参考資料

Application Note

C2000 マイコンでのデバイス リセットなしのライブ ファームウェア アップデート

Baskaran Chidambaram, Sira Rao, Matt Kukucka

概要

このドキュメントでは、TMS320F28003x または TMS320F28P55x など複数のフラッシュ バンクを備えたデバイスで、デバイスリセットなしのライブ ファームウェア アップデート (LFU) について詳しく説明しています。

目次

図の一覧

商標

1 はじめに

サーバー電源のようなアプリケーションでは、ダウンタイムを短縮するためにシステムが継続的に動作することが望まれます。ただし、ファームウェアのアップグレード時には通常、バグ修正、新機能の追加、および/またはパフォーマンスの向上などのために、システムが運用を停止し、ダウンタイムが発生します。これは冗長モジュールで対応できますが、システム全体のコストは増加することになります。ライブ ファームウェア アップデート (LFU) と呼ばれる別の方法により、システムがまだ動作している間にファームウェアをアップデートすることができます。新しいファームウェアへの切り替えは、デバイスをリセットしてもしなくても実行できますが、リセットしないで実行する場合はその方法がより複雑になります。

2 主なイノベーション

提案された設計には、次のような 2 つの重要な技術革新があります。

- 新しいファームウェアの変数を初期化し、古いファームウェアの制御 ISR *と組み合わせて実行されるコンパイラ* LFU 初期化ルーチン。これに先立ち、古いファームウェアの制御 ISR と組み合わせて、新しいファームウェアのダウンロードとインストール (プログラミング) が行われます。これらのステップは時間がかかる場合がありますが、タンデムの実行は、LFU の実行に必要な手順を実行している間、アプリケーションの機能が損なわれないことを意味します。
- 最適なタイミング (アイドル時間の開始時) で新しいファームウェアの切り替え、非常に短い期間 (100 CPU クロック サイクル未満) の割り込みが無効になりますが、これは、マイコンでの *ハードウェア LFU サポート* (割り込みベクタと機能ポインタの交換) によって実現できます。このステップは非常に短く、これを前のステップから切り離すことで、新しいファームウェアを非常に迅速に起動できます。

3 LFU のビルディング ブロック

LFU の設計は、次のようないくつかのビルディング ブロックで構成されています。

- LFU コマンドを発行するデスクトップ ホスト アプリケーション
- ホストと通信し、LFU を実装するためのターゲット デバイスのフラッシュ上の LFU ブートローダ
- ホストをターゲットに接続する通信ペリフェラル (SCI または CAN など)
- ダウンロードしてアクティブ化する LFU 対応アプリケーション
- LFU サポート付きコンパイラ
- LFU 関連のハードウェア サポートを備えたターゲット マイコン(複数の物理的に独立したフラッシュ バンクを持つフラッシュ メモリなど)。デュアル バンクまたは複数のフラッシュ バンクを使用すると、1 つのフラッシュ バンクに常駐したアプリケーション ファームウェアを実行しながら、もう 1 つのフラッシュ バンクを更新できます。

4 提案されたソリューションの詳細

4.1 フラッシュ バンク構成

デュアルバンク フラッシュは、図 4-1 に示すようにパーティション分割されています。各バンクの 2 つのセクタは、フラッシュ バンク選択ロジック、SCI カーネル、フラッシュ API で構成される LFU ブートローダに割り当てられます。これらはファームウェアのアップグレード中に変更されることはありません。バンク 1 には、バンク選択ロジックが含まれていません。バンク内の残りのフラッシュ セクタはアプリケーションに割り当てられます。バンク選択ロジックを使用すると、ブートローダはアプリケーション ファームウェアでプログラムされるフラッシュ バンクの種類と、最新のアプリケーション ファームウェア バージョンを格納しているバンクを決定できます。したがって、バンク選択ロジックがソフトウェア システムのエントリ ポイントです。SCI カーネルは、ホストからの画像の転送と、フラッシュ プログラミング API (フラッシュまたは ROM に常駐) を使用したフラッシュのプログラミングを実装します。セクタ 2 のいくつかの場所は、以下の情報を保存するために予約されています。

- 開始 — フラッシュ消去が完了し、プログラム/検証が開始しようとしていることを示します
- キー – この場所に特定のパターンが含まれている場合、バンク内のファームウェアは有効と見なされます
- ファームウェア リビジョン番号 (REV) – バンク 0 とバンク 1 の間での新しいファームウェア バージョンを決定するためにバンク選択ロジックによって使用されます

