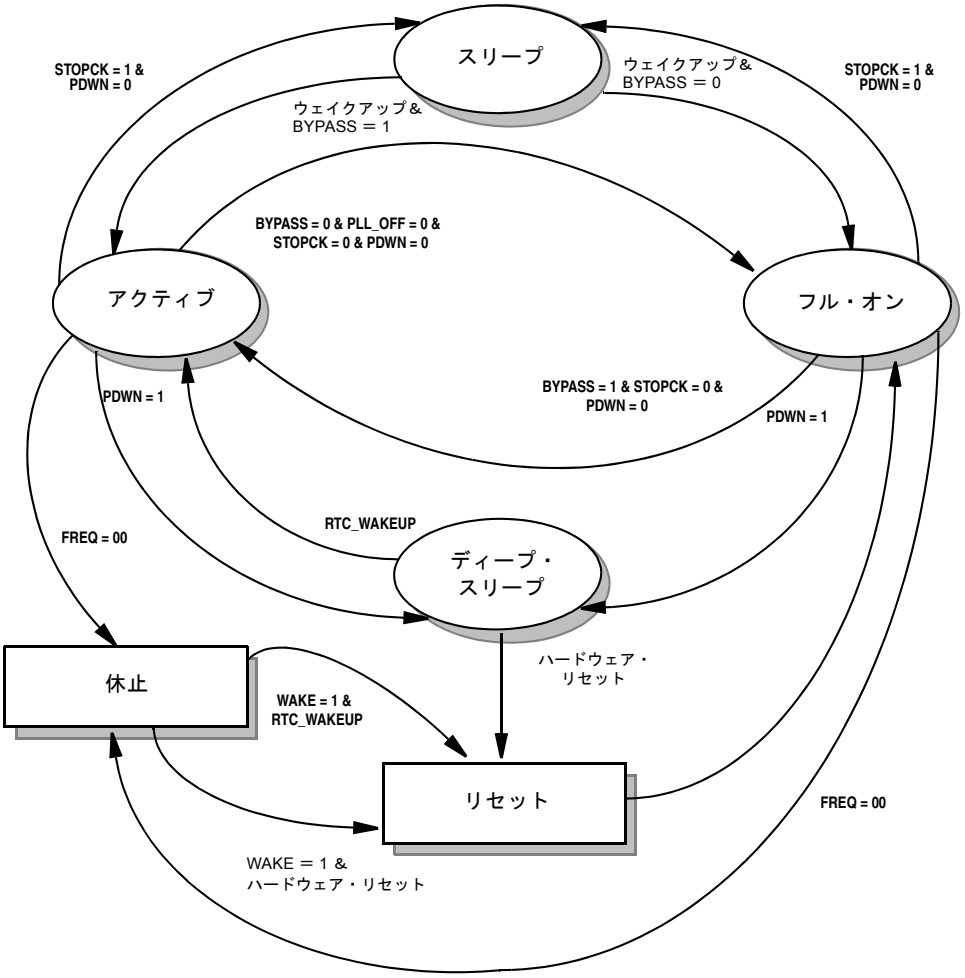


図 8-6.　動作モード遷移

図 8-6に示すモード遷移に加えて、PLLはアクティブ動作モードにおいても変更できます。つまり、PLLへの電源を印加したり除去したりできます。また、新しいクロックイン対VCOクロック（CLKIN対VCO）逓倍率をプログラムできます。後述しますが、PLLに対するこれらの変更はすぐには有効になりません。動作モード遷移と同様に、これらの変更を有効にするにはPLLプログラミング・シーケンスを実行する必要があります（8-20ページの「PLLプログラミング・シーケンス」を参照）。

- PLLディスエーブル：アクティブ・モードでバイパスされるだけでなく、PLLへの電源も除去できます。

  PLLから電源が除去されると、比較的わずかではありますが、節電になります。PLLへの電源を除去するには、PLL_CTLレジスタのPLL_OFFビットをセットしてから、PLLプログラミング・シーケンスを実行します。

- PLL イネーブル：PLL がパワーダウンされている場合、後で実行が必要なときに、再び電源を印加できます。

  PLLへの電源は、フル・オン動作モードやスリープ動作モードに遷移する前に再び印加する必要があります。PLLに電源を印加するには、PLL_CTLレジスタのPLL_OFFビットをクリアしてから、PLLプログラミング・シーケンスを実行します。

- アクティブ・モードでの新しい逓倍率：アクティブ・モードでは、新しいクロックイン対VCOクロック（CLKIN対VCO）逓倍率をプログラムできます。

  CLKIN対VCO乗算器の変更はアクティブ・モードでは実現しませんが、フル・オン・モードに遷移する前にアクティブ・モードでPLLを新しい比率にロックすれば、遷移時間が減少します。これは、PLLがすでに新しい比率にロックされているからです。なお、新しい比率にロックするには、PLLをパワーアップしておく必要があり

ます。新しいCLKIN対VCO乗算器をプログラムするには、PLL_CTLレジスタに新しいMSEL[5:0]値またはDF値、あるいはその両方を書き込んでから、PLLプログラミング・シーケンスを実行します。

- フル・オン・モードでの新しい逓倍率：フル・オン・モードでは、逓倍率も変更できます。

  この場合、PLLがロックしている間に、PLL状態は自動的にアクティブ・モードに遷移します。ロッキングの後で、PLLはフル・オン状態に戻ります。新しいCLKIN対VCO乗算器をプログラムするには、PLL_CTLレジスタに新しいMSEL[5:0]値またはDF値、あるいはその両方を書き込んでから、PLLプログラミング・シーケンスを実行します（8-20ページを参照）。

表 8-6に、可能な動作モード遷移を要約します。

表 8-6に示す以外のモード遷移を試みると、予測できない動作を招きます。

表 8-6.　可能な動作モード遷移

新しいモード	現在のモード			
	フル・オン	アクティブ	スリープ	ディープ・スリープ
フル・オン	–	可能	可能	–
アクティブ	可能	–	可能	可能
スリープ	可能	可能	–	–
ディープ・スリープ	可能	可能	–	–

▶ 動作モード遷移のプログラミング

動作モードは、PLLコントロール・レジスタ（PLL_CTL）のビットPLL_OFF、BYPASS、STOPCK、PDWNの状態によって定義されます。しかし、PLL_CTLレジスタのビットを変更するだけでは動作モードやPLLの動作は変更されません。PLL_CTLレジスタへの変更が実現するのは、リスト8-1に示す特定のコード・シーケンスを実行した後です。このコード・シーケンスは、まず、プロセッサを既知のアイドル状態にします。アイドル状態になると、PLLはPLL_CTLレジスタへの変更を認識して実装します。変更が有効になった後、プロセッサは新しい動作モードを含む（プログラムされた場合）新しい設定で動作します。

PLL プログラミング・シーケンス

PLLコントロール・レジスタ（PLL_CTL）のMSELまたはDFに新しい値が割り当てられた場合には、リスト8-1に示す命令シーケンスによってそれらの変更が有効になります。PLLプログラミング・シーケンスは、動作状態間を遷移するときにも実行されます。

分周器比率ビットCSELとSSELに対する変更は、動的に行うことができます。この変更には、PLLプログラミング・シーケンスの実行は不要です。

リスト 8-1. PLL プログラミング・シーケンス

```
CLI R0 ;    /* 割込みをディスエーブル */
IDLE ;    /* パイプラインをフラッシュし、コアをアイドル状態にします */
STI R0 ;    /* ウェイクアップ後、割込みを再びイネーブル */
```

シーケンス内の最初の2つの命令は、割込みをディスエーブルにしてコアをアイドル状態にします。割込みマスク（IMASK）はR0レジスタに保存され、命令パイプラインは停止します。その後、PLLステート・マシンはPLL_CTLレジスタの変更をPLLにロードします。

PLL_CTL レジスタの変更に新しいCLKIN対VCO乗算器が含まれる場合、または変更によってPLLに電源が再印加される場合には、PLLの再ロックが必要です。再ロックするには、まずPLLロック・カウンタがクリアされてから、SCLKサイクルごとに1回のインクリメントを開始します。PLLロック・カウンタがPLLロック・カウント・レジスタ（PLL_LOCKCNT）にプログラムされた値に到達した後、PLL は PLL ステータス・レジスタ（PLL_STAT）のPLL_LOCKEDビットをセットし、PLLはPLLウェイクアップ割込みをアサートします。

PLL_CTL レジスタの設定に応じて、プロセッサは次のいずれかの動作を行います。

- PLL_CTL レジスタがアクティブ動作モードまたはフル・オン動作モードに入るようプログラムされている場合には、8-22ページの「PLLプログラミング・シーケンスの続行」で説明するように、PLLがウェイクアップ信号を生成してから、プロセッサはシーケンス内のSTI命令に進みます。

  状態変更によって、アクティブ・モードからフル・オン・モード、またはフル・オンからアクティブに入ると、PLL自身が生成するウェイクアップ信号を使用して、コアのアイドル状態を終了することができます。このウェイクアップ信号は、PLL自身、または別のペリフェラル、ウォッチドッグなどのタイマ、RTC、およびその他のソースから生成されます。プロセッサをアイドル状態からウェイクアップさせるイベントの詳細については、4-27 ページの「SIC_IWR レジスタ」を参照してください。

- PLL_CTL レジスタがスリープ動作モードに入るようプログラムされている場合には、プロセッサはすぐにスリープ・モードに遷移して、ウェイクアップ信号を待機してから続行します。

ウェイクアップ信号がアサートされると、「PLLプログラミング・シーケンスの続行」の節で説明するように、命令シーケンスはSTI命令に進み、プロセッサは次のように遷移します。

- PLL_CTL レジスタの BYPASS ビットがセットされている場合はアクティブ・モード
- BYPASSビットがクリアされている場合はフル・オン・モード

- PLL_CTLレジスタがディープ・スリープ動作モードに入るようプログラムされている場合には、プロセッサはすぐにディープ・スリープ・モードに遷移し、RTC割込みまたはハードウェア・リセット信号を待機します。
  - 後述するように、RTC割込みによってプロセッサはアクティブ動作モードに入り、シーケンス内のSTI命令に進みます。
  - 3-14ページの「ハードウェア・リセット」で説明するように、ハードウェア・リセットによってプロセッサはリセット・シーケンスを実行します。
- 動作モード遷移がプログラムされていない場合には、次の節で説明するように、PLLはウェイクアップ信号を生成し、プロセッサはシーケンス内のSTI命令に進みます。

PLL プログラミング・シーケンスの続行

続いて、8-20ページのリスト8-1に示す命令シーケンスは、STI命令に進みます。割込みが再びイネーブルにされ、IMASKが復元され、通常のプログラム・フローが再開されます。

スプリアス動作を防止するには、この命令シーケンスの実行中にDMAを中断してください。

例

以下のコード例は、さまざまな動作モード遷移を達成する方法を示します。分かりやすくするため、いくつかのセットアップ・コードは除去されており、次のように仮定しています。

- P0はPLLコントロール・レジスタ（PLL_CTL）をポイントします。P1はPLL分周レジスタをポイントします。
- PLLウェイクアップ割込みは、ウェイクアップ信号としてイネーブルにされます。
- PLL_CTL のMSEL[5:0]とDFは、それぞれ（b#011111）と（b#0）に設定され、31xというCLKIN対VCO乗算器を表します。

アクティブ・モードからフル・オン・モード

リスト8-2に、アクティブ動作モードからフル・オン・モードに遷移するためのコードを示します。

リスト 8-2. アクティブ・モードからフル・オン・モードへの遷移

```
CLI R2;    /* 割込みをディスエーブルにし、IMASKをR2にコピー */
R1.L = 0x3E00;    /* BYPASSビットをクリア */
W[P0] = R1;    /* PLL_CTLに書き込む */


IDLE;    /* パイプラインをフラッシュし、アイドル状態に入り、PLLのウェイクアップを待機 */
STI R2;    /* PLLのウェイクアップ後、割込みとIMASKを復元 */
...    /* これでプロセッサはフル・オン・モード */
```

フル・オン・モードからアクティブ・モード

リスト8-3に、フル・オン動作モードからアクティブ・モードに遷移するためのコードを示します。

リスト 8-3. フル・オン・モードからアクティブ・モードへの遷移

```
CLI R2;    /* 割込みをディスエーブルにし、IMASKをR2にコピー */
R1.L = 0x3F00;    /* BYPASSビットをセット */
W[P0] = R1;    /* PLL_CTLに書き込む */


IDLE;    /* パイプラインをフラッシュし、アイドル状態に入り、PLLウェイクアップを待機 */
STI R2;    /* PLLのウェイクアップ後、割込みとIMASKを復元 */
...    /* これでプロセッサはアクティブ・モード */
```

フル・オン・モードで、CLKIN 対 VCO 乗算器を 31x から 2x に変更

リスト8-4に、フル・オン動作モードでCLKIN対VCO乗算器を31xから2xに変更するためのコードを示します。

リスト 8-4. CLKIN 対 VCO 乗算器の変更

```
CLI R2;    /* 割込みをディスエーブルにし、IMASKをR2にコピー */
R1.L = 0x0400;    /* VCO乗算器を2xに変更 */
W[P0] = R1;    /* PLL_CTLに書き込む */

IDLE;    /* パイプラインをフラッシュし、アイドル状態に入り、PLLのウェイクアップを待機 */
STI R2;    /* PLLのウェイクアップ後、割込みとIMASKを復元 */
```

```
...    /* プロセッサはフル・オン・モードになり、CLKIN対VCO乗算器は2xに設定
済み */
```

■ 電源電圧の動的制御

クロック周波数制御に加えて、プロセッサはコア・プロセッサを異なる電圧レベルで動作させる機能も提供します。消費電力は電圧の2乗に比例するため、低い電圧を使用すれば、消費電力の大幅な低減を達成できます。

プロセッサは3つのパワー・ドメインを使用します。これらのパワー・ドメインを表 8-7に示します。各パワー・ドメインには、別個のV_{DD}電源があります。なお、プロセッサの内部ロジックとプロセッサI/Oの多くは、ある範囲の電圧で動作させることができます。各パワー・ドメインで許容される電圧範囲と消費電力データの詳細については、『ADSP-BF531/ADSP-BF532/ADSP-BF533 Embedded Processor Data Sheet』を参照してください。

表 8-7.　パワー・ドメイン

パワー・ドメイン	V_{DD} 範囲
RTC 以外のすべての内部ロジック	可変
リアルタイム・クロック I/O と内部ロジック	可変
その他すべての I/O	可変

■ 電源管理

プロセッサに内蔵のスイッチング・レギュレータ・コントローラは、いくつかの外付けハードウェアを使用して、外付けパワー・トランジスタを備えた外部V_{DDEXT}電源から内部電圧レベルを生成できます（図 8-7 を参照）。システムのニーズに応じてこの電圧レベルを減らすことで、節電を図れます。

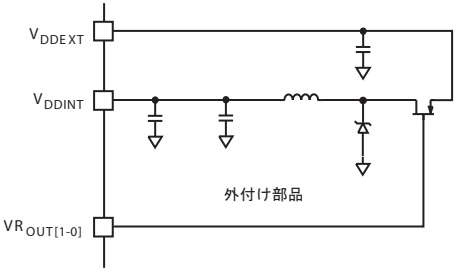


図 8-7.　プロセッサの電圧レギュレータ

V_{DDINT} 電圧を上げると、外付け FET のスイッチ・オン時間が長くなります。電源電圧を下げずに十分な電流を提供するには、V_{DDEXT} 電源に適切な容量性バイパスを設ける必要があります。

▶ VR_CTL レジスタ

内蔵のコア電圧レギュレータ・コントローラは、V_{DDINT}電源の内部ロジック電圧レベルを管理します。電圧レギュレータ・コントロール・レジスタ（VR_CTL）は、レギュレータを制御します（図 8-8を参照）。VR_CTLへの書込みによって、PLL再ロック・シーケンスが開始されます。

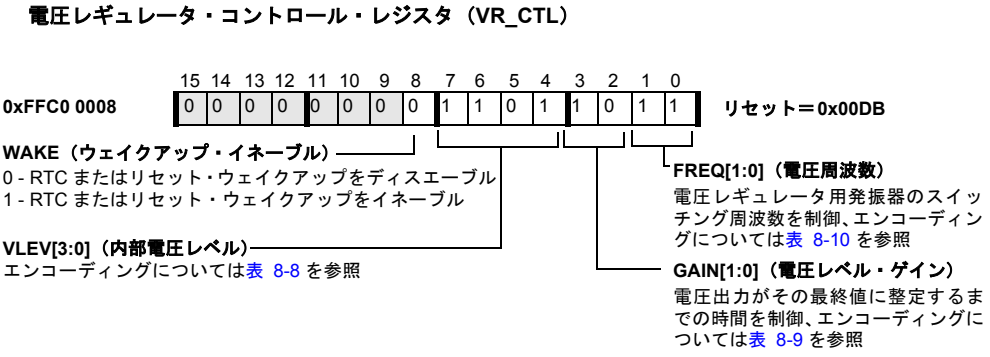


図 8-8.　電圧レギュレータ・コントロール・レジスタ

VR_CTL レジスタの以下のフィールドは、内部ロジック電圧レベルの制御に使用されます。

- WAKE ― ウェイクアップ・イネーブル（WAKE）制御ビットを使用すれば、RTCからの割込み時またはRESET# ピンでのローになるエッジで、電圧レギュレータをパワーダウン状態（FREQ=00）からウェイクアップさせることができます。
- VLEV[3:0] ― 電圧レベル（VLEV）フィールドでは、公称内部電圧レベルを識別します。適切なVLEV電圧範囲および関連する電圧許容偏差については、『ADSP-BF531/ADSP-BF532/ADSP-BF533 Embedded Processor Data Sheet』を参照してください。
- FREQ[1:0] ― 周波数（FREQ）フィールドでは、電圧レギュレータ用発振器のスイッチング周波数を制御します。周波数設定を高くすると、スイッチング・コンデンサとインダクタの値を小さくできますが、より大きなEMI（電磁干渉）が生じる可能性があります。

オンボード・レギュレーションをバイパスするには、FREQ フィールドにb#00 の値をプログラムし、VROUT ピンをフローティングのままにしておきます。

- GAIN[1:0] ― ゲイン（GAIN）フィールドでは、スイッチング・レギュレータ・ループの内部ループ・ゲインを制御します。このビットでは、電圧出力がその最終値に整定するまでの時間を制御します。一般に、ゲインを高くするとセトリング・タイムは短縮されますが、プロセスでのオーバーシュートは大きくなります。

表 8-8は、VLEV[3:0]の電圧レベル値を示します。

表 8-8.　VLEV のエンコーディング

電圧	予備
0000–0101	予備
0110	.85 ボルト
0111	.90 ボルト
1000	.95 ボルト
1001	1.00 ボルト
1010	1.05 ボルト
1011	1.10 ボルト
1100	1.15 ボルト
1101	1.20 ボルト
1110	1.25 ボルト
1111	1.30 ボルト

表 8-9は、FREQ[1:0]によって設定されるスイッチング周波数値を示します。

表 8-9.　FREQ のエンコーディング

FREQ	値
00	オンボード・レギュレーションをパワーダウン／バイパス
01	333kHz
10	667kHz
11	1MHz

表 8-10は、GAIN[1:0]によって設定されるゲイン・レベルを示します。

表 8-10. GAIN のエンコーディング

GAIN	値
00	5
01	10
10	20
11	50

▶ 電圧の変化

動作電圧の軽微な変化に対しては、アプリケーション・プログラムによる特別な注意や対策なしに対応できます。電圧許容偏差と許容される変化レートについては、『ADSP-BF531/ADSP-BF532/ADSP-BF533 Embedded Processor Data Sheet』を参照してください。

大幅な節電のためにプロセッサの動作電圧を減らしたり、大幅な性能向上のために動作電圧を上げたりすると、動作電圧レベルの大きな変更が必要となります。動作電圧を変更しても予測可能な動作を保証するには、プロセッサを既知の安定した状態にしてから、動作電圧を変更してください。

電圧を変更するときには、PLLプログラミング・シーケンスに従った手順をお勧めします。VR_CTLレジスタで電圧レベルを変更した後、プロセッサがアイドル状態に入ると、PLLは自動的にアクティブ・モードに入ります。その時点で電圧レベルが変化し、PLLは新しい電圧に再ロックされます。PLL_LOCKCNTが満了した後、デバイスはフル・オン状態に戻ります。電圧を変更するときには、PLL周波数だけを変更する場合よりも大きなPLL_LOCKCNT値が必要なことがあります。詳細については『ADSP-BF531/ADSP-BF532/ADSP-BF533 Embedded Processor Data Sheet』を参照してください。

電圧が新しいレベルに変更された後では、コア・クロック周波数（CCLK）などの動作パラメータが新しい動作電圧レベルに対して『ADSP-BF531/ADSP-BF532/ADSP-BF533 Embedded Processor Data Sheet』で指定された範囲内である限り、プロセッサは任意の動作モードに安全に復帰することができます。

▶ コアのパワーダウン（休止状態）

プロセッサの内部電源レギュレータを遮断するには、VR_CTL レジスタの FREQ ビットに b#00 を書き込みます。これによって、CCLK と SCLK の両方がディスエーブルにされます。さらに、内部電源電圧（V_{DDINT}）が 0V に設定されるため、プロセッサからのリーク電流がなくなります。内部電源レギュレータをウェイクアップさせるには、リアルタイム・クロック・ウェイクアップを使用するか、$\overline{\text{RESET}}$ ピンをアサートします。

V_{DDINT} が外部から供給されるように内蔵の電源コントローラがバイパスされた場合、コアをパワー・ダウンする唯一の方法は、外部の V_{DDINT} 電圧源を除去することです。

コアがパワー・ダウンされると、V_{DDINT} は 0V に設定されるため、プロセッサの内部状態は維持されません。したがって、内部的に格納された重要な情報（メモリ内容、レジスタ内容など）を不揮発性記憶装置に書き込んでから、電源を切断する必要があります。

V_{DDINT} をパワー・ダウンしても、V_{DDEXT} には影響を与えません。プロセッサにはまだ V_{DDEXT} が印加されていますが、特に指定のない限り、外部ピンは 3 ステート・レベルに維持されます。

内部電源をパワー・ダウンするには：

1. 必要ならば、SIC_IWR レジスタでウェイクアップに対してリアルタイム・クロックがイネーブルにされていることを確認します。
2. VR_CTL に書き込んで、FREQ ビットを b#00 に設定し、WAKE ビットを 1 に設定します。
3. 次のコード・シーケンスを実行します。

```
CLI R0 ;
IDLE ;
```

4. アイドル状態に達すると、V_{DDINT} は 0V に遷移します。
5. RTC やリセット割込みによってプロセッサがウェイクアップすると、PLL は再ロックされ、BMODE[1:0] のピン設定によって定義されたブート・シーケンスが有効になります。

プロセッサをウェイクアップさせる前に、V_{DDINT} が 0V への遷移を完了できなかった場合には、好ましくない結果を招くことがあります。

第9章　ダイレクト・メモリ・アクセス

このプロセッサはダイレクト・メモリ・アクセス（DMA）を使用して、メモリ空間内でデータを転送したり、メモリ空間とペリフェラルとの間でデータを転送したりします。プロセッサは、データ転送動作を指定して通常処理に復帰しますが、完全統合されたDMAコントローラは、プロセッサ動作とは独立してデータ転送を実行します。

DMAコントローラでは、何種類かのデータ転送を実行できます。

- メモリとメモリの間（MDMA）（9-50ページの「メモリDMA」）
- メモリとシリアル・ペリフェラル・インターフェース（SPI）の間（第10章「SPI互換ポート・コントローラ」）
- メモリとシリアル・ポート（SPORT）の間（第12章「シリアル・ポート・コントローラ」）
- メモリとUARTポートの間（第13章「UARTポート・コントローラ」）
- メモリとパラレル・ペリフェラル・インターフェース（PPI）の間（第11章「パラレル・ペリフェラル・インターフェース」）

システムは、メモリDMAコントローラ（MDMA）など、DMA対応の6つのペリフェラルを内蔵しています。以下の12本のDMAチャンネルによって、これらのデバイスをサポートします。

- PPI送／受信DMAコントローラ
- SPORT0受信DMAコントローラ

- SPORT0送信DMAコントローラ
- SPORT1受信DMAコントローラ
- SPORT1送信DMAコントローラ
- SPI送／受信DMAコントローラ
- UART受信DMAコントローラ
- UART送信DMAコントローラ
- MDMAストリーム1送信（デスティネーション）
- MDMAストリーム1受信（ソース）
- MDMAストリーム0送信（デスティネーション）
- MDMAストリーム0受信（ソース）

この章では、DMA動作の設定方法に加えて、すべてのDMAチャンネルに共通の機能についても説明します。特定のペリフェラル機能の詳細については、該当するペリフェラルの章を参照してください。DMA動作の性能とバス調停については、7-9ページの「DAB、DCB、DEBの性能」を参照してください。

プロセッサでのDMA転送は、ディスクリプタ・ベースまたはレジスタ・ベースとすることができます。ディスクリプタ・ベースのDMA転送では、DMAシーケンスを開始するために一連のパラメータをメモリ内に格納する必要があります。この種の転送では、複数のDMAシーケンスを連結できます。ディスクリプタ・ベースのDMA動作では、現在のシーケンスが完了した後で別のDMA転送を自動的に設定して開始するようにDMAチャンネルをプログラムできます。レジスタ・ベースのDMAでは、プロセッサはDMA転送の開始をDMAコントロール・レジスタに直接プログラムできます。完了時に必要ならば、コントロール・レジスタを元の設定値で自動的に更新して連続転送を実現できます。

DMA レジスタとメモリ DMA レジスタ

便宜上、この章の説明ではDMAレジスタとメモリDMAレジスタの総称（ペリフェラル固有ではない）名を使用します。

- DMAレジスタの総称名を表 9-1に示します。
- メモリDMAレジスタの総称名を9-7ページの表9-3に示します。

DMAレジスタは3つのカテゴリに分類されます。

- パラメータ・レジスタ（DMAx_CONFIG、DMAx_X_COUNTなど）

  ディスクリプタ・エレメントから直接ロードできるのは、パラメータ・レジスタだけです。ディスクリプタ・エレメントについては、9-6ページの表9-2「命名規則 : DMA MMRとディスクリプタ・エレメント」を参照してください。

DMA*x*の文字*x*は、DMA対応の特定のペリフェラルを表します。たとえば、デフォルトのチャンネル・マッピングを持つDMAの場合、DMA6_CONFIGは、UART *RX*ペリフェラルのDMA_CONFIGレジスタを表します。デフォルトのDMAチャンネル・マッピングについては、9-31ページの表9-16を参照してください。

- カレント・レジスタ（DMAx_CURR_ADDR、DMAx_CURR_X_COUNTなど）
- コントロール／ステータス・レジスタ（DMAx_IRQ_STATUS、DMAx_PERIPHERAL_MAPなど）

表 9-1にはDMAレジスタの総称名を示します。この表にはレジスタごとに、MMRオフセット、レジスタの簡単な説明、レジスタ種類、リセット値も示します。

表 9-1.　DMA メモリマップド・レジスタの総称名

MMR オフセット	MMR 総称名	MMR 説明	レジスタ種類
0x00	NEXT_DESC_PTR	次のディスクリプタへのリンク・ポインタ	パラメータ
0x04	START_ADDR	カレント・バッファの開始アドレス	パラメータ
0x08	DMA_CONFIG	イネーブル・ビットを含む、DMA 設定レジスタ	パラメータ
0x0C	予備	予備	
0x10	X_COUNT	内側ループ・カウント	パラメータ
0x14	X_MODIFY	内側ループのアドレス・インクリメント、バイト単位	パラメータ
0x18	Y_COUNT	外側ループ・カウント（2D のみ）	パラメータ
0x1C	Y_MODIFY	外側ループのアドレス・インクリメント、バイト単位	パラメータ
0x20	CURR_DESC_PTR	現在のディスクリプタ・ポインタ	カレント
0x24	CURR_ADDR	現在の DMA アドレス	カレント
0x28	IRQ_STATUS	割込みステータス・レジスタ：完了ステータス、DMA エラー割込みステータス、チャンネル状態（実行／フェッチ／休止）が含まれます	コントロール／ステータス
0x2C	PERIPHERAL_MAP	ペリフェラルと DMA チャンネルのマッピング：この DMA チャンネルに関連付けるペリフェラルを指定する 4 ビット値が含まれています（MDMA チャンネルでは読出し専用）	コントロール／ステータス
0x30	CURR_X_COUNT	現在のカウント（1D）または行内 X カウント（2D）、X_COUNT からカウント・ダウン	カレント
0x34	予備	予備	
0x38	CURR_Y_COUNT	現在の行カウント（2D のみ）、Y_COUNT からカウント・ダウン	カレント
0x3C	予備	予備	

すべてのDMAレジスタは、16ビット・エンティティとしてアクセスできます。しかし、以下のレジスタは、32ビット・レジスタとしてもアクセスできます。

- NEXT_DESC_PTR
- START_ADDR
- CURR_DESC_PTR
- CURR_ADDR

これら4本のレジスタが16ビット・エンティティとしてアクセスされる場合、下位16ビットにだけアクセスできます。

■ DMA MMR の命名規則

ディスクリプタ・エレメント名とDMAレジスタ総称名との間には混乱が生じる可能性があるため、この章では、表 9-2に示す命名規則を使用します。

- 左の列には、DMAエンジンの一般的な動作の説明に使用するMMRの総称名を示します。

なお、左の列の総称名は、実際にはプロセッサ内のリソースにマッピングされません。

- 中央の列には特定のMMR名を示します。プロセッサ・リソースには特定のMMR名だけがマッピングされます。

  DMA*x*の文字*x*は、DMAチャンネルの番号を表します。たとえば、DMA3_IRQ_STATUSは、DMAチャンネル#3のIRQ_STATUS MMRです。

  チャンネル番号は、デフォルトで割り当てたり、プログラムしたりできます。DMAチャンネル番号とデフォルトのペリフェラル・マッピングについては、9-31ページの表9-16を参照してください。

- 最後の列には、メモリ内の各ディスクリプタ・エレメントに割り当てられるマクロを示します。

  最後の列のマクロ名は、DMAエンジンの動作方法を明確にするためにのみ役立ちます。

表 9-2.　命名規則：DMA MMR とディスクリプタ・エレメント

MMR 総称名	特定の MMR 名 （x = DMA チャンネル番号）	メモリ内の対応するディスクリプタ・エレメントの名前
DMA_CONFIG	DMAx_CONFIG	DMACFG
NEXT_DESC_PTR	DMAx_NEXT_DESC_PTR	NDPH（上位 16 ビット）、 NDPL（下位 16 ビット）
START_ADDR	DMAx_START_ADDR	SAH（上位 16 ビット）、 SAL（下位 16 ビット）
X_COUNT	DMAx_X_COUNT	XCNT
Y_COUNT	DMAx_Y_COUNT	YCNT
X_MODIFY	DMAx_X_MODIFY	XMOD
Y_MODIFY	DMAx_Y_MODIFY	YMOD
CURR_DESC_PTR	DMAx_CURR_DESC_PTR	N/A
CURR_ADDR	DMAx_CURR_ADDR	N/A
CURR_X_COUNT	DMAx_CURR_X_COUNT	N/A
CURR_Y_COUNT	DMAx_CURR_Y_COUNT	N/A
IRQ_STATUS	DMAx_IRQ_STATUS	N/A
PERIPHERAL_MAP	DMAx_PERIPHERAL_MAP	N/A

■ メモリ DMA レジスタの命名規則

メモリDMAレジスタの名前は、他のDMAレジスタの名前とは少し異なります。メモリDMAストリームは別のチャンネルに再割当てできないのに対して、DMAに関連付けられたペリフェラルは、0～7の任意のDMAチャンネルにマッピングできます。

表 9-3は、メモリDMAレジスタの命名規則を示します。各名前では、文字*yy*は4つの値を持つことがあります。

- S0：メモリDMAソース・ストリーム0
- D0：メモリDMAデスティネーション・ストリーム0
- S1：メモリDMAソース・ストリーム1
- D1：メモリDMAデスティネーション・ストリーム1

表 9-3.　メモリ DMA レジスタの命名規則

MMR 総称名	メモリ DMA MMR 名 (yy = S0、S1、D0、または D1)	メモリ内の対応するディスクリプタ・エレメントの名前
DMA_CONFIG	MDMA_yy_CONFIG	DMACFG
NEXT_DESC_PTR	MDMA_yy_NEXT_DESC_PTR	NDPH（上位 16 ビット）、 NDPL（下位 16 ビット）
START_ADDR	MDMA_yy_START_ADDR	SAH（上位 16 ビット）、 SAL（下位 16 ビット）
X_COUNT	MDMA_yy_X_COUNT	XCNT
Y_COUNT	MDMA_yy_Y_COUNT	YCNT
X_MODIFY	MDMA_yy_X_MODIFY	XMOD
Y_MODIFY	MDMA_yy_Y_MODIFY	YMOD
CURR_DESC_PTR	MDMA_yy_CURR_DESC_PTR	N/A
CURR_ADDR	MDMA_yy_CURR_ADDR	N/A
CURR_X_COUNT	MDMA_yy_CURR_X_COUNT	N/A

表 9-3.　メモリ DMA レジスタの命名規則（続き）

MMR 総称名	メモリ DMA MMR 名 （yy = S0、S1、D0、または D1）	メモリ内の対応するディスクリプタ・エレメントの名前
CURR_Y_COUNT	MDMA_yy_CURR_Y_COUNT	N/A
IRQ_STATUS	MDMA_yy_IRQ_STATUS	N/A
PERIPHERAL_MAP	MDMA_yy_PERIPHERAL_MAP	N/A

■ DMAx_NEXT_DESC_PTR/MDMA_yy_NEXT_DESC_PTR レジスタ

ネクスト・ディスクリプタ・ポインタ・レジスタ（DMAx_NEXT_DESC_PTR/MDMA_yy_NEXT_DESC_PTR）では、現在のディスクリプタ・ブロックによって指定されたDMA動作が完了したときに、次のディスクリプタ・ブロックの先頭を探す場所を指定します。このレジスタは、スモールとラージのディスクリプタ・リスト・モードで使用されます。これらのモードでは、ディスクリプタ・フェッチの最初に32ビットのNEXT_DESC_PTRレジスタがCURR_DESC_PTRレジスタにコピーされます。続いて、CURR_DESC_PTRレジスタは、ディスクリプタ・フェッチ時にディスクリプタの各エレメントが読出されるたびにインクリメントされます。

スモールとラージのディスクリプタ・リスト・モードでは、DMA動作を開始する前にNEXT_DESC_PTRレジスタ（CURR_DESC_PTRレジスタではありません）をMMRアクセスによって直接プログラムする必要があります。

ディスクリプタ・アレイ・モードでは、ネクスト・ディスクリプタ・ポインタ・レジスタは無視され、フェッチ動作はCURR_DESC_PTRレジスタによってのみ制御されます。

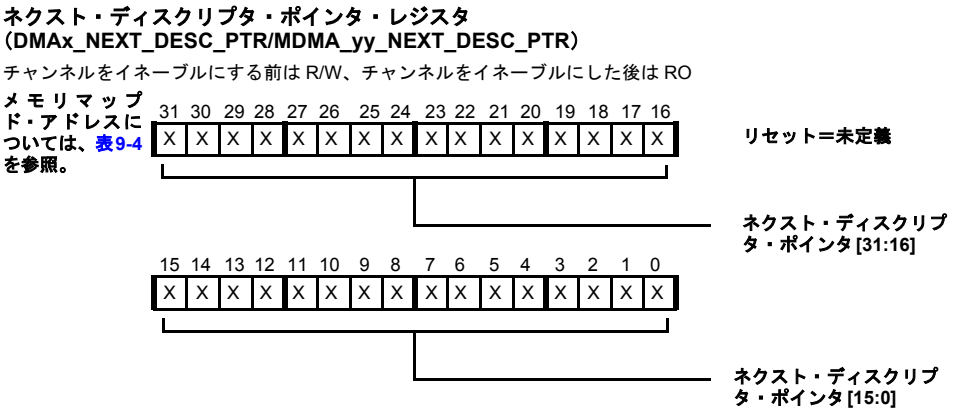


図 9-1. ネクスト・ディスクリプタ・ポインタ・レジスタ

表 9-4. ネクスト・ディスクリプタ・ポインタ・レジスタのメモリマップド・アドレス

レジスタ名	メモリマップド・アドレス
DMA0_NEXT_DESC_PTR	0xFFC0 0C00
DMA1_NEXT_DESC_PTR	0xFFC0 0C40
DMA2_NEXT_DESC_PTR	0xFFC0 0C80
DMA3_NEXT_DESC_PTR	0xFFC0 0CC0
DMA4_NEXT_DESC_PTR	0xFFC0 0D00
DMA5_NEXT_DESC_PTR	0xFFC0 0D40
DMA6_NEXT_DESC_PTR	0xFFC0 0D80
DMA7_NEXT_DESC_PTR	0xFFC0 0DC0
MDMA_D0_NEXT_DESC_PTR	0xFFC0 0E00
MDMA_S0_NEXT_DESC_PTR	0xFFC0 0E40
MDMA_D1_NEXT_DESC_PTR	0xFFC0 0E80
MDMA_S1_NEXT_DESC_PTR	0xFFC0 0EC0

■ DMAx_START_ADDR/MDMA_yy_START_ADDR レジスタ

図 9-2に示すスタート・アドレス・レジスタ（DMAx_START_ADDR/MDMA_yy_START_ADDR）には、現在DMAのターゲットになっているデータ・バッファの開始アドレスが含まれています。

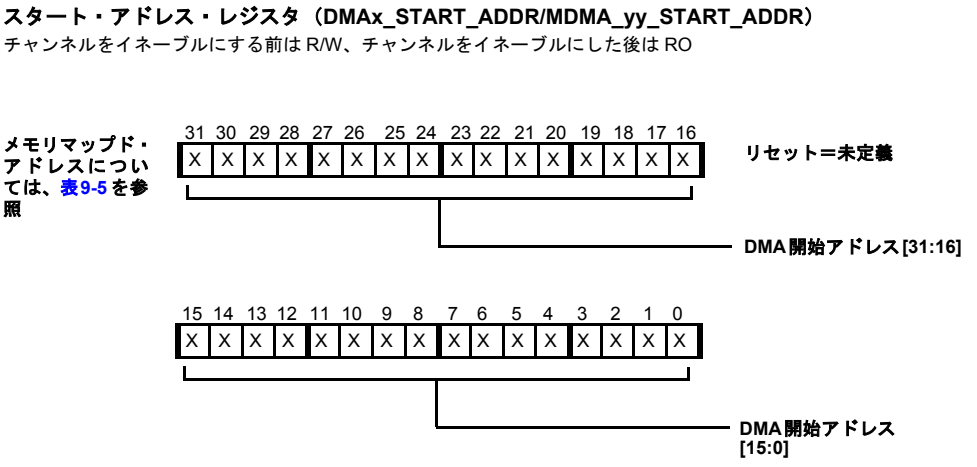


図 9-2.　スタート・アドレス・レジスタ

表 9-5.　スタート・アドレス・レジスタのメモリマップド・アドレス

レジスタ名	メモリマップド・アドレス
DMA0_START_ADDR	0xFFC0 0C04
DMA1_START_ADDR	0xFFC0 0C44
DMA2_START_ADDR	0xFFC0 0C84
DMA3_START_ADDR	0xFFC0 0CC4
DMA4_START_ADDR	0xFFC0 0D04
DMA5_START_ADDR	0xFFC0 0D44
DMA6_START_ADDR	0xFFC0 0D84
DMA7_START_ADDR	0xFFC0 0DC4
MDMA_D0_START_ADDR	0xFFC0 0E04
MDMA_S0_START_ADDR	0xFFC0 0E44
MDMA_D1_START_ADDR	0xFFC0 0E84
MDMA_S1_START_ADDR	0xFFC0 0EC4

■ DMAx_CONFIG/MDMA_yy_CONFIG レジスタ

図 9-3 に示すDMA設定レジスタ（DMAx_CONFIG/MDMA_yy_CONFIG）は、DMAのパラメータと動作モードの設定に使用されます。なお、DMAの実行中にDMA_CONFIGレジスタに書込みを行うと、DMA_ENビットを0に設定して書き込んだ場合を除いて、DMAエラーが発生します。

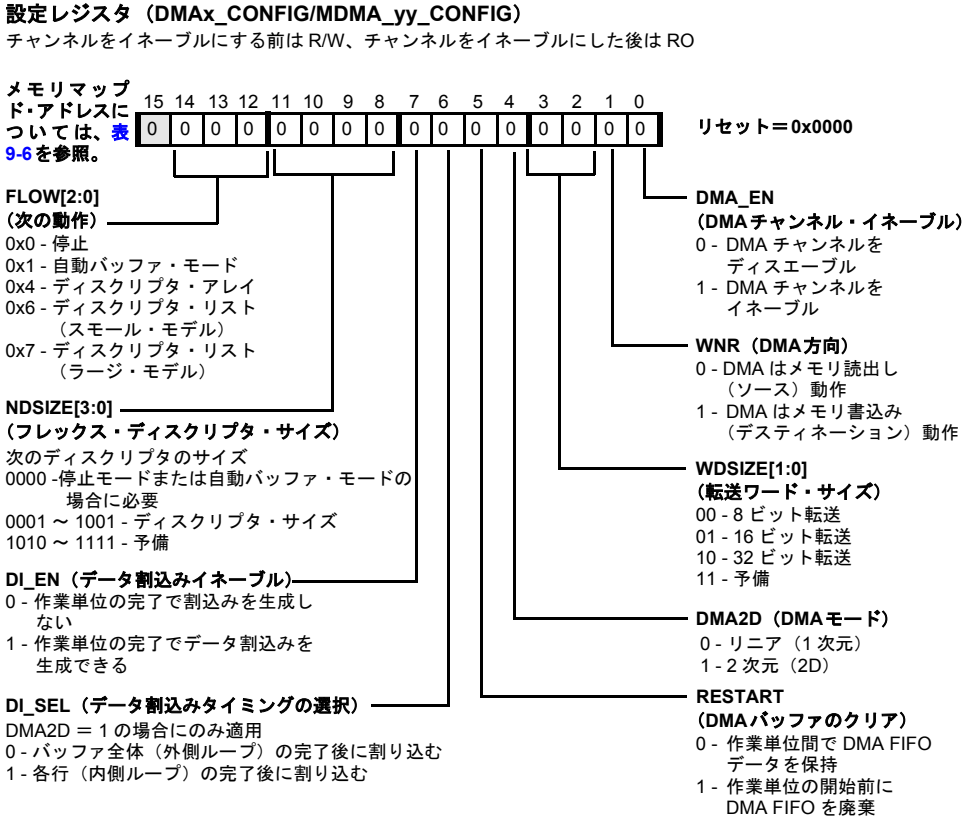


図 9-3.　設定レジスタ

表 9-6.　設定レジスタのメモリマップド・アドレス

レジスタ名	メモリマップド・アドレス
DMA0_CONFIG	0xFFC0 0C08
DMA1_CONFIG	0xFFC0 0C48
DMA2_CONFIG	0xFFC0 0C88
DMA3_CONFIG	0xFFC0 0CC8
DMA4_CONFIG	0xFFC0 0D08
DMA5_CONFIG	0xFFC0 0D48
DMA6_CONFIG	0xFFC0 0D88
DMA7_CONFIG	0xFFC0 0DC8
MDMA_D0_CONFIG	0xFFC0 0E08
MDMA_S0_CONFIG	0xFFC0 0E48
MDMA_D1_CONFIG	0xFFC0 0E88
MDMA_S1_CONFIG	0xFFC0 0EC8

DMAx_CONFIG レジスタのフィールドは、DMAのパラメータと動作モードの設定に使用されます。

- FLOW[2:0]（次の動作）。このフィールドでは、現在のDMA転送に続く、次のDMA転送のタイプを指定します。フローのオプションは次のとおりです。

- 0x0 — 停止。現在の作業単位が完了すると、割込み通知の後で（選択された場合）DMAチャンネルは自動的に停止します。DMAx_IRQ_STATUS レジスタの DMA_RUN ステータス・ビットは1から0に変化しますが、DMAx_CONFIG レジスタの DMA_EN ビットは変化しません。この状態では、チャンネルは休止します。ペリフェラル割込みは依然としてDMAユニットによって除外されます。チャンネルを再開するには、DMAx_CONFIG レジスタへの書込みによって、DMA_EN ビットを1に設定して次の作業単位を指定します。

0x1 — 自動バッファ・モード。このモードでは、メモリ内のディスクリプタは使用されません。その代わりに、ユーザがプログラムしたDMAx MMR設定に基づいて、DMAが連続循環バッファ方式で実行されます。作業単位が完了すると、パラメータ・レジスタがカレント・レジスタに再ロードされ、DMAがゼロ・オーバーヘッドですぐに再開されます。自動バッファ・モードを停止するには、DMAx_CONFIGレジスタのDMA_ENビットに0を書き込みます。

0x4 — ディスクリプタ・アレイ・モード。このモードでは、NDPHエレメントやNDPLエレメントを含まないメモリからディスクリプタをフェッチします。このディスクリプタはネクスト・ディスクリプタ・ポインタ・エントリを含まないため、DMAエンジンのデフォルトでは、CURR_DESC_PTRレジスタを使用してディスクリプタを1ステップずつ実行します。したがって、一群のディスクリプタがアレイのようにメモリ内で互いに続きます。

0x6 — ディスクリプタ・リスト（スモール・モデル）モード。このモードでは、NDPLを含みNDPHを含まないメモリからディスクリプタをフェッチします。したがって、ネクスト・ディスクリプタ・ポインタ・フィールドの上位16ビットは、NEXT_DESC_PTRレジスタの上位16ビットから取り出されるため、すべてのディスクリプタはメモリ内の特定の64Kページに制限されます。

0x7 — ディスクリプタ・リスト（ラージ・モデル）モード。このモードでは、NDPHとNDPLを含むメモリからディスクリプタをフェッチします。したがって、メモリ内でのディスクリプタ配置に最大の柔軟性が与えられます。

- NDSIZE[3:0]（フレックス・ディスクリプタ・サイズ）。このフィールドでは、ロードすべきメモリ内のディスクリプタ・エレメントの数を指定します。停止モードや自動バッファ・モードの場合には、

このフィールドは0である必要があります。NDSIZEとFLOWでYMODを逸脱するディスクリプタを指定した場合には、DMAエラーが発生します。

- DI_EN（データ割込みイネーブル）。このビットでは、作業単位の完了でデータ割込みの生成を可能にするかどうかを指定します。
- DI_SEL（データ割込みタイミングの選択）。このビットでは、バッファ全体を完了した後、または内側ループの各行を完了した後でのデータ割込みのタイミングを指定します。このビットは、2D DMA動作でのみ使用されます。
- RESTART（DMAバッファのクリア）。このビットでは、次の作業単位を開始する前に、チャンネルのデータFIFOに保持された受信データを保持する（RESTART = 0）か、廃棄する（RESTART = 1）かを指定します。受信データが自動的に廃棄されるのは、DMA_ENビットが0から1に変化するときであり、通常はチャンネルが最初にイネーブルにされたときです。作業単位が連続データストリームを構成する場合には、一般に受信されたFIFOデータは作業単位間で保持されるはずです。しかし、新しい作業単位が新しいデータストリームを開始する場合には、以前に受信したデータをクリアするため、RESTARTビットを1に設定してください。

RESTARTビットは、メモリ書込みDMAチャンネルにのみ適用されます。このビットは、メモリ読出しDMAチャンネルとMDMAチャンネルの場合には留保され、0であることが必要です。

メモリ書込みDMAチャンネルでは、RESTARTビットはDMAx_CONFIGレジスタへの書込みによって開始された最初の作業単位にのみ影響を与えます。RESTARTビットは、DMAディスクリプタのDMACFGエレメントで設定されている場合には効果がありません。

- DMA2D（DMAモード）。このビットでは、DMAモードがX_COUNTとX_MODIFYだけを伴う（1次元DMA）のか、Y_COUNTとY_MODIFYも伴う（2次元DMA）のかを指定します。

- WDSIZE[1:0]（転送ワード・サイズ）。DMAエンジンでは、8/16/32ビット項目の転送が可能です。各要求／許諾によって1つのメモリ・アクセスが生じます（ただし、16ビットのメモリ・ポートや16ビットのDMAアクセス・バスを通じて32ビット・データを転送するには、2つのサイクルが必要です）。DMAアドレス・ポインタ・レジスタのインクリメント・サイズ（ストライド）は、転送ユニット・サイズの倍数（8ビットでは1、16ビットでは2、32ビットでは4）である必要があります。

- WNR（DMA方向）。このビットでは、DMA方向を指定します。つまり、メモリ読出し（0）またはメモリ書込み（1）です。

- DMA_EN（DMAチャンネル・イネーブル）。このビットでは、特定のDMAチャンネルをイネーブルにするかどうかを指定します。

ペリフェラルDMAチャンネルがイネーブルにされると、ペリフェラルからの割込みはDMA要求を示します。チャンネルがディスエーブルにされると、DMAユニットではペリフェラル割込みを無視し、それを割込みコントローラに直接渡します。予想外の結果を回避するには、DMAチャンネルをイネーブルにしてからペリフェラルをイネーブルにし、ペリフェラルをディスエーブルにしてからDMAチャンネルをディスエーブルにしてください。

■ DMAx_X_COUNT/MDMA_yy_X_COUNT レジスタ

2D DMAの場合、図 9-4に示す内側ループ・カウント・レジスタ（DMAx_X_COUNT/MDMA_yy_X_COUNT）には内側ループ・カウントが含まれています。1D DMAの場合には、読み出すエレメントの数を指定します。詳細については、9-47ページの「2次元DMA」を参照してください。X_COUNTでの値0は、65,536個のエレメントに対応します。

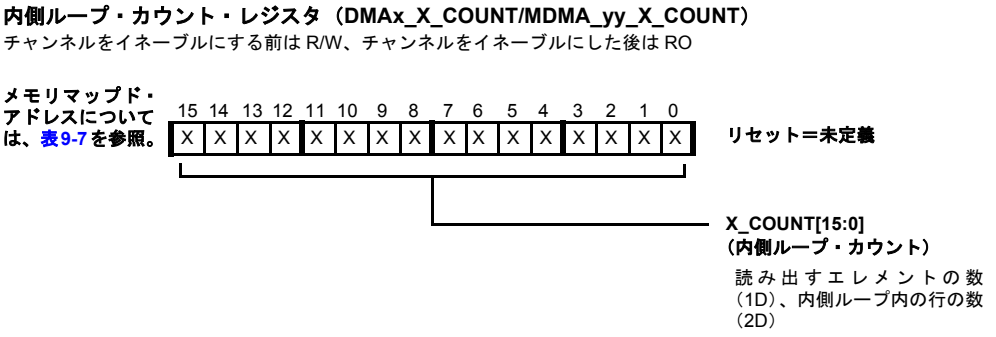


図 9-4.　内側ループ・カウント・レジスタ

表 9-7.　内側ループ・カウント・レジスタのメモリマップド・アドレス

レジスタ名	メモリマップド・アドレス
DMA0_X_COUNT	0xFFC0 0C10
DMA1_X_COUNT	0xFFC0 0C50
DMA2_X_COUNT	0xFFC0 0C90
DMA3_X_COUNT	0xFFC0 0CD0
DMA4_X_COUNT	0xFFC0 0D10
DMA5_X_COUNT	0xFFC0 0D50
DMA6_X_COUNT	0xFFC0 0D90
DMA7_X_COUNT	0xFFC0 0DD0
MDMA_D0_X_COUNT	0xFFC0 0E10
MDMA_S0_X_COUNT	0xFFC0 0E50
MDMA_D1_X_COUNT	0xFFC0 0E90
MDMA_S1_X_COUNT	0xFFC0 0ED0

■ DMAx_X_MODIFY/MDMA_yy_X_MODIFY レジスタ

内側ループ・アドレス・インクリメント・レジスタ（DMAx_X_MODIFY/MDMA_yy_X_MODIFY）には、符号付き、2の補数のバイト・アドレス・インクリメントを含んでいます。1D DMAでは、このインクリメントは、各エレメントの転送後に適用されるストライドです。

なお、X_MODIFYは、DMA転送サイズとは無関係にバイト単位で指定されます。

2D DMAでは、このインクリメントは、各内側ループでの最終エレメントまで（最終エレメント自身を含まず）内側ループ内の各エレメントを転送した後で適用されます。各内側ループの最終エレメントの後では、代わりにY_MODIFYレジスタが適用されます（ただし、各作業単位の最後の転送を除きます）。作業単位の最後の転送では、常にX_MODIFYレジスタが適用されます。

X_MODIFYフィールドは0に設定できます。この場合、DMAは同じアドレスとの間で繰返し実行されます。これが役立つのは、たとえば、データ・レジスタと外部メモリマップド・ペリフェラルとの間でデータを転送する場合です。

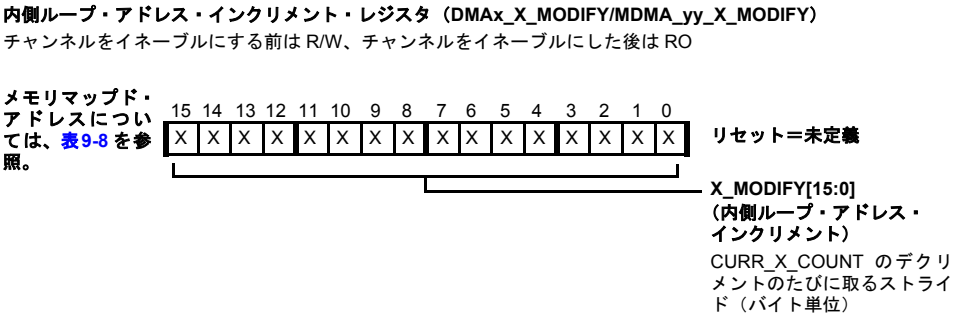


図 9-5.　内側ループ・アドレス・インクリメント・レジスタ

表 9-8.　内側ループ・アドレス・インクリメント・レジスタのメモリマップド・アドレス

レジスタ名	メモリマップド・アドレス
DMA0_X_MODIFY	0xFFC0 0C14
DMA1_X_MODIFY	0xFFC0 0C54
DMA2_X_MODIFY	0xFFC0 0C94
DMA3_X_MODIFY	0xFFC0 0CD4
DMA4_X_MODIFY	0xFFC0 0D14
DMA5_X_MODIFY	0xFFC0 0D54
DMA6_X_MODIFY	0xFFC0 0D94
DMA7_X_MODIFY	0xFFC0 0DD4
MDMA_D0_X_MODIFY	0xFFC0 0E14
MDMA_S0_X_MODIFY	0xFFC0 0E54

表 9-8.　内側ループ・アドレス・インクリメント・レジスタのメモリマップド・アドレス（続き）

レジスタ名	メモリマップド・アドレス
MDMA_D1_X_MODIFY	0xFFC0 0E94
MDMA_S1_X_MODIFY	0xFFC0 0ED4

■ DMAx_Y_COUNT/MDMA_yy_Y_COUNT レジスタ

2D DMAの場合、外側ループ・カウント・レジスタ（DMAx_Y_COUNT/MDMA_yy_Y_COUNT）には外側ループ・カウントが含まれています。これは1D DMAモードでは使用されません。このレジスタには、2D DMAシーケンスの外側ループ内の行数が含まれています。詳細については9-47ページの「2次元DMA」を参照してください。

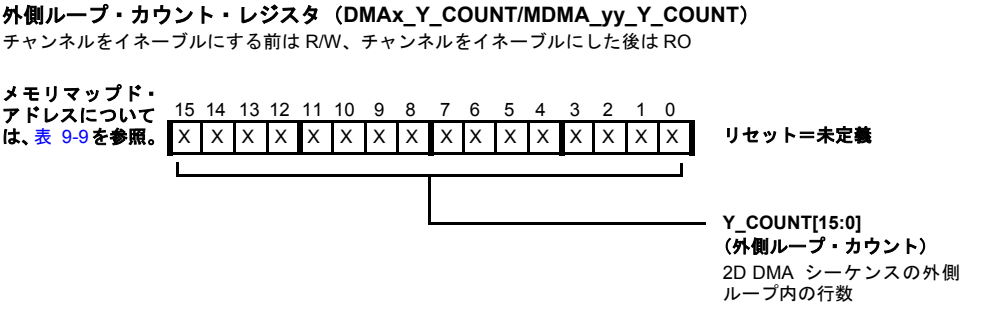


図 9-6.　外側ループ・カウント・レジスタ

表 9-9.　外側ループ・カウント・レジスタのメモリマップド・アドレス

レジスタ名	メモリマップド・アドレス
DMA0_Y_COUNT	0xFFC0 0C18
DMA1_Y_COUNT	0xFFC0 0C58
DMA2_Y_COUNT	0xFFC0 0C98
DMA3_Y_COUNT	0xFFC0 0CD8

表 9-9.　外側ループ・カウント・レジスタのメモリマップド・アドレス

レジスタ名	メモリマップド・アドレス
DMA4_Y_COUNT	0xFFC0 0D18
DMA5_Y_COUNT	0xFFC0 0D58
DMA6_Y_COUNT	0xFFC0 0D98
DMA7_Y_COUNT	0xFFC0 0DD8
MDMA_D0_Y_COUNT	0xFFC0 0E18
MDMA_S0_Y_COUNT	0xFFC0 0E58
MDMA_D1_Y_COUNT	0xFFC0 0E98
MDMA_S1_Y_COUNT	0xFFC0 0ED8

■ DMAx_Y_MODIFY/MDMA_yy_Y_MODIFY レジスタ

外側ループ・アドレス・インクリメント・レジスタ（DMAx_Y_MODIFY/MDMA_yy_Y_MODIFY）には、符号付き、2の補数値が含まれています。このバイト・アドレス・インクリメントは、CURR_Y_COUNT レジスタのデクリメントのたびに適用されます（ただし、CURR_Y_COUNT も満了する2Dアレイ内の最終項目を除きます）。この値は、1つの「行」の最終ワードと次の「行」の先頭ワードとのオフセットです。詳細については、9-47ページの「2次元DMA」を参照してください。

なお、Y_MODIFYは、DMA転送サイズとは無関係にバイト単位で指定されます。

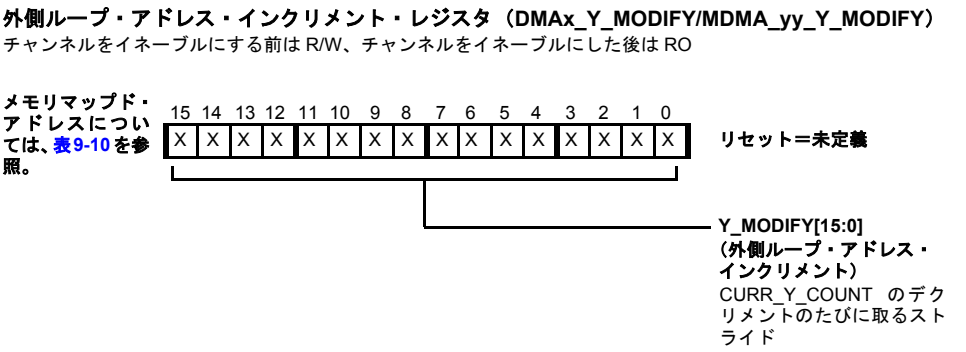


図 9-7.　外側ループ・アドレス・インクリメント・レジスタ

表 9-10.　外側ループ・アドレス・インクリメント・レジスタのメモリマップド・アドレス

レジスタ名	メモリマップド・アドレス
DMA0_Y_MODIFY	0xFFC0 0C1C
DMA1_Y_MODIFY	0xFFC0 0C5C
DMA2_Y_MODIFY	0xFFC0 0C9C
DMA3_Y_MODIFY	0xFFC0 0CDC
DMA4_Y_MODIFY	0xFFC0 0D1C
DMA5_Y_MODIFY	0xFFC0 0D5C
DMA6_Y_MODIFY	0xFFC0 0D9C
DMA7_Y_MODIFY	0xFFC0 0DDC
MDMA_D0_Y_MODIFY	0xFFC0 0E1C
MDMA_S0_Y_MODIFY	0xFFC0 0E5C
MDMA_D1_Y_MODIFY	0xFFC0 0E9C
MDMA_S1_Y_MODIFY	0xFFC0 0EDC

■ DMAx_CURR_DESC_PTR/MDMA_yy_CURR_DESC_PTR レジスタ

カレント・ディスクリプタ・ポインタ・レジスタ（DMAx_CURR_DESC_PTR/MDMA_yy_CURR_DESC_PTR）には、次にロードするディスクリプタ・エレメントのメモリ・アドレスが含まれています。ディスクリプタ（FLOW＝4、6、または7）を伴うFLOWモード設定の場合、このレジスタはDMA作業ブロックが開始される前に、ディスクリプタ・エレメントを適切なMMRに読み出すために使用されます。ディスクリプタ・リスト・モード（FLOW = 6または7）の場合、各ディスクリプタのロードの前に、このレジスタはNEXT_DESC_PTRレジスタから初期化されます。その後、各ディスクリプタ・エレメントが読み出されるたびに、CURR_DESC_PTRレジスタ内のアドレスがインクリメントされます。

ディスクリプタ全体が読み出されると、CURR_DESC_PTRレジスタには次の値が含まれます。

```
Descriptor Start Address + Descriptor Size (# of elements)
```
（ディスクリプタ開始アドレス+ディスクリプタ・サイズ（エレメントの数））

ディスクリプタ・アレイ・モード（FLOW = 4）の場合、DMA動作を開始する前に、MMRアクセスによってこのレジスタ（NEXT_DESC_PTRレジスタではありません）をプログラムする必要があります。

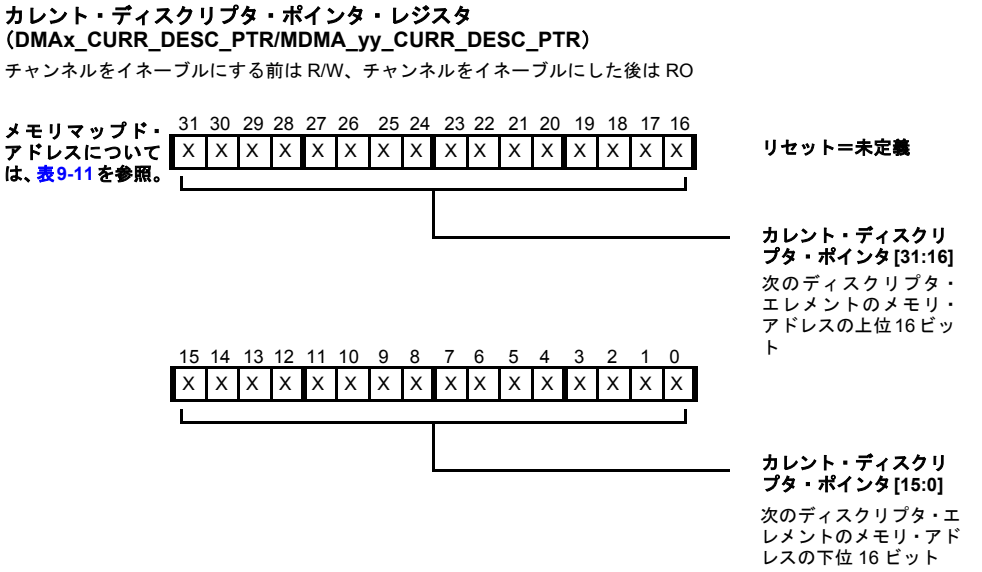


図 9-8.　カレント・ディスクリプタ・ポインタ・レジスタ

表 9-11.　カレント・ディスクリプタ・ポインタ・レジスタのメモリマップド・アドレス

レジスタ名	メモリマップド・アドレス
DMA0_CURR_DESC_PTR	0xFFC0 0C20
DMA1_CURR_DESC_PTR	0xFFC0 0C60
DMA2_CURR_DESC_PTR	0xFFC0 0CA0
DMA3_CURR_DESC_PTR	0xFFC0 0CE0
DMA4_CURR_DESC_PTR	0xFFC0 0D20
DMA5_CURR_DESC_PTR	0xFFC0 0D60
DMA6_CURR_DESC_PTR	0xFFC0 0DA0
DMA7_CURR_DESC_PTR	0xFFC0 0DE0
MDMA_D0_CURR_DESC_PTR	0xFFC0 0E20

表 9-11. カレント・ディスクリプタ・ポインタ・レジスタの メモリマップド・アドレス（続き）

レジスタ名	メモリマップド・アドレス
MDMA_S0_CURR_DESC_PTR	0xFFC0 0E60
MDMA_D1_CURR_DESC_PTR	0xFFC0 0EA0
MDMA_S1_CURR_DESC_PTR	0xFFC0 0EE0

■ DMAx_CURR_ADDR/MDMA_yy_CURR_ADDR レジスタ

図 9-9に示すカレント・アドレス・レジスタ（DMAx_CURR_ADDR/MDMA_yy_CURR_ADDR）には、特定DMAセッションの現在のDMA転送アドレスが含まれています。DMAセッションの最初に、CURR_ADDR レジスタはSTART_ADDR レジスタからロードされ、転送が行われるたびにインクリメントされます。カレント・アドレス・レジスタには32個のビットが含まれています。

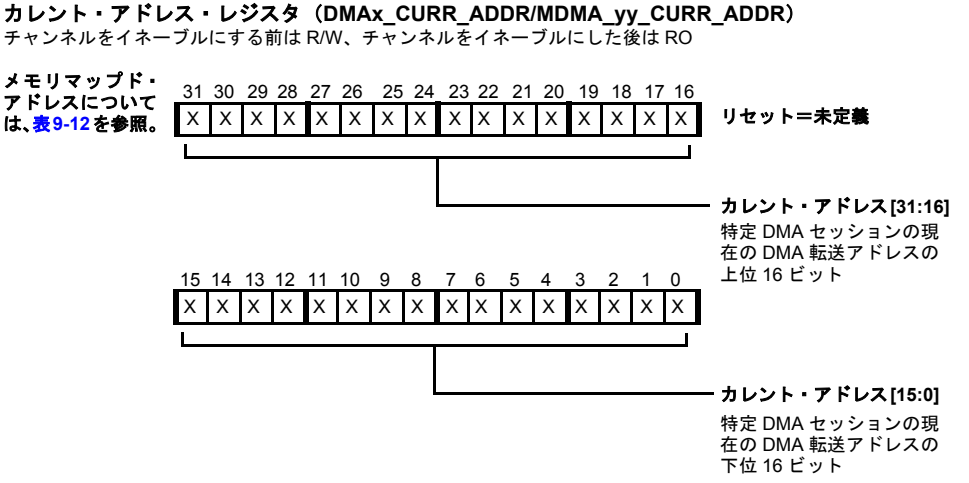


図 9-9. カレント・アドレス・レジスタ

表 9-12. カレント・アドレス・レジスタのメモリマップド・アドレス

レジスタ名	メモリマップド・アドレス
DMA0_CURR_ADDR	0xFFC0 0C24
DMA1_CURR_ADDR	0xFFC0 0C64
DMA2_CURR_ADDR	0xFFC0 0CA4
DMA3_CURR_ADDR	0xFFC0 0CE4
DMA4_CURR_ADDR	0xFFC0 0D24
DMA5_CURR_ADDR	0xFFC0 0D64
DMA6_CURR_ADDR	0xFFC0 0DA4
DMA7_CURR_ADDR	0xFFC0 0DE4
MDMA_D0_CURR_ADDR	0xFFC0 0E24
MDMA_S0_CURR_ADDR	0xFFC0 0E64
MDMA_D1_CURR_ADDR	0xFFC0 0EA4
MDMA_S1_CURR_ADDR	0xFFC0 0EE4

■ DMAx_CURR_X_COUNT/MDMA_yy_CURR_X_COUNTレジスタ

カレント内側ループ・カウント・レジスタ（DMAx_CURR_X_COUNT/MDMA_yy_CURR_X_COUNT）は、各DMAセッションの最初にX_COUNTレジスタによってロードされ（1D DMAの場合）、各行のDMAの最後にもX_COUNTレジスタによってロードされます（2D DMAの場合）。それ以外の場合には、エレメントが転送されるたびにデクリメントされます。このレジスタでのカウントの満了は、DMAの完了を示します。2D DMAでは、CURR_X_COUNTレジスタ値が0になるのは転送全体が完了したときだけです。行間ではX_COUNTレジスタの値と等しくなります。

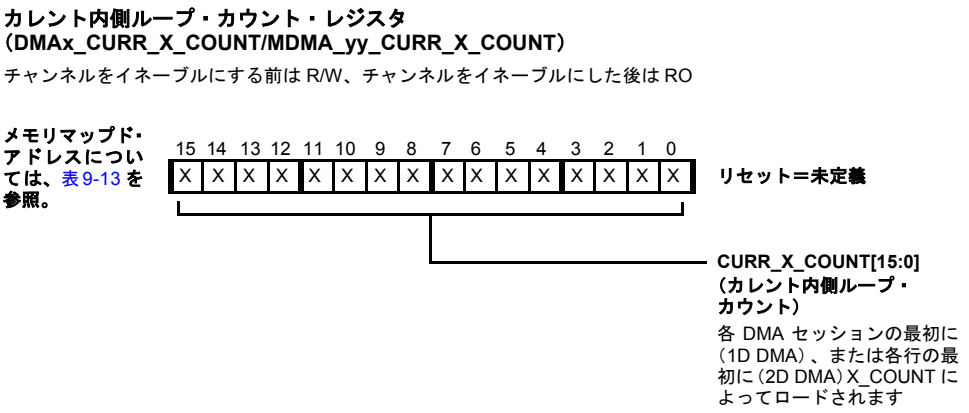


図 9-10. カレント内側ループ・カウント・レジスタ

表 9-13. カレント内側ループ・カウント・レジスタのメモリマップド・アドレス

レジスタ名	メモリマップド・アドレス
DMA0_CURR_X_COUNT	0xFFC0 0C30
DMA1_CURR_X_COUNT	0xFFC0 0C70
DMA2_CURR_X_COUNT	0xFFC0 0CB0
DMA3_CURR_X_COUNT	0xFFC0 0CF0
DMA4_CURR_X_COUNT	0xFFC0 0D30
DMA5_CURR_X_COUNT	0xFFC0 0D70
DMA6_CURR_X_COUNT	0xFFC0 0DB0
DMA7_CURR_X_COUNT	0xFFC0 0DF0
MDMA_D0_CURR_X_COUNT	0xFFC0 0E30
MDMA_S0_CURR_X_COUNT	0xFFC0 0E70
MDMA_D1_CURR_X_COUNT	0xFFC0 0EB0
MDMA_S1_CURR_X_COUNT	0xFFC0 0EF0

■ DMAx_CURR_Y_COUNT/MDMA_yy_CURR_Y_COUNT レジスタ

カレント外側ループ・カウント・レジスタ（DMAx_CURR_Y_COUNT/MDMA_yy_CURR_Y_COUNT）は、各2D DMAセッションの最初にY_COUNT レジスタによってロードされます。これは1D DMAには使用されません。2D DMA動作中にCURR_X_COUNT レジスタが満了（1からX_COUNT または1から0に遷移）するたびに、このレジスタはデクリメントされ、行全体の転送の完了を示します。2D DMAセッションが完了した後、CURR_Y_COUNT = 1、CURR_X_COUNT = 0になります。

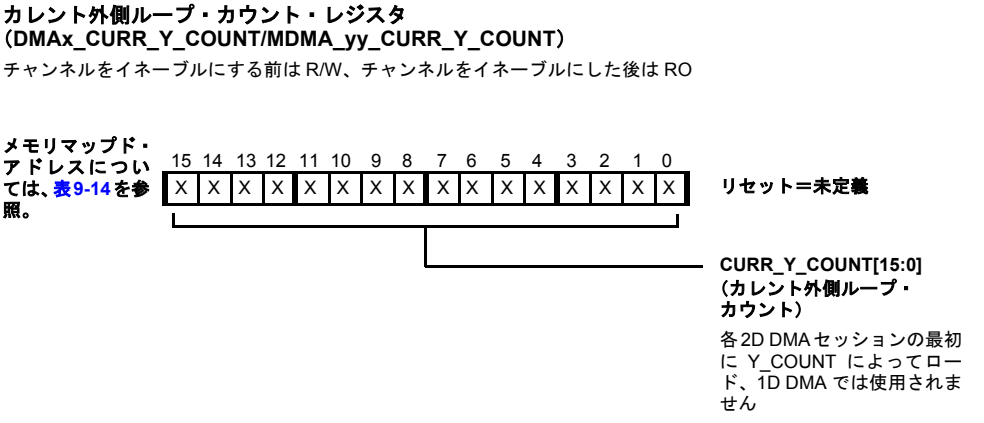


図 9-11. カレント外側ループ・カウント・レジスタ

表 9-14. カレント外側ループ・カウント・レジスタの
メモリマップド・アドレス

レジスタ名	メモリマップド・アドレス
DMA0_CURR_Y_COUNT	0xFFC0 0C38
DMA1_CURR_Y_COUNT	0xFFC0 0C78
DMA2_CURR_Y_COUNT	0xFFC0 0CB8
DMA3_CURR_Y_COUNT	0xFFC0 0CF8
DMA4_CURR_Y_COUNT	0xFFC0 0D38
DMA5_CURR_Y_COUNT	0xFFC0 0D78
DMA6_CURR_Y_COUNT	0xFFC0 0DB8
DMA7_CURR_Y_COUNT	0xFFC0 0DF8
MDMA_D0_CURR_Y_COUNT	0xFFC0 0E38
MDMA_S0_CURR_Y_COUNT	0xFFC0 0E78
MDMA_D1_CURR_Y_COUNT	0xFFC0 0EB8
MDMA_S1_CURR_Y_COUNT	0xFFC0 0EF8

■ DMAx_PERIPHERAL_MAP/MDMA_yy_PERIPHERAL_MAP レジスタ

各DMAチャンネルのペリフェラル・マップ・レジスタ（DMAx_PERIPHERAL_MAP/MDMA_yy_PERIPHERAL_MAP）には、以下のビットが含まれています。

- チャンネルを特定のペリフェラルにマッピングするビット。
- チャンネルがペリフェラルDMAチャンネルであるか、メモリDMAチャンネルであるかを識別するビット。

なお、DMAチャンネルとペリフェラルとの間には、1:1のマッピングが存在しなければなりません。ユーザには、複数のDMAチャンネルが同じペリフェラルにマッピングされないこと、および複数のペリフェラルが同じDMAポートにマッピングされないことを保証

する責任があります。複数のチャンネルが同じペリフェラルにマッピングされた場合には、1つのチャンネルだけが接続されます（最低優先順位のチャンネル）。存在しないペリフェラル（たとえば、PMAP フィールドの 0xF）がチャンネルにマッピングされた場合には、そのチャンネルはディスエーブルにされます。つまり、DMA 要求は無視され、DMA 許諾は発行されません。また、DMA 要求は、ペリフェラルから割込みコントローラに転送されません。

2つのチャンネルの DMA チャンネル優先順位をスワップするには、以下の手順に従ってください。チャンネル 6 と 7 が関係すると想定します。

1. チャンネル 6 と 7 で DMA がディスエーブルにされていることを確認します。
2. DMA6_PERIPHERAL_MAP に 0x7000 を、DMA7_PERIPHERAL_MAP に 0x6000 を書き込みます。
3. チャンネル 6 または 7、あるいはその両方で DMA をイネーブルにします。

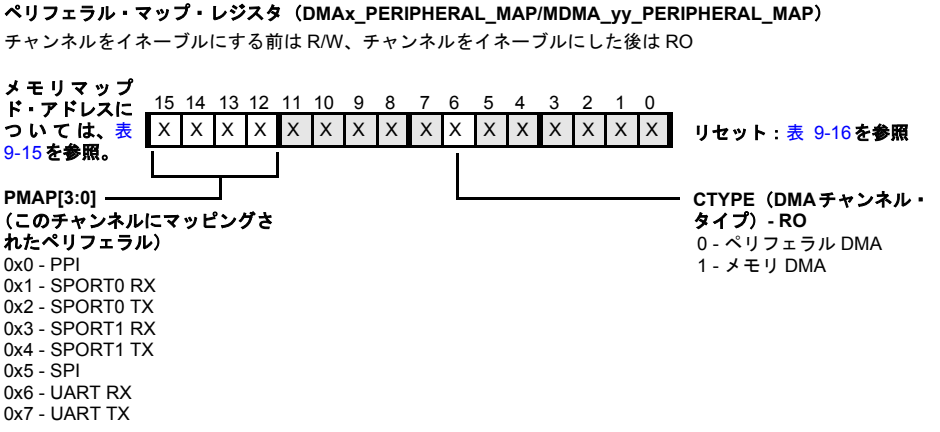


図 9-12. ペリフェラル・マップ・レジスタ

表 9-15.　ペリフェラル・マップ・レジスタのメモリマップド・アドレス

レジスタ名	メモリマップド・アドレス
DMA0_PERIPHERAL_MAP	0xFFC0 0C2C
DMA1_PERIPHERAL_MAP	0xFFC0 0C6C
DMA2_PERIPHERAL_MAP	0xFFC0 0CAC
DMA3_PERIPHERAL_MAP	0xFFC0 0CEC
DMA4_PERIPHERAL_MAP	0xFFC0 0D2C
DMA5_PERIPHERAL_MAP	0xFFC0 0D6C
DMA6_PERIPHERAL_MAP	0xFFC0 0DAC
DMA7_PERIPHERAL_MAP	0xFFC0 0DEC
MDMA_D0_PERIPHERAL_MAP	0xFFC0 0E2C
MDMA_S0_PERIPHERAL_MAP	0xFFC0 0E6C
MDMA_D1_PERIPHERAL_MAP	0xFFC0 0EAC
MDMA_S1_PERIPHERAL_MAP	0xFFC0 0EEC

表 9-16は、DMA対応のペリフェラルごとに、ペリフェラル・マップのバイナリ設定を示します。

表 9-16.　ペリフェラル・マッピング

DMA チャンネル	デフォルトのペリフェラル・マッピング	デフォルトの PERIPHERAL_MAP 設定（2 進）	備考
0（最高優先順位）	PPI	b#0000 0000 0000 0000	
1	SPORT0 RX	b#0001 0000 0000 0000	
2	SPORT0 TX	b#0010 0000 0000 0000	
3	SPORT1 RX	b#0011 0000 0000 0000	
4	SPORT1 TX	b#0100 0000 0000 0000	
5	SPI	b#0101 0000 0000 0000	
6	UART RX	b#0110 0000 0000 0000	

表 9-16. ペリフェラル・マッピング（続き）

DMA チャンネル	デフォルトのペリフェラル・マッピング	デフォルトの PERIPHERAL_MAP 設定（2 進）	備考
7	UART TX	b#0111 0000 0000 0000	
8	Mem DMA ストリーム 0 デスティネーション	b#0000 0000 0100 0000	再割当て不可
9	Mem DMA ストリーム 0 ソース	b#0000 0000 0100 0000	再割当て不可
10	Mem DMA ストリーム 1 デスティネーション	b#0000 0000 0100 0000	再割当て不可
11 （最低優先順位）	Mem DMA ストリーム 1 ソース	b#0000 0000 0100 0000	再割当て不可

■ DMAx_IRQ_STATUS/MDMA_yy_IRQ_STATUS レジスタ

図 9-13に示す割込みステータス・レジスタ（DMAx_IRQ_STATUS/MDMA_yy_IRQ_STATUS）には、DMAチャンネルについて以下の項目を記録するビットが含まれています。

- イネーブルで動作中、イネーブルで停止中、またはディスエーブル。
- データまたはDMAディスクリプタをフェッチしている。
- グローバルなDMA割込みまたはチャンネル割込みがアサートされていることを検出した。
- DMAエラーの発生を記録した。

なお、最後のメモリ・アクセス（読出し／書込み）が完了すると、DMA_DONE割込みがアサートされます。

ペリフェラルへのメモリ転送の場合、割込みの発生時にチャンネルのDMA FIFOには4つまでのデータ・ワードが存在することがあります。この時点では、次の作業単位をすぐに開始するのが普通です。しかし、アプリケーションが、最後のデータ項目がいつペリ

フェラルに実際に転送されたかを知る必要がある場合には、アプリケーションはDMA_RUNビットをテストしたりポーリングできます。FIFO内に未配信の送信データがある限り、DMA_RUNビットは1です。

メモリ書込みDMAチャンネルの場合、最後のDMA_DONEイベントが通知された後では、DMA_RUNビットの状態は無意味になります。これはDMA FIFOのステータスを示しません。

割込みを使用してDMA動作の終了を通知したくないMemDMA転送の場合には、トランザクションの完了を判断するために、ソフトウェアでDMA_DONEビットをポーリングしてください（DMA_RUNビットではありません）。

割込みステータス・レジスタ（DMAx_IRQ_STATUS/MDMA_yy_IRQ_STATUS）

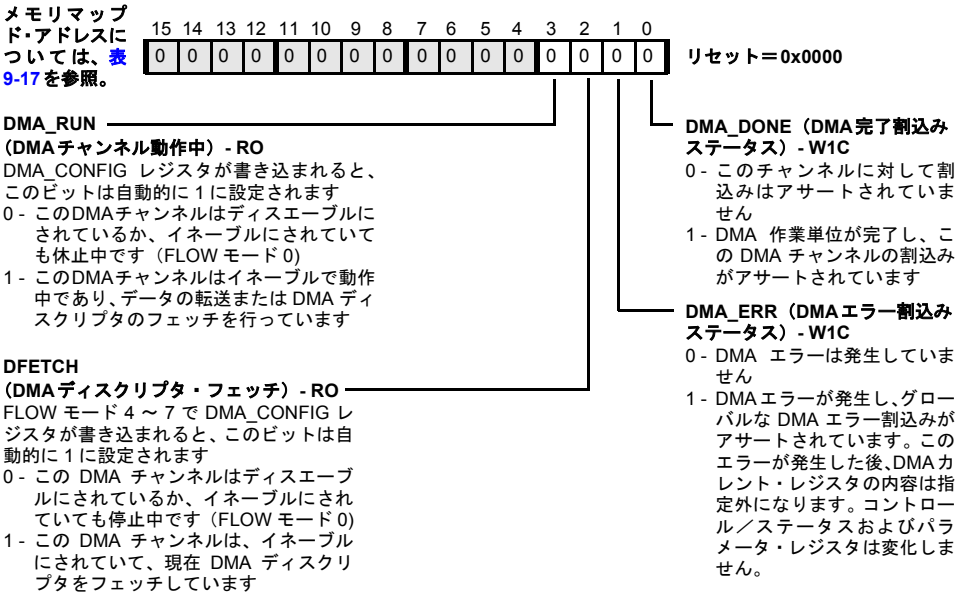


図 9-13. 割込みステータス・レジスタ

表 9-17.　割込みステータス・レジスタのメモリマップド・アドレス

レジスタ名	メモリマップド・アドレス
DMA0_IRQ_STATUS	0xFFC0 0C28
DMA1_IRQ_STATUS	0xFFC0 0C68
DMA2_IRQ_STATUS	0xFFC0 0CA8
DMA3_IRQ_STATUS	0xFFC0 0CE8
DMA4_IRQ_STATUS	0xFFC0 0D28
DMA5_IRQ_STATUS	0xFFC0 0D68
DMA6_IRQ_STATUS	0xFFC0 0DA8
DMA7_IRQ_STATUS	0xFFC0 0DE8
MDMA_D0_IRQ_STATUS	0xFFC0 0E28
MDMA_S0_IRQ_STATUS	0xFFC0 0E68
MDMA_D1_IRQ_STATUS	0xFFC0 0EA8
MDMA_S1_IRQ_STATUS	0xFFC0 0EE8

プロセッサは、3つの割込みソースによるフレキシブルな割込み制御構造をサポートします。

- データ駆動型割込み（表 9-18を参照）
- ペリフェラル・エラー割込み
- DMA エラー割込み（たとえば、不良ディスクリプタまたはバス・エラー）

データおよびペリフェラル・エラー割込み、ならびにDMAエラー割込みに対しては、別個の割込み要求（IRQ）レベルが割り当てられます。

表 9-18. データ駆動型割込み

割込み名	説明
割込みなし	特定の作業単位に対して割込みをディスエーブルにできます。
ペリフェラル割込み	これらはペリフェラル（非 DMA）割込みです。
行完了	行の完了時にDMA割込みが発生できます(CURR_X_COUNT満了)。
バッファ完了	バッファ全体の完了時に DMA 割込みが発生できます（CURR_X_COUNT と CURR_Y_COUNT の満了)。

すべてのDMAチャンネルの論理和（OR）をとって、1つのシステム・レベルのDMAエラー割込みにします。各チャンネルの個々のIRQ_STATUSワードを読み出して、DMAエラー割込みを引き起こしたチャンネルを識別できます。

なお、割込みインジケータDMA_DONEとDMA_ERRは、write-1-to-clear（W1C）です。

ペリフェラルをDMAモードから非DMAモードへ切り替えるときには、共有の DMA ／割込み要求ラインで想定外の割込みが生成されないよう、モード切替え中は（適切なペリフェラル・レジスタまたはSIC_IMASKによって)ペリフェラルの割込みをディスエーブルにしてください。

フレックス・ディスクリプタ構造

DMAフレックス・ディスクリプタは可変サイズのデータ構造であり、その内容はDMAパラメータ・レジスタにロードされます。ディスクリプタ内のレジスタのシーケンスは、3つの類似の変動の中で基本的には固定されていますが、ディスクリプタの長さは完全にプログラマブルです。DMAチャンネル・レジスタは、作業単位当たり最もよく再ロードされるレジスタが最下位のMMRアドレスにくるように順序付けされます。ユーザは、ディスクリプタを使用するかどうか選択できます。ディスクリプタを使用

しない場合、ユーザはDMA MMRに直接書込みを行ってDMAを起動し、連続動作には自動バッファ・モード、シングル・バッファ動作には停止モードを使用できます。

ディスクリプタを使用するには、ユーザは最下位のMMRアドレスから始めて、ディスクリプタからロードするDMAレジスタの数をDMAx_CONFIGレジスタのNDSIZEフィールドにプログラムします。ユーザは、1（START_ADDRの下位16ビット）～9（すべてのDMAパラメータ）のエントリからディスクリプタ・サイズを選択できます。

ディスクリプタ値シーケンスの3つの変動は、ネクスト・ディスクリプタ・ポインタが組み込まれているかどうかと、その種類（組み込まれている場合）に依存します。

- 組込みなし（ディスクリプタ・アレイ・モード）
- ネクスト・ディスクリプタ・ポインタの下位 16 ビット（ディスクリプタ・リスト、スモール・モデル）
- ネクスト・ディスクリプタ・ポインタの全 32 ビット（ディスクリプタ・リスト、ラージ・モデル）

ディスクリプタからロードされていない他のすべてのレジスタは以前の値を保持しますが、レジスタCURR_ADDR、CURR_X_COUNT、CURR_Y_COUNTは、ディスクリプタ・フェッチとDMA動作開始との間に再ロードされます。

DMA設定によっては、チェーン内のディスクリプタを変更できない場合もあります（スモール／ラージ・リスト・モードとアレイ・モード）。これらは、DMA方向、ワード・サイズ、メモリ空間です（つまり、内部メモリと外部メモリとの切り替えです）。

シングル・ディスクリプタ・チェーンでは、さまざまなメモリ空間に存在するデータ・バッファのシーケンスの転送を制御できません。その代わりに、複数のデータ・バッファを同じ空間内のバッファのチェーンにグルー

プ化しますが、それらのチェーンはリンクしません。最初のチェーンを転送し、その最終割込みを待ってから、DMA_CONFIGレジスタへのMMR書込みによって次のチェーンを開始します。

なお、ユーザは、各チェーンのデータ・バッファを同じメモリ空間に配置しなければなりません。しかし、ディスクリプタ構造自身はどのメモリ空間にも配置でき、1つの空間のディスクリプタから別の空間のディスクリプタに制約なしにリンクできます。

表 9-19は、上述の3つのモードでのディスクリプタ・エレメントのオフセットを示します。なお、表中の名前は、最終的にロードされる実際のMMRではなく、メモリ内のディスクリプタ・エレメントを示します。

表 9-19. パラメータ・レジスタとディスクリプタ・オフセット

ディスクリプタ・オフセット	ディスクリプタ・アレイ・モード	スモール・ディスクリプタ・リスト・モード	ラージ・ディスクリプタ・リスト・モード
0x0	SAL	NDPL	NDPL
0x2	SAH	SAL	NDPH
0x4	DMACFG	SAH	SAL
0x6	XCNT	DMACFG	SAH
0x8	XMOD	XCNT	DMACFG
0xA	YCNT	XMOD	XCNT
0xC	YMOD	YCNT	XMOD
0xE		YMOD	YCNT
0x10			YMOD

DMA 動作フロー

図 9-14 と図 9-15 は、DMA フローを示します。

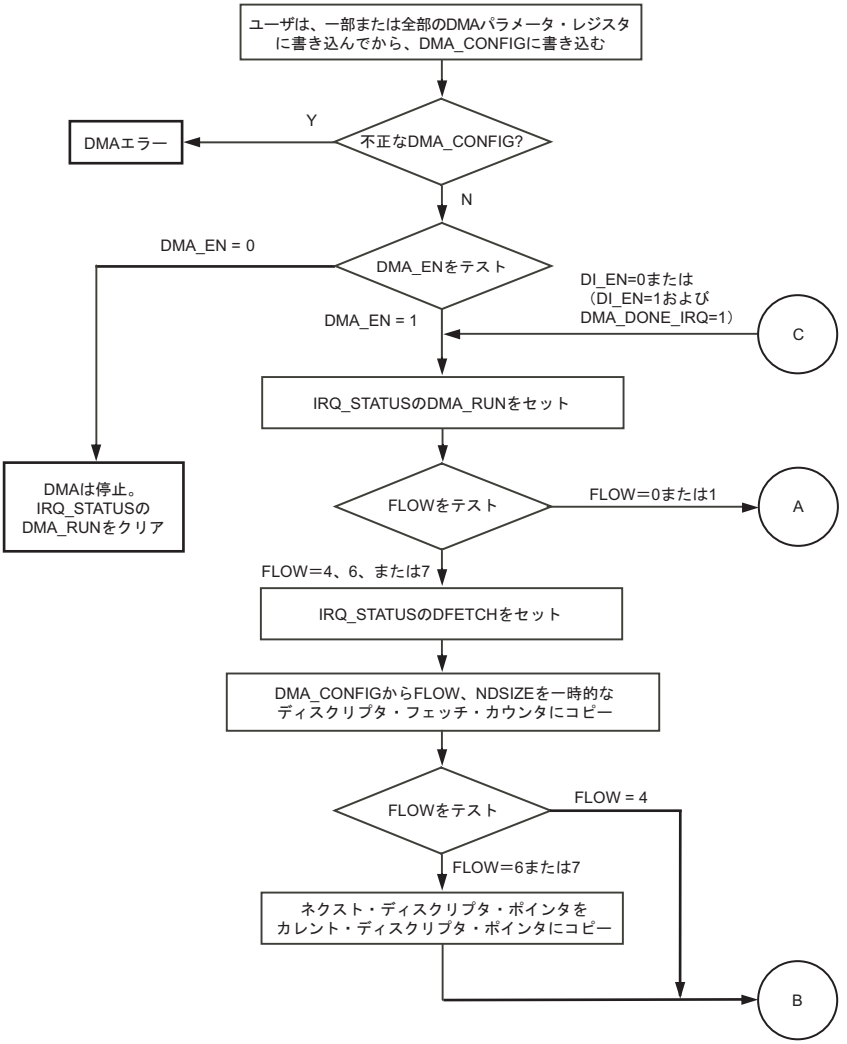


図 9-14. DMA フロー — DMA コントローラから見た場合（1/2）

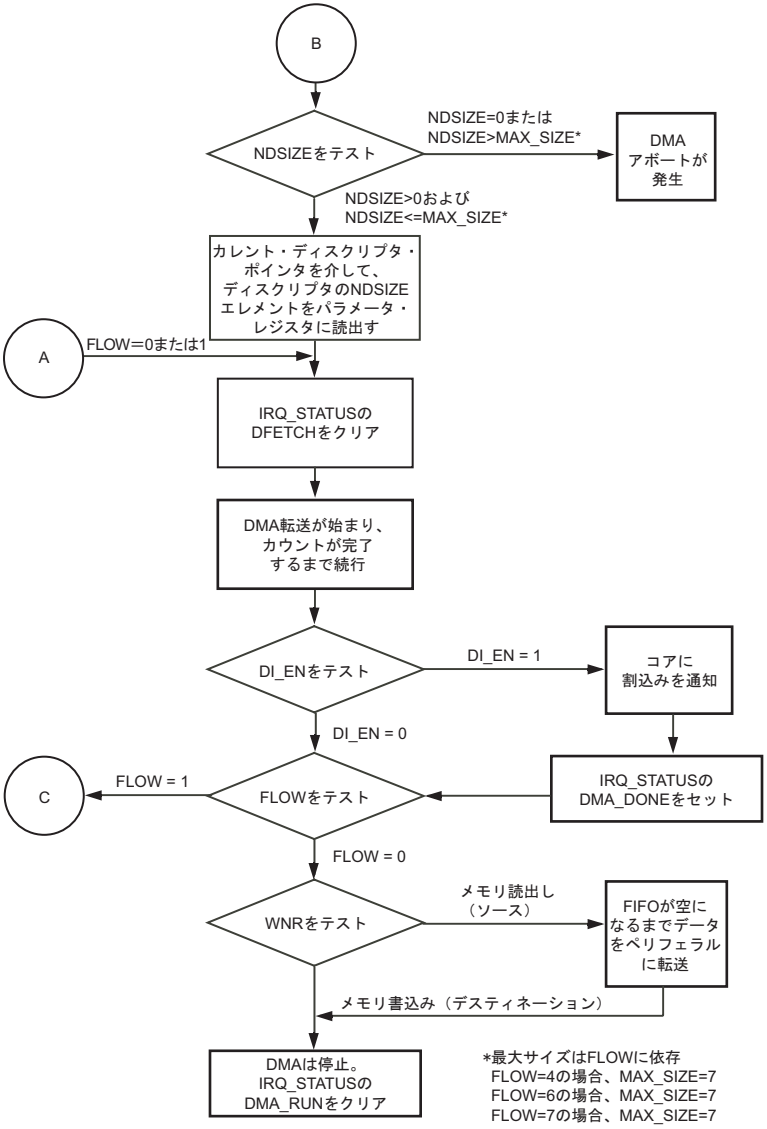


図 9-15. DMA フロー — DMA コントローラから見た場合（2/2）

■ DMA の起動

ここでは、DMAを「ゼロから」起動する方法について説明します。これは、FLOW = 0モードによってDMAが休止した後、DMAを起動する方法と似ています。

特定のチャンネルで初めて DMA を起動する前に、すべてのパラメータ・レジスタを初期化してください。NEXT_DESC_PTR レジスタと START_ADDR レジスタの上位 16 ビットの初期化には特にご注意ください。これは、選択された FLOW 動作モードによってはアクセスできないこともあるためです。

特定のチャンネルでDMA動作を開始するには、最初に、一部または全部のDMAパラメータ・レジスタに直接書き込む必要があります。最低限、この段階ではNEXT_DESC_PTRレジスタ（またはFLOW = 4モードではCURR_DESC_PTRレジスタ）に書き込む必要がありますが、DMA動作の全過程にわたって静的である可能性のある他のDMAレジスタ（たとえば、X_MODIFY、Y_MODIFY）に書き込むこともできます。DMA_CONFIG内のNDSIZEとFLOWの内容は、メモリ内のディスクリプタ・エレメントからフェッチされたレジスタを（もしあれば）示します。ディスクリプタ・フェッチが（もしあれば）完了した後、DMA_CONFIGにDMA_EN = 1を書き込んで、DMA動作が開始されます。

DMA_CONFIGが直接書き込まれると、DMAコントローラではこれをチャンネルで初めてDMAを起動したりエンジンの停止後（FLOW = 0）にDMAを起動したりするときに発生する、特殊な起動条件として認識します。

ディスクリプタ・フェッチが完了し、DMA_EN = 1であるとき、DMA_CONFIGに読み出されたDMACFGディスクリプタ・エレメントが制御を引き継ぎます。これ以前の時点では、DMA_CONFIGへの直接書込みに制御がありました。つまり、WDSIZE、DI_EN、DI_SEL、RESTART、DMA2Dの各フィールドは、メモリから読み出されたディスクリプタ内のDMACFG値から取得されますが、初めにDMA_CONFIGレジスタに書き込まれたこれらのフィールド値は無視されます。

図 9-14と図 9-15に示すように、DMA設定プロセスの過程は、起動時にDMA_CONFIGのFLOWビットとNDSIZEビットによって決まります。FLOW値では、メモリ内のディスクリプタ・エレメントからカレント・レジスタをさらにロードすべきかどうかが決まります。NDSIZEビットでは、DMAの起動前にフェッチするディスクリプタ・エレメントの数が決まります。ディスクリプタに組み込まれていないDMAレジスタは、以前の値から変更されません。

FLOW値でスモールまたはラージのディスクリプタ・リスト・モードを指定する場合には、NEXT_DESC_PTRがCURR_DESC_PTRにコピーされます。その後、メモリから新しいディスクリプタ・エレメントのフェッチが実行され、フェッチのたびにインクリメントされるCURR_DESC_PTRによってインデックス付けされます。NDPLまたはNDPH、あるいはその両方がディスクリプタの一部である場合には、これらの値はNEXT_DESC_PTRにロードされますが、カレント・ディスクリプタのフェッチはCURR_DESC_PTRを使用して続行されます。ディスクリプタ・フェッチが完了した後、CURR_DESC_PTRはディスクリプタの最後を越えてメモリ内の次の16ビット・ワードを指し示します。

NDPHもNDPLもディスクリプタの一部でない場合には（つまり、ディスクリプタ・アレイ・モードで、FLOW = 4）、NDPH/NDPLからCURR_DESC_PTRへの転送は発生しません。その代わりに、CURR_DESC_PTR内の値でディスクリプタ・フェッチのインデックス付けが始まります。

DMACFGがディスクリプタの一部でない場合には、起動時にMMRアクセスによって書き込まれた以前のDMA_CONFIG設定が作業単位の動作を制御します。DMACFGがディスクリプタの一部である場合には、MMRアクセスによってプログラムされたDMA_CONFIG値がメモリからの最初のディスクリプタのロードだけを制御します。それ以降のDMA作業は、ディスクリプタのDMACFGの下位バイトと、ディスクリプタからロードされたパラメータ・レジスタによって制御されます。MMRアクセスによってプログラムされた値に含まれるビットDI_EN、DI_SEL、DMA2D、WDSIZE、WNRは無視されます。

IRQ_STATUS レジスタ内のステータス・ビット DMA_RUN と DFETCH は、DMA チャンネルの状態を示します。DMA_CONFIG への書込み後、DMA_RUN ビットと DFETCH ビットは自動的に 1 に設定されます。メモリから最初のディスクリプタをロードした結果として、データ割込みの通知はありません。

上の手順の後で、カレント・レジスタは適切なディスクリプタ・エレメントから自動的にロードされ、以前の内容は次のように上書きされます。

- START_ADDR は CURR_ADDR にコピーされます
- X_COUNT は CURR_X_COUNT にコピーされます
- Y_COUNT は CURR_Y_COUNT にコピーされます

その後、9-39 ページの図 9-15 に示すように、DMA データ転送動作が始まります。

■ DMA のリフレッシュ

作業単位が完了すると、DMA コントローラは以下の動作を行います。

- メモリと DMA ユニットとの間ですべてのデータの転送を完了します。
- DI_EN によってイネーブルにされた場合には、コアへの割込みを通知し、チャンネルの IRQ_STATUS レジスタの DMA_DONE ビットをセットします。
- FLOW = 0（停止）の場合のみ：

  チャンネルの DMA FIFO 内のデータがペリフェラルに転送された後で、IRQ_STATUS の DMA_RUN ビットをクリアして動作を停止させます。

- FLOWモード4、6、7でのフェッチ中に、DMAコントローラはIRQ_STATUSのDFETCHビットを1に設定します。この時点で、DMA動作はFLOWの値（FLOW = 4、6、または7）に依存します。

  FLOW = 4（ディスクリプタ・アレイ）の場合：

  CURR_DESC_PTRの内容を介して新しいディスクリプタをメモリからDMAレジスタにロードすると同時に、CURR_DESC_PTRをインクリメントします。フェッチが開始される前に、ディスクリプタ・サイズはDMA_CONFIG値のNDSIZEフィールドから得られます。

  FLOW = 6（ディスクリプタ・リスト・スモール）の場合：

  32ビットのNEXT_DESC_PTRをCURR_DESC_PTRにコピーします。次に、CURR_DESC_PTRの新しい内容を介してディスクリプタをメモリからDMAレジスタにフェッチすると同時に、CURR_DESC_PTRをインクリメントします。ロードされる最初のディスクリプタ・エレメントはNEXT_DESC_PTRの下位16ビットの新しい16ビット値であり、残りのディスクリプタ・エレメントがそれに続きます。NEXT_DESC_PTRの上位16ビットは、前の値を保持します。これによって、ディスクリプタ・リストのラージ・モデルよりも短く効率的なディスクリプタがサポートされ、アプリケーションはメモリの同じ64Kバイト範囲内にチャンネルのディスクリプタを配置できます。

  FLOW = 7（ディスクリプタ・リスト・ラージ）の場合：

  32ビットのNEXT_DESC_PTRをCURR_DESC_PTRにコピーします。次に、CURR_DESC_PTRの新しい内容を介してディスクリプタをメモリからDMAレジスタにフェッチすると同時に、CURR_DESC_PTRをインクリメントします。ロードされる最初のディスクリプタ・エレメントはフルNEXT_DESC_PTRの新しい32ビット値であり、残りのディスクリプタ・エレメントがそれに続きます。NEXT_DESC_PTRの上位16

ビットは、その前の値とは違うことがあります。これによってサポートされる完全にフレキシブルなディスクリプタ・リストは、内部メモリまたは外部メモリのどこにでも配置できます。

なお、1 つの 64K バイト領域にディスクリプタを持つディスクリプタ・チェーンから、その領域の外部にディスクリプタを持つ別のチェーンにリンクする必要がある場合には、1 つのディスクリプタだけが FLOW = 7 を使用する必要があります。つまり、64K バイト範囲を出ているリンクを含むディスクリプタです。同じ 64K バイト領域に置かれている他のすべてのディスクリプタでは、FLOW = 6 を使用できます。

- FLOW ＝1、4、6、または7（それぞれ、自動バッファ、ディスクリプタ・アレイ、ディスクリプタ・リスト・スモール、ディスクリプタ・リスト・ラージ）の場合：

  カレント・レジスタを（再）ロードします。

  CURR_ADDR は START_ADDR からロード
  CURR_X_COUNT は X_COUNT からロード
  CURR_Y_COUNT は Y_COUNT からロード

  続いて IRQ_STATUS の DFETCH ビットがクリアされてから、図 9-15 に示すように、DMA 転送が再開されます。

■ DMA 転送を停止するには

FLOW = 0モードでは、作業単位の完了後、DMAは自動的に停止します。

DMAの制御にディスクリプタのリストまたはアレイが使用され、どのディスクリプタにもDMACFGエレメントが含まれる場合、チャンネルを適切に停止させるために、最終のDMACFGエレメントではFLOW = 0に設定してください。

自動バッファ（FLOW = 1）モードの場合、またはDMACFGエレメントを持たないディスクリプタのリストまたはアレイが使用される場合には、DMA_ENビットを0にした値をDMAx_CONFIGレジスタにMMR書込みすることで、DMA転送プロセスを終了させる必要があります。レジスタ全体に0を書き込むと、DMAは常に適切に終了します（DMAのアボートなし）。

チャンネルを再びイネーブルにする前に、低速なメモリ読出し動作（たとえば、低速な外部メモリからの読出し）が開始されていれば、その読出し動作が完了していることを確認します。このような読出しが完了するまでは、チャンネルを再びイネーブルにしないでください。

■ DMA 転送をトリガするには

FLOW＝0モードでDMAが停止している場合、内部DMA FIFOの内容が完全に処理されるまでは、DMA割込みステータス・レジスタのDMA_RUNビットはセットされたままです。DMA_RUNビットがクリアされると、DMA設定レジスタへの再書込みによってDMAを再起動すると安全です。DMAシーケンスは、以前の設定で繰り返されます。

同様に、FLOW＝0ディスクリプタによって一時的に停止したディスクリプタ・ベースのDMAシーケンスは、設定レジスタへの新しい書込みによって続行できます。DMAコントローラがメモリからのDMACFGフィールドのロードによってFLOW＝0条件を検出した場合には、動作モードがディスクリプタ・アレイまたはディスクリプタ・リストであっても、ネクスト・ディスクリプタ・ポインタはすでに更新されています。

DMAが停止して再起動された場合には、ネクスト・ディスクリプタ・ポインタは有効なままです。DMA_RUNビットがクリアされるとすぐに、ソフトウェアはDMAを再起動し、DMAコントローラに次のディスクリプタをフェッチさせることができます。そのために、ソフトウェアは、DMA_ENビットをセットしてFLOWフィールドとNDSIZEフィールドに適切な値を持つ値を設定レジスタに書き込みます。FLOWが0x4、0x6、または0x7に等しい場合には、次のディスクリプタがフェッチされます。この動作モードでは、設定レジスタをすぐに上書きするために、NDSIZEフィールドは少なくともDMACFGフィールドまで及ぶ必要があります。

ディスクリプタ・チェーン内のすべてのDMACFGフィールドで、FLOWフィールドとNDSIZEフィールドがゼロに設定されている場合には、個々のDMAシーケンスはソフトウェアによってトリガされるまでは開始されません。これは、DMAをシステム内の他のイベントと同期させる必要がある場合に便利であり、一般には割込みサービス・ルーチンによって実行されます。次のDMAシーケンスをトリガするには、1回のMMR書込みが必要です。

特にMemDMAチャンネルに適用された場合、このようなシナリオは重要な役割を果たします。通常、MemDMAのタイミングは制御できません。ディスクリプタのチェーンまたはリングをこのように停止させることによって、DMAトランザクション全体をソフトウェアによって個々にトリガされる要素に分解できます。

MemDMAのソース・チャンネルとデスティネーション・チャンネルは、ディスクリプタ構造内で異なることもあります。しかし、DMAが停止したときに、ワーク・カウントの合計は一致しなければなりません。MemDMAが停止するたびに、デスティネーション・チャンネルとソース・チャンネルはちょうど同じワード数の後で、いずれも同じFLOW＝0モードを提供するはずです。したがって、これらのチャンネルは後から起動する必要があります。

2 次元 DMA

2次元（2D）DMAでは、64×64Kエレメントまでの任意の行／列サイズに加えて、± 32Kバイトまでの任意のX_MODIFY値とY_MODIFY値を利用できます。さらに、Y_MODIFYは負とすることができるため、インターリーブされたデータストリームを実装できます。X_COUNT値とY_COUNT値では、行と列のサイズを指定します。ここで、X_COUNTは2以上である必要があります。

開始アドレスと変更値はバイト単位であり、DMA転送ワード・サイズ（DMA_CONFIGのWDSIZE[1:0]）の倍数に整列する必要があります。不整列の場合には、DMAエラーが発生します。

X_MODIFY値は、CURR_X_COUNTレジスタをデクリメントする転送のたびに適用されるバイト・アドレス・インクリメントです。内側ループ・カウントがCURR_X_COUNTを1から0にデクリメントして終了した場合には、X_MODIFY値は適用されません。ただし、CURR_Y_COUNTが1であり、CURR_X_COUNTが1から0にデクリメントした場合には、最後の転送で適用されます。

Y_MODIFY値は、CURR_Y_COUNTがデクリメントされるたびに適用されるバイト・アドレス・インクリメントです。しかし、外側ループ・カウント（CURR_Y_COUNT）も1から0にデクリメントして満了するアレイ内の最終項目に対しては、Y_MODIFY値は適用されません。

最後の転送が完了した後、CURR_Y_COUNT ＝ 1、CURR_X_COUNT = 0、そしてCURR_ADDRは最後の項目のアドレス＋X_MODIFYに等しくなります。なお、DMAチャンネルが自動的にリフレッシュするようにプログラムされている場合（自動バッファ・モード）、これらのレジスタは最初のデータ転送と同時にX_COUNT、Y_COUNT、START_ADDRからロードされます。

■ 例

例1：サイズ（N×M）ピクセルのビデオ・フレーム・バッファから、16×8ブロックのバイトを取り出します。

```
X_MODIFY = 1
X_COUNT = 16
Y_MODIFY = N-15  (1つの行の終端から別の行の先頭までのオフセット)
Y_COUNT = 8
```

上の記述によって、開始アドレスからの以下のアドレス・オフセットが得られます。

```
0,1,2,...15,
N,N + 1, ... N + 15,
2N, 2N + 1,... 2N + 15, ...
7N, 7N + 1,... 7N + 15,
```

例2：（R、G、Bピクセル）×（N×Mイメージ・サイズ）バイトのビデオ・データストリームを受け取ります。

```
X_MODIFY = (N * M)
X_COUNT = 3
Y_MODIFY = 1 - 2(N * M)  (負)
Y_COUNT = (N * M)
```

上の記述によって、開始アドレスからの以下のアドレス・オフセットが得られます。

```
0, (N * M), 2(N * M),
1, (N * M) + 1, 2(N * M) + 1,
2, (N * M) + 2, 2(N * M) + 2,
    ...
(N * M) - 1, 2(N * M) - 1, 3(N * M) - 1,
```

■ 2D DMA のその他の例

フレックス・ディスクリプタで利用できるDMAスタイルには、以下の例が含まれます。

- 完了時に停止する単一のリニア・バッファ（FLOW ＝停止モード）
- 1を超えるストライドを持つリニア・バッファ（X_MODIFY > 1）
- フル・バッファごとに割込みする循環自動リフレッシュ・バッファ
- 分数バッファ（たとえば、1/2、1/4）で割込みする類似のバッファ（2D DMA）
- 1D DMA、{リンク・ポインタ、32ビット・アドレス}を含む3ワード・ディスクリプタのリンク・リングによって定義される一連の同じピンポン・バッファを使用
- 1D DMA、{リンク・ポインタ、32ビット・アドレス、長さ、config}を含む 5 ワード・ディスクリプタのリンク・リストを使用（ADSP-2191スタイル）
- 2D DMA、共通データ・ページ内のベースDMAアドレスだけを指定する、1ワード・ディスクリプタのアレイを使用
- 2D DMA、すべてを指定する、9ワード・ディスクリプタのリンク・リストを使用

メモリ DMA

ここでは、メモリDMA（MDMA）コントローラについて説明します。MDMAコントローラは、さまざまなメモリ空間の中でメモリ間DMA転送を実現します。これらには、L1メモリや外部の同期／非同期メモリを含みます。

各MDMAコントローラに含まれるDMA FIFOは、L1または外部アクセス・バス（EAB）との間でのデータ転送に使用される8ワード×16ビットのFIFOブロックです。一般に、これは外部メモリと内部メモリとの間のデータ転送に使用されます。EABバスでのブートROMからのDMAにも対応します。このFIFOを使用すれば、2つのL1メモリ位置の間または2つの外部メモリ位置の間で転送されるDMAデータを保持できます。

プロセッサは4本のMDMAチャンネルを提供します。

- 2本のソース・チャンネル（メモリからの読出し用）
- 2本のデスティネーション・チャンネル（メモリへの書込み用）

各ソース／デスティネーション・チャンネルは「ストリーム」を形成し、この2つのストリームは、DMA優先順位8～11に配線接続されます。

- 優先順位8：メモリDMAデスティネーション・ストリームD0
- 優先順位9：メモリDMAソース・ストリームD0
- 優先順位10：メモリDMAデスティネーション・ストリームD1
- 優先順位11：メモリDMAソース・ストリームD1

ラウンド・ロビン・スケジューリングが使用される場合を除いて、メモリDMAストリーム0はメモリDMAストリーム1よりも優先順位が高くなります。なお、メモリ書込み用にソース・ストリームをプログラムしたり、メモリ読出し用にデスティネーション・ストリームをプログラムしたりすることは不正な行いとなります。

これらのチャンネルでは、8/16/32ビットのメモリDMA転送が可能ですが、MDMA転送の両端は、同じワード・サイズにプログラムする必要があります。つまり、MDMA転送ではデータのパックやアンパックを行いません。各読出しは1回の書込みにつながります。特定ストリームのMDMA FIFOの両端には、同時に優先順位が与えられます。各ペアは、深さ8ワードの16ビットFIFOを共有します。ソースDMAエンジンはFIFOをフィルしますが、デスティネーションDMAエンジンはFIFOを空にします。FIFOの深さによって、外部アクセス・バス（EAB）とDMAアクセス・バス（DAB）のバースト転送は重なり合うことができるため、内部メモリと外部メモリとの間のブロック転送でスループットが大幅に向上します。各MDMAペアに操作パラメータを提供する（ソース・チャンネルとデスティネーション・チャンネルに1つずつ）には、2つの別個のディスクリプタ・ブロックが必要です。

ソースとデスティネーションのDMAエンジンは1つのFIFOバッファを共有するため、ディスクリプタ・ブロックは同じデータ・サイズを持つように設定する必要があります。合計カウントが同じである限り、両端で異なる組合わせのディスクリプタを持つことは可能です。

MDMA転送動作を開始するには、ソース・ストリーム用とデスティネーション・ストリーム用のMMRが、それぞれペリフェラルDMAの場合と似た方法で書き込まれます。

なお、ソース・ストリーム用のDMA_CONFIGレジスタには、デスティネーション・ストリーム用のDMA_CONFIGレジスタよりも前に書き込む必要があります。

デスティネーションDMA_CONFIGレジスタが書き込まれると、3 SCLKサイクルの遅延の後でMDMA動作が開始されます。

最初に、いずれのMDMAストリームがディスクリプタを使用するように選択された場合でも、ディスクリプタはメモリからフェッチされます。デスティネーション・ストリームのディスクリプタは最初にフェッチされます。続いて、メモリから最後のディスクリプタ・ワードが返されてから4

SCLKサイクルの遅延の後で（または、メモリ・パイプライン処理によって、最後のディスクリプタ・ワードのフェッチから一般に8 SCLKサイクルで）、ソースMDMAストリームはソース・バッファからデータのフェッチを始めます。得られたデータはMDMAストリームの8段FIFOに格納されます。続いて、2 SCLKサイクルの遅延の後で、デスティネーションMDMAストリームはデスティネーション・メモリ・バッファへのデータ書込みを開始します。

■ MDMA 帯域幅

ソースとデスティネーションが異なるメモリ空間（1つは内部、1つは外部）にある場合には、内部と外部のメモリ転送は一般に同時かつ連続であり、内部と外部のメモリ・インターフェースでは100%のバス使用率が維持されます。この性能は、コア―システムのクロック周波数比率によって影響を受けます。およそ2.5:1より低い比率では、同期とパイプライン遅延によって、システム・クロック・ドメインでのバス使用率は低下します。たとえば2:1のクロック比率では、DMAは一般にシステム・クロック・レートの2/3で動作します。さらに高いクロック比率では、フル帯域幅が維持されます。

ソースとデスティネーションが同じメモリ空間にある（両方とも内部、または両方とも外部）場合には、MDMAストリームは、一般にソース・データのバーストをFIFOにプリフェッチしてから自動的に向きを変えて、使用可能なすべてのデータをFIFOからデスティネーション・バッファに配信します。バースト長はトラフィックに依存し、SCLK単位で3＋DMAでのメモリ遅延（一般に、内部転送では7、外部転送では6）に等しくなります。

DMA 性能の最適化

DMAシステムは、チャンネル当たり最大のスループットと内部バスの最大の使用率を提供すると同時に、メモリ・アクセスに固有の遅延に対処できるように設計されています。

DMAアーキテクチャの重要な特長は、ペリフェラルDMAバス（DMAアクセス・バス（DAB））での動作と、DMAとメモリ（DMAコア・バス（DCB）とDMA外部バス（DEB））との間のバスでの動作を分離することです。各ペリフェラルDMAチャンネルには、DABバスとメモリ・バスとの間に専用のデータFIFOがあります。これらのFIFOでは、伝送用にデータをメモリから自動的にプリフェッチし、後でのメモリ書込み用に受信データをバッファします。これによって、ペリフェラルにはパイプライン化されたメモリ・アクセスの合計遅延に比べてきわめて低遅延のDMA転送が認められます。したがって、各DMAチャンネルのリピート間隔（帯域幅）は可能な限り高速になります。

ペリフェラルDMAチャンネルの最大転送速度は、いずれの方向でもチャンネル当たり、2システム・クロック当たりで16ビット1ワードです。

MDMAチャンネルの最大転送速度はチャンネル当たり、1システム・クロック（SCLK）当たりで16ビット1ワードです。

すべてのDMAチャンネルのトラフィックが全体として取得された場合：

- ペリフェラルとDMAユニットとの間の最大転送レートは、システム・クロック当たり16ビット1転送です。
- DMAユニットと内部メモリ（L1）との間の最大転送レートは、システム・クロック当たり16ビット1転送です。
- DMAユニットと外部メモリとの間の最大転送レートは、システム・クロック当たり16ビット1転送です。

実際の性能を制約する問題点には、以下のものが含まれます。

- 同じメモリへのコア・アクセスで競合する内部または外部メモリへのアクセス。これによって、たとえば、同じL1バンクへのアクセス、SDRAMページのオープン／クローズ、キャッシュ・ラインのフィル中に遅延が発生することがあります。
- DABバスでのRXからTXへの方向転換のたびに、1 SCLKサイクルの遅延を招きます。
- DCB バスでの内部メモリの同じバンクに対する方向転換（たとえば、書込み後の読出し）によって、遅延を招くことがあります。
- DEB バスでの外部メモリに対する方向転換（たとえば、読出し後の書込み）によって、それぞれ複数サイクルの遅延を招くことがあります。
- DMAx_CONFIG、DMAx_IRQSTAT、DMAx_PERIPHERAL_MAP以外のDMAレジスタへのMMRアクセスでは、転送された16ビット・ワード当たり1サイクルだけ、すべてのDMA動作がストールします。それと対照的に、コントロール／ステータス・レジスタへのMMRアクセスでは、ストールや待ち状態は発生しません。
- コントロール／ステータス・レジスタ以外のDMAレジスタからの読出しでは、1つのPABバス待ち状態を使用し、コアは複数コア・クロックだけ遅れます。
- ディスクリプタ・フェッチでは、メモリからの 16 ビット・ワード読出し当たり1 DMAメモリ・サイクルを消費しますが、DABバスでの転送は遅延しません。
- DMAチャンネルの初期化では、DMA動作が1サイクルだけストールします。これが発生するのは、DMA_ENが0から1に変化したとき、またはDMAx_CONFIGレジスタでRESTARTビットが1に設定されたときです。

これらの要因のいくつかは、アプリケーション・ソフトウェアの適切な設計によって最小限に抑えることができます。バンク内やページ内でのデータ・バッファの割当てに注意し、クリティカルなDMA動作中にはキャッシュ動作を低く計画することによって、内部メモリと外部メモリの競合を避けるようにソフトウェアを構成することも可能です。さらには、ディスクリプタや自動バッファリングを使用することで、不必要なMMRアクセスを最小限に抑えることができます。

