TRAVEO™ T2G MCU の保護設定

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は、TRAVEO™ T2G ファミリ MCU の保護ユニットの機能および設定方法について説明します。本書は、さまざまな操作に基づいてシステムセキュリティを強化するためのガイドとして使用できます。また、各保護ユニットの構造, アクセス属性, およびいくつかの使用例についても説明します。

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

TRAVEO™ T2G ファミリ

目次

目次

1 はじめに

このアプリケーションノートでは、TRAVEO™ T2G ファミリ MCU の保護ユニットについて説明します。本シリーズは、複数の 32 ビット Arm® Cortex® CPU ベースの CPUSS, 暗号化コンポーネントとしての enhanced Secure Hardware Extension (eSHE), CAN FD, メモリ, およびアナログとデジタルの周辺機能を 1 チップに搭載します。

保護ユニットは、セキュリティシステム設計の重要な部分であり、さまざまな操作に基づいてセキュリティを強化します。保護ユニットは、特定のプロパティに従ってバス上の転送を許可または制限します。保護違反は、バス転送のアドレス領域, アクセス属性と保護ユニットのアドレス範囲, アクセス属性の不一致によって発生します。

これらのシリーズは、3 種類の保護ユニットを持ちます。

- メモリ保護ユニット (MPU)
- 共有メモリ保護ユニット (SMPU)
- 周辺機能保護ユニット (PPU)

メモリ保護は MPU および SMPU によって、周辺リソース保護は PPU によって行います。

MPU, SMPU, および PPU 保護属性の定義は、一貫したソフトウェアインタフェースを保証するため、Arm® 定義 (メモリ領域とアクセス属性定義の観点から) に従います。

セキュリティ要件によっては、SMPU および PPU レジスタは、システム全体の保護を実行する”セキュア”CPU によって制御される必要があります。

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 reference manual の Protection unit 章を参照してください。

さらに、このアプリケーションノートでは、サンプルドライバライブラリ(SDL)を使用したサンプルコードについて説明します。このアプリケーションノートに記載されているプログラムコードは、SDL に含まれるものです。SDL については、その他の参考資料を参照してください。

SDL には設定部分とドライバの部分があります。設定部分では、主に目的の動作をさせるためのパラメータ値を設定します。ドライバの部分は設定部分のパラメータ値に基づいて各レジスタを設定します。設定部分は、お客様のシステムに合わせて設定できます。これらのサンプルプログラムは CYT2B7 シリーズを使用します。

2 保護ユニット

2.1 保護ユニットの配置

図 1 に、CYT2B シリーズの MPU, SMPU, および PPU の配置を示します。

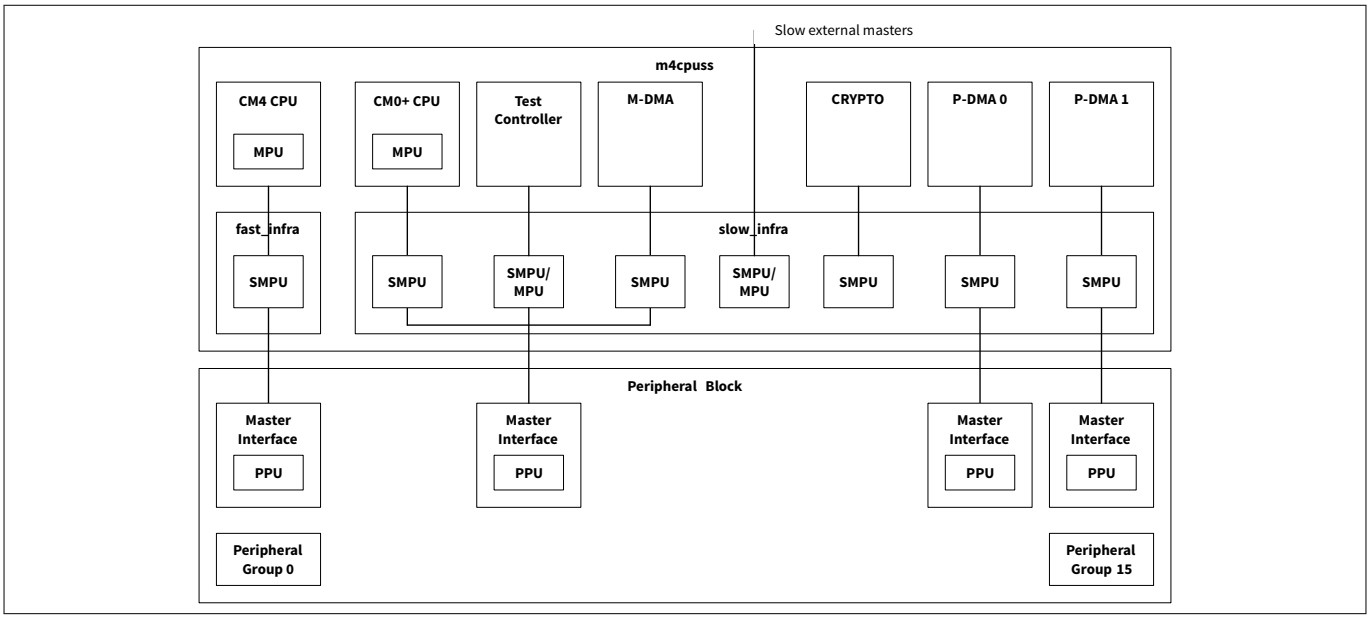

図 1 CYT2B シリーズの保護ユニット配置

その他のシリーズの配置については architecture reference manual を参照してください。

2.2 保護ユニットの概要

MPU は、単一のマスタに関連付けられます。MPU は 2 種類あります。

- CPU の一部として実装される MPU: このタイプは Arm® CPU に実装されます。
- バスインフラストラクチャの一部として実装される MPU: このタイプはテストコントローラなどのバスマスタに実装されます。

ただし、ペリフェラル DMA (P-DMA 0/1), メモリ DMA (M-DMA), および暗号 (CRYPTO) コンポーネントには MPU はありません。これらのマスタは、チャネルやコンポーネントをプログラムしたバス転送のアクセス制御属性を継承します。

SMPU は、すべてのバスマスタによって共有されます。SMPU 領域構造の単一のセットは、システム内のすべての SMPU に同じ保護情報を提供します。

PPU は、すべてのバスマスタによって共有されます。PPU は、ペリフェラルグループ内の周辺機能へのアクセスを制御します。PPU は 2 種類あります。

- Fixed PPU: 保護されるアドレスは固定され、ソフトウェアによる変更はできません。
- Programmable PPU: 保護されるアドレスはソフトウェアによってプログラムできます。

MPU と SMPU は、PPU より高い優先順位を持ちます。さらに、Programmable PPU は、Fixed PPU よりも高い優先順位を持ちます。

保護ユニットの詳細については、architecture reference manual を参照してください。

3 動作概要

3.1 バス転送の保護プロパティ

保護ユニットは、以下のバス転送のプロパティを識別します。

- アクセスするアドレス範囲。MPU, SMPU, および PPU の保護構造の定義は Arm® の定義に従います。したがって、保護ユニットの領域アドレスと領域サイズを設定する場合は以下に注意してください。
  - 領域アドレスのベースアドレスは領域サイズに調整される必要があります。領域サイズが 64 KB の場合、ベースアドレスは 64 KB の倍数 (例えば 0x00010000 または 0x00020000 など) に配置する必要があります。詳細については、registers reference manual を参照してください。
- 以下のような属性にアクセス:
  - リード/ライト: ライトアクセスとリードアクセスを区別します。
  - 実行: データアクセスとコードアクセスを区別します。
  - ユーザ/特権: 特権アクセスとユーザアクセスを区別します。
  - セキュア/非セキュア: セキュアアクセスと非セキュアアクセスを区別します。非セキュア属性の場合、非セキュアアクセスとセキュアアクセスの両方が許可されます。
  - 保護コンテキスト: 異なる保護コンテキストを区別します。

すべてのバスマスタが、これらのアクセス属性すべてを提供するわけではありません。保護コンテキスト属性を持つバスマスタはありません。また Arm® CPU は、セキュア属性を持ちません。

バスマスタによって提供されないアクセス属性は、PROT_MPUx_MS_CTL および PROT_SMPU_MSx_CTL レジスタによって提供されます。これらのレジスタは、ブート処理中またはセキュア CPU によって設定されます。図 2 に、PROT_MPUx_MS_CTL のレジスタ構造を示します。

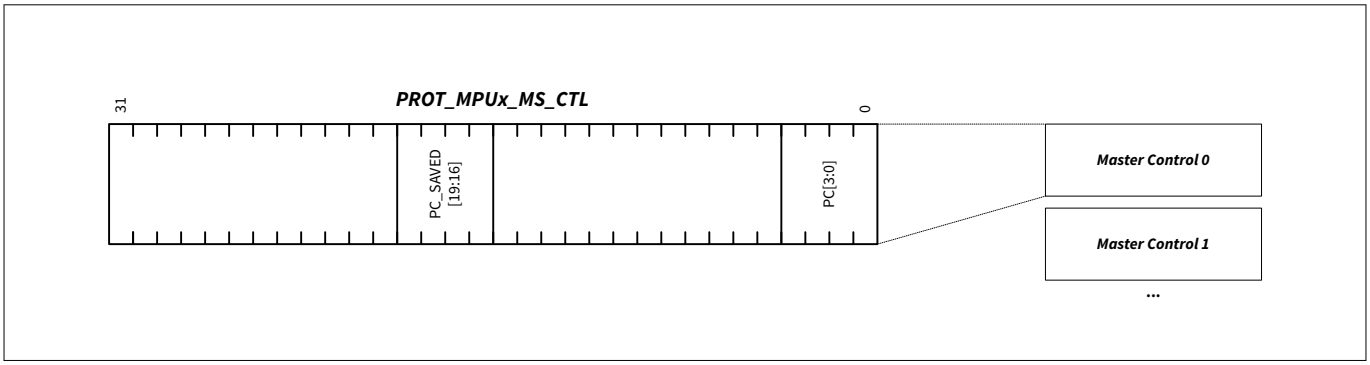

図 2 PROT_MPUx_MS_CTL レジスタ PROT_SMPU_MSx_CTL レジスタ

このレジスタは、マスタアクセスに保護コンテキスト属性を付与します。

- PROT_MPUx_MS_CTL.PC: 自身のアクセスに付与する保護コンテキスト属性を設定します。
- PROT_MPUx_MS_CTL.PC_SAVED: ブートプロセスによって設定されます。このフィールドは、CM0+ マスタにのみ存在します。

図 3 に、PROT_SMPU_MSx_CTL のレジスタ構造を示します。

3 動作概要

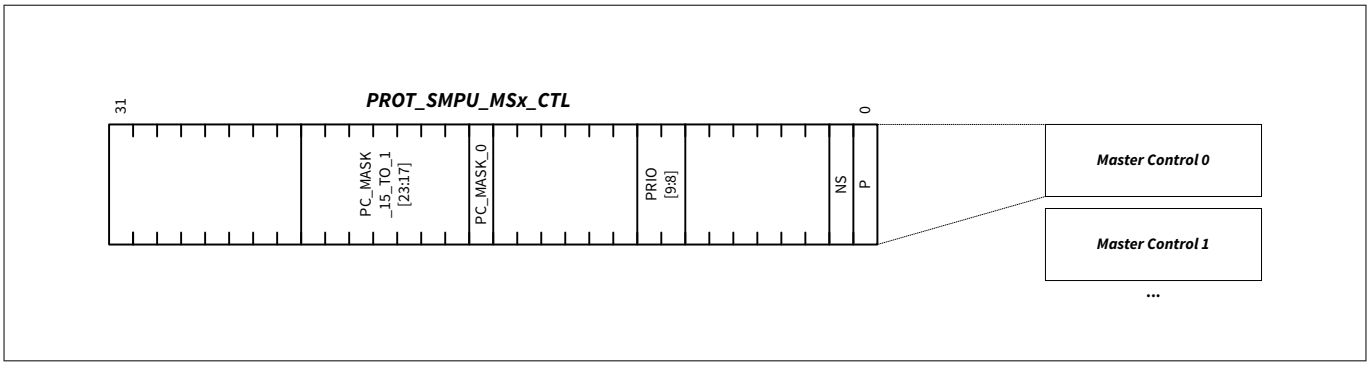

図 3　　PROT_SMPU_MSx_CTL レジスタ

このレジスタは、バスマスタアクセスに以下の属性を付与します。

- PROT_SMPU_MSx_CTL.P: 独自にユーザ/特権属性を持たないバスマスタにユーザ/特権属性を付与します。
- PROT_SMPU_MSx_CTL.NS: 独自にセキュア/非セキュア属性を持たないバスマスタにセキュア/非セキュア属性を付与します。
- PROT_SMPU_MSx_CTL.PC_MASK_15_TO_1 および PC_MASK_0: バスマスタが MPUx_MS_CTL.PC に設定可能な保護コンテキストを制限します。

***注**:　　CPUSS_CM0_PC_CTL.VALID[3:1] の 1 つのビットが 1 (関連する保護コンテキストハンドラが有効) の場合、PROT_SMPU_MSx_CTL.PC_MASK_15_TO_1[19:17] の関連ビットへのアプリケーションソフトウェアの書込み転送は常に 0 を書き込みます。これにより、有効な保護コンテキストハンドラを使用して保護コンテキスト 1, 2, または 3 に入ると、アプリケーションソフトウェアはそれらの保護コンテキストを設定できません。また、CPUSS_CMx_PC_CTL.VALID[3:1] ビットが 1 の場合、アプリケーションソフトウェアは PROT_SMPU_MSx_CTL.PC_MASK_15_TO_1[19:17] の関連ビットをクリアする必要があります。詳細については registers reference manual を参照してください。*

- PC_MASK_0 は、常に 0 です。すなわち、バスマスタは PC = 0 を設定できません。
- PROT_SMPU_MSx_CTL.PRIO: バス調停時の優先順位を設定します。

しかし、すべてのバスマスタがこれらのレジスタフィールドを持つわけではありません。表 1 に、各マスタのもつレジスタフィールドを示します。

表 1　　各バスマスタに提供されるレジスタフィールド

レジスタフィールド	CM0+ CPU	CRYPTO コンポーネント	P-DMA 0	P-DMA 1	M-DMA	CM4F CPU	テストコントローラ
PROT_MPUx_MS_CTL.PC	有	–	–	–	–	有	有
PROT_MPUx_MS_CTL.PC_SAVED	有	–	–	–	–	–	–
PROT_SMPU_MSx_CTL.P	-	-	-	-	-	-	有
PROT_SMPU_MSx_CTL.NS	有	–	–	–	–	有	有
PROT_SMPU_MSx_CTL.PC_MASK_15_TO_1 および PC_MASK_0	有	–	–	–	–	有	有
PROT_SMPU_MSx_CTL.PRIO	有	有	有	有	有	有	有

P-DMA0/1, M-DMA, および CRYPTO コンポーネントは MPU を持ちません。これらの周辺機能には、属性を設定するフィールドがありません。

各バスマスタには、関連する SMPU MS_CTL レジスタがあります。ただし、セキュアシステムにおいて、各マスタが独自の特権, セキュリティ, 調停優先順位および保護コンテキスト設定をしないように、このレジスタは通常、セキュアマスタ (CM0+) によってのみ制御できます。

3.2 属性の継承

P-DMA, M-DMA, および CRYPTO コンポーネントはチャネルまたはコンポーネントをプログラムしたバス転送のアクセス制御属性を継承します。継承されたアクセス属性は SMPU と PPU によって許可/制限されます。

図 4 に、属性の継承と、その動作例を示します。

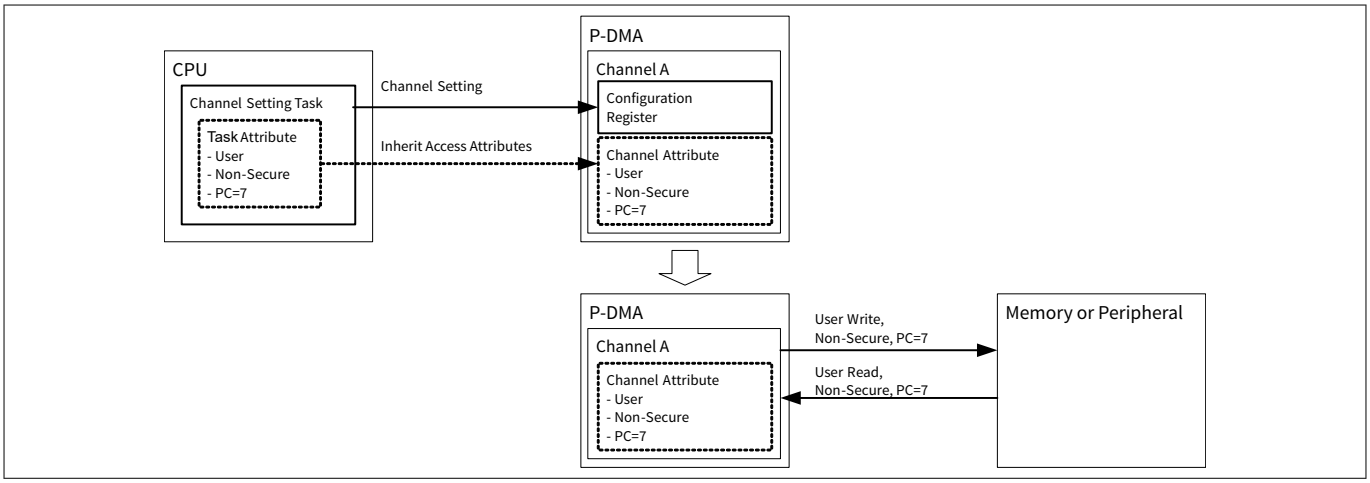

図 4 属性継承の設定と動作例

3.3 ユーザ/特権属性の切換え

ここでは、ユーザ/特権属性をサポートする両 CPU での属性切換えについて説明します。CPU は、以下の 2 つの動作モードと 2 つの特権レベルをサポートします。

- 動作モード
  - スレッドモード: このモードは、アプリケーションソフトウェアを実行するために使用されます。このモードは、特権レベルまたはユーザレベルで実行可能です。
  - ハンドラモード: このモードは、例外を処理するために使用されます。このモードは、特権レベルでのみ実行可能です。
- 特権レベル
  - ユーザレベル: ソフトウェアはアクセスが制限されます。
  - 特権レベル: ソフトウェアは、すべての命令を実行でき、すべての CPU リソースにアクセスが可能です。

