Vol.2022-OS-154 No.6
2022/3/15

Cortex-M 用 TrustZone を用いた IoT 機器向けタスク隔離実行基盤の設計

松下 意悟[1,a)]　内匠 真也[2]　藤松 由里恵[2]　金井 遵[2]　毛利 公一[1]

概要：IoT 機器で利用される CPU の 1 つである Arm® Cortex®-M シリーズが搭載する Trusted Execution Environment 技術である TrustZone® をベースに，この機器上のタスクが持つデータを保護し実行する，タスク隔離実行基盤の構築を進めている． 本論文では，この隔離実行基盤の設計と実装，機能評価を行ったので報告する． IoT 機器は，小型化や省電力性が求められることから，ハードウェアの制約がある．一方で，スマートウォッチに代表されるようなウェアラブルデバイスでは，個人の生体情報を含むセンシティブなデータを扱い，動的にタスクを追加し実行することといったことも求められる．そこで，TrustZone を用い，Secure World へタスクや関数を動的にロードし，実行を可能にするプラットフォームを提案し，これらの要求を達成するシステムの構築を可能にする． 提案システムでは，Non-Secure World を実行主体とし，Secure World 自体を Non-Secure World のタスクの 1 つとしてみなして扱うよう設計している．すなわち，Secure World は，Non-Secure World でスケジューリングされた時にのみ動作する．Secure World 内で動作するタスクや関数のスケジューリングは，Secure World 内に配置されるランタイムにより行われる．これにより，Non-Secure World 側の OS は 1 つのタスクを操作するだけで Secure World を扱い，タスクの隔離実行を実現することができる．

1. はじめに

通信技術や半導体技術の向上により，さまざま機器がインターネットに接続するようになり，IoT(Internet of Things) 機器として急速に広まっている．IoT 機器には，スマートウォッチに代表されるような個人の活動データを収集し続けるようなウェアラブル機器も含まれ，この機器が扱うデータはセンシティブなものになっている．そして，IoT 機器を対象としてマルウェアも増加しており，2018 年には，VPNFilter[1] が猛威を振るった．VPNFilter は，ルータや NAS といった機器を対象としたマルウェアであり，その活動の中には感染した機器内部にある情報の収集や，機器のファームウェアの書き換えが挙げられる．このような事例から IoT 機器のセキュリティを確保することが重視され，日本では，総務省と国立研究開発法人情報通信研究機構が ISP と連携し，IoT 機器のセキュリティの調査 NOTICE (National Operation Towards IoT Clean Environment) を 2019 年 2 月 20 日から開始している [2].

[1] 立命館大学
Ritsumeikan University
[2] (株) 東芝研究開発センター
Corporate Research & Development Center, Toshiba Corporation
[a)] imatsushita@asl.cs.ritsumei.ac.jp

IoT 機器は小型化や省電力性を確保するために非力な CPU や少量のメモリ，MMU を搭載しないなどハードウェアの制約がある．さらに，ウェアラブル機器といった個人の生体情報を含むセンシティブなデータを扱うものが多く登場している．こうした特徴を持つ IoT 機器では，以下に示す要求がある．

- ユーザの利便性向上のためのタスクの動的ロード
- 短いユーザ応答速度の実現
- データの機密性・完全性を確保したタスク実行

ウェアラブル機器では，機器を使うユーザが利便性向上のためにさまざまなタスクをインストールし実行する．そのため，従来の組込み機器の要求に加え，タスクの動的なロードを可能にしたいという要求がある．また，ユーザが快適に機器を利用するためには，ユーザ応答速度が短くすることが求められる．これらの要求を満たした上で，動的にロードしたタスクがセンシティブなデータを処理するといったことも考えられることから，データの機密性や完全性を確保しつつタスクを実行する必要がある．

本論文では，このような要求を持つシステムを構築するために， IoT 機器で利用される CPU の 1 つである Arm Cortex-M シリーズが搭載する Trusted Execution Environment (以下，TEE) 技術である TrustZone に着目し，センシティブなデータを安全に処理できるプラットフォー

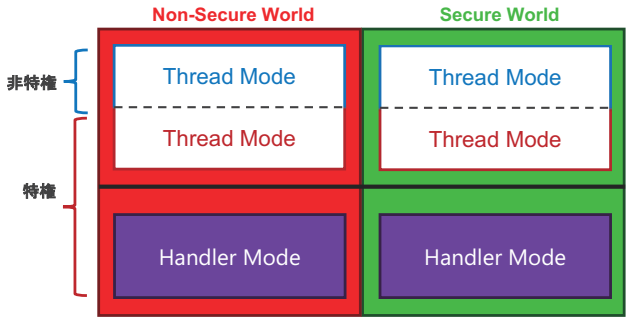

図 1 Secure World と Non-Secure World

ムの実現を目標とする．このプラットフォームは，OS の上で実行するタスクや関数を Secure World に動的にロードし，それらが扱うレジスタやメモリ領域を Non-Secure World のタスク・OS から秘匿された状態で実行するものである． TrustZone で想定されている関数の隔離実行だけでなく，タスクの隔離実行も可能にすることにより，Secure World で実行するソフトウェア構成の自由度の向上が見込まれる．タスクや関数を隔離実行することで，それらが扱う処理中のデータの傍受や改ざんなどの不正なアクセスを防ぎ，機器上のセンシティブなデータの機密性や完全性を確保する．また，Secure World のタスクや関数の実行時には，Non-Secure World の割込みを許可することで，短いユーザ応答時間を実現する．

