life.augmented

# UM1718
# ユーザ・マニュアル

## STM32CubeMX：STM32 の設定と初期化の C コード生成

## 概要

STM32CubeMX は、STM32 製品向けのグラフィカルなツールです。このツールは、STM32Cube 構想の一環として（セクション 1：参照）、スタンドアロンのアプリケーションとして提供されるとともに STM32CubeIDE ツールチェーンにも含まれます。

STM32CubeMX の主要な機能は次のとおりです。

- STM32 のポートフォリオ全体から**マイクロコントローラを容易に選択**
- ST マイクロエレクトロニクス製ボードの一覧から**ボードを選択**
- **容易なマイクロコントローラ設定**（ピン、クロック・ツリー、ペリフェラル、ミドルウェア）および対応する初期化 C コードの容易な生成
- 以前に保存した設定を新しいマイクロコントローラ・プロジェクトにインポートすることで、**容易に他のマイクロコントローラに切り替え**
- **互換性のあるマイクロコントローラへ現在の設定を容易にエクスポート**
- **設定レポートの生成**
- 選択した統合開発環境ツールチェーン向けに**組込み C プロジェクトを生成**。STM32CubeMX のプロジェクトは、生成された初期化 C コード、MISRA 2004 準拠の STM32 HAL ドライバ、ユーザ設定に必要なミドルウェア・スタック、選択した IDE でプロジェクトを開いてビルドするためのすべての関連ファイルなどで構成されています。
- ユーザ定義のアプリケーション・シーケンスで使用できる**消費電力計算機能**
- STM32CubeMX を最新の状態に維持する**自己更新機能**
- ユーザ・アプリケーション開発に必要な STM32Cube 組込みソフトウェアのダウンロードと更新（STM32Cube 組込みソフトウェア製品の詳細については付録 Eを参照してください）

STM32CubeMX では、ユーザ・インタフェースを提供し、STM32 マイクロコントローラの設計とファームウェアのソリューションに準拠した C コードを生成しますが、ペリフェラルとファームウェアの実際の詳しい実装については、製品の技術ドキュメントを参照する必要があります。

以下のドキュメントを www.st.com で入手できます。

- STM32 マイクロコントローラのリファレンス・マニュアルおよびデータシート
- 以下のマイクロコントローラ向けの STM32Cube HAL/LL ドライバ・ユーザ・マニュアル：STM32C0 (UM2985)、STM32F0 (UM1785)、STM32F1 (UM1850)、STM32F2 (UM1940)、STM32F3 (UM1786)、STM32F4 (UM1725)、STM32F7 (UM1905)、STM32G0 (UM2303)、STM32G4 (UM2570)、STM32H7 (UM2217)、STM32L0 (UM1749)、STM32L1 (UM1816)、STM32L4/L4+ (UM1884)、STM32L5 (UM2659)、STM32MP1 (https://wiki.st.com/stm32mpu)、STM32U5 (UM2883)、STM32WL (UM2642)、STM32WB (UM2442)

# 目次

# 表の一覧

# 図の一覧

# 1 STM32Cube の概要

STM32Cube は開発の工数、時間、コストを削減して開発業務を効率化するために ST マイクロエレクトロニクスが独自に提唱している取り組みです。32 bit Arm®(a) Cortex® コアに基づく STM32 デバイスのポートフォリオ全体に対応しています。

STM32Cube は、以下の要素から構成されています。

- STM32CubeMX：グラフィック･ウィザードにより初期化 C コードを生成する、グラフィック･インタフェースを備えたソフトウェア設定ツールです。
- 総合的な組込みソフトウェア･プラットフォーム：シリーズごとに STM32F2 用の STM32CubeF2、STM32F4 用の STM32CubeF4 などが用意されています。
  - STM32Cube HAL：STM32 ポートフォリオの製品間で最大限の移植性を実現する STM32 抽象化レイヤ組込みソフトウェアです。
  - Low Layer（LL）API：専門家向けの高速軽量なレイヤを提供する API です。HAL よりもハードウェア寄りのレイヤを実装します。LL API を使用できるペリフェラルは一部のものに限られます。
  - 一貫性のあるミドルウェア･コンポーネント群：RTOS、USB、TCP/IP などがあります。
  - すべての組込みソフトウェア･ユーティリティ：必要なすべてのサンプルとともに提供されます。

arm

a. Arm は、米国内およびその他の地域における Arm Limited 社（またはその子会社）の登録商標です。

# 2 STM32CubeMX 入門

## 2.1 原則

要件（コア·アーキテクチャ、機能、メモリ容量、性能など）に最も適したマイクロコントローラを迅速に見極める必要があります。ボード設計では、設計するボードのレイアウトに対してマイクロコントローラのピン設定を最適化し、アプリケーションの要件を満たすこと（ペリフェラルの動作モードの選択）が重要です。一方、組込みシステムの開発では、特定のターゲット·デバイス向けに新しいアプリケーションを開発し、既存の設計を別のマイクロコントローラに移行することがより重要になります。

新しいプラットフォームに移行し、新しいファームウェア·ドライバに合わせて C コードを更新するために費やされる時間は、プロジェクトに無用の遅延をもたらします。STM32CubeMX は、STM32Cube 構想の一環として開発されています。その目的は、ソフトウェアを最大限再利用し、目標とするシステムの構築に要する時間を最短化するという重要な要件に応えることにあります。

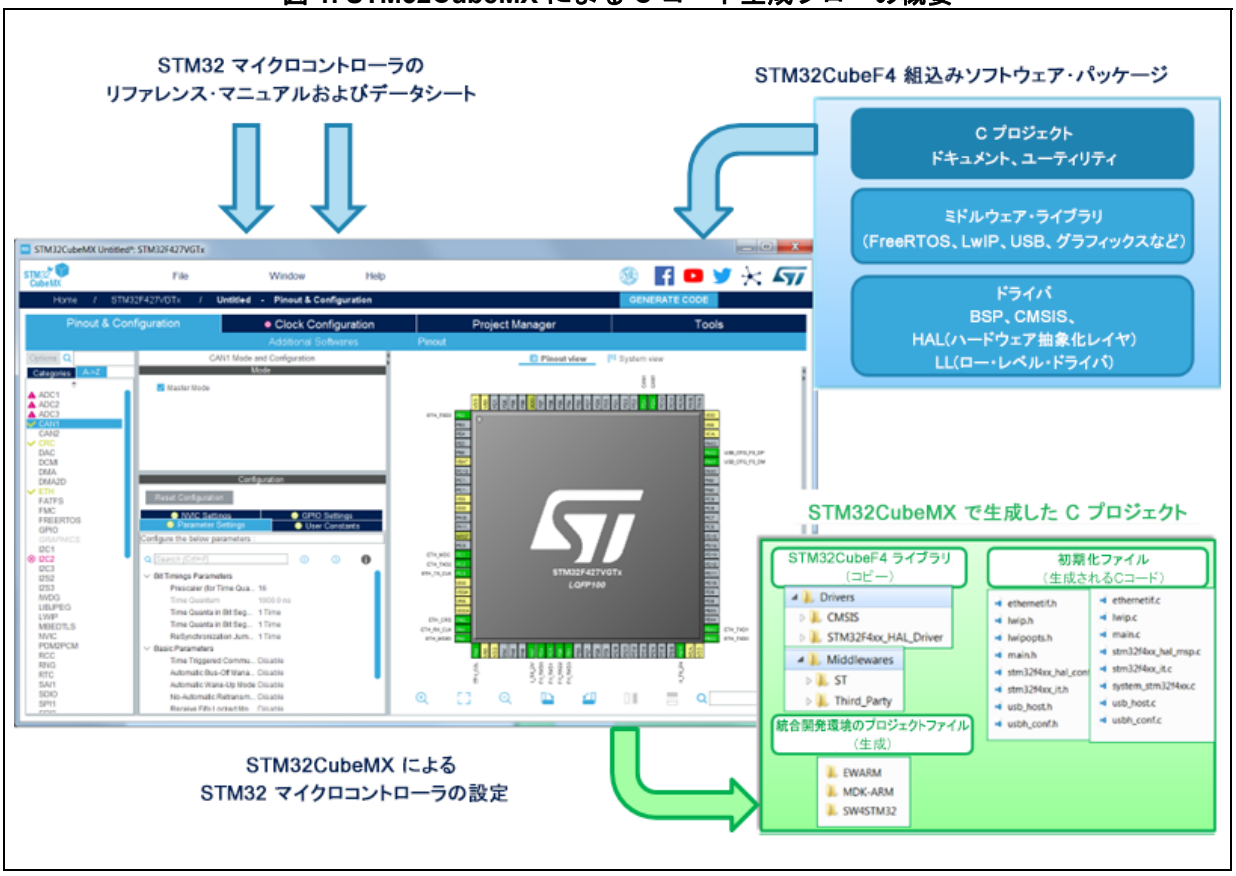
- ソフトウェアの再利用やアプリケーション設計の移植性は、STM32Cube ファームウェア·ソリューションで実現しています。このソリューションは、STM32 ポートフォリオ全体で共用できるハードウェア抽象化レイヤーを提供します。
- STM32CubeMX には、STM32 マイクロコントローラ、ペリフェラル、ミドルウェア（LwIPとUSB通信プロトコル·スタック、小型組込みシステム用のFatFsファイルシステム、FreeRTOS）に関する知見が組み込まれていることから、移行に要する時間の最適化を実現できます。

STM32CubeMX のグラフィック·インタフェースには次の機能があります。

- マイクロコントローラのピン、クロック·ツリーおよび選択したペリフェラルとミドルウェアの動作モードを短時間で容易に設定します。
- ボード設計向けのピン設定レポートを生成します。
- ユーザ定義の動作モードにデバイスを設定するために必要なすべてのライブラリおよび初期化 C コードを収めた完全なプロジェクトを生成します。選択したアプリケーション開発環境（サポート対象の IDE を選択可能）から目的のプロジェクトを直接開き、そのままアプリケーション開発に進むことができます（図 1 を参照）。

設定作業では、STM32CubeMX によって競合や無効な設定が検出され、わかりやすいアイコンや便利なツールチップで指摘されます。

図 1. STM32CubeMX による C コード生成フローの概要



## 2.2 主な機能

STM32CubeMX は次の機能を備えています。

- **プロジェクト管理**

  STM32CubeMX では、プロジェクトの作成、保存、および以前に保存したプロジェクトのロードが可能です。

  – STM32CubeMX の起動時に、新規プロジェクトの作成、または保存されているプロジェクトのロードを選択できます。

  – プロジェクトを保存すると、プロジェクトで指定した各種設定が .ioc ファイルに保存されます。このファイルは、同じプロジェクトを STM32CubeMX に次回ロードするときに使用できます。

  STM32CubeMX では、保存されているプロジェクトを新規プロジェクトにインポートすることもできます。

  STM32CubeMX のプロジェクトには次の 2 種類があります。

  – マイクロコントローラ設定のみのプロジェクト: 専用のプロジェクト･フォルダに .ioc ファイルが保存されます。

  – マイクロコントローラ設定と生成された C コードを持つプロジェクト: 生成された C ソース･コードとともに .ioc ファイルが専用のプロジェクト･フォルダに保存されます。プロジェクトごとに設定できる .ioc ファイルは 1 つのみです。

- **マイクロコントローラ、ボード、またはサンプルに基づいて簡単に作成できるプロジェクト**

  新規プロジェクト･ウィンドウでは、ST マイクロエレクトロニクスの STM32 ポートフォリオにあるマイクロコントローラ、ボード、サンプル･プロジェクトを選択することでプロジェクトを作成できます。さまざまなフィルタ･オプションを使用して、目的のマイクロコントローラやボードを容易に選択できます。［Cross Selector］タブでは、競合製品と特性を比較してマイクロコントローラを選択することも可能です。この比較の基準は調整できます。

- **簡単なピン配置設定**

  – **[Pinout]** ビューの一覧からペリフェラルを選択して、アプリケーションに必要なペリフェラル･モードを設定できます。その設定に応じ、STM32CubeMX によってピンが割り当てられ、設定されます。

  – 上級ユーザは、**[Pinout]** ビューを使用してペリフェラルの機能を物理ピンに直接マッピングすることも可能です。STM32CubeMX の競合解決機能によって信号が意図せずに他のピンに移動しないように、信号をピンにロックする機能もあります。

  – ピン配置の設定は .csv ファイルとしてエクスポートできます。

- **完全なプロジェクトの生成**

  生成するプロジェクトは、ピン配置、一連の IDE で使用できるファームウェアとミドルウェアの初期化 C コードなどで構成されます。プロジェクトは STM32Cube の組込みソフトウェア･ライブラリに基づいています。プロジェクトで可能な操作は次のとおりです。

  – 前回設定したピン配置から、ミドルウェア、クロック･ツリー、サービス（RNG、CRC など）、ペリフェラルのパラメータなどの設定に進むことができます。STM32CubeMX によって、該当の初期化 C コードが生成されます。その結果、設定用と初期化用に生成された main.c ファイルと C ヘッダ･ファイル、必要な HAL とミドルウェア･ライブラリ、選択した IDE 固有のファイルを収めたプロジェクト･ディレクトリが作成されます。

  – ユーザ専用セクションにユーザ定義の C コードを追加することで、生成されたソースファイルを変更できます。次回の C コード生成の際も、このユーザ C コードが STM32CubeMX で保持されています（このユーザ C コードは、現在の設定との間に関連性がなくなるとコメントアウトされます）。

  – STM32CubeMX では、Freemarker のユーザ定義テンプレートである .ftl ファイルを使用してユーザ･ファイルを生成できます。

– [**Project settings**]メニューでは、生成した C コードを使用する開発ツールチェーン（IDE）を選択できます。STM32CubeMX では、IDE に関連するプロジェクト･ファイルはプロジェクト･フォルダに追加されます。したがって、プロジェクトを新規プロジェクトとして 3rd パーティの IDE に直接インポートできます。サポートされている IDE として、IAR™ EWARM、Keil™ MDK-ARM、Atollic® TrueSTUDIO®、AC6 System Workbench for STM32 などがあります。

---

**警告：** **STM32CubeMX の次期バージョンから TrueSTUDIO、SW4STM32、GPDSC がサポート対象外になります。進化した STM32CubeMX の利点をプロジェクトに反映するために、STM32CubeIDE をはじめとする他の IDE の導入をお勧めます。**

---

- **消費電力の計算**

  マイクロコントローラの部品番号とバッテリのタイプを選択して、アプリケーションのライフ･サイクルの各段階とそのパラメータ（周波数、有効化するペリフェラル、ステップの期間などの選択）の定義に着手できます。STM32CubeMX の消費電力計算機能で、該当の消費電力とバッテリ寿命の推定値が得られます。

- **クロック･ツリーの設定**

  STM32CubeMX では、デバイスのリファレンス･マニュアルに記述されているような、グラフィック形式のクロック･ツリーを提供します。デフォルト設定（クロック･ソース、プリスケーラ、周波数の値）はユーザ側で変更できます。その変更に伴ってクロック･ツリーが更新されます。無効な設定や制限事項があればハイライトされ、説明がツールチップに表示されます。クロック･ツリー設定の競合は競合解決機能で解決できます。ユーザが指定した設定と正確に一致する設定が存在しない場合、STM32CubeMX ではそれに最も近いものが提示されます。

- **STM32CubeMXと STM32Cube マイクロコントローラ･パッケージの自動更新**

  STM32CubeMX は、アップデータの機能を備えています。このアップデータでは、更新の有無を自動的に確認するか、オンデマンドで確認するかを設定できます。また、STM32Cube ファームウェア･ライブラリ･パッケージの更新同様に、STM32CubeMX 自身の更新にも対応しています。このアップデータ機能では、前回インストールしたパッケージを削除することもできます。

- **レポートの生成**

  ユーザ設定作業の内容を記述したレポートを .pdf 形式と .csv 形式で生成できます。

- **CMSIS-Pack フォーマットの組込みソフトウェア･パッケージのサポート**

  STM32CubeMXでは、CMSIS-Pack フォーマットで提供される組込みソフトウェア･パッケージの更新をダウンロードで入手できます。これらの新しいリリースに属するソフトウェア･コンポーネントから該当のものを選択し、現在のプロジェクトに追加できます。

- **STM32PackCreator によるソフトウェア･パッケージの生成**

  STM32PackCreator は、STM32CubeMX とともに Utilities フォルダにインストールされるグラフィカル･ツールです。ソフトウェア･パッケージの作成や STM32CubeMX 向けに機能強化した STM32Cube 拡張パッケージの作成が可能です。STM32CubeMX のツール･ビューにある ST ツールのタブからこのツールを起動できます。

- **コンテキスト･ヘルプ**

  コア、シリーズ、ペリフェラル、ミドルウェアの上にマウス･カーソルを移動すると、コンテキスト･ヘルプのウィンドウが表示されます。このヘルプには、選択した項目に関する簡単な説明と、関連するドキュメントへのリンクが記述されています。

- **ST ツールへのアクセス**

  STM32CubeMX プロジェクトのツール･タブではツールを直接起動できるほか、www.st.com のツールのダウンロード･ページにアクセスすることもできます。

- **ビデオ･チュートリアル**

  STM32CubeMX からビデオ･チュートリアルを参照して再生できます。ビデオ･チュートリアルのブラウザには［Help］メニューからアクセスできます。

## 2.3 ルールと制限事項

- ここで生成できる C コードは、ペリフェラルとミドルウェアを初期化するコードのみです。これらのコードは、STM32Cube の HAL ファームウェア･ライブラリに基づいています。
- STM32CubeMX の C コード生成機能で扱うことができるコードは、STM32Cube組込みソフトウェア･パッケージに付属するドライバを使用するペリフェラルおよびミドルウェア･コンポーネントの初期化コードのみです。ペリフェラルおよびミドルウェア･コンポーネントの中には、このコード生成機能に未対応のものがあります。
- ピン割当てルールの説明については付録 A を参照してください。
- STM32CubeMX による C コード生成の設計上の選択肢や制限事項の説明については、付録 B を参照してください。

# 3 STM32CubeMX のインストールと実行

## 3.1 システム要件

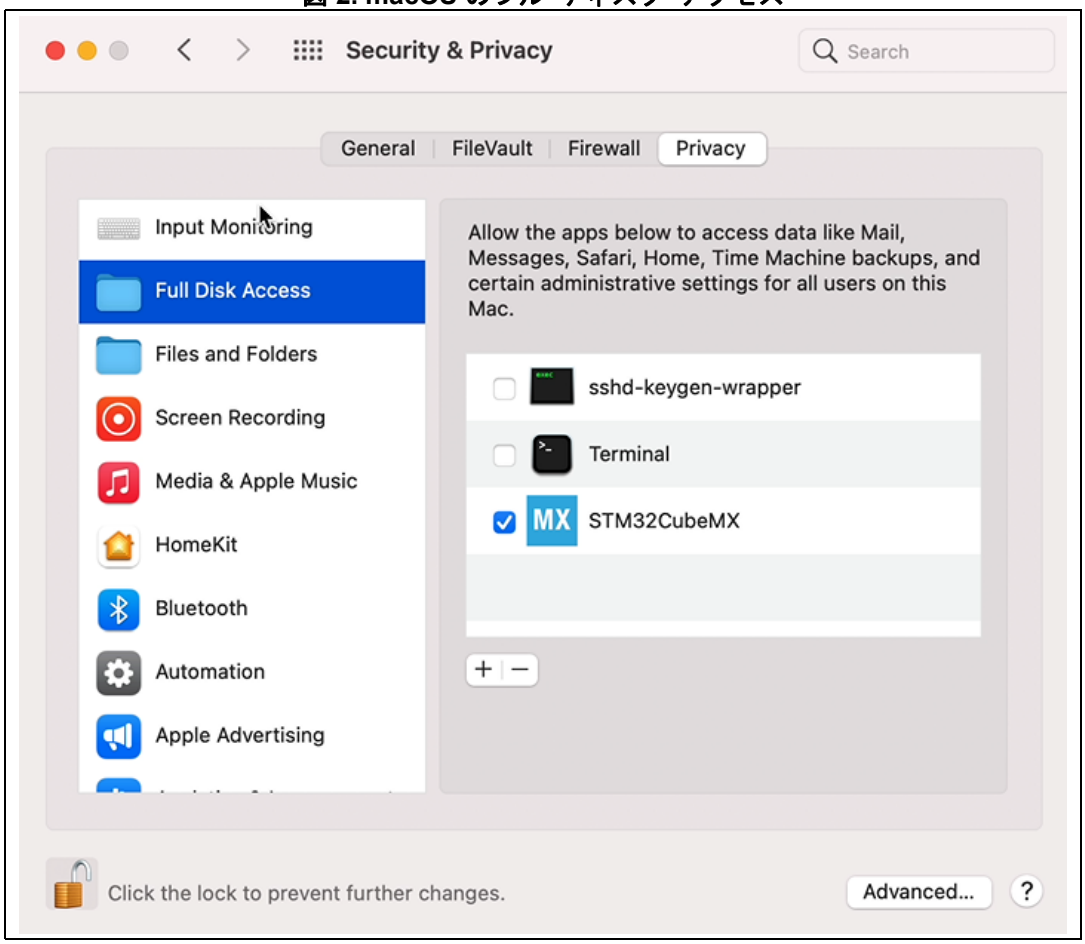
### 3.1.1 サポート対象のオペレーティング·システムとアーキテクチャ

- Windows® 8.1：64-bit（x64）
- Windows® 10：64-bit（x64）
- Windows® 11：64-bit（x64）
- Linux®：64-bit（x64）（Ubuntu LTS および Fedora で動作確認済み）
- macOS®：64-bit（x64）（macOS Catalina 以降で動作確認済み）

macOS の場合、ファイル·システムからのプロジェクト·ファイルのロードや他のパッケージのインストールにはフル·ディスク·アクセスが必要です。STM32CubeMX のフル·ディスク·アクセスを有効にするには、次の手順を実行します。

1. ［System preferences］に移動して［Security & Privacy］ウィンドウを開きます（図 2）。
2. ［Privacy］タブを選択します。
3. 左側のパネルから［Full Disk Access］を選択します。
4. STM32CubeMX のチェックボックスをクリックしてフル·ディスク·アクセスを有効にします。

**図 2. macOS のフル·ディスク·アクセス**



### 3.1.2 メモリ要件

- 推奨最小RAM容量：2 GB

### 3.1.3 ソフトウェア要件

STM32CubeMX の自己更新パッケージをダウンロードし、STM32CubeMX を次回起動したときに更新プロセスを完了するには管理者権限が必要です。

#### Java™ Runtime Environment

STM32CubeMX の V6.2.0 以降、その実行に必要な Java Runtime Environment （JRE™[a]）は STM32CubeMX に組み込まれ、ユーザのPCにインストールされている JRE は不要になりました。

- STM32CubeMX 6.3 にバンドルされている JRE は AdoptOpenJDK-11.0.10+9 と JavaFX-11.0.2 です。
- STM32CubeMX 6.2 にバンドルされた JRE は BellSoft の Liberica 1.8.0_265 です。

STM32CubeMX V6.2.0 より前のバージョンでは、JRE をインストールする必要があります。JRE のバージョンに関する制約は、次のとおりです。

- 64 bit バージョンが必須です。32 bit バージョンはサポートされていません。
- STM32PackCreator コンパニオン･ツールには JavaFX に対応した JRE が必要です。
- JRE の最小バージョンは 1.8_45 です（1.8_251 には既知の制約があります）。
- バージョン 11 はサポート対象、バージョン 7、9、10、および 12 以降はサポート対象外です。

ST マイクロエレクトロニクスでは、以下の JRE の使用を推進しています。

- Oracle [a]（ライセンス料が発生します）
- Amazon Corretto™[b]（OpenJDK に基づく無料のソリューションです。JDK インストーラの使用をお勧めします）

その他の JRE では、STM32CubeMX の動作が保証されません。

#### macOS のソフトウェア要件

- macOS コンピュータには、Xcode をインストールする必要があります。
- Apple® M1 プロセッサを搭載した macOS コンピュータには Xcode と Rosetta の両方をインストールする必要があります。

## 3.2 STM32CubeMXスタンドアロン･バージョンのインストールとアンインストール

### 3.2.1 STM32CubeMXスタンドアロン･バージョンのインストール

STM32CubeMX をインストールするには､管理者権限があることを確認したうえで以下の手順を実行します。

1. インターネット･ブラウザで www.st.com/stm32cubemx のページを開きます
2. ［Get Software］をクリックして、ソフトウェアのダウンロード･セクションに移動します。

a. Oracle および Java は Oracle 社およびその関連会社の登録商標です。

b. その他の商標は、それぞれの所有者に帰属します。

**Windows の場合**

a) STM32CubeMX-Win の行の［Get latest］をクリックしてパッケージをダウンロードします。
b) ダウンロードしたパッケージを展開（解凍）します。
c) 管理者権限があることを確認します。
d) SetupSTM32CubeMX-VERSION-Win.exe をダブルクリックしてインストール･ウィザードを起動します。

**注：** **Windows へのインストールが正常に終了すると、デスクトップに STM32CubeMX のアイコンが表示され、プログラムのメニューから STM32CubeMX アプリケーションを使用できるようになります。STM32CubeMX の .ioc ファイルは立方体のアイコンで表示され、これをダブルクリックすると STM32CubeMX でプロジェクトが開きます。Windows で使用する場合、プログラムのメニューでは最後にインストールした STM32CubeMX のみが有効になります。別のインストール先フォルダを指定すれば、旧バージョンを PC 上に残しておくことができます（ただし、お勧めしません）。別のフォルダを指定しない場合、旧バージョンは新バージョンのインストールによって上書きされます。**

**Linux の場合**：

a) **STM32CubeMX-Lin** の行の［Get software］をクリックしてパッケージをダウンロードします。
b) ダウンロードしたパッケージを展開（解凍）します。
c) インストール先ディレクトリにアクセスできる管理者権限があることを確認します。root（または sudo）でインストーラを実行し、STM32CubeMX を共有ディレクトリにインストールできます。
d) ファイルを実行可能とするために、**chmod 777 SetupSTM32CubeMX-VERSION** を実行してプロパティを変更します。
e) **SetupSTM32CubeMX-VERSION** ファイルをダブルクリックするか、コンソール･ウィンドウから起動します。

**macOS の場合**：

a) **STM32CubeMX-Mac** の行の［Get software］をクリックしてパッケージをダウンロードします。
b) ダウンロードしたパッケージを展開（解凍）します。
c) 管理者権限があることを確認します。
d) アプリケーション･ファイルの **SetupSTM32CubeMX-VERSION.app** をダブルクリックしてインストール･ウィザードを起動します。

エラーが発生する場合は、次のコマンドを試してください。- $sudo xattr -cr ~/SetupSTM32CubeMX-VERSION.app

### 3.2.2 コマンドラインからの STM32CubeMX のインストール

コンソール·ウィンドウからインストールを開始する方法には、コンソールの対話モードを使用する方法とスクリプトを使用する方法の 2 通りがあります。

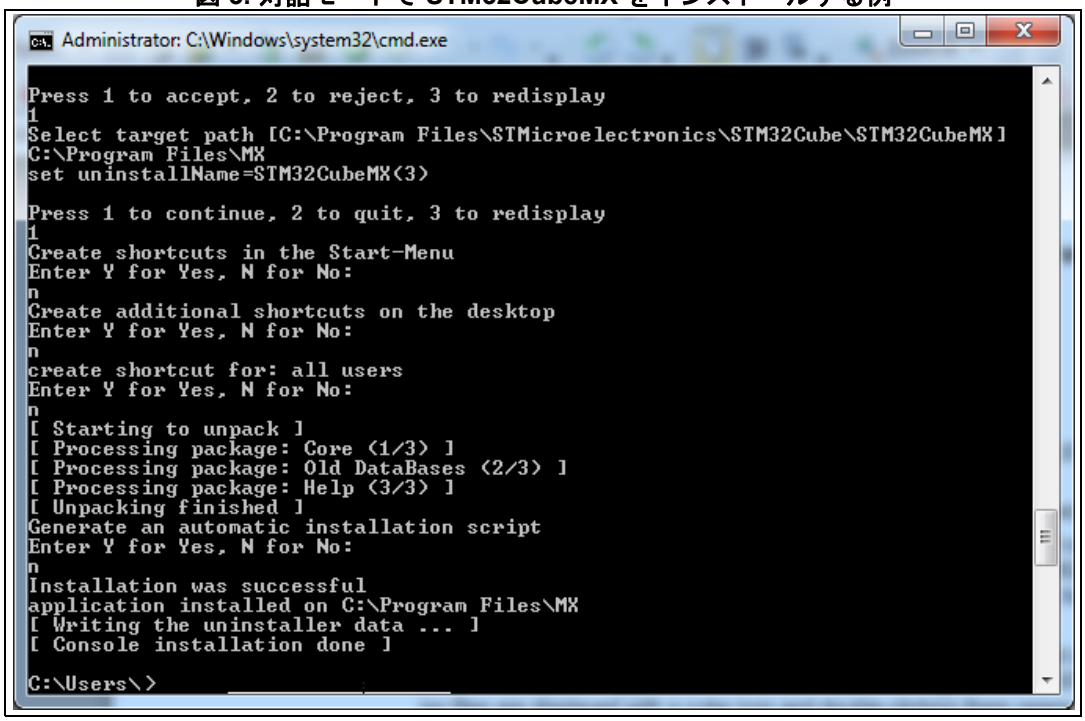
#### 対話モード

対話モードのインストールでは、以下の手順を実行します。

1. 自動展開のインストール·ファイル（SetupSTM32CubeMX-VERSION-Win.exe）をフォルダに展開（解凍）します。
2. 管理者権限でコンソール·ウィンドウを開きます。
3. 展開先のフォルダに移動します（cd <フォルダ·パス>）。
4. コマンド **`jre\bin\java -jar SetupSTM32CubeMX-<VERSION>.exe -console`** を実行します。

インストールのステップごとに回答が求められます（図 3 を参照）。

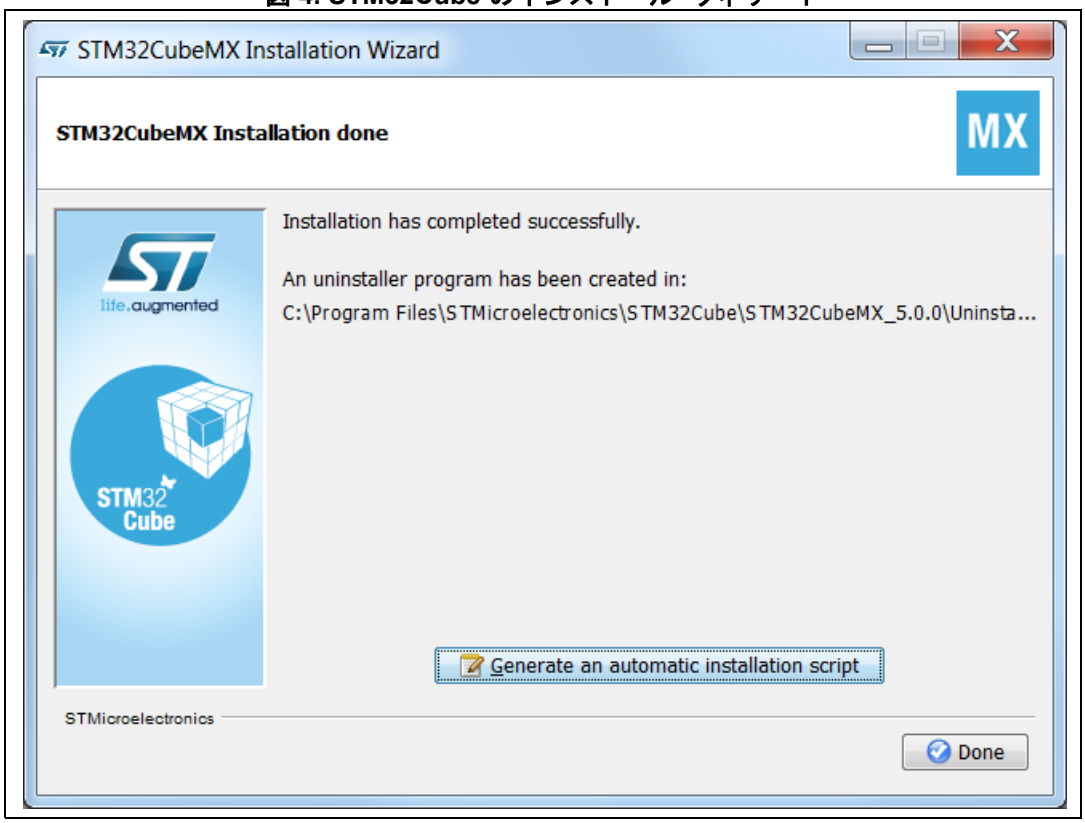
**図 3. 対話モードで STM32CubeMX をインストールする例**



### 自動インストール·モード

STM32CubeMX のインストールをグラフィック·ウィザードまたはコンソール·モードのどちらで実行した場合も、その終了時に、ユーザ固有の設定を記述した自動インストールのスクリプトを生成できます（図 4 を参照）。

**図 4. STM32Cube のインストール·ウィザード**




つづいて、管理者権限でコンソール·ウィンドウに下記のコマンドを入力し、実行することでインストールを開始できます。

```
SetupSTM32CubeMX-VERSION-Win.exe ABSOLUTE_PATH_TO_AUTO_INSTALL.xml
```

## 3.2.3 STM32CubeMX のスタンドアロン·バージョンのアンインストール

### macOS® 上の STM32CubeMX のアンインストール

Mac 上の STM32CubeMX をアンインストールするには次の 2 通りの方法があります。

1. STM32CubeMX.VERSION.app をゴミ箱に移動します。
2. 次のコマンドラインを使用します。
   a) STM32CubeMX 6.2.x 以降のバージョンのみ：
   cd SetupSTM32CubeMX-VERSION.app/Contents/Resources/Uninstaller
   ./uninstall.sh
   b) STM32CubeMX 6.1.x 以前のバージョンのみ：
   java -jar SetupSTM32CubeMX-VERSION.app/Contents/Resources/Uninstaller/uninstaller.jar.

### Linux® 上の STM32CubeMX のアンインストール

Linux 上の STM32CubeMX をアンインストールするには次の 2 通りの方法があります。

1. シェル･プロンプトからアンインストール･スクリプトを実行します。
   a) STM32CubeMX 6.2.x 以降のバージョン：
      cd <STM32CubeMX のインストール･パス>/Uninstaller
      uninstall.sh
   b) STM32CubeMX 6.1.x 以前のバージョン：
      java -jar <STM32CubeMX のインストール･パス>/Uninstaller/uninstaller.jar.
2. ファイル･エクスプローラから次の手順を実行します。
   a) <STM32CubeMX のインストール･パス>/Uninstaller に移動します。
   b) STM32CubeMX 6.2.x 以降のバージョン：
      uninstall.sh スクリプトをダブルクリックします。
   c) STM32CubeMX 6.1.x 以前のバージョン：
      デスクトップで、アンインストールを開始するショートカットをダブルクリックします。

### Windows® 上の STM32CubeMX のアンインストール

Windows 上の STM32CubeMX をアンインストールするには次の 3 通りの方法があります。

1. Windows のコントロール･パネルを使用する方法：
   a) Windows の**コントロール･パネル**にある［**プログラムと機能**］を選択して、コンピュータにインストールされているプログラムのリストを表示します。
   b) **STM32CubeMX** を右クリックして［**Uninstall**］を選択します。
2. シェル･プロンプトから管理者権限で次のコマンドを実行します。
   a) STM32CubeMX 6.2.x 以降のバージョン：
      cd <STM32CubeMX のインストール･パス>/Uninstaller
      uninstall.bat
   b) STM32CubeMX 6.1.x 以前のバージョン：
      java -jar <STM32CubeMX のインストール･パス>/Uninstaller/uninstaller.jar
3. Windowsのファイル･エクスプローラを使用する方法：
   a) STM32CubeMX 6.2.x 以降のバージョン：
      STM32CubeMX のインストール･ディレクトリにある Uninstaller フォルダに移動します。
      uninstall.bat を右クリックして［管理者として実行］を選択します。
   b) STM32CubeMX 6.1.x 以前のバージョンのみ：
      STM32CubeMX のインストール･ディレクトリにある Uninstaller フォルダに移動します。
      startuninstall.exe またはデスクトップ上のアンインストールのショートカットをダブルクリックします。

## 3.3 STM32CubeMX の起動

プロキシのもとで STM32CubeMX を実行する場合はセクション 3.4.1 : を参照してください。

### 3.3.1 スタンドアロン･アプリケーションとしての STM32CubeMX の実行

Windows 上で STM32CubeMX をスタンドアロン･アプリケーションとして実行するには、［プログラム ファイル］→［ST Microelectronics］→［STM32CubeMX］から STM32CubeMX を選択するか、デスクトップ上の STM32CubeMX アイコンをダブルクリックします。

Linux 上で STM32CubeMX をスタンドアロン･アプリケーションとして実行するには、STM32CubeMX のインストール先ディレクトリで STM32CubeMX の実行可能ファイルを起動します。

MacOS 上で STM32CubeMX をスタンドアロン･アプリケーションとして実行するには、Launchpad から STM32CubeMX アプリケーションを起動します。

**注 :** **MacOSの場合、STM32CubeMXのデスクトップ･アイコンはありません。**

### 3.3.2 コマンドライン･モードでの STM32CubeMX の実行

他のツールとの統合を容易にするために、STM32CubeMX にはコマンドライン･モードが用意されています。一連のコマンド（表 1 に一覧を示しました）を使用して、以下の操作を実行できます。

- マイクロコントローラのロード
- 既存の設定のロード
- 現在の設定の保存
- プロジェクト･パラメータの設定と対応するコードの生成
- テンプレートに基づくユーザ･コードの生成
- 品名によって特定されるボードのロード
- 組込みソフトウェア･パッケージ（ソフトウェア･パッケージ本体と STM32Cube マイクロコントローラ･パッケージ）のリストのリフレッシュおよびパッケージのインストールと削除
- プロジェクトに追加するその他のソフトウェア･パッケージ･コンポーネントの選択

コマンドライン･モードには次の 3 種類があります。

- 対話型のコマンドライン･モードで STM32CubeMX を実行するには、以下のコマンドラインを使用します。
  - Windows の場合 :

    ```
    cd <STM32CubeMX のインストール･パス >
    jre\bin\java -jar STM32CubeMX.exe -i
    ```

  - Linux および MacOS の場合 :

    ```
    STM32CubeMX -i
    ```

    アプリケーションがコマンドを受け付けられる状態にあることを示す「MX>」プロンプトが表示されます。
- スクリプトからコマンドを取得してコマンドライン･モードで STM32CubeMX を実行するには、以下のコマンドラインを使用します。
  - Windows の場合 :

    ```
    cd <STM32CubeMX のインストール･パス >
    jre\bin\java -jar STM32CubeMX.exe -s < スクリプト･ファイル名 >
    ```

  - Linux および MacOS の場合 :

**STM32CubeMX –s <スクリプト･ファイル名>**

スクリプト･ファイルには、実行するすべてのコマンドを記述しておく必要があります。以下にスクリプト･ファイルの内容の例を示します。

```
load STM32F417VETx
project name MyFirstMXGeneratedProject
project toolchain "MDK-ARM v4"
project path C:\STM32CubeProjects\STM32F417VETx
project generate
exit
```

- スクリプトからコマンドを取得し、UI を使用しないコマンドライン･モードで STM32CubeMX を実行するには、以下のコマンドラインを使用します。
  - Windows の場合：

    **cd <STM32CubeMX のインストール･パス>**

    **jre\bin\java -jar STM32CubeMX.exe –q <スクリプト･ファイル名>**
  - Linux および MacOS の場合：

    **STM32CubeMX –q <スクリプト･ファイル名>**

    この場合も、MX プロンプトが表示されていればコマンドを入力できます。

**表 1. コマンドラインの要約**

| コマンド | 目的 | 例 |
|---|---|---|
| help | 使用可能なコマンドを一覧表示します。 | help |
| swmgr refresh | ダウンロード可能な組込みソフトウェア･パッケージ･バージョンのリストをリフレッシュします。 | swmgr refresh |
| swmgr install stm32cube_ <シリーズ> _<バージョン> ask | 指定した STM32Cube マイクロコントローラ･パッケージ･バージョンをインストールします。 | swmgr install stm32cube_f1_1.8.0 ask |
| swmgr remove stm32cube_ <シリーズ> _<バージョン> | 指定した STM32Cube マイクロコントローラ･パッケージ･バージョンを削除します。 | swmgr remove stm32cube_f1_1.8.0 |
| swmgr install <パッケージ･ベンダ>.<パッケージ名>. <パッケージ･バージョン> ask | 指定したパッケージ･バージョンをインストールします。 | swmgr install STMicroelectronics. X-CUBE-NFC4.1.4.1 ask |
| swmgr remove <パッケージ･ベンダ>.<パッケージ名>. <パッケージ･バージョン> | 指定したパッケージ･バージョンを削除します。 | swmgr remove STMicroelectronics. X-CUBE-BLE1.4.2.0 |
| swmgr install <ファイル名パス> <ライセンスモード (accept\|ask) | 組込みソフトウェア･パッケージをインストールします。 | swmgr install "C:\repo\packs\STMicroelectronics. X-CUBE-BLE1.4.2.0.pack" accept |

表 1. コマンドラインの要約 （続き）

| コマンド | 目的 | 例 |
|---|---|---|
| pack enable <ベンダ> <パッケージ>[/bundle] <バージョン> <クラス> <グループ>[/subgroup] [variant] | プロジェクトに追加するソフトウェア･パッケージ･コンポーネントを選択します。<br>2 番目および 5 番目のパラメータにある "/" の記述は、それぞれバンドルおよびサブグループの明示的な指定です（参照：Arm CMSIS パッケージの pdsc フォーマット）。<br>有効にするコンポーネントのパッケージ、バンドル、クラス、グループ、サブグループの名前を調べるには、コンポーネントを選択して［Additional Software］ウィンドウの［Hide/Show details］をクリックします。 | pack enable STMicroelectronics "X-CUBE-BLE1/BlueNRG-MS" 1.0.0 "Wireless" "Controller" |
| pack validate | プロジェクトの中で、前回の "pack validate" コマンド呼び出し以降に有効にしたすべてのパッケージ･コンポーネントを対象として実行します。 | pack validate |
| load <マイクロコントローラ> | 選択したマイクロコントローラをロードします。 | load STM32F101RCTx<br>load STM32F101Z(F-G)Tx |
| load <ボード品名> <allmodes\|nomode> | すべてのペリフェラルをデフォルト･モードに設定した状態（allmodes）または全く設定しない状態（nomode）で、選択したボードをロードします。 | loadboard NUCLEO-F030R8 allmodes<br>loadboard NUCLEO-F030R8 nomode |
| config load <ファイル名> | 以前に保存した設定をロードします。 | config load C:\Cube\ccmram\ccmram.ioc |
| config save <ファイル名> | 現在の設定を保存します。 | config save C:\Cube\ccmram\ccmram.ioc |
| config saveext <ファイル名> | 現在の設定を、そのすべてのパラメータとともに保存します。値がデフォルト設定のままでユーザが変更していないパラメータも保存されます。 | config saveext C:\Cube\ccmram\ccmram.ioc |
| config saveas <ファイル名> | 現在のプロジェクトを新しい名前で保存します。 | config saveas C:\Cube\ccmram2\ccmram2.ioc |
| csv pinout <ファイル名> | 現在のピン設定を .csv ファイルとしてエクスポートします。エクスポートしたファイルは、ボード･レイアウト･ツールにインポートできます。 | csv pinout mypinout.csv |
| script <ファイル名> | スクリプト･ファイルにあるすべてのコマンドを実行します。1 行に記述するコマンドは 1 つのみとする必要があります。 | script myscript.txt |
| project couplefilesbyip <0\|1> | このオプションでは 0 または 1 を指定します。0 を指定すると、ペリフェラルの初期化コードが main に生成され、1 を指定するとペリフェラルの初期化コードがそれぞれに専用の .c/.h ファイルに生成されます。 | project couplefilesbyip 1 |

表 1. コマンドラインの要約 （続き）

| コマンド | 目的 | 例 |
|---|---|---|
| setDriver <ペリフェラル名> <HAL \| LL> | STM32CubeMX でサポートしているシリーズでは、LL ドライバまたは HAL ドライバに基づいてペリフェラルの初期化コードを生成できます。<br>このコマンドラインにより、HAL ベースまたは LL ベースのコードのどちらでコードを生成するかをペリフェラルごとに選択できます。<br>デフォルト設定の場合、HAL ドライバに基づいてコードが生成されます。 | setDriver ADC LL<br>setDriver I2C HAL |
| generate code <パス> | 「STM32CubeMX 生成コード」のみを生成します。STM32Cube ファームウェア･ライブラリやツールチェーンのプロジェクト･ファイルなどを含む完全なプロジェクトは生成しません。<br>プロジェクトを生成するには、"project generate" を使用します。 | generate code C:\mypath |
| set tpl_path <パス> | ユーザ･テンプレート･ファイル（.ftl）を保存したソース･フォルダへのパスを設定します。<br>コード生成では、このフォルダに保存されたすべてのテンプレート･ファイルが使用されます。 | set tpl_path C:\myTemplates\ |
| set dest_path <パス> | 生成先フォルダへのパスを設定します。このフォルダには、ユーザ･テンプレートに従って生成されたコードが保存されます。 | set dest_path C:\myMXProject\inc\ |
| get tpl_path | ユーザ･テンプレートのソース･フォルダへのパス名を取得します。 | get tpl_path |
| get dest_path | ユーザ･テンプレートの生成先フォルダへのパス名を取得します。 | get dest_path |
| SetStructure <Advanced/Basic> | 生成するプロジェクト構造を選択します。 | SetStructure Basic |
| SetCopyLibrary <copy all / copy only / copy as reference> | プロジェクトへのリファレンス･ライブラリのコピー方法を選択します。 | SetCopyLibrary "copy all" |
| project setCustomFWPath <カスタム･ファームウェアの保存場所> | STM32Cube マイクロコントローラ･ソフトウェア･ライブラリへのパスとして、STM32Cube リポジトリへのパス（[Help] → [Updater settings] で指定）とは異なるパスを指定します。 | project SetCustomFwPath "F:/SharedRepository/STM32Cube_FW_F0_V1.11.0" |
| project toolchain <ツールチェーン> | プロジェクトで使用するツールチェーンを指定します。<br>"project generate" コマンドを実行すると、ここで指定したツールチェーンのプロジェクトが生成されます。 | project toolchain EWARM<br>project toolchain "MDK-ARM V4"<br>project toolchain "MDK-ARM V5"<br>project toolchain TrueSTUDIO<br>project toolchain SW4STM32 |

表 1. コマンドラインの要約 （続き）

| コマンド | 目的 | 例 |
|---|---|---|
| project name <名前> | プロジェクト名を指定します。 | project name ccmram |
| project path <パス> | プロジェクトの生成先へのパスを指定します。 | project path C:\Cube\ccmram |
| project generate | 完全なプロジェクトを生成します。 | project generate |
| exit | STM32CubeMX のプロセスを終了します。 | exit |

## 3.4 STM32CubeMX による更新の取得

STM32CubeMX には、インターネットにアクセスして以下の処理を実行する機能があります。

- 組込みソフトウェア･パッケージのダウンロード：STM32Cube マイクロコントローラ･パッケージ（フル･リリースとパッチ）、および Arm® CMIS-Pack フォーマットに基づく 3rd パーティのパッケージ（.pack）をダウンロードできます。
- 3rd パーティ･パッケージのユーザ定義リストの管理
- STM32CubeMX および組込みソフトウェア･パッケージの更新の確認
- STM32CubeMX の自己更新の実行
- STM32 マイクロコントローラの説明およびドキュメントのリフレッシュ

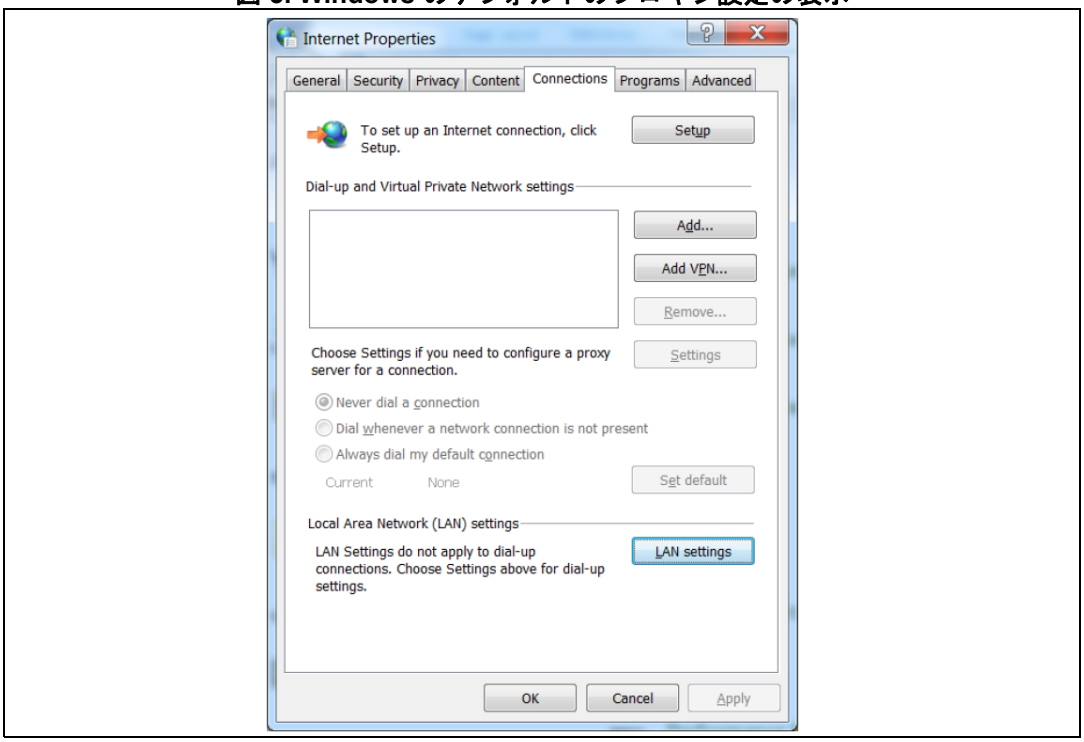
インストールおよび更新に関連するサブメニューが **[Help]** メニューにあります。また、[Home] ページからもそれらのサブメニューにアクセスできます。

インターネットにアクセスできないコンピュータではオフラインの更新も可能です（セクション 3.4.3 : を参照）。この更新を実行するには、ファイルシステムを参照して、存在する STM32Cube マイクロコントローラ･パッケージを選択します。

STM32CubeMX を実行している PC を、プロキシ･サーバを使用したコンピュータ･ネットワークに接続している場合、自己更新の入手やファームウェア･パッケージのダウンロードでは、そのプロキシ･サーバ経由で STM32CubeMX からインターネットにアクセスする必要があります。このような接続の設定については、セクション 3.4.2 : の説明を参照してください。

Windows のデフォルトのプロキシ設定を確認するには、コントロール･パネルで [Internet Options] を選択し、**[Connections]** タブで LAN の設定を選択します（図 5 を参照）。

図 5. Windows のデフォルトのプロキシ設定の表示




各種のプロキシが存在し、次のようなコンピュータ･ネットワーク設定が可能です。

- プロキシを使用しない設定：アプリケーションは直接ウェブにアクセスします（Windows のデフォルト設定）。
- ログイン/パスワードを設定していないプロキシ
- ログイン/パスワードを設定したプロキシ：インターネット･ブラウザを使用する際に、ログイン/パスワードの入力を促すダイアログ･ボックスが表示されます。
- ログイン / パスワードを設定したウェブ･プロキシ：インターネット･ブラウザを使用する際に、ログイン/パスワードの入力を促すウェブ･ページが表示されます。

必要に応じて、IT 管理者にプロキシに関する情報（プロキシの種類、HTTP アドレス、ポート）を問い合わせてください。

STM32CubeMX は、ウェブ･プロキシには対応していません。その場合は更新メカニズムを使用できないので、http://www.st.com/stm32cube からダウンロードした STM32Cube マイクロコントローラ･パッケージを手動でリポジトリにコピーする必要があります。その手順は以下のとおりです。

1. http://www.st.com/stm32cube にアクセスして、該当の STM32Cube マイクロコントローラ･パッケージを［Associated Software］のセクションからダウンロードします。
2. その ZIP パッケージを STM32Cube のリポジトリに展開します。デフォルトのリポジトリ･フォルダの場所は、図 6 に示した **［Updater settings］** タブで確認します（別の場所または名前を使用するには、この設定の変更が必要になる場合があります）。

### 3.4.1 プロキシ･サーバのもとでの STM32CubeMX の実行

プロキシが全面的な SSL インスペクションを実装している場合、プロキシ証明書を使用するように STM32CubeMX を設定する必要があります。

- Windows の場合：

  一般的には、Windows の証明書リストを使用することになります。

  a) STM32CubeMX の実行可能ファイルの実行には、その他の設定は一切不要です（STM32CubeMX は、Windows の証明書リストを使用するように設定済みです）。

  b) コマンドラインから STM32CubeMX を実行できるように、コマンドラインを次のように調整する必要があります。

  ```
  <STM32CubeMX install path>\jre\bin\java -
  Djavax.net.ssl.trustStoreType=WINDOWS-ROOT -jar STM32CubeMX.exe
  ```

- Mac と Linux の場合、および Windows で証明書ストアにプロキシ証明書が存在しない場合、証明書を手動でインポートする必要があります。そのためには、次のようにコマンド･プロンプトから keytool を使用します。

  ```
  $ cd <STM32CubeMX のインストール･ディレクトリ>/jre
  $ bin/keytool -importcert -alias <使用する証明書のエイリアス名>
  -keystore lib/
  security/cacerts -file  <使用するプロキシ証明書ファイルへのパス>.crt
  ```

  パスワードの要求に対して changeit を入力します。

  証明書を信頼するかどうかに対して yes を入力します。

  Windows の場合のみ、ファイル <STM32CubeMX のインストール･ディレクトリ>/STM32CubeMX.l4j.ini で次の行を削除します。

  ```
  -Djavax.net.ssl.trustStoreType=WINDOWS-ROOT
  ```

### 3.4.2 アップデータの設定

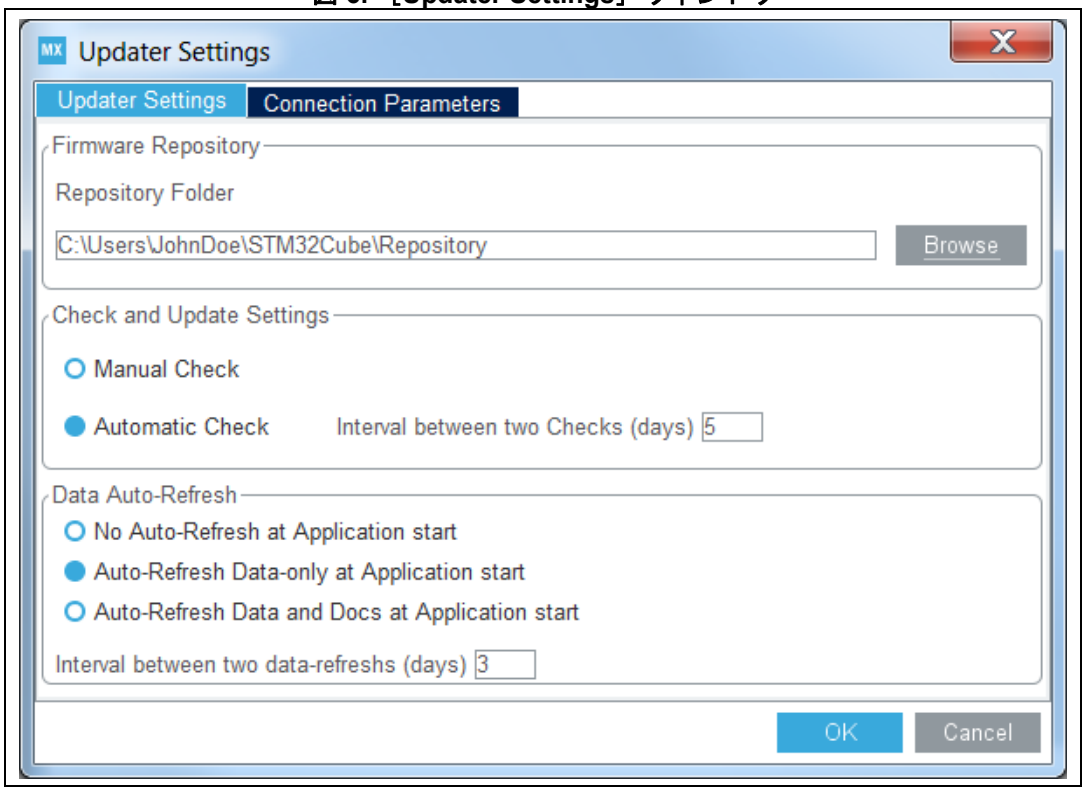
STM32Cube のライブラリ･パッケージの新規インストールまたは更新では、アップデータ･ツールを次の手順で設定する必要があります。

1. **[Help]** → **[Updater Settings]** を選択して **[Updater Settings]** ウィンドウを開きます。
2. **[Updater Settings]** タブで以下の手順を実行します（図 6 を参照）。

   a) ダウンロードしたパッケージの保存先とするリポジトリのフォルダを指定します。

   b) 更新の自動確認を有効または無効にします。

図 6. ［Updater Settings］ウィンドウ




3. **［Connection Parameters］**タブで、使用するネットワークの設定に適したプロキシ･サーバの設定を指定します。以下のプロキシのタイプのいずれかを選択できます（図 7 を参照）。
   - プロキシなし
   - システムのプロキシ･パラメータを使用：

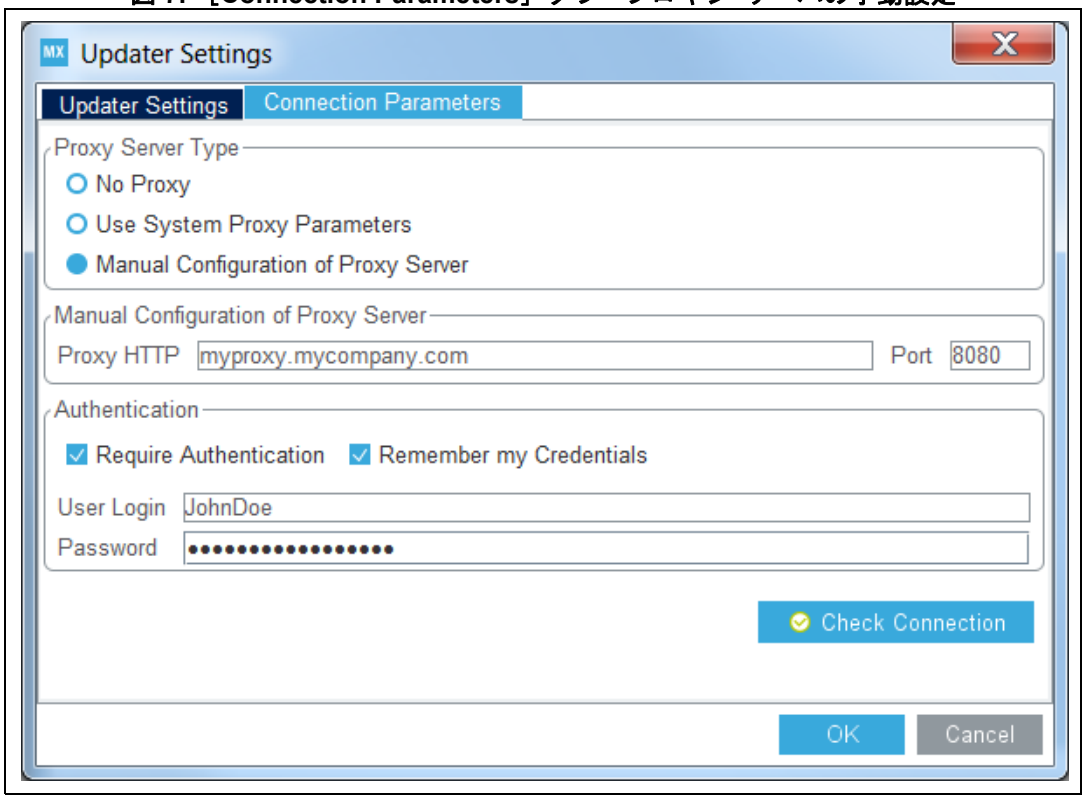
     Windows の場合、プロキシのパラメータは PC のシステム設定から取得されます。

     ログインとパスワードを設定していないプロキシ･サーバを使用する場合は［Require Authentication］のチェックを外します。
   - プロキシ･サーバの手動設定

     プロキシ･サーバの HTTP アドレスとポート番号を入力します。ログイン/パスワード情報を入力します。ログイン/パスワードを設定しないプロキシ･サーバを使用する場合は［Require Authentication］のチェックを外します。
4. STM32CubeMX で暗号化したログイン/パスワード情報がファイルに保存されないようにするには、**［Remember my Credentials］**のチェックを外します。この設定にすると、STM32CubeMX を起動するたびにログイン/パスワード情報を入力する必要があります。
5. **［Check Connection］**ボタンをクリックして、接続が正常に動作するかどうかを検証します。緑色のチェック･マーク 

が表示されれば、接続は正常に機能しています。

図 7. ［Connection Parameters］タブ - プロキシ･サーバの手動設定




6. ［Help］→［Install New Libraries］サブメニューを選択して、インストール可能なパッケージの一覧から目的のパッケージを選択します。
7. ツールの更新を手動で確認するように設定している場合は、［Help］→［Check for Updates］を選択して、インストール可能な新しいツールのバージョンやファームウェア･ライブラリのパッチがあるかどうかを確認します。

### 3.4.3 STM32 マイクロコントローラ･パッケージのインストール

新しい STM32 マイクロコントローラ･パッケージをダウンロードするには、以下のステップに従います。

1. **［Help］→［Manage embedded software packages］**を選択して、**［Embedded Software Packages Manager］**を開きます（図 8 参照）。または、［Home］ページの［Install/Remove］ボタンを使用します。

   展開ボタンと縮小ボタン  を使用すると、パッケージのリストをそれぞれ展開表示または縮小表示できます。

   STM32CubeMX を使用してインストールした場合、ダウンロードできるすべてのパッケージとそのバージョン番号が表示されます。また、現在 PC にパッケージがインストールされていればそのバージョン番号、および www.st.com から入手可能な最新バージョンも表示されます。

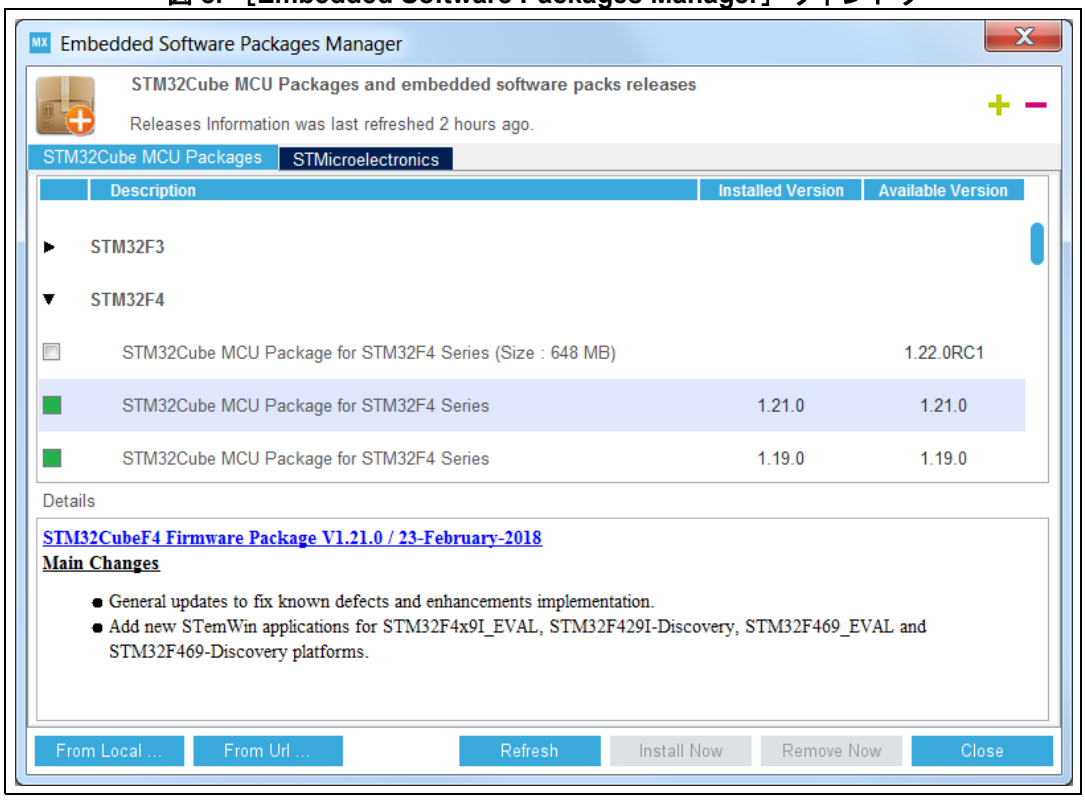
   インターネットにアクセスできない場合は［From Local...］を選択し、目的のダウンロード済み STM32Cube マイクロコントローラ･パッケージの ZIP ファイルを選択します。そのファイルに対して整合性チェックが実行され、STM32CubeMX でそのファイルを問題なく扱うことができるかどうかが確認されます。

   インストールされているパッケージのバージョンが、www.st.com から入手可能な最新バージョンと一致している場合は、リストの中でパッケージに緑色のマークが表示されます。

2. 選択するパッケージのチェックボックスをチェックし、[Install Now]をクリックしてダウンロードを開始します。

その例を図 8 に示します。

**図 8. ［Embedded Software Packages Manager］ウィンドウ**



### 3.4.4 STM32 マイクロコントローラ･パッケージのパッチのインストール

セクション 3.4.3： で説明した手順に従い、STM32 マイクロコントローラ･パッケージのパッチをダウンロードします。

STM32Cube_FW_F7_1.4.1 のようなライブラリのパッチは、バージョン番号の 3 桁目がゼロではないことからパッチであることが容易にわかるようになっています（例：バージョン 1.4.1 の '1'）。

パッチは、ライブラリ全体を収めたパッケージではなく、更新が必要なライブラリ･ファイルのみの集合です。パッチが適用されたファイルは、元のパッケージよりも前に表示されます（例：STM32Cube_FW_F7_1.4.1 は、STM32Cube_FW_F7_1.4.0 パッケージを補完するパッケージです）。

バージョン 4.17 より前の STM32CubeMX では、パッチが元のベースライン･ディレクトリにコピーされます（例：パッチ･ファイル STM32Cube_FW_F7_V1.4.1 は、STM32Cube_FW_F7_V1.4.0 ディレクトリにコピーされます）。

STM32CubeMX のバージョン 4.17 以降は、パッチをダウンロードすると専用のディレクトリが作成されます。たとえば、パッチ･ファイル STM32Cube_FW_F7_V1.4.1 をダウンロードすると、STM32Cube_FW_F7_V1.4.1 ディレクトリが作成されます。このディレクトリには、元の STM32Cube_FW_F7_V1.4.0 のベースラインと STM32Cube_FW_F7_V1.4.1 パッケージに収録されたパッチ･ファイルが保存されます。

このような仕組みがあることから、あるプロジェクトではパッチを適用していない元のパッケージを使用し、別のプロジェクトではパッチでアップグレードしたバージョンを使用するといった使い分けが可能です。

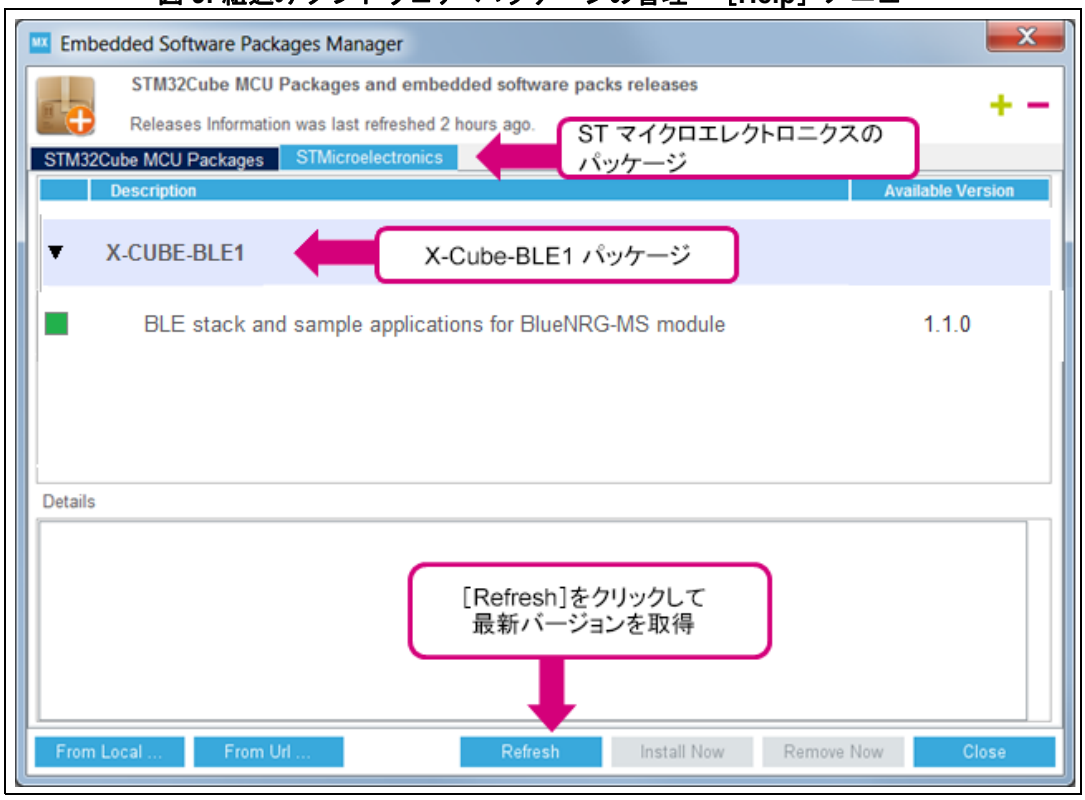
### 3.4.5 組込みソフトウェア･パッケージのインストール

リリース 4.24 以降の STM32CubeMX では、Arm® Keil™ CMSIS-Pack フォーマット（.pack）で用意された 3rd パーティ製組込みソフトウェア･パッケージを選択できるようになりました。パッケージの内容は、パッケージ記述ファイル（.pdsc）に記述されています。http://www.keil.com で参考ドキュメントを入手可能です。

1. **［Help］→［Manage embedded software packages］**を選択して、**［New Libraries Manager］ウィンドウ**を開きます（図 9 参照）。あるいは、［Home］ページまたはプロジェクトの **［Pinout & Configuration］**ビュー（**［Software Packs］→［Manage Software Packs］**を選択）にある［Install/Remove］ボタンを使用します。

   展開/縮小ボタン  を使用すると、パッケージのリストを展開表示または縮小表示できます。

図 9. 組込みソフトウェア･パッケージの管理 - ［Help］メニュー




2. **[From Local...]** ボタンをクリックしてコンピュータのファイル･システムを参照し、組込みソフトウェア･パッケージを選択します。STM32Cube マイクロコントローラ･パッケージは ZIP アーカイブ、組込みソフトウェア･パッケージは.pack アーカイブとしてそれぞれ提供されます。

   この操作は、以下の場合に必要になります。

   – インターネットにはアクセスできないものの、組込みソフトウェア･パッケージがコンピュータのローカル･ストレージに存在する場合。

   – 組込みソフトウェア･パッケージが非公開であることから、インターネットで入手できない場合。そのようなパッケージの場合、STM32CubeMX では更新を検出できず、その適用を促すこともできません。

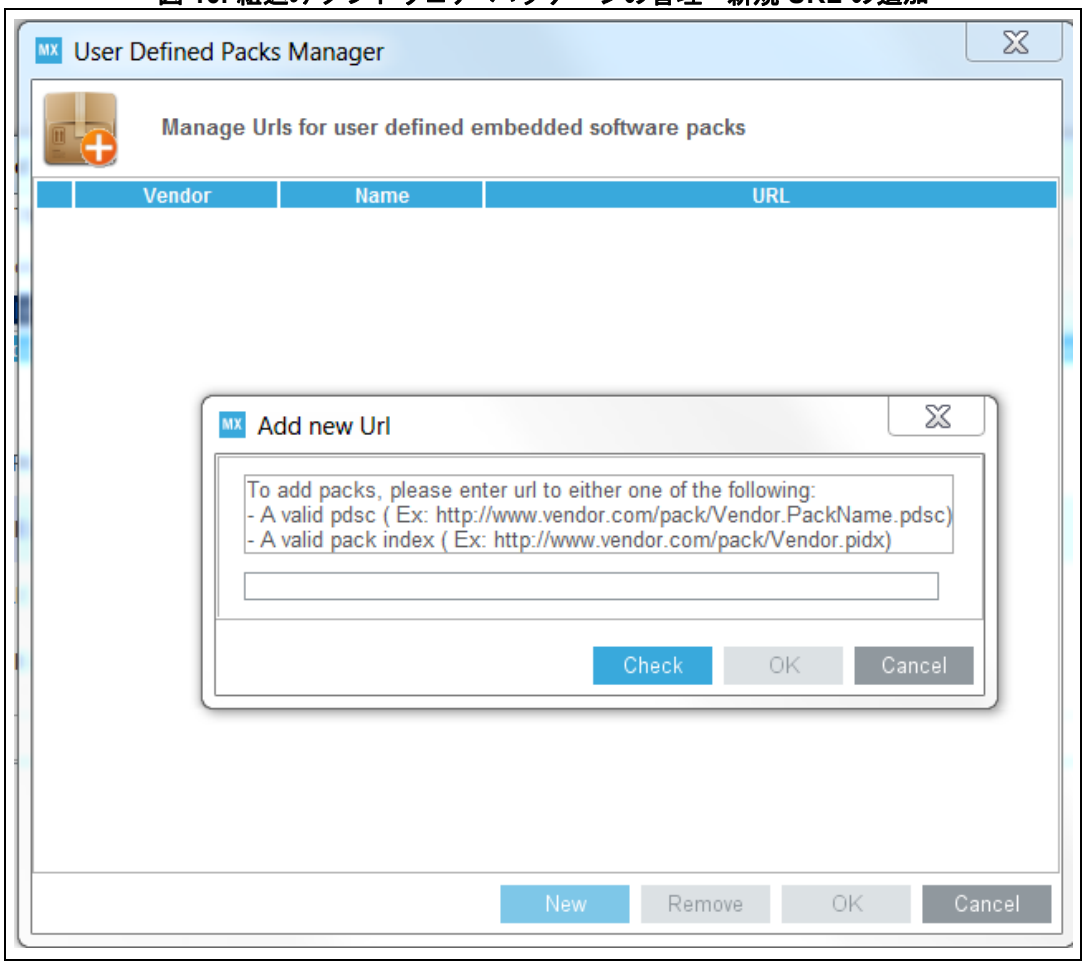
3. **[From URL...]** ボタンをクリックして、パッケージの .pdsc ファイルまたはベンダのパッケージ･インデックス（.pidx）のダウンロード元であるインターネット上の場所を指定します。

   次の手順に従います。

   a) **[From URL...]** を選択して **[New]** をクリックします（図 10 を参照）。

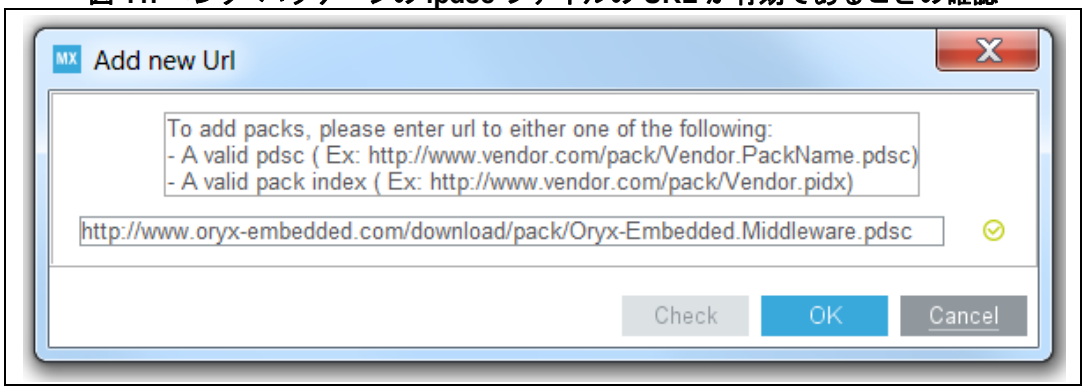
   b) .pdsc ファイルの URL を指定します。たとえば、Oryx 組込みミドルウェア･パッケージの URLは、https://www.oryx-embedded.com/download/pack/Oryx-Embedded.Middleware.pdscです（図 11 を参照）。

図 10. 組込みソフトウェア･パッケージの管理 - 新規 URL の追加




c) **[Check]** ボタンをクリックして、指定した URL が有効であることを確認します（図 11 参照）。

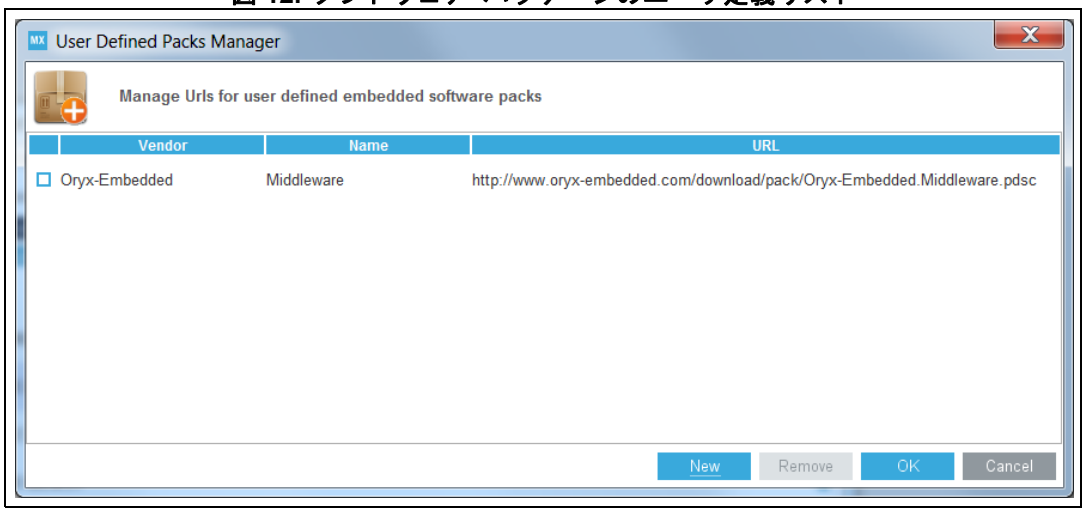
図 11. ベンダ･パッケージの .pdsc ファイルの URL が有効であることの確認



d) **[OK]** をクリックします。パッケージの pdsc 情報がユーザ定義パッケージ･リストに表示されるようになります（図 12 を参照）。

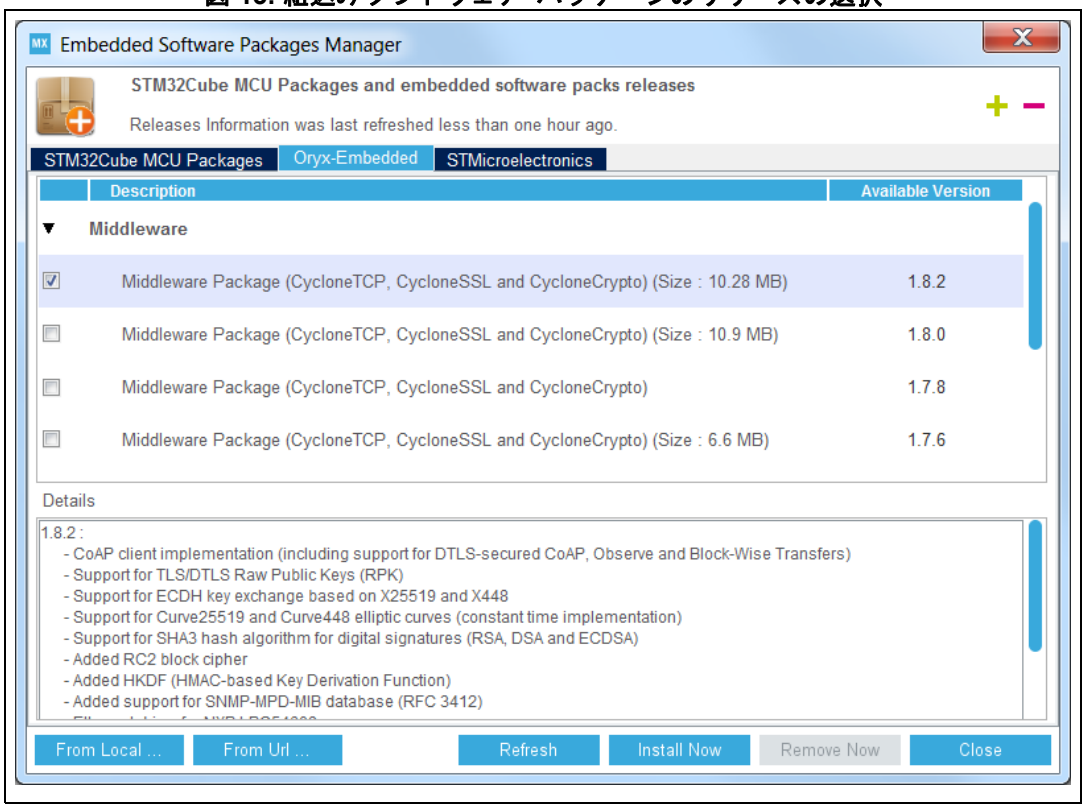
このリストから URL を削除するには、URL のチェックボックスを選択して **[Remove]** をクリックします。

**図 12. ソフトウェア･パッケージのユーザ定義リスト**




e) **[OK]** をクリックしてウィンドウを閉じ、pdsc 情報の取得を開始します。正常に完了すると、インストール可能なライブラリのリストに、利用可能なパッケージのバージョンが表示されます。該当のチェックボックスを使用して特定のリリースを選択します。

**図 13. 組込みソフトウェア･パッケージのリリースの選択**



f) **[Install Now]** をクリックして、ソフトウェア･パッケージのダウンロードを開始します。インストールの進捗状況を示す進捗バーが表示されます。パッケージが使用許諾契約を伴う場合、その契約へのユーザの同意を求めるポップアップ･ウィンドウが表示されます（図 14 を参照）。インストールが正常に終了すると、そのパッケージのチェックボックスが緑色に変化します（図 15 を参照）。

これで、このソフトウェア･パッケージのソフトウェア･コンポーネントをプロジェクトに追加できます。

**図 14. ライセンス契約への同意**

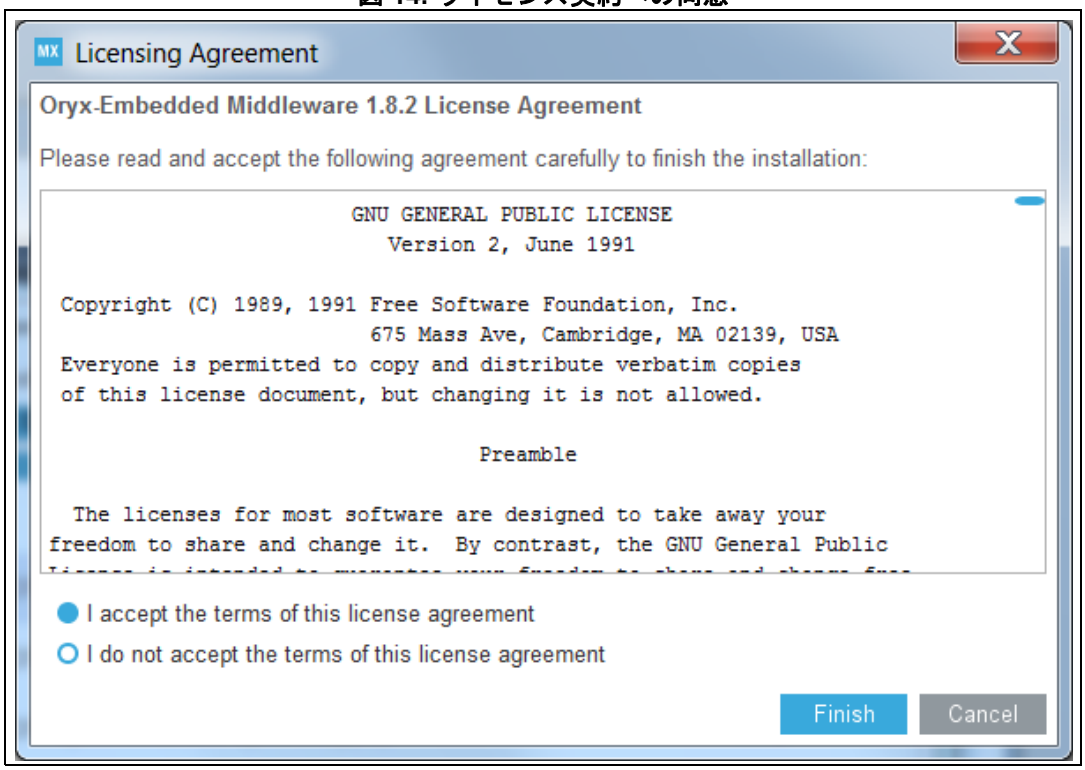
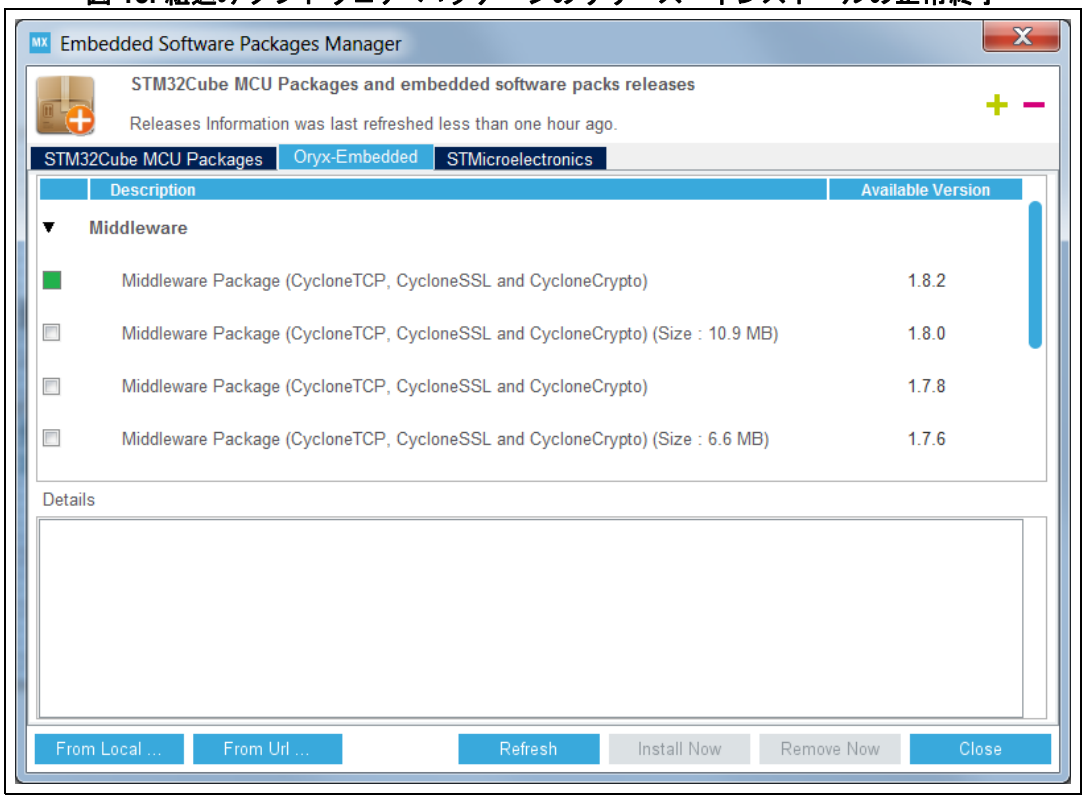


図 15. 組込みソフトウェア･パッケージのリリース - インストールの正常終了




### 3.4.6 インストール済みの組込みソフトウェア･パッケージの削除

リポジトリの古いバージョンのライブラリをクリーンアップして空きディスク領域を広げるには、次の手順に従います（図の16 ~ 18 参照）。

1. **[Help]→[Manage embedded software packages]**を選択して、**[Embedded Software Packages Manager]** を開きます。または、［Home］ページの［Install/Remove］ボタンを使用します。
2. 緑色のチェックボックスをクリックして、stm32cube リポジトリに存在するパッケージを選択します。
3. **[Remove Now]**ボタンをクリックして削除を確定します。進捗ウィンドウが開き、削除の状況が表示されます。

図 16. ライブラリの削除

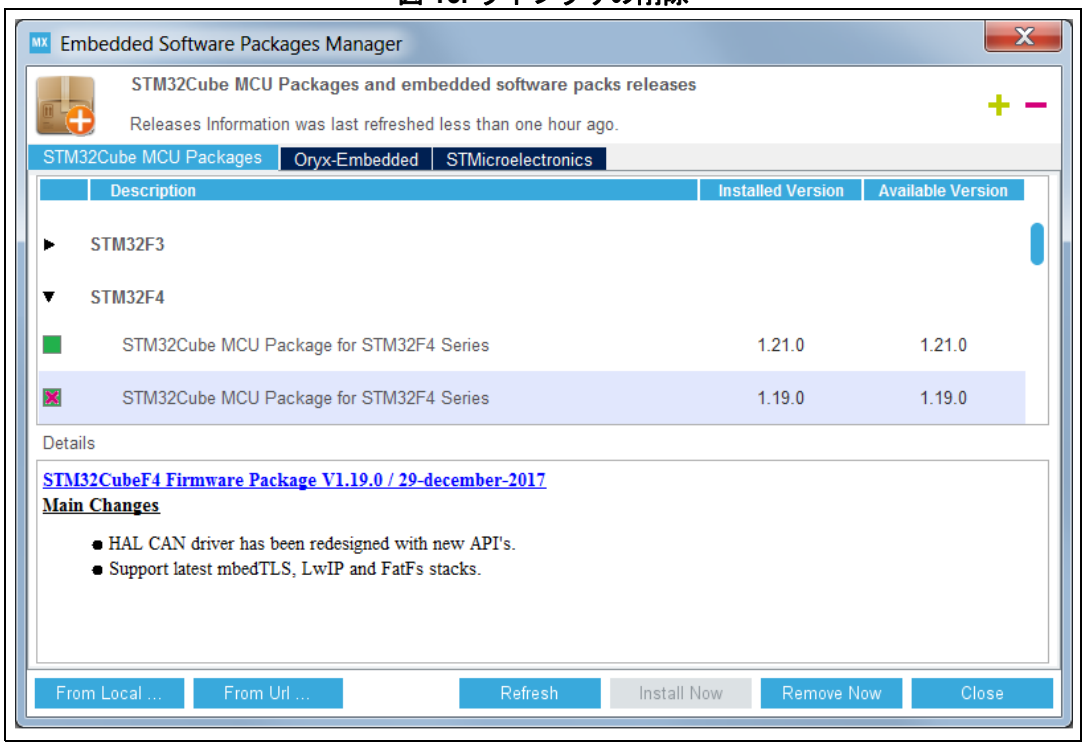



図 17. ライブラリ削除の確認メッセージ

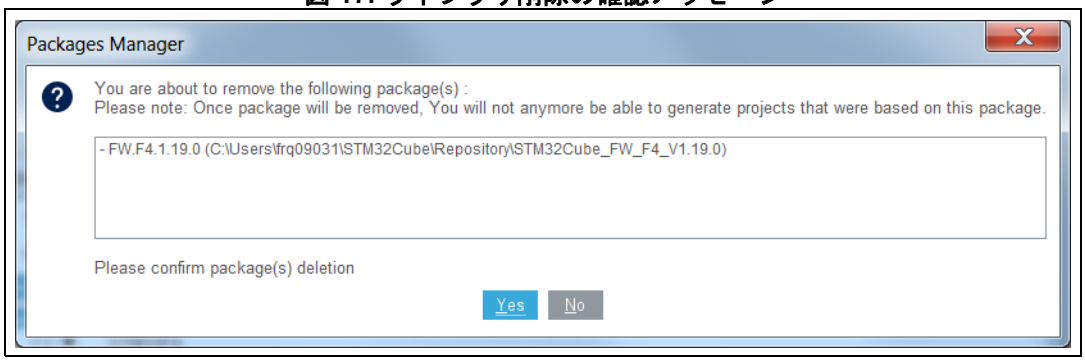
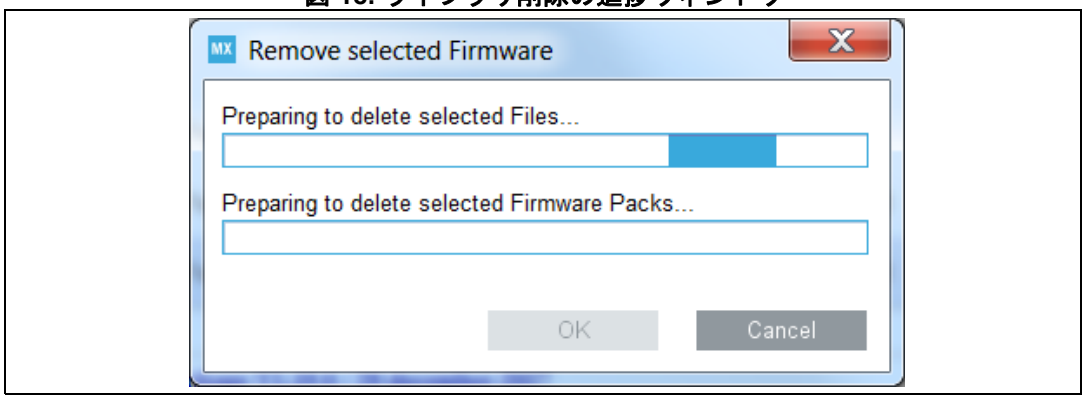



図 18. ライブラリ削除の進捗ウィンドウ



### 3.4.7 更新の確認

STM32CubeMX では、現在インストールされているバージョンの <1429/> の更新またはリポジトリ・フォルダにインストールされている組込みソフトウェア・パッケージの更新があるかどうかを確認できます（図 19）。

自動的に更新を確認するようにアップデータを設定している場合は、更新が用意されているかどうかが定期的に確認されます。

この自動的な確認を［Updater Settings］ウィンドウで無効にしている場合は、ユーザが手動で更新の有無を確認できます。

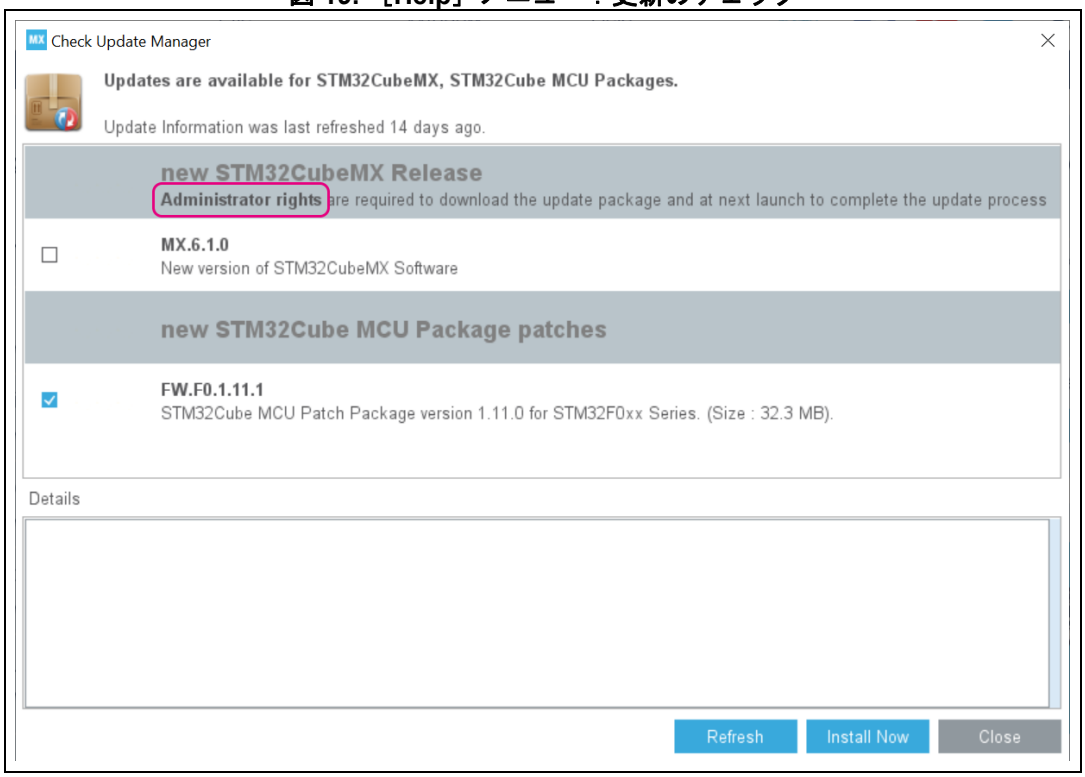
1. **［Update Manager］**ウィンドウを開くアイコンをクリックするか、**［Help］→［Check for Updates］**を選択します。現在のインストール環境で利用可能なすべての更新が一覧表示されます。
2. チェックボックスをチェックしてパッケージを選択し、［Install Now］をクリックして更新をダウンロードします。

---

**警告：** **STM32CubeMX の自己更新を実行する場合は、自己更新パッケージをダウンロードするとき、および下記の更新プロセスを完了する STM32CubeMX を起動するときに管理者権限が必要です。**
**1. 管理者アカウントで STM32CubeMX を起動します。**
**2.［Help］→［Check for updates］メニューに移動し、MX 更新パッケージを選択して、［Install now］をクリックするとダウンロードが始まります。**
**3. 管理者アカウントで STM32CubeMX を再起動して、更新プロセスを完了します。**

---

**図 19.［Help］メニュー：更新のチェック**



# 4 STM32CubeMXのユーザ･インタフェース

STM32CubeMX のユーザ･インタフェースには、次の 3 つの主要ビューがあり、それぞれに次の各ページ間の移動に便利なブレッドクラムが表示されます。

1. **［Home］**ページ
2. **［New Project］**ウィンドウ
3. プロジェクト･ページ

各ビューには、1 回のクリックで操作や設定選択が可能なパネル、ボタン、メニューがあります。

この後のセクションで、このユーザ･インタフェースの詳細を説明します。

C コードの生成では、さまざまな設定ビューに順不同で移動できますが、以下の順序に従うことをお勧めします。

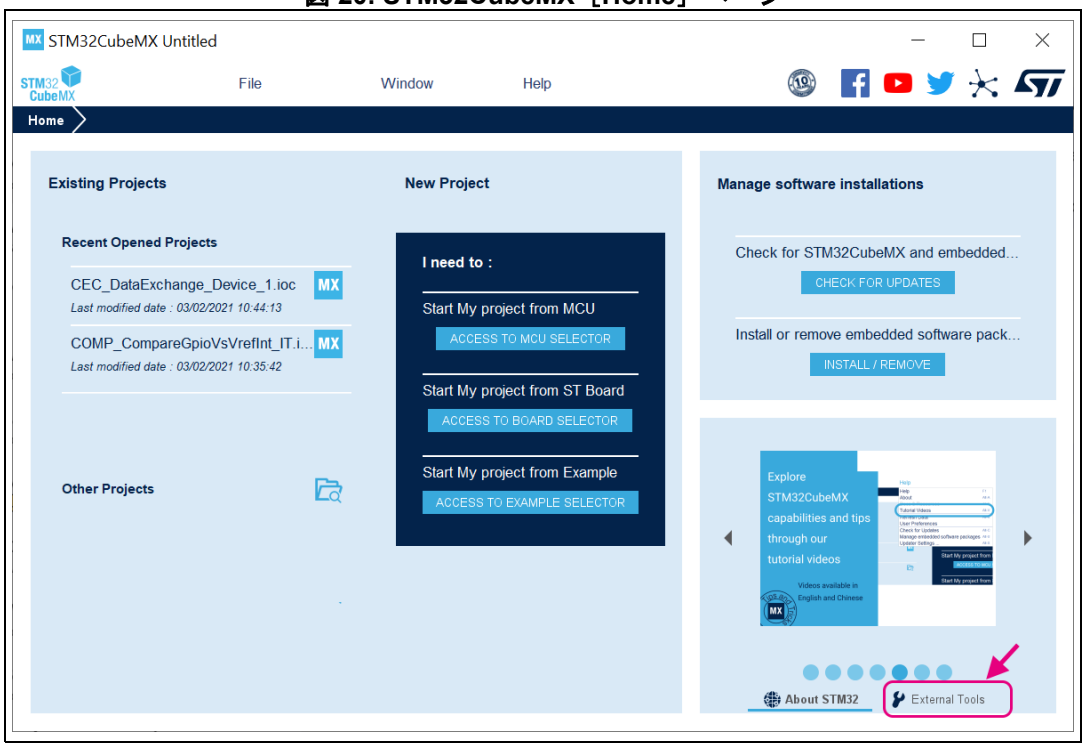
1. **［Project Manager］**ビューでプロジェクトを設定します。
2. **［Pinout & Configuration］**ビューの**［Mode］**パネルで、RCCペリフェラルの設定として外部クロックとマスタ出力クロックを有効にします。アプリケーションに関連するオーディオ入力クロックも、必要に応じて有効にします。これにより、**［Clock configuration］**ビューに、別の関連オプションが自動的に表示されます（図 97 参照）。つづいて、アプリケーションに関連する機能（ペリフェラル、ミドルウェア）とそれらの動作モードを選択します。
3. 必要に応じて［Clock Configuration］ビューでクロック･ツリー設定を調整します。
4. **［Pinout & Configuration］**ビューの［Configuration］パネルで、ペリフェラルの初期化に必要なパラメータとミドルウェアの動作モードを設定します。
5. 