特権レベルは、CONTROL レジスタによって切り換えます。これは、CPU 固有のレジスタです。特権レベルからユーザレベルへの切換えは、CONTROL レジスタを使用します。ただし、CONTROL レジスタは特権レベルでのみ書込み可能なレジスタです。したがって、ユーザレベルから特権レベルに切り換える場合、常にハンドラモードを経由する必要があります。例外または割込みが発生すると CPU はハンドラモードに遷移します。図 5 に、SVC (スーパバイザコール) 命令例外によるユーザ/特権レベルの切換え例を示します。SVC 命令は、例外を生成し、ハンドラモードへ遷移できます。

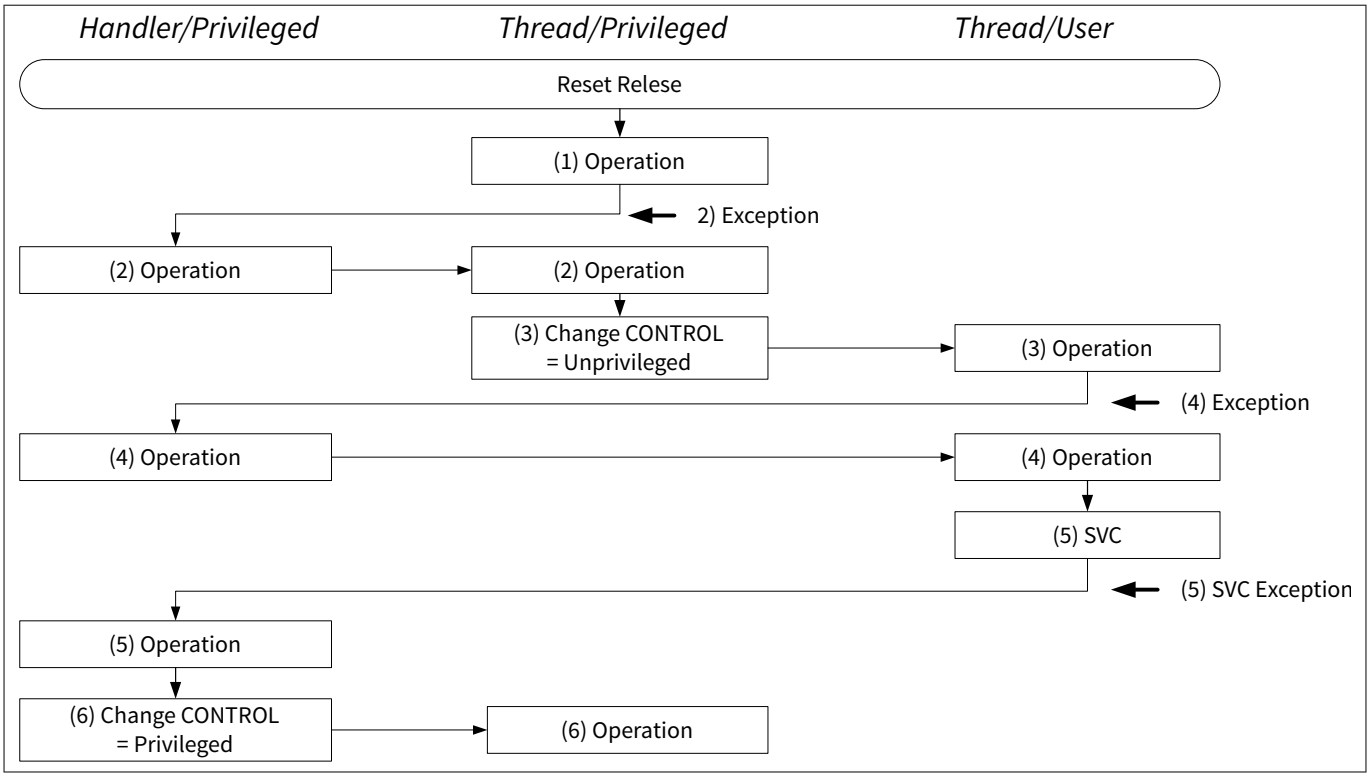

図 5　　CPU のユーザ/特権レベルの切換え例

1. リセット解除後、CPU はスレッドモード/特権レベルで起動されます。
2. スレッドモード/特権レベル中に例外が発生すると、ハンドラモード/特権レベルに遷移しハンドラ処理から復帰すると、再度スレッドモード/特権レベルに遷移します。
3. スレッドモード/特権レベルでは、CONTROL レジスタによってスレッドモード/ユーザレベルへ遷移します。
4. スレッドモード/ユーザレベル中に例外が発生すると、ハンドラモード/特権レベルに遷移しハンドラ処理から復帰すると再度、スレッドモード/ユーザレベルに遷移します。
5. スレッドモード/ユーザレベルからスレッドモード/特権レベルへの切換え時は、SVC 命令を使用しハンドラモード/特権レベルに遷移します。SVC 命令は、SVC 例外を発生します。
6. ハンドラモード/特権レベルに遷移後、CONTROL レジスタによって特権レベルを設定します。CPU は、ハンドラ処理から復帰後、スレッドモード/特権レベルに遷移します。

詳細については、Arm® ドキュメンテーションセット (CM4, CM7, および CM0+) を参照してください。

事前に SVC ハンドラを登録する必要があります。

3.3.1　　ユーザ/特権属性の切換え手順

ここでは、特権モードとユーザモードを切り換える方法について説明します。

3.3.2　　条件

表 2 に、SVC 命令を使用したユーザ/特権属性の切換えのための SDL 関数を示します。

表 2　　関数一覧

機能	説明	備考
`SetUserMode()`	特権レベルをユーザに変更します。	-

(続く)

表 2 (続き) 関数一覧

機能	説明	備考
SetPrivilegedMode()	特権レベルを特権に変更します。	-
SVC_SetPrivilegedMode()	SVC 割込みを生成し、特権レベルを特権に変更します。	-
Cy_SysLib_SvcHandler(pSvcArgs)	SVC ハンドラ pSvcArgs: SVC インデックス	インデックスが"2"の場合、特権に変更します。

以下のコードに、SVC を使用した切換え例を示します。

Code Listing 1: SVC を使用したユーザ/特権の切換え例

```
int main(void)
{
    SystemInit();      /* CPUs are started in Thread/Privileged mode after reset release. */

    __enable_irq();    /* CPUs are started in Thread/Privileged mode after reset release. */


    :
    /* The CPU works in user mode from here */
    SetUserMode();   /* Change Privileged Mode to User Mode. See Code Listing 2. */


    :
    /* The CPU works in privileged mode from here */
    SVC_SetPrivilegedMode();   /* Change from User Mode to Privileged Mode. See Code Listing 3.
*/

    :

    /* The CPU works in user mode again from here */
    SetUserMode();   /* Change from Privileged Mode to User Mode. See Code Listing 2. */

    :

    for(;;);
}
```

Code Listing 2 SetUserMode() 関数

```
void SetUserMode(void)
{
    __ASM("MRS r0, CONTROL"); // Read CONTROL register into R0   /* Read CONTROL Register */
    __ASM("ORR r0, r0, #1");  // nPRIV -> 1   /* (3) Change to User Mode */
    __ASM("MSR CONTROL, r0"); // Write R0 into CONTROL register   /* Write back to CONTROL
Register */
}
```

Code Listing 3 SVC_SetPrivilegedMode() 関数

```
void SVC_SetPrivilegedMode(void)
{
    /* Set Index to “2” */
    /* (5) SVC Exception with Index 2. It calls the SVC handler. See Code Listing 4. */
    __ASM("SVC 0x02"); // SVC index = 2: Get privileged mode
}
```

Code Listing 4 SVC ハンドラ

```
void Cy_SysLib_SvcHandler(uint32_t* pSvcArgs)
{
    uint8_t svcIdx = ((char*)pSvcArgs[6])[-2];

    switch(svcIdx)
    {
    case 0:         /* SVC Processing for Index 0. */
        :
        break;
    case 1:         /* SVC Processing for Index 1. */
        :
        break;
    case 2:
        SetPrivilegedMode();        /* SVC Processing for Index 2. Change to Privileged Mode.
See Code Listing 5. */
        break;
    default:
        break;
    }
}
```

Code Listing 5 SetPrivilegedMode() 関数

```
void SetPrivilegedMode(void)
{
    __ASM("MRS r0, CONTROL"); // Read CONTROL register into R0    /* Read CONTROL Register */
    __ASM("BIC r0, r0, #1");  // nPRIV -> 0    /* (6) Change to Privileged Mode */
    __ASM("MSR CONTROL, r0"); // Write R0 into CONTROL register    /* Write back to CONTROL
Register */
}
```

3.4 保護コンテキスト属性の設定

保護コンテキスト (PC) は、セキュリティおよび安全を目的とし、ソフトウェア実行の分離に使用されます。PC は、マスタによって開始されるすべてのバス転送の PC 属性として使用されます。SMPU と PPU は、PC 属性をもとにバス転送を許可または制限します。

3 動作概要

本シリーズは、8 個の PC をサポートします。8 つの PC のうち、PC0 および PC1 は特殊な PC で、これらの PC はハードウェアによって制御されます。さらに、PC0 は無制限のアクセス権を持ちます。

特定のバスマスタには、PC フィールド (PROT_MPUx_MS_CTL.PC および PROT_SMPU_MSx_CTL.PC_MASK_15_TO_1 および PC_MASK_0) が関連付けられます。

バスマスタの PC は、関連する PROT_MPUx_MS_CTL.PC フィールドの再プログラミングで変更できます。PROT_SMPU_MSx_CTL.PC_MASK_15_TO_1 および PC_MASK_0 フィールドは、関連するバスマスタが設定可能な PC を制限します。

例えば、PROT_SMPU_MSx_CTL.PC_MASK_15_TO_1 と PC_MASK_0 =“0x06” (PC1, 2="1")を設定した場合、設定可能な PC は“PC=1”と“PC=2”です。バスマスタは許可されない PC (PC = 0, 3, 4, 5, 6, および 7) へ変更できません。

図 6 に、PC 変更フローの例を示します。

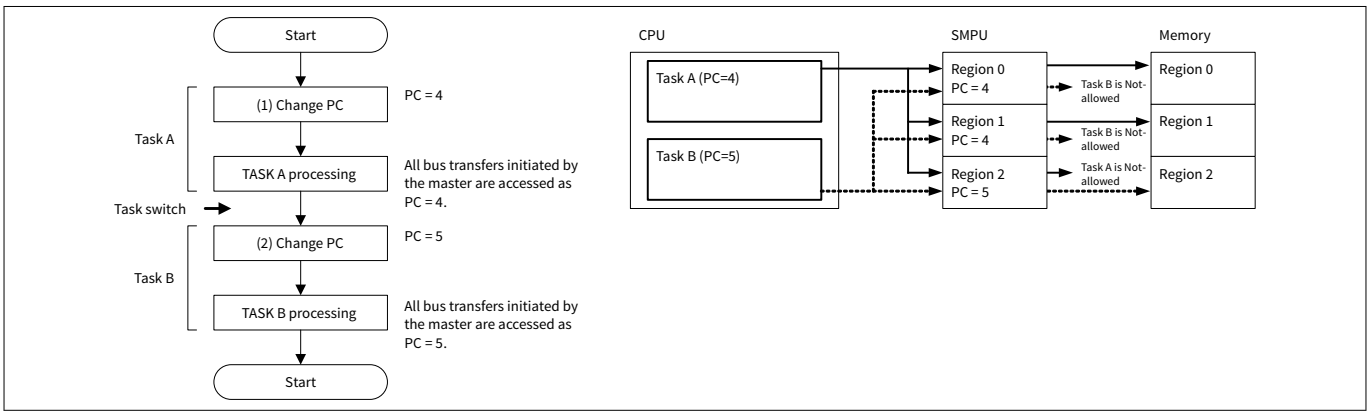

図 6　　PC の変更フローと動作

注:　各マスタが設定可能な PC 値は PROT_SMPU_MSx_CTL.PC_MASK_15_TO_1 および PC_MASK_0 によって制限されます。

注:　CM7 キャッシュメモリを有効にした時、PC やセキュアなどのアクセス属性が変更された場合、新しい PC や非セキュアなどのアクセスが許可されないデータがキャッシュメモリに保存されることがあります。この場合、保護コンテキストとセキュアの属性を切り換える前に、キャッシュメモリをクリーンし、無効化する必要があります。キャッシュメモリの処理については、AN224432 - TRAVEO™ T2G のマルチコアハンドリングガイド [5] を参照してください。

これによって、SMPU や PPU の設定を変更せずに、PC フィールドの再プログラミングのみで、単一のバスマスタが異なる保護役割を持てます。

3.4.1　保護コンテキスト属性の切換え手順

ここでは、図 6 に示す保護コンテキストを切り換える方法について説明します。

- 領域 0 および 1: PC = 4 アクセスは許可されます。
- 領域 2: PC = 5 アクセスは許可されます。

3.4.2　設定

表 3 および表 4 に保護コンテキスト切換えの SDL パラメータと関数を示します。

表 3　　パラメーター覧

パラメータ	説明	値
RESERVED_MEMORY_BLOCK_SIZE	各領域のメモリサイズを定義	0x400
PROTECTION_CONTEXT_OF_TASK_A	TASK A の保護コンテキスト番号を定義	4u
PROTECTION_CONTEXT_OF_TASK_B	TASK B の保護コンテキスト番号を定義	5u
PC_MASK_OF_TASK_A	PC = 4 を有効にするために PROT_SMPU_MSx_CTL.PC_MASK の値を定義	-
PC_MASK_OF_TASK_B	PC = 5 を有効にするために PROT_SMPU_MSx_CTL.PC_MASK の値を定義	-
gReservedRam.taskA_Region0/1/2	領域 0/1/2 の開始アドレスとメモリサイズを設定	メモリサイズ = RESERVED_MEMORY_BLOCK_SIZE
gSmpuStructConfigOfTask(A/B).address	SMPU 領域設定 (ベースアドレス)	gReservedRam.taskA_Region0/1/2
gSmpuStructConfigOfTask(A/B).regionSize	SMPU 領域設定 (領域サイズ)	CY_PROT_SIZE_1KB (1 KB)
gSmpuStructConfigOfTask(A/B).subregions	SMPU 領域設定 (サブリージョン設定)	0x00u (未使用)
gSmpuStructConfigOfTask(A/B).userPermission	SMPU 領域設定 (ユーザアクセス属性設定)	CY_PROT_PERM_RWX (=0x07u) ユーザはフルアクセス
gSmpuStructConfigOfTask(A/B).privPermission	SMPU 領域設定 (特権アクセス属性設定)	CY_PROT_PERM_RWX (= 0x07u) 特権はフルアクセス
gSmpuStructConfigOfTask(A/B).secure	SMPU 領域設定 (非セキュアアクセス属性設定)	False (非セキュア)
gSmpuStructConfigOfTask(A/B).pcMatch	SMPU 領域設定 (PC マッチ設定)	False (PC フィールドは"matching"に参加)
gSmpuStructConfigOfTask(A/B).pcMask	SMPU 領域設定 (PC_MASK 設定)	領域 0/1: PC_MASK_OF_TASK_A Region2: PC_MASK_OF_TASK_B
PROT_SMPU_SMPU_STRUCT0/1/2	PROT_SMPU_SMPU_STRUCT0/1/2 のベースアドレスを定義。 製品によって異なります。registers reference manual を参照してください。	–
CPUSS_MS_ID_CM4	バスマスタ ID を定義。 製品によって異なります。マスタ識別子を参照してください。	14

表 4 関数一覧

関数	説明	値
Cy_Prot_ConfigBusMaster (busMaster, privileged, secure, pcmask)	PROT_PROT_SMPU_MSx_CTL 設定 busMaster: マスタ識別子 privileged; P フィールド設定 セキュア: NS フィールド設定 pcmask: PC_MASK 設定 registers reference manual を参照してください。	busMaster: CPUSS_MS_ID_CM4 privileged; true (ユーザモード) secure: false (非セキュア) pcmask: PC_MASK 設定
Cy_Prot_ConfigSmpuSlaveStruct (*base, *config)	SMPU 領域設定 *base: レジスタベースアドレス *config: パラメータ	*base: PROT_SMPU_SMPU_STRUCT0/1/2 *config:gSmpuStructConfigOfTask(A/B)
Cy_Prot_EnableSmpuSlaveStruct(*base)	SMPU 領域有効化 *base: レジスタベースアドレス	*base: PROT_SMPU_SMPU_STRUCT0/1/2
Cy_Prot_SetActivePC (busMaster, PC)	PROT_MPU_MSx_CTL 設定 busMaster: マスタ識別子 PC: PC 値	busMaster: CPUSS_MS_ID_CM4 PC: PROTECTION_CONTEXT_OF_TASK_A または、 PROTECTION_CONTEXT_OF_TASK_B

以下は、SDL のドライバ部分のレジスタ表記を説明します。

- addrMpu->un**MS_CTL**.u32Register は registers reference manual に記載される PROT_MPUx_MS_CTL レジスタです。その他のレジスタについても、同様の意味です。“**x**” はバスマスタ識別子を示します。
- パフォーマンス改善策
- レジスタ設定のパフォーマンス向上のため、SDL は完全な 32 ビットデータをレジスタに書き込みます。各ビットフィールドはビット書込み可能なバッファ内に事前に生成され、最終的な 32 ビットデータとしてレジスタに書き込まれます。
```
tempSL_ATT0.u32Register = base->unSL_ATT0.u32Register;
tempSL_ATT0.stcField.u1PC1_UR = (config->userPermission & CY_PROT_PERM_R);
tempSL_ATT0.stcField.u1PC1_UW = (config->userPermission & CY_PROT_PERM_W) >> 1;
tempSL_ATT0.stcField.u1PC1_PR = (config->privPermission & CY_PROT_PERM_R);
tempSL_ATT0.stcField.u1PC1_PW = (config->privPermission & CY_PROT_PERM_W) >> 1;
tempSL_ATT0.stcField.u1PC1_NS = !(config->secure); base->unSL_ATT0.u32Register =
tempSL_ATT0.u32Register;
```

レジスタの共用体と構造体については `hdr/rev_x/ip` の `cyip_prot_v2.h` および `cyip_peri_ms_v2.h` を参照してください。

