Arm Keil Microcontroller Development Kit (MDK) v6

Getting Started Guide

◆著作権

序章

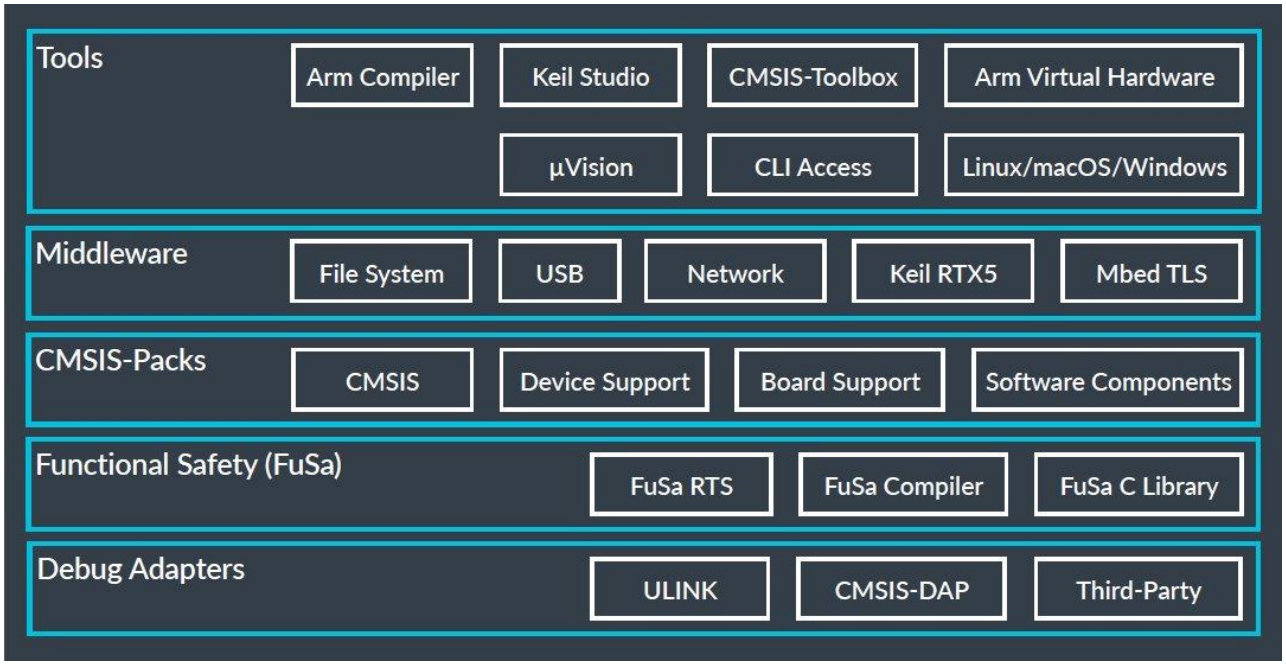
この度は、弊社より Arm 製ソフトウェアツールをご購入いただきありがとうございます。
このガイドブックは Arm 開発ソフトウェア Arm Keil Microcontroller Development Kit(MDK) v6 の導入に際し、速やかな立ち上げを支援します。
本ドキュメントは Arm の提供する" ***Arm Keil Microcontroller Development Kit(MDK) v6 Getting Started Guide Issue 05 (Document ID: 109350_v6_05_en)***" の内容に基づき翻訳、作成されたものです。内容につきましては全て上記ドキュメントをマスターといたしておりますので、ご使用の際には必ず上記ドキュメントを参照の上、本ドキュメントは参考資料として用いる形をお取りくださいますようお願い申し上げます。

目次

1. MDK とは？

Arm Keil Microcontroller Development Kit (MDK) は、Arm Cortex-M および Ethos-U プロセッサベースの組み込みアプリケーション開発用ソフトウェアツールを集めた製品です。MDK は、CLI または IDE(デスクトップベースまたはブラウザベース)の使用および継続的インテグレーション(CI) のワークフローにツールをデプロイできる柔軟性によって、ソフトウェア開発を簡単かつ生産的にします。

図 1-1: MDK の概要図



1.1 ツールファミリ

MDK は以下を含みます:

- Keil Studio
- Keil uVision
- Arm Compiler for Embedded Version 6 は革新的な LLVM と Clang のテクノロジに基づいており、C++17 を含む最新の言語標準をサポートしています。
- Arm Virtual Hardware (AVH)

MDK は、Common Microcontroller Software Interface Standard (CMSIS) に基づく開発フローを使用します。組み込みシステムでは製品開発に数年を要することが多いため、MDK では製品開発の開始から完成、メンテナンスまでの全ライフサイクルをサポートします。

MDK は、Linux、macOS、および Windows のホストをサポートしています。

Arm Virtual Hardware simulation models (Fixed Virtual Platform models または FVP) は現状 macOS では使用できません。uVision は Windows でのみ実行可能です。

1.2 CMSIS-Pack

CMSIS-Packs には、デバイスとボードのサポート、ソフトウェアコンポーネント、ミドルウェア、コードテンプレート、およびサンプルプロジェクトが含まれています。新しいデバイスサポートやミドルウェアの更新はツールチェインから独立しており、これらはいつでもツールに追加できます。IDE および CLI ツールはアプリケーションのビルディング・ブロックとして使用できる、ソフトウェアコンポーネントを管理します。

1.3 機能安全(Functional Safety：FuSa)

MDK-Professional エディションでは、機能安全アプリケーション開発に必要なコンポーネントが含まれています：

- Arm Compiler for Embedded FuSa
- 認証済み C ライブラリ
- FuSa Run-Time System (RTS)

1.4 デバッグアダプタ

MDK は、Arm ULINK ファミリのデバッグアダプタおよびトレースアダプタと連携します：

- ULINKpro は、独自のストリーミングトレーステクノロジを使用してアプリケーションをプログラミング、デバッグ、および分析できるデバッグおよびトレースユニットです。
- ULINKplus は分離されたデバッグ接続、パワー計測、テスト自動化のための I/O が組み合わされています。
- ULINK2

また、さまざまなサードパーティ製ツール、スターターキット、デバッグアダプタ(ST-Link、JLink など)を使用して MDK を拡張して使用することもできます。

1.5 MDK のエディション

MDK には、以下のエディションがあります。

- **MDK-Community**：評価、ホビー、メーカ、アカデミックおよび学生による非商用利用向け
- **MDK-Essential**：Arm Cortex-M ベースのマイクロコントローラプロジェクトの商業開発用
- **MDK-Professional**： 機能安全(FuSa) 要件やシミュレーションモデルを使用した DevOps の必要性を伴うプロフェッショナル向け。本エディションのオールインワンソリューションでは、Embedded FuSa 用の Arm Compiler が含まれており、すべての Arm Virtual Hardware Fixed Virtual Platform (FVP) にアクセス可能。また、PK51、DK251、PK166 および Arm Compiler 5 等の旧ツールも利用可能

product selector では、エディションごとに利用可能な機能についてのオーバービューがあります。

1.6 ライセンスのタイプ

すべての MDK のエディションではライセンスコードを使用したアクティベーションが必要です。

MDK は Arm 製品を使用する資格がユーザに紐づけられる user-based ライセンシング(UBL) をサポートしています。ユーザは複数のデバイスにおける同一製品の使用を含む、同時使用について制限のない Arm 製品ライセンスを使用する権利を持ちます。たとえば、サービスアカウントで 1 つのライセンスを使用して、任意の数のデバイス上で Arm 開発ツールを使用して製品の自動ビルドやテストを行えます。

ライセンスを取得する方法は複数あります:
Arm またはライセンス管理者から提供されるアクティベーションコードを使用する
ライセンス管理者が管理するローカルライセンスサーバへのアクセスを行う

ライセンス認証の詳細については、User-based Licensing User Guide 内"Activate your product using an activation code" および"Activate your product using a license server" の章を参照してください。

Keil Studio for Visual Studio Code を使用している場合は、"Activate your license to use Arm tools" の章を参照して、**Arm License Management Utility** ユーザインターフェイスを開き、アクティベーションコードまたはライセンスサーバの情報を入力してください。

管理者でありシリアル番号を使用して Arm user-based licensing portal でアカウントに製品を追加する、あるいは既存の製品にライセンスを追加する必要がある場合は、Accessing the Arm License Portal video tutorial を参照してください。より詳細については User-based Licensing Administration Guide を参照してください。アクティベーションコードの作成方法は Cloud-based Licenses and Activation Codes video tutorial を参照してください。また、User-based Licensing Administration Guide も併せて参照してください。

user-based ライセンシングサポートと下位互換性の詳細については、User based licensing User Guide を参照してください。Backwards compatibility の章では、Keil MDK Professional を含む製品ライセンスを使用する MDK(MDK 5.36 以前)や PK51、PK166、および DK251 などの古い製品のライセンスの利用について説明があります。

1.7 ダウンロードオプション

MDK v6 には様々なインストール方法によってインストールできる各種ツールが含まれます。

表 1-1: MDK v6 のダウンロードオプション

Tool	keil.com	PDH	Artifactory	その他
uVision(MDK v5)		ダウンロード		
Keil Studio				VS Code Marketplace
Arm Compiler for Embedded		ダウンロード	ダウンロード	
LLVM			ダウンロード	
GCC		ダウンロード	ダウンロード	
Arm CMSIS-Toolbox			ダウンロード	
Arm Debugger			ダウンロード	
Arm License Manager			ダウンロード	
Arm Virtual Hardware FVPs			ダウンロード	
uVision Project Converter			ダウンロード	
Arm Compiler for Embedded FuSa*		ダウンロード		
Functional Safety Run-Time System*		ダウンロード		
PK51*	ダウンロード			
DK251*	ダウンロード			
PK166*	ダウンロード			

- *(アスタリスク) でマークされたすべてのツールは MDK-Professional エディションをご購入の場合のみ使用いただけます。
- Product Download Hub(PDH) へのアクセスには Arm アカウントの登録が必要です。

Artifactory からのダウンロード

Artifactory からツールをダウンロードする最も簡単な方法は vcpkg の使用です。vcpkg は開発環境を簡単に構築または再作成できる管理ユーティリティです。
CLI または Arm Environment Manager extension for VS Code (Keil Studio Pack の一部として利用可能) を介してツールをダウンロードおよびインストールします。

Install tools on the command line using vcpkg learning path では vcpkg を利用して PC またはサーバにどのように追加するかや `vcpkg-configuration.json` ファイルを使用したツールのダウンロード方法について説明があります。

Arm の公式サンプルには、コンフィギュレーション済みの `vcpkg-configuration.json` ファイルが付属しています。このファイルは、Keil Studio Pack を使用して Visual Studio Code で `.uvpmw/.uvprojx` ファイルを変換するときにも作成されます。

環境内のツールを追加または変更するには、インストールするパッケージを `vcpkg-configuration.json` ファイルの"`requires`"セクションに追加します。ファイルを保存すると、新しく指定されたパッケージがダウンロードされ、アクティブ化されます。

`curl` や `wget` などのアプリケーションを使用してツールを直接ダウンロードすることもできます。

1.8 MDK ドキュメンテーションへのアクセス

MDK は Arm Developer においてのすべてのコンポーネントに関するドキュメントを提供しています。

ツールの使用開始にあたり以下のドキュメントを参照することを推奨します:

- Arm Keil Studio Visual Studio Code Extensions User Guide
- Arm Keil Studio Cloud User Guide
- uVision User's Guide
- Licensing User's Guide

Get help

Keil MDK 製品に関する提案や、問題を発見した場合は Arm まで報告してください。問題の報告の際には icense code と製品のバージョンを書き添えてサポートケースを起こしてください。

MDK-Community エディションをお使いの場合は Keil Support Forum で問題を報告してください。

オンラインでの学習

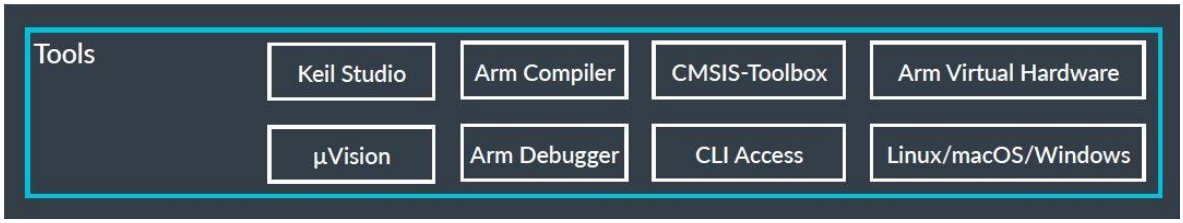
Arm の Learning Paths では、Arm Cortex ベースのマイクロコントローラのプログラミングについてより詳しく学ぶことができます。このサイトには初心者から上級者まで、あらゆるレベルの方向けのチュートリアルがあります。

ツールやソフトウェア開発の様々な側面を紹介するビデオがあります。

2. ツール

MDK v6 に含まれるソフトウェアツールについて詳しく説明します。

図 2-1: ツールの概要図



2.1 Keil Studio

Keil Studio は、Cortex-M デバイス用の組み込み、IoT、および機械学習ソフトウェアに対する評価と開発のための完全な開発ツールです。クラウド環境での開発のためのブラウザベースの統合開発環境(IDE: Integrated Development Environment) として、または Visual Studio Code を使用したデスクトップ開発用拡張セットとして使用できます。

Keil Studio Cloud は、コード用のクラウドホスト型ワークスペース、包括的なバージョン管理システム、強力な C/C++ エディタを提供します。以下が可能です:

- 任意のコンピュータからプロジェクトを編集し、同僚と共有、デスクトップ用にエクスポートする
- Arm Compiler for Embedded を使用したプロジェクトのコンパイル
- サポートされている開発ボードで直接プロジェクトを実行する
- サポートされているブラウザ(Chromium ベースのブラウザ) からのデバッグ。ソフトウェアのインストールは不要

2.1.1 Keil Studio Pack for Visual Studio Code

Arm Keil Studio Pack には、CMSIS ソリューション(csolution プロジェクト) を管理し、Visual Studio Code を使用して、選択したハードウェア上で組み込みアプリケーションを作成、ビルド、テスト、およびデバッグできる拡張機能が含まれています。

使用可能な拡張機能の情報、および Visual Studio Code でパックをインストールする方法については、Arm Keil Studio Pack for Visual Studio Code のページを参照してください。作業環境の設定方法と開始方法については、Get started with an example project を参照してください。