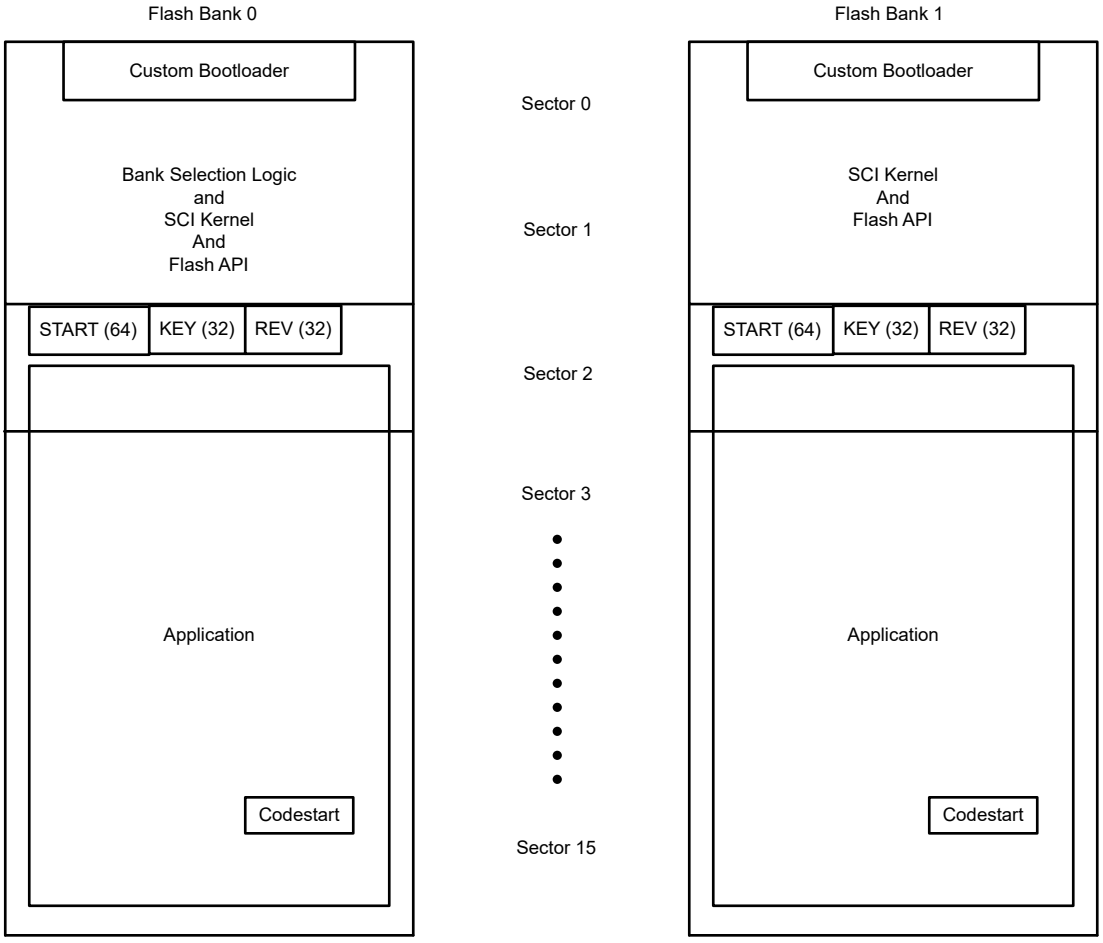

図 4-1. デュアル バンク フラッシュ パーティショニング

4.2 LFU の概念とパフォーマンスに影響を及ぼす要因

LFU 対応ファームウェアを作成するときの主な考慮事項は、LFU 中の動作の継続性と LFU 切り替え時間です。これら 2 つは密接に関連しています。動作の継続性は、状態を維持し、RAM 内の既存の静的変数とグローバル変数をファームウェア バージョン間で同じアドレスに保ち、新しいファームウェアがアクティブになったときにそれらの変数を再初期化しないことによって実現されます。これは、LFU コンパイラのサポートによって可能になります。

新しいファームウェアをアクティブにするには、新しいファームウェアの LFU エントリ ポイントに分岐して、コンパイラの LFU 初期化ルーチンを実行し、新しいイメージの main() 内部に到着して、追加の初期化を行います。このとき、割り込みが短時間ディセーブルになり、割り込みがディセーブルの必要がある初期化 (割り込みベクタの更新、関数ポインタの更新など) が実行されてから、割り込みが再度イネーブルになります。この最後の期間は、LFU 切り替え時間として定義されています。

フラッシュ バンクを入れ替え、どちらのフラッシュ バンクでも固定のアドレス空間にマップし *アクティブ* バンクとして扱えるハードウェア サポートがあれば、LFU は単純化されます。*非アクティブ*なバンクは、別のアドレス空間にマップされ、更新されるバンクです。C2000 マイコンは現在のところフラッシュ バンクの入れ替えをサポートしていないため、アクティブ バンクと非アクティブ バンクを追跡し、リンカ コマンド ファイルを使用して、特定のバンク向けのアプリケーション ファームウェアを作成する必要があります。

