Vol.2021-OS-151 No.6
2021/3/2

IoT 機器向けアプリケーション隔離実行基盤の検討

松下 意悟[1]　鄭 俊俊[1]　藤松 由里恵[2]　金井 遵[2]　鬼頭 利之[2]　毛利 公一[1]

概要：IoT 機器においてセキュアな実行環境を構築するために，アプリケーション間の強固な隔離を実現する実行プラットフォームの構築の検討を進めている．本論文では，IoT 機器で利用される CPU の 1 つである ARM Cortex-M シリーズが搭載している TEE(Trusted Execution Environment) 技術である TrustZone をベースとして実現することを検討したので報告する．IoT 機器は，非力な CPU と少ない量のメモリで構成され，この機器上で動作する OS は，Intel 64 アーキテクチャ CPU 上で動く Windows や Linux と比べてセキュリティ機能の実装が進んでおらず，アプリケーション間の分離が弱い．我々は，アプリケーション間の強固な隔離を可能にすることでこの問題を解決する． 本論文では，センサからデータ取得と署名を安全に行いつつサーバに送信するようなシステムを前提とし，それに必要な要素として Secure Task 実行機能を提案する．

1. はじめに

近年，PC やスマートフォン以外にも，あらゆるものをインターネットに接続させ，それぞれが相互に通信することにより新たなサービスを提供する，IoT(Internet of things) への取り組みが広まっている．2020 年には，日本でも低遅延，高信頼性，多数同時接続を特徴とする 5G 通信のサービスが開始され，これまで以上に急速な IoT 化が進むと予想される．また，IoT 機器は，高機能化，小型化しており，多くのセンシティブな情報を取り扱うようになってきている．

それに伴い，IoT 機器を対象とする攻撃も増えてきており，機器固有の脆弱性を突く攻撃も増えてきている [1]．2016 年には Mirai と呼ばれるマルウェアが IoT 機器を乗っ取り，Botnet を形成することにより大規模な DDoS 攻撃が発生した [2]．日本においても，IoT 機器のセキュリティを確保することが重視され，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 [3]．

IoT 機器は，非力な CPU と少ない量のメモリで構成されることから，この機器上で動作する OS のセキュリティ機能は，Intel 64 アーキテクチャ CPU 上で動く Windows®*1 や Linux®*2 と比べて限られたものとなっている．そのため，ソフトウェアの耐タンパ性は低く，IoT 機器が扱うセンシティブな情報が漏洩する可能性が高い．この問題を解決するために，我々は，IoT 機器で利用される CPU の 1 つである ARM®*3 Cortex®-M シリーズが搭載している TEE(Trusted Execution Environment) 技術である TrustZone®に着目し，センシティブ情報を安全に処理できることを目指す．Intel®*4 の TEE 技術である Intel SGX では，いくつかのセキュリティ機能 [4], [5], [6] が研究されているが，ARM Cortex-M シリーズの研究はあまり進んでいない．また，TrustZone では Intel SGX のように，隔離実行空間を動的に複数個生成できず，隔離実行可能な処理が限られる．そこで，我々は，ARM Cortex-M 向けの TrustZone を用いて，隔離実行空間を動的に複数生成するプラットフォームの実現を目指す．

以下，本論文では， 2 章で我々の提案するシステムの対象としている IoT 機器のシステムについて述べ，3 章で IoT 機器のセキュリティの現状について述べる．4 章では，ARM アーキテクチャについて述べ，5 章で提案システムについて述べた後，6 章で ARM Cortex-M を搭載したボードの TrustZone を利用したサンプルプログラムの動作について述べる．7 章で関連研究について述べ，8 章でまとめる．

[1] 立命館大学
Ritsumeikan University
[2] 株式会社 東芝
Toshiba Corporation

*1 Windows またはその他のマイクロソフト製品の名称及び製品名は米国 Microsoft Corporation の，米国及びその他の国における商標または商標登録です．
*2 Linux は米国およびその他の国における Linus Torvalds の登録商標です．
*3 ARM,Cortex,TrustZone は米国及びその他の国における ARM Ltd. の商標です．
*4 Intel,Intel SGX,Intel 64 アーキテクチャは米国及び Intel Corporation またはその子会社の商標及び登録商標です．

