ルネサスRAファミリ ビギナーズガイド

Richard Oed

2023.01

Richard Oed
日本語訳：阿部 直樹 Naoki Abe

ルネサスRAファミリビギナーズガイド

ルネサスRAファミリビギナーズガイドは、RAファミリの最初の一歩を踏み出すエンジニア向けの入門ガイドとして、またRAファミリに既に精通している人向けのリファレンスブックとして使用することを目的としています。
この本が参照しているFSPのバージョンはv.3.7.0であることに注意してください。

RAファミリのMCUは継続的に開発されているため、このPDFを公開する際に最新ではなくなったセクションが存在する可能性がある点についてご理解ください。 この本がより役立つように**www.renesas.com/ra-book**には オンラインのサポートシステムもあります。

免責事項

この本は情報提供の目的で提供されており、正確性と完全性を保証するものではありません。
内容については、デザインリファレンスガイドとして参照することを意図したものではなく、この本の使用から生じるいかなる結果に対しても責任を負いません。

目次

はじめに

ネットワークデバイスが普及した現代の世界では、セキュリティはありとあらゆる場面で直面する課題です。ネットワークに接続されたすべてのアプリケーションは攻撃の標的であり、ハッカーがネットワークにアクセスして、保護の対象となる価値のあるデータにアクセスする潜在的な危険性があります。

急速に成長しているIoT分野では、組み込み分野向けのセキュリティがさらに重要になります。なぜならば費用効果が高く、スケーラブルでエネルギー効率の高いセキュリティソリューションが必要であり、構成もシンプルにできるためです。現在、組み込みアプリケーションの開発者は利用可能な開発リソースや対応する開発時間が少なく、加えて十分な予算が組まれていないにもかかわらずセキュリティのトピックを真剣に受け止めざるを得なくなっています。

このため、時間というプレッシャーのもとで初めてセキュリティのトピックに対処することを余儀なくされるエンジニアの数は絶えず増加しています。この理由として挙げられる要因は、新しい法律、競争圧力、知的財産の保護などさまざまです。企業の多くにとってセキュリティは多面的で複雑なトピックであるため、セキュリティについて開発当初より十分な検討が困難な場合があります。一般的にセキュリティを実現するためにはハードウェアおよびソフトウェアコンポーネントを含みながら、全体的に検討する必要があります。つまり、セキュリティは開発プロセスの最後のみに追加することはできず、最初からシステムアプローチの一部である必要があります。多くの場合では、企業はこれを実現するための必要な能力を欠いており、さらにセキュリティの実装により製品に追加のコストが発生しています。実際、一部の企業はこの課題解決に非常に苦しんでいるのです。

ルネサスの開発部門は、お客様を支援することをビジネスにしています。焦点は、将来の組み込みシステムのセキュリティ要件をサポートし、費用効果が高く、スケーラブルで、エネルギー効率の高いセキュリティソリューションの開発にあります。

お客様に最高のサービスを提供するためのこの取り組みの成果が、ルネサスRAファミリです。 RAファミリは、ルネサスとそのパートナの完全なエコシステムを備えたArm Cortex-Mコアに基づくソフトウェアの柔軟性を備えた、業界をリードするIoTセキュリティを提供します。これにより、お客様はセキュリティを含むアプリケーションの目標をはるかに迅速に達成することができるのです。

セキュリティの実現のみならず、お客様の開発作業を細部までこだわってシンプルにすることが 32ビットRA MCUファミリを提供する我々の思いです。この本が、RAファミリを使ったプロジェクトの開発を楽しむのに役立つことを願っています。

Bernd Westhoff
Director – RA MCU Marketing
Renesas Electronics

1　ルネサスRAファミリの概要

2019年10 月に導入されたRA Family of microcontrollers (MCU) は、ルネサスの32 ビットMCU 範囲を拡張しました。これは以前にリリースされた、MCU と民生グレードの品質と保証されたソフトウェアおよび開発ツールを組み合わせたArm® Cortex®-M コアをベースとしたRenesas Synergy™プラットフォームと、業界トップクラスの単位周波数あたりのCoreMark® パフォーマンスと大容量コードフラッシュメモリ、SRAM を搭載した独自の32bitコアを特徴とするルネサスeXtreme (RX)ファミリを補完しています。これらは、お客様に独自の差別化と価値を提供します。新しいRAファミリは、低電力アプリケーション用のRA2シリーズ、低電力且つ高性能とセキュリティが必要なアプリケーション用のRA4シリーズ、コネクティビティとセキュリティを備えた高度なパフォーマンスを提供するRA6シリーズ、およびヒューマンマシンインタフェース、コネクティビティ、セキュリティ、アナログ機能を使用するアプリケーション用に最高のパフォーマンスを備えたRA8シリーズで構成されています。

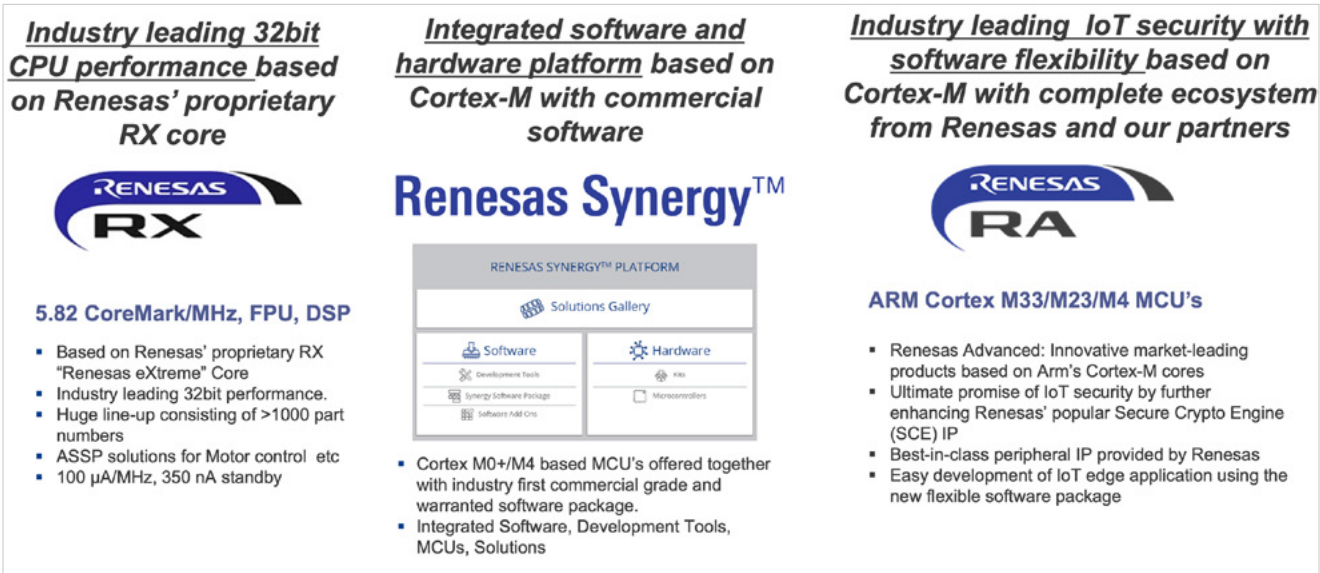

図1-1: RA ファミリは、ルネサスの32 ビットマイクロコントローラであるRXとソフトウェアプラットフォームであるSynergyを補完します

RAファミリの最初にリリースされたデバイスには、シンプルなAES (Advanced Encryption Standard)アクセラレーションからマイクロコントローラ内で分離された統合的な暗号サブシステムまで、ハードウェアベースのセキュリティ機能が組み込まれています。NIST CAPV (Cryptographic Algorithm Validation Program)認定によるセキュア暗号エンジンは、対称および非対称の暗号化と復号化、ハッシュ関数、真正乱数生成器(TRNG)、およびキー生成とMCU 固有キーラッピングを含む高度なキー処理を提供します。アクセス管理回路は、正しいプロトコルに従わない場合に暗号エンジンをシャットダウンし、また専用RAM は、平文キーがどのCPU またはペリフェラルバスにも公開されないようにします。加えて複数のグループには、Arm のv8-M TrustZone®も組み込まれています。

ファミリ全体がPSA Certified® Level 1であり、改ざん検出を提供し、サイドチャネル攻撃に対する耐性を強化します。これらの機能により、開発者は、産業用およびビルオートメーション、メータ、ヘルスケア、および家電アプリケーション用の高性能なIoTエンドポイントとEdge デバイスでセキュリティと安全性を向上させることができます。

RAファミリ内での移行は、その機能とピンの互換性、および異なるシリーズ間での周辺IP(Intellectual Property)ブロックの共通性があるために簡単に行うことができます。キーパーツ、シリコン、ツール、ソフトウェアはすべて、連携して動作するように最適化されており、必要に応じて新しいブロックを作成しますが、すでに最先端にある実績のあるIPブロックを再利用することができます。これにより、開発者にとってワークフローのさまざまな部分の間で非互換性に遭遇することを恐れることなく、ハードウェアとソフトウェアの開発を開始できます。

エンジニアは、Eclipse ベースのe² studio、コンパイラ、オンチップデバッグ、評価キット、デザインファイル、回路図、PCB (Printed Circuit Board) レイアウト、BOM (Bill of Material) などの統合開発環境(IDE) を提供するRA ファミリのツールエコシステムによってサポートされます。ルネサスが開発した使いやすいフレキシブルソフトウェアパッケージ(FSP)は、過去の資産を再利用しルネサスのすぐに使えるソフトウェア例と組み合わせることができるオープンアーキテクチャを提供します。また、ルネサスは包括的なRA パートナエコシステムを構築し、すぐに機能する追加のソフトウェアおよびハードウェアビルディングブロックを提供しました。

これにより、開発者は基本的なタスクに注意を払うという負担を軽減し、開発者が本当にやりたいことに集中できるようになり付加価値の高いアプリケーションを作成することができます。

1.1 今日の組込みシステム設計における課題

1960年代初頭にマサチューセッツ工科大学(M.I.T.)によって開発されたアポロガイダンスコンピュータから、モノのインターネット(IoT)に使用される現在のハイエンドの自動車アプリケーションまたはネットワーク網に繋がったデバイスまで、組み込みシステムは過去10年間で大幅に変更されました。今世紀に先駆けて、入力用のプッシュボタンや文字LED、あるいは出力用のニキシー管などの非常に基本的なインタフェースをいくつか採用しました。そして、それらのソフトウェアは、主にmain() 内に単純なループとして実装され、限られた量のタスクを処理するための割り込みを持つ単一の関数で構成されていました。数MIPS (毎秒100万命令)のマイクロコントローラ(MCU)は、この種のアプリケーションには、数kBのメモリと基本的なシリアル通信インタフェースで十分でした。

しかし、今日の組み込みシステムは高度に相互接続されており、イーサネット、ワイヤレス、グラフィックディスプレイなど、さまざまなインタフェースが必要です。これらはすべて最適に構成し処理する必要があり、データやメッセージを相互に交換するだけでなく、外部とも交換することがあります。これには、クロック速度60MHz以上、数MBのフラッシュメモリ、おそらく数万kBのオンチップSRAMを搭載したMCUが必要になる可能性があります。

リアルタイムオペレーティングシステム(RTOS)は、異なるスレッドに優先順位を付けて同時に実行する必要があるため、システムにとって有用であるだけでなく、場合によっては不可欠なものになります。このようなシステムの開発は、接続性の向上、機能豊富なヒューマンマシンインタフェース、およびセキュリティ要求により、ハードウェア要件は少なくなりますが、ソフトウェア中心になるため従来システムの設計方法ではもはや不可能です。

また近年、ますます開発サイクルは短くなり新機能への要求は急速に増えています。これらはすべて新しい課題に頻繁に取り組まなければならない技術者に大きな負担をかけるだけでなく、巨大な投資でもあり、そのすべてが企画当初から可視化できるわけではありません。つまり、ソフトウェア設計者は、要求に応じてアプリケーションを開発するための支援を必要とし、すべてのコードを最初から記述することなく、データとインタフェースを効率的に処理できるようにする必要があります。低レベルのドライバやセキュリティルーチン自体を記述するのではなく、アプリケーションへの価値の追加に集中したいと考えています。

ここでは、開発者が接続されたIoTエンドポイントや人工知能をEdgeデザインで即座に開始できるようにするRAファミリのフレキシブルソフトウェアパッケージのステージに入ります。ボードサポートパッケージ(BSP)、高性能で高効率なドライバ、CMSIS RTOS準拠のミドルウェア、AmazonのFreeRTOS®やMicrosoftのAzure® RTOS とともに、従来のコードやその他の利用可能なリアルタイムオペレーティングシステムの再利用を可能にするオープンソフトウェアエコシステムを提供し、エンジニアがアプリケーションに必要な機能を簡単に作成できるようにします。

1.2 RAファミリ マイクロコントローラ

ルネサスRAファミリのマイクロコントローラは、RA2、RA4、RA6、RA8の4シリーズで構成されています。これらのシリーズは、小型、バッテリで動作するセンサアプリケーションから、高性能、処理集約型の組込みシステムまで使用されます。アナログ、接続性、ヒューマン·マシン·インタフェース、セキュリティ、モータ、インバータ制御などのための周辺機器が組み込まれているため、急速に拡大するIoTやエッジコンピューティング市場に適していますが、それだけに限られるわけではありません。

すべてのRA MCU は32 ビットArm® Cortex®-M コアに基づいています。M23 コア上のRA2 シリーズ、RA4 およびRA6 シリーズデバイスはM4F コア、またはArm のv8-M TrustZone®を搭載したM33 コアのいずれかに基づいています。最後に、RA8 シリーズはArm Cortex-M7F コアの上に構築されます。これらすべてのシリーズには、ネストされたベクタ割り込みコントローラ(NVIC)、アームメモリプロテクションユニット(MPU)、シリアルワイヤデバッグ(SWD)、組み込みトレースバッファ(ETB)など、Arm の標準ペリフェラルが含まれており、開発が容易です。また、Armとしてはソリューションがなく、追加のパフォーマンスや機能が必要な場合に

は独自のIntellectual Property (IP)ブロックが追加されています。これらの追加IPブロックは、RAファミリの要求に合わせておりルネサスの実績ある技術に基づいています。

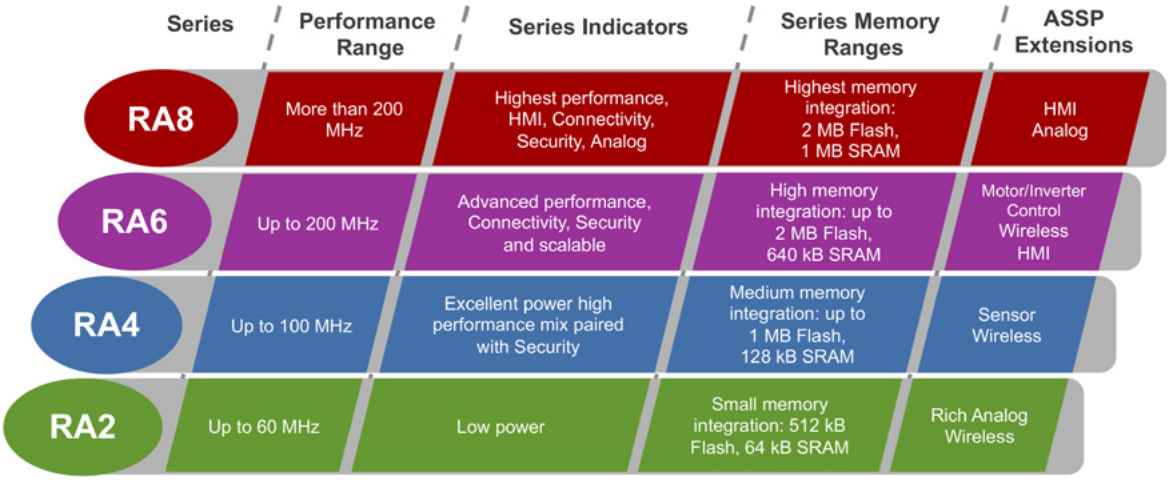

図1-2: マイクロコントローラのRA ファミリの初期シリーズ

現在リリースされている3シリーズのRAマイクロコントローラは、いくつかのグループに分かれており、以下の特長があります。

- **RA2 シリーズ- 低電力:** 最大60MHz の周波数を持つArm Cortex-M23 コアに基づく、最大512kB フラッシュメモリと64kB SRAM のメモリ。電源電圧範囲は1.6 V ～5.5 V です。周辺機器には、USB Full-Speed、CAN、24-bit Sigma-Delta ADコンバータ(ADC)、16-bit デジタル-アナログコンバータ(DAC)、静電容量式タッチセンシング、およびセキュリティと安全機能が含まれます。
- **RA4 シリーズ- 高性能& 優れた低消費電力性能:** TrustZone を搭載したArm Cortex-M33 または最大100MHz の周波数を搭載したArm Cortex-M4 コアのいずれかのCPUが組み込まれています。最大1MB のフラッシュメモリと128kB のSRAM。電源電圧範囲は1.6 V ～5.5 V です。ペリフェラルには、静電容量式タッチセンシング、セグメントLCD コントローラ、USB Full-Speed、CAN、セキュリティおよび安全機能、ならびにAD変換器およびタイマが含まれます。RA4W1 グループデバイスには、Bluetooth® low energy (BLE) 5.0 の機能を有しています。。
- **RA6 シリーズ- Advanced Performance:** Arm Cortex-M33 with TrustZone またはArm Cortex-M4F コアのいずれかに基づきます。最大240MHz の周波数。最大2MB のフラッシュメモリと640kB のSRAM。電源電圧範囲は2.7 V ～3.6 V です。ペリフェラルには、AD変換器、タイマ、外部メモリバス、Ethernet、USB Full- and-High-Speed、CAN、セキュリティと安全機能、TFT 用の静電容量式タッチセンシングとグラフィックLCD コントローラ、および2D グラフィックエンジンが含まれます。RA6T1 とRA6T2グループには、高分解能PWM タイマや高機能アナログブロックなど、モータ制御用の拡張機能が付属しています。

今後新しいデバイスが導入され、各シリーズは徐々に拡張されます。

各シリーズのすべてのMCUは（一部はファミリ全体に拡張されておりますが）機能とピン配置に互換性を持っています。小さいパッケージのデバイスのペリフェラルは、主に大きい方のデバイスのサブセットです。これにより、デバイス間でのスケーラビリティとコードの再利用が可能になります。異なるシリーズ間の類似パッケージには、互いにほぼ同一のピン配列があります。このように、開発者は最初から最終的なデバイスを選択する必要はありません。後で別のデバイスに変更することが可能であるためです。また、最終製品の柔軟な製造オプションのために、相互に複数のパッケージフットプリントを持つPCBレイアウトを作成することも可能です。

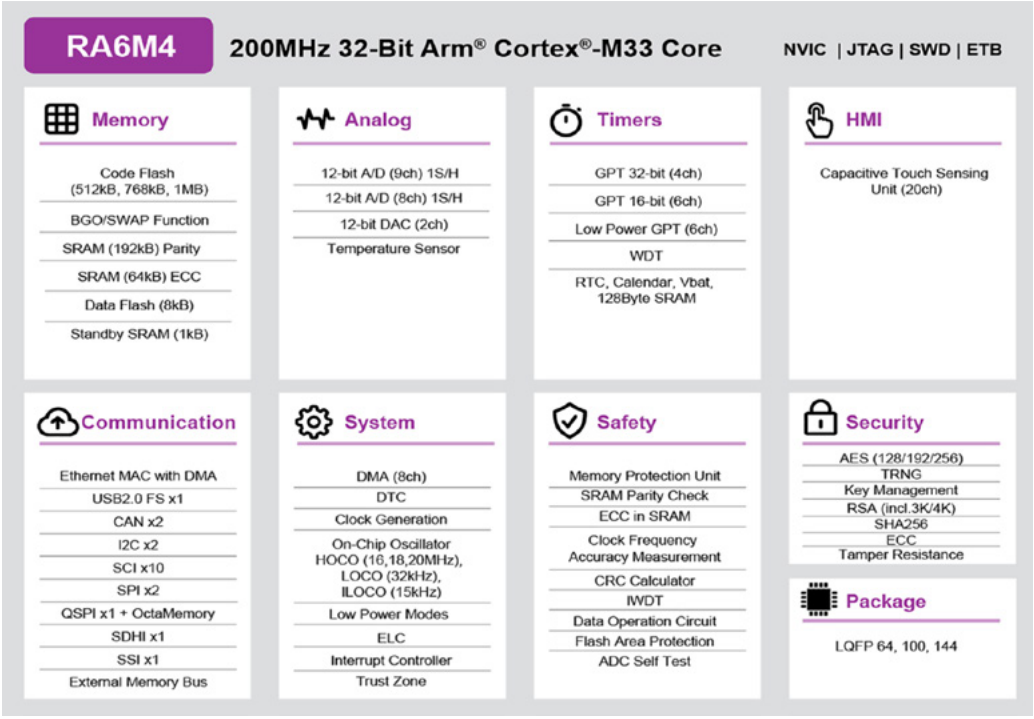

図1-3: RA デバイスのRA6M4 グループのブロック図

図1-3 に、RA ファミリの高性能RA6 シリーズであるRA6M4 グループデバイスの主な機能とペリフェラルを示します。本書の例とプロジェクトは、主にこのマイクロコントローラ用の評価キットに基づいています。

RAファミリの部品番号の中の異なる数字や文字を、特に初心者のために解読しにくい場合があるので、図1-4では、それぞれの分野の意味を説明しています。

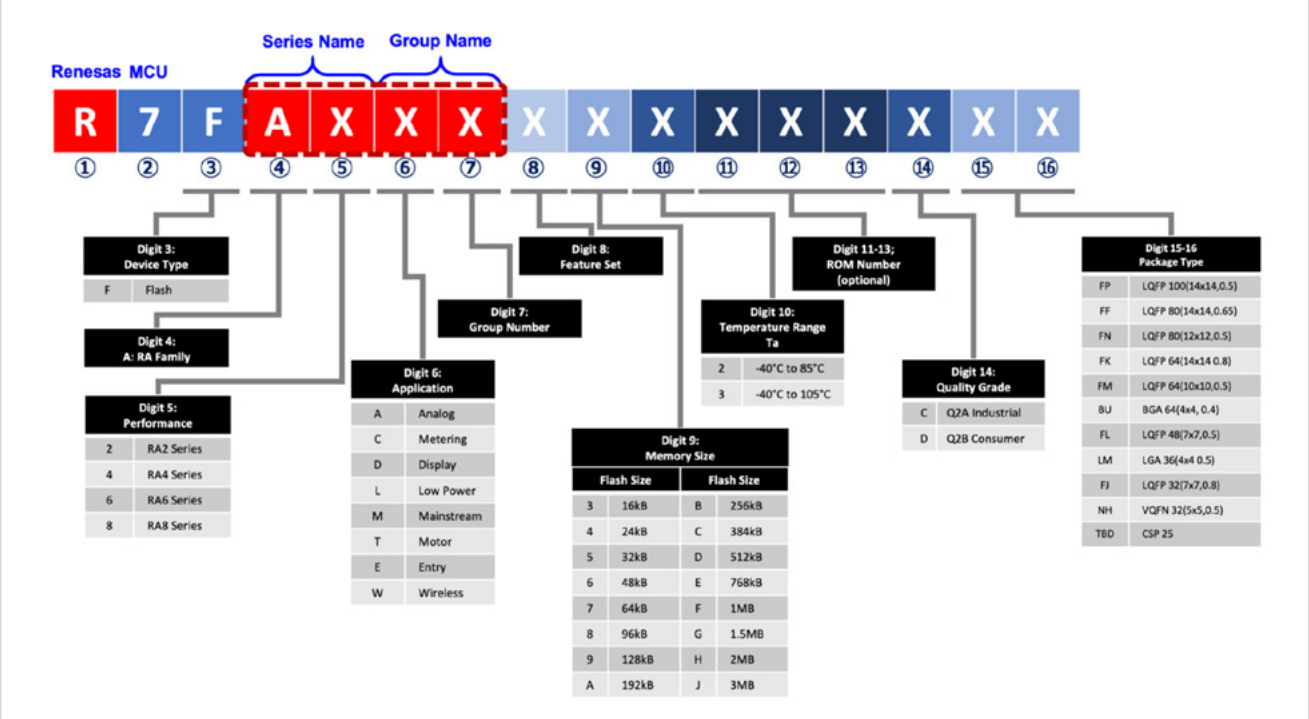

図1-4: RA ファミリのパーツ番号

1.3 フレキシブルソフトウェアパッケージ (FSP)

RAマイクロコントローラのRenesas Flexible Software Package (FSP) は、IoTのためのセキュアな接続デバイス用のソフトウェアをすばやく汎用的に作成する方法を提供し、特にRA マイクロコントローラのアーキテクチャ用に最適化されています。TCP/IP やセキュリティ用のすぐに使用できるミドルウェアスタックやプロトコル、MCU の起動と初期化コードを提供するボードサポートライブラリ(BSP)、ルネサスの開発キット、すべてのペリフェラルのハードウェア抽象化レイヤ(HAL) ドライバなどを特徴としています。これらのドライバは、パフォーマンスが高いだけでなく、メモリフットプリントも非常に小さくなります。

ドライバ、スタック、ミドルウェアのすべての機能は、使いやすいAPI (Application Programming Interface) を介してアクセスでき、RTOS (Real-Time Operating Systems) やベアメタル実装で動作します。また、Arm のTrustZone は、ソフトウェアのすべてのレベルに組み込まれています。FSPはオープンソースであり、完全なソースコードが提供されていますが、ルネサス製ハードウェアに限定されます。

FSP には、上記のソフトウェアに加えて、Arm のCortex Microcontroller Software Interface Standard (CMSIS) RTOS インタフェースを介してアクセスされるReal-Time Operating System としてAmazon のFreeRTOS®やMicrosoftのAzure® RTOSも含まれています。この標準インタフェースにより、ソフトウェアエンジニアはFSP の利点を失うことなく任意のRTOS を使用できます。

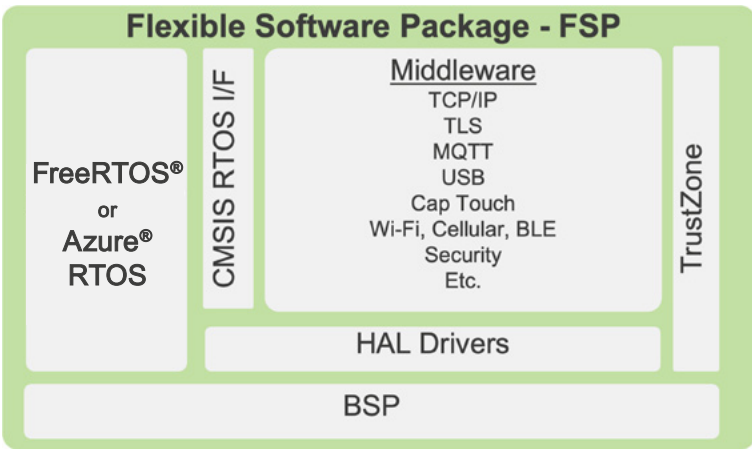

図1-5: フレキシブルソフトウェアパッケージのアーキテクチャ

FSPをできるだけ簡単に、手間をかけずに使用できるように、直感的なコンフィギュレータとコードジェネレータが付属しており、MCUとその周辺機器だけでなく、RTOSとミドルウェアモジュールも初期化できます。 また、開発者はFSPが提供するものに縛られることはありません。独自のアプリケーションコードとミドルウェアモジュールをいつでも統合できます。 これは逆の場合にも有効です。FSPはArmソフトウェアエコシステム全体と一緒に使用できます。

FSPの詳細に興味がある場合は、しばらくお待ちください。 それらについては、第2章と第3章で説明します。

1.4 RAツールとキット：簡単な概要

ルネサスでは、RAファミリの性能を探るために活用できる有用なソフトウェア・ハードウェア開発ツールを創出するために、評価段階だけでなく生産フェーズまでユーザーをサポートするために、膨大な努力を重ねてきました。

第1.3章で説明したフレキシブルソフトウェアパッケージのほかに、作成時点で利用可能な開発ツールには次のものがあります:

- **e² studio:** Eclipse-based Integrated Development Environment (IDE) には、RA ファミリのプロジェクトを作成、コンパイル、デバッグするために必要なすべてのツールが含まれおり、Microsoft Windows®やLinux上で動作します。このコンフィギュレータを使用すると、クロックモジュールやピン機能の選択など、さまざまなハードウェア機能にグラフィカルに簡単にアクセスできます。また、ミドルウェア、ドライバ、ボードサポートパッケージ、RTOS のソフトウェアモジュールを選択することもできます。ユーザーズガイドを詳細に調べる必要はありません。

  これらのコンフィギュレータは、必要なすべての設定と初期化コードを自動的に作成し、設計時に問題のあるセットアップを検出するためのエラーチェック機能を備えています。コード生成には、e² studio はGNU Arm® Embedded Toolchain のGCC コンパイラを使用します。

- **QE Tools:** QEは「Quick and Efficient」の頭字語で、Bluetooth® low energy 5.0用QEはe² studioに直接的に統合されています。QE for Capacitive Touch はタッチユーザインタフェースの初期設定と感度の調整を簡素化し、BLE 5.0 用QE はBluetooth Low Energy スタックの通信機能のテストに使用できます。

- **RA Smart Configurator:** RA Smart Configurator (RASC) は、サードパーティ製ツールおよびIDE のサポートを提供します。サードパーティツールを使用する場合、開発者がRA マイクロコントローラのソフトウェアシステム(BSP、HALドライバ、ミドルウェア、RTOS) を設定できるデスクトップアプリケーションです。

- **評価キット:** ルネサスでは、各RA シリーズの複数の製品に個別のキットを提供し、各MCU グループのデバイスごとに少なくとも1 つのキットを提供します。デバッガを統合し、設計者がデバイスのデジタルピンとアナログピン、およびその接続性とセキュリティ機能を完全に評価できるようにします。これは、評価用の自身のハードウェアが準備できていなくても可能です。

  評価キットは、Digilent PmodsやArduino™互換ボードなど、最も一般的なエコシステムと互換性のあるボード用のコネクタを介してさらに拡張できます。BOM、図面、デザイン、製造ファイルを含むデザインパッケージが提供されるため、このキットはカスタムハードウェアを構築するためのクイックスタートとして機能し、アプリケーションの開発をさらに高速化します。

- サードパーティツールとルネサスRAパートナーエコシステム:包括的なパートナーエコシステムは、ルネサスRAファミリMCUと一緒にすぐに機能するさまざまなソフトウェアとハードウェアのビルディングブロックを提供します。開発者は、セキュリティ、安全性、接続性などのテクノロジーを含む、さまざまなソリューションから選択できます。

ソフト・ハード以外のデザイン支援も提供しています。RAファミリー(**https://www.renesas.com/ra**)のルネサスホームページでは、幅広いホワイトペーパー、教材ビデオ、サンプルプロジェクトが掲載されています。開発者やシステム設計者が、最小限の労力でスピードアップできるように支援します。また、いくつかのトピックスを深く調査したい場合は、ルネサスアカデミー(**https://academy.renesas.com/?eid=1618**)がトレーニングコースを探すのに適しています。

開発中に問題が発生したり、困難になったりすることもありますが、マウスを数回クリックするだけでヘルプが表示されます。まずは、RAフォーラム（**https://www.renesas.com/ra/forum**）またはナレッジ ベース（**https://ja-support.renesas.com/**）にアクセスしてみてください。 フォーラムとナレッジ ベースには、e² studio から直接アクセスすることもできます。
[Help] → [RA Helpdesk] または [Help] → [RenesasRulz Community Forum] とアクセスしてください。

本章で学んで会得するポイント

- ルネサスRAファミリのマイクロコントローラは、セキュアで接続されたインテリジェントなIoTの確立する方法を提供しています。
- RAファミリは、さまざまなソフトウェアとハードウェアのツール、およびRA Partner Ecosystemによってサポートされています。
- ヘルプはルネサスホームページから簡単にアクセスできます。

2　フレキシブルソフトウェアパッケージ(FSP)

この章で学ぶこと:

- フレキシブルソフトウェアパッケージ (FSP) のさまざまなコンポーネントと、それらがどのように階層化されているか。
- さまざまなレイヤの詳細とそれらがどのように連携しているか
- FreeRTOS™、Azure® RTOSの仕様とRTOS を使用する利点

マイクロコントローラ (MCU) は、その上で実行されているソフトウェアが他タスクに依存していたり、ソフトウェアエコシステムが複雑すぎて開発者にとって使いにくい場合、その潜在能力を十分に引き出すことができません。ソフトウェア開発ツールとRAパートナエコシステムの組合せと共に、RAファミリのような適切に考え出されたコンセプトは、エンジニアがプログラミングする際に記述や保守が容易であるだけでなく、このファミリのマイクロコントローラの性能と特徴を最大限に引き出すアプリケーションを実現するのに役立ちます。

開発者は、MCU、ソフトウェア、ツール、キット、ソリューションのすべての主要な部分を完全に連携させることで達成され、ルネサスはこの目標を達成するために多くの労力を費やしています。

フレキシブルソフトウェアパッケージ (FSP)は、ソフトウェアを念頭において開発されたMCUのRAファミリのアーキテクチャ用に特に最適化されています。FSPの開発における第一の目標は、通信やセキュリティのような組込みシステムにおける一般的なユースケースの実装を容易にするために、軽量で高効率な機能とドライバをエンジニアに提供することでした。これらは、既存コードの柔軟な再利用と3rdパーティとの協調を可能にするオープンソフトウェアエコシステムをも提供します。

FSP は、ミドルウェアスタック、プロダクション対応のRTOS に依存しないハードウェア抽象化レイヤ(HAL) ドライバを統合し、その基礎となるのはBoard Support Package(BSP)です。FSPには、広く採用されているリアルタイムオペレーティングシステム(RTOS)としてAmazon Web Services® and the Microsoft Azure™ RTOSが含まれています。これにより、拡張性が高く、すべての機能にシンプルで堅牢なAPI (Application Programming Interface)呼び出しを介してアクセスできるようになります。これにより組み込みシステム設計用の最適化された使いやすい高品質ソフトウェアパッケージが作成されることとなり、シンプルで堅牢なAPI (Application Programming Interface)呼び出しを介してアクセスできるようになります。FSP全体がルネサスによって完全にサポートされ、開発者は統合開発環境内から、または **https://github.com/renesas/fsp/releases** でFSPのGitHub®リポジトリから表示させるかまたはダウンロードすることによってソースコードを完全に可視化できます。

2.1 FSPの概要

前述のように、フレキシブルソフトウェアパッケージ (FSP) は包括的なソフトウェアです。その目的は、組み込みシステムソフトウェア開発中にほとんどのシナリオをカバーするメモリフットプリントが少ない、高速で効率的なドライバとスタックを提供することです。FSPに含まれるのは、以下の部分になります。

- RA ファミリのすべてのハードウェアキットおよびマイクロコントローラ用にカスタマイズされたボードサポートパッケージ(BSP)。サポートされているすべてのモジュールのスタートアップコードを提供し、FSP モジュールを問題なく連携させるための基盤を提供します。独自のハードウェアを使用する開発者は、e² studio に組み込まれているユーザパッククリエータを使用して、エンドプロダクトや独自のボードに合わせて調整することにより、BSP を利用できます。

- RTOS に依存しないHAL (Hardware Abstraction Layer) ドライバで、すべてのオンチップペリフェラルとシステムサービスに対して、効率的なドライバに小さなメモリフットプリントを提供します。ビット設定を抽象化し、アドレスを登録してあるため、マイクロコントローラ内の基盤となるハードウェアのドキュメントを深く調べる必要はありません。

- ミドルウェアスタックとプロトコルは、RTOSの有無にかかわらず動作し、Arm®の統合APIを使用します。これにより、WiFi、Bluetooth®Low Energy、MQTT 接続などの接続機能をクラウドサービスに簡単に実装できます。USB 通信、グラフィック処理、静電容量式タッチなど、その他のスタックも含まれています。

- FreeRTOS™リアルタイムオペレーティングシステムは、プリエンプティブスケジューリング、オブジェクトのためのRAMの柔軟な割り当て、タスク通知、キュー、セマフォ、バッファのための異なる方法を備えたマルチタスクリアルタイムカーネルを提供します。ライブラリFreeRTOS+FAT とFreeRTOS+TCP は、接続が必要なアプリケーションに追加機能を提供します。FreeRTOS の使用はオプションです。FSPはベアメタルインプリメンテーションやその他のRTOSでも使用できます。

- Microsoft Azure® RTOS Real-Time Operating SystemはAzureRTOS ThreadX®などのコンポーネントと、ファイルシステム（FileX®）、グラフィカルユーザインターフェイス（GUIX™）、USBおよびTCP/IP通信（USBX™およびNetX Duo）などのさまざまなスタックおよびミドルウェアで構成されています。ThreadX RTOSは、プリエンプティブスケジューラ、スレッド間通信と同期、ソフトウェアタイマ、および小さなメモリフットプリントを提供します。 そのメモリ管理により、RAMの柔軟な割り当てとスタックオーバーフローの検出が可能になります。これらのパッケージはすべてFSPエコシステムに統合され、Azure IoTへの接続を簡素化します。

- その他のサードパーティ製ソフトウェアソリューションもFSP に含まれています。例としては、Arm® Cortex® Microcontroller Software Interface Standard (CMSIS) ハードウェアアブストラクションレイヤ、Arm Mbed™ Crypto およびTLS 暗号化ライブラリ、Arm Littlefs フェイルセーフファイルシステム、Segger のemWin 組み込みグラフィックスライブラリおよびJ-Link®デバッガソフトウェア、TES D/AVE 2D グラフィックスレンダリングライブラリなどがあります。

FSP の開発中の目標の1 つは、十分に文書化された統一的で直感的なAPI を備えた使いやすいソフトウェアを作成することでした。エンジニアは、モジュールごとに、サンプルコードを含む詳細なユーザードキュメントを手元に用意しています。これには、GitHubリポジトリ、または開発環境内の必要な場所に情報を表示するe² studioのスマートマニュアル機能が含まれます。DoxygenはFSPのデフォルトのドキュメントツールとして使用されているため、さまざまなモジュールのソースコード全体のDoxygenコメントにも詳細が記載されています。

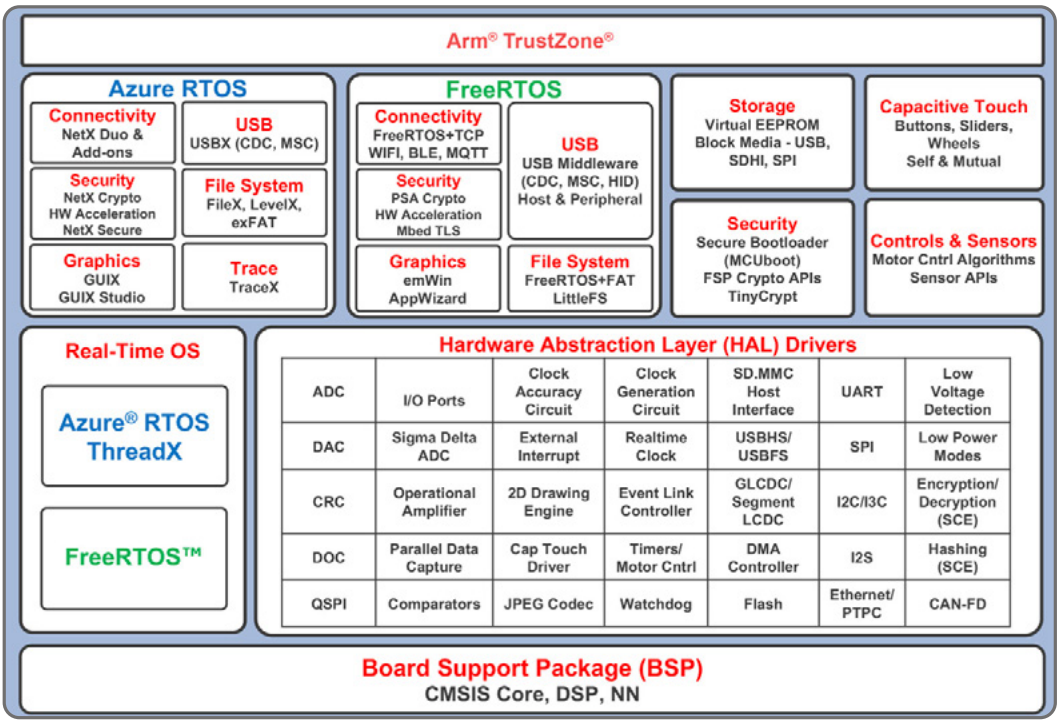

図2-1: フレキシブルソフトウェアパッケージ のさまざまなレイヤとその機能

各モジュールのビルド時設定も含まれています。これらを使用して、アプリケーションが必要とする機能セットのモジュールのサイズを最適化できます。FSP は、使用するモジュールが必要とするペリフェラルをMCU が持っている限り、そのモジュールがRA ファミリのどのマイクロコントローラでも同じであるため、拡張性が高くなります。FSPの品質は、ピアレビュー、自動化された要求ベースのテスト、および自動化された静的解析によって保証されています。

図2-1 に、各レイヤのアーキテクチャを示します。それぞれのレイヤは、APIコールを介してユーザアプリケーションからアクセスできますが、アプリケーションはインタフェースしているレイヤのAPI関数のみを呼び出すことが強く推奨されます。ISO/IEC 9899:1999 (C99) C プログラミング言語規格に従ってプログラミングされている間は、すべてのFSP モジュールは、e² studioまたはサポートされている開発環境のいずれかでC およびC++ で記述されるアプリケーションを作成できます。Arm のTrustZone®に対する認識は、設計者が安全なプログラムを簡単に作成できるように、すべてのレベルのソフトウェアに組み込まれています。

FSPの更新とバグ修正は定期的に行われ、ロードマップに従ってスケジュールされたリリースが行われ、メジャーリリースまたはマイナーリリースごとに新機能が導入されます。 長期的には、ルネサスは他のメーカーがメンテナンスをやめるポイントをはるかに超えて自社製品をサポートすることで高い評価を得ています。 これは「As is」で配布されるソフトウェアに開発者が取り残されないことを意味します。

2.2 ボードサポートパッケージ(BSP) の概要

ボードサポートパッケージは、フレキシブルソフトウェアパッケージ (FSP) の最下層を構築し、他のFSPモジュールが問題なく連携できるようにするための基盤を提供します。 これを実現するための主な責任は、MCUをリセットしてユーザアプリケーションに戻すことです。 そこに向かう途中で、クロック、割り込み、スタック、ヒープ、スタックモニター、およびCランタイム環境をセットアップします。また、ポートのI/Oピンを構成し、ボード固有の初期化を実行します。

し したがって、このパッケージはボードとMCU の特定の組み合わせで固有です。どちらも、IDE (Integrated Development Environment) のプロジェクトウィザードを使用してe² studio でプロジェクトをセットアップするときに選択されます。ルネサスが提供

するすべてのキットは、BSP がサポートしています。e² studio 内のコンフィギュレータは、FSP から必要なファイルを抽出し、ユーザインタフェースで行われた入力に基づいて設定します。ボードサポートパッケージ 自体は大量のデータ駆動型で、設定ファイル、ヘッダファイル、API (Application Programming Interfaces) で構成されています。

BSP のコアは、CMSIS (Arm® Cortex® Microcontroller Software Interface Standard) に準拠しており、その規格の要件と命名規則に従います。ボードサポートパッケージは、各MCU のデータの静的マクロリストを使用します。マクロ自体はすべてのプロセッサで同一ですが、MCU ごとに異なる値を持ちます。

BSP はパッケージを使用して任意のプロジェクトで使用可能なパブリック関数を提供し、それによってサポートされるMCU およびボード全体で共通の機能にアクセスできます。機能にはハードウェアとソフトウェアのロック/アンロック、割り込み処理、コールバック関数の登録とフラグのクリア、ソフトウェア遅延、レジスタ保護などがあります。これらのルーチンの名前は、R_BSP_ で始まり、関連するマクロはBSP_ で、データ型定義は_bsp で始まり、FSP の他の部分と容易に区別できます。唯一の例外は、CMSIS コアで説明されている機能を提供するルーチンです。さらに、BSP には、読み取りや書き込みなどのピン機能や、ボード上で使用可能なLED に関する情報(数とI/O ピン) を返すためのルーチンがいくつか含まれています。これらの機能は、アプリケーションコードで直接使用できます。

新しいキットとデバイスは使用可能になるとBSP に追加され、現在および新しいデザインの確実な長期ベースが保証されます。カスタムボードのボードレベルのサポートはe² studio に組み込まれているUser Pack Creator を使用して簡単に生成できます。

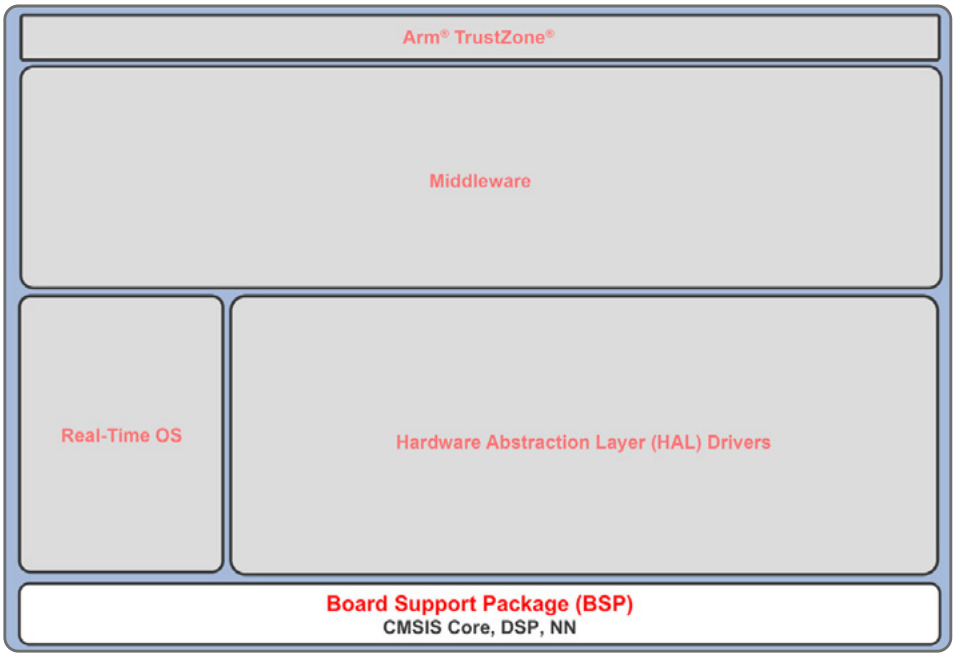

図2-2: ボードサポートパッケージ(BSP)はフレキシブルソフトウェアパッケージ(FSP)の基本機能です

2.3 HAL ドライバの概要

ボードサポートパッケージ(BSP)の上にはハードウェア抽象化レイヤ(HAL)があります。これは効率的なデバイスドライバにペリフェラル用の小さなメモリフットプリントを提供し、MCU のレジスタと連携して使いやすいインタフェースを実装し、プログラマをハードウェアから分離します。これらはモジュールの集合(図2-3 を参照) であり、それぞれがSPI (シリアルペリフェラルインタフェース) やADC (アナログ- デジタルコンバータ) などのRA ファミリのマイクロコントローラで使用可能なペリフェラル用のドライバです。また、その名前はr_ で