関数ポインタと割り込みベクタは、main() 内で再初期化する必要があります。これは、フラッシュ バンク間で位置が異なるためです。C2000 マイコンは多数の割り込みベクタ (通常は 192) をサポートしているため、それらすべてを再初期化するのは実用的ではありません。通常、これらのベクタは少数しか使われず、残りはデフォルトのベクタに割り当てられます。LFU 固有のハードウェア機能 (割り込みベクタの入れ替え、RAM ブロックの入れ替え) により、LFU の切り替え時間を *大幅に*短縮できます。

4.3 LFU 用ハードウェア サポート

4.3.1 複数のフラッシュ バンク

古いファームウェアから新しいファームウェアへの制御のシームレスな転送を可能にするには、マルチバンクフラッシュをサポートすることが重要な機能です。デバイスで使用されるフラッシュテクノロジーでは、フラッシュ バンクへの同時読み取りと書き込みは許可されていないため、このモデルでは 1 つのバンクでファームウェアを実行し、他のバンクをプログラムできます。

4.3.2 割り込みベクタ テーブル スワップ

最適化の機会を評価するために切り替え時間の複数の要素を分析し、割り込みベクタ マップ エントリの更新が切り替え時間に影響を与える主要な要因の 1 つであることが判明しました。更新されるベクトルの数は、いくつかのテーブル全体 (192 のベクトル) に異なります。1 つのエントリの更新には約 5 サイクルかかることがあり、その結果、ベクタテーブルの更新には最大 960 サイクル (200MHz で 4.8us) を要する可能性があります。

切り替え時間を短縮するために、シャドウ ベクタ メモリと、それをアクティブ ベクタ メモリと交換する機能が実装されています。切り替えコードは、アプリケーションの実行中にシャドウ ベクタ メモリを更新できます。ベクタ メモリが更新されると、スワップは 1 クロック サイクルで完了します。どちらのメモリもピンポン方式で使用して、連続するソフトウェアのアップグレードが可能です。

図 4-2 に、割り込みベクタ スワップの代表的な実装を示します。図 4-2(a) はスワップ前の構成、図 4-2(b) はスワップ後の構成です。ベクトル メモリ全体は 2 つのブロックに分かれています。つまり、ブロック A はアドレス 0x0000_0D00 から 0x0000_0EFF まで、ブロック B はアドレス 0x0100_0900 から 0x0100_0AFF までです。ブロック A はアクティブ ベクタ テーブルを保持し、ブロック B はシャドウ ベクタ テーブルを保持します。LFU の間、シャドウ メモリ エントリは切り替え前に更新され、切り替え中にスワップが実行されます。これにより、切り替え時間が最大 960 サイクルから 1 サイクルに短縮されます。