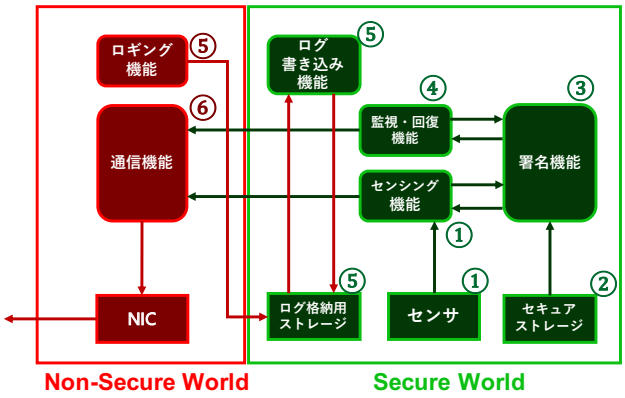

図 1 対象とする IoT 機器のシステム概観

2. 対象とする IoT 機器

本章では，提案システムの対象とする IoT 機器の構成について述べる．対象とする IoT 機器は，Armv8-M アーキテクチャを採用した CPU を搭載し，TrustZone が利用可能であるものを想定する．この機器は，センシングを行い，センサデータを外部のサーバに送信することを主な機能とした上で，サイバーレジリエンス機能を有し，Non-Secure World 側のプログラムの動作に問題が生じたり，DoS 攻撃を受けたとしても最低限の機能は継続して動作し，自律回復を行う．この IoT 機器のソフトウェアとハードウェアの関係を図 1 に示し，それぞれの機能について以下で述べる．

(1) センシング機能
この機能は，IoT 機器に接続されたセンサの値を定期的に読み取り，外部のサーバに取得した値を送信する機能である．外部のサーバに送信する際に，後述する署名機能を用いてデータに署名し，Non-Secure World 側の通信機能を通して送信される．センシング機能が利用するセンサも Secure World 内からのみアクセスできるものとすることで，Non-Secure World 内のプログラムが不正にセンシティブデータを取得することを防ぐ．

(2) 署名機能
この機能は，IoT 機器が外部に送信するデータなどに署名する機能である．データに署名することによりデータの改ざんの検知を可能にする．署名そのものが改ざんされないようにするため，Secure World 内で動作させる必要がある．

(3) セキュアストレージ
セキュアストレージは，Secure World でのみアクセスが可能なデータ格納領域である．適応例においては署名機能が利用し，署名に必要な秘密鍵の情報を格納する．Secure World でのみアクセス可能とすることで，秘匿して運用する必要のあるデータを保護することが可能となる．

(4) 監視・回復機能
この機能は，Non-Secure World 側のプログラムを監視し，可能であれば回復を行う機能である．Non-Secure World 側のプログラムを監視し，プログラムの実行に問題がある場合は，プログラムの再ロード等を行うことによりプログラムの回復を目指す．

(5) ログ書き込み機能
この機能は，改変されないログの保存を実現する．保存したログを Non-Secure World から改ざんされない様にするため，ログの保存領域であるログ格納用ストレージは Secure World 内に置く．Non-Secure World 内で発生するログを保存するロギング機能は，Secure World 内のログ格納用ストレージにアクセスすると例外が発生し，ログを書き込む事ができない．そこで，Secure World 側のログ書き込み機能がこの例外を捕捉し，ロギング機能が書き込む予定であったログを代わりに書き込むことにより，Non-Secure World からログを保存できるが，ログの変更や削除をさせないログの保存機能を実現する．

(6) 通信機能
この機能は，Non-Secure World 側に存在し，署名されたセンサデータや監視機能によって得られたアプリケーションのハッシュ値を外部のサーバに送信する．

3. IoT 機器のセキュリティの現状

本章では, IoT 機器のセキュリティについて, 現状の ARM Cortex-M シリーズ CPU 上で動作する OS のセキュリティ機能と Armv8-M アーキテクチャを採用した ARM Cortex-M シリーズが持つセキュリティ機能である TrustZone について述べる．

3.1 IoT 機器向け OS のセキュリティ機能

IoT 機器で利用される ARM Cortex-M シリーズ CPU 上で動作する OS が提供するセキュリティ機能としては，スタックオーバフローの検知が主であり，Intel 64 アーキテクチャ CPU 上で動作する Linux などの MMU を用いたアプリケーション間のメモリ領域の分離は行われていない．また，CPU の計算資源やメモリの量が潤沢に用意されていないことから，Linux 上で動作する ASLR や kASLR などのセキュリティ機能を移植することは難しい．

3.2 Trusted Execution Environment(TEE)

IoT 機器向けの OS が持つセキュリティ機能は従来から存在するものとなっているが，CPU のセキュリティ機能は新しいものがある．ARM Cortex-M シリーズでは，Armv8-M アーキテクチャにおいて Armv8-M Security Extension をサポートし，これにより TrustZone が実現され

