TRAVEO™ T2G ファミリのフラッシュアクセス手順

本書について

適用範囲と目的

このアプリケーションノートでは、TRAVEO™ T2G ファミリ MCU におけるフラッシュアクセス手順および使い方の例について説明します。このドキュメントでは、eCT40 フラッシュメモリの特長, ブロックダイヤグラム, セクタ設定, および SROM API を使用したコードフラッシュの書換え例も説明します。

対象者

本書は TRAVEO™ T2G ファミリを使用するすべての人を対象とします。

関連製品ファミリ

- CYT2 シリーズ
- CYT3 シリーズ
- CYT4 シリーズ
- CYT6 シリーズ

目次

目次

1 はじめに

このアプリケーションノートは、TRAVEO™ T2G ファミリマイコンに内蔵しているフラッシュのアクセス手順について説明します。TRAVEO™ T2G ファミリは、コードフラッシュ, ワークフラッシュ, および Supervisory flash を搭載しています。コードフラッシュはユーザプログラムを格納するためのフラッシュメモリであり、ワークフラッシュは重要な不揮発性データまたはパラメータを格納するためのフラッシュメモリです。Supervisory flash はセキュアブート動作を実現するためのフラッシュブートコードまたは公開鍵を格納するために使用されます。「Erase All」や「Program Row」などのフラッシュメモリ書込み操作は、システムコールを介して Arm® Cortex®-M0+の IRQ0 割込み処理の中で実行されます。このアプリケーションノートでは、フラッシュメモリの消去や書込みを行う上でのソフトウェア設定例と使い方を説明します。

このアプリケーションノートで使用される機能と用語を理解するため、architecture technical reference manual (TRM)の「Code Flash」,「Work Flash」, および「Nonvolatile Memory Programming」章を参照してください。

2 フラッシュメモリ概要

2.1 フラッシュメモリの特長

TRAVEO™ T2G ファミリのマイコンは、コードフラッシュ, ワークフラッシュ, および Supervisory flash に分割された eCT フラッシュメモリを搭載します。このアプリケーションノートは、コードフラッシュおよびワークフラッシュ領域のみを対象とします。表 1 に CYT4B シリーズのコードフラッシュおよびワークフラッシュメモリの特長を示します。各デバイスで使用可能なメモリサイズについては、デバイスデータシートを参照してください。

表 1 CYT4B シリーズのフラッシュメモリの概要

特長	コードフラッシュ	ワークフラッシュ
メモリサイズ	最大 8384 KB (8128 KB + 256 KB)	最大 256 KB (192 KB + 64 KB)
書込みサイズ	64 ビット, 256 ビット, 4096 ビット	32 ビット
ECC 機能	64 ビット + 8 ビット 1 ビットエラー訂正, 2 ビットエラー検出 (SECDED)	32 ビット + 7 ビット 1 ビットエラー訂正, 2 ビットエラー検出 (SECDED)
セクタ消去サイズ	32 KB (大セクタ) 8 KB (小セクタ)	2 KB (大セクタ) 128 B (小セクタ)
セキュリティ	対応	対応
シングルバンク/デュアルバンク動作	対応	対応
プログラム実行	対応	非対応
書込み・消去中の読み出し	対応	対応
プログラム・消去回数/ データ保持年数@85°C	1,000 回/20 年	250,000 回/10 年

2.2 ブロックダイヤグラム

eCT フラッシュメモリは、CPU サブシステムの一部です。Arm® Cortex®-M4/-M7_x CPU コアおよび CM0+ CPU コアは、Fast infrastructure および Slow infrastructure を介して eCT フラッシュメモリへのアクセスが可能です。しかし、TRAVEO™ T2G ファミリデバイスの設計では、CM0+コアのみが SROM API の実行により、コードフラッシュおよびワークフラッシュへ書き込めます。したがって、CM4/CM7_x で動作するユーザアプリケーションは、IPC を介してコマンドを送る必要があります。そして、CM0+の IRQ0 ハンドラの中で SROM API が実行されます。

2 フラッシュメモリ概要

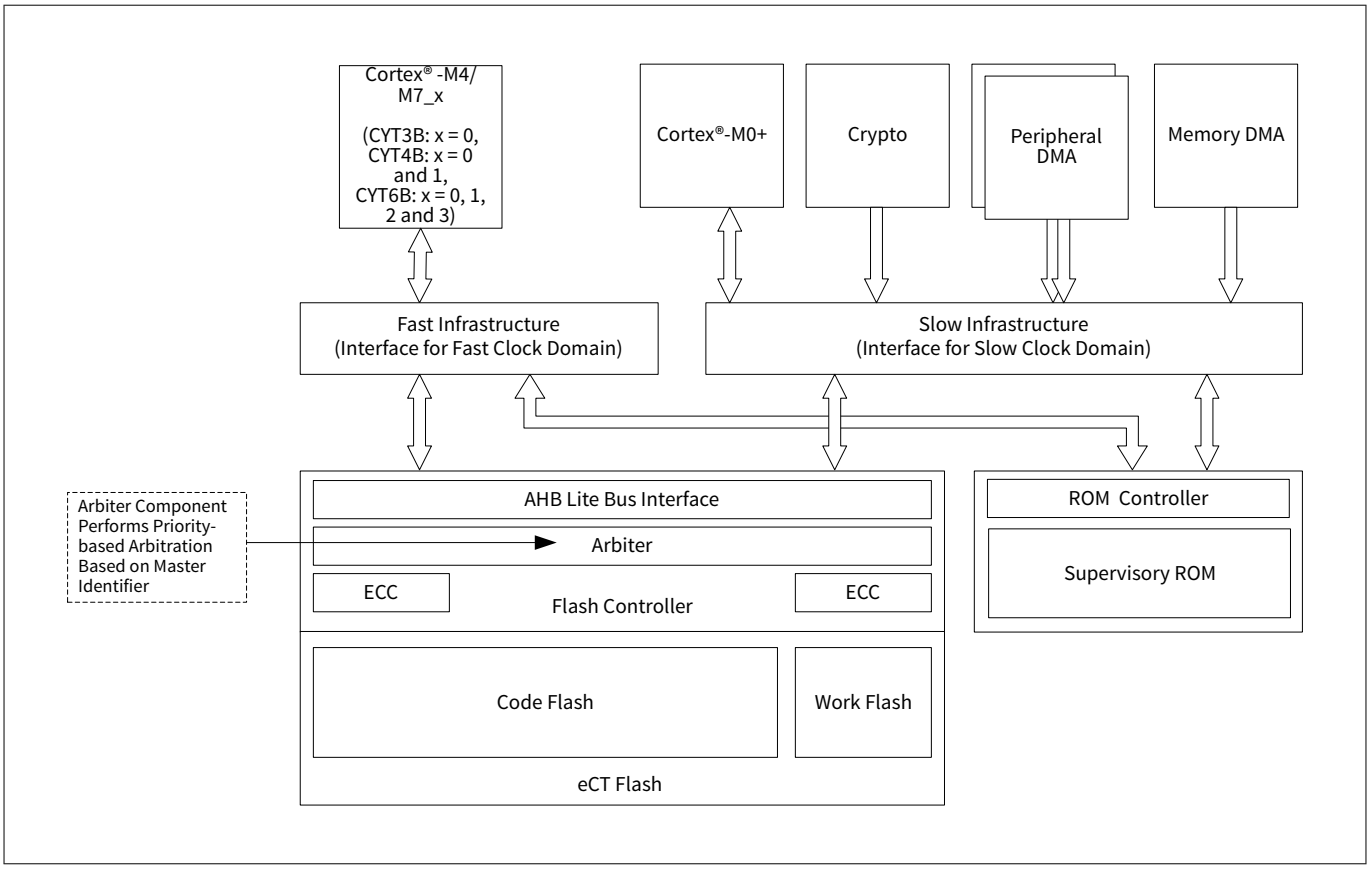

図 1　　フラッシュメモリ インタフェースのブロックダイヤグラム

注:　　図 1 に簡易ブロックダイヤグラムを示します。詳細は architecture TRM を参照してください。

表 2 に、マスタ識別子を示します。他のシリーズのマスタ識別子については、デバイスデータシートを参照してください。

表 2　　調停ブロックとバスマスタ

マスタ識別子	バスマスタ					
	CYT2B7 シリーズ	**CYT3BB シリーズ**	**CYT4BF シリーズ**	**CYT6BJ シリーズ**	**CYT3DL シリーズ**	**CYT4DN シリーズ**
0	CM0+ CPU	CM0+ CPU	CM0+ CPU	CM0+ CPU	CM0+ CPU	CM0+ CPU
1	CRYPTO コンポーネント	CRYPTO コンポーネント	CRYPTO コンポーネント	CRYPTO コンポーネント	CRYPTO コンポーネント	CRYPTO コンポーネント
2	P-DMA 0	P-DMA 0	P-DMA 0	P-DMA 0	P-DMA 0	P-DMA 0
3	P-DMA 1	P-DMA 1	P-DMA 1	P-DMA 1	P-DMA 1	P-DMA 1
4	M-DMA 0	M-DMA 0	M-DMA 0	M-DMA 0	M-DMA 0	M-DMA 0
5	-	SDHC	SDHC	SDHC	-	-
6	-	Ethernet 0	-	-	-	-
7	-	-	-	CM7_2 CPU	-	-
8	-	-	-	CM7_3 CPU	-	-

(続く)

表 2 (続き) 調停ブロックとバスマスタ

マスタ識別子	バスマスタ					
	CYT2B7 シリーズ	CYT3BB シリーズ	CYT4BF シリーズ	CYT6BJ シリーズ	CYT3DL シリーズ	CYT4DN シリーズ
9	-	-	Ethernet 0	Ethernet 0	Ethernet 0	Ethernet 0
10	-	-	Ethernet 1	Ethernet 1	-	JPEG デコーダ
11	-	-	-	-	-	M-DMA1
12	-	-	-	-	VIDEO サブシステム	VIDEO サブシステム
13	-	-	CM7_1 CPU	CM7_1 CPU		CM7_1 CPU
14	CM4 CPU	CM7_0 CPU	CM7_0 CPU	CM7_0 CPU	CM7_0 CPU	CM7_0 CPU
15	DAP Tap コントローラ	DAP Tap コントローラ	DAP Tap コントローラ	DAP Tap コントローラ	DAP Tap コントローラ	DAP Tap コントローラ

2.3 セクタ設定

図 2 に CYT4B/6B シリーズのフラッシュメモリのセクタ設定を示します。CYT4B シリーズには 8128 KB (32 KB 大セクタ)、CYT6B シリーズは 16256 KB (32 KB 大セクタ) と 256KB (8 KB 小セクタ) のコードフラッシュと追加のワークフラッシュを搭載します。ワークフラッシュは、コードフラッシュよりも頻繁にデータを書き込めるように最適化します。Supervisory flash 領域には、ハードウェア IP のトリミングパラメータ, システム設定パラメータ, 保護設定, セキュリティ設定, ブートスクリプト等のデータを格納します。フラッシュメモリサイズやセクタ設定は製品により異なります。詳細はデバイスデータシートおよび architecture TRM を参照してください。

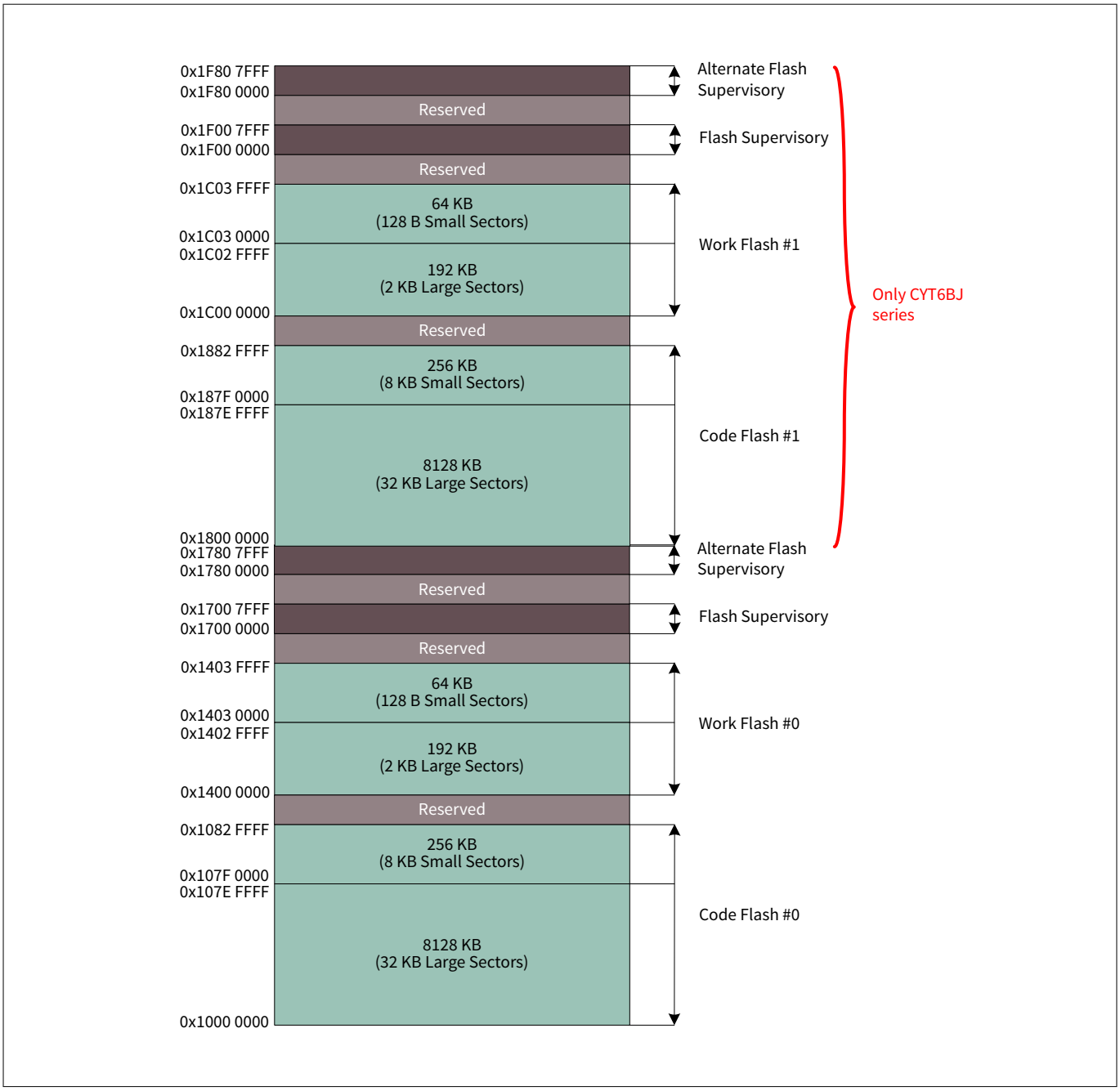

図 2　　CYT4B/6B シリーズにおけるセクタ設定

2.3.1　バンクモードと Remap 機能

2.3.1.1　シングルバンクモード

コードフラッシュ領域, ワークフラッシュ領域, および Supervisory 論理領域が、単一の連続したアドレス領域としてマッピングされます。

2.3.1.2　デュアルバンクモード

フラッシュマッピングは 2 面に分割され、各半分は別々のアドレス領域として表示されます。このモードでは、同一領域のファームウェアアップデートに対応するために実行するプログラムを入れ替えられます。