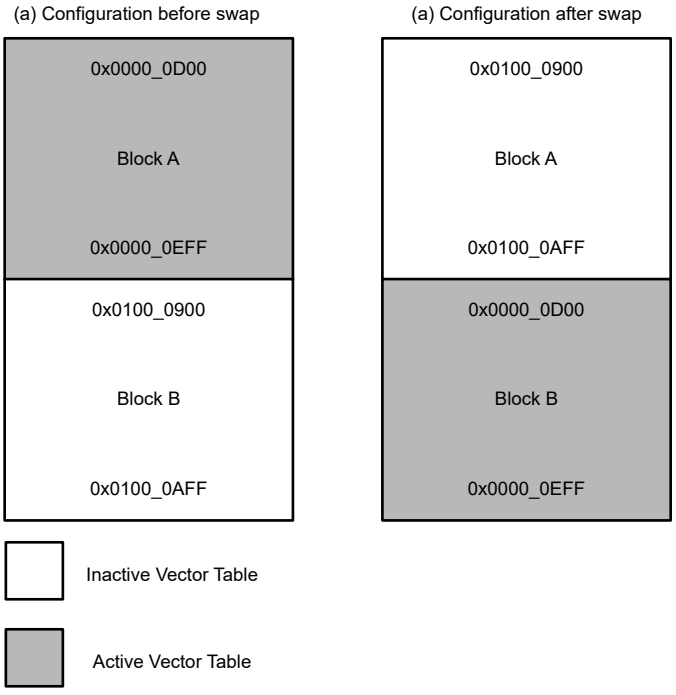

図 4-2. 割り込みベクタ スワップ

4.3.3 RAM ブロック スワップ

ベクタ テーブル スワップと同様に、図 4-3 に示すように、物理 RAM メモリ ブロックをスワップできます。

物理メモリ ブロック 1 に現在のファームウェアの機能ポインタが含まれている場合、LFU を切り替える前に、ブロック 2 の物理メモリ内の同じ相対位置に新しいファームウェアの機能ポインタを実装できます。LFU の切り替え中は、必要なユーザー アプリケーション コードによって、わずか 1 CPU クロック サイクルで単純なスワップ動作が開始されます。これにより、ユーザー アプリケーション コードで LS0 内の機能ポインタを維持しながら、LS0 のアドレス範囲にマッピングされた 2 つの異なる物理ブロックを持つことができます。スワップ後、以前 Block1 アドレス空間にマップされていた物理 RAM ブロックが Block0 アドレス空間にマップされ、その逆も同様で、新しいファームウェアでシームレスな機能ポインタ アクセスが可能になります。

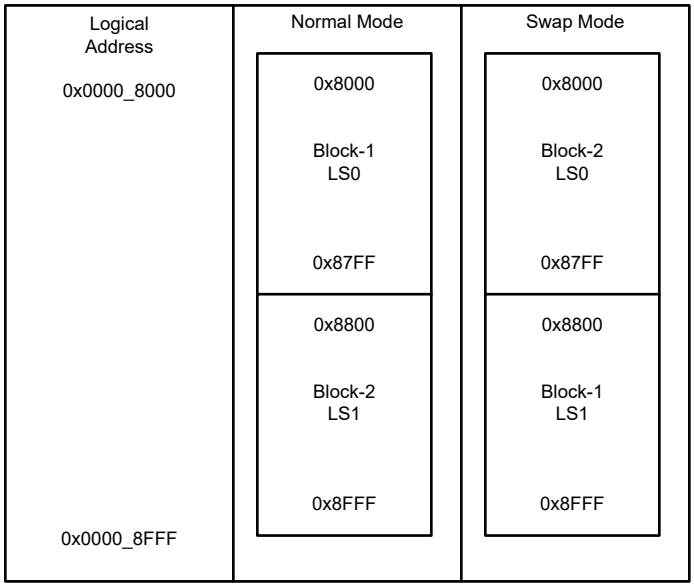

図 4-3. RAM ブロック スワップ

4.3.4 ハードウェア レジスタ フラグ

実行は、デバイスのリセット後、main() への C 初期化ルーチン呼び出しによって、または LFU 中に、LFU コンパイラ初期化ルーチンを呼び出す LFU エントリ ポイントから main() に到達することができます。前者の場合、main() でデバイス固有の初期化が発生する必要がありますが、LFU の場合は発生しません。これら 2 つを区別するため、LFU が現在アクティブかどうかを示すレジスタ内のハードウェア フラグがサポートされています。

4.4 LFU コンパイラのサポート

コンパイラは、LFU を次のようにサポートしています。

1. LFU 初期化ルーチン。これは __TI_auto_init_warm という名前です
2. 変数の LFU 属性
3. LFU モード

コンパイラは、新しいファームウェア実行可能ファイルをビルドする際に、参照イメージとして古いファームウェア実行可能ファイルを提供される必要があります。これにより、コンパイラは共通の変数とそれらの場所を識別し、新しい変数と削除された変数も識別できます。

コンパイラは、*preserve* と *update* という 2 つの新しい LFU 属性を変数に定義します。「preserve」は、ファームウェア バージョン間で共通の変数のアドレスを維持するために使用されます。「update」は新しい変数を示します。これらの変数は、コンパイラが制約なしでアドレスを割り当てでき、コンパイラの LFU 初期化ルーチン (__TI_auto_init_warm()) 中に初期化されます。これらの属性の使用例を次に示します。

float32_t __attribute__((preserve)) BUCK_update_test_variable1_cpu;

float32_t __attribute__((update)) BUCK_update_test_variable2_cpu;

上述の割り当てにより生成されるメモリ マップ ファイルには、preserve 変数に対応する「.TI.bound」セクションと、すべての update 変数が収集される単一の「.TI.update」セクションが含まれています。

アプリケーション開発者は、作業を簡単にするため、さまざまな **LFU モード**を利用できます。デフォルト モードは *preserve* と呼ばれます (上述の対応する変数属性と混同しないでください)。このモードには、次のプロパティがあります。

- 参照 (*古い*) イメージを指定すると、共通変数を *preserve* として指定する必要がなくなります。これは共通変数のデフォルト属性で、これらの変数はコンパイラの LFU 初期化ルーチンで初期化されません。その結果、変数の状態が維持されます。
- 属性が指定されていない、新しい変数にはアドレスが割り当てられますが、これらの変数はコンパイラの LFU 初期化ルーチンで初期化されません。コンパイラの LFU 初期化ルーチンは、*update* 属性で宣言されたときのみ変数を初期化します。