過度の方向転換に起因する効率損失（スラッシング）を最小限に抑えるには、次の節で説明するプロセッサのトラフィック制御機能を使用します。

■ 優先順位付けとトラフィック制御

DMAチャンネルは、その優先順位に厳密に基づいてサービスを許可されるのが普通です。チャンネルの優先順位は、単にそのチャンネル番号であり、優先順位の低い番号が最初に許可されます。したがって、高いデータ・レートや低い遅延条件を持つペリフェラルは、DMAx_PERIPHERAL_MAPレジスタを使用して小さい番号の（優先順位の高い）チャンネルに割り当てます。メモリDMAストリームは、ペリフェラルよりも常に低い優先順位ですが、サービスを連続的に要求するので、ペリフェラルDMAによって使用されていないタイム・スロットがMDMA転送に適用されるようにします。デフォルトでは、複数のMDMAストリームがイネーブルにされて準備が完了すると、最高優先順位のMDMAストリームだけが許可されます。しかし、複数のMDMAストリームが使用可能な帯域幅を共有することが望ましい場合には、一定回数の転送に対して各ストリームを順番に選択するように、MDMA_ROUND_ROBIN_PERIODをプログラムできます。

プロセッサDMAには、DABバス優先順位付けとメモリ・バス（DCBおよびDEB）優先順位付けという、完全に独立した2つの同時優先順位付けプロセスがあります。DABバスを介してDMAを要求し、転送の処理準備ができたデータFIFOを持つペリフェラルは、DABバス・サイクルを求めて互いに競争します。同様に（ただし別個に）、メモリ・サービス（プリフェッチまたはポストライト）を必要とするFIFOを持つチャンネルは、

メモリ・バスへのアクセスを求めて競争します。複数のMDMAストリームは、1つのユニットとしてメモリ・アクセスを求めて競争し、メモリ転送が競合しない場合には、ソースとデスティネーションが共に許可されることもあります。このように、内部—外部または外部—内部のメモリ転送は、フル・システム・クロック・レート（SCLK）で行われることがあります。メモリ競合の例には、同じメモリ空間への同時アクセスと、ディスクリプタの同時フェッチが含まれます。ペリフェラルがDMAを要求していても、そのFIFOの準備ができていない（たとえば、送信FIFOが空、受信FIFOが満杯）場合には、特殊な処理が行われることもあります。詳細については、9-62ページの「緊急のDMA転送」を参照してください。

DMAリソースの使用を最適化する際に、トラフィック制御は重要なポイントです。トラフィック制御とは、同じ方向の転送を自動的にグループ化することによって、データ・バスでの転送方向の変更頻度に影響を与える方法の1つです。DMAブロックでは、DMA_TC_PERレジスタとDMA_TC_CNTレジスタによって制御されるトラフィック制御メカニズムを提供します。このメカニズムによる最適化では、プロセッサのリアルタイム介入がなく、DMA作業単位ストリームに転送バーストをプログラムする必要もありません。トラフィックは、簡単なカウンタによって、3つのバス（DAB、DCB、DEB）ごとに独立して制御できます。さらに、MDMAストリーム間での転送の交代は、DMA_TC_CNTレジスタのMDMA_ROUND_ROBIN_COUNTフィールドで制御できます。9-60ページの「MDMA優先順位とスケジューリング」を参照してください。

トラフィック制御機能を使用して、DMAシステムは、トラフィック制御カウンタがタイムアウトするか、トラフィックが自然に停止または方向転換するまでは、これまでの転送と同じ読出し／書込み方向に進むDABまたはメモリ・バスでのデータ転送を優先的に許可します。トラフィック・カウンタがゼロに達すると、優先権は逆の流れ方向に変更されます。このような方向の優先権は、あたかも逆方向のチャンネルの優先順位が16だけ減少したかのように機能します。

たとえば、チャンネル3と5がDABアクセスを要求していても、低優先順位のチャンネル5が「トラフィックと一緒」に進み、高優先順位のチャンネル3が「トラフィックと逆」に進む場合には、チャンネル3の実際の優先順位は19になり、その代わりにチャンネル5が許可されます。次のサイクルで、チャンネル3と6だけがDAB転送を要求しており、これらの転送要求が共に「トラフィックと逆」であった場合には、これらの実際の優先順位はそれぞれ19と22になります。チャンネルの1つ（チャンネル3）が許可されますが、その方向は現在の流れとは逆です。バスの方向転換に必要な遅延を除いて、バス・サイクルは浪費されません。

このタイプのトラフィック制御は、利用率（効率）を改善するための遅延のトレードオフを表します。トラフィック・タイムアウトを高くすると、各要求がその許可を待つ時間は増大しますが、混雑したシステムにおいては、達成可能な最大帯域幅を大幅に改善（90%以上）することがよくあります。

優先的なDMA優先順位付けをディスエーブルにするには、DMA_TC_PERレジスタを0x0000にプログラムします。

▶ DMA_TC_PER レジスタと DMA_TC_CNT レジスタ

DMAトラフィック制御カウンタ周期レジスタ（DMA_TC_PER）とDMAトラフィック制御カウンタ・レジスタ（DMA_TC_CNT）は、他のDMAレジスタと連携してトラフィック制御を定義します。

MDMA_ROUND_ROBIN_COUNTフィールドは、MDMAラウンド・ロビン周期に残る現在の転送カウントを示します。DMA_TC_PERが書き込まれたり、別のMDMAストリームが許可されたり、すべてのMDMAストリームがアイドル状態になったりするたびに、このフィールドはMDMA_ROUND_ROBIN_PERIODに初期化されます。その後、各MDMA転送によって0までカウント・ダウンします。このカウントが1から0にデクリメントすると、次に使用可能なMDMAストリームが選択されます。

DAB_TRAFFIC_COUNT フィールドは、DAB トラフィック周期に残る現在のサイクル・カウントを示します。DMA_TC_PER が書き込まれたり、DAB バスが方向転換したりアイドル状態になったりするたびに、このフィールドは DAB_TRAFFIC_PERIOD に初期化されます。その後、各システム・クロック（DMA ストールの場合を除く）で DAB_TRAFFIC_PERIOD から 0 までカウント・ダウンします。このカウントが非ゼロである間は、同じ方向の DAB アクセスは優先的に扱われます。このカウントが 1 から 0 までデクリメントすると、逆方向の DAB アクセスが優先的に扱われ、方向転換をもたらします。このカウントが 0 で、DAB バス・アクセスが発生すると、新しいバーストを始めるために、カウントは DAB_TRAFFIC_PERIOD から再ロードされます。

DEB_TRAFFIC_COUNT フィールドは、DEB トラフィック周期に残る現在のサイクル・カウントを示します。DMA_TC_PER が書き込まれたり、DEB バスが方向転換したりアイドル状態になったりするたびに、このフィールドは DEB_TRAFFIC_PERIOD に初期化されます。その後、各システム・クロック（DMA ストールの場合を除く）で DEB_TRAFFIC_PERIOD から 0 までカウント・ダウンします。このカウントが非ゼロである間は、同じ方向の DEB アクセスは優先的に扱われます。このカウントが 1 から 0 までデクリメントすると、逆方向の DEB アクセスが優先的に扱われ、方向転換をもたらします。このカウントが 0 で、DEB バス・アクセスが発生すると、新しいバーストを始めるために、カウントは DEB_TRAFFIC_PERIOD から再ロードされます。

DCB_TRAFFIC_COUNT フィールドは、DCB トラフィック周期に残る現在のサイクル・カウントを示します。DMA_TC_PER が書き込まれたり、DCB バスが方向転換したりアイドル状態になったりするたびに、このフィールドは DCB_TRAFFIC_PERIOD に初期化されます。その後、各システム・クロック（DMA ストールの場合を除く）で DCB_TRAFFIC_PERIOD から 0 までカウント・ダウンします。このカウントが非ゼロである間は、同じ方向の DCB アクセスは優先的に扱われます。このカウントが 1 から 0 までデクリメントすると、逆方向の DCB アクセスが優先的に扱われ、方向転換をもたらします。このカウントが 0 で、DCB バス・アクセスが発生すると、新しいバーストを始めるために、カウントは DCB_TRAFFIC_PERIOD から再ロードされます。

DMA トラフィック制御カウンタ周期レジスタ（DMA_TC_PER）

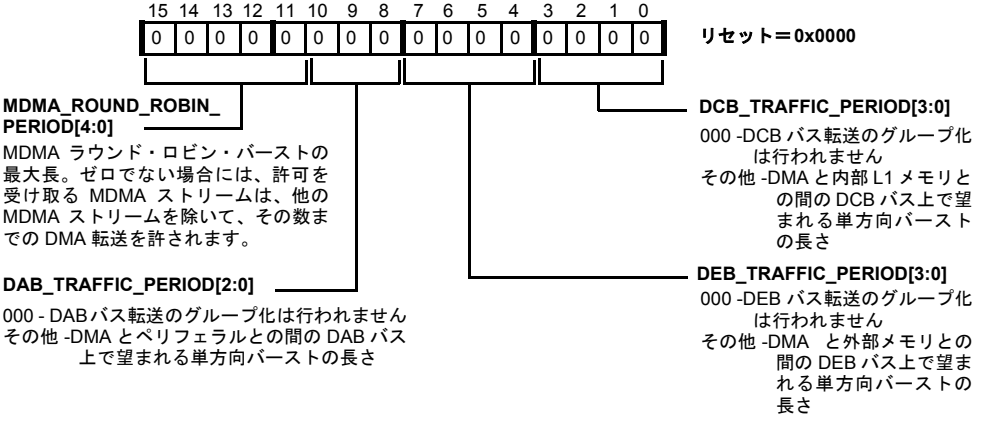


図 9-16. DMA トラフィック制御カウンタ周期レジスタ

DMA トラフィック制御カウンタ・レジスタ（DMA_TC_CNT）
RO

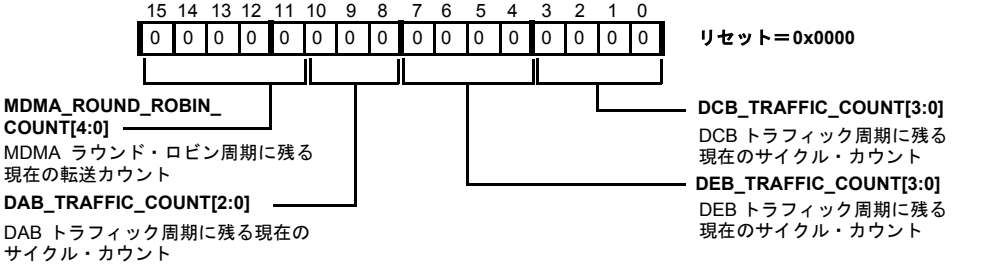


図 9-17. DMA トラフィック制御カウンタ・レジスタ

■ MDMA 優先順位とスケジューリング

すべてのMDMA動作は、ペリフェラルDMA動作よりも低い優先順位を持っています。したがって、MDMAはペリフェラルDMAトラフィックによって使用されないメモリ帯域幅を効果的に使用します。

2つのMDMAストリーム（S0-D0とS1-D1）が使用される場合には、ユーザは固定ストリーム優先順位またはラウンド・ロビン方式によって帯域幅を割り当てることができます。これは、DMA_TC_PERレジスタのMDMA_ROUND_ROBIN_PERIODフィールドをプログラミングすることで選択されます（9-55ページの「優先順位付けとトラフィック制御」を参照）。

このフィールドが0に設定された場合には、MDMAは固定優先順位によってスケジュールされます。ストリーム0での転送実行の準備が完了するたびに、MDMAストリーム0はMDMAストリーム1よりも優先順位が高くなります。一般に、MDMAストリームでは使用可能なすべてのサイクルでデータを転送できるため、ありとあらゆるMDMAストリーム0の動作が完了するまで、MDMAストリーム1のトラフィックは無期限に遅延させられることがあります。この方式は、低期間であっても遅延に敏感なデータ・バッファを長期間で低優先順位のバックグラウンド転送に割り込ませて、すぐに移動させる必要のあるシステムで適切な場合があります。

MDMA_ROUND_ROBIN_PERIODフィールドに1 <= P <= 31の範囲の非ゼロ値が設定されている場合には、ラウンド・ロビン・スケジューリング方式が使用されます。2つのMDMAストリームには、P回までのデータ転送のバーストで交互にバス・アクセスが許可されます。この方式は、2つの転送プロセスが共存する必要があり、それぞれが使用可能な帯域幅の保証された一部分を持つシステムで使用できます。たとえば、1つのストリームを内部—外部移動用にプログラムし、もう1つのストリームを外部—内部移動用にプログラムして、それぞれにほぼ等しいデータ帯域幅を割り当てることができます。

ラウンド・ロビン動作では、任意の時点でのMDMAストリーム選択は「フリー」または「ロック」です。最初、この選択はフリーです。MDMAが使用できるフリー・サイクルで（優先されるペリフェラルDMAアクセスがないときに）は、MDMAストリームの一方または両方がアクセスを要求した場合に優先順位の高いストリームが許可され（競合した場合にはストリーム0）、続いてそのストリームの選択が「ロック」されます。DMA_TC_CNTレジスタのMDMA_ROUND_ROBIN_COUNTカウンタ・フィールドにはMDMA_ROUND_ROBIN_PERIODから周期Pがロードされ、MDMA転送が始まります。カウンタは、データ転送のたび（各データ・ワードがメモリに書き込まれるとき）にデクリメントされます。カウント1に対応する転送の後で、このMDMAストリーム選択は、もう一方のストリームにゼロ・オーバーヘッドで自動的に渡されます。そして、MDMA_ROUND_ROBIN_COUNTカウンタには、MDMA_ROUND_ROBIN_PERIODからの周期値Pが再ロードされます。このサイクルでは、もう一方のMDMAストリームで転送実行の準備ができている場合には、そのストリーム選択は新しいMDMAストリームでロックされます。もう一方のMDMAストリームで転送実行の準備ができていない場合には、転送は実行されず、次のサイクルでストリーム選択はロック解除されて再びフリーになります。

1つのMDMAストリームだけがアクティブのときにラウンド・ロビン方式が使用された場合には、P回のMDMAデータ・サイクルごとに1回のアイドル・サイクルが発生し、帯域幅は1/(P+1)だけ若干低下します。しかし、両方のMDMAストリームが使用された場合には、メモリDMAは（たとえば、メモリに対する読出し／書込み方向の逆転に通常伴うオーバーヘッド・サイクル以外には）ストリームの交替のための追加オーバーヘッドなしで連続的に動作できます。MDMAストリームの交替頻度を制限するさまざまなラウンド・ロビン周期値Pを選択することで、最大の転送効率を維持できます。

■ 緊急の DMA 転送

一般に、特定ペリフェラルのDMA転送は一定の間隔で発生します。通常、この間隔が短いほど、そのペリフェラルに割り当てられる優先順位は高くなります。全ペリフェラルの平均帯域幅が合計帯域幅の大部分を占めない場合には、全ペリフェラルの要求は必要に応じて許可してください。

ときには、瞬間的なDMAトラフィックが使用可能な帯域幅を超えて、輻輳を引き起こすこともあります。このようなことが起きるのは、SDRAMページ・スワップやキャッシュ・ライン・フィルのためにL1または外部メモリが一時的にストールした場合です。ディスクリプタ・フェッチを行ったりDMAやペリフェラル内のFIFOをフィルするために、1つまたは複数のDMAチャンネルが多数の要求を開始した場合にも輻輳が発生することがあります。

輻輳が持続する場合には、低優先順位のDMAペリフェラルはデータ不足に陥るかもしれません。たとえペリフェラルの優先順位が低いにしても、ペリフェラルの一定の間隔が終了するまでに必要なデータ転送が行われない場合には、システム障害につながることもあります。この可能性を最小限に抑えるため、DMAユニットは、データ不足が深刻なペリフェラルを検出し、最高の優先順位で優先的にサービスを許可します。

DMAチャンネルのメモリ・サービス要求は、次の2つの条件が満足された場合に「緊急」であると定義されます。

- チャンネルのFIFOで、DABバス転送の準備ができていない（つまり、送信FIFOが空であるか、受信FIFOが満杯）。
- ペリフェラルは、そのDMA要求ラインをアサートしている。

ディスクリプタ・フェッチは、データ切れのペリフェラルに対するDMA作業単位チェーンを開始または続行する必要がある場合に緊急であるといえます。MDMAチャンネルからのDMA要求は緊急ではありません。

1つまたは複数のDMAチャンネルが緊急のメモリ要求を表明すると、2つのイベントが発生します。

- 緊急でないすべてのメモリ要求は、優先順位が 32 だけ減らされることにより、緊急の要求だけが許可されることが保証されます。緊急の要求が複数存在し、それらの間で方向の優先権が観測される場合には、緊急の要求は互いに競合します。
- 結果として生じるメモリ転送は、ターゲットとするメモリ・システム（L1または外部）で優先処理とマークされます。そのメモリ・システム内で先行する、以前のすべての不完全なメモリ転送も同様です。したがって、ペリフェラルの緊急の要求に対処できるように、一連の外部メモリ・コア・アクセスには数サイクルの遅延が生じることもあります。

緊急のDMA転送の優先的な処理は、完全に自動化されています。この機能のために、ユーザ制御は必要ありません。

DMA のソフトウェア管理

DMAとMDMAを管理するソフトウェア・タスクの開発には、いくつかの同期方式と制御方式を使用できます（9-50ページの「メモリDMA」も参照）。このようなソフトウェアには、他のソフトウェア・タスクから新しいDMA転送の要求を受け付け、このような転送を既存の転送キューに統合し、転送が完了すると他のタスクに確実に通知できる機能が要求されます。

プロセッサでは、各DMAペリフェラルとMDMAストリームを別個のタスクによって管理したり、他のストリームと共に管理したりできます。各DMAチャンネルには、直交性のある独立したコントロール・レジスタ、リソース、割込みがあるため、1つのチャンネルに対する制御方式の選択は、他のチャンネルでの制御方式の選択に影響を与えません。たとえば、1つのペリフェラルではリンクト・ディスクリプタ・リストの割込み駆動

方式を使用し、別のペリフェラルでは要求駆動型の“一度にバッファ”方式を同時に使用して、IRQ_STATUS レジスタのポーリングによって同期をとることができます。

■ ソフトウェアと DMA の同期

ソフトウェアDMA管理の重要な要素は、DMAバッファの完了とソフトウェアとの同期です。そのために最も良い方法は、割込み、IRQ_STATUSのポーリング、またはその両方の組合わせです。アドレスやカウントのポーリングでは、パイプライン長に匹敵する緩い許容誤差内での同期のみを提供できます。

割込みベースの同期方式では、割込みのオーバーラン、つまり割込み処理における過度の遅延によって、割込みイベントごとにDMAチャンネルの割込みハンドラを呼出しできない状態を回避する必要があります。一般に、システム設計では、チャンネルごとにただ1つの割込みがスケジュールされる（たとえば、ディスクリプタ・リストの最後で）か、または十分な割込み時間間隔を設定してすべての割込みが処理されることを保証する必要があります。なお、すべての割込みチャンネルには専用の独立した割込みがあるため、さまざまなペリフェラルの割込み間での相互作用は簡単に管理できます。

レジスタCURR_ADDR、CURR_DESC_PTR、CURR_X/Y_COUNTのポーリングは、DMAとデータ処理の同期を正しくとるための方法としては推奨されません。その理由は、DMA FIFOとDMA／メモリのパイプライン処理によります。カレント・アドレス・レジスタ、ポインタ・レジスタ、カウント・レジスタでは、対応するメモリ操作の完了に先立っていくつかのサイクルを変更します。このことは、メモリ読出し／書込み命令によって、動作の結果が最初にコアから見える時間として測定されます。たとえば、外部メモリへのDMAメモリ書込み動作では、チャンネルAによるDMA書込みが開始されて、SDRAMは、多くのシステム・クロック・サイクルを費やすページ・オープン動作を実行すると想定します。その後、DMAエンジンは、チャンネルBによる別のDMA動作に進むことができます。このDMA

動作は、それ自体では遅延を発生しませんが、チャンネルAによる遅い動作の背後にストールされます。チャンネルBを監視しているソフトウェアは、CURR_ADDRレジスタの内容を検査しても、チャンネルBのCURR_ADDRによってポイントされるメモリ位置が書き込まれたかどうかを確実に判断できません。

しかし、DMA／メモリ・パイプラインの長さを考慮した場合、カレント・アドレス・レジスタ、ポインタ・レジスタ、カウント・レジスタのポーリングでは、ソフトウェアによるDMAの緩い同期を認めることができます。ペリフェラルDMAチャンネルのDMA FIFOの長さは4つのロケーション（4つの8/16ビット・データ・エレメント、または2つの32ビット・データ・エレメント）であり、MDMA FIFOの場合は8つのロケーション（4つの32ビット・データ・エレメント）です。これらのFIFOが不完全な作業（開始されていても、まだ完了していない読出しを含む）で満たされている場合には、DMAはカレント・アドレス／ポインタ／カウント・レジスタを進めません。

さらに、内部メモリへの結合されたDMAとL1パイプラインの長さは、およそ6つの8/16ビット・データ・エレメントです。DMAと外部バス・インターフェース・ユニット（EBIU）パイプラインの長さは、MMR読出しからDMAレジスタの更新が見えるポイントから、メモリへのDMAアクセスとコア・アクセスが厳密に順序付けされるポイントまで測定した場合に、およそ3つのデータ・エレメントです。DMA FIFO長とDMA／メモリ・パイプライン長が加算された場合には、同時に実行中の不完全なメモリ操作の最大数を見積もることができます。（なお、DMA／メモリ・パイプラインには他のDMAチャンネルからのトラフィックが含まれることもあるため、これは最大値です）。

たとえば、ペリフェラルDMAチャンネルが100データ・エレメントの作業単位を内部メモリに転送しており、そのCURR_X_COUNTレジスタが60個の残りエレメントを示すため、少なくとも最初の40エレメントの処理は開始されていると想定します。パイプライン長の合計は4（PDMA FIFO）

＋6（DMA／メモリ・パイプライン）の合計、つまり10個のデータ・エレメントを超えません。したがって、最初の40－10＝30データ・エレメントのDMA転送は完了したと判断して安全です。

正確な同期を実現するには、ソフトウェアはカレント・アドレス／ポインタ／カウント・レジスタをポーリングするのではなく、割込みを待つかチャンネルのIRQ_STATUSレジスタを調べて、DMAの完了を確認してください。DMAシステムが割込みを発行したり、IRQ_STATUSビットを変更したりした場合には、作業単位の最後のメモリ操作が完了しており、DSPコードから確実に見えることが保証されます。メモリ読出しDMAでは、最終のメモリ読出しデータはDMAのFIFOに安全に受信されています。メモリ書込みDMAでは、DMAユニットはデータが書き込まれたEBIUやL1メモリからのアクノレッジを受信しています。

以下の例では、ソフトウェアとさまざまなスタイルのDMAとの同期をとる方法を示します。

▶ シングル・バッファ DMA 転送

ペリフェラルのDMA動作がシングル・バッファの孤立した転送から構成される場合には、同期は簡単です。DMA動作は、チャンネルのコントロール・レジスタへのソフトウェア書込みによって開始されます。ユーザは、メモリ内のシングル・ディスクリプタを使用するよう選択できます。その場合、ソフトウェアはDMA_CONFIGレジスタとNEXT_DESC_PTRレジスタにだけ書き込む必要があります。あるいは、ユーザはソフトウェアからすべてのMMRレジスタに直接書き込んで、DMA_CONFIGレジスタへの書込みで終わるようにも選択できます。

DMAの完了を通知する最も簡単な方法は、割込みです。これは、DMA_CONFIGレジスタのDI_ENビットと、システム割込みコントローラの必要なセットアップによって選択されます。割込みを使用しないほうが望ましい場合には、ソフトウェアはIRQ_STATUSレジスタを読み出し、DMA_RUNビットをテストすることによって完了をポーリングできます。このビットがゼロの場合、バッファ転送は完了しています。

▶ 自動バッファリングによる連続転送

ペリフェラルのDMAデータが信号データの安定した周期的なストリームから構成される場合には、DMA自動バッファリング（FLOW = 1）が有効なオプションであることもあります。ここでは、ループに対してプロセッサとDMAのオーバーヘッドがない1次元または2次元のインデックス付けを使用して、メモリ・バッファは循環アドレッシング方式でDMAを送／受信します。同期オプションには、以下のものが含まれます。

- 1D、割込み駆動：ソフトウェアは各バッファの終わりに割込みされます。重要な設計ポイントとして、ソフトウェアは、次のDMA転送の前にバッファ内の最初の項目を処理しなければならないことです。次のDMA転送では、最初のバッファ位置はソフトウェアによって処理される前に上書きされたり再読出しされることがあります。システム設計によって、あらゆる状況でデータ・リピート周期が割込み遅延よりも長いことが保証される場合には、この方式を使用できることがあります。

- 2D、割込み駆動（ダブル・バッファリング）：DMA バッファは複数のサブバッファに分割され、割込みは各DMA内側ループの完了時に通知されるよう選択されます（DMA_CONFIGでDI_SEL = 1に設定）。このようにして、伝統的なダブル・バッファまたは「ピンポン」方式を実装できます。

  たとえば、次に示す設定によって、1Kワード・バッファ内の2つの512ワード・サブバッファを使用して、16ビットのペリフェラル・データを受信できます。

  START_ADDR = バッファのベース・アドレス
  DMA_CONFIG = 0x10D7 (FLOW = 1, DI_EN = 1, DI_SEL = 1, DMA2D = 1, WDSIZE = 01, WNR = 1, DMA_EN = 1)
  X_COUNT = 512
  X_MODIFY = 2、16 ビット・データの場合

Y_COUNT = 2、2 つのサブバッファの場合
Y_MODIFY = 2、連続的なサブバッファの場合は X_MODIFY と同じ

- 2D、ポーリング : 割込みオーバーヘッドが受け入れがたくても、アドレス／カウント・レジスタ・ポーリングの緩い同期が許容される場合には、2Dマルチバッファ同期方式を使用できます。たとえば、受信データは16個の32ビット・エレメントのパケットで処理する必要があるとします。次に示す設定によって、4つの部分から成る2D DMAバッファを割り当て、4つのサブバッファがそれぞれ1つのパケットを保持することができます。

  START_ADDR = バッファのベース・アドレス
  DMA_CONFIG = 0x101B (FLOW = 1, DI_EN = 0, DMA2D = 1, WDSIZE = 10, WNR = 1, DMA_EN = 1)
  X_COUNT = 16
  X_MODIFY = 4、32 ビット・データの場合
  Y_COUNT = 4、4 つのサブバッファの場合
  Y_MODIFY = 4、連続的なサブバッファの場合は X_MODIFY と同じ

  同期コアでは、Y_COUNT を読み出して現在転送中のサブバッファを判定し、1 つのフル・サブバッファにパイプライン処理を担当させることができます。たとえば、Y_COUNT の値が 3 である場合には、サブバッファ 3 が転送中であると想定できますが、サブバッファ 2 の一部はまだ受信されていないかもしれません。しかし、ソフトウェアは、サブバッファ 1 または 0 の処理を安全に進めることができます。

- 1D非同期FIFO : システムの設計によって、ペリフェラルのデータの処理とデータのDMAレートが定常状態で相互に関連していることが保証され、しかも短期の遅延変動を許容する必要がある場合には、簡単なFIFOを構築することが適切かもしれません。ここでは、DMAチャンネルは割込みもポーリングもなしで、1D自動バッファ・モード・アドレッシングを使用してプログラムできます。

▶ ディスクリプタ構造

簡単な1Dアレイや2Dアレイでないメモリ・データ構造では、DMAディスクリプタを使用してデータを送／受信できます。たとえば、1パケットのデータをメモリ内の複数の異なる位置から送信する場合（1つの位置からのヘッダ、メモリ・プール・アロケータによって管理される数ブロックのメモリのリストからのペイロード、チェックサムを含んでいる小さなトレーラ）、メモリ領域ごとに別個のDMAディスクリプタを準備して、DMA_CONFIGでの適切なFLOW設定の選択に基づいて、ディスクリプタをアレイ単位またはリスト単位でグループ化できます。

ソフトウェアは、1つまたは複数のディスクリプタに対する割込み通知を選択することによって、構造の転送の進行と同期をとることができます。たとえば、ソフトウェアはヘッダのディスクリプタとトレーラのディスクリプタに対する割込み通知を選択できますが、ペイロード・ブロックのディスクリプタに対しては選択できません。

DMAディスクリプタのリストやアレイを構築する際には、DMA_CONFIGディスクリプタ・エレメント内のさまざまなフィールドの意味を再確認する必要があります。特に、以下のことに注意してください。

- DMA_CONFIG の下位バイトでは、**現在の**ディスクリプタによって実行されるDMA転送を指定します（たとえば、割込みイネーブル、2Dモード）。
- DMA_CONFIG の上位バイトでは、チェーン内の**次の**ディスクリプタのフォーマットを指定します。特定のディスクリプタのNDSIZEフィールドとFLOWフィールドは、ディスクリプタ自身のフォーマットには対応せず、次のディスクリプタへのリンクを指定します（ある場合）。

一方、DMAユニットが再起動されているときには、DMAチャンネルのDMA_CONFIGレジスタに書き込まれたDMA_CONFIG値の2つのバイトはカレント・ディスクリプタに対応するはずです。最小限、FLOW、NDSIZE、WNR、DMA_ENの各フィールドは、すべてカレント・ディスクリプタと一致する必

要があります。WDSIZE、DI_EN、DI_SEL、RESTART、DMA2Dの各フィールドは、メモリから読み出されたディスクリプタのDMA_CONFIG値から取得されます（そして、レジスタに最初に書き込まれたフィールド値は無視されます）。

▶ ディスクリプタ・キューの管理

システム設計者は、他のソフトウェアからDMA要求を受け付けるDMAマネージャ機能を作成することもできます。DMAマネージャ・ソフトウェアは、新しい作業要求が受信されるタイミングやそれらの要求の内容を前もって知ることはできません。ソフトウェアは、DMAディスクリプタの循環リンク・リストを使用して、これらの転送を管理できます。なお、各ディスクリプタのNDPHメンバーとNDPLメンバーは次のディスクリプタを指し示し、最後のディスクリプタは先頭を指し示します。

このディスクリプタ・リストに書き込むコードでは、プロセッサの循環アドレッシング・モードを使用できます（I、L、M、Bレジスタ）。したがって、循環構造を管理するために、比較命令や条件付き命令を使用する必要はありません。この場合、各ディスクリプタのNDPHメンバーとNDPLメンバーは起動時に1回書き込んで、各ディスクリプタの新しい内容が書き込まれるときには飛ばすこともできます。

ディスクリプタ・キューの同期をとる方法としては、割込みの使用をお勧めします。ディスクリプタ・キューは、少なくとも最後の有効なディスクリプタでは常に割込みを生成するようにできています。

割込みを使用してディスクリプタ・キューを管理するには、2つの一般的な方法があります。

- 全ディスクリプタで割込みを生成
- 最小の割込み：最後のディスクリプタでのみ生成

全ディスクリプタでの割込みを使用するディスクリプタ・キュー

このシステムでは、DMAマネージャ・ソフトウェアは、すべてのディスクリプタでの割込みをイネーブルにすることによって、DMAユニットとの同期をとります。この方式を使用するのは、システム設計によって各割込みイベントが（割込みオーバーランなしで）別々に処理されることを保証できる場合に限ってください。

ディスクリプタ・キューの同期を維持するため、非割込みソフトウェアはキューに追加されたディスクリプタのカウントを保持しますが、割込みハンドラは、キューから取り除かれた完了済みディスクリプタのカウントを保持します。これらのカウントが等しくなるのは、DMAチャンネルがすべてのディスクリプタを処理した後で休止したときだけです。

新しい作業要求が受信されるたびに、DMAマネージャ・ソフトウェアは新しいディスクリプタを初期化して、FLOW値0でDMA_CONFIG値を書き込みます。次に、このソフトウェアはディスクリプタ・カウントを比較して、DMAチャンネルが実行中かどうかを判定します。DMAチャンネルが休止中（カウントが等しい）の場合には、ソフトウェアはそのカウントをインクリメントしてから、新しいディスクリプタのDMA_CONFIG値をDMAチャンネルのDMA_CONFIGレジスタに書き込むことによってDMAユニットを開始します。

カウントが等しくない場合には、ソフトウェアは最後から2番目のディスクリプタのDMA_CONFIG値について、その上半分（FLOWとNDSIZE）が新たにキュー登録されたディスクリプタを記述するように変更します。残りのディスクリプタ・データ構造が前もって初期化されている場合には、この操作はDMAチャンネルを混乱させません。しかし、DMAチャンネルによってDMA_CONFIGの新しい値が読み出されたか古い値が読み出されたかを正しく判定するには、ソフトウェアとDMAの同期をとることが必要です。

この同期動作は、割込みハンドラで実行してください。まず、割込みと同時に、ハンドラはチャンネルのIRQ_STATUSレジスタを読み出してください。DMA_RUNステータス・ビットがセットされている場合には、そのチャンネルは別のディスクリプタの処理に進んでおり、割込みハンドラはそのカウントをインクリメントして終了できます。しかし、DMA_RUNステータス・ビットがセットされていない場合には、処理すべきディスクリプタがなくなったか、最後のディスクリプタのキュー登録が遅すぎた（つまり、最後から2番目のディスクリプタのDMA_CONFIGエレメントの変更は、そのエレメントがDMAユニットに読み出された後で行われた）ために、チャンネルは休止しています。この場合、割込みハンドラは、最後のディスクリプタに適切なDMA_CONFIG値をDMAチャンネルのDMA_CONFIGレジスタに書き込み、完了したディスクリプタ・カウントをインクリメントし、終了してください。

システムの割込み遅延が大きく、チャンネルのいずれかのDMA割込みがドロップされる場合にも、やはり、このシステムは機能しなくなることがあります。複数のディスクリプタの割込みを確実に同期させることのできる割込みハンドラは複雑になる必要があり、信頼性の高い動作を保証するために複数のMMRアクセスを実行します。このようなシステム環境では、最小割込み同期方式が好まれます。

最小の割込みを使用するディスクリプタ・キュー

このシステムでは、任意の時点でキュー内で可能なDMA割込みイベントは1つだけです。このシステム用のDMA割込みハンドラも、きわめて短くできます。ここではディスクリプタ・キューは「アクティブ」部分と「待機」部分にまとめられ、割込みは、各部分の最後のディスクリプタでのみイネーブルにされます。

新しいDMA要求が処理されるたびに、ソフトウェアの非割込みコードは新しいディスクリプタの内容をフィルして、それをキューの待機部分に追加します。ディスクリプタのDMA_CONFIGワードでは、FLOW値がゼロになっているはずです。DMAキューの完了割込みが発生する前に複数の要求が受信された場合には、非割込みコードは後のディスクリプタをキューに登

録します。これにより、キューの待機部分は、DMAユニットによって処理されているキューのアクティブ部分から切断されます。つまり、最後を除くすべてのアクティブ・ディスクリプタにはFLOW値＞＝4が含まれ、割込みイネーブルは設定されていません。一方、最後のアクティブ・ディスクリプタには0のFLOWが含まれ、割込みイネーブル・ビットDI_ENが1に設定されています。また、最後を除くすべての待機ディスクリプタにはFLOW値＞＝4が含まれ、割込みイネーブルは設定されていません。一方、最後の待機ディスクリプタには0のFLOWが含まれ、割込みイネーブル・ビットは1に設定されています。これによって、DMAユニットは、アクティブ・キュー全体を自動的に処理してから、1つの割込みを発行できます。また、この配置では、1回のDMA_CONFIGレジスタ書込みによって、割込みハンドラ内の待機キューを簡単に起動できます。

新しい待機ディスクリプタをキューに登録した後、非割込みソフトウェアは、待機キュー内の最初の待機ディスクリプタの起動に使用する希望のDMA_CONFIG値（または、待機中のディスクリプタがないことを示す0）を含んでいるメモリ・メールボックス位置に、その割込みハンドラ向けのメッセージを残すはずです。

綿密な同期対策が講じられる場合を除いて、DMAユニットによるアクティブ・ディスクリプタ・キューの処理が開始された後では、ソフトウェアはアクティブ・ディスクリプタ・キューの内容を直接変更しないことが大切です。ディスクリプタ・キューの最も簡単な実装形態では、DMAマネージャ・ソフトウェアはアクティブ・キュー上のディスクリプタを変更しません。その代わりに、DMAマネージャはDMAキュー完了割込みによって、アクティブ・キュー全体の処理が完了したことを知らされるまで待ちます。

DMAキュー完了割込みが受信されると、割込みハンドラは非割込みソフトウェアからメールボックスを読み出し、その中の値をDMAチャンネルのDMA_CONFIGレジスタに書き込みます。この1回のレジスタ書込みによってキューが再起動され、待機キューは実質的にアクティブ・キューに変換されます。その後、割込みハンドラは非割込みソフトウェアにメッセージ

を戻して、アクティブ・キューに受け付けられた最後のディスクリプタの位置を知らせます。一方で、割込みハンドラがそのメールボックスを読み出し、実行すべき作業がないことを示すゼロのDMA_CONFIG値を見つけた場合には、非割込みソフトウェアに適切なメッセージ（たとえば、ゼロ）を戻して、キューが停止したことを知らせます。この簡単なハンドラは、ごくわずかな命令数で記述できるはずです。

新しいDMA作業要求を受け付ける非割込みソフトウェアでは、新しい作業の起動と割込みハンドラの同期をとる必要があります。キューが停止した場合（つまり、割込みソフトウェアからのメールボックスがゼロの場合）には、非割込みソフトウェアがキューの起動を担当します（最初のディスクリプタのDMA_CONFIG値を、チャンネルのDMA_CONFIGレジスタに書き込みます）。しかし、キューが停止していない場合には、非割込みソフトウェアはDMA_CONFIGレジスタに書き込んではいけません（DMAエラーが発生します）。その代わりに、ディスクリプタを待機キューに登録して、割込みハンドラに割り当てられたメールボックスを更新してください。

DMA エラー（アボート）

DMAコントローラは、DMAプロセスの異常終了（つまり、アボート）を引き起こす条件にフラグを立てます。この機能は、DMA関連のプログラミング・エラーを検出する方法の1つとして、システム開発とデバッグ用のツールとして提供されます。DMAエラー（アボート）は、下記の場合にDMAチャンネル・モジュールによって検出されます。DMAエラーが発生すると、チャンネルはすぐに停止して（DMA_RUNが0になります）、プリフェッチされたデータは廃棄されます。さらに、DMA_ERROR割込みがアサートされます。

DMAコントローラ全体に対してただ1つのDMA_ERROR割込みがあり、いずれかのチャンネルがエラー条件を検出するたびにアサートされます。

DMA_ERROR割込みハンドラは、チャンネルごとに以下の動作を行う必要があります。

- 各チャンネルのIRQ_STATUSレジスタを読み出して、DMA_ERRビットがセット（ビット1）されているチャンネルを探します。
- そのチャンネルでの問題を解決します（たとえば、レジスタ値を修正します）。
- DMA_ERRビットをクリアします（IRQ_STATUSにビット1＝1を書き込みます）。

以下のエラー条件は、DMAハードウェアによって検出され、DMAアボート割込みにつながります。

- 設定レジスタに無効な値が含まれています。

  - 不正なWDSIZE値（WDSIZE = b#11）
  - ビット15が0に設定されていない
  - 不正なFLOW値（FLOW = 2、3、5）
  - NDSIZE値がFLOWと一致しない。9-77 ページの表 9-20 を参照。

- チャンネルの実行中に、レジスタ書込みの拒否が発生しました。DMA_RUN = 1のとき、書込みできるレジスタはDMA_CONFIGとIRQ_STATUSだけです。
- メモリ・アクセス中にアドレス・アライメント・エラーが発生しました。たとえば、DMA_CONFIGレジスタでWDSIZE = 1（16ビット）であっても、アドレスの最下位ビット（LSB）が0と等しくありません。あるいは、WDSIZE = 2（32ビット）であっても、アドレスの2つのLSBが00と等しくありません。
- メモリ空間の遷移が試みられました（外部—内部、またはその逆）。たとえば、現在のDMAアドレス（CURR_ADDR）が0xF000 0000の境界を越えました。あるいは、カレント・ディスクリプタ・ポインタ（CURR_DESC_PTR）が0xF000 0000の境界を越えました。

- メモリ・アクセス・エラーが発生しました。未実装の内部アドレスやキャッシュとして定義された内部アドレスに対してアクセスが試みられたか、または外部アクセスによってエラーが発生しました（外部メモリ・インターフェースによって通知されます）。

DMAハードウェアによって検出されない禁止状態もあります。このような状態に対しては、DMAアボートは通知されません。

- DMA_CONFIG方向ビット（WNR）が、マッピングされたペリフェラルの方向と一致しません。
- DMA_CONFIG方向ビットが、MDMAチャンネルの方向と一致しません。
- DMA_CONFIG ワード・サイズ（WDSIZE）は、マッピングされたペリフェラルによってサポートされていません。
- MDMAストリームのソースとデスティネーションでのDMA_CONFIGワード・サイズが等しくありません。
- ディスクリプタ・チェーンは、同じ内部／外部メモリ空間内にないデータ・バッファを示します。
- 2D DMAで、X_COUNT = 1です。

表 9-20.　正当な NDSIZE 値

FLOW	NDSIZE	注
0	0	
1	0	
4	0 < NDSIZE <= 7	ディスクリプタ・アレイ、ディスクリプタ・ポインタのフェッチなし
6	0 < NDSIZE <= 8	ディスクリプタ・リスト、スモール・ディスクリプタ・ポインタをフェッチ
7	0 < NDSIZE <= 9	ディスクリプタ・リスト、ラージ・ディスクリプタ・ポインタをフェッチ

第10章　SPI互換ポート・コントローラ

このプロセッサのシリアル・ペリフェラル・インターフェース（SPI）ポートは、SPI互換の多種多様なペリフェラル・デバイスにI/Oインターフェースを提供します。

SPIポートは、設定可能な一連のオプションによって、他のSPI互換デバイスとのグルーレスなハードウェア・インターフェースを提供します。SPIは、2本のデータ・ピン、デバイス・セレクト・ピン、クロック・ピンから構成される、4線式インターフェースです。SPIは全二重の同期シリアル・インターフェースであり、マスター・モード、スレーブ・モード、マルチマスター環境をサポートします。SPI互換のペリフェラル・システムは、プログラマブルなボーレートとクロック位相／極性にも対応します。SPIは、マルチマスター・シナリオに対応してデータ競合を回避するために、オープン・ドレイン・ドライバを使用します。

SPI互換インターフェースへのインターフェースに使用できる、SPI互換の代表的なペリフェラル・デバイスには以下のものが含まれます。

- 他のCPUまたはマイクロコントローラ
- コーデック
- A/Dコンバータ
- D/Aコンバータ
- サンプル・レート・コンバータ
- SP/DIFまたはAES/EBUデジタル・オーディオ送受信機

- LCDディスプレイ
- シフト・レジスタ
- SPIエミュレーション付きのFPGA

SPIは、複数のSPI互換デバイスとの通信を可能にする業界標準の同期シリアル・リンクです。SPIペリフェラルは、2本のデータ・ピン（MOSIとMISO）、1本のデバイス・セレクト・ピン（$\overline{\text{SPISS}}$）、ゲーテッド・クロック・ピン（SCK）から構成される、同期型の4線式インターフェースです。2本のデータ・ピンによって、他のSPI互換デバイスへの全二重動作が可能になります。SPIには、プログラマブルなボーレート、クロック位相、クロック極性も含まれます。

SPIは、他のデバイスとインターフェースをとることによってマルチマスター環境で動作でき、マスター・デバイスまたはスレーブ・デバイスとして機能します。マルチマスター環境では、SPIインターフェースは、データ・バス競合を回避するためにオープン・ドレイン出力を使用します。

図 10-1はSPIのブロック図を示します。このインターフェースは基本的にはシフト・レジスタであり、他のSPIデバイスとの間で、SCKレートで一度に1ビットずつ、データビットを連続的に送／受信します。SPIデータは、シフト・レジスタを使用して、同時に送／受信されます。SPI転送が行われるとき、データが送信される（シフト・レジスタから連続的にシフト・アウトされます）と同時に、新しいデータが受信されます（同じシフト・レジスタの他端に連続的にシフト・インされます）。SCKは、2本のシリアル・データ・ピンでデータのシフト操作とサンプリングの同期をとります。

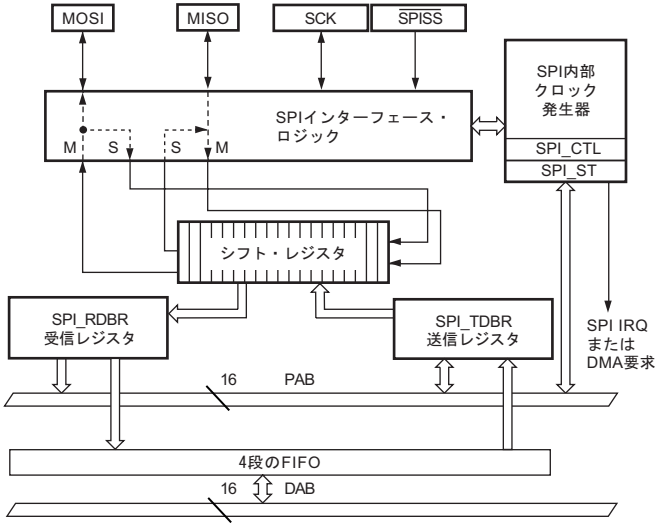


図 10-1. SPI ブロック図

SPIデータ転送中に、1つのSPIデバイスはSPIリンク・マスターとして機能します。つまり、SPIシリアル・クロックを生成し、SPIデバイス選択信号（SPISS）をアサートすることによって、データ・フローを制御します。もう1つのSPIデバイスはスレーブとして機能し、マスターからの新しいデータをそのシフト・レジスタに受け付けると同時に、そのSPI送信データ・ピンを通じて、要求されたデータをシフト・レジスタから送信します。他のマイクロコントローラやマイクロプロセッサと同様に、複数のプロセッサが交替でマスター・デバイスになることができます。1つのマスター・デバイスは、複数のスレーブに同時にデータをシフトすることもできます（ブロードキャスト・モードと呼ばれます）。しかし、任意の時点で出力を駆動してマスターにデータを書き戻せるのは1つのスレーブだけです。ブロードキャスト・モードは必ず守らねばなりません。このモードでは、マスターからのデータを受信するよう複数のスレーブを選択できますが、データをマスターに送り返すようにできるのは、一度に1つのスレーブだけです。

複数のプロセッサがそのSPIポートを介して接続されるマルチマスター環境やマルチデバイス環境では、すべてのMOSIピン、MISOピン、SCKピンが、それぞれ共に接続されています。

マルチスレーブ環境の場合、プロセッサは、SPIスレーブ・デバイスに対する専用のSPIスレーブ・セレクト信号である7つのプログラマブル・フラグPF1～PF7を利用できます。

ⓘ リセット時に、SPIはディスエーブルにされ、スレーブとして設定されます。

インターフェース信号

以下の節では、SPI信号について説明します。

■ シリアル・ペリフェラル・インターフェースのクロック信号（SCK）

SCK信号はSPIクロック信号です。この制御信号は、マスターによって駆動され、データの転送レートを制御します。マスターは、さまざまなボーレートでデータを送信できます。SCK信号は、送信されるビットごとに1回循環します。これは、デバイスがマスターとして設定されている場合には出力信号であり、デバイスがスレーブとして設定されている場合には入力信号です。

SCKはゲーテッド・クロックであり、データ転送中には転送されるワードの長さの間だけアクティブです。アクティブ・クロック・エッジの数は、データ・ラインで駆動されたビットの数と同じです。シリアル・ペリフェラル・スレーブ・セレクト入力（$\overline{\text{SPISS}}$）が非アクティブ（ハイレベル）に駆動された場合には、スレーブ・デバイスはシリアル・クロックを無視します。

SCKは、MISOラインとMOSIラインで駆動されたデータをシフト・アウト／シフト・インするために使用されます。データは、クロックのアクティブ・エッジで常にシフト・アウトされ、クロックの非アクティブ・エッジでサンプリングされます。データを基準にしたクロック極性とクロック位相は、SPIコントロール・レジスタ(SPI_CTL)で設定可能であり、転送フォーマットを定義します。

■ シリアル・ペリフェラル・インターフェースのスレーブ・セレクト入力信号

$\overline{\text{SPISS}}$信号は、SPIシリアル・ペリフェラル・スレーブ・セレクト入力信号です。これはアクティブ・ロー信号であり、プロセッサがスレーブ・デバイスとして設定されたとき、プロセッサをイネーブルにするために使用されます。この入力専用ピンは、チップ・セレクトのように振る舞い、スレーブ・デバイス用にマスター・デバイスによって提供されます。マスター・デバイスの場合には、このピンは、マルチマスター環境でのエラー信号入力として機能できます。マルチマスター・モードで、マスターの$\overline{\text{SPISS}}$入力信号がアサート（ローレベルに駆動）され、SPI_CTLレジスタのPSSEビットがイネーブルにされた場合には、エラーが発生しています。これは、他のデバイスもマスター・デバイスになろうとしていることを意味します。

$\overline{\text{SPISS}}$信号は、PF0ピンと同じピンです。

■ マスター出力／スレーブ入力（MOSI）

MOSIピンは、マスター出力／スレーブ入力ピンであり、双方向I/Oデータ・ピンの1つです。プロセッサがマスターとして設定されている場合、MOSIピンはデータ送信（出力）ピンになり、出力データを送信します。プロセッサがスレーブとして設定されている場合、MOSIピンはデータ受信（入力）ピンになり、入力データを受信します。SPI相互接続では、データは、マスターのMOSI出力ピンからシフト・アウトされ、スレーブのMOSI入力にシフト・インされます。

■ マスター入力／スレーブ出力（MISO）

MISOピンは、マスター入力／スレーブ出力ピンであり、双方向I/Oデータ・ピンの1つです。プロセッサがマスターとして設定されている場合、MISOピンはデータ受信（入力）ピンになり、入力データを受信します。プロセッサがスレーブとして設定されている場合、MISOピンはデータ送信（出力）ピンになり、出力データを送信します。SPI相互接続では、データは、スレーブのMISO出力ピンからシフト・アウトされ、マスターのMISO入力ピンにシフト・インされます。

任意の時点でデータの送信を許されるのは、1つのスレーブだけです。

図 10-2のSPI設定例では、プロセッサをスレーブSPIデバイスとして使用する方法を示します。8ビットのホスト・マイクロコントローラはSPIマスターです。

プロセッサはその SPI インターフェースを介してブートできるため、ユーザのアプリケーション・コードとデータは、実行の前にダウンロードできます。

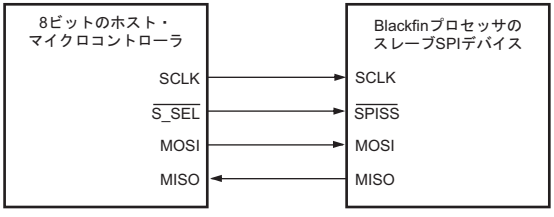


図 10-2. スレーブ SPI デバイスとしての ADSP-BF533

■ 割込み出力

SPIには、データ割込みとエラー割込みという、2つの割込み出力信号があります。

SPIデータ割込み信号の振る舞いは、SPIコントロール・レジスタの転送開始モード・ビット・フィールド（TIMOD）に依存します。DMAモード（TIMOD = 1X）では、データ割込みはDMA要求として機能し、DMA FIFOで書込み（TIMOD = 11）または読出し（TIMOD = 10）の準備ができたときに生成されます。非DMAモード（TIMOD = 0X）では、データ割込みが生成されるのは、SPI_TDBRで書込み（TIMOD = 01）の準備ができたとき、またはSPI_RDBRで読出し（TIMOD = 00）の準備ができたときです。

DMAモードと非DMAモードのいずれでも、モード・フォルト・エラーが発生すると、マスターでSPIエラー割込みが生成されます。DMAモードでは、アンダーフロー（TIMOD = 11の場合はTIMOD = 11）またはオーバーフロー（TIMOD = 10の場合はRBSY）のエラー条件があるときにも、エラー割込みを生成できます。非DMAモードでは、アンダーフロー条件とオーバーフロー条件は、SPI_STATレジスタのそれぞれTXEビットとRBSYビットをセットしますが、エラー割込みを生成しません。

この割込み出力の詳細については、10-9ページの「SPI_CTLレジスタ」のTIMODビットの説明を参照してください。

SPI レジスタ

SPIペリフェラルには、ユーザ・アクセス可能な多くのレジスタが含まれています。これらのレジスタのいくつかは、DMAバスを通してもアクセスできます。4本のレジスタには制御情報とステータス情報が含まれています（SPI_BAUD、SPI_CTL、SPI_FLG、SPI_STAT）。2本のレジスタは送／受信データのバッファリングに使用されます（SPI_RDBRとSPI_TDBR）。DMA

関連のレジスタの詳細については、第9章「ダイレクト・メモリ・アクセス」を参照してください。シフト・レジスタSFDRはSPIモジュールの内部にあり、直接にアクセスできません。

これらのレジスタのビットを使用して、エラーやその他の条件を通知する方法については、10-30ページの「エラー信号とフラグ」を参照してください。SPIレジスタとビット機能の詳細については、10-21ページの「レジスタ機能」を参照してください。

■ SPI_BAUD レジスタ

SPIボーレート・レジスタ（SPI_BAUD）は、マスター・デバイスのビット転送速度の設定に使用されます。スレーブとして設定されたとき、このレジスタに書き込まれた値は無視されます。シリアル・クロック周波数は、次の式によって決定されます。

SCK周波数＝（ペリフェラル・クロック周波数SCLK）／（2×SPI_BAUD）

このレジスタに値0または1を書き込むと、シリアル・クロックはディスエーブルにされます。したがって、最大シリアル・クロック・レートは、システム・クロック・レートの4分の1です。

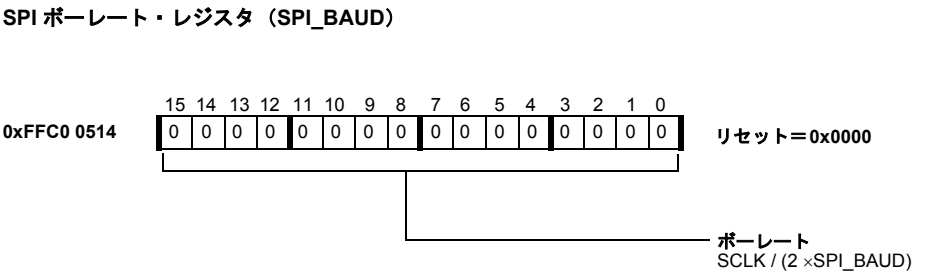


図 10-3. SPI ボーレート・レジスタ

表 10-1 は、SPI_BAUD に対するいくつかのボーレート値を示します。

表 10-1. SPI マスターのボーレート例

SPI_BAUD の 10 進値	SPI クロック（SCK）の除算係数	100MHz での SCLK のボーレート
0	N/A	N/A
1	N/A	N/A
2	4	25MHz
3	6	16.7MHz
4	8	12.5MHz
65,535 (0xFFFF)	131,070	763Hz

■ SPI_CTL レジスタ

SPI コントロール・レジスタ（SPI_CTL）は、SPI システムを設定し、イネーブルにするために使用されます。このレジスタは、SPI インターフェースをイネーブルにし、デバイスをマスターまたはスレーブとして選択し、データ転送フォーマットとワード・サイズを決定するために使用されます。

「ワード」という用語は、SPI_CTL のワード長（SIZE）ビットに応じて、8 ビットまたは 16 ビットの単一データ転送を表します。また、ハードウェアによって変更できる 2 つの特殊なビットもあります（SPE と MSTR）。

TIMOD フィールドは、送／受信バッファに転送を開始させるアクションの指定に使用されます。00 に設定された場合、受信バッファが読み出されると SPI ポート・トランザクションが開始されます。最初の SPI ポート・トランザクションを開始するために読出しが必要なため、最初の読出しからのデータは捨てる必要があります。01 に設定された場合、送信バッファが書き込まれるとトランザクションが開始されます。値 10 では DMA 受信モードが選択され、DMA 受信モードの SPI をイネーブルにして最初のトランザクションが開始されます。それ以降の個々のトランザクションは、

SPI_RDBRのDMA読出しによって開始されます。値11ではDMA送信モードが選択され、トランザクションはSPI_TDBRのDMA書込みによって開始されます。

PSSEビットは、マスターの$\overline{\text{SPISS}}$入力をイネーブルにするために使用されます。使用しない場合には、$\overline{\text{SPISS}}$をディスエーブルにして、チップ・ピンを汎用I/Oとして解放することができます。

EMISOビットは、MISOピンを出力としてイネーブルにします。これが必要となるのは、マスターがさまざまなスレーブに同時に送信したい（ブロードキャスト）環境です。マスターにデータを送信して戻せるのは、1つのスレーブに限られます。マスターが受信を希望するスレーブを除いて、他のすべてのスレーブではこのビットをクリアしてください。

ステータス・レジスタのMODFビットがセットされているとき、SPEビットとMSTRビットはハードウェアによって変更できます。10-30ページの「モード・フォルト・エラー（MODF）」を参照してください。

図 10-4にSPI_CTLのビット説明を示します。

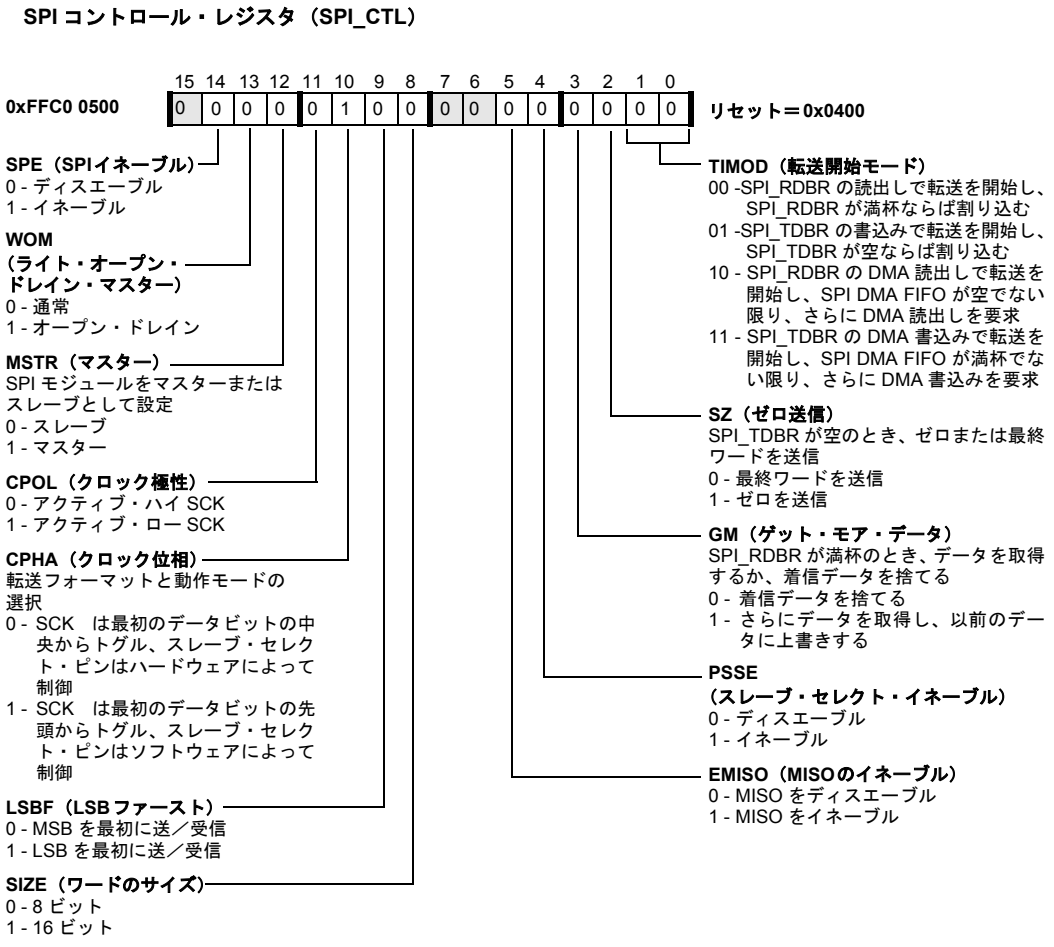


図 10-4. SPI コントロール・レジスタ

■ SPI_FLG レジスタ

SPIがマスターとしてイネーブルにされた場合、SPIでは、SPIフラグ・レジスタ（SPI_FLG）を使用して、個々のスレーブ・セレクト・ラインとして使用される7つまでの汎用プログラマブル・フラグ・ピンをイネーブルにします。スレーブ・モードでは、SPI_FLGビットに効果はなく、各SPIでは、$\overline{\text{SPISS}}$入力をスレーブ・セレクトとして使用します。図 10-5には、SPI_FLGレジスタの図を示します。

SPI フラグ・レジスタ（SPI_FLG）

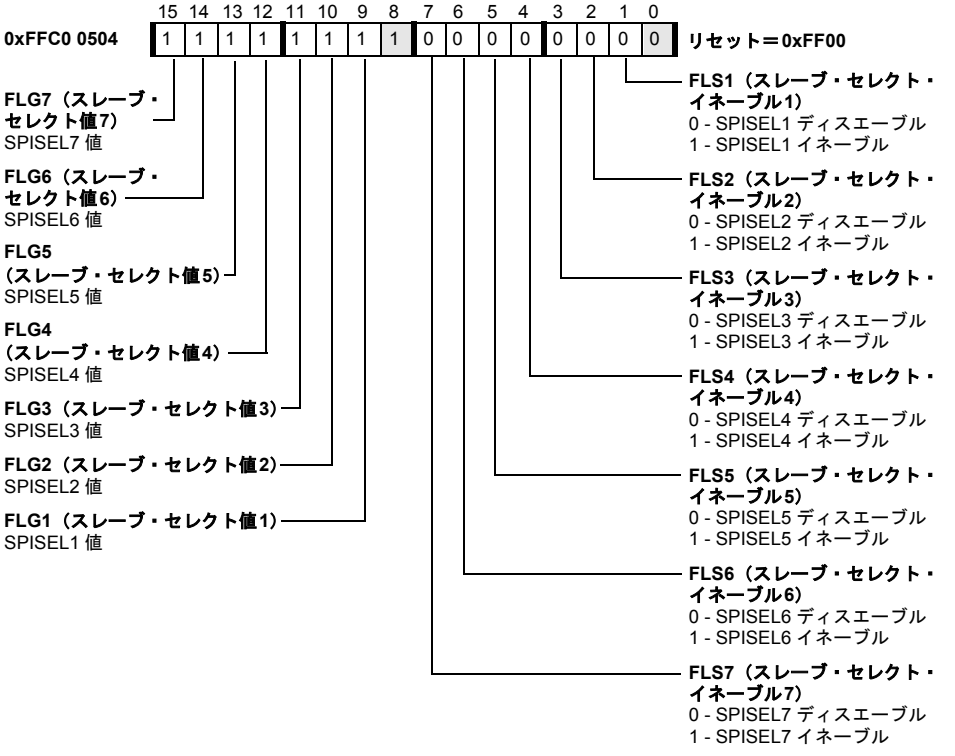


図 10-5. SPI フラグ・レジスタ

SPI_FLG レジスタは、2組のビットから構成されています。

- スレーブ・セレクト・イネーブル（FLSx）ビット

  各 FLSx ビットは、プログラマブル・フラグ（PFx）ピンに対応します。FLSx ビットがセットされると、対応する PFx ピンはスレーブ・セレクトとして駆動します。たとえば、SPI_FLG に FLS1 がセットされた場合には、PF1 がスレーブ・セレクト（SPISEL1）として駆動します。表 10-2 は、FLSx ビットと対応する PFx ピンとの関連を示します。

  FLSx ビットがセットされない場合には、汎用のプログラマブル・フラグ・レジスタ（FIO_DIR など）が対応する PFx ピンの設定と制御を行います。

- スレーブ・セレクト値（FLGx）ビット

  PFx ピンがスレーブ・セレクト出力として設定されると、FLGx ビットでは出力に駆動される値を決定できます。SPI_CTL の CPHA ビットがセットされた場合には、出力値は FLGx ビットのソフトウェア制御によって設定されます。SPI プロトコルを使用すると、転送されるワードとワードの間でスレーブ・セレクト・ラインはアサートされたままでいる（ローレベル）か、アサート解除されることができます。ユーザは、適切な FLGx ビットをセットまたはクリアする必要があります。たとえば、PF3 をスレーブ・セレクトとして駆動するには、SPI_FLG の FLS3 をセットする必要があります。SPI_FLG の FLG3 をクリアすると、PF3 はローレベルに駆動されます。FLG3 をセットすると、PF3 はハイレベルに駆動されます。転送と転送の間で FLG3 をセットおよびクリアすることによって、PF3 ピンはハイレベルとローレベルを循環できます。そうでない場合には、PF3 は転送と転送の間でアクティブ（ローレベル）なままです。

  CPHA ＝ 0 の場合には、SPI ハードウェアが出力値を設定し、FLGx ビットは無視されます。SPI プロトコルでは、転送されたワードと

ワードの間でスレーブ・セレクトはアサート解除される必要があります。この場合、SPI ハードウェアがピンを制御します。たとえば、PF3 をスレーブ・セレクト・ピンとして使用するには、SPI_FLG の FLS3 ビットをセットするだけで済みます。SPI ハードウェアが自動的に PF3 ピンを駆動するため、FLG3 ビットに書き込む必要はありません。

表 10-2. SPI_FLG ビットと PFx ピンとのマッピング

ビット	名前	機能	PFx ピン	デフォルト
0		予備		0
1	FLS1	SPISEL1 イネーブル	PF1	0
2	FLS2	SPISEL2 イネーブル	PF2	0
3	FLS3	SPISEL3 イネーブル	PF3	0
4	FLS4	SPISEL4 イネーブル	PF4	0
5	FLS5	SPISEL5 イネーブル	PF5	0
6	FLS6	SPISEL6 イネーブル	PF6	0
7	FLS7	SPISEL7 イネーブル	PF7	0
8		予備		1
9	FLG1	SPISEL1 値	PF1	1
10	FLG2	SPISEL2 値	PF2	1
11	FLG3	SPISEL3 値	PF3	1
12	FLG4	SPISEL4 値	PF4	1
13	FLG5	SPISEL5 値	PF5	1
14	FLG6	SPISEL6 値	PF6	1
15	FLG7	SPISEL7 値	PF7	1

▶ スレーブ・セレクト入力

SPIがスレーブ・モードにある場合には、SPISSはスレーブ・セレクト入力として機能します。マスターとしてイネーブルにされると、SPISSはマルチマスター環境でのSPIのエラー検出入力として機能できます。SPI_CTLのPSSEビットは、この機能をイネーブルにします。PSSE = 1の場合、SPISS入力はマスター・モードのエラー入力です。そうでない場合、SPISSは無視されます。

▶ 複数のスレーブSPIシステムに対するSPI_FLGのFLSビットの用途

SPI_FLGレジスタのFLSxビットは、複数のスレーブSPI環境で使用されます。たとえば、プロセッサ・マスターを含めてシステム内に8つのSPIデバイスがある場合には、マスター・プロセッサは、他の7つのデバイスにまたがってSPIモードのトランザクションをサポートできます。この設定では、このマルチスレーブ環境に必要なマスター・プロセッサはただ1つです。たとえば、SPIがマスターだとします。プロセッサ・マスター上の7本のフラグ・ピン（PF1～PF7）は、スレーブSPIデバイスの各SPISSピンに接続できます。この設定では、SPI_FLGのFLSxビットは3つのケースで使用できます。

ケース1と2では、プロセッサはマスターであり、SPIインターフェースを備えた7つのマイクロコントローラ／ペリフェラルはスレーブです。プロセッサは以下の動作が可能です。

1. ブロードキャスト・モードで、7つのSPIデバイスすべてに同時に送信できます。ここで、すべてのFLSxビットはセットされています。
2. 一度に1つのスレーブSPIデバイスだけをイネーブルにすることで、1つのSPIデバイスから送／受信できます。

   ケース3では、SPIポートを介して接続された8つのデバイスすべては、他のプロセッサでもかまいません。

3. すべてのスレーブもプロセッサである場合、リクエスタは、一度に（他の 6 つのスレーブ・プロセッサの EMISO ビットをクリアしてイネーブルにされた）ただ 1 つのプロセッサからデータを受信して、同時に 7 つのプロセッサすべてにブロードキャスト・データを送信できます。この EMISO 機能は、他のマイクロコントローラでも使用できます。したがって、この機能を組み込んだ他の SPI デバイスでも、EMISO 機能を使用できます。

図 10-6では、1つのプロセッサがマスターであり、3つのプロセッサ（または他のSPI互換デバイス）がスレーブになっています。

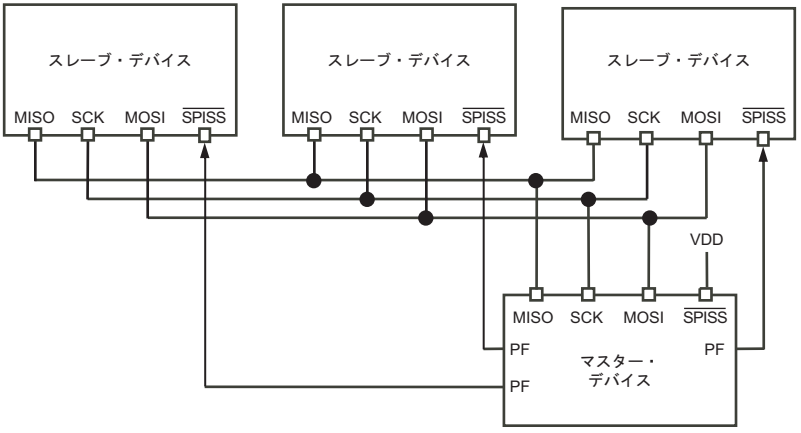


図 10-6. 単一マスター、複数スレーブ構成

■ SPI_STAT レジスタ

SPIステータス・レジスタ（SPI_STAT）は、SPI転送の完了タイミングや、送／受信エラーの発生の有無を検出するために使用されます。SPI_STAT レジスタは、いつでも読み出すことができます。

SPI_STATには、読出し専用ビットと、スティッキー・ビットがあります。SPIについてだけの情報を提供するビットは、読出し専用です。これらのビットは、ハードウェアによってセット／クリアされます。エラー条件が

発生すると、スティッキー・ビットがセットされます。これらのビットはハードウェアによってセットされ、ソフトウェアによってクリアされる必要があります。スティッキー・ビットをクリアするには、ユーザはSPI_STATの希望するビット位置に1を書き込む必要があります。たとえば、TXEビットがセットされた場合、ユーザはTXEエラー条件をクリアするためにSPI_STATのビット2に1を書き込む必要があります。これによって、ユーザはSPI_STATの値を変更せずに読み出すことができます。

スティッキー・ビットはリセット時にクリアされますが、SPIディスエーブル時にはクリアされません。

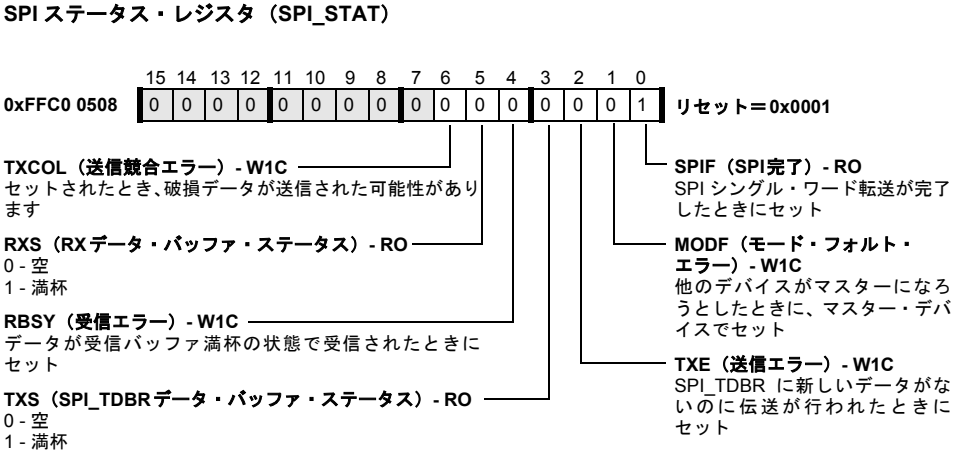


図 10-7. SPI ステータス・レジスタ

送信バッファは、書き込まれた後で満杯になります。転送が始まり、シフト・レジスタに送信値がロードされると、このバッファは空になります。シフト・レジスタ値が受信バッファにロードされると、転送の最後で受信バッファが満杯になります。受信バッファが読み出されると、このバッファは空になります。

SPIポートがディスエーブルにされると、SPIFビットがセットされます。

DMAモードに入ると同時に、送信バッファと受信バッファは空になります。つまり、DMAモードに入ると同時に、TXSビットとRXSビットは最初にクリアされます。

SPI送信用のDMAを使用するとき、DMA_DONE割込みは、DMA FIFOが空であることを知らせます。しかし、この時点では、SPI DMA FIFO内には送信されるのを待っているデータがまだあるかもしれません。したがって、ソフトウェアでは、SPI_STATレジスタのTXSが連続する2つの読出しの間ローレベルになる（この時点でSPI DMA FIFOは空になります）まで、TXSをポーリングする必要があります。その後SPIFビットがローレベルになると、最後のワードは転送されており、別のモードのためにSPIをディスエーブル／イネーブルにできます。

■ SPI_TDBR レジスタ

SPI送信データ・バッファ・レジスタ（SPI_TDBR）は、16ビットの読出し／書込みレジスタです。データは、このレジスタにロードされてから送信されます。データ転送の直前に、SPI_TDBRのデータはシフト・データ・レジスタ（SFDR）にロードされます。SPI_TDBRの読出しはいつでも行うことができ、SPI転送を妨げることも開始することもありません。

DMAの送信動作がイネーブルにされると、DMAエンジンは、データ転送の直前に送信用のデータをこのレジスタにロードします。このモードでは、SPI_TDBRへの書込みを行わないでください。というのも、このデータは、送信されるDMAデータを上書きするからです。

DMAの受信動作がイネーブルにされると、SPI_TDBRの内容が繰り返し送信されます。このモードではSPI_TDBRへの書込みが認められ、このデータは送信されます。

ゼロ送信制御ビット（SPI_CTLレジスタのSZ）がセットされている場合には、特定の状況のもとでSPI_TDBRは0にリセットできます。

転送の実行中にSPI_TDBRへの複数の書込みが発生した場合には、最後に書き込まれたデータだけが送信されます。SPI_TDBRに書き込まれた中間の値は送信されません。SPI_TDBRへの複数の書込みは可能ですが、お勧めできません。

SPI 送信データ・バッファ・レジスタ（SPI_TDBR）

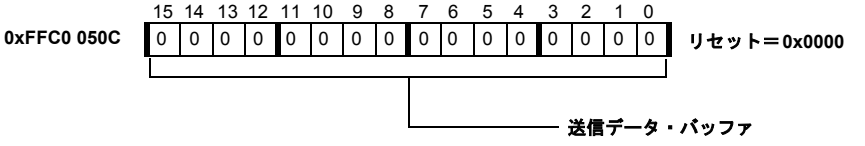


図 10-8. SPI 送信データ・バッファ・レジスタ

■ SPI_RDBR レジスタ

SPI受信データ・バッファ・レジスタ（SPI_RDBR）は、16ビットの読出し専用レジスタです。データ転送の最後に、シフト・レジスタ内のデータがSPI_RDBRにロードされます。DMA受信動作中に、SPI_RDBR内のデータは

DMAによって自動的に読み出されます。SPI_RDBRがソフトウェアを介して読み出されると、RXSビットがクリアされ、SPI転送を開始できます（TIMOD = 00の場合）。

SPI 受信データ・バッファ・レジスタ（SPI_RDBR）
RO

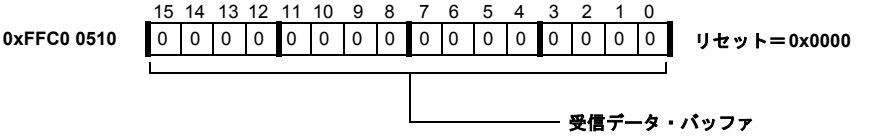


図 10-9. SPI 受信データ・バッファ・レジスタ

■ SPI_SHADOW レジスタ

SPI RDBRシャドウ・レジスタ（SPI_SHADOW）は、デバッグ・ソフトウェアで使用するために提供されています。このレジスタは、受信データ・バッファSPI_RDBRとは異なるアドレスにありますが、その内容はSPI_RDBRの内容と同じです。ソフトウェアがSPI_RDBRを読み出すと、SPI_STATのRXSビットがクリアされ、SPI転送を開始できます（SPI_CTLのTIMOD = 00の場合）。SPI_SHADOWレジスタが読み出されるときには、このようなハードウェア・アクションは行われません。SPI_SHADOWレジスタは読出し専用です。

SPI RDBR シャドウ・レジスタ（SPI_SHADOW）
RO

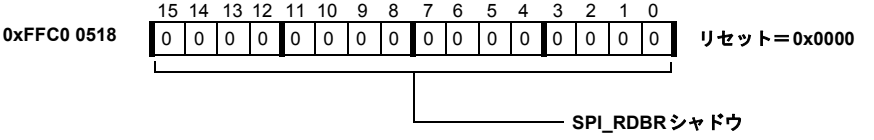


図 10-10.SPI RDBR シャドウ・レジスタ

■ レジスタ機能

表 10-3にSPIレジスタの機能をまとめます。

表 10-3. SPI レジスタのマッピング

レジスタ名	機能	注
SPI_CTL	SPI ポートの制御	SPE ビットと MSTR ビットはハードウェアによっても変更できます（MODF がセットされている場合）
SPI_FLG	SPI ポートのフラグ	ビット 0 と 8 は予備
SPI_STAT	SPI ポートのステータス	SPIF ビットは、SPI_CTL の SPE をクリアしてセットできます
SPI_TDBR	SPI ポート送信データ・バッファ	レジスタの内容は、ハードウェアによっても変更できます（DMA によって、または SPI_CTL で SZ = 1 のとき、あるいはその両方の場合）
SPI_RDBR	SPI ポート受信データ・バッファ	レジスタが読み出されると、ハードウェア・イベントがトリガされます
SPI_BAUD	SPI ポートのボー制御	値 0 または 1 でシリアル・クロックをディスエーブルします
SPI_SHADOW	SPI ポートのデータ	レジスタの内容は SPI_RDBR と同じですが、読み出されても何の措置もとられません

SPI 転送フォーマット

SPIはシリアル・クロック位相と極性の4つの異なる組合わせをサポートします。これらの組合わせは、SPI_CTLのCPOLビットとCPHAビットを使用して選択されます。

10-23ページの「CPHA＝0の場合のSPI転送プロトコル」と、10-23ページの「CPHA＝1の場合のSPI転送プロトコル」では、CPHAビットによって定義される2つの基本転送フォーマットを示します。SCKに関して2つの波形を示します（1つはCPOL = 0の場合、もう1つはCPOL = 1の場合）。SCKピン、MISOピン、MOSIピンは、マスターとスレーブとの間で直結され

ているため、これらの図は、マスターやスレーブのタイミング図と解釈することもできます。MISO信号はスレーブからの出力であり（スレーブ伝送）、MOSI信号はマスターからの出力です（マスター伝送）。SCK信号はマスターによって生成され、$\overline{\text{SPISS}}$信号はマスターからスレーブへのスレーブ・デバイス・セレクト入力です。これらの図は、最上位ビット（MSB）ファースト（LSBF = 0）による8ビット転送（SIZE = 0）を表します。SPI_CTLのSIZEビットとLSBFビットは、自由に組み合わせることができます。たとえば、最下位ビット（LSB）ファーストによる16ビット転送という設定も可能です。

同じ通信リンクに属するマスター・デバイスとスレーブ・デバイスは、クロック極性とクロック位相を同じ設定にしてください。マスターからの転送フォーマットは、スレーブ・デバイスのさまざまな条件に合わせて調整するため、転送ごとに異なる場合もあります。

CPHA = 0の場合、スレーブ・セレクト・ライン$\overline{\text{SPISS}}$は各シリアル転送の間で非アクティブ（ハイレベル）であることが必要です。これは、SPIハードウェア・ロジックによって自動的に制御されます。CPHA = 1の場合、$\overline{\text{SPISS}}$は連続した転送の間でアクティブ（ローレベル）のままであるか、非アクティブ（ハイレベル）になることができます。これは、SPI_FLG.の操作を介して、ソフトウェアによって制御する必要があります。

図 10-11は、CPHA = 0の場合のSPI転送プロトコルを示します。なお、SCKはデータ転送の途中でトグル動作を開始し、SIZE = 0、LSBF = 0です。

図 10-12は、CPHA = 1の場合のSPI転送プロトコルを示します。なお、SCKはデータ転送の初めにトグル動作を開始し、SIZE = 0、LSBF = 0です。

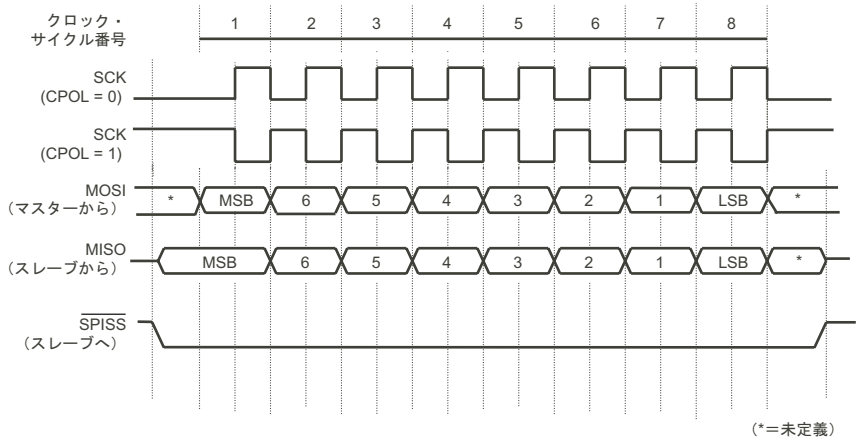


図 10-11.CPHA ＝ 0 の場合の SPI 転送プロトコル

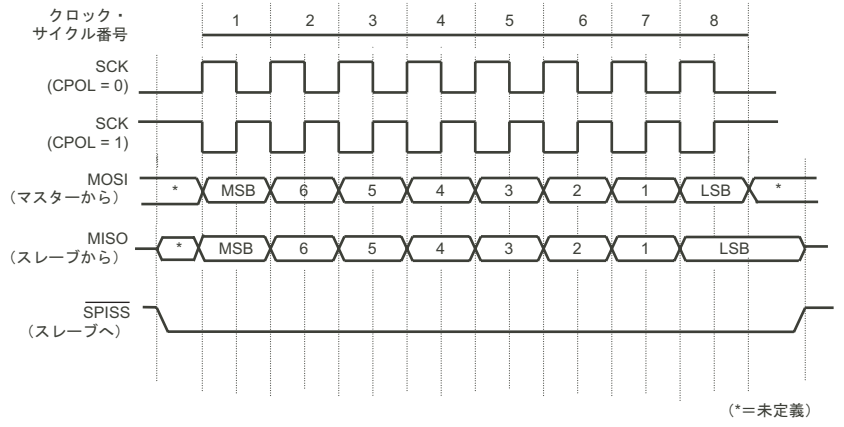


図 10-12.CPHA ＝ 1 の場合の SPI 転送プロトコル

SPI の一般的な動作

SPIは、シングル・マスター環境でもマルチマスター環境でも使用できます。いずれの構成でも、MOSI、MISO、SCKの信号は、すべて一緒に接続されます。ブロードキャスト・モードが選択されていない限り、SPIの送信と受信は常に同時にイネーブルにされます。ブロードキャスト・モードでは、複数のスレーブの受信をイネーブルにできますが、ただ1つのスレーブを送信モードにしてMISOラインを駆動することが必要です。送信も受信も必要ない場合には、単に無視できます。ここでは、クロック信号、マスター／スレーブとしてのSPI動作、エラー生成について説明します。

SPIモジュール設定を変更するときには、データ破壊を避けるための対策をとる必要があります。データ転送中に設定を変更してはいけません。クロック極性の変更は、スレーブが選択されていないときにだけ行ってください。これに対する例外は、SPI通信リンクがシングル・マスターとシングル・スレーブで構成され、CPHA = 1であり、スレーブのスレーブ・セレクト入力が常にローレベルに接続されている場合です。この場合、スレーブは常に選択されており、データ破壊を避けるには、マスター・デバイスとスレーブ・デバイスの両方が設定された後で、スレーブをイネーブルにしてください。

マルチマスターまたはマルチスレーブSPIシステムでは、データ出力ピン（MOSIとMISO）は、オープン・ドレイン出力として振る舞うように設定できます。これによって、ピン・ドライバに対する競合と損傷の可能性を防止できます。このオプションが選択された場合、MOSIピンとMISOピンには外付けのプルアップ抵抗が必要です。

このオプションはWOMビットによって制御されます。WOMがセットされ、SPIがマスターとして設定されると、MOSIで駆動出力されたデータがロジック・ハイであるとき、MOSIピンは3ステートになります。駆動されたデータがロジック・ローであるとき、MOSIピンは3ステートになりません。

同様に、WOMがセットされ、SPIがスレーブとして設定されると、MISOで駆動出力されたデータがロジック・ハイである場合に、MISOピンは3ステートになります。

■ クロック信号

SCK信号は、データ転送時にはワードの転送期間中にだけアクティブなゲーテッド・クロックです。アクティブ・エッジの数は、データ・ラインで駆動されるビット数に等しくなります。クロック・レートは、SCLKレートの4分の1まで高くできます。マスター・デバイスの場合、クロック・レートはSPI_BAUDの16ビット値によって決定されます。スレーブ・デバイスの場合、SPI_BAUDの値は無視されます。SPIデバイスがマスターであるとき、SCKは出力信号です。SPIがスレーブであるとき、SCKは入力信号です。スレーブ・セレクト入力が非アクティブ（ハイレベル）に駆動された場合、スレーブ・デバイスはシリアル・クロックを無視します。

SCK信号は、MISOラインとMOSIラインに駆動されたデータをシフト・アウト／シフト・インするために使用されます。データは、常にクロックの1つのエッジ（アクティブ・エッジ）でシフト・アウトされ、クロックの反対側のエッジ（サンプリング・エッジ）でサンプリングされます。データを基準にしたクロック極性とクロック位相は、SPI_CTLにプログラマブルであり、転送フォーマットを定義します。

■ マスター・モード動作

SPIがマスターとして設定され、DMAモードは選択されない場合、インターフェースは次のように動作します。

1. コアは SPI_FLG に書き込んで、1 つまたは複数の SPI フラグ・セレクト・ビット（FLSx）をセットします。これによって、マスターの設定中に、希望するスレーブが適切に選択解除されることが保証されます。
2. コアは SPI_BAUD レジスタと SPI_CTL レジスタに書き込んで、デバイスをマスターとしてイネーブルにし、適切なワード長、転送フォーマット、ボーレート、その他の必要な情報を指定して SPI システムを設定します。
3. CPHA = 1 の場合、コアは SPI_FLG の 1 つまたは複数の SPI フラグ・ビット（FLGx）をクリアして希望するスレーブをアクティブにします。
4. SPI_CTL の TIMOD ビットは、SPI 転送開始モードを決定します。SPI リンクでの転送は、コアによる送信データ・バッファ（SPI_TDBR）へのデータ書込み、または受信データ・バッファ（SPI_RDBR）のデータ読出しと同時に始まります。
5. 続いて SPI は、SCK 上にプログラムされたクロック・パルスを生成し、MOSI からのデータのシフト・アウトと MISO からのデータのシフト・インを同時に行います。シフトの前に、シフト・レジスタには SPI_TDBR レジスタの内容がロードされます。転送の最後に、シフト・レジスタの内容は SPI_RDBR にロードされます。
6. 新しいそれぞれの転送開始コマンドを使用し、SPI 転送開始モードに基づいて、SPI はワードの送／受信を続行します。

送信バッファが空のままであるか、受信バッファが満杯のままである場合には、デバイスは SPI_CTL の SZ ビットと GM ビットの状態に基づいて動作します。SZ = 1 で送信バッファが空の場合には、デバイスは MOSI ピンで 0 を繰り返し送信します。新しい転送開始コマンドごとに 1 ワードが送信されます。SZ = 0 で送信バッファが空の場合には、デバイスは送信バッファが

空になる前に送信した最後のワードを繰り返し送信します。GM = 1で受信バッファが満杯の場合には、デバイスはMISOピンから新しいデータの受信を続行し、SPI_RDBRバッファ内の古いデータに上書きします。GM = 0で受信バッファが満杯の場合には、着信データは捨てられ、SPI_RDBRは更新されません。

■ マスターからの転送開始（転送モード）

デバイスがマスターとしてイネーブルにされると、転送の開始は、SPI_CTLの2つのTIMODビットによって定義されます。これら2つのビットとインターフェースのステータスに基づいて、新しい転送はSPI_RDBRの読出しまたはSPI_TDBRへの書込みと同時に開始されます。これを表 10-4に要約します。

SPIポートがTIMOD＝01またはTIMOD＝11でイネーブルにされた場合には、ハードウェアはすぐに最初の割込みまたはDMA要求を発行します。

表 10-4. 転送開始

TIMOD	機能	転送開始のタイミング	アクション、割込み
00	送／受信	以前の転送が完了しSPI_RDBRの読出しと同時に、新しいシングル・ワード転送を開始します。	受信バッファが満杯の場合、割込みはアクティブ。 SPI_RDBRの読出しによって割込みをクリア。
01	送／受信	以前の転送が完了しSPI_TDBRへの書込みと同時に、新しいシングル・ワード転送を開始します。	送信バッファが空の場合、割込みはアクティブ。 SPI_TDBRへの書込みによって割込みをクリア。

表 10-4. 転送開始（続き）

TIMOD	機能	転送開始のタイミング	アクション、割込み
10	DMA での受信	SPI を DMA モード用にイネーブルにすると同時に新しいマルチワード転送を開始します。個々のワード転送は、SPI_RDBR の DMA 読出しで始まり、転送が完了するまで続きます。	SPI DMA FIFO が空でない限り、DMA 読出しを要求します。
11	DMA での送信	SPI を DMA モード用にイネーブルにすると同時に新しいマルチワード転送を開始します。個々のワード転送は、SPI_TDBR への DMA 書込みで始まり、転送が完了するまで続きます。	SPI DMA FIFOが満杯でない限り、DMA 書込みを要求します。

■ スレーブ・モード動作

デバイスがスレーブとしてイネーブルにされ、DMAモードが選択されていない場合、転送の開始はCPHAの状態に応じて、$\overline{\text{SPISS}}$選択信号のアクティブ状態（ローレベル）への遷移によってトリガされるか、クロック（SCK）の最初のアクティブ・エッジによってトリガされます。

以下の手順では、スレーブ・モードでのSPI動作を示します。

1. コアは SPI_CTL に書き込んで、シリアル・リンクのモードを SPI マスターで設定されたモードと同じになるよう定義します。
2. データ転送の準備をするため、コアは SPI_TDBR に送信されるデータを書き込みます。
3. $\overline{\text{SPISS}}$ の立下がりエッジが検出されると、スレーブは SCK のアクティブ・エッジでデータの送信を開始し、SCK の非アクティブ・エッジでデータのサンプリングを開始します。

4. 送／受信は、SPISS が解放されるか、スレーブが正しい数のクロック・サイクルを受信するまで続行されます。
5. スレーブ・デバイスは、SPISS での新しい立下がりエッジ遷移または SCK のアクティブ・クロック・エッジ、あるいはその両方で送／受信を続行します。

送信バッファが空のままであるか、受信バッファが満杯のままである場合、デバイスは、SPI_CTL の SZ ビットと GM ビットの状態に基づいて動作します。SZ = 1 で送信バッファが空である場合には、デバイスは MISO ピンで 0 を繰り返し送信します。SZ = 0 で送信バッファが空である場合には、送信バッファが空になる前に送信した最後のワードを繰り返し送信します。GM = 1 で受信バッファが満杯の場合には、デバイスは MOSI ピンから新しいデータの受信を続行し、SPI_RDBR 内の古いデータに上書きします。GM = 0 で受信バッファが満杯の場合には、着信データは捨てられ、SPI_RDBR は更新されません。

■ 転送の準備ができたスレーブ

デバイスがスレーブとしてイネーブルにされると、デバイスを新しい転送用に準備するには、表 10-5 に示すアクションが要求されます。

表 10-5. 転送準備

TIMOD	機能	アクション、割込み
00	送／受信	受信バッファが満杯のとき、割込みはアクティブ。 SPI_RDBR の読出しで割込みをクリアします。
01	送／受信	送信バッファが空のとき、割込みはアクティブ。 SPI_TDBR への書込みで割込みをクリアします。
10	DMA で受信	SPI DMA FIFO が空でない限り、DMA 読出しを要求します。
11	DMA で送信	SPI DMA FIFOが満杯でない限り、DMA書込みを要求します。

エラー信号とフラグ

デバイスのステータスは、SPI_STAT レジスタによって示されます。詳細については、10-16ページの「SPI_STAT レジスタ」を参照してください。

■ モード・フォルト・エラー（MODF）

マスターとしてイネーブルにされたデバイスの$\overline{\text{SPISS}}$入力ピンが、システム内の他のデバイスによってローレベルに駆動されると、SPI_STAT のMODF ビットがセットされます。この現象が発生するのは、マルチマスター・システムにおいて、他のデバイスもマスターになろうとしているときです。この機能をイネーブルにするには、SPI_CTL のPSSE ビットをセットする必要があります。2つのドライバ間でのこのような競合は、駆動ピンを損傷させる可能性があります。このエラーが検出されるとすぐに、以下のアクションがとられます。

- SPI_CTL のMSTR 制御ビットがクリアされ、SPIインターフェースはスレーブとして設定されます。
- SPI_CTL のSPE 制御ビットがクリアされ、SPIシステムはディスエーブルにされます。
- SPI_STAT のMODF ステータス・ビットがセットされます。
- SPIエラー割込みが生成されます。

これら4つの条件は、MODF ビットがソフトウェアによってクリアされるまで持続します。MODF ビットがクリアされるまでは、SPIはスレーブとしてさえもイネーブルにできません。MODF がセットされている間は、ユーザはSPE もMSTR もセットできません（ハードウェアによる禁止）。

MODF がクリアされると、割込みは非アクティブにされます。SPI をマスターとして再びイネーブルにする前に、$\overline{\text{SPISS}}$ 入力ピンの状態をチェックして、ピンがハイレベルであることを確認します。そうでない場合には、SPE と MSTR がセットされると、すぐに別のモード・フォルト・エラー条件が発生します。

SPE と MSTR がクリアされると、SPI データとクロック・ピン・ドライバ（MOSI、MISO、SCK）がディスエーブルにされます。しかし、スレーブ・セレクト出力ピンは、プログラマブル・フラグ・レジスタによって再び制御されるようになります。プロセッサがまだこれらのラインを駆動している場合には、これはスレーブ・セレクト・ラインでの競合につながることがあります。MODF エラーが発生するとスレーブ・セレクト出力ドライバがディスエーブルにされるようにするには、プログラマブル・フラグ・レジスタを適切にプログラム設定する必要があります。

MODF 機能をイネーブルにするとき、プログラムでは、スレーブ・セレクトとして使用されるすべての PFx ピンを入力として設定する必要があります。プログラムでこれを行うには、PFx ピンの方向を設定してから、SPI を設定します。これによって、MODF エラーが発生し、スレーブ・セレクトが PFx ピンとして自動的に再設定されると、スレーブ・セレクト出力ドライバはディスエーブルにされます。

■ 送信エラー（TXE）

送信の条件がすべて満足され、SPI_TDBR 内に新しいデータがない（SPI_TDBR が空である）場合、SPI_STAT に TXE ビットがセットされます。この場合、送信の内容は SPI_CTL の SZ ビットの状態に依存します。TXE ビットはスティッキーです（W1C）。

■ 受信エラー（RBSY）

新しい転送が完了しても、SPI_RDBRから以前のデータを読出しできる前であれば、SPI_STATレジスタのRBSYフラグがセットされます。SPI_RDBRが新たに受信されたデータで更新されるかどうかは、SPI_CTLレジスタのGMビットの状態で決まります。RBSYビットはスティッキーです（W1C）。

■ 送信競合エラー（TXCOL）

SPI_TDBRへの書込みがシフト・レジスタのロードと同時に起きると、SPI_STATのTXCOLフラグがセットされます。SPI_TDBRへの書込みは、ソフトウェアまたはDMAを介して可能です。TXCOLビットは、破損データがシフト・レジスタにロードされ、送信された可能性を示します。この場合、SPI_TDBR内のデータは送信された内容と一致しないことがあります。このエラーは、適切なソフトウェア制御によって容易に回避できます。TXCOLビットはスティッキーです（W1C）。

SPI 転送の開始と終了

SPI転送の開始と終了は、デバイスの設定がマスターであるかスレーブであるか、CPHAモードが選択されたかどうか、および転送開始モード（TIMOD）が選択されたかどうかに依存します。CPHA = 0のマスターSPIでは、転送はTIMODに応じて、SPI_TDBRが書き込まれたとき、またはSPI_RDBRが読み出されたときに開始されます。転送の最初に、イネーブルにされたスレーブ・セレクト出力はアクティブ（ローレベル）に駆動されます。しかし、SCK信号は、SCKの最初のサイクルの前半は非アクティブのままです。CPHA = 0のスレーブでは、転送は$\overline{\text{SPISS}}$入力がローレベルになるとすぐに開始されます。

CPHA = 1 の場合、スレーブ・デバイスとマスター・デバイスの両方で、転送は SCK の最初のアクティブ・エッジで始まります。マスター・デバイスの場合、転送が完了したと見なされるのは、最後のデータを送信すると同時に最後のデータビットを受信した後です。スレーブ・デバイスの転送は、SCK の最後のサンプリング・エッジの後で終了します。

RXS ビットでは、受信バッファの読出しのタイミングを定義します。TXS ビットでは、送信バッファをフィルできるタイミングを定義します。シングル・ワード転送の最後には RXS ビットがセットされて、新しいワードの受信と、受信バッファ SPI_RDBR へのラッチが示されます。マスター SPI の場合、RXS は SCK の最後のサンプリング・エッジのすぐ後でセットされます。スレーブ SPI の場合、RXS は CPHA や CPOL とは無関係に、最後の SCK エッジのすぐ後でセットされます。一般に、遅延は数 SCLK サイクルであり、TIMOD やボーレートとは無関係です。SPI_RDBR が満杯のときに割込みを生成するように設定された場合（TIMOD = 00）、割込みは、RXS がセットされてから 1 SCLK サイクル後にアクティブになります。この割込みに依存しない場合には、RXS ビットをポーリングして転送の終了を検出できます。

他の SPI デバイスとのソフトウェア互換性を維持するため、SPIF ビットもポーリングに使用できます。このビットの振る舞いは、他の市販デバイスとは少し異なる場合があります。スレーブ・デバイスの場合、SPIF は転送開始（CPHA = 0 では $\overline{\text{SPISS}}$ がローレベルになり、CPHA = 1 では SCK の最初のアクティブ・エッジ）のすぐ後でクリアされ、RXS と同時にセットされます。マスター・デバイスの場合、SPIF は CPHA や CPOL とは無関係に、転送開始（TIMOD に応じて、SPI_TDBR の書込みまたは SPI_RDBR の読出しによる）のすぐ後でクリアされ、最後の SCK エッジの 1/2 SCK 周期後にセットされます。

SPIF がセットされるタイミングはボーレートに依存します。一般に、SPIF は RXS の後でセットされますが、最低のボーレート（SPI_BAUD < 4）のときを除きます。SPIF ビットは、RXS がセットされる前にセットされます。これは、遅延のために、新しいデータが SPI_RDBR にラッチされる前となります。したがって、SPI_BAUD = 2 または SPI_BAUD = 3 の場合には、

SPI_RDBR を読み出すため、RXS は SPIF の前にセットする必要があります。大きな SPI_BAUD 設定の場合には、RXS がセットされてから、SPIF がセットされることが保証されます。

SPI ポートを使用して送信と受信を同時に行う場合、または受信動作と送信動作を頻繁に切り替える場合には、TIMOD = 00 モードが最高の動作オプションの場合もあります。このモードでは、ソフトウェアは、SPI_RDBR レジスタからダミー読出しを実行して、最初の転送を開始します。最初の転送がデータ送信に使用される場合には、ソフトウェアは、送信される値を SPI_TDBR レジスタに書き込んでから、ダミー読出しを実行してください。送信される値が任意である場合には、ランダム値ではなくゼロ・データが送信されるように、SZ ビットを設定することをお勧めします。SPI ストリームの最終ワードを受信するとき、ソフトウェアでは、SPI_RDBR レジスタからの読出しが別の転送を開始しないように保証してください。SPI_RDBR の最終の読出しアクセスの前に、SPI ポートをディスエーブルにすることをお勧めします。SPI_SHADOW レジスタを読み出しても、割込み要求はクリアされないため、十分ではありません。

CPHA ビットをセットしたマスター・モードでは、トランザクションを開始する前に、必要なスレーブ・セレクト信号をソフトウェアで手動アサートしてください。すべてのデータが転送された後、一般に、ソフトウェアはスレーブ・セレクトを再び解放します。完全な転送のために、SPI スレーブ・デバイスがスレーブ・セレクト・ラインのアサートを必要とする場合、TIMOD = 00 または TIMOD = 10 のモードで動作しているときにだけ SPI 割込みサービス・ルーチンで実行できます。TIMOD = 01 または TIMOD = 11 では、転送がまだ進行中のときに割込みが要求されます。

DMA

SPIポートでもダイレクト・メモリ・アクセス（DMA）を使用できます。DMAの詳細については、9-1ページの「ダイレクト・メモリ・アクセス」を参照してください。

■ DMA 機能

SPIの1つのDMAエンジンは、SPI送信チャンネルまたは受信チャンネルをとして利用できるように設定できますが、両方を同時に利用することはできません。したがって、送信チャンネルとして設定された場合、受信されたデータは基本的に無視されます。

受信チャンネルとして設定された場合、何が送信されるかは無関係です。DMAアクセス・バス（DAB）でのスループットを向上させるため、16ビット×4ワードのFIFO（バースト機能なし）が搭載されています。

DMAをSPI送信に使用するとき、DMA_DONE割込みは、DMA FIFOが空であることを示します。しかしこの時点では、SPI DMA FIFO内には、送信されるのを待つデータがまだあるかもしれません。したがって、SPI_STATレジスタのTXSが2つの連続した読出しの間ローレベルになる（この時点でSPI DMA FIFOは空になります）まで、ソフトウェアでTXSをポーリングする必要があります。その後SPIFビットがローレベルになると、最終ワードは転送されており、SPIを別のモード用にイネーブル／ディスエーブルにできます。

SPIポートがディスエーブルにされると、4ワードのFIFOはクリアされます。

■ マスター・モードの DMA 動作

DMAエンジンをデータの送信または受信用に設定してマスターとしてイネーブルにすると、SPIインターフェースは、次のように動作します。

1. SPI DMA チャンネルをイネーブルにし、必要な作業単位、アクセス方向、ワード・カウントなどを設定するために、プロセッサ・コアは適切な DMA レジスタに書き込みます。詳細については、9-1ページの「ダイレクト・メモリ・アクセス」を参照してください。
2. プロセッサ・コアは、SPI_FLG レジスタに書き込み、1 つまたは複数の SPI フラグ・セレクト・ビット（FLSx）を設定します。
3. プロセッサ・コアは、SPI_BAUD レジスタと SPI_CTL レジスタに書き込み、適切なワード長、転送フォーマット、ボーレートなどを指定することによって、デバイスをマスターとしてイネーブルにし、SPI システムを設定します。TIMOD フィールドは、「DMA で受信」（TIMOD = 10）または「DMA で送信」（TIMOD = 11）のモードを選択するように設定してください。
4. 受信用に設定された場合、SPI のイネーブルと同時に受信転送が開始されます。それ以降の転送は、SPI が SPI_RDBR レジスタからデータを読み出すとき、および SPI DMA FIFO に書き込むときに開始されます。続いて SPI は、メモリへの DMA 書込みを要求します。DMA の許可と同時に、DMA エンジンは SPI DMA FIFO からワードを読み出し、メモリに書き込みます。

   送信用に設定された場合、SPI はメモリからの DMA 読出しを要求します。DMA の許可と同時に、DMA エンジンはメモリからワードを読み出し、SPI DMA FIFO に書き込みます。SPI が、SPI DMA FIFO からのデータを SPI_TDBR レジスタに書き込むと、SPI リンクでの転送が開始されます。
5. その後 SPI は、SCK 上にプログラムされたクロック・パルスを生成し、MOSI からのデータのシフト・アウトと MISO からのデータのシフト・インを同時に実行します。受信転送の場合、シフト・レジス

タ内の値は、転送の最後に SPI_RDBR レジスタにロードされます。送信転送の場合、SPI_TDBR レジスタ内の値は、転送の初めにシフト・レジスタにロードされます。

6. 受信モードでは、SPI DMA FIFO 内にデータがある（FIFO が空でない）限り、SPI はメモリへの DMA 書込みの要求を続行します。DMA エンジンは SPI DMA ワード・カウント・レジスタが 1 から 0 に遷移するまで、SPI DMA FIFO からのワード読出しとメモリへの書込みを続行します。SPI は SPI DMA モードがディスエーブルにされるまで、ワードの受信を続行します。

   送信モードでは、SPI DMA FIFO に空領域がある（FIFO が満杯でない）限り、SPI はメモリからの DMA 読出しの要求を続行します。DMA エンジンは SPI DMA ワード・カウント・レジスタが 1 から 0 に遷移するまで、メモリからのワード読出しと SPI DMA FIFO への書込みを続行します。SPI は SPI DMA FIFO が空になるまで、ワードの送信を続行します。

受信DMA動作では、DMAエンジンが受信データストリームについていけない場合、受信バッファはGMビットの状態に基づいて動作します。GM = 1でDMA FIFOが満杯の場合には、デバイスはMISOピンから新しいデータの受信を続行し、SPI_RDBRレジスタ内の古いデータに上書きします。GM = 0でDMA FIFOが満杯の場合には、受信データは捨てられ、SPI_RDBRレジスタは更新されません。受信DMAの実行中に、送信バッファは空でありTXEはセットされていると見なされます。SZ = 1の場合、デバイスはMOSIピンで0を繰り返し送信します。SZ = 0の場合、SPI_TDBRレジスタの内容を繰り返し送信します。このモードでは、TXEアンダーラン条件はエラー割込みを生成できません。

送信DMA動作の場合、マスターSPIがワード転送を開始するのは、DMA FIFO内にデータがあるときだけです。DMA FIFOが空の場合、SPIは、DMAエンジンがDMA FIFOに書き込むのを待ってから転送を開始します。送信DMAモードに設定された場合には、SPI_RDBRレジスタ内のデータやRXSビットとRBSYビットのステータスを含めて、SPI受信動作のすべての側面を無視してください。このモードでは、RBSYオーバーラン条件は

エラー割込みを生成できません。DMA FIFOにデータがない場合、マスターSPIは転送を開始しないため、このモード（マスターDMA TXモード）ではTXEアンダーラン条件は発生できません。

アクティブなSPI送信DMA動作中は、DMAデータへの上書きを防ぐため、SPI_TDBRレジスタへの書込みを行わないでください。アクティブなSPI受信DMA動作中のSPI_TDBRレジスタへの書込みは許されます。SPI_RDBRレジスタからの読出しはいつでも許されます。

DMA要求が生成されるのは、DMA FIFOが空でないとき（TIMOD = 10のとき）、またはDMA FIFOが満杯でないとき（TIMOD = 11のとき）です。

エラー割込みが生成されるのは、RBSYオーバーフロー・エラー条件があるとき（TIMOD = 10のとき）です。

マスターSPI DMAシーケンスには、複数のDMA作業単位のバックツーバック送信または受信、あるいはその両方が伴うことがあります。SPIコントローラでは、最小のコア介入でこのようなシーケンスをサポートします。

■ スレーブ・モードDMA動作

DMAエンジンをデータの送信または受信用に設定してスレーブとしてイネーブルにされると、転送の開始はCPHAの状態に応じて、$\overline{\text{SPISS}}$信号のアクティブ・ロー状態への遷移によってトリガされたり、SCKの最初のアクティブ・エッジによってトリガされたりします。

以下の手順は、マスター・コマンドに応じて、SPIスレーブでのSPI受信／送信DMAシーケンスを示します。

1. プロセッサ・コアでは、適切な DMA レジスタに書き込んで SPI DMA チャンネルをイネーブルにし、必要な作業単位、アクセス方向、ワード・カウントなどを設定します。詳細については、9-1 ページの「ダイレクト・メモリ・アクセス」を参照してください。

2. プロセッサ・コアでは、SPI_CTL レジスタに書き込んで、シリアル・リンクのモードを SPI マスターで設定されたモードと同じに定義します。TIMOD フィールドは、「DMA で受信」（TIMOD = 10）または「DMA で送信」（TIMOD = 11）のモードを選択するように設定されます。

3. 受信用に設定された場合、スレーブ・セレクト入力がアクティブになると、スレーブはアクティブ SCK エッジでデータの送／受信を開始します。シフト・レジスタ内の値は、転送の最後に SPI_RDBR レジスタにロードされます。SPI は、SPI_RDBR レジスタからデータを読み出して SPI DMA FIFO に書き込むときに、メモリへの DMA 書込みを要求します。DMA の許可と同時に、DMA エンジンは SPI DMA FIFO からワードを読み出し、メモリに書き込みます。

   送信用に設定された場合、SPI はメモリからの DMA 読出しを要求します。DMA の許可と同時に、DMA エンジンは、メモリからワードを読み出して SPI DMA FIFO に書き込みます。続いて SPI は、SPI DMA FIFO からデータを読み出して SPI_TDBR レジスタに書き込み、次の転送の開始を待ちます。スレーブ・セレクト入力がアクティブになると、スレーブはアクティブ SCK エッジでデータの送／受信を開始します。SPI_TDBR レジスタ内の値は、転送の初めにシフト・レジスタにロードされます。

4. 受信モードでは、SPI DMA FIFO 内にデータがある（FIFO が空でない）限り、SPI スレーブは、メモリへの DMA 書込みの要求を続行します。DMA エンジンは、SPI DMA ワード・カウント・レジスタが 1 から 0 に遷移するまで、SPI DMA FIFO からのワード読出しとメモリへの書込みを続行します。SPI スレーブは、スレーブ・セレクト入力がアクティブである限り、アクティブ SCK エッジでワードの受信を続行します。

送信モードでは、SPI DMA FIFO に空領域がある（FIFO が満杯でない）限り、SPI スレーブは、メモリからの DMA 読出しの要求を続行します。DMA エンジンは、SPI DMA ワード・カウント・レジスタが 1 から 0 に遷移するまで、メモリからのワード読出しと SPI DMA FIFO への書込みを続行します。SPI スレーブは、スレーブ・セレクト入力がアクティブである限り、アクティブ SCK エッジでワードの送信を続行します。

受信DMA動作では、DMAエンジンが受信データストリームについて行けない場合に、受信バッファはGMビットの状態に基づいて動作します。GM = 1でDMA FIFOが満杯の場合、デバイスはMOSIピンから新しいデータの受信を続行し、SPI_RDBRレジスタ内の古いデータに上書きします。GM = 0でDMA FIFOが満杯の場合には、着信データは捨てられ、SPI_RDBRレジスタは更新されません。受信DMAの実行中、送信バッファは空であり、TXEはセットされていると見なされます。SZ = 1の場合には、デバイスは、MISOピンで0を繰り返し送信します。SZ = 0の場合には、SPI_TDBRレジスタの内容を繰り返し送信します。このモードでは、TXEアンダーラン条件はエラー割込みを生成できません。

送信DMA動作では、DMAエンジンが送信ストリームについて行けない場合に、送信ポートはSZビットの状態に基づいて動作します。SZ = 1でDMA FIFOが空の場合、デバイスはMISOピンで0を繰り返し送信します。SZ = 0でDMA FIFOが空の場合には、DMAバッファが空になる前に送信した最後のワードを繰り返し送信します。送信DMAモードに設定した場合には、SPI_RDBRレジスタ内のデータ、RXSビットとRBSYビットのステータスなど、SPI受信動作のすべての側面を無視してください。このモードでは、RBSYオーバーラン条件はエラー割込みを生成できません。

アクティブなSPI送信DMA動作中はDMAデータの上書きを防ぐため、SPI_TDBRレジスタへの書込みは行わないでください。アクティブなSPI受信DMA動作中のSPI_TDBRレジスタへの書込みは可能です。SPI_RDBRレジスタからの読出しはいつでも可能です。

DMA要求が生成されるのは、DMA FIFOが空でない（TIMOD = 10である）とき、またはDMA FIFOが満杯でない（TIMOD = 11である）ときです。

エラー割込みが生成されるのは、RBSYオーバーフロー・エラー条件がある（TIMOD = 10である）とき、またはTXEアンダーフロー・エラー条件がある（TIMOD = 11である）ときです。

タイミング

イネーブル・リード時間（T1）、イネーブル遅延時間（T2）、順次転送遅延時間（T3）は、それぞれ、常にSCK周期の2分の1以上であることが必要です。図 10-13を参照してください。連続したワード転送（T4）間の最小時間は2 SCK周期です。これは、1つのワードのSCKの最終アクティブ・エッジから、次のワードのSCKの最初のアクティブ・エッジまで測定します。これは、SPIの設定（CPHA、MSTRなど）とは無関係です。

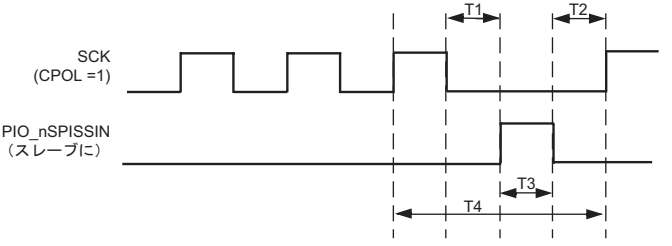


図 10-13.SPI のタイミング

CPHA = 0のマスター・デバイスでは、SCK周期の少なくとも2分の1の間は、スレーブ・セレクト出力は非アクティブ（ハイレベル）になります。この場合、T1とT2は、それぞれ常にSCK周期の2分の1に等しくなります。

第11章 パラレル・ペリフェラル・インターフェース

パラレル・ペリフェラル・インターフェース（PPI）は、16ビットまでのデータに対応できる半二重方式の双方向ポートです。PPIには専用のクロック・ピン、3本の多重フレーム同期ピン、4本の専用データ・ピンがあります。PFピンを再設定することで、12本までのデータ・ピンを追加使用できます。2つの8ビット・データ・サンプルを1つの16ビット・ワードとしてパックできるため、最高のシステム・スループットは8ビット・データで実現します。このような場合、以前のサンプルは8つの最下位ビット（LSB）に置かれます。

PPI_CLKピンは、SCLK/2までの外部クロック入力を受け付けることができます。クロックを内部的に供給することはできません。表11-1にPPIのピン・インターフェースを示します。

プログラマブル・フラグ・ピンがPPI用に設定された場合、プログラマブル・フラグMMR内のそのビット位置は0として読み返されます。

表 11-1. PPI ピン

信号名	機能	方向	代替機能
PPI15	データ	双方向	PF4、SPI イネーブル出力
PPI14	データ	双方向	PF5、SPI イネーブル出力
PPI13	データ	双方向	PF6、SPI イネーブル出力
PPI12	データ	双方向	PF7、SPI イネーブル出力
PPI11	データ	双方向	PF8
PPI10	データ	双方向	PF9
PPI9	データ	双方向	PF10

表 11-1. PPI ピン（続き）

信号名	機能	方向	代替機能
PPI8	データ	双方向	PF11
PPI7	データ	双方向	PF12
PPI6	データ	双方向	PF13
PPI5	データ	双方向	PF14
PPI4	データ	双方向	PF15
PPI3	データ	双方向	N/A
PPI2	データ	双方向	N/A
PPI1	データ	双方向	N/A
PPI0	データ	双方向	N/A
PPI_FS3	フレーム同期 3/ フィールド	双方向	PF3、SPI イネーブル出力
PPI_FS2	フレーム同期 2/ VSYNC	双方向	タイマ 2
PPI_FS1	フレーム同期 1/ HSYNC	双方向	タイマ 1
PPI_CLK	SLCK/2 まで	入力クロック	N/A

PPI レジスタ

PPIには、その動作を調整する5本のメモリマップド・レジスタ（MMR）があります。これらのレジスタは、PPIコントロール・レジスタ（PPI_CONTROL）、PPI ステータス・レジスタ（PPI_STATUS）、遅延カウント・レジスタ（PPI_DELAY）、転送カウント・レジスタ（PPI_COUNT）、ラインズ・パー・フレーム・レジスタ（PPI_FRAME）です。

以下の節では、これらのMMRの説明とビット図を示します。

■ PPI_CONTROL レジスタ

PPIコントロール・レジスタ（PPI_CONTROL）は、PPIの動作モード、制御信号極性、ポートのデータ幅を設定します。このMMRのビット図については、11-4ページの図11-1を参照してください。

POLCビットとPOLSビットは、それぞれPPI_CLK信号とPPI_FS1/PPI_FS2信号の選択的な信号反転を可能にします。これによって、幅広い制御信号極性をもつデータ・ソースとレシーバに接続するためのメカニズムが提供されます。通常、リモートのデータ・ソース／レシーバは、設定可能な信号極性も提供します。したがって、POLCビットとPOLSビットにより、その柔軟性が高まります。

DLEN[2:0]フィールドは、どのモードでもPPIポートの幅を指定するようにプログラムされます。なお、8～16ビットの任意の幅を利用できますが、9ビットのポート幅は例外です。DLEN設定の結果としてPPIでは使用されないPFピンは、通常のPF能力では自由に使用できます。

ITU-R 656モードでは、DLENフィールドに10ビットのポート幅を超えるものを設定しないでください。PPIが新たなピンを確保するため、他のペリフェラルからは使用できなくなります。

SKIP_ENビットがセットされると、PPIを通じて読み出されるデータ・エレメントの選択的なスキッピングがイネーブルにされます。データ・エレメントを無視することで、PPIはDMA帯域幅を節約できます。

SKIP_ENビットがセットされると、SKIP_EOビットによってPPIは入力データストリーム内の奇数エレメントまたは偶数エレメントを無視できます。これが役立つのは、たとえば、カラー・ビデオ信号をYCbCrフォーマット（Cb、Y、Cr、Y、Cb、Y、Cr、Y...）で読み出すときです。エレメントを1つおきにスキップすることで、PPIはLuma（Y）値またはChroma（CrまたはCb）値だけを読み出すことができます。また、2つのプロセッサを同じ着信ビデオ・ストリームに同期させるときにも役立ちます。1つのプロセッサでLuma処理を行い、もう1つのプロセッサ（最初のプロセッ

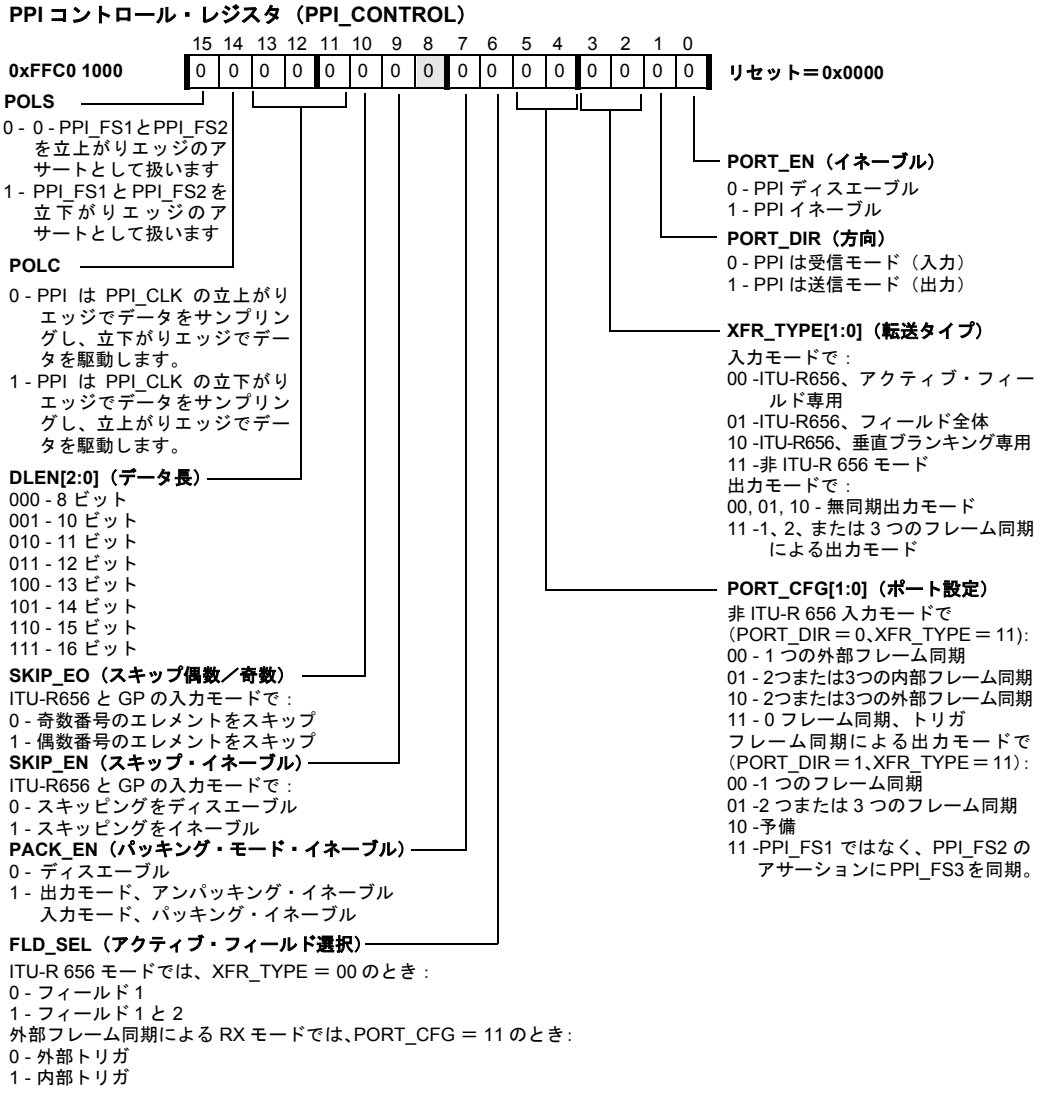


図 11-1. PPI コントロール・レジスタ

サとはSKIP_EOビットの設定が違います）でChroma処理を行うことができます。このスキッピング機能は、外部フレーム同期によるITU-R 656モードとRXモードで有効です。

PACK_EN ビットが意味を持つのは、PPIポート幅（DLEN[2:0]で選択）が8ビットの場合だけです。DMAバス上のPPI_CLKによって開始されたすべてのイベント（つまり、入力動作または出力動作）では、16ビットのエンティティを扱います。つまり、どのPPI_CLKでも10ビットの入力ポート幅では、依然として16ビットの入力ワードが得られます。上位6ビットは0です。同様に、8ビットのポート幅でも16ビットの入力ワードが得られ、上位8ビットはオール0です。8ビット・データの場合には、16ビット・ワードごとに2バイトのデータが存在するように、この情報をパックすると効率が上がるのが普通です。これはPACK_ENビットの機能です。このビットがセットされると、すべてのRXモードに対するパッキングがイネーブルにされます。

次のデータが、DMAを介してPPIに伝送されるとします。

0xCE, 0xFA, 0xFE, 0xCA....