2 フラッシュメモリ概要

図 3 に CYT4B シリーズのコードフラッシュメモリのマッピングを示します。コードフラッシュおよびワークフラッシュは、シングルバンクモードとデュアルバンクモードの両方をサポートします。ファームウェアアップデートには、デュアルバンクモードと Read-while-write (RWW) を使用できます。フラッシュメモリのサイズとメモリマッピングは製品の仕様によって異なります。詳細はデバイスデータシートおよび architecture TRM を参照してください。

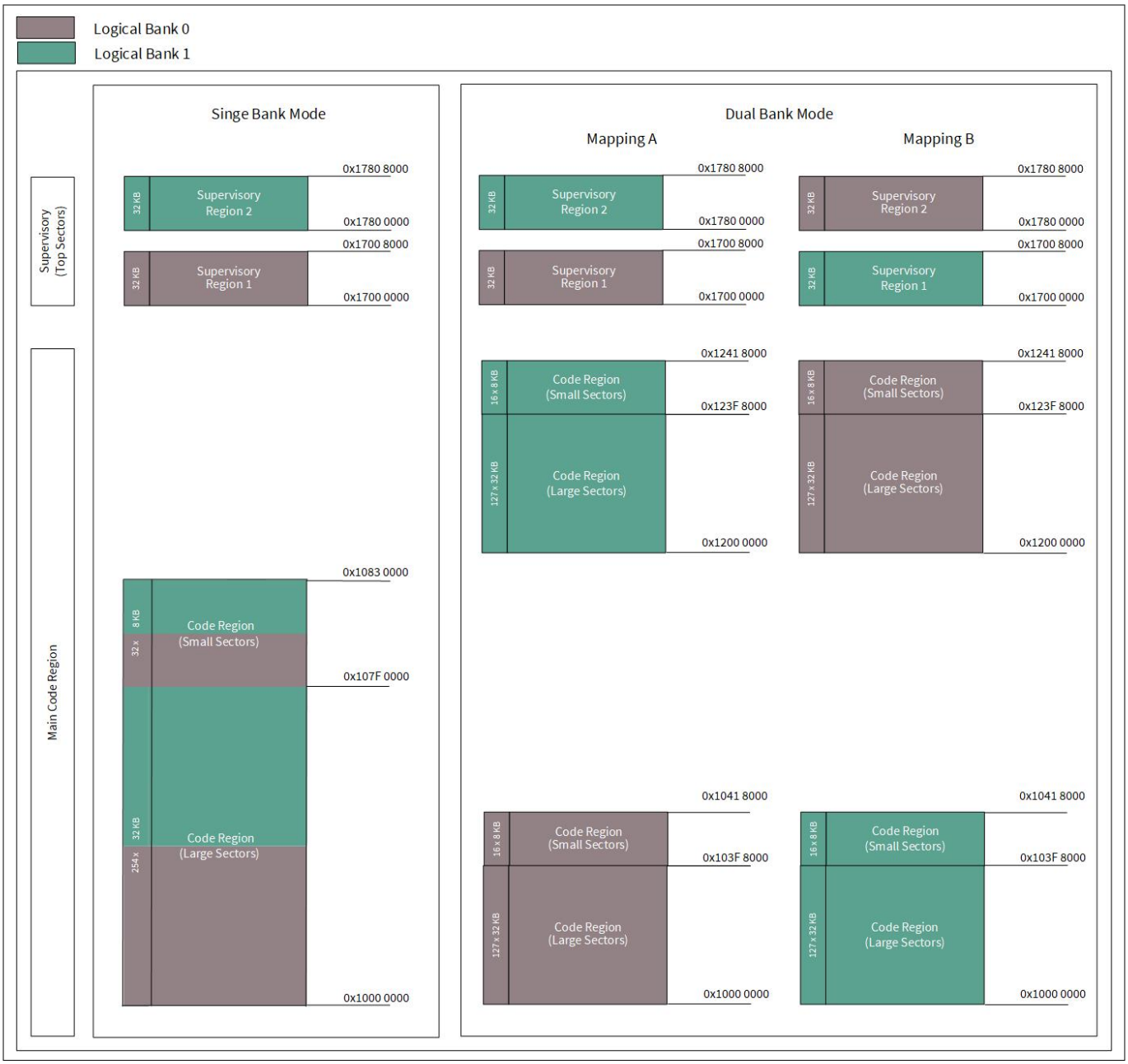

図 3　CYT4B シリーズにおけるシングルバンク/デュアルバンクモードのメモリマップ

3 フラッシュメモリの書換え手順

フラッシュメモリの全消去を行う Erase All 動作とコードフラッシュメモリに対して 64 ビットのデータ書込みを行う Program Row の動作例を説明します。さらに、サンプルドライバライブラリ (SDL) を使用して低電力モードの手順を使用する方法について説明します。このアプリケーションノートのプログラムコードは SDL の一部です。SDL のその他の参考資料を参照してください。

SDL には設定部とドライバ部があります。設定部では、主に目的の動作のパラメータ値を設定します。ドライバ部は設定部のパラメータ値に基づいて各レジスタを設定します。お客様のシステムに応じて設定部を設定できます。

TRAVEO™ T2G ファミリには、TRAVEO™ T1G ファミリに搭載していた自動アルゴリズムやコマンドシーケンサの構造は存在せず、フラッシュメモリ操作はシステムコールとして実装されます。システムコールは CM0+ IRQ0 割込みハンドラの中で実行されます。SROM コードは変更できません。CM4/CM7_x ユーザコードはプロセス間通信 (IPC) に対して SROM ファンクションのオペコードとパラメータを IPC DATA レジスタに書込みシステムコールを要求します。その結果、IRQ0 割込みが呼び出され、要求された SROM API が実行されます。

図 4 に CYT2B シリーズにおける IPC を使用したシステムコールインタフェースを示します。システムコールは、CM0+, CM4, または DAP によって要求されます。それぞれに専用のプロセス間通信 (IPC) 構造があり、CM0+にシステムコールを要求できます。フラッシュメモリを書き換えるために CM4 ユーザソフトウェアがシステムコールを要求する場合、CM0+との通信には IPC チャネル構造体 1 と IPC 割込み構造体 0 を使用します。各デバイスで利用可能な IPC チャネル構造のサポート数については、デバイスデータシートと architecture TRM を参照してください。

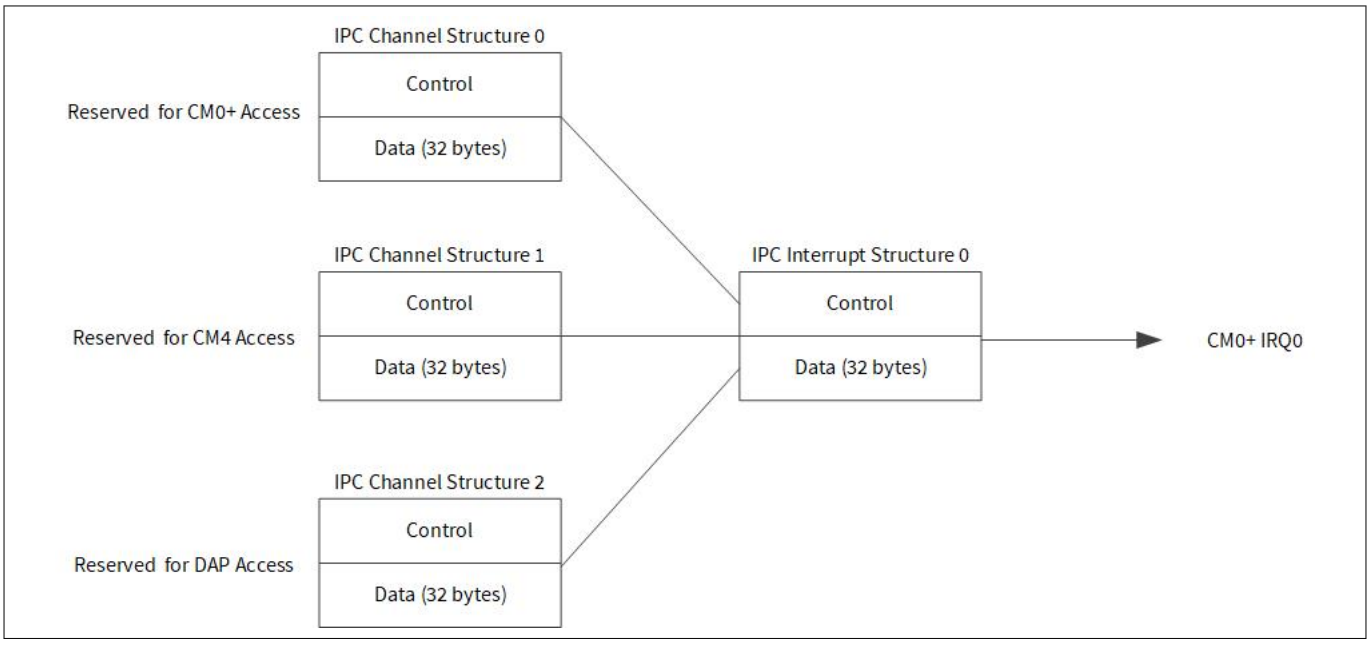

図 4 CYT2B シリーズにおける IPB を使用したシステムコールインタフェース

***注**: デバッグ実行中のメモリ表示をサポートする IDE を介して RWW 実行中のフラッシュメモリからデータ読出しを行うと、バスエラーが発生する可能性があります。エラーが発生する場合は、デバッガのメモリビューウィンドウを閉じるか、デバッガのライブメモリ機能を無効にしてください。*

3.1 システムコールの IRQ の設定

ユーザソフトウェアは、システムコール管理のために CM0+ IRQ0 および IRQ1 割込みを正しく設定する責任があります。ブートコードは自動的に CPUSS_CM0_SYSTEM_INT_CTL0.CPU_INT_VALID ビットを'1'にし、CPUSS_CM0_SYSTEM_INT_CTL0.CPU_INT_IDX [2:0]ビットを b'000 に設定します。したがって、システムコールの CM0+ IRQ0 へのシステム割込み 0 (IPC 割込み構造 0 割込み) のマッピングはブートコードによって行われ、

3 フラッシュメモリの書換え手順

CM0+ IRQ0 は IPC 割込み構造 0 割込みによってトリガされます。しかし、ソフトウェアは CM0+ IRQ0 と IRQ1 が有効であり、正しい優先順位で設定されていることを確認する必要があります。これはブートコードによって自動的に行われないためです。また、ソフトウェアは、ユーザ CM0+ベクタテーブルの IRQ0 および IRQ1 ベクタエントリが、デフォルトの SROM ベクタテーブル (それぞれアドレス 0x00000040 および 0x00000044) のベクタエントリと同一であることを確認する必要があります。これは、RAM 領域にある場合、ランタイム中に SROM ベクタテーブルからユーザベクタテーブルに値をコピーすることで実現できます。それ以外の場合は、ハードコーディングされた値を使用し、ターゲット MCU またはリビジョンが変更された場合に再確認する必要があります。

CPU がスレッドモードでコードを実行しているとき、CONTROL レジスタは、プロセススタックポインタ (PSP) またはメインスタックポインタ (MSP) を使用するように設定できます。ハンドラモードでは、常に MSP が使用されます。CPU はスレッドモードに入り、リセットから抜けると MSP を使用することに注意すべきです。さらに、ソフトウェアはシステムコールの割込みを設定する際に特別な注意を払う必要があります。これは、CM0+の CPU モード (スレッドまたはハンドラ) およびシステムコールが引き起こされたときに使用されたスタックポインタ (MSP または PSP) に依存するためです。

ケース 1

CM0+がハンドラモードのときにのみソフトウェアがシステムコールをトリガする場合は、ソフトウェアが CM0+ IRQ1 を IRQ0 よりも高い優先順位で設定していること確認してください。これは、IRQ0 の優先度を「1」に設定することで実行できます。デフォルトでは、IRQ1 の優先度は「0」に設定されます。

ケース 2

ソフトウェアが他の CPU ステータス (スレッドモードや PSP など) で CM0+のシステムコールをトリガする場合、ソフトウェアはさらに、システムコールのマネージャーとして機能する別の CM0+割込み (IRQ2 など) を使用する必要があります。このアプローチは、原則として任意の CPU ステータス (ハンドラモードを含む) にも使用できます。したがって、これはすべての CPU ステータスでシステムコールを管理するためのより一般的なアプローチです。

CM0+ IRQ をセットアップするアプローチには、次の手順が含まれます。

1. システムコールマネージャ関数 (例えば「Sys_Call_Manager」) をユーザベクタテーブルの CM0+ IRQ2 の IRQ ハンドラとして設定してください。
2. IPC Interrupt Structure 0 割込みを CM0+ IRQ2 にマップしてください。
3. IRQ0 および IRQ1 に関して最も低い優先度を IRQ2 に設定してください。IRQ0 と IRQ1 を同じく最も高い優先順位に設定してください。例えば、IRQ2 の優先度を 1 に設定してください。IRQ0 と IRQ1 の優先順位はデフォルトで 0 です。
4. IRQ2 ハンドラはソフトウェアで IRQ0 をトリガします。
5. IRQ2 ハンドラは、IRQ0 の Pending Bit をクリアします。

したがって、CM0+ベクタテーブルには表 3 に示すように最初に 3 つの割込みのエントリがあります。

表 3　　CY4TB シリーズのフラッシュメモリの概要

割込み番号	ハンドラ
…	…
IRQ0	Contents of address (0x00000040)
IRQ1	Contents of address (0x00000044)
IRQ2	Sys_Call_Manager
…	…

また「Sys_Call_Manager」を CM0+ IRQ2 ハンドラとして直接割り当てる代わりに、ディスパッチャ実装を使用するときは、他の複数のシステム割込みとも組み合わせられます。

ケース 2 のシステムコールに必要な割込み設定の疑似コードを以下に示します。

```
/* IRQ2 handler function for IPC Interrupt structure 0 interrupt. This is the system call
manager function */

void Sys_Call_Manager()
{
  /* Trigger IRQ0 in Software by writing to ISPR register */
  CM0P_SCS_ISPR = 1;

  /* Read back the register to ensure that the write has happened */
  CM0P_SCS_ISPR;

  /* Clear the NVIC Pending bit of IRQ0. This is done as a fallback in case the   system  call
was suppressed (e.g., by disabled interrupts) */
  CM0P_SCS_ICPR = 1;

  /* Read back the register to ensure that the write has happened */
  CM0P_SCS_ICPR;
}
/* Application function for interrupt configurations */

void interrupt_configure()
{
  /* Enable CM0+ IRQ0, IRQ1 and IRQ2 */
  CM0P_SCS_ISER = 7;

  /* Set Priority 0 for IRQ0, IRQ1 and Priority 1 for IRQ2 */
  CM0P_SCS_IPR0 = 0x00400000;

  /* Connect IPC Interrupt Structure 0 Interrupt (System Interrupt 0) to
  IRQ2. The interrupt triggers Sys_Call_Manager */
  CPUSS_CM0_SYSTEM_INT_CTL0.CPU_INT_IDX = 2;

  /* Clear the PRIMASK register to enable the interrupts. This could also
  be done by the application at a later point in time */
  __ASM("cpsie i");
}
```