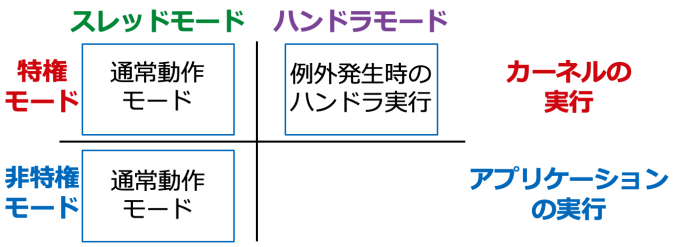

図 2 Cortex-M の動作モードと特権モード

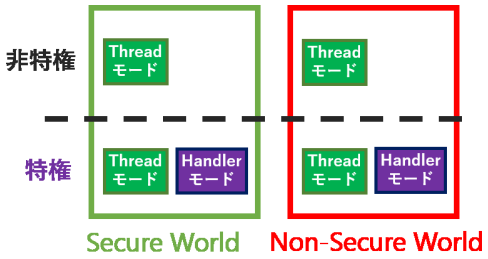

図 3 Cortex-M 向け TrustZone の 2 つの実行空間

る．TrustZone は TEE の実装の 1 つであり，これは，Intel SGX，RISC-V Keystone，ARM Cortex-A 向けの TrustZone と同様に，通常の実行空間とは別の信頼される隔離実行空間を提供する．それぞれ実装が異なる部分もあり，Intel SGX と RISC-V Keystone は，信頼される隔離実行空間を動的に複数生成できるのに対し，ARM Cortex-A および Cortex-M の TrustZone では，信頼される隔離実行空間は起動時に生成されるもの 1 つのみとなっている．また，Intel SGX では，他の TEE の実装と異なり隔離実行空間の動作権限は，最低の Ring 3 のみとなっている [7]．Cortex-A 向け TrustZone との違いは，モニタモードを介して環境の分離を行うかであり，Cortex-M 向け TrustZone では，ワールド遷移におけるオーバヘッドを削減するために，このモニタモードを提供せずにワールドの分離を行う．ARM Cortex-M 向け TrustZone では，通常の実行空間を Non-Secure World，信頼される実行空間を Secure World と呼ぶ．ワールドの遷移には，ワールド遷移用の命令を使用する必要があり，Secure World から Non-Secure World への遷移では，レジスタの値を自動的に Secure World 側に保存するため，Secure World が安全に隔離される．

4. ARM Cortex-M

本章では，ARM Cortex-M シリーズのアーキテクチャである ARMv8-M アーキテクチャにおける特権レベル，メモリ保護，TrustZone について述べる．

4.1 特権レベル

ARM Cortex-M シリーズの権限分離のモデルとして，Privilege Level を提供している．Cortex-M には，**図 2** に示すように 2 つの動作モードと 2 つの特権レベルが存在する．特権モードは，カーネルの実行に使用され，非特権モードではアプリケーションの実行に使用される．動作モードは，スレッドモードとハンドラモードが存在し，スレッドモードでは，通常動作を行い，ハンドラモードでは例外発生時のハンドラ実行時の動作を行う．例外発生時はカーネルが対処を行うため，ハンドラモードは特権モードにのみ存在する．

4.2 メモリ保護

ARM Cortex-M シリーズでは，メモリ保護を行うために，MPU(Memory Protection Unit) が提供される．MPU は，最大 16 個のメモリ領域に対して，特権レベルに応じた，読み取り専用，読み書き可能，アクセス不可，実行不可の属性を与えることができる．

4.3 TrustZone

TrustZone は，セキュリティ拡張機能である，Armv8-M Security Extension で提供される．

World と状態

TrustZone では，信頼される実行空間を Secure World と呼び，通常の実行空間を Non-Secure World と呼ぶ．**図 3** に示すように両方の World それぞれに特権モードと動作モードが存在する．また，Secure World で実行中の CPU 状態を Secure State，Non-Secure World で実行中の CPU 状態を Non-Secure State と呼ぶ．

メモリアドレスによる環境分離

TrustZone では，メモリの属性に基づき実行空間を分離している．メモリ領域を Secure World または Non-Secure World のどちらかのリソースであるかを設定し，これを基にアクセス制御を行うことにより環境が分離される．Secure World が保有するメモリ領域を Secure Memory と呼び，Non-Secure World が保有するメモリ領域を Non-Secure Memory と呼ぶ．メモリ領域をどちらの World のリソースとするかは，Secure Attribution Unit(SAU) と Implementation Defined Attribution Unit(IDAU) で管理される．SAU と IDAU はセキュアレベルが高い順に以下に示す 3 つのセキュア属性をメモリ領域に対して付与することが可能である．

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

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となり，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となる．NSC 属性が与えられた領域は，Secure Memory に属しながらも，Non-Secure State でもアクセスが可能であり，この領域を利用して，Secure World から Non-Secure