始まり、フレキシブルソフトウェアパッケージ(FSP) の他の部分と容易に区別できるようになっています。これらのモジュールはすべて本質的にRTOS に依存せずに使用することができます。

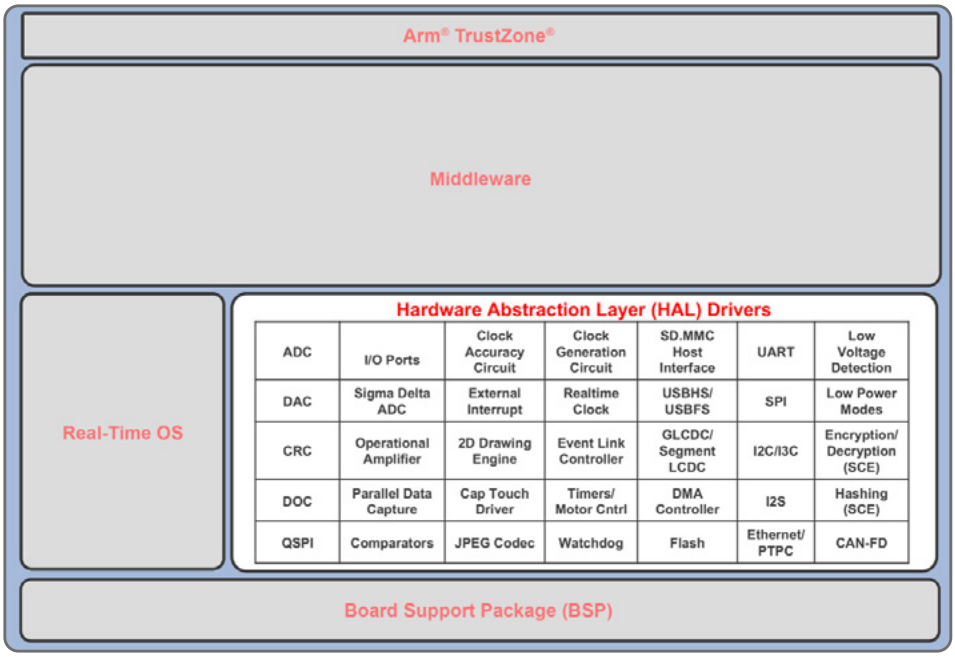

図2-3: HAL レイヤは、オンチップペリフェラルのドライバを提供します

ハードウェアを抽象化するためのインタフェースは、HAL のすべてのモジュールで一貫しており、拡張することができます。ペリフェラルの中には、複数のインタフェースをサポートするものもあれば、複数のペリフェラルでサポートされるものもあります。その利点は、要件をより高いレベルで変更できるため、柔軟性が得られることです。たとえば、コードが元々専用のハードウェアSPI ペリフェラル用に記述されていたものの、後でSCI (Serial Communication Interface)のSPI 機能を使用する必要がある場合は、構成情報を変更するだけで十分です。アプリケーションコード自体は変更されません(図2-4 を参照)。API 機能はHAL インタフェースを介して直接アクセスできますが、ほとんどの機能はFSP で使用可能なさまざまなミドルウェアスタックとプロトコルを介してアクセスすることもできます。これが実際には推奨される方法です。

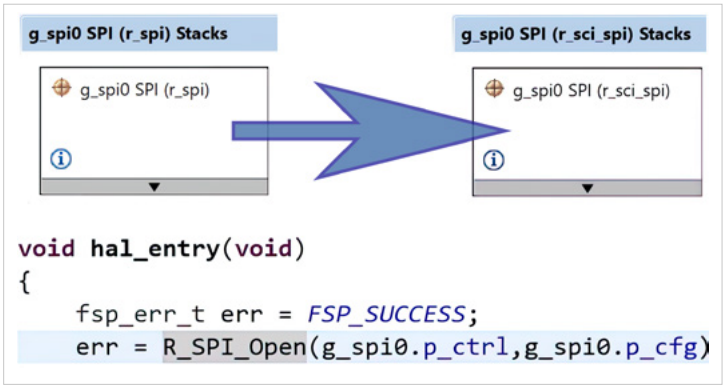

図2-4: 基盤となるペリフェラルが変更された場合でも、この例ではSPI ペリフェラルからSCI ポートへの変更は、アプリケーションコードは同じままです

HALドライバは静的および動的分析用の業界標準ツールを使用して実行されるユニットおよびシステムテストを通じて保証されます。すべてのドライバはデフォルトでノンブロッキングであり、実行ステータスを返します。また、メモリ自体は割り当てられません。ワーキングメモリは呼び出しルーチンによってドライバに渡されます。

2.4 ミドルウェアの概要

HALレイヤの上とユーザアプリケーションのすぐ下に、ミドルウェアレイヤが配置され、アプリケーションにスタックとプロトコルが提供されます。その目的は、イーサネットやUSBなどの複雑なペリフェラルや、マイクロコントローラ(MCU)のRAファミリのさまざまなセキュリティ機能の使用を容易にすることです。ミドルウェア層のモジュールは、さまざまなシステムレベルや技術固有のサービスの抽象化を提供し、シンプルなAPIで豊富な機能を実現します。FreeRTOS™と統合して、複数のユーザスレッド間のリソースの競合や同期を管理しますが、ベアメタル実装やその他のリアルタイムオペレーティングシステムでも使用できます。

フレキシブルソフトウェアパッケージ (FSP) のミドルウェアモジュールは、FreeRTOS+TCP からのスケーラブルでスレッドセーフなTCP/IP スタックや、MCU のシリコン上のEthernet ペリフェラルにアクセスするための使いやすいAPI など、簡単に接続できるオプションを提供します。その他のモジュールは、デバイス設定を含むWiFi およびソケット実装をサポートします。また、Bluetooth® low energy (BLE) 5.0 機能が利用可能であり、RA4W1 グループデバイスなどの選択されたマイクロコントローラ上でBLE 無線ペリフェラルを制御することができます。

一部のクラスを搭載したホストモードやデバイスモードでの通信もサポートされています。上記のミドルウェアモジュールはすべて、異なるコンフィギュレータまたはQE Tools を使用して簡単に設定できます。

FreeRTOS+FATやAzure RTOS FileXを使用すると、ファイルベースのアクセスとストレージを可能にし、スレッド対応でスケーラブルなDOS互換ファイルシステムを使用できます。FAT12、FAT16、またはFAT32 ファイルシステムを使用した実装を可能にします。Arm のLittleFS は、特にマイクロコントローラと組み込みシステム用に開発された、小さなフェイルセーフファイルシステムもサポートされています。

その他のモジュールは、ソフトウェアエンジニアがエンドツーエンドのクラウド接続でアプリケーションを作成するのに役立ちます。これらのモジュールには、ユーザのソフトウェアをAmazon Web Services®、Microsoft Azure™、およびGoogle Cloud Platform™サービスに直接接続するためのサポートが含まれます。通信はイーサネットまたはWi-Fi インタフェースを介して提供され、MQTT プロトコルをサポートします。接続のセキュリティは、Arm のMbed™ Crypto インタフェースのTransport Layer Security (TLS) プロトコルによって保証されます。
FSP の暗号API もArm のMbed Crypto に基づいており、ハードウェアおよびソフトウェア暗号化をサポートしています。楕円曲線暗号(ECC) とTrue Random Number Generation (TRNG) と同様に、AES 128 とAES 256、SHA-256 とSHA-224 の計算、およびRSA 2048 が処理されます。Arm PSA Crypto API のMbed Crypto 実装のハードウェアアクセラレーションも使用可能でセキュアなデバッグです。

静電容量式タッチセンシングを使用するアプリケーションでは、ルネサス静電容量式タッチセンシングユニット(CTSU)テクノロジに基づくドライバおよびミドルウェアをすぐに利用できます。これらには、QE for Capacitive Touch ツールで簡単に作成できるボタンやスライダーなどのウィジェットの統合を簡素化するためのAPI が含まれています。

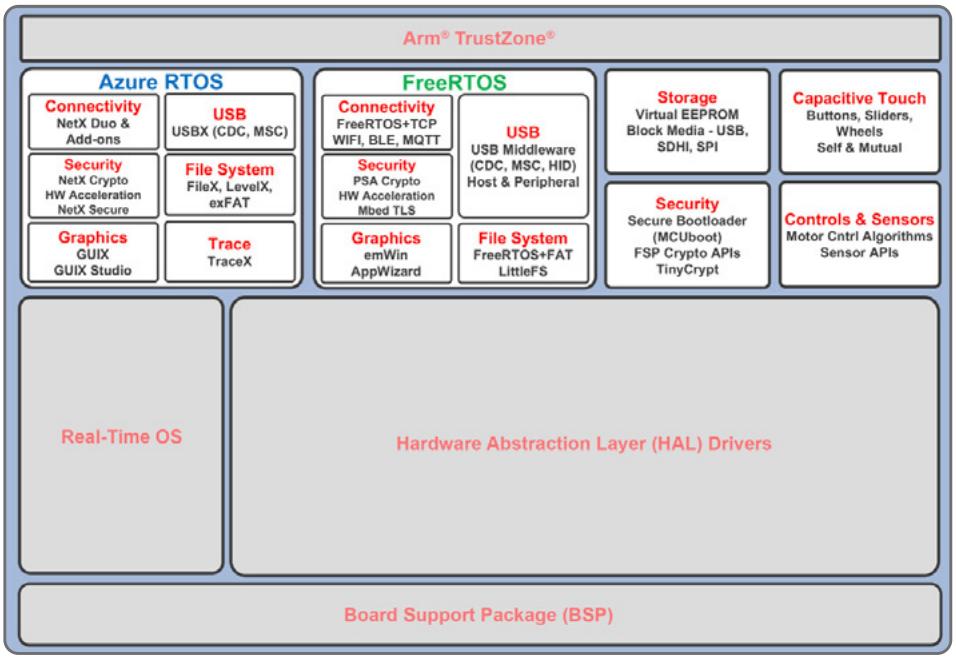

図2-5: FSP の異なるミドルウェアモジュールは、簡単な接続とその他の機能を提供します

RA6M3 Group デバイスなど一部のMCU では、グラフィックスをサポートするミドルウェアを使用する準備ができています。FSP には、高品質のGUI 用のSegger emWin グラフィックスライブラリが付属しています。画像処理、2Dアクセラレーション、ウィンドウマネージャ、イベント処理、回転/スケーリング、アルファ/アンチエイリアスなどの機能を実装しています。

ミドルウェアスタックとプロトコルを使用する開発者はより高いレベルの抽象化、さまざまなプロセッサでのコードの再利用が可能となります。したがって、より一貫性のあるコードの恩恵を受けながら、潜在的には製品とアプリケーション間で一般的に使用されるタスクを再発明するという負担から解放されます。 同時に、フレキシブルソフトウェアパッケージの他の部分は、マイクロコントローラで利用可能なハードウェアと直接インタフェースする必要性をさらに減らします。

2.5 リアルタイムオペレーティングシステム

FreeRTOS™はリアルタイムオペレーティングシステム (RTOS)であり、ルネサスRAファミリに適しています。メモリ資源が不足し、検証されたロバスト性が必要な組込みシステムにおいて、マイクロコントローラと小型マイクロプロセッサ専用に開発されました。今日では、組み込みオペレーティングシステムのデファクトスタンダードとなっています。マルチタスクスケジューラ、オブジェクトのメモリ割り当てのためのいくつかのオプション、タスク通知、キュー、セマフォなどを提供します。

FreeRTOS の処理オーバヘッドは、メモリフットプリントと同様に非常に小さくなります。通常、FreeRTOS カーネルバイナリイメージには、6 ～12KB のフラッシュメモリと、カーネル自体に数百バイトのRAM が必要です。オペレーティングシステムはシンプルに設計されおり、そのカーネルは、3つの共通ソースファイルと、使用中のマイクロコントローラに固有の1つのファイルのみで構成されています。

この機能が豊富で完全に維持されたリアルタイムオペレーティングシステムでは、15 年以上にわたって世界の主要なMCU ベンダと一緒に開発され、現在はAmazon Web Services®によって所有されています。MIT オープンソースライセンスの下でソースコードとして自由に配布され、RA ファミリに適合したバージョンがFlexible Software Package (FSP) に統合されています。オペレーティングシステムは厳密に品質管理されており、コアソースファイルはMISRA®コーディング標準ガイドラインに準拠しています。

FreeRTOS は、プリエンプティブまたは協調動作を可能にし、複数のスレッド、キュー、ミューテックス、セマフォ、およびソフトウェアタイマをサポートし、非常に柔軟なタスク優先度管理を備えています。タスク通知は軽量メカニズムを使用し、高速で汎用性が

あります。トレース記録と、実行時にスレッドの統計を収集するためのプロビジョニングが利用できます。
RTOS のさまざまなリソース(スレッド、ミューテックス、メモリ管理など) は、e² studio内から、または他の統合開発環境(IDE) で使用可能なスマートコンフィギュレータを使用して、GUIを介して設定できます。これにより、RTOS 機能のインプリメンテーションが非常に容易になります。

2.5.1 RTOS を使用する理由

それほど昔の話ではありませんが、組み込みシステムのためのソフトウェア開発は比較的簡単でした。ほとんどのシステムは、main() 関数内の無限ループで実行されていた単一のバックグラウンドタスクで構成されていました。I/O ピンからの入力の読み出し(押しボタンの状態など)や、電圧·電流の計測値を乗算して消費電力を求めたり、ディスクリート出力を更新したりするなどの演算を実行します。

マイクロコントローラに接続されている少数のペリフェラルが割り込みを発行し、アナログ- デジタルコンバータからの新しい変換結果が利用可能であり、読み出す必要があることをCPU に通知しました。または、RS-232 ポートが最後のワードの送信を終了し、新しいデータを待機させます。これらの非同期イベントは、従来、割り込みサービスルーチン内で処理されていました。また、保留中の割り込みがない場合、ソフトウェアはバックグラウンドループでアイドル状態であり、割り込みが発生するのを待機していました。

組み込みシステムがより複雑になり、外部への接続性が高まり、サービスを提供するユーザインタフェースが増え、実行するタスクが増えるにつれ、上記のベアメタルセットアップを維持することは困難になり、オペレーティングシステムは好ましいものになりつつあります。なぜでしょうか?それはバックグラウンドループの応答時間は、コード内の意思決定ツリーの影響を受け、ソフトウェアに変更が加えられると変更されるため、ベアメタル実装では予測が難しく、決定論的ではありません。もう1つの欠点は、コードがシリアルに実行されるため、バックグラウンドループ内のすべてのタスク（または「スレッド」）の優先度が同じであるということでした。つまり、上記の例のプッシュボタンと電力計算の読み取りは、たとえボタンが複数回押されたとしても、常に互いに次々に実行されていたことになる。言い換えれば、これらの事象のいくつかは、組み込みコンピューティングでは無意味なものとして見落とされている可能性があります。現在のシステムは、より応答性が高く、より予測可能である必要があり、リアルタイムである必要があります。

1つの可能性としては、必要な機能を自分で記述することです。しかし、独自のスケジューラやタスク間通信の設計と実装は複雑でエラーを起こしやすいものです。さらに、このようなソフトウェアは、新しいプロセッサに移植することが困難であり、加えて維持が困難です。したがって、リアルタイム·オペレーティング·システムは、特にRTOS が開発初期段階からソフトウェアに含まれている場合、プログラマの仕事を楽にするのです。

オペレーティングシステムの重要な必要性を見てきましたが、次の質問としてこのようなものがあります。なぜ、Windows®やLinuxのような既製のOSを使用しないのでしょうか。それらは私たちの日常生活の一部でもあり、それらを活用すればうまくいくように思えます。しかしながら、これらを使用しない理由はいくつかあります。まず、非常に多くの機能を提供していますが、そのほとんどは組み込みシステムでは不要であり、構成も不十分です。次に、リソースに制限のある組み込みアプリケーションでは使用可能なものよりも多くのリソースとメモリ領域が必要です。最後に、リアルタイムシステムの要求を満たすには、タイミングのばらつきがまだ大きすぎます。

これらはすべて、組み込みシステムで実行されるOS に特別な要件があることを示しています。最初の最も重要なものは、タイミングの予測可能性です。RTOS は決定論的である必要があります。つまり、割り込みを無効にする時間など、ブロック時間の上限を開発者が把握し、使用できるようにする必要があります。

2 番目の要件は、RTOS がタイミングとスケジューリングを管理する必要があることです。タイミング制約と期限を認識し、高

分解能のタイムサービスを提供する必要があります。最後に、オペレーティングシステムは高速で、小規模に構成可能である必要があります。

RTOSによって利用可能になるもう1つの主要な機能は、スレッド管理です。これにより、スレッドの状態、キューイングを処理することにより、スレッドを使用してMCUで準並列タスクを実行できます（スレッドは軽量プロセスと解釈されます）。 プロセス、およびプリエンプティブスレッドと割り込み処理を可能にします。 アプリケーションに提供されるその他のサービスは、スケジューリング、スレッド同期、スレッド間通信、リソース共有、および時間参照としてのリアルタイムクロックです。

多くのソフトウェア設計者は、RTOSの複雑さと学習曲線を恐れているため、RTOSを使用するという考えに頭を悩ませています。 ただし、リアルタイムオペレーティングシステムには、リアルタイムイベントへの応答性の向上、スレッドの優先順位付けと簡単な挿入、最初のステップをマスターした後の開発時間の短縮、アプリケーションへのサービスの追加など、多くの利点があります。 したがって、利点は最初の困難をはるかに上回ります。

RTOSを使用することにはもちろんいくつかの欠点があります。それは追加のRAMとフラッシュメモリが必要であり、各スレッドは独自のスタックを必要とし、メモリの必要性をさらに増やしてしまうことです。 コストも要因になる可能性がありますが、このオペレーティングシステムは無料で提供されるため、これはFreeRTOS™の議論ではありません。

RTOSを使用するメリットを説明しましたが、今度はFreeRTOSとAzure RTOSの様々な特長をみてみましょう。

2.5.2 FreeRTOS™の主な特長

FreeRTOS™はメモリリソースが不足しており、実証済みの堅牢性が必要な組み込みシステムのマイクロコントローラおよび小型マイクロプロセッサ向けに特別に開発されました。 現在、これは組み込みオペレーティングシステムのデファクトスタンダードです。マルチタスクスケジューラ、オブジェクトのメモリ割り当てのためのいくつかのオプション、およびタスク通知、キュー、セマフォ、およびさまざまなバッファのためのメソッドを提供します。

FreeRTOSの処理オーバヘッドはメモリフットプリントと同様で非常に小さいです。 通常FreeRTOSカーネルバイナリイメージには、6～12KBのフラッシュメモリとカーネル自体に数百バイトのRAMが必要ですが、マイクロコントローラのリソースは主にアプリケーションに費やすことができます。 このオペレーティングシステムはシンプルに設計されており、カーネルは3つの共通ソースファイルと、使用中のマイクロコントローラに固有の1つのファイルのみで構成されています。

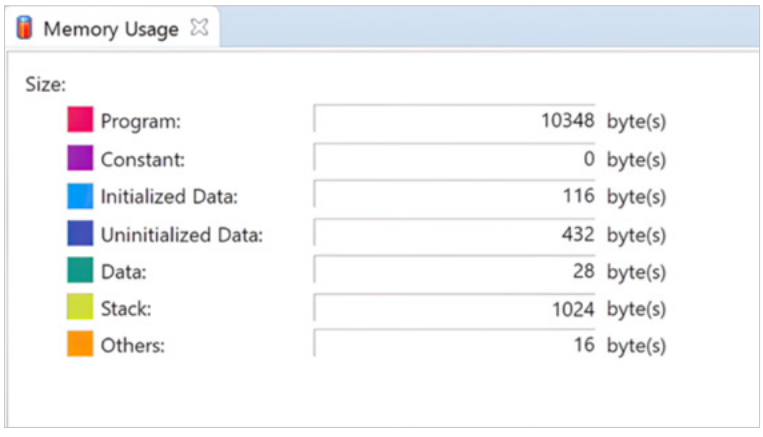

図2-6: FreeRTOS のメモリフットプリントは非常に小さくなります。この例では、スレッドのない空のFreeRTOS FSP プロジェクトでは、10KB をわずかに超えるフラッシュメモリのみが使用され、これにはBSP を介したMCU の初期化が含まれます

この豊富な機能を有し完全にメンテナンスされたリアルタイムオペレーティングシステムは、世界をリードするMCUベンダと共同で開発されました。15年以上の期間にわたって、現在はAmazon Web Services®が所有しており、MITの下でソースコードとして自由に配布されています。またオープンソースライセンスとRAファミリに適合したバージョンは、フレキシブルソフトウェアパッケージ （FSP）に統合されています。 オペレーティングシステムは非常に厳密に品質管理されており、コアとなるソースファイルはMISRA®コーディング標準ガイドラインに準拠しています。

FreeRTOSはプリエンプティブまたは協調動作を可能にし、複数のスレッド、キュー、ミューテックス、セマフォ、およびソフトウェアタイマをサポートし、非常に柔軟なタスク優先度管理を備えています。 タスク通知はシンプルなメカニズムを使用しており、高速で汎用性が高いです。 トレースの記録と、実行時にスレッドの統計を収集するためのプロビジョニングが利用できます。

RTOS のさまざまなリソース(スレッド、ミューテックス、メモリ管理など) は、e² studio内から、または他の統合開発環境(IDE) で使用可能なスマートコンフィギュレータを使用して、GUIを介して設定できます。これにより、RTOS 機能のインプリメンテーションが非常に容易になります。

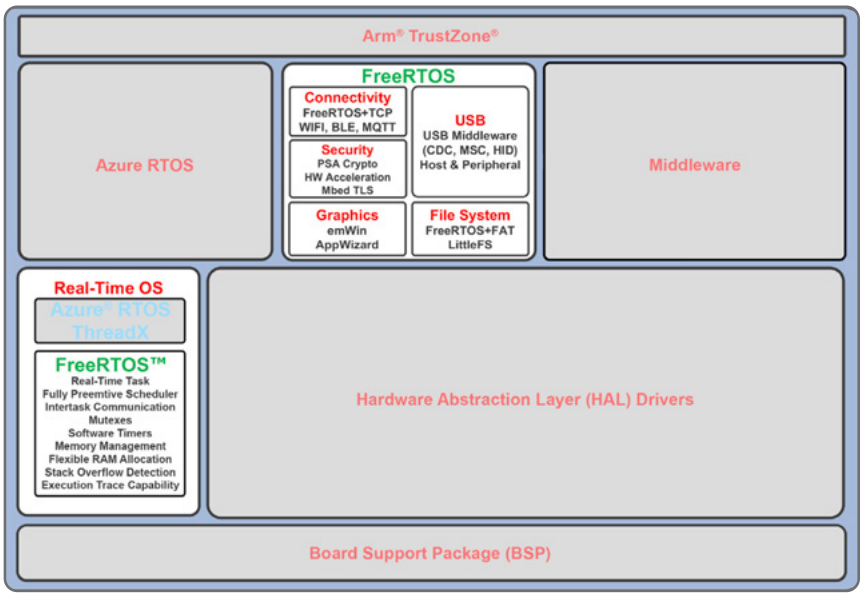

図2-7: FreeRTOSはマルチタスクスケジューラ、オブジェクトのメモリ割り当てのオプション、およびタスク通知、キュー、セマフォ、およびバッファを提供します

FreeRTOS のリアルタイムスケジューリングアルゴリズムにより、実行可能な状態としてキューに入れられた最も高いプライオリティのスレッドが常に実行されるようになります。同じプライオリティの複数のスレッドがレディ状態にある場合、2つのオプションがあります。タイムスライシングがFSP のデフォルトであるRTOS の設定ファイルで無効になっている場合、RTOS ティック割り込みが発生すると、スケジューラはスレッド間を切り替えません。タイムスライシングが有効な場合、スケジューラはティック割り込みごとに同じ優先順位のスレッド間で切り替わります。タイムスライシングの設定に関係なく、ラウンドロビンが使用され、ある時点ですべてのスレッドが実行状態になるようにします。

スレッドに優先順位を付ける際、アーキテクチャに最適化された優先順位付け方法が有効になっている場合はソフトウェアエンジニアは最大32の優先順位レベルを使用できます。また、この方法が無効になっており一般の設定の場合は無限の数になります。各優先順位レベルはカーネル内のRAM を消費し実行時間に悪影響を与えるため、レベル数はできるだけ低く保つ必要があります。また、同じレベルを共有するスレッドがラウンドロビンを使用してスケジューリングされ各スレッドが確実に実行されるようになるため、アプリケーションが10 ～15 を超える異なる優先順位を必要とすることは非常にまれです。

スレッド、キュー、セマフォなどFreeRTOS 内のカーネルオブジェクトは、コンパイル時に完全に静的に作成することも実行時に動的に作成することもできます。どちらのバージョンにもメリットとデメリットがあり、これは開発者が決定することになります。また、これはアプリケーションの要件に基づいて決定する必要があります。動的メモリのオブジェクトの割り当ては、オブジェクトが削除されるとメモリを再利用できるため、より簡単であり、アプリケーションの最大RAM 使用量を削減できる可能性があります。動的割り当ては自動的に行われるため、開発者はこの点を気にする必要はありません。

メモリの静的割り当てにより、開発者は目的のメモリ位置にオブジェクトを配置できるため、より多くの制御が可能になります。また、リンク時にすでにアプリケーションに必要なRAM の量を決定することもできます。静的割り当てと動的割り当ての決定は最終的なものではありません。必要に応じて、両方の方法を同じプログラム内で使用することができ、最大限の柔軟性と常に悩ましいリソースである「メモリ」の最適な管理を可能にします。

RTOS によって管理されるその他のリソースは、ソフトウェアタイマです。これらのアプリケーションタイマは、ワンショットとオートリロードの2つの動作モードで使用できます。ワンショットタイマは、タイマが一周した後に1 回だけユーザ関数を呼び出し、定期タイマは一定の間隔で繰り返しユーザ関数に入ります。FreeRTOSにおけるソフトウェアタイマの実装は、タイマが満了しない限りいかなる処理時間も消費せず、ティック割込みにオーバヘッドを加えないので非常に効率的です。また、割り込みが無効になっている間はリンクリスト構造体をウォークせず、割り込みサービスルーチン(ISR)内からタイマコールバック関数に入ることもありません。

ソフトウェアタイマは使用前に作成し、スケジューラの起動後に自動的に作成されるタイマサービスタスクのコンテキストで実行する必要があります。タイマの作成はAPI 呼び出しを介して実行することもコンパイル時に静的に実行することもできます。タイマの周期はティック単位で指定され、タイマの周期はカーネルのティック周期よりも細かくすることはできません。

スレッドの同期とスレッド間の通信または割り込みとスレッド間の通信は、組み込みシステムではもう1つの大きなトピックです。FreeRTOSはそのためのいくつかのメカニズムを提供しており、この処理を非常に簡単なものにしています。スレッド間通信の主な形態はキューです。これらは先入れ先出し(FIFO) バッファであり、メッセージをコピーすることによって新しいデータがキューの末尾に追加されます。つまり、メッセージ自体は参照されるだけでなく、完全にコピーされるのです。

セマフォと相互排他(ミューテックス)は、優先順位の反転を防ぎ同期の目的に役立ちます。別のスレッドをブロック解除したり値を更新したりするような単純なイベントの場合、バイナリまたはカウントセマフォおよび場合によってはキューへの直接タスク通知に代わる軽量で高速な方法があります。これらを使用するとスレッドは他のスレッドやISR と対話でき、他のオブジェクトを必要とせずメモリと時間を節約できます。

ここで説明する最後の同期機能は、メッセージキューとストリームバッファです。ストリームバッファを使用すると、開発者はスレッド間またはISR からスレッドにバイトストリームを渡すことができます。このバイトストリームは任意の長さを持たせることができ、一度に任意の数のバイトを読み書きできます。一方、メッセージバッファは可変長の離散メッセージを渡すことを可能にします。長さは問題ではありませんが、例えば送信者によってメッセージバッファに書き込まれた32バイトのメッセージは、受信者によって32バイトのメッセージとして読み取られる必要があります。個々のバイトとして読み出すことはできません。

最後に、FreeRTOS には、異なるイベントのシーケンスを表示する組み込みトレース機能と、オーバーフローなどのスタック関連の問題を検出する可能性の2つの非常に重要な機能も含まれています。どちらも、アプリケーションのトラブルシューティングに非常に役立ちます。

2.5.3 Microsoft Azure® RTOS ThreadX®の主な特長

ThreadX®は、MicrosoftAzure®RTOSの一部です。 このリアルタイムオペレーティングシステム（RTOS）は、もともとExpress Logicによって開発されたもので、ハイエンドでグラフィックが豊富なアプリケーションだけでなく、メモリが制限されるような組み込みシステムを対象としています。小さなフラッシュメモリ、小さなRAM要件、短いコンテキストスイッチング時間が特長です。

マルチスレッドRTOSとしてThreadXは複数の高度なスケジューリングアルゴリズムを使用し、リアルタイムのイベントトレース、メッセージパッシング、割り込み管理、およびその他の多くのサービスを提供します。コンシューマ、医療用電子機器、および産業用自動化市場で100億を超える実績のある信頼性の記録を持ち、多くの重要な安全基準および品質基準を満たしています。

提供されるその他の高度な機能には、たとえば、picokernel™アーキテクチャ、Preemption-Threshold™スケジューリング、Event-Chaining™などの豊富なシステムサービスのセットがあります。 また、FileX®、GUIX™、TraceX®、NetX Duo™、LevelX™、USBX™などのMicrosoft Azure RTOSの他のコンポーネントとも統合されます。 RAMメモリのフットプリントは非常に小さく、設計者は非常に高速な実行と低いオーバヘッドのために最適化が容易で、マイクロコントローラのリソースを主にアプリケーションに使うことができます。

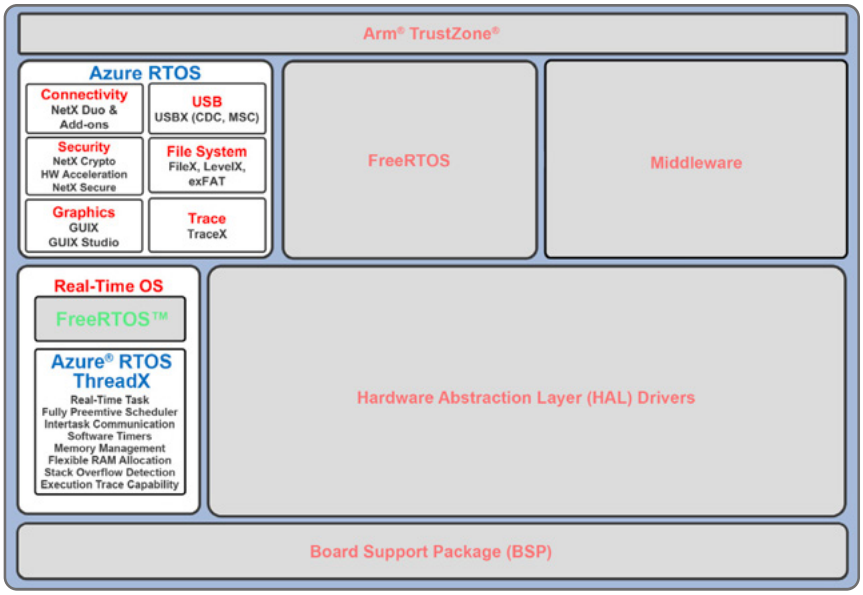

図2-8: ThreadX®には、プリエンプションしきい値マルチタスクスケジューラ、イベントチェーンと通信、メモリ管理、およびその他のシステムサービスが含まれています

強化されたリアルタイムスケジューリングアルゴリズムと効率的なマルチタスクルーチンは、ラウンドロビンスケジューリングとタイムスライシング、およびプリエンプティブとPreemption-Threshold™スケジューリングを提供します。スレッドに優先順位を付けるために、ソフトウェアエンジニアは最大1024の優先順位レベルを使用できますが、デフォルトの数は32です。

組み込みシステムのリソースは主に実行時間と使用可能なメモリによって制限されます。ThreadXは、それらを管理するためのいくつかの強力なオプションを提供します。 メモリ割り当ては高速である必要があるため、ThreadXは、固定サイズのメモリブロックのプールをいくつでも作成および管理する機能を提供します。プールは固定サイズであるため、メ

モリの断片化は問題になりませんが、メモリブロックのサイズはアプリケーションが要求する最大のメモリ容量と同じかそれ以上である必要があります。そうでない場合、割り当てはできません。ただし、そのシナリオに対して十分な大きなサイズにすることは、リクエストが異なるブロックサイズで受信されることになるので、メモリを浪費する可能性があります。その回避策は、異なるサイズのメモリブロックを含むいくつかのメモリブロックプールを作成することです。これらのブロックの高速割り当てと割り当て解除は、これらの割り当てが使用可能なリストの先頭で行われるため、この欠点を補います。 これにより、可能な限り最速のリンクリスト処理が提供され、実際のメモリブロックがキャッシュに保持される可能性があります。

ThreadXを使用すると、ブロックサイズが固定されたブロックプールだけでなく、標準のCヒープと同様に動作する複数のバイトプールを作成できます。 メモリは最初の方法に基づいて、バイト単位の目的のサイズでのみ割り当てられます。その欠点はCのヒープのようにバイトプールが簡単に断片化されてしまうことでやや非決定的な動作が発生することです。

RTOSによって管理されるその他のリソースはアプリケーションタイマであり、無制限の数のソフトウェアタイマを作成できます。 これらのアプリケーションタイマは、ワンショット、周期的、相対の3つの動作モードで使用できます。 ワンショットタイマはユーザ関数のみを呼び出します。タイマの期限が切れた後に1回、定期的なタイマが一定の間隔の後にユーザ関数を繰り返し呼び出します。相対タイマは、単一の連続的にインクリメントする32ビットティックカウンタです。すべてのタイマの有効期限は、ティックで指定されます。 たとえば1ティックは10ミリ秒に相当しますが、これはもちろん調整可能です。

スレッドの同期とタスク間の通信は組み込みシステムのもう1つの大きなトピックです。ThreadXはそのためのいくつかのメカニズムを提供し、この処理を楽にします。セマフォと相互排除（ミューテックス）は、優先順位の逆転を防ぎ、無制限の数のオブジェクトを作成できるようにするのに役立ちます。オプションの優先度継承と同様に、洗練されたコールバック関数とイベントチェーンも利用できます。 さらに、ThreadXはFIFOまたは優先順位のいずれかで中断でき、優先度の低いスレッドが処理時間を費やす問題を回避できます。

イベントフラグはもう1つのスレッド同期機能です。 イベントフラググループは32ビットで構成され、各ビットは異なる論理イベントを表し、スレッドはこれらのビットのサブセットを待つことができます。 イベントフラグはイベントチェーンもサポートします。 セマフォと同様に無制限の数のオブジェクトを作成でき、フラグの実行時のパフォーマンスに関する情報を利用できます。

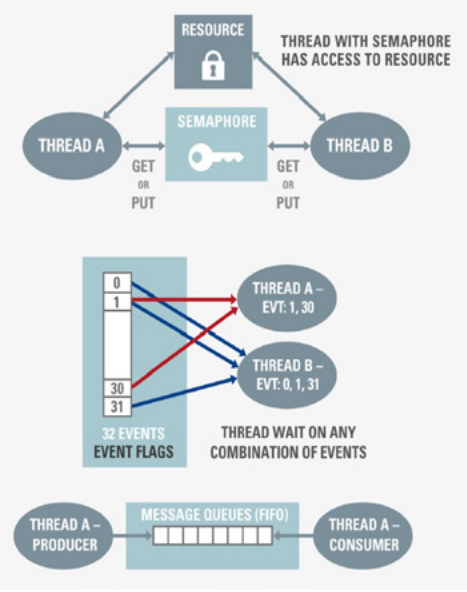

図2-9: ThreadX®の同期と通信機能の特長

ここで紹介する最後の同期機能はメッセージキューです。 ThreadXはタスク間通信にProducer/Consumerモデルを使用して、1〜16個の4バイトワードのメッセージサイズをサポートします。メールボックスはサイズ1のメッセージキューの特殊なケースです。アプリケーションスレッドはさらなる通知のためにコールバック関数を登録できます。イベントフラグと同様に実行時のパフォーマンスに関する情報を取得することができます。

またデバッグの目的で重要な機能が実装されています。それはTraceX®でサポートされている組み込みのイベントトレース機能です。これにより、さまざまなイベントのシーケンスを表示できます。 TraceXはスタンドアロンのWindows®アプリケーションですが、プログラマはe² studio内から起動できます。

そして最後に、ThreadXは、Misra-C：2004およびMisra-C2012のすべての「required」および「mandatory」ルールに準拠し、EAL4 + Common Criteriaセキュリティ認定を受けており、次の標準に対して事前認定されています。

- IEC 61508 SIL 4
- IEC 62304 Class C
- UL/IEC 60730-1 H
- CSA E60730-1 H
- ISO 26262 ASIL D
- UL/IEC 60335-1 R
- UL 1998
- EN 50128 SW-SIL 4

本章で学んで会得するポイント:

- フレキシブルソフトウェアパッケージ (FSP) はレイヤに組み込まれており、シンプルで標準化されたAPI 呼び出しを介してすべての機能にアクセスできます
- ミドルウェアは豊富な接続性とセキュリティ、およびグラフィックと静電容量式タッチ機能を提供し、コードをゼロから記述する必要はありません
- FreeRTOS™とThreadX®はパワフルで使いやすいオペレーティングシステムで、メモリフットプリントが小さくマイクロコントローラのほとんどのリソースをアプリケーションで使用できます

3　フレキシブルソフトウェアパッケージのAPI の概要

この章で学ぶこと:

- FSPコードの構文、スタイル、命名規則
- アプリケーションプログラミングインタフェースとその使用方法の詳細

ルネサスの設計者は、RAファミリのマイクロコントローラ用のフレキシブルソフトウェアパッケージ（FSP）を開発する際、ユーザにとってより使いやすいものを目指しました。アプリケーションプログラミングインタフェース（API）のアーキテクチャは非常に単純で包括的であり、FSPの複雑さをカプセル化しています。よって開発者がさまざまな機能を完全に制御できるようにFSPは非常に優れた機能を有しており、加えて非常に簡単に使用できるのが特長です。USB転送のような複雑なタスクをプログラミングすることさえ、数行のコードに縮小されマニュアルを読まずとも、または特定のマイクロコントローラ周辺機器のレジスタセットのそれぞれ特性を調査する必要なしに実現することができます。これは、プログラミングとテストにかなりの時間を要する低レベルのコードを書く代わりに、自分のアプリケーションのフィーチャーセットに集中できるので、開発者にとって大きなメリットとなります。ではAPIの詳細を見てみましょう。

3.1　API の概要

アプリケーションは、必要な機能がどのレイヤに存在していても、直感的でシンプルで統一されたAPI 呼び出しによって、フレキシブルソフトウェアパッケージ のすべての機能にアクセスできます。これにより、たとえばマイクロコントローラの別の周辺機器を特定のタスクに使用する必要がある場合など、理解しやすく保守が簡単で、移植が簡単なコードを非常に簡単に記述できます。 こういったことは最近のデザインにとっては良く求められていることです。

この例では、オンチップのSPI ペリフェラルからSCI ポートのSPI 機能への変更は、FSP が使用されている場合、簡単な変更1つのみであることを意味します。FSP コンフィギュレータでは、スタックタブのSPI (r_spi) のSPI ドライバをSPI(r_sci_spi) のSPI ドライバに置き換えます。どちらのドライバもg_spi0 と同じインスタンスを使用するため、アプリケーションコードに変更はありません。

FSP の各層のアーキテクチャを図3-1 に示します。下部はRA MCU で、ボードサポートパッケージ(BSP) が上部に配置されています。BSP は、MCU をメインアプリケーションにリセットし、上記ソフトウェアに他のサービスを提供する責任を負います。次のレイヤは、開発者がマイクロコントローラのレジスタセットを直接処理する必要がないようにするハードウェア抽象化レイヤ(HAL) で、RA ファミリ全体にわたってHAL 上のソフトウェアをより移植性の高いものにします。

HAL の上には、ミドルウェアスタックとプロトコルレイヤとリアルタイムオペレーティングシステムがあります。ミドルウェア レイヤは、接続オプション、グラフィック処理機能、およびセキュリティ機能をユーザアプリケーションに提供します。最後に、最上位にはユーザプログラムが存在し、統合されたAPI インタフェースを介して以下のレイヤを呼び出します。FSPの層と個々の部分に関する詳細は、第2章で参照可能です。

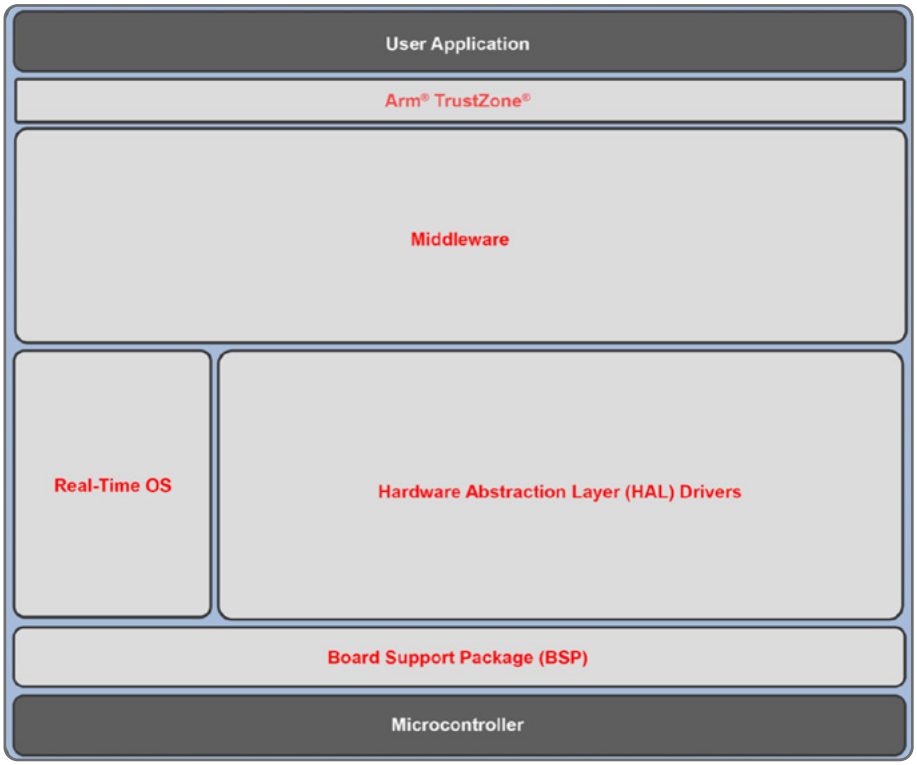

図3-1: フレキシブルソフトウェアパッケージのさまざまなレイヤ

それぞれの異なるレイヤーは、API 呼び出しを介して直接、またはスタック方式でアクセスされます。たとえば、USB 転送を必要とするアプリケーションは、USB HCDC ドライバをコールします。このドライバは、基本的なUSB ドライバを使用します。基本的な USB ドライバは、DMAC (ダイレクトメモリアクセスコントローラ) モジュールの2 つのインスタンス(1 つは送信用、もう1 つは受信用) を使用します(図3-2 を参照)。

もちろん、エンドアプリケーションはHALドライバとボードサポートパッケージを直接呼び出すこともできますが、ミドルウェアスタックとプロトコルレイヤを使用する方がはるかに簡単です。この場合、基本となる部分の詳細な知識は必要ありません。

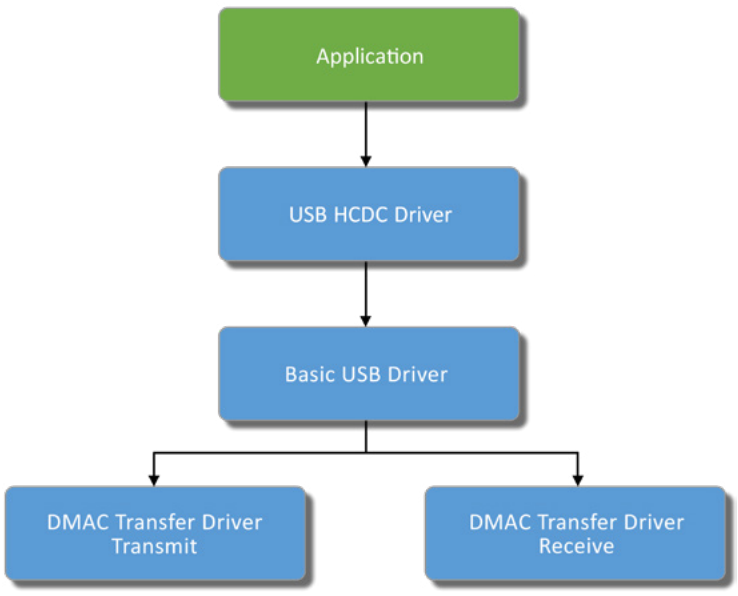

図3-2: 簡単に使用できるようにモジュールを積み重ねることができます

3.2 API 構文

API の詳細を深く調べる前に、異なるAPI とファイルの命名規則について簡単に確認しておく必要があります。一旦理解すると、アプリケーションのプログラミングが容易になるだけでなく、数か月後に別のプログラマのコードまたは自分のコードを理解できるようになります。後者は、われわれが過小評価すべきではなく、常に行うべき努力です。フレキシブルソフトウェアパッケージ (FSP) によって提供される明確な構造は、何よりも役立ちます。

一般に、内部関数は"NounNounVerb" の命名規則に従います(例: communicationAbort())。データは関数の出力引数に戻され、最初のパラメーターは常にその制御構造を指すポインタになります。この構造体はモジュールインスタンスのメモリを保持するため、ユーザアプリケーションはどこに配置するかを柔軟に決定できます。

FSPで一般的に使用されるプレフィックスのリストを以下に示します:

- **R_BSP_xxx:** 共通BSP 関数の接頭辞(例: R_BSP_VersionGet())。
- **BSP_xxx:** BSP マクロのプレフィックス(BSP_IO_LEVEL_LOW など)。
- **FSP_xxx:** 一般的なFSP のプレフィックスは、主にエラーコード(FSP_ERR_INVALID_ARGUMENT など) とバージョン情報(FSP_VERSION_BUILD など) を定義します。
- **g_<interface>_on_<instance>:** インタフェースAPI 関数をモジュールが提供する関数に結び付けるインスタンスの定数グローバル構造体の名前。たとえば、HAL レイヤ関数の場合はg_spi_on_spi です。
- **r_<interface>_api.h:** インタフェースモジュールヘッダファイルの名前(r_spi_api.h など)。
- **R_<MODULE>_<Function>:** FSP ドライバAPI の名前(例: R_SPI_WriteRead())。
- **RM_<MODULE>_<Function>:** ミドルウェア関数の名前(例: RM_BLE_ABS_Open())。

オペレーティングシステムも命名規則に従います。 FreeRTOS™ではファイルスコープの静的関数には接頭辞prvが付いています。 API関数は、スキーム<return type> <Filename> <Operation>に準拠しています。ここで、戻り値の型は変数の規則に従います。関数が無効な場合は、接頭辞「v」が付きます。 [ファイル]フィールドには、API関数が定義されているファイルの名前が含まれ、[操作]フィールドには、UpperCamelCase構文で実行されている操作の名前が含まれます。 例としては、tasks.cで定義されてRTOSスケジューラーを開始するvoid関数であるvTaskStartScheduler（void）や、非標準タイプunsigned BaseType_tのデータを返す関数であるuxQueueMessagesWaiting（）がファイルキューで定義されておっり、キューで待機しているメッセージの数を返します。

同様に、マクロにも有効です。これらのマクロには、小文字で定義されているファイルの接頭辞が付けられ、その後にすべて大文字の名前が付けられます。ここでは、tasks.h ファイルで定義されているマクロtaskWAITING_NOTIFICATION の例を示します。

FreeRTOS の命名規則の詳細については、FreeRTOS Coding Standard, Testing and Style Guide on the Internet (**https://www.freertos.org/FreeRTOS-Coding-Standard-and-Style-Guide.html**)) を参照してください。

MicrosoftAzureRTOSの命名規則も似ています。 これらは次のルールで作成されます：<ID>_<名詞>_<動詞>となります。IDはモジュール、名詞は問題のオブジェクト（タイマ、セマフォなど）、動詞は実行するアクション（作成、閉じる、受け取る、など）。たとえば、ThreadX®でメッセージキューを作成するケースはtx_queue_create（）となります。

以下は、さまざまなAzure®RTOSモジュールのいくつかのIDの要約です。

- **fx：**FileX®関連の機能、例：fx_directory_create ()
- **gx：** GUIX™関連の機能 例：gx_display_create ()
- **lx：** LevelX™関連の機能 例：lx_nand_flash_open ()
- **nxd：** NetX DuoTM関連の機能 例：nxd_ipv6_enable ()
- **tx：** ThreadX®関連の機能 例：tx_thread_create ()
- **ux：** USBXTM関連の機能 例：ux_device_stack_initialize ()

API の形式的な構文を理解するだけでなく、いくつかの定義に従うことも非常に役立ちます。多くの場合、私たちの誰もが多少異なる意味で用語を解釈していて、混乱を引き起こしています。したがって、本書とFSPで使われている用語のリストを次に示します:

- **モジュール:** モジュールは、周辺機器ドライバ、純粋なソフトウェア、またはその間にあるもので、FSP の構成要素です。モジュールは、通常、独立したユニットですが、他のモジュールに依存する可能性があります。アプリケーションは、複数のモジュールを組み合わせてユーザに必要な機能を提供することで構築できます。

- **モジュールインスタンス:** モジュールの単一および独立したインスタンス化(設定)。たとえば、r_dmac モジュールの2 つのインスタンスを使用して、ポートとポートとの間でデータを転送する必要がある場合があります。

- **インタフェース:** インタフェースには、同様の機能を持つモジュールで共有できるAPI 定義が含まれています。これらは、モジュールが共通の機能を提供する方法です。これらを介して、同じインタフェースに準拠するモジュールを同じ意味で使用できます。インタフェースは、2つのモジュール間の契約と考えてください。ここで、モジュールは、契約で合意された情報を使用して連携することに同意します。インタフェースは定義のみであり、コードサイズには追加されません。

- **インスタンス:** インタフェースは提供される機能を指定しますが、インスタンスは実際にそれらを実装します。各インスタンスは、インタフェースからの列挙型、データ構造、API プロトタイプを使用して、特定のインタフェースに結び付けられます。これにより、アプリケーションは必要なときにインスタンスをスワップアウトできます。

- **ドライバ:** ドライバは、RA ファミリMCU のレジスタを直接変更する特定の種類のモジュールです。

- **スタック:** FSP アーキテクチャは、モジュールが連携してスタックを形成できるように設計されています。スタックは、最上位モジュールとそのすべての依存関係で構成されます。

- **アプリケーション:** ルネサスが提供するサンプルコードが使用されている場合でも、ユーザが所有し維持するコードです。

- **コールバック関数:** USB が何らかのデータを受信した場合など、イベントが発生したときに呼び出されます。これらはアプリケーションの一部であり、割り込みに使用する場合は、割り込みサービスルーチン内で実行され、他の割り込みが実行されないようにできるだけ短くする必要があります。

3.3 API 定数、変数、およびその他のトピック

前述のように、モジュールはフレキシブルソフトウェアパッケージの構成要素です。 これらは異なるタスクを実行できますが、図3-3に示すように、すべてのモジュールは、機能を上向きに提供し、下向きに機能を要求するという基本概念を共有しています。 したがって、FSPを使用する最も単純なアプリケーションは、ユーザコードが上にある1つのモジュールで構成されます。

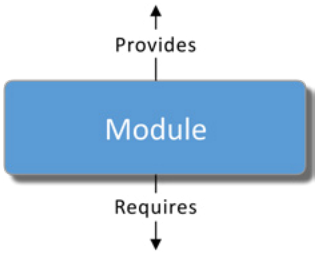

図3-3: モジュールは、機能を上向きに提供し、下向きに機能を要求するという基本概念を共有しています

モジュールは互いに重ね合わせることができ、図3-2に示すようにFSPスタックを構築します。スタック・プロセスは、あるモジュールが提供するものと別のモジュールが要求するものを照合することによって実行されます。Block Mediaの実装には、たとえばSD/MMC ドライバが必要です。このドライバには、データ転送インタフェースが必要です。データ転送インタフェースはデータトランスファコントローラ(DTC)ドライバモジュールによって提供できます。そのコードは意図的にBlock Media モジュールには含まれておらず、基盤となるDTC モジュールを、UART などによって他のモジュールでも再利用することができます。

モジュールをスタックする機能は、アプリケーションが必要とする柔軟性を確保するため、大きなメリットを提供します。モジュールが他のユーザも使用している場合、異なるユーザデザイン間での移植性は困難になります。設計されたFSPアーキテクチャは、同じインタフェースに準拠するモジュールが同じように使用できるので、FSPインタフェースの使用を通してモジュールを入れ替える(例えば、SPIドライバとUARTドライバを交換する)可能性を提供します。つまり、すべてのモジュールは契約で合意された情報を共有し協力することに同意します。

RA ファミリのMCU では、異なるペリフェラル間で機能が重複する場合があります。たとえば、IIC ペリフェラルまたは IIC モードの SCI ペリフェラルを使用して IIC 通信を行うことができ、両方のペリフェラルが異なる機能セットを提供します。インタフェースは、ほとんどのユーザが期待する一般的な機能を提供する方法で組み込まれていますが、より高度な機能の一部は省略されています。ただし、ほとんどの場合、インタフェース拡張を介して引き続き使用することができます。

各FSPインタフェースには、少なくとも3つのデータ構造が含まれています。一つ目は<interface> _ctrl_tというモジュール依存の構造で、これはモジュールを使用する際にユニークな識別子として扱われます。2つ目は、open () 呼び出し中の初期構成用のモジュールへの入力としての<interface> _cfg_t *という名前の構成構造体で、3つ目は、制御構造体へのポインタ、制御構造体へのポインタの3つのポインタで構成されるインスタンス構造体です。この後者の構造体の名前は、g_ <interface> _on_ <instance>として標準化されています。 g_spi_on_spiと構造自体は、インスタンスのヘッダーファイルのextern宣言を介して使用できます。 単純なドライバの場合、これらの構造を組み合わせることができますが、この方法で作成するとインタフェースの機能が拡張されます。

すべてのインタフェースにはサポートされているすべての関数の関数ポインタを含むAPI 構造が含まれています。たとえば、DAC (digital-to-analog converter) の構造体dac_api_t には、open、close、write、start、stop、versionGet などの関数へのポインタが含まれています。API 構造は、同じインタフェースのインスタンスである他のモジュールに対してモジュールをスワップインおよびスワップアウトできるようにするものです。

モジュールはコールバック関数を介してイベントが発生すると、ユーザアプリケーションにアラートを送信します。このようなイベントの例としては、UART チャネルを介して受信されるバイトなどがあります。ユーザアプリケーションが割り込みに対応できるようにするにはコールバックも必要です。これらは割り込みサービスルーチン内から呼び出されるので、できるだけ短くする必要があります。短くしないとシステムのリアルタイムパフォーマンスに悪影響を及ぼす可能性があります。

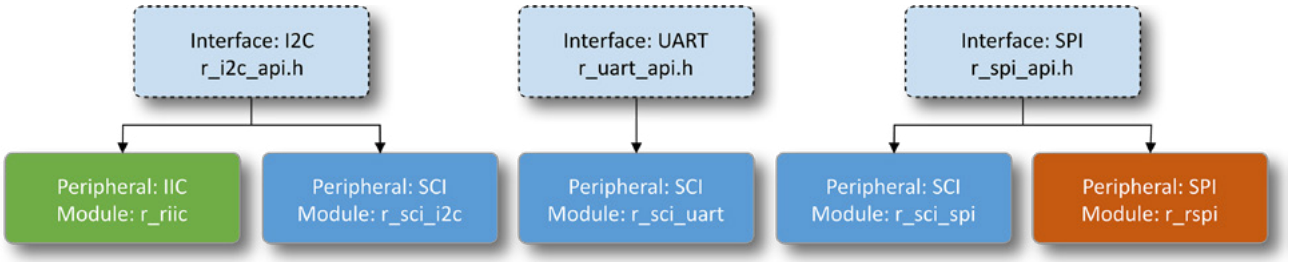

図3-4: RA ファミリのマイクロコントローラでは、一部のペリフェラルはインタフェースとインスタンスの間で1 対1 のマッピングを持ち、その他のペリフェラルは1 対1 のマッピングを持ちます

一方、提供される機能はインタフェースによって決まり、インスタンスによって実際に実装されます。各インスタンスは特定のインタフェースに結び付けられており、インタフェースからの列挙、データ構造、API プロトタイプを使用します。これにより、インタフェースを使用するアプリケーションが必要なときにインスタンスをスワップアウトできるようになり、コードや使用されているペリフェラルに変更が必要になった場合に多くの時間を節約することができます。RA ファミリMCU では、IIC などの一部のペリフェラルに1対1のマッピング(IIC インタフェースにのみマップされます)があり、SCI などの他のアプリケーションには1対多のマッピング(IIC、UART、SPI の3つのインタフェースが実装されます)があります。このマッピングのグラフィカル表現については、図3-4 を参照してください。

3.4 API の使用法

すべての説明を終えたので、フレキシブルソフトウェアパッケージ (FSP) を使った開発がどれほど簡単か実際に確認してみましょう。そのためには、単純なSPIを例にして、APIの使い方や、各項目の情報の入手先を説明します。PC で実行する場合は、第5 章を参照後にこのセクションを参照してみてください。

FSP モジュールを使用するための最初の手順は、必要な機能に適したインタフェースを常に選択することです。この例では、SCI(Serial Communications Interface)のSPI インターフェースブロックを介して通信します。このブロックは、MCU のすべてのシリーズのRA ファミリで使用できます。したがって、e² studio 内では、FSP コンフィギュレータのr_sci_spi 上のSPI ドライバを選択します。これは、最初にコンフィギュレータのStacks タブの左側にあるHAL/Common スレッドをハイライト表示にし、次にHAL/Common Stacks の右側にあるr_sci_spi でNew Stack → Driver Connectivity → SPI (r_sci_spi) を選択して行います。追加されると、追加の必須エントリは、ユーザが機能を選択または決定する必要があるレベルまでは自動的に追加されます。SPIの場合、r_dtc (データトランスファコントローラ) モジュールの各g_transfer0Transferおよびg_transfer1Transferドライバのインスタンスが追加されます。 前者はSPIポートへのデータ転送を処理し、デフォルトでSCI0のTXI割り込みに接続されますが、後者はRXI割り込みに関連付けられシリアルポートからデータを取得します。

ユーザは他には何もする必要はありません。上の例の場合、受信、送信、およびエラー割り込みはFSPコンフィグレータによって自動的に許可されます。必要に応じて、g_spi0 と呼ばれるモジュールインスタンス(デフォルトでは、数は同じタイプのドライバが追加される順序に依存します) に、必要に応じてユーザ名を付けることができます(例: my_spi)。これにより、抽象化レイヤーがもう1つ追加されます。これにより、ソフトウェアデザイナーが名前を適切に選択すると、コードの移植性とメンテナンス性が向上し、読みやすくなることは言うまでもありません。

g_spi0 SPI(r_sci_spi) モジュール上のProperties ビューは、SPI ポートのビットレート、ビット順序などのパラメータを設定する場所になります。グラフィカルユーザインタフェースですでに行われているすべての設定では、正しい値でポートを手動で初期化する必要はありません。これはすべてFSPコンフィグレータによって行われます。

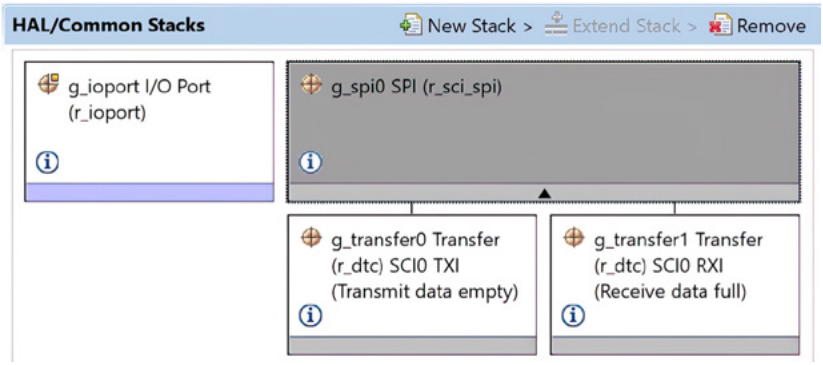

図3-5: e² studioのSPI スタック

現時点では、HAL/Common Stacks ウィンドウの他の項目を気にする必要はありません。これは、I/O ポート用のボードサポートパッケージ(BSP)で必要とされる基本的なドライバです。このドライバは、コンフィギュレータによって自動的に追加され、起動時に初期化されます。

残されているのは、SCI-SPI のI/O ピンがFSP コンフィギュレータのPins タブで適切に設定されていて、このペリフェラルを介して実際にデータを送受信できるかどうかを確認するためのものです。ビューの左側にあるペリフェラルリストを展開すると、使用可能なインタフェースが表示されます。Connectivity:SCI ツリーをさらに展開すると、SCIO エントリを選択でき、このポートのPin Configuration が表示されます。Operation Mode でSimple SPI を選択すると、TXD_MOSI、RXD_MISO、SCK、およびCTS_RTS_SS ドロップダウンリストボックスが有効になり、使用されるピンを実際のハードウェアと一致させることができます。

<ctrl>+<s> を使用して変更を保存し、Generate Project Content アイコンをクリックすると、必要なソースファイルとヘッダファイルが作成され、p_ctrl およびp_cfg という名前のコントロール構造と設定構造がそれぞれspi_ctrl_t* およびspi_cfg_t* タイプになり、g_spi0 API インタフェースによって提供されるポインタを使用してアクセスできるアプリケーション固有のヘッダファイルに配置されます。

セットアップが完了し、FSP ファイルが作成されると、インタフェースとの対話が簡単になります。SPI の例の場合、SPI ペリフェラルは、次の関数を呼び出すことによってオープンされ、設定されます:

```
err=g_spi0.p_api->open(g_spi0.p_ctrl,g_spi0.p_cfg);
```

g_spi0 は、Properties ビューでインスタンスを設定するときに指定された名前で、p_api、p_ctrl、p_cfg はコンフィギュレータによって生成された構造体へのポインタです。同様に、SPI に書き込むには、以下の関数呼び出しが必要です:

```
err=g_spi0.p_api->write(g_spi0.p_ctrl,tx_buffer,length,bit_width);
```

ここで、tx_buffer は、SPI 経由で送信されるデータを保持するユーザ定義の配列へのポインタで、32 ビット符号なし整数および bit_width として書き込まれるユニットの数を長くします。これはspi_bit_width_t 型で、ユニットのサイズを保持します。正確な構文は、RA Flexible Software Package Documentation のインターフェースリファレンスの章から、またはe² studio のスマートドキュメント機能を使用して抽出できます。

上記の例のようにインタフェース関数を使用する代わりに、ダイレクトインプリメンテーション関数R_<MODULE>_<Function>を使用することもできます。その場合、SPI ペリフェラルを開いて設定するためのコードは次のようになります:

```
err = R _ SPI _ OPEN(&g _ spi _ ctrl, &g _ spi0 _ cfg);
```

g_spi0_ctrl とg_spi0_cfg はFSP コンフィギュレータによって作成されたデータ構造体です。同様に、SPI ポートへの書き込みには、この関数呼び出しが必要です:

```
err = R _ SPI _ WRITE(&g _ spi0 _ ctrl, tx _ buffer, length, bit _ width);
```

tx_buffer は再びポート上に書き込まれるデータを保持する配列へのポインタで、length はユニットの数、bit_width はユニットのサイズを保持します。bit_width の代わりに、FSP によって提供されるマクロSPI_BIT_WIDTH_8_BITS も使用できます。

これら2 つの短い例は、フレキシブルソフトウェアパッケージの機能の非常に優れたデモンストレーションです。ポートを設定しデータの配列を送信するには、各ケースで2行のコードのみを記述する必要があります。他のすべてはコンフィギュレータおよびFSPのドライバによってすでに作成され実行されています。マイクロコントローラのユーザーズマニュアルを読んで、関連するさまざまなレジスタについて学ぶ必要はなく、アプリケーションの開発中に多くの時間を節約できると私は考えています。

本章で学んで会得するポイント:

- FSPは明確な命名規則に従い、コーディングを簡単にします
- FSPモジュールは、アプリケーションに豊富な機能を提供するために簡単にスタックできます
- API の構文は簡単で直感的です
- インタフェース関数を使用する代わりに、インプリメンテーション関数も使用できます

4　ツールチェーンの入手と実行

この章で学ぶこと:

- RA ファミリのツールチェーンの入手先とインストール方法
- e² studio を初めて起動し、新しいプロジェクトを作成する手順

本章では、ルネサスRA マイコンファミリのソフトウェアプロジェクトに必要な2つのツール、GCC Arm®組み込みツールチェーンを備えたe² studio 統合開発環境(IDE) およびフレキシブルソフトウェアパッケージ(FSP) をダウンロードしてインストールします。

次の項では、そのために必要なすべてのステップの概要を説明します。インストールにあたって何をどこにインストールするかを選択すると、インストーラが必要なすべてのことを処理するため、ツールのインストールに手間がかからないことがわかります。最後にインストールが完了したら、短いサニティチェックを実行して、すべてが期待どおりに機能していることを確認します。

4.1　e² studio とFSP のダウンロードとインストール

e² studioには、RA ファミリのマイクロコントローラ用のプロジェクトを作成、コンパイル、デバッグするために必要なすべてのツールが含まれています。一般的なEclipse™ IDE に基づいていますが、ルネサスではいくつかのソリューション指向のコンポーネントとプラグインが追加され、さらに強力になりました。これは特に新しいプロジェクトを生成する簡単な方法を提供し、ピン構成やソフトウェアスタックの追加など、さまざまなハードウェアおよびソフトウェア機能へのグラフィカルなアクセスを、ユーザーズマニュアルを深く調べることなく簡単に可能にするコンフィギュレータに特に当てはまります。これらのコンフィギュレータは、必要なすべての設定と初期化コードを自動的に作成し、設計時に既に問題のある組み合わせを検出するためのエラーチェック要素を含み、アプリケーション自体に値を追加しないコードの書き込みやデバッグに費やされる時間を節約します。少なくとも、組み込みシステム用のソフトウェアを初めて書いたときに、このようなツールが利用可能であれば、私はもっと幸せだったでしょう。

次は、e² studio とFSP をWindows®ワークステーションにダウンロードしてインストールするさまざまなフェーズについて説明します。FSP/e² studioツールチェーンを初めて使用するユーザを対象としています。e² studio の現行リリースとGCC Arm® Embedded Tools をすでにインストールしている場合は、FSP パックを個別にダウンロードするだけで処理を短縮できます。これらは、FSP のGitHub ページ(**https://github.com/renesas/fsp/releases**)) のAssets セクションでアクセスできます。FSP_Packs_<version>.zip という名前のzip ファイルとFSP_Packs_<version>.exe という名前のインストーラの2つのバージョンを使用できます。

Arm 用IAR Embedded Workbench®またはKeil® MDK Microcontroller Development Kit を使用する場合は、FSP 用のコンフィギュレータを利用することもできます。RA Smart Configurator (RASC) はこれらのツールチェーンと統合され、同じ機能を提供します。GitHub のAssets セクションでも利用できます。

4.1.1 インストーラのダウンロード

最初の手順は、GitHub®(**https://github.com/renesas/fsp/releases**) のFSP ページのアセットセクションで、e² studio インストーラ を使用するFSP の最新版を見つけることです。setup_fsp_v3_7_0_e2s_v2022-04.exe という名前のインストーラの実行可能ファイル(setup_fsp_v2_1_0_e2s_v2020-10.exe など) があります。このファイルをハードディスクにダウンロードします。ダウンロードが進行中で、サイズが1.3GB を超えるかなり大きなファイルですが、ページの他の項目を確認するには時間がかかります。そこから、ZIPファイルとしてFSP用のHTML形式の最新文書をロードしたり、README.mdファイルのリリースノートを読んだりすることもできます。

4.1.2 ツールチェーンのインストール

e² studio インストーラによるFSP のダウンロードが完了したら、ハードディスク上のファイルを見つけてダブルクリックし、セットアップ処理を開始します。インストールファイルの解凍には時間がかかります。すべてのユーザまたは現在のユーザの開発ツールをインストールする場合は、最初に表示される画面がクエリになります。最適なインストールタイプを選択します。次に、Windows®ユーザアカウント制御ダイアログボックスが表示され、「Do you want to allow this app to make changes to your device?」と尋ねられます。この質問に「Yes」と答えてください。

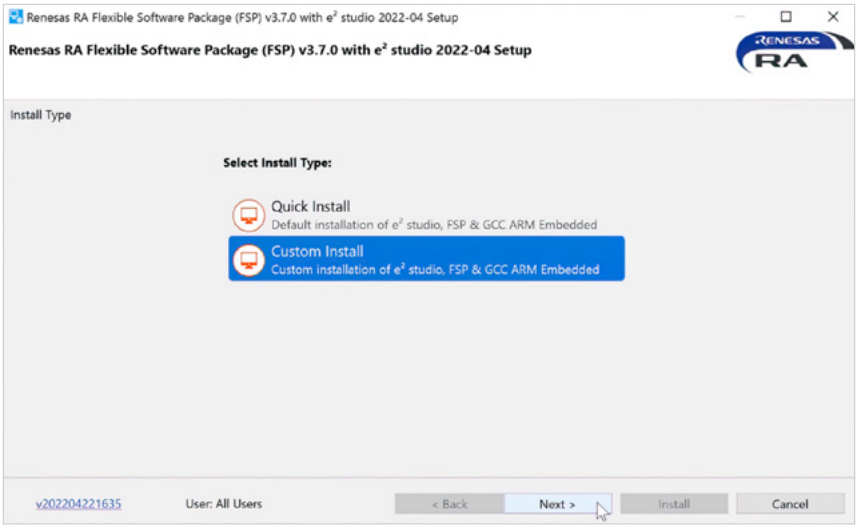

図4-1: カスタム インストールは、追加のツールや機能をインストールできるため、最適な方法です

次の画面では、インストーラがインストールタイプを尋ねます(図4-1 を参照)。クイックインストールには、e² studioをインストールするだけの2つのオプション、フレキシブルソフトウェアパッケージとGCC Arm®組み込みコード生成ツール、およびデフォルト設定のユーザーズマニュアルがあり、ユーザの操作を最小限に抑えながら実行されます。またはカスタムインストールがあって、これによりIDE とフレキシブルソフトウェアパッケージ をインストールできるだけでなく、FreeRTOS™のデバッグサポート、GIT バージョン制御システムの統合、または静電容量式タッチおよびBluetooth®用のQE ツールなどのオプションコンポーネントもインストールできます。可能な限り高速にインストールしたい場合は、クイックインストールを使用します。ただし、追加の機能をインストールする場合や、その他の機能を使用できるかどうかに興味がある場合は、カスタムインストールが適切な選択です。また一旦決定した場合でも、後でインストーラを再実行することで、いつでも変更できます。

この検証の目的で、Custom Installを選択し、[Next]をクリックします。インストール・ルーチンは、インストール・ディレクトリの準備ができていること、および必要なすべてのコンポーネントが使用可能であることを検査します。ここでは、ソフトウェアがインストールされるフォルダを変更することもできます。デフォルトのパスはC:\Renesas\RA\ e2studio_v<e2 studio version>_fsp_v<fsp_version> です(例: C:\Renesas\RA\e2studio_v2022-04_fsp_v3.7.0.)。必要に応じて、任意のパスに変更します。次に、[Next]をクリックして[Results]画面を閉じます。

[Extra Features]ダイアログが表示されたら、表示されたリストを参照し、インストールに追加する機能を選択します。選択に問題がなければ、[Next] をクリックして移動します。

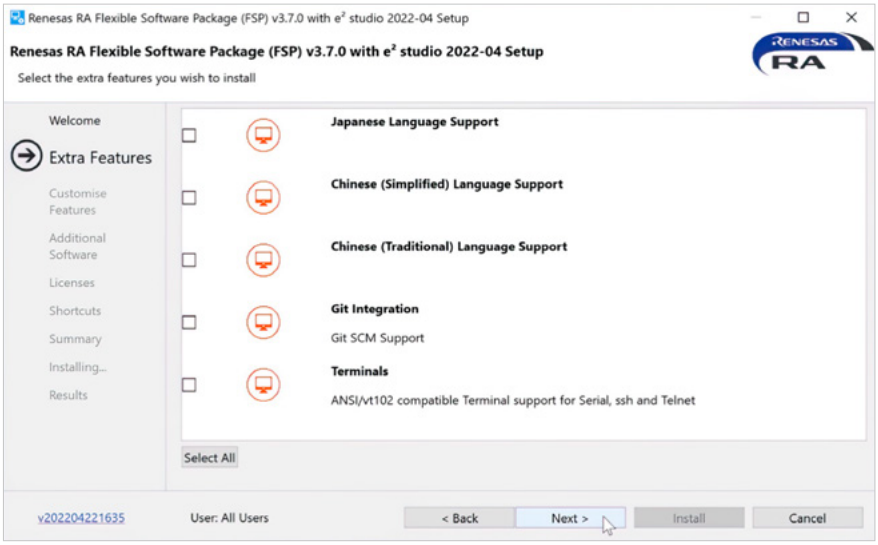

図4-2:この画面では、追加機能をインストールできます

Customize Features という名前の別の画面が表示されます(図4-3 を参照)。ここで、EclipseのGitの統合や、シリアル、ssh、Telnetのターミナルサポートなど、インストールにこういったコンポーネントを追加できます。完了した場合は、[Next]を再度クリックします。

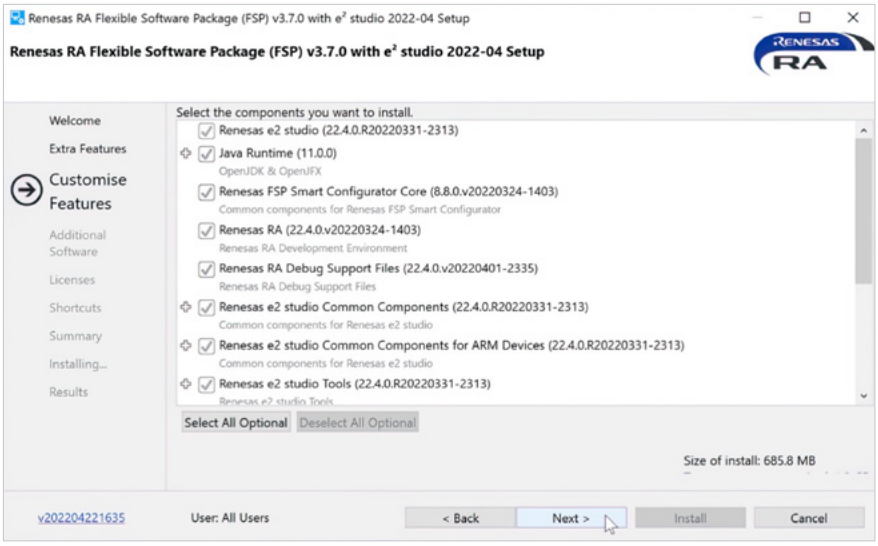

図4-3: このウィンドウでは、さらにコンポーネントをインストールできます

Additional Software という名前の次の画面に、Renesas RA とRenesas QE という2 つのタブが表示されます(図4-4 を参照)。Renesas RA タブで、Toolchains の下でGNU ARM Embedded が選択されていることを確認します。これは、コードのコンパイルとリンクに必要なものです。2 番目のタブで、インストールに使用するQE Tools を選択します。これらの詳細については、第1 章を参照してください。準備ができたら、すぐに[Next]をクリックします。

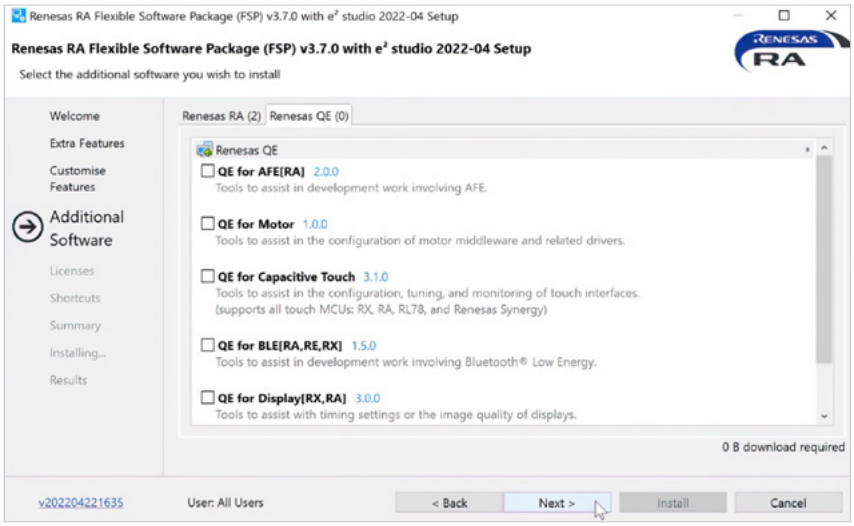

図4-4: インストールするQE Tools を選択します

これで、インストールするコンポーネントの各種ソフトウェア契約が表示されます。それぞれをよく読み、「I accept the terms of the Software Agreements」の横にあるチェックボックスをオンにして、[Next]を再度クリックします。表示されるショートカット画面で、提案された[Start]メニューグループに同意するか、より意味のあるものに変更するか、チェックボックスをオフにします。[Next]をクリックして次のウィンドウに移動すると、インストールするソフトウェアの概要が表示されます。これを参照し、[Install]をクリックしてインストールを開始します。

インストール中に、「Would you like to install this device software?」 (図4-5参照)を確認するWindowsセキュリティダイアログが表示されますので、「Always trust software from “Renesas Electronics Corporation”」の横にあるチェックボックスをオンにして、[Install]をクリックします。

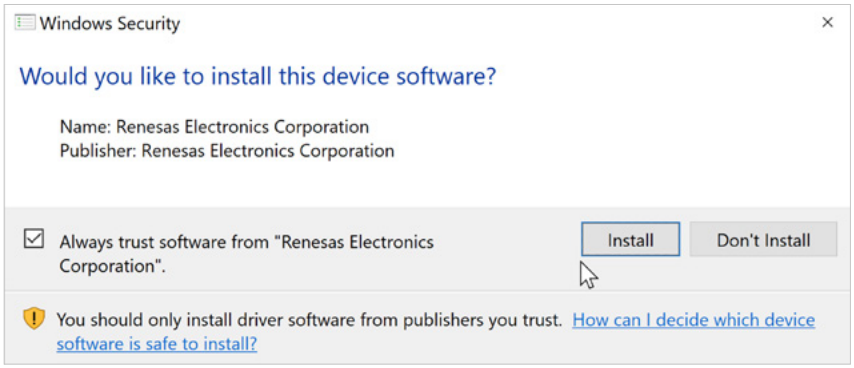

図4–5: デバッグを開始するためにインストールを許可してください

インストールが完了すると、結果ウィンドウが表示されます。すべてのチェックボックスをオフにし、FSP およびGCC Arm Embedded の詳細と、FSP ユーザーズマニュアルへのリンクを提供するリンクを参照してください。最後に、OK をクリックしてインストーラを閉じます。これでインストールが完了し、開発環境の最初のテストの準備が整いました。

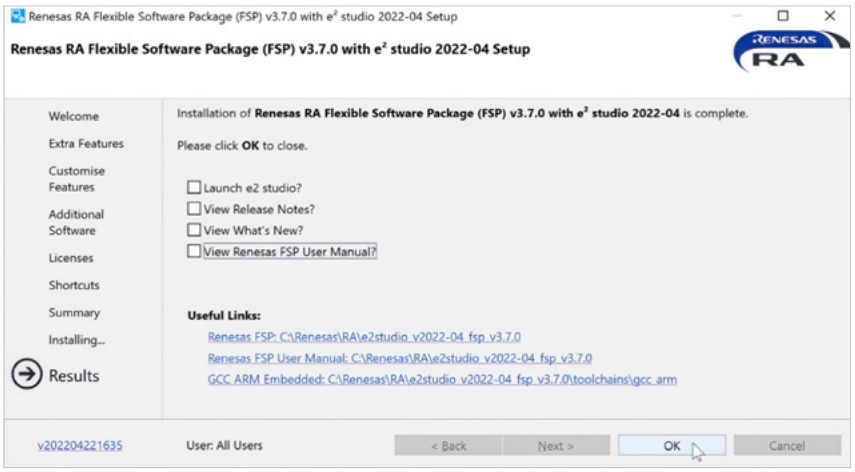

図4-6: インストールが完了すると、このウィンドウが表示されます。すべてのチェックボックスをオフにし、OK をクリックしてインストーラを閉じます

4.2 はじめて起動するとき

Windows®ワークステーションのスタートメニューからe² studioを開きます。Renesas RA <バージョン番号> メニューグループに表示されます。インストール中にショートカットを変更した場合は、ここに入力したグループに表示されます。スタートメニューにまったく見つからない場合は、C:\Renesas\RA\e2studio_v<e2 studio version number>_fsp_v<fsp version number>\eclipse にあるインストールディレクトリを参照してください。

e² studio がワークスペースのフォルダを要求します。既定値を受け入れるか、独自のフォルダを選択できます。[Launch]をクリックします。IDEを初めて起動する場合は、追加のサポートファイルが抽出されて更新されるため、起動に少し時間がかかることがあります。Logging/Reportingダイアログが表示されます。決定を行い、適用を選択します。[Welcome]画面が表示されたら、画面の右上隅にある[Hide]アイコンを選択して画面を閉じます。

次の手順では、最初のシンプルなプロジェクトを作成して、すべてが機能していることを確認します。そのためには、メニューバーのFile → New → Renseas C/C++Project → Renseas RAと移動します。これにより、Renesas RA Project という新しいウィンドウが表示されます。サイドバーで[Renesas RA]を選択し、次に[Renesas RA C/C++ Project]を選択します(図4-7 参照)。[Next] をクリックします。プロジェクトコンフィギュレータが起動します。プロジェクトの名前(“MyRaProject” など) を入力し[Next]をクリックします。

ここでは「MyRenesas」のログイン資格情報を尋ねる画面が表示されます。それらを入力し、使用状況データの記録と送信に同意するかどうかを決定します。 後で気が変わった場合は、e² studioの設定で決定を変更することができます。この画面では、まだ登録していない場合は、「MyRenesas」に新規登録することもできます。

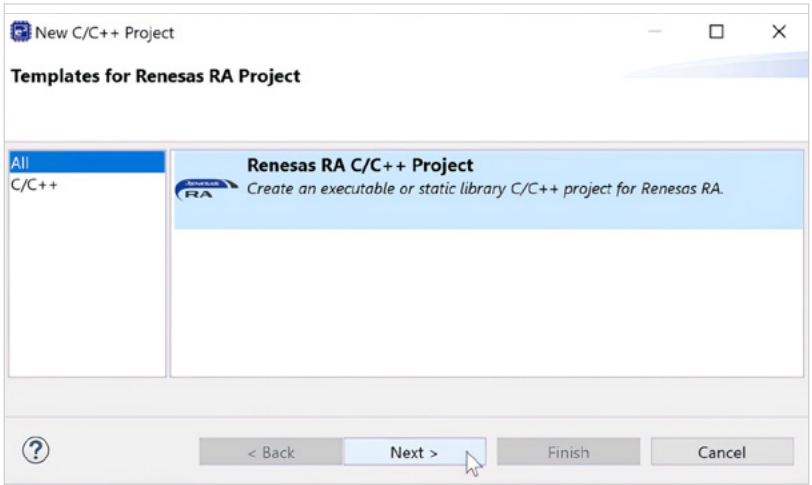

図4-7: RA ファミリのプロジェクトコンフィギュレータを呼び出すRenesas RA C/C++ Project エントリの選択

[Login]をクリックして、Device and Tools Selection 画面に移動します。まず、ボードを選択します。開発環境をテストしたいだけなので、EK-RA6M4 を選択し、対応するデバイスをR7FA6M4AF3CFB に設定します(まだ表示されていない場合)。ツールチェーンを見てみましょう。GNU Arm® Embedded と読んでください。[Next]をクリックします。

GNU ツールをインストールしたが、ここには表示されていない場合は、手動で統合する必要があります。そのためには、e² studio のヘルプメニューに移動し、[Add Renesas Toolchains]を選択します。Scan オプションは、既定の場所にインストールされているツールチェーンに対してのみ機能することに注意してください。デフォルトの場所にGNU ツールをインストールしていない場合は、入力フィールドにカスタムフォルダへのパスを入力するか、Add コマンドとファイルブラウザを使用して場所を参照します。

表示される画面で、TrustZone®以外のプロジェクトとセキュアでないTrustZone プロジェクトのどちらかを選択できるようになりました。フラット(非TrustZone) プロジェクトを選択したまま(図4-8 を参照)、[Next]をクリックします。Build Artifact and RTOS Selectionウィンドウが表示されます。設定はそのままにしておきます。[Build Artifact Selection]で[Executable]を選択し、[RTOS Selection]で[No RTOS]を選択します。RTOS の選択を気にする必要はありません。FreeRTOS™とAzure® RTOS ThreadX®は、必要に応じてRA コンフィギュレータで後からアクティブにできます。

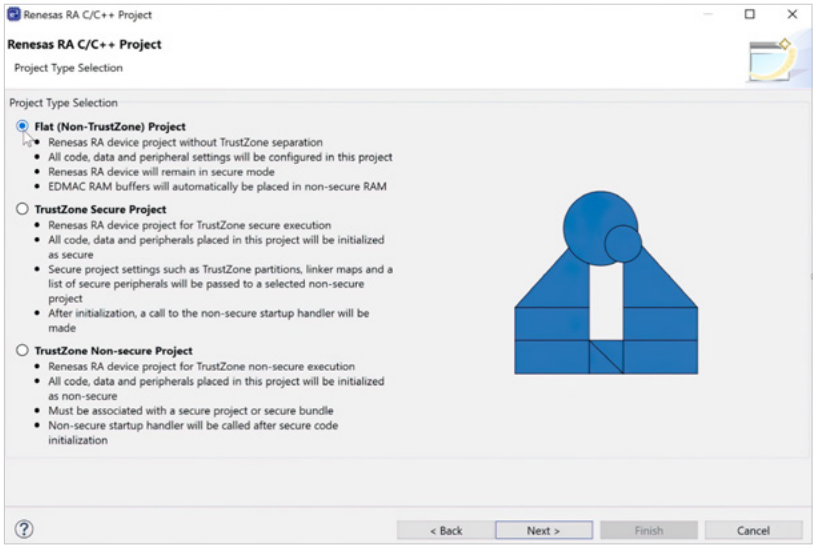

図4-8: ここでは、すでに選択されているフラットプロジェクトをアクティブのままにしておきます

[Next]をクリックして、Project Template Selection という名前の次の画面に移動します。このテストでは、サンプルパッケージを使用すれば十分ですので、Bare Metal - Blinky を選択します。Finish をクリックします。これで、コンフィギュレータに、正しいデバイスとボード依存のBoard Support Library、およびサンプルプロジェクトを含むプロジェクトテンプレートを作成するための十分な情報があります。

コンフィギュレータは、必要なファイルをすべて抽出し、FSP Configuration Perspectiveを開くかどうかを尋ねます。Open Perspective に答えると、パースペクティブが表示されます。コンフィギュレータでは、FSP とプロセッサのほとんどのオンチップハードウェアを設定できます。この例で使用したテンプレートの場合、すべてがすでに設定されているので、ウィンドウの右上隅にあるGenerate Project Content という小さな緑の矢印をクリックすると、コンフィギュレータは必要な設定ファイルをすべて作成し、プロジェクトに追加します。

最後に、IDE 上部のツールバーの左隅にある小さな「ハンマー」記号をクリックすると、プロジェクトがコンパイルおよびリンクされます。完了すると、エラー0、警告0 がステータスとして表示されます。これで、すべてが設定され、作業が完了し、独自のプロジェクトを開始する準備が整いました。

また、インストール中に予期しない問題が発生した場合は、e² studioからでもヘルプがいつでも入手できます。Help→Renesas Help→Renesas Helpdeskに移動すると、ルネサスのホームページのデザイン&サポートページに移動します。もう1 つは、FSP Configuration PerspectiveのSummary タブのSupport アイコンをクリックすることです。これにより、Webブラウザで Knowledge Baseを開くこともできます。

4.3 インストールを最新の状態に保つ

ツールをインストールしたら、適切な形に保つことができます。つまり、ツールを最新の状態に保つことができます。または、Eclipse™ベースの開発環境が提供するすべての可能性について、追加のソフトウェアをインストールする必要があります。インストールを最新の状態に保つには、**https://github.com/renesas/fsp/releases** のGitHub®リポジトリーから最新のドキュメントとインストールをロードします。e² studio 内からアップデート機能を使用する(Help → Check for updates)ことは、アップデートがRA ファミリのマイクロコントローラには何のメリットも追加されず、検索に時間がかかる可能性があるため、お勧めしません。

e² studio 内からアップデート機能を使用することはお勧めしません(ヘルプ→アップデートの確認)。次の画面で、ウィンドウ上部の「操作」の下にあるドロップダウンリストから「-すべての利用可能なサイト-」エントリを選択すると、すべてのサイトが検索され、ダウンロード可能なソフトウェアのリストが表示されます。さらに、ルネサスRA Partner Ecosystem から追加のソフトウェアとソリューションを見つけることができる、ルネサスのWeb サイト(**https://www.renesas.com/ra**) のRA ファミリページに移動することもできます

インストールが機能していることがわかったので、次に開発キットをチェックアウトし、独自のコードを作成します。

本章で学んで会得するポイント:

- GitHub®は、ツールチェーンの最新リビジョンをダウンロードする場所です
- e² studioインストーラを使用する場合、ツールチェーンの設定は簡単です
- コンフィギュレータは、プロジェクトとコンフィギュレーションを作成する手順を示します

5 e² studioの操作

この章で学ぶこと:

- パースペクティブ、ビュー、エディタの違い
- 操作はコンフィギュレータごとに異なり、使用方法も異なる
- プロジェクトの読み込みと書き出しの方法

ルネサスフレキシブルソフトウェアパッケージ(FSP)の内容とそのアプリケーションプログラミングインタフェースの使い方がわかったので、e² studioのマイクロコントローラ「RAファミリ」の開発環境を見ていきます。

ルネサスが開発･保守するe² studio は、さまざまなプログラミング言語やターゲットプラットフォーム向けの広く普及しているオープンソースの統合開発環境(IDE) であるEclipse™に基づいています。Eclipse は簡単にカスタマイズおよび拡張できるため、世界中の数千人の開発者やデファクトスタンダードに適したIDE です。

e² studioは、Eclipse のすべての利点を活用し、RA ファミリのすべての機能をサポートするための追加のビューとコンフィギュレータパースペクティブを備えています。このパースペクティブには、あらゆる規模と複雑さのプロジェクトを作成、コンパイル、デバッグするために必要なすべてのツールが含まれており、ソフトウェアデザインの準備、ビルド、デバッグの3つのフェーズを通じて開発者をガイドします。また、最新のEclipse SDK およびCDT ツールを使用するために定期的に更新されます。

この章の以下の部分では、Eclipse ワークベンチの詳細と、そのさまざまな要素の使用方法について説明します。いくつかの背景について説明しますが、ここではすべてを説明するわけではありません。詳細については、e² studio 内のヘルプ→ヘルプコンテンツを参照して、ワークベンチユーザーガイドを参照してください。

5.1 Eclipse の理念の簡単な紹介

ワークベンチと呼ばれるe² studio のメインウィンドウは、パースペクティブ、ビュー、エディタ、メニュー、ツールバーなどのいくつかの基本的なユーザインタフェース要素で作成されます。

e² studio を起動すると、最後に使用したパースペクティブが開きます。パースペクティブは、ビュー、エディタ、ツールバーのセットと、ワークベンチ内での配置です。ウィンドウ、ツールバー、ビューを再配置すると、これらの変更は現在のパースペクティブに保存され、次回開いたときに再び利用できます。

e² studioでは、あらかじめ定義されたパースペクティブがいくつかあり、C/C++ パースペクティブやリソースパースペクティブなど、それらのパースペクティブを同時に複数表示することができます。パースペクティブから別のパースペクティブへの変更は、ツールバーの右側にあるパースペクティブ一覧の横にある四角いアイコンをクリックするか、メインメニューバーから[Perspective → Open Perspective → Other …]として行うことができます。どちらの方法でも、使用可能なすべてのパースペクティブを表示したポップアップウィンドウが表示されます。同じメニューエントリを使用するか、パースペクティブリスト内のパースペクティブを右クリックして、パースペクティブを閉じることができます。開いているパースペクティブの名前を右クリックすると、それを切り離してワークベンチ内で自由に移動することもできます。