Code Listing 6 に保護コンテキストの切換え例を示します。

3 動作概要

Code Listing 6 保護コンテキスト切換え例

```
#define RESERVED_MEMORY_BLOCK_SIZE (0x400) // 1K   /*Define each region size */

#define PROTECTION_CONTEXT_OF_TASK_A (4u)   /* Define Protection context for Task A (PC=4) */
#define PROTECTION_CONTEXT_OF_TASK_B (5u)   /* Define Protection context for Task B (PC=5) */

#define PC_MASK_OF_TASK_A (1u<<(PROTECTION_CONTEXT_OF_TASK_A-1u))   /* Define PC_Mask for each
SMPU region. */
#define PC_MASK_OF_TASK_B (1u<<(PROTECTION_CONTEXT_OF_TASK_B-1u))   /* Define PC_Mask for each
SMPU region. */

struct
   /* Define SRAM region. */
{
    uint8_t taskA_Region0[RESERVED_MEMORY_BLOCK_SIZE];
    uint8_t taskA_Region1[RESERVED_MEMORY_BLOCK_SIZE];
    uint8_t taskB_Region2[RESERVED_MEMORY_BLOCK_SIZE];
} gReservedRam;

cy_stc_smpu_cfg_t gSmpuStructConfigOfTaskA =
   /* Configure SMPU for region 0 and 1. (PC=4 access has permissions) */
{
    .address        = NULL,              // Will be updated in run time
    .regionSize     = CY_PROT_SIZE_1KB,
    .subregions     = 0x00u,
    .userPermission = CY_PROT_PERM_RWX,
    .privPermission = CY_PROT_PERM_RWX,
    .secure         = false,             // Non secure
    .pcMatch        = false,
    .pcMask         = PC_MASK_OF_TASK_A, // only enable for task A
};




cy_stc_smpu_cfg_t gSmpuStructConfigOfTaskB =
   /* Configure SMPU for region 2. (PC=5 access has permissions) */
{
    .address        = NULL,              // Will be updated in run time
    .regionSize     = CY_PROT_SIZE_1KB,
    .subregions     = 0x00u,
    .userPermission = CY_PROT_PERM_RWX,
    .privPermission = CY_PROT_PERM_RWX,
    .secure         = false,             // Non secure
    .pcMatch        = false,
    .pcMask         = PC_MASK_OF_TASK_B, // only enable for task B      /*Enabled PC=5 by
PC_MASK */
};
```

```
int main(void)
{
    SystemInit();

    cy_en_prot_status_t status;

   /*Set PROT_SMPU_MS14_CTL.PC_MASK. See Configuration Example of SMPU for SMPU setting details.
(*) */
    /* Setting for MS14_CTL (for CM4) to allow the PC value to become 4 or 5 */
    status = Cy_Prot_ConfigBusMaster(CPUSS_MS_ID_CM4, true, false, (PC_MASK_OF_TASK_A|
PC_MASK_OF_TASK_B));
    CY_ASSERT(status == CY_PROT_SUCCESS);

    /* Setting for SMPU_STRUCT 0 */
    /* Setting SMPU_STRUCT 0 for task A */
   /*Enable SMPU region 0.  For details on setting SMPU, see Configuration Example of SMPU. */
    gSmpuStructConfigOfTaskA.address = (uint32_t*)gReservedRam.taskA_Region0;
   /*Enable SMPU region 0.  For details on setting SMPU, see Configuration Example of SMPU. */
    status = Cy_Prot_ConfigSmpuSlaveStruct(PROT_SMPU_SMPU_STRUCT0, &gSmpuStructConfigOfTaskA);
   /*Enable SMPU region 0.  For details on setting SMPU, see Configuration Example of SMPU. */
    CY_ASSERT(status == CY_PROT_SUCCESS);

    /* Enable SMPU_STRUCT 0 */
   / *Enable SMPU region 0.  For details on setting SMPU, see Configuration Example of SMPU. */
    status = Cy_Prot_EnableSmpuSlaveStruct(PROT_SMPU_SMPU_STRUCT0);

    CY_ASSERT(status == CY_PROT_SUCCESS);

    /* Setting for SMPU_STRUCT 1 */
    /* Setting SMPU_STRUCT 1 for task A */
   /*Enable SMPU region 1.  For details on setting SMPU, see Configuration Example of SMPU. */
    gSmpuStructConfigOfTaskA.address = (uint32_t*)gReservedRam.taskA_Region1;
   /*Enable SMPU region 1.  For details on setting SMPU, see Configuration Example of SMPU. */
    status = Cy_Prot_ConfigSmpuSlaveStruct(PROT_SMPU_SMPU_STRUCT1, &gSmpuStructConfigOfTaskA);
   /*Enable SMPU region 1.  For details on setting SMPU, see Configuration Example of SMPU. */
    CY_ASSERT(status == CY_PROT_SUCCESS);

    /* Enable SMPU_STRUCT 1 */
    /*Enable SMPU region 1.  For details on setting SMPU, see Configuration Example of SMPU. */
    status = Cy_Prot_EnableSmpuSlaveStruct(PROT_SMPU_SMPU_STRUCT1);
    CY_ASSERT(status == CY_PROT_SUCCESS);

    /* Setting for SMPU_STRUCT 2 */
    /* Setting SMPU_STRUCT 2 for task B */
   /*Set SMPU region 2. For details on setting SMPU, see Configuration Example of SMPU. */
    gSmpuStructConfigOfTaskB.address = (uint32_t*)gReservedRam.taskB_Region2;
   /*Set SMPU region 2. For details on setting SMPU, see Configuration Example of SMPU. */
    status = Cy_Prot_ConfigSmpuSlaveStruct(PROT_SMPU_SMPU_STRUCT2, &gSmpuStructConfigOfTaskB);
   /*Set SMPU region 2. For details on setting SMPU, see Configuration Example of SMPU. */
    CY_ASSERT(status == CY_PROT_SUCCESS);

    /* Enable SMPU_STRUCT 2 */
    /*Enable SMPU region 2. For details on setting SMPU, see Configuration Example of SMPU. */
```

```
    status = Cy_Prot_EnableSmpuSlaveStruct(PROT_SMPU_SMPU_STRUCT2);
    CY_ASSERT(status == CY_PROT_SUCCESS);
   for(;;)
    {
        /* Setting for MPU so that CM4 PC for task A   */
      /*(1) Change protection context to PC=4 for TASK A. See Code Listing 7. */
        status = Cy_Prot_SetActivePC(CPUSS_MS_ID_CM4, PROTECTION_CONTEXT_OF_TASK_A);
        CY_ASSERT(status == CY_PROT_SUCCESS);

        /* Do task A */
       /*Access to RAM region 0 and 1. See Code Listing 8 */
        Routine_TaskA();

        /* Setting for MPU so that CM4 PC for task B   */
    /*(2) Change protection context to PC=5 for TASK B. See Code Listing 7. */
        status = Cy_Prot_SetActivePC(CPUSS_MS_ID_CM4, PROTECTION_CONTEXT_OF_TASK_B);
        CY_ASSERT(status == CY_PROT_SUCCESS);

        /* Do task B */
       /*Access to RAM region 2. See Code Listing 8. */
        Routine_TaskB();
    }
}
```

注: (*) このプロセスは、対応するマスタが設定可能な保護コンテキスト値を指定します。セキュアシステムでは、セキュアマスタによって実行されます。詳細については、バス転送の保護プロパティを参照してください。

Code Listing 7 Cy_Prot_SetActivePC() 関数

```
cy_en_prot_status_t Cy_Prot_SetActivePC(en_prot_master_t busMaster, uint32_t pc)
{
    cy_en_prot_status_t status = CY_PROT_SUCCESS;
    un_PROT_MPU_MS_CTL_t tProtMpuMsCtl = {0ul};
    volatile stc_PROT_MPU_t* addrMpu = (stc_PROT_MPU_t*)(&PROT->CYMPU[busMaster]);

    if(pc > (uint32_t)CY_PROT_MS_PC_NR_MAX)   /* Define Protection context for Task A (PC=4) */
    {
        /* Invalid PC value - not supported in device */
        status = CY_PROT_BAD_PARAM;
    }
    else
    {
        tProtMpuMsCtl.stcField.u4PC = pc;    /* Change protection context. */
        addrMpu->unMS_CTL.u32Register = tProtMpuMsCtl.u32Register;
        status = ((addrMpu->unMS_CTL.stcField.u4PC != pc) ? CY_PROT_FAILURE : CY_PROT_SUCCESS);
    }

    return status;
}
```

Code Listing 8 Routine_TaskA() 関数と Routine_TaskB() 関数

```
void Routine_TaskA(void)
    /* Access to region 0 and 1 with TASK A. If these regions are accessed with TASK B, it will
cause bus fault. */
{
    for(uint32_t i = 0; i < RESERVED_MEMORY_BLOCK_SIZE; i++)
    {
        gReservedRam.taskA_Region0[i] += 1;
    }

    for(uint32_t i = 0; i < RESERVED_MEMORY_BLOCK_SIZE; i++)
    {
        gReservedRam.taskA_Region1[i] += 1;
    }
}

void Routine_TaskB(void)
    /* Access to region 2 with TASK B. If these regions are accessed with TASK A, it will cause
bus fault. */
{
    for(uint32_t i = 0; i < RESERVED_MEMORY_BLOCK_SIZE; i++)
    {
        gReservedRam.taskB_Region2[i] += 1;
    }
}
```

3.5 バス転送の評価

3.5.1 評価プロセス

保護ユニットによるバス転送の評価は、2 つの独立したプロセスに分かれます。

- マッチングプロセス: このプロセスは、各保護構造に対して転送アドレスがアドレス範囲内に含まれるかを評価します。
- アクセス評価プロセス: このプロセスは、各保護構造に対してバス転送属性がアクセス制御属性と一致するかを評価します。

以下の疑似コードは、バス転送の評価プロセスを示します。

```
/* Matching Process */
match = 0;
for (i = n-1; i >= 0; i--)   // n: number of protection regions
   if (Match (“transfer address”, “protection context”) {
      match = 1; break;
   }

/* Access Evaluation Process */
if (match)
   AccessEvaluate (“access attributes”, “protection context”);
else
   “access allowed”
```

***注**:　保護構造が一致しない場合、アクセスは許可されます。*

***注**:　複数の保護構造が一致する場合、アクセス評価のためのアクセス属性は、最高のインデックスをもつ保護構造によって評価されます。*

保護ユニットは保護構造を降順で評価します。言い換えると、インデックスの高い保護構造はインデックスの低い保護構造よりも優先されます。

転送アドレスが不一致の場合、次に高いインデックスを持つ保護構造が評価されます。転送アドレスが一致の場合、バス転送はアクセス評価プロセスによって評価されます。アクセス評価プロセスによってバス転送属性が不一致の場合、アクセス違反として検出されます。したがって、この場合、バス転送属性は次に高いインデックスをもつ保護構造によって評価されません。

3.5.2　PC_MATCH 動作

SMPU は PC_MATCH フィールドを持ちます。PC_MATCH は、"matching"と"access evaluation"プロセスを制御します。

- PC_MATCH = 0 の場合

以下の疑似コードは PC_MATCH = 0 の時の評価プロセスを示します。

```
match = 0;
for (i = n-1; i >= 0; i--)   // n: number of protection regions
   if (Match (“transfer address”) {
      match = 1; break;
   }

if (match)
   AccessEvaluate (“access attributes”, “protection context”);
else
   “access allowed”
```

PC_MATCH = “0” の場合、保護コンテキストはアクセス評価プロセスでのみ評価されます。

図 7 に、領域 5 の PC_MATCH = “0” および領域 4 の PC_MATCH = “0” 時の動作例を示します。表 5 に、各領域の設定を示します。

表 5　　PC_MATCH 動作の領域設定 1

領域	PC_MATCH	領域アドレス	保護コンテキスト	ユーザ
領域 4	0	AA	4	リード/ライト (R/W)
領域 5	0	AA	5	リードのみ

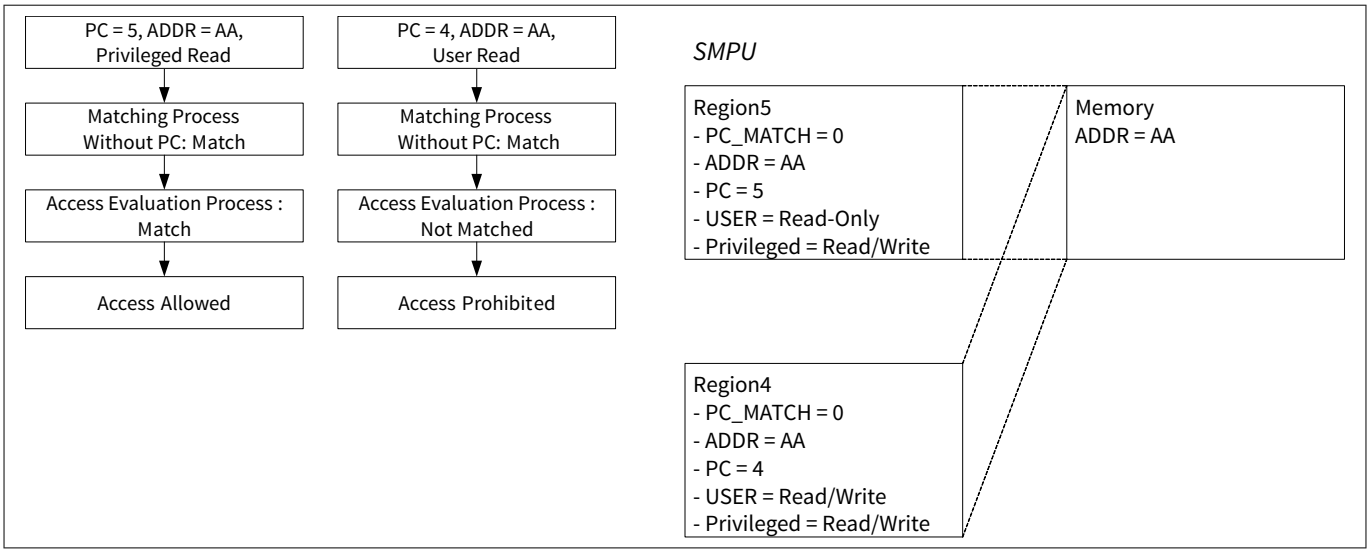

図 7　　PC_MATCH 動作例 1

この場合、保護コンテキストはマッチングプロセスによって評価されません。したがって、PC = 4, PC = 5 の両アクセスともマッチングプロセスで「一致」します。保護コンテキストは、アクセス評価プロセスによって評価されます。PC=5 の場合はアクセスが許可され、PC=4 の場合はアクセスが許可されません。結果として、PC = 4 のアクセスは、領域 4 よりも高いインデックスを持つ領域 5 によって禁止されているため PC = 4 のアクセスはこのアドレスにアクセスできません。

- PC_MATCH = 1 の場合

以下の疑似コードは PC_MATCH = 1 の時の評価プロセスを示します。

```
match = 0;
for (i = n-1; i >= 0; i--)   // n: number of protection regions
   if (Match (“transfer address”, “protection context”) {
      match = 1; break;
   }

if (match)
   AccessEvaluate (“access attributes”, “protection context”);
else
   “access allowed”
```

PC_MATCH = “1” の場合、保護コンテキストはアクセス評価プロセスだけでなく、マッチングプロセスによっても評価されます。

図 8 に、領域 5 の PC_MATCH = “1” および領域 4 の PC_MATCH = “0” 時の動作例を示します。また表 6 に、各領域の設定を示します。

表 6　　PC_MATCH 動作の領域設定 2

領域	PC_MATCH	領域アドレス	保護コンテキスト	ユーザ
領域 4	0	AA	4	リード/ライト (R/W)
領域 5	1	AA	5	リードのみ

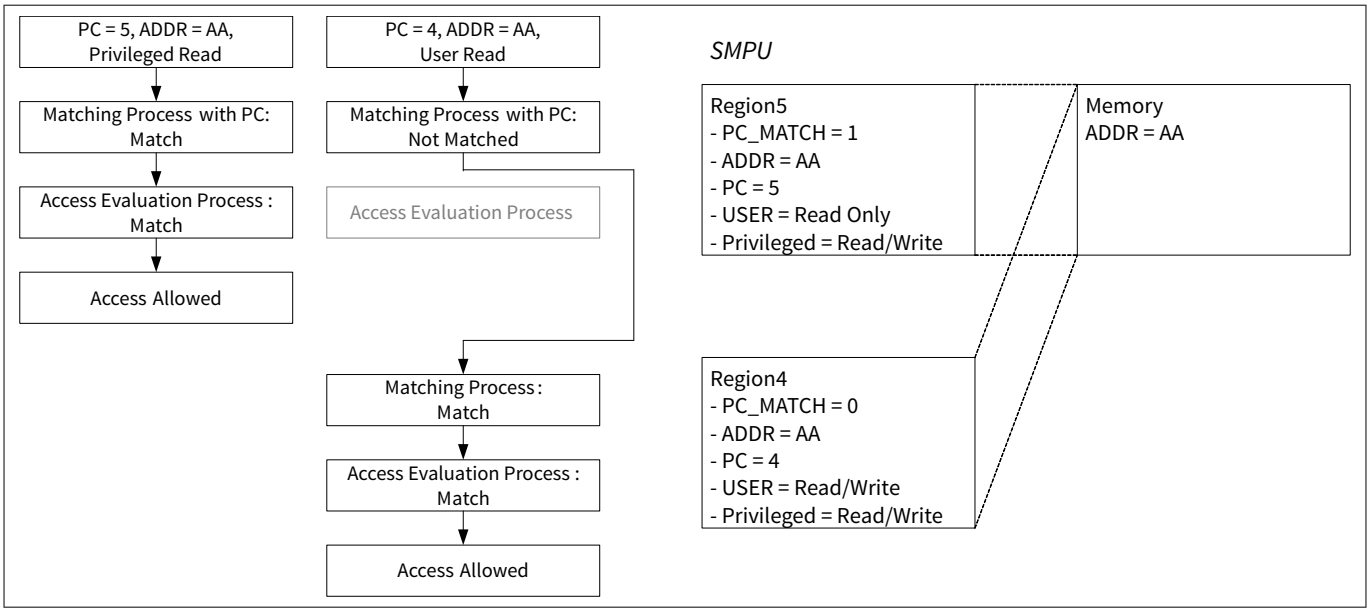

図 8　　PC_MATCH 動作例 2

この場合、保護コンテキストもマッチングプロセスによって評価されます。PC = 5 アクセスは、マッチングプロセスによって「一致」となり、アクセス評価プロセスによって評価されます。しかし、PC=4 のアクセスは "Not matched"であり、マッチングプロセスでは次に優先順位の高い Region4 と評価される。マッチング処理後のアクセス評価処理で評価され、アクセスが許可される。