をクリックして、初期化 C コードを生成します。

## 4.1 ［Home］ページ

［Home］ページは、STM32CubeMX を起動すると最初に表示されるウィンドウです（図 20 参照）。このウィンドウを閉じると、アプリケーションが終了します。このページには、いくつかの最上位メニューへのショートカット、STM32 の最新ニュースを表示する画像カルーセル、ソーシャル･ネットワーク･サイトや外部ツールへのリンクが表示されます。最上位メニューとソーシャル･ネットワークへのリンクには、この手順以降のプロジェクト･ページに移動した後も引き続きアクセスできます。

**図 20. STM32CubeMX［Home］ページ**



### 4.1.1 ［File］メニュー

［File］メニューとショートカットの説明については 表 2 を参照してください。

**表 2. ［Home］ページのショートカット**

| 名前<br>キーボード<br>ショートカット | 説明 | ［Home］ページのショートカット |
|---|---|---|
| **[New Project...]**<br>*Ctrl-N* | サポートされているすべてのマイクロコントローラおよび ST マイクロエレクトロニクスのボード製品群が表示される新規プロジェクトのウィンドウを開きます。ここでマイクロコントローラまたはボードを選択します(1)。 | ボードを指定して新規プロジェクトを作成する場合は、次のショートカットをクリックします。<br>  <br>マイクロコントローラを指定して新規プロジェクトを作成する場合は、次のショートカットをクリックします。<br>  |
| **[Load Project...]**<br>*Ctrl-L* | STM32CubeMX の .ioc 設定ファイルを選択して、既存の STM32CubeMX プロジェクト設定をロードします（注意：参照）。 | [Other Projects]の下の参照アイコン  をクリックします。 |
| **[Import Project...]**<br>*Ctrl-I* | インポートする設定ファイルとインポート設定を選択する新規ウィンドウが開きます。空のマイクロコントローラ設定からプロジェクトを開始した場合にのみ、このインポートを実行できます。それ以外の場合、このメニューは無効になっています(2)。 | なし |
| **[Save Project]**<br>*Ctrl-S* | 現在のプロジェクト設定（ピン配置、クロック･ツリー、ペリフェラル、ミドルウェア、消費電力計算機能）を新規プロジェクトとして保存します。<br>この操作により、定義したプロジェクト設定に基づく .ioc ファイルを含むプロジェクト･フォルダが作成されます。 | なし |
| **[Save Project as...]**<br>*Ctrl-A* | 現在のプロジェクトを保存します。 | なし |
| **[Close Project]**<br>*Ctrl-C* | 現在のプロジェクトを閉じ、［Home］ページに戻ります。 | なし |
| **[Recent Projects]**<br>なし | 最近保存した 5 件のプロジェクトが一覧表示されます。 | **[Recent Project]** の下のプロジェクト名の横に表示される  アイコンをクリックします。 |
| **[Generate Report]**<br>*Ctrl-R* | プロジェクトの現在の設定を 2 つのドキュメント（pdf とテキスト･フォーマット）に保存します。 | なし |
| **[Exit]**<br>*Ctrl-X* | 必要に応じてプロジェクトの保存を促したうえで、アプリケーションを終了します。 | ウィンドウを閉じてアプリケーションを終了するには  をクリックします。 |

1. **[New project...]** について：この段階でエラー･メッセージが表示されないように、インターネット接続が使用可能であること（［Help］→［Updater Settings］メニューの［Connection Parameters］タブ）、またはデータの自動リフレッシュ設定が［No Auto-Refresh at application start］に設定されていることを確認してください（［Help］→［Updater Settings］メニューの［Updater Settings］タブ）。
2. **[Import Project...]** について：インポートの競合の確認によって検出された警告またはエラーがステータス･ウィンドウに表示されます。その場合はインポートを取り消すことができます。

**注意：** **プロジェクトのロードについて：STM32CubeMX では、古いバージョンのツールで作成されたプロジェクトを検出できます。その場合は、最新バージョンの STM32CubeMX データベースと STM32Cube ファームウェアに移行するか、そのまま継続するかの選択が求められます。**
**バージョン 4.17 よりも前の STM32CubeMX では、［Continue］をクリックした場合でも、データベースは、プロジェクトで使用する SMT32Cube ファームウェア･バージョンと「互換性がある」最新バージョンにアップグレードされます。**
**バージョン 4.17 以降では、［Continue］をクリックすると、プロジェクトの作成に使用されたデータベースがそのまま使用されます。必要とするバージョンのデータベースがコンピュータ上に存在しない場合は、そのバージョンが自動的にダウンロードされます。**
**新しいバージョンの STM32CubeMX にアップグレードする場合は、必ず既存のプロジェクトのバックアップを作成したうえで、新規プロジェクトを読み込むようにします（ユーザ･コードを使用したプロジェクトで特に重要です）。**

### 4.1.2 ［Window］メニューと［Outputs］タブ

**［Window］**メニューでは**［Outputs］**機能を使用できます。

**表 3. ［Window］メニュー**

| 名前 | 説明 |
|---|---|
| ［Outputs］ | **［Window］**メニューの**［Outputs］**を選択または選択解除すると、STM32CubeMX のプロジェクト･ページ下部で以下の[Output]タブが表示または非表示になります（図 21 参照）。<br>– **［MCUs Selection］**タブ：特定のファミリの中で、ユーザが最後にマイクロコントローラを選択したときに指定した検索基準（シリーズ、ペリフェラル、パッケージなど）に一致するすべてのマイクロコントローラが一覧表示されます(1)。<br>– **［Output］**タブ：ユーザが実行した操作の一覧と、それに伴って生成されたエラーや警告が表示されます（すべての操作を網羅した一覧ではありません。図 22 参照）。 |
| ［Font Size］ | STM32CubeMX のフォント･サイズ設定を変更できます。変更を有効にするには、STM32CubeMX を再起動する必要があります。 |

1. 現在の設定とは異なるマイクロコントローラをリストから選択すると、現在のプロジェクト設定がリセットされ、その新しいマイクロコントローラに切り換わります。この操作を続行する前に、続行の確認を促すメッセージが表示されます。

図 21. ［Window］メニュー

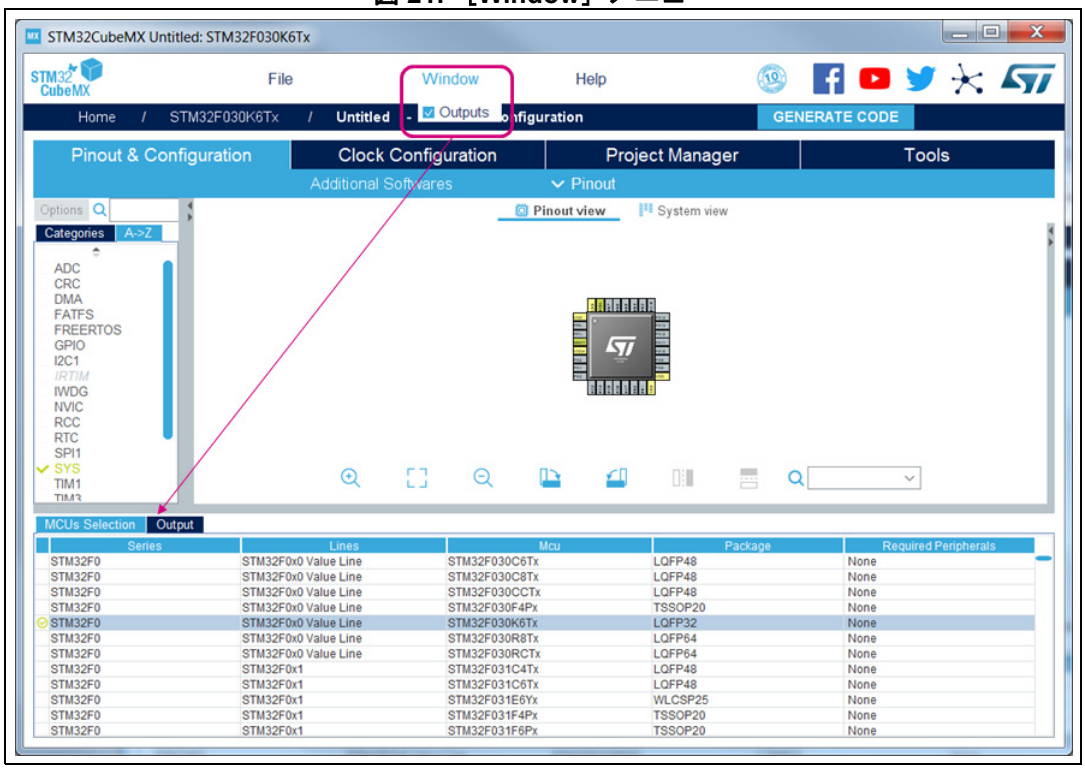
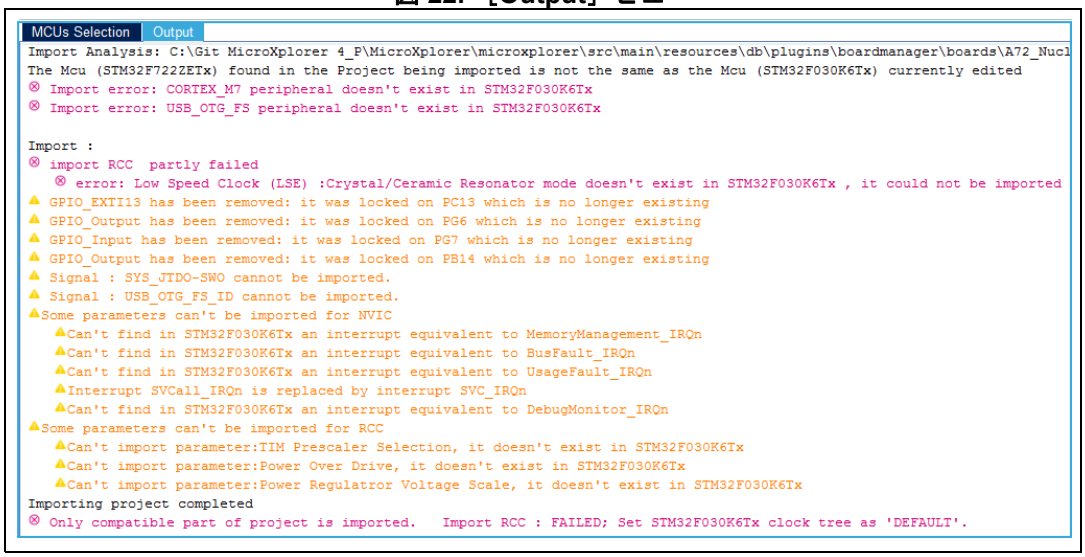



図 22. ［Output］ビュー



### 4.1.3 ［Help］メニュー

［Help］メニューとショートカットの説明については 表 4 を参照してください。

**表 4. ［Help］メニューのショートカット**

| 名前<br>キーボード<br>ショートカット | 説明 | ［Home］ページのショートカット |
|---|---|---|
| **［Help］**<br>*F1* | STM32CubeMX のユーザ･マニュアルを表示します。 | なし |
| **［About］**<br>*Alt-A* | バージョン情報を表示します。 | なし |
| **［Docs & Resources］**<br>*Alt-D* | 現在のプロジェクトで使用しているマイクロコントローラに用意されている正式ドキュメントを表示します。 | なし |
| **［Video Tutorials］**<br>*Alt-V* | ビデオ･チュートリアル･ブラウザを表示します。ここには推奨ビデオの一覧が表示され、1 回のクリックでビデオを再生できます。 | なし |
| **［Refresh Data］**<br>*Alt-R* | STM32CubeMX のデータベースを STM32 マイクロコントローラの最新情報（公式ドキュメントの説明とリスト）でリフレッシュすることを促すダイアログ･ウィンドウが表示されるほか、正式ドキュメントを一度にまとめてダウンロードできます。 | なし |
| **［Check for Updates］**<br>*Alt-C* | ダウンロード可能なソフトウェアおよびファームウェアのリリース更新を表示します。 | CHECK FOR UPDATES を クリックします。 |
| **［Manage embedded software packages］**<br>*Alt-U* | インストール可能な組込みソフトウェア･パッケージをすべて表示します。<br>チェックボックスが緑色のパッケージは、すでにリポジトリ･フォルダにインストール済みです（リポジトリ･フォルダの場所は［Help］→［Updater Settings］メニューで指定します）。 | INSTALL/REMOVE を クリックします。 |
| **［Updater Settings...］**<br>*Alt-S* | アップデータの設定ウィンドウを表示します。手動更新と自動更新、インターネット接続用プロキシ、ダウンロードしたソフトウェアやファームウェア･リリースを保存するリポジトリ･フォルダなどの設定が可能です。 | なし |
| **［User Preferences］** | 機能の使用状況に関する統計情報の収集を有効または無効にするユーザ設定ウィンドウを表示します。 | なし |

### 4.1.4 ソーシャル･プラットフォームへのリンク

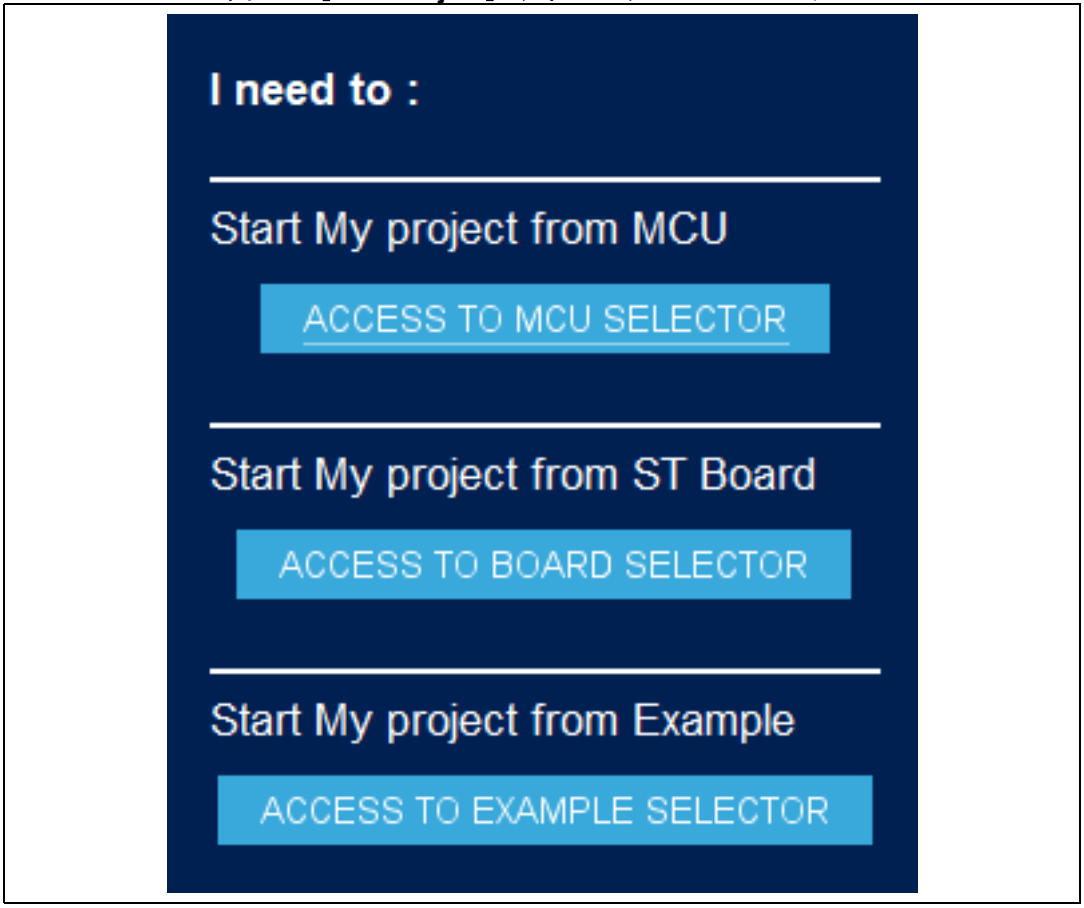
Facebook™、Twitter™、STM32 の YouTube™ チャネルなどのよく知られたソーシャル･プラットフォームや ST Community にある開発者コミュニティに、STM32CubeMX のツールバーからアクセスできます（図 23 参照）。

**図 23. ソーシャル･プラットフォームへのリンク**

## 4.2 ［New Project］ウィンドウ

［New Project］ウィンドウには、［File］メニューからアクセスできるほか、［Home］ページのショートカットからは直接アクセスできます（図 24 参照）。

**図 24. ［New Project］ウィンドウのショートカット**




このウィンドウの主な目的は、ユーザのアプリケーションのニーズに最適なマイクロコントローラまたはボードの品名を STM32 のポートフォリオから選択すること、またはサンプル･プロジェクトを元にプロジェクトの作成を開始することです。

このウィンドウでは次の 3 つのタブを選択できます。

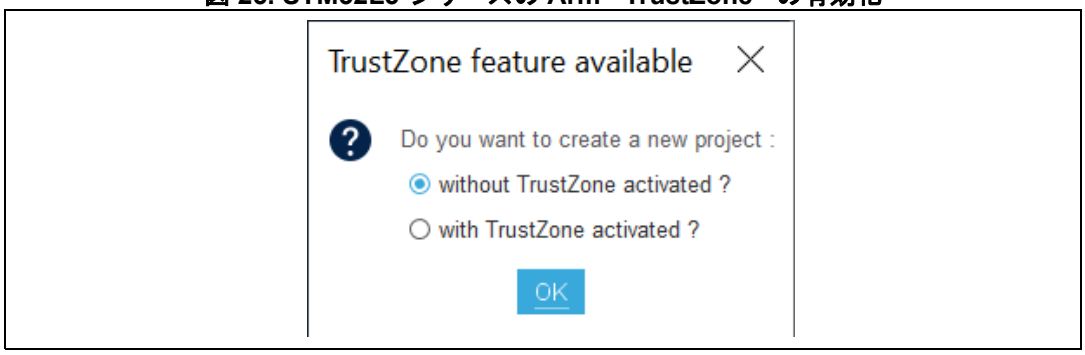
- **［MCU Selector］**タブ（ターゲット･プロセッサが一覧表示されます）
- **［Board Selector］**タブ（ST マイクロエレクトロニクスのボードが一覧表示されます）
- **［Example Selector］**タブ（サンプル･プロジェクトを参照し、開くことができます）

［New Project］ウィンドウには**［Cross Selector］**タブもあります（特定のマイクロコントローラと MPU の品名および選択基準に基づいて、STM32 のポートフォリオから最適な代替製品を探し出すことができます）。

STM32L5 シリーズの場合、Arm Cortex-M33 プロセッサとその Armv8-M 向け Arm® TrustZone® のセキュリティ機能に、ST が実装したセキュリティ機能が統合されています。STM32L5 マイクロコントローラまたはボードを選択すると、Arm® TrustZone®（ハードウェア･セキュリティ）をアクティブ化するかどうかを選択する必要があります（図 25 参照）。その選択に従ってプロジェクトが調整されます。

- Arm® TrustZone® をアクティブ化しない場合、ソリューションは他の STM32Lx シリーズと同じになります。
- Arm® TrustZone® をアクティブ化すると、プロジェクト設定と生成されるプロジェクトには、上記のセキュリティ機能に基づく特徴が得られます（本マニュアルの関連セクションを参照してください）。

**図 25. STM32L5 シリーズの Arm® TrustZone® の有効化**




これらの選択機能によって表示されるビューは次の手順で調整できます（図 26 参照）。

- 列を左クリックすると並べ替えが可能です。
- 列を追加または削除するには右クリックします。

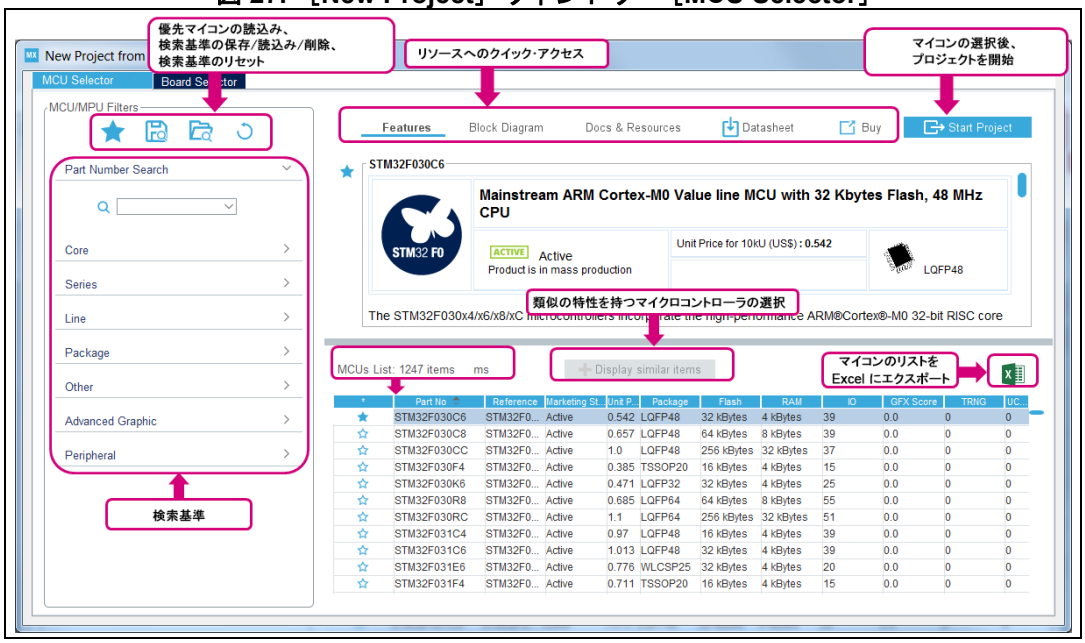
**図 26. 選択機能によって表示されるビューの調整**


## 4.2.1 ［MCU Selector］

### マイクロコントローラの選択

［MCU Selector］では、シリーズ、製品ライン、パッケージ、ペリフェラルなどの条件の組合せ、価格、メモリ容量、I/O 数などのマイクロコントローラの特性（図 27 参照）、およびマイクロコントローラのグラフィック機能によって一覧表示を絞り込むフィルタ機能を使用できます。

図 27. ［New Project］ウィンドウ - ［MCU Selector］



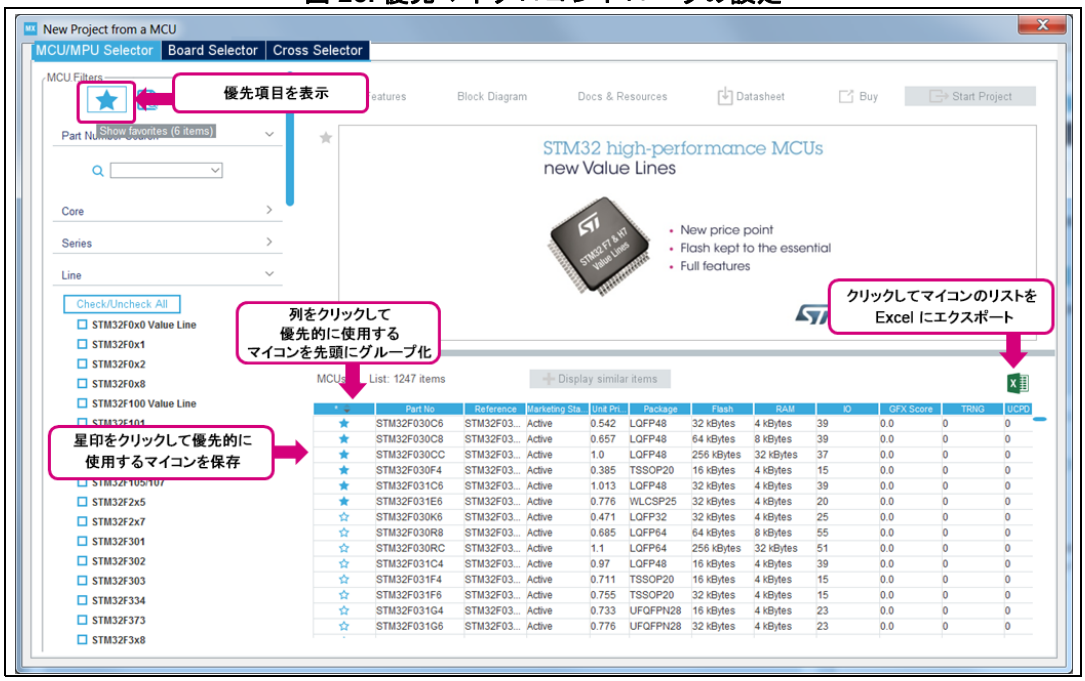

## Excel へのエクスポート機能

Export アイコンをクリックすると、マイクロコントローラの表の情報を Excel ファイルに保存できます。

## 優先マイクロコントローラの表示機能

マイクロコントローラのリストで アイコンをクリックすると、そのマイクロコントローラを優先マイクロコントローラとして登録できます（図 28 を参照）。

図 28. 優先マイクロコントローラの設定



### 指定の機能に近い機能を有するマイクロコントローラの表示

基準に一致したマイクロコントローラの数が 50 未満の場合、それ以外で指定の機能に近い機能を持つマイクロコントローラを一覧表示できます（図 29 を参照）。**[Display similar items]** ボタンをクリックすると、そのようなコントローラが表示されます（図 30 を参照）。デフォルトでは、最初に一致度の順、つづいて部品番号の順にマイクロコントローラが並べ替えられます。指定の機能に近い機能を持つマイクロコントローラ（一致度が 100% 未満）の行は灰色で表示され、基準に一致していないセルが濃い灰色でハイライトされます。

**図 29. ［New Project］ウィンドウ - 指定の機能に近い機能を有するマイクロコントローラの一覧**

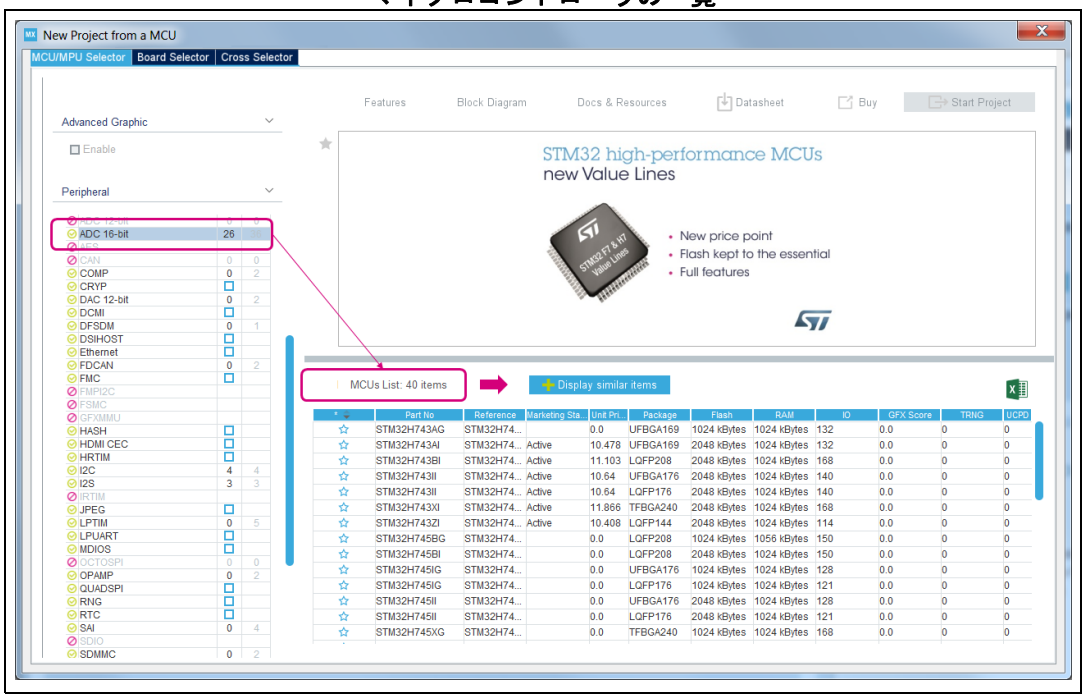
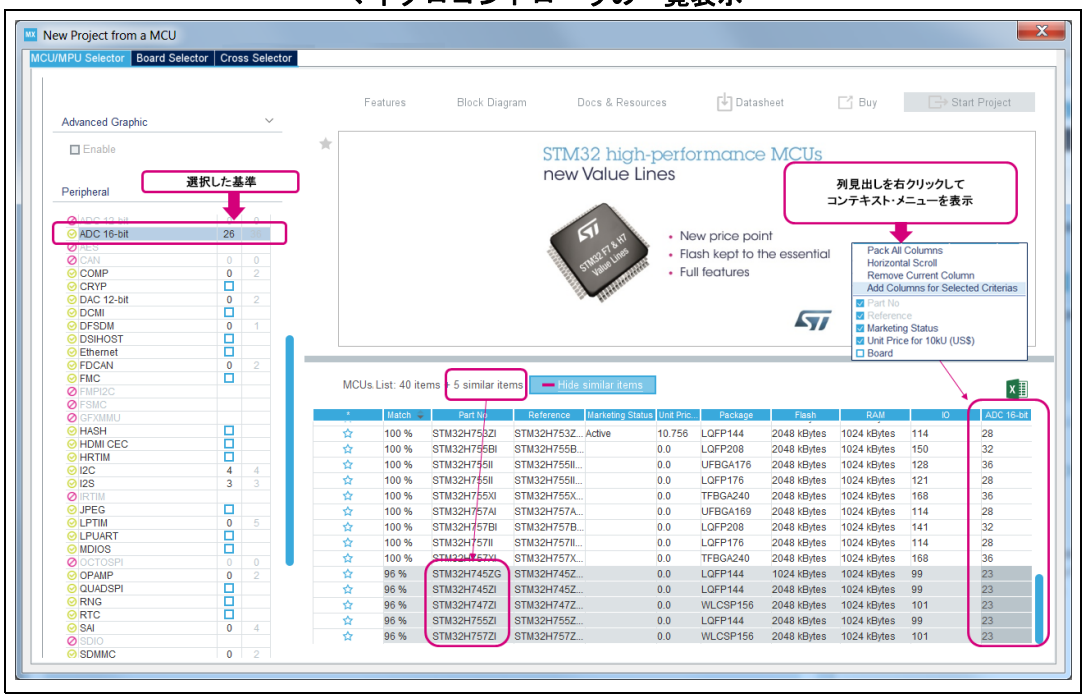


図 30. ［New Project］ウィンドウ - 指定の機能に近い機能を有する
マイクロコントローラの一覧表示




注： **一致度の値は、ユーザが選択した基準ごとに計算されます。以下に例を示します。**
**- CAN ペリフェラルの 4 件のインスタンスを扱うマイクロコントローラを要求した場合、3 件のインスタンスのみを扱うマイクロコントローラは、CAN のフィルタ基準に対する一致度が 75% になります。**
**- 最高価格のフィルタ基準を選択した場合、その最高価格をマイクロコントローラの実際の価格で除算した値が、そのマイクロコントローラの一致度になります。最低価格のフィルタ基準を選択した場合、マイクロコントローラの価格をその最低価格で除算した値が、そのマイクロコントローラの一致度になります。**
**最終的に、すべてのフィルタ基準に対する一致度の平均値が［Match］列に表示されます。**

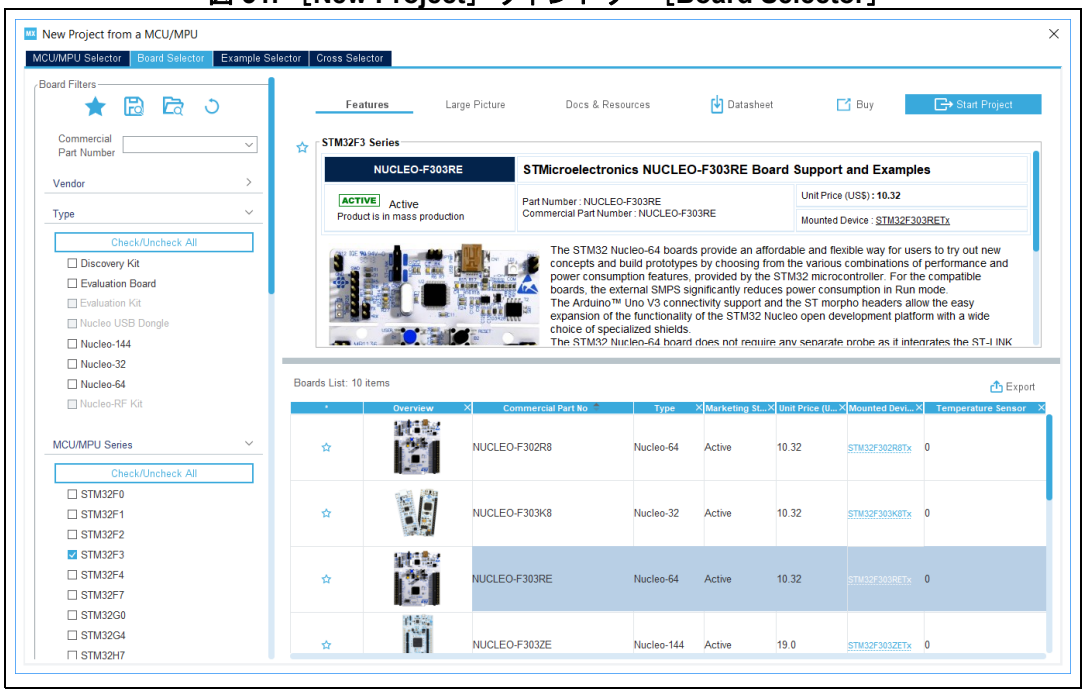
## 4.2.2 ［Board Selector］

**［Board Selector］**では、STM32 ボードのタイプ、シリーズ、ペリフェラルに基づいてボードを絞り込むことができます（図 31 参照）。デフォルトのボード設定のみが提示されます。ジャンパの再設定やはんだブリッジによって得られるボードの代替設定には対応していません。

ボードを選択すると、関連するマイクロコントローラの品名で **［Pinout］**ビューが初期化され、そのマイクロコントローラの LCD、各種ボタン、通信インタフェース、LED などのボード機能に対するピン割当てが表示されます。必要に応じて、デフォルトのペリフェラル･モードでビューを初期化することもできます。

ボード設定を選択すると、信号が「ロック済み」に変化し、STM32CubeMX の競合解決機能で自動的には移動できなくなります（ペリフェラル･モードの選択など、ペリフェラル･ツリーに対するユーザ操作では、ロック済み信号は移動しません）。この機能により、ユーザ設定のボードとの互換性を保持できます。

図 31. ［New Project］ウィンドウ - ［Board Selector］



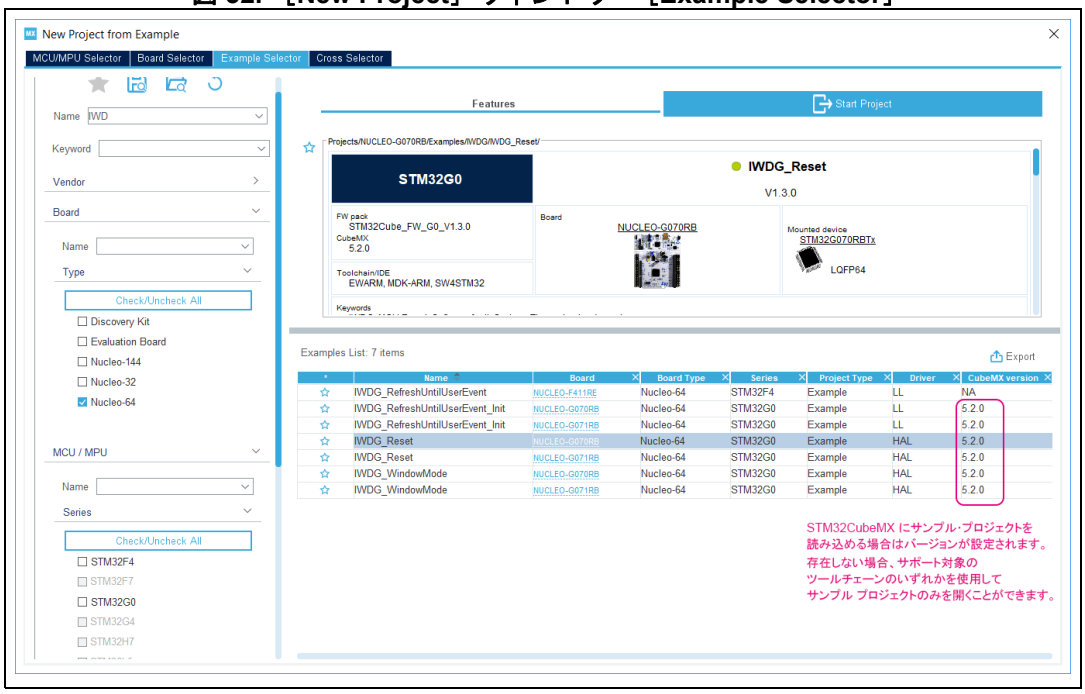

## 4.2.3 ［Example Selector］

［Example Selector］では、多数のサンプルを参照し、選択したサンプルを元に新規プロジェクトを開始できます。

**注：** **各サンプルは、特定のボード向けに設計されています。したがって、マイクロコントローラは、そのボードで使用できるものに限られます。**

フィルタ･パネルにより、ボードの特定のタイプ、シリーズ、ペリフェラル、ミドルウェア、その他の特性により、サンプルの一覧を絞り込むことができます（図 32 参照）。

図 32. ［New Project］ウィンドウ - ［Example Selector］



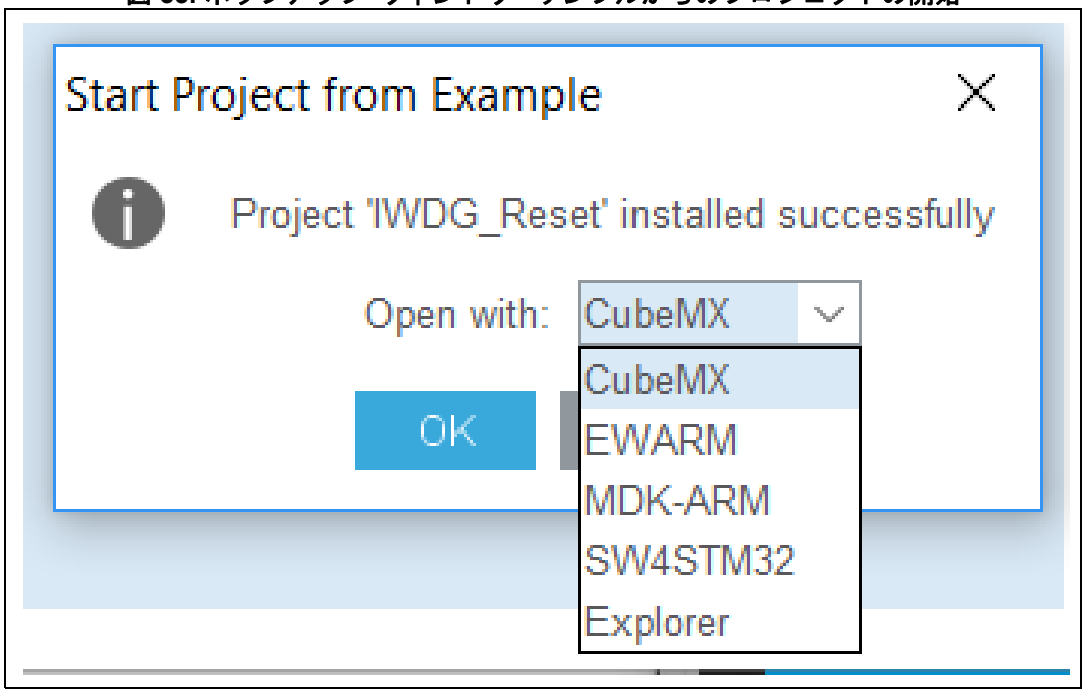

サンプルを選択して［Start project］をクリックすると、そのサンプルが STM32CubeMX によって新規プロジェクトとしてコピーされます（この段階でデフォルトの保存場所を変更できます）。

**警告：** **サンプルによっては、［Start Project］ボタンのほかに［Under Development］の警告アイコンが表示されるものがあります。このようなサンプルから作成したプロジェクトは機能しない場合があります（コンパイルできません）。この問題に対する修正を開発中です。**

新規作成したプロジェクトを開くには以下の方法があります（図 33 参照）。

- STM32CubeMX を使用する方法（CubeMX バージョンが設定された状態で一覧表示されるサンプルでのみ使用できます）
- ファイル･エクスプローラを使用する方法
- サポート対象のツールチェーンのいずれかを使用する方法（コンピュータにツールチェーンがインストール済みである場合）

図 33. ポップアップ･ウィンドウ - サンプルからのプロジェクトの開始




注： **サンプルに必要な STM32Cube マイクロコントローラ･パッケージがリポジトリに見つからない場合、STM32CubeMX によって自動的にダウンロード･プロセスが始まります。**

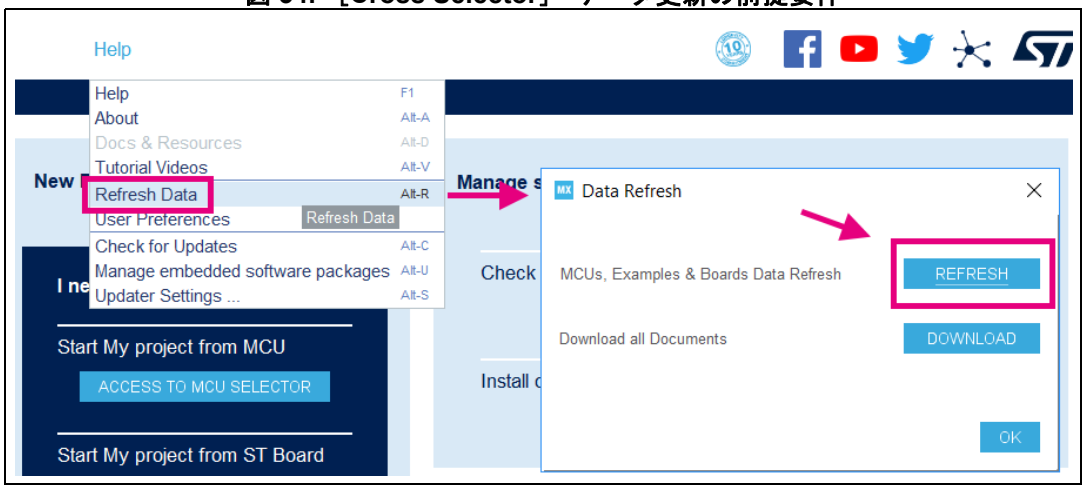
### 4.2.4 [Cross Selector]

#### 品名選択

[Cross Selector] では、現在使用中のマイクロコントローラまたは MPU（ST 製品または他の半導体ベンダの製品）に代わる最適な STM32 ポートフォリオの製品を見つけることができます。

この機能を使用するには、STM32CubeMX のデータが最新であることが必要です。[Help] メニューの [Refresh Data] を使用すれば確実に最新の状態を維持できます（図 34 参照）。

図 34. [Cross Selector] - データ更新の前提要件



［Home］ページの［Start my project from Cross Selector］セクションで［ACCESS TO CROSS SELECTOR］をクリックすると、［Cross Selector］タブに［New Project］ウィンドウが開きます。

2 つのドロップダウン･メニューにより、比較する製品のベンダと品名を選択できます（図 35 参照）。品名はその一部を入力するだけでかまいません。一致する製品の一覧が表示されます（図 36 参照）

**図 35. ［Cross Selector］- ベンダによる品名選択**

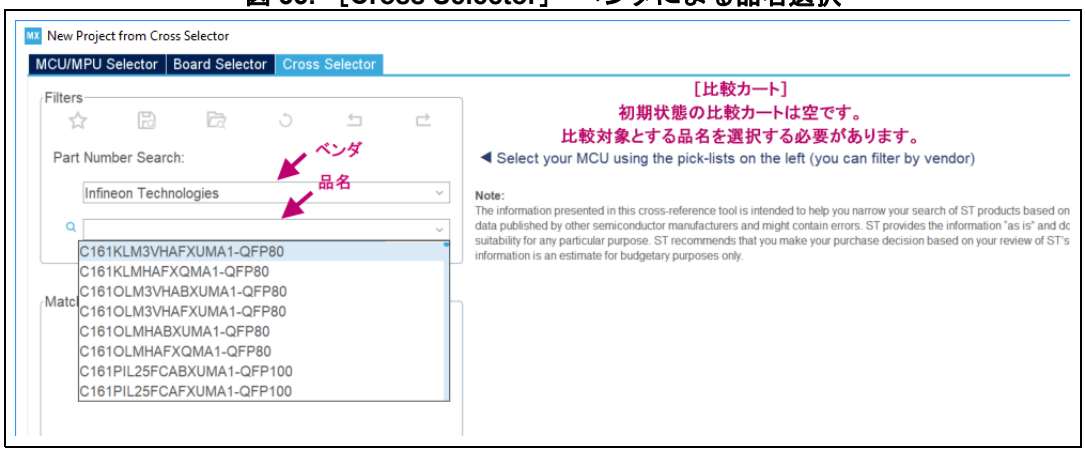


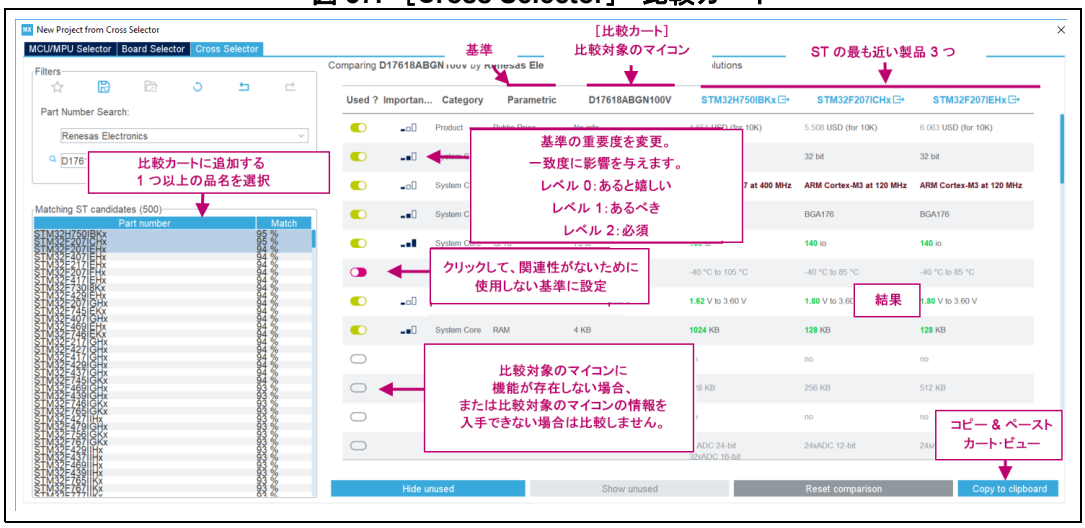

**図 36. ［Cross Selector］- 部分的な品名からの自動補完**


## 比較カート

品名を選択すると、［Matching ST candidates］パネルに、一致する ST の品名候補の一覧が一致率とともに表示されます。

デフォルトでは、最も一致率の高い製品が 3 点選択され、比較元の品名とともに比較カートに追加されます（図 37 参照）。

図 37. ［Cross Selector］- 比較カート




この選択は［Matching ST candidates］パネルで、いつでも変更できます。

比較方法のカスタマイズも可能です。プロジェクトとは無関係と見なせる機能を比較から除外することやその重要度を調整することなどができます。このような選択によって一致率の計算結果が変化します。

比較対象となる品名で目的の機能がサポートされていない場合や、機能の情報が得られない場合は、比較が無効になります。

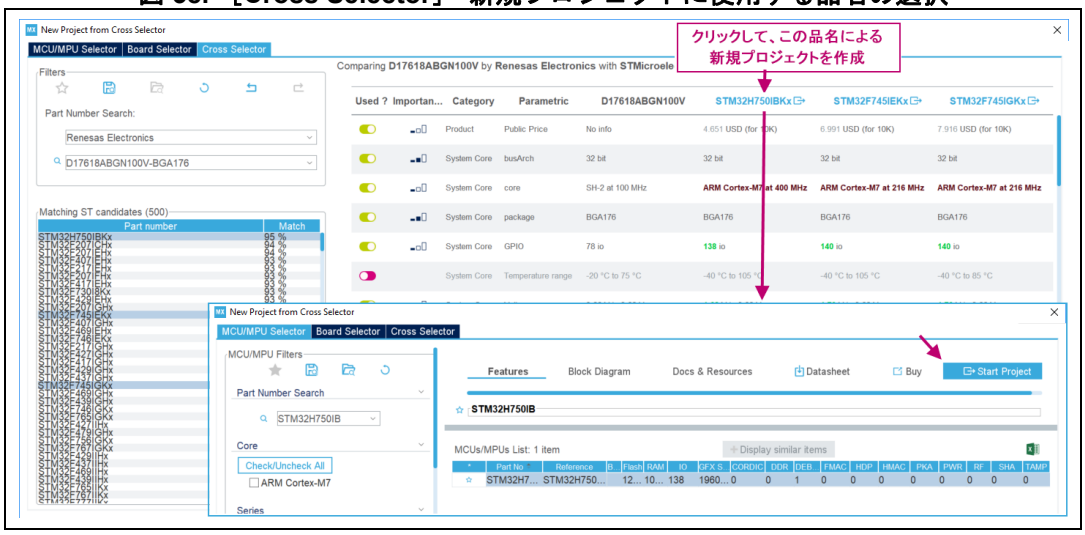
比較カート･ビューの操作およびコピーの保存のために、以下の操作のボタンが用意されています。

- 比較に使用しない基準を非表示にするか、すべての基準を表示するかを選択する
- 比較設定を STM32CubeMX のデフォルト設定に戻す
- 現在の比較カート･ビューをドキュメントまたはメールにコピー & ペーストする

## 新規プロジェクトに使用するマイクロコントローラまたは MPU の選択

比較カートの STM32 の品名をクリックすると、［MCU/MPU Selector］タブでその製品が選択され、Start Project をクリックすると､その品名を使用する新規プロジェクトが作成されます（図 38 参照）。

図 38. ［Cross Selector］- 新規プロジェクトに使用する品名の選択




［Cross Selector］タブをクリックすると、カートに戻り、現在の選択を別の品名に変更できます。

## 4.3 プロジェクト･ページ

STM32 の品名またはボードを選択するか、以前に保存したプロジェクトを読み込むと、以下のビューを表示したプロジェクト･ページが開きます（各ビューの詳細な説明は、それぞれの該当セクションを参照してください）。

- **［Pinout & Configuration］**
- **［Clock Configuration］**
- **［Project Manager］**
- **［Tools］**

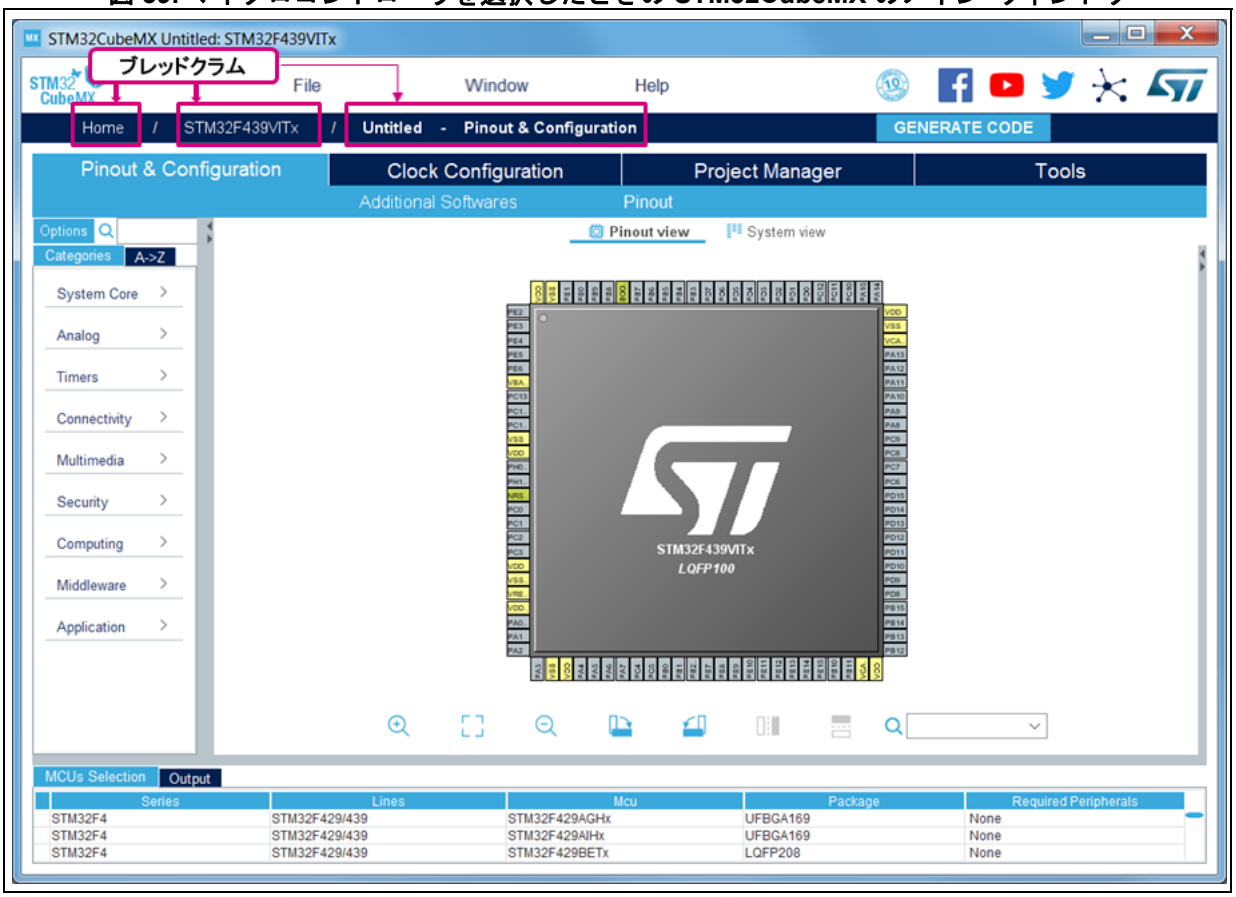
プロジェクト設定に影響を与えずに、これらのビューを切り替えることができます。

GENERATE CODE ボタンは常に表示され、クリックすると現在のプロジェクト設定に対応したコードを生成できます。

また、便利なナビゲーション･ブレッドクラム（図 39 参照）により、現在 STM32CubeMX ユーザ･インタフェースのどこを操作しているかを把握でき、そこから以下の場所に移動できます。

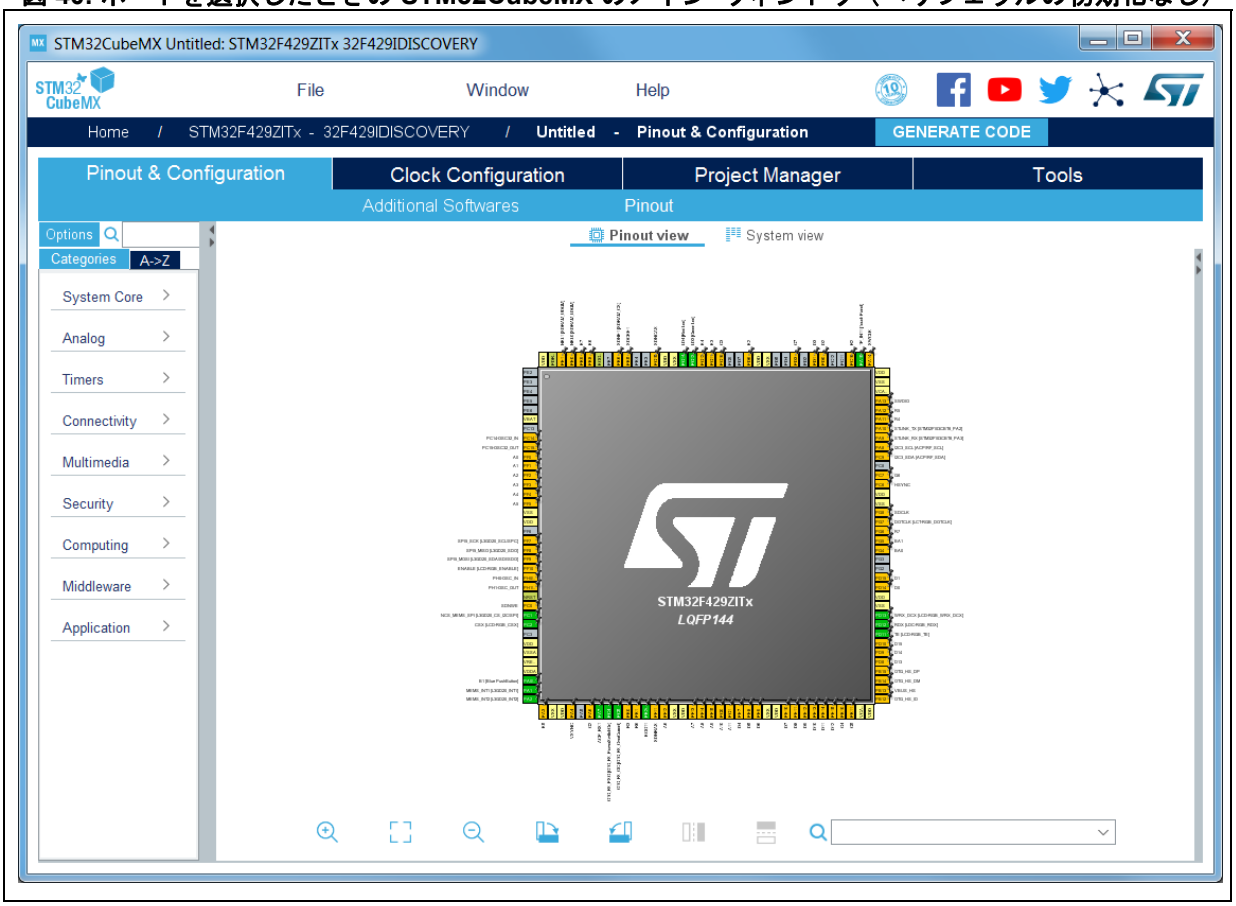
- ［Home］ブレッドクラムをクリックすると［Home］ページに移動します
- 品名をクリックすると新規プロジェクト･ウィンドウに移動します
- プロジェクト名（名前をまだ指定していない場合は［Untitled］）をクリックするとプロジェクト･ページに戻ります

図 39. マイクロコントローラを選択したときの STM32CubeMX のメイン･ウィンドウ



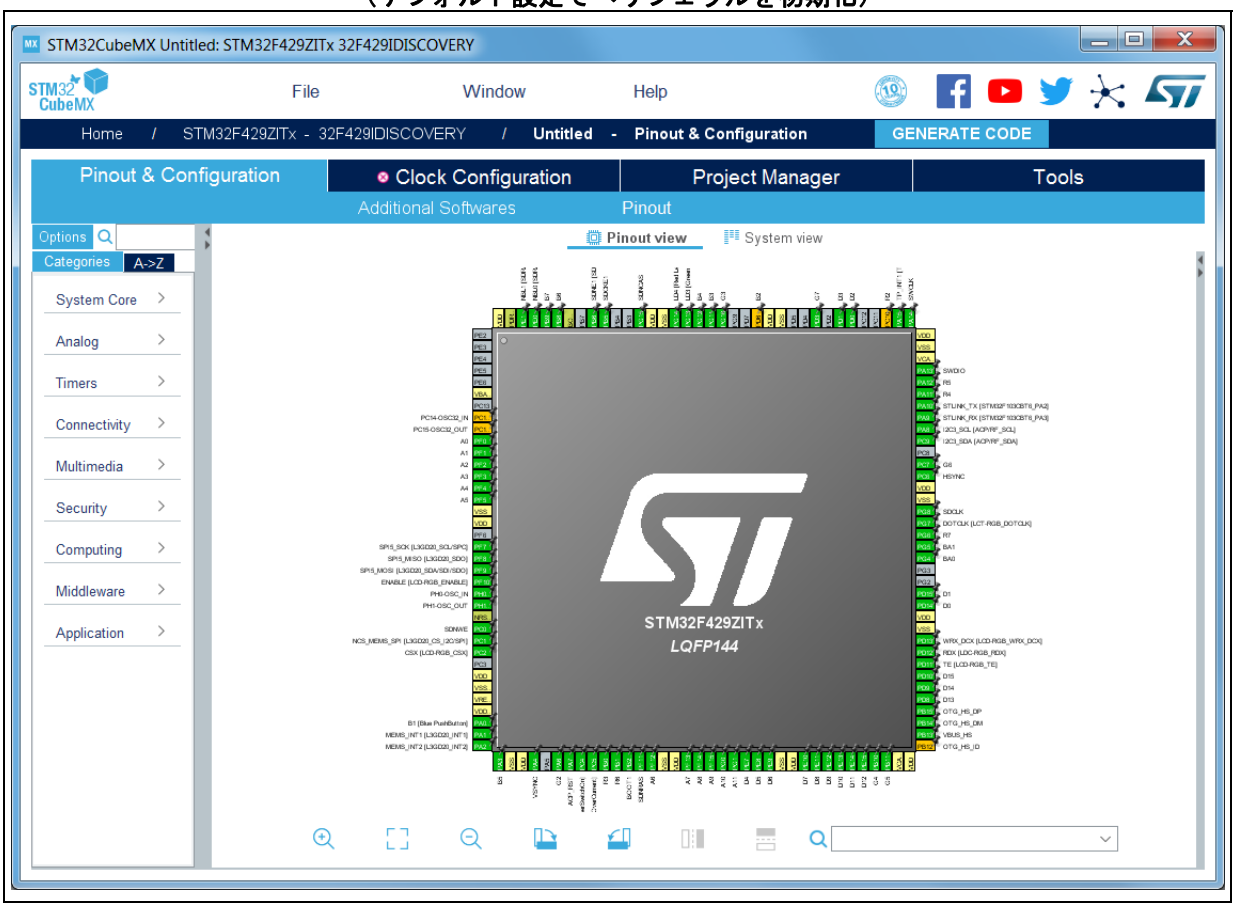
ボードを選択し、すべてのペリフェラルをデフォルト･モードに初期化することを要求するダイアログ･ウィンドウで［No］を選択すると、そのボードのピン配置が自動的に設定されます。なお、ペリフェラル･モードを設定していない状態では、GPIO として設定したピンのみが設定済みとして緑色でハイライトされます。この状態で、アプリケーションに必要なペリフェラル･モードを手動でペリフェラル･ツリーから選択できます（図 40 を参照）。

**図 40. ボードを選択したときの STM32CubeMX のメイン･ウィンドウ（ペリフェラルの初期化なし）**



ボードを選択して、すべてのペリフェラルをデフォルト･モードに初期化することに同意した場合は、ピン配置およびボード上で使用可能なペリフェラルに対するデフォルト･モードの両方が自動的に設定されます。これは、ユーザ･アプリケーションに関連するペリフェラルだけでなく、ボード上で使用可能なすべてのペリフェラルについて、C 初期化コードが STM32CubeMX で生成されることを意味します（図 41 参照）。

**図 41. ボードを選択したときの STM32CubeMX のメイン･ウィンドウ（デフォルト設定でペリフェラルを初期化）**




## 4.4 ［Pinout & Configuration］ビュー

**[Pinout & Configuration]** ビューは、以下のメイン･パネル、機能、メニューを備えています。

- **コンポーネント･リスト**：アルファベット順およびカテゴリ別にコンポーネントの一覧を表示できます。デフォルトのコンポーネント･リストは、選択したマイクロコントローラがサポートするペリフェラルとミドルウェアによって構成されます。このリストからコンポーネントを選択すると、新たに 2 つのパネル（**[Mode]** と **[Configuration]**）が表示され、コンポーネントの機能モードと、生成するコードで指定する初期化パラメータを設定できます。
- **[Pinout]** ビュー：選択したパッケージ（BGA、QFP など）のピン配置がグラフィック表示されます。各ピンにはそれぞれの名前（PC4 など）のほか、現在のオルタネート機能があればその割当てが表示されます。

- **[System view]**：GPIO、ペリフェラル、ミドルウェア、その他のソフトウェアなど、ソフトウェアで設定可能なあらゆるコンポーネントの概要が表示されます。クリック可能なボタンを使用して特定のコンポーネントの設定オプション（[Mode]パネルと［Configuration］パネル）を開くことができます。ボタン･アイコンの色には、設定ステータスが反映されます。
- **[Software Packs]**メニュー：次の 2 つのサブメニューがあります。
  - **[Select Components]**：デフォルトでは使用できないソフトウェア･コンポーネントを現在のプロジェクトに選択します。この選択に応じて**[Pinout & Configuration]**ビューが更新されます。
  - **[Manage Software Packs]**：ソフトウェア･パッケージをインストールまたはアンインストールします。
- **[Additional Software]**機能：デフォルトでは使用できないソフトウェア･コンポーネントを現在のプロジェクトに選択できます。追加のソフトウェア･コンポーネントを選択すると、それに応じて**[Pinout & Configuration]**ビューが更新されます。
- **[Pinout]**メニュー：ピン配置関連の操作を実行できます。ピン配置の設定をクリアする操作や、ピン配置設定を .csv ファイルにエクスポートする操作などがあります。

### ヒント

- 各パネルのサイズは自由に変更できます。パネルの境界にマウス･カーソルを置くとポインタが両方向矢印になります。右クリックしてパネルを拡大または縮小する方向にドラッグします。
- ［Configuration］、［Mode］、［System view］の各パネルは、 で開くことができ、 で閉じることができます。

## 4.4.1 コンポーネント･リスト

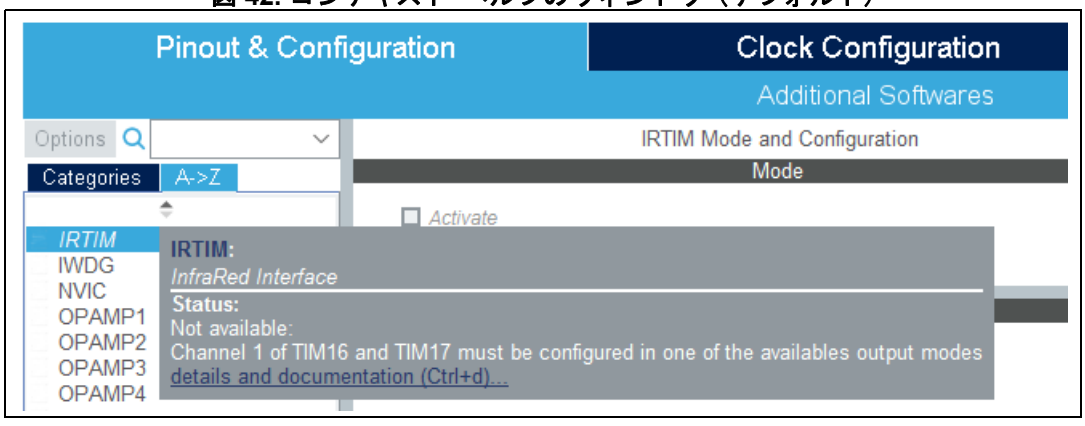
コンポーネント･リストには、プロジェクトで使用できるすべてのコンポーネントが表示されます。コンポーネント･リストからコンポーネントを選択すると、［Mode］パネルと［Configuration］パネルが表示されます。

### コンテキスト･ヘルプ

ペリフェラルまたはミドルウェアの短縮名の上にマウス･カーソルを置くと**コンテキスト･ヘルプ**のウィンドウが表示されます。

このウィンドウにはデフォルトで拡張名が表示されるほか、設定の競合のソースがあれば、それも表示されます（図 42 を参照）。

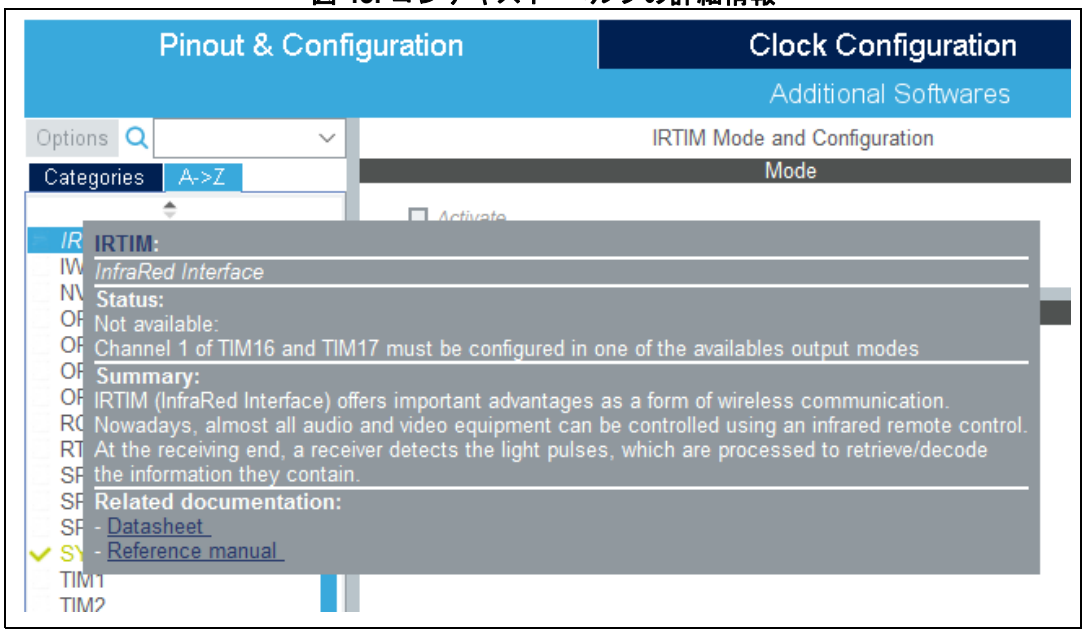
**図 42. コンテキスト･ヘルプのウィンドウ（デフォルト）**



［details and documentation］リンク（または CTRL+D）をクリックすると、概要や関連ドキュメントのリンクなどの追加情報が表示されます（図 43 を参照）。ペリフェラルの場合、［Datasheet］または［Reference manual］をクリックすると、STM32CubeMX リポジトリ･フォルダに格納されている該当のドキュメントが開き、現在の状態に関連する章が表示されます。マイクロコントローラのデータシートとリファレンス･マニュアルは、ユーザの要求によってのみ STM32CubeMX のリポジトリにダウンロードされるので、その入手には有効なインターネット接続が必要です。

- インターネット接続を確認するには、**［Help］→［Updater Settings］**メニューで **［Connection］**タブを開きます。
- 現在選択しているマイクロコントローラのリファレンス･マニュアルのダウンロードを要求するには、**［Help］→［Refresh Data］**メニュー･ウィンドウを選択して **［Refresh］**をクリックします。

**図 43. コンテキスト･ヘルプの詳細情報**



## アイコンとカラー･スキーム

表 5 に、コンポーネント･リスト･ビューで使用されるアイコンとカラー･スキーム、および［Mode］パネルでそれに対応するカラー･スキームを示します。

**表 5. コンポーネント･リスト、モード･アイコンとカラー･スキーム**

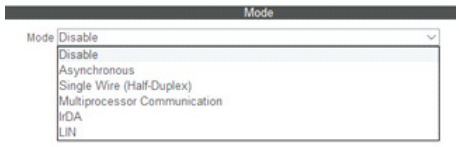
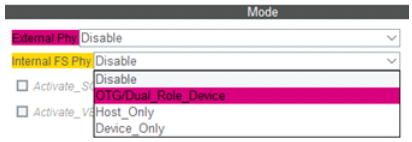
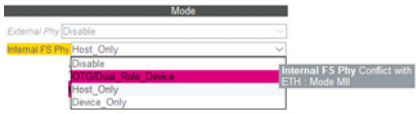
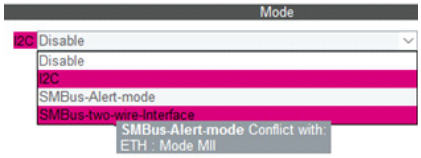
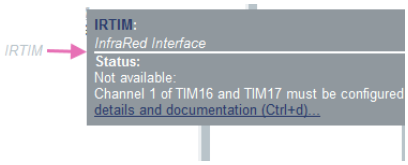
| 表示 | コンポーネントの状態 | 対応する［Mode］ビュー /ツールチップ |
|---|---|---|
| **黒のプレーン･テキスト**<br>例：<br>UART5 | ペリフェラルは未設定（モードが未設定）で、どのモードも選択できます。 |  |
| **灰色のイタリック･テキスト**<br>例：<br>*LWIP* | 解消されていない制約があることからペリフェラルを使用できません。ツールチップを参照してください。 |  |
|  <br>例：<br>ETH | ペリフェラルは設定（モードが 1 つ以上設定）されていて、それ以外のモードもすべて選択できます。緑のチェック･マークは、すべてのパラメータが適切に設定されていること、×記号は適切に設定されていないことを表します。 |  |
|  <br>例：<br>USB_OTG_HS | ペリフェラルは未設定（モードが未設定）で、選択できないモードが 1 つ以上あります。 |  |
| 例：<br>USB_OTG_HS | ペリフェラルは設定（モードが 1 つ設定）されていて、それ以外のモードの中に選択できないものが 1 つ以上あります。 |  |

表 5. コンポーネント･リスト、モード･アイコンとカラー･スキーム （続き）

| 表示 | コンポーネントの状態 | 対応する［Mode］ビュー /ツールチップ |
|---|---|---|
| 例：<br>I2C2 | ペリフェラルは未設定（モードが未設定）で、選択できるモードがありません。<br>ペリフェラル名の上にマウス･カーソルを置くと、競合を説明するツールヒントが表示されます。 |  |
| 例：IRTIM | 制約があることからペリフェラルを使用できません。 |  |

### 4.4.2 コンポーネントの［Mode］パネル

左側パネルのコンポーネント･リストからコンポーネントを選択すると、**［Mode］**パネルが表示されます。

**［Mode］**パネルは、選択したペリフェラルや動作モードに基づいてマイクロコントローラのピンを設定するときに効果的です。STM32 マイクロコントローラでは、同じピンをさまざまなペリフェラルで複数の機能（オルタネート機能）に使用できます。したがって、このツールでは、ユーザが選択したペリフェラルに最適なピン配置が検索されます。STM32CubeMX では、自動的には解決できない競合がハイライトされます（表 5 参照）。

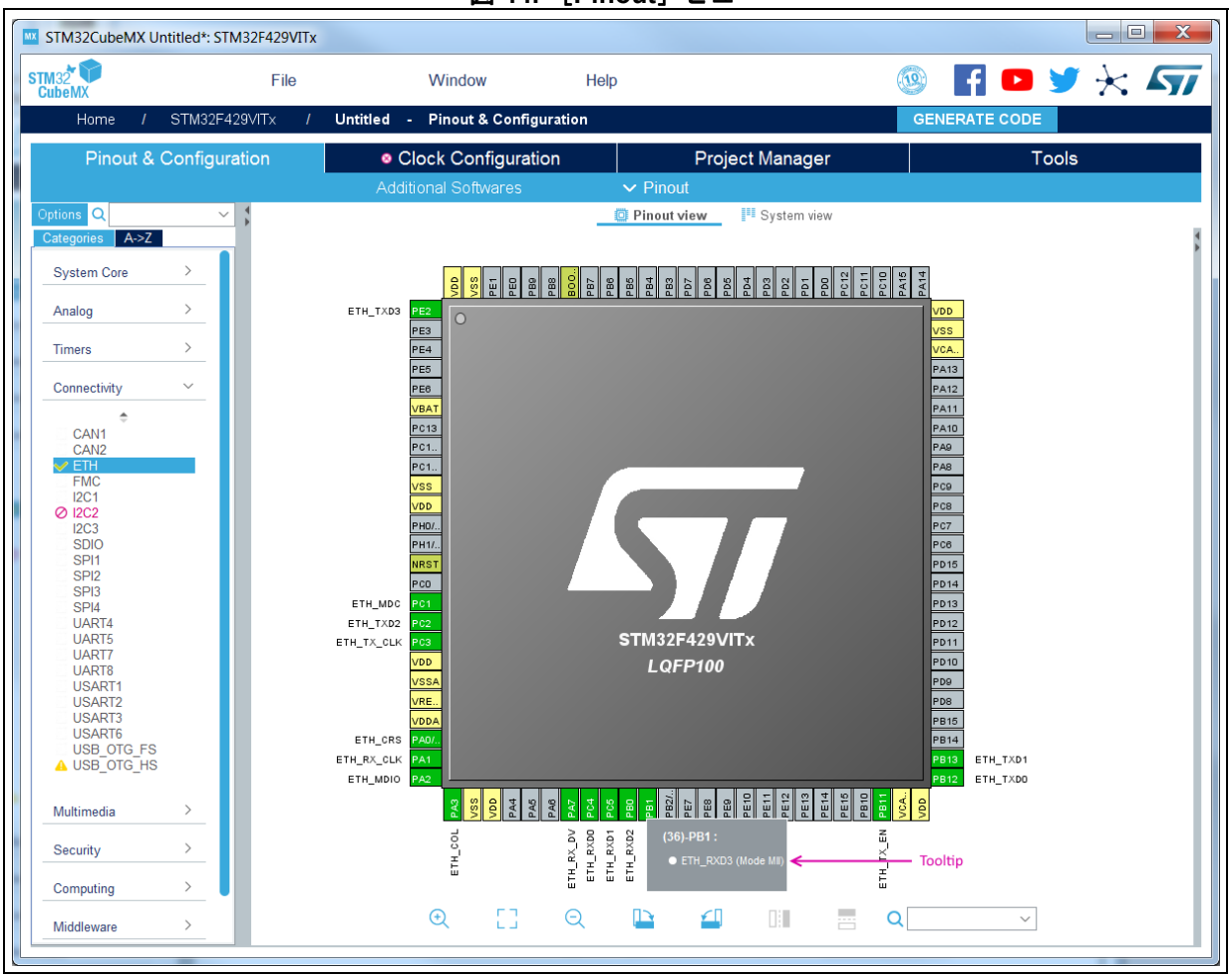
**［Mode］**パネルでは、プロジェクトで使用するミドルウェアやその他のソフトウェア･コンポーネントを有効にすることもできます。

注： **ミドルウェアの中には（USB、FATS、LwIP）、ミドルウェア･モードを有効にする前に、ペリフェラル･モードを有効にする必要があるものがあります。ツールヒントに設定方法のガイダンスが表示されます。FatFs では、ユーザ定義モードが導入されています。これにより、事前にペリフェラル･モードを定義しなくても、STM32CubeMX で FatFs コードを生成できるようになります。その後は、ユーザの判断で、生成した user_diskio.c/.h ドライバ･ファイルを必要なコードで更新することによって、ミドルウェアをユーザ定義ペリフェラルに接続します。**

### 4.4.3 ［Pinout］ビュー

Pinout view を選択すると、選択した品名について、選択したパッケージ（BGA、QFP など）のピン配置がグラフィック表示されます。各ピンには、名前（PC4 など）と設定の状態が表示されるほか、オルタネート機能が設定されていれば、その割当て（ETH_MII_RXD0 など）も表示されます。図 44 の例を参照してください。

図 44. ［Pinout］ビュー




**[Pinout]**ビューは、**[Mode]**パネルでのコンポーネントの設定に一致するように自動的に更新されます。

**[Mode]**パネルではなく、**[Pinout]**ビューでピンを直接割り当てるには、マイクロコントローラに関する十分な知識が必要です。各ピンには、割り当てることができる特定の機能があるからです。

## ヒント

メニューとショートカットの一覧は、表 2：［Home］ページのショートカット を参照してください。

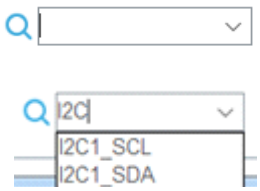
- マウスのホイールを使用すると、ビューを拡大または縮小できます。
- チップ図をクリックしてドラッグすることで図を移動できます。
- ［Best Fit］をクリックすると、最適な位置とサイズに図を再設定できます。
- ［Pinout］→［Export pinout］メニューを使用すると、ピン配置の設定を .csv テキスト･フォーマットでエクスポートできます。
- 基本的な制御機能の中には、ピン･ブロックの整合性維持など、組込み済みのものがあります。詳細については 付録 A：STM32CubeMX のピン割当てルールを参照してください。

### 4.4.4 ピン配置のメニューとショートカット

表 6. ピン配置のメニューとショートカット

| 名前またはアイコン | ショートカット | 説明 |
|---|---|---|
| [Keep Current Signals Placement] | *Ctrl-K* | 新しいペリフェラル動作モードに一致するようにピン割当てを移動する処理が実行されないようにします。このチェックボックスのチェックは外しておき、ピン割当てをピン単位でブロックできる新しいピン設定機能を使用することをお勧めします。 |
| [Show User Label] | なし | [Pinout] ビューにユーザ定義のラベルを表示します。 |
| [Undo Mode and pinout] | *Ctrl-Z* | 最後に指定した設定内容を、一度に 1 つずつ指定前の設定に戻します。 |
| [Redo Mode and pinout] | *Ctrl-Y* | 元に戻した手順を、一度に 1 つずつ再度実行します。<br>**警告**（制約）: プラットフォーム設定タブの設定は復元されません。 |
| [Disable All Modes] | *Ctrl-D* | 有効に設定済みのすべてのペリフェラルおよびミドルウェアのモードを「無効」にリセットします。その結果、これらのモードに設定されて緑色表示であったピンは、「未使用」（灰色表示）にリセットされます。<br>ペリフェラルおよびミドルウェアのラベルは、未使用の場合は緑色から黒色、使用不可の場合は灰色に変化します。 |
| [Clear Pinouts] | *Ctrl-P* | [Pinout] ビューでユーザが設定したピン配置をクリアします。<br>この操作により、設定済みのピンはすべてリセット状態に戻り、有効になっていたペリフェラルおよびミドルウェアの各モードも、ピンで信号を使用しているかどうかに関係なく、すべて無効になります。 |
| [Pins/Signals Option] | *Ctrl-O* | すべての設定済みのピンを一覧表示します。ピンに割り当てられた信号名、リストにある各ピンのラベル名を指定できる[Label]フィールドも表示されます。<br>このメニューをアクティブにするには、ピンを 1 本以上設定している必要があります。<br>ピンのアイコンをクリックすると、ピンに対する信号の割当てをピン単位でロックまたはロック解除できます。<br>複数の行を選択し、右クリックして表示されるコンテキスト･メニューを使用すると、選択した信号のすべてを一度にロックまたはロック解除できます。<br>列見出しをクリックすると、名前のアルファベット順やマイクロコントローラ上での配置に従って行を並べ替えることができます。 |
| [Clear Single Mapped Signals] | *Ctrl-M* | モードが関連付けられていない信号（オレンジ色でハイライトされ、ロックされていない信号）のピンへの割当てをクリアします。 |
| [List Pinout Compatible MCUs] | *Alt-L* | 現在のプロジェクトのピン設定に最も一致するマイクロコントローラのリストを表示します。一致には次のような種類があります。<br>– 完全一致<br>– ハードウェア互換性のある部分一致 : ピンの位置は同じでも、ピン名が変更される可能性がある場合。<br>– ハードウェア互換性のない部分一致 : すべての信号をマッピングできるが、すべてが同じピン位置に割り当てられるとは限らない場合。<br>セクション 15 : チュートリアル 5 : 互換性のあるマイクロコントローラへの現在のプロジェクト設定のエクスポートを参照してください。 |
| [Export pinout with Alternate functions] | - | ピン設定を、オルタネート機能の情報とともに .csv テキスト･ファイルとして出力します。 |
| [Export pinout without Alternate functions] | *Ctrl-U* | ピン設定を、オルタネート機能の情報を除外して .csv テキスト･ファイルとして出力します。 |

表 6. ピン配置のメニューとショートカット （続き）

| 名前またはアイコン | ショートカット | 説明 |
|---|---|---|
| [Reset used GPIOs] | *Alt-G* | 設定済みのすべての GPIO ピンのうち、設定を解除する GPIO ピンの数を指定するウィンドウを表示します。 |
| [Set unused GPIOs] | *Ctrl-G* | 未使用のすべての GPIO ピンのうち、何らかの設定を適用する GPIO ピンの数を指定するウィンドウを表示します。<br>それらの GPIO ピンのモードとして、入力、出力、またはアナログ（消費電力を最適化する場合の推奨設定）を指定します。<br>**注意： このメニューを使用する前に、マイクロコントローラのデバッグ機能を使用するようにデバッグ･ピン（SYS ペリフェラルで提供）が設定されていることを確認します。** |
| [Layout reset] | - | - |
|  | - | [Pinout] ビューを拡大します。 |
|  | - | チップのピン配置図を画面に合わせたサイズに調整します。 |
|  | - | [Pinout] ビューを縮小します。 |
|  | - | 時計方向に 90 度、回転します。 |
|  | - | 反時計方向に 90 度、回転します。 |
|  | - | 水平方向に反転して上面図と底面図を入れ替えます。 |
|  | - | 垂直方向に反転して上面図と底面図を入れ替えます。 |
|  | - | この検索フィールドでは、[Pinout] ビューにあるピン名、信号名、信号ラベル、代替ピン名を検索できます。<br>検索基準との一致が見つかると、それらのピン（複数可）が [Pinout] ビュー上で点滅します。<br>点滅を停止するには [Pinout] ビューをクリックします。 |

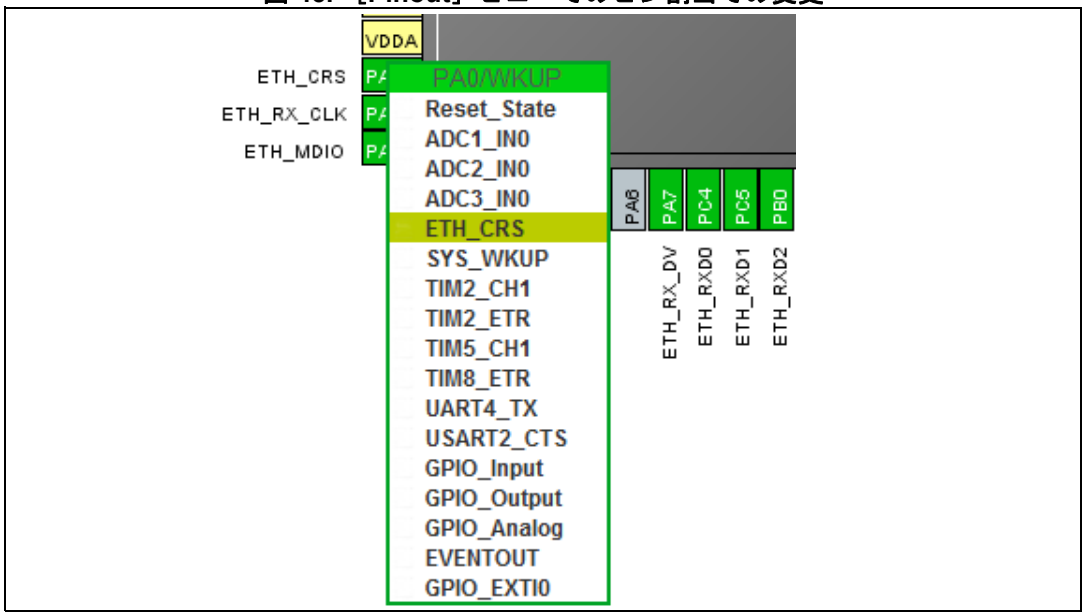
### 4.4.5 ［Pinout］ビューの高度な操作方法

#### 手動によるピン割当ての変更

手動でピン割当てを変更するには次の手順に従います。

1. **［Pinout］**ビューでピンをクリックして、選択可能な他のすべてのオルタネート機能を一覧表示します。現在の割当ては青色でハイライトされます（図 45 を参照）。
2. 新しい機能をクリックして選択すると、それがピンに割り当てられます。

**図 45. ［Pinout］ビューでのピン割当ての変更**




#### 手動による別のピンへの機能の再マッピング

手動で機能を別のピンに再マッピングするには、次の手順に従います。

1. **［Pinout］**ビューで、Ctrl キーを押しながら、ピンを左クリックしたままにします。そのピンの機能の再割当て先候補となるピンがあれば青色で点滅表示されます。
2. その機能を再割当て先のピンにドラッグします。

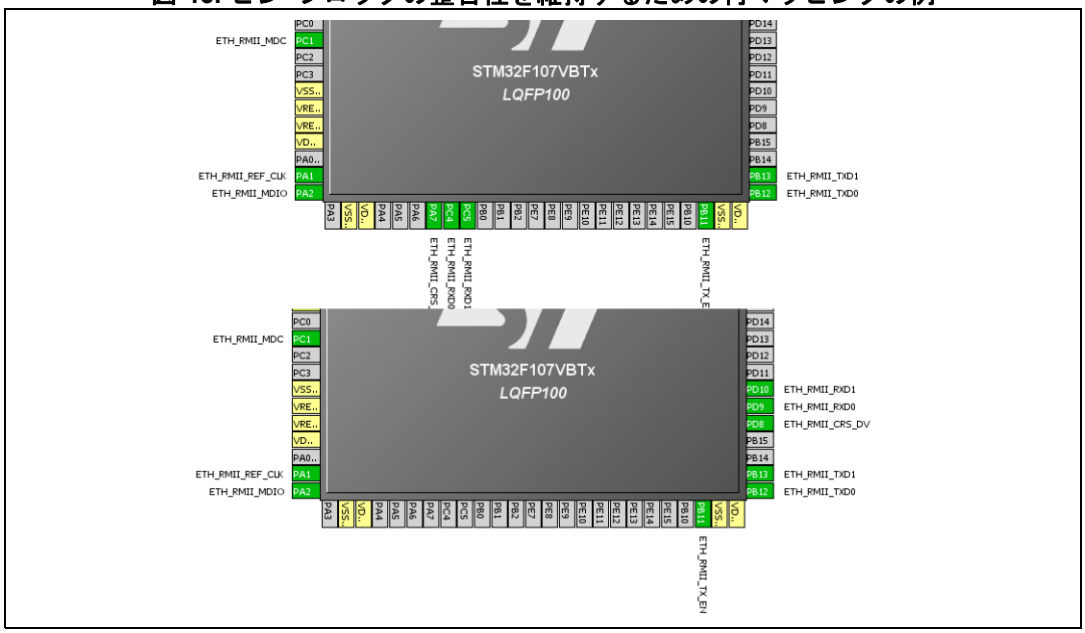
**注意：** **［Pinout］ビューでピン割当てを実行すると、それまでの割当てがすべて上書きされます。**

#### 再割当て先のピンがあいまいな場合の手動による再マッピング

ピン･ブロックの整合性を維持するマイクロコントローラ（STM32F100x/F101x/F102x/F103x および STM32F105x/F107x）の場合、再割当て先のピンがあいまいであることが考えられます。たとえば、目的の再割当て先であるピンのほかにも、割当て先とすることができるブロックが複数存在する場合です。選択可能なすべての代替再マッピング･ブロックを表示するには、再割当て先のピンの上にマウス･カーソルを置きます。

**注：** **「ピン･ブロック」は、特定のペリフェラル･モードを実現するためにまとめて割り当てる必要があるピンのグループです。図 46 に示すように、STM32F107xx マイクロコントローラには、RMII 同期モードで Ethernet ペリフェラルを設定するために使用できる 2 つのピン･ブロックとして、{PC1, PA1, PA2, PA7, PC4, PC5, PB11, PB12, PB13, PB5} と {PC1, PA1, PA2, PD10, PD9, PD8, PB11, PB12, PB13, PB5} があります。**

図 46. ピン･ブロックの整合性を維持するための再マッピングの例




### ピン競合の解決

同じ 1 本のピンを複数のペリフェラル･モードで使用する場合に発生するピン競合を解決するために、STM32CubeMX はペリフェラル･モード機能を他のピンに再割当てしようとします。ピン競合を解決できなかったペリフェラルは赤紫色でハイライトされ、競合について説明するツールチップが表示されます。

モードの再マッピングで競合を解決できない場合、次の操作を試すことができます。

- Keep Current Signals Placement ボックスがチェックされている場合は、該当のペリフェラルを新しいシーケンスで選択し直します。
- **[Keep Current Signals Placement]** ボックスのチェックを外し、STM32CubeMX ですべての再マッピングの組合せを試して解決策を探します。
- ペリフェラルで目的とするモードで使用する信号の中に、割当て可能なピンが存在しないものがあると、そのモードは使用できません。その場合は、そのモードを**手動で再マッピング**します。

## 4.4.6 [Keep Current Signals Placement]

このチェックボックスは **[Pinout]** メニューで表示されます。設定作業中はいつでもチェックでき、またチェックを外すことができます。デフォルトではチェックされていません。

最適なペリフェラル配置（同時に使用するペリフェラルの数が最大になる配置）が得られるように、このチェックボックスはチェックを外しておくことをお勧めします。

ボード設計との一致が目的の場合は、**[Keep Current Signals Placement]**チェックボックスをチェックする必要があります。

### [Keep Current Signals Placement] のチェックを外した場合

STM32CubeMX では、現在のピン配置設定と競合するような新しい要求（新しいペリフェラル･モードまたは新しいペリフェラル･モード機能の選択）を処理するために、それまでマッピングされていたブロックを他のピンに再マッピングできます。

### ［Keep Current Signals Placement］をチェックした場合

現在のペリフェラル･モードに対応するすべての機能のピン割当て（マッピング）が必ず維持されます。割当てが完了すると、特定のピンに割り当てたペリフェラル･モード機能を STM32CubeMX で別のピンに移動することはできなくなります。新しい設定要求は、現在のピン設定を変更しなくても実現可能な場合に処理されます。

この機能は次の場合に便利です。

- **［Peripherals］**パネルで設定したペリフェラルに対応するすべてのピンをロックする場合
- ピンに対する機能のマッピングを維持しながら、**[Pinout]**ビューで手動で再マッピングする場合

### ヒント

モードが選択不可になった場合（赤紫色でハイライトされた場合）、このモードのピンの別の再マッピング設定を探し出すには次の手順を試します。

1. ［Pinout］ビューで、モードが再び選択可能になるまで、割り当てられている機能を 1 つずつ選択解除します。
2. つづいて、同じモードを再度選択して、新しいシーケンスでピン配置設定を続行します（再マッピングの例については、付録 A : STM32CubeMX のピン割当てルールを参照）。この操作には時間を要するので、**[Keep Current Signals Placement]**チェックボックスのチェックを外すことをお勧めします。

**注 :** **［Keep Current Signals Placement］のチェックを外しても、STM32CubeMX によって GPIO_ 機能が移動することはありません（GPIO_EXTI 機能を除く）。**

## 4.4.7 ピンへの信号の固定と信号へのラベルの割当て

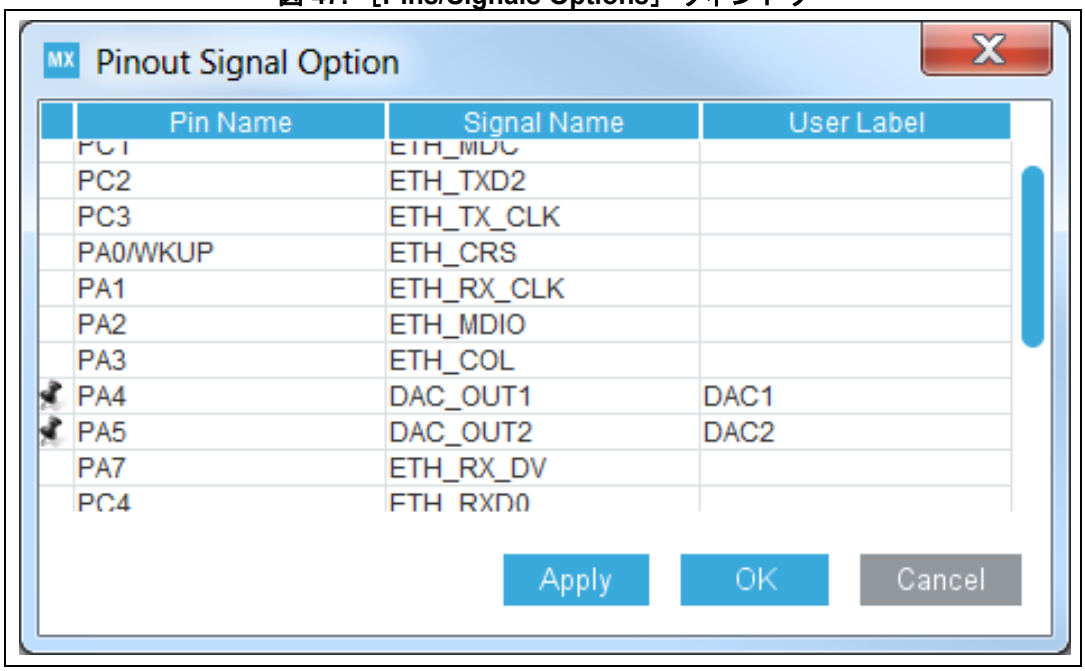
STM32CubeMX は、選択的に信号をピンにロックできる機能を備えています。この機能を使用すると、STM32CubeMX で競合を解決する際に、ピンに固定された信号を他のピンに自動的には移動できなくなります。また、信号にラベルを割り当てることもできます。このラベルは、コード生成で使用します（詳細についてはセクション 6.1 : を参照）。

次に示すさまざまな方法で、ピンへの信号の固定や固定解除、信号へのラベルの割当てができます。

1. **［Pinout］**ビューで、信号が割り当てられているピンを右クリックします。これにより、コンテキスト･メニューが開きます。
   a) ピンに固定していない信号をピンに固定するには、**[Signal Pinning]**を選択します。これにより、それらのピンにピン･アイコンが表示されます。これで、ピン割当ての競合を解決する場合などでも、その信号を自動的には移動できなくなります。
   b) ピンに固定した信号の固定を解除するには、**[Signal Unpinning]**を選択します。ピン･アイコンが削除されます。これにより、［Keep Current Signals Placement］のチェックを外しておけば、この信号を別のピンに移動して、ペリフェラル･モードなどの競合を解決できます。
   c) **[Enter User Label]**を選択して、この信号にユーザ定義ラベルを指定します。**[Pinout]**ビューで、デフォルトの信号名が新しいラベルで置き換えられます。

2. **［Pinout］**メニューで**［Pins/Signals Options］**を選択します。
［Pins/Signals Options］ウィンドウ（図 47 を参照）に、設定されているすべてのピンが一覧表示されます。

**図 47. ［Pins/Signals Options］ウィンドウ**




a) 1 列目をクリックすると、ピンに対して信号を個別に固定または固定解除できます。

b) 複数の行を選択して右クリックし、表示されたコンテキスト･メニューで**［Signal(s) Pinning］**または**［Signal(s) Unpinning］**を選択します。

c) ［User Label］フィールドを選択することでフィールドを編集状態にして、ユーザ定義ラベルを入力します。

d) ピン名または信号名の列見出しをクリックすると、そのアルファベット順で一覧を並べ替えることができます。もう一度クリックすると、デフォルトの順序（マイクロコントローラのピン配置に従った順序）に戻ります。

**注：** **信号が固定されていても、[Pinout] ビューでピンへの信号割当てを手動で変更できます。ピンをクリックすると、そのピンに割当て可能な他の信号が表示されるので、そこから適切な信号を選択します。**

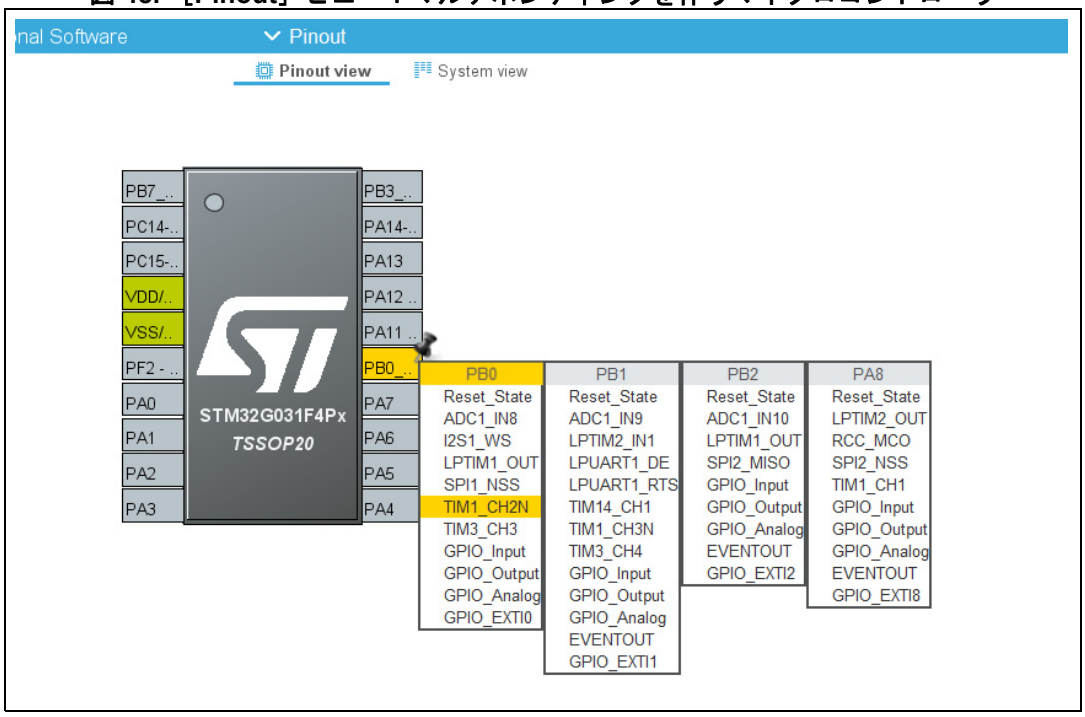
### 4.4.8 マルチボンディング･パッケージのピン配置

SO-8N、TSSOP-20、WLCSP-18 のようにピンが少ない（20 ピン未満）パッケージ向けにマルチボンディングが導入されています。この技術では、マイクロコントローラの複数のパッドで、このようなパッケージの同じピンを共有します。

STM32G0 シリーズでは、STM32G031/G041 マイクロコントローラに導入されました。

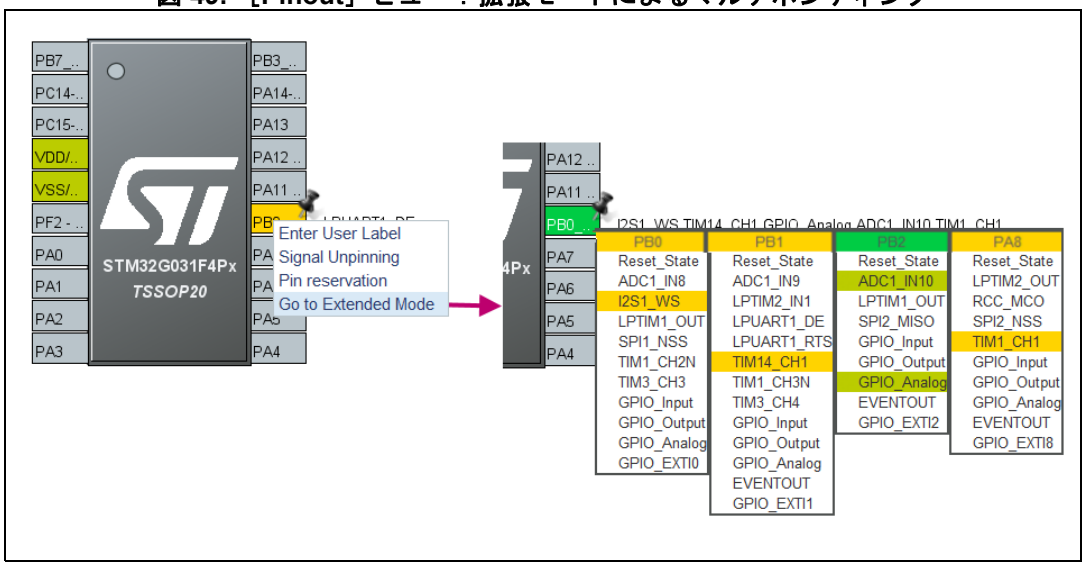
STM32CubeMX の［Pinout］ビューでは、ピンに接続するすべての信号を表示したうえで、ピンごとに 1 つの信号のみを選択できるようにすることができます。ただし、他のアナログ GPIO との組み合わせが可能なアナログ信号を除きます。

図 48. ［Pinout］ビュー：マルチボンディングを伴うマイクロコントローラ




STM32CUbeMX は、ピンを右クリックすることで選択する拡張モードも備えています。このモードにより、1 つのピンに複数の信号を選択できます。このモードは、ループバック･テストなどのテストを目的としています。この機能は慎重に使用する必要があります。電気的な競合やデバイスを損傷する原因となる消費電力の増加が発生する可能性があるからです。

図 49. ［Pinout］ビュー：拡張モードによるマルチボンディング



### 4.4.9 ［System view］

System view を選択すると、GPIO、ペリフェラル、ミドルウェアなどソフトウェアで設定可能なコンポーネントがすべて表示されます。クリック可能なボタンにより、コンポーネントのモードと設定のオプションを表示できます。このボタンのアイコンには、コンポーネントの設定状態が反映されます（設定状態と［System view］に表示されるアイコンについては表 7 を参照してください）。

［Configuration］パネルでコンポーネントの設定を変更すると、その設定の状態に応じて［System view］が自動的に更新されます。

［Mode］パネルでコンポーネントを無効化すると、［System view］が自動的に更新され、そのコンポーネントのボタンは表示されなくなります。

**図 50. ［System view］**

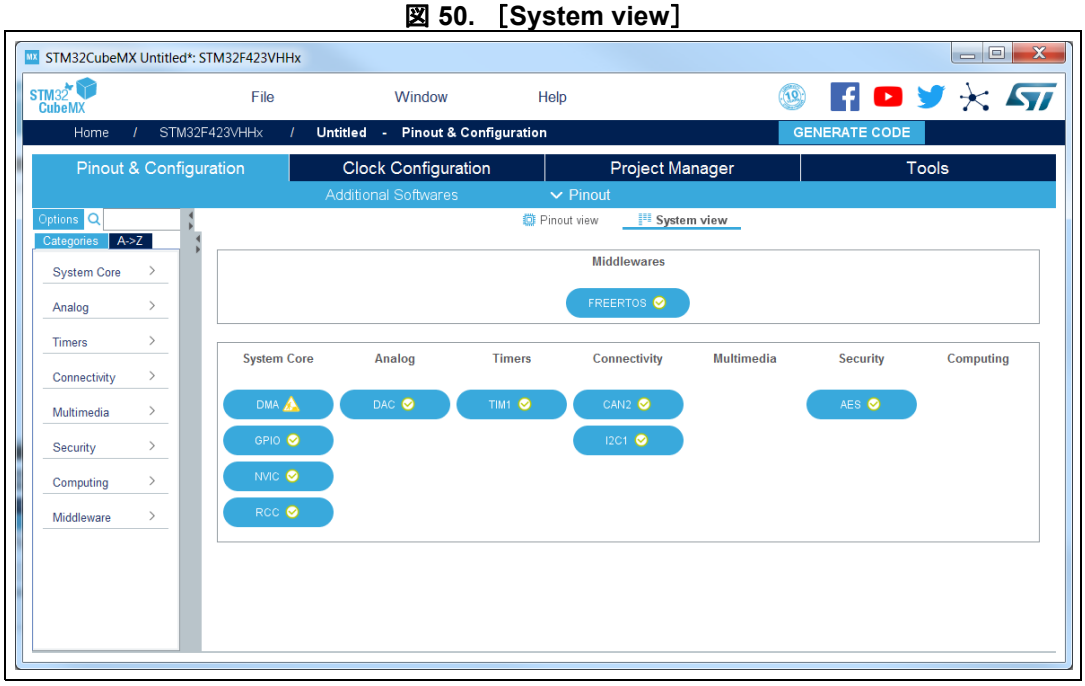
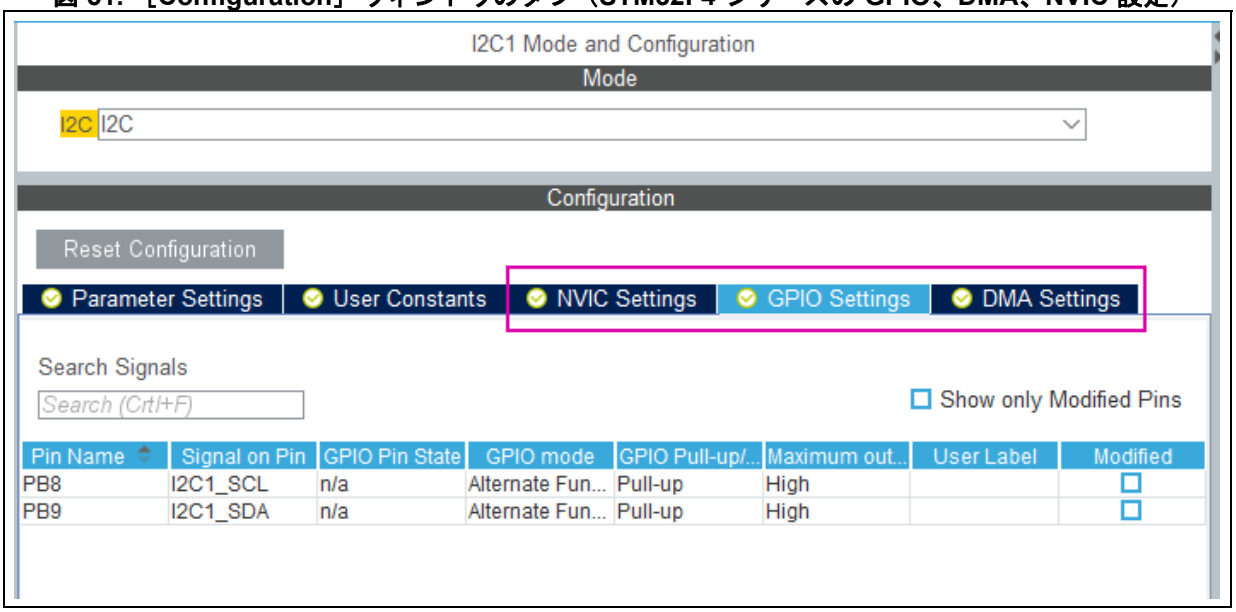



**表 7. 設定状態**

| アイコン | 説明 |
|---|---|
|  | 設定が完了して適切な状態になっています。 |
|  | 設定は正しいものの、未設定の部分があります（それが必須設定でない場合もあります）。 |
|  | 設定が無効で、生成する C プロジェクトが機能するためには修正が必要です。 |

GPIO、DMA、および NVIC の設定にアクセスするには、他のペリフェラルと同様に専用のボタンを使用するか、［Configuration］パネルのタブを使用します（図 51 参照）。

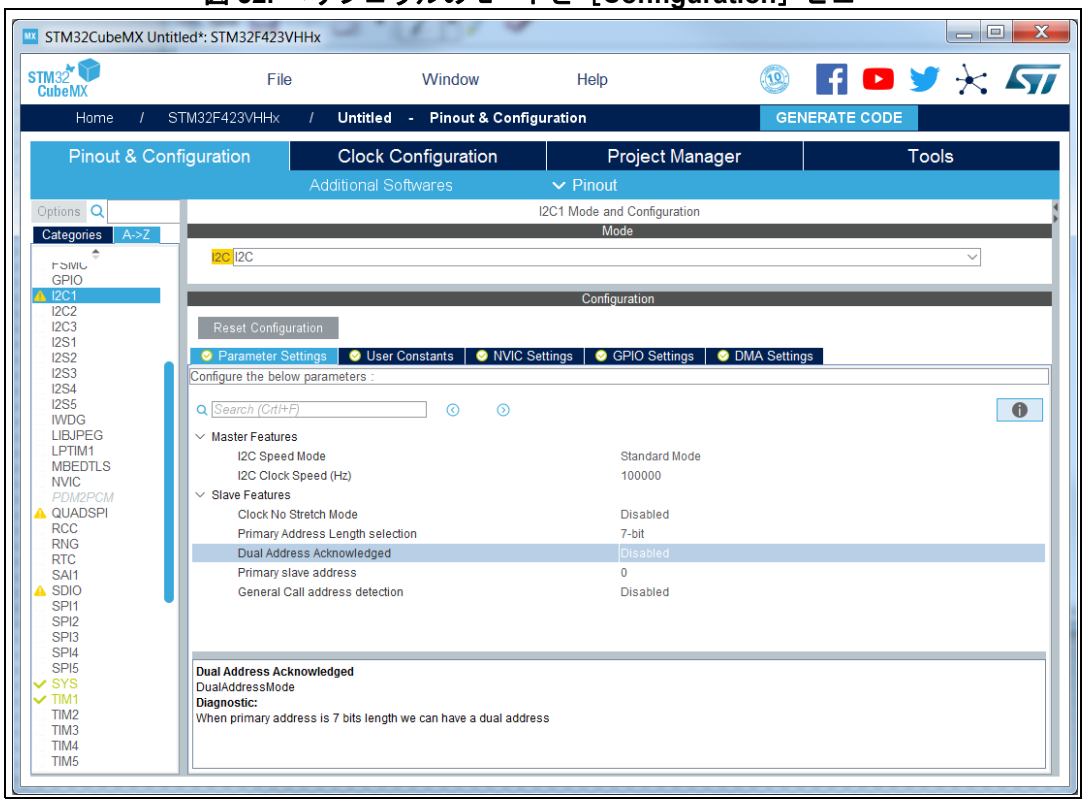
**図 51. ［Configuration］ウィンドウのタブ（STM32F4 シリーズの GPIO、DMA、NVIC 設定）**