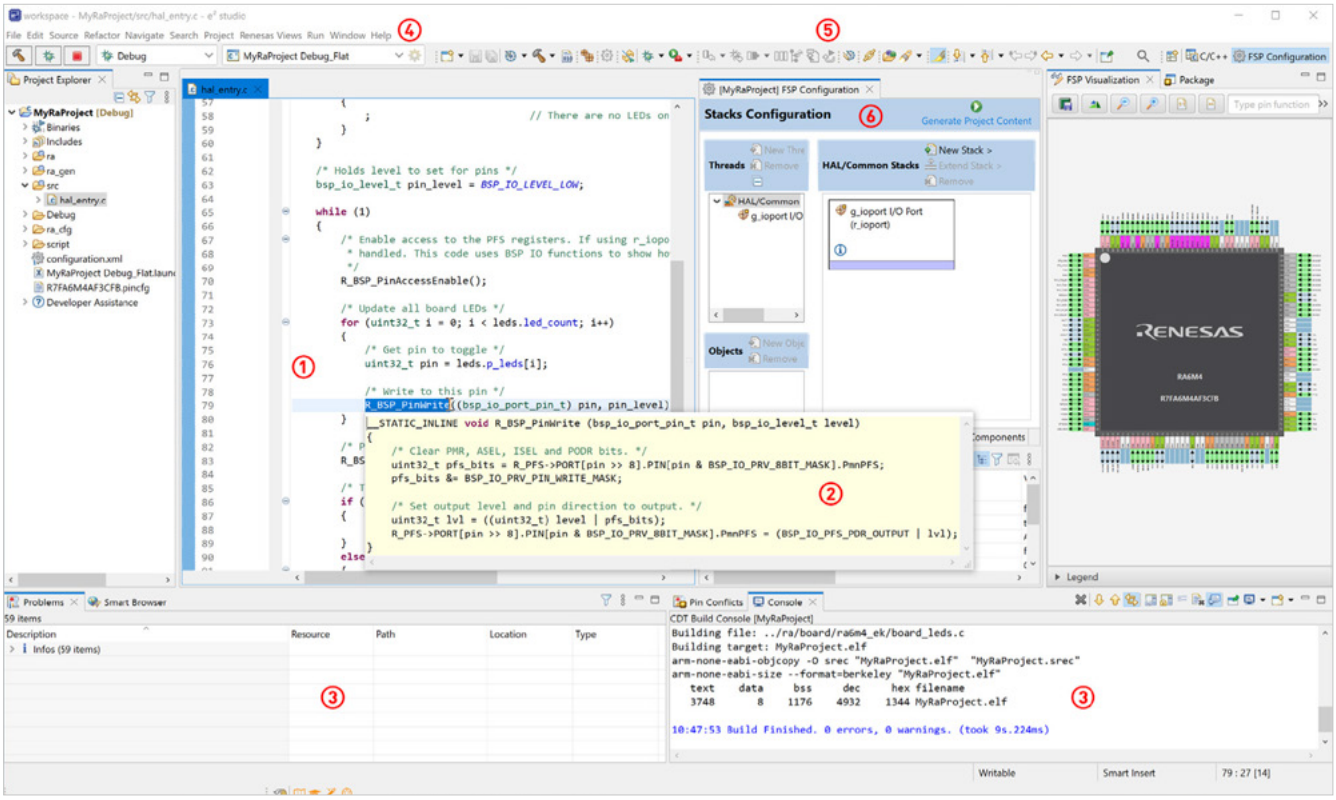

図5-1: e² studioのワークベンチは、エディタ①、スマートマニュアル②、ビュー③、メニューバー④、ツールバー⑤、パースペクティブ⑥ で構成されています

5.1.1 e² studioのパースペクティブ

- **C/C++:** e² studioのスタンダードパースペクティブで、プログラムの開発やソースコードの編集に使用されます。デフォルトではエディタが中央に開き、Project Explorer ビューが左側に開き、プロジェクトを管理できます。アウトラインビューには、エディタでアクティブなソースファイルのすべての変数と定義が表示されます。下部には、問題(構文エラーやコンパイルエラーなど)、タスク(Doxygen タスクの一覧)、プロパティなど、いくつかのスタック形式のノートブックが表示されます。
- **デバッグ:** このパースペクティブは、プログラムの実行、および実行時に発生する問題の診断とデバッグに使用されます。e² studio では、デフォルトで以下のビューが開きます。Debug ビュー(はい、Debug パースペクティブには Debug ビューが含まれています)。Debug ツールバー、エディタウィンドウ、アウトラインビュー(C/C++ パースペクティブと同様) が右側に、分析および視覚化のためのスタックビューが下部に、タブ付きノートブックが上部に、変数、ブレークポイント、レジスタなどを監視するために表示されます。
- **リソース:** デフォルトでは、このパースペクティブにはC/C++ パースペクティブからのほとんどのビューが含まれます。また、下部のウィンドウにはタスクビューのみが表示されます。一般的なリソースの表示とナビゲーションを提供します。
- **FSP 設定:** これはパースペクティブの1 つで、最初に多くの時間を費やすことになります。ワークベンチの最大部分は、FSP フレキシブルソフトウェアパッケージ Configuration パースペクティブ(パースペクティブには他のパースペクティブも含めることができます) によって占有されます。このパースペクティブには、クロックコンフィギュレータやピンコンフィギュレータなど、さまざまなFSP コンフィギュレータ用のタブ付きビューがいくつかあります。また、さまざまなペリフェラル、スレッド、ドライバの設定を行うことができるプロパティビュー、およびPackage ビューも含まれています。Package ビューでは、現在のピン割り当てを示すマイクロコントローラ用に選択されたパッケージの外観が、設定の問題が検出されたかどうかを示すステータスとともに表示されます。必要に応じて、C/C++ パースペクティブおよびデバッグパースペクティブからFSP コンフィギュレータにもアクセスできます。
- **チーム同期:** このパースペクティブでは、チームの他のメンバの変更とマージできます。
- **トレース:** このパースペクティブでは、統計ビューとヒストグラムビュー、およびデバッグコールフロービューが開きます。

パースペクティブに変更を加えて気に入らない場合は、メインツールバーでパースペクティブの名前を右クリックし、「リセット」を選択することで、いつでもデフォルトにリセットできます。これは、大量のビューやエディタを開いたりドッキングを解除したりして、集中的なコーディングやデバッグセッション中に作成した混乱を解消するのに役立ちます。画面の大きさが十分でないため、一度完了すると、非常に混乱したワークベンチになる可能性があります。

5.1.2 ビュー

ビューは、レジスタのセット、プロパティ、ファイルのリストなどを調べることができるウィンドウです。ビューには、実行中のスレッドのリアルタイムチャートや、変数に接続するゲージやコントローラなどの代替表現を含めることもできます。ビューには、独自のメニューまたはツールバーを含めることができます。ビューのメニューまたはツールバーによってトリガされるアクションは、そのビュー内の項目にのみ影響しますが、同じパースペクティブ内に存在する場合でも、他のビュー内の項目には影響しません。

複数のビューを同じウィンドウに重ねることができます。これをタブ付きノートブックと呼びます。また、1 つのノートブックから別のノートブックにビューを移動したり、ビューを完全にドッキング解除したりすることもできます。e² studioの優れた機能の1 つで、ワークフローに合わせてすべてを配置でき、パースペクティブの最後のレイアウトが自動的に保存されます。

e² studioでは、すでに多数のビューが用意されているほか、メモリー使用状況ビュー、障害状況ビュー、RTOS リソースビューなどの追加ビューが作成されており、統合開発環境の汎用性がさらに高まりました。 .これらはすべて、メインメニューバーの[Renesas Views]にあります。。

5.1.3 エディタ

これは、すべてのソフトウェア開発者がほとんどの時間を費やすビューです(エディタは特別なタイプのビューと見なされます)。コード補完やキーワードの強調表示など、通常の機能や期待される機能に加えて、e² studio に組み込まれているエディタには、スマートマニュアルと呼ばれる特殊な機能が含まれています。FSP やエディタビュー内のマイクロコントローラ自体でコンテキストアウェアヘルプを取得するため、何千ページものドキュメントを検討する必要がありません。

e² studio内の任意のキーワード、機能、変数、マクロの上にマウスを置くと、関連する情報を表示するウィンドウが表示されます。変数については、宣言が提示され、構造体と列挙型についてはすべてのメンバ、関数については説明、プロトタイプ、パラメータの詳細が提示されます。変数がマイクロコントローラのレジスタに関連付けられている場合は、ビット定義に関する特定の情報も表示されます。Smart Manual は、使用可能な場合は、該当するアプリケーションノートやメディアリッチな説明資料を引き出し、C/C++ パースペクティブの下部にあるSmart Browser ビューに表示します。

この機能は、FSP とRA マイクロコントローラのマニュアルとe² studio を切り替える必要がないため、非常に有用で時間を節約できます。必要な場所(マウスカーソルの右側) に関連情報が表示されます。

コード補完は、IDE のこれらの機能の1 つであり、コード開発中に多くの時間を節約します。変数の後に<ctrl>-<space>を押しても、API-structure のメンバのように使用可能なオプションが表示されます。メンバのいずれかをクリックすると、それらがコードに挿入されます。ここでも、マニュアルを参照する必要はなく、ポイントしてクリックするだけです。

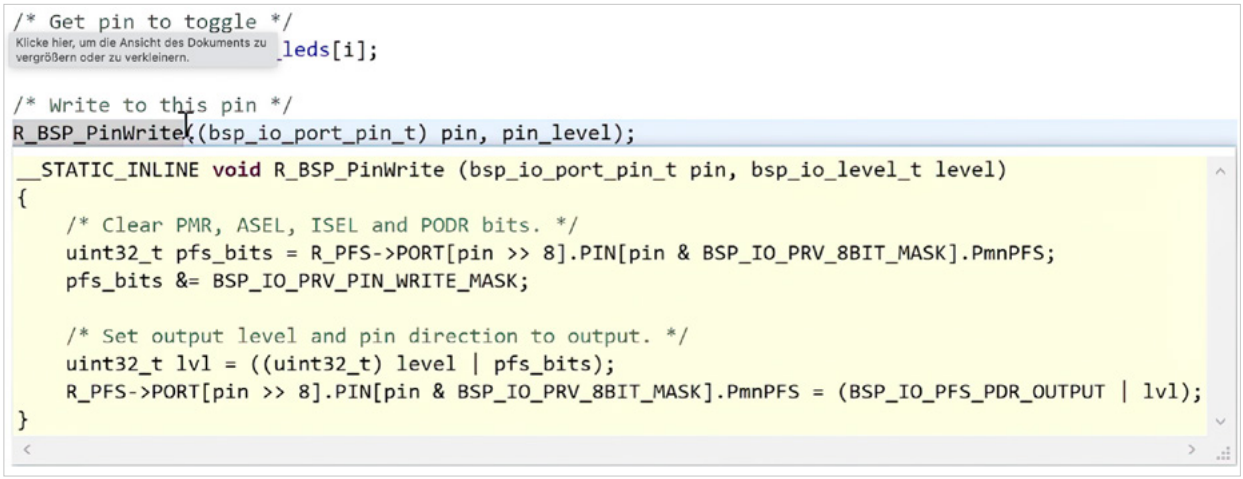

図5-2: e² studio のスマートマニュアル機能は、必要な場所に情報を表示します

5.2　コフィギュレータの簡単な紹介

e² studio内のコンフィギュレータは、RA ファミリマイクロコントローラ(MCU) の固有のデバイスオプションを介してソフトウェア開発者をグラフィカルにガイドし、その使用方法をアドバイスします。その例としては、ピンの選択や競合の警告などがあります。また、コンフィギュレータはスタートアップコードを生成したり、プロジェクト内にFSP ソフトウェアコンポーネントを配置したりします。

5.2.1　プロジェクトコンフィギュレータ

開発者に役立つコンフィギュレータがいくつかあり、ユーザーが最初に体験するのはプロジェクトコンフィギュレータです。プロジェクトコンフィギュレータは、新規プロジェクトをゼロから作成するプロセス、またはコンフィギュレータが提供するテンプレートから作成するプロセスを実行します。使用するツールチェーン、使用するデバイスとボード、サンプルプロジェクトを作成するかどうかなど、プロジェクトのほとんどの設定を選択できます。

この処理の最後に、プロジェクトが自動的に生成され、e² studio のワークベンチに追加されます。手動で作成する必要はありません。make-file のパラメータや設定を調べたり、どのヘッダファイルをインクルードするかを調べたり、大規模なRA ファミリのマイクロコントローラから特定のシリーズやデバイスを使用するようにコンパイラにアドバイスしたりする必要はありません。最後のコンフィギュレータ画面で[Finish]をクリックすると、すべての設定が完了します。

コード生成が完了すると、プロジェクトコンフィギュレータはFSP Configuration パースペクティブに切り替わり、さまざまなFSP コンポーネントの設定を行うことができます。

5.2.2　FSPコンフィギュレータ

FSP コンフィギュレータ がパースペクティブに表示されると、選択したボードとデバイス、使用されているツールチェーンを示す、プロジェクトの読み出し専用のサマリが表示されます。また、プロジェクト内のさまざまなソフトウェアコンポーネントがバージョン番号とともに一覧表示されます。ページの下部にあるショートカットを使用すると、YouTube™のRenesas FSP プレイリストに素早くアクセスできます。このプレイリストには、RA ファミリに関するいくつかの短いビデオ、Renesas.com のRenesas FSP Support ページ、およびフレキシブルソフトウェアパッケージ のユーザーズマニュアルが含まれています(図5-3 を参照)。

コンフィギュレータの下部にある次のタブはBSP (ボードサポートパッケージ) タブで、デバイスやボードの選択など、ボード設定の側面を表示および編集できます。関連するプロパティ ビューで、メインスタック(スレッドコンテキスト外で使用されるスタック)のサイズやMCU の一部のセキュリティ機能など、BSP の追加設定を行うことができます。

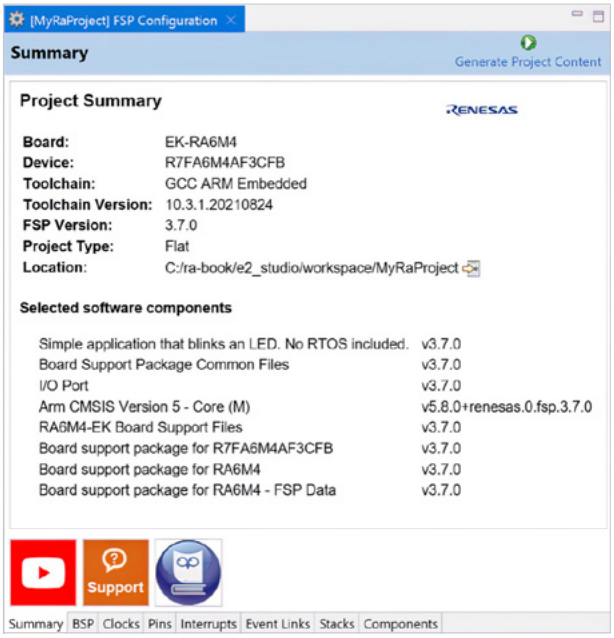

図5-3: FSP Configurator のSummary タブ

[Clocks]という名前の次のタブは、初期クロック構成を割り当てるためのものです。オンチップクロックシステムの図を示し(図5-4 を参照)、クロッキングツリーに変更を加えることができます。項目の上にマウスを置くと、その簡単な説明が表示されます。互換性のない設定を行うと、それぞれのメンバが赤色で強調表示され、問題の説明が表示されます。また、タブ自体に小さな感嘆符が表示され、問題の有無が示されます。

[Pins] タブは、RA MCU の初期ピンセットアップをカバーしています。ピンは、ポートまたはペリフェラルに基づいて表示されます。コンフィギュレータの右側のパッケージビューには、コンフリクトや不完全な設定がある場合、設定されたピンとマーキングエラーをハイライト表示するデバイスのパッケージが表示されます。これらは、ProblemビューとPin conflictビューにも表示されます。

以下のタブ[Interrupts]では、ユーザー定義(非FSP) ドライバがRA プロジェクト内でICU (Microprocessors Interrupt Controller Unit) をどのように使用するか、およびどの割り込みサービスルーチン(ISR) をICU イベント(割り込み) に結び付けるかを指定できます。また、割り当てられたすべてのICU イベントの完全なリストも表示されます。このテーブルには、コンフィギュレータのStacks ビューで作成されたFSP モジュールインスタンスによって生成されたイベントも含まれます。

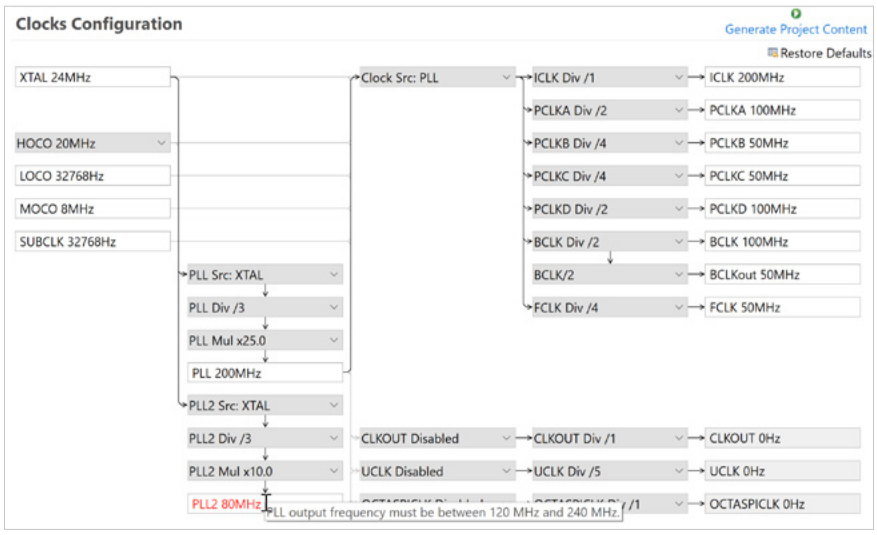

図5-4: 意図的にエラー設定をしたFSP-Configurator のクロックビュー

Event Linkタブは、Interruptsタブと同様の目的を果たします。ここでは、ユーザーは、RA プロジェクト内で独自のドライバがどのようにEvent Link Controller (ELC)を使用するかを指定し、そのようなドライバが一連のELC イベントを生成するか、一連の周辺機能を介して一連のELC イベントを消費するかを宣言できます。

次に、Threads タブで、RA プロジェクト内でスレッドを追加および設定できます。個々のスレッドには異なるモジュールとオブジェクトを追加でき、それらのプロパティはProperties ビューで変更できます。Threads ビューには、さまざまなスレッドのスタックが表示され、モジュールのグラフィカル設定が作成されます。新しいスレッドは簡単に追加でき、必要なすべてのモジュールがレベルに自動的に挿入され、開発者の介入が必要になります。この場合、マウスをモジュールの上に置くと、モジュールは赤色でマークされ、必要な設定や問題の説明が表示されます。解決した場合、モジュールは標準色に戻ります。

FSP Configuration パースペクティブの最後のタブにはComponents という名前が付けられ、ユーザーは異なるFSP コンポーネントを表示および選択できます。また、使用可能なRA CMSIS ソフトウェアコンポーネントも一覧表示されます。現在のプロジェクトへのモジュールの追加や削除などの変更は、Threads タブで行うことをお勧めします。これは、後のプロジェクトでもモジュールの設定が可能になるためです。

FSP Configuration ですべての設定が行われると、関連するソースコードをFSP から生成または抽出し、プロジェクトに追加することができます。これは、パースペクティブの上部にあるGenerate Project Content シンボルをクリックして行います。クリックを忘れた場合は、心配しないでください。これは、設定または生成されたファイルへの変更が検出された場合にも発生します。

FSP コンフィギュレータは、プロジェクトの準備や初期設定に必要なすべての手順をガイドするため、優れたツールです。プロジェクト·コンフィギュレータと同様に、何千ページものドキュメントを使用し、それらを深く調査する必要はありません。これは、コンフィギュレータが必要な情報を抽象レベルで提供し、すべての設定が正しい、妥当で、一貫性があることを保証するためです。このようなコンフィギュレータを使用せずに、最後のプロジェクトでどのくらいの時間を費やしたかを覚えておけば、ピン配線や割り込み設定のすべてが適切であり、相互に依存する多くのハードウェアレジスタを扱うことができれば、ルネサスがこれらのツールに取り組んでいるすべての労力を確実に理解できるでしょう。このような方法であれば、それほど問題はありません。

5.3 プロジェクトのインポート・エクスポート・実行

プロジェクトをインポートまたはエクスポートする必要がある場合があります。ルネサスウェブサイトの多数のサンプルプロジェクトのいずれかを使用する場合や、ピアのいずれかと最新の開発を共有する必要がある場合があります。インポートやエクスポートを行う場合でも、インポートウィザードまたはFSP エクスポートウィザードを使用してe² studio内から簡単に行うことができます。

5.3.1 プロジェクトのインポート

インポートウィザードを呼び出すには、Project Explorer ビューで右クリックしてメニュー表示からImport を選択するか、メニューバーに移動してFile → Import を選択して開く方法の2 つがあります。どちらの場合も、インポートウィザードのウィンドウが表示されたら、「General」エントリを展開し、「Existing Projects into Workspace」または「Rename & Import Existing C/C++ Project into Workspace」を選択します。後で、読み込んだプロジェクトの新しい名前を入力する機会が得られます。

どちらの場合も、インポートするプロジェクトが存在するソースディレクトリを選択するか、アーカイブファイルを選択するかの選択肢が表示されます。フォルダ内に1 つ以上のプロジェクトが見つかった場合は、それらすべてがインポート対象としてマークされます。インポートしないプロジェクトの選択をキャンセルします。Finish をクリックすると、プロジェクトがワークスペースにインポートされます。Copy projects into workspace の横にあるチェックボックスをオンにすると、そこでコピーされます。インポートしたプロジェクトを実行する前に、再コンパイルする必要があります。

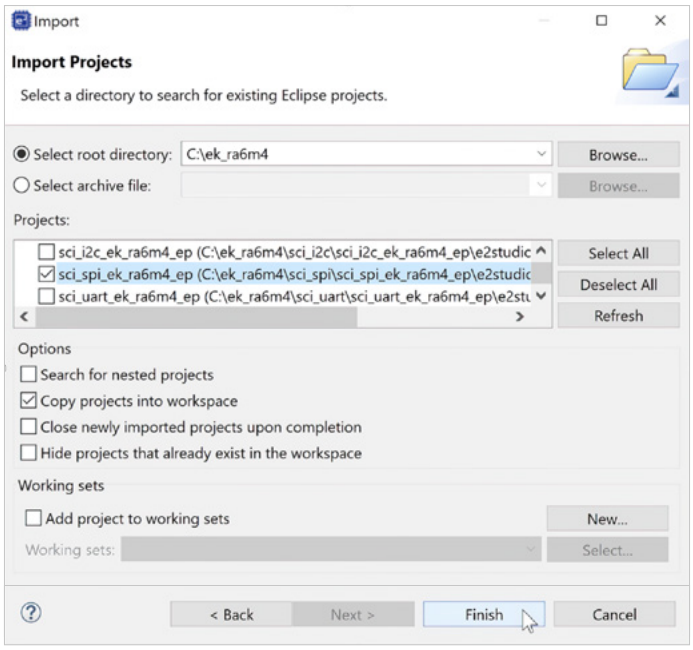

図5-5: 特定の場所で見つかったプロジェクトがインポート対象として一覧表示されます

5.3.2　プロジェクトのエクスポート

プロジェクトの書き出しは、読み込みと同じくらい簡単です。ここでも、タスクを達成するには2つの方法があります。1つ目は、e² studio のメニューバーに移動し、File → Export を選択する方法です。2つ目は、エクスポートするプロジェクトを右クリックし、Export FSP Project を選択します。最初のバージョンを使用している場合、エクスポートウィンドウで実行する必要がある追加の手順が2つあります。1つ目はGeneral エントリを展開し、2つ目はリストからRenesas FSP Archive File エントリを選択する手順です。どちらのバージョンでも、FSP エクスポートウィザードが表示されます。

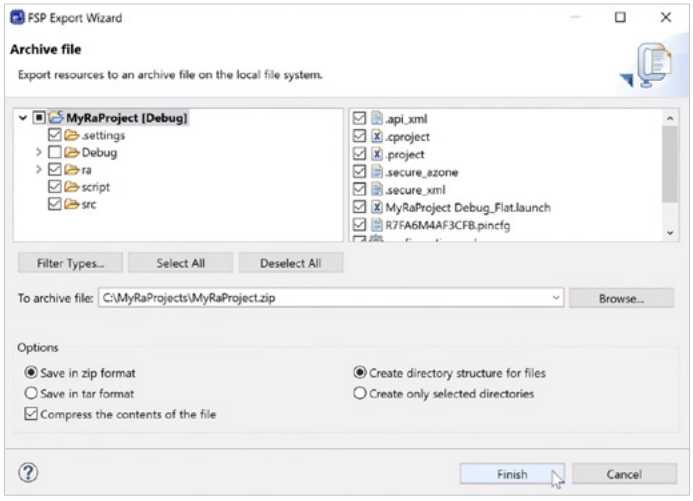

図5-6: e² studioのFSP エクスポートウィザード

FSPエクスポートウィザードが表示されたら、エクスポートするプロジェクトとファイルを、圧縮形式と目的のディレクトリ構造とともに選択できます。Finish をクリックすると、エクスポート処理が終了します。

デフォルトでは、Debugエントリとraエントリはチェックされません。情報を含むconfiguration.xml ファイルはエクスポートの一部であり、ファイルを開いて(FSP Configurator が起動します)、Create Project Content を再度クリックすることで、見つからないコンテンツを再作成することができるため、エクスポートに含める必要はありません。これにより、エクスポート中に省略されたファイルが復元されます。この方法でエクスポートされたプロジェクトは、他のワークステーションで簡単に共有およびインポートすることができます。

本章で学んで会得するポイント:

- e^2 studioは、機能をグループ化するためのさまざまなパースペクティブとビューを提供します
- スマートマニュアル機能、プロジェクトコンフィギュレータ、FSPコンフィギュレータは開発を高速化し、エラーを削減します
- プロジェクトはインポート·エクスポートウィザードを使って簡単にインポート·エクスポートできます

6　RAファミリ用のハードウェア評価キット

この章で学ぶこと:

- RA Family のさまざまな開発キットとその主な機能は何か

すべての開発プロジェクトで、最初のテストを実行するためにハードウェアが必要になる時があります。そして、すべてのエンジニアにとって必要になるのはほとんどの場合、自分のボードが使用できるようになるよりもずっと前のタイミングです。あるいは、ハードウェア担当者は貴重なプロトタイプボードを、ソフトウェア開発担当に壊されてしまうかもしれないので貸したくないというケースもあるでしょう（なお、これは私の話ではありません）。

ルネサスでは、ソフトウェア設計者がすぐにプログラムをテストできるように、開発プラットフォームにもなる簡単で手間のかからないさまざまな使いやすいキットを提供しています。ハードウェア設計者にとっても、キットのさまざまなコネクタでマイクロコントローラ(MCU) のピンのほとんどに便利にアクセスできるため、ボードとしてメリットがあります。必要に応じて、Digilent Pmod™、Arduino™ (Uno R3) コネクタ、MikroElektronika™ mikroBUS コネクタなどの標準化されたインタフェースを介して、一般的なエコシステムアドオンを使用してボードの機能を強化することもできます。これらのサードパーティモジュールは、個別に使用することも、同時に使用することもできます。

各MCU グループには、デバイスのセットごとに1つの評価キット(EK)があります。これは、メモリサイズが最大で、オンチップペリフェラルの選択肢が最大のものです。各MCU グループでは、マイクロコントローラのRA ファミリへの簡単なエントリを提供し、より小さいメモリや少ないペリフェラルが必要な場合は、後でMCU グループ内のデバイスを切り替えることができます。すべてのEKは、ボード全体で共通のベースライン機能を提供し、PCB に特殊な機能を配置するのに十分な柔軟性があります。すべてのキットには、ハードウェアを理解するために使用できる事前にプログラムされたクイックスタートサンプルプロジェクトが付属しています。また、エンジニアがより複雑なカスタム組み込みアプリケーションのビルディングブロックとして利用できる追加のサンプルコードも用意されています。

すべてのキットは、学習曲線をユーザに対してフラットに保つように構築されています。またこれらは、RAファミリマイクロコントローラの評価、プロトタイピング、開発において最適な出発点を提供します。

物理的には、ボードは3つの異なるセクションに分割されています。PCBの上部には、システム制御およびEcosystemアクセス領域があり、サードパーティコネクタ、USBコネクタ、デバッグインタフェース、ユーザボタン、ユーザLEDなどがあります。中央には、Special Feature Access Area という名前のオプション領域があり、ボード上のMCU に固有のインタフェース、追加メモリ、またはオプションが配置されます。下部には、マイクロコントローラ付きのMCU ネイティブピンアクセス領域、ピンアクセス用の40ピンのオスヘッダ、クロックソースがあります

すべてのキットに共通する機能がいくつかあります:

- 電源
- デバッグインタフェース
- USBフルスピード接続
- ユーザLED
- ユーザボタン
- マイコン電流測定テストポイント
- MCU デジタルおよびアナログピンへのアクセス

評価キットの他に、他のボードやシステムも利用できます。これらは、静電容量式タッチ、モータ制御、グラフィックスなどの特定のアプリケーションの特別なニーズに対応し、それらが提供するアプリケーションスペースに合わせて特別に調整された1つ以上のボードを含みます。すべてのキットに共通するのは、モジュール性と拡張性、そしてシンプルでエラーのない構成です。

各キットには独自のWeb サイトがあり、URL は**https://www.reneasas.com/**<キット名> のシンタックスに従います。したがって、ウェブブラウザに**https://www.renesas.com/ek-ra6m4** と入力すると、ラボで使用するEK-RA6M4 のページにアクセスできます。ここでは、各ボードの詳細情報だけでなく、キットのユーザーズマニュアル、クイックスタートガイド、ホワイトペーパ、サンプルプロジェクトなどもダウンロードできます。また、回路図、部品表、3D および機械図面、独自の開発のベースとして使用できる製造および設計ファイルを含む、いわゆるデザインパッケージもダウンロードできます。

最後になりましたが、評価キットは、FCC Notice- EMC/EMI のパート15 とCE クラスA、廃棄物、リサイクル、材料選択のためのEU RoHS と中国SJ/T 113642014、安全のためのUL 94V-0 など、多くの国際規格に標準に準拠しています。

6.1 EK-RA6M4 評価キット

EK-RA6M4 は、RA6M4 MCU グループのマイクロコントローラ(MCU)用の評価キットです。144 ピンLQFP パッケージでTrustZone®、200MHz Arm® Cortex®-M33 コアを搭載したR7FA6M4AF3CFB MCU を中心に構築されています。コード用に1MB のオンチップフラッシュメモリと256KB の内部SRAM を搭載しています。このキットを使用すると、RA6M4 MCU グループの機能を簡単に評価し、高度な組み込みシステムアプリケーションを開発することができます。

複数のクロックソースからは、高精度の24.000MHz および32,768Hz 基準クロックが供給されます。追加の低精度クロックには、RA マイクロコントローラ内部でアクセスできます。電流測定ポイントは、正確な消費電流測定を可能にするために、PCB上にも配置されています。

外部への接続は、USBフルスピードホストとデバイスポート、およびイーサネットコネクタを介して利用できます。マイクロコントローラのほとんどの機能へのアクセスは、4つの40ピンオス·ヘッダと、最も一般的なサードパーティ·エコシステムの5つへの拡張コネクタによって提供されています。オンボードのJ-Link®デバッガを介して、ソフトウェアとハードウェアの簡単なデバッグが可能です。このボードは、フレキシブルソフトウェアパッケージ(FSP) とe² studioで完全にサポートされています。

評価キットのその他の機能は次のとおりです:

- メモリ:
  - 内蔵: 1MB コードフラッシュメモリ+ 256KB SRAM
  - オフチップ: 64MB Octo-SPI フラッシュメモリ+ 32MB Quad-SPI フラッシュメモリ
- 接続インタフェース:
  - USB (デバッグx1、USBフルスピードホストおよびデバイスx1、micro-ABコネクタ)
  - RMII イーサネット(RJ45 コネクタ)
- デバッグモード:
  - デバッグ·オンボード(SWD)
  - デバッグイン(ETM、SWD、JTAG)
  - デバッグアウト(SWD)
- ユーザインタフェース
  - x ユーザボタン
  - リセットボタン
  - 3 x ユーザLED
  - 電源LED
  - デバッグLED
- 拡張:
  - ネイティブピンアクセス用 40 ピンオスヘッダ 4 個
  - 2 x SeedGrove®拡張コネクター(I²C/アナログ)
  - SparkFun® Qwiic®コネクタ
  - 2 x Digilent Pmod™コネクタ (SPIおよびUART)
  - Arduino™ Uno R3コネクタ
  - MikroElektronika™ microbus connector

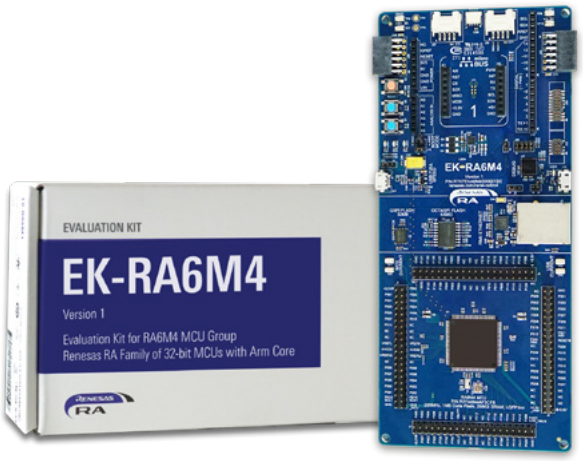

図6-1: EK-RA6M4 評価キット

このキットのドキュメントは、ルネサスのWebサイトから入手できます。キットの最初の動かし方を案内するクイックスタートガイド、ボードのすべての技術的な詳細を説明するユーザーマニュアル、およびキットの回路図、BOM、ガーバーファイルを含むデザインファイルパッケージです。これは今後の演習で使用するボードなので、ユーザーズマニュアルをダウンロードして、不明点がある場合は参照することをお勧めします。

6.2 グラフィックス拡張ボード付きEK-RA6M3G 評価キット

強力なヒューマンマシンインタフェース(HMI) やカラフルなディスプレイに関しては、RA ファミリのマイクロコントローラ内のRA6M3 グループが適しています。TFT用オンチップグラフィックLCDコントローラとその2Dグラフィックエンジンを評価するために、ルネサスはEK-RA6M3Gグラフィックス評価キットを提供しています。これにより、エンジニアはRA6M3 MCUグループの機能をシームレスに評価し、フレキシブルソフトウェアパッケージ(FSP)とe² studio IDEを使用して独自のグラフィックスベースの組み込みシステムアプリケーションを開発することができます。このキットは、176 ピンLQFP パッケージの120MHz Arm® Cortex®-M4 コアを搭載したR7FA6M3AH3CFC MCU を中心に構築されています。コード用に2MB のオンチップフラッシュメモリと640KB の内部SRAM を搭載しています。

EK-RA6M3Gキットは、MCUを搭載したEK-RA6M3ボードと、静電容量式タッチオーバーレイの4.3インチTFTカラー液晶を搭載したグラフィックス増設ボードの2つのボードから構成されています。LCD の LED バックライトを駆動するために、グラフィックス拡張ボードにはバックライトコントローラも含まれています。

外部への接続は、フルスピードおよびハイスピードのUSB ポート、およびイーサネットコネクタを介して利用できます。マイクロコントローラの機能へのアクセスは、4つの40ピン オスヘッダと、最も一般的なサードパーティエコシステム用の4つの拡張コネクタによって提供されます。オンボードのJ-Link®デバッガを介して、ソフトウェアとハードウェアの簡単なデバッグが可能です。

複数のクロックソースからは、高精度の24.000MHz および32,768Hz 基準クロックが供給されます。追加の低精度クロックには、RA マイクロコントローラ内部でアクセスできます。電流測定ポイントは、PCB 上に配置され、MCU およびUSB ポートの詳細な電流消費測定を可能にします。

図6-2: グラフィックス拡張ボード搭載のEK-RA6M3G 評価ボード

評価キットのその他の機能は次のとおりです:

- **メモリ:**
  - 内蔵: 2MB コードフラッシュメモリ+ 640KB SRAM
  - オフチップ: 32MB Quad-SPI フラッシュメモリ
- **接続インタフェース:**
  - 3 x USB (1 x デバッグ、1 x USB フルスピード、1 x USB ハイスピード)
  - RMII イーサネット(RJ45 コネクタ)
- **デバッグモード:**
  - デバッグ・オンボード(SWD)
  - デバッグイン(ETM、SWD、JTAG)
  - デバッグアウト(SWD)
- **ユーザインタフェース:**
  - 2 x ユーザボタン
  - リセットボタン
  - 3 x ユーザLED
  - 電源LED
  - デバッグLED
- **拡張:**
  - ネイティブピンアクセス用 40ピン オスヘッダ 4 個
  - 2 x SeedGrove®拡張コネクタ (I²C)
  - 2 x Digilent Pmod™コネクタ(SPIおよびUART)
  - Arduino™ Uno R3コネクタ
  - MikroElektronika™ microbus connector
- **グラフィックス拡張ボード:**
  - 4.3- 解像度480 x 272ピクセルのインチTFTカラーLCDとコントローラ付き静電容量式タッチオーバーレイ
  - バックライトコントローラ

このキットには、ホワイトペーパ、アプリケーションノート、キットの初回起動時に使用するクイックスタートガイド、ユーザーズマニュアル、キットのすべての技術詳細、ボードの回路図、BOM、ガーバーファイルなどを含むデザインファイルパッケージが含まれています。グラフィックの開発は、Segger™ emWin Embedded GUI Library と、それに付随するホストベースのAppWizard によってサポートされ、emWin ライブラリ用のすぐに実行できるアプリケーションを作成します。どちらも、強力なHMI の開発を非常に簡単にします。

6.3 RA6T1 モータ制御評価システム

ルネサスRAファミリマイクロコントローラ(MCU)のRA6T1グループ用Motor Control Evaluation Systemは、低電圧永久磁石同期モータ(ブラシレスDCモータ)ソリューションを提供します。モータ制御システムの評価に必要なすべてのハードウェアを使いやすいキットとしてまとめています。これにより、問題なく評価が可能となり、モータシステムの効率的な開発が可能となります。

この包括的なソリューションは、永久磁石同期モータ、3シャント電流検出、過電流保護、およびUSBポートを介したルネサスモータワークベンチとの通信をサポートします。Motor Workbench では、制御システムのパラメータやリアルタイム監視を自動調整できます。キット自体は、RA6T1 CPU カード、48 V インバータボード、1つの永久磁石同期モータ、および必要なケーブルで構成されています。付属のCPUカードを別のCPUカードに交換することで、お使いのモータ制御システムを別のMCUをベースに評価することができます。

図6-3: Motor Workbench では、パラメータのオートチューニングと実行中のシステムのモニタが可能

このキットは、512KB および64KB のSRAM のオンチップフラッシュメモリを搭載した120MHz R7FA6T1AD3CFP MCU を中心に構築されています。本MCU は、Arm® Cortex®-M4 コアをベースに、高分解能PWM タイマ、12 ビットA/D コンバータ、高速アナログコンパレータ、各種インタフェースを搭載しています。

キットのその他の機能には以下が含まれます:

- **RA6T1 CPUカード:**
  - R7FA6T1AD3CFP MCU
  - ボード接続コネクタ
  - オンボードJ-Link™用USBコネクタ
  - ルネサスMotor Workbench通信用SCIコネクタ
  - コネクタ
    - *CAN*
    - *SPI*
    - *ホールセンサ信号入力*
    - *エンコーダ信号入力*
    - *Arm®デバッガ用10/20ピンスルーホール*
  - 2インバータ用のスルーホール
  - 2 x ユーザコントロール用LED
  - MCU 外部リセット用スイッチ
- **インバータボード:**
  - トグルスイッチ
  - プッシュスイッチ
  - 3 x ユーザLED
  - インバータ制御回路ブロック電源用LED
  - ボード接続コネクタ
  - USBコネクタ
  - モータコネクタ
  - 可変抵抗器

図6-4: RA6T1 モータ制御評価システム

キットには、簡単に起動できるようにプログラム済みのソフトウェアが付属しています。モータを回転させるには、ハードウェアコンポーネントの組み立てのみが必要です。他のシステムと同様に、ホワイトペーパ、アプリケーションノート、キットの初回起動時に使用するクイックスタートガイド、ユーザーズマニュアルなど、キットのWeb サイトから完全なドキュメント一式を入手できます。このマニュアルには、ボードのすべての技術詳細と、ボードの回路図、BOM、ガーバーファイルなどを含むデザインファイルパッケージが含まれています。サンプルコードもダウンロードでき、開発者は独自の設計をすぐに開始することができます。

6.4 EK-RA6M5 評価キット

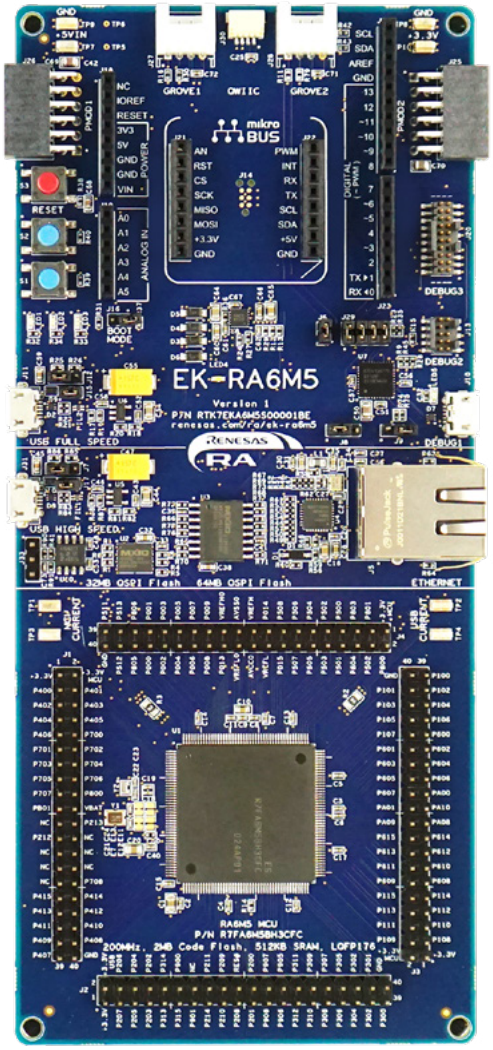

図 6-5: EK-RA6M5評価キット

強化されたセキュリティ、イーサネットまたはCAN接続、および大容量の組み込みメモリを必要とするIoTアプリケーション向けに、ルネサスはRA6M5グループのマイクロコントローラを提供しています。 お客様の設計をサポートするために、EK-RA6M5評価キットを提供しています。 これは、176ピンLQFPパッケージの動作周波数200MHz Arm®Cortex®-M33コアに基づくR7FA6M5BH3CFCMCUをメインCPUとして構築されています。 プロセッサには、コード用の2MBのオンチップフラッシュメモリと512KBの内部SRAMが付属しています。 消費電力は低く、フラッシュからCoreMark®プログラムを実行すると107µA/MHzを実現します。

コアに組み込まれたArmの®TrustZone®とセキュア暗号エンジンにより、RA6M5グループのMCUは、セキュリティアプリケーションに最適です。 セキュリティ機能と豊富な通信を必要とする分野としては、スマートメータやその検針用のデバイス、およびロボット工学、自動販売機、UPS（無停電電源装置）などの産業用アプリケーションが挙げられます。

EK-RA6M5は、さまざまな方法で外部デバイスと通信します。1つのUSBフルスピードと1つの高速ホストおよびデバイスポート、イーサネットインターフェイス、およびCAN FD（フレキシブルデータレート）に準拠するCAN（コントローラーエリアネットワーク）バストランシーバを備えています。 CAN FDは、元のCANバス標準を拡張し、より高いデータレートと、より大きなペイロードを可能にします。 評価キットにはオンボードのJ-Link®デバッガが付属しているため、ソフトウェアとハードウェアのデバッグは簡単です。 消費電流の測定ポイントを準備しており介して正確な電流測定が可能です。 外部の24.000MHzおよび32.768Hzの基準クロックは、正確なタイムベースを提供します。 また、RA6M5 MCUの内部でも追加の低精度クロックを利用できます。

- **メモリ**
  - MCU内蔵: 2MB Flash memory、512KB SRAM
  - ボード実装: 64MB Octo-SPI Flashメモリ、32MB Quad-SPI Flashメモリ
- **有線通信**
  - CAN FD
  - 3 x USB (1 x デバッグ用I/F、1 x USB full speed, 1 x USB high speed)
  - Ethernet (RMII, PHY, RJ45コネクタ)
- **デバッグモード**
  - Debug on-board (SWD)
  - Debug in (ETM, SWD, JTAG)
  - Debug out (SWD)
- **拡張用コネクタ**
  - 4 x 40-pin ネイティブピンアクセス用のオスヘッダ
  - 2 x SeedGrove® 拡張コネクタ (I²C)
  - 2 x Digilent Pmod™ コネクタ (SPI、UART)
  - SparkFun® Qwiic™
  - Arduino™ Uno R3コネクタ
  - MikroElektronika™ microbusコネクタ
- **ユーザインタフェース**
  - 3 x ユーザLED
  - 2 x ユーザボタン
  - 電源LED
  - デバッグLED
  - リセットボタン

他の評価キットと同様に、ユーザーマニュアル、アプリケーションノート、その他のドキュメント、およびサンプルプロジェクトが利用可能です。 開発者は、評価キットのWebサイトから回路図、設計ファイル、部品表、および機械図面を含む完全な設計パッケージをダウンロードすることもできます。

本章で学んで会得するポイント:

- RAファミリ用の様々な評価キットは、学習曲線をフラットに保ちます
- 各MCU グループ内にはデバイスごとに1つの評価キットがあり、一部のキットには特別な機能があります

