日本語参考資料
最新版英語ユーザー・ガイドはこちら

ADuCM3027/ADuCM3029 ユーザー・ガイド
UG-1091

ADuCM3027/ADuCM3029 のセットアップと使用方法

はじめに

このユーザーガイドでは、ADuCM3027/ADuCM3029 マイクロコントローラの機能と特徴について説明します。各セクションでは、異なる機能を説明します。

概要

ADuCM3027/ADuCM3029 プロセッサは、処理、制御、接続のために超低消費電力で統合化されたミックスド・シグナルのマイクロ・コントローラ・システムです。MCU サブシステムは、ARM® Cortex™-M3 プロセッサ、一連のデジタル・ペリフェラル、キャッシュ組み込み SRAM とフラッシュ・メモリだけでなく、A/D コンバータ（ADC）サブシステムや、クロック、リセット、パワー・マネージメント機能を備えたアナログ・サブシステムをベースにしています。

超低ダイナミック・モードと休止モードのパワー・マネージメントに対応するため、ADuCM3027/ADuCM3029 プロセッサは、ダイナミック / ソフトウェア制御のクロック・ゲートや電力ゲートなど、多数の電力モードと機能を備えています。

ADuCM3027/ADuCM3029 の完全な仕様については、製品データシートと ADuCM302x Ultra Low Power ARM Cortex-M3 MCU with Integrated Power Management Hardware Reference Manual を参照してください。

特長

ADuCM3027/ADuCM3029 デバイスに共通するシステム機能を次に示します。

- 最大 26 MHz ARM Cortex®-M3 プロセッサ
- 最大 256 KB 組み込みフラッシュ・メモリ、誤差補正コード（ECC）付き
- 有効電力を下げるための 4 KB キャッシュ（オプション）
- パリティ付き 64 KB システム SRAM
- パワー・マネージメント・ユニット（PMU）
- 多層 AMBA（先進マイクロコントローラ・バス・アーキテクチャ）バス・マトリックス
- 中央ダイレクト・メモリ・アクセス（DMA）コントローラ
- ビーパー・インターフェース
- 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）× 3、インター集積回路（I^2C）、ユニバーサル非同期レシーバー／トランスミッタ（UART）ペリフェラル・インターフェース
- AES（高度暗号化規格）と SHA（セキュア・ハッシュ・アルゴリズム）-256 による暗号化サポート
- リアルタイム・クロック（RTC）× 2
- 汎用タイマー × 3、ウォッチドッグ・タイマー × 1
- プログラマブル汎用入出力（GPIO）ピン
- プログラマブル・ジェネレータ多項式を使用したハードウェア巡回冗長検査（CRC）
- パワーオン・リセット（POR）と電源モニタ（PSM）
- 12 ビット逐次比較レジスタ（SAR）ADC

ADUCM3027/ADUCM3029 機能ブロック図

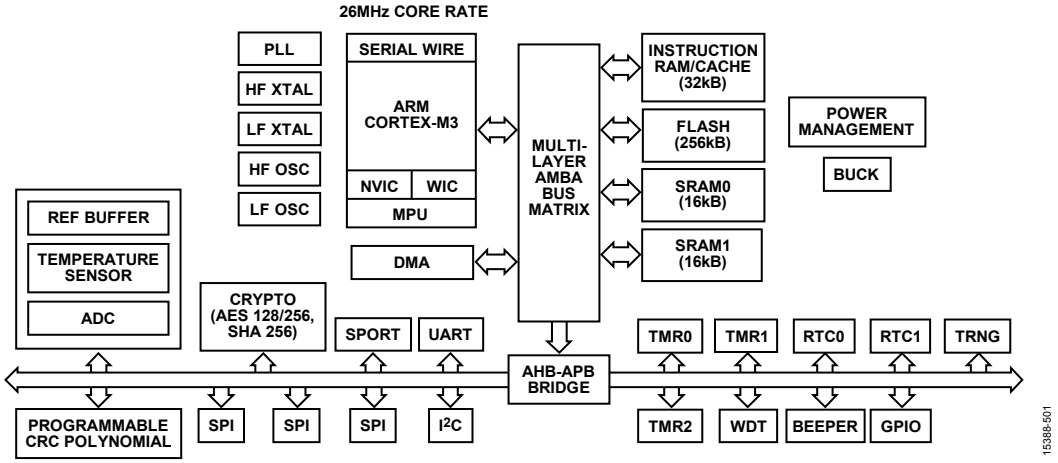


図 1.

Rev. 0

アナログ・デバイセズ株式会社
本　　社／〒105-6891　東京都港区海岸 1-16-1　ニューピア竹芝サウスタワービル
電話 03（5402）8200
大阪営業所／〒532-0003　大阪府大阪市淀川区宮原 3-5-36　新大阪トラストタワー
電話 06（6350）6868

目次

改訂履歴

3/2017—Revision 0: Initial Version

はじめに

ここでは、ADuCM3029/ADuCM3027 マイクロコントローラでアプリケーションを開発するために必要なツールおよびサポート・パッケージについて説明します。ここでは、ADuCM3027/ADuCM3029 を設定するために使用するプログラムをダウンロード、インストール、構成する方法について説明します。

ここでは、IAR ワークベンチを統合開発環境（IDE）として使用し、アプリケーションを開発する手順について説明します。ここでは、ボード・サポート・パッケージ（BSP）ドライバが提供するサンプル・コードをダウンロードして実行する方法についても説明します。

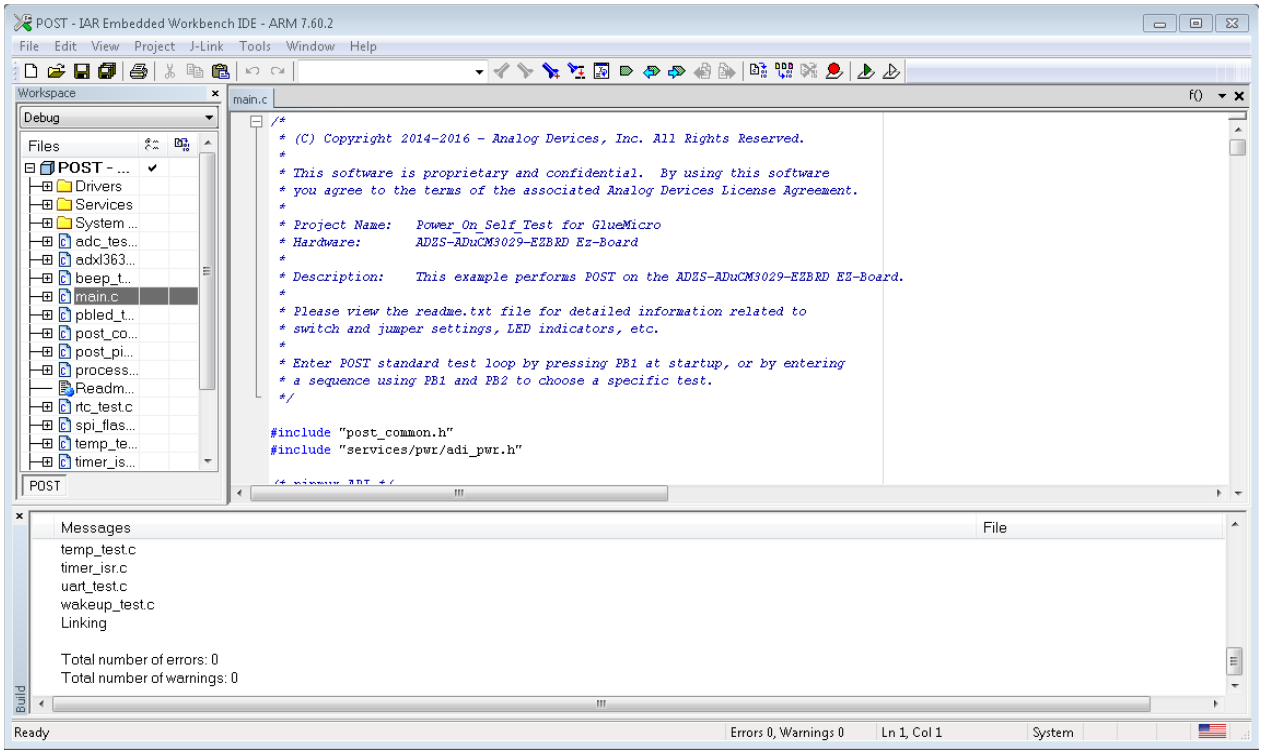


図 2. IAR Embedded Workbench

ソフトウェアのインストール

ADuCM3027/ADuCM3029 を使用したアプリケーション開発に必要なソフトウェア・ツールは、http://www.analog.com/jp/design-center/evaluation-hardware-and-software/evaluation-boards-kits/eval-aducm3029-ezkit.html でダウンロードできます。

表 1. 必要なソフトウェア・ツール

Tool	Functions
IAR Embedded Workbench	Used for compiling, debugging, and code development
Segger J-Link Software	J-Link software and documentation pack includes USB drivers for the emulator, J-Link Commander, and so on
ADuCM3027/ ADuCM3029 BSP Drivers	Includes ADuCM3027/ADuCM3029 peripheral drivers and libraries, IAR configuration files, and example programs

Segger J-Link Driver のインストール

Segger J-Link USB ドライバをインストールしてから、IAR Embedded Workbench などのシリアル・ワイヤ・インターフェースを使用して、デバッグ・コードをダウンロードする必要があります。

J-Link USB ドライバを使用するには、次の手順に従ってください。

1. 最新の Segger J-Link ソフトウェアと技術文書パックをダウンロードします。
2. ダウンロード・ディレクトリで実行可能なソフトウェア・インストーラを開きます。
3. 画面上の指示に従い、インストールを完了します。図 3 に示すように、［**Install USB Driver for J-Link**］オプションがチェックされていることを確認します。

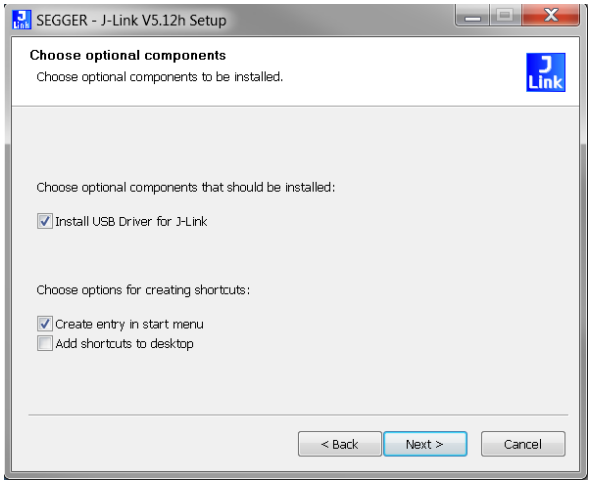


図 3. Segger J-Link ドライバのインストール・オプション

4. エミュレータ・ボードを差し込み、デバイス・マネージャを開きます。
5. USB コントローラの Windows® デバイス・マネージャに表示されるエミュレータ・ボートを確認します（図 4 を参照）。

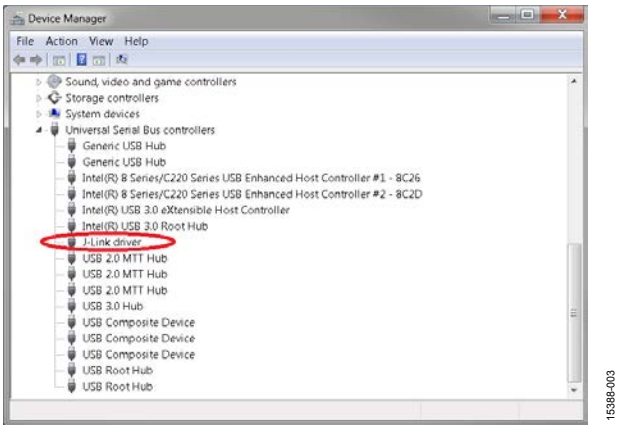


図 4. デバイス・マネージャ

ソフトウェアのインストール手順の後に、J-Link の USB ドライバがインストールされ、検証されます。

IAR ツールのインストール

IAR Embedded Workbench と内蔵の IAR C/C++ コンパイラは、ARM ベースのアプリケーションで業界最速の性能を発揮し、最もコンパクトなコードを生成します。そのため、アナログ・デバイセズは、ADuCM3027/ADuCM3029 向けに IAR ワークベンチの BSP ドライバを作成しました。

KickStart エディションは無料のスターター・キットで、IAR の評価版です。このエディションでは、コード・サイズ（32 kB）と提供されるサービスやサポートの両方に制限があります。

IAR ソフトウェアは、IAR Web サイトからダウンロードできます。無料のお試し版ソフトウェアをダウンロードします。

IAR のインストールとライセンス詳細を追加する手順については、IAR Web サイトの Installation and Licensing Guide for IAR Embedded Workbench® を参照してください。

BSP のサンプルは、IAR バージョン 7.40.2 向けです。別のバージョンを使用すると、プロジェクトに互換性の問題が発生する可能性があります。

ADuCM3027/ADuCM3029 ボード・サポート・パッケージ（BSP）

ADuCM3027/ADuCM3029 BSP には、ADuCM3027/ADuCM3029 の開発を簡単にする構成ファイル、サポート・ファイル、コンポーネントが含まれています。

BSP の内容は次のとおりです。

- ADuCM3027/ADuCM3029 プロセッサに必要なデバイス・ドライバとサービスのソース・ファイル。
- デバイス・ドライバとサービスの例。
- ツール・チェーンをサポート。これらのコンポーネントは、IAR 組み込みワークスペースにインストールされ、ADuCM3027/ADuCM3029 を認識するようにツール・チェーンを構成します。
- 技術文書。

IAR Embedded Workbench をインストールしてから BSP をインストールする必要があります。

BSP をインストールするには、次の手順に従います。

1. ADuCM3027/ADuCM3029 BSP インストーラを http://www.analog.com/jp/design-center/evaluation-hardware-and-software/evaluation-boards-kits/eval-aducm3029-ezkit.html#eb-relatedsoftware でダウンロードします。
2. インストーラ・アプリケーションを実行します。
3. 指示に従ってインストールを完了します。
4. インストールするツール・チェーンがサポートする IAR Workbench のバージョンを選択します（図 5 を参照）。

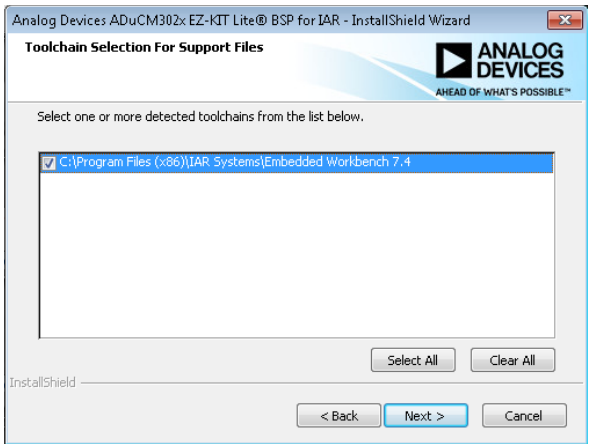


図 5. ツールチェーンの選択ダイアログ・ボックス

5. BSP のドライバとソース・ファイルのロケーションを選択します（図 6 を参照）。Windows のアクセス制限があるので、ロケーションは **Program Files** フォルダ以外に設定することをお勧めします。

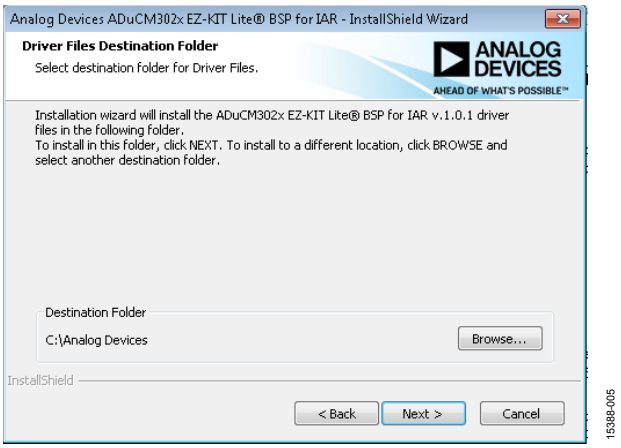


図 6. BSP インストール先フォルダの選択ダイアログ・ボックス

6. インストール・プロセスを完了します。詳細な情報とガイダンスについては、BSP のインストール・フォルダに含まれるファイルと技術文書を確認できます。

IAR 構成

ここでは、ADuCM3027/ADuCM3029 を適切に動作させるための IAR 構成手順について説明します。デフォルト値から変更する必要があるセクションのみを説明しています。

1. 全般オプションで、アナログ・デバイセズの ADuCM3029/ADuCM3027 デバイス・オプションがターゲットとして選択されていることを確認します。
2. **[C/C++ Compiler] > [Optimizations]** で、アプリケーションのニーズに合わせて、速度、コード・サイズ、バランスに最適な各オプションを選択します。コンパイラは、レジスタへの書込みを最適化するのが望ましいと判断する場合があります。ただし、最適化を実行すると、予期せぬ動作が発生する場合があります。このような状況では、次のコードを使用して、構成機能を最適化から保護することを推奨します。

```
#pragma optimize=none
```

3. **[C/C++ Compiler] > [Preprocessor]** で、実行するコードに合わせてディレクトリを含むパスを指定します。

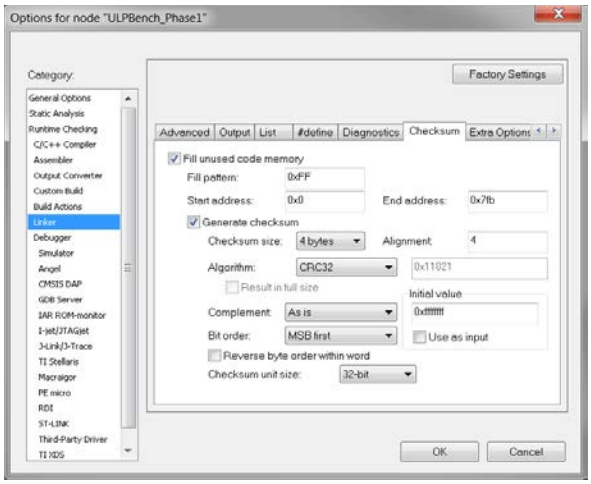


図 7. ［Linker］>［Checksum］設定タブ

4. シグネチャ・フィールドに保存された 32 ビットの CRC チェックサムを使用すれば、ユーザー・コードからユーザー・スペースの整合性チェックを要求できます。そのため、図 7 と図 8 に示すようにチェックサムを構成する必要があります。どちらも［Linker］メニューの構成です。

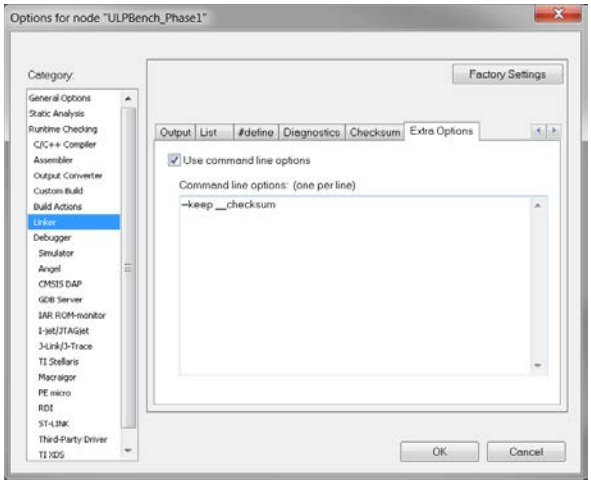


図 8. ［Linker］>［Extra Options］設定タブ

5. ［Driver］ドロップダウン・メニューで、［J-Link/J-Trace］をデバッガとして選択します。［Debugger］>［Download］メニューで、［Verify download］ と ［Use flash loader(s)］の両方のボックスがオンになっていることを確認します(図 9 と図 10 を参照）。

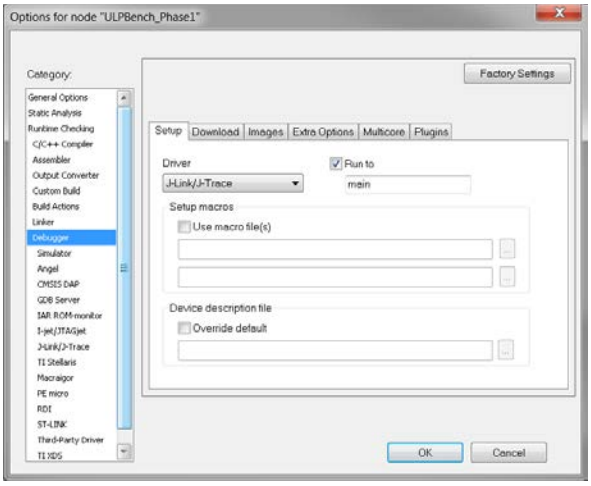


図 9. ［Debugger］>［Setup］タブ

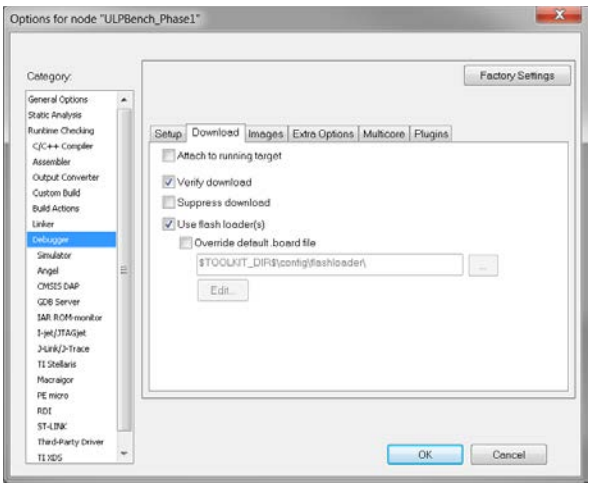


図 10. ［Debugger］>［Download］タブ

6. 図 11 と図 12 に、J-Link/J-trace の設定を示します。［**Reset**］ドロップダウン・メニューで、［**Halt after bootloader**］ターゲットを使用して方策をリセットしてください。そうしないと、カーネルが破損し、デバイスがロックダウンされます。この破損から回復するには、カーネルを再フラッシュする必要があります。

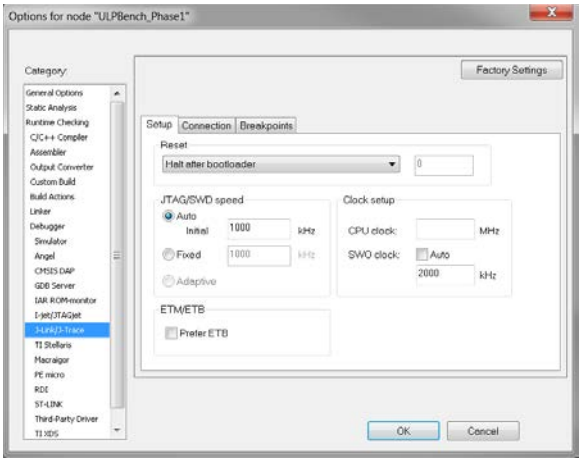


図 11. ［**J-Link/J-Trace**］>［**Setup**］設定タブ

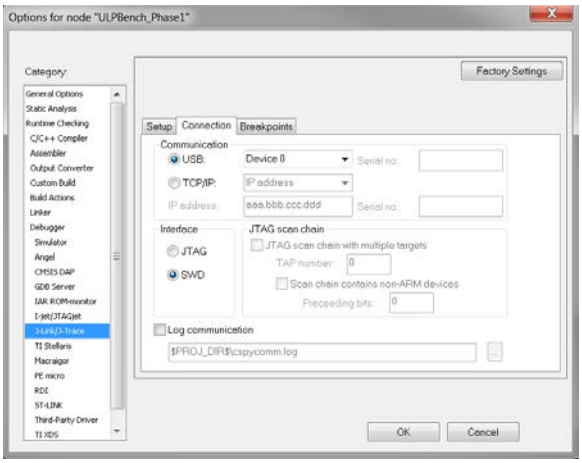


図 12. ［**J-Link/J-Trace**］>［**Connection**］設定タブ

デモ・プロジェクトの構築

IAR 提供のコード

ADuCM3027/ADuCM3029 BSP ディレクトリの **examples** フォルダには、いくつかのサンプル・プロジェクトがあります。**BSP インストール・フォルダ > ¥ADuCM302x_EZ_Kit_Lite > examples** サンプル・プロジェクトを開くには、［**File**］メニューから、［**Open**］>［**Workspace…**］を選択してワークスペース・ファイルに移動します。