3.2 Flash Erase All 処理の手順

図 5 に、CM4/7 CPU コアからの IPC と CM0+ IRQ0 ハンドラを使用した「Erase All」処理の設定手順を示します。この例では、コードフラッシュ全体の消去動作を示し、Supervisory flash 領域とワークフラッシュの消去動作は含みません。CM4/7 コードは SRAM に配置され実行されることを想定します。「Erase All」API のオペコードは 0x0A です。以下の全体動作で使用するパラメータの詳細は、architecture TRM の「Nonvolatile Memory Programming」章を参照してください。

注: *以下のセクションでは、IRQ0 がシステムコールをトリガすることを想定します。つまり、CM0+ がハンドラモードのときにシステムコールがトリガされます。その他すべての CM0+ CPU 状態については、システムコールの IRQ の設定 で説明されているように、割込みを設定してください。*

3.2.1 事前の設定

1. メインフラッシュ組込み動作を有効にする (FLASHC_FM_CTL_ECT_MAIN_FLASH_SAFETY_MainFlashWriteEnable =1) [CM4/7 ユーザコード]
2. SRAM スクラッチアドレスに 4 ワードを割り当て (uint32_t SramScratch[4];) [CM4/7 ユーザコード]

注: CM7_0 と CM7_1 の両方に対応する CYT4B シリーズを使用する場合は、キャッシュコヒーレンスを考慮する必要があります。""SramScrach"/"data" で示される SRAM 領域は複数のコアで共有されているため、この領域はキャッシュ可能に設定しないでください。

3.2.2 CM0+ IRQ0 ハンドラに対する「Erase All」API の要求

1. IPC1 チャネル構造体のロック取得 [CM4/7 ユーザコード]
2. SRAM スクラッチメモリに対して API パラメータを書込み [CM4/7 ユーザコード]
   a. SramScratch[0]に「Erase All」オペコード(0x0A) を書込み (SramScratch[0] = 0x0A000000;)
      - Bits [31:24]: Erase All opcode = 0x0A
      - Bits [23:00]: 0x00 (未使用)
3. IPC1_DATA0 に SRAM スクラッチメモリを割り当て [CM4/7 ユーザコード]。
4. IPC_NOTIFY レジスタを介して通知イベントを生成 [CM4/7 ユーザー コード]。その後、IPC0 割込み構造を介して CM0+に API 通知イベントを生成。

3.2.3 CM0+ IRQ0 ハンドラにおける「Erase All」API の実行

1. IPC0 割込み構造体を介して API 通知イベントを検出 [CM0+ユーザコード]
2. IPC1_DATA0 レジスタから SRAM_SCRACH_ADDR (&SramScratch[0]) を読み出し [CM0+ IRQ0 ハンドラ]
3. SRAM_SCRATCH_ADDR (&SramScratch[0]) からオペコード (0x0A) を読み出し [CM0+ IRQ0 ハンドラ]
4. 不明なオペコードの場合、SRAM_SCRATCH_ADDR (&SramScratch[0]) に不良コードを書込み [CM0+ IRQ0 ハンドラ]
5. Erase All を実行 [CM0+ IRQ0 ハンドラ]
6. 実行結果を取得し、成功コードを SRAM スクラッチアドレスに書込み [CM0+ IRQ0 ハンドラ]
7. 実行結果が失敗の場合、SRAM スクラッチアドレスに不良コードを書込み [CM0+ IRQ0 ハンドラ]
8. IPC_RELEASE レジスタを介してリリースイベントを生成 [CM0+ IRQ0 ハンドラ]。その後、IPC2 割込み構造体を介して API 通知イベントが発生

3.2.4 CM0+における「Erase All」処理結果の取得

1. IPC2 割込み構造体からのリリース割込みを検出 [CM4/7 ユーザコード]
2. SRAM_SCRATCH_ADDR (&SramScratch[0]) からステータスを読み出し、API 実行結果を取得 [CM4/7 ユーザコード]
3. ステータスが成功の場合、Erase All 処理を終了 [CM4/7 ユーザコード]

3 フラッシュメモリの書換え手順

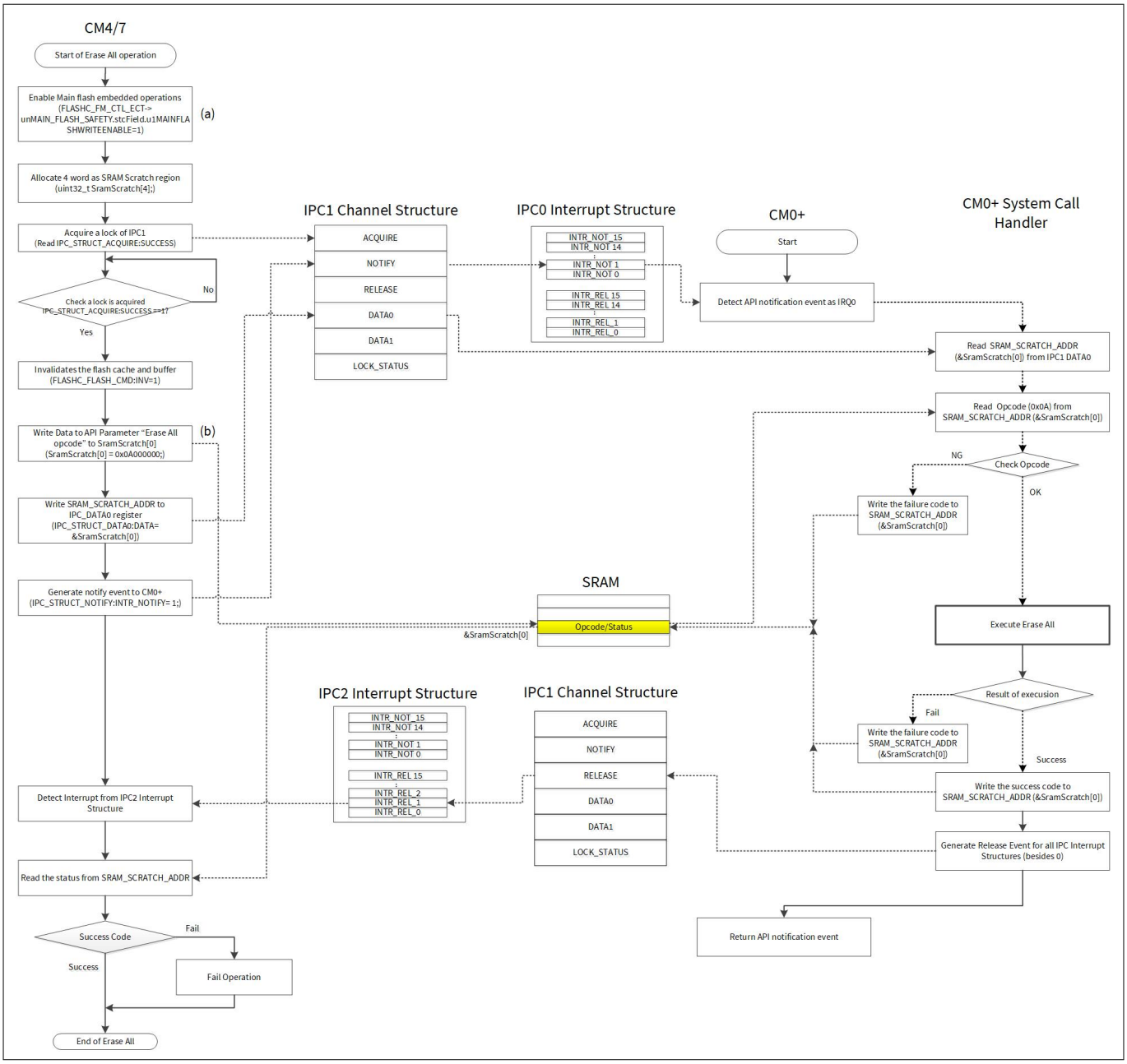

図 5　　Flash Erase All 処理の手順例

3.2.5　設定とサンプルコード

表 4 に Flash Erase All の SDL の設定部の関数を示します。

表 4　　Flash Erase All 処理の関数一覧

関数	説明	備考
`Cy_Flashc_MainWriteEnable()`	メインフラッシュの書込みを有効にします	Code Listing 2 を参照してください
`Cy_Flash_EraseAll()`	フラッシュ全体を消去します。この API は、メイン (コード) フラッシュ配列のみを消去します	Code Listing 3 を参照してください

3 フラッシュメモリの書換え手順

Code Listing 1 は Flash Erase All 動作のサンプルプログラムを示します。architecture TRM を参照してください。以下の説明は、SDL のドライバ部のレジスタ表記を理解するのに役立ちます。

- FLASHC_FM_CTL_ECT->unMAIN_FLASH_SAFETY.stcField.u1MAINFLASHWRITEENABLE は、registers TRM に記載されている FM_CTL_ECT_FLASHC_MAIN_FLASH_SAFETY.MainFlashWriteEnable 設定です。他のレジスタも同様に記述されます。

レジスタの結合と構造表現の詳細については、*hdr/rev_x/ip* の *cyip_flashc_v2.h* を参照してください。

Code Listing 1 Flash Erase All 動作の例

```
int main(void)
{
:
    /* Enable main flash embedded operations */  /* Enable writing main flash. See Code Listing
2. See (a) of Figure 5. */
    Cy_Flashc_MainWriteEnable();

    /* Call Function "Cy_Flash_EraseAll" and get the API operation result */
    cy_un_flash_context_t flashContext = { 0ul };
    cy_en_flashdrv_status_t status =
Cy_Flash_EraseAll(&flashContext,CY_FLASH_DRIVER_NON_BLOCKING);

    if(status != CY_FLASH_DRV_SUCCESS)  /* Enable writing main flash. See Code Listing 3. See
(a) of Figure 5. */
    {
        //Erase All operation failed.
        while(1);
    }

    for(;;);
}
```

Code Listing 2 Cy_Flashc_MainWriteEnable() 関数

```
__STATIC_INLINE void Cy_Flashc_MainWriteEnable(void)
{
    FLASHC_FM_CTL_ECT->unMAIN_FLASH_SAFETY.stcField.u1MAINFLASHWRITEENABLE = 1ul;  /* Enable
writing main flash. */
}
```

Code Listing 3 Cy_Flash_EraseAll() 関数

```
cy_en_flashdrv_status_t Cy_Flash_EraseAll(cy_un_flash_context_t* context,
cy_en_flash_driver_blocking_t block)  /* Erase the whole flash. */
{
    /* Prepares arguments to be passed to SROM API */
    un_srom_api_args_t apiArgs = { 0ul };
    apiArgs.EraseAll.arg0.opcode = CY_SROM_OP_FLASH_ERASE_ALL;

    /* Call SROM API driver and process response */
    return Cy_Flash_CallSromApiForFlashWrite(&apiArgs, block);
}
```

3.3 Program Row 処理の手順

図 6 に、CM4/7 によるフラッシュメモリへの Program Row 処理の設定例と、IPC と CM0+ IRQ0 ハンドラの動作を示します。この例では、64 ビットのテストデータをコードフラッシュに書き込む操作を示します。書込み先のコードフラッシュのアドレスは 0x10000000 です。テストデータは 0x55AA55AA x 2 です。オペコードが 0x06 の Program Row API がシステムコールとして使用されます。以下の全体動作で使用するパラメータの詳細は、architecture TRM の「Nonvolatile Memory Programming」章を参照してください。

注: 以下のセクションでは、IRQ0 がシステムコールをトリガすることを想定します。つまり、CM0+ がハンドラモードのときにシステムコールがトリガされます。その他すべての CM0+ CPU 状態については、システムコールの IRQ の設定 で説明されているように、割込みを設定してください。

3.3.1 パラメータの事前設定

1. メインフラッシュ組込み動作を有効にします (FLASHC_FM_CTL_ECT_MAIN_FLASH_SAFETY_MainFlashWriteEnable =1) [CM4/7 ユーザコード]
2. SRAM スクラッチアドレスに 4 ワードを割り当て (uint32_t SramScratch[4];) [CM4/7 ユーザコード]
3. コードフラッシュへの書込みデータのバッファとして 2 ワードを割り当て (uint32_t data[2];) [CM4/7 ユーザコード]

注: CM7_0 と CM7_1 の両方に対応する CYT4B シリーズを使用する場合は、キャッシュコヒーレンスを考慮する必要があります。"SramScrach"/"data" で示される SRAM 領域は複数のコアで共有されているため、この領域はキャッシュ可能に設定しないでください。

3.3.2 CM0+ IRQ0 に対する「Program Row」API の要求

1. IPC1 チャネル構造体のロック取得してください。[CM4/7 ユーザコード]
2. SRAM に対して API パラメータを更新してください。[CM4/7 ユーザコード]
   a. SramScratch[0]に Program Row オペコード (0x06)を書き込んでください (SramScratch[0] = 0x06000000)。
      - Bits [31:24]: Program Row opcode = 0x06
      - Bits [23:16]: Skip blank check= 0x00 (Perform blank check)
      - Bits [15:08]: Blocking mode=0x00 (non-blocking)
      - Bits [08:00]: 0x00 (未使用)

b. SramScratch[1]にデータサイズを書き込んでください (SramScratch[1] = 0x00000003;)。
- Bits [31:24]: 割込みマスク、非ブロッキング=0x00 (FM 割込みマスクが設定されていない) の場合にのみ適用可能
- Bits [23:16]: 0x00 (未使用)
- Bits [15:08]: Data location=0x00 (Page latch)
- Bits [08:00]: Data size for code flash=0x03 (64 bits)

c. SramScratch[2]にプログラムされたフラッシュアドレスを書き込んでください (SramScratch[2] = 0x10000000)。
- Bits [31:0]: Flash address to be programmed = 0x10000000

d. SramScratch[3]に対して SRAM_SCRATCH_DATA_ADDR を書き込んでください (SramScratch[3] = &data[0])。

e. 32 ビットのプログラムデータを data[0]に書き込んでください (data[0] = 0x55AA55AA)。

f. 32 ビットのプログラムデータを data[1]に書き込んでください (data[1] = 0x55AA55AA)。

3. IPC1_DATA0 レジスタに SRAM スクラッチメモリを割り当ててください。
4. IPC1_NOTIFY レジスタ [CM4/7 ユーザコード] への書込みにより通知イベントを生成してください。その後、IPC0 割込み構造体を介して CM0+に対して API 通知イベントを生成してください。