以下，本論文では，2 章で Arm Cortex-M のセキュリティ拡張機能である TrustZone について述べ，3 章で提案システムについて述べる．4 章でその実装について述べた後に，5 章で機能評価について述べる，6 章で関連研究について述べ，7 章でまとめる．

2. Cortex-M 用 TrustZone

2.1 実行モードと遷移

Arm Cortex-M シリーズでは，Armv8-M アーキテクチャを採用するものからセキュリティ拡張機能として CMSE (Cortex-M Security Extensions) を搭載し，これは Arm Cortex-M 用 TrustZone とも呼ばれる．TrustZone は，**図 1** に示すように，通常の実行空間である Non-Secure World の他に信頼される実行空間である Secure World を生成する．各ワールドには，通常の動作を行う Thread Mode と例外処理を行う Handler Mode が存在し，メモリや例外，外部割込みもどちらかのワールドに属するようになる．一方，CPU は分割できないため，Security State と呼ばれる状態を新たに持つ．Security State は，Secure State と Non-Secure State の 2 つの状態を持つ．Secure State で CPU を実行している状態を「Secure World で動作する」と呼び，Non-Secure State で CPU を実行している状態を「Non-Secure World で動作する」と呼ぶ．Security State を変更することを「ワールドを遷移する」と呼ぶ．

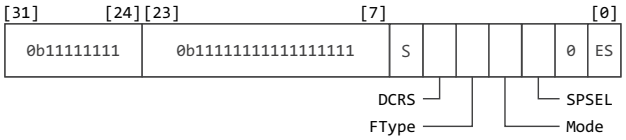

図 2 EXC_RETURN のビットフィールド

Non-Secure World から Secure World へ遷移するには以下に示す 2 つの方法がある．

SG 命令を利用する方法

SG 命令を実行することにより，Security State を Non-Secure State から Secure State に変更できる．この命令は，後述するリンク時にアドレスが決定される NSC 領域で実行されるもののみが有効なものとなるため，これを用いた Secure World への遷移は Secure World のプログラム開発者によって決められた方法でのみ行われる．

BX EXC_RETURN 命令を用いる方法

BX 命令は，分岐命令の 1 つであり，オペランドに EXC_RETURN と呼ばれる値を指定することで，Non-Secure World から Secure World へ遷移する場合がある．BX EXC_RETURN は, Handler Mode から直前の実行モードへの制御の移行を行うものであり，TrustZone が有効である時，Secure World 実行中に Non-Secure World 例外が発生することにより Non-Secure World の例外ハンドラに制御が移る状況がある．この Non-Secure World の例外ハンドラを抜けて Secure World に戻る際のワールド遷移では，BX EXC_RETURN 命令が行われる．EXC_RETURN は，**図 2** に示すようなビットフィールドを持ち，例外が発生し例外ハンドラに制御が移るまでの間にその値が決定され，CPU によって LR レジスタへ自動的に格納される．例えば，Non-Secure World から Secure World に遷移する時の EXC_RETURN の値は，6 ビット目が 1 であるものになる．

2.2 Security State で分離されるメモリ領域

TrustZone では，Security State によってアクセスできる領域を設定することにより各ワールドがアクセスできるメモリ領域を分離している．この設定は SAU(Secure Attribution Unit) と IDAU(Implementation Defined Attribution Unit) で管理され，メモリ領域に対して以下の 3 つの属性を付与することができる．

- Secure 属性
- NSC (Non-Secure Callable) 属性
- Non-Secure 属性

Secure 属性が与えられた領域は，Secure State の場合のみアクセス可能でありこの領域を Secure Memory と呼ぶ．Non-Secure 属性が与えられた領域は，Secure State と Non-Secure State の両方でアクセス可能であり，この領域を Non-Secure Memory と呼ぶ．NSC 属性が与えられた領

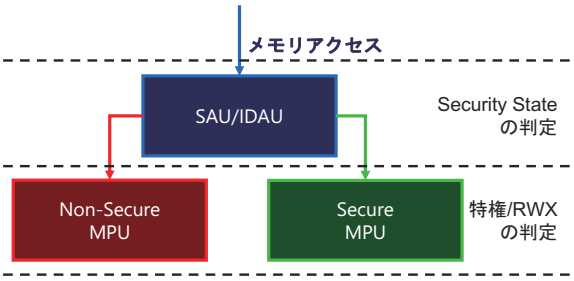

図 3　メモリアクセス時の SAU/IDAU と MPU の関係

域は，Secure Memory に属しながらも Non-Secure State からアクセス可能な領域であり，この領域で SG 命令を実行することにより，Non-Secure World から Secure World に遷移する．