- PACK_ENをセットした場合：

  8ビットのポート幅用に設定されたPPIに読み出されます。

  ```
  0xCE, 0xFA, 0xFE, 0xCA...
  ```

  DMA バスに転送されます。

  ```
  0xFACE, 0xCAFE, ...
  ```

- PACK_EN をクリアした場合：

  PPI に読み出されます。

  ```
  0xCE, 0xFA, 0xFE, 0xCA, ...
  ```

  DMA バスに転送されます。

  ```
  0x00CE, 0x00FA, 0x00FE, 0x00CA, ...
  ```

TX モードの場合、PACK_EN をセットすると、バイトのアンパッキングがイネーブルにされます。メモリ内の次のデータが、DMA を介して PPI を通じて転送出力されるとします。

```
0xFACE CAFE....
```

（0xFA と 0xCA は、それぞれの 16 ビット・ワードの 2 つの最上位ビット（MSB）です）。

- PACK_EN をセットした場合：

  PPI に DMA されます。

  ```
  0xFACE, 0xCAFE, ...
  ```

  8 ビットのポート幅用に設定された PPI を通じて転送出力されます（なお、LSB が最初に転送されます）。

  ```
  0xCE, 0xFA, 0xFE, 0xCA, ...
  ```

- PACK_EN をクリアした場合：

  PPI に DMA されます。

  ```
  0xFACE, 0xCAFE, ...
  ```

  8 ビットのポート幅用に設定された PPI を通じて転送出力されます。

  ```
  0xCE, 0xFE, ...
  ```

FLD_SELビットは、主にITU-R 656のアクティブ・フィールド専用モードで使用されます。FLD_SELビットは、各ビデオ・フレームのフィールド1にだけ転送するのか、それともフィールド1と2の両方に転送するのかを決定します。したがって、アクティブ・ビデオのおよそ半数のフィールドにのみ転送することによって、DMA帯域幅の節約を可能にします。

PORT_CFG[1:0]フィールドは、PPIの動作モードの設定に使用されます。このフィールドは、ポートのデータ転送方向を設定するPORT_DIRビットと連携して機能します。後述するように、XFR_TYPE[1:0]フィールドも動作モードの設定に使用されます。PPIの可能な動作モードについては、表11-2を参照してください。

表 11-2. PPI の可能な動作モード

PPI モード	同期の数	PORT_DIR	PORT_CFG	XFR_TYPE	POLC	POLS	FLD_SEL
RX モード、0 フレーム同期、外部トリガ	0	0	11	11	0または1	0または1	0
RX モード、0 フレーム同期、内部トリガ	0	0	11	11	0または1	0または1	1
RXモード、1つの外部フレーム同期	1	0	00	11	0または1	0または1	X
RX モード、2 つまたは 3 つの外部フレーム同期	3	0	10	11	0または1	0または1	X
RX モード、2 つまたは 3 つの内部フレーム同期	3	0	01	11	0または1	0または1	X
RX モード、ITU-R 656、アクティブ・フィールド専用	組込み	0	XX	00	0または1	0	0または1
RX モード、ITU-R 656、垂直ブランキング専用	組込み	0	XX	10	0または1	0	X
RX モード、ITU-R 656、フィールド全体	組込み	0	XX	01	0または1	0	X
TX モード、0 フレーム同期	0	1	XX	00、01、10	0または1	0または1	X

表 11-2.　PPI の可能な動作モード（続き）

PPI モード	同期の数	PORT_DIR	PORT_CFG	XFR_TYPE	POLC	POLS	FLD_SEL
TX モード、1 つの内部または外部フレーム同期	1	1	00	11	0または1	0または1	X
TXモード､2つの外部フレーム同期	2	1	01	11	0または1	0または1	X
TX モード、2 つまたは 3 つの内部フレーム同期､FS3 を FS1 のアサーションに同期	3	1	01	11	0または1	0または1	X
TX モード、2 つまたは 3 つの内部フレーム同期､FS3 を FS2 のアサーションに同期	3	1	11	11	0または1	0または1	X

XFR_TYPE[1:0] フィールドは、PPI のさまざまな動作モードを設定します。XFR_TYPE[1:0] が PPI_CONTROL の他のビットと連携して PPI の動作モードを決定する詳細については、表 11-2 を参照してください。

PORT_EN ビットがセットされると、PPI の動作をイネーブルにします。

なお、入力ポートとして設定された場合、適切な同期信号が受信されるまでは PPI をイネーブルにしてもデータ転送は開始されません。出力ポートとして設定された場合には、フレーム同期（タイマ・ユニット）がイネーブルにされるとすぐに転送（適切な同期信号を含めて）が始まります。したがって、それまでに、すべてのフレーム同期を設定する必要があります。詳細については11-30 ページの「GP モードでのフレーム同期」を参照してください。

■ PPI_STATUS レジスタ

PPIステータス・レジスタ（PPI_STATUS）には、PPIの現在の動作状態についての情報を提供するビットが含まれています。

読出し時にはレジスタ全体がクリアされます。したがって、ステータス・ワードを解析して、セットされたビットを判断する必要があります。

ERR_DETビットは、ITU-R 656の制御ワード・プリアンブルでエラーが検出されたかどうかを示すスティッキー・ビットです。このビットは、ITU-R 656モードでのみ有効です。ERR_DET = 1の場合には、プリアンブルでエラーが検出されました。ERR_DET = 0の場合には、プリアンブルでエラーは検出されませんでした。

ERR_NCORビットはスティッキーであり、ITU-R 656モードでのみ関係します。ERR_NCOR = 0でERR_DET = 1の場合には、発生したすべてのプリアンブル・エラーが訂正されました。ERR_NCOR = 1の場合には、プリアンブルでエラーは検出されましたが、訂正されませんでした。SIC_IMASKレジスタでこの条件がマスクされていない限り、この状況ではPPIエラー割込みが生成されます。

FT_ERRビットはスティッキーであり、セットされると、フレーム・トラック・エラーが発生したことを示します。これはRXモードでのみ有効です。この条件では、PPI_FRAMEにプログラムされたフレーム当たりのライン数は「フレーム開始検出」条件とは調和しません（11-13ページの情報メモを参照）。SIC_IMASKレジスタでこの条件がマスクされていない限り、フレーム・トラック・エラーによってPPIエラー割込みが生成されます。

FLDビットは、F（ITU-R 656モード）またはPPI_FS3（他のRXモード）の状態の変化と同時に、セットまたはクリアされます。これは入力モードでのみ有効です。FLDの状態は、F信号またはPPI_FS3信号の現在の状態を反映します。つまり、FLDビットは、PPIによって処理されている現在のビデオ・フィールドを常に反映します。

OVR ビットはスティッキーであり、セットされると、PPI FIFO がオーバーフローしてそれ以上のデータを受け付けられないことを示します。SIC_IMASK レジスタでこの条件がマスクされていない限り、FIFO オーバーフロー・エラーによって PPI エラー割込みが生成されます。

PPI FIFO は 16 ビット幅であり、16 のエントリがあります。

UNDR ビットはスティッキーであり、セットされると、PPI FIFO がアンダーランしてデータ不足であることを示します。SIC_IMASK レジスタでこの条件がマスクされていない限り、FIFO アンダーラン・エラーによって PPI エラー割込みが生成されます。

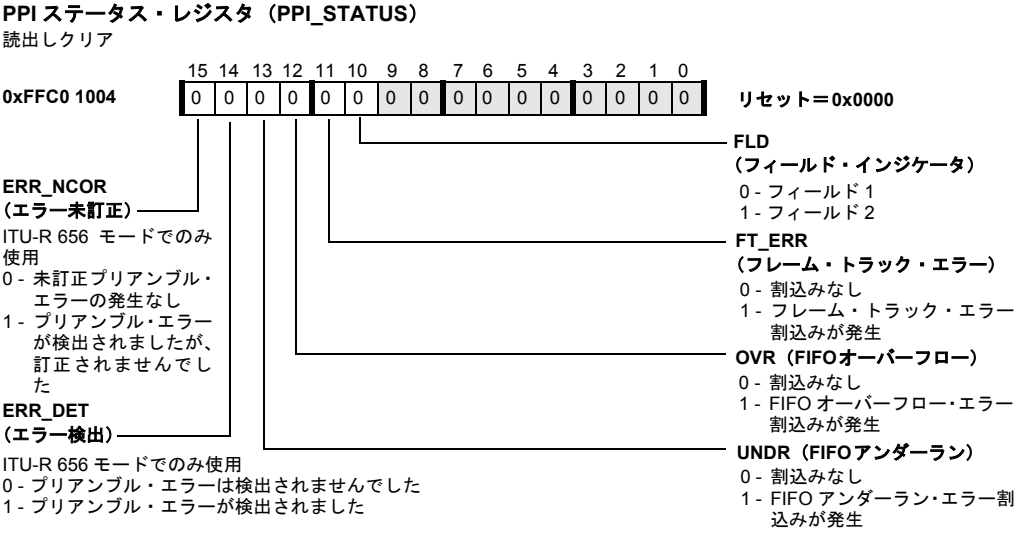


図 11-2. PPI ステータス・レジスタ

■ PPI_DELAY レジスタ

遅延カウント・レジスタ（PPI_DELAY）は、0フレーム同期によるITU-R 656モードとGPモードを除いて、すべての設定で使用できます。このレジスタには、PPI_FS1のアサーションからデータの読出しや書込みが開始されるまでに生じる遅延のPPI_CLKサイクル数が含まれています。

なお、少なくとも 1 つのフレーム同期を使用する TX モードでは、PPI_DELAY レジスタで指定された値を越えて、1サイクルの遅延があります。

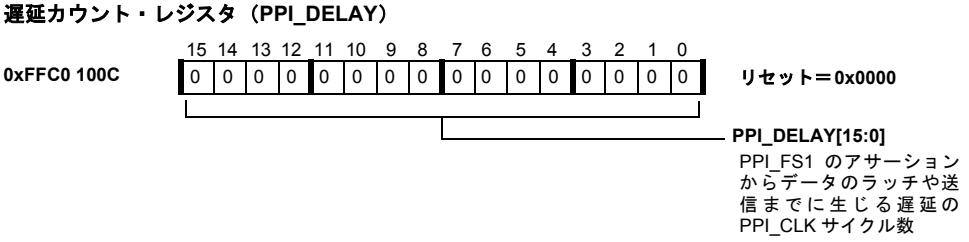


図 11-3. 遅延カウント・レジスタ

■ PPI_COUNT レジスタ

転送カウント・レジスタ（PPI_COUNT）は、繰り返しハードウェア・フレーム同期（外部的または内部的に生成）が発生する場合にのみ使用され、ITU-R 656モードや0フレーム同期によるモードでは必要ありません。RXモードの場合、このレジスタは、ライン当たりでPPIに読み出すサンプル数－1を保持します。TXモードの場合、ライン当たりでPPIを通じて書き込むサンプル数－1を保持します。レジスタ自身は、実際には転送ごとに

デクリメントしません。したがって、新ラインのデータの初めには、このレジスタの値を書き換える必要はありません。たとえば、PPIを通じて100個のサンプルを受信または送信するには、PPI_COUNTを99に設定します。

PPI_COUNTにプログラムされたサンプル数と、PPI_FS1によって指定された「水平」期間中に予想されるサンプル数とが一致するように注意してください。

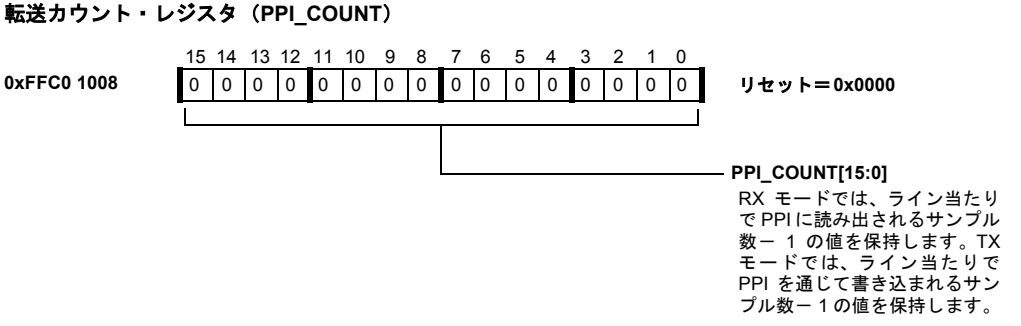


図 11-4. 転送カウント・レジスタ

■ PPI_FRAME レジスタ

ラインズ・パー・フレーム（PPI_FRAME）レジスタは、2つまたは3つの外部フレーム同期によるTXモードと、2つまたは3つのフレーム同期によるすべてのRXモードで使用されます。このレジスタは、データのフレーム当たりで予想されるライン数を保持します。通常、フレームはビデオの感覚で、奇数フィールドおよび偶数フィールドとして定義されます（フィールド1とフィールド2、PPI_FS3またはITU-R 656 F信号によって指定）。しかし、2つのフレーム同期しか使用できない場合には、フレームは、2つのフレーム同期の同時期のアサーションとすることができます。ラインは、完全なPPI_FS1サイクル、つまり完全なITU-R 656 SAV-EAVサイクルと定義されます。

PPI_FRAMEによって指定されたライン数が転送される前に「フレーム開始検出」が発生した場合には、フレーム・トラック・エラーが生じて、PPI_STATUSのFT_ERRビットがセットされます。しかし、PPIは、PPI_FRAMEにプログラムされた値に自動的に再初期化されて、データ転送が続行されます。PPIは、外部フレーム開始検出条件に合わせて再同期されます。

ITU-R 656モードでは、フレーム開始検出はフィールド・インジケータであるFの立下がりエッジで発生します。これはフィールド1の先頭で発生します。

3つの外部フレーム同期によるRXモードでは、フレーム開始検出は、PPI_FS3がローレベルである間に、PPI_FS2のアサーションに続いてPPI_FS1のアサーションが行われるときにおきます。これはフィールド1の先頭で発生します。PPI_FS3は、PPI_FS2がアサートされる時ではなく、PPI_FS1がアサートされる場合でのみ、ローレベルである必要があります。また、PPI_FS3は、PPIがイネーブルになった後の初期のフレームの開始によってのみ、同期動作として使用されます。その後は無視されます。

3つの外部フレーム同期によるRXモードを使用し、必要な同期は2つだけである場合には、PPIを3フレーム同期動作用に設定し、PPI_FS3ピンにGNDへの外部プルダウンを提供します。

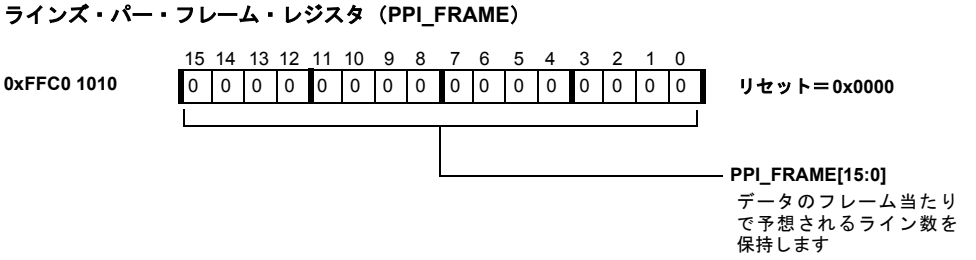


図 11-5. ラインズ・パー・フレーム・レジスタ

ITU-R 656 モード

PPIは、ITU-R 656フレームのデータに対して3つの入力モードを提供します。ここでは、これらのモードについて説明します。PPIはITU-R 656出力モードには明示的に対応しませんが、PPIをこの状況で使用するための推奨事項も提供します。

■ ITU-R 656 の背景

ITU-R 656勧告（以前の名前はCCIR-656）に基づいて、デジタル・ビデオ・ストリームには、525/60（NTSC）システムと625/50（PAL）システムに対する特性があります（11-15ページの図11-6と11-16ページの図11-7を参照）。プロセッサは、ITU-R 656のビット・パラレル・モードだけを提供します。8ビットと10ビットのビデオ・エレメント幅に対応します。

このモードでは、水平（H）信号、垂直（V）信号、フィールド（F）信号は制御ワードを形成する一連のバイトの形で、ビデオ・データストリームの組込み部分として送信されます。Start of Active Video（SAV）信号とEnd of Active Video（EAV）信号は、各ラインに読み出されるデータ・エレメントの先頭と最後を示します。SAVはHの1-0遷移で発生し、EAVはHの0-1遷移で始まります。ビデオのフィールド全体は、アクティブ・ビデオ＋水平ブランキング（EAVとSAVコードとの間のスペース）と垂直ブランキング（V＝1であるスペース）で構成されます。ビデオのフィールドは、Fビットの遷移で始まります。「奇数フィールド」はF＝0の値で示され、偶数フィールドはF＝1の値で示されます。プログレッシブ・ビ

デオは、フィールド1とフィールド2を区別しません。一方、インターレース・ビデオは、各フィールドの1つおきの行を結合して実際のビデオ・イメージを作成するため、各フィールドを一意的に扱う必要があります。

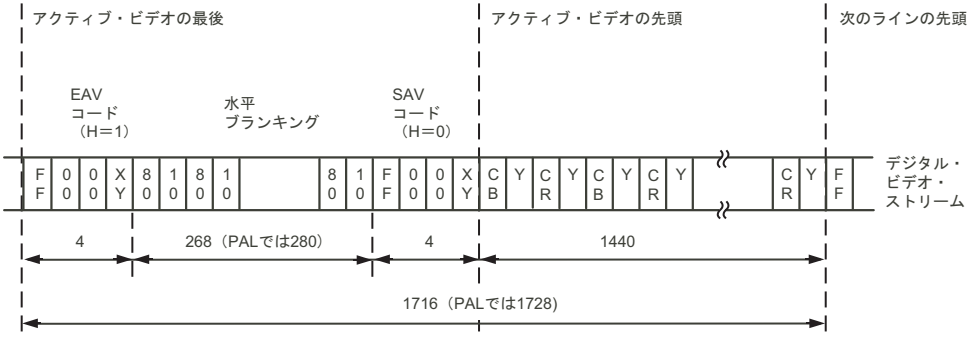


図 11-6. NTSC（PAL）システム用の ITU-R 656 8 ビット・パラレル・データ・ストリーム

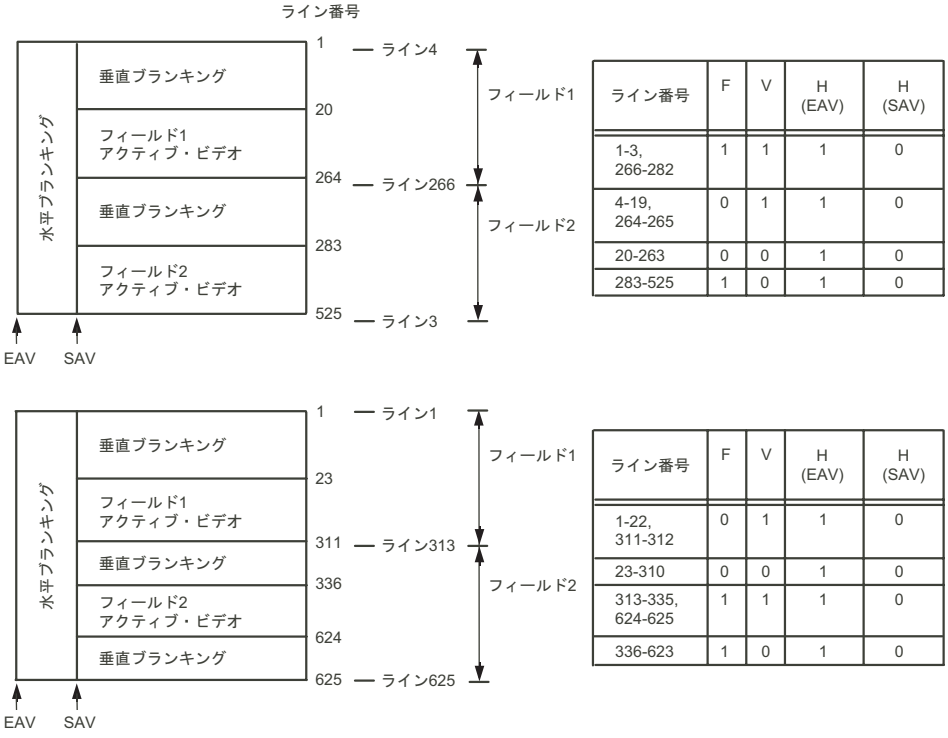


図 11-7. ITU-R BT.656-4 用 NTSC/PAL システムの代表的な ビデオ・フレーム分割

SAVコードとEAVコードの詳細を表 11-3に示します。なお、3バイトの定義済みプリアンブル（0xFF、0x00、0x00）に続いて、XYステータス・ワードがあります。XYステータス・ワードには、F（フィールド）ビット、V（垂直ブランキング）ビット、H（水平ブランキング）ビットに加えて、1ビットのエラー検出／訂正用の4つの保護ビットも含まれています。なお、FとVは、EAVシーケンスの一部としてのみ変化が可能です（つまり、H = 0からH = 1への遷移）。ビット定義は次のとおりです。

- フィールド1に対してはF = 0
- フィールド2に対してはF = 1

- 垂直ブランキング中は`V = 1`
- 垂直ブランキング中でなければ`V = 0`
- SAVでは`H = 0`
- EAVでは`H = 1`
- `P3 = V XOR H`
- `P2 = F XOR H`
- `P1 = F XOR V`
- `P0 = F XOR V XOR H`

多くのアプリケーションでは、標準のNTSC/PALフォーマット以外のビデオ・ストリーム（たとえば、CIF、QCIF）を採用できます。このため、プロセッサのインターフェースには、さまざまな行長とフィールド長に対応できるだけの十分な柔軟性があります。一般に、着信ビデオのEAV/SAVコードが適切である限り、PPIではそれを読み出せます。つまり、CIFイメージを「656対応」にフォーマットできます。この場合、EAV値とSAV値でラインごとのイメージの範囲を定義し、`V`コードと`F`コードを使用して、フィールドとフレームを区切ることができます。

表 11-3. 8/10 ビット ITU-R 656 ビデオ用のコントロール・バイト・シーケンス

	8 ビット・データ								10 ビット・データ	
	D9 (MSB)	D8	D7	D6	D5	D4	D3	D2	D1	D0
プリアンブル	1	1	1	1	1	1	1	1	1	1
	0	0	0	0	0	0	0	0	0	0
	0	0	0	0	0	0	0	0	0	0
コントロール・バイト	1	F	V	H	P3	P2	P1	P0	0	0

■ ITU-R 656 入力モード

図 11-8に、ITU-R 656入力モードでの一般的なデータ移動を示します。この図では、クロックCLKはビデオ・ソースによって提供されるか、システムによって外部から提供されます。

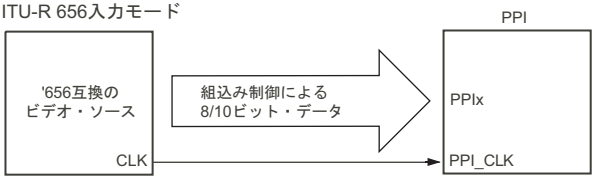


図 11-8. ITU-R 656 入力モード

ITU-R 656入力に対しては、フィールド全体、アクティブ・ビデオ専用、垂直ブランキング期間専用という、3つのサブモードがサポートされます。図 11-9に、これら3つのサブモードを示します。

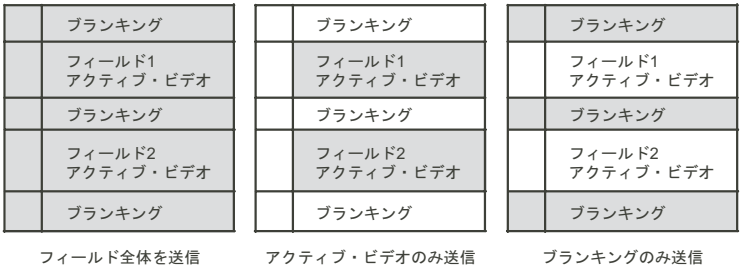


図 11-9. ITU-R 656 入力サブモード

▶ フィールド全体

このモードでは、着信ビットストリームの全体がPPIを通じて読み出されます。これには、アクティブ・ビデオだけではなく、水平／垂直ブランキング期間に組み込まれる補助データやコントロール・バイト・シーケンスも含まれます。データ転送は、フィールド1への同期が行われた直後に開始されますが、F＝0割当てを含む最初のEAVコードは組み込まれていません。

PPIをイネーブルにした後で転送される最初のラインには、その最初の4バイト・プリアンブルが欠落しています。しかし、それ以降のラインとフレームには、すべての制御コードが完全に備わっています。

このモードの1つの副次的な利点として、「ループバック」機能のイネーブルが挙げられます。この機能では、1～2フレームのデータをPPIを通じて読み出し、その後、互換性のあるビデオ表示デバイスに出力できます。もちろん、これにはPPIピンでの多重化が必要になりますが、PPIとの間で656データを読出し／書込みできることを確認する便利な方法が提供されます。

▶ アクティブ・ビデオ専用

このモードが使用されるのは、フィールドのアクティブ・ビデオ部分だけが対象であり、ブランキング期間は対象でない場合です。PPIでは、EAVとSAVとの間のすべてのデータだけでなく、V＝1の場合に存在するすべてのデータも無視されます（読み出されません）。このモードでは、コントロール・バイト・シーケンスはメモリに格納されず、PPIによって除外されます。フィールド1の先頭に同期した後、PPIはSAVを認識するまでは、着信サンプルを無視します。

このモードでは、ユーザは`PPI_FRAME` MMRにフレーム当たりの合計（アクティブ＋垂直ブランキング）ライン数を指定します。

▶ 垂直ブランキング期間（VBI）専用

このモードでは、V＝1がコントロール・バイト・シーケンス内にある間だけ、データ転送はアクティブです。これは、ビデオ・ソースが垂直ブランキング期間（VBI）の中央にあることを示し、補助データ伝送に使用されることがあります。ITU-R 656勧告では、これらの補助データ・パケットのフォーマットを指定しますが、PPIにはパケット自身をデコードする機能はありません。この作業はソフトウェアで行う必要があります。水平ブランキング・データはログに記録され、そこでVBIの行と合流します。コントロール・バイト・シーケンス情報は、常にログに記録されます。ユーザは、`PPI_FRAME` MMRにフレーム当たりの合計ライン数（アクティブ＋垂直ブランキング）を指定します。

なお、VBIは、各フィールド内で2つの領域に分割されます。PPIの立場からは、この2つの別個の領域は1つの連続的な空間と見なされます。しかし、フレーム同期はフィールド1の先頭から始まることを忘れないでください。これは、必ずしも垂直ブランキングの先頭に対応するとは限りません。たとえば、525/60システムでは、フィールド1の先頭（F＝0）はVBIのライン4に対応します。

■ ITU-R 656 出力モード

PPIは、ITU-R 656出力ストリームを適切なプリアンブルとブランキング期間でフレーム化する機能を明示的には提供しません。しかし、0フレーム同期によるTXモードは、このプロセスを手動で行うことができます。基本的に、このモードでは、PPIを通じてメモリからのストリーミング動作を提供します。データと制御コードは、ビデオ・ストリームを送信する前にメモリ内で設定できます。2D DMAエンジンを使用すれば、これはいろいろな方法で実行できます。たとえば、必要ならば、1ラインのブランキング（H＋V）をバッファに格納してDMAコントローラによってN回送信してから、DMAアクティブ・ビデオに進むことができます。あるいは、1つのフィールド全体（制御コードとブランキング付き）をバッファ内で静的に設定すると同時に、DMAエンジンは、アクティブ・ビデオ領域だけをフレームごとにバッファに転送することができます。

■ ITU-R 656 モードでのフレーム同期

ITU-R 656モードでの同期は、常にフィールド・インジケータFの立下がりエッジで発生します。これは、フィールド1の先頭に対応します。したがって、データがPPIに受信されるまでに、2つまでのフィールドを無視できます（たとえば、PPI—カメラ・チャンネルが確立される前にフィールド1が開始された場合）。

ITU-R 656モードでは、すべてのH信号とV信号がデータストリームに組み込まれているため、`PPI_COUNT`レジスタは必要ありません。しかし、`PPI_FRAME`レジスタは、同期エラーをチェックするために使用されます。

ユーザは、このMMRを期待されるライン数にわたって各フレームのビデオにプログラムします。PPIでは、フレームの開始からフレーム終了条件（F＝1からF＝0への遷移）がデコードされるまでに発生するEAV—SAVの遷移数を記録します。この時点で、処理された実際のライン数がPPI_FRAME内の値と比較されます。一致しない場合には、PPI_STATUSレジスタのFT_ERRビットがアサートされます。たとえば、SAV遷移が行われなかった場合には、現在のフィールドにはNUM_ROWS - 1の行しかありません。しかし、次のフレームの先頭で再同期が行われます。

フィールド全体の受信が完了すると、PPI_STATUSレジスタでフィールド・ステータス・ビットがトグルされます。このようにして、割込みサービス・ルーチン（ISR）は、どのフィールドが読み出されたかを認識できます。

汎用 PPI モード

汎用（GP）PPIモードは、多種多様なデータ・キャプチャ／伝送アプリケーション向けに用意されています。表11-4に、これらのモードを要約します。特定のモードで特定のPPI_FSxフレーム同期が使用されていない場合、ピンは、その多重化された代替プロセッサ機能に使用できる（つまり、タイマ・ピンやフラグ・ピンとして）ことを意味します。これに対する例外として、PPIが2フレーム同期モード用に設定されている場合、PPI_FS3は、たとえPPIによって使用されていなくても、汎用フラグとして使用できません。

表 11-4.　汎用 PPI モード

GP PPI モード	PPI_FS1 方向	PPI_FS2 方向	PPI_FS3 方向	データ方向
RXモード、0フレーム同期、外部トリガ	入力	未使用	未使用	入力
RXモード、0フレーム同期、内部トリガ	未使用	未使用	未使用	入力
RX モード、1 つの外部フレーム同期	入力	未使用	未使用	入力

表 11-4. 汎用 PPI モード（続き）

GP PPI モード	PPI_FS1 方向	PPI_FS2 方向	PPI_FS3 方向	データ方向
RX モード、2 つまたは 3 つの外部フレーム同期	入力	入力	入力	入力
RX モード、2 つまたは 3 つの内部フレーム同期	出力	出力	出力	入力
TX モード、0 フレーム同期	未使用	未使用	未使用	出力
TX モード、1 つの外部フレーム同期	入力	未使用	未使用	出力
TX モード、2 つの外部フレーム同期	入力	入力	出力	出力
TX モード、1 つの内部フレーム同期	出力	未使用	未使用	出力
TX モード、2 つまたは 3 つの内部フレーム同期	出力	出力	出力	出力

図 11-10は、GPモードの一般的なフローを示します。図の上部には、1つの外部フレーム同期によるRXモードの例を示します。PPIは、ハードウェア・フレーム同期パルス（PPI_FS1）を受信した後で、PPI_DELAYにプログラムされたPPI_CLKサイクルだけ遅延します。続いてDMAコントローラは、PPI_COUNTによって指定されたサンプル数を転送します。この後で、しかも次のPPI_FS1フレーム同期が到着するまでに到着するすべてのサンプルは無視され、DMAバスに転送されません。

指定されたPPI_COUNTのサンプルが読み出される前に次のPPI_FS1フレーム同期が到着した場合には、サンプル・カウンタは 0 に再初期化されて、再び PPI_COUNT までのカウント・アップを開始します。この場合、DMA チャンネル設定は PPI 転送プロセスとの同期を失うことがあります。

図 11-10の下部に、1つの内部フレーム同期によるTXモードの例を示します。PPI_FS1がアサートされた後、1 PPI_CLKサイクルの遅延があり、続いてPPI_DELAYにプログラムされたPPI_CLKサイクルにわたる遅延があります。次に、DMAコントローラは、PPI_COUNTによって指定されたサンプル数を転送します。次のPPI_FS1同期とプログラムされた遅延が発生するまでは、これ以上のDMAは行われません。

指定されたPPI_COUNTのサンプルが転送される前に、次のPPI_FS1フレーム同期が到着した場合、同期が優先されて、新しいライン転送シーケンスが開始されます。この場合、DMA チャンネル設定はPPI 転送プロセスとの同期を失うことがあります。

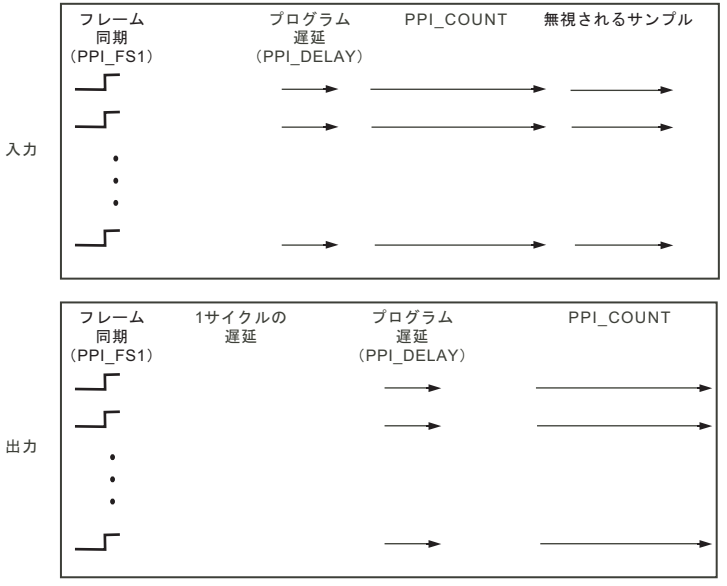


図 11-10. GP モードの一般的なフロー
（PPI_FS1 のポジティブ・アサーションを想定）

■ データ入力（RX）モード

PPIは、いくつかのデータ入力モードをサポートします。これらのモードの違いは、主にデータのフレーム化方法です。モードごとのPPIの設定方法については、11-7ページの表11-2を参照してください。

▶ フレーム同期なし

これらのモードは、受信データのフレーム化に周期的なフレーム同期が生成されないアプリケーション群をカバーします。データ転送を開始するには2つのオプションがあり、いずれもPPI_CONTROLレジスタによって設定されます。

- 外部トリガ：FLD_SEL = 0でPORT_CFG = b#11の場合、外部ソースはトランザクションの開始時に1つのフレーム同期（PPI_FS1に接続）を送信します。
- 内部トリガ：ソフトウェアは、FLD_SEL = 1およびPORT_CFG = b#11でPORT_EN = 1に設定してプロセスを開始します。

それ以降のすべてのデータ操作はDMAによって行われます。たとえば、交互に使用される1Kのメモリ・バッファを設定できます。一方が満杯になると、DMAは2番目のバッファで続行します。それと同時に、別のDMA動作によって、最初のメモリ・バッファは再利用のためにクリアされます。

フレーム同期なしのRXモードでのクロック・ドメイン同期のために、モードがイネーブルになってから有効データを受信するまでに2 PPI_CLKサイクル以上の遅延が生じることがあります。このため、有効データの開始の検出はソフトウェアで管理してください。

▶ 1 つ、2 つ、または 3 つの外部フレーム同期

1同期モードは、A/Dコンバータ（ADC）アプリケーション向けに用意されています。図 11-11の上部に、このモード用の代表的なシステム・セットアップを示します。

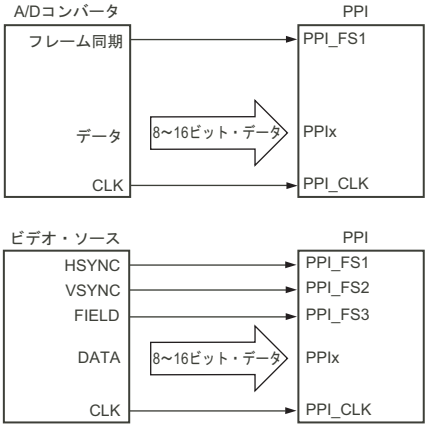


図 11-11. RX モード、外部フレーム同期

図 11-11の下部に示す3同期モードは、ITU-R 601勧告に基づいてハードウェア・シグナリング（HSYNC、VSYNC、FIELD）を使用するビデオ・アプリケーションに対応します。このモードでのフレーム同期用のマッピングは、PPI_FS1 = HSYNC、PPI_FS2 = VSYNC、PPI_FS3 = FIELDです。このモードでのフレーム同期の詳細については、11-30ページの「GPモードでのフレーム同期」を参照してください。

3同期モードで設定された場合、2同期モードは、外付け抵抗によってPPI_FS3をGNDにプルすることで暗黙的に利用できます。

▶ 2 つまたは 3 つの内部フレーム同期

このモードは、マスター・プロセッサへのスレーブとする、ビデオ・ソースへのインターフェースに役立つことがあります。つまり、プロセッサは、PPI_FS1 と PPI_FS2 をアサートしてから PPI にデータを読み出すことによって、ビデオ・ソースからの読出しタイミングを制御します。PPI_FS3 フレーム同期は、どのフィールドが現在転送されているかを示しますが、これは出力であるため、使用しない場合はフローティングのままにしておけます。図 11-12 に、このモードのサンプル・アプリケーションを示します。

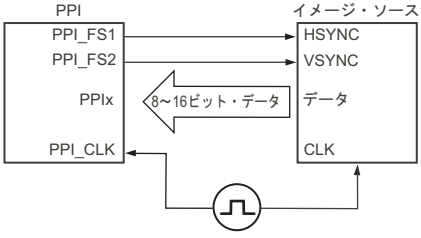


図 11-12. RX モード、内部フレーム同期

■ データ出力（TX）モード

PPIは、いくつかのデータ出力モードを提供します。これらのモードの違いは、主にデータのフレーム化方法です。モードごとのPPIの設定方法については、11-7ページの表11-2を参照してください。

▶ フレーム同期なし

このモードは、DMAコントローラによって指定されたデータ・ブロックがフレーミングなしでPPIを通じて送信されます。つまり、DMAチャンネルが設定されてイネーブルにされ、PPIが設定されてイネーブルにされると、データ転送はPPI_CLKに同期してすぐに開始されます。このモードについては、図 11-13を参照してください。

このモードでは、PPIのイネーブルと有効データの伝送との間に最大16 SCLKサイクル（＞8ビット・データの場合）または32 SCLKサイクル（8ビット・データの場合）の遅延があります。さらに、DMAは少なくとも16サンプル（＞8ビット・データの場合）または32サンプル（8ビット・データの場合）を送信するように設定する必要があります。

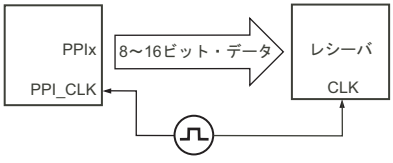


図 11-13. TX モード、0 フレーム同期

▶ 1 つまたは 2 つの外部フレーム同期

これらのモードでは、外付けレシーバは、PPIから送信されたデータをフレーム化することができます。1同期と2同期の両方のモードが利用できます。図 11-14の上部の図は1同期の場合を示し、下部の図は2同期モードを示します。

外部フレーム同期のアサーションと PPI を通じての有効データの転送出力との間には、1.5 PPI_CLK サイクル＋ PPI_DELAY にプログラムされた値という必須遅延があります。

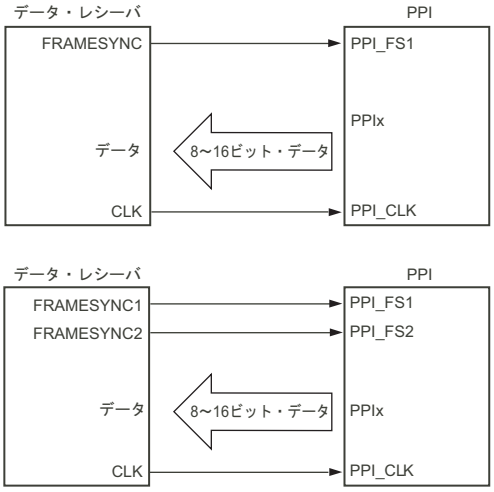


図 11-14. TX モード、1 つまたは 2 つの外部フレーム同期データ

▶ 1 つ、2 つ、または 3 つの内部フレーム同期

1同期モードは、単一フレーム同期によるD/Aコンバータ（DAC）へのインターフェース向けに用意されています。11-30ページの図11-15の上部に、このタイプの接続例を示します。

3同期モードは、図 11-15の下部に示すように、ビデオやグラフィックス表示への接続に役立ちます。この場合、2同期モードは、PPI_FS3を未接続のままにしておくことで暗黙的に利用できます。

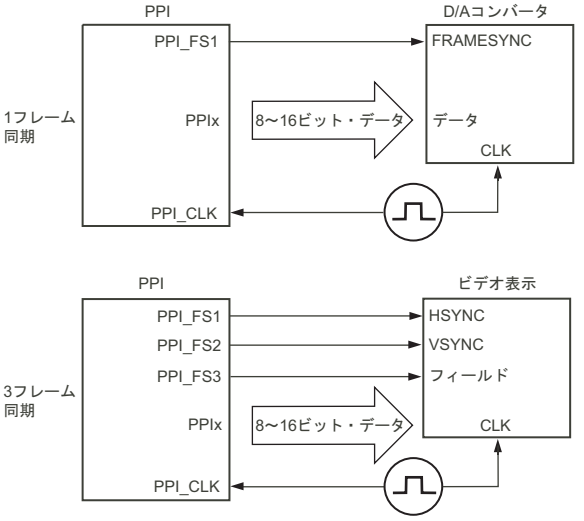


図 11-15. PPI GP 出力

■ GP モードでのフレーム同期

GPモードでのフレーム同期は、内部フレーム同期によるモードと外部フレーム同期によるモードでは動作が異なります。

▶ 内部フレーム同期によるモード

内部フレーム同期によるモードでは、PPI_FS1とPPI_FS2はそれぞれ、タイマ1とタイマ2のパルス幅変調（PWM）回路に直接リンクされます。これにより、既存のTIMERxレジスタを使用して、これらの信号に対して任意のパルス幅と周期をプログラムできます。この機能は、広範囲のタイミング・ニーズに対処します。なお、これらのPWM回路は、通常のタイマPWM動作時のSCLKやPF1によってではなく、PPI_CLKによって駆動されます。設定されたPPIモードでPPI_FS2が使用されない場合には、タイマ2は機能に制限を受けることなく通常どおりに動作します。PPI_FS3の状態は、PPI_FS1またはPPI_FS2、あるいはその両方の状態に完全に依存します。したがって、PPI_FS3には固有のプログラマビリティはありません。

PPI_FS1やPPI_FS2に内部フレーム同期モードでの動作をプログラムするには：

1. DMA を PPI 用に設定してイネーブルにします。11-33 ページの「DMA 動作」を参照してください。
2. TIMER1_WIDTH と TIMER1_PERIOD（PPI_FS1 の場合）または TIMER2_WIDTH と TIMER2_PERIOD（PPI_FS2 の場合）を介して、フレーム同期信号ごとに幅と周期を設定します。
3. TIMER1_CONFIG を PWM_OUT モードに設定します（PPI_FS1 の場合）。使用する場合には、TIMER2_CONFIG を PWM_OUT モードに設定します（PPI_FS2 の場合）。これには、タイマごとの CLK_SEL = 1 と TIN_SEL = 1 の設定も含まれます。
4. PPIを設定してイネーブルにするため、PPI_CONTROLに書き込みます。
5. タイマ 1 やタイマ 2 をイネーブルにするため、TIMER_ENABLE に書き込みます。

PPIとタイマ・ペリフェラルとの間には、適切なフレーム同期極性を保証することが重要です。そのために、PPI_CONTROL[15:14] = b#10 またはb#11 の場合は、TIMER1_CONFIG と TIMER2_CONFIG で PULSE_HI ビットがクリアされていることを確認してください。同

様に、PPI_CONTROL[15:14] = b#00 またはb#01 の場合には、TIMER1_CONFIG と TIMER2_CONFIG で PULSE_HI ビットをセットしてください。

内部フレーム同期を必要としない別の PPI モードに切り替えるには:

1. PPI をディスエーブルにします（PPI_CONTROL を使用）。
2. タイマをディスエーブルにします（TIMER_DISABLE を使用）。

▶ 外部フレーム同期によるモード

外部フレーム同期による RX モードでは、PPI_FS1 ピンと PPI_FS2 ピンがエッジ・センシティブ入力になります。このようなモードでは、TMR1 ピンと TMR2 ピンを必要としない目的に対して、タイマ 1 とタイマ 2 を使用できます。しかし、PPI が PPI_FSx のフレーム同期入力機能に TMRx ピンを使用している場合には、TMRx ピンへのタイマ・アクセスはディスエーブルにされます。PPI_FS2 を要求しないモードの場合、タイマ 2 はその機能を制約されず、あたかも PPI が使用されていないかのように動作することができます（つまり、TMR2 ピンはタイマとしても使用可能になります）。タイマの設定と使い方の詳細については、第 15 章「タイマ」を参照してください。

3 つの外部フレーム同期による RX モードでは、PPI_FS3 がローレベルである間に PPI_FS2 のアサーションに続いて PPI_FS1 のアサーションが行われると、フレーム開始の検出が行われます。これはフィールド 1 の先頭で発生します。PPI_FS3 は、PPI_FS2 がアサートされる時ではなく、PPI_FS1 がアサートされる場合でのみ、ローレベルである必要があります。また、PPI_FS3 は、PPI がイネーブルになった後の初期のフレームの開始によってのみ、同期動作として使用されます。その後は無視されます。

外部フレーム同期による TX モードでは、PPI_FS1 ピンと PPI_FS2 ピンはエッジ・センシティブ入力として扱われます。このモードでは、フレーム同期に関係するタイマを入力として設定したり、TIMER_ENABLE レジスタに

よってイネーブルにする必要はありません。さらに、たとえタイマ・ピンがPPIによって引き継がれた場合でも、実際のタイマ自身は使用できます。この場合、タイムベース（TIMERx_CONFIGのTIN_SELによって設定）がPPI_CLKである必要はありません。

ただし、使用するタイマのピンが外部フレーム同期に接続されている場合には、TIMERx_CONFIGのOUT_DISビットでそのピンをディスエーブルにしてください。このようにすれば、このモードでもPPI動作に影響を与えることなく、タイマそのものを非PPI向けに設定し、使用可能にすることができます。詳細については、第15章「タイマ」を参照してください。

DMA 動作

PPIは、プロセッサのDMAエンジンと共に使用する必要があります。ここでは、この2つの関係を説明します。DMAエンジンの詳細については、DMAレジスタとDMA動作の説明も含めて、第9章「ダイレクト・メモリ・アクセス」を参照してください。

PPI DMAチャンネルは、送信または受信動作用に設定することができ、最大スループットは(PPI_CLK)×(16ビット/転送)となります。データ長が8ビットを超えるモードでは、PPI_CLKサイクル当たりただ1つのエレメントを駆動できるため、パッキングが不可能となって帯域幅が減少します。最高のスループットは、8ビット・データとPACK_EN = 1（パッキング・モード・イネーブル）で達成されます。なお、16ビットのパッキング・モードでは、偶数のデータ・エレメントが必要です。

PPIインターフェースを使用するには、PPIのDMAチャンネルを設定する必要があります。行、フレーム、または部分フレームの転送完了と同時に割込みを生成するのはDMAエンジンです。また、PPIを通じて転送されるデータのソース・ポイントやデスティネーション・ポイントを調整するのもDMAエンジンです。

プロセッサの2D DMA機能によって、プロセッサは、ラインの最後、1フレームのビデオが転送された後、またはDMAエラーの発生時に割込みを受けることができます。実際に、DMAx_XCOUNT MMRとDMAx_YCOUNT MMRの指定によって、フレキシブルなデータ割込みポイントが可能になります。たとえば、DMAレジスタでXMODIFY = YMODIFY = 1と想定します。データ・フレームが320×240バイト（それぞれ320バイトの240行）を含んでいる場合には、以下の条件が保持されます。

- XCOUNT = 320、YCOUNT = 240、DI_SEL = 1（DMAx_CONFIG内のDI_SELビット）に設定すると、フレーム全体に対して転送される行ごとに割込みが生じます。
- XCOUNT = 320、YCOUNT = 240、DI_SEL = 0 に設定すると、フレームの完了時（320バイトの240行が転送されたとき）にのみ割込みが生じます。
- XCOUNT = 38,400（320×120）、YCOUNT = 2、DI_SEL = 1に設定すると、フレームの半分が転送されたときと、フレーム全体が転送されたときに割込みが生じます。

次に、PPIによるDMA動作の一般的な設定手順を示します。DMAの設定の詳細については、9-1ページの「ダイレクト・メモリ・アクセス」を参照してください。

1. DMA レジスタを、希望する DMA 動作モードに合わせて適宜設定します。
2. DMA チャンネルの動作をイネーブルにします。
3. 適切な PPI レジスタを設定します。
4. `PPI_CONTROL` のビット 0 に `1` を書き込んで、PPI をイネーブルにします。

データ転送シナリオ

図 11-16は、PPIを使用してビデオで転送するための2つの方法を示します。これらの図はきわめて一般化されており、的確なPPIモードと設定を考慮した後でのみ（たとえば、フィールド1のみの転送、奇数および偶数エレメントの転送）、帯域幅の計算を行う必要があります。

図の上部には、一例として、JPEG圧縮の場合に適切な状況を示します。ビデオの最初のN行は、PPIを介してL1メモリにDMAされます。L1内では、圧縮アルゴリズムがデータに作用して、圧縮結果はプロセッサからSPORTを介して送信されます。なお、この方法では、SDRAMアクセスは必要ありません。

図の下部では、MPEG-2やMPEG-4など、いっそう複雑な圧縮アルゴリズムを検討します。ここでは、生のビデオがSDRAMに直接転送されます。それとは無関係に、メモリDMAチャンネルは、中間処理ステージとしてSDRAMとL1メモリとの間でデータ・ブロックを転送します。最後に、圧縮されたビデオがSPORTを介してプロセッサから出力されます。

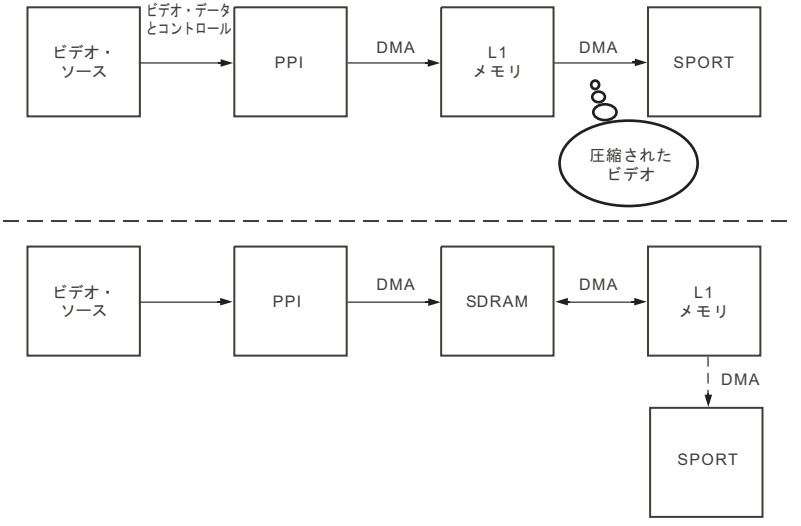


図 11-16. PPI のデータ転送シナリオ

第12章　シリアル・ポート・コントローラ

このプロセッサには、2つの同じ同期シリアル・ポート、つまりSPORTがあります。SPORTは、さまざまなシリアル・データ通信プロトコルをサポートし、マルチプロセッサ・システムにおいてプロセッサ間を直接に相互接続できます。

シリアル・ポート（SPORT0とSPORT1）は、多種多様なペリフェラル・シリアル・デバイスにI/Oインターフェースを提供します。SPORTは同期シリアル・データ転送のみを提供します。プロセッサは、UARTを介して非同期のRS-232データ転送を提供します。各SPORTには送信用のピン・グループ（一次データ、二次データ、クロック、フレーム同期）と、受信用のピン・グループがあります。受信機能と送信機能は別個にプログラムされます。各SPORTは全二重デバイスであり、双方向に同時にデータ転送が可能です。SPORTは、メモリマップド・レジスタへの書込みによって、ビットレート、フレーム同期、ワード当たりのビット数をプログラムできます。

ここでは、レジスタとピンに対する命名規則として小文字xで数字を表します。この章では、たとえば、RFSxピンという名前はRFS0とRFS1を示します（それぞれ、SPORT0とSPORT1に対応）。この章では、LSBは最下位ビットを表し、MSBは最上位ビットを表します。

2つのSPORTには同じ機能があり、同じようにプログラムされます。各SPORTには、専用のコントロール・レジスタとデータ・バッファがあります。

SPORTは、フレーム同期パルスを使用して各ワードやパケットの先頭を示します。また、ビット・クロックは、各データビットの先頭をマークします。外部のビット・クロックとフレーム同期は、TXバッファとRXバッファに使用できます。

SPORTは、クロックとフレームの一連の同期オプションによってH.100をはじめとするさまざまなシリアル通信プロトコルを使用でき、業界標準の多数のデータ・コンバータやコーデックに対してグルーレスなハードウェア・インターフェースを提供します。

SPORTは、外部生成されたクロックではSCLK/2までのクロック・レートで動作でき、内部生成されたシリアル・ポート・クロックではシステム・クロック・レートの1/2で動作できます。SPORTの外部クロックは、常にSCLK周波数よりも低い必要があります。独立した送／受信クロックによって、きわめて柔軟なシリアル通信が実現します。

SPORTのクロックとフレーム同期は、システムによって内部的に生成したり、外部ソースから受信したりできます。SPORTは、LSBファーストまたはMSBファーストの伝送フォーマットで動作でき、3～32ビットのワード長を選択できます。選択可能な送信モードに加えて、オプションでは、ハードウェアでのμ則／A則圧伸機能も提供されます。SPORTのデータはDMAブロック転送を使用して、オンチップ・メモリとオフチップ・メモリとの間を自動的に転送できます。さらに、各SPORTはTDM（時分割多重）マルチチャンネル・モードを提供します。

各SPORTは、以下の機能を提供します。

- 独立した送／受信機能を提供します。
- 長さ3～32ビットのシリアル・データ・ワードを、MSBファーストまたはLSBファーストで転送します。
- I^2Sシリアル・デバイスにインターフェースするための代替フレーミングと制御に加えて、その他のオーディオ・フォーマット（たとえば、左寄せステレオ・シリアル・データ）も提供します。

- FIFOとダブル・バッファ・データ（送／受信機能にはデータ・バッファ・レジスタとシフト・レジスタがあります）によって、SPORTの要求に応える時間が与えられます。
- 各SPORTに2本の同期送信データ・ピン、2本の同期受信データ・ピン、バッファを用意することで、利用できる合計データストリームを倍増させます。
- 送／受信されたワードに対して、μ則／A則のハードウェア圧伸を実行します（詳細については、12-37ページの「圧伸」を参照）。
- シリアル・クロック信号とフレーム同期信号を広範囲の周波数で内部的に生成したり、外部ソースからクロックとフレーム同期入力を受け付けたりします。
- データ・ワードごとのフレーム同期信号の有無、フレーム信号の内部生成または外部生成、フレーム信号のアクティブ・ハイまたはアクティブ・ロー、設定可能な2つのパルス幅とフレーム信号タイミングを選択できます。
- プロセッサ制御のもとで、オンチップ・メモリとの間で割込み駆動型のシングル・ワード転送を行います。
- DMAマスター制御のもとで、メモリとの間でダイレクト・メモリ・アクセス転送を行います。DMAは、自動バッファ・ベース（同一転送範囲の繰返し）またはディスクリプタ・ベース（さまざまなDMAパラメータによる個別の転送または転送範囲の繰返し）とすることができます。
- オンチップ・メモリとの間でDMA転送を行います。各SPORTでは、データのブロック全体を自動的に送／受信できます。
- 複数のデータ・ブロックにわたってDMA動作のチェイニングが可能です。

- TDMインターフェース用のマルチチャンネル・モードがあります。各SPORTでは、1024までの合計チャンネルで構成されるストリームをもとに、128の連続チャンネル上に時分割多重されたシリアル・ビットストリームからデータを選択的に送／受信できます。このモードは、複数プロセッサ用のネットワーク通信方式として役立ちます。プロセッサが使用できる128チャンネルは、0から895＝（1023－128）までの任意のチャンネル位置から始めるように選択できます。なお、マルチチャンネル・セレクト・レジスタとWSIZEレジスタでは、アクティブ領域内でアクセスできる128チャンネルのサブセットを制御します。

表 12-1には、各SPORTのピンを示します。

表 12-1. シリアル・ポート（SPORT）ピン

ピン[1]	説明
DTxPRI	送信データ・プライマリ
DTxSEC	送信データ・セカンダリ
TSCLKx	送信クロック
TFSx	送信フレーム同期
DRxPRI	受信データ・プライマリ
DRxSEC	受信データ・セカンダリ
RSCLKx	受信クロック
RFSx	受信フレーム同期

1　ピン名の中の小文字 x は、0 または 1（SPORT0 または SPORT1 に対応）の値を表します。

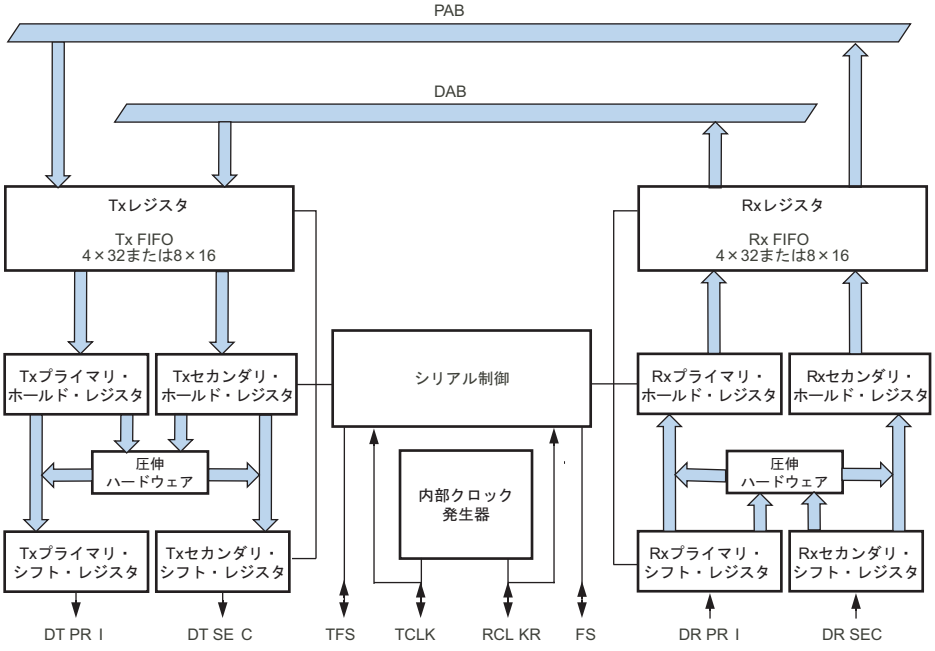
SPORTは、そのDRxPRI入力とDRxSEC入力でシリアル・データを受信し、そのDTxPRI出力とDTxSEC出力でシリアル・データを送信します。また、同時に送／受信して全二重動作を実現できます。送信の場合、データビット（DTxPRIとDTxSEC）は送信クロック（TSCLKx）に同期します。受信の場合、データビット（DRxPRIとDRxSEC）は受信クロック（RSCLKx）に同期します。シリアル・クロックは、プロセッサが生成した場合には出力で

あり、クロックが外部生成された場合には入力です。フレーム同期信号RFSxとTFSxは、シリアル・データ・ワードの先頭またはシリアル・ワードのストリームの先頭を示すために使用されます。

プライマリとセカンダリのデータ・ピンは、シリアル・ポートのデータ・スループットを増やすための方法を提供します。これらのピンは、完全に独立したSPORTとしては動作せずに、別個のデータに対して（クロックとフレーム同期を共有して）同期したスタイルで動作します。プライマリ・ピンとセカンダリ・ピンで受信されたデータはメイン・メモリでインターリーブされ、データ・アドレス・ジェネレータ（DAG）ユニットでストライドを設定して取り出すことができます。DAGの詳細については、第5章「データ・アドレス・ジェネレータ」を参照してください。同様に、TXの場合には、データを交互にTXレジスタに書き込んでください。つまり、プライマリ、セカンダリ、プライマリ、セカンダリといった順です。これは、プロセッサの強力なDAGによって容易に実現されます。

シリアル・クロック信号に加えて、データは、フレーム同期信号によって通知される必要があります。フレーミング信号は、個々のワードの先頭またはワードのブロックの先頭で発生できます。

次の図は、1つのSPORTの簡略ブロック図です。送信されるデータはペリフェラル・バスを介して、内部のプロセッサ・レジスタからSPORTのSPORTx_TXレジスタに書き込まれます。このデータはオプションでハードウェアによって圧縮され、TXシフト・レジスタに自動転送されます。シフト・レジスタ内のビットはTSCLKxピンでのシリアル・クロックに同期して、MSBファーストまたはLSBファーストで、SPORTのDRxPRI/DRxSECピンからシフト・アウトされます。SPORTの受信部分はRSCLKxピンでのシリアル・クロックに同期して、DTxPRI/DTxSECピンからデータを受け付けます。ワード全体が受信されると、データはオプションで拡張され、SPORTのSPORTx_RXレジスタに自動転送されてからRX FIFOに送られて、プロセッサから使用可能になります。



注1：すべての幅広の矢印データ・パスは、SLENに応じて16/32ビット幅です。SLEN＝2～15では、8段FIFO付きの16ビット・データ・パスが使用されます。SLEN＝16～31では、4段FIFO付きの32ビット・データ・パスが使用されます。
注2：TxレジスタはTx FIFOの下部であり、RxレジスタはRx FIFOの上部です。

図 12-1. SPORT のブロック図

図 12-2は、SPORTの可能なポート接続を示します。なお、シリアル・デバイスAとBは共通のフレーム同期とクロックを共有するため、同期している必要があります。シリアル・デバイスCとDについても同様です。

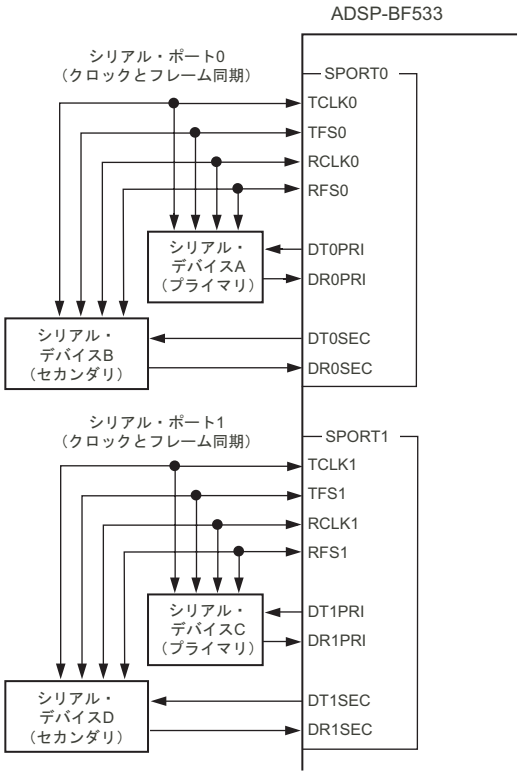


図 12-2. SPORT の接続

図 12-3は、3つの送信チャンネルと2つの受信チャンネルがプロセッサに接続されたステレオ・シリアル・デバイスの例を示します。

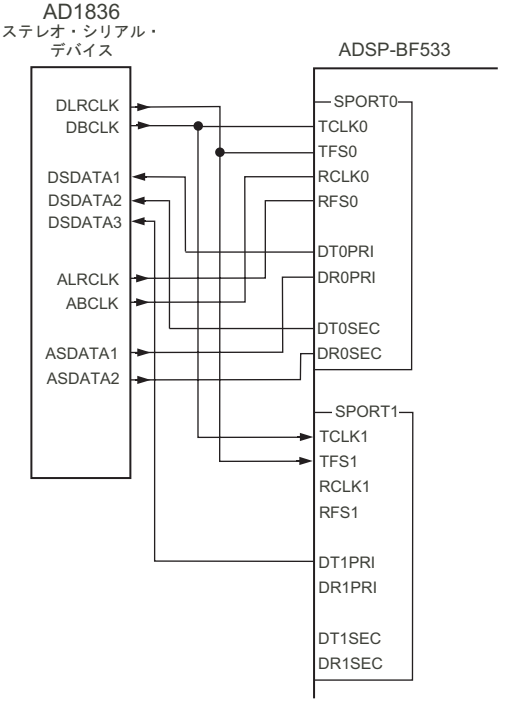


図 12-3. ステレオのシリアル接続

SPORT の動作

ここでは、SPORTの最も一般的な用途の例を挙げて、SPORTの一般的な動作を説明します。SPORT機能は設定可能であるため、この説明は可能な設定の一例にすぎません。

SPORTで伝送準備をするには、SPORTのSPORTx_TXレジスタに書き込みます。TFS信号は、シリアル・データの伝送を開始させます。伝送が始まったら、SPORTx_TXレジスタに書き込まれる各値は、FIFOを通じて内部の送信シフト・レジスタに転送されます。その後、ビットはSPORTx_TCR1レジスタの指定に基づいて、MSBファーストまたはLSBファーストで送信されます。各ビットは、TSCLKxの駆動エッジでシフト・アウトされます。TSCLKxの駆動エッジは、立上がりまたは立下がりに設定できます。SPORTはTX FIFOに空きがある限り、送信割込みを生成したり、DMA転送を要求したりします。

SPORTに受信されたビットは、内部の受信レジスタに蓄積されます。完全なワードが受信されると、そのワードはSPORT FIFOレジスタに書き込まれ、そのSPORT用の受信割込みが生成されるか、DMA転送が開始されます。DMAブロック転送が実行される場合には、割込みの生成が異なります。DMAの詳細については、第9章「ダイレクト・メモリ・アクセス」を参照してください。

SPORTのディスエーブル

SPORTは、プロセッサ・ハードウェアまたはソフトウェア・リセットによって自動的にディスエーブルにされます。また、SPORTの送信または受信イネーブル・ビット（それぞれ、SPORTx_TCR1レジスタのTSPENとSPORTx_RCR1レジスタのRSPEN）をクリアしても、SPORTを直接ディスエーブルにできます。SPORTへの影響は、それぞれの方法によって異なります。

プロセッサ・リセットでは、SPORTx_TCR1、SPORTx_TCR2、SPORTx_RCR1、SPORTx_RCR2レジスタ（TSPENとRSPENのイネーブル・ビットを含む）、およびTDIVx、RDIVx、SPORTx_TFSDIVx、SPORTx_RFSDIVxのクロックおよびフレーム同期デバイザ・レジスタをクリアしてSPORTをディスエーブルにします。進行中の動作はすべてアボートされます。

TSPEN と RSPEN のイネーブル・ビットをクリアすると、SPORT がディスエーブルにされ、すべての進行中の動作がアボートされます。ステータス・ビットもクリアされます。設定ビットは影響を受けずに残り、変更や上書きのためにソフトウェアによって読出しできます。SPORT の出力クロックをディスエーブルにするには、ディスエーブルにする SPORT を設定します。

なお、TSPEN/RSPEN を介して SPORT をディスエーブルにすると、TFSx/RFSx ピンや TSCLKx/RSCLKx ピンで現在アクティブなパルスが短くなることもあります（これらの信号が内部的に生成されるように設定されている場合）。

SPORT は、SPORTx_TCR1 または SPORTx_RCR1 レジスタでイネーブルにされてから 3 シリアル・クロック・サイクル以内に、データの送／受信を開始する用意ができます。この時点からは、シリアル・クロック・サイクルのロスはありません。最初の内部フレーム同期は、SPORT の用意ができてから 1 フレーム同期の遅延後に行われます。外部フレーム同期は、SPORT の用意ができるとすぐに行われます。

SPORT のマルチチャンネル動作をディスエーブルにするときには、まず TXEN をディスエーブルにしてから、RXEN をディスエーブルにします。なお、再びイネーブルにする前に、TXEN と RXEN の両方をディスエーブルにする必要があります。TX または RX だけをディスエーブルにすることは許されません。

SPORT モードの設定

SPORT 設定を行うには、設定レジスタでビット値とフィールド値の設定を行います。各 SPORT は、設定してからイネーブルにする必要があります。SPORT がイネーブルにされると、SPORT 設定レジスタへのそれ以上の書込みは禁止されます（ただし、SPORTx_RCLKDIV、SPORTx_TCLKDIV、マルチチャンネル・モード・チャンネル・セレクト・レジスタを除く）。他

のすべてのSPORT設定レジスタで値を変更するには、SPORTx_TCR1のTSPENまたはSPORTx_RCR1のRSPEN、あるいはその両方をクリアして、SPORTをディスエーブルにします。

各SPORTには、専用のコントロール・レジスタとデータ・バッファがあります。これらのレジスタについては、以下の節で詳しく説明します。特に指定のない限り、SPORTレジスタ内のすべてのコントロール・ビットとステータス・ビットはアクティブ・ハイです。

レジスタの書込みと実効遅延

SPORTがディスエーブルにされる（TSPENとRSPENがクリアされる）と、SPORTレジスタの書込みは、書込みが行われたSCLKサイクルの最後に内部的に完了して、レジスタは新しく書き込まれた値を次のサイクルで読み返します。

SPORTの送信（TSPENセット）または受信（RSPENセット）がイネーブルにされると、SPORTの対応する設定レジスタ書込みがディスエーブルにされます（ただし、SPORTx_RCLKDIV、SPORTx_TCLKDIV、マルチチャンネル・モード・チャンネル・セレクト・レジスタを除く）。SPORTx_TXレジスタの書込みは常にイネーブルです。SPORTx_RX、SPORTx_CHNL、SPORTx_STATは読出し専用レジスタです。

SPORTがディスエーブルにされている間にSPORTレジスタに書き込んだ場合には、コントロール・ビットやモード・ビットへの変更は一般に、SPORTが再びイネーブルにされたときに有効になります。

SPORTがディスエーブルにされている間（TSPEN/RSPEN = 0）、多くの設定レジスタでは変更だけが可能です。変更は、SPORTが再びイネーブルにされた後で有効になります。このルールの唯一の例外は、TCLKDIV/RCLKDIVレジスタとマルチチャンネル・セレクト・レジスタです。

SPORTx_TCR1 レジスタと SPORTx_TCR2 レジスタ

各SPORTの送信部分に対する主なコントロール・レジスタは、送信設定レジスタ、SPORTx_TCR1、SPORTx_TCR2です。

送信設定1レジスタのビット0（TSPEN）が1に設定された場合、SPORTの送信がイネーブルにされます。ハード・リセットやソフト・リセット時にはこのビットがクリアされ、すべてのSPORT伝送がディスエーブルにされます。

SPORTの送信がイネーブルにされた場合（TSPENセット）、対応するSPORT設定レジスタの書込みは許されません（ただし、SPORTx_TCLKDIVとマルチチャンネル・モード・チャンネル・セレクト・レジスタを除く）。禁止されたレジスタへの書込みは無効になります。SPORTがイネーブルにされている間、SPORTx_TCR1はビット0（TSPEN）を除いて書き込まれません。次に例を示します。

```
write (SPORTx_TCR1, 0x0001) ;    /* SPORT TXイネーブル */

write (SPORTx_TCR1, 0xFF01) ;    /* 無視、無効 */

write (SPORTx_TCR1, 0xFFF0) ;    /* SPORTディスエーブル、SPORTx_TCR1
はまだ0x0000と等しい */
```

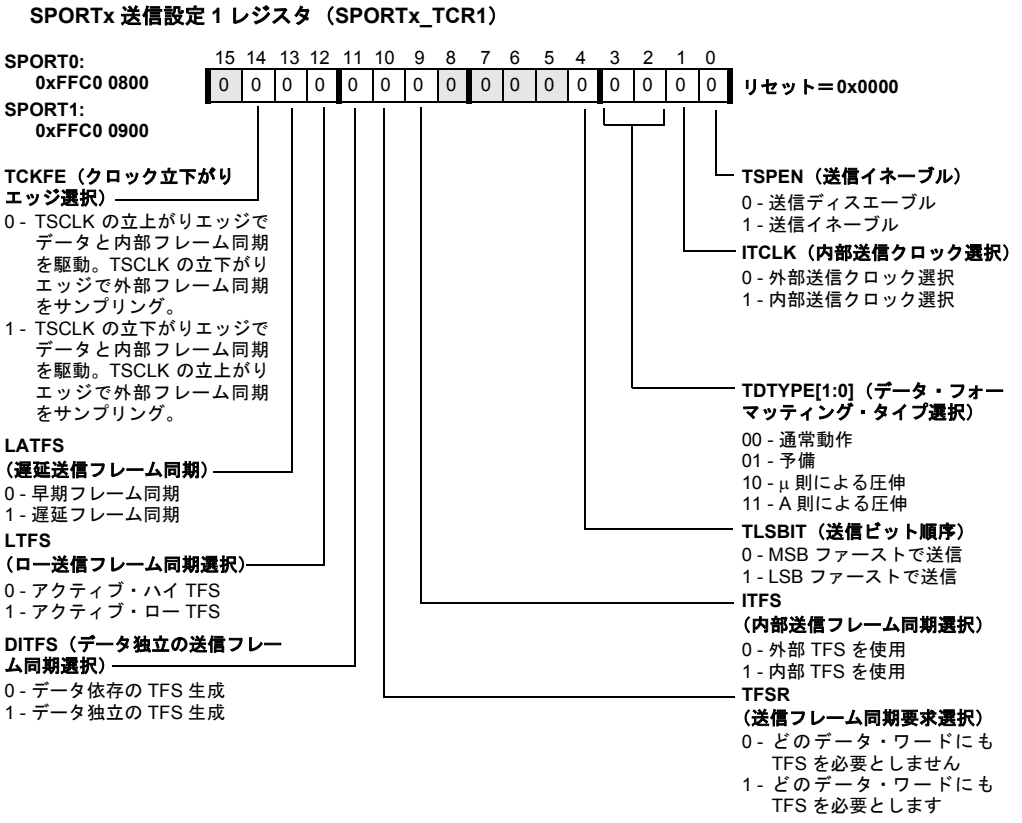


図 12-4. SPORTx 送信設定 1 レジスタ

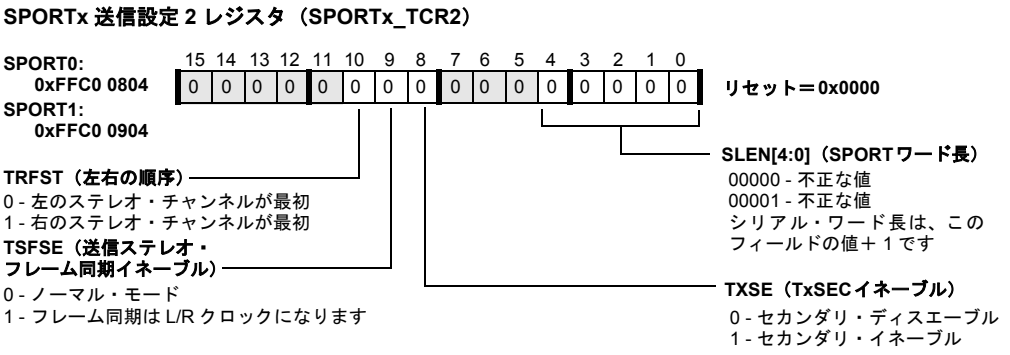


図 12-5. SPORTx 送信設定 2 レジスタ

送信設定レジスタSPORTx_TCR1とSPORTx_TCR2のビットの詳細を示します。

- **送信イネーブル**（TSPEN）：このビットは、SPORTの送信をイネーブルにする（セットされている場合）かディスエーブルにする（クリアされている場合）かを選択します。

  TSPENをセットすると、SPORT TX割込みがすぐにアサートされ、TXデータ・レジスタが空であり、フィルする必要があることを示します。この方法では、送信データ書込みコードをTX割込みサービス・ルーチン（ISR）に集中化できるため、一般的に望ましいといえます。このため、コードによりISRを初期化し、TX割込みの処理準備をしてからTSPENをセットしてください。

  同様に、DMA転送が使用される場合には、DMAコントロールを正しく設定してからTSPENをセットしてください。すべてのDMAコントロール・レジスタを設定してから、TSPENをセットしてください。
  TSPENをクリアすると、SPORTはデータ、TSCLK、フレーム同期ピンの駆動を停止します。さらに、SPORTの内部回路もシャットダ

ウンされます。低消費電力アプリケーションでは、SPORTが使用されなくなるたびにTSPENをクリアすることでバッテリ寿命を延ばすことができます。

すべての SPORT コントロール・レジスタをプログラムしてから、TSPENをセットしてください。代表的なSPORT初期化コードでは、該当する場合はDMAコントロールも含めて、最初にすべてのコントロール・レジスタを設定します。コードの最後のステップでは、SPORTx_TCR1にTSPENを含めて必要なすべてのビットを書き込みます。

- **内部送信クロック選択**（ITCLK）：このビットは、TSCLKピンでの内部送信クロック（セットされている場合）または外部送信クロック（クリアされている場合）を選択します。外部クロックが選択されると、TCLKDIVのMMR値は使用されません。
- **データ・フォーマッティング・タイプ選択**：2 つの TDTYPE ビットは、シングル動作とマルチチャンネル動作に使用されるデータ・フォーマットを指定します。
- **ビット順序選択**（TLSBIT）：TLSBIT ビットで、SPORT を通じて送信されるデータ・ワードのビット順序を選択します。
- **シリアル・ワード長選択**（SLEN）：シリアル・ワード長（SPORTを通じて送信される各ワードのビット数）は、SLENフィールドの値に1を加算して計算されます。

```
シリアル・ワード長＝SLEN＋1;
```

SLENフィールドには2～31の値を設定できます。このフィールドには、0と1は不正な値です。SLENフィールドに対する3つの一般的な設定は次のようになります。つまり、完全な16ビット・ワードを送信するには15、8ビット・バイトを送信するには7、24ビット・ワードを送信するには23です。プロセッサは、DMAまたはMMR

書込み命令によって、16/32 ビット値を送信バッファにロードできます。SLEN フィールドでは、これらのビットのうち、シリアル・リンクを通じてレジスタからシフト・アウトするビット数を SPORT に指示します。シリアル・ポートでは、送信バッファからビット [SLEN:0] を転送します。

フレーム同期信号は、SLEN によってではなく、SPORTx_TFSDIV レジスタと SPORTx_RFSDIV レジスタによって制御されます。送信されるバイトやワードごとにフレーム同期パルスを生成するには、フレーム同期デバイダ・レジスタに適切なフレーム同期デバイダをプログラムする必要があります。SLEN に 7 を設定しても、送信されるバイトごとにフレーム同期パルスは生成されません。

- **内部送信フレーム同期選択**（ITFS）：このビットは、SPORT が内部 TFS を使用する（セットされている場合）か、外部 TFS を使用する（クリアされている場合）かを選択します。
- **送信フレーム同期要求選択**（TFSR）：このビットは、SPORT がデータ・ワードごとに送信フレーム同期を必要とする（セットされている場合）か、必要としない（クリアされている場合）かを選択します。

SPORT の設定中には、TFSR ビットはセットされるのが普通です。フレーム同期パルスは、各ワードやデータ・パケットの先頭をマークするために使用されます。多くのシステムでは、適切に機能するためにフレーム同期を必要とします。

- **データ独立の送信フレーム同期選択**（DITFS）：このビットは、内部フレーム同期選択（ITFS = 1）の場合に、SPORT がデータ独立の TFS（選択した間隔での同期）を生成するか、データ依存の TFS（SPORTx_TX にデータが存在するときに同期）を生成するかを選択します。外部フレーム同期が選択された場合には、DITFS ビットは無視されます。

フレーム同期パルスは、データ・ワードの先頭をマークします。

DITFSがセットされた場合には、SPORTx_TXレジスタへのロードの有無とは無関係に、フレーム同期パルスは時間どおりに発行されます。DITFSがクリアされた場合には、フレーム同期パルスはSPORTx_TXデータ・レジスタがロードされた場合にのみ生成されます。レシーバが規則正しいフレーム同期パルスを必要とする場合には、DITFSをセットしてください。これにより、プロセッサはSPORTx_TXレジスタに時間どおりにロードし続けます。レシーバが不定期の遅延フレーム同期パルスを許容できる場合には、DITFSをクリアしてください。これにより、SPORTは古いデータの重複送信や破損データの送信（プロセッサがSPORTx_TXレジスタのロードに遅れた場合）を防ぐことができます。

- **ロー送信フレーム同期選択**（LTFS）：このビットは、アクティブ・ローのTFS（セットされている場合）またはアクティブ・ハイのTFS（クリアされている場合）を選択します。
- **遅延送信フレーム同期**（LATFS）：このビットは、遅延フレーム同期（セットされている場合）または早期フレーム同期（クリアされている場合）を設定します。
- **クロックの駆動／サンプル・エッジ選択**（TCKFE）：このビットは、データの駆動、内部生成されたフレーム同期の駆動、外部生成されたフレーム同期のサンプリングのために、SPORTで使用するTCLKx信号のエッジを選択します。セットされた場合、データと内部生成されたフレーム同期は立下がりエッジで駆動され、外部生成されたフレーム同期は立上がりエッジでサンプリングされます。クリアされた場合、データと内部生成されたフレーム同期は立上がりエッジで駆動され、外部生成されたフレーム同期は立下がりエッジでサンプリングされます。
- **TxSec イネーブル**（TXSE）：このビットは、シリアル・ポートのセカンダリ送信をイネーブルにします（セットされている場合）。

- **ステレオ・シリアル・イネーブル**（TSFSE）：このビットは、シリアル・ポートのステレオ・シリアル動作モードをイネーブルにします（セットされている場合）。デフォルトでは、このビットはクリアされ、通常のクロッキングとフレーム同期がイネーブルになります。
- **左右の順序**（TRFST）：このビットがセットされた場合、右のチャンネルがステレオ・シリアル動作モードで最初に送信されます。デフォルトでは、このビットはクリアされ、左のチャンネルが最初に送信されます。

SPORTx_RCR1 レジスタと SPORTx_RCR2 レジスタ

各SPORTの受信部分の主なコントロール・レジスタは、受信設定レジスタ、SPORTx_RCR1、SPORTx_RCR2です。

受信設定1レジスタのビット0（RSPEN）が1に設定された場合、SPORTは受信が可能になります。ハード・リセットまたはソフト・リセット時にはこのビットはクリアされ、すべてのSPORT受信がディスエーブルにされます。

SPORTの受信がイネーブルにされる（RSPENがセットされる）と、対応するSPORT設定レジスタの書込みは許可されません（ただし、SPORTx_RCLKDIVとマルチチャンネル・モード・チャンネル・セレクト・レジスタを除く）。禁止されたレジスタへの書込みは無効です。SPORTがイネーブルにされている間、SPORTx_RCR1はビット0（RSPEN）を除いて書き込まれません。次に例を示します。

```
write (SPORTx_RCR1, 0x0001) ;  /* PORT RXイネーブル */
write (SPORTx_RCR1, 0xFF01) ;  /* 無視、無効 */
write (SPORTx_RCR1, 0xFFF0) ;  /* SPORTディスエーブル、SPORTx_RCR1はまだ0x0000と等しい */
```

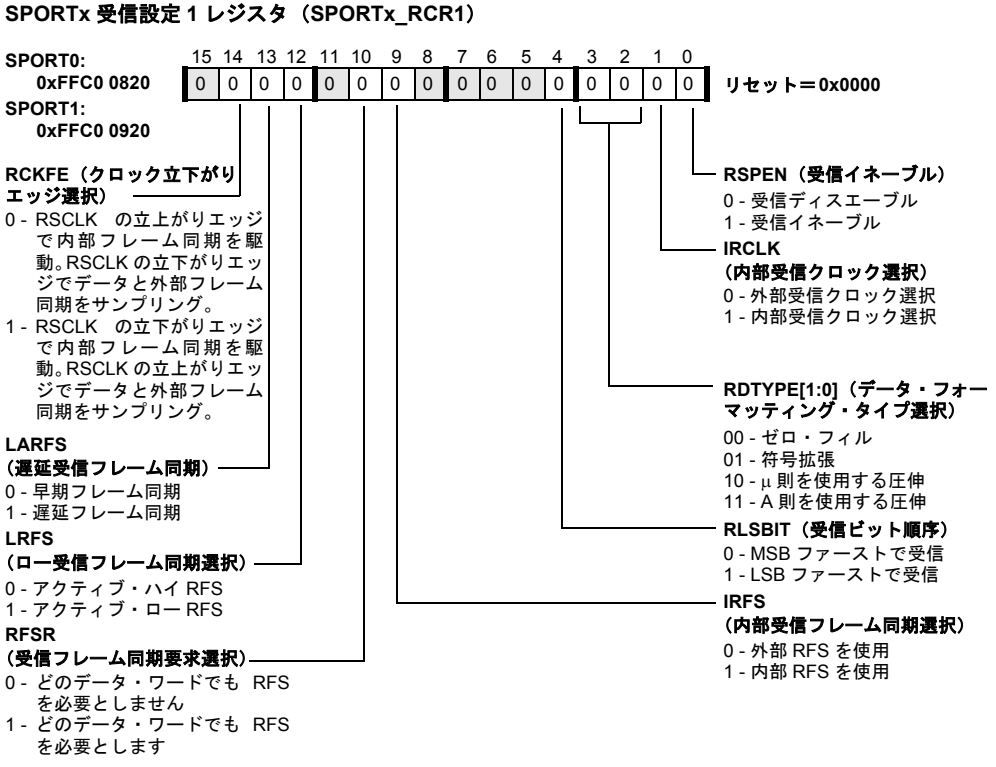


図 12-6. SPORTx の受信設定 1 レジスタ

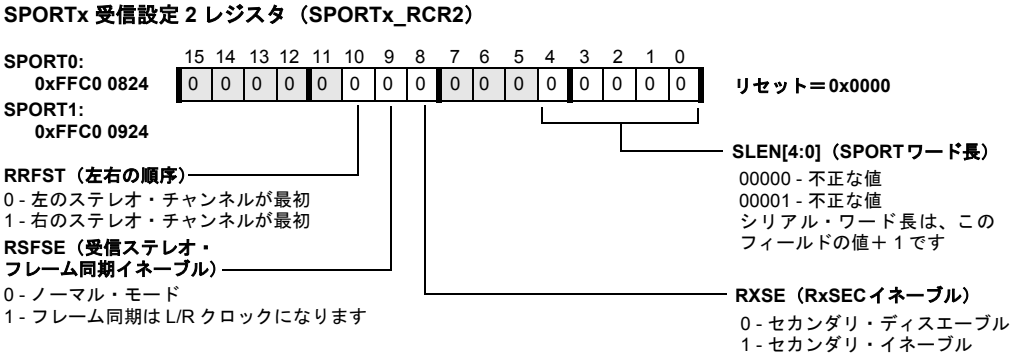


図 12-7. SPORTx 受信設定 2 レジスタ

受信設定レジスタ SPORTx_RCR1 と SPORTxRCR2 のビットの詳細を示します。

- **受信イネーブル**（RSPEN）：このビットは、SPORTの受信をイネーブルにする（セットされている場合）かディスエーブルにする（クリアされている場合）かを選択します。RSPEN ビットをセットするとSPORTがオンになり、データ受信ピンに加えて、受信ビット・クロックと受信フレーム同期ピンからも(そうプログラムされている場合）データをサンプリングします。

  RSPEN をセットすると、SPORTx レシーバがイネーブルにされてSPORTx RX割込みを生成できます。このため、コードではISRとDMAコントロール・レジスタを初期化し、RX割込みを処理する準備ができてからRSPENをセットしてください。RSPEN をセットすると、DMA要求も生成されます（DMAがイネーブルで、データが受信された場合）。すべてのDMAコントロール・レジスタを設定してから、RSPEN をセットしてください。RSPEN をクリアすると、SPORTはデータの受信を停止します。また、SPORTの内部受信

回路もシャットダウンされます。低消費電力アプリケーションでは、SPORTが使用されなくなるたびにRSPENをクリアすることで、バッテリ寿命を延ばすことができます。

すべての SPORT コントロール・レジスタをプログラムしてから、RSPENをセットしてください。代表的なSPORT初期化コードでは、該当する場合はDMAコントロールも含めて、最初にすべてのコントロール・レジスタを設定します。コードの最後のステップでは、SPORTx_RCR1にRSPENを含めて必要なすべてのビットを書き込みます。

- **内部受信クロック選択**（IRCLK）：このビットは、内部受信クロック（セットされている場合）または外部受信クロック（クリアされている場合）を選択します。外部クロックが選択されると、RCLKDIVのMMR値は使用されません。
- **データ・フォーマッティング・タイプ選択**（RDTYPE）：2つのRDTYPEビットは、シングル動作とマルチチャンネル動作に使用される4つのデータ・フォーマットの1つを指定します。
- **ビット順序選択**（RLSBIT）：RLSBIT ビットは、SPORT を通じて受信されるデータ・ワードのビット順序を選択します。
- **シリアル・ワード長選択**（SLEN）：シリアル・ワード長（SPORTを通じて受信される各ワードのビット数）は、SLENフィールドの値に1を加算して計算されます。SLENフィールドには2～31の値を設定できます。このフィールドには、0と1は不正な値です。

フレーム同期信号は、SLENによってではなく、SPORTx_TFSDIVレジスタとSPORTx_RFSDIVレジスタによって制御されます。送信されるバイトまたはワードごとにフレーム同期パルスを生成するには、フレーム同期デバイダ・レジスタに適切なフレーム同期デバイダをプログラムする必要があります。SLENに7を設定しても、送信されるバイトごとにフレーム同期パルスは生成されません。

- **内部受信フレーム同期選択**（IRFS）：このビットは SPORT が内部 RFS を使用する（セットされている場合）か、外部RFS を使用する（クリアされている場合）かを選択します。
- **受信フレーム同期要求選択**（RFSR）：このビットは SPORT がどのデータ・ワードにも受信フレーム同期を必要とする（セットされている場合）か、必要としない（クリアされている場合）かを選択します。
- **ロー受信フレーム同期選択**（LRFS）：このビットはアクティブ・ロー RFS（セットされている場合）またはアクティブ・ハイ RFS（クリアされている場合）を選択します。
- **遅延受信フレーム同期**（LARFS）：このビットは遅延フレーム同期（セットされている場合）または早期フレーム同期（クリアされている場合）を設定します。
- **クロックの駆動／サンプル・エッジ選択**（RCKFE）：このビットはデータのサンプリング、外部生成されたフレーム同期のサンプリング、内部生成されたフレーム同期の駆動のために、SPORTで使用するRSCLKクロック信号のエッジを選択します。セットされた場合、内部生成されたフレーム同期は立下がりエッジで駆動され、データと外部生成されたフレーム同期は立上がりエッジでサンプリングされます。クリアされた場合には、内部生成されたフレーム同期は立上がりエッジで駆動され、データと外部生成されたフレーム同期は立下がりエッジでサンプリングされます。
- **RxSec イネーブル**（RXSE）：このビットはシリアル・ポートのセカンダリ受信をイネーブルにします（セットされている場合）。
- **ステレオ・シリアル・イネーブル**（RSFSE）：このビットは、シリアル・ポートのステレオ・シリアル動作モードをイネーブルにします（セットされている場合）。デフォルトでは、このビットはクリアされ、通常のクロッキングとフレーム同期がイネーブルにされます。

- **左右の順序**（RRFST）：このビットがセットされた場合には、ステレオ・シリアル動作モードで右のチャンネルが最初に受信されます。デフォルトでは、このビットはクリアされ、左のチャンネルが最初に受信されます。

データ・ワード・フォーマット

SPORTを通じて転送されるデータ・ワードのフォーマットは、送信SLENと受信SLEN、RDTYPE、TDTYPE、RLSBIT、およびSPORTx_TCR1、SPORTx_TCR2、SPORTx_RCR1、SPORTx_RCR2レジスタのTLSBITビットの組合わせによって設定されます。

SPORTx_TX レジスタ

SPORTx送信データ・レジスタ（SPORTx_TX）は、書込み専用のレジスタです。読出しを行うと、ペリフェラル・アクセス・バス（PAB）エラーが発生します。このレジスタへの書込みによって、トランスミッタFIFOへの書込みが行われます。16ビット幅のFIFOは、ワード長＜＝16では8段であり、ワード長＞16では4段です。このFIFOは、プライマリ・データとセカンダリ・データの両方に共通であり、両方のデータを格納します。図 12-8に、FIFOでのデータ順序を示します。

図に示すように、FIFO内ではプライマリ・データとセカンダリ・データのインターリービングを保持することが重要です。つまり、FIFOへのPAB/DMA書込みでは、最初はプライマリ、次にセカンダリという順序に従う必要があります（セカンダリがイネーブルの場合）。DAB/PAB書込みでは、そのサイズはデータ・ワード長に一致する必要があります。16ビット以下のワード長では、16ビット書込みを使用します。16ビットを超えるワード長では、32ビット書込みを使用します。

図 12-8. SPORT 送信 FIFO のデータ順序

送信がイネーブルにされると、FIFOからのデータは、TXSE と SLEN に基づいてTXホールド・レジスタで組み立てられてから、プライマリとセカンダリのシフト・レジスタにシフト・インされます。ここからは、データは、DTPRI ピンと DTSEC ピンで連続的にシフト・アウトされます。

TSPEN = 1 でTX FIFOに新たなワード用の空きがあるとき、SPORT TX割込みがアサートされます。SPORT DMAがイネーブルである場合には、この割込みは発生しません。DMA動作については、第9章「ダイレクト・メモリ・アクセス」を参照してください。

送信フレーム同期が行われ、シリアル・シフト・レジスタに新しいデータがロードされない場合には、SPORTステータス・レジスタで送信アンダーフロー・ステータス・ビット（TUVF）がセットされます。マルチチャンネル・モード（MCM）では、シリアル・シフト・レジスタがロードされないときはいつでもTUVFがセットされ、現在のイネーブルにされたチャンネルで伝送が開始されます。TUVF ステータス・ビットはスティッキーなWrite-1-to-Clear（W1C）ビットであり、シリアル・ポートをディスエーブルにしても（TXEN = 0 を書き込む）クリアされます。

ソフトウェアによってコア・プロセッサがSPORTx_TX書込みによるフルTX FIFOへの書込みを試みた場合、新しいデータは失われ、FIFO内のデータには上書きは行われません。TOVFステータス・ビットがセットされ、SPORTエラー割込みがアサートされます。TOVFビットはスティッキー・ビットであり、SPORTx_TXのディスエーブルによってのみクリアされます。コア・プロセッサがこの種のエラーを起こさずにSPORTx_TXレジスタにアクセスできるかどうかを判断するには、まずレジスタのステータスを読み出します。FIFO内に他のワード用の空きがある場合には、SPORTステータス・レジスタのTXFビットは0です。

たとえSPORTがディスエーブルにされていても、SPORTxステータス・レジスタのTXFとTOVFのステータス・ビットは、コア・プロセッサからの書込みと同時に更新されます。

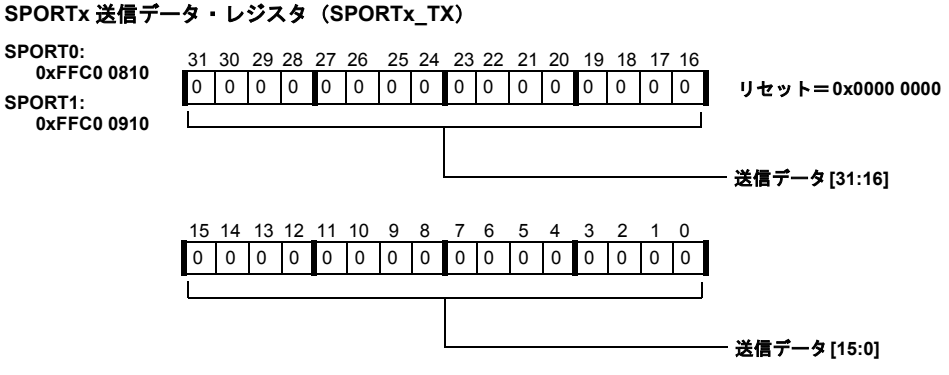


図 12-9. SPORTx 送信データ・レジスタ

SPORTx_RX レジスタ

SPORTx受信データ・レジスタ（SPORTx_RX）は、読出し専用レジスタです。書込みを行うとPABエラーが発生します。プライマリ・データとセカンダリ・データ用に同じ場所が読み出されます。このレジスタ空間からの読出しによって、受信FIFOが読み出されます。この16ビットFIFOは、受信ワード長＜＝16では8段、長さ＞16ビットでは4段です。このFIFOは、プライマリとセカンダリの受信データによって共有されます。データの格納方法は設定ビットSLENとRXSEの設定値に依存するため、PAB/DMA読出しを使用する読出しの順序が重要となります。

図 12-10には、FIFO内のデータ記憶とデータ順序を示します。

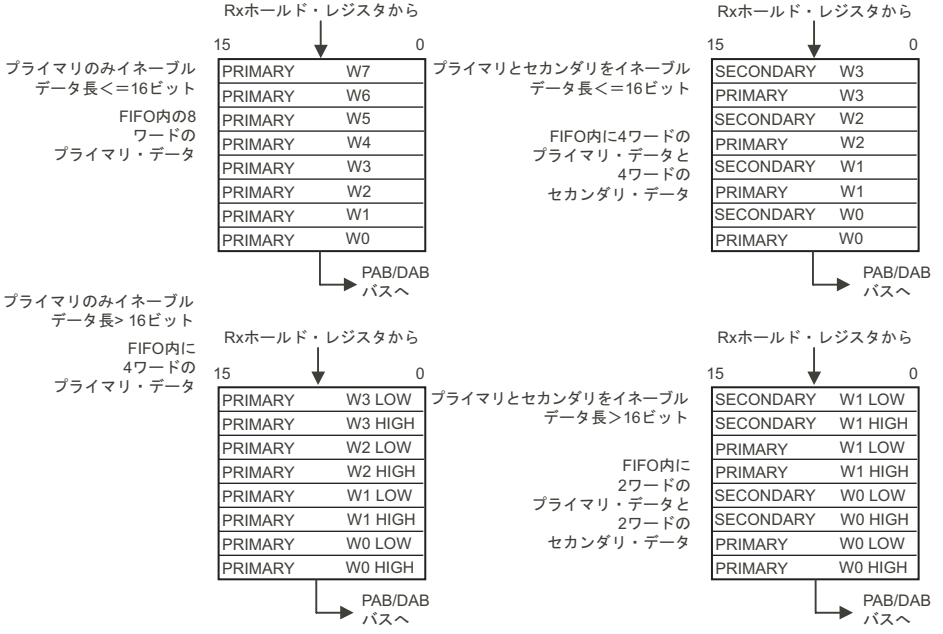


図 12-10.SPORT 受信 FIFO のデータ順序

プライマリ・データとセカンダリ・データ用のFIFOから読み出すとき、最初にプライマリを読み出し、続いてセカンダリを読み出します。DAB/PAB読出しのサイズは、データ・ワード長に一致する必要があります。16ビットまで（16ビットを含む）のワード長では、16ビット読出しを使用します。16ビットを超えるワード長では、32ビット読出しを使用します。

受信がイネーブルにされると、DRPRIピンからのデータはRXプライマリ・シフト・レジスタにロードされ、DRSECピンからのデータはRXセカンダリ・シフト・レジスタにロードされます。ワードの転送が完了すると、データはそれぞれプライマリ・データ用とセカンダリ・データ用のRXホールド・レジスタにシフト・インされます。ホールド・レジスタからのデータは、RXSEとSLENに基づいてFIFOに転送されます。

RSPEN = 1でRX FIFOがワードを受信すると、SPORT RX割込みが生成されます。コア・プロセッサがFIFO内のすべてのワードを読み出すと、RX割込みがクリアされます。SPORT RX割込みが設定されるのは、SPORT RX DMAがディスエーブルにされている場合だけです。そうでない場合、FIFOはDMA読出しによって読み出されます。

プログラムによって、コア・プロセッサが空のRX FIFOから読出しを試みた場合には、古いデータが読み出され、SPORTx_STATレジスタのRUVFフラグがセットされ、SPORTエラー割込みがアサートされます。RUVFビットはスティッキー・ビットであり、SPORTがディスエーブルにされたときにだけクリアされます。コアがこのエラーを起こさずにRXレジスタにアクセスできるかどうかを判断するには、まずRX FIFOステータス（SPORTxステータス・レジスタのRXNE）を読み出します。たとえSPORTがディスエーブルにされていても、RUVFステータス・ビットは更新されます。

RXシフト・レジスタで新しいワードが組み立てられ、RXホールド・レジスタがデータをFIFOに移動しなかった場合、SPORTx_STATレジスタのROVFステータス・ビットがセットされます。ホールド・レジスタ内の以前に書き込まれたワードは上書きされます。ROVFビットはスティッキー・ビットであり、SPORT RXをディスエーブルにしてのみクリアされます。

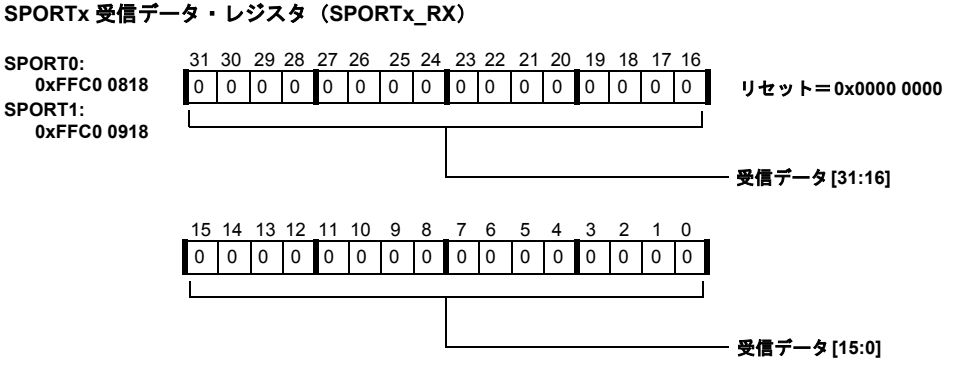


図 12-11. SPORTx 受信データ・レジスタ

SPORTx_STAT レジスタ

SPORTステータス・レジスタ（SPORTx_STAT）を使用すれば、そのステータスが満杯であるか空であるかを調べて、SPORT RXまたはTX FIFOへのアクセスが可能かどうかを判断できます。

SPORTステータス・レジスタのTXFビットは、TX FIFO内に空領域があるかどうかを示します。RXNEステータス・ビットは、RX FIFO内にワードがあるかどうかを示します。TXHREビットは、TXホールド・レジスタが空であるかどうかを示します。

TXシフト・レジスタが空である場合、送信アンダーフロー・ステータス・ビット（TUVF）は、外部ソースまたは内部ソースからTFS信号が発生するたびにセットされます。SPORT設定レジスタのDITFS制御ビットをクリアすれば、SPORTx_TXが空になってもTFSの内部生成を抑制できます。TUVF

ステータス・ビットは、スティッキーなWrite-1-to-Clear（W1C）ビットであり、シリアル・ポートをディスエーブルにしても（TXEN = 0を書き込む）クリアされます。

連続送信（TFSR = 0）では、TXホールド・レジスタ内に新しいワードがない場合には、送信されたワードの最後でTUVFがセットされます。

満杯のTX FIFOにワードが書き込まれると、TOVFビットがセットされます。これはスティッキーなW1Cビットであり、TXEN = 0を書き込んでもクリアされます。たとえSPORTがディスエーブルにされても、TXFとTOVFの両方が更新されます。

SPORT RXホールド・レジスタが満杯で、シフト・レジスタに新しい受信ワードが受信されると、SPORTステータス・レジスタの受信オーバーフロー・ステータス・ビット（ROVF）がセットされます。これはスティッキーなW1Cビットであり、シリアル・ポートをディスエーブルにしても（RXEN = 0を書き込む）クリアされます。

空のRX FIFOから読出しを試みると、RUVFビットがセットされます。これはスティッキーなW1Cビットであり、RXEN = 0の書込みによってもクリアされます。たとえSPORTはディスエーブルにされても、RUVFビットは更新されます。

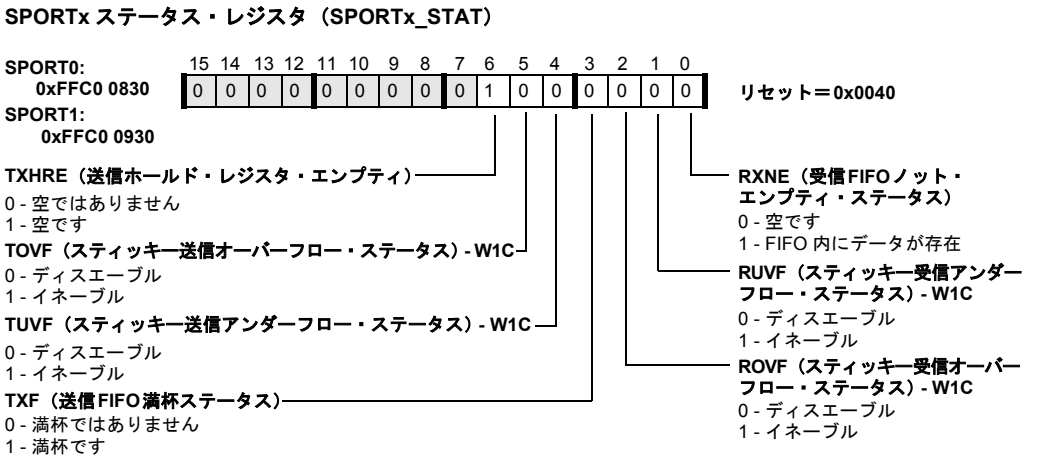


図 12-12.SPORTx のステータス・レジスタ

■ SPORT RX、TX、およびエラー割込み

RSPEN がイネーブルで、RX FIFO 内に何かのワードが存在するとき、SPORT RX割込みがアサートされます。RX DMAがイネーブルである場合には、SPORT RX割込みはオフにされ、DMAはRX FIFOを処理します。

TSPEN がイネーブルで、TX FIFOにワード用の空きがあるとき、SPORT TX割込みがアサートされます。TX DMAがイネーブルである場合には、SPORT TX割込みはオフにされ、DMAはTX FIFOを処理します。

いずれかのスティッキーなステータス・ビット（ROVF、RUVF、TOVF、TUVF）がセットされると、SPORTエラー割込みがアサートされます。ROVF ビットと RUVF ビットをクリアするには、RSPEN に0を書き込みます。TOVF ビットと TUVF ビットをクリアするには、TSPEN に0を書き込みます。

■ PAB エラー

SPORTは、不正なレジスタ読出し動作や書込み動作に対してPABエラーを生成します。次に例を示します。

- 書込み専用レジスタ（たとえば、SPORT_TX）の読出し
- 読出し専用レジスタ（たとえば、SPORT_RX）の書込み
- 誤ったサイズによるレジスタの書込み／読出し（たとえば、16ビット・レジスタの32ビット読出し）
- 予約されたレジスタ位置へのアクセス

SPORTx_TCLKDIV レジスタと SPORTx_RCLKDIV レジスタ

内部生成されるクロックの周波数は、システム・クロック周波数（SCLKピン上）と16ビットのシリアル・クロック分周率レジスタ（SPORTx送信シリアル・クロック・デバイダ・レジスタ、SPORTx_TCLKDIV、SPORTx受信シリアル・クロック・デバイダ・レジスタ、SPORTx_RCLKDIV）の値の関数です。

図 12-13.SPORTx 送信シリアル・クロック・デバイダ・レジスタ

図 12-14.SPORTx 受信シリアル・クロック・デバイダ・レジスタ

SPORTx_TFSDIV レジスタと SPORTx_RFSDIV レジスタ

16 ビットの SPORTx 送信フレーム同期デバイダ・レジスタ（SPORTx_TFSDIV）と SPORTx 受信フレーム同期デバイダ・レジスタ（SPORTx_RFSDIV）では、フレーム同期が内部生成される場合に、TFS または RFS パルスを生成するまでにカウントする送／受信クロック・サイクルの数を指定します。このように、フレーム同期を使用して周期的な転送を開始できます。シリアル・クロック・サイクルのカウントは、内部／外部生成されたシリアル・クロックに適用されます。

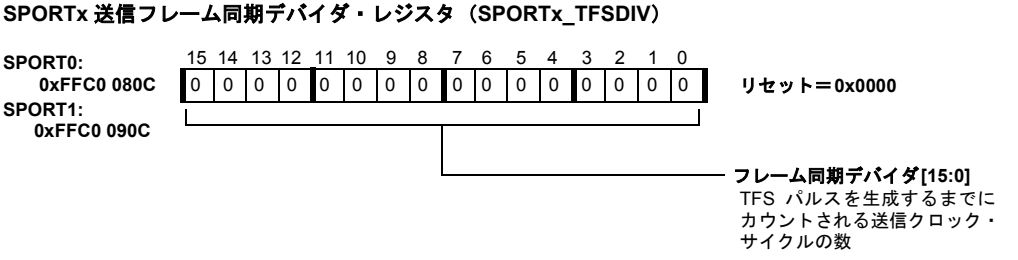


図 12-15.SPORTx 送信フレーム同期デバイダ・レジスタ

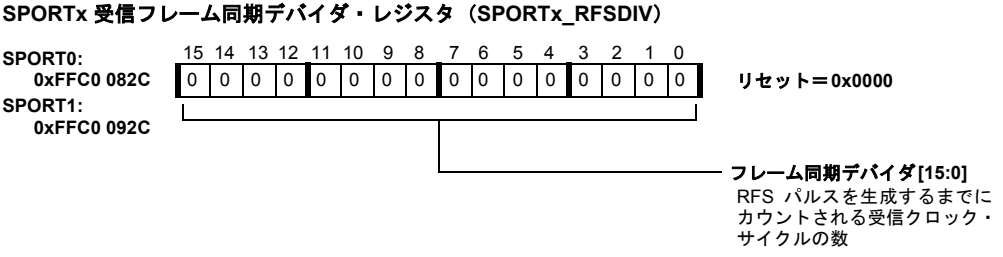


図 12-16.SPORTx 受信フレーム同期デバイダ・レジスタ

クロック周波数とフレーム同期周波数

内部ソースまたは外部ソースの最大シリアル・クロック周波数はSCLK/2です。内部生成されるクロックの周波数は、システム・クロック周波数（SCLK）と16ビットのシリアル・クロック分周率レジスタSPORTx_TCLKDIVとSPORTx_RCLKDIVの値の関数です。

SPORTx_TCLK周波数＝
（SCLK周波数）/（2×（SPORTx_TCLKDIV+1））

SPORTx_RCLK周波数＝
（SCLK周波数）/（2×（SPORTx_RCLKDIV+1））

内部シリアル・クロックがイネーブルである間に、SPORTx_TCLKDIVまたはSPORTx_RCLKDIVの値が変更された場合には、TSCLKまたはRSCLK周波数の変更は、TFSまたはRFSの次の前縁に続くTSCLKまたはRSCLKの駆動エッジの先頭で有効になります。

内部フレーム同期が選択されて（SPORTx_TCR1レジスタのITFS＝1またはSPORTx_RCR1レジスタのIRFS＝1）フレーム同期が要求されないとき、SPORTx_TCLKDIVまたはSPORTx_RCLKDIVの値が変更された場合には、最初のフレーム同期ではクロック・デバイダを更新しません。2番目のフレーム同期では更新が行われます。

SPORTx_TFSDIV レジスタと SPORTx_RFSDIV レジスタは、フレーム同期が内部生成される場合に、TFS または RFS パルスを生成するまでにカウントされる送／受信クロック・サイクルの数を指定します。これによって、フレーム同期は周期的な転送を開始できます。シリアル・クロック・サイクルのカウントは、内部生成／外部生成されたシリアル・クロックに適用されます。

フレーム同期パルス間でのサイクル数の式は、次のとおりです。

フレーム同期アサーション間での送信シリアル・クロックの数＝TFSDIV+1
フレーム同期アサーション間での受信シリアル・クロックの数＝RFSDIV+1

シリアル・クロック周波数と希望するフレーム同期周波数が与えられた場合、TFSDIV や RFSDIV の正しい値を決定するには、次の式を使用します。

SPORTxTFS 周波数＝（TSCLKx 周波数）/（SPORTx_TFSDIV+1）
SPORTxRFS 周波数＝（RSCLKx 周波数）/（SPORTx_RFSDIV+1）

したがって、フレーム同期は（TFSDIV = 0 の場合は送信用に、RFSDIV = 0 の場合は受信用に）連続してアクティブになります。しかし、TFSDIV（または RFSDIV）の値は、シリアル・ワード長－ 1（SPORTx_TCR2 または SPORTx_RCR2 の SLEN フィールドの値）を下回ってはいけません。さもなければ、外部デバイスが現在の動作をアボートしたり、その他の予測できない結果を招くことがあります。SPORT が使用されていない場合には、TFSDIV（または RFSDIV）デバイザを使用して、外部クロックを分周したり周期的なパルスや割込みを生成したりするためのカウンタとすることができます。SPORT が機能するためには、この動作モード用にイネーブルにされている必要があります。

■ 最大クロック・レートの制約

外部生成された遅延送信フレーム同期は、データ出力への到着が遅れることもあるため、最大シリアル・クロック速度が制限されることもあります。正確なタイミング仕様については、『ADSP-BF531/ADSP-BF532/ADSP-BF533 Embedded Processor Data Sheet』を参照してください。

▶ フレーム同期とクロックの例

次のコード例では、クロックとフレーム同期の設定を示します。

```
r0 = 0x00FF;
p0.l = SPORT0_RFSDIV & 0xFFFF;
p0.h = (SPORT0_RFSDIV >> 16) & 0xFFFF;
w[p0] = r0.l; ssync;
p0.l = SPORT0_TFSDIV & 0xFFFF;
w[p0] = r0.l; ssync;
```

ワード長

各SPORTチャンネル（送／受信）では、3～32ビットのワード長を独立して扱います。長さが32ビット未満でLSB位置に置かれている場合には、データはSPORTデータ・レジスタ内で右寄せされます。ワード長を決定するには、各SPORTのSPORTx_TCR2レジスタとSPORTx_RCR2レジスタのシリアル・ワード長（SLEN）フィールドの値を使用して、次の式で計算します。

シリアル・ワード長＝SLEN + 1

SLEN値には、0や1を設定しないでください。許容される値は2～31です。連続動作（現在のワードの最終ビットの直後に、次のワードの先頭ビットが続くとき）は、4以上のワード・サイズに制限されます（つまり、SLEN ≥ 3）。

ビット順序

シリアル・ワードがMSBファーストで送信されるかLSBファーストで送信されるかは、ビット順序によって決まります。ビット順序は、SPORTx_RCR1 レジスタと SPORTx_TCR1 レジスタの RLSBIT ビットと TLSBIT ビットによって選択されます。RLSBIT（または TLSBIT）＝ 0 の場合、シリアル・ワードはMSBファーストで受信（または送信）されます。RLSBIT（または TLSBIT）＝ 1 の場合、シリアル・ワードはLSBファーストで受信（または送信）されます。

データ型

SPORTx_TCR1 レジスタの TDTYPE フィールドと、SPORTx_RCR1 レジスタの RDTYPE フィールドでは、シングル動作とマルチチャンネル動作に対して4つのデータ・フォーマットの1つを指定します。表 12-2 を参照してください。

表 12-2. TDTYPE、RDTYPE、データ・フォーマッティング

TDTYPE または RDTYPE	SPORTx_TCR1 データ・フォーマッティング	SPORTx_RCR1 データ・フォーマッティング
00	通常動作	ゼロ・フィル
01	予備	符号拡張
10	μ 則を使用する圧伸	μ 則を使用する圧伸
11	A 則を使用する圧伸	A 則を使用する圧伸