7. ルネサスEK-RA6M4 評価キットの初回起動

この章で学ぶこと:

- ルネサス評価キットをワークステーションに接続する方法
- デバッグ設定とは何か、どのように作成するか
- 統合開発環境でプログラムをダウンロードして起動する方法

本章では、EK-RA6M4 評価キット(EK) が動作し、Windows®ワークステーションおよびe² studio のデバッガと通信していることを確認します。このために、4.2 章で生成したプロジェクトを使用します。この演習を行わなかった場合も心配しないでください。本書のWebサイトからダウンロードできます。ただし、後者のケースでは、第4 章で説明したように、開発ツールチェーンをすでにインストールしてテストしているはずです。

7.1 接続とOut-Of-The Box デモ

まだEK を箱から開梱していないなら、まさに今がそのときですので開けてみましょう。内容を確認し、EKメインボード、USB micro-B ケーブル1本、micro USB OTG アダプター1本、クロスオーバーEthernet ケーブル1本がすべて揃っていることを確認します。EK-RA6M4 クイックスタートガイドおよびユーザーズマニュアルで構成されるキットのマニュアルは、**https://www.renesas.com/ra/ek-RA6M4** のキットのWeb サイトから、文書のリンクに従ってダウンロードできます。ボードにはSEGGER J-Link® On-Board (OB) デバッガが含まれており、完全なデバッグおよびプログラミング機能を提供するため、キットに付属のEK およびUSB ケーブルを使用する際にエミュレータは必要ありません。

開始するには、USB ケーブルのmicro-B 側をUSB デバッグポートJ10 (システム制御およびエコシステム アクセスエリアの右下にある)に挿入し、もう一方の端をPCの空いているUSB ポートに挿入します。白色LED4 は、ボードに電源が入っていることを示し、「EK-RA6M4」という文字のハイフンを形成します。ボードが起動し、動作するとすぐに、あらかじめプログラムされたデモプログラムが実行され、1 秒間隔で青色のLED1 が10% の光度で点滅します。緑色のLED2 は、フル強度で常時オンになり、赤色のLED3 はオフになります。ユーザーボタンS1 を使用すると、LED1 の強度を10% から50% に、90% から10% に切り替えることができます。押しボタンS2を作動させるたびに、LED1の点滅周波数を1Hzから5Hzから10Hzに、そして1Hzに戻すことができます。

デバッグポート以外のオレンジ色のデバッグLED (LED5) が点滅し続けると、評価キットはJ-Link ドライバを検出できないことを意味します。その場合は、USB ケーブルをボード右側のデバッグポートJ10 に左辺のUSB Full Speed ポートJ11 に接続していないことを再度確認してください。また、画面のデバイスマネージャを開き、Universal Serial Busコントローラツリーを展開して、インストールプロセス中にJ-Linkドライバがホストワークステーションにコピーされていることを確認します。J-Link ドライバがそこにリストされているはずです (図7-1 を参照)。もしインストールされていない場合は、カスタムインストール機能を使用してプラットフォームインストーラを再実行し、ルネサスエレクトロニクス株式会社からのソフトウェアのインストールを許可してください。これにより問題が修正されます。

このボードには、WEBサーバー、Quad-SPI、Octo-SPIの速度比較など、他のデモプログラムも付属しています。これらのデモを実行するには、評価キットと通信するためのターミナルプログラムがワークステーションに必要です。接続部とデモの設定については、『RA6M4 クイックスタートガイド』を参照してください。デモを実行する必要はありません。

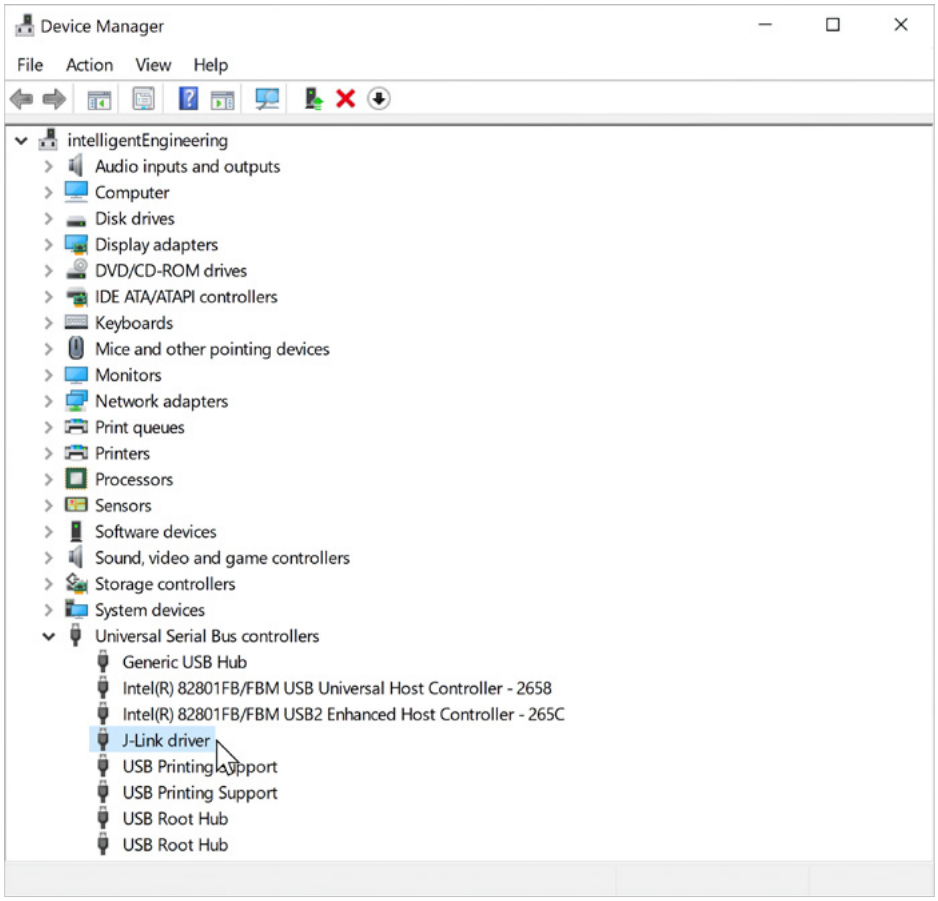

図7-1: 黄色のデバッグLED の点滅が停止しない場合は、J-Link ドライバが正しくインストールされていることを確認してください。

7.2 ダウンロードとテストの例

EK-RA6M4 をWindowsワークステーションに接続したまま、オペレーティングシステムのスタートメニューからe² studio を開きます。ワークスペースの場所の入力を求められた場合は、4.2 章で試行したものを使用します(すでにリストされているはずです)。試行していなかった場合は、このドキュメント用に作成されたWeb サイト(**www.renesas.com/ra-book**) からプロジェクトをダウンロードできます。ダウンロードしたら、5.3.1 章で概説した手順に従ってワークスペースにインポートします。この場合は、任意のフォルダを使用します。

プログラムをキットにダウンロードして実行する前に、デバッグ構成を作成する必要があります。デバッグ 記号  の横にある小さな矢印をクリックし、ドロップダウンリストボックスから[Debug Configurations] を選択します。

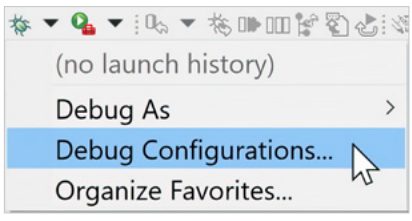

図7-2: デバッグを開始するには、ドロップダウンリストボックスからDebug Configurations を選択します。

表示されたウィンドウで、左側のツリービューのRenesas GDB Hardware Debugging の下のMyRaProject Debug_Flat を強調表示します。このプロジェクトに別の名前を使用した場合は、使用した名前を選択します。

プロジェクトを選択すると、デバッグコンフィギュレーションの新しい画面が表示され、その個々のオプションがすべて表示されます(図7-3 を参照)。ここでは、テストが目的で何も変更する必要はありません。下部のデバッグ をクリックすると、デバッガが起動します。[Confirm Perspective Switch]ダイアログが表示されたら、[Yes]を選択します。

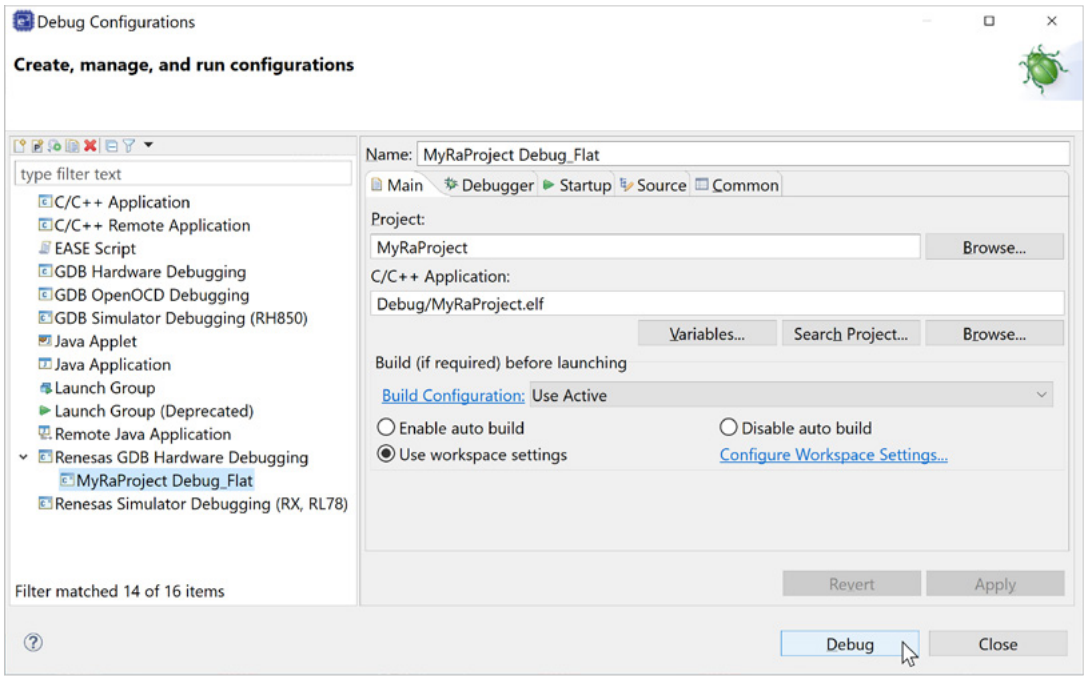

図7-3: 「Renesas GDB Hardware Debugging」でプロジェクトを選択したら、何も変更する必要はありません。

J-Link® ファームウェア更新 という名前の2 番目のダイアログが表示され、オンボードデバッガ用の新しいファームウェアバージョンをインストールするように求められる場合があります。[Yes]をクリックしてアップデートを許可することを強くお勧めします。

Windowsワークステーションのセキュリティ設定によっては、セキュリティ警告を表示するダイアログウィンドウが表示され、「Windows Defender Firewall has blocked some features of E2 Server GDB on all public and private networks.」と表示される場合があります。続行するには、E2 Server GDB がプライベートネットワーク上で通信することを許可します。このためには、それぞれのチェックボックスを選択し、[Allow access]をクリックします。

デバッグパースペクティブが開くと(図7-4 を参照)、デバッガはプログラムカウンタをプログラムのエントリポイントであるリセットハンドラに設定します。[Resume] ボタン をクリックすると、プログラムはmain() 関数内のhal_entry() の呼び出し行で次の停止まで実行されます。「再開」をもう一度クリックすると、プログラムは実行を継続し、評価キットの青色、緑色、赤色のLED (LED 1 ～3) が1 秒間隔で同時に点滅します。

最後の手順は、Disconnect ボタン をクリックしてプログラムの実行を停止し、デバッガを切断することです。

e² studio のインストレーションが評価キットと一緒に動作することが確認できました。これで、マイクロコントローラのRA ファミリ用のプログラムを初めて書くことができます。次章では、マイクロコントローラのRA ファミリ用に初めてのプログラムを作成します。

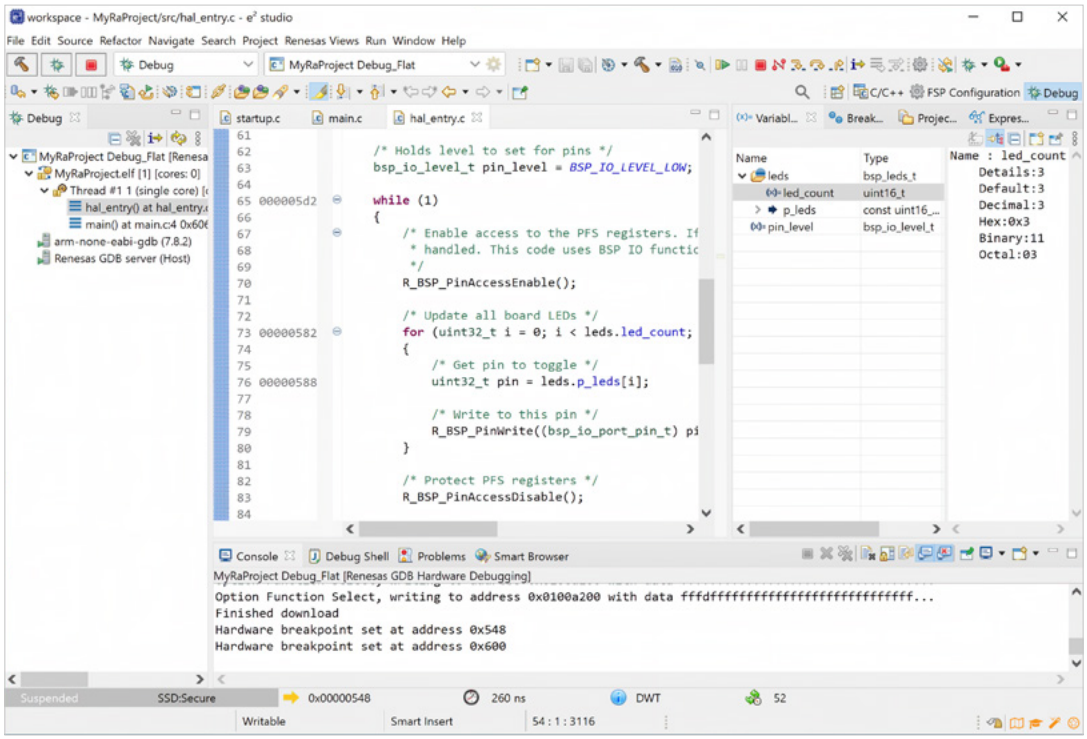

図7-4: e² studio のデバッグパースペクティブ

本章で学んで会得するポイント:

- 任意のハードウェアにプログラムをダウンロードしてデバッグするには、まずデバッグ構成を作成する必要があります
- ロードされると、デバッガはプログラムカウンタをエントリポイントに設定します。次の停止はmain() になります

8 HELLO WORLD! – HI BLINKY!

この章で学ぶこと:

- EK-RA6M4 評価キット用のプロジェクトをゼロから作成する方法
- FSPコンフィギュレータでフレキシブルソフトウェアパッケージ (FSP) の設定を変更する方法
- EKボードのユーザーLED を切り替えるコードの記述方法
- プログラムのダウンロードとテストの方法

プログラミング言語を初めて書いた(そしてまだ書いている)プログラムの中で、最初のプログラムは、単に文字列"Hello World"を標準出力デバイスに置くだけのものでした。私にとっては、Pascalで始まったように、エディタに"Writeln ('Hello World')"とタイプしていました。その後、同じようなラインを他のいくつかの言語で書きました。主に、新しい開発環境のインストールのサニティチェックとして書きました。

1980年代後半に組込みシステムのプログラムに移ったとき、文字列を送ることができる画面はありませんでした。では、プロセッサに生命の兆候を与えるように指示するにはどうすればよいでしょうか。当時LEDはまだ普及しておらず、ごくわずかなI/Oピンを切り替えて、オシロスコープで波形を観察していました。長年にわたりLEDはコモディティ化し、それらをボード上に配置し、それらの点滅を新しい「Hello World」として使用します。

また、これもこの章の目的です。RA6M4 グループデバイスの評価キット (EK)のLED を切り替えます。前の章で学んだことをすべて使用します。コードをほぼ最初から記述し、コンフィギュレータを使用して新しいプロジェクトを作成し、Flexible Software Package (FSP) のAPI を採用し、最後にコードをダウンロード、デバッグ、実行します。この演習では、すべてのコンテンツをまとめています。

前提条件として、e² studio とFSP をWindows®ワークステーションにインストールし(詳しくは第4 章を参照)、第7章で説明したように設定が機能していることを確認しておく必要があります。以前の演習をやっていた方にとっては、目次から本章に直接移動する人たちのために、先に説明したトピックのいくつかをもう一度取り上げることにしたので、ご容赦ください。

この演習では、RA6M4 マイクロコントローラとそのすべてのペリフェラルをすぐに検索できるように、このような作業に非常に適したEK-RA6M4 をもう一度使用します。外部ハードウェアは簡単に取り付けることができます。ほとんどのピンは、MCUネイティブピンアクセス領域のブレークアウトピンヘッダまたはボードのシステム制御およびエコシステムアクセス領域のエコシステムコネクタを介してアクセスできるため、外部ハードウェアを簡単に接続できます。またRA6M4 MCU をRAファミリのRA6シリーズのスーパーセットデバイスとすることで、このシリーズのほとんどの機能を評価し、結果をRAファミリの他製品にも適用することができます。図 8-1 に、ボードのブロック図を示し、主要コンポーネントを強調表示します。

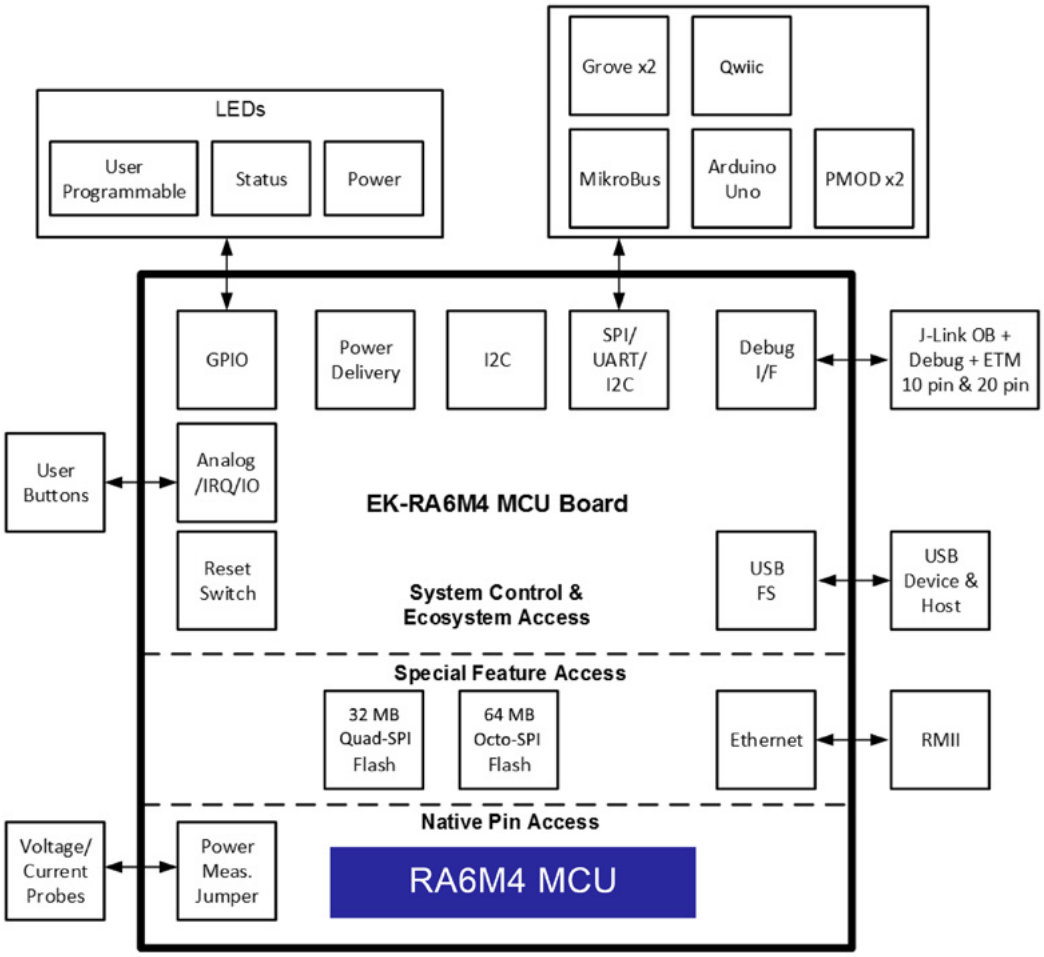

図8-1: EK-RA6M4 評価キットのブロック図

8.1 プロジェクトコンフィギュレータを使用したプロジェクトの作成

Windows®ワークステーションのスタートメニューからe² studio を開きます。開発環境が起動して実行されたら、Welcome 画面が表示されている場合は、他のウィンドウの表示をブロックするように、画面を閉じます。

e² studioでマイクロコントローラ用の新しいプログラムを作成するには、必ずプロジェクトを作成する必要があるため、これが最初のステップです。これを行うには[File]→[New]→[Renesas C/C++ Project]→[Renesas RA]に移動するかプロジェクトエクスプローラービューを右クリックして「新規」→「C /C++プロジェクト」を選択します。どちらの方法でも、使用するテンプレートを尋ねるダイアログが表示されます。左側のサイドバーでRenesas RA を選択し、メインウィンドウからRenesas RA C/C++ プロジェクト を選択します(図4-7 参照)。それから[Next]をクリックします。

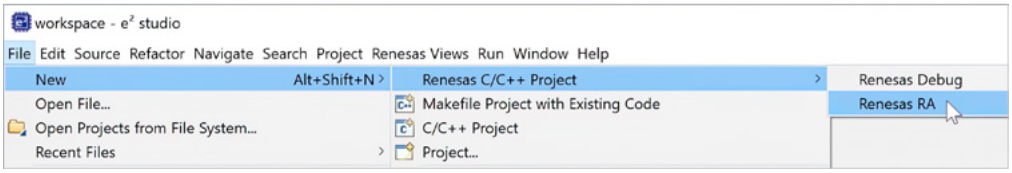

図8-2: まずプロジェクトコンフィギュレータを呼び出します。

プロジェクトコンフィギュレータが表示されたら、プロジェクトにMyBlinkyProject などの名前を付け、プロジェクトのデフォルトの場所(e² studioワークスペース) を受け入れるか、環境設定のフォルダに変更します。次の をクリックして、Device and Tools Selection 画面に移動します。

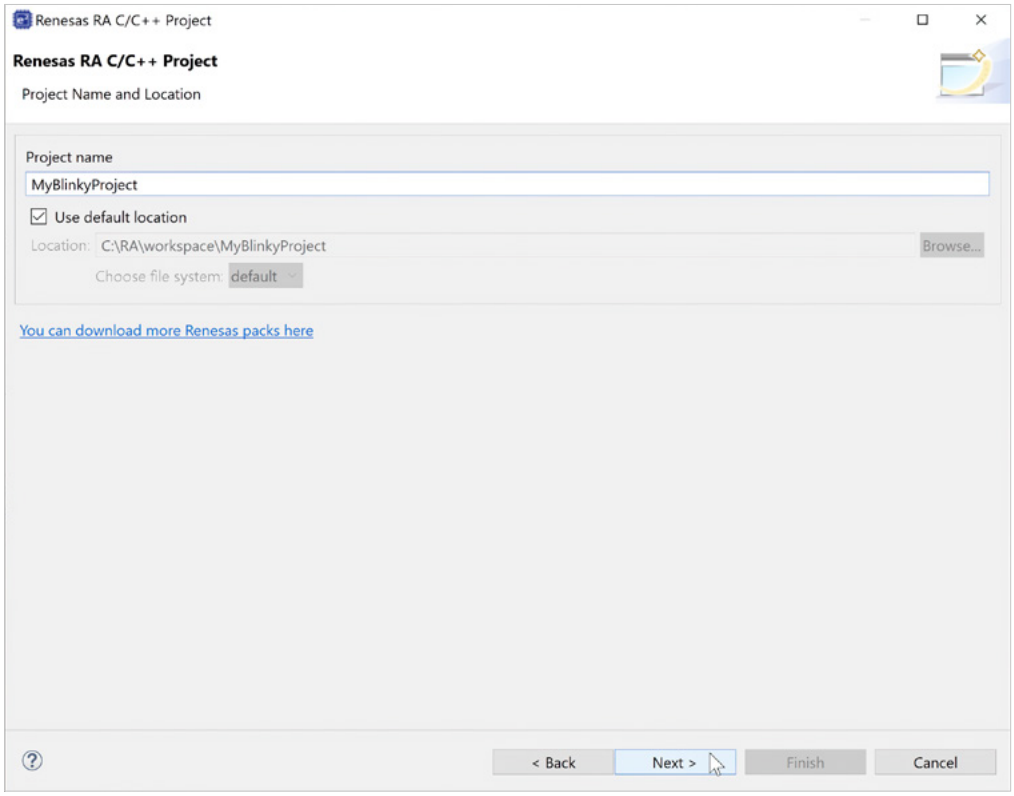

図8-3: プロジェクトコンフィギュレータの最初の画面には、主にプロジェクトの名前と場所が表示されます。

以前にダウンロードしたのと同じバージョンのフレキシブルソフトウェアパッケージが表示されているはずです。[Board] で、ドロップダウンリストからEK-RA6M4 を選択します。これは、小さい"Hello World" プログラムに使用するハードウェアです。このリストには通常、RA ファミリの評価ボードに加え、カスタムユーザーボード参入が含まれ、選択したFSP バージョン用にインストールされたRenesas CMSIS パックファイルから作成されます。R7FA6M4AF3CFB がデバイスの横に表示されていることを確認します。自動的に挿入されている必要があります。表示されていない場合は、ドロップダウンリスト内を移動して見つけます。ツールチェーンフレームで、GCC ARM® Embedded、10.3.1.20210824以降がリストされていること、およびデバッガフレームで [J-Link® Arm] が選択されていることを確認します。これらの分野はあらかじめ入力しておく必要があります。入力していない場合は、上記の値で変更してください。

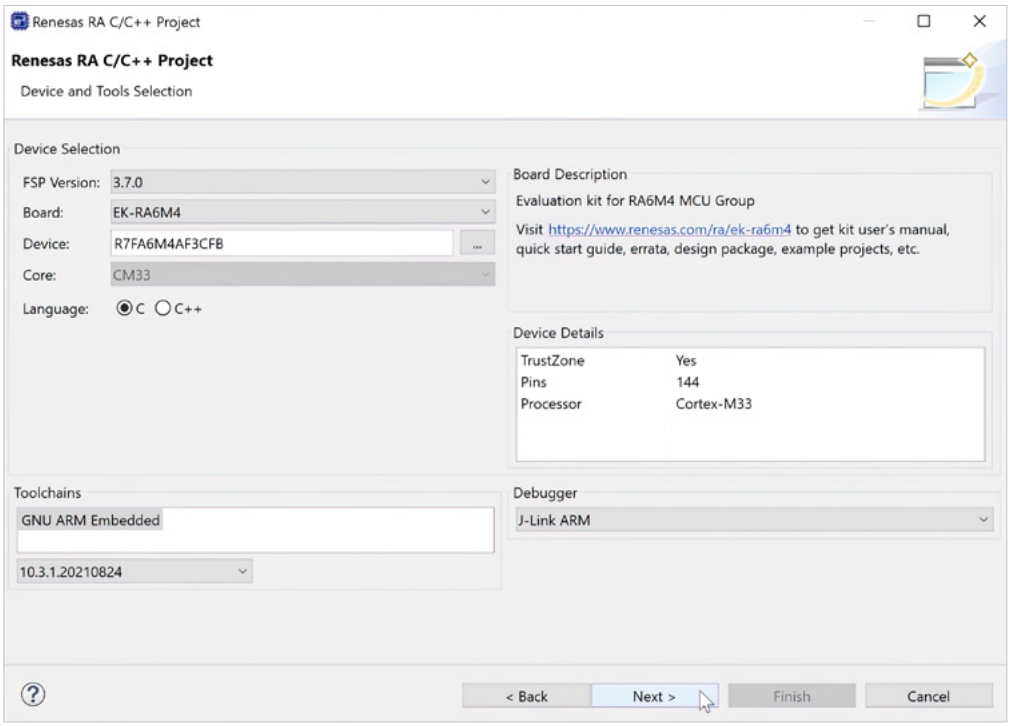

図8-4: このページでは、プロジェクトのボードとデバイスを選択できます。

見た目がすべて適切な場合は、[Next]をクリックして[Project Type Selection]画面を開きます。ここでは、プロジェクトをいわゆるFlat projectにするかどうかを選択できます。ここでは、すぐに実行できるTrustZone®を分離しないプロジェクト、安全なスタートアップコードやその他のセキュアコードを含むTrustZone プロジェクト、またはセキュアなプロジェクトと一緒に使用するためにノンセキュアコードを含む非TrustZone プロジェクトを選択できます。この章の行使では、Flat (Non-TrustZone) プロジェクト を選択し、[Next]をクリックして続行します。

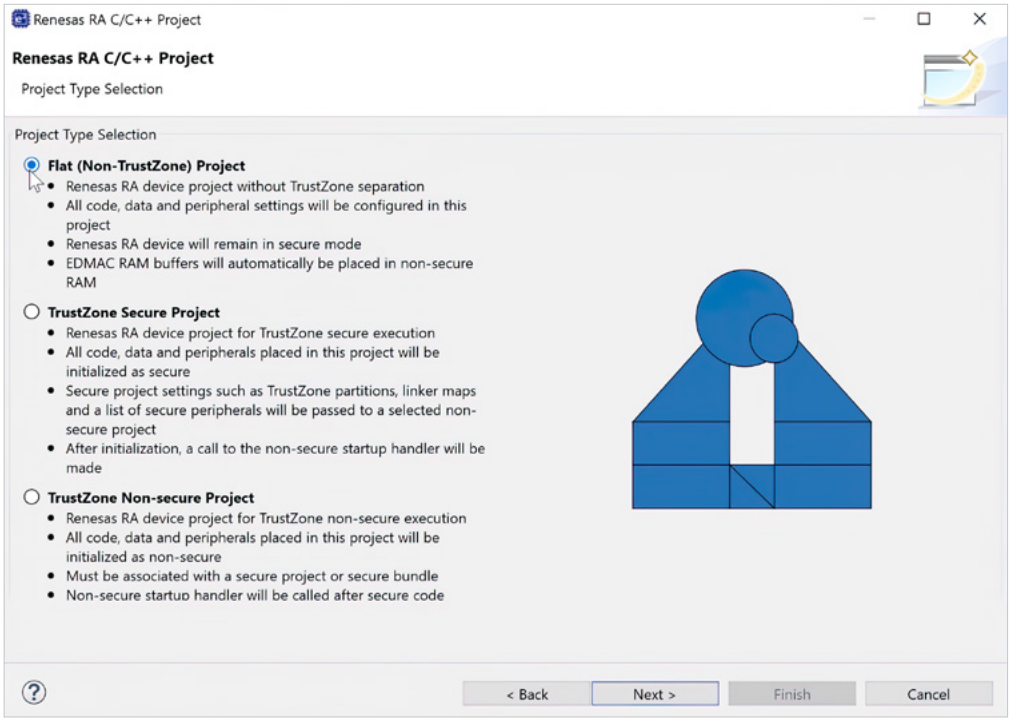

図8-5: 「Project Type Selection」画面では、TrustZone プロジェクトとTrustZone 以外のプロジェクトを選択できます。

次のページは、[Build Artifact and RTOS Selection] 画面で、ビルドのタイプを設定できます。この画面は、TrustZone 以外のデバイスまたはTrustZone デバイスのFlat Projectが前のウィンドウで選択された場合にのみ表示されます。使用可能なオプションは、[Executable]、自己完結型ELF (実行可能およびリンク可能形式) 実行可能ファイルの作成、オブジェクトコードライブラリを作成する[Static Library]、およびStatic Libraryで使用するように設定されたアプリケーションプロジェクトを作成する[Executable using an RA Static Library]です。ページの右側で、ドロップダウンリストを使用すると、プロジェクトに任意のリアルタイムオペレーティングシステム(RTOS) を選択できます。

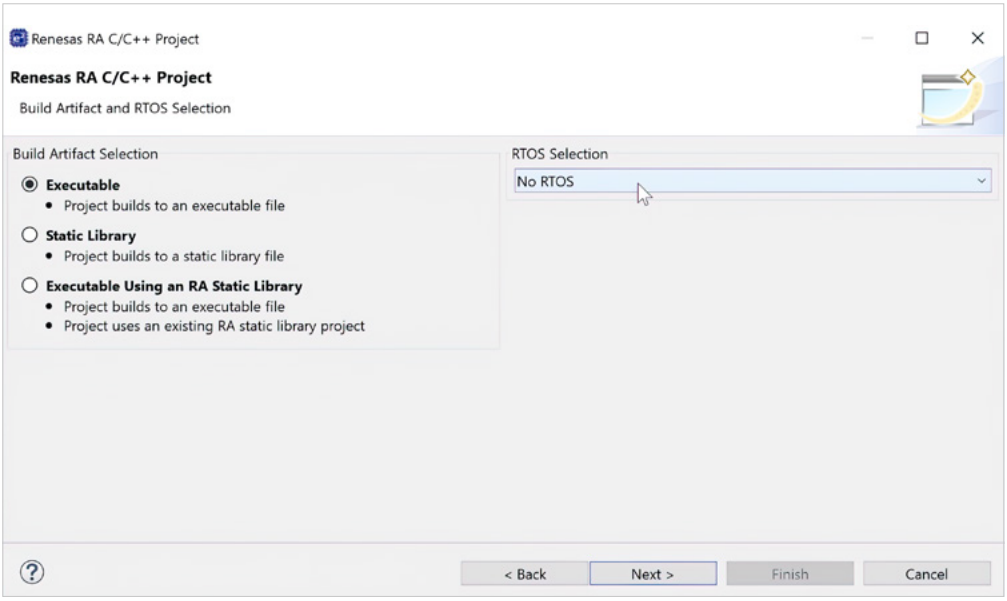

図8-6: RTOS なしで実行可能プロジェクトをビルドします。

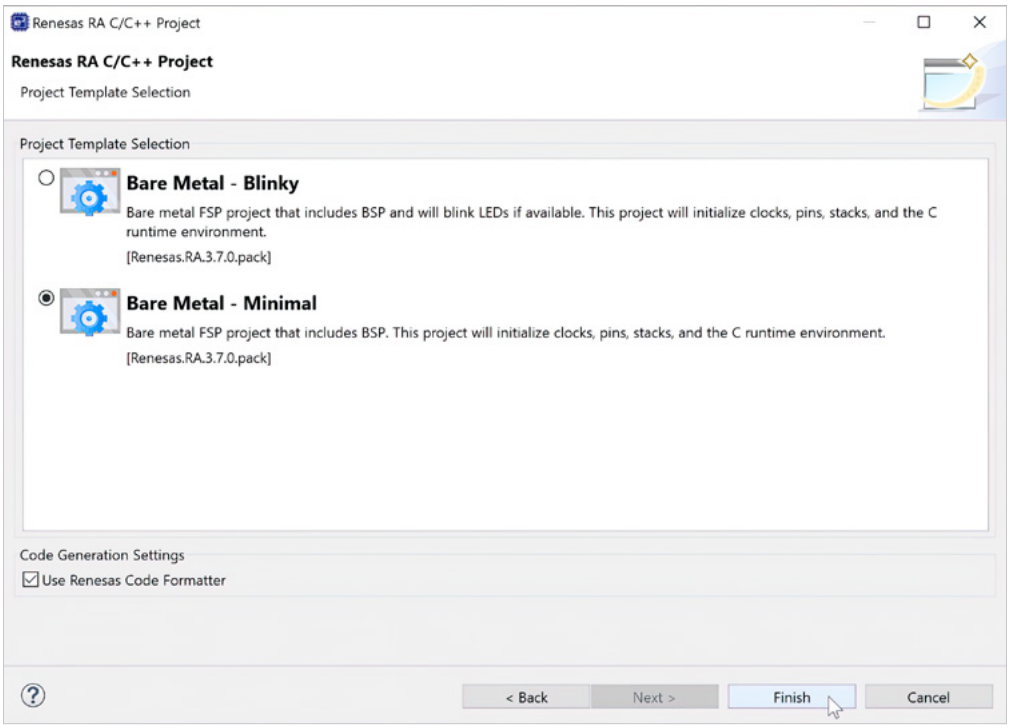

図8-7: 「Project Template Selection」は、プロジェクトの最初のコンテンツで使用可能なテンプレートが表示されます。

この小実習では、[Executable]と[No RTOS]を選択してから、[Next]をクリックします。

プロジェクトテンプレートの選択ページが表示され、最初のプロジェクトコンテンツのテンプレートを選択できます。プロジェクトテンプレートには、複数の項目が含まれている場合があります。少なくとも、選択したボード/デバイスの併用に適したボードサポートパッケージで構成されています。一部のテンプレートには完全なサンプルプロジェクトも含まれていますが、プロジェクトコンフィギュレータには、前の画面で行った選択に一致するテンプレートのみが表示されます。この例では、[Bare Metal – Minimal]エントリを選択します。これにより、評価キットのボードサポートパッケージがロードされます。[Finish] をクリックして、コンフィギュレーションまたはプロジェクトを終了します。

プロジェクトコンフィギュレータが閉じ、最後のステップでプロジェクトに必要なすべてのファイルが作成されます。この後処理が完了すると、FSP Configuration Perspectiveを開くかどうかを尋ねるダイアログが表示されます。[Open Perspective] を選択します。

8.2 FSP コンフィギュレータを使用したランタイム環境の設定

FSPコンフィギュレータが起動すると、プロジェクトの読み出し専用の概略と、選択したソフトウェアコンポーネントの概略が表示されます。また、YouTube™のRenesas RA チャネルへのショートカット、Renesas.com のRenesas Design & Support ページ、Knowredge base とRenesas Rulz Forum、およびハードドライブのFSP ユーザーズマニュアルにアクセスできます。

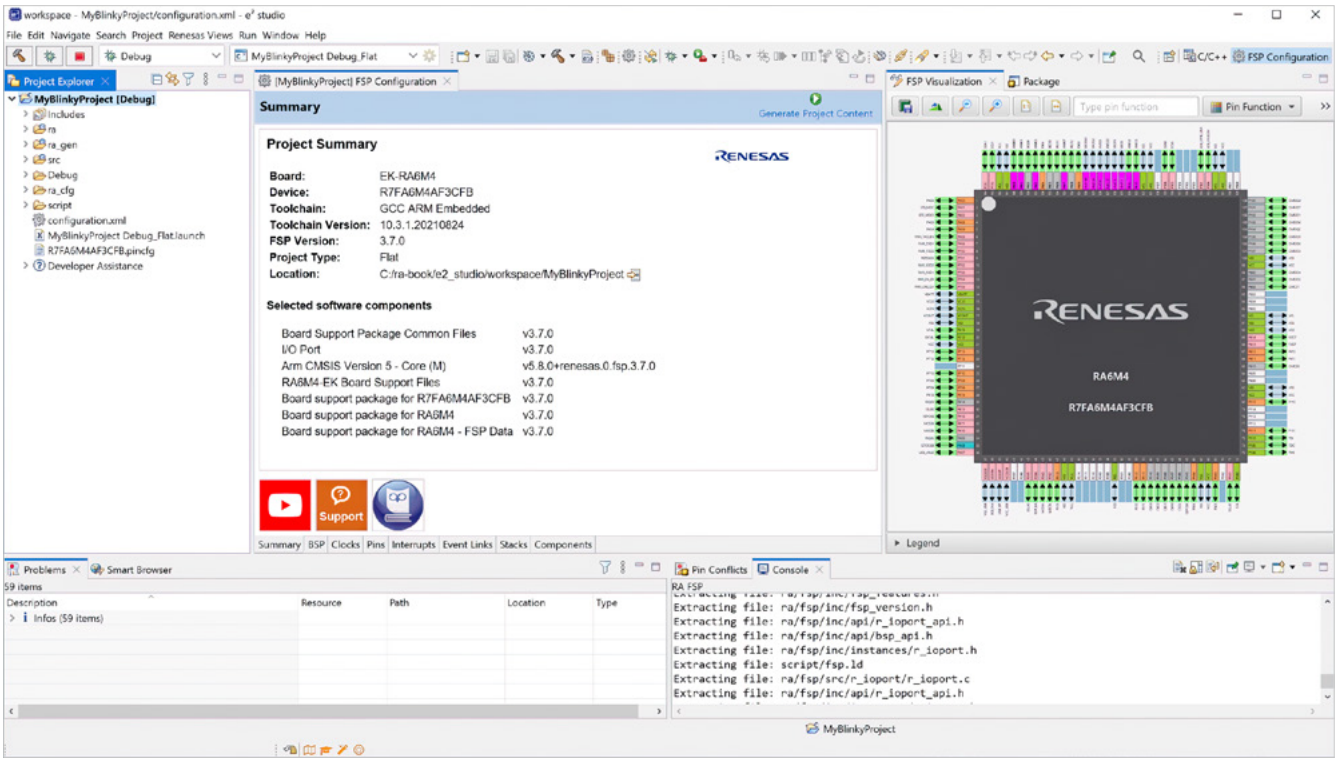

図8-8: e² studio内のFSP Configuration パースペクティブ

BSP と呼ばれる以下のタブでは、ボードやデバイスの選択など、セットアップのいくつかの側面を表示および編集できます。このタブのプロパティビューでは、メインスタックの大きさやMCU の一部のセキュリティ機能など、ボードサポートパッケージの追加設定を行うことができます。以下のClocks タブで、プロジェクトの初期クロック設定を割り当てることができます。潜在的な問題は赤色で強調表示され、ハイライトの上にマウスを置くと、競合または不完全な条件の説明が表示されます。

4 番目のタブであるPinsは、RA MCU のピン割り当てをカバーしています。ポートまたはペリフェラルに基づいてピンを一覧表示できます。コンフィギュレータの右側にあるパッケージビューには、互換性のない設定や欠落している設定がある場合に、設定されたピンとマーキングエラーを強調表示するデバイスのパッケージが表示されます。 これらは、ProblemsビューとPin Conflictsビューにも表示されます。 このようにして、起こりうる間違いを最小限に抑えます。

次の はInterrupts タブです。ユーザー定義(非FSP) ドライバがマイクロコントローラのICU (Interrupt Controller Unit) をどのように使用し、どのISR (Interrupt Service Routine) をICEイベント(割り込み) に結び付けるかを指定できます。また、コンフィギュレータのStacks ビューで作成されたFSP モジュールインスタンスによって生成されたものを含め、割り当てられたすべてのICU イベントの一覧表示をすることもできます。

Event Linksタブも同様の目的に使用できます。ここでは、ドライバがRA プロジェクト内でELC （Event Link Controller） をどのように使用するかを指定し、そのようなドライバがELC イベントのセットを生成するか、一連の周辺機能を介してELC イベントのセットを消費するかを宣言できます。

多くの時間を費やすページはStacks ページです。これによりRTOS スレッドとカーネルオブジェクト、およびFSP ソフトウェアスタックを作成することができます。様々なオブジェクトとモジュールを追加したり、Propertiesビューでプロパティを変更したりできます。これらはすべて、ユーザーの介入が必要なレベルまでは自動的に挿入されます。この場合、マウスをモジュールの上に置くと、注意が必要なモジュールが赤色でマークされ、必要な設定や問題の説明が表示されます。解決した場合、モジュールは標準色に戻ります。Stacksビュー自体には、各種の層がグラフィカルに表示されるため、各種モジュールを簡単に追跡できます。この例では、1つのモジュールを持つ1つのスレッドのみが示されています。g_ioportl/ Oポート（r_ioport）ドライバを備えたHAL/共通スレッドです。 これはProject Configuratorによって自動的に挿入され、数行のコードで点滅するLEDのプログラムを作成できるようになります。

ComponentsはさまざまなFSPモジュールを表示し、それらの選択を許可する最後のタブの名前です。 また、利用可能なRACMSISソフトウェアコンポーネントも一覧表示されます。 ただし、モジュールをそこで構成することもできるため、[Stacks]ページから現在のプロジェクトにモジュールを追加または現在のプロジェクトからモジュールを削除することをお勧めします。

このプロジェクトでは、プロジェクトコンフィギュレータによってすでに設定されているので、FSP コンフィギュレータで必要な設定を変更する必要はありません。最後のステップとして、現在の設定に基づく追加のソースコードを作成する必要があります。FSPコンフィグレータの右上にある [Generate Project Content] ボタンをクリックします。これにより、必要なファイルがFSP から抽出され、コンフィギュレータで行った設定に調整され、プロジェクトに追加されます。

8.3 コードの最初の行の記述

自動生成されたすべてのファイルが配置された状態で、作成されたものを確認することができます。IDE の左側にあるProject Explorer には、現在含まれているすべてが一覧表示されます。ra_gen フォルダは、チャネル番号などの構成を保持します。src ディレクトリは、hal_entry.c という名前のファイルで構成されます。これは後で編集します。ra_gen フォルダにmain.cというファイルがありますが、ユーザーコードはhal_entry.c に移動する必要があります。そうしないと、FSP コンフィギュレータで変更を加えてプロジェクトコンテンツを再作成すると、プロジェクトコンテンツの生成をクリックするたびにこのファイルが上書きされるため、変更内容が失われます。

プロジェクトには、FSP のソース、インクルード、および構成ファイルを含む、名前に「ra」または「fsp」が含まれる複数のディレクトリも含まれます。これらのフォルダ(およびサブフォルダ)の内容は変更しないでください。これらのフォルダには、コンフィギュレータによって生成されたファイルが含まれており、そこで行われた変更は、次にプロジェクトコンテンツが生成または更新されたときに失われます。ユーザーが編集可能なソースファイルは、\srcフォルダーまたはユーザーが追加したその他のフォルダーのルートに直接あるファイルです。

これで、RA Family of microcontrollers の最初の実ソースコードを作成することができます。EK-RA6M4評価キットの緑色のLED2と赤色のLED3を毎秒交互に切り替える計画であるため、それらのオンとオフを切り替えたり、遅延ループを実行したりするためのコードを追加する必要があります。 では、それはどうやって実現するのでしょうか?