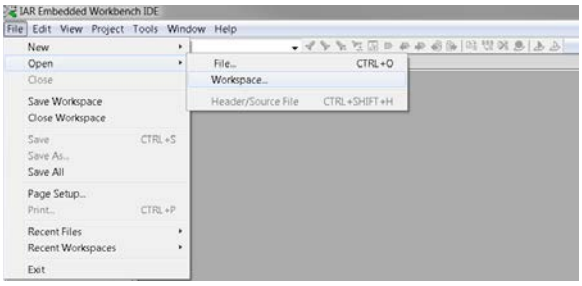


図 13. IAR ワークスペースを開く

BSP の **examples** フォルダには、関連するプロジェクトがあります（図 14 を参照）。

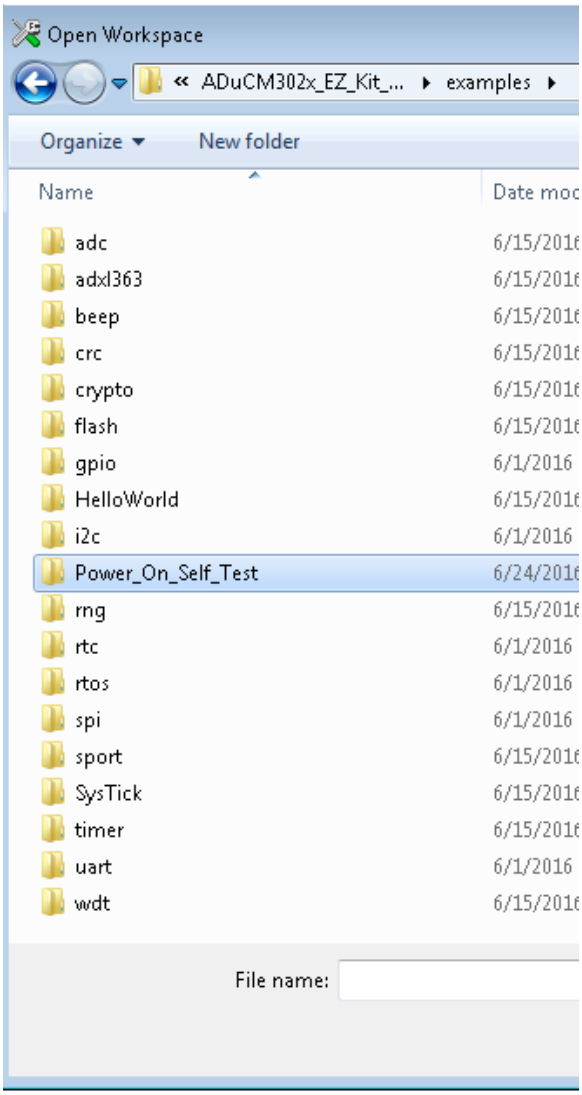


図 14. BSP フォルダ内にあるプロジェクト

各サンプルには、ADuCM3027/ADuCM3029 のペリフェラルへの接続に使用できる充実した下位レベルのペリフェラル・ライブラリがあります。ライブラリとサンプルのわかりやすい技術文書が含まれています。

プロジェクトの変更

プロジェクトを変更するには、ワークスペースで各プロジェクトをクリックし、表示されるメニューから［**Set as Active**］をクリックします（図 15 を参照）。

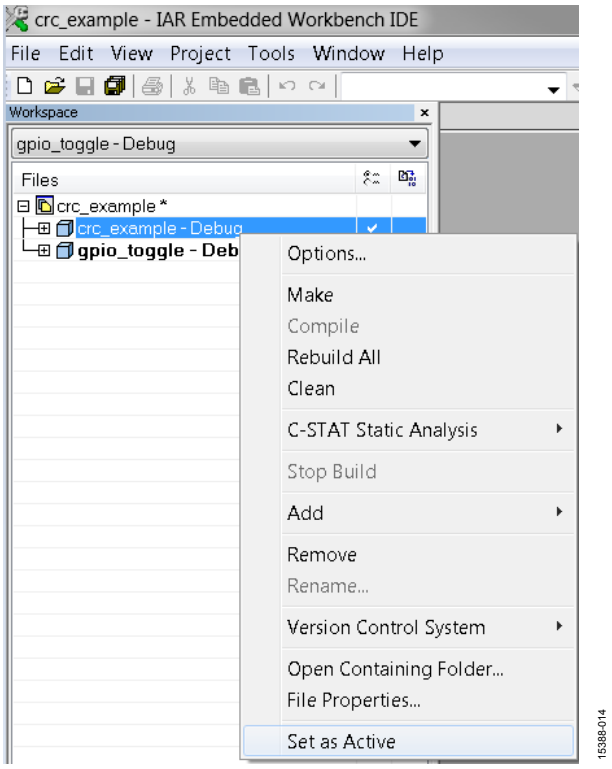


図 15. プロジェクトの変更

アプリケーションのビルド

アプリケーション・コードをビルドまたはコンパイルするには、［**Project**］＞［**Make**］をクリックする（または F7 を押す）（図 16 を参照）か、Make のツールバー・ボタン（図 17 を参照）をクリックします。

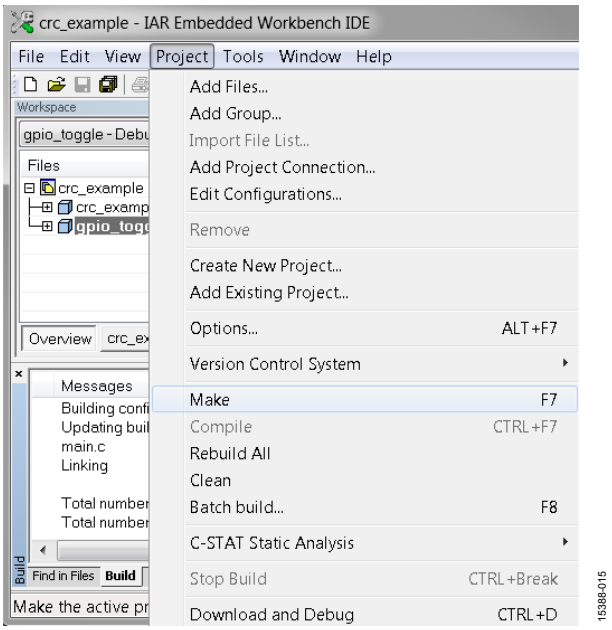


図 16. ［**Project**］＞［**Make**］を使用したアプリケーションのビルド

図 17. ツールバー上にある IAR Make ボタンを使用したアプリケーションのビルド

アプリケーション・コード全体を再ビルドするには、［**Project**］＞［**Rebuild All**］をクリックします。このアクションでは、すべてのプロジェクト・ファイルのクリーン、再コンパイル、およびリンクが実行されます。

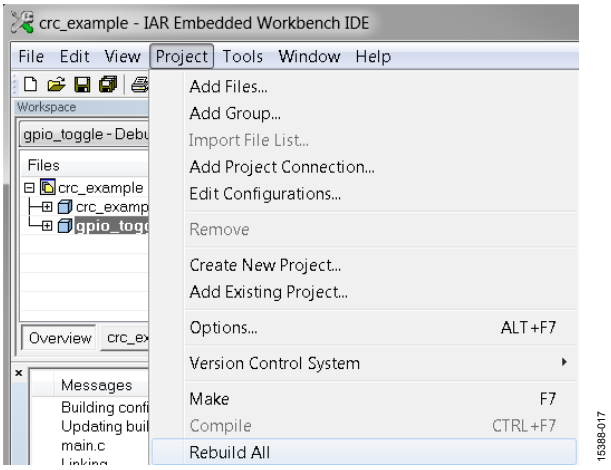


図 18. ［**Rebuild All**］を使用したアプリケーションのビルド

常に、エラーなしでコードをコンパイルできます（図 19 を参照）。

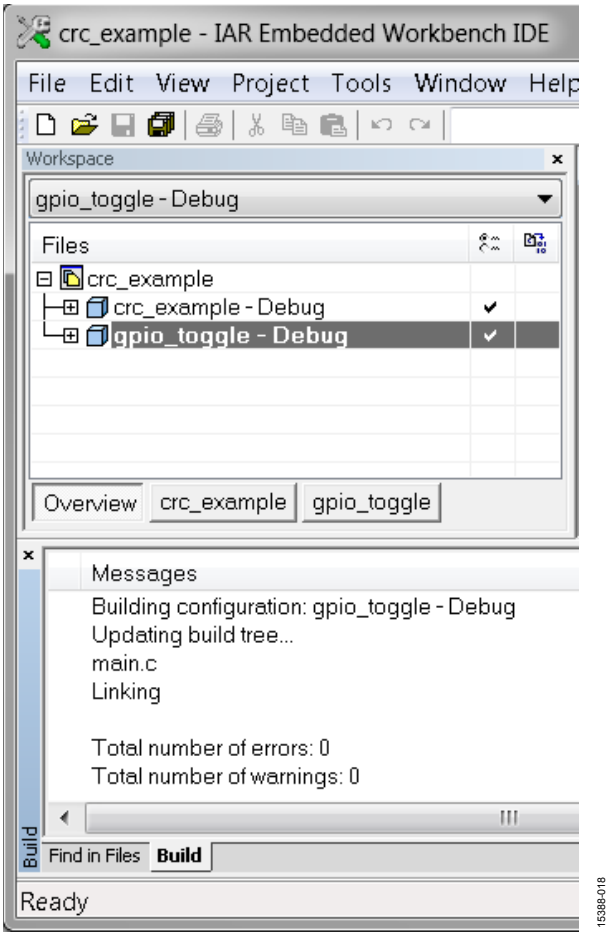


図 19. プロジェクトの再コンパイル

アプリケーション・コードのダウンロード

コードをダウンロードするには、［Project］>［Download］>［Download active application］をクリックします。

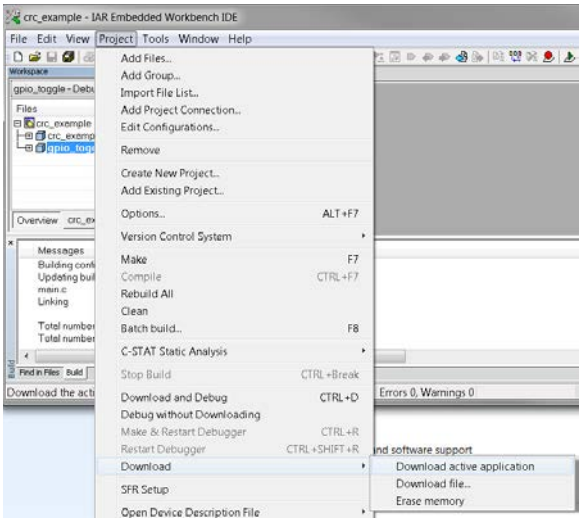


図 20. アプリケーションのダウンロード

デバイスの電源を入れ直したら、ADuCM3027/ADuCM3029 上でコードが実行されます。

アプリケーション・コードのデバッグ

次の手順を使用して、プロジェクトをダウンロードしてデバッグします。

1. デバッグのボタンをクリックします。コードのデバッグは、メイン機能の最初に実行されます。シングル・ステップ、ステップ・オーバー、ブレーク・ポイントのデバッグ機能を使用できます。

図 21. IAR デバッグのツールバー・ボタン

2. ツールバーにある Go アイコンをクリックするか（図 22 を参照）、［Debug］>［Go］を選選するか、F5 キーを押します（図 24 を参照）。ADuCM3027/ADuCM3029 でコードが実行されます。

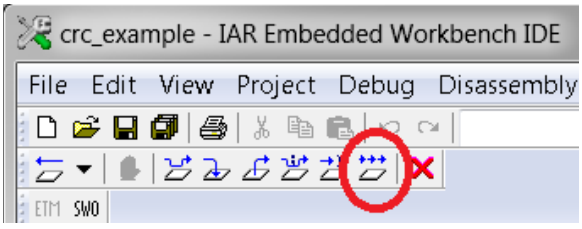


図 22. IAR Go のツールバー・ボタン

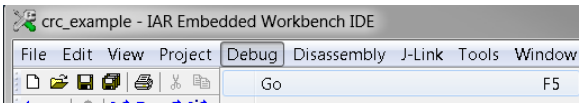


図 23. ［Debug］>［Go］

サンプル・プロジェクトの実行

ADuCM3027/ADuCM3029 BSP には、マイクロコントローラをテストして評価するためのサンプル・プロジェクトが多数あります。このユーザー・ガイドでは、ADuCM3029 EZ-Kit 評価ボードと IAR ワークベンチを使用して、BSP パッケージからサンプル・プロジェクトを実行します。

ここでは、LED_polled_button サンプル・プロジェクトを使用します。この例では、GPIO サービスを使用して、ADuCM3029/ADuCM3027 EZ-Kit ボード上のプッシュ・ボタン（GPIO を入力として設定）と LED（GPIO を出力として設定）を構成します。

LED_polled_button プロジェクトを実行するには、

1. ［Project］>［Add Existing Project…］を選択します。

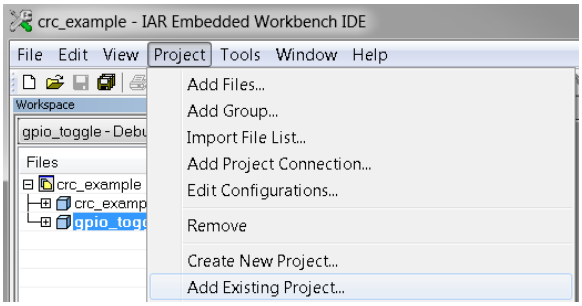


図 24. ［Add Existing Project…］メニュー・オプション

2. ファイル選択ダイアログ・ボックスで、BSP フォルダに移動してから、**examples** フォルダに移動します。**examples** フォルダで、**gpio¥LED_button_polled¥ADuCM3029¥iar** に移動します。**LED_button_polled.ewp** ファイルをクリックしてから、［Open］ボタンをクリックします（図 25 を参照）。

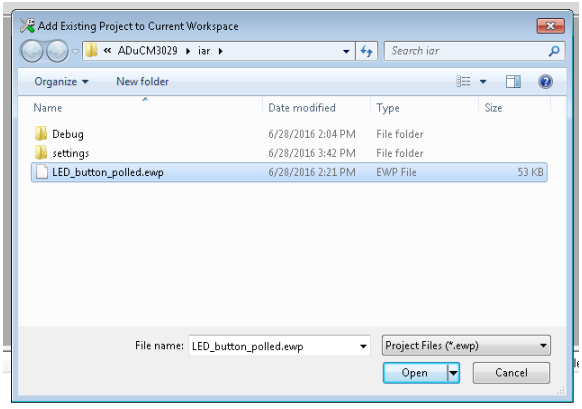


図 25. プロジェクト選択のダイアログ・ボックス

3. ［Project］ドロップダウン・メニューをクリックし、［Make］を選択してプロジェクトをビルドします。ビルド・コンソールには、エラーや警告メッセージが表示されません。

4. アプリケーションをデバッグするには、［**Project**］ドロップダウン・メニューをクリックし、［**Download and Debug**］を選択するか、Ctrl + D を押します（図 26 を参照）。アプリケーションのダウンロード後に、IDE はデバッグ・モードに切り替わり、［**Disassembly**］ウィンドウなどの別のウィンドウ、デバッグ・ツールバー、デバッグ・ロガーなどが表示されます。［**Debug**］>［**Go**］をクリックして、アプリケーションを実行します。

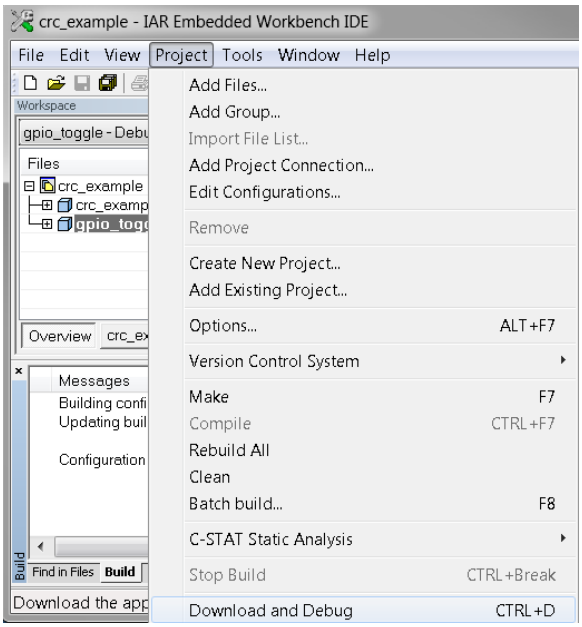


図 26. ダウンロードとデバッグ

ADUCM3027/ADUCM3029 プロセッサの電力最適化

低消費電力 MCU を選択するには、データシートを確認して電気仕様を分析する必要があるので、手間がかかります。多くの場合、該当するシステム・レベルの使用事例にこれらの仕様を関連付けるのは困難です。

ペリフェラルの動作を考慮して実際の使用事例のシナリオをエミュレートしながら、各電力モードを評価するのは、電力要件が厳しいアプリケーションで正しい MCU を選択する場合に欠かせない手順です。低消費電力アプリケーション向けに MCU を選択する場合、次の項目を評価します。

- 低消費電力モードの可用性や、SRAM の内容を保持する機能に与えるこれらのモードの影響
- システムの残りの部分が消費電力モードである場合に、RTC を実行した場合の消費電力
- 低消費電力モードからのウェイクアップ時間
- アプリケーションに必要な電源電圧の範囲。設計者は、部品の要件に合わせて電源電圧を調整して選択できます。
- アクティブ・モードでの消費電力
- コア・アクティビティ - アルゴリズム処理など
- ペリフェラルのアクティビティ - DMA 動作
- コアとペリフェラルの同時アクティビティ
- 消費電力を低く抑えながら、システム条件を満たすようにコアとペリフェラル・クロックの周波数を選択できる柔軟性
- ペリフェラル・アクティビティの最中に、CPU を低消費電力モードに移行できるハードウェア DMA ブロック

ADuCM3027/ADuCM3029 プロセッサは、処理、制御、接続のための超低消費電力の内蔵ミックスド・シグナル MCU システムです。MCU システムは、26 MHz のピーク性能で最大 33 MIPS を実現する ARM® Cortex®-M3 プロセッサをベースとし、一連のデジタル・ペリフェラル、組み込み SRAM とフラッシュ・メモリだけでなく、ADC サブシステムや、クロック、リセット、パワー・マネージメント機能を備えたアナログ・サブシステムを組み合わせています。

ADuCM3027/ADuCM3029 プロセッサは、市場でも数少ない、キャッシュ・コントローラを備えた低消費電力 MCU のうちの 2 つです。同じデータや命令に繰り返しアクセスするプログラムでは、キャッシュ・メモリが効率よく使用されるので、全体の消費電力を削減できます。

MCU の消費電力は、主にシステムの動作電圧と周波数に依存します。ADuCM3027/ADuCM3029 プロセッサは、バッテリ駆動のアプリケーションまたは自己給電（エナジー・ハーベスト）アプリケーションのビルドに便利な、複数の電力モードを内蔵しています。

ここでは、ADuCM3027/ADuCM3029 プロセッサの電力モードについて詳細に説明し、複数のシナリオにおける電力測定の例を示し、開発者が低消費電力アプリケーションの要件に最も適した電力モードを選択できるようにします。

ADUCM3029/ADUCM3027 プロセッサのパワー・マネージメント

ADuCM3029/ADuCM3027 プロセッサは、高度にカスタマイズされたパワー・マネージメントとクロック・システムを内蔵しているので、電力と性能のバランスを柔軟に保つことができます。パワー・マネージメント・ブロックは、多数のアプリケーション・シナリオに適した内蔵レギュレータ、クロック・ゲート方式、スイッチで構成されます。

パワー・マネージメント・システムは、次の機能で構成されます。

- 内蔵の 1.2 V LDO とオプションの容量性降圧レギュレータ
- 休止モードでの低スタンバイ電流向けの内蔵電力スイッチ
- スリープ・モードでリークを減らす電力ゲート
- 電圧範囲を選択できる電源モニタ

ADUCM3029/ADUCM3027 プロセッサの電力モード

パワー・マネージメント・システムには、次の低消費電力モードがあります。

- カスタマイズされたクロック・ゲート機能を備えたアクティブ・モード
- スマート・ペリフェラルを使用した Flexi™ モード。
- オプションの SRAM データ保持機能を備えた休止モード
- SRAM データ保持期間なしのシャットダウン・モード

モードごとに、低消費電力と実行可能な機能の間でトレードオフが存在します。表 2 に、各低消費電力モードのシステム・ブロックのステータスを示します。

表 2. 電力モードのシステム・ブロックのステータス

Functional Block	ARM Cortex-M3 Core	Buck	PERIPHERAL -DMA	HF-XTAL	HFOSC	LFXTAL	PLL	LFOSC	RTC0	RTC1	ADC	SRAM	FLASH
Active Mode	On	User[1]	User[1]	User[1]	User[1]	User[1]	User[1]	On	User[1]	User[1]	User[1]	On	On
Flexi Mode	Off	User[1]	User[1]	User[1]	User[1]	User[1]	User[1]	On	User[1]	User[1]	User[1]	On	On
Hibernate Mode	Off	Off	Off	Off	Off	User	Off	On	User[1]	User[1]	Off	On[2]	Off
Shutdown Mode	Off	Off	Off	Off	Off	User	Off	Off	User[1]	Off	Off	Off	Off

[1] この機能ブロックは、ユーザー・アプリケーション・コードでオンまたはオフに設定できます。
[2] 保持可能な SRAM サイズは設定可能です。

アクティブ・モード

アクティブ・モード（フル・オン・モードとも呼ばれる）では、ARM Cortex-M3 がアクティブで、フラッシュ・メモリや SRAM からの命令を実行します。ユーザーの判断ですべてのペリフェラルをイネーブルまたはディスエーブルにでき、最適化されたクロック管理によってアクティブ・モード電力を向上させることができます。

アクティブ・モードでは、次の電力節約オプションを使用できます。

- 降圧コンバータの使用
- キャッシュの有効化
- ダイナミック・クロック・スケーリング
- クロック・ゲート

降圧コンバータ

オプションの降圧コンバータ機能を使用すれば、アクティブ・モードで電力を節約できます。降圧コンバータは、リニア電圧レギュレータを介してデジタル・コア領域に給電します。バッテリ電圧（VBAT）が 2.3 V を下回ると、降圧コンバータはバイパス・モードに移行します。バイパス・モードに移行した後、降圧コンバータの出力が入力に追従します。図 27 に、降圧コンバータがイネーブルになる設計で推奨される外部回路を示します。

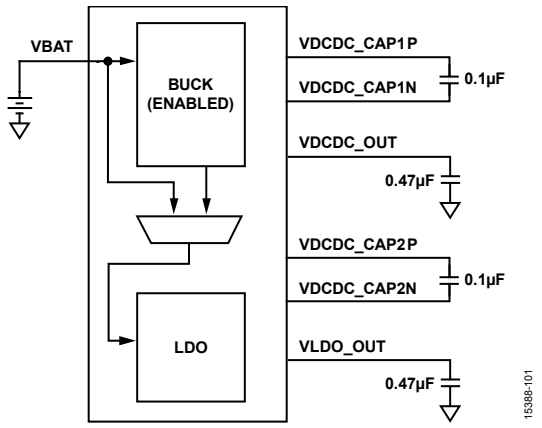


図 27. 降圧コンバータがイネーブルになる設計の外部回路

オプションの降圧コンバータを使用しない設計では、VDCDC_CAP1P、VDCDC_CAP1N、VDCDC_OUT、VDCDC_CAP2P、VDCDC_CAP2N ピンを未接続のままにしてください。

降圧コンバータを使用するのは、プロセッサのみです。降圧コンバータの出力には、外部負荷を接続できません。

次のコードで CTL1.HPBUCKEN ビットを設定して、降圧コンバータをイネーブルにします。

```
*pREG_PMG0_CTL1 |= (1<<ITP_PMG_CTL1_HPBUCKEN);
```

図 28 では、次の条件で素数を計算する場合の ADuCM3027/ADuCM3029 プロセッサの消費電力を比較します。

- VBAT = 3.0 V
- HCLK = PCLK = 26 MHz
- キャッシュ・メモリはディスエーブル

降圧コンバータは、高い VBAT 値で消費電流を節約します。具体的には、VBAT ≥ 3 V の場合に有効電流が約 50 % 減少します。

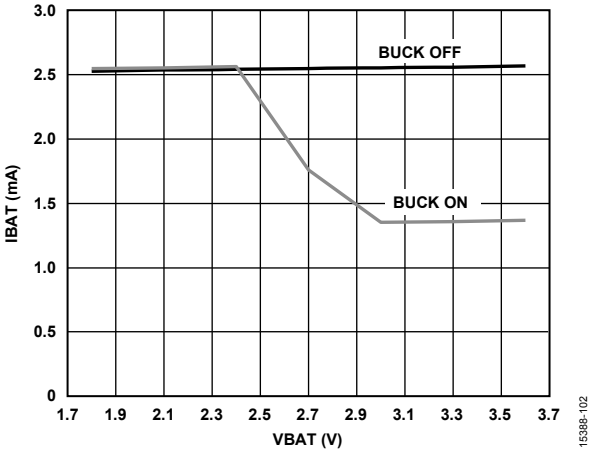


図 28. 降圧コンバータがアクティブ・モード消費電力（キャッシュ・メモリはイネーブル）に与える影響

キャッシュ・メモリを使用すれば、フラッシュ・メモリからデータにアクセスするための平均時間が短くなります。CPU がアルゴリズムを実行する必要がある場合または同じデータに繰り返しアクセスする必要がある場合、キャッシュ可能なメモリを使用すれば、内部命令 SRAM から実行されるので消費電力が減ります。キャッシュ・コントローラがイネーブルになっている場合、4 KB の命令 SRAM はキャッシュ・メモリとして予約されます。