これらのフォーマットは、SPORTx_RX バッファと SPORTx_TX バッファにロードされるシリアル・データ・ワードに適用されます。SPORTx_TX データ・ワードでは、有効ビットだけが送信されるため、実際にはゼロ詰めも符号拡張も行われません。

圧伸

圧伸（COMPAND：COMpressingとexPANDingの短縮形）とは、送信すべきビット数を最小限に抑えるために、データを対数的にエンコーディングおよびデコーディングするプロセスです。SPORTは、μ則とA則という、最も広く使用される2つの圧伸アルゴリズムを利用します。プロセッサはCCITT G.711仕様に基づいて、データの圧伸を行います。圧伸のタイプはSPORTごとに独立して選択できます。

圧伸がイネーブルにされた場合、SPORTx_RXレジスタ内の有効データは受信した8つのLSBを16ビットに符号拡張し、右寄せ、拡張した値になります。SPORTx_TXへの書込みによって、16ビット値が8つのLSBに圧縮され（送信ワードの幅に符号拡張され）、内部送信レジスタに書き込まれます。圧伸規格では13ビット（A則）または14ビット（μ則）の最大ワード長に対応するだけですが、16ビットまでのワード長を使用できます。ワード値の絶対値が最大許容値を超える場合、値は正または負の最大値に自動的に圧縮されます。

圧伸動作は、16ビットを超える長さには対応できません。

クロック信号のオプション

各SPORTには送信クロック信号（TSCLK）と受信クロック信号（RSCLK）があります。クロック信号は、SPORTx_TCR1レジスタとSPORTx_RCR1レジスタのTCKFEビットとRCKFEビットによって設定されます。シリアル・クロック周波数は、SPORTx_TCLKDIVレジスタとSPORTx_RCLKDIVレジスタで設定されます。

送／受信に対して単一のクロックが望ましい場合には、受信クロック・ピンを送信クロックに接続できます。

送／受信クロックは独立して内部生成したり、外部ソースから入力したりできます。クロック源は、SPORTx_TCR1設定レジスタのITCLKビットとSPORTx_RCR1設定レジスタのIRCLKビットによって決定されます。

IRCLKまたはITCLK＝1の場合、クロック信号はコアによって内部生成され、TSCLK ピンまたは RSCLK ピンが出力です。クロック周波数は、SPORTx_RCLKDIVレジスタのシリアル・クロック・デバイザの値によって決定されます。

IRCLKまたはITCLK＝0の場合、クロック信号はTSCLKピンまたはRSCLKピンでの入力として受け付けられ、SPORTx_TCLKDIV/SPORTx_RCLKDIVレジスタのシリアル・クロック・デバイザは無視されます。外部生成されたシリアル・クロックは、コア・システム・クロックに同期したり、互いに同期したりする必要はありません。コア・システム・クロックには、RSCLKやTSCLKよりも高い周波数が要求されます。

フレーム同期のオプション

フレーミング信号は、各シリアル・ワード転送の始まりを示します。各SPORTのフレーミング信号は、TFS（送信フレーム同期）とRFS（受信フレーム同期）です。いろいろなフレーミング・オプションを使用できます。これらのオプションは、SPORT設定レジスタ（SPORTx_TCR1、SPORTx_TCR2、SPORTx_RCR1、SPORTx_RCR2）で設定されます。SPORTのTFS信号とRFS信号は独立しており、コントロール・レジスタで別個に設定されます。

■ フレーム付きとフレームなし

SPORT通信では、オプションで複数のフレーム同期信号を使用できます。フレーム同期信号が必要かどうかは、TFSR（送信フレーム同期要求選択）とRFSR（受信フレーム同期要求選択）の制御ビットによって決まります。これらのビットは、SPORTx_TCR1レジスタとSPORTx_RCR1レジスタにあります。

TFSR＝1またはRFSR＝1の場合、どのデータ・ワードにもフレーム同期信号が必要です。SPORTによる連続送信を可能にするには、それぞれの新しいデータ・ワードをSPORTx_TXホールド・レジスタにロードしてから、以前のワードをシフト・アウトして送信してください。

TFSR＝0またはRFSR＝0の場合、対応するフレーム同期信号は必要ありません。通信を開始するには1つのフレーム同期が必要ですが、最初のビットが転送された後では無視されます。その後、データ・ワードは、フレームなしで連続的に転送されます。

フレーム同期を必要としない状態では、フレームなしの連続したデータ・フローを保証するため、割込みやDMA要求はあまり頻繁に処理されない場合があります。データのアンダーフローやオーバーフローを検出するには、ステータス・ビットを監視したり、SPORTエラー割込みをチェックしたりします。

図12-17には、以下の特性を持つ、フレーム付きシリアル転送を示します。

- フレーム付きモードとフレームなしモードは、SPORTx_TCR1レジスタとSPORTx_RCR1レジスタのTFSRビットとRFSRビットで決まります。
- フレーム付きモードでは、どのワードにもフレーミング信号が必要です。フレームなしモードでは、最初のワード後のフレーミング信号を無視します。
- フレームなしモードは、連続受信に適しています。
- アクティブ・ローまたはアクティブ・ハイのフレーム同期は、SPORTx_TCR1レジスタとSPORTx_RCR1レジスタのLTFSビットとLRFSビットで選択されます。

他のタイミング例については、12-69ページの「タイミング例」を参照してください。

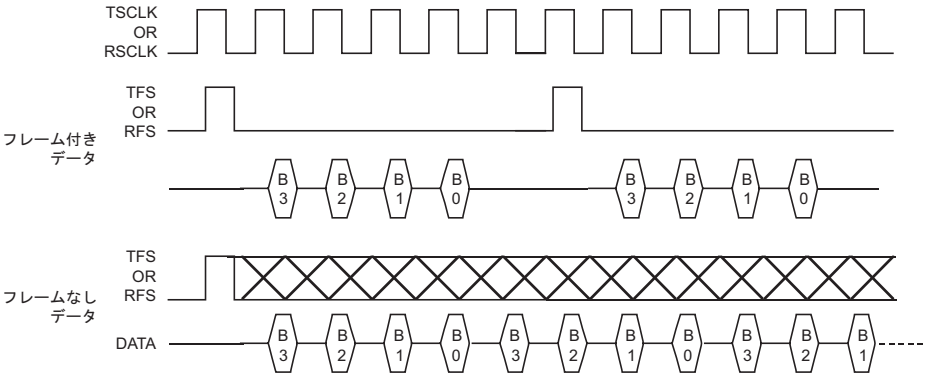


図 12-17.フレーム付きデータとフレームなしデータ

■ 内部フレーム同期と外部フレーム同期

送／受信フレーム同期は、内部的に独立して生成したり、外部ソースから入力したりできます。フレーム同期ソースは、SPORTx_TCR1 レジスタと SPORTx_RCR1 レジスタの ITFS ビットと IRFS ビットによって決まります。

ITFS＝1 または IRFS＝1 の場合、対応するフレーム同期信号は SPORT によって内部生成され、TFS ピンまたは RFS ピンが出力です。フレーム同期信号の周波数は、SPORTx_TFSDIV レジスタまたは SPORTx_RFSDIV レジスタのフレーム同期デバイザの値によって決まります。

ITFS＝0 または IRFS＝0 の場合、対応するフレーム同期信号は TFS ピンまたは RFS ピンでの入力として受け付けられ、SPORTx_TFSDIV/SPORTx_RFSDIV レジスタのフレーム同期デバイザは無視されます。

信号の内部生成／外部生成とは関係なく、すべてのフレーム同期オプションを使用できます。

■ アクティブ・ローのフレーム同期とアクティブ・ハイのフレーム同期

フレーム同期信号は、アクティブ・ハイまたはアクティブ・ロー（つまり、反転）にすることができます。フレーム同期のロジック・レベルは、SPORTx_TCR1レジスタとSPORTx_RCR1レジスタのLTFSビットとLRFSビットによって決まります。

- LTFS＝0またはLRFS＝0の場合、対応するフレーム同期信号はアクティブ・ハイです。
- LTFS＝1またはLRFS＝1の場合、対応するフレーム同期信号はアクティブ・ローです。

デフォルトでは、アクティブ・ハイのフレーム同期になります。プロセッサのリセット後、LTFSビットとLRFSビットは0に初期化されます。

■ データとフレーム同期のサンプリング・エッジ

データとフレーム同期は、SPORTクロック信号の立上がりエッジまたは立下がりエッジでサンプリングできます。シリアル・データとフレーム同期の駆動エッジとサンプリング・エッジは、SPORTx_TCR1レジスタと、SPORTx_RCR1レジスタのTCKFEビットとRCKFEビットによって選択されます。

SPORTトランスミッタでは、SPORTx_TCR1レジスタでTCKFE = 1に設定すると、データと内部生成されたフレーム同期の駆動にTSCLKxの立下がりエッジが選択され、外部生成されたフレーム同期のサンプリングにTSCLKxの立上がりエッジが選択されます。TCKFE = 0に設定すると、データと内部生成されたフレーム同期の駆動にTSCLKxの立上がりエッジが選択され、外部生成されたフレーム同期のサンプリングにTSCLKxの立下がりエッジが選択されます。

SPORT レシーバでは、SPORTx_RCR1 レジスタで RCKFE = 1 に設定すると、内部生成されたフレーム同期の駆動に RSCLKx の立下がりエッジが選択され、データと外部生成されたフレーム同期のサンプリングに RSCLKx の立上がりエッジが選択されます。RCKFE = 0 に設定すると、内部生成されたフレーム同期の駆動に RSCLKx の立上がりエッジが選択され、データと外部生成されたフレーム同期のサンプリングに RSCLKx の立下がりエッジが選択されます。

なお、外部生成されたデータとフレーム同期信号では、サンプリング用に選択されたエッジとは反対のエッジで状態を変更してください。たとえば、外部生成されたフレーム同期をクロックの立上がりエッジでサンプリングする（SPORTx_TCR1 レジスタの TCKFE = 1）には、フレーム同期はクロックの立下がりエッジで駆動する必要があります。

一緒に接続された 2 つの SPORT の送／受信機能は、トランスミッタの TCKFE とレシーバの RCKFE に対して常に同じ値を選択してください。これによって、トランスミッタがデータを駆動するエッジと、レシーバがデータをサンプリングするエッジは反対になります。

図 12-18 では、同じクロックとフレーム同期を共有するために、TCKFE = RCKFE = 0 で送／受信が一緒に接続されます。

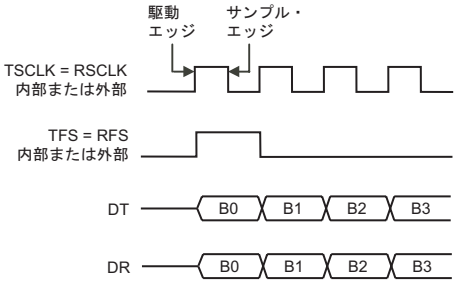


図 12-18.TCKFE ＝ RCKFE ＝ 0 で送／受信を接続した例

図 12-19では、同じクロックとフレーム同期を共有するために、TCKFE = RCKFE = 1で送／受信が一緒に接続されます。

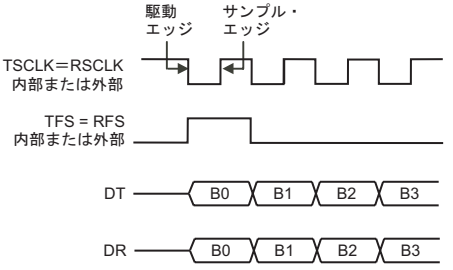


図 12-19.TCKFE ＝ RCKFE ＝ 1 で送／受信を接続した例

■ 早期フレーム同期と遅延フレーム同期（通常タイミングと代替タイミング）

フレーム同期信号は、各データ・ワードの最初のビットの処理中（遅延）、または最初のビットの直前にあるシリアル・クロック・サイクルの処理中（早期）に発生することができます。このオプションを設定するのは、SPORTx_TCR1レジスタと、SPORTx_RCR1レジスタのLARFSビットとLATFSビットです。

LATFS = 0またはLARFS = 0の場合、早期フレーム同期が設定されます。これは通常の動作モードです。このモードでは、送信データ・ワードの最初のビットが使用可能であり、受信データ・ワードの最初のビットはフレーム同期がアサートされた後のシリアル・クロック・サイクルでサンプリングされ、ワード全体が送信または受信されるまではフレーム同期は再チェックされません。マルチチャンネル動作では、これはマルチチャンネル・フレーム遅延が1のケースに対応します。

早期フレーミング・モードでデータ伝送が連続である（つまり、各ワードの最終ビットの直後に次のワードの最初のビットがある）場合には、各ワードの最終ビットの処理中にフレーム同期信号が発生します。早期フレーミング・モードでは、内部生成されたフレーム同期は1クロック・サイクルの間アサートされます。連続動作は、4以上のワード・サイズに限定されます（SLEN ≥ 3）。

LATFS = 1またはLARFS = 1の場合、遅延フレーム同期が設定されます。これは代替動作モードです。このモードでは、送信データ・ワードの最初のビットが使用可能であり、受信データ・ワードの最初のビットは、フレーム同期がアサートされたのと同じシリアル・クロック・サイクルでサンプリングされます。マルチチャンネル動作では、これはフレーム遅延が0のケースです。受信データビットはシリアル・クロック・エッジによってサンプリングされますが、フレーム同期信号は各ワードの最初のビットの処理中にのみチェックされます。遅延フレーミング・モードでは、内部生成されたフレーム同期はデータ・ワードの長さ全体にわたってアサートされたままです。外部生成されたフレーム同期は、最初のビットの処理中にのみチェックされます。

図 12-20は、フレーム信号タイミングの2つのモードを示します。

- SPORTx_TCR1またはSPORTx_RCR1レジスタのLATFSまたはLARFSビットの場合：早期フレーム同期ではLATFS = 0またはLARFS = 0、遅延フレーム同期ではLATFS = 1またはLARFS = 1。
- 早期フレーミングでは、フレーム同期はデータに1サイクルだけ先行します。遅延フレーミングでは、フレーム同期は最初のビットでのみチェックされます。
- データは、MSBファースト（TLSBIT = 0またはRLSBIT = 0）またはLSBファースト（TLSBIT = 1またはRLSBIT = 1）で送信されます。
- フレーム同期とクロックは、内部的または外部的に生成されます。

他の例については、12-69ページの「タイミング例」を参照してください。

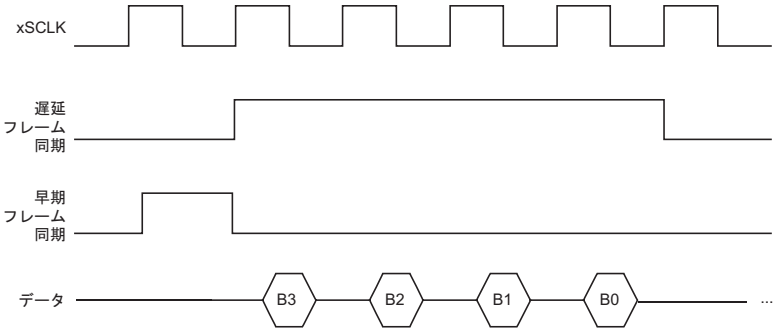


図 12-20.通常フレーミングと代替フレーミング

■ データ独立の送信フレーム同期

通常、内部生成された送信フレーム同期信号（TFS）が出力されるのは、SPORTx_TXバッファに送信準備のできたデータがある場合だけです。データ独立の送信フレーム同期選択ビット（DITFS）によって、新しいデータの有無とは無関係にTFS信号の連続生成が可能になります。このオプションは、SPORTx_TCR1レジスタのDITFSビットによって設定されます。

DITFS＝ 0の場合、内部生成されたTFSが出力されるのは、新しいデータ・ワードがSPORTx_TXバッファにロードされたときだけです。データがSPORTx_TXにロードされると、次のTFSが生成されます。この動作モードによって、データは使用可能な場合にだけ送信できます。

DITFS＝ 1の場合、新しいデータがSPORTx_TXバッファで使用可能かどうかとは無関係に、内部生成されたTFSは、そのプログラムされた間隔で出力されます。TFSがアサートされるたびに、SPORTx_TX内に存在するデータは再び送信されます。このとき、SPORTx_STATレジスタのTUVF（送信アンダーフロー・ステータス）ビットがセットされ、古いデータが再び送信さ

れます。外部生成されたTFSが発生したとき、SPORTx_TXバッファに新しいデータがない場合にも、TUVFステータス・ビットはセットされます。なお、この動作モードでは、データは指定されたタイミングでのみ送信されます。

内部生成されたTFSが使用される場合、転送を開始するには、SPORTx_TXデータ・レジスタに1回書き込むことが必要です。

SPORT とメモリとの間のデータ転送

送／受信データをSPORTとオンチップ・メモリとの間で転送する方法としては、シングル・ワード転送とDMAブロック転送という2つの方法があります。

SPORT DMAチャンネルがイネーブルにされてない場合、SPORTは、データ・ワードを受信したりデータ・ワードの送信を必要としたりするたびに割込みを生成します。SPORT DMAは、割込みが生成される前にシリアル・データのブロック全体または複数のブロックを送／受信するためのメカニズムを提供します。SPORTのDMAコントローラはDMA転送を処理するので、プロセッサ・コアは、データのブロック全体が送／受信されるまでは実行を継続することができます。これにより、割込みサービス・ルーチン（ISR）は、シングル・ワードではなくデータのブロックに作用できるため、オーバーヘッドが著しく減少します。

DMAの詳細については、第9章「ダイレクト・メモリ・アクセス」を参照してください。

ステレオ・シリアル動作

SPORTでは、よく使用されるI^{2}Sフォーマットを含めて、複数のステレオ・シリアル・モードをサポートできます。これらのモードを使用するには、SPORT_RCR2レジスタまたはSPORT_TCR2レジスタのビットを設定しま

す。SPORT_RCR2 またはSPORT_TCR2 のRSFSE またはTSFSE をセットすると、フレーム同期ピンの動作は、I²Sや左寄せのステレオ・シリアル・データで要求される左右クロックに変更されます。このビットをセットすると、SPORTはLRCLK スタイルの特殊なフレーム同期を生成したり受け付けたりできるようになります。他のすべてのSPORT制御ビットは有効なままであり、適切に設定してください。12-49ページの図12-21と12-50ページの図12-22は、ステレオ・シリアル・モード動作のタイミング図を示します。

表 12-3に、SPORTx_TCR1 とSPORTx_RCR1 のビットを使用して設定できるいくつかのモードを示します。この表ではSPORTの受信側のビットを示しますが、SPORTの送信部分の設定には、対応するビットを使用できます。標準を変更しなくても、ユーザのニーズに応じてセット／クリアできる制御フィールドは、「X」で示されています。

表 12-3.　ステレオ・シリアル設定

ビット・フィールド	ステレオ・オーディオ・シリアル方式		
	I²S	左寄せ	DSP モード
RSFSE	1	1	0
RRFST	0	0	0
LARFS	0	1	0
LRFS	0	1	0
RFSR	1	1	1
RCKFE	1	0	0
SLEN	2 – 31	2 – 31	2 – 31
RLSBIT	0	0	0
RFSDIV （内部 FS が選択された場合。）	2 – Max	2 – Max	2 – Max
RXSE （RX と TX にはセカンダリ・イネーブルを使用できます。）	X	X	X

なお、0または1で示されたビットの多くは、ユーザの好みに応じて変更できます。したがって、この他にも多くの「ほぼ標準」モードのステレオ・シリアル動作が生まれます。これらのモードは、わずかに非標準のインターフェースを持つコーデックとのインターフェースに役立つことがあります。表 12-3に示す設定値は、よく使用される多くのコーデックへのグルーレスなインターフェースを提供します。

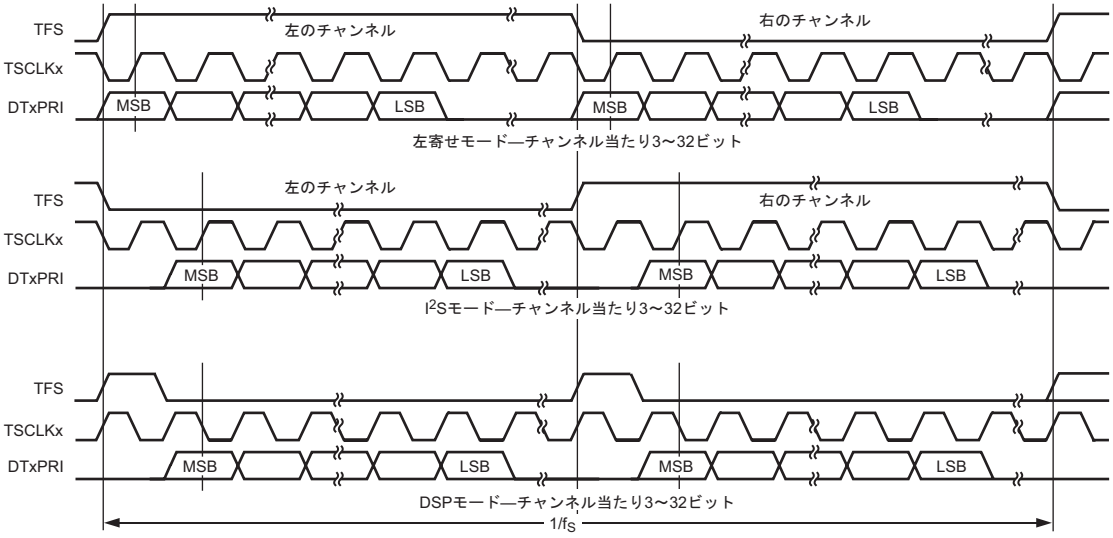
なお、RFSDIVやTFSDIVは、依然としてSLEN以上であることが必要です。I²S動作の場合、RFSDIVやTFSDIVは、一般にシリアル・クロック・レートの1/64です。RSFSEをセットした場合、フレーム同期周期と周波数を計算する式（12-33ページの「クロック周波数とフレーム同期周波数」を参照）はまだ適用されますが、今では周期の2分の1と周波数の2倍を表します。たとえば、RFSDIVまたはTFSDIV = 31に設定すると、その場合のLRCLKは32シリアル・クロック・サイクルごとに遷移し、64シリアル・クロック・サイクルの期間を持ちます。

LRFSビットは、「右」チャンネルと見なされるフレーム同期ピンの極性を決定します。したがって、LRFS = 0に設定すると、RFSピンまたはTFSピンでのローレベル信号が左チャンネルであることを示します。これはデフォルトの設定です。

送／受信される最初のワードが左のチャンネルであるか右のチャンネルであるかは、RRFSTビットとTRFSTビットによって決まります。ビットがセットされた場合には、送／受信される最初のワードは右のチャンネルです。デフォルトでは、左のチャンネルのワードを最初に送／受信します。

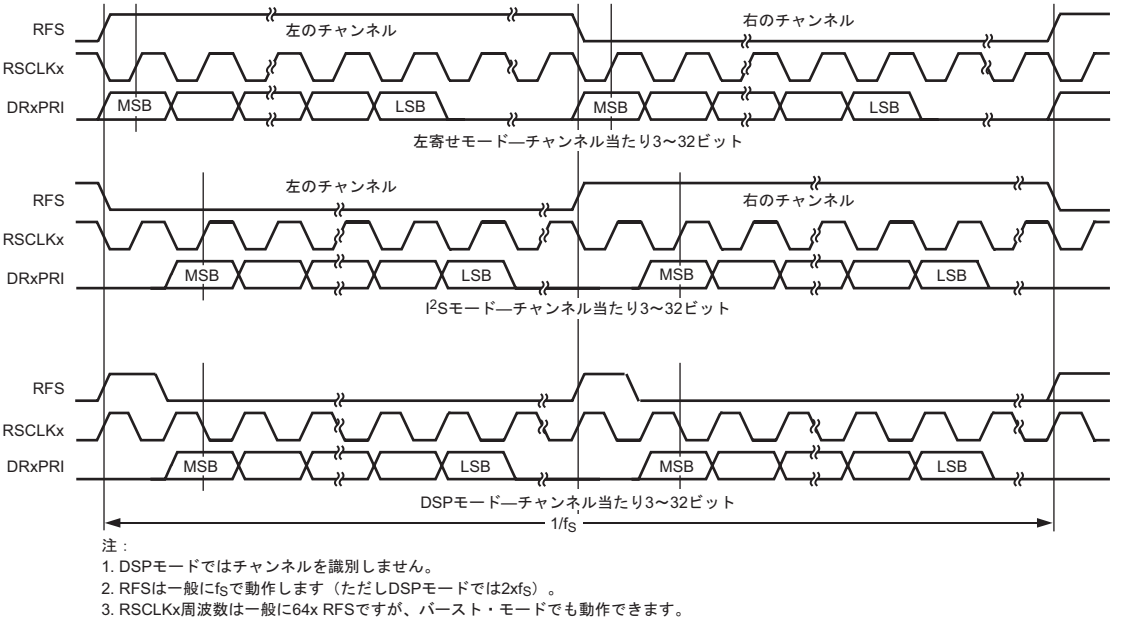
セカンダリのDRxSECピンとDTxSECピンは、ステレオ・シリアル・モードと釣り合いのよいシリアル・ポートの便利な拡張機能です。1つのSPORTを使用して、データの複数のI²Sストリームを送／受信できます。なお、プライマリ・ピンとセカンダリ・ピンは、クロックとLRCLK（フレーム同期）ピンを共有するので同期しています。SPORTの送信側と受信側は同期する必要はありませんが、設計によっては、1つのクロックを共有する

こともあります。12-8ページの図12-3では、プロセッサとAD1836コーデックとの間で複数のステレオ・シリアル接続を行った例を示しています。



注：
1. DSPモードではチャンネルを識別しません。
2. TFSは一般にf_Sで動作します（ただしDSPモードでは$2 \times f_S$）。
3. TSCLKx周波数は一般に64×TFSですが、バースト・モードでも動作できます。

図 12-21.SPORT のステレオ・シリアル・モード、送信



注：
1. DSPモードではチャンネルを識別しません。
2. RFSは一般にf_Sで動作します（ただしDSPモードでは2xf_S）。
3. RSCLKx周波数は一般に64x RFSですが、バースト・モードでも動作できます。

図 12-22.SPORT のステレオ・シリアル・モード、受信

マルチチャンネル動作

SPORTが提供するマルチチャンネル動作モードによって、SPORTは時分割多重（TDM）のシリアル・システムで通信できます。マルチチャンネル通信では、シリアル・ビットストリームの各データ・ワードは別個のチャンネルを占有します。各ワードは、次の連続したチャンネルに属します。たとえば、24ワード・ブロックのデータには、24チャンネルごとに1つのワードが含まれることになります。

SPORTは、特定チャンネル用のワードを自動的に選択しながら、他のワードを無視することができます。送／受信用には128までのチャンネルを使用できます。各SPORTは、任意の128チャンネルからデータを選択的に送／受信できます。これらの128チャンネルは、合計1024のチャンネル

から任意に選択できます。チャンネルを選択的にイネーブルにするため、RXとTXでは、同じ128チャンネル領域を使用する必要があります。SPORTは、各チャンネルで以下の動作を行うことができます。

- データの送信
- データの受信
- データの送／受信
- 何もしない

マルチチャンネル・モードでは、データ圧伸とDMA転送も使用できます。

SPORTがマルチチャンネル・モードにあって非アクティブなタイム・スロットが発生しない限り、SPORTがイネーブルである（SPORTx_TCR1レジスタのTSPEN ＝1）場合には、DTPRIピンは常に駆動されます（3ステートではありません）。SPORTがマルチチャンネル・モードにあって非アクティブなタイム・スロットが発生しない限り、SPORTがイネーブルでありセカンダリ送信がイネーブルである（SPORTx_TCR2レジスタのTXSE = 1）場合には、DTSECピンは常に駆動されます（3ステートではありません）。

マルチチャンネル・モードでは、RSCLKは外部から供給したり、SPORTによって内部生成したりできます。RSCLKは、送信機能と受信機能の両方に使用されます。SPORTがマルチチャンネル・モードでだけ使用される場合には、TSCLKを非接続のままにしておきます。RSCLKが外部的／内部的に供給される場合には、レシーバ回路とトランスミッタ回路の両方に内部的に配布されます。

SPORTマルチチャンネル送信セレクト・レジスタとSPORTマルチチャンネル受信セレクト・レジスタは、SPORTx_TXやSPORTx_RXの動作をマルチチャンネル・モード用にイネーブルにする前にプログラムする必要があります。このことは、「DMA データ・アンパック・モード」では特に重要です。なぜなら、SPORT FIFO 動作はRSPENとTSPENがセットされた直後に始まり、RXとTXの両方がイ

ネーブルにされるからです。SPORTx_MCMC2 の MCMEN ビットは、SPORTx_TX や SPORTx_RX の動作をイネーブルにする前にイネーブルにする必要があります。SPORT のマルチチャンネル動作をディスエーブルにするには、まず TXEN をディスエーブルにしてから、RXEN をディスエーブルにします。なお、再びイネーブルにする前に、TXEN と RXEN の両方をディスエーブルにする必要があります。TX または RX だけをディスエーブルにすることは許されません。

図 12-23は、以下の特性を持つマルチチャンネル転送のタイミング例を示します。

- 同じシリアル・バスを共有する異なるチャンネルでシリアル・データを送／受信する場合には、TDM方式を使用します。
- 送／受信チャンネルを独立に選択できます。
- RFSはフレームの開始を示します。
- TFSは、外部ロジックの「送信データ有効」として使用され、送信チャンネル区間でのみ真です。
- チャンネル0と2で受信し、チャンネル1と2で送信します。
- マルチチャンネル・フレーム遅延は1に設定されます。

他の例については、12-69ページの「タイミング例」を参照してください。

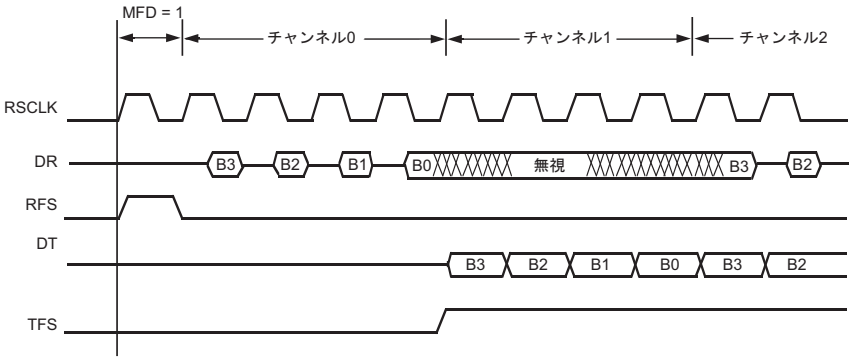


図 12-23.マルチチャンネル動作

■ SPORTx_MCMCn レジスタ

SPORT ごとに 2 本の SPORTx マルチチャンネル設定レジスタ（SPORTx_MCMCn）があります。SPORTx_MCMCn レジスタは、SPORT のマルチチャンネル動作の設定に使用されます。2本のコントロール・レジスタを次に示します。

SPORTx マルチチャンネル設定レジスタ 1（SPORTx_MCMC1）

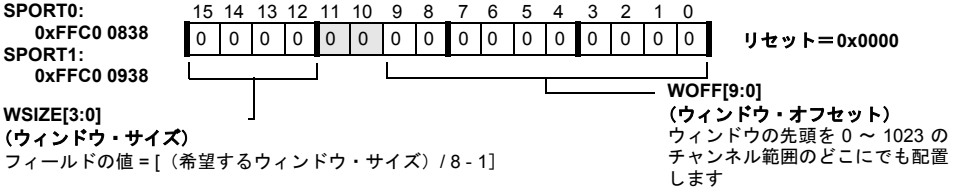


図 12-24.SPORTx マルチチャンネル設定レジスタ 1

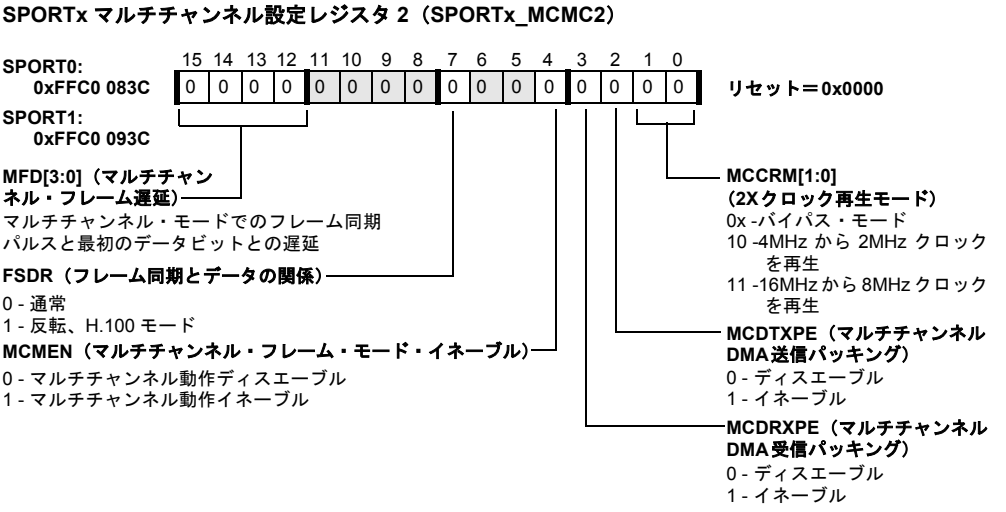


図 12-25.SPORTx マルチチャンネル設定レジスタ 2

■ マルチチャンネル・イネーブル

SPORTx_MCM2 レジスタの MCMEN ビットをセットすると、マルチチャンネル・モードがイネーブルにされます。MCMEN = 1 の場合、マルチチャンネル動作がイネーブルにされます。MCMEN = 0 の場合、すべてのマルチチャンネル動作がディスエーブルにされます。

MCMEN ビットをセットすると、SPORT の受信側と送信側の**両方**で、マルチチャンネル動作がイネーブルにされます。したがって、受信側の SPORT がマルチチャンネル・モードにある場合には、送信側の SPORT もマルチチャンネル・モードにあることが必要です。

マルチチャンネル・モードでは、ステレオ・シリアル・フレーム同期モードや遅延フレーム同期機能をイネーブルにしないでください。これらの機能は、マルチチャンネル・モードと不適合です。

表 12-4は、SPORTがマルチチャンネル・モードにある場合のSPORT設定レジスタ内のビットの依存関係を示します。

表 12-4. マルチチャンネル・モード設定

SPORTx_RCR1 または SPORTx_RCR2	SPORTx_TCR1 または SPORTx_TCR2	注
RSPEN	TSPEN	両方ともセットまたはクリア
IRCLK	–	独立
–	ITCLK	無視
RDTYPE	TDTYPE	独立
RLSBIT	TLSBIT	独立
IRFS	–	独立
–	ITFS	無視
RFSR	TFSR	無視
–	DITFS	無視
LRFS	LTFS	独立
LARFS	LATFS	両方とも 0 であることが必要
RCKFE	TCKFE	両方とも同じ値にセットまたはクリア
SLEN	SLEN	両方とも同じ値にセットまたはクリア
RXSE	TXSE	独立
RSFSE	TSFSE	両方とも 0 であることが必要
RRFST	TRFST	無視

■ マルチチャンネル・モードでのフレーム同期

マルチチャンネル・システムにおけるすべての送／受信デバイスは、同じタイミング・リファレンスを持つ必要があります。RFS信号は、このリファレンスに使用され、マルチチャンネル・データ・ワードのブロックまたはフレームの先頭を示します。

SPORTでマルチチャンネル・モードがイネーブルにされている場合、トランスミッタとレシーバでは、いずれもRFSをフレーム同期として使用します。このことは、RFSが内部生成されたか外部生成されたかとは無関係です。RFS信号はチャンネルの同期をとり、各マルチチャンネル・シーケンスを再起動するために使用されます。RFSのアサーションは、チャンネル0データ・ワードの先頭を示します。

マルチチャンネル・モード設定では、RFSは、SPORTのSPORTx_TXチャンネルとSPORTx_RXチャンネルの両方で使用されるため、SPORTx_RCR1とSPORTx_TCR1、およびSPORTx_RCR2とSPORTx_TCR2内の対応するビット・ペアは常に同じにプログラムしてください（ただし、RXSEとTXSEのペア、およびRDTYPEとTDTYPEのペアについては、例外となることもあります）。これは、たとえSPORTx_RX動作がイネーブルにされていない場合でも当てはまります。

マルチチャンネル・モードでは、遅延（代替）フレーム・モードに似たRFSタイミングには自動的に入ります。MFDが0に設定された場合、送信データ・ワードの最初のビットは使用可能であり、受信データ・ワードの最初のビットは、フレーム同期がアサートされたのと同じシリアル・クロック・サイクルでサンプリングされます。

TFS信号は、イネーブルにされたワードの伝送中にアクティブである送信データ有効信号として使用されます。SPORTのデータ送信ピンはタイム・スロットがアクティブでないときに3ステートであり、TFS信号はデータ送信ピンの出力イネーブル信号として機能します。マルチチャンネル・モードでは、ITFSがクリアされているかどうかとは無関係に、SPORTはTFSを駆動します。マルチチャンネル・モードでは、TFSピンは依然として

LTFSビットに従います。LTFSがセットされた場合には、送信データ有効信号はアクティブ・ローになります。つまり、TFSピンでのローレベル信号はアクティブ・チャンネルを示します。

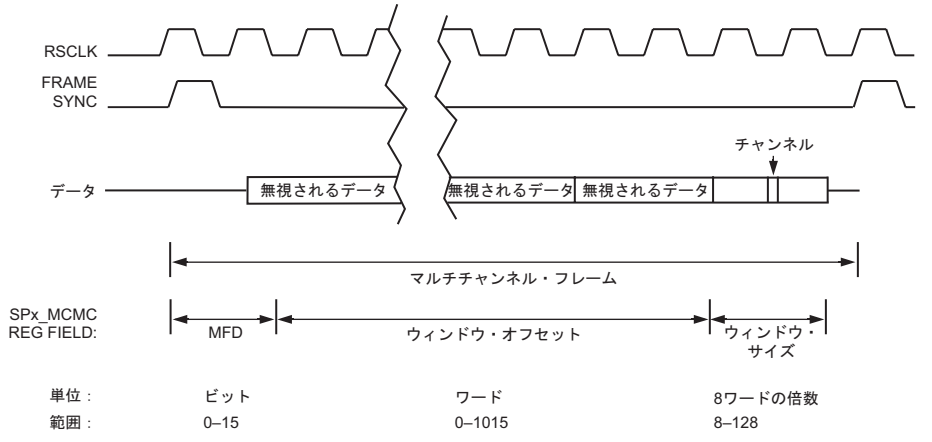
最初のRFSが受信され、フレーム転送が始まったら、完全なフレームが転送されるまで、SPORTは他のすべてのRFS信号を無視します。

MFD > 0の場合には、前のフレームの最後のチャンネルの処理中にRFSが発生することもあります。これは許容され、遅延したチャンネル0の開始点が完全なフレームの外側にある限り、フレーム同期は無視されません。

マルチチャンネル・モードでは、RFS信号はブロックまたはフレームの開始リファレンスとして使用され、その後は、それ以上のRFS信号を必要とすることなく、ワード転送が連続的に実行されます。したがって、内部生成されたフレーム同期は常にデータ独立です。

■ マルチチャンネル・フレーム

マルチチャンネル・フレームには、ウィンドウ・サイズとウィンドウ・オフセットによる指定に基づいて、複数のチャンネルが含まれています。完全なマルチチャンネル・フレームは、チャンネル0から始まる1～1024のチャンネルで構成されます。SPORT用に選択されるマルチチャンネル・フレームの特定チャンネルは、ウィンドウ・オフセット、ウィンドウ・サイズ、マルチチャンネル・セレクト・レジスタの組合わせです。



注：フレーム長は、フレーム同期分割または外部フレーム同期周期によって設定されます。

図 12-26.マルチチャンネル・パラメータの関係

■ マルチチャンネル・フレーム遅延

SPORTx_MCMC2 の4ビットMFDフィールドは、マルチチャンネル・モードでのフレーム同期パルスと最初のデータビットとの間の遅延を指定します。MFDの値は、遅延のシリアル・クロック・サイクル数です。マルチチャンネル・フレーム遅延によって、プロセッサは、さまざまな種類のインターフェース・デバイスを取り扱うことができます。

MFDの値が0の場合、フレーム同期は最初のデータビットと同時になります。MFDに許される最大値は15です。最後のフレームからのデータが受信される前に、新しいフレーム同期が発生することもあります。データのブロックはバック・ツー・バックで発生するからです。

■ ウィンドウ・サイズ

ウィンドウ・サイズ（WSIZE[3:0]）は、マルチチャンネル・セレクト・レジスタでイネーブル／ディスエーブルにできるチャンネル数を定義します。このようなワードの範囲は、アクティブ・ウィンドウと呼ばれます。チャンネル数は0～15の範囲の任意の値とすることができ、8を増分とする8～128のアクティブ・ウィンドウ・サイズに対応します。デフォルト値0は、8チャンネルという最小のアクティブ・ウィンドウ・サイズに対応します。WSIZEレジスタからアクティブ・ウィンドウ・サイズを計算するには、次の式を使用します。

アクティブ・ウィンドウ内のワード数＝8×（WSIZE+1）

DMAバッファ・サイズは常に固定されているため、DMA帯域幅を節約するには、小さなウィンドウ・サイズ（たとえば、32ワード）を定義して、小さなDMAバッファ・サイズ（この例では、128ワードではなく32ワード）を得ることができます。SPORTがイネーブルにされている間は、ウィンドウ・サイズを変更できません。

マルチチャンネル・セレクト・ビットは、イネーブルにされていても、選択されたウィンドウの範囲外になる場合には無視されます。

■ ウィンドウ・オフセット

ウィンドウ・オフセット（WOFF[9:0]）は、アクティブ・ウィンドウの先頭を1024のチャンネル範囲のどこに置くかを指定します。値0ではオフセットを指定しません。最大値は896であり、128チャンネルすべてを使用できます。一例として、プログラムでは、ウィンドウ・サイズが8（WSIZE = 0）、オフセットが93（WOFF = 93）のアクティブ・ウィンドウを定義できます。この8チャンネル・ウィンドウは、93から100の範囲に存在します。SPORTがイネーブルにされている間は、ウィンドウ・オフセットもウィンドウ・サイズも変更できません。

ウィンドウ・サイズとウィンドウ・オフセットの組合わせによって、ウィンドウの一部がチャンネル・カウンタの範囲外に置かれる場合には、フレーム内の範囲外チャンネルはいずれもイネーブルにされません。

■ SPORTx_CHNL レジスタ

SPORTxカレント・チャンネル・レジスタ（SPORTx_CHNL）の10ビットCHNLフィールドは、マルチチャンネル動作時に現在処理されているチャンネルを示します。このフィールドは、読出し専用のステータス・インジケータです。CHNL[9:0]フィールドは、各チャンネルが処理されるたびに1だけインクリメントされます。カウンタは、定義されたウィンドウの上端で停止します。チャンネル・セレクト・レジスタは、各フレーム同期で0から再開されます。一例として、ウィンドウ・サイズが8、オフセットが148の場合には、カウンタは0～156の値を表示します。

ウィンドウ・サイズが完了すると、次のフレームの準備のためにチャンネル・カウンタは0にリセットされます。RSCLKとプロセッサ・クロックとの間には同期遅延があるため、チャンネル・レジスタ値は近似値となります。処理されているチャンネルよりも進むことはありませんが、遅れることはあります。

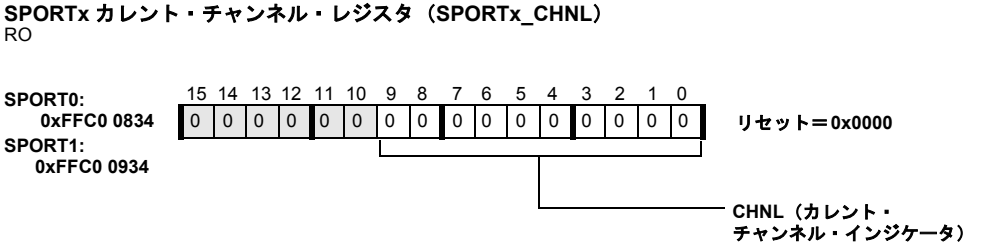


図 12-27.SPORTx カレント・チャンネル・レジスタ

■ SPORTx_MCMC2 内のその他のマルチチャンネル・フィールド

SPORTx_MCMC2 レジスタの FSDR ビットは、フレーム同期と受信するクロックとの間のタイミング関係を変更します。この変更によって、SPORT は H.100 プロトコルに準拠することができます。

通常（FSDR = 0 の場合）、データは TFS が生成されるのと同じエッジで送信されます。たとえば、TFS での立上がりエッジによって、データは LATFS の設定タイミングに応じて TSCLK の同じ立上がりエッジまたは次の立上がりエッジで送信されます。

フレーム同期／データの関係が使用された場合（FSDR = 1）、フレーム同期はクロックの立下がりエッジで変更されることが予想され、クロックの立上がりエッジでサンプリングされます。このことは、受信されたデータが受信クロックの立下がりエッジでサンプリングされる場合にも当てはまります。

■ チャンネル・セレクト・レジスタ

チャンネルは、長さ 3～32 ビットのマルチビット・ワードであり、TDM チャンネルの 1 つに属します。マルチチャンネル通信時に送／受信されるワードを選択するため、特定のチャンネルを個々にイネーブル／ディスエーブルできます。イネーブルにされたチャンネルからのデータ・ワードは送／受信されますが、ディスエーブルにされたチャンネルのワードは無視されます。使用可能な 1024 のチャンネルのうち、128 までの連続チャンネルを選択できます。個々のチャンネルをイネーブル／ディスエーブルにするには、SPORTx_MRCSn と SPORTx_MTCSn のマルチチャンネル・セレクト・レジスタが使用されます。SPORTx_MRCSn レジスタではアクティブ受信チャンネルを指定し、SPORTx_MTCSn レジスタではアクティブ送信チャンネルを指定します。

各マルチチャンネル・セレクト・レジスタは、4本のレジスタで構成されます。4本のレジスタのそれぞれには、32のチャンネルに対応して32のビットがあります。ビットをセットするとそのチャンネルがイネーブルにされるため、SPORTは複数ワード・ブロックのデータから（受信または送信用に）そのワードを選択します。

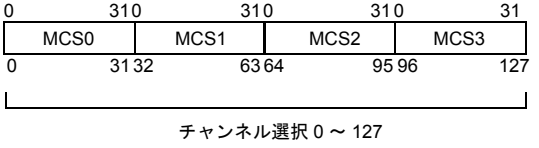


図 12-28.マルチチャンネル・セレクト・レジスタ

チャンネル選択ビット0は、常にアクティブ・ウィンドウの最初のワードに対応します。フレーム内でのチャンネルの絶対位置を決定するには、チャンネル選択位置にウィンドウ・オフセット・ワードを加算します。たとえば、MCS2のビット7をセットすると、イネーブルにされるアクティブ・ウィンドウのワード71が選択されます。MCS1のビット2をセットすると、アクティブ・ウィンドウのワード34が選択され、以下も同様です。

SPORTx_MTCSnレジスタの特定ビットをセットすると、SPORTは、データストリームのそのチャンネルの位置にあるワードを送信します。SPORTx_MTCSnレジスタのビットをクリアすると、そのチャンネルのタイム・スロットの間、SPORTのデータ送信ピンは3ステートになります。

SPORTx_MRCSnレジスタの特定ビットをセットすると、SPORTは、データストリームのそのチャンネルの位置にあるワードを受信します。受信されたワードは、SPORTx_RXバッファにロードされます。SPORTx_MRCSnレジスタのビットをクリアすると、SPORTはそのデータを無視します。

圧伸は、すべてのチャンネルに選択したり、どのチャンネルにも選択しなかったりできます。A則またはμ則の圧伸は、SPORTx_TCR1 レジスタの TDTYPE フィールドと SPORTx_RCR1 レジスタの RDTYPE フィールドで選択され、すべてのアクティブ・チャンネルに適用されます。(圧伸の詳細については、12-37ページの「圧伸」を参照してください)。

▶ SPORTx_MRCSn レジスタ

マルチチャンネル・セレクト・レジスタは、個々のチャンネルをイネーブル／ディスエーブルにするために使用されます。SPORTxマルチチャンネル受信セレクト・レジスタ（SPORTx_MRCSn）は、アクティブ受信チャンネルを指定します。32ビットのレジスタが4本あり、128のチャンネルに対応しています。ビットをセットするとそのチャンネルがイネーブルになり、シリアル・ポートでは、そのワードを複数ワード・ブロックのデータからの受信用に選択します。たとえば、ビット0をセットするとワード0が選択され、ビット12をセットするとワード12が選択され、以下も同様です。

SPORTx_MRCSn レジスタの特定ビットをセットすると、シリアル・ポートでは、データストリームのそのチャンネルの位置にあるワードを受信します。受信されたワードは、RXバッファにロードされます。SPORTx_MRCSn レジスタのビットをクリアすると、シリアル・ポートはデータを無視します。

SPORTx マルチチャンネル受信セレクト・レジスタ（SPORTx_MRCSn）

すべてのビットで、0 =チャンネル・ディスエーブル、1 =チャンネル・イネーブルであるため、SPORT は、そのワードを複数のワード・ブロックのデータから選択します。

メモリマップド・アドレスの場合、表 12-5 を参照してください。

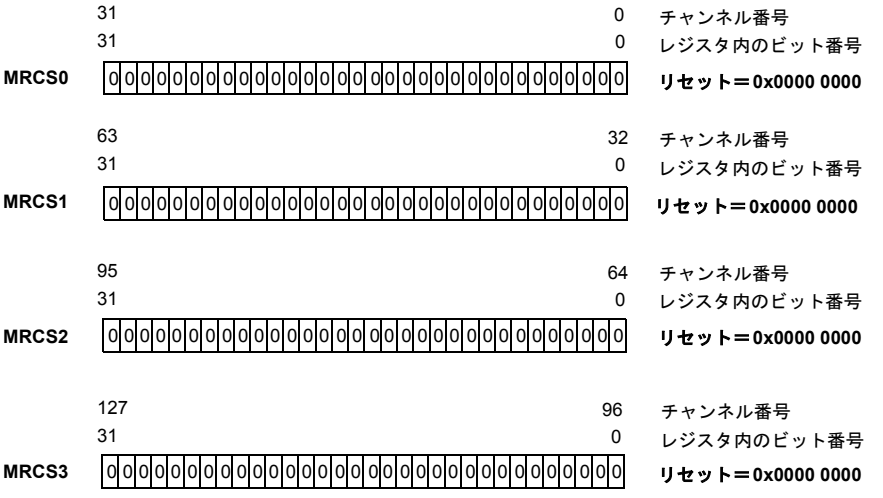


図 12-29.SPORTx マルチチャンネル受信セレクト・レジスタ

表 12-5. SPORTx マルチチャンネル受信セレクト・レジスタのメモリマップド・アドレス

レジスタ名	メモリマップド・アドレス
SPORT0_MRCS0	0xFFC0 0850
SPORT0_MRCS1	0xFFC0 0854
SPORT0_MRCS2	0xFFC0 0858
SPORT0_MRCS3	0xFFC0 085C
SPORT1_MRCS0	0xFFC0 0950
SPORT1_MRCS1	0xFFC0 0954
SPORT1_MRCS2	0xFFC0 0958
SPORT1_MRCS3	0xFFC0 095C

▶ SPORTx_MTCSn レジスタ

マルチチャンネル・セレクト・レジスタは、個々のチャンネルをイネーブル／ディスエーブルにするために使用されます。4本のSPORTxマルチチャンネル送信セレクト・レジスタ（SPORTx_MTCSn）は、アクティブ送信チャンネルを指定します。32ビットのレジスタが4本あり、128のチャンネルに対応しています。ビットをセットするとそのチャンネルがイネーブルになり、シリアル・ポートでは、そのワードを複数ワード・ブロックのデータからの送信用に選択します。たとえば、ビット0をセットするとワード0が選択され、ビット12をセットするとワード12が選択され、以下も同様です。

SPORTx_MTCSn レジスタの特定ビットをセットすると、シリアル・ポートでは、データストリームのそのチャンネルの位置にあるワードを送信します。SPORTx_MTCSn レジスタのビットをクリアすると、シリアル・ポートのデータ送信ピンは、そのチャンネルのタイム・スロットの間、3ステートになります。

SPORTx マルチチャンネル送信セレクト・レジスタ（SPORTx_MTCSn）
すべてのビットで、0 ＝チャンネル・ディスエーブル、1 ＝チャンネル・イネーブルであるため、SPORTは、そのワードを複数のワード・ブロックのデータから選択します。
メモリマップド・アドレスの場合、表 12-5 を参照してください。

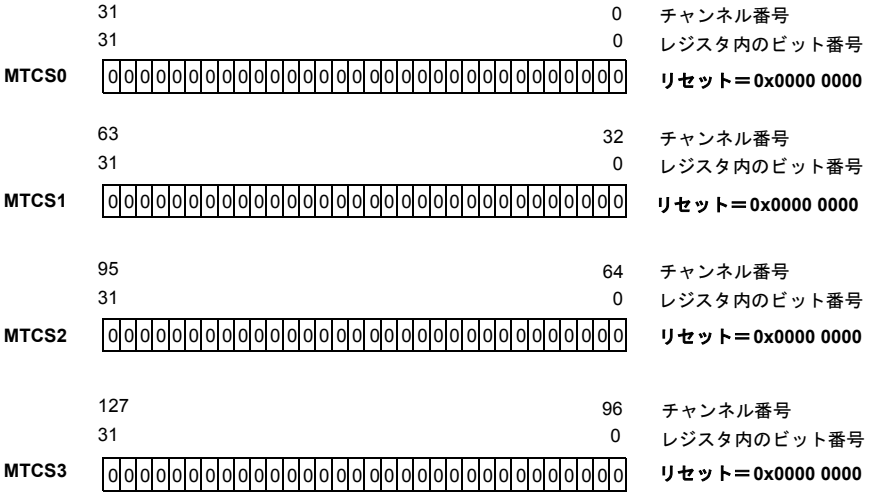


図 12-30.SPORTx マルチチャンネル送信セレクト・レジスタ

表 12-6. SPORTx マルチチャンネル送信セレクト・レジスタのメモリマップド・アドレス

レジスタ名	メモリマップド・アドレス
SPORT0_MTCS0	0xFFC0 0840
SPORT0_MTCS1	0xFFC0 0844
SPORT0_MTCS2	0xFFC0 0848
SPORT0_MTCS3	0xFFC0 084C
SPORT1_MTCS0	0xFFC0 0940
SPORT1_MTCS1	0xFFC0 0944
SPORT1_MTCS2	0xFFC0 0948
SPORT1_MTCS3	0xFFC0 094C

■ マルチチャンネル DMA のデータ・パッキング

マルチチャンネル DMA のデータ・パッキングとアンパッキングは、SPORTx_MCMC2 マルチチャンネル設定レジスタの MCDTXPE ビットと MCDRXPE ビットによって指定されます。

これらのビットがセットされた場合には、データがパックされることを示し、SPORT は、DMA バッファに格納されたデータがイネーブルにされた SPORT チャンネルにだけ対応することを期待します。たとえば、MCM フレームがイネーブルにされた 10 のチャンネルを含む場合、SPORT は、DMA バッファがフレームごとに 10 の連続したワードを含むことを期待します。イネーブルにされるチャンネルの合計数を変更するには、DMA バッファ・サイズを変更する必要があります。また、SPORT がイネーブルにされている間は、再設定は許されません。

これらのビットがクリアされた場合には（デフォルトであり、データがパックされていないことを示します）、SPORT はイネーブルかどうかとは無関係に、DMA バッファがアクティブ・ウィンドウ内のチャンネルごとに 1 ワードを持つことを期待します。したがって、DMA バッファ・サイズは、ウィンドウのサイズに等しくなければなりません。たとえば、チャ

ンネル1と10がイネーブルにされ、ウィンドウ・サイズが16である場合、DMAバッファ・サイズは16ワードになる必要があります。送／受信されるデータは、バッファのアドレス1と10に置かれ、DMAバッファ内の残りのワードは無視されます。このモードによって、SPORTがイネーブルである間には、イネーブルにされるチャンネルの数を変更できますが、いくつか注意が必要です。まず、チャンネル・レジスタを読み出して、アクティブ・ウィンドウが処理中でないことを確認します。チャンネル・カウントが0の場合、マルチチャンネル・セレクト・レジスタを更新できます。

H.100 標準プロトコルのサポート

プロセッサは、H.100標準プロトコルに対応します。この標準に対応するには、以下のSPORTパラメータを設定する必要があります。

- 外部フレーム同期の設定。外部バス・マスターによって生成されるフレーム同期。
- TFSR/RFSRの設定（フレーム同期が必要）
- LTFS/LRFSの設定（アクティブ・ローのフレーム同期）
- 外部クロックの設定
- MCMENの設定（マルチチャンネル・モードの選択）
- MFD ＝ 0（フレーム同期と最初のデータビットとの間にフレーム遅延なし）
- SLEN＝7（8ビット・ワード）
- FSDR ＝ 1（H.100 の設定、1/2 クロック・サイクルの早期フレーム同期をイネーブル）

■ 2X クロック再生コントロール

SPORTは提供された2X入力クロックからデータレート・クロックを再生できます。これによって、適切な位相関係を持つ4MHzの着信クロックから2MHzを再生したり16MHzの着信クロックから8MHzを再生することで、MVIP-90（2Mbpsデータ）とHMVIP（8Mbpsデータ）のH.100互換モードを実装できます。適用可能なクロック・モードは、2ビットのモード信号（SPORTx_MCMC2レジスタのMCCRM[1:0]）によって選択され、これには非分割、つまり通常動作用のバイパス・モードも含まれます。MCCRM = 00では非分割、つまりバイパス・モードが選択され（H.100互換）、MCCRM = 10ではMVIP-90のクロック分割が選択され（4MHzから2MHzを抽出）、MCCRM = 11ではHMVIPのクロック分割が選択されます（16MHzから8MHzを抽出）。

SPORT ピン／ライン終端

プロセッサは、SPORTを含めて、すべての出力ピンで非常に高速なドライバを持っています。データ・ライン、クロック・ライン、フレーム同期ラインでの接続が6インチを超える場合には、ポイント・ツー・ポイント接続でのストリップ・ラインに対して直列終端の使用を検討してください。低速のシリアル・クロックを使用するときにも、エッジ・レートの関係で、この方法が必要なこともあります。

タイミング例

この章の本文には、いくつかのタイミング例を掲載しています（12-38ページの「フレーム付きとフレームなし」、12-43ページの「早期フレーム同期と遅延フレーム同期（通常タイミングと代替タイミング）」、12-56ページの「マルチチャンネル・モードでのフレーム同期」）。ここでは、フレーミング・オプションのその他の組合わせを示す例を追加します。

これらのタイミング例では信号間の関係を示しますが、プロセッサの実際のタイミング・パラメータを示すようには調整されていません。実際のタイミング・パラメータと値については、『ADSP-BF531/ADSP-BF532/ADSP-BF533 Embedded Processor Data Sheet』を参照してください。

以下の例では、4ビットのワード長を想定します（SLEN = 3）。フレーミング信号はアクティブ・ハイです（LRFS = 0およびLTFS = 0）。

図 12-31～図 12-36に、データを受信するためのフレーミングを示します。

図 12-31と図 12-32は、不連続データ（ワード間に任意の数のTSCLKサイクルやRSCLKサイクルが存在）と連続データ（ワード間にTSCLKサイクルやSCLKサイクルがない）に対する通常のフレーミング・モードを示します。

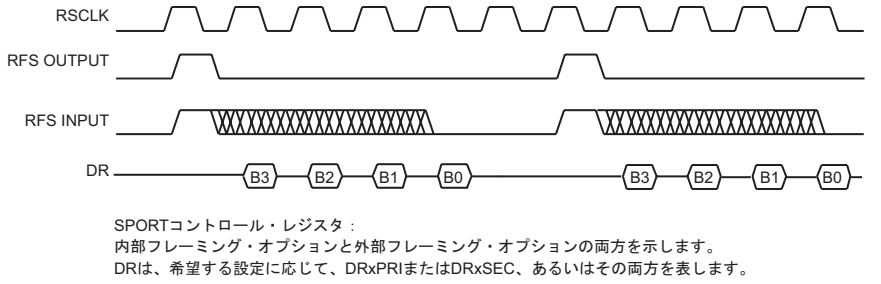


図 12-31.SPORT の受信、通常のフレーミング

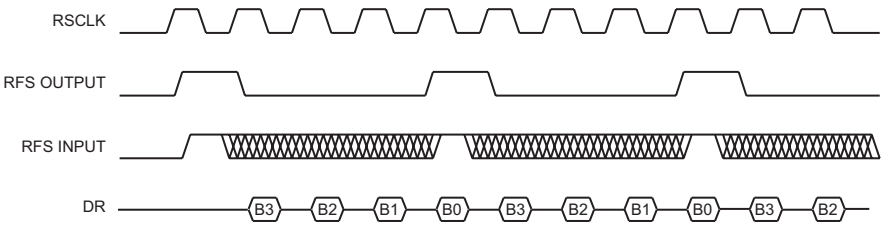


図 12-32.SPORT の連続受信、通常のフレーミング

図 12-33と図 12-34は、代替フレーミング・モードでの不連続受信と連続受信を示します。これら4つの図では、外部生成されたフレーム同期の入力タイミング条件と、内部生成されたフレーム同期の出力タイミング特性を示します。なお、出力は入力タイミング条件を満たします。したがって、2つのSPORTチャンネルを使用して、一方のSPORTチャンネルがもう一方のSPORTチャンネルにRFSを提供できます。

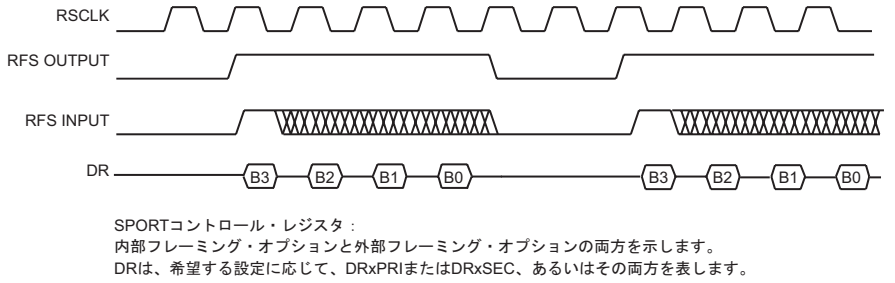


図 12-33.SPORT の受信、代替フレーミング

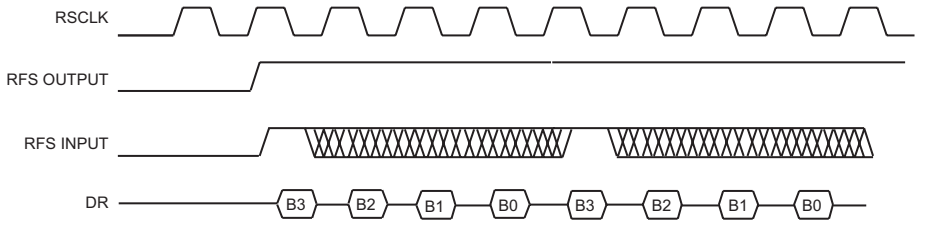


図 12-34.SPORT の連続受信、代替フレーミング

図 12-35と図 12-36は、フレームなしモードでの、通常のフレーミングと代替フレーミングによる受信動作を示します。1つのフレーム同期信号は、最初のワードの先頭でのみ発生します。ただし、ノーマル・モードでは最初のビットの1 RSCLK前になり、代替モードでは最初のビットと同時になります。このモードは、マルチワード・バースト（連続受信）に適しています。

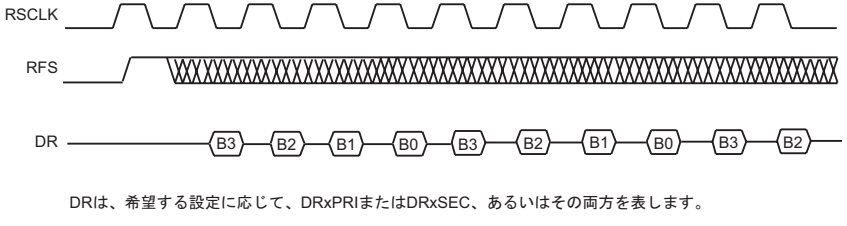


図 12-35.SPORT の受信、フレームなしモード、通常のフレーミング

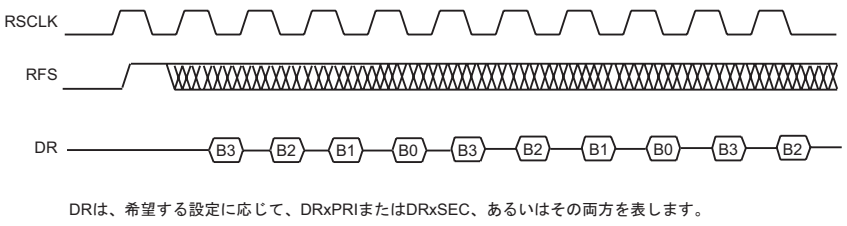


図 12-36.SPORT の受信、フレームなしモード、代替フレーミング

図 12-37～図 12-42は、データを送信するためのフレーミングを示します。これらは図 12-31～図 12-36にとてもよく似ています。

図 12-37と図 12-38は、不連続データ（ワード間に任意の数のTSCLKサイクルが存在）と連続データ（ワード間にTSCLKサイクルがない）に対する通常のフレーミング・モードを示します。図 12-39と図 12-40は、代替フレーミング・モードでの不連続伝送と連続伝送を示します。すでに受信タイミング図で述べたように、RFS出力はRFS入力タイミング条件を満たします。

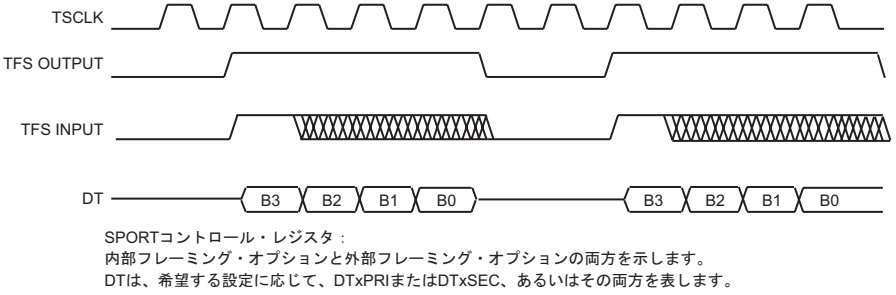
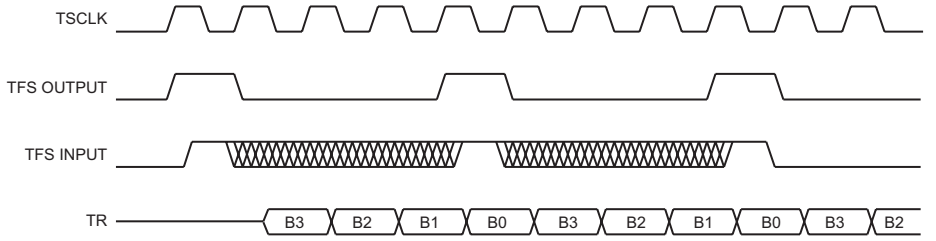
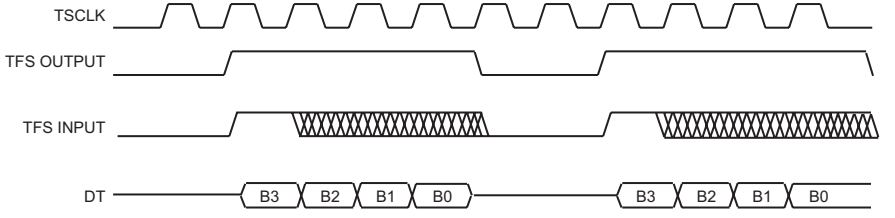


図 12-37.SPORT の送信、通常のフレーミング



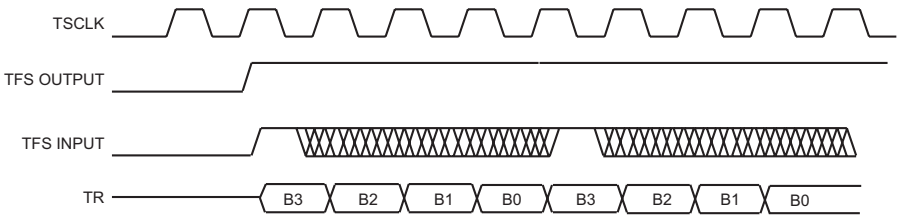
SPORTコントロール・レジスタ：
内部フレーミング・オプションと外部フレーミング・オプションの両方を示します。
DRIは、希望する設定に応じて、DRxPRIまたはDRxSEC、あるいはその両方を表します。

図 12-38.SPORT の連続送信、通常のフレーミング



SPORTコントロール・レジスタ：
内部フレーミング・オプションと外部フレーミング・オプションの両方を示します。
DTは、希望する設定に応じて、DTxPRIまたはDTxSEC、あるいはその両方を表します。

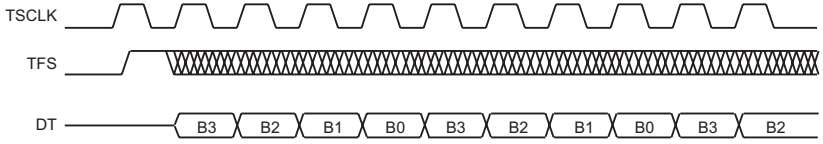
図 12-39.SPORT の送信、代替フレーミング



SPORTコントロール・レジスタ：
内部フレーミング・オプションと外部フレーミング・オプションの両方を示します。
DRは、希望する設定に応じて、DRxPRIまたはDRxSEC、あるいはその両方を表します。

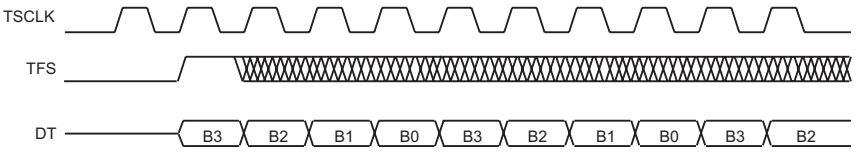
図 12-40.SPORT の連続送信、代替フレーミング

図 12-41 と図 12-42 は、フレームなしモードにおいて、通常のフレーミングと代替フレーミングによる送信動作を示します。1つのフレーム同期信号は、最初のワードの先頭でのみ発生します。ただし、ノーマル・モードでは最初のビットの1 TSCLK 前になり、代替モードでは最初のビットと同時になります。



DTは、希望する設定に応じて、DTxPRIまたはDTxSEC、あるいはその両方を表します。

図 12-41.SPORT の送信、フレームなしモード、通常のフレーミング



DTは、希望する設定に応じて、DTxPRIまたはDTxSEC、あるいはその両方を表します。

図 12-42.SPORT の送信、フレームなしモード、代替フレーミング

第13章　UARTポート・コントローラ

非同期シリアル・インターフェースLSI（UART）は、PCスタイルの業界標準UARTと互換性のある全二重ペリフェラルです。UARTは、データ・フォーマットの直列／並列変換を行います。シリアル通信は、さまざまなワード長、ストップ・ビット、パリティ生成オプションに対応する非同期プロトコルに従います。UARTには、割込み処理ハードウェアが組み込まれています。割込みは、12種類のイベントから生成できます。

UARTは、半二重IrDA®（赤外線通信協会）SIR（9.6/115.2 Kbpsレート）プロトコルに対応します。これはモードを使用する機能です。

モデム・ステータス／制御機能は、UARTモジュールでは提供していませんが、汎用I/O（GPIO）ピンを使用して実装できます。

UARTは、TXとRXのDMAマスター・チャンネルを別個に提供するDMA対応のペリフェラルであり、DMA動作モードまたはプログラムされた非DMA動作モードで使用できます。非DMAモードは、割込みまたはポーリングを使用して、データ・フローをソフトウェア管理することが必要です。DMA方式は、DMAエンジン自身がデータを転送するので、ソフトウェア介入を最小限に抑えられます。DMAの詳細については、第9章「ダイレクト・メモリ・アクセス」を参照してください。

UARTで使用するハードウェア支援のオートボー検出メカニズムを提供するには、どのペリフェラル・タイマでも使用できます。詳細については、第15章「タイマ」を参照してください。

シリアル通信

UARTは、以下のオプションを持つ非同期シルアル通信プロトコルに従います。

- 5～8のデータビット
- 1、1 ½、または2つのストップ・ビット
- なし、偶数、または奇数パリティ
- ボーレート＝SCLK/(16 × Divisor)。ここでSCLKはシステム・クロック周波数、デバイザには1～65536の値を使用できます。

すべてのデータ・ワードには、1つのスタート・ビットと少なくとも1つのストップ・ビットが必要です。したがって、オプションのパリティ・ビットを付加して、ワード当たり7～12ビットの範囲となります。送／受信される文字フレームのフォーマットは、ライン・コントロール・レジスタ（UART_LCR）によって制御されます。データは常に最下位ビット（LSB）ファーストで送／受信されます。

図 13-1は、TXピンで測定される代表的な物理ビットストリームを示します。

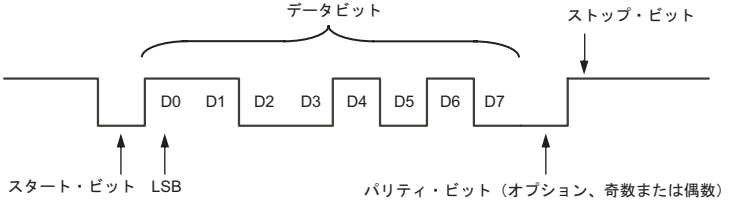


図 13-1. TX ピンでのビットストリーム

UART コントロール・レジスタとステータス・レジスタ

このプロセッサは、一連のPCスタイルの業界標準コントロール・レジスタとステータス・レジスタを各UARTに提供します。これらのメモリマップド・レジスタ（MMR）はバイト幅レジスタであり、最上位バイトがゼロ・フィルされたハーフ・ワードとしてマッピングされます。

業界標準のインターフェースに合わせて、複数のレジスタが同じアドレス位置にマッピングされます。デバイザ・ラッチ・レジスタ（UART_DLHとUART_DLL）は、送信ホールディング・レジスタ（UART_THR）、受信バッファ・レジスタ（UART_RBR）、割込みイネーブル・レジスタ（UART_IER）とアドレスを共有します。規定の時間にアクセス可能なレジスタ・セットは、ライン・コントロール・レジスタ（UART_LCR）のデバイザ・ラッチ・アクセス・ビット（DLAB）によって制御されます。これらのレジスタにアクセスするために、ソフトウェアは16ビット・ワードのロード／ストア命令を使用する必要があります。

送／受信チャンネルは、いずれもバッファリングされます。UART_THRレジスタは送信シフト・レジスタ（TSR）をバッファし、UART_RBRレジスタは受信シフト・レジスタ（LSR）をバッファします。ソフトウェアは、シフト・レジスタに直接アクセスできません。

■ UART_LCR レジスタ

UARTライン・コントロール・レジスタ（UART_LCR）は、送／受信される文字フレームのフォーマットを制御します。SBビットは、たとえUARTクロックがディスエーブルにされていても機能します。TXピンは、通常はハイレベルに駆動されるため、UARTが使用されない場合には、フラグ出力ピンとして使用できます。

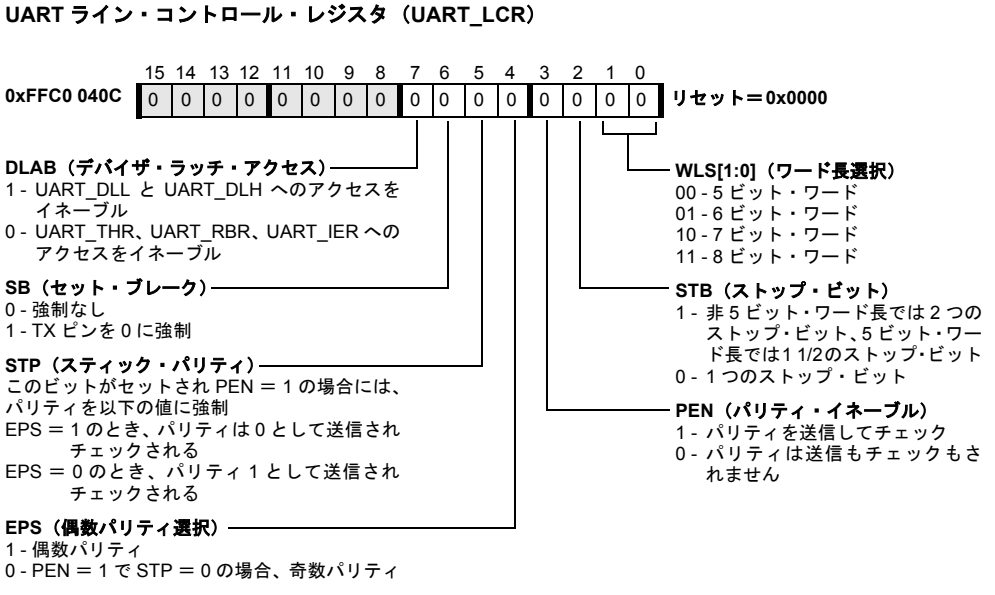


図 13-2. UART ライン・コントロール・レジスタ

■ UART_MCR レジスタ

図 13-3に示すように、モデム・コントロール・レジスタ（UART_MCR）はUARTポートを制御します。たとえモデム機能が提供されていない場合でも、モデム・コントロール・レジスタは、ループバック・モードに対応するために使用可能です。

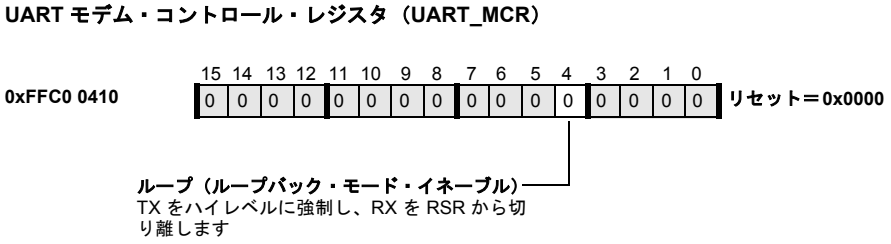


図 13-3. UART モデム・コントロール・レジスタ

ループバック・モードでは、TXピンがハイレベルに強制され、レシーバの入力はRXピンから切り離されますが、内部的に送信出力にリダイレクトされます。

■ UART_LSR レジスタ

図 13-4に示すように、UARTライン・ステータス・レジスタ（UART_LSR）には、UARTステータス情報が含まれています。

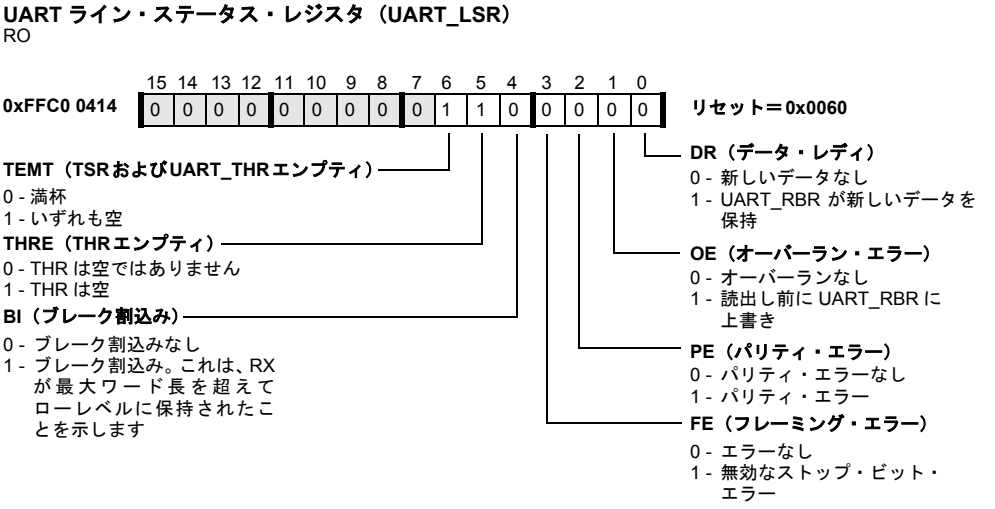


図 13-4. UART ライン・ステータス・レジスタ

UARTライン・ステータス・レジスタ（UART_LSR）が読み出されると、ブレーク割込み（BI）、オーバーラン・エラー（OE）、パリティ・エラー（PE）、フレーミング・エラー（FE）の各ビットがクリアされます。UART受信バッファ・レジスタ（UART_RBR）が読み出されると、データ・レディ（DR）ビットがクリアされます。

これらの読出し動作には破壊的性質があるため、特別な注意が必要です。詳細については、6-75ページの「投機的ロード実行」と6-76ページの「条件付きロード動作」を参照してください。

THRE ビットは、UART 送信チャンネルで新しいデータの準備ができたこと、およびソフトウェアが UART_THR に書込みできることを示します。UART_THR への書込みは、THRE ビットをクリアします。このビットは、データが UART_THR から送信シフト・レジスタ（TSR）にコピーされると再びセットされます。TEMT ビットを評価すれば、最近開始された送信動作が完了したかどうかを判断できます。

■ UART_THR レジスタ

UART 送信ホールディング・レジスタ（UART_THR）への書込みによって、送信動作が開始されます。データは内部の送信シフト・レジスタ（TSR）に転送され、そこで必要に応じてスタート・ビット、ストップ・ビット、パリティ・ビットが付加され、SCLK/(16 × Divisor）と等しいボーレートでシフト・アウトされます。すべてのデータ・ワードは、スタート・ビットの 1 から 0 への遷移で始まります。UART_THR から送信シフト・レジスタへのデータ転送によって、UART ライン・ステータス・レジスタ（UART_LSR）の送信ホールディング・レジスタ・エンプティ（THRE）ステータス・フラグがセットされます。

書込み専用の UART_THR レジスタは、読出し専用の UART_RBR レジスタや UART_DLL レジスタと同じアドレスにマッピングされます。UART_THR にアクセスするには、UART_LCR の DLAB ビットをクリアする必要があります。DLAB ビットがクリアされると、このアドレスへの書込みは UART_THR レジスタをターゲットにし、このアドレスからの読出しは UART_RBR レジスタを返します。

なお、データは最下位ビット（LSB）ファースト（ビット0）で送／受信され、最上位ビット（MSB）が続きます。

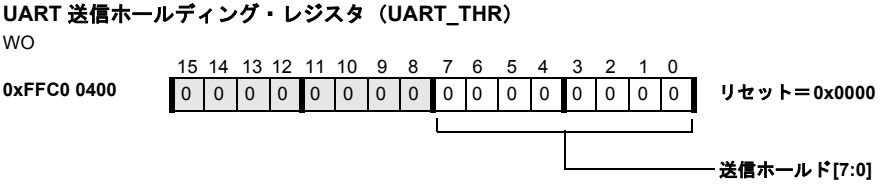


図 13-5. UART 送信ホールディング・レジスタ

■ UART_RBR レジスタ

受信動作は送信設定と同じデータ・フォーマットを使用しますが、ストップ・ビットの数は常に1と想定されます。スタート・ビットの検出後、受信されたワードは、SCLK/(16×デバイザ)のボーレートで受信シフト・レジスタ（RSR）にシフト・インされます。ストップ・ビットを含めて、適切なビット数が受信された後、図 13-6に示すように、データとステータスが更新されて、受信シフト・レジスタはUART受信バッファ・レジスタ（UART_RBR）に転送されます。受信されたワードがUART_RBRバッファに転送され、適切な同期遅延の後で、データ・レディ（DR）ステータス・フラグが更新されます。

ボーレートの16倍に等しいサンプリング・クロックによって、データはビットのできるだけ中間点近くでサンプリングされます。内部サンプル・クロックは非同期受信データレートと正しく一致しないこともあるため、サンプリング・ポイントは各ビットの中心からドリフトします。サンプリング・ポイントは、各スタート・ビットと再び同期がとられます。したがって、エラーはシングル・ワードの長さにわたってのみ累計されます。受信フィルタでは、サンプリング・クロック周期の2倍より小さいスプリアス・パルスを除去します。

読出し専用のUART_RBR レジスタは、書込み専用のUART_THR レジスタやUART_DLL レジスタと同じアドレスにマッピングされます。UART_RBRにアクセスするには、UART_LCR のDLAB ビットをクリアする必要があります。DLABビットがクリアされると、このアドレスへの書込みはUART_THR レジスタをターゲットにし、このアドレスからの読出しはUART_RBR レジスタを返します。

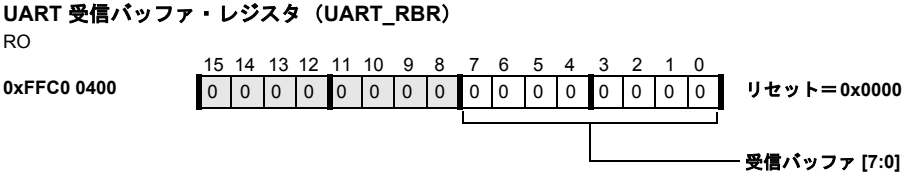


図 13-6. UART 受信バッファ・レジスタ

■ UART_IER レジスタ

UART割込みイネーブル・レジスタ（UART_IER）は、UARTデータ・レジスタの空状態または満杯状態のシステム処理に対する要求をイネーブルにするために使用されます。アクションの手段としてポーリングが使用される場合を除いて、このレジスタのERBFI ビットやETBEI ビットは一般にセットされます。

システムDMAをイネーブルにせずにこのレジスタを設定すると、UARTは割込みを使用して、プロセッサにデータ在庫状態を通知します。このモードで適切な動作を実現するには、システム割込みをイネーブルにし、適切な割込み処理ルーチンを用意しておく必要があります。下位互換性のために、UART_IIR は依然として正しい割込みステータスを反映します。

DMAがイネーブルであるかどうかとは無関係に、UARTは、データ送信、データ受信、ライン・ステータス・イベントを独立して扱うために、3つの独立した割込みチャンネルを備えています。

システムDMAをイネーブルにした状態で、UARTは、プロセッサとの間でデータを転送するためにDMAを使用します。専用のDMAチャンネルは、受信動作や送信動作に使用できます。ライン・エラー処理は、受信／送信の設定とは完全に独立して設定できます。

UART_IER レジスタは、UART_DLH と同じアドレスにマッピングされます。UART_IERにアクセスするには、UART_LCRのDLABビットをクリアする必要があります。

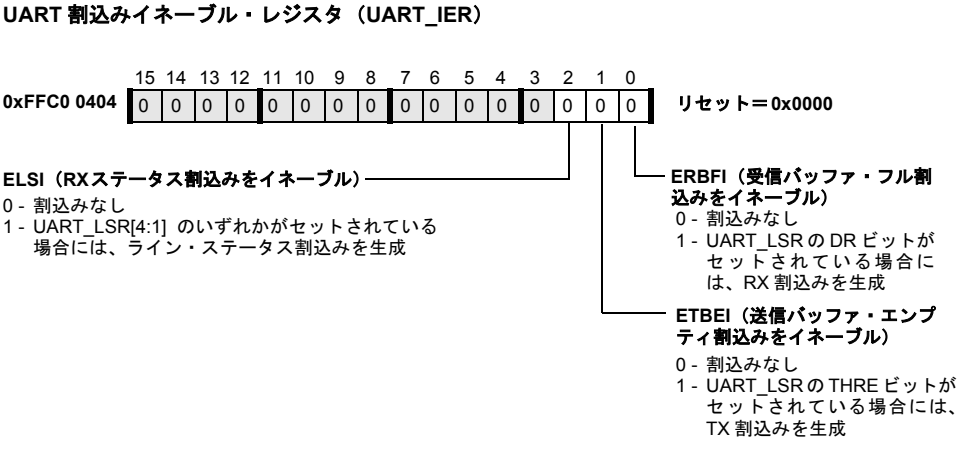


図 13-7. UART 割込みイネーブル・レジスタ

UARTのDMAをイネーブルにするには、まずシステムDMAコントロール・レジスタを設定してから、UART_IER レジスタでUARTのERBFI割込みやETBEI割込みをイネーブルにします。これは、割込み要求ラインがDMA要求ラインの代わりになるためです。これらの要求を受信すると同時に、DMAがイネーブルであるかどうかに応じて、DMA制御ユニットは、ダイレクト・メモリ・アクセスを生成するか、UART割込みをシステム割込み処理ユニットに渡します。しかし、UARTのエラー割込みは、DMAユニットを完全にバイパスして、システム割込み処理ユニットに直接移されます。

UARTライン・ステータス・レジスタ（UART_LSR）の各ビットによって、次のいずれかの条件が発生している場合には、ELSIビットは、独立した割込みチャンネルでの割込み生成をイネーブルにします。

- 受信オーバーラン・エラー（OE）
- 受信パリティ・エラー（PE）
- 受信フレーミング・エラー（FE）
- ブレーク割込み（BI）

UART_IERレジスタでETBEIビットがセットされると、UARTモジュールはすぐに割込みまたはDMA要求を発行します。文字列の伝送を開始するとき、最初の文字に対する特殊な扱いは必要ありません。ETBEIビットをセットすると、割込みサービス・ルーチンは最初の文字をメモリからロードし、それを通常の方法でUART_THRレジスタに書き込みます。したがって、文字列伝送が完了した場合には、ETBEIビットをクリアしてください。

■ UART_IIR レジスタ

旧デバイスとの互換性のために、UART割込み識別レジスタ（UART_IIR）は依然としてUART割込みステータスを反映します。旧来の動作では、すべてのUART割込みソースを1つの割込みチャンネルにバンドルして、これらすべてを同じソフトウェア・ルーチンで処理することが必要な場合もあります。このためには、システム割込みコントローラ（SIC）を使用して、すべてのUART割込みを同じ割込み優先順位にグローバルに割り当てます。

保留中の割込みビット（NINT）はクリアされると、割込みが保留にされていることを通知します。STATUSフィールドでは、優先順位が最も高い保留中の割込みを示します。受信ライン・ステータスは最も高い優先順位を持っています。UART_THRエンプティ割込みは、最も低い優先順位を持っています。両方の割込みが通知されている場合には、UART_IIRは0x06を示します。

UART割込みが保留にされた場合、割込みサービス・ルーチン（ISR）は割込みラッチを明示的にクリアする必要があります。次の図では、3つのラッチをクリアする方法を示します。

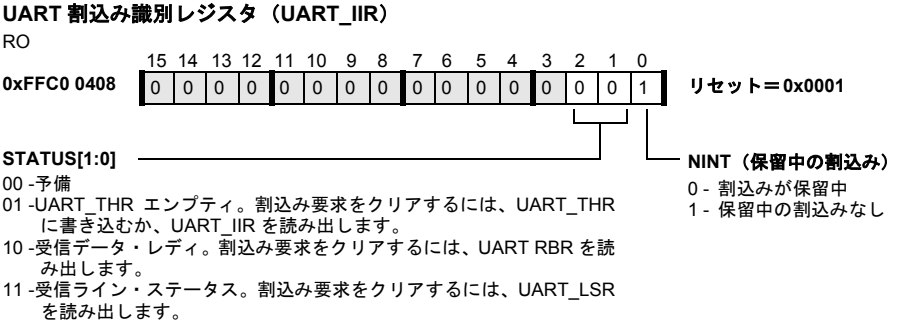


図 13-8. UART 割込み識別レジスタ

TX割込み要求をクリアするには、UART_THRレジスタに新しいデータを書き込むか、UART_IIRレジスタを読み出します。サービス・ルーチンがそれ以上のデータ送信を望まない場合には、UART_IIRレジスタ読出しの特殊な役割に注目してください。

ソフトウェアが伝送を停止させた場合には、UART_IIRレジスタを読み出して割込み要求をリセットする必要があります。UART_IIRレジスタが0x04または0x06を示す（優先順位の高い別の割込みが保留中であることを示す）限り、UART_IIRを読み出してもUART_THRエンプティ・ラッチをクリアできません。

SICがライン・ステータス割込みまたは受信データ割込みに低い割込み優先順位を割り当てた場合には、デッドロック条件が発生することもあります。これを回避するには、UART_THRエンプティ・イベントに対して、イネーブルにされたUART割込みの最低の優先順位を割り当てます。

これらの読出し動作には破壊的性質があるため、特別な注意が必要です。詳細については、6-75ページの「投機的ロード実行」と6-76ページの「条件付きロード動作」を参照してください。

■ UART_DLL レジスタと UART_DLH レジスタ

ビットレートは、システム・クロック（SCLK）と16ビット・デバイザによって判定されます。デバイザは、UARTデバイザ・ラッチ・ロー・バイト・レジスタ（UART_DLL）とUARTデバイザ・ラッチ・ハイ・バイト・レジスタ（UART_DLH）に分かれています。これらのレジスタは16ビット・デバイザを形成します。ボークロックは16で除算されるため、次のようになります。

ボーレート＝SCLK /（16×デバイザ）

UART_DLL＝UART_DLH＝0の場合、デバイザ＝65,536

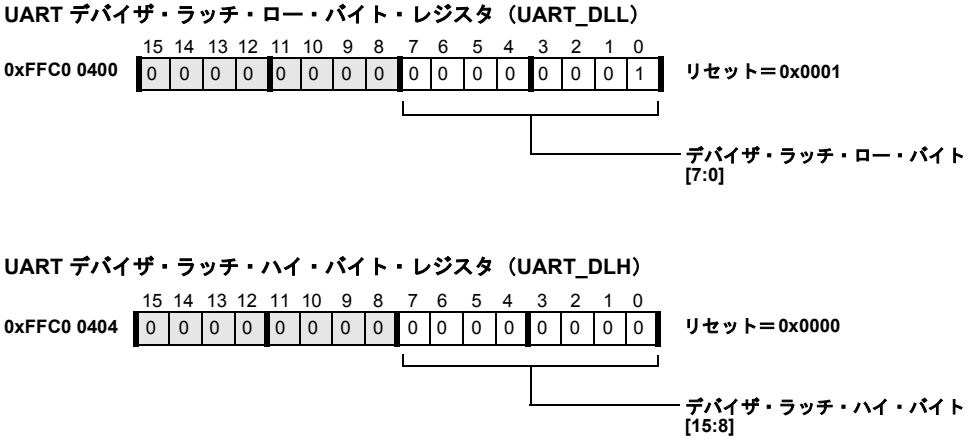


図 13-9. UART デバイザ・ラッチ・レジスタ

UART_DLL レジスタは、UART_THR レジスタやUART_RBR レジスタと同じアドレスにマッピングされます。UART_DLH レジスタは、割込みイネーブル・レジスタ（UART_IER）と同じアドレスにマッピングされます。UARTデバイザ・ラッチ・レジスタにアクセスするには、その前にUART_LCR のDLAB ビットをセットする必要があります。

なお、UART_DLH とUART_DLL によって形成される16ビット・デバイザは0x0001にリセットされるため、デフォルトでは可能な最高のクロック周波数が得られます。UARTが使用されない場合には、UART クロックをディスエーブルにすると節電になります。UART_DLH レジスタとUART_DLL レジスタは、UCEN ビットをセットする前または後にソフトウェアによってプログラムできます。

表 13-1は、多くの標準ボーレートのサポートに必要な除算係数の例を示します。

表 13-1. 100MHz SCLK による UART のボーレート例

ボーレート	DL	実際値	% エラー
2400	2604	2400.15	.006
4800	1302	4800.31	.007
9600	651	9600.61	.006
19200	326	19171.78	.147
38400	163	38343.56	.147
57600	109	57339.45	.452
115200	54	115740.74	.469
921600	7	892857.14	3.119
6250000	1	6250000	–

SCLK周波数の選択には注意が必要です。希望するボーレートの偶数倍では、低エラー率につながることがあります。

■ UART_SCR レジスタ

8 ビットの UART スクラッチ・レジスタ（UART_SCR）の内容は 0x00 にリセットされます。これは、汎用のデータ記憶に使用され、UART ハードウェアを制御することはありません。

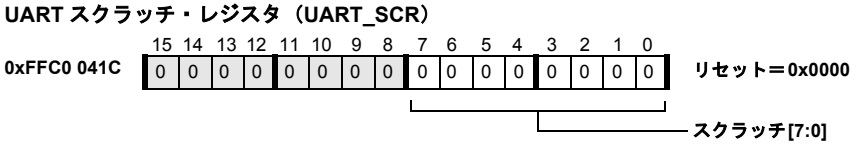


図 13-10.UART スクラッチ・レジスタ

■ UART_GCTL レジスタ

UART グローバル・コントロール・レジスタ（UART_GCTL）には、内部 UART クロック用と、UART の IrDA 動作モード用のイネーブル・ビットがあります。

UCEN ビットは、以前の UART 実装環境には存在しなかったことに注意してください。これは、UART が使用されない場合に、節電するために採用されました。コードを移植する際には、必ずこのビットをイネーブルにしてください。

IrDA TX極性変更ビットとIrDA RX極性変更ビットは、IrDAモードでのみ有効です。2つの強制エラー・ビットFPEとFFEは、テスト目的で用意されています。これらは、特にループバック・モードでのソフトウェア・デバッグに役立ちます。

UART グローバル・コントロール・レジスタ（UART_GCTL）

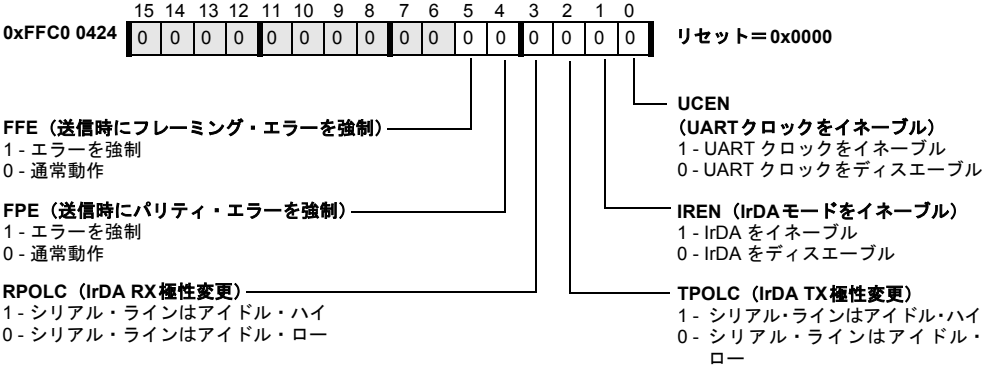


図 13-11. UART グローバル・コントロール・レジスタ

非 DMA モード

非DMAモードでは、UARTはプロセッサ・コアを利用してデータを送／受信します。文字を送信するには、文字をUART_THRにロードします。受信されたデータは、UART_RBRから読み出すことができます。プロセッサは、一度に1文字を読出し／書込みする必要があります。

データの欠落やシリアル・データストリームの位置合わせ不良を防ぐため、UARTライン・ステータス・レジスタ（UART_LSR）は、THREとDRというハンドシェイキング用の2つのステータス・フラグを提供します。

THRE フラグは、UART_THR で新しいデータの準備ができたときにセットされ、プロセッサが UART_THR に新しいデータをロードしたときにクリアされます。空でない UART_THR に書込みを行うと、レジスタは新しい値で上書きされ、以前の文字は送信されません。

DR フラグは、UART_RBR で新しいデータが使用可能であることを通知します。プロセッサが UART_RBR から読み出すと、このフラグは自動的にクリアされます。満杯でない UART_RBR に読出しを行うと、これまでに受信された値が返されます。UART_RBR が正しいタイミングで読み出されない場合には、新しく受信されたデータが UART_RBR を上書きし、オーバーラン（OE）フラグがセットされます。

割込みをディスエーブルにした状態でこれらのステータス・フラグをポーリングすれば、データの転送準備完了を判断できます。なお、ポーリングはプロセッサに負荷をかけるため、一般にリアルタイムの信号処理環境では使用されません。ソフトウェアは UART クロックをイネーブルにする前に、UART_THR レジスタに 2 つまでのワードを書き込むことができます。UCEN ビットがセットされるとすぐに、この 2 つのワードが送信されます。

あるいは、割込みサービス・ルーチン（ISR）によって、UART の書込み／読出しを行うこともできます。UART TX、UART RX、UART エラー用に、別個の割込みラインが提供されています。独立した割込みは、UART_IER レジスタによって個々にイネーブルにできます。

ISR は UART 割込み識別レジスタ（UART_IIR）内のステータス・ビット・フィールドを評価して、信号割込みソースを判断できます。複数のソースが信号を出している場合には、優先順位の最も高いソースがステータス・フィールドに表示されます。割込みは、プロセッサの割込みコントローラによって割り当てられ、マスク解除される必要があります。ISR は、割込みラッチを明示的にクリアする必要があります。13-12 ページの図 13-8 を参照してください。

DMA モード

このモードでは、UARTとメモリとの間のデータ転送は、受信（RX）と送信（TX）の別個のDMAチャンネルによって行われます。ソフトウェアはデータを転送する必要がなく、ディスクリプタ・メカニズムまたは自動バッファ・モードを通じて適切な転送を設定するだけですみます。

UART DMAチャンネルでは新たなバッファリングは提供されないため、遅延条件は非DMAモードの場合と同じです。しかし、この遅延は、プロセッサ負荷や割込み優先順位によってではなく、バス・アクティビティと調停メカニズムによって決定されます。詳細については、第9章「ダイレクト・メモリ・アクセス」を参照してください。

保留中の割込みのラッチされた要求をクリアするには、DMA割込みルーチンは対応するDMA IRQステータス・レジスタに明示的に1を書き込む必要があります。

UARTのDMAをイネーブルにするには、まずシステムDMAコントロール・レジスタを設定してから、UART_IERレジスタでUARTのERBFI割込みやETBEI割込みをイネーブルにします。これは、割込み要求ラインはDMA要求ラインの代わりになるためです。これらの要求の受信と同時に、DMAがイネーブルにされているかどうかに応じて、DMA制御ユニットはダイレクト・メモリ・アクセスを生成するか、UART割込みをシステム割込み処理ユニットに渡します。しかし、UARTのエラー割込みはDMAユニットを完全にバイパスして、システム割込み処理ユニットに直接移されます。

UARTのDMAでは、8ビット動作をサポートします。

ミキシング・モード

非DMAモードとDMAモードでは、異なる同期メカニズムを使用します。したがって、非DMAモードとDMAモードを切り替えるには、その前に、すべてのシリアル通信を完了する必要があります。つまり、非DMA伝送からDMA伝送に切り替える前に、UART_LSRのステータス・ビットTHREとTEMTをテストして、UART_THRと内部送信シフト・レジスタ（TSR）の両方が空であることを確認してください。空でない場合には、プロセッサは、適切なUART送信DMA設定レジスタ（UART_CONFIG_TX）内の2ビットのDMAバッファ・ステータス・フィールドがクリアされるまで待機する必要があります。

DMA動作から非DMA動作に切り替える場合には、受信（RX）と送信（TX）のDMAチャンネルがDMA FIFOに含まれるデータも含めて、それらのデータを完全に転送していることを確認します。DMA RX割込みは、最後のデータ・ワードがメモリに書き込まれた（そしてDMA FIFOを離れた）ことを示します。一方、DMA TX割込みは、最後のデータ・ワードがメモリを離れた（そしてDMA FIFOに入った）ことを示します。プロセッサはDMAチャンネルをディスエーブルにする前に、TXチャンネルのIRQ_STATUSレジスタのDMA_RUNステータス・ビットがクリアされていることをテストして、TX FIFOが空になるまで待機する必要があります。

IrDA サポート

標準のUART機能は別として、UARTは赤外線通信協会（IrDA）の勧告に基づいて、赤外線信号による半二重シリアル・データ通信にも対応します。IrDA SIR（9.6/115.2 Kbpsレート）と呼ばれる物理レイヤは、RZI（return-to-zero-inverted）変調をベースにしています。パルス位置変調は提供していません。

RZI変調は16xのデータレート・クロックを使用し、通常はUARTによって送信されるNRZ（non-return-to-zero）コードを反転および変調して実現されます。受信側では、16xクロックを使用して、IrDAのパルス・サンプル・ウィンドウを決定します。このウィンドウから、RZI変調されたNRZコードが再生されます。

IrDAのサポートをイネーブルにするには、UARTグローバル・コントロール・レジスタのIRENビットをセットします。IrDAアプリケーションは、外付けトランシーバを必要とします。

■ IrDA トランスミッタの説明

UARTによって送信されるIrDAパルスを生成するには、まずトランスミッタの通常のNRZ出力が反転されます。これにより、0は16 UARTクロック周期のハイ・パルスとして送信され、1は16 UARTクロック周期のロー・パルスとして送信されます。その後、パルスの前縁は6 UARTクロック周期だけ遅延します。同様に、パルスの後縁は8 UARTクロック周期だけ切り捨てられます。このため、元の0の最終表現は、16サイクルUARTクロック周期内のわずか3/16クロック周期のハイ・パルスとなります。図 13-12に示すように、このパルスはビット時間の中央を中心にしています。最終のIrDAパルスは、オフチップ赤外線ドライバに供給されます。

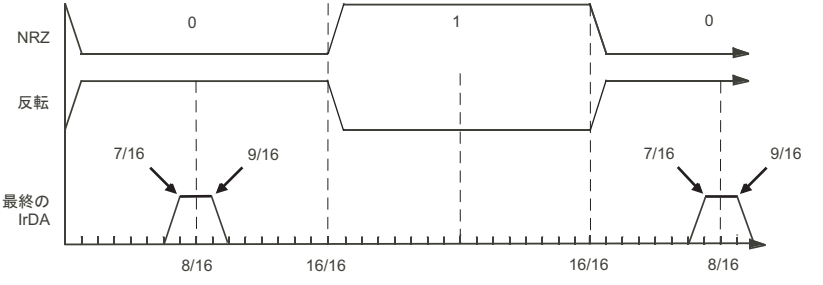


図 13-12.IrDA の送信パルス

この変調方式は、16のUARTクロック・サイクルのうち3サイクルのハイレベルUARTからパルス幅出力が保証されます。13-14ページの表13-1に示すように、ボーレート・ジェネレータに付随する誤差項は非常に小さく、多くの赤外線トランシーバ仕様の許容誤差内に十分収まります。

■ IrDA レシーバの説明

IrDAのレシーバ機能は、送信機能よりも複雑です。レシーバは、IrDAパルスを識別してノイズを除去しなければなりません。そのためにレシーバは、予想されるパルスの中央を中心とする狭いウィンドウでIrDAパルスを探します。

グリッチ・フィルタ処理を実現するには、最初のパルスが現れてから16のシステム・クロックをカウントします。カウンタが切れたときにパルスが存在しない場合には、それはグリッチと見なされます。そうでない場合、それは0と解釈されます。このように判断する理由は、オンチップ容量クロスカップリングに起因するグリッチは、一般にシステム・クロック周期の何分の1も持続しないからです。チップの外部にありトランスミッタの一部ではないソースは、適切な遮蔽によって回避できます。この他の唯一のグリッチのソースは、トランスミッタ自身です。プロセッサが仕様を満たすかどうかは、トランスミッタに依存します。トランスミッタが仕様を外れる場合には、予測できない結果が発生することがあります。4ビット・カウンタは、最小のコストで保護レベルを向上させます。なお、システム・クロックはシステムによって変化することがあるため、許容される最長のグリッチは、システム・クロック周波数に逆比例します。

受信サンプリング・ウィンドウは、16xビットタイム・サンプル・クロックでクロック駆動されるカウンタによって決まります。サンプリング・ウィンドウは、スタート・ビットを中心にしてサンプリング・ウィンドウを置くことによって、各スタート・ビットに再同期されます。

受信データの極性は、IRPOLビットを使用して選択できます。図 13-13は、各極性タイプの例を示します。

- IRPOL ＝ 0 では、受信データ入力はアイドル 0 であり、各アクティブ 1 遷移は UART NRZ 値 0 に対応すると想定します。
- IRPOL ＝ 1 では、受信データ入力はアイドル 1 であり、各アクティブ 0 遷移は UART NRZ 値 0 に対応すると想定します。

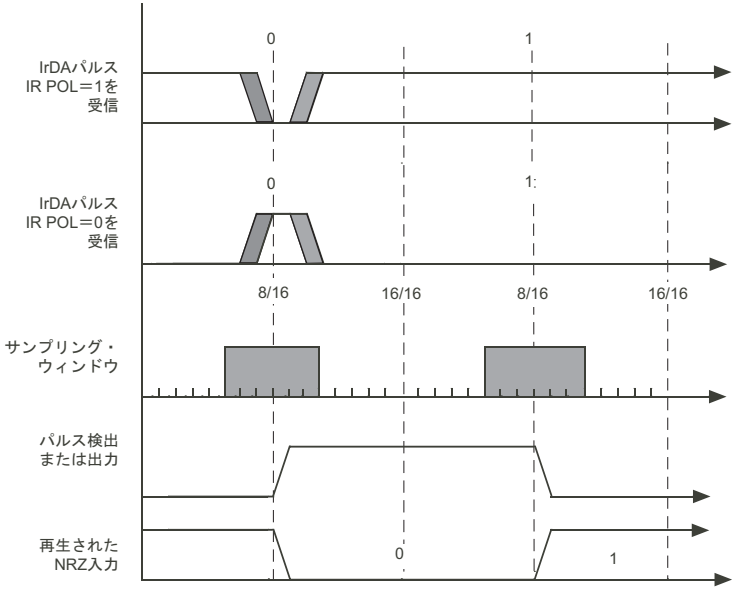


図 13-13.IrDA レシーバのパルス検出

第14章　プログラマブル・フラグ

このプロセッサは、16本の双方向プログラマブル・フラグ（PFx）、つまり汎用I/OピンPF[15:0]を提供します。各ピンは、フラグ方向レジスタ（FIO_DIR）を使用して、入力または出力として個々に設定できます。出力として設定されると、フラグ・データ・レジスタ（FIO_FLAG_D）に直接書き込むことで、すべてのPFxピンの状態を指定できます。出力として設定されると、フラグ・セット（FIO_FLAG_S）、フラグ・クリア（FIO_FLAG_C）、フラグ・トグル（FIO_FLAG_T）の各レジスタに書き込まれた状態によって、出力PFxピンによって駆動される状態が決まります。ピンの入出力設定とは無関係に、レジスタ（FIO_FLAG_D、FIO_FLAG_S、FIO_FLAG_C、FIO_FLAG_T）の読出しによって各ピンの状態が返されます。

各PFxピンは、割込みを生成するように設定できます。PFxピンが入力として設定されると、ピンの状態（ハイまたはロー）、エッジ遷移（ローからハイまたはハイからロー）、または両方のエッジ遷移（ローからハイとハイからロー）に基づいて、割込みを生成できます。入力センシティビティは、フラグ極性レジスタ（FIO_POLAR）、フラグ割込みセンシティビティ・レジスタ（FIO_EDGE）、フラグ・セット・オン・ボス・エッジ・レジスタ（FIO_BOTH）によってビット単位で定義されます。入力極性は、フラグ極性レジスタによってビット単位で定義されます。PFx入力がイネーブルにされ、PFxピンが出力として設定されると、ピンの割込みをイネーブルにすれば、PFxピンの設定によって割込みを生成できます。

プロセッサは、PFxピンに2つの独立した割込みチャンネルを提供します。これらは、割込みAおよび割込みBと呼ばれますが、機能は同じです。各割込みチャンネルには、フラグ割込みマスク・データ・レジスタ（FIO_MASKx_D）、フラグ割込みマスク・セット・レジスタ（FIO_MASKx_S）、

フラグ割込みマスク・クリア・レジスタ（FIO_MASKx_C）、フラグ割込みマスク・トグル・レジスタ（FIO_MASKx_T）という、4本のマスク・レジスタが関係付けられています。

各PFxピンは、これら8本のレジスタのそれぞれのビットによって表されます。マスク・セット・レジスタのビットに1を書き込むと、そのPFxピンに対する割込み生成がイネーブルにされます。一方、マスク・クリア・レジスタのビットに1を書き込むと、そのPFxピンに対する割込み生成がディスエーブルにされます。

割込みマスクをトグルするには、マスク・トグル・レジスタのビットに1を書き込みます。さらに、マスク・ビットを直接設定するには、マスク・データ・レジスタに書き込みます。この柔軟なメカニズムによって、各ビットはフラグ割込みAを生成したり、フラグ割込みBを生成したり、フラグ割込みAとBの両方を生成したり、いずれも生成しないことができます。

PFピンがシステムで使用されない場合、入力バッファをディスエーブルにして、未使用ピンでは外部プルアップもプルダウンも不要にできます。デフォルトでは、入力バッファはディスエーブルにされます。入力バッファをイネーブルにするには、フラグ入力イネーブル・レジスタ（FIO_INEN）のビットを使用します。

PFxピンは、パラレル・ペリフェラル・インターフェース（PPI）、タイマ、シリアル・ペリフェラル・インターフェース（SPI）によって多重使用されます。表 14-1は、プログラマブル・フラグ・ピンとその多重化された機能を示します。

表 14-1. プログラマブル・フラグ・ピンと機能

PF ピン	PF ピンを共有するペリフェラル		
	PPI	SPI	タイマ 0、1、2
0		スレーブ・セレクト入力 ($\overline{\text{SPISS}}$)	
1		スレーブ・セレクト・イネーブル 1 (SPISEL1)	入力クロック
2		スレーブ・セレクト・イネーブル 2 (SPISEL2)	
3	フレーム同期 3	スレーブ・セレクト・イネーブル 3 (SPISEL3)	
4	I/O #15	スレーブ・セレクト・イネーブル 4 (SPISEL4)	
5	I/O #14	スレーブ・セレクト・イネーブル 5 (SPISEL5)	
6	I/O #13	スレーブ・セレクト・イネーブル 6 (SPISEL6)	
7	I/O #12	スレーブ・セレクト・イネーブル 7 (SPISEL7)	
8	I/O #11		
9	I/O #10		
10	I/O #9		
11	I/O #8		
12	I/O #7		
13	I/O #6		
14	I/O #5		
15	I/O #4		

表 14-2は、PFピンを共有するペリフェラル機能の使い方について説明します。

表 14-2. PF ピンを共有するペリフェラル機能の使い方

PF ピン	PF ピンを共有するペリフェラル機能の使い方 （ペリフェラルはイネーブルにされていると想定）		
	PPI	SPI	タイマ 0、1、2
0		SPI_CTL の PSSE に「1」を書き込む	
1		SPI_FLG の FLS1 に「1」を書き込む	TIMERx_CONFIG の CLK_SEL に「1」を書き込む
2		SPI_FLG の FLS2 に「1」を書き込む	
3	PPI_CTLのPORT_CFGに「01」を書き込む（PORT_DIR = ‘1’の場合）か、PORT_CFG に「10」を書き込む（PORT_DIR = ‘0’の場合）	SPI_FLG の FLS3 に「1」を書き込む	
4	PPI_CTL の DLEN に「111」を書き込む	SPI_FLG の FLS4 に「1」を書き込む	
5	PPI_CTL の DLEN に「110」を書き込む	SPI_FLG の FLS5 に「1」を書き込む	
6	PPI_CTL の DLEN に「101」を書き込む	SPI_FLG の FLS6 に「1」を書き込む	
7	PPI_CTL の DLEN に「100」を書き込む	SPI_FLG の FLS7 に「1」を書き込む	
8	PPI_CTL の DLEN に「011」を書き込む		
9	PPI_CTL の DLEN に「010」を書き込む		
10	PPI_CTL の DLEN に「001」を書き込む		

表 14-2. PF ピンを共有するペリフェラル機能の使い方

PF ピン	PF ピンを共有するペリフェラル機能の使い方 （ペリフェラルはイネーブルにされていると想定）		
	PPI	SPI	タイマ 0、1、2
11	PPI_CTL の DLEN に「001」を書き込む		
12	PPI がイネーブルの場合、常にイネーブル		
13	PPI がイネーブルの場合、常にイネーブル		
14	PPI がイネーブルの場合、常にイネーブル		
15	PPI がイネーブルの場合、常にイネーブル		

詳細については、第11章「パラレル・ペリフェラル・インターフェース」、第10章「SPI互換ポート・コントローラ」、第15章「タイマ」を参照してください。

プログラマブル・フラグ・レジスタ（MMR）

プログラマブル・フラグ・レジスタは、システムのメモリマップド・レジスタ（MMR）の一部です。プログラマブル・フラグMMRのアドレスを付録Bに示します。フラグ設定レジスタへのコア・アクセスには、システム・バスを利用します。

■ FIO_DIR レジスタ

フラグ方向レジスタ（FIO_DIR）は読出し／書込みレジスタです。各ビットの位置は、PFx ピンに対応します。ロジック 1 では、PFx ピンは出力として設定され、FIO_FLAG_D レジスタに格納された状態が駆動されます。ロ

ジック0では、PFxピンは入力として設定されます。このレジスタのリセット値は0x0000であり、リセットと同時に、すべてのPFピンは入力になります。

なお、PFxピンを入力として使用する場合、フラグ入力イネーブル・レジスタで対応するビットもセットしてください。

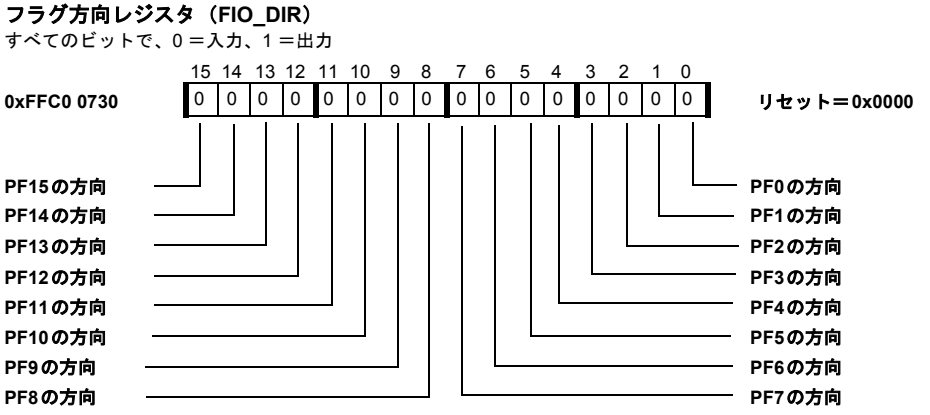


図 14-1. フラグ方向レジスタ

■ フラグ値レジスタの概要

プロセッサには、4本のフラグ値レジスタがあります。

- フラグ・データ・レジスタ（FIO_FLAG_D）
- フラグ・セット・レジスタ（FIO_FLAG_S）
- フラグ・クリア・レジスタ（FIO_FLAG_C）
- フラグ・トグル・ダイレクト・レジスタ(FIO_FLAG_T)

これらのレジスタは、以下の目的で使用されます。

- 入力として定義されたPFxピンの値を感知
- 出力として定義されたPFxピンの状態を指定
- PFxピンによって生成された割込みをクリア

各PFxピンは、4本のレジスタのそれぞれのビットによって表されます。

フラグ・データ、フラグ・セット、フラグ・クリア、またはフラグ・トグルのいずれかのレジスタを読み出すと、PFxピンの値が返されます。返された値は、各ピンの極性とセンシティビティの設定に基づいて、出力として定義されたPFxピンの状態と、入力として定義されたPFxピンのセンスを示します。

リセット後のフラグ・データ、フラグ・セット、フラグ・クリア、またはフラグ・トグルの各レジスタを読み出すと、0x0000が得られます。これらのピンは入力ですが、入力バッファはイネーブルにされていないためです。FIO_POLAR、FIO_EDGE、FIO_BOTHレジスタの設定に基づいて、これらのレジスタから読み出した値を解釈する方法については、表 14-3のガイダンスを参照してください。