2.2 Keil uVision

Keil uVision は、組み込みアプリケーションを迅速かつ有効に開発するために必要なすべてのツールを統合した Windows ベースのソフトウェア開発プラットフォームです。uVision は、ソースコードエディタ、プロジェクトの作成および保守を行うプロジェクトマネージャ、組み込みアプリケーションのアセンブル、コンパイル、およびリンク用の make ツールが組み合わさっています。

uVision では、アプリケーションの構築とデバッグに対して個別のモードが用意されています。Arm Virtual Hardware シミュレーションモデルを使用するか、または直接ハードウェア上で(たとえば、Arm Keil ULINK ファミリのデバッグおよびトレースアダプタを使用して)、アプリケーションをデバッグできます。サードパーティ製のデバッグプローブを使用してアプリケーションを分析することもできます。ULINK デバッグおよびトレースアダプタは、事前設定済みフラッシュプログラミングアルゴリズムによってアプリケーションプログラムのフラッシュダウンロードを行います。

uVision はアプリケーションのテストと検証を十分に行えるよう、統計データと実行解析レポートを提供します。これは、セーフティクリティカルなシステムに取り組む場合に特に重要です。

uVision は以下も含んでいます:

- **System Viewer** : ペリフェラルレジスタに関する情報を表示し、実行時にプロパティ値を手動で変更します。
- **Logic Analyzer** : 時間グラフ上で値の変化の表示、信号と変数の変化の解析、およびそれらの依存関係または相関関係を表示します。
- **Template editor** : 共通のテキストシーケンス、ヘッダの説明、および汎用コードブロックを作成します。
- **Source Browser** : コード化された操作を素早く行えます。
- **Configuration Wizard** : GUI を使用して、デバイスやスタートアップコードの設定を管理します。
- **Multi-Project Manager** :相互に論理的に依存関係のある uVision プロジェクトをひとつのマルチプロジェクトにまとめます。これにより、組み込みアプリケーション設計の一貫性と透明性が向上します。

より詳細については uVision のドキュメンテーションを参照してください。

2.3 Arm Compiler for Embedded

Arm Compiler for Embedded は小型センサから 64 ビットデバイスまでを含む、組み込みベアメタルソフトウェア、ファームウェア、リアルタイムオペレーティングシステム(RTOS) アプリケーションの開発および最適化のための最先端の組み込み C および C++ コンパイルツールチェーンです。

Arm Compiler for Embedded は Arm アーキテクチャに沿って開発されています。そのため、最新のアーキテクチャの機能や拡張に最も早く完全且つ正確なサポートを提供しており、どの Arm ソリューションが、要件やデザインの検証に最適かを評価できます。

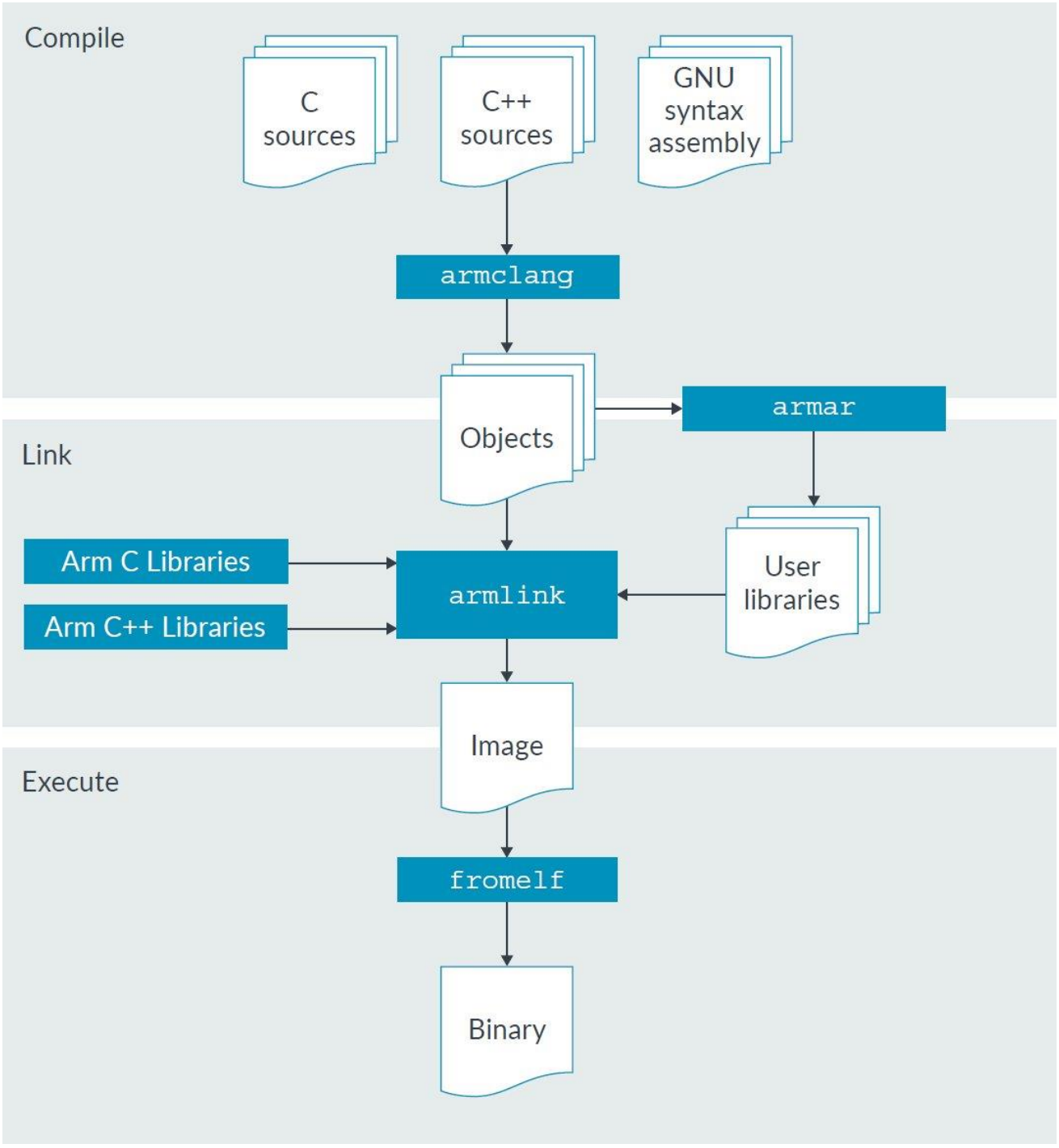
Arm Compiler for Embedded は家電製品、ネットワーク、ストレージ、電気通信、セキュリティ、セーフティクリティカルシステムなど、さまざまな業界の大手企業で使用されています。

Arm Compiler for Embedded は以下のツールチェインコンポーネントから構成されます:

- **armclang** : モダンな LLVM と Clang テクノロジに基づいたコンパイラと統合アセンブラ。armclang コンパイラは、GNU 構文アセンブリと C++17 を含む最新の言語標準をサポートします。GCC 用に書かれていたソースコードと高度に互換性があります。ANSI/ISO C および C++、ABI for the Arm architecture、ABI for the 64-bit Arm architecture、Arm C Language Extensions (ACLE) などの仕様が実装されています。
- **armlink** : オブジェクトとライブラリを組み合わせて実行可能イメージを生成するリンカです。
- **Arm C ライブラリ** : 組み込みシステム用のランタイムサポートライブラリ。これらのライブラリにはパフォーマンスとコード密度に対する最適化が含まれます。
- **Arm C++ ライブラリ** : LLVM libc++ プロジェクトに基づくライブラリです。
- **fromelf** : イメージの変換と逆アセンブリ用ツールです。
- **armar** : ELF オブジェクトファイルの設定をまとめて収集できるアーカイバ。
- **Arm Compiler for Embedded FuSa**(MDK-Professional 版のみ) : 自動車、産業、医療、鉄道、および航空を含むセーフティクリティカル市場向けの組み込みソフトウェアの開発に適した、安全認証済みの C/C++ ツールチェインです。
- **FuSa C ライブラリ** (MDK-Professional 版のみ)

次の図は、一般的な組み込みアプリケーションのビルドプロセスで、さまざまなツールチェインコンポーネントがどのように相互作用するかを示しています:

図 2-2：Arm Compiler ワークフロー図

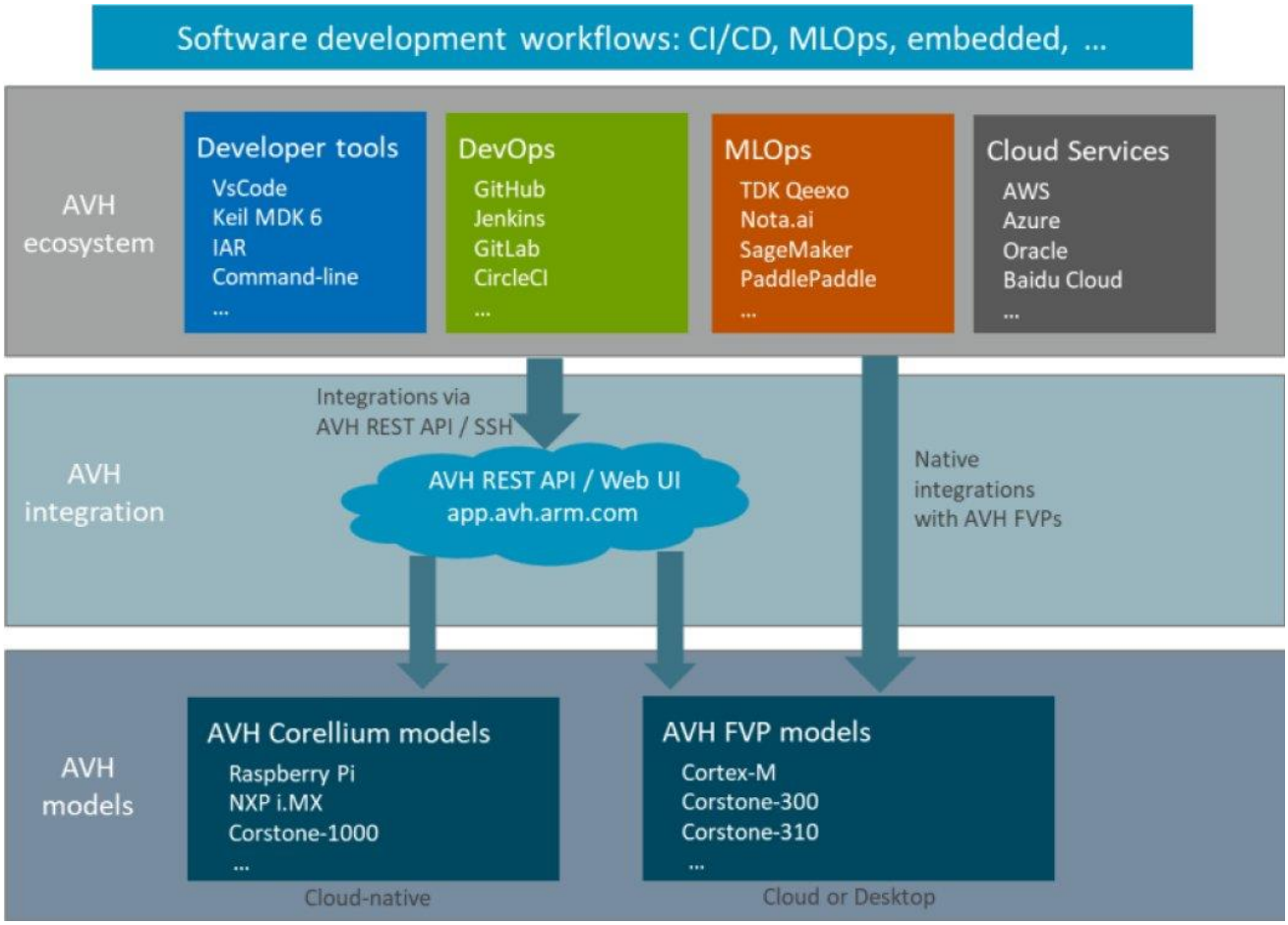


より詳細については、Arm Compiler for Embedded のドキュメントを参照して下さい。

2.4 Arm Virtual Hardware

Arm Virtual Hardware (AVH) では仮想ターゲットを使用して Arm ベースのプロセッサのソフトウェア開発を行えます。AVH は開発プロセスの簡略化、自動化、加速化に役立ち、メンテナンスコストを削減します。AVH を使用することでプロトタイピング、ビルド、および展開サイクルがより高速化し、組み込みアプリケーションのマーケット提供までの時間が短縮されます。

図 2-3: Arm Virtual Hardware 概要図



物理的なボードの設定やメンテナンスに手間をかけることなく、シリコンが出来上がる前に AVH でアプリケーションの開発やテストを開始できます。AVH は、Arm ベースの SoC の正確なシミュレーションを提供し、仮想モデルからターゲットハードウェアへのシームレスな移行を可能にします。

AVH は、組み込みおよび IoT プロジェクトのための継続的インテグレーションや継続的デリバリ環境を可能にし、MLOps のような開発プロセスをサポートします。IoT エンジニアやデータサイエンティストをこのような開発プロセスに参加することは、Internet of Things を数千、場合によっては数百万のデバイスに拡張するための鍵となります。AVH を使用すると、複数の仮想ボードを秒単位で起動し、複雑なマルチデバイスコンフィギュレーションに対して迅速に実験やテストを行えます。

MDK での AVH

MDK では、FVP(Fixed Virtual Platform Model) に基づく AVH をダウンロード、インストール、実行できます。FVP は、Arm Cortex-M ベースのコアおよび Corstone-300 や Corstone-310 などのリファレンスプラットフォームの正確なシミュレーションモデルです。

FVP モデルは、ターゲット環境で実行されるスタンドアロンプログラムです。これらはクラウドネイティブ環境およびデスクトップ環境で使用でき、コマンドラインまたは開発ツール上で実行できます。

現在利用可能なボードモデルや使用サンプルなどの情報については、AVH の User Guide および Solutions Overview を参照してください。

3. インストール

ここでは MDK v6 に含まれるソフトウェアツールのインストール方法を解説します。

MDK v6 では、単一のインストーラは提供されなくなりました。かわりに、次の開発プロジェクトで必要なツールを柔軟にインストールすることができるようになりました。

インストールオプションは以下の通りです:

- Windows 環境を使用中であれば、他のすべてのツールを含む uVision のインストーラをダウンロードできます。詳細については Windows での Keil uVision のインストール の章を参照してください。

- VS Code Marketplace を使用することで、Visual Studio Code 内からデスクトップマシンに Keil Studio をインストールできます。詳細については Keil Studio のインストール の章を参照してください。これには、Artifactory を使用して追加のツールをインストールする必要があります。

- Keil Studio 用の追加のツールをインストールするか、サーバ上で実行すると、Artifactory を使用してコンパイラ、モデル、CMSIS-Toolbox および Arm Debugger にアクセスできます。詳細についてはその他のツールのインストール の章を参照してください。

- Arm Compiler for Embedded FuSa や、FuSa C library などの機能安全コンポーネントへのアクセスは Arm の Product Download Hub からのみ可能です。

3.1 ソフトウェアおよびハードウェアの要求事項

MDK では以下のような最小ハードウェアおよびソフトウェア要件があります:

- 最新の 64-bit デスクトップオペレーティングシステム(Linux、macOS、Windows)を実行している PC
- GB の RAM と 8 GB のハードディスク容量
- 1280 x 720 ピクセル以上の画面解像度
- マウスまたはその他のポインティングデバイス