デフォルトでは、キャッシュ・メモリがスタートアップ時にディスエーブルになります。キャッシュ・メモリをイネーブルにするには、次の手順に従います。

1. キャッシュ・イネーブル・ステータス・ビット（CACHESTAT レジスタのビット 0）の読出しを実行し、キャッシュ・メモリがディスエーブルであることを確認します。クリアされるまでこのビットをポーリングします。
2. CACHEKEY レジスタを使用して、ユーザー・キーの書込みを実行します。例えば、次の値を書き込みます。

```
*pREG_FLCC0_CACHE_KEY = 0xF123F456;
```

3. 命令キャッシュ・イネーブル・ビット（CACHESETUP レジスタの ICEN）を次のように設定します。

```
*pREG_FLCC0_CACHE_SETUP |= (1 <<
BITP_FLCC_CACHE_SETUP_ICEN);
```

図 29 では、次の条件で素数を計算する場合の ADuCM3027/ADuCM3029 プロセッサの消費電力を比較します。

- VBAT = 3.0 V
- HCLK = PCLK = 26 MHz
- 降圧コンバータはディスエーブル

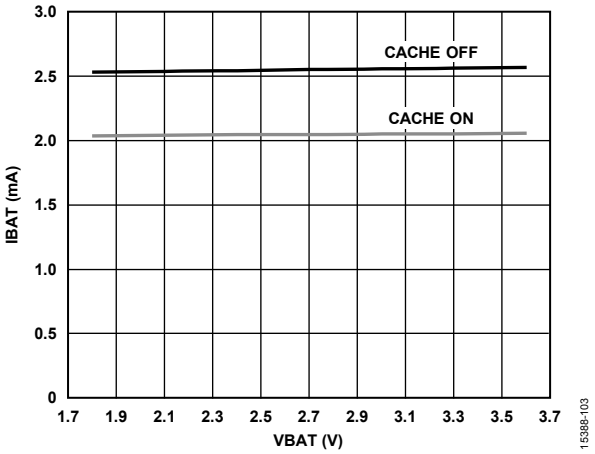


図 29. キャッシュ・メモリがアクティブ・モード消費電力に与える影響

キャッシュ・メモリをイネーブルにすると、有効電力の平均消費量を最大で 18 % 減らすことができます。

ダイナミック・クロック・スケーリング

動的なクロック／周波数のスケーリングは、消費電力を削減するうえで実績のある手法です。ADuCM3027/ADuCM3029 プロセッサには、CPU とペリフェラル・クロックの周波数を動的に変更できる、柔軟なクロック・アーキテクチャが組み込まれています。クロック分周器とフェーズ・ロック・ループ（PLL）を組み合わせることで、固定のクロック方式よりも消費電力を低く維持したまま、最適なシステム・クロック周波数を柔軟に導出できます。プログラマブルなクロック分周器でシステム内のクロックを生成し、分周器を即座に構成できます。

図 30 では、次の条件で素数を計算する場合の ADuCM3027/ADuCM3029 プロセッサの消費電力をプロットしています。

- VBAT = 3.0 V
- HCLK = PCLK（ルート・クロックの源は HFOSC)
- 降圧コンバータはディスエーブル
- キャッシュはディスエーブル

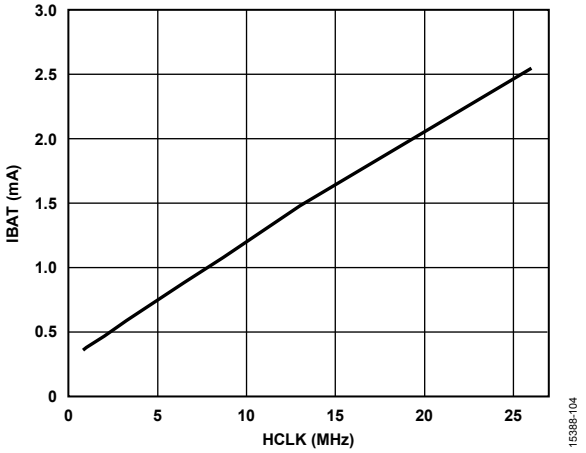


図 30. コア・クロック周波数がアクティブ・モード消費電力に与える影響

コア・クロックの周波数が減少すると、消費電力も減少します。

クロック・ゲート

システムは、高度なクロック・ゲーティングと自動クロック・ゲート技術を使用します。大部分のペリフェラルは、ディスエーブルの場合に自動的にクロック・ゲーティングされ、イネーブルな場合のみクロックが実行されます。例外は、I^2C、GPIO、汎用タイマー（GPTMR）です。これらのブロックは、CLKCON5 レジスタを使用して手動でクロック・ゲーティングする必要があります。CLKCON5.PERCLKOFF ビットをセットして、ペリフェラル・クロックを完全にゲーティングします。

クロック・ゲーティングされたペリフェラルへのアクセスは、CLKCON5 レジスタのクロック・ゲート設定をオーバーライドします。

コアがデータを処理し、ペリフェラルのアクティビティが望ましくないアプリケーション・シナリオでは、ペリフェラル・クロック（PCLK）をオフにして電力を節約する必要があります。図 31 に、次の条件で素数を計算する場合の ADuCM3027/ADuCM3029 プロセッサの消費電力を示します。

- VBAT = 3.0 V
- 降圧コンバータはディスエーブル
- キャッシュ・メモリはディスエーブル
- HCLK = PCLK = 26 MHz

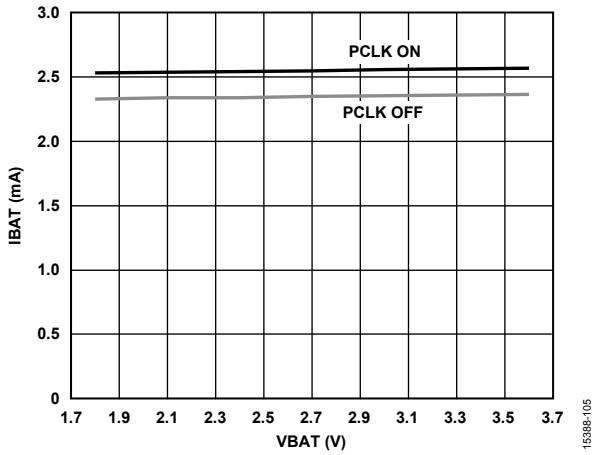


図 31. ペリフェラル・クロック・ゲートがアクティブ・モードの消費電力に与える影響

図 31 に示すように、ペリフェラル・クロックがゲーティングされると、有効電流が 0.2 mA ほど減少します。

アクティブ・モードでは、ここで説明される 4 つの技術を組み合わせて、電力を最大限に節約できます。

Flexi モード

Flexi モードは柔軟なスリープ・モードで、コアが処理を開始する前に、ペリフェラルのデータ転送を待機する必要がある場合に便利です。Flexi モードでは、コアがクロック・ゲーティングされ、システムの残りの部分はアクティブになります。Flexi モードを使用すると、プロセッサがウェイクアップしてデータを処理する前に、非常に低速のアクティビティを完了することが予想される場合（センサーから特定のバイト数の読出しを実行するなど）、大幅に有効電力を削減できます。

CPU が SPI DMA を構成し、DMA の完了を待機する必要があるシナリオを考えます。図 32 に、次の条件で DMA にアクセスして SPI 経由でデータを転送する ADuCM3027/ADuCM3029 プロセッサの消費電力を示します。

- VBAT = 3.0 V
- 降圧コンバータはディスエーブル
- キャッシュはディスエーブル
- ODR = 6.5 MHz
- SPI_DIV = 49

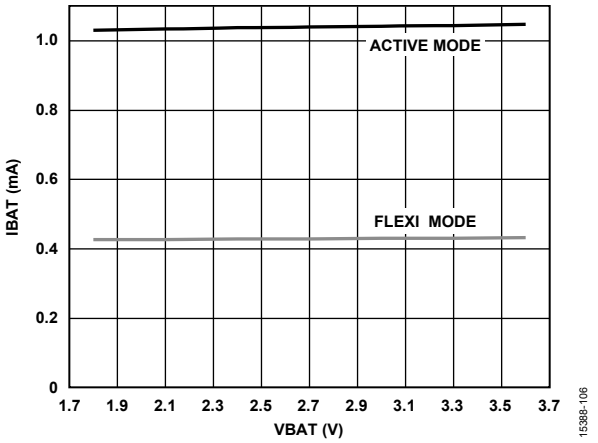


図 32. Flexi モードが消費電力に与える影響

図 32 に示すように、DMA の実行中にコアをアクティブ・モードに維持せずに Flexi モードを使用すると、電力を約 66 % 節約できます。

Flexi モードを終了するには、多数のウェイクアップ・ソースを使用できます（DMA 割込み、外部割込み、タイマー割込みなど）。通常は、1 つの CPU クロック・サイクルのみで終了します。

Flexi モードで降圧コンバータもイネーブルにすると、さらに電力を節約できます。図 33 に、次の条件で DMA にアクセスして SPI 経由でデータを転送しながら、Flexi モードで降圧コンバータを使用する VBAT 全体の ADuCM3027/ADuCM3029 プロセッサの消費電力を示します。

- VBAT = 3.0 V
- キャッシュ・メモリはディスエーブル
- ODR = 6.5 MHz
- SPI_DIV = 49

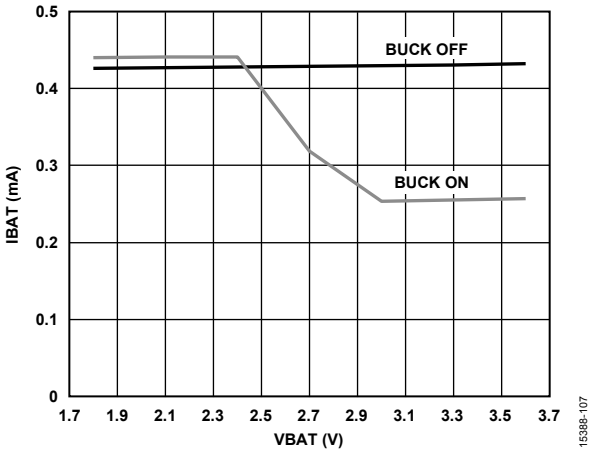


図 33. 降圧コンバータが Flexi モードの消費電力に与える影響

図 33 に示す、アクティブ・モードでの降圧コンバータによる影響と同様な電力の向上パターンが図 28 でも見られます。具体的には、VBAT ≥ 3 V の場合に、50 % の電力の向上が観察されています。

休止モード

休止モードでは、ARM Cortex-M3 コアとすべてのデジタル・ペリフェラルはオフになり、設定可能な SRAM データ保持期間、ポート・ピンのデータ保持期間、限られた数のウェイクアップ中断、オプションでアクティブな RTC を使用できます。休止モードでは、すべての GPIO ピンの状態が保持されます。ADuCM3027/ADuCM3029 プロセッサは、RTC ブロックに SensorStrobe™ 機構も組み込んでいます。これにより、超低消費電力センサーのデータ計測が可能になります。

休止モードに移行する前に、特定のシーケンスを実行して休止モードに正しく切り替わるように、イネーブルなペリフェラルの大部分をプログラムする必要があります。休止モードの間、複数のシステム・メモリ・マップ・レジスタ（MMR）とペリフェラル・レジスタは保持されます。詳細については、ADuCM302x Ultra Low Power ARM Cortex-M3 MCU with Integrated Power Management Hardware Reference に記載された関連するペリフェラル情報を参照してください。

設定可能な保持可能 SRAM

ADuCM3027/ADuCM3029 プロセッサでは、休止モードで保持する SRAM ブロック・サイズに 8 KB（デフォルト）、16 KB、24 KB、32 KB を使用できます。保持する必要がある SRAM が多くなるほど、休止モードでの消費電力は高くなります（図 34 参照）。

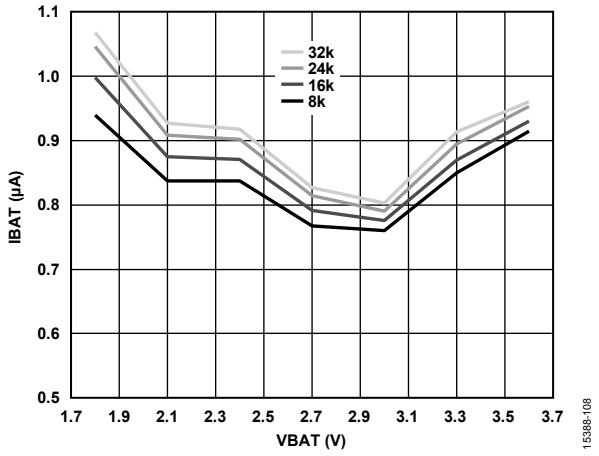


図 34. 保持される SRAM 値に対する VBAT 電源ピン（I_{BAT}）での電流

SRAMRET レジスタで適切なビットをセットすることで、SRAM の保持サイズを設定できます。例えば、休止モード中に 32 KB の SRAM をイネーブルにするには、SRAMRET レジスタで次のコードを使用します。

```
*pREG_PMG_PWRKEY = 0x4859;
*pREG_PMG_SRAMRET |=
((1 << BITP_PMG_SRAMRET_SRAM_RET1_EN)|
(1 << BITP_PMG_SRAMRET_SRAM_RET2_EN));
```

パリティをイネーブルにする場合、休止モードからのウェイクアップで保持されない SRAM 領域の初期化が必要になることがあります。

ウェイクアップ・ソース

次のイベントが発生すると、休止モードからデバイスをウェイクアップできます。

- 外部割込み 0 ～外部割込み 3
- RTC0 と RTC1 の割込み
- バッテリ電圧範囲の割込み
- UART Rx ピン・アクティビティ

2 つのリアルタイム・クロックのうち、推奨される休止モードからのウェイクアップ・ソースは RTC1 のみです。休止モードとシャットダウン・モードの両方を使用する必要があるアプリケーションでは、シャットダウン・モードの終了に使用できるのは RTC0 のみであることが理由です。

これらのイベントにおいて、休止モードからのウェイクアップ時間は、フラッシュから実行する場合は約 14 µs、SRAM から実行する場合は約 10 µs です。

RTC クロック源

ADuCM3027/ADuCM3029 プロセッサでは、RTC1 ブロックの選択肢が 2 つあります。

- 低消費電力の内部 RC 発振器（LFOSC）
- 外部水晶発振器（LFXTAL）

LFOSC または LFXTAL の実装では、精度と消費電力の間にトレードオフがあります。LFXTAL は LFOSC よりも正確ですが（水晶発振器の製造業者によって異なる）、FOSC のほうが消費電力は低くなります（図 35 を参照）。

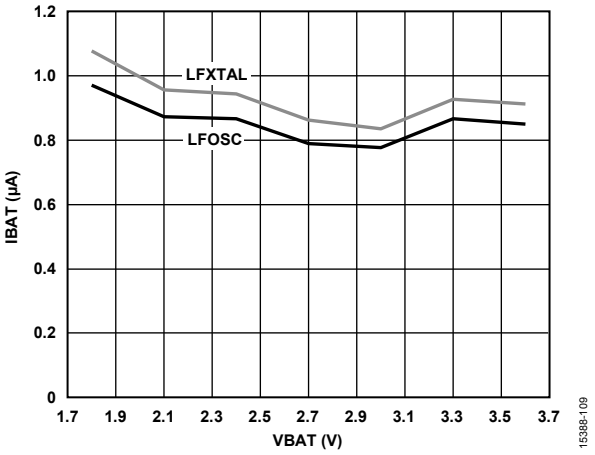


図 35.RTC1 をウェイクアップ・ソースに使用した場合の休止電流（LFOSC 対 LFXTAL）

SensorStrobe

SensorStrobe 機構を使用すれば、休止モードを含む一部の電力モードで ADuCM3027/ADuCM3029 プロセッサをプログラマブル・クロック・ジェネレータとして使用できます。この方法では、ADuCM3027/ADuCM3029 プロセッサによって管理されるタイミング領域を外部センサーで設定できます。SensorStrobe の出力は、FLEX_RTC からのプログラマブル分周期器の値で、FLEX_RTC は最大 30.7 µs の分解能で動作します。センサーとマイクロコントローラは同期するので、マイクロコントローラとセンサーでデータを再サンプリングして時間を合わせる必要はありません。

シャットダウン・モード

シャットダウン・モードは強力なディープ・スリープ・モードで、デジタル回路とアナログ回路の電源がダウンします。デジタル・コアと SRAM メモリの内容は保持されません。ただし、ウェイクアップ割込み構成であるパッドの状態は保存されます。

パッドの構成は、シャットダウン・モードからのウェイクアップ後に保存およびロックされます。PGM_TST_CLR_LATCH_GPIOS レジスタへの 0x58FA の書込みを実行し、パットの状態をロック解除する必要があります（ISR ルーチン内が望ましい）。

```
*pREG_PMG0_TST_CLR_LATCH_GPIOS = 0x58FA;
```

さらに、次のオプションから選択して、適切なウェイクアップ・ソースを構成する必要があります。

- 外部割込み 0 ～外部割込み 2
- 外部リセット
- バッテリが 1.6 V を下回る
- RTC0 タイマー

RTC0 ブロックはこのモードでイネーブルにでき（オプション）、RTC0 割込みでプロセッサを定期的にウェイクアップできます。

シャットダウン・モードでは LFOSC がディスエーブルになるので、RTC0 のクロック源を LFXTAL にする必要があります。

RTC0 ブロックがウェイクアップ・ソースとして機能するには電力が必要なので、シャットダウン・モード中の消費電力が増加します（図 36 を参照）。

デバイスがシャットダウン・モードからウェイクアップすると、POR シーケンスに移行し、コードの先頭から実行が開始されます。

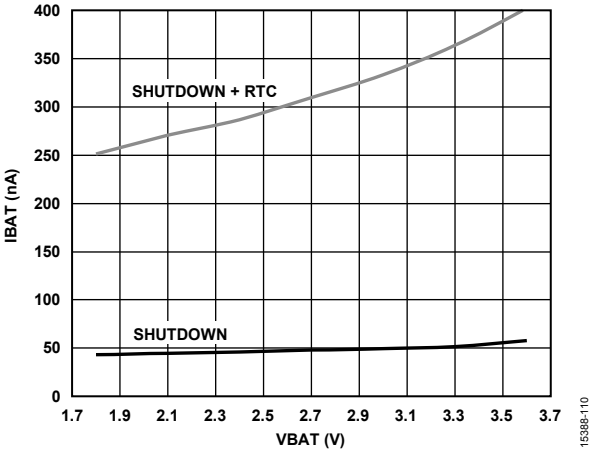


図 36. シャットダウン・モード電流（外部ソース対 RTC0）

ADUCM3029/ADUCM3027 プロセッサ・ブート・カーネルの使用

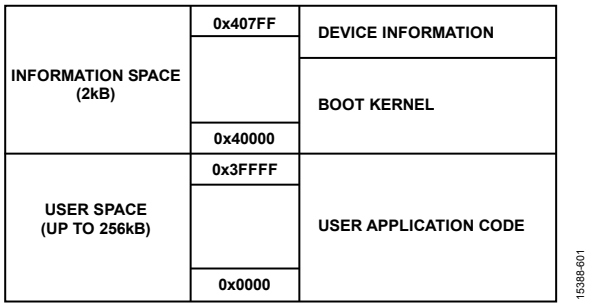


図 37. フラッシュ情報メモリ・スペース

ADuCM3029/ADuCM3027 プロセッサには、ユーザー・アプリケーション・コード（ユーザー・スペース）を含む内蔵フラッシュ・メモリと 2 KB の専用メモリ・バンク、構成された情報スペースがあります（図 37 を参照）。

デバイスによっては、128 KB のユーザー・スペースがあります。詳細については、ADuCM3027/ADuCM3029 データシートを参照してください。

図 37 に示すように、情報スペース・ブロックは、さらにフラッシュ・メモリの上位 2 KB のブート・カーネルとデバイス情報に分割されます。ブート・カーネルは、セキュリティ保護された環境を実装しています。また、オプションでユーザー・アプリケーション・コードの読出し／書込みが保護され、リセット時にフラッシュ・メモリからアプリケーションを実行できます。また、ブート・カーネルは UART ダウンローダからファームウェアをアップグレードする機構も備えています。

ここでは、オンチップ・フラッシュ・メモリの情報スペース領域だけでなく、ブート・プロセスと UART ダウンローダを使用してプロセッサ・ファームウェアへのフィールド・アップグレードを実行する方法について説明します。

ブート・カーネルの概要

ブート・カーネルは、特定のチェック（ユーザー・アプリケーションの CRC 整合性を含む）の実行後、ユーザー・アプリケーションに移行したり、SYS_BMODE0 ブート・ピン・モードによっては、リセット時に UART ダウンローダ・モードに移行し、フラッシュ・メモリでユーザー・アプリケーションをアップグレードします。

ブート・カーネルは、UART ポート経由でユーザー・アプリケーションへのフィールド内アップデートをサポートしています。セキュリティ上の理由から、ブート・カーネル自体はフラッシュ・プログラムの機能を備えていません。ただし、ファームウェア・アップデート・コードには、ユーザー・フラッシュをアップデートするためのフラッシュ・ドライバ・コードがあるので、UART ポートからデバイスにダウンロードできます。このコードは SSL（Second Stage Loader）と呼ばれ、SRAM から実行できます。SSL は、実行アクセスを提供する前に認証する必要があります。次のセクションでは、セキュリティ・スキームの実装について説明します。ユーザー・コードを読出し／書込み保護できる知的財産のセキュリティを実現し、セキュリティ環境を提供するカーネルの重要な部分について説明します。

シリアル・ダウンロード機能を使用すれば、ターゲット・システムに直接ハンダ処理されたデバイスを再プログラムできるので、外部のデバイス・プログラマが不要になり、システムからデバイスをスワップする必要がなくなります。SYS_BMODE0 ピンと UART ポートに関連するハードウェア・インフラストラクチャがターゲット・ボードに実装されている場合は、シリアル・ダウンロード機能を使用することでシステム・アップグレードをフィールドで実行できます。

セキュリティ・オプションの設定

ブート・カーネルでは、デバイスのセキュリティ・オプションを柔軟に設定でき、ユーザー・フラッシュ・メモリの 0 ページで事前に定義されたロケーションにある特定のキーとパラメータをプログラムできます。このカーネルは、ユーザー・コードのセキュリティと整合性を提供します。これは、ユーザー・フラッシュ・メモリの最初のページにあるユーザー定義のパラメータによって決定されます。表 3 に、キーとパラメータのリストに加え、ユーザー・フラッシュ・メモリのロケーションを示します。

表 3. キーとパラメータのリスト

Address Range	Size	Description
0x0000_0180 to 0x0000_018F	128 bits (16 bytes)	Read protection key hash
0x0000_0190 to 0x0000_0193	32 bits (4 bytes)	CRC of read protection key hash
0x0000_0194 to 0x0000_0197	32 bits (4 bytes)	Length of user boot loader or entire user code (used for CRC verification before boot)
0x0000_0198 to 0x0000_019B	32-bit word	In-circuit write protection if set to NOWR
0x0000_019C to 0x0000_019F	32-bit word	CPU write protection of individual flash blocks

保護キー・ハッシュの読出し

ユーザー・フラッシュ・メモリの最初のページにあるアドレス 0x00000180 で 128 ビットの読出し保護キー・ハッシュをプログラムします。このハッシュの値は、システム内で希望するセキュリティの種類によって異なります。このセキュリティは、デバイスへの読出しアクセスを定義するからです。キー・ハッシュは、シリアル・ワイヤ・デバッガ（SWD）の状態だけでなく、UART 経由でアップグレードするためにダウンロードされる SSL のアクセス権限を定義します。

- すべてのロジック・ハイ・メモリ・セルのフラッシュ・メモリ（有効なキー・ハッシュ CRC）がリセット状態の場合は、ユーザーが読出し保護を必要としないことを示します。この場合、SWD インタフェースは、ブート中に自動的にイネーブルになります。
- リセット以外の値は SWD をロックするので、SWD はデバイスにアクセスできません。
- キー・ハッシュは 128 ビットの切り詰められたユーザー・キー（128 ビット長）のセキュア・ハッシュ・アルゴリズム（SHA-256）で、UART ダウンロード・フェーズ中に SSL と一緒に送信できます。ユーザー・キーが有効で、受信したキーのハッシュが保存されたキー・ハッシュに一致する場合、SSL はすべての権限で実行されます。ユーザー・キーがキー・ハッシュ・チェックに失敗した場合、SSL のみがユーザー・フラッシュへの書込み権限を持ちます。