実際には、インタフェース機能を使用してAPI を使用する選択肢と、BSP 実装関数を使用する選択肢の2つがあります。どちらがいいと思いますか?分からない場合は、第2章を確認できますので参照してみてください。

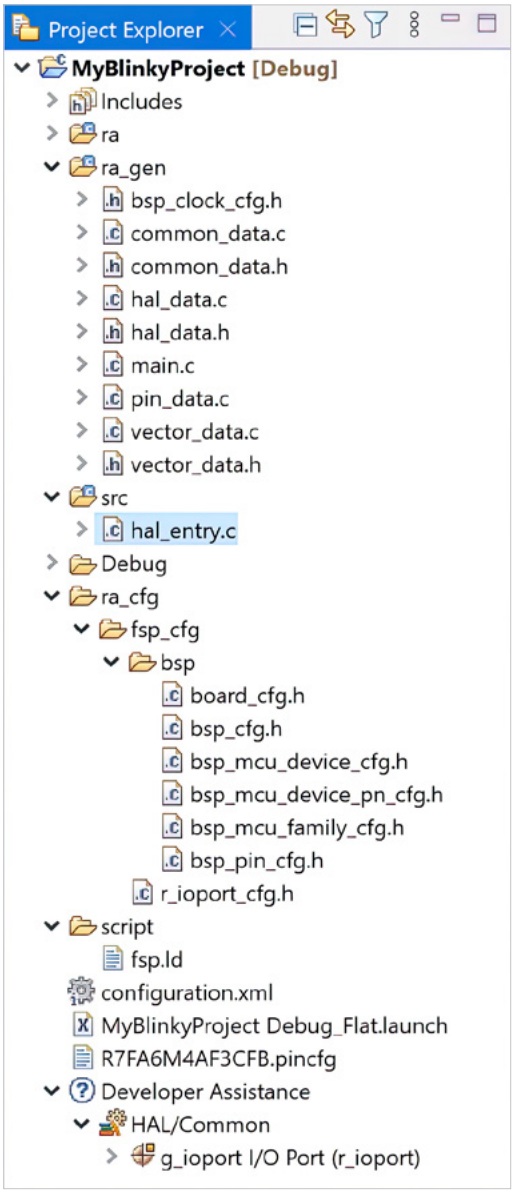

図8–9: FSP Configurator が必要なファイルを作成した後のプロジェクトツリー

ファイルra_gen\common_data.cのコードを見ると、I/Oポートドライバインスタンスg_ioportには次の定義があります:

```
const ioport_instance_t g_ioport = {.p_api = &g_ioport_on_ioport,
                                    .p_ctrl = &g _ioport_ctrl,
                                    .p_cfg = &g_bsp_pin_cfg, };
```

g_ioport_on_ioport は構造体で、ポートに対して可能なアクションを宣言し、g_ioport インスタンスのAPI ポインタに割り当てられます。構造体の上にマウスを置くと、構造体の内容を簡単に表示できます。これにより、そのメンバーの1つである.pinWriteがピン書き込み関数へのポインタであることがわかります。

したがって、LED をオンにすると、次のように記述できます:

```
g_ioport.p_api->pinwrite (&g_ioport_ctrl, pin, BSP_IO_LEVEL_LOW);
```

しかし、これは実際には、どのI/O ポートLED2 とLED3 が接続されているか、および使用可能なLED の数を知る必要があることを意味します。よって、ボードに関するドキュメントを読むか、回路図を精査して正しいポートを見つけます。または、シンプルにFSP に頼ることもできます。bsp_leds_t 式の構造体(board_leds.h で宣言されています) を作成し、board_leds.c で定義されているグローバルBSP 構造体g_bsp_leds を割り当てると、うまくいきます。どちらのファイルもプロジェクトのra\boards\ra6m4_ek フォルダ内に存在します。したがって、評価キットのLED に関する情報を取得するには、次の2 行のコードで十分です:

```
extern bsp_leds_t g_bsp_leds;
bsp_leds_t Leds = g_bsp_leds;
```

これで、LED 構造を使用してボード上のすべてのLED にアクセスでき、次の文で緑色のLED2 をオンにできます(ポートをLOW レベルに設定すると、LED がオンになり、HIGHレベル に設定するとオフになります):

```
g_ioport.p_api->pinWrite(&g_ioport_ctrl,
                         Leds.p_leds[BSP_LED_LED2],
                         BSP_IO_LEVEL_LOW);
```

ピンレベルをハイに設定してLED3 をオフにするには、このステートメントの後に2番目の記述を続ける必要があります。
最後に、LED をユーザーフレンドリーな方法でトグルさせるための遅延を指定する必要があります。そのためには、BSP API を再度呼び出すことができます:

```
R_BSP_SoftwareDelay(1, BSP_DELAY_UNITS_SECONDS);
```

R_BSP_SoftwareDelay 関数の最初のパラメータは遅延するユニット数で、2 番目のパラメータは指定されたユニットのベース(この場合は秒) です。その他のオプションはミリ秒とマイクロ秒です。

一旦、これを行うと必要になるのは、コードの3行をコピー/ペーストし、2番目のセットのLEDのピンレベルを反転させることだけです。最後に、プログラムを無期限に実行したいので、コードの周りにwhile(1)ループを作成する必要があります。

ここで残されているのは、hal_entry.c に次のコード行を入力することです。 関数シグネチャの直後に/* TODO: add your own code here */ 行。Project ConfiguratorとFSP Configuratorによって挿入された他のコードはそのままにしておいてください。マイクロコントローラが正しく動作するために必要です。

```
extern bsp_leds_t g_bsp_leds;
bsp_leds_t Leds = g_bsp_leds;

 while (1)
 {
     g_ioport.p_api->pinWrite(&g_ioport_ctrl,
                              Leds.p_leds[BSP_LED_LED2],
                              BSP_IO_LEVEL_LOW);
     g_ioport.p_api->pinWrite(&g_ioport_ctrl,
                              Leds.p_leds[BSP_LED_LED3],
                              BSP_IO_LEVEL_HIGH);
     R_BSP_SoftwareDelay(1, BSP_DELAY_UNITS_SECONDS);

     g_ioport.p_api->pinWrite(&g_ioport_ctrl,
                              Leds.p_leds[BSP_LED_LED2],
                              BSP_IO_LEVEL_HIGH);
     g_ioport.p_api->pinWrite(&g_ioport_ctrl,
                              Leds.p_leds[BSP_LED_LED3],
                              BSP_IO_LEVEL_LOW);
     R_BSP_SoftwareDelay(1, BSP_DELAY_UNITS_SECONDS);
}
```

コードの記述中は、いつでもe² studio のオートコンプリート機能を使用できます。<ctrl>-<space>を押すだけで、組織や関数の補完候補を表示する画面が表示されます。エントリをクリックすると、自動的にコードに挿入されます。

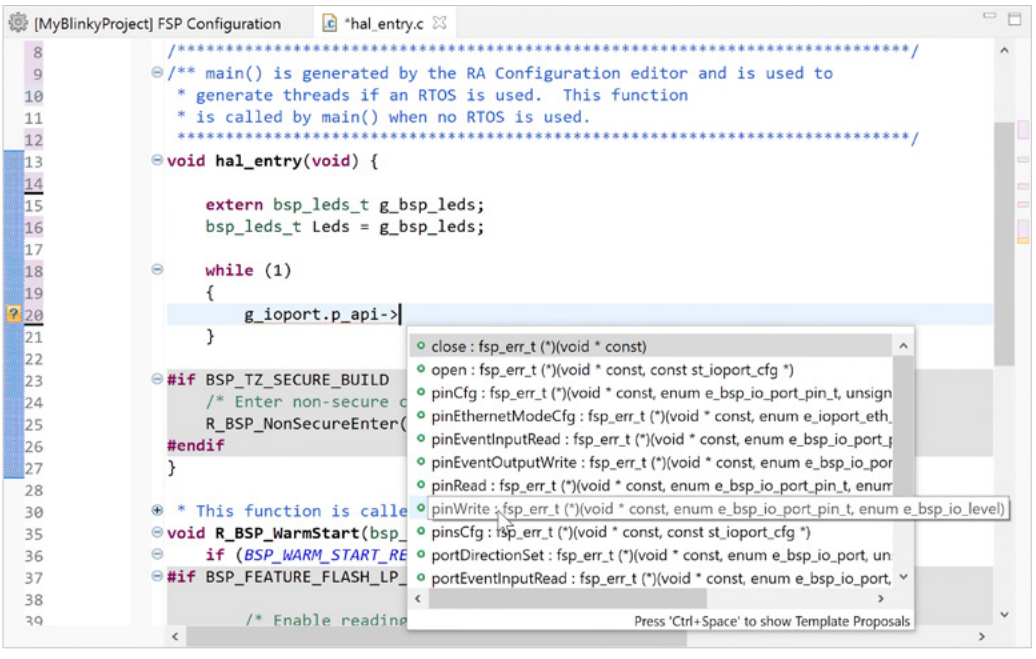

図8-10: 変数または関数で<ctrl>-<space> を押すと、e² studioのコード補完機能が有効になります。

プログラムの作成中に役立つもう1つのツールは、プロジェクトエクスプローラからアクセスできる開発者支援です。FSP Configurator を使用してプロジェクトのソフトウェアスタックを設定した後、このツールはアプリケーションコードの使用をすばやく開始することをサポートします。開発者支援にアクセスするには、まずプロジェクトエクスプローラを展開して、ツールが表示されるようにします。ツールが表示されたら、スタックモジュールとそのAPI が表示されるまでツリーをさらに展開します。使用するAPIを選択し、その呼び出しをソースファイルにドラッグアンドドロップします。

これで順番は変わりました。hal_entry.c に上記のコード行を入力してください。 プロジェクト内のファイルそのためには、プロジェクトのsrc フォルダを展開し、ファイルをダブルクリックします。エディタで開きます。すべてを自分で入力したくない場合は、本書のウェブサイト(**www.renesas.com/ra-book**) から完全なプロジェクトをダウンロードすることもできます。

8.4 最初のプロジェクトのコンパイル

すべての入力が完了すると、プログラムをビルドする準備が整います。ビルドには、デバッグとリリースの2 つの設定があります。デバッグ構成には、変数名や関数名など、プログラムのデバッグに必要なすべての情報が含まれ、ループ展開など、コンパイラの一定の最適化もオフになります。これにより、デバッグが容易になりますが、コードは大きくなり、低速になります。解除設定では、出力ファイルからこの情報がすべて削除され、完全な最適化が有効になります。これにより、コードが小さく高速化されますが、たとえば、メモリ内のアドレスがわからない限り、変数を表示することはできなくなります。

最初のテストでは、デバッグ 構成(デフォルトでもある) が使用されます。プロジェクトをビルドするには、メインメニューバーのビルドボタン  をクリックすると、プロセスが開始されます。すべてを正しく実行すると、コンパイルはエラーとワーニングが"0"で終了します。コンパイル時のエラーがある場合は、コードに戻ってすべてを正しく入力したかどうかを再確認する必要があります。正しく入力されていない場合は、それに応じてコードを変更します。エラーを見つけやすくするために、コンパイラのフィードバックがエディタウィンドウに直接挿入されます(挿入可能な場合のみ)。

プログラムのビルドが成功すると、出力ファイルMyBlinkyProject.elf が作成されます。これは、実行してデバッグする前にプロセッサにダウンロードする必要があります。

8.5 最初のプロジェクトのダウンロードとデバッグ

次のステップは、評価キット(EK) でプログラムを実際に実行することです。これで、キットをWindows®ワークステーションに接続できるようになりました。ボードに同梱されているUSBケーブルのmicro-B側を、システム制御およびエコシステムアクセスエリアの右下側にあるUSBデバッグポートJ10に挿入し、もう一方の端をPCの空いているポートに挿入します。「EK-RA6M4」の文字でハイフンを形成している白色のLED4 が点灯し、ボードの電源が入っていることを示します。キットが箱から出たばかりの場合、事前にプログラムされたデモが実行され、すべてが期待どおりに機能していることを通知します。PC画面では、J-Link®オンボード・デバッガのドライバのインストールを示すダイアログが表示される場合があります。これは自動的に完了します。また、J-Link® Debug Probeの更新を要求する画面が表示される場合があります。この更新を実行できるようにすることを強くお勧めします。

しばらくしてもUSB ポート以外のオレンジ色のデバッグLED5 の点滅が停止しない場合は、ワークステーションのJ-Link®ドライバに問題がある可能性があります。その場合は、7.1 章を参照して、考えられる解決策を確認してください。

DOWNLOAD

プログラムをダウンロードするには、まずデバッグ構成を作成する必要があります。デバッグ 記号の横にある小さな矢印 をクリックし、ドロップダウンリストボックスからDebug Configurations を選択します。

表示されたウィンドウで、Renesas GDB Hardware Debugging の下のMyBlinkyProject Debug_Flat を強調表示します。Project configuratorで必要な設定はすべてすでに行われているため、このダイアログでは何も変更しません。画面の右下隅にあるデバッグボタンをクリックするだけです。これにより、デバッガが起動し、コードがEK上 のRA6M4 MCUにダウンロードされ、デバッグパースペクティブに切り替えるかどうかを尋ねられます。ここではスイッチで応答します。Debug perspectiveが開き、プログラムカウンタがプログラムのエントリポイントであるリセットハンドラに設定されます。このデバッグ設定は一度だけ作成する必要があります。次回デバッグシンボル をクリックするだけで、デバッガを起動できます。

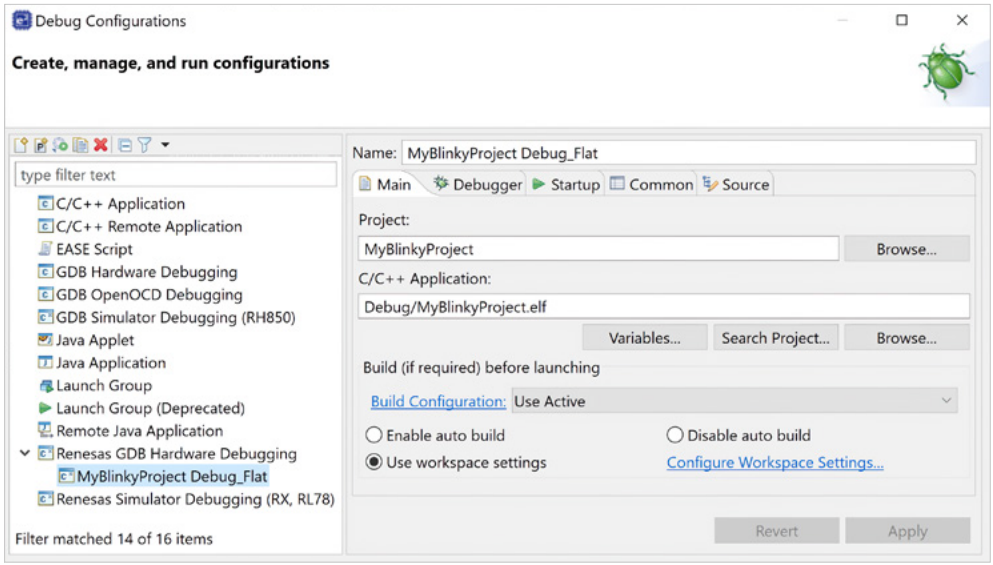

図8-11: MyBlinkyProject Debug_flat を選択したら、複数のタブで変更を加える必要はありません。

RUNNING

再開 ボタンをクリックすると、次の停止はmain() で、hal_entry() の呼び出しで行われます。ボタンをもう一度クリックすると、プログラムは実行を継続し、意図したとおりに1 秒間隔で評価キットの緑色と赤色のLED を切り替えます。

WATCHING THE RESULTS

すべてが期待どおりに機能している場合は、メインメニューバーのSuspend ボタン をクリックします。終了はせずにプログラムの実行を停止します。エディタビューで、ファイルhal_entry.c を使用してタブをアクティブ化し、 いずれかのラインを右クリックして、ポートに書き込みます。表示されるメニューで、ライン単位で実ラインを選択します。実ラインが再開され、クリックしたラインでプログラムが停止します。次に、右側の変数を使用してビューを確認します。Leds 構造が一覧表示されます。展開して、さまざまな分野を参照して分析します。このビューは、より大きなプロジェクトをデバッグする場合に便利です。

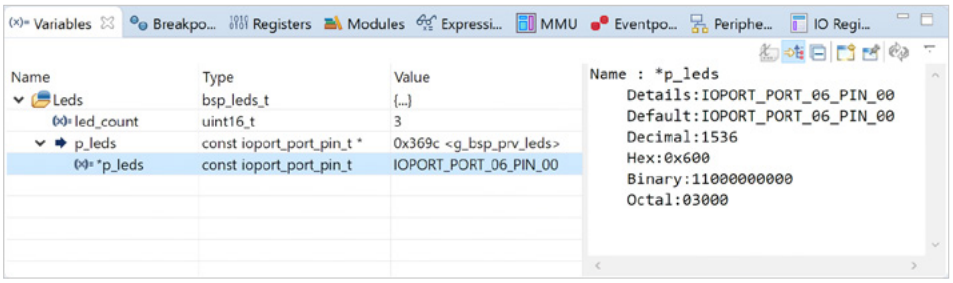

図8-12: 変数ビューで変数とその値を確認できます。

最後の手順は、Terminate ボタン をクリックしてデバッグセッションを終了し、プログラムの実行を停止することです。

おめでとうございます。

RAファミリの最初のプログラムをマスターしました。

本章で学んで会得するポイント:

- プロジェクトコンフィギュレータは、新規プロジェクトに必要なすべてのファイルと設定を作成します
- FSP コンフィギュレータを使用すると、プログラマはグラフィカル･ユーザ･インタフェースに基づいてFSPとランタイム環境を簡単に設定できます
- プロジェクトをデバッグするには、デバッグ構成が必要です。自動的に作成され、有効化する必要があります
- 必要な機能の実装に必要なコード行はごく少ないものです

9　リアルタイム･オペレーティング･システムの組込み

この章で学ぶこと:

- スレッド、セマフォ、キューとは何か。また、その使い方
- e² studioでスレッドとセマフォをプログラムに追加する方法
- RTOS上にプッシュボタンでLEDをトグルさせるプログラムを実装する方法

前章の演習では、すでにルネサスのマイクロコントローラ(MCU)のRAファミリ用フレキシブルソフトウェアパッケージ (FSP)の優れた部分を使用してみました。この章では、LED 用のスレッドを利用したFreeRTOS™リアルタイムオペレーティングシステムと、プッシュボタンと同期するためのセマフォを使用して、小規模なアプリケーションを作成します。これには、実際に必要な手順はほとんどありません。

ここでは最初から完全なプロジェクトを作成しますので、以前の演習を行っていない場合も心配しないでください。

9.1　スレッド、セマフォ、キュー

実際にこの演習を行う前に、この章と次の章で使用する用語をいくつか定義して、すべての用語が共通の理解の元で使用されるようにします。
まず、「スレッド」という用語を定義する必要があります。「タスク」という表現に慣れている場合は、スレッドを一種のタスクと考えて構いません。また、両方のフレーズを同じ意味で使用するものもあります。リアルタイムオペレーティングシステム(RTOS) を使用する場合、マイクロコントローラ上で実行されるアプリケーションは、いくつかの小さな半独立したコードの塊に分割され、それぞれがプログラムの単一の側面を通常制御します。これらの小さな部分はスレッドと呼ばれます。1つのアプリケーション内に複数のスレッドを存在させることができますが、RA ファミリマイクロコントローラはシングルコアデバイスであるため、一度にアクティブにできるのは1 つだけです。各スレッドには独自のスタック空間があり、セキュアなコンテキストが必要な場合は、MCU のセキュア側に配置できます。すべてのスレッドには、アプリケーション内の他のスレッドに関して割り当てられた優先順位もあり、running、ready、blocked、suspended などの異なる状態にすることができます。FreeRTOS™では、例えば eTaskGetState() API 関数を呼び出してスレッドの状態を照会できます。スレッド間のシグナリング、同期または通信は、セマフォ、キュー、ミューテックス、通知、タスクへの直接通知、またはストリームとメッセージバッファによって実行されます。Azure RTOSも似たような機能を提供します。

セマフォは、RTOS のリソースで、イベントのシグナリングとスレッド同期(生産者/消費者方式) に使用できます。セマフォを使用すると、アプリケーションはイベントが発生してセマフォが送信されるまでスレッドをサスペンドできます。RTOS を使用しない場合は、フラグ変数を常にポーリングするか、割り込みサービスルーチン(ISR) 内で特定のアクションを実行するコードを作成し、その他の割り込みをかなりの時間ブロックする必要があります。セマフォを使用すると、ISR をすばやく終了し、関連するスレッドに操作を延期することができます。

FreeRTOSとAzure RTOSは、カウントセマフォとバイナリセマフォを提供します。 バイナリセマフォは0と1の2つの値しか想定できないため、タスク間または割り込みとタスク間の同期を実装するのに理想的ですが、カウントセマフォのカウントは0から最大カウントまでです。 FSPコンフィギュレータのセマフォはデフォルトで256なので、設計者はより複雑な同期操作を実行できます。

各セマフォには2つの基本操作が関連付けられています。xSemaphoreTake ()（FreeRTOSの場合）またはtx_semaphore_get ()（Azure RTOSの場合）はセマフォを1つ減らし、xSemaphoreGive () またはtx_semaphore_set () はセマフォを1つ増やします。。どちらの関数も、割り込みサービスルーチン(xSemaphoreTakeFromISR() およびxSemaphoreGiveFromThread()) 内から呼び出すことができるものと、スレッドの通常のコンテキストから呼び出すことができるものの2つの方法があります。

最後に話をする必要がある用語は、この演習では使用しませんが、次章の演習では使用するキューです。メッセージキューは、スレッド間通信の主要な方法であり、タスク間または割り込みとタスク間でメッセージを送信できるようにします。1 つ以上のメッセージをメッセージキュー内に置くことができます。より大きなバッファへのポインタになることもあるデータは、キューにコピーされます。つまり、参照ではなくメッセージ自体が格納されます。 新しいメッセージは通常、キューの最後に配置されますが、先頭に直接送信することもできます。 受信したメッセージは前面から削除されます。

許容されるメッセージサイズは、設計時にFSPコンフィギュレータを介して指定されます。デフォルトのサイズは4 バイトで、キューに格納できる項目数を表すデフォルトのキュー長は20 です。すべての項目は同じサイズである必要があります。FreeRTOS のキュー数に制限はありません。唯一の制限は、システムで使用可能なメモリです。メッセージはxQueueSend() 関数を使用してキューに入れられ、xQueueReceive() によってキューから読み取られます。セマフォと同様に、関数には2つのバージョンがあります。1つはスレッドのコンテキストから呼び出すことができるもので、もう1つはISR の内部から呼び出すことができるものです。またAzure RTOSでは、2つの関数はそれぞれtx-qeue_send () およびtx_queue_receive () と呼ばれます。

9.2 e² studioを使用したFreeRTOS へのスレッドの追加

以下の演習は、再びEK-RA6M4 評価キットに基づいています。今回は、ボードの左上にある青色のプッシュボタンS1 を使用して、アプリケーションにイベントを通知し、それに応じて緑色のLED2 を切り替えます。実装にはFreeRTOS を使用し、イベントの処理はセマフォを介して通知されるスレッド内で行われます。

通常、最初の手順は、第4 章と第8 章で既に演習したプロジェクトコンフィギュレータを使用して新しいプロジェクトを作成することです。開始するには、File → New → Renesas C/C++ Project → Renesas RAに移動します。表示されたウィンドウのサイドバーでRenesas RA を選択し、Renesas RA C/C++ Project エントリを強調表示します。「次へ」をクリックし、表示される画面にプロジェクト名(MyRtosProject など) を入力します。ここでも、[Next]をクリックします。デバイスとツールの選択ウィンドウが表示されます。まず、ボードを選択します。リストにない場合は、EK-RA6M4 を選択し、対応するデバイスをR7FA6M4AF3CFB に設定します。ツールチェーンを参照してください。GCC Arm® Embedded と表示されているはずです。[Next] をクリックして続行します。

TrustZone®以外とセキュアでないTrustZone プロジェクトを選択できるようになりました。Flat (Non-TrustZone) Projectを選択したまま、[Next]をクリックします。Build ArtifactとRTOS Selectionのウィンドウが表示されます。設定はそのままにしておきます。つまり[Build Artifact Selection]で[Exacutble]を選択し および[RTOS Selection]で[Free RTOS]を選択します。[Next] をクリックして、Project Template Selection という名前の次の画面に移動します。ここでは、FreeRTOS - Minimal - Static Allocation を選択します。

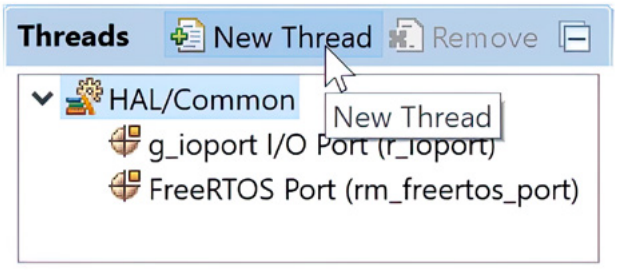

図9-1: FSP コンフィギュレータが表示された後、1 つのスレッドのみが表示されます。[New Thread] ボタンを選択して別のスレッドを追加します。

最後に、[Finish]をクリックし、プロジェクトがコンフィギュレータによって生成された後、e² studio からFSP Configuration Perspectiveに切り替えるよう求められます。Perspectiveが表示されたら、[Stacks] タブに直接移動します。このタブには、HAL/共通スレッドの2つのエントリが表示されます。スレッド枠：1つはI / Oポート用のドライバーを含み、もう1つはFreeRTOS用です。枠の上部にある[New Thread] アイコンをクリックします。(新しいスレッドを追加するには、図9-1 を参照してください)。

Properties ビューで新しいスレッドのプロパティを変更します。Symbol の名前をled_thread に、Name をLED Thread に変更します。その他のプロパティはデフォルト値のままにします。LED スレッドスタックペインで、New Stack ボタンアイコンをクリックし、r_icu で External IRQ Driver(r_icu) を選択します(図9-2 を参照)。

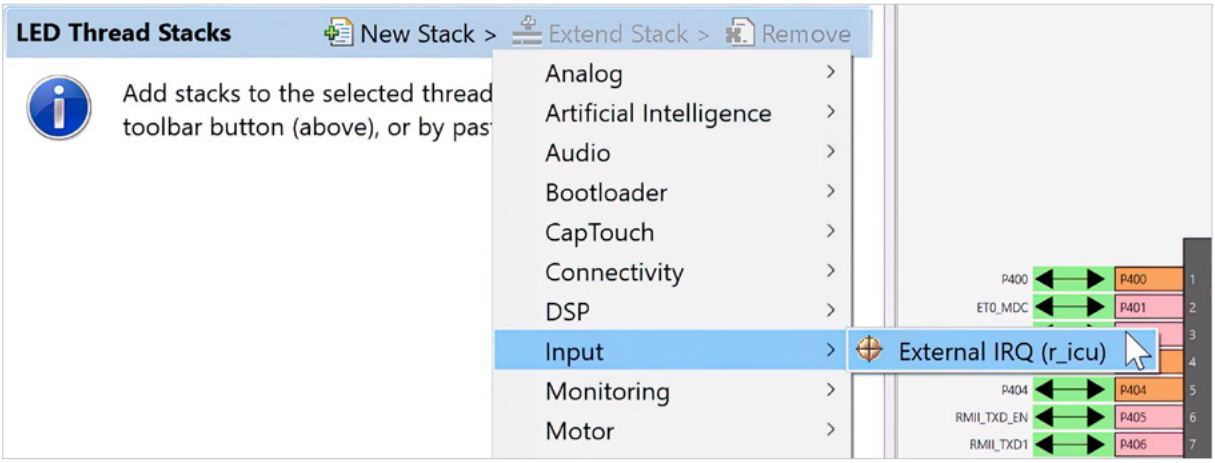

図9-2: 新しいドライバの追加には、マウスを数回クリックするだけです。

これにより、外部割り込み用のドライバが追加されます。新しいドライバのプロパティを確認し、いくつかの変更を行います。まず、S1が接続されているピンがIRQ10に接続されているため、[Channel]を[0] から[10] に変更します。同じ理由で、名前をg_external_irq10、またはその他の好きな名前に変更します。

唯一の割り込みにはプライオリティ12 が割り当てられており、起動時にFSP によって有効にされることはありません。他の優先度も選択できますが、大規模なシステムであっても、割り込み優先度の衝突が発生することはほとんどないため、まず12を設定することをお勧めします。優先度15 はシステムティックタイマ(systick)用に予約されているため、他の割り込みでは使用しないでください。

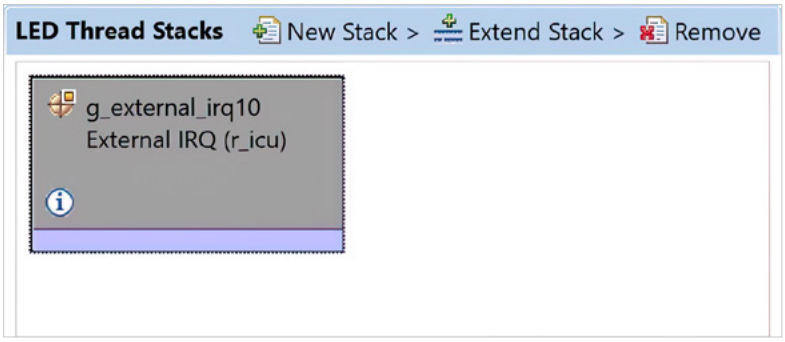

図9-3: スタック要素のグレーのカラーバーは、このドライバがモジュールインスタンスであり、他のFSP モジュールインスタンスによってのみ参照される可能性があることを示しています。

[Trigger]を[Rising]から[Falling]に変更し、ボタンが押されたことを認識するようにして、[Digital Filtering]を[Disabled]から[Enabled]に変更します。Digital Filtering Sample Clock の設定はPCLK/64 のままにしてください。これはボタンのデバウンスに役立ちます。最後に、コールバック行のNULL をexternal_irq10_callback に置き換えます。この関数は、S1 を押すたびに呼び出されます。コールバック関数自体のコードは、後でアプリケーションの作成時に追加します。図9-4 に、必要な設定の概要を示します。

Property	Value
Common	
Parameter Checking	Default (BSP)
Module g_external_irq10 External IRQ (r_icu)	
Name	g_external_irq10
Channel	10
Trigger	Falling
Digital Filtering	Enabled
Digital Filtering Sample Clock (Only valid when Digit	PCLK / 64
Callback	external_irq10_callback
Pin Interrupt Priority	Priority 12
Pins	
IRQ10	P005

図9-4: アプリケーションに必要なIRQ ドライバのプロパティ

これで、プログラムをコンパイルしてダウンロードできるようになるまで、実行する必要があるステップがいくつか残りました。次に、セマフォを追加します。

そのためには、[LED Thread Objects」ペインの[New Object」ボタンをクリックします。このペインが表示されず、[HAL/Common Objects]ペインが表示される場合は、[Threads]ペインの[LED Thread]を強調表示し、表示されるようになります。ボタンを押すと、LED Thread(バイナリセマフォ)に通知する必要があります。セマフォのSymbol プロパティをg_s1_semaphore に変更し、Memory Allocation をStatic のままにします。FSPコンフィギュレータ のStacks タブは図9-5 のようになります。

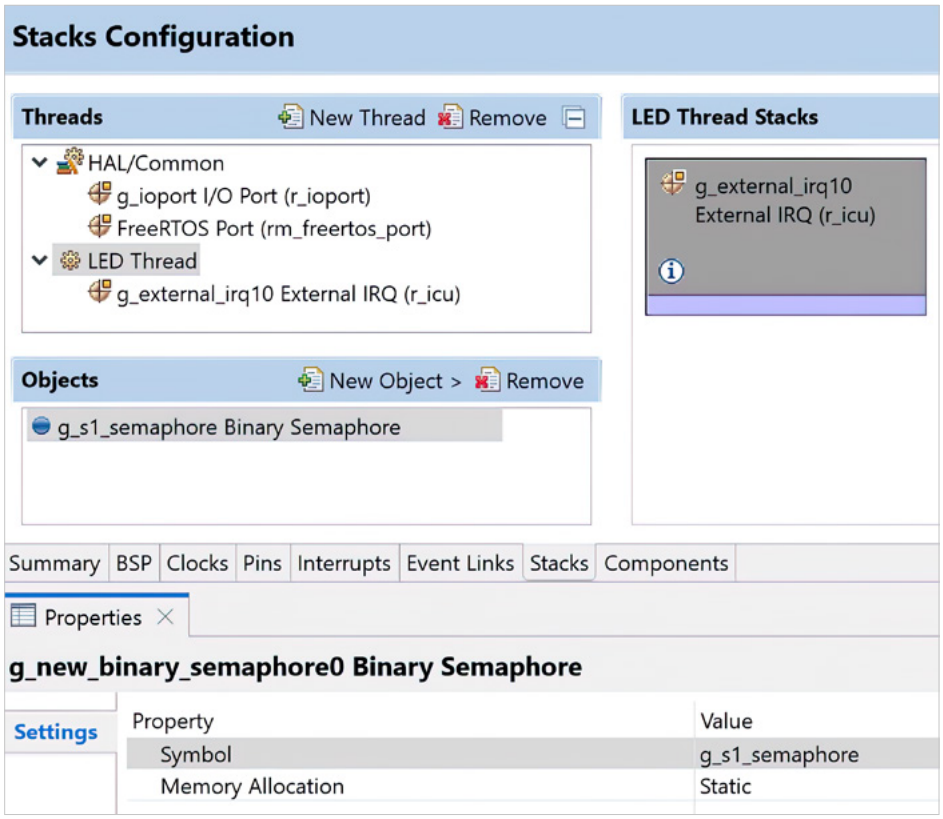

図9-5: LED スレッドとセマフォが追加された後のStacks タブは次のようになります。

FSP コンフィギュレータの最後の手順は、S1がIRQ10入力として接続されているI/Oピンを設定することです。そのためには、コンフィギュレータ内部のPins タブをアクティブにし、Ports → P0を展開してP005を選択します。RA6M4 評価キットでは、これはポートS1 が接続されています。右側のPin Configuration ペインでシンボリック名S1を指定し、その他の設定が図9-6 の設定と同じであることを確認します。通常は、コンフィギュレータによってすでに設定されているはずです。調整しない場合は調整します。右側のパッケージビューアでピン135/P005 が強調表示され、ピンの位置をグラフィカルに参照できるようになります。

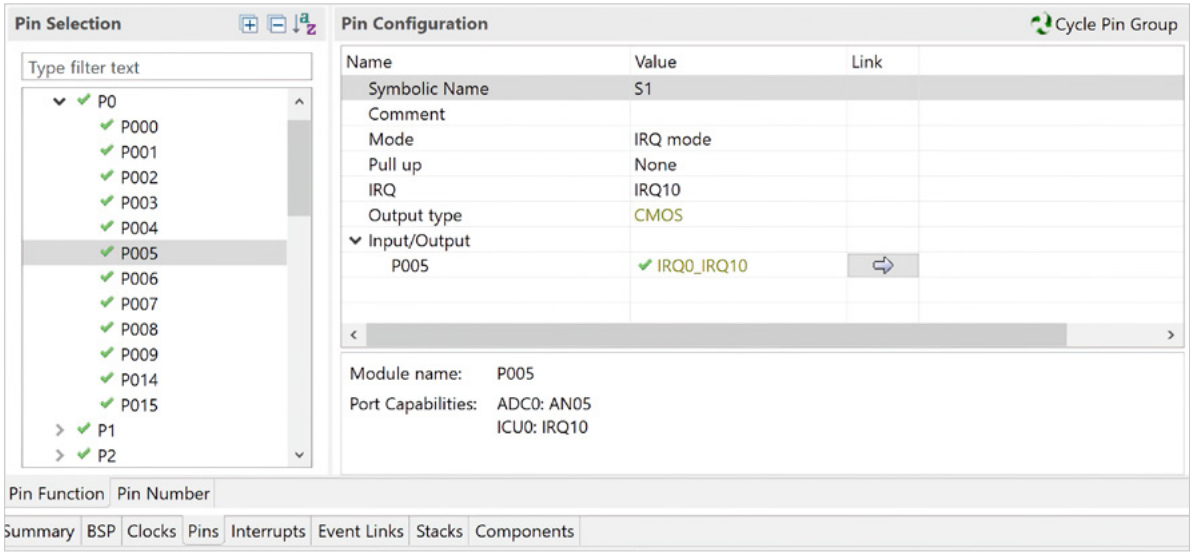

図9-6: ポートP005 はIRQ10 として適切に設定されているはずです。

これで、コンフィギュレータでの設定は完了です。変更を保存し、上部にあるプロジェクトコンテンツの生成アイコンをクリックして、必要なファイル、フォルダ、設定を作成します。

最後に実行する必要があるタスクは、Leds構造の初期化に必要なコードを追加し、LED をトグルしてセマフォを読み取るための行をいくつか記述し、セマフォを設定するコールバック関数を作成することです。このための完全なコードは、この章の最後で確認できます。

押しボタンの処理とLED のトグルにLED Threadを使用しているため、今回は関連するコードをled_thread_entry.c ファイルに追加する必要があります。Project Explorer でファイル名をダブルクリックしてエディタで開きます。表示されていない場合は、プロジェクトフォルダーを展開してからsrc ディレクトリを展開します。第8 章の演習と同様に、LED の構造を追加し、初期化します。LED2 が接続されているI/O ピンのレベルの別の変数を定義する必要があります。led_level という名前を付けます。ioport_level_t 型である必要があり、IOPORT_LEVEL_HIGH に初期化する必要があります(“high” レベルはEK-RA6M4 の”on” に対応します)。

次の手順は、ボード上のS1 に接続されているIRQ10 を開いて有効にすることです。そのためには、IRQ FSP ドライバのopen およびenable 機能を使用します。これで初期化は完了です。

```
g_external_irq10.p_api->open(g_external_irq10.p_ctrl,
                             g_external_irq10.p_cfg);
g_external_irq10.p_api->enable(g_external_irq10.p_ctrl);
```

while (1) ループ内で、いくつかのステートメントを追加し、vTaskDelay(1); ステートメントを削除する必要があります。led_level の値をLED2 のI/O ピンの出力レジスタに書き込むための関数呼び出しから開始し、その後にピンレベルをトグルする記述が続きます。そのためには、いくつかの方法があります。独自に実装するか、第8章の演習を確認するか、章の最後にあるコードを調べてください。e² studio のSmart Manual 機能はとても役に立ちますので、覚えておいてください。

while(1) ループ内の最後の文は、セマフォの住所と定数portMAX_DELAY をパラメータとして指定したxSemaphoreTake() の呼び出しです。後者は、IRQ 10 割り込みサービスルーチンによって呼び出されたコールバック関数からセマフォが送信されるまで、RTOS にスレッドを無期限にサスペンドするように通知します。

最後に行うことは、コールバック関数自体を追加することです。この関数は、割り込みサービスルーチンのコンテキストで実行されるので、できるだけ短くする必要があります。この関数の記述は簡単です。Project Explorer のr_icu で開発者アシスタンス→ のLED スレッド→ g_external_irq10 外部IRQ ドライバに移動し、表示される一覧の最後にあるコールバック関数定義をソースファイルにドラッグアンドドロップします。

```
void external_irq10_callback(external_irq_callback_args_t *p_args)
```

コールバック関数内に、以下の2 行のコードを追加します:

```
FSP_PARAMETER_NOT_USED(p_args);
xSemaphoreGiveFromISR(g_s1_semaphore, NULL);
```

最初の行のマクロは、コールバック関数がパラメータp_args を使用せず、コンパイラからの警告を回避することをコンパイラに伝えます。2 番目のマクロは、ボタンS1 が押されるたびにセマフォを設定します。この関数呼び出しはISR のコンテキスト内で行われるため、interrupts-save バージョンのgev-function を使用する必要があることに注意してください。この呼び出しの2 番目のパラメータは*pxHigherPriorityTaskWoken です。セマフォが1 つ以上のタスクをブロックし、セマフォが使用可能になるのを待機している可能性があり、割り込みが発生したときに実行されたものよりもこれらのタスクの1 つの優先度が高い場合、このパラメータはxSemaphoreGiveFromISR() の呼び出し後にtrue になります。その場合、割り込みが終了する前にコンテキストの切り替えを実行する必要があります。この例では、このセマフォに依存する他のタスクがないため、このパラメータをNULL に設定できます。

すべてのコーディングが完了したら、「Build」アイコン(「ハンマー」)をクリックしてプロジェクトをビルドします。エラーがゼロでコンパイルされない場合は、プログラムに戻り、「Problems」ビューに表示されているコンパイラーのフィードバックの助けを借りて問題を修正してください。

プロジェクトが正常にビルドされた場合は、Debugアイコンの横にある小さな矢印をクリックし、Debug Configurationsを選択してRenesas GDB Hardware Debuggingを展開します。MyRtosProject Debug_FlatまたはプロジェクトにIt付けた名前を選択し、Debugをクリックします。デバッガが起動します。詳細は、第8章の関連セクションを参照してください。デバッガが起動して実行されたら、再開を2 回クリックします。これでプログラムが実行され、EKボード でS1 を押すたびに、緑色のLED2 がトグルします。

注意事項：実際のアプリケーションでは、プログラムの適切な動作を保証するために、エラーチェックを実行する必要があります。この例ではわかりやすく簡潔にするために省略されています。

```
#include “led_thread.h”

 void led_thread_entry(void *pvParameters)
 {
    FSP_PARAMETER_NOT_USED (pvParameters);
    extern bsp_leds_t g_bsp_leds;
    bsp_leds_t Leds = g_bsp_leds;

    uint8_t led_level = BSP_IO_LEVEL_HIGH;

    g_external_irq10.p_api->open(g_external_irq10.p_ctrl,
                                 g_external_irq10.p_cfg);
    g_external_irq10.p_api->enable(g_external_irq10.p_ctrl);

    while (1)
    {
        g_ioport.p_api->pinWrite(&g_ioport_ctrl
                                 Leds.p_leds[BSP_LED_LED2],led_level);
        if (led_level == BSP_IO_LEVEL_HIGH)
        {
            led_level = BSP_IO_LEVEL_LOW;
        }
        else
        {
            led_level = BSP_IO_LEVEL_HIGH;
        }

        xSemaphoreTake(g_s1_semaphore, portMAX_DELAY);
    }
 }

 /* callback function for the S1 push button; sets the semaphore */
 void external_irq10_callback(external_irq_callback_args_t *  p_args)
 {
    FSP_PARAMETER_NOT_USED(p_args);
    xSemaphoreGiveFromISR(g_s1_semaphore, NULL);
 }
```

おめでとうございます。

この練習を無事に終了しました!