3.2 Keil uVision のインストール

ここでは Keil uVision のインストールとアクティベート方法について説明します。

Keil uVision のインストール

MDK をダウンロードし、インストーラを実行します。インストーラの指示に従って、MDK (uVision) をローカルコンピュータにインストールします。インストールを行うと Arm CMSIS および MDK-Middleware の Software Pack も追加されます。インストールが完了すると、Pack Installer が自動的に起動し、ここからさらに Software Pack を追加できます。少なくとも、対象のマイクロコントローラデバイスをサポートする Software Pack をインストールする必要があります。

Keil uVision のアクティベーション
インストールが完了したら、uVision にライセンスを登録する必要があります。ライセンスを購入していない場合は、こちらの手順に従うことで評価目的専用に提供されている無料の MDK-Community ライセンスを取得できます。

MDK v5.37 以降の uVision は User Based Licensing(UBL) をサポートしています。それ以前のバージョン(PK51、DK251、PK166 などのレガシーツールを含む) は、Keil node-locked ライセンスまたは FlexNet Floating License を使用した場合のみアクティベーションできます。旧バージョンのツールにアクセスする必要がある場合は、古い uVision のバージョンが使用できる MDK-Professional エディションを購入してください。

Keil Quick Tip では手順を短いビデオで紹介しています。

複数のバージョンの Keil IDE を同じフォルダにインストールした場合、UBL をアクティブ化すると、μVision に登録されている他のすべてのライセンスが上書きされることに注意してください。そのため、他の Keil ツールチェイン(PK51/DK251/PK166) は、アクティベートされた UBL ライセンスを使用する Keil MDK がインストールされているのとはフォルダとは別のフォルダにインストールする必要があります。詳細については Keil Licensing User's Guide を参照してください。

新規プロジェクトの作成
開発を始めるには uVision を使用した新しいプロジェクトの作成の手順に進んでください。

3.3 Keil Studio のインストール

ここでは Keil Studio のインストールとアクティベート方法について説明します。

Keil uVision のインストール
Keil Studio は、Arm Keil Studio Pack (拡張機能のコレクション) を Visual Studio Code に追加することでインストールされます。このパックは、Arm ベースのマイクロコントローラ(MCU) デバイス上の組み込みシステムおよび IoT ソフトウェア開発用のソフトウェア開発環境を提供します。

Visual Studio Code で、**View** ー **Extensions** を選択し、検索ボックスに"Keil Studio Pack" と入力してパックを検索します。**Install** ボタンをクリックして、拡張機能のセットをダウンロードしインストールを行います。

含まれる各拡張についての詳細は Arm Keil Studio Pack のページを参照してください。

Keil Studio のアクティベーション
インストールが完了したら、ライセンスを登録する必要があります。Keil Studio では User-based Licensing(UBL) のみサポートしています。ライセンスを購入していない場合は、こちらの手順に従うことで評価目的専用に提供されている無料の MDK-Community ライセンスを取得できます。

Keil Quick Tip では手順を短いビデオで紹介しています。

新規 solution の作成
最初のプロジェクト作成を行うには Keil Studio VS Code 拡張を使用した新しい solution の作成 の手順に進みます。

3.4 その他のツールのインストール

その他のツールのインストールについては以下のリンク先の内容に従ってください:

- Arm CMSIS-Toolbox
- Arm Compiler for Embedded
- Arm Virtual Hardware FVPs
- Arm GNU Toolchain (GCC)
- LLVM Embedded Toolchain
- Arm Compiler for Embedded FuSa
- Arm Debugger
- MDK Vision project converter

4. CMSIS コンポーネント

Common Microcontroller Software Interface Standard(CMSIS) は、Arm Cortex-M ベースプロセッサ向けのコードを記述するための一連のライブラリ、API、ソフトウェアコンポーネント、およびツールのセットです。

CMSIS は、多くのマイクロコントローラのメーカによってサポートされており、異なるマイクロコントローラの内部の詳細を知らなくとも、マイクロコントローラ向けのコードを記述するための標準化された方法を提供します。CMSIS を使用すると、コードの記述と再利用のプロセスが簡単になります。1 つのマイクロコントローラ用に書かれたブートコードを変更することなく別のマイクロコントローラに移植できるため、開発プロセスを高速化できます。

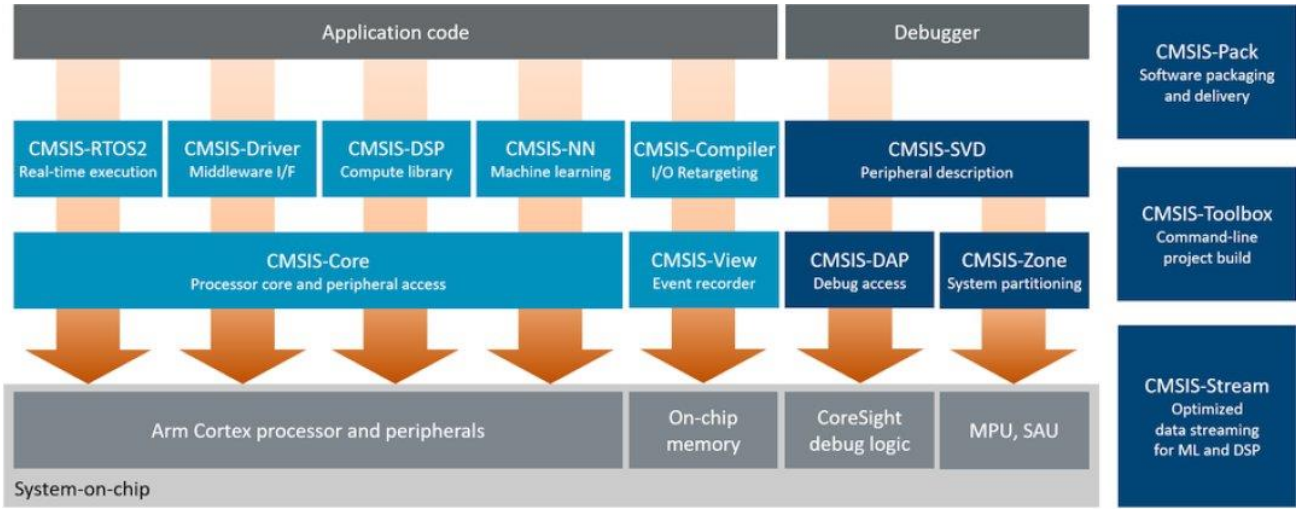
事前に記述された CMSIS の関数およびライブラリを使用すると、マイコンごとにそれらのリソースを直接操作する方法を学習しなくても、さまざまなマイコンのハードウェアリソースを制御できます。これにより、プロジェクトのビルドとデバッグにかかる時間が短縮され、新しいアプリケーションを市場に提供するプロセスが高速化されます。

また、CMSIS にはデジタル信号処理、機械学習やニューラルネットワークなどの機能を追加したり、複数のタスクやリソースを管理することでアプリケーションの機能を拡張しやすくするコンポーネントも含まれています。

CMSIS は、Apache 2.0 に従い GitHub 上で開発、公開されています。

CMSIS の概要

図 4-1: CMSIS の構造



開発者の目にふれる重要な CMSIS コンポーネント:

- CMSIS-Core : プロセッサコアおよびデバイスペリフェラルへのアクセスを標準化し、異なる Cortex-M コントローラ間で実行されるコードの記述を容易にします。
- CMSIS-RTOS2 : Arm Cortex プロセッサをベースとするデバイス用の汎用リアルタイムオペレーティングシステムインターフェイス。CMSIS-RTOS2 は、複数のタスクとリソースを管理および調整するプロセスを簡略化します。また、異なる RTOS カーネル間の移行プロセスにも役立ちます。

- CMSIS-Driver ： ペリフェラルとデバイスを設定および制御するための標準化された API を提供します。CMSIS-Driver はプラットフォームに依存しないように設計されており、サポート対象の様々なマイコンデバイス間でのコードの再利用を容易にします。
- CMSIS-Compiler ： 標準 C ランタイムライブラリで I/O 動作をリターゲットするためのソフトウェアコンポーネントと、例外や割り込み処理などのコア関数の標準 API を提供します。
- CMSIS-View ： 組み込みアプリケーションの開発およびデバッグ中に、マイクロコントローラ、ペリフェラル、ハードウェアコンポーネント、およびソフトウェアコンポーネントの内部動作を可視化します。
- CMSIS-DSP ： 広範にわたるデジタル信号処理関数およびルーチンを提供します。CMSIS-DSP アルゴリズムは、効率のために最適化されており、アプリケーションのパフォーマンスを最大化し、リソース使用量を最小化するのに役立ちます。また、カスタムデジタル信号処理ルーチンのベースとして CMSIS-DSP を使用することもできます。
- CMSIS-NN ： Arm Cortex-M プロセッサ上のニューラルネットワークのパフォーマンスを最大化し、メモリフットプリントを最小化するために開発された効率的なニューラルネットワークカーネルのコレクションです。CMSIS-NN が提供する一連のニューラルネットワーク操作を使用するか、独自の専用モデルをデプロイすることができます。

  CMSIS-NN では、クラウドではなくエッジで推論を実行できます。エッジコンピューティングにより、プライバシーとセキュリティが向上し、レイテンシと帯域幅が削減されます。
- CMSIS-Stream ： Python パッケージと、サンプルのストリームを処理するために組み込みデバイスで使用する一連の C++ ヘッダです。CMSIS-Stream は、少ないメモリの使用、最小オーバーヘッド、予測可能なスケジューリング、およびモジュール設計を提供します。また、グラフィカル表示も提供します。
- CMSIS-Toolbox ： Open-CMSIS-Pack フォーマットの Software Pack を操作するためのコマンドラインツールです。このフォーマットは、Keil Studio Cloud と Visual Studio Code の Keil Studio 拡張で使用される csolution プロジェクト形式の基本となります。
- CMSIS-Zone ： Arm Cortex-M を使用する組み込みアプリケーションでのパーティショニング、メモリ管理およびアクセス権の制御の簡略化に役立ちます。
- CMSIS-DAP ： Debug Access Port (DAP) へのアクセスを提供し、ホストコンピュータ上のマイクロプロセッサとデバッグツール間の USB 経由の通信を有効にします。

4.1 CMSIS の基本概念

この章では、CMSIS の使用を開始する前に知っておくと便利な概念をいくつかまとめており、より詳細な情報へのリンクを提供します。

4.1.1 CMSIS-Pack

CMSIS-Pack は、Arm Cortex ベースのマイクロコントローラ用ソフトウェアコンポーネントを提供するための標準化されたパッケージングフォーマットです。これは、プロジェクトへのコンポーネントの統合を簡略化し、バージョニングのサポートおよびさまざまなデバイス、ツールチェインや開発環境での互換性を確保します。

CMSIS-Pack には、デバイス固有の情報、共通の機能を提供するミドルウェアコンポーネント、特定のユースケースのコードライブラリなど、アプリケーションレベルのコンポーネントを含めることができます。

CMSIS-Pack は、特定のタイプの Software Pack の一つです。

4.1.2 Software Pack

特定のハードウェアプラットフォームまたは特定の目的に合わせて調整された、すぐに活用できるコンポーネントとツールのセットです。このセットは、パックに含まれるコンテンツを記述する **Pack Description(PDSC)** ファイルとともにバンドルされ、バージョンの履歴や依存関係に関する情報を提供します。

MDK は、Software Pack によるプロダクトライフサイクル管理を容易にするツールを提供します。さらに、CMSIS-Toolbox を使用して、Open-CMSIS-Pack フォーマットの Software Pack を操作できます。このフォーマットは、Keil Studio Cloud と Visual Studio Code の Keil Studio 拡張で使用される csolution プロジェクト形式の基本となります。

Software Pack は、組み込み開発用の汎用リソースを提供するように設計されています。また、さらに特殊な **Device Family Packs (DSP)** および **Board Support Packs (BSP)** と呼ばれる Software Pack の種類もあります。これらは、特定のマイクロコントローラファミリまたはハードウェアボードのサポートを提供するために より多くの調整が行われています。

基本的な Software Pack、DSP、および BSP の操作方法については、Open-CMSIS-Pack のドキュメントの Pack Tutorials の章を参照してください。

4.1.3 Software コンポーネント

組み込みシステム開発では、Software コンポーネントはより広範のシステム内で特定の機能を果たすソフトウェアのモジュールであり、再利用可能な部分となります。複数のコンポーネントをまとめて、Software Pack または CMSIS-Pack にまとめることができます。詳細については、本書の追加の Software コンポーネントの概要 の章を参照してください。

4.1.4 CMSIS solutions

CMSIS solutions(csolution とも呼ばれます) は、より大きなアプリケーションの一部であり、個別にビルドできる関連プロジェクトのグループです。`*.csolution.yaml` ファイルを編集することで solution を定義できます。

CMSIS-Toolbox は、`*.csolution.yaml` および`*.cproject.yaml` をアプリケーションビルドプロセス中にユーザ入力として受け取ります。

より詳細な情報およびサンプルプロジェクトについては、Keil Studio Cloud User Guide 内 CMSIS-Toolbox documentation および Work with CMSIS solutions の章を参照してください。

Keil Studio Visual Studio Code 拡張パック、Arm Keil Studio Pack の CMSIS solutions 拡張も、solution の作業をサポートします。詳細については、Arm Keil Studio Visual Studio Code Extensions User Guide を参照してください。

uVision を使用している場合、`csolution` プロジェクトを CPRJ ファイルフォーマットに変換するために CMSIS-Toolbox 内の csolution build tool を利用できます。

4.1.5 CMSIS プロジェクト

CMSIS プロジェクトは`*.csolution.yaml` および`*.cproject.yaml` をアプリケーションビルドプロセス中にユーザ入力として受け取ります。

より詳細な情報およびサンプルプロジェクトについては、Keil Studio Cloud User Guide 内 CMSIS-Toolbox documentation および Work with CMSIS solutions の章を参照してください。

Keil Studio Visual Studio Code 拡張パック、Arm Keil Studio Pack の CMSIS solutions 拡張も、CMSIS solution の作業をサポートします。詳細については、Arm Keil Studio Visual Studio Code Extensions User Guide を参照してください。

uVision を使用している場合、`csolution` プロジェクトを CPRJ ファイルフォーマットに変換するために CMSIS-Toolbox 内の csolution build tool を利用できます。

4.2 CMSIS software コンポーネントの概要

開発者の目にふれる重要な CMSIS コンポーネント:

- CMSIS-Core : プロセッサコアおよびデバイスペリフェラルへのアクセスを標準化し、異なる Cortex-M コントローラ間で実行されるコードの記述を容易にします。
- CMSIS-RTOS2 : Arm Cortex プロセッサをベースとするデバイス用の汎用リアルタイムオペレーティングシステムインターフェイス。CMSIS-RTOS2 は、複数のタスクとリソースを管理および調整するプロセスを簡略化します。また、異なる RTOS カーネル間の移行プロセスにも役立ちます。
- CMSIS-Driver : ペリフェラルとデバイスを設定および制御するための標準化された API を提供します。CMSIS-Driver はプラットフォームに依存しないように設計されており、サポート対象の様々なマイコンデバイス間でのコードの再利用を容易にします。
- CMSIS-Compiler : 標準 C ランタイムライブラリで I/O 動作をリターゲットするためのソフトウェアコンポーネントと、例外や割り込み処理などのコア関数の標準 API を提供します。
- CMSIS-View : 組み込みアプリケーションの開発およびデバッグ中に、マイクロコントローラ、ペリフェラル、ハードウェアコンポーネント、およびソフトウェアコンポーネントの内部動作を可視化します。
- CMSIS-DSP : 広範にわたるデジタル信号処理関数およびルーチンを提供します。CMSIS-DSP アルゴリズムは、効率のために最適化されており、アプリケーションのパフォーマンスを最大化し、リソース使用量を最小化するのに役立ちます。また、カスタムデジタル信号処理ルーチンのベースとして CMSIS-DSP を使用することもできます。
- CMSIS-NN : Arm Cortex-M プロセッサ上のニューラルネットワークのパフォーマンスを最大化し、メモリフットプリントを最小化するために開発された効率的なニューラルネットワークカーネルのコレクションです。

- CMSIS-Stream ： Python パッケージと、サンプルのストリームを処理するために組み込みデバイスで使用する一連の C++ ヘッダです。CMSIS-Stream は、少ないメモリの使用、最小オーバーヘッド、予測可能なスケジューリング、およびモジュール設計を提供します。また、グラフィカル表示も提供します。
- CMSIS-Toolbox ： Open-CMSIS-Pack フォーマットの Software Pack を操作するためのコマンドラインツールです。このフォーマットは、Keil Studio Cloud と Visual Studio Code の Keil Studio 拡張で使用される csolution プロジェクト形式の基本となります。
- CMSIS-Zone ： Arm Cortex-M を使用する組み込みアプリケーションでのパーティショニング、メモリ管理およびアクセス権の制御の簡略化に役立ちます。
- CMSIS-DAP ： CMSIS-DAP は、CoreSight デバッグおよびトレース機能の一部として多くの Arm Cortex プロセッサで利用可能な CoreSight Debug Access Port (DAP)に対する標準化されたアクセスを提供するプロトコル仕様およびオープンソースファームウェア実装です。

4.3 CMSIS base software コンポーネントの概要

CMSIS base software コンポーネントは、マイコンデバイスの基本的な機能のソフトウェア抽象化を提供します。これらのコンポーネントは、Arm::CMSIS Software Pack によって提供されます。

4.3.1 CMSIS-Core

CMSIS-Core(Cortex-M) は、Cortex-M デバイスの基本的なランタイムシステムを実装し、プロセッサコアおよびデバイスのペリフェラルにアクセスできます。以下の機能を定義しています：

- Cortex-M プロセッサレジスタ用のハードウェア抽象化レイヤ(HAL：hardware abstraction layer)
- システム例外とインターフェイスをとるときに使用する標準のシステム例外名。これにより、異なるプラットフォーム間の互換性を確保しやすくなります。
- ヘッダファイルの整理に使用するメソッド、およびデバイス固有の割り込みの命名の規則。このように標準化をおこなうことで、新しい Cortex-M マイクロコントローラ製品の学習が容易になり、ソフトウェアの移植性が向上します。
- 各マイクロコントローラのベンダが使用するシステム初期化の方法。たとえば、標準化された `SystemInit()` 関数は、デバイスのクロックシステムを設定するために不可欠です。
- 標準 C 関数では使用できない特定の CPU の命令を生成するために使用する組み込み関数
- SysTick タイマの設定を簡略化できる、システムクロック周波数を決定するための標準化された変数

組み込みアプリケーションでの CMSIS-Core の使用方法はこちらで学習できます。

4.3.2 CMSIS-RTOS2

CMSIS-RTOS2 は、Arm Cortex マイクロプロセッサに基づくデバイス用の汎用 RTOS インターフェイスを提供し、RTOS 機能を必要とするソフトウェアコンポーネント用の標準化された API を提供します。標準化された API を使用すると、どの RTOS を設計プロセスの後の段階に使用するかが決まり、柔軟性が向上します。詳細については、CMSIS-RTOS2 のドキュメンテーション を参照してください。

CMSIS-RTOS2 は、Arm Cortex マイクロプロセッサをベースとするデバイス用の汎用 RTOS インターフェイスを提供し、RTOS 機能を必要とするソフトウェアコンポーネント用の標準化された API を提供します。標準

化された API を使用すると、どの RTOS を設計プロセスの後の段階に使用するかが決まり、柔軟性が向上します。詳細については、CMSIS-RTOS2 のマニュアル を参照してください。

CMSIS-RTOS2 は、多くのアプリケーションで必要とされる一連の基本機能を提供します。これにより、学習のための労力が削減されるとともにソフトウェアコンポーネントの共有が簡略化されます。CMSIS-RTOS2 を使用するミドルウェアコンポーネントは RTOS に依存しないため簡単に適用できます。

CMSIS では、オープンソースの CMSIS-RTOS2 実装に含めることで、さらなる開発の出発点を提供する CMSIS-RTOS2 のプロジェクトテンプレートも提供しています。

RTOS 設計の利点

組み込みアプリケーションのための 2 つの基本的な設計コンセプトがあります。**無限ループ**設計(単純なアプリケーションに適しています)と **RTOS** 設計です。RTOS ベースでの設計には様々な利点があります:

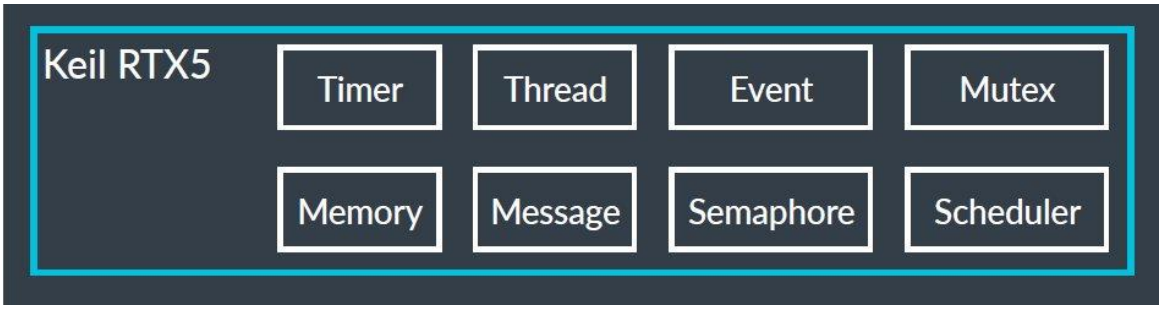
- RTOS はスレッド優先度とランタイムスケジューリングを確実に処理します。
- RTOS はスレッド間の通信用に、明確に定義されたインターフェイスを提供します。
- プリエンプティブな RTOS では優先順位の高いスレッドがタイムクリティカルなデータ処理を実行できるため、割り込み機能の複雑さを軽減します。
- プリエンプティブなマルチタスク処理はより大きな開発チームにまたがるアプリケーションの継続的な改善をより単純にします。これは、より重要なスレッドの応答時間をリスクにすることなく新しい機能を追加できるためです。
- 無限ループをベースとしたアプリケーションでは、多くの場合発生した割り込みをポーリングします。一方、RTOS カーネルはそれ自体が割り込み駆動型であるためポーリングを大幅に排除できます。そのため CPU はスリープ状態になるか、スレッドを処理するかをより頻繁に判断し、切り替えることができます。
- 実世界ではアプリケーションは複数の異なるタスクを処理する必要があります。RTOS ベースのアプリケーションはソフトウェア上でこのモデルを再現します。

最新の RTOS カーネルは、割り込みシステムと密接に連携するように設計されています。これは、割り込み(ボタン、センサ、タイマおよびペリフェラルなどのハードウェアコンポーネントからの信号) に効率的かつ迅速に応答する必要があるシステムや、リアルタイム要件を備えた組み込みシステムの場合に不可欠です。

Keil RTX5

Keil RTX version 5(RTX5) は、ARM Cortex-M および Cortex-A プロセッサをベースとするデバイス用のリアルタイムオペレーティングシステム(RTOS) であり、CMSIS-RTOS2 API をネイティブインターフェイスとして実装しています。

図 4-2: RTX5 の概要図



より詳細については Theory of Operation および Tutorial のページを参照してください。

4.3.3 CMSIS-Driver

CMSIS-Driver API は、ミドルウェアスタックとユーザアプリケーションのためのペリフェラルドライバインターフェイスを定義しています。API は特定の RTOS に依存することなく汎用的に設計されており、サポート対象の様々なマイコンデバイス全体で再利用できるようになっています。サポート対象のペリフェラルの種類に対し様々なユースケースを幅広くカバーしていますが、すべての潜在的なユースケースまでは考慮対象とはなっていません。詳細については、CMSIS-Driver のドキュメントを参照してください。

CMSIS Software Pack はヘッダファイルとドキュメントとともに、**CMSIS-Driver** コンポーネントクラスのもとで API を公開しています。これらのヘッダファイルは、標準化されたペリフェラルドライバインターフェイスの実装に対するリファレンスとなります。

これらの実装は、通常は **CMSIS-Driver** コンポーネントクラスのもとで、関連マイコンのファミリやシリーズに対する Device Family Pack (DFP) 内で公開されています。DFP には、仕様でカバーされる標準のペリフェラルドライバのセットに加えて、メモリバス、汎用入出力(GPIO)、ダイレクトメモリアクセス(DMA) などの **Device** コンポーネントクラスに加え、さらにデバイス固有のインターフェイスが含まれる場合があります。

標準のペリフェラルドライバインターフェイスは、マイクロコントローラのペリフェラルを、通信スタック、ファイルシステム、またはグラフィカルユーザインターフェイスを実装するミドルウェアに接続します。各インターフェイスは、デバイス内の同じタイプの物理インターフェイスを表す複数のインスタンスを提供します。
たとえば、2 つの物理シリアルペリフェラルインターフェイス(SPI)には、ドライバをミドルウェアまたはユーザアプリケーションに接続するために使われる個別のアクセス構造体があります。

詳細については、Theory of Operation のページを参照してください。

4.4 CMSIS 拡張ソフトウェアコンポーネントの概要

CMSIS 拡張ソフトウェアコンポーネントは、Arm プロセッサ上で実行するために最適化された特定の機能を実装しています。各コンポーネントは個別の CMSIS-Pack で提供されます。

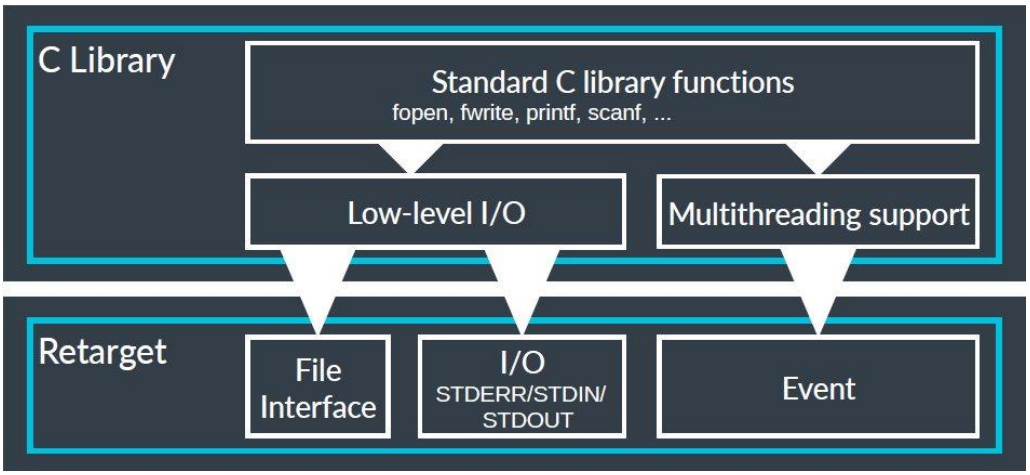
4.4.1 CMSIS-Compiler

CMSIS-Compiler は、標準 C ランタイムライブラリの入出力(I/O)操作をマイクロコントローラまたは開発ボードの特定の I/O インターフェイスで動作するよう適応させるのに役立つソフトウェアコンポーネントを提供します。(リターゲットと呼ばれる操作)

CMSIS-Compiler は、リターゲット用に次のインターフェイスをサポートしています。

- ファイルの読み書きのためのファイルインターフェイス
- 標準 I/O ストリームのリターゲットのための I/O インターフェイス(stderr, stdin, stdout)
- 任意の RTOS を使用したマルチスレッドセーフティのための OS インターフェイス

図 4-3: CMSIS-Compiler の概要図



標準 C ライブラリ関数はプラットフォームに依存しませんが、マルチスレッドサポートと低レベル I/O の実装はターゲットとなるコンパイラツールチェーンに合わせて調整されます。

CMSIS-Compiler のドキュメントを参照し、template を利用して使用を開始してください。

4.4.2 CMSIS-View

CMSIS-View は Arm Cortex-M プロセッサを実装したデバイス上の組み込みソフトウェアプログラムの動作を分析するのに役立つ方法、ソフトウェアコンポーネントおよびユーティリティを提供します。

図 4-4: CMSIS-View の概要図

CMSIS-View を使用すると、メモリとタイミングのオーバーヘッドを最小限に抑えながら、組み込みシステムの動作状況を確認できます。event statistics 機能を使用すると、コードの実行に関する統計データを収集し、分析できます。

CMSIS-View は、シンプルなデバッグアダプタのみを使用することですべての Cortex-M デバイスで動作します。コンパイラに依存しない実装により、アプリケーションプロジェクトとの統合が簡単になります。CMSIS-View では、CMSIS-RTX および CMSIS-FreeRTOS の RTOS-aware なデバッグや、Arm Virtual Hardware Fixed Virtual Platform(FVP) モデル(セミホスティング経由) のリグレッションテストで使用できるログ機能も有効になります。

CMSIS-View のドキュメントを参照し、利用可能なサンプルプロジェクトを確認してください。

4.4.3 CMSIS-DSP

CMSIS-DSP は、Arm Cortex-M および Cortex-A プロセッサでの使用に最適化された一般的なデジタル信号処理(DSP) 機能を実装するオープンソースソフトウェア ライブラリです。

CMSIS-DSP はさまざまなコンピューティングのカテゴリをカバーし、複数のデータ型を持つカーネルを提供します。Python ラッパーも利用可能で、C の API に可能な限り近い API を使用して Python でアルゴリズムを設計するのに役立ちます