エッジ・センシティブとして設定されたピンの場合、これらのレジスタの1つからの値“1”の読出しはスティッキーです。つまり、いったんセットされると、ユーザ・コードによってクリアされるまでセットされたままです。レベル・センシティブ・ピンの場合、ピン状態はすべてのサイクルでチェックされるため、ピンでの元のレベルが変化すると読出し値も変化します。

表 14-3.　フラグ値レジスタのピン解釈

FIO_POLAR	FIO_EDGE	FIO_BOTH	MMR 設定の効果
0	0	X	ハイのピンは 1 として読み出され、ローのピンは 0 として読み出されます。
0	1	0	立上がりエッジが発生した場合、ピンは 1 として読み出され、そうでない場合、ピンは 0 として読み出されます。
1	0	X	ローのピンは 1 として読み出され、ハイのピンは 0 として読み出されます。
1	1	0	立下がりエッジが発生した場合、ピンは 1 として読み出され、そうでない場合、ピンは 0 として読み出されます。
X	1	1	何らかのエッジが発生した場合、ピンは 1 として読み出され、そうでない場合、ピンは 0 として読み出されます。

フラグ・セット、フラグ・クリア、フラグ・トグルの各レジスタの詳細については、14-9ページの「FIO_FLAG_S、FIO_FLAG_C、およびFIO_FLAG_Tレジスタ」を参照してください。

■ FIO_FLAG_D レジスタ

図 14-2に示すフラグ・データ・レジスタ（FIO_FLAG_D）が書き込まれると、すべてのPFxピンの状態を直接指定します。このレジスタが読み出されると、PFx ピンの値を返します。

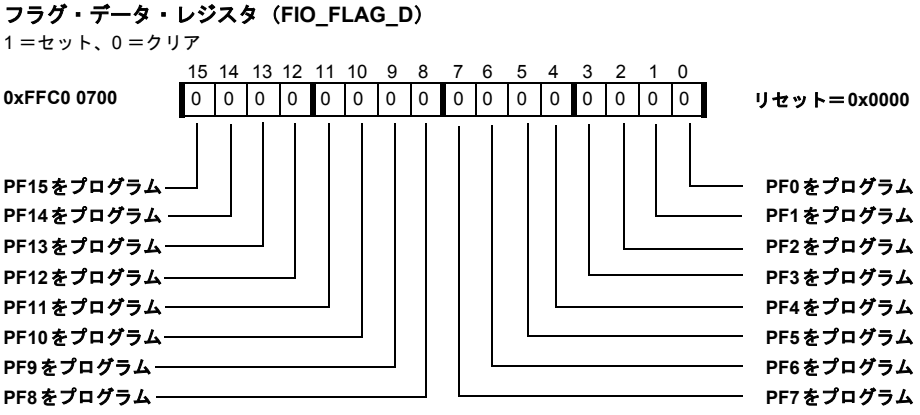


図 14-2. フラグ・データ・レジスタ

■ FIO_FLAG_S、FIO_FLAG_C、および FIO_FLAG_T レジスタ

フラグ・セット・レジスタ（FIO_FLAG_S）、フラグ・クリア・レジスタ（FIO_FLAG_C）、フラグ・トグル・レジスタ（FIO_FLAG_T）は、以下の目的で使用されます。

- 各出力PFxピンに関係付けられた出力状態をセット、クリア、またはトグルします。
- 各入力PFxピンから取り込まれた、ラッチされた割込み状態をクリアします。

このメカニズムは、これまで行われてきた読出し—修正—書込みメカニズムでの潜在的な問題を回避するために使用されます。これらのレジスタを読み出すと、フラグ・ピン状態が示されます。

図 14-3と図 14-4には、それぞれフラグ・セット・レジスタとフラグ・クリア・レジスタを示します。図 14-5にはフラグ・トグル・レジスタを示します。

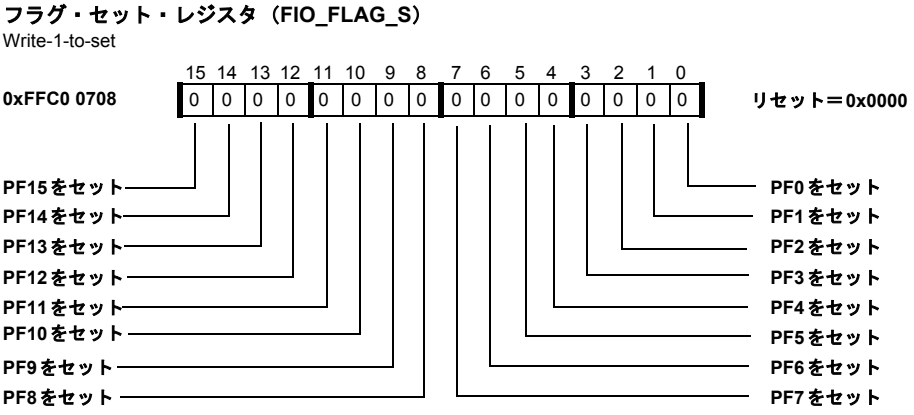


図 14-3. フラグ・セット・レジスタ

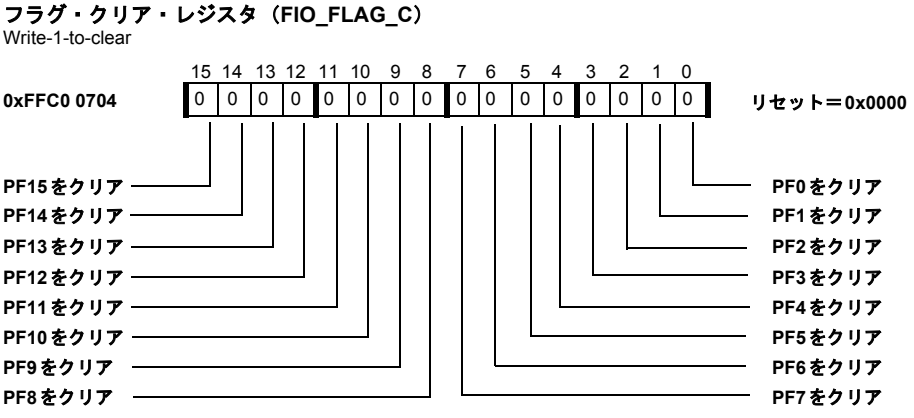


図 14-4. フラグ・クリア・レジスタ

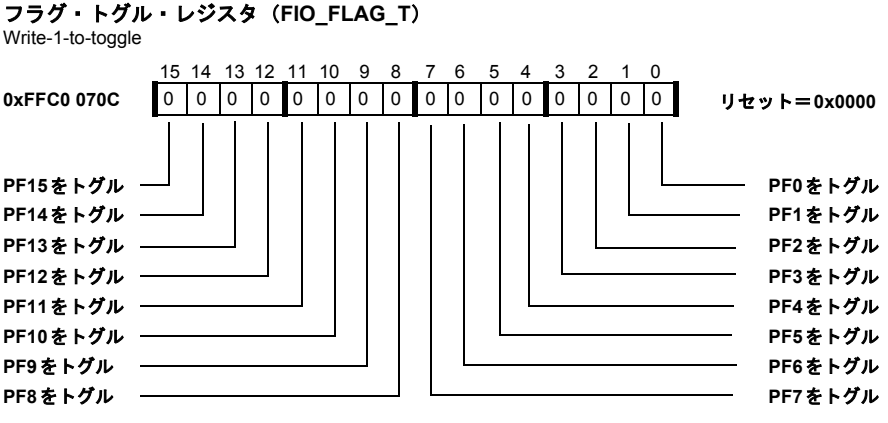


図 14-5. フラグ・トグル・レジスタ

これらのレジスタの動作を示す一例として、PF[0]は出力として設定されていると想定します。フラグ・セット・レジスタに0x0001を書き込むと、他のPFxピンの状態に影響を与えることなく、PF[0]ピンでロジック1が駆動されます。フラグ・クリア・レジスタに0x0001を書き込むと、他のPFxピンの状態に影響を与えることなく、PF[0]ピンでロジック0が駆動され

ます。フラグ・トグル・レジスタに0x0001を書き込むと、PF[0]でのピン状態は、既存のピン状態に応じて、ロジック0からロジック1、またはロジック1からロジック0に変化します。

フラグ・セット、フラグ・クリア、またはフラグ・トグルの各レジスタに0を書き込んでも、フラグ・ピンの値には影響がありません。したがって、無視されます。

フラグ・セット・レジスタまたはフラグ・クリア・レジスタを読み出すと、以下の値が示されます。

- 出力として定義されローに駆動されたPFxピンでは0
- 出力として定義されハイに駆動された（上の例のPF[0]を含む）ピンでは1
- 入力として定義されたPFxピンの現在のセンス

入力センスは、FIO_POLARとFIO_EDGEの設定だけではなく、各ピンでのロジック・レベルにも基づいています。

■ FIO_MASKA_D、FIO_MASKA_C、FIO_MASKA_S、FIO_MASKA_T、FIO_MASKB_D、FIO_MASKB_C、FIO_MASKB_S、FIO_MASKB_T レジスタ

フラグ・マスク割込みレジスタ（FIO_MASKA_D、FIO_MASKA_C、FIO_MASKA_S、FIO_MASKA_T、FIO_MASKB_D、FIO_MASKB_C、FIO_MASKB_S、FIO_MASKB_T）は、write-1-to-set、write-1-to-clear、write-1-to-toggleの各レジスタの相補ペアとして実装されています。この実装によって、読出し―修正―書込みアクセス（つまり、データ・レジスタによるマスク値の直接指定）を必要とすることなく、PFxピンのプロセッサ割込み機能をイネーブル／ディスエーブルにできます。

次に示す4本の専用レジスタでは、フラグ割込みAとフラグ割込みBの両方がサポートされます。

- フラグ・マスク割込みデータ・レジスタ
- フラグ・マスク割込みセット・レジスタ
- フラグ・マスク割込みクリア・レジスタ
- フラグ割込みトグル・レジスタ

フラグ割込みAをサポートするレジスタの図については、14-16ページの「FIO_MASKA_D、FIO_MASKA_C、FIO_MASKA_S、FIO_MASKA_Tレジスタ」を参照してください。

フラグ割込みBをサポートするレジスタの図については、14-18ページの「FIO_MASKB_D、FIO_MASKB_C、FIO_MASKB_S、FIO_MASKB_Tレジスタ」を参照してください。

各PFxピンは、8本のレジスタのそれぞれのビットによって表されます。表 14-4は、マスク・セット、マスク・クリア、マスク・トグルの各レジスタのビットに1を書き込むことの効果を示します。

表 14-4. ビットに 1 を書き込むことの効果

レジスタ	レジスタのビットに 1 を書き込むことの効果
マスク・セット	その PFx ピンに対する割込み生成をイネーブル
マスク・クリア	その PFx ピンに対する割込み生成をディスエーブル
マスク・トグル	割込み生成機能の状態を変更

[A&B]のマスク・データ、セット、クリア、またはトグルの各レジスタを読み出すと、現在のマスク[A&B]データの値が返されます。

割込みAと割込みBは独立して動作します。たとえば、フラグ・マスク割込みAセット・レジスタのビットに1を書き込んでも、フラグ割込みBには影響を与えません。この機能によって、PFxピンはフラグ割込みAを生成したり、フラグ割込みBを生成したり、フラグ割込みAとBの両方を生成したり、いずれも生成しないことが可能になります。

なお、フラグ割込みは、その割込みに対するマスク解除されたすべてのPFピンの論理和によって生成されます。たとえば、フラグ割込みAに対してPF[0]とPF[1]の両方がマスク解除されている場合、PF[0]またはPF[1]によってトリガされると、フラグ割込みAが生成されます。

立上がりエッジまたは立下がりエッジでトリガされる割込みを使用する場合、対応する割込みが処理されるたびに、適切なFIO_FLAG_Cビットに1を書き込むことによって割込み条件をクリアする必要があります。

リセット時に、すべての割込みはマスクされます。

▶ フラグ割込みの生成フロー

図 14-6は、フラグ割込みAまたはフラグ割込みBのイベントを生成できるプロセスを示します。なお、このフローは、ただ1つのプログラマブル・フラグ「FlagN」に対するものです。しかし、フラグ割込みは、その割込みに対してマスク解除されたすべてのPFxピンの論理和によって生成されます。たとえば、フラグ割込みAに対してPF[0]とPF[1]だけがマスク解除されている場合、この割込みはPF[0]またはPF[1]によってトリガされたときに生成されます。

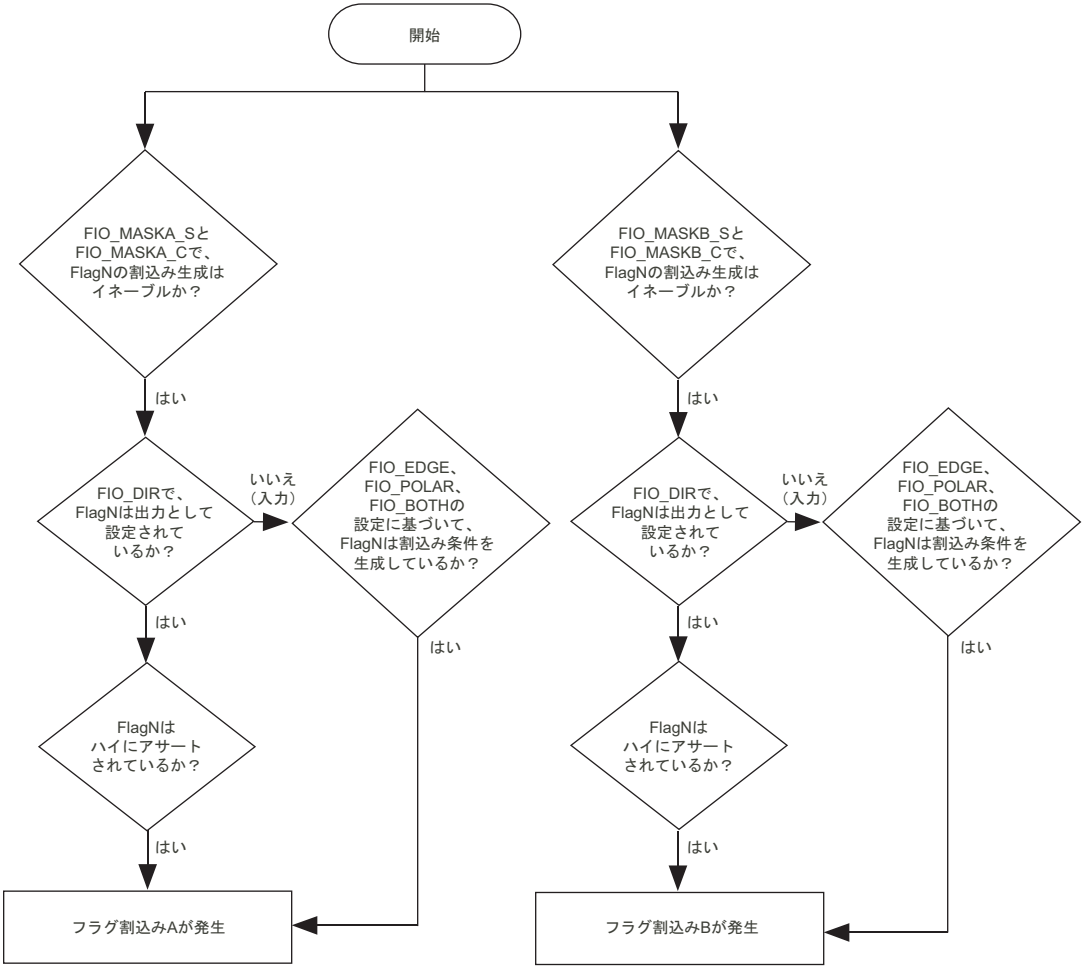


図 14-6. フラグ割込みの生成フロー

▶ FIO_MASKA_D、FIO_MASKA_C、FIO_MASKA_S、FIO_MASKA_T レジスタ

次の4本のレジスタはフラグ割込みAに対応します。詳細については、14-12ページを参照してください。

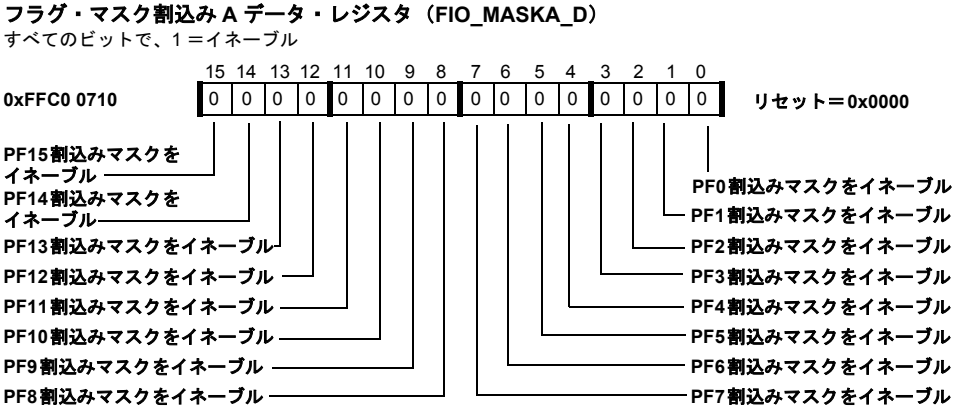


図 14-7. フラグ・マスク割込み A データ・レジスタ

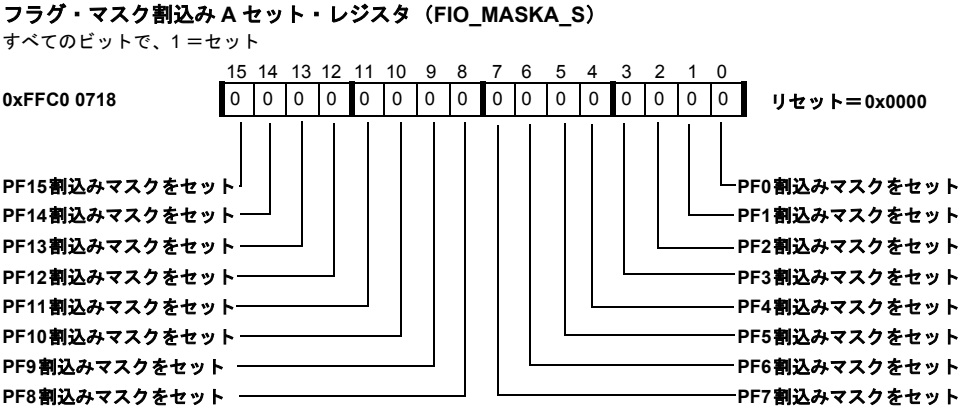


図 14-8. フラグ・マスク割込み A セット・レジスタ

フラグ・マスク割込み A クリア・レジスタ（FIO_MASKA_C）
すべてのビットで、1＝クリア

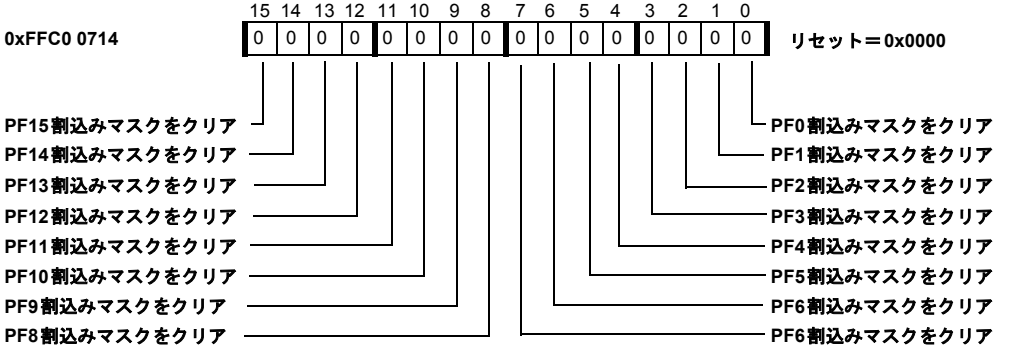


図 14-9. フラグ・マスク割込み A クリア・レジスタ

フラグ・マスク割込み A トグル・レジスタ（FIO_MASKA_T）
すべてのビットで、1＝トグル

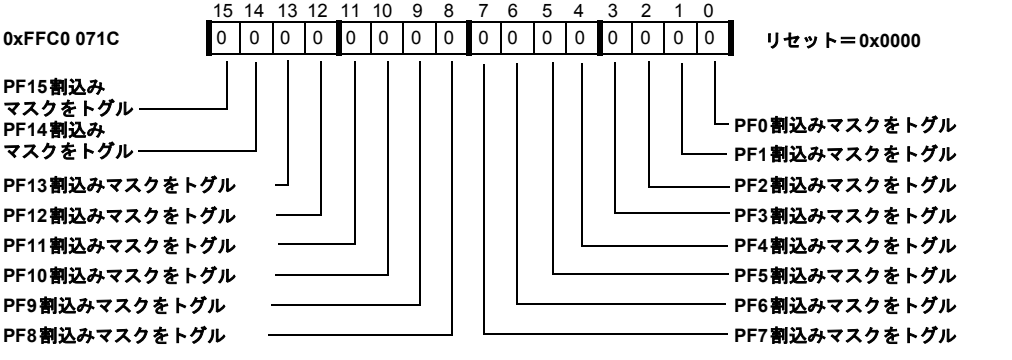


図 14-10.フラグ・マスク割込み A トグル・レジスタ

▶ FIO_MASKB_D、FIO_MASKB_C、FIO_MASKB_S、FIO_MASKB_T レジスタ

次の4本のレジスタはフラグ割込みBに対応します。詳細については、14-12ページを参照してください。

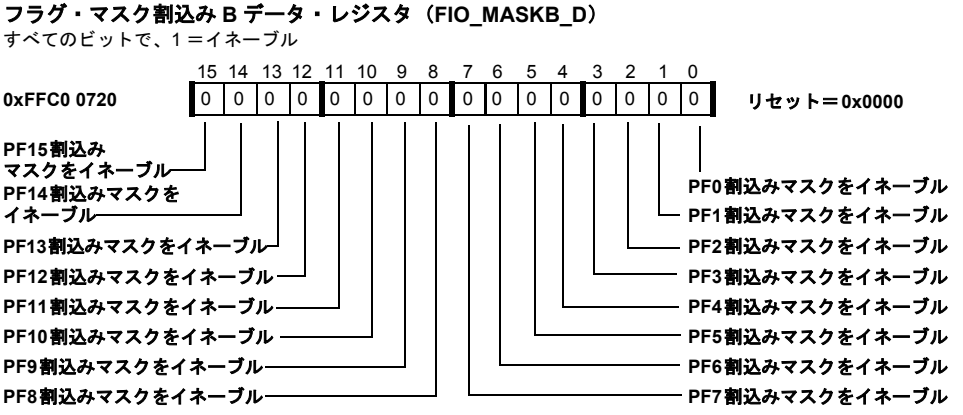


図 14-11. フラグ・マスク割込み B データ・レジスタ

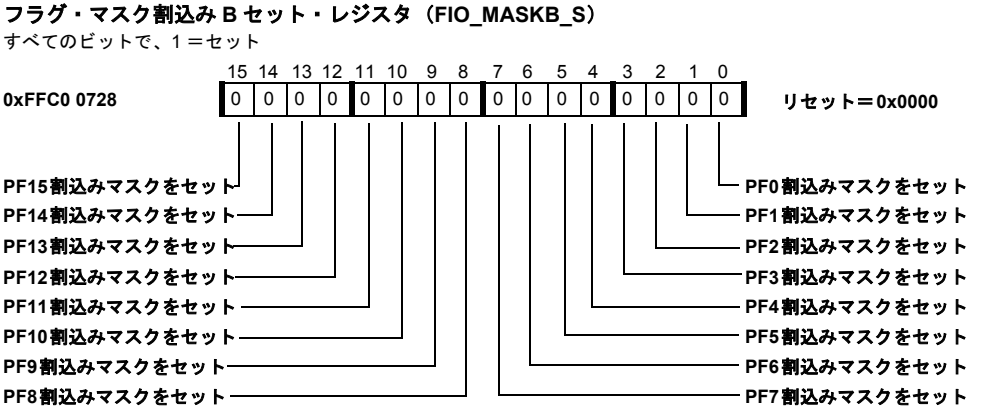


図 14-12. フラグ・マスク割込み B セット・レジスタ

フラグ・マスク割込み B クリア・レジスタ（FIO_MASKB_C）
すべてのビットで、1＝クリア

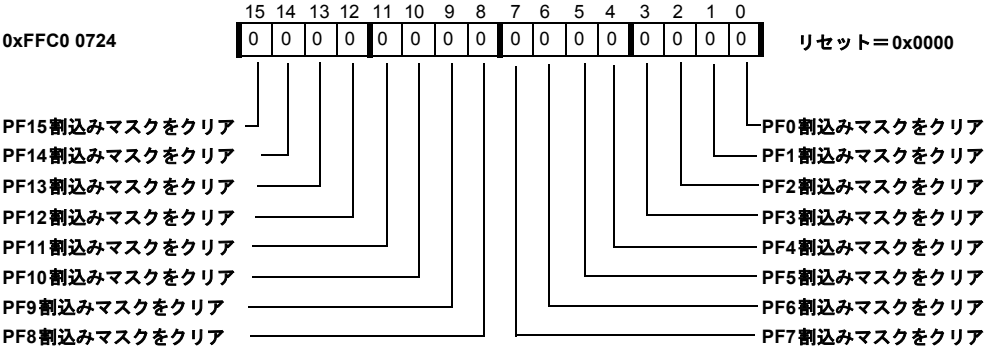


図 14-13.フラグ・マスク割込み B クリア・レジスタ

フラグ・マスク割込み B トグル・レジスタ（FIO_MASKB_T）
すべてのビットで、1＝トグル

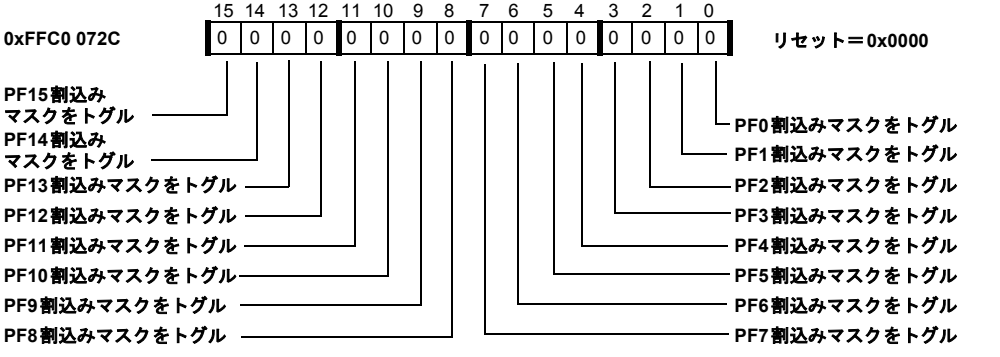


図 14-14.フラグ・マスク割込み B トグル・レジスタ

■ FIO_POLAR レジスタ

フラグ極性レジスタ（FIO_POLAR）は、フラグ入力ソースの極性を設定するために使用されます。アクティブ・ハイまたは立上がりエッジを選択するには、このレジスタのビットに0を設定します。アクティブ・ローまたは立下がりエッジを選択するには、このレジスタのビットに1を設定します。

このレジスタは、出力として定義されているPFxピンには影響がありません。このレジスタの内容はリセット時にクリアされ、デフォルトでアクティブ・ハイ極性になります。

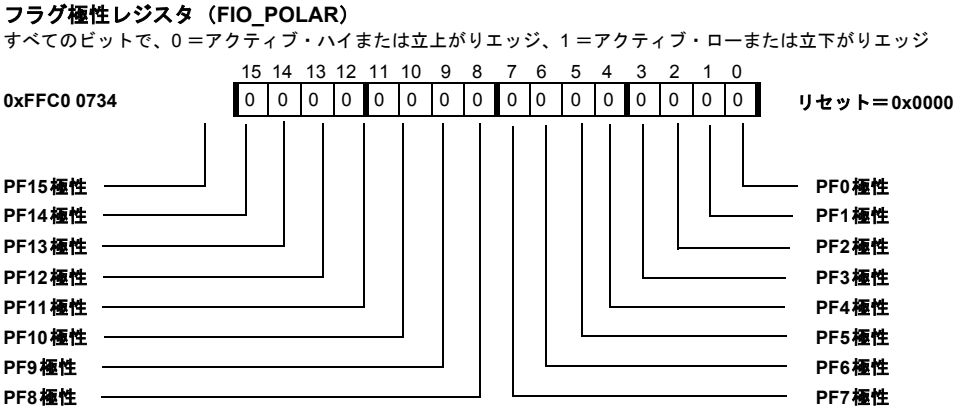


図 14-15.フラグ極性レジスタ

■ FIO_EDGE レジスタ

フラグ割込みセンシティビティ・レジスタ（FIO_EDGE）は、各フラグをレベル・センシティブまたはエッジ・センシティブのソースとして設定するために使用されます。エッジ・センシティブ・モードを使用する場合、システム・クロック・レートによって短いイベントが捕捉されない状況を防ぐためにエッジ検出回路が使用されます。このレジスタは、出力として定義されているPFxピンには影響を与えません。

このレジスタの内容はリセット時にクリアされ、デフォルトでレベル・センシティブになります。

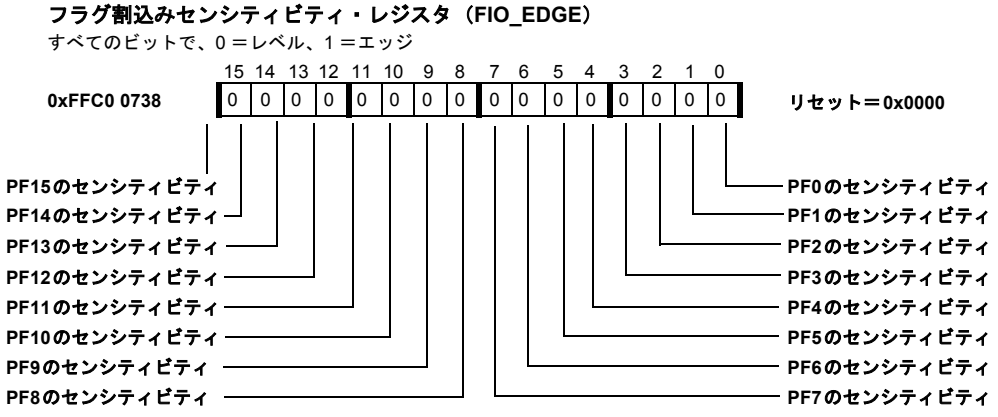


図 14-16.フラグ割込みセンシティビティ・レジスタ

■ FIO_BOTH レジスタ

フラグ・セット・オン・ボス・エッジ・レジスタ（FIO_BOTH）は、立上がりエッジと立下がりエッジの両方で、割込み生成をイネーブルにするために使用されます。

フラグ割込みセンシティビティ・レジスタで特定のPFxピンがエッジ・センシティブに設定された場合、フラグ・セット・オン・ボス・エッジ・レジスタのPFxピンのビットを「両エッジ」に設定すると、立上がりエッジと立下がりエッジの両方で割込みが生成されるようになります。このレジスタは、レベル・センシティブまたは出力として定義されているPFxピンには影響を与えません。

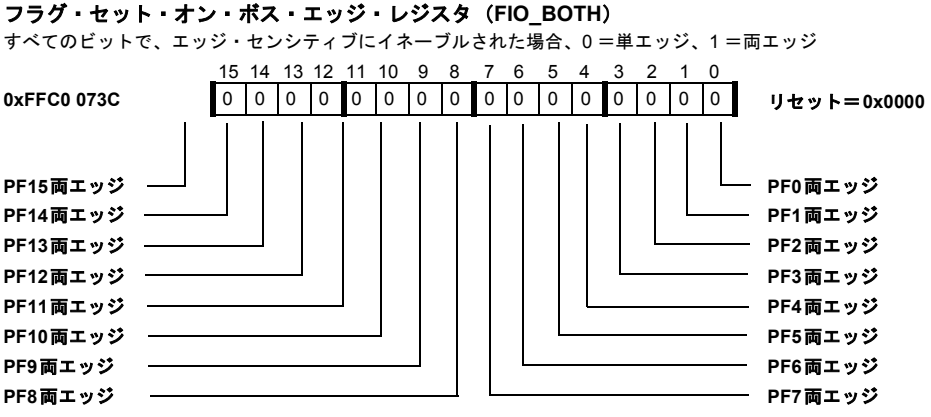


図 14-17.フラグ・セット・オン・ボス・エッジ・レジスタ

■ FIO_INEN レジスタ

フラグ入力イネーブル・レジスタ（FIO_EDGE）は、入力として使用されているフラグ・ピンでの入力バッファをイネーブルにするために使用されます。特定のPFxピンがシステムで使用されていないとき、入力バッファをディスエーブルにしておくと、プルアップやプルダウンの必要がなくなります。デフォルトでは、入力バッファはディスエーブルにされます。

なお、PFxピンが入力として使用されている場合には、フラグ入力イネーブル・レジスタの対応するビットをセットしておく必要があります。そうでない場合、フラグ・ピンを変更してもプロセッサによって認識されません。

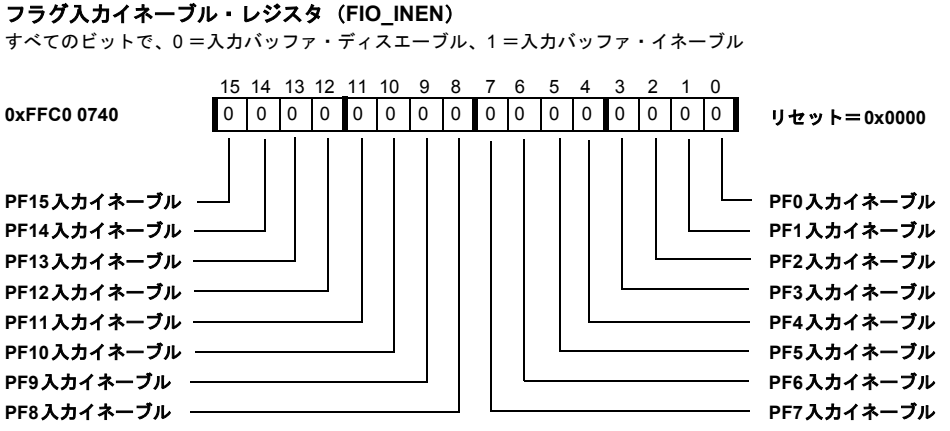


図 14-18.フラグ入力イネーブル・レジスタ

性能／スループット

PFxピンは、システム・クロック（SCLK）に同期します。出力として設定された場合、プログラマブル・フラグはシステム・クロック・サイクルごとに1回遷移できます。

入力として設定された場合、全体的なシステム設計では、コアとシステム・クロックとの間の潜在的な遅延を考慮してください。PFxピンの状態を変更すると、3 SCLKサイクルの遅延が経過してからプロセッサによる検出が可能になります。レベル・センシティブの割込み生成に設定された場合、フラグがアサートされてからそのプログラム・フローが割り込まれるまでに4 SCLKサイクルの最小遅延があります。エッジ・センシティブの割込み生成に設定された場合、1 SCLKサイクルの遅延が追加されるため、エッジがアサートされてからコア・プログラム・フローが割り込まれるまでに合計で5 SCLKサイクルの遅延が生じます。

第15章　タイマ

このプロセッサは、3つの同一の32ビット汎用タイマ、コア・タイマ、ウォッチドッグ・タイマを備えています。

汎用タイマは、次の3つのいずれのモードにも個々に設定できます。

- パルス幅変調（PWM_OUT）モード
- パルス幅カウントおよびキャプチャ（WDTH_CAP）モード
- 外部イベント（EXT_CLK）モード

コア・タイマは、さまざまなシステム・タイミング機能に合わせて周期割込みを生成できます。

ウォッチドッグ・タイマを使用すれば、ソフトウェア・ウォッチドッグ機能を実装できます。ソフトウェア・ウォッチドッグは、タイマがソフトウェアによって更新される前に切れた場合に、Blackfinプロセッサ・コアへのイベントを生成することでシステムの可用性を高めることができます。

汎用タイマ

各汎用タイマには、TMRxという1本の専用の双方向ピンがあります。このピンは、PWM_OUTモードでは出力ピンとして機能し、WDTH_CAPモードとEXT_CLKモードでは入力ピンとして機能します。これらの機能を提供するため、各タイマには4本のレジスタがあります。範囲と精度については、タイマ・カウンタ（TIMERx_COUNTER）、タイマ周期（TIMERx_PERIOD）、タイマ・パルス幅（TIMERx_WIDTH）の各レジスタは32ビット幅です。図 15-1を参照してください。

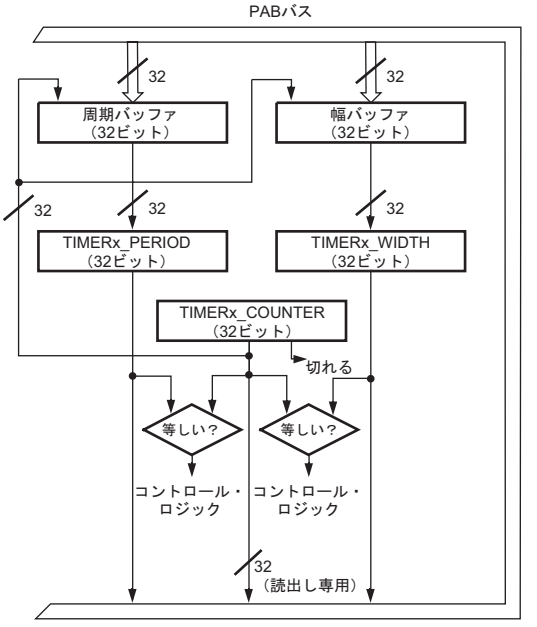


図 15-1. タイマ・ブロック図

各汎用タイマ用のレジスタは、次のとおりです。

- タイマ設定（TIMERx_CONFIG）レジスタ
- タイマ・カウンタ（TIMERx_COUNTER）レジスタ
- タイマ周期（TIMERx_PERIOD）レジスタ
- タイマ・パルス幅（TIMERx_WIDTH）レジスタ

内部的にクロック駆動される場合、クロック源はプロセッサのペリフェラル・クロック（SCLK）です。ペリフェラル・クロックは133MHzで動作していると想定した場合、タイマ・カウントの最大周期は（（2^{32} -1）/ 133MHz）＝32.2秒です。

タイマ・イネーブル（TIMER_ENABLE）レジスタを使用すれば、3つのタイマすべてを同時にイネーブルにできます。レジスタには、タイマごとに1つ、合計で3つの「Write-1-to-set」制御ビットが含まれています。同様に、3つのタイマを同時にまたは独立してディスエーブルにできるように、タイマ・ディスエーブル（TIMER_DISABLE）レジスタには3つの「Write-1-to-clear」制御ビットが含まれています。タイマ・イネーブル・レジスタまたはタイマ・ディスエーブル・レジスタをリードバックすれば、タイマのイネーブル・ステータスをチェックできます。1は、対応するタイマがイネーブルにされていることを示します。タイマは、TIMENxビットがセットされた3 SCLKサイクル後でカウントを開始します。

タイマ・ステータス（TIMER_STATUS）レジスタには、タイマごとの割込みラッチ・ビット（TIMILx）とオーバーフロー／エラー・インジケータ・ビット（TOVF_ERRx）が含まれています。これらのスティッキー・ビットはタイマ・ハードウェアによってセットされ、ソフトウェアによってポーリングされることがあります。これらのビットは、ビットに1を書き込むことによって、ソフトウェアで明示的にクリアする必要があります。

タイマの割込みをイネーブルにするには、タイマの設定（TIMERx_CONFIG）レジスタでIRQ_ENAビットをセットし、IMASKレジスタとSIC_IMASKレジスタの対応するビットをセットして、タイマの割込みをマスク解除します。IRQ_ENAビットをクリアした状態では、タイマは、そのタイマ割込みラッチ（TIMILx）ビットをセットしません。タイマ割込みを認めることなくTIMILxビットをポーリングするため、タイマの割込みがマスクされた状態のままでプログラムはIRQ_ENAビットをセットできます。

割込みの再発行を防ぐには、割込みをイネーブルにした状態で、割込みサービス・ルーチン（ISR）がRTI命令の前にTIMILxラッチをクリアするようにします。外部クロック（EXT_CLK）モードの場合、タイマ・イベントを確実に捕捉するには、割込みルーチンの先頭でラッチをリセットしてください。タイマ割込みをイネーブルにするには、適切なタイマ設定（TIMERx_CONFIG）レジスタでIRQ_ENAビットをセットします。

タイマ・レジスタ

タイマ・ペリフェラル・モジュールは、汎用のタイマ機能を提供します。このモジュールは、3つの同一のタイマ・ユニットで構成されます。

各タイマは、4本のレジスタを提供します。

- TIMERx_CONFIG[15:0]―タイマ設定レジスタ
- TIMERx_WIDTH[31:0]―タイマ・パルス幅レジスタ
- TIMERx_PERIOD[31:0]―タイマ周期レジスタ
- TIMERx_COUNTER[31:0]―タイマ・カウンタ・レジスタ

3つのタイマでは、3本のレジスタが共有されます。

- TIMER_ENABLE[15:0]―タイマ・イネーブル・レジスタ
- TIMER_DISABLE[15:0]―タイマ・ディスエーブル・レジスタ
- TIMER_STATUS[15:0]―タイマ・ステータス・レジスタ

アクセスのサイズが強化されます。タイマ設定レジスタへの32ビット・アクセスや、タイマ・パルス幅、タイマ周期、またはタイマ・カウンタ・レジスタへの16ビット・アクセスは、メモリマップド・レジスタ（MMR）エラーにつながります。タイマ・イネーブル、タイマ・ディスエーブル、タイマ・ステータスの各レジスタには、16ビットと32ビットの両方のアクセスが可能です。32ビット読出しでは、上位ワードはオール0を返します。

■ TIMER_ENABLE レジスタ

タイマ・イネーブル・レジスタ（TIMER_ENABLE）を使用すれば、3つのタイマすべてを同時にイネーブルにし、完全に同期して実行させることができます。各タイマには、1つのW1S制御ビットがあります。1を書き込む

と、対応するタイマがイネーブルになりますが、0を書き込んでも効果はありません。3つのビットは、個々にセットしたり、任意の組合わせでセットしたりできます。タイマ・イネーブル・レジスタを読み出すと、対応するタイマのイネーブル・ステータスが示されます。1は、タイマがイネーブルにされていることを示します。未使用ビットを読み出すと0が返されます。

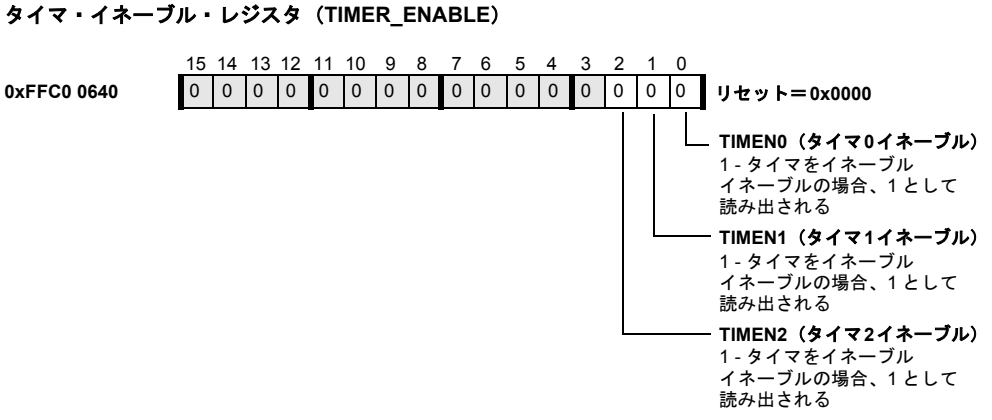


図 15-2. タイマ・イネーブル・レジスタ

■ TIMER_DISABLE レジスタ

タイマ・ディスエーブル・レジスタ（TIMER_DISABLE）を使用すれば、3つのタイマすべてを同時にディスエーブルにできます。各タイマには、1つのW1C制御ビットがあります。1を書き込むと、対応するタイマがディスエーブルになります。0を書き込んでも効果はありません。3つのビットは、個々にクリアしたり、任意の組合わせでクリアしたりできます。タイマ・ディスエーブル・レジスタの読出しでは、タイマ・イネーブル・レジスタの読出しと同じ値が返されます。1は、タイマがイネーブルにされていることを示します。未使用ビットを読み出すと0が返されます。

タイマ・ディスエーブル・レジスタ（TIMER_DISABLE）

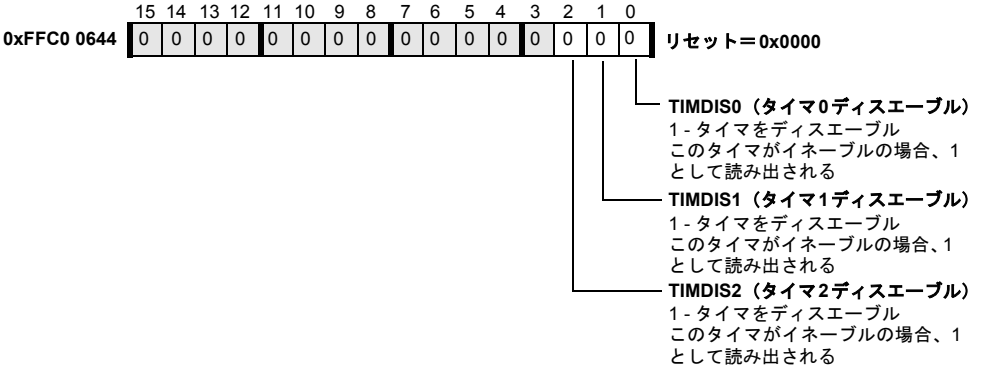


図 15-3. タイマ・ディスエーブル・レジスタ

PWM_OUTモードでは、TIMER_DISABLEに1を書き込んでも、対応するタイマはすぐには停止しません。むしろ、タイマは動作を継続して、現在の周期（PERIOD_CNT = 1の場合）またはパルス（PERIOD_CNT = 0の場合）の最後できれいに停止します。必要ならば、プロセッサは、まずTIMER_DISABLEの対応するビットに1を書き込んでから、TIMER_STATUSの対応するTRUNxビットに1を書き込むことで、PWM_OUTモードにあるタイマをすぐに停止させることができます。15-23ページの「PWM_OUTモードでのタイマの停止」を参照してください。

WDTH_CAPモードとEXT_CLKモードでは、TIMER_DISABLEに1を書き込むと、対応するタイマはすぐに停止します。

■ TIMER_STATUS レジスタ

タイマ・ステータス・レジスタ（TIMER_STATUS）は、3つのタイマすべてのステータスを示し、1回の読出しで3つのタイマすべてのステータスをチェックするために使用されます。ステータス・ビットは、スティッキーでW1Cです。TRUNxビットは、自分自身をクリアすることができ、PWM_OUTモードのタイマが周期の最後で停止するときにその動作を行います。ステータス・レジスタの読出しアクセス時には、すべての予備ビットや未使用ビットは0を返します。

各タイマが生成するユニークな割込み要求信号は、TIMERx_CONFIGレジスタの対応するIRQ_ENAビットによってゲートされます。共有されるタイマ・ステータス・レジスタ（TIMER_STATUS）はこれらの割込みをラッチするため、ユーザはユニークな割込み信号を参照することなく、割込みソースを判断できます（たとえば、3つのタイマすべてに同じ割込み優先順位が割り当てられた場合）。割込みビットはスティッキーであり、割込みが再発行されないことを保証するには、割込みサービス・ルーチン（ISR）によってクリアする必要があります。

割込み要求を示すために、TIMILxビットはタイマ設定レジスタのIRQ_ENAビットと連携して機能します。割込み条件またはエラーが発生し、IRQ_ENAがセットされた場合には、TIMILxビットがセットされ、コアへの割込みがアサートされます。この割込みは、システム割込みコントローラによってマスクされることがあります。割込み条件またはエラーが発生し、IRQ_ENAがクリアされた場合には、TIMILxビットはセットされず、割込みはアサートされません。TIMILxがすでにセットされており、IRQ_ENAに0が書き込まれた場合には、TIMILxはセットされたままであり、割込みはアサートされたままです。15-43ページの図15-24を参照してください。

TRUNxビットの読出し値は、全モードでタイマ・スレーブ・イネーブル・ステータスを反映します。つまり、セットされたTRUNxは実行中を示し、クリアされたTRUNxは停止を示します。TIMER_ENABLEレジスタとTIMER_DISABLEレジスタのTIMENxビットやTIMDISxビットの読出し値は、タイマがイネーブルであるかどうかを反映します。一方、TRUNxビットは、タイマが実際に実行中かどうかを示します。WDTH_CAPモードとEXT_CLKモードでは、TIMENxとTRUNxからの読出しは常に同じ値を返します。

TIMER_DISABLEレジスタへのW1C操作は、全モードで対応するタイマをディスエーブルにします。PWM_OUTモードでは、ディスエーブルにされたタイマは現在の周期（PERIOD_CNT = 1）またはパルス（PERIOD_CNT = 0）が完了するまで動作を継続します。この最終周期では、TIMENxビットは0を返しますが、TRUNxビットはまだ1として読み出されます。15-17ページの図15-10を参照してください。この状態でのみ、TRUNxはW1Cビットになります。タイマをディスエーブルにしたこの最終周期では、TRUNxに1を書き込むとTRUNxがクリアされ、タイマ・カウンタが現在のサイクルの最後に到達するのを待つことなく、タイマはすぐに停止します。

TRUNxビットへの書込みは、他のモードでは影響がなく、タイマがイネーブルにされていない場合にも影響がありません。PWM_OUTモードでTRUNxビットに1を書き込んでも、最初にディスエーブルにされなかったタイマには影響がありません。

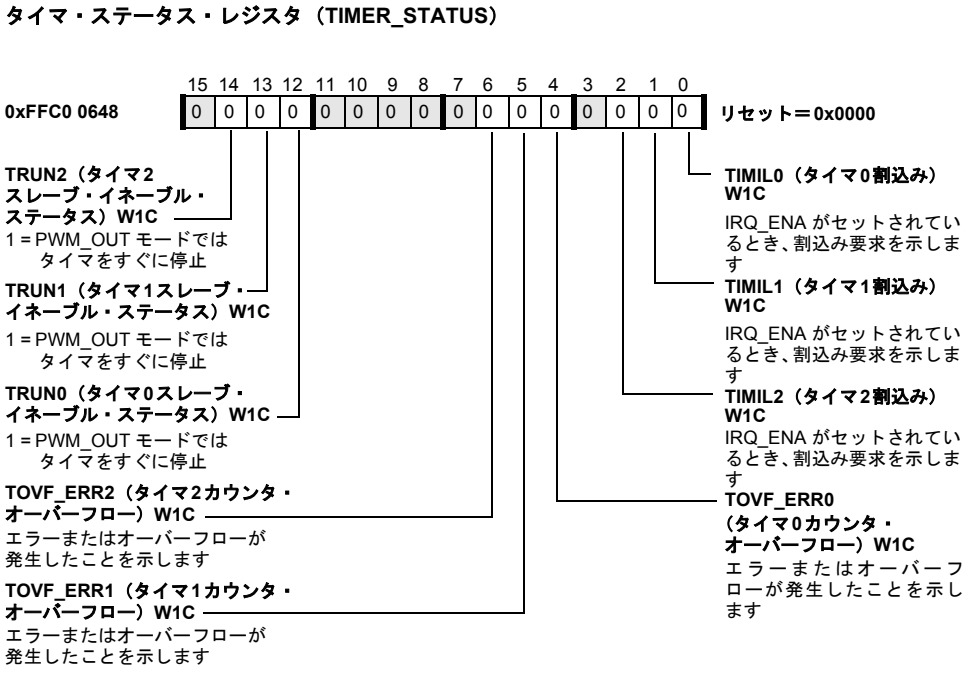


図 15-4. タイマ・ステータス・レジスタ

■ TIMERx_CONFIG レジスタ

各タイマの動作モードは、そのタイマ設定レジスタ（TIMERx_CONFIG）によって指定されます。TIMERx_CONFIG レジスタに書込みできるのは、タイマが動作中でないときに限られます。PWM_OUT モードでタイマをディスエーブルにした後、TIMERx_CONFIGを再プログラムする前に、TIMER_STATUSのTRUNx ビットをチェックしてタイマが動作を停止したことを確認します。TIMERx_CONFIG レジスタは、いつでも読出しできます。ERR_TYP フィールドは読出し専用です。このフィールドは、リセット時とタイマがイネーブルにされたときにクリアされます。TOVF_ERRx がセットされるたびに、ERR_TYP[1:0] には検出されたエラーのタイプを識別するコードがロードされます。この値は、次のエラーが発生するかタイマ・イネーブルが発生

するまで保持されます。エラー条件の概要については、15-45ページの表15-1を参照してください。TIMERx_CONFIGレジスタはTMRxピンの動作も制御します。このピンは、OUT_DISビットがクリアされると、PWM_OUTモード（TMODE = 01）で出力になります。

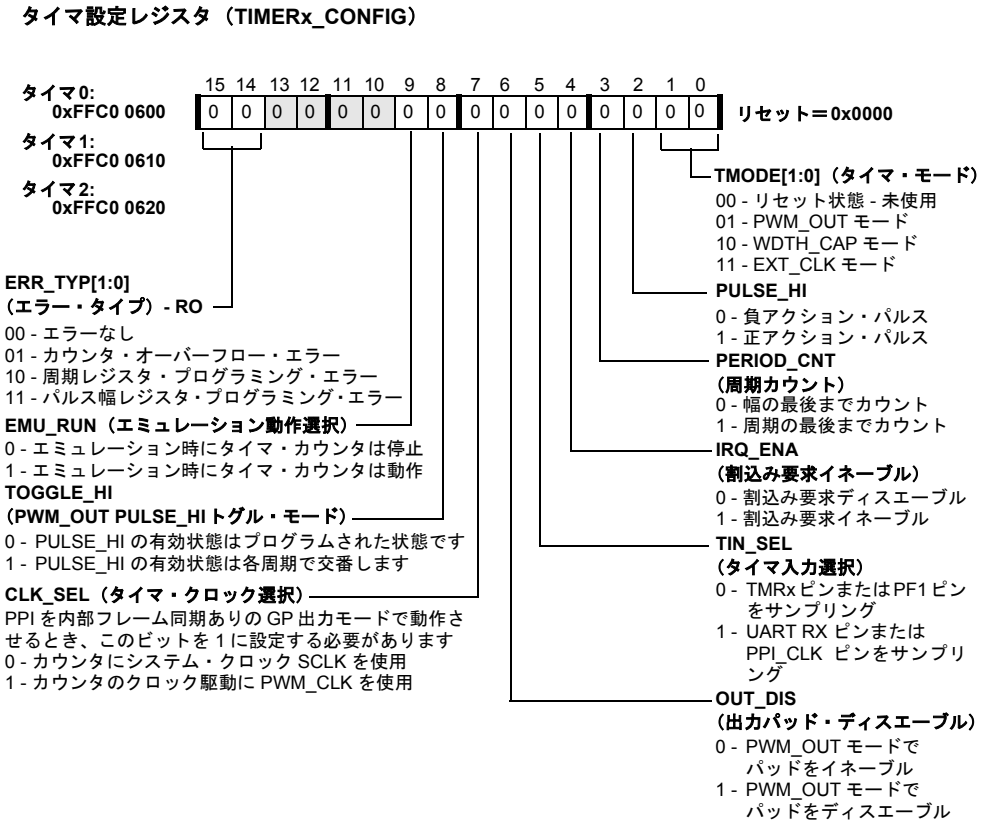


図 15-5. タイマ設定レジスタ

■ TIMERx_COUNTER レジスタ

これらの読出し専用レジスタは、ディスエーブルにされたときにその状態を保持します。イネーブルにされると、タイマ・カウンタ・レジスタ（TIMERx_COUNTER）は設定とモードに基づいて、ハードウェアによって再び初期化されます。タイマ・カウンタ・レジスタは、タイマが動作中かどうかとは無関係にいつでも読出しでき、コヒーレントな32ビット値を返します。インクリメント・カウンタは動作モードに応じて、SCLK、TMRxピン、プログラマブル・フラグ・ピンPF1、またはパラレル・ポート・クロックPPI_CLKという、4つの異なるソースによってクロック駆動できます。

プロセッサ・コアが外部エミュレータ・デバッガによってアクセスされている間、すべてのコード実行は停止します。デフォルトでは、TIMERx_COUNTERもエミュレーション・アクセス中にはカウントを停止して、ソフトウェアとの同期を維持します。停止中には、カウントは進みません。PWM_OUTモードでは、TMRxピンの波形が「伸ばされます」。WDTH_CAPモードでは、測定値は正しくありません。EXT_CLKモードでは、TMRxでの入力イベントが捕捉されないこともあります。エミュレーションの停止中には、レジスタの読出し／書込み、以前にアサートされた割込み（クリアされていない場合）、WDTH_CAPモードでのTIMERx_PERIODとTIMERx_WIDTHのロードなど、その他すべてのタイマ機能はアクティブのままです。

アプリケーションによっては、エミュレーションを停止したプロセッサ・コアとは非同期に、タイマにカウントを継続してほしい場合があります。この動作をイネーブルにするには、TIMERx_CONFIGのEMU_RUNビットをセットします。

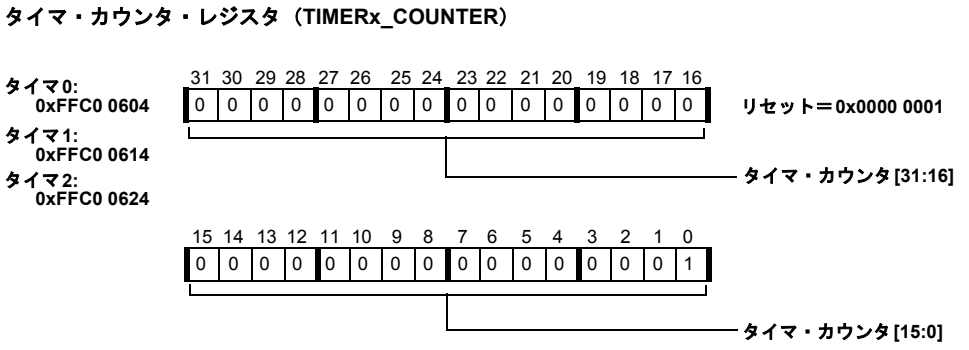


図 15-6. タイマ・カウンタ・レジスタ

■ TIMERx_PERIOD レジスタと TIMERx_WIDTH レジスタ

タイマがイネーブルで動作中であり、ソフトウェアがタイマ周期レジスタとタイマ・パルス幅レジスタに新しい値を書き込んだ場合、その書込みはバッファリングされ、現在の周期の最後までレジスタは更新されません（タイマ・カウンタ・レジスタがタイマ周期レジスタと等しいとき）。

タイマ周期レジスタ（TIMERx_PERIOD）とタイマ・パルス幅レジスタ（TIMERx_WIDTH）の使い方は、タイマのモードによって変化します。

- パルス幅変調モード（PWM_OUT）では、タイマ周期とタイマ・パルス幅（デューティ・サイクル）のレジスタ値が同時に変化するため、タイマ周期とタイマ・パルス幅のレジスタ値は「オンザフライ」で更新できます。

- パルス幅および周期キャプチャ・モード（WDTH_CAP）では、タイマ周期とタイマ・パルス幅のバッファ値は、適切なタイミングでキャプチャされます。その後、タイマ周期レジスタとタイマ・パルス幅レジスタは、それぞれのバッファから同時に更新されます。このモードでは、両方のレジスタが読出し専用です。
- 外部イベント・キャプチャ・モード（EXT_CLK）では、タイマ周期レジスタは書込み可能であり、「オンザフライ」で更新できます。タイマ・パルス幅レジスタは使用されません。

タイマ周期レジスタやタイマ・パルス幅レジスタに新しい値が書き込まれない場合には、以前の周期からの値が再利用されます。32ビットのタイマ周期レジスタとタイマ・パルス幅レジスタへの書込みはアトミックです。上位ワードに書き込むには、下位ワードにも書き込むことが必要です。

タイマ周期レジスタやタイマ・パルス幅レジスタに書き込まれた値は、必ずバッファ・レジスタに格納されます。タイマ周期レジスタやタイマ・パルス幅レジスタからの読出しは、常に周期やパルス幅の現在のアクティブ値を返します。書き込まれた値は、アクティブになるまでは読み出されません。タイマがイネーブルにされると、現在の周期の最後にタイマ周期レジスタとタイマ・パルス幅レジスタがそれぞれのバッファから更新されるまで、書き込まれた値はアクティブになりません。15-2ページの図15-1を参照してください。

タイマがディスエーブルにされると、バッファ・レジスタへの書込みは、すぐにタイマ周期レジスタやタイマ・パルス幅レジスタにコピーされ、最初のタイマ周期で使用できるよう準備されます。たとえば、タイマがイ

ネーブルにされた後、最初の3つのタイマ周期ごとに別の設定値を使用するために、タイマ周期レジスタやタイマ・パルス幅レジスタの値を変更するには、次の手順に従ってください。

1. レジスタ値の最初のグループをプログラムします。
2. タイマをイネーブルにします。
3. レジスタ値の 2 番目のグループをすぐにプログラムします。
4. 最初のタイマ割込みを待ちます。
5. レジスタ値の 3 番目のグループをプログラムします。

タイマ割込みが受信されると、それぞれの新しい設定値がプログラムされます。

非常に小さな周期（10 カウント未満）の PWM_OUT モードでは、バッファ・レジスタからの更新と更新の間には、タイマ周期レジスタとタイマ・パルス幅レジスタの両方に書き込む時間が不足することもあります。次の周期では、1 つの古い値と 1 つの新しい値を使用する可能性があります。パルス幅＞＝周期のエラーを防止するには、値を減らすときにタイマ・パルス幅レジスタに書き込んでからタイマ周期レジスタに書き込み、値を増やすときにタイマ周期レジスタに書き込んでからタイマ・パルス幅レジスタに書き込んでください。

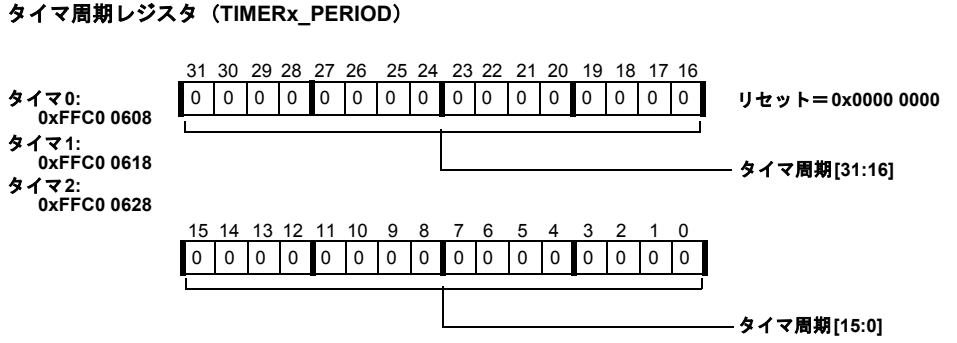


図 15-7. タイマ周期レジスタ

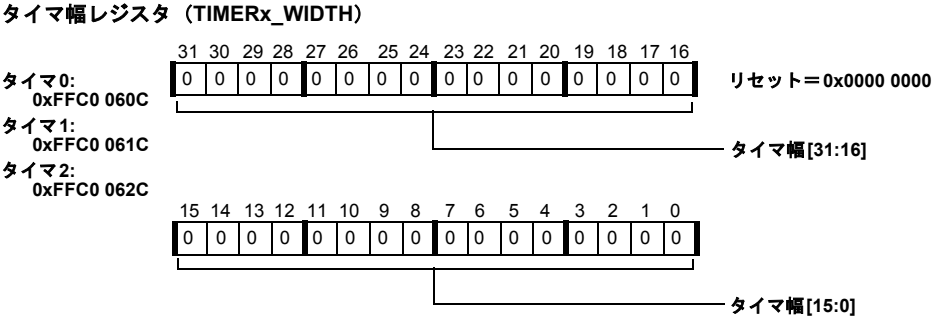


図 15-8. タイマ幅レジスタ

タイマの使い方

個々のタイマをイネーブルにするには、TIMER_ENABLE レジスタでそのタイマの TIMEN ビットをセットします。個々のタイマをディスエーブルにするには、TIMER_DISABLE レジスタでそのタイマの TIMDIS ビットをセットします。3つのタイマすべてを同時にイネーブルにするには、TIMER_ENABLE レジスタで3つの TIMEN ビットをすべてセットします。

タイマをイネーブルにする前に、対応するタイマ設定（TIMERx_CONFIG）レジスタを必ずプログラムしてください。このレジスタでは、タイマの動作モード、TMRx ピンの極性、タイマ割込み動作を定義します。タイマの実行中には、動作モードを変更しないでください。

タイマのイネーブル／ディスエーブル・タイミングの例を図 15-9、図 15-10、図 15-11 に示します。

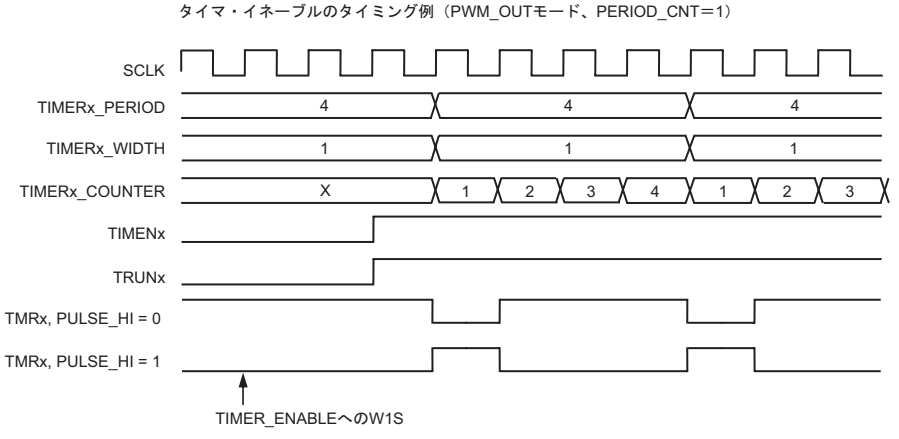


図 15-9. タイマ・イネーブルのタイミング

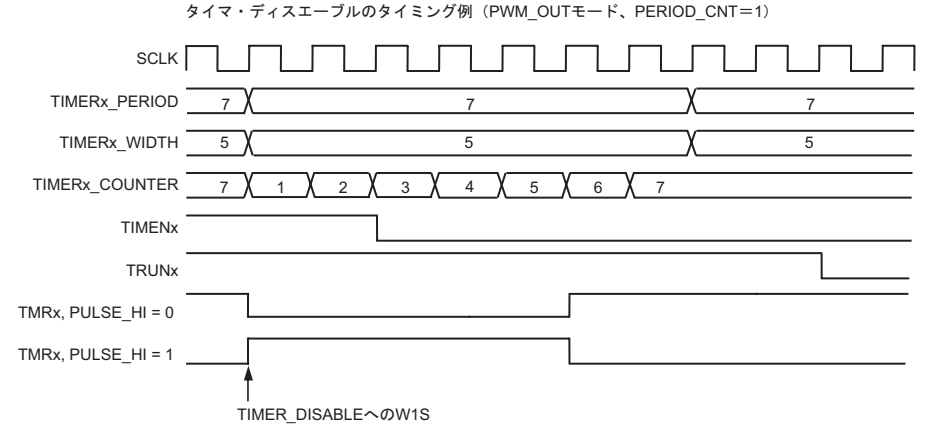


図 15-10.タイマ・ディスエーブルのタイミング

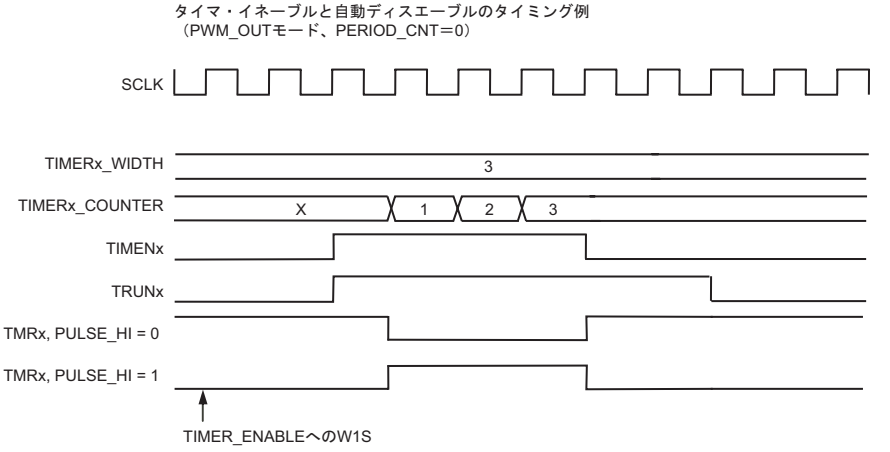


図 15-11. タイマ・イネーブルと自動ディスエーブルのタイミング

タイマがディスエーブルにされた場合、タイマ・カウンタ・レジスタはその状態を保持します。タイマが再びイネーブルにされた場合、タイマ・カウンタは動作モードに基づいて再び初期化されます。タイマ・カウンタ・レジスタは読出し専用です。ソフトウェアでは、タイマ・カウンタ値を直接に上書きしたりプリセットしたりできません。

■ パルス幅変調（PWM_OUT）モード

タイマ設定（TIMERx_CONFIG）レジスタでTMODEフィールドをb#01に設定すると、PWM_OUTモードがイネーブルにされます。PWM_OUTモードでは、タイマのTMRxピンは出力です。この出力をディスエーブルにするには、タイマ設定レジスタでOUT_DISビットをセットします。

PWM_OUT モードでは、ビット PULSE_HI、PERIOD_CNT、IRQ_ENA、OUT_DIS、CLK_SEL、EMU_RUN、TOGGLE_HI がそれぞれ独立して機能します。これらのビットは、個々にセットしたり、任意に組み合わせてセットしたりできます。しかし、一部の組合わせは使用できません（PERIOD_CNT = 0 や OUT_DIS = 1 での TOGGLE_HI = 1 など）。

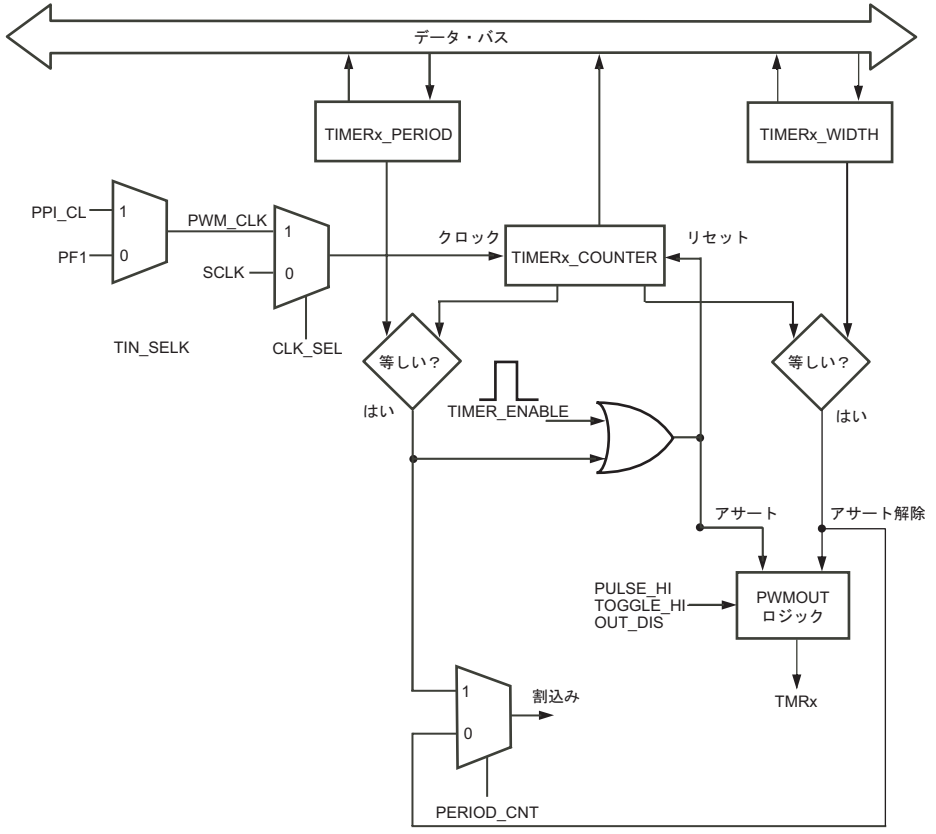


図 15-12.タイマのフロー図、PWM_OUT モード

タイマがイネーブルにされると、タイマ・カウンタ・レジスタには開始値がロードされます。CLK_SEL = 0の場合、タイマ・カウンタは0x1から開始します。CLK_SEL = 1の場合、EXT_CLKモードのようにタイマ・カウンタは0x0にリセットされます。タイマは、タイマ周期レジスタの値までカウント・アップします。CLK_SELの設定がどちらであっても、タイマ・カウンタがタイマ周期に等しくなると、タイマ・カウンタは次のクロックで0x1にリセットされます。

PWM_OUTモードでは、タイマが1つのパルスを生成するか多くのパルスを生成するかは、PERIOD_CNTビットによって制御されます。PERIOD_CNTがクリアされると（PWM_OUT単一パルス・モード）、タイマはTIMERx_WIDTHレジスタを使用し、1つのアサート・エッジと1つのアサート解除エッジを生成してから、割込みを生成し（イネーブルの場合）停止します。PERIOD_CNTがセットされると（PWM_OUT連続パルス・モード）、タイマはTIMERx_PERIODレジスタとTIMERx_WIDTHレジスタを使用して、反復のある（そしておそらく変調された）波形を生成します。タイマは各周期の最後に割込みを生成し（イネーブルの場合）、ディスエーブルにされた後でのみ停止します。PERIOD_CNT ＝0の設定では幅の最後までカウントし、PERIOD_CNT＝1の設定では周期の最後までカウントします。

動作モードによっては、TIMERx_PERIODレジスタとTIMERx_WIDTHレジスタは読出し専用です。これらのレジスタに書き込む前に、TIMERx_CONFIGレジスタのTMODEフィールドにb#01を設定してください。

▶ 出力パッド・ディスエーブル

PWM_OUTモードで出力ピンをディスエーブルにするには、タイマ設定レジスタで OUT_DIS ビットをセットします。これによって、PULSE_HI や TOGGLE_HIの設定とは無関係に、TMRxピンは3ステートになります。これによって、出力信号が使用されていないときに消費電力を削減できます。

▶ 単一パルス生成

PERIOD_CNT ビットがクリアされている場合、PWM_OUT モードは TMRx ピン上に単一パルスを生成します。このモードは、正確な遅延を実現するためにも使用できます。パルス幅はタイマ・パルス幅レジスタによって定義され、タイマ周期レジスタは使用されません。

パルスの最後に、タイマ割込みラッチ・ビット TIMILx がセットされ、タイマは自動的に停止します。PULSE_HI ビットがセットされた場合には、TMRx ピン上にアクティブ・ハイ・パルスが生成されます。PULSE_HI がセットされない場合には、パルスはアクティブ・ローです。

▶ パルス幅変調波形の生成

PERIOD_CNT ビットがセットされている場合、内部的にクロック駆動されたタイマは、明確な周期とデューティ・サイクルを持つ矩形信号を生成します。このモードは、リアルタイム信号処理のための周期割込みも生成します。

32 ビットのタイマ周期（TIMERx_PERIOD）レジスタとタイマ・パルス幅（TIMERx_WIDTH）レジスタは、タイマ・カウント周期とパルス幅変調された出力パルス幅の値によってプログラムされます。

タイマがこのモードでイネーブルにされると、タイマ・カウンタがタイマ・パルス幅レジスタの値に等しくなるたびに TMRx ピンはアサート解除された状態にされ、周期が終了する（またはタイマが開始される）とピンは再びアサートされます。

TMRx ピンのアサーション・センスを制御するため、対応する TIMERx_CONFIG レジスタの PULSE_HI ビットが使用されます。ロー・アサーション・レベルの場合には、このビットをクリアします。ハイ・アサーション・レベルの場合には、このビットをセットします。タイマが PWM_OUT モードでディスエーブルにされると、TMRx ピンはアサート解除されたレベルに駆動されます。

イネーブルである場合、タイマ割込みは各周期の最後に生成されます。割込みサービス・ルーチン（ISR）は、割込みラッチ・ビット（TIMILx）をクリアする必要があり、周期や幅の値を変更することもあります。パルス幅変調（PWM）アプリケーションでは、ソフトウェアはタイマの実行中に、周期とパルス幅の値を更新する必要があります。ソフトウェアが周期レジスタまたはパルス幅レジスタを更新すると、新しい値は周期が終了するまで特殊なバッファ・レジスタによって保持されます。その後、新しい周期とパルス幅の値は同時にアクティブになります。古い値が使用されている間に、新しいタイマ周期レジスタとタイマ・パルス幅レジスタの値が書き込まれます。タイマ・カウンタの値が現在のタイマ周期値に等しくなると、新しい値がロードされて使用されます。タイマ周期レジスタとタイマ・パルス幅レジスタからの読出しは、周期が終了するまで古い値を返します。

PWM_OUTモードでは、TOVF_ERRxステータス・ビットはエラー条件を示します。スタートアップ時にTIMERx_PERIOD = 0またはTIMERx_PERIOD = 1である場合、またはタイマ・カウンタ・レジスタがロール・オーバーした場合には、TOVF_ERRxビットがセットされます。また、タイマ・パルス幅レジスタがタイマ周期レジスタ以上である場合には、タイマ・カウンタ・レジスタがロール・オーバーしたときにもセットされます。TOVF_ERRxビットがセットされると、ERR_TYPビットがセットされます。

TMRx出力ピンで最大周波数を生成するには、周期値を2に、パルス幅を1に設定します。これによってTMRxは各SCLKクロックをトグルさせ、50%のデューティ・サイクルが得られます。周期には2～（$2^{32}-1$）までの任意の値をプログラムできます。パルス幅には1～（周期－1）までの任意の値をプログラムできます。PERIOD_CNT＝0の場合、パルス幅には1～（$2^{32}-1$）までの任意の値をプログラムできます。

TIMERx_WIDTH値がTIMERx_PERIOD値に等しい場合には、ハードウェアはエラーを報告します。しかしこれは100%のデューティ・サイクルを持つPWMパターンを実装するためには依然として有効な動作です。そうする場合、一般にソフトウェアはTOVL_ERRxフラグを無視する必要があります。

周期値を超えるパルス幅値はお勧めできません。同様に、TIMERx_WIDTH＝0も有効な動作ではありません。0%のデューティ・サイクルには対応していません。

▶ PWM_OUT モードでのタイマの停止

あらゆる形式の PWM_OUT モードで、タイマは、ディスエーブル動作（TIMER_DISABLEへのW1C）を「停止は保留中」条件として扱います。ディスエーブルにされると、タイマは現在の波形を自動的に完了してから、きれいに停止します。これによって、現在のパルスの切り捨てとTMRxピンでの望ましくないPWMパターンを防止できます。プロセッサがタイマの実行停止を確認するには、TIMER_STATUSレジスタの対応するTRUNxビットをポーリングして0を読み出すか、または最後の割込みを待機します（イネーブルの場合）。なお、タイマが停止してTRUNxが0として読み出されるまでは、タイマの再設定はできません（TIMERx_CONFIGに新しい値を書き込むことができません）。

PWM_OUTの単一パルス・モード（PERIOD_CNT = 0）では、TIMER_DISABLEに書き込んでタイマを停止させる必要はありません。パルスの最後に、タイマは自動的に停止し、TIMER_ENABLE（とTIMER_DISABLE）の対応するビットはクリアされ、対応するTRUNxビットはクリアされます。15-18ページの図15-11を参照してください。複数のパルスを生成するには、TIMER_ENABLEに1を書き込み、タイマの停止を待ってから、TIMER_ENABLEにさらに1を書き込みます。

必要ならば、プロセッサは、タイマをPWM_OUTモードにしてすぐに停止させることができます。それには、まずTIMER_DISABLEの対応するビットに1を書き込んでから、TIMER_STATUSの対応するTRUNxビットに1を書き込みます。これによってタイマは停止し、保留中の停止が現在の周期の最後

（PERIOD_CNT = 1）や現在のパルス幅の最後（PERIOD_CNT = 0）を待機しているかどうかとは無関係です。この機能を使用すれば、エラー回復シーケンスの処理中に、タイマの直接制御を回復できることがあります。

この機能は注意してご使用ください。TMRx ピンで生成された PWM パターンが破損する可能性もあります。

PWM_OUT の連続パルス・モード（PERIOD_CNT = 1）では、各タイマは、各周期の最後にその TIMENx ビットをサンプリングします。TIMENx がローレベルの場合、タイマは最初の周期の最後にきれいに停止します。これは（TRUNx への W1C を除いて）、ディスエーブルにされてから現在の周期の最後までに再びイネーブルにされたタイマは何も起きなかったかのように実行を続けることを意味します。一般に、ソフトウェアは PWM_OUT タイマをディスエーブルにしてから、タイマが停止するまで待つようにします。PERIOD_CNT = 0 の場合、タイマは最初のパルスの最後に必ず停止します。

▶ 外部的にクロック駆動される PWM_OUT

デフォルトでは、タイマは SCLK によって内部的にクロック駆動されます。あるいは、タイマ設定（TIMERx_CONFIG）レジスタの CLK_SEL ビットがセットされている場合には、タイマは PWM_CLK によってクロック駆動されます。PWM_CLK は、通常は PF1 ピンから入力されますが、タイマが PPI と連携するように設定されている場合には、PPI_CLK ピンから得られることもあります。タイマは設定に応じて、PWM_CLK 入力でさまざまな信号を受信することがあります。PERIOD_CNT ビットでの選択に基づいて、PWM_OUT モードはパルス幅変調波形を生成したり、TIMERx_WIDTH レジスタによって定義されたパルス幅を持つ単一パルスを生成します。

CLK_SEL がセットされると、カウンタはスタートアップ時に 0x0 にリセットされ、PWM_CLK の各立上がりエッジでインクリメントされます。TMRx ピンは、PWM_CLK の立上がりエッジで遷移します。PWM_CLK の立下がりエッジを選択する方法はありません。このモードでは、PULSE_HI ビットは生成されたパルスの極性だけを制御します。タイマ割込みは、TMRx ピンでの対応するエッジの少し前で発生することがあります（割込みは SCLK エッジで発

生し、ピンは後のPWM_CLKエッジで遷移します）。新しい周期とパルス幅の値は、割込みが発生したらすぐにプログラムすると依然として安全です。周期が終了した後、カウンタは0x1の値にロール・オーバーします。

PWM_CLKクロック波形は、50%のデューティ・サイクルを持つ必要はありませんが、最小のPWM_CLKクロック・ロー時間は1 SCLK周期であり、最小のPWM_CLKクロック・ハイ時間は1 SCLK周期です。これは、最大のPWM_CLKクロック周波数がSCLK/2であることを意味します。

PF1が入力ピンとして機能するとき、PF1ピンはタイマをクロック駆動するだけです。いずれかのタイマが、CLK_SEL = 1およびTIN_SEL＝0でPWM_OUTモードにあるとき、FIO_DIRレジスタのPF1ビットは無視され、PF1は入力にされます。

▶ PULSE_HI トグル・モード

PERIOD_CNT ＝ 1 による PWM_OUT モードで生成された波形は通常、TIMERx_WIDTHレジスタによって固定されたアサーション時間とプログラマブルなアサート解除時間を持っています。2つのタイマが同じ周期設定によって同期して動作しているとき、図 15-13に示すように、パルスはアサート側のエッジに合わせられます。

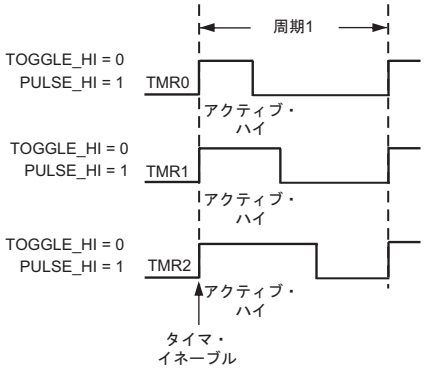


図 15-13.アサート側エッジに合わせられたパルスを持つタイマ

TOGGLE_HI モードでは、生成された出力波形のアサート側エッジとアサート解除側エッジのタイミングを制御できます。2つのタイマ出力のアサート側エッジ間での位相はプログラマブルです。PULSE_HI ビットの有効状態はすべての周期で反転します。隣接するアクティブ・ローとアクティブ・ハイのパルスはひとまとめにされて、完全に任意の矩形波形の半分を2つ生み出します。有効な波形は、PULSE_HI がセットされている場合は依然としてアクティブ・ハイであり、PULSE_HI がクリアされている場合はアクティブ・ローです。モードが PWM_OUT で PERIOD_CNT ＝ 1 の場合を除いて、TOGGLE_HI の値には効果がありません。

TOGGLE_HI モードでは、PULSE_HI がセットされると、アクティブ・ロー・パルスは、1番目、3番目、およびすべての奇数番号周期で生成されます。そしてアクティブ・ハイ・パルスは、2番目、4番目、およびすべての偶数番号周期で生成されます。PULSE_HI がクリアされると、アクティブ・ハイ・パルスは、1番目、3番目、およびすべての奇数番号周期で生成され、アクティブ・ロー・パルスは、2番目、4番目、およびすべての偶数番号周期で生成されます。

1つの周期の最後でアサート解除された状態は、次の周期の先頭でアサートされた状態と一致するため、カウント＝パルス幅の場合にのみ出力波形は遷移します。最終的に、2つのカウンタ周期ごとに繰り返し、最初の周期の最後（または2番目の周期の先頭）を中心とする出力波形パルスとなります。

図 15-14は、同じ周期設定で動作する3つのタイマによる例を示します。ソフトウェアが実行時にPWM設定を変更しない場合、デューティ・サイクルは50%です。TIMERx_WIDTH レジスタの値は、信号間での位相を制御します。

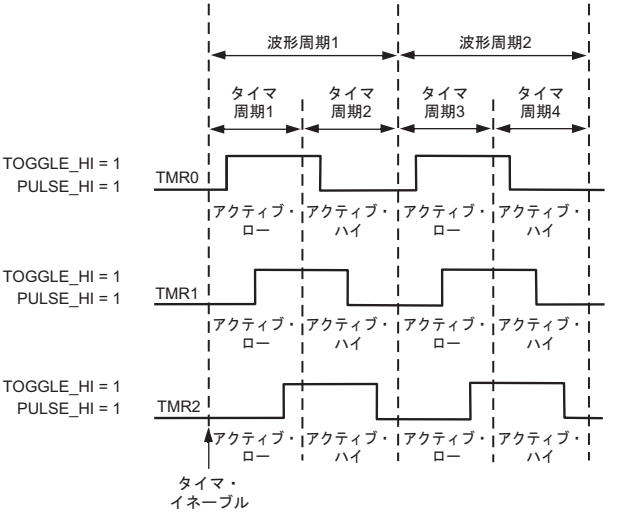


図 15-14.同じ周期設定を持つ 3 つのタイマ

同様に、2つのタイマを使用し、パルスを中心に合わせてタイマの1つの信号極性を反転することによって、重なりのないクロックを生成できます（図 15-15を参照）。

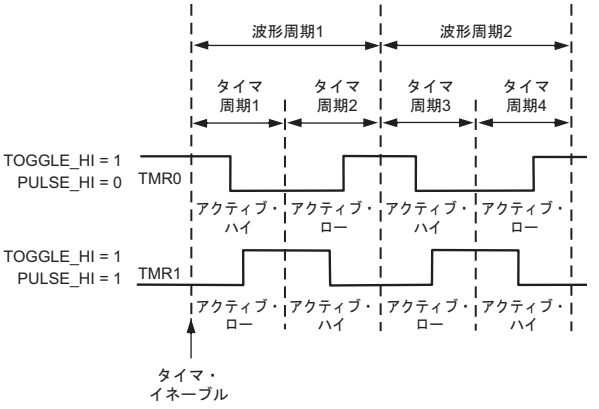


図 15-15.重なりのないクロックを持つ 2 つのタイマ

TOGGLE_HI＝0の場合、ソフトウェアは波形周期ごとに1回、タイマ周期レジスタとタイマ・パルス幅レジスタを更新します。TOGGLE_HI = 1の場合、ソフトウェアは波形周期ごとに2回、半分の大きさの値によってタイマ周期レジスタとタイマ・パルス幅レジスタを更新します。奇数番号の周期では、中央揃えされたパルスを得るために、幅ではなく（周期－幅）をタイマ・パルス幅レジスタに書き込みます。

たとえば、TOGGLE_HI = 0の場合の擬似コードが次のとおりだとします。

```
int period, width ;
for (;;) {
     period = generate_period(...) ;
     width = generate_width(...) ;

     waitfor (interrupt) ;

     write(TIMERx_PERIOD, period) ;
     write(TIMERx_WIDTH, width) ;
}
```

TOGGLE_HI＝1の場合、擬似コードは次のようになります。

```
int period, width ;
int per1, per2, wid1, wid2 ;

for (;;)  {
     period = generate_period(...) ;
     width = generate_width(...) ;

     per1 = period/2 ;
     wid1 = width/2 ;

     per2 = period/2 ;
     wid2 = width/2 ;

     waitfor (interrupt) ;

     write(TIMERx_PERIOD, per1) ;
     write(TIMERx_WIDTH, per1 - wid1) ;

     waitfor (interrupt) ;

     write(TIMERx_PERIOD, per2) ;
     write(TIMERx_WIDTH, wid2) ;

}
```

この例に示すように、生成されるパルスは対称である必要はありません（wid1とwid2が等しい必要はありません）。生成されるパルスの位相を調整するため、周期はオフセットできます（per1とper2が等しい必要はありません）。

タイマ・スレーブ・イネーブル・ビット（TIMER_STATUS レジスタの TRUNx ビット）は、TOGGLE_HI モードで偶数番号の周期の最後にのみ更新されます。TIMER_DISABLE に 1 が書き込まれると、カウンタ周期の現在のペア（1 つの波形周期）は、タイマがディスエーブルにされる前に完了します。

TOGGLE_HI ＝ 0 では、次の場合にエラーが報告されます。

TIMERx_WIDTH >= TIMERx_PERIOD、TIMERx_PERIOD ＝ 0、または TIMERx_PERIOD ＝ 1

■ パルス幅カウントおよびキャプチャ（WDTH_CAP）モード

WDTH_CAP モードでは、TMRx ピンは入力ピンです。外部から印加される矩形波形の周期とパルス幅を確認するために、内部的にクロック駆動されるタイマが使用されます。このモードをイネーブルにするには、TIMERx_CONFIG（タイマ設定レジスタ）の TMODE フィールドに b#10 を設定します。

このモードでイネーブルにされると、タイマは TIMERx_COUNTER レジスタのカウントを 0x0000 0001 にリセットし、TMRx ピンで前縁を検出してからカウントを開始します。

タイマは最初の前縁を検出すると、インクリメントを開始します。波形の後縁を検出すると、タイマは TIMERx_COUNTER レジスタの現在の 32 ビット値を幅バッファ・レジスタにキャプチャします。次の前縁で、タイマは TIMERx_COUNTER レジスタの現在の 32 ビット値を周期バッファ・レジスタに転送します。カウント・レジスタは再び 0x0000 0001 にリセットされ、タイマはディスエーブルにされるまでカウントとキャプチャを継続します。

このモードでは、ソフトウェアは波形のパルス幅とパルス周期の両方を測定できます。TMRx ピンの前縁と後縁の定義を制御するため、TIMERx_CONFIG レジスタの PULSE_HI ビットがセットまたはクリアされます。PULSE_HI ビットがクリアされた場合、測定は立下がりエッジによって開始され、タイマ・カウンタ・レジスタは立上がりエッジでタイマ・パルス幅バッファ・レジスタ

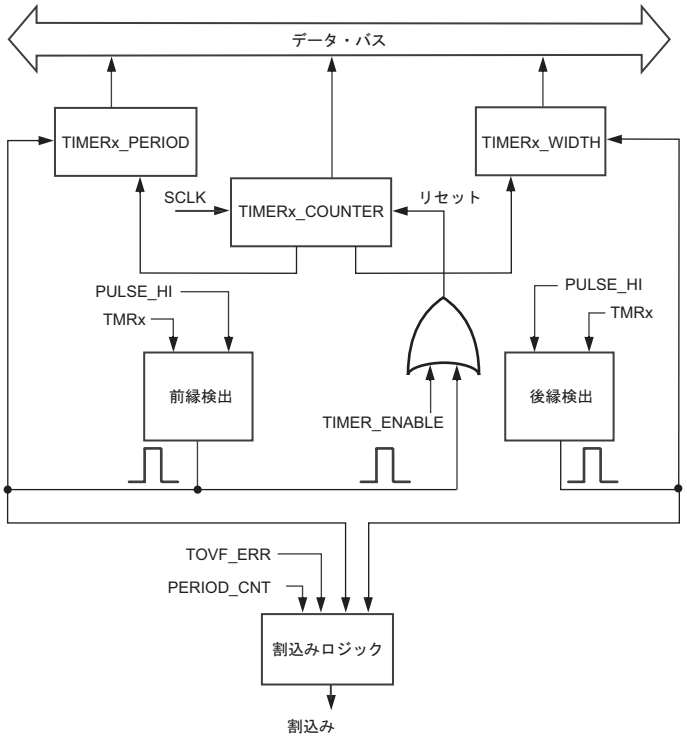


図 15-16.タイマのフロー図、WDTH_CAP モード

にキャプチャされ、タイマ周期は次の立下がりエッジでキャプチャされます。PULSE_HI ビットがセットされると、測定は立上がりエッジで開始され、タイマ・カウンタ・レジスタは立下がりエッジでタイマ・パルス幅バッファ・レジスタにキャプチャされ、タイマ周期は次の立上がりエッジでキャプチャされます。

WDTH_CAPモードでは、次の3つのイベントは常に1つの単位として同時に発生します。

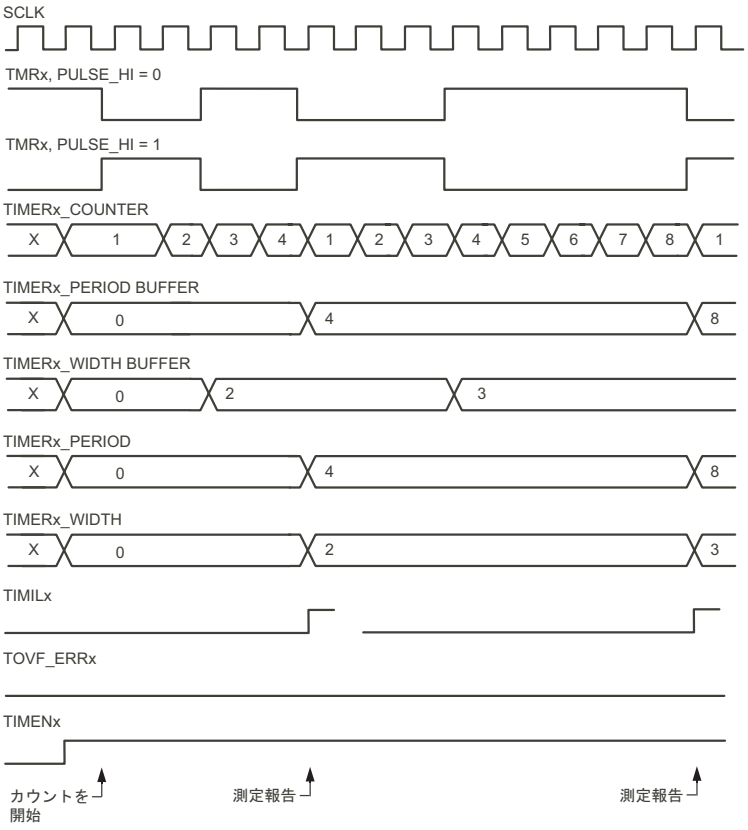
1. TIMERx_PERIOD レジスタは、周期バッファ・レジスタをもとに更新されます。
2. TIMERx_WIDTH レジスタは、幅バッファ・レジスタをもとに更新されます。
3. タイマ割込みラッチ・ビット（TIMILx）はセットされます（イネーブルの場合）が、エラーを生成しません。

このトランザクション群が実行される時点は、TIMERx_CONFIGレジスタのPERIOD_CNTビットによって制御されます。総合して、これら3つのイベントは測定報告と呼ばれます。測定報告では、タイマ・カウンタ・オーバーフロー・エラー・ラッチ・ビット（TOVF_ERRx）はセットされません。測定報告は、入力信号周期ごとに多くて1回行われます。

現在のタイマ・カウンタ値は、入力信号の後縁と前縁でそれぞれ常に幅バッファ・レジスタと周期バッファ・レジスタにコピーされますが、これらの値はソフトウェアからは見えません。測定報告イベントは、キャプチャされた値を目に見えるレジスタにサンプリングし、タイマ割込みを設定してTIMERx_PERIODとTIMERx_WIDTHで読出しの準備ができたことを通知します。PERIOD_CNTビットがセットされると、測定報告は、周期バッファ・レジスタが（前縁で）その値をキャプチャした直後に行われます。PERIOD_CNTビットがクリアされると、測定報告は、幅バッファ・レジスタが（後縁で）その値をキャプチャした直後に行われます。

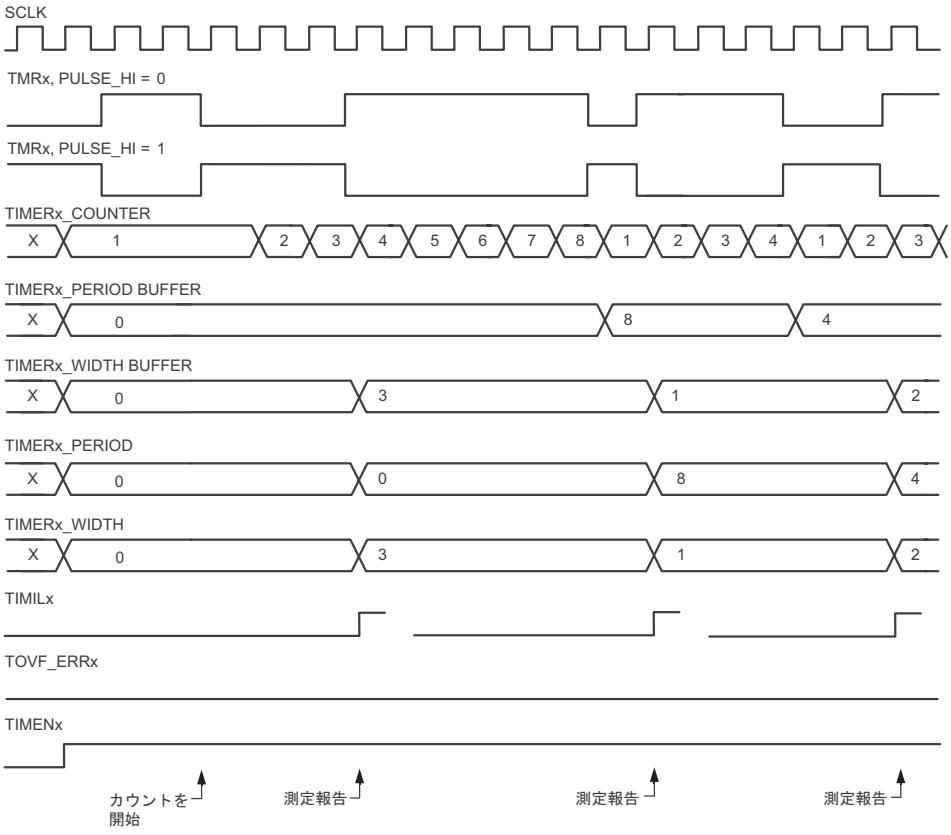
PERIOD_CNTビットがセットされ、前縁が発生した場合（図 15-17を参照）には、TIMERx_PERIODレジスタとTIMERx_WIDTHレジスタは、終わったばかりの周期で測定されたパルス周期とパルス幅を報告します。PERIOD_CNTビットがクリアされ、後縁が発生した場合（図 15-18を参照）には、

TIMERx_WIDTH レジスタは、終わったばかりのパルスで測定されたパルス幅を報告しますが、TIMERx_PERIOD レジスタは、前の周期の最後で測定されたパルス周期を報告します。



注：簡便化をはかり、TMRxエッジとバッファ・レジスタ更新との間の同期遅延は示しません。

図 15-17.周期キャプチャ測定報告のタイミング例（WDTH_CAP モード、PERIOD_CNT ＝ 1）



注：簡便化をはかり、TMRxエッジとバッファ・レジスタ更新との間の同期遅延は示しません。

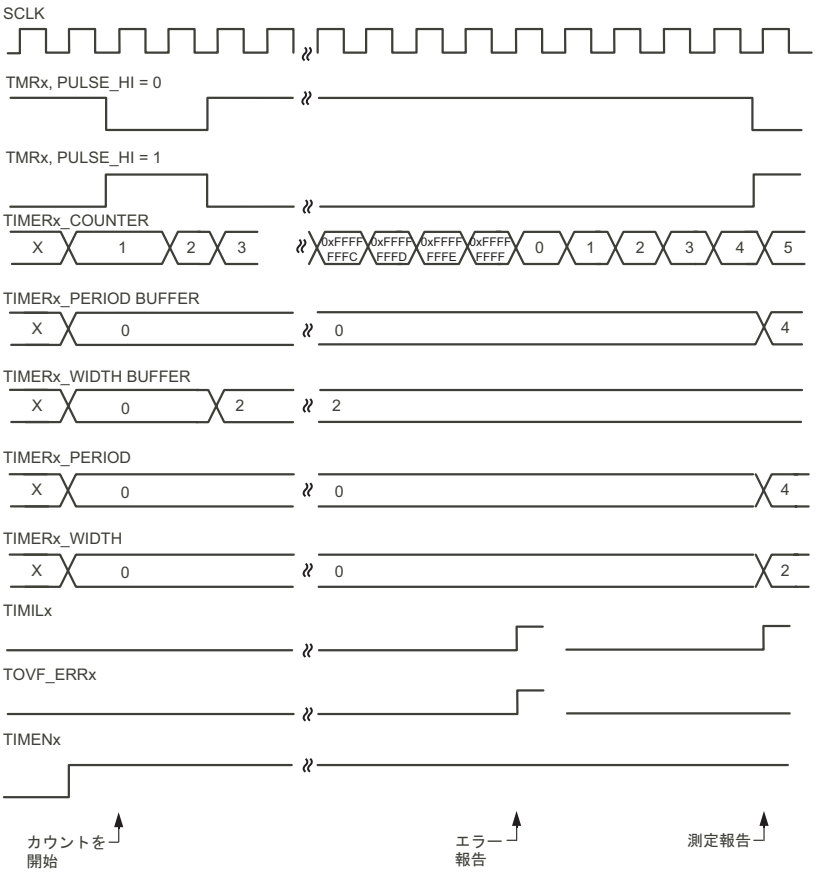
図 15-18.幅キャプチャ測定報告のタイミング例（WDTH_CAP モード、PERIOD_CNT ＝ 0）

PERIOD_CNT ビットがクリアされ、最初の後縁が発生した場合、最初の測定報告では最初の周期値がまだ測定されていないため、周期値は有効ではありません。図 15-18に示すように、この場合のTIMERx_PERIOD値を読み出すと0が返されます。1つの前縁と1つの後縁だけを持つ波形のパルス幅を測定するには、PERIOD_CNT ＝0に設定します。ここでPERIOD_CNT ＝1の場

合には、周期値が周期バッファ・レジスタにキャプチャされません。代わりに、カウンタ範囲を超えてカウンタがラップすると、エラー報告割込みが生成されます（イネーブルの場合）。この場合、TIMERx_WIDTH と TIMERx_PERIOD は両方とも0を示します（これは、幅バッファ・レジスタにキャプチャされた値を TIMERx_WIDTH にコピーする測定報告が行われなかったためです）。図 15-19 の最初の割込みを参照してください。

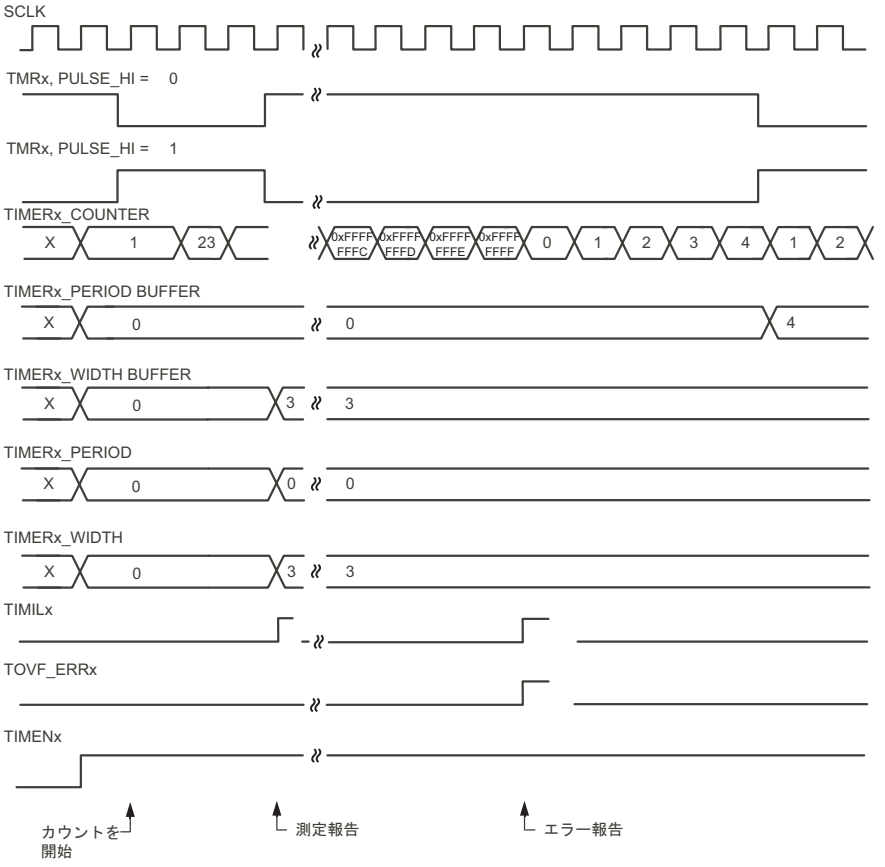
上述の PERIOD_CNT＝0モードを使用して単一パルスの幅を測定するとき、測定間隔を終了させる割込みを受け取った後で、タイマをディスエーブルにすることをお勧めします。必要ならば、別の測定を準備するために、タイマを適宜再イネーブルにできます。この手順によって、タイマが幅測定の後で自走したり、タイマ・カウントのオーバーフローによって生成されたエラーを記録したりすることを防止できます。

前縁がないときにタイマ・カウンタ・レジスタが0xFFFF FFFFから0にラップした場合、タイマ割込みが生成されます（イネーブルの場合）。その時点で、TIMER_STATUS レジスタの TOVF_ERRx ビットと TIMERx_CONFIG レジスタの ERR_TYP ビットがセットされ、カウンタの範囲を超える周期によるカウントのオーバーフローを示します。これはエラー報告と呼ばれます。タイマが WDTH_CAP モードで割込みを生成した場合、その原因は、エラーが発生した（エラー報告）か、新しい測定の読出し準備ができている（測定報告）かのいずれかです。ただし、同時に両方が発生することはありません。エラーが通知されたとき、TIMERx_PERIOD レジスタと TIMERx_WIDTH レジスタは更新されません。詳細については、図 15-19 と図 15-20 を参照してください。



注：簡便化をはかり、TMRxエッジとバッファ・レジスタ更新との間の同期遅延は示しません。

図 15-19.周期オーバーフローに続く周期キャプチャのタイミング例（WDTH_CAP モード、PERIOD_CNT ＝ 1）



注：簡便化をはかり、TMRxエッジとバッファ・レジスタ更新との間の同期遅延は示しません。

図 15-20.幅キャプチャに続く周期オーバーフローのタイミング例（WDTH_CAP モード、PERIOD_CNT = 0）

TIMILx と TOVF_ERRx はいずれもスティッキー・ビットであり、ソフトウェアで明示的にクリアする必要があります。タイマがオーバーフローし PERIOD_CNT = 1 の場合には、TIMERx_PERIOD レジスタも TIMERx_WIDTH レジ

スタも更新されませんでした。タイマがオーバーフローしPERIOD_CNT = 0の場合には、前の測定報告で後縁が検出された場合に限り、TIMERx_PERIODレジスタとTIMERx_WIDTHレジスタが更新されました。

ソフトウェアは、測定報告割込み間でのエラー報告割込みの数をカウントして、0xFFFF FFFFより長い入力信号周期を測定できます。各エラー報告割込みは、周期に対する合計に2^{32} SCLKのフル・カウントを加算しますが、幅は曖昧です。たとえば、図 15-19では、周期は0x1 0000 0004ですが、パルス幅は0x0 0000 0002または0x1 0000 0002とすることができます。

TMRxピンに印加される波形は、50%のデューティ・サイクルを持つ必要はありませんが、最小のTMRxロー時間は1 SCLK周期であり、最小のTMRxハイ時間は1 SCLK周期です。これは、最大のTMRx入力周波数が50%のデューティ・サイクルを持つSCLK/2であることを意味します。これらの条件のもとで、WDTH_CAPモード・タイマは、**周期 = 2**と**パルス幅 = 1**を測定します。

▶ オートボー・モード

3つのタイマのうちの1つは、非同期シリアル・インターフェースLSI（UART）にオートボー検出機能を提供します。TIMERx_CONFIGレジスタのタイマ入力選択（TIN_SEL）ビットによって、タイマはWDTH_CAPモード用にイネーブルにされると、TMRxピンの代わりにUARTポート受信データ（RX）ピンをサンプリングします。

オートボー検出が完了するまでは、UARTをイネーブルにしないでください。

ソフトウェア・ルーチンは、シリアル・ストリーム・ビット・セルのパルス幅を検出できます。タイマのサンプル・ベースはUART動作と同期 — すべてフェーズ・ロック・ループ（PLL）クロックから取得 — しているため、パルス幅を使用してUARTのボーレート・デバイダを計算できます。

デバイザ＝（（TIMERx_WIDTH）/（16×キャプチャされたUARTビット数））

タイマ・カウントの数、つまりキャプチャされる信号の分解能を増やすには、単にシングル・ビットのパルス幅を測定するのではなく、対象となるパルスをより多くのビットに拡大することをお勧めします。図 15-21 に示すように、オートボー検出には一般にNULL文字（ASCII 0x00）が使用されます。

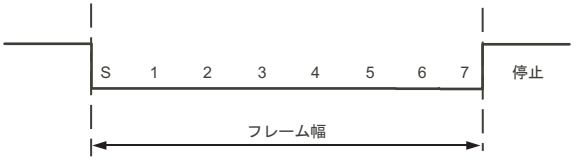


図 15-21.オートボー検出文字 0x00

図 15-21 のフレーム例には、8つのデータビットと1つのスタート・ビットが含まれているため、次の式を適用します。

デバイザ＝`TIMERx_WIDTH / (16×9)`

実際のUART RX信号には非対称な立下がり／立上がりエッジが含まれることもあり、サンプリング・ロジック・レベルは信号電圧範囲のちょうど中央にはありません。高いビットレートでは、このようなパルス幅ベースのオートボー検出は、アナログ信号コンディショニングを追加しなければ適切な結果を返せないことがあります。信号周期の測定はこの問題の回避につながるため、強くお勧めします。

たとえば、オートボー検出バイトとしてASCII文字「@」（40h）を事前定義し、その後の2つの立下がりエッジ間で周期を測定します。図 15-22 に示すように、スタート・ビットの立下がりエッジとビット6の後の立下がりエッジとの間で周期を測定します。この周期には8つのビットが含まれるため、次の式を適用します。

デバイザ＝`TIMERx_PERIOD / (16×8)`

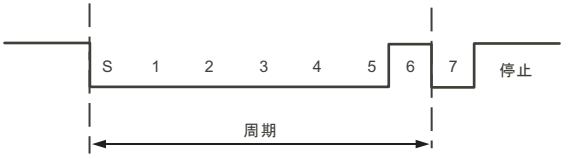


図 15-22.オートボー検出文字 0x40

■ 外部イベント（EXT_CLK）モード

EXT_CLKモードでは、TMRxピンは入力です。タイマは、外部ソースによってクロック駆動されるカウンタとして機能します。外部ソースは、システム・クロックに非同期にすることもできます。TIMERx_COUNTERの現在のカウントは、検出された前縁イベントの数を表します。このモードをイネーブルにするには、TIMERx_CONFIGレジスタのTMODEフィールドにb#11を設定します。TIMERx_PERIODレジスタは、最大のタイマ外部カウントの値によってプログラムされます。

TMRxピンに印加される波形は、50%のデューティ・サイクルを持つ必要はありません。しかし、最小のTMRxロー時間は1 SCLK周期であり、最小のTMRxハイ時間は1 SCLK周期です。これは、最大のTMRx入力周波数がSCLK/2であることを意味します。

周期には1～（$2^{32}-1$）の任意の値をプログラムできます。

タイマがイネーブルにされた後、タイマ・カウンタ・レジスタは0x0にリセットされ、TMRxピンでの最初の前縁を待ちます。このエッジでは、タイマ・カウンタ・レジスタは値0x1にインクリメントされます。その後の前縁のたびに、カウント・レジスタはインクリメントされます。周期値に到達した後、TIMILxビットがセットされ、割込みが生成されます。次の前縁では、タイマ・カウンタ・レジスタには再び0x1がロードされます。タイマは、ディスエーブルにされるまでカウントを継続します。PULSE_HIビットでは、前縁が立上がりであるか（PULSE_HI がセット）、立下がりであるか（PULSE_HIがクリア）を決定します。

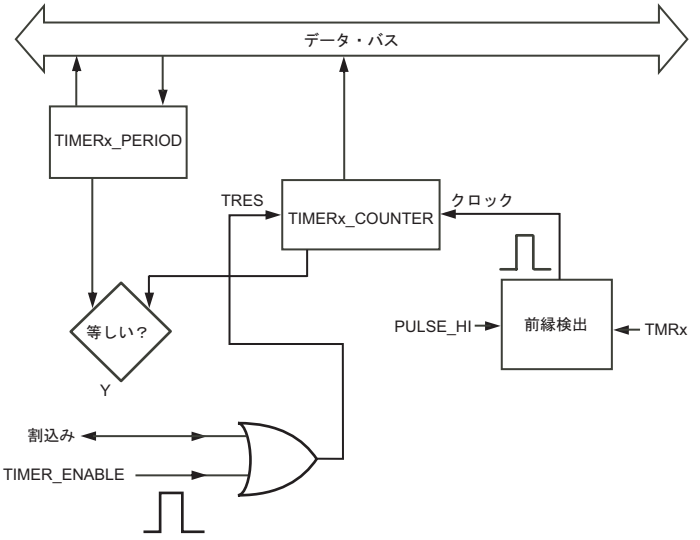


図 15-23.タイマのフロー図、EXT_CLK モード

このモードでは、設定ビットTIN_SELとPERIOD_CNTには効果がありません。タイマ・カウンタ・レジスタが0xFFFF FFFFから0にラップした場合、あるいはスタートアップ時またはタイマ・カウンタ・レジスタのロール・オーバー（カウント＝周期からカウント＝0x1に）時に周期=0である場合には、TOVF_ERRxビットとERR_TYPビットがセットされます。タイマ・パルス幅レジスタは未使用です。

■ PPI でのタイマの使い方

特定のPPIモード用のフレーム同期信号を生成するために、タイマが2つまで使用されます。PPIで使用するタイマの詳しい設定方法については、PPIに関して記載した11-30ページの「GPモードでのフレーム同期」を参照してください。

■ 割込み

3つのタイマは、それぞれ1つの割込みを生成できます。得られた3つの割込み信号は、優先順位付けとマスキングのために、システム割込みコントローラ・ブロックに転送されます。タイマ・ステータス(TIMER_STATUS)レジスタがタイマ割込みをラッチすることで、ソフトウェアは割込みソースを判定することができます。これらのビットはW1Cであり、割込みの再発行を防止するには、RTIの前にクリアする必要があります。

割込み生成をイネーブルにするには、IRQ_ENAビットをセットし、システム割込みマスク・レジスタ（SIC_IMASK）で割込みソースをマスク解除します。割込み生成なしにTIMILxビットをポーリングするにはIRQ_ENAをセットしますが、割込みはマスクされたままにしておきます。IRQ_ENAによってイネーブルにされた場合、エラー条件によって割込み要求も生成されます。

システム割込みコントローラは、柔軟な割込み処理を可能にします。すべてのタイマは、同じ割込みチャンネルを共有することも共有しないこともできるため、1つの割込みルーチンが複数のタイマを処理することもあります。PWMモードでは、より多くのタイマが同じ周期設定で動作し、割込み要求を同時に発行することができます。この場合、サービス・ルーチンは、TIMER_STATUSレジスタに0x07を書き込むことによって、すべてのTIMILxラッチ・ビットを同時にクリアできます。

割込みがイネーブルである場合には、RTI命令が実行される前に、割込みサービス・ルーチン（ISR）がTIMERx_STATUSレジスタのTIMILxビットをクリアするようにしてください。これによって、割込みの再発行を防止できます。システム・レジスタへの書込みには遅延が生じることを忘れないでください。ごく少数の命令でTIMILxクリア・コマンドをRTI命令から分離している場合には、追加のSSYNC命令が挿入されることもあります。EXT_CLKモードでは、タイマ・イベントを確実に捕捉するために、割込みサービス・ルーチン（ISR）の先頭でTIMERx_STATUSレジスタのTIMILxビットをリセットします。