PC_MATCH を使用することで、保護コンテキストに応じて、同じアドレスに異なる属性を割り当てられます。

注:　*PC_MATCH は SMPU のみ提供されます。PPU は、すべての保護コンテキストにアクセス属性を割り当てられるためこの機能はサポートされません。*

注:　*リージョンアドレスに属性が 1 つしかない場合、PC_MATCH を"0" で使用します。*

3.6　マスタ識別子

各バスマスタは固有のマスタ識別子を持ちます。この識別子は、保護ユニットのレジスタサフィックスや保護ユニットによるアクセス違反をしたマスタの識別に使用されます。表 7 に、マスタ識別子を示します。使用する製品のバスマスタ識別子はデータシートを参照してください。

表 7　　マスタ識別子

マスタ識別子	バスマスタ			
	CYT2B7 シリーズ	CYT4BF シリーズ	CYT6BJ シリーズ	CYT4DN シリーズ
0	CM0+ CPU	CM0+ CPU	CM0+ CPU	CM0+ CPU

(続く)

表 7 (続き) マスタ識別子

マスタ識別子	バスマスタ			
	CYT2B7 シリーズ	CYT4BF シリーズ	CYT6BJ シリーズ	CYT4DN シリーズ
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
2	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
4	M-DMA	M-DMA	M-DMA	M-DMA
5	-	SDHC	SDHC	-
7	-	-	CM7_2 CPU	-
8	-	-	CM7_3 CPU	-
9	-	Ethernet 0	Ethernet 0	Ethernet 0
10	-	Ethernet 1	Ethernet 1	JPEG Decoder
11	-	-	-	AXI DMA
12	-	-	-	Video Subsystem
13	-	CM7_1 CPU	CM7_1 CPU	CM7_1 CPU
14	CM4 CPU	CM7_0 CPU	CM7_0 CPU	CM7_0 CPU
15	Test Controller	Test Controller	Test Controller	Test Controller

3.7 保護違反

CPU の一部として実装される MPU がアクセス違反を検出した場合、プログラマブル優先順位 MemManage フォルトまたは HardFault ハンドラが呼び出されます。TCM 以外のアクセスで MPU フォルトが発生した場合、そのアクセスに対する AXI または AHB への転送は実行されません。MPU の詳細については、CM4, CM7, および CM0+ の Arm® ドキュメンテーションセットを参照してください。

バスインフラストラクチャの一部として実装される MPU と SMPU が保護違反を引き起こすバス転送を検出した場合、そのバス転送はバスエラーになります。

PPU 保護違反となる書込み転送を検出した場合、バッファリングが有効 (CPUSS_BUFF_CTL.WRITE_BUFF = 1) な場合、バスマスタはバスエラーを認識しません。これは、バスインフラストラクチャ内の AHB-Lite ブリッジが書込み転送をバッファリングし、OK 応答をマスタに返信するためです。この場合、システムは PPU の Fault report によって行われます。バッファリングが無効 (CPUSS_BUFF_CTL.WRITE_BUFF = 0) の場合、PPU に違反する書き込み転送はバス エラーを引き起こします。PPU 保護違反となる読出しアクセスは常にバスエラーになります。

保護ユニットに違反するバス転送は、対象メモリ配置またはペリフェラルレジスタには到達しません。

バスインフラストラクチャの一部として実装される MPU, SMPU, および PPU によって検出された保護違反は、Fault report structure に取込まれます。Fault report structure はエラーの発生を示す割込みを生成できます。加えて、違反したバス転送に関する以下の情報が Fault report structure に通知されます。

Fault report structure は、次の情報を取得します。

- 違反したアドレス
- 違反した属性
  - ユーザリード/ユーザライト/ユーザ実行
  - 特権リード/特権ライト/特権実行

  - 非セキュア
  - 保護コンテキスト
- 違反したマスタ識別子
- 違反を検出した保護ユニットまたはフォルトタイプ [1)]

[1] 検出されたフォルトによります。レジスタ・リファレンス・マニュアル参照

4 保護ユニット構造

4.1 MPU 構造

図 9 に、バスインフラストラクチャの一部として実装される MPU 構造について示します。CM4, CM7, および CM0+の一部として実装される MPU については、Arm® ドキュメンテーションセット (CM4, CM7, および CM0+) を参照してください。.

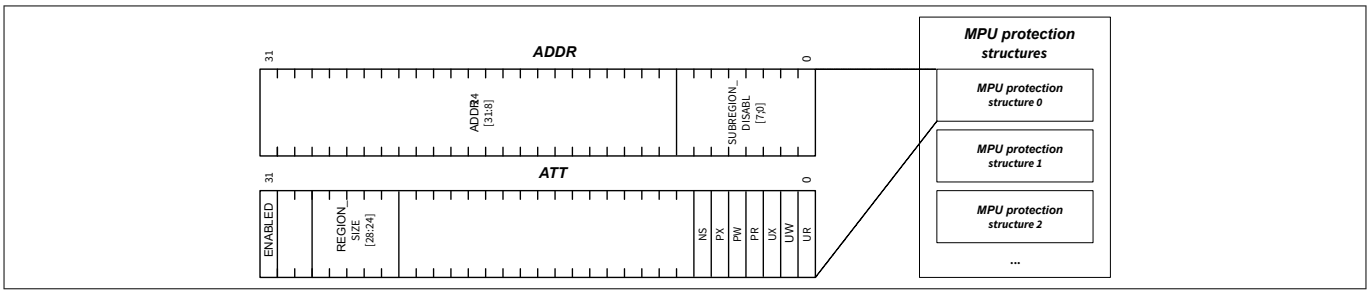

図 9 MPU 構造

MPU 保護は、マスタアクセスによって許可または制限されるプロパティを設定します。MPU は以下のプロパティを識別します。

- アドレス範囲
  - ADDR.ADDR24/31/8 領域のベースアドレスを指定します
  - ATT.REGION_SIZE [28:24]: 領域のサイズを指定します。領域サイズの範囲は [256 B, 4 GB] です。
  - ADDR.SUBREGION_DISABLE [7:0]: 領域内の 8 つサブリージョンの設定を個別に無効化します。
- アクセス属性
  - ATT.UR: ユーザリードアクセスの制御
  - ATT.UW: ユーザライトアクセスの制御
  - ATT.UX: ユーザ実行アクセスの制御
  - ATT.PR: 特権リードアクセスの制御
  - ATT.PW: 特権ライトアクセスの制御
  - ATT.PX: 特権実行アクセスの制御
  - ATT.NS: セキュアアクセスの制御
- 領域のイネーブル
  - ATT.ENABLED: 領域の有効/無効を制御

MPU は保護コンテキストを提供しません。この MPU タイプの定義は、一貫したソフトウェアインタフェースを保証するため、Arm® MPU (メモリ領域とアクセス属性の定義) に従います。

領域は 8 つの等しいサイズの領域に分割できます。ADDR.SUBREGION_DISABLE によって領域内のサブリージョンは個別に有効化できます。

例えば、分割されたサブ領域 [0: 7] の SUBREGEON_DESABLE が 0x82 (ビット 1 と 7 が"1") の場合、サブリージョン 1, 7 は無効、サブリージョン 0, 2, 3, 4, 5, および 6 は有効です。表 8 に、開始アドレスが 0x10005400 で、範囲が 0x10005400～0x100055ff (512 バイト) の場合の、8 つのサブリージョンの有効/無効の状態を示します。

表 8 各サブリージョン領域と状態

サブリージョン	範囲	状態
サブリージョン 0	0x10005400～0x1000543f	イネーブル
サブリージョン 1	0x10005440～0x1000547f	ディセーブル

(続く)

表 8 (続き) 各サブリージョン領域と状態

サブリージョン	範囲	状態
サブリージョン 2	0x10005480～0x100054bf	イネーブル
サブリージョン 3	0x100054c0～0x100054ff	イネーブル
サブリージョン 4	0x10005500～0x1000553f	イネーブル
サブリージョン 5	0x10005540～0x1000557f	イネーブル
サブリージョン 6	0x10005580～0x100055bf	イネーブル
サブリージョン 7	0x100055c0～0x100055ff	ディセーブル

4.2 SMPU 構造

図 10 に、SMPU 構造を示します。

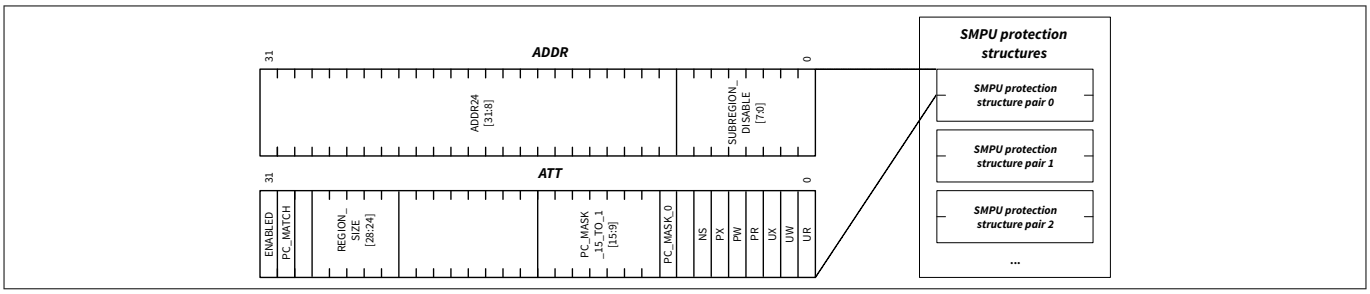

図 10 SMPU 構造

SMPU 保護構造は、マスタアクセスによって許可または制限されるプロパティを設定します。SMPU は次のプロパティを識別します。

- アドレス範囲
  - ADDR.ADDR24 [31:8]: 領域のベースアドレスを指定します。
  - ATT.REGION_SIZE [28:24]: 領域のサイズを指定します。領域サイズの範囲は [256 B, 4 GB] です。
  - ADDR.SUBREGION_DISABLE [7:0]: 領域内の 8 つサブリージョンを個別に無効化します。
- アクセス属性
  - ATT.UR: ユーザリードアクセスの制御
  - ATT.UW: ユーザライトアクセスの制御
  - ATT.UX: ユーザ実行アクセスの制御
  - ATT.PR: 特権リードアクセスの制御
  - ATT.PW: 特権ライトアクセスの制御
  - ATT.PX: 特権実行アクセスの制御
  - ATT.NS: セキュアアクセスの制御
  - ATT.PC_MASK_15_TO_1 および PC_MASK_0: 保護コンテキスト属性の制御
  - The PC_MASK_0 は、常に”1”です。言い換えると、PC=0 は常に許可されます。
  - ATT.PC_MATCH: PC フィールドが"matching"プロセスまたは"access evaluation"プロセスへの参加を制御します。詳細については、PC_MATCH 動作を参照してください。
- 領域のイネーブル
  - ATT.ENABLED: 領域の有効/無効を制御

SMPU の SUBREGION 機能は MPU と同じです。

4.3 PPU 構造

図 11 に、PPU 構造を示します。

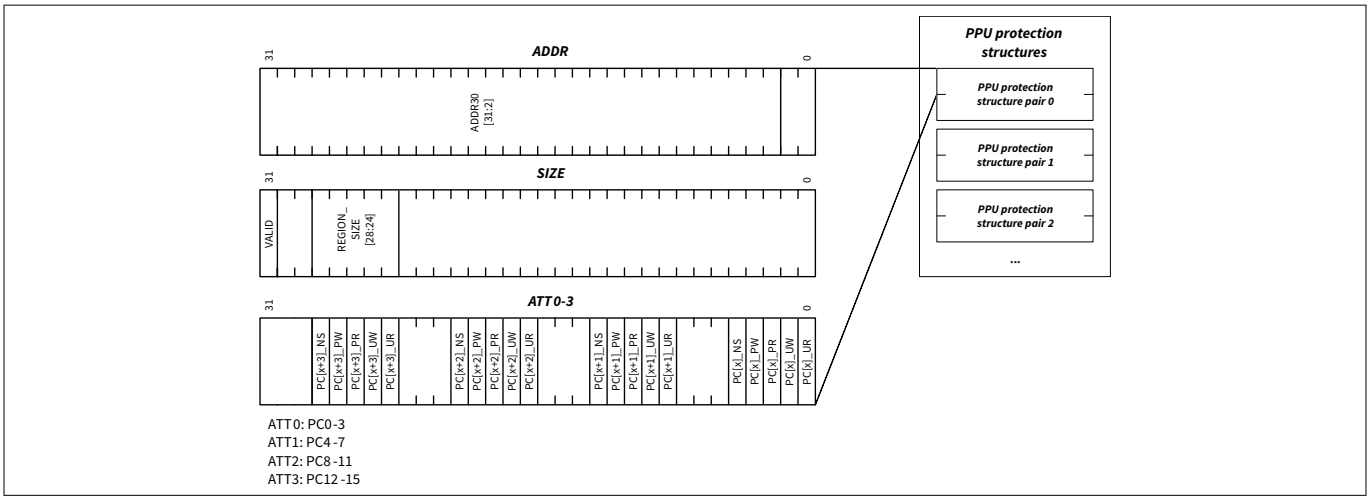

図 11 PPU 構造

PPU 保護構造は、マスタアクセスによって許可または制限されるプロパティを設定します。また、すべての保護コンテキストに対して独立して属性を設定できます。PPU は次のプロパティを識別します。

- アドレス範囲
  - ADDR.ADDR30 [31:2]: 領域のベースアドレスを指定します。
  - SIZE.REGION_SIZE [28:24]: 領域のサイズを指定します。領域サイズの範囲は [4 B, 2 GB] です。

Fixed PPU 構造は、固定された定数アドレス領域を持ちます。

- アクセス属性
  - ATT.PCx_UR: PCx のユーザリードアクセスの制御
  - ATT.PCx_UW: PCx のユーザライトアクセスの制御
  - ATT.PCx_PR: PCx の特権リードアクセスの制御
  - ATT.PCx_PW: PCx の特権ライトアクセスの制御
  - ATT.PCx_NS: PCx のセキュアアクセスの制御
- 領域のイネーブル
  - SIZE.VALID: 領域の有効/無効を制御

注: 本シリーズでは、保護コンテキストは 0~7 をサポートします。したがって、ATT0 と ATT1 レジスタのみ存在します。

4.4 保護ペア構造

保護ユニットのレジスタは、他の周辺機能と同じレジスタです。さらに、保護構造のレジスタは、周辺機能のレジスタと同様に他の保護構造のアドレス範囲に含められます。したがって保護構造は、保護構造によって保護できます。

保護構造を保護する保護構造をマスタ構造と呼び、マスタによって保護される保護構造をスレーブ構造と呼びます。スレーブは周辺機能を保護します。

スレーブとマスタの保護構造を保護ペアと呼びます。SMPU と PPU には、この保護ペアがあります。図 12 に、保護ペアの構造を示します。

4 保護ユニット構造

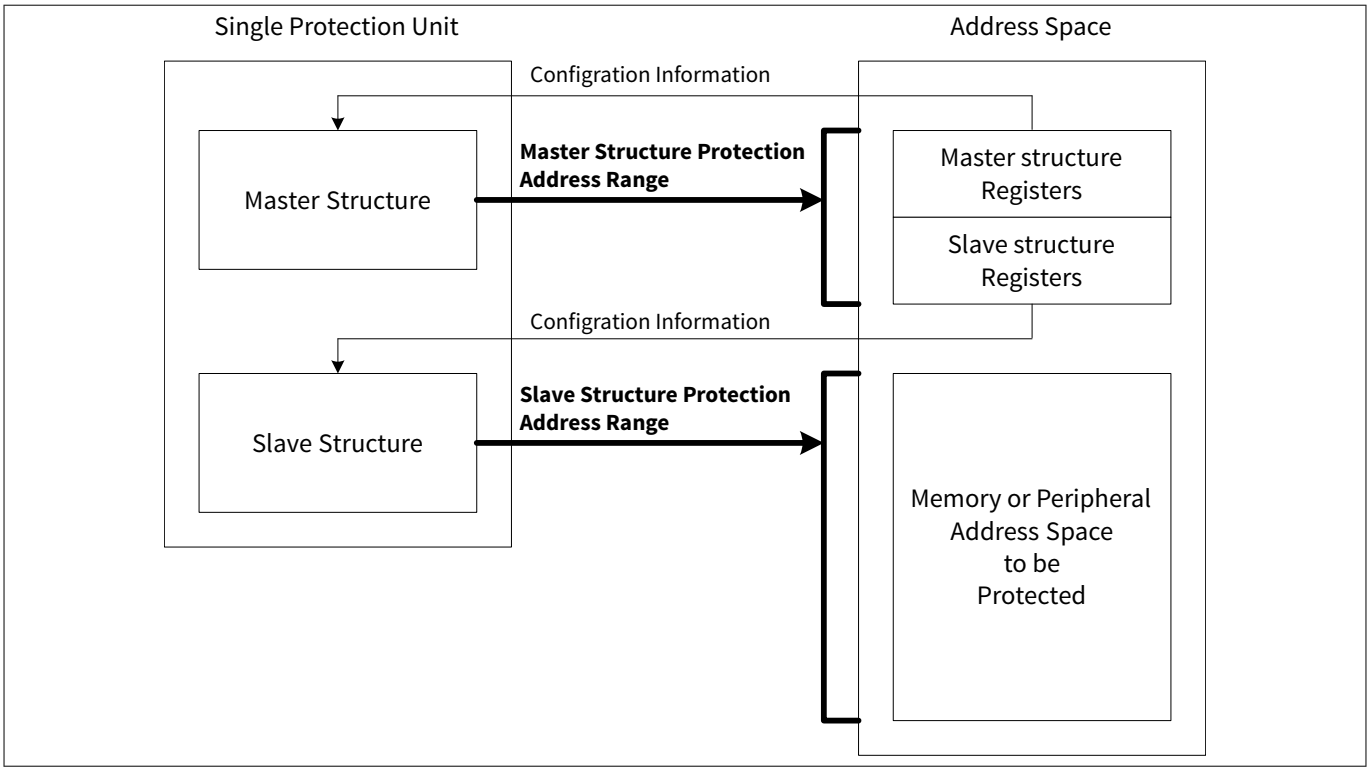

図 12　　保護ペア構造

スレーブを保護するマスタ構造には以下の特長があります。

- アドレス範囲
  - ADDR.ADDR: 読出し専用です。固定された定数のアドレス領域です。
  - ATT.REGION_SIZE: 読出し専用です。固定された定数のアドレス領域です。
  - ADDR.SUBREGION_DISABLE: 読出し専用です。固定された定数のアドレス領域です。
- アクセス属性
  - ATT.UR: “1”固定です。ユーザリードは常に許可されます。
  - ATT.UW: ユーザライトアクセスの制御
  - ATT.UX: “0” 固定です。ユーザ実行アクセスは許可されません。
  - ATT.PR: “1”固定です。特権リードは常に許可されます。
  - ATT.PW: 特権ライトアクセスの制御
  - ATT.PX: “0”固定です。特権実行アクセスは許可されません。
  - ATT.NS: セキュアアクセスの制御