### 4.4.10 コンポーネント設定パネル

このパネルは、左側パネルのコンポーネント名をクリックすると表示されます。ここでは、選択した動作モードでペリフェラルまたはミドルウェアを初期化するために必要な機能パラメータを設定できます（図 52 参照）。STM32CubeMX では、これらの設定を使用して、対応する初期化 C コードが生成されます。

図 52. ペリフェラルのモードと［Configuration］ビュー




［Configuration］ウィンドウには、以下の各タブがあります。

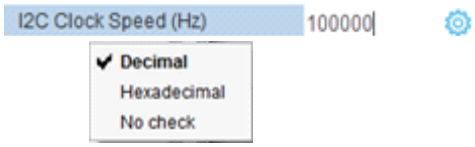
- **[Parameter Settings]**：選択したペリフェラルまたはミドルウェアのライブラリ専用パラメータを設定します。
- **[NVIC Settings]**、**[GPIO Settings]**、**[DMA Settings]**：選択したペリフェラルのパラメータを設定します（設定の詳細については、セクション 4.4.14：、セクション 4.4.12：、およびセクション 4.4.13：を参照）。
- **[User constants]**：プロジェクト全体に共通のユーザ定義定数を作成します（複数作成可。ユーザ定数の詳細については、セクション 4.4.11：を参照してください）。

無効な設定が検出されて、次のいずれかの処理が適用されます。

- 選択した値が最小閾値よりも小さい場合は有効な最小値に再設定され、最大閾値よりも大きい場合は有効な最大値に再設定されます。
- 前の値が最大閾値でも最小閾値でもなかった場合は、前の有効値に再設定されます。
- 赤紫色でハイライトされます。

表 8 に、ペリフェラルとミドルウェアの設定のボタンとメッセージを示します。

**表 8. ペリフェラルとミドルウェアの［Configuration］ウィンドウに表示されるボタンとツールチップ**

| ボタンとメッセージ | アクション |
|---|---|
|  | 説明パネルの表示と非表示を切り替えます。 |
|  | パラメータの設定について説明し、その値の有効範囲（最小値～最大値）を示します。<br>ツールヒントを表示するには、選択可能な値のリストにあるパラメータ値の上にマウス･カーソルを置きます。 |
|  | 歯車アイコンをクリックすると､16 進数値または 10 進数値のどちらを表示するか、またはどの値も確認しないオプション（［No check］オプション）を指定できます。 |
|  | 検索 |
|  | コンポーネントの設定をデフォルト（STM32CubeMX の初期設定）に戻します。 |

## ［No check］オプション

デフォルトでは、入力したパラメータ値が有効であるかどうかが STM32CubeMX によって確認されます。パラメータに対して［No Check］オプションを選択すると、この確認が実行されなくなります。これにより、STM32CubeMX の設定で既知ではない可能性がある任意の値（定数など）を入力できます。

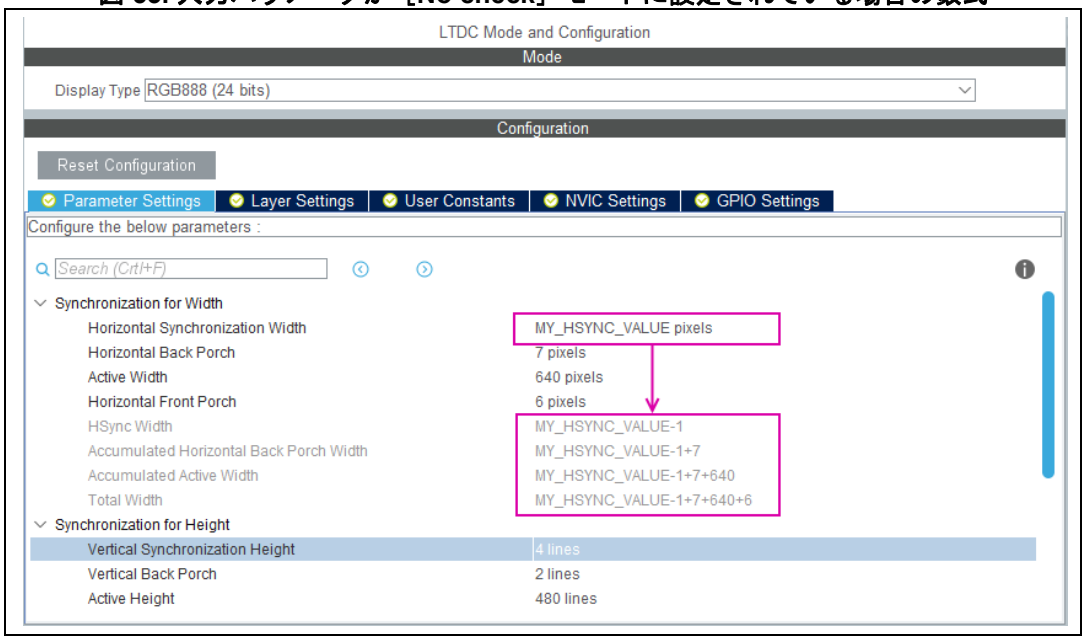
この有効性の確認は、値が整数型（16 進数または 10 進数）のパラメータでのみ省略可能です。指定できる値の事前定義リストから値を選択するパラメータや、整数型以外またはテキスト型のパラメータでは、この有効性の確認を省略できません。

デフォルトのモードに戻して 10 進値または 16 進値の有効性の確認が実行されるようにするには､10 進値または 16 進値を入力して、それらの有効性の確認を実行するオプションをチェックします。

**注意：** **［No Check］に設定した別のパラメータに依存しているパラメータは次のように処理されます。**

- 最小許容値または最大許容値の範囲に収まっているかどうかの評価を別のパラメータに依存しているパラメータの場合：依存相手のパラメータが［No Check］に設定されている場合、最小値または最大値に対する評価と確認は実行されません。
- 現在値の評価を別のパラメータに依存しているパラメータの場合：依存相手のパラメータが［No Check］に設定されている場合、このパラメータの値は自動的には得られなくなります。その代わり、このパラメータの値は、［No Check］に設定されているパラメータの文字列を変数とした式のテキストで置き換えられます（図 53 参照）。

図 53. 入力パラメータが［No check］モードに設定されている場合の数式




## 4.4.11 ［User Constants］設定ウィンドウ

**[User Constants]** タブでは、ユーザ定数を定義できます（図 54 参照）。定数は、STM32CubeMX のユーザ･プロジェクトの main.h ファイルに自動的に生成されます（図 55 を参照）。定義された定数は、ペリフェラルとミドルウェアのパラメータの設定に使用できます（図 56 を参照）。

図 54. ［User Constants］タブ

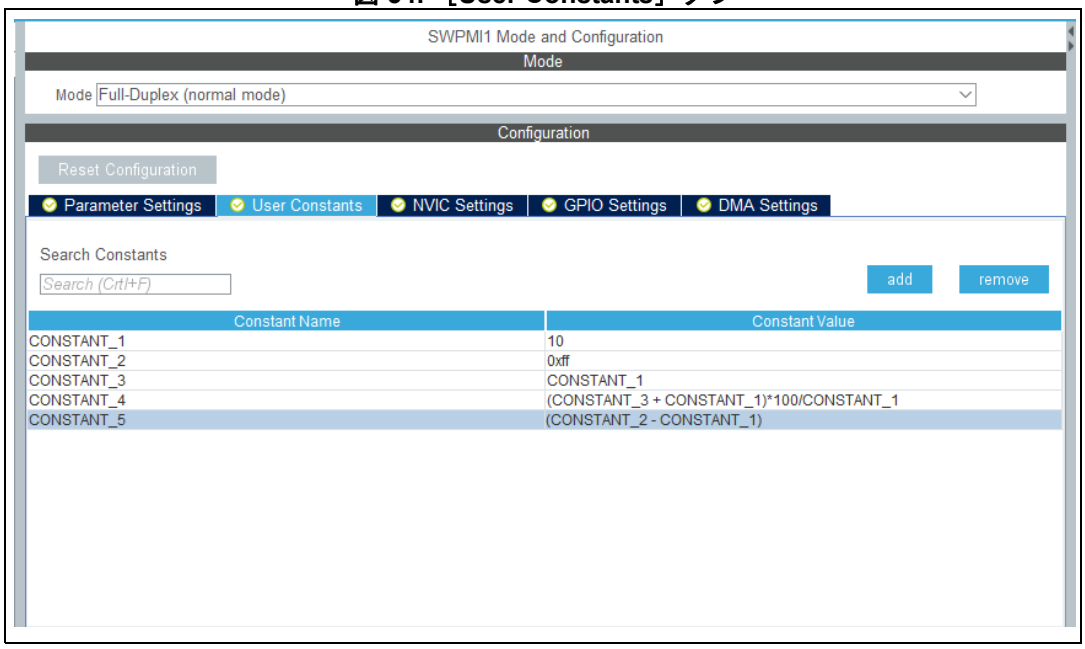
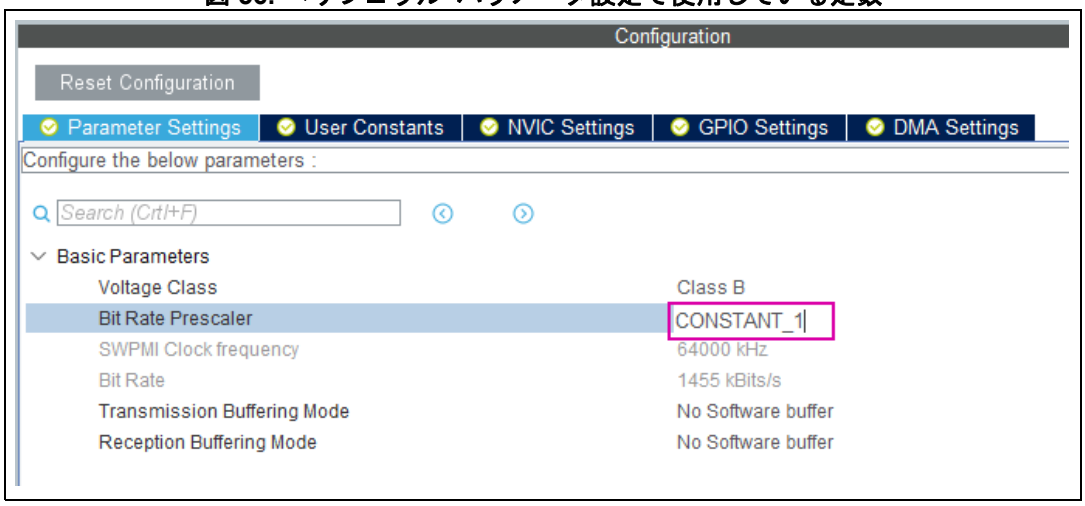


図 55. 生成された main.h からの抜粋

```
/* Includes ------------------------------------------------------------------*/

/* USER CODE BEGIN Includes */

/* USER CODE END Includes */

/* Private define ------------------------------------------------------------*/
#define CONSTANT_1 10
#define CONSTANT_2 0xff
#define CONSTANT_3 CONSTANT_1
#define CONSTANT_4 (CONSTANT_3+CONSTANT_1)*100/CONSTANT_1
#define CONSTANT_5 (CONSTANT_2 - CONSTANT_1)

/* USER CODE BEGIN Private defines */

/* USER CODE END Private defines */
```

図 56. ペリフェラル･パラメータ設定で使用している定数



### ユーザ定数の作成/編集

**[Add]** ボタンをクリックして **[User Constants]** タブを開き、新しいユーザ定義定数を作成します（図 57 を参照）。

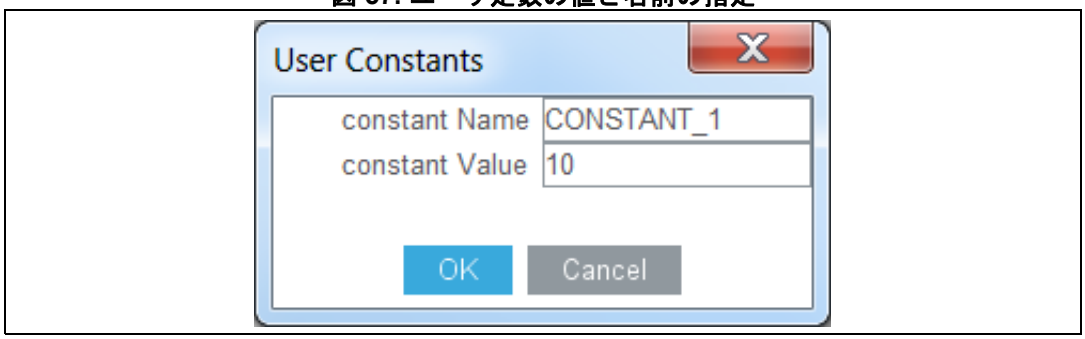
定数の構成は次のとおりです。

- 名前：次の規則に従う必要があります。
  - 一意であること。
  - C/C++ のキーワードではないこと。
  - 空白を使用していないこと。
  - 先頭が数字ではないこと。
- 値：

  定数値には、以下を使用できます（例は 図 54 を参照してください）。
  - 単純な 10 進値または 16 進値
  - 定義済みの定数
  - 数値またはユーザ定義数値定数をオペランドとして、算術演算子（減算、加算、除算、乗算、剰余）で結合した式
  - 文字列：文字列値は "constant_for_usart" のように二重引用符で囲む必要があります。

定数は、定義した後でもその名前と値を変更できます。変更するには、そのユーザ定数を指定している行をダブルクリックします。**[User Constants]** タブが開き、編集できるようになります。定数名の変更は、その定数が使用されているすべての箇所に適用されます。この変更によってペリフェラル設定やミドルウェア設定の状態が変化することはありません。一方、定数値を変更すると、それを使用しているパラメータが変化し、最大閾値の超過などの無効な設定が発生する原因となることがあります。無効なパラメータ設定は、赤紫色でハイライトされます。

**図 57. ユーザ定数の値と名前の指定**



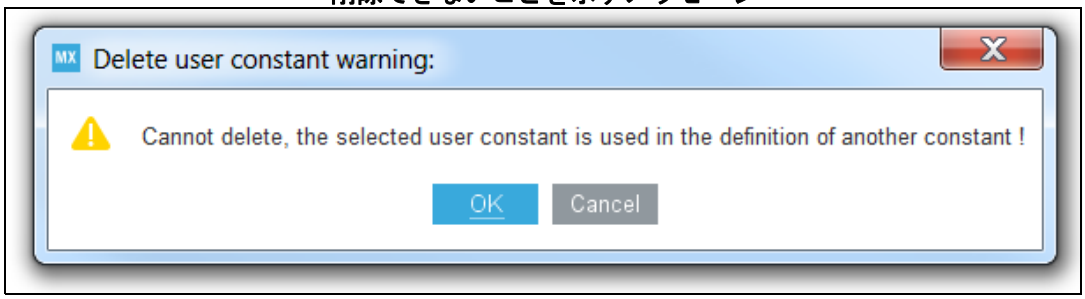
### ユーザ定数の削除

既存のユーザ定義定数を削除するには［**Remove**］ボタンをクリックします。

次の場合を除き、この操作によってユーザ定数は自動的に削除されます。

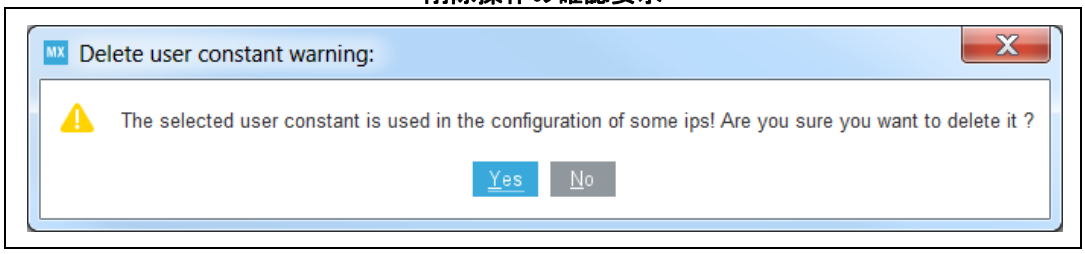
- 定数が別の定数の定義に使用されている場合。ポップアップ･ウィンドウが開いて、説明のメッセージが表示されます（図 58 を参照）。

**図 58. 別の定数の定義に既に使用されている定数は削除できないことを示すメッセージ**



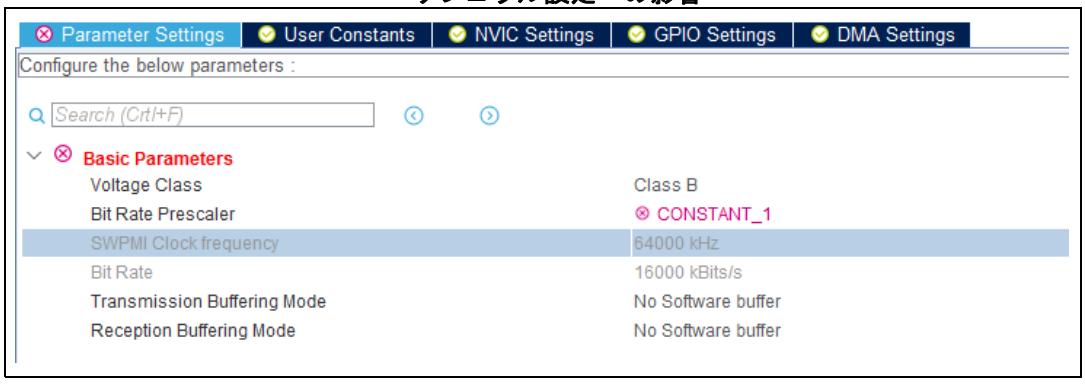

- 定数がペリフェラルまたはミドルウェアのライブラリ･パラメータの設定に使用されている場合。このような定数を削除するとペリフェラル設定またはミドルウェア設定が無効になるので、削除しても良いか確認することが要求されます（図 59 参照）。

**図 59. パラメータ設定で使用されているユーザ定数の削除 - 削除操作の確認要求**




［**Yes**］をクリックすると、ペリフェラル設定が無効になります（図 60 参照）。

**図 60. ペリフェラル設定で使用されているユーザ定数の削除 - ペリフェラル設定への影響**



### ユーザ定数の検索

[**Search Constants**] フィールドを使用すると、すべてのユーザ定数の一覧から名前または値で定数を検索できます（図 61 と図 62 を参照してください）。

**図 61. ユーザ定数リストを名前で検索**

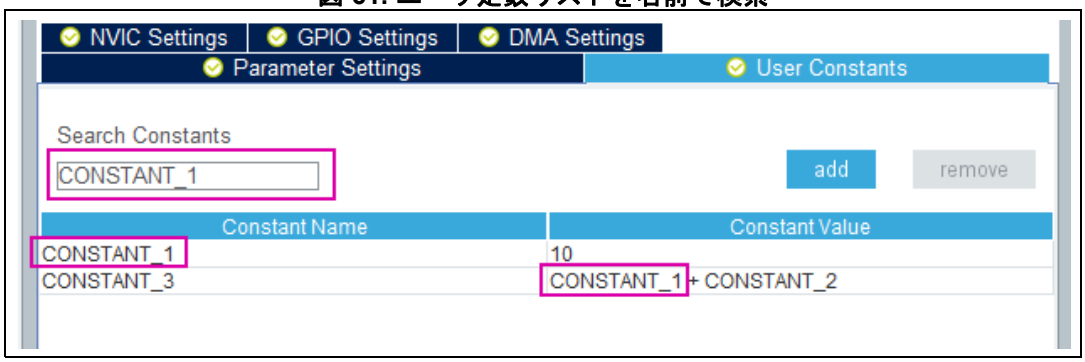
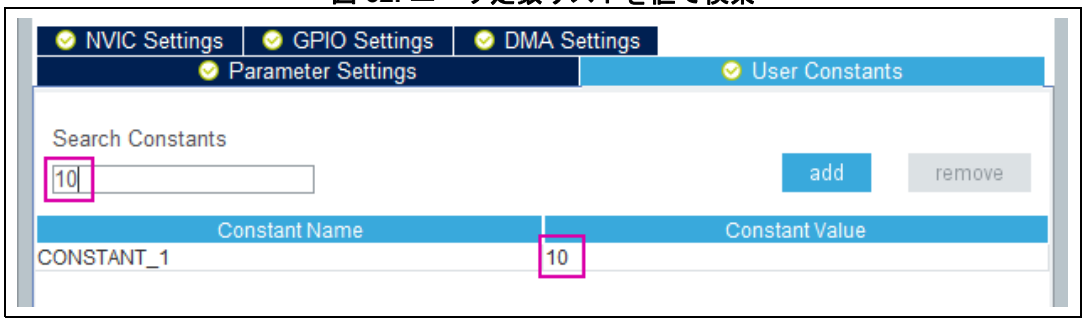



**図 62. ユーザ定数リストを値で検索**

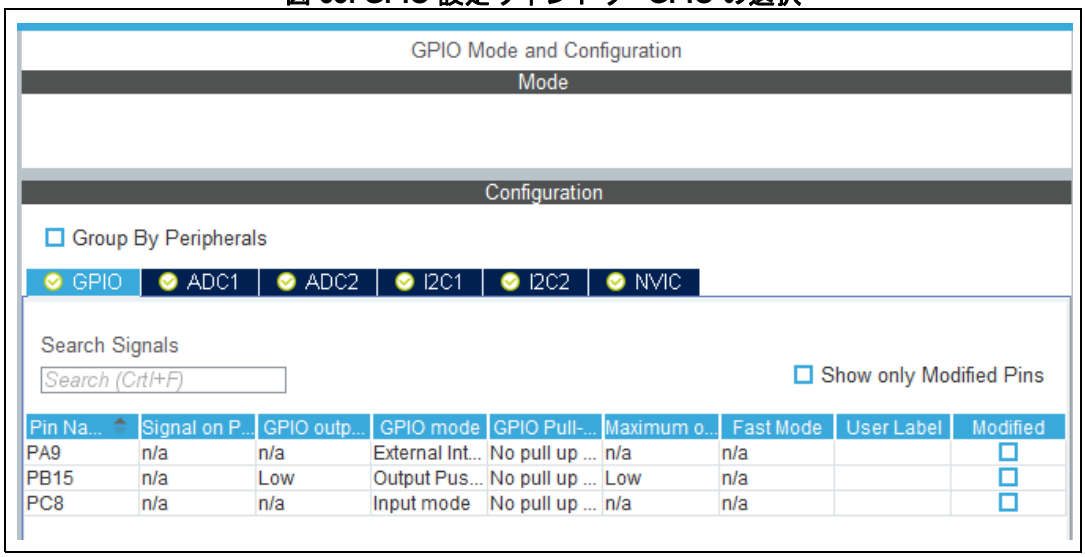


### 4.4.12 GPIO設定ウィンドウ

［System view］パネルで［GPIO］をクリックすると GPIO 設定ウィンドウが開き、GPIO ピンを設定できます（図 63 参照）。設定にはデフォルト値が設定されていますが、ペリフェラル設定によっては不適切であることが考えられます。特に、ペリフェラルの通信速度に対して GPIO の速度が十分であるかどうかを確認し、必要に応じて内部プルアップ接続を選択します。

注： **ペリフェラル･インスタンス設定ウィンドウの専用ウィンドウで、特定のペリフェラル･インスタンスの GPIO 設定にアクセスすることもできます。さらに、GPIO は、出力モード（デフォルトの出力レベル）に設定できます。生成されるコードが設定に応じて更新されます。**

**図 63. GPIO 設定ウィンドウ - GPIO の選択**




特定の行をクリックするか、複数の行を選択して、それに対応する GPIO パラメータを表示します。

- **GPIO PIN State**

  GPIO 出力レベルのデフォルト値を変更します。デフォルトでは［Low］に設定されていますが、［High］に変更できます。

- **GPIO mode**（アナログ、入力、出力、オルタネート機能）

  **[Pinout]** ビューでペリフェラル･モードを選択すると、関連するオルタネート機能と GPIO モードでピンが自動的に設定されます。

- **GPIO Pull-up/Pull-down**

  デフォルト値に設定されており、他に選択可能な値がある場合はその値に設定できます。

- **Maximum output speed**（通信ペリフェラル専用）

  消費電力を最適化するために、デフォルトでは［Low］に設定されていますが、アプリケーション要件に適合するように高い周波数に変更できます。

- **User Label**

  デフォルト名（GPIO_input など）をユーザ定義の名前に変更します。**[Pinout]** ビューが必要に応じて更新されます。**[Find]** メニューには、ここで指定した新しい名前でこの GPIO が表示されます。

**[Group by Peripherals]** チェックボックスを使用すると、ペリフェラルのすべてのインスタンスを同じウィンドウにグループ化できます（図 64 を参照）。

**図 64. ペリフェラル別にグループ化した GPIO 設定**

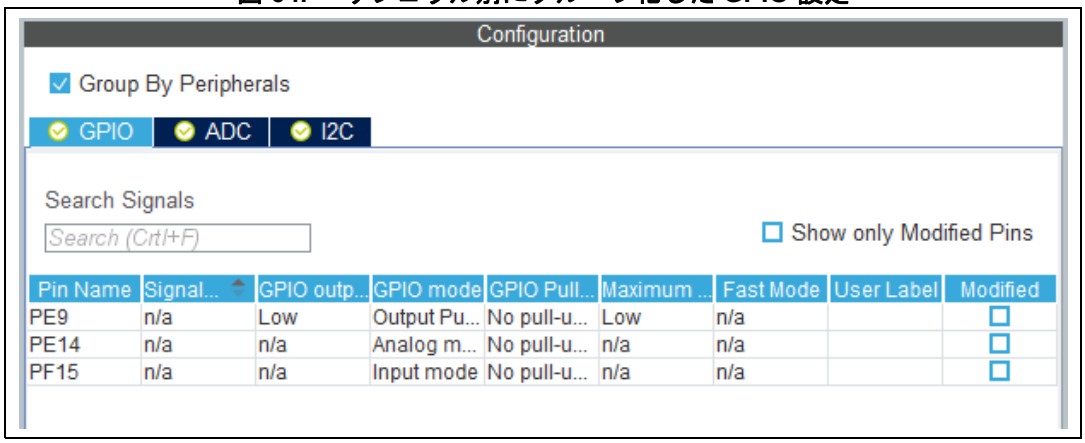
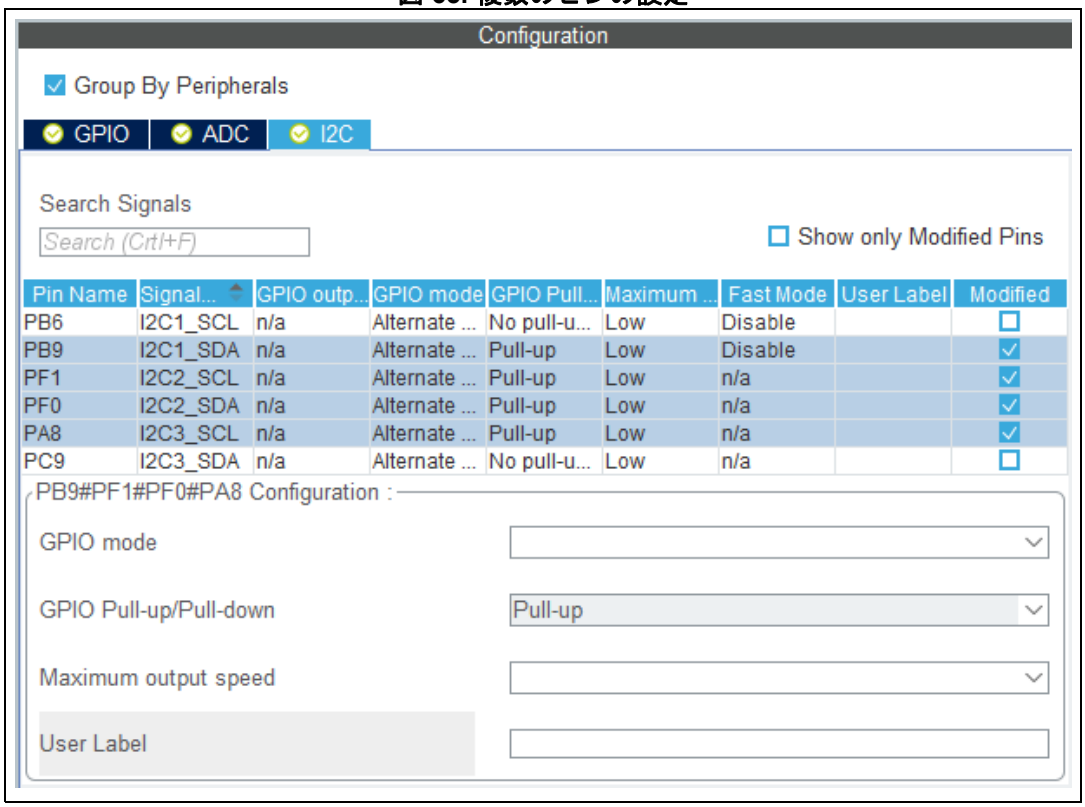



図 65 に示すように、複数行を選択して、複数のピンの設定を指定の内容に同時に変更できます。

**図 65. 複数のピンの設定**



### 4.4.13 DMA 設定ウィンドウ

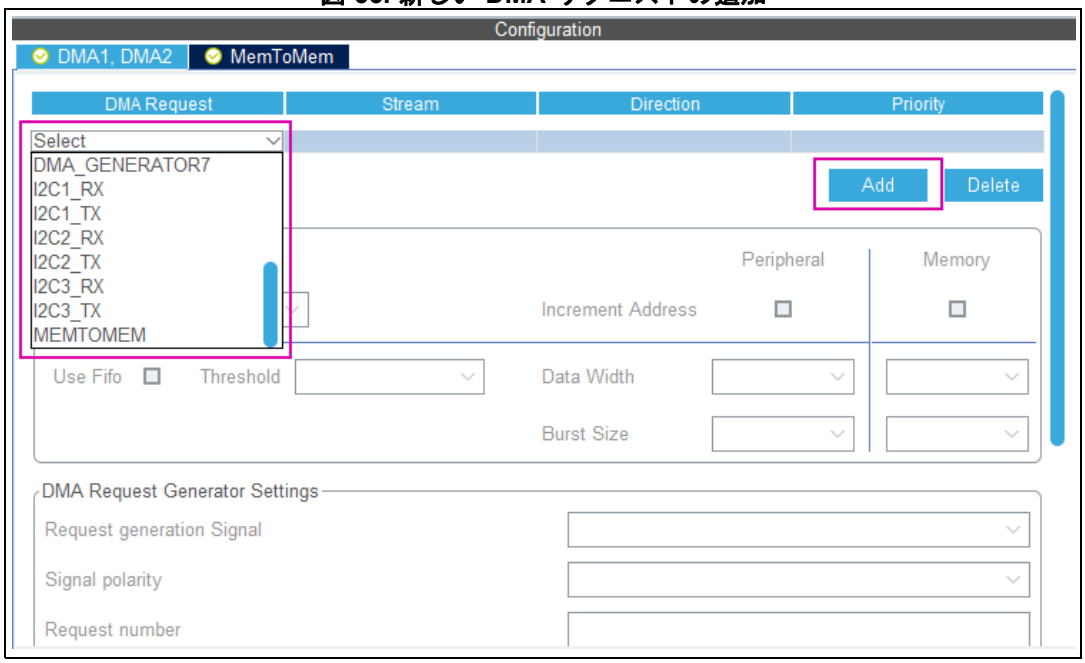
［System］ビューで［DMA］をクリックすると、DMA 設定ウィンドウが開きます。

このウィンドウでは、マイクロコントローラで使用できる汎用 DMA コントローラを設定できます。DMA インタフェースを使用すると、CPU が動作しているときでもメモリとペリフェラルの間でデータを転送できます。メモリ間の転送がサポートされていれば、そのような転送も可能です。

注： **ペリフェラルの中には、USB や Ethernet のように専用の DMA コントローラを備えたものがあります。このようなコントローラは、デフォルトで有効になっているか、ペリフェラルの［Configuration］ウィンドウで有効にします。**

**DMA** 設定ウィンドウで［**Add**］をクリックすると、DMA 設定ウィンドウの末尾に新しい行が追加されます。そこに、ペリフェラル信号にマッピングできる **DMA リクエスト**のコンボ･ボックスが表示され、そこからの選択が提案されます（図 66 を参照）。

**図 66. 新しい DMA リクエストの追加**

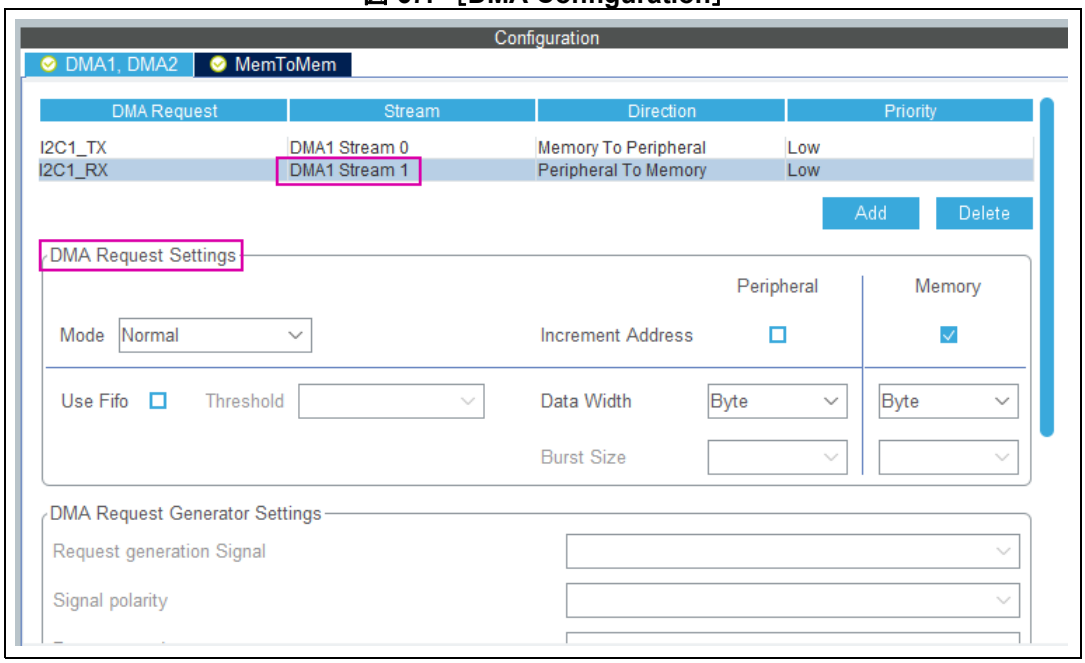



DMA リクエストを選択すると、使用可能なすべてのストリームのうちの 1 つ、その方向、および優先度が DMA リクエストに自動的に割り当てられます。DMA チャネルを設定している場合は、開始アドレスなどの DMA 転送のランタイム･パラメータは、アプリケーション･コード側で全面的に記述する必要があります。

DMA リクエスト（STM32F4 マイクロコントローラではチャネルと呼びます）を使用して、ペリフェラルとメモリとの間でデータを転送するためにストリームを予約します（図 67 を参照）。ストリームの優先度を使用して、次に DMA 転送するストリームを決定します。

DMA コントローラはデュアル優先度システムに対応しています。このシステムでは、まずソフトウェア優先度が適用され、それが等しい場合は、ストリーム番号で決まるハードウェア優先度が適用されます。

図 67. ［DMA Configuration］

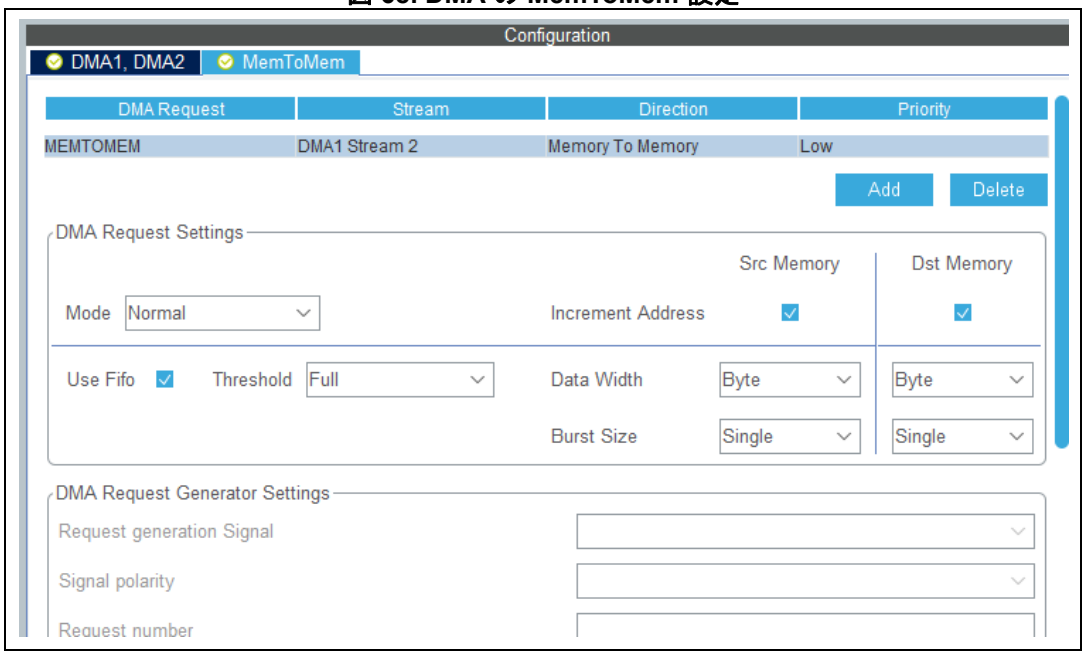



［**DMA Configuration**］ウィンドウでは、次の DMA 設定を実行できます。

- **Mode** : 通常モード、サーキュラ･モードまたはSDIO ペリフェラルでのみ使用可能なペリフェラル･フロー･コントローラ･モード。
- **Increment Address** : ペリフェラル･アドレスとメモリ･アドレスのインクリメントのタイプ。アドレス固定、または転送後にアドレスをインクリメントするポストインクリメントです。チェックボックスをチェックすると、その該当アドレスに対してポストインクリメント･モードが有効になります。
- **ペリフェラルのデータ幅** : 8、16 または 32 ビット
- デフォルトのダイレクト･モードから、プログラム可能な閾値に基づいて FIFO モードに切り替える機能 :
  a) ［**Use FIFO**］チェックボックスをチェックします。
  b) 次に、**ペリフェラルとメモリのデータ幅**（8 bit、16 bit、または 32 bit）を設定します。
  c) **シングル転送**または**バースト転送**を選択します。バースト転送を選択する場合は、バーストサイズ（1、4、8、または 16）を選択します。

メモリ間転送（MemToMem）の場合、DMA 設定は転送元メモリと転送先メモリに適用されます。

図 68. DMA の MemToMem 設定




## 4.4.14 NVIC 設定ウィンドウ

**[System]** ビューで **[NVIC]** をクリックすると、ネスト化されたベクタ割込みコントローラの設定ウィンドウが開きます（図 69 参照）。

割込みのマスク解除と割込みハンドラを次の 2 つのタブで管理します。

- **[NVIC]** タブでは、NVIC コントローラでのペリフェラル割込みを有効にして、その優先度を設定します。
- **[Code generation]** タブでは、割込みに関連するコードを生成する際のオプションを選択できます。

### ［NVIC］タブ･ビューを使用した割込みの有効化

**[NVIC]** ビュー（図 69 参照）には、使用できるすべての割込みが表示されるのではなく、**[Pinout & Configuration]** パネルで選択したペリフェラルで使用できる割込みのみが表示されます。システム割込みは表示されますが、無効にすることはできません。

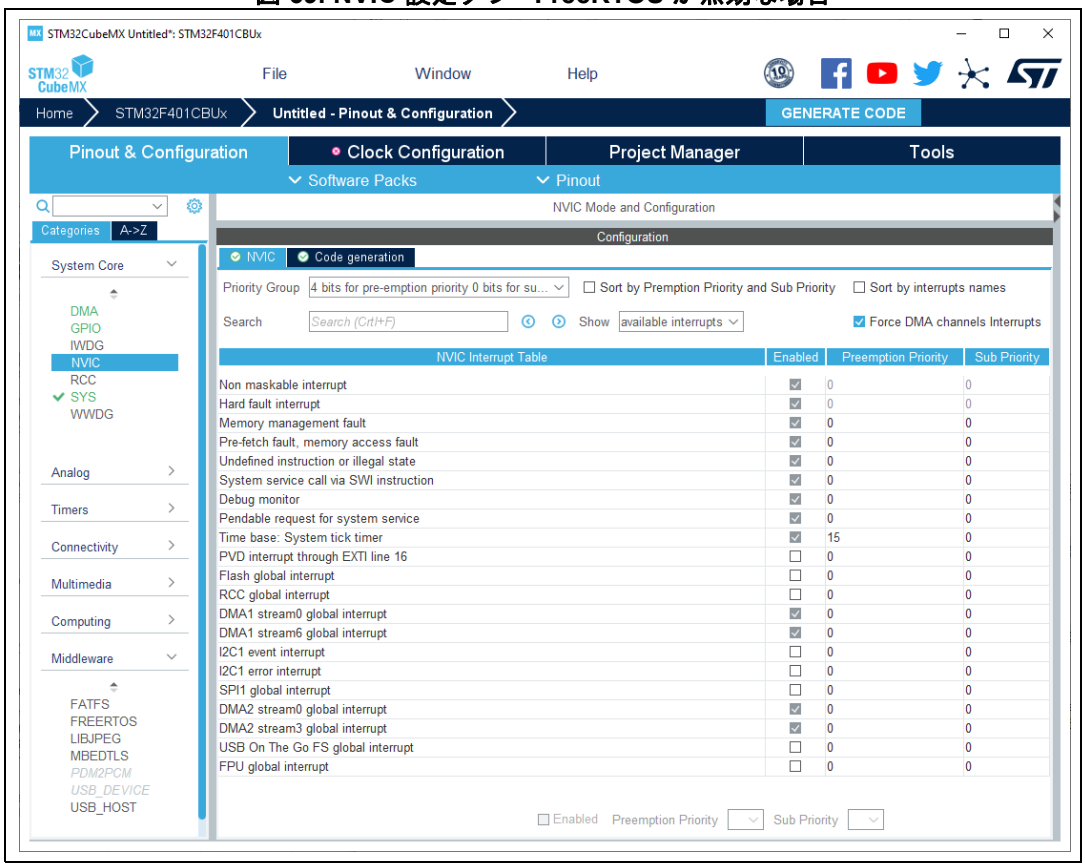
有効にした割込みのみを表示するには **[Show only enabled interrupts]** ボックスをチェックします。

プロジェクトで DMA チャネルを設定している場合、［Force DMA channels Interrupts］をチェックすると、生成したコードで DMA チャネル割込みが自動的に有効になります。

**[Search] フィールド**に文字列を入力すると、その値に従って割込みベクタをフィルタ処理できます。たとえば、**[Pinout]** パネルで UART ペリフェラルを有効にした後、NVIC の［Search］フィールドに「UART」と入力して、その横にある緑の矢印をクリックすると、すべての UART 割込みが表示されます。

**ペリフェラル割込み**を有効にすると、このペリフェラルに対し、NVIC 関数呼出しである **HAL_NVIC_SetPriority** と **HAL_NVIC_EnableIRQ** が生成されます。

図 69. NVIC 設定タブ - FreeRTOS が無効な場合

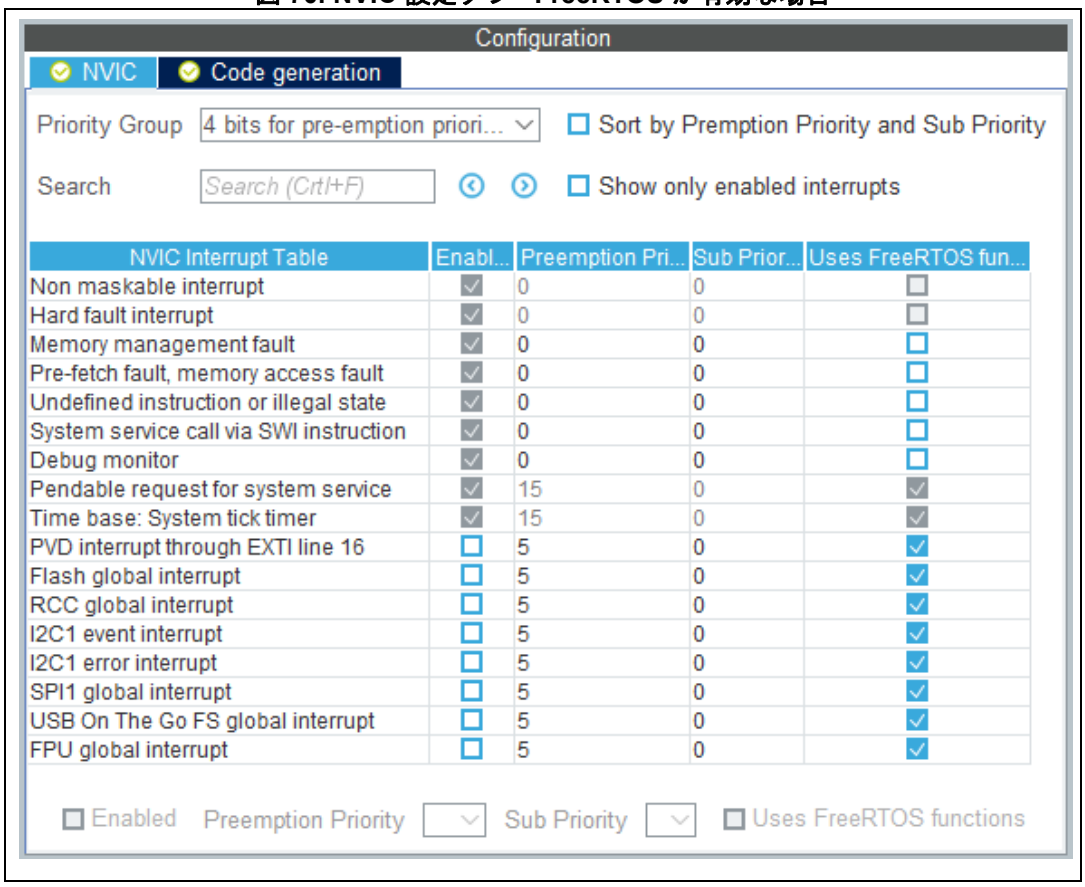



FreeRTOS が有効な場合、別の列が追加で表示されます（図 70 を参照）。

この場合、割込みの影響を受けない FreeRTOS API を呼び出すすべての割込みサービス･ルーチン（ISR）の優先度は、LIBRARY_MAX_SYSCALL_INTERRUPT_PRIORITY パラメータ（値が大きいほど優先度は低い）で定義されている優先度よりも低いことが必要です。対応するチェックボックスをチェックすることで、この制限が確実に適用されます。

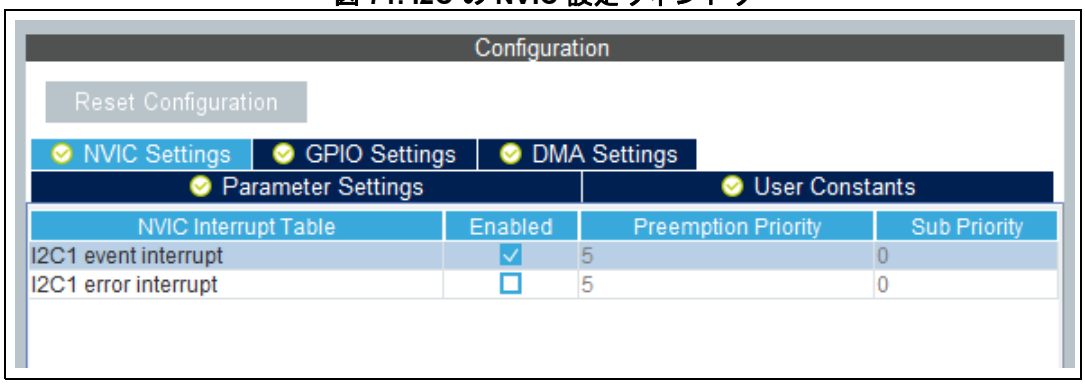
ISR でそのような関数を使用しない場合は、このチェックボックスのチェックを外し、任意の優先度レベルを設定できます。複数の行を同時にチェックまたはチェックを外すことができます（図 70 で青色にハイライトされている行を参照してください）。

図 70. NVIC 設定タブ - FreeRTOS が有効な場合




ペリフェラル専用割込みには、設定ウィンドウの［NVIC］ウィンドウからアクセスすることもできます（図 71 参照）。

図 71. I2C の NVIC 設定ウィンドウ



STM32CubeMX の NVIC 設定は、優先順位グループの選択、割込みの有効化と無効化、および割込み優先レベル（横取り優先順位レベルとサブ優先順位レベル）の設定で構成されます。

1. **優先度グループ**を選択します。

   いくつかのビットを使用して NVIC 優先度レベルを定義します。これらのビットは、横取り優先順位とサブ優先順位に相当する 2 つの優先順位グループに分割されます。たとえば、STM32F4 マイクロコントローラの場合、NVIC 優先順位グループ 0 は、0 bit プリエンプションと 4 bit のサブ優先順位に相当します。

2. 割込みテーブルで 1 行または複数の行をクリックして、1 つまたは複数の割込みベクタを選択します。割込みテーブルの下のウィジェットを使用して、1 つずつまたは一括してベクタを設定します。
   - **［Enabled］チェックボックス**：チェックすると割込みが有効になり、チェックを外すと無効になります。
   - **［Preemption Priority］**：横取り優先順位のレベルを選択します。横取り優先順位は、ある割込みが他の割込みに割り込むことができるかどうかを定義します。
   - **［Sub Priority］**：サブ優先順位のレベルを選択します。サブ優先順位は、割込みの優先順位レベルを定義します。

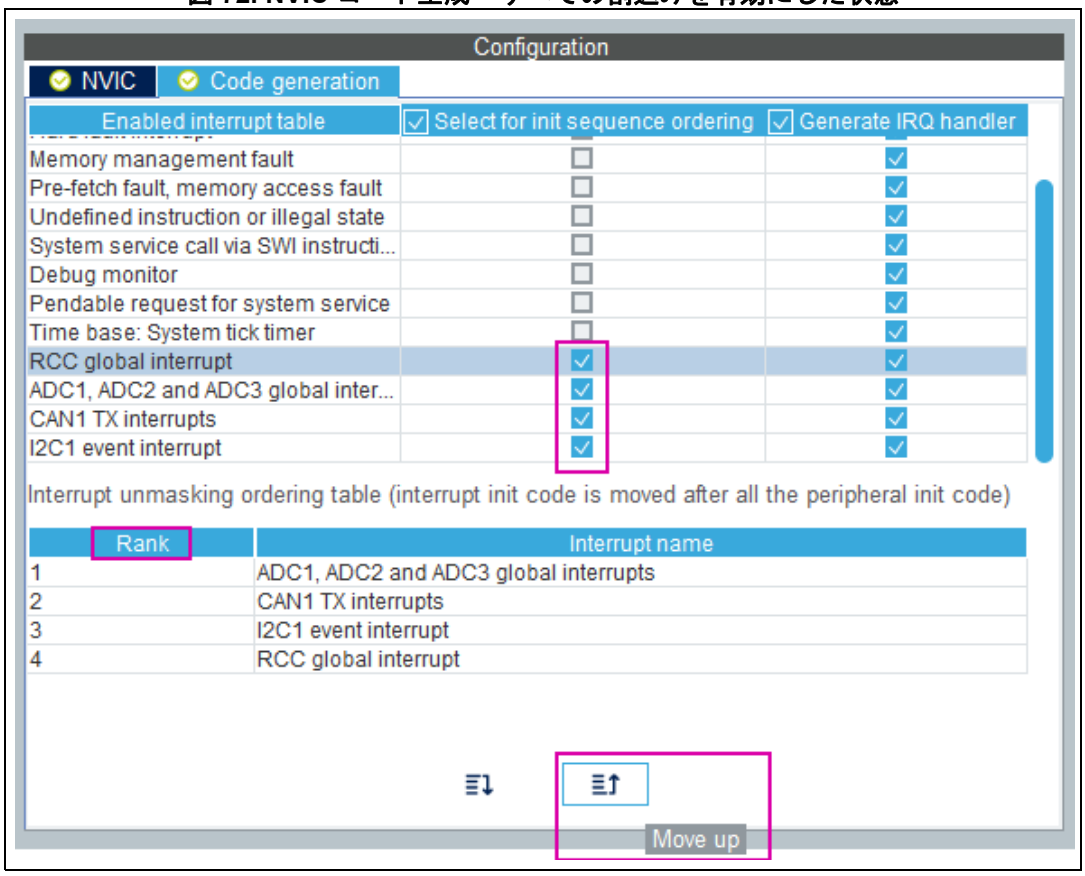
### 割込み処理のコード生成オプション

**［Code Generation］**ビューでは、割込み初期化と割込みハンドラ向けに生成するコードをカスタマイズできます。

- **シーケンスの順序指定と IRQ ハンドラのコード生成のすべての割込みに対して指定する方法または指定しない方法**

  列名の先頭にあるチェックボックスを使用すると、すべての割込みを一括して設定できます（図 72 を参照）。システム割込みは、ソフトウェア･ソリューションで制御できないので、初期化シーケンスの順序変更で扱うことはできません。

図 72. NVIC コード生成 – すべての割込みを有効にした状態




- **割込みのデフォルトの初期化シーケンス**

  デフォルトでは、GPIO の設定とペリフェラルのクロックの有効化が完了した後、ペリフェラル MSP の初期化関数の中で割込みが有効になります。

  次の CAN の例は、このシーケンスを示しています。この例では、stm32xxx_hal_msp.c ファイルの中で、ペリフェラルの msp_init 関数から HAL_NVIC_SetPriority 関数と HAL_NVIC_EnableIRQ 関数を呼び出しています。

  割込みを有効にするコードは太字で示されています。

```
void HAL_CAN_MspInit(CAN_HandleTypeDef* hcan)
{
GPIO_InitTypeDef GPIO_InitStruct;
if(hcan->Instance==CAN1)
{
/* ペリフェラル･クロック有効 */
__CAN1_CLK_ENABLE();
/**CAN1 GPIO Configuration
PD0      ------> CAN1_RX
PD1      ------> CAN1_TX
*/
GPIO_InitStruct.Pin = GPIO_PIN_0|GPIO_PIN_1;
GPIO_InitStruct.Mode = GPIO_MODE_AF_PP;
```

```
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_VERY_HIGH;
  GPIO_InitStruct.Alternate = GPIO_AF9_CAN1;
  HAL_GPIO_Init(GPIOD, &GPIO_InitStruct);

/* Peripheral interrupt init */
  HAL_NVIC_SetPriority(CAN1_TX_IRQn, 2, 2);
  HAL_NVIC_EnableIRQ(CAN1_TX_IRQn);
  }
 }
```

**EXTI GPIO** についてのみ、MX_GPIO_Init 関数の内部で割込みを有効にします。

```
/* GPIO ピンの設定：MEMS_INT2_Pin */
 GPIO_InitStruct.Pin = MEMS_INT2_Pin;
 GPIO_InitStruct.Mode = GPIO_MODE_EVT_RISING;
 GPIO_InitStruct.Pull = GPIO_NOPULL;
 HAL_GPIO_Init(MEMS_INT2_GPIO_Port, &GPIO_InitStruct);

 /* EXTI interrupt init */
 HAL_NVIC_SetPriority(EXTI15_10_IRQn, 0, 0);
 HAL_NVIC_EnableIRQ(EXTI15_10_IRQn);
```

ペリフェラルによっては、割込みを実際に有効にするために、アプリケーション側で別の関数呼び出しを必要とするものがあります。たとえば、タイマ･ペリフェラルの場合、割込みモードでタイマ入力キャプチャ（IC）測定を開始するには、HAL_TIM_IC_Start_IT 関数を呼び出す必要があります。

- **割込み初期化シーケンスの設定**

  複数のペリフェラルについて［**Select for Init sequence ordering**］をチェックすると、ペリフェラルごとの HAL_NVIC 関数呼出しが同一の専用関数 **MX_NVIC_Init**（main.c で定義します）に移動します。さらに、各ペリフェラルに対する HAL_NVIC 関数は、［**Code generation**］ビューの下部で指定した順序で呼び出されます（図 73 を参照）。

  たとえば、図 73 に示すように設定すると、次のコードが生成されます。

```
/** NVIC の設定
*/
void MX_NVIC_Init(void)
{
 /* CAN1_TX_IRQn 割込み設定 */
 HAL_NVIC_SetPriority(CAN1_TX_IRQn, 2, 2);
 HAL_NVIC_EnableIRQ(CAN1_TX_IRQn);
 /* PVD_IRQn 割込み設定 */
 HAL_NVIC_SetPriority(PVD_IRQn, 0, 0);
 HAL_NVIC_EnableIRQ(PVD_IRQn);
 /* FLASH_IRQn 割込み設定 */
 HAL_NVIC_SetPriority(FLASH_IRQn, 0, 0);
 HAL_NVIC_EnableIRQ(CAN1_IRQn);
 /* RCC_IRQn 割込み設定 */
 HAL_NVIC_SetPriority(RCC_IRQn, 0, 0);
```

```
  HAL_NVIC_EnableIRQ(CAN1_IRQn);
  /* ADC_IRQn 割込み設定 */
  HAL_NVIC_SetPriority(ADC_IRQn, 0, 0);
  HAL_NVIC_EnableIRQ(ADC_IRQn);
}
```

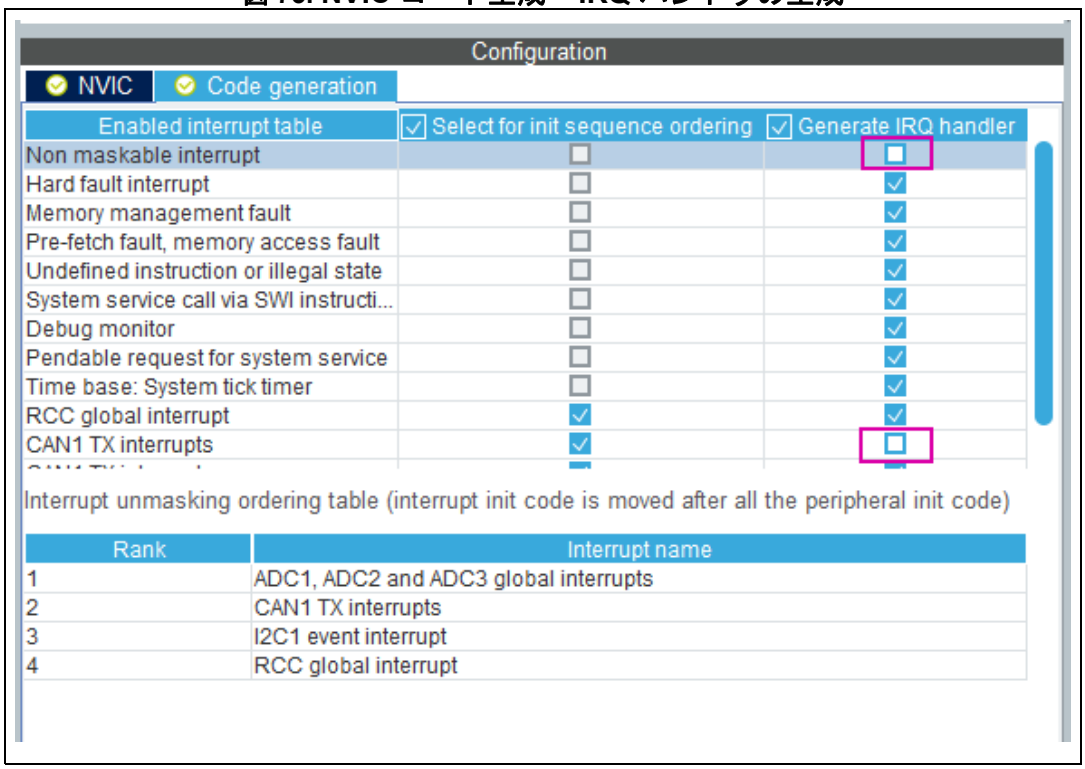
- **割込みハンドラのコード生成**

  デフォルトでは、STM32CubeMX によって stm32xxx_it.c ファイルに割込みハンドラが生成されます。次に例を示します。

```
void NMI_Handler(void)
{
 HAL_RCC_NMI_IRQHandler();
}
void CAN1_TX_IRQHandler(void)
{
 HAL_CAN_IRQHandler(&hcan1);
}
```

  **[Generate IRQ Handler]** 列では、割込みハンドラ関数呼出しを生成するかどうかを制御できます。図 73 に示す **[Generate IRQ Handler]** 列で CAN1_TX 割込みと NMI 割込みのチェックを外すと、前述のコードが stm32xxx_it.c ファイルから削除されます。

**図 73. NVIC コード生成 – IRQ ハンドラの生成**

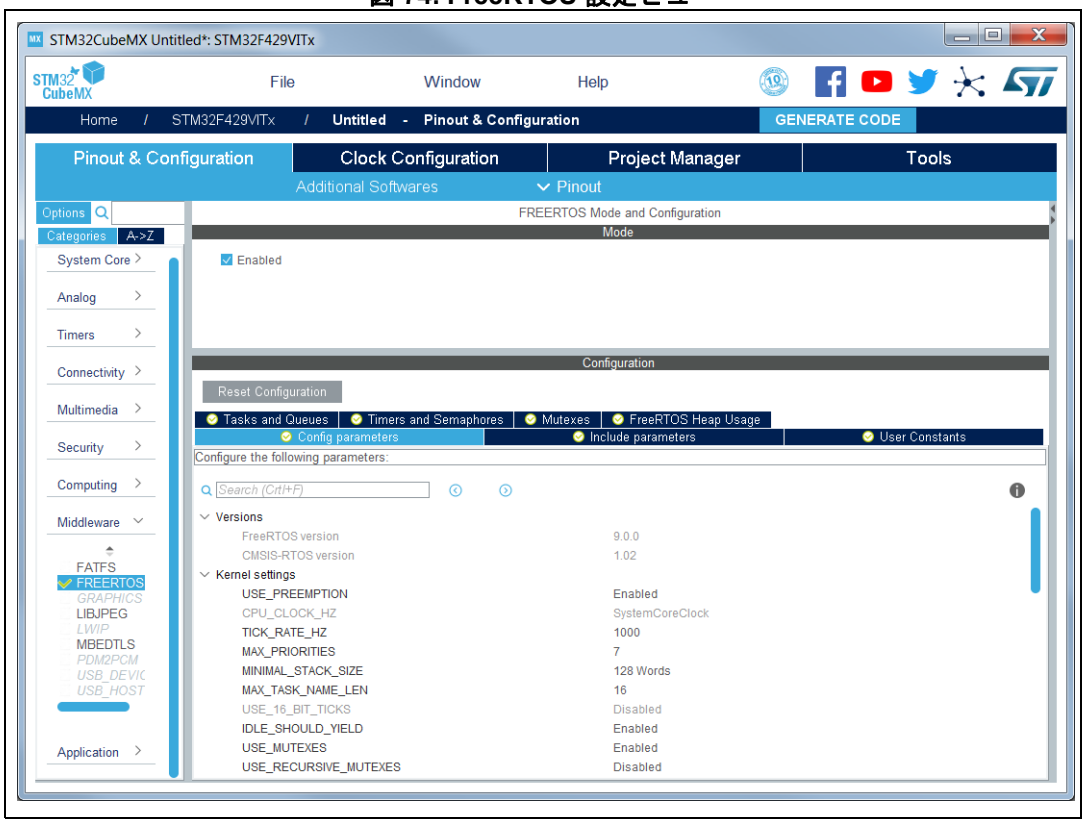


### 4.4.15 ［FreeRTOS Configuration］パネル

STM32CubeMX の［FreeRTOS Configuration］ウィンドウでは、リアルタイム OS アプリケーションに必要なすべてのリソースを設定し、それらのリソースに対応するヒープを予約できます。FreeRTOS の各要素は、生成されるコードの中で CMSIS-RTOS API 関数を使用して定義および作成されます。次の手順に従います。

1. **［Pinout & Configuration］**タブの［FREERTOS］をクリックして［Mode］パネルと［Configuration］パネルを表示します（図 74 参照）。
2. ［Mode］パネルで FreeRTOS を有効化します。
3. ［Configuration］パネルに移動して、FreeRTOS ネイティブのパラメータ、およびタスク、タイマ、キュー、セマフォなどのオブジェクトを設定します。［Config parameters］タブで、カーネルとソフトウェアを設定します。［Include parameters］タブでアプリケーションに必要な API 関数を選択します。これによってコード･サイズが最適化されます。設定パラメータとインクルード･パラメータはどちらも FreeRTOSConfig.h ファイルに記述されます。

**図 74. FreeRTOS 設定ビュー**



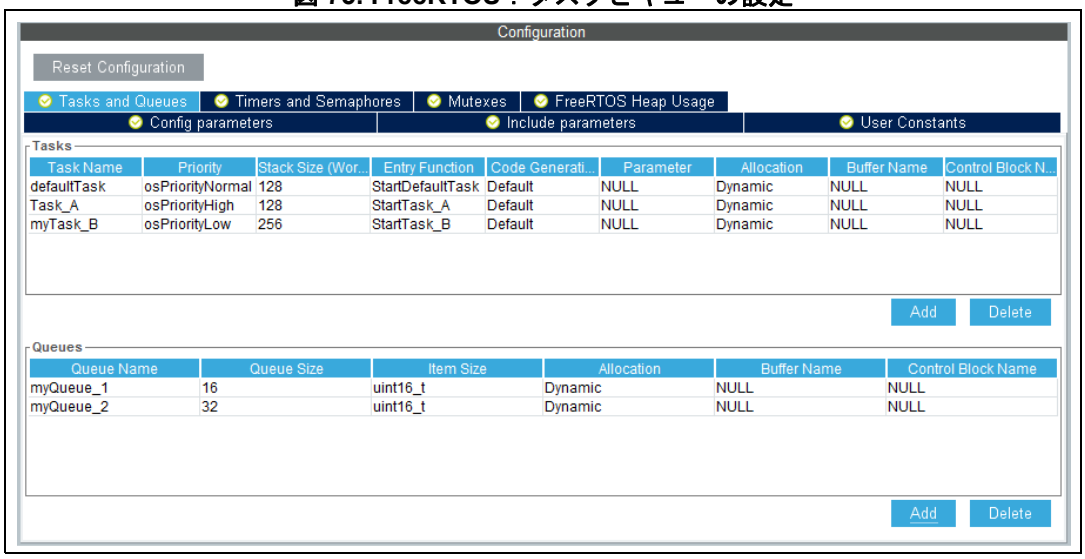
### ［Tasks and Queues］タブ

あらゆる RTOS と同様に、FreeRTOS はリアルタイム･アプリケーションを独立したタスク群として構造化し、一度に 1 つのタスクのみを実行します。キューは、タスク間通信に使用します。キューを使用して、タスクとタスクの間または割込みとタスクの間でメッセージを交換できます。

STM32CubeMX では、**FreeRTOS の［Tasks and Queues］**タブを使用して、そのようなタスクとキューを作成および設定できます（図 75 参照）。これに対応する初期化コードは main.c に生成されますが、**［Project Settings］**メニューで「ペリフェラルとミドルウェアごとに .c ファイルと .h ファイルのペアとしてコードを生成する」オプションを設定している場合は freeRTOS.c に生成されます。

これに対応する初期化コードは main.c に生成されますが、**［Project Manager］**メニューで「ペリフェラルとミドルウェアごとに .c ファイルと .h ファイルのペアとしてコードを生成する」オプションを設定している場合は freeRTOS.c に生成されます。

**図 75. FreeRTOS：タスクとキューの設定**




- Tasks

  **［Tasks］**セクションで**［Add］**ボタンをクリックすると**［New Task］**ウィンドウが開きます。ここでは、タスクの**名前**、**優先度**、**スタック･サイズ**、および**エントリ関数**を設定できます（図 76 を参照）。これらの設定はいつでも更新できます。タスクの行をダブルクリックすると［New Task］ウィンドウが再び開き、設定を編集できます。

  エントリ関数は、次のように weak または external として生成できます。

  – タスクを **weak** として生成すると、デフォルトで生成される定義とは異なる定義を指定できます。

  – タスクを **extern** とすると、その関数定義はユーザ側で用意する必要があります。

  生成される関数定義にはデフォルトでユーザ･セクションがあるのでカスタマイズが可能です。

- Queues

  **［Queues］**セクションで**［Add］**ボタンをクリックすると**［New Queue］**ウィンドウが開きます。ここでは、キューの**名前**、**サイズ**、および**項目サイズ**を設定できます（図 76 参照）。キュー･サイズは、キューに一度に保持できる項目の最大数に相当します。項目サイズは、キューに格納する各データ項目のサイズです。項目サイズは、次のようにバイト数またはデータ型として表現できます。

- 1 バイト：uint8_t 型、int8_t 型、char 型、および portCHAR 型
- 2 バイト：uint16_t 型、int16_t 型、short 型、および portSHORT 型
- 4 バイト：uint32_t 型、int32_t 型、int 型、long 型、および float 型
- 8 バイト：uint64_t 型、int64_t 型、および double 型

FreeRTOS によるヒープ使用量の計算機能では、ユーザ入力から自動的に項目サイズが得られない場合、デフォルトで 4 バイトが使用されます。

これらの設定はいつでも更新できます。キューの行をダブルクリックすると［New Queues］ウィンドウが再び開き、設定を編集できます。

**図 76. FreeRTOS：新しいタスクの作成**

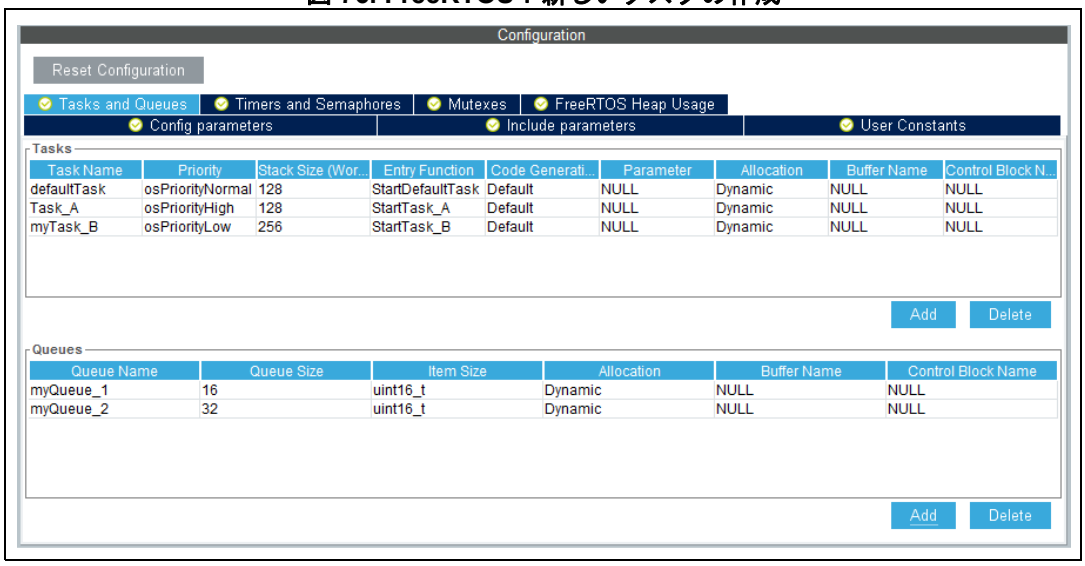


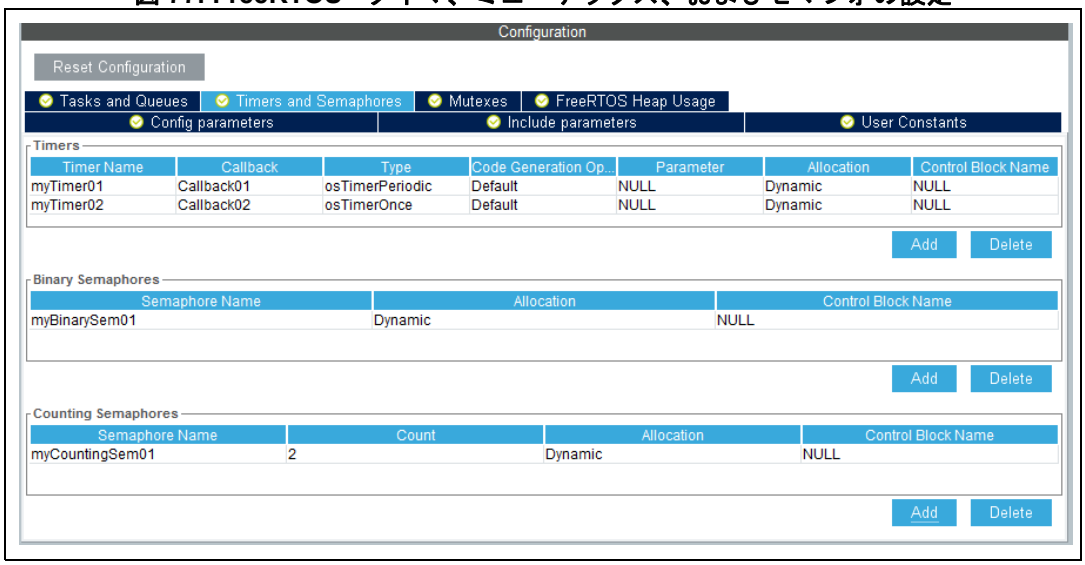
図 75 に基づいて生成されたコードを次に示します。

```
/* Create the thread(s) */
  /* defaultTask の定義と作成 */
  osThreadDef(defaultTask, StartDefaultTask, osPriorityNormal, 0, 128);
  defaultTaskHandle = osThreadCreate(osThread(defaultTask), NULL);

  /* Task_A の定義と作成 */
  osThreadDef(Task_A, StartTask_A, osPriorityHigh, 0, 128);
  Task_AHandle = osThreadCreate(osThread(Task_A), NULL);

  /* Task_B の定義と作成 */
  osThreadDef(Task_B, StartTask_B, osPriorityLow, 0, 256);
  Task_BHandle = osThreadCreate(osThread(Task_B), NULL);

  /* queue(s) の作成 */
  /* myQueue_1 の定義と作成 */
  osMessageQDef(myQueue_1, 16, 4);
  myQueue_1Handle = osMessageCreate(osMessageQ(myQueue_1), NULL);

  /* myQueue_2 の定義と作成 */
  osMessageQDef(myQueue_2, 32, 2);
  myQueue_2Handle = osMessageCreate(osMessageQ(myQueue_2), NULL);
```

## タイマ、ミューテックス、およびセマフォ

FreeRTOS のタイマ、ミューテックス、およびセマフォは、FreeRTOS の **[Timers and Semaphores]** タブで作成できます。まず、これらのオブジェクトを設定タブで有効にする必要があります（図 77 参照）。

**図 77. FreeRTOS - タイマ、ミューテックス、およびセマフォの設定**



各オブジェクトの専用セクションで、[Add]ボタンをクリックすると、[New <オブジェクト名>]ウィンドウが開きます。ここで、そのオブジェクト固有のパラメータを指定できます。オブジェクトの設定はいつでも変更できます。該当する行をダブルクリックすると[New <オブジェクト名>]ウィンドウが再び開き、設定を編集できます。

注： **新しく作成したオブジェクトを画面上で確認できない場合は、ウィンドウを拡張します。**

- タイマ

  タイマを作成する前に、[Config parameters]タブの[software timer definitions]セクションで、タイマを使用できるようにしておく必要があります（USE_TIMERS の定義）。同じセクションで、タイマ･タスクの優先度、キューの長さ、およびスタックの深さも設定できます。

  タイマは、1 回限り（1 回のみ実行）または自動再ロード（定期的に実行）として作成できます。タイマ名とそれに対応するコールバック関数名を指定する必要があります。このコールバック関数のコードの記述、および CMSIS-RTOS の osTimerStart 関数を呼び出す場合のタイマ期間（タイマを開始してからそのコールバック関数が実行されるまでの時間）の指定はユーザ側で行う必要があります。

- ミューテックス/セマフォ

  ミューテックス、再帰的ミューテックス、および計数セマフォを作成する前に、[Config parameters]タブの[Kernel settings]セクションで、それらを使用できるようにしておく必要があります（USE_MUTEXES、USE_RECURSIVE_MUTEXES、USE_COUNTING_SEMAPHORES の定義）。

  図 77 に基づいて生成されたコードを次に示します。

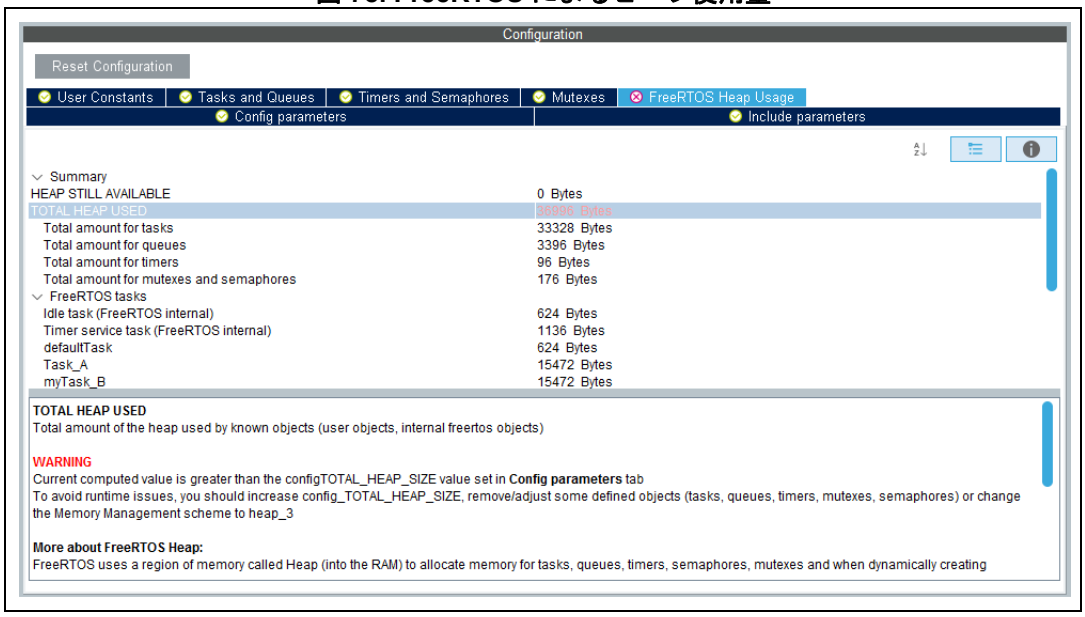
```
/*  semaphores(s) の作成 */
/* myBinarySem01 の定義と作成 */
osSemaphoreDef(myBinarySem01);
myBinarySem01Handle = osSemaphoreCreate(osSemaphore(myBinarySem01), 1);

/* myCountingSem01 の定義と作成 */
osSemaphoreDef(myCountingSem01);
myCountingSem01Handle = osSemaphoreCreate(osSemaphore(myCountingSem01),
7);

  /* タイマの作成 */
/* myTimer01 の定義と作成 */
osTimerDef(myTimer01, Callback01);
myTimer01Handle = osTimerCreate(osTimer(myTimer01), osTimerPeriodic,
NULL);

/* myTimer02 の定義と作成 */
osTimerDef(myTimer02, Callback02);
myTimer02Handle = osTimerCreate(osTimer(myTimer02), osTimerOnce, NULL);

/* ミューテックスの作成 */
/* myMutex01 の定義と作成 */
osMutexDef(myMutex01);
myMutex01Handle = osMutexCreate(osMutex(myMutex01));
```

```
  /* 再帰的ミューテックスの作成 */
  /* myRecursiveMutex01 の定義と作成 */
  osMutexDef(myRecursiveMutex01);
  myRecursiveMutex01Handle =
osRecursiveMutexCreate(osMutex(myRecursiveMutex01));
```

### FreeRTOS によるヒープ使用量

[**FreeRTOS Heap Usage**] タブには、現在使用されているヒープが表示され、[**Config Parameters**] タブで設定した TOTAL_HEAP_SIZE パラメータと比較されます。この合計ヒープ使用量は、最大閾値 TOTAL_HEAP_SIZE を超えると赤紫色で表示され、タブに赤紫色の×が表示されます（図 78 参照）。

**図 78. FreeRTOS によるヒープ使用量**




## 4.4.16 HAL タイムベース･ソースの設定

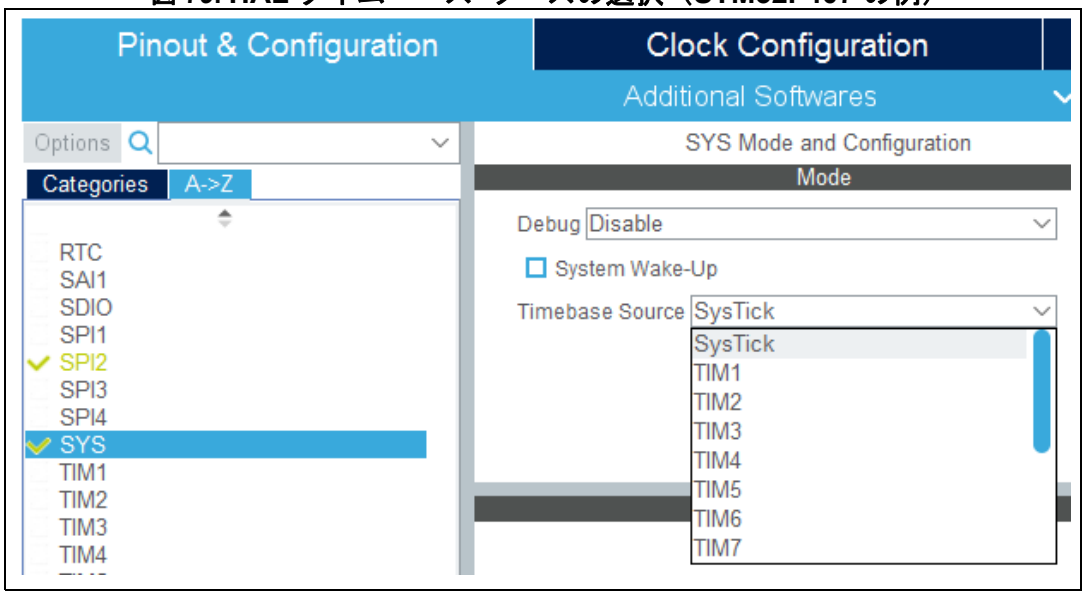
デフォルトの STM32Cube HAL は、独自のタイムベース･ソース Arm® Cortex® システム･タイマ（SysTick）を基準に構築されます。

なお、HAL タイムベース関連の関数には weak 属性が定義されているので、このタイムベース･ソースをオーバーロードして別のハードウェア･タイムベース･ソースを使用できます。RTOS を使用するアプリケーションでは、HAL タイムベース･ソースの使用を強くお勧めします。その理由は、このミドルウェアは SysTick 設定（ティックと優先度）に対する全面的な制御機能を有し、ほとんどの RTOS では SysTick の優先度が強制的に最も低く設定されるからです。

アプリケーションが HAL プログラミング･モデルに従っているのであれば、SysTick を使用することもできます。これは、デッド･ロックの問題が発生しないように、割込みサービス要求のコンテキストの範囲では HAL タイムベース･サービスをアプリケーションから呼び出さないということです。

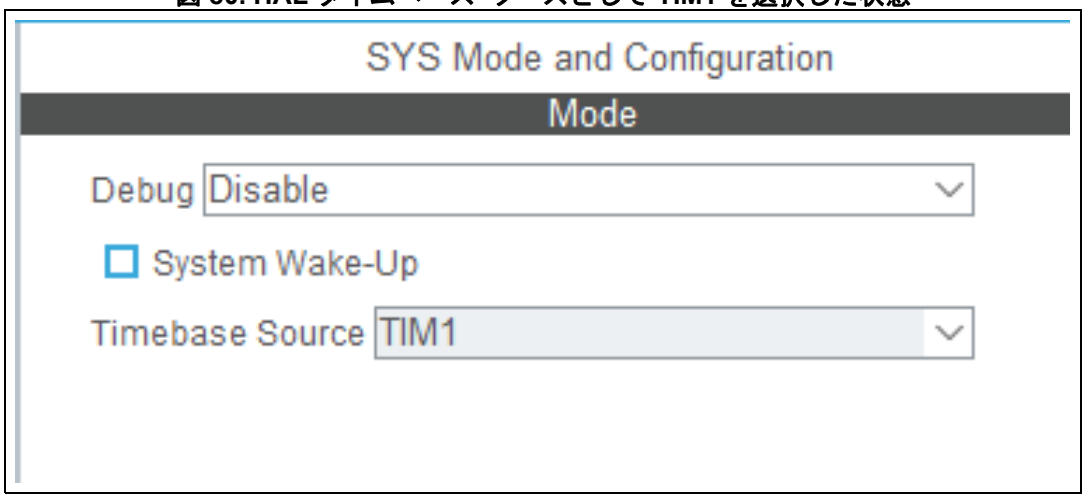
HAL タイムベース･ソースを変更するには、**コンポーネント･リスト**のパネルで SYS ペリフェラルに移動して、使用可能なクロック･ソース（SysTick、TIM1、TIM2 など）からクロックを選択します（図 79 参照）。

図 79. HAL タイムベース･ソースの選択（STM32F407 の例）




タイムベース･ソースとして使用されているペリフェラルは灰色で表示され、選択できなくなります（図 80 を参照）。

図 80. HAL タイムベース･ソースとして TIM1 を選択した状態

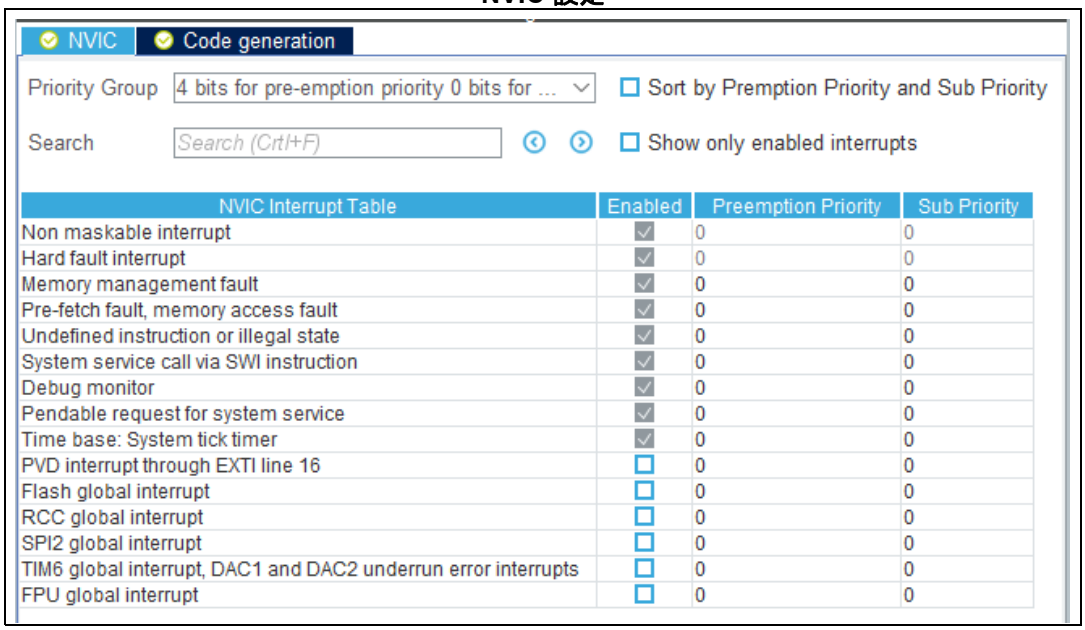


次の例に示すように、HAL タイムベース･ソースの選択と FreeRTOS の使用は、生成されるコードに影響します。

### SysTick を使用して FreeRTOS を使用しない設定の例

図 81に示すように、SysTick を使用して FreeRTOS を使用しない場合、SysTick の優先度は 0 （高）に設定されます。

**図 81. HAL タイムベースとして SysTick を使用して FreeRTOS を使用しない場合の NVIC 設定**




| NVIC Interrupt Table | Enabled | Preemption Priority | Sub Priority |
|---|---|---|---|
| Non maskable interrupt | ☑ | 0 | 0 |
| Hard fault interrupt | ☑ | 0 | 0 |
| Memory management fault | ☑ | 0 | 0 |
| Pre-fetch fault, memory access fault | ☑ | 0 | 0 |
| Undefined instruction or illegal state | ☑ | 0 | 0 |
| System service call via SWI instruction | ☑ | 0 | 0 |
| Debug monitor | ☑ | 0 | 0 |
| Pendable request for system service | ☑ | 0 | 0 |
| Time base: System tick timer | ☑ | 0 | 0 |
| PVD interrupt through EXTI line 16 | ☐ | 0 | 0 |
| Flash global interrupt | ☐ | 0 | 0 |
| RCC global interrupt | ☐ | 0 | 0 |
| SPI2 global interrupt | ☐ | 0 | 0 |
| TIM6 global interrupt, DAC1 and DAC2 underrun error interrupts | ☐ | 0 | 0 |
| FPU global interrupt | ☐ | 0 | 0 |

この設定に応じて、次のように割込み優先度（main.c）とハンドラ･コード（stm32f4xx_it.c）が生成されます。

- main.c ファイル

```
/* SysTick_IRQn interrupt configuration */
  HAL_NVIC_SetPriority(SysTick_IRQn, 0, 0);
```

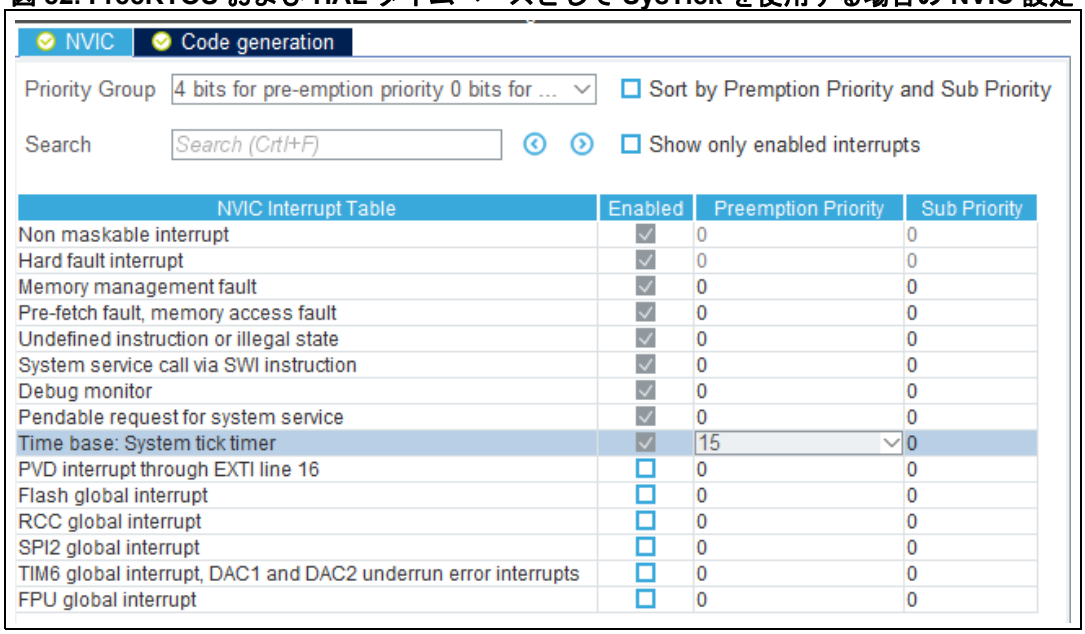
- stm32f4xx_it.c ファイル

```
/**
* @brief この関数でシステム･ティック･タイマを処理
*/
void SysTick_Handler(void)
{
  /* ユーザ･コードの開始 SysTick_IRQn 0 */
 /* ユーザ･コードの終了 SysTick_IRQn 0 */
  HAL_IncTick();
  HAL_SYSTICK_IRQHandler();
  /* ユーザ･コードの開始 SysTick_IRQn 1 */

  /* ユーザ･コードの終了 SysTick_IRQn 1 */
}
```

### SysTick と FreeRTOS を使用する設定の例

図 82に示すように、SysTick と FreeRTOS を使用する場合、SysTick の優先度は 15 （低）に設定されます。

**図 82. FreeRTOS および HAL タイムベースとして SysTick を使用する場合の NVIC 設定**




次のコードに示すように、SysTick 割込みハンドラは、CMSIS-os の osSystickHandler 関数を使用するように更新されます。

- main.c ファイル

```
/* SysTick_IRQn 割込み設定 */
HAL_NVIC_SetPriority(SysTick_IRQn, 15, 0);
```

- stm32f4xx_it.c ファイル

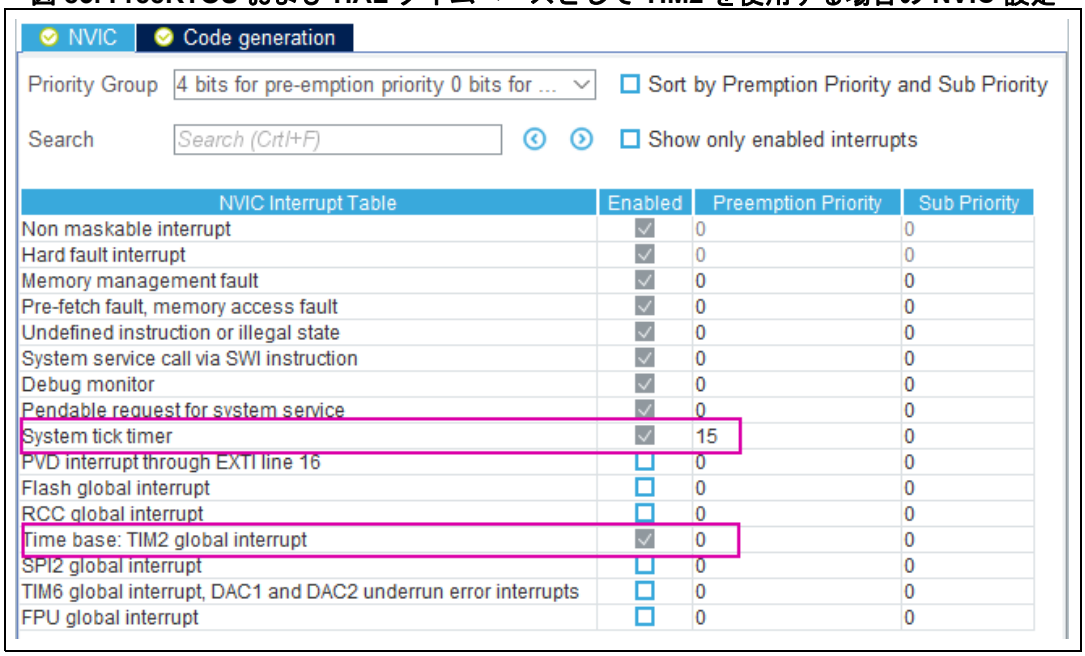
```
/**
* @brief この関数でシステム･ティック･タイマを処理
*/
void SysTick_Handler(void)
{
  /* ユーザ･コードの開始 SysTick_IRQn 0 */

  /* ユーザ･コードの終了 SysTick_IRQn 0 */
  HAL_IncTick();
  osSystickHandler();
  /* ユーザ･コードの開始 SysTick_IRQn 1 */

  /* ユーザ･コードの終了 SysTick_IRQn 1 */
}
```

### HAL タイムベース･ソースとして TIM2 を使用する設定の例

HAL タイムベース･ソースとして TIM2 を使用する場合、HAL タイムベース関連の関数（TIM2 を HAL タイムベース･ソースとして設定する HAL_InitTick 関数など）をオーバーロードする stm32f4xx_hal_timebase_TIM.c ファイルが新たに生成されます。

TIM2 タイムベースの割込み優先度は 0 （高）に設定されます。SysTick の優先度は、FreeRTOS を使用する場合は 15（低）、使用しない場合は 0（高）に設定されます。

**図 83. FreeRTOS および HAL タイムベースとして TIM2 を使用する場合の NVIC 設定**




この設定に応じて、次のように stm32f4xx_it.c ファイルが生成されます。

- SysTick_Handler は、FreeRTOS を使用する場合は osSystickHandler を呼び出し、使用しない場合は HAL_SYSTICK_IRQHandler を呼び出します。
- TIM2 グローバル割込みを処理するために、TIM2_IRQHandler を生成します。

## 4.5 STM32MP1 シリーズの［Pinout & Configuration］ビュー

STM32MP1 シリーズの **［Pinout & Configuration］**ビューでは、以下を実行できます。

- コンポーネントを 1 つまたは複数のランタイムコンテキストに割り当てる
- ペリフェラルをブート･デバイスとして設定する
- ブート･ローダで管理するペリフェラルを選択する
- GPIO を 1 つのランタイムコンテキストに割り当てる（図 85 参照）

これらの機能を次の 2 種類のパネルに用意しています（図 84 参照）。

1. コンポーネント･ツリー･パネル：サポート対象のすべてのペリフェラルとミドルウェアが一覧表示されます（［Show contexts］オプションを有効にする必要があります）。
2. 各コンポーネントの［Mode］パネル：コンポーネント名をクリックすると、このパネルが開きます。

図 84. STM32MP1 のブート･デバイスとランタイムコンテキスト

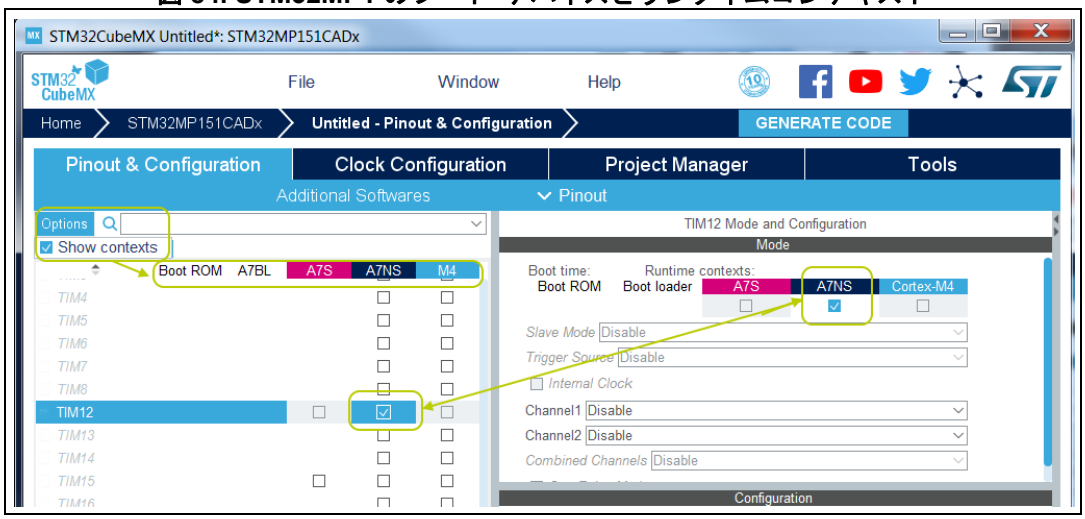
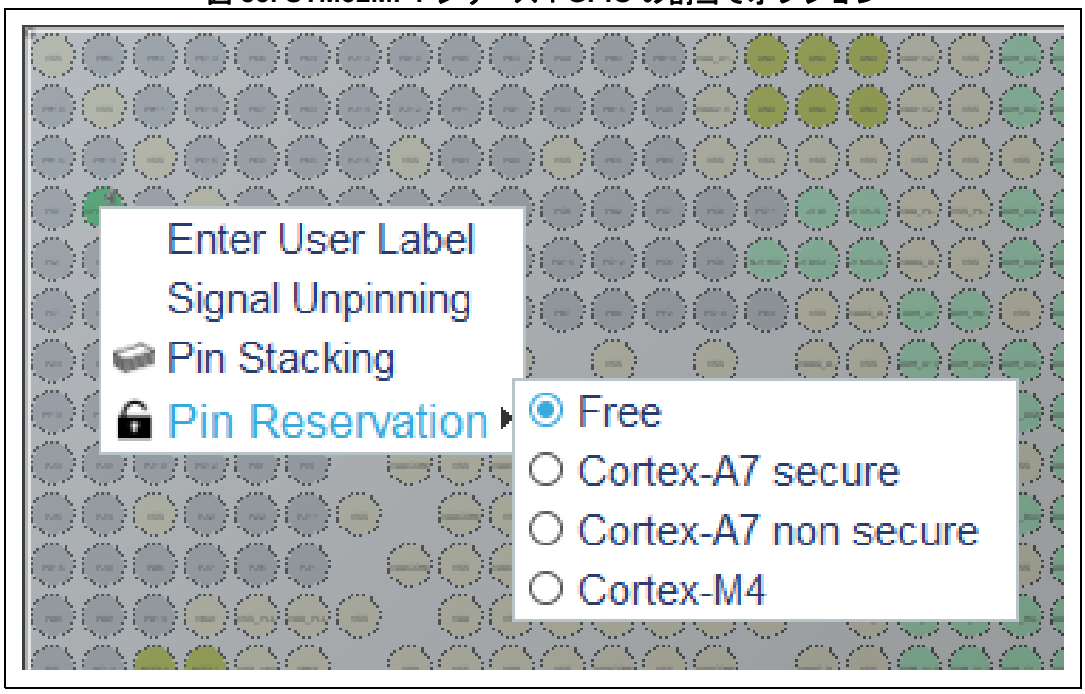



図 85. STM32MP1 シリーズ：GPIO の割当てオプション




### 4.5.1 実行時の設定

STM32MP1 は、マルチコア（Arm® Cortex®-A7 デュアルコアおよび Cortex®-M4）でマルチファームウェアのデバイスです。各ファームウェアは、これらのコアの 1 つで実行されます。ファームウェアとコアを関連付けて、ファームウェアのコードを実行するランタイムコンテキストを定義します。

次の 3 つのランタイムコンテキストを使用できます。

1. Linux カーネルを実行する非セキュアな Cortex-A7
2. SP_min を実行するセキュアな Cortex-A7
3. STM32Cube のファームウェアを実行する Cortex-M4

コンポーネントをランタイムコンテキストに割り当てるということは、実行時にそのコンポーネントを制御するコンテキストを指定することです。Cortex-A7 コンテキストへの割当ては、デバイス･ツリーのコード生成に反映されます。一方、Cortex-M4 コンテキストへの割当ては、STM32Cube ベースの C コード生成に反映されます（詳細は、コード生成のセクションを参照してください）。

コンテキストへのコンポーネントの割当ては、コンテキスト専用の列で指定します。

## 4.5.2 ブート段階の設定

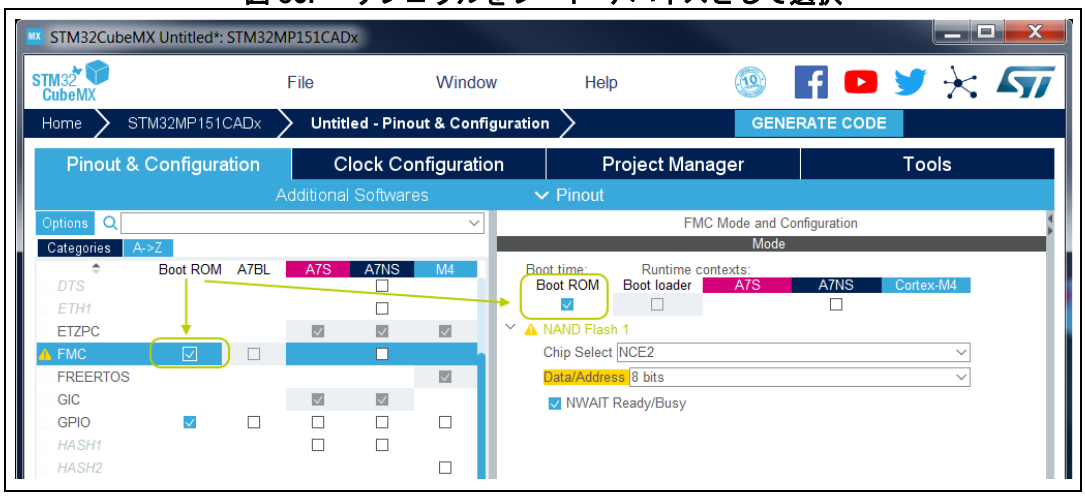
### ブート ROM ペリフェラルの選択

マイクロプロセッサを起動して動作状態にするには複数の実行段階が必要です。

まず、ROM に組み込まれたバイナリ･コードを実行します。デフォルト設定を使用して、ブート検出に関与するクロック･ツリーとすべてのペリフェラルが初期化されます。

ブート ROM プログラムで管理されるペリフェラルをブート･デバイスとして選択できます。これは［Boot ROM］列で選択します（図 86 参照）。

**図 86. ペリフェラルをブート･デバイスとして選択**




ブート･デバイスとして設定したペリフェラルでは特定のピン配置が使用されます。つまり、ブート ROM で認識できるピンに特定の信号を排他的にマッピングする必要があります。ブート ROM プログラムではこれらの信号とピンのみが考慮されます。

ROM ブート可能なペリフェラルに機能モードを設定すると、そのモードに関連するピン配置は、ブート ROM コードで使用される特定の信号のピンを除き、ランタイムコンテキストにおけるピン配置と同じになります。

ブート段階（ブート ROM コードの実行）では、そのペリフェラルは、すべてのピンのうち、ブート可能な信号とピンのみを使用して動作します。ブートが完了して実行段階に移行すると、そのペリフェラルは、選択された機能モードに必要なすべての信号を使用して動作します。

### ブート･ローダ（A7BL）ペリフェラルの選択

ボードの起動時、ファームウェア（セキュアな Cortex-A7 では SP_min、非セキュアな Cortex-A7 では Linux カーネル）が実行される各 Cortex-A7 ランタイムコンテキスト（セキュアと非セキュア）の起動に先立って、初期のブート段階が実行されます。この段階は DDR で U-Boot が再配置されるよりも前の時点です。