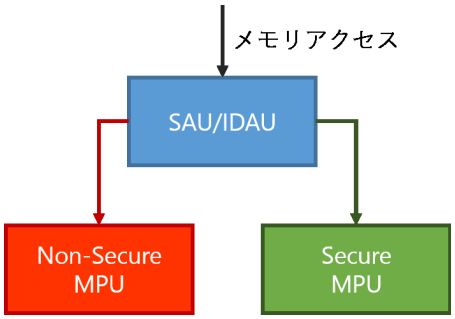

図 4 SAU/IDAU と MPU の関係

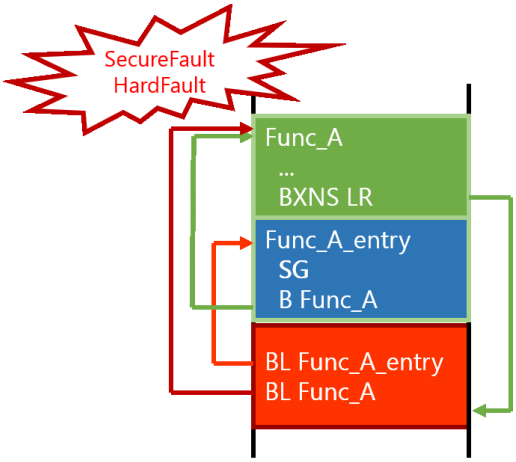

図 5 SG 命令によるワールド遷移

World に遷移する．

SAU と IDAU の違いは，設定を動的に行うか静的に行うかである．SAU は実行中にソフトウェアにより動的にメモリ領域のセキュア属性を変更することが可能である．SAU によって動的に定義できる領域数は，チップベンダによって決定され，SAU の設定を変更する場合は，Secure State でのみ可能である．一方 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することができない．しかし，IDAU でセキュア属性を設定したメモリ領域に対して SAU は上書きすることが可能である．この時，SAU はセキュア属性を完全に上書きするのではなく，どちらかで設定されたセキュア属性のうちセキュアレベルが高い方の属性が有効となる．メモリアクセス制御において MPU と SAU/IDAU の関係は，図 4 に示すように，SAU/IDAU によるアクセス制御の後に MPU のアクセス制御が行われる．

World 遷移

Non-Secure World から Secure World への遷移は，CPU の状態を Non-Secure State から Secure State に変更する SG 命令を伴って行われる．この時のコードとメモリの様子を図 5 に示し，以下で説明する．

図 5 の一番下の赤枠が Non-Secure 属性のメモリ領域で，現在実行中のコードとする．緑枠の部分が Secure 属性のメモリ領域で，青枠が NSC 属性のメモリ領域とする．Secure World の Func_A 関数を呼び出す時，ワールド遷移が必要となり，ワールド遷移には，NSC 属性のメモリ上で SG 命令を呼び出す必要がある．そのため，BL 命令を用いて，NSC 中の Func_A_entry 関数を呼び出し，SG 命令を実行する．この時，SG 命令以外を実行すると，Non-Secure State で Secure Memory 中の命令を実行することになるため，Secure Fault または Hard Fault が発生する．SG 命令後，B 命令で Func_A に分岐する事ができる．

反対に，Secure World から Non-Secure World へ遷移する場合について述べる．Secure State から Non-Secure State に変更する場合は次の 2 つの命令どちらかを利用する．

- BLXNS 命令（Non-Secure State に遷移するリンク付き分岐命令）
- BXNS 命令（Non-Secure State に遷移する分岐命令）

これらの命令は SG 命令とは違い，NSC 領域でなくとも実行が可能である．これら 2 つの命令どちらかが実行されると，自動的に遷移直前のレジスタの値を Secure Memory 内のスタック (Secure Stack) に保存し，レジスタをゼロクリアし，Non-Secure State になり分岐が行われる．BLXNS 命令はリンク付き分岐を行うため，返り値を格納するリンクレジスタ (LR) に FNC_RETURN と呼ばれる特別な値を格納する．Non-Secure State で LR が FNC_RETURN である場合，自動的に Secure State に切り替わり，Secure Stack に保存していたレジスタ値を復元する．

5. 提案システム

本章では，既存の TrustZone を用いたシステムの構成とその問題点について述べた後に，提案システムの概要と実装する必要のある機能が何か，3 章で述べた対象システムを基に検討を行った結果について述べる．

5.1 既存の TrustZone を用いたシステム構造

既存の TEE を用いたシステムは，主に以下の 2 つの構造に分けられる．

- Secure World と Non-Secure World で別の OS を動作させ，Non-Secure World から Secure World のアプリケーションのサービスを受ける構造
- Non-Secure World 側でのみ OS を動作させ，Non-Secure World 側のアプリケーションから Secure World 中の関数を呼び出す構造

前者のシステムは，Global Platform が定義している TEE の仕様 [8] に基づくもので，実装例として OP-TEE が挙げられる．このシステムでは，Non-Secure World のアプリケーションが利用する Secure World のサービスを Secure World の Trusted OS 上で動作するアプリケーションとして