3.3.3 CM0+ IRQ0 における「Program Row」API の実行

1. IPC0 割込み構造体を介して API 通知イベントを検出 [CM0+ユーザコード]
2. IPC1_DATA0 から SRAM_SCRATCH_ADDR を読み出し [CM0+ IRQ0 ハンドラ]
3. SRAM からオペコード (0x06), プログラムデータサイズ, プログラム対象のアドレス, およびプログラムデータを読み出し [CM0+ IRQ0 ハンドラ]
4. 不明なオペコードの場合、SRAM_SCRATCH_ADDR (&SramScratch[0]) に不良コードを書込み [CM0+ IRQ0 ハンドラ]
5. Program Row 処理を実行 [CM0+ IRQ0 ハンドラ]
6. 実行結果が成功の場合、成功コードを SRAM_SCRATCH_ADDR (&SramScratch[0]) に書込み [CM0+ IRQ0 ハンドラ]
7. 実行結果が失敗の場合、SRAM_SCRATCH_ADDR (&SramScratch[0]) に不良コードを書込み [CM0+ IRQ0 ハンドラ]

3.3.4 CM0+における「Program Row」処理結果の取得

1. IPC2 割込み構造体からのリリース割込みを検出 [CM4/7 ユーザコード]
2. ステータスが成功の場合、Program Row 処理を終了 [CM4/7 ユーザコード]

3 フラッシュメモリの書換え手順

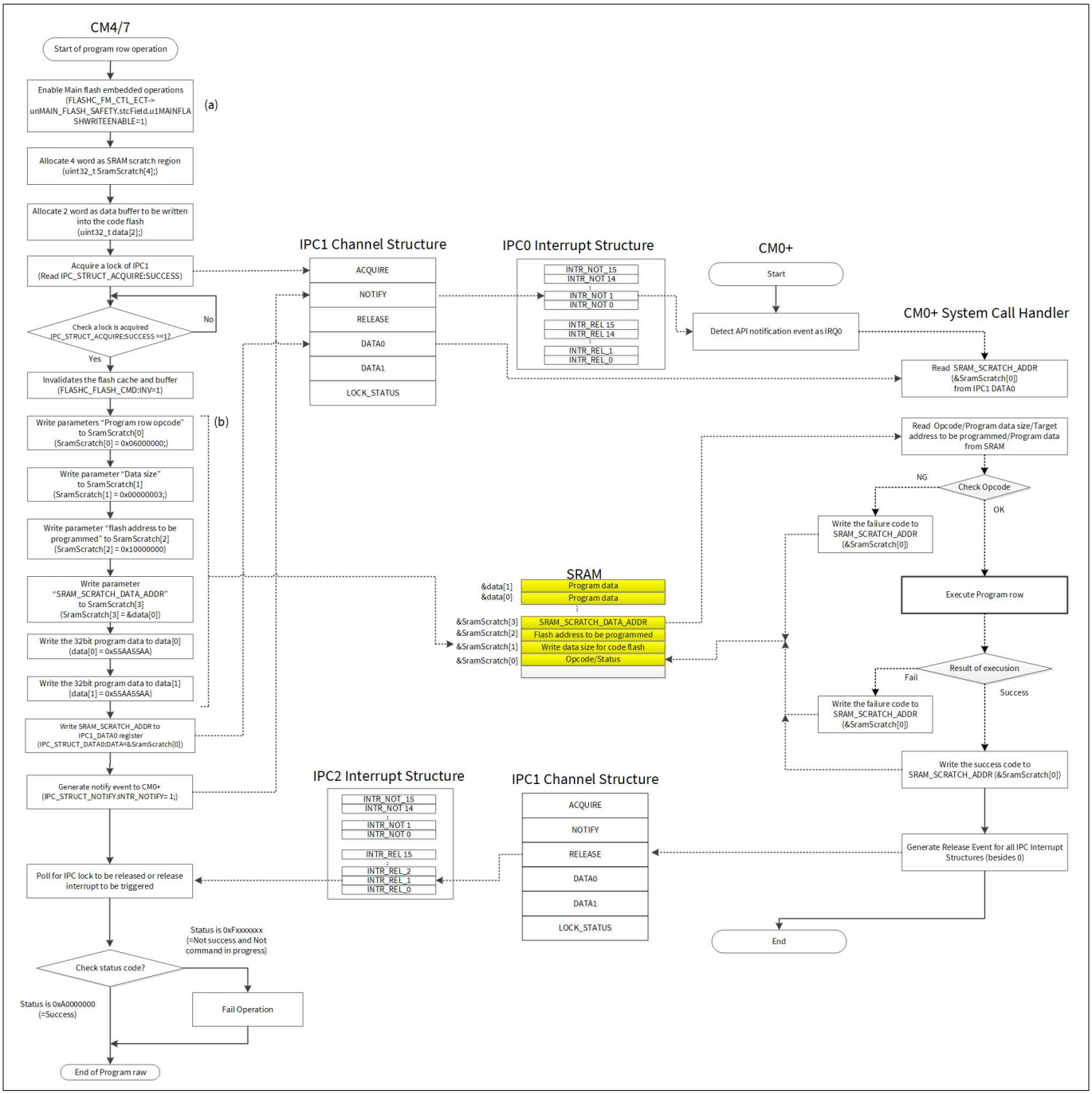

図 6　　Flash Program Row 処理の設定例

3.3.5　設定とサンプルコード

表 5 に Program Row の SDL の設定部の関数を示します。

表 5　　Program Row 処理の関数一覧

関数	説明	備考
`Cy_FlashInit()`	フラッシュ IP を初期化します	Code Listing 5 を参照してください

(続く)

表 5　　(続き) Program Row 処理の関数一覧

関数	説明	備考
Cy_Flashc_MainWriteEnable()	メインフラッシュの書込みを有効にします	Code Listing 2 を参照してください
Cy_Flashc_WorkWriteEnable()	ワークフラッシュの書込みを有効にします	Code Listing 6 を参照してください
Cy_FlashWriteCode()	コードフラッシュのアドレスにデータをプログラムします	Code Listing 7 を参照してください
Cy_Flash_ProgramRow()	フラッシュの単一行にデータの配列を書込みます	Code Listing 8 を参照してください

Code Listing 4 は、ProgramRow 動作のサンプルプログラムを示します。

Code Listing 4 Program Row 動作の例

```
int main(void)
{
:
    // Initialization
    Cy_FlashInit(false /*blocking*/);  /* Initialize flash IP. See Code Listing 5. */
:
    /** Programming **/
    for(uint32_t i_addr = SectorAddr; i_addr < SectorAddr + SectorSizeInByte;
i_addr+=SIZE_4096_BIT_IN_BYTE)
    {
        // Flash
        Cy_FlashWriteCode(i_addr, (uint32_t*)programData,
CY_FLASH_PROGRAMROW_DATA_SIZE_4096BIT,CY_FLASH_DRIVER_BLOCKING);  /* Program data to address in
code flash. See Code Listing 7. */
    }

}
```

3 フラッシュメモリの書換え手順

Code Listing 5 Cy_ FlashInit() 関数

```
void Cy_FlashInit(bool non_blocking)
{
    if(non_blocking == true)
    {
        /******  Setting for IPCs    ******/
        Cy_Srom_SetResponseHandler(Cy_FlashHandler, CPUIntIdx3_IRQn);
        NVIC_SetPriority(CPUIntIdx3_IRQn, 3ul);
        NVIC_EnableIRQ(CPUIntIdx3_IRQn);
        g_NB_ModeEnabled = true;
    }
    else
    {
        g_NB_ModeEnabled = false;
    }

    /*  Flash Write Enable   */
    Cy_Flashc_MainWriteEnable();  /* Enable writing main flash.  See Code Listing 2. See (a) of
Figure 6. */
    Cy_Flashc_WorkWriteEnable();

    g_completeFlag = true;
}
```

Code Listing 6 Cy_Flashc_WorkWriteEnable() 関数

```
__STATIC_INLINE void Cy_Flashc_WorkWriteEnable(void)
{
    FLASHC_FM_CTL_ECT->unWORK_FLASH_SAFETY.stcField.u1WORKFLASHWRITEENABLE = 1ul;  /* Enable
writing work flash. */
}
```

3 フラッシュメモリの書換え手順

Code Listing 7 Cy_ FlashWriteCode() 関数

```
void Cy_FlashWriteCode(uint32_t writeAddr, const uint32_t* data,
cy_en_flash_programrow_datasize_t size, cy_en_flash_driver_blocking_t blocking)
{
    CY_ASSERT(Cy_Flash_MainBoundsCheck(writeAddr) == CY_FLASH_IN_BOUNDS);

    uint32_t status;
    cy_stc_flash_programrow_config_t programRowConfig = {0};

    if(blocking == CY_FLASH_DRIVER_NON_BLOCKING)
    {
        CY_ASSERT(g_NB_ModeEnabled== true);

        /* Only for non-blocking operation */
        g_completeFlag = false;
    }

    // Program code flash
    programRowConfig.blocking = CY_FLASH_PROGRAMROW_BLOCKING;
    programRowConfig.skipBC   = CY_FLASH_PROGRAMROW_SKIP_BLANK_CHECK;
    programRowConfig.dataSize = size;
    programRowConfig.dataLoc  = CY_FLASH_PROGRAMROW_DATA_LOCATION_SRAM;
    programRowConfig.intrMask = CY_FLASH_PROGRAMROW_NOT_SET_INTR_MASK;
    programRowConfig.destAddr = (uint32_t*)writeAddr;
    programRowConfig.dataAddr = data;
    status = Cy_Flash_ProgramRow(NULL, &programRowConfig, blocking);  /* Program addressed
flash. See Code Listing 8. See (b) of Figure 6. */

    CY_ASSERT(status == CY_FLASH_DRV_SUCCESS);
}
```

3 フラッシュメモリの書換え手順

Code Listing 8 Cy_Flash_ProgramRow() 関数

```
cy_en_flashdrv_status_t Cy_Flash_ProgramRow(cy_un_flash_context_t* context, const
cy_stc_flash_programrow_config_t* config, cy_en_flash_driver_blocking_t block)
{
    /* Checks if input pointers are not NULL */
    if(config == NULL)
    {
        return CY_FLASH_DRV_INVALID_INPUT_PARAMETERS;
    }

    /* Checks if input address are valid */
    if((Cy_Flash_BoundsCheck((uint32_t)config->destAddr) == CY_FLASH_OUT_OF_BOUNDS) || (config-
>dataAddr == NULL))
    {
        return CY_FLASH_DRV_INVALID_INPUT_PARAMETERS;
    }

    /* Prepares arguments to be passed to SROM API */
    un_srom_api_args_t apiArgs = { 0ul };
    apiArgs.ProgramRow.arg0.opcode         = CY_SROM_OP_FLASH_PROGRAM_ROW;
    apiArgs.ProgramRow.arg0.blocking       = config->blocking;
    apiArgs.ProgramRow.arg0.skipBlankCheck = config->skipBC;
    apiArgs.ProgramRow.arg1.dataLoc        = config->dataLoc;
    apiArgs.ProgramRow.arg1.dataSize       = config->dataSize;
    apiArgs.ProgramRow.arg1.interruptMask  = config->intrMask;
    apiArgs.ProgramRow.arg2.dstAddr        = (uint32_t)config->destAddr;
    apiArgs.ProgramRow.arg3.srcAddr        = (uint32_t)config->dataAddr;

    /* Call SROM API driver and process response */
    return Cy_Flash_CallSromApiForFlashWrite(&apiArgs, block);
}
```

4 デュアルバンク手法を用いたフラッシュメモリの書換え

ここでは、フラッシュデュアルバンク手法を用いて over-the-air (OTA) ファームウェア更新機能を有効にするフラッシュメモリの書換え例を示します。

4.1 コンセプト

TRAVEO™ T2G ファミリの MCU では、OTA 機能は 2 種類のレジスタ制御によって実行します。Flash Main Bank Mode レジスタは、フラッシュバンクモードを「シングルバンクモード」もしくは「デュアルバンクモード」に設定するために使用されます。Flash Main Remap レジスタは、フラッシュメモリの領域を「マッピング A」もしくは「マッピング B」に設定するために使用されます。Flash Main Bank Mode レジスタおよび Flash Main Remap レジスタはリセットによってクリアされます。ROM ブートと Flash ブートはこれらのレジスタに対しての設定を行いません。したがって、CM0+ユーザプログラムが実行開始時には、TRAVEO™ T2G ファミリ MCU のフラッシュメモリはシングルバンクモードで動作します。その後、ユーザプログラムでデュアルバンクおよび Remap 機能を設定する必要があります。

この例では、Remap パラメータをワークフラッシュに保存し、CM0+プログラムがワークフラッシュから Remap 情報を読み出すことで、マップ切替えを行う例を説明します。ユーザが意図しないバンク切替え動作は重大な障害を引き起こす可能性があります。これを防ぐために、ワークフラッシュに格納する切替え情報の二重化やチェックコードの等の安全メカニズムを実装することを推奨します。ワークフラッシュに格納する Remap パラメータの保護としてメモリ保護機能 (MPU) や共有メモリ保護機能 (SMPU) によるメモリの保護が可能です。詳細は、architecture TRM の「Protection Unit」章を参照してください。

4.1.1 ファームウェアアップデート前のユーザアプリケーションの実行

ファームウェアアップデート前のアプリケーションを実行する場合のフラッシュメモリと関連機能の動作について説明します。図 7 に全体プロセスを示します。

1. リセット後、CM4/7 と CM0+は ROM 内のブートコードを実行します。CM4/7 は CM0+によって起動されるまで、ROM ブート内の wait for interrupt (WFI) ループに入ります。
2. ROM ブートおよびフラッシュブートの完了後、CM0+はコードフラッシュ内にある CM0+ユーザプログラムコードを実行します。
3. CM0+ユーザプログラムコードはフラッシュメインバンクモードをデュアルバンクモードに設定します。フラッシュメモリ領域は 2 面に分割され、各半分は別々のアドレス領域として表示されます。
4. CM0+ユーザプログラムコードは、特定のワークフラッシュ領域に格納されている Remap パラメータを確認します。ワークフラッシュから読み出されたデータの値に応じて、CM0+ユーザプログラムはマップ切替えを行います。初期状態でファームウェアアップデート要求を受信していないときは、オリジナルコードの実行を示すパラメータが格納されるため、マップ切替え制御を実行しません。
5. CM0+ユーザプログラムで CM4/7 ユーザプログラムのプログラムカウンターを設定した後、バンク 0 の CM4/7 ユーザプログラムが起動します。