CMSIS-DSP のドキュメントを参照し、learning path を利用して使用を開始してください。

4.4.4 CMSIS-NN

CMSIS-NN オープンソースソフトウェアライブラリは、効率的なニューラルネットワークカーネルのコレクションを通じて、Arm Cortex-M プロセッサ上で実行されるニューラルネットワークのパフォーマンスを最大化し、メモリの使用量を最小限に抑えます。CMSIS-NN が提供するニューラルネットワーク操作のセットを使用することも、独自に特化したモデルをデプロイすることもできます。

CMSIS-NN を使用すると、クラウドではなくエッジで推論を実行できます。エッジコンピューティングにより、プライバシーとセキュリティが向上し、レイテンシとバンド幅が削減されます。

CMSIS-NN 関数にはいくつかのバリエーションがあります。CMSIS-NN は、ターゲットプロセッサアーキテクチャの機能に応じて、コンパイル時に最適なソリューションを自動的に選択します。

利用可能な機能の詳細については、CMSIS-NN のドキュメントを参照してください。または、サンプルから利用を開始してください。

4.5 CMSIS ツールの概要

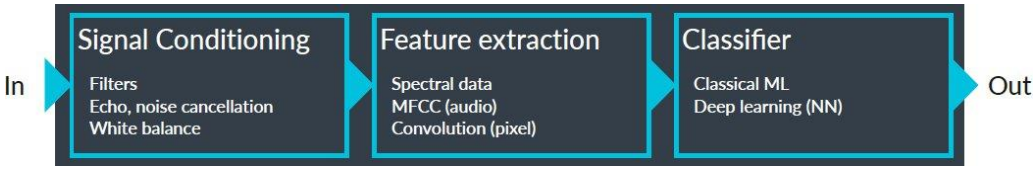
MDK には、CMSIS ベースのコンポーネントを操作するための便利なツールも含まれています。

4.5.1 CMSIS-Stream

CMSIS-Stream は、DSP/ML アプリケーションの処理ステップ間のデータブロックストリーミングを最適化する Python パッケージです。モジュール設計が可能になり、DSP パイプラインの開発と実装が容易になります。ツールはビルド時に処理ノードのスケジュールを最適化し、メモリ使用量を削減します。このプロセスにより、compute graph の形式で設計のわかりやすいイメージが生成されます。

compute graph は、アプリケーション内の処理ノードまたはコンポーネント間のデータ フローの構造と順序を示す有向グラフです。データ形式、First In First Out(FIFO) バッファ、データストリーム、および処理ステップを記述するために Python スクリプトファイルを使用します。CMSIS-Stream ツールは、ビルド時に compute graph を最適化された処理ステップに変換します。

図 4-5：CMSIS-Stream の概要図



CMSIS-Stream は、ML ソフトウェアスタックを最適化するために必要となる、最適化された DSP パイプラインを作成するためのツールを提供します。computing graph が提供する視覚的な表現は、複数の相互接続されたコンポーネントを含む複雑な DSP または ML ワークフローに対して有効です。

信号調整と特徴抽出を最適化することにより、ML 分類器(Classifier) の複雑さを軽減します。
DSP プリプロセスを増やすと、ML アプリケーションに必要とされる全体的なパフォーマンスが低下します。

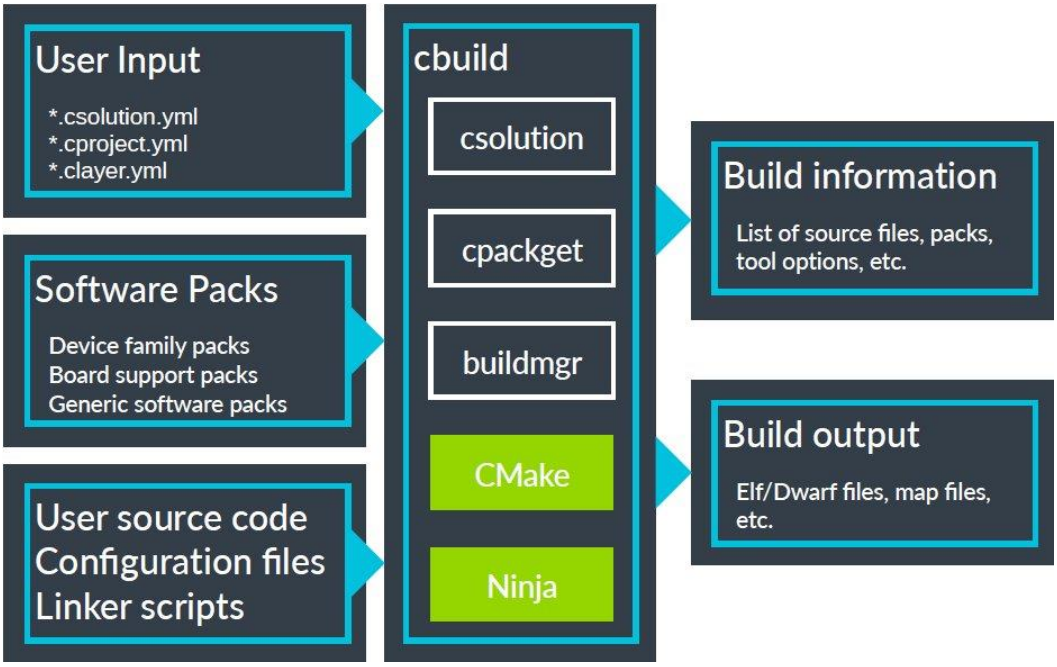
CMSIS-Stream は、非対称マルチプロセッシング(AMP) システムでも機能するデータ管理用のインターフェイス、ヘッダファイル、テンプレート、メソッド、および使い始める際に役立つサンプルも提供します。

CMSIS-Stream のドキュメントを参照し、サンプルを確認してください。

4.5.2 CMSIS-Toolbox

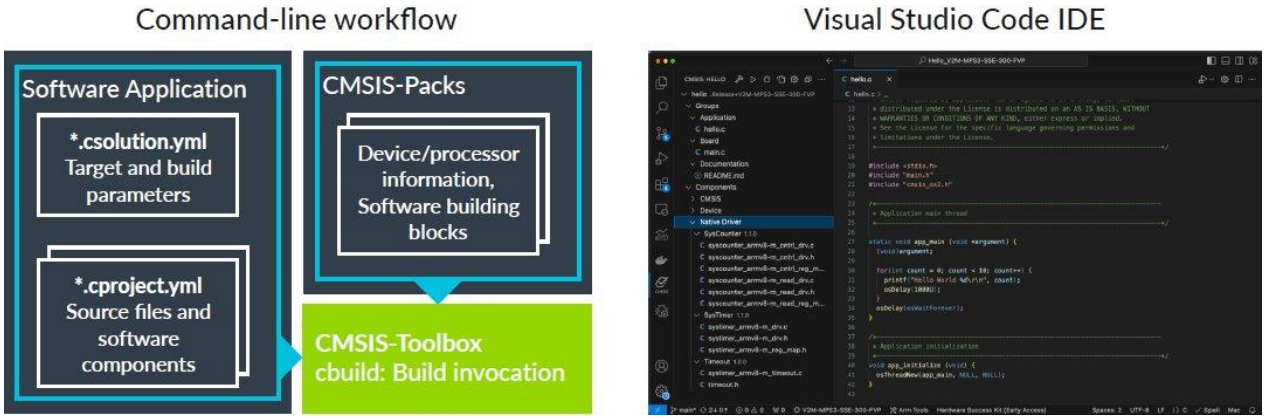
CMSIS-Toolbox は、CMSIS-Packs に基づく組み込みアプリケーションの作成とビルドのためのコマンドラインツールを提供します。CMSIS-Toolbox は、Arm Compiler for Embedded, GCC, IAR や LLVM などの複数の処理系をサポートしています。これらのツールは、ソフトウェアコンポーネントやソフトウェアおよびハードウェアサポートを含む CMSIS-Packs の作成、保守、配布にも役立ちます。

図 4-6：CMSIS-Toolbox の概要図



コマンドラインツールはスタンドアロンでの使用や、Visual Studio Code の拡張機能、あるいは Continuous Integration (CI) 用の DevOps システムの拡張機能に統合して使用することもできます。ツールはすべてのホストプラットフォーム(Windows、Mac、Linux) で使用でき、柔軟性のある形でデプロイできます。
cbuild、csolution および cpackget のコマンドラインからの使い方や、シンタックス、使用例などの詳細については、Build Tools のドキュメントを参照してください。

図 4-7: CLI と IDE のワークフロー



Software pack を使用することでデバイスやボードを選択したり、再利用可能なソフトウェアコンポーネントにアクセスできるプロジェクトを作成したりできるようになり、ツールのセットアップが簡単になります。

solution を独立管理するプロジェクトに取り込む機能により、マルチプロセッサアプリケーションやユニットテストなど、多くのユースケースが簡単になります。

CMSIS-Toolbox は、コンフィギュレーションファイルの管理と更新が簡単に行えるバージョン管理された Software pack を備えており、product lifecycle management(PLM)にも対応しています。

ソフトウェアレイヤによって、事前にコンフィギュレーション済みのソースファイルとソフトウェアコンポーネントのセットを使用して、類似のアプリケーション間でのコードの再利用が可能になります。

CMSIS-Toolbox は以下をサポートします:

- アプリケーションを異なるハードウェア(テストボード、製品ハードウェア、仮想ハードウェア等) にデプロイできる、Multiple hardware targets
- ソフトウェアのテストと検証(デバッグビルド、テストビルド、リリースビルド等) をサポートする、Multiple build types
- 検証中に異なるツールチェインを選択できる、Multiple toolchains (同じユーザ入力ファイルのセット内であっても) とコマンドラインオプション

CMSIS-Toolbox は、ビルドプロセスに CMake バックエンドを使用します。CMake を CMSIS-Toolbox とともに使用すると、solution 用の `compile_commands.json` ファイルの生成が簡略化されます。これらの JSON ファイルには、プロジェクトファイルのリストとビルドプロセスで使用されるコンパイラのコマンドが含まれており、IntelliSense を強化するためのさまざまな Visual Studio Code 拡張で使用できます。

より詳細については、CMSIS-Toolbox のドキュメントを参照してください。

4.5.3 CMSIS-Zone

CMSIS-Zone は、Arm Cortex-M プロセッサを使用した組み込みアプリケーションにおけるパーティション分割、メモリ管理、およびアクセスパーミッション指定の複雑さを軽減するのに役立ちます。

CMSIS-Zone を使用すると、セキュアモードと非セキュアモードの両方で、メモリ領域に対するアクセスとセキュリティ権限を指定できます。その後、XML ベースのゾーンファイルを使用してアプリケーションでメモリ管理とパーティション生成用のヘッダファイルを生成できます。

詳細については、CMSIS-Zone のドキュメントを参照してください。

4.5.4 CMSIS-DAP

CMSIS-DAP は、CoreSight オンチップデバッグおよびトレース機能の一部として、多くの Arm Cortex プロセッサで利用可能な CoreSight デバッグアクセスポート(DAP) に対する標準化されたアクセスを組み込みソフトウェア開発者に提供します。

CMSIS-DAP は、組み込みアプリケーションが実行されるマイクロプロセッサと、ホスト コンピュータで実行されるデバッグツール間の標準化された通信を可能にします。CMSIS-DAP では、デバイスのデバッグポートを USB ポートに接続するデバッグユニットのインターフェイスファームウェアを提供します。

これにより、ホストコンピュータ上で実行されるソフトウェアデバッグツールを USB やデバッグユニットを使用して接続できるようになります。

より詳細については、CMSIS-DAP のドキュメントを参照してください。

5. その他の Software コンポーネントと Pack

組み込みシステム用のソフトウェアを設計および実装するには、複数のコンポーネントを使用するモジュールアーキテクチャが必要です。Software Pack は、特定の目的のためにまとめられたコンポーネント(ミドルウェア、ソースコード、ライブラリ、サンプルプロジェクトなど) の集まりです。

Pack は、特定のマイクロコントローラファミリまたは開発プラットフォーム向けにすぐに使用できるコンポーネントを提供するために使われます。これにより、特定の組み込みシステム向けに開発環境の設定とコード作成のプロセスが簡略化されます。

CMSIS-Pack は、特定のタイプのソフトウェアパックを示します。Arm Cortex-M プロセッサのコア機能にアクセスしてコンフィギュレーションするための一貫したインターフェイスを定義する CMSIS Standard に準拠しており、簡単にプロジェクトに対してソフトウェアコンポーネントを統合し、保守できます。

CMSIS-Pack には、ソフトウェアコンポーネントに属するファイルに関するメタデータが含まれており、コンポーネントのオリジナルのベンダからの情報が保持されます。メタデータには、ツールチェイン、デバイス、プロセッサの依存関係情報が含まれているので、アプリケーションへの統合が簡単になります。

CMSIS-Pack システムのもう 1 つの利点は、一貫性のあるソフトウェアコンポーネントのアップグレードプロセスが可能になり、ユーザアプリケーションの一部に含まれている互換性がない可能性のあるコンフィギュレーションファイルを識別できることです。さらに、ソフトウェアコンポーネントプロバイダはインターフェイスと、他のソフトウェアコンポーネントとの関係を定義できます。

Arm は公開されている CMSIS-Pack のリストを管理しています。

詳細については、CMSIS-Pack のドキュメントを参照してください。

5.1 Software Pack とプロダクトライフサイクルの管理

MDK を使用すると、Software Pack の複数のバージョンをインストールできます。これにより、多くのプロジェクトで一般的な製品ライフサイクル管理(product lifecycle management:PLM) が可能になります。

図 5-1: PLM のステージを示す図

PLM のフェーズは 4 つあります。

- Concept:主要なプロジェクト要件の定義と機能プロトタイプを伴う調査
- Design:最終的な技術的特徴と要件に基づいた製品のプロトタイプテストと実装

- Release：製品の製造と市場投入
- Service：顧客サポートを含む製品のメンテナンス。最終の段階的な廃止または end-of-life

Concept と Design フェーズでは、通常、最新の Software Pack を使用して、新機能やバグ修正を迅速に取り入れます。製品リリース前にコンポーネントを既知のテスト済み状態にするためにソフトウェアをフリーズします。します。Service フェーズでは、現場で顧客をサポートするため固定されたバージョンのソフトウェアコンポーネントを使用します。

プロジェクトで CMSIS-Pack を厳密なセマンティックバージョニングすることでインストールされているソフトウェアパックのバージョンの管理が容易になります。

5.2 追加の Software コンポーネントの概要

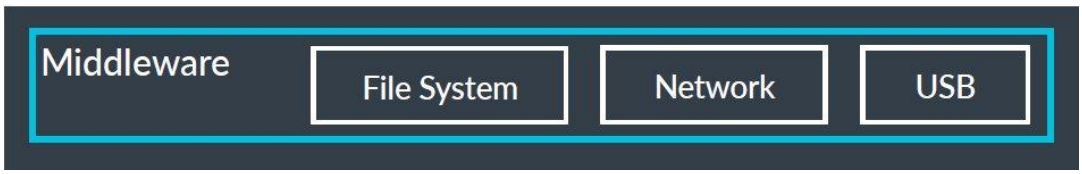
次の表は、組み込みアプリケーションで頻繁に使用されるソフトウェアコンポーネントの一覧です。