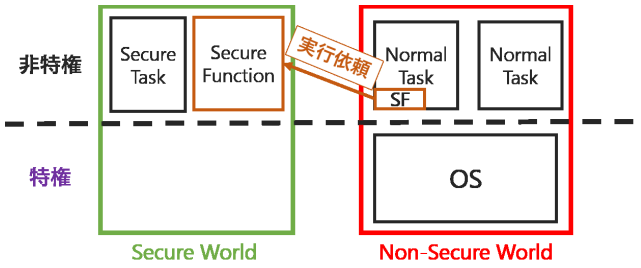

図 6 提案手法の概要

実装する必要がある．後者のシステムは，FreeRTOS™*5で実現可能なものであり，Non-Secure World 上のアプリケーションから Secure World 中の関数を呼ぶことをサポートしているが，FreeRTOS 上で動作するアプリケーション自体を Secure World で動作させるものではない．この両方のシステムは，Non-Secure World 側のプログラムと Secure World 側のプログラムを別々に作成する必要があるため，Secure World に新たな処理を追加する場合，Non-Secure World にもその処理を呼び出すための処理を記述する必要がある．

5.2 提案システム概要

提案システムは，Non-Secure World から Secure World で実行するアプリケーションを動的に作成する隔離実行基盤である．既存の Non-Secure World 内の OS とその上で動作するアプリケーションに加えて，Secure World 側で動作する非特権アプリケーションを扱えるようにし，IoT 機器上の様々な処理を Secure World で動作することを実現する．Secure World 側でアプリケーションを動作させることによりセンサデータや署名処理に利用される鍵などを，Non-Secure World 側のアプリケーションから不正に取得されることや改ざんされることを防ぐ．また，Secure World 側にアプリケーションを用意することを実行基盤が行うため，Secure World 側のアプリケーションとそのアプリケーションの呼び出しといった実装コストを削減することができる．

提案システムの概要を**図 6** に示す．Non-Secure World 側では，既存の OS とその上で動くアプリケーションが動作し，このアプリケーションを Normal Task と呼ぶ．一方，Secure World 側で OS から隔離して動作するアプリケーションを Secure Task と呼び，関数単位で Secure World で動作するものを Secure Function と呼ぶ．

5.3 実装する必要がある機能

2 章で述べた IoT 機器を実現するために，提案手法にはどのような機能が必要となるか検討を行った．検討を行った機能を Secure Task 実行前と実行中の 2 つのタイミングに分けて以下で述べる．

Secure Task 実行前に必要な機能

- Secure World 中に Task を生成する機能
  Secure Task を実装するためには，Non-Secure World 側の OS により生成される Task を Secure World 側に移動しなくてはならない．この方法として，Secure Task の実行コードが配置されているメモリ領域に対して，SAU により Secure 属性を付与する方法が考えられる．
- Secure Task が使用するペリフェラル領域のセキュア属性変更機能
  Secure Task が利用するセンサの値が Non-Secure World 側から取得される場合，センシティブデータの漏洩となる．そのため，Secure Task が利用するセンサがマップされているペリフェラル領域に対しては，Secure 属性を付与することにより，Non-Secure World からのアクセスを防ぐ．
- 例外ハンドラへの処理の追加機能
  Secure Task が継続的にセンシングを行う場合，タイマ割り込みをセンサの値の読み取りの契機としたり，**図 1** の (5) のログ保存機能においては，ロギング機能がログ格納用ストレージに対するアクセスで発生する例外を捕捉し処理を行うため，例外ハンドラの処理を追加する機能が必要となる．
- リモート認証機能
  Secure Task が開発者が想定した通りにメモリ上にロードされたか検証を行うために必要となる．

Secure Task 実行中に必要な機能

- タスク間通信
  図 1 では，Secure Task である署名機能がセンシング機能と監視・回復機能から得られるデータに対して署名を行うことや，センシング機能と監視・回復機能が Non-Secure World 側の通信機能にデータを送信するなど，Task 間通信を行う．このため，Secure Task においても Task 間通信機能を提供する必要がある．
- Secure Task のスケジューリング及びコンテキストスイッチ
  Secure Task は OS とは別の隔離空間で実行されるため，Secure Task を OS がスケジューリングし，適切に Normal Task とのコンテキストスイッチを行えるようにする必要がある．
- セキュアストレージ機能
  Non-Secure World 側から自由にアクセスされてはならない署名に利用する鍵やログを保存するために，セキュアストレージ機能が必要となる．

*5 FreeRTOS は Amazon Web Services,Inc. の商標です．

表 1　LPC55S69 のスペック

項目	説明
ボード名	LPC55S69-EVK
マイクロコントローラ	Arm Cortex-M33 x2
Flash	640KB
SRAM	320KB
メモリ拡張	MicroSD cart slot (4bit SDIO)
接続	USB 経由の SPI・UART ポート