上記は SMPU マスタ構造の例です。

マスタ構造の保護領域は固定です。読出しアクセスは常に許可され、実行アクセスは許可されません。SMPU は有効または無効に設定できますが、PPU は無効にできません。

5 保護ユニットの設定例

保護ユニットの使用例を、以下の使用仮定に従って説明します。

注: ここで示すアドレスや周辺チャネル番号は CYT2B シリーズのものです。実際のアドレスと周辺チャネル番号は、technical reference manual を参照してください。

5.1 CPU の一部として実装される MPU の設定例

ここでは、オペレーティングシステム (OS) が使用する領域を、タスクアクセスから保護する方法および MPU の設定例を示します。

5.1.1 使用例

ここでは、MPU の設定例を示します。MPU は、ユーザ/特権, リード/ライトおよび実行アクセスを区別します。表 9 に、MPU のアクセス制限を示します。

表 9 CPU の一部として実装される MPU のアクセス制限例

領域	属性
領域 0 (バックグラウンドアドレス) ベースアドレス: 0x00000000 サイズ: 4GB	特権: リード/ライト ユーザ: リード/ライト 実行は許可されます。
領域 1 (コードフラッシュ) ベースアドレス: 0x10000000 サイズ: 8MB	特権: リードのみ ユーザ: リードのみ y 実行は許可されます。
領域 2 (ワークフラッシュ) ベースアドレス: 0x14000000 サイズ: 256KB	特権: リードのみ ユーザ: アクセス禁止 実行は許可されません。
領域 3 (SRAM) ベースアドレス: 0x08000000 サイズ: 1MB	特権: リード/ライト ユーザ: リード/ライト 実行は許可されます。
領域 4 (周辺機能レジスタ) ベースアドレス: 0x40000000 サイズ: 64MB	特権: リード/ライト ユーザ: リード/ライト 実行は許可されません。
領域 5 (Arm®システムレジスタ) ベースアドレス: 0xE0000000 サイズ: 512MB	特権: リード/ライト ユーザ: リード/ライト 実行は許可されません。
その他の領域 (未使用)	-

領域 0 はバックグラウンドとして使用されます。別の領域によって属性が指定されない場合、バックグラウンド領域が適用されます。

CPU の一部として実装される MPU が有効の時、未設定領域へのアクセスはアクセス違反の要因となります。意図しないアクセス違反を防止するため、このタイプの MPU はオーバラッピングをサポートします。これにより、大きい番号の優先順位が最も高く、小さい番号 (領域 0) の優先順位が最も低くなります。領域 0 とオーバラップする領域は、領域 0 よりも高い優先順位を持ちます。つまり、領域 0 はバックグラウンド領域 (全領域の属性指定) として使用できます。

MPU の詳細については、CM4, CM7, および CM0+ の Arm® ドキュメンテーションセットを参照してください。

領域 1 は、特権/ユーザアクセスの両方で読出し専用です。実行は許可されますがデータを書込みできません。領域 2 は、特権アクセスの読出し専用です。ユーザはアクセスできません。特権/ユーザアクセスの両方でデータは書込みできず、実行も許可されません。領域 3 は特権/ユーザアクセスいずれもアクセスでき、実行も許可されます。領域 4 および 5 は、特権/ユーザアクセスいずれもアクセスできますが、実行は許可されません。

5.1.2 設定手順

このタイプの MPU は、CPU 固有のレジスタによって設定されます。設定中は MPU を無効にする必要があり、特権レベルでのアクセスが必要です。図 13 に、MPU の設定方法例を示します。

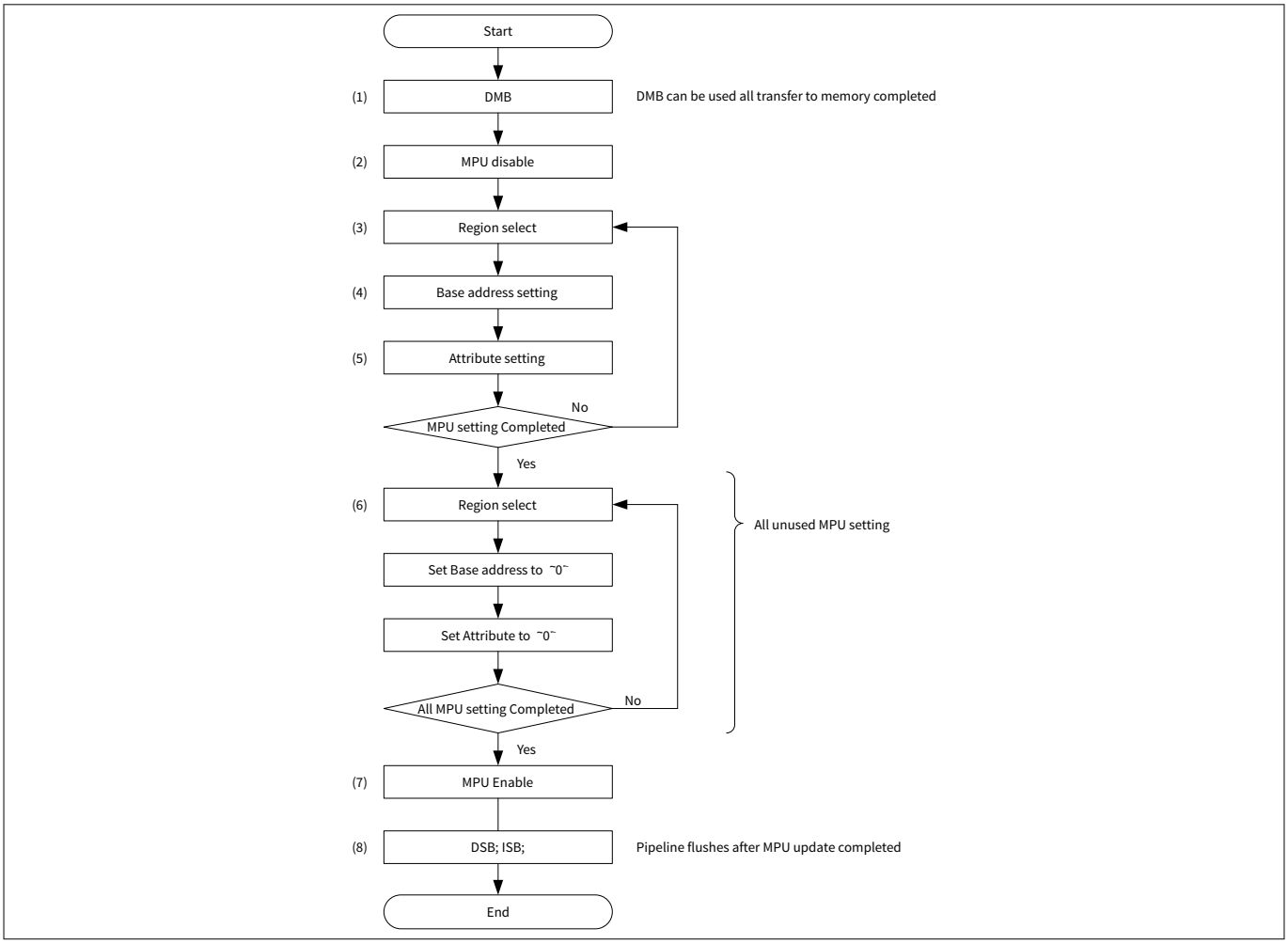

図 13 CPU の一部として実装される MPU の設定手順例

設定中は MPU を無効にする必要があります。はじめに、MPU_RNR レジスタで設定する領域を指定します。MPU_RNR を設定後、MPU_RBAR, MPU_RASR レジスタによってベースアドレス, 領域サイズ, およびアクセス属性を設定します。各領域に対して、この設定を繰り返します。また、未使用領域の MPU_RBAR および MPU_RASR レジスタは“0”に設定します。最後に MPU を有効にします。

また MPU_RBAR レジスタで直接 MPU を設定する領域番号を指定することもできます。MPU_RASR は、Normal memory, Strongly-ordered, および Device といったサブリージョンとメモリ属性の設定にも使用します。

詳細については、Arm® ドキュメンテーションセット (CM4, CM7, および CM0+) を参照してください。

5.1.3 設定

表 10 および表 11 に MPU 設定の SDL 設定部のパラメータと関数を示します。

5 保護ユニットの設定例

表 10　　パラメーター覧

パラメータ	説明	値
CY_MPU_MAX_NUM	MPU がサポートする領域数を定義 MPU_TYPE.DREGION の値	-
MPU_RBAR_ADDR_Msk	ベースアドレスマスクを定義	(0x7FFFFFFUL << MPU_RBAR_ADDR_Pos) MPU_RASR_SRD_Pos = 5u
MPU_RASR_SRD_Pos	MPU_RASR のサブリージョン無効フィールドの位置を定義。	8ul
MPU_CTRL_ENABLE_Msk	MPU 有効ビットマスクを定義	1ul
CY_MPU_DISABLE_USE_DEFAULT_MAP	MPU_CTRL のデフォルトメモリマップへの特権アクセスを有効にする。	0ul
CY_MPU_DISABLED_DURING_FAULT_NMI	MPU_CTRL のハードフォールト, NMI, および FAULTMASK ハンドラ中の MPU 操作を有効にする。	0ul
BACKGROUND_REGION_ADDR	バックグラウンドのベースアドレスを定義	0x00000000ul
CODE_FLASH_REGION_ADDR	コードフラッシュのベースアドレスを定義	0x10000000ul
WORK_FLASH_REGION_ADDR	ワークフラッシュのベースアドレスを定義	0x14000000ul
SRAM_REGION_ADDR	SRAM ベースアドレスを定義	0x08000000ul
PERI_REGISTER_REGION_ADDR	周辺機能レジスタのベースアドレスを定義	0x40000000ul
ARM_SYS_REGISTER_REGION_ADDR	Arm®システムレジスタのベースアドレスを定義	0xE0000000ul
BACKGROUND_MPU_NO	バックグラウンドの領域番号を定義	0ul
CODE_FLASH_MPU_NO	コードフラッシュの領域番号を定義	1ul
WORK_FLASH_MPU_NO	ワークフラッシュの領域番号を定義	2ul
SRAM_MPU_NO	SRAM の領域番号を定義	3ul
PERI_REGISTER_MPU_NO	周辺機能レジスタの領域番号を定義	4ul
ARM_SYS_REGISTER_MPU_NO	Arm®システムレジスタの領域番号を定義	5ul
g_mpuCfg.addr	ベースアドレスの設定	–
g_mpuCfg.size	領域サイズの設定	-

(続く)

表 10 (続き) パラメーター覧

パラメータ	説明	値
g_mpuCfg.permission	領域のアクセス許可を設定 CY_MPU_ACCESS_P_FULL_ACCESS: 特権: リード/ライト, ユーザ: リード/ライト CY_MPU_ACCESS_P_PRIV_RO: 特権: リードのみ, ユーザ: アクセス禁止 CY_MPU_ACCESS_P_RO: 特権: リードのみ, ユーザ: リードのみ	-
g_mpuCfg.attribute	領域のメモリアクセス属性を設定 CY_MPU_ATTR_NORM_MEM_WT: 通常, 共有不可, 外部および内部ライトスルー。書き込み割り当てはありません。 CY_MPU_ATTR_SHR_DEV: デバイス、共有可能。 CY_MPU_ATTR_STR_ORD_DEV: 強く順序付けられており、共有可能。	-
g_mpuCfg.execute	領域の命令アクセス禁止設定 CY_MPU_INST_ACCESS_EN: 命令フェッチ許可 CY_MPU_INST_ACCESS_DIS: 命令フェッチ禁止	-
g_mpuCfg.srd	サブリージョン無効化を設定	-
g_mpuCfg.enable	領域有効を設定 CY_MPU_ENABLE: 領域有効 CY_MPU_DISABLE: 領域無効	-

5 保護ユニットの設定例

表 11　　関数一覧

機能	説明	備考
`Cy_MPU_Setup (cfg[], cfgSize, privDefMapEn, faultNmiEn)`	MPU 設定 cfg[]: MPU 設定パラメータアドレス cfgSize: 設定パラメータのサイズ privDefMapEn: デフォルトのメモリマップへの特権ソフトウェアアクセスを有効にします。 faultNmiEn: ハードフォールト、NMI、および FAULTMASK ハンドラ中に MPU の操作を有効にします。	-

以下のコードは MPU の設定例を示します。

5 保護ユニットの設定例

Code Listing 9: MPU の設定例

```
#define MPU_TYPE_DREGION_Pos  8UL                                /*!< MPU TYPE: DREGION Position
*/
#define MPU_TYPE_DREGION_Msk  (0xFFUL << MPU_TYPE_DREGION_Pos)  /*!< MPU TYPE: DREGION Mask */
#define CY_MPU_MAX_NUM        ((MPU->TYPE & MPU_TYPE_DREGION_Msk) >> MPU_TYPE_DREGION_Pos)

#define MPU_RBAR_ADDR_Pos     5UL                                /*!< MPU RBAR: ADDR Position */
#define MPU_RBAR_ADDR_Msk     (0x7FFFFFFUL << MPU_RBAR_ADDR_Pos)/*!< MPU RBAR: ADDR Mask */

#define MPU_RASR_SRD_Pos      8UL   /*!< MPU RASR: Sub-Region Disable Position */
#define MPU_CTRL_ENABLE_Msk   (1UL  /*<< MPU_CTRL_ENABLE_Pos*/)  /*!< MPU CTRL: ENABLE Mask */

#define MPU_CFG_ARRAY_SIZE(array) (sizeof(array)/sizeof(cy_stc_mpu_region_cfg_t))

/** Specifies enable/disable privileged software access to the default memory map */
typedef enum
{
    CY_MPU_DISABLE_USE_DEFAULT_MAP = (0ul), /**< If the MPU is enabled, disables use of the
default memory map. Any memory access to a location not covered by any enabled region causes a
fault.  */
:
} cy_en_mpu_privdefena_t;

/** Specifies enable/disable the operation of MPU during hard fault, NMI, and FAULTMASK
handlers. */
typedef enum
{
    CY_MPU_DISABLED_DURING_FAULT_NMI = (0ul),  /**< MPU is disabled during hard fault, NMI, and
FAULTMASK handlers, regardless of the value of the ENABLE bit.  */
:
} cy_en_mpu_hfnmiena_t;

/*  Define each region base address  */
#define BACKGROUND_REGION_ADDR       (0x00000000ul) // Back Ground Region Start Address
#define CODE_FLASH_REGION_ADDR       (0x10000000ul) // Code Flash Region Start Address
#define WORK_FLASH_REGION_ADDR       (0x14000000ul) // Work Flash Region Start Address
#define SRAM_REGION_ADDR             (0x08000000ul) // System RAM Region Start Address
#define PERI_REGISTER_REGION_ADDR    (0x40000000ul) // Peripheral Register Region Start Address
#define ARM_SYS_REGISTER_REGION_ADDR (0xE0000000ul) // ARM System Registers Region Start Address

/*  Define each region number  */
#define BACKGROUND_MPU_NO       (0)
#define CODE_FLASH_MPU_NO       (1)
#define WORK_FLASH_MPU_NO       (2)
#define SRAM_MPU_NO             (3)
#define PERI_REGISTER_MPU_NO    (4)
#define ARM_SYS_REGISTER_MPU_NO (5)

cy_stc_mpu_region_cfg_t g_mpuCfg[] =
{
    /*  Region 0 configuration */
    /*** Back Ground Region ***/
```

```
    {
        .addr       = BACKGROUND_REGION_ADDR,
        .size       = CY_MPU_SIZE_4GB,
        .permission = CY_MPU_ACCESS_P_FULL_ACCESS,
        .attribute  = CY_MPU_ATTR_NORM_MEM_WT,
        .execute    = CY_MPU_INST_ACCESS_EN,
        .srd        = 0x00u,
        .enable     = CY_MPU_ENABLE
    },

    /*  Region 1 configuration  */
    /*** Code Flash Region ***/
    {
        .addr       = CODE_FLASH_REGION_ADDR,
        .size       = CY_MPU_SIZE_8MB,
        .permission = CY_MPU_ACCESS_P_RO,
        .attribute  = CY_MPU_ATTR_NORM_MEM_WT,
        .execute    = CY_MPU_INST_ACCESS_EN,
        .srd        = 0x00u,
        .enable     = CY_MPU_ENABLE
    },

    /*  Region 2 configuration  */
    /*** Work Flash Region ***/
    {
        .addr       = WORK_FLASH_REGION_ADDR,
        .size       = CY_MPU_SIZE_256KB,
        .permission = CY_MPU_ACCESS_P_PRIV_RO,
        .attribute  = CY_MPU_ATTR_NORM_MEM_WT,
        .execute    = CY_MPU_INST_ACCESS_DIS,
        .srd        = 0x00u,
        .enable     = CY_MPU_ENABLE
    },

    /*  Region 3 configuration  */
    /*** System RAM Region ***/
    {
        .addr       = SRAM_REGION_ADDR,
        .size       = CY_MPU_SIZE_1MB,
        .permission = CY_MPU_ACCESS_P_FULL_ACCESS,
        .attribute  = CY_MPU_ATTR_NORM_MEM_WT,
        .execute    = CY_MPU_INST_ACCESS_EN,
        .srd        = 0x00u,
        .enable     = CY_MPU_ENABLE
    },

    /*  Region 4 configuration  */
    /*** Peripheral Register Region ***/
    {
        .addr       = PERI_REGISTER_REGION_ADDR,
        .size       = CY_MPU_SIZE_64MB,
        .permission = CY_MPU_ACCESS_P_FULL_ACCESS,
        .attribute  = CY_MPU_ATTR_SHR_DEV,
```

```
        .execute    = CY_MPU_INST_ACCESS_DIS,
        .srd        = 0x00u,
        .enable     = CY_MPU_ENABLE
    },

    /*  Region 5 configuration  */
    /*** ARM System Registers Region ***/
    {
        .addr       = ARM_SYS_REGISTER_REGION_ADDR,
        .size       = CY_MPU_SIZE_512MB,
        .permission = CY_MPU_ACCESS_P_FULL_ACCESS,
        .attribute  = CY_MPU_ATTR_STR_ORD_DEV,
        .execute    = CY_MPU_INST_ACCESS_DIS,
        .srd        = 0x00u,
        .enable     = CY_MPU_ENABLE
    },
};

int main(void)
{
    SystemInit();

    __enable_irq();

    /********* Core MPU setting **************/
    /*  Configure MPU See Code Listing 10  */
    CY_ASSERT(Cy_MPU_Setup(g_mpuCfg, MPU_CFG_ARRAY_SIZE(g_mpuCfg),
CY_MPU_DISABLE_USE_DEFAULT_MAP, CY_MPU_DISABLED_DURING_FAULT_NMI) == CY_MPU_SUCCESS);
:
    for(;;);
}
```