本章で学んで会得するポイント:

- FSPの機能は、包括的なAPIを利用することで容易に利用できます
- FSPは、ほとんどの非ユーザコード関連を処理します
- FreeRTOS™の使用は簡単で、FSPコンフィグレータは直感的に使用できるため、スレッドとセマフォの追加はほとんど労力を要しません

10 フレキシブルソフトウェアパッケージを使用したUSB経由のデータ送信

この章で学ぶこと:

- フレキシブルソフトウェアパッケージ のミドルウェアを使用してUSB 転送を設定する方法
- MCU が送信したデータをホストワークステーションで受信する方法

本書のこの部分では、ルネサスRA ファミリのマイクロコントローラ用フレキシブルソフトウェアパッケージ (FSP) のUSB ミドルウェアを使用して、ユーザーボタンS1 を押すたびにUSB ポートを介して現在のLED2 の状態をテキスト文字列としてWindows®ワークステーションに送信します。9 章とは異なり、この演習ではリアルタイムオペレーティングシステムとセマフォを使用しませんが、グローバル変数を使用して、プッシュスイッチがアクティブ化され、緑色のLED2 の状態が変更されたことを示します。

LED状態の更新 (ONまたはOFF) とUSBポートへの書き込みは、IRQ10のコールバックルーチンで行われ、ボタンが押された情報を保持するグローバル変数の更新が行われます。ポートへの書き込みは、ホストにLEDに関する情報を送信する、ディスク転送のトリガーとなります。hal_entry() 関数内の無限ループに戻ると、USB のイベントが処理され、グローバル変数をfalse に設定し、次の文字列と次のLED レベルをそれぞれの変数に割り当てることによって準備されたLED 状態の次の更新が準備されます。図 10-1 に、プログラムの流れと割り込みのコールバック関数の詳細を示します。

ポートの設定のほとんどはFSPコンフィギュレータでグラフィカルに行われるため、アプリケーションプログラマが記述する必要があるコードはほとんどありません。この演習をプログラミングすると、USBのような複雑な通信システムを構築する場合でも、FSPがユーザに提供する使いやすさを再び体験することができます。

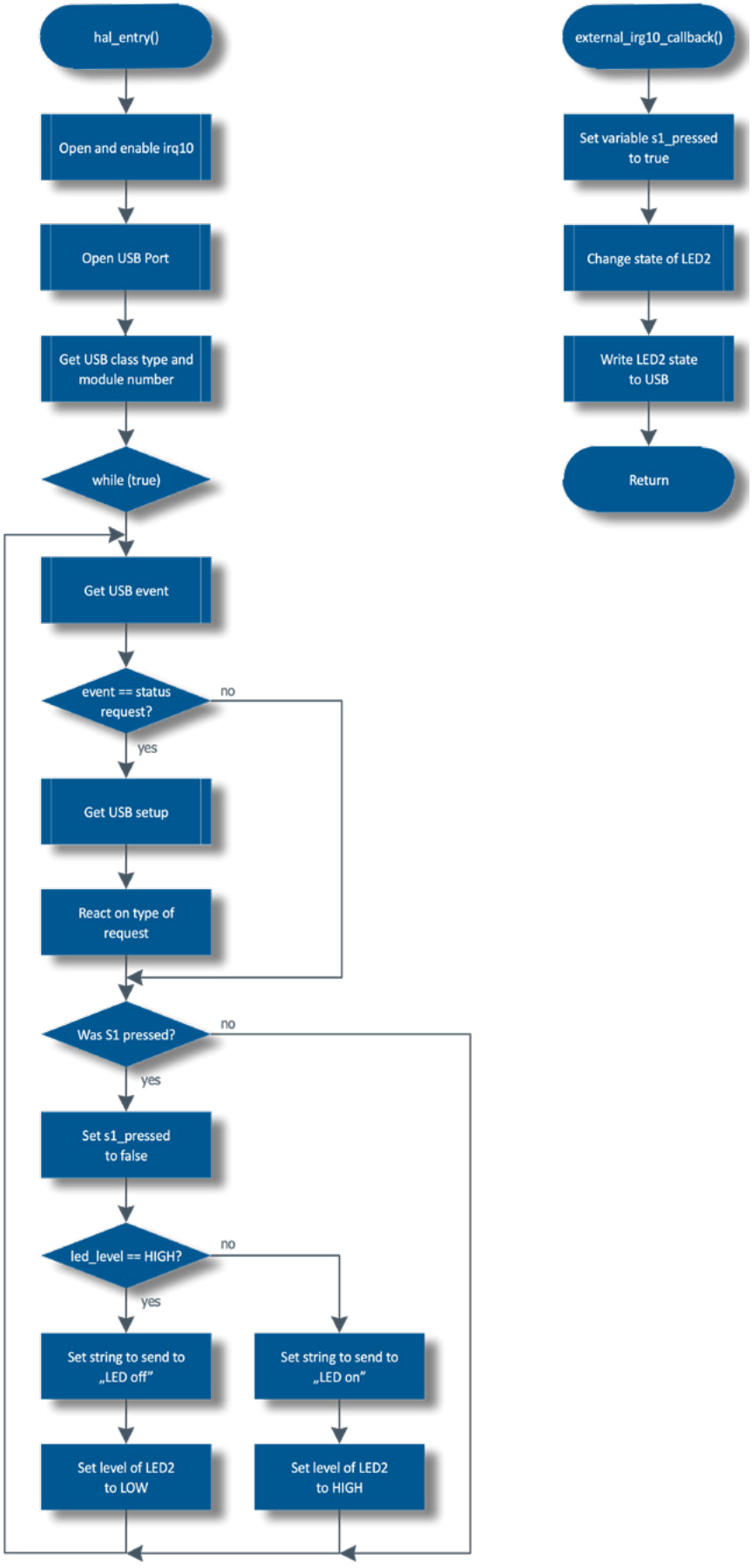

図10-1: この章の演習のフローチャート

10.1 FSP コンフィギュレータを使用したUSB ポートの設定

前回の演習後にe² studio を閉じた場合は、再度開いて新しいプロジェクトを作成します。今やあなたはRA のエキスパートで、実行するタスクのほとんどは前回の演習ですでに説明されているので、すべての手順 詳しくは説明しません。新しいプロジェクトにMyUSBProject という名前を付け、Device and Tools Selection 画面に移動したらEK-RA6M4 をボードとして選択します。この評価キットをラボ用に再度使用します。「Project Type Selection」ページで、[Flat](Non-TrustZone) プロジェクトを有効にしておき、「RTOS Selection」で[No RTOS]エントリが有効になっていることを確認します。最後に、[Finish]をクリックする前に、「Project Template Selection」ページで[Bare Metal – Minimal]を選択します。

プロジェクトコンフィギュレータがプロジェクトを作成し、FSP コンフィギュレータが表示したら、Stacks タブに直接移動します。まず、ユーザプッシュボタンS1 に接続されている外部割り込み用のモジュールを追加する必要があります。HAL/Common Stacks ペインでNew Stack をクリックし、Driver → Input → External IRQ (r_icu) を選択します。

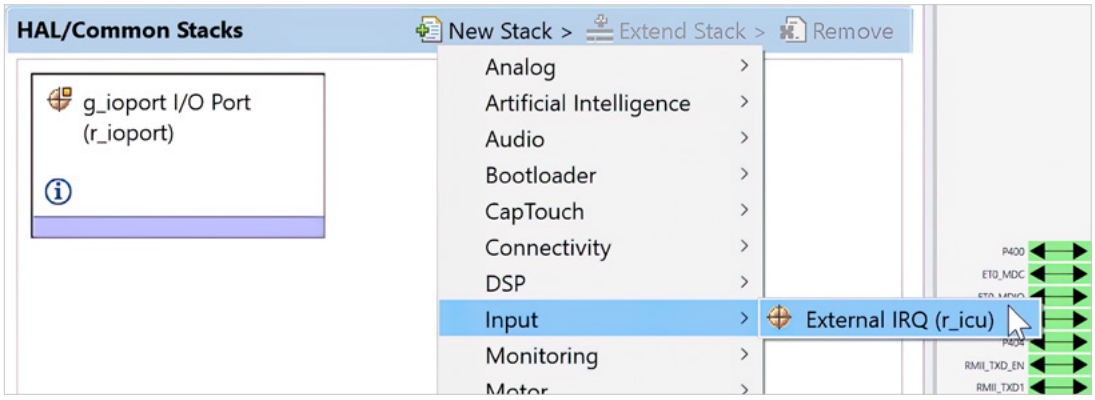

図10-2: S1 割り込み用ドライバの追加

Properties ビューで、割り込みの名前をg_external_irq10 に、そのChannel を10 に変更します。これは割り込みで使用されるチャネルであるためです。Digital Filtering を有効にし、Trigger をFalling に設定します。これは、スイッチのデバウンスに役立ちます。最後に、この割り込みのコールバック関数の名前を指定する必要があります。external_irq10_callback という名前を付け、Priority を14 に変更します。これは、USB 割り込みがプッシュボタンよりも優先されるようにするためです(図10-3 を参照)。

Property	Value
Common	
Parameter Checking	Default (BSP)
Module g_external_irq10 External IRQ (r_icu)	
Name	g_external_irq10
Channel	10
Trigger	Falling
Digital Filtering	Enabled
Digital Filtering Sample Clock (Only valid when Digit	PCLK / 64
Callback	external_irq10_callback
Pin Interrupt Priority	Priority 14
Pins	
IRQ10	P005

図10-3: IRQ10 に必要な設定

次に、USB ペリフェラル通信デバイスクラス(PCDC) のミドルウェアをシステムに追加します。新しいスタックを作成し、Connectivity → USB PCDC (r_usb_pcdc) と選択します(図10-4 を参照)。

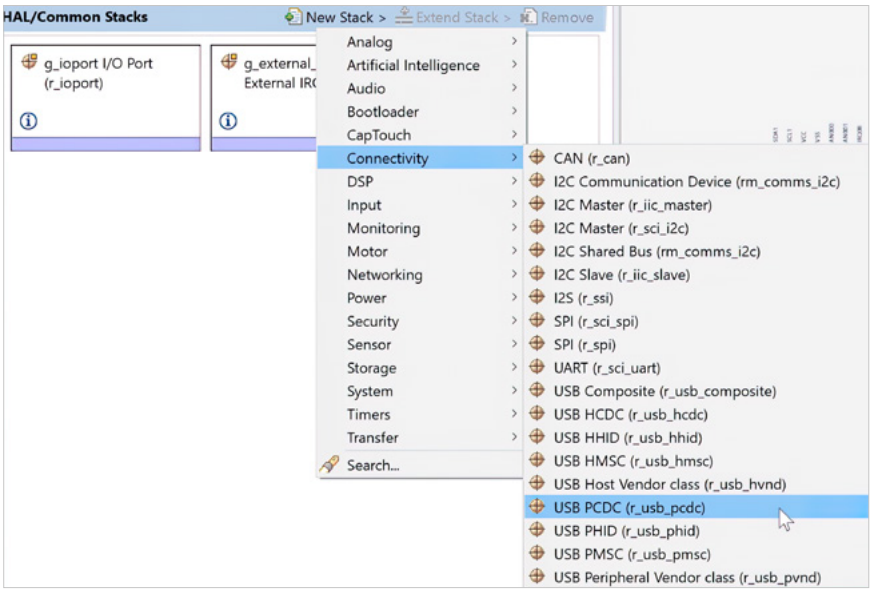

図10-4: USB 周辺通信デバイスクラスドライバのミドルウェアをシステムに追加する必要があります

これにより、実際にはプロジェクトに4 つのモジュールが追加されます。フルスピードUSB ポート用の実際のPCDC ドライバ(アプリケーションレベル用のUSB PCDC インターフェイスを実装) と、ベーシックUSB ドライバです。スタックには、ピンク色のバーが付いた2つのモジュールも表示されます。これにより、送受信用にDMAC (Direct Memory Access Controller)用のオプションドライバを追加できます。USB write API 関数を使用してステータスメッセージを直接送信するため、追加する必要はありません。モジュールの他のカラーバーの意味がどうなっているか不思議に思うかもしれませんが、これは非常に単純です。グレーは、他の1つのモジュールインスタンスのみによって参照される可能性のあるモジュールインスタンスをマークし、青は複数のスタックにわたっても、他の複数のモジュールインスタンスによって参照される可能性のある共通モジュールインスタンスをマークします。カラーバーの小さな三角形を使用すると、モジュールツリーを展開したり折りたたんだりできます。

USBポートをPCDCデバイスとして実装することにより、USBを仮想COMポートとして使用できます。 これにより、Windows®に登録した後、ターミナルアプリケーションを使用してポートとの通信が可能になるため、ホスト側でのレシーバのセットアップが簡単になります。 これで私たちは評価キットと対話することができるようになります。

すべてのスタックが追加されると、HAL /CommonStacksペインは図10-5のようになります。

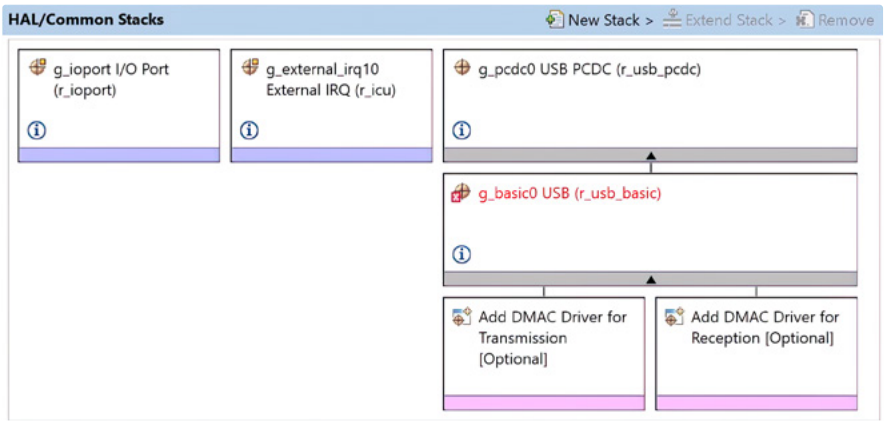

図10-5: USB ドライバを追加した後のスタックペイン

ベーシックUSBドドライバのプロパティを変更する必要はありません。 DMAサポートが無効に設定されていることを確認してください。g_basic0 USBドライバ（r_usb_basic）セクションのUSB記述子の名前に注意してください：g_usb_descriptor。この名前の構造は、システムにUSBの機能を説明するために後で作成されるため、この名前を覚えておく必要があります。 図10-6に、モジュールのプロパティを示します。

Property	Value
▾ Common	
Parameter Checking	Default (BSP)
PLL Frequency	Not Supported
CPU Bus Access Wait Cycles	Not Supported
Battery Charging	Not Supported
Power IC Shutdown Polarity	Active High
Dedicated Charging Port (DCP) M	Not Supported
Notifications for SET_INTERFACE/	Enabled
Double Buffering	Enabled
Continuous Transfer Mode	Not Supported
LDO Regulator	Not Supported
DMA Support	Disabled
DMA Source Address	DMA Disabled
DMA Destination Address	DMA Disabled
USB Compliance Test mode	Disabled
USB TPL table name	NULL
▾ Module g_basic0 USB (r_usb_basic)	
Name	g_basic0
USB Mode	Peri mode
USB Speed	Full Speed
USB Module Number	USB_IP0 Port
USB Device Class	Peripheral Communications Device (
USB Descriptor	g_usb_descriptor
USB Compliance Callback	NULL
USBFS Interrupt Priority	Priority 12
USBFS Resume Priority	Priority 12
USBFS D0FIFO Interrupt Priority	Priority 12
USBFS D1FIFO Interrupt Priority	Priority 12
USBHS Interrupt Priority	Not Supported
USBHS D0FIFO Interrupt Priority	Not Supported
USBHS D1FIFO Interrupt Priority	Not Supported
USB RTOS Callback	NULL
USB Callback Context	NULL

図10-6: ベーシックUSBドライバの設定。 USB記述子の名前に注意してください

モジュールg_basic0USB（r_usb_basic）が赤で表示されていることに気付くでしょう。 マウスをその上に置くと、「the USB requires a 48 MHz clock.」と表示されます。 これはコンフィギュレータの[Clocks]タブで修正できますが、これを行う前に、最初にUSBポートの正しい動作モードを設定します。

これを行うには、[Pins]タブに切り替えて、最初に[Peripherals]ドロップダウンリストを展開し、次に[Pin Selection]ペインの[Connectivity：USB]リストを展開します。 [Pin Configuration]ペインで、[Operation Mode]を[Custom]から[Device]に変更します。これが使いたいモードです。
この設定によって入力/出力ピンの割り当てが変わることに注意してください。

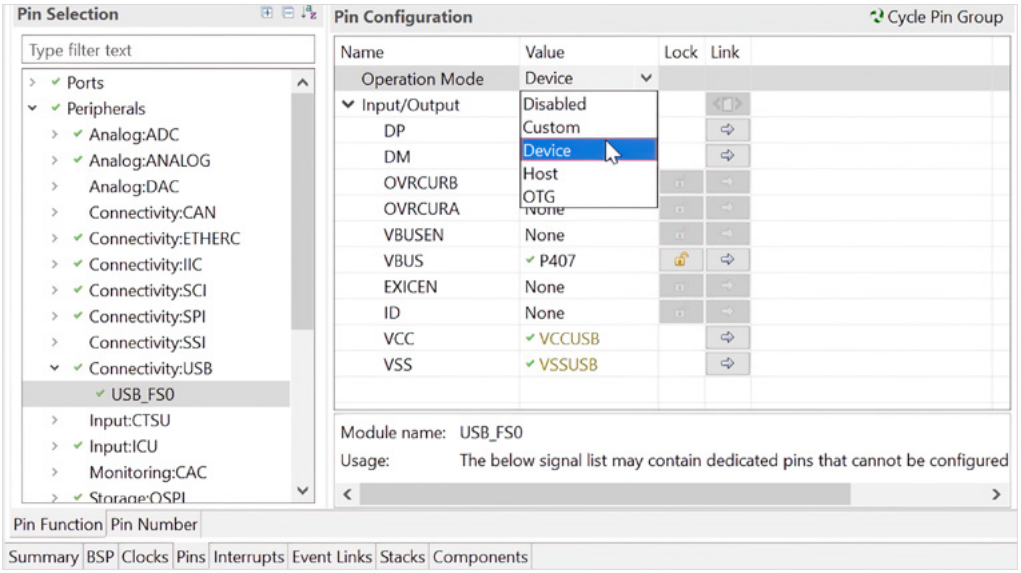

図10-7: USB ポートはデバイスモードを使用するため、それに応じて変更してください。

これで、ポートの設定はほぼ完了です。最後の手順は、USBフルスピード(FS)モジュールの動作クロックであるUSBクロック(UCLK)を有効にして、48MHzの必要な周波数に設定することです。このためには、クロック生成回路を設定できるClocks タブを有効にします。まず、ペインの下部にあるUCLK を[PLL2]をソースにして[Disable]から[Enable]に変更することで、USBクロックを有効にします。次に、特にUSB モジュール専用のPLL であるPLL2 を、高速オンチップオシレータ(HOCO)をソースとして有効にします。USB は48MHz の周波数を必要とするのに対して、UCLK 周波数は40MHz になっているため、赤色で強調表示されています。これを解決するために PLL2 の逓倍値を24 に変更すると、PLL2 の周波数は240MHz となりUCLK 5分周 をセットすることで、48MHz の正しいUCLK 周波数が設定されます。最後の変更として、標準PLL のソースをXTAL からHOCO にも変更します。変更するフィールドが不明な場合は、図 10-8 を参照してください。

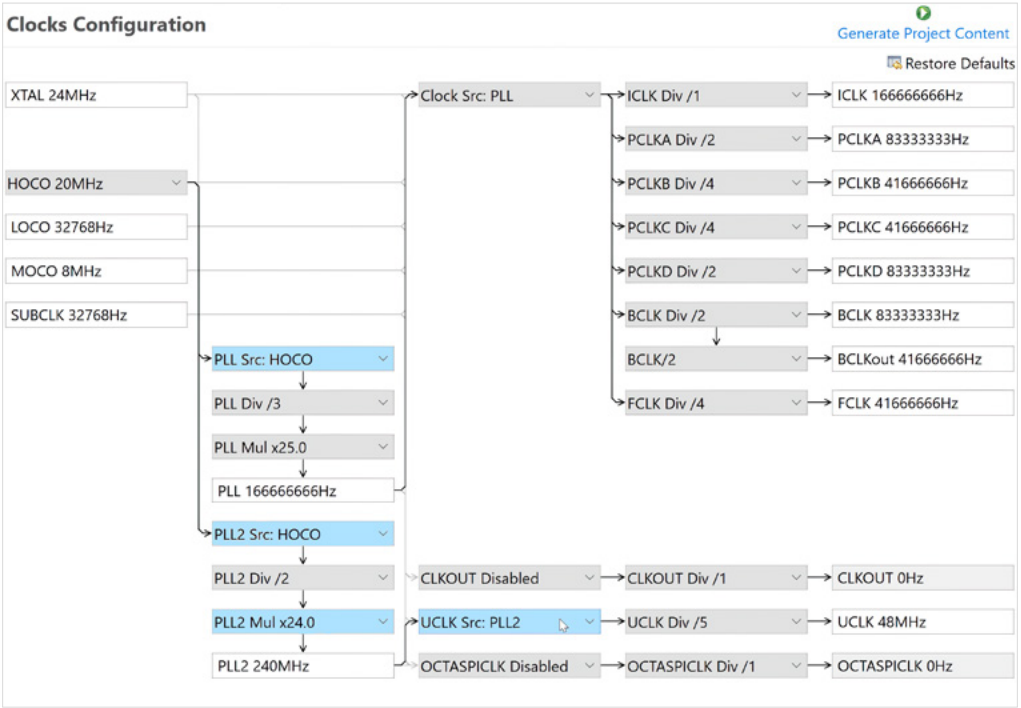

図10-8: USB FS には48MHz クロックが必要なので、クロック生成回路を適宜変更する必要があります。必要な変更が強調表示されます。

これらの変更により、g_basic0 USBドライバのエラーが解消されたため、[Stacks]タブは図10-9のようになります。

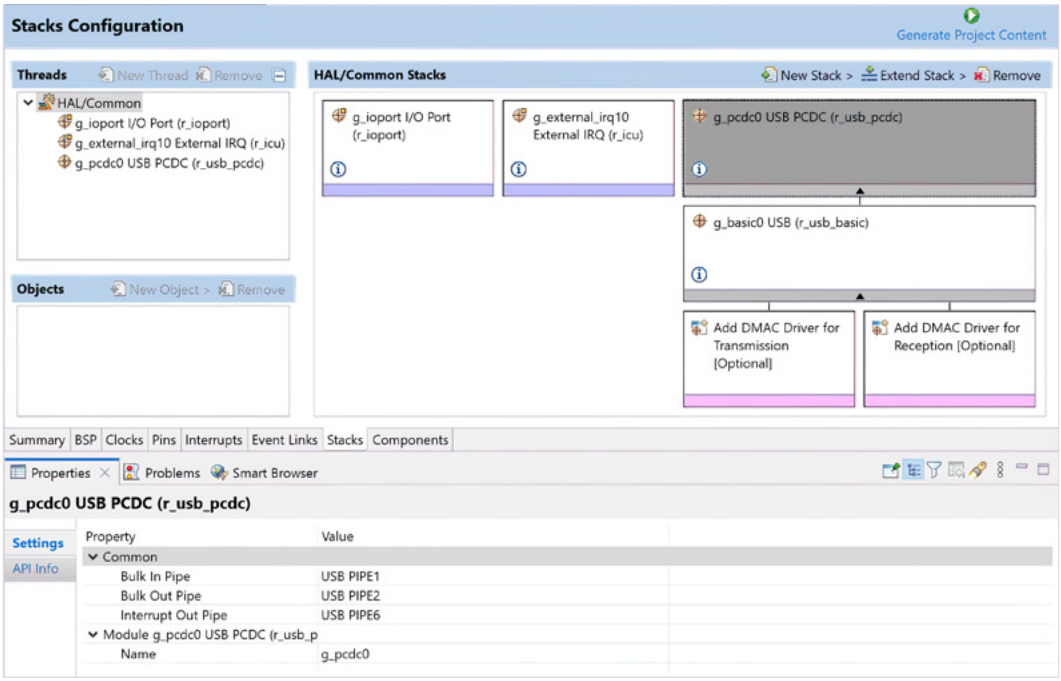

図10-9: すべての追加と変更が行われると、[Stacks]タブは次のようになります。

FSP コンフィギュレータで行う必要がある設定を終了します。コンフィギュレーションを保存し、画面の右上隅にあるGenerate Project Content ボタンをクリックしてファイルを抽出し、必要な設定を作成します。最後のステップとして、C/C++ パースペクティブに戻ります。

10.2 コードの作成

ここでは、USBポートの初期化およびUSBへの書込みを実行するために必要とされるコードを追加することになります。この演習には、多くのタイプ入力が含まれているため、ルネサスインターネットサイトの RA Book のウェブページからこの演習用のソリューションをダウンロードし、実際のコーディングを行わずに説明に沿って実行することができます。

すべてを最初から作成する場合は、まずProject Explorer でファイルhal_entry.c をダブルクリックして開くことから始めます。プログラムを正しく動作させるには、いくつかのグローバル変数が必要です。まず、hal_entry() 関数の先頭直前にUSB ドライバステータスの列挙型を宣言します。型はusb_status_tである必要があり、usb_eventという名前を付けます。次に、型usb_setup_t（r_usb_basic_api.hで宣言されている）の構造を追加し、usb_setupという名前を付けます。この変数は、後でUSB イベントの一部をデコードするときに使用し、USB イベントループ内で初期化されます。

次に、USBモジュール番号を保持する変数が必要です。uint8_t 型にして、g_usb_module_number という名前を付け、それに値"0x00" を割り当てます。最後に、usb_class_t 型のUSB クラス型の構造体を宣言し、g_usb_class_type という名前を付け、それに値"0x00" を割り当てます。使用する各種の型の詳細については、FSP(Renesas Flexible Software Package)ユーザーズマニュアルを参照してください。このマニュアルは、FSPs GitHub®サイトからダウンロードできます。

これらの追加が行われると、コードのこの部分は次のようになります:

```
/* Global variables for the USB */
usb_status_t usb_event;
usb_setup_t usb_setup;

uint8_t g_usb_module_number = 0x00;
usb_class_t g_usb_class_type = 0x00;
```

独自のコードは、いくつかのstaticグローバル変数にも依存します。USB 用の変数のすぐ下に追加してください:

```
/* Global variables for the program */
static char send_str[12] = { "LED on\n\r" };
static volatile uint8_t s1_pressed = false;
static uint8_t led_level = BSP_IO_LEVEL_HIGH;
```

send_str という名前の文字の配列は、USB 経由で送信するテキストを保持するために使用されます。LED2 をON に切り替えると、この変数が初めて使用されるため、"LED on\n\r" に初期化します。次の変数s1_pressed はuint8_t 型で、ユーザボタンS1 のコールバックルーチン内で値がtrue に変更されるため、volatile として宣言する必要があります。デフォルトではfalse で、IRQ10 割り込みのコールバックルーチンによってtrue に設定され、ボタンが押されたことを通知し、イベントが発生したことをメインプログラムに通知します。

この変数がvolatile として宣言されていない場合、C コンパイラのオプティマイザは変数が使用されるたびにその値を再度読み込まない可能性があり、hal_entry() 内のループは変更に気付かない可能性があります。

3 番目の変数はLED2 のレベルを保持し、起動時にBSP_IO_LEVEL_HIGH に初期化する必要があります。この値は、S1 がアクティブになるたびにトグルします。

これにより、すべてのグローバル変数が宣言され、hal_entry() 関数内にコードを記述することで続行できます。まず、ビットレート、ストップビット数、データビット数、パリティタイプなど、仮想UART 通信ポートの設定を保持するスタティック変数が必要です。この変数はusb_pcdc_linecoding_t 型で、g_line_coding という名前を付けることができます。"add your own code here" プレースホルダを宣言に置き換えてください。この変数は、後でUSB イベントハンドラループ内で初期化されます。

```
static usb_pcdc_linecoding_t g_line_coding;
```

次に、評価キットのS1 に接続されている外部IRQ10 を開いて有効にするコードを記述します。第9章と同様に、IRQ FSPドライバの各機能を使用します:

```
g_external_irq10.p_api->open (g_external_irq10.p_ctrl,
                              g_external_irq10.p_cfg);
g_external_irq10.p_api->enable (g_external_irq10.p_ctrl);
```

割込みを許可した状態で、USB をオープンし、クラスタイプとモジュール番号を取得する必要があります。r_usb_basic モジュール上のg_basic0 USB ドライバの関連する関数をそのために使用し、これらの関数に制御構造体、および設定構造体(Open() 関数の場合) への参照とそれぞれの変数を渡します。これらの行の記述に役立つように、コード補完機能とe² studio の開発者アシスタンスの両方があることを覚えておいてください。

```
R_USB_Open (&g_basic0_ctrl, &g_basic0_cfg);
R_USB_ClassTypeGet (&g_basic0_ctrl, &g_usb_class_type);
R_USB_ModuleNumberGet (&g_basic0_ctrl, &g_usb_module_number);
```

これで、割り込みとUSBポートの初期化が完了しました。プログラムのこの部分は無限に実行されるため、次のコードはすべてwhile (1) ループ内に配置する必要があります。まず、ポートのUSB関連イベントを取得して処理するためのコードを記述します。USB ドライバにはいくつかのイベントが関連付けられていますが、簡潔にするためにUSB_STATUS_REQUEST イベントのみを処理します。より完全なイベントハンドラに関心がある場合は、Flexible Software Package (FSP) User's Manual のUSB Peripheral Communications Device Class (r_usb_pcdc) のマニュアルを参照してください。このようなハンドラのコード例とフローチャートがあります。

最初のタスクは、R_USB_EventGet() 関数を呼び出してusb_event 変数を初期化し、USB_STATUS_REQUEST イベントが発生した場合にのみ実行するハンドラを記述することです。if - then - else 構文内で、まずUSB ポートのセットアップを取得し、次に通信設定が要求されたかどうかを判断します。Yes の場合、g_line_coding 変数を渡して仮想UART 設定を構成します。

そうでない場合は、ホストがUART 設定を受信するかどうかを照会します。Yesの場合は、ホストに送信します。最後に、イベントが発行された場合、ここでは処理はせず、単に確認します。

ハンドラのバージョンの完全なコードは次の通りです:

```
/* Obtain USB related events */
 R_USB_EventGet (&g_basic0_ctrl, &usb_event);

/* USB event received by R_USB_EventGet */
if (usb_event == USB_STATUS_REQUEST)
{
    R_USB_SetupGet (&g_basic0_ctrl, &usb_setup);

    if (USB_PCDC_SET_LINE_CODING == (usb_setup.request_type
                                     & USB_BREQUEST))
    {
        /* Configure virtual UART settings */
        R_USB_PeriControlDataGet (&g_basic0_ctrl,
                          (uint8_t*) &g_line_coding, LINE_CODING_LENGTH);
    }
    else if (USB_PCDC_GET_LINE_CODING == (usb_setup.request_type
                                          & USB_BREQUEST))
    {
        /* Send virtual UART settings back to host */
        R_USB_PeriControlDataSet (&g_basic0_ctrl,
                                  (uint8_t*) &g_line_coding,
                                  LINE_CODING_LENGTH);

    }
    else if (USB_PCDC_SET_CONTROL_LINE_STATE == (usb_setup.request_type
                                                 & USB_BREQUEST))
    {
        /* Acknowledge all other status requests */
        R_USB_PeriControlStatusSet (&g_basic0_ctrl, USB_SETUP_STATUS_ACK);
    }
    else
    {
    }
}
```

ハンドラ内でLINE_CODING_LENGTH という名前が2 回使用されていることがわかります。まだその値を定義していないので、ファイルの先頭に戻って、符号なしの値0x07 でLINE_CODING_LENGTH を定義してください。

```
#define LINE_CODING_LENGTH          (0x07U)
```

while (1) ループ内に戻り、s1_pressed の真の値で示されるように、S1 がアクティブ化されたらLED2 のレベルを変更するコードを追加します。これは第9 章で書いたものと似ていますが、今回はUSB を介して送信される文字列をsend_str 変数にコピーし、s1_pressed 変数をfalse に設定する必要もあります:

```
if (s1_pressed == true)
{
    s1_pressed = false;

    if (led_level == BSP_IO_LEVEL_HIGH)
    {
        strcpy (send_str, “LED off\n\r”);
        led_level = BSP_IO_LEVEL_LOW;
    }

else
    {
        strcpy (send_str, “LED on\n\r”);
        led_level = BSP_IO_LEVEL_HIGH;
    }
}
```

最後に追加するコードは、外部IRQ10 のコールバック関数のコードです。hal_entry () 関数の閉じ括弧の直後に配置します。コールバックの詳細については、9 章を参照してください。まず、g_bsp_leds 構造体をインポートし、ローカルのLeds 変数を初期化する必要があります。次に、s1_pressed をtrue に設定してイベントが発生したことを通知し、次に新しい値をピンレジスタに書き込みます。最後に、r_usb_basic モジュールのR_USB_Write() API を使用して、USB ポート経由で文字列を送信します。

```
void external_irq10_callback(external_irq_callback_args_t *p_
args)
{
    FSP_PARAMETER_NOT_USED(p_args);
    extern bsp_leds_t g_bsp_leds;
    bsp_leds_t Leds = g_bsp_leds;

    s1_pressed = true;
    g_ioport.p_api->pinWrite (&g_ioport_ctrl,
                              Leds.p_leds[BSP_LED_LED2],
                               led_level);

    R_USB_Write (&g_basic0_ctrl,
                 (uint8_t*) send_str,
                 ((uint32_t) strlen (send_str)),
                 (uint8_t) g_usb_class_type);
}
```

g_usb_descriptor という名前のUSB ディスクリプタを覚えておいてください。ここでは、注意が必要です。USB には、デバイス、その設定、およびベンダー情報の正確な説明が必要です。このファイルは非常に複雑で、コードが484行となってかなり長くなっています。このファイルの説明は、FSP ユーザーズマニュアルのr_usb_basic セクションと、このディスクリプタの構築方法の詳細な説明が記載されているユニバーサルシリアルバス改訂2.0 規格(**http://www.usb.org/developers/docs/**) にあります。

なお、ショートカットが2つあります。1 つは、本書の演習用のソースファイルを本書のWeb サイトからダウンロードすることです **www.renesas.com/ra-book**。もう1 つは、FSP コンフィギュレータがプロジェクトのra ディレクトリに配置したテンプレートを使用することです。これはr_usb_pcdc_descriptor.c.template という名前で、プロジェクトエクスプローラのra → fsp → src → r_usb_pcdc フォルダに移動してアクセスできます(図10-10 を参照)。このファイルをhal_entry.c が存在するsrc フォルダにコピーし、r_usb_descriptor.c に名前を変更します。独自の製品のID に合わせてベンダーID と製品ID を変更します。まだそれがない場合は、一時的に"0x045BU" と"0x5310U" の値を使用してください。これにより、実際に行う設定と書き込むコードは終わりです。

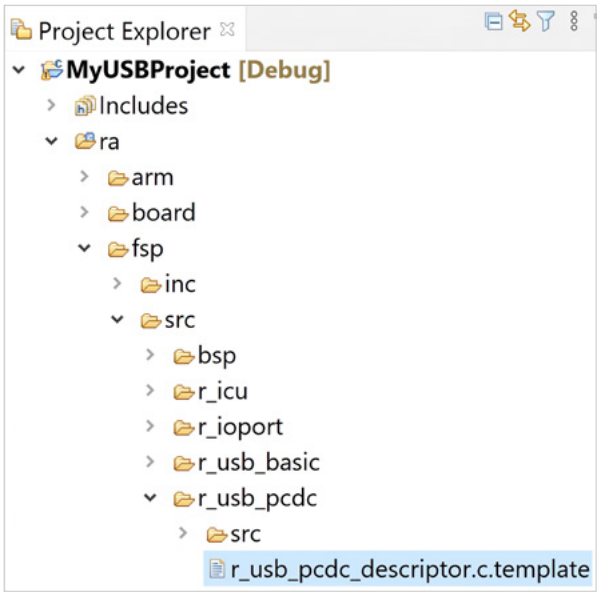

図10-10: FSP コンフィギュレータがUSB ディスクリプタのテンプレートを自動的に作成した場合

すべてのテンプレートは、プロジェクトをビルドするためのものです。初めて行う場合は、プロジェクトに含まれるすべてのFSPモジュールのコードもコンパイルする必要があるため、時間がかかります。エラーがゼロで警告がゼロのプロジェクトが構築されたらすぐに、EK-RA6M4 評価キットを接続し、デバッグセッションを開始します。Debug Perspective を開いた状態で、Resume を2 回クリックしてプログラムを開始します。クイックテストとして、S1 を1 回押してLED2 がトグルするかどうかを確認します。

10.3 ホスト側の受信設定

プログラムの実行中に、2 本目のUSB タイプA - micro-B ケーブルを、評価キットのSystem Control & Ecosystem Access エリアの左下にあるJ11 というラベルの付いたUSB ポートに接続します。もう一方の端をWindows®ワークステーションに挿入し、Windows®がボードを認識してエニュメレーションし、ドライバをインストールするまで数秒待ちます。

ターミナルエミュレータプログラムを起動します。この演習の展開中に、**https://ttssh2.osdn.jp/**からダウンロードできるTera Termを使用しましたが、非常に便利であることがわかりました。Tera Termでは、CDCシリアルポートが表示されています。図10-11 ではCOM3 ですが、他のPC では異なる可能性があります。不明な場合は、Windows®のデバイスマネージャを使用して、ボードが接続されているポートを確認してください。

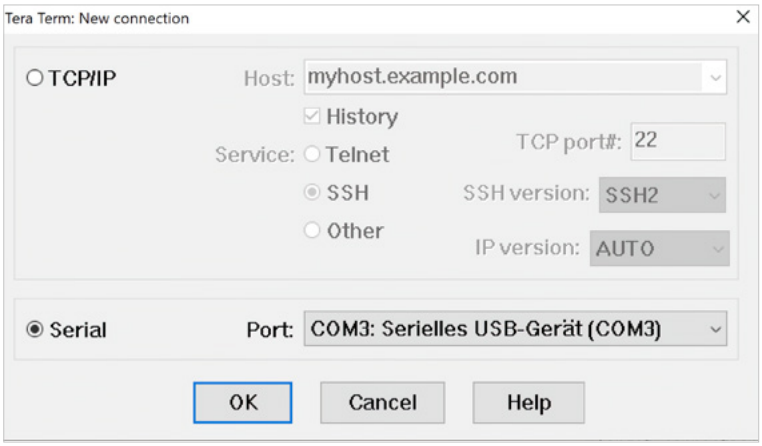

図10-11: Windows®がボードを正しく認識した場合、シリアル接続としてTera Term に一覧表示されます。

ボードがまったく表示されていない場合や、デバイスマネージャがエラーを示している場合は、ドライバに問題がある可能性があります。解決するには、ルネサスナレッジベースの最新のサポートエントリーを参照してください。
https://en-support.renesas.com/knowledgeBase/18959077

接続後でTera Term の実行中にS1 を数回押すと、緑色のLED2 トグルとその状態が図10-12 に示すようにターミナルに出力されます。

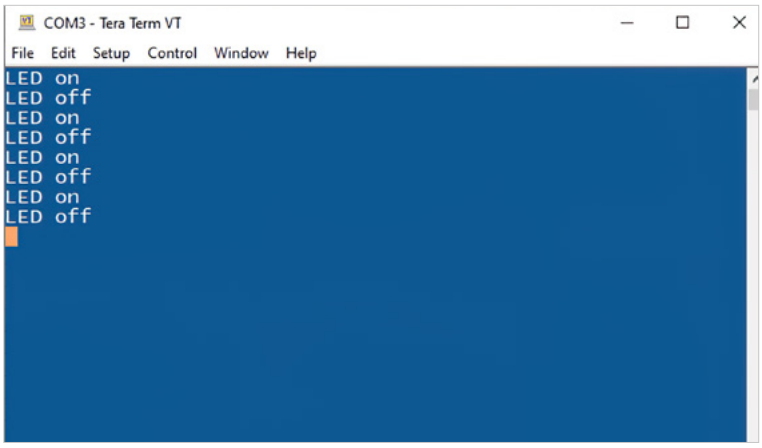

図10-12: 転送実行中は、S1 を押すたびにLED2 の状態が表示されます。

```
#include „hal_data.h“

FSP_CPP_HEADER
void R_BSP_WarmStart(bsp_warm_start_event_t event);
FSP_CPP_FOOTER

#define LINE_CODING_LENGTH          (0x07U)

/* Global variables for the USB*/
usb_status_t usb_event;
usb_setup_t usb_setup;
uint8_t g_usb_module_number = 0x00;
usb_class_t g_usb_class_type = 0x00;

/* Global variables for the program */
static char send_str[12] = { „LED on\n\r“ };
static volatile uint8_t s1_pressed = false;
static uint8_t led_level = BSP_IO_LEVEL_HIGH;

/*******************************************************************************************************/
/* main() is generated by the RA Configuration editor and is used to generate threads if  */
/* an RTOS is used.  This function is called by main() when no RTOS is used.             */
/*******************************************************************************************************/
void hal_entry(void) {
    static usb_pcdc_linecoding_t g_line_coding;

    /* open and enable irq10 */
    g_external_irq10.p_api->open (g_external_irq10.p_ctrl, g_external_irq10.p_cfg);
    g_external_irq10.p_api->enable (g_external_irq10.p_ctrl);

    /* Open USB instance */
    R_USB_Open (&g_basic0_ctrl, &g_basic0_cfg);

    /* Get USB class type */
    R_USB_ClassTypeGet (&g_basic0_ctrl, &g_usb_class_type);

    /* Get module number */
    R_USB_ModuleNumberGet (&g_basic0_ctrl, &g_usb_module_number);

    while (true)
    {
        /* Obtain USB related events */
        R_USB_EventGet (&g_basic0_ctrl, &usb_event);

        /* USB event received by R_USB_EventGet */
        if (usb_event == USB_STATUS_REQUEST)
        {
            R_USB_SetupGet (&g_basic0_ctrl, &usb_setup);

            if (USB_PCDC_SET_LINE_CODING == (usb_setup.request_type & USB_BREQUEST))
            {
                /* Configure virtual UART settings */
                R_USB_PeriControlDataGet (&g_basic0_ctrl,
                                          (uint8_t*) &g_line_coding, LINE_CODING_LENGTH);
            }
            else if (USB_PCDC_GET_LINE_CODING == (usb_setup.request_type & USB_BREQUEST))
            {
                /* Send virtual UART settings back to host */
```

```
                R_USB_PeriControlDataSet (&g_basic0_ctrl,
                                          (uint8_t*) &g_line_coding, LINE_CODING_LENGTH);

            }
            else if (USB_PCDC_SET_CONTROL_LINE_STATE == (usb_setup.request_type
                                                         & USB_BREQUEST))
            {
                 /* Acknowldedge all other status requests */
                 R_USB_PeriControlStatusSet (&g_basic0_ctrl, USB_SETUP_STATUS_ACK);
            }
            else {
            }
        }

        if (s1_pressed == true)
        {
            s1_pressed = false;

            if (led_level == BSP_IO_LEVEL_HIGH)
            {
                 strcpy (send_str, „LED off\n\r“);
                 led_level = BSP_IO_LEVEL_LOW;
            }

            else
            {
                strcpy (send_str, „LED on\n\r“);
                led_level = BSP_IO_LEVEL_HIGH;
            }
        }
    }

#if BSP_TZ_SECURE_BUILD
    /* Enter non-secure code */
    R_BSP_NonSecureEnter();
#endif
}

/*****************************************************************************************/
/* callback function for the S1 push button; writes the new level to LED2 and sends it */
/* through the USB to the PC                                                             */
/*****************************************************************************************/
void external_irq10_callback(external_irq_callback_args_t *p_args)
{
    /* Not currently using p_args */
    FSP_PARAMETER_NOT_USED(p_args);
    extern bsp_leds_t g_bsp_leds;
    bsp_leds_t Leds = g_bsp_leds;

    s1_pressed = true;
    g_ioport.p_api->pinWrite (&g_ioport_ctrl, Leds.p_leds[BSP_LED_LED2], led_level);

    R_USB_Write (&g_basic0_ctrl,
                 (uint8_t*) send_str,
                 ((uint32_t) strlen (send_str)),
                 (uint8_t) g_usb_class_type);
}
```

おめでとうございます。

この練習を無事に終了しました!