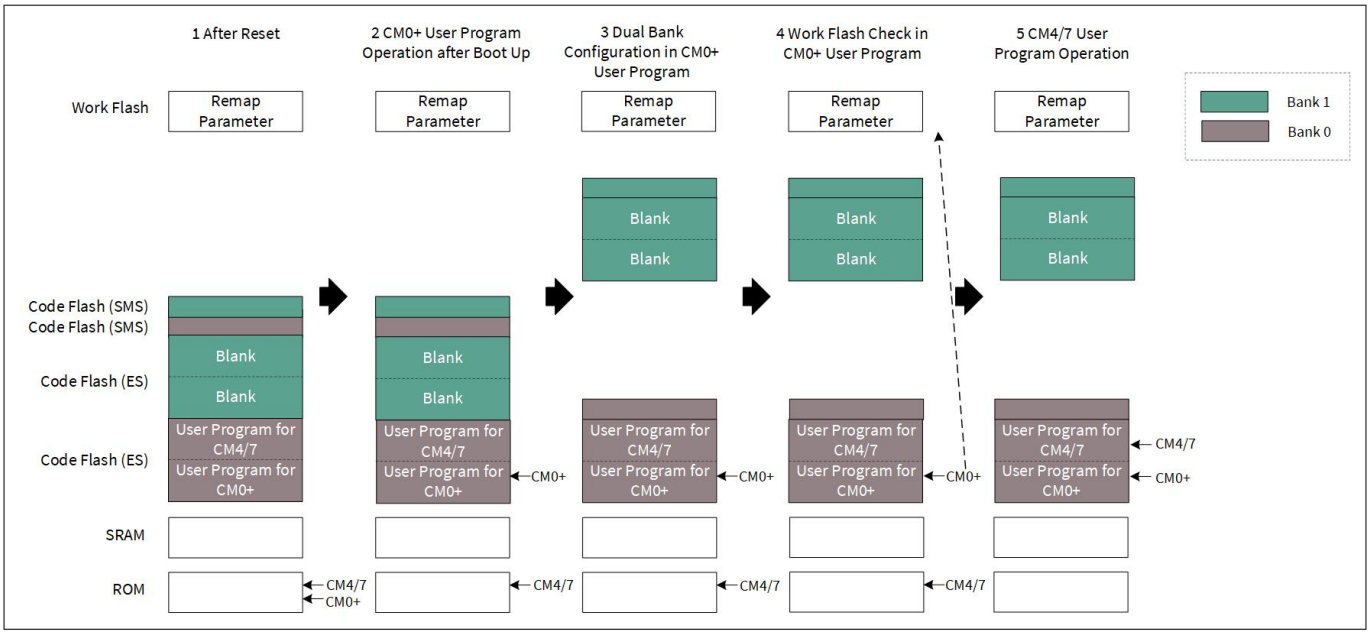

図 7　　最初のアプリケーションの開始

4.1.2　フラッシュメモリの再プログラミング

フラッシュデュアルバンクモードにおけるファームウェアアップデートについて説明します。図 8 に全体プロセスを示します。

1. バンク 0 にある CM4/7 ユーザプログラムコードは、CAN FD や Ethernet の車載 LAN を介してファームウェアアップデート要求およびデータを受信します。
2. CM0+ユーザプログラムがコードフラッシュバンク 1 の全領域および Remap パラメータが格納されているワークフラッシュ領域の消去を開始します。
3. フラッシュ消去完了後、CM0+ユーザプログラムは同じ CM0+ユーザプログラムコードをコードフラッシュバンク 1 にコピーし、CM4/7 の更新ユーザプログラムコードを書き込みます。
4. 最後に、CM0+はワークフラッシュ内の Remap パラメータを更新します。Remap パラメータは、次回のリセット起動後に実行するアプリケーションコードの切替え有無を判断するために使用されます。Remap パラメータは、アプリケーションの使用方法に応じて、任意のキーコードまたはユーザプログラムコードの開始アドレスなどで構成されます。

注:　*CYT4B シリーズでキャッシュを有効にする場合は、次のメモリに対して MPU をキャッシュ不可に設定する必要があります。*

- *IPC によって使用される SRAM スクラッチ領域*
- *ワークフラッシュ領域*

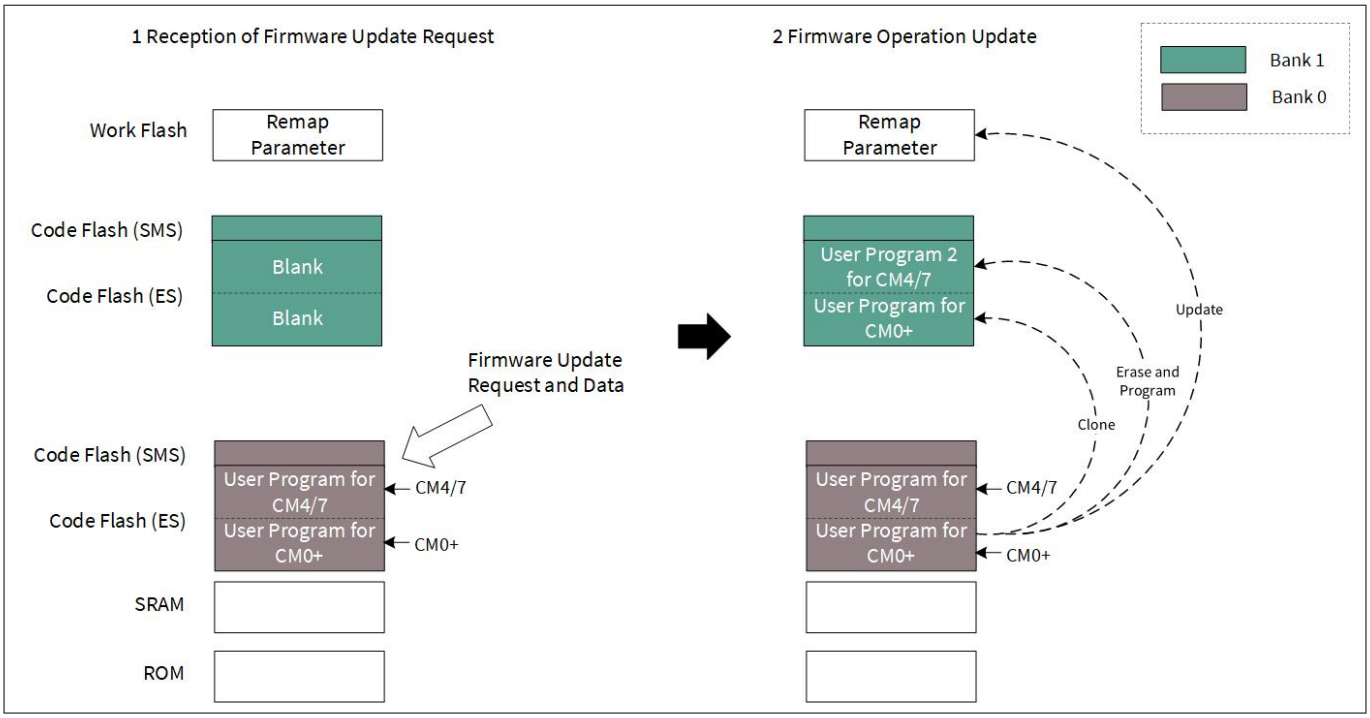

図 8　　フラッシュメモリの再プログラミング

4.1.3　新しい CM4/7 ユーザプログラムコードの開始

コードフラッシュバンクを切替え後に新しい CM4/7 ユーザプログラムコードが開始する手順例を説明します。図 9 に全体プロセスを示します。

1. 新しいユーザプログラムコードの書込みおよび Remap パラメータの更新が完了すると、ユーザプログラムコードは MCU をリセットします。リセット後、CM4/7 と CM0+は ROM 内のブートコードを実行します。
2. CM0+ユーザプログラムがフラッシュメインバンクモードをデュアルバンクモードに設定します。フラッシュマッピングは 2 面に分割され、各半分は別々のアドレス領域として表示されます。その後、CM0+ユーザプログラムはワークフラッシュから Remap パラメータを読み出します。
3. ワークフラッシュから読み出したデータが更新したユーザプログラムコードの実行を示す期待値であれば、CM0+は RAM 実行処理に遷移し、Flash Main Remap レジスタ操作によりフラッシュ領域をマッピング B に変更します。
4. マッピング B への切替え処理が完了すると、コードフラッシュバンク 0 とコードフラッシュバンク 1 が入れ替わります。CM4/7 と CM0 +は同じアドレスを使用してバンク 1 のフラッシュメモリに格納されていたデータを読み出せます。
5. CM0+がフラッシュ実行に戻ります。これにより、同じ CM0+ユーザプログラムが存在することになります。CM0+は CM4/7 ユーザプログラムコードのプログラムカウンターを設定し、その後、新しい CM4/7 ユーザプログラムが実行されます。

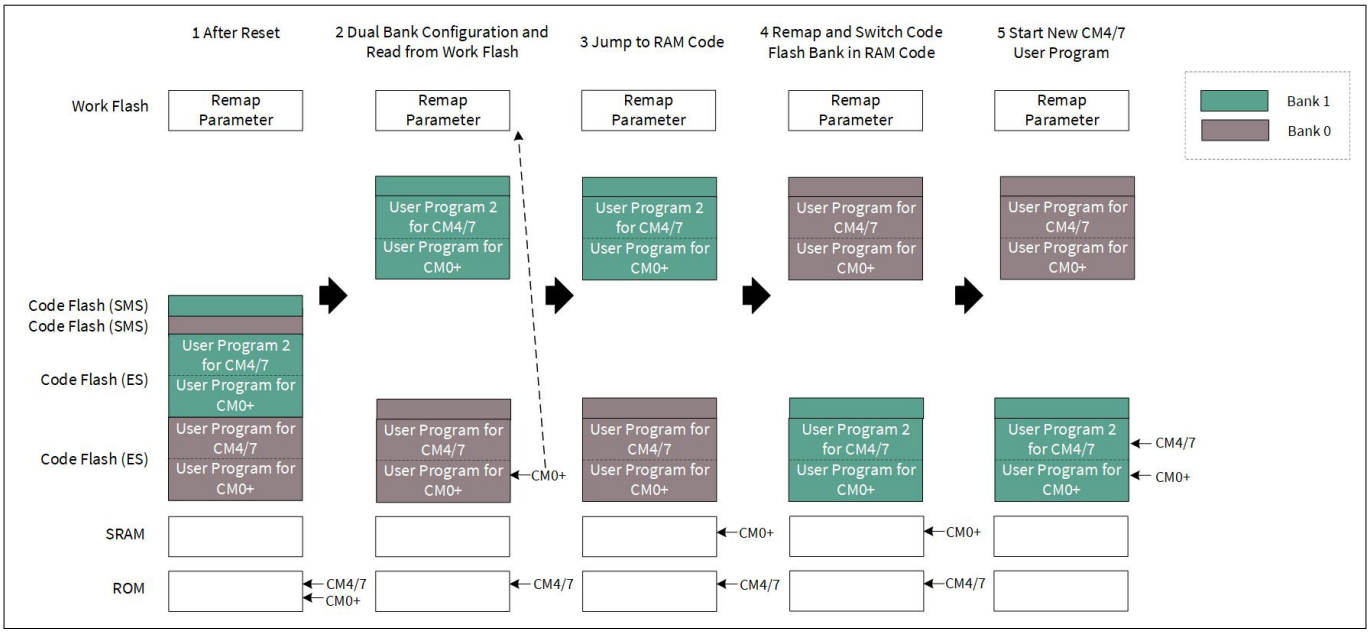

図 9　　新しい CM4/7 ユーザプログラムコードの開始

4.2　デュアルバンクおよび Remap 機能を用いたフラッシュメモリの書換え手順

ここでは、デュアルバンクモードと Remap 機能を使用したフラッシュ書換え手順の例を示します。この例では、LED ON/OFF を制御する CM4/7 のユーザプログラムがファームウェアアップデート要求によって更新される手順例を示します。ワークフラッシュ SA0 セクタは Remap パラメータの格納先として使用され、CM0+ユーザコードはワークフラッシュからデータを読み出し、マップ切替えを設定します。ワークフラッシュから読み出されたデータが “0xAAAA_AAAA” の場合、マッピング A のコード (ファームウェアアップデート前のコード) が実行されます。読み出したデータが “0xBBBB_BBBB” の場合、マッピング B のコード (ファームウェアアップデート後のコード) が実行されます。ワークフラッシュに書き込まれた Remap パラメータに依存して、ユーザアプリーションコードが切り替わります。

4 デュアルバンク手法を用いたフラッシュメモリの書換え

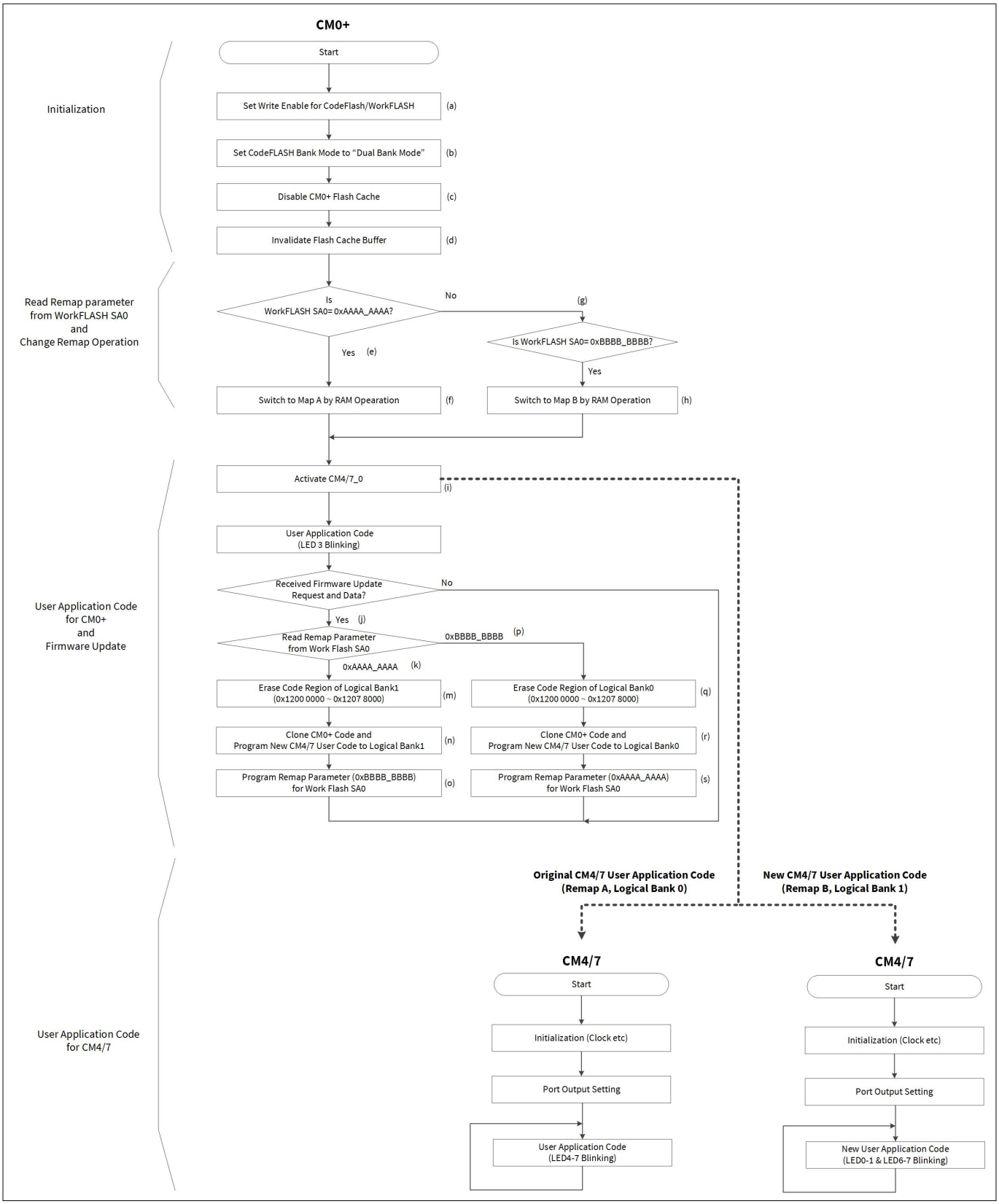

図 10　　新しい CM4/7 ユーザプログラムコードの開始手順

4.2.1　　設定とサンプルコード

フラッシュメモリ書換え手順の SDL の設定部のパラメータを表 6 に、関数を表 7 に示します。

表 6　　フラッシュメモリ書換え手順の設定パラメーター覧

パラメータ	説明	値
flash_remap_config	フラッシュ Remap パラメータ	0
Memory_result	フラッシュメモリフラグ	1
Update_request	ファームウェア更新リクエスト	0