MDK-Middleware Software Pack のコンポーネントを含みます。

表 5-1：よく使われるソフトウェアコンポーネント

ソフトウェアコンポーネント	説明
CMSIS-FreeRTOS	CMSIS-RTOS2 対応 FreeRTOS カーネル
CMSIS-mbedTLS	CMSIS-Pack で提供される MbedTLS のコピー
Synchronous Data Stream (SDS)	データストリーム管理フレームワーク
Network	Ethernet またはシリアルプロトコルを使用した TCP/IP ネットワーク用 MDK ミドルウェアコンポーネント
File System	各種ストレージタイプに対するファイルアクセス用 MDK ミドルウェアコンポーネント
USB	USB ホストとデバイス間の通信のための MDK ミドルウェアコンポーネント。標準 USB device class をサポート
IoT Clients	各種クラウドサービスプロバイダ向けオープンソースクライアント
Open-source components	アプリケーションの機能拡張に使用できるオープンソースコンポーネント

次の図は、MDK-Middleware Software Pack のコンポーネントを示しています。
追加の Software Pack をインストールすると、さらに多くのソフトウェアコンポーネントが利用可能になります。



5.2.1 CMSIS-FreeRTOS

FreeRTOS は、組み込みマイクロコントローラ向けの市場をリードするリアルタイムオペレーティングシステム(RTOS) です。専門的に開発され、厳格な品質管理が行われ、堅牢で、完全にサポートが行われ、ドキュメントも存在しています。無料です。独自のソースコードを公開する必要がなく、商用製品開発に自由に使用でき、知的財産権侵害のリスクもありません。

Arm は、リアルタイムオペレーティングシステム(RTOS) 用の CMSIS‑RTOS2 API をサポートする FreeRTOS の実装を作成しました。この Software-Pack を使用すると、ネイティブ FreeRTOS 実装か、CMSIS‑RTOS2 API に準拠し FreeRTOS を内部で使用する実装のいずれを使用するかを選択できます。CMSIS‑RTOS2 API を使用すると、プログラマはさまざまな RTOS カーネル(Keil RTX5 など) で使用できるポータブルなアプリケーションコードを作成できます。

CMSIS‑FreeRTOS のドキュメントを参照し、サンプルプロジェクトからスタートすることができます。

5.2.2 CMSIS-mbedTLS

mbed TLS は、暗号化プリミティブ、X.509 certificate 操作、SSL/TLS および DTLS プロトコルを実装する C ライブラリです。コードサイズが小さいため、組み込みシステムに特に適しています。

詳細については、CMSIS-mbedTLS GitHub リポジトリのドキュメントを参照してください。

5.2.3 Synchronous Data Stream (SDS) フレームワーク

Synchronous Data Stream(SDS) フレームワークは、データストリーム管理システムを実装し、デジタル信号処理(Digital Signal Processing：DSP) と機械学習(Machine Learning：ML) アルゴリズムを統合する組み込みアプリケーションを開発および最適化するためのメソッドとツールを提供します。このフレームワークは、CMSIS‑DSP ライブラリの compute graph streaming で使用できます。

SDS は、センサおよびオーディオデータインターフェイス用のフレキシブルなデータストリーム管理を実装しています。タイムドリフトのプロビジョニングを含む複数のインターフェイスからのデータストリームをサポートします。また、分析や開発のためにリアルワールドのデータを記録したり、Arm Virtual Hardware を使用してアルゴリズム検証のためにリアルワールドのデータを再生したりすることもできます。SDS データファイルには、次のような用途があります：

- フィルタ設計者などの DSP 開発ツールに入力を提供する
- ML モデルの分類、トレーニング、パフォーマンスの最適化に入力を提供する
- オフラインツールを使って Cortex‑M ターゲット上で DSP アルゴリズムが動作するか検証する

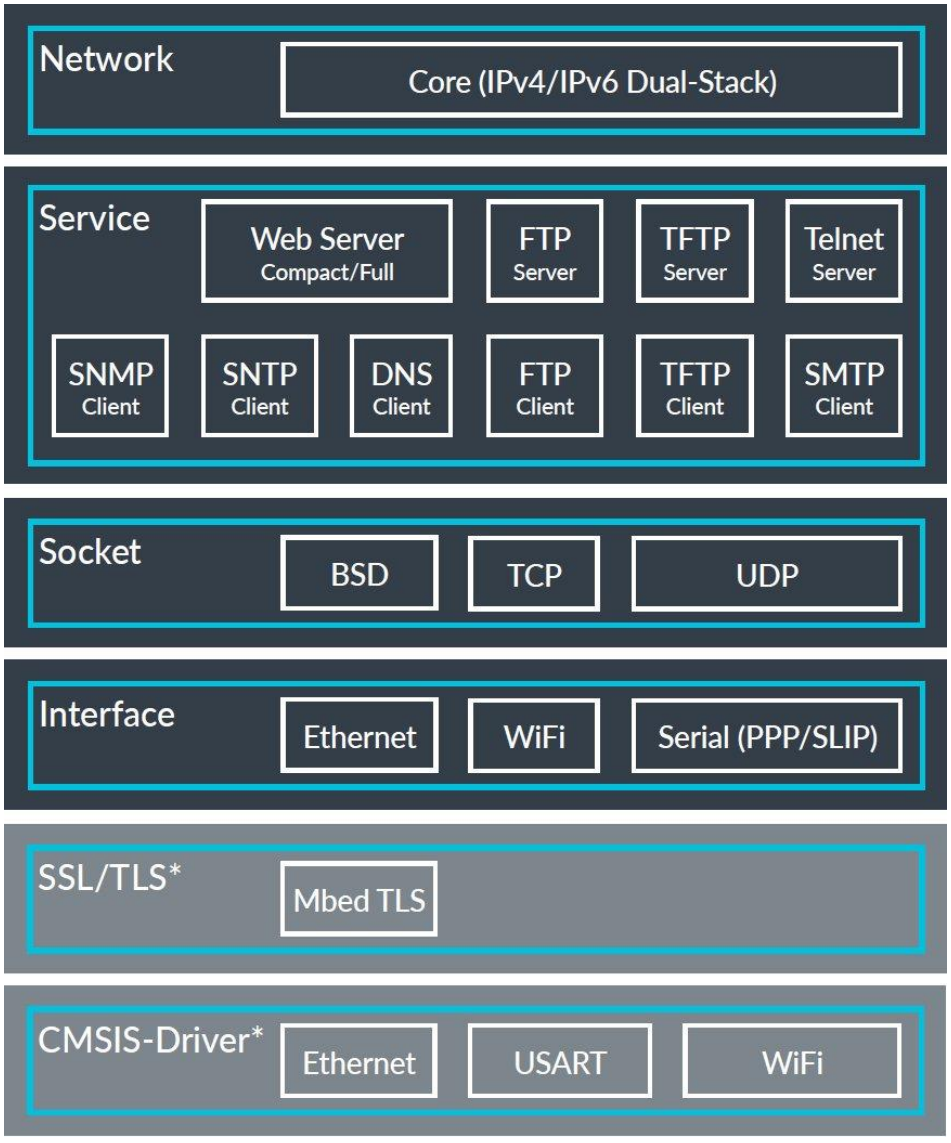
SDS は、YAML ベースのメタデータファイルを使用してバイナリデータ形式を定義します。また、記録、再生、ビジュアライズ、データ変換用の Python ベースのツールも含まれています。

SDS-Framework のドキュメントを参照し、サンプルからスタートすることができます。

5.2.4 Network コンポーネント

MDK-Middleware Pack の Network コンポーネントには、IPv4 および IPv6 ネットワーク アプリケーションを作成するためのサービス、プロトコルソケット、および物理通信インターフェイスが含まれています。

図 5-2: MDK-Middleware Network コンポーネントの概要図



* これらのコンポーネントは Network コンポーネントには含まれません。

Mbed TLS、**Ethernet**、**USART**、および **Wi-Fi** コンポーネントは Network コンポーネントと連携して動作しますが、別々の Pack の一部です。

このサービスは、一般的なネットワークタスク用のプログラムテンプレートを提供します。

すべてのサービスは、通信のネットワークソケットに依存します。Network コンポーネントは、Tenable Security Center(TSC)、User Datagram Protocol(UDP)、および Berkeley Software Distribution(BSD) ソケットをサポートします。

物理インターフェイスは、Ethernet、WiFi、および Serial Line Internet Protocol(SLIP) または Point-to-Point Protocol(PPP) を使用したシリアル接続のいずれかになります。

ドライバは、マイクロコントローラのペリフェラルまたは外部コンポーネントへのインターフェイスを提供します:

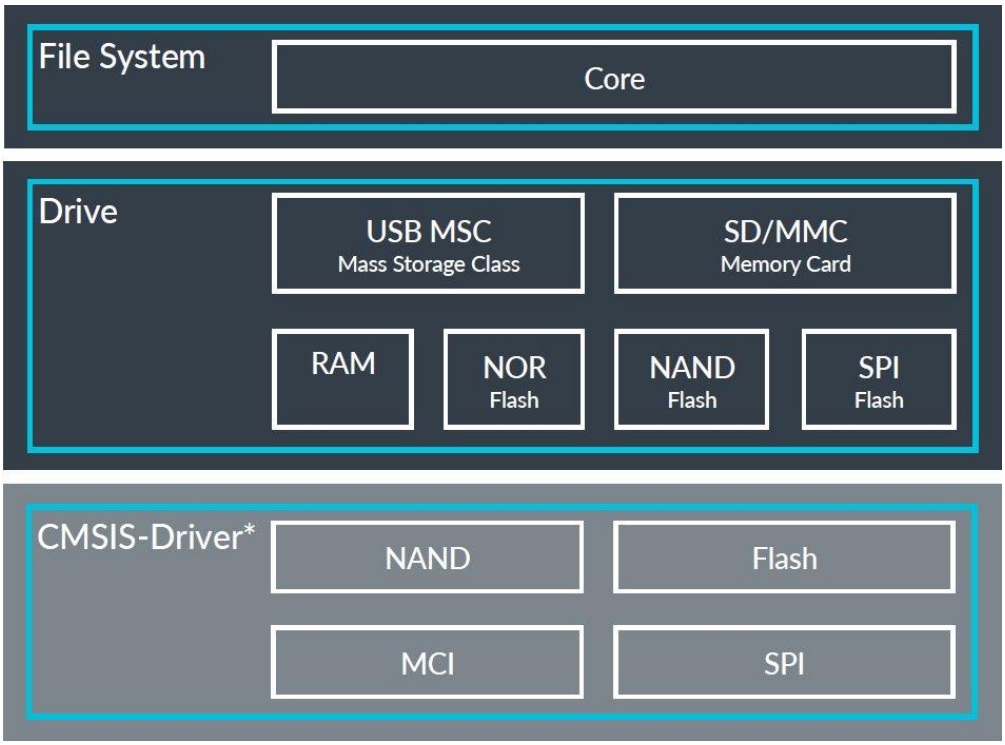
- イーサネットには、Ethernet Media Access Control(MAC) アドレスと Ethernet physical layer(PHY) ドライバが必要です。
- PPP または SLIP インターフェイスは、universal synchronous/asynchronous receiver/transmitter(USART) とモデムを使用します。
- WiFi インターフェイスには WiFi モジュールドライバが必要です。

Network コンポーネントのドキュメントを参照し、サンプルからスタートすることができます。

5.2.5 File System コンポーネント

MDK-Middleware Pack の File System コンポーネントを使用すると、組み込みアプリケーションで、RAM、Flash、メモリカード、USB メモリデバイスなどのストレージデバイス内のファイルを作成、保存、読み取り、および変更ができます。

図 5-3: MDK-Middleware File System コンポーネントの概要図



* これらのコンポーネントは File System コンポーネントには含まれません。

File System コンポーネントは以下のような構造となっています:

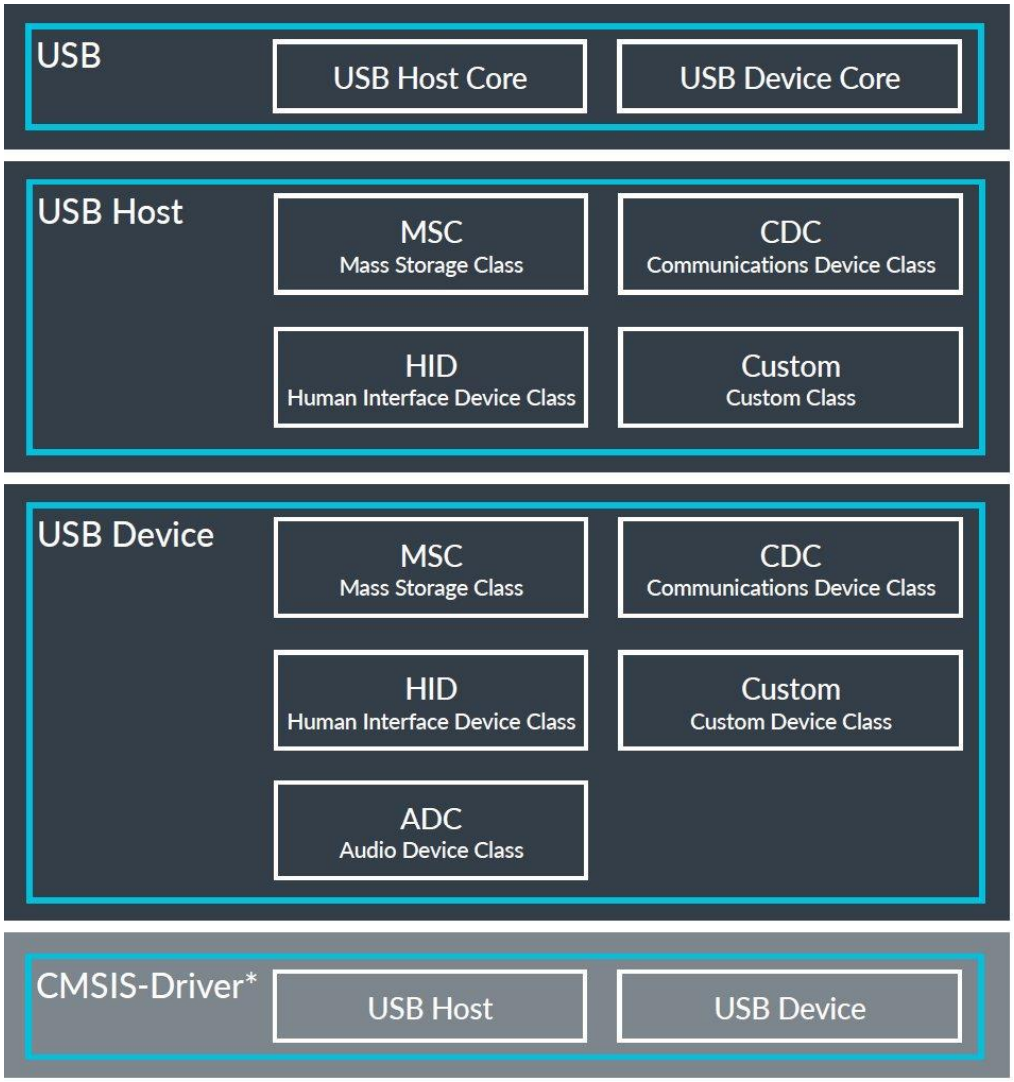
- ストレージデバイスは、アクセスできるドライブとして参照されます。
- 同一ストレージデバイスの複数のインスタンスを実装できます。(たとえば、システムに 2 つの SD カードを接続する場合など)
- File System コアはスレッドセーフな操作をサポートし、NOR および Serial Peripheral Interface(SPI) フラッシュ用の Embedded File System(EFS)、または File Allocation Table (FAT) ファイルシステムを使用します。
  FAT ファイルシステムには 2 つのバリエーションがあります:
  ー 最大 255 文字をサポートする long file name バリアント
  ー 8.3 形式のファイル名のみをサポートする short file name バリアント
- Core では、複数のストレージデバイスに同時にアクセスできます。(たとえば、内蔵 Flash から外部 USB デバイスにデータをバックアップするなど)
- ドライブにアクセスするために、次のストレージデバイスをサポートするドライバが用意されています。
  ー Flash チップ(NAND, NOR, および SPI)
  ー Memory card インターフェイス(SD/SDxC/MMC/eMMC)
  ー USB デバイス
  ー On-chip RAM、Flash および外部メモリインターフェイス