このブート･ローダの段階で管理できるデバイスは、ブート･ローダ（A7BL）の列で定義します。

この割当ては、生成される各種デバイス･ツリーに反映されます（詳細はコード生成のセクションを参照してください）。

## 4.6 STM32H7 デュアルコア製品ラインの［Pinout & Configuration］ビュー

STM32H7 製品ラインの中には、Arm Cortex-M7 コア、Arm Cortex-M4 コア、および 3 つの電源ドメインを備えた製品があります。

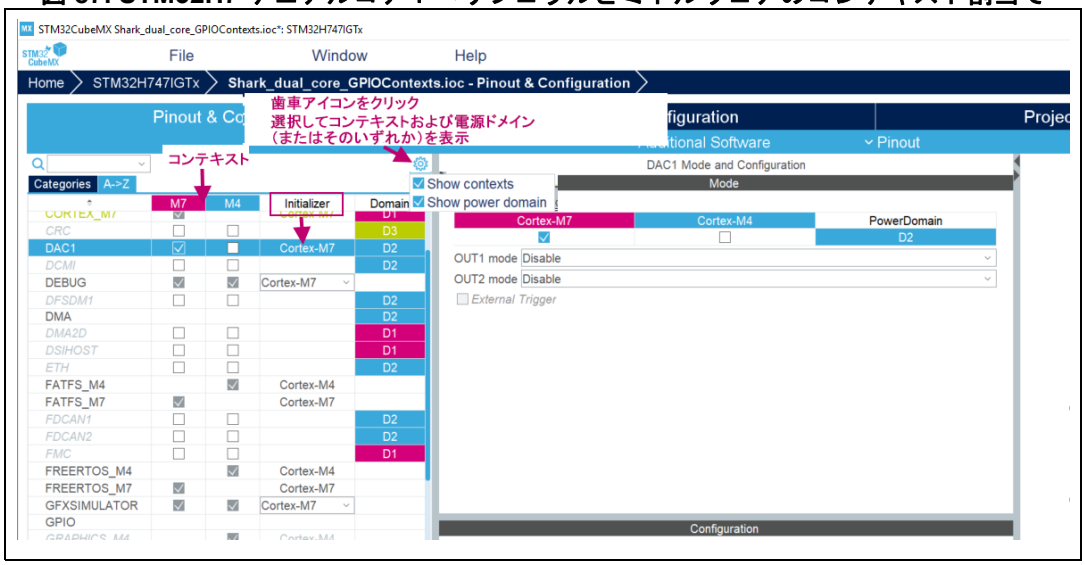
そのような製品の［**Pinout & Configuration**］ビューでは、以下を実行できます。

- ペリフェラルとミドルウェアのそれぞれを、一方のコアのコンテキスト、または可能であれば両方のコアのコンテキストに割り当てます。両方のコンテキストを選択した場合、どちらのコアでペリフェラルまたはミドルウェアの初期化関数を呼び出すかを指示するために、初期化を実行するコアを指定します。
- 各ペリフェラルが属する電源ドメインを表示します。
- GPIO をコアに割り当てるか、GPIO を必要とする可能性がある他のコンポーネントのために、この時点では GPIO をどこにも割り当てない状態にしておきます。この場合、GPIO を予約しているコンポーネントと同じコアで GPIO が初期化されます。

ペリフェラルとミドルウェアに対するこれらの設定は、次の 2 種類のパネルで指定できます。

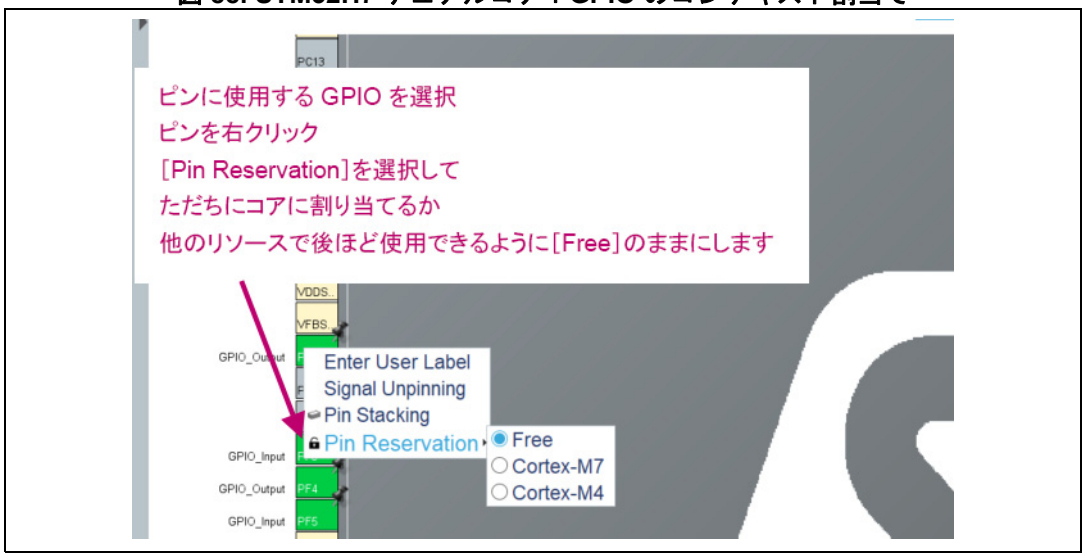
1. コンポーネント･ツリー･パネル：サポート対象のすべてのペリフェラルとミドルウェアが一覧表示されます（歯車のアイコンをクリックすると［Show contexts］オプションが有効になります）（図 87 参照）。
2. 各コンポーネントの［Mode］パネル：コンポーネント名をクリックすると、このパネルが開きます。

**図 87. STM32H7 デュアルコア：ペリフェラルとミドルウェアのコンテキスト割当て**




GPIO は（図 88参照）、［**Pinout**］ビューで直接割り当てるか、ミドルウェアのプラットフォーム設定パネルでの選択に従い、後ほど自動的に割り当てられます。

図 88. STM32H7 デュアルコア：GPIO のコンテキスト割当て




## 4.7 ［Pinout & Configuration］ビューでのセキュリティの有効化（STM32L5 シリーズと STM32U5 シリーズのみ）

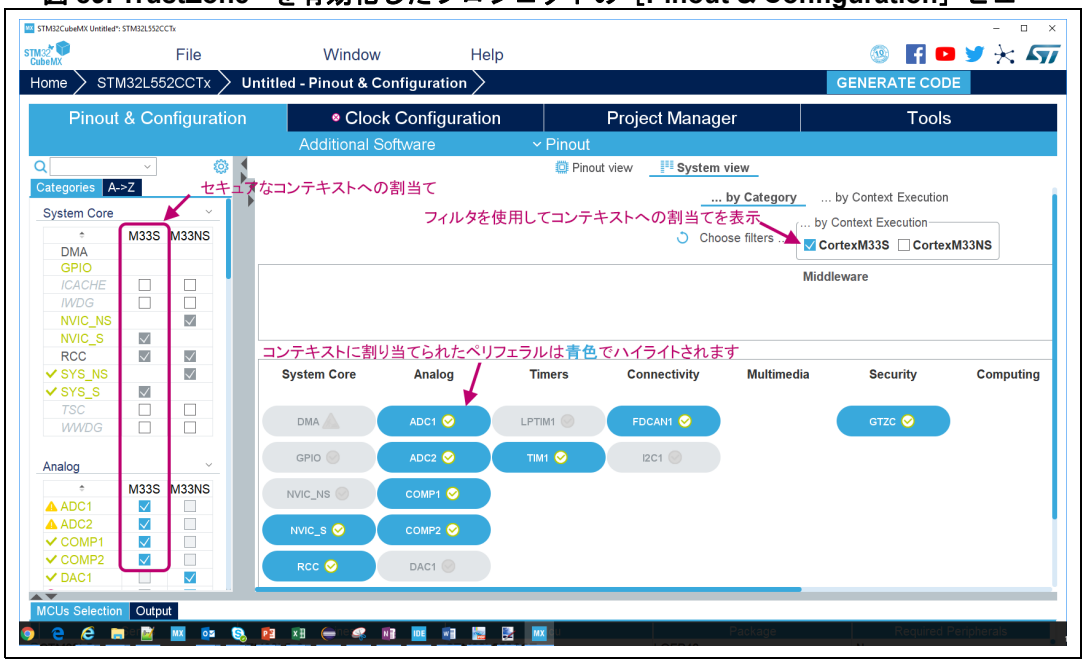
STM32L5 マイクロコントローラ･シリーズでは、Arm Cortex-M33 プロセッサと、同プロセッサの Armv8-M 向け Arm® TrustZone® のセキュリティ機能が、ST の実装によるセキュリティ機能に統合されています。

STM32L5 マイクロコントローラは、以下をサポートします。

- 次の 2 つのレベルの特権
  - 特権なし：ソフトウェアにはシステム･リソースに対する限定的なアクセス権のみがあります。
  - 特権あり：ソフトウェアはシステム･リソースに対するフル･アクセス権を持ち、セキュリティの制約の対象になります。
- 2 つのセキュリティ状態であるセキュアと非セキュア：TrustZone® セキュリティは Flash_OPTR レジスタの TZEN オプション･ビットをセットするとアクティブ化されます。セキュリティ状態はモードや特権からは独立しています。したがって、両方のモードと両方の特権レベルで各セキュリティ状態を実現できます。

STM32CubeMX では、TrustZone® をアクティブ化するかどうかを、プロジェクトの作成時に選択します（セクション 4.2：［New Project］ウィンドウ参照）。TrustZone® を有効にすると、それに応じて STM32CubeMX の［Pinout & Configuration］ビューが調整されます。
この調整では、セキュアなコンテキスト（M33S）と非セキュアなコンテキスト（M33NS）に分割され、セキュリティ関連の設定オプションが追加されます（図 89 参照）。

図 89. TrustZone® を有効化したプロジェクトの［Pinout & Configuration］ビュー




### 4.7.1 ペリフェラル、GPIO EXTI、DMA リクエストへの特権アクセス

STM32CubeMX では TrustZone® とは関係なく、次の特権アクセスが有効になります。

- 各ペリフェラルへのアクセス : 図 90 に示した GTZC 設定パネルで設定（セクション 4.7.5 : 参照）。
- 各 GPIO EXTI へのアクセス : 図 91 に示した GPIO 設定パネルで設定。
- 各 DMA チャネルへのアクセス : 図 92 に示した DMA 設定パネルで設定（セクション 4.7.4 : 参照）。

**注 :** **TrustZone® がアクティブな場合、すべての RCC レジスタを特権モードに設定すること、またはどの RCC レジスタも特権モードに設定しないことができます。STM32CubeMX では、RCC の［Mode］パネルにある、［Privileged-only attribute］チェックボックスによってこの設定を選択します（図 93 参照）。特権モードでは、すべての RCC レジスタの設定が、PWR_CR_PRIVEN ビットの設定によって特権アプリケーション用に予約されます。これは、TrustZone® をアクティブ化するとセキュリティで保護されます。**

図 90. ペリフェラルに対する特権の設定

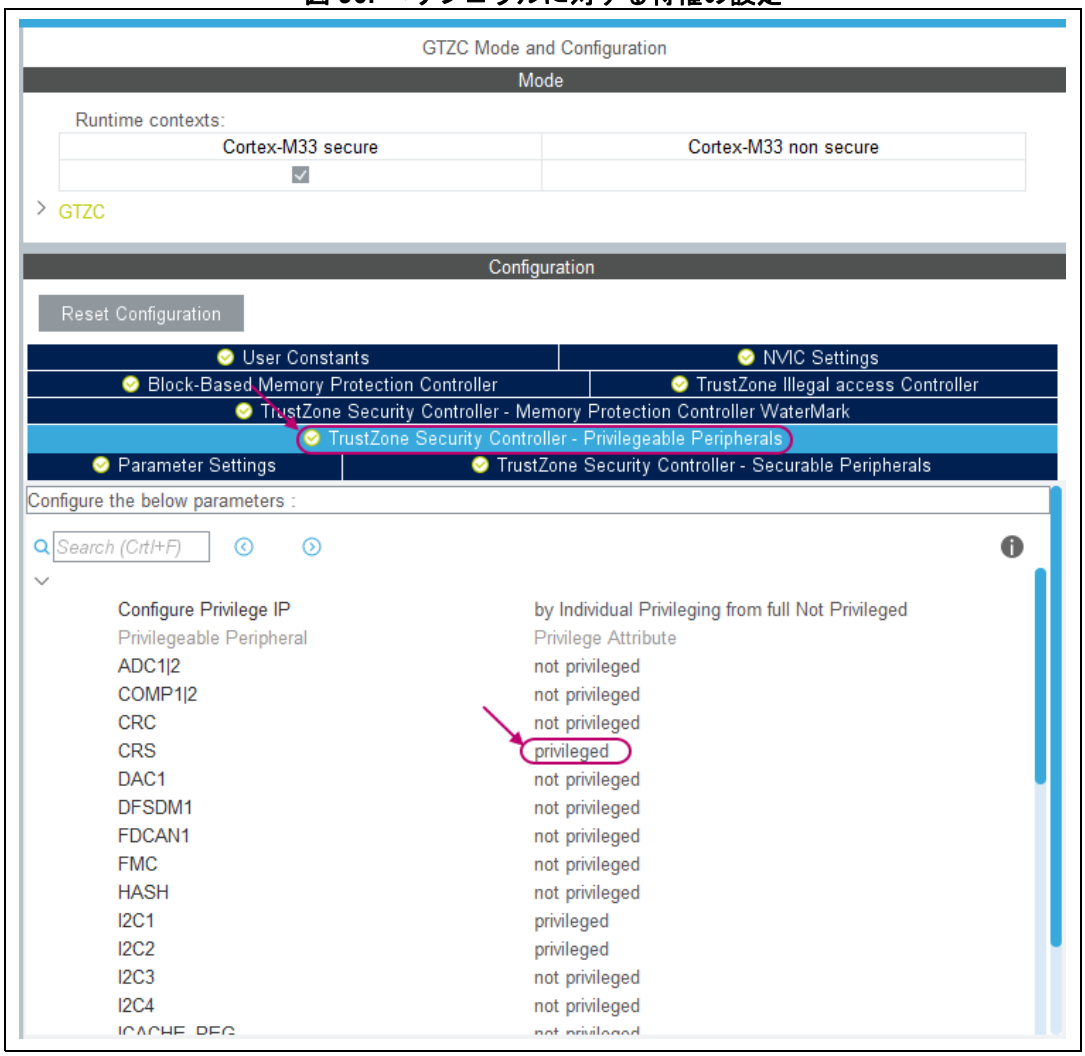


図 91. GPIO EXTI に対する特権の設定

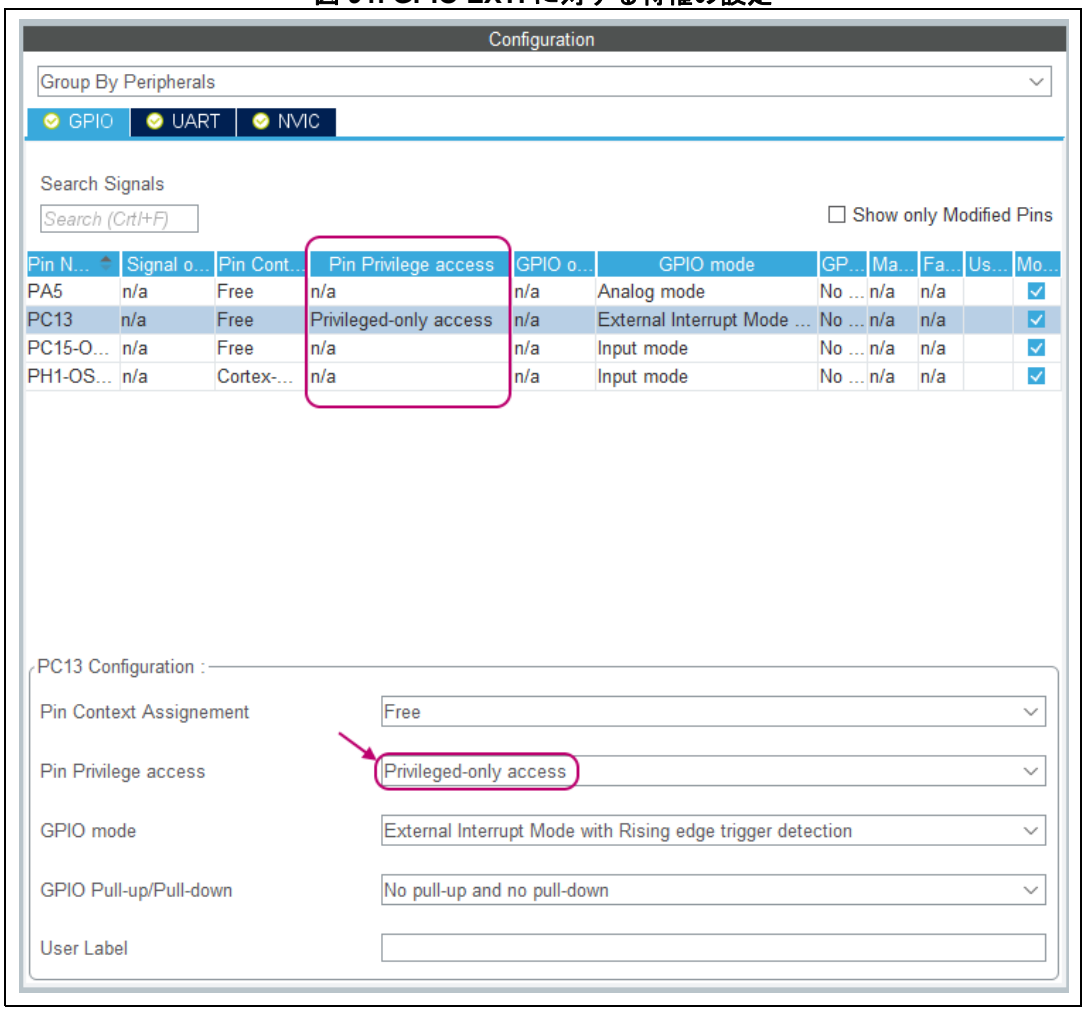


図 92. DMA リクエストのセキュリティと特権の設定

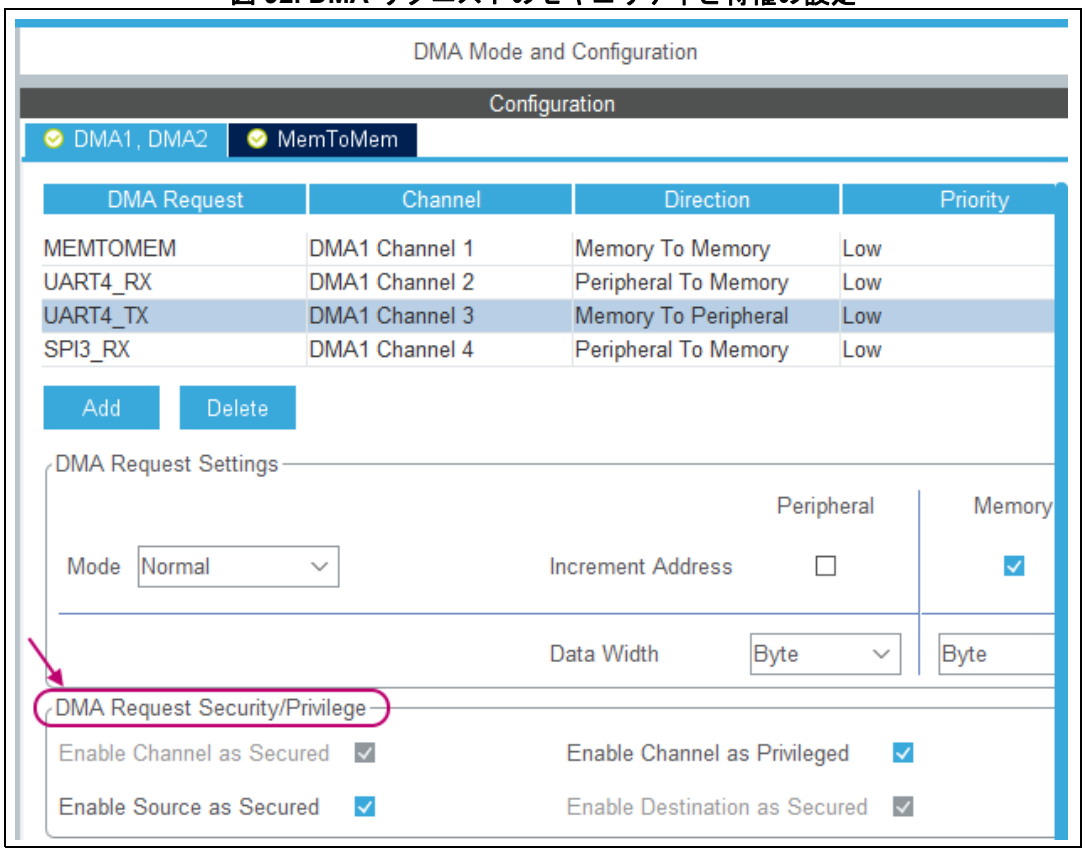
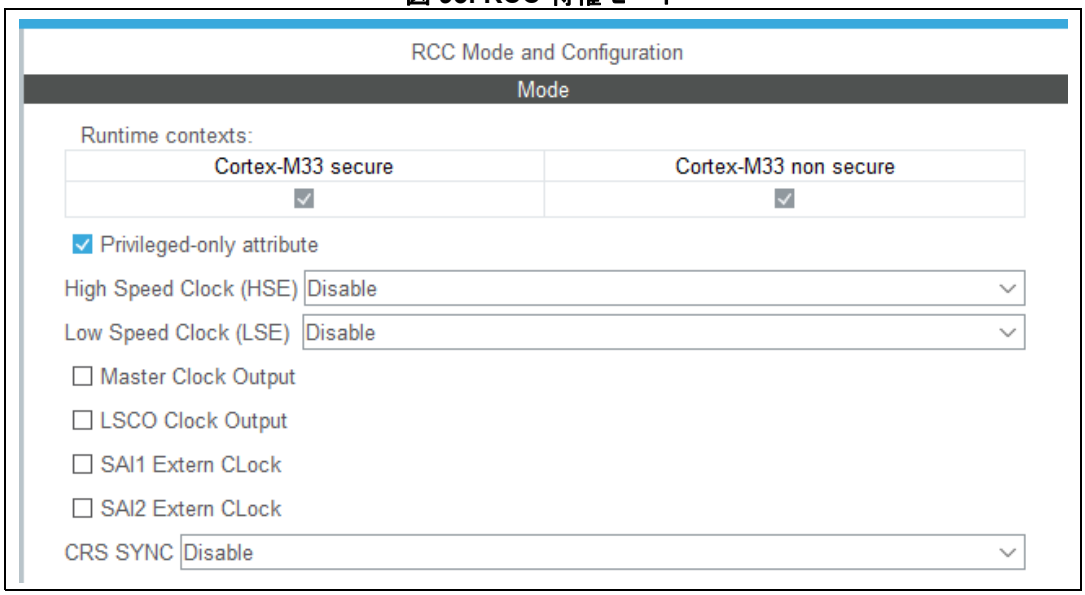



図 93. RCC 特権モード



### 4.7.2 GPIO、ペリフェラル、ミドルウェアに対するセキュアなコンテキストまたは非セキュアなコンテキストの割当て

STM32CubeMX では以下の設定が可能です。

- 各ペリフェラルとミドルウェアをコンテキストの 1 つに割り当てる
- GPIO の入力または出力をコンテキストの 1 つに割り当てるか、GPIO を必要とする可能性がある他のコンポーネントのために、GPIO をどこにも割り当てない状態にしておく。この場合は、GPIO を予約しているコンポーネントと同じコンテキストに GPIO が割り当てられます。デフォルトでは、すべての IO がセキュリティで保護されます。

割り当ては、次のように該当のパネルで設定します。

- ペリフェラルとミドルウェアのみの場合：［Show contexts］オプションが有効な場合は（有効にするには歯車アイコンをクリック）コンポーネント･ツリー･パネルで割り当て。または［Mode］パネルで割り当て。
- ペリフェラルのみの場合：GTZC 設定パネル（ペリフェラル専用）で割り当て。
- GPIO のみの場合：GPIO ピンを右クリックして［Pin Reservation］を選択し、［Configuration］パネルまたは［Pinout］ビューで割り当て。
- DMA リクエストの場合：DMA の［Configuration］パネルで割り当て。

**注：** **RCC リソースは、[Clock Configuration] ビューからセキュリティで保護できます（セクション 4.8.2：参照）。**

**注：** **ペリフェラルを必要とするミドルウェアは、そのペリフェラルが既に割り当てられているコンテキストにのみ割り当てることができます。**

### 4.7.3 ペリフェラル割込みのための NVIC とコンテキストの割当て

TrustZone® を有効にすると、割込みコントローラが非セキュアなコンテキスト用の NVIC_NS とセキュアなコンテキスト用の NVIC_S に分割されます。SysTick もコンテキストごとに 1 つずつ合計 2 つのインスタンスを使用できるようになり、それぞれ SYS_NS と SYS_S に表示されます。

デフォルトでは、すべての割込みがセキュリティで保護されます。

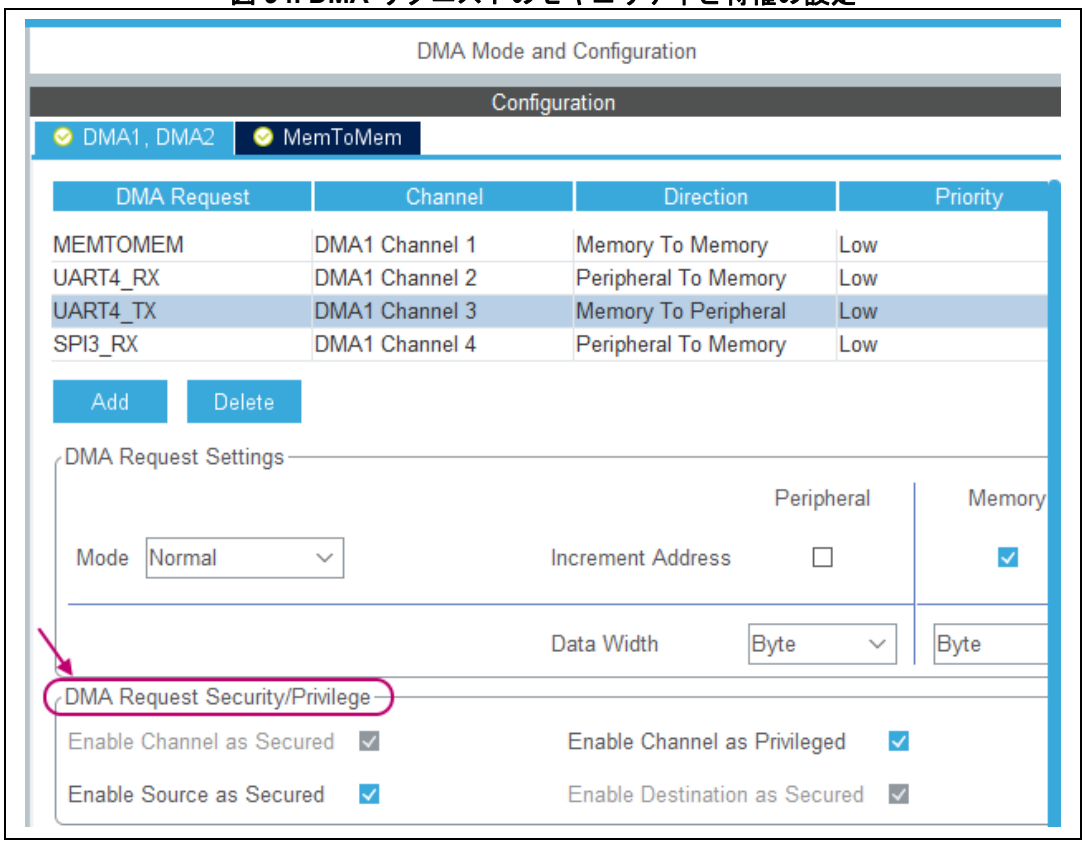
ペリフェラルの割込みは、コンテキストに対応する割込みコントローラに自動的に割り当てられます。

- 非セキュアなコンテキストに割り当てられたペリフェラルの割込みは NVIC_NS で有効になります。
- セキュアなコンテキストに割り当てられたペリフェラルの割込みは NVIC_S で有効になります。

### 4.7.4 DMA（コンテキスト割当てと特権アクセスの設定）

STM32CubeMX では、DMA チャネルを特権ありに設定できるほか、状況によっては、DMA のチャネル、転送元、転送先をセキュリティで保護できます（図 94 参照）。

図 94. DMA リクエストのセキュリティと特権の設定




DMA チャネルは、デフォルトでは特権なしに設定されますが、いつでも特権ありに設定できます。

DMA のチャネル、転送元、転送先をセキュリティで保護するかどうかの選択は、リクエストの特性で決まります。

次の 4 種類の状況があります。

- メモリ間転送リクエストまたは DMA ジェネレータ･リクエストの場合：チャネルはデフォルトではセキュリティで保護されませんが、保護することは可能です。転送元と転送先は、チャネルをセキュリティで保護した場合にのみ保護できます。
- 非セキュアなコンテキストに割り当てられたペリフェラルに対するリクエストの場合：チャネル、転送元、転送先を保護できません（該当のチェックボックスが無効になっています）。したがって、これらも強制的に非セキュアなコンテキストに割り当てられます。
- セキュアなコンテキストに割り当てられたペリフェラルに対する、ペリフェラルからメモリへのリクエストの場合：チャネルと転送元は自動的に保護されます（該当のチェックボックスが有効になり、無効化できません）。転送先についてはセキュリティで保護するかどうかを選択できます。
- セキュアなコンテキストに割り当てられたペリフェラルに対する、メモリからペリフェラルへのリクエストの場合：チャネルと転送先は自動的に保護されます（該当のチェックボックスが有効になり、無効化できません）。転送元についてはセキュリティで保護するかどうかを選択できます。

### 4.7.5 GTZC

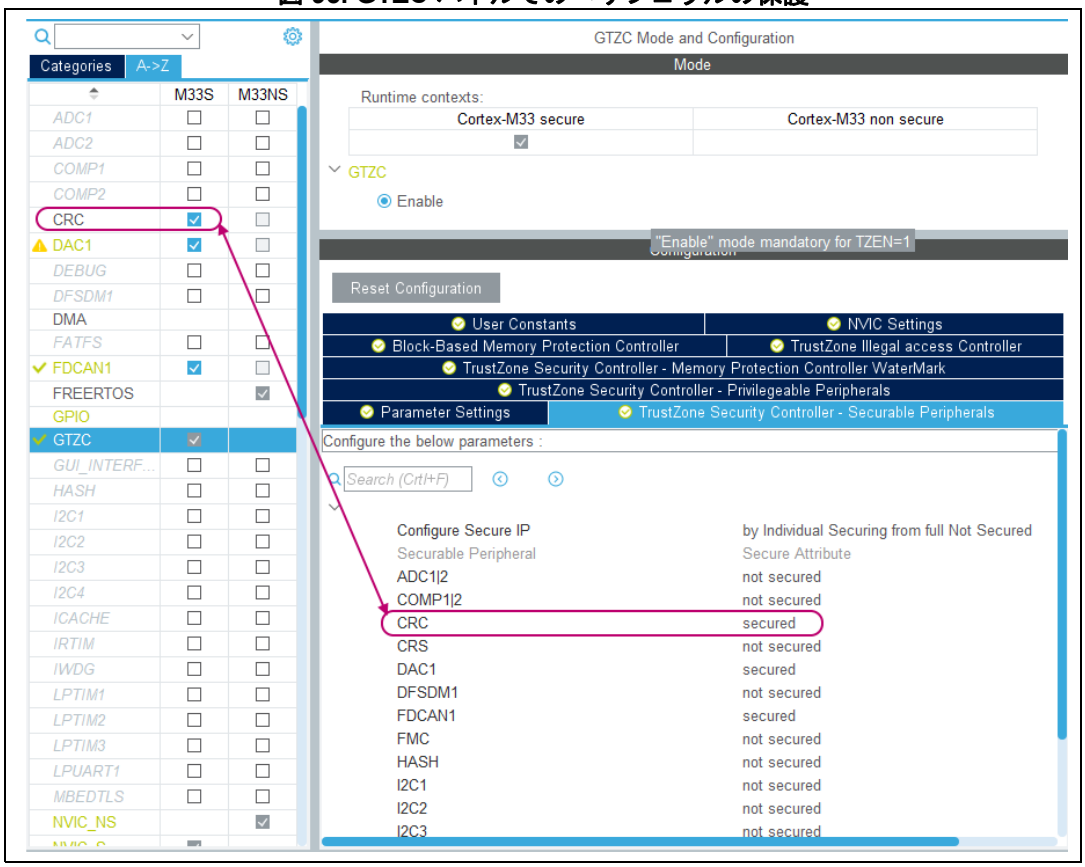
TrustZone® システム･セキュリティを設定するために、STM32L5 シリーズはグローバル TrustZone® セキュリティ･コントローラ（GTZC）を備えています。詳細は、リファレンス･マニュアル RM0438 を参照してください。

STM32CubeMX では、TrustZone® がアクティブなプロジェクトに対してデフォルトで GTZC が有効になり、無効化できません。TrustZone® がアクティブではないプロジェクトでは、GTZC を有効化できますが、機能は特権の設定に限られます。

GTZC は CubeMX で設定可能な次の 3 つのブロックから構成されます。この設定には、GTZC 設定パネルの専用タブを使用します。

- TZSC（TrustZone® セキュリティ･コントローラ）
  - セキュリティで保護するペリフェラルまたは特権を設定するペリフェラルを指定し、ウォーターマーク･メモリ･ペリフェラル･コントローラ（MPCWM）の非セキュアな領域のサイズを制御します。TZSC ブロックは、セキュリティで保護可能な各ペリフェラルのセキュア状態を RCC および I/O ロジックと共有することにより、そのセキュア状態を複数のペリフェラル（RCC、GPIO など）に通知します。
  - 特権は［TrustZone® Security Controller – Privilegeable Peripherals］タブで設定します。
  - セキュア状態は［TrustZone® Security Controller – Securable Peripherals］タブで設定します（セキュア状態は、ツリー･ビューまたは［Mode］パネルでのコンテキスト（M33S または M33NS）への割り当てに一致します）。
  - MPCWM は［TrustZone® Security Controller – Memory Protection Controller Watermark］タブで設定します。
- MPCBB（ブロックベースのメモリ保護コントローラ）
  - 関連する SRAM のすべてのブロック（256 バイトのページ）のセキュア状態を制御します。［Block-Based Memory Protection Controller］タブで設定します。
- TZIC（TrustZone® 不正アクセス･コントローラ）
  - システムで発生したすべての不正アクセス･イベントを収集し、NVIC へのセキュア割込みを生成します。［TrustZone® Illegal access Controller］タブで設定します。

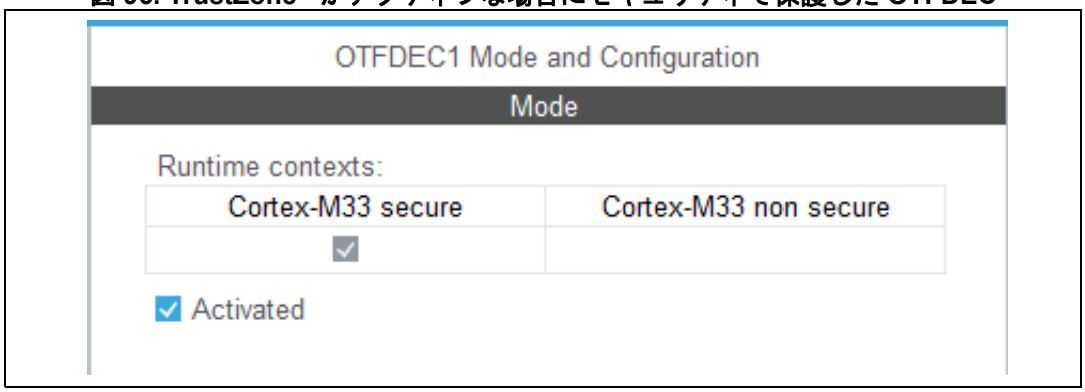
図 95. GTZC パネルでのペリフェラルの保護




## 4.7.6 OTFDEC

オンザフライ復号エンジン（On-The-Fly DECryption engine：OTFDEC）を使用すると、読出しリクエストのアドレス情報に基づいてオンザフライで AHB トラフィックを復号できます。製品でセキュリティが有効になっている場合、セキュリティで保護したホストによってのみ OTFDEC をプログラムできます。

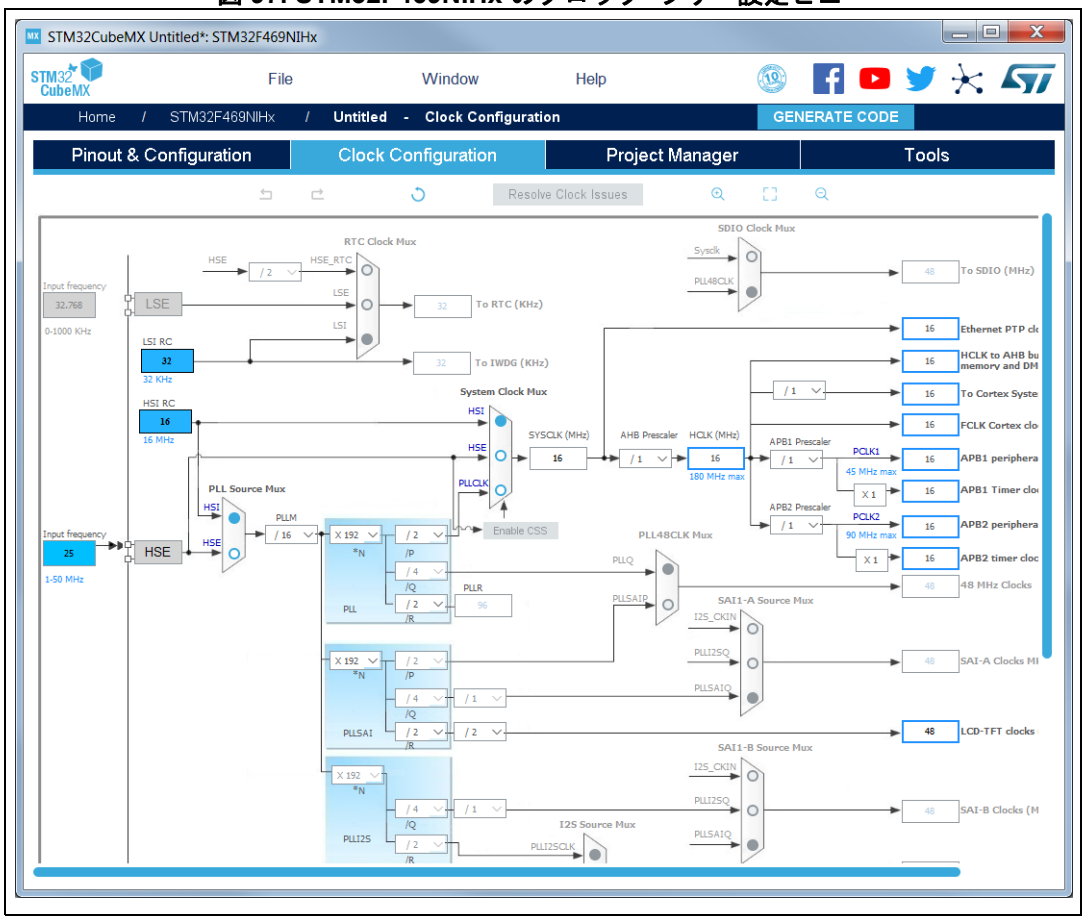
図 96. TrustZone® がアクティブな場合にセキュリティで保護した OTFDEC



## 4.8 ［Clock Configuration］ビュー

STM32CubeMX の［**Clock Configuration**］ウィンドウ（図 97 参照）には、クロックのパス、クロック･ソース、分周回路、および逓倍回路の概要が表示されます。ドロップダウン･メニューとボタンを使用し、アプリケーションの要件に合わせて実際のクロック･ツリー設定を変更できます。

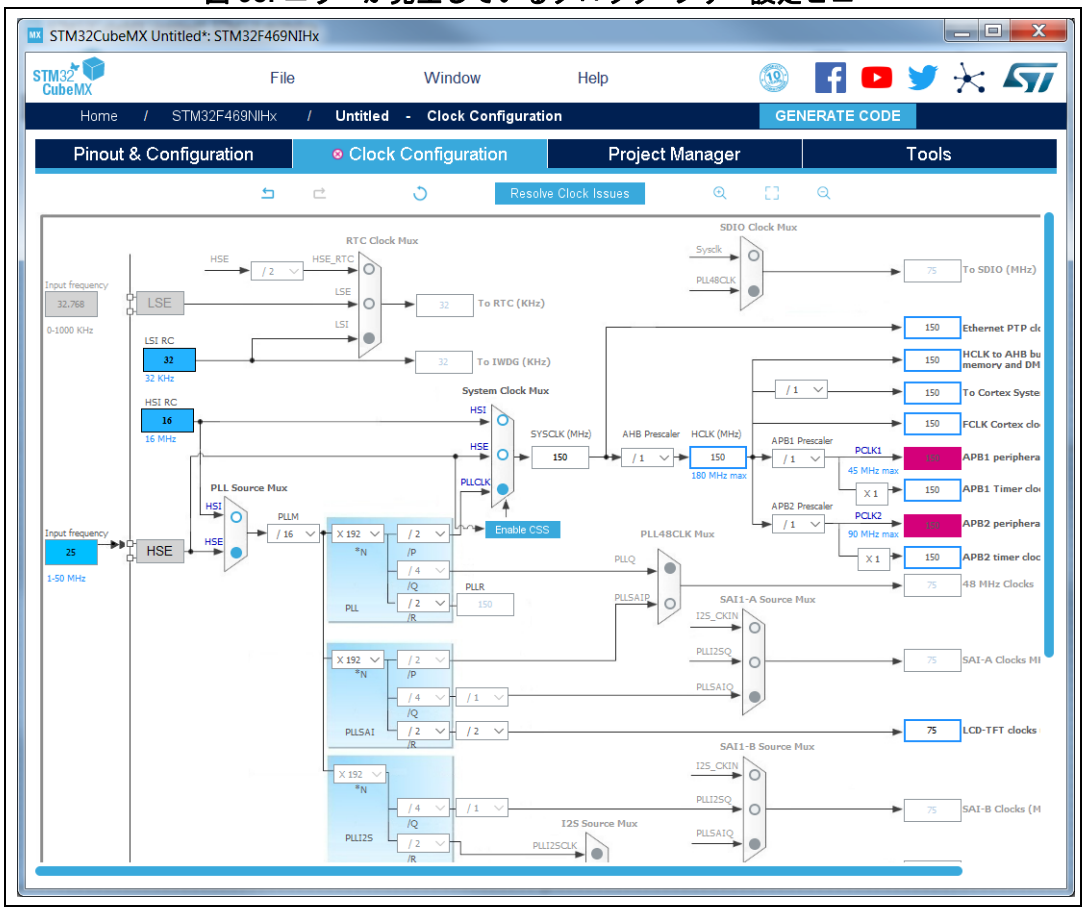
**図 97. STM32F469NIHx のクロック･ツリー設定ビュー**




実際のクロック速度が表示され、これらはアクティブになっています。使用中のクロック信号は、青色でハイライトされています。

範囲外の設定値は 図 98 に示したようにハイライトされ、問題が発生している可能性があることが示されます。そのような設定上の問題を自動的に解決する方法が提示されます。

**図 98. エラーが発生しているクロック･ツリー設定ビュー**




ここでは逆方向の設定機能が用意されています。目的のクロック速度を青色のフィールドに入力すると、その速度が得られるように STM32CubeMX によって分周回路と逓倍回路が再設定されます。クロック値のフィールドを右クリックすると、得られたクロック値をロックして変更不可にすることができます。

STM32CubeMX によって次の初期化コードが生成されます。

- main.c：関連する HAL_RCC 構造の初期化と関数呼出し
- stm32xxxx_hal_conf.h：オシレータ周波数と $V_{DD}$ の値

## 4.8.1 クロック･ツリーの設定機能

### 外部クロック･ソース

外部クロック･ソースを使用する場合、**[Pinout]** ビューで RCC ペリフェラルに表示される該当のクロック･ソースを有効にしておく必要があります。

### ペリフェラルのクロックを設定するオプション

クロック･ペリフェラルに対応する他のパスは灰色表示され、編集できません。これらをアクティブにするには、**[Pinout]**ビューでペリフェラル（USB など）を正しく設定する必要があります。このビューでは、次のような設定が可能です。

- **CPU（HCLK）、バス、またはペリフェラルのクロック周波数値の入力**

  STM32CubeMX により、ペリフェラルに対する他の制約（USB クロック最小値など）を考慮しながら、プリスケーラと分周回路を調整して要求した周波数を実現するクロック･ツリー設定が提示されます。解決策が見つからない場合、STM32CubeMX から別のクロック･ソースへの切替えが提案されることもあれば､要求した周波数を実現できる解決策が存在しないという結論になることもあります。

- **現在値を保持する必要がある周波数フィールドのロック**

  周波数フィールドを右クリックして**[Lock]**を選択すると、STM32CubeMX でクロック設定の新しい解決策を検索するときに、そのフィールドに現在割り当てている周波数値が保持されます。

  値を保持する必要がなくなったら、ロックした周波数フィールドのロックを解除できます。

- **システム･クロック（SYSCLK）を駆動するクロック･ソースの選択**
  - ユーザ定義の周波数を提供する外部オシレータ･クロック（HSE）
  - 定義された固定周波数を提供する内部オシレータ･クロック（HSI）
  - メイン PLL クロック
- **二次ソースの選択（該当製品で使用可能なソース）**
  - ロースピードの内部（LSI）/外部（LSE）クロック
  - I2S 入力クロック
  - その他のソース
- **プリスケーラ、分周回路、および逓倍回路の値の選択**
- **マイクロコントローラでサポートしているクロック･セキュリティ･システム（CSS）を HSE に対して有効化**

  この機能は、HSE クロックを直接または PLL 経由で間接的にシステム･クロック･ソースとして使用している場合にのみ利用できます。この機能を使用すると、HSE の障害を検出してソフトウェアに通知でき、それを受けてマイクロコントローラで対策をとることができます。

- **マイクロコントローラでサポートしている CSS を LSE に対して有効化**

  この機能は、LSE と LSI が有効で、RTC または LCD のクロック･ソースに LSE または LSI のどちらかを選択している場合にのみ利用できます。

- **ツールバーの［Reset］ボタンによるクロック･ツリーのデフォルト設定へのリセット**

  この機能では、STM32CubeMX のデフォルトのクロック･ツリー設定が読み込まれます。

- **ツールバーの［Undo/Redo］ボタンによるユーザ設定手順の取り消しとやり直し**
- **設定上の問題の検出と解決**

  コードを生成する前に、誤ったクロック･ツリー設定を検出します。エラーは赤紫色でハイライトされ、**[Clock Configuration]**ビューに赤紫色の×が表示されます（図 98 参照）。

  問題を手動で解決するほか、問題が検出された場合にのみ有効になる**[Resolve Clock Issue]**ボタンをクリックして自動的に解決することもできます。

  基本的な解決プロセスは次の手順で実行されます。

  a) HSE 周波数をその最大値に設定します（オプション）。

  b) HCLK 周波数、ペリフェラル周波数の順に最大値または最小値に設定します（オプション）。

  c) 逓倍回路入力を変更します（オプション）。

d) 最後に、逓倍回路と分周回路の値を繰り返し調整して問題を修正します。解決策が見つかると、クロック･ツリーで赤紫色にハイライトされていた箇所が通常の表示になります。見つからない場合は、エラー･メッセージが表示されます。

注： **外部クロック、I2S 入力クロック、およびマスタ･クロックがクロック･ツリーに表示されるようにするには、[Pinout]ビューの RCC 設定でこれらのクロックを有効にしておく必要があります。この情報は、ツールチップとしても表示されます。**

このツールでは、自動的に次の操作が実行されます。

- 選択したクロック･ソース、クロック周波数、およびプリスケーラ、逓倍回路、分周回路の値に従って、バス周波数、タイマ、ペリフェラル、およびマスタ出力クロックが調整されます。
- 指定した設定の有効性が確認されます。
- 無効な設定が赤紫色でハイライトされ､有効な設定にするための手順がツールチップで表示されます。

RCC 設定（RCC の［**Pinout & Configuration**］ビューで設定）に従って［**Clock Configuration**］ビューが調整されます。または、［Clock Configuration］ビューに従って RCC 設定が調整されます。

- RCC の［**Pinout**］ビューで外部クロックと出力クロックを有効にすると、それらは［**Clock Configuration**］ビューで設定可能になります。
- RCC の設定ビューでタイマのプリスケーラを有効にすると､タイマ･クロックの逓倍回路が適切に選択されます。

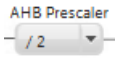
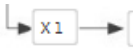
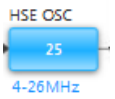
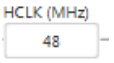
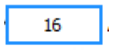
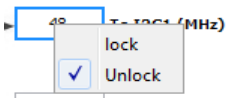
逆に、クロック･ツリーの設定によって、［Configuration］ビューで次の RCC パラメータが変化することがあります。

- Flash遅延：$V_{DD}$ 電圧、HCLK 周波数、および電源オーバードライブ状態から自動的に得られる待機状態数。
- 電力レギュレータの電圧スケール：HCLK 周波数から自動的に得られます。
- 電源オーバードライブは、HCLK 周波数に応じて自動的に有効になります。電源ドライブが有効な場合、AHB ドメインと APB ドメインに対する最大許容周波数が高くなります。これらは、［**Clock Configuration**］ビューに表示されます。

起動時に使用されるデフォルトの最適なシステム設定値は､system_stm32f4xx.c ファイルで定義されています。このファイルは、STM32CubeMX によって、STM32CubeF4 マイクロコントローラ･パッケージからコピーされます。ユーザ定義クロック設定への切替えは、後で main 関数で実行されます。

図 97 に、STM32F429x マイクロコントローラのクロック･ツリー設定の例を示します。表 9 では、各クロックの設定で使用できるウィジェットについて説明しています。

**表 9. ［Clock Configuration］ビューのウィジェット**

| フォーマット | ペリフェラル･インスタンスの設定ステータス |
|---|---|
|  | アクティブなクロック･ソース |
|  | 使用不可の設定は不鮮明な輪郭または灰色で表示（クロック･ソース、分周回路など） |
|  | プリスケーラ、分周回路、逓倍回路の選択に使用する灰色のドロップダウン･リスト |
|  | 逓倍回路の選択 |
|  | ユーザ定義周波数値 |
|  | 自動的に得られる周波数値 |
|  | ユーザが変更可能な周波数フィールド |
|  | 輪郭が青い長方形を右クリックして、周波数フィールドのロックまたは非ロックを指定。ロックすると、クロック･ツリー設定が更新されても、この周波数値が保持されます。 |

### 4.8.2 クロック･リソースのセキュリティ保護（STM32L5 シリーズのみ）

TrustZone® セキュリティがアクティブな場合、RCC でセキュリティ設定レジスタを使用して、システム･クロック･リソースへの非セキュアなアクセスを阻止できます。

この機能に伴い、STM32CubeMX では、以下のクロック･リソースをセキュリティで保護できます。

- 固定周波数のシステム･クロック･ソース：HSI、LSI、RC48
- 周波数設定可能なシステム･クロック･ソース：HSE（+CSS）、MSI、LSE（+CSS）
- 2 つのマルチプレクサ：CLK48 クロックのマルチプレクサ、システム･クロック（+MCO ソース）のマルチプレクサ
- その他のシステム設定：位相ロック･ループである PLLSYS、PLLSAI1、PLLSAI2 およびバス･プリスケーラである AHB/APB1/APB2

［Clock Configuration］ビューでは、このような保護可能なリソースには鍵のアイコンが表示されます。セキュリティ保護を有効にするには、リソースを右クリックして［Secure］チェックボックスを選択します。セキュリティ保護したリソースは緑色の四角でハイライトされます。

設定可能なリソースは、右クリックして表示される［Lock］チェックボックスを選択することで、それ以上設定が変更されないようにロックできます。

セキュリティ保護と設定の両方が可能なすべてのリソースには、1 回のクリックでロックまたはアンロックできるショートカット･ボタンも用意されています。

ペリフェラルをセキュリティ保護すると、関連するクロック、リセット、クロック･ソース、クロックの有効化も保護されます。STM32CubeMX では、ペリフェラルのセキュリティ保護を［Pinout & Configuration］ビューで設定します。これにより、そのクロック･ソースが自動的にセキュリティで保護され、クロック設定ビューでは緑色の四角でハイライト表示されます。

**表 10. ［Clock Configuration］ビューのセキュリティ設定**

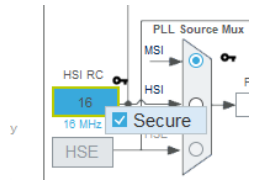
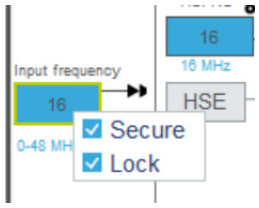
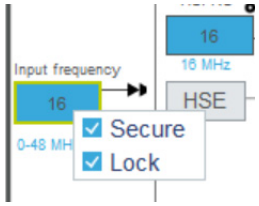
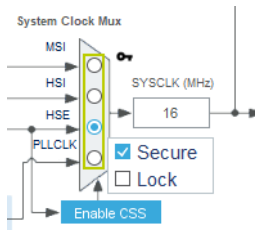
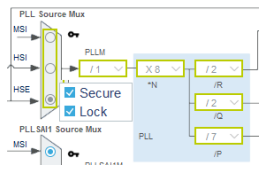
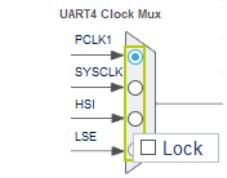
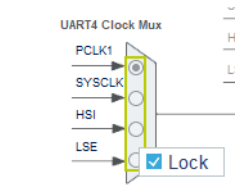
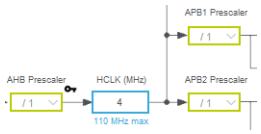
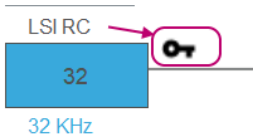
| ビュー | 説明 |
|---|---|
|  | セキュリティで保護され、設定不可のシステム･クロック･リソースの例 |
|  | セキュリティで保護され､編集可能な状態を維持しているシステム･クロック HSE のクロック･ソースの例。周波数の値を変更できます。 |
|  | セキュリティで保護され、編集がロックされたシステム･クロック HSE のクロック･ソースの例。周波数の値を変更できません。 |
|  | セキュリティで保護され、ロックされていないシステム･クロックのマルチプレクサの例。クロック･ソースを変更できます。 |
|  | セキュリティで保護され、ロックされたメイン PLL マルチプレクサの例。クロック･ソースは HSE に固定され、変更できません。PLLxxM、PLLxxN、PLLxxP、PLLxxQ、PLLxxR はセキュリティで保護され、編集もロックされています。 |

表 10. ［Clock Configuration］ビューのセキュリティ設定 （続き）

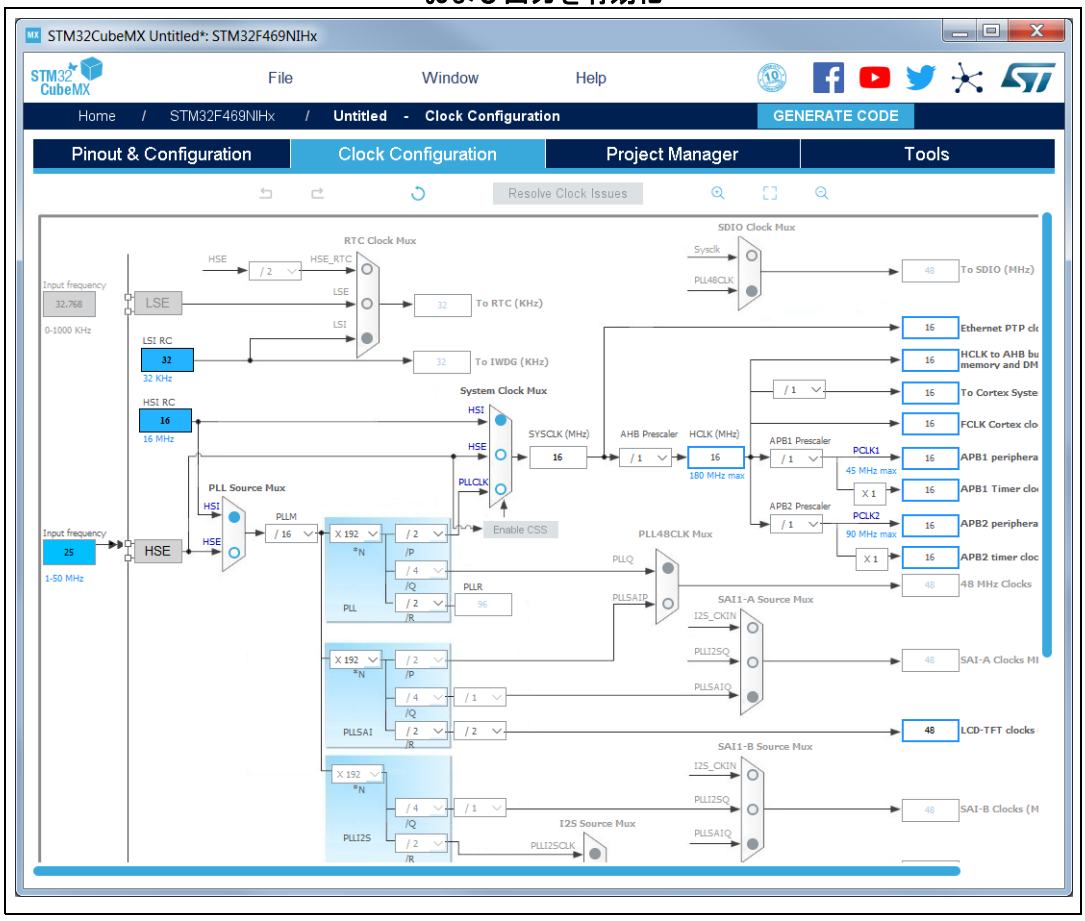
| ビュー | 説明 |
|---|---|
|  | UART4 クロック･ソース･マルチプレクサの例。［Pinout & Configuration］ビューで UART4 ペリフェラルのセキュリティ保護が設定されているので、クロック･ソースがセキュリティで保護されます。クロック･ソースは PCLK1 に設定されていますが、[Lock] チェックボックスがチェックされていないため変更が可能です。 |
|  | UART4 クロック･ソース･マルチプレクサの例。［Pinout & Configuration］ビューで UART4 ペリフェラルのセキュリティ保護が設定されているので、クロック･ソースがセキュリティで保護されます。クロック･ソースは PCLK1 に設定され、［Lock］がオンであるため変更できません。 |
|  | AHB プリスケーラへのアクセスのセキュリティ保護とロックの例。プリスケーラ APB1 と APB2 もロックされています。 |
|  | セキュリティ保護可能なリソースであることが鍵アイコンで示されている LSI の例。 |
|  | すべてをロックまたはアンロックするボタン（セキュリティ保護可能なリソースでのみ有効） |

### 4.8.3 推奨事項

**[Clock Configuration]** ビューで設定できるのはクロックだけではありません。RCC ペリフェラルや RTC ペリフェラルも設定できます。

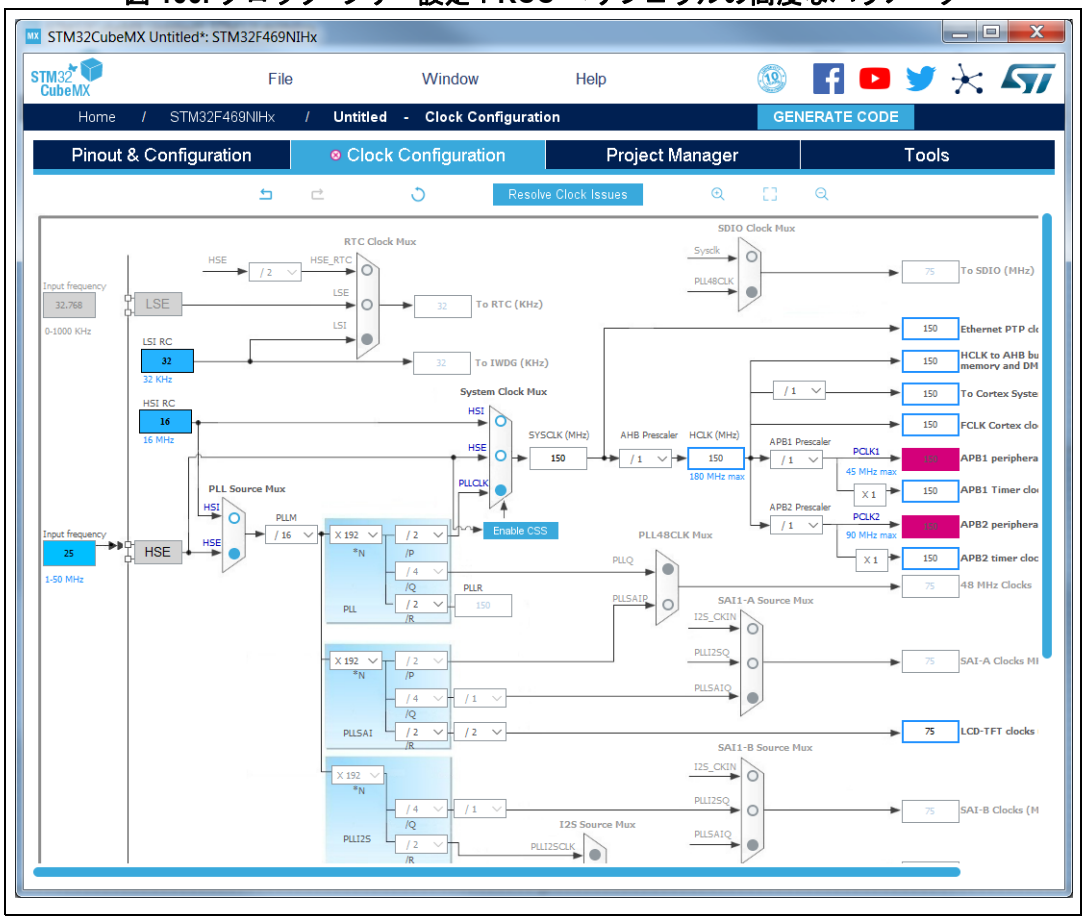
1. **[Pinout & Configuration]** ビューから RCC の［Mode］パネルに移動し、必要に応じて各種クロックを有効にします。外部クロック、マスタ出力クロック、オーディオ I2S 入力クロックがあれば、それらも有効にすることができます。つづいて、RCC 設定パネルに移動し、必要に応じてデフォルト設定を調整します。変更は **[Clock Configuration]** ビューに反映されます。定義した設定により、RCC 設定も変更される場合があります（図 99 参照）。

**図 99. クロック･ツリー設定：[Pinout] ビューで RTC、RCC クロック･ソースおよび出力を有効化**



2. 次に［**Pinout & Configuration**］ビューの［**RCC Configuration**］に移動します。ここで定義した高度な設定の設定値は、［**Clock Configuration**］ビューに反映されます。定義した設定により、RCC 設定も変更される場合があります。

**図 100. クロック･ツリー設定：RCC ペリフェラルの高度なパラメータ**




## 4.8.4 STM32F43x/42x の電源オーバードライブ機能

STM32F42x/43x マイクロコントローラは、十分な $V_{DD}$ 電源電圧が供給されていれば（例：$V_{DD}$ > 2.1 V）、AHB/APB バスの最大周波数（例：HCLK で 180 MHz）で動作できる電源オーバードライブ機能を実装しています。

表 11 に、電源オーバードライブ機能に関連するさまざまなパラメータと、STM32CubeMX でそれらにアクセスできるユーザ･インタフェースを示します。

表 11. 電圧スケールに対する電源オーバードライブと HCLK 周波数

| パラメータ | STM32CubeMX のパネル | 値 |
|---|---|---|
| $V_{DD}$ voltage | Configuration （RCC） | 事前定義されている範囲内でユーザが定義します。電源オーバードライブに影響を及ぼします。 |
| Power Regulator Voltage Scale | | HCLK 周波数と電源オーバードライブから自動的に得られます（表 12 参照）。 |
| Power Over Drive | | この値は、HCLK と $V_{DD}$ の値による制約を受けます（表 12 を参照）。この値が有効になるのは、$V_{DD}$ ≥ 2.2 V の場合のみです。<br>$V_{DD}$ ≥ 2.2 V の場合、HCLK から自動的に得られますが、複数の選択肢がある場合（例：HCLK = 130 MHz）はユーザが設定可能です。 |
| HCLK/AHB clock maximum frequency value | Clock Configuration | 最大許容値であることを示すために青色で表示されます。たとえば、電源オーバードライブを有効化できない場合（$V_{DD}$ ≤ 2.1 V の場合）、HCLK の最大値は 168 MHz、有効化できる場合は 180 MHz です。 |
| APB1/APB2 clock maximum frequency value | | 最大許容値であることを示すために青色で表示されます |

表 12 に、電源オーバードライブ･モードと HCLK 周波数の関係を示します。

表 12. 電源オーバードライブと HCLK 周波数の関係

| HCLK 周波数範囲：<br>電源オーバードライブ（POD）を有効にするには<br>$V_{DD}$ > 2.1 V が必要 | 対応する電圧スケールと<br>電源オーバードライブ（POD） |
|---|---|
| ≤ 120 MHz | スケール 3<br>POD は無効です。 |
| 120 ~ 144 MHz | スケール 2<br>POD は無効または有効のどちらも可能 |
| 144 ~ 168 MHz | POD が無効な場合はスケール 1<br>POD が有効な場合はスケール 2 |
| 168 ~ 180 MHz | POD を有効にする必要がある<br>スケール 1 （それ以外に対応していない周波数範囲） |

### 4.8.5 クロックツリーの用語

表 13. 用語

| 略号 | 定義 |
|---|---|
| HSI | 高速内部オシレータ：リセット後に有効、HSE より低精度 |
| HSE | 高速外部オシレータ：外部クロック回路が必要 |
| PLL | 位相ロック･ループ：上記のクロック･ソースの逓倍に使用 |
| LSI | 低速内部クロック：省電力クロック、通常はウォッチドッグ･タイマに使用 |
| LSE | 低速外部クロック：外部クロックから供給 |

表 13. 用語 （続き）

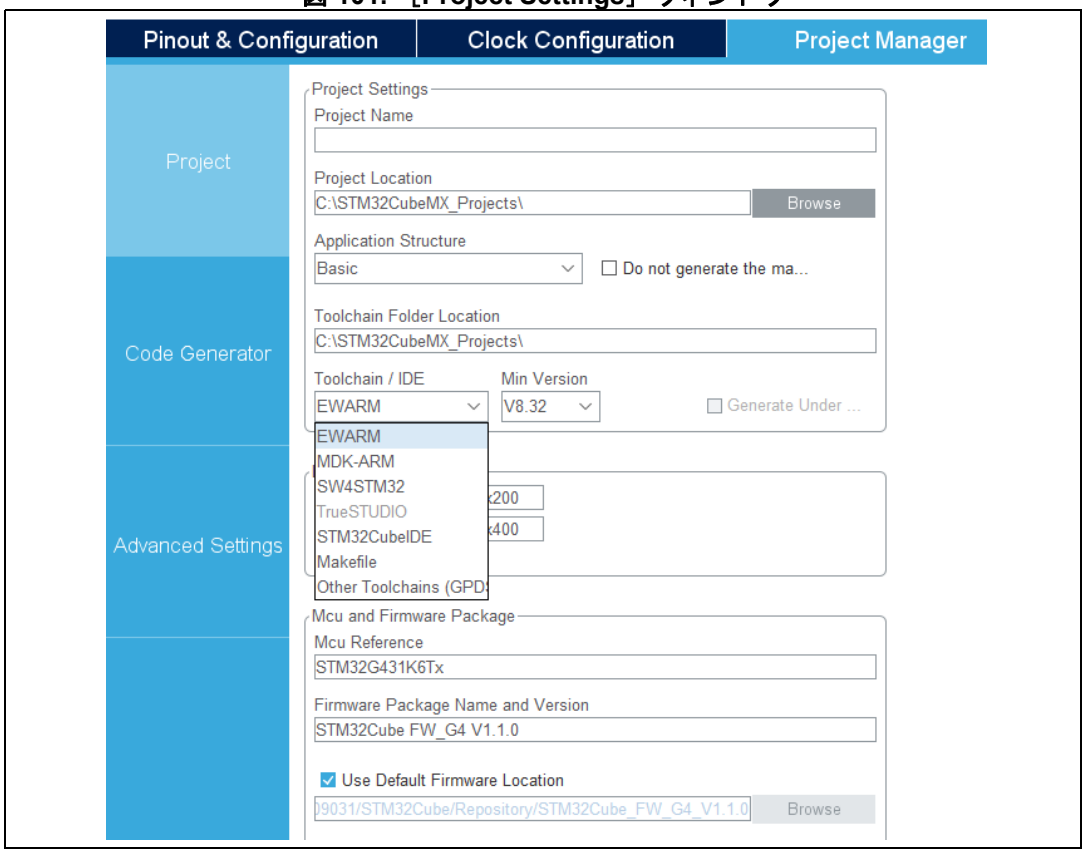
| 略号 | 定義 |
|---|---|
| SYSCLK | システム･クロック |
| HCLK | 内部 AHB クロック周波数 |
| FCLK | Cortex フリーランニング･クロック |
| AHB | アドバンスト･ハイパフォーマンス･バス |
| APB1 | 低速アドバンスト･ペリフェラル･バス |
| APB2 | 高速アドバンスト･ペリフェラル･バス |

## 4.9 ［Project Manager］ビュー

このビューには次の 3 つのタブがあります（図 101 参照）。

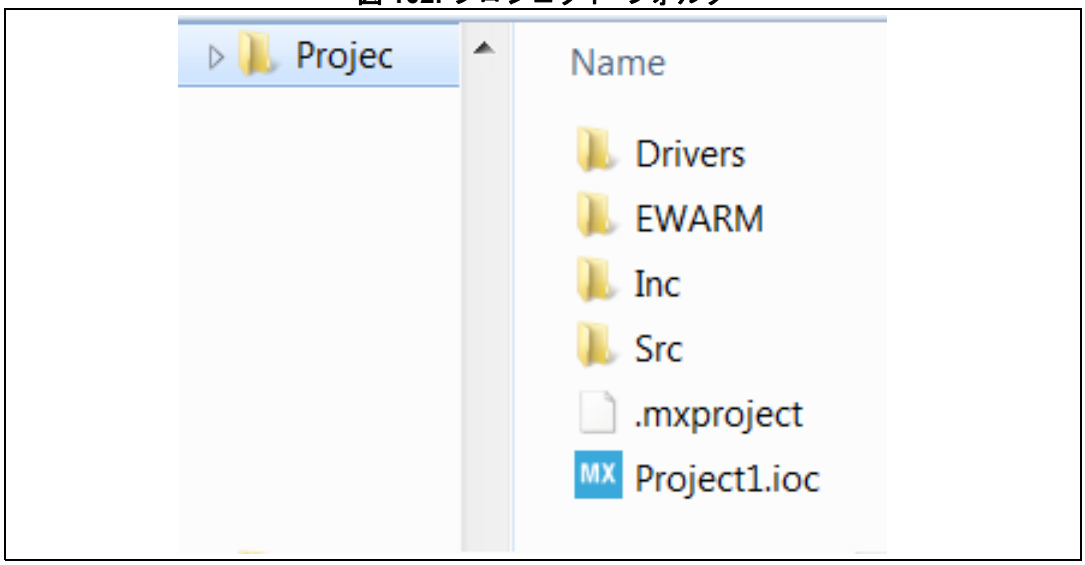
- 全般的なプロジェクト設定：プロジェクト名、保存場所、ツールチェーン、ファームウェアのバージョンを指定できます。
- コード生成:ペリフェラルの初期化コードの保存場所やライブラリのコピーとリンクのオプションなど、コード生成に関するオプションの設定や、コードをカスタマイズするためのテンプレートの選択などが可能です。
- 詳細設定：STM32CubeMX の初期化関数の呼び出し順序を設定する専用のタブです。

図 101. ［Project Settings］ウィンドウ




コードは、図 102 に示すプロジェクト･フォルダ･ツリー内に生成されます。

図 102. プロジェクト･フォルダ




注： **プロジェクト設定オプションによっては、プロジェクトを保存すると読出し専用になるものがあります。これらのオプションを変更するには、[File] → [Save Project As] メニューを使用して、プロジェクトを新規プロジェクトとして保存する必要があります。**

### 4.9.1 ［Project］タブ

**[Project Settings]** ウィンドウの **[Project]** タブでは、以下のオプションを設定できます（図 101 を参照）。

- ［Project Settings］:
  - ［Project Name］: 指定したプロジェクト保存場所に作成するプロジェクト･フォルダおよび .ioc ファイルに使用する名前
  - ［Project Location］: プロジェクト･フォルダを保存するディレクトリ
  - ［Application Structure］: ［Basic］または［Advanced］のいずれかのオプションを選択します。

    ［Basic］: 使用するミドルウェアが 1 つのみのプロジェクトまたはミドルウェアを使用しないプロジェクトに適した構造です。この構造は、IDE 設定フォルダをソースおよびインクルード･サブフォルダと同じ階層に組織化して配置します（図 103 を参照）。

    ［Advanced］: 複数のミドルウェア･コンポーネントを使用するプロジェクトに適した構造です。ミドルウェア･アプリケーションの統合が容易です（図 104 を参照）
  - ［Toolchain Folder Location］: デフォルトでは、.ioc ファイルと同じ階層にあるプロジェクト･フォルダに配置されます。
  - ［Toolchain/IDE］: 選択したツールチェーン[a]

a. STM32CubeMX の次期バージョンから TrueSTUDIO、SW4STM32、GPDSC がサポート対象外になります。進化した STM32CubeMX の利点をプロジェクトに反映するために、STM32CubeIDE をはじめとする IDE の導入をお勧めます。

– ［OpenSTLinux Settings］（STM32MP1 シリーズのみ）: 生成されるデバイス･ツリーの保存場所と現在のプロジェクトのマニフェストのバージョンと内容（図 105 参照）。これらの情報により、ソフトウェア･コンポーネントの適切なバージョンが、Cortex® M の場合は STM32CubeMP1、Cortex® A の場合は Linux、tf-a、u-boot で決まります。OpenAMP/RPM リンクやリソース管理 API に関連する Cortex® A 向け各種ソフトウェア･コンポーネントに 1 つのバージョンの Cube ファームウェアで対応しようとする場合に、この点への配慮が特に重要です。

［Toolchain/IDE］で［Makefile］を選択すると、GCC ベースの汎用 makefile が生成されます。

［Other Toolchains（GPDSC）］を選択すると gpdsc ファイルが生成されます。gpdsc ファイルには、プロジェクトのビルドに必要なドライバと起動ファイルをはじめとする各種ファイルのリストとパスなど、プロジェクトに関する一般的な内容が記述されます。この機能により、gpdsc をサポートするあらゆるツールチェーンに STM32CubeMX のプロジェクト生成を拡張できます。それらのツールチェーンは、gpdsc ファイルの情報を処理することにより、STM32CubeMX で生成された C プロジェクトを読み込むことができるからです。組込みプロジェクトの記述を標準化するために、gpdsc ソリューションは CMSIS-Pack に基づいています。

- SW4STM32 および Atollic® TrueSTUDIO® ツールチェーン向けのその他のプロジェクト設定：

  オプションの［**Generate under root**］チェックボックスをチェックすると、ツールチェーンのプロジェクト･ファイルが STM32CubeMX のユーザ･プロジェクトのルート･フォルダに生成されます。このチェックボックスをチェックしていない場合は、専用のツールチェーン･フォルダに生成されます。

  ルート･フォルダに STM32CubeMX プロジェクトを生成すると、SW4STM32 や TrueStudio®などの Eclipse ベースの IDE を使用した場合に、以下の Eclipse 機能の利点が得られます。

  – プロジェクトをインポートするとき、必要に応じて Eclipse のワークスペースにプロジェクトをコピーできます。
  – Eclipse のワークスペースで GIT や SVN などのソース管理システムを使用できます。

  プロジェクトをワークスペースにコピーするオプションを使用すると、2 つの異なるプロジェクトが存在することになるので、Eclipse での変更と STM32CubeMX での変更が同期しなくなります。

- ［Linker Settings］: アプリケーションに割り当てるヒープとスタックの最小サイズを指定します。デフォルトでヒープ･サイズとして 0x200、スタック･サイズとして 0x400 が提示されます。アプリケーションでミドルウェア･スタックを使用する場合、これらにデフォルトよりも大きい値を指定することが必要になる場合もあります。
- 複数バージョンのファームウェア･パッケージが用意されている場合のバージョンの選択：連続する複数のバージョンで同じ API を実装し、同じマイクロコントローラをサポートしている場合に、バージョンの選択が必要になります。デフォルトでは、使用可能な最新のバージョンが選択されます。
- ファームウェアの保存場所の選択オプション

  デフォルトの保存場所は、［**Help**］→［**Updater Settings**］メニューで指定した場所です。

  ［**Use Default Firmware Location**］チェックボックスのチェックを外すと、プロジェクトに使用するファームウェアにデフォルトとは異なるパスを指定できます（図 106 参照）。

図 103. アプリケーション構造に［Basic］を選択した場合

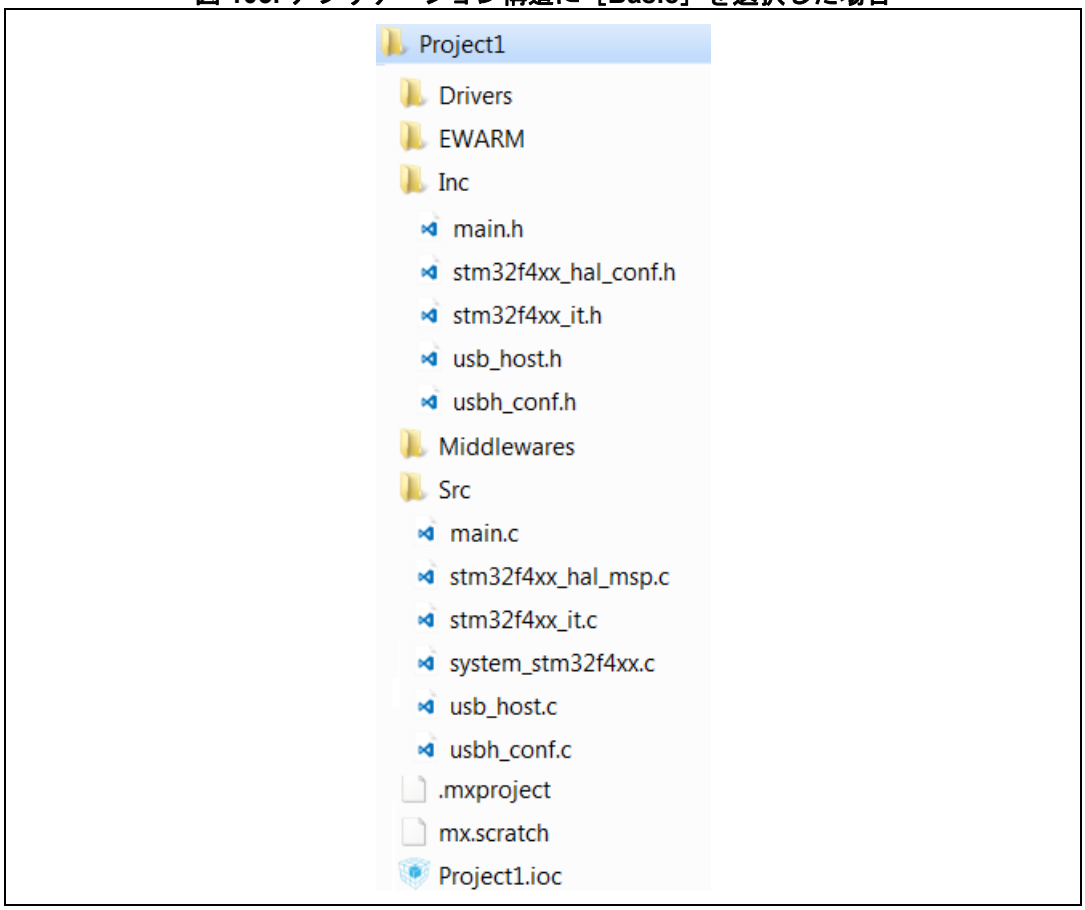


図 104. アプリケーション構造に［Advanced］を選択した場合

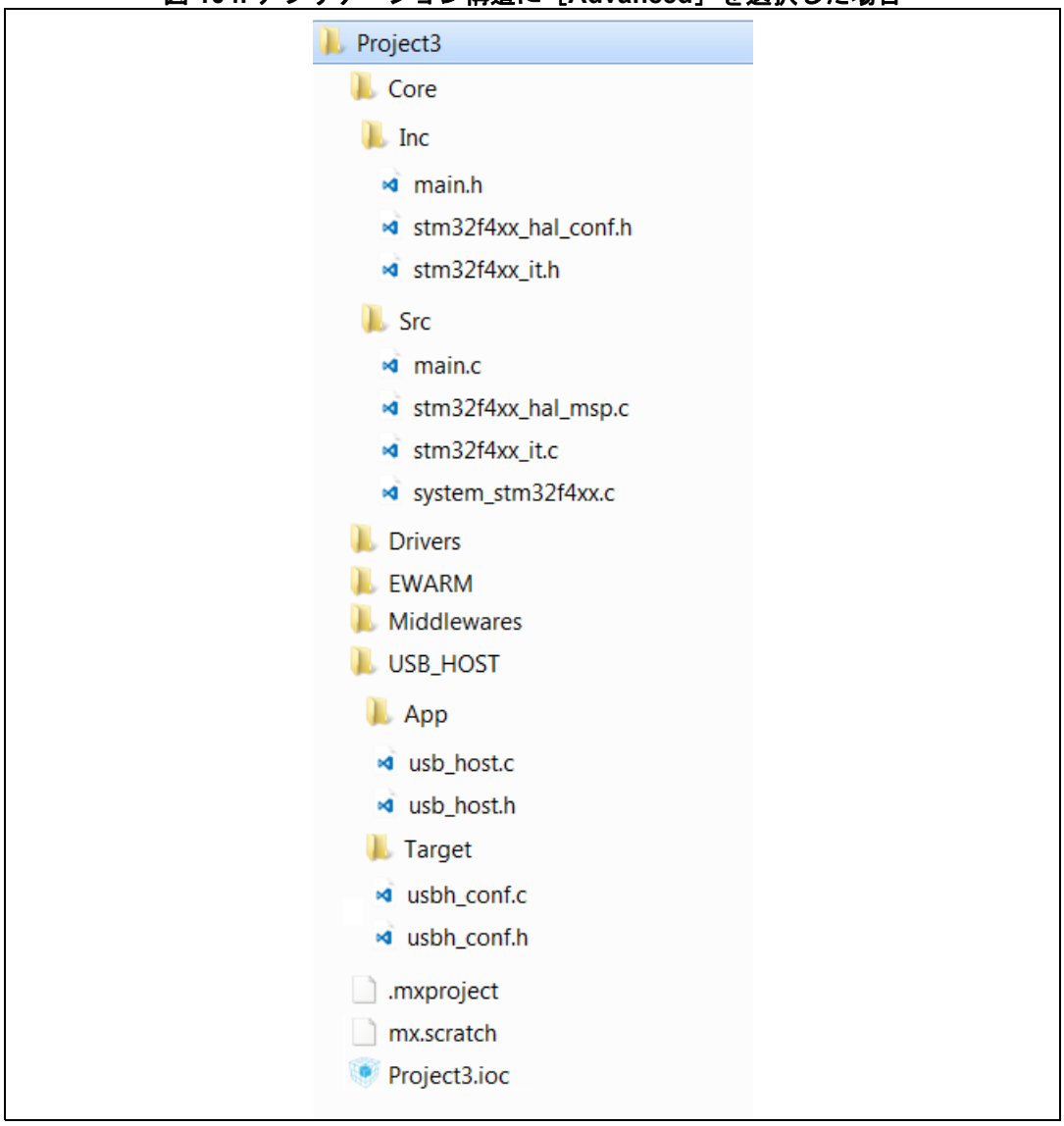



図 105. ［OpenSTLinux Settings］（STM32MP1 シリーズのみ）

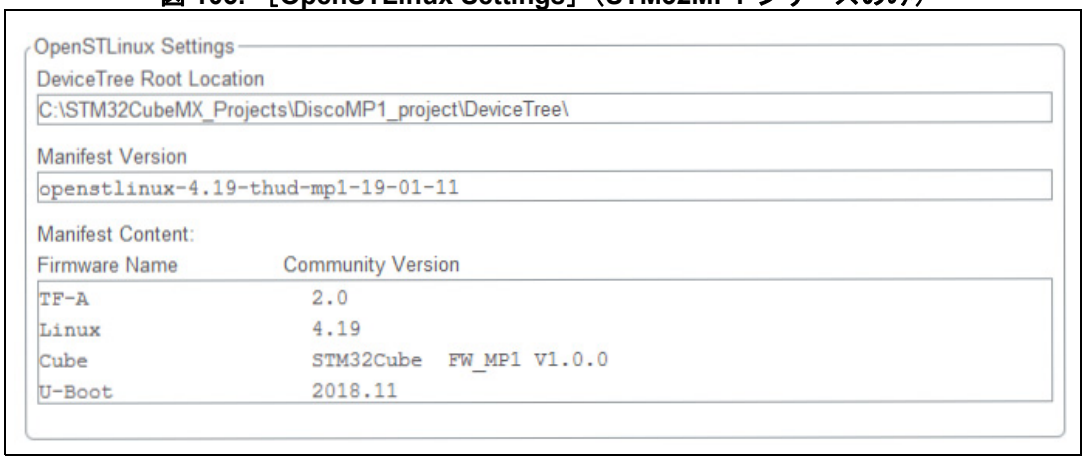
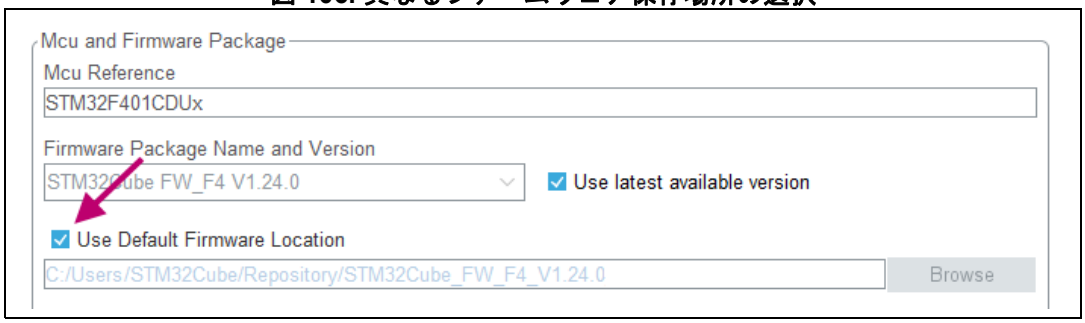
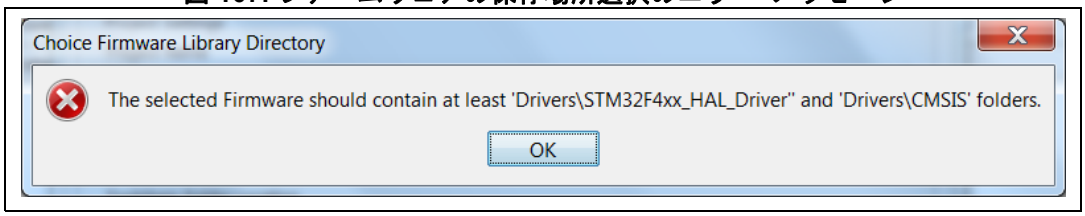


図 106. 異なるファームウェア保存場所の選択



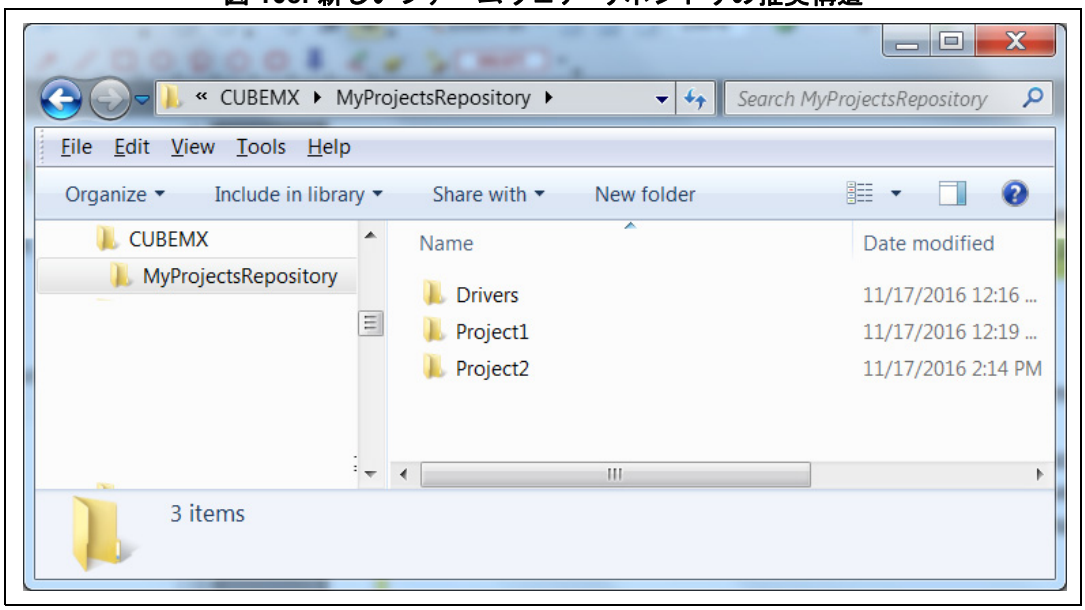

新たに指定する保存場所には、該当の STM32Cube マイクロコントローラ･パッケージから取得した HAL ドライバと CMSIS ドライバを格納した Drivers ディレクトリが存在する必要があります。このフォルダが見つからないと、エラー･メッセージが表示されます（図 107 を参照）。

図 107. ファームウェアの保存場所選択のエラー･メッセージ




共通のファームウェア保存場所を使用するすべてのプロジェクトで同じ Drivers フォルダを再利用するには、**[Code Generator]** タブの **[Add the library files as reference]** を選択します（図 108 を参照）。

図 108. 新しいファームウェア･リポジトリの推奨構造



**注意：** **STM32CubeMX では、このデフォルトの保存場所でのみファームウェア更新を管理できます。別の場所を選択すると、自動更新機能の利点が得られなくなります。その場合は、新しいバージョンのドライバを手動でプロジェクト･フォルダにコピーする必要があります。**

### 4.9.2 ［Code Generator］タブ

**[Code Generator]**タブでは､コード生成に関する以下のオプションを指定できます(図 109 を参照)。

- ［STM32Cube Firmware Library Package］オプション
- ［Generated files］オプション
- ［HAL Settings］オプション
- カスタム･コード･テンプレートのオプション

#### ［STM32Cube Firmware Library Package］オプション

以下の操作が可能です。

- ［Copy all used libraries into the project folder］

  ユーザの設定に関連するドライバのライブラリ（HAL、CMSIS）およびミドルウェア･ライブラリ（FatFs、USB など）が、STM32CubeMX によってユーザのプロジェクト･フォルダにコピーされます。
- ［Copy only the necessary library files］:

  ユーザ設定に関連するライブラリ･ファイル（HAL ライブラリの SDIO HAL ドライバなど）のみが、STM32CubeMX によってユーザのプロジェクト･フォルダにコピーされます。
- ［Add necessary library files as reference in the toolchain project configuration file］

  デフォルトでは、必要なライブラリ･ファイルがユーザのプロジェクトにコピーされます。STM32CubeMX のリポジトリにあるファイルを設定ファイルから参照するには、このオプションを選択します。そのようにすると、ユーザのプロジェクト･フォルダには、ライブラリ･ファイルのコピーではなく､STM32CubeMX のリポジトリにあるファイルへの参照が保持されます。

#### ［Generated files］オプション

この領域では以下のオプションを定義できます。

- ペリフェラルの初期化コードを .c ファイルと .h ファイルのペアとして生成するか、すべてを main.c ファイルに生成するかを指定するオプション。
- 以前に生成したファイルのバックアップをバックアップ･ディレクトリに作成するオプション。

  それまでに生成した .c/.h ファイルには拡張子 .bak が付加されます。

  C コードの再生成時にユーザ･コードを保持するオプション。

  このオプションは､STM32CubeMX で生成したファイルにあるユーザ･セクションのみに適用されます。手動で追加したユーザ･ファイルや ftl テンプレートから生成したユーザ･ファイルには適用されません。
- これまでに生成したファイルを､現在の設定で不要になった場合に削除するオプション。たとえば、以前のコード生成で有効にした UARTペリフェラルを現在の設定で無効にすると、uart.c/.h ファイルが削除されます。

#### ［HAL Settings］オプション

この領域では、HAL の設定に関するオプションを選択できます。

- 消費電力を最適化するためにすべての空きピンをアナログに設定するオプション。
- フル･アサート機能を有効化/無効化するオプション：設定ファイル stm32xx_hal_conf.h に記述した Define 文をコメントアウトするかどうかを指定します。

### カスタム･コード･テンプレートのオプション

カスタム･コードを生成するには、[**Template Settings**]にある[**Settings**]ボタンをクリックして、[Template Settings]ウィンドウを開きます（図 110 を参照）。

このウィンドウでは、使用するコード･テンプレートが保存されているソース･ディレクトリ、および対応するコードの生成先ディレクトリを選択する必要があります。

デフォルトのソース･ディレクトリは、STM32CubeMX のインストール先フォルダにある extra_template を参照します。このフォルダは、すべてのユーザ定義テンプレートを保存することを目的としています。デフォルトの生成先フォルダは、ユーザのプロジェクト･フォルダに置かれます。

テンプレートを選択すると、STM32CubeMX ではそのテンプレートを使用してユーザのカスタム･コードが生成されます（セクション 6.3：カスタム･コードの生成を参照）。

図 111 は、図 110 に示したテンプレート設定による生成結果です。sample_h.ftl テンプレートの定義に従って sample.h ファイルが生成されています。

**図 109. [Project Settings] - [Code Generator]**

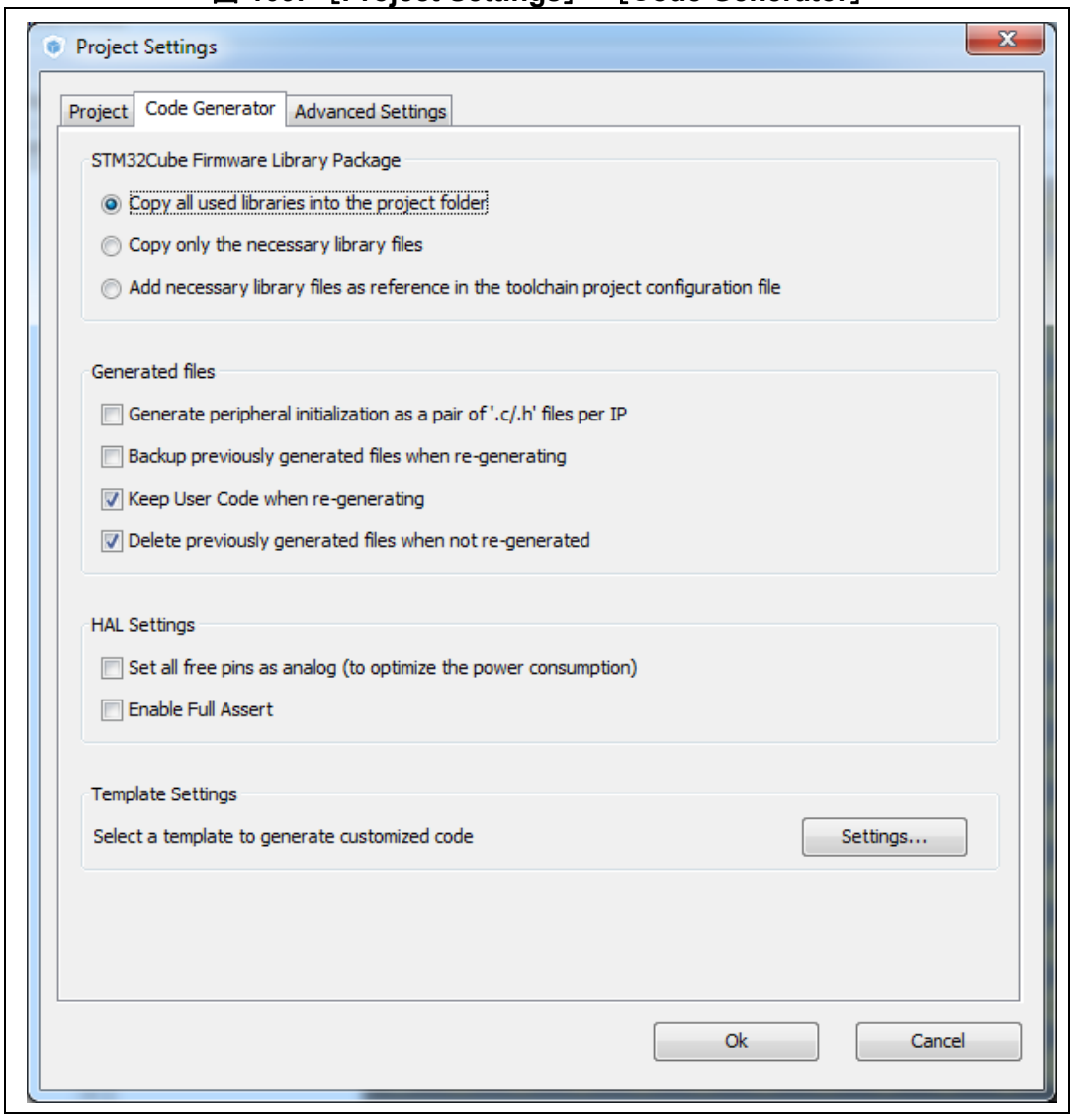


図 110. ［Template Settings］ウィンドウ

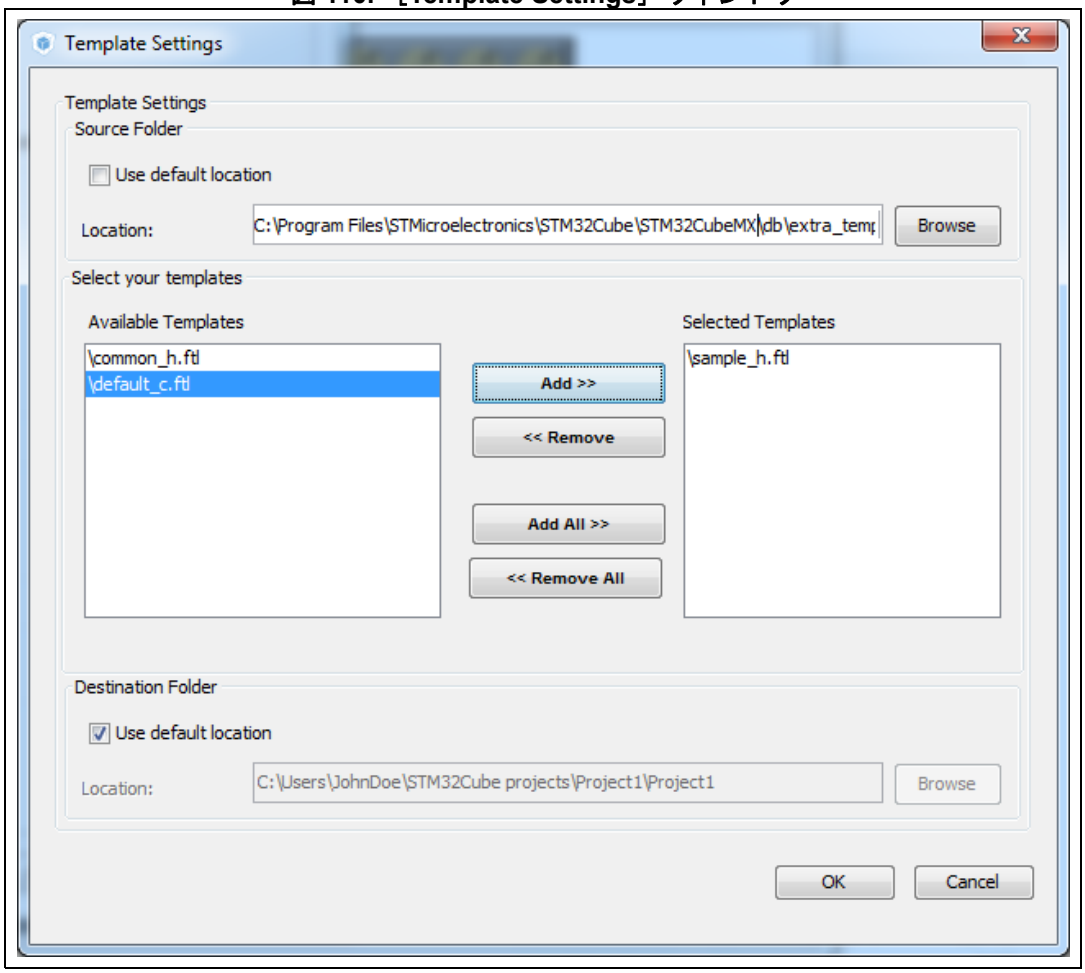
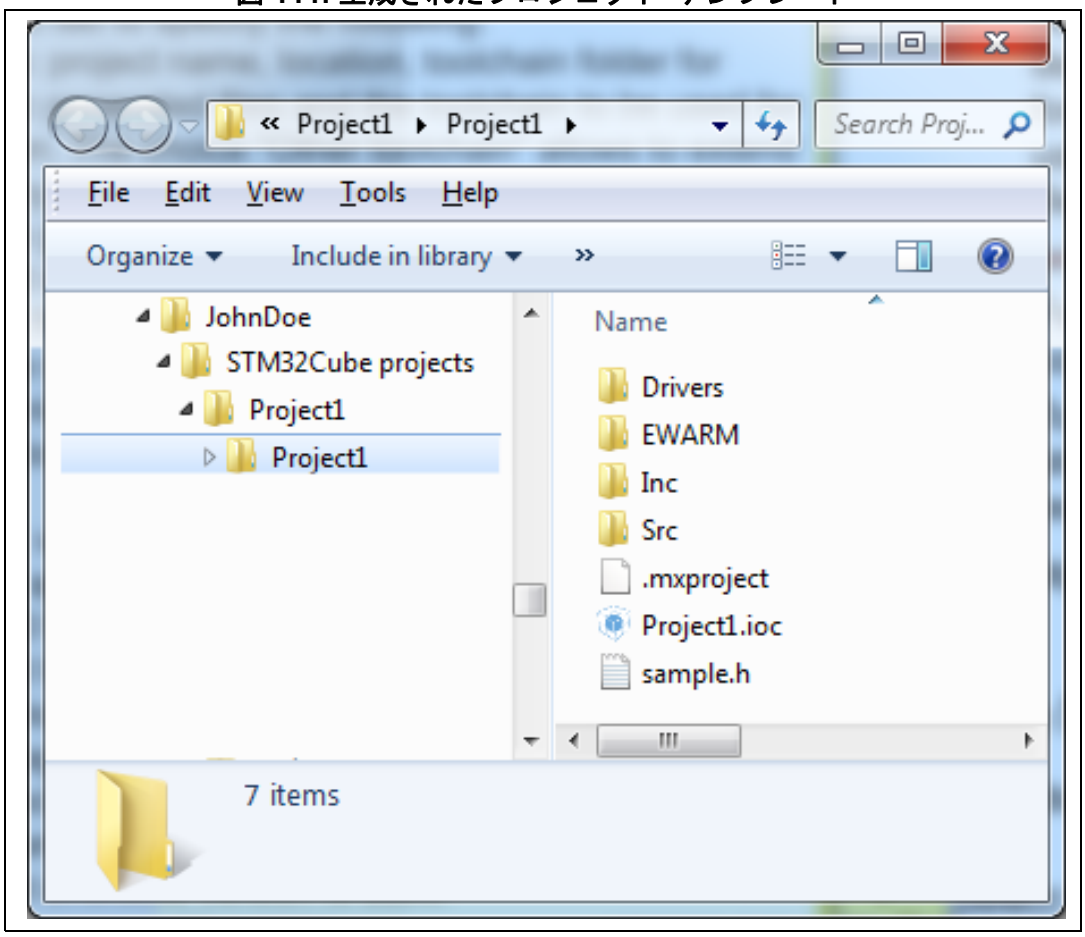


図 111. 生成されたプロジェクト･テンプレート




### 4.9.3 ［Advanced Settings］タブ

このタブには次の 3 つのパネルがあります（図 112 参照）。

- **［Driver Selector］**パネル：ペリフェラル･インスタンスの初期化コードを生成する際に使用するドライバ（HAL または LL）を選択します。
- **［Generated Function Calls］**パネル：関数呼び出しの生成の要否、生成する場合は static として生成するかどうか、どのような順序で生成するかを選択します。
- **［Register CallBack］**パネル：stm32xxxx_hal_conf.h ファイルで定義する必要があるレジスタ･コールバックの対象になるペリフェラルを選択します。

たとえば、このパネルで ADC を有効にすると、次の定義が生成されます。

```
#define USE_HAL_ADC_REGISTER_CALLBACKS 1U
```

#### 一部のペリフェラルまたはミドルウェアの初期化コードを生成しない選択

STM32CubeMX は、デフォルトで初期化コードを生成します。この自動生成は、［Generate Code］列のチェックボックスでペリフェラルごとまたはミドルウェアごとに無効化できます。

### 初期化関数の呼び出し順序の設定

デフォルトでは、生成したコードによってペリフェラルとミドルウェアの初期化関数が、STM32CubeMX でそれらを有効にした順序で呼び出されます。上向きと下向きの矢印ボタンを使用して［Rank］番号を変更することで、この呼び出し順序を変更できます。

リセット･ボタンをクリックすれば、アルファベット順に戻すことができます。

### 初期化関数の呼び出しの無効化

**［Not to be generated］**チェックボックスをチェックすると、該当のペリフェラル初期化関数の呼び出しが STM32CubeMX で生成されなくなります。その呼び出しはユーザ･コード側で実行する必要があります。

### 特定のペリフェラル･インスタンスに対するコード生成を HAL または LL のどちらに基づいて実行するかの選択

STM32CubeMX 4.17 以降および STM32L4 シリーズでは、一部のペリフェラルに対して、HAL ドライバの代わりに Low Layer（LL）ドライバに基づく初期化コードを STM32CubeMX で生成できます。**［Driver Selector］**セクションで LL ドライバまたは HAL ドライバを選択できます。その設定に従ってコードが生成されます（セクション 6.2 : Low Layer ドライバを使用する STM32Cube コードの生成を参照）。

**図 112. ［Advanced Settings］ウィンドウ**

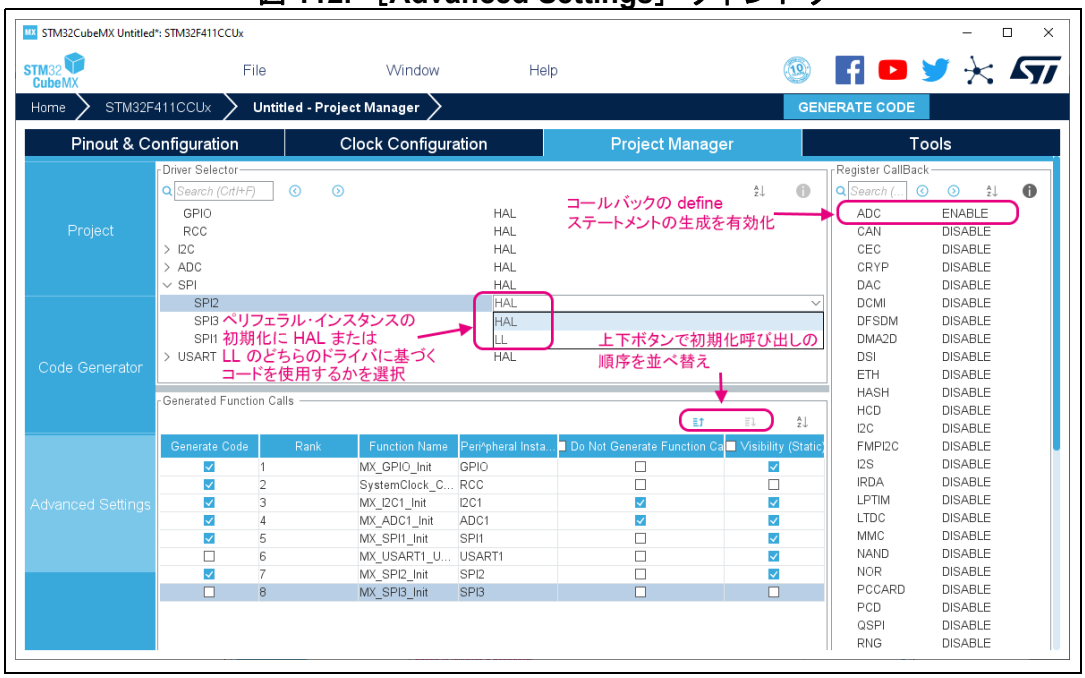



図 112 に示す MX_I2C1_init 関数のように、**［Visibility（Static）］**オプションを選択しないと、キーワード static を指定しない関数定義を生成できるので、その関数を現在のファイル外でも認識できる