5 保護ユニットの設定例

Code Listing 10 Cy_MPU_Setup()関数

```
cy_en_mpu_status_t Cy_MPU_Setup(const cy_stc_mpu_region_cfg_t cfg[], uint8_t cfgSize,
cy_en_mpu_privdefena_t privDefMapEn, cy_en_mpu_hfnmiena_t faultNmiEn)
{
:
    // Ensure all memory accesses are completed before new memory access is committed
    __DMB();   /*  (1) Run Data memory barrier instruction  */

    // Disable the MPU
    MPU->CTRL = 0ul;   /*  (2) Disable MPU  */

    uint32_t i_mpuRegionNo;
    for(i_mpuRegionNo = 0ul; i_mpuRegionNo < CY_MPU_MAX_NUM; i_mpuRegionNo++)
    {
        // Select which MPU region to configure
        MPU->RNR = i_mpuRegionNo;   /*  (3) Select Region by MPU_RNR register  */

        if(i_mpuRegionNo < cfgSize)
        {
            // Configure  region  base  address register
            // VALID and REGION field of RBAR register will be 0 since this function sets RNR
register manually.
            MPU->RBAR = (cfg[i_mpuRegionNo].addr & MPU_RBAR_ADDR_Msk);   /*  (4) Set base
address of this region  */

            uint32_t srd;
            if(cfg[i_mpuRegionNo].size < CY_MPU_SIZE_256B)
            {
                srd = 0ul;
            }
            else
            {
                srd = (cfg[i_mpuRegionNo].srd << MPU_RASR_SRD_Pos);
            }

            // Configure region attribute and size register
                     /*  (5) Set access attribute of this region  */
            MPU->RASR = ((uint32_t)cfg[i_mpuRegionNo].size       |
                         (uint32_t)cfg[i_mpuRegionNo].permission |
                         (uint32_t)cfg[i_mpuRegionNo].attribute  |
                         srd                                     |
                         (uint32_t)cfg[i_mpuRegionNo].enable);
        }
   /*  (6) Unused MPU setting  */
        else // Disables unused regions
        {
            // Configure region base address register
            MPU->RBAR = 0ul;

            // Configure region attribute and size register
            MPU->RASR = 0ul;
        }
```

```
    }

    // Enable the MPU
      /*  (7) Enabling MPU  */
    MPU->CTRL = ((uint32_t)privDefMapEn | (uint32_t)faultNmiEn | MPU_CTRL_ENABLE_Msk);

    // Ensure all memory accesses are completed before next instruction is executed
    __DSB();   /*  (8) Run Data memory barrier and Instruction Synchronization Barrier
instruction  */

    // Flush the pipeline and ensure all previous instructions are completed before executing
new instructions
    __ISB();   /*  (8) Run Data memory barrier and Instruction Synchronization Barrier
instruction  */

    return CY_MPU_SUCCESS;
}
```

5.2 バスインフラストラクチャの一部として実装される MPU の設定

この MPU は、MPU.ADDR および MPU.ATT レジスタによって設定され、テストコントローラによって使用されます。ただし、通常この MPU はセキュリティ要件に応じてセキュア CPU である CM0+により設定されます。

5.3 SMPU の設定例

SMPU は、メモリ保護機能を提供し、すべてのバスマスタによって共有されます。すべてのバスマスタは、各領域で同じ制限を受けます。

SMPU はマスタ/スレーブによる保護ペア構造を持ちます。したがって、スレーブへの設定はマスタの設定によって制限されます。

5.3.1 使用上の想定

SMPU は、ユーザ/特権, セキュア/非セキュア, および保護コンテキストを区別します。

表 12 に、SMPU のアクセス制限の例を示します。

表 12 SMPU のアクセス制限例

領域	特権	ユーザ	セキュア	許可される保護コンテキスト	PC_MATCH	リソース
領域 2 ベースアドレス: 0x08019000 サイズ: 4KB	リード/ライト 実行は許可されます。	リード/ライト 実行は許可されません。	非セキュア	PC = 6	Access 評価	SRAM
領域 3 ベースアドレス: 0x08018000 サイズ: 4KB	リード/ライト 実行は許可されます。	リード/ライト 実行は許可されません。	非セキュア	PC = 5	Access 評価	SRAM

領域 2 および 3 は、保護コンテキストによってアクセスが制限されます。

領域 2 は保護コンテキスト = 6、領域 3 は保護コンテキスト = 5 でアクセス可能です。

5.3.2 SMPU の設定手順

図 14 に設定手順例を示します。

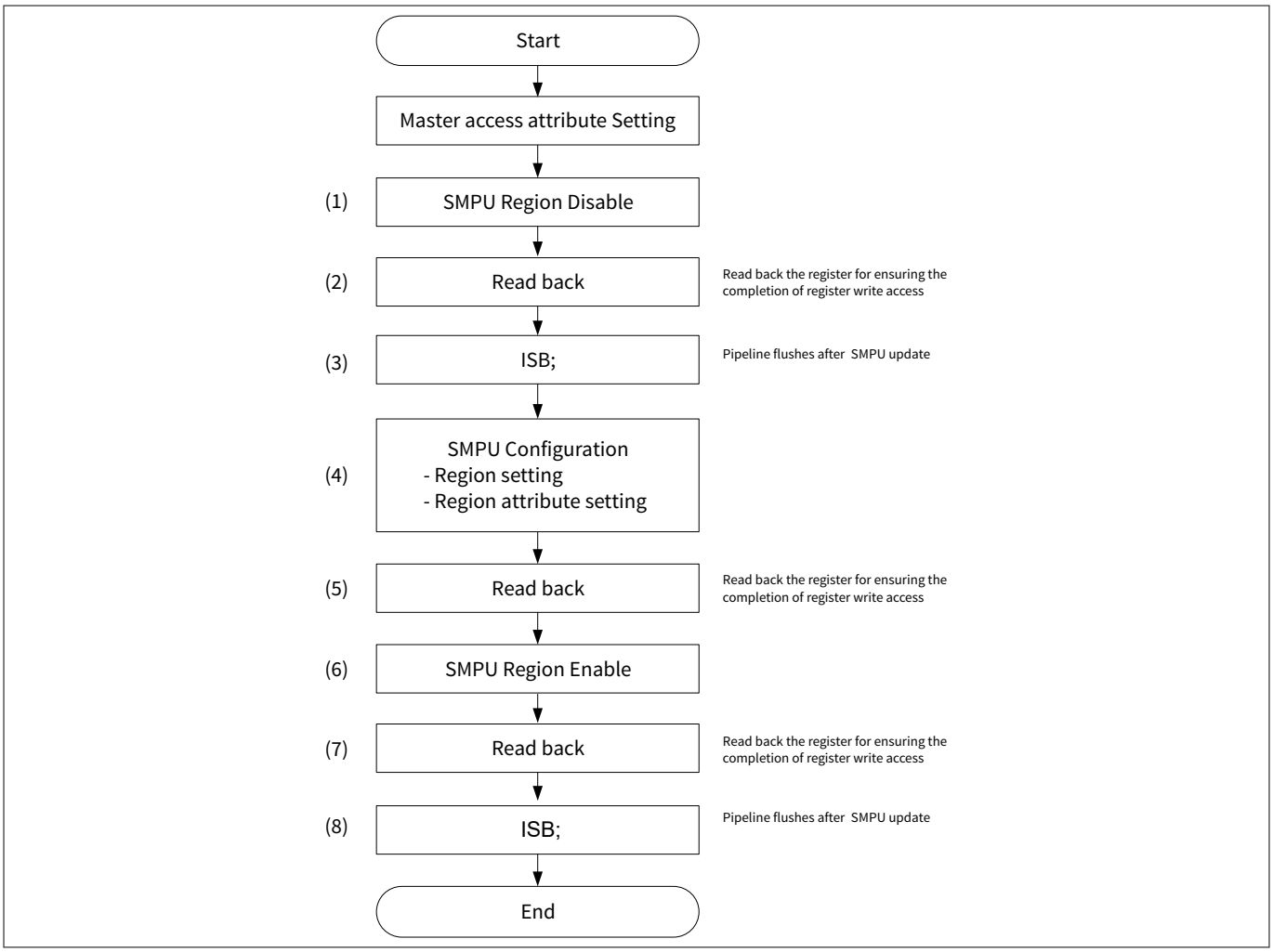

図 14 SMPU 設定手順例

スレーブ構造 (PROT_SMPU_STRUCTx_ADDR0 および PROT_SMPU_STRUCTx_ATT0) を設定するためのアクセス属性は、マスタ構造 (PROT_SMPU_STRUCTx_ADDR1 および PROT_SMPU_STRUCTx_ATT1) によって許可されます。

SMPU 設定後にレジスタライトアクセスを完了するため、レジスタをリードバックする必要があります。

5.3.3 設定

表 13 および表 14 に SMPU 設定の SDL 設定部のパラメータと関数を示します。

表 13 パラメーター覧

パラメータ	説明	値
MASTER_ID_OF_THIS_CPU	PROT_SMPU_MSx_CTL を設定するマスタを定義	CPUSS_MS_ID_CM0 (CM0+)

(続く)

表 13 (続き) パラメーター一覧

パラメータ	説明	値
TP_PRIVILEGED	PROT_SMPU_MSx_CTL.P 値を定義	1ul (特権モード)
TP_SECURE	PROT_SMPU_MSx_CTL.NS 値を定義	0ul (非セキュア)
TP_PROT_CONTEXT	PROT_SMPU_MSx_CTL.PC の値を定義	6ul
TP_PERMITTED_ADDR	SMPU 領域 2 のベースアドレスを定義	0x08019000UL
TP_PERMITTED_CONTEXT	領域 2 にアクセス可能な PC 値を定義	6ul
TP_PROHIBITED_ADDR	SMPU 領域 3 のベースアドレスを定義	0x08018000UL
TP_PROHIBITED_CONTEXT	領域 3 にアクセス可能な PC 値を定義	5ul
smpuStruct(2 or 3)Config.address	領域 2 または 3 のベースアドレスを設定	領域 2: TP_PERMITTED_ADD 領域 3: TP_PROHIBITED_ADDR
smpuStruct(2 or 3)Config.regionSize	領域 2 または 3 のサイズを設定	CY_PROT_SIZE_4KB (4 KB)
smpuStruct(2 or 3)Config.subregions	領域 2 または 3 のサブリージョンを設定	0x00ul (未使用)
smpuStruct(2 or 3)Config.userPermission	領域 2 または 3 のユーザアクセスの属性を設定	CY_PROT_PERM_RWX (フルアクセス)
smpuStruct(2 or 3)Config.privPermission	領域 2 または 3 の特権アクセスの属性を設定	CY_PROT_PERM_RWX (フルアクセス)
smpuStruct(2 or 3)Config.secure	領域 2 または 3 の非セキュアアクセスの属性を設定	0ul (非セキュア)
smpuStruct(2 or 3)Config.pcMatc	領域 2 または 3 の PC_MATCH を設定	0ul (アクセス評価)

(続く)

表 13 (続き) パラメーター一覧

パラメータ	説明	値
smpuStruct(2 or 3)Config.pcMask	領域 2 または 3 の PC_MASK を設定	領域 2: 1ul << (TP_PERMITTED_CONTEXT - 1) (保護コンテキスト = 6 は許可) 領域 3: 1ul << (TP_PROHIBITED_CONTEXT - 1) (保護コンテキスト = 5 は許可)
PROT_SMPU_SMPU_STRUCT2	SMPU (領域 2) レジスタベースアドレス設定	0x40232080ul
PROT_SMPU_SMPU_STRUCT3	SMPU (領域 3) レジスタベースアドレス設定	0x402320C0ul

表 14 関数一覧

機能	説明	値
`Cy_Prot_ConfigBusMaster(busMaster, privileged, secure, pcMask)`	PROT_SMPU_MS0_CTL レジスタ設定 busMaster; 設定レジスタ番号表示 特権: P フィールド設定値 セキュア: NS フィールド設定値 pcMask: 関連するマスタによって設定可能な保護コンテキストを指定	busMaster; MASTER_ID_OF_THIS_CPU 特権: TP_PRIVILEGED セキュア: TP_SECURE pcMask: 1 << (TP_PROT_CONTEXT-1) (保護コンテキスト = 6)
`Cy_Prot_DisableSmpuSlaveStruct(*base)`	SMPU 領域無効化 *base: SMPU レジスタベースアドレス	*base: PROT_SMPU_SMPU_STRUCT2 または PROT_SMPU_SMPU_STRUCT3
`Cy_Prot_EnableSmpuSlaveStruct(*base)`	SMPU 領域有効化 *base: SMPU レジスタベースアドレス	*base: PROT_SMPU_SMPU_STRUCT2 または PROT_SMPU_SMPU_STRUCT3
`Cy_Prot_ConfigSmpuSlaveStruct(*base, config)`	SMPU 設定 *base: SMPU レジスタベースアドレス Config: 設定データ	*base: PROT_SMPU_SMPU_STRUCT2 または PROT_SMPU_SMPU_STRUCT3 Config:smpuStruct(2 or 3)Config

5 保護ユニットの設定例

Code Listing 11 に SMPU の設定例を示します。

5 保護ユニットの設定例

Code Listing 11: SMPU の設定例

```
typedef enum
    /*  Selection of Master CPU ID  */
{
    CPUSS_MS_ID_CM0                 =  0ul,
    CPUSS_MS_ID_CRYPTO              =  1ul,
    CPUSS_MS_ID_DW0                 =  2ul,
    CPUSS_MS_ID_DW1                 =  3ul,
    CPUSS_MS_ID_DMAC                =  4ul,
    CPUSS_MS_ID_SLOW0               =  5ul,
    CPUSS_MS_ID_SLOW1               =  6ul,
    CPUSS_MS_ID_CM4                 = 14ul,
    CPUSS_MS_ID_TC                  = 15ul
} en_prot_master_t;

    /*  Define Master CPU ID to CM0+  */
#define MASTER_ID_OF_THIS_CPU CPUSS_MS_ID_CM0

    /*  Selection of SMPU structure attribute  */
typedef enum
{
    CY_PROT_PERM_DISABLED = 0x00ul, /**< Read, Write and Execute disabled */
    CY_PROT_PERM_R        = 0x01ul, /**< Read enabled */
    CY_PROT_PERM_W        = 0x02ul, /**< Write enabled */
    CY_PROT_PERM_RW       = 0x03ul, /**< Read and Write enabled */
    CY_PROT_PERM_X        = 0x04ul, /**< Execute enabled */
    CY_PROT_PERM_RX       = 0x05ul, /**< Read and Execute enabled */
    CY_PROT_PERM_WX       = 0x06ul, /**< Write and Execute enabled */
    CY_PROT_PERM_RWX      = 0x07ul  /**< Read, Write and Execute enabled */
}cy_en_prot_perm_t;

#define TP_PRIVILEGED         (1ul)             /* privileged */
#define TP_SECURE             (0ul)             /* non secure */
#define TP_PROT_CONTEXT       (6ul)             /* enable context 6 */

/* This area is going to be prohibited accessing from a master who has TP_PROHIBITED_CONTEXT as
context */
#define TP_PROHIBITED_ADDR    (0x08018000UL)
#define TP_PROHIBITED_CONTEXT (5ul)

/* This area is going to be permitted accessing from a master who has TP_PERMITTED_CONTEXT as
context */
#define TP_PERMITTED_ADDR     (0x08019000UL)
#define TP_PERMITTED_CONTEXT  (TP_PROT_CONTEXT)

const cy_stc_smpu_cfg_t smpuStruct2Config =
    /*  Configure SMPU for Region 2.  */
{
    .address        = (uint32_t*)(TP_PERMITTED_ADDR),
    .regionSize     = CY_PROT_SIZE_4KB,                 // 4KB: 0x1000 Byte
    .subregions     = 0x00ul,
```

```
    .userPermission = CY_PROT_PERM_RWX,
    .privPermission = CY_PROT_PERM_RWX,
    .secure         = 0ul,                                  // Non secure
    .pcMatch        = 0ul,
    .pcMask         = 1ul << (TP_PERMITTED_CONTEXT - 1),  // enable context
"TP_PERMITTED_CONTEXT"
};

const cy_stc_smpu_cfg_t smpuStruct3Config =
    /*  Configure SMPU for Region 3.  */
{
    .address        = (uint32_t*)(TP_PROHIBITED_ADDR),
    .regionSize     = CY_PROT_SIZE_4KB,                 // 4KB: 0x1000 Byte
    .subregions     = 0x00ul,
    .userPermission = CY_PROT_PERM_RWX,
    .privPermission = CY_PROT_PERM_RWX,
    .secure         = 0ul,                                  // Non secure
    .pcMatch        = 0ul,
    .pcMask         = 1ul << (TP_PROHIBITED_CONTEXT - 1), // enable context
"TP_PERMITTED_CONTEXT"
};

int main(void)
{
    SystemInit();

    cy_en_prot_status_t status;
:
    /*************************/
    /* 1. Setting for MSx_CTL */
    /*************************/
    /* 1.1 Setting for MSx_CTL (for this CPU) to allow the PC value to become "TP_PROT_CONTEXT"
*/
    /*  Set PROT_SMPU_MS0_CTL.PC_MASK. See Code Listing 12.  */
    status = Cy_Prot_ConfigBusMaster(MASTER_ID_OF_THIS_CPU, TP_PRIVILEGED, TP_SECURE, 1 <<
(TP_PROT_CONTEXT-1));
    CY_ASSERT(status == CY_PROT_SUCCESS);

    /*************************/
    /* 2. Setting for MPU PC  */
    /*************************/
    /*  Change protection context to TP_PROT_CONTET (PC=6). See Code Listing 7.  */
    /* 2.1 Setting for MPU so that this CPU's PC value becomes "TP_PROT_CONTEXT" */
    status = Cy_Prot_SetActivePC(MASTER_ID_OF_THIS_CPU, TP_PROT_CONTEXT);
    CY_ASSERT(status == CY_PROT_SUCCESS);

    /*******************************/
    /* 3. Setting for SMPU_STRUCT 2 */
    /*******************************/
    /* 3.1 Disable SMPU_STRUCT 2 */
    /*  SMPU Region 2 disabled. See Code Listing 13.  */
    status = Cy_Prot_DisableSmpuSlaveStruct(PROT_SMPU_SMPU_STRUCT2);
    CY_ASSERT(status == CY_PROT_SUCCESS);
```

```
    /* 3.2 Setting SMPU_STRUCT 2 for PERMITTED area */
    /*  Configure SMPU Region 2. See Code Listing 14.  */
    status = Cy_Prot_ConfigSmpuSlaveStruct(PROT_SMPU_SMPU_STRUCT2, &smpuStruct2Config);
    CY_ASSERT(status == CY_PROT_SUCCESS);



    /* 3.3 Enable SMPU_STRUCT 2 */
    /*  SMPU Region 2 enabled. See Code Listing 15.  */
    status = Cy_Prot_EnableSmpuSlaveStruct(PROT_SMPU_SMPU_STRUCT2);
    CY_ASSERT(status == CY_PROT_SUCCESS);

    /*******************************/
    /* 4. Setting for SMPU_STRUCT 3 */
    /*******************************/
    /* 4.1 Disable SMPU_STRUCT 3 */
    /*  SMPU Region 3 disabled. See Code Listing 13.  */
    status = Cy_Prot_DisableSmpuSlaveStruct(PROT_SMPU_SMPU_STRUCT3);
    CY_ASSERT(status == CY_PROT_SUCCESS);

    /* 4.2 Setting SMPU_STRUCT 3 for PROHIBITED area */
    /*  Configure SMPU Region 3. See Code Listing 14.  */
    status = Cy_Prot_ConfigSmpuSlaveStruct(PROT_SMPU_SMPU_STRUCT3, &smpuStruct3Config);
    CY_ASSERT(status == CY_PROT_SUCCESS);

    /* 4.3 Enable SMPU_STRUCT 3 */
    /*  SMPU Region 3 enabled. See Code Listing 15  */
    status = Cy_Prot_EnableSmpuSlaveStruct(PROT_SMPU_SMPU_STRUCT3);
    CY_ASSERT(status == CY_PROT_SUCCESS);
:
    for(;;);
}
```