Theory of Operation を参照し、サンプルからスタートすることができます。

5.2.6 USB コンポーネント

MDK-Middleware Pack の USB コンポーネントを使用すると、USB Device および USB Host のアプリケーションを作成できます。USB コンポーネントは USB プロトコルを処理するため、アプリケーションの要求事項に集中できます。

図 5-4: MDK-Middleware USB コンポーネントの概要図



* これらのコンポーネントは USB コンポーネントには含まれません。

USB コンポーネントは以下のような構造となっています:

- USB Host は、USB バス経由での他の USB デバイスペリフェラルとの通信に使用されます。
- USB Device は USB Host に接続できるデバイスペリフェラルを実装します。
- USB Host および USB Device 用の USB API は、マイクロコントローラペリフェラルへのインターフェイスを提供します。

以下の USB Class がサポートされています:

- Human Interface Device (HID)

- Mass Storage Class (MSC)
- Communication Device Class (CDC)
- Audio Device Class (ADC) (USB Device のみ)
- Custom Class (新規または未サポートの USB Class 実装用)
- 複数の Device Class をサポートする Composite USB Devices

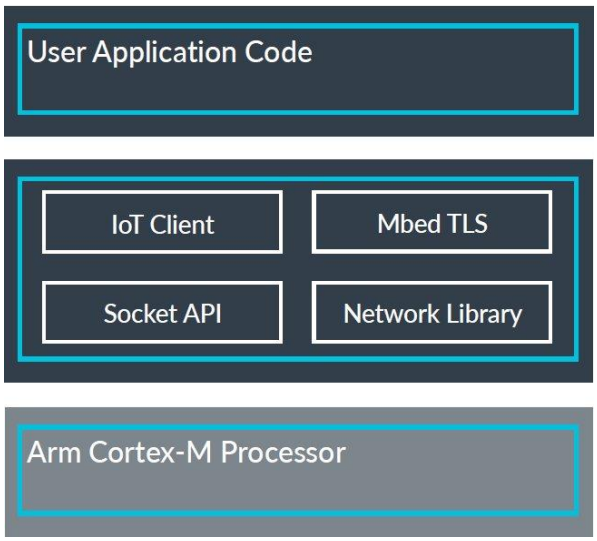
USB コンポーネントのドキュメントを参照し、USB Device のサンプルおよび USB Host のサンプルからスタートすることができます。

5.2.7 IoT クライアント

Internet of Things(IoT) は、データを収集、処理、交換する接続されたエンドノード デバイスを表します。これらのデバイスは、処理能力、データ分析、およびストレージ機能を提供するクラウドサービスにインターネット経由で接続されます。IoT クライアントは、エンドノードデバイス上で実行され、クラウドサービスへの接続を確立するソフトウェアインターフェイスです。

多くのクラウドサービスプロバイダは、組み込みシステム向け IoT クライアントを実装するオープンソースソフトウェアを提供しています。Arm は、クラウドサービスとの通信に信頼性の高い MDK-Middleware Network コンポーネントを使用するようにこれらのクライアントを適応させました。または、CMSIS-WiFi ドライバでサポートされている WiFi デバイスを使用することもできます。

図 5-5: IoT アプリケーションソフトウェアスタック



ほとんどの IoT クライアントは、IoT アプリケーション用の軽量メッセージングプロトコルである Message Queuing Telemetry Transport(MQTT) プロトコルを使用します。TCP ソケット(セキュアでない接続の場合)または TLS ソケット(暗号化による安全な接続の場合) を使用して TCP/IP 経由で通信します。

MDK は、Amazon Web Services に接続するために必要な基本的な構成要素を提供する CMSIS-Pack を提供します。Amazon AWS IoT Pack には、組み込み C を使用してデバイスから AWS IoT に接続するための SDK が用意されています。

Software Pack は汎用(デバイスに依存しない) であり、pack index にリストされています。

5.2.8 オープンソースコンポーネントの概要

MDK v6 では、組み込みアプリケーションの機能を拡張するために使用できるオープンソースコンポーネントが多数あります。このセクションでは、市場で入手可能なオープンソースコンポーネントのいくつかについて説明します。

LGVL

LVGL(Light and Versatile Graphics Library) は、組み込みシステムでの使用に適した、メモリ使用量が少ないグラフィカルユーザインターフェイスを作成できる組み込みグラフィックスライブラリです。LVGL は、あらゆるマイクロコントローラ、マイクロプロセッサ、およびディスプレイタイプで使用できます。

Pack をダウンロードし、詳細については LVGL リポジトリまたはドキュメントを参照してください。

lwIP

lwIP は、TCP/IP プロトコルスイートの軽量実装です。一般的な TCP/IP プロトコルのほとんどを完全にサポートするとともに、リソースの使用量を削減するため、組み込みアプリケーションでの使用に最適です。

Pack をダウンロードし、詳細については lwIP リポジトリまたはドキュメントを参照してください。

6. 新規アプリケーションの作成

MDK で CMSIS を使用してアプリケーションを作成およびビルドする方法について詳しく学習します。

6.1 Keil Studio VS Code 拡張を使用した新しい solution の作成

このセクションでは Keil Studio VS Code extensions を使用して単純な"Hello world" サンプル solution を作成、実行、デバッグするための基本的なワークフローについて説明し、Arm Keil Studio Visual Studio Code Extensions User Guide 内のより詳細な手順へのリンクを提供します。ワークフローには次の手順が含まれます:

- solution の作成
- ソフトウェアツールの管理
- solution にソフトウェアコンポーネントを追加する
- solution にソースコードを追加する
- virtual hardware の設定
- solution のビルド
- solution の実行
- solution のデバッグ

6.1.1 solution の作成

選択したハードウェアに必要なすべての基本ファイルを使用して新しい solution を作成します。詳細については、Arm Keil Studio Visual Studio Code Extensions User Guide の Create a solution を参照してください

この手順では、Arm V2M-MPS3-SSE-300-FVP virtual hardware 用の solution を作成する方法について説明します。他のスターターキットまたはボードの場合は手順を調整してください。

1. Visual Studio Code で Keil Studio Pack をインストールします。
2. Activity Bar で **CMSIS**  をクリックして **CMSIS** ビューを開きます。
3. **Create a New Solution** をクリックします。**Create New Solution** ビューが開きます。
4. **Target Board** ドロップダウンリスト内で V2M-MPS3-SSE-300-FVP virtual hardware を探し、**Select** をクリックします。
5. **Templates and Examples** ドロップダウンリスト内で **Blank solution** を選択します。

6. solution に含めるプロジェクトの名前を入力します。(例:`helloworld`)

7. solution 名を入力します。

8. solution ファイルを保存する場所を指定します。

9. **Create** をクリックします。確認のダイアログボックスが開きます。**Open** をクリックして新しい solution を開き、**Explorer** ビューでファイルを確認します。

10. **Arm Environment Activation** ダイアログが表示されます。Environment Manager extension がワークスペースを自動的にアクティブ化し solution の作成時に生成された `vcpkg-configuration.json` ファイルで指定されたツールをダウンロードできることを確認します。

6.1.2 ソフトウェアツールの管理

新しい solution には、vcpkg を使用して自動的にダウンロードされる一連のソフトウェアツールが含まれます。ツールのダウンロードは、solution の作成時に生成される `vcpkg-configuration.json` ファイルを使用して制御されます。この例では、`vcpkg-configuration.json` ファイル内で使用する virtual hardware モデルを指定する必要があります。

1. `vcpkg-configuration.json` のファイルを開きます。

2. Keil Studio は、この JSON ファイル用のグラフィカルユーザインターフェイスを提供しています。右上隅の **Open Preview to the Side**  をクリックします。

3. **Arm Tools** エディタで、**Arm Virtual Hardware for Cortex-M based on Fast Models** のエントリを見つけて、最新のバージョンを選択します。`vcpkg-configuration.json` ファイルの`"requires":`ブロックの末尾に次のような行が追加されたことに注意してください。

```
"arm:models/arm/avh-fvp": "^11.22.39",
```

ファイルを保存したら、Arm Environment Manager がそれを読み取り、環境をアクティブ化します。VS Code の Status bar にインストールされている Arm ツールの数が表示されます。

`vcpkg-configuration.json` ファイルでは使用するツールを指定できます。完全なバージョン指定を行うと、そのバージョンのみがダウンロードされ、使用されます。`^` 指定子を使用すると、指定したバージョンで始まる任意のバージョンを使用できます。`*` 指定子を使用すると、常に最新バージョンのツールが使用されます。

6.1.3 solution にソフトウェアコンポーネントを追加する

使用する関連するソフトウェアコンポーネントを選択します。詳細については、Manage software components を参照してください。

1. CMSIS ビューで、マウスを Project ヘッダの上に移動し  をクリックします。**Software Components** ビューが開きます。

2. **Software packs: Solution** ドロップダウンリストで、**All packs** を選択します。

3. **Software Components** ビューの見出しの横にある矢印をクリックして、コンポーネントのリストを表示します。次のコンポーネントが選択されていることを確認してください:
   - **CMSIS** > **Core**, **OS Tick** および **RTOS2** > **Keil RTX5** (**Variant** ドロップダウンリストで、**Source** が選択されている)
   - **Device** > **Definition** および **Startup**
   - **Device** > **Native driver** > **SysCounter**, **SysTimer** および **Timeout**

solution に必要な Pack がインストールされていない場合は、インストールするように求められます。同様に、追加するコンポーネントがマシンにインストールされていないコンポーネントへの依存関係がある場合も、追加するように求められます。

6.1.4 solution にソースコードを追加する

`main.h` ヘッダファイルと `helloworld.c` ファイルを solution に追加し、solution 固有のコードをファイルに追加します。

1. **CMSIS** ビューの project outline で、**Groups** > **Source Files** に移動します。

2. **Source Files** 見出しの横にある+ をクリックし、**Add New File** をクリックします。

3. 開いたダイアログボックスで、ファイルに `main.h` という名前を付け、**Save** をクリックします。

4. 次のコードをコピーして `main.h` のファイルに貼り付けます:

```
#ifndef MAIN_H__
#define MAIN_H__

/* Prototypes */
extern void app_initialize (void);

#endif
```

5. **Source Files** 見出しの横にある+ をクリックし、**Add New File** をクリックします。

6. 開いたダイアログボックスで、ファイルに `helloworld.c` という名前を付け、**Save** をクリックします。

7. 次のコードをコピーして `helloworld.c` のファイルに貼り付けます:

```
#include <stdio.h>
#include "main.h"
#include "cmsis_os2.h"
/*----------------------------------------------------------------------------
* Application main thread
*----------------------------------------------------------------------------*/
static void app_main (void *argument) {
  (void)argument;

  for(int count = 0; count < 10; count++) {
    printf("Hello World %d¥r¥n", count);
    osDelay(1000U);
  }
  osDelay(osWaitForever);
}

/*----------------------------------------------------------------------------
* Application initialization
*----------------------------------------------------------------------------*/
void app_initialize (void) {
  osThreadNew(app_main, NULL, NULL);
}
```

8. `main.c` のファイルを開きます。既存のコードを削除し、次のコードに置き換えます:

```
#include "RTE_Components.h"
#include CMSIS_device_header
#include "cmsis_os2.h"
#include "main.h"
int main() {
  osKernelInitialize(); // Initialize CMSIS-RTOS2
  app_initialize(); // Initialize application
  osKernelStart(); // Start thread execution
    for (;;) {
    }
}
```

6.1.5 virtual hardware の設定

この例で使用されている V2M-MPS3-SSE-300-FVP などの virtual hardware 上でプロジェクトをビルドして実行するには、プロジェクトディレクトリにコンフィギュレーションファイルを追加して、さらに `vcpkg-configuration.json` ファイルで使用するモデルを指定する必要があります。

Arm Virtual Hardware simulation models (Fixed Virtual Platform model、または FVP) は現時点では macOS で利用できません。Docker を使用して Linux コンテナ内の macOS で実行できます。こちらの learning path を参照してください。

1. Activity バーで、**Explorer**  をクリックします。

2. project ヘッダで、**New File...**  をクリックします。

3. 新しいファイルの名前として `fvp_config.txt` と入力します。

4. `fvp_config.txt` のファイルを開き、次のコードをコピーして貼り付けます:

```
# Parameters:
# instance.parameter=value #(type, mode) default = 'def value' :
description : [min..max]
#--------------------------------------------------------------------------------
cpu0.semihosting-enable=1 # (bool, init-time) default
= '0': Enable semihosting SVC traps.
mps3_board.visualisation.disable-visualisation=1 # (bool, init-time) default
= '0': Enable/disable visualisation
#--------------------------------------------------------------------------------
```

6.1.6 solution のビルド

solution をビルドするには、  をクリックします。新しい Terminal ビューが開き、ビルド操作が表示されます。

Solution をビルドするためのその他のオプションについては、Arm Keil Studio Visual Studio Code Extensions User Guide 内の Build the example project を参照してください。

6.1.7 solution の実行

AVH FVP でアプリケーションを実行する前に、対応する task を作成する必要があります。次の手順を実行します:

1. **Terminal** > **Configure Tasks…** に移動し `virtual-hardware run: Run Program` を選択します。

2. `tasks.json` のファイル内で、`"problemMatcher": []` の後に新しい行を作成し、次の 2 行を入力します:

```
"config": "${workspaceFolder}/fvp_config.txt",
"args": ["--simlimit", "20"],
```