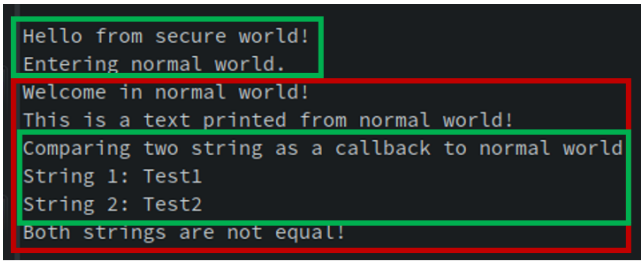

図 7　TrustZone を用いたプログラムの出力

6. TrustZone を用いたプログラムの調査

本章では，提案システムを実装する開発ボードにおける TrustZone の動作について述べる．開発ボードは，Arm Cortex-M33 を搭載する NXP 社の LPC55S69-EVK を利用する．この開発ボードのスペックを**表 1** に示す．

LPC55S69 は Arm Cortex-M33 を 2 つ搭載しており，そのうち片方にのみ MPU と FPU が搭載され TrustZone が利用可能である．開発環境は，NPX 社が提供する Eclipse®[*6]ベースの IDE である MCUXpressoIDE を利用する．以下では，TrustZone を用いたサンプルプロジェクトである”lpcxpresso55s69_hello_world_s”と”lpcxpresso55s69_hello_world_ns”の概要，メモリ配置，ワールド間のシンボル解決と遷移コード，ボードの初期化処理について述べる．

6.1 概要

この 2 つのプロジェクトは，両方を組み合わせることで動作する．”lpcxpresso55s69_hello_world_s”は，Secure World で実行されるプログラムで，”lpcxpresso55s69_hello_world_ns”は Non-Secure World で実行されるプログラムである．このプロジェクトを実行したときの IDE 内のプロンプトに**図 7** に示すものが出力される．

図 7 の緑枠の部分が Secure World 内の処理による出力で，赤枠の部分が Non-Secure World 内の処理による出力である．これらのプロジェクトでは，Secure World の処理が先に実行され最初の 2 行を出力した後，Secure World から Non-Secure World に遷移する．そして，Non-Secure World で次の 2 行を出力し，Secure World の関数である StringCompare_NSE 関数を呼び出し，2 つめの緑枠の出力が行われる．StringCompare_NSE 関数は，文字列比較を行うもので，比較する文字列は緑枠内で出力されている”Test1”と”Test2”である．また，この関数の返り値は文字列が同一かどうかを返すものとなっており，Non-Secure World 側に，文字列が不一致であることを返し，Non-Secure World 側で最終行の出力が行われる．

[*6] Eclipse は米国およびその他の国における Eclipse Foundation,Inc. の商標もしくは登録商標です．

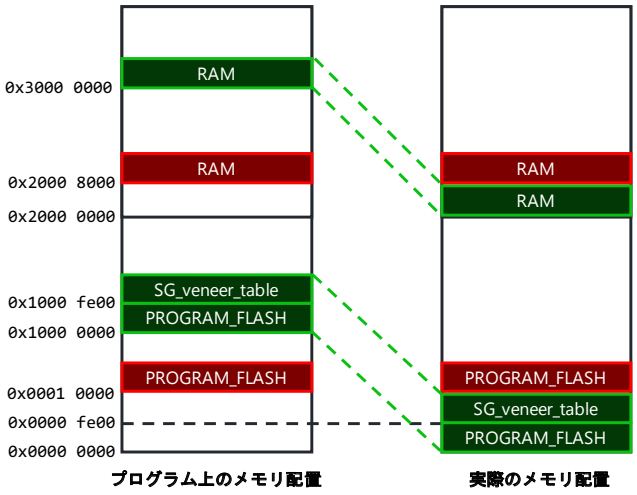

図 8　サンプルプロジェクトのメモリ配置

6.2 メモリ配置

両方のプロジェクトでは，それぞれ別のメモリ配置設定が行われており，この設定を図に表すと**図 8** の左側のような配置となる．図 8 において，緑枠の PROGRAM_FLASH が Secure World のコードが配置される場所で，RAM が Secure World で利用される RAM の領域であり，SG_veneer_table がワールド遷移で必要となる NSC 領域の配置位置となる．赤枠も同様に，PROGRAM_FLASH が Non-Secure World のコードが配置される場所で，RAM が Non-Secure World で利用される RAM の領域となる．しかし，Flash メモリ上では図 8 の右側に示す配置となる．これは，この開発ボードが搭載している MCU ファミリである LPC55S6x MCU ファミリの TrustZone の実装によって，メモリアドレスの 28bit 目が Secure World の領域かどうかを示すフラグとなっているためである．

また，両方のワールドの PROGRAM_FLASH の先頭は**図 9** に示すベクタテーブルとなっている．

6.3 ワールド間のシンボル解決

これらのサンプルプロジェクトでは，Secure World 側のプログラムと Non-Secure World 側のプログラムが別々にコンパイルされており，ワールドを跨いで関数を実行する際には，遷移する先の関数のアドレスを知っている必要がある．この問題を解決するために IDE 内蔵のリンカには，Secure World の関数のアドレスのシンボル解決