キー・ハッシュ CRC

キー・ハッシュには、アドレス 0x00000190 に保存された 32 ビットの CRC チェックサムがあります。関連付けられた 4 バイトのチェックサムが有効な場合のみ、キー・ハッシュは有効です。キー・ハッシュには、フラッシュの改ざん攻撃から保護する別のキー・ハッシュがあります。実用上の理由から、キー・ハッシュの有効な CRC をキー・ハッシュ自体と一緒に保存する必要があります。

回路内書込み保護キー

ユーザー・フラッシュ・メモリのアドレス 0x00000198 にある 32 ビット回路内書込み保護キーは、デバイスの回路内プログラミングを防止します。回路内再プログラミングを無効にするには、NoWr ASCII 文字列（null 文字で終了しない）の 16 進数をこのアドレスにプログラムします。この場合、デバイスへの SWD アクセスはロックされるので、UART ダウンローダを経由するのがデバイス・コードをアップデートする唯一の方法です。

回路内書込み保護と読出し保護（読出しと書込み両方の保護を提供）を使用します。回路内書込み保護だけでは、意味がありません。

書込み保護

コードが誤って重要なフラッシュ・メモリ・ブロック（ユーザー・コード・ブート・ローダなど）を消去したり、再プログラミングしたりすることを防ぐため、ページをロックできます。フラッシュ・コントローラには、ブロックにグループ化されたページのプログラムを無効にするハードウェア・レジスタがあります。このレジスタは、ハードウェアから自動的にロードされるのではなく、カーネルから書き込まれます。カーネルが、ユーザー・フラッシュ・アドレス 0x0000019C から書込み保護ワードを読み出し、フラッシュ・コントローラの書込み保護レジスタに書き込みます。誤った書込みから保護されるページに応じて、このロケーションに対して適切なワードの書出しを実行できます。このページは 4 つのグループで保護され、32 ビット・ワード内の各ビットは 4 つの連続したフラッシュ・ページに対応します。詳細については、ADuCM302x Ultra Low Power ARM Cortex-M3 MCU with Integrated Power Management Hardware Reference Manual を参照してください。

ユーザー・コード長

フラッシュ・メモリ・アドレス 0x00000194 に 32 ビットの値が保存されていますが、これは CRC 保護されたユーザー・コード長を定義します。このフィールドでプログラムされた値は、CRC 保護が望まれるユーザー・フラッシュ・メモリのページ数を定義します。フラッシュ・メモリの 0 ページ～ N ページまで CRC 保護が必要になり、合計 N + 1 ページが保護されます。

アドレス 0x00000194 のフィールドでの有効な値は、256 KB ADuCM3029 プロセッサで 0 ～ 127、128 KB ADuCM3027 プロセッサで 0 ～ 63 です。この範囲外の値は無効と処理され、CRC チェックは失敗します。

ユーザー・コード CRC

ユーザー・コード CRC は、N ページの末尾に保存されます。32 ビット CRC（MSB ファースト）と 0x4C11DB7 の多項式がカーネルで生成されます。ページ番号が N の場合、CRC はフラッシュ・メモリ・アドレス（N × 0x800）+ 0x7FC に配置されます。例えば、N = 5 の場合は合計 6 ページの CRC が保護され、CRC はアドレス 0x2FFC に保存されます。予想される CRC ロケーションに 0xFFFFFFFF をプログラミングして CRC チェックを無効にするオプションがあります。カーネルが CRC ロケーションでこの値を発見すると、CRC チェックをスキップします。

ブート・コード・フロー

ここでは、前に説明したユーザー・プログラマブル・パラメータに基づいて、カーネルが動作する方法について説明します。図 38 に、ブート・コードのフローチャートを示します。

リセット後、ブート・カーネルは、ユーザー・フラッシュ・メモリの 0 ページに保存されたすべてのパラメータを検査します。ユーザー読出し保護キー・ハッシュがプログラムされず（すべてが FF に設定される）、キー・ハッシュ CRC が有効な場合、ユーザーは読出し保護を要求していません。SWD 自体は有効でも、ユーザー・コード CRC の状態によってフラッシュの読書きが保護される場合があります。

CRC が有効な場合、ユーザー・フラッシュ・メモリへの読書きは制限されません。

0xFFFFFFFF を CRC ロケーションにプログラムして CRC を無効にすると、ユーザ・フラッシュ・メモリへのアクセスも制限されません。

CRC が無効の場合、ユーザー・フラッシュ・メモリに対する読出し／書込みアクセスは許可されず、保護されます。フラッシュ大容量消去コマンドのみが許可されます。この場合、ユーザー・コードの実行は許可されません。

定義された CRC ロケーションで有効な CRC（または 0xFFFFFFFF）をプログラムする場合は慎重に行ってください。そうしないと、ユーザー・フラッシュ・メモリの読み出しはカーネルによって保護されます。この場合、フラッシュ・ベースのアプリケーションは、ユーザー・フラッシュ・メモリが大容量消去されるまでロードできません。

ユーザー読出し保護キー・ハッシュがノンリセット値でプログラムされている場合、つまり、読出し保護が有効であるか、キー・ハッシュ CRC が無効である場合、SWD はカーネルによって無効化され、デバイスへの SWD アクセスは不可能になります。製品開発の完了後にフィールドでの SWD アクセスが不要になった場合のみ、このプログラムを実行します。ただし、読出し保護を有効にすると、CRC 保護が有効で CRC が破損している場合に限り SWD がオープンになるので、CRC が誤って破損した場合にデバイスを修復できます。この場合、SWD はオープンになりますが、ユーザー・フラッシュ・メモリは読出し、書込み、またはページ消去のアクセスなしで保護されます（デバイスの回復は可能ですが、ユーザー・コードの機密性は保持されます）。ただし、大容量消去が引き続き可能なので、ユーザー・フラッシュ・メモリはここでもオープンになります（読出し／書込みアクセス）。

ユーザー・フラッシュ・メモリ（ユーザー・スペース）は、出荷時は完全にブランクです。そのため、セキュリティ・キーとパラメータのいずれもプログラムされていません。そのため、キー・ハッシュ CRC などの大部分のパラメータや、ユーザー・コード長は無効な値になり、パラメータはすべて 0xFF に設定されます。この場合、カーネルはユーザー・フラッシュ・メモリのチェックを実行し、ブランクであるか、完全にプログラムされていないか識別します。ユーザー・フラッシュがブランクの場合、カーネルはすべてのチェックをスキップし、SWD をオープンにします。開発のために SWD を介してデバイスに接続できるよう SWD をオープンにする他、ブート・カーネルは UART ダウンローダ・モードに移行して、SSL の受信を待機します。

デバイスが SWD 経由でプログラムされると、ユーザ・フラッシュ・メモリはブランクでなくなり、カーネルは SYS_BMODE0 ピンの状態を利用してユーザー・コードを実行するべきか（前に説明したすべてのチェックの実行後）、またはブート・カーネルは UART ダウンローダー・モードに移行する必要があるか判断します。

SYS_BMODE0 ピンがアサートされている（ロー）の場合、カーネルは UART ダウンローダに移行し、SSL のダウンロードを待機します。

ISYS_BMODE0 ピンがアサート解除されている（ハイ）の場合、カーネルはすべてのセキュリティ・チェックを実行した後にユーザー・フラッシュ・メモリにあるユーザー・リセット・ベクトルにジャンプします。

SYS_BMODE0 ピンのサンプリングなしでカーネルが UART ダウンロード・モードに移行するのは、ユーザー・フラッシュ・メモリがブランクの場合のみです。

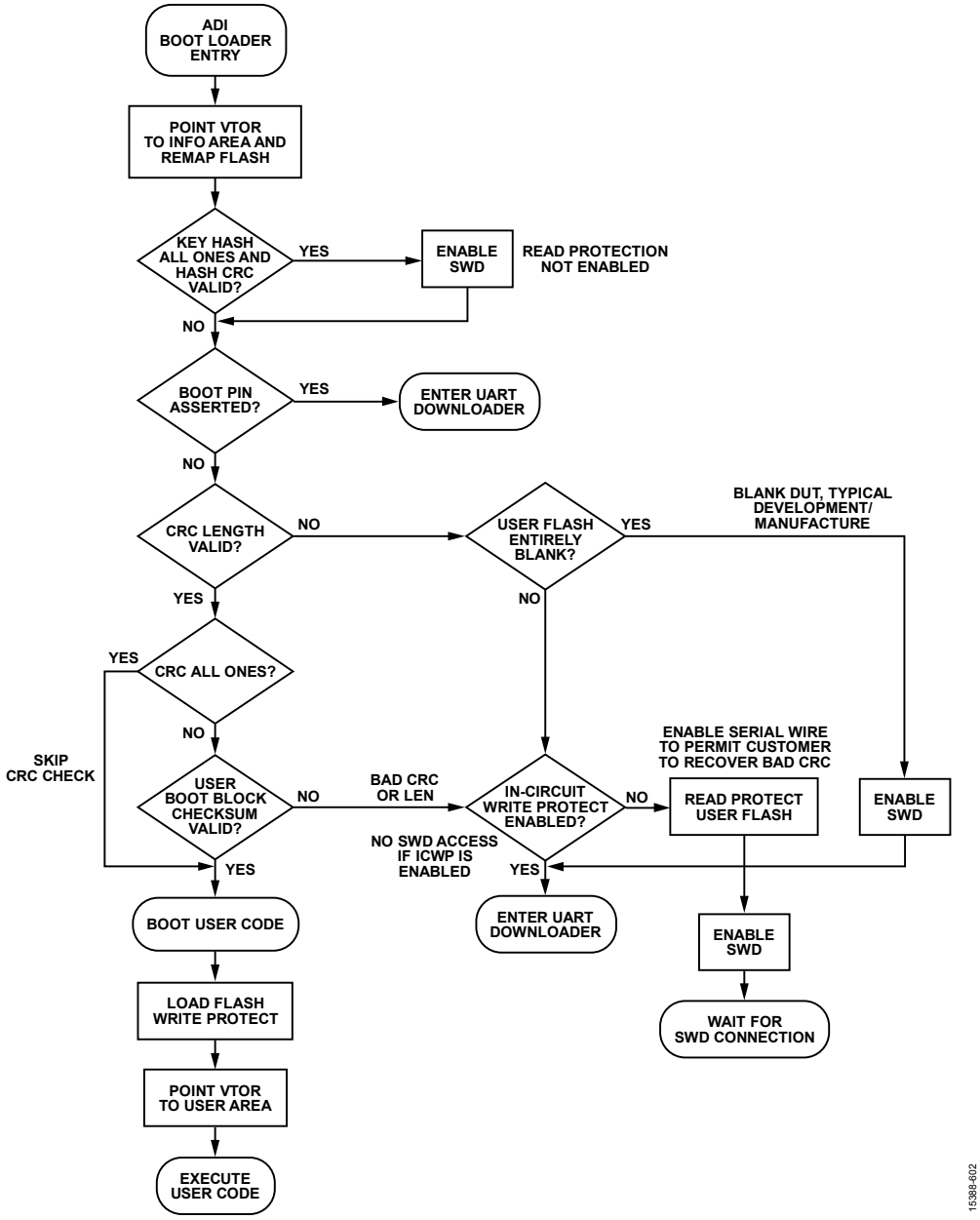


図 38. ブート・カーネルのフローチャート

UART 経由でダウンロードされた SSL コードは、SRAM にマップする必要があります。UART ダウンローダ・モードでは、SSL が SRAM にロードされ、フラッシュ・プログラム機能を実行します。カーネルは SSL を認証し、認証が成功した場合のみ実行を許可します。このコードは、ユーザー・フラッシュ・メモリの実際のファームウェア(ユーザー・アプリケーションなど)のダウンロードとアップグレードを実行します。カーネルは、ユーザー・フラッシュ・メモリへの直接アップデートをサポートしていません。このようなアクションを実行するには、SSL が必要です。

カーネルは特定のプロトコルに従い、プロセッサに SSL をダウンロードします。これは送信元のホストに従う必要があります。SSL がカーネルと同じパケット・プロトコルに従う場合、ホスト・インターフェースは簡略化されます（カーネルや SSL との通信が同型であるなど）。プロトコルの詳細については、次のセクションで説明します。

UART ダウンローダ

SYS_BMODE0 ピン（GPIO17）がローの場合、ADuCM3027/ADuCM3029 プロセッサは UART ダウンローダ・モードに移行します。パワーオンまたはハード・リセット時に、デバイスが条件を検出すると、シリアル・ダウンロード・モードに移行します。このモードでは、カーネル内のオンチップ・ローダ・ルーチンが開始され、デバイスの UART ポートを構成し、特定のシリアル・ダウンロード・プロトコルを介してホストと通信することで、ファームウェアのアップグレード・プロセスを管理します。図 39 に、UART ダウンローダのフローを示します。

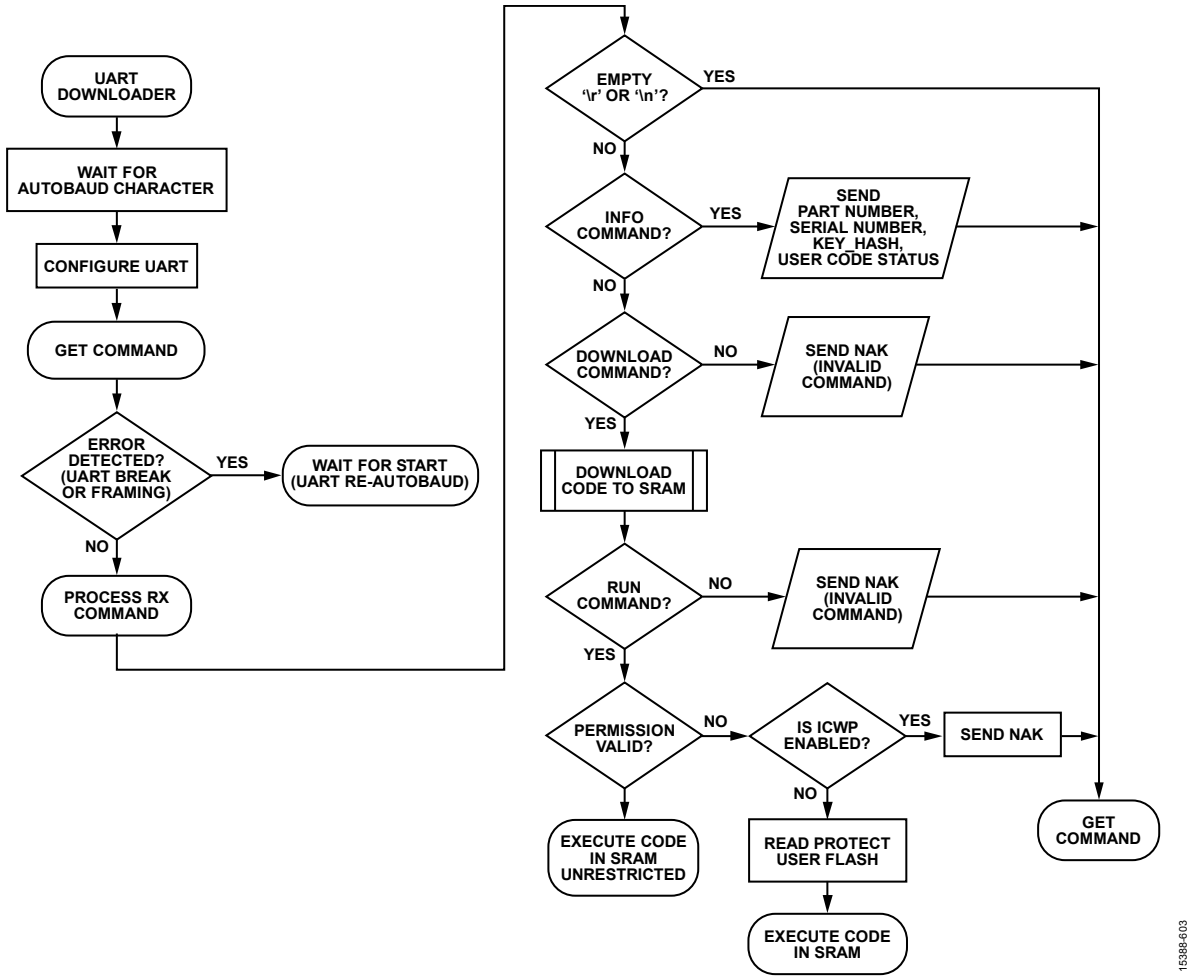


図 39. UART ダウンローダのフローチャート

プロトコル

SYS_BMODE0 ピンをアサートすることでシリアル・ダウンローダがトリガされた後に、カーネルはホストからキャリッジ・リターン文字（ASCII 0x0D 文字、図 40 を参照）が送信されるまで待機し、UART オートボー・プロセスを開始します。

カーネルは UART オートボー機能を使用して、ホストのボーレートを検出した後、ホストのボーレートで、パリティなしの 8 データ・ビットで UART ポートを送信または受信に構成します。6.5 MHz のリセット・ペリフェラル・クロック（PCLK）によって、UART は最大 230,400 bps のボーレートをサポートするようにカーネルで構成できます。この値よりもボーレートが高くなるとエラーが増え、データ転送の信頼性が低くなります。ただし、SSL のロード後は、SSL が PCLK を増加させ（最大 26 MHz）、UART から 2 番目のオートボー検出を実行する場合は、ボーレートが高くなります。

オートボー文字の受信後、カーネルは必要なクロック分周器の値を計算して UART を構成します。この時点で、カーネルは 57 バイトの ID データ・パケットの一部としてデバイス情報を送信し（表 4 を参照）、オートボー検出プロセスが成功したことを通知します。

パケット構造

オートボーのアクノレッジには、プロセッサ通信の準備が整ったことをホストに通知することに加えて、デバイスに関する情報、ユーザー・フラッシュ・メモリの状態、セキュリティの制限も含まれます。オートボーアクノレッジの後に、データ転送自体は、データ転送パケット形式によって管理されるように開始できます（表 5 参照）。

パケット開始 ID フィールド、ID0 および ID1

最初の転送フィールドは、2 バイトのパケット開始 ID フィールド（ID0 および ID1）で、2 つの開始文字（ID0 では 0x07 および ID1 では 0x0E）で構成されます。これらのバイトは定数で、ローダーが有効なデータ・パケットの開始を検出するために使用されます。

データ・バイト・フィールドの数

次の転送フィールドは、データ・バイト・フィールドの合計数です。1 バイトのコマンド（CMD）、4 バイトのアドレス（値）、残りのペイロード（データ）が含まれます。データ・バイトの最低数は 5 で、コマンドとアドレスのみに対応します。データ・バイトの最大数は 255 で、コマンド、アドレス、最大 250 バイトのデータをサポートします。

コマンド機能フィールド（CMD）、データ・バイト 1

コマンド機能フィールドは、データ・パケットの機能について説明します。これらのコマンドはカーネルでサポートされ、ASCII 形式で表現されます。

- W（0x57）- 書込みコマンド
- R（0x52）- 実行コマンド
- I（0x45）- 情報コマンド

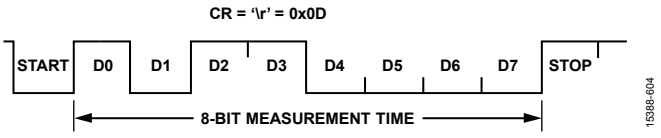


図 40. オートボー文字

表 4. 自動ボー応答

Bytes	Contents
1 to 15	Product identifier: ADuCM302xand six spaces, where x = 7 or 9
16 to 18	Hardware and firmware version numbers
19	User code blank; x means the code to execute, and a dash (-) means the user code is blank
20	User code checksum; P means that the checksum passed, and F means that the checksum failed
21	Write protection enabled; W means that write protection is disabled, and a dash (-) means that the write protection is enabled
22	Read protection enabled; R means that read protection is disabled, and a dash (-) means that read protection is enabled
23	Space
24 to 55	128-bit serial number, as a 32-digit uppercase hexadecimal number (for example, 0123456789ABCDEF0123456789ABCDEF)
56	Line feed
57	Carriage return

表 5. UART パケット構造

ID0	ID1	Number of Data Bytes	CMD	Value	Data	Checksum
0x07	0x0E	5 to 255	W, R, or I	h, u, m, l	xx	CS

書込みコマンド

表 6 に示す書込みコマンド・パケットには、データ・バイトの数（5 + n、n はバイト単位のペイロードのサイズ）、書込みコマンド（W）、32 ビットの開始アドレス、ペイロード内の n データ・バイトが含まれます。

カーネルが書込みコマンド・パケットを受信すると、その後ペイロード・バイトは到着時に SRAM に配置され、開始アドレスから開始されます。チェックサムが正しくない場合または受信アドレスが範囲外である場合、カーネルはノー・アクノレッジ・コマンドを送信します。ホストがローダからノー・アクノレッジを受信した場合は、ダウンロード・プロセスをアボートし、再起動します。

実行コマンド

ホストは、すべてのデータ・パケットをカーネルに転送した後に、カーネルにコードの実行を開始するように命令する最終的なパケットを転送できます。最終的なパケットは、実行コマンド・パケットを送信して実行されます。これは、実行コマンド（R）と実行を開始する 32 ビットのアドレスで構成されます（表 7 を参照）。

カーネルが実行コマンド・パケットを受信すると、承認チェックに合格した場合に限り、パケットで指定されたアドレスにジャンプします。

情報コマンド

ホストはいつでも 表 8 に示す情報コマンド・パケットを送信できます。このパケットは、コマンド（I）と 32 ビット・アドレスで構成されます。カーネルで適切に受信されるには、パケットで値フィールドが必要ですが、値フィールドの内容は関連がありません。

カーネルが情報コマンド・パケットを受信すると、57 バイトの ID パケットで応答します（表 9 を参照）。

表 6. 書込みコマンド・パケット

ID0	ID1	Number of Data Bytes	CMD	Value	Data	Checksum
0x07	0x0E	5 + n	W (0x57)	Start address	n bytes	CS

表 7. 実行コマンド・パケット

ID0	ID1	Number of Data Bytes	CMD	Value	Checksum
0x07	0x0E	5	R (0x52)	Start address	CS

表 8. 情報コマンド・パケット

ID0	ID1	Number of Data Bytes	CMD	Value	Checksum
0x07	0x0E	5	I (0x52)	0xXXXXXXXX	CS

値フィールド（データ・バイト 2 ～ データ・バイト 5）

値フィールドには、h、u、m、ロケーションを含む 32 ビット・アドレスが含まれます。MSB は h ロケーション（データ・バイト 2）にあり、LSB は l ロケーション（データ・バイト 5）にあります。前に説明したように、

- 書込みコマンド・パケットでは、値フィールドはデータ・ペイロードの書込みを実行するメモリ内の開始アドレスを示します。
- 実行コマンド・パケットでは、値フィールドは SSL コードが開始する SRAM 内のアドレスを示します。
- 情報コマンド・パケットでは、値フィールドに意味はありません。

データ・フィールド（データ・バイト 6 ～ データ・バイト 255）

ユーザー・コードは、1 度に 1 バイトずつダウンロードされ、データ・フィールドには、最大 250 バイトを挿入できます。通常、データは Intel® HEX 拡張 16 バイト・レコード形式から取り除かれ、ホストによって再アセンブリされ、一連の書込みコマンド・パケットを使用して、ADuCM3027/ADuCM3029 プロセッサに送信されます。

チェックサム・フィールド（CS）

データ・パケット・チェックサムは、チェックサム・フィールドに書込まれます。この 2 つの補数チェックサムは、バイト・フィールドの番号からデータ・フィールドの最後までまたがる 16 進数の値の合計から計算されます。このため、データ・バイト・フィールドの番号からチェックサム・フィールドを含むパケットのすべてのバイトの合計の 8 ビット LSB は 0 になります。

コマンドのアクノレッジ

ローダ・ルーチンは、受信した各データ・パケットに対して、負の応答のノー・アクノレッジ・コマンド（0x07）または正の応答のアクノレッジ（0x06）を発行します。

ローダは、次の条件を満たした場合にノー・アクノレッジを送信します。

- ローダが誤ったチェックサムを受信した。
- UART フレーミングまたはブレーク・エラーが発生した（このエラーは、UART リンクが無効な場合にホストに到達しない場合があります）。
- SRAM コードの検証に失敗した。

これらの条件のいずれかを満たした場合は、ターゲットをリセットし、ファームウェアのアップグレード・プロセスを再起動する必要があります。これらの条件のいずれも満たさない場合は、アクノレッジ・コマンドが送信されます。

読出し保護キーとハッシュ

読出し保護キーを使用して、故障解析中にデバイスにアクセスできます。デバイスが読出し保護されている場合に、現在のフラッシュ・メモリの内容の故障分析が必要なときは、ユーザー・フラッシュ・メモリに保存されているハッシュに対応するキーを送信して SWD インターフェースを有効にできます。キーはデバイス専用にし、デバイス専用の ID（情報スペースに保存されているシリアル番号など）をベースにすることが推奨されます。