本章で学んで会得するポイント:

- USB ポートの設定は、FSP Configurator とUSB ミドルウェアによって簡素化されています
- USB転送を行うためには、USBディスクリプタファイルが不可欠です

11 セキュリティとTRUSTZONE®

この章で学ぶこと:

- TrustZone とは何か、またデバイスの保護にどのように役立つか。
- ルネサスのTrustZone実装が提供するメリット。
- アプリケーションのセキュア部分と非セキュア部分の分離がどのように機能するか。
- デバイスライフサイクルマネジメントを実装する方法。

セキュリティは重要です。また、それは後で追加することはできないため、初期段階から考える必要があります。後で追加する場合は少なくとも高額な再設計の費用が必要でしょうし、場合によってはアプリケーション全体を見直すために一旦破棄する場合もあるでしょう。これは建物の土台と考えてください。それ自体は魅力的ではないし残りの機能を一緒に作成するよりも、設計と構築に時間がかかる場合さえあります。 設計のみを見直しているとき、土台がいかに重要か、あなたは気付くことができないかもしれません。 それでも、この接続された世界のすべてのアプリケーションにはセキュリティが不可欠なのです。

セキュリティが必要なのはいつでしょうか。製品がWiFi、Bluetooth、USB、Ethernetなどの有線またはワイヤレスインターフェイスを介して通信している場合に必要になってきます。鍵、証明書、パスワード、個人データ、測定データ、アルゴリズムなどの重要な情報を保存する場合は、この情報が必要になります。また、ファームウェアの更新、機能のアップグレード、有料サービスなど、製品がアップグレード可能な場合も必要になります。

そのため、セキュリティは非常に多くのアプリケーション領域に影響を与えます。スマートグリッド、スマートシティ、スマートビルディング、スマートホーム、スマートライティング、スマートウォッチ、スマート家電、スマートクロスなど、「スマートグリッド」であるすべてに関係しています。センサ、ロボット、ビルオートメーション、農業なども安全に保つ必要があるため、IoTや産業用アプリケーションにも影響します。また、セキュリティを重視しなければ、生命がすぐに危険にさらされる医療アプリケーションも存在することも覚えておいてください。

ルネサスのRAファミリは、Arm® Cortex®-M4、-M23、または-M33 コアを使用するデバイス、Cortex-M4 または-M23 コアを搭載したデバイスのセキュアメモリ保護ユニット、またはCortex-M33 コアを搭載したマイクロコントローラにルネサス独自のTrustZone®を実装することによって、セキュアなアプリケーションの作成をサポートしています。そして、後者はこの章のトピックとなります。

最後に、デバイスのグループに応じて、RA ファミリのマイクロコントローラは、NIST CAVP とPSA レベル1 およびレベル2、さらには追加のNIST およびPSA に対して認定されています。このファミリには、信頼できるFirmware-M（TF-M）API準拠のデバイスも含まれています。 これにより、ユーザは、接続された環境向けの安全な製品を開発するための高いレベルの保証と確証を得ることができます。

11.1 TrustZone®とは何か、またどのように役立つか?

TrustZone®は、Arm v8-M アーキテクチャの一部としてArm®によって開発されたコアテクノロジであり、ハードウェアの隔離によるセキュリティに対するシステム全体のアプローチを提供します。これは、セキュアなMCU 領域と非セキュアなMCU 領域の論理パーティションへのソフトウェアの分離を要求することによって保護された環境を作成できます。通常は、Arm® Cortex®-M33 コアデバイスに標準実装されArm Cortex-M23 コアデバイスにはオプションで実装されます。

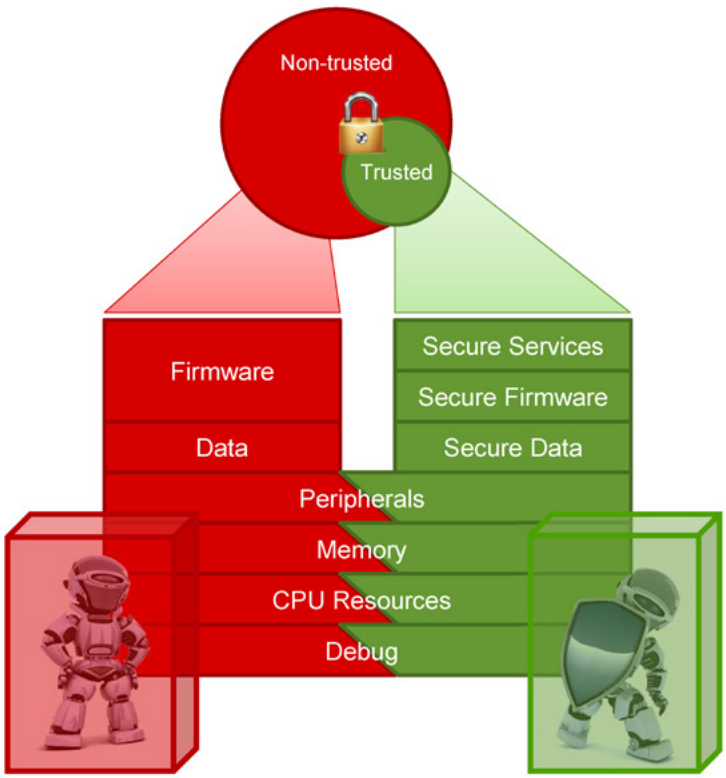

図11-1: TrustZone®を使用すると、マイクロコントローラのリソースを非セキュア領域とセキュア領域に分割できます。

メモリ領域は、3 つの異なるゾーンに分割されます。セキュア領域は、キーストレージやデータ復号化などの信頼されたIP または保護されたIP に使用され、非セキュア領域は通常のアプリケーションに使用され、非セキュアなコール可能領域は、他の2 つのパーティション間のゲートウェイとして機能します。後者は、非セキュア領域に存在するコードがセキュア領域内のサービスを呼び出すことを可能にします(図11-2 を参照)。これは、セキュア領域と非セキュア領域の分離を可能にするベニアまたはガード関数を介して行われます。

TrustZoneを使用すると、非セキュア領域、いわゆる非セキュア処理環境 (NSPE) または略して非セキュア環境からのコードは、セキュア領域 (セキュア処理環境 (SPE) またはセキュア環境) のアイテムに直接アクセスできません。これは、たとえば、セキュア領域内のキーや証明書を非セキュア領域で変更したり、使用したりすることができないことを意味します。セキュアな環境に存在するコードのみがこれを行うことができます。これにより、IP保護とサンドボックスモデル化が可能になり、主要コンポーネントの攻撃対象領域が削減されます。また、組み込みデバイスのセキュリティ評価が簡素化されます。したがって、保護したいマイクロコントローラに知的財産(IP)があればセキュア領域に置いてください。

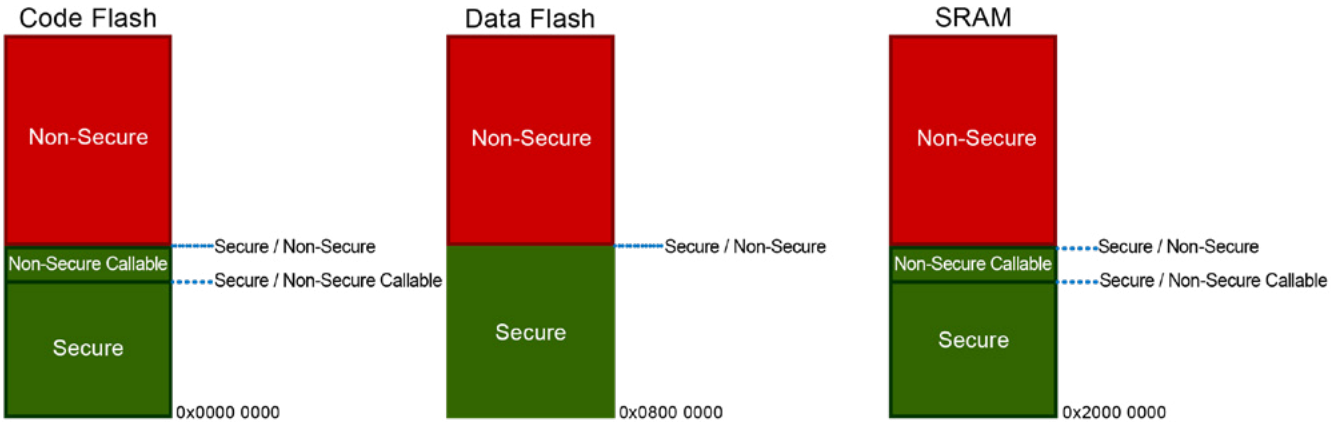

図11-2: TrustZone を使用すると、セキュア領域と非セキュア領域にセグメンテーションできます。

非セキュア領域の関数とデータは、セキュアおよび非セキュアステートからアクセスできます。一方、非セキュア領域のルーチンは非セキュアコール可能領域(NSC)に存在するガード関数の呼び出しを介してセキュア領域のサービスにのみアクセスできます。これによりセキュアな環境への定義済みのエントリポイントが提供されます。

Armの当初のTrustZoneの定義は、Flashメモリ、RAM、そしてSysTickタイマやMPUのような組み込み部品に適用されアプリケーションコードとデータをセキュアなアイテムから切り離すことを求めていますが、DMAコントローラの保護、周辺機器の離反の保護は含まれていません。さらに、ペリフェラルとメモリアクセス制御のための実装定義属性ユニット(IDAU)を提供します。

ルネサスはそのコンセプトを拡張しTrustZoneフィルタを他のバスやピン、周辺機器に適用しました。バスフィルタは非セキュアコードがDMA、DTC、および同様のメカニズムを介してセキュアなコードとデータを抽出するのを防ぎます。つまり、ペリフェラルがメモリから読み出せないようにします。一方、ピンとペリフェラルに適用されるTrustZoneフィルタは、ピンをロックダウンしどこからでもアクセスできるようにするのではなく、セキュアな環境でのみ使用できるようにします。また、セキュアな環境と非セキュアな環境の間で、たとえば送受信のためにチャネルを分割することも可能です。

これにより外部インタフェースが保護されるだけでなく、非セキュアコードがピン経由で盗聴されるのを防ぐことができます。たとえば、非セキュアな環境に存在する悪意のあるコードはセキュアな領域に割り当てられたピンのステータスをモニタし、UARTを介した転送などのコンテンツを再構築することはできません。非セキュアコードは、出力を乗っ取ることもできません。

起動時にセキュリティギャップが直接発生しないようにするためにTrustZone を備えたRA マイクロコントローラは、常にセキュアメモリからセキュア領域にブートします。これにより、セキュアでないコードが最初にセキュアな領域内のデータ、アルゴリズム、またはペリフェラルを侵害することを禁止します。また、メモリセキュリティ属性はSCI (オンボード書き込み)ブートモードを介して不揮発性メモリに設定されます。属性は非セキュアアプリケーション自体が実行される前にIDAU レジスタにロードされ、アプリケーションはその内容を変更できず、読み出しのみが行われます。

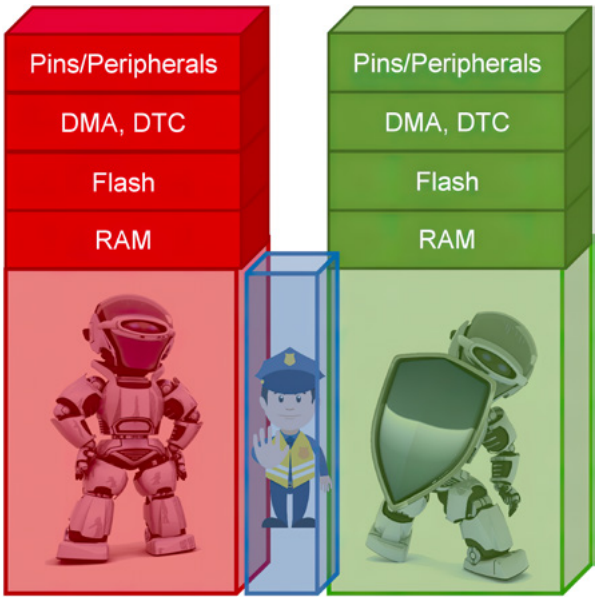

図11-3: ルネサスのTrustZone 実装では、メモリだけでなく、他のバス、ピン、ペリフェラルにもフィルタが適用されます。

注意すべき点が1つあります。アプリケーションのセキュリティはTrustZoneを単独で使用しても達成されません。 単に使用するだけではデバイスが安全になるわけではありません。 TrustZoneはセキュリティツールボックスのもう1つのツールであり正しい方法で使用する必要があります。 安全なアプリケーションに脆弱性（ずさんなコードの記述など）がある場合、TrustZone自体は攻撃やエクスプロイトを回避することができません。

ただし、TrustZone はRoot of Trust(RoT) を提供し、実現するのに役立ちます。RoT は、常に信頼できるソースであり、すべてのセキュアな操作、データ、およびアルゴリズムが依存するベースです。これはコードやデータの完全性と分離、システムの検証、機密性、または認証などのサービスを、現在または他の信頼できるシステムやデバイスで利用できるようにするだけでなく、キー、証明書、またはパスワードを追加で組み込みます。しかし、これは一方でRoTが侵された場合、全てが台無しになってしまったことを意味します。

TrustZone は機密データをセキュアな領域に保存できるようにすることで、データとファームウェアの保護を提供します。また、重要なソフトウェアをセキュアなファームウェアとして事前にロードし、セキュアな環境からのみアクセスを許可するように周辺機器を構成して、動作の保護を提供することもできます。

TrustZone の詳細と詳細説明については、Arm (**https://developer.arm.com/ip-products/security-ip/trustzone**) のドキュメントを参照してください。ただし、Arm では、Trusted とNon-Trusted について説明することが多いことに注意してください。定義としてはセキュアおよび非セキュアとして読み替えてください。

11.2 セキュアと非セキュア領域の仕切り

セキュアな環境とノンセキュアな環境の間でプログラムを分割する必要があることがわかったのですが、それに応じてソフトウェアを分割するにはどうすればよいでしょうか。TrustZone®ベースのシステムは常にセキュアと非セキュアの2 つの異なるプロジェクトで構成されます。どちらもSRAM とコードおよびデータフラッシュメモリを使用できますが、セキュアコードのみがセキュア領域と非セキュア領域の両方に直接アクセスできます。

これらのプロジェクトは、e² studio のプロジェクトコンフィギュレータを使用して設定します。新しいプロジェクトを作成すると、プロジェクトのタイプを選択するよう求められます:

- Flat (Non-TrustZone) Project
- TrustZone Secure Project
- TrustZone Non-secure Project

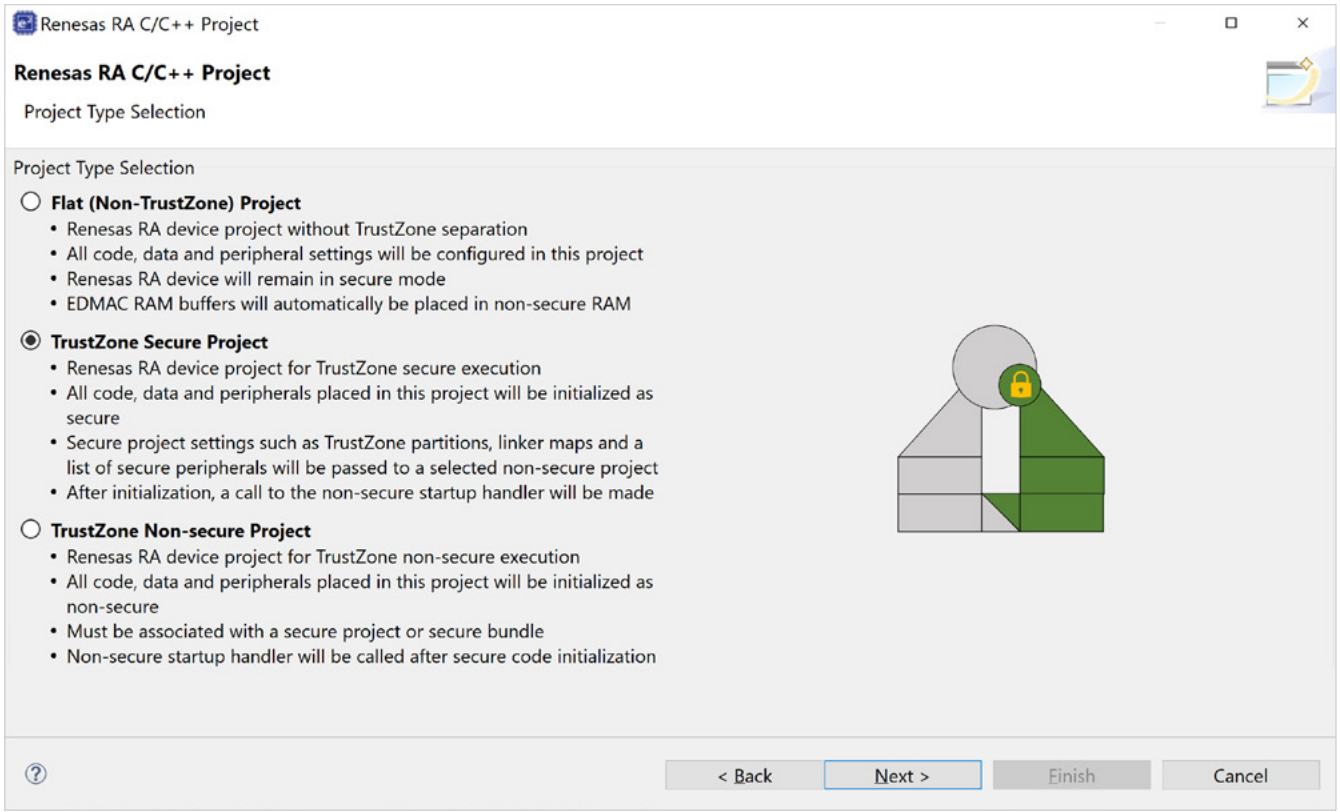

図11-4: プロジェクトコンフィギュレータのType Selector ページでは、Flat、Secure、Non-secureのプロジェクトを選択できます。

Flat (Non-TrustZone)プロジェクトを選択した場合、マイクロコントローラはブート後もセキュアモードのままになることに注意してください。また、TrustZone プロジェクトを設定するときは、セキュアパーティションと非セキュアパーティション間の接続が正しく管理されるように注意する必要があります。これはセキュアでないプロジェクトをセキュアなプロジェクトまたはバンドルに関連付けることによって行います。

セキュアプロジェクトを作成したらセキュアスタックとドライバを非セキュアな領域で使用できるようにすることができます。このためには、一番上のモジュールを右クリックし、ポップアップメニューからNon-secure Callable を選択します。そのエントリを選択した後の左側の小さな矢印に注意してください。これは、このモジュールが現在非セキュア呼び出し可能であることを示しています(図11-5 を参照)。

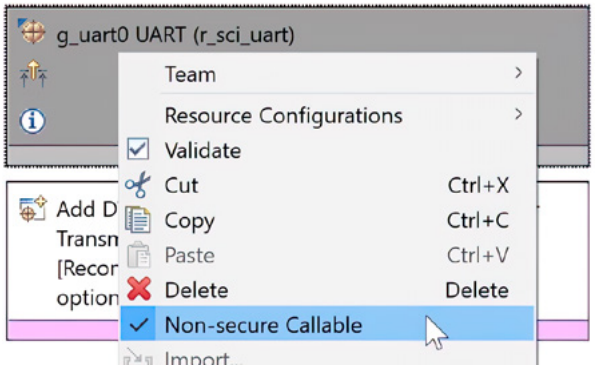

図11-5: セキュア プロジェクトのスタックの最上位モジュールは、Non-secure Callable にします

最後に、e² studio内からメモリーをパーティション分割することもできます。Run → Renesas Debug Tools → Renesas Device Partition Managerとするとユーティリティが実行されます。デバイスパーティションマネージャを使用すると、開発中にライフサイクル状態管理を実行し、さらにIDAU 領域のセットアップとクエリ、消去されたフラッシュメモリブロックのロック解除を行うことができます。

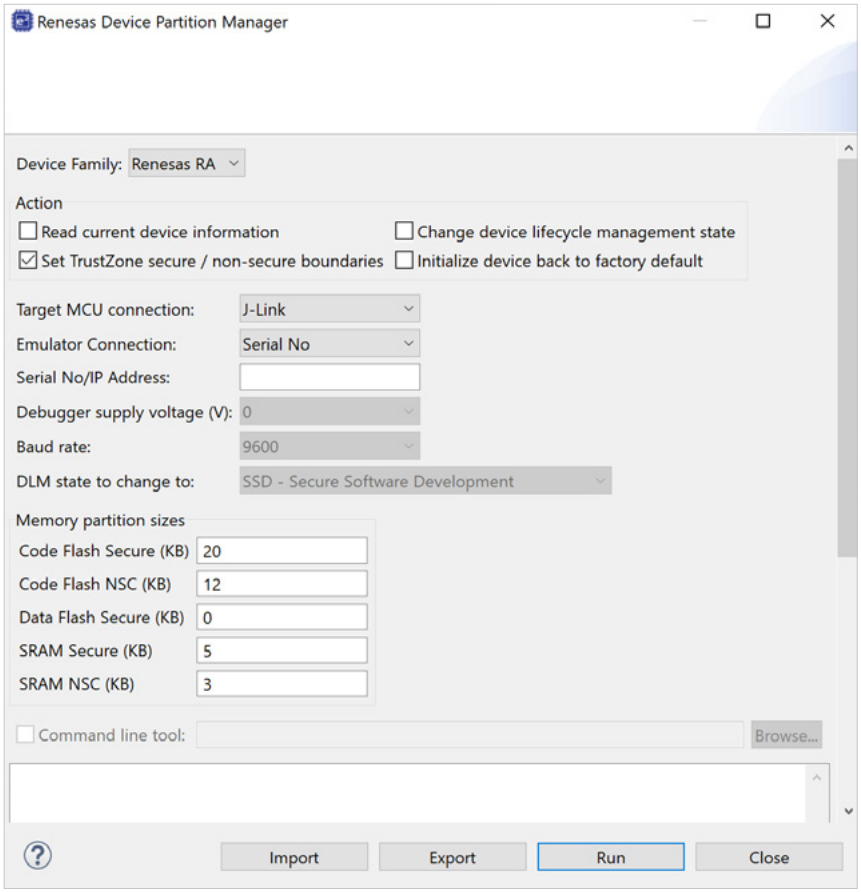

図11-6: Device Partition Managerを使用すると、さまざまなメモリパーティションのサイズを定義できます

11.2.1 境界を越えた関数呼び出し

次にたとえば、非セキュアな領域に存在するアプリケーションの一部がセキュアな領域に存在するFlash ペリフェラルを呼び出して、非セキュアなデータFlash をプログラムする場合はどうすればよいでしょうか。このために、ARM®v8M Cortex®-M33 コア(SG またはSecure Gateway) の命令セットに新しい命令が追加されました。この命令は、メモリのセキュア部分と非セキュア部分の間の非セキュア呼び出し可能(NSC)領域に配置する必要があります。これによりセキュア領域内の別の場所にSG 命令が見つかった場合でも、エントリポイントとして使用できなくなります。SG 命令の後、セキュア側のコードを呼び出すことができます(図11-7 を参照)。

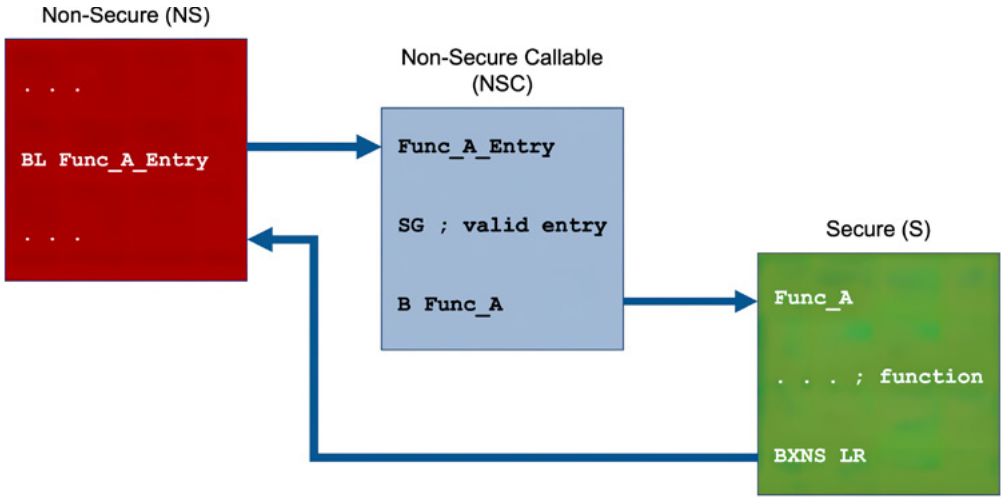

図11-7: 非セキュアコンテキストからのセキュア関数の呼び出し

セキュア側から非セキュア側への復帰は、BL Func_A_Entry ブランチ中に配置されたリンクレジスタ(LR)に含まれるアドレスに分岐するBXNS LR (BXNS = branch with exchange to non-secure state)命令を介して実行されます。関数の復帰時に、関数の復帰状態がLR 内の復帰アドレスのLSB に格納されます。このビットは非セキュアコードから呼び出されたセキュアAPI がセキュアアドレスを指す偽のリターンアドレスに戻らないように、呼び出し元関数への到着時のリターン状態と照合されます。

非セキュア領域に存在するコードからセキュア領域のコードへのコールが、最初の命令がNSC 領域のSG オペコードでない場合に実行されると、CM33 コアを搭載したマイクロコントローラでセキュアフォルトが発生します。この障害はセキュアな状態で処理されます。

セキュアなコードから非セキュアなコードを呼び出すことも可能ですが、データが漏洩する可能性がありセキュリティ上の問題であるため推奨されません。セキュアコードはパラメータを介して一部のレジスタ値を非セキュア領域に転送でき、コンパイラは残りのレジスタから他のセキュアデータをクリアします。このメカニズムは、セキュアソフトウェアのリターンアドレスを非表示にし、非セキュア領域ドのコードがリターンアドレスを操作しないようにします(図11-8 を参照)。

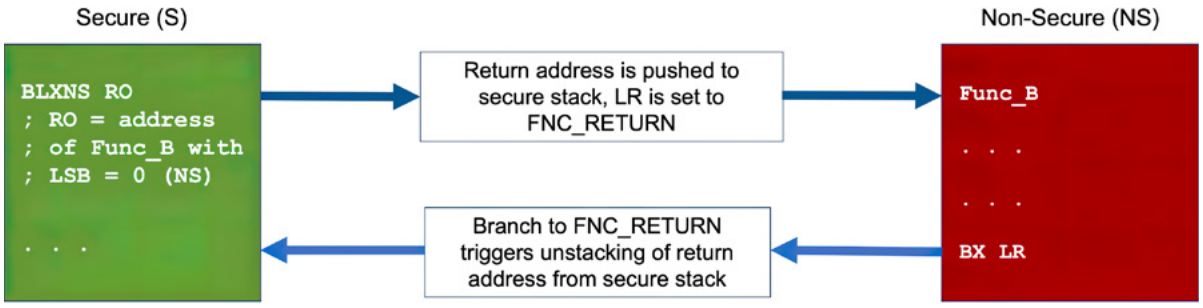

図11-8: セキュアコンテキストからの非セキュア関数の呼び出し

セキュアコードからBLXNS (リンク付き分岐および非セキュア状態への交換)命令を介して非セキュアコードを呼び出すためのより良いアプローチは、リセット後にセキュア領域内のコードを初期化し、その後プログラム制御を非セキュア領域に渡しFSP コールバックを介してデータ転送を管理することです。

11.2.2 セキュアコードから非セキュアコードへのコールバック

ペリフェラルがセキュア側にあるがその割り込みを処理するコールバック関数が非セキュアな領域に配置されているシナリオをどのように処理するか考えてみましょう。通常、FSP コールバックではコールバック構造体はISR によってスタックに割り当てられ、その後コールバック関数によって使用されます。割り込みサービスルーチン(ISR) とコールバックが異なる領域に存在するため、コールバック関数がセキュアな領域のコールバック構造にアクセスしようとするため、セキュリティ障害が発生します。

FSP は両方の環境で使用可能なメモリ領域にコールバック構造を割り当てることによって、この問題を解決します。これはガード関数であるcallbackSet() API を使用して初期化されます。これは、ガード関数(ガード関数の説明については 11.2.3 章を参照) で、セキュア領域へのアクセスを許可します。この呼び出しは次のようになります:

```
fsp_err_t (* callbackSet)(uart_ctrl_t * const p_ctrl,
                          void (* p_callback) (uart_callback_args_t *),
                          void const * const p_context,
                          uart_callback_args_t * const p_callback_memory);
```

コールバック関数ポインタとコンテキストポインタは、モジュールのコンフィギュレーション構造で既に提供されていますが、セキュア側のコンフィギュレーション構造は非セキュア側の構造とは別に構築されるため、再度作成する必要があります。volatile コールバックメモリへのポインタは、両方の領域からアクセス可能な構造体のメモリを割り当てることができる場所を指します。これにより、セキュアな障害が排除されます。

11.2.3 ガードファンクション

ガードファンクションのアプリケーションプログラミングインタフェース(API)は、セキュアでないプロジェクトからセキュアな領域に存在するドライバへのアクセスを可能にします。ルネサスが実装しているこの機能は独特であり特許を取得しています。フレキシブルソフトウェアパッケージ（FSP）は、FSPコンフィギュレータで非セキュアな呼び出し可能としてマークされたすべての最上位モジュールおよび/またはドライバAPIのガード関数を自動的に生成し、それらをNSC領域のプロジェクトに追加します。 さらに、FSPは対応するNSCインスタンスの非セキュアモジュールインスタンスを作成します。

これらのインスタンスは通常と同じように使用できますが、p_ctrl およびp_cfg メンバがNULL と同等のFSP_SECURE_ARGUMENT に設定され、それらのp_api メンバが実際のメンバ関数ではなくガード関数を指します。ガード関数自体には、セキュア領域のメモリにハードコードされたp_ctrl メンバとp_cfg メンバがあります。セキュアメモリと非セキュアメモリの両方にドライバが存在し、異なるチャネルが異なる側で使用されているため、p_ctrlやp_cfg構造を操作することで、セキュアチャネルに非セキュアコードから直接アクセスできる可能性がなくなります。また、ガード関数は呼び出し元がセキュアメモリを上書きしないように入力ポインタをチェックします。

```
const uart_api_t g_uart0_api = {
                .open            = guard_g_uart0_R_SCI_UART_Open,
                .close           = guard_g_uart0_R_SCI_UART_Close,
                .write           = guard_g_uart0_R_SCI_UART_Write,
                .read            = guard_g_uart0_R_SCI_UART_Read,
                .infoGet         = guard_g_uart0_R_SCI_UART_InfoGet,
                .baudSet         = guard_g_uart0_R_SCI_UART_BaudSet,
                .versionGet      = guard_g_uart0_R_SCI_UART_VersionGet,
                .communicationAbort = guard_g_uart0_R_SCI_UART_Abort,
                .callbackSet     = guard_g_uart0_R_SCI_UART_CallbackSet,
};
/* Create non-secure instance that has a non-secure callable API */

const uart_instance_t g_uart0 =
{
.p_ctrl = FSP_SECURE_ARGUMENT, // CTRL is static in guard function
.p_cfg  = FSP_SECURE_ARGUMENT, // CFG is static in guard function
.p_api = &g_uart0_api
};
```

さらに、設計者はセキュアでないプログラムに限られた範囲のAPIのみを公開したい場合、追加レベルのアクセス制御を追加するか、またはガード機能を削除するかを選択することができます。SCI の例に従いセキュアな領域のプログラムはチャネルをオープンし所望のボーレートに設定することができますが、g_uart0_write_guard() API 以外をすべて削除することで非セキュアなアプリケーションの開発者は書き込みAPI のみを使用できます。

11.3 デバイスライフサイクルマネジメント

デバイスのライフサイクルは、デバイスのライフサイクルのさまざまなフェーズを定義し、デバッグインタフェース、シリアルプログラミングインタフェース、およびルネサステストモードの機能を制御します。このようにセキュアな領域に存在するコードの記述と、非セキュアな領域で実行されているアプリケーションの記述は、セキュリティ上の理由から分離でき製品は、2つの独立したチームによって開発することができます。つまり、ルートオブトラスト(Root of Trust)または分離されたサブシステムを作成するセキュアな開発者のチームと、そのRoTまたはサブシステムを使用するアプリケーションを作成する非セキュアな環境の設計者です。このデザイン分割はフレキシブルソフトウェアパッケージ(FSP) およびe² studioでサポートされています。

セキュア領域のコードの準備ができたらデバイスに事前にプログラムしセキュア領域をロックするNSECSD にライフサイクルを設定するか、非セキュアプロジェクトによってバンドルとして参照することができます。アプリケーション設計者は、ここから引き継ぎ非セキュア領域でアプリケーションを記述しデバッグしてデバイスに書き込みます。必要に応じて使用されるフラッシュメモリブロックのプログラムおよび消去機能を無効にすることもできます。最後の手順として、ライフサイクル状態をデプロイ済み、デバッグロック、またはブートロックのいずれかに設定します。これにより、デバイス全体が保護されプログラミングインタフェースとデバイスをデバッグ、読み取り、またはプログラムすることができなくなります。図11-9 は可能な状態と遷移を示し、図11-10 の表は各ライフサイクルの説明を示しています。

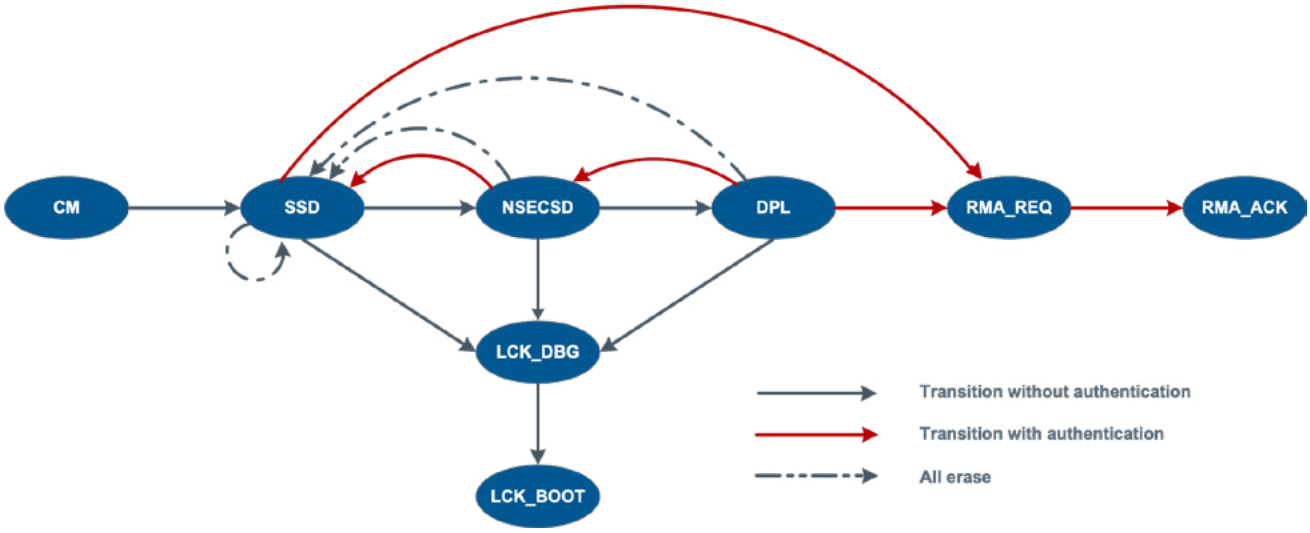

図11-9: デバイスライフサイクル管理のさまざまな状態

3 つの異なるデバッグアクセスレベルがあり、ライフサイクル状態に応じて変化します:

- **DBG2:** デバッガ接続は許可されており、メモリやペリフェラルへのアクセスに制限はありません。
- **DBG1:** デバッガ接続は許可されますが、アクセスは非セキュアメモリ領域とペリフェラルのみに制限されます。
- **DBG0:** デバッグ接続は一切許可されません。

Lifecycle	Definition	Debug Level	Serial Programming	Test Mode
CM	"Chip Manufacturing" The state when the customer received the device.	DBG2	Available, cannot access code/data Flash memory	Not available
SSD	"Secure Software Development" The secure part of the application is being developed.	DBG2	Available can program/erase/read all code/data Flash memory areas	Not available
NSECSD	"Non-Secure Software Development" The non-secure part of the application is being developed.	DBG1	Available can program/erase/read non-secure code/data Flash memory areas	Not available
DPL	"Deployed" The device is in the field.	DBG0	Available cannot access code/data Flash memory area	Not available
LCK_DBG	"Locked Debug" The debug interface is permanently disabled.	DBG0	Available cannot access code/data Flash memory area	Not available
LCK_BOOT	"Locked Boot Interface" The debug interface and the serial programming interface are permanently disabled.	DBG0	Not available	Not available
RMA_REQ	"Return Material Authorization Request" Request for RMA. The customer must send the device to Renesas in this state.	DBG0	Available cannot access code/data Flash memory area	Not available
RMA_ACQ	"Return Material Authorization Acknowledged" Failure analysis at Renesas	DBG2	Available cannot access code/data Flash memory area	Available

図11-10: デバイスサイクル管理の各段階の説明

あるステートから別のステートへの遷移は、Renesas Flash Programmer またはRenesas Device Partition Manager のいずれかを使用して実行できますが、後者の場合は一部のステートしか選択できません。状態間の遷移は、認証キーを使用して保護できます。各マイクロコントローラのユーザーズマニュアルで、さまざまなDLM 状態とデバイス固有の遷移の詳細を確認できます。

11.4 TrustZone®の使用事例

TrustZone®が何であるか、データと知的財産(IP) 保護の導入にどのように役立つか、ルネサスによる実装の利点が何であるか、セキュアな領域と非セキュアな領域の分離がどのように見えるかがわかったので、いくつかのユースケースを確認してみましょう。この章の次の部分では、TrustZone がIP の保護をどのように支援し、法的に関連するコードの分離をサポートし、Root of Trust (RoT) を安全に保つかについて説明します。

11.4.1 事前にプログラムされたアルゴリズムのIP保護

アプリケーションが改ざんから保護されるべき機能的アルゴリズムにアクセスしなければならない場合、セキュアアルゴリズムの実装と残りのコードとの間で開発を分割することができることは大きな利点となります。アルゴリズムの設計者は、まず、e² studioのプロジェクトおよびFSPコンフィギュレータを使用して、定義されたアプリケーションプログラミングインタフェース(API)でセキュアプロジェクトを作成し、アルゴリズムを記述し、使用可能なデバッグインタフェースのいずれかを使用してデバッグします。次に、必要に応じてマイクロコントローラに書き込み、プログラムを無効にするかフラッシュメモリの使用済みブロックの機能を消去して保護します。安全なプロジェクトはTrustZone を自動的に設定します。事前にプログラムされたデバイスをアプリケーション開発者に渡す前に、セキュアチームはライフサイクル状態を非セキュアソフトウェア開発(NSECSD) に設定し、デバッガやフラッシュプログラマがアルゴリズムを読み取れないようにします。

その後、アプリケーションライタはe² studioで非セキュアプロジェクトを作成しアプリケーションを記述しデバッグインタフェースのいずれかを使用してデバッグします。アプリケーションはセキュアプロジェクトの公開されたAPI のいずれかを問題なく呼び出すことができます。完了すると、最終コードをマイクロコントローラにプログラムし、使用されているフラッシュメモリブロックのプログラムまたは消去を無効にし、デバイスのライフサイクル状態をデプロイ済み(DPL)、デバッグロック(LCK_DBG)、またはブートロック(LCK_BOOT)に設定します。これで、ユニット全体が保護され、顧客への出荷準備が整いました。

TrustZone の大きな利点は、TrustZone がアルゴリズムを不注意や悪意のある誤用から保護し、セキュアなパーティと非セキュアなパーティの間でアプリケーション設計を分割できることです。しかし、修正が必要なセキュアアルゴリズムに欠陥がある場合はどうなるでしょうか?　図11-9から、セキュアな開発者によって消去機能が無効になっていなければ、保護アルゴリズムを消去することで、SSD状態に戻る道があることがわかります。これにより、事前にプログラムされたデバイスの廃棄を最小限に抑えることができます。

11.4.2　スマートメータにおける法的に関連するコードの分離

スマートメータのヨーロッパ仕様では、認定を受ける合法的に関連するコードが定義されています。このコードは、他のメータから隔離する必要があります。現在、ほとんどの顧客は、2つのマイクロコントローラを使用して、何らかの物理的分離を採用しています。これは高価ですが、認証を簡素化します。

もう1つの可能性は、TrustZoneを使用する1つのマイクロコントローラで、表示用のコードやDLMS / Cosem (Device Language Message Specification / Companion Specification for Energy Metering)のように、法的に関連するコード、データ、ペリフェラル、およびアプリケーションコード間で論理的な分割を行うことです。これにより、TrustZone は、1 つのデバイスでコードの誤用や破損に対する証明可能な隔離と保護を提供します。

11.4.3 Root of Trustの保護

11.1章で説明したように、Root of Trust (RoT)は、完全な製品のセキュリティ基盤を構築するため、保護する必要があります。すべての上位レベルのセキュリティは、RoTの上に構築され、認証されたファームウェアアップデートとセキュアな通信を提供します。さらに、RoT は上位レベルのセキュリティ障害からのリカバリを可能にしますが、それが侵害された場合、その上に構築されたものは安全ではなくなります。

つまり、すべてのファクトリキー、デバイスID、チェックサム、Flashイメージの検証、暗号サービス、キーと証明書、および機密の高い日付情報は、セキュア領域内に保持する必要があります。プライマリアプリケーション、ユーザインタフェース、インタフェースプロトコル、サービス、その他の安全でないものは非セキュア領域に置かれるべきです。攻撃されうる界面をできるだけ小さくセキュアな環境に保つために、アプリケーション全体の実行はセキュアではない領域に保つ必要があります。

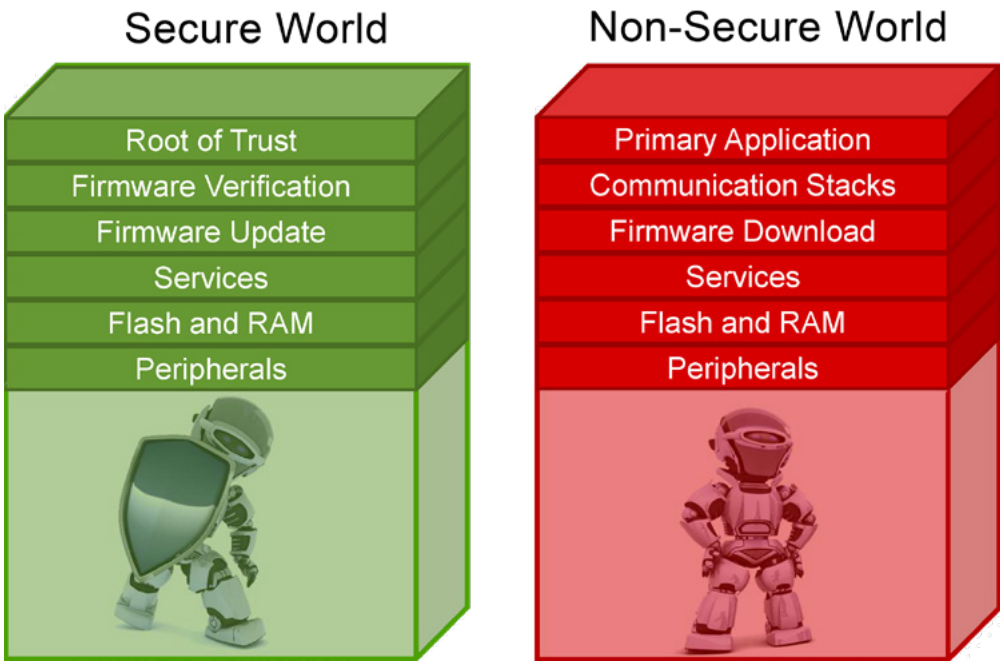

図11-11: セキュアな領域と非セキュアな領域で保持する内容。

本章で学んで会得するポイント:

- TrustZone は、セキュア領域と非セキュア領域の分離を容易にします。
- ルネサスのTrustZoneの実装により、起動時のセキュリティギャップが解消されます。
- セキュアと非セキュアの2つのプロジェクトが必要です。
- non-secure callable セクションは、ガードファンクション を介したセキュア領域へのコールを許可します。
- ライフサイクルマネジメントは、異なるチーム間で開発を分割するのに役立ちます。

12 WHERE TO GO FROM HERE

この章で学ぶこと:

- RAパートナのエコシステムとは何か、またどのように開発時間の短縮に役立つか
- サンプルプロジェクトの入手先
- その他のオンライントレーニング、ホワイトペーパー、アプリケーションノートの入手先

これまでRA Familyマイクロコントローラの基本的な機能、そのプログラムの作成方法、およびソフトウェアとハードウェアのツールに関して利用可能なサポートについて学んだのですが、もっとありますか?と尋ねるかもしれません。答えはyes です。この章では、ルネサスまたはそのパートナーから入手可能な追加サポートへのリンクとヒントを提供します。

12.1 ルネサスRAパートナエコシステム

組込みシステムの設計複雑性はこの数年にわたって指数関数的に増加し、それに伴い設計上の課題が出てきますし新しい技術も開発されています。EdgeやIoTデバイスの開発者は、開発難易度が上がり開発時間が短縮され続けるにつれてプロジェクトを時間どおりに提供することに苦労しています。これにより、ソフトウェアであれハードウェアであれ、柔軟ですぐに使用できるソリューションとビルディングブロックの必要性が高まっています。

ルネサスは、50社を超えるパートナとともに、RAパートナエコシステム内の幅広いサードパーティ製アプリケーションとビルディングブロックを使用して、このアプローチをサポートしています。このエコシステムは、エンジニアがすぐに使えるプラグアンドプレイオプションを提供することで、モノのインターネット(IoT)のための組み込みシステムの開発を加速するのに役立ちます。これらは、セキュリティ、音声ユーザインタフェース、グラフィックス、機械学習、人間-機械インタフェース、クラウド接続などにまで及びます。新しい各パートナビルディングブロックソリューションには、製品が実際の顧客の問題を解決するように設計されていることを示す「RA READY」バッジがラベル付けされています。パートナエコシステムのWebサイトで詳細を確認できます。:
https://wwwrenesas.com/ra-partners

図12-1: Renesas RA Partner Ecosystemは、50社を超えるパートナのすぐに使用できるソフトウェアとハードウェアのコンポーネントで構成されています。

12.2 サンプルプロジェクト

開発をスピードアップしプロジェクトを時間どおりに提供できるようにルネサスでは無償でダウンロードして使用できるサンプルプロジェクトをいくつか用意しています。フレキシブルソフトウェアパッケージ(FSP)のユーザーズマニュアルには、API およびモジュールごとに短いサンプルが用意されていますが、より包括的なサンプルプロジェクトバンドルも用意されており、選択した評価キットですぐに実行できます。これらのバンドルについては、ツールのインストール方法とプロジェクトのインポートおよび実行方法を説明した「Usage Guide」があります。

バンドルには、A/D コンバータ、フラッシュメモリ、USB、SPI またはUART 接続、FreeRTOS™、Azure RTOSなど、ほとんどのオンチップペリフェラルのプロジェクトなどが含まれます。バンドル内の各プロジェクトには、提供される機能、ハードウェア要件、ジャンパ設定、およびハードウェア接続の概要を示すReadMe ファイルが付属しています。

その他のサンプルプロジェクトも利用できます。たとえば、IEC 60730/60335 セルフテストライブラリ、モータ制御ソフトウェア、またはセンサネットワークソリューションのサンプルコードなども利用できます。ダウンロードは、評価キットのホームページまたはRAマイクロコントローラのホームページからアクセスできます。

12.3 オンライントレーニング、ホワイトペーパー、アプリケーションノート

この本が終わりに近づいてきましたが、「マイクロコントローラのRAファミリについてもっと知ることができるのはどこか」という疑問が沸いてくるかもしれません。このトピックを更に詳しく知りたい方には個人教育のための情報、トレーニング、チュートリアルが豊富に用意されています。本章の次の段落では、それらのいくつかを取り上げますが、他にもいくつかあり、それらのほとんどへのリンクはRAファミリのウェブサイト(ツールとサポートの下にある**https://www.renesas.com/ra**)にあります。

12.3.1 オンライントレーニング

オンライントレーニングを提供するサイトには、ルネサスアカデミーとルネサスのYouTube™公式RAチャンネルの2つがあります。ルネサスアカデミーは、すべての学習教材を1カ所にまとめたオンライン学習プラットフォームで、自分の好きなもの、好きなときに、自分のペースで学習する便利さを提供しています。マイクロコントローラのRA ファミリだけでなく、Synergy、RX、RZ、アナログ製品、パワー製品など、ルネサスの他の製品ファミリも対象としています。

必要なトレーニングのカタログはオンラインですばやく見つけることができます。各コースはスケジュールに合わせて短いモジュールに分類されています。必要なのはルネサスアカデミーサイト(**https://academy.renesas.com**)での簡単なワンタイム登録だけで、好きなトレーニングを行う準備が整っています。現在のコースには、RAファミリの異なるマイクロコントローラへの導入やRAパートナエコシステムのトレーニングなどのトピックが含まれています。新しいコースや更新されたコース、コンテンツ、機能が継続的に追加されるためこまめに確認してください。

ルネサスのWebサイト（**https://www.renesas.com/eu/en/products/**）には、RAファミリのビデオライブラリ（**https://www.renesas.com/eu/en/products/microcontrollers-microprocessors/ra-cortex-m-mcus/ra-video-library**）もあります。 そこから、さまざまなトピックをカバーするビデオにアクセスできます。 これらにはプロセッサとキットのプレゼンテーション、RAファミリまたはRAエコシステムの紹介、およびいくつかのウェビナーが含まれますが、これだけではありません。 またツールの使用方法に関するビデオには独自のWebサイトがあります。RAファミリソフトウェアおよびツールコースページ（**https://www.renesas.com/eu/en/software-tool/ra-software-tool-course**）があり、ここではツールの紹介と「ハウツー」トレーニング、およびソリューション開発について学ぶことができます。

最後にルネサスのYouTubeの公式RAプラットフォームチャンネル(**https://www.youtube.com/playlist?list=PLgUXqPkOStPv2KcNs_1Wc5MY31GOPPAd3**)は、マイクロコントローラのRAファミリのチュートリアルとトレーニングビデオを提供します。RAマイクロコントローラの紹介、ツール&キット、ボードやe² studioの使い方を説明したビデオなどがあります。追加コンテンツが継続的に追加されます。

12.3.2 ホワイトペーパーとアプリケーションノート

ルネサスは、ハードウェアとソフトウェアの設計トピック、セキュリティ、IoT接続性、モータ制御、ヒューマン·マシン·インターフェースなど、数多くのホワイトペーパーとアプリケーションノートを提供しています。これらはすべて基礎をしっかりと学ばなくても設計上の問題を解決するために役立ち、開発をスピードアップします。サンプルプロジェクトと同様に、マイクロコントローラのページから、または使用する評価キットのページからアクセスできます。また、RAファミリのホームページが良い出発点となるでしょう。

ホワイトペーパーとアプリケーションノートルネサスが提供するすべてのホワイトペーパーを参照する場合は、次のページに移動し、目的のパーツ番号をフィルタとして入力します。**https://www.renesas.com/eu/en/support/document-search?doc_category_tier_1=&doc_category_tier_2=&doc_category_tier_3=&doc_category_tier_4=&doc_part_numbers=&title=&sort_order=DESC&sort_by=field_document_revision_date#documentation-tools-results**。

本章で学んで会得するポイント:

- サンプルプロジェクトは、アプリケーション全体またはRAマイクロコントローラの特定の部分を対象としています
- オンラインビデオは、ツールやデバイスに慣れるための迅速な方法を提供します
- RA パートナエコシステムは、サードパーティからすぐに使用できるビルディングブロックを提供します
- ホワイトペーパーやアプリケーションノートは、幅広いトピックをカバーしています

著者について

Richard Oedは、組み込み分野において23年以上フィールドアプリケーションおよびシステムエンジニアとして活躍し、DSP、マイクロコントローラ、マイクロプロセッサ、およびデータコンバータの開発協力やソフトウェア開発を行っています。2015年秋からフリーランスの作家、著者、ジャーナリストとして活躍。 Richardは3つの特許の共同発明者であり、IEEE、ACM、VDEのメンバーです。彼はまた、**www.renesas.com/synergy-book** でルネサスから入手できるコンテンツである「Basics of the Renesas Synergy-Platform」の著者でもあります。

Before purchasing or using any Renesas Electronics products listed herein, please refer to the latest product manual and/or data sheet in advance.

Renesas Electronics Corp.

www.renesas.com

Document No. R01TU0337ED0200