4.5 アプリケーションの LFU のフロー

LFU に関連する詳細手順を以下に示します。

1.*バンク選択ロジックにブートしてアプリケーションを実行します*:デバイスのリセット時に、デフォルトのブートから実行が開始され、フラッシュのエントリ ポイント 0x80000 に移行します。このアドレスにはバンク選択ロジック機能が存在します。この機能は、フラッシュ バンク内の有効なアプリケーションをチェックして、最新バージョンを選択し、対応するファームウェア バージョンのエントリ ポイント (codestart) へ分岐します。このエントリ ポイントは、C ランタイム初期化ルーチンへのゲートウェイで、アプリケーションの main() です。

2.*LFU を開始します*:ユーザーは、ホストで起動される LFU コマンドを使用し、ホスト側からターゲット マイコンに LFU を開始します。

3.*アプリケーションが LFU コマンドを受信します*:(図 4-4 のステップ 1) アプリケーションは、SCI 受信割り込み ISR (CommandLogISR) で LFU コマンドを受信します。

4.*LFU コマンドを解析して LFU ブートローダーに分岐します*:(図 4-4 のステップ 2) 特定のバックグラウンド タスク機能 (Run LFU) が LFU コマンドを解析し、SCI 割り込みをディセーブルにして、固定アドレスにある同じフラッシュ バンクの LFU ブートローダーの LFU 機能に分岐します。

5.*新しいファームウェアをダウンロードしてフラッシュにプログラムします*:(図 4-4 のステップ 3) LFU ブートローダーの LFU 機能は、ホストからアプリケーションのイメージを受信し、そのイメージを非アクティブなフラッシュ バンクにプログラムします。この時点で、古いファームウェアのバックグラウンド タスク機能の実行は停止していますが、アプリケーションに影響を及ぼさないよう、制御 ISR は引き続き古いファームウェアから実行されます。

6.*新しいファームウェアの LFU エントリ ポイントに分岐します*:(図 4-4 のステップ 4) 新しいファームウェアのダウンロードとプログラムが正常に完了すると、カスタム ブートローダは新しいアプリケーション イメージの LFU エントリ ポイント (C_int_LFU) に分岐します。このエントリ ポイントはフラッシュ バンクの固定アドレスにあり、通常のフラッシュ ブートのエントリ ポイント (codestart) とは異なります。

6.*コンパイラの LFU 初期化ルーチンを実行し、新しいファームウェアの main() に分岐します*:LFU エントリ ポイントの関数は、次の処理を行います。

a. コンパイラの LFU 初期化ルーチン (__TI_auto_init_warm) が開始されます。これにより、初期化が必要であると示された変数がすべて初期化されます (図 4-4 のステップ 5)。

b. ハードウェア LFU レジスタで、LFU が進行中であることを示すフラグがセットされます。

c. main() が呼び出されます (図 4-4 のステップ 6)。

7.*切り替えの前に、main() で LFU 固有の初期化を実行します*:(図 4-4 のステップ 7) main() は上述のフラグをチェックし、LFU が進行中かどうかに応じて、初期化を進めます。

フラグがセットされている場合、LFU の初期化機能 (Init_LFU) により、ユーザーが指定したすべてのコードがフラッシュから RAM メモリにコピーされます。次に、非アクティブな割り込みベクタ テーブルを、新しいファームウェアに対応する割り

込みベクタの位置で更新します。同様に、非アクティブな機能ポインタのセットも、新しいファームウェアの機能ポインタの位置に対応して更新されます。

8.*最適な LFU 切り替えポイントを待ちます*: (図 4-4 のステップ 8) ソフトウェア フラグ付きの単純なステート マシンを使用して、制御 ISR の終了とアイドル時間の開始を判定します。これは、制御ループ割り込み間のアイドル時間を最大限に利用できるため、切り替えに最適な時点です (IdentifyIdleTime)。

9.*LFU 切り替えを実行します*: (図 4-4 のステップ 9) すでに新しいファームウェアの main() 内が実行されていても、アプリケーションの機能に影響を与えないように、古い制御ループ ISR は引き続き実行されていることに留意してください。最適な LFU 切り替えポイントが特定されると、LFU 切り替えステップ (ActivateApp) が発生します。まず、グローバル割り込みがディセーブルになります。ハードウェア割り込みベクタ テーブル スワップと RAM ブロック スワップが実行されます。その後で、スタック ポインタが再初期化され、グローバル割り込みが再度イネーブルになります。これで、新しいファームウェアの ISR とバックグラウンド タスク関数の実行が開始され、LFU の切り替えが完了します。グローバル割り込みが短時間ディセーブルになるため、この間に発生した割り込みはラッチされたままとなり、グローバル割り込みが再度イネーブルになったとき CPU に割り込みが発生します。