表 7　　フラッシュメモリ書換え手順の関数一覧

関数	説明	備考
Cy_Flashc_MainWriteEnable()	メインフラッシュの書込みを有効にします	Code Listing 2 を参照してください
Cy_Flashc_WorkWriteEnable()	ワークフラッシュの書込みを有効にします	Code Listing 6 を参照してください
Cy_Flashc_SetMainBankMode()	メインフラッシュのバンクモードを設定します	Code Listing 10 を参照してください
Cy_Flashc_CM0_CacheDisable()	フラッシュキャッシュを無効にします	Code Listing 11 を参照してください
Cy_Flashc_InvalidateFlashCacheBuffer()	フラッシュキャッシュとバッファを無効にします	Code Listing 12 を参照してください
EraseSectorWorkSectors_SA0()	ワークフラッシュのセクタ SA0 を消去します	Code Listing 13 を参照してください
EraseCodeFlashLogicalBank()	コードフラッシュロジックバンクを消去します	Code Listing 14 を参照してください
CloneCodeFlashtoBank1_UpdateCM7_CODE()	コードフラッシュをバンク 1 にコピーします	Code Listing 15 を参照してください
ProgramRow_B()	データ B にプログラムします	Code Listing 16 を参照してください
CloneCodeFlashtoBank0_UpdateCM7_CODE()	コードフラッシュをバンク 0 にコピーします	Code Listing 17 を参照してください
ProgramRow_A()	データ A にプログラムします	Code Listing 18 を参照してください

Code Listing 9 は、フラッシュメモリ書換え手順のサンプルプログラムを示します。

Code Listing 9 フラッシュメモリ書換え手順の例

```
uint32_t flash_remap_config = 0;
bool Memory_result = 1;

int main(void)
{
:
    /* Set Write Enable for Code/Work Flash */  /* Enable writing main flash. See Code Listing
2. Enable writing work flash. See Code Listing 6. See (a) of Figure 10. */

    Cy_Flashc_MainWriteEnable();
    Cy_Flashc_WorkWriteEnable();

    /* Set Code Flash Bank Mode to "Dual Bank Mode" */  /* Sets Dual bank mode. See Code
Listing 10. See (b) of Figure 10. */
    Cy_Flashc_SetMainBankMode(CY_FLASH_DUAL_BANK_MODE);

    /* Disable CM0+ Flash Cache. */  /* Disables Flash cache. See Code Listing 11. See (c) of
Figure 10. */
    Cy_Flashc_CM0_CacheDisable();
    /* Invalidate Flash Cache Buffer */
    Cy_Flashc_InvalidateFlashCacheBuffer();  /* Invalidates the flash cache and buffer. See
Code Listing 12. See (d) of Figure 10. */



    /* Read value from 0x14000000 */
    flash_remap_config = *(volatile unsigned long *)0x14000000;

    if(flash_remap_config != 0xAAAAAAAA && flash_remap_config  != 0xBBBBBBBB)
    {
        Memory_result = 0;
        /* clear */
        EraseSectorWorkSectors_SA0();  /* If WorkFlash SA0 is neither 0xAAAAAAAA nor
0xBBBBBBBB, erase WorkFlash SA0. See Code Listing 13. */
    }

    /* Read Remap parameter from WorkFlash SA0 and Change Remap Operation */
    if(flash_remap_config == 0xAAAAAAAA)
    {
        /* Switch to Map A */
        //SwitchMapA();  /* Read WorkFlash SA0 =0xAAAAAAAA. Switch to Flash Map A. See Code
Listing 10. See (e) and (f) of Figure 10. */
        Cy_Flashc_SetMainBankMode(CY_FLASH_MAPPING_A);
    }
    else if(flash_remap_config == 0xBBBBBBBB)
    {
        /* Switch to Map B */
        //SwitchMapB();    /* Read WorkFlash SA0 =0xAAAAAAAA. Switch to Flash Map A. See Code
Listing 10. See (g) and (h) of Figure 10. */
        Cy_Flashc_SetMainBankMode(CY_FLASH_MAPPING_B);
    }
```

```
    /* Enable CM7_0. CY_CORTEX_M7_APPL_ADDR is calculated in linker script, check it in case of
problems. */
    Cy_SysEnableApplCore(CORE_CM7_0, CY_CORTEX_M7_0_APPL_ADDR);  /* Active CM7_0. See (i) of
Figure 10. */


    /* The flag "Update_request" will change to 1 after push button*/
    int Update_request = 0;


    if((Update_request == 1) && (Cy_GPIO_Read(USER_BUTTON_PORT, USER_BUTTON_PIN) == 1/* avoid
to button long presses */ ))  /* Update request. See (j) of Figure 10. */
    {
        /* Flag clear */
        Update_request = 0;

        if(Memory_result == 0 || flash_remap_config == 0xAAAAAAAA)
        {
            /* MAP A BANK 1: 0x12000000 */

            /* Erase Code Region of Logical Bank 1(0x1200 0000~ 0x12078000) */  /* Read
WorkFlash SA0 =0xAAAAAAAA. Erase Code Flash Logical Bank 1. See Code Listing 14. See (k) and
(m) of Figure 10. */
            EraseCodeFlashLogicalBank();
            /* Clone CM0+ Code and Program New CM4/7
User Code to Logical Bank1(0x1200 0000~ 0x121F8000) */  /* Copy to Bank 1. See Code Listing 15.
See (n) of Figure 10. */
            CloneCodeFlashtoBank1_UpdateCM7_CODE();
            /* Program Remap Parameter (0xBBBB BBBB) for Work Flash SA0 */
            EraseSectorWorkSectors_SA0();
            ProgramRow_B();  /* Program Data B. See Code Listing 16. See (o) of Figure 10. */
        }
        else
        {
            if(flash_remap_config  == 0xBBBBBBBB)
            {
                /* MAP B BANK 0: 0x12000000 */
                /* Read WorkFlash SA0 =0xBBBBBBBB. Erase Code Flash Logical Bank 0. See  Code
Listing 14, Code Listing 18. See (p), (q) of Figure 10. */
                /* Erase Code Region of Logical Bank 0(0x1200 0000~ 0x12078000) */
                EraseCodeFlashLogicalBank();
                /* Clone CM0+ Code and Program New CM4/7
User Code to Logical Bank0(0x1200 0000~ 0x121F8000) */  /* Copy to Bank 0. See Code Listing 17.
See (r) of Figure 10. */
                CloneCodeFlashtoBank0_UpdateCM7_CODE();
                /* Program Remap Parameter (0xAAAA_AAAA) for Work Flash SA0 */
                EraseSectorWorkSectors_SA0();
                ProgramRow_A();  /* Program Data A. See Code Listing 18. See (s) of Figure 10.
*/
            }
        }
```

Code Listing 10 Cy_Flashc_SetMainBankMode() 関数

```
__STATIC_INLINE void Cy_Flashc_SetMainBankMode(cy_en_bankmode_t mode)  /* Sets bank mode for
main flash. */
{
   FLASHC->unFLASH_CTL.stcField.u1MAIN_BANK_MODE = mode;
   FLASHC->unFLASH_CTL.u32Register;
}
```

Code Listing 11 Cy_Flashc_CM0_CacheDisable() 関数

```
__STATIC_INLINE void Cy_Flashc_CM0_CacheDisable(void)  /* Disables Flash cache. */
{
    FLASHC->unCM0_CA_CTL0.stcField.u1CA_EN = 0ul;
}
```

Code Listing 12 Cy_Flashc_InvalidateFlashCacheBuffer() 関数

```
__STATIC_INLINE void Cy_Flashc_InvalidateFlashCacheBuffer(void)  /* Invalidates the flash cache
and buffer. */
{
    FLASHC->unFLASH_CMD.stcField.u1INV = 0x1ul;

    // Wait for completion (HW will clear bit)
    while(FLASHC->unFLASH_CMD.stcField.u1INV != 0);
}
```

Code Listing 13 EraseSectorWorkSectors_SA0() 関数

```
void EraseSectorWorkSectors_SA0(void)  /* Erase workflash sector SA0. */
{
    uint32_t status = CY_FLASH_DRV_SUCCESS;

    /* 1.1. Erase workflash sector SA0    */
    status = Cy_Flash_EraseSector(&FlashContext, &eraseSectorConfigSA0,
CY_FLASH_DRIVER_BLOCKING);
    /* 1.2. Status check    */
    CY_ASSERT(status == CY_FLASH_DRV_SUCCESS);
    /* 1.3. Verify    */
    status = Cy_Flash_BlankCheck(&FlashContext, &blankCheckConfig, CY_FLASH_DRIVER_BLOCKING);
    CY_ASSERT(status == CY_FLASH_DRV_SUCCESS);
}
```

Code Listing 14 EraseCodeFlashLogicalBank() 関数

```
void EraseCodeFlashLogicalBank(void)  /* Erase Code Flash Logical Bank. */
{
    uint32_t status = CY_FLASH_DRV_SUCCESS;

    for(uint32_t addrOffset = 0; addrOffset < CODE_FLASH_REGION_SIZE; addrOffset +=
CODE_FLASH_SECTOR_SIZE)
    {
        // Set the address to be erased.
        // Note: This firmware always erases logical area 0 from (0x1200_0000 ~ ).
        eraseCodeSectorConfig.Addr = (uint32_t*)(CODE_FLASH_LOGICAL_TOP_ADDR1 + addrOffset);
        // Issuing the erase sector
        status = Cy_Flash_EraseSector(&FlashContext, &eraseCodeSectorConfig,
CY_FLASH_DRIVER_BLOCKING);
        // Checking the result
        CY_ASSERT(status == CY_FLASH_DRV_SUCCESS);
    }
}
```

Code Listing 15 CloneCodeFlashtoBank1_UpdateCM7_CODE() 関数

```
void CloneCodeFlashtoBank1_UpdateCM7_CODE(void)  /* Copy Code Flash to Bank 1. */
{

    uint32_t status = CY_FLASH_DRV_SUCCESS;

    /* 2.1. Program flash whole sector    */
    for(uint32_t addrOffset = 0; addrOffset < SIZE; addrOffset += BYTE_NUM_OF_4096BIT /
*0x1000*/)
    {
        // Copy the Flash contents into temporaly buffer
        memcpy((void*)bufToBePrograMed, (void*)(CODE_FLASH_LOGICAL_TOP_ADDR0/* 0x10000000*/ +
addrOffset), BYTE_NUM_OF_4096BIT);

        ProgramRow.destAddr = (uint32_t*)(CODE_FLASH_LOGICAL_TOP_ADDR1 + addrOffset);
        ProgramRow.dataAddr = (uint32_t*)&bufToBePrograMed[addrOffset];

        status = Cy_Flash_ProgramRow(&FlashContext, &ProgramRow, CY_FLASH_DRIVER_BLOCKING);
        CY_ASSERT(status == CY_FLASH_DRV_SUCCESS);
    }

    /* CM7 data upodat */
    for(uint32_t i = 0; i<2 ;i+=1)
    {
        uint32_t ADDR_LED = 0x10080410;
        programRow_LED.destAddr = (uint32_t*)(ADDR_LED + (i<<3));
        programRow_LED.dataAddr = (uint32_t*)&CM7_DATA2[i<<3];
        programRow_LED.dataSize = CY_FLASH_PROGRAMROW_DATA_SIZE_64BIT;
        status = Cy_Flash_ProgramRow(&FlashContext, &programRow_LED, CY_FLASH_DRIVER_BLOCKING);
        CY_ASSERT(status == CY_FLASH_DRV_SUCCESS);
    }

}
```

Code Listing 16 ProgramRow_B() 関数

```
void ProgramRow_B(void)
{
    uint32_t status = CY_FLASH_DRV_SUCCESS;  /* Program Data B. */

    /* 2.1. Program flash whole sector    */
    for(uint32_t offsetAddr = 0; offsetAddr < TEST_ERASE_SEC_SIZE; offsetAddr +=
TEST_PROGRAM_SIZE)
    {
        programRowConfigSA0.destAddr = (uint32_t*)(TEST_FLASH_ADDR + offsetAddr);
        programRowConfigSA0.dataAddr = writeData_B;
        status = Cy_Flash_ProgramRow(&FlashContext, &programRowConfigSA0,
CY_FLASH_DRIVER_BLOCKING);
        CY_ASSERT(status == CY_FLASH_DRV_SUCCESS);
    }
}
```

Code Listing 17 CloneCodeFlashtoBank0_UpdateCM7_CODE() 関数

```
void CloneCodeFlashtoBank0_UpdateCM7_CODE(void)  /* Copy Code Flash to Bank 0. */
{
    uint32_t status = CY_FLASH_DRV_SUCCESS;
    /* 2.1. Program flash whole sector    */
    for(uint32_t addrOffset = 0; addrOffset < SIZE ;addrOffset += BYTE_NUM_OF_4096BIT /
*0x1000*/)
    {
        // Copy the Flash contents into temporaly buffer
        memcpy((void*)bufToBePrograMed, (void*)(CODE_FLASH_LOGICAL_TOP_ADDR0 + addrOffset),
BYTE_NUM_OF_4096BIT);

        ProgramRow.destAddr = (uint32_t*)(CODE_FLASH_LOGICAL_TOP_ADDR1 + addrOffset);
        ProgramRow.dataAddr = (uint32_t*)&bufToBePrograMed[addrOffset];
        status = Cy_Flash_ProgramRow(&FlashContext, &ProgramRow, CY_FLASH_DRIVER_BLOCKING);
        CY_ASSERT(status == CY_FLASH_DRV_SUCCESS);
    }
    /* CM7 data upodat */
    uint32_t ADDR_LED = 0x10080410;
    for(uint32_t i = 0; i<2 ;i+=1)
    {
        programRow_LED.destAddr = (uint32_t*)(ADDR_LED + (i<<3));
        programRow_LED.dataAddr = (uint32_t*)&CM7_DATA1[i<<3];
        programRow_LED.dataSize = CY_FLASH_PROGRAMROW_DATA_SIZE_64BIT;
        status = Cy_Flash_ProgramRow(&FlashContext, &programRow_LED, CY_FLASH_DRIVER_BLOCKING);
        CY_ASSERT(status == CY_FLASH_DRV_SUCCESS);
    }
}
```