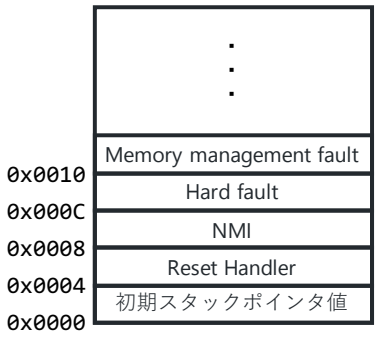

図 9 Cortex-M33 のベクタテーブル

```
7    1000fe00 <StringCompare_NSE>:
8    1000fe00:   e97f e97f    sg
9    1000fe04:   f7f0 bb90    b.w 10000528 <__acle_se_StringCompare_NSE>

439  10000528 <__acle_se_StringCompare_NSE>:
440  10000528:   b590         push    {r4, r7, lr}
441  1000052a:   b089         sub sp, #36 ; 0x24
442  1000052c:   af00         add r7, sp, #0
560  10000688:   4774         bxns    lr
```

SG命令で Secure Stateへ移行

bxns lr によるNon-Secureへの帰還

図 10 StringCompare_NSE 関数のディスアセンブル結果

を行う Secure Gateway import libraries 機能がある．この機能は，–cmse-implib オプションにより有効になり，–out-implib オプションに指定したオブジェクトファイルに cmse_nonsecure_entry 属性が付与された関数のシンボルテーブルを出力する．Non-Secure World 側のプログラムでは，これを基にシンボル解決が行われる．

6.4 ワールド遷移コード

Non-Secure World から Secure World の関数を実行するためには，Non-Secure State で NSC 領域中の SG 命令を実行した後に，Secure Memory 内の関数を実行するという処理が必要である．このワールド遷移に必要なコードは関数に cmse_nonsecure_entry 属性を付与することによりコンパイラによって生成される．”lpcxpresso55s69_hello_world_s” プロジェクト内で Non-Secure World から呼ばれる関数である StringCompare_NSE 関数のディスアセンブル結果を**図 10** に示す．

StringCompare_NSE ラベルでは，SG 命令と__acle_se_StringCompare_NSE ラベルへの分岐命令が実行され，__acle_se_StringCompare_NSE ラベル内には，StringCompare_NSE 関数の処理が書き出される．また，StringCompare_NSE ラベルは図 7 の NSC 領域である SG_veneer_table 領域に出力される．__acle_se_StringCompare_NSE ラベルの最後では，BXNS 命令が実行され Non-Secure World へ処理が戻る．

6.5 各ワールドの初期化処理

それぞれのプロジェクトの成果物であるバイナリファイルは，IDE 内蔵の Flash ツールによってボードの Flash

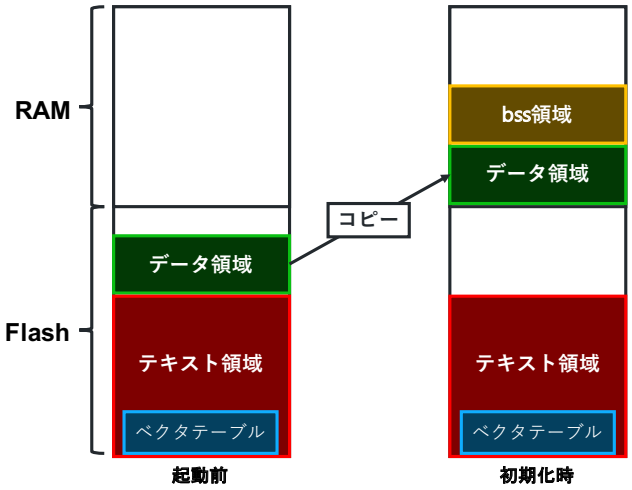

図 11 ボード起動前と初期化時のメモリレイアウト

メモリに書き込まれる．Flash ツールでは，Flash メモリ上のどのアドレスから書き込むかを指定することができ，デフォルトでは 0x0 番地から書き込まれる．”lpcxpresso55s69_hello_world_s” プロジェクトでは，書き込む際に 0x1000 0000 番地から書き込む設定がされているが，実際には，図 8 に示したように 0x0 番地に書き込まれる．また，”lpcxpresso55s69_hello_world_ns” プロジェクトのバイナリは，0x0001 000 番地から書き込まれる．この Flash メモリ上のプログラムを実行する流れを以下で述べる．

ARM Cortex-M では，起動時にベクタテーブルを中心に処理が実行される．図 9 に示すように，ベクタテーブルの先頭はスタックポインタの初期値が格納され，続いてリセットハンドラのアドレスが格納される．ボードの起動時には，ベクタテーブルのオフセット 0x0 番地の値をスタックポインタにセットし，オフセット 0x4 番地に格納されているアドレスにあるリセットハンドラを実行する．ベクタテーブルの先頭物理アドレスは，VTOR(Vector Table Offset Register) の値で決まり，デフォルトでは 0x0 番地となっており，最初に Secure World 側のプログラムが実行される．リセットハンドラでは，Flash メモリに書き込まれたバイナリのデータ領域の値を RAM にコピーし，bss 領域を 0 埋めする．この処理を**図 11** に示す．この処理が終われば，Secure World 側の main 関数が実行される．Secure World 側の main 関数では，Non-Secure World 側のベクタテーブルを基に Non-Secure World のスタックポインタの値と VTOR の値をセットし，Non-Secure World 側のリセットハンドラを実行し，Secure World と同様に Non-Secure World の初期化処理を行う．