ハッシュは、割込みベクトルの後のユーザー・フラッシュ・メモリに保存されます。これは、秘密のカスタム・キーのハッシュです。セキュリティ上の理由から、このキーはデバイス専用にし、キーのロック解除が特定のデバイス 1 台に対して有効になることを強く推奨します。デバイスのキーをユニークに維持するには、どのキーが特定のデバイスに属するか関連付けるデバイス ID が必要です。簡単なキー管理では、キーをマスター・シークレットのハッシュとデバイス ID にすることをお勧めします。例えば

読出し保護キー = ハッシュ（マスター・シークレット || ユニークなデバイス ID）

キー・ハッシュ = ハッシュ （読出し保護キー）

カーネルが UART ローダ・モードの場合は、読出し保護キーを受信できます。次に、ブート・ローダは読出し保護キーのハッシュを実行し、保存されているキー・ハッシュと比較します。マッチが成功したら、ブートローダはすべての権限を有効にして、SRAM にダウンロードされた SSL コードが実行されることを許可します。キー・ハッシュのチェックに失敗した場合、カーネルはユーザー・フラッシュ・メモリの ICWP キーをチェックします。ユーザーがアドレス 0x00000198 の値を NoWr ASCII 文字列の 16 進相当以外の値にプログラムすることで ICWP がオフになっている場合、SSL は、フラッシュの読出しおよび書込みアクセスの保護の後に実行を許可されます。この場合、SSL は最初にユーザー・フラッシュ・メモリの大容量消去を発行してから、ユーザー・フラッシュ・メモリ・スペースへの読書きを試みます。ユーザーが ICWP も有効にしている場合、SSL はキー・ハッシュ認証が合格するまで実行を許可されません。

128 ビットの読出し保護キーが SRAM コードの一部として渡されます。このキーは、アドレス 0x20000180 で開始するデータ・ペイロードとしてビッグエンディアン形式で SRAM に保存する必要があり、カーネルによって正しく解析できるように、メモリ内で特定の形式に一致させる必要があります。具体的には、読出し保護キーが ABCDEFGHIJKLMNOP として表現され、各文字が 1 バイトを表す場合（A が最初のバイトで P が最後のバイト）、メモリ内のバイトの必要な配置を 表 9 に示します。

表 9. SRAM の読出し保護

Address	Byte 0	Byte 1	Byte 2	Byte 3
0x20000180	D	C	B	A
0x20000184	H	G	F	E
0x20000188	L	K	J	I
0x2000018C	P	O	N	M

表 10 に、読出し保護キーが 0x00010203040506070-8090A0B0C0D0E0F の場合のメモリの書込み方法を示します。

表 10. SRAM の読出し保護キーの例

Address	Byte 0	Byte 1	Byte 2	Byte 3
0x20000180	0x03	0x02	0x01	0x00
0x20000184	0x07	0x06	0x05	0x04
0x20000188	0x0B	0x0A	0x09	0x08
0x2000018C	0x0F	0x0E	0x0D	0x0C

カーネルは、このキーの SHA-256 ハッシュを計算し、128 ビット・ハッシュに切り詰め、アドレス 0x00000180 のユーザー・フラッシュ・メモリの 0 ページに保存されているハッシュと比較します。ユーザーは、同様のパターンを使用して、128 ビットに切り詰められたキーのハッシュを保存する必要があります。表 10 に示すサンプルのキーの SHA-256 ハッシュは、0xBE45CB2605BF36BEBDE68-4841A28F0FD43C69850A3DCE5FEDBA69928EE3A8991 です。つまり、ユーザー・フラッシュ・メモリ・スペースに適切に保存する必要がある 128 ビットの切り詰められたハッシュは、表 11 に示すように配置された 0x43C69850A3DCE-5FEDBA69928EE3A8991 です。

表 11. フラッシュ・メモリの読出し保護キー・ハッシュの例

Address	Byte 0	Byte 1	Byte 2	Byte 3
0x00000180	0x50	0x98	0xC6	0x00
0x00000184	0xFE	0xE5	0xDC	0xA3
0x00000188	0x28	0x99	0xA6	0xDB
0x0000018C	0x91	0x89	0x3A	0xEE

キー・ハッシュの CRC32 は、0x04C11DB7 の多項式と 0xFFFFFFFF のシード値で計算され、アドレス 0x00000190 のフラッシュ・メモリ・スペースに LSB 形式で保存されます。

メモリ構成

表 12 に、ユーザー・フラッシュ・メモリの 0 ページに保存されている各キーとパラメータ、関連付けられたアドレス、プロジェクトをデフォルトのスタートアップ・ファイルで作成する場合の 0 ページにプログラムされる値を示します。

表 12. 0 ページのメモリ構成

Content	Address Range		Size (Bytes)	Section Name	Default Content
	Start Address	End Address			
Vector Table	0x0000_0000	0x0000_017F	384	.intvec	Vector table
Read Protection Key Hash	0x0000_0180	0x0000_018F	16	ReadProtection KeyHash	0xFFFFFFFF 0xFFFFFFFF 0xFFFFFFFF 0xFFFFFFFF
CRC of Read Protection Key	0x0000_0190	0x0000_0193	4	CRC_ReadProtection KeyHash	0xA79C3203
Number of Pages the CRC Computes	0x0000_0194	0x0000_0197	4	NumCRCPages	0
Checksum	0x0000_07FC	0x0000_07FF	4	Checksum	Checksum of 0 to 0x7FB (if enabled in tools by the user)
Page 0 User Memory	0x0000_01A0	0x0000_07FC	1628	Page0_region	User application

IAR ワークベンチの CRC の処理

アプリケーション・イメージの部分から、フラッシュ・メモリの最初の複数のページにロードする CRC を計算します。CRC 計算に関連する最後のページ番号を 32 ビット整数としてアドレス 0x194 に保存します。例えば、0 ページだけが CRC 計算に関連する場合、アドレス 0x194 に値 0x00 を保存します。最初の 3 ページに対して CRC が計算される場合、値を 0x02 にする必要があります。

CRC の計算では、計算に含まれる最後のページの最後の 4 バイトは除外されます。これらの 4 バイトは、CRC 値自体を保存するために使用されます。例えば、最後のページが 0 ページの場合、CRC はアドレス 0x000 からアドレス 0x7FB まで計算されます。ツールは、計算された CRC 値をアドレス 0x7FC に 32 ビット整数として保存します。

標準の CRC 計算は、CRC32 と 0x04C11DB7 の多項式で、初期値 0xFFFFFFFF で MSB ファースト形式で保存されます。単位サイズは 32 ビットなので、ツールは CRC の計算時にイメージから一度に 32 ビットを読み出す必要があります。

［Checksum］タブ

IAR ツールの［**Linker**］カテゴリにある［**Checksum**］タブを使用して、ユーザー・アプリケーション・コードの CRC を生成できます。正しい CRC を保存するには、次の設定を使用する必要があります（図 41 を参照）。

- ［**Fill unused code memory**］ボックスをオンにします。
- ［**End address:**］フィールドを 0x7FB に設定します（この値は、ページ番号に応じて変更します）。
- ［**Generate checksum**］ボックスをオンにします。
- ［**Checksum size:**］プルダウン・メニューから［**4 bytes**］オプションを選択します。
- ［**Alignment:**］フィールドを 4（4 バイトを示す）に設定します。
- ［**Algorithm:**］プルダウン・メニューから［**CRC32**］オプションを選択します。
- ［**Initial value**］フィールドを 0xFFFFFFFF に設定し、［**Use as input**］ボックスがオンになっていることを確認します。
- ［**Checksum unit size:**］プルダウン・メニューから［**32-bit**］オプションを選択します。

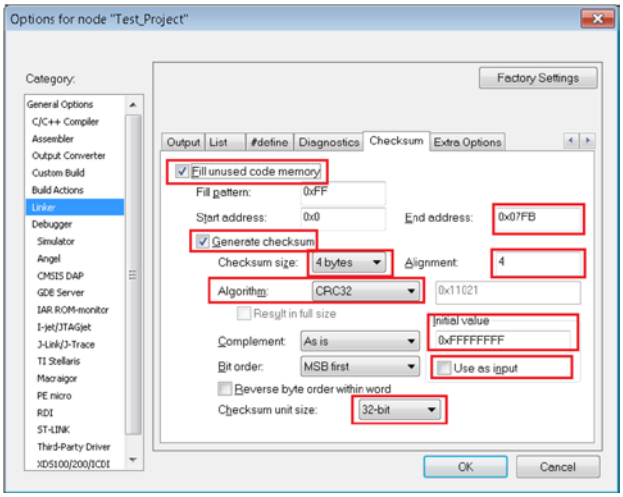


図 41. チェックサムの設定

CCSFP

CrossCore® Serial Flash Programmer（CCSFP）は、アナログ・デバイセズが提供する PC ベースのホスト・ユーティリティです。UART ポート経由で、ユーザー・コードをアップグレードできます。CCSFP のグラフィカル・ユーザー・インターフェース（GUI）には、UART をアップグレードするための次のオプションがあります。

- ターゲット・プロセッサ
- UART PC ポート番号
- ボー・レート
- アップグレードに使用する SSL 16 進数ファイル
- アップグレードするユーザー・アプリケーションの 16 進数ファイル
- SSL へのキー認証

図 42 に、CCSFP の GUI を示します。［**Second stage kernel**］フィールドで SSL を指定する必要があります。これはプロセッサの SRAM にダウンロードされ、実行されてから、認証に基づいて［**File to download**］フィールドにあるユーザー・アプリケーションがフラッシュに送信されます。認証の 128 ビットキーは、［**Key**］フィールドに入力できます。

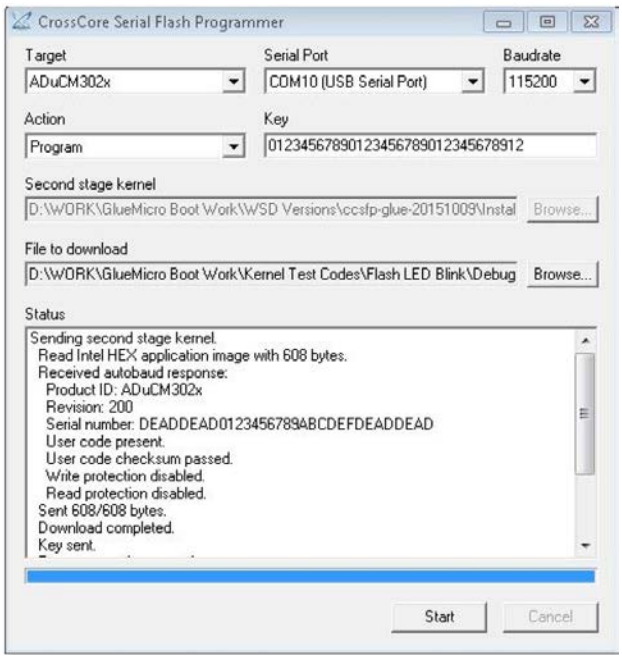


図 42. CrossCore シリアル·フラッシュ・プログラマ GUI

［**Status**］ウィンドウには、UART ダウンロード・プロセスの状態、デバイス関連情報、カーネルに返されるコマンドのステータスがあります。図 42 に示すように、［**Status**］ウィンドウには、カーネルが送信したデバイス情報が表示され、製品 ID、シリアル番号、ユーザー・コード・ステータスが表示されます。SSL がダウンロードされたら、［**Status**］ウィンドウに表示される［**Download completed**］メッセージに示すように、SSL がカーネルによって認証され、実際のユーザー・アプリケーションが送信されます。

図 43 に、デバイス上の SSL の実行、ユーザー・アプリケーションの受信、ユーザー・フラッシュ・メモリ・スペースへの書込みを示します。

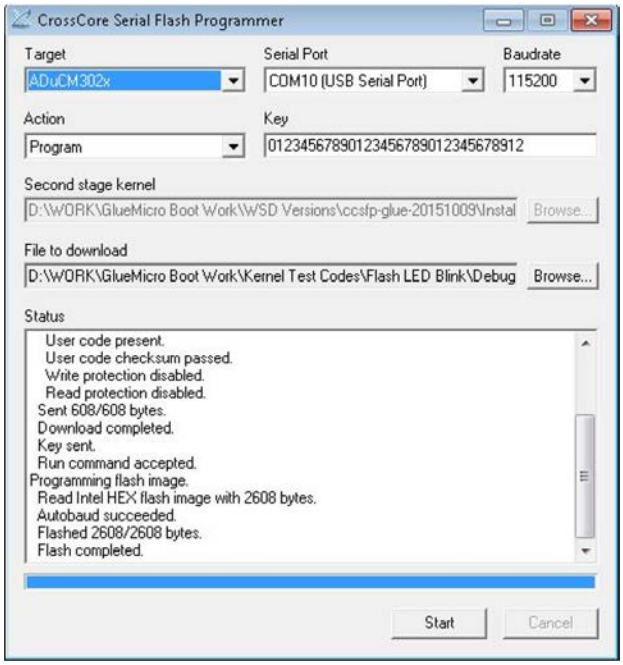


図 43. CCSFP 経由で SSL によってユーザー・フラッシュ・メモリに書込まれているユーザー・アプリケーション・コード

ADUCM3027/ADUCM3029 のキャッシュ・メモリ

ADuCM3027/ADuCM3029 のメモリは、ECC 内の最大 256 KB の内蔵フラッシュ・メモリ、32 KB のパリティ付きデータ・スタティック・ランダム・アクセス・メモリ（SRAM）、32 KB のパリティ付きユーザー構成可能な命令／データ SRAM です。4 キロバイトの SRAM をキャッシュ・メモリに使用して、フラッシュ・メモリへのアクセスを減らすことで、有効電力の消費を削減できます。

ADuCM3027/ADuCM3029 のキャッシュは、命令コード（ICODE）とデータ・コード（DCODE）の読書き用の下位の電力キャッシュ・コントローラ、双方向結合の 4 KB の命令キャッシュ、256 ビットのライン・サイズです。命令キャッシュには、最も長い間使われていないものが置換されるポリシーがあります。キャッシュがフラッシュへの書込みを実行すると、コアはフラッシュ・コントローラの APB（アドバンスト・ペリフェラル・バス）インターフェースからのみフラッシュへの書込みを発行できます。コアがフラッシュに配置されている場合、キャッシュを有効にすると、実行速度を上昇できます。詳細については、キャッシュが実行速度に与える影響のセクションを参照してください。消費電流の詳細については、消費電流の比較のセクションを参照してください。

ここでは、オンボード・キャッシュ・コントローラを使用して、SRAM の一部をユーザー・コードの命令／データ・キャッシュとして使用する方法を説明します。これ以外の場合では、ユーザー・コードはフラッシュ・メモリから実行されます。

ブロック図

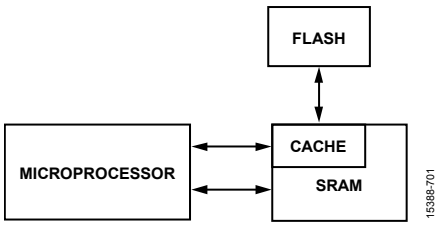


図 44. ブロック図

ADuCM3027/ADuCM3029 キャッシュ・アーキテクチャは、デジタル・キャッシュ・コントローラ、システム SRAM の一部として実装されるキャッシュ・メモリ、デジタル・フラッシュ・コントローラ、フラッシュ・メモリで構成されます。キャッシュ・アーキテクチャは、高速の SRAM メモリを使用することで命令とデータ・アクセスの平均遅延を減らし、比較的に消費電力が高いフラッシュ・メモリへのアクセスの頻度を減らします。

キャッシュを有効にしてフラッシュ・メモリからコードを実行すると、頻繁に使用される命令は SRAM の専用領域に自動的にキャッシュされます。ロック機能と制御機能が提供されますが、大部分のアプリケーションでユーザーの作業は不要です。

フラッシュ・コントローラ

フラッシュ・コントローラは、2 つの AMBA AHB（アドバンスト・マイクロコントローラ・バス・アーキテクチャ、高性能バス）ポートを備えたキャッシュ・コントローラ・モジュールと組み合わせます。一方のポートはデータの読出し用（DCODE）で、他方のポートは命令の読出し用（ICODE）です。フラッシュ・コントローラは、ICODE と DCODE の同時読出しアクセスをサポートします。競合が発生した場合は、DCODE の優先度が高くなります。

フラッシュ・コントローラは、ICODE の読出し性能を最適化するため、プリフェッチ機構を実装しています。この機構は、ICODE インターフェースの連続したアドレスの読出しを実行するときに最適な性能を発揮します。同時読出しは、ICODE 読出しがプリフェッチのバッファ済みデータを返す場合に可能です。

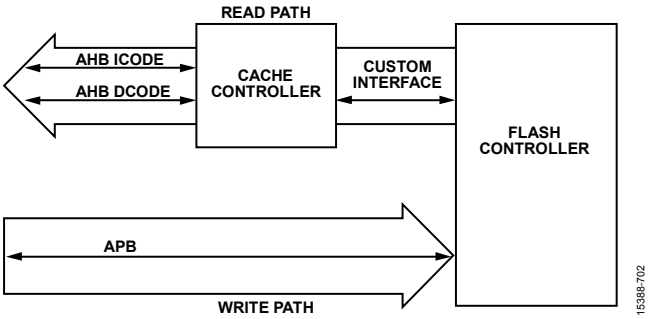


図 45. フラッシュ・コントローラとキャッシュ・コントローラ

キャッシュの効果

キャッシュが実行速度に与える影響

フラッシュ・メモリと SRAM メモリには、電力と性能について異なるプロファイルがあります。

キャッシュ・アーキテクチャは、実行中にユーザー・コードの一部を SRAM にコピーするので、命令とデータの読出しの遅延が短くなります。キャッシュ・メモリからの命令またはデータの読出しが成功する度に、全体的なシステム性能が向上します。

キャッシュとフラッシュを組み合わせて使用する場合、速度の増加はコードのタイプに依存します。一般に、実際のコードを観察すると、キャッシュに完全に収まるループ・コードでは、読書き速度はフラッシュだけを使用する場合よりも 15 % ～ 20 % 速くなります。キャッシュに収まらない線形コードでは、読書き速度は 10 % ～ 15 % 向上します。

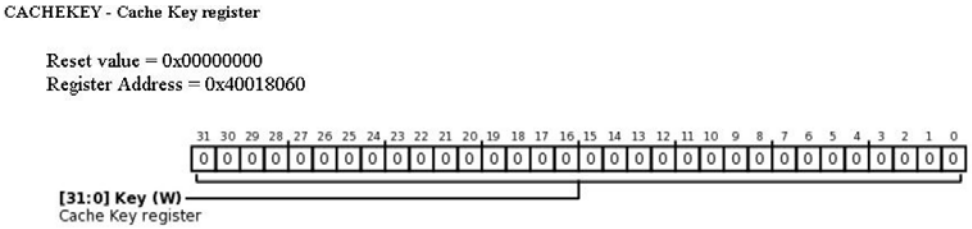
通常、キャッシュを使用すると、実行速度は向上しますが、向上の度合いは使用するコードのタイプによって異なります。コードに、キャッシュに完全に収まるループがある場合は、命令アクセスの大部分は、高速の SRAM から実行されるため、実行速度は大幅に向上します。一般に、コードが線形であったり、セグメント間のジャンプ幅が大きすぎてキャッシュに収まらない場合、命令アクセスの大部分は、速度の遅いフラッシュ・メモリから実行されるため、実行速度は大幅には向上しません。

各キャッシュ・ミスは、フラッシュ・メモリ 64 ビットの呼び出しで構成されるキャッシュ・ライン・フィルになります。

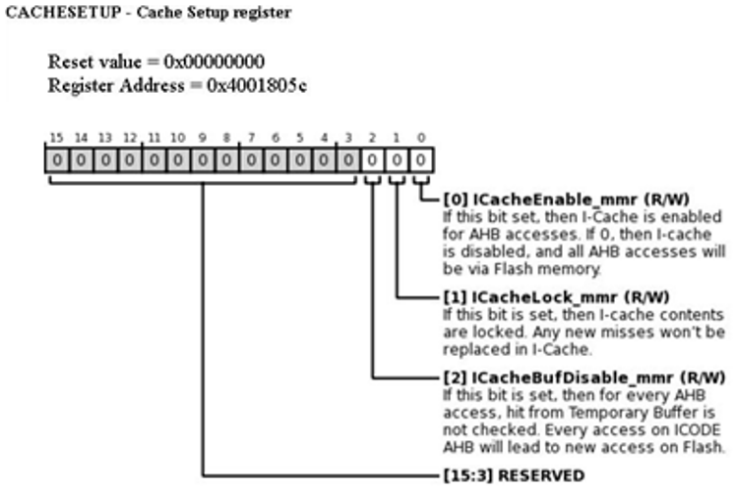
キャッシュをイネーブルまたはディスエーブルにするには、次の指示に従います。

1. 命令キャッシュ（ICACHE）は、デフォルトではディスエーブルです。ICACHE をイネーブルにするか、切り替えるには、0xF123F456 キーを CACHEKEY レジスタに書込む必要があります。
2. ここでは CACHESETUP レジスタの詳細を示します。
3. ICACHE をイネーブルにするには、CACHESETUP レジスタの ICacheEnable_mmr ビットをセットします。ICACHE をディスエーブルするには、このビットをクリアします。
4. ICacheLock_mmr ビットを使用すれば、今後のミスでキャッシュが置き換えられないようにキャッシュをロックできます。この機能は、ループがキャッシュのサイズよりも大きな場合に役立ち、適切なロックを配置することで電力を節約できます。
5. ICACHE の内容をロックするには、CACHESETUP レジスタの ICacheLock_mmr ビットをセットします。CACHESETUP レジスタの ICacheLock_mmr ビットをクリアしてキャッシュのロックを解除します。

キャッシュ・キー・レジスタ



キャッシュ・セットアップ・レジスタ



キャッシュが電流に与える影響

SRAM のアクセスで消費される電流は、フラッシュのアクセスで消費される電流よりも少なくなります。そのため、フラッシュ・メモリからのコード実行中にキャッシュがイネーブルな場合、消費電流の値は、フラッシュから直接コードを実行する場合と SRAM から直接コードを実行する場合の間に収まります。ここでは、各キャッシュへのアクセスがミスしないことを前提にしています。この場合、電流はフラッシュのみから実行する場合よりも高くなります。キャッシュ・ライン・フィルでは、フラッシュから直接実行する場合よりもフラッシュからの読出し回数が約 2 倍以上になります。この読出し速度は、同様に読出しを実行しているフラッシュのプリフェッチ・バッファでない場合の 4 倍になります。この場合、キャッシュがミスすると、読出しもミスします。

$$電流_{SRAM} \leq 電流_{CACHE}$$

キャッシュを使用する場合は、消費電流は、キャッシュ・ミス率に比例します。この結果は、キャッシュ・ヒットごとのスカラ電流の削減に起因します。データまたはコードへのアクセスが高電流のフラッシュ・メモリではなく、低電力の SRAM から実行されるためです。そのため、多数の小さなループで構成されるコードは、線形コードや、大きすぎてキャッシュ・メモリに収まらないセグメントで構成されるコードよりも大きな影響を受けます。

キャッシュの使用量は、消費電流からも推測できます。キャッシュとフラッシュを使用した消費電流が SRAM を使用した消費電流に近い場合は、キャッシュのヒット率が高いと推測されます。キャッシュとフラッシュを使用した消費電流がフラッシュを使用した消費電流に近い場合は、キャッシュのヒット率が低いと推測されます。

消費電流の比較

ループ・コード（この例では素数コード）の場合、フラッシュとキャッシュを使用した消費電流（980 µA）は、SRAM の消費電流（950 µA）に非常に近いです。これは、キャッシュに収まるほど小さい多数のループでコードが構成されているからです。そのため、フラッシュへのアクセスは比較的頻度が少なく、大部分のコードは低消費電力キャッシュから実行されます。アクセスは最低限なので、SRAM を使用した場合の消費電流と比較すると、フラッシュとキャッシュを使用した場合の消費電流では、最低限の増加しか観察されません。

線形コード（この例では ULPBench コード）では、フラッシュとキャッシュを使用した場合の消費電流は、SRAM のみを使用した場合の消費電流から逸脱しています。これは、ULPBench コードがほぼ線形でキャッシュにうまく収まらないので、フラッシュ・メモリへのアクセス回数が、フラッシュから直接実行する場合と同様の回数であるためです。キャッシュ・ミスが高くなるので、消費電流は大幅に逸脱します。

表 13. 消費電流の比較

Type of Code	SRAM (µA)	Flash (µA)	Flash and Cache (µA)	Cache Misses (~12 sec of Execution)
Loop code (prime number)	950	1280	980	24
Linear code (ULPBench)	911	1493	1083	~800000

ADUCM3027/ADUCM3029 のデュアル RTC 機能

多くのアプリケーションで、RTC はセンサー・データのタイム・スタンプに使用されます。RTC は、MCU がディープ・スリープ・モードの場合でも実行される必要があります。低消費電力 RTC は、バッテリの寿命を長持ちさせるために必須です。