ようになります（図 113 を参照）。

**図 113. C 言語の「static」キーワードを指定せずに生成した初期化関数**

```
/* Private function prototypes --------
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_LPTIM1_Init(void);
static void MX_LPTIM2_Init(void);
void MX_I2C1_Init(void);
static void MX_I2C2_Init(void);
static void MX_SPI1_Init(void);
static void MX_SPI2_Init(void);
static void MX_USART1_UART_Init(void);
static void MX_USART2_Init(void);
```

**注意：** **STM32MP1 シリーズのみ：**
**デフォルトでは、SystemClock_Config 関数が STM32Cube Cube ファームウェアの main() 関数で呼び出されます。[Project Manager] → [Advanced Settings] パネルの [Do Not Generate Function Call] 列のチェックボックスが、デフォルトではチェックされていないためです（図 112 参照）。この設定は、STM32Cube ファームウェアをエンジニアリング･モード（Cortex-M4 スタンドアロン･モード）で実行するときに有効です。STM32Cube ファームウェアをプロダクション・モードで実行するときは無効です。有効にするには、main() 関数で SystemClock_Config() が呼び出されないように、[Project Manager] → [Advanced Settings] パネルの [Do Not Generate Function Call] 列のチェックボックスをチェックする必要があります。**

## 4.10 ［Import Project］ウィンドウ

**［Import Project］**メニューには、保存済みの設定を他のマイクロコントローラに容易に移植できる機能が用意されています。デフォルトでは以下の設定がインポートされます。

- **［Pinout］**タブ：マイクロコントローラのピンとそれに対応するペリフェラル･モード。インポート先のマイクロコントローラに同じペリフェラル･インスタンスが存在しない場合、インポートは失敗します。
- **［Clock configuration］**タブ：クロック･ツリーのパラメータ
- **［Configuration］**タブ：ペリフェラルおよびミドルウェア･ライブラリの初期化パラメータ
- **［Project Settings］**：ツールチェーンの選択およびコード生成のオプション

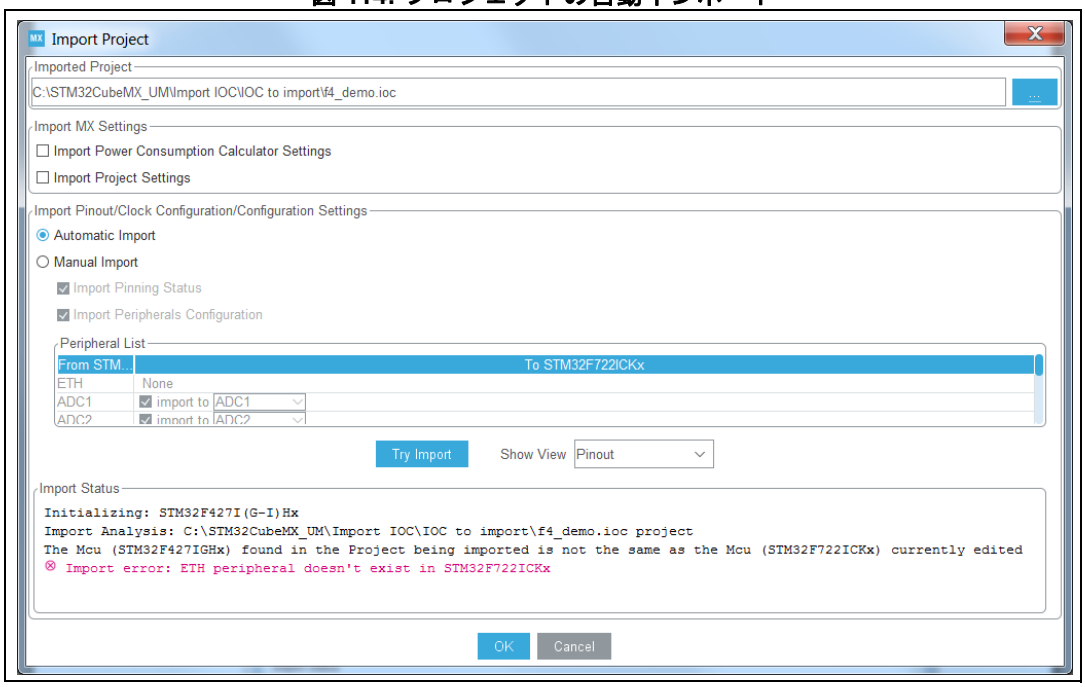
プロジェクトをインポートするには以下の手順に従います。

1. 新規プロジェクトを開始してマイクロコントローラを選択すると**［File］**メニューに表示される**［Import project］**アイコン  を選択します。
   新規プロジェクトに何の設定も定義せず、マイクロコントローラを選択した直後の状態にあれば、このメニューは有効な状態で保持されます。プロジェクトに何らかを設定すると、このメニューは無効になります。
2. プロジェクトのインポート専用のウィンドウを開くには、**［File］→［Import Project］**を選択します。このウィンドウでは、以下のオプションを指定できます。
   – 現在の空のプロジェクトにインポートするプロジェクトの STM32CubeMX 設定ファイル（.ioc）のパス名。
   – **［Power Consumption Calculator］**タブで定義した設定をインポートするかどうかの指定。

– **[Project] → [Settings]** メニューを介して定義したプロジェクト設定（IDE の選択、コード生成のオプション、および詳細設定）をインポートするかどうかの指定。
– **[Project] → [Settings]** メニューを介して定義したプロジェクト設定（IDE の選択、コード生成のオプション）をインポートするかどうかの指定。
– 設定全体をインポートするか（自動インポート）、設定の一部のみをインポートするか（手動インポート）の指定。

a) プロジェクトの自動インポート（図 114 を参照）

**図 114. プロジェクトの自動インポート**

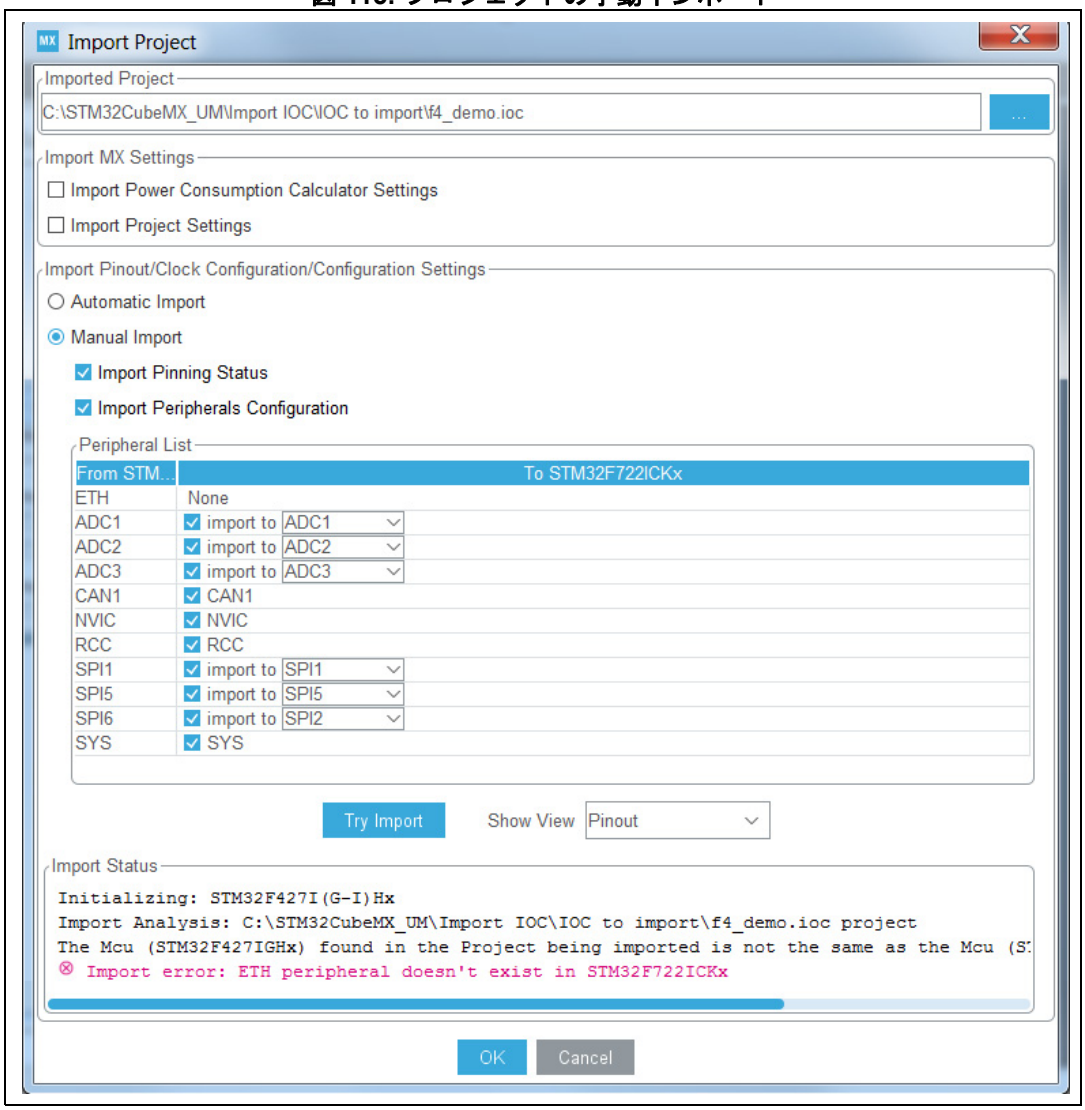


b) プロジェクトの手動インポート

この場合、チェックボックスを使用して目的のペリフェラルを手動で選択できます（図 115 参照）。

インポートするには［**Try Import**］オプションを選択します。

**図 115. プロジェクトの手動インポート**



［Peripheral List］のセクションには以下の情報が表示されます。

– インポートするプロジェクトに設定されているペリフェラルのインスタンス。
– 現在選択しているマイクロコントローラに対して設定されているペリフェラルのインスタンスのうち、設定をインポートする必要があるもの。インポート先とすることができるペリフェラル･インスタンスが複数存在する場合は、そのいずれか 1 つを選択する必要があります。

ピン数が少ない小型パッケージやペリフェラル･オプションが少ないローエンドのマイクロコントローラをインポートすると、競合が発生する可能性があります。

そのような競合がないかどうかを確認するには**[Try Import]**をクリックします。[Import Status]セクションと［Peripheral List］セクションが更新され、インポートが正常に終了したかどうかが示されて、正常に終了していない場合はエラー（図 116 を参照）または警告が表示されます。

– 警告アイコンは、ペリフェラル･インスタンスを複数回選択しているため、インポート要求の一方が実行されないことを示しています。
– ×印は、ピン配置に競合があり、そのままでは設定をインポートできないことを示しています。

手動インポートを使用することで、インポートの選択基準を詳しく設定し、インポートの試行で判明した問題点を解決できます。図 117 は、インポートの試行が成功した例です。いくつかのペリフェラルに対するインポート要求の選択を解除することによって、この結果が得られています。

**[Show View]**機能により、各種の設定タブ（ピン配置、クロック･ツリー、ペリフェラル設定）を切り換えて、実際に現在のプロジェクトにインポートする前に、「インポートの試行」でインポートの影響を確認できます（図 117 を参照）。

図 116. ［Import Project］メニュー - インポートの試行で発生したエラー

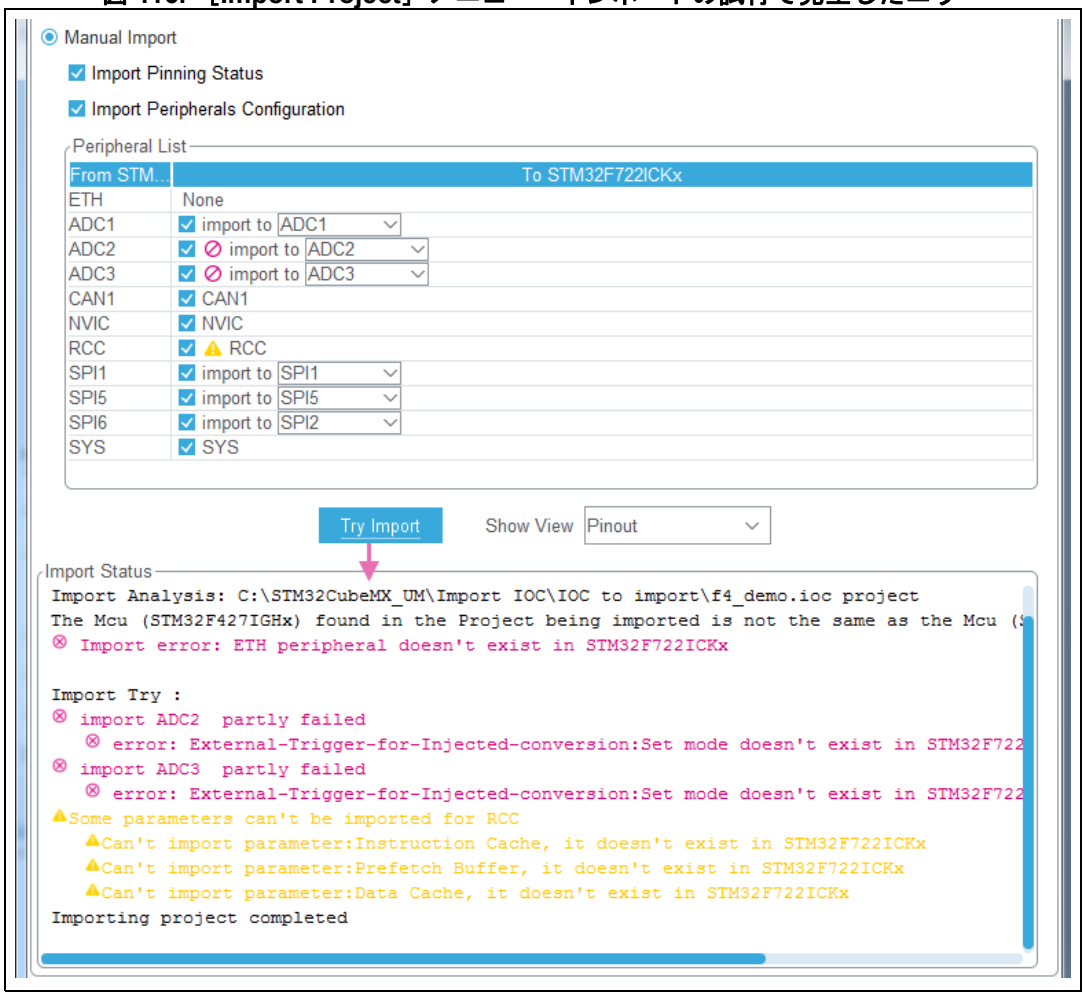
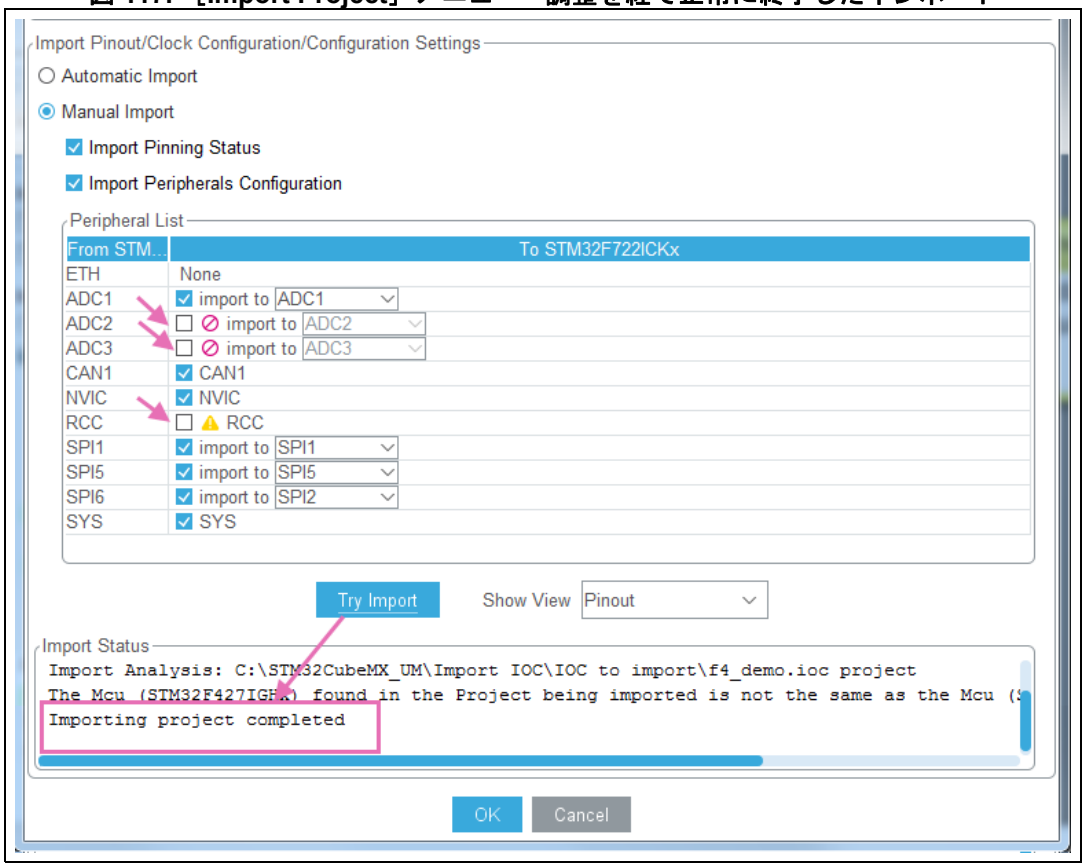


図 117. ［Import Project］メニュー - 調整を経て正常に終了したインポート




3. 現在の状態でインポートを実行する場合は **[OK]**、インポートせずに空のプロジェクトに戻る場合は **［Cancel］** をそれぞれ選択します。

   インポートが完了すると、インポート･アイコンは灰色表示に変化します。マイクロコントローラが設定済みとなり、以降は空ではない設定をインポートできないためです。

## 4.11 ［Set unused GPIOs］/［Reset used GPIOs］ウィンドウ

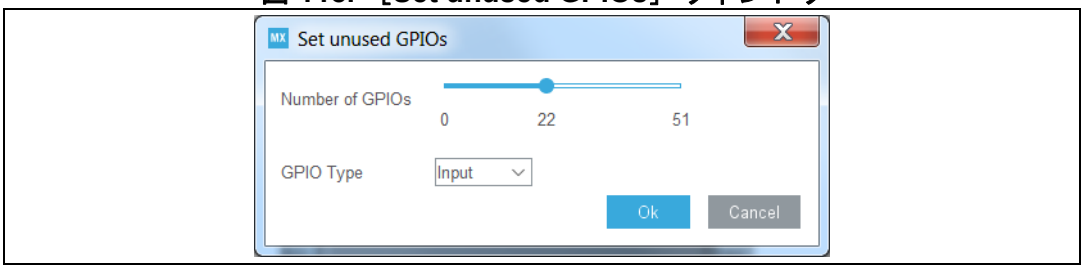
これらのウィンドウでは、一度に複数のピンを同じ GPIO モードに設定できます。

これらのウィンドウを開くには以下の手順を実行します。

- STM32CubeMX のメニュー･バーから、**［Pinout］→［Set unused GPIOs］**を選択します。

**注：** **GPIO の数を選択すれば、使用可能なピンの中から実際に設定またはリセットするピンが STM32CubeMX によって選択されます。**

**図 118. ［Set unused GPIOs］ウィンドウ**




- STM32CubeMX のメニュー･バーから、**［Pinout］→［Reset used GPIOs］**を選択します。

  ツールバーで［Keep Current Signals Placement］オプションをチェックしているかどうかに応じて、使用されていない他の GPIO に GPIO シグナルを移動できるかどうかが STM32CubeMX の競合解決機能によって判断されます。

  – ［Keep Current Signals Placement］オプションをチェックしていない場合、STM32CubeMX の競合解決機能では、別のペリフェラル･モードに適合するように GPIO シグナルを未使用のピンに移動できます。

  – ［Keep Current Signals Placement］オプションをチェックしている場合、GPIO 信号が移動しないので、設定できるペリフェラル･モードの数が制限されます。

  図 120 と 図 121 を参照して、設定できるペリフェラル･モードに対する制限を確認してください。

**図 119. ［Reset used GPIOs］ウィンドウ**

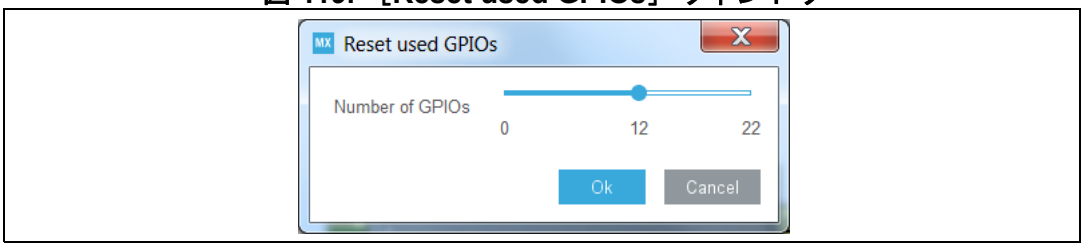


**図 120. ［Keep Current Signals Placement］オプションをチェックした状態で設定された未使用 GPIO ピン**

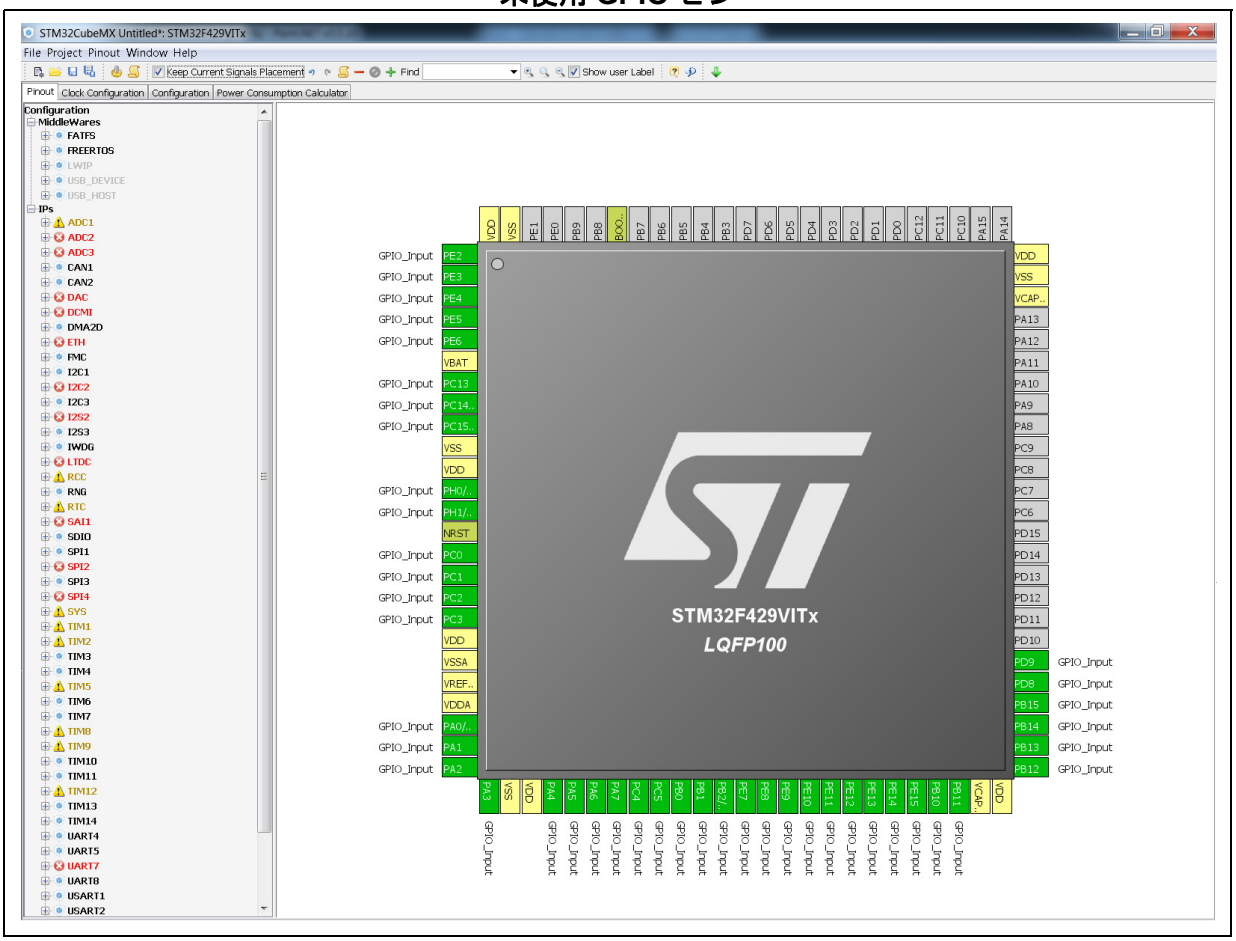
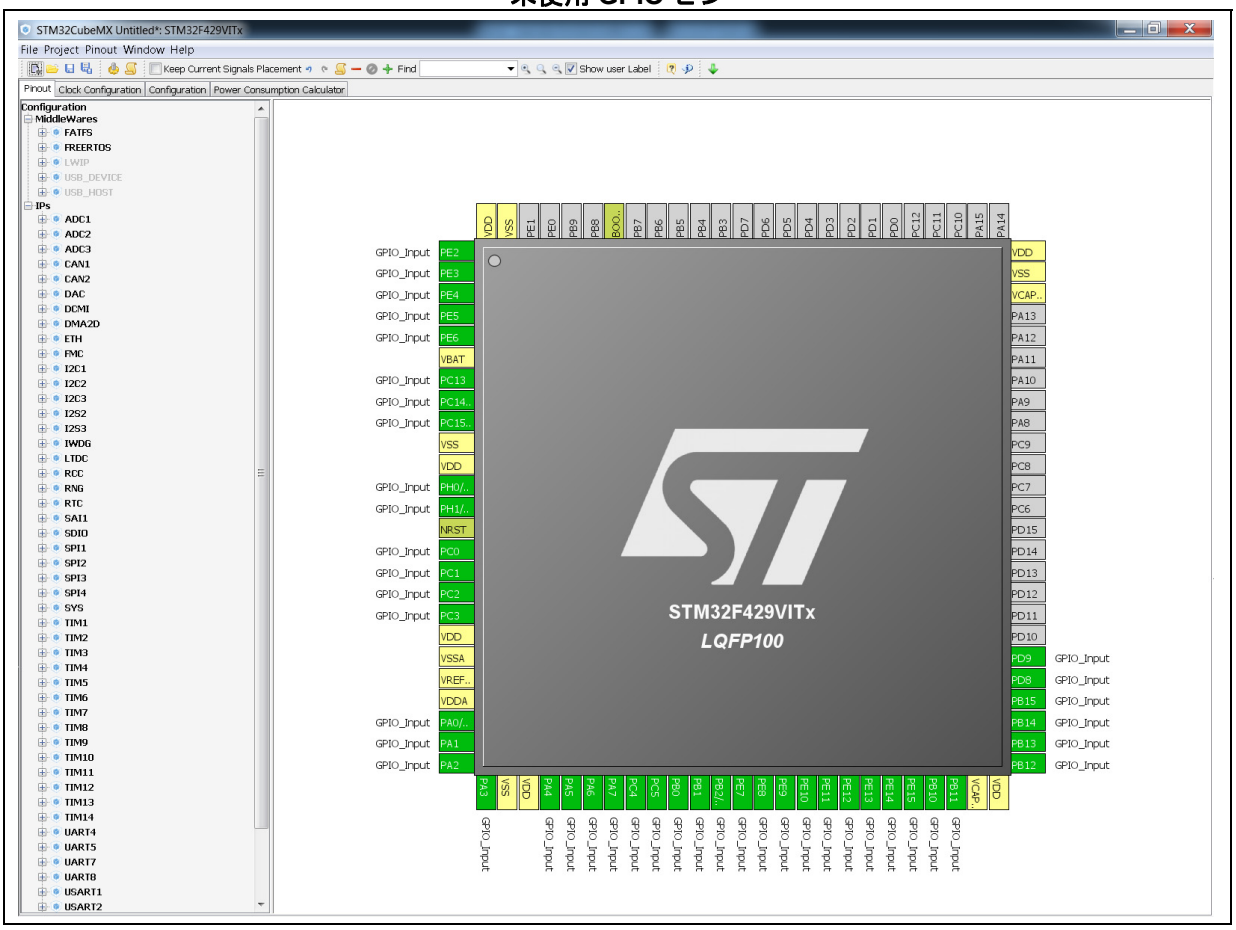


**図 121. ［Keep Current Signals Placement］オプションのチェックを外した状態で設定された未使用 GPIO ピン**




## 4.12 ［Update Manager］ウィンドウ

STM32CubeMX のメニュー･バーにある［**Help**］メニューから、次の 3 つのウィンドウにアクセスできます。

1. ［**Help**］→［**Check for updates**］を選択すると［**Check Update Manager**］ウィンドウが開き、ダウンロードできる最新のソフトウェア･バージョンがあるかどうかを確認できます。
2. ［**Help**］→［**Manage embedded software packages**］を選択すると［**Embedded Software Package Manager**］ウィンドウが開き、ダウンロードできる組込みソフトウェア･パッケージがあるかどうかを確認できます。パッケージ更新の有無の確認や、インストール済みソフトウェア･パッケージの削除も可能です。
3. ［**Help**］→［**Updater settings**］を選択すると［**Updater Settings**］ウィンドウが開き、更新機能を設定できます（プロキシの設定、手動更新と自動更新の切り換え、組込みソフトウェア･パッケージを保存するリポジトリ･フォルダの場所など）。

これらのウィンドウの詳細についてはセクション 3.4 : STM32CubeMX による更新の取得を参照してください。

## 4.13 ［Additional software component selection］ウィンドウ

プロジェクトで作業しているときであれば、**[Pinout & Configuration]**タブの**[Additional Software]**をクリックすることで［Additional software component selection］ウィンドウを開くことができます。現在のプロジェクトに追加するソフトウェア･コンポーネントを選択できます。この機能は、現在、マルチコアの製品には対応していません。

このウィンドウは 図 122 に示すように、4 つのパネルで構成されます。

- **［Filters］**パネル

  ［Show/hide filters］ボタンで表示と非表示を切り替えることができます。このパネルはウィンドウの左側に配置され、パッケージ･コンポーネント･リストに適用するフィルタ基準を設定します。

- **［Packs］**パネル

  プロジェクトで使用するために選択できるソフトウェア･コンポーネントをパッケージ単位で一覧表示するメイン･パネルです。

- **［Component dependencies］**パネル

  ［Show/hide dependencies］ボタンで表示と非表示を切り替えることができます。［Packs］パネルで選択したコンポーネントに依存関係がある場合に、それが表示されます。何らかの依存関係が見つかると解決法が提示されます。

  解決されない依存関係は、赤紫色のアイコンでハイライトされます。

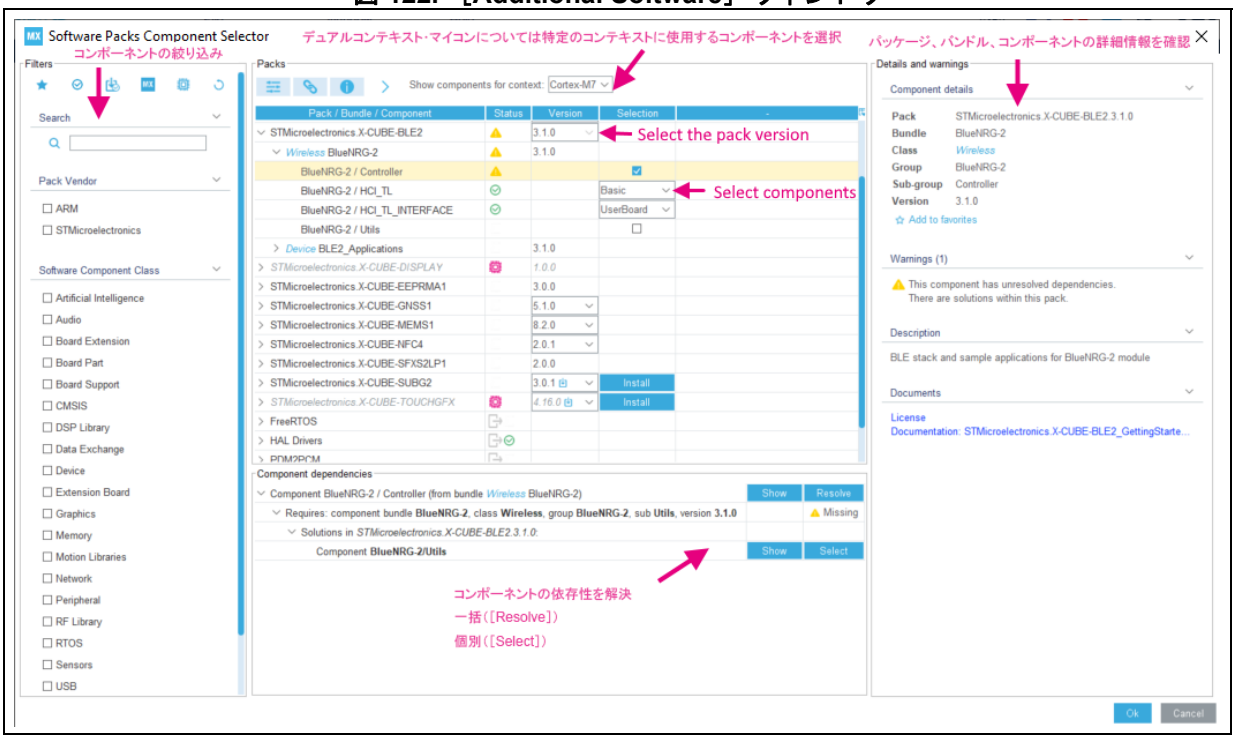
  解決法の候補からコンポーネントを選択することによって解決した依存関係は緑色のアイコンでハイライトされます。

- **［Details and warnings］**パネル

  ［Show/hide details］ボタンで表示と非表示を切り替えることができます。このパネルはウィンドウの右側に配置されます。［Pack］パネルで選択したエレメントの情報を表示します。

  このエレメントとして、パッケージ、バンドル、コンポーネントが等が表示されます。入手可能でありながら、まだインストールされてないバージョンのパッケージをインストールするオプションが提示されます。現在のプロジェクトをより新しいバージョンのパッケージに移行できる機能を提供し、非互換性が表示され、自動的には解消できません。

図 122. ［Additional Software］ウィンドウ




STM32CubeMX の 統合 CMSIS-Pack を通じて追加ソフトウェア･コンポーネントを処理する方法の詳細については、セクション 10 : CMSIS-Pack 規格に基づく追加ソフトウェア･ コンポーネントのサポートを参照してください。

### 4.13.1 ソフトウェア･コンポーネントの概要

Arm® Keil™ CMSIS-Pack 規格では、ソフトウェア･パッケージとして配布するソフトウェア･コンポーネントのパッケージ･フォーマット（*.pdsc）を規定しています。ソフトウェア･パッケージとは、*.pdsc 記述ファイルを収めた ZIP ファイルです。

STM32CubeMX では、パッケージの .pdsc ファイルを解析してソフトウェア･コンポーネントのリストを抽出します。このリストが、［Packs］パネルに表示されます。

Arm® Keil™ CMSIS-Pack 規格は、ソフトウェア･コンポーネントをファイルのリストとして定義します。これらのコンポーネントまたは対応する個別ファイルのそれぞれで、必要に応じて、満足すべき条件を参照できます。この条件を満足しないコンポーネントやファイルは、所定のコンテキストに適用されません。これらの条件が **［Component dependencies］**パネルに表示されます。

コンポーネントには名前がありません。各コンポーネントは、クラス名、グループ名、およびバージョンの組合せによって、所定のベンダのパッケージとして一意に識別されます。サブグループやバリアントなどのカテゴリを追加で割り当てることもできます。これらの詳細が **［Details and warnings］**パネルに表示されます。

### 4.13.2 ［Filters］パネル

 をクリックすると［Filters］パネルが開きます。

ソフトウェア･コンポーネント･リストをフィルタ処理するには、パッケージのベンダ名とソフトウェア･コンポーネント･クラスを選択するか、検索フィールドにテキスト文字列を入力します。

フィルタ処理したソフトウェア･コンポーネントは、縮小表示の表として示されます。左矢印をクリックすると表が展開されて、フィルタ条件に一致するすべてのコンポーネントが表示されます。

**表 14. ［Additional Software］ウィンドウ - フィルタのアイコン**

| アイコン | 説明 |
|---|---|
|  | お気に入りパッケージのみを表示します。<br>パッケージをお気に入りに設定するには、**[Details and warnings]**パネルの ☆ Add to favorites をクリックします。 |
|  | 選択したコンポーネントのみを表示します。<br>コンポーネントは［Packs］パネルのチェックボックスで選択します。同じコンポーネントで選択できる実装が複数存在する場合は、該当のバリアントを選択します。 |
|  | インストール済みのパッケージのみを表示します。<br>インストールされていないパッケージの表示と非表示を切り替えることができます。<br>インストールされていないパッケージは  アイコンで識別できます。 |
|  | 現在作業しているバージョンの STM32CubeMX と互換性のあるパッケージのみを表示します。<br>互換性のないパッケージは  アイコンで識別できます。 |
|  | 現在のプロジェクトで使用しているマイクロコントローラとこのバージョンで互換性のあるパッケージのみを表示します。 |
|  | すべてのフィルタをリセットします。 |

### 4.13.3 ［Packs］パネル

［Packs］パネルはデフォルトで縮小表示されます。既知の各パッケージには、名前と特定の 1 つのバージョン（デフォルトでは最新バージョン）が表示されます。パッケージのバージョンの状態またはコンポーネントの状態のハイライトのみを目的とするアイコンが使用されます（［Packs］パネルのアイコンの表を参照してください）。**[Details and warnings]** パネルと **[Component dependencies]** パネルには詳細情報が表示されます。

デフォルトのビューは、左端の矢印をクリックすることで展開表示できます。これによって、その下のレベルであるバンドルまたは最上位コンポーネントが表示されます。最下位のレベルは、コンポーネントのレベルです。

ハイライトされている制約やアクションのアイコンをクリックすると、それに関連する別のパネル（［Details and warnings］または［Component dependencies］の解決パネル）が開きます。

**注：** **パッケージの中には、Arm®のコアあるいは STM32 シリーズまたは STM32 マイクロコントローラに対する条件を伴うものが存在します。それらのパッケージは、選択したマイクロコントローラがその条件を満たしている場合にのみ表示されます。たとえば、"<accept Dcore="Cortex-M4"/>" を条件とするパッケージは、表示はされますが、Arm® Cortex®-M4 コアを搭載していないマイクロコントローラでは灰色で表示され、選択できません。**

**注：** **パッケージは API をプロモートし、それらは［exposed API］エントリに表示される場合があります。API 名をクリックすると、［Details and warnings］パネルで詳細情報を確認できます。API を実**

**装するコンポーネントを選択すると、その API 自体が選択されます。STM32CubeMX は、API の .h 定義ファイルと、API を実装する .c ファイルの両方を持つプロジェクトを生成します。**

**注：　コンポーネントのパネルに灰色で表示されるコンポーネントは読出し専用です。これらは、STM32Cube マイクロコントローラ組込みソフトウェア･パッケージに付属し、STM32CubeMX でネイティブに使用できるソフトウェア･コンポーネント（HAL ペリフェラル･ドライバまたはミドルウェア）です。**

**表 15. ［Additional Software］ウィンドウ - ［Packs］パネルに表示される列**

| 列名 | 説明 |
|---|---|
| ［Pack/Bundle/Component］ | パッケージ･レベルでは、ソフトウェア･パッケージの名前が表示されます。<br>バンドル･レベルでは、クラス名_バンドル名（存在する場合）が表示されます。<br>コンポーネント･レベルでは、グループ名_サブグループ名（存在する場合）が表示されます。<br>クラス名は、Arm CMSIS 規格で標準化されています。(1) |
| ［Version］ | 使用可能なパッケージ･バージョンのリストから選択したバージョンが表示されます。<br>バンドルやコンポーネントは、パッケージのバージョンを継承する場合と、独自の固有バージョンである場合があります。このバージョンは **［Details and warnings］**パネルに表示されます。 |
| ［Selection］ | 存在する実装が 1 つのみの場合は、チェックボックスでコンポーネントを選択します。実装のバリアントが存在する場合は、そのリストから選択します。 |

1. Arm® Keil™ CMIS-Pack のウェブ･サイト（http://www.keil.com）に以下のクラスが記述されています。
   - Data Exchange：データ交換用ソフトウェア･コンポーネント
   - File System：ファイル･ドライブのサポートとファイル･システム
   - Graphics：ユーザ･インタフェース用グラフィック･ライブラリ
   - Network：インターネット･プロトコルを使用するネットワーク･スタック
   - RTOS：リアルタイム･オペレーティング･システム
   - Safety：安全規格に従ってアプリケーション･ソフトウェアを試験するためのコンポーネント
   - Security：セキュアな通信や保存を実現するための暗号化
   - USB：USB スタック
   - Wireless：Bluetooth®、Wi-Fi®、ZigBee® などの通信スタック

**表 16. ［Additional Software］ウィンドウ - ［Packs］パネルのアイコン**

| アイコン | 説明 |
|---|---|
|  | ユーザのお気に入りパッケージのリストにパッケージが追加済みです。<br>お気に入りリストへのパッケージの追加や削除には **［Details and warnings］**パネルを使用します。 |
|  | パッケージのバージョンに STM32CubeMX のバージョンとの互換性がありません。<br>解決策：互換性のあるバージョンを選択します。 |
|  | パッケージのバージョンがインストールされていません。<br>解決策：**［Details and warnings］**パネルに移動して、プロジェクトで使用するパッケージ･バージョンをダウンロードします。 |
|  | 選択できないコンポーネントです。<br>解決策：このコンポーネントが含まれるパッケージをダウンロードします。 |
|  | コンポーネントが選択されていて、1 つ以上の条件がまだ満たされていません。<br>このようなアイコンが表示された行を選択すると、**［Component dependencies］**パネルで依存関係、ステータス、解決策（見つかった場合）のリストが更新されます。 |
|  | コンポーネントが 1 つ以上選択されていて、条件があればそのすべてが満たされています。 |

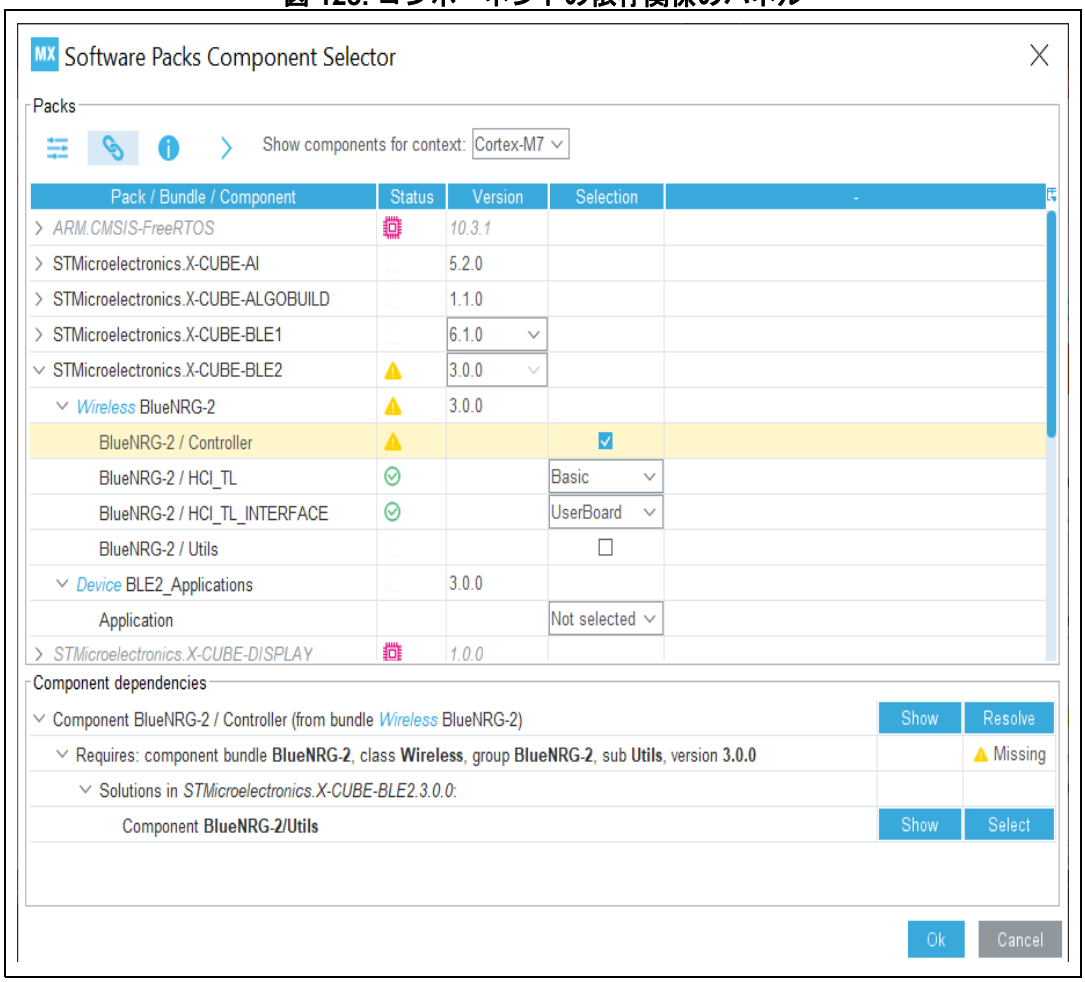
表 16. ［Additional Software］ウィンドウ - ［Packs］パネルのアイコン （続き）

| アイコン | 説明 |
|---|---|
|  | 切り替え先にすることができる別のパッケージ･バージョンを入手できます。<br>解決策：**[Details and warnings]**パネルを使用して変更作業を進めます。 |
|  | 現在選択中のマイクロコントローラに対して STM32CubeMX でネイティブに提供されているコンポーネントをハイライトします。これらは、ペリフェラル･ドライバとミドルウェア･スタックに相当します。<br>それらのコンポーネントの依存関係は自動的には解決できません。STM32CubeMX の［Pinout］ビューに移動して、関連するペリフェラル･インスタンスまたはミドルウェアを［Mode］パネルで有効化します。［Component Selector］には選択済み（緑のチェックボックス）として表示されます。 |

### 4.13.4 ［Component dependencies］パネル

条件とは、依存関係についてソフトウェア･コンポーネントに適用するルールです。コンポーネントを選択したとき、解決が必要な依存関係が存在しなければ緑色のアイコンが表示されます。解決が必要な場合は、警告アイコンが表示されます。  をクリックして依存関係のパネルを開きます（図 123 参照）。

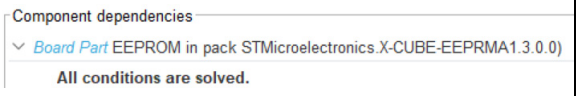
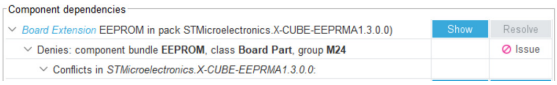
図 123. コンポーネントの依存関係のパネル



コンポーネントを選択するとパネルが更新され、解決が必要な依存関係の詳細が表示されるほか、適用可能な解決策があればそれも表示されます（表 17 参照）。

- ［Show］ボタンをクリックすると依存関係を解決するコンポーネントが表示されます。
- ［Select］ボタンをクリックすると依存関係を解決するコンポーネントを選択できます。
- ［Resolve］ボタンが有効であれば、それをクリックすると依存関係が自動的に解決されます。

**表 17. ［Component dependencies］パネルのコンテキスト･ヘルプ**

| コンテキスト･ヘルプ | 説明 |
|---|---|
|  | 解決が必要な依存関係がありません。 |
|  | 解決が必要な依存関係であり、問題が発生しています（解決策が見つからないか、競合が発生しています）。 |
|  | 解決が必要な依存関係であり、1 つ以上の解決策が見つかりました。 |

### 4.13.5 ［Details and warnings］パネル

 をクリックしてパネルを表示します（図 124 参照）。

このパネルは、**［Packs］**パネルで行を選択すると更新されます。

このパネルからは、次の操作が可能です。

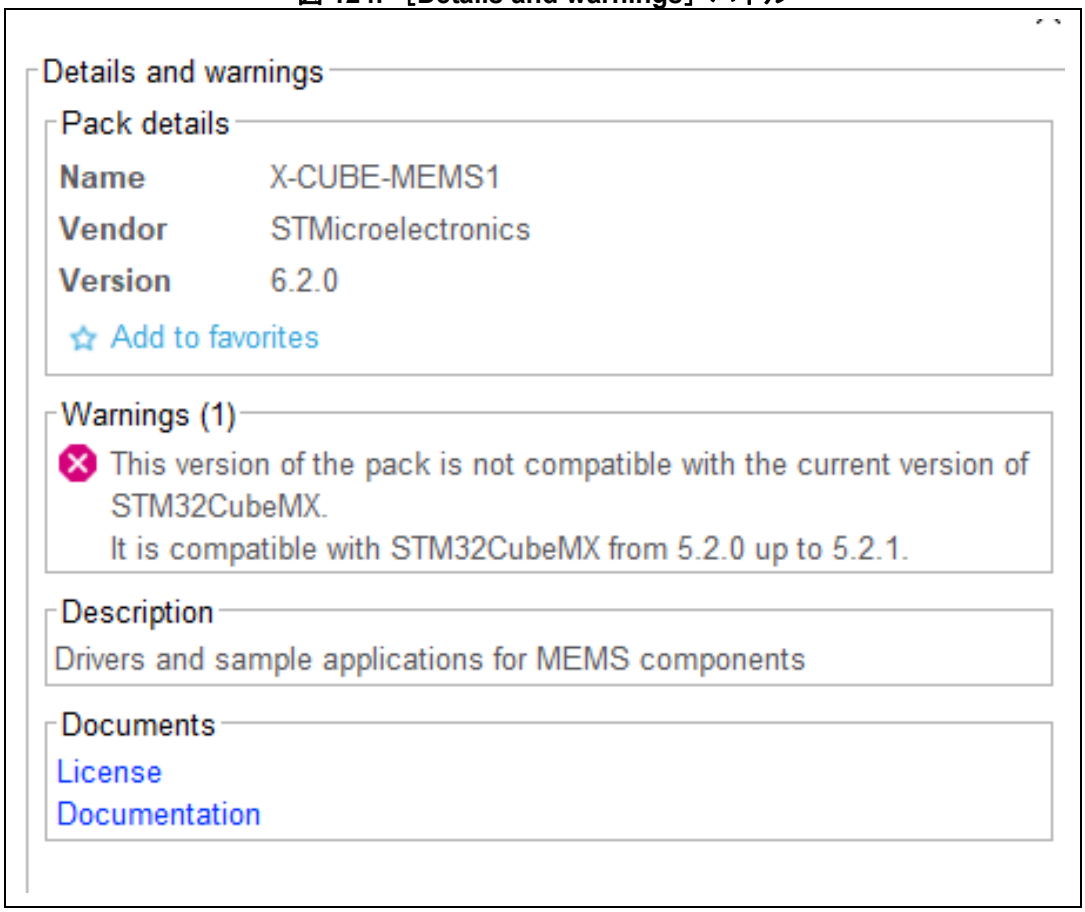
- お気に入りパッケージのリストに対するパッケージの追加と削除
- パッケージのインストール
- リンクを介したパッケージのドキュメントへのアクセス
- 新しいパッケージ･バージョンへのプロジェクトの移行

プロジェクトを新しいソフトウェア･パッケージ･バージョンに移行するには、次の手順を実行します。

1. プロジェクトを開きます。
2. 新しいパッケージ･バージョンに移行します。
3. コードを生成します。

既知の問題：ステップ 3 の後にステップ 2 を実行すると（コード生成後に移行すると）エラーが発生します（誤ったファイル･パスの生成とプロジェクトのコンパイルの失敗）。この問題を修正するには、プロジェクトを新規プロジェクトとして保存し、コードを再度生成する必要があります。このパネルでは、お気に入りパッケージのリストに対するパッケージの追加と削除、パッケージのインストール、リンクを介したパッケージのドキュメントへのアクセスの各操作が可能です。

図 124. ［Details and warnings］パネル




### 4.13.6 追加ソフトウェア･コンポーネントのツリー表示の更新

アプリケーションに必要なソフトウェア･コンポーネントの選択が完了したら（図 125 を参照）、**[OK]** をクリックして STM32CubeMX のウィンドウを更新します。［Additional Software］のツリー表示に、選択したコンポーネントが表示されます（図 126）。

現在選択している追加のソフトウェア･コンポーネントがツリー表示に表示されます（図 126 を参照）。このソフトウェア･コンポーネントは［Mode］パネルで有効化する必要があります。設定パネルで何らかのパラメータが提示された場合は、さらに詳細な設定が可能です。コンポーネント名にマウス･カーソルを置くと、ドキュメントへのリンクを含むコンテキスト･ヘルプが表示されます。

図 125. 追加のソフトウェア･コンポーネントの選択

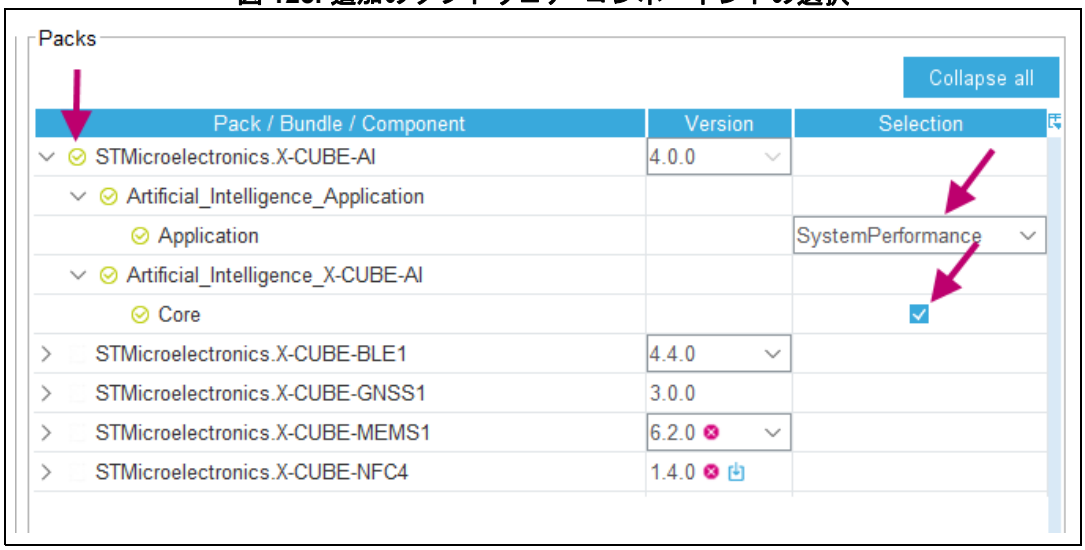
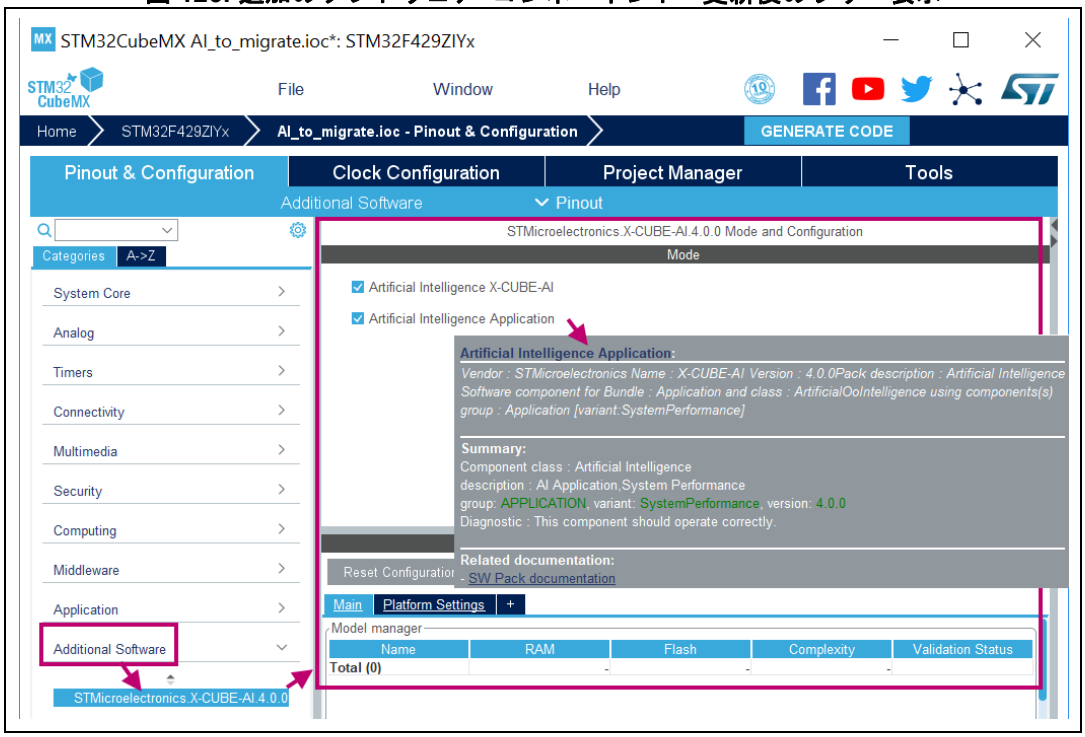



図 126. 追加のソフトウェア･コンポーネント - 更新後のツリー表示



## 4.14 ［LPBAM Scenario & Configuration］ビュー

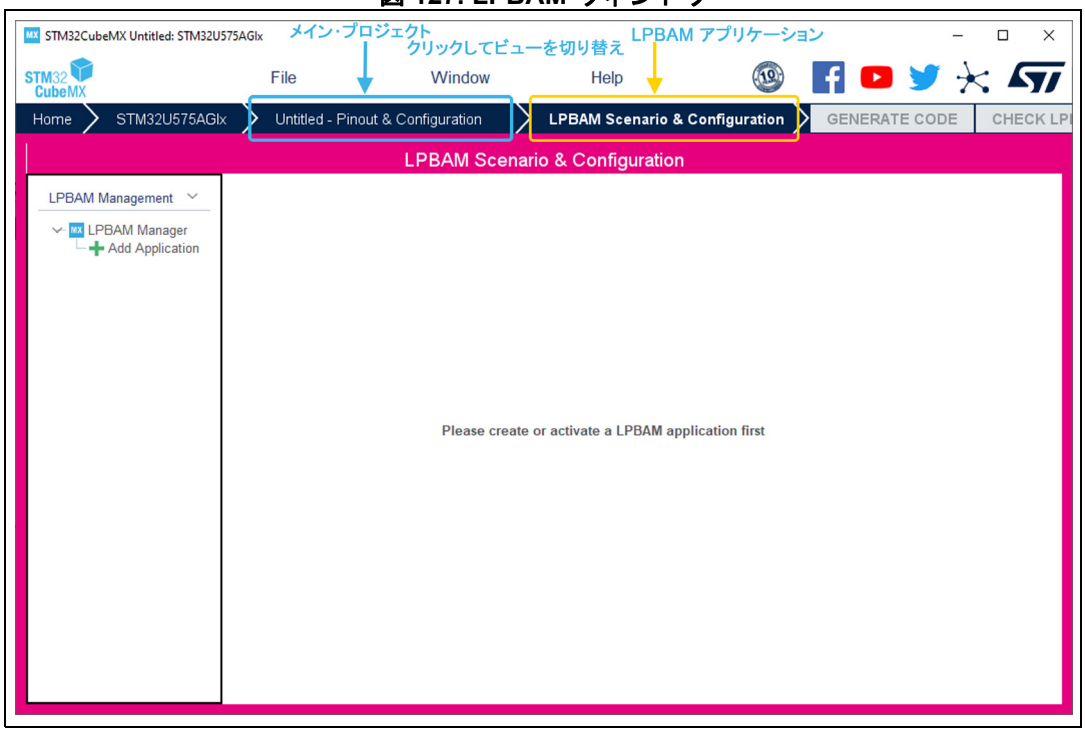
STM32CubeMX 6.5.0 以降、TrustZone をアクティブ化していないプロジェクトと STM32U575/585 製品ラインでは、必要に応じて［LPBAM Scenario & Configuration］ビューを使用し、LPBAM アプリケーションを作成できるようになりました（図 127 参照）。

このビューでは、以下の操作を実行できます。

- LPBAM アプリケーションの追加と削除
- LPBAM アプリケーションごとのキューの作成
- LPBAM ファームウェア API を使用した、キューごとの機能ノードの作成（この API は SmartRun ドメインのペリフェラル用に提供されています）
- SmartRun ドメインのペリフェラルを対象とした、LPBAM アプリケーションごとのピン配置、クロック･ツリー、HAL 関連の設定

このビューの使用方法の詳細は、セクション 18 : LPBAM プロジェクトの作成 を参照してください。

**図 127. LPBAM ウィンドウ**



## 4.15 ［About］ウィンドウ

このウィンドウには STM32CubeMX のバージョン情報が表示されます。

このウィンドウを開くには、STM32CubeMX のメニュー･バーから **[Help]** → **[About]** を選択します。

**図 128. ［About］ウィンドウ**

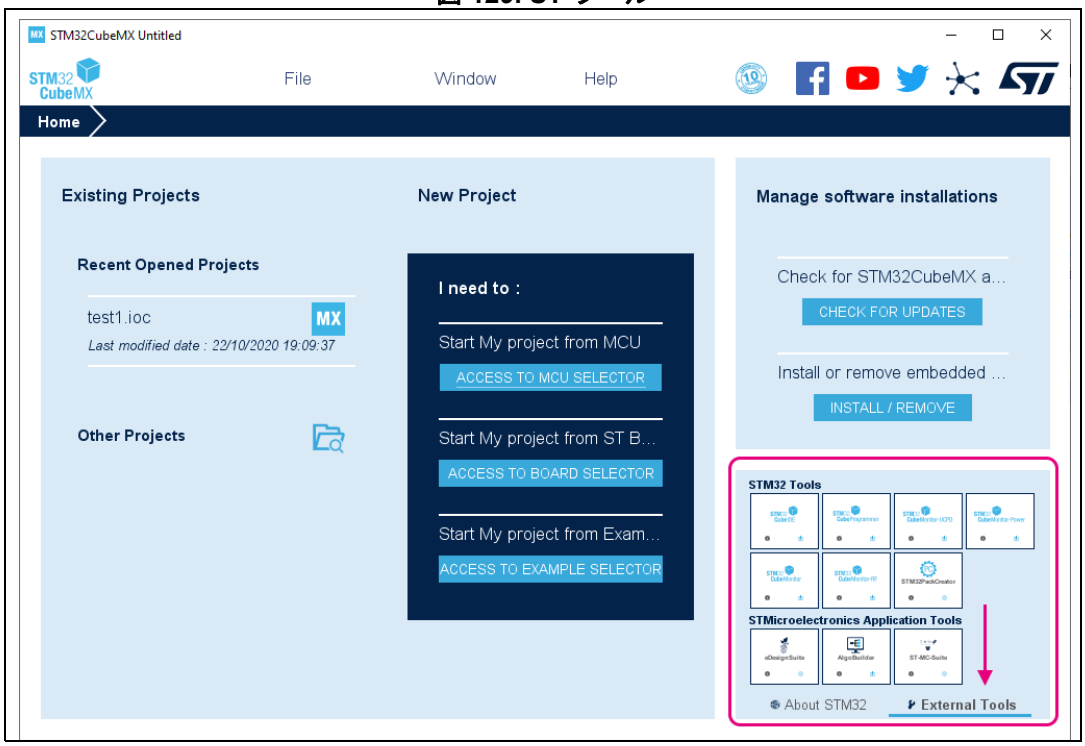
# 5 STM32CubeMX の各種ツール

## 5.1 外部ツール

このパネルには、[Home] ページからアクセスできます。STM32 製品ポートフォリオに関連するツールの概要が表示されます（図 129 参照）。

- をクリックするとツールに関する情報が表示されます。
- をクリックすると www.st.com でツールの Web ページが表示されます。
- をクリックするとツールが起動します。

**図 129. ST ツール**



## 5.2 ［Power Consumption Calculator］ビュー

組込みシステム･アプリケーションの増加に伴い、消費電力が大きな懸案になっています。消費電力を最小限にするために、STM32CubeMX では［**Power Consumption Calculator**］タブを用意しています（図 130 を参照）。ここでは、マイクロコントローラ、バッテリ･モデル、およびユーザ定義の消費電力シーケンスを指定することで、次の結果が得られます。

- 平均消費電流

  消費電力値は、データシートから取得するか、指定したバス周波数またはコア周波数から補完することで得られます。
- バッテリ寿命
- 平均 DMIPS 値

  DMIPS 値は、マイクロコントローラのデータシートから直接取得します。補完や外部入力の操作では得られません。
- 最高周辺温度（$T_{AMAX}$）

  このツールでは、チップの内部消費電力、パッケージタイプ、および最高接合部温度（105 ℃）に基づいて最高周辺温度を計算し、適切な動作条件を実現します。

  現在の $T_{AMAX}$ の実装では、I/O の消費電力が考慮されていません。正確に推定するには、［Additional Consumption］フィールドを使用して、I/O の消費電力を指定する必要があります。I/O の動的な消費電流の数式は、マイクロコントローラのデータシートに記載されています。

開発では、**[Power Consumption Calculator]** ビューを使用して、組込みアプリケーションによる消費電力の推定値を表示し、電力シーケンスの各ステップでこの値を低減できます。

- 省電力モードを使用できる場合は、そのモードを必ず使用します。
- ステップの要件に基づいてクロック･ソースと周波数を調整します。
- フェーズごとに必要なペリフェラルのみを有効にします。

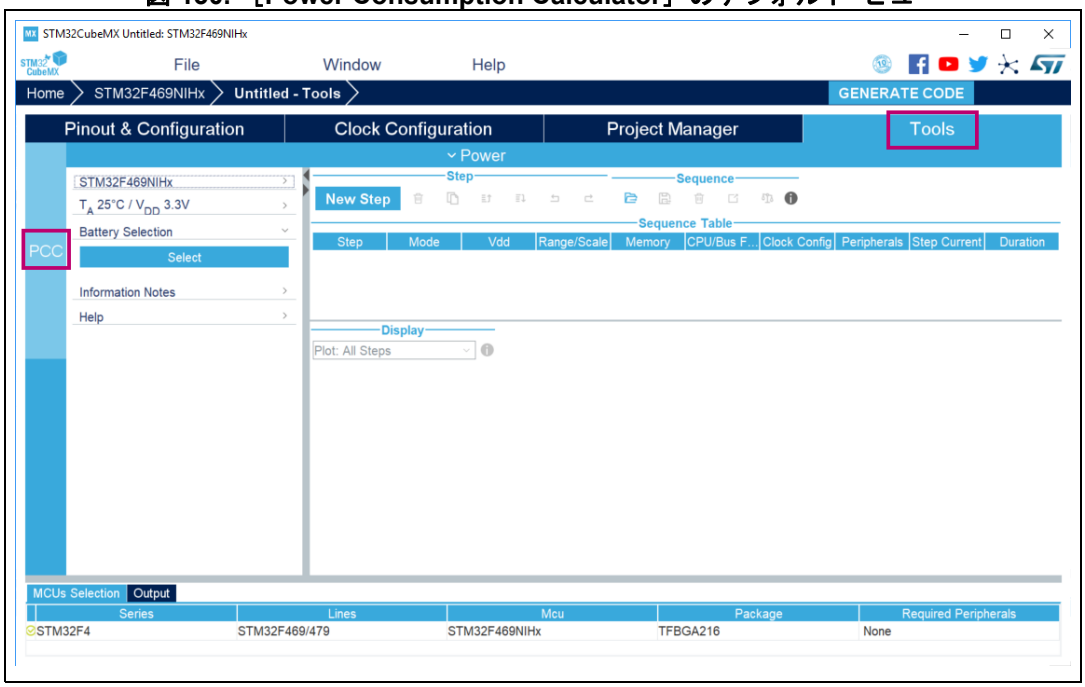
各ステップでは、使用可能な電源としてバッテリではなく $V_{BUS}$ を選択できるので、バッテリ寿命の延長に効果的です。さまざまな電圧レベルにおける消費電力実測値があれば、STM32CubeMX から電圧値の選択肢も提案されます（図 133 参照）。

STM32L0、STM32L1、STM32L4、STM32L4+、STM32G0、STM32G4、STM32H7、STM32WB の各シリーズには、別のオプションとしてトランジション･チェッカーが用意されています。トランジション･チェッカーを有効にすると、現在設定しているシーケンスにある無効なトランジションが検出されます。これにより、新しいステップを追加する際に、使用可能なトランジションのみが必ず提示されます。

### 5.2.1 消費電力シーケンスの構築

最初に表示されるデフォルト・ビューを図 130 に示します。

**図 130. ［Power Consumption Calculator］のデフォルト・ビュー**




#### $V_{DD}$ 値の選択

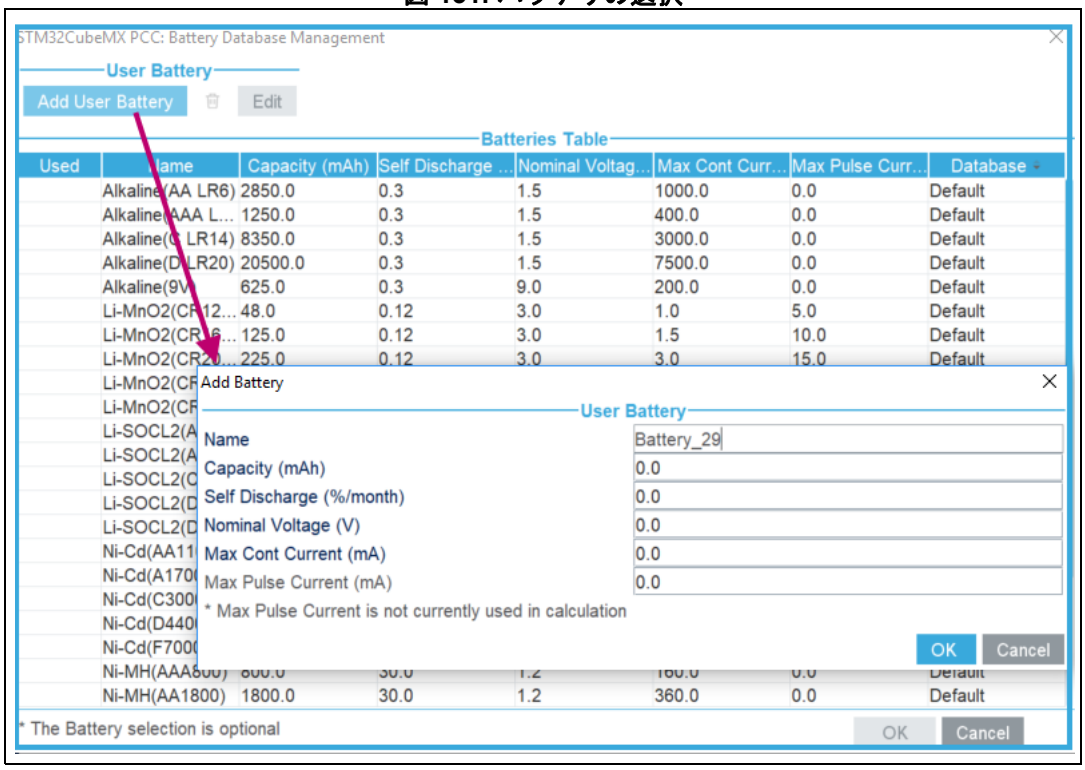
$V_{DD}$ 値に複数の選択肢がある場合は、このビューでいずれかの値を選択する必要があります。

### バッテリ·モデルの選択（オプション）

必要に応じてバッテリ·モデルを選択できます。この選択は、消費電力シーケンスの設定後でも可能です。

事前定義済みのバッテリを選択するか、アプリケーションに最適な新しいバッテリを指定します（図 131 を参照）。

**図 131. バッテリの選択**




### 電力シーケンスのデフォルト·ビュー

これで、電力シーケンスの構築に進むことができます。

### シーケンス·ステップの管理

［Step］の各種ボタンを使用して、シーケンスの中でステップを再構成できます。これらのボタンによる操作として、新しいステップの**追加**、ステップの**削除**、ステップの**複製**、シーケンスでの**上への移動**と**下への移動**があります（図 132 を参照）。

［Power Consumption Calculator］ビューの［Undo］ボタンまたはメイン·ツールバーの［Undo］アイコンをクリックすると、最後に実行した操作をやり直すことができます。Redo は再実行になり、操作は同じです。

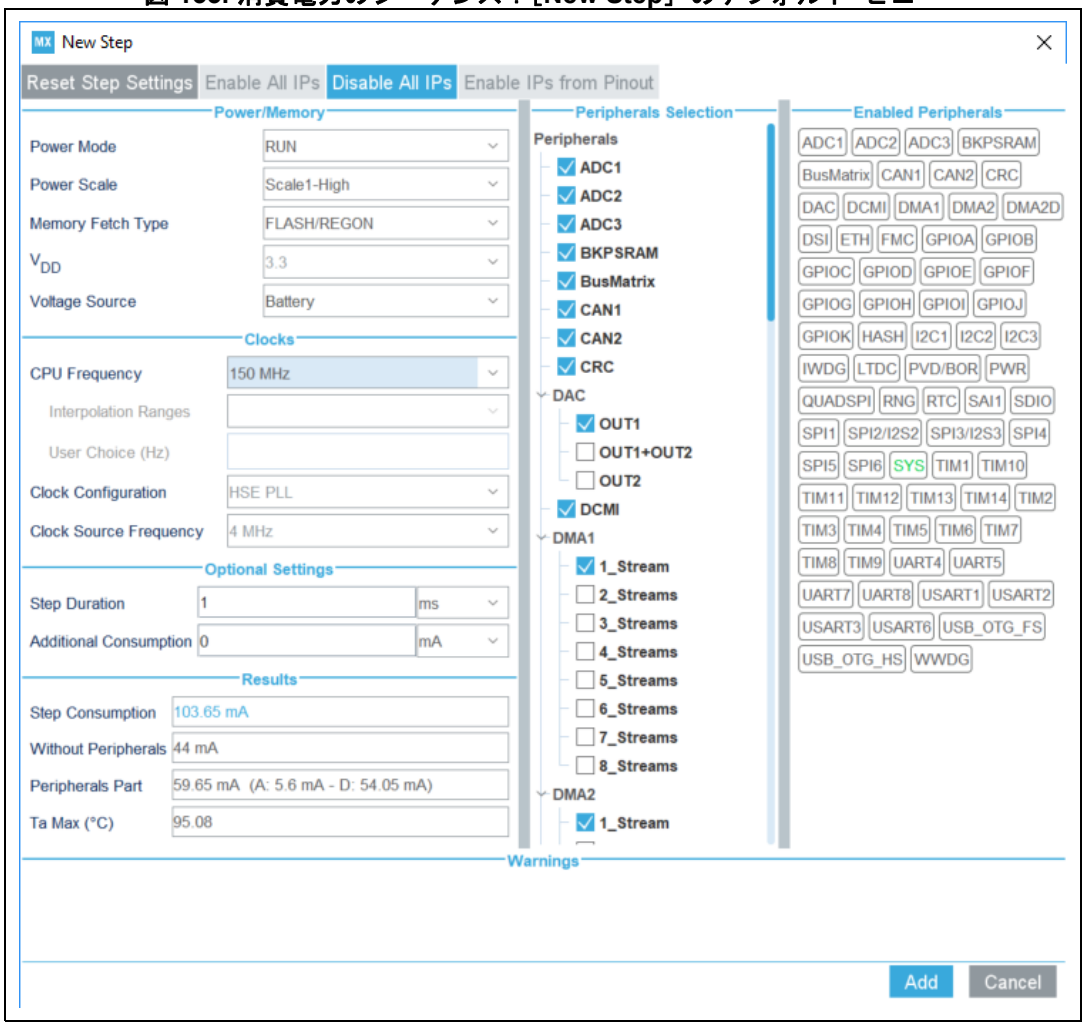
**図 132. ステップ管理機能**

### ステップの追加

新しいステップを追加するには、次の 2 つの方法があります。

- ［Power Consumption Calculator］パネルで **［Add］** をクリックします。ステップ設定が空白の **［New Step］** ウィンドウが開きます。
- もしくは、［Sequence Table］からステップを選択して **［Duplicate］** をクリックします。複製したステップを表示した **［New Step］** ウィンドウが開きます（図 133 参照）。

**図 133. 消費電力のシーケンス：［New Step］のデフォルト・ビュー**




ステップを設定すると、消費電流と $T_{AMAX}$ 値の計算結果が表示されます。

### ステップの編集

ステップを編集するには、目的のステップを［Sequence Table］でダブルクリックして**［Edit Step］**ウィンドウを開きます。

### ステップの移動

デフォルトでは、シーケンスの末尾に新しいステップが追加されます。［Sequence Table］でステップをクリックして選択し、**［Up］**ボタンと**［Down］**ボタンを使用してシーケンスの中で別の順序の位置に移動します。

### ステップの削除

削除するステップを選択して**［Delete］**ボタンをクリックします。

### トランジション･チェッカーの使用

電力モードの間ですべてのトランジションが可能なわけではありません。［Power Consumption Calculator］の電源メニューでは、無効なトランジションを検出するためや、シーケンス設定を有効なトランジションのみに制限するために、トランジション･チェッカーの使用が提案されます。

シーケンスを設定する前にトランジション･チェッカー･オプションを有効にすると、有効なトランジション･ステップのみを確実に選択できます。

設定済みのシーケンスでトランジション･チェッカー･オプションを有効にすると、すべてのトランジションが有効な場合はシーケンスに緑色の枠が表示されます（図 134 参照）。無効なトランジションが 1 つでもある場合は赤紫色の枠が表示され、無効なステップが赤紫色でハイライトされます（図 135 参照）。後者の場合、**[Show log]**ボタンをクリックすると、トランジションの問題を解決する方法を知ることができます（図 136 参照）。

**図 134. 設定済みのシーケンスに対するトランジション･チェッカー･オプションの使用 - すべてのトランジションが有効な場合**

Sequence Table

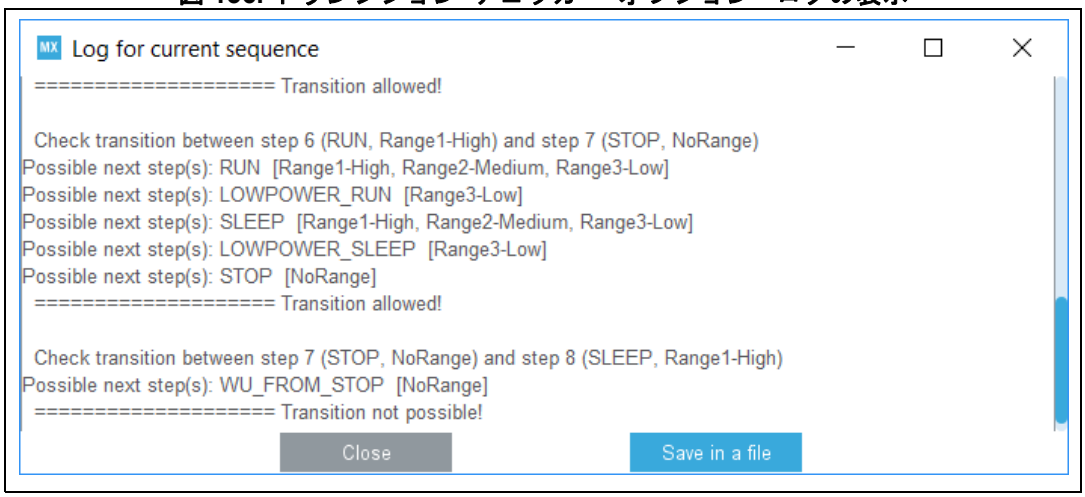
| Step | Mode | Vdd | Range/Scale | Memory | CPU/Bus Freq | Clock Config | Peripherals | Step Current | Duration |
|---|---|---|---|---|---|---|---|---|---|
| 1 | RUN | 3.0 | Range3-Low | FLASH | 1000000 Hz | MSI | | 166.9 µA | 1 ms |
| 2 | RUN | 3.0 | Range2-Medi... | FLASH | 8 MHz | HSEBYP | | 1.3 mA | 1 ms |
| 3 | RUN | 3.0 | Range1-High | FLASH | 8 MHz | HSEBYP | COMP1 COM... | 1.55 mA | 1 ms |
| 4 | SLEEP | 3.0 | Range1-High | FLASH | 8 MHz | HSEBYP | | 380 µA | 1 ms |
| 5 | RUN | 3.0 | Range3-Low | FLASH | 4.2 MHz | MSI | COMP1 COM... | 623.66 µA | 1 ms |
| 6 | RUN | 3.0 | Range1-High | FLASH | 8 MHz | HSEBYP | | 1.55 mA | 1 ms |
| 7 | STOP | 3.0 | NoRange | n/a | 0 Hz | ALL CLOCKS... | | 410 nA | 1 ms |

**図 135. 設定済みのシーケンスに対するトランジション･チェッカー･オプションの使用 - 1 つ以上のトランジションが無効な場合**

Sequence Table

| Step | Mode | Vdd | Range/Scale | Memory | CPU/Bus Freq | Clock Config | Peripherals | Step Current | Duration |
|---|---|---|---|---|---|---|---|---|---|
| 1 | RUN | 3.0 | Range3-Low | FLASH | 1000000 Hz | MSI | | 166.9 µA | 1 ms |
| 2 | RUN | 3.0 | Range2-Medi... | FLASH | 8 MHz | HSEBYP | | 1.3 mA | 1 ms |
| 3 | RUN | 3.0 | Range1-High | FLASH | 8 MHz | HSEBYP | COMP1 COM... | 1.55 mA | 1 ms |
| 4 | SLEEP | 3.0 | Range1-High | FLASH | 8 MHz | HSEBYP | | 380 µA | 1 ms |
| 5 | RUN | 3.0 | Range3-Low | FLASH | 4.2 MHz | MSI | COMP1 COM... | 623.66 µA | 1 ms |
| 6 | RUN | 3.0 | Range1-High | FLASH | 8 MHz | HSEBYP | | 1.55 mA | 1 ms |
| 7 | STOP | 3.0 | NoRange | n/a | 0 Hz | ALL CLOCKS... | | 410 nA | 1 ms |
| 8 | SLEEP | 3.0 | Range1-High | FLASH | 8 MHz | HSEBYP | | 380 µA | 1 ms |

図 136. トランジション･チェッカー･オプション - ログの表示



### 5.2.2 消費電力シーケンスでのステップの設定

［**Edit Step**］ウィンドウと［**New Step**］ウィンドウでステップを設定します。グラフィック･インタフェースでは、事前に定義されている順序でパラメータを設定します。

パラメータの名前は、選択したマイクロコントローラのシリーズに応じて異なります。各パラメータの詳細については、セクション 5.2.4 : の用語のほか、付録 D : STM32 マイクロコントローラの消費電力パラメータまたはデータシートの電気的特性のセクションを参照してください。

選択できる値が 1 つのみのパラメータには、自動的にその値が設定されます（このパラメータは変更できず、灰色で表示されます）。ここでは、選択したマイクロコントローラに関連する選択肢のみが提示されます。

新しいステップを設定するには、次の手順を実行します。

1. ［**Add**］または［**Duplicate**］をクリックして［**New step**］ウィンドウを開くか、［Sequence Table］でステップをダブルクリックして［**Edit step**］ウィンドウを開きます。
2. 開いているステップのウィンドウで、次の順序で選択します。
   - ［**Power Mode**］

     ［Power Mode］を変更すると、ステップ設定全体がリセットされます。
   - ［**Peripherals**］

     ［Power Mode］を設定すると、いつでもペリフェラルを選択または選択解除できます。
   - ［**Power scale**］

     この機能は、STM32L1 では［Power Consumption Range］、STM32F4 では［Power Scale］です。

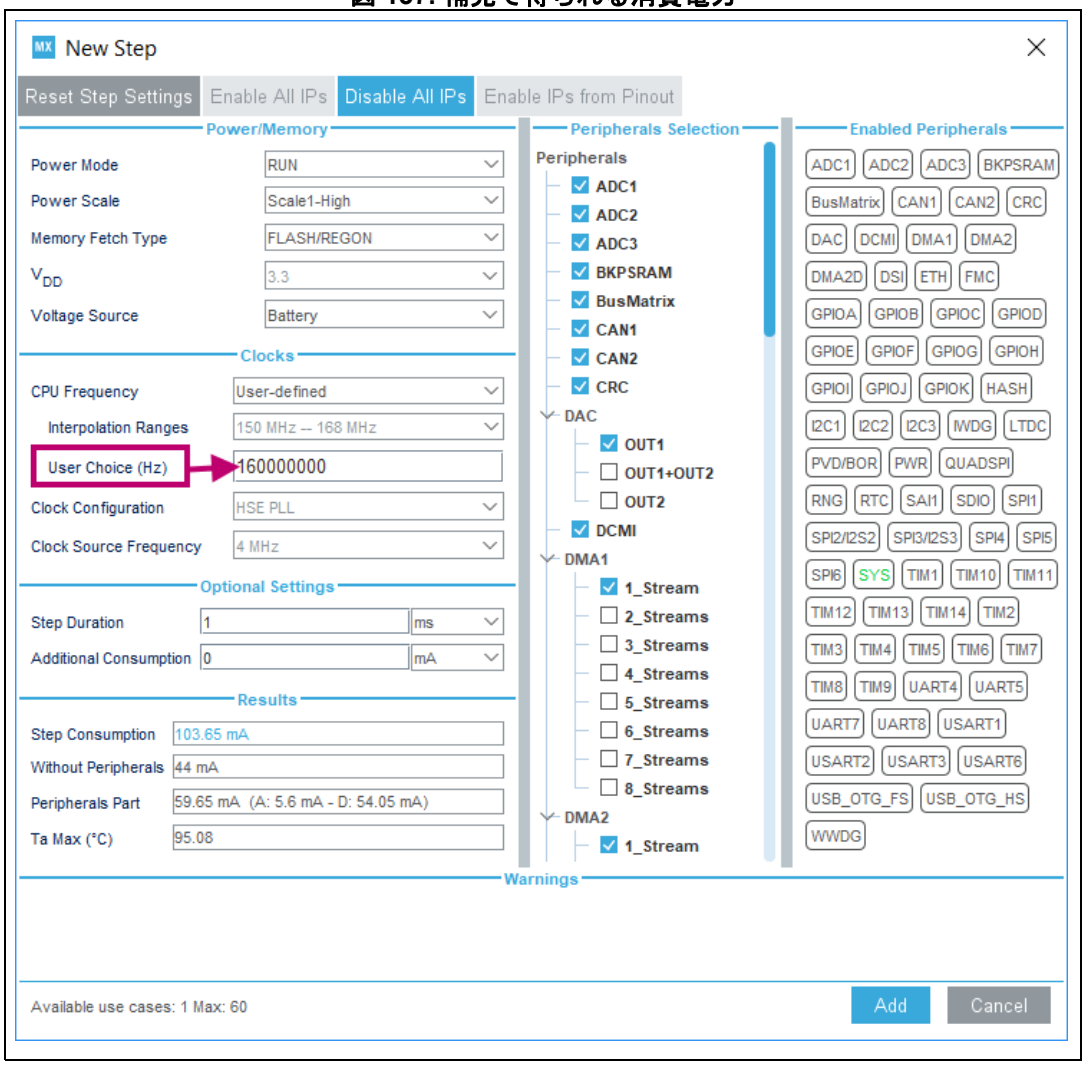
     ［Power Mode］または［Power Consumption Range］を変更すると、その下にある設定がすべて破棄されます。
   - ［**Memory Fetch Type**］
   - ［$V_{DD}$］：複数の選択肢がある場合に使用できます。
   - ［Voltage Source］：バッテリまたは VBUS を選択します。
   - ［**Clock Configuration**］

     ［Clock Configuration］を変更すると、その下にある周波数の選択がリセットされます。
   - 複数の選択が可能な場合は［**CPU Frequency**］（STM32F4）および［**AHB Bus Frequency/ CPU Frequency**］（STM32L1）、アクティブ･モードの場合はユーザ指定の周波数などが選択できます。この場合、消費電力値は補完によって得られます（補完の使用を参照）。
3. オプション設定
   - ［**Step Duration**］：デフォルト値は 1 ミリ秒
   - ［**Additional Consumption**］：この値（mA 単位）には、たとえば、アプリケーションで使用する外部コンポーネント（外部レギュレータ、外部プルアップ、LED、他のディスプレイなど）によって発生する値を指定します。マイクロコントローラの消費電力にこの値を加算した値で、ステップ全体の消費電力が決まります。
4. 設定が完了すると［**Add**］ボタンがアクティブになります。このボタンをクリックするとステップが作成され、［Sequence Table］に追加されます。

### 補完の使用

アクティブ·モード（RUN モード、SLEEP モード）用に設定したステップでは、[CPU Frequency] で［User Defined］を選択して周波数（Hz 単位）を入力することによって、周波数の補完を使用できるようになります（図 137 を参照）。

**図 137. 補完で得られる消費電力**

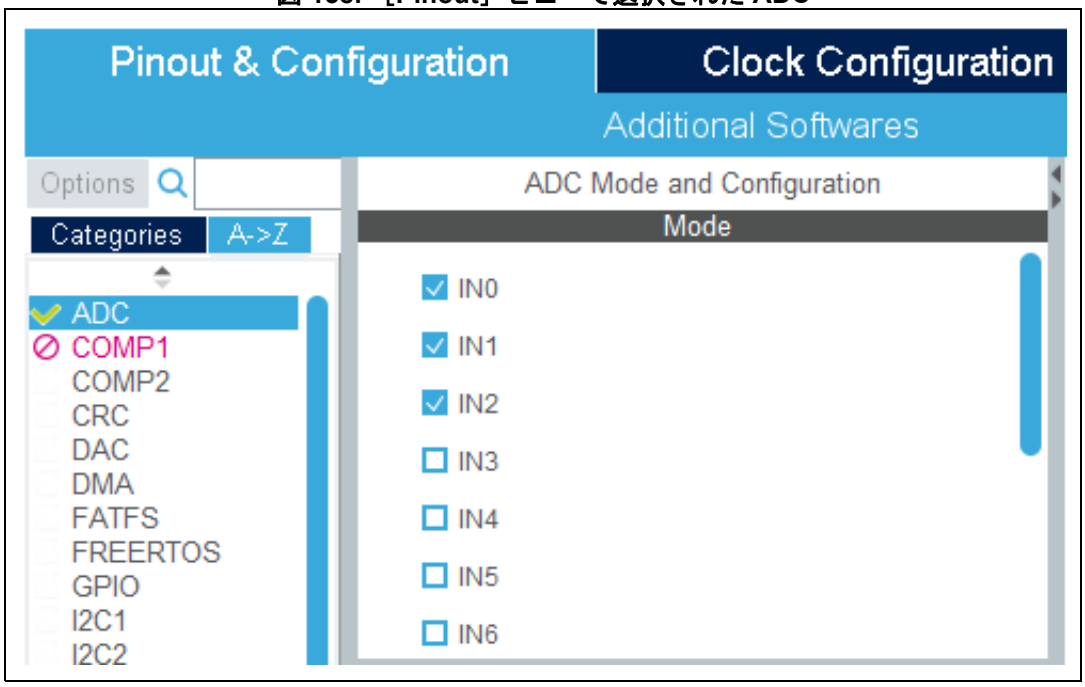


### ピン配置のインポート

図 138 は、[Pinout] ビューで ADC を設定している例を示しています。[Power Consumption Calculator] ビューで [Enable IPs from Pinout] をクリックすると、ADC ペリフェラルと GPIO A が選択されます（図 139 参照）。

[Enable IPs from Pinout] ボタンを使用すると、[Pinout] ビューで設定したペリフェラルを自動的に選択できます。

**図 138. [Pinout] ビューで選択された ADC**

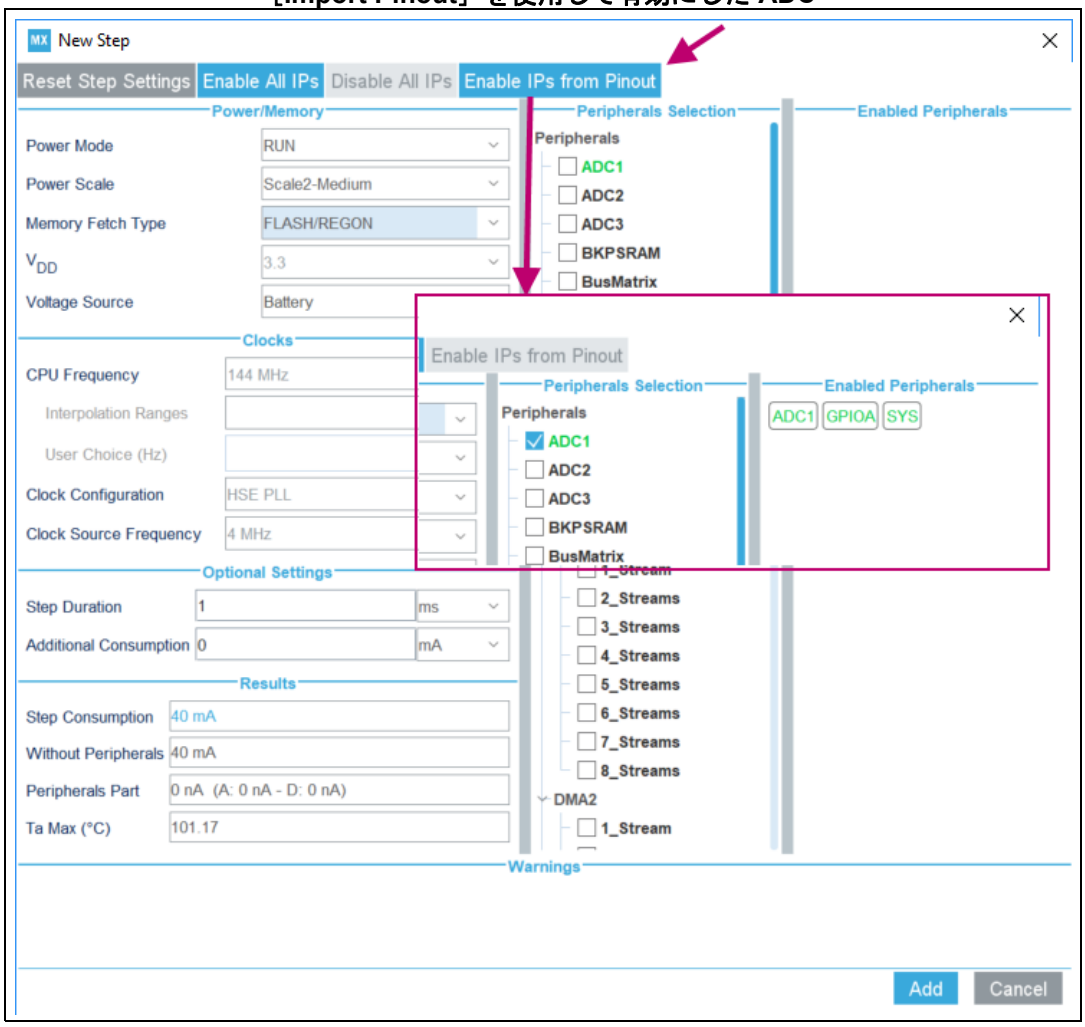


### すべてのペリフェラルの選択または選択解除

［Enable All IPs］をクリックすると、すべてのペリフェラルを一括して選択できます

［Disable All IPs］をクリックすると、すべてのペリフェラルが消費電力への算入対象から除外されます。

**図 139. ［Power Consumption Calculator］のステップ設定ウィンドウ：［Import Pinout］を使用して有効にした ADC**

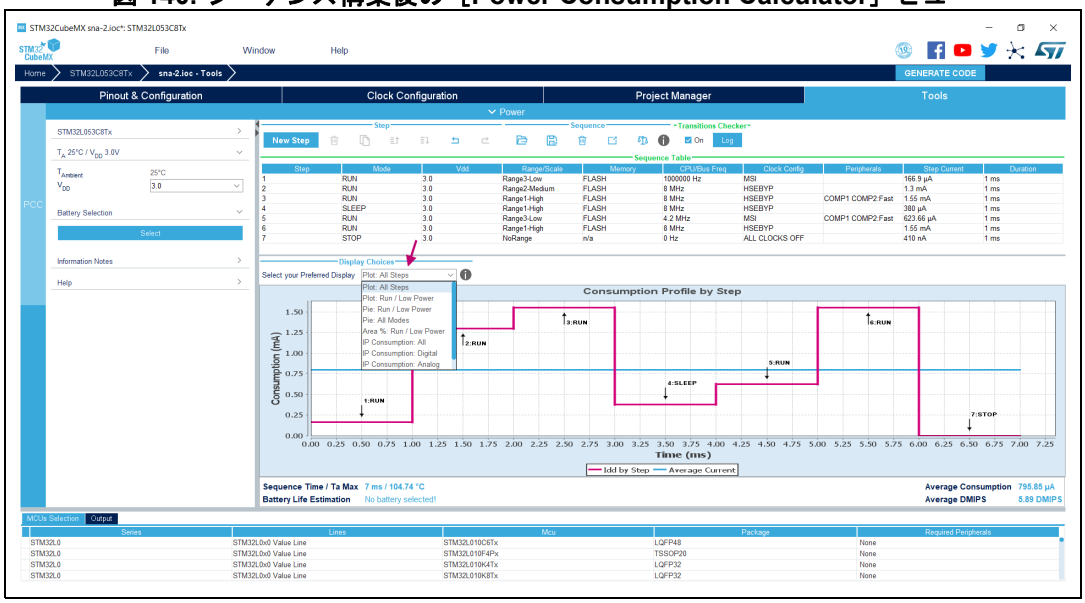


### 5.2.3 ユーザ定義消費電力シーケンスの管理と結果の確認

消費電力シーケンスを設定すると、[Power Consumption Calculator] ビューが更新されます（図 140 を参照）。

- [Sequence Table] には、すべてのステップとそのパラメータ値が表示されます。[Category] 列には、消費電力値がデータシートから取得されているか、補完で得られているのかが示されます。
- シーケンス・チャート・エリアには、表示タイプ（すべてのステップのプロット、省電力モードと RUN モードの比較プロットなど）に応じて消費電力シーケンスのさまざまなビューが表示されます。
- [Results Summary] には、シーケンス合計時間、最大周辺温度（$T_{AMAX}$）、平均消費電力推定値、DMIPS などが表示されるほか、有効なバッテリ設定を選択している場合はバッテリ寿命も表示されます。

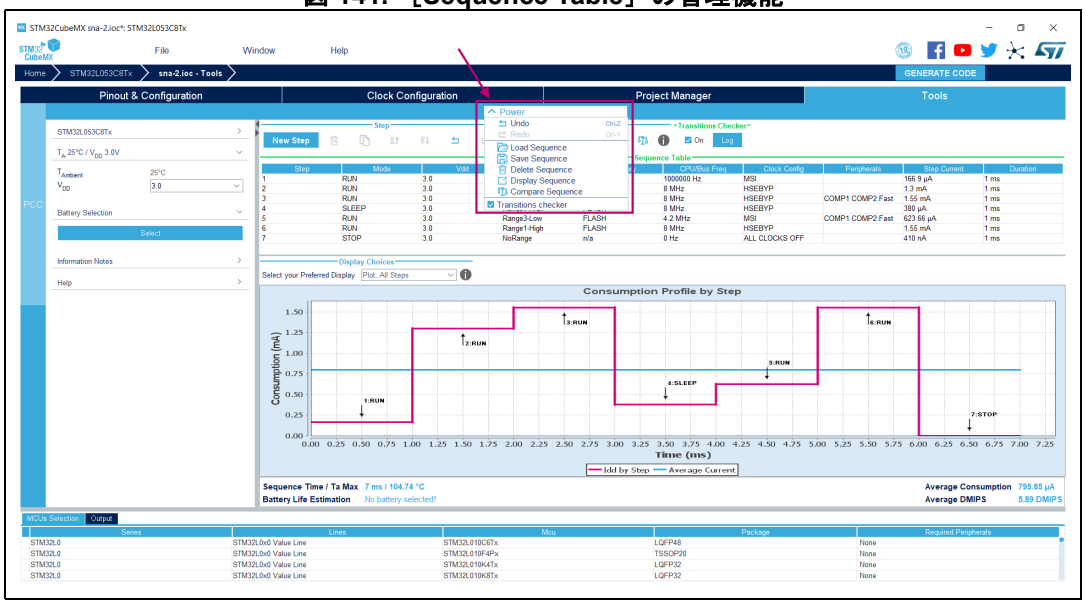
**図 140. シーケンス構築後の［Power Consumption Calculator］ビュー**



### シーケンス全体の管理（ロード、保存、および比較）

電源メニュー（図 141 参照）では、現在のシーケンスの保存と削除、または以前に保存したシーケンスとの比較が可能です。以前に保存したシーケンスは専用のポップアップ·ウィンドウに表示されます。

**図 141. ［Sequence Table］の管理機能**

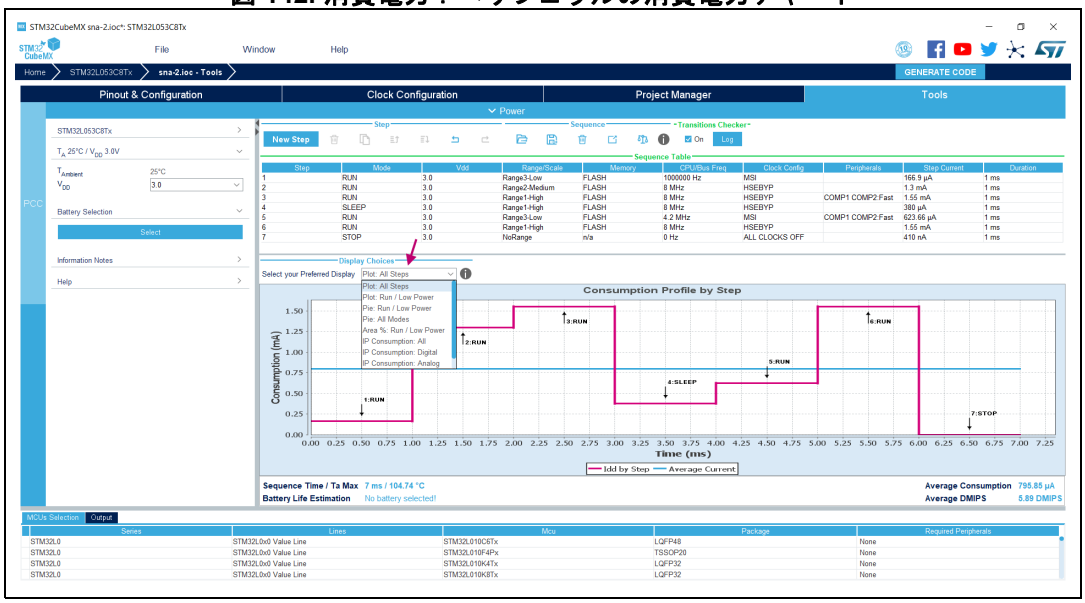


### 結果チャートと表示オプションの管理

表示エリアで、表示するチャートのタイプ（シーケンス·ステップ、円グラフ、ペリフェラルごとの消費電力など）を選択します。**[External Display]** をクリックして、専用のウィンドウにチャートを表示することもできます（図 142 を参照）。

チャートを右クリックするとコンテキスト·メニューが開きます。**[Properties]**、**[Copy]**、**[Save]**（png 画像ファイルとして保存）、**[Print]**、**[Zoom]** の各メニュー項目があるほか、**[Auto Range]** を選択すると、ズーム操作を実行する前のビューに表示がリセットされます。チャートの中をクリックして左から右へドラッグし、領域を指定することでも**ズーム**が可能です。チャートをクリックして左にドラッグすると**ズームがリセット**されます。

**図 142. 消費電力：ペリフェラルの消費電力チャート**

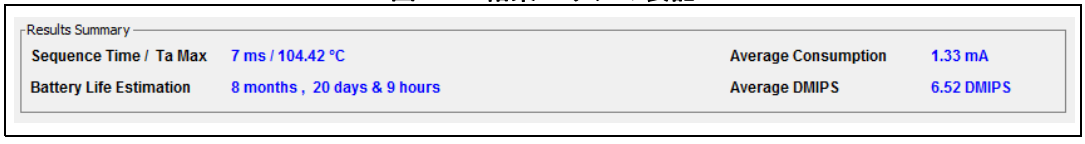



### [Results Summary] エリアの概要

このエリアには、次の情報が表示されます（図 143 を参照）。

- シーケンスの各ステップ所要時間を合算したシーケンス合計時間
- 各ステップの消費電力をステップ所要時間で重み付けした値を合計した平均消費電力
- Dhrystone ベンチマークに基づく平均 DMIPS （Dhrystone Million Instructions per Second）。定義されたシーケンスの CPU パフォーマンスをハイライトします。
- 選択したバッテリ·モデルの平均消費電力とバッテリの自己放電に基づくバッテリ寿命推定値
- $T_{AMAX}$：シーケンス中に発生した最大周辺温度値

**図 143. 結果エリアの表記**



### 5.2.4 消費電力シーケンス·ステップ·パラメータの用語

消費電力シーケンス·ステップの特性を記述するパラメータの用語を次に示します（付録 D：STM32 マイクロコントローラの消費電力パラメータを参照）。

- Power Mode

  消費電力を低減するには、マイクロコントローラの動作モードを実行モードから省電力モードに切り替えることをお勧めします。実行モードでは最大電力が必要ですが、省電力モードは限られたリソースで動作できます。

- $V_{CORE}$ 範囲（STM32L1 の場合）または Power Scale（STM32F4 の場合）

  これらのパラメータをソフトウェアで設定して、デジタル·ペリフェラルの電源範囲を制御します。

- Memory Fetch Type

  このフィールドでは、アプリケーションの C コード実行可能なメモリの位置が示されます。［RAM］、［Flash］、［Flash with ART ON］、または［Flash with ART OFF］のいずれかを選択できます。［Flash with ART ON］と［Flash with ART OFF］は、Flash·メモリから実行するプログラムの実行速度を高速化する独自仕様の適応型リアルタイム（ART）メモリ·アクセラレータを備えたファミリでのみ選択できます。