SAU と IDAU の違いは属性の付与を動的に行えるかどうかである．IDAU は，メモリ領域に対してセキュア属性を静的に割り当てるもので，実行中にソフトウェアによってその設定を変更できないが，SAU はソフトウェアによってメモリ領域の属性を Secure State 時に変更することが可能である．メモリアクセスする場合の MPU との関係は，**図 3** に示すように MPU よりも前段に SAU/IDAU が存在し，Security State によるアクセスが判定されたのちに，MPU によるアクセスの判定を受ける．

2.3　ベクタテーブル

TrustZone によりワールドが 2 つに分かれるため，それぞれのワールド用に各例外ハンドラのアドレスを格納するベクタテーブルが定義可能である．Secure World のベクタテーブルには，Non-Secure World のものと比較してオフセット 0x1C の場所に SecureFault が追加される．SecureFault は，Non-Secure World から不正に Secure World の資源にアクセスするときに発生するため，Secure World はこれのための例外ハンドラが必要となる．また，ベクタテーブルのベースアドレスを示すレジスタである VTOR は，それぞれのワールド用にバンク分けされ，Secure World の VTOR は VTOR_S，Non-Secure World の VTOR は VTOR_NS と表される．

2.4　例外優先度

TrustZone 有効時には Non-Secure World の例外優先度の上限を制限でき，この例外優先度の上限を利用することにより Secure World で Non-Secure World の例外すべてをマスクできる．具体的には，AIRCR レジスタの PRIS ビットを 1 にセットすると，**図 4** に示すように Non-Secure World の例外が設定できる優先度は 0x80 から 0xFE の間にマップされる．つまり，Non-Secure World から見える優先度は 0x0 が最も高い優先度になるが，Secure World から見たときは 0x80 として扱われる．一方で Secure World は，0x0 から 0xFE までの優先度を設定できる．従って，

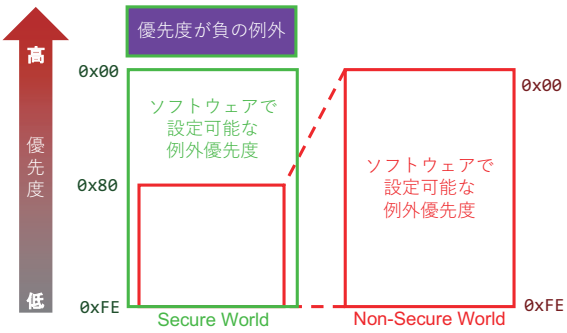

図 4　TrustZone 有効時の例外優先度

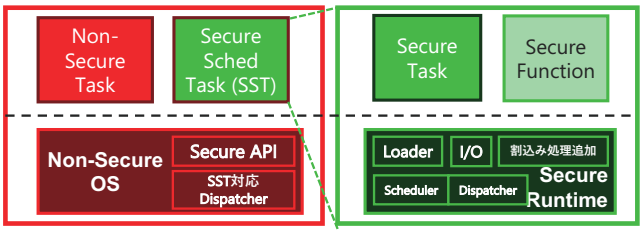

図 5　提案システムのソフトウェア構成

0x80 以下の優先度の例外をマスクすることで，Non-Secure World のすべての例外をマスクできる．

3.　データ保護のための隔離実行基盤

3.1　システム概要

提案システムは，タスクと関数を Secure World に動的にロードし，Non-Secure World からそれらが扱うレジスタやメモリ領域を秘匿された状態で実行するものである．Non-Secure World をこのシステムの実行主体とし，Secure World は Non-Secure World から 1 つのタスクとみなして扱われる．Secure World にロードされたタスクと関数は，Secure World 側のランタイムによってスケジューリングされ，実行される．提案システムのソフトウェア構成を**図 5** に示し，以下でそれぞれ述べる．

3.2　Secure World のプログラミングモデル

Secure World へ動的にロードし実行されるユーザプログラムであるタスクと関数は，それぞれ Secure Task と Secure Function と呼ぶ．

Secure Task

Secure Task は，Secure World で継続的に実行されるプログラムであり，Secure World で動作するランタイムによってスケジューリングされ，他の Secure Task や Secure Function と切り替えられながら実行される．

Secure Function

Secure Function は，後述する Non-Secure Task の一部として呼び出される Secure World で動作するプログラムであり，Secure Function が呼び出された時に処理を行い，それが終了すると呼出し元の Non-Secure Task に戻る．

3.3 Secure World のランタイム

Secure World には, Secure Task と Secure Funciton を管理するランタイムである Secure Runtime がある．Secure Runtime は Secure World の特権処理を行うプログラム群であり，以下に示すプログラムで構成される．

Secure Scheduler

Secure Scheduler は，Secure World で動作する Secure Task と Secure Function をスケジューリング対象とするスケジューラである．