3. ファイルを保存します。

virtual hardware でアプリケーションを実行するには以下を行います:

1. **Device Manager** ビューに移動し、"Corstone SSE 300 Ethos U55" を選択します。

2. **CMSIS** ビューに移動して  をクリックします。

3. `virtual-hardware: Run Program` を選択して task を実行します。

4. Windows では、user access control(UAC) を使用してモデルの実行を有効化しなければならない場合があります。

5. **Terminal** タブで出力をチェックします。

6.1.8 solution のデバッグ

AVH FVP でデバッグを開始する前に、対応する launch configuration を作成する必要があります。

1. **Run** > **Add Configuration...** に移動し、`Arm Debugger: Launch FVP` を選択します。

2.  をクリックして、**Run and Debug Configuration** ビジュアルエディタで `launch.json` のファイルを開きます。

3. **FVP Parameters** セクションで、次を入力します:

```
"fvpParameters": "\"${workspaceFolder}\"/fvp_config.txt"
```

4. **Target** セクションで、**Configuration Database Entry** を展開し"Corstone SSE-300 Ethos-U55 (MPS3) - Cortex-M55" をターゲットとして選択します。

5. `launch.json` のファイルを保存します。

デバッグセッションを開始するには以下を行います:

1. **CMSIS** ビューの上部にある  をクリックします。

2. **Arm Debugger FVP** configuration を選択します。

3. Windows では、user access control(UAC) を使用してモデルの実行を有効化しなければならない場合があります。

4. デバッガは `main` で停止します。これでアプリケーションをデバッグできます。

6.2 uVision を使用した新しいプロジェクトの作成

このセクションでは、uVision を使用して単純な"Hello world" サンプル project を作成、実行、デバッグするための基本的なワークフローについて説明し、uVision User's Guide 内のより詳細な手順へのリンクを提供します。ワークフローには次の手順が含まれます:

- project の作成
- project にソフトウェアコンポーネントを追加する
- project にソースコードを追加する
- project 設定の修正
- project のビルド
- uVision での virtual hardware の設定
- project の実行とデバッグ

6.2.1 project の作成

選択したハードウェアに必要なすべての基本ファイルを含む新しい project を作成します。詳細については、uVision User's Guide 内の Creating Applications を参照してください。

この手順では、Arm V2M-MPS3-SSE-300-FVP virtual hardware 用の project を作成する方法について説明します。他のスターターキットまたはボードの場合は手順を調整してください。

1. uVision をインストールします。

2. uVision のメニューバーから **Project** > **New µVision Project** を選択します。

3. 空のフォルダを選択し、project の名前を入力します。(例:`hello`) **Save** をクリックすると、指定した名前 (`hello.uvprojx`) の空のプロジェクトファイルが作成されます。**Select Device for Target** ダイアログボックスが開きます。

4. SSE-300-MPS3 を選択して **OK** をクリックします。

デバイスを選択することで、コンパイラ制御、リンカのメモリレイアウト、フラッシュプログラミングアルゴリズムなどの不可欠なツール設定が行われます。

6.2.2 project にソフトウェアコンポーネントを追加する

Manage Run-Time Environment ダイアログボックスが開き、選択したデバイスに対する、インストール済みで使用可能なソフトウェアコンポーネントが表示されます。

使用する関連ソフトウェアコンポーネントを選択します。詳細については、Managing Run-Time Environment を参照してください。

以下のコンポーネントを選択します:

- ::CMSIS:CORE
- ::CMSIS:OS Tick (API):SysTick
- ::CMSIS:RTOS2 API:Keil RTX5:Source
- ::CMSIS-Driver:USAART (API):USART ('1' をセット
- ::CMSIS-Compiler:CORE
- ::CMSIS-Compiler:STDOUT (API):Custom
- ::Device:Definition
- ::Device:Startup:C Startup
- ::Device:USART Retarget
- ::Device:Native Driver:SysCounter
- ::Device:Native Driver:SysTimer
- ::Device:Native Driver:Timeout
- ::Device:Native Driver:UART

6.2.3 project にソースコードを追加する

main.h ヘッダファイルと helloworld.c ファイルを project に追加し、project 固有のコードをファイルに追加します。

1. **Project** ウィンドウで、**Source Group 1** を右クリックし、**Add New Item to Group** ダイアログボックスを開きます。

2. **Header File (.h)** をクリックし、`main` という名前を追加して、**OK** をクリックします。

3. 次のコードをコピーして `main.h` のファイルに貼り付けます:

```
#ifndef MAIN_H__
#define MAIN_H__

/* Prototypes */
extern void app_initialize (void);
extern int stdio_init (void);
#endif
```

4. **Project** ウィンドウで、**Source Group 1** を右クリックし、**Add New Item to Group** ダイアログボックスを開きます。

5. **C File (.c)** をクリックし、`helloworld` という名前を追加して、**OK** をクリックします。

6. 次のコードをコピーして `helloworld.c` のファイルに貼り付けます:

```
#include <stdio.h>
#include "main.h"
#include "cmsis_os2.h"
const osThreadAttr_t app_main_attr = {
  .attr_bits = osThreadPrivileged //Set thread to privileged
};

/*----------------------------------------------------------------------------
* Application main thread
*----------------------------------------------------------------------------*/
static void app_main (void *argument) {
  (void)argument;
  for(int count = 0; count < 10; count++) {
    printf("Hello World %d¥r¥n", count);
    osDelay(1000U);
  }
  osDelay(osWaitForever);
}

/*----------------------------------------------------------------------------
* Application initialization
*----------------------------------------------------------------------------*/
void app_initialize (void) {
  osThreadNew(app_main, NULL, &app_main_attr);
}
```

7. **Project** ウィンドウで、**Source Group 1** を右クリックし、**Add New Item to Group** ダイアログボックスを開きます。

8. **C File (.c)** をクリックし、`main` という名前を追加して、**OK** をクリックします。

9. 次のコードをコピーして `main.c` のファイルに貼り付けます:

```
#include "RTE_Components.h"
#include CMSIS_device_header
#include "cmsis_os2.h"
#include "main.h"
int main() {
  osKernelInitialize(); // Initialize CMSIS-RTOS2
  app_initialize(); // Initialize application
  osKernelStart(); // Start thread execution
    for (;;) {
    }
}
```

6.2.4 project 設定の修正

プロジェクトをビルドする前に、次のプロジェクト設定を修正します。

1. uVision メニューバーから **Project** > **Options for target 'Target 1'** を選択するか  をクリックします。

2. **Linker** タブを選択します。

3. **Use Memory Layout from Target Dialog** の選択を解除します。

4. **disable Warnings** ボックスに"`6314`" を追加します。

5. **Scatter file** の横にある **Edit** をクリックします。**OK** をクリックします。

6. `hello.sct` の内容を次のコードに置き換えます:

```
LR_ROM0 0x10000000 0x00200000 {
  ER_ROM0 0x10000000 0x00200000 {
    *.o (RESET, +First)
    *(InRoot$$Sections)
    *(+RO +XO)
  }
  RW_NOINIT 0x30000000 UNINIT (0x00020000 - 0x00000C00 - 0x00000200 - 0) {
    *.o(.bss.noinit)
    *.o(.bss.noinit.*)
  }
  RW_RAM0 AlignExpr(+0, 8) (0x00020000 - 0x00000C00 - 0x00000200 - 0 -
AlignExpr(ImageLength(RW_NOINIT), 8)) {
    *(+RW +ZI)
  }
  ARM_LIB_HEAP (AlignExpr(+0, 8)) EMPTY 0x00000C00 { ; Reserve empty region for
heap
  }
  ARM_LIB_STACK (0x30000000 + 0x00020000 - 0) EMPTY -0x00000200 { ; Reserve empty
region for stack
  }
  RW_RAM1 0x00000000 0x00080000 {
    .ANY (+RW +ZI)
  }
  RW_RAM2 0x01000000 0x00100000 {
    .ANY (+RW +ZI)
  }
  RW_RAM3 0x20000000 0x00020000 {
    .ANY (+RW +ZI)
  }
}
```

6.2.5 project のビルド

project をビルドするには、  をクリックします。**Build Output** ウィンドウにビルドの進行状況が表示されます。

project をビルドするためのその他のオプションについては、uVision User's Guide 内の Build the project を参照してください。

6.2.6 uVision での virtual hardware の設定

この例で使用されている V2M-MPS3-SSE-300-FVP などの virtual hardware 上でプロジェクトを実行するには、モデルをコンフィギュレーションする必要があります。

1. uVision メニューバーから **Project** > **Options for target 'Target 1'** を選択するか  をクリックします。

2. **Debug** タブを選択します。

3. 右側で"Models ARMv8-M Debugger" を選択し、**Settings** をクリックします。新しいウィンドウが開きますので、以下の設定を入力します:
   a. **Command** ボックスに、AVH FVP モデルへのパスを入力します。例:
   ```
   C:¥Keil_v5¥ARM¥avh-fvp¥bin¥models¥FVP_Corstone_SSE-300.exe
   ```
   b. **Target** ボックスに `cpu` と入力します。**OK** を 2 回クリックします。

4. **disable Warnings** ボックスに"`6314`" を追加します。

5. **Scatter file** の横にある **Edit** をクリックします。**OK** をクリックします。

6.2.7 project の実行とデバッグ

virtual hardware 上でアプリケーションを実行またはデバッグするには:

1. uVision メニューバーから、**Debug** > **Start/Stop Debug Session** を選択するか、  をクリックします。uVision **Debug** ビューが開きます。

Windows では、user access control(UAC) を使用してモデルの実行を有効化しなければならない場合があります。

2. デバッガは `main` で停止します。これでアプリケーションを実行できます。

3. uVision のメニューバーから、**Debug** > **Run** を選択するか、  をクリックします。

4. **Telnet** ウィンドウの出力を確認します。

5. アプリケーションをデバッグするには、  、  、または  をクリックします。

6.  をクリックしてプログラムの実行を停止します。

7. uVision メニューバーから、**Debug** > **Start/Stop Debug Session** を選択するか、  をクリックしてデバッグセッションを終了します。

6.2.8 csolution フォーマットでの project の保存

Keil Studio でプロジェクトを使用したい場合は、csolution 形式で保存する必要があります。uVision のメニューバーで、**Project** > **Export** > **Save project to csolution format** を選択します。csolution と cproject YAML ファイルがプロジェクトディレクトリに保存されます。

7. 用語集

このセクションでは、Keil Studio と CMSIS の重要な概念を簡単に解説しています。より詳細な情報と、さらなる詳しい説明のあるリソースへのリンクについては、CMSIS の基本概念を参照してください。

CMSIS-Pack:
: 組み込みソフトウェアライブラリ、ドキュメント、デバイスパラメータ、評価ボードサポートを配布するためのオープンなパッケージの標準。CMSIS-Pack 標準は現在、Open-CMSIS-Pack プロジェクトの一部です。

CMSIS-Toolbox:
: Open-CMSIS-Pack 形式で定義されたコンポーネントを操作するためのコマンドラインツール群。CMSIS-Toolbox には、CMSIS-Packs のインストール、組み込みソフトウェアプロジェクトの定義と調整、ビルドの組織化を行うためのツールが含まれています。

CMSIS context:
: プロジェクト、ビルドタイプ(`Debug` や `Release` など)、ターゲット(ハードウェア) を組み合わせた CMSIS solution 内のビルドコンフィギュレーション。コンテキストは、 `Project.BuildType+Target` の形式で定義されます。詳細については、CMSIS context のドキュメントを参照してください。

CMSIS software components:
: CMSIS-Pack 内にパッケージ化された組み込みソフトウェア抽象化ライブラリ。詳細については、このガイドの CMSIS コンポーネントのセクションを参照してください。

CMSIS solution (別名 csolution):
: CMSIS-Toolbox で使用される YAML ベースのプロジェクト形式。詳細については、CMSIS Solution Project File Format を参照してください。

所有権通知

所有権通知については、オリジナルの Keil Microcontroller Development Kit (MDK) Version v6 Getting Started Guide 内の **Proprietary Notice** のセクションを参照してください。

製品およびドキュメント情報

製品およびドキュメント情報については、オリジナルの Keil Microcontroller Development Kit (MDK) Version v6 Getting Started Guide 内の **Product and document information** のセクションを参照してください。

凡例

以下のサブセクションでは、Arm のドキュメントで使用される表記規則について説明します。

Glossary

"Arm Glossary"は、Arm のドキュメントで使用されている用語と、それらの用語の定義のリストです。Arm Glossary には、Arm において用いる意味が一般に受け入れられている意味と異なる場合を除き、業界標準の用語は含まれていません。

詳細については、Arm® Glossary を参照してください: developer.arm.com/glossary

表記規則

Arm のドキュメントでは、特定の意味を伝えるために、表記規約を使用します。

表記	用法
斜体	引用
強調文字	メニュー名などのインターフェイス要素 信号名 記述リストの用語(適切な場合)
等幅表記 (monospace)	コマンド、ファイル名、プログラム名、ソースコードなど、キーボードから入力できるテキスト
下線付き等幅表記 (monospace underline)	コマンドまたはオプションの許可される省略形。完全なコマンドまたはオプション名の代わりに下線付きのテキストの内容を入力できます。
<and>	コードまたはコードフラグメント内に現れるアセンブラ構文内で置き換え可能な用語を囲っています。 例: `MRC p15, 0, <Rd>, <CRn>, <CRm>, <Opcode_2>`
小型英大文字 (SMALL CAPITALS)	Arm Glossary で定義されている特定の技術的意味を持つ用語。 例:**IMPLEMENTATION DEFINED**, **IMPLEMENTATION SPECIFIC**, **UNKNOWN**, および **UNPREDICTABLE** など

	推奨事項。これらの推奨事項に従わないと、システムの故障や損傷につながるおそれがあります。
	システム要件。これらの要件を守らないと、システムの故障や破損につながる可能性があります。
	システムの要件。これらの要件に従わないと、システムが故障したり破損したりすることが予測されます。
	注意が必要な重要な情報
	作業をより簡単に、より良好に、またはより迅速に行うことができる有用な tip
	お読みいただいている情報に関係する重要なポイントのリマインダ

役立つリソース

このドキュメントには、この製品に固有の情報が含まれています。その他の情報の詳細については、次のリソースを参照してください。

Arm のドキュメントへのアクセスは、その Confidentiality (機密性) に応じて異なります:

- Non-Confidential (非機密) ドキュメントは developer.arm.com/documentation から入手できます。以下の表に各ドキュメントに対するオンラインバージョンのリンクがあります。
- Confidential (機密) ドキュメントは製品パッケージを通じてライセンシにのみ提供されます。

Arm product resources	Document ID	Confidentiality
Arm Keil Studio Cloud User Guide	102497	Non-Confidential
Arm Keil Studio Visual Studio Code Extensions User Guide	108029	Non-Confidential
uVision User Guide	101407	Non-Confidential