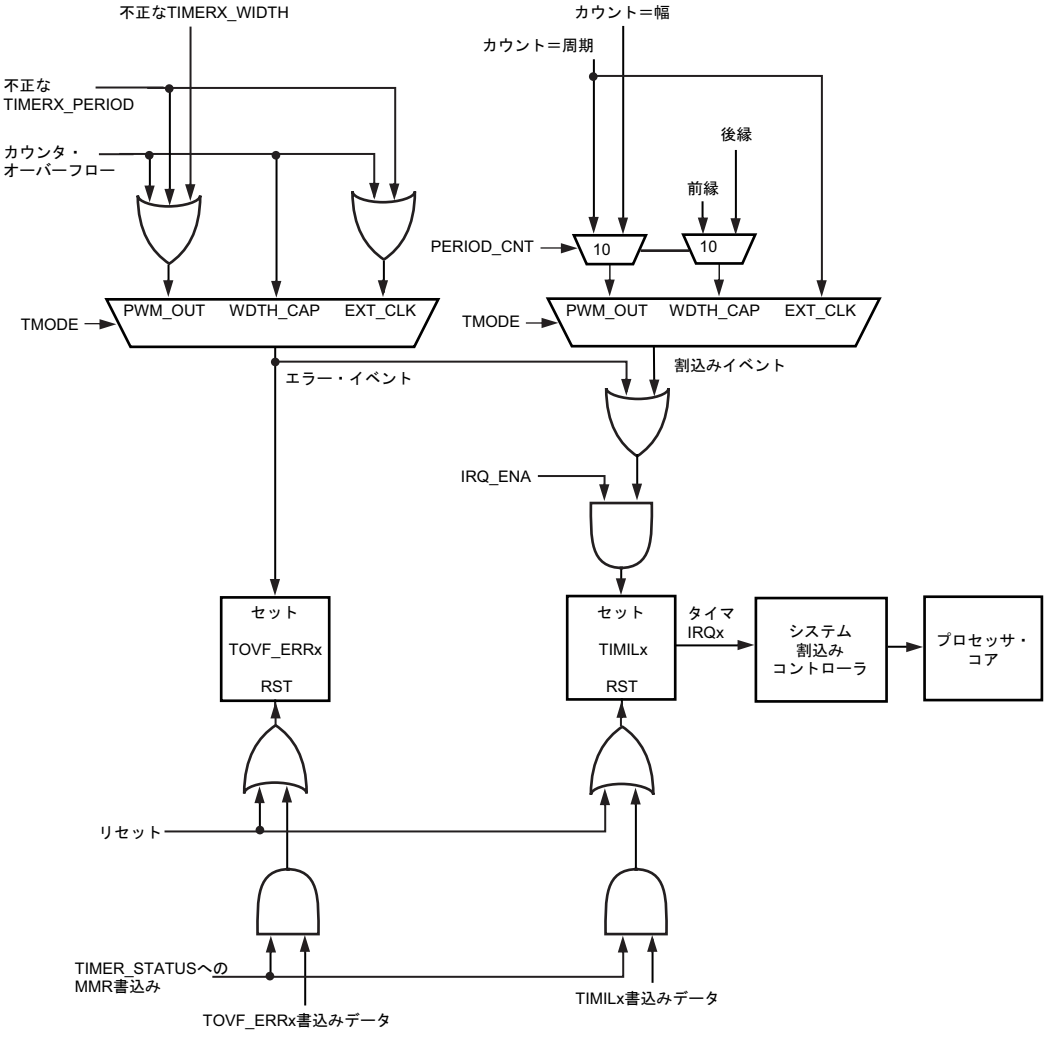


図 15-24.タイマ割込み構造

■ 不正状態

表 15-1では、以下の定義が使用されています。

- **スタートアップ**:TIMER_ENABLEへの書込みによってタイマがイネーブルにされた後、タイマ・カウンタが動作している最初のクロック周期です。
- **ロールオーバー**：現在のカウントがTIMERx_PERIODの値と一致し、カウンタに値1が再ロードされるときです。
- **オーバーフロー**：タイマ・カウンタは、0xFFFF FFFFという可能な最大カウント値を保持していたとき、ロールオーバーする代わりにインクリメントされました。カウンタには、次の大きな値を表すだけの十分な範囲がないため、誤って0x0000 0000という新しい値をロードします。
- **変化なし**：新しいエラーはありません。
  - ERR_TYPが変化なし：以前に報告されたエラー・コード、またはこのタイマがイネーブルにされて以降にエラーがなかった場合には00が表示されます。
  - TOVF_ERR が変化なし：このタイマがイネーブルにされて以降にエラーがなかった場合、またはソフトウェアがW1Cを実行して以前のエラーをクリアした場合には0が読み出されます。前のエラーがソフトウェアによってアクノレッジされていない場合には、TOVF_ERRは1が読み出されます。

ソフトウェアでは、TOVF_ERRを読み出してエラーをチェックしてください。TOVF_ERRがセットされている場合には、ソフトウェアはERR_TYPを読み出して詳細を得ることができます。エラーが検出されたら、ソフトウェアで1を書き込んでTOVF_ERRをクリアし、エラーをアクノレッジしてください。

次の表は、「～モードの～イベントでは、TIMERx_PERIODが～でTIMERx_WIDTHが～の場合、ERR_TYPは～でありTOVF_ERRは～です」と読むことができます。

スタートアップ・エラー条件は、タイマの起動を妨げません。同様に、オーバーフローやロールオーバーのエラー条件は、タイマを停止させません。不正なケースでは、TMRxピンは望ましくない動作をすることがあります。

表 15-1. 不正状態の概要

モード	イベント	TIMERx_PERIOD	TIMERx_WIDTH	ERR_TYP	TOVF_ERR
PWM_OUT, PERIOD_CNT = 1	スタートアップ（TIMERx_WIDTH については境界条件テストを実行しません）	== 0	任意	b#10	セット
		== 1	任意	b#10	セット
		>= 2	任意	変化なし	変化なし
	ロールオーバー	== 0	任意	b#10	セット
		== 1	任意	b#11	セット
		>= 2	== 0	b#11	セット
		>= 2	< TIMERx_PERIOD	変化なし	変化なし
		>= 2	>= TIMERx_PERIOD	b#11	セット
	オーバーフロー、TIMERx_PERIOD==0 など別のエラーもない限り不可能	任意	任意	b#01	セット

表 15-1. 不正状態の概要（続き）

モード	イベント	TIMERx_ PERIOD	TIMERx_ WIDTH	ERR_TYP	TOVF_ERR
PWM_OUT, PERIOD_ CNT = 0	スタートアップ	任意	== 0	b#01	セット
		このケースは、スタートアップ時には検出されませんが、カウンタがその範囲全体までカウントするとオーバーフロー・エラーにつながります。			
		任意	>= 1	変化なし	変化なし
	ロールオーバー	このモードではロールオーバーは不可能です。			
	オーバーフロー、TIMERx_WIDTH==0 など別のエラーもない限り不可能	任意	任意	b#01	セット
WDTH_CAP	スタートアップ	このモードでは、TIMERx_PERIOD と TIMERx_WIDTH は読出し専用、エラーは不可能。			
	ロールオーバー	このモードでは、TIMERx_PERIOD と TIMERx_WIDTH は読出し専用、エラーは不可能。			
	オーバーフロー	任意	任意	b#01	セット
EXT_CLK	スタートアップ	== 0	任意	b#10	セット
		>= 1	任意	変化なし	変化なし
	ロールオーバー	== 0	任意	b#10	セット
		>= 1	任意	変化なし	変化なし
	オーバーフロー、TIMERx_WIDTH==0 など別のエラーもない限り不可能	任意	任意	b#01	セット

■ まとめ

表 15-2は、各タイマ・モードでの制御ビットとレジスタの使い方を要約します。

表 15-2. 制御ビットとレジスタの使い方一覧表

ビット／レジスタ	PWM_OUT モード	WDTH_CAP モード	EXT_CLK モード
TIMER_ENABLE	1 - タイマをイネーブル 0 - 効果なし	1 - タイマをイネーブル 0 - 効果なし	1 - タイマをイネーブル 0 - 効果なし
TIMER_DISABLE	1 - 周期の最後でタイマをディスエーブル 0 - 効果なし	1 - タイマをディスエーブル 0 - 効果なし	1 - タイマをディスエーブル 0 - 効果なし
TMODE	b#01	b#10	b#11
PULSE_HI	1 - ハイレベル幅を生成 0 - ローレベル幅を生成	1 - ハイレベル幅を測定 0 - ローレベル幅を測定	1 - 立上がりエッジをカウント 0 - 立下がりエッジをカウント
PERIOD_CNT	1 - PWM を生成 0 - 単一幅パルス	1 - 周期測定後の割込み 0 - 幅測定後の割込み	未使用
IRQ_ENA	1 - 割込みをイネーブル 0 - 割込みをディスエーブル	1 - 割込みをイネーブル 0 - 割込みをディスエーブル	1 - 割込みをイネーブル 0 - 割込みをディスエーブル
TIN_SEL	CLK_SEL に依存： CLK_SEL ＝ 1 の場合、 1 - PPI_CLK をカウント 0 - PF1 クロックをカウント CLK_SEL ＝ 0 の場合、 未使用	1 - RX 入力を選択 0 - TMRx 入力を選択	未使用
OUT_DIS	1 - TMRx ピンをディスエーブル 0 - TMRx ピンをイネーブル	未使用	未使用

表 15-2. 制御ビットとレジスタの使い方一覧表（続き）

ビット／レジスタ	PWM_OUT モード	WDTH_CAP モード	EXT_CLK モード
CLK_SEL	1 - PWM_CLK がタイマをクロック駆動 0 - SCLK がタイマをクロック駆動	未使用	未使用
TOGGLE_HI	1 - 2 つのカウンタ周期ごとに 1 つの波形周期 0 - 1 つのカウンタ周期ごとに 1 つの波形周期	未使用	未使用
ERR_TYP	適宜、b#00、b#01、b#10、またはb#11を報告	適宜、b#00 または b#01 を報告	適宜、b#00、b#01、または b#10 を報告
EMU_RUN	0 - エミュレーション時に停止 1 - エミュレーション時にカウント	0 - エミュレーション時に停止 1 - エミュレーション時にカウント	0 - エミュレーション時に停止 1 - エミュレーション時にカウント
TMR ピン	OUT_DIS に依存： 1 - 3 ステート 0 - 出力	TIN_SEL に依存： 1 - 未使用 0 - 入力	入力
周期	R/W：周期値	RO：周期値	R/W：周期値
幅	R/W：幅値	RO：幅値	未使用
カウンタ	RO：SCLK または PWM_CLK でカウント・アップ	RO:SCLK でカウント・アップ	RO：イベントでカウント・アップ
TRUNx	読出し：タイマ・スレーブ・イネーブル・ステータス 書込み： 1 - ディスエーブルの場合、タイマを停止 0 - 効果なし	読出し：タイマ・スレーブ・イネーブル・ステータス 書込み： 1 - 効果なし 0 - 効果なし	読出し：タイマ・スレーブ・イネーブル・ステータス 書込み： 1 - 効果なし 0 - 効果なし

表 15-2. 制御ビットとレジスタの使い方一覧表（続き）

ビット／レジスタ	PWM_OUT モード	WDTH_CAP モード	EXT_CLK モード
TOVF_ERR	周期＝ 0 または 1 の場合、スタートアップまたはロールオーバー時にセット 幅＞＝周期の場合、ロールオーバー時にセット カウンタがラップした場合にセット	カウンタがラップした場合にセット	カウンタがラップした場合にセット、あるいは周期＝ 0 の場合には、スタートアップまたはロールオーバー時にセット
IRQ	IRQ_ENA に依存： 1 - TOVF_ERRがセットされた場合、カウンタが周期と等しく PERIOD_CNT ＝ 1 の場合、またはカウンタが幅と等しく PERIOD_CNT ＝ 0 の場合にセット 0 - セットされません	IRQ_ENA に依存： 1 - TOVF_ERR がセットされた場合、カウンタが周期をキャプチャし PERIOD_CNT ＝ 1 の場合、またはカウンタが幅をキャプチャし PERIOD_CNT ＝ 0 の場合にセット 0 - セットされません	IRQ_ENA に依存： 1 - カウンタが周期に等しいか TOVF_ERR がセットされた場合にセット 0 - セットされません

コア・タイマ

コア・タイマは、周期割込みを生成できるプログラマブルなインターバル・タイマです。コア・タイマは、コア・クロック（CCLK）レートで動作します。タイマには、4本のコア・メモリマップド・レジスタ（MMR）、タイマ・コントロール・レジスタ（TCNTL）、タイマ・カウント・レジスタ（TCOUNT）、タイマ周期レジスタ（TPERIOD）、タイマ・スケール・レジスタ（TSCALE）が含まれます。

図 15-25は、コア・タイマのブロック図を示します。

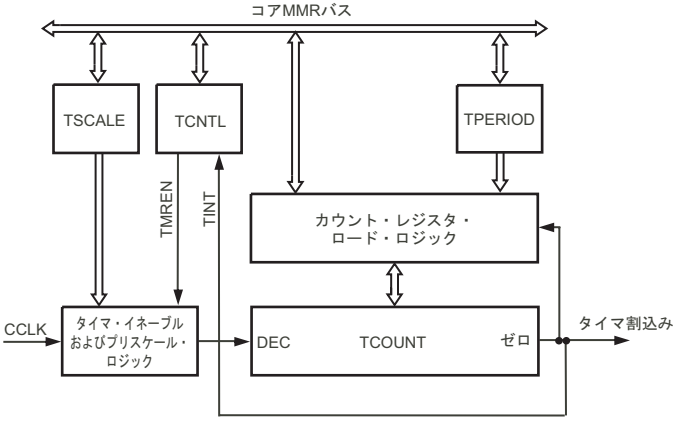


図 15-25.コア・タイマのブロック図

■ TCNTL レジスタ

コア・タイマ・コントロール・レジスタ（TCNTL）のTMRENビットをセットしてタイマがイネーブルにされると、TCOUNTレジスタはTSCALE＋1個のクロック・サイクルごとに1回デクリメントされます。TCOUNTレジスタの値が0になると割込みが生成され、TCNTLレジスタのTINTビットがセットされます。TCNTLレジスタのTAUTORLDビットがセットされた場合には、TCOUNTレジスタにはTPERIODレジスタの内容が再ロードされ、再びカウントが始まります。

コア・タイマを低消費電力モードにするには、TCNTL レジスタの TMPWR ビットをクリアします。タイマを使用する前に、TMPWR ビットをセットしてください。これによって、タイマ・ユニットにクロックが復元されます。TMPWR がセットされると、TCNTL レジスタの TMREN ビットをセットしてコア・タイマをイネーブルにできます。

TMPWR ＝ 0 のときに TMREN がセットされた場合、ハードウェアの動作は不定です。

コア・タイマ・コントロール・レジスタ（TCNTL）

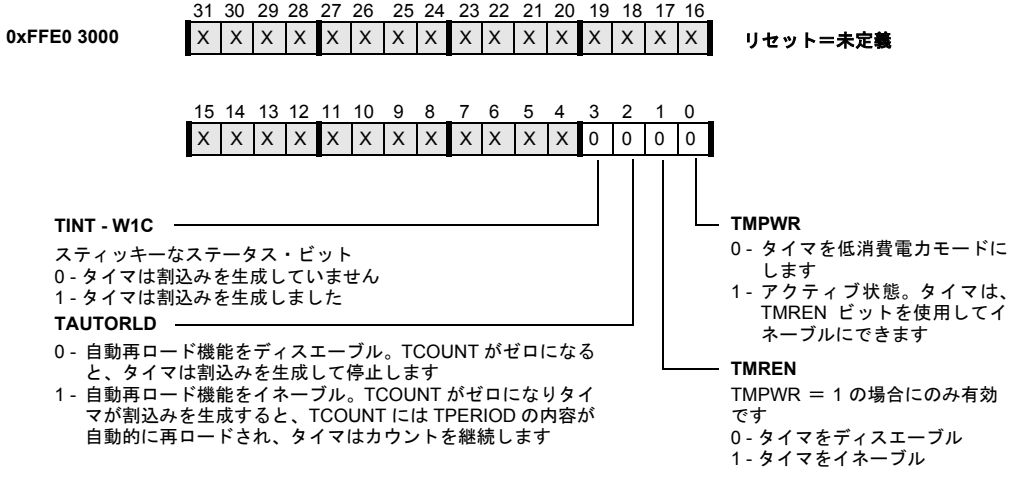


図 15-26.コア・タイマ・コントロール・レジスタ

■ TCOUNT レジスタ

コア・タイマ・カウント・レジスタ（TCOUNT）は、TSCALE＋1個のクロック・サイクルごとに1回デクリメントされます。TCOUNTの値が0になると割込みが生成され、TCNTLレジスタのTINTビットがセットされます。

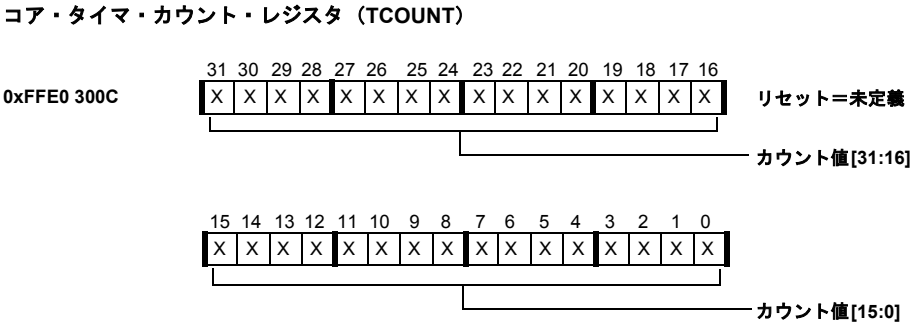


図 15-27.コア・タイマ・カウント・レジスタ

■ TPERIOD レジスタ

自動再ロードがイネーブルにされると、TCOUNTが0になるたびに、TCOUNTレジスタにはコア・タイマ周期レジスタ（TPERIOD）の値が再ロードされます。

TPERIODレジスタの値を確実なものにするために、どちらのレジスタへの最初の書き込みも双方のレジスタを初期化します。最初のカウント値に異なる値を望む場合は、データをTPERIODへ書き込んだ後にTCOUNTへ書き込みを行ってください。

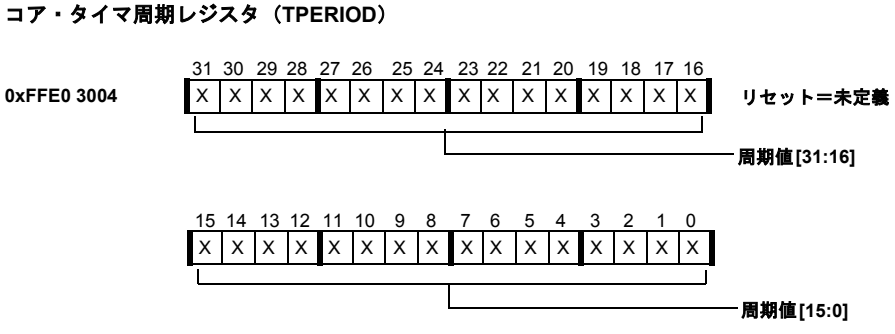


図 15-28.コア・タイマ周期レジスタ

■ TSCALE レジスタ

コア・タイマ・スケール・レジスタ（TSCALE）には、TCOUNTのデクリメント間隔となるサイクル数よりも1少ないスケーリング値が格納されます。たとえば、TSCALEレジスタの値が0の場合、カウンタ・レジスタは、クロック・サイクルごとに1回デクリメントされます。TSCALEが1の場合には、カウンタは2サイクルごとに1回デクリメントされます。

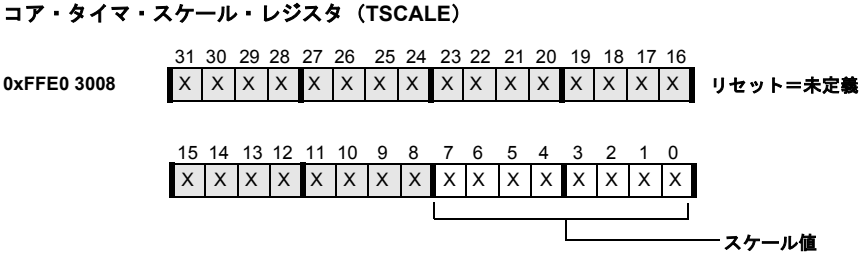


図 15-29.コア・タイマ・スケール・レジスタ

ウォッチドッグ・タイマ

このプロセッサには、ソフトウェア・ウォッチドッグ機能の実装に使用できる32ビットのタイマがあります。ソフトウェア・ウォッチドッグ機能では、タイマがソフトウェアによる更新前に切れた場合にプロセッサ・コアへのイベントを生成することによって、システムの信頼性を高めることができます。ウォッチドッグ・タイマのプログラム方法に応じて生成されるイベントは、リセット、マスク不可能割込み、または汎用割込みとすることができます。ウォッチドッグ・タイマは、システム・クロック（SCLK）によってクロック駆動されます。

■ ウォッチドッグ・タイマの動作

ウォッチドッグ・タイマを使用するには :

1. ウォッチドッグ・カウント・レジスタ（WDOG_CNT）にカウント値を書き込んで、ウォッチドッグ・タイマのカウント値を設定します。なお、ウォッチドッグ・タイマがイネーブルにされていない間に WDOG_CNT レジスタにロードしても、WDOG_STAT レジスタにプリロードされます。
2. ウォッチドッグ・コントロール・レジスタ（WDOG_CTL）では、タイムアウト時に生成されるイベントを選択します。
3. WDOG_CTL でウォッチドッグ・タイマをイネーブルにします。これによってウォッチドッグ・タイマはカウント・ダウンを開始し、WDOG_STAT レジスタの値をデクリメントします。WDOG_STAT が 0 になると、プログラムされたイベントが生成されます。イベントの生成を禁止するには、ソフトウェアは、WDOG_STAT に（任意の値で）書込みを実行して WDOG_CNT から WDOG_STAT にカウント値を再ロードするか、またはウォッチドッグ・タイマが切れる前に WDG_CTL でウォッチドッグ・タイマをディスエーブルにする必要があります。

■ WDOG_CNT レジスタ

ウォッチドッグ・カウント・レジスタ（WDOG_CNT）は、32 ビットの符号なしカウント値を保持します。WDOG_CNT レジスタには、32 ビットの読出し／書込みだけでアクセスする必要があります。

ウォッチドッグ・カウント・レジスタは、プログラマブルなカウント値を保持します。ウォッチドッグ・カウント・レジスタへの有効な書込みによっても、ウォッチドッグ・カウンタはプリロードされます。安全を考慮して、ウォッチドッグ・カウント・レジスタを更新できるのは、ウォッチ

ドッグ・タイマがディスエーブルにされているときだけです。タイマがイネーブルにされている間にウォッチドッグ・カウント・レジスタに書き込んでも、このレジスタの内容は変更されません。

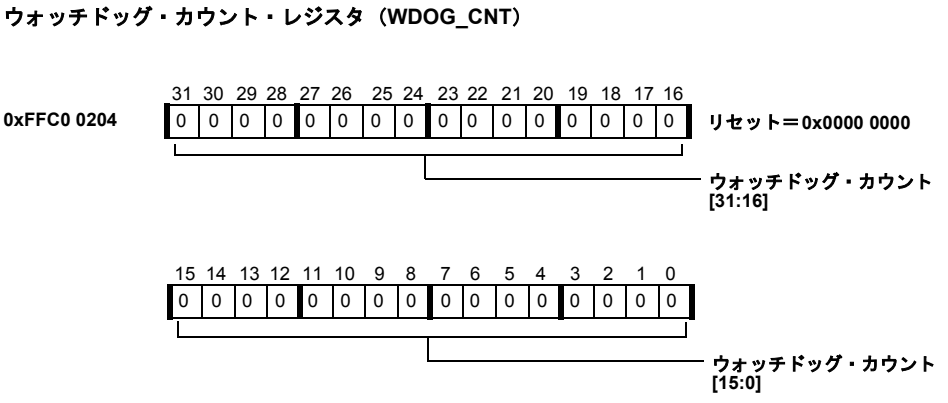


図 15-30.ウォッチドッグ・カウント・レジスタ

■ WDOG_STAT レジスタ

32ビットのウォッチドッグ・ステータス・レジスタ（WDOG_STAT）には、ウォッチドッグ・タイマの現在のカウント値が含まれています。WDOG_STATへの読出しによって、現在のカウント値が返されます。ウォッチドッグ・タイマがイネーブルにされると、WDOG_STATは、各SCLKサイクルで1だけデクリメントされます。WDOG_STATが0になると、ウォッチドッグ・タイマはカウントを停止し、ウォッチドッグ・コントロール・レジスタ（WDOG_CTL）で選択されたイベントが生成されます。

値は、WDOG_STATに直接格納することはできませんが、代わりにWDOG_CNTからコピーされます。これには、次の2つの動作があります。

- ウォッチドッグ・タイマがディスエーブルにされている場合、WDOG_CNTレジスタに書き込むとWDOG_STATレジスタにプリロードされます。

- ウォッチドッグ・タイマがイネーブルにされている場合、WDOG_STAT レジスタに書き込むと WDOG_CNT の値がロードされます。

プロセッサが WDOG_STAT に（任意の値の）書込みを実行すると、WDOG_CNT の値が WDOG_STAT にコピーされます。一般に、ソフトウェアは初期化時に WDOG_CNT の値を設定し、ウォッチドッグ・タイマが切れるまでに周期的に WDOG_STAT に書き込みます。これによって、ウォッチドッグ・タイマには WDOG_CNT からの値が再ロードされ、選択されたイベントの生成が防止されます。

WDOG_STAT レジスタは、32 ビット、符号なしのシステム・メモリマップド・レジスタであり、32 ビットの読出しと書込みでアクセスされる必要があります。

ユーザが、（SCLK * カウント）レジスタ・サイクルまでにカウンタを再ロードしない場合には、ウォッチドッグ割込みまたはリセットが生成され、ウォッチドッグ・コントロール・レジスタの TRO ビットがセットされます。その場合、カウンタはデクリメントを停止し、ゼロにとどまります。

カウンタにゼロをロードしてカウンタがイネーブルにされた場合には、ウォッチドッグ・コントロール・レジスタの TRO ビットがすぐにセットされ、カウンタはゼロにとどまり、デクリメントされません。

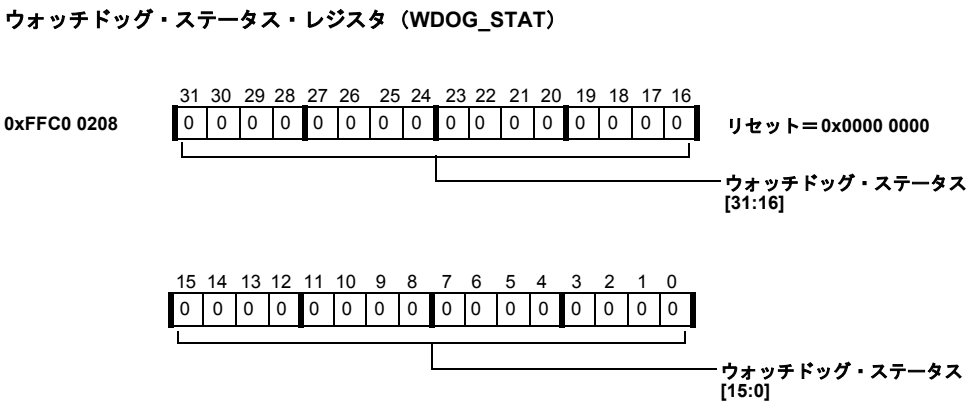


図 15-31.ウォッチドッグ・ステータス・レジスタ

■ WDOG_CTL レジスタ

ウォッチドッグ・コントロール・レジスタ（WDOG_CTL）は、ウォッチドッグ・タイマの制御に使用される、16ビットのシステム・メモリマップド・レジスタです。

ICTL[1:0]フィールドは、ウォッチドッグ・タイマが切れたときに生成されるイベントの選択に使用されます。なお、汎用割込みオプションが選択された場合には、システム割込みマスク・レジスタ（SIC_IMASK）は、その割込みをマスク解除するように適切に設定してください。ウォッチドッグ・イベントの生成がディスエーブルにされている場合には、ウォッチドッグ・タイマは前述のように動作します。ただし、ウォッチドッグ・タイマが切れたときにイベントは生成されません。

TMR_EN[7:0]フィールドは、ウォッチドッグ・タイマをイネーブル／ディスエーブルにするために使用されます。このフィールドにディスエーブル値以外の値を書き込むと、ウォッチドッグ・タイマがイネーブルにされます。このマルチビット・ディスエーブル・キーは、ウォッチドッグ・タイマが間違ってディスエーブルにされる可能性を最小限に抑えます。

タイマがロール・オーバーしたかどうかをソフトウェアで確認するには、ウォッチドッグ・コントロール・レジスタのTROステータス・ビットを調べます。これは、ウォッチドッグ・タイマ・カウントが0になるたびにセットされるスティッキー・ビットであり、ウォッチドッグ・タイマを無効にし、このビットに1を書き込むことによってのみクリアされます。

ウォッチドッグ・コントロール・レジスタ（WDOG_CTL）

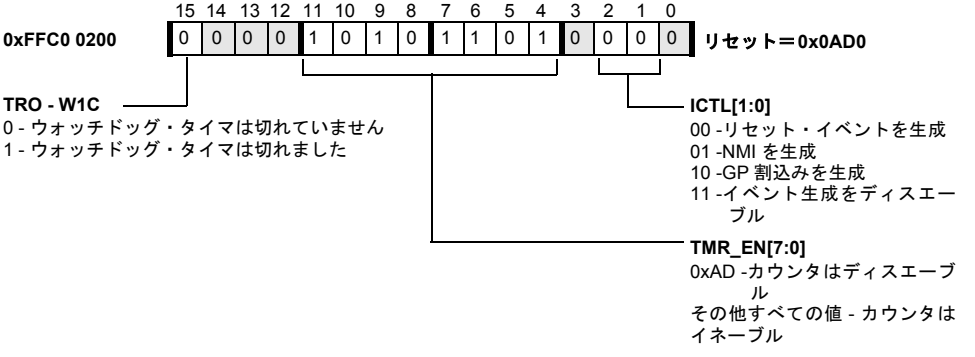


図 15-32.ウォッチドッグ・コントロール・レジスタ

なお、プロセッサがエミュレーション・モードにあるとき、ウォッチドッグ・タイマ・カウンタは、たとえイネーブルにされていてもデクリメントされません。

第16章　リアルタイム・クロック

リアルタイム・クロック（RTC）は、時刻、アラーム、ストップウォッチ・カウントダウンなど、一連のデジタル・ウォッチ機能をプロセッサに提供します。これは一般に、リアルタイム・ウォッチまたはライフ・カウンタの実装に使用されます。ライフ・カウンタでは、システムが最後にリセットされてからの経過時間をカウントします。

RTCウォッチ機能は、プロセッサの外部にある32.768kHz水晶発振器によってクロック駆動されます。RTCは専用の電源ピンを使用し、リセットには左右されません。このため、たとえ残りのプロセッサ部分がパワーダウンされた場合でも、RTCは機能を維持することができます。

RTC入力クロックは、バイパス可能なプリスケーラによって1Hz信号まで分周されます。バイパスされた場合、RTCは32.768kHzの水晶発振器周波数でクロック駆動されます。通常動作では、プリスケーラはイネーブルにされています。

RTCの主な機能は、正確な日数と時刻を維持することです。そのために、RTCでは4つのカウンタを使用します。

- 60秒カウンタ
- 60分カウンタ
- 24時間カウンタ
- 32768日カウンタ

RTCは、1秒間に1回、60秒カウンタをインクリメントします。他の3つのカウンタについては、適切な場合にインクリメントします。32768日カウンタは、毎日真夜中（0時0分0秒）にインクリメントされます。割込みは定期的に（毎秒、毎分、毎時、または毎日）発行できます。これらの割込みは、それぞれ独立して制御できます。

RTCは、RTCアラーム・レジスタ（RTC_ALARM）によってプログラムされた2つのアラーム機能を提供します。その1つは時刻アラームです（時、分、秒）。アラーム割込みがイネーブルの場合、RTCは毎日の指定された時間に割込みを生成します。もう1つのアラーム機能では、アプリケーションは時間だけでなく、日も指定できます。日アラーム割込みがイネーブルの場合、RTCは指定された日時に割込みを生成します。このアラーム割込みと日アラーム割込みは、独立してイネーブル／ディスエーブルにできます。

RTCは、カウントダウン・タイマとして機能するストップウォッチ機能を提供します。アプリケーションは、RTCストップウォッチ・カウント・レジスタ（RTC_SWCNT）に2番目のカウントをプログラムできます。ストップウォッチ割込みがイネーブルで、指定の秒数が経過したとき、RTCは割込みを生成します。

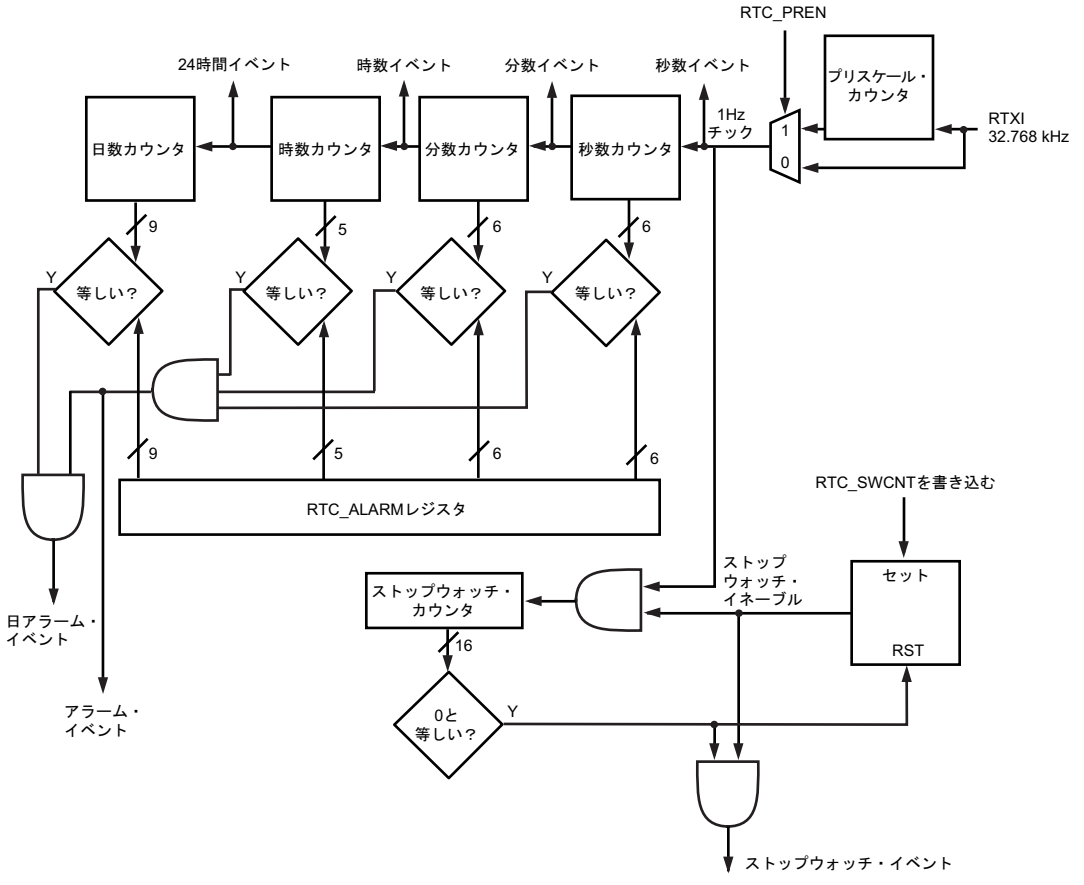


図 16-1. RTC ブロック図

インターフェース

RTCの外部インターフェースは2本のクロック・ピンで構成され、外付け部品と共にRTCの基準クロック回路を形成します。RTCは、ペリフェラル・アクセス・バス（PAB）を通じてプロセッサ・システムと内部的にインターフェースをとり、割込みインターフェースを通じてSIC（システム割込みコントローラ）にインターフェースをとります。

RTCには専用の電源ピンがあり、コア電源がオフにされた場合も含めて、クロック機能に電力を常時供給しています。

RTC クロックの条件

RTCタイマは、プロセッサの外部にある32.768kHz水晶発振器によってクロック駆動されます。RTCシステム・メモリマップド・レジスタ（MMR）は、この水晶発振器によってクロック駆動されます。プリスケーラがディスエーブルにされると、RTC MMRは32.768kHzの水晶発振器周波数でクロック駆動されます。プリスケーラがイネーブルにされると、RTC MMRは1Hzのレートでクロック駆動されます。

ソフトウェアからRTCカウンタをディスエーブルにする方法はありません。特定のシステムがRTC機能を必要としない場合には、ハードウェアのタイオフ（tie-off）によってディスエーブルにできます。RTXIピンをEGNDに接続（tie）し、RTCVDDピンをEVDDに接続し、RTXOピンを無接続のままにしておきます。さらに、RTC_PRENに0を書き込むと、若干の節電になります。

RTC プログラミング・モデル

RTCプログラミング・モデルは、一連のシステムMMRから構成されます。ソフトウェアはRTCを設定し、これらのレジスタへの読出し／書込みを通じてRTCのステータスを決定できます。RTC割込みコントロール・レジスタ（RTC_ICTL）とRTC割込みステータス・レジスタ（RTC_ISTAT）は、RTC割込み管理機能を提供します。

なお、ソフトウェアは、RTCのカウント機能をディスエーブルにできません。しかし、すべてのRTC割込みは、ディスエーブルにする（つまり、マスクする）ことができます。リセット時には、すべての割込みがディスエーブルにされます。RTCの状態は、システムMMRステータス・レジスタによっていつでも読み出すことができます。

16-3ページの図16-1に示す、主なリアルタイム・クロック機能は、独立したRTC V_{dd}電源によって電力を供給されるレジスタとカウンタから構成されます。このロジックはリセットされません。RTC V_{dd}が最初にパワーオンされたときには、このロジックは不明の状態で立ち上がります。

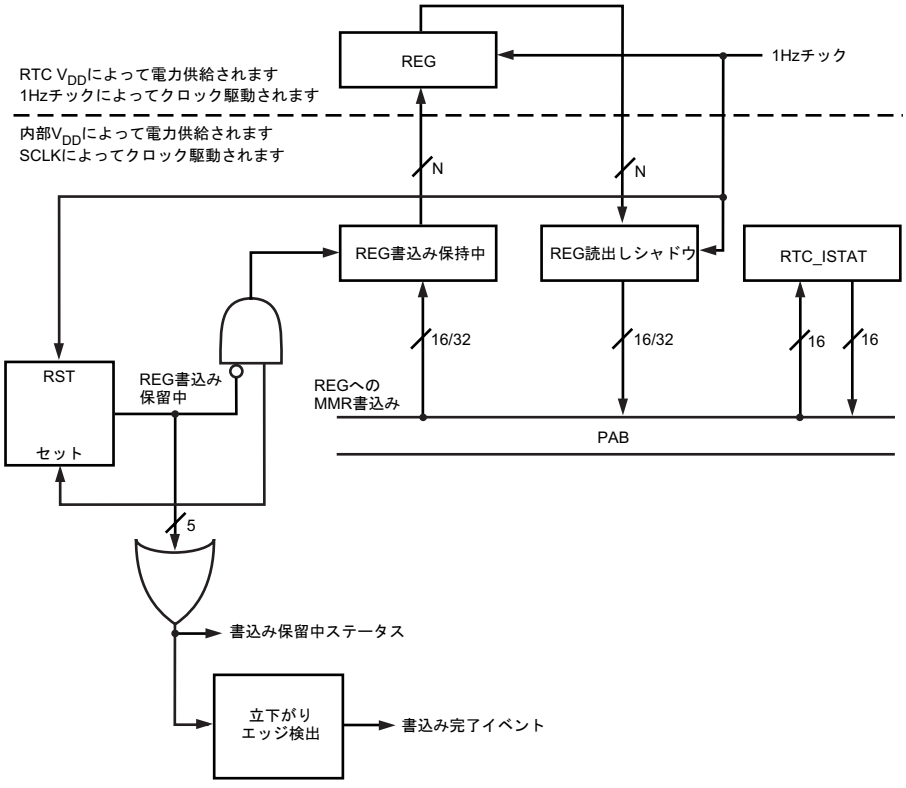


図 16-2. RTC レジスタのアーキテクチャ

RTCには、プロセッサ・コアやその他のペリフェラルと同じ内部V_{dd}によって電力供給されるロジックも含まれています。このロジックには、若干の制御機能、PAB書込みデータ用の保持レジスタ、RTC V_{dd}から電力供給される5本のレジスタのそれぞれに対してプリフェッチされるPAB読

出しデータ・シャドウ・レジスタが含まれています。このロジックは、他のペリフェラルと同じシステム・リセットによってリセットされ、同じSCLKによってクロック駆動されます。

16-5ページの図16-2は、RTC V_{dd}によって電力供給されるRTC MMR、それに対応して内部V_{dd}によって電力供給される書込み保持レジスタ、および読出しシャドウ・レジスタ間の接続を示します。この図で、「REG」は、RTC_STAT、RTC_ALARM、RTC_SWCNT、RTC_ICTL、RTC_PRENの各レジスタを意味します。RTC_ISTATレジスタは、PABにだけ接続されます。

1Hz RTCクロックの立上がりエッジは「1Hzチック」です。ソフトウェアは、秒数イベント・フラグのセットを待つか、秒数割込みを待つ（イネーブルの場合）ことによって、この1Hzチックに同期することができます。

■ レジスタの書込み

RTC割込みステータス・レジスタ（RTC_ISTAT）以外のすべてのRTC MMRへの書込みは、書込み保持レジスタに保存されてから、RTCの1Hzクロックに同期されます。RTC_ISTATの書込み保留中ステータス・ビットは、書込みの進捗を示します。書込み保留中ステータス・ビットは、書込みが開始されるとセットされ、すべての書込みが完了するとクリアされます。書込み保留中ステータス・ビットの立下がりエッジでは、RTC_ISTATの書込み完了フラグがセットされます。このフラグは、割込みを起こすようにRTC_ICTLで設定できます。ソフトウェアは、1つのRTC MMRへの書込み完了を待ってから別のRTC MMRに書き込む必要はありません。実行中の書込みがある場合には、書込み保留中ステータス・ビットがセットされます。書込み完了フラグがセットされるのは、すべての書込みが完了したときだけです。

ペリフェラルのリセット時に実行中の書込みはアボートされます。すべての RTC 書込みが完了するまでは、SCLK を停止させたり（ディープ・スリープ・モードに入ります）、内部 V_{dd} 電源を切断したりしないでください。

同じレジスタに書き込む際には、前の書込みの完了を待ってからにしてください。前の書込みが完了していない場合には、同じレジスタへの後からの書込みは無視されます。

書き込まれたレジスタを、書込み完了フラグがセットされる前に読み出すと、古い値が返されます。読出し／書込みを行う前に、書込み保留中ステータス・ビットを必ずチェックしてください。

■ 書込み遅延

RTC MMRへの書込みは、1HzのRTCクロックに同期されます。時刻を設定するときには、RTC MMRに書き込む際の遅延は考慮しないでください。リアルタイム・クロックを設定する最も正確な方法としては、秒数（1Hz）イベント・フラグを監視するか、またはこのイベントに対する割込みをプログラムしてから、割込みサービス・ルーチン（ISR）でRTCステータス・レジスタ（RTC_STAT）に現在の時刻を書き込みます。新しい値は、インクリメンタに先行して挿入されます。ハードウェアは書き込まれる値に1秒を加算し（数、時、日には適切な桁上げを行って）、インクリメントされた値を次の1Hzチックでロードして、その時点における時刻を表します。

任意の時点でポストされた書込みは、1Hzクロックに適切に同期されます。書込みは1Hzクロックの立上がりエッジで完了します。1Hzチックの直前にポストされた書込みは、1秒後の1Hzチックまでは完了しないことがあります。1Hzチック後の最初の990msでポストされた書込みは、次の1Hzチックで完了します。しかし最も簡単で、最も予測可能かつお勧めの方法としては、秒数割込みまたはイベントの直後にRTC_STAT、RTC_ALARM、RTC_SWCNT、RTC_ICTL、またはRTC_PRENへの書込みだけをポストします。これら5本のレジスタは、同じ秒で書込みできる場合もあります。

RTC_ISTATレジスタのW1Cビットは、すぐに有効になります。

■ レジスタの読出し

RTC MMRの読出しに際しては、値は読出しシャドウ・レジスタから得られるため、遅延はありません。これらのシャドウは、その秒に対するRTC割込みやイベント・フラグがアサートされるまでに、更新されて読出しの準備が行われます。SCLKの始動後に内部V_{dd}ロジックがその初期化シーケンスを完了すると、同期上の理由からRTC MMRの読出しが危険となる時点はありません。RTC MMRはコヒーレントな値を常に返しますが、それらの値は不明の場合もあります。

■ ディープ・スリープ

ダイナミック・パワー・マネジメント・コントローラ（DPMC）の状態がディープ・スリープである場合、システム内のすべてのクロック（RTXIとRTC 1Hzチックを除く）は停止します。この状態で、RTC V_{dd}カウンタはインクリメントを続行します。内部V_{dd}シャドウ・レジスタは更新されませんが、読み出すこともできません。

ディープ・スリープ状態では、RTC_ISTATの全ビットがクリアされます。ディープ・スリープ中に発生するイベントは、RTC_ISTATに記録されません。内部V_{dd} RTCコントロール・ロジックは、SCLKが再起動してから1RTXI周期（30.52µs）内に仮想1Hzチックを生成します。これによって、すべてのシャドウ・レジスタに最新の値がロードされ、秒数イベント・フラグがセットされます。他のイベント・フラグもセットされる場合があります。RTCイベントまたはハードウェア・リセットによって、システムがディープ・スリープ状態からウェイクアップすると、その秒（その秒のみ）の間に発生したすべてのRTCイベントは、RTC_ISTATで報告されます。

システムがディープ・スリープ状態からウェイクアップしたとき、ソフトウェアは、RTC_ISTATのビットをW1Cする必要はありません。すべてのW1Cビットは、すでにハードウェアによってクリアされています。RTCの内部V_{dd}ロジックがそのリスタート・シーケンスを完了すると、秒数イ

ベント・フラグがセットされます。ソフトウェアでは、秒数イベント・フラグがセットされるまで待ってから、RTCレジスタの読出し／書込みを始めてください。

■ プリスケーラのイネーブル

RTCプリスケーラ・イネーブル・レジスタ（RTC_PREN）の1つのアクティブ・ビットは、同期パスを使用して書き込まれます。ビットのクリアは、32.768kHzのクロックに同期します。この高速な同期によって、モジュールは、書込みの完了をまる1秒待つことなく、高速モードに入ることができます（プリスケーラをバイパス）。ただし、プリスケーラをイネーブルにして、モジュールがすでに実行中であった場合には、まる1秒の待機が必要になります。

RTC_PRENビットをセットすると、1Hzクロックの最初の立上がりエッジは、プリスケーラがイネーブルにされてから32.768kHzクロックの1～2サイクル後で発生します。プリスケール・カウンタをイネーブル／ディスエーブルにしても、書込み完了ステータス／割込みはいつものように機能します。新しいRTCクロック・レートは、書込み完了ステータスがセットされる前に有効になります。

■ イベント・フラグ

パワーアップ時のレジスタ内の値が不定なため、各レジスタに正しい値が書き込まれる前にイベント・フラグがセットされることがあります。1Hz クロック・エッジを使用しているため、RTC_STAT への書込みとRTC_ALARMへの書込みの間にまる1秒の開きが生じることがあります。リセット後の RTC_ALARM の値が RTC_STAT に書き込まれた値と同じである場合には、これによって、RTC_STAT と RTC_ALARM とが有効になるまで1秒の遅延が新たに生じることになります。これらのレジスタでの書込みが完了するのを待ってから、それらの値に関係付けられたフラグや割込みを使用します。

次に示すのは、フラグと、それらが有効になる条件の一覧です。

- 秒（1Hz）イベント・フラグ

  1Hzクロックの立上がりエッジと、シャドウ・レジスタがディープ・スリープからウェイクアップして更新された後に、常にセットされます。これは、RTC 1Hzクロックが動作している限り有効です。このフラグや割込みは、他のフラグを検証するために使用します。

- 書込み完了

  常に有効です。

- 書込み保留中ステータス

  常に有効です。

- 分イベント・フラグ

  RTC_STATの秒フィールドが有効になった後でのみ有効です。このフラグ値を使用したり割込みをイネーブルにしたりする前に、書込み完了と書込み保留中ステータスのフラグやそれらの割込みを使用してRTC_STAT値を検査します。

- 時イベント・フラグ

  RTC_STATの分フィールドが有効になった後でのみ有効です。このフラグ値を使用したり割込みをイネーブルにしたりする前に、書込み完了と書込み保留中ステータスのフラグやそれらの割込みを使用してRTC_STAT値を検証します。

- 24時間イベント・フラグ

  RTC_STATの時フィールドが有効になった後でのみ有効です。このフラグ値を使用したり割込みをイネーブルにしたりする前に、書込み完了と書込み保留中ステータスのフラグやそれらの割込みを使用してRTC_STAT値を検証します。

- ストップウォッチ・イベント・フラグ

  RTC_SWCNTレジスタが有効になった後でのみ有効です。このフラグ値を使用したり割込みをイネーブルにしたりする前に、書込み完了と書込み保留中ステータスのフラグやそれらの割込みを使用してRTC_SWCNT値を検証します。

- アラーム・イベント・フラグ

  RTC_STATレジスタとRTC_ALARMレジスタが有効になった後でのみ有効です。このフラグ値を使用したり割込みをイネーブルにしたりする前に、書込み完了と書込み保留中ステータスのフラグやそれらの割込みを使用して、RTC_STAT値とRTC_ALARM値を検証します。

- 日アラーム・イベント・フラグ

  アラームと同じです。

同じ秒の最初に一緒にポストされた書込みは、次の1Hzチックで一緒に有効になります。次のシーケンスは安全であり、前の状態からのスプリアス割込みにつながることはありません。

1. 1Hz チックだけ待ちます。
2. 1 を書き込んで、アラーム、日アラーム、ストップウォッチ、インターバルごとの RTC_ISTAT フラグをクリアします。
3. RTC_STAT、RTC_ALARM、RTC_SWCNT 用の新しい値を書き込みます。
4. アラーム、日アラーム、ストップウォッチ、インターバルごとの割込みをイネーブルにして、RTC_ICTL の新しい値を書き込みます。
5. 1Hz チックだけ待ちます。
6. これで新しい値は同時に有効になりました。

■ 割込み

RTCでは、以下を含めて、いくつかのプログラマブルなインターバルで割込みを提供できます。

- 毎秒
- 毎分
- 毎時
- 毎日
- プログラマブルな値からのカウントダウン時
- 毎日特定の時刻
- 特定の日時

RTCは、任意の1Hzレジスタ（RTC_STAT、RTC_ALARM、RTC_SWCNT、RTC_ICTL、RTC_PREN）へのすべての保留中書込みの完了時に割込みを提供するようにプログラムできます。割込みは、RTC割込みコントロール・レジスタ（RTC_ICTL）を使用して、個々にイネーブル／ディスエーブルできます。割込みステータスは、RTC割込みステータス・レジスタ（RTC_ISTAT）を読み出すことで判断できます。

RTC割込みは、RTC_ISTATレジスタにラッチされたイベントがRTC_ICTLレジスタでイネーブルにされるたびに設定されます。保留中のRTC割込みは、RTC_ISTAT内のイネーブルにされセットされたすべてのビットがクリアされるたびに、または保留中のイベントに対応するRTC_ICTL内のすべてのビットがクリアされるときにクリアされます。

16-14ページの図16-3に示すように、RTCは、スリープ状態からのウェイクアップとイベント処理用に、プロセッサ・コアへの割込み要求（IRQ）を生成します。RTCは、ディープ・スリープからのウェイクアップや内部V_{dd}のパワーオフ状態からのウェイクアップに対して、別個の信号を生成します。ディープ・スリープ・ウェイクアップ信号は、RTC_ICTLでイネーブルにされたRTCインターバル・イベントが発生したとき、1Hzチックでアサートされます。ディープ・スリープ・ウェイクアップ信号のアサーションによって、プロセッサのコア・クロック（CCLK）とシステム・クロック（SCLK）が再起動されます。イネーブルにされたイベントがRTCディープ・スリープ・ウェイクアップ信号をアサートする場合にも、SCLKが再起動されたらRTC IRQがアサートされます。

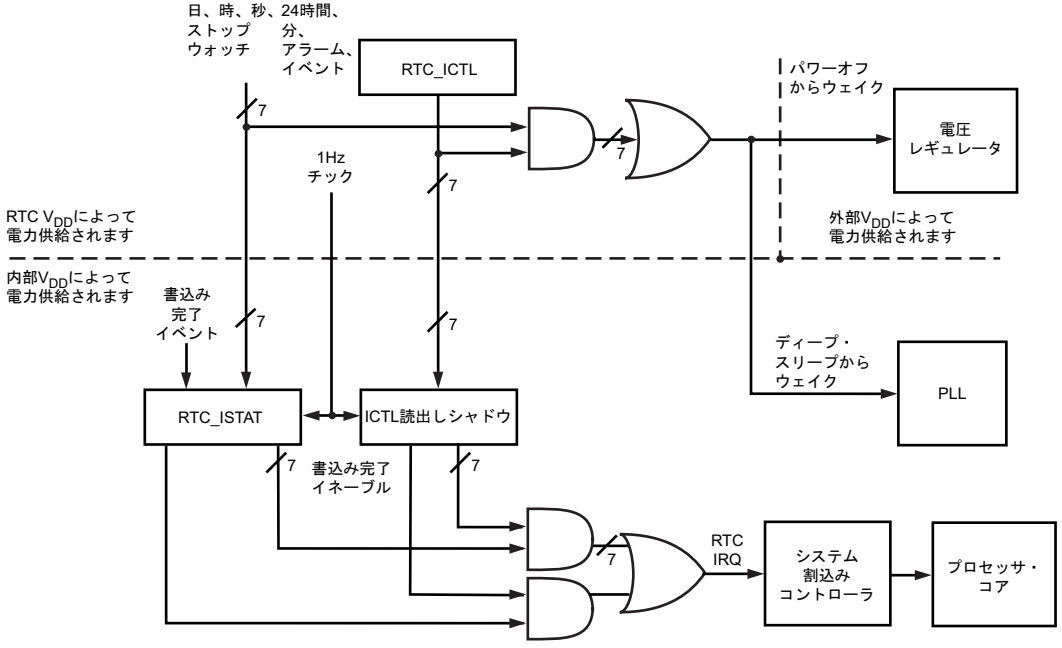


図 16-3. RTC の割込み構造

RTC_STAT レジスタ

RTC ステータス・レジスタ（RTC_STAT）は、現在の時刻を読出し／書込みするために使用されます。読出しによって返される 32 ビット値は、日、時、分、および秒カウンタの現在の状態を常に反映しています。読出し／書込みは、32 ビット・トランザクションであることが必要です。16 ビット・トランザクションの場合には、MMR エラーにつながります。読出しでは、一貫性のある 32 ビット値が常に返されます。時、分、および秒のフィールドは、一般に実際の時刻に合わせて設定されます。日カウンタの値は、毎日の真夜中にインクリメントされ、前回変更されて以降に経過し

た日数を記録します。この値は、特定の暦日には対応しません。15ビットの日カウンタは、オーバーフローするまでに89年と260日または261日（うるう年による）の範囲を提供します。

1Hzチック後に、RTC_STATに現在の時刻をプログラムします。次の1Hzチックで、RTC_STATは、新しい、インクリメントされた値を取得します。次に例を示します。

1. 1Hz チックだけ待ちます。
2. RTC_STAT を読み出し、10:45:30 を取得します。
3. RTC_STAT に現在の時刻、13:10:59 を書き込みます。
4. RTC_STAT を読み出すと、まだ古い時刻 10:45:30 を取得します。
5. 1Hz チックだけ待ちます。
6. RTC_STAT を読み出すと、新しい現在の時刻 13:11:00 を取得します。

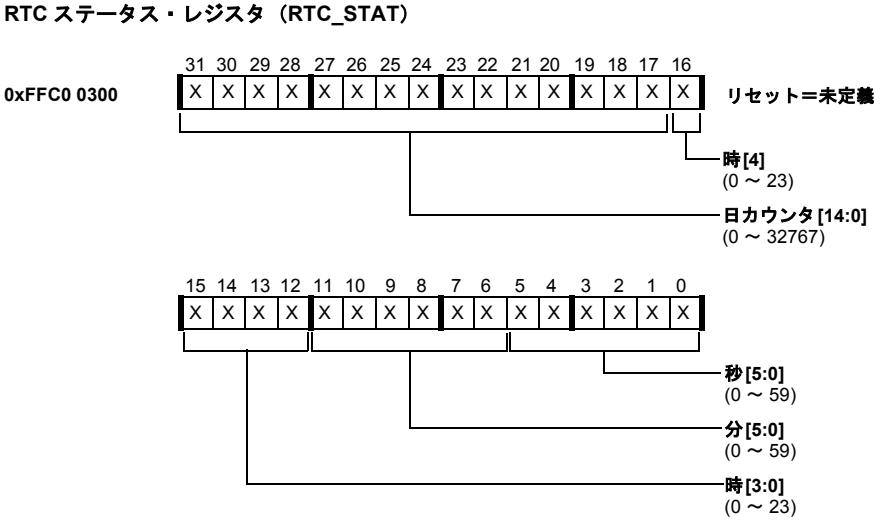


図 16-4. RTC ステータス・レジスタ

RTC_ICTL レジスタ

8 つの RTC 割込みイベントは、RTC 割込みコントロール・レジスタ（RTC_ICTL）によって、個々にマスクしたりイネーブルにしたりできます。秒割込みは、各 1Hz クロック・チックで生成されます（イネーブルの場合）。分割込みは、秒カウンタを 59 から 0 に進める 1Hz クロック・チックで生成されます。時割込みは、分カウンタを 59 から 0 に進める 1Hz クロック・チックで生成されます。24 時間割込みは、時刻を真夜中（00:00:00）に進める 1Hz クロック・チックで、24 時間に一度発生します。これらの割込みは、いずれもプロセッサへのウェイクアップ要求を生成できます（イネーブルの場合）。実装されたすべてのビットは、読出し／書込みです。

このレジスタは、リセット時に部分的にのみクリアされるため、イベントによっては初めからイネーブルであるように思われることがあります。しかし、PLL への RTC ウェイクアップと RTC 割込みは特殊な処理が行われ、RTC_ICTL レジスタへの最初の書込みが完了するまではマスクされます（ローに強制されます）。したがって、たとえ RTC_ICTL のいくつかのビットが非ゼロとして読み出されても、すべての割込みは、あたかもシステム・リセット時にディスエーブルにされた（RTC_ICTL の全ビットがゼロである）かのように振る舞います。リセットの直後に RTC 割込みが必要ない場合

には、後からの読出し—修正—書込みアクセスが意図したとおりに機能するように、RTC_ICTL に 0x0000 を書き込むことをお勧めします。

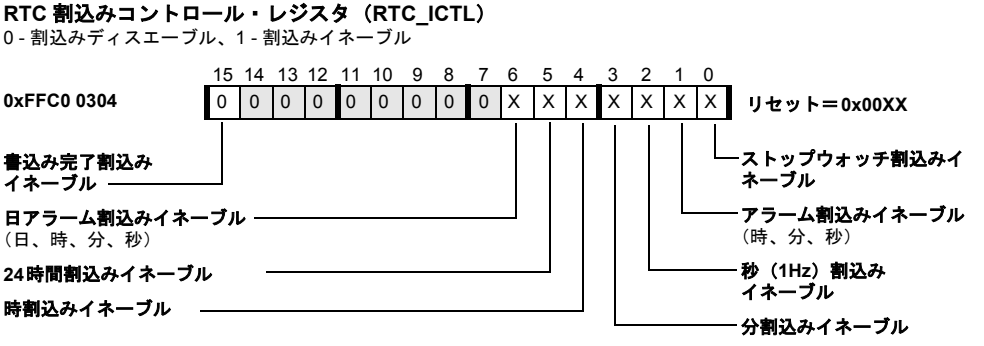


図 16-5. RTC 割込みコントロール・レジスタ

RTC_ISTAT レジスタ

RTC割込みステータス・レジスタ（RTC_ISTAT）は、すべてのRTC割込みのステータスを提供します。これらのビットはスティッキーです。各ビットは、対応するイベントによってセットされると、このレジスタへのソフトウェア書込みによってクリアされるまでセットされたままです。イベント・フラグは常にセットされており、RTC_ICTLの割込みイネーブル・ビットによってマスクされません。値は、それぞれのビット位置に1を書き込むことによってクリアされます。ただし、書込み保留中ステータス・ビッ

トは読出し専用です。レジスタのビットに0を書き込んでも、効果はありません。このレジスタは、リセット時とディープ・スリープ時にクリアされます。

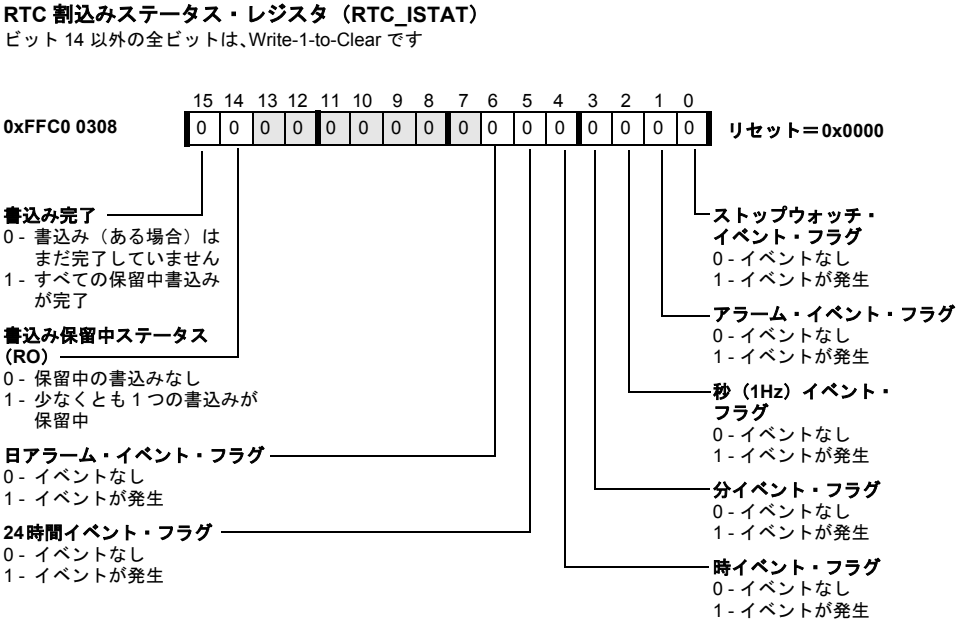


図 16-6. RTC 割込みステータス・レジスタ

RTC_SWCNT レジスタ

RTCストップウォッチ・カウント・レジスタ（RTC_SWCNT）には、ストップウォッチのカウントダウン値が含まれています。ストップウォッチは、プログラムされた値から秒数をカウント・ダウンし、カウントが0になると割込みを生成します（イネーブルの場合）。カウンタは、この時点でカウントを停止し、RTC_SWCNTに新しい値が書き込まれるまではカウントを再開しません。カウンタは、いったん動作すると、新しい値で上書きされ

ることがあります。これによって、ストップウォッチは、1秒の精度を持つウォッチドッグ・タイマとして使用できます。動作中のストップウォッチに0を書き込むと、ストップウォッチは停止し、早期に割込みが生じます。ストップウォッチ・イベント・フラグは、以下のいずれかの条件が発生したとき、1Hzチックでセットされます。

- ストップウォッチ・カウンタが0x0000までデクリメントする
- RTC_SWCNT への 0x0000 の書込みが完了し、ストップウォッチが動作していた（現在のストップウォッチ・カウントは0より大でした）
- RTC_SWCNT への 0x0000 の書込みが完了し、ストップウォッチが停止しました（現在のストップウォッチ・カウントは0でした）

レジスタには、0～（$2^{16}-1$）秒の任意の値をプログラムできます。これは18時間、12分、15秒の範囲です。

一般に、ソフトウェアは1Hzチックだけ待ってから、RTC_SWCNTを書き込みます。1秒後に、RTC_SWCNTは新しい値に変化し、デクリメントを開始します。レジスタの書込みではほぼ1秒を占有するので、値Nを書き込んでからストップウォッチ割込みまでの時間は約N + 1秒です。正確な遅延を生成するには、ソフトウェアはN – 1を書き込んでほぼN秒の遅延を得ることで補正できます。これは、ストップウォッチでは1秒の遅延を達成できないことを意味します。1Hzチックの直後に値1を書き込むと、ほぼ2秒後のストップウォッチ割込みにつながります。1秒待つには、ソフトウェアでは次の1Hzチックを待つことになります。

RTCストップウォッチ・カウント・レジスタはリセットされません。最初のパワーアップ後、このレジスタは動作していることがあります。ストップウォッチが使用されない場合、これに0を書き込んで停止させれば、若干の節電になります。

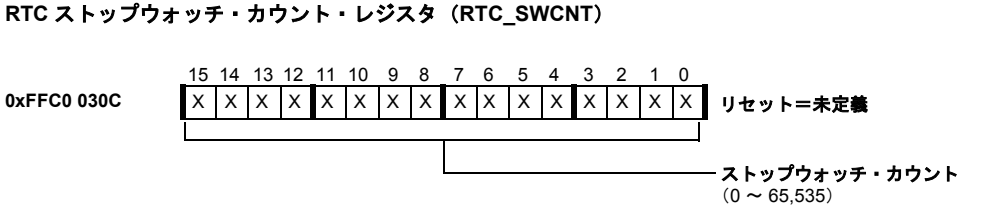


図 16-7. RTC ストップウォッチ・カウント・レジスタ

RTC_ALARM レジスタ

RTCアラーム・レジスタ（RTC_ALARM）は、アラーム割込みが発生する時刻（時、分、秒単位）として、ソフトウェアによってプログラムされます。読出し／書込みはいつでも行えます。アラーム割込みは、時、分、秒のフィールドがRTCステータス・レジスタのフィールドと最初に一致するたびに発生します。日割込みは、日、時、分、秒のフィールドがRTCステータス・レジスタのフィールドと最初に一致するたびに発生します。

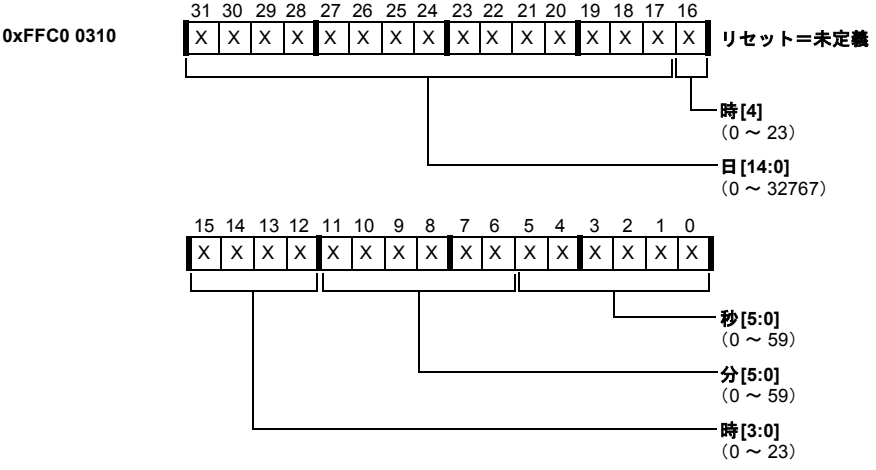


図 16-8. RTC アラーム・レジスタ

RTC_PREN レジスタ

RTCプリスケーラ・イネーブル・レジスタ（RTC_PREN）には、1つのアクティブ・ビットがあります。このビットがセットされると、プリスケーラがイネーブルにされ、RTCは1Hzの周波数で動作します。このビットがクリアされると、プリスケーラはディスエーブルにされ、RTC は32.768kHzの水晶発振器周波数で動作します。

RTCが適切なレートで動作するには、ソフトウェアは最初のパワーアップ後にプリスケーラ・イネーブル・ビットをセットする必要があります。RTC_PRENに書き込み、書込み完了イベントを待ってから、他のレジスタをプログラミングします。プロセッサがブートするたびに、RTC_PRENに1を書き込むと安全です。最初はビットがセットされますが、状態は変更されないため、それに続く書込みは効果がありません。

RTC_PREN のビットをクリアしてプリスケーラをディスエーブルにするには、前もって、RTC MMR への書込みが進行中でないことを必ず確認してください。通常動作時には、このビットのセットとクリアによって高速モードと低速モードを切り替えないでください。カウンタによるリアルタイムの正確なトラッキングが乱される原因となります。このような潜在的なエラーを回避するには、スタートアップ時に RTC_PREN によって RTC を初期化し、通常動作時にはプリスケーラの状態を動的に変更しないでください。

プリスケーラをイネーブルにしない状態での実行は、主にテスト・モードとして提供されています。すべての機能が32,768倍の速度で動作します。普通のソフトウェアでは、RTC_PRENに0をプログラムすることはありません。そうする唯一の理由は、1Hzチックは予想されるように、RTC_PRENの

0→1遷移の数RTXIサイクル後で発生するため、1Hzチックをより正確な外部イベントに同期することです。100µs以内の同期を達成するには、次のシーケンスに従ってください。

1. RTC_PREN に 0 を書き込みます。
2. 書込みの完了を待ちます。
3. 外部イベントを待ちます。
4. RTC_PREN に 1 を書き込みます。
5. 書込みの完了を待ちます。
6. RTC_STAT に時刻を再プログラムします。

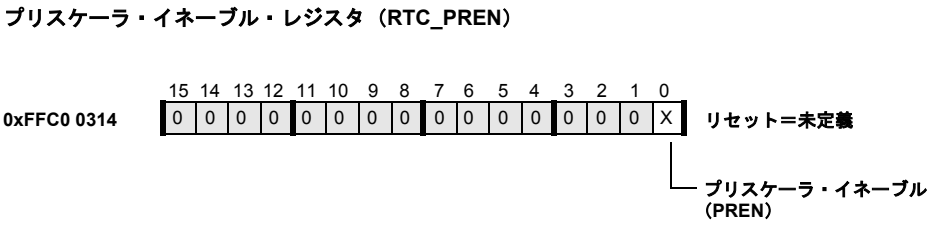


図 16-9. プリスケーラ・イネーブル・レジスタ

状態遷移のまとめ

表 16-1は、各RTC MMRがシステム状態によってどう影響を受けるかを示します。フェーズ・ロック・ループ（PLL）の状態（リセット、フル・オン、アクティブ、スリープ、ディープ・スリープ）は、第8章「ダイナミック・パワー・マネジメント」で定義されています。「無電源」とは、プロセッサのどの電源ピンもエネルギー源に接続されていないことを意味します。「オフ」とは、プロセッサ・コア、ペリフェラル、メモリが電力供給されていない（内部V_{dd}がオフになっている）ことを意味しますが、RTCは依然として電力供給されて動作しています。外部V_{dd}は依然として電力供給されていることがあります。「書込みどおり」と記述されたレジ

スタは、ソフトウェアがレジスタに書き込んだ最後の値を保持しています。RTC V_{dd}電源が印加されて以降、レジスタが書き込まれていない場合、状態は不明です（RTC_STAT、RTC_ALARM、RTC_SWCNTの全ビット、およびRTC_ISTAT、RTC_PREN、RTC_ICTLの一部のビットの場合）。

表 16-1.　RTC MMR に対する状態の影響

RTC V_{dd}	IV_{dd}	システム状態	RTC_ICTL	RTC_ISTAT	RTC_STAT RTC_SWCNT	RTC_ALARM RTC_PREN
オフ	オフ	無電源	X	X	X	X
オン	オン	リセット	書込みどおり	0	カウント	書込みどおり
オン	オン	フル・オン	書込みどおり	イベント	カウント	書込みどおり
オン	オン	スリープ	書込みどおり	イベント	カウント	書込みどおり
オン	オン	アクティブ	書込みどおり	イベント	カウント	書込みどおり
オン	オン	ディープ・スリープ	書込みどおり	0	カウント	書込みどおり
オン	オフ	オフ	書込みどおり	X	カウント	書込みどおり

表 16-2に、さまざまなシステム状態遷移イベントにおける、RTCに関するソフトウェアの責任を要約します。

表 16-2. RTC システム状態遷移イベント

以下のイベントで：	以下のシーケンスを実行：
無電源からのパワーオン	RTC_PREN = 1 を書き込みます。 書込み完了を待ちます。 RTC_STAT に現在の時刻を書き込みます。 必要ならば、RTC_ALARM に書き込みます。 RTC_SWCNT に書き込みます。 RTC_ISTATに書き込んで保留中のRTCイベントをクリアします。 RTC_ICTL に書き込んで、希望する RTC 割込みをイネーブルにするか、すべての RTC 割込みをディスエーブルにします。
リセット後のフル・オン または パワーオフからパワーオン後のフル・オン	秒イベントを待つか、RTC_PREN = 1 を書き込んで書込み完了を待ちます。 RTC_ISTATに書き込んで保留中のRTCイベントをクリアします。 RTC_ICTL に書き込んで、希望する RTC 割込みをイネーブルにするか、すべての RTC 割込みをディスエーブルにします。 必要に応じて RTC MMR を読み出します。
ディープ・スリープからのウェイク	秒イベント・フラグのセットを待ちます。 RTC_ISTAT に書き込んで RTC ディープ・スリープ・ウェイクアップをアクノレッジします。 必要に応じて RTC MMR を読み出します。 これで PLL 状態はアクティブです。必要に応じてフル・オンに遷移します。
スリープからのウェイク	RTC からのウェイクアップの場合、秒イベント・フラグがセットされます。この場合、RTC_ISTAT に書き込んで RTC ウェイクアップ IRQ をアクノレッジします。 必ず、必要に応じて RTC MMR を読み出します。
スリープになる前に	RTC によるウェイクアップが望ましい場合： 必要に応じて RTC_ALARM や RTC_SWCNT に書き込んで、ウェイクアップ・イベントをスケジュールします。 RTC_ICTL に書き込んで、ウェイクアップ用の希望する RTC 割込みソースをイネーブルにします。 書込み完了を待ちます。 システム割込みウェイクアップ・イネーブル・レジスタ（SIC_IWR）で、ウェイクアップ用の RTC をイネーブルにします。

表 16-2. RTC システム状態遷移イベント（続き）

以下のイベントで：	以下のシーケンスを実行：
ディープ・スリープになる前に	必要に応じて RTC_ALARM や RTC_SWCNT に書き込んで、ウェイクアップ・イベントをスケジュールします。 RTC_ICTL に書き込んで、ディープ・スリープ・ウェイクアップ用の希望する RTC イベント・ソースをイネーブルにします。 書込み完了を待ちます。
オフになる前に	必要に応じて RTC_ALARM や RTC_SWCNT に書き込んで、ウェイクアップ・イベントをスケジュールします。 RTC_ICTL に書き込んで、パワーアップ・ウェイクアップ用の希望する RTC イベント・ソースをイネーブルにします。 書込み完了を待ちます。 電圧レギュレータ・コントロール・レジスタ（VR_CTL）のウェイク・ビットをセットします。

第17章　外部バス・インターフェース・ユニット

外部バス・インターフェース・ユニット（EBIU）は、外部メモリへのグルーレス・インターフェースを提供します。このプロセッサは、シンクロナスDRAM（SDRAM）に対応しており、PC100とPC133のSDRAM規格に準拠します。EBIUでは、SRAM、ROM、FIFO、フラッシュ・メモリ、ASIC/FPGA設計などの非同期インターフェースも提供します。

概要

EBIUは、コアやDMAチャンネルからの外部メモリ要求を処理します。要求の優先順位は、外部バス・コントローラによって決定されます。要求がEBIU SDRAMコントローラによって処理されるか、EBIU非同期メモリ・コントローラによって処理されるかは、要求のアドレスによって決まります。

EBIUは、システム・クロック（SCLK）によってクロック駆動されます。プロセッサにインターフェースされたすべての同期メモリは、SCLK周波数で動作します。コア周波数とSCLK周波数との比率は、フェーズ・ロック・ループ（PLL）システム・メモリマップド・レジスタ（MMR）を使用してプログラムすることができます。詳細については、8-5ページの「コア・クロック／システム・クロックの比率制御」を参照してください。

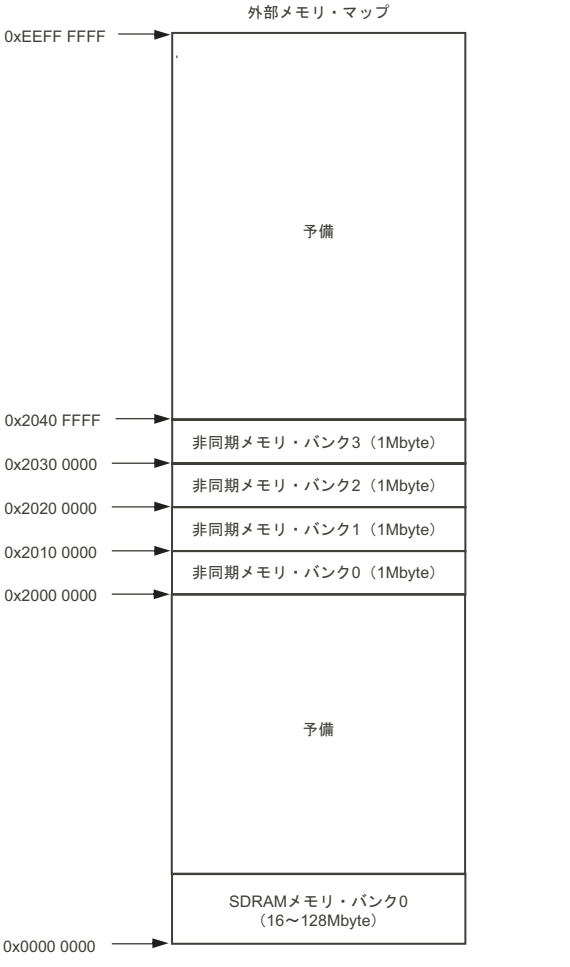
17-3ページの図 17-1は外部メモリ空間を示します。SDRAM専用にメモリ領域が1つ用意されています。SDRAMインターフェースのタイミングとSDRAM領域のサイズはプログラマムすることができます。SDRAMメモリ空間のサイズは16～128Mバイトの範囲にすることができます。

16Mバイト未満のSDRAMへの接続方法については、18-8ページの「16Mバイトよりも小さいSDRAMの利用」を参照してください。

SDRAMメモリ空間の開始アドレスは0x0000 0000です。SDRAMメモリ空間の最後からアドレス0x2000 0000までの領域は予備になっています。

次の4つの領域は、非同期メモリのサポート専用です。さまざまなメモリ・デバイスに対応するために、各非同期メモリ領域は個別にプログラムできます。各領域には、EBIUからの専用のメモリ・セレクト出力ピンがあります。

次の領域は、予備のメモリ空間です。この領域を参照しても、外部バス・トランザクションは生成されません。書込みは外部メモリ値に影響を与えず、読出しは未定義の値を返します。EBIUは、内部バス上にエラー応答を生成します。これによって、コア・アクセスに対するハードウェア例外が生成されたり、オプションではDMAチャンネルから割込みが生成されます。



注：上の図では、予備オフチップ・メモリ領域にラベルが付けてあります。その他すべてのオフチップ・システム・リソースは、コアとシステムの両方でアクセス可能です。

図 17-1. 外部メモリ・マップ

■ ブロック図

図 17-2は、EBIUとそのインターフェースの概念ブロック図を示します。上線付きで示される信号名はアクティブ・ロー信号です。

一度にアクセスできるのは1つの外部メモリ・デバイスに限られるため、メモリ・タイプごとのコントロール・ピン、アドレス・ピン、データ・ピンはこのデバイス側で多重化されています。非同期メモリ・コントローラ（AMC）とSDRAMコントローラ（SDC）は、共有されるピン・リソースを効果的に共用します。

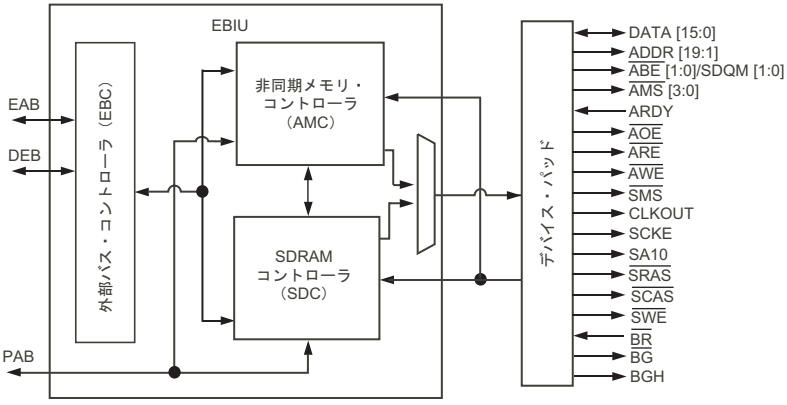


図 17-2. 外部バス・インターフェース・ユニット（EBIU）

■ 内部メモリ・インターフェース

EBIUは、プロセッサ内部にある3つのバスのスレーブとして機能します。

- 外部アクセス・バス（EAB）：コアからの外部バス要求のために、コア・メモリ・マネジメント・ユニットによって管理されます。
- DMA外部バス（DEB）：DMAチャンネルからの外部バス要求のために、DMAコントローラによって管理されます。
- ペリフェラル・アクセス・バス（PAB）：コアからのシステムMMR要求のために、コアによって管理されます。

これらはEBIUやパッド・レジスタと同様に、SCLKによってクロック駆動される同期インターフェースです。EABは、非同期外部メモリとシンクロナスDRAM外部メモリへのアクセスを提供します。外部アクセスは、EBIUのアクセスに使用される内部アドレスに応じて、非同期メモリ・コントローラ（AMC）またはSDRAMコントローラ（SDC）によって制御されます。AMCとSDCは外部ピンに対して同じインターフェースを共有するため、アクセスは直列化され、EABからの要求に基づいて調停する必要があります。

3番目のバス（PAB）は、EBIUのステータス・レジスタとメモリマップド・コントロール・レジスタへのアクセスにのみ使用されます。PABは、AMCとSDCに対して別個に接続され、EABバスとの調停は不要であり、EABバスからアクセス・サイクルを受け取る必要もありません。

外部バス・コントローラ（EBC）ロジックは、EABバスやDEBバスから来る外部メモリに対するアクセス要求を調停する必要があります。EBCロジックはバス選択に基づいて、読出し／書込み要求を適切なメモリ・コントローラに転送します。パッド・ロジックでは、AMCとSDCが共有リソースへのアクセスを求めて競合します。この競合は、EBCアービタによって規定された順序により、パイプライン方式で解決されます。多くの状況では、コアからのトランザクションはDMAアクセスよりも優先され

ます。しかし、DMAコントローラがトランザクションの過度のバックアップを検出した場合には、その優先順位を一時的にコアよりも上にするよう要求できます。

■ 外部メモリ・インターフェース

AMCとSDCでは、外部インターフェース・アドレス、データ・ピン、およびいくつかの制御信号を共有します。以下のピンは共有されます。

- ADDR[19:1]、アドレス・バス
- データ[15:0]、データ・バス
- $\overline{\text{ABE[1:0]}}$/SDQM[1:0]、AMCバイト・イネーブル／SDCデータ・マスク
- $\overline{\text{BR}}$、$\overline{\text{BG}}$、$\overline{\text{BGH}}$、外部バス・アクセス制御信号

他の信号は、2つのコントローラ間で多重化されません。

以下の表は、各インターフェースに関係付けられた信号を示します。

表 17-1. 非同期メモリ・インターフェース信号

パッド	ピン・タイプ[1]	説明
DATA[15:0]	I/O	外部データ・バス
ADDR[19:1]	O	外部アドレス・バス
$\overline{\text{AMS[3:0]}}$	O	非同期メモリ・セレクト
$\overline{\text{AWE}}$	O	非同期メモリ書込みイネーブル
$\overline{\text{ARE}}$	O	非同期メモリ読出しイネーブル

表 17-1. 非同期メモリ・インターフェース信号

パッド	ピン・タイプ[1]	説明
AOE	O	非同期メモリ出力イネーブル 多くの場合、AOE ピンは、外部メモリマップド非同期デバイスの OE ピンに接続してください。実際のシステムで使用するインターフェース信号を決定する、AOE 信号と ARE 信号との間のタイミング情報については、『ADSP-BF531/ADSP-BF532/ADSP-BF533 Embedded Processor Data Sheet』を参照してください。
ARDY	I	非同期メモリ・レディ応答 これは同期入力です
ABE[1:0]/SDQM[1:0]	O	バイト・イネーブル

1 ピン・タイプ：I＝入力、O＝出力

表 17-2. SDRAM インターフェース信号

パッド	ピン・タイプ[1]	説明
DATA[15:0]	I/O	外部データ・バス
ADDR[19:18], ADDR[16:1]	O	外部アドレス・バス SDRAM アドレス・ピンに接続します。バンク・アドレスは ADDR[19:18] で出力され、SDRAM BA[1:0] ピンに接続してください。
SRAS	O	SDRAM 行アドレス・ストローブ・ピン SDRAM の RAS ピンに接続します。
SCAS	O	SDRAM 列アドレス・ストローブ・ピン SDRAM の CAS ピンに接続します。
SWE	O	SDRAM 書込みイネーブル・ピン SDRAM の WE ピンに接続します。
ABE[1:0]/ SDQM[1:0]	O	SDRAM データ・マスク・ピン SDRAM の DQM ピンに接続します。
SMS	O	SDRAM 用に設定された外部メモリ・バンクのメモリ・セレクト・ピン SDRAM の CS（チップ・セレクト）ピンに接続します。アクティブ・ロー。

表 17-2. SDRAM インターフェース信号（続き）

パッド	ピン・タイプ[1]	説明
SA10	O	SDRAM A10 ピン AMC がバスを使用している間、SDRAM インターフェースは、このピンを使用してリフレッシュを可能にします。SDRAM の A[10] ピンに接続します。
SCKE	O	SDRAM クロック・イネーブル・ピン SDRAM の CKE ピンに接続します。
CLKOUT	O	SDRAM クロック出力ピン システム・クロック周波数で切り替えます。SDRAM の CLK ピンに接続します。

1　ピン・タイプ：I＝入力、O＝出力

■ EBIU プログラミング・モデル I

ここでは、EBIUのプログラミング・モデルについて説明します。このモデルは、EBIUのプログラムに使用される、システム・メモリマップド・レジスタをベースにしています。

EBIUには、6本のコントロール・レジスタと1本のステータス・レジスタがあります。

- 非同期メモリ・グローバル・コントロール・レジスタ（EBIU_AMGCTL）
- 非同期メモリ・バンク・コントロール0レジスタ（EBIU_AMBCTL0）
- 非同期メモリ・バンク・コントロール1レジスタ（EBIU_AMBCTL1）
- SDRAMメモリ・グローバル・コントロール・レジスタ（EBIU_SDGCTL）
- SDRAMメモリ・バンク・コントロール・レジスタ（EBIU_SDBCTL）
- SDRAM更新レート・コントロール・レジスタ（EBIU_SDRRC）
- SDRAMコントロール・ステータス・レジスタ（EBIU_SDSTAT）

これらのレジスタについては、この章のAMCとSDCの節で詳しく説明します。

■ エラー検出

0x0000 0000～0xEEFF FFFFという範囲をアドレス指定するバス動作については、たとえそのバス動作が予備メモリやディスエーブルされた機能をアドレス指定するものであっても、EBIUは応答します。応答に際して、EBIUはバス動作を完了（バス・マスタによって指定された適切な数のアクノレッジをアサート）して、以下のエラー条件に対するバス・エラー信号をアサートします。

- 予備のオフチップ・メモリ空間へのアクセス
- ディスエーブルにされた外部メモリ・バンクへのアクセス
- SDRAMメモリ・バンクの未実装領域へのアクセス

コアが誤ったバス動作を要求した場合、EBIUからのバス・エラー応答はコア内部のHWE割込みに割り当てられます（この割込みは、コアでマスクすることができます）。DMAマスターが誤ったバス動作を要求した場合には、バス・エラーはそのコントローラでキャプチャされ、オプションでコアへの割込みを生成できます。

非同期メモリ・インターフェース

非同期メモリ・インターフェースによって、さまざまなメモリ・タイプとペリフェラル・タイプに対するグルーレス・インターフェースが可能になります。これらには、SRAM、ROM、EPROM、フラッシュ・メモリ、FPGA/ASIC回路が含まれます。4つの非同期メモリ領域がサポートされます。表 17-3に示すように、それぞれに専用のメモリ・セレクト信号が関係付けられています。

表 17-3.　非同期メモリ・バンクのアドレス範囲

メモリ・バンク選択	開始アドレス	終了アドレス
$\overline{\text{AMS[3]}}$	2030 0000	203F FFFF
$\overline{\text{AMS[2]}}$	2020 0000	202F FFFF
$\overline{\text{AMS[1]}}$	2010 0000	201F FFFF
$\overline{\text{AMS[0]}}$	2000 0000	200F FFFF

■ 非同期メモリ・アドレスのデコード

各非同期メモリ・バンクに割り当てられるアドレス範囲は1Mバイトで固定されています。しかし、有効に設定したメモリ・バンク全体にメモリを配置する必要はありません。複数のメモリ・バンクにまたがる非常に大きなメモリ構造に対応する必要のあるSDRAMメモリとは異なり、ここに格納されるコードやデータは、比較的簡単にコードやデータの構造を利用できる非同期メモリ・バンクの1つに収まるよう制約されます。

なお、部分的に実装されたAMCバンクの未実装メモリへのアクセスは、バス・エラーにはつながらず、有効なAMCアドレスのエイリアスになります。

非同期メモリ信号は、17-6ページの表 17-1で定義されます。これらのピンのタイミングはプログラム可能であり、さまざまな速度のデバイスへの柔軟なインターフェースが可能です。インターフェース例については、第18章「システム設計」を参照してください。

■ EBIU_AMGCTL レジスタ

非同期メモリ・グローバル・コントロール・レジスタ（EBIU_AMGCTL）では、コントローラ全体の設定を行います。このレジスタには、バンク・イネーブルやその他の情報が含まれています。AMCが使用中である間は、このレジスタをプログラムしないでください。外部メモリマップド非同期

デバイスにアクセスするためにプロセッサを設定する際には、EBIU_AMGCTL レジスタは、他のコントロール・レジスタの後に書き込んでください。

非同期メモリ・グローバル・コントロール・レジスタ（EBIU_AMGCTL）

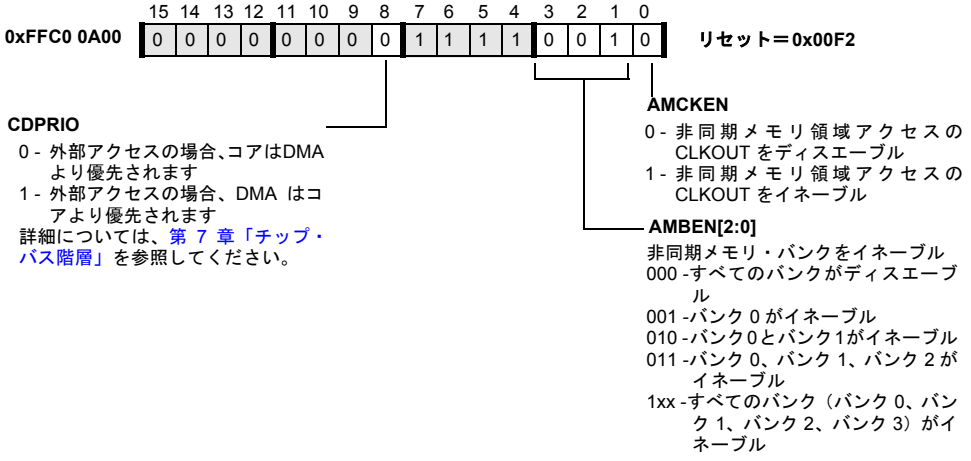


図 17-3. 非同期メモリ・グローバル・コントロール・レジスタ

ディスエーブルにされた非同期メモリ・バンクにバス動作がアクセスした場合には、EBIUは転送にアクノレッジを返し、要求側のバスでエラー信号をアサートして応答します。このエラー信号は、要求側のバス・マスタに送り返されます。これによって、コアが要求を出した転送の場合、コアへのハードウェア例外が生成されます。DMAが管理する要求の場合、エラーは各ステータス・レジスタで捕捉されます。バンクにメモリがフル実装されていない場合､おそらくメモリはバンク内の複数のアドレス領域にエイリアスされます｡EBIUはこのエイリアシング条件を検出しないため、エラー応答はアサートされません。

クロックを必要とする外部デバイスの場合､CLKOUT をイネーブルにするには、EBIU_AMGCTL レジスタのAMCKEN ビットをセットします。CLKOUT を使用しないシステムでは、AMCKEN ビットに0を設定します。

■ EBIU_AMBCTL0 レジスタと EBIU_AMBCTL1 レジスタ

EBIU非同期メモリ・コントローラには、2本の非同期メモリ・バンク・コントロール・レジスタ（EBIU_AMBCTL0とEBIU_AMBCTL1）があります。これらのレジスタには、セットアップ、ストローブ、ホールド・タイム用のカウンタのビット、メモリのタイプとサイズを決定するビット、およびARDYの使い方を設定するビットが含まれています。AMCが使用中である場合、これらのレジスタをプログラムしないでください。

AMCのタイミング特性は、以下の4つのパラメータを使用してプログラムできます。

- セットアップ：メモリ・サイクルの先頭（$\overline{\text{AMS[x]}}$ロー）と、読出しイネーブル・アサーション（$\overline{\text{ARE}}$ロー）または書込みイネーブル・アサーション（$\overline{\text{AWE}}$ロー）との間の時間
- 読出しアクセス：読出しイネーブル・アサーション（$\overline{\text{ARE}}$ロー）とアサート解除（$\overline{\text{ARE}}$ハイ）との間の時間
- 書込みアクセス：書込みイネーブル・アサーション（$\overline{\text{AWE}}$ロー）とアサート解除（$\overline{\text{AWE}}$ハイ）との間の時間
- ホールド：読出しイネーブル・アサート解除（$\overline{\text{ARE}}$ハイ）または書込みイネーブル・アサート解除（$\overline{\text{AWE}}$ハイ）と、メモリ・サイクルの最後（$\overline{\text{AMS[x]}}$ハイ）との間の時間

これらのパラメータは、それぞれEBIUクロック・サイクルを単位としてプログラムできます。さらに、これらのパラメータには最小値があります。

- セットアップ ≥ 1サイクル
- 読出しアクセス ≥ 1サイクル
- 書込みアクセス ≥ 1サイクル
- ホールド ≥ 0サイクル

非同期メモリ・バンク・コントロール 0 レジスタ（EBIU_AMBCTL0）

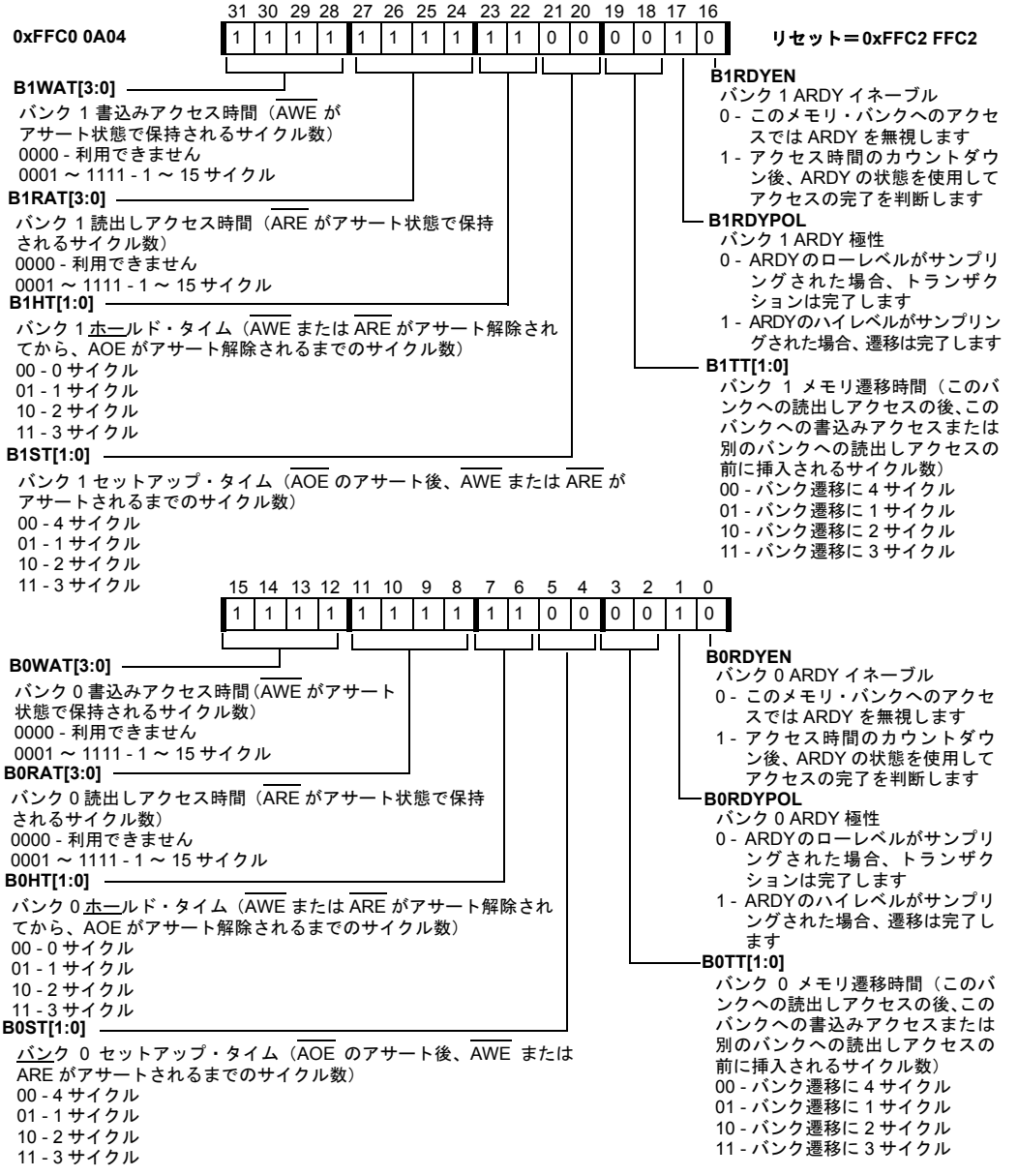


図 17-4. 非同期メモリ・バンク・コントロール 0 レジスタ

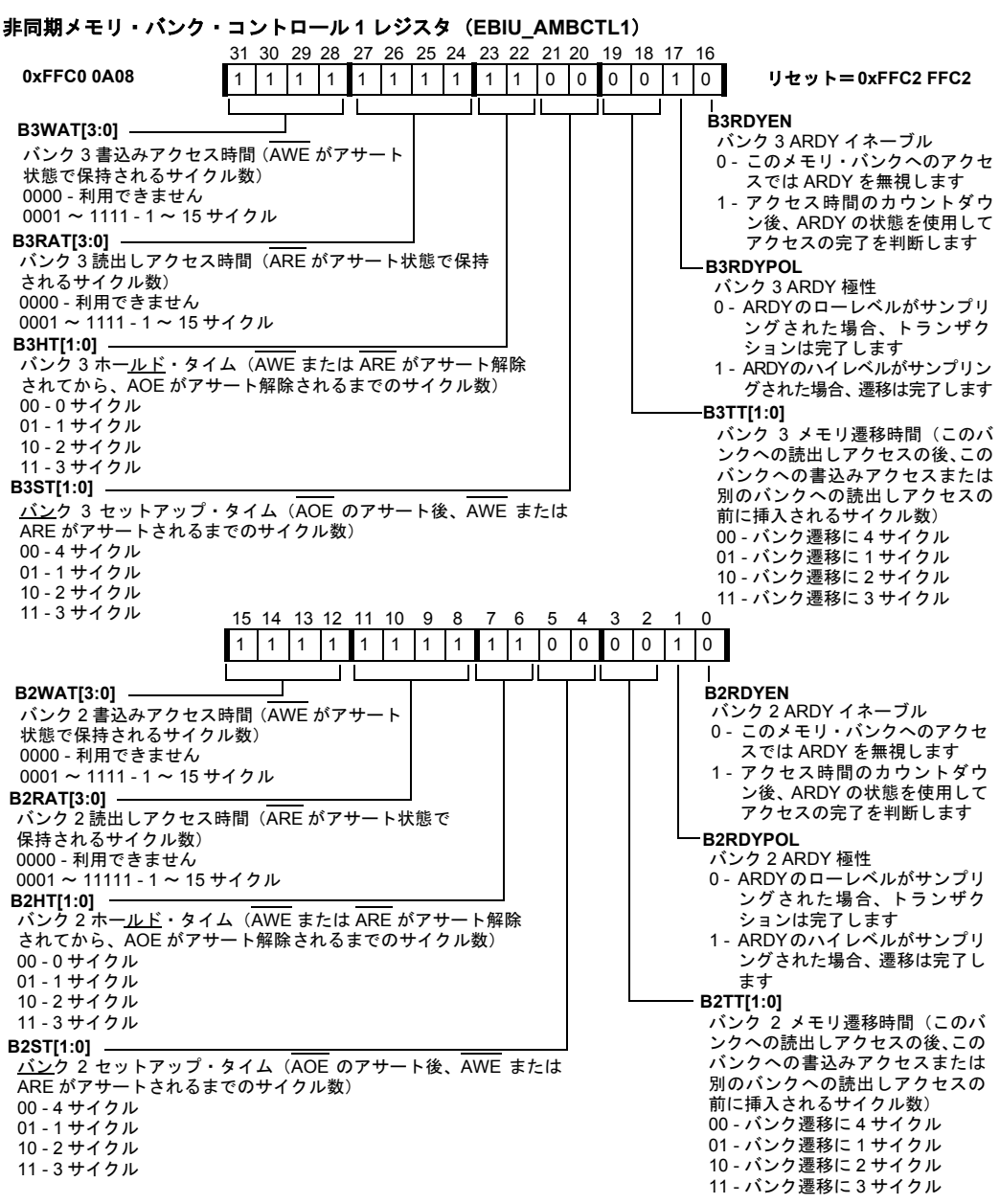


図 17-5. 非同期メモリ・バンク・コントロール 1 レジスタ

■ バス競合の回避

3ステートのデータ・バスは、システム内の複数のデバイスによって共有されるため、競合を回避するように注意してください。競合が生じると、消費電力が過大になり、デバイス障害につながることもあります。競合が発生するのは、1つのデバイスがバスを解放しようとし、もう1つのデバイスが確保しようとしている時です。最初のデバイスは3ステートになるのが遅く、2番目のデバイスはすばやく駆動される場合、これらのデバイスは競合します。

競合が起こり得るケースは2つあります。最初のケースは、読出しの後に同じメモリ空間への書込みが続く場合です。この場合、データ・バス・ドライバは、読出しによってアドレス指定されたメモリ・デバイスのドライバと競合する可能性があります。2番目のケースは、2つの異なるメモリ空間からのバック・ツー・バック読出しです。この場合、2つの読出しによってアドレス指定された2つのメモリ・デバイスは、2つの読出し動作間の遷移時に競合する可能性があります。

競合を回避するには、非同期メモリ・バンク・コントロール・レジスタで、ターンアラウンド時間（バンク遷移時間）を適切にプログラムしてください。この機能によって、ソフトウェアは、この種のアクセス間のクロック・サイクル数をバンクごとに設定できます。EBIUは、遷移の発生のために最小限1サイクルを提供します。

▶ ARDY 入力制御

各バンクは、読出し／書込みアクセス・タイマがカウントダウンした後でARDY入力をサンプリングするようにプログラムしたり、またはこの入力信号を無視するようにプログラムしたりできます。サンプル・ウィンドウでイネーブルおよびディスエーブルにされる場合には、ARDYを使用して、必要に応じてアクセス時間を拡張できます。なお、ARDYは同期してサンプリングされます。したがって、以下のことがいえます。

- プロセッサに対するARDYのアサーションとアサート解除は、データシートのセットアップ・タイムとホールド・タイムを満たす必要があります。これらの同期仕様を満たさない場合には、内部的に準安定性の動作をもたらすことがあります。ARDYの同期遷移を保証するには、プロセッサのCLKOUT信号を使用してください。
- ARDYピンは、内部バンク・カウンタが0になる前に、サイクル上の外部インターフェースで安定している（アサートまたはアサート解除される）必要があります。つまり、$\overline{\text{AWE}}$または$\overline{\text{ARE}}$のスケジュールされた立上がりエッジの前に、複数のCLKOUTサイクルが必要です。これによって、アクセスが拡張されるかどうか決定されます。
- ARDYのアサーションによってトランザクションが拡張されたら、そのトランザクションは、ARDYのアサートがサンプリングされた後のサイクルで完了します。

ARDYの極性は、バンクごとにプログラム可能です。ARDYは、ARDYイネーブルがアサートされているバンクに対してアクセスが始まるまではサンプリングされません。したがって、ARDYはデフォルトで駆動される必要はありません。詳細については、17-22ページの「ウェイト・ステートの追加」を参照してください。

■ プログラマブルなタイミング特性

ここでは、EBIUのプログラマブルなタイミング特性について説明します。タイミング関係は、AMCのプログラミング、コアからの開始であるかメモリDMA（MemDMA）からの開始であるか、およびトランザクションの順序（読出しの後に読出しが続く、読出しの後に書込みが続く、など）に依存します。

▶ コア命令による非同期アクセス

次のタイプのコア命令によって外部メモリ・アクセスが発生することがあります。

```
R0.L = W[P0++] ; /* 外部メモリからの読出し。ここで、P0は外部メモリ内の場所を指します */
```

または：

```
W[P0++] = R0.L ; /* 外部メモリへの書込み */
```

非同期読出し

図 17-6は、非同期読出しバス・サイクルを示します。タイミングは、セットアップ＝2サイクル、読出しアクセス＝2サイクル、ホールド＝1サイクル、遷移時間＝1サイクルでプログラムされています。

非同期読出しバス・サイクルは、次のように行われます。

1. セットアップ周期の初めに、$\overline{\text{AMS[x]}}$、アドレス・バス、$\overline{\text{ABE[1:0]}}$ が有効になり、$\overline{\text{AOE}}$ がアサートされます。
2. 読出しアクセス周期の初めで 2 セットアップ・サイクル後に、$\overline{\text{ARE}}$ がアサートされます。
3. ホールド周期の初めに、読出しデータは EBIU クロックの立上がりエッジでサンプリングされます。この立上がりエッジの後、$\overline{\text{ARE}}$ ピンはアサート解除されます。
4. ホールド周期の最後では、このバス・サイクルの後に同じにメモリ空間への別の非同期読出しが続く場合を除いて、$\overline{\text{AOE}}$ はアサート解除されます。また、次のサイクルが同じメモリ・バンクに対するものである場合を除いて、$\overline{\text{AMS[x]}}$ もアサート解除されます。
5. 同じメモリ・バンクの別の読出しが内部的にキュー登録される場合を除いて、AMC は、プログラムされたメモリ遷移時間サイクル数を付加します。

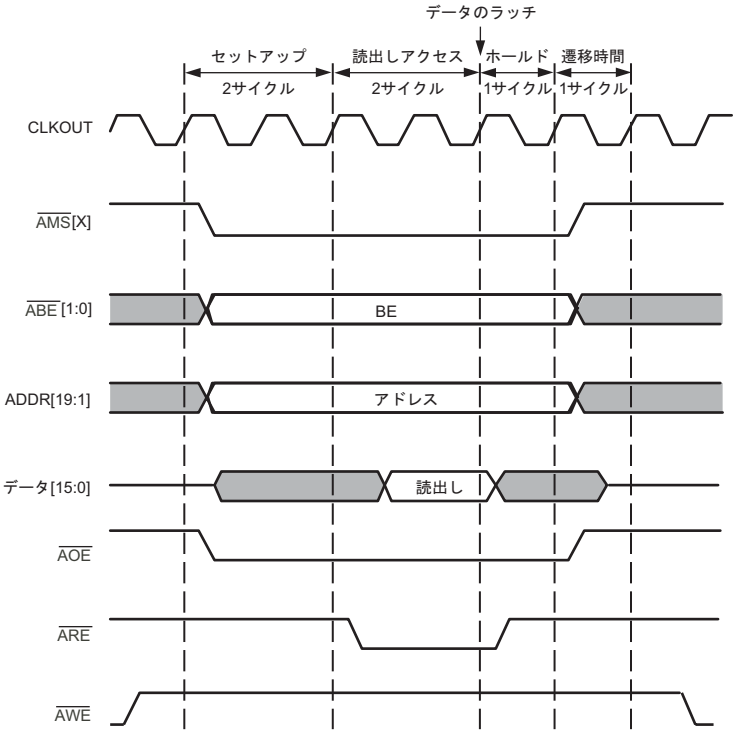


図 17-6. 非同期読出しバス・サイクル

非同期書込み

図 17-7は、非同期書込みバス・サイクルの後に同じバンクへの非同期読出しサイクルが続く場合を示します。タイミングは、セットアップ＝2サイクル、書込みアクセス＝2サイクル、読出しアクセス＝3サイクル、ホールド＝1サイクル、遷移時間＝1サイクルでプログラムされています。

非同期書込みバス・サイクルは次のように行われます。

1. セットアップ周期の初めに、 $\overline{\text{AMS[x]}}$、アドレス・バス、データ・バス、$\overline{\text{ABE[1:0]}}$ が有効になります。
2. 書込みアクセス周期の初めに、$\overline{\text{AWE}}$ がアサートされます。
3. ホールド周期の初めに、$\overline{\text{AWE}}$ がアサート解除されます。

非同期読出しバス・サイクルは次のように行われます。

1. セットアップ周期の初めに、$\overline{\text{AMS[x]}}$、アドレス・バス、$\overline{\text{ABE[1:0]}}$ が有効になり、$\overline{\text{AOE}}$ がアサートされます。
2. 読出しアクセス周期の初めに、$\overline{\text{ARE}}$ がアサートされます。
3. ホールド周期の初めに、読出しデータは EBIU クロックの立上がりエッジでサンプリングされます。$\overline{\text{ARE}}$ 信号は、この立上がりエッジの後でアサート解除されます。
4. ホールド周期の最後では、このバス・サイクルの後に同じメモリ空間への別の非同期読出しが続く場合を除いて、$\overline{\text{AOE}}$ がアサート解除されます。また、次のサイクルが同じメモリ・バンクに対するものである場合を除いて、$\overline{\text{AMS[x]}}$ もアサート解除されます。
5. 同じメモリ・バンクの別の読出しが内部的にキュー登録される場合を除いて、AMC はプログラムされたメモリ遷移時間サイクル数を付加します。

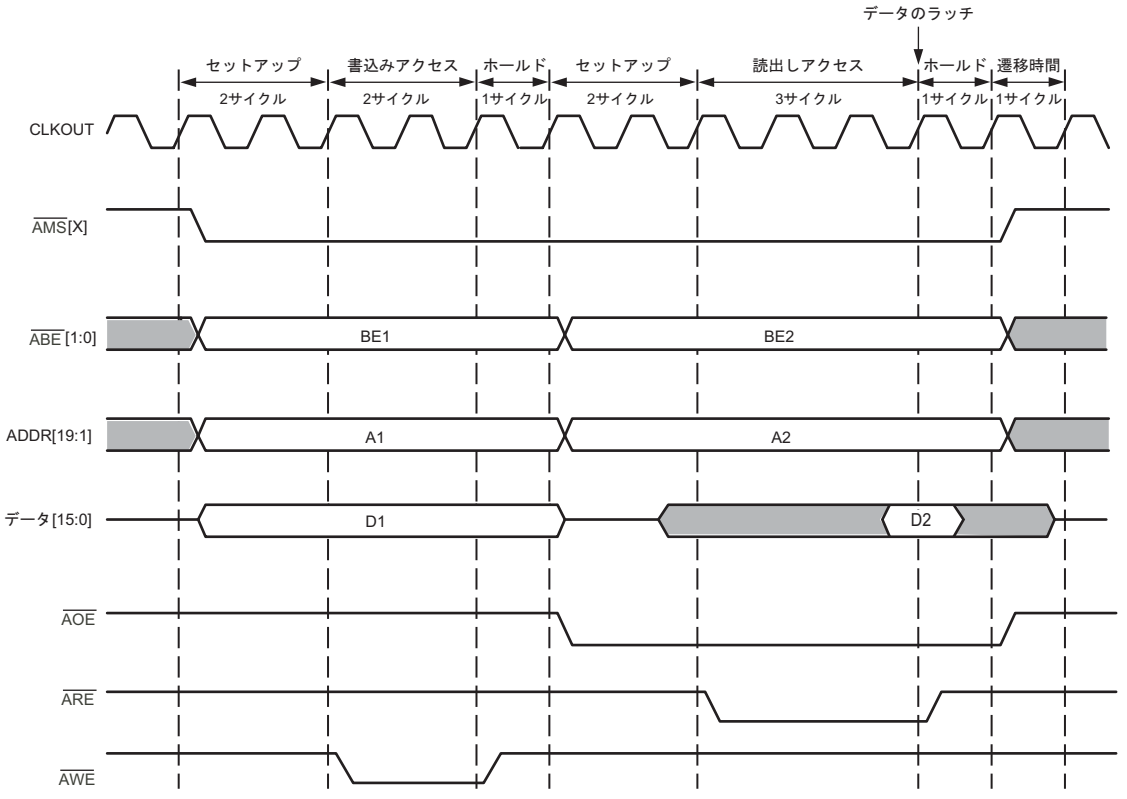


図 17-7. 非同期書込みと読出しのバス・サイクル

▶ ウェイト・ステートの追加

ARDY ピンは、ウェイト・ステートの挿入に使用されます。入力は、EBIU 内部クロックに同期してサンプリングされます。EBIU はプログラムされたストローブ周期の終わりより前のクロック・サイクルで、ARDY のサンプリングを開始します。ARDY のデアサートがサンプリングされた場合には、アクセス周期が拡張されます。これ以降、ARDY ピンは各クロック・エッジでサンプリングされます。読出しデータは、ARDY のアサートがサンプリングされた後のクロック・エッジでラッチされます。ARDY のアサートがサン

プリングされた後、読出しまたは書込みイネーブルは、1クロック・サイクルの間アサートに保たれます。この動作の一例を図 17-8に示します。ここでは、セットアップ＝2サイクル、読出しアクセス＝4サイクル、ホールド＝1サイクルです。なお、ARDY入力を利用するには、読出しアクセス周期は最小の2サイクルにプログラムする必要があります。

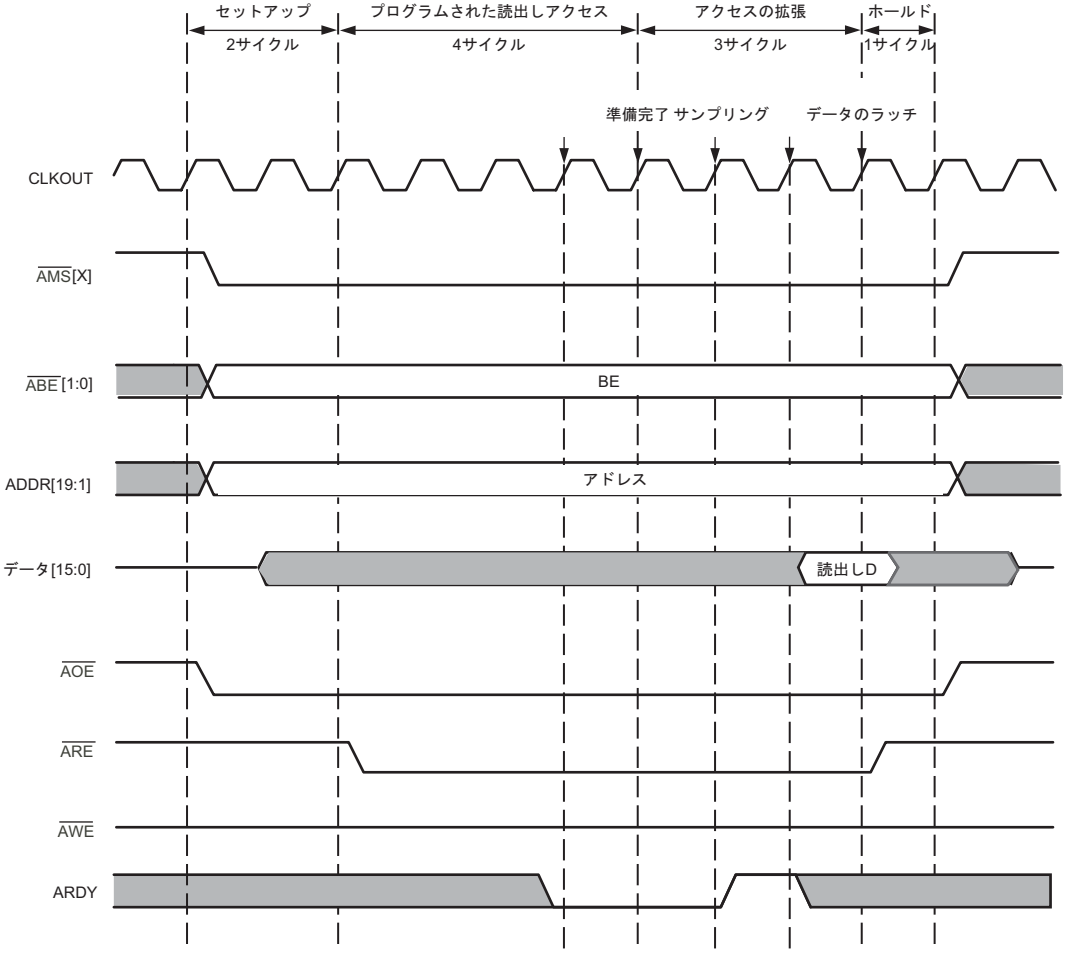


図 17-8. ARDY を使用した待ち状態の挿入

▶ バイト・イネーブル

$\overline{\text{ABE[1:0]}}$ピンは、すべての非同期読出しと16ビット非同期書込みの処理中、両方ともローレベルです。16ビット・メモリの上位バイトに非同期書込みが行われるとき、$\overline{\text{ABE1}}$＝0、$\overline{\text{ABE0}}$＝1です。16ビット・メモリの下位バイトに非同期書込みが行われるとき、$\overline{\text{ABE1}}$＝1、$\overline{\text{ABE0}}$＝0です。

SDRAM コントローラ（SDC）

SDRAMコントローラ（SDC）によって、プロセッサは、『ADSP-BF531/ADSP-BF532/ADSP-BF533 Embedded Processor Data Sheet』で指定された最大周波数で、シンクロナスDRAM（SDRAM）との間でデータを転送できます。プロセッサは、×4、×8、×16構成を含む64～512 Mビットの標準SDRAMから成る1つの外部バンクとのグルーレス・インターフェースを提供します。SDRAMの最大合計容量は128Mバイトです。このバンクは、SMSメモリ・セレクト・ピンによって制御されます。このインターフェースには、大きなメモリ・アレイの容量性負荷に対処するために、プロセッサとSDRAMとの間に追加バッファを提供するタイミング・オプションが備わっています。

SDRAMクロック出力CLKOUTの立上がりエッジで、すべての入力がサンプリングされ、すべての出力が有効です。

EBIU SDCは、標準のSDRAMとのグルーレス・インターフェースを提供します。SDRAMコントローラには以下の機能があります。

- ×4、×8、×16の設定で、64Mビット、128Mビット、256Mビット、512MビットのSDRAMを提供します。
- 外部SDRAMで128MバイトまでのSDRAMを提供します。
- 512バイト、1Kバイト、2Kバイト、4KバイトのSDRAMページ・サイズを提供します。

- SDRAM内の4つの内部バンクを提供します。
- プログラマブルなリフレッシュ・カウンタを使用して、SDRAMの必要な更新レートとさまざまなクロック周波数とを調整します。
- プロセッサとSDRAMとの間の追加バッファを提供するために、複数のタイミング・オプションを提供します。
- 独立したピン（SA10）を使用することで、非同期メモリ・コントローラがEBIUポートを制御している間に、SDCはSDRAMをプリチャージして、Auto-RefreshコマンドやSelf-Refreshコマンドを発行できます。
- 標準 SDRAM のセルフリフレッシュと、モバイル SDRAM のパーシャル・アレイ・セルフリフレッシュを提供します。
- 2つのSDRAMパワーアップ・オプションを提供します。
- インターリーブされたSDRAMバンク・アクセスを提供します。