Secure Loader

Secure Loader は，Secure Task と Secure Funtion を Secure World にロードするソフトウェアである．Secure Loader は，Non-Secure World の指示を受けてこれらをロードする．

Secure IO

Secure IO は，Secure World で閉じて IO 処理を行う機能である．Secure World で扱われるデータは，完全性と機密性を確保するために Non-Secure World を介してやり取りされてはならないため，Secure Runtime でこの機能を提供する必要がある．

特定の例外ハンドラへの処理の追加機能

Secure World で，例外，外部割込みを利用した Secure Task や Secure Function を実現するための例外処理を追加するための機能である．

3.4 Non-Secure World のプログラミングモデル

Non-Secure World には, Non-Secure Task と Non-Secure Function の 2 種類のユーザプログラムがある．

Non-Secure Task

Non-Secure World 側で継続的に実行されるプログラムであり，Non-Secure World の OS によってスケジューリングされ，他の Non-Secure Task や Non-Secure Function と切り替えられながら実行される．

Non-Secure Function

Non-Secure Function は，Secure Task から呼び出され，これの一部として呼び出された時に実行される Non-Secure World の関数のことを指す．Secure Task の一部として実行されるため，実行後は呼び出した Secure Task に戻る．

3.5 Non-Secure World の OS と Secure Sched Task

Non-Secure World 側の OS を Non-Secure OS と呼び，これは従来の OS の振る舞いに加えて，Secure World のスケジューリングや Secure Task，Secure Function のロードのために以下の機能が追加される．

Secure Sched Task

Secure Sched Task は，Non-Secure World 側に配置される Secure World そのものの実行状態を表す TCB (Task Control Block) であり，Non-Secure OS はこれを扱うことで，Secure World の実行状態を管理する．

Main Scheduler

Main Scheduler は, Non-Secure Task, Non-Secure Function，Secure Sched Task をスケジューリングする．Secure Sched Task を次に実行するタスクとする場合，タスクへのコンテキストスイッチではなく，Secure World への遷移を行う．

Secure API

Secure API は，Non-Secure Task が Secure Task と Secure Function を Secure World に動的ロードするためにインターフェースである．Secure API は，Secure Loader と通信することで，Non-Secure Task からロードを依頼された Secure Task と Secure Function を Secure World にロードする．

3.6 Secure Task の実行

ある Secure Task(ST_n) を実行するタイミングは，以下に示す条件が揃う時である．

- Main Scheduler が Secure Sched Task を実行する
- Secure World 内部で例外により Secure Scheduler に制御が移る
- ST_n がスケジューラに次に実行されるタスクとして選択され，ディスパッチされる

従って，Main Scheduler は，Secure Task を実行するかどうかを決めることはできない．そのため，Secure Task の実行時のコンテキストは，Secure World に閉じており，Non-Secure World 側のいかなるソフトウェアも特定の Secure Task の実行の妨害やコンテキストの改ざん及び奪取はできない．

3.7 Secure Function の実行

Secure Function の呼出しでは，Non-Secure World から Secure World に遷移する．Secure Sched Task は Secure World の実行状態を反映するため，Secure Sched Task を実行中状態に変更しつつ関数の実行しなければならない．そのため，**図 6** と以下に示す処理が必要がある．

(1) 呼出し元 Task のサスペンド
(2) Secure Sched への Secure Function の実行要求
(3) Main Scheduler への Secure Sched Task の実行要求

この Secure Function の呼出しから Non-Secure World の呼出し元タスクに戻る際には，以下に示す処理を行う必要がある．この処理を図にしたものを**図 7** に示す．

(1) 呼出し元のリジューム
(2) Secure Sched Task の次に呼出し元 (返る先) のタスクを実行するように Main Scherduler に依頼する

Secure Function の返り値は，Arm の呼出し規約 [3] により R0 に格納される．Secure Function の呼出し元タスクがこの返り値を受け取るためには，Non-Secure Task のリ