Code Listing 18 ProgramRow_A() 関数

```
void ProgramRow_A(void)  /* Program Data A. */
{
    uint32_t status = CY_FLASH_DRV_SUCCESS;

    /* 2.1. Program flash whole sector    */
    for(uint32_t offsetAddr = 0; offsetAddr < TEST_ERASE_SEC_SIZE; offsetAddr +=
TEST_PROGRAM_SIZE)
    {
        programRowConfigSA0.destAddr = (uint32_t*)(TEST_FLASH_ADDR + offsetAddr);
        programRowConfigSA0.dataAddr = writeData_A;
        status = Cy_Flash_ProgramRow(&FlashContext, &programRowConfigSA0,
CY_FLASH_DRIVER_BLOCKING);
        CY_ASSERT(status == CY_FLASH_DRV_SUCCESS);
    }
    /* 2.2. Veryfy    */
    for(uint32_t offsetAddr = 0; offsetAddr < TEST_ERASE_SEC_SIZE; offsetAddr += 4)
    {
        CY_ASSERT(*(uint32_t*)(TEST_FLASH_ADDR + offsetAddr) == DATA_TO_BE_PROGRAMED_A);
    }
}
```

5 ワークフラッシュ読出し手順

TRAVEO™ T2G ファミリでは、ワークフラッシュをさまざまな CPU コア経由で読み出せます。ただし、1 つの CPU だけに修正不可能な ECC エラー処理を割り当てられます。そのエラーを処理する CPU コアは他の CPU コアにそのエラーを通知する必要があります。しかし、これには時間がかかります。さらに、CYT3/CYT4/CYT6 シリーズでは、AXI バス (64 ビット) を介したワークフラッシュの読出しは、8 ビット, 16 ビット, または 32 ビットのロード命令が使用されている場合でも、常に 64 ビットすべての ECC エラーをチェックします。これにより、アドレス指定されたメモリに ECC エラーがない場合でも、ECC エラーが発生する可能性があります。したがって、ワークフラッシュの読出しには AHB (32 ビット) DMA (M-DMA または P-DMA) チャネルを使用することを推奨します。各 CPU コアには DMA チャネルを割り当てる必要があります。DMA チャネルは、ECC ダブルビットエラーを含む、修正不可能なエラーを検出できます。DMA は、障害のあるアクセスが発生した個々のチャネルのためにバスエラーを報告します。エラー検出は 32 ビット単位で行われます。

アプリケーションが ECC エラーと他のエラーを区別する必要がある場合、1 つの CPU だけで Fault Structure を処理します。同じ Fault を複数のコアの Fault Structure に確実にルーティングすることはできません。

ここでは、AHB (M-DMA または P-DMA) チャネルを使用したワークフラッシュの読出し手順の例を示します。

5.1 DMA 経由のワークフラッシュ読出し手順

図 11 に DMA 経由のワークフラッシュ読出し手順の例を示します。

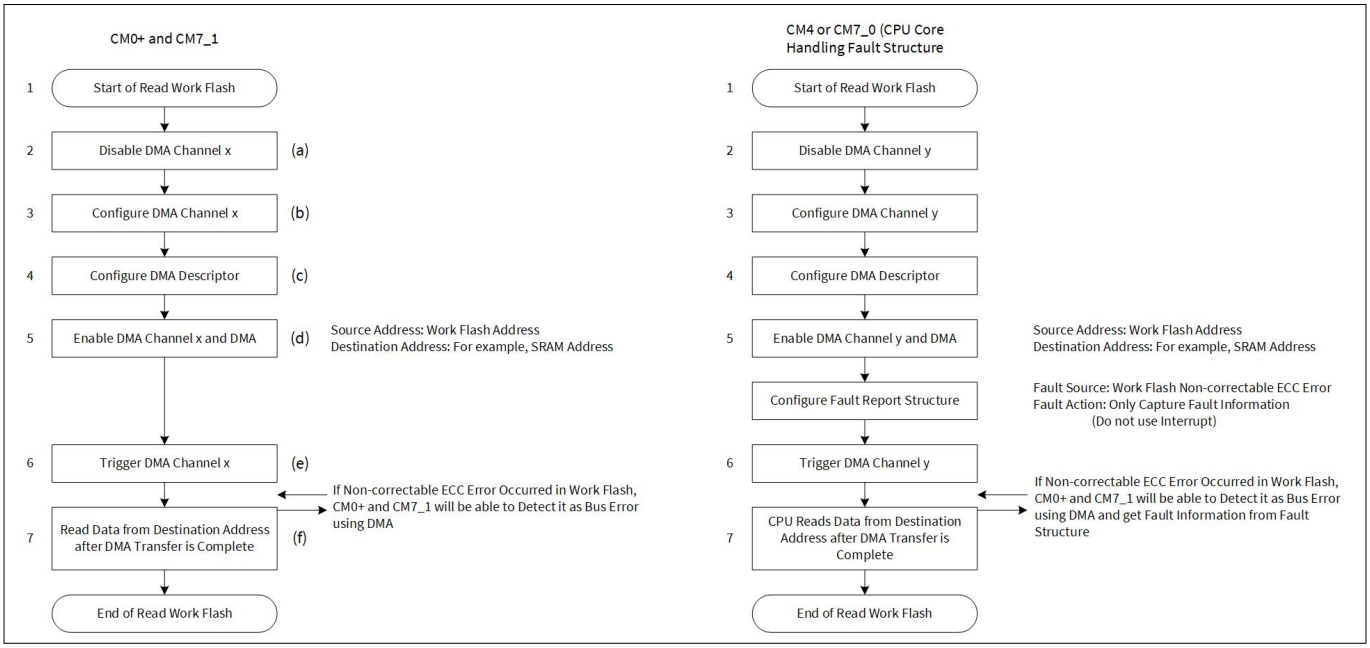

図 11 DMA 経由のワークフラッシュ読出し手順

1. DMA チャネルを無効にしてください。
2. 設定チャネルに対応するチャネル優先度やポインタアドレスなどを DMA チャネルに設定してください。
3. ディスクリプタタイプ、転送元アドレス (ワークフラッシュアドレス)、転送先アドレスなどを DMA ディスクリプタに設定してください。
4. DMA チャネル設定後、DMA チャネルと DMA を有効にしてください。
5. フォールトソース (ワークフラッシュの修正不可能な ECC エラー) やフォールトアクション (フォールト情報のみを取得; 割込み未使用) などを Fault Report Structure に設定してください。
6. DMA チャネルをトリガしてください。例えば、トリガマルチプレクサ経由でソフトウェアトリガを使用してください。
7. DMA 転送完了後、CPU が転送先アドレスからデータを読み出します。

もし、ワークフラッシュにて修正不可能な ECC エラーが発生した場合、各 CPU は DMA を使用することで、それをバスエラーとして検出できます。Fault Structure を処理する CPU コアは、Fault Structure からフォールト情報を得られます。

DMA とフォールト設定手順の詳細については、関連資料の AN219842, AN220191, および registers TRM を参照してください。

5.1.1 設定とサンプルコード

DMA 経由のワークフラッシュ読出し手順の SDL の設定部のパラメータを表 8 に、関数を表 9 に示します。

表 8 DMA 経由のワークフラッシュ読出し手順の設定パラメーター覧

パラメータ	説明	値
BUFFER_SIZE	バッファサイズを定義します	36
DMAC_CHANNEL	M-DMA チャネルを定義します	0
DMAC_SW_TRIG	SW トリガを定義します	TRIG_OUT_MUX_3_MDMA_TR_IN0
.MDMA_Descriptor	M-DMA のディスクリプタポインタ	&stcDescr
.preemptable	チャネルのプリエンプタブル	0ul
.priority	チャネルの優先度	0ul
.enable	チャネル イネーブル	1ul
.deact	DESCR_CTL WAIT_FOR_DEACT	0ul
.intrType	DESCR_CTL INTR_TYPE	CY_MDMA_INTR_DESCR_CMPLT
.trigoutType	DESCR_CTL TR_OUT_TYPE	CY_MDMA_TRIGOUT_DESCR_CMPLT
.dataPrefetch	DESCR_CTL DATA_PREFETCH	0ul
.chStateAtCmplt	DESCR_CTL CH_DISABLE	CY_MDMA_CH_DISABLED
.triginType	DESCR_CTL TR_IN_TYPE	CY_MDMA_TRIGIN_DESCR
.dataSize	DESCR_CTL DATA_SIZE	CY_MDMA_BYTE
.srcTxfrSize	DESCR_CTL SRC_TRANSFER_SIZE	0ul
.destTxfrSize	DESCR_CTL DST_TRANSFER_SIZE	0ul
.descrType	DESCR_TYPE	CY_MDMA_MEMORY_COPY_TRANSFER
.srcAddr	DESCR_SRC	au8SrcBuffer
.destAddr	DESCR_DST	au8DestBuffer
.xCount	DESCR_X_CTL X_COUNT	BUFFER_SIZE
.descrNext	DESCR_NEXT_PTR	NULL

表 9 DMA 経由のワークフラッシュ読出し手順の関数一覧

関数	説明	備考
Cy_MDMA_Disable()	M-DMA ディセーブルの設定	Code Listing 20 を参照してください

(続く)

表 9 (続き) DMA 経由のワークフラッシュ読出し手順の関数一覧

関数	説明	備考
Cy_MDMA_Chnl_DeInit()	チャネルに対応するレジスタのすべてのコンテンツをクリアします	Code Listing 21 を参照してください
Cy_MDMA_Chnl_Init()	M-DMA チャネル初期化を設定します	Code Listing 22 を参照してください
Cy_MDMA_Descr_Init()	M-DMA ディスクリプタ初期化を設定します	Code Listing 23 を参照してください
Cy_MDMA_Chnl_Enable()	M-DMA チャネルイネーブル化を設定します	Code Listing 24 を参照してください
Cy_MDMA_Enable()	M-DMA イネーブルを設定します	Code Listing 25 を参照してください
Cy_TrigMux_SwTrigger()	ソフトウェアトリガを生成します	Code Listing 26 を参照してください

Code Listing 19 は、フラッシュ書換え手順のサンプルプログラムを示します。

5 ワークフラッシュ読出し手順

Code Listing 19 DMA 経由のワークフラッシュ読出し手順の例

```
#define BUFFER_SIZE     (36ul)  /* Defines buffer size. */
#define DMAC_CHANNEL    (0)  /* Defines M-DMA. */
#define DMAC_SW_TRIG    (TRIG_OUT_MUX_3_MDMA_TR_IN0)   /* Defines SW trigger */
/* Configure M-DMA channel. */
static const   cy_stc_mdma_chnl_config_t   chnlConfig =
{
    .MDMA_Descriptor = &stcDescr,
    .preemptable     = 0ul,
    .priority        = 0ul,
    .enable          = 1ul,         /* enabled after initialization */
};
static const cy_stc_mdma_descr_config_t  stcDmaDescrConfig =  /* Configure M-DMA descriptor. */
{
    .deact          = 0ul,
    .intrType       = CY_MDMA_INTR_DESCR_CMPLT,
    .trigoutType    = CY_MDMA_TRIGOUT_DESCR_CMPLT,
    .dataPrefetch   = 0ul,
    .chStateAtCmplt = CY_MDMA_CH_DISABLED,
    .triginType     = CY_MDMA_TRIGIN_DESCR,
    .dataSize       = CY_MDMA_BYTE,
    .srcTxfrSize    = 0ul,
    .destTxfrSize   = 0ul,
    .descrType      = CY_MDMA_MEMORY_COPY_TRANSFER,
    .srcAddr        = au8SrcBuffer,
    .destAddr       = au8DestBuffer,
    .xCount         = BUFFER_SIZE,
    .descrNext      = NULL
};

int main(void)
{
:
    /* Disable DMA CH0  */  /* Disable M-DMA. See Code Listing 20. See (a) of Figure 11. Clears
all the content of registers corresponding to the channel. See Code Listing 21. */
    Cy_MDMA_Disable(DMAC);
    Cy_MDMA_Chnl_DeInit(DMAC, DMAC_CHANNEL);
    /* Configure DMA CH0 */  /* Configures M-DMA channel initialize. See Code Listing 22. See
(b) of Figure 11. */
    Cy_MDMA_Chnl_Init(DMAC, DMAC_CHANNEL, &chnlConfig);
    /* Configure DMA Descriptor */  /* Configures M-DMA descriptor initialize. See Code Listing
23. See (c) of Figure 11. */
    Cy_MDMA_Descr_Init(&stcDescr,&stcDmaDescrConfig);
    /* Enable DMA channel 0 and DMA */  /*Configures M-DMA channel enable. See Code Listing 24.
Configures M-DMA enable. See Code Listing 25. See (d) of Figure 11. */
    Cy_MDMA_Chnl_Enable(DMAC, DMAC_CHANNEL);
    Cy_MDMA_Enable(DMAC);

    /* SW Trigger */  /* Generates a Software trigger. See Code Listing 26. See (e) of Figure
11. */
    Cy_TrigMux_SwTrigger(DMAC_SW_TRIG, TRIGGER_TYPE_EDGE, 1ul);
```

```
    /* Read DATA from destination after DMA transfer completed.*/
    for(uint32_t i = 0ul; i < BUFFER_SIZE; i++)
    {
        READ_DATA[i] = au8DestBuffer[i];  /* CPU read Data. See (f) of Figure 11. */
    }

    /* Fault status read  */
    status = Cy_SysFlt_GetErrorSource(FAULT_STRUCT0);

    for(;;);
}
```

Code Listing 20 Cy_MDMA_Disable() 関数

```
void Cy_MDMA_Disable(volatile stc_DMAC_t *pstcMDMA)  /* Write to M-DMA_CTL_ENABLED bit */
{
    pstcMDMA->unCTL.stcField.u1ENABLED  =       0ul;
}
```

Code Listing 21 Cy_MDMA_Chnl_DeInit() 関数

```
void Cy_MDMA_Chnl_DeInit(volatile stc_DMAC_t *pstcMDMA, uint32_t chNum)  /* Clears all the
content of registers corresponding to the channel bit */
{
    pstcMDMA->CH[chNum].unCTL.u32Register               = 0ul;
    pstcMDMA->CH[chNum].unIDX.u32Register               = 0ul;
    pstcMDMA->CH[chNum].unCURR.u32Register              = 0ul;
    pstcMDMA->CH[chNum].unINTR_MASK.u32Register         = 0ul;
    pstcMDMA->CH[chNum].unINTR_SET.u32Register          = 0ul;
}
```