ADuCM3027/ADuCM3029 RTC の注目すべき機能には、次のものがあります。

- デュアル RTC（RTC0 と RTC1）。両方の RTC は、ウェイクアップ・タイマーとして使用できます。
- SensorStrobe と入力キャプチャ機能。

ここでは、必要な電力モードと機能によって、RTC0 と RTC1 のどちらを選択するかのガイドラインを提供します。

RTC 機能の比較

ADuCM3027/ADuCM3029 には、RTC0 と RTC1（別名 FLEX_RTC）の 2 つの RTC があります。表 14 に、RTC0 と RTC1 の違いを示します。

電力に関する考慮事項

表 15 に、さまざまな使用事例で RTC0 と RTC1 を使用する場合の消費電流を示します。次の 4 つのシナリオを考えましょう。

- シナリオ 1.デバイスはアクティブ・モードと休止モードの間で切り替わり、アプリケーションでは高い精度が必要。このシナリオでは、RTC0 または RTC1 を使用できますが、RTC1 のほうが消費電力が少ないので、電力の観点から RTC1 を推奨します。
- シナリオ 2.デバイスはアクティブ・モードと休止モードの間で切り替わり、アプリケーションでは高い精度は不要。このシナリオでは、RTC0 または RTC1 を使用できますが、RTC1 のほうが消費電力が少ないので、電力の観点から RTC1 を推奨します。
- シナリオ 3.デバイスは、アクティブ・モードとシャットダウン・モードの間で切り替わる。このシナリオでは、RTC0 のみを実行できます。
- シナリオ 4.デバイスは、アクティブ・モード、シャットダウン・モード、休止モードの間で切り替わる。RTC0 のみを使用できます。このシナリオでは、RTC1 はシャットダウン・モードでアクティブでないからです。

スリープ電力モード（休止モードまたはシャットダウン・モード）の電流を測定するため、ADuCM3027/ADuCM3029 マイクロ・コントローラを低消費モードからウェイクアップして LED を切り替える RTC で構成される、基本的なプログラムが使用されました。

まとめ

シャットダウンモードを使用して RTC が必要なアプリケーションでは、RTC0 を使用します。

RTC1 は、アクティブ・モードや Flexi モードの他に、休止モードのみを使用するアプリケーションで超低消費電力を可能にする機能が豊富な RTC です。RTC1 が適している代表的なアプリケーションは、ADuCM3027/ADuCM3029 マイクロコントローラが出力パルスを汎用入出力から外部センサーに送信するアプリケーションです。SensorStrobe 機構は、RTC1 でのみ使用できます。

表 14. RTC0 と RTC1 の差のまとめ

Feature	RTC0	RTC1
Resolution of the Time Base (Prescaling)	RTC0 counts time at 1 Hz in units of seconds only	RTC1 can prescale the clock by any power of 2 from 1 to 15, counting time in units of any of these 15 possible prescale settings
Wake-Up Timer	The wake-up time is specified in units of seconds	The wake-up time can be specified in units of any power of 2 multiple of 30.7 µs up to 1 second
Number of Alarms	1 alarm only, which uses an absolute, nonrepeating alarm time	2 alarms: one absolute alarm time and one periodic alarm, repeating every 60 prescaled time units
Power Domain	Powered off VBAT domain and is always on; RTC0 can function in all power modes	Powered off 1.2 V (VREG) domain; RTC1 can function in all power modes, except shutdown mode
SensorStrobe and Input Capture Features	Not supported	Supports four input capture channels and one SensorStrobe channel; refer to AN-1427 for further information on these features
Source Clock	Low frequency crystal (LFXTAL)	Depending on the low frequency multiplexer (LFMUX) configuration, the RTC is clocked by LFXTAL or the low frequency oscillator (LFOSC)

表 15. 各使用事例での消費電力の比較

Scenario Number	Use Case[1]	Recommended RTC	Sleep Mode Current (nA)
1	Active to hibernate	RTC1 (LFXTAL)	830
2	Active to hibernate	RTC1 (LFOSC)	750
3	Active to shutdown	RTC0 (LFXTAL)	330
4	Active to hibernate	RTC0 (LFXTAL)	830

[1] デバイスは、この列のモードの間で切り替わります。

ADUCM3027/ADUCM3029 DC/DC コンバータのメリット

ここでは、チャージ・ポンプ・コンバータとインダクタ・コンバータのメリットとデメリットについて説明します。頻繁に使用されるのは後者です。ここでは、このアーキテクチャがADuCM3027/ADuCM3029 マイクロコントローラで使用される理由を、価格、地域、使いやすさなどの多くの側面から説明します。

DC/DC コンバータは、ADuCM3027/ADuCM3029 マイクロコントローラなどの差動電圧領域を管理する必要がある設計において、重要な構成要素です。

ここでは、状況を理解できるように、DC/DC 変換の方法を簡単に説明します。ADuCM3027/ADuCM3029 では、チャージ・ポンプ・コンバータを使用することが選択されていますが、これは他の構成と比較して利点があるためです。

ADuCM3027/ADuCM3029 が対象とする超低消費電力アプリケーションでは、容量性 DC/DC コンバータが誘導性変換ソリューションの優れた代替品になります。

ここでは、詳細と例を示して、このチャージ・ポンプ・コンバータ・ソリューションの品質と利点を明らかにします。図 46 に、ADuCM3027/ADuCM3029 マイクロコントローラの降圧イネーブル設計を示します。ADuCM3027/ADuCM3029 は、チャージ・ポンプ・コンバータを使用しています。チャージ・ポンプ・コンバータは、同様の特性を持つ市販のマイクロコントローラでは使用されません。このようなマイクロコントローラは、従来のインダクタ・コンバータ・アーキテクチャを使用しています。

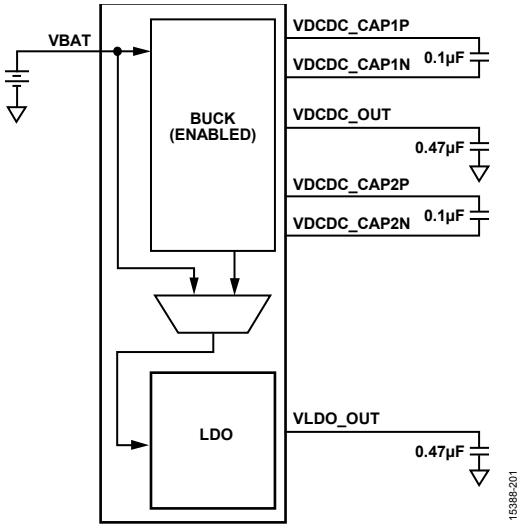


図 46.ADuCM3027/ADuCM3029 降圧イネーブル設計

DC/DC の基本

ADuCM3027/ADuCM3029 プロセッサは、超低消費電力アプリケーションを対象にしています。このようなアプリケーションでは、電力効率は大きな考慮事項の 1 つです。そのため、可能な限り効率よく電力を使用する必要がある設計では、DC/DC コンバータは設計に欠かせません。

DC/DC 電圧変換には、さまざまな方法があります。このような変換では、デバイスに給電するために DC 電圧のステップ・アップとステップ・ダウンが行われます。

DC/DC 変換方法

システムのさまざまな電力領域を安定させるため、最もよく使用される方法は、スイッチング変換とリニア・レギュレーションです。設計またはアプリケーションの条件を最もよく満たす方法を選択してください。

リニア電圧レギュレータ

リニア電圧レギュレータは、可能な電圧を散逸する抵抗分周器のネットワークで構成されます。リニア電圧レギュレータは、使用と実装が簡単で低価格なため、広く使用されています。

超低消費電力アプリケーションにおいて、リニア電圧レギュレータはスイッチング・コンバータよりも効率が低くなります。リニア電圧レギュレータでは、出力電流は入力電流とほぼ同じで、動作原理は余剰電圧を散逸するというものです。コンバータを切り替えると、同じアクションを効率的に実行できます。

ADP165/ADP166 デバイスは、超低静止電流、低ドロップアウト（LDO）、リニア電圧レギュレータです。グラウンド電流は、入出力電流の差を表します。図 47 に、ADP165/ADP166 グラウンド電流と負荷電流（I_{LOAD}）を表します。リニア電圧レギュレータでは小さな差を示しています。

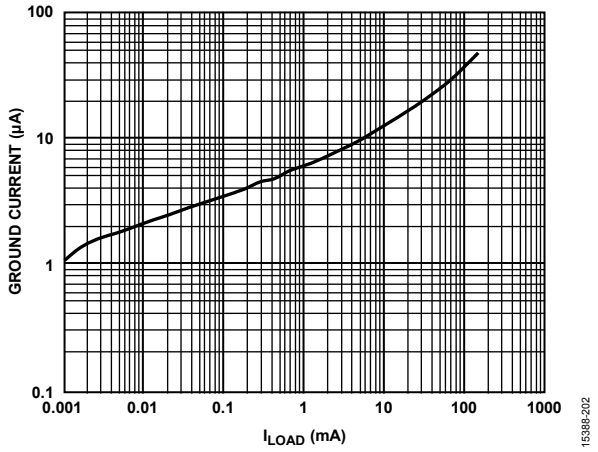


図 47. ADP165/ADP166 グラウンド電流と負荷電流（I_{LOAD}）

ソリューションの電力効率を分析するため、典型的なリニア電圧レギュレータに基づくアプリケーションを考えます。入力電圧が 3 V、出力電圧が 1 V、出力電流が 1 µA の場合は、入力電流は約 1 µA です。このシナリオでは、効率は 33 % になります（式 3 を参照）。

$$効率 = \frac{Energy\ Output}{Energy\ Input} \times 100\ \% \quad (1)$$

$$効率 = \frac{I_{OUT} \times V_{OUT} \times t}{I_{IN} \times V_{IN} \times t} \times 100\ \% \quad (2)$$

$$リニア電圧レギュレータの効率 = \frac{1 \times 1 \times t}{1 \times 3 \times t} \times 100\ \% = 33\ \% \quad (3)$$

リニア電圧レギュレータの代わりにスイッチング・コンバータを使用すると、入力電流は式 4 に示すように 1/3 µA で 100 % の効率になり、最適な性能を発揮します。

$$効率 = \frac{1 \times 1 \times t}{1/3 \times 3 \times t} \times 100\ \% = 100\ \% \quad (4)$$

一般に、スイッチング・コンバータは、リニア電圧レギュレータよりも効率的です。さらに、同じ変換を達成するために散逸が大きくなるので、リニア電圧レギュレータでは、効率の損失により温度が高くなります。さらに、リニア電圧レギュレータでは、温度を下げるために多大な投資が必要になります。

従来、シンプルで低価格であることから、アナログ・デバイセズでは、ADuCM3027/ADuCM3029 マイクロプロセッサ以前の設計においてリニア電圧レギュレータを使用していました。現在では、リップル出力を安定させるために、チャージ・ポンプ・コンバータの出力にリニア電圧レギュレータを配置するのが一般的です。

スイッチング・コンバータ

スイッチング・コンバータでは、スイッチに加え、インダクタやコンデンサなどの損失が少ない部品を使用して電圧を安定化させます。通常、これらの部品はスイッチング・トランジスタによって充電および放電されます。ここでは、チャージ・ポンプ・コンバータとインダクタ・コンバータの 2 種類のスイッチング・コンバータについて説明します。

インダクタ・コンバータは、マイクロコントローラの設計において高い効率で超低消費電力を実現するために最も一般的に使用されるコンバータです。この効率と幅広いゲイン範囲は、このアーキテクチャの魅力です。

チャージ・ポンプまたはスイッチド・キャパシタ・コンバータは、誘導性コンバータに代わるものです。チャージ・ポンプ・プロセスは、スイッチを接続または切断することで、キャパシタの充電および放電を実行します。このプロセスは、インダクタなしで実現するので、スペースとコストを節約できます。

ADP2503/ADP2504 は、安定化された出力電圧よりも大きい、小さい、または等しい入力電圧で動作させることができます。ADM660/ADM8660 は、低出力電流（最大 50 mA）で 90 % を超える電流を実現できるチャージ・ポンプ電圧コンバータです。図 48 および図 49 に、ADP2503/ADP2504 デバイスおよび ADM660/ADM8660 デバイスの入力電圧と出力電流の効率を示します。

2 つのグラフで観察されるように、チャージ・ポンプ・コンバータは、出力負荷電流の出力形状により、インダクタ・コンバータよりも効率が低くなります。逆に、チャージ・ポンプ・コンバータは、低負荷電流アプリケーションに適切なソリューションです。

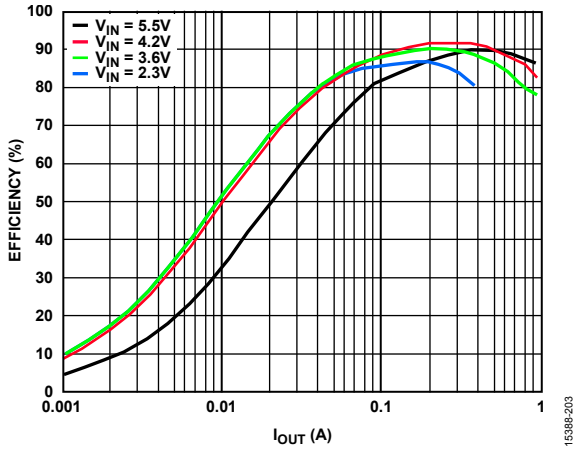


図 48. ADP2503/ADP2504 効率と出力電流（I_{OUT}）の関係

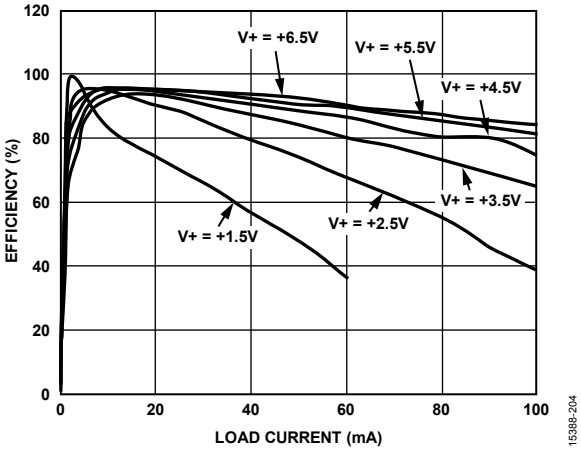


図 49. ADM660/ADM8660 効率と出力電流（IOUT）の関係

ADuCM3027/ADuCM3029 マイクロコントローラには、チャージ・ポンプ・コンバータ出力にリニア電圧レギュレータが接続されているので、デジタル・コアとメモリ電源を調整して安定化できます。さらに、デバイスには、リニア電圧レギュレータのみを使用して、電圧を削減して調整するチャージ・ポンプ・コンバータのバイパス機能があります。この機能を使用して、従来のソリューションを使用するか、効率を向上させ、電力を節約するために、2 つの 0.1 µF キャパシタを追加してでもチャージ・ポンプ・ブロックを使用するか選択できます。

通常、インダクタ・コンバータでは、チャージ・ポンプ・コンバータの設計で不便なリニア電圧レギュレータを出力で使用する必要ありません。ただし、市場で入手できるマイクロコントローラの一部では、インダクタ・コンバータにリニア電圧レギュレータを含むインダクタ・コンバータ・ソリューションを使用しています。このデメリットにもかかわらず、チャージ・ポンプ・コンバータには、検討に値する次のようなメリットがあります。

- 面積
- 厚み
- 価格
- 設計のシンプルさ
- 使いやすさ
- 電磁場干渉（EMI）

次のセクションでは、インダクタ・コンバータと比較したチャージ・ポンプ・コンバータのメリットについて説明します。

キャパシタとインダクタ・コンバータの比較面積

チャージ・ポンプ・コンバータでは、DC/DC 電圧を実現するためのインダクタは不要ですが、インダクタ・コンバータでは、このタスクを実行するためにインダクタ、キャパシタ、抵抗などの部品が必要です。この事実により、回路基板（PCB）が小さくなり、面積とコストを削減できます。

ここでは、ADuCM3027/ADuCM3029 チャージ・ポンプ・コンバータを使用するために必要な部品を、市販されている超低消費電力マイクロコントローラの他の同様なソリューションで使用されるインダクタ・コンバータと比較します。これらのアーキテクチャで必要な部品は、対応するデータシートに記載されています。

表 16 および表 17 に示す 2 つの部品表は、チャージ・ポンプ・コンバータでは、誘導性コンバータよりも面積が小さいことを示します。キャパシタの寸法（長さ × 幅 × 厚さ）は 0.6 mm × 0.3 mm × 0.3 mm です。インダクタの寸法は 1 mm × 0.5 mm × 0.55 mm です。キャパシタもインダクタも、0603 パッケージに収納されています。

表 16. ADuCM3027/ADuCM3029 のチャージ・ポンプ・コンバータ部品の面積

Component	Value (µF)	Quantity	Area (mm^2)
Capacitor	0.1	2	0.36
Capacitor	0.47	1	0.18
Total			0.52

表 17. インダクタ・コンバータ部品の面積

Component	Value	Quantity	Area (mm^2)
Capacitor	1 µF	2	0.36
Inductor	2.2 µH	2	1
Total			1.36

予想どおり、チャージ・ポンプ・コンバータは誘導性コンバータよりも採用する部品数が少なく、小さいので、このタイプのコンバータの面積は小さくなります。インダクタ・コンバータは、チャージ・ポンプ・コンバータの 2 倍以上の面積を使用します。

インダクタ・コンバータ・ソリューションでは、全体の厚さはインダクタに依存します。インダクタ（0.55 mm）はキャパシタ（0.3 mm）よりも厚いことが理由です。

インダクタの寸法はキャパシタの寸法よりも大きいので、インダクタは設計全体の面積を決定します。

顕微鏡で見ると、寸法の差が明らかになります。図 50 および図 51 では、2.2 µH のインダクタと 0.1 µF のキャパシタの寸法を比較しています。

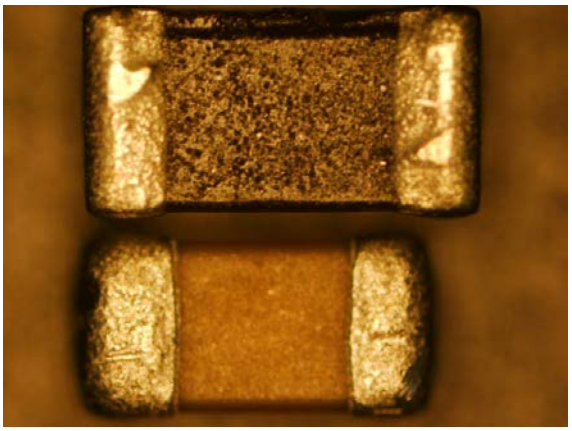


図 50. 0.1 µF のキャパシタ（下）と 2.2 µH のインダクタ（上）の長さの比較

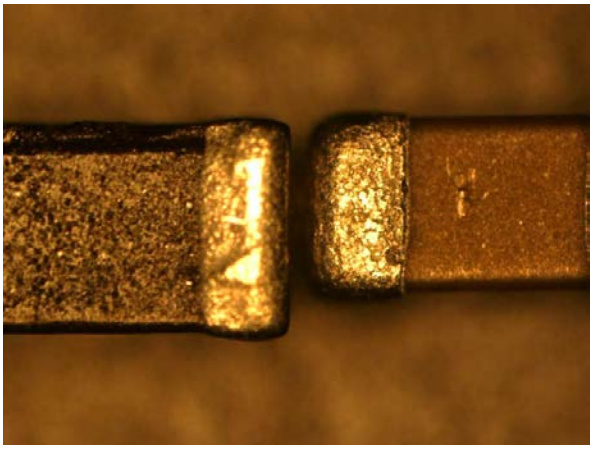


図 51.0.1 µF のキャパシタ（右）と 2.2 µH のインダクタ（下）の幅の比較

価格

インダクタ・コンバータで作業する場合に考慮する問題には、部品の多さと、部品のコストがあります。キャパシタの容量は大きくなり、インダクタの価格は、キャパシタの価格よりも大幅に高くなります。

表 18 と表 19 に、ADuCM3027/ADuCM3029 ソリューションとインダクタ・ベースのソリューションの価格を示します。

表 18. ADuCM3027/ADuCM3029 のチャージ・ポンプ・コンバータ部品の価格

Component	Value (µF)	Quantity	Price ($)
Capacitor	0.1	2	0.1046
Capacitor	0.47	1	0.459
Total			0.5636

表 19. インダクタ・コンバータ部品の価格

Component	Value	Quantity	Price ($)
Capacitor	1 µF	2	1.068
Inductor	2.2 µH	2	0.538
Total			1.606

両方の材料リストを検討すると、各部品の価格の差は約 1 ドルです。この差は小さいように見えますが、製品が多数になり、コストを乗算すると影響が大きくなり、価格の差は増大します。インダクタ内の設計では、部品が増えると関連する面積が広くなるからです。小さな PCB 設計が可能になれば、コストを減らすことができます。

価格は、市販されている最も安価で、小さな部品を基準にしています。このインダクタの品質が十分出ない場合、消費電力が増え、効率が低下します。そのため、安いインダクタを使用すると、設計者はインダクタベースの性能を利用できなくなります。逆に、この制約を避けることは、インダクタ・コンバータの価格がさらに上がることを意味します。

チャージ・ポンプ・コンバータと従来のインダクタ・ベースのソリューションを比較すると、面積と価格のメリットは明白です。

効率

面積を減らすと同時に、組み込みコンポーネントを採用して集積度を向上させることもできます。これは、インダクタではなく、チャージ・ポンプ・コンバータの効率を高める場合に適したシナリオです。

チャージ・ポンプ・コンバータは、インダクタ・コンバータよりも効率が悪いとみなされます。これは、入力電圧と負荷が変化する場合に当てはまります。

低負荷が管理される超低消費電力アプリケーションでチャージ・ポンプ・コンバータを使用する場合、負荷の変動は問題ではありません。最適な効率で、低負荷電流を実現します。チャージ・ポンプ・コンバータは、低負荷で適切な効率を発揮します。これは、超低消費電力アプリケーションで統合を適用すると、簡単に実現できます。必要な負荷が小さくなると、チャージ・ポンプ・コンバータで統合と効率が向上します。

チャージ・ポンプ・コンバータを適切に構成すると、入出力電圧比（V_{IN}/V_{OUT}）に従ってゲインを変更できます。このプロセスでは、効率を向上させてインダクタ・コンバータと同じ性能を実現できます。

インダクタ・ベースのソリューションは、パルス幅変調（PWM）を使用して、デューティ・サイクルを調整して適切なゲインを達成します。この安定化により、効率は高くなりますが、負荷が下がると効率も減少します。PWM の最中はノイズの影響も表れます。この結果、インダクタが高額になり、コストが増えます。

インダクタ・コンバータの設計で統合が必要な場合、組み込みインダクタが機能するには、高周波スイッチングが必要です。高周波スイッチングでは、消費電力が増え、効率が失われ、望ましくない結果が得られます。

電磁場干渉（EMI）

チャージ・ポンプ・コンバータは、電磁場干渉の影響を受けません。インダクタの磁気放射とは異なり、このような放射は懸念事項にはなりません。

EMI は、インダクタの使用時に悪影響を与えます。スイッチング・インダクタが放射アンテナと同様に動作する場合は、特に悪影響が大きくなります。設計や基板のその他の部分で、予想外の干渉が発生する可能性があります。さらに、無線周波タスクが実行される場合は、感度の問題があります。

インダクタ・コンバータでは、効率を向上させるために負荷を抑える必要がある場合、PWM をパルス周波数変調（PFM）に置き換えます。PWM が実行されると、スイッチング・ノイズと出力電圧のリップルは、コンバータの出力電圧のシンプルなフィルタで簡単に向上します。ただし、PFM の手法では周波数帯が可変になり、フィルタの共振周波数が生成されることがあります。さらに、この広い周波数スペクトラムにより、EMI が高くなります。

まとめ

インダクタ・コンバータは、超低消費電力アプリケーションを考えると、多くの意味で適切なソリューションではありません。インダクタ・コンバータは、負荷が減ると効率が下がり、面積が広く、部品が高額になり、コストも高くなることがあります。インダクタの放射には、低負荷で EMI が発生するなどの問題があります。

表 20 に、3 種類の DC/DC コンバータのメリットとデメリットを示します。対象のアプリケーションに最適なタイプのコンバータを評価します。

結論を述べると、チャージ・ポンプ・コンバータは、超低消費電力アプリケーションにとって最適なソリューションです。他のソリューションでは、負荷が下がると効率が悪くなりますが、チャージ・ポンプ・コンバータでは、他の状況でも効率が向上します。

表 20. DC/DC コンバータの各タイプの比較

Type of Converter	Advantages	Disadvantages
LDO	Simple Low cost No inductor No EMI	Less efficient than charge pumps and inductives
Charge Pump	Simple Low cost No inductor Cheaper than inductive Low loads Small area More efficient than LDO Low EMI	Less efficient than inductive at high loads EMI (less than inductive)
Inductive	Most efficient (not in low loads)	EMI Area Cost Poor efficiency at low loads Complex design