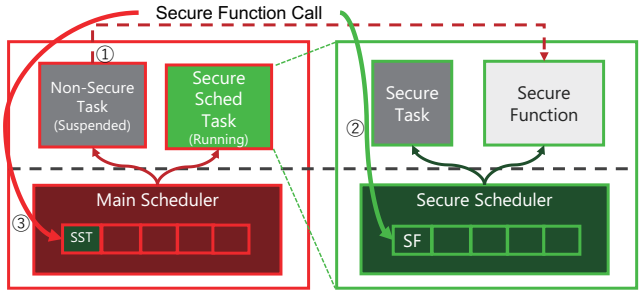

図 6　Secure Function Call のスケジューリング

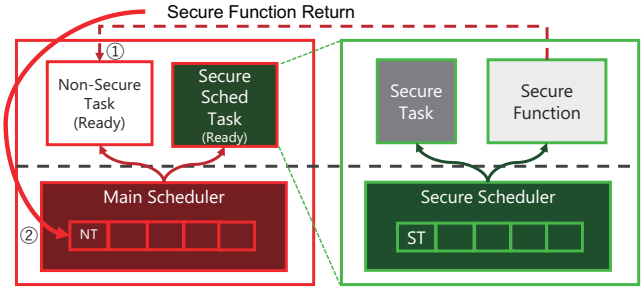

図 7　Secure Function Return のスケジューリング

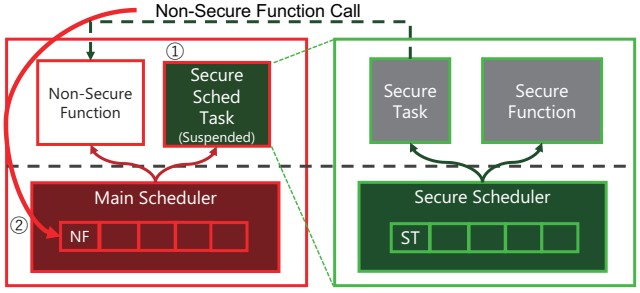

図 8　Non-Secure Function Call のスケジューリング

ジューム処理と Main Scheduler への呼出し元タスクの実行を依頼する必要がある．

3.8　Non-Secure Function の実行

Non-Secure Function の呼出しでは，Secure World から Non-Secure World に遷移するため，Secure Sched Task をサスペンド状態に変更しつつ関数を実行する．Non-Secure Function の呼出しで必要となる処理を**図 8** と以下に示す．

(1) Secure Sched Task のサスペンド
(2) Main Scheduler への Non-Secure Function の実行要求

Secure Function Call とは逆に Secure World から Non-Secure World への遷移し，想定外の Secure World 側の実行によるコンテキストの破壊を防ぐため，Secure Sched Task は実行状態をサスペンド状態にする．その後，Non-Secure Function を実行することで，Secure のコンテキストと Non-Secure のコンテキストを一致させて動作させる．

Non-Secure Function から Secure Task に返る際には，**図 9** と以下に示す処理が必要となる．

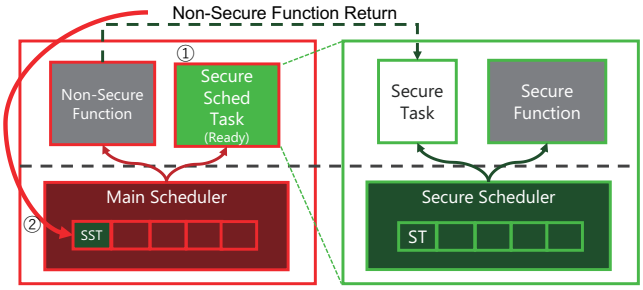

図 9　Non-Secure Function Return のスケジューリング

(1) Secure Sched Task をリジュームする
(2) Main Scheduler に Secure Sched Task を実行するように依頼する

この場合は，Secure World に帰還するため Secure Sched Task を実行中状態にする必要があり，また，Secure Function と同様に，Non-Secure Function からの返り値は呼出し規約により R0 に格納される．この返り値を呼出し元である Secure Task が受け取るために，Secure Sched Task のリジューム処理とこれを次に実行するタスクとするよう Main Scheduler に依頼し，Secure Task が実行されるようにする．

3.9　ワールド間の割込みネストの防止

Non-Secure World と Secure World の両方で割込みが発生する場合があり，この際に片方のワールドで割込み処理を行なっている間にもう片方のワールドで割込みが発生し，またその割込み処理中に片方のワールドの割込みが発生するといった状態になることが考えられる．この状態が続くと，システムの応答時間が想定以上に長くなるといった問題が起きる．この問題は，一方のワールドの割込みハンドラ実行中は他方の割込みを禁止することにより，割込み処理中に割込み処理をさせないことで解決できる．

提案システムでは，Secure World の割込み優先度を Non-Secure World よりも優位にし，Non-Secure World の割込みを Secure World の Handler Mode でマスクする方針をとる．この理由として，Non-Secure World で Secure World の例外をマスクする場合，SecureFault が Non-Secure World からマスクされてしまうことが問題であるからである．Non-Secure World から Secure World の資源に対して不正なアクセスを行うと，Secure HardFault または SecureFault が発生する．どちらが発生するかは，SHCSR (System Handler Control and Status Register) によって決めることができ，SecureFault が発生する場合，不正アクセスにより SecureFault ハンドラへ制御が渡る．この SecureFault ハンドラは，Secure World 側に存在し，優先度の設定によっては Non-Secure World でこの例外をマスクすることができる．Non-Secure World で Secure 側の割込みをマスクする場合，SecureFault が発生する事態となっても Secure 側

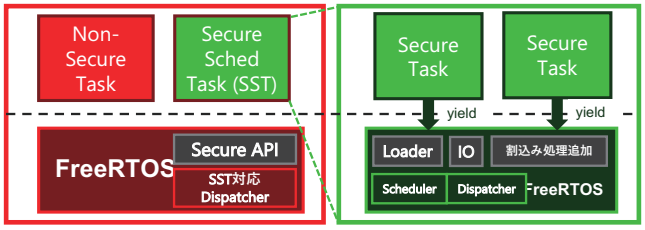

図 10 実装した提案システムのソフトウェア構成

に制御が渡らず，Secure World 側で不正なアクセスに対して対処できない．また，Handler Mode でのみ割込みを禁止することにより，Secure Task や Secure Function の実行モードである Thread Mode 中に Non-Secure World の例外が発生しても遅延なく割込み処理ができ，ユーザ応答速度が向上する．