  ART アクセラレータで得られる性能は、ウェイト·ステートを発生せずに Flash メモリからプログラムを実行する場合と同等です。消費電力の観点では、RAM からプログラムを実行する場合と同等です。STM32CubeMX では、同じ選択肢の中で［RAM］と［Flash memory with ART ON］の両方の設定を用意しています。

- ［Clock Configuration］

  この操作は、マイクロコントローラの消費電力の計算に使用する AHB バス周波数または CPU 周波数を設定します。選択肢が 1 つのみの場合は、自動的にその周波数が設定されます。

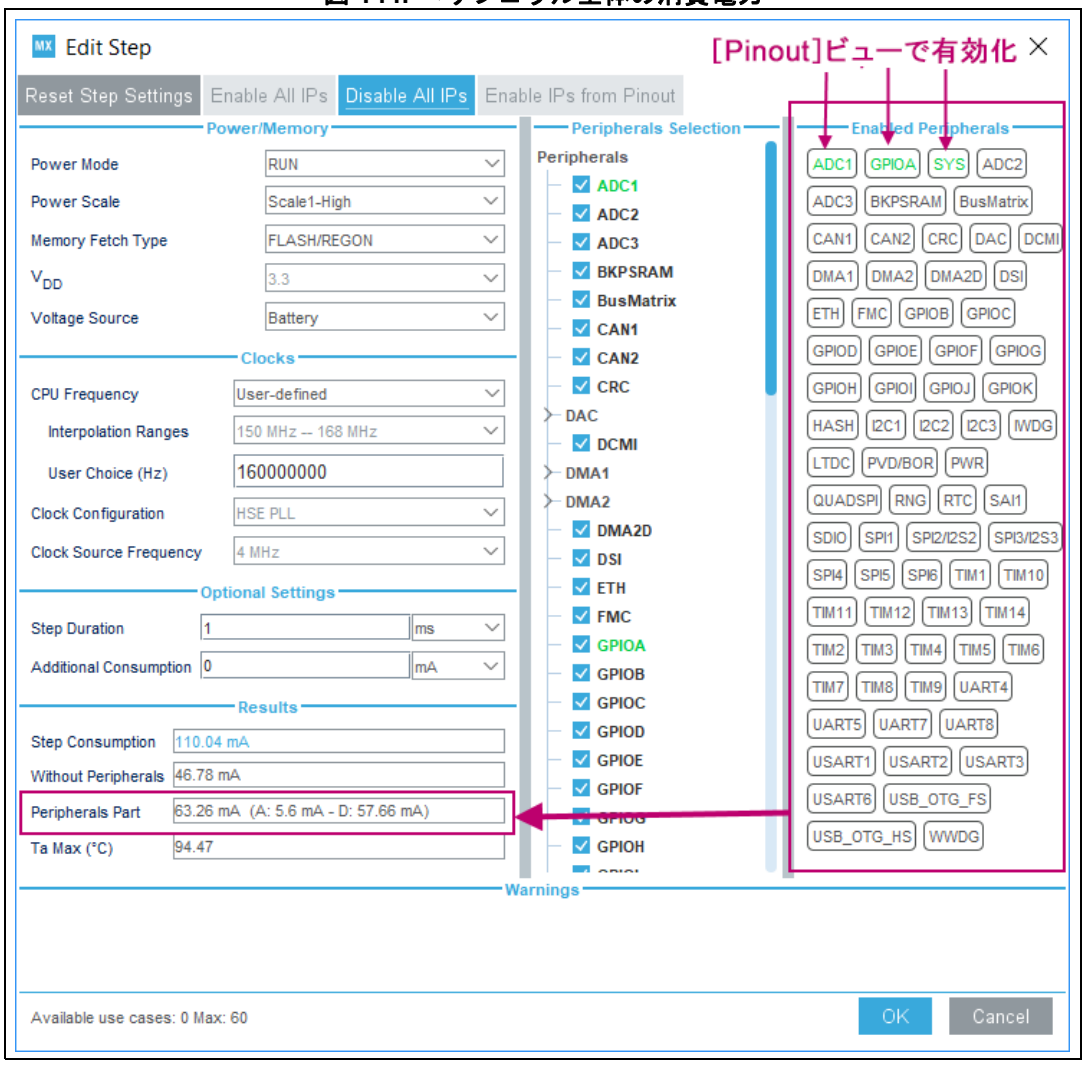
  ［Clock Configuration］ドロップダウン·リストでは、次のアプリケーション·クロックを設定できます。

  – 内部または外部のオシレータ·ソース：MSI、HSI、LSI、HSE、または LSE
  – オシレータ周波数
  – PLL のオン、LSE のバイパス、AHB のプリスケーラ値、デューティを指定した LCD などのその他の決定要因となるパラメータ

- ペリフェラル

  ペリフェラル·リストには、選択した電力モードで使用できるペリフェラルが表示されます。消費電力は、ペリフェラルがクロック動作のみの状態（実行中のプログラムで使用されていない状態など）であることを前提として求められます。ペリフェラルは、それぞれで有効または無効にすることができます。ペリフェラルの個々の消費電力は、ツールヒントに表示されます。ペリフェラルのアナログ部品とデジタル部品による合計消費電力は、ステップの［Results］エリアに表示されます（図 144 を参照）。

図 144. ペリフェラル全体の消費電力




次のように、アプリケーションに関連するペリフェラルを選択できます。

– 選択しない（**[Disable All]**）
– 一部を選択（ペリフェラルごとのチェックボックスを使用）
– すべて選択（**[Activate All]**）
– 以前に定義したピン割当て設定のペリフェラルをすべて使用（**[Import Pinout]**）

消費電力を計算する際は、選択した有効なペリフェラルのみが考慮されます。

- Step Duration

ステップ所要時間のデフォルト値を変更できます。シーケンスを構築する方法として、アプリケーションの実際の消費電力シーケンスに従ってステップを作成する方法と、モードごとの所要時間が全体に占める比率を定義してステップを作成する方法の 2 種類があります。たとえば、アプリケーションがモードごとに必要とする時間の比率が RUN モードで 30%、SLEEP モードで 20%、STOP モードで 50% の場合、30 ミリ秒の RUN モード、20 ミリ秒の SLEEP モード、50 ミリ秒の STOP モードの 3 つのステップで構成するシーケンスを設定する必要があります。

- Additional Consumption

  このフィールドには、ユーザ固有の設定によって別途発生する消費電力を入力できます。たとえば、マイクロコントローラに接続した他のデバイスに、そのマイクロコントローラから電源を供給する場合が考えられます。

### 5.2.5 バッテリの用語

- 容量（mAh）

  1 回のバッテリ放電で供給可能なエネルギー量。

- 自己放電（%/月）

  バッテリを指定期間の間使用しない状態（開路状態）で、内部リークによって失われるバッテリ容量。

- 公称電圧（V）

  完全充電したバッテリから供給される電圧。

- 最大連続電流（mA）

  バッテリ寿命の間にバッテリを損傷せずに供給できる最大電流。

- 最大パルス電流（mA）

  例外的に供給できる最大パルス電流。たとえば、起動時にアプリケーションをオンに切り替えたときに流れる電流が考えられます。

### 5.2.6 SMPS 機能

一部のマイクロコントローラ（STM32L496xxxxP など）は、外部スイッチング電源（SMPS）に接続して、消費電力をさらに低減できます。

このようなマイクロコントローラ向けに、消費電力計算ツールには次の機能が用意されています。

- 現在のプロジェクトで SMPS を使用することの選択：

  左側パネルで**［Use SMPS］**ボックスをチェックします（図 145 を参照）。デフォルトでは、ST SMPS モデルを使用します。

- 別の SMPS モデルを選択するには**［Change］**ボタンをクリックします。

  別の SMPS モデルを追加するための［SMPS Database Management］ウィンドウが開きます（図 146 を参照）。現在のシーケンスで使用する別の SMPS モデルを選択できます（図 147、図 148、および図 149 を参照）。

- ［SMPS Checker］を有効にして、現在のシーケンスに無効な SMPS トランジションがあるかどうかを確認します。

  この確認を実行するには、チェッカーを有効にするチェックボックスをチェックし、**［Help］**ボタンをクリックして基準の状態図を開きます（図 150 を参照）。

- ステップごとに SMPS モードを設定します（図 151 参照）。

  ［SMPS Checker］が有効な場合、現在のステップで有効な SMPS モードのみが提案されます。

図 145. 現在のプロジェクトで使用する SMPS の選択

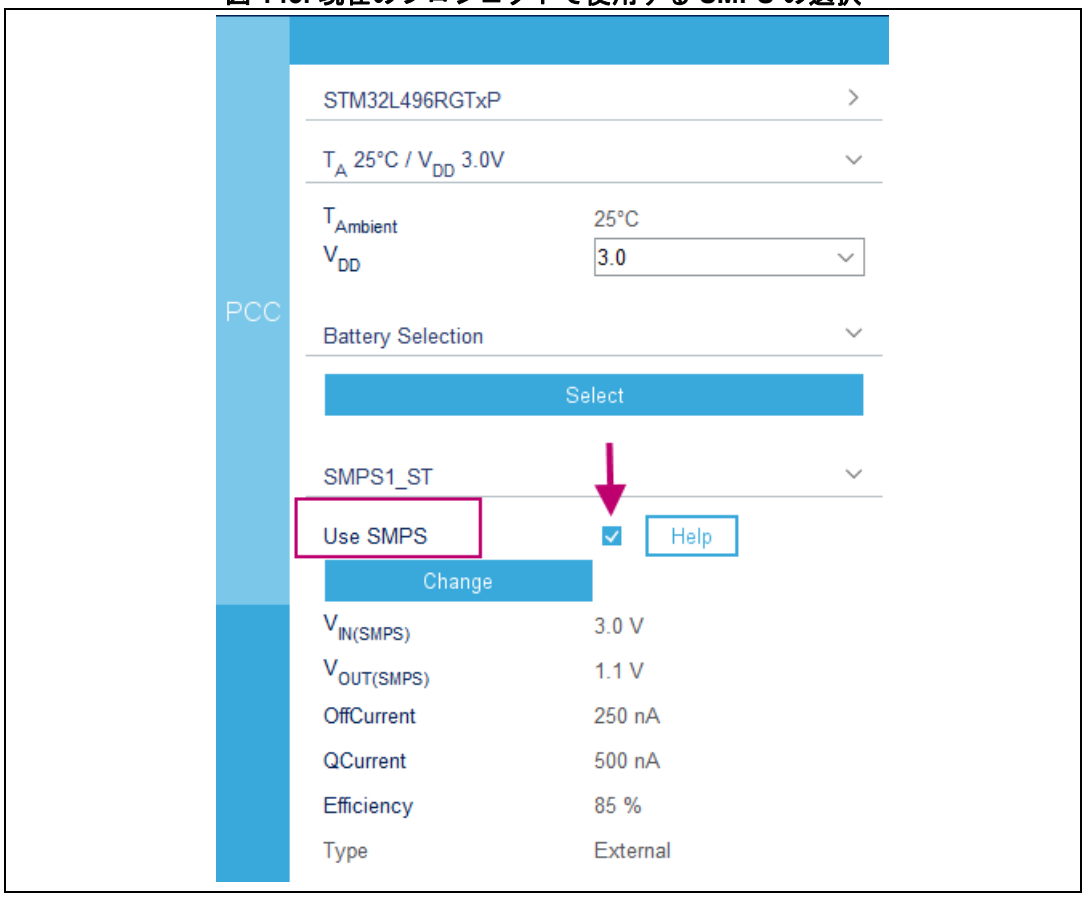


図 146. SMPS データベース - 新しい SMPS モデルの追加

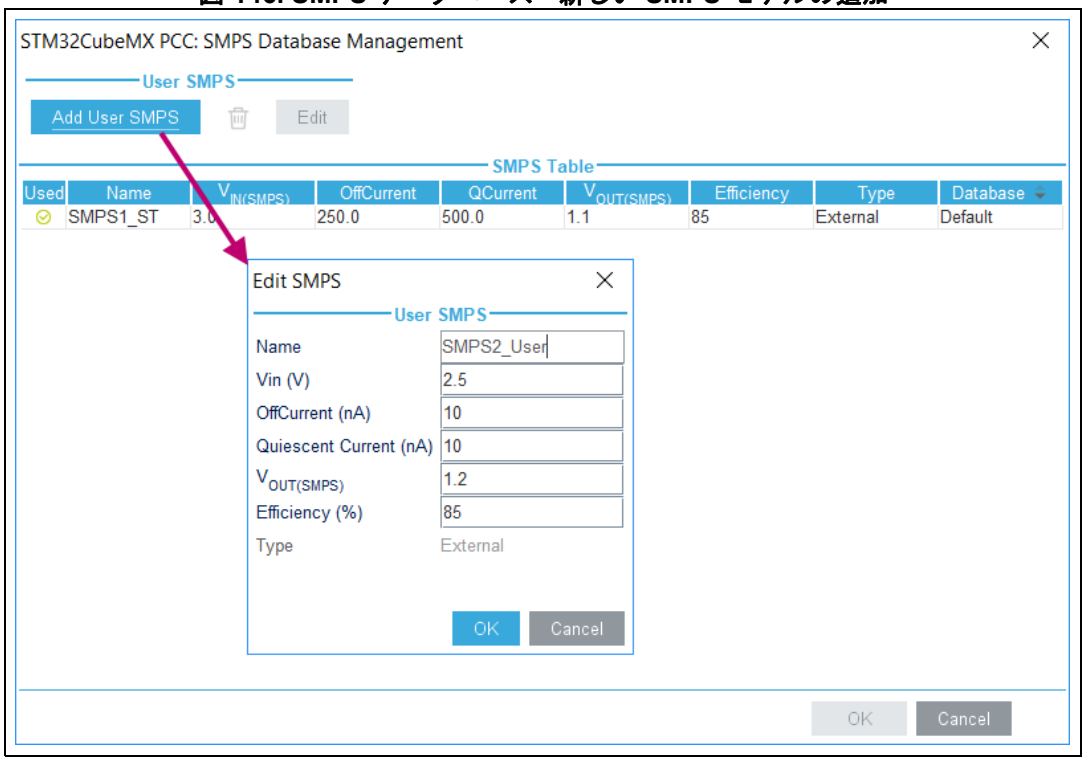



図 147. SMPS データベース - 別の SMPS モデルの選択

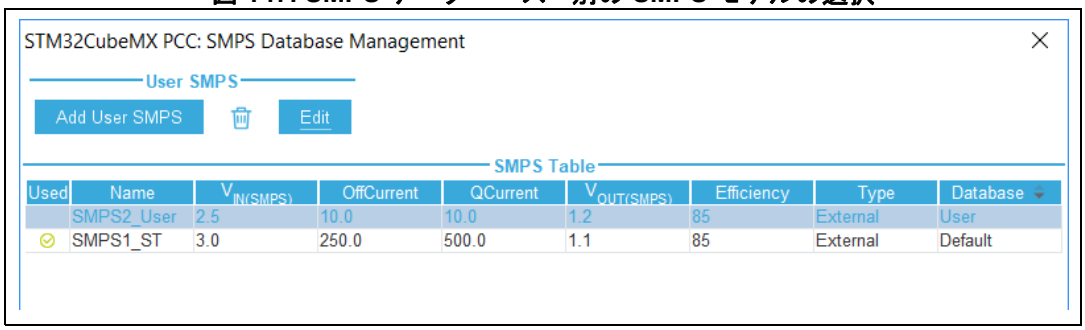


図 148. 新しい SMPS モデルで更新した現在のプロジェクト設定

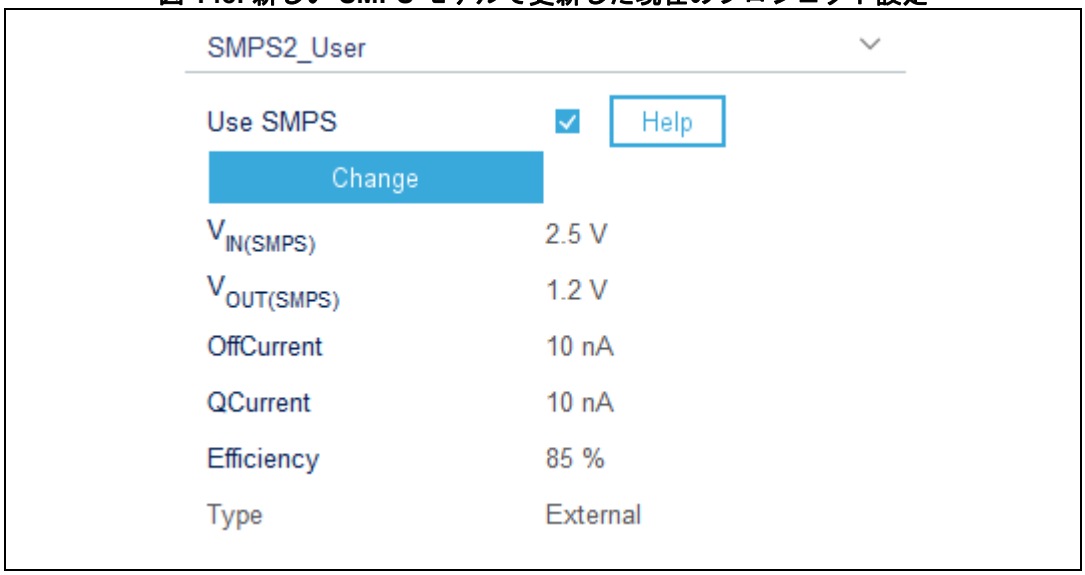



図 149. 新しいモデルを選択した［SMPS Database Management］ウィンドウ

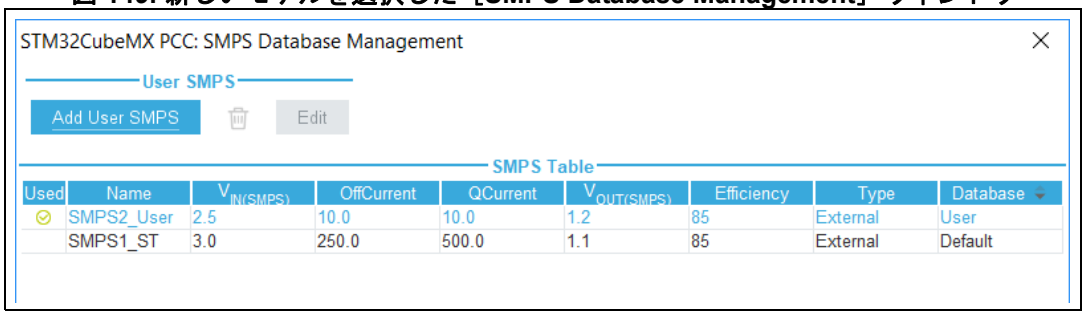


図 150. SMPS トランジション･チェッカーと状態図のヘルプ･ウィンドウ

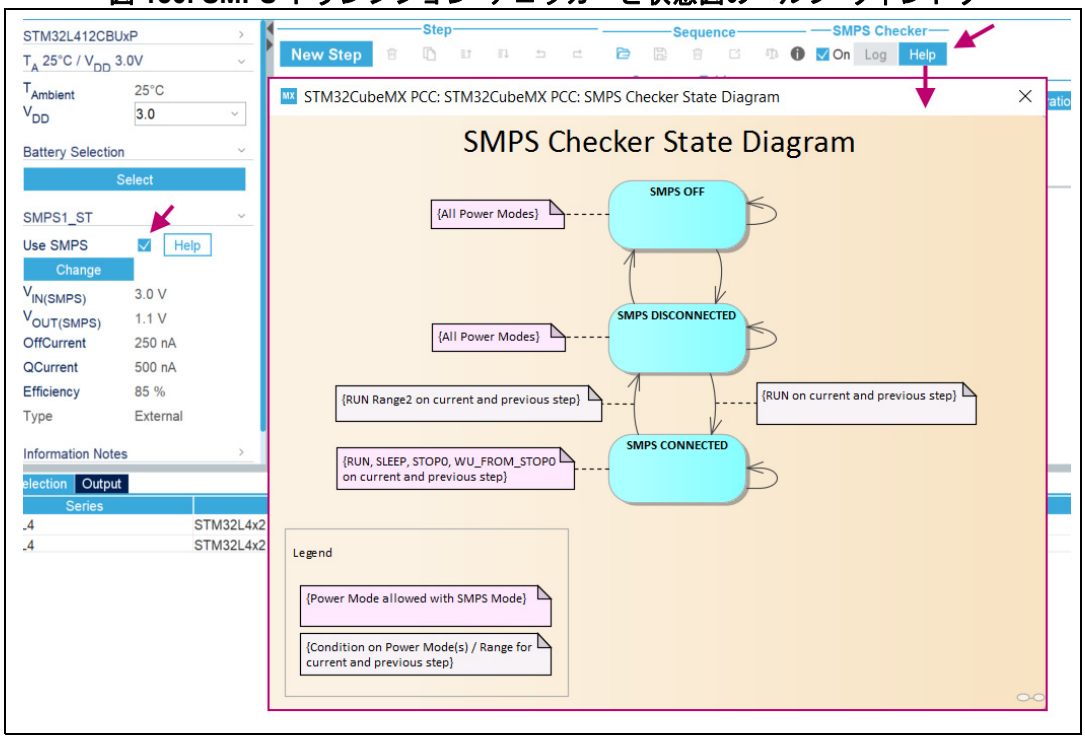
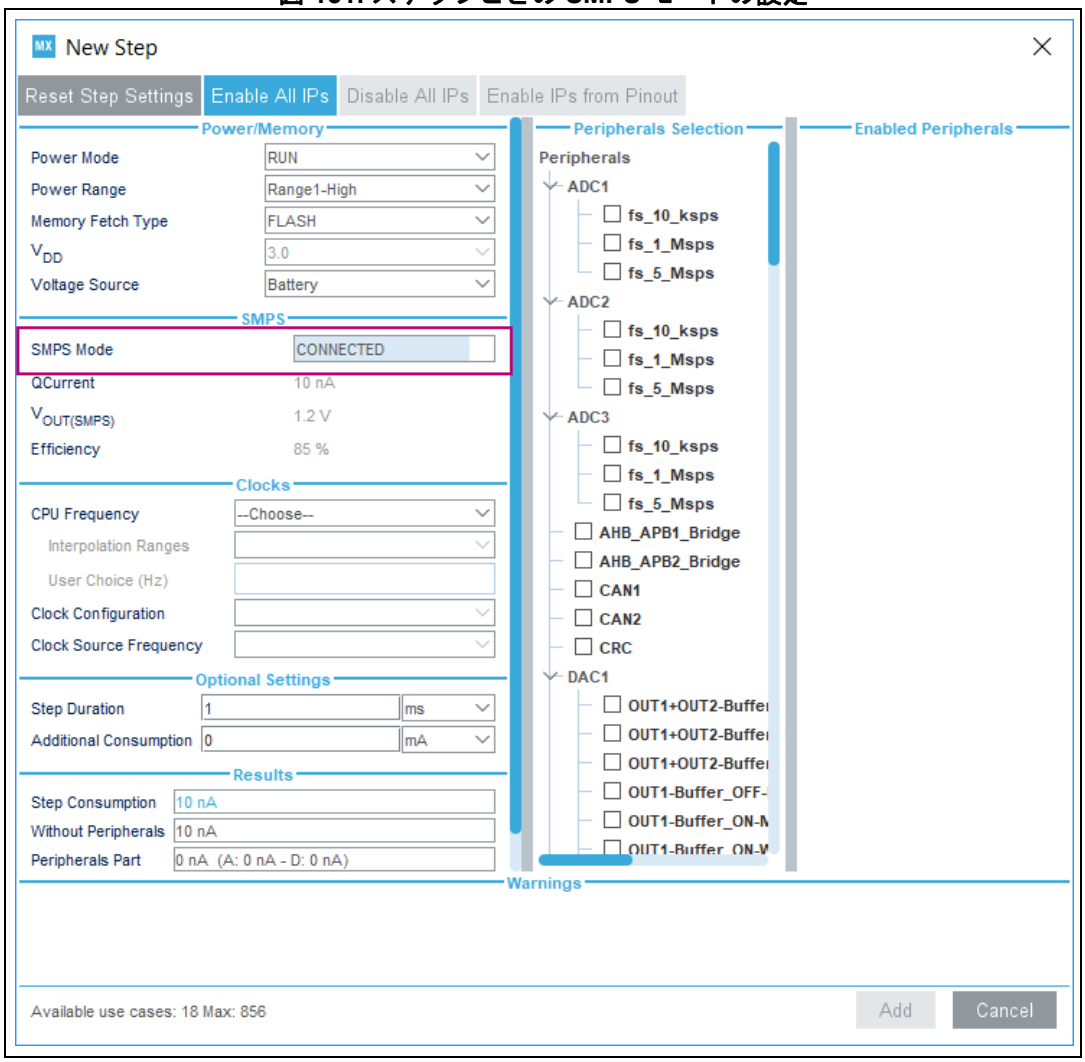


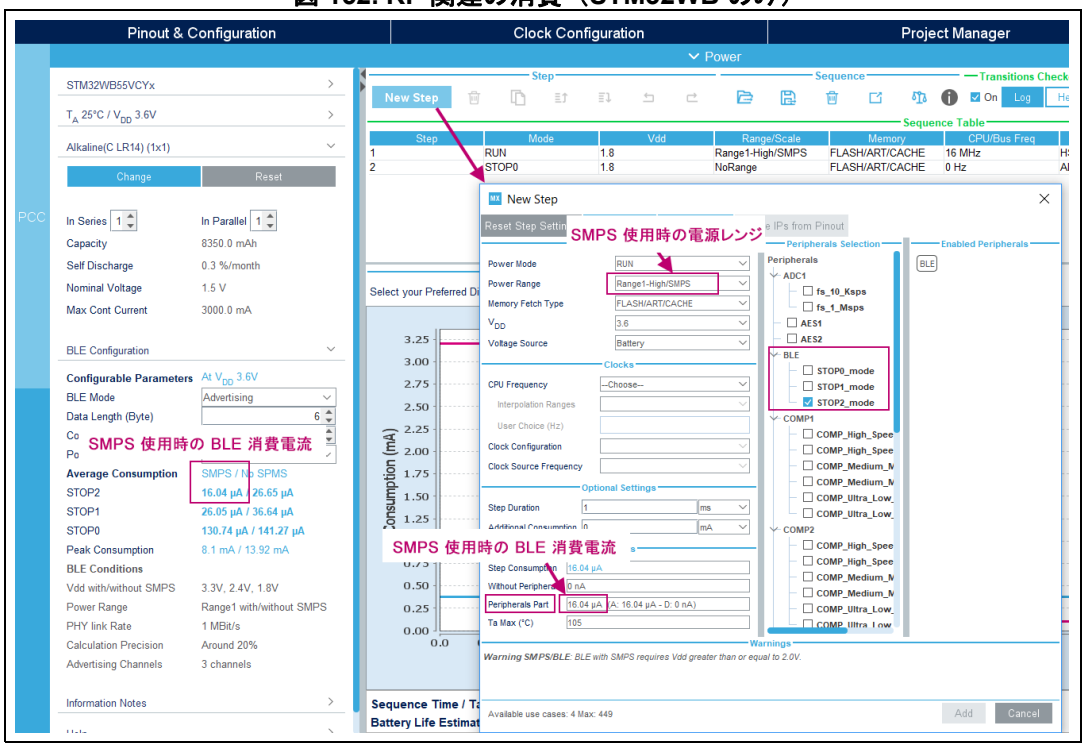
図 151. ステップごとの SMPS モードの設定



### 5.2.7 BLE および ZigBee のサポート（STM32WB シリーズのみ）

消費電力ツールでは、RF ペリフェラルと対応する BLE 機能モードに伴う消費電力を、SMPS 機能の使用と組み合わせて考慮できます。

**図 152. RF 関連の消費（STM32WB のみ）**




BLE モードは左側パネルで選択して、ユーザのアプリケーションに関連する設定を反映するように設定できます。BLE が有効化される新しいステップを追加するごとに、ペリフェラル消費電力の部分が適宜更新されます（図 153 参照）。ZigBee でも、同様の手順を使用します（図 154 参照）。

図 153. RF BLE モードの設定（STM32WB シリーズのみ）

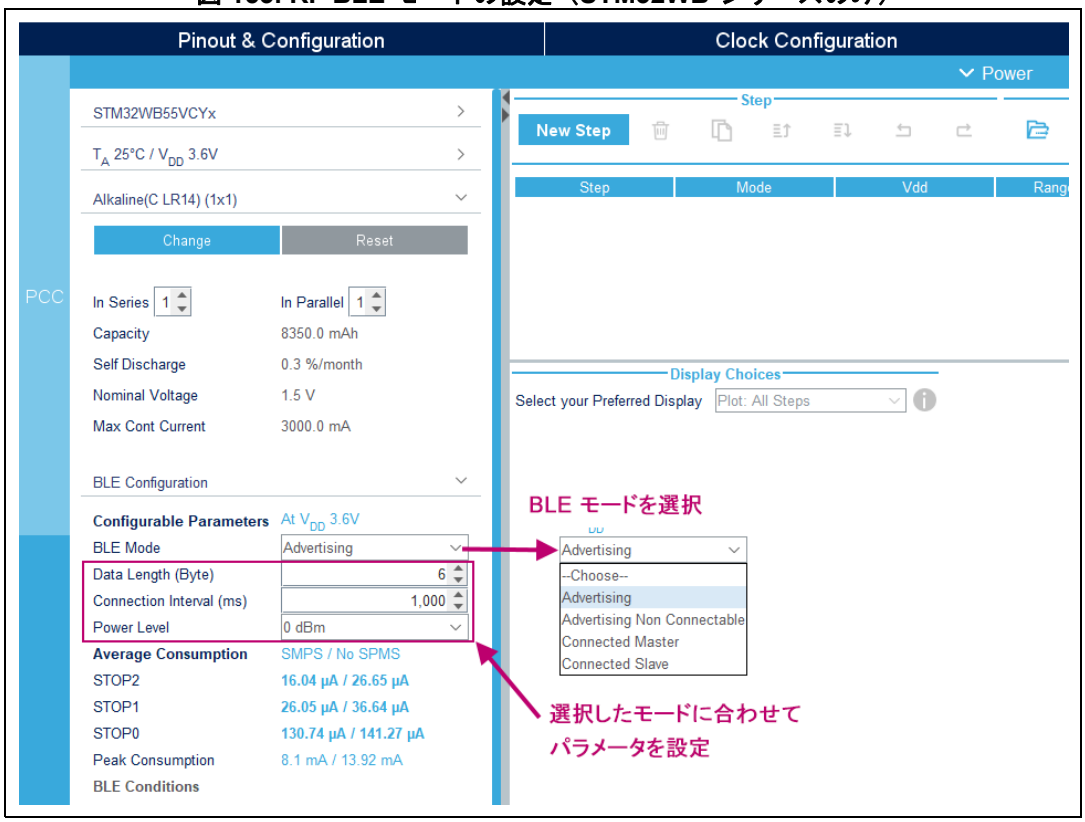
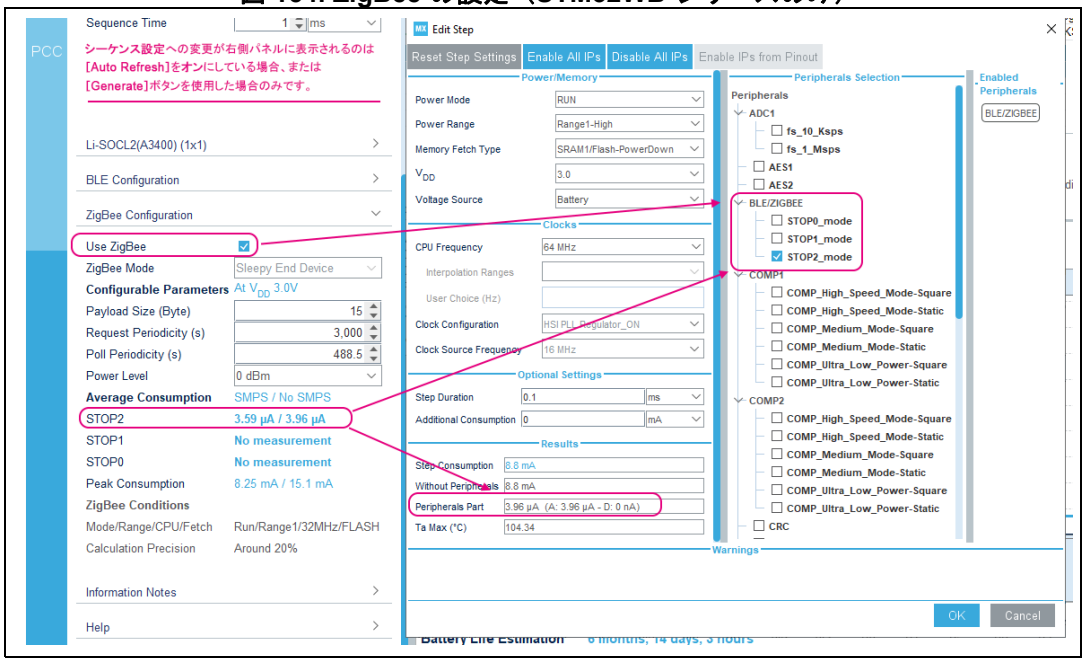


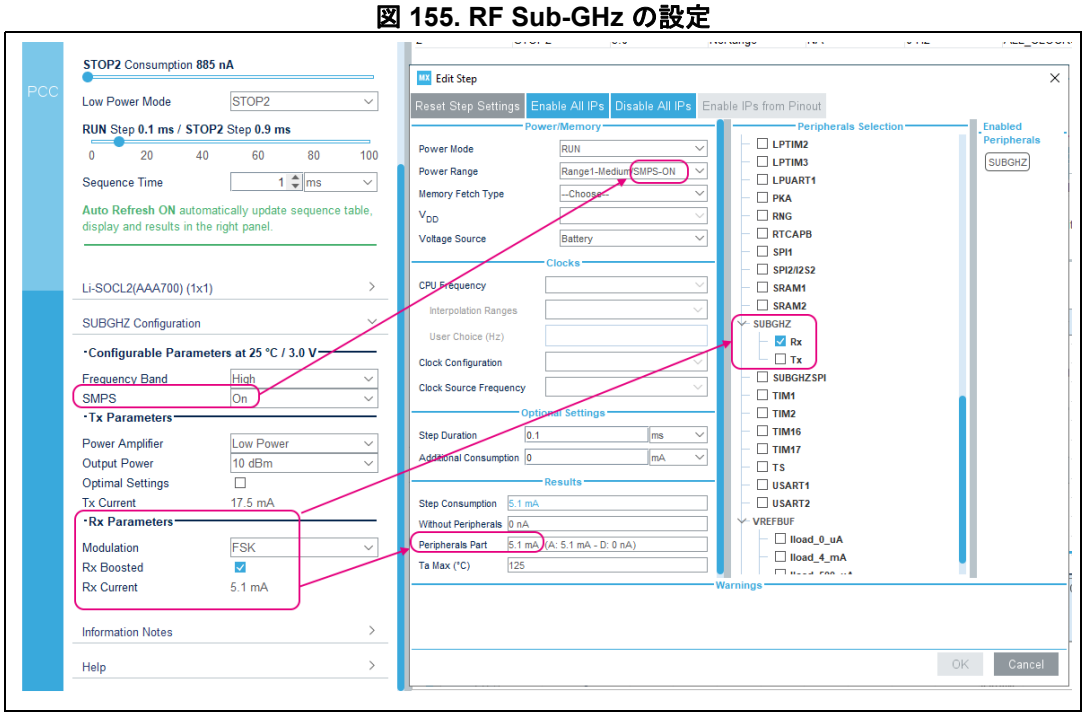

図 154. ZigBee の設定（STM32WB シリーズのみ）



### 5.2.8 Sub-GHz のサポート（STM32WL シリーズのみ）

Sub-GHz の使用は左側パネルで有効化し、ユーザのアプリケーションに関連する設定を反映するように設定できます。ZigBee を有効にするステップを追加するたびに、ペリフェラル消費電力の部分が適宜更新されます（図 155 参照）。

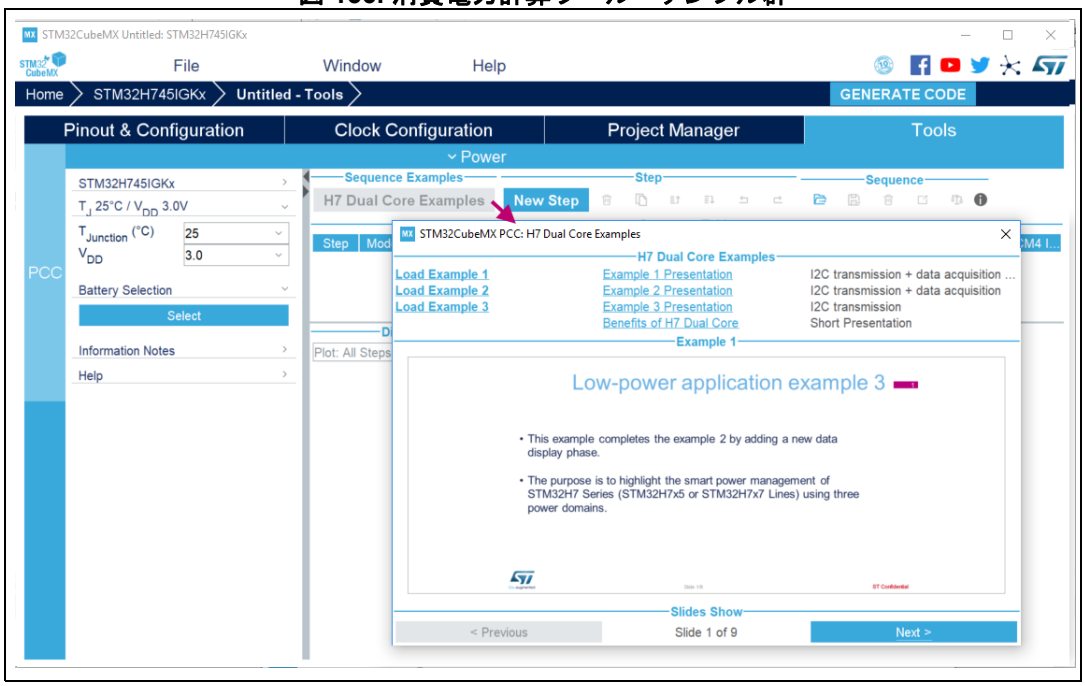
**図 155. RF Sub-GHz の設定**




### 5.2.9 サンプル機能（STM32MP1 と STM32H7 デュアルコアのみ）

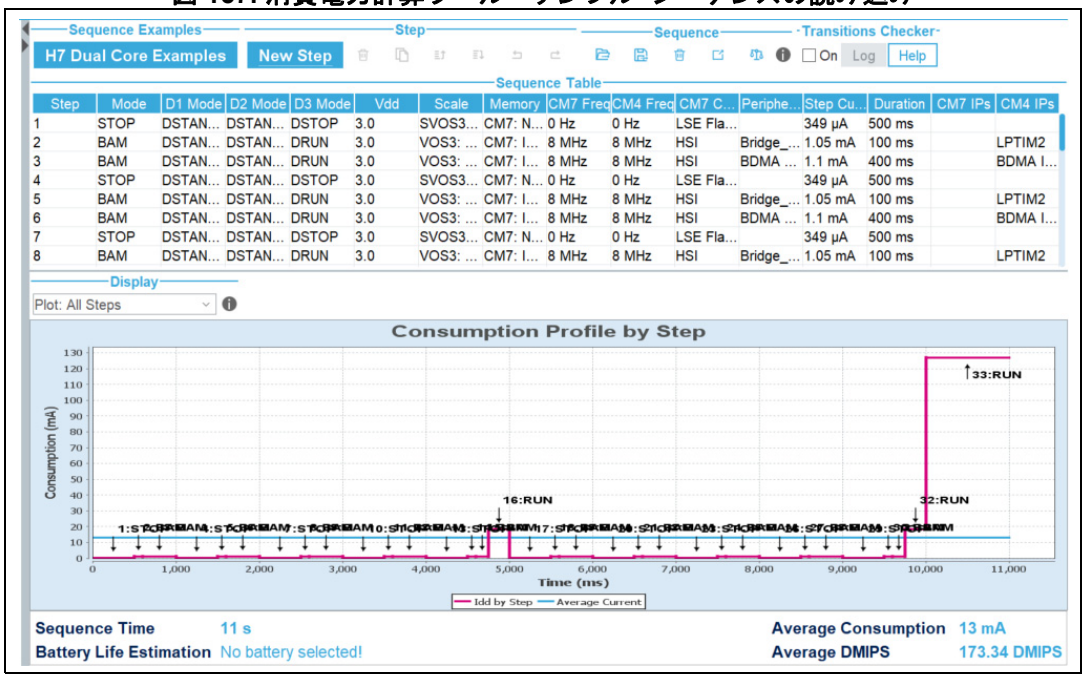
［Sequence Examples］セクションを使用して PCC ツールからサンプル･シーケンスにアクセスできます。各サンプルには、説明のスライドと PCC に読み込める既製のシーケンスが含まれます（図 156 参照）。

図 156. 消費電力計算ツール - サンプル群




[Load Example N] をクリックすると、対応する番号 N のサンプル·シーケンスが読み込まれます（図 157 参照）

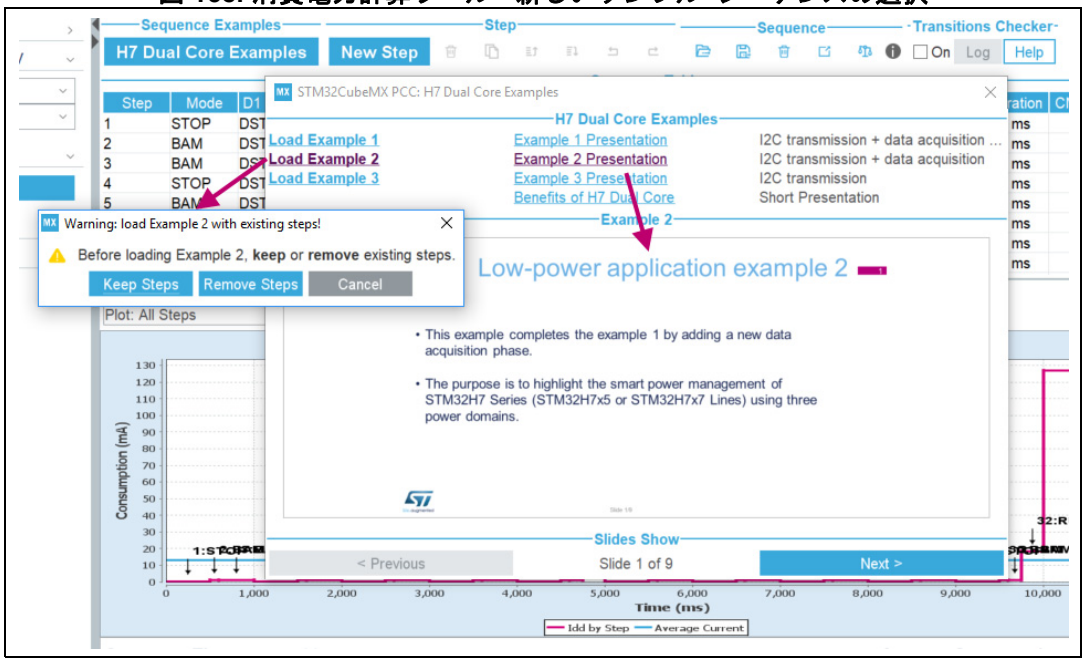
図 157. 消費電力計算ツール - サンプル·シーケンスの読み込み




[Example N Presentation] をクリックすると、そのサンプルに関する説明が表示されます。

サンプルは、いつでも変更できます。新しいシーケンスは現在のシーケンスに追加することも、現在のシーケンスを置き換えることも可能です（図 158 参照）。

**図 158. 消費電力計算ツール - 新しいサンプル·シーケンスの選択**




注： **サンプルは特定の品名に対して提供されています。別の品名に対して使用するには調整が必要になる場合があります。また、読み込んだ後に、各ステップを編集して設定を確認することをお勧めします。**

## 5.3 DDR スイート（STM32MP1 シリーズのみ）

DDR SDRAM は入念な PCB 設計が必要な、複雑な高速デバイスです。

STM32MP15 デバイスは、以下の種類の DDR をサポートしています。

- LPDDR2
- LPDDR3
- DDR3/DDR3L

これらは JEDEC 規格（インタフェース、コマンド、タイミング、パッケージ、ボール配置に関する標準化）で仕様が規定されています。

STM32CubeMX は、STM32MP1 の DDR サブシステム向けに包括的なツール·スイートを提供するために機能が拡張されています。DDR スイートの主要な機能は次のとおりです。

- **DDR の設定**：編集可能なパラメータを減らし、それらに基づいて DDR コントローラと PHY レジスタを自動的に管理します。
- **DDR のテスト**：豊富なテスト·リストに基づいて提供されます。基本的なテストから、ストレス·テストまで用意され、ユーザが独自のテストを開発することも可能です。
- **DDR のチューニング**：ボード設計を補完するためのバイト·レーンの遅延を提示します。

**DDR の設定**には、他のペリフェラルと同様に［**Pinout & Configuration**］ビューからアクセスできます。コンポーネント·パネルの DDR をクリックすると［Mode］および［Configuration］パネルが表示されます。

**DDR テスト･スイートのテストとチューニング**の機能には、**[Tools]** ビューからアクセスできます。

DDR テスト・スイートは、次の 2 つの重要な概念に基づいて構築されています。

- DDR コントローラと PHY の設定における重要な入力である **DDR タイミング**
- ボード設計の不完全性を補完する DDR 信号のチューニング

### 5.3.1 DDR 設定

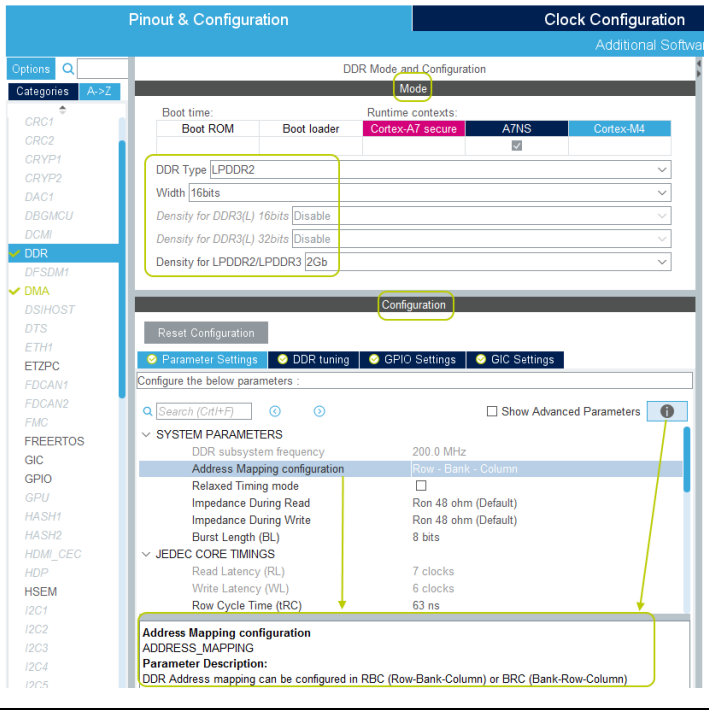
STM32CubeMX では、DDR のシステム･パラメータと JEDEC のコア･タイミングを設定できます。タイミング･パラメータは、DDR のデータシートから入手できます。

#### DDR の種類、幅と密度

DDR の設定手順を進めるには、DDR の種類、幅、密度に関するパラメータの設定が必要です。**[Pinout & Configuration]** ビューで DDR を選択したうえで、[Mode] パネルで設定します。

図 159 の LPDDR2 の設定例を参照してください。

**図 159. DDR のピン配置と構成の設定**




別の例 : "DDR3 16 bits 2 Gb" を 2 チップ使用する構成の設定は "DDR3/DDR3L"、"32 bits"、"4 Gb" になります。

**注 :** **DDR IP のコンテキストは変更できません。DDR は、"Cortex-A7 NS" で識別される "Cortex-A7 非セキュア" コンテキストにツールで固定されています。**

### DDR 設定

パラメータをクリックすると、DDR 設定のフッタ部分にその他の詳細情報が表示されます。

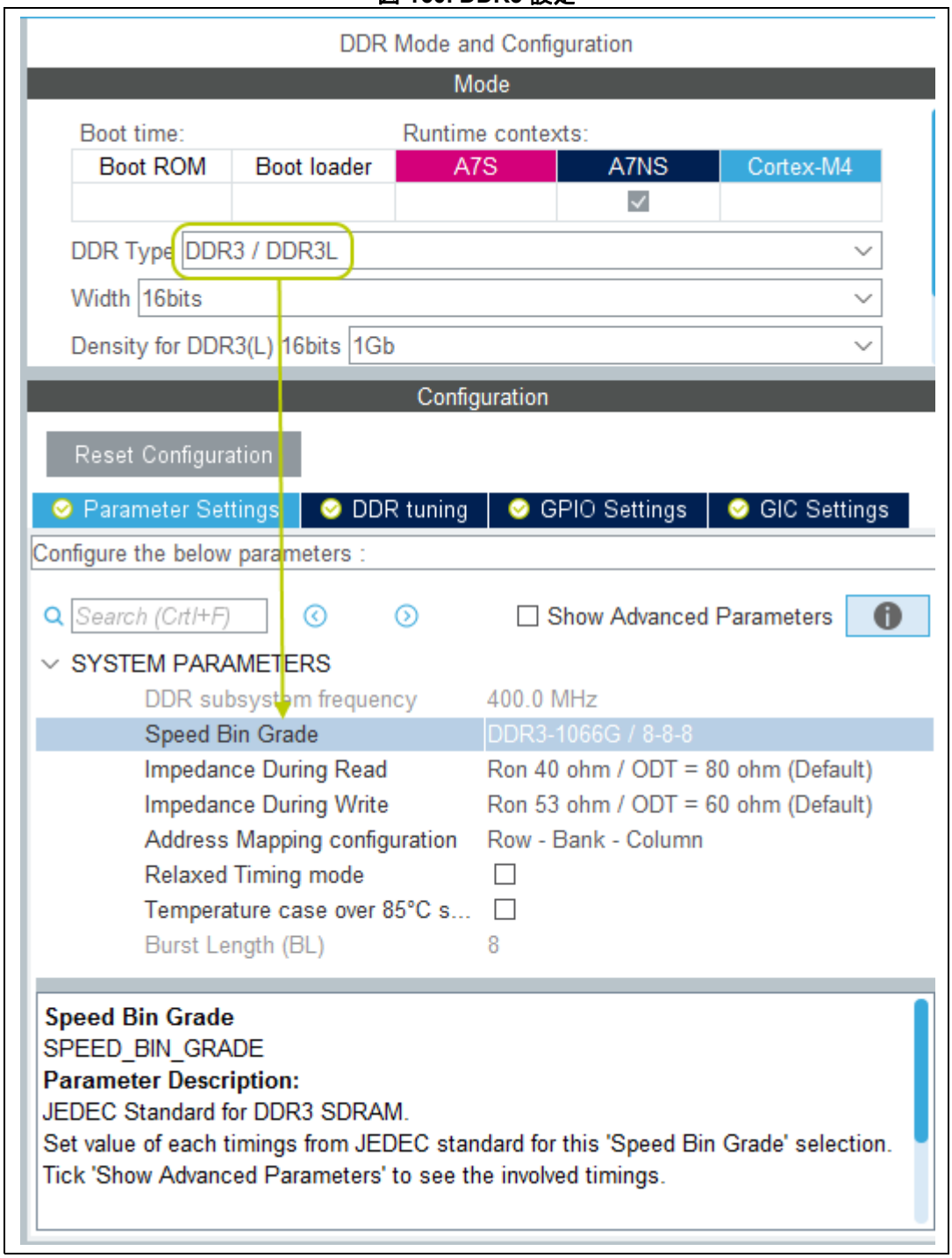
- DDR の周波数は［Clock Configuration］タブの設定から取得されるので、DDR 設定からは変更できません。
- ［Relaxed Timing mode］は、DDR の主要タイミング値の緩和（$t_{RC}$、$t_{RCD}$、$t_{RP}$ のタイミングに $t_{CK}$ の 1 サイクル分を加算）を試みる、開発の初期段階で使用するモードです。
- その他のパラメータは、ユーザの DDR データシートから入手する必要があります。
- 読取り専用のパラメータがあります。これらは、情報の提供のみを目的としており、DDR の種類によって決まります。

［Generate Code］をクリックすると、これらのパラメータに基づいてデバイス･ツリーの DDR ノード（DDR コントローラおよび DDR PHY レジスタの値）が自動的に計算されます。

### DDR3 設定

DDR3 の設定は簡単です。タイミング･パラメータを手動で編集する必要はなく、**[Speed Bin Grade]** の組み合わせを選択するだけで設定できます。

図 160. DDR3 設定




［Speed Bin Grade］の組み合わせは、選択した DDR に適合している必要があります。正確に適合する組み合わせが選択リストに存在しない場合は、より高速な DDR の Speed Bin/Grade が得られるように "1066E / 6-6-6" を指定する必要があります。タイミングを緩和した設定では "1066G / 8-8-8" を指定できます。

この後でタイミングを編集することもできますが、上級ユーザ向けです。使用するパラメータのリストを表示するには［Show Advanced Parameters］を選択します。

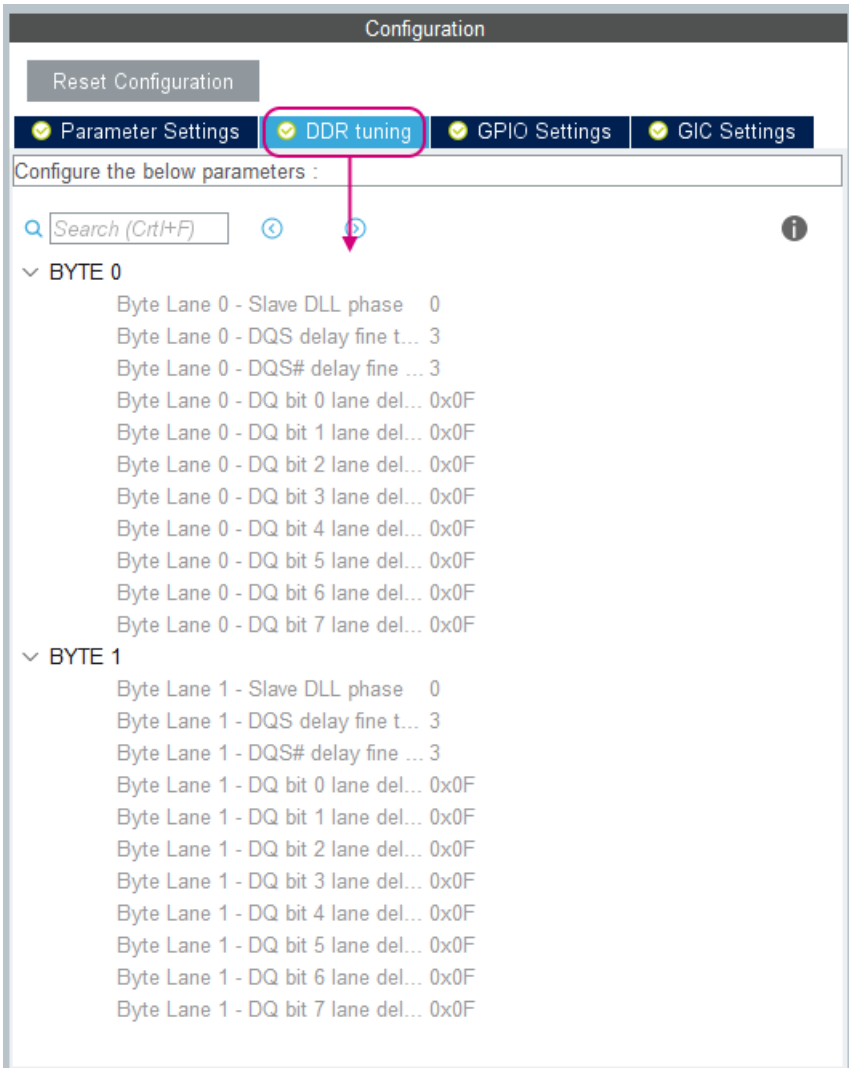
### ［DDR tuning］タブ（読取り専用）

チューニング･パラメータの変更は［DDR tuning］タブで確認できます。これらのパラメータは DDR の設定パネルでは読取り専用で（図 161 参照）、変更は調整後に反映されます。これらは DQS の位置と DQ ラインの遅延に関連するパラメータです。

– "Slave DDL Phase"、"DQS delay fine tuning"、"DQS# delay fine tuning" は、特定のバイトに対応する DQS ストローブ信号の位置を定義します。この位置は、DQ ラインのアイ･ダイアグラムが最適になる位置です。

– "DQ bit x lane delay fine tuning" は、各ビットへの配線長のばらつきを補償するために、特定のバイトの特定のビット x に適用する遅延を定義します。

**図 161. DDR のチューニング･パラメータ**



### 5.3.2 ターゲットへの接続と DDR レジスタの読込み

DDR のテストとチューニングに対応するために、STM32CubeMX とターゲットとの接続を確立する必要があります。具体的には、**DDR DDR インタラクティブ・プロトコル**を使用して次のように **U-Boot SPL** に接続します。

- DDR 対話プロトコルは、**Basic boot scheme U-Boot SPL** バイナリのみで用意され、UART4 ペリフェラルのインスタンスに対してサポートされています。
- U-Boot SPL は、UART4 上で STM32CubeMX との接続を検出すると初期化プロセスを停止し、STM32CubeMX からのコマンドを受け付けるようになります。

この接続には次の 2 つの状況があります。

1. Flash メモリに U-Boot SPL バイナリが存在している
2. U-Boot SPL を SYSRAM に読み込み。これは DDR テスト及びチューニングがされていない状態、U-Boot SPL を SYSRAM に読み込む必要がある

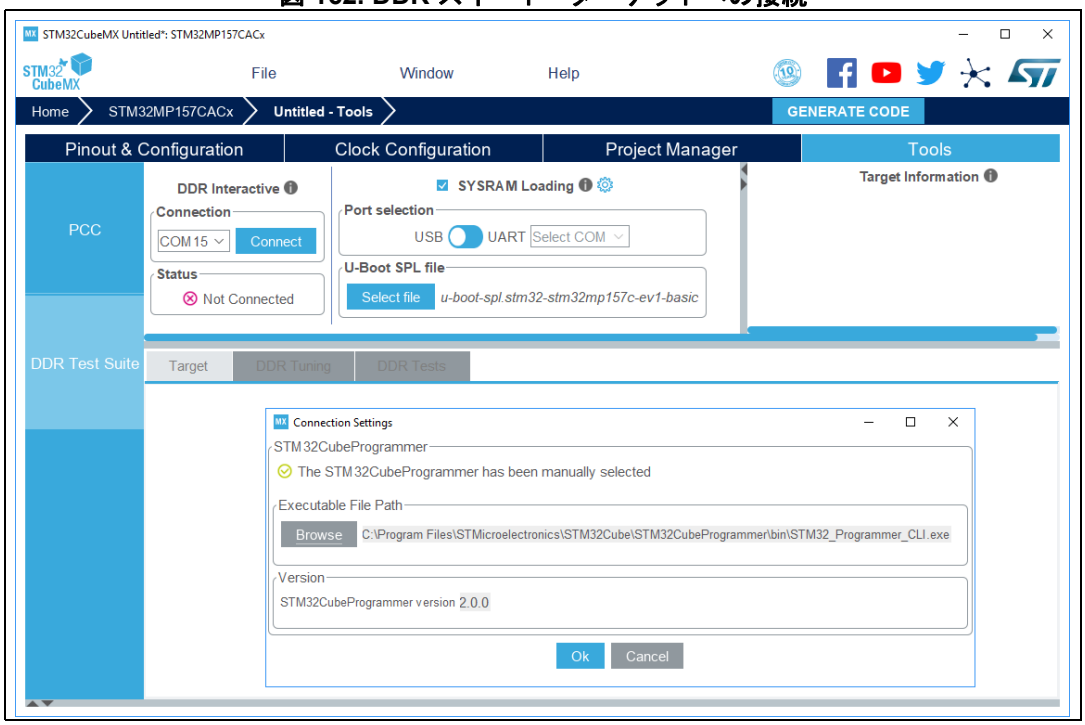
#### 前提条件

- ファームウェアをアップグレードするための ST-Link USB ドライバのインストール：Windows の場合、STSW-LINK009 の最新バージョンを使用する必要があります。Linux の場合は、STSW-LINK007 ドライバを使用する必要があります。どちらも www.st.com からダウンロードできます。
- STM32CubeProgrammer のインストール（SYSRAM への読込み専用）：インストーラは www.st.com からダウンロードできます。

#### ターゲットへの接続

ターゲットに接続するには、図 162 に示すように COM ポートを選択する必要があります。

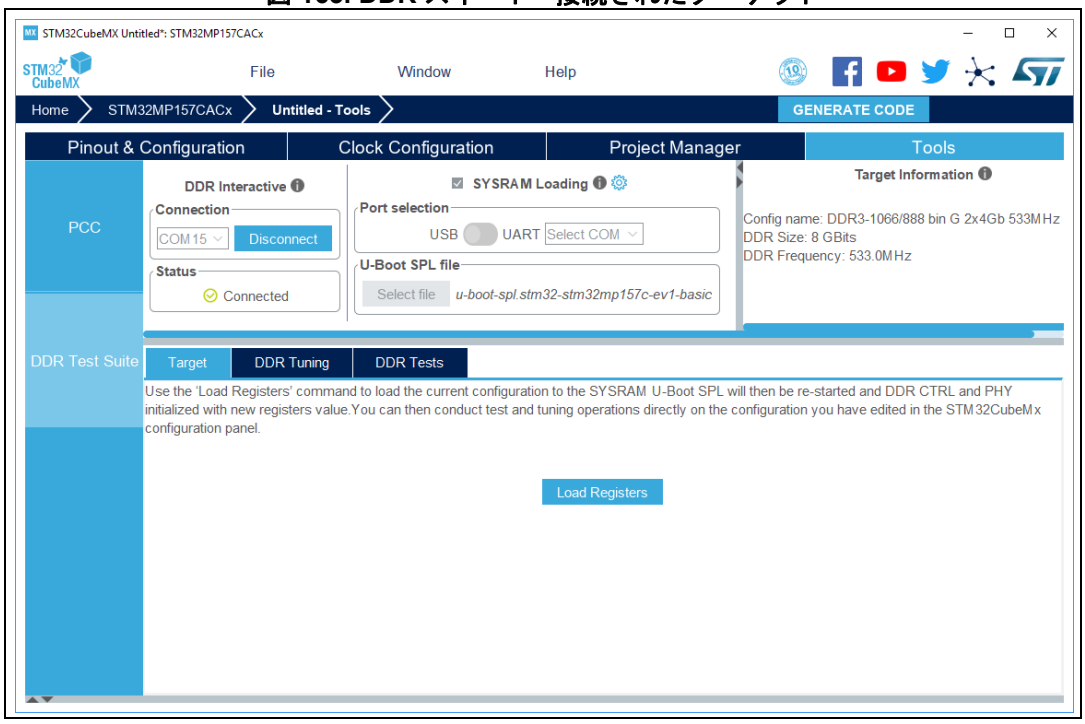
**図 162. DDR スイート - ターゲットへの接続**



SYSRAM に U-Boot SPL を読み込む必要がある場合、STM32CubeProgrammer ツールを使用して UART または USB を介した読込みが可能です。STM32CubeMX で STM32CubeProgrammer ツールを自動的に検出できない場合は、ツールの場所を接続設定のウィンドウで指定する必要があります。このウィンドウは  をクリックすると開きます。ビルド･イメージのフォルダで U-Boot SPL のファイルを手動で選択する必要があります。

接続が確立されると、さまざまなサービスやターゲットの情報が得られます（図 163 参照）。

**図 163. DDR スイート - 接続されたターゲット**




## 出力/ログ･メッセージ

STM32CubeMX は、DDR スイート関連の動作ログ（図 164 参照）と対話プロトコルの通信ログ（図 165 参照）を出力します。[Window] メニューで出力を有効にすると表示されます。

**図 164. DDR 動作ログ**

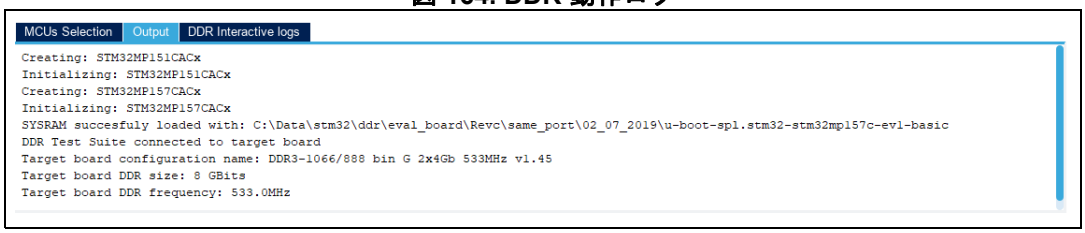
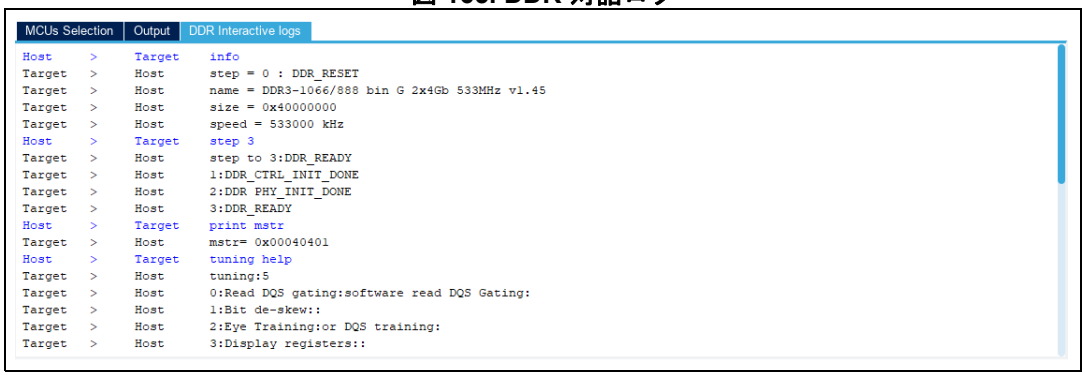


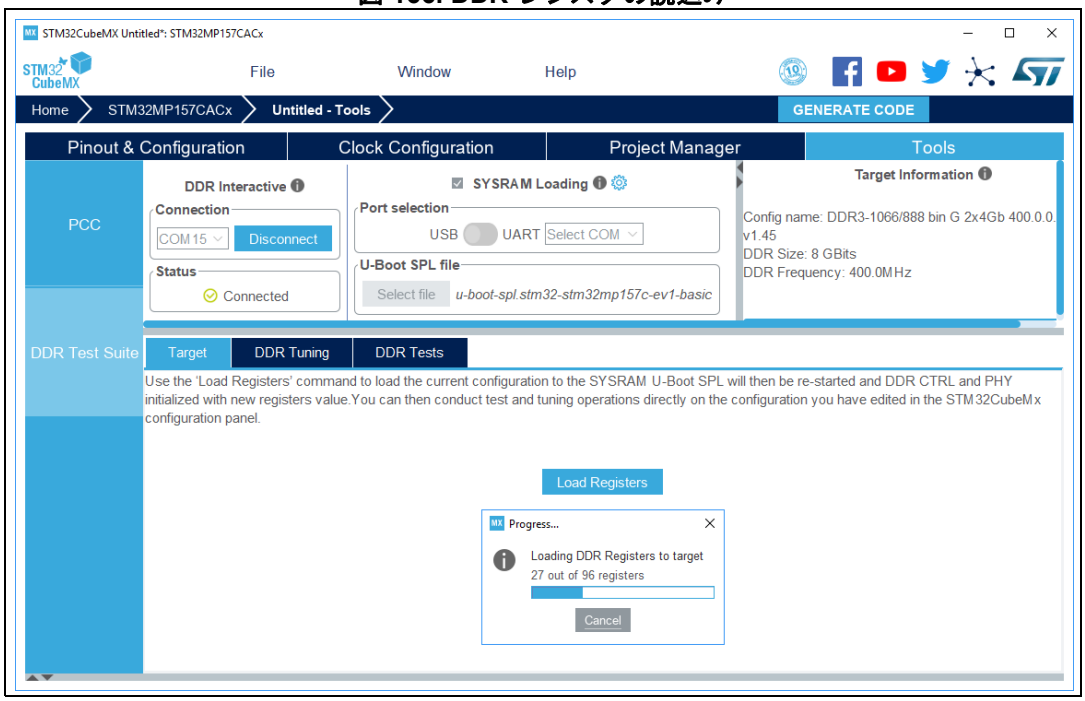
図 165. DDR 対話ログ




## DDR レジスタの読込み（任意）

DDR 対話モードで接続すると、DDR の現在の設定を SYSRAM に読み込むことができます。

図 166. DDR レジスタの読込み




使用する U-Boot SPL に既に、必要な DDR 設定が存在する場合は、このステップは省略してもかまいません。このステップを使用すると、これらのレジスタ値によって DDR コントローラと PHY を初期化し、デバイス·ツリーの生成や専用の U-Boot SPL バイナリ·ファイルなしで、すばやく設定をテストできます。

### 5.3.3 DDR のテスト

#### 前提条件

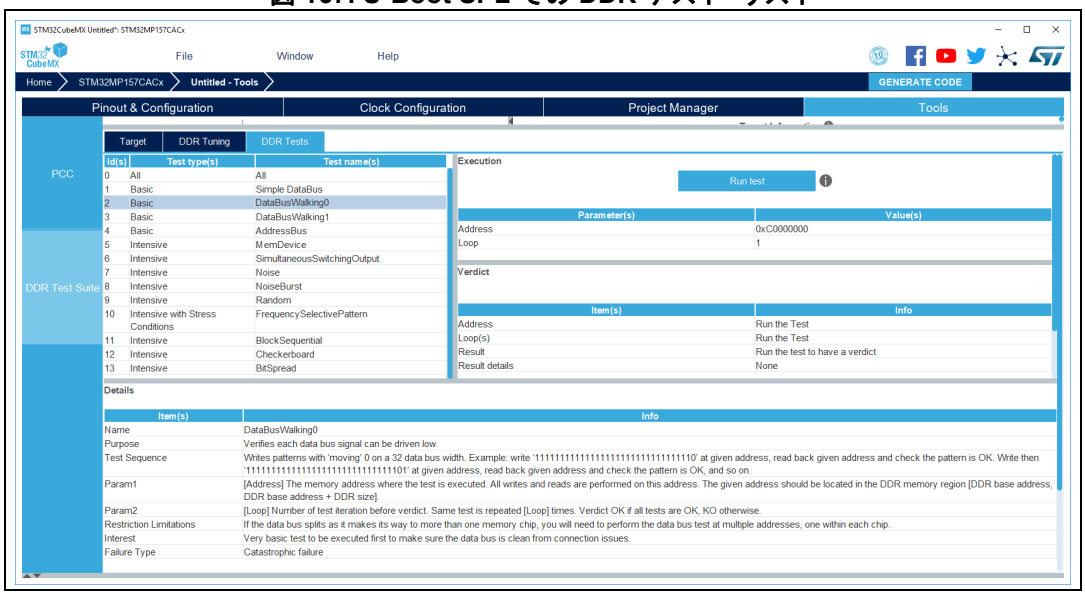
DDR をテストするには、以下の条件を満たしている必要があります。

- DDR スイートが接続状態にあること
- U-Boot SPL（デバイス･ツリーの DDR レジスタ･ファイルを使用）または DDR レジスタ（セクション 5.3.2：参照）から DDR の設定を取得してメモリで使用できること

#### DDR のテスト･リスト

DDR のテストは U-Boot SPL の一部として実行されます（図 167 参照）。

**図 167. U-Boot SPL での DDR テスト･リスト**




U-Boot SPL を変更して新しいテストを追加できます。

ほとんどのテストは、実行前に設定するパラメータを伴います。このパラメータとして以下があります。

- ［Address］：テストの実行対象とするメモリ･アドレス。書込みと読出しは、すべてこのアドレスに対して実行されます。指定するアドレスが DDR メモリ領域（DDR ベース･アドレス ~ DDR ベース･アドレス + DDR サイズの範囲）に存在する必要があります。
- STM32MP15 の場合、DDR ベース･アドレスは 0xC0000000 です（たとえば 4 Gbit の DDR サイズは 0x20000000 です）。
- ［Loop］：判定までに実行するテストの反復回数。同じテストが［Loop］回だけ繰り返されます。
- ［Size］：テスト対象領域のバイト単位のサイズです。このサイズは、4 以上の 4 の倍数とする必要があります（32 bit の符号なし整数の読み書きを実行します）。このサイズには、最大で DDR のサイズを指定できます。
- ［Pattern］：読み書きの動作に使用する 32 bit のパターンです。

DDR スイートには、ユーザが誤った値を指定しないように自動修正機能が組み込まれています。

すべてのテストは、データ･キャッシュを無効、命令キャッシュを有効にして実行されます。

### DDR のテスト結果

U-Boot SPL によってテストの判定結果が表示されます。テストに使用されたパラメータ、合否の状態（Pass または Fail）、結果の詳細が表示されます（図 168 参照）。出力とログ･パネルでテストの履歴を確認できます（図 169 参照）。

**図 168. DDR テスト･スイートの結果**

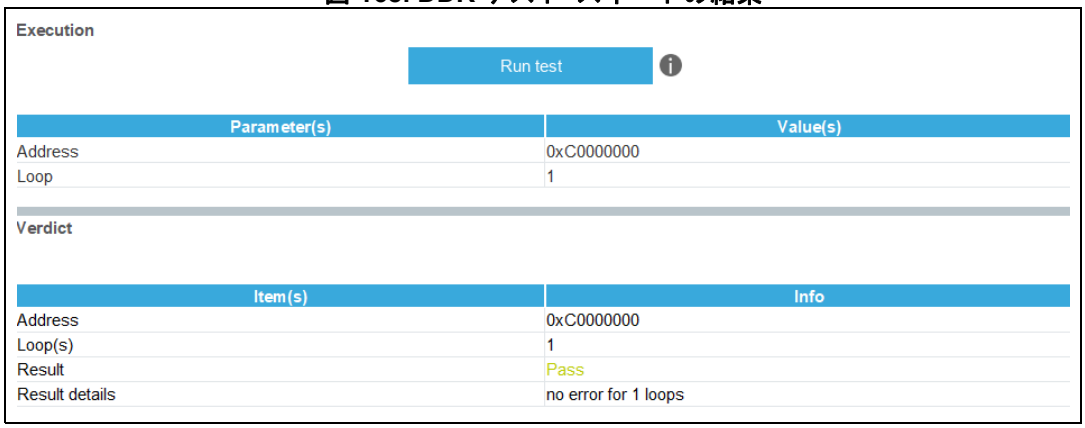
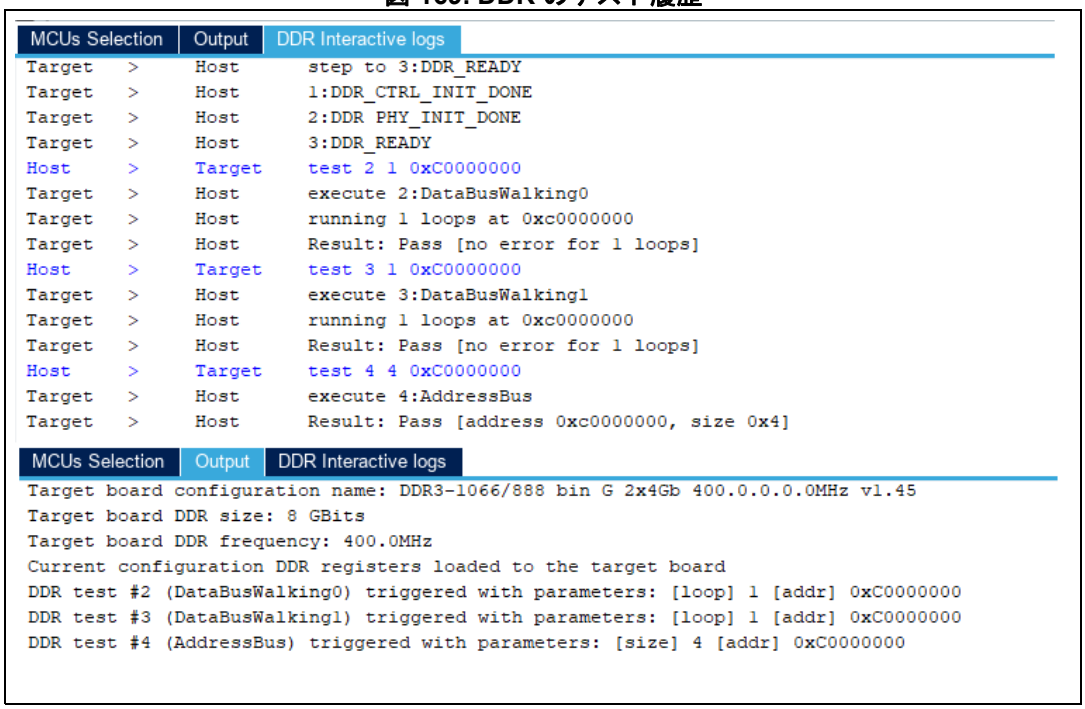



**図 169. DDR のテスト履歴**



### 5.3.4 DDR のチューニング

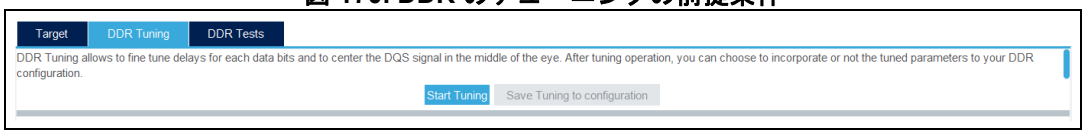
#### 前提条件

DDR のチューニングを進めるには、以下の前提条件を満たす必要があります。

- DDR スイートが接続状態にあること
- U-Boot SPL（デバイス·ツリーの DDR レジスタを使用）または DDR レジスタ（DDR レジスタの読込み（任意）参照）から有効な DDR の設定を取得してメモリで使用できること

DDR のチューニング機能により、ハードウェア設計の若干の不完全性は補完できるため最適の動作が得られます（DDR の配線経路設計の指針については、www.st.com から入手可能なアプリケーション·ノート AN5122 を参照してください）。

**図 170. DDR のチューニングの前提条件**




#### チューニング可能な信号

チューニングが可能な信号は次のとおりです。

- DQS 信号：各データ·バイトの位置
- 8 つの DQ ビット：各データ·バイトの遅延

DDR レジスタの一部は、対応するチューニング済み設定の保存専用です。

- "Slave DDL Phase"、"DQS delay fine tuning"、"DQS# delay fine tuning" は、特定のバイトに対する DQS ストローブ信号の位置を定義します。この位置は、DQ ラインのアイ·ダイアグラムが最適になる位置です。
- "DQ bit x lane delay fine tuning" は、各ビットへの配線長のばらつきを補償するために、特定のバイトの特定のビット x に適用する遅延を定義します。

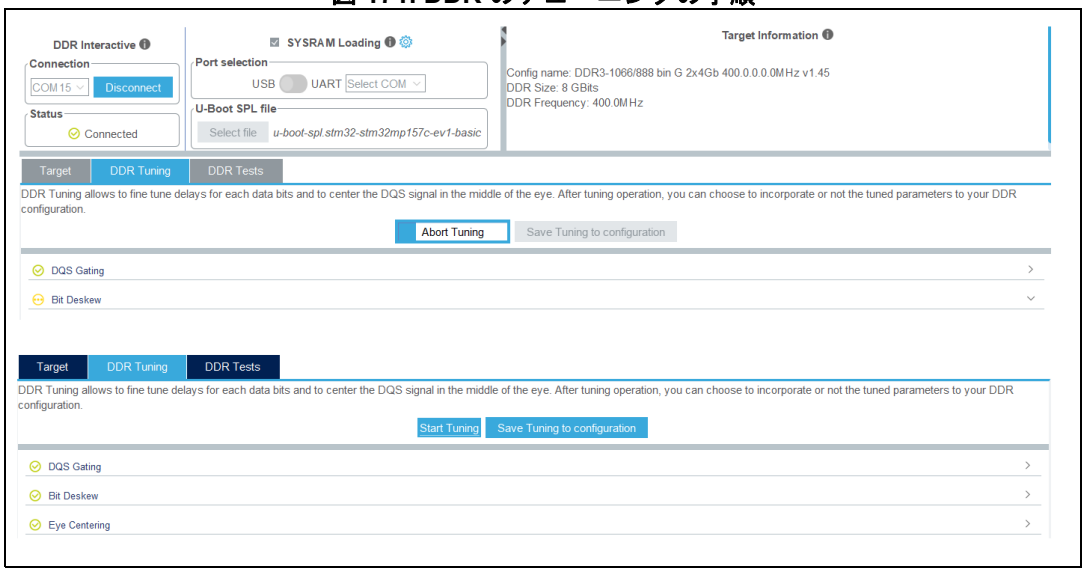
**注：** **チューニングしたパラメータのばらつきが一定の範囲に収まっていることを確認するために、複数のボードでチューニングを実行することをお勧めします。**

#### チューニングの手順

チューニングは、次の 3 つの連続するステップで実行します（図 171 参照）。

1. DQS のゲーティング
2. ビットのデスキュー
3. アイ·トレーニング

図 171. DDR のチューニングの手順

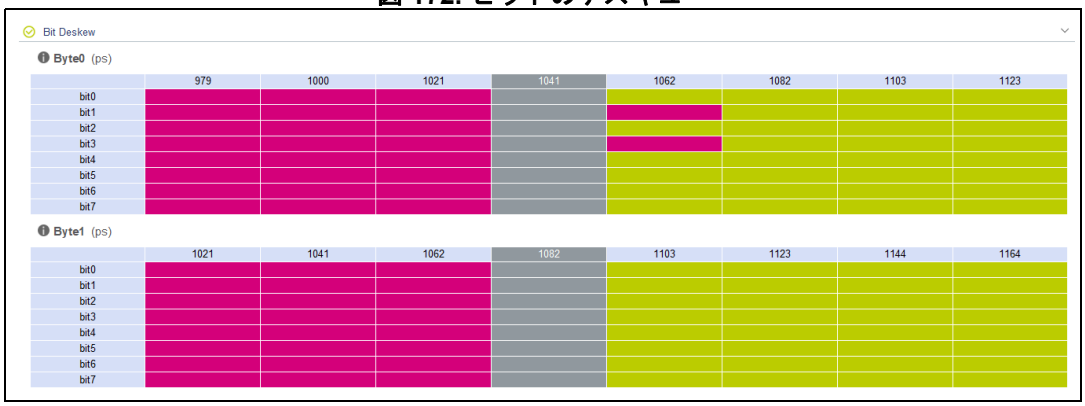



## ビットのデスキュー

[Bit Deskew] パネル（図 172 参照）には、以下の特性がグラフィック表示されます。

- 特定バイトに対する DQS 信号の最適位置：DQ ラインの遅延の調整に使用します。
- 対象バイトの各 DQ ラインに適用する遅延。遅延は 20.56 ps を単位として4 段階の設定が可能です。したがって、ビット･レーンの遅延は 0 ~ 61.68 ps の範囲でチューニングできます。

図 172. ビットのデスキュー

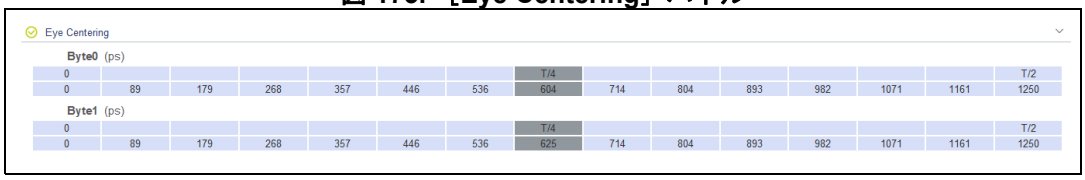



## アイ･トレーニング（センタリング）

[Eye Centering] パネル（図 173）には、各バイトの半周期で DQS 信号が占める最終的な最適位置が表示されます。

- DQS の位置は 36 ~ 144 度の範囲で粗調整できます（1/4 周期が 90 度）。
- チューニング値は、この粗調整値の周辺-61.68 ~ +82.24 ps の範囲で8段階で微調整できます

図 173. ［Eye Centering］パネル




## チューニングの結果の反映

DDR スイートでは、チューニングが完了したパラメータを現在の DDR 設定に反映できます（図 174 参照）。［DDR tuning］タブがチューニング結果に応じて更新されます（図 175 参照）。

図 174. DDR のチューニング - 設定への保存

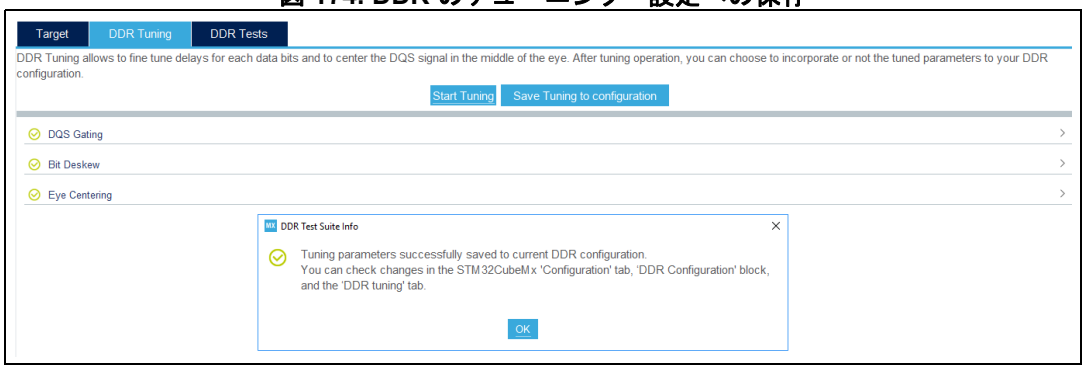
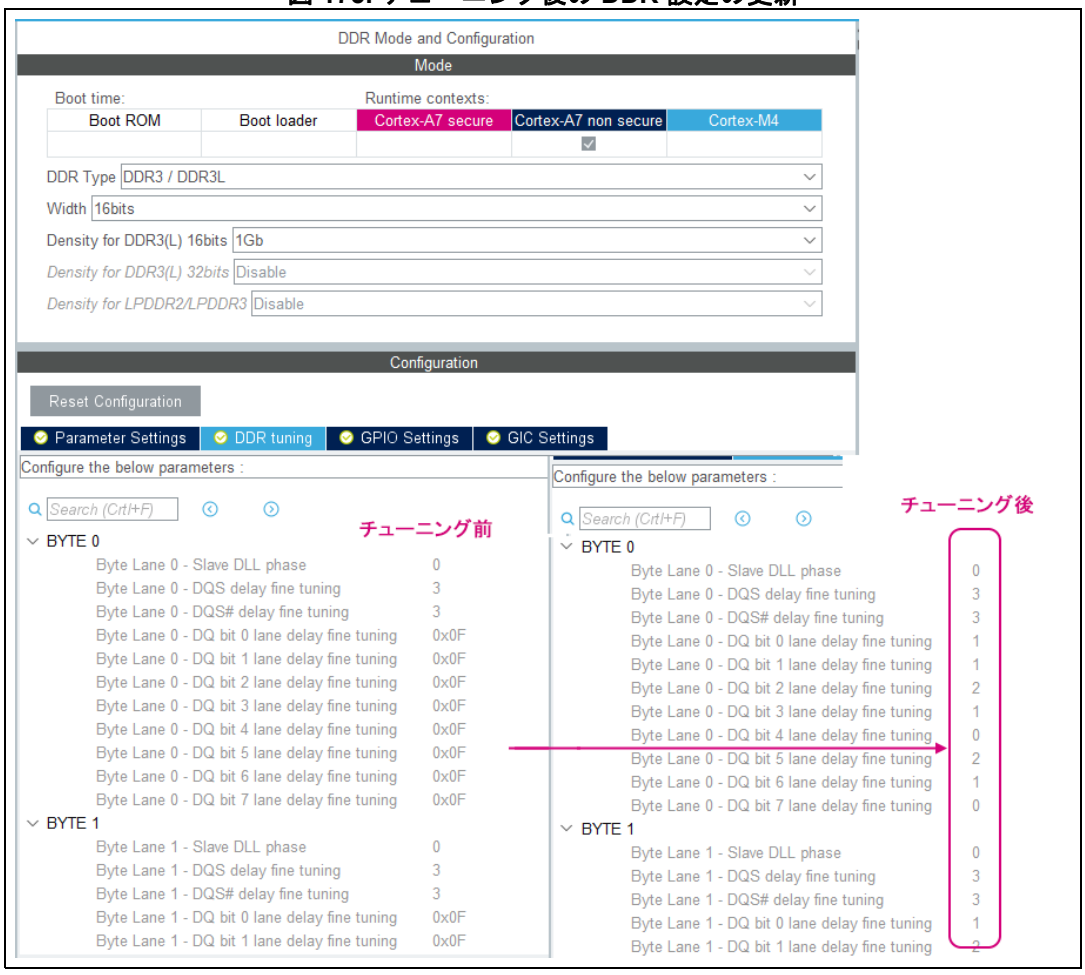


図 175. チューニング後の DDR 設定の更新



# 6 STM32CubeMX の C コード生成の概要

## 6.1 HAL ドライバのみによる STM32Cube のコード生成（デフォルト･モード）

C コード生成プロセスで STM32CubeMX によって実行されるアクションを次に示します。

1. 該当の STM32Cube マイクロコントローラ･パッケージがない場合は、それがユーザ･リポジトリからダウンロードされます。STM32CubeMX のリポジトリ･フォルダは、**[Help]** → **[Updater Settings]** メニューで指定します。
2. ファームウェア･パッケージから該当のファイルが Drivers/CMSIS フォルダと Drivers/STM32F4_HAL_Driver フォルダにコピーされます。ミドルウェアを選択している場合は Middleware フォルダにもコピーされます。
3. マイクロコントローラの設定に対応した初期化 C コード（.c/.h ファイル）が生成され、Inc フォルダと Src フォルダに格納されます。デフォルトでは、次のファイルが含まれます。
   - **stm32f4xx_hal_conf.h** ファイル：有効な HAL モジュールを定義し、高速外部オシレータ周波数などのいくつかのパラメータを事前定義済みのデフォルト値に設定するか、ユーザ設定（クロック･ツリー）に従って設定します。
   - **stm32f4xx_hal_msp.c**（MSP = MCU Support package）: ユーザ設定（ピン割当て、クロックの有効化、DMA と割込みの使用）に従ってペリフェラル･インスタンスを設定するすべての初期化関数を定義します。
   - **main.c** は次の各処理を実行します。

     HAL_init() 関数を呼び出してマイクロコントローラを既知の状態にリセットします。この関数は、すべてのペリフェラルをリセットし、Flash メモリ･インタフェースと SysTick を初期化します。

     システム･クロックを設定および初期化します。

     ペリフェラルによって使用されていない GPIO を設定および初期化します。

     設定されているペリフェラルごとに、ペリフェラル初期化関数を定義して呼び出します。ペリフェラル初期化関数は、ハンドル構造体を定義して、対応するペリフェラルの HAL init 関数に渡します。これを受け取った HAL init 関数は、ペリフェラルの HAL MSP 初期化関数を呼び出します。イーサネットや USB などの LwIP ミドルウェアを使用すると、基盤となる初期化 C コードが main.c から LwIP 初期化 C コード自体に移動することに注意してください。
   - **main.h** ファイル：

     次のように、**[Pinout]** タブで設定されたピン･ラベルに対応する define ステートメント、および **[Configuration]** タブで追加されたユーザ･プロジェクト定数を記述します（例については、図 176 と図 177 を参照）。

```
#define MyTimeOut       10
#define LD4_Pin         GPIO_PIN_12
#define LD4_GPIO_Port   GPIOD
#define LD3_Pin         GPIO_PIN_13
#define LD3_GPIO_Port   GPIOD
#define LD5_Pin         GPIO_PIN_14
#define LD5_GPIO_Port   GPIOD
#define LD6_Pin         GPIO_PIN_15
#define LD6_GPIO_Port   GPIOD
```

図 176. define ステートメントを生成するピンのラベル

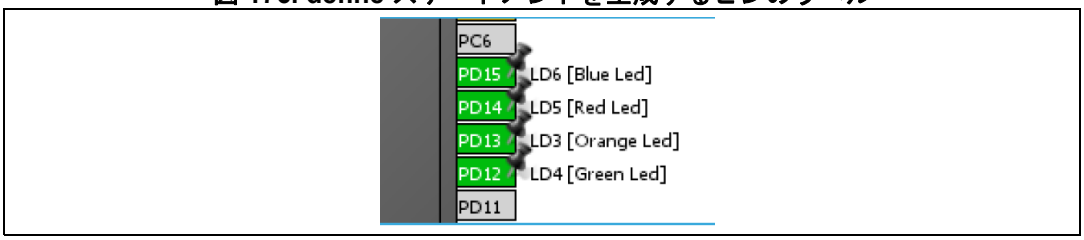
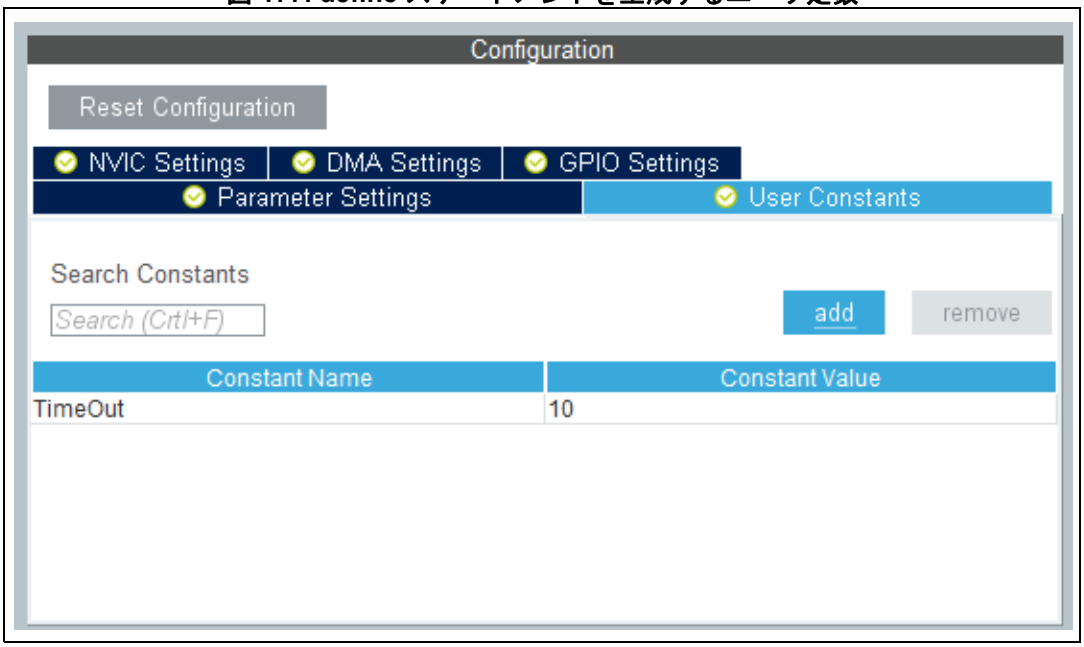



図 177. define ステートメントを生成するユーザ定数

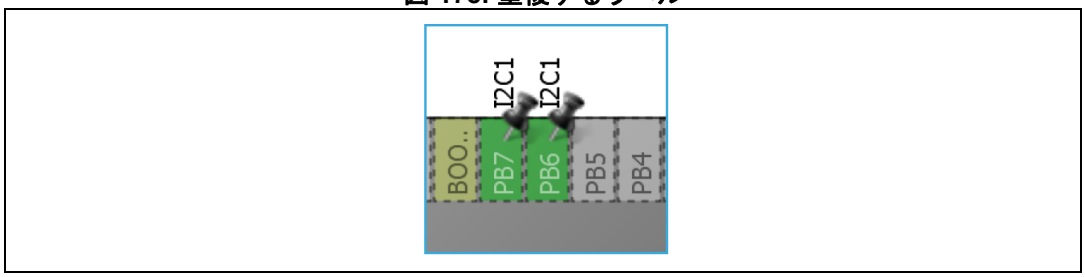



重複するラベルが存在する場合は、ピンのポート文字とインデックス番号で構成する一意のサフィックスが追記され、関連する define ステートメントの生成に使用されます。

図 178 に示す重複した I2C1 ラベルの例で生成したコードを次に示します。ここでは、ポート B のピン 6 については、本来の define ステートメントの I2C1 ラベルを変更せず、ピン 7 の define ステートメントにサフィックス B7 が追加されています。

```
#define I2C1_Pin            GPIO_PIN_6
 #define I2C1_GPIO_Port      GPIOB
#define I2C1B7_Pin          GPIO_PIN_7
#define I2C1B7_GPIO_Port    GPIOB
```

図 178. 重複するラベル




生成されたプロジェクトをコンパイルするために、define ステートメントは厳密な命名規約に従う必要があります。先頭は、文字またはアンダースコアおよび対応するラベルとします。マイナス記号、丸かっこ、角かっこなどの特殊文字は使用できません。ラベルに使用されている特殊文字は、define での名前では自動的にアンダースコアに置き換えられます。

ラベルに [] または () で囲まれた文字列が使用されていると、define での名前では最初の文字列のみが使用されます。たとえば、「**LD6** [Blue Led]」というラベルに対応する define ステートメントは、次のようになります。

```
#define LD6_Pin    GPIO_PIN_15
#define LD6_GPIO_Port    GPIOD
```

define ステートメントは、生成された初期化コードで GPIO を設定するために使用します。次に示す例では、ラベルがそれぞれ Audio_RST_Pin および LD4_Pin であるピンを、該当の define ステートメントで初期化しています。

```
/* Configure GPIO pins : LD4_Pin Audio_RST_Pin */
GPIO_InitStruct.Pin = LD4_Pin | Audio_RST_Pin;
GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
GPIO_InitStruct.Pull = GPIO_NOPULL;
GPIO_InitStruct.Speed = GPIO_SPEED_LOW;
HAL_GPIO_Init(GPIOD, &GPIO_InitStruct);
```

4. 最後に Projects フォルダが生成されます。このフォルダには、ユーザのプロジェクト設定に適合するツールチェーン固有のファイルが格納されます。IDE 固有のプロジェクト·ファイルをダブルクリックすると IDE が起動し、プロジェクトがロードされて、編集、ビルド、およびデバッグができるようになります。

## 6.2 Low Layer ドライバを使用する STM32Cube コードの生成

STM32CubeMX では、STM32H7 と STM32P1 シリーズを除く、すべての STM32 シリーズについて、ペリフェラルの HAL ドライバまたはペリフェラルの Low Layer （LL）ドライバのどちらかに基づいて、ペリフェラルの初期化コードを生成できます。

どちらを選択するかは、［Project Manager］ビューで設定します（セクション 4.9.3：［Advanced Settings］タブ 参照）。

LL ドライバを使用できるペリフェラルは、アクセスの最適化を必要として、ソフトウェア設定が複雑ではないものに限られます。LL サービスを使用すると、関連するペリフェラル·レジスタの内容を変更することによってより直接的な操作を実行できます。

- サポートされているペリフェラルの例：RCC、ADC、GPIO、I2C、SPI、TIM、USART
- LL ドライバでサポートされていないペリフェラルの例：USB、SDMMC、FSMC

LL ドライバは、STM32CubeL4 パッケージに含まれています。

- STM32Cube_FW_L4_V1.6\Drivers\STM32L4xx_HAL_Driver フォルダの Inc ディレクトリと Src ディレクトリで、HAL ドライバ（**stm32l4_hal_<ペリフェラル名>**）の次に置かれています。
- 次の命名規約に従っているので、簡単に認識できます。**stm32l4_ll_<ペリフェラル名>**

HAL ドライバと LL ドライバの詳細については、ユーザ･マニュアル『Description of STM32L4/L4+ HAL and low-layer drivers』（UM1884）を参照してください。

LL ドライバまたは HAL ドライバのどちらを使用するかはペリフェラル単位で指定するので、同一プロジェクトに HAL ドライバと LL ドライバの混在が可能です。

次の各表に、STM32CubeMX の 3 種類のプロジェクト生成オプションとして、HAL のみとしたコード、LL のみとしたコード、および HAL と LL が混在するコードの主な違いを示します。

**表 18. LL と HAL のコード生成の比較：STM32CubeMX プロジェクトで扱うドライバ**

| プロジェクト設定と付属するドライバ | HAL のみ | LL のみ | HAL と LL の混在 | コメント |
|---|---|---|---|---|
| CMSIS | はい | はい | はい | - |
| STM32xxx_HAL_Driver | HAL ドライバファイルのみ | LL ドライバファイルのみ | HAL と LL のドライバファイルの混在 | プロジェクト設定オプションで［Copy only the necessary files］を設定すると、ペリフェラルの選択に応じた個々の設定に必要なドライバ･ファイルのみがコピーされます。［all used libraries］オプションを設定すると、ドライバ･ファイル一式がコピーされます。 |

**表 19. LL と HAL のコード生成の比較：STM32CubeMX によるヘッダファイルの生成**

| 生成されるヘッダ･ファイル | HAL のみ | LL のみ | HAL と LL の混在 | コメント |
|---|---|---|---|---|
| main.h | はい | はい | はい | このファイルには、include ステートメントおよびユーザ定数（GPIO ラベルとユーザ定数）の define ステートメントが含まれます。 |
| stm32xxx_hal_conf.h | はい | いいえ | 可能 | このファイルは、プロジェクトに必要な HAL モジュールを有効化します。 |
| stm32xxx_it.h | はい | はい | はい | 割込みハンドラのヘッダ･ファイル。 |
| stm32xx_assert.h | いいえ | はい | はい | このファイルには､関数パラメータのチェックに使用する assert マクロと関数が含まれます。 |

表 20. LL と HAL の比較：STM32CubeMX によるソースファイルの生成

| 生成される<br>ソース･ファイル | HAL のみ | LL のみ | HAL と LL の混在 | コメント |
|---|---|---|---|---|
| main.c | はい | はい | はい | このファイルには、main 関数のほか、STM32CubeMX で生成される関数が必要に応じて記述されます。 |
| stm32xxx_hal_msp.c | はい | いいえ | 可能 | このファイルには、次の関数が含まれます。<br>– HAL_MspInit<br>– HAL ドライバを使用するペリフェラルの場合：HAL_<ペリフェラル名>_MspInit、HAL_<ペリフェラル名>_MspDeInit<br>これらの関数は、HAL ドライバを使用するペリフェラルでのみ使用できます。 |
| stm32xxx_it.c | はい | はい | はい | 割込みハンドラのソース･ファイル。 |

表 21. LL と HAL の比較：STM32CubeMX による関数と関数呼出しの生成

| 生成される<br>ソースファイル | HAL のみ | LL のみ | HAL と LL の混在 | コメント |
|---|---|---|---|---|
| Hal_init() | main.c で呼び出し | 未使用 | main.c で呼び出し | このファイルは、次の機能を実行します。<br>– Flash メモリのプリフェッチおよび命令とデータのキャッシュを設定<br>– タイムベース･ソースとして SysTick タイマを選択<br>– NVIC グループ優先度を設定<br>– マイクロコントローラの低レベル初期化 |
| Hal_msp_init() | stm32xxx_hal_msp.c に生成され、HAL_init() で呼び出し | 未使用 | stm32xxx_hal_msp.c に生成され、HAL_init() で呼び出し | この関数は、ペリフェラルのリソースを設定します (1)。 |
| MX_<ペリフェラル名>_Init() | [1]：ペリフェラルを設定し、HAL_<ペリフェラル名 >_Init() を呼び出し | [2]：LL 関数を使用してペリフェラルとペリフェラルのリソースを設定(1)。<br><br>LL_Peripheral_Init() を呼び出し | – <ペリフェラル名> で HAL ドライバが選択されていると、[1]：ペリフェラルを設定し、HAL_<ペリフェラル名>_Init() を呼び出しに続いて関数の生成と呼出しを実行<br>– <ペリフェラル名> で LL ドライバが選択されていると、[2]：LL 関数を使用してペリフェラルとペリフェラルのリソースを設定。に続いて関数の生成と呼出しを実行 | このファイルでペリフェラルを設定します。<br><br><ペリフェラル名> で LL ドライバが選択されていると、ペリフェラルのリソースも設定します (1)。 |

表 21. LL と HAL の比較：STM32CubeMX による関数と関数呼出しの生成 （続き）

| 生成されるソースファイル | HAL のみ | LL のみ | HAL と LL の混在 | コメント |
|---|---|---|---|---|
| HAL_<ペリフェラル名>_MspInit() | [3] : <ペリフェラル名> で HAL ドライバが選択されていると、stm32xxx_hal_msp.c に生成 | 未使用 | <ペリフェラル名> には HAL ドライバのみ選択可能。[3] : <ペリフェラル名> で HAL ドライバが選択されていると、stm32xxx_hal_msp.c に生成 に続いて関数を生成し、呼び出し | ペリフェラルのリソースを設定します (1)。 |
| HAL_<ペリフェラル名>_MspDeInit() | [4] : <ペリフェラル名> で HAL ドライバが選択されていると、stm32xxx_hal_msp.c に生成 | 未使用 | <ペリフェラル名> には HAL ドライバのみ選択可能。[4] : <ペリフェラル名> で HAL ドライバが選択されていると、stm32xxx_hal_msp.c に生成 に続いて関数を生成し、呼び出し | この関数を使用して、ペリフェラルのリソースを解放できます。 |

1. ペリフェラルには次のリソースがあります。
   - ペリフェラル·クロック
   - ピン配置設定（GPIO）
   - ペリフェラルの DMA リクエスト
   - ペリフェラルの割込みリクエストと優先度

図 179. HAL ベースのペリフェラルの初期化：usart.c のコードの一部

```
USART Peripheral initialization - HAL-based
void MX_USART1_UART_Init(void)
{                                                    ペリフェラルの設定
  huart1.Instance = USART1;
  huart1.Init.BaudRate = 115200;
  huart1.Init.WordLength = UART_WORDLENGTH_7B;
  huart1.Init.StopBits = UART_STOPBITS_1;
  ...
  if (HAL_UART_Init(&huart1) != HAL_OK)
  {
    Error_Handler();
  }
}
void HAL_UART_MspInit(UART_HandleTypeDef* uartHandle)
{                                                    ペリフェラルのリソースを設定
  GPIO_InitTypeDef GPIO_InitStruct;
  if(uartHandle->Instance==USART1)
  {
    /* Peripheral clock enable */
    __HAL_RCC_USART1_CLK_ENABLE();
    /* USART1 GPIO Configuration */
    GPIO_InitStruct.Pin = GPIO_PIN_10;
    GPIO_InitStruct.Mode = GPIO_MODE_AF_PP;
    GPIO_InitStruct.Pull = GPIO_PULLUP;
    ...
    HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);
  }
}
void HAL_UART_MspDeInit(UART_HandleTypeDef* uartHandle)
{                                                    ペリフェラルのリソースをリリース
  if(uartHandle->Instance==USART1)
  {
      /* Peripheral clock disable */
    __HAL_RCC_USART1_CLK_DISABLE();
    /* USART1 GPIO Configuration */
    HAL_GPIO_DeInit(GPIOA, GPIO_PIN_10);
    HAL_GPIO_DeInit(GPIOB, GPIO_PIN_6);
  }
}
```