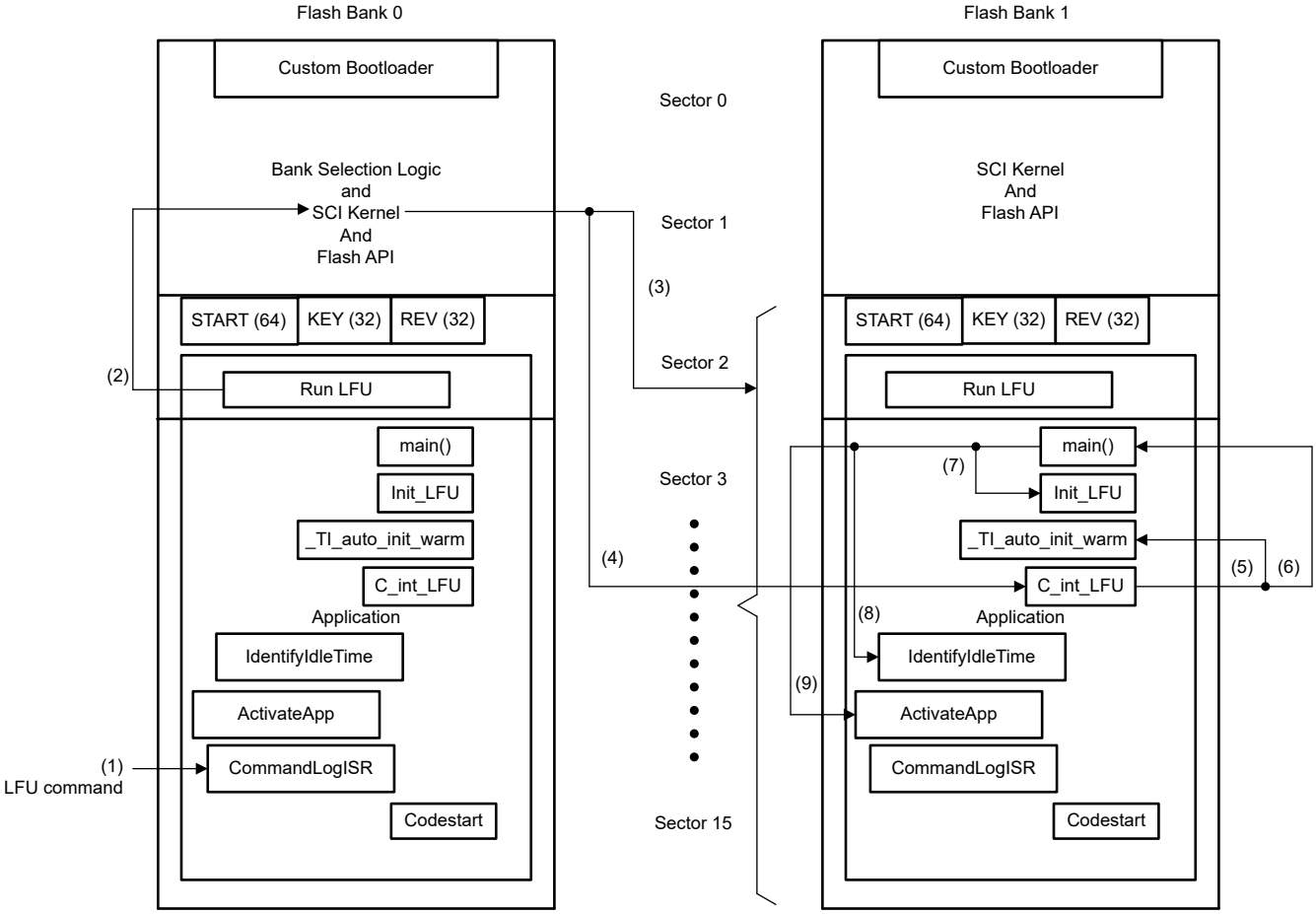

図 4-4. アプリケーションの LFU のフロー

5 結果と結論

図 5-1 では、LFU の切り替え前に発生するステップとは、ファームウェアおよびプログラムフラッシュ バンクのダウンロード、LFU コンパイラ初期化ルーチン (function __TI_auto_init_warm())、main() での LFU 固有の初期化 (関数 init_lfu ()) です。