以上のことから，提案システムでは，Secure World の Handler Mode では，Non-Secure World の割込みをマスクする．

4. 実装

本章では，1 章で述べたウェアラブルデバイスのような IoT 機器が求める要件のうち，データの機密性・完全性を確保したタスク実行を実現するために必要な実装について述べる．また，Secure Task，Secure Function の動的ロードに向けて必要となる位置独立性を持ったバイナリの生成についても述べる．

4.1 ハードウェア構成

我々は，提案システムを NXP 社の開発ボードである LPC55S69-EVK を対象に実装した．LPC55S69-EVK は，Cortex-M33 コアを 2 つ搭載しそのうち 1 つのコアが TrustZone に対応している．また，実行するプログラムの作成は同じく NXP 社が提供する MCUXpressoIDE を利用した．

4.2 ソフトウェア構成

提案システムの実装において，**図 10** に示すように Non-Secure World と Secure Runtime に FreeRTOS™V10.3.0 を採用した．また，提案システムで動作させるタスクは，自身が Yield することにより制御を Secure Runtime に返すことで，他のタスクへの切り替えが行われる．

4.3 起動からタスクが動作する状態までの実行の流れ

提案システムでは，**図 11** に示すように，CPU が起動されると Secure World の Reset 例外のハンドラである Reset ハンドラが実行される．そして，提案システムのための Secure World の初期化処理を行なったのちに Non-Secure World へ制御を渡し Main Scheduler を含む FreeRTOS が起動する．FreeRTOS は起動後，タスクを切り替えるために PendSV 例外を発行し，コンテキストスイッチを行う．

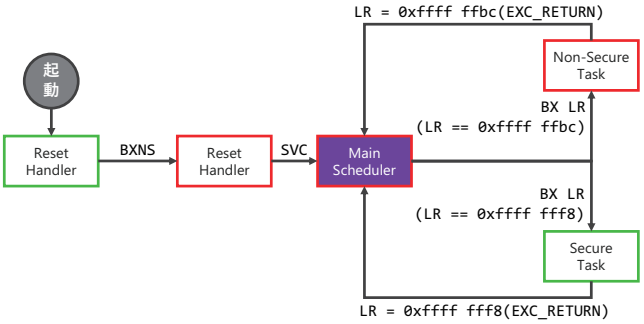

図 11 起動から定常状態への遷移

PendSV 例外が発行される時点に実行されているタスクが Non-Secure Task の場合，LR に 0xffff ffbc の値を取る EXC_RETURN が CPU により自動的に格納され，Secure Task の場合は，LR が 0xffff fff8 の値を取る EXC_RETURN となる．PendSV ハンドラ内では，LR の値をタスクが持つスタックに退避しタスクを再開する際に，このスタックから LR に格納されていた EXC_RETURN の値を取り出し，BX LR 命令によってタスクを再開する．EXC_RETURN の値によって Secure State を含む直前のコンテキストを復元するため，Secure Task であっても Non-Secure Task と同様にタスクを再開できる．

4.4 Secure Sched Task への初回タスク切り替え

提案システムでは，Secure Sched Task がディスパッチされた時に，前に実行していた Secure World の状態に EXC_RETURN を用いて遷移する．しかし，この EXC_RETURN の値は Secure State 中に Non-Secure World 側の例外・外部割込みが起きた時にのみ CPU が自動的に LR に格納するため，起動時に Secure World から Non-Secure World へ遷移する際には，Non-Secure World に LR を通して EXC_RETURN の値は渡されない．また，LR の値と同様に Secure World のコンテキストも Secure State 中の Non-Secure 例外・外部割込みによって Secure Memory に自動的に退避され，それを前提に EXC_RETURN の値に対する BX 命令実行時にコンテキストを復元する．図 11 で示した Secure World 起動時はまだ Non-Secure World が起動しておらず，Non-Secure World への遷移に Non-Secure World の例外・外部割込みは利用できない．この問題を解消するために，遷移時点の Secure World に帰るための EXC_RETURN と Secure Memory に退避される Secure World のコンテキストを用意し，Non-Secure World に遷移する．これにより，Secure Sched Task の初回実行時もそれ以降の再開時も同じ BX LR 命令を用いて Secure World へ遷移できる．

4.5 Secure Task の位置独立性

提案システムでは，Secure Task は実行時に動的に Non-

Fault Status Registers

Name	Value	Description
IPSR	0x00000007	Exception Status Register (Secure Fault)
CFSR	0x00000000	Configurable fault Status Register
DFSR	0x00000000	Debug fault Status Register
AFSR	0x00000000	Auxiliary fault Status Register
SFSR	0x00000048	Secure fault Status Register
SFAR	0x30020000	Secure fault Address Register