図 180. LL ベースのペリフェラルの初期化：usart.c のコードの一部

```
USART Peripheral Initialization using LL drivers
void MX_USART1_UART_Init(void)
{
  LL_USART_InitTypeDef USART_InitStruct;
  LL_GPIO_InitTypeDef GPIO_InitStruct;
  /* Peripheral clock enable */
  LL_APB2_GRP1_EnableClock(LL_APB2_GRP1_PERIPH_USART1);

    /**USART1 GPIO Configuration      ペリフェラルのリソースを設定
    PA10      ------> USART1_RX
    PB6       ------> USART1_TX
    */
  GPIO_InitStruct.Pin = LL_GPIO_PIN_10;
  GPIO_InitStruct.Mode = LL_GPIO_MODE_ALTERNATE;
  GPIO_InitStruct.Speed = LL_GPIO_SPEED_FREQ_VERY_HIGH;
  GPIO_InitStruct.Pull = LL_GPIO_PULL_UP;
  GPIO_InitStruct.Alternate = LL_GPIO_AF_7;
  LL_GPIO_Init(GPIOA, &GPIO_InitStruct);

  GPIO_InitStruct.Pin = LL_GPIO_PIN_6;
  GPIO_InitStruct.Mode = LL_GPIO_MODE_ALTERNATE;
  GPIO_InitStruct.Speed = LL_GPIO_SPEED_FREQ_VERY_HIGH;
  GPIO_InitStruct.Pull = LL_GPIO_PULL_UP;
  GPIO_InitStruct.Alternate = LL_GPIO_AF_7;
  LL_GPIO_Init(GPIOB, &GPIO_InitStruct);
                                        ペリフェラルの設定
  USART_InitStruct.BaudRate = 115200;
  USART_InitStruct.DataWidth = LL_USART_DATAWIDTH_7B;
  USART_InitStruct.StopBits = LL_USART_STOPBITS_1;
  USART_InitStruct.Parity = LL_USART_PARITY_NONE;
  USART_InitStruct.TransferDirection = LL_USART_DIRECTION_TX_RX;
  USART_InitStruct.HardwareFlowControl = LL_USART_HWCONTROL_NONE;
  USART_InitStruct.OverSampling = LL_USART_OVERSAMPLING_16;

  LL_USART_Init(USART1, &USART_InitStruct);
  LL_USART_ConfigAsyncMode(USART1);
}
```

図 181. HAL と LL の比較：main.c のコードの一部

main.c HAL-based

```
  */
/* Includes ------------------------------------------
#include "main.h"
#include "stm32l4xx_hal.h"
#include "usart.h"
#include "gpio.h"

void SystemClock_Config(void);
void Error_Handler(void);

int main(void)
{
  /* Reset of all peripherals,
  Initializes the Flash interface and the Systick.
  **/
  HAL_Init();

  /* Configure the system clock */
  SystemClock_Config();

  /* Initialize all configured peripherals */
  MX_GPIO_Init();
  MX_USART1_UART_Init();
```

main.c LL-based

```
  */
/* Includes ------------------------------------------
#include "main.h"
#include "usart.h"
#include "gpio.h"

void SystemClock_Config(void);
void Error_Handler(void);

int main(void)
{
  /* Reset of all peripherals,
  Initializes the Flash interface and the Systick.
  **/
  LL_Init();

  /* Configure the system clock */
  SystemClock_Config();

  /* Initialize all configured peripherals */
  MX_GPIO_Init();
  MX_USART1_UART_Init();
```

## 6.3 カスタム･コードの生成

STM32CubeMX は、FreeMarker テンプレート･エンジン（http://www.freemarker.org を参照）によるカスタム･コード生成をサポートしています。

### 6.3.1 FreeMarker ユーザ･テンプレート向け STM32CubeMX データ･モデル

STM32CubeMX は、次に示すいずれかのマイクロコントローラ設定情報の FreeMarker テンプレート･ファイル（.ftl 拡張子）に基づいてカスタム･コードを生成できます。

- ユーザ設定で使用されているマイクロコントローラのペリフェラルのリスト
- それらのペリフェラルのパラメータ値のリスト
- それらのペリフェラルで使用するリソースのリスト：GPIO、DMA リクエスト、割込み

ユーザ･テンプレート･ファイルには、STM32CubeMX のデータ･モデルとの互換性が必要です。そのためには、まずテンプレートが次の一連の行で始まる必要があります。

```
[#ftl]
[#list configs as dt]
[#assign data = dt]
[#assign peripheralParams =dt.peripheralParams]
[#assign peripheralGPIOParams =dt.peripheralGPIOParams]
[#assign usedIPs =dt.usedIPs]
```

さらに、次の行で終わる必要があります。

```
[/#list]
```

参考としてサンプル･テンプレート･ファイルが用意されています（図 182を参照）。

テンプレートにユーザ固有のコードがあれば、STM32CubeMX ではそのコードも生成されます。

次の例に示すように、サンプル･テンプレートを使用すると、データを生成した ftl コマンド名が、データに続いてコメントとして出力されます。

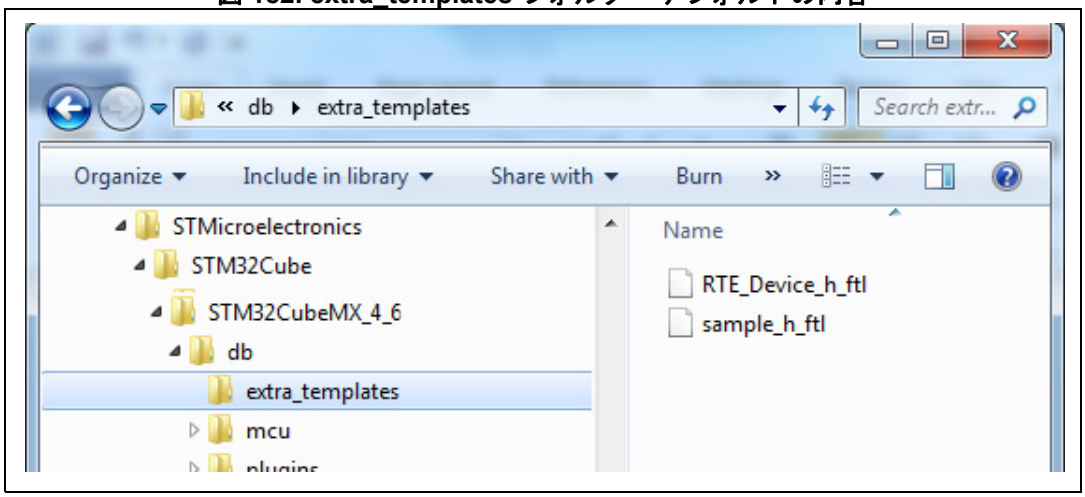
テンプレートに記述されている FreeMarker コマンド：

```
${peripheralParams.get("RCC").get("LSI_VALUE")}
```

生成されたコード：

```
LSI_VALUE :  32000       [peripheralParams.get("RCC").get("LSI_VALUE")]
```

**図 182. extra_templates フォルダ – デフォルトの内容**



### 6.3.2 ユーザ･テンプレートの保存と選択

FreeMarker テンプレート･ファイルは、STM32CubeMX のインストール先パスの db/extra_templates フォルダまたは他の任意のフォルダに配置できます。

実際のプロジェクトでは、**[Project Manager]** ビュー･メニューの［Code Generator］タブからアクセス可能な **[Template Settings]** ウィンドウで、そのプロジェクトに関連するテンプレート･ファイルを選択します（セクション 4.9：参照）。

### 6.3.3 カスタム･コードの生成

カスタム･コードを生成するには、STM32CubeMX のインストール先パスの db/extra_templates フォルダに FreeMarker テンプレート･ファイルを配置する必要があります（図 183を参照）。

<ユーザ･ファイル名>.<ファイル拡張子> の名前でカスタム･ファイルを生成するには、テンプレート･ファイルに <ユーザ･ファイル名>_<ファイル拡張子>.ftl の命名規約に従った名前を割り当てる必要があります。

デフォルトでは、カスタム･ファイルは、ユーザ･プロジェクト･ルート･フォルダに、.ioc ファイルと並んで生成されます（図 184を参照）。

別のフォルダにカスタム･コードを生成するには、extra_template フォルダ以下と同じフォルダ構造を、生成先フォルダに作成する必要があります（図 185を参照）。

**図 183. ユーザ･テンプレートが置かれた extra_templates フォルダ**

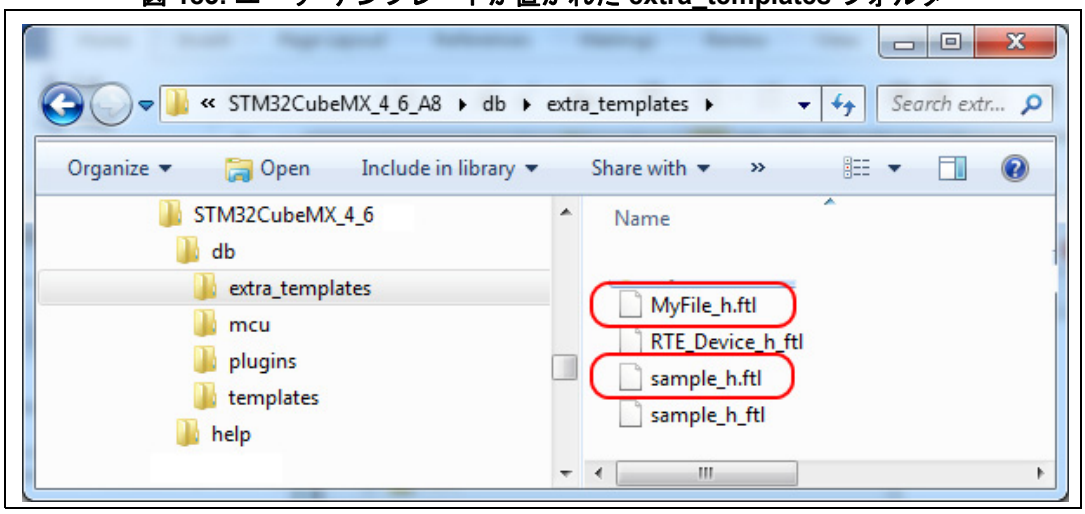


図 184. 対応するカスタム・ファイルが生成されたプロジェクト・ルート・フォルダ

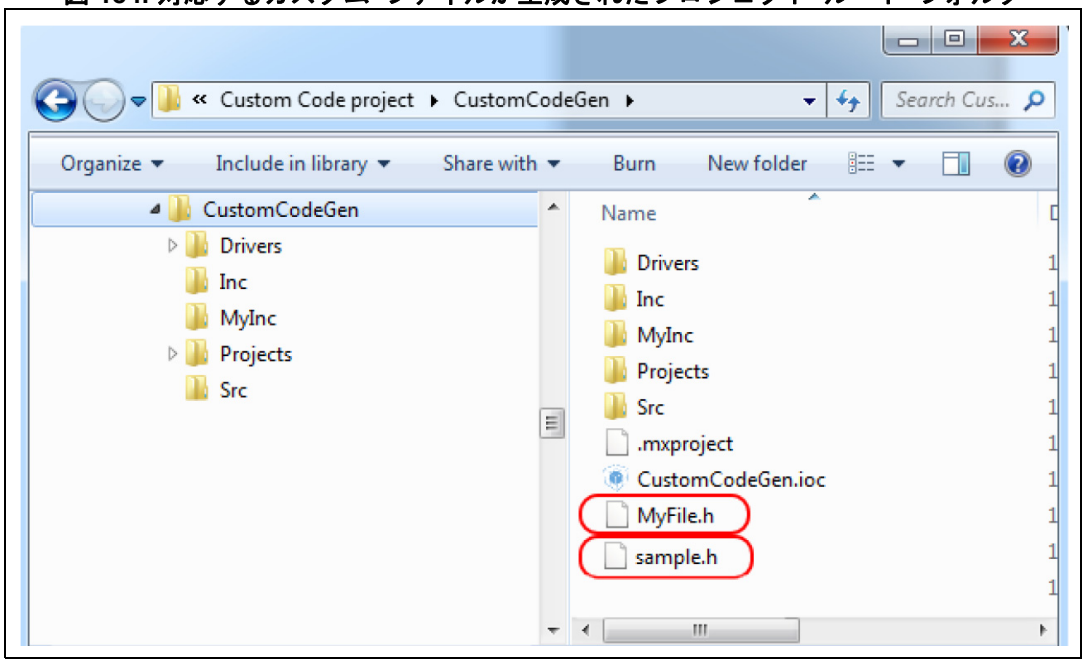



図 185. テンプレートのユーザ・カスタム・フォルダ

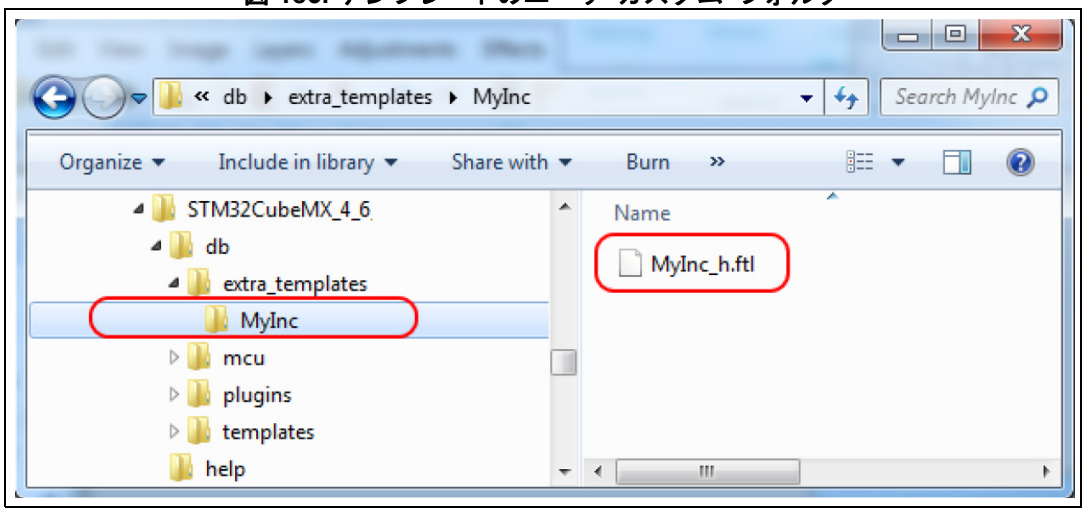
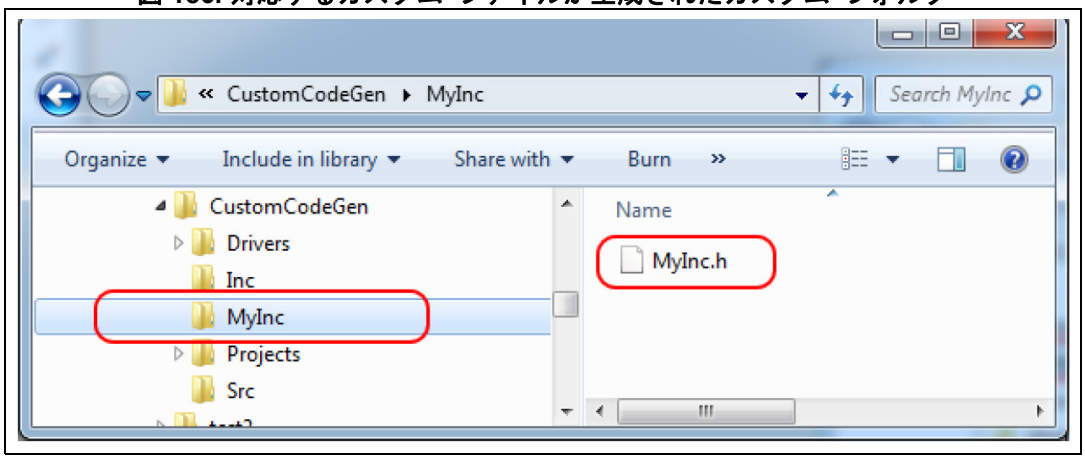


図 186. 対応するカスタム･ファイルが生成されたカスタム･フォルダ




## 6.4 C プロジェクト生成の追加設定

STM32CubeMX では、.extSettings ファイルを使用して、他のプロジェクト設定を指定できます。このファイルは、.ioc ファイルと同じプロジェクト･フォルダの同じ階層に配置する必要があります。

たとえば、外部ツールが STM32CubeMX を呼び出してプロジェクトを生成し、特定のプロジェクト設定を必要とする場合に、追加設定を使用できます。

### 使用可能なエントリと構文

すべてのエントリはオプションです。エントリは、ProjectFiles、Groups、Others の 3 つのカテゴリに分類されています。

- **[ProjectFiles]**：追加するインクルード･ディレクトリを指定するセクション。

  **構文**

  ```
  HeaderPath = <include directory 1 path>;< include directory 2 path >
  ```

  **例**

  ```
  HeaderPath=../../IIR_Filter_int32/Inc ;
  ```

- **[Groups]**：ファイルの新しいグループを作成し、グループにファイルを追加するセクション。

  **構文**

  ```
  <Group name> = <file pathname1>;< file pathname2>
  ```

  **例**

  ```
  Doc=$ PROJ_DIR$\..\readme.txt
  Lib=C:\libraries\mylib1.lib; C:\libraries\mylib2.lib;
  Drivers/BSP/MyRefBoard = C:\MyRefBoard\BSP\board_init.c;
  C:\MyRefBoard\BSP\board_init.h;
  ```

- **[Others]**：HAL モジュールを有効にして、プリプロセッサの define ステートメントを指定するセクション。

  – プリプロセッサの define ステートメントの有効化

  プリプロセッサの define ステートメントは、[Others] 行の後、次の構文で指定できます。

  **構文**

  ```
  Define = <define1_name>;<define2_name>
  ```

  **例**

```
Define= USE_STM32F429I_DISCO
```

– 生成される stm32f4xx_hal_conf.h での HAL モジュールの有効化

HAL モジュールは、[Others] 行の後、次の構文で有効化できます。

**構文**

```
HALModule = <ModuleName1>; <ModuleName1>;
```

**例**

```
HALModule=I2S;I2C
```

### .extSettings ファイルの例と生成された結果

例として、STM32CubeMX の［Board Selector］で STM32F429I-DISCO ボードを選択して、新しいプロジェクトを作成します。**［Project Manager］**ビューの［Project］タブで EWARM ツールチェーンを選択します。このプロジェクトを、MyF429IDiscoProject として保存します。プロジェクト･フォルダに、生成された .ioc ファイルと並んで、次の内容を持つ .extSettings テキスト･ファイルが配置されます。

### [Groups]

```
Drivers/BSP/STM32F429IDISCO=C:\Users\frq09031\STM32Cube\Repository\STM32Cube_FW_F4_V1.14.0\Drivers\BSP\STM32F429I-Discovery\stm32f429i_discovery.c;
C:\Users\frq09031\STM32Cube\Repository\STM32Cube_FW_F4_V1.14.0\Drivers\BSP\STM32F429I-Discovery\stm32f429i_discovery.h
Lib=C:\Users\frq09031\STM32Cube\Repository\STM32Cube_FW_F4_V1.14.0\Middlewares\Third_Party\FreeRTOS\Source\portable\IAR\ARM_CM4F\portasm.s
Doc=$PROJ_DIR$\..\readme.txt
```

### [Others]

```
Define = USE_ STM32F429I_DISCO
HALModule = UART;SPI
```

プロジェクト生成時にこのファイルが存在する場合は次の更新がトリガされます。

- EWARM フォルダのプロジェクト MyF429IDiscoProject.ewp ファイル（図 187 を参照）。
- プロジェクトの Inc フォルダの stm32f4xx_hal_conf.h ファイル（図 188 を参照）。
- 図 189 と図 190 に示すような、EWARM ユーザ･インタフェースのプロジェクト･ビュー。

図 187. プリプロセッサの define ステートメントを記述して
プロジェクトの .ewp ファイル（EWARM IDE）を更新

```
<settings>
  <name>ICCARM</name>
  <archiveVersion>2</archiveVersion>
  <data>
    <version>28</version>
    <wantNonLocal>1</wantNonLocal>
    <debug>1</debug>
    <option>
      <name>CCDefines</name>
      <state>USE_HAL_DRIVER</state>
      <state>STM32F429xx</state>
      <state>USE_STM32F429I_DISCO</state>
    </option>
```

図 188. stm32f4xx_hal_conf.h ファイルを更新して、選択したモジュールを有効化

stm32f4xx_hal_conf.h

```
/* #define HAL_RTC_MODULE_ENABLED    */
/* #define HAL_SAI_MODULE_ENABLED    */
/* #define HAL_SD_MODULE_ENABLED     */
/* #define HAL_MMC_MODULE_ENABLED    */
#define HAL_SPI_MODULE_ENABLED
/* #define HAL_TIM_MODULE_ENABLED    */
#define HAL_UART_MODULE_ENABLED
/* #define HAL_USART_MODULE_ENABLED    */
/* #define HAL_IRDA_MODULE_ENABLED    */
/* #define HAL_SMARTCARD_MODULE_ENABLED    */
/* #define HAL_WWDG_MODULE_ENABLED    */
/* #define HAL_PCD_MODULE_ENABLED    */
/* #define HAL_HCD_MODULE_ENABLED    */
```

図 189. EWARM IDE でグループに新しいグループと新しいファイルを追加

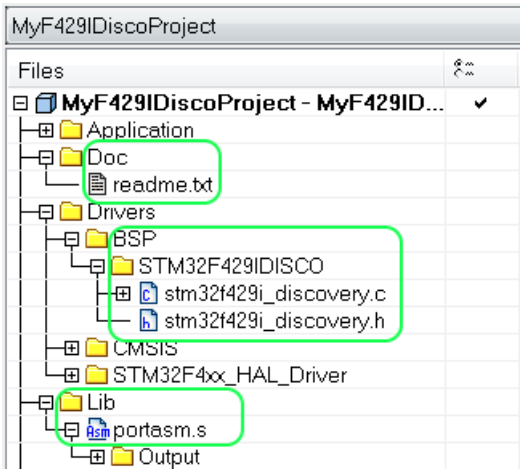
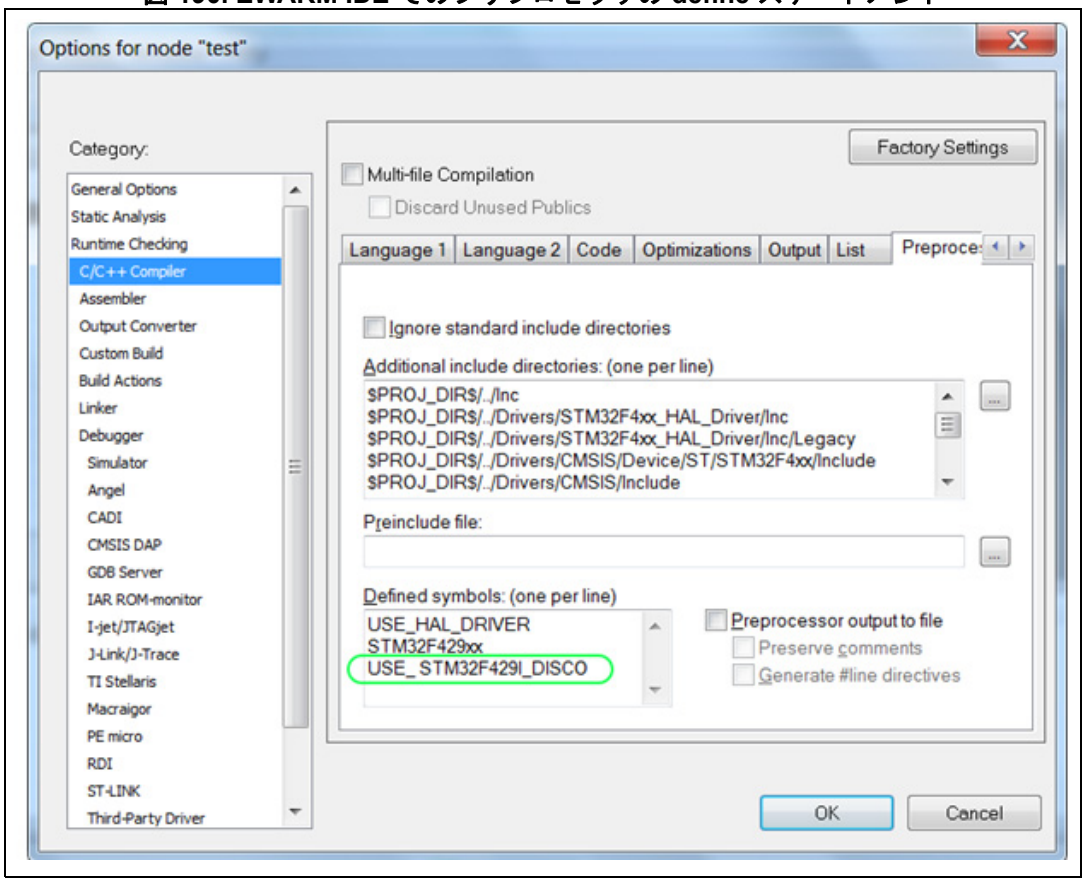


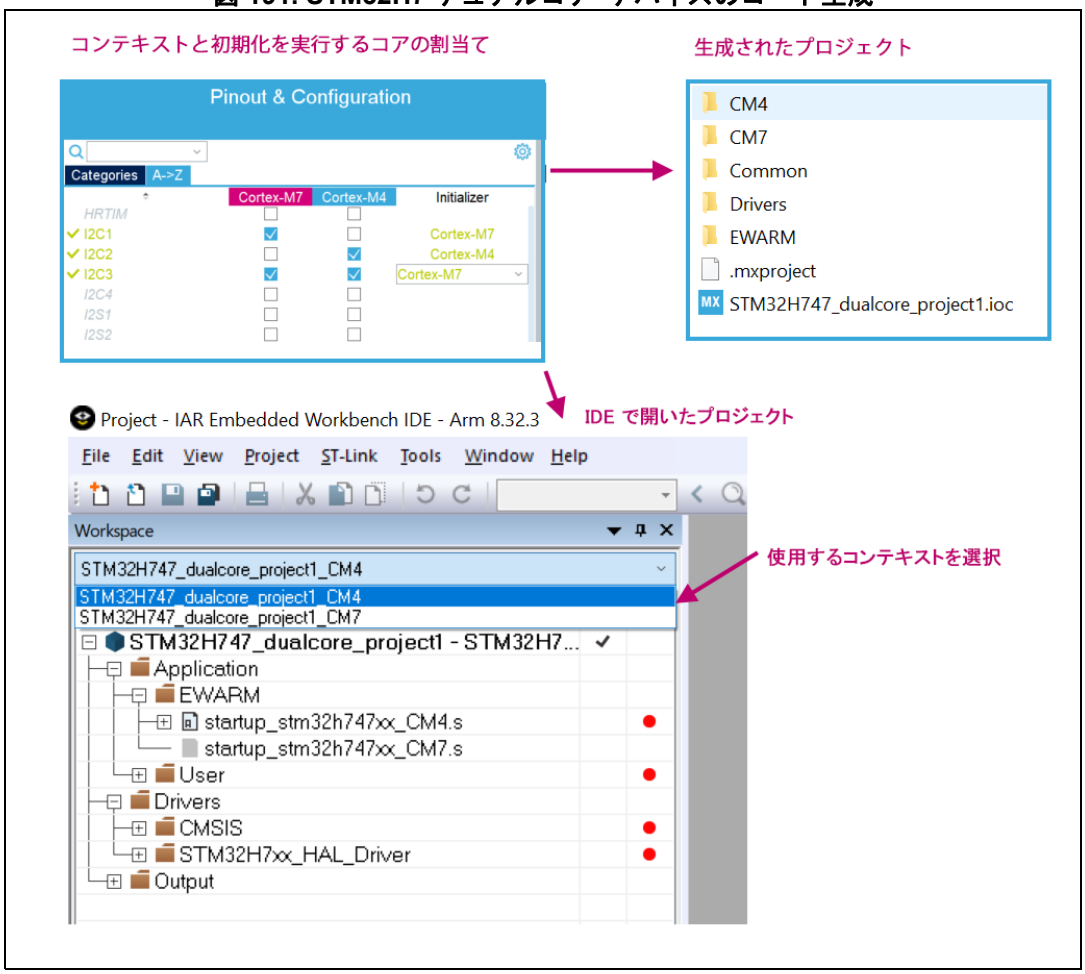
図 190. EWARM IDE でのプリプロセッサの define ステートメント



# 7　デュアルコア·マイクロコントローラのコード生成（STM32H7 デュアルコア製品ラインのみ）

Arm Cortex-M デュアルコア製品を使用する場合、STM32CubeMX は、ユーザ·インタフェースでのコンテキストの割当てと初期化方法の選択に従って、2 つのコア両方のコードを自動的に生成します（詳細はセクション 4.6 : STM32H7 デュアルコア製品ラインの［Pinout & Configuration］ビューを参照）。

**図 191. STM32H7 デュアルコア·デバイスのコード生成**




## 生成される初期化コード

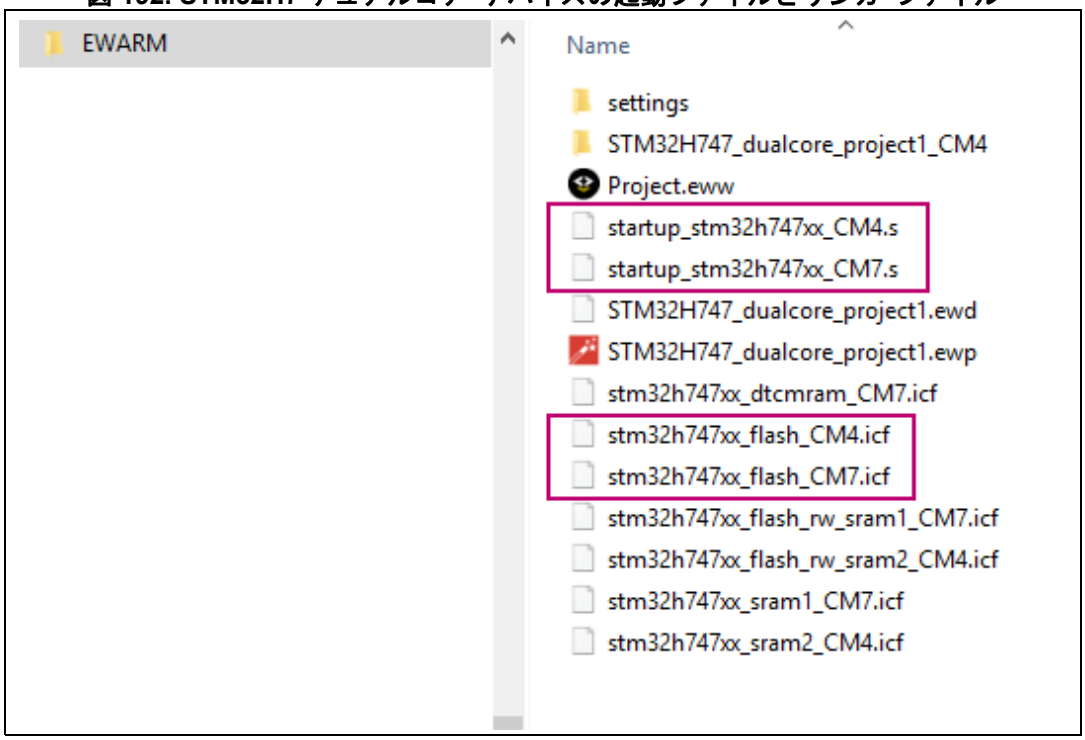
コードは、CM4、CM7、Common の各フォルダに生成されます。Common フォルダには、クロック·ツリー設定を収めた system_stm32h7xx.c が保存されます。

ペリフェラルまたはミドルウェアを両方のコンテキストに割り当てた場合、両方のコンテキストに対して MX_<名前>_init 関数が生成されますが、これを呼び出すのは初期化を実行する側のコアだけです。

### 生成される起動ファイルとリンカ･ファイル

プロジェクトの各設定（_M4 または _M7）には、末尾にそれぞれ _M4 と _M7 が付加された起動ファイルとリンカ･ファイルが生成されます。

**図 192. STM32H7 デュアルコア･デバイスの起動ファイルとリンカ･ファイル**




### 生成されるブート･モード･コード

現在、STM32CubeMX でサポートしているブート･モードは 1 つのみなので、両方の ARM Cortex-M コアが同時にブートします。

将来、次の別のブート･モードも導入され、［Project Manager］ビューからプロジェクトのオプションとして選択できるようになる予定です。

- ARM Cortex-M7 コアがブートして、Arm Cortex-M4 がゲートされるモード
- ARM Cortex-M4 コアがブートして、Arm Cortex-M7 がゲートされるモード
- Flash から第 1 のコアがブートし、そのコアによって 第 2 のコアのコードが SRAM に読み込まれて、第 2 のコアのブートが有効になるモード

STM32CubeMX では、STM32CubeH7 マイクロコントローラ･パッケージに付属するテンプレート･ファイルが参考として使用されます。

# 8 TrustZone® を有効にしたコード生成（STM32L5 シリーズのみ）

STM32CubeMX の［Project Manager］ビューでは、プロジェクト生成のオプションをすべて使用できます。

ただし、ツールチェーンの選択は、Cortex®-M33 コアをサポートする、以下の IDE とコンパイラに限定されます。

- EWARM v8.32 以上
- MDK-ARM v5.27 以上（ARM コンパイラ 6）
- STM32CubeIDE（GCC v4.2 以上）

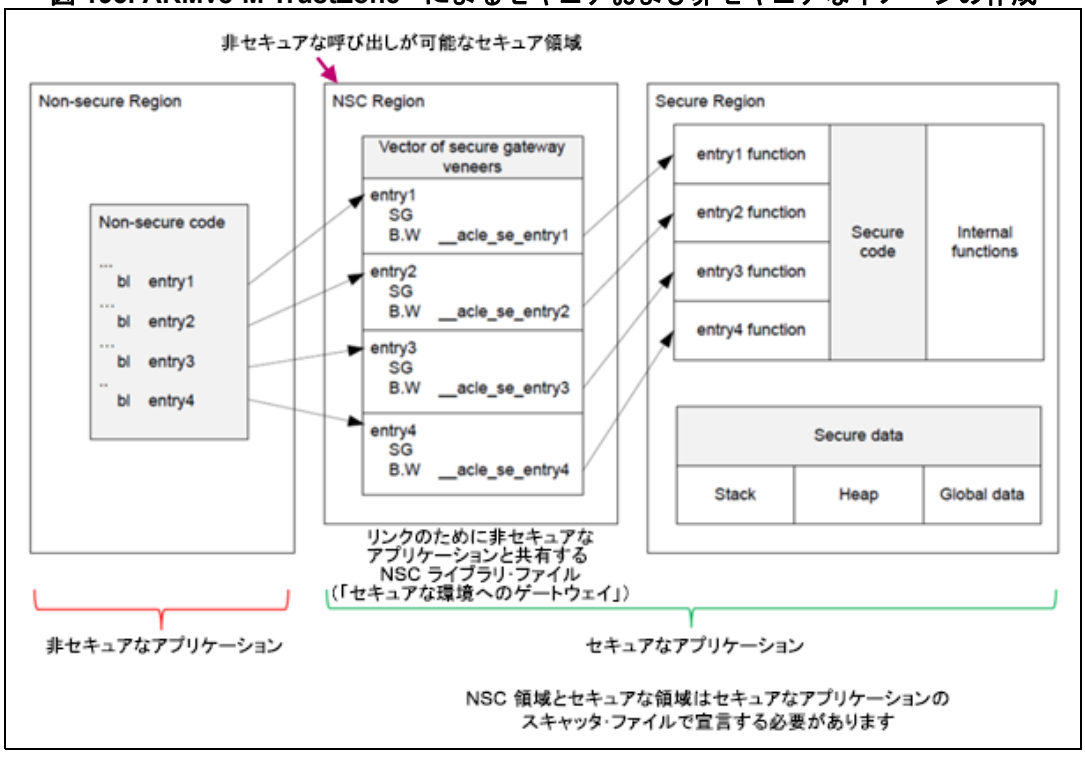
STM32CubeMX では、製品の選択時に TrustZone® を有効にするかどうかを選択する必要があります。

- TrustZone® を有効にした場合、STM32CubeMX によってセキュアと非セキュアな 2 つの C プロジェクトが生成されます。コンパイル後は、コンテキストごとに 1 つ、合計 2 つのイメージをダウンロードできます。
- TrustZone® を無効にした場合は、TrustZone®をサポートしていない他の製品と同様に、１つの非セキュアな C プロジェクトが生成されます。

## 特異性

TrustZone® を有効にする場合、セキュアと非セキュアなイメージを確実に構築できるように、プロジェクト生成を調整する必要があります。

**図 193. ARMv8-M TrustZone® によるセキュアおよび非セキュアなイメージの作成**



プロジェクトで TrustZone® を有効にすると、STM32CubeMX は次の 3 つのフォルダを作成します。

- 非セキュアなコード用の NonSecure
- セキュアなコード用の Secure
- 非セキュアな呼び出しが可能な領域用の Secure_nsclib

図 194（TZ_BasicStructure_project_inCubeIDE.png を使用）と 図 195（STM32L5_STM32CubeMX_Project_settings_inCubeIDE.png を使用）を参照してください。

**図 194. STM32L5 で TrustZone® を有効にしたプロジェクトの［Project Explorer］ビュー**

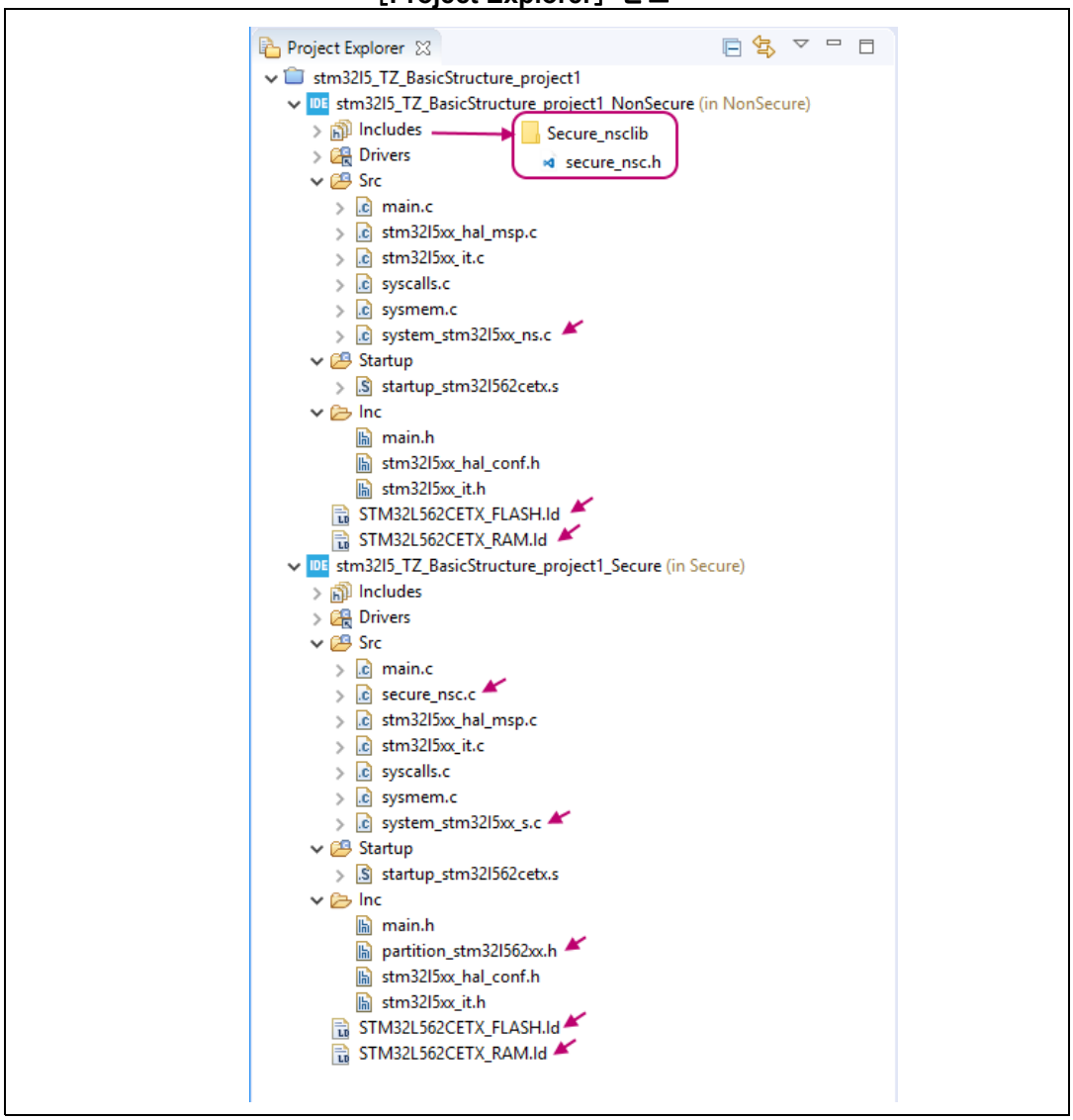
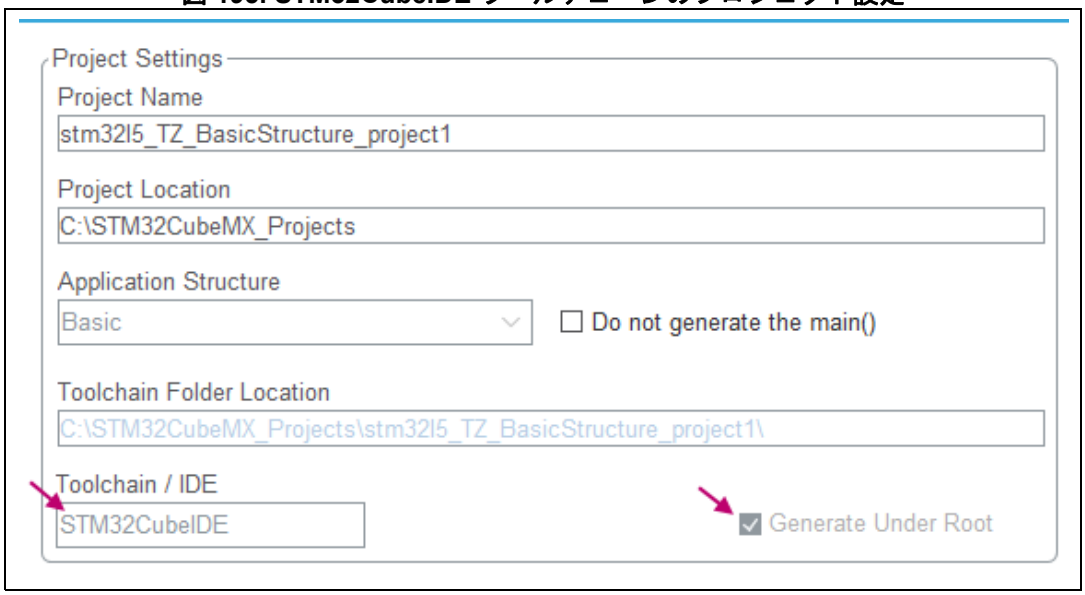


図 195. STM32CubeIDE ツールチェーンのプロジェクト設定




STM32CubeMX は固有なファイルも生成します。その詳細を表 22 に示します。

表 22. TrustZone® を有効にした場合に生成されるファイル

| ファイル | フォルダ | 詳細 |
|---|---|---|
| 製品コアのセキュアな部分と非セキュアな部分を分割する .h 「テンプレート」ファイル<br>例：partition_stm32l552xx.h | Secure | CMSIS CORE V5.3.1 の partition_ARMCM33.h テンプレートに基づいた、ARMCM33 のセキュアなゾーンと非セキュアなゾーン用の初期設定。<br>セキュリティ属性ユニット（SAU: Security Attribution Unit）の CTRL レジスタ、スリープと例外処理の設定動作、浮動小数点ユニット、割込みターゲットを初期化します。 |
| secure_nsc.h ファイル | Secure_nsclib | 非セキュアな呼び出しが可能な API のリストをユーザが記述する必要があります。<br>STM32L5Cube 組込みソフトウェア･パッケージには参考としてテンプレートが付属しています。Templates\TrustZone®\Secure_nsclib フォルダに保存されています。 |
| System_stm32l5xx_s.c | Secure | システムがセキュリティを実装する場合に、セキュアなアプリケーションで使用する、CMSIS Cortex-M33 デバイスのペリフェラル･アクセス･レイヤ･システムのソース･ファイル |

表 22. TrustZone® を有効にした場合に生成されるファイル （続き）

| ファイル | フォルダ | 詳細 |
|---|---|---|
| System_stm32l5xx_ns.c | NonSecure | システムがセキュリティを実装する場合に、非セキュアなアプリケーションで使用する、CMSIS Cortex-M33 デバイスのペリフェラル·アクセス·レイヤ·システムのソース·ファイル |
| STM32L562CETX_FLASH<br>STM32L562CETX_RAM<br>または<br>STM32L552CETX_FLASH<br>STM32L552CETX_RAM | Secure、NonSecure | セキュアおよび非セキュアなメモリ·レイアウト用のリンカ·ファイル。<br>ファイル拡張子と命名規則は、次のとおりです。<br>– .icf （EWARM）<br>– .sct （MDK-ARM）<br>– .ld （GCC コンパイラ·ツールチェーン） |

# 9 デバイス･ツリーの生成（STM32MP1 シリーズのみ）

Linux のデバイス･ツリーは、non-discoverable ハードウェアを記述する方法として使用されます。ST マイクロエレクトロニクスでは、DDR 設定をはじめとするすべてのプラットフォーム設定データにデバイス･ツリーを広く使用しています。

Linux 開発者は、デバイス･ツリーのソース･ファイル（dts）を手動で編集できますが、これに代わる方法として STM32CubeMX では部分的なデバイス･ツリー生成サービスを提供し、工数の削減や初心者の負担軽減を図っています。ボード･レベルの設定に対応するデバイス･ツリーを部分的に生成することを意図しています。「部分的に」とは、全体（ボード規模）のデバイス･ツリーは生成されず、通常であれば膨大な工数を必要とし、コンパイル･エラーや機能障害の原因となりやすい、次のような主要セクションのみが生成されるということです。

- フォルダ構造、およびフォルダへのファイルの配分
- dtsi とヘッダのインクルード
- pinCtrl とクロック生成
- システムオンチップ（SoC）デバイスのノード配置
- マルチコア関連の設定（Etzpc バインド、リソース･マネージャのバインド、ペリフェラル割当て）

## 9.1 デバイス･ツリーの概要

どのようなソフトウェアも、適切に動作するにはその実行場所であるプラットフォームのハードウェア記述を取得する必要があります。この記述として、CPU の種類、メモリ･サイズ、ピン設定などがあります。現在の Linux カーネルと U-boot では、そのような non-discoverable なハードウェア記述を、独立したバイナリであるデバイス･ツリー BLOB （dtb）で定義します。デバイス･ツリー BLOB は、OpenSTLinux ディストリビューションで提供される dtc コンパイラを使用して、デバイス･ツリー･ソース･ファイル（dts）からコンパイルされます。

デバイス･ツリーの構造は、2 つのデバイス･ツリー･ソース･インクルード･ファイル（.dtsi）をインクルードするボード･レベル･ファイル（.dts）から構成されます。この 2 つのインクルード･ファイルは、SoC レベルのファイルと、ピンの多重化設定を記述する -pinctrl ファイルです。

デバイス･ツリーの構造は、C 言語の階層化された構造体にきわめて類似しており、"root" （/）を最上位として、その下のサブノードに該当する各ペリフェラルで詳細な記述を階層構造で提供します（図 196、197、198 を参照）。

STM32CubeMX で生成されるコードでは、ユーザ設定での必要性に応じて、SoC デバイス定義の完成や変更のためにオーバーロード･メカニズムが広く使用されます。

図 196. STM32CubeMX で生成された DTS – 抜粋 1

```
システムとボードの情報
    model = "STMicroelectronics custom STM32CubeMX board";
    compatible = "st,stm32mp157c-project2-mx", "st,stm32mp157";

      memory@c0000000 {
        ...
    };

    /* USER CODE BEGIN root */
    /* USER CODE END root */          ← ユーザ・カスタマイズ

    clocks {                          ← 完全なクロック設定
        clk_lsi: clk-lsi {
            #clock-cells = <0>;
            compatible = "fixed-clock";
            clock-frequency = <32000>;
            u-boot,dm-pre-reloc;
        };
        ...
    };

}; /*root*/

&pinctrl {                            ← GPIO を含むピン制御の設定
    u-boot,dm-pre-reloc;
    tim1_pins_mx: tim1_mx-0 {
        pins {
            pinmux = <STM32_PINMUX('A', 8, AF1)>, /* TIM1_CH1 */
                     <STM32_PINMUX('A', 9, AF1)>; /* TIM1_CH2 */
            bias-disable;
            drive-push-pull;
            slew-rate = <0>;
        };
    };
};
```

図 197. STM32CubeMX で生成された DTS – 抜粋 2

```
&m4_rproc{                            ← マルチコア管理
    recovery;

    m4_system_resources{
        status = "okay";

        /* USER CODE BEGIN m4_system_resources */
        /* USER CODE END m4_system_resources */
    };

    status = "okay";

    /* USER CODE BEGIN m4_rproc */
    /* USER CODE END m4_rproc */
};

&m4_timers1{                          ← Cortex-M4 のランタイムコンテキストへのペリフェラルの割当て
    pinctrl-names = "rproc_default", "rproc_sleep";
    pinctrl-0 = <&tim1_pins_mx>;
    pinctrl-1 = <&tim1_sleep_pins_mx>;
    status = "okay";

    /* USER CODE BEGIN m4_timers1 */
    /* USER CODE END m4_timers1 */
};
```

図 198. STM32CubeMX で生成された DTS – 抜粋 3

```
&timers2{                    右の各設定によるペリフェラル·ノード構造   PinCtrl 設定
    status = "okay";                                                  Status 設定
                                                                      ユーザ·カスタマイズ
    /* USER CODE BEGIN timers2 */
    /* USER CODE END timers2 */

    pwm{
        pinctrl-names = "default", "sleep";
        pinctrl-0 = <&tim2_pwm_pins_mx>;
        pinctrl-1 = <&tim2_pwm_sleep_pins_mx>;
        status = "okay";

        /* USER CODE BEGIN timers2_pwm */
        /* USER CODE END timers2_pwm */
    };
};

/* USER CODE BEGIN dts_addons */
/* USER CODE END dts_addons */
```

詳細は、https://elinux.org より入手可能な Thomas Petazzoni 著の『Device Tree for Dummies』を参照してください。

STM32MP1 シリーズのデバイス·ツリーの仕様については、ST の Wiki（https://wiki.st.com/stm32mpu）を参照してください。

## 9.2 STM32CubeMX のデバイス·ツリー生成

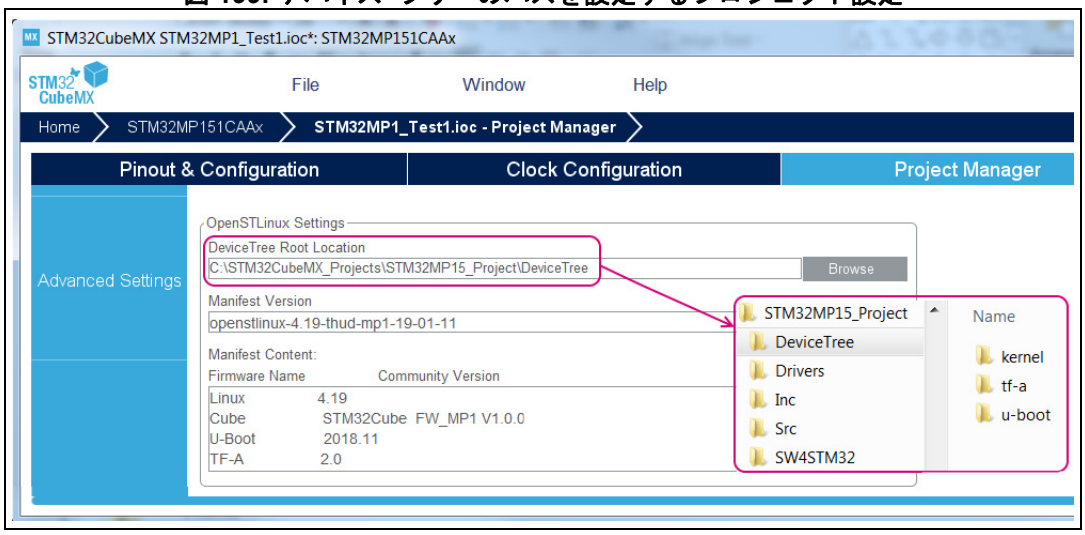
STM32MP1 シリーズでは、STM32CubeMX のコード生成機能が拡張され、サポート対象のファームウェアをターゲットとする以下のデバイス·ツリー（DT）を生成できるようになりました。

- TF-A と SP_min の両方を設定するための単一の DT
- U-Boot 設定用の DT
- Linux カーネル設定用の DT

DTS の生成機能には、同じ GENERATE CODE ボタンからアクセスできます。

DT の生成パスは、［Project Manager］ビューの［Advanced Settings］タブにある［OpenSTLinux Settings］で設定できます（図 199 参照）。STM32CubeMX は、デバイス･ツリーごとにデバイス･ツリー･ソース（DTS）ファイルを生成します。

**図 199. デバイス･ツリーのパスを設定するプロジェクト設定**




デバイス･ツリー構造は、以下より構成されます。

- すべてのクロック･ツリー
- すべてのピン制御
- すべてのマルチコア参照定義
- 一連のデバイス･ノードとサブノード
- ブート可能なデバイス･ツリーをすべて記述できるユーザ･セクション（次回の生成でも失われません）

生成される DTS ファイルにはユーザ設定が反映されます。このような設定として、ランタイムコンテキストやブート･ローダへのペリフェラルの割当てや、クロック･ツリーの設定などがあります。

STM32CubeMX の DT 生成機能では、さまざまな DT 間の一貫性が確実に保持されます。さらに、TF-A デバイス･ツリーと U-Boot デバイス･ツリーに DDR 設定ファイルも生成されます。

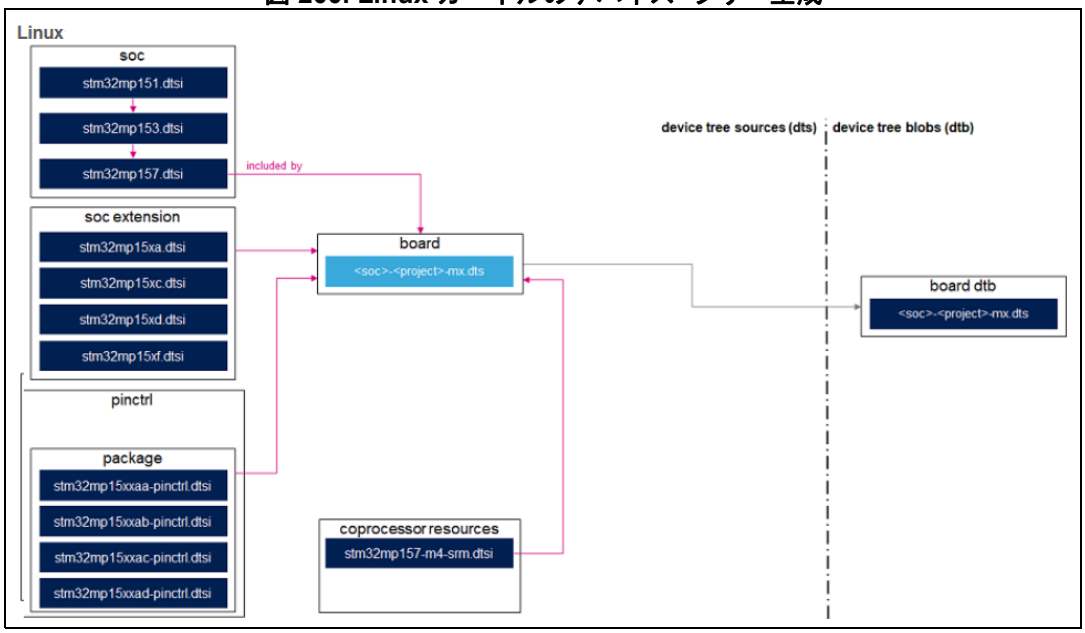
これらのファイルおよびそれらがインクルードするファイルがコンパイルされ、ターゲットとなるファームウェアのデバイス･ツリー BLOB が作成されます。

### 9.2.1 Linux カーネルのデバイス･ツリー生成

Linux の場合、STM32CubeMX ではボード規模のファイルのみが生成されます。このファイルは "soc" ファイルと選択したパッケージに対応する "pinctrl" ファイルをインクルードします。

STM32CubeMX によって生成されるデバイス･ツリーのノードは、ユーザ･セクションの記述によって完成できます。これは、Linux カーネルのソース･コード のフォルダ Documentation/devicetree/bindings/ に用意されているデバイス･ツリー･バインドに従います。

図 200. Linux カーネルのデバイス･ツリー生成

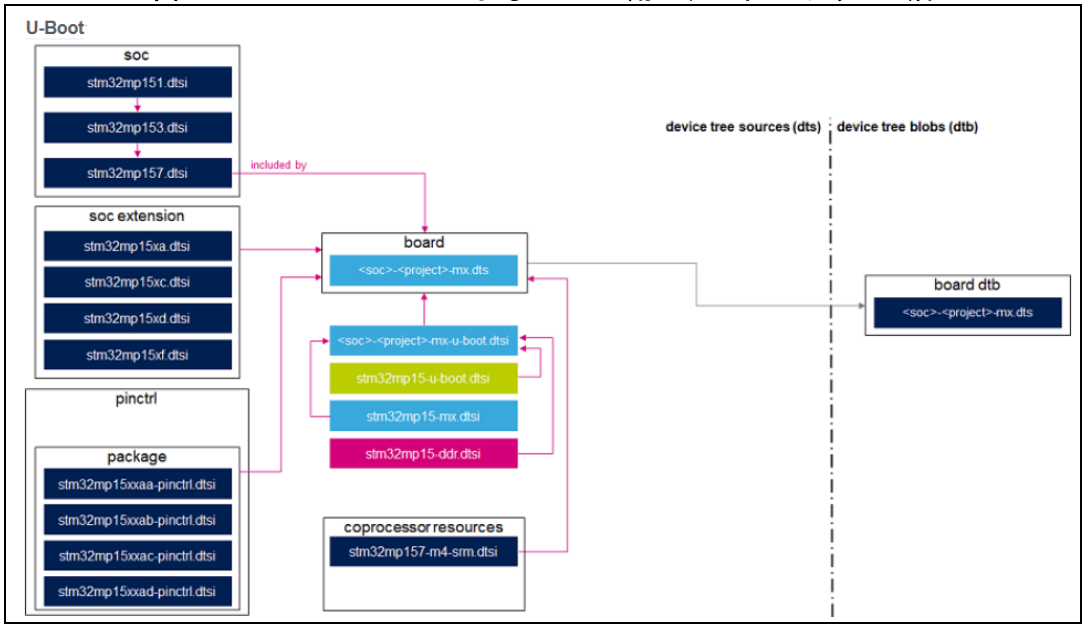



## 9.2.2 U-boot 用のデバイス･ツリー生成

STM32CubeMX は、U-Boot 用に Linux の dts ファイルのコピーを作成し、2 つの新しいファイルとして "ddr" の設定用ファイルと U-Boot のアドオン用ファイルを作成します。その主な機能は必要に応じて "u-boot,dm-pre-reloc" プロパティを使用することです。

STM32CubeMX によって生成されるデバイス･ツリーのノードは、ユーザ･セクションの記述によって完成できます。これは、U-Boot のソース･コード のフォルダ Documentation/devicetree/bindings/ に用意されているデバイス･ツリー･バインドに従います。

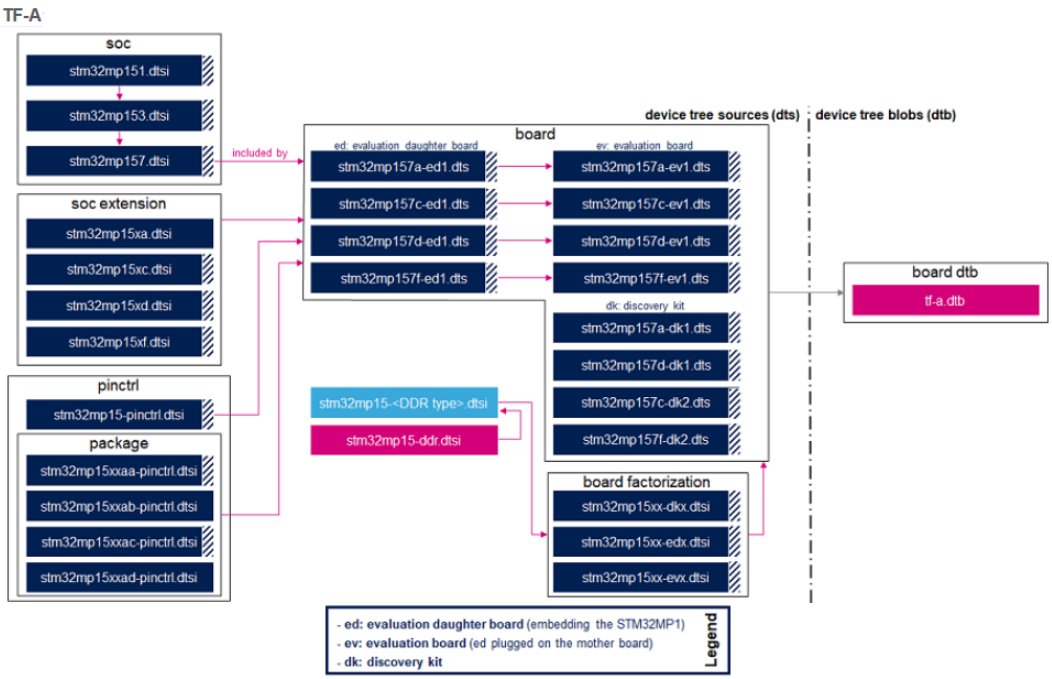
図 201. STM32CubeMX による U-boot 用のデバイス･ツリー生成



### 9.2.3 TF-A 用のデバイス･ツリー生成

STM32CubeMX は、容量を節約するために Linux のボード規模 dts ファイルの軽量化バージョンである TF-A 用のボード規模 dts ファイルを生成します。このファイルは、TF-A に付属する dtsi ファイルとして "soc" 側と "pinctrl" 側で軽量化済みのファイルをインクルードします。U-Boot 用に生成された "ddr" 設定ファイルが TF-A でもそのまま再利用されます。

**図 202. STM32CubeMX による TF-A 用のデバイス･ツリー生成**




Note: The hashed areas on Linux device tree files represent the fact that these files are partly copied in TF-A.

STM32CubeMX によって生成されるデバイス･ツリーのノードは、ユーザ･セクションの記述によって完成できます。これは、TF-A のソース･コード のフォルダ Documentation/devicetree/bindings/ に用意されているデバイス･ツリー･バインドに従います。

# 10 CMSIS-Pack 規格に基づく追加ソフトウェア・コンポーネントのサポート

CMSIS-Pack 規格は、ソフトウェア・コンポーネント、デバイス・パラメータ、および評価ボード・サポートの配信メカニズムを記述しています。

XML ベースのパッケージ記述（pdsc）ファイルは、ソフトウェア・パッケージ（ファイルのコレクション）の内容を記述しています。この記述では、ソースコード、ヘッダファイル、ソフトウェア・ライブラリ、ドキュメント、およびソースコード・テンプレートを扱っています。ソフトウェア・パッケージは、ファイル・コレクション一式と pdsc ファイルで構成され、ZIP 形式で提供されます。ソフトウェア・パッケージをインストールすると、そこに収録されたすべてのファイルに開発ツールからアクセスできます。

ソフトウェア・コンポーネントとは、ソース・モジュール、ヘッダファイル、設定ファイル、およびライブラリの集合です。ソフトウェア・コンポーネントを収めるパッケージには、サンプル・プロジェクトとユーザ・コード・テンプレートも追加できます。

詳細については、ウェブ・サイト http://www.keil.com を参照してください。

STM32CubeMX は、ソフトウェア・パッケージで提供される、3rd パーティ製および ST マイクロエレクトロニクス製の他の組込みソフトウェア・ソリューションをサポートします。STM32CubeMX では次の操作が可能です。

1. ソフトウェア・パッケージのインストールおよび更新の確認（セクション 3.4.5 : を参照）。
2. 現在のプロジェクトで使用するソフトウェア・コンポーネントの選択（セクション 4.13 : を参照）。選択したコンポーネントはツリー・ビューに表示されます（図 203 を参照）。
3. ツリー・ビューからのソフトウェア・コンポーネントの有効化（図 204 を参照）。コンテキスト・ヘルプを使用すると、選択したコンポーネントの詳細を確認できます。
4. ソフトウェア・コンポーネントの設定（図 204 を参照）。この機能は、STM32CubeMX の独自形式に従ったファイルで構成されているコンポーネントにのみ使用できます。
5. 選択したツールチェーンに適した C プロジェクトの生成（図 205 を参照）。
   a) ソフトウェア・コンポーネントのファイルが、自動的にプロジェクトにコピーされます。
   b) ソフトウェア・コンポーネントの設定と初期化のコードが自動的に生成されます。この機能は、STM32CubeMX の独自形式に従ったファイルで構成されているコンポーネントにのみ使用できます。

図 203. CMSIS-Pack ソフトウェア・コンポーネントの選択

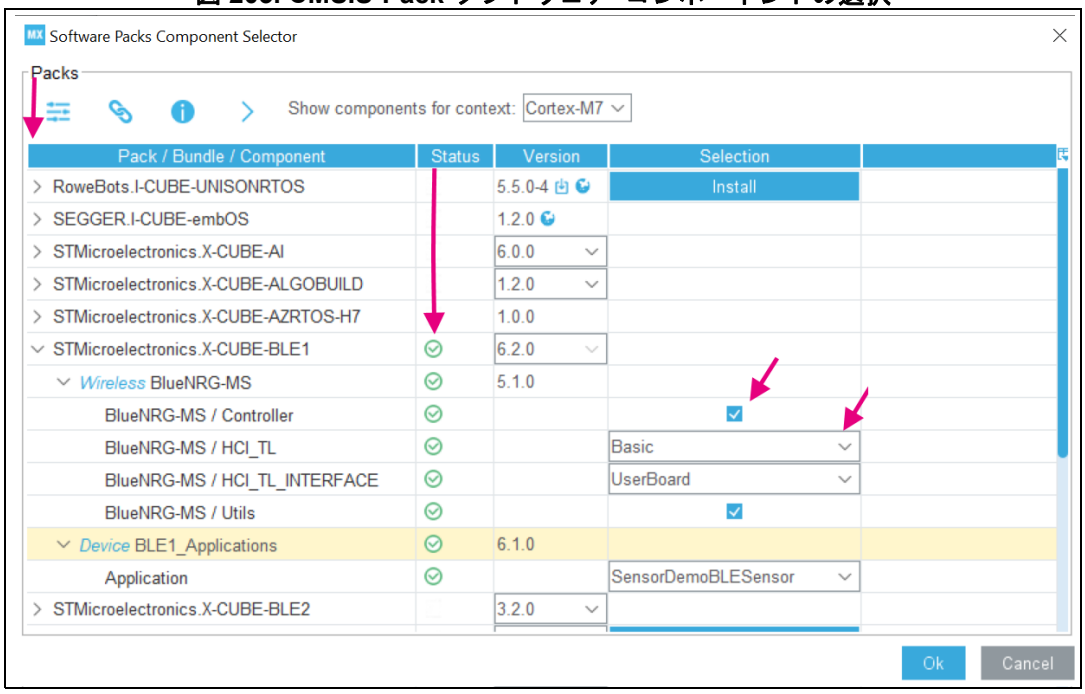



図 204. CMSIS-Pack ソフトウェア・コンポーネントの有効化と設定

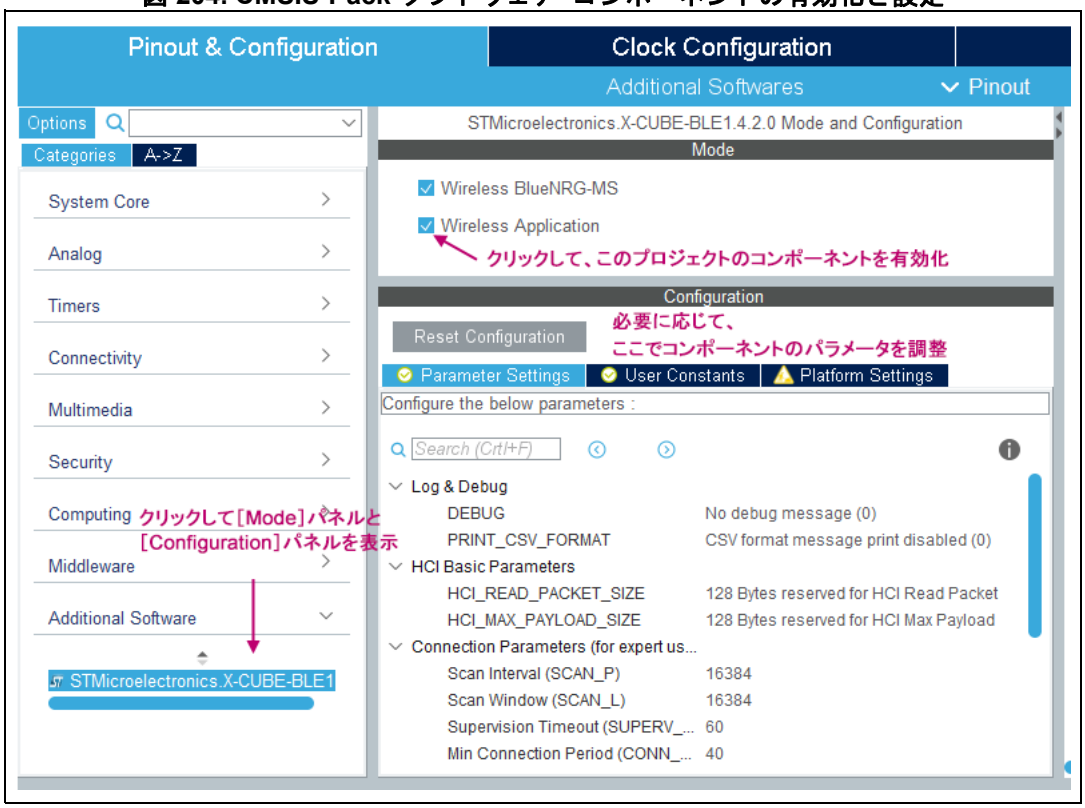
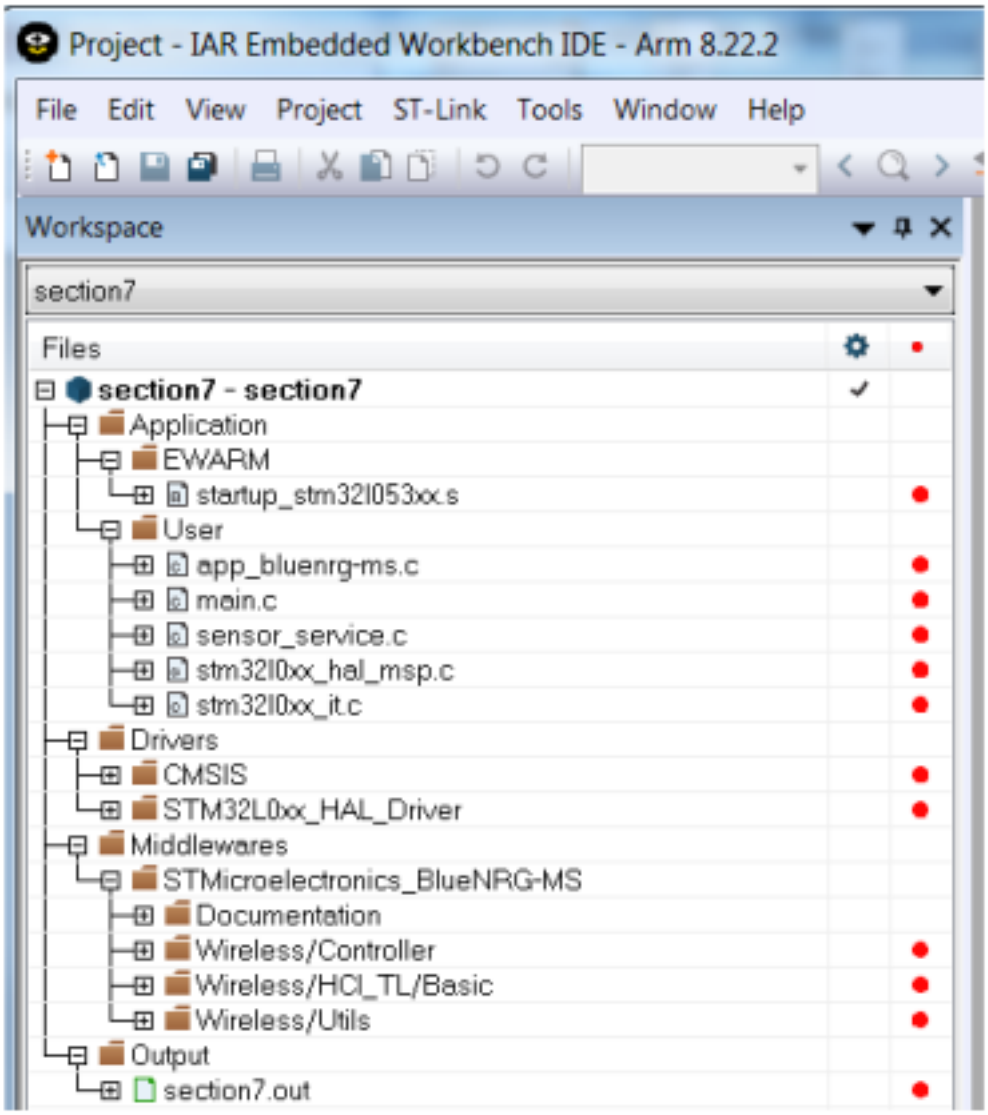


図 205. CMSIS-Pack ソフトウェア･コンポーネントから生成されたプロジェクト



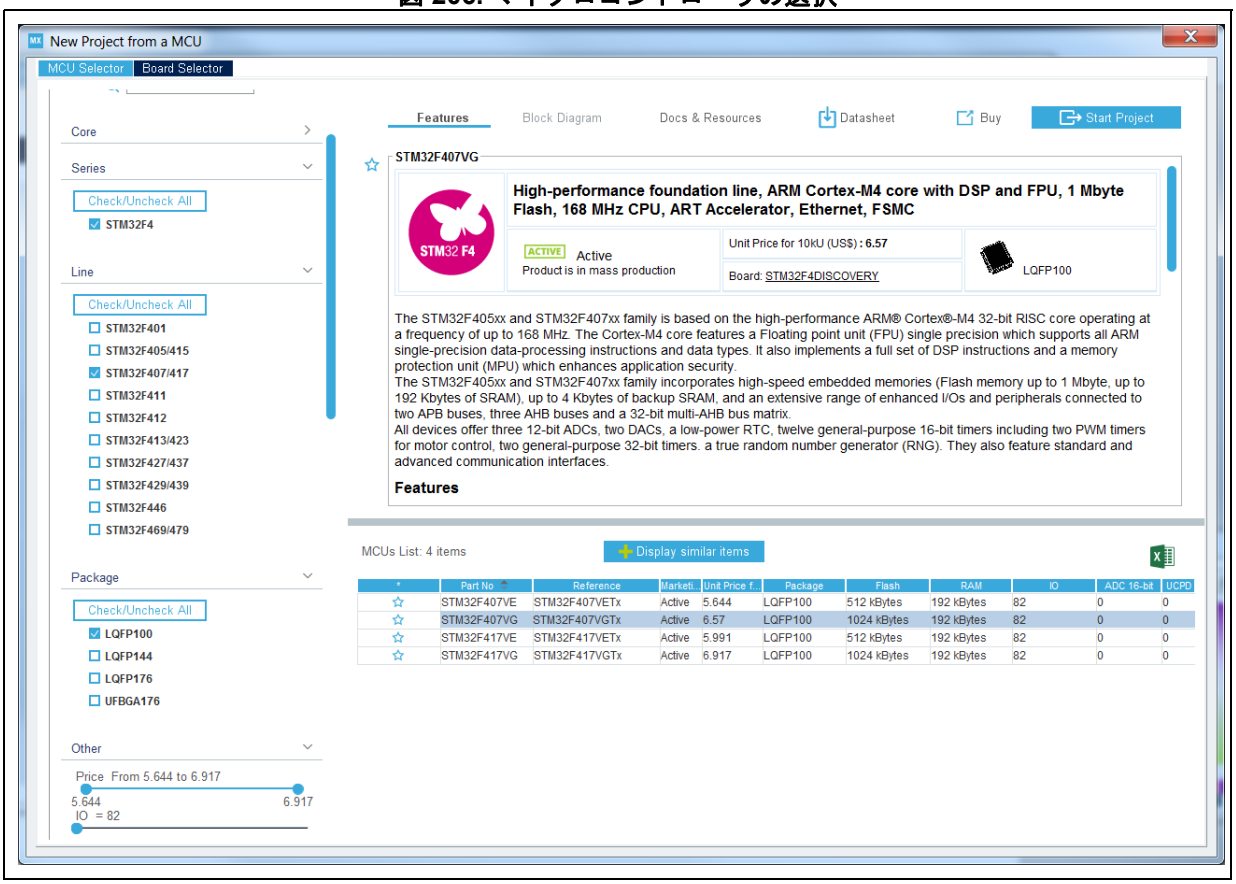
# 11 チュートリアル 1：ピン配置からプロジェクトの C コードの生成まで STM32F4 シリーズのマイクロコントローラを使用する場合

このセクションでは、設定と C コード生成のプロセスについて説明します。例として、STM32F4DISCOVERY ボード上で LED の点灯を切り替える簡単なアプリケーションを使用します。

## 11.1 新しい STM32CubeMX プロジェクトの作成

1. メイン·メニュー·バーから **[File]** → **[New project]** を選択するか、[Home] ページで **[New project]** を選択します。
2. **[MCU Selector]** タブを選択し、[Series] で [STM32F4]、[Lines] で [STM32F407]、および [Package] で [LQFP100] を選択して、STM32 のポートフォリオを絞り込みます（図 206 を参照）。
3. [MCUs List] から [STM32F407VGTx] を選択して **[OK]** をクリックします。

**図 206. マイクロコントローラの選択**




選択したマイクロコントローラ·データベースのデータが STM32CubeMX の各ビューに表示されます（図 207）。必要に応じ、**[Window]** → **[Outputs]** サブメニューの選択を解除して、下部の [MCUs Selection] ウィンドウを非表示にできます（図 208 を参照）。

図 207. ［MCUs Selection］を表示した［Pinout］ビュー

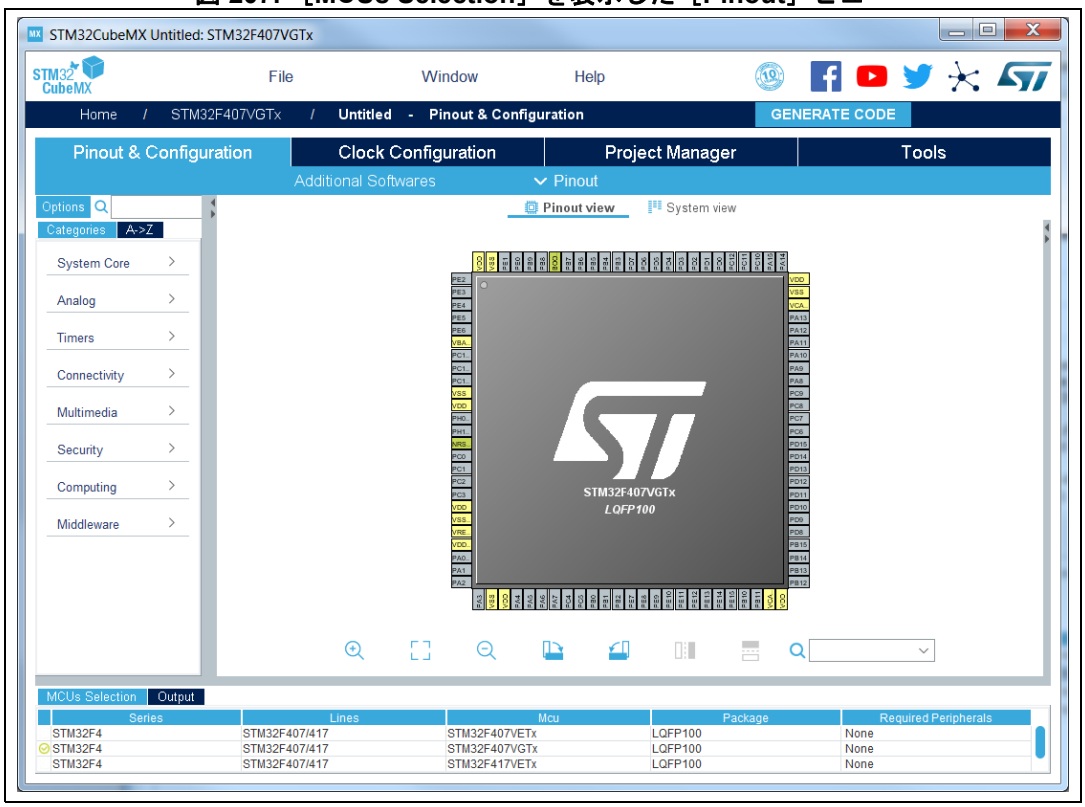
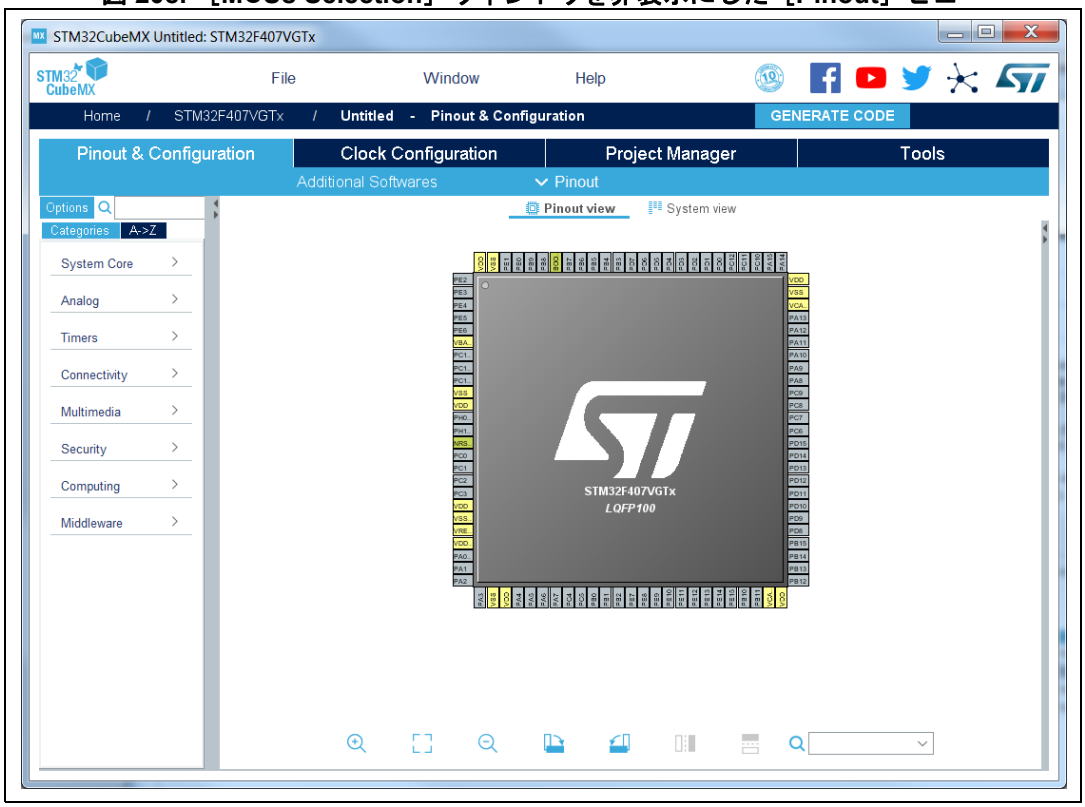



図 208. ［MCUs Selection］ウィンドウを非表示にした［Pinout］ビュー



## 11.2 マイクロコントローラのピン配置の設定

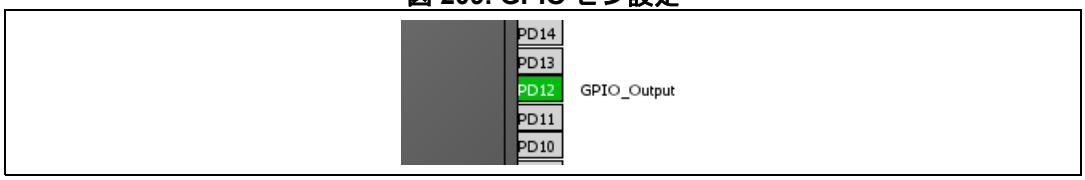
メニュー、高度なアクション、および競合の解決については、セクション 4：と付録 Aを参照してください。

1. STM32CubeMX では、デフォルトで［**Pinout**］ビューが表示されます。
2. デフォルトでは Keep Current Signals Placement はチェックされていないので、STM32CubeMX でペリフェラル機能を移動して、対応するペリフェラル･モードの最大数を設定できる最適なピン割り当てを見つけることができます。

   マイクロコントローラのピン設定は STM32F4DISCOVERY ボードに一致する必要があるので、STM32CubeMX で Keep Current Signals Placement をチェックして、ピンに対するペリフェラル機能の割当て（マッピング）を維持します。

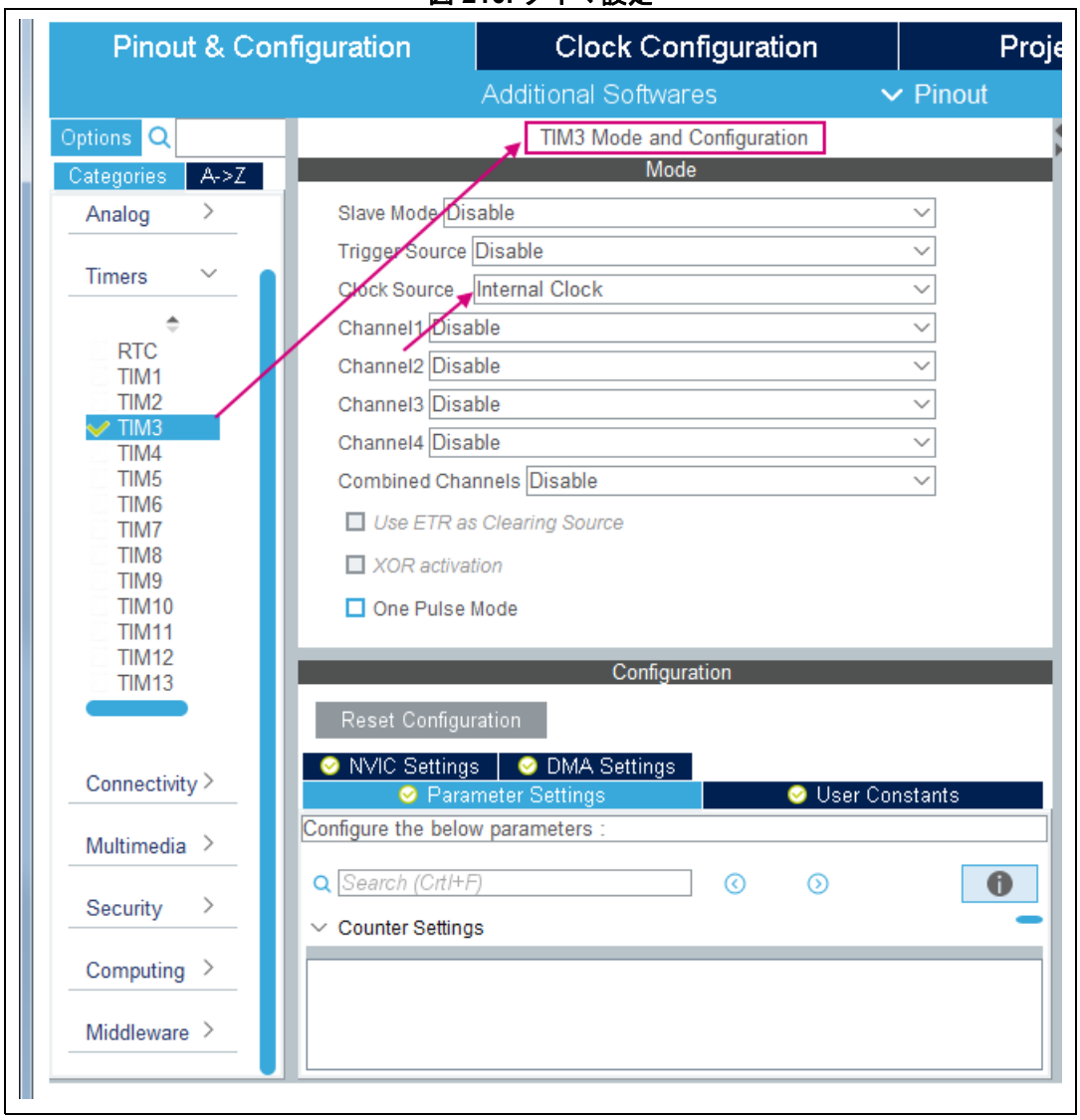
   この設定はユーザ設定として保存され、ツールを次に開くときや別のプロジェクトをロードするときに復元されます。
3. 必要なペリフェラルとペリフェラル･モードを選択します。
   a) ［**Pinout**］ビューで［PD12］を右クリックして、STM32F4DISCOVERY の緑色 LED に信号が出力されるように GPIO を設定し、次のようにピンに対して GPIO_output を選択します。

**図 209. GPIO ピン設定**



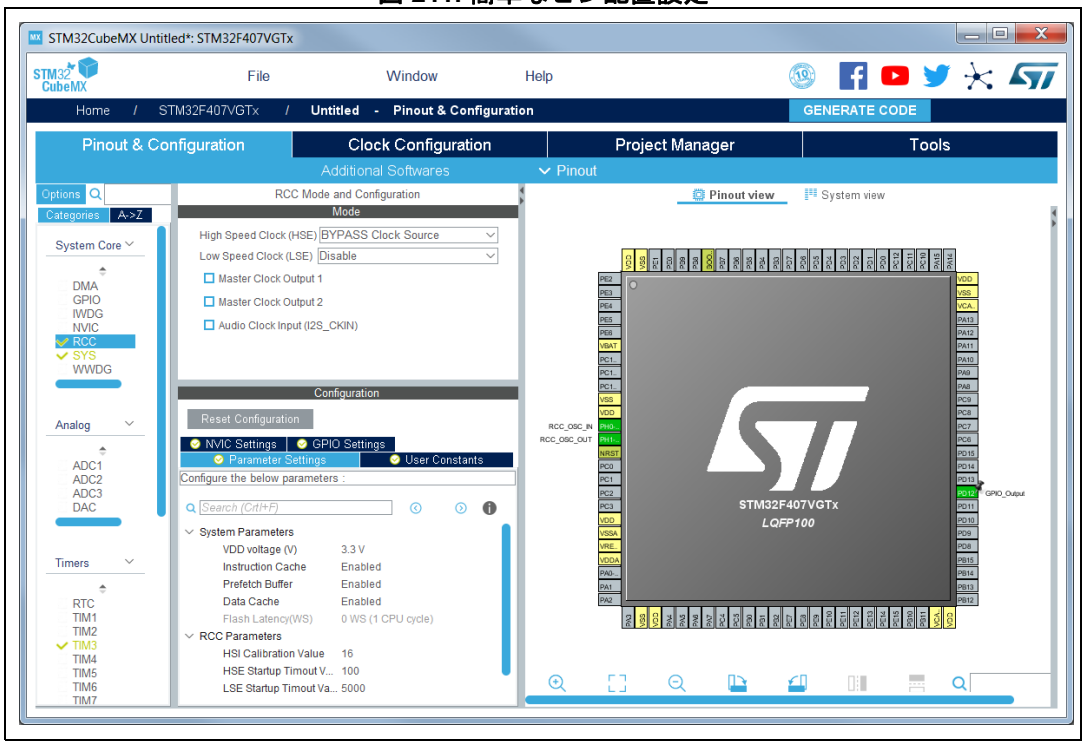
b) LED を切り替えるタイムベースとして使用するタイマを有効にします。それには、ペリフェラル·ツリーで［TIM3］の［Clock Source］として［Internal Clock］を選択します（図 210 を参照）。

**図 210. タイマ設定**



c) 外部オシレータをクロック･ソースとして使用できるように、RCC を設定することもできます（図 211 を参照）。

**図 211. 簡単なピン配置設定**



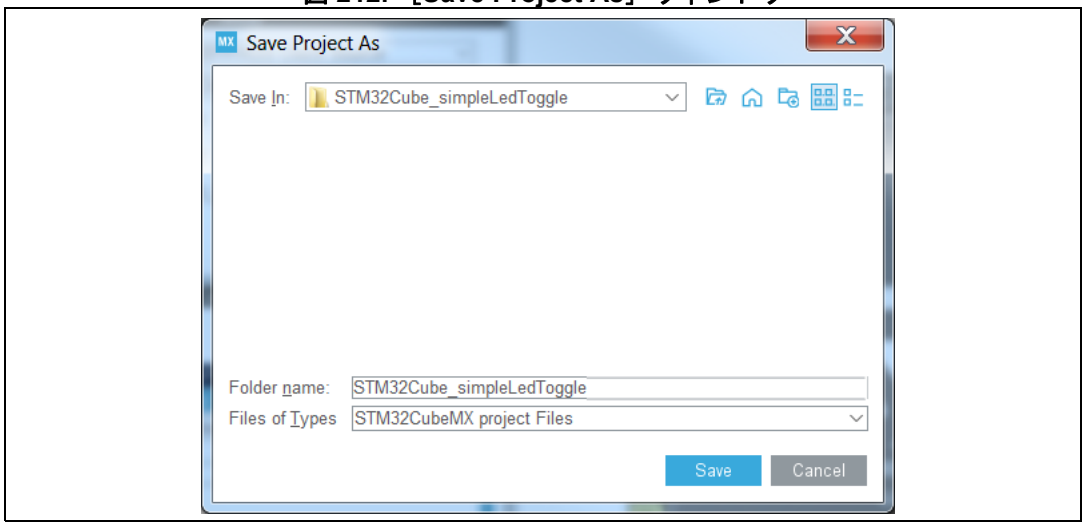

これで、この例のピン配置設定は完了です。

**注：** **STM32CubeMX 4.2 から、[Board selector] タブで ST ディスカバリ･ボードの設定を直接ロードすることによって、ピン配置設定をスキップできるようになりました。**

## 11.3 プロジェクトの保存

1. プロジェクトを保存するには をクリックします。
   初めて保存するときは、プロジェクトの保存先フォルダとファイル名を選択します。
   STM32CubeMX の設定ファイルであることを示すために、自動的に .ioc 拡張子が付加されます。

**図 212. ［Save Project As］ウィンドウ**




2. をクリックすると、別の名前または別の場所にプロジェクトが保存されます。

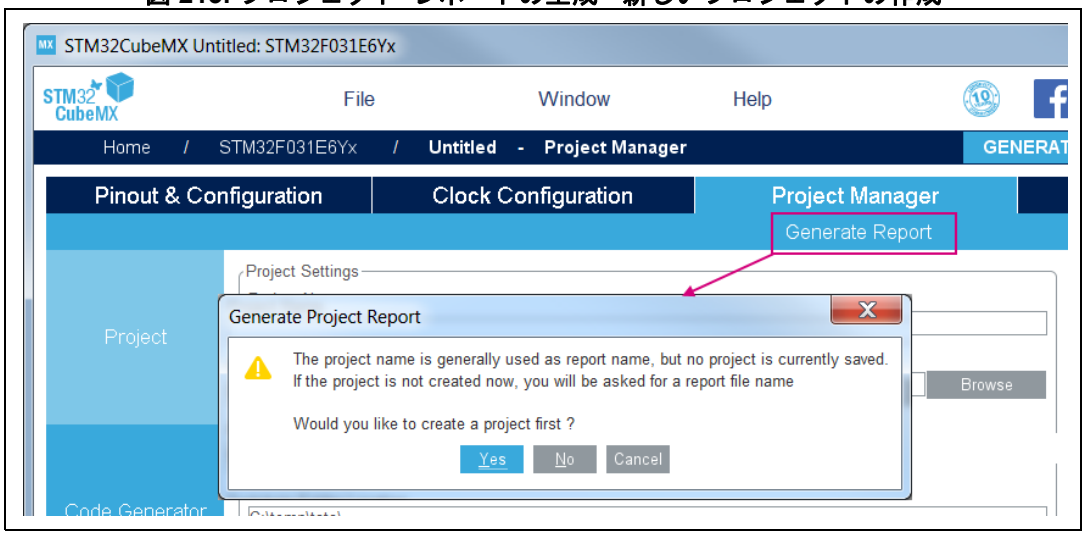
## 11.4 レポートの生成

設定中はいつでもレポートを生成できます。

1. をクリックすると、.pdf と .txt のフォーマットでレポートが生成されます。

   プロジェクト･ファイルをまだ作成していない場合は、先にプロジェクトを保存する必要があることを示す警告が表示され、プロジェクト名と保存先フォルダの指定が求められます（図 213 を参照）。プロジェクトの .ioc ファイルが作成されるほか、同じ名前の .pdf レポートと .txt レポートが生成されます。

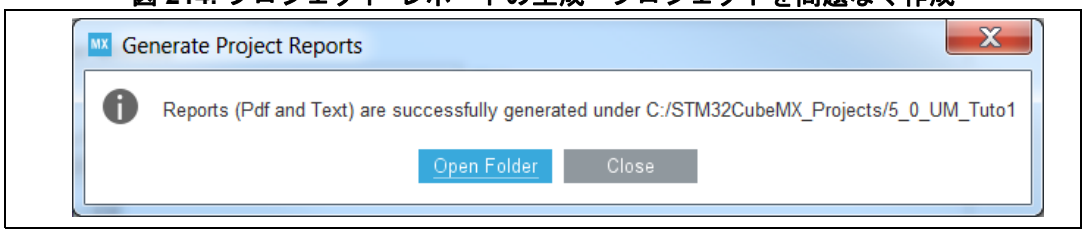
**図 213. プロジェクト･レポートの生成 - 新しいプロジェクトの作成**




［**No**］をクリックした場合、レポートのみの名前と場所の指定が求められます。

図 214 に示すように、動作が正常に終了すると確認メッセージが表示されます。

**図 214. プロジェクト･レポートの生成 - プロジェクトを問題なく作成**




2. Adobe Reader で .pdf レポートを開くか、任意のテキスト･エディタで .txt レポートを開きます。レポートには、プロジェクトのすべての設定とマイクロコントローラ設定がまとめられています。

## 11.5 マイクロコントローラのクロック･ツリーの設定

ここでは、STM32F4 マイクロコントローラに基づいてアプリケーションに必要なクロックの設定方法について説明します。

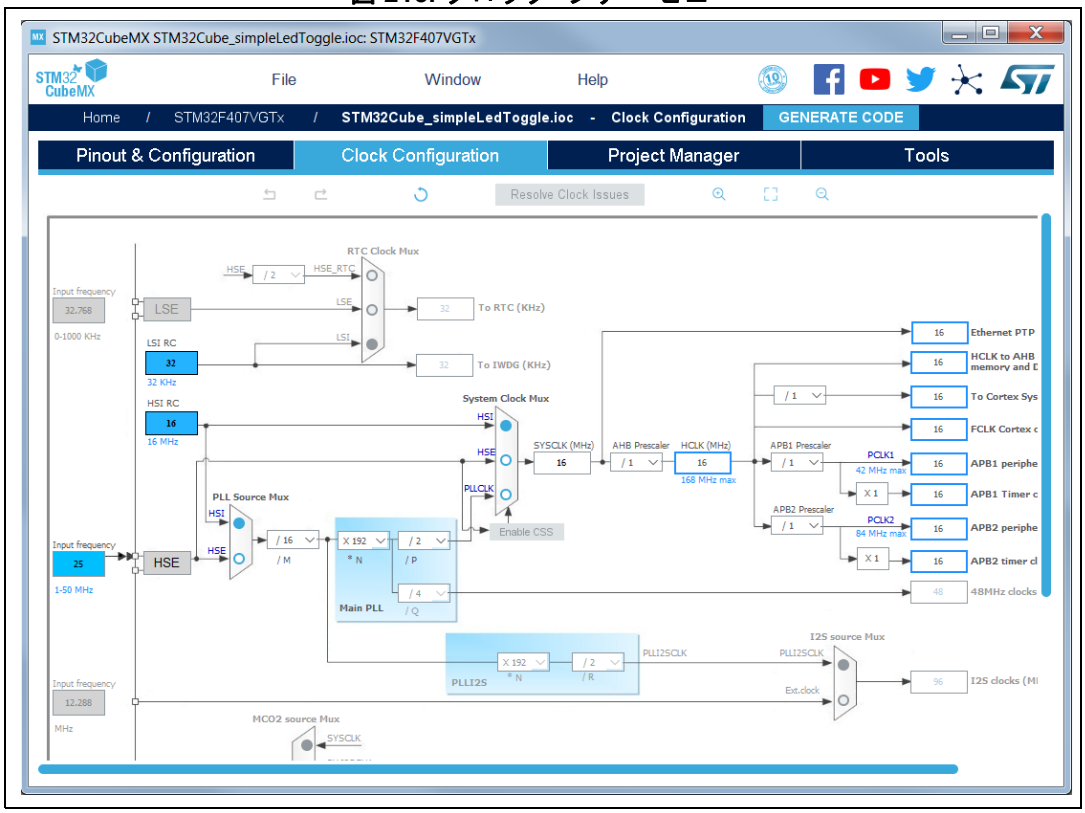
STM32CubeMX では、ユーザが選択したクロック･ソースとプリスケーラから、自動的にシステム、CPU、および AHB/APB バスの各クロック周波数が生成されます。最小条件と最大条件を動的に検証することにより、誤った設定があれば検出され、赤紫色でハイライトされます。設定が存在しない場合や誤っている場合に講じるべき措置が、ツールチップに詳しく表示されます。ペリフェラルのパラメータ（UART のボー･レートに対する制限など）の中には、周波数の選択による影響を受けるものがあります。

STM32CubeMX では、クロック･ツリー･ビューで定義されたクロック設定を使用して、各ペリフェラル･クロックの初期化 C コードが生成されます。各クロック（Hz の単位で表した周波数値の HSE、HSI、および外部クロック）は、生成された C コードにより、プロジェクトの main.c と stm32f4xx_hal_conf.h の中で RCC 初期化の過程で設定されます。

次の手順に従って、マイクロコントローラのクロック･ツリーを設定します。

1. **[Clock Configuration]** タブをクリックして、クロック･ツリーを表示します（図 215 を参照）。
   内部クロック（HSI、LSI）、システム･クロック（SYSCLK）、およびペリフェラル･クロックの各周波数フィールドは編集できません。システム･クロックとペリフェラル･クロックは、クロック･ソースを選択して調整できるほか、必要に応じ、PLL、プリスケーラと逓倍回路を使用して調整することもできます。

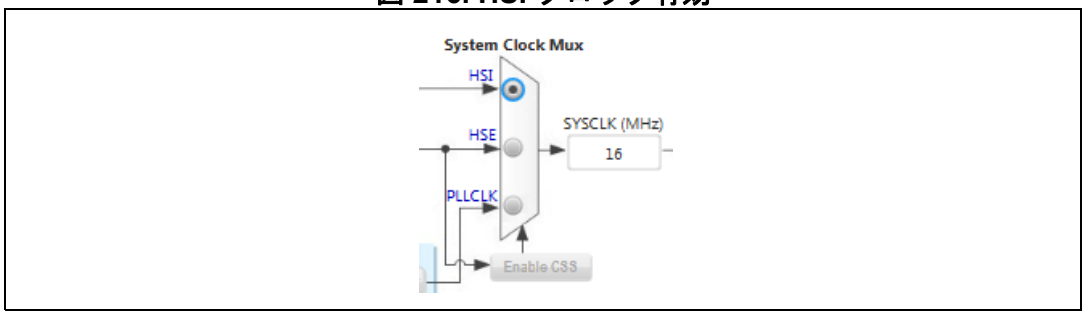
**図 215. クロック･ツリー･ビュー**



2. 最初に、マイクロコントローラのシステム･クロックを駆動するクロック･ソース（HSE、HSI、または PLLCLK）を選択します。

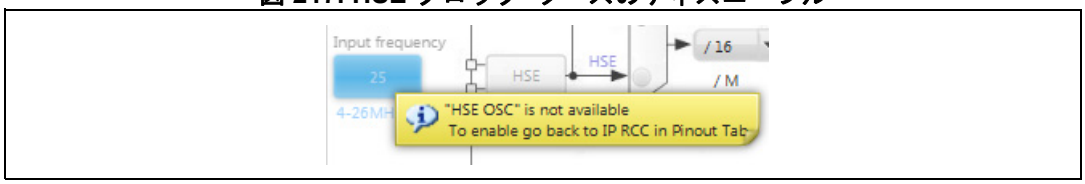
   このチュートリアルで使用している例では、HSI を選択して内部の 16 MHz クロックを使用しています（図 216 を参照）。

**図 216. HSI クロック有効**




外部クロック･ソース（HSE または LSE）を使用するには、**[Pinout]** ビューで RCC ペリフェラルを設定する必要があります。外部クロックのクリスタルを接続するためにピンを使用するからです（図 217 を参照）。

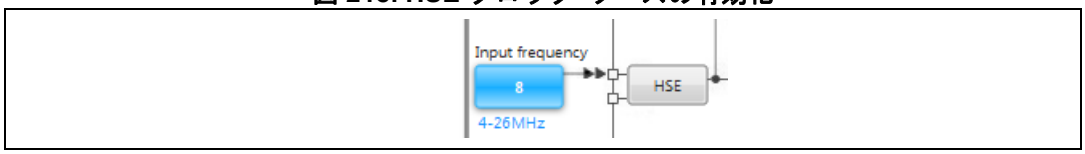
**図 217. HSE クロック･ソースのディスエーブル**




STM32F4DISCOVERY ボードの他のクロック設定オプションを次に示します。

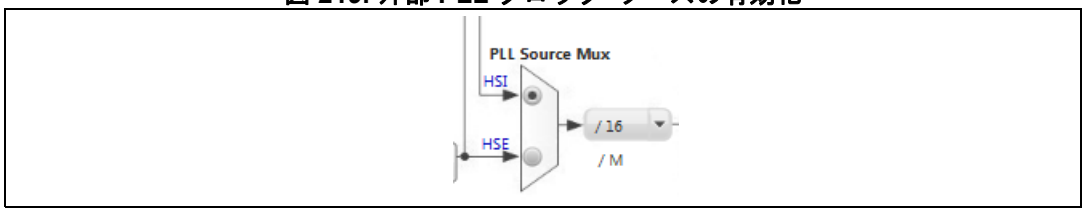
– 外部 HSE ソースを選択して HSE の［Input frequency］ボックスに「8」を入力します。ディスカバリ･ボードに 8 MHz のクリスタルを接続しているからです。