■ 用語の定義

以下に、この章で使用する用語の定義を示します。

▶ Bank Activate コマンド

Bank Activateコマンドによって、SDRAMは（行アドレスによって指定される）行内に（バンク・アドレスによって指定される）内部バンクを開きます。SDRAMに対してBank Activateコマンドが発行されると、SDRAMは専用バンク内に新しい行アドレスを開きます。開いている内部バンクと行内のメモリは、開いているページと呼ばれます。Bank Activateコマンドは、読出しコマンドや書込みコマンドの前に適用する必要があります。

▶ バースト長

SDRAMデバイスが1つの書込みコマンドまたは読出しコマンドを検出した後、それぞれ格納または配信するワード数は、バースト長によって決まります。バースト長を選択するには、SDRAMのパワーアップ・シーケンス中に、SDRAMのモード・レジスタ内の特定ビットに書き込みます。

SDRAMへのバースト時には、SDCはバースト長＝1モードだけが可能ですが、SDCは、読出しまたは書込みコマンドをすべてのサイクルで適用し、データへのアクセスを続けます。したがって、実際のバースト長は1よりずっと大きくなります。つまり、バースト長＝1を設定しても、性能スループットは低下しません。

▶ Burst Stop コマンド

Burst Stopコマンドは、バースト読出し／書込みの動作を終了または中断するための方法の1つです。

SDRAM のバースト長は常に 1 になるよう固定されているため、SDCはBurst Stopコマンドに対応しません。

▶ バースト・タイプ

SDRAMが読出しコマンドを検出した後でバースト・データを配信するアドレス順、および書込みコマンドを検出した後でバースト・データを格納するアドレス順は、バースト・タイプによって決まります。バースト・タイプは、SDRAMのパワーアップ・シーケンス中にSDRAMでプログラムされます。

SDRAMのバースト長は常に1になるようプログラムされているため、バースト・タイプは重要ではありません。しかし、SDRAMのパワーアップ・シーケンス中には、SDCはバースト・タイプを常にsequential-accesses-only（順次アクセスのみ）に設定します。

▶ CAS 遅延（CL）

列アドレス・ストローブ（CAS）遅延は、SDRAMが読出しコマンドを検出してからデータをその出力ピンに提供するまでのクロック・サイクル単位での遅延です。CAS遅延は、パワーアップ・シーケンス中にSDRAMモード・レジスタでプログラムされます。

CAS遅延の値は、SDRAMのスピード・グレードとアプリケーションのクロック周波数によって決まります。SDCは、2または3クロック・サイクルのCAS遅延に対応できます。選択されたCAS遅延値は、SDRAMのパワーアップ・シーケンスよりも前に、SDRAMメモリ・グローバル・コントロール・レジスタ（EBIU_SDGCTL）にプログラムする必要があります。17-34ページの「EBIU_SDGCTLレジスタ」を参照してください。

▶ CBR（CAS Before RAS）リフレッシュまたは自動リフレッシュ

SDCリフレッシュ・カウンタがタイム・アウトすると、SDCはSDRAMの4つのバンクすべてにプリチャージしてから、それらのバンクにAuto-Refreshコマンドを発行します。これにより、SDRAMは内部CBRリフレッシュ・サイクルを生成します。内部リフレッシュが完了すると、4つの内部SDRAMバンクすべてがプリチャージされます。

▶ DQM データ I/O マスク機能

SDQM[1:0]ピンは、SDRAMへの8/16ビット書込みでのバイト・マスク機能を提供します。DQMピンは、プリチャージや特定の書込み動作中に、SDRAMの出力バッファをブロックするために使用されます。読出しサイクルでは、SDQM[1:0]ピンはデータのマスクに使用されません。

▶ 内部バンク

SDRAMには複数の内部メモリ・バンクがあります。SDCは、内部バンク間のインターリーブされたアクセスを提供します。バンク・アドレスは、行アドレスの一部と考えることができます。SDCでは、インターフェー

スをとるすべてのSDRAMには4つの内部バンクがあると想定し、アクティブにされた各バンクは、ユニークな行アドレスを持つことができます。

▶ モード・レジスタ

SDRAMデバイスに含まれる内部設定レジスタによって、SDRAMデバイスの機能を指定できます。パワーアップ後、SDRAMメモリ空間に読出しや書込みを実行する前に、アプリケーションはSDCをトリガして、SDRAMのモード・レジスタを書き込む必要があります。SDRAMのモード・レジスタの書込みをトリガするには、SDRAMメモリ・グローバル・コントロール・レジスタ（EBIU_SDGCTL）のPSSEビットに1を書き込んでから、SDRAMアドレス空間への読出しまたは書込み転送を発行します。最初の読出しまたは書込みによって、SDRAMパワーアップ・シーケンスの実行がトリガされ、SDRAMのモード・レジスタには、EBIU_SDGCTLレジスタからのCAS遅延がプログラムされます。SDRAMに対するこの最初の読出しまたは書込みは、完了に多くのサイクルを必要とします。

大部分のアプリケーションで、SDRAMパワーアップ・シーケンスとモード・レジスタの書込みは一度だけ行う必要があります。パワーアップ・シーケンスが完了したら、モード・レジスタへの変更が必要な場合を除いて、PSSEビットを再びセットすべきではありません。この場合には、18-10ページの「PLL遷移時におけるSDRAMのリフレッシュ管理」を参照してください。

低消費電力SDRAMデバイスにも、拡張モード・レジスタが含まれている場合があります。EBIUでは、EBIU_SDGCTLレジスタのEMRENビットによって、パワーアップ時に拡張モード・レジスタのプログラミングが可能です。

▶ ページ・サイズ

ページ・サイズとは、同じ行アドレスを持ち、別の行をアクティブにすることなく、連続した読出しまたは書込みコマンドでアクセスできるメモリ量です。16ビットのSDRAMバンクの場合、ページ・サイズを計算するには、次の式を使用します。

- 16ビットのSDRAMバンク：ページ・サイズ＝$2^{(CAW + 1)}$

ここで、CAWはSDRAMの列アドレス幅に1を足した値です。なぜなら、SDRAMバンクは16ビット幅であるからです（1アドレス・ビット＝2バイト）。

▶ Precharge コマンド

Prechargeコマンドは、アクティブ・ページ内の特定の内部バンク、またはページ内のすべての内部バンクを閉じます。

▶ SDRAM バンク

SDRAMバンクは、16Mバイト、32Mバイト、64Mバイト、または128Mバイトに設定することができ、SMSピンによって選択されるメモリ領域です。

「SDRAM内部バンク」はSDRAMの内部にあり、バンク・アドレスによって選択されます。一方、「SDRAMバンク」つまり「外部バンク」は、SMSピンによってイネーブルにされます。両者を混同しないでください。

▶ Self-Refresh

SDRAMがSelf-Refreshモードにある場合、SDRAMの内部タイマは、外部の制御入力なしでAuto-Refreshサイクルを定期的に開始します。SDCは、SDRAMをこの低消費電力モードにするために、Self-Refreshコマンドを含む一連のコマンドを発行する必要があり、Self-Refreshモードを終

了するために、別の一連のコマンドを発行する必要があります。Self-Refreshモードへの切り替えはSDRAMメモリ・グローバル・コントロール・レジスタ（EBIU_SDGCTL）で設定可能であり、SDRAMアドレス空間へのアクセスによって、SDCはSDRAMをSelf-Refreshモードから終了させます。17-40ページの「Self-Refreshモードへの出入り（SRFS）」を参照してください。

▶ t_{RAS}

Bank Activateコマンドを発行してからPrechargeコマンドを発行するまで、およびSelf-Refreshコマンドと Self-Refreshの終了との間に必要な遅延です。SDRAMメモリ・グローバル・コントロール・レジスタ（EBIU_SDGCTL）のTRASビット・フィールドは4ビット幅で、1～15クロック・サイクル長にプログラムできます。17-43ページの「Bank Activateコマンド遅延の選択（TRAS）」を参照してください。

▶ t_{RC}

同じSDRAM内部バンクに対して、Bank Activateコマンドを連続して発行するために必要な遅延です。この遅延は、直接にプログラムできません。t_{RC}遅延は、TRASとTRPのフィールドをプログラミングすることによって $t_{RAS} + t_{RP} \geq t_{RC}$になるようにする必要があります。

▶ t_{RCD}

Bank Activateコマンドを発行してから、最初のReadまたはWriteコマンドの開始までに必要な遅延です。SDRAMメモリ・グローバル・コントロール・レジスタ（EBIU_SDGCTL）のTRCDビット・フィールドは3ビット幅で、1～7クロック・サイクル長になるようにプログラムできます。

▶ t_{RFC}

Auto-Refreshコマンドを発行してからBank Activateコマンドを発行するまでと、Auto-Refreshコマンドを連続して発行するために必要な遅延です。この遅延は直接にプログラムされず、t_{RC}に等しいと考えられます。t_{RC}遅延は、TRASとTRPのフィールドをプログラミングすることによって$t_{RAS} + t_{RP} \geq t_{RC}$になるようにする必要があります。

▶ t_{RP}

Prechargeコマンドを発行してから、以下の操作までに必要な遅延です。

- Bank Activateコマンドの発行
- Auto-Refreshコマンドの発行
- Self-Refreshコマンドの発行

SDRAMメモリ・グローバル・コントロール・レジスタ（EBIU_SDGCTL）のTRPビット・フィールドは3ビット幅で、1～7クロック・サイクル長になるようにプログラムできます。17-44ページの「プリチャージ遅延の選択（TRP）」を参照してください。

▶ t_{RRD}

Bank A Activateコマンドの発行とBank B Activateコマンドの発行との間に必要な遅延です。この遅延は直接のプログラムができず、t_{RCD}＋1とされています。

▶ t_{WR}

Writeコマンド（書込みデータの駆動）とPrechargeコマンドとの間に必要な遅延です。SDRAMメモリ・グローバル・コントロール・レジスタ（EBIU_SDGCTL）のTWRビット・フィールドは2ビット幅で、1～3クロック・サイクル長になるようにプログラムできます。

▶ t_{XSR}

Self-Refreshモードを終了してからAuto-Refreshコマンドを発行するまでに必要な遅延です。この遅延は直接にプログラムされず、t_{RC}に等しいと考えられます。t_{RC}遅延は、TRASとTRPのフィールドをプログラミングすることによって$t_{RAS} + t_{RP} \geq t_{RC}$になるようにする必要があります。

■ 利用可能な SDRAM 設定

次の表は、可能なすべてのバンク・サイズ、バンク幅、SDCにグルーレスにインターフェースできる個別SDRAM部品の設定を示します。

表 17-4.　利用可能な SDRAM ディスクリート部品設定

バンク・サイズ（M バイト）	バンク幅（ビット）	SDRAM		
		サイズ（M ビット）	設定	チップ数
16	16	32	2M × 4 × 4 バンク	4
16	16	64	2M × 8 × 4 バンク	2
16	16	128	2M × 16 × 4 バンク	1
32	16	64	4M × 4 × 4 バンク	4
32	16	128	4M × 8 × 4 バンク	2
32	16	256	4M × 16 × 4 バンク	1
64	16	128	8M × 4 × 4 バンク	4
64	16	256	8M × 8 × 4 バンク	2
64	16	512	8M × 16 × 4 バンク	1
128	16	256	16M × 4 × 4 バンク	4
128	16	512	16M × 8 × 4 バンク	2
128	16	1024	16M × 16 × 4 バンク	1

■ SDRAM システムのブロック図の例

図 17-9は、SDRAMインターフェースのブロック図を示します。この例で、SDRAMインターフェースは2つの64Mビット（x8）SDRAMデバイスに接続されて、16Mバイトのメモリから成る1つの外部バンクを形成します。2つのSDRAMデバイスは、同一のアドレスおよび制御バスに接続されます。

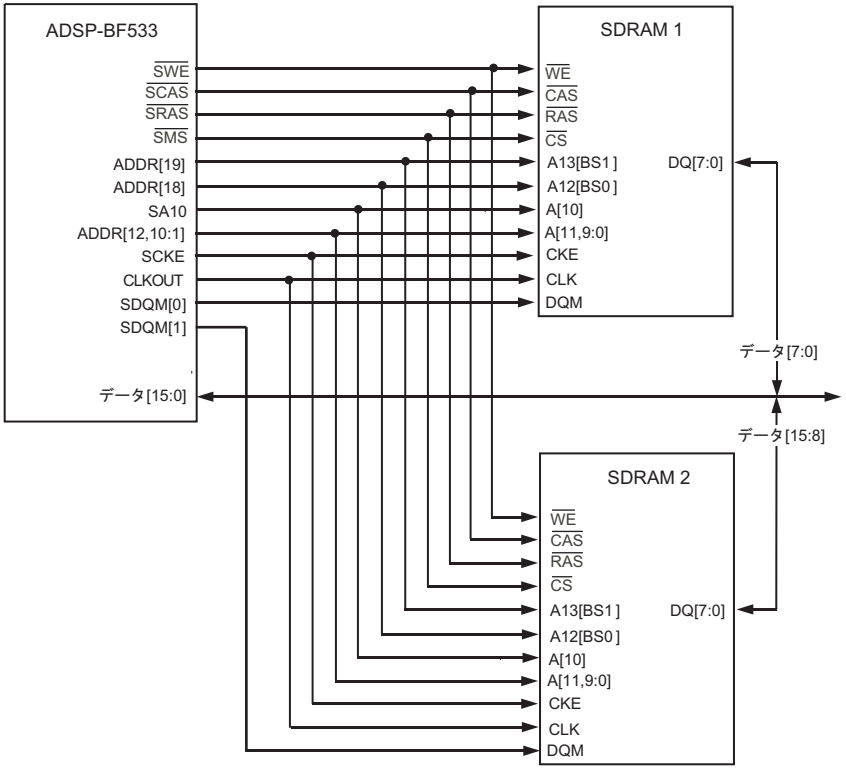


図 17-9. 16M バイト SDRAM システムの例

■ EBIU_SDGCTL レジスタ

SDRAMメモリ・グローバル・コントロール・レジスタ（EBIU_SDGCTL）には、SDRAMアクセスのタイミングと設定に関連するすべてのプログラマブルなパラメータが含まれています。図 17-10は、EBIU_SDGCTLレジスタのビット定義を示します。

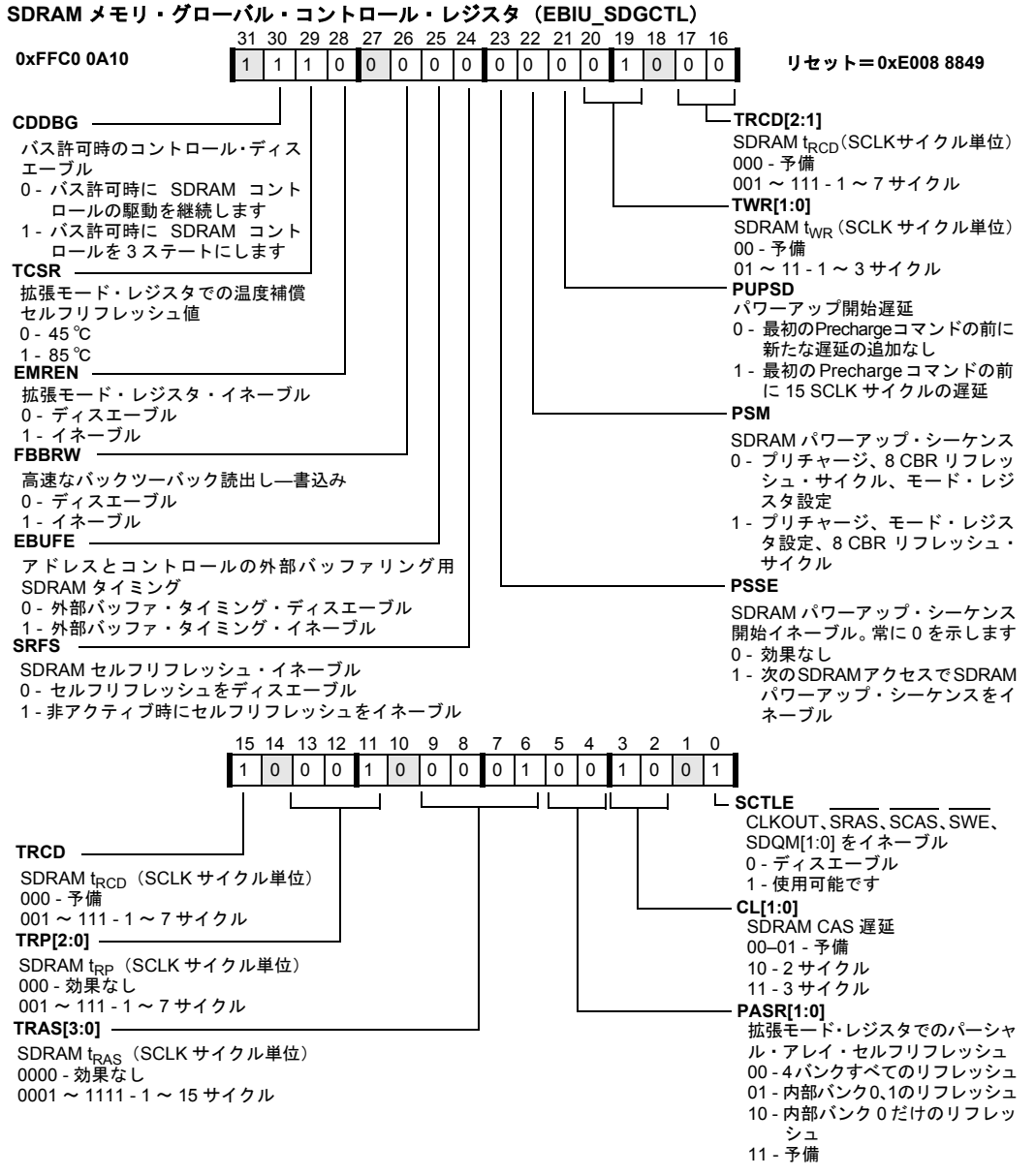


図 17-10.SDRAM メモリ・グローバル・コントロール・レジスタ

SCTLE ビットは、SDC をイネーブル／ディスエーブルにするために使用されます。SCTLE がディスエーブルにされた場合には、SDRAM アドレス空間へのアクセスによって内部バス・エラーが生成され、外部からのアクセスは行われません。詳細については、17-9 ページの「エラー検出」を参照してください。SCTLE がディスエーブルにされると、すべての SDC 制御ピンは非アクティブ状態になり、SDRAM クロックは動作しません。SCTLE ビットは、SDC 動作用にイネーブルにする必要があり、デフォルトではリセット時にイネーブルにされます。

CAS 遅延（CL）、SDRAM t_{RAS} タイミング（TRAS）、SDRAM t_{RP} タイミング（TRP）、SDRAM t_{RCD} タイミング（TRCD）、SDRAM t_{WR} タイミング（TWR）の各ビットは、使用する SDRAM のタイミング仕様とシステム・クロック周波数に基づいてプログラムしてください。

PSM ビットと PSSE ビットは、SDRAM パワーアップ（初期化）シーケンスを指定およびトリガするために連携して働きます。PSM ビットが 1 に設定された場合、SDC は Precharge All コマンドを実行し、続いて Load Mode Register コマンドを実行し、さらに 8 つの Auto-Refresh サイクルを実行します。PSM ビットがクリアされた場合、SDC は Precharge All コマンドを実行し、続いて 8 つの Auto-Refresh サイクルを実行し、さらに Load Mode Register コマンドを実行します。SDC が SDRAM パワーアップ・シーケンスを実行する前に、次の 2 つのイベントを行う必要があります。

- SDRAM パワーアップ・シーケンスをイネーブルにするため、PSSE ビットを 1 に設定する必要があります。
- SDRAM パワーアップ・シーケンスが行えるように、SDC に対して外部バスを許可するため、イネーブルにされた SDRAM アドレス空間への読出し／書込みアクセスを行う必要があります。

SDRAMパワーアップ・シーケンスが行われると、その直後に、SDRAMパワーアップ・シーケンスのトリガに使用されたSDRAMへの読出し／書込み転送が続きます。なお、SDRAMパワーアップ・シーケンスの完了には多くのサイクルを要するため、SDRAMへのこの最初のアクセスには遅延があります。

SDCパワーアップ・シーケンスを実行する前に、SDRAM仕様に基づいて、SDRAMが安定した電力を受け取り、適切な時間だけクロック駆動されることを確認します。

パワーアップ・スタート遅延（PUPSD）ビットは、オプションでパワーアップ・スタート・シーケンスを15 SCLKサイクルだけ遅延させます。これは、外部SDRAMを共有するマルチプロセッサ・システムに役立ちます。パワーアップの前にバスが他のプロセッサに対してあらかじめ許可されており、バス所有権を切り替えるときにSelf-Refreshモードが使用された場合には、SDRAMの終了セルフリフレッシュ時間（t_{XSR}）を満たすためにPUPSDビットを使用して、セルフリフレッシュからパワーアップ・シーケンスでの最初のPrechargeコマンドまでに十分な非アクティブ期間を保証できます。

SRFSビットが1に設定されると、Self-Refreshモードがトリガされます。SDCがいずれかのアクティブな転送を完了すると、SDCは一連のコマンドを実行してSDRAMをSelf-Refreshモードにします。イネーブルにされたSDRAMバンクへの次のアクセスによって、SDCはSDRAMをSelf-Refreshから抜け出させてアクセスを実行させるためのコマンドを実行します。SRFSビットの詳細については、17-40ページの「Self-Refreshモードへの出入り（SRFS）」を参照してください。

EBUFEビットは、外部バッファ・タイミングをイネーブル／ディスエーブルにするために使用されます。SDRAMコントロール入力の駆動にバッファ付きSDRAMモジュールまたはディスクリート・レジスタ・バッファが使用される場合、EBUFEを1に設定してください。この設定を使用すると、読出しアクセスと書込みアクセスに1サイクルのデータ・バッファリ

ングが追加されます。EBUFE ビットの詳細については、17-41 ページの「SDRAM バッファ・タイミング・オプションの設定（EBUFE）」を参照してください。

FBBRW ビットは、SDRAM 読出しの後に続けて書込みを連続したサイクルでの実行を可能にします。多くのシステムでは、これは不可能です。なぜなら、SDRAM データ・ピンのターン・オフ時間が長すぎて、プロセッサからの後続の書込みとのバス競合が発生するからです。このビットが 0 の場合、直後に書込みアクセスが続く読出しアクセス間に 1 つのクロック・サイクルが挿入されます。

EMREN ビットは、スタートアップ時に拡張モード・レジスタのプログラミングを可能にします。拡張モード・レジスタは、特定のモバイル低消費電力 SDRAM において、SDRAM 消費電力の制御に使用されます。EMREN ビットがイネーブルにされた場合、TCSR ビットと PASR[1:0] ビットは、拡張モード・レジスタに書き込まれる値を制御します。PASR ビットは、Self-Refresh 時にリフレッシュされる SDRAM 内部バンクの数を決定します。TCSR ビットは、システムにとって最悪ケースの温度範囲を SDRAM に通知することで、Self-Refresh 時に必要な SDRAM 内部バンクのリフレッシュ頻度を示します。

外部メモリ・インターフェースが外部コントローラに許可されているとき、CDDBG ビットは SDRAM コントロール信号のイネーブル／ディスエーブルに使用されます。このビットが 1 に設定される場合、バス許可がアクティブのとき、制御信号は 3 ステート状態になります。そうでない場合、これらの信号は許可時に駆動し続けられます。このビットがセットされ、外部バスが許可された場合には、すべての SDRAM 内部バンクは外部コントローラによって変更されたと考えられます。このことは、外部バスの制御が再確立された後では、使用の前に各バンクでプリチャージが必要なことを意味します。このピンによって影響される制御信号は、$\overline{\text{SRAS}}$、$\overline{\text{SCAS}}$、$\overline{\text{SWE}}$、SMS、SA10、SCKE、CLKOUT です。

なお、このレジスタ内のすべての予備ビットには、常に 0 を書き込む必要があります。

▶ SDRAM クロック・イネーブルの設定（SCTLE）

SCTLE ビットによって、ソフトウェアはすべてのSDRAM制御ピンをディスエーブルにできます。これらのピンは、SDQM[3:0]、$\overline{\text{SCAS}}$、$\overline{\text{SRAS}}$、$\overline{\text{SWE}}$、SCKE、CLKOUT です。

- SCTLE = 0

  すべてのSDRAM制御ピンをディスエーブルにします（制御ピンはネゲート、CLKOUT ローレベル）

- SCTLE = 1

  すべてのSDRAM制御ピンをイネーブルにします（CLKOUT トグル）

なお、CLKOUT機能は、AMCとも共有されます。たとえSCTLEがディスエーブルにされていても、CLKOUT はAMCのCLKOUTイネーブル（EBIU_AMGCTL レジスタのAMCKEN ）によって独立してイネーブルできます。

システムがSDRAMを使用しない場合には、SCTLE は0に設定してください。

SCTLE が0である間に、SDRAMアドレス空間へのアクセスが発生した場合には、アクセスによって内部バス・エラーが生成され、外部からのアクセスは行われません。詳細については、17-9ページの「エラー検出」を参照してください。綿密なソフトウェア制御によってSCTLE ビットをSelf-Refreshモードと組み合わせて使用すれば、さらに消費電力を下げることができます。しかし、SDRAMへのAuto-Refreshコマンドの生成にSDCが必要な場合には、SCTLE を常時イネーブルにしておく必要があります。

▶ Self-Refresh モードへの出入り（SRFS）

SDCは、SDRAM Self-Refreshモードを提供します。Self-Refreshモードでは、SDRAMが外部制御なしでリフレッシュ動作を内部的に実行することで、SDRAMの消費電力を減らします。

EBIU_SDGCTLのSRFSビットは、Self-Refreshモードの開始をイネーブルにします。

- SRFS = 0

  Self-Refreshモードをディスエーブルにします

- SRFS = 1

  Self-Refreshモードをイネーブルにします

SRFSが1に設定された場合、SDCがアイドル状態に入ると、必要ならばPrechargeコマンドを発行してから、Self-Refreshコマンドを発行します。内部アクセスが保留中の場合、SDCは、保留中のSDRAMアクセスとそれ以降の保留中のアクセス要求を完了するまで、Self-Refreshコマンドの発行を遅延させます。詳細については、17-58ページの「SDCコマンド」を参照してください。

SDRAMデバイスがSelf-Refreshモードに入ると、SDRAMコントローラは、SDRAMコントロール・ステータス・レジスタ（EBIU_SDSTAT）のSDSRAビットをアサートします。

SDRAMデバイスは、SDCがコアまたはDMAのアクセス要求を受信したときにだけSelf-Refreshモードを終了します。アクセス要求の前にSRFSビットがクリアされた場合には、SDCはアクティブのままでいます。しかし、SRFSビットがまだセットされている場合には、保留中のアクセスがなくなるとすぐにSDCはセルフリフレッシュに再び入ります。

なお、SRFSビットが1に設定されると、SDCは保留中のアクセスを完了する時にSelf-Refreshモードに入ります。Self-Refreshモードへのエントリをキャンセルする方法はありません。

▶ SDRAMバッファ・タイミング・オプションの設定（EBUFE）

システム全体のタイミング条件を満たすため、並列に接続された複数のSDRAMデバイスを使用するシステムは、プロセッサと複数のSDRAMデバイスとの間にバッファリングを必要とする場合があります。一般に、このバッファリングはレジスタとドライバから構成されます。

このようなタイミング条件を満たし、中間登録を可能にするため、SDCはSDRAMアドレスと制御信号のパイプライン処理を提供します。

EBIU_SDGCTLレジスタのEBUFEビットは、このモードをイネーブルにします。

- `EBUFE = 0`

  外部バッファ・タイミングをディスエーブルにします

- `EBUFE = 1`

  外部バッファ・タイミングをイネーブルにします

`EBUFE = 1`の場合、SDRAMコントローラは書込みアクセスでデータを1サイクルだけ遅延させ、外部バッファ・レジスタによるアドレスと制御のラッチを可能にします。読出しアクセスで、SDRAMコントローラは、外部バッファ・レジスタによって追加される1サイクルの遅延に対処するため、1サイクル後でデータをサンプリングします。外部バッファ・タイミングがイネーブルにされると、すべてのアクセスの遅延は1サイクルだけ増やされます。

▶ CAS 遅延値の選択（CL）

CAS遅延値は、SDRAMがReadコマンドを検出してから、その出力ピンでデータを提供するまでの遅延をクロック・サイクル数で定義します。

CAS遅延は、書込みサイクルには適用されません。

SDRAMメモリ・グローバル・コントロール・レジスタ（EBIU_SDGCTL）のCLビットでは、CAS遅延値を選択します。

- CL = 00

  予備

- CL = 01

  予備

- CL = 10

  2クロック・サイクル

- CL = 11

  3クロック・サイクル

一般に、CAS遅延の値は動作周波数によって決定されます。この値の設定の詳細については、SDRAMデバイスのマニュアルを参照してください。

▶ SDQM 動作

SDQM[1:0]（データI/Oマスク）ピンによって、SDCはバイト転送時にバイトをマスクすることができます。書込みサイクルの場合、データ・マスクの遅延は0サイクルであるため、対応するSDQM[x]ピンのローレベルがサンプリングされるとデータ書込みが許可され、SDQM[x]ピンのハイレベルがサンプリングされるとデータ書込みがブロックされます。なお、これらはバイト単位で行われます。

▶ 並列リフレッシュ・コマンドの実行

SDCにはAuto-Refreshコマンドと非同期メモリ・アクセスとの並列実行を可能にする、独立したアドレス・ピン（SA10）があります。このピンによって、SDCはSDRAMにPrechargeコマンドを発行してから、Auto-Refreshコマンドを発行することができます。さらに、SA10ピンによって、SDCは非同期メモリ・アクセスと並列に、Self-Refreshモードに出入りすることができます。

SA10ピンは、SDRAMのA10ピンに直結してください（ADDR[10]ピンには接続しません）。Prechargeコマンドの処理中、SA10はPrecharge Allを実行すべきことを示すために使用されます。Bank Activateコマンドの処理中に、SA10は内部行アドレス・ビットを出力します。このビットは、A10 SDRAM入力に多重化してください。Readコマンドと Writeコマンドの処理中、SA10はSDRAMの自動プリチャージ機能をディスエーブルにするために使用されます。

▶ Bank Activate コマンド遅延の選択（TRAS）

t_{RAS}値（Bank Activateコマンド遅延）は、SDCがBank Activateコマンドを発行してからPrechargeコマンドを発行するまでに必要な遅延をクロック・サイクル数で定義します。SDRAMも、少なくともt_{RAS}によって指定される期間は、Self-Refreshモードにとどまる必要があります。t_{RP}値とt_{RAS}値は、t_{RFC}、t_{RC}、およびt_{XSR}の値を定義します。詳細については、17-30ページを参照してください。

t_{RAS} パラメータによって、プロセッサはシステムのSDRAMデバイスのタイミング条件に適合できます。

SDRAMメモリ・グローバル・コントロール・レジスタ（EBIU_SDGCTL）のTRASビットは、t_{RAS}値を選択します。1～15クロック・サイクルの任意の値を選択できます。次に例を示します。

- TRAS = 0000

  効果なし

- TRAS = 0001

  1クロック・サイクル

- TRAS = 0010

  2クロック・サイクル

- TRAS = 1111

  15クロック・サイクル

この値の設定の詳細については、SDRAMデバイスのマニュアルを参照してください。

▶ プリチャージ遅延の選択（TRP）

t_{RP}値（Precharge遅延）は、SDCがPrechargeコマンドを発行してからBank Activateコマンドを発行するまでに必要な遅延をクロック・サイクル数で定義します。このt_{RP}は、PrechargeとAuto-Refreshとの間、およびPrechargeとSelf-Refreshとの間に必要な時間も指定します。t_{RP}値とt_{RAS}値は、t_{RFC}、t_{RC}、およびt_{XSR}の値を定義します。

このパラメータによって、アプリケーションはSDRAMのタイミング条件に対処できます。

SDRAMメモリ・グローバル・コントロール・レジスタ（EBIU_SDGCTL）のTRPビットは、t_{RP}値を選択します。1～7クロック・サイクルの任意の値を選択できます。次に例を示します。

- TRP = 000

  効果なし

- TRP = 001

  1クロック・サイクル

- TRP = 010

  2クロック・サイクル

- TRP = 111

  7クロック・サイクル

▶ Precharge 遅延への書込みの選択（TWR）

t_{WR}値は、SDCがWriteコマンド（書込みデータを駆動）を発行してからPrechargeコマンドを発行するまでに必要な遅延をクロック・サイクル数で定義します。

このパラメータによって、アプリケーションはSDRAMのタイミング条件に対処できます。

SDRAMメモリ・グローバル・コントロール・レジスタ（EBIU_SDGCTL）のTWRビットは、t_{WR}値を選択します。1～3クロック・サイクルの任意の値を選択できます。次に例を示します。

- TWR = 00

  予備

- TWR = 01

  1クロック・サイクル

- TWR = 10

  2クロック・サイクル

- TWR = 11

  3クロック・サイクル

■ EBIU_SDBCTL レジスタ

SDRAMメモリ・バンク・コントロール・レジスタ（EBIU_SDBCTL）には、外部バンク固有のプログラマブルなパラメータがあります。これらによって、ソフトウェアはSDRAMのいくつかのパラメータを制御できます。外部バンクはSDRAMのさまざまなサイズに合わせて設定でき、SDRAMメモリ・グローバル・コントロール・レジスタ（EBIU_SDGCTL）で定義されたアクセス・タイミング・パラメータを使用します。EBIU_SDBCTLレジスタはパワーアップの前にプログラムし、SDCがアイドル状態のときにだけ変更してください。

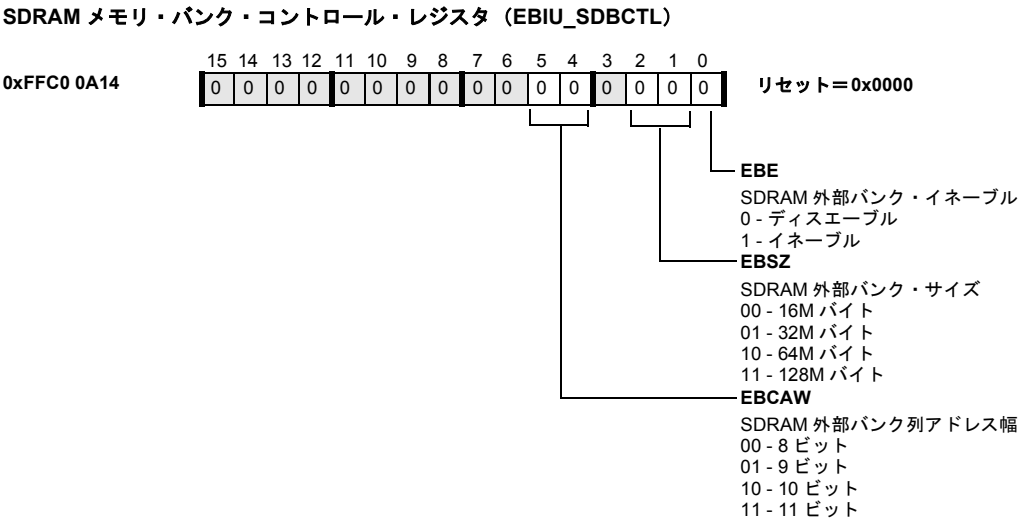


図 17-11. SDRAM メモリ・バンク・コントロール・レジスタ

EBIU_SDBCTL レジスタは、SDRAM バンク・インターフェースの設定情報を格納します。EBIU は、×4、×8、×16 の設定を持つ 64M ビット、128M ビット、256M ビット、512M ビットの SDRAM デバイスに対応します。17-32 ページの表 17-4 は、SDRAM 集積度と I/O 幅を、利用可能な EBSZ エンコーディングにマッピングします。バンク開始アドレスのデコードの詳細については、17-52 ページの「SDRAM 外部メモリのサイズ」を参照してください。

SDC は、EBCAW パラメータから内部 SDRAM のページ・サイズを決定します。512B、1K バイト、2K バイト、4K バイトのページ・サイズがサポートされます。17-48 ページの表 17-5 は、ページ・サイズと、内部アドレス（IA[31:0]、コアや DMA から見たもの）の行、バンク、列、バイト・アドレスへの分解を示します。列アドレスとバイト・アドレスを組み合わせて、ページ内のアドレスが構成されます。

EBIU_SDBCTL レジスタの EBE ビットは、外部 SDRAM バンクのイネーブル／ディスエーブルに使用されます。SDRAM がディスエーブルにされた場合には、SDRAM アドレス空間へのアクセスは内部バス・エラーを生成し、外部からのアクセスは行われません。詳細については、17-9 ページの「エラー検出」を参照してください。

表 17-5.　内部アドレス・マッピング

バンク幅（ビット）	バンク・サイズ（M バイト）	列アドレス幅（CAW）	ページ・サイズ（K バイト）	バンク・アドレス	行アドレス	ページ	
						列アドレス	バイト・アドレス
16	128	11	4	IA[26:25]	IA[24:12]	A[11:1]	IA[0]
16	128	10	2	IA[26:25]	IA[24:11]	IA[10:1]	IA[0]
16	128	9	1	1A[26:25]	IA[24:10]	IA[9:1]	IA[0]
16	128	8	.5	IA[26:25]	IA[24:9]	IA[8:1]	IA[0]
16	64	11	4	IA[25:24]	IA[23:12]	IA[11:1]	IA[0]
16	64	10	2	IA[25:24]	IA[23:11]	IA[10:1]	IA[0]
16	64	9	1	IA[25:24]	IA[23:10]	IA[9:1]	IA[0]
16	64	8	.5	IA[25:24]	IA[23:9]	IA[8:1]	IA[0]
16	32	11	4	IA[24:23]	IA[22:12]	IA[11:1]	IA[0]
16	32	10	2	IA[24:23]	IA[22:11]	IA[10:1]	IA[0]
16	32	9	1	IA[24:23]	IA[22:10]	IA[9:1]	IA[0]
16	32	8	.5	IA[24:23]	IA[22:9]	IA[8:1]	IA[0]
16	16	11	4	IA[23:22]	IA[21:12]	IA[11:1]	IA[0]
16	16	10	2	IA[23:22]	IA[21:11]	IA[10:1]	IA[0]
16	16	9	1	IA[23:22]	IA[21:10]	IA[9:1]	IA[0]
16	16	8	.5	IA[23:22]	IA[21:9]	IA[8:1]	IA[0]

16Mバイト未満のSDRAMへの接続方法については、18-8ページの「16Mバイトよりも小さいSDRAMの利用」を参照してください。

■ EBIU_SDSTAT レジスタ

SDRAMコントロール・ステータス・レジスタ（EBIU_SDSTAT）は、SDCの状態に関する情報を提供します。この情報を使用すれば、SDC制御パラメータを変更しても安全なタイミングを判定できます。あるいは、これをデバッグ用に使用することも可能です。このレジスタのSDEASEビットはスティッキーです。このビットがセットされたら、これをクリアするには、ソフトウェアでこのビットに明示的に1を書き込む必要があります。他のステータス・ビットについては、書込みは効果がなく、SDCによってのみ更新されます。このSDC MMRは16ビット幅です。

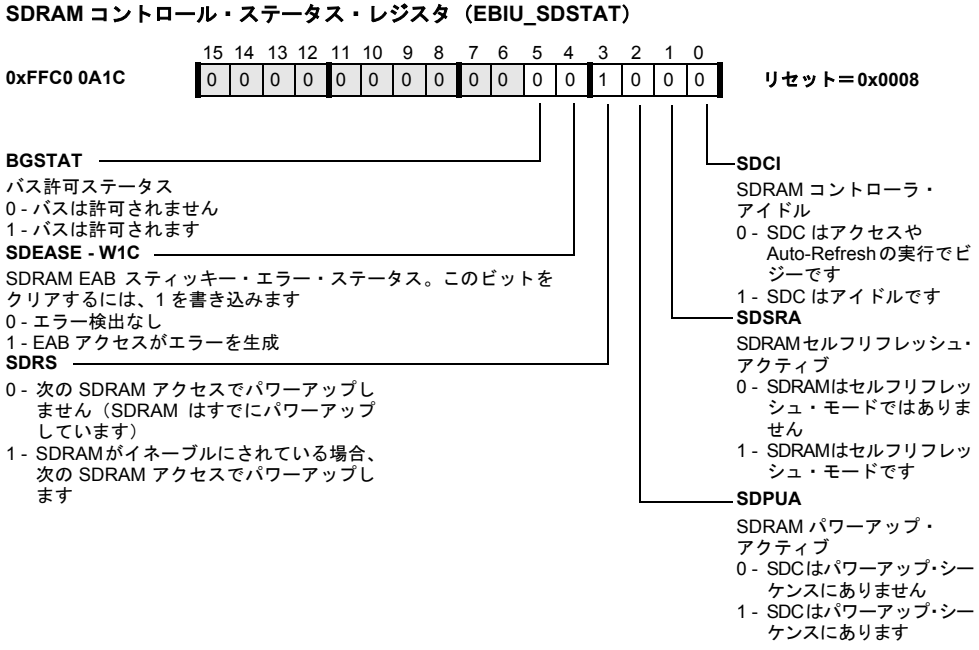


図 17-12.SDRAM コントロール・ステータス・レジスタ

■ EBIU_SDRRC レジスタ

SDRAMリフレッシュ・レート・コントロール・レジスタ（EBIU_SDRRC）は、Auto-Refreshのタイミングを指定するための柔軟なメカニズムを提供します。SDRAMに供給されるクロックは変動することがあるため、SDCは、プログラマブルなリフレッシュ・カウンタを提供します。このカウンタの周期は、このレジスタのRDIVフィールドにプログラムされた値をベースにしています。このカウンタは、供給されるクロック・レートとSDRAMデバイスの必要な更新レートを調整します。

リフレッシュ・カウンタの隣接するタイムアウト間の遅延（SDRAMクロック・サイクル単位で）の設定は、RDIVフィールドに書き込む必要があります。リフレッシュ・カウンタのタイムアウトは、すべての外部SDRAMデバイスに対してAuto-Refreshコマンドをトリガします。SDRAMパワーアップ・シーケンスがトリガされる前に、RDIV値をEBIU_SDRRCレジスタに書き込みます。この値は、SDCがアイドル状態のときにだけ変更してください。

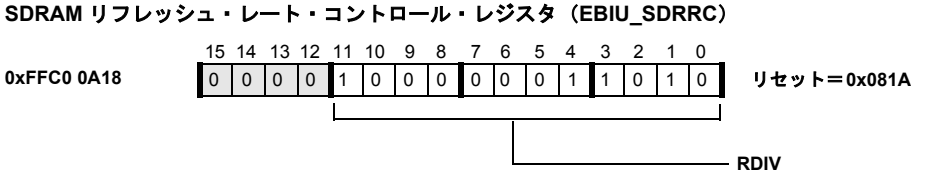


図 17-13.SDRAM リフレッシュ・レート・コントロール・レジスタ

EBIU_SDRRCレジスタに書き込む値を計算するには、次の式を使用します。

$RDIV = ((f_{SCLK} \times t_{REF}) / NRA) - (t_{RAS} + t_{RP})$

ここで、

- f_{SCLK}＝SDRAMクロック周波数（システム・クロック周波数）

- t_{REF}＝SDRAMリフレッシュ周期
- NRA ＝ SDRAM内の行アドレス数（SDRAM全体をリフレッシュするためのリフレッシュ・サイクル）
- t_{RAS}＝クロック・サイクル数で表した、ActiveからPrechargeまでの時間（SDRAMメモリ・グローバル・コントロール・レジスタのTRAS）
- t_{RP}＝クロック・サイクル数で表した、RASからPrechargeまでの時間（SDRAMメモリ・グローバル・コントロール・レジスタのTRP）

この式は、必要なリフレッシュ間のクロック・サイクル数を計算し、同じ内部バンクに対するBank Activateコマンド間の必要な遅延（$t_{RC} = t_{RAS} + t_{RP}$）を減算します。$t_{RC}$値が減算されることによって、SDRAMサイクルがアクティブである間にリフレッシュのタイムアウトが発生した場合、SDRAMの更新レート仕様が満たされることが保証されます。式の結果は、常に整数に切り捨てます。

次に示すのは、133MHz のクロックを備えたシステムでの代表的なSDRAMに対するRDIVの計算例です。

f_{SCLK} = 133 MHz

t_{REF} = 64 ms

NRA = 4096 行アドレス

t_{RAS} = 2

t_{RP} = 2

RDIVの式は次のように得られます。

$RDIV = ((133 \times 10^{6} \times 64 \times 10^{-3}) / 4096) - (2 + 2) = 2074$クロック・サイクル

これは、RDIVが0x81A（16進）であり、SDRAMリフレッシュ・レート・コントロール・レジスタに0x081Aを書き込むべきことを意味します。

なお、SDRAMコントローラがイネーブルにされている場合には、RDIVには非ゼロ値をプログラムする必要があります。RDIV＝0の場合、SDRAMコントローラの動作は利用できず、望ましくない動作を招くことがあります。RDIVの値は、0x001～0xFFFの範囲とすることができます。

SDRAMを使用する場合、PLL周波数の変更プロセスの詳細については、18-10ページの「PLL遷移時におけるSDRAMのリフレッシュ管理」を参照してください。

■ SDRAM 外部メモリのサイズ

プロセッサによってアドレス指定される外部SDRAMメモリの合計量は、EBIU_SDBCTLレジスタのEBSZビットによって制御されます（表17-6を参照）。EBSZ値で示される範囲を超えてのアクセスは、内部バス・エラーにつながり、アクセスは行われません。詳細については、17-9ページの「エラー検出」を参照してください。

表 17-6.　バンク・サイズのエンコーディング

EBSZ	バンク・サイズ（M バイト）	有効な SDRAM アドレス
00	16	0x0000 0000 – 0x00FF FFFF
01	32	0x0000 0000 – 0x01FF FFFF
10	64	0x0000 0000 – 0x03FF FFFF
11	128	0x0000 0000 – 0x07FF FFFF

■ SDRAM アドレスのマッピング

SDRAMにアクセスするため、SDCは内部にある32ビットの非多重化アドレスを、SDRAMデバイスの行アドレス、列アドレス、バンク・アドレス、バイト・データ・マスクに多重化します。図 17-14を参照してください。最下位ビットはバイト・データ・マスクにマッピングされ、次のビットは列アドレスにマッピングされ、その次のビットは行アドレスにマッピングされ、最後の2ビットはバンク・アドレスにマッピングされます。このマッピングは、SDRAMメモリ・バンク・コントロール・レジスタにプログラムされたEBSZとEBCAWのパラメータをベースにしています。

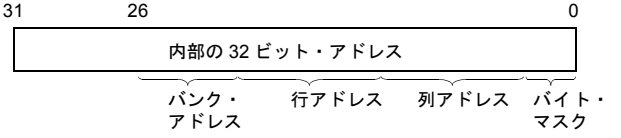


図 17-14.多重化された SDRAM アドレッシング方式

▶ 16 ビット幅の SDRAM アドレスの多重化

次の表は、アドレス・ピンとSDRAMデバイス・ピンとの接続を示します。

表 17-7.　16 ビット・バンク用の SDRAM アドレス接続

外部アドレス・ピン	SDRAM アドレス・ピン
ADDR[19]	BA[1]
ADDR[18]	BA[0]
ADDR[16]	A[15]
ADDR[15]	A[14]
ADDR[14]	A[13]
ADDR[13]	A[12]
ADDR[12]	A[11]

表 17-7.　16 ビット・バンク用の SDRAM アドレス接続（続き）

外部アドレス・ピン	SDRAM アドレス・ピン
SA[10]	A[10]
ADDR[10]	A[9]
ADDR[9]	A[8]
ADDR[8]	A[7]
ADDR[7]	A[6]
ADDR[6]	A[5]
ADDR[5]	A[4]
ADDR[4]	A[3]
ADDR[3]	A[2]
ADDR[2]	A[1]
ADDR[1]	A[0]

■ データ・マスク（SDQM[1:0]）のエンコーディング

SDRAMへの書込み転送中、SDQM[1:0]ピンはアクセスされないバイトへの書込みをマスクするために使用されます。17-55ページの表 17-8は、内部転送アドレス・ビットIA[0]と転送サイズに基づいて、16ビット幅SDRAM用のSDQM[1:0]のエンコーディングを示します。

SDRAMバンクへの読出し転送中、読出しは転送サイズとは無関係に、常にバンク内の全バイトについて行われます。つまり、16ビットSDRAMバンクの場合、SDQM[1:0]はオール0です。

SDQM[1:0]ピンがハイレベルとなる唯一のケースは、SDRAMへの書込み転送中にバイトがマスクされた場合です。それ以外の場合、SDQM[1:0]ピンはローレベルに保持されます。

表 17-8. 書込み中の SDQM[1:0] のエンコーディング

内部アドレス IA[0]	内部転送サイズ		
	バイト	2バイト	4バイト
0	SDQM[1] = 1 SDQM[0] = 0	SDQM[1] = 0 SDQM[0] = 0	SDQM[1] = 0 SDQM[0] = 0
1	SDQM[1] = 0 SDQM[0] = 1		

■ SDC 動作

SDCでは、読出し／書込み動作にバースト長＝1を使用します。ページ・ミスが発生するたびに、SDCはPrechargeコマンドを実行し、続いてBank Activateコマンドを実行してから、ReadまたはWriteコマンドを実行します。ページ・ヒットがあった場合には、Prechargeコマンドを必要とすることなく、ReadまたはWriteコマンドをすぐに出すことができます。

SDRAM Readコマンドの場合、Readコマンドの開始からSDRAMのデータが使用可能になるまでには、CAS遅延に等しい遅延があります。この遅延は、任意の単一読出し転送には常に存在します。2回目以降の読出しには遅延はありません。

プログラマブルなリフレッシュ・カウンタが提供されます。このカウンタは、使用するクロック周波数に基づいて、必要な更新レートでバックグラウンドのAuto-Refreshサイクルを生成するようにプログラムできます。リフレッシュ・カウンタの周期は、SDRAMリフレッシュ・レート・コントロール・レジスタのRDIVフィールドで指定されます。

Auto-RefreshコマンドをどのAMCアクセスとも並列に実行できるようにするため、独立したA10ピン（SA10）が提供されています。Auto-Refreshコマンドを発行する前に、すべてのSDRAM内部バンクがプリチャージされます。

内部にある32ビットの非多重化アドレスは、行アドレス、列アドレス、バンク選択アドレス、データ・マスクに多重化されます。16ビット幅SDRAMのビット0は、データ・マスクの生成に使用されます。次に低いビットは列アドレスにマッピングされ、次のビットは行アドレスにマッピングされ、最後の2ビットは内部バンク・アドレスにマッピングされます。このマッピングは、SDRAMメモリ・バンク・コントロール・レジスタにプログラムされたEBCAWとEBSZの値に基づいています。

■ SDC の設定

プロセッサのハードウェア／ソフトウェア・リセット後に、SDCクロックがイネーブルにされます。しかし、SDCは、これ以外にも設定および初期化する必要があります。SDCをプログラミングしてパワーアップ・シーケンスを実行する前に、SDRAMによって指定される適切な時間にわたって電力が安定した後、SDRAMへのクロックをイネーブルにしてください。SDRAM用にSDCを設定してSDRAMパワーアップ・シーケンスを開始するには、SDRAMリフレッシュ・レート・コントロール・レジスタ（EBIU_SDRRC）、SDRAMメモリ・バンク・コントロール・レジスタ（EBIU_SDBCTL）、SDRAMメモリ・グローバル・コントロール・レジスタ（EBIU_SDGCTL）に書き込み、SDRAMアドレス空間への転送を開始する必要があります。SDRAMコントロール・ステータス・レジスタのSDRSビットをチェックすれば、SDCの現在の状態を判定できます。このビットがセットされている場合、SDRAMパワーアップ・シーケンスは開始されていません。

SDRAM更新レートを設定するには、EBIU_SDRRCレジスタのRDIVフィールドに書き込んでください。

使用するSDRAMメモリ設定とサイズ（EBCAWとEBSZ）を記述し、外部バンク（EBE）をイネーブルにするには、EBIU_SDBCTLレジスタに書き込んでください。なお、SDRAMパワーアップ・シーケンスが開始されるまでは、EBEビットの状態とは無関係に、SDRAMアドレス空間へのアクセスは内部バス・エラーを生成し、外部からのアクセスは行われません。詳細については、17-9ページの「エラー検出」を参照してください。SDRAMパワーアップ・シーケンスが完了した後、外部バンクがディスエーブルにされている場合には、外部バンクへの転送はハードウェア・エラー割込みを招き、SDRAM転送は行われません。

EBIU_SDGCTLレジスタは、以下の目的で書き込まれます。

- SDRAMサイクル・タイミング・オプションを設定する（CL、TRAS、TRP、TRCD、TWR、EBUFE）
- SDRAMクロックをイネーブルにする（SCTLE）
- SDRAMパワーアップ・シーケンスの開始を選択してイネーブルにする（PSM、PSSE）

なお、SCTLEがディスエーブルにされている場合、SDRAMアドレス空間へのアクセスは内部バス・エラーを生成し、外部からのアクセスは行われません。詳細については、17-9ページの「エラー検出」を参照してください。

EBIU_SDGCTLレジスタのPSSEビットが1に設定され、イネーブルにされたSDRAMアドレス空間に転送が行われると、SDCはSDRAMパワーアップ・シーケンスを開始します。厳密なシーケンスは、EBIU_SDGCTLレジスタのPSMビットによって決定されます。SDRAMパワーアップ・シーケンスのトリガに使用される転送は、読出しまたは書込みとすることができます。この転送は、SDRAMパワーアップ・シーケンスが完了したときに行われます。この最初の転送では、SDRAMパワーアップ・シーケンスを行う必要があるため、完了までに多くのサイクルを必要とします。

■ SDC コマンド

ここでは、SDCがSDRAMインターフェースの管理に使用するコマンドをそれぞれ説明します。これらのコマンドは、メモリ読出しまたはメモリ書込みと同時に、自動的に開始されます。SDRAMインターフェース用のオンチップ・コントローラで使用される、さまざまなコマンドのまとめは次のとおりです。

- Precharge All：すべてのバンクをプリチャージします。
- Single Precharge：1つのバンクをプリチャージします。
- Bank Activate：要求されたSDRAM内部バンク内のページをアクティブにします。
- Load Mode Register：パワーアップ・シーケンス中にSDRAMの動作パラメータを初期化します。
- Load Extended Mode Register：パワーアップ・シーケンス中にモバイルSDRAMの動作パラメータを初期化します。
- Read/Write
- Auto-Refresh：SDRAMは、CAS before RASリフレッシュを実行します。
- Self-Refresh：SDRAM をセルフリフレッシュ・モードにします。このモードでは、SDRAMはパワーダウンし、そのリフレッシュ動作を内部的に制御します。
- NOP/Command Inhibit：無動作

次の表には、SDCコマンド処理中のSDRAMピンの状態を示します。

表 17-9. SDC コマンド処理中のピン状態

コマンド	$\overline{\text{SMS}}$	$\overline{\text{SCAS}}$	$\overline{\text{SRAS}}$	$\overline{\text{SWE}}$	SCKE	SA10
Precharge All	ローレベル	ハイレベル	ローレベル	ローレベル	ハイレベル	ハイレベル
Single Precharge	ローレベル	ハイレベル	ローレベル	ローレベル	ハイレベル	ローレベル
Bank Activate	ローレベル	ハイレベル	ローレベル	ハイレベル	ハイレベル	
Load Mode Register	ローレベル	ローレベル	ローレベル	ローレベル	ハイレベル	
Load Extended Mode Register	ローレベル	ローレベル	ローレベル	ローレベル	ハイレベル	ローレベル
Read	ローレベル	ローレベル	ハイレベル	ハイレベル	ハイレベル	ローレベル
Write	ローレベル	ローレベル	ハイレベル	ローレベル	ハイレベル	ローレベル
Auto-Refresh	ローレベル	ローレベル	ローレベル	ハイレベル	ハイレベル	
Self-Refresh	ローレベル	ローレベル	ローレベル	ハイレベル	ローレベル	
NOP	ローレベル	ハイレベル	ハイレベル	ハイレベル	ハイレベル	
Command Inhibit	ハイレベル	ハイレベル	ハイレベル	ハイレベル	ハイレベル	

▶ Precharge コマンド

Precharge Allコマンドによって、すべての内部バンクを同時にプリチャージしてから自動リフレッシュができます。特定の内部SDRAMバンクにおける読出し／書込み時のページ・ミスについては、SDCはそのバンクへのSingle Prechargeコマンドを使用します。

▶ Bank Activate コマンド

Bank Activateコマンドは、次のデータ・アクセスが別のページにある場合に必要です。SDCは、Prechargeコマンドの後に続けてBank Activateコマンドを実行して、希望するSDRAM内部バンク内のページをアクティブにします。

SDCはバンクのインターリーブに対応します（同時に最大4つの内部SDRAMバンクをオープンします）。これにより、有効ページ・サイズは4になります。アドレス・マッピングで各内部バンクの開始アドレスを示します。

バンクのインターリーブは、同じページ内でストールなしに4つの内部SDRAMバンクをスイッチすることによって行われます。

▶ Load Mode Register コマンド

Load Mode Registerコマンドは、SDRAM動作パラメータを初期化します。このコマンドは、SDRAMパワーアップ・シーケンスの一部です。Load Mode Registerコマンドでは、SDRAMのアドレス・バスをデータ入力として使用します。パワーアップ・シーケンスを開始するには、SDRAMメモリ・グローバル・コントロール・レジスタ（EBIU_SDGCTL）のPSSEビットに1を書き込み、SDRAMアドレス空間内のイネーブルにされた任意のアドレスに書込みまたは読出しを行ってパワーアップ・シーケンスをトリガします。パワーアップ・シーケンスの厳密な順序は、EBIU_SDGCTLレジスタのPSMビットによって決定されます。

Load Mode Registerコマンドでは、以下のパラメータを初期化します。

- バースト長＝1、ビット2～0、常に0
- ラップ・タイプ＝順次、ビット3、常に0
- Ltmode＝遅延モード（CAS遅延）、ビット6～4、EBIU_SDGCTLレジスタでプログラム可能

- ビット14～7、常に0

Load Mode Registerコマンドの実行中、未使用のアドレス・ピンは0に設定されます。Load Mode Registerコマンドの後に続く2クロック・サイクル中に、SDCはNOPコマンドだけを発行します。

拡張モード・レジスタを内蔵する低消費電力のモバイルSDRAMにおいて、EBIU_SDGCTLレジスタのEMRENビットがセットされている場合には、パワーアップ・シーケンス中にこのレジスタがプログラムされます。

拡張モード・レジスタは、以下のパラメータで初期化されます。

- パーシャル・アレイ・セルフリフレッシュ、ビット2～0、ビット2は常に0、ビット1～0はEBIU_SDGCTLでプログラマブル
- 温度補償セルフリフレッシュ、ビット4～3、ビット3は常に1、ビット4はEBIU_SDGCTLでプログラマブル
- ビット12～5、常に0、ビット13は常に1

▶ Read/Write コマンド

Read/Writeコマンドは、次の読出し／書込みアクセスが現在のアクティブ・ページにある場合に実行されます。Readコマンドの処理中に、SDRAMは列アドレスをラッチします。ActivateコマンドとReadコマンドとの間の遅延は、t_{RCD}パラメータによって決定されます。データは、CAS遅延が満たされた後でSDRAMから入手できます。

Writeコマンドでは、SDRAMは列アドレスをラッチします。書込みデータも同じサイクルで有効です。ActivateコマンドとWriteコマンドとの間の遅延は、t_{RCD}パラメータによって決定されます。

SDCは、SDRAMの自動プリチャージ機能を使用しません。この機能は、ReadまたはWriteコマンドの処理中に、SA10をハイレベルでアサートすることによってイネーブルにされます。

▶ Auto-Refresh コマンド

Auto-Refreshコマンドが出されると、SDRAMはリフレッシュ・アドレス・カウンタを内部的にインクリメントし、そのアドレスに対してCAS before RAS（CBR）リフレッシュを内部的に実行します。SDCリフレッシュ・カウンタがタイム・アウトした後、SDCはAuto-Refreshコマンドを生成します。SDRAMタイミング仕様で指定されたt_{REF}期間内にすべてのアドレスがリフレッシュされるように、SDRAM更新レート・コントロール・レジスタのRDIV値を設定する必要があります。このコマンドは、イネーブルされているかどうかとは無関係に外部バンクに発行されます（SDRAMメモリ・グローバル・コントロール・レジスタのEBE）。Auto-Refreshコマンドを実行する前に、SDCは外部バンクに対してPrecharge Allコマンドを実行します。次のActivateコマンドは、t_{RFC}仕様（$t_{RFC} = t_{RAS} + t_{RP}$）が満たされるまでは出されません。

Auto-Refreshコマンドは、パワーアップ・シーケンスの一部としてSDCによっても発行され、Self-Refreshモードの終了後にも発行されます。

▶ Self-Refresh コマンド

Self-Refreshコマンドによって、リフレッシュ動作は外部制御なしで、SDRAMによって内部的に実行されます。つまり、SDRAMがSelf-Refreshモードにある間は、SDC は Auto-Refresh サイクルを生成しません。Self-Refreshコマンドを実行する前に、すべての内部バンクはプリチャージされます。Self-Refreshモードをイネーブルにするには、SDRAMメモリ・グローバル・コントロール・レジスタ（EBIU_SDGCTL）のSRFSビットに1を書き込みます。Self-Refreshコマンドを発行した後、SDCはSCKEをローレベルに駆動します。これによって、SDRAMはパワーダウン・モードになります（SCKE＝0、$\overline{\text{SRAS}}$/$\overline{\text{SMS}}$/$\overline{\text{SCAS}}$/$\overline{\text{SWE}}$＝$\overline{1}$）。Self-Refreshモードを終了する前に、SDCはSCKEをアサートします。SDRAMは、少なくともt_{RAS}の期間、そして SDRAM 空間への内部アクセスが行われるまではSelf-Refreshモードにとどまります。内部アクセスが行われて、SDCがSDRAMをSelf-Refreshモードから終了させると、SDCはt_{XSR}仕様（t_{RFC}

= t_{RAS} + t_{RP}）を満たすように待機してから、Auto-Refresh コマンドを発行します。Auto-Refresh コマンドの後、SDC は t_{RFC} 仕様（t_{XSR} = t_{RAS} + t_{RP}）を満たすように待機してから、SDRAM に Self-Refresh モードを終了させた転送に対する Activate コマンドを実行します。したがって、Self-Refresh モードにある間に転送が SDC によって受信されてから、その転送に対して Activate コマンドが行われるまでの遅延は、2 ×（t_{RAS} + t_{RP}）となります。

なお、Self-Refresh モード中には、CLKOUT は SDC によってディスエーブルにされません。しかし、ソフトウェアは EBIU_SDGCTL の SCTLE ビットをクリアすることで、クロックをディスエーブルにできます。アプリケーション・ソフトウェアでは、コントローラに Self-Refresh モードを終了させる SDRAM アドレス空間への転送の前に、適用可能なすべてのクロック・タイミング仕様が満たされることを保証してください。SCTLE ビットがクリアされているときに、SDRAM アドレス空間への転送が発生した場合には、内部バス・エラーが生成され、アクセスは外部的に行われず、SDRAM は Self-Refresh モードに残されます。詳細については、17-9 ページの「エラー検出」を参照してください。

▶ No Operation/Command Inhibit コマンド

SDRAM への無動作（NOP）コマンドは、現在進行中の動作に影響を与えません。Command Inhibit コマンドは、NOP コマンドと同じです。しかし、SDRAM はチップ選択されません。SDC が SDRAM にアクティブにアクセスしており、効果のないコマンドを追加挿入する必要がある場合、NOP コマンドが出されます。SDC が SDRAM にアクセスしていない場合、Command Inhibit コマンドが出されます。

■ SDRAM のタイミング仕様

さまざまなSDRAMベンダーの主要なタイミング条件とパワーアップ・シーケンスに対応するため、SDCは、t_{RAS}、t_{RP}、t_{RCD}、t_{WR}、およびパワーアップ・シーケンス・モードが設定可能になっています（詳細については、17-34ページの「EBIU_SDGCTLレジスタ」を参照）。CAS遅延は、動作周波数に基づいて、EBIU_SDGCTLレジスタでプログラムしてください。（詳細については、SDRAMベンダーのデータシートを参照してください。）

この他のパラメータについては、SDCでは次のように想定します。

- バンク・サイクル時間：$t_{RC} = t_{RAS} + t_{RP}$
- リフレッシュ・サイクル時間：$t_{RFC} = t_{RAS} + t_{RP}$
- セルフリフレッシュ終了時間：$t_{XSR} = t_{RAS} + t_{RP}$
- Load Mode RegisterからActivateまでの時間：t_{MRD}またはt_{RSC}＝3クロック・サイクル
- ページ・ミスの損失＝$t_{RP} + t_{RCD}$

■ SDRAM Performance

7-12ページの表7-2は、16ビット幅SDRAMへのコアまたはDMAの読出し／書込みアクセスに対する、データ・スループット・レートを示します。この例で、すべてのサイクルはSCLKサイクルであり、次のSCLK周波数とSDRAMパラメータが使用されると想定します。

- SCLK周波数＝133MHz
- CAS遅延＝2サイクル（CL = 2）
- SDRAMバッファリングなし（EBUFE = 0）
- RASプリチャージ（t_{RP}）＝2サイクル（TRP = 2）

- RASからCAS遅延（t_{RCD}）＝2サイクル（TRCD = 2）
- アクティブ・コマンド時間（t_{RAS}）＝5サイクル（TRAS = 5）

外部バッファ・タイミング（SDRAMメモリ・グローバル・コントロール・レジスタのEBUFE = 1）またはCAS遅延＝3（SDRAMメモリ・グローバル・コントロール・レジスタのCL = 11）、あるいはその両方が使用された場合、すべてのアクセスで、選択された機能ごとに1サイクルだけ余分に必要となります。

バス要求と許可

プロセッサは、データとアドレス・バスの制御を外部デバイスに明け渡すことができます。プロセッサがそのメモリ・インターフェースを3ステートにすることで、外部コントローラは、非同期または同期の外部メモリ・パーツにアクセスできます。

■ 動作

外部デバイスがバスへのアクセスを必要とするとき、バス要求（$\overline{\text{BR}}$）信号をアサートします。$\overline{\text{BR}}$信号は、EAB要求と調停されます。保留中の内部要求がない場合には、外部バス要求は許可されます。プロセッサは、次のようにしてバス許可を実行します。

- データ・バス、アドレス・バス、非同期メモリ制御信号を3ステートにします。同期メモリ制御信号は、オプションで3ステートにできます。
- バス許可（$\overline{\text{BG}}$）信号をアサートします。

バスが外部デバイスに許可され、外部メモリに対して命令フェッチやデータ読出し／書込み要求が行われた場合、プロセッサはプログラム実行を停止させることがあります。外部デバイスが$\overline{\text{BR}}$を解放すると、プロセッサは$\overline{\text{BG}}$をアサート解除して、停止したポイントから実行を続行します。

プロセッサは、別の外部ポート・アクセスを開始する準備ができたときに $\overline{\text{BGH}}$ ピンをアサートしますが、バスはすでに許可されているので、プロセッサは阻止されます。

バスが許可された場合、SDSTAT レジスタの BGSTAT ビットはセットされます。外部バス許可によって遅延の生じるトランザクションの開始を回避するために、プロセッサはこのビットを使用してバス・ステータスをチェックします。

第18章　システム設計

この章には、ユーザがBlackfinプロセッサをベースとするシステムを開発する際に役立つハードウェア、ソフトウェアおよびシステム設計に関する情報を記載しています。システム内に実装される設計オプションは、コスト、性能、およびシステム要件に応じて異なります。多くの場合、この章で取り上げる設計に関する事項は、本マニュアルの他の節で詳細に説明されています。このようなケースに該当する場合には、この章で説明を繰り返すのではなく、該当する節を参照するよう示しています。

ピンの説明

160ピンPBGAパッケージのピン番号を含めて、ピンに関する詳細については、『ADSP-BF531/ADSP-BF532/ADSP-BF533 Embedded Processor Data Sheet』を参照してください。

■ 未使用ピンに関する推奨処置

ピンの詳細については、『ADSP-BF531/ADSP-BF532/ADSP-BF533 Embedded Processor Data Sheet』を参照してください。

プロセッサのリセット

$\overline{\text{RESET}}$ピンを使用して設定するハードウェア・リセット・モードに加えて、このプロセッサはいくつかのソフトウェア・リセット・モードも提供します。各種のモードに関する詳細については、3-13ページの「システム・リセットとパワーアップ」を参照してください。

リセット後のプロセッサの状態については、3-11ページの「リセット状態」で説明しています。

プロセッサのブート

各種の方法でプロセッサをブートできます。その方法として、外部16ビット・メモリからの実行、8ビットのフラッシュ・メモリからコードをロードするように設定されたROMからのブート、またはシリアルROM（8ビット、16ビット、または24ビットのアドレス範囲）からのブートがあります。このブート・モードに関する詳細については、3-19ページの「ブート方式」を参照してください。

図 18-1と図 18-2は、それぞれ8ビットと16ビットのブーティングに必要とされる接続を図示しています。8ビットと16ビットの両方のペリフェラルに対して、同じ方法でアドレス接続が行われる点に注意してください。バイト幅メモリを使用する場合にアクセスできるのは、各16ビット・ワードの下位バイトのみに限られます。

例えば、コア読出し動作の場合には、以下のようになります。

```
R0 = W[P0] (Z) ;    //P0は16ビットのアライメントされたASYNCメモリ位置をポイントします
```

8ビット・デバイスから読み出された実際の値は、R0の下位8ビットのみに含まれます。

コア書込み動作の場合には、以下のようになります。

```
W[P0] = R0.L ;    //P0は16ビットのアライメントされたASYNCメモリ位置をポイントします
```

8ビット・デバイスに書き込まれる8ビット値が、R0の下位バイトに最初にロードされることが必要です。

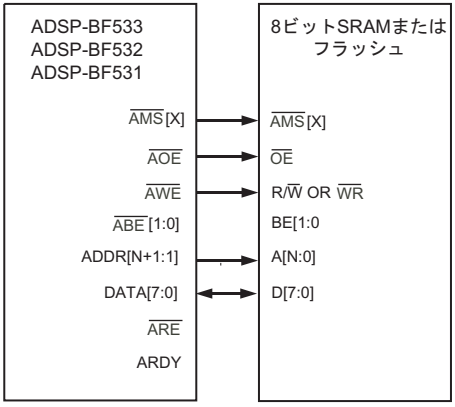


図 18-1. 8 ビット SRAM またはフラッシュとのインターフェース

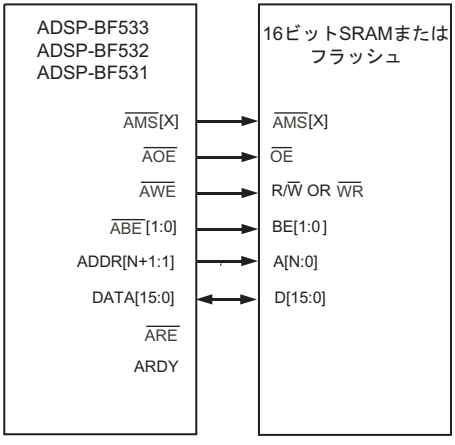


図 18-2. 16 ビット SRAM またはフラッシュとのインターフェース

クロックの管理

システムは水晶発振子を使用するか、または外部の水晶発振子から供給されるバッファされた成形クロックを使用して、クロック入力を駆動することができます。外部クロックは、プロセッサのCLKINピンに接続します。通常の動作時に規定値よりも低い周波数でCLKINを停止、変更、または動作させることはできません。プロセッサはクロック入力（CLKIN）を使用して、オンチップ・クロックを生成します。これらのクロックには、コア・クロック（CCLK）とペリフェラル・クロック（SCLK）が含まれます。

■ コア・クロックとシステム・クロックの管理

プロセッサはCLKINピン上に供給されるクロック入力を逓倍することによって、PLLのVCOクロックを生成します。このVCOクロックが分周されて、コア・クロック（CCLK）とシステム・クロック（SCLK）が生成されます。コア・クロックは、PLL_DIVレジスタのCSELビットの設定でプログラミングされる分周比に基づいて生成されます。システム・クロックは、PLL_DIVレジスタのSSELビットの設定でプログラミングされる分周比に基づいて生成されます。CCLKとSCLKの各周波数を設定および変更する方法に関する詳細については、第8章「ダイナミック・パワー・マネジメント」を参照してください。

割込みの設定とサービス

各種の割込みを利用できます。これには、コア割込みとペリフェラル割込みの両方が含まれています。プロセッサは、デフォルトのコア優先順位をシステム・レベルの割込みに割当てます。ただし、システム割込み割当てレジスタ（SIC_IARx）を使用して、これらのシステム割込みのマッピング変更を行うことが可能です。この詳細については、4-32ページの「システム割込み割当てレジスタ（SIC_IARx）」を参照してください。

プロセッサ・コアは、ネストされた割込みとネストされない割込みを提供します。各種モードのサービス・イベントに関する説明については、4-52ページの「割込みのネスティング」を参照してください。

セマフォ

セマフォは、同じシステム内部で実行される複数のプロセッサまたはプロセス／スレッド間の通信に対応するメカニズムを提供します。セマフォは、リソースの共有を調整する目的に利用されます。例えば、ある1つのプロセスが1つの特定リソースを使用しているときに、別のプロセスがその同じリソースの利用を要求する場合、最初のプロセスが該当リソースの使用をすでに完了していることを通知するまで、そのプロセスは待機しなければなりません。このシグナリングがセマフォによって達成されます。

セマフォのコヒーレンシは、テストおよびセットバイト（アトミック）命令（TESTSET）の使用によって保証されます。TESTSET命令は、以下の機能を実行します。

- Pレジスタによってポイントされたメモリ位置にハーフ・ワードをロードします。Pレジスタのアライメントは、ハーフ・ワードの境界で実行される必要があります。
- 値がゼロに等しい場合に、CCを設定します。
- 値をその最初の場所に戻して保存します（ただし、下位バイトの最上位ビット（MSB）は1に設定されます）。

TESTSETによってトリガされるイベントは、アトミック・オペレーションです。アドレスが指定されるメモリに対応するバスが取り込まれ、このバスはストア・オペレーションが完了するまで解放されません。マルチスレッド・システムでは、セマフォの整合性を維持するために、TESTSET命令が必要です。

ストア・オペレーションがストアまたは書込みバッファを通して確実にフラッシュされることを保証するために、セマフォを解放した直後にSSYNC命令を発行してください。

TESTSET命令を利用して、バイナリ・セマフォを実装するか、またはその他の任意タイプの除外メソッドを実装することができます。TESTSET命令は、マルチサイクルのバス・ロック・メカニズムに対して要求されるシステム・レベルの条件をサポートします。

プロセッサは、TESTSET命令の使用を外部メモリ領域のみに制限します。この領域以外のメモリ・マップ・エリアに対してTESTSET命令を使用すると、挙動の信頼性が損なわれる結果になる可能性があります。

■ クエリ・セマフォのコード例

リスト 18-1は、共有リソースが使用できるかをチェックするクエリ・セマフォのコード例を記載しています。

リスト 18-1. クエリ・セマフォ

```
/* クエリー・セマフォ。その値が非ゼロの場合に、「ビジー」であることを示します。ビジー状態が解除されるまで（またはスレッドのスケジュール変更が行われるまで ― 以下の注を参照）待ってください。P0はセマフォのアドレスを保持します */
QUERY:
TESTSET ( P0 ) ;
IF !CC JUMP QUERY ;
/* この時点で現在のスレッドに対してセマフォが許可され、[P0]のセマフォの値が非ゼロであるため、競合している他のすべてのスレッドが後回しにされます。現在のスレッドはthread_idをセマフォの場所に書き込み、現在のリソース所有者を示すことができます */
R0.L = THREAD_ID ;
B[P0] = R0 ;
```

```
/* 共有リソースを用いた実行が完了した時点で、ゼロ・バイトを [P0] に書き込みます
*/
R0 = 0 ;
B[P0] = R0 ;
SSYNC ;
/* 注：QUERYループでビジー・アイドリングを使用するのではなく、オペレーティン
グ・システム・コールを使用して、現在のスレッドのスケジュール変更を行うことができます */
```

データ遅延、レイテンシおよびスループット

DMAおよび外部メモリ・バスのレイテンシと性能の見積り評価に関する詳細については、第7章「チップ・バス階層」を参照してください。

バスの優先順位

各種の内部バス間における優先順位に関する説明については、第7章「チップ・バス階層」を参照してください。

外部メモリの設計に関する事項

この節では、外部メモリの設計に関する事項について説明します。

■ 非同期メモリ・インターフェースの例

この節では、16ビット幅のSRAMとのグルーレス接続を紹介します。このインターフェースでは、ARDYを外部からアサートする必要がありません。その理由は、メモリのアクセス時間を確定化するうえで、内部の待ち状態カウンタが十分に対応するためです。

16ビット・メモリをサポートするために必要とされるシステムの相互接続を図 18-3に示します。

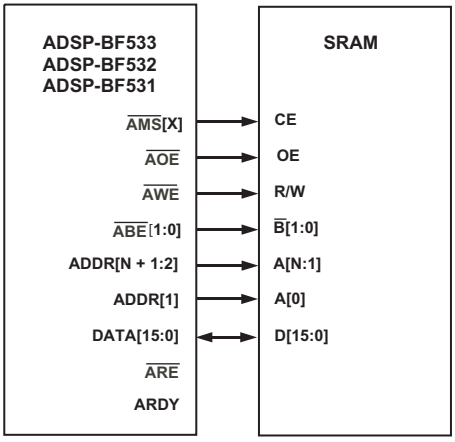


図 18-3. 16 ビット SRAM とのインターフェース

■ 16M バイトよりも小さい SDRAM の利用

ADSP-BF531/ADSP-BF532/ADSP-BF533では、16Mバイトよりも小さいSDRAMを使用することが可能です。ただし、その際には結果としてメモリ・マップが変更される方法を理解する必要があります。2Mバイトの SDRAM（512K×16ビット×2バンク）が外部メモリ・インターフェースにマッピングされる例を図 18-4に示します。この例では、各バンクあたり11個の行アドレスと8個の列アドレスが割当てられています。表 17-5を参考にすると、8個の列アドレスが割当てられたデバイスで利用可能な最低のバンク・サイズ（16Mバイト）は、2本のバンク・アドレス・ライン（IA[23:22]）と13本の行アドレス・ライン（IA[21:9]）で構成されます。したがって、例示するSDRAMとの接続時には、1本のプロセッサ・バンク・アドレス・ラインと2本の行アドレス・ラインが使用されません。

そのために、プロセッサの外部メモリ・マップでエイリアシングが引き起こされ、これが原因でSDRAMがプロセッサのメモリ・スペースの非連続的領域にマッピングされる結果になります。

図 18-4に示す表を参考にして、表の各行が2^{19}バイト、すなわち512Kバイトに相当することを確認してください。このような方法により、2MバイトSDRAMのマッピングは、図の左側のメモリ・マッピングで示すようにBlackfinメモリで非連続的となります。

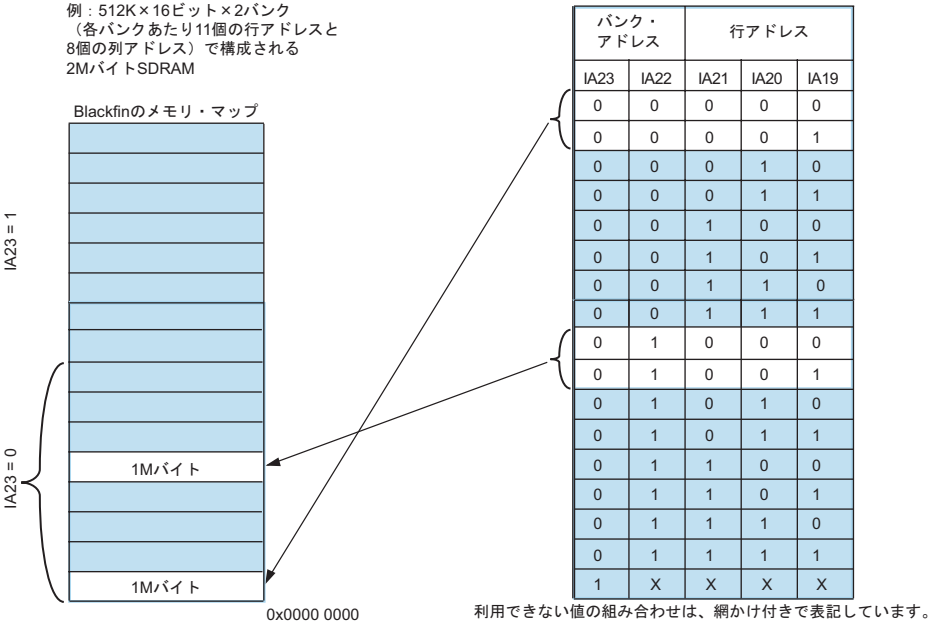


図 18-4. 容量の小さい SDRAM の使用

■ PLL 遷移時における SDRAM のリフレッシュ管理

プロセッサのSDRAMのリフレッシュ・レートはSCLKの周波数に基づくので、SDRAMを設定した後でSCLKの周波数を下げると、リフレッシュ・レートの設定が不適切になり、そのためにSDRAMに保存されるデータが損なわれる可能性があります。ただし、SDRAMを設定した後でSCLKの周波数を高くする場合には、プロセッサが不必要に高い速度でメモリをリフレッシュするだけにとどめられるので、結果としてSDRAMの利用効率が低下するのみに過ぎません。

SDRAMを使用するシステムで、PLLのVCO周波数を変更するための推奨手順は、以下のとおりです。

1. SSYNC 命令を発行して、現在実行中のメモリ動作がすべて完了していることを確認します。
2. EBIU_SDGCTL の SRFS ビットに 1 を書き込んで、SDRAM を自己リフレッシュ・モードに設定します。
3. 希望する PLL プログラミング・シーケンスを実行します（詳細については、第 8 章「ダイナミック・パワー・マネジメント」を参照してください）。
4. PLLが新しいVCO周波数にセトリングしたことを示すウェイクアップが発生した後で、SDRAM のリフレッシュ・レート・コントロール・レジスタ（EBIU_SDRRC）で新しい SCLK 周波数に適した値を設定して、そのプログラミング変更を実行します。
5. EBIU_SDGCTL の SRFS ビットをクリアして、SDRAM の自己リフレッシュ・モード設定を解除します。SDRAM モード・レジスタを変更する必要がある場合には、変更する値を EBIU_SDGCTL にも同様に書き込んで、PSSE ビットが設定されていることを確認してください。

VCO周波数を実際に変更するのではなく、PLL_DIV のSSEL ビットを使用してSCLK周波数を変更するときには、以下のステップに従ってください。

1. SSYNC 命令を発行して、現在実行中のメモリ動作がすべて完了していることを確認します。
2. EBIU_SDGCTL の SRFS ビットに 1 を書き込んで、SDRAM を自己リフレッシュ・モードに設定します。
3. 希望の値を SSEL ビットに書込みます。
4. SDRAM のリフレッシュ・レート・コントロール・レジスタ（EBIU_SDRRC）で新しい SCLK 周波数に適した値を設定して、そのプログラミング変更を実行します。

5. EBIU_SDGCTL の SRFS ビットをクリアして、SDRAM の自己リフレッシュ・モード設定を解除します。SDRAM モード・レジスタを変更する必要がある場合には、変更する値を EBIU_SDGCTL にも同様に書き込んで、PSSE ビットが設定されていることを確認してください。

SCLK をもっと高い値に変更する場合、厳密にはステップ 2 と 4 は必要ありませんが、SCLK を下げるときには、この各ステップを必ず実行しなければならない点に注意してください。

SDRAM のリフレッシュに関する詳細については、第 17 章「SDRAM コントローラ（SDC）」の「外部バス・インターフェース・ユニット」を参照してください。

■ バス競合の回避

3 ステートのデータ・バスは、システム内部の数多くのデバイスによって共有されるので、競合を回避するように注意してください。競合が起こると、消費電力が過剰に増加し、デバイスの動作不良が発生する可能性があります。競合は、1 つのデバイスのバス接続が切断されようとしているときに別のデバイスのバス接続が行われると、その間に発生します。最初のデバイスが低速で 3 ステートの状態に入るときに、2 番目のデバイスが高速に駆動する場合、この 2 個のデバイスのバス競合が発生します。

競合の発生には、2 つのケースがあります。最初のケースは、同じメモリ・スペースで読出し動作が行われた後で、書込み動作が行われる場合です。このケースでは、データ・バス・ドライバが、読出し動作によってアドレシングされたメモリ・デバイスのドライバと競合する可能性があります。2 番目のケースは、2 つの異なるメモリ・スペースから連続的に読出し動作が実行される場合です。このケースでは、最初の読出し動作が終了して次の読出し動作が開始されるまでの間に、2 回の読出し動作でアドレシングされる 2 個のメモリ・デバイスが競合する可能性があります。

この競合を回避するために、非同期メモリ・バンク・コントロール・レジスタでターンアラウンド時間（バンク遷移時間）を適切にプログラミングしてください。この機能性により、このようなタイプのアクセス間クロック・サイクル数をソフトウェアにより各バンク単位で設定することができます。外部バス・インターフェース・ユニット（EBIU）は、最低限1サイクルで遷移を発生するように設定します。

高周波数設計に関する考慮事項

プロセッサは非常に高速のクロック周波数による動作が可能であるため、回路基板の設計とレイアウトを行う際には、シグナル・インテグリティとノイズの問題を考慮に入れることが必要です。以下の各節では、これらの考慮すべき事項について説明し、信号処理システムの設計とデバッグを実施する際に利用する各種の技法を提案します。

■ シリアル・ポート上のポイント・ツー・ポイント接続

シリアル・ポートは低速で動作しますが、その場合でも出力ドライバはエッジ・レートが高速であるために、伝送距離が比較的長いときには、ドライバに信号源終端が要求される場合があります。

ポイント・ツー・ポイント接続用のピンの近くに、1本の直列終端抵抗を追加してください。シリアル・ポートを使用するアプリケーションでは通常、距離が6インチを超えるときに、この終端方式が利用されます。伝送ラインの終端に関する推奨方法の詳細については、18-17ページの「推奨する参考文献」を参照してください。さらに、出力ドライバの立上がり時間と立下がり時間のデータに関しては、『ADSP-BF531/ADSP-BF532/ADSP-BF533 Embedded Processor Data Sheet』を参照してください。

■ シグナル・インテグリティ

高速信号にかかる容量性負荷を可能な限り低減してください。バスの負荷については、待ち状態で動作するデバイス（たとえば、DRAM）に対してバッファを使用する方法で低減できます。この方法によって、ゼロ待ち状態のデバイスに接続される信号の容量が低減されるので、これらの信号のスイッチ動作が高速化されると同時に、ノイズが起因して発生する電流スパイクが低く抑えられます。

信号の配線長（インダクタンス）を可能な限り短くして、リンギングを低く抑えることも必要です。外部メモリや読出し、書込み、およびアクノレッジの各ストローブなどの特定信号に対しても、細心の注意を払ってください。

シグナル・インテグリティを推進する上でのその他の推奨および提案事項は、以下のとおりです。

- クロストークを低減するために、プリント回路基板（PCB）上に1枚以上のグラウンド・プレーンを使用してください。その際には、各グラウンド・プレーン間に数多くのビアを必ず使用します。これらのプレーンをPCBの中央の位置に配置してください。
- クロック、ストローブ、バス要求などの重要度の高い信号をグラウンド・プレーンに隣接する信号層に配置し、これらの信号をその他の重要度の低い信号と引き離し、この各信号が直角に交差するようにレイアウトして、クロストークを低く抑えるようにしてください。
- クロストークを低減し、さらにインピーダンスと遅延の制御を改善できるようにするために、伝送ラインのインピーダンスを低く抑える設計を行います。

- 回路基板を実験的にレイアウトし、反射の問題からクロストークとノイズの問題を切り離してください。その方法として、パルス発生器から信号配線を駆動し、その他の部品と信号が受動的であるときに、反射の問題を調べてください。

■ デカップリング・コンデンサとグラウンド・プレーン

グラウンドと電源にグラウンド・プレーンを使用することが必要です。図18-5に示すように、パッケージのVDDEXTとVDDINTの各ピンの非常に近い個所にコンデンサを外付けしてください。その際には、短く幅の広いパターン配線を使用します。コンデンサのグラウンド接続端をプロセッサのパッケージ・フットプリント内部のグラウンド・プレーン（ボード裏面のグラウンド・プレーンの真下）に直接的に接続してください。フットプリントの外部に接続しないでください。表面実装型のコンデンサはその直列インダクタンス値が低いので、このタイプのコンデンサの利用を推奨します。

可能な限り短いパターン配線を使用して、電源プレーンを電源ピンに直接接続してください。グラウンド・プレーンの効果が低下しないようにするために、ビアやパターン配線の処理の際に、グラウンド・プレーンに数多くの穴あけを行ってはいけません。これに加えて、容量の大きいタンタル・コンデンサをいくつかボード上に実装することも必要です。

設計には、バイパス・コンデンサを使用するか、またはバイパス・コンデンサとタンタル・コンデンサを組み合わせて使用することもできます。回路基板の設計に際しては、グラウンド・プレーンを通過するフィードスルーを最小限に抑えるように努めてください。

■ オシロスコープ・プローブ

高速計測を実施するときには、「差込み」タイプまたはこれと同様の短い（＜0.5インチ）グラウンド・クリップをオシロスコープ・プローブの先端部に必ず装着してください。このプローブは、負荷容量が3pF以下の低容

量の能動プローブとします。グラウンドのリード線が4インチの標準的なグラウンド・クリップを使用すると、表示されるトレース上にリンギングが確認され、過度のオーバーシュートとアンダーシュートが信号上に発生することが考えられます。信号を高い精度で表示するには、1GHz以上のサンプリング・オシロスコープが必要です。

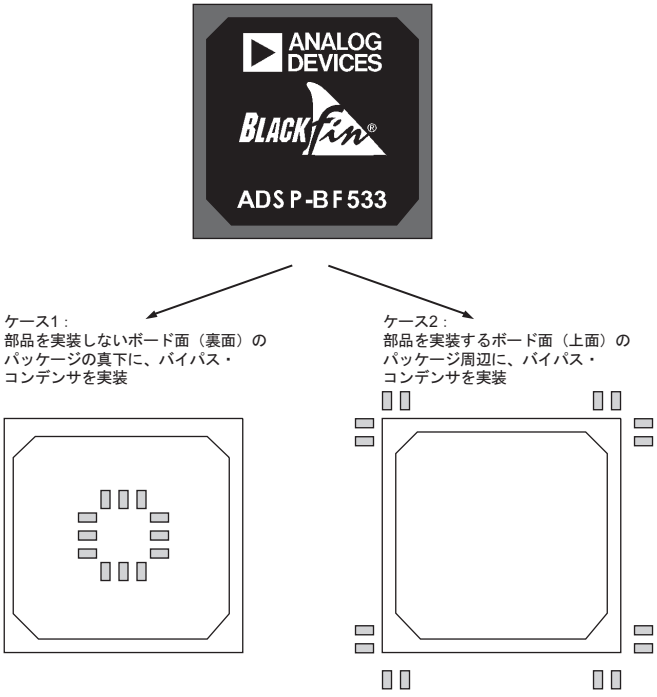


図 18-5. バイパス・コンデンサの配置

■ 推奨する参考文献

さらなる詳細については、*High-Speed Digital Design: A Handbook of Black Magic*（高速デジタル設計：ブラックマジック・ハンドブック）、Johnson & Graham、Prentice Hall, Inc.、ISBN 0-13-395724-1を参照してください。

このハンドブックは、最新の高周波数デジタル回路設計で遭遇する諸問題を取り扱う技術資料です。これは、実務的なアイデアを網羅した優れた情報源です。このハンドブックに掲載される題目として、以下の項目が含まれています。

- ロジック・ゲートの高速特性
- 計測技術
- 伝送ライン
- グラウンド・プレーンおよび層のスタッキング
- 終端
- ビア
- 電源システム
- コネクタ
- リボン・ケーブル
- クロックの分配
- クロック発振器

第19章 Blackfinプロセッサのデバッグ

Blackfinプロセッサのデバッグ機能は、ソフトウェアのデバッギングに利用されます。この機能はさらに、オペレーティング・システム（OS）のカーネルで見られる場合のあるいくつかのサービスの補完も行います。この機能性はプロセッサのハードウェアに実装され、複数のレベルにグループ化されます。

利用可能なデバッグ機能の要約を表19-1に示します。

表 19-1. Blackfin デバッグ機能

デバッグ機能	説明
ウォッチポイント	アドレス範囲、および条件が満たされた場合にプロセッサを停止するよう、その条件を指定します。
トレース履歴	プログラム・カウンタ最後の 16 の不連続値をオンチップのトレース・バッファに格納します。
サイクル・カウント	すべてのコード・プロファイリング機能に対応する機能性を提供します。
性能モニタリング	非割込み方式による内部リソースのモニタと計測を可能にします。

ウォッチポイント・ユニット

ウォッチポイント・ユニットは、命令バスとデータ・バス両方のアドレスをモニタすることによって、プログラムの挙動性をチェックするためのいくつかのメカニズムを提供します。このユニットは特定のアドレスが一致する回数をカウントした後で、このカウントに基づいてイベントをスケジュール化します。

これに加えて、ウォッチポイント・ユニットから提供される情報はコードの最適化に役立ちます。このユニットはさらに、コードのパッチングによる実行ファイルの保持を容易にします。

ウォッチポイント・ユニットには、スーパーバイザとエミュレータの各モード時にアクセス可能な以下のメモリマップド・レジスタ（MMR）が内蔵されています。

- ウォッチポイント・ステータス・レジスタ（WPSTAT）
- 6個の命令ウォッチポイント・アドレス・レジスタ（WPIA[5:0]）
- 6個の命令ウォッチポイント・アドレス・カウント・レジスタ（WPIACNT[5:0]）
- 命令ウォッチポイント・アドレス・コントロール・レジスタ（WPIACTL）
- 2個のデータ・ウォッチポイント・アドレス・レジスタ（WPDA[1:0]）
- 2個のデータ・ウォッチポイント・アドレス・カウント・レジスタ（WPDACNT[1:0]）
- データ・ウォッチポイント・アドレス・コントロール・レジスタ（WPDACTL）

以下の2つの動作によって、命令ウォッチポイントが実行されます。

- 6個の命令ウォッチポイント・アドレス・レジスタWPIA[5:0]の値が、命令バス上のアドレスと比較されます。
- アドレスが一致するごとに、命令ウォッチポイント・アドレス・カウント・レジスタWPIACNT[5:0]の該当するカウント値がデクリメントされます。

6個の命令ウォッチポイント・アドレス・レジスタは、さらに3つの範囲の命令アドレス範囲ウォッチポイントにグループ化されます。その範囲はWPIA0～WPIA1、WPIA2～WPIA3、WPIA4～WPIA5の各アドレスによって識別されます。

WPIA0、WPIA1、WPIA2、WPIA3、WPIA4、WPIA5に格納されるアドレス範囲は、以下の条件を満足する必要があります。

```
WPIA0 <= WPIA1

WPIA2 <= WPIA3

WPIA4 <= WPIA5
```

以下の2つの動作によって、データ・ウォッチポイントが実行されます。

- 2 個のデータ・ウォッチポイント・アドレス・レジスタWPDA[1:0]の値が、データ・バス上のアドレスと比較されます。
- アドレスが一致するごとに、データ・ウォッチポイント・アドレス・カウント・レジスタWPDACNT[1:0]の該当するカウント値がデクリメントされます。

2個のデータ・ウォッチポイント・アドレス・レジスタは、さらに1つのデータ･アドレス範囲ウォッチポイントWPDA[1:0]にグループ化されます。

命令およびデータ・カウント値レジスタには、ウォッチポイントが一致しなければならない回数から1を差し引いたデータがロードされることが必要です。カウント値がゼロに達した後でウォッチポイントが一致すると、例外またはエミュレーション・イベントがトリガされる結果になります。


イベントが発生した後で、カウント値を再初期化する必要がある点に注意してください。

命令ウォッチポイントとデータ・ウォッチポイントを組み合わせて、イベントをトリガすることも可能です。WPIACTL レジスタの WPAND ビットを1に設定すれば、命令アドレス・ウォッチポイントが一致し、これと同時にデータ・アドレス・ウォッチポイントが一致するときに限り、イベントがトリガされます。WPAND ビットを0に設定すると、イネーブルにされたウォッチポイントまたはウォッチポイント範囲のどれかが一致するときに、イベントがトリガされます。

ウォッチポイント・ユニットをイネーブルにするには、WPIACTL レジスタの WPPWR ビットの設定が必要です。WPPWR ビットを1に設定する場合には、WPIACTL と WPDACTL の各MMRで特定のイネーブル・ビットを使用して、個々のウォッチポイントおよびウォッチポイント範囲をイネーブルにすることが可能です。WPPWR ビットを0に設定すると、すべてのウォッチポイント・アクティビティがディスエーブルになります。

■ 命令ウォッチポイント

すべての命令ウォッチポイントは、表 19-2に示すようにWPIACTLレジスタの3つのビットによって制御されます。

表 19-2. WPIACTL 制御ビット

ビット名	説明
EMUSWx	命令アドレスが一致するときに、エミュレーション・イベントまたは例外イベントのどちらを発生させるかを決定します。
WPICNTENx	アドレスの一致回数をカウントする 16 ビット・カウンタをイネーブルにします。このカウンタをディスエーブルにすると、一致が検出されるたびにイベントが発生します。
WPIAENx	アドレスのウォッチポイント・アクティビティをイネーブルにします。

2つのウォッチポイントの関連付けによって、1つのウォッチポイント範囲が形成される場合には、表 19-3に示すように2つの追加ビットが使用されます。

表 19-3. WPIACTL ウォッチポイント範囲制御ビット

ビット名	説明
WPIRENxy	1 つのウォッチポイント範囲を形成するために関連付けられる 2 つのウォッチポイントを指定します。
WPIRINVxy	識別される範囲内または範囲外のどちらのアドレスによってイベントを発生させるかを決定します。

コードのパッチングにより、ソフトウェアで既存のコード・セクションを新しいコードに置き換えることができます。ウォッチポイント・レジスタを使用して、既存コードの開始アドレスで例外をトリガします。その後、例外ルーチンが、新しいコードを含むメモリ位置の方向に移動します。

プロセッサ上でコード・パッチングを実行するときには、既存コードの開始アドレスをWPIAnレジスタのどれか1つに書込み、該当するEMUSWxビットを設定することで、例外がトリガされます。例外サービス・ルーチンでWPSTATレジスタが読み出され、例外をトリガしたウォッチポイントが確認されます。次に、新しいコードの開始アドレスをRETXレジスタに書き込むと、例外から新しいコードにコードが返されます。コード・パッチングには例外メカニズムが使用されるので、優先順位が同じであるか、またはもっと高いイベント・サービス・ルーチン（例外、NMI、およびリセットの各ルーチン）をパッチングすることはできません。

WPSTAT MMRに書込みを行うと、スティッキーなステータス・ビットがすべてクリアされます。書き込まれたデータ値は無視されます。

■ WPIAn レジスタ

ウォッチポイント・ユニットをイネーブルにすると、命令ウォッチポイント・アドレス・レジスタ（WPIAn）の値が命令バス上のアドレスと比較されます。アドレスが一致するごとに、命令ウォッチポイント・アドレス・カウント・レジスタ（WPIACNTn）の該当するカウント値がデクリメントされます。

命令ウォッチポイント・アドレス・レジスタWPIA[5:0]を図 19-1に示します。

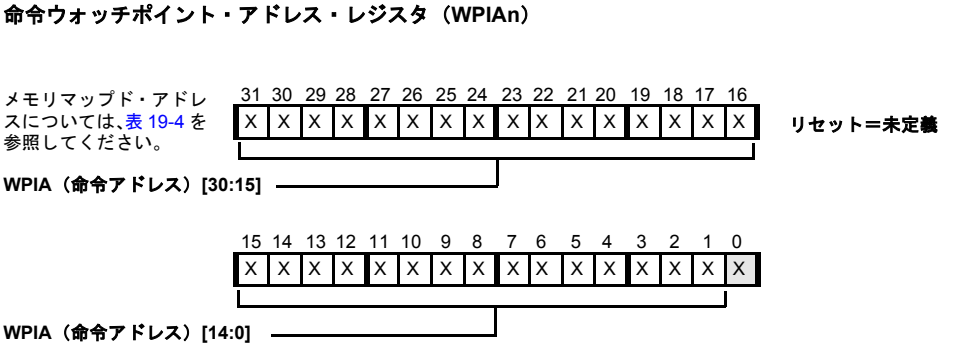


図 19-1. 命令ウォッチポイント・アドレス・レジスタ

表 19-4. 命令ウォッチポイント・アドレス・レジスタのメモリマップド・アドレス

レジスタ名	メモリマップド・アドレス
WPIA0	0xFFE0 7040
WPIA1	0xFFE0 7044
WPIA2	0xFFE0 7048
WPIA3	0xFFE0 704C
WPIA4	0xFFE0 7050
WPIA5	0xFFE0 7054

■ WPIACNTn レジスタ

ウォッチポイント・ユニットをイネーブルにすると、アドレスまたはアドレス・バスが`WPIAn`レジスタの値と一致するごとに、命令ウォッチポイント・アドレス・カウント・レジスタ（`WPIACNT[5:0]`）のカウント値がデクリメントされます。イベントをトリガする前に、ウォッチポイントが一致しなければならない回数から1を差し引いた値を`WPIACNTn`レジスタにロードしてください（図 19-2を参照）。プログラミングされたカウント値を経過すると、その後で`WPIACNTn`レジスタは0x0000にデクリメントします。

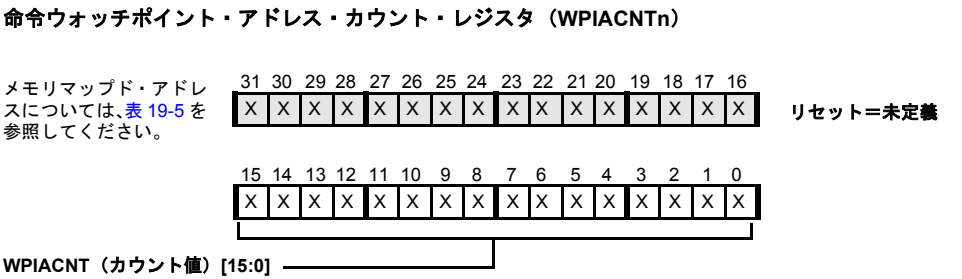


図 19-2. 命令ウォッチポイント・アドレス・カウント・レジスタ

表 19-5. 命令ウォッチポイント・アドレス・カウント・レジスタのメモリマップド・アドレス

レジスタ名	メモリマップド・アドレス
WPIACNT0	0xFFE0 7080
WPIACNT1	0xFFE0 7084
WPIACNT2	0xFFE0 7088
WPIACNT3	0xFFE0 708C
WPIACNT4	0xFFE0 7090
WPIACNT5	0xFFE0 7094

■ WPIACTL レジスタ

命令ウォッチポイント・アドレス・コントロール・レジスタ（WPIACTL）の3つのビットを使用して、各命令ウォッチポイントを制御します。このレジスタの上位ハーフのビット構成を図 19-3 で説明しています。19-10 ページの図 19-4 では、このレジスタの下位ハーフのビット構成を説明しています。このレジスタのビットに関する詳細については、19-5 ページの「命令ウォッチポイント」を参照してください。

WPPWR ビットを設定しない限り、WPIACTL レジスタのビットの設定効果はありません。

命令ウォッチポイント・アドレス・コントロール・レジスタ（WPIACTL）
範囲比較で IA は命令アドレスを示します。

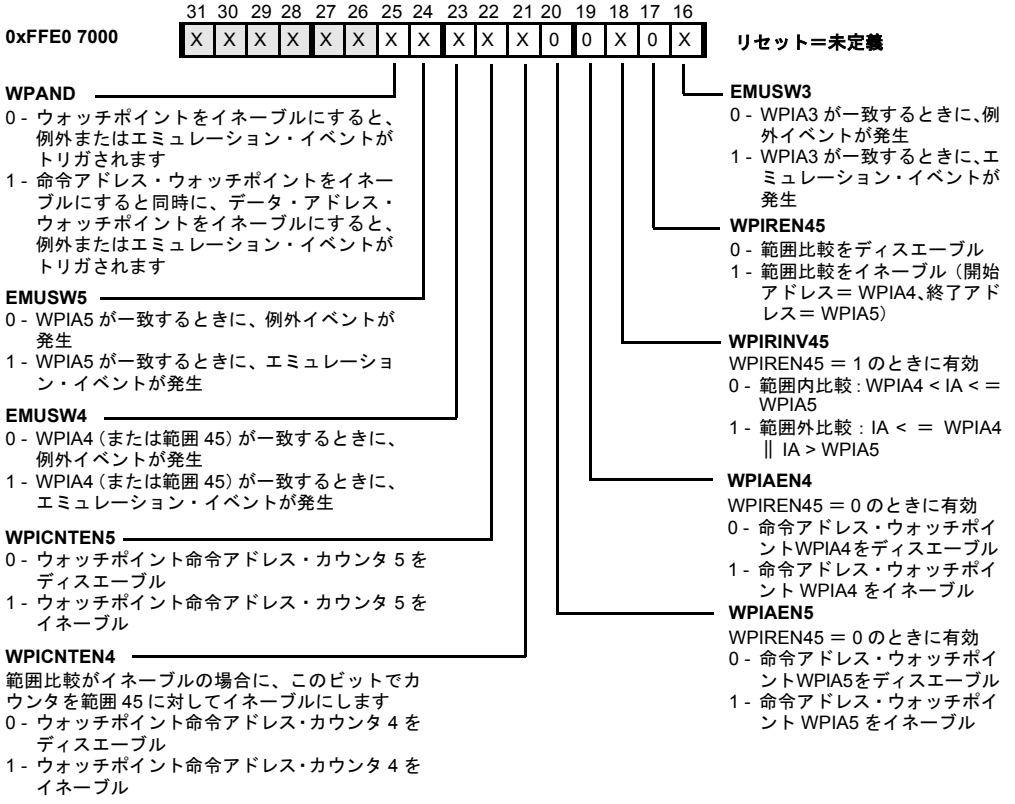


図 19-3. 命令ウォッチポイント・アドレス・コントロール・レジスタ（WPIACTL）[31:16]

命令ウォッチポイント・アドレス・コントロール・レジスタ（WPIACTL）
範囲比較で IA は命令アドレスを示します。

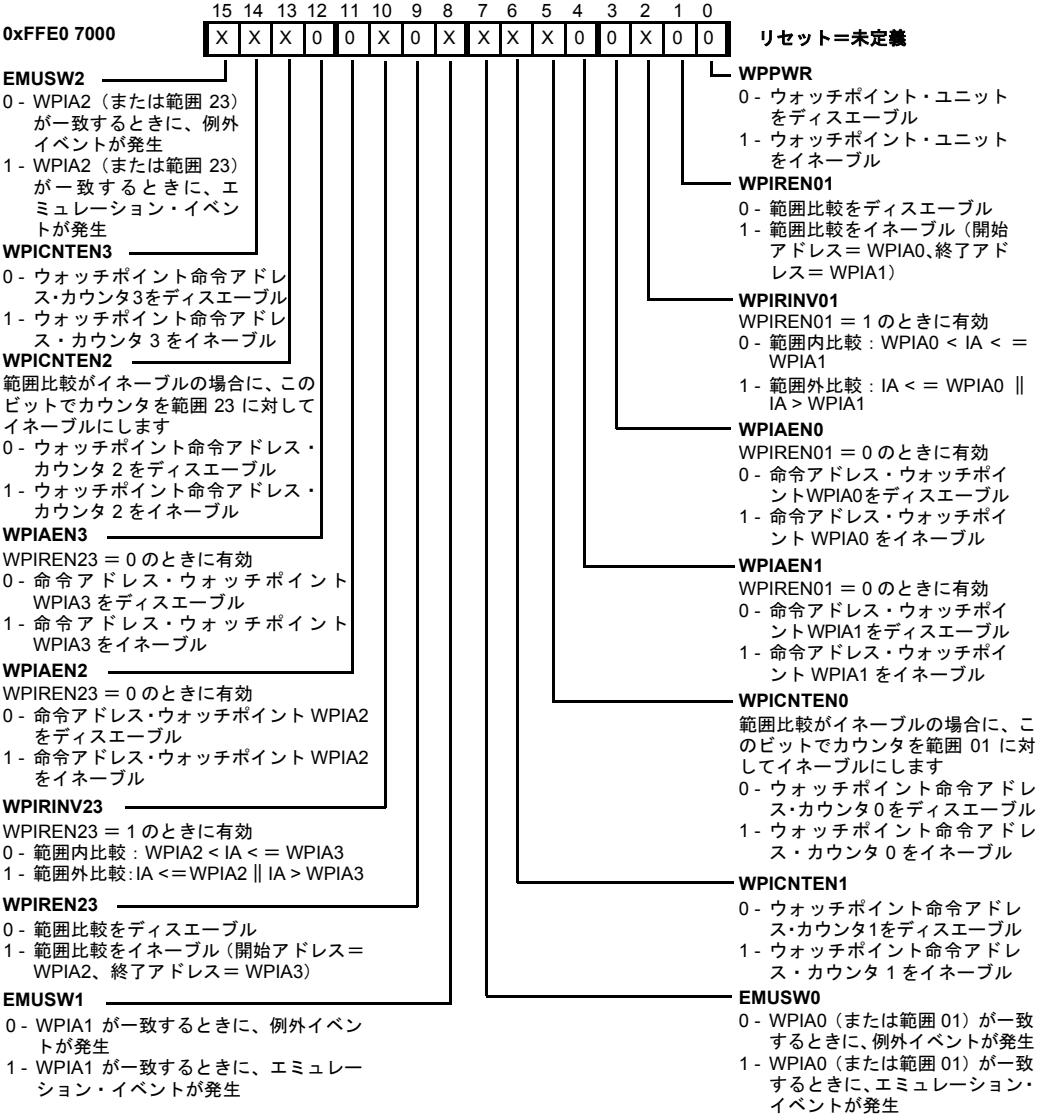


図 19-4. 命令ウォッチポイント・アドレス・コントロール・レジスタ（WPIACTL）[15:0]

■ データ・アドレス・ウォッチポイント

すべてのデータ・ウォッチポイントは、表 19-6に示すようにWPDACTLレジスタの4つのビットによって制御されます。

表 19-6. データ・アドレス・ウォッチポイント

ビット名	説明
WPDACCn	読出しまたは書込みアクセスのどちらに一致を適用するかを決定します。
WPDSRCn	ユニットによってモニタされる DAG を決定します。
WPDCNTENn	アドレスの一致回数をカウントするカウンタをイネーブルにします。このカウンタをディスエーブルにすると、一致が検出されるたびにイベントが発生します。
WPDAENn	アドレスのウォッチポイント・アクティビティをイネーブルにします。

2つのウォッチポイントの関連付けによって、1つのウォッチポイント範囲が形成される場合には、2つの追加ビットが使用されます。表 19-7を参照してください。

表 19-7. WPDACTL ウォッチポイント制御ビット

ビット名	説明
WPDREN01	1 つのウォッチポイント範囲を形成するために関連付けられる 2 つのウォッチポイントを指定します。
WPDRINV01	識別される範囲内または範囲外のどちらのアドレスによってイベントを発生させるかを決定します。

データ・アドレス・ウォッチポイントは、常にエミュレーション・イベントをトリガします。

■ WPDAn レジスタ

ウォッチポイント・ユニットをイネーブルにすると、データ・ウォッチポイント・アドレス・レジスタ（WPDAn）の値がデータ・バス上のアドレスと比較されます。アドレスが一致するごとに、データ・ウォッチポイント・アドレス・カウント・レジスタ（WPDACNTn）の該当するカウント値がデクリメントされます。

データ・ウォッチポイント・アドレス・レジスタWPDA[1:0]を図 19-5に示します。

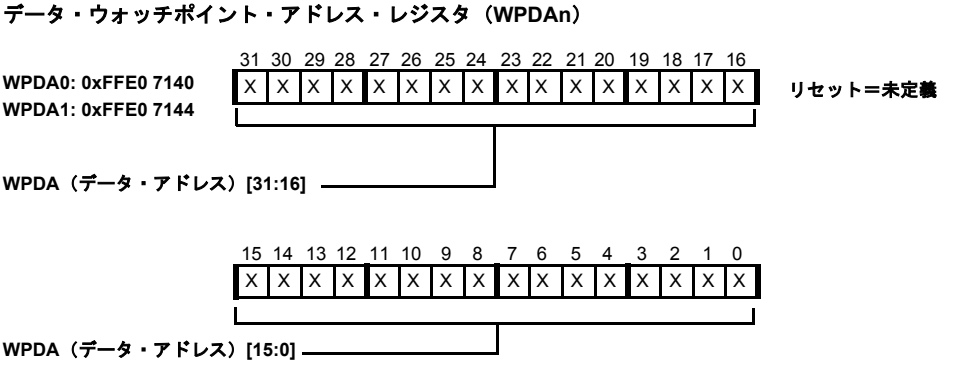


図 19-5. データ・ウォッチポイント・アドレス・レジスタ

■ WPDACNTn レジスタ

ウォッチポイント・ユニットをイネーブルにすると、アドレスまたはアドレス・バスがWPDAnレジスタの値と一致するごとに、データ・ウォッチポイント・アドレス・カウント値レジスタ（WPDACNTn）のカウント値がデクリメントされます。イベントをトリガする前に、ウォッチポイントが一致しなければならない回数から1を差し引いた値をWPDACNTnレジスタにロードしてください。プログラミングされたカウント値を経過すると、その後

でWPDACNTnレジスタは0x0000にデクリメントします。データ・ウォッチポイント・アドレス・カウント値レジスタWPDACNT[1:0]を図 19-6に示します。

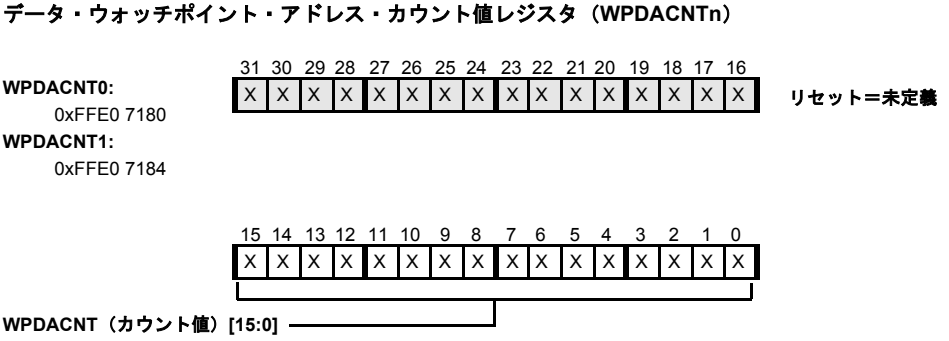


図 19-6. データ・ウォッチポイント・アドレス・カウント値レジスタ

■ WPDACTL レジスタ

データ・ウォッチポイント・アドレス・コントロール・レジスタ（WPDACTL）のビットに関する詳細については、19-11ページの「データ・アドレス・ウォッチポイント」を参照してください。

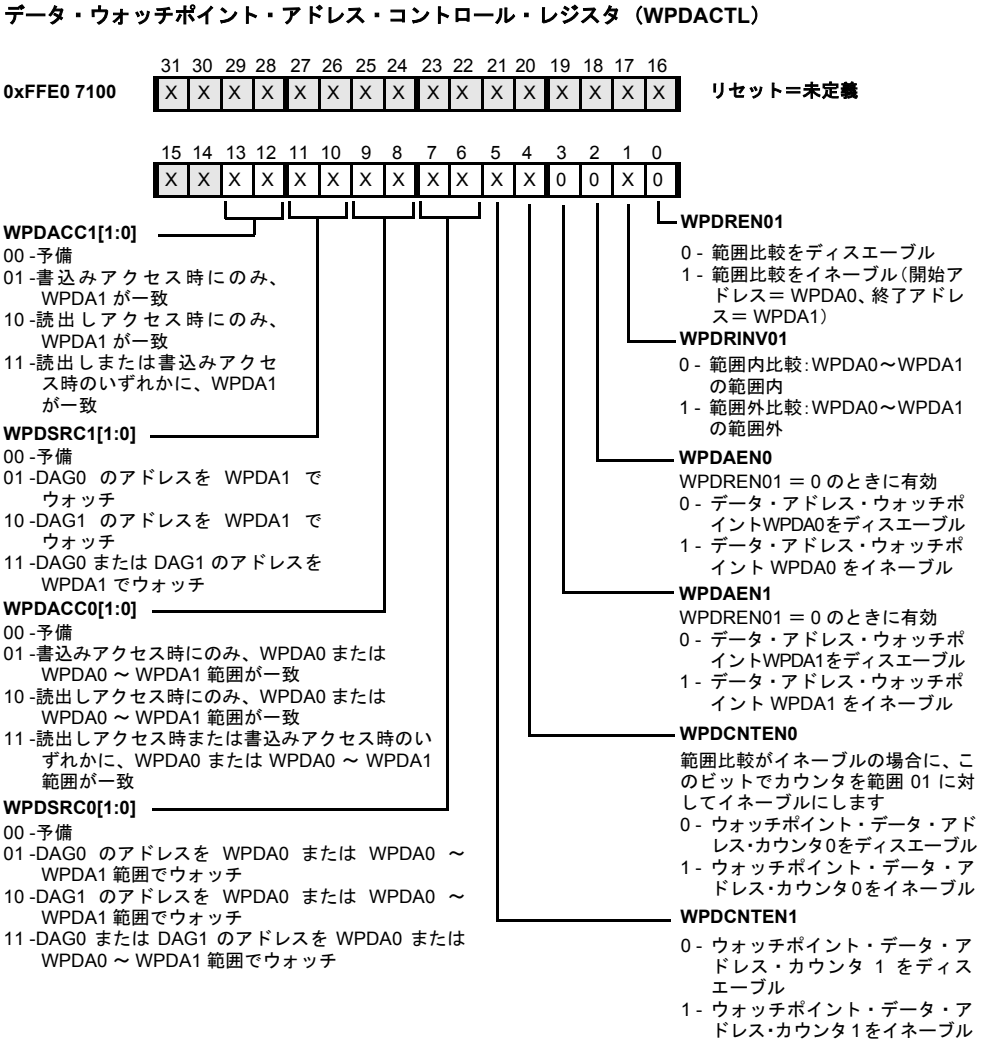


図 19-7. データ・ウォッチポイント・アドレス・コントロール・レジスタ

■ WPSTAT レジスタ

ウォッチポイント・ステータス・レジスタ（WPSTAT）は、ウォッチポイントのステータスをモニタします。このレジスタの読出しおよび書込み動作を実行できるのは、スーパーバイザまたはエミュレータ・モード時のみに限られます。ウォッチポイントまたはウォッチポイント範囲が一致するときに、このレジスタはウォッチポイントのソースを反映します。WPSTAT レジスタのステータス・ビットはスティッキーであるため、値に関係なくこのレジスタに書込みを行うと、ステータス・ビットのすべてがクリアされます。