図 12 Non-Secure World から Secure Task のメモリを読み取るプログラム実行時に発生した例外に関連するレジスタの値

Secure World から Secure World に挿入され, Secure World で実行する．これは, Secure Task は任意のアドレスの RAM に配置されることを意味し, Secure Task のコードはどのアドレスに配置されても実行可能でなくてはならない．これを実現するために, 位置独立性を持つ Position Independent Executalble (以下，PIE) 形式のバイナリである必要があり，GCC ではコンパイル時に”-fPIE”オプションを付与することで，PIE 形式のバイナリを生成することができる．

5. 機能評価

本章では，4 章で実装した提案システムの機能評価について述べる．機能評価では，Secure Task が保持する情報が Non-Secure World に流出しない実装となっていることを示す．

5.1 Secure Task のメモリの保護

評価方法

Secure Task は Non-Secure Task から隔離して実行するため，Non-Secure Task は Secure Task が利用するメモリ領域にアクセスすることは禁じられる．これを示すために，Non-Secure World 側の最高権限状態である Handler Mode で Secure Task のメモリを読み取るプログラムを作成した．このプログラムを実行することにより，Handler Mode で Secure Task のコードが配置される 0x30020000 番地に読み取りアクセスを行う．期待される結果は，Secure Memory に対する不正アクセスから SecureFault が発生し，SecureFault の原因アドレスを示す SFAR (SecureFault Addrss Register) に 0x30020000 が格納されることである．

評価結果

Non-Secure World 側の最高権限状態である Handler Mode で Secure Task のメモリを読み取るプログラムを実行すると，SecureFault 例外が発生し例外関連のレジスタに**図 12** に示す値がそれぞれ格納された．図 12 の IPSR (Interrupt Program Status Register) は，発生した例外をが何であるかを示すレジスタであり，0x7 が格納されたことから例外番号 7 である SecureFault が発生したことがわかる．また，SFAR が SecureFault が発生した原因のアドレスとして 0x30020000 が格納されており，期待通りの動作をしたことがわかる．以上のことから，Secure Task が扱うメモリ領域は，Non-Secure Task から隔離されることが示された．

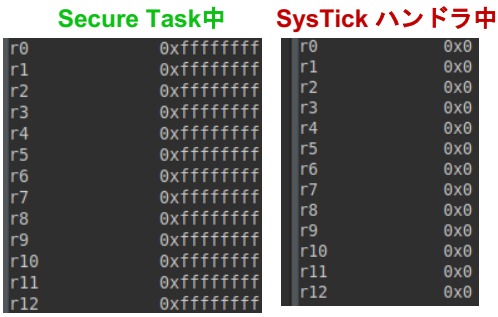

図 13 Secure Task 中のレジスタ値

5.2 例外発生時の Secure Task のレジスタ情報保護

評価方法

Secure Task 実行時に Non-Secure World から例外・外部割込みが発生した場合，Secure Task の実行が中断され Non-Secure World 側の例外ハンドラが実行される．この時に，Secure Task が利用している汎用レジスタの値が Non-Secure World 側の例外ハンドラで取得できないことが望まれる．これを示すために，実行されると汎用レジスタ全てを 0xffffffff の値で埋め，その後はビジーループで待機する Secure Task を作成した．Non-Secure World では，定期的に FreeRTOS によって設定されたタイマ割込みである SysTick 例外が発生するため，Secure Task のビジーループ中に Non-Secure World 側の SysTick 例外が発生し，この Secure Task から Non-Secure World の SysTick ハンドラに制御が移る．SysTick ハンドラに制御が移った直後に GDB を用いて汎用レジスタ群の値を確認し，Secure Task が設定した値である 0xffffffff と異なる場合は，Secure Task のレジスタ情報が Non-Secure World に流出しないことが示される．

評価結果

Secure Task のビジーループのレジスタ値と Non-Secure World の SysTick ハンドラ実行直後のレジスタ値を**図 13** に示す．これを見ると，Secure Task 実行時には汎用レジスタである R0-R12 までの値が全て 0xffffffff となっているが，図 13 を見るとこれらの値は全て 0 となっている．従って，Secure Task が利用する汎用レジスタの値は，例外・外部割込みの発生によって Non-Secure World に遷移したとしても Non-Secure World から取得できないことが示された．