5 ワークフラッシュ読出し手順

Code Listing 22 Cy_MDMA_Chnl_Init() 関数

```
cy_en_mdma_status_t Cy_MDMA_Chnl_Init(volatile stc_DMAC_t *pstcMDMA, uint32_t chNum, const
cy_stc_mdma_chnl_config_t* chnlConfig)  /* Configures M-DMA channel initialize */
{
    cy_en_mdma_status_t retVal = CY_MDMA_ERR_UNC;

    if ((pstcMDMA != NULL) && (chnlConfig != NULL))
    {
        /* Set current descriptor */
        pstcMDMA->CH[chNum].unCURR.u32Register   =  (uint32_t)chnlConfig->MDMA_Descriptor;

        /* Set if the channel is preemptable  */
        /* There is no the parameter in MDMA */

        /* Set channel priority */
        pstcMDMA->CH[chNum].unCTL.stcField.u2PRIO    =   chnlConfig->priority;

        /* Set enabled status */
        pstcMDMA->CH[chNum].unCTL.stcField.u1ENABLED =   chnlConfig->enable;

        retVal = CY_MDMA_SUCCESS;
    }
    else
    {
        retVal = CY_MDMA_INVALID_INPUT_PARAMETERS;
    }

    return (retVal);
}
```

Code Listing 23 Cy_MDMA_Descr_Init() 関数

```
cy_en_mdma_status_t Cy_MDMA_Descr_Init(cy_stc_mdma_descr_t* descriptor, const
cy_stc_mdma_descr_config_t* config)  /* Configures M-DMA descriptor */
{
    cy_en_mdma_status_t retVal = CY_MDMA_ERR_UNC;

    if ((descriptor != NULL) && (config != NULL))
    {
        /* Descriptor[0] */
        descriptor->unMDMA_DESCR_CTL.stcField.u2WAIT_FOR_DEACT     =       config->deact;
        descriptor->unMDMA_DESCR_CTL.stcField.u2INTR_TYPE          =       config->intrType;
        descriptor->unMDMA_DESCR_CTL.stcField.u2TR_OUT_TYPE        =       config->trigoutType;
        descriptor->unMDMA_DESCR_CTL.stcField.u2TR_IN_TYPE         =       config->triginType;
        descriptor->unMDMA_DESCR_CTL.stcField.u1DATA_PREFETCH      =       config->dataPrefetch;
        descriptor->unMDMA_DESCR_CTL.stcField.u2DATA_SIZE          =       config->dataSize;
        descriptor->unMDMA_DESCR_CTL.stcField.u1CH_DISABLE         =       config-
>chStateAtCmplt;
        descriptor->unMDMA_DESCR_CTL.stcField.u1SRC_TRANSFER_SIZE  =       config->srcTxfrSize;
        descriptor->unMDMA_DESCR_CTL.stcField.u1DST_TRANSFER_SIZE  =       config->destTxfrSize;
        descriptor->unMDMA_DESCR_CTL.stcField.u3DESCR_TYPE         =       config->descrType;

        /* Descriptor[1] */
        descriptor->u32MDMA_DESCR_SRC                              =       (uint32_t)config-
>srcAddr;

        /* after 3rd word of descriptor depends on descriptor type */

        switch(config->descrType)
        {
        case (uint32_t)CY_MDMA_SINGLE_TRANSFER:
        {
            /* Descriptor[2] */
            descriptor->u32MDMA_DESCR_DST                          =       (uint32_t)config-
>destAddr;

            /* Descriptor[3] -> NEXT_PTR */
            descriptor->unMDMA_DESCR_X_SIZE.u32Register            =       (uint32_t)config-
>descrNext;
            break;
        }
        case (uint32_t)CY_MDMA_1D_TRANSFER:
        {
            /* Descriptor[2] */
            descriptor->u32MDMA_DESCR_DST                          =       (uint32_t)config-
>destAddr;

            /* Descriptor[3] */
            descriptor->unMDMA_DESCR_X_SIZE.stcField.u16X_COUNT    =       (uint32_t)((config-
>xCount) - 1ul);

            /* Descriptor[4] */
            descriptor->unMDMA_DESCR_X_INCR.stcField.u16SRC_X_INCR =       (uint32_t)config-
```

```
>srcXincr;
            descriptor->unMDMA_DESCR_X_INCR.stcField.u16DST_X_INCR  =       (uint32_t)config-
>destXincr;

            /* Descriptor[5] -> NEXT_PTR */
            descriptor->unMDMA_DESCR_Y_SIZE.u32Register             =       (uint32_t)config-
>descrNext;
            break;
        }
        case (uint32_t)CY_MDMA_2D_TRANSFER:
        {
            /* Descriptor[2] */
            descriptor->u32MDMA_DESCR_DST                           =       (uint32_t)config-
>destAddr;

            /* Descriptor[3] */
            descriptor->unMDMA_DESCR_X_SIZE.stcField.u16X_COUNT     =       (uint32_t)((config-
>xCount) - 1ul);

            /* Descriptor[4] */
            descriptor->unMDMA_DESCR_X_INCR.stcField.u16SRC_X_INCR  =       (uint32_t)config-
>srcXincr;
            descriptor->unMDMA_DESCR_X_INCR.stcField.u16DST_X_INCR  =       (uint32_t)config-
>destXincr;

            /* Descriptor[5] */
            descriptor->unMDMA_DESCR_Y_SIZE.stcField.u16Y_COUNT     =       (uint32_t)((config-
>yCount) - 1ul);

            /* Descriptor[6] */
            descriptor->unMDMA_DESCR_Y_INCR.stcField.u16SRC_Y_INCR  =       (uint32_t)config-
>srcYincr;
            descriptor->unMDMA_DESCR_Y_INCR.stcField.u16DST_Y_INCR  =       (uint32_t)config-
>destYincr;

            /* Descriptor[7] */
            descriptor->u32MDMA_DESCR_NEXT_PTR                      =       (uint32_t)config-
>descrNext;
            break;
        }
        case (uint32_t)CY_MDMA_MEMORY_COPY_TRANSFER:
        {
            /* Descriptor[2] */
            descriptor->u32MDMA_DESCR_DST                           =       (uint32_t)config-
>destAddr;

            /* Descriptor[3] */
            descriptor->unMDMA_DESCR_X_SIZE.stcField.u16X_COUNT     =       (uint32_t)((config-
>xCount) - 1ul);

            /* Descriptor[4] -> NEXT_PTR */
            descriptor->unMDMA_DESCR_X_INCR.u32Register             =       (uint32_t)config-
>descrNext;
```

```
            break;
        }
        case (uint32_t)CY_MDMA_SCATTER_TRANSFER:
        {
            /* Descriptor[2] -> X_SIZE */
            descriptor->u32MDMA_DESCR_DST                           =       (uint32_t)((config-
>xCount) - 1ul);

            /* Descriptor[3] -> NEXT_PTR */
            descriptor->unMDMA_DESCR_X_SIZE.u32Register             =       (uint32_t)config-
>descrNext;
            break;
        }
        default:
        {
            /* Unsupported type of descriptor */
            break;
        }
        }

        retVal = CY_MDMA_SUCCESS;
    }
    else
    {
        retVal = CY_MDMA_INVALID_INPUT_PARAMETERS;
    }

    return retVal;
}
```

Code Listing 24 Cy_MDMA_Chnl_ Enable() 関数

```
__STATIC_INLINE void Cy_MDMA_Chnl_Enable(volatile stc_DMAC_t *pstcMDMA, uint32_t chNum)  /*
Configures M-DMA channel enable. */
{
    pstcMDMA->CH[chNum].unCTL.stcField.u1ENABLED = 1ul;
}
```

Code Listing 25 Cy_MDMA_ Enable() 関数

```
void Cy_MDMA_Enable(volatile stc_DMAC_t *pstcMDMA)  /* Write to M-DMA_CTL_ENABLED bit */
{
    pstcMDMA->unCTL.stcField.u1ENABLED  =       1ul;
}
```

Code Listing 26 Cy_TrigMux_SwTrigger() 関数

```
cy_en_trigmux_status_t Cy_TrigMux_SwTrigger(uint32_t trigLine, en_trig_type_t trigType,
uint32_t outSel)  /* Generates a Software trigger */
{
    cy_en_trigmux_status_t retVal = CY_TRIGMUX_INVALID_STATE;

    if (PERI->unTR_CMD.stcField.u1ACTIVATE == 0)
    {
        PERI->unTR_CMD.stcField.u8TR_SEL     = Cy_TrigMux_GetNo(trigLine);
        PERI->unTR_CMD.stcField.u5GROUP_SEL  = Cy_TrigMux_GetGroup(trigLine);
        PERI->unTR_CMD.stcField.u1TR_EDGE    = trigType;
        PERI->unTR_CMD.stcField.u1OUT_SEL    = outSel;
        PERI->unTR_CMD.stcField.u1ACTIVATE   = 1;

        retVal = CY_TRIGMUX_SUCCESS;
    }
    return retVal;
}
```

6 用語と略語

AHB	Advanced high-performance bus
AXI	Advanced extensible interface
eCT	Embedded charge-trap
ECC	Error correcting code (エラー訂正コード)
OTA	Over-the-air (無線)
RAM	Random access memory
SRAM	Static random-access memory
P-DMA	Peripheral DMA
M-DMA	Memory DMA
IPC	Inter-processer communication
IRQ	Interrupt request (割込み要求)
DAP	Debug access port
RWW	Read-while-write

7 関連資料

[1] デバイスデータシート

- CYT2B7 datasheet 32-bit Arm® Cortex®-M4F microcontroller TRAVEO™ T2G family
- CYT2B9 datasheet 32-bit Arm® Cortex®-M4F microcontroller TRAVEO™ T2G family
- CYT4BF datasheet 32-bit Arm® Cortex®-M7 microcontroller TRAVEO™ T2G family
- CYT4DN datasheet 32-bit Arm® Cortex®-M7 microcontroller TRAVEO™ T2G family (Doc No. 002-24601)
- CYT3BB/4BB datasheet 32-bit Arm® Cortex®-M7 microcontroller TRAVEO™ T2G family
- CYT3DL datasheet 32-bit Arm® Cortex®-M7 microcontroller TRAVEO™ T2G family (Doc No. 002-27763)
- CYT4EN datasheet 32-bit Arm® Cortex®-M7 microcontroller TRAVEO™ T2G family (Doc No. 002-30842)
- CYT2CL datasheet 32-bit Arm® Cortex®-M4F microcontroller TRAVEO™ T2G family (Doc No. 002- 32508)
- CYT6BJ datasheet 32-bit Arm® Cortex®-M7 microcontroller TRAVEO™ T2G family (Doc No. 002-33466)

[2] ボディコントローラ Entry ファミリ

- TRAVEO™ T2G automotive body controller entry family architecture technical reference manual (TRM)
- TRAVEO™ T2G automotive body controller entry registers technical reference manual (TRM) for CYT2B7
- TRAVEO™ T2G automotive body controller entry registers technical reference manual (TRM) for CYT2B9

[3] ボディコントローラ High ファミリ

- TRAVEO™ T2G automotive body controller high family architecture technical reference manual (TRM)
- TRAVEO™ T2G automotive body controller high registers technical reference manual (TRM) for CYT4BF
- TRAVEO™ T2G automotive body controller high registers technical reference manual (TRM) for CYT3BB/4BB
- TRAVEO™ T2G automotive body controller high registers technical reference manual (TRM) for CYT6BJ (Doc No. 002-36068)

[4] Cluster 2D ファミリ

- TRAVEO™ T2G automotive cluster 2D family architecture technical reference manual (TRM) (Doc No. 002-25800)
- TRAVEO™ T2G automotive cluster 2D registers technical reference manual (TRM) for CYT4DN (Doc No. 002-25923)
- TRAVEO™ T2G automotive cluster 2D registers technical reference manual (TRM) for CYT3DL (Doc No. 002-29854)

[5] アプリケーションノート

- AN219842 – TRAVEO™ T2G の割込み使用方法
- AN220191 – TRAVEO™ T2G ファミリ Direct Memory Access (DMA) コントローラの使用方法

8 その他の参考資料

さまざまな周辺機器にアクセスするためのサンプルソフトウェアとしてのスタートアップを含むサンプルドライバライブラリ (SDL) が提供されます。SDL は、公式の AUTOSAR 製品でカバーされないドライバの顧客へのリファレンスとしても機能します。SDL は自動車規格に適合していないため、生産目的で使用できません。このアプリケーションノートのプログラムコードは SDL の一部です。SDL を入手するには、テクニカルサポートに連絡してください。

改訂履歴

版数	発行日	変更内容
**	2018-08-29	これは英語版 002-20242 Rev. **を翻訳した日本語版 002-24287 Rev. **です。
*A	2019-08-22	これは英語版 002-20242 Rev. *A を翻訳した日本語版 002-24287 Rev. *A です。英語版の改訂内容: Changed target parts number (CYT2B/CYT4B series) Added Flash Rewriting Procedure using Dual Bank and Remap Function
*B	2020-05-12	これは英語版 002-20242 Rev. *B を翻訳した日本語版 002-24287 Rev. *B です。英語版の改訂内容: Added a part number CYT2/CYT3/CYT4 series.
*C	2021-02-15	これは英語版 002-20242 Rev. *C を翻訳した日本語版 002-24287 Rev. *C です。英語版の改訂内容: Moved to Infineon Template Changed the document title. Added Work Flash Reading Procedure. Added Application Notes to Related Documents.
*D	2022-07-12	これは英語版 002-20242 Rev. *D を翻訳した日本語版 002-24287 Rev. *D です。英語版の改訂内容: Added example of SDL code and description in all instances.
英語版*E	2023-11-10	本版は英語版のみの発行です。英語版の改訂内容: Template update; no content update
*E	2024-06-13	これは英語版 002-20242 Rev. *F を翻訳した日本語版 002-24287 Rev. *E です。英語版の改訂内容: Added specifications for CYT6BJ

Trademarks

All referenced product or service names and trademarks are the property of their respective owners.

Edition 2024-06-13

Published by

Infineon Technologies AG

81726 Munich, Germany

Do you have a question about any aspect of this document?

Email: erratum@infineon.com

Document reference
IFX-acw1681800151040

重要事項

本文書に記載された情報は、いかなる場合も、条件または特性の保証とみなされるものではありません（「品質の保証」）。

本文に記された一切の事例、手引き、もしくは一般的価値、および／または本製品の用途に関する一切の情報に関し、インフィニオンテクノロジーズ（以下、「インフィニオン」）はここに、第三者の知的所有権の不侵害の保証を含むがこれに限らず、あらゆる種類の一切の保証および責任を否定いたします。

さらに、本文書に記載された一切の情報は、お客様の用途におけるお客様の製品およびインフィニオン製品の一切の使用に関し、本文書に記載された義務ならびに一切の関連する法的要件、規範、および基準をお客様が遵守することを条件としています。

本文書に含まれるデータは、技術的訓練を受けた従業員のみを対象としています。本製品の対象用途への適合性、およびこれら用途に関連して本文書に記載された製品情報の完全性についての評価は、お客様の技術部門の責任にて実施してください。

警告事項

技術的要件に伴い、製品には危険物質が含まれる可能性があります。当該種別の詳細については、インフィニオンの最寄りの営業所までお問い合わせください。

インフィニオンの正式代表者が署名した書面を通じ、インフィニオンによる明示の承認が存在する場合を除き、インフィニオンの製品は、当該製品の障害またはその使用に関する一切の結果が、合理的に人的傷害を招く恐れのある一切の用途に使用することはできないこと予めご了承ください。