図 19-8にウォッチポイント・ステータス・レジスタを示します。

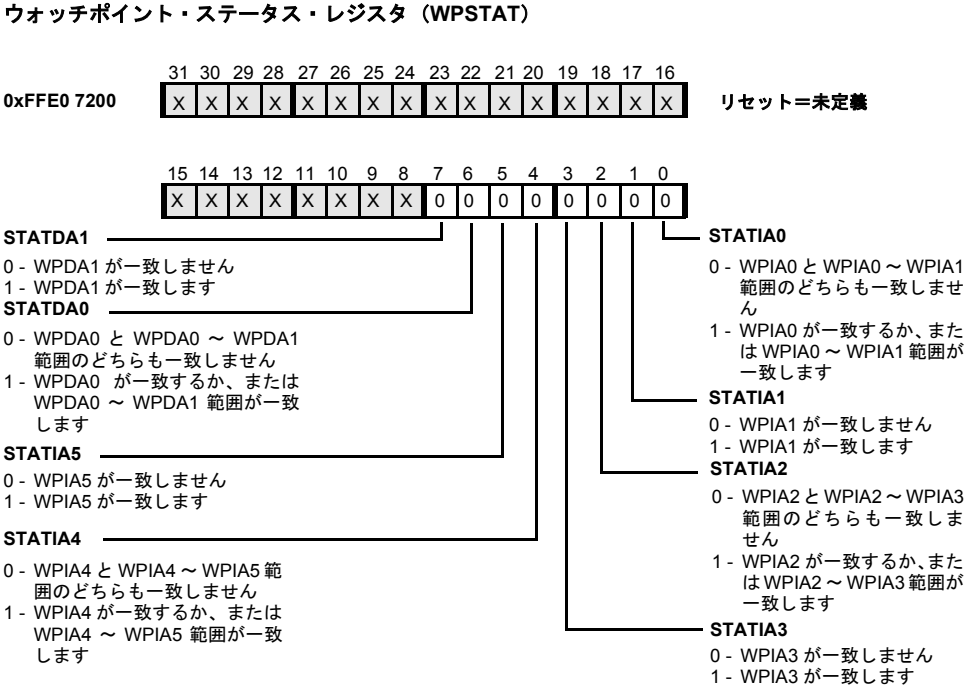


図 19-8. ウォッチポイント・ステータス・レジスタ

トレース・ユニット

トレース・ユニットには、プログラム・シーケンサによって取り込まれるプログラム・フローのうち最も新しい16の変更に関する履歴がストアされます。このような履歴の追跡により、ユーザはプログラム・シーケンサの最新のパスを再作成することができます。

トレース・バッファをイネーブルにして、その満杯時に例外を発生するように設定できます。例外に関連付けられる例外サービス・ルーチンが、トレース・バッファのエントリをメモリに保存します。したがって、トレース・バッファがイネーブルに設定された後以降のプログラム・シーケンサのパス全体を再作成することが可能です。

ゼロ・オーバーヘッド・ループによるプログラム・フローの変更は、トレース・バッファにストアされません。ゼロ・オーバーヘッド・ループ内で停止されるデバッギング・コードについては、LC0とLC1のループ・カウント・レジスタで繰返しカウントを実行できます。

最後のエントリ、または最後2つのエントリの1つと一致するプログラム・フロー変更の記録を省略するように、トレース・バッファを設定することが可能です。記録からこれらのエントリのいずれか1つを省略すると、プログラム内のループによるトレース・バッファのオーバーフローが防止されます。ゼロ・オーバーヘッド・ループはトレース・バッファに記録されないので、この機能を利用して、4段までネスティングされるループからのトレースのオーバーフローを防止することができます。

トレース・バッファ・レジスタ（TBUF）はその読出し動作時に、最大で16までのエントリが含まれるトレース・ユニット・スタックから先頭の値を返します。各エントリには、一組の分岐元および分岐先アドレスが含まれます。TBUFからの読出し動作時には、分岐先アドレスを先頭として最も新しいエントリが最初に返されます。その次の読出しで、分岐元アドレスが返されます。

TBUF に格納される有効エントリの数は、TBUFCNT レジスタの TBUFSTAT フィールドに保持されます。1 回の読出し動作が実行されるたびに、TBUFCNT はデクリメントします。各エントリは 2 個のデータに相当するので、TBUFCNT の読出しを合計 2 回行うと、TBUF レジスタは空き状態になります。

トレース・バッファの最後のエントリ 2 つのうち、いずれかと同じ不連続性は記録されません。

トレース・バッファからの読出しは破壊的動作であるため、コードの割込み不能セクションで TBUF の読出しを実行することを推奨します。

シングル・レベルの圧縮が実行されている場合には、分岐先アドレスの最下位ビット（LSB）が設定される点に注意してください。2 レベルの圧縮が実行されている場合には、分岐元アドレスの LSB が設定されます。

■ TBUFCTL レジスタ

トレース・バッファ・コントロール・レジスタ（TBUFCTL）の 2 つの制御ビットを使用して、トレース・ユニットをイネーブルにします。最初に、TBUFPWR ビットを設定してトレース・ユニットをアクティブにする必要があります。TBUFPWR = 1 の場合に TBUFEN を 1 に設定すれば、トレース・ユニットがイネーブルになります。

トレース・バッファ・コントロール・レジスタ（TBUFCTL）を図 19-9 で説明します。TBUFOVF = 1 の場合、トレース・ユニットは例外、NMI、およびリセットの各ルーチンの不連続性を記録しません。

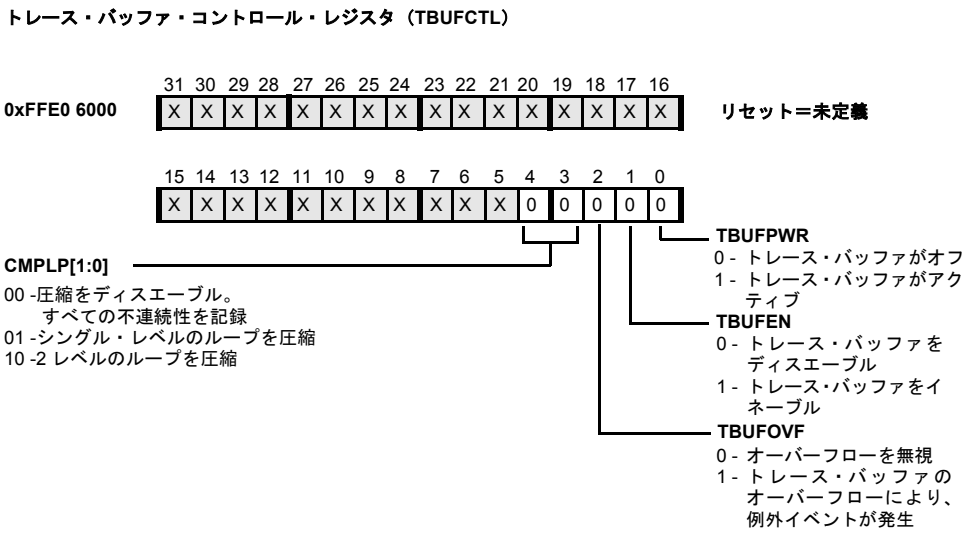


図 19-9. トレース・バッファ・コントロール・レジスタ

■ TBUFSTAT レジスタ

トレース・バッファ・ステータス・レジスタ（TBUFSTAT）を図 19-10に示します。TBUF からの2回の読出しが実行されるたびに、TBUFCNT が1だけデクリメントします。

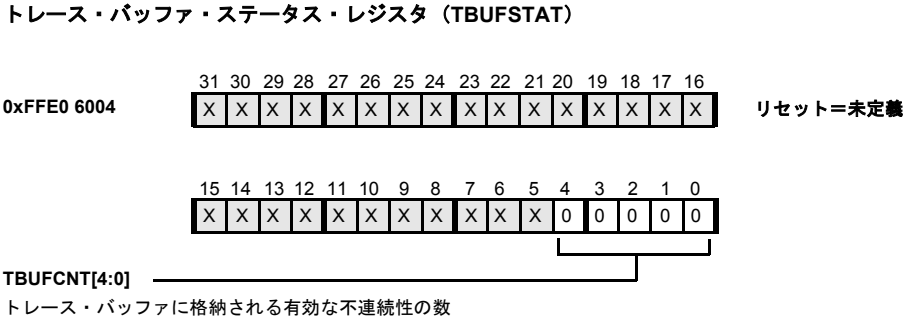


図 19-10. トレース・バッファ・ステータス・レジスタ

■ TBUF レジスタ

トレース・バッファ・レジスタ（TBUF）を図 19-11に示します。最初の読出しで、最新の分岐先アドレスが返されます。2回目の読出しで、最新の分岐元アドレスが返されます。

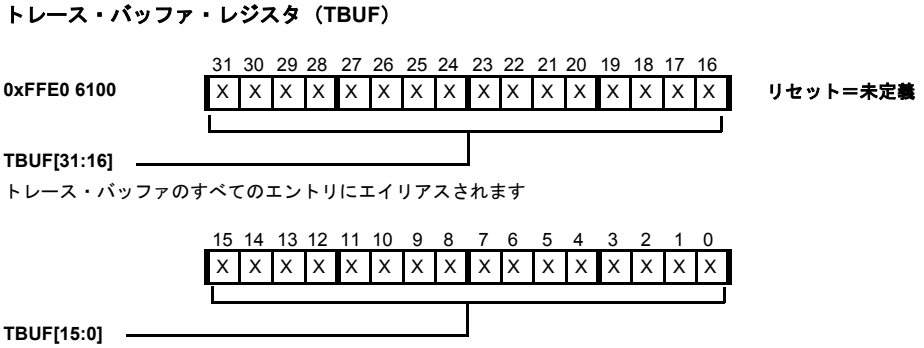


図 19-11. トレース・バッファ・レジスタ

トレース・ユニットは、以下の場合にプログラム・フローの変更を記録しません。

- エミュレータ・モード
- 例外または優先順位の高いサービス・ルーチン（TBUFOVF＝1の場合）

  例外サービス・ルーチンの場合には、プログラム・フローの不連続性をTBUFから読出し、リスト19-1に示すコードによってこれをメモリに保存できます。

TBUF の読出し中は、新しい不連続性が記録されることのないように、トレース・バッファを必ずディスエーブルにしてください。

▶ メモリで実行トレースを再作成するためのコード

リスト19-1には、メモリで実行トレース全体を再作成するためのコードを記載しています。

リスト 19-1. メモリにおける実行トレースの再作成

```
[--sp] = (r7:7, p5:2); /* このルーチンで使用されたレジスタを保存します */
p5 = 32;  /* TBUFを空き状態にするためには、32回の読み出しが必要です */
p2.l = buf;  /* ポインタはソフトウェア・トレース・バッファのヘッダ（最初の場所）をポイントします */
p2.h = buf;  /* ヘッダには、その後のトレース・ダンプに利用できる最初の空き状態のバッファ位置がストアされます */


p4 = [p2++];  /* 利用可能な最初の空き状態のバッファ位置をバッファ・ヘッダから取得します */
p3.l = TBUF & 0xffff;  /* TBUFの下位16ビット */
p3.h = TBUF >> 16;  /* TBUFの上位16ビット */

lsetup(loop1_start, loop1_end) lc0 = p5;
loop1_start: r7 = [p3];  /* TBUFからの読み出し */
loop1_end: [p4++] = r7;  /* メモリへの書き込みとインクリメント */
[p2] = p4;  /* 次に利用可能なバッファ位置をポイントするポインタがbufのヘッダに保存されます */
(r7:7, p5:3) = [sp++];  /* 保存されたレジスタを復元します */
```

パフォーマンス・モニタリング・ユニット

2個の32ビット・カウンタであるパフォーマンス・モニタ・カウンタ・レジスタ（PFCNTR[1:0]）とパフォーマンス・コントロール・レジスタ（PFCTL）は、パフォーマンス・モニタリング期間中におけるプロセッサ・コア・ユニット内部からのイベントの発生回数をカウントします。これらのレジスタは、チップ上の各種リソース間の負荷分散の程度を示すフィードバック情報を提供するので、予測および実際の利用率の比較と解析を行うことができます。さらに、予測ミスやホールド・サイクルなどのイベントをモニタすることも可能です。

■ PFCNTRn レジスタ

パフォーマンス・モニタ・カウンタ・レジスタ PFCNTR[1:0] を図 19-12 に示します。PFCNTR0 レジスタには、パフォーマンス・カウンタ0のカウント値が格納されます。PFCNTR1 レジスタには、パフォーマンス・カウンタ1のカウント値が格納されます。

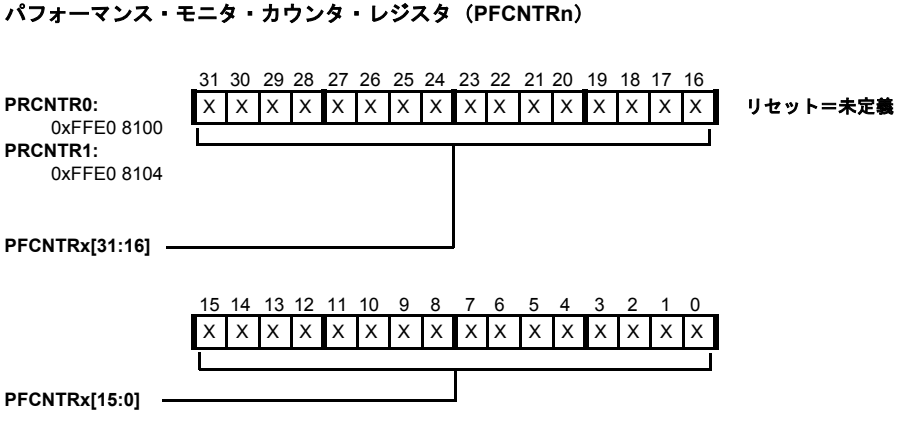


図 19-12.パフォーマンス・モニタ・カウンタ・レジスタ

■ PFCTL レジスタ

パフォーマンス・モニタリング・ユニットをイネーブルにするには、図19-13に示すようにパフォーマンス・モニタ・コントロール・レジスタ（PFCTL）のPFPWRビットを設定します。このユニットをイネーブルにした時点で、個々のカウント・イネーブル・ビット（PFCENn）が有効になります。ユーザ・モード、スーパーバイザ・モード、またはこの両方のモードでPFCENxビットを使用して、パフォーマンス・モニタをイネーブルまたはディスエーブルにしてください。トリガするイベントのタイプを選択するときには、PEMUSWxビットを使用してください。

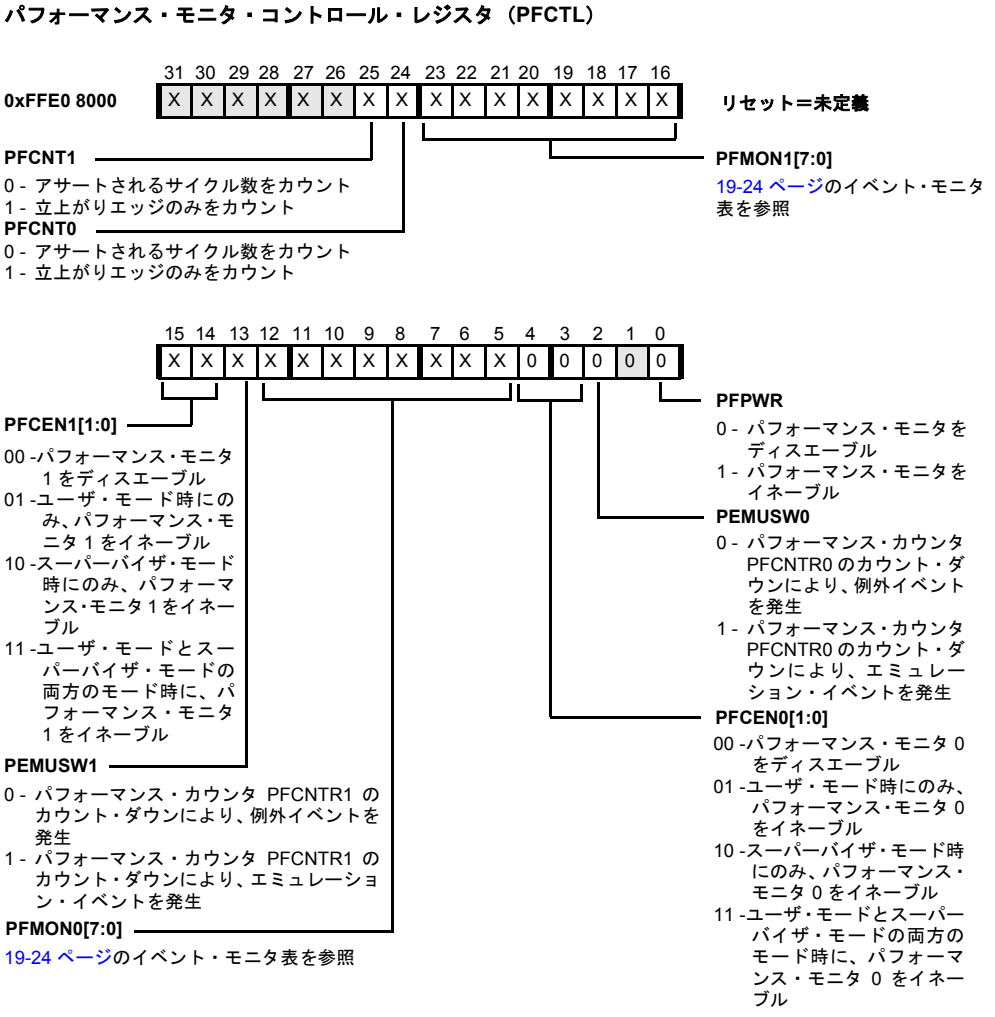


図 19-13.パフォーマンス・モニタ・コントロール・レジスタ

■ イベント・モニタ表

表 19-8には、パフォーマンス・モニタ・カウンタ・レジスタ（PFMON0 またはPFMON1）をインクリメントするイベントをまとめて記載しています。

表 19-8. イベント・モニタ表

PFMONx フィールド	カウント値をインクリメントするイベント
0x00	ループ 0 の繰返し
0x01	ループ 1 の繰返し
0x02	ループ・バッファ 0 が最適化されません。
0x03	ループ・バッファ 1 が最適化されません。
0x04	PC 不変の分岐（トレース・バッファをイネーブルにすることが要求されます。19-17 ページの「TBUFCTL レジスタ」を参照）
0x06	条件付き分岐
0x09	コール、リターン、分岐を含み、割込みを含まないトータル分岐（トレース・バッファをイネーブルにすることが要求されます。19-17 ページの「TBUFCTL レジスタ」を参照）
0x0A	CSYNC、SSYNC によるストール
0x0B	EXCPT 命令
0x0C	CSYNC、SSYNC 命令
0x0D	コミットされた命令
0x0E	割込みの取り込み
0x0F	ミスアライメントのアドレス違反例外
0x10	DAG レジスタ上での書込み後の読出し動作の危険性によるストール・サイクル
0x13	コンピュータ上での RAW データ・ハザードによるストール・サイクル
0x80	DMA の衝突によるコード・メモリ・フェッチの延期（各イベントあたり最低 2 カウント）

表 19-8. イベント・モニタ表（続き）

PFMONx フィールド	カウント値をインクリメントするイベント
0x81	コード・メモリ TAG のストール（キャッシュ・ミス、または FlushI オペレーション、各 FlushI あたり 3 カウント）。コード・メモリのストールによって、プロセッサのストールが発生するのは、命令アセンブリ・ユニットの FIFO が空き状態のときのみに限られる点に注意してください。
0x82	コード・メモリ・フィルのストール（キャッシュ可能またはキャッシュ不能）。コード・メモリのストールによって、プロセッサのストールが発生するのは、命令アセンブリ・ユニットの FIFO が空き状態のときのみに限られる点に注意してください。
0x83	コード・メモリの 64 ビット・ワードがプロセッサの命令アセンブリ・ユニットに転送された
0x90	メモリに対するプロセッサのストール
0x91	プロセッサのストールによって隠されない、プロセッサに対するデータ・メモリのストール
0x92	データ・メモリのストア・バッファの満杯ストール
0x93	優先順位の高いコードから低いコードへの遷移による、データ・メモリの書込みバッファの満杯ストール
0x94	プロセッサからコミットされたデータの欠落によるデータ・メモリのストア・バッファのフォワード・ストール
0x95	データ・メモリのフィル・バッファのストール
0x96	データ・メモリ・アレイまたは TAG の衝突ストール（DAG 対 DAG 間、または DMA 対 DAG 間）
0x97	データ・メモリ・アレイの衝突ストール（DAG 対 DAG 間、または DMA 対 DAG 間）
0x98	データ・メモリのストール
0x99	プロセッサに送信されるデータ・メモリのストール
0x9A	バンク A に対するデータ・メモリ・キャッシュ・フィルの実行完了
0x9B	バンク B に対するデータ・メモリ・キャッシュ・フィルの実行完了
0x9C	バンク A から転送されるデータ・メモリ・キャッシュ・ビクティム
0x9D	バンク B から転送されるデータ・メモリ・キャッシュ・ビクティム

表 19-8. イベント・モニタ表（続き）

PFMONx フィールド	カウント値をインクリメントするイベント
0x9E	データ・メモリ・キャッシュの高優先度フィル要求
0x9F	データ・メモリ・キャッシュの低優先度フィル要求

サイクル・カウンタ

サイクル・カウンタは、プログラム実行中のCCLKサイクルをカウントします。プロセッサがユーザまたはスーパーバイザのモードに入っている期間中に、実行、待ち状態、割込み、イベントを含むすべてのサイクルがカウントされますが、エミュレータ・モード時にはサイクル・カウンタはカウント動作を停止します。

サイクル・カウンタは64ビットであり、1サイクルごとにインクリメントします。カウント値は、CYCLESとCYCLES2の2個の32ビット・レジスタにストアされます。最下位32ビット（LSB）がCYCLESにストアされます。最上位32ビット（MSB）がCYCLES2にストアされます。

読出しデータの一貫性を保証するために、最初のCYCLESの値の読出し時に、その時点のCYCLES2レジスタの値がシャドーレジスタに書き込まれます。次のCYCLES2からの読出しは、シャドーレジスタから行われます。

ユーザ・モード時には、この2個のレジスタからの読出しが可能ですが、書込みは実行できません。スーパーバイザとエミュレータの各モード時には、これらのレジスタは読出し／書込みレジスタとして使用できます。

サイクル・カウンタをイネーブルにするには、SYSCFGレジスタのCCENビットを設定します。コードの数をベンチマークするために、サイクル・カウンタを使用する方法を以下の例に示します。

```
R2 = 0;
CYCLES = R2;
```

```
CYCLES2 = R2;
R2 = SYSCFG;
ビットSET(R2,1);
SYSCFG = R2;
/* ベンチマークとするコードをこの行位置に挿入します */
R2 = SYSCFG;
ビットCLR(R2,1);
SYSCFG = R2;
```

■ CYCLES および CYCLES2 レジスタ

実行サイクル・カウント・レジスタ（CYCLESおよびCYCLES2）を図 19-14に示します。この64ビット・カウンタは、各CCLKサイクルごとにインクリメントします。CYCLESレジスタには、サイクル・カウンタの64ビット・カウント値のうち最下位32ビットが格納されます。最上位32ビットはCYCLES2に格納されます。

CYCLESおよびCYCLES2レジスタはシステムMMRではなく、システム・レジスタです。システム・レジスタの詳細なリストについては、2-8ページを参照してください。

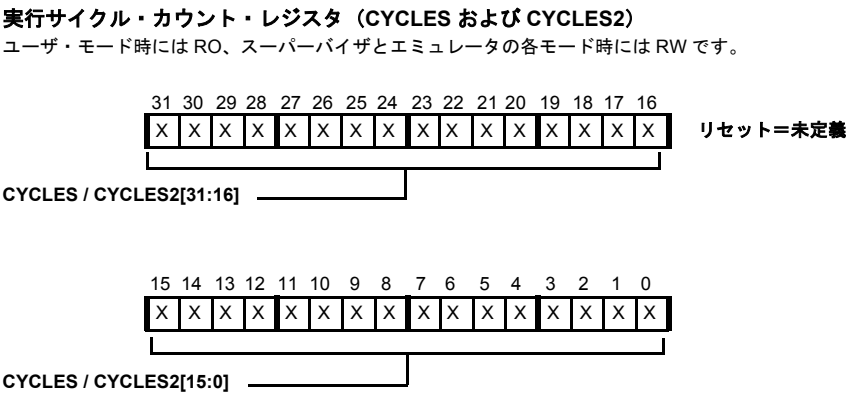


図 19-14.実行サイクル・カウント・レジスタ

製品識別レジスタ

32ビットのDSPデバイスIDレジスタ（DSPID）は、コアの識別および改訂フィールドを含むコアMMRです。

■ DSPID レジスタ

図 19-15に示すDSPデバイスIDレジスタ（DSPID）は読出し専用レジスタであり、コアの一部として使用されます。

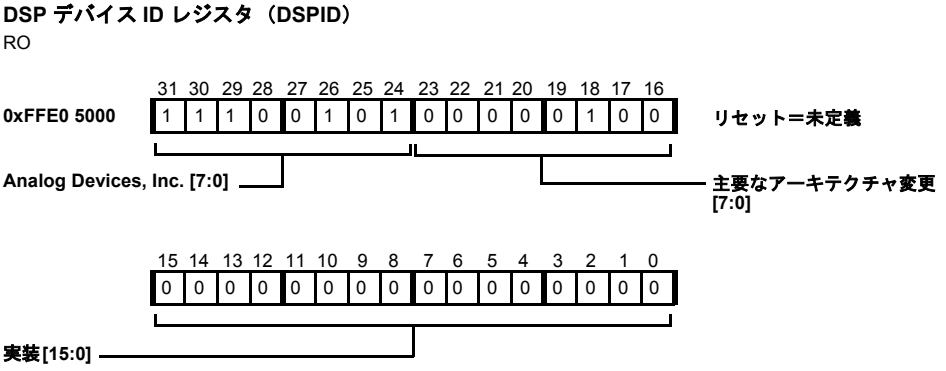


図 19-15.DSP デバイス ID レジスタ

付録A　Blackfinプロセッサのコアメモリマップド・レジスタの割当て

Blackfinプロセッサのメモリマップド・レジスタ（MMR）は、アドレス範囲0xFFE0 0000～0xFFFF FFFFにあります。

すべてのコアMMRは、32ビットの読出し／書込みアクセスによってアクセスされる必要があります。

この付録では、コアMMRアドレスとレジスタ名を一覧にします。MMRの詳細については、「参照先」列に示すページを参照してください。このマニュアルのPDF版では、「参照先」列にあるリファレンスをクリックして、MMRの補足情報にジャンプしてください。

L1 データ・メモリ・コントローラ・レジスタ

L1データ・メモリ・コントローラ・レジスタ（0xFFE0 0000～0xFFE0 0404）

表 A-1.　L1 データ・メモリ・コントローラ・レジスタ

メモリマップド・アドレス	レジスタ名	参照先
0xFFE0 0004	DMEM_CONTROL	6-28 ページの「DMEM_CONTROL レジスタ」
0xFFE0 0008	DCPLB_STATUS	6-67 ページの「DCPLB_STATUS レジスタとICPLB_STATUS レジスタ」
0xFFE0 000C	DCPLB_FAULT_ADDR	6-69 ページの「DCPLB_FAULT_ADDR レジスタとICPLB_FAULT_ADDR レジスタ」
0xFFE0 0100	DCPLB_ADDR0	6-65 ページの「DCPLB_ADDRx レジスタ」

表 A-1.　L1 データ・メモリ・コントローラ・レジスタ（続き）

メモリマップド・アドレス	レジスタ名	参照先
0xFFE0 0104	DCPLB_ADDR1	6-65 ページの「DCPLB_ADDRx レジスタ」
0xFFE0 0108	DCPLB_ADDR2	6-65 ページの「DCPLB_ADDRx レジスタ」
0xFFE0 010C	DCPLB_ADDR3	6-65 ページの「DCPLB_ADDRx レジスタ」
0xFFE0 0110	DCPLB_ADDR4	6-65 ページの「DCPLB_ADDRx レジスタ」
0xFFE0 0114	DCPLB_ADDR5	6-65 ページの「DCPLB_ADDRx レジスタ」
0xFFE0 0118	DCPLB_ADDR6	6-65 ページの「DCPLB_ADDRx レジスタ」
0xFFE0 011C	DCPLB_ADDR7	6-65 ページの「DCPLB_ADDRx レジスタ」
0xFFE0 0120	DCPLB_ADDR8	6-65 ページの「DCPLB_ADDRx レジスタ」
0xFFE0 0124	DCPLB_ADDR9	6-65 ページの「DCPLB_ADDRx レジスタ」
0xFFE0 0128	DCPLB_ADDR10	6-65 ページの「DCPLB_ADDRx レジスタ」
0xFFE0 012C	DCPLB_ADDR11	6-65 ページの「DCPLB_ADDRx レジスタ」
0xFFE0 0130	DCPLB_ADDR12	6-65 ページの「DCPLB_ADDRx レジスタ」
0xFFE0 0134	DCPLB_ADDR13	6-65 ページの「DCPLB_ADDRx レジスタ」
0xFFE0 0138	DCPLB_ADDR14	6-65 ページの「DCPLB_ADDRx レジスタ」
0xFFE0 013C	DCPLB_ADDR15	6-65 ページの「DCPLB_ADDRx レジスタ」
0xFFE0 0200	DCPLB_DATA0	6-63 ページの「DCPLB_DATAx レジスタ」
0xFFE0 0204	DCPLB_DATA1	6-63 ページの「DCPLB_DATAx レジスタ」
0 xFFE0 0208	DCPLB_DATA2	6-63 ページの「DCPLB_DATAx レジスタ」
0xFFE0 020C	DCPLB_DATA3	6-63 ページの「DCPLB_DATAx レジスタ」
0xFFE0 0210	DCPLB_DATA4	6-63 ページの「DCPLB_DATAx レジスタ」
0xFFE0 0214	DCPLB_DATA5	6-63 ページの「DCPLB_DATAx レジスタ」
0xFFE0 0218	DCPLB_DATA6	6-63 ページの「DCPLB_DATAx レジスタ」
0xFFE0 021C	DCPLB_DATA7	6-63 ページの「DCPLB_DATAx レジスタ」
0xFFE0 0220	DCPLB_DATA8	6-63 ページの「DCPLB_DATAx レジスタ」

表 A-1.　L1 データ・メモリ・コントローラ・レジスタ（続き）

メモリマップド・アドレス	レジスタ名	参照先
0xFFE0 0224	DCPLB_DATA9	6-63 ページの「DCPLB_DATAx レジスタ」
0xFFE0 0228	DCPLB_DATA10	6-63 ページの「DCPLB_DATAx レジスタ」
0xFFE0 022C	DCPLB_DATA11	6-63 ページの「DCPLB_DATAx レジスタ」
0xFFE0 0230	DCPLB_DATA12	6-63 ページの「DCPLB_DATAx レジスタ」
0xFFE0 0234	DCPLB_DATA13	6-63 ページの「DCPLB_DATAx レジスタ」
0xFFE0 0238	DCPLB_DATA14	6-63 ページの「DCPLB_DATAx レジスタ」
0xFFE0 023C	DCPLB_DATA15	6-63 ページの「DCPLB_DATAx レジスタ」
0xFFE0 0300	DTEST_COMMAND	6-47 ページの「DTEST_COMMAND レジスタ」
0xFFE0 0400	DTEST_DATA0	6-50 ページの「DTEST_DATA0 レジスタ」
0xFFE0 0404	DTEST_DATA1	6-49 ページの「DTEST_DATA1 レジスタ」

L1 命令メモリ・コントローラ・レジスタ

L1 命令メモリ・コントローラ・レジスタ（0xFFE0 1004～0xFFE0 1404）

表 A-2.　L1 命令メモリ・コントローラ・レジスタ

メモリマップド・アドレス	レジスタ名	参照先
0xFFE0 1004	IMEM_CONTROL	6-8 ページの「IMEM_CONTROL レジスタ」
0xFFE0 1008	ICPLB_STATUS	6-67 ページの「DCPLB_STATUS レジスタと ICPLB_STATUS レジスタ」
0xFFE0 100C	ICPLB_FAULT_ADDR	6-69 ページの「DCPLB_FAULT_ADDR レジスタと ICPLB_FAULT_ADDR レジスタ」
0xFFE0 1100	ICPLB_ADDR0	6-66 ページの「ICPLB_ADDRx レジスタ」
0xFFE0 1104	ICPLB_ADDR1	6-66 ページの「ICPLB_ADDRx レジスタ」

表 A-2.　L1 命令メモリ・コントローラ・レジスタ（続き）

メモリマップド・アドレス	レジスタ名	参照先
0xFFE0 1108	ICPLB_ADDR2	6-66 ページの「ICPLB_ADDRx レジスタ」
0xFFE0 110C	ICPLB_ADDR3	6-66 ページの「ICPLB_ADDRx レジスタ」
0xFFE0 1110	ICPLB_ADDR4	6-66 ページの「ICPLB_ADDRx レジスタ」
0xFFE0 1114	ICPLB_ADDR5	6-66 ページの「ICPLB_ADDRx レジスタ」
0xFFE0 1118	ICPLB_ADDR6	6-66 ページの「ICPLB_ADDRx レジスタ」
0xFFE0 111C	ICPLB_ADDR7	6-66 ページの「ICPLB_ADDRx レジスタ」
0xFFE0 1120	ICPLB_ADDR8	6-66 ページの「ICPLB_ADDRx レジスタ」
0xFFE0 1124	ICPLB_ADDR9	6-66 ページの「ICPLB_ADDRx レジスタ」
0xFFE0 1128	ICPLB_ADDR10	6-66 ページの「ICPLB_ADDRx レジスタ」
0xFFE0 112C	ICPLB_ADDR11	6-66 ページの「ICPLB_ADDRx レジスタ」
0xFFE0 1130	ICPLB_ADDR12	6-66 ページの「ICPLB_ADDRx レジスタ」
0xFFE0 1134	ICPLB_ADDR13	6-66 ページの「ICPLB_ADDRx レジスタ」
0xFFE0 1138	ICPLB_ADDR14	6-66 ページの「ICPLB_ADDRx レジスタ」
0xFFE0 113C	ICPLB_ADDR15	6-66 ページの「ICPLB_ADDRx レジスタ」
0xFFE0 1200	ICPLB_DATA0	6-61 ページの「ICPLB_DATAx レジスタ」
0xFFE0 1204	ICPLB_DATA1	6-61 ページの「ICPLB_DATAx レジスタ」
0xFFE0 1208	ICPLB_DATA2	6-61 ページの「ICPLB_DATAx レジスタ」
0xFFE0 120C	ICPLB_DATA3	6-61 ページの「ICPLB_DATAx レジスタ」
0xFFE0 1210	ICPLB_DATA4	6-61 ページの「ICPLB_DATAx レジスタ」
0xFFE0 1214	ICPLB_DATA5	6-61 ページの「ICPLB_DATAx レジスタ」
0xFFE0 1218	ICPLB_DATA6	6-61 ページの「ICPLB_DATAx レジスタ」
0xFFE0 121C	ICPLB_DATA7	6-61 ページの「ICPLB_DATAx レジスタ」
0xFFE0 1220	ICPLB_DATA8	6-61 ページの「ICPLB_DATAx レジスタ」
0xFFE0 1224	ICPLB_DATA9	6-61 ページの「ICPLB_DATAx レジスタ」

表 A-2.　L1 命令メモリ・コントローラ・レジスタ（続き）

メモリマップド・アドレス	レジスタ名	参照先
0xFFE0 1228	ICPLB_DATA10	6-61 ページの「ICPLB_DATAx レジスタ」
0xFFE0 122C	ICPLB_DATA11	6-61 ページの「ICPLB_DATAx レジスタ」
0xFFE0 1230	ICPLB_DATA12	6-61 ページの「ICPLB_DATAx レジスタ」
0xFFE0 1234	ICPLB_DATA13	6-61 ページの「ICPLB_DATAx レジスタ」
0xFFE0 1238	ICPLB_DATA14	6-61 ページの「ICPLB_DATAx レジスタ」
0xFFE0 123C	ICPLB_DATA15	6-61 ページの「ICPLB_DATAx レジスタ」
0xFFE0 1300	ITEST_COMMAND	6-25 ページの「ITEST_COMMAND レジスタ」
0XFFE0 1400	ITEST_DATA0	6-27 ページの「ITEST_DATA0 レジスタ」
0XFFE0 1404	ITEST_DATA1	6-26 ページの「ITEST_DATA1 レジスタ」

割込みコントローラ・レジスタ

割込みコントローラ・レジスタ（0xFFE0 2000～0xFFE0 2110）

表 A-3.　割込みコントローラ・レジスタ

メモリマップド・アドレス	レジスタ名	参照先
0xFFE0 2000	EVT0 (EMU)	4-39 ページの「コア・イベント・ベクトル・テーブル」
0xFFE0 2004	EVT1 (RST)	4-39 ページの「コア・イベント・ベクトル・テーブル」
0xFFE0 2008	EVT2 (NMI)	4-39 ページの「コア・イベント・ベクトル・テーブル」
0xFFE0 200C	EVT3 (EVX)	4-39 ページの「コア・イベント・ベクトル・テーブル」
0xFFE0 2010	EVT4	4-39 ページの「コア・イベント・ベクトル・テーブル」

表 A-3.　割込みコントローラ・レジスタ（続き）

メモリマップド・アドレス	レジスタ名	参照先
0xFFE0 2014	EVT5 (IVHW)	4-39 ページの「コア・イベント・ベクトル・テーブル」
0xFFE0 2018	EVT6 (TMR)	4-39 ページの「コア・イベント・ベクトル・テーブル」
0xFFE0 201C	EVT7 (IVG7)	4-39 ページの「コア・イベント・ベクトル・テーブル」
0xFFE0 2020	EVT8 (IVG8)	4-39 ページの「コア・イベント・ベクトル・テーブル」
0xFFE0 2024	EVT9 (IVG9)	4-39 ページの「コア・イベント・ベクトル・テーブル」
0xFFE0 2028	EVT10 (IVG10)	4-39 ページの「コア・イベント・ベクトル・テーブル」
0xFFE0 202C	EVT11 (IVG11)	4-39 ページの「コア・イベント・ベクトル・テーブル」
0xFFE0 2030	EVT12 (IVG12)	4-39 ページの「コア・イベント・ベクトル・テーブル」
0xFFE0 2034	EVT13 (IVG13)	4-39 ページの「コア・イベント・ベクトル・テーブル」
0xFFE0 2038	EVT14 (IVG14)	4-39 ページの「コア・イベント・ベクトル・テーブル」
0xFFE0 203C	EVT15 (IVG15)	4-39 ページの「コア・イベント・ベクトル・テーブル」
0xFFE0 2100	EVT_OVERRIDE	
0xFFE0 2104	IMASK	4-35 ページの「IMASK レジスタ」
0xFFE0 2108	IPEND	4-37 ページの「IPEND レジスタ」
0xFFE0 2110	IPRIO	6-43 ページの「IPRIO レジスタと書込みバッファの深さ」

表 A-3.　割込みコントローラ・レジスタ（続き）

メモリマップド・アドレス	レジスタ名	参照先
0xFFE0 210C	ILAT	4-36 ページの「ILAT レジスタ」
0XFFE0 2110	IPRIO	6-43 ページの「IPRIO レジスタと書込みバッファの深さ」

コア・タイマ・レジスタ

コア・タイマ・レジスタ（0xFFE0 3000～0xFFE0 300C）

表 A-4.　コア・タイマ・レジスタ

メモリマップド・アドレス	レジスタ名	参照先
0xFFE0 3000	TCNTL	15-50 ページの「TCNTL レジスタ」
0xFFE0 3004	TPERIOD	15-53 ページの「TPERIOD レジスタ」
0xFFE0 3008	TSCALE	15-54 ページの「TSCALE レジスタ」
0xFFE0 300C	TCOUNT	15-52 ページの「TCOUNT レジスタ」

デバッグ、MP、エミュレーション・ユニット・レジスタ

デバッグ、MP、エミュレーション・ユニット・レジスタ（0xFFE0 5000～0xFFE0 5008）

表 A-5.　デバッグ・レジスタとエミュレーション・ユニット・レジスタ

メモリマップド・アドレス	レジスタ名	参照先
0xFFE0 5000	DSPID	19-29 ページの「DSPID レジスタ」

トレース・ユニット・レジスタ

トレース・ユニット・レジスタ（0xFFE0 6000～0xFFE0 6100）

表 A-6.　トレース・ユニット・レジスタ

メモリマップド・アドレス	レジスタ名	参照先
0xFFE0 6000	TBUFCTL	19-17 ページの「TBUFCTL レジスタ」
0xFFE0 6004	TBUFSTAT	19-18 ページの「TBUFSTAT レジスタ」
0xFFE0 6100	TBUF	19-19 ページの「TBUF レジスタ」

ウォッチポイント・レジスタとパッチ・レジスタ

ウォッチポイント・レジスタとパッチ・レジスタ（0xFFE0 7000～0xFFE0 7200）

表 A-7.　ウォッチポイント・レジスタとパッチ・レジスタ

メモリマップド・アドレス	レジスタ名	参照先
0xFFE0 7000	WPIACTL	19-8 ページの「WPIACTL レジスタ」
0xFFE0 7040	WPIA0	19-6 ページの「WPIAn レジスタ」
0xFFE0 7044	WPIA1	19-6 ページの「WPIAn レジスタ」
0xFFE0 7048	WPIA2	19-6 ページの「WPIAn レジスタ」
0xFFE0 704C	WPIA3	19-6 ページの「WPIAn レジスタ」
0xFFE0 7050	WPIA4	19-6 ページの「WPIAn レジスタ」
0xFFE0 7054	WPIA5	19-6 ページの「WPIAn レジスタ」
0xFFE0 7080	WPIACNT0	19-7 ページの「WPIACNTn レジスタ」
0xFFE0 7084	WPIACNT1	19-7 ページの「WPIACNTn レジスタ」
0xFFE0 7088	WPIACNT2	19-7 ページの「WPIACNTn レジスタ」
0xFFE0 708C	WPIACNT3	19-7 ページの「WPIACNTn レジスタ」
0xFFE0 7090	WPIACNT4	19-7 ページの「WPIACNTn レジスタ」
0xFFE0 7094	WPIACNT5	19-7 ページの「WPIACNTn レジスタ」
0xFFE0 7100	WPDACTL	19-13 ページの「WPDACTL レジスタ」
0xFFE0 7140	WPDA0	19-12 ページの「WPDAn レジスタ」
0xFFE0 7144	WPDA1	19-12 ページの「WPDAn レジスタ」
0xFFE0 7180	WPDACNT0	19-12 ページの「WPDACNTn レジスタ」
0xFFE0 7184	WPDACNT1	19-12 ページの「WPDACNTn レジスタ」
0xFFE0 7200	WPSTAT	19-15 ページの「WPSTAT レジスタ」

パフォーマンス・モニタ・レジスタ

パフォーマンス・モニタ・レジスタ（0xFFE0 8000～0xFFE0 8104）

表 A-8.　パフォーマンス・モニタ・レジスタ

メモリマップド・アドレス	レジスタ名	参照先
0xFFE0 8000	PFCTL	19-22 ページの「PFCTL レジスタ」
0xFFE0 8100	PFCNTR0	19-21 ページの「PFCNTRn レジスタ」
0xFFE0 8104	PFCNTR1	19-21 ページの「PFCNTRn レジスタ」

付録B　システムMMRの割当て

ここでは、システム・メモリマップド・レジスタ（MMR）の一般的な情報を提供します。

- システムMMRのアドレス範囲は0xFFC0 0000～0xFFDF FFFFです。
- すべてのシステム MMR は、16 ビット幅または 32 ビット幅です。16ビット幅のMMRは、16ビットの読出し／書込み動作でアクセスされる必要があります。32ビット幅のMMRは、32ビットの読出し／書込み動作でアクセスされる必要があります。16ビット・アクセスが必要か32ビット・アクセスが必要かを判断するには、MMRの説明をチェックしてください。
- この付録で定義されていないすべてのシステムMMR空間は、内部使用専用に予約されています。

この付録では、MMRアドレスとレジスタ名を一覧にします。MMRの詳細については、「参照先」列に示すページを参照してください。このマニュアルのPDF版では、「参照先」列にあるリファレンスをクリックして、MMRの補足情報にジャンプしてください。

ダイナミック・パワー・マネジメント・レジスタ

ダイナミック・パワー・マネジメント・レジスタ（0xFFC0 0000～0xFFC0 00FF）

表 B-1.　ダイナミック・パワー・マネジメント・レジスタ

メモリマップド・アドレス	レジスタ名	参照先
0xFFC0 0000	PLL_CTL	8-8 ページの「PLL_CTL レジスタ」
0xFFC0 0004	PLL_DIV	8-7 ページの「PLL_DIV レジスタ」
0xFFC0 0008	VR_CTL	8-27 ページの「VR_CTL レジスタ」
0xFFC0 000C	PLL_STAT	8-10 ページの「PLL_STAT レジスタ」
0xFFC0 0010	PLL_LOCKCNT	8-11 ページの「PLL_LOCKCNT レジスタ」

システム・リセットと割込みコントロール・レジスタ

システム・リセットと割込みコントローラ・レジスタ（0xFFC0 0100～0xFFC0 01FF）

表 B-2.　システム割込みコントローラ・レジスタ

メモリマップド・アドレス	レジスタ名	参照先
0xFFC0 0100	SWRST	3-17 ページの「SWRST レジスタ」
0xFFC0 0104	SYSCR	3-15 ページの「SYSCR レジスタ」
0xFFC0 010C	SIC_IMASK	4-30 ページの「SIC_IMASK レジスタ」
0xFFC0 0110	SIC_IAR0	4-32 ページの「システム割込み割当てレジスタ（SIC_IARx）」
0xFFC0 0114	SIC_IAR1	4-32 ページの「システム割込み割当てレジスタ（SIC_IARx）」
0xFFC0 0118	SIC_IAR2	4-32 ページの「システム割込み割当てレジスタ（SIC_IARx）」
0xFFC0 0120	SIC_ISR	4-29 ページの「SIC_ISR レジスタ」
0xFFC0 0124	SIC_IWR	4-27 ページの「SIC_IWR レジスタ」

ウォッチドッグ・タイマ・レジスタ

ウォッチドッグ・タイマ・レジスタ（0xFFC0 0200～0xFFC0 02FF）

表 B-3.　ウォッチドッグ・タイマ・レジスタ

メモリマップド・アドレス	レジスタ名	参照先
0xFFC0 0200	WDOG_CTL	15-58 ページの「WDOG_CTL レジスタ」
0xFFC0 0204	WDOG_CNT	15-55 ページの「WDOG_CNT レジスタ」
0xFFC0 0208	WDOG_STAT	15-56 ページの「WDOG_STAT レジスタ」

リアルタイム・クロック・レジスタ

リアルタイム・クロック・レジスタ（0xFFC0 0300～0xFFC0 03FF）

表 B-4.　リアルタイム・クロック・レジスタ

メモリマップド・アドレス	レジスタ名	参照先
0xFFC0 0300	RTC_STAT	16-14 ページの「RTC_STAT レジスタ」
0xFFC0 0304	RTC_ICTL	16-16 ページの「RTC_ICTL レジスタ」
0xFFC0 0308	RTC_ISTAT	16-17 ページの「RTC_ISTAT レジスタ」
0xFFC0 030C	RTC_SWCNT	16-18 ページの「RTC_SWCNT レジスタ」
0xFFC0 0310	RTC_ALARM	16-20 ページの「RTC_ALARM レジスタ」
0xFFC0 0314	RTC_PREN	16-21 ページの「RTC_PREN レジスタ」

パラレル・ペリフェラル・インターフェース（PPI）レジスタ

パラレル・ペリフェラル・インターフェース（PPI）レジスタ（0xFFC0 1000～0xFFC0 10FF）

表 B-5.　パラレル・ペリフェラル・インターフェース（PPI）レジスタ

メモリマップド・アドレス	レジスタ名	参照先
0xFFC0 1000	PPI_CONTROL	11-3 ページの「PPI_CONTROL レジスタ」
0xFFC0 1004	PPI_STATUS	11-9 ページの「PPI_STATUS レジスタ」
0xFFC0 1008	PPI_COUNT	11-11 ページの「PPI_COUNT レジスタ」
0xFFC0 100C	PPI_DELAY	11-11 ページの「PPI_DELAY レジスタ」
0xFFC0 1010	PPI_FRAME	11-12 ページの「PPI_FRAME レジスタ」

UART コントローラ・レジスタ

UART コントローラ・レジスタ（0xFFC0 0400～0xFFC0 04FF）

表 B-6.　UART コントローラ・レジスタ

メモリマップド・アドレス	レジスタ名	参照先
0xFFC0 0400	UART_THR	13-7 ページの「UART_THR レジスタ」
0xFFC0 0400	UART_RBR	13-8 ページの「UART_RBR レジスタ」
0xFFC0 0400	UART_DLL	13-13 ページの「UART_DLL レジスタと UART_DLH レジスタ」
0xFFC0 0404	UART_DLH	13-13 ページの「UART_DLL レジスタと UART_DLH レジスタ」
0xFFC0 0404	UART_IER	13-9 ページの「UART_IER レジスタ」

表 B-6.　UART コントローラ・レジスタ（続き）

メモリマップド・アドレス	レジスタ名	参照先
0xFFC0 0408	UART_IIR	13-11 ページの「UART_IIR レジスタ」
0xFFC0 040C	UART_LCR	13-4 ページの「UART_LCR レジスタ」
0xFFC0 0410	UART_MCR	13-5 ページの「UART_MCR レジスタ」
0xFFC0 0414	UART_LSR	13-6 ページの「UART_LSR レジスタ」
0xFFC0 041C	UART_SCR	13-15 ページの「UART_SCR レジスタ」
0xFFC0 0424	UART_GCTL	13-15 ページの「UART_GCTL レジスタ」

SPI コントローラ・レジスタ

SPI コントローラ・レジスタ（0xFFC0 0500～0xFFC0 05FF）

表 B-7.　SPI コントローラ・レジスタ

メモリマップド・アドレス	レジスタ名	参照先
0xFFC0 0500	SPI_CTL	10-9 ページの「SPI_CTL レジスタ」
0xFFC0 0504	SPI_FLG	10-12 ページの「SPI_FLG レジスタ」
0xFFC0 0508	SPI_STAT	10-16 ページの「SPI_STAT レジスタ」
0xFFC0 050C	SPI_TDBR	10-18 ページの「SPI_TDBR レジスタ」
0xFFC0 0510	SPI_RDBR	10-19 ページの「SPI_RDBR レジスタ」
0xFFC0 0514	SPI_BAUD	10-8 ページの「SPI_BAUD レジスタ」
0xFFC0 0518	SPI_SHADOW	10-20 ページの「SPI_SHADOW レジスタ」

タイマ・レジスタ

タイマ・レジスタ（0xFFC0 0600～0xFFC0 06FF）

表 B-8.　タイマ・レジスタ

メモリマップド・アドレス	レジスタ名	参照先
0xFFC0 0600	TIMER0_CONFIG	15-9 ページの「TIMERx_CONFIG レジスタ」
0xFFC0 0604	TIMER0_COUNTER	15-11 ページの「TIMERx_COUNTER レジスタ」
0xFFC0 0608	TIMER0_PERIOD	15-12 ページの「TIMERx_PERIOD レジスタと TIMERx_WIDTH レジスタ」
0xFFC0 060C	TIMER0_WIDTH	15-12 ページの「TIMERx_PERIOD レジスタと TIMERx_WIDTH レジスタ」
0xFFC0 0610	TIMER1_CONFIG	15-9 ページの「TIMERx_CONFIG レジスタ」
0xFFC0 0614	TIMER1_COUNTER	15-11 ページの「TIMERx_COUNTER レジスタ」
0xFFC0 0618	TIMER1_PERIOD	15-12 ページの「TIMERx_PERIOD レジスタと TIMERx_WIDTH レジスタ」
0xFFC0 061C	TIMER1_WIDTH	15-12 ページの「TIMERx_PERIOD レジスタと TIMERx_WIDTH レジスタ」
0xFFC0 0620	TIMER2_CONFIG	15-9 ページの「TIMERx_CONFIG レジスタ」
0xFFC0 0624	TIMER2_COUNTER	15-11 ページの「TIMERx_COUNTER レジスタ」
0xFFC0 0628	TIMER2_PERIOD	15-12 ページの「TIMERx_PERIOD レジスタと TIMERx_WIDTH レジスタ」
0xFFC0 062C	TIMER2_WIDTH	15-12 ページの「TIMERx_PERIOD レジスタと TIMERx_WIDTH レジスタ」
0xFFC0 0640	TIMER_ENABLE	15-4 ページの「TIMER_ENABLE レジスタ」
0xFFC0 0644	TIMER_DISABLE	15-6 ページの「TIMER_DISABLE レジスタ」
0xFFC0 0648	TIMER_STATUS	15-7 ページの「TIMER_STATUS レジスタ」

プログラマブル・フラグ・レジスタ

プログラマブル・フラグ・レジスタ（0xFFC0 0700～0xFFC0 07FF）

表 B-9.　プログラマブル・フラグ・レジスタ

メモリマップド・アドレス	レジスタ名	参照先
0xFFC0 0700	FIO_FLAG_D	14-9 ページの「FIO_FLAG_D レジスタ」
0xFFC0 0704	FIO_FLAG_C	14-9 ページの「FIO_FLAG_S、FIO_FLAG_C、および FIO_FLAG_T レジスタ」
0xFFC0 0708	FIO_FLAG_S	14-9 ページの「FIO_FLAG_S、FIO_FLAG_C、および FIO_FLAG_T レジスタ」
0xFFC0 070C	FIO_FLAG_T	14-9 ページの「FIO_FLAG_S、FIO_FLAG_C、および FIO_FLAG_T レジスタ」
0xFFC0 0710	FIO_MASKA_D	14-12 ページの「FIO_MASKA_D、FIO_MASKA_C、FIO_MASKA_S、FIO_MASKA_T、FIO_MASKB_D、FIO_MASKB_C、FIO_MASKB_S、FIO_MASKB_T レジスタ」
0xFFC0 0714	FIO_MASKA_C	14-12 ページの「FIO_MASKA_D、FIO_MASKA_C、FIO_MASKA_S、FIO_MASKA_T、FIO_MASKB_D、FIO_MASKB_C、FIO_MASKB_S、FIO_MASKB_T レジスタ」
0xFFC0 0718	FIO_MASKA_S	14-12 ページの「FIO_MASKA_D、FIO_MASKA_C、FIO_MASKA_S、FIO_MASKA_T、FIO_MASKB_D、FIO_MASKB_C、FIO_MASKB_S、FIO_MASKB_T レジスタ」
0xFFC0 071C	FIO_MASKA_T	14-12 ページの「FIO_MASKA_D、FIO_MASKA_C、FIO_MASKA_S、FIO_MASKA_T、FIO_MASKB_D、FIO_MASKB_C、FIO_MASKB_S、FIO_MASKB_T レジスタ」

表 B-9.　プログラマブル・フラグ・レジスタ（続き）

メモリマップド・アドレス	レジスタ名	参照先
0xFFC0 0720	FIO_MASKB_D	14-12 ページの「FIO_MASKA_D、FIO_MASKA_C、FIO_MASKA_S、FIO_MASKA_T、FIO_MASKB_D、FIO_MASKB_C、FIO_MASKB_S、FIO_MASKB_T レジスタ」
0xFFC0 0724	FIO_MASKB_C	14-12 ページの「FIO_MASKA_D、FIO_MASKA_C、FIO_MASKA_S、FIO_MASKA_T、FIO_MASKB_D、FIO_MASKB_C、FIO_MASKB_S、FIO_MASKB_T レジスタ」
0xFFC0 0728	FIO_MASKB_S	14-12 ページの「FIO_MASKA_D、FIO_MASKA_C、FIO_MASKA_S、FIO_MASKA_T、FIO_MASKB_D、FIO_MASKB_C、FIO_MASKB_S、FIO_MASKB_T レジスタ」
0xFFC0 072C	FIO_MASKB_T	14-12 ページの「FIO_MASKA_D、FIO_MASKA_C、FIO_MASKA_S、FIO_MASKA_T、FIO_MASKB_D、FIO_MASKB_C、FIO_MASKB_S、FIO_MASKB_T レジスタ」
0xFFC0 0730	FIO_DIR	14-5 ページの「FIO_DIR レジスタ」
0xFFC0 0734	FIO_POLAR	14-20 ページの「FIO_POLAR レジスタ」
0xFFC0 0738	FIO_EDGE	14-21 ページの「FIO_EDGE レジスタ」
0xFFC0 073C	FIO_BOTH	14-22 ページの「FIO_BOTH レジスタ」
0xFFC0 0740	FIO_INEN	14-23 ページの「FIO_INEN レジスタ」

SPORT0 コントローラ・レジスタ

SPORT0 コントローラ・レジスタ（0xFFC0 0800～0xFFC0 08FF）

表 B-10. SPORT0 コントローラ・レジスタ

メモリマップド・アドレス	レジスタ名	参照先
0xFFC0 0800	SPORT0_TCR1	12-12 ページの「SPORTx_TCR1 レジスタと SPORTx_TCR2 レジスタ」
0xFFC0 0804	SPORT0_TCR2	12-12 ページの「SPORTx_TCR1 レジスタと SPORTx_TCR2 レジスタ」
0xFFC0 0808	SPORT0_TCLKDIV	12-31 ページの「SPORTx_TCLKDIV レジスタと SPORTx_RCLKDIV レジスタ」
0xFFC0 080C	SPORT0_TFSDIV	12-32 ページの「SPORTx_TFSDIV レジスタと SPORTx_RFSDIV レジスタ」
0xFFC0 0810	SPORT0_TX	12-23 ページの「SPORTx_TX レジスタ」
0xFFC0 0818	SPORT0_RX	12-26 ページの「SPORTx_RX レジスタ」
0xFFC0 0820	SPORT0_RCR1	12-18 ページの「SPORTx_RCR1 レジスタと SPORTx_RCR2 レジスタ」
0xFFC0 0824	SPORT0_RCR2	12-18 ページの「SPORTx_RCR1 レジスタと SPORTx_RCR2 レジスタ」
0xFFC0 0828	SPORT0_RCLKDIV	12-31 ページの「SPORTx_TCLKDIV レジスタと SPORTx_RCLKDIV レジスタ」
0xFFC0 082C	SPORT0_RFSDIV	12-32 ページの「SPORTx_TFSDIV レジスタと SPORTx_RFSDIV レジスタ」
0xFFC0 0830	SPORT0_STAT	12-28 ページの「SPORTx_STAT レジスタ」
0xFFC0 0834	SPORT0_CHNL	12-60 ページの「SPORTx_CHNL レジスタ」
0xFFC0 0838	SPORT0_MCMC1	12-53 ページの「SPORTx_MCMCn レジスタ」
0xFFC0 083C	SPORT0_MCMC2	12-53 ページの「SPORTx_MCMCn レジスタ」
0xFFC0 0840	SPORT0_MTCS0	12-65 ページの「SPORTx_MTCSn レジスタ」
0xFFC0 0844	SPORT0_MTCS1	12-65 ページの「SPORTx_MTCSn レジスタ」

表 B-10.　SPORT0 コントローラ・レジスタ（続き）

メモリマップド・アドレス	レジスタ名	参照先
0xFFC0 0848	SPORT0_MTCS2	12-65 ページの「SPORTx_MTCSn レジスタ」
0xFFC0 084C	SPORT0_MTCS3	12-65 ページの「SPORTx_MTCSn レジスタ」
0xFFC0 0850	SPORT0_MRCS0	12-63 ページの「SPORTx_MRCSn レジスタ」
0xFFC0 0854	SPORT0_MRCS1	12-63 ページの「SPORTx_MRCSn レジスタ」
0xFFC0 0858	SPORT0_MRCS2	12-63 ページの「SPORTx_MRCSn レジスタ」
0xFFC0 085C	SPORT0_MRCS3	12-63 ページの「SPORTx_MRCSn レジスタ」

SPORT1 コントローラ・レジスタ

SPORT1 コントローラ・レジスタ（0xFFC0 0900～0xFFC0 09FF）

表 B-11.　SPORT1 コントローラ・レジスタ

メモリマップド・アドレス	レジスタ名	参照先
0xFFC0 0900	SPORT1_TCR1	12-12 ページの「SPORTx_TCR1 レジスタと SPORTx_TCR2 レジスタ」
0xFFC0 0904	SPORT1_TCR2	12-12 ページの「SPORTx_TCR1 レジスタと SPORTx_TCR2 レジスタ」
0xFFC0 0908	SPORT1_TCLKDIV	12-31 ページの「SPORTx_TCLKDIV レジスタと SPORTx_RCLKDIV レジスタ」
0xFFC0 090C	SPORT1_TFSDIV	12-32 ページの「SPORTx_TFSDIV レジスタと SPORTx_RFSDIV レジスタ」
0xFFC0 0910	SPORT1_TX	12-23 ページの「SPORTx_TX レジスタ」
0xFFC0 0918	SPORT1_RX	12-26 ページの「SPORTx_RX レジスタ」
0xFFC0 0920	SPORT1_RCR1	12-18 ページの「SPORTx_RCR1 レジスタと SPORTx_RCR2 レジスタ」

表 B-11.　SPORT1 コントローラ・レジスタ（続き）

メモリマップド・アドレス	レジスタ名	参照先
0xFFC0 0924	SPORT1_RCR2	12-18 ページの「SPORTx_RCR1 レジスタと SPORTx_RCR2 レジスタ」
0xFFC0 0928	SPORT1_RCLKDIV	12-31 ページの「SPORTx_TCLKDIV レジスタと SPORTx_RCLKDIV レジスタ」
0xFFC0 092C	SPORT1_RFSDIV	12-32 ページの「SPORTx_TFSDIV レジスタと SPORTx_RFSDIV レジスタ」
0xFFC0 0930	SPORT1_STAT	12-28 ページの「SPORTx_STAT レジスタ」
0xFFC0 0934	SPORT1_CHNL	12-60 ページの「SPORTx_CHNL レジスタ」
0xFFC0 0938	SPORT1_MCMC1	12-53 ページの「SPORTx_MCMCn レジスタ」
0xFFC0 093C	SPORT1_MCMC2	12-53 ページの「SPORTx_MCMCn レジスタ」
0xFFC0 0940	SPORT1_MTCS0	12-65 ページの「SPORTx_MTCSn レジスタ」
0xFFC0 0944	SPORT1_MTCS1	12-65 ページの「SPORTx_MTCSn レジスタ」
0xFFC0 0948	SPORT1_MTCS2	12-65 ページの「SPORTx_MTCSn レジスタ」
0xFFC0 094C	SPORT1_MTCS3	12-65 ページの「SPORTx_MTCSn レジスタ」
0xFFC0 0950	SPORT1_MRCS0	12-63 ページの「SPORTx_MRCSn レジスタ」
0xFFC0 0954	SPORT1_MRCS1	12-63 ページの「SPORTx_MRCSn レジスタ」
0xFFC0 0958	SPORT1_MRCS2	12-63 ページの「SPORTx_MRCSn レジスタ」
0xFFC0 095C	SPORT1_MRCS3	12-63 ページの「SPORTx_MRCSn レジスタ」

DMA/ メモリ DMA コントロール・レジスタ

DMA コントロール・レジスタ（0xFFC0 0B00～0xFFC0 0FFF）

表 B-12. DMA トラフィック・コントロール・レジスタ

メモリマップド・アドレス	レジスタ名	参照先
0xFFC0 0B0C	DMA_TC_PER	9-57 ページの「DMA_TC_PER レジスタと DMA_TC_CNT レジスタ」
0xFFC0 0B10	DMA_TC_CNT	9-57 ページの「DMA_TC_PER レジスタと DMA_TC_CNT レジスタ」

各DMAチャンネルには、そのDMAチャンネルに関連付けられたベース・アドレスからの固定オフセットを持つ同一のMMRセットがあるため、表 B-13と表 B-14に示すMMR情報を参照すると便利です。表 B-13には、各DMAチャンネルのベース・アドレスとチャンネルを識別するレジスタ接頭辞を示します。表 B-14には、レジスタ接尾辞とベース・アドレスからのそのオフセットを示します。

たとえば、DMAチャンネル0のY_MODIFYレジスタはDMA0_Y_MODIFYと呼ばれ、そのアドレスは0xFFC0 0C1Cです。同様に、メモリDMAストリーム0のソース・カレント・アドレス・レジスタはMDMA_S0_CURR_ADDRと呼ばれ、そのアドレスは0xFFC0 0E64です。

表 B-13. DMA チャンネルのベース・アドレス

DMA チャンネル識別マーク	MMR ベース・アドレス	レジスタ接頭辞
0	0xFFC0 0C00	DMA0_
1	0xFFC0 0C40	DMA1_
2	0xFFC0 0C80	DMA2_
3	0xFFC0 0CC0	DMA3_

表 B-13.　DMA チャンネルのベース・アドレス（続き）

DMA チャンネル識別マーク	MMR ベース・アドレス	レジスタ接頭辞
4	0xFFC0 0D00	DMA4_
5	0xFFC0 0D40	DMA5_
6	0xFFC0 0D80	DMA6_
7	0xFFC0 0DC0	DMA7_
Mem DMA ストリーム 0 デスティネーション	0xFFCO 0E00	MDMA_D0_
Mem DMA ストリーム 0 ソース	0xFFC0 0E40	MDMA_S0_
Mem DMA ストリーム 1 ソース	0xFFCO 0E80	MDMA_D1_
Mem DMA ストリーム 1 ソース	0xFFC0 0EC0	MDMA_S1_

表 B-14.　DMA レジスタの接尾辞とオフセット

レジスタ接頭辞	ベースからのオフセット	参照先
NEXT_DESC_PTR	0x00	9-8 ページの「DMAx_NEXT_DESC_PTR/MDMA_yy_NEXT_DESC_PTR レジスタ」
START_ADDR	0x04	9-10 ページの「DMAx_START_ADDR/MDMA_yy_START_ADDR レジスタ」
CONFIG	0x08	9-12 ページの「DMAx_CONFIG/MDMA_yy_CONFIG レジスタ」
X_COUNT	0x10	9-17 ページの「DMAx_X_COUNT/MDMA_yy_X_COUNT レジスタ」
X_MODIFY	0x14	9-18 ページの「DMAx_X_MODIFY/MDMA_yy_X_MODIFY レジスタ」

表 B-14. DMA レジスタの接尾辞とオフセット（続き）

レジスタ接頭辞	ベースからのオフセット	参照先
Y_COUNT	0x18	9-20 ページの「DMAx_Y_COUNT/MDMA_yy_Y_COUNT レジスタ」
Y_MODIFY	0x1C	9-21 ページの「DMAx_Y_MODIFY/MDMA_yy_Y_MODIFY レジスタ」
CURR_DESC_PTR	0x20	9-23 ページの「DMAx_CURR_DESC_PTR/MDMA_yy_CURR_DESC_PTR レジスタ」
CURR_ADDR	0x24	9-25 ページの「DMAx_CURR_ADDR/MDMA_yy_CURR_ADDR レジスタ」
IRQ_STATUS	0x28	9-32 ページの「DMAx_IRQ_STATUS/MDMA_yy_IRQ_STATUS レジスタ」
PERIPHERAL_MAP	0x2C	9-29 ページの「DMAx_PERIPHERAL_MAP/MDMA_yy_PERIPHERAL_MAP レジスタ」
CURR_X_COUNT	0x30	9-26 ページの「DMAx_CURR_X_COUNT/MDMA_yy_CURR_X_COUNT レジスタ」
CURR_Y_COUNT	0x38	9-28 ページの「DMAx_CURR_Y_COUNT/MDMA_yy_CURR_Y_COUNT レジスタ」

外部バス・インターフェース・ユニット・レジスタ

外部バス・インターフェース・ユニット・レジスタ（0xFFC0 0A00～0xFFC0 0AFF）

表 B-15.　外部バス・インターフェース・ユニット・レジスタ

メモリマップド・アドレス	レジスタ名	参照先
0xFFC0 0A00	EBIU_AMGCTL	17-10 ページの「EBIU_AMGCTL レジスタ」
0xFFC0 0A04	EBIU_AMBCTL0	17-12 ページの「EBIU_AMBCTL0 レジスタと EBIU_AMBCTL1 レジスタ」
0xFFC0 0A08	EBIU_AMBCTL1	17-12 ページの「EBIU_AMBCTL0 レジスタと EBIU_AMBCTL1 レジスタ」
0xFFC0 0A10	EBIU_SDGCTL	17-34 ページの「EBIU_SDGCTL レジスタ」
0xFFC0 0A14	EBIU_SDBCTL	17-46 ページの「EBIU_SDBCTL レジスタ」
0xFFC0 0A18	EBIU_SDRRC	17-50 ページの「EBIU_SDRRC レジスタ」
0xFFC0 0A1C	EBIU_SDSTAT	17-49 ページの「EBIU_SDSTAT レジスタ」

付録C　テスト機能

この章では、プロセッサのテスト機能について説明します。

JTAG 規格

プロセッサは、JTAG（Joint Test Action Group）規格とも呼ばれるIEEE 1149.1規格と完全な互換性があります。

JTAG規格では、組み立てられたプリント回路ボードのテスト、メンテナンス、サポートを支援するよう構築できる回路を定義します。この回路に内蔵される標準インターフェースを通じて、命令とテスト・データの通信が可能です。バウンダリスキャン・レジスタなど、一連のテスト機能が定義されており、部品は、プリント回路ボードのテストを支援するように設計された最小の命令セットに対応できます。

この規格で定義するテスト・ロジックを集積回路に組み込むと、以下に対して標準化された手法が提供されます。

- プリント回路ボード上で組み立てられた集積回路間の相互接続のテスト
- 集積回路自身のテスト
- 通常の部品動作時に行う、回路動作の観察または変更

テスト・ロジックは、バウンダリスキャン・レジスタやその他のビルディング・ブロックから構成されます。テスト・ロジックは、テスト・アクセス・ポート（TAP）を通じてアクセスされます。

JTAG 規格の詳細については、『IEEE Standard Test Access Port and Boundary-Scan Architecture』（ISBN 1-55937-350-4）を参照してください。

バウンダリスキャン・アーキテクチャ

バウンダリスキャン・テスト・ロジックは、以下の要素から構成されます。

- 5本のピンで構成されるTAP（表 C-1を参照）
- テスト・レジスタを通じてすべてのイベントの順序付けを制御するTAPコントローラ
- 5ビットの命令コードを解釈して、希望のテスト動作を実行するテスト・モードを選択する命令レジスタ（IR）
- JTAG規格によって定義される複数のデータ・レジスタ

表 C-1.　テスト・アクセス・ポート・ピン

ピン名	入力／出力	説明
TDI	入力	テスト・データ入力
TMS	入力	テスト・モード選択
TCK	入力	テスト・クロック
$\overline{\text{TRST}}$	入力	テスト・リセット
TDO	出力	テスト・データ出力

TAPコントローラは、16ステートの同期式有限ステート・マシンであり、TCKピンとTMSピンによって制御されます。図中の各ステートへの遷移は、TCKの立上がりエッジで発生し、TMSピンのステート（ここでは、ロジック1またはロジック0）によって定義されます。動作の詳細については、JTAG規格を参照してください。

図 C-1には、TAPコントローラのステート図を示します。

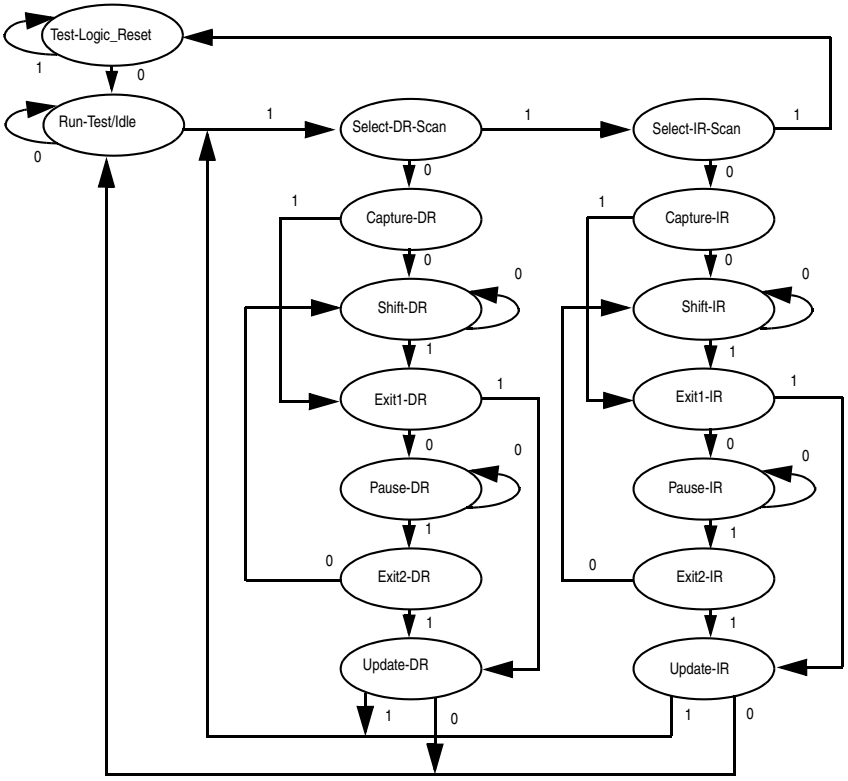


図 C-1.　TAP コントローラのステート図

注：

- 5 TCKサイクルの後でTMSがハイレベルに保持されている場合、TAPコントローラはTest-Logic-Resetステートに入ります。
- TRST が非同期にアサートされている場合、TAP コントローラはTest-Logic-Resetステートに入ります。
- 外部システム・リセットは、TAP コントローラのステートに影響を与えることはなく、TAPコントローラのステートも、外部システム・リセットに影響を与えることはありません。

■ 命令レジスタ

命令レジスタは5ビット幅であり、32個までのバウンダリスキャン命令に対応します。

命令レジスタは、パブリック命令とプライベート命令の両方を保持します。JTAG規格では、いくつかのパブリック命令が必須であり、その他のパブリック命令はオプションです。プライベート命令は、メーカー使用のために予約されています。

表C-2のバイナリ・デコード列には、パブリック命令のデコードを一覧にします。レジスタ列には、シリアル・スキャン・パスを一覧にします。

表 C-2. パブリック JTAG スキャン命令のデコード

命令名	バイナリ・デコード 01234	レジスタ
EXTEST	00000	バウンダリスキャン
SAMPLE/PRELOAD	10000	バウンダリスキャン
BYPASS	11111	バイパス

図 C-2には、表C-2で示されるパスに対する命令ビットのスキャン順序を示します。

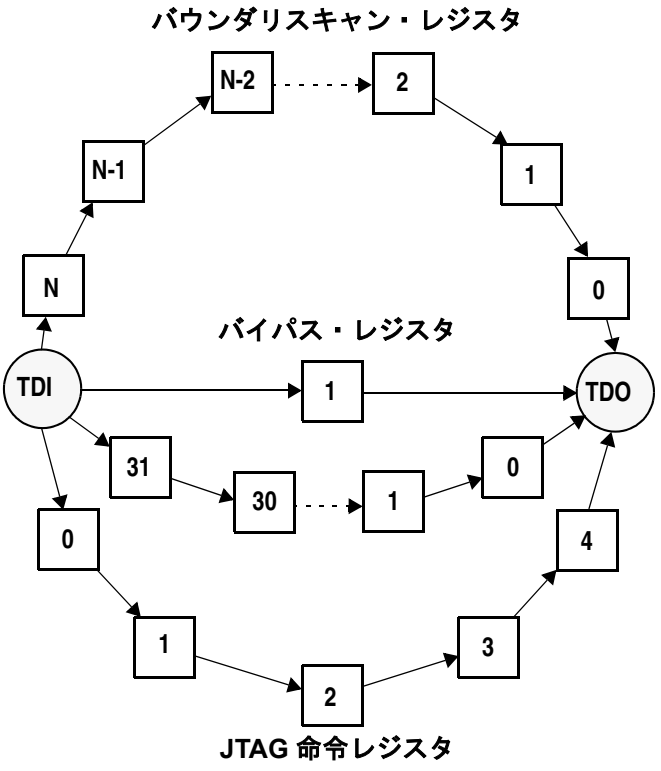


図 C-2.　シリアル・スキャン・パス

■ パブリック命令

次の節では、パブリックなJTAGスキャン命令について説明します。

▶ EXTEST—バイナリ・コード 00000

EXTEST命令では、TDIピンとTDOピンとの間を接続するバウンダリスキャン・レジスタを選択します。この命令によって、デバイスの外部で内蔵回路のテストができます。

EXTEST命令によって、内部データをバウンダリ出力に駆動したり、外部データをバウンダリ入力でキャプチャしたりできます。

バウンダリ出力がオーバードライブされたり、バウンダリ入力で信号が受信されたりした場合に、内部ロジックを保護するには、プロセッサの出力ピンでは他にデータを駆動するものがないことを確認してください。

▶ SAMPLE/PRELOAD—バイナリ・コード 10000

SAMPLE/PRELOAD命令では2つの機能を実行し、TDIとTDOとの間を接続するバウンダリスキャン・レジスタを選択します。この命令による内部ロジックへの影響はありません。

命令のSAMPLE部分によって、バウンダリスキャン・セルで入力と出力のスナップショットをキャプチャできます。データは、TCKの立上がりエッジでサンプリングされます。

命令のPRELOAD部分によって、データはデバイス・ピンにロードでき、EXTEST命令によってボード上に駆動できます。データはTCKの立下がりエッジでピンにプリロードされます。

▶ BYPASS—バイナリ・コード 11111

BYPASS命令では、TDIとTDOに接続されるバイパス・レジスタを選択します。この命令による内部ロジックへの影響はありません。TDIとTDOとの間ではデータ反転が行われてはいけません。

■ バウンダリスキャン・レジスタ

バウンダリスキャン・レジスタは、EXTEST命令とSAMPLE/PRELOAD命令によって選択されます。これらの命令によって、プロセッサのピンを制御し、ボードレベル・テスト用にサンプリングできます。

付録D　数値フォーマット

ADSP-BF53x　Blackfinファミリーのプロセッサは、ハードウェアで8/16/32/40ビットの固定小数点データに対応します。演算ユニット内の特殊機構によって、ソフトウェアで他のフォーマットにも対応できます。この付録では、これらのデータ・フォーマットのさまざまな側面を説明します。また、ソフトウェアでブロック浮動小数点フォーマットを実装する方法についても説明します。

符号なしと符号付き：2の補数フォーマット

符号なし整数値は正であり、ビットに符号情報は含まれません。したがって、符号なし整数の値は通常の2進法で解釈されます。多倍長数の最下位ワードは、符号なし数値として扱われます。

ADSP-BF53x Blackfinファミリーによって対応される符号付き数値は、2の補数フォーマットです。符号付き絶対値、1の補数、BCD、およびn増しフォーマットは対応されません。

整数と小数

ADSP-BF53x Blackfinファミリーは、小数と整数の両方のデータ・フォーマットに対応します。整数では、小数点はLSBの右側にあり、すべての絶対値ビットは1以上の重みを持つと想定されます。このフォーマットを図D-1に示します。なお、2の補数フォーマットでは、符号ビットは負の重みを持っています。

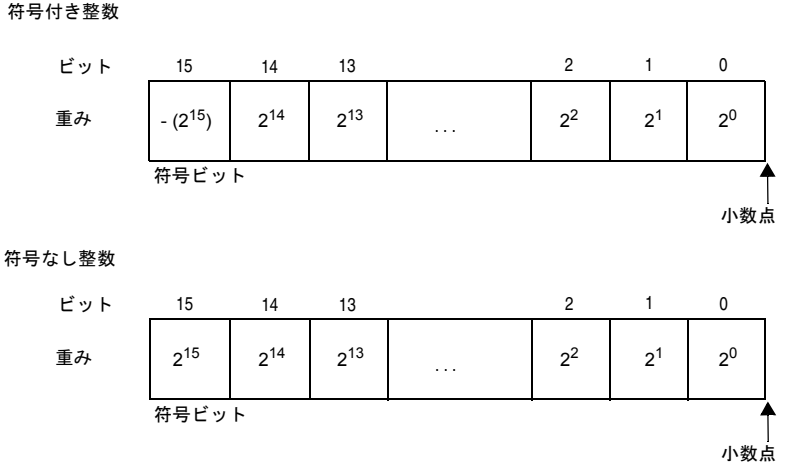


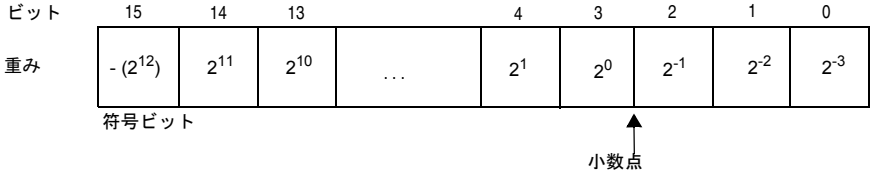
図 D-1.　整数フォーマット

小数フォーマットでは、仮想小数点は数値の中に置かれ、絶対値ビットの一部または全部は1より小さな重みを持ちます。図 D-2に示すフォーマットでは、仮想小数点は3つのLSBの左側に置かれ、ビットはこの図に示された重みを持ちます。

Blackfinプロセッサ・ファミリーのネイティブ・フォーマットは、符号付き小数1.Mフォーマットと符号なし小数0.Nフォーマットです。ここで、Nはデータ・ワード内のビット数であり、M＝N－1です。

フォーマットの記述に使用される表記は、ピリオド（.）で区切られた2つの数値から構成されます。最初の数値は小数点の左側にあるビット数であり、2番目の数値は小数点の右側にあるビット数です。たとえば、16.0フォーマットは整数フォーマットであり、すべてのビットは小数点の左側にあります。図 D-2のフォーマットは13.3です。

符号付き小数（13.3）



符号なし小数（13.3）

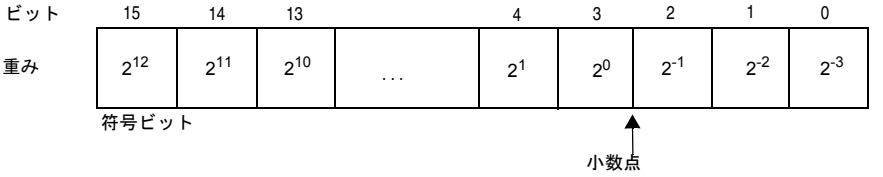


図 D-2.　小数フォーマットの例

表D-1は、16ビットによる小数フォーマットで表現できる、符号付き数値の範囲を示します。

表 D-1.　小数フォーマットとその範囲

フォーマット	整数ビットの数	小数ビットの数	10 進での正の最大値（0x7FFF）	10 進での負の最大値（0x8000）	10 進での 1 LSB の値（0x0001）
1.15	1	15	0.999969482421875	–1.0	0.000030517578125
2.14	2	14	1.999938964843750	–2.0	0.000061035156250
3.13	3	13	3.999877929687500	–4.0	0.000122070312500
4.12	4	12	7.999755859375000	–8.0	0.000244140625000
5.11	5	11	15.999511718750000	–16.0	0.000488281250000
6.10	6	10	31.999023437500000	–32.0	0.000976562500000
7.9	7	9	63.998046875000000	–64.0	0.001953125000000
8.8	8	8	127.996093750000000	–128.0	0.003906250000000
9.7	9	7	255.992187500000000	–256.0	0.007812500000000
10.6	10	6	511.984375000000000	–512.0	0.015625000000000
11.5	11	5	1023.968750000000000	–1024.0	0.031250000000000
12.4	12	4	2047.937500000000000	–2048.0	0.062500000000000
13.3	13	3	4095.875000000000000	–4096.0	0.125000000000000
14.2	14	2	8191.750000000000000	–8192.0	0.250000000000000
15.1	15	1	16383.500000000000000	–16384.0	0.500000000000000
16.0	16	0	32767.000000000000000	–32768.0	1.000000000000000

2 進乗算

加算と減算では、両方のオペランドを同じフォーマット（符号付きまたは符号なし、小数点は同じ位置）にする必要があり、結果のフォーマットは入力フォーマットと同じです。入力は符号付きであっても符号なしであっても、加算と減算は同じように実行されます。

しかし、乗算では、入力はさまざまなフォーマットを持つことができ、結果は入力のフォーマットに依存します。ADSP-BF53x Blackfin ファミリーのアセンブリ言語では、入力として、両方とも符号付き、両方とも符号なし、またはそのいずれか（混合モード）を指定できます。結果における小数点の位置は、各入力の小数点の位置から得られます。これを図 D-3 に示します。2つの16ビット数値の積は32ビット数値です。入力のフォーマットがM.NとP.Qである場合には、積のフォーマットは（M＋P）.（N＋Q）となります。たとえば、2つの13.3数値の積は26.6数値です。2つの1.15数値の積は2.30数値です。

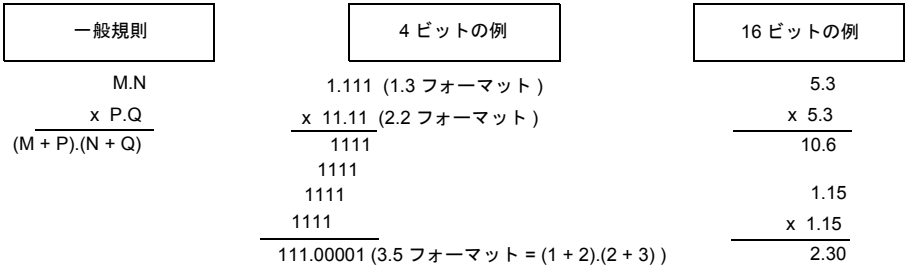


図 D-3.　乗算結果のフォーマット

■ 小数モードと整数モード

2つの2の補数値の積には、2つの符号ビットがあります。これらのビットの1つは冗長であるため、結果全体を1ビット左にシフトできます。さらに、入力の1つが1.15数値であった場合には、左シフトによって、結果はもう一方の入力と同じフォーマットを持つことになります(16ビットの追加精度)。たとえば、1.15数値と5.11数値を乗算すると、6.26数値が得られます。これを1ビット左にシフトすると、結果は5.27数値、つまり5.11数値に16個のLSBを加えたものになります。

ADSP-BF53x Blackfinファミリーで提供される手段（符号付き小数モード）によって、乗算器の結果は、必ず1ビット左にシフトされてから、結果レジスタに書き込まれます。このため、両方のオペランドが符号付きである場合の余分な符号ビットは削除され、正しくフォーマットされた結果が得られます。

両方のオペランドが1.15フォーマットであるとき、結果は2.30です（30個の小数ビット)。左シフトによって乗算器の結果は1.31になり、これは1.15に丸めることができます。したがって、符号付き小数データ・フォーマットを使用する場合には、1.15フォーマットを使用すると最も便利です。

データ・フォーマットの詳細については、2-12ページの表 2-2に示されるデータ・フォーマットを参照してください。

ブロック浮動小数点フォーマット

ブロック浮動小数点フォーマットを利用すると、固定小数点プロセッサは浮動小数点演算に要求されるオーバーヘッドなしで、浮動小数点フォーマットという広いダイナミック・レンジの一部を得ることができます。しかし、ブロック浮動小数点フォーマットを維持するには、追加のプログラミングが必要となります。

浮動小数点数に含まれる指数は、実際の値における小数点の位置を示します。ブロック浮動小数点フォーマットでは、一連（ブロック）のデータ値が共通の指数を共有します。1ブロックの固定小数点値をブロック浮動小数点フォーマットに変換するには、各値を同じ量だけ左にシフトし、そのシフト値をブロック指数として格納します。

一般に、ブロック浮動小数点フォーマットを使用すれば、無視できるMSBをシフト・アウトして、各値で利用可能な精度を増やすことができます。また、データ値がオーバーフローする可能性をなくすこともできます。図D-4を参照してください。この例では、3つのデータ・サンプルのそれぞれには、2つ以上の無視できる冗長な符号ビットがあります。各データ値はオーバーフローする前に、この2ビット（2桁）だけ成長することができます。これらのビットはガード・ビットと呼ばれます。

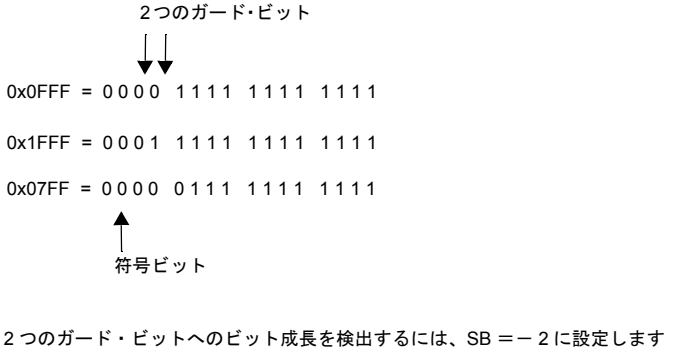


図 D-4. ガード・ビットを持つデータ

あるプロセスで、どの値も2つのガード・ビットを超えて成長しないことがわかっている場合には、そのプロセスはデータの損失なしに実行できます。しかし、その後で、次のプロセスの前にブロックを調整してガード・ビットを置き換える必要があります。

図 D-5は、処理後で調整前のデータを示します。ブロック浮動小数点の調整は、次のように行われます。

- SIGNビットS命令の出力はSBであり、SBはEXPADJ命令で引数として使用されると想定します(これらの命令の使い方と構文については『ADSP-BF53x Blackfin Processor Instruction Set Reference』を参照してください)。最初に、SBの値は＋2であり、2つのガード・ビットに対応しています。処理中に、結果として得られる各データ値は、EXPADJ命令によって検査されます。この命令は、冗長な符号ビットの数をカウントし、冗長な符号ビットの数が2より少ない場合にはSBを調整します。この例では、処理後にはSB＝＋1であり、2つのガード・ビットを維持するためにデータのブロックを1ビット右にシフトする必要があることを示しています。
- 処理後にSBが0であった場合には、ブロックを2ビット右にシフトする必要があります。いずれの場合も、シフトを反映するためにブロック指数が更新されます。

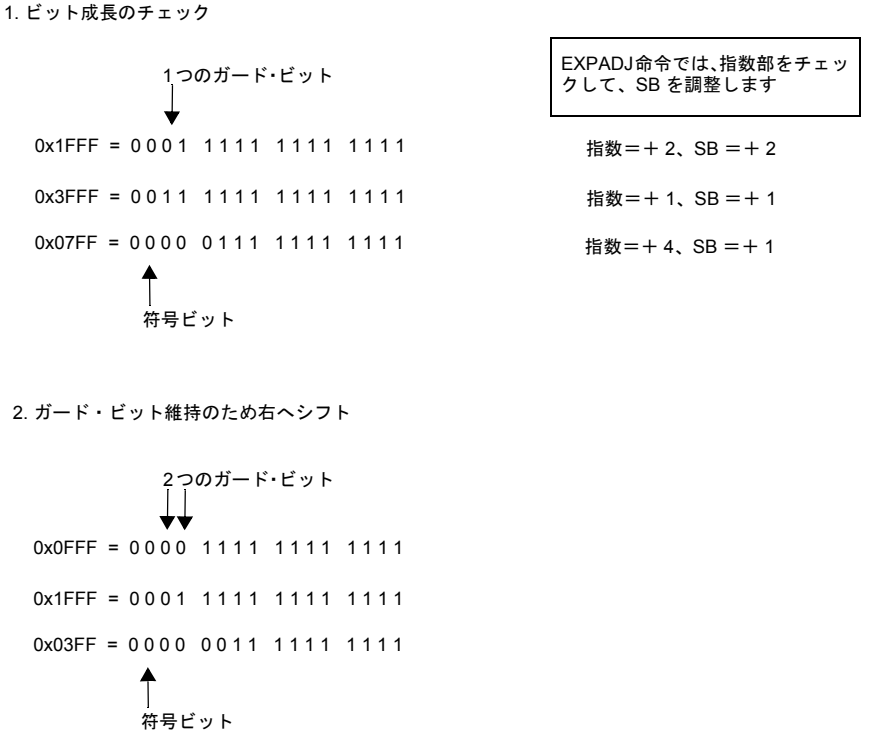


図 D-5.　ブロック浮動小数点の調整

付録G　用語集

ALU

算術／論理演算ユニットを参照してください。

AMC（非同期メモリ・コントローラ）

SRAM、ROM、フラッシュなど、複数のバンクの非同期メモリをサポートする設定可能なメモリ・コントローラであり、各バンクはさまざまなタイミング・パラメータで独立してプログラムできます。

Bank Activateコマンド

Bank Activateコマンドによって、SDRAMは（行アドレスによって指定される）行内に（バンク・アドレスによって指定される）内部バンクを開きます。SDRAMに対してBank Activateコマンドが発行されると、SDRAMは専用のバンク内に新しい行アドレスを開きます。開いている内部バンクと行内のメモリは、開いているページと呼ばれます。Bank Activateコマンドは、読出しコマンドや書込みコマンドの前に適用する必要があります。

Burst Stopコマンド

Burst Stopコマンドは、バースト読出しまたは書込みの動作を終了させる方法の1つです。

CAM（コンテント・アドレッサブル・メモリ）

アソシエイティブ・メモリとも呼ばれます。記憶の各ビットとの比較ロジックを内蔵するメモリ・デバイスです。データ値は、メモリ内のすべてのワードにブロードキャストされます。そのデータは格納された値と比較され、一致する値にはフラグが立てられます。

CAS（列アドレス・ストローブ）

列アドレス・ラインが有効であることを示すために、SDCからDRAMデバイスに送信される信号です。

CAS遅延（tAA、tCAC、CL）

列アドレス・ストローブ（CAS）遅延は、SDRAMが読出しコマンドを検出してから、データをその出力ピンに提供するまでの、クロック・サイクル単位での遅延です。

CBR（CAS Before RAS）メモリ・リフレッシュ

DRAMデバイスには、リフレッシュ行アドレス用のビルトイン・カウンタがあります。RASをアクティブにする前に、CASをアクティブにすることによって、このカウンタは、アドレス入力の代わりに行アドレスを供給するよう選択されます。

CEC

コア・イベント・コントローラを参照してください。

CPLB

キャッシュ可能性保護索引バッファを参照してください。

DAB

DMAアクセス・バスを参照してください。

DAG

データ・アドレス・ジェネレータを参照してください。

DCB

DMA**コア・バス**を参照してください。

DEB

DMA**外部バス**を参照してください。

DFT（Design For Testability＝テスト性を配慮した設計）

デジタル・システムの設計者がシステムのテスト性を保証するために役立つ、一連の技法です。

Direct Memory Access（DMA）

システム・デバイスとメモリとの間でデータを転送する方法の1つであり、データはプロセッサを介さずに、DMAポートを通じて転送されます。

DMA

Direct Memory Accessを参照してください。

DMAアクセス・バス（DAB）

DMAチャンネルがペリフェラルによってアクセスされる手段を提供するバスです。

DMA外部バス（DEB）

DMAチャンネルがオフチップ・メモリにアクセスするための手段を提供するバスです。

DMAコア・バス（DCB）

DMAチャンネルがオンチップ・メモリにアクセスするための手段を提供するバスです。

DMAチェーン

複数のDMAシーケンスのリンクまたはチェーンです。チェーンされたDMAでは、現在のDMAが完了して次のDMAシーケンスが自動的に初期化されるときに、I/Oプロセッサは次のDMAディスクリプタをDMAパラメータ・レジスタにロードします。

DMAディスクリプタ・レジスタ

Direct Memory Access（DMA）プロセスのための初期化情報を保持するレジスタです。

DPMC（ダイナミック・パワー・マネジメント・コントローラ）

プロセッサの制御ブロックの1つであり、これによってユーザは、プロセッサの性能特性と消費電力を動的に制御できます。

DQMデータI/Oマスク機能

SDQM[1:0]ピンは、SDRAMへの8ビット書込みでバイト・マスク機能を提供します。

DRAM（ダイナミックRAM）

半導体メモリの一種であり、データは、コンデンサとトランジスタで構成されるセル・アレイに電荷として格納されます。これらのセルは、行と列で構成される格子内のチップ上に配置されます。コンデンサは徐々に放電して、セルはその情報を失っていくため、セル・アレイを定期的にリフレッシュする必要があります。

DSP

デジタル・シグナル・プロセッサを参照してください。

EAB

外部アクセス・バスを参照してください。

EBC

外部バス・コントローラを参照してください。

EBIU

外部バス・インターフェース・ユニットを参照してください。

EPB

外部ポート・バスを参照してください。

EPROM（消去可能プログラマブル読出し専用メモリ）

半導体メモリの一種であり、データは外部電源なしでほぼ無期限に電荷を保持する、分離された（「フローティング」）トランジスタ・ゲートに電荷として格納されます。EPROMは、フローティング・ゲートに電荷を「注入」することによってプログラムされます。これは、比較的高電圧（通常は12～25V）を必要とするプロセスです。パッケージ内の石英ガラスの窓を通じてチップの表面に紫外線が照射されると、フローティング・ゲートが放電されるため、チップの再プログラムが可能になります。

EVT（イベント・ベクトル・テーブル）

メモリに格納されているテーブルであり、16個の32ビット・エントリを含んでいます。各エントリには、割込みサービス・ルーチン（ISR）用のベクトル・アドレスが含まれています。イベントが発生すると、対応するEVTエントリ内のアドレス位置から命令のフェッチが開始されます。ISRを参照してください。

FFT（高速フーリエ変換）

一連の離散的データ値のフーリエ変換を計算するアルゴリズムです。FFTでは、データ・ポイントの有限な集合（たとえば、リアルワールドの信号の定期的なサンプリング）を、その成分周波数によって表現します。あるいは逆に、FFTでは、周波数データから信号を復元します。FFTを使用すれば、2つの多項式を乗算することもできます。

FIFO（先入れ先出し）

ハードウェア・バッファまたはデータ構造の1つであり、アイテムは格納された順に取り出されます。

HLL（高水準言語）

アセンブリ言語を超えるレベルの抽象化を提供するプログラミング言語であり、通常は英語に似たステートメントを使用して、各コマンドやステートメントは複数のマシン命令に対応します。

IDLE

この命令を実行すると、プロセッサは動作を停止して、割込みが発生するまで現在の状態を保持します。その後、プロセッサは割込みを処理してから、通常の実行を続行します。

IrDA（Infrared Data Association＝赤外線通信協会）

赤外線スペクトルを使用するデバイスの品質と相互運用性を保証するための規格を制定した非営利の業界団体です。

ISR（割込みサービス・ルーチン）

特定の割込みが発生したときに実行されるソフトウェアです。下位メモリに格納されたテーブルには、ポインタ（ベクトルとも呼ばれます）が含まれています。このポインタによって、プロセッサは対応するISRを処理します。EVTを参照してください。

JTAG（Joint Test Action Group）

電子デバイスをテストするためのテスト・アクセス・ポート用のIEEE 1149.1規格を定義する、IEEE標準化作業部会です。

JTAGポート

チャンネルやポートの1つであり、システム・テスト用のIEEE規格である1149.1 JTAG規格に対応します。この規格は、システム内の各コンポーネントのI/Oステータスを連続的にスキャンする方法を定義します。

LIFO（後入れ先出し）

データ構造の1つであり、最後に格納されたアイテムが次に取り出されるアイテムとなります。

LRU

最長時間未使用アルゴリズムを参照してください。

LSB

最下位ビットを参照してください。

MAC（積和）

2つの数値を乗算してから、3番目の数値を加算して結果を得る数値演算です。**積和演算器**を参照してください。

MMR（メモリマップド・レジスタ）

プロセッサによってあたかもレジスタであるかのように使用される、メイン・メモリ内の位置です。

MMU

メモリ・マネジメント・ユニットを参照してください。

MSB（最上位ビット）

2進数の通常の表現において最初または左端のビットであり、最大の重み（2（n－1））を持つ2進数のビットです。

NMI（マスク不能割込み）

別の割込みによって無効にできない、優先順位の高い割込みです。

NRZ（非ゼロ復帰）

2進符号化方式の1つであり、1は信号の変化によって表され、0は変化がないことによって表されます。符号化されたビット間では、基準（ゼロ）電圧に戻りません。この方法では、クロック信号が不要になります。

NRZI（非ゼロ復帰逆転）

2進符号化方式の1つであり、0は信号の変化によって表され、1は変化がないことによって表されます。符号化されたビット間では、基準（ゼロ）電圧に戻りません。この方法では、クロック信号が不要になります。

PAB

ペリフェラル・アクセス・バスを参照してください。

PC（プログラム・カウンタ）

次に実行される命令のアドレスを含むレジスタです。

PF（プログラマブル・フラグ）

汎用のI/Oピンです。各PFピンは、入力ピンまたは出力ピンとして個々に設定でき、さらには、割込みを生成するようにも設定できます。

PLL

フェーズ・ロック・ループを参照してください。

Prechargeコマンド

Prechargeコマンドでは、アクティブ・ページ内の特定の内部バンク、またはページ内のすべての内部バンクを閉じます。

PWM（パルス幅変調）

これはPulse Duration Modulation（PDM）とも呼ばれ、パルスの持続時間が変調電圧によって変更されるパルス変調方式です。

RAS（行アドレス・ストローブ）

行アドレス・ラインが有効であることを示すために、SDCからDRAMデバイスに送信される信号です。

ROM（読出し専用メモリ）

固定した内容で製造されたデータ記憶デバイスです。この用語は一般に、不揮発性半導体メモリを表すために使用されます。

RTC

リアルタイム・クロックを参照してください。

RZ（ゼロ復帰変調）

2進符号化方式の1つであり、どのビットでも2つの信号パルスが使用されます。0は低電圧レベルから高電圧レベルへの変化によって表され、1は高電圧レベルから低電圧レベルへの変化によって表されます。符号化されたビット間では、基準（ゼロ）電圧への復帰が行われます。

RZI（ゼロ復帰逆転変調）

2進符号化方式の1つであり、どのビットでも2つの信号パルスが使用されます。1は低電圧レベルから高電圧レベルへの変化によって表され、0は高電圧レベルから低電圧レベルへの変化によって表されます。符号化されたビット間では、基準（ゼロ）電圧への復帰が行われます。

SDC（SDRAMコントローラ）

SDRAMで構成される同期メモリのバンクをサポートする、設定可能なメモリ・コントローラです。

SDRAM（Synchronous Dynamic Random Access Memory）

DRAMの一形態であり、他の制御信号と共にクロック信号を含んでいます。このクロック信号によって、SDRAMデバイスは、一連の連続ビットをクロック同期出力する「バースト」アクセス・モードに対応できます。

SDRAMバンク

外部メモリの領域であり、16Mバイト、32 Mバイト、64 Mバイト、または128 Mバイトに設定することができ、$\overline{\text{SMS}}$ピンによって選択されます。

Self-Refresh

SDRAMがSelf-Refreshモードにある場合、SDRAMの内部タイマは外部の制御入力なしで、Auto-Refreshサイクルを定期的に開始します。SDCはSDRAMを低消費電力モードにするために、Self-Refreshコマンドを含む一連のコマンドを発行する必要があり、Self-Refreshモードを終了するために、別の一連のコマンドを発行する必要があります。Self-Refreshモードへの切り替えはSDRAMメモリ・グローバル・コントロール・レジスタ（EBIU_SDGCTL）でプログラムされ、SDRAMアドレス空間へのアクセスによって、SDCはSDRAMをSelf-Refreshモードから終了させます。17-40ページの「Self-Refreshモードへの出入り（SRFS）」を参照してください。

Serial Peripheral Interface（SPI）

集積回路の接続に使用する同期シリアル・プロトコルです。

SIC（システム割込みコントローラ）

プロセッサの2レベル・イベント制御メカニズムの一部です。SICは、コア・イベント・コントローラ（CEC）と連携して、すべてのシステム割込みの優先順位付けと制御を行います。SICは、コアの優先順位付けされた汎用割込み入力とペリフェラル割込みソースとの間のマッピングを提供します。

SIMD（Single Instruction, Multiple Data）

並列コンピュータ・アーキテクチャの1つであり、1つの命令で、複数のデータ・オペランドが同時に処理されます。

SP（スタック・ポインタ）

スタックの先頭を指すレジスタです。

SPI

シリアル・ペリフェラル・インターフェースを参照してください。

SRAM

スタティックRAMを参照してください。

TAP（テスト・アクセス・ポート）

JTAGポートを参照してください。

TDM

時分割多重を参照してください。

UART

非同期シリアル・インターフェースLSIを参照してください。

W1C

Write-1-to-Clearを参照してください。

W1S

Write-1-to-Setを参照してください。

Write-1-to-Clear（W1C）ビット

1を書き込むことによってクリア（＝0）できる制御ビットまたはステータス・ビットです。

Write-1-to-Set（W1S）ビット

1を書き込むことによってセットされる制御ビットまたはステータス・ビットです。0を書き込んでもクリアできません。

アイソクロナス

データを一定の時間制約のもとで配信する必要のあるプロセスです。

圧伸

(圧縮／伸張)。送信すべきビット数を最小限に抑えるため、データを対数的にエンコードおよびデコードするプロセスです。

後更新アドレッシング

データ転送時にDAGがアドレスを提供し、命令が実行された後で自動インクリメントするプロセスです。

インデックス

アレイ要素（たとえば、ライン・インデックス）の選択に使用されるアドレス部分です。

インデックス・レジスタ

アドレスを保持し、メモリへのポインタとして機能するDAGレジスタです。

ウェイ

*N*ウェイ・キャッシュ内のライン記憶要素のアレイです。

エッジ・センシティブ割込み

CLKINの立上がりエッジでサンプリングしたとき、入力信号が1つのサイクルでハイレベル（非アクティブ）、次のサイクルでローレベル（アクティブ）かどうかをプロセッサが検出する信号または割込み。

エンディアン・フォーマット

マルチバイト番号におけるバイトの順序。

外部アクセス・バス（EAB）

外部メモリにアクセスするために、コア・メモリ・マネジメント・ユニットによって管理されるバスです。

外部バス・インターフェース・ユニット（EBIU）

外部メモリへのグルーレスなインターフェースを提供するコンポーネントです。EBIUは、コアやDMAチャンネルからの外部メモリに対する要求を処理します。

外部バス・コントローラ（EBC）

外部アクセス・バス（EAB）とDMA外部バス（DEB）との間を調停するコンポーネントであり、1サイクル当たり多くても1つのリクエスタが許可されます。

外部ポート

チャンネルまたはポートの1つであり、プロセッサの内部アドレス・バスとデータ・バスをチップ外に拡張して、オフチップ・メモリとペリフェラルにプロセッサのインターフェースを提供します。

外部ポート・バス（EPB）

EBIUの出力を外部デバイスに接続するバスです。

間接分岐

実行時に評価される、データ・アドレス・ジェネレータからの動的アドレスを使用するJump命令やCall/Return命令です。

キャッシュ可能性保護索引バッファ（CPLB）

コア・メモリ・マップのアクセス特性を記述する記憶領域です。

キャッシュ・タグ

キャッシュされたラインが表す、メモリ内の特定アドレス・ソースを識別するために、キャッシュされたデータ・ラインと一緒に格納される上位アドレス・ビットです。

キャッシュ・ヒット

キャッシュ内の有効な存在するエントリによって満足されたメモリ・アクセスです。

キャッシュ・ブロック

キャッシュ・ミスの結果として、キャッシュと次のレベルのメモリとの間で転送されるメモリの最小単位です。

キャッシュ・ミス

キャッシュ内のどの有効なエントリにも一致しないメモリ・アクセスです。

キャッシュ・ライン

キャッシュ・ブロックと同じです。このドキュメントでは、**キャッシュ・ブロック**の代わりに**キャッシュ・ライン**が使用されます。

グルーレス

外部ハードウェアが不要です。

コア

コアは、CPU、L1メモリ、イベント・コントローラ、コア・タイマ、性能モニタリング・レジスタという機能ブロックから構成されます。

コア・イベント・コントローラ（CEC）

CECは、システム割込みコントローラ（SIC）と連携して、すべてのシステム割込みの優先順位付けと制御を行います。CECでは、SICから転送された割込みと汎用割込みを処理します。

最下位ビット（LSB）

2進数の通常の表現において最後または右端のビットであり、1の数を指定する2進数のビットです。

最長時間未使用アルゴリズム

キャッシュで使用される置換アルゴリズムであり、最も長期間にわたって使用されていないラインを最初に置き換えます。

算術／論理演算ユニット（ALU）

算術機能、比較機能、および論理機能を実行するプロセッサ・コンポーネントです。

システム

システムには、ペリフェラル・セット（タイマ、リアルタイム・クロック、プログラマブル・フラグ、UART、SPORT、PPI、SPI）、外部メモリ・コントローラ（EBIU)、メモリDMAコントローラ、ならびにこれらのペリフェラル間のインターフェース、オプションの外部（オフチップ）リソースが含まれます。

システム・クロック（SCLK）

PLL内のプログラマブルな分周器比率によって決定される周波数でクロック・パルスを配信するコンポーネントです。

システム割込みコントローラ（SIC）

ペリフェラル割込みソースから、コア・イベント・コントローラ（CEC）の優先順位付けされた汎用割込み入力にイベントをマッピングおよびルーティングするコンポーネントです。

シフタ

論理シフトと算術シフトを実行する演算ユニットです。

時分割多重（TDM）

別個の信号を1つのチャンネルで送信するための方式です。送信時間はセグメントに分解され、各セグメントは1つの要素を伝達します。各ワードは連続する次のチャンネルに属するため、たとえば、24ワードのデータ・ブロックには、24つのチャンネルごとに1ワードが格納されています。

ジャンプ

プログラムの流れをプログラム・メモリの別の部分に永続的に移すことです。

循環アドレッシング

DAGが一連のレジスタを「ラップ」したり1ステップずつ反復実行するプロセスです。

条件付き分岐

定義された条件に基づいて実行される、Jump命令やCall/Return命令です。

乗算器

演算ユニットの1つであり、固定小数点の乗算を行い、固定小数点の乗算／加算と乗算／減算の演算を実行します。

シリアル・ポート（SPORT）

プロセッサ上の高速な同期入出力デバイスです。プロセッサで使用する2つの同期シリアル・ポートは、多種多様なデジタルおよびミックスド・シグナル・ペリフェラル・デバイスに対して安価なインターフェースを提供します。

スタック

後入れ先出し（LIFO）順にアクセスされるアイテムを格納するためのデータ構造です。データ・アイテムがスタックに追加されるとき、アイテムは「プッシュ」されると言います。データ・アイテムがスタックから取り出されるとき、アイテムは「ポップ」されると言います。

スタティックRAM（SRAM）

定期的なリフレッシュ操作を必要としない、非常に高速な読出し／書込みメモリです。

精度

数値の記憶フォーマットにおいて、2進小数点の後にあるビット数です。

セット

*N*ウェイ・キャッシュのウェイにおける*N*ライン格納位置のグループであり、アドレスのインデックス・フィールドによって選択されます。

セット・アソシエイティブ

ライン配置を複数のセット（またはウェイ）に限定するキャッシュ・アーキテクチャです。

ダイレクト・マップ方式

キャッシュ内での各ラインの出現場所が固定されているキャッシュ・アーキテクチャです。1ウェイ・アソシエイティブとも呼ばれます。

多機能計算

複数の計算命令の並列実行です。これらの命令は、1つのサイクルで完了し、演算ユニットの並列演算とメモリ・アクセスを組み合わせます。複数の演算は、あたかも対応する単機能計算に含まれているかのように実行されます。

ダーティ、変更済み

タグとともに格納される状態ビットであり、データ・キャッシュ・ライン内のデータが、ソース・メモリからコピーされて以降に変更されたかどうか、したがって、そのソース・メモリ内で更新する必要があるかどうかを示します。

遅延

メモリ・アクセスのための正しい場所を見つけ出し、そこへのアクセスを準備するためのオーバヘッド時間です。

直交

独立しているという特性です。直交命令セットを使用すれば、レジスタを参照する命令ではどのレジスタでも使用できます。

直接分岐

プログラム・ラベルなどの実行時に変化しない絶対アドレスを使用したり、PC相対アドレスを使用したりするJump命令やCall/Return命令です。

ディスクリプタ・ブロック、DMA

一連のDMAシーケンスを記述するために、DMAコントローラによって使用される一連のパラメータです。

ディスクリプタ・ロード、DMA

DMAコントローラがデータ・メモリからDMAディスクリプタをダウンロードし、DMAパラメータ・レジスタを自動的に初期化するプロセスです。

デジタル・シグナル・プロセッサ（DSP）

デジタル形式に変換されたアナログ情報の高速操作用に指定された集積回路です。

データ・アドレス・ジェネレータ（DAG）

データがメモリとレジスタの間で転送されるときにメモリ・アドレスを提供する処理コンポーネントです。

データ・レジスタ（Dreg）

演算ユニット内に置かれているレジスタであり、乗算器演算、ALU演算、またはシフタ演算のオペランドと結果を保持します。

データ・レジスタ・ファイル

一連のデータ・レジスタであり、データを演算ユニットとメモリとの間で転送し、オペランドのローカル・ストレージを提供します。

内部メモリ・バンク

1つのSDRAMには、複数の内部メモリ・バンクがあります。各内部バンクは、同時にアクティブ（オープン）とすることができます。

SDCでは、インターフェースをとるすべてのSDRAMには4つの内部バンクがあるものと想定します。

入力クロック

フェーズ・ロック・ループ（PLL）モジュールによる正確な内部クロック逓倍を可能にするため、タイミング信号の安定したストリームを生成して、周波数、デューティ・サイクル、および安定性を提供するデバイスです。

排他的、クリーン

データ・キャッシュ・ラインの状態であり、そのラインが有効であること、およびラインに含まれているデータがソース・メモリ内のデータと一致することを示します。クリーンなキャッシュ・ライン内のデータは、置き換えの前にソース・メモリに書き込む必要はありません。

バースト・タイプ

SDRAMが読出しコマンドを検出した後でバースト・データを配信するアドレス順、または書込みコマンドを検出した後でバースト・データを格納するアドレス順は、バースト・タイプによって決まります。バースト・タイプは、SDRAMのパワーアップ・シーケンス中にSDRAMでプログラムされます。

バースト長

SDRAMデバイスが、1つの書込みコマンドまたは読出しコマンドを検出した後、それぞれ格納または配信するワード数はバースト長によって決まります。バースト長を選択するには、SDRAMのパワーアップ・シーケンス中にSDRAMのモード・レジスタ内の特定ビットに書き込みます。

ハーバード・アーキテクチャ

プログラムとデータの保管に別個のバスを使用するプロセッサ・メモリ・アーキテクチャです。2本のバスによって、プロセッサはデータ・ワードと命令ワードを同時にフェッチできます。

ビクティム

ダーティ・キャッシュ・ラインであり、これを置き換えてキャッシュ・ライン割当て用の領域を解放するには、その前にメモリに書き込む必要があります。

ビット反転アドレッシング

データ・アドレス・ジェネレータ（DAG）が、格納されたアドレスを反転せずに、データ移動時にビット反転されたアドレスを提供するアドレッシング・モードです。

非同期シリアル・インターフェースLSI（UART）

非同期シリアル通信に必要な送受信回路を含むモジュールです。

フェーズ・ロック・ループ（PLL）

低周波数の入力クロック信号から最高速度のマスター・クロックを生成するオンチップ周波数合成器です。

フォン・ノイマン・アーキテクチャ

ほとんどの非DSPマイクロプロセッサで使用されるアーキテクチャです。このアーキテクチャでは、メモリ・アクセスのために1つのアドレスとデータ・バスを使用します。

ブート・メモリ空間

パワーアップ後、またはソフトウェア・リセット後に、すぐに実行されるプログラムに指定される内部メモリ空間です。

フラッシュ・メモリ

単一トランジスタ・セルの消去可能メモリの一種であり、消去はブロック単位で、あるいはチップ全体に対してのみ行うことができます。

フル・アソシエイティブ

各ラインをキャッシュ内のどこにでも置けるキャッシュ・アーキテクチャです。

ページ・サイズ

同じ行アドレスを持ち、別の行をアクティブにすることなく、連続した読出しコマンドまたは書込みコマンドでアクセスできるメモリの量です。

ベース・アドレス

循環バッファの開始アドレスです。

ベース・レジスタ

循環バッファの開始アドレスを含んでいるデータ・アドレス・ジェネレータ（DAG）レジスタです。

ペリフェラル

コアの一部として組み込まれていない機能ブロックであり、一般にはシステム・レベル動作のサポートに使用されます。

ペリフェラル・アクセス・バス（PAB）

EBIUのメモリマップド・レジスタにアクセスを提供するためのバスです。

飽和（ALU飽和モード）

正のすべての固定小数点オーバーフローが正の最大固定小数点数を返し、負のすべてのオーバーフローが負の最大数を返す状態です。

前更新アドレッシング

データ転送時にDAGがアドレスを提供し、命令が実行される前に自動インクリメントするプロセスです。

無効

キャッシュ・ラインの状態を表します。キャッシュ・ラインが無効であれば、キャッシュ・ラインの一致は起こりません。

メモリ・マネジメント・ユニット（MMU）

キャッシュ可能性保護索引バッファ（CPLB）を使用することで、メモリの保護と選択的なキャッシュをサポートするプロセッサのコンポーネントです。

モディファイ・アドレッシング

DAGが、値またはレジスタの内容によってインクリメントされるアドレスを生成するプロセスです。

モディファイ・レジスタ

レジスタ転送時にインデックス・レジスタがプリモディファイまたはポストモディファイされる、インクリメントまたはステップのサイズを提供するDAGレジスタです。

モード・レジスタ

SDRAMデバイスに含まれる内部設定レジスタであり、SDRAMデバイスの機能を指定できます。

有効

タグとともに格納される状態ビットであり、対応するタグとデータが最新で正しく、メモリ・アクセス要求を満足させるために使用できることを示します。

ライト・スルー

キャッシュ書込みポリシーの1つであり、ストア・スルーとも呼ばれます。書込みデータは、キャッシュ・ラインとソース・メモリの両方に書き込まれます。変更されたキャッシュ・ラインは、置き換えられるときにソース・メモリに **書き込まれません**。

ライト・バック

キャッシュ書込みポリシーの1つであり、コピーバックとも呼ばれます。書込みデータは、キャッシュ・ラインにだけ書き込まれます。変更されたキャッシュ・ラインがソース・メモリに書き込まれるのは、キャッシュ・ラインを置き換えるときだけです。

リアルタイム・クロック（RTC）

時刻、アラーム、およびストップウォッチ・カウントダウン機能など、プロセッサのデジタル・ウォッチ機能用のタイミング・パルスを生成するコンポーネントです。

リトル・エンディアン

プロセッサのネイティブなデータ格納フォーマットです。ワードとハーフ・ワードは、最下位バイトをデータ記憶場所の最下位バイト・アドレスに置き、最上位バイトを最上位バイト・アドレスに置いた状態でメモリ（とレジスタ）に格納されます。

ループ

複数回実行される命令群です。

レベル1（L1）メモリ

メモリとコアの間に介在するメモリ・サブシステムなしで、コアによって直接アクセスされるメモリです。

レベル2（L2）メモリ

コアから1レベル以上隔たったメモリです。L2メモリにはL1メモリより大きな容量がありますが、アクセスに要する遅延は増えます。

レベル・センシティブ割込み

信号や割込みの1つであり、CLKINの立上りエッジでサンプリングされたときに、入力信号がロー（アクティブ）である場合にプロセッサによって検出されます。

レングス・レジスタ

循環バッファでのアドレスの範囲を設定するDAGレジスタです。

割込み

通常の処理を中断し、割込みサービス・ルーチン（ISR）を通じて制御の流れを一時的に変えるイベントです。ISRを参照してください。

I 索引

E

F

G

H

I

N

R

S

T

U

う

え

き

く

け

こ

さ

し

す

せ

そ

た

ち

つ

て

と

な

に

ね

は

ひ

ふ

み

む

め

も

ゆ

よ

ら

り

ろ

わ