6. 関連研究

組込み機器を対象として，アプリケーションを隔離実行する研究が存在する．本章では，TEE を利用するものとそうでないものに分けてそれぞれについて述べる．

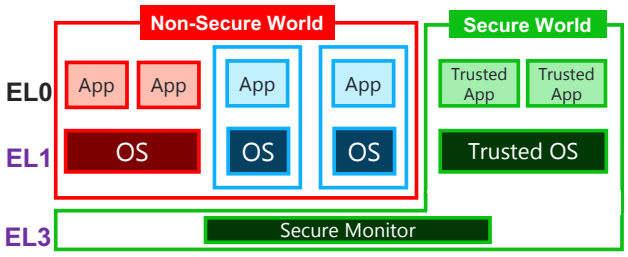

図 14 TrustICE や SANCTUARY が取る隔離実行空間の生成

6.1 TEE を利用しない隔離実行

TrustLite[4] および TyTAN[5] は，Execution-aware Memory Protection Unit(EA-MPU) と呼ぶハードウェアを FPGA 上に構築し，アプリケーションの隔離を行う．EA-MPU は実行中の命令がどのアプリケーションかであるか判別し，実行中のアプリケーションが保有する RAM や Flash 領域のみアクセス可能にすることで，アプリケーション間の分離を行う．

6.2 TEE を利用する隔離実行

TrustICE[6] と SANCTUARY[7] では，ARM Cortex-A シリーズを対象として，**図 14** に示すように複数の隔離実行空間を Non-Secure World 側に生成する．TrustICE では，Non-Secure World 中の隔離実行空間を ICE とよび，ある時間に実行される ICE を 1 つに制限し，実行されない ICE が利用するメモリ領域を Secure Memory にすることにより，ICE 同士の隔離を実現している．しかし，この手法ではマルチコアの場合でも実行されるタスクが 1 つに制限されるという欠点がある．一方，SANCTUARY では隔離実行空間 1 つに対して 1 つの CPU コアを割り当て，TZASC (TrustZone Address Space Controller) を利用し，CPU コアごとにアクセスできるメモリ領域を制限することで，複数の隔離実行空間を同時に動作させることを可能としつつ TrustICE のようなタスク間の分離を可能としている．

Arm Cortex-M シリーズ向けの実装は，uTango[8] があり，これは TrustICE と同様の手法を取る．uTango では，Non-Secure 中の隔離空間をそれぞれ NSVW (Non-Secure Virtual World) と呼び，ある NSVW を実行中には，他の NSVM のペリフェラル領域を含むメモリに対して SAU を用いて Secure 属性とすることで NSVM 同士の隔離を行っている．

これらの手法では，隔離実行空間を切り替える際にワールド管理を行う Secure World で動作するソフトウェアに遷移した上で SAU などを用いて隔離実行空間ごとにアクセスできるメモリ領域の設定を行う点でオーバヘッドが発生すると考えられる．

7. おわりに

本論文では，IoT 機器上で実行されるタスクが扱うデータを保護するための隔離実行基盤を提案し，実装と機能評価を行なった．提案システムは，Secure World で動的にロードした関数やタスクを実行することで，ウェアラブル機器などのユーザが後から追加する機能に対しても隔離実行をすることを実現する．タスクや関数を隔離環境で実行することにより，実行中のレジスタ値やメモリ上の値の不正な取得や改ざんを防ぎ，機器上のデータの完全性と機密性が確保できる．

現在の実装では，Secure Task が RAM 上に静的に展開された条件での実行ができるのみに留まっているため，今後は，Secure Loader や Secure World 側の IO 処理などを実装することが必要である．

参考文献

[1] Trend Micro: ネットワーク機器を狙う IoT ボット「VPNFilter」、世界で 50 万台以上に感染，Trend Micro（オンライン），入手先 ⟨https://blog.trendmicro.co.jp/archives/17484⟩ （参照 2022-01-25）．

[2] 総務省：IoT 機器調査及び利用者への注意喚起の取組「NOTICE」の実施，総務省（オンライン），入手先 ⟨https://www.soumu.go.jp/menu_news/s-news/01cyber01_02000001_00011.html⟩ （参照 2022-01-25）．

[3] Arm Limited: *Procedure Call Standard for the Arm Architecture* (2020).

[4] Koeberl, P., Schulz, S., Sadeghi, A.-R. and Varadharajan, V.: TrustLite: A Security Architecture for Tiny Embedded Devices, *Proceedings of the Ninth European Conference on Computer Systems*, EuroSys '14, New York, NY, USA, Association for Computing Machinery (2014).

[5] Brasser, F., El Mahjoub, B., Sadeghi, A.-R., Wachsmann, C. and Koeberl, P.: TyTAN: Tiny Trust Anchor for Tiny Devices, *Proceedings of the 52nd Annual Design Automation Conference*, DAC '15, New York, NY, USA, Association for Computing Machinery (2015).

[6] Sun, H., Sun, K., Wang, Y., Jing, J. and Wang, H.: TrustICE: Hardware-Assisted Isolated Computing Environments on Mobile Devices, *2015 45th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, pp. 367–378 (2015).

[7] Brasser, F., Gens, D., Jauernig, P., Sadeghi, A.-R. and Stapf, E.: SANCTUARY: ARMing TrustZone with User-space Enclaves, *Proceedings 2019 Network and Distributed System Security Symposium*, San Diego, CA, Internet Society (2019).

[8] Oliveira, D., Gomes, T. and Pinto, S.: uTango: an open-source TEE for the Internet of Things, *arXiv:2102.03625 [cs]* (2021).