7. 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して実行する手法はいくつか存在する．TrustLite[9] および TyTAN[10] は，EA-MPU(Execution-aware Memory Pro-

tection Unit) 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プログラムであるか判別し，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，プログラム間の分離を行う．TrustICE[11] や SANCTUARY[12] では，ARM Cortex-A シリーズを対象として，隔離実行空間を Non-Secure World 側に生成する．TrustICE では，ある時間に実行されるタスクを 1 つに制限することにより，Non-Secure World 中においてもタスクの隔離を実現している．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．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，TrustZone Address Space Controller(TZASC) 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が可能となる．ARM Cortex-M シリーズ向けの実装は，uTango[13] があり，これは TrustICE と同様の手法を取る．

8. おわりに

本論文では，ARM Cortex-M シリーズを搭載する IoT 機器向けの隔離実行基盤について述べた．IoT 機器向けの OS では，ハードウェアの制約からアプリケーション間の分離が不十分であり，この機器が取り扱うセンシティブなデータの安全な取扱いが難しい．この問題を解決するために，TrustZone を利用し，アプリケーション自体を Secure World で動作させる実行基盤により，安全なセンシティブデータの利用，署名処理の隠蔽，Non-Secure World のアプリケーションの監視の実現を可能にする．そのために，想定する利用例を実現するために提案システムに必要な機能の検討を行い，ARM Cortex-M シリーズの TrustZone について調査し，Cortex-M33 を搭載する開発ボードである LPC55s69-EVK における TrustZone を利用したプログラムの動作を明らかにした．今後は，対象とする IoT 機器が実現可能となる提案システムの機能の実装を行うとともに，Secure Function についても実装の検討を行う．

参考文献

[1] サイバーセキュリティ研究所サイバーセキュリティ研究室：NICTER 観測レポート 2018，国立行政法人情報通信研究機構（オンライン），入手先 ⟨https://www.nict.go.jp/cyber/report/NICTER_report_2018.pdf⟩（参照 2021-01-28）．

[2] 宮田健：IoT デバイスを狙うマルウェア「Mirai」とは何か——その正体と対策，TechFactory（オンライン），入手先 ⟨https://techfactory.itmedia.co.jp/tf/articles/1704/13/news010.html⟩（参照 2021-01-28）．

[3] 総務省，国立研究開発法人情報通信研究機構：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 2021-01-28）．

[4] Bauman, E. and Lin, Z.: A Case for Protecting Computer Games With SGX, *Proceedings of the 1st Workshop on System Software for Trusted Execution*, SysTEX '16, New York, NY, USA, Association for Computing Machinery (2016).

[5] Baumann, A., Peinado, M. and Hunt, G.: Shielding Applications from an Untrusted Cloud with Haven, *ACM Transactions on Computer Systems*, Vol. 33, No. 3, pp. 1–26 (2015).

[6] Lazard, T., Götzfried, J., Müller, T., Santinelli, G. and Lefebvre, V.: TEEshift: Protecting Code Confidentiality by Selectively Shifting Functions into TEEs, *Proceedings of the 3rd Workshop on System Software for Trusted Execution*, Association for Computing Machinery, New York, NY, USA, pp. 14–19 (2018).

[7] 須崎有康，塚本明，小島一元，中島健太，Thuc, H. T.，師尾彬：3 種類の TEE 比較（Intel SGX, ARM TrustZone, RISC-V Keystone），slideshare（オンライン），入手先 ⟨https://www.slideshare.net/suzaki/3teeintel-sgx-arm-trustzone-riscv-keystone⟩（参照 2021-01-28）．

[8] GlobalPlatform: Introduction to Trusted Execution Environments, GlobalPlatform (online), available from ⟨https://globalplatform.wpengine.com/resource-publication/introduction-to-trusted-execution-environments/⟩ (accessed 2021-01-28).

[9] 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, (online), DOI: 10.1145/2592798.2592824 (2014).

[10] 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, (online), DOI: 10.1145/2744769.2744922 (2015).

[11] 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).

[12] 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).

[13] Pinto, S. and Pinto, S.: May the Trust be with You: Empowering TrustZone-M with Multiple Trusted Environments, blackhat (online), available from ⟨https://www.blackhat.com/asia-20/briefings/schedule/index.html#may-the-trust-be-with-you-empowering-trustzone-m-with-multiple-trusted-environments-18294⟩ (accessed 2021-01-28).