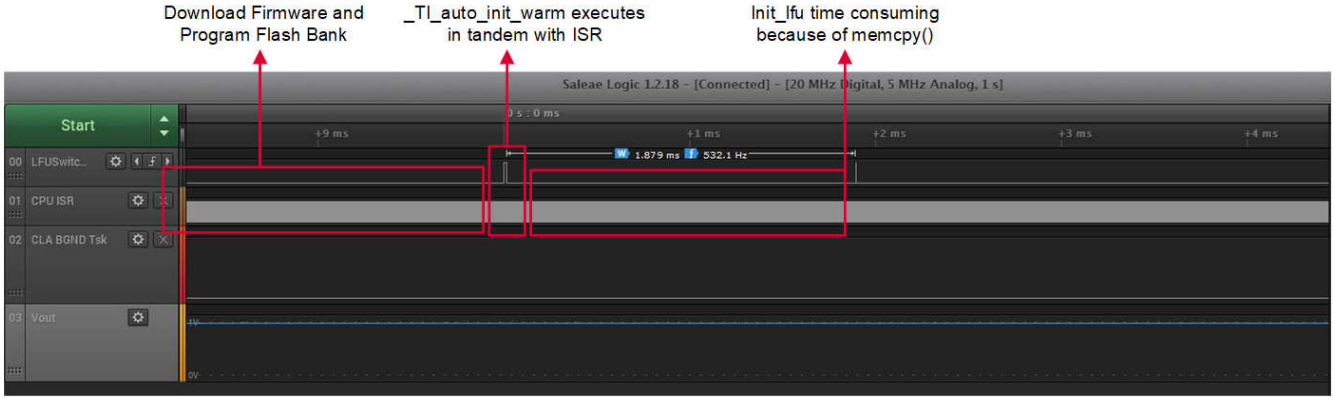

図 5-1. LFU 切り替え前のステップ

図 5-2 に、実際の LFU 切り替えを示します。一番上の波形 (00) は LFU の切り替えを表し、2 番目の波形 (01) は ISR の CPU 負荷 (古いファームウェアで 80%、新しいファームウェアで 40%) を表し、下側の波形 (03) は調整された出力電圧を表します。この切り替えは 0.6µs (または 72 CPU クロック サイクル) で発生し、関数呼び出しと GPIO セット/リセット時間を含みます。切り替えは、ISR の終了後約 40 サイクルで発生します。ISR が終了するのに約 20 サイクルかかり、次に正しい時間が切り替わるのを待機しているループを終了するのに約 15 サイクルかかります。

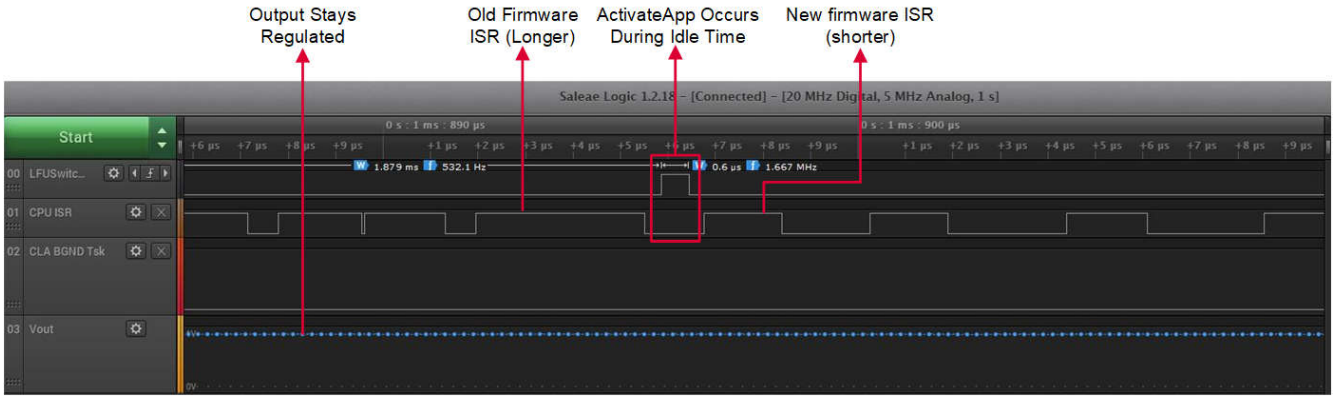

図 5-2. LFU 切り替えステップ

このアプリケーション ノートは、リアルタイム制御アプリケーション向けの LFU、特にダウンタイムなしで動作を必要とする高可用性システムの体系的な実装を示します。新しいアプリケーションの LFU ソフトウェアフロー、ハードウェア LFU サポート、コンパイラ LFU サポートなど、利用可能な LFU ビルディングブロックを使用することで、新しいファームウェアへの切り替えは CPU クロックサイクルの 10 秒以内に完了できます。

6 実装例

F28003x および F28P55x デバイスの LFU ハードウェア機能を示す例は、C2000WARE-DIGITALPOWER-SDK および C2000WARE でリリースされています。これらのソフトウェアを利用すると、開発期間を短縮できます。