5 保護ユニットの設定例

Code Listing 12: Cy_Prot_ConfigBusMaster()関数

```
cy_en_prot_status_t Cy_Prot_ConfigBusMaster(en_prot_master_t busMaster, bool privileged, bool
secure, uint32_t pcMask)
{
    cy_en_prot_status_t status = CY_PROT_SUCCESS;
    un_PROT_SMPU_MS0_CTL_t tProtSmpuMs0Ctl = {0};
    uint32_t * addrMsCtl = (uint32_t *)(PROT_BASE + (uint32_t)((uint32_t)busMaster <<
CY_PROT_MSX_CTL_SHIFT));
:
        tProtSmpuMs0Ctl.stcField.u1NS = !secure;
        tProtSmpuMs0Ctl.stcField.u1P = privileged;
       /*  Set available PC values to PROT_SMPU_MS0_CTL.PC_MASK. (*) */
        tProtSmpuMs0Ctl.stcField.u15PC_MASK_15_TO_1 = pcMask;

        *addrMsCtl = tProtSmpuMs0Ctl.u32Register; // regVal;
             /*   Read back  */
        status = ((*addrMsCtl != tProtSmpuMs0Ctl.u32Register) ? CY_PROT_FAILURE :
CY_PROT_SUCCESS);
:
    return status;
}
```

注: (*) このプロセスは、対応するマスタが設定可能な保護コンテキスト値を指定します。セキュアシステムでは、セキュアマスタによって実行されます。詳細については、バス転送の保護プロパティを参照してください。

Code Listing 13: Cy_Prot_DisableSmpuSlaveStruct() 関数

```
cy_en_prot_status_t Cy_Prot_DisableSmpuSlaveStruct(volatile stc_PROT_SMPU_SMPU_STRUCT_t* base)
{
    cy_en_prot_status_t status = CY_PROT_SUCCESS;

    base->unATT0.stcField.u1ENABLED &= ~CY_PROT_STRUCT_ENABLE; /* (1) Disable SMPU structure  */

    // Check if the SMPU structure really was enabled.
    // Note this also ensures previous write access to complete before execution of below ISB.
    status = (base->unATT0.stcField.u1ENABLED == CY_PROT_STRUCT_ENABLE) ?   /* (2) Read back  */
                CY_PROT_FAILURE : CY_PROT_SUCCESS;

    // Flush the pipeline and ensure all previous instructions are completed before executing
new instructions
    __ISB();   /* (3) Run Instruction Synchronization Barrier instruction  */

    return status;
}
```

5 保護ユニットの設定例

Code Listing 14: Cy_Prot_ConfigSmpuSlaveStruct() 関数

```
cy_en_prot_status_t Cy_Prot_ConfigSmpuSlaveStruct(volatile stc_PROT_SMPU_SMPU_STRUCT_t* base,
const cy_stc_smpu_cfg_t* config)
{
    cy_en_prot_status_t status = CY_PROT_SUCCESS;
    un_PROT_SMPU_SMPU_STRUCT_ADDR0_t tprotSmpuSmpuStruct_ADDR0 = { 0 };
    un_PROT_SMPU_SMPU_STRUCT_ATT0_t  tprotSmpuSmpuStruct_ATT0 = { 0 };

    if(((uint32_t)config->pcMask & CY_PROT_SMPU_PC_LIMIT_MASK) != 0UL)
    {
        /* PC mask out of range - not supported in device */
        status = CY_PROT_BAD_PARAM;
    }
    else
    {

        /*  (4)-1 Set SMPU region  */
        tprotSmpuSmpuStruct_ADDR0.stcField.u8SUBREGION_DISABLE = config->subregions;
        tprotSmpuSmpuStruct_ADDR0.stcField.u24ADDR24 = (uint32_t)((uint32_t)config->address >>
CY_PROT_ADDR_SHIFT);




        /*  (4)-2 Set SMPU region attribute */
        tprotSmpuSmpuStruct_ATT0.stcField.u1PC_MASK_0 = 1; // This value is read only. The
default value is "1".
        tprotSmpuSmpuStruct_ATT0.stcField.u1NS = !(config->secure);
        tprotSmpuSmpuStruct_ATT0.stcField.u15PC_MASK_15_TO_1 = config->pcMask;
        tprotSmpuSmpuStruct_ATT0.stcField.u5REGION_SIZE = config->regionSize;
        tprotSmpuSmpuStruct_ATT0.stcField.u1PC_MATCH = config->pcMatch;
        tprotSmpuSmpuStruct_ATT0.stcField.u1UR = (config->userPermission & CY_PROT_PERM_R);
        tprotSmpuSmpuStruct_ATT0.stcField.u1UW = (config->userPermission & CY_PROT_PERM_W) >> 1;
        tprotSmpuSmpuStruct_ATT0.stcField.u1UX = (config->userPermission & CY_PROT_PERM_X) >> 2;
        tprotSmpuSmpuStruct_ATT0.stcField.u1PR = (config->privPermission & CY_PROT_PERM_R);
        tprotSmpuSmpuStruct_ATT0.stcField.u1PW = (config->privPermission & CY_PROT_PERM_W) >> 1;
        tprotSmpuSmpuStruct_ATT0.stcField.u1PX = (config->privPermission & CY_PROT_PERM_X) >> 2;

        base->unATT0.u32Register = tprotSmpuSmpuStruct_ATT0.u32Register;   // attReg;
        base->unADDR0.u32Register = tprotSmpuSmpuStruct_ADDR0.u32Register; // addrReg;

        status = ((base->unADDR0.u32Register != tprotSmpuSmpuStruct_ADDR0.u32Register) ||
                  (base->unATT0.u32Register != tprotSmpuSmpuStruct_ATT0.u32Register))   //* (5)
Read back */
                    ? CY_PROT_FAILURE : CY_PROT_SUCCESS;
    }

    return status;
}
```

Code Listing 15: Cy_Prot_EnableSmpuSlaveStruct() 関数

```
cy_en_prot_status_t Cy_Prot_EnableSmpuSlaveStruct(volatile stc_PROT_SMPU_SMPU_STRUCT_t* base)
{
    cy_en_prot_status_t status = CY_PROT_SUCCESS;

    base->unATT0.stcField.u1ENABLED = CY_PROT_STRUCT_ENABLE;   /* (6) Enable SMPU structure */

    // Check if the SMPU structure really was enabled.
    // Note this also ensures previous write access to complete before execution of below ISB.
    status = (base->unATT0.stcField.u1ENABLED != CY_PROT_STRUCT_ENABLE) ?   /* (7) Read back */
                CY_PROT_FAILURE : CY_PROT_SUCCESS;

    // Flush the pipeline and ensure all previous instructions are completed before executing
new instructions
    __ISB();   /* (8) Run Instruction Synchronization Barrier instruction */

    return status;
}
```

5.4 PPU の設定例

PPU は、周辺の保護機能を提供し、すべてのバスマスタによって共有されます。

一般的に、周辺機能のレジスタアドレスは固定されます。そのため、PPU は 2 種類あります。

- 固定保護アドレス範囲をもつ Fixed PPU
- プログラム可能な保護アドレス範囲を持つ Programmable PPU

通常、プログラマブル PPU の保護ベースアドレスとサイズは、ブートプロセスで保護コンテキスト 0 を使用して CM0+によって設定されます。

ここでは、Fixed PPU の設定方法について説明します。Fixed PPU は、保護されたアドレス領域が固定されることを除いて Programmable PPU と同じです。

PPU は、マスタ/スレーブ設定の保護ペア構造も持ちます。したがって、スレーブ構造の属性設定はマスタ構造設定によって制限されます。

5.4.1 使用例

PPU は、ユーザ/特権, セキュア/非セキュアおよび保護コンテキストを区別します。Fixed PPU は以下のユースケースに従って説明します。

- Fixed PPU228 を使用 (GPIO_ENH Port#0)
- 保護コンテキスト= 5: 特権およびユーザアクセスは GPIO Port#0 へのアクセスが許可されません。
- 保護コンテキスト= 6: 特権およびユーザアクセスは GPIO Port#0 へのアクセスが許可されます。

表 15 に、本使用例での PPU のアクセス制限の例を示します。

表 15 PPU のアクセス制限例

設定する PPU	保護コンテキスト	特権	ユーザ	セキュア
PPU_FX228	PC = 5	アクセス禁止	アクセス禁止	非セキュア
	PC = 6	リード/ライト	リード/ライト	

注: 使用する製品の実際のアドレスとペリフェラルチャネル番号についてはデータシートを参照してください。

5.4.2 PPU の設定手順

図 15 に設定手順例を示します。

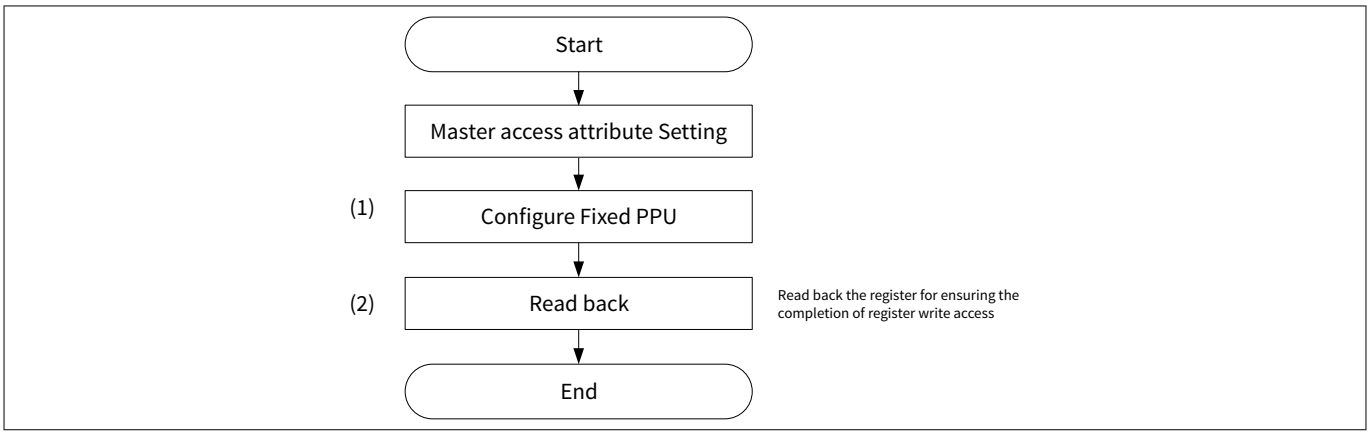

図 15 PPU 設定手順例

スレーブ構造 (PROT_PPU_FXx_SL_ATT0) を設定するためのアクセス属性は、マスタ構造 (PROT_PPU_FXx_MS_ATT0) によって許可されます。

PPU 設定後にレジスタライトアクセスを完了するため、レジスタをリードバックする必要があります。

5.4.3 設定

表 16 および表 17 に Fixed PPU 設定の SDL 設定部のパラメータと関数を示します。

表 16 パラメーター覧

パラメータ	説明	値
MASTER_ID_OF_THIS_CPU	PROT_SMPU_MSx_CTL を設定するマスタを定義	CPUSS_MS_ID_CM0 (CM0+)
PERI_MS_PPU_FX_GPIO_PRT0_PRT	対象の Fixed PPU のレジスタアドレスを定義設定	(volatile stc_PERI_MS_PPU_FX_t*) &PERI_MS->PPU_FX[228] (Fixed PPU 228 のベースアドレスを示します)
TP_PRIVILEGED	PROT_SMPU_MSx_CTL.P 値を定義	1ul (特権モード)
TP_SECURE	PROT_SMPU_MSx_CTL.NS 値を定義	0ul (非セキュア)
TP_PERMITTED_PC	アクセスを許可される保護コンテキストを定義	CY_PROT_PC6 (6ul)
TP_PROHIBITED_PC	アクセスを禁止される保護コンテキストを定義	CY_PROT_PC5 (5ul)

(続く)

表 16 (続き) パラメーター一覧

パラメータ	説明	値
TP_PERMITTED_PC_MASK	バスマスタの PROT_SMPU_MSx_CTL.PC_MASK 値を定義	1ul << (TP_PERMITTED_PC-1ul)
TP_PROHIBITED_PC_MASK	バスマスタの PROT_SMPU_MSx_CTL.PC_MASK 値を定義	1ul << (TP_PROHIBITED_PC-1ul)
ppuFixedAttr_AllEnable.userPermission	Fixed PPU の対象 PC に対するユーザアクセス属性を設定	CY_PROT_PERM_RWX (フルアクセス)
ppuFixedAttr_AllEnable.privPermission	Fixed PPU の対象 PC に対する特権アクセス属性を設定	CY_PROT_PERM_RWX (フルアクセス)
ppuFixedAttr_AllEnable.secure	Fixed PPU の対象 PC に対する非セキュアアクセス属性を設定	TP_SECURE (非セキュア)
ppuFixedAttr_AllDisable.userPermission	Fixed PPU の対象 PC に対するユーザアクセス属性を設定	CY_PROT_PERM_DISABLED (アクセス禁止)
ppuFixedAttr_AllDisable.privPermission	Fixed PPU の対象 PC に対する特権アクセス属性を設定	CY_PROT_PERM_DISABLED (アクセス禁止)
ppuFixedAttr_AllDisable.secure	Fixed PPU の対象 PC に対する非セキュアアクセス属性を設定	TP_SECURE (非セキュア)

表 17 関数一覧

機能	説明	値
`Cy_Prot_ConfigPpuFixedSlaveStruct(*base, setPC, config)`	Fixed PPU レジスタを設定 *base; Fixed PPU のベースアドレス setPC: 設定する PC config: アクセス属性	*base: PERI_MS_PPU_FX_GPIO_PRT0_PRT setPC: TP_PERMITTED_PC または TP_PROHIBITED_PC config: ppuFixedAttr_AllEnable または &ppuFixedAttr_AllDisable