UART ソフトウェア・フロー制御

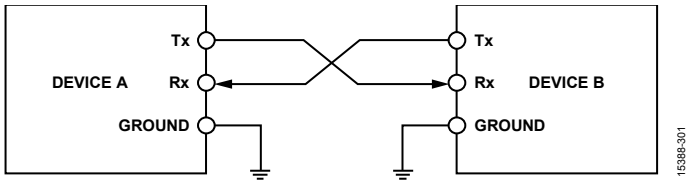


図 52. ソフトウェア・フロー制御のブロック図

フロー制御は、2 つのノード間のデータ転送を管理するプロセスで、速度の速いトランスミッタが速度の遅いレシーバーに過剰な負荷をかけることを防ぎます。フロー制御は、レシーバーが転送速度を制御する機構を提供し、受信ノードが転送ノードからのデータの負荷を過剰に受けることを防ぎます。

UART フロー制御は、低速および高速のデバイスが、データを失うことなく UART を使用して通信するための方法です。2 つのユニットが UART 経由で通信できるケースを考えます。トランスミッタ Tx は、長いバイト・ストリームをレシーバー Rx に送信しています。Rx は Tx よりも低速のデバイスで、ある時点で Rx は転送されるデータ速度に対応できなくなります。そのため、Rx は受信できるようになるまで、データの一部または空のバッファを処理する必要があります。Rx は、データを受信する準備が整うまで転送を停止するよう Tx に命令します。この転送を待機する方法をフロー制御と呼びます。

フロー制御では、追加の信号情報をトランスミッタに通知して、転送を停止（一時停止）または開始（再開）します。従来の UART ハードウェア・フロー制御では、送信リクエスト（RTS）と送信クリア（CTS）の 2 つの信号が追加で必要でした。これらの信号のロジック・レベルは、トランスミッタがデータの送信を続行するか、または停止する必要があるか定義します。ソフトウェア・フロー制御では、特殊文字を通常のデータ行で送信して転送を開始または停止するので、使用する信号が少なくなります。

ここでは、ADuCM3027/ADuCM3029 を使用した UART ソフトウェア・フロー制御機構について説明します。

UART フロー制御

ハードウェア・フロー制御

ハードウェア・フロー制御機構は、帯域外信号を使用して、データのフローを制御します。データ信号の他に、RTS と CTS の追加の信号が必要です。これらのフロー制御信号は、2 台のデバイス間で交差結合します。1 台のデバイスの RTS がリモート・デバイスの CTS に接続されます。逆の場合も同様です。

各デバイスは、RTS を使用して新しいデータを受信する準備が整っているか信号で通知し、CTS 信号の読出しでデータを他方のデバイスに送信できるか確認します。デバイスでさらにデータを受信する準備が整っている場合、RTS はアサートされたままです。デバイスは、受信バッファがフルになると、RTS 信号をアサート解除します。

他方のデバイスは、フロー制御信号を遵守し、RTS 信号が再度アサートされるまで転送を一次停止する必要があります。

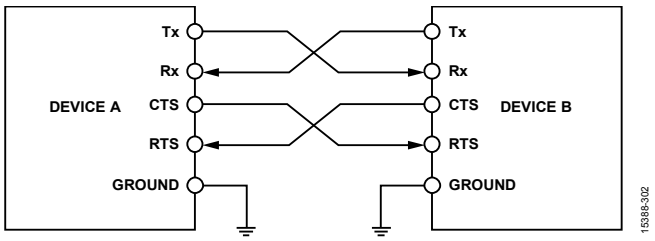


図 53. ハードウェア・フロー制御

フロー制御は双方向なので、両方のデバイスが転送で停止を要求できることを意味します。一方のデバイスが転送の停止を絶対に要求しない場合は（例： デバイスが常に高速でデータを受信できる）、他方のデバイスの CTS 信号はアサートされたロジック・レベルに連結できます。このため、速度の速いデバイスの RTS ピンでその他の機能を実行できます。

XON 信号および XOFF 信号を使用したソフトウェア・フロー制御

ソフトウェア・フロー制御では、追加の帯域外信号は不要です。Rx、Tx、グラウンドの 3 つの信号だけが必要です。ソフトウェア・フロー制御は、特殊な制御フロー文字を使用して実行できます。制御フロー文字は、通常の Tx 行および Rx 行で送信されます。これらの文字は、通常は ASCII コードで、具体的には転送を再開および中断する場合は、それぞれ XON（0x11）と XOFF（0x13）です。

デバイス A が XOFF をデバイス B に送信すると、デバイス B は、デバイス A から XON 文字を受信するまでデバイス A への転送を中止します。データに XON／XOFF 文字が含まれる場合は、XON／XOFF 文字の前にエスケープ文字を挿入します。このケースで使用されるエスケープ文字は、¥ と ASCII 値 92（0x5C）です。このエスケープ文字に遭遇すると、後続の文字は、フロー制御信号ではなく、データ文字とみなされます。データ自体にエスケープ文字が含まれる場合は、データ内に存在する現在のエスケープ文字の前に別のエスケープ文字を置きます。

シーケンス図

デバイス ADuCM3027/ADuCM3029 MCU とピアの間のデータ通信を考えます。MCU が送信し、ピアが受信しています。ピアの速度が MCU よりも遅い場合は、データ転送でピアが過剰な負担を受けます。

この段階では、ピアは XOFF 文字を送信して、データを再度処理できるようになるまで転送を一次停止します。MCU は、ピアから XON 文字を受信するまで待機します。データを受信する準備が整うと、ピアは XON 文字を送信して、転送を再開するよう MCU に命令します。この方法では、ソフトウェア・フロー制御を使用すると、データが失われません。図 54 に、ここで説明した通信のシーケンス図を示します。

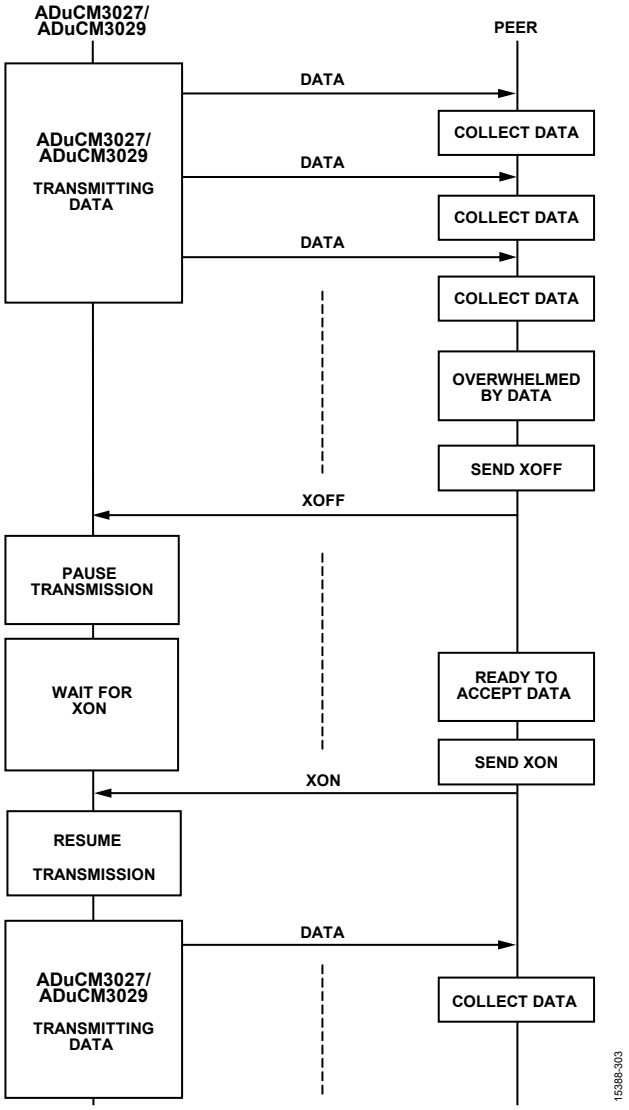


図 54. ソフトウェア・フロー制御のシーケンス図

システムの説明

ADuCM3027/ADuCM3029 を使用した UART ソフトウェア・フロー制御のデモは、EZ-Kit 評価キットを使用して実行されます。端末プログラム（HyperTerminal など）を実行している PC は EZ-Kit UART ポートに接続されます。

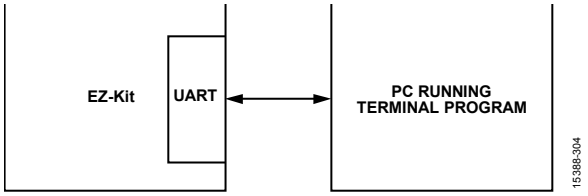


図 55. 接続図

ピア・デバイスからのフロー制御の処理

ADuCM3027/ADuCM3029 BSP には、UART を含むすべてのペリフェラルのドライバが含まれます。UART ドライバ機能の他に、ソフトウェア・フロー制御機構が実装されています。

adi_uart_Write_fc 関数は、XOFF 文字と XON 文字を送信し、UART 割込みサービス・ルーチンは、PC から受信した XON 信号と XOFF 信号を処理します。

adi_uart_Write_fc 関数

adi_uart_Write_fc 関数を使用して書込みが発行されると、グローバルな RECV_XON フラグがチェックされ、ピアがデータを受信できるかどうか認識されます。RECV_XON フラグが false の場合は、XOFF 信号が受信され、ピアがデータを受信できないことを意味するので、失敗が返されます。IRECV_XON フラグが true の場合は、ピアでデータを受信する準備が整っています。この場合、データが送信され、成功が返されます。

フロー制御コードの例

次のコードを使用して、フロー制御を使用したデータ転送を行います。

```
ADI_UART_RESULT adi_uart_Write_fc(
ADI_UART_HANDLE const hDevice, void *const pBuffer, uint32_t nBufSize)
{
    /* Return code */
    ADI_UART_RESULT eResult;
    /* If there is no XOFF received, safe to transmit data */
    if(RECV_XON == true)
        eResult = adi_uart_Write (hDevice, pBuffer, nBufSize);
    /* If XOFF is received, return fail */
    else
        eResult = ADI_UART_FAILED;
    return eResult;
}
```

図 56 に、adi_uart_Write_fc 関数の設計を示します。書込みが発行されると、RECV_XON がチェックされ、フラグが true の場合は、書込みが処理されます。フラグが true でない場合は、失敗が返されます。図 56 の RECV_XON == TRUE ブロックは、RECV_XON フラグのチェックを示します。

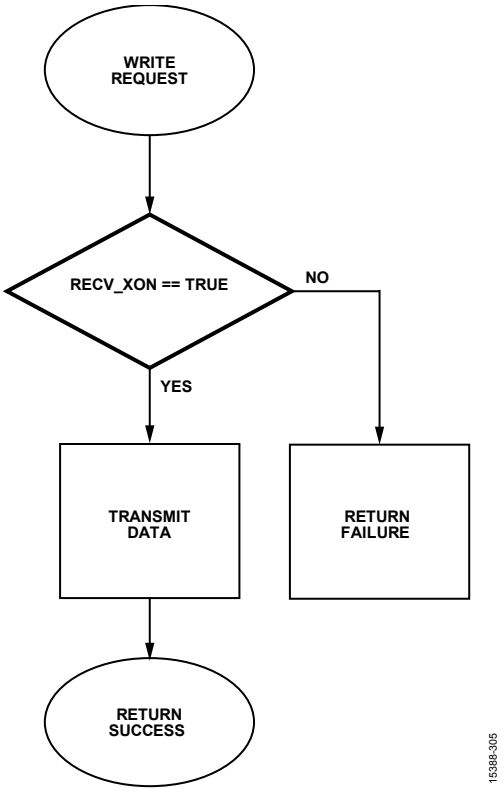


図 56. adi_uart_Write_fc 関数のフローチャート

割込みサービス・ルーチン（ISR）を使用したピアからのプロセス制御信号

UART 経由で受信されたデータは監視され、制御信号とデータのどちらであるかチェックされます。受信されたデータがエスケープ文字の場合は、後続のデータが制御信号ではなくデータとみなされるように、エスケープ・フラグ（bEscFlag）がアサートされます。XOFF 信号または XON 信号が受信されたら、エスケープ・フラグが設定されたかどうかのチェックが行われます。エスケープ・フラグがセットされていない場合、グローバル・フラグ（RECV_XON）が更新されます。

エスケープ・フラグのセットされずに XOFF 信号を受信した場合は、RECV_XON フラグがアサート解除され、XOFF 信号を受信し、データ転送は発生しないことを意味します。同じ方法で、RECV_XON フラグは、エスケープ・フラグがセットされずに XON 信号を受信した場合にアサートされます。

データ処理コードの例

次のコードは、受信したデータを処理します。

```
switch (readVal)
{
  /* If an escape is received */
  case FCESCAPE:
    /* If escape already received,
       consider it as data */
    if(bEscFlag == true)
      bEscFlag = false;
    else
      bEscFlag = true;
    break;
  /* If an XON is received */
  case XON:
    /* If escape received before,
       consider it as data */
    if(bEscFlag == true)
      bEscFlag = false;
      /* Valid control signal,
         update send flag */
    else
      RECV_XON = true;
    break;
  /* If an XOFF is received */
  case XOFF:
    /* If escape received before,
       consider it as data */
    if(bEscFlag == true)
      bEscFlag = false;
    /* Valid control signal,
       update send flag */
    else
      RECV_XON = false;
    break;
  default:
    break;
}
```

図 57 に、ピアからのフロー制御信号を処理するアルゴリズムを示します。受信されたデータがエスケープ文字であるか最初にチェックされます。エスケープ文字の場合、後続のデータは制御信号ではなくデータとみなされます。受信されたデータは、XON と XOFF の制御信号がチェックされ、グローバルな RECV_XON フラグが必要に応じて更新されます。図 57 の灰色のブロックは、RECV_XON フラグの更新を示します。

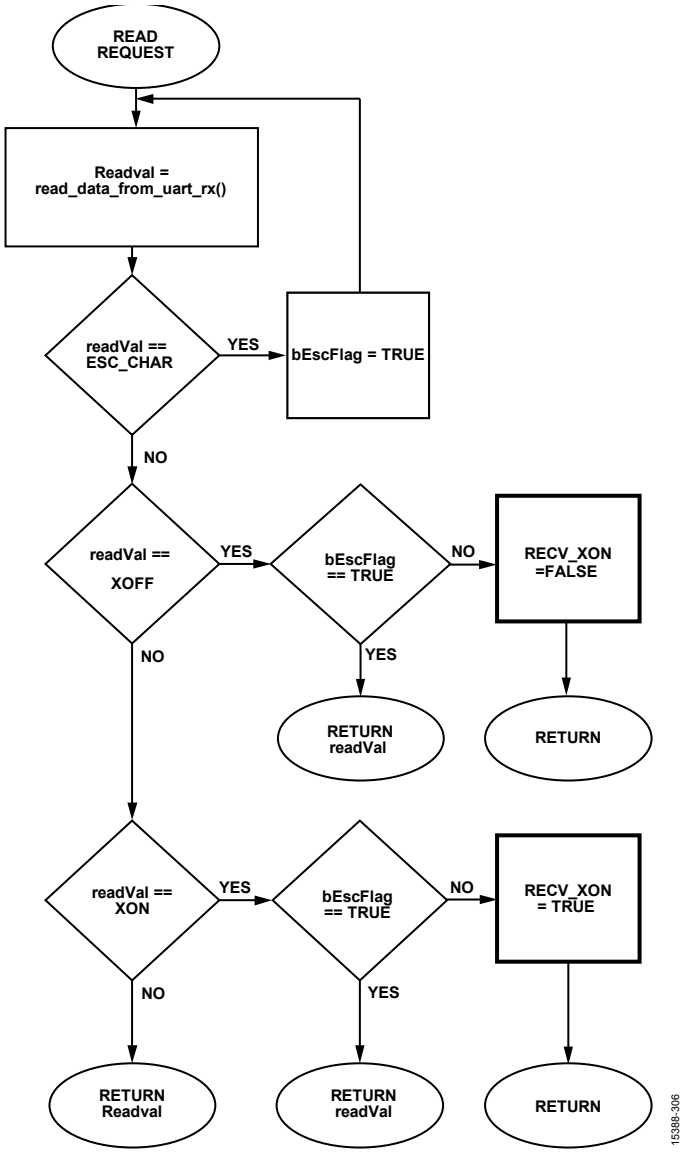


図 57. ISR の制御信号の処理フローチャート

データ・キャプチャ

データ・キャプチャのセットアップでは、MCU が PC に接続され、PC を実行している端末プログラムと通信します。シリアル・ポート・モニタなどの UART スニファは、ボー・レート 9600 で発生するデータ通信を監視します。ここで説明するように、データ・キャプチャは、スニファを使用して実行されます。

ADuCM3027/ADuCM3029 のフロー制御文字の処理

図 58 に、ピアからのフロー制御信号を処理する例を示します。

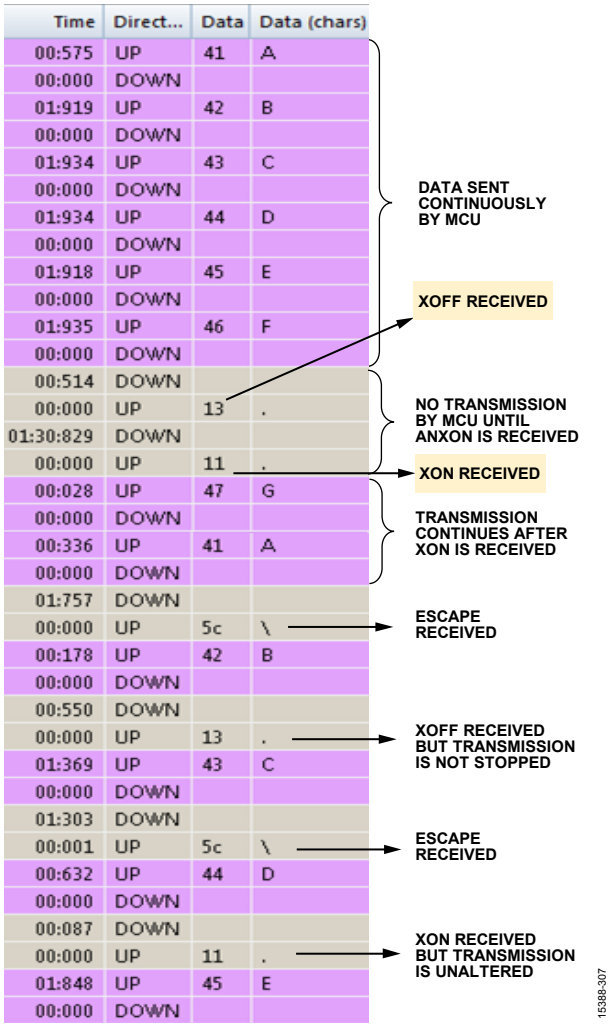


図 58. スニファ・プログラムを使用したデータ・キャプチャ

受信データ・フローの制御

受信データ・フローを制御して ADuCM3027/ADuCM3029 MCU から制御信号を送信する場合は、シンプルな手順を実装します。この場合、MCU はピアと比較して速度が遅くなります。MCU から制御信号を送信する機構はアプリケーションに固有で、制御信号を送信するユーザー独自のアルゴリズムを記述できます。

図 59 に示すように、XOFF 信号は MCU から送信された 5 回の転送の後に毎回送信されます。XON 信号は、短い間隔の後に送信されます。この実装は簡単で、デモの目的のみで説明されます。ユーザーは独自の機構を開発してデータを処理し、XON および XOFF を送信できます。

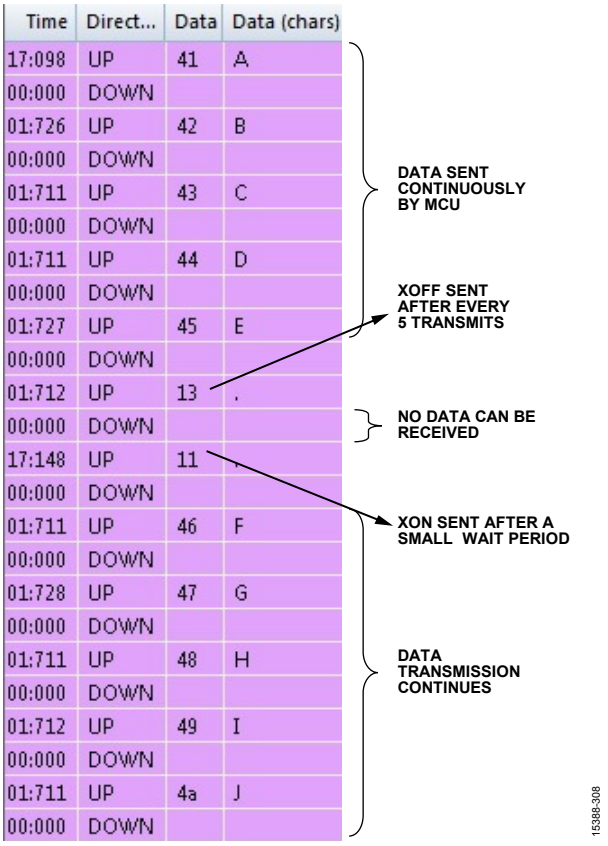


図 59. 制御信号を送信する ADuCM3027/ADuCM3029

UART フロー制御方法

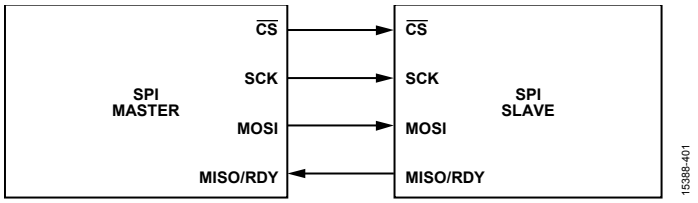


図 60. SPI 信号

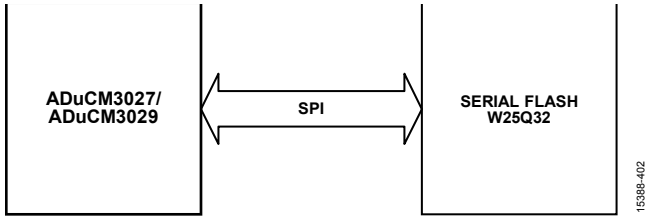


図 61. アプリケーションのブロック図

SPI は、他の SPI 互換デバイスに対する全二重動作を可能にする業界標準の同期シリアル・リンクです。

ADuCM3027/ADuCM3029 SPI には、半二重動作とフロー・制御オプションを提供する拡張動作モードがあります。SPI データ転送は DMA トランザクションを使用して、ADuCM3027/ADuCM3029 コアをスリープ・モードにします。半二重モードで複数のバイトを転送する他に、この消費電力の削減は、ワイヤレス・センサー・ネットワークなどのバッテリ駆動の設計において重要です。

ADuCM3027/ADuCM3029 SPI には次の機能があります。

- 連続読出しモード
- 半二重動作の読出しコマンド・モード
- フロー制御
- $\overline{CS}$ ソフトウェアのオーバーライド
- 3 ピン SPI マスターまたはスレーブ・モードをサポート
- LSB ファースト転送のオプション
- 割込みモード。割込みは、1、2、3、4、5、6、7、8 バイトの後に使用できます。

ここでは、読出しコマンド・モードとフロー制御方法について理解します。これらの方法は、システムでセンサー、シリアル・フラッシュ・デバイス、ADC、RF トランシーバーなどの SPI スレーブを使用する場合に、システムの消費電力を抑えます。

SPI 読出しコマンド・モード

標準の SPI マスターは、シリアル・クロック（SCK）、マスター・アウト、スレーブ・イン（MOSI）、マスター・イン、スレーブ・アウト（MISO）とチップ選択（$\overline{CS}$）のラインを使用して、スレーブと通信します。SCK、MOSI、MISO 信号はスレーブで共有でき、各スレーブは独自の $\overline{CS}$ ラインを持ちます。SPI 転送の間、データは同時に送信および受信されます。シリアル・クロック・ラインは、2 つのシリアル・データ・ラインの情報のシフトとサンプリングを同期します。

通常、SPI 転送は全二重です。転送は、マスターによって制御されます。スレーブからデータを受信するには、マスターがクロックを提供する必要があります。通常はデータを MOSI ラインで送信する必要がある場合に開始されます。

大部分の SPI スレーブでは、通信を成功させるためにマスターが使用する必要があるプロトコルを指定します。プロトコルは、コマンドと同じ程度シンプルで、アドレス（オプション）とデータ（オプション）が続きます。

例えば、書込みコマンドは単方向で、マスターがコマンド、アドレス（オプション）、データをスレーブに書き込みます。

読出しコマンドでは、マスターがコマンドとアドレス（オプション）を送信し、スレーブからデータの読出しを実行する必要があります。データが複数バイトの場合、マスターのソフトウェアが MOSI にダミー・データを書き込む必要があります。これにより、クロックの動作を継続してすべてのデータ・バイトを正常に読み出すことができます。