F28003x

TIDM-02011 は、F28003x デバイスの LFU ハードウェア サポートを活用し、デバイスをリセットせず LFU を行う方法を示しています。C28x CPU と CLA (Control Law Accelerator) の両方について LFU が示されており、『C2000™ デジタル電源降圧コンバータ ブースタパック』リファレンス デザインに実装されています。『TIDM-02011 設計ガイド』では、C28x CPU または CLA で動作するメイン制御ループを使用した LFU 機能について詳しく説明しています。

TIDA-010062 では、F28003x デバイスで、デバイスをリセットせず LFU を行う方法も示されています。この例では、CLA で実行されるメイン制御ループと、C28x CPU で実行されるバックグラウンド プロセスを持つ LLC ステージでの LFU 機能が示されています。『TIDA-010062 設計ガイド』では、リファレンス デザインの LFU の実装方法について詳しく説明しています。

注

C2000™ デバイスに、デバイスをリセットせず LFU を行う方法を実装する詳細については、『TIDM-02011 設計ガイド』を参照してください。

F28P55x

F28P55x デバイスの LFU ハードウェア機能を示すため、2 つのサンプル アプリケーションが開発されました。C28x CPU、CLA (Control Law Accelerator)、NPU (Neural Processing Unit) に LFU が実装されています。アプリケーションの使用事例の関係で、CPU と CLA には独立のアプリケーションが存在し、CPU と CLA のどちらかが LED の点滅を制御していることに留意してください。

以下の例は、C2000Ware (C2000Ware_x_xx_xx_xx\driverlib\f28p55x\examples\flash) にあります。

1. lfu_cpu_cpu：CPU および NPU でデバイスをリセットせず LFU を行う
2. lfu_cla_npu：CLA および NPU でデバイスをリセットせず LFU を行う

NPU の LFU を示すために、CPU アプリケーションと CLA アプリケーションの両方に LFU_BANK0_NPU/LFU_BANK1_NPU ビルド構成 (*RUNNING_ON_NPU* 定義済みシンボルを追加) が含まれています。アーク フォルト検出モデルと関連するテスト ベクトル (Model Composer から生成される) は、バックグラウンド ループが実行される前に NPU により実行され、理想的な出力と比較されます。アプリケーションに NPU サポートを初めて追加するとき、NPU を正しく初期化するにはリセットが必要なことに留意してください。

7 改訂履歴

Changes from Revision B (September 2022) to Revision C (October 2025)	Page
• F28003x および F28P55x デバイスの *実装例*を追加	9

Changes from Revision A (August 2021) to Revision B (September 2022)	Page
• ドキュメント全体にわたって表、図、相互参照の採番方法を更新。	2
• セクション 2 を追加。	2

重要なお知らせと免責事項

TI は、技術データと信頼性データ (データシートを含みます)、設計リソース (リファレンス デザインを含みます)、アプリケーションや設計に関する各種アドバイス、Web ツール、安全性情報、その他のリソースを、欠陥が存在する可能性のある「現状のまま」提供しており、商品性および特定目的に対する適合性の黙示保証、第三者の知的財産権の非侵害保証を含むいかなる保証も、明示的または黙示的にかかわらず拒否します。

これらのリソースは、TI 製品を使用する設計の経験を積んだ開発者への提供を意図したものです。(1) お客様のアプリケーションに適した TI 製品の選定、(2) お客様のアプリケーションの設計、検証、試験、(3) お客様のアプリケーションに該当する各種規格や、その他のあらゆる安全性、セキュリティ、規制、または他の要件への確実な適合に関する責任を、お客様のみが単独で負うものとします。

上記の各種リソースは、予告なく変更される可能性があります。これらのリソースは、リソースで説明されている TI 製品を使用するアプリケーションの開発の目的でのみ、TI はその使用をお客様に許諾します。これらのリソースに関して、他の目的で複製することや掲載することは禁止されています。TI や第三者の知的財産権のライセンスが付与されている訳ではありません。お客様は、これらのリソースを自身で使用した結果発生するあらゆる申し立て、損害、費用、損失、責任について、TI およびその代理人を完全に補償するものとし、TI は一切の責任を拒否します。

TI の製品は、TI の販売条件 、 TI の総合的な品質ガイドライン 、 ti.com または TI 製品などに関連して提供される他の適用条件に従い提供されます。TI がこれらのリソースを提供することは、適用される TI の保証または他の保証の放棄の拡大や変更を意味するものではありません。 TI がカスタム、またはカスタマー仕様として明示的に指定していない限り、TI の製品は標準的なカタログに掲載される汎用機器です。

お客様がいかなる追加条項または代替条項を提案する場合も、TI はそれらに異議を唱え、拒否します。

最終更新日：2025 年 10 月