Code Listing 16 に、Fixed PPU の設定例を示します。

5 保護ユニットの設定例

Code Listing 16: Fixed PPU の設定例

```
#define PERI_MS_PPU_FX_GPIO_PRT0_PRT            ((volatile stc_PERI_MS_PPU_FX_t*) &PERI_MS-
>PPU_FX[228])

typedef enum
    /*  Selection of Master CPU ID  */
{
    CPUSS_MS_ID_CM0                 =  0ul,
    CPUSS_MS_ID_CRYPTO              =  1ul,
    CPUSS_MS_ID_DW0                 =  2ul,
    CPUSS_MS_ID_DW1                 =  3ul,
    CPUSS_MS_ID_DMAC                =  4ul,
    CPUSS_MS_ID_SLOW0               =  5ul,
    CPUSS_MS_ID_SLOW1               =  6ul,
    CPUSS_MS_ID_CM4                 = 14ul,
    CPUSS_MS_ID_TC                  = 15ul
} en_prot_master_t;

    /*  Define Master CPU ID to CM0+  */
#define MASTER_ID_OF_THIS_CPU CPUSS_MS_ID_CM0


typedef enum
    /*  Define protection context number  */
{
    CY_PROT_PC0  = 0ul,  /**< PC = 0 */
    CY_PROT_PC1  = 1ul,  /**< PC = 1 */
    CY_PROT_PC2  = 2ul,  /**< PC = 2 */
    CY_PROT_PC3  = 3ul,  /**< PC = 3 */
    CY_PROT_PC4  = 4ul,  /**< PC = 4 */
    CY_PROT_PC5  = 5ul,  /**< PC = 5 */
    CY_PROT_PC6  = 6ul,  /**< PC = 6 */
    CY_PROT_PC7  = 7ul,  /**< PC = 7 */
    CY_PROT_PC_NUM
} cy_en_prot_pc_t;

#define TP_PRIVILEGED         (1ul)             /* privileged */
#define TP_SECURE             (0ul)             /* non secure */
#define TP_PERMITTED_PC       (CY_PROT_PC6)     /* context 6 */
#define TP_PROHIBITED_PC      (CY_PROT_PC5)     /* context 5 */
#define TP_PERMITTED_PC_MASK  (1ul << (TP_PERMITTED_PC-1ul))
#define TP_PROHIBITED_PC_MASK (1ul << (TP_PROHIBITED_PC-1ul))

typedef enum
    /*  Selection of Fixed PPU structure attribute  */
{
    CY_PROT_PERM_DISABLED = 0x00ul, /**< Read, Write and Execute disabled */
    CY_PROT_PERM_R        = 0x01ul, /**< Read enabled */
    CY_PROT_PERM_W        = 0x02ul, /**< Write enabled */
    CY_PROT_PERM_RW       = 0x03ul, /**< Read and Write enabled */
    CY_PROT_PERM_X        = 0x04ul, /**< Execute enabled */
    CY_PROT_PERM_RX       = 0x05ul, /**< Read and Execute enabled */
```

```
    CY_PROT_PERM_WX       = 0x06ul, /**< Write and Execute enabled */
    CY_PROT_PERM_RWX      = 0x07ul  /**< Read, Write and Execute enabled */
}cy_en_prot_perm_t;

    /*  Configure Fixed PPU for full access  */
const cy_stc_ppu_gr_cfg_t  ppuFixedAttr_AllEnable =
{
    .userPermission = CY_PROT_PERM_RWX,
    .privPermission = CY_PROT_PERM_RWX,
    .secure         = TP_SECURE,
};

    /*  Configure Fixed PPU for no access  */
const cy_stc_ppu_gr_cfg_t  ppuFixedAttr_AllDisable =
{
    .userPermission = CY_PROT_PERM_DISABLED,
    .privPermission = CY_PROT_PERM_DISABLED,
    .secure         = TP_SECURE,
};

int main(void)
{
    SystemInit();

    cy_en_prot_status_t status;

    __enable_irq();

    Cy_SysEnableApplCore(CY_CORTEX_M4_APPL_ADDR);

    /*********************************************/
    /* 1. Setting for GPIO 0 Register attribute */
    /*********************************************/
    /*  Configure Fixed PPU for protection context = 5. See See Code Listing 17.  */
    /* 1_1. Set permissions so that master whose PC is 5 can not access GPIO 0 */
    status = Cy_Prot_ConfigPpuFixedSlaveStruct(PERI_MS_PPU_FX_GPIO_PRT0_PRT, TP_PROHIBITED_PC,
&ppuFixedAttr_AllDisable);
    CY_ASSERT(status == CY_PROT_SUCCESS);

    /*  Configure Fixed PPU for protection context = 6. See Code Listing 17.  */
    /* 1_2. Set permissions so that master whose PC is 6 can access GPIO 0 */
    status = Cy_Prot_ConfigPpuFixedSlaveStruct(PERI_MS_PPU_FX_GPIO_PRT0_PRT, TP_PERMITTED_PC,
&ppuFixedAttr_AllEnable);
    CY_ASSERT(status == CY_PROT_SUCCESS);

    /*********************************************/
    /* 2. Setting for MPUx (MPU for this Master) */
    /*********************************************/
    /* 2_1. Setting for MSx_CTL to allow the PC value to become "TP_PERMITTED_PC" or
"TP_PROHIBITED_PC" */
    /*  Set PROT_SMPU_MS0_CTL.PC_MASK for protection context = 5 and 6. For setting SMPU
details, see Configuration Example of SMPU.  */
    status = Cy_Prot_ConfigBusMaster(MASTER_ID_TO_BE_CHECKED, TP_PRIVILEGED, TP_SECURE,
```

5 保護ユニットの設定例

```
    TP_PERMITTED_PC_MASK|TP_PROHIBITED_PC_MASK);
    CY_ASSERT(status == CY_PROT_SUCCESS);

    for(;;);
}
```

5 保護ユニットの設定例

Code Listing 17: Cy_Prot_ConfigPpuFixedSlaveStruct() 関数

```
cy_en_prot_status_t Cy_Prot_ConfigPpuFixedSlaveStruct(volatile stc_PERI_MS_PPU_FX_t* base,
cy_en_prot_pc_t setPC, const cy_stc_ppu_gr_cfg_t* config)
{
    cy_en_prot_status_t status = CY_PROT_SUCCESS;
    un_PERI_MS_PPU_PR_SL_ATT0_t tempSL_ATT0 = { 0 };
    un_PERI_MS_PPU_PR_SL_ATT1_t tempSL_ATT1 = { 0 };
    un_PERI_MS_PPU_PR_SL_ATT2_t tempSL_ATT2 = { 0 };
    un_PERI_MS_PPU_PR_SL_ATT3_t tempSL_ATT3 = { 0 };

:

    switch(setPC)
    {
    :
    /* Set Fixed PPU region attribute for protection context = 1 */
    case CY_PROT_PC1:
        /* (1) Set Fixed PPU region attribute for protection context = 1 */
        tempSL_ATT0.u32Register = base->unSL_ATT0.u32Register;
        tempSL_ATT0.stcField.u1PC1_UR = (config->userPermission & CY_PROT_PERM_R);
        tempSL_ATT0.stcField.u1PC1_UW = (config->userPermission & CY_PROT_PERM_W) >> 1;
        tempSL_ATT0.stcField.u1PC1_PR = (config->privPermission & CY_PROT_PERM_R);
        tempSL_ATT0.stcField.u1PC1_PW = (config->privPermission & CY_PROT_PERM_W) >> 1;
        tempSL_ATT0.stcField.u1PC1_NS = !(config->secure);
        base->unSL_ATT0.u32Register = tempSL_ATT0.u32Register;
        status = (base->unSL_ATT0.u32Register != tempSL_ATT0.u32Register) ? CY_PROT_FAILURE :
CY_PROT_SUCCESS;    /* (2) Read back */
        break;

    /* Set Fixed PPU region attribute for protection context = 2 */
    case CY_PROT_PC2:
        tempSL_ATT0.u32Register = base->unSL_ATT0.u32Register;
        tempSL_ATT0.stcField.u1PC2_UR = (config->userPermission & CY_PROT_PERM_R);
        tempSL_ATT0.stcField.u1PC2_UW = (config->userPermission & CY_PROT_PERM_W) >> 1;
        tempSL_ATT0.stcField.u1PC2_PR = (config->privPermission & CY_PROT_PERM_R);
        tempSL_ATT0.stcField.u1PC2_PW = (config->privPermission & CY_PROT_PERM_W) >> 1;
        tempSL_ATT0.stcField.u1PC2_NS = !(config->secure);
        base->unSL_ATT0.u32Register = tempSL_ATT0.u32Register;
        status = (base->unSL_ATT0.u32Register != tempSL_ATT0.u32Register) ? CY_PROT_FAILURE :
CY_PROT_SUCCESS;    /* Read back */
        break;

    /* Set Fixed PPU region attribute for protection context = 3 */
    case CY_PROT_PC3:
        tempSL_ATT0.u32Register = base->unSL_ATT0.u32Register;
        tempSL_ATT0.stcField.u1PC3_UR = (config->userPermission & CY_PROT_PERM_R);
        tempSL_ATT0.stcField.u1PC3_UW = (config->userPermission & CY_PROT_PERM_W) >> 1;
        tempSL_ATT0.stcField.u1PC3_PR = (config->privPermission & CY_PROT_PERM_R);
        tempSL_ATT0.stcField.u1PC3_PW = (config->privPermission & CY_PROT_PERM_W) >> 1;
        tempSL_ATT0.stcField.u1PC3_NS = !(config->secure);
        base->unSL_ATT0.u32Register = tempSL_ATT0.u32Register;
        status = (base->unSL_ATT0.u32Register != tempSL_ATT0.u32Register) ? CY_PROT_FAILURE :
```

```
CY_PROT_SUCCESS;    /* Read back */
        break;

    /* Set Fixed PPU region attribute for protection context = 4 */
    case CY_PROT_PC4:
        tempSL_ATT1.u32Register = base->unSL_ATT1.u32Register;
        tempSL_ATT1.stcField.u1PC4_UR = (config->userPermission & CY_PROT_PERM_R);
        tempSL_ATT1.stcField.u1PC4_UW = (config->userPermission & CY_PROT_PERM_W) >> 1;
        tempSL_ATT1.stcField.u1PC4_PR = (config->privPermission & CY_PROT_PERM_R);
        tempSL_ATT1.stcField.u1PC4_PW = (config->privPermission & CY_PROT_PERM_W) >> 1;
        tempSL_ATT1.stcField.u1PC4_NS = !(config->secure);
        base->unSL_ATT1.u32Register = tempSL_ATT1.u32Register;
        status = (base->unSL_ATT1.u32Register != tempSL_ATT1.u32Register) ? CY_PROT_FAILURE :
CY_PROT_SUCCESS;    /* Read back */
        break;

    /* Set Fixed PPU region attribute for protection context = 5 */
    case CY_PROT_PC5:
        tempSL_ATT1.u32Register = base->unSL_ATT1.u32Register;
        tempSL_ATT1.stcField.u1PC5_UR = (config->userPermission & CY_PROT_PERM_R);
        tempSL_ATT1.stcField.u1PC5_UW = (config->userPermission & CY_PROT_PERM_W) >> 1;
        tempSL_ATT1.stcField.u1PC5_PR = (config->privPermission & CY_PROT_PERM_R);
        tempSL_ATT1.stcField.u1PC5_PW = (config->privPermission & CY_PROT_PERM_W) >> 1;
        tempSL_ATT1.stcField.u1PC5_NS = !(config->secure);
        base->unSL_ATT1.u32Register = tempSL_ATT1.u32Register;
        status = (base->unSL_ATT1.u32Register != tempSL_ATT1.u32Register) ? CY_PROT_FAILURE :
CY_PROT_SUCCESS;    /* Read back */
        break;

    /* Set Fixed PPU region attribute for protection context = 6 */
    case CY_PROT_PC6:
        tempSL_ATT1.u32Register = base->unSL_ATT1.u32Register;
        tempSL_ATT1.stcField.u1PC6_UR = (config->userPermission & CY_PROT_PERM_R);
        tempSL_ATT1.stcField.u1PC6_UW = (config->userPermission & CY_PROT_PERM_W) >> 1;
        tempSL_ATT1.stcField.u1PC6_PR = (config->privPermission & CY_PROT_PERM_R);
        tempSL_ATT1.stcField.u1PC6_PW = (config->privPermission & CY_PROT_PERM_W) >> 1;
        tempSL_ATT1.stcField.u1PC6_NS = !(config->secure);
        base->unSL_ATT1.u32Register = tempSL_ATT1.u32Register;
        status = (base->unSL_ATT1.u32Register != tempSL_ATT1.u32Register) ? CY_PROT_FAILURE :
CY_PROT_SUCCESS;    /* Read back */
        break;

    /* Set Fixed PPU region attribute for protection context = 7 */
    case CY_PROT_PC7:
        tempSL_ATT1.u32Register = base->unSL_ATT1.u32Register;
        tempSL_ATT1.stcField.u1PC7_UR = (config->userPermission & CY_PROT_PERM_R);
        tempSL_ATT1.stcField.u1PC7_UW = (config->userPermission & CY_PROT_PERM_W) >> 1;
        tempSL_ATT1.stcField.u1PC7_PR = (config->privPermission & CY_PROT_PERM_R);
        tempSL_ATT1.stcField.u1PC7_PW = (config->privPermission & CY_PROT_PERM_W) >> 1;
        tempSL_ATT1.stcField.u1PC7_NS = !(config->secure);
        base->unSL_ATT1.u32Register = tempSL_ATT1.u32Register;
        status = (base->unSL_ATT1.u32Register != tempSL_ATT1.u32Register) ? CY_PROT_FAILURE :
CY_PROT_SUCCESS;    /* Read back */
```

```
            break;

        default:
            return CY_PROT_BAD_PARAM;
        }

        return status;
    }
```

6 用語集

用語	説明
CRYPTO	Cryptography component (暗号コンポーネント) 。詳細は architecture reference manual の Cryptography block 章を参照してください。
DMB	Data memory barrier。この命令は Thumb 命令セットです。
DSB	Data synchronization barrier。この命令は Thumb 命令セットです。
ISB	Instruction synchronization barrier。この命令は Thumb 命令セットです。
M-DMA	Memory DMA。詳細は architecture reference manual の Direct memory access 章を参照してください。
MPU	Memory protection unit (メモリ保護ユニット)
PC	Protection context (保護コンテキスト)。詳細は architecture reference manual の Prptection unit 章の“Protection context”を参照してください。
P-DMA	Peripheral DMA。詳細は architecture reference manual の Direct memory access 章を参照してください。
PPU	Peripheral protection unit (周辺機能保護ユニット)
SMPU	Shared memory protection unit (共有メモリ保護ユニット)

7 関連資料

以下は TRAVEO™ T2G ファミリのデータシートおよびテクニカルリファレンスマニュアルです。これらの資料を入手する場合はテクニカルサポートに連絡してください。

[1] デバイスデータシート

- CYT2B6 datasheet 32-bit Arm® Cortex®-M4F microcontroller TRAVEO™ T2G family
- CYT2B7 datasheet 32-bit Arm® Cortex®-M4F microcontroller TRAVEO™ T2G family
- CYT2B9 datasheet 32-bit Arm® Cortex®-M4F microcontroller TRAVEO™ T2G family
- CYT2BL datasheet 32-bitArm® Cortex®-M4F microcontroller TRAVEO™™ T2G family
- CYT3BB/4BB datasheet 32-bit Arm® Cortex®-M7 microcontroller TRAVEO™ T2G family
- CYT4BF datasheet 32-bit Arm® Cortex®-M7 microcontroller TRAVEO™ T2G family
- CYT6BJ datasheet 32-bit Arm® Cortex®-M7 microcontroller TRAVEO™ T2G family (Doc No. 002-33466)
- CYT3DL datasheet 32-bit Arm® Cortex®-M7 microcontroller TRAVEO™ T2G family
- CYT4DN datasheet 32-bit Arm® Cortex®-M7 microcontroller TRAVEO™ T2G family
- 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

[2] ボディコントローラ Entry ファミリ

- TRAVEO™ T2G automotive body controller entry family architecture reference manual
- TRAVEO™ T2G automotive body controller entry registers reference manual for CYT2B7
- TRAVEO™ T2G automotive body controller entry registers reference manual for CYT2B9
- TRAVEO™ T2G automotive body controller entry registers reference manual for CYT2BL (Doc No. 002-29852)

[3] ボディコントローラ High ファミリ

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

[4] Cluster 2D ファミリ

- TRAVEO™ T2G automotive cluster 2D architecture reference manual
- TRAVEO™ T2G automotive cluster 2D registers reference manual for CYT3DL
- TRAVEO™ T2G automotive cluster 2D registers reference manual for CYT4DN
- TRAVEO™ T2G automotive cluster 2D registers reference manual for CYT4EN (Doc No. 002-35181)

[5] Cluster Entry ファミリ

- TRAVEO™ T2G automotive cluster entry family architecture reference manual
- TRAVEO™ T2G automotive cluster entry registers reference manual for CYT2CL

[6] アプリケーションノート

- AN224432 - TRAVEO™ T2G のマルチコアハンドリングガイド

8 その他の参考資料

各種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、スタートアップを含むサンプルドライバライブラリ (SDL) を提供します。SDL は、公式の AUTOSAR 製品でカバーされていないドライバについて、お客様への参照としても機能します。SDL は自動車用規格に適合していないため、量産用としては使用できません。このアプリケーションノートに記載されているプログラムコードは、SDL に含まれるものです。SDL を入手する場合は、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
**	2018-06-05	これは英語版 002-19843 Rev. **を翻訳した日本語版 002-23843 Rev. **です。
*A	2018-12-05	これは英語版 002-19843 Rev. *A を翻訳した日本語版 002-23843 Rev. *A です。
*B	2019-03-06	これは英語版 002-19843 Rev. *B を翻訳した日本語版 002-23843 Rev. *B です。
*C	2019-08-20	これは英語版 002-19843 Rev. *C を翻訳した日本語版 002-23843 Rev. *C です。
*D	2020-04-15	これは英語版 002-19842 Rev. *D を翻訳した日本語版 002-23842 Rev. *D です。英語版の改訂内容: Changed target parts number (CYT2/ CYT4 series). Added target parts number (CYT3 series).
英語版*E	-	本版は英語版のみの発行です。英語版の改訂内容: Moved to Infineon Template Updated code examples using SDL
*E	2021-06-21	これは英語版 002-19843 Rev. *F を翻訳した日本語版 002-23843 Rev. *E です。英語版の改訂内容: Added note for 3.1 Protection Properties of Bus Transfer Added note for 3.4 Protection Context Attribute Setting. Added target parts number (CYT3DL series)
*F	2022-02-16	これは英語版 002-19843 Rev. *G を翻訳した日本語版 002-23843 Rev. *F です。英語版の改訂内容: Fixed of errors description. Change function name from GetUserMode/GetPrivilegedMode/ SVC_GetPrivilegedMode to SetUserMode/SetPrivilegedMode/ SVC_SetPrivilegedMode in 表 2 and Code Listing 1 to Code Listing 5.
*G	2022-10-27	これは英語版 002-19843 Rev. *H を翻訳した日本語版 002-23843 Rev. *G です。英語版の改訂内容: Changed description in はじめに. Added description in バス転送の保護プロパティ. Added Note in PC_MATCH 動作.
*H	2024-03-27	これは英語版 002-19843 Rev. *I を翻訳した日本語版 002-23843 Rev. *H です。英語版の改訂内容: Template update; no content update.
*I	2025-05-08	これは英語版 002-19843 Rev. *J を翻訳した日本語版 002-23843 Rev. *I です。英語版の改訂内容: Updated and fixed related to CYT6BJ

Trademarks

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

Edition 2025-05-08

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-mpm1681719678863

重要事項

本手引書に記載された情報は、本製品の使用に関する手引きとして提供されるものであり、いかなる場合も、本製品における特定の機能性能や品質について保証するものではありません。本製品の使用の前に、当該手引書の受領者は実際の使用環境の下であらゆる本製品の機能及びその他本手引書に記された一切の技術的情報について確認する義務が有ります。インフィニオンテクノロジーズはここに当該手引書内で記される情報につき、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。

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。