**図 218. HSE クロック･ソースの有効化**




– 外部 PLL クロック･ソースを選択し、PLL 入力クロック･ソースとして HSI または HSE を選択します。

**図 219. 外部 PLL クロック･ソースの有効化**



3. PLL とプリスケーリングは使用せず、HSI を使用してコア･クロックとペリフェラル･クロックを 16 MHz に維持します。

**注 :** **必要に応じ、PLL、プリスケーラ、および逓倍回路を使用して、システム･クロックとペリフェラル･クロックを詳しく調整します。**

**システム･クロックから独立した他のクロック･ソースは、次の手順で設定できます。**

- USB OTG FS、乱数発生回路、および SDIO の各クロックは、PLL の独立出力で駆動されます。
- $I^2S$ ペリフェラルは、専用の内部クロック（PLLI2S）を備えていますが、独立した外部クロック･ソースからこのクロックを得ることもできます。
- USB OTG HS と Ethernet の各クロックは、外部ソースから得られます。

4. 必要に応じ、外部回路に 2 つのクロックを出力できるマイクロコントローラ･クロック出力（MCO）ピンに対してプリスケーラを設定します。
5. プロジェクトを保存するには をクリックします。
6. **[Configuration]** タブに移動して、プロジェクト設定を続行します。

## 11.6 マイクロコントローラの初期化パラメータの設定

**注意 :** **STM32CubeMX によって生成される C コードでは、STM32Cube ファームウェアのライブラリを使用した、マイクロコントローラのペリフェラルとミドルウェアの初期化を対象としています。**

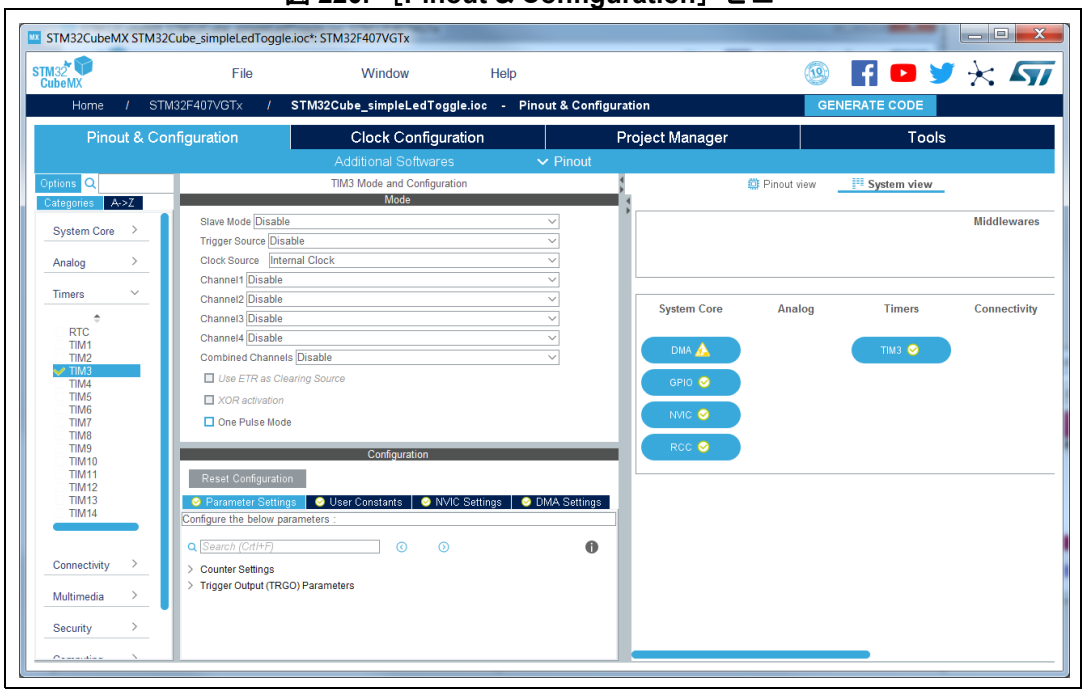
### 11.6.1 初期条件

**[Pinout & Configuration]** タブの **[Mode]** および **[Configuration]** パネルを使用して、アプリケーションに必要なすべてのコンポーネント（ペリフェラル、ミドルウェア、追加のソフトウェア）を 1 つずつ選択、設定します（図 220 参照）。

ペリフェラルを正しく設定していないと、ツールチップと警告メッセージが表示されます（詳細については セクション 4 : STM32CubeMXのユーザ･インタフェース を参照）。

**注 :** **RCC ペリフェラルの初期化では、このビューで指定したパラメータ設定およびクロック･ツリー･ビューで指定した設定（クロック･ソース、周波数、プリスケーラ値など）が使用されます。**

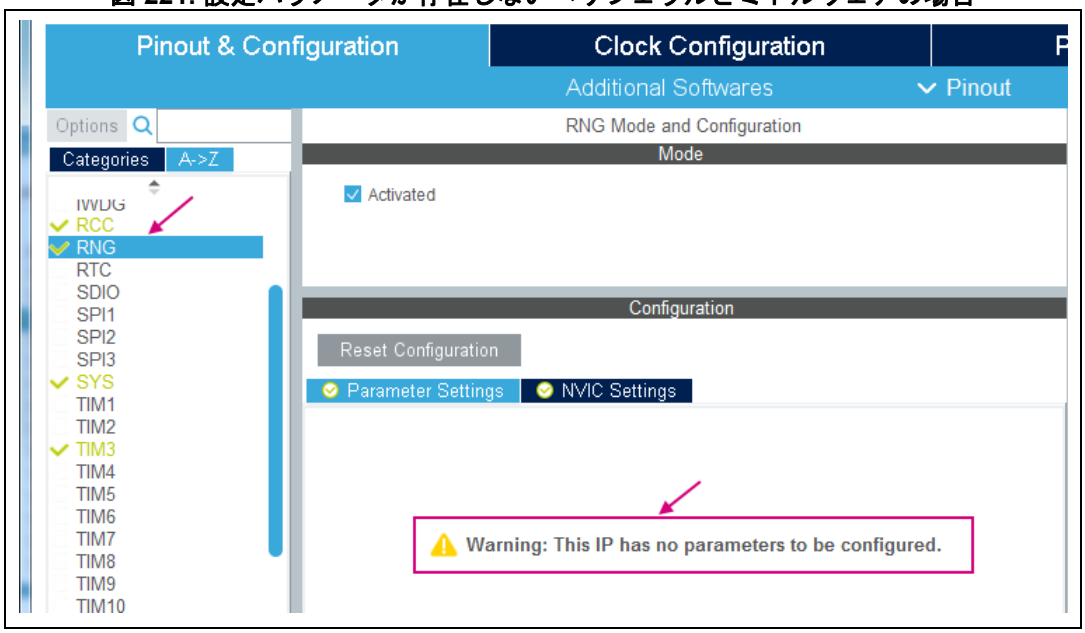
図 220. ［Pinout & Configuration］ビュー




## 11.6.2 ペリフェラルの設定

メイン･パネルには、各ペリフェラル･インスタンスに対応する専用ボタンがあります。次の図に示すように、一部のペリフェラル･モードには設定可能なパラメータが存在しません。

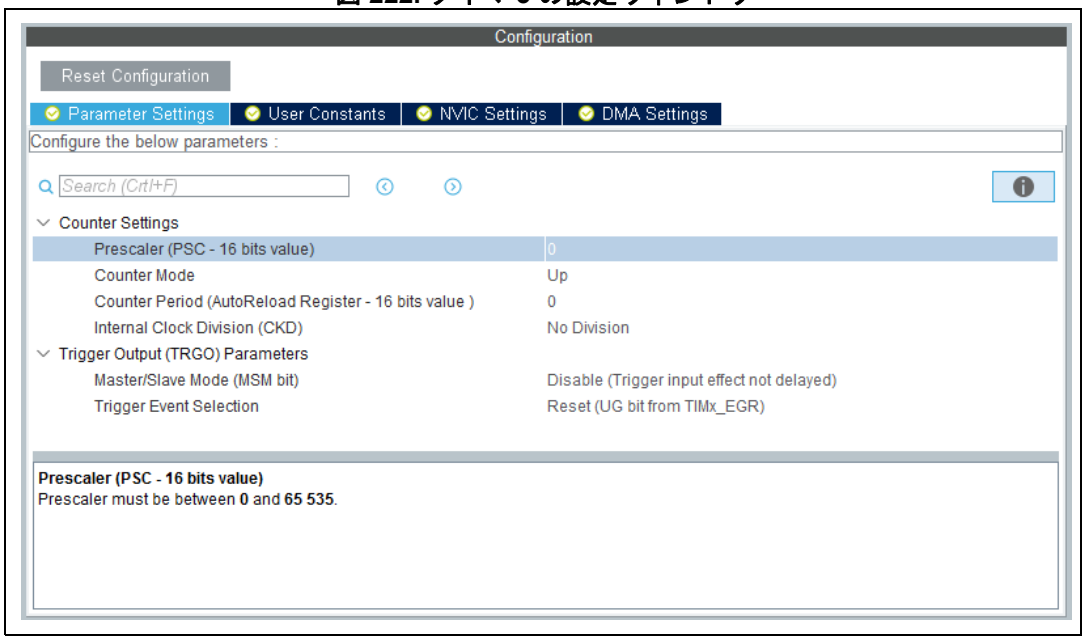
図 221. 設定パラメータが存在しないペリフェラルとミドルウェアの場合



次のステップに従って、ペリフェラル設定に進みます。

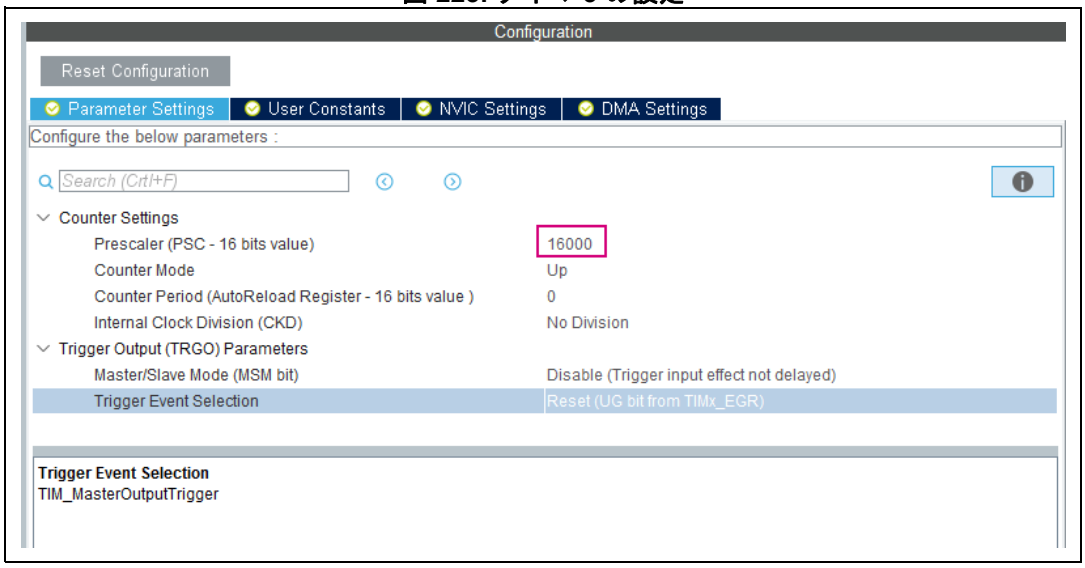
1. ペリフェラルに対応するボタンをクリックして、その設定ウィンドウを開きます。
   このチュートリアルの例では次の操作を実行します。
   a) ［**TIM3**］をクリックして、タイマ設定ウィンドウを開きます。

**図 222. タイマ 3 の設定ウィンドウ**




b) 16 MHz の APB クロックを使用するので（クロック･ツリー･ビュー）、プリスケーラを 16000 に設定します。また、カウンタ周期を 1000 に設定して、LED が 1 ミリ秒間隔で点滅するようにします。

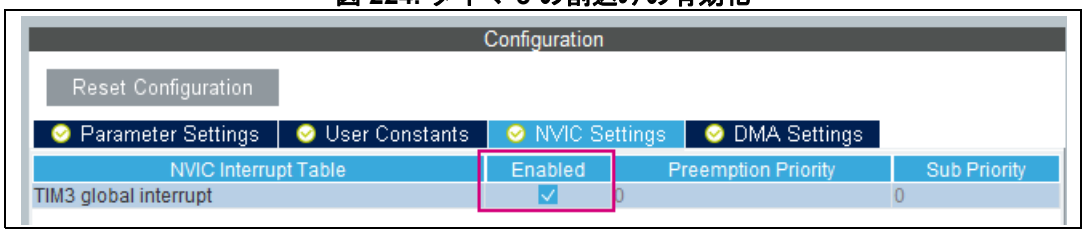
**図 223. タイマ 3 の設定**



2. 次の操作が可能な場合は、必要に応じて実行します。
   - ［**NVIC Settings**］タブを選択して NVIC 設定を表示し、このペリフェラルの割込みを有効にします。
   - ［**DMA Settings**］タブを選択して DMA 設定を表示し、このペリフェラルの DMA 転送を設定します。

     チュートリアルの例では、DMA を使用せず、GPIO 設定は変更しません。図 224 に示すように割込みを有効にします。
   - ［**GPIO Settings**］タブを選択して GPIO 設定を表示し、このペリフェラルの GPIO を設定します。
   - 以下の項目を挿入します。
   - ［**User Constants**］タブを選択して、プロジェクトで使用する定数を指定します。

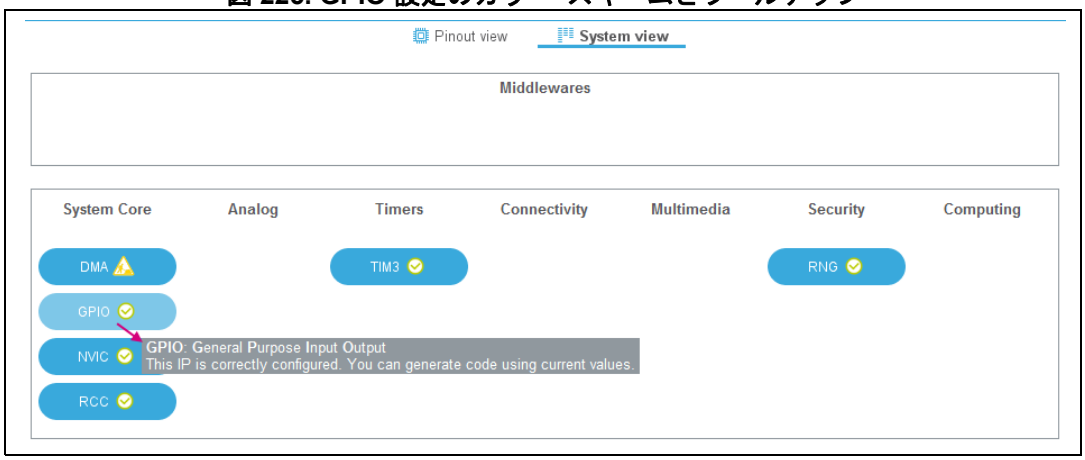
**図 224. タイマ 3 の割込みの有効化**




## 11.6.3 GPIO の設定

このウィンドウから、すべてのピン設定を調整できます。小さなアイコンとツールヒントで設定ステータスが示されます。

**図 225. GPIO 設定のカラー･スキームとツールチップ**

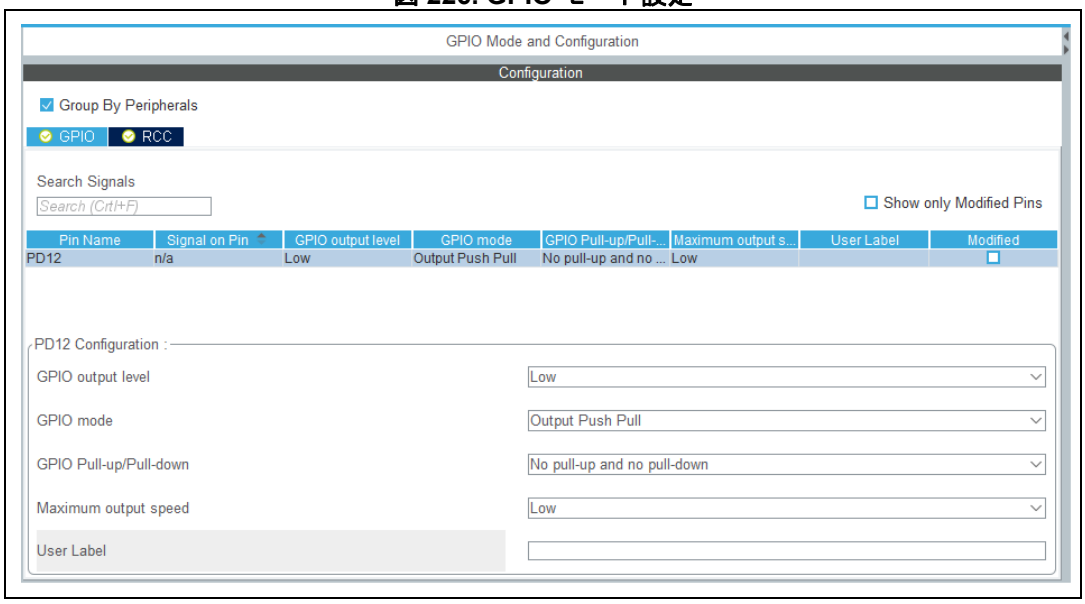


次の手順に従って、GPIO を設定します。

1. ［Configuration］ビューで**［GPIO］ボタン**をクリックして**［Pin Configuration］**ウィンドウを開きます。
2. 最初のタブには、GPIO モードが割り当てられているものの、専用のペリフェラルとミドルウェア用ではないピンが表示されます。ピン名を選択して、そのピンの設定を開きます。

   このチュートリアルの例では、［PD12］を選択して出力プッシュプル·モードに設定し、STM32F4DISCOVERY LED を駆動するようにします（図 226 を参照）。

**図 226. GPIO モード設定**



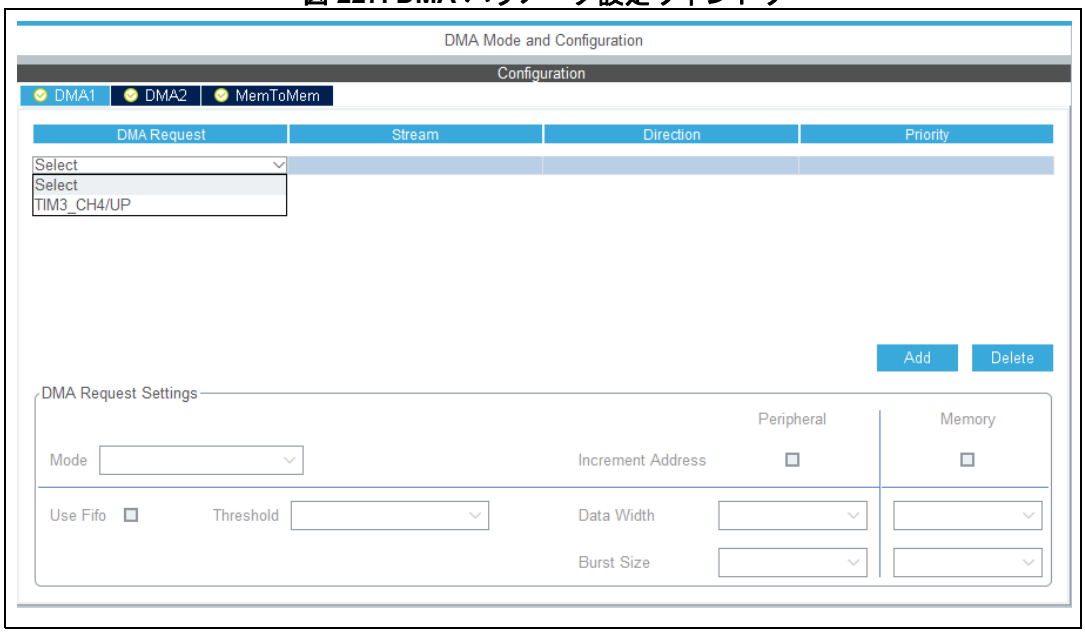
### 11.6.4 DMA の設定

この例では不要です。CPU に対する負荷を軽減するために、DMA 転送の使用をお勧めします。[DMA Configuration］ウィンドウでは、DMA を短時間で次のように簡単に設定できます（図 227 参照）。

1. 新しい DMA リクエストを追加して、使用可能な設定を一覧から選択します。
2. 使用可能なストリームからストリームを選択します。
3. 転送方向として［Memory to Peripheral］または［Peripheral to Memory］を選択します。
4. 優先度を選択します。
5. FIFO を有効にします。

**注：** **ペリフェラルとミドルウェアに対する DMA は、ペリフェラルとミドルウェアの設定ウィンドウでも設定できます。**

**図 227. DMA パラメータ設定ウィンドウ**

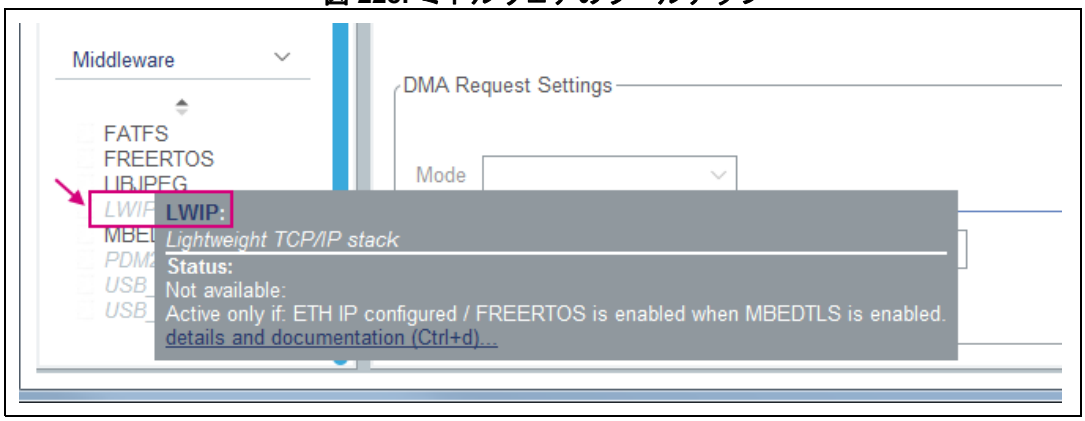


### 11.6.5 ミドルウェアの設定

このチュートリアルでは、この設定は不要です。

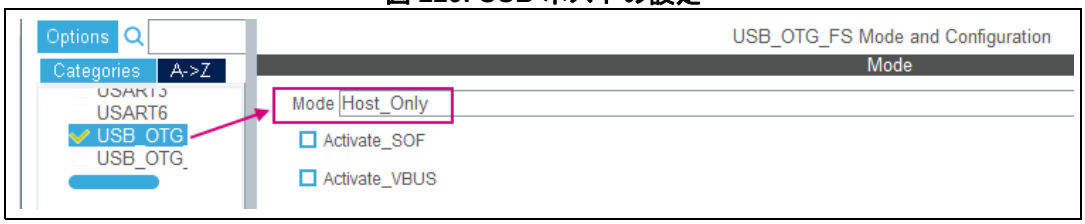
ミドルウェア･モードでペリフェラルが必要な場合、ミドルウェア･モードを使用可能にするには、**[Pinout]** ビューでそのペリフェラルを設定する必要があります。以下に示すように、ツールチップで手順が案内されます。

**図 228. ミドルウェアのツールチップ**



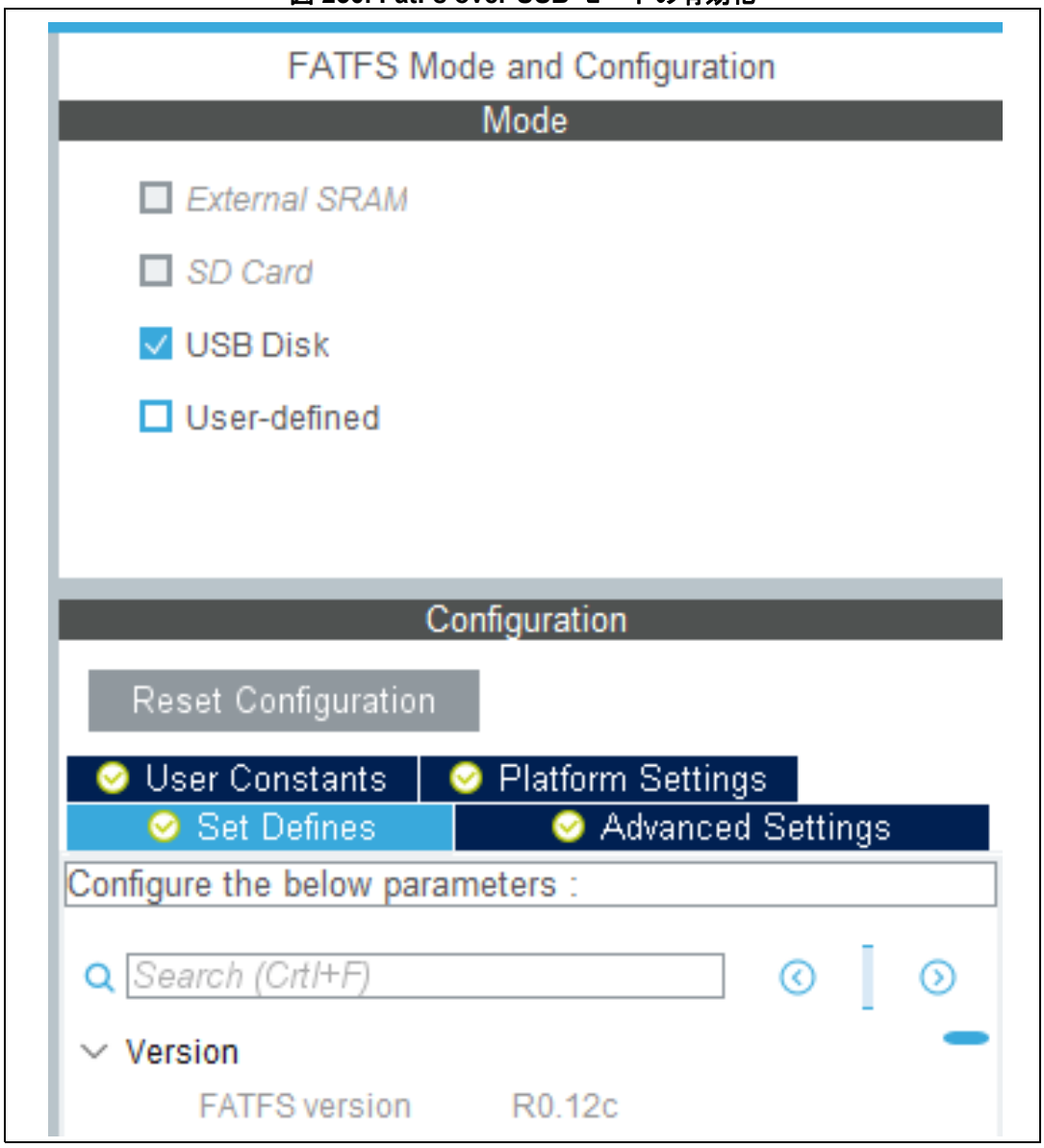

1. **[Pinout]** ビューで USB ペリフェラルを設定します。

**図 229. USB ホストの設定**




2. USB ホストのミドルウェアから MSC_FS クラスを選択します。
3. ツリー･パネルでチェックボックスをチェックして、FatFs の USB モードを有効にします。

図 230. FatFs over USB モードの有効化

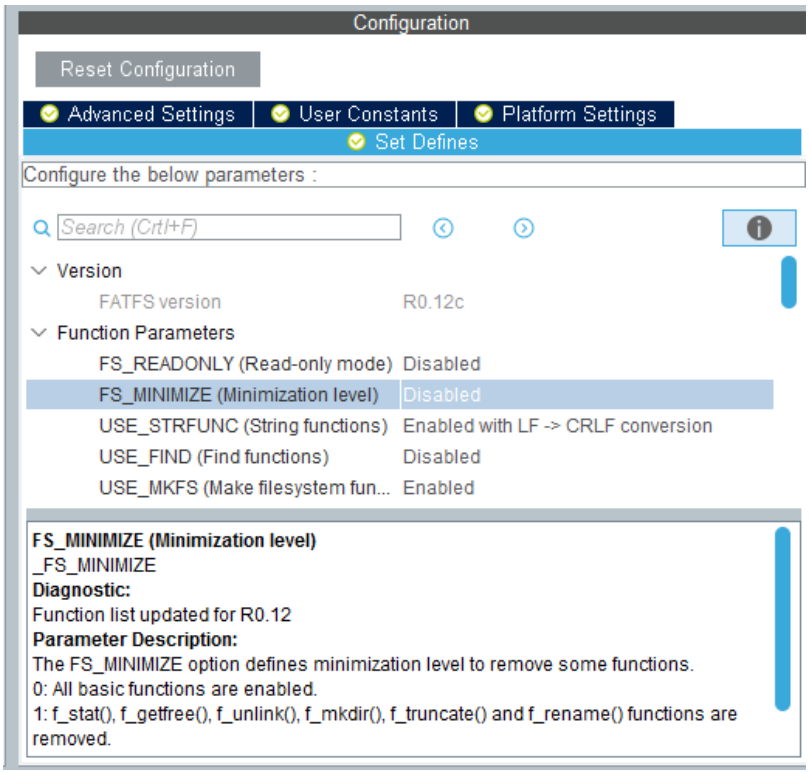


4. ［**Configuration**］ビューを選択します。［FatFs］ボタンと［USB］ボタンが表示されます。

**図 231. FatFs と USB を有効にした［System］ビュー**

5. デフォルト設定の FatFs と USB は、すでに設定済み  になっています。デフォルト設定を表示するには［**FatFs**］ボタンと［**USB**］ボタンをクリックします。ウィンドウ下部に表示されるガイドラインに従って設定を変更することもできます。

**図 232. FatFs の define ステートメント**



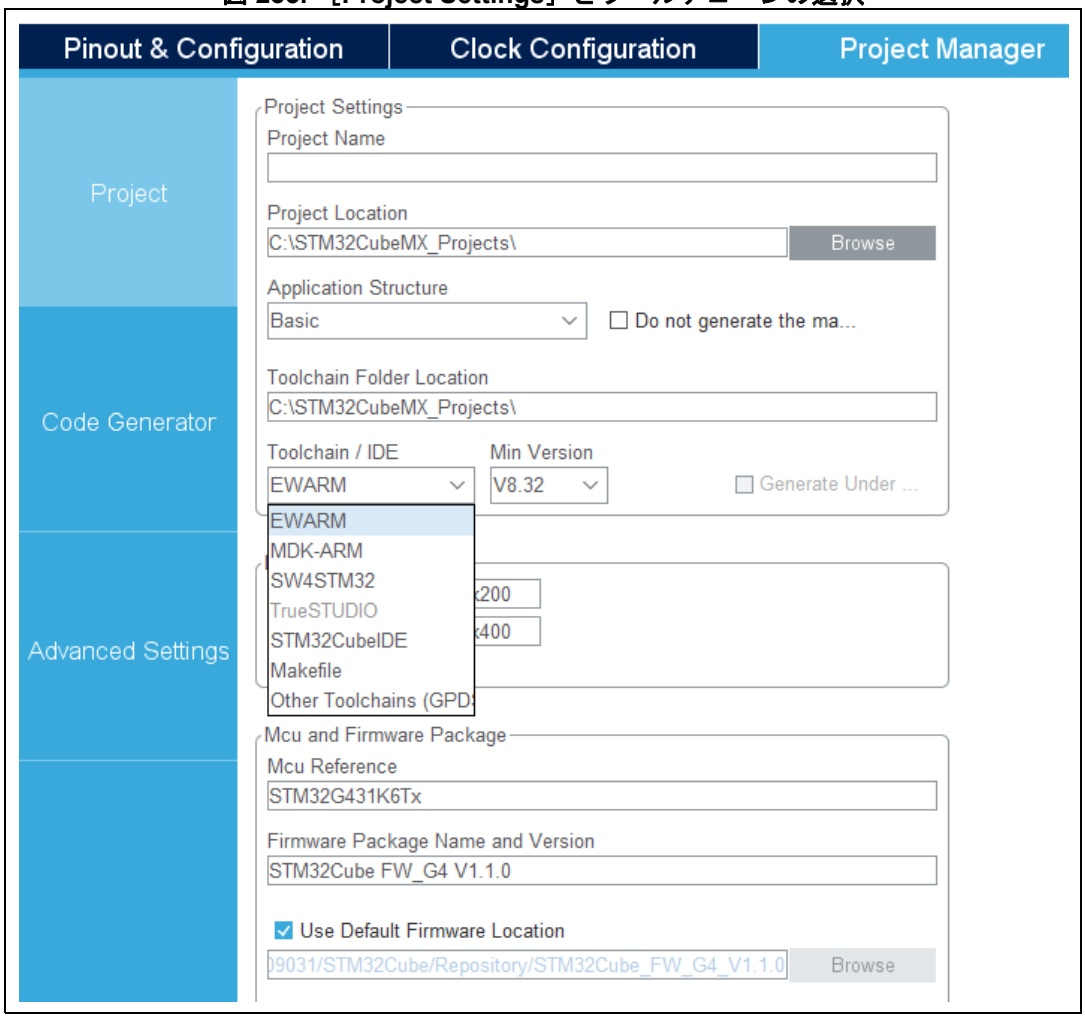
## 11.7 C プロジェクト全体の生成

### 11.7.1 プロジェクト·オプションの設定

図 233 に示すように、C コードを生成する前にデフォルトのプロジェクト設定を調整できます。

1. **［Project Manager］**ビューを選択して、プロジェクト設定と生成オプションを変更します。
2. **［Project］タブ**を選択して、プロジェクトの**名前**と**場所**、およびプロジェクトを生成するための**ツールチェーン**と**ツールチェーンのバージョン**を選択します（図 233 参照）。

**図 233. ［Project Settings］とツールチェーンの選択**

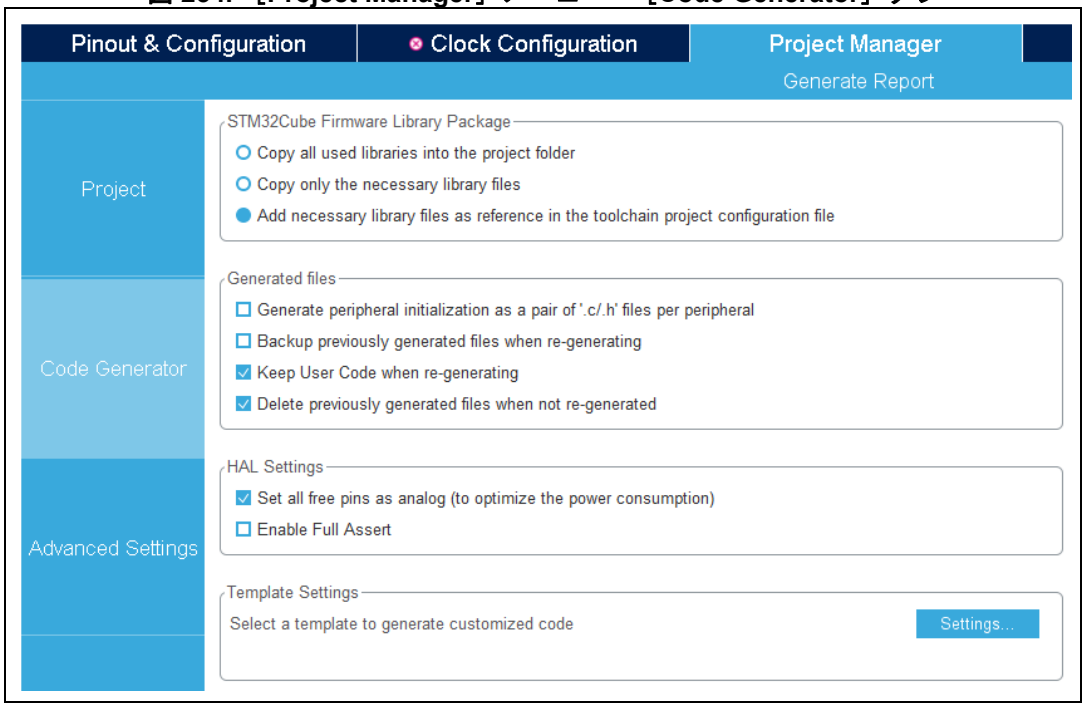



3. **［Code Generator］**タブを選択して、次の C コード生成オプションを選択します。
   - Projects フォルダにコピーするライブラリ·ファイル
   - C コードの再生成（C コード再生成時に維持またはバックアップするファイルなど）
   - HAL 固有のアクション（マイクロコントローラの消費電力削減を目的として、すべての未使用ピンをアナログ I/O として設定するなど）

   このチュートリアルでは、図 234 のとおりに設定して **［OK］**をクリックします。

注： ファームウェア･パッケージが見つからない場合はダイアログ･ウィンドウが表示されます。ファームウェア･パッケージのダウンロード方法の説明については、次のセクションを参照してください。

図 234. ［Project Manager］メニュー - ［Code Generator］タブ

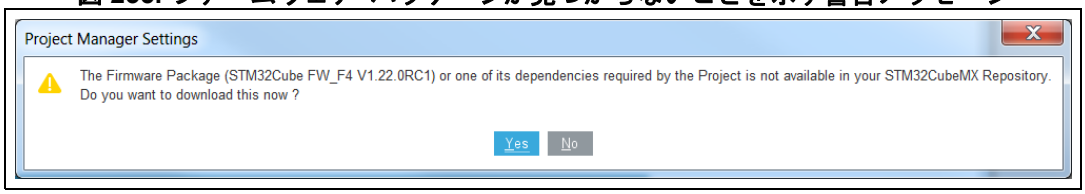



## 11.7.2 ファームウェア･パッケージのダウンロードと C コードの生成

1. GENERATE CODE をクリックして C コードを生成します。

   STM32CubeMX は、C コードを生成する際、プロジェクトをコンパイルできるように、該当する STM32Cube マイクロコントローラ･パッケージからプロジェクト･フォルダにファイルをコピーします。初めてプロジェクトを生成するときは PC にファームウェア･パッケージが存在しないので、次の警告メッセージが表示されます。

図 235. ファームウェア･パッケージが見つからないことを示す警告メッセージ

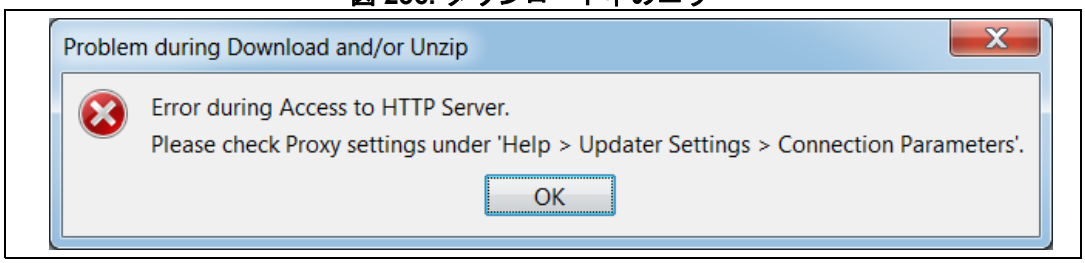



2. STM32CubeMX から、該当するファームウェア･パッケージをダウンロードするか、このまま操作を続行するかの確認を求められます。プロジェクト一式を取得するには［**Download**］をクリックします。これは、選択している IDE でそのまま使用できるプロジェクトです。

   ［**Continue**］をクリックすると、Inc フォルダと Src フォルダのみが作成され、STM32CubeMX で生成された初期化ファイルがそこに保持されます。プロジェクト全体を取得するには、必要なファームウェアとミドルウェアのライブラリを手動でコピーする必要があります。

ダウンロードが失敗した場合、エラー・メッセージが表示されます。

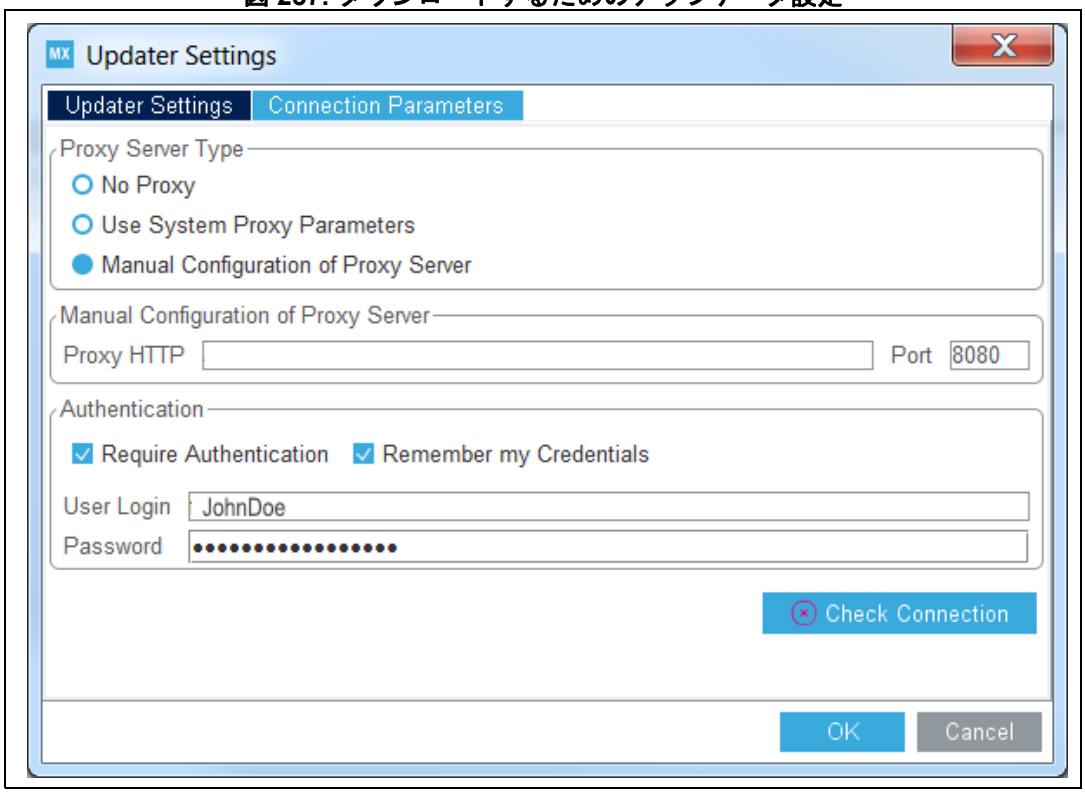
**図 236. ダウンロード中のエラー**




この問題を解決するには、次の 2 つのステップを実行します。解決しない場合はこのステップを省略します。

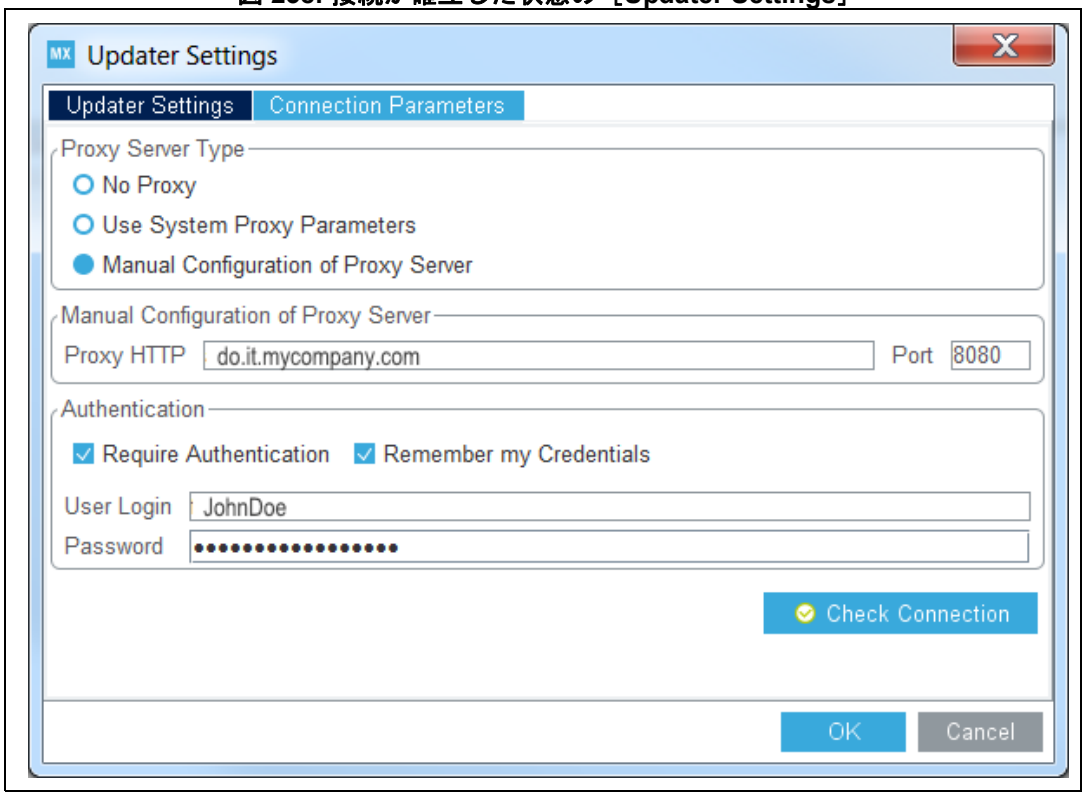
3. **[Help] → [Updater Settings] メニュー**を選択して、ネットワーク設定に一致するように接続パラメータを調整します。

**図 237. ダウンロードするためのアップデータ設定**




4. **[Check Connection]** をクリックします。接続が確立されると、チェック・マークが緑色に変わります。

図 238. 接続が確立した状態の［Updater Settings］




5. 接続が機能したら、GENERATE CODE をクリックして C コードを生成します。C コード生成プロセスが始まり、次の図に示すように進捗が表示されます。

図 239. ファームウェア·パッケージのダウンロード

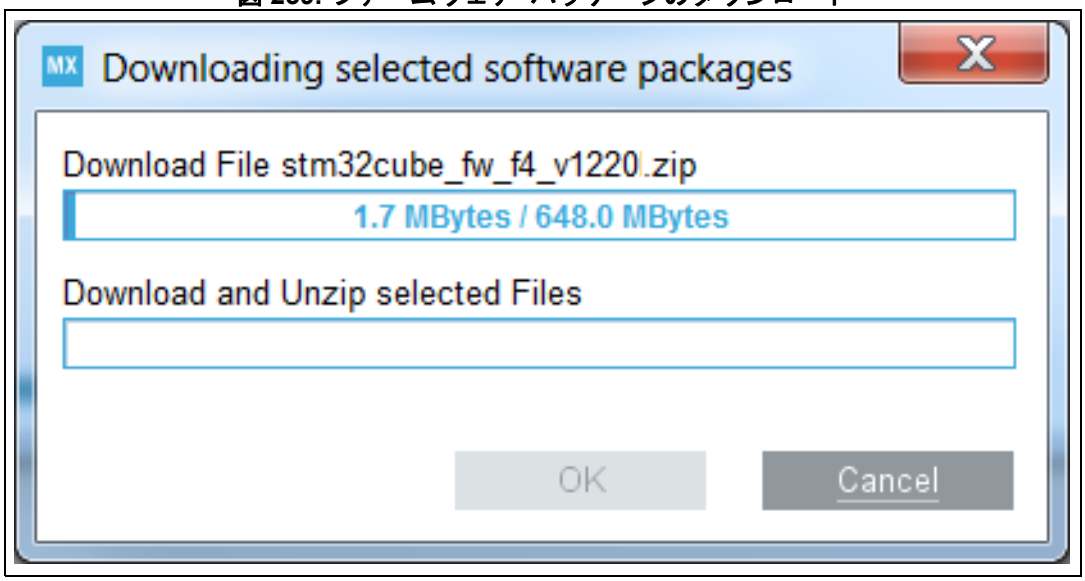
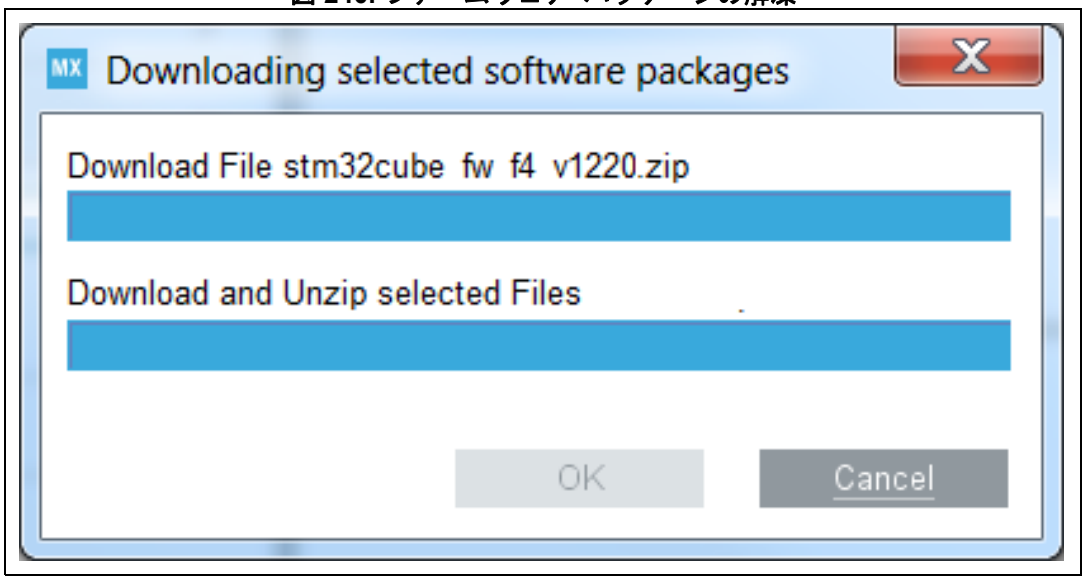
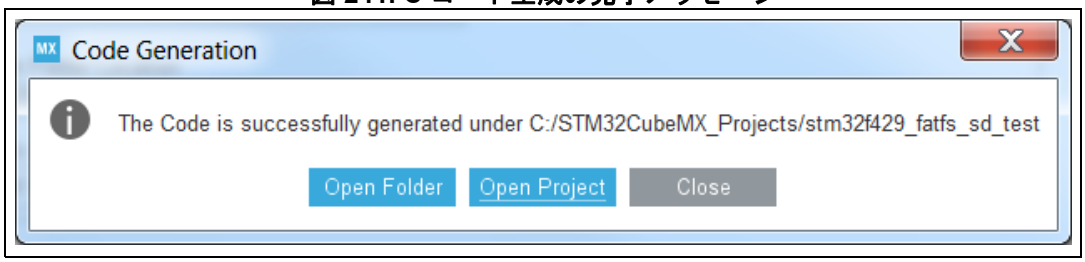


図 240. ファームウェア･パッケージの解凍



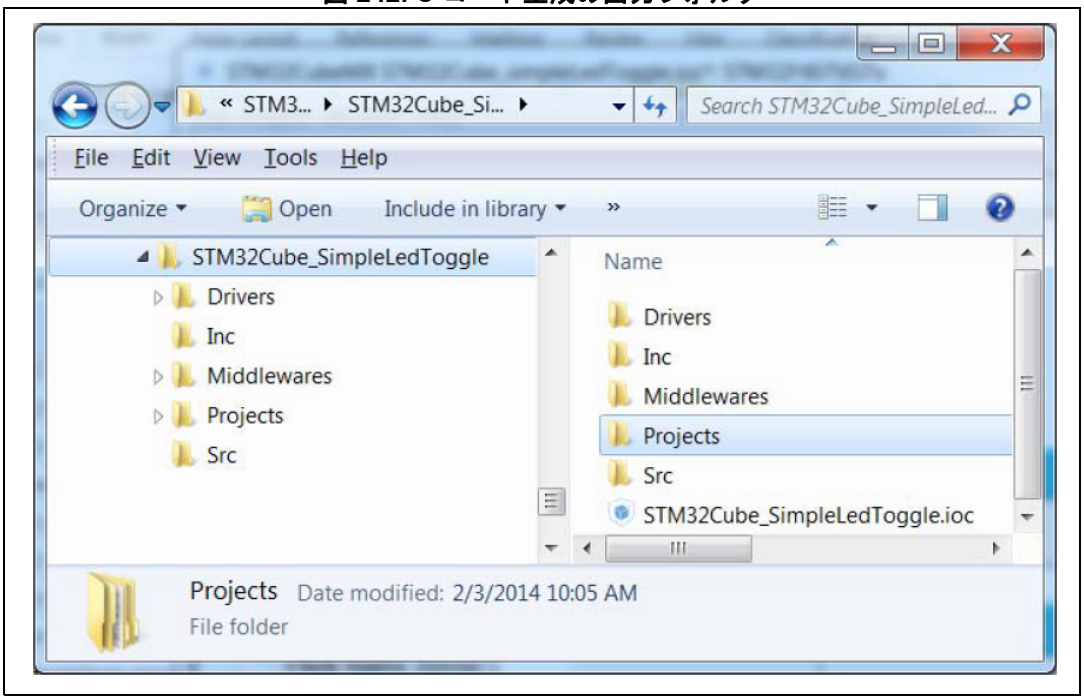

6. 最後に、C コード生成が正常終了したことを示す確認メッセージが表示されます。

図 241. C コード生成の完了メッセージ



7. **[Open Folder]** をクリックして、生成されたプロジェクトの内容を表示するか、**[Open Project]** をクリックして IDE で直接プロジェクトを開きます。つづいてセクション 11.8：に進みます。

**図 242. C コード生成の出力フォルダ**




生成されたプロジェクトの内容を次に示します。

- STM32CubeMX .ioc プロジェクト･ファイル。ルート･フォルダにあります。プロジェクトのユーザ設定および STM32CubeMX ユーザ･インタフェースを通じて生成された設定を収めています。
- Drivers フォルダと Middlewares フォルダには、ユーザ設定に関連するファームウェア･パッケージ･ファイルのコピーが保持されています。
- Projects フォルダには IDE 固有のフォルダがあり、IDE でプロジェクトを開発してデバッグするうえで必要なすべてのファイルを収めています。
- Inc フォルダと Src フォルダには、main.c をはじめとして、ミドルウェア、ペリフェラル、および GPIO の初期化のために STM32CubeMX で生成されたファイルがあります。STM32CubeMX で生成されたファイルにはユーザ専用セクションがあり、そこにはユーザ定義の C コードを追加できます。

**注意：** **ユーザ･セクションに記述した C コードは、次回の C コード生成でも保持されます。ユーザ･セクション以外の箇所に記述した C コードは上書きされます。**

ユーザ･セクションを移動した場合やユーザ･セクションのデリミタの名前を変更した場合、ユーザの C コードは保持されません。

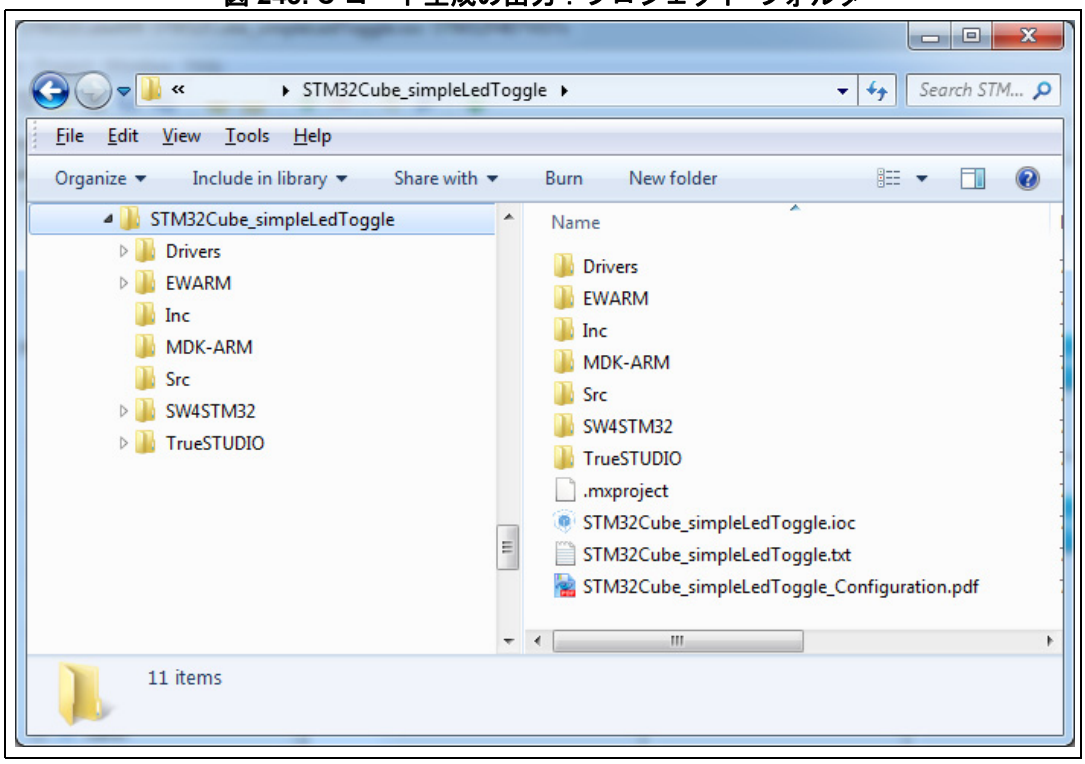
## 11.8 C コード・プロジェクトのビルドと更新

生成した初期化 C コードを IAR™ EWARM ツールチェーンで使用してプロジェクトを完成する方法について説明します。完成すると、TIM3 の周波数に従って LED が点滅するようになります。

C コード生成のために選択したツールチェーン用にフォルダが用意されます。**[Project Manager]** メニューからツールチェーンを選択して [Generate Code] を再度クリックすることにより、複数のツールチェーンでプロジェクトを生成できます。

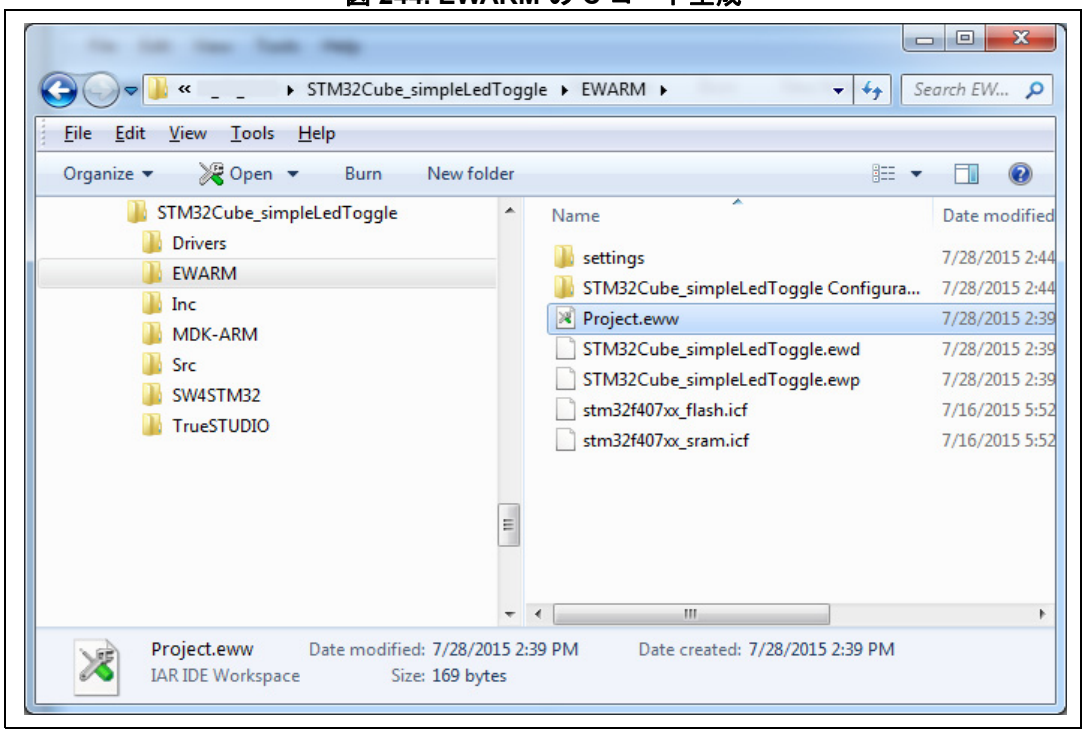
1. 表示されたダイアログ・ウィンドウで **[Open Project]** をクリックするか、STM32CubeMX で生成されたプロジェクト・ディレクトリのツールチェーン・フォルダにある該当の IDE ファイルをダブルクリックすることで、IDE ツールチェーンで直接プロジェクトを開きます（図 241 を参照）。

**図 243. C コード生成の出力 : プロジェクト・フォルダ**



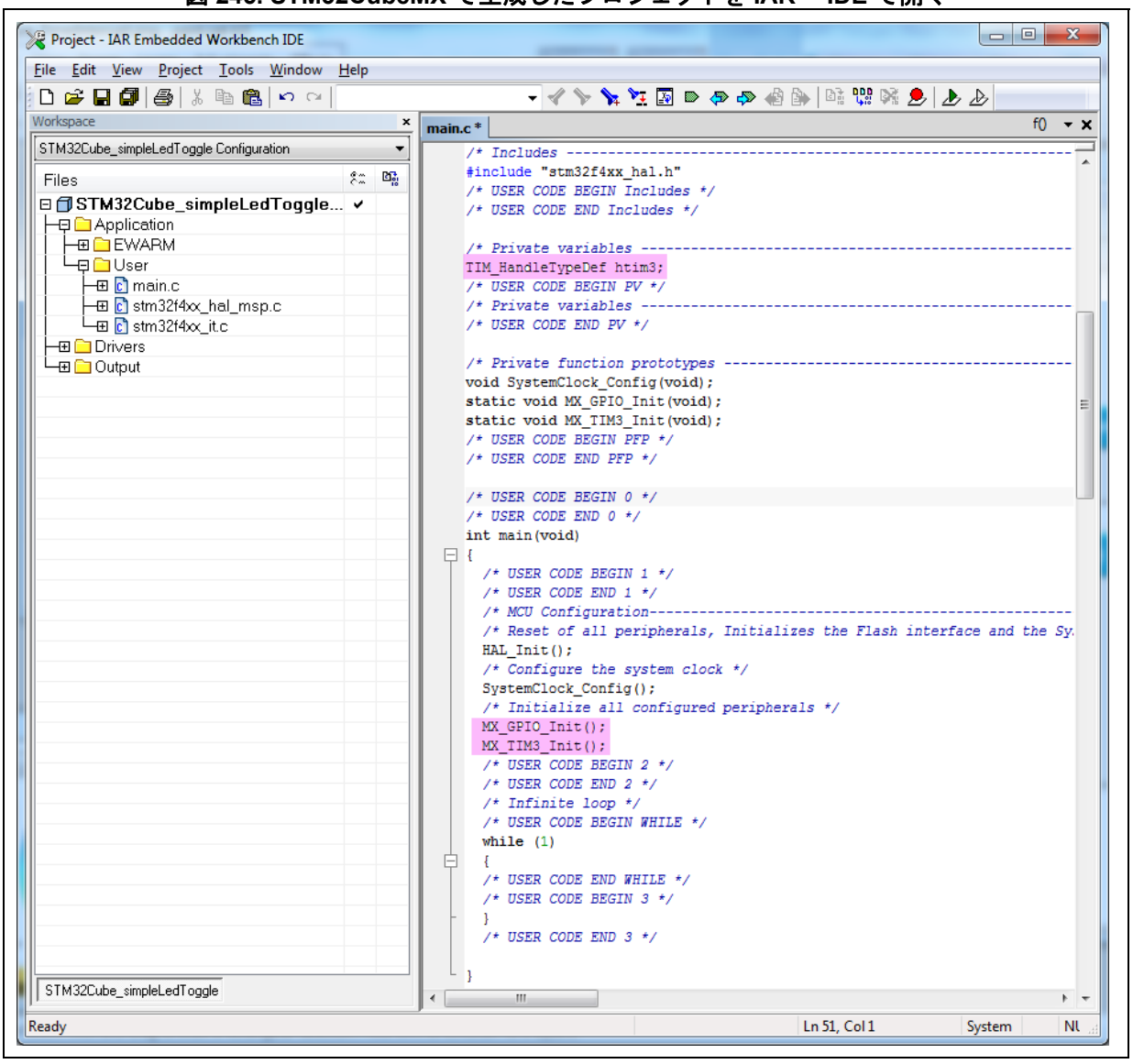
2. 例として、.eww ファイルを選択して、プロジェクトを IAR™ EWARM IDE にロードします。

**図 244. EWARM の C コード生成**



3. main.c ファイルを選択して、エディタで開きます。

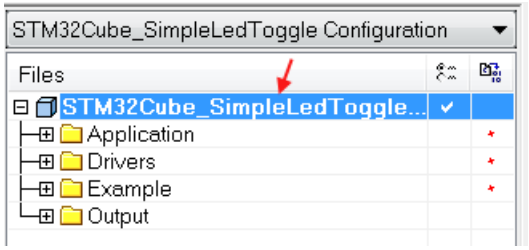
**図 245. STM32CubeMX で生成したプロジェクトを IAR™ IDE で開く**




htim3 構造体ハンドラ、システムクロック、GPIO、および TIM3 初期化関数が定義されています。初期化関数は main.c から呼び出されます。この時点では、ユーザ C コード･セクションには何も記述されていません。

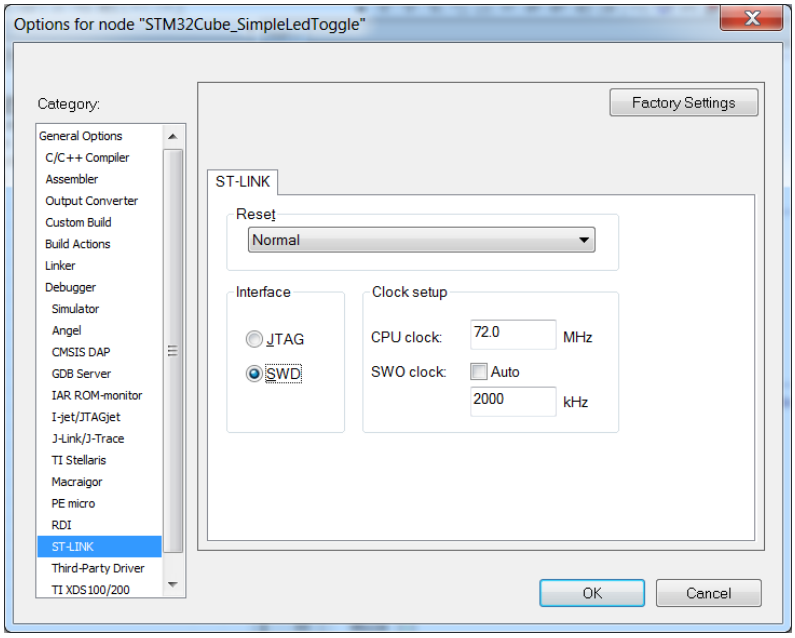
4. IAR™ IDE で、プロジェクト名を右クリックして［**Options**］を選択します。

**図 246. IAR™ のオプション**



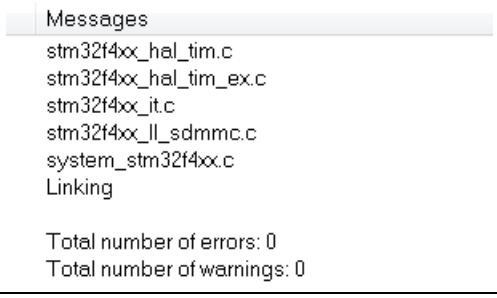

5. ［ST-LINK］カテゴリをクリックして、STM32F4DISCOVERY ボードと通信するために［SWD］が選択されていることを確認します。**［OK］**をクリックします。

**図 247. SWD 接続**




6. **［Project］→［Rebuild all］**を選択します。プロジェクトのビルドが正常に終了したことを確認します。

**図 248. プロジェクトのビルドのログ**



7. 専用のユーザ･セクションに**のみ**ユーザ C コードを追加します。

**注 :** **ユーザ･セクションには main while(1) ループが記述されています。**

例 :

a) main.c ファイルを編集します。

b) タイマ 3 を開始するために、次に示す C コードでユーザ･セクション 2 を更新します。

**図 249. ユーザ･セクション 2**

```
  HAL_Init();
  /* Configure the system clock */
  SystemClock_Config();
  /* Initialize all configured peripherals */
  MX_GPIO_Init();
  MX_TIM3_Init();

  /* USER CODE BEGIN 2 */
  HAL_TIM_Base_Start_IT(&htim3);
  /* USER CODE END 2 */

  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
  while (1)
  {
```

c) 次に示す C コードをユーザ･セクション 4 に追加します。

**図 250. ユーザ･セクション 4**

```
/* USER CODE BEGIN 4 */

void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)
{
  if ( htim->Instance == htim3.Instance )
  {
    HAL_GPIO_TogglePin(GPIOD, GPIO_PIN_12);
  }
}
/* USER CODE END 4 */
```

この C コードは、HAL タイマ･ドライバ（stm32f4xx_hal_tim.h）に定義されている weak 属性のコールバック関数を実装して、タイマ･カウンタの周期が経過するたびに、緑色の LED を駆動する GPIO ピンの出力を切り替えます。

8.  を使用して、ボードの再ビルドとプログラミングを実行します。プロジェクト･オプションとして、[ST-LINK] オプションの [SWD] をチェックしていることを確認します。チェックしていないとプログラミングが失敗します。
9.  を使用して、プログラムを起動します。STM32F4DISCOVERY ボードの緑色の LED が 1 秒おきに点滅します。
10. マイクロコントローラの設定を変更するには、STM32CubeMX のユーザ･インタフェースに戻り、変更を実装して C コードを再生成します。プロジェクトが更新されます。[Project Manager] の [Code Generator] タブで ☑ Keep User Code when re-generating オプションを有効にしている場合はユーザ･セクションの C コードが保持されます。

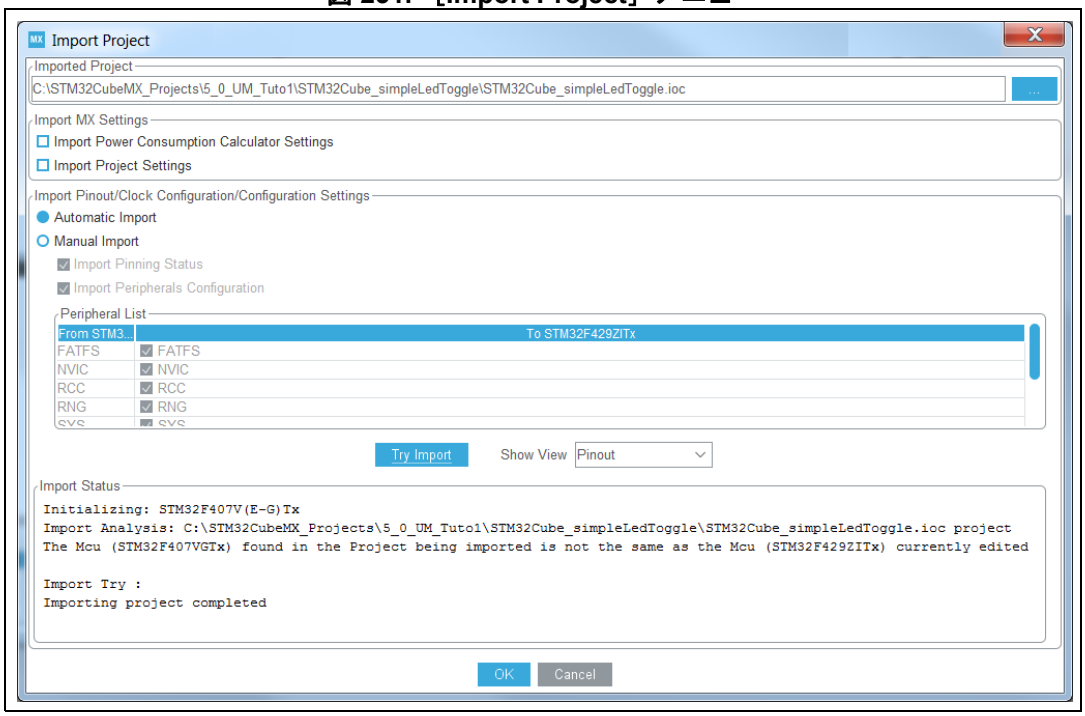
## 11.9 別のマイクロコントローラへの切替え

STM32CubeMX では、プロジェクト設定を同一シリーズのマイクロコントローラにロードできます。

次のように実行します：

1. **［File］→［New Project］**を選択します。
2. 同一シリーズに属するマイクロコントローラを選択します。たとえば、32F429IDISCOVERY ボードのコア･マイクロコントローラである STM32F429ZITx を選択できます。
3. **［File］→［Import project］**を選択します。**［Import project］**ウィンドウで、ロードする .ioc ファイルを参照します。現在選択しているマイクロコントローラ（STM32F429ZITx）が、.ioc ファイルで指定されているマイクロコントローラ（STM32F407VGTx）と異なることを警告するメッセージが表示されます。いくつかのインポート･オプションが表示されます（図 251 を参照）。
4. **［Try Import］**ボタンをクリックします。［Import Status］をチェックして、インポートが正常に終了したかどうかを確認します。
5. **［OK］**をクリックして、実際にプロジェクトをインポートします。出力タブが表示されて、インポートの結果が報告されます。
6. 32F429IDISCOVERY ボードでは、緑色の LED が PG13 に接続されています。Ctrl キーを押しながら**［PD12］**を右クリックして、［PG13］までドラッグ･アンド･ドロップします。
7. **［Project Manager］**の［Project］タブで、新しいプロジェクト名とフォルダの場所を設定します。**［Generate］アイコン**をクリックしてプロジェクトを保存し、コードを生成します。
8. ダイアログ･ウィンドウで**［Open the project］**を選択します。ユーザ･コードでユーザ･セクションが更新されるので、PG13 の GPIO 設定が更新されていることを確認します。プロジェクトをビルドして、ボードをFlashします。プログラムを起動して、1 秒ごとに LED が点滅することを確認します。

**図 251. ［Import Project］メニュー**



# 12 チュートリアル 2 - STM32429I-EVAL 評価 ボードを使用した、SD カード上の FatFs の例

このチュートリアルでは、FatFs のファイル･システム･ミドルウェアを使用して、STM32429I-EVAL1 の SD カード上にファイルを作成して書き込みます。

プロジェクトを生成してチュートリアル 2 を実行するには、次の手順に従います。

1. STM32CubeMX を起動します。
2. **［File］→［New Project］**を選択します。プロジェクトのウィンドウが開きます。
3. **［Board Selector］**タブをクリックして、ST 製ボードのリストを表示します。
4. ボードのタイプとして**［Evaluation Board］**、シリーズとして**［STM32F4］**を選択して、リストを絞り込みます。
5. アプリケーションで使用するペリフェラルのコードのみが生成されるように、［Initialize all peripherals with their default Mode］プロンプトで［Yes］と回答します。
6. ［STM32429I-EVAL1］ボードを選択して**［OK］**をクリックします。すべてのペリフェラルをそれぞれのデフォルト･モードに初期化するかどうかを確認するダイアログ･ボックスが表示されるので［No］と回答します（図 252 を参照）。**［Pinout］**ビューがロードされて、評価ボード上のマイクロコントローラのピン配置設定に一致した表示になります（図 253 を参照）。

**図 252. ボードのペリフェラル初期化ダイアログ･ボックス**

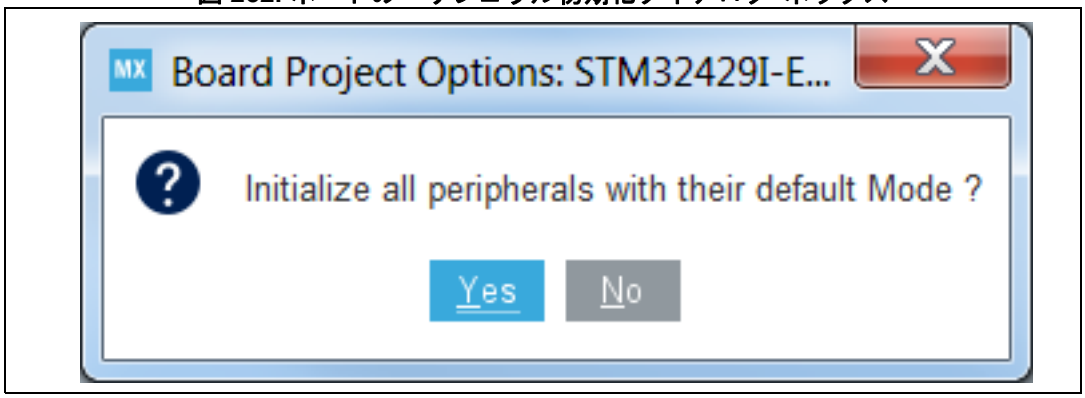
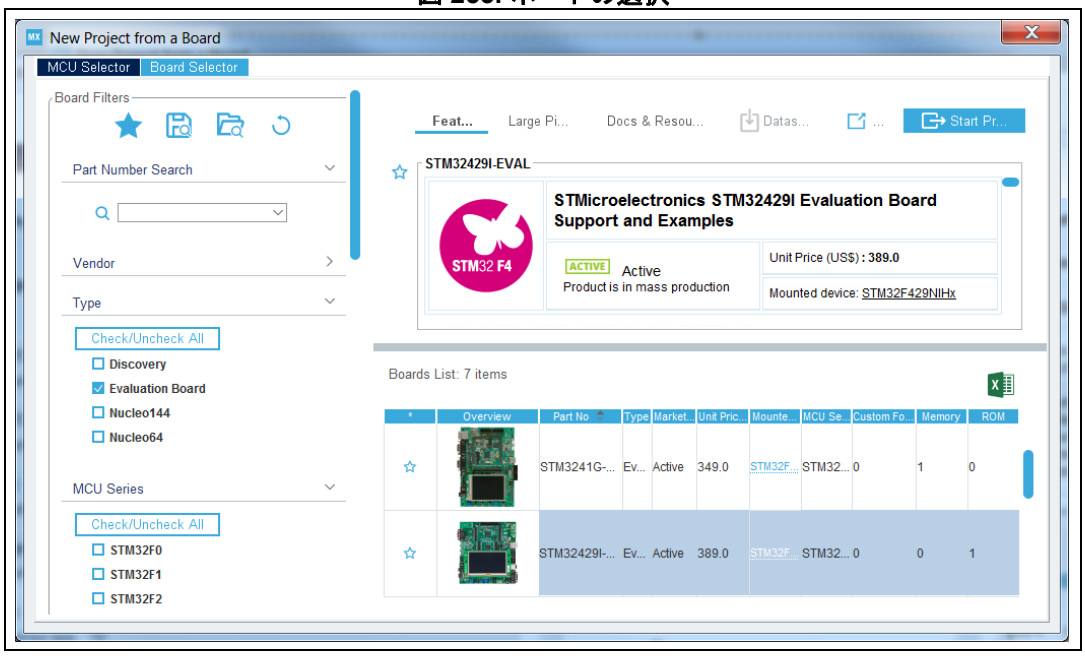
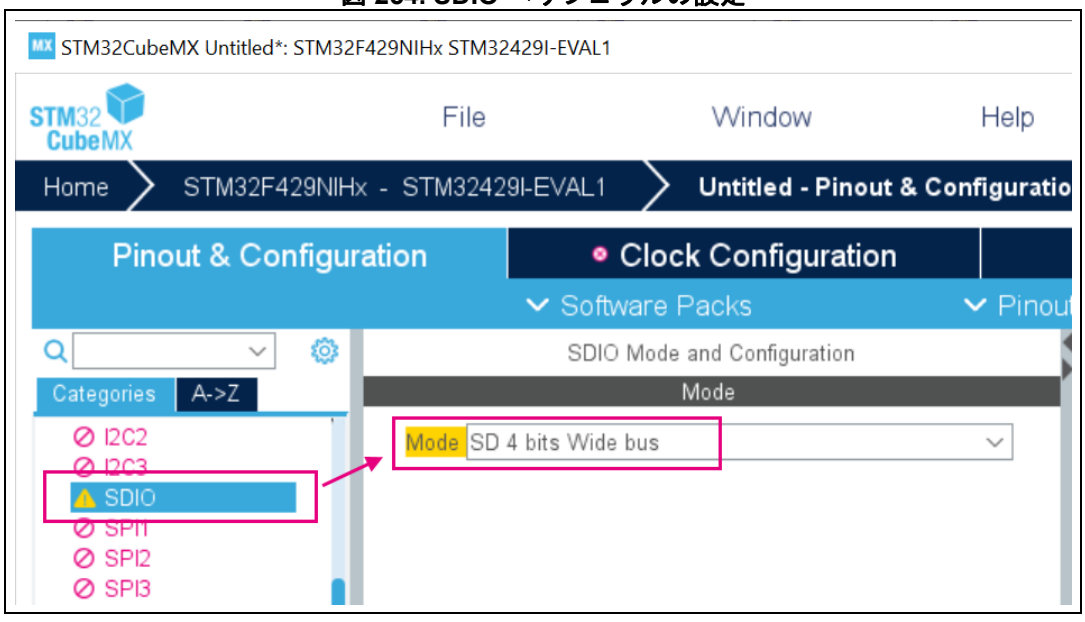


**図 253. ボードの選択**



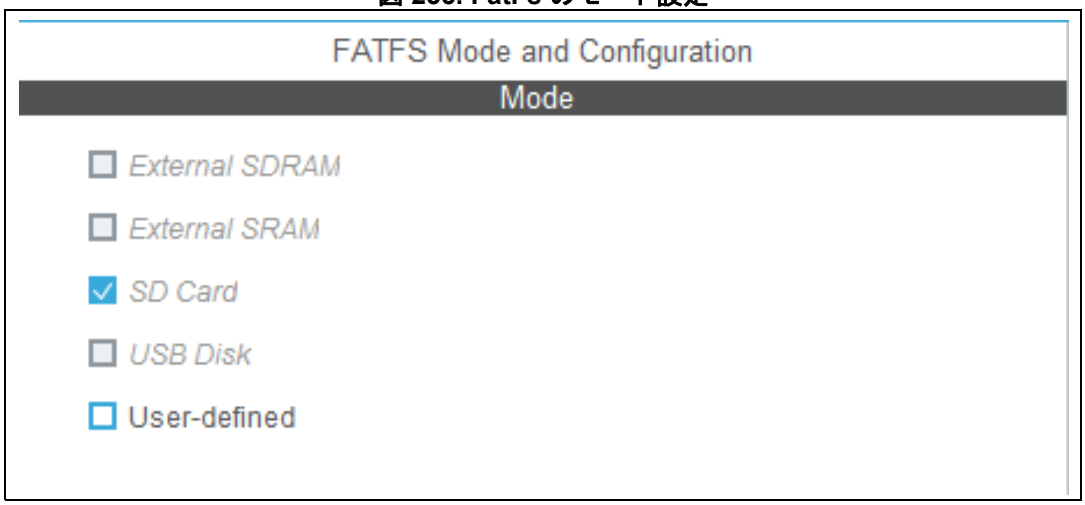

7. 左側パネルのペリフェラル･ツリーで［SDIO］ペリフェラルを展開して、［SD 4 bits Wide bus］を選択します（図 254 参照）。［Configuration］パネルの［DMA settings］タブで DMA リクエスト SDIO_RX と SDIO_TX を追加します。
8. 最後に、ペリフェラル･ツリー･パネルに戻り、［Configuration］パネルで NVIC を選択して SDIO グローバル割込みを有効にします。

**図 254. SDIO ペリフェラルの設定**



9. ［Middlewares］カテゴリで、FatFs のモードとして［SD Card］をチェックします（図 255 参照）。

**図 255. FatFs のモード設定**




右側の［Pinout］ビューで、SDIO 検出に使用するピンを GPIO 入力として有効にします。

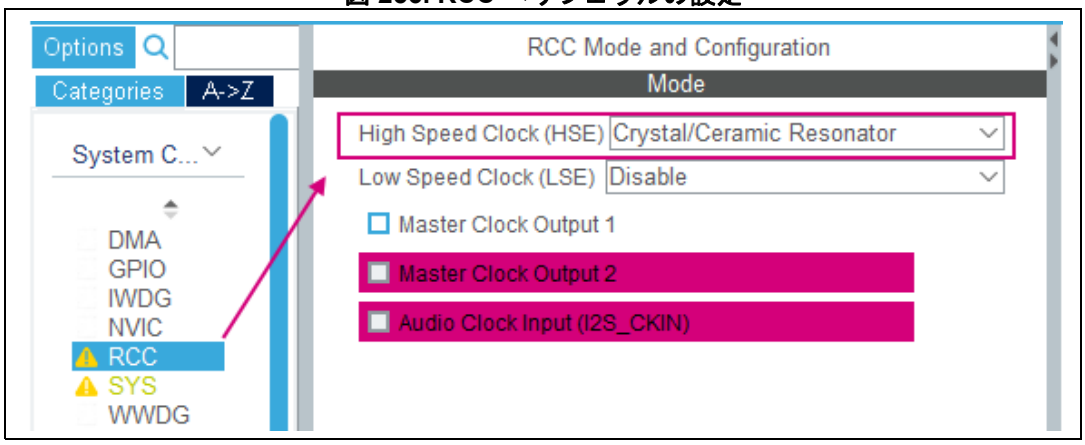
［Mode］パネルの下の［Configuration］パネルで［Platform settings］タブに移動し、先ほど有効にしたピンを使用して SD_detection を設定します。

最後に FatFs の［Advanced Settings］タブに移動し、［Use DMA template］を有効にします。

10. 次のようにクロックを設定します。

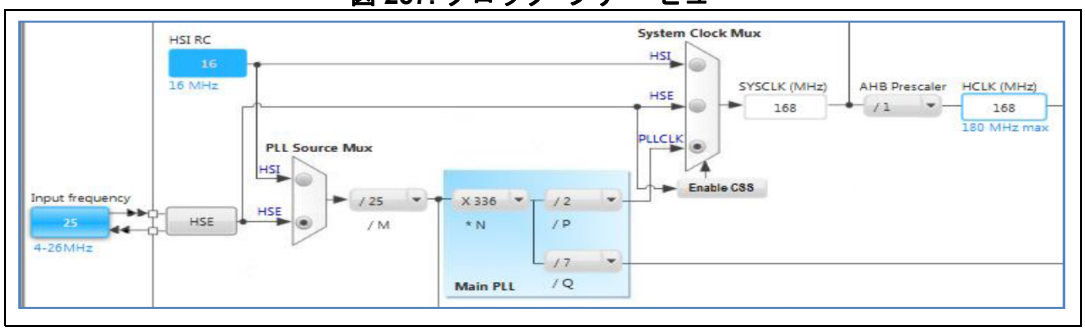
a) **［Pinout］**ビューで RCC ペリフェラルを選択します（図 256 を参照）。

**図 256. RCC ペリフェラルの設定**



b)　クロックのタブでクロック･ツリーを設定します（図 257 を参照）。

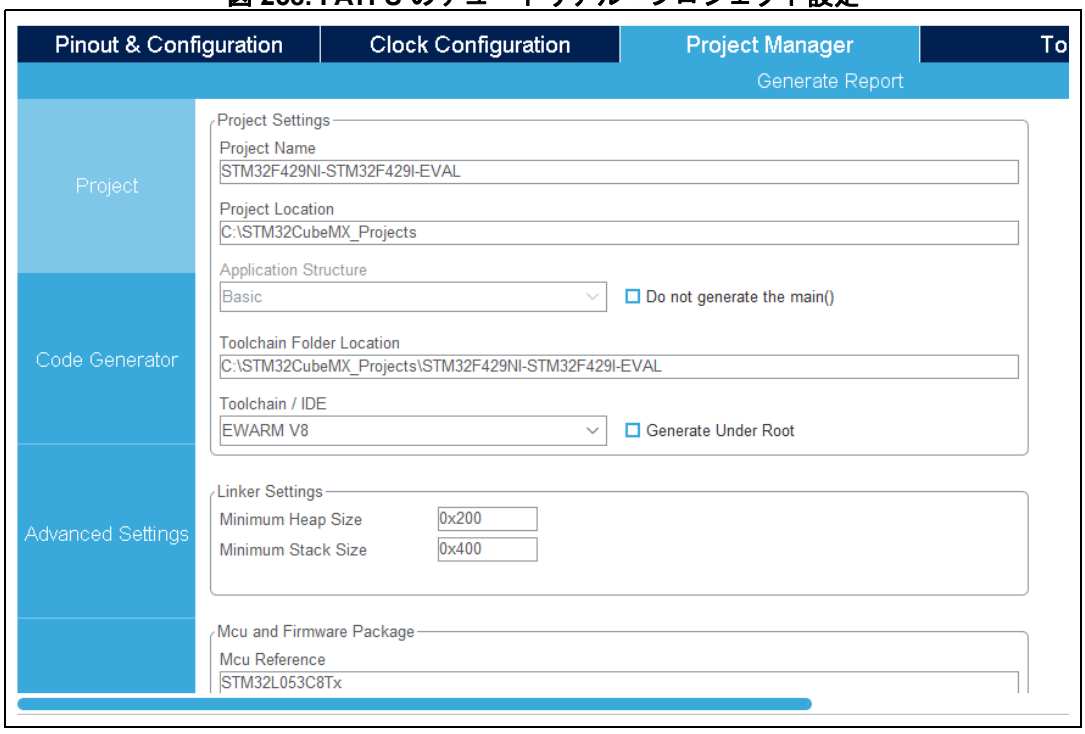
**図 257. クロック･ツリー･ビュー**




11. **[Project]** タブで、プロジェクト名と保存先フォルダを指定します。つづいて、EWARM IDE ツールチェーンを選択します。

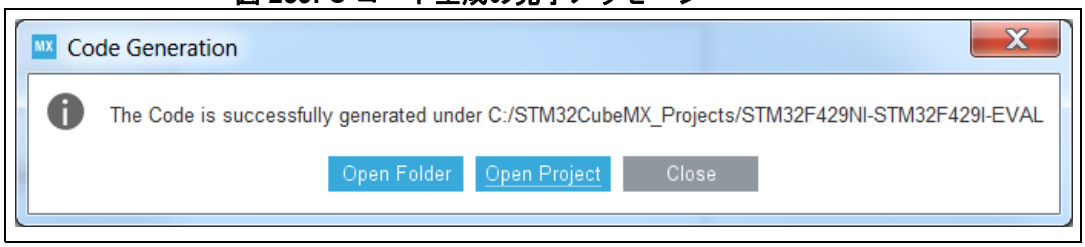
    プロジェクトのヒープとスタックのサイズは、FATFS アプリケーションの最低要件に調整できることに注意してください。

**図 258. FATFS のチュートリアル - プロジェクト設定**



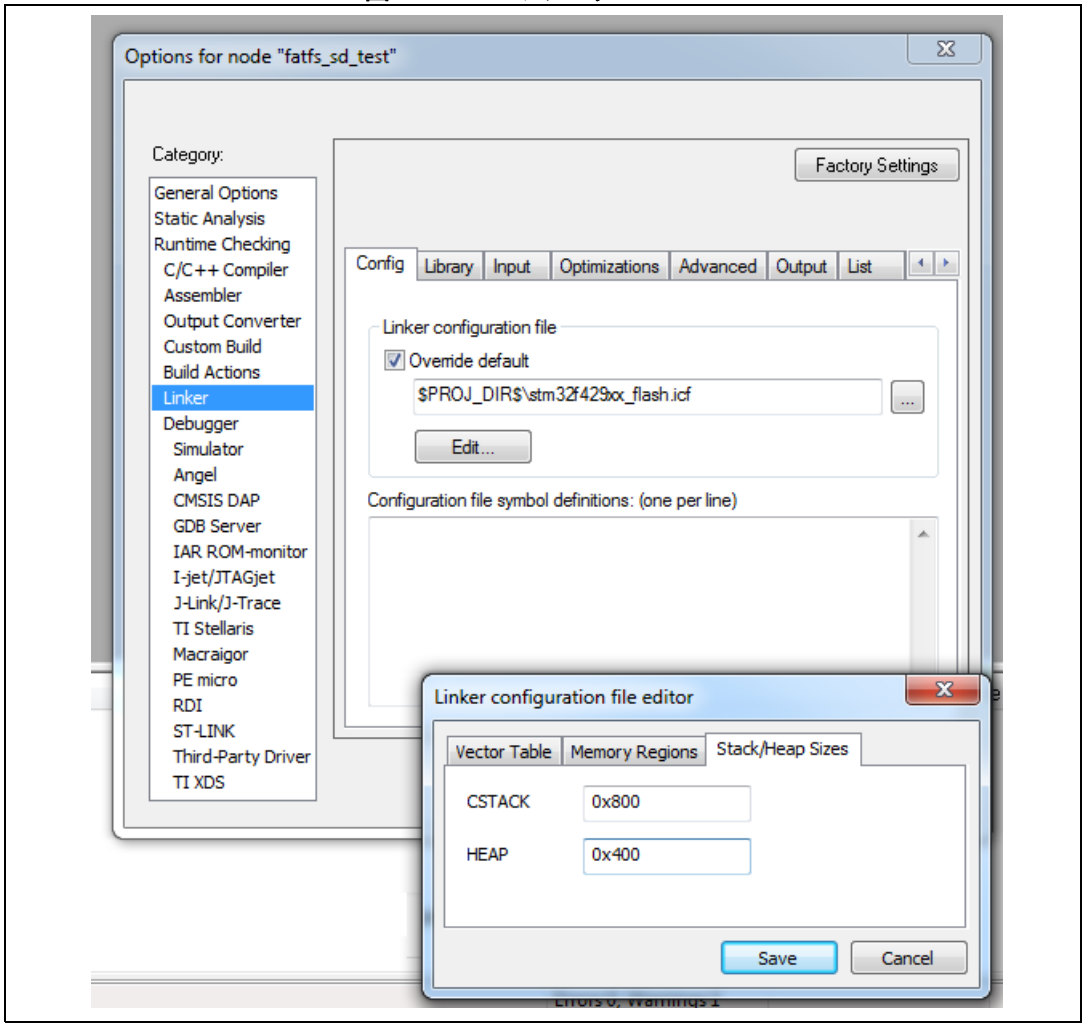

12. **[OK]** をクリックします。ツールバーのメニューで GENERATE CODE をクリックして、プロジェクトを生成します。
13. コード生成が完了したら、**[Code Generation]** ダイアログ･ウィンドウで **[Open Project]** をクリックします（図 259 を参照）。これにより、IDE で直接プロジェクトが開きます。

図 259. C コード生成の完了メッセージ




14. IDE でプロジェクト名を右クリックして［**Options**］を選択し、さらに［**Linker**］を選択して、ヒープ･サイズとスタック･サイズが十分であることを確認します。STM32CubeMX で生成したプロジェクト･フォルダの icf ファイルが使用されるように［**Override default**］をチェックします。ヒープとスタックのサイズが未設定の場合は、CubeMX のユーザ･インタフェース（[Project Manager］の［Project］タブの［Linker Settings]）で設定します（図 260 参照）。

図 260. IDE のワークスペース




**注：　MDK-Arm ツールチェーンを使用する場合、Application/MDK-ARM フォルダに移動して、startup_xx.s ファイルをダブルクリックし、ヒープ･サイズとスタック･サイズを編集して調整します。**

15. Application/User フォルダに移動します。main.c ファイルをダブルクリックして編集します。
16. このチュートリアルでは、FatFs のファイル･システム･ミドルウェアを使用して、評価ボードの SD カード上にファイルを作成して書き込みます。
    a) 起動時は、すべての LED がオフになっています。
    b) 赤色の LED がオンになった場合は、エラーが発生しています（FatFs の初期化、ファイルの読取りアクセスや書込みアクセスなどのエラー）。
    c) オレンジ色の LED がオンになった場合は、FatFs リンクが SD ドライバに正常にマウントされています。
    d) 青色の LED がオンになった場合は、ファイルが SD カードに正常に書き込まれています。
    e) 緑色の LED がオンになった場合は、ファイルが SD カードから正常に読み取られています。
17. 使用例を実装する場合、次のコードで main.c を更新します。
    a) main.c のプライベート変数を専用のユーザ･コード･セクションに挿入します。

```
/* USER CODE BEGIN PV */
/* Private variables ---------------------------------------------*/
FATFS SDFatFs;  /* File system object for SD card logical drive */
FIL MyFile;     /* File object */
const char wtext[] = "Hello World!";
static uint8_t buffer[_MAX_SS]; /* a work buffer for the f_mkfs() */
/* USER CODE END PV */
```

    b) main 関数のローカル変数を挿入します。

```
int main(void)
{

  /* USER CODE BEGIN 1 */
  FRESULT res;                /* FatFs function common result code */
  uint32_t byteswritten, bytesread; /* File write/read counts */
  char rtext[256];                /* File read buffer */
  /* USER CODE END 1 */

  /* MCU Configuration----------------------------------------*/

  /* Reset of all peripherals, Initializes the Flash interface and the
Systick. */
HAL_Init();
```

    c) main 関数の中で、初期化呼出しより後で while ループより前の箇所に、SD カードとの間で読取りと書込みを実際に実行するユーザ･コードを挿入します。

```
int main(void)
{
       ....
       MX_FATFS_Init();

         /* USER CODE BEGIN 2  */
       /*##-0- Turn all LEDs off(red, green, orange and blue)  */
          HAL_GPIO_WritePin(GPIOG, (GPIO_PIN_10 | GPIO_PIN_6 | GPIO_PIN_7 |
       GPIO_PIN_12), GPIO_PIN_SET);
       /*##-1- FatFS: Link the SD disk I/O driver ###########*/
           if(retSD == 0){
             /* success: set the orange LED on */
```

```
      HAL_GPIO_WritePin(GPIOG, GPIO_PIN_7, GPIO_PIN_RESET);
/*##-2- Register the file system object to the FatFs module ###*/
    if(f_mount(&SDFatFs, (TCHAR const*)SDPath, 0) != FR_OK){
     /* FatFs Initialization Error : set the red LED on */
        HAL_GPIO_WritePin(GPIOG, GPIO_PIN_10, GPIO_PIN_RESET);
        while(1);
      }   else    {
/*##-3- Create a FAT file system (format) on the logical drive #*/
 /* WARNING: Formatting the uSD card will delete all content on the
device */
if(f_mkfs((TCHAR const*)SDPath, FM_ANY, 0, buffer, sizeof(buffer))
!= FR_OK){
    /* FatFs Format Error : set the red LED on */
    HAL_GPIO_WritePin(GPIOG, GPIO_PIN_10, GPIO_PIN_RESET);
    while(1);
  } else {
/*##-4- Create & Open a new text file object with write access#*/
  if(f_open(&MyFile, "Hello.txt", FA_CREATE_ALWAYS | FA_WRITE) !=
FR_OK){
  /* 'Hello.txt' file Open for write Error : set the red LED on */
  HAL_GPIO_WritePin(GPIOG, GPIO_PIN_10, GPIO_PIN_RESET);
  while(1);
      } else {
 /*##-5- Write data to the text file ####################*/
   res = f_write(&MyFile, wtext, sizeof(wtext), (void
*)&byteswritten);
  if((byteswritten == 0) || (res != FR_OK)){
    /* 'Hello.txt' file Write or EOF Error : set the red LED on */
    HAL_GPIO_WritePin(GPIOG, GPIO_PIN_10, GPIO_PIN_RESET);
    while(1);
    } else {
 /*##-6- Successful open/write : set the blue LED on */
    HAL_GPIO_WritePin(GPIOG, GPIO_PIN_12, GPIO_PIN_RESET);
    f_close(&MyFile);
 /*##-7- Open the text file object with read access #*/
    if(f_open(&MyFile, "Hello.txt", FA_READ) != FR_OK){
   /* 'Hello.txt' file Open for read Error : set the red LED on */
    HAL_GPIO_WritePin(GPIOG, GPIO_PIN_10, GPIO_PIN_RESET);
    while(1);
    } else {
 /*##-8- Read data from the text file #########*/
    res = f_read(&MyFile, rtext, sizeof(wtext), &bytesread);
    if((byteswritten == 0)|| (res != FR_OK)){
  /* 'Hello.txt' file Read or EOF Error : set the red LED on */
    HAL_GPIO_WritePin(GPIOG, GPIO_PIN_10, GPIO_PIN_RESET);
    while(1);
    } else {
  /* Successful read : set the green LED On */
    HAL_GPIO_WritePin(GPIOG, GPIO_PIN_6, GPIO_PIN_RESET);
 /*##-9- Close the open text file #################*/
    f_close(&MyFile);
```

```
        }}}}}}}
       /*##-10- Unlink the micro SD disk I/O driver #########*/
         FATFS_UnLinkDriver(SDPath);

       /* USER CODE END 2 */

       /* Infinite loop */
       /* USER CODE BEGIN WHILE */
while (1)
```

# 13 チュートリアル 3 - 消費電力計算機能の使用による 組込みアプリケーションの消費電力の最適化など

## 13.1 チュートリアルの概要

このチュートリアルは、STM32CubeMX の消費電力計算機能（[Power Consumption Calculator]）の特徴、およびアプリケーション･シーケンスに対する省電力手法の効果を評価する際にこの機能で得られる利点を中心に取り上げています。

アプリケーションの消費電力を削減するうえで重要な考慮事項を次に示します。

- 動作電圧の低減
- エネルギー消費モードを使用する時間の短縮

  低消費電力と性能との最適な妥協点を実現する設定は、開発段階で選択する必要があります。
- 非アクティブ･モードと省電力モードを使用する時間の最大化
- 最適なクロック設定の使用

  コアは常時比較的速い速度で動作する必要があります。マイクロコントローラが所定の動作を実行するためにアクティブな動作モードを長時間継続する場合、動作周波数を低くすると、エネルギー消費量が増加するからです。
- 現在のアプリケーション状態に関連するペリフェラルのみを有効にして、他のペリフェラルにはクロック･ゲーティングを適用
- 低電力機能を備えたペリフェラルがある場合は、可能な限りそれを使用（I2C によるマイクロコントローラのウェイクアップなど）
- 状態遷移数の最小化
- コード実行中のメモリ･アクセスの最適化
  - Flash･メモリより RAM からのコード実行を優先
  - 可能であれば CPU の周波数と Flash メモリの動作周波数との一致を図り、ウェイト･ステートの発生を排除することを検討

次のチュートリアルでは、消費電力を最小限に抑えて長いバッテリ寿命を実現できるようにアプリケーションを調整するうえで、STM32CubeMX の消費電力計算機能が効果的な様子について説明します。

**注：** **消費電力計算機能では、I/O の動的な消費電流、および消費電流に影響する可能性がある外部ボード･コンポーネントが考慮されていません。それらを考慮するために、消費電力値を指定できる［Additional Consumption］フィールドが用意されています。**

## 13.2 アプリケーション例の説明

このアプリケーションは、NUCLEO-L476RG ボードを使用して設計されています。このボードは STM32L476RGTx を採用し、2.4 V バッテリで動作します。

このアプリケーションの主な目的は、ADC を測定し、変換結果を UART 経由で転送することです。ここでは以下を使用します。

- 複数の省電力モード : ローパワー RUN、ローパワー SLEEP、SLEEP、STOP、および STANDBY
- 複数のペリフェラル : USART、DMA、タイマ、COMP、DAC、および RTC
  - RTC を使用して、カレンダを実行しながら、指定時間経過後に CPU を STANDBY からウェイクアップします。
  - ADC 測定値を DMA で ADC からメモリに転送します。
  - USART を DMA とともに使用して、仮想 COM ポート経由でデータを送受信し、CPU を STOP モードからウェイクアップします。

このような複雑なアプリケーションを最適化するプロセスとして、まず機能の実現のみを考慮したシーケンスを記述し、次に STM32L476RG マイクロコントローラの省電力機能を段階的に導入します。

## 13.3 消費電力計算機能の使用

### 13.3.1 消費電力シーケンスの作成

次の手順に従って、シーケンスを作成します（図 261 を参照）。

1. STM32CubeMX を起動します。
2. **[New Project]** をクリックして、**[Board Selector]** タブから Nucleo-L476RG ボードを選択します。
3. **[Power Consumption Calculator]** タブをクリックして、[Power Consumption Calculator] ビューを選択します。最初のシーケンスは参考として作成されます。
4. 合計消費電流が最小になるように、このシーケンスを変更します。このためには、次の操作を行います。
   a) $V_{DD}$ 電源として 2.4 V を選択します。この値は、手順を追って調整できます（図 262 を参照）。
   b) バッテリとして Li-MnO2 (CR2032) を選択します。このステップは省略してもかまいません。バッテリ・タイプは後で変更できます（図 262 を参照）。

図 261. 消費電力計算の例

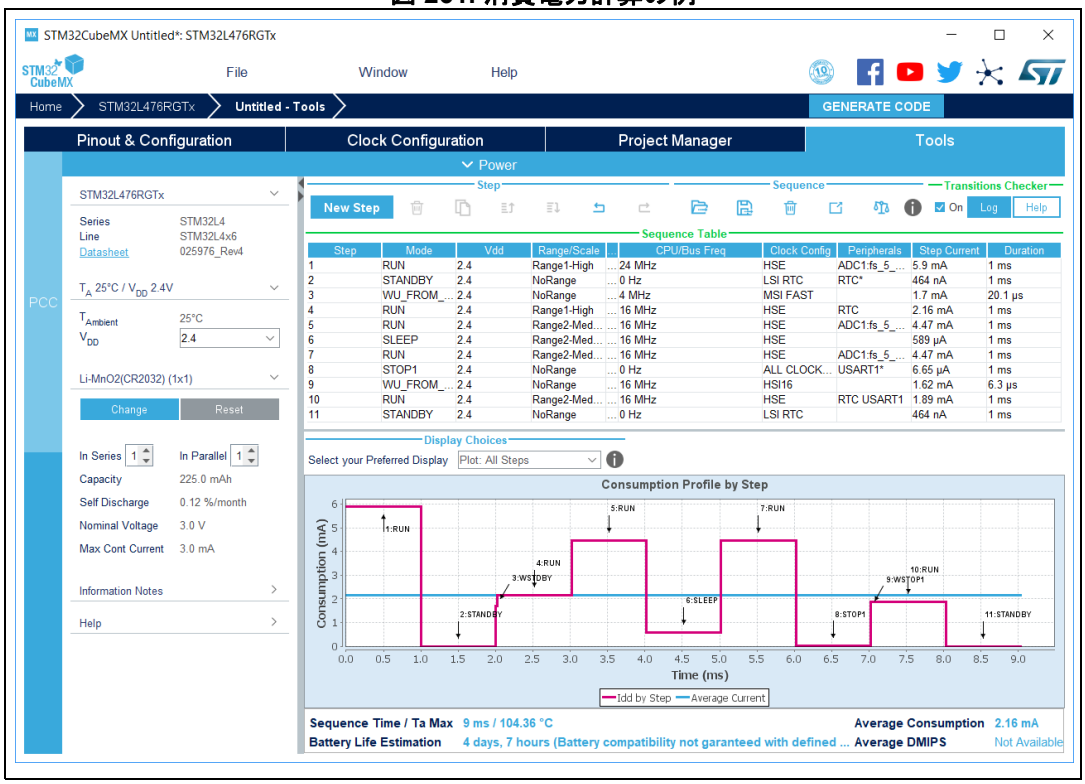