ただし、一部の SPI スレーブでは、MOSI の読出しコマンドの転送の後に、1 つの $\overline{CS}$ トランザクションで MISO で読出しを実行します。この条件の例を図 62 に示します。

ADuCM3027/ADuCM3029 は、このような半二重動作をサポートする読出しコマンド・モードを提供します。読出しコマンド・モードは、ソフトウェアの負荷を減らし、結果としてコア実行サイクルを減らすのに役立ちます。このモードでは、トランザクションで送信するバイト数と受信するバイト数を指定する必要があります。また、MOSI で転送が進行中の場合は、MISO のデータを無視するかどうか指定することもできます。

データ読出しコマンド・モードを使用すると、1 バイトを送信して、スレーブからデータ・バイトのセットを受信できます。このモードは、スレーブがセンサーまたは ADC で、測定されたデータまたは処理されたデータのセットを送信する場合に便利です。

アプリケーションのシナリオは、システムの説明のセクションで説明しますが、ADuCM3027/ADuCM3029 は SPI マスターで、シリアル・フラッシュ W25Q32 は SPI スレーブです（図 61 参照）。読出しコマンド・モードは、データのページがフラッシュ・メモリから読出す必要がある場合に便利です。

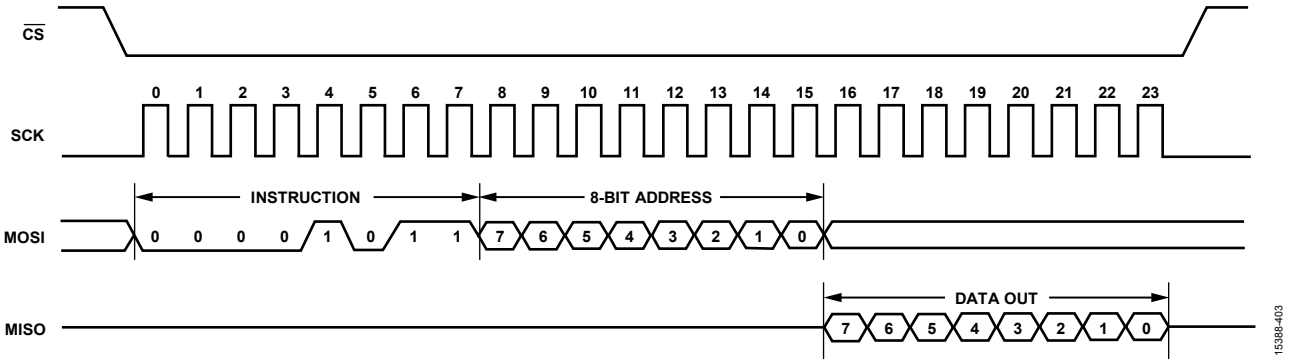


図 62. 読出しコマンド・モード

システムの説明

読出しコマンド・モードを示すために、次のセットアップを使用します。

- ファームウェア - IAR 用の ADuCM3027/ADuCM3029 BSP からのパワーオン・セルフ・テスト・アプリケーション
- ハードウェア - ADuCM3027/ADuCM3029 EZ-Kit ボード

信号をキャプチャするため、SPI ラインにオシロスコープが接続されます。オシロスコープのプロット図 63 ～ 図 67 に、SPI マスターである ADuCM3027/ADuCM3029 と SPI スレーブであるシリアル・フラッシュ W25Q32 間の SPI 転送を示します。

読出しコマンド・モードを使用するトランザクションを決定するのは、ユーザー・アプリケーションです。

読出しコマンド・モードなし

リファレンス・アプリケーションでは、フラッシュ・メモリの 4 KB セクタの消去プロセスで読出しコマンド・モードを使用しません。消去コマンドの転送と個別のチップ選択フレームのアドレスを確認すれば、このモードが存在しないことがわかります（図 63 および図 64）。

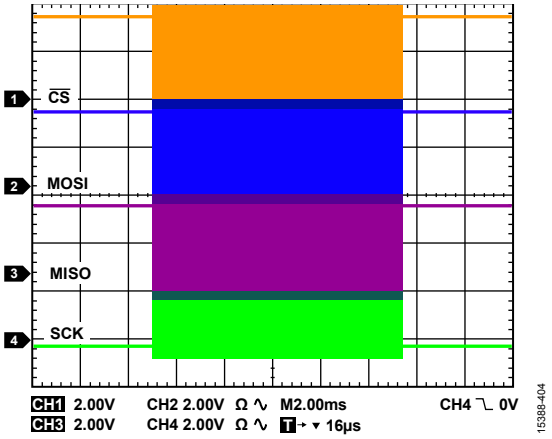


図 63. セクタ消去

図 64 に、消去シーケンス全体の $\overline{CS}$ フレーム・キャプチャを示します。$\overline{CS}$ ラインは、コマンド・バイトの転送の切り替えを示します。

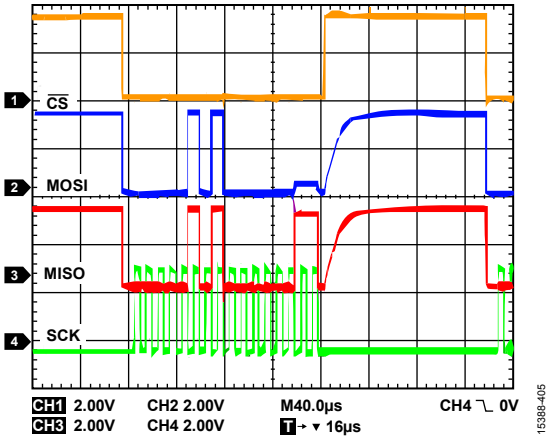


図 64. セクタ消去 — シングル $\overline{CS}$ フレーム・キャプチャ

読出しコマンド・モードあり

図 65 に、外部フラッシュからの 1 ページの読出しを示します。フラッシュの 1 ページのサイズは、256 バイトです。この読出しシーケンスは、読出しコマンド・モードを使用し、ページ全体の読出しが 1 回の $\overline{CS}$ トランザクションで発生します。

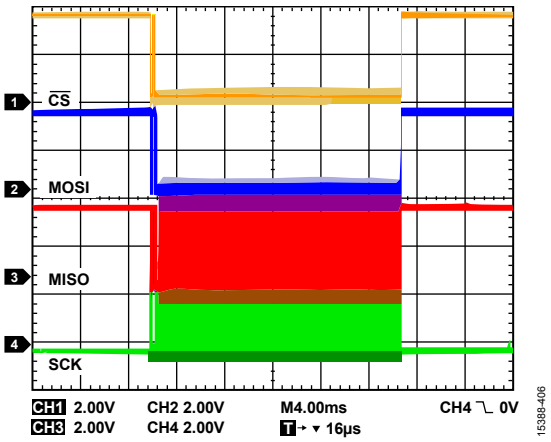


図 65.ページ読出しシーケンス

図 66 に、ADuCM3027/ADuCM3029 がコマンドとアドレス・バイトを転送するページ読出しシーケンスの開始を示します。この転送の後に、シリアル・フラッシュ・メモリからのページ・データが続きます。

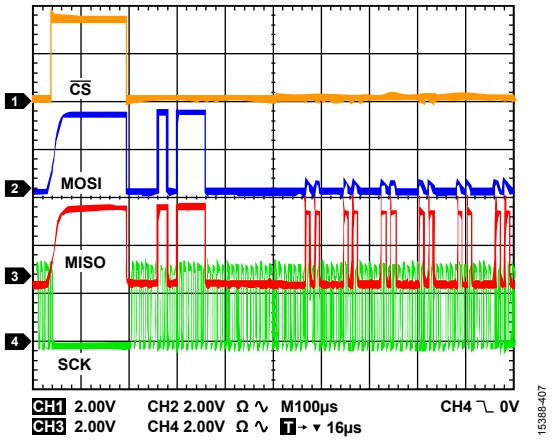


図 66. ページ読出しの開始シーケンス
（コマンドとアドレス・バイト）

図 67 に、1 バイトの読出しを示します。これはページ読出しシーケンスの一部です。

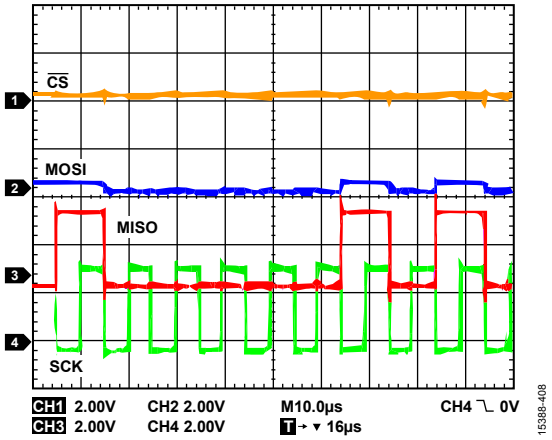


図 67. ページ読出し - 1 データ・バイト

フロー制御モード

フロー制御は、マスターとスレーブ間のデータ・フローの同期に必要です。ADuCM3027 は、差別化機能であるフロー制御を SPI で提供します。読出しコマンド・モードでは、フロー制御を使用して複数のデータ・バイトを受信できます。

フロー制御では、SPI マスターとスレーブ間のデータ転送は、定期的なデータ読出しまたはオンデマンドのデータ読出しに関するアプリケーション要件に基づいて制御されます。

ADuCM3027/ADuCM3029 の SPI マスターは、次のモードのフロー制御をサポートします。

- ピンベースのフロー制御、SPI スレーブで制御。
- タイマーベースのフロー制御、SPI マスターで制御。

フロー制御モードは、次のセクションで詳細に説明します。SPI フロー制御レジスタ（SPI_FLOW_CTL）のモード・フィールドは、フロー制御モードを 3 つのモードのいずれかに構成します。

フロー制御機構は、ADuCM3027/ADuCM3029 が SPI マスターとして構成されている場合に限り使用できます。

ピンベースのフロー制御

専用の RDY ピンの使用

一部の SPI スレーブには、SPI マスター（この場合は、ADuCM3027/ADuCM3029）の SPI_RDY ピンに接続される専用の RDY ピンがあります。SPI_RDY ピンは、各 SPI インスタンスに専用のピン（GPIO の代替機能）です。

例えば、CAT64LC40 シリアル・フラッシュは、専用の RDY ピンを使用して、SPI マスターにデータが使用できるか信号で通知します。

スレーブが専用の RDY ピンをサポートしていない場合、ADuCM3027/ADuCM3029 の RDY ピンは、SPI スレーブの割込みピンに配線できます。スレーブは、RDY ピンを使用して、アクイジションとデータ処理が完了したことを示します。マスターは、このピンのアクティブなレベルが表示されるまで SPI クロックを提供しません。

RDY ピンがアサートされたときに読出しを実行するバイト数を構成できます。この構成を実行するには、SPI フロー制御レジスタ（SPI_FLOW_CTL）の RDBURSTSZ フィールドを設定します。MISO でこのバイトのバーストを受信した後、SPI マスターは、次の RDY ピンのアサーションを待機し、次のバイトのセットを受信します。このプロセスは、SPI カウント・レジスタ（SPI_CNT）のセットのすべてのバイトが受信されるまで繰り返されます。

読出しコマンド・モードを使用すると、最大 16 バイトを転送できます。この転送は、SPI 読出し制御レジスタ（SPI_RD_CTL）の TXBYTES フィールドを使用して構成されます。フロー制御を使用した場合に 1 回のバーストで受信されたバイト数は、SPI 読出し制御レジスタ（SPI_RD_CTL）の RDBURSTSZ フィールドで設定されますが、受信バイトの合計数に上限は課せられません。

MISO ピンの使用

一部の SPI スレーブには、専用の RDY ピンはありませんが、MISO ピンを再利用して MISO で送信する準備が整ったことを SPI マスターに通知するという対策があります。

ADuCM3027/ADuCM3029 SPI マスターは、MISO ラインのアクティブ・レベルの遷移を待機し、これが検出されると、RDBURSTSZ + 1 個のバイトの読出しを実行して、次のアクティブ・レベルが MISO で検出されるまで待機状態に戻ります。

MISO/RDY ピンの極性は、SPI 読出し制御レジスタ（SPI_RD_CTL）を使用して構成できます。

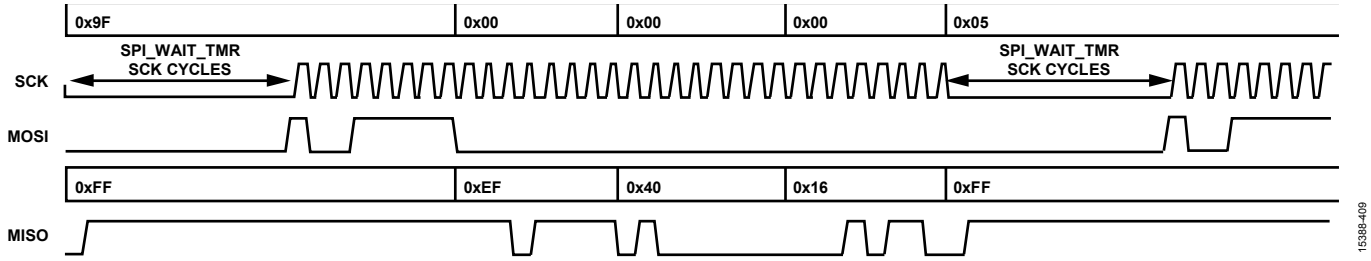


図 68. ソフトウェア・フロー制御とタイマー

タイマーベースのフロー制御

専用ピンがないスレーブから、マスターにデータが使用できるかどうかを通知するには、マイクロコントローラで 16 ビット・タイマーを使用してデータの読出し中に待機状態を導入します。タイマーがトリガすると、マスターはバイトのバースト（RDBURSTSZ + 1）の読出しを実行し、タイマーを再起動します。タイマーは SPI クロック・レート（SCK）で出力され、タイマーのトリガを待機する SCK サイクルの数は、SPI_WAIT_TMR レジスタを使用して設定できます。この動作の例を図 68 に示します。

このスキームを、フロー制御で SCK の待機と駆動に使用する場合は、最後の SCK エッジをサンプリング・エッジにする必要があります。待ち時間が終わったら、SCK 駆動エッジが次のデータ転送を実行します。

システムの説明

ここでは、ハードウェア・フロー制御モードを使用して、フロー制御機能がシステム内の電力節約にどのように役立つか説明します。

これを示すために使用するシステムは、ADuCM3027/ADuCM3029 MCU と SPI 経由で接続されているセンサー（加速度センサー）です。

電力効率が高いシステムを設計するには、処理が不要な場合にコアをスリープ・モードにすることが必要です。このようなシステムでは、センサーの読出しが使用できるようになった後に、コアはウェイク・アップし、センサーからデータを受信して処理します。

ADuCM3027/ADuCM3029 のフロー制御と読出しコマンド・モードは、MCU をさらにオフロードしてこの処理の効率を向上させます。システムは Flexi モードに切り替わります。このモードでは、コアがスリープ状態、SPI ペリフェラルと DMA はアクティブに維持されます。

センサーはデータを測定し、RDY ピンを使用して、データを使用できることを SPI ペリフェラルにストローブ信号で送信します。ADuCM3027/ADuCM3029 MCU の SPI インスタンスごとに専用の SPI_RDY ピン（GPIO の代替機能）があります。

SPI は、MCU をウェイクアップせずに、読出しコマンド・モードを使用してデータ・セットの読出しを実行します。このスキームを効果的に使用するには、センサーが複数バイトのデータ転送に対応できる必要があります。加速度センサーの場合、x、y、z 軸の読出しは、SPI 経由で 6 バイトで送信されます。

DMA を使用すると、CPU の介入なく割り当てられたメモリ・スペースにデータを転送できます。

アプリケーションは毎回測定した後に即座にデータを収集するか、構成されたバイト数がセンサーによって収集された後にスレーブからバッファ済みデータを収集できます。

ユーザー定義のバイトのセットが収集された後に、SPI ペリフェラルまたは DMA が MCU をウェイクアップしてセンサー・データを処理できます。

図 69 に、アクティビティが検出されるたびに加速度センサーから SPI データの読出しを実行するデータのアプリケーション・フロー図を示します。ADXL345 などのセンサーでは、データ対応の割込みを使用して、1 回の SPI トランザクションで x、y、z 軸の値を読み出します。別のセンサーでは、FIFO 構成を実行して、マスターが FIFO を読み出すまでセンサーに多数のサンプルを保存できます。

まとめ

読出しコマンド・モードやフロー制御などの ADuCM3027/ADuCM3029 SPI の各機能により、デバイスは SPI ペリフェラルが MCU をオフロードするバッテリ駆動システムに最適で、データ収集に個別に使用できます。

このデバイスの適合性は、システム設計でセンサーのバッテリ寿命が重要になるワイヤレス・センサー・ネットワークで大きな利点があります。また、オンボード・データ・アクイジションとセンサー・データ分析を使用したスマート・センサー設計の構成要素としても機能します。

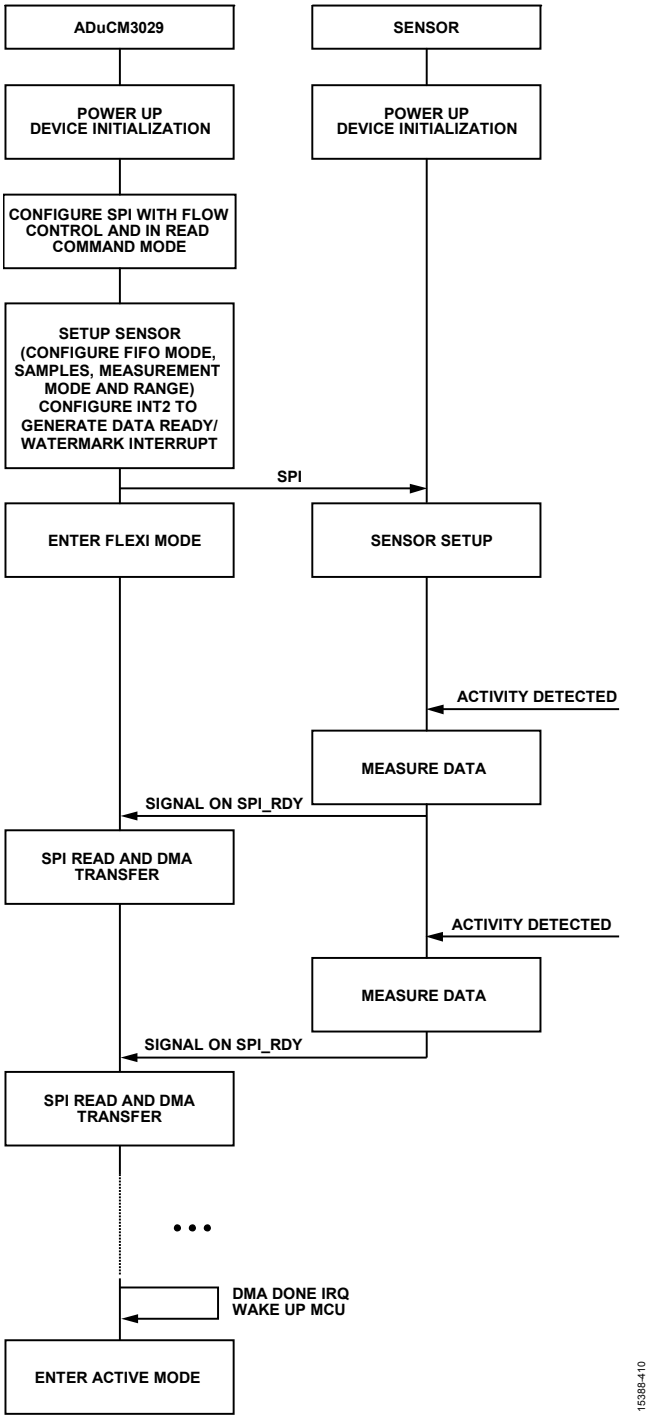


図 69. アプリケーションのフローチャート

SLEEP ON EXIT

ARM® Cortex™-M プロセッサは、電力と効率のバランスが優れた、低消費電力アプリケーションに最適です。これらのプロセッサには、Sleep on exit と呼ばれる機能があり、クロック・サイクルと電力を節約できます。

ADuCM3027/ADuCM3029 プロセッサの MCU サブシステムは、ARM® Cortex™-M3 プロセッサをベースにしています。Sleep on exit 機能は、マイクロコントローラがスリープの場合に割込みハンドラで電力を節約します。

Sleep on exit が有効な場合、ISR が完了すると、プロセッサは直接スリープに移行します。複数の割込みが発生する場合、割込みはネスト構造になります。これらの割込みの実行後、プロセッサは自動的にスリープ・モードに戻ります。

メリット

Sleep on exit 機能は、システムがスリープで割込みを実行するときだけウェイクアップする割込み駆動アプリケーションでメリットがあります。

Sleep on exit が無効の場合、割込みが到達するワークフローでは、命令の実行にかかる時間が長くなります。

Sleep on exit 機能を無効にして割込みを実行する手順は、次のようになります。

1. プロセッサをウェイク・アップします。
2. 必要な情報すべてと電流の状態をスタックにプッシュします。
3. 割込みコードを実行します。
4. スタックの情報をポップし、レジスタを復元します。
5. スリープ・モードに戻ります。

割込みを実行する命令はたくさんあります。そのため、割込み駆動アプリケーションでは、コアはスタックに命令をプッシュした後に再度ポップするので、コンテキストの切り替えに費やされる時間は最適ではありません。

Sleep on exit 機能を有効にすると、プロセスを簡略化できます。プロセッサは、割込みの終了後、即座にスリープに移行します。デバイスは、通常のスレッドに戻らず、割込み構成を維持します。これにより、プッシュとポップのタスクがスタックに含まれるのを避けることができ、不要な命令の実行に必要な電力とクロック・サイクルを削減できます。

図 70 に、Sleep on exit 機能を使用する場合のフローチャートを示します。この機能を使用する手順は、次のとおりです。

1. プログラムが開始します。
2. 割込み待機（WFI）またはイベント待機（WFE）の命令により、スリープ・モードが起動します。
3. システムはスリープ・モードに移行します。
4. デバイスは、中断またはイベントでウェイクアップします。
5. SLEEPONEXIT ビットがセットされている場合、割込みが完了すると、システムは自動的にスリープ・モードに戻ります。

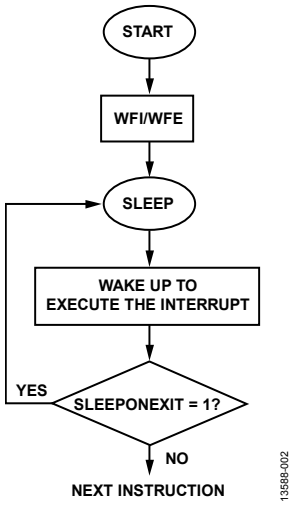


図 70. Sleep on exit のフローチャート

SLEEP ON EXIT 機能を有効にする

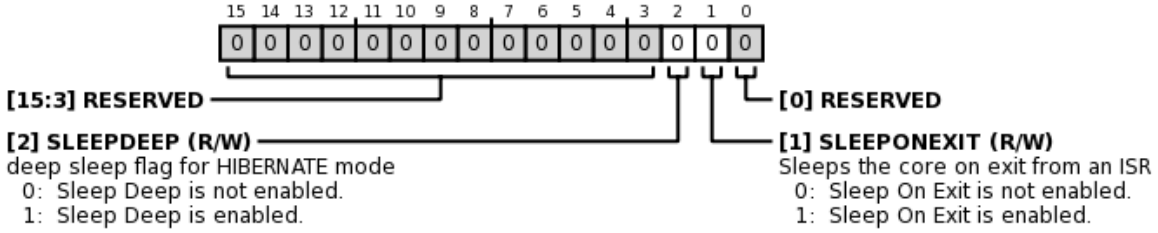
ARM® Cortex™-M ネスト型ベクタ割り込みコントローラ（NVIC）には、システム・コントロール・レジスタと SLEEPONEXIT というビット・フィールドがあります。Sleep on exit 機能を有効にするのに必要なことは、SLEEPONEXIT ビットをセットするだけです。

システム・コントロール・レジスタのアドレスは 0xE000ED10 です。ADuCM3027/ADuCM3029 のシステム制御レジスタ セクションに、ADuCM3027/ADuCM3029 マイクロコントローラのこのレジスタとビット・フィールドを示します。

ADUCM3027/ADUCM3029 のシステム制御レジスタ

INTCON0 - System Control Register

Reset value = 0x0000
Register Address = 0xe000ed10



Bits	Name	Description	Default	Access	Visibility
15:3		Reserved	0		
2	SLEEPDEEP	deep sleep flag for HIBERNATE mode 'b0 - Sleep Deep is not enabled 'b1 - Sleep Deep is enabled	'h0	R/W	Public
1	SLEEPONEXIT	Sleeps the core on exit from an ISR 'b0 - Sleep On Exit is not enabled 'b1 - Sleep On Exit is enabled	'h0	R/W	Public
0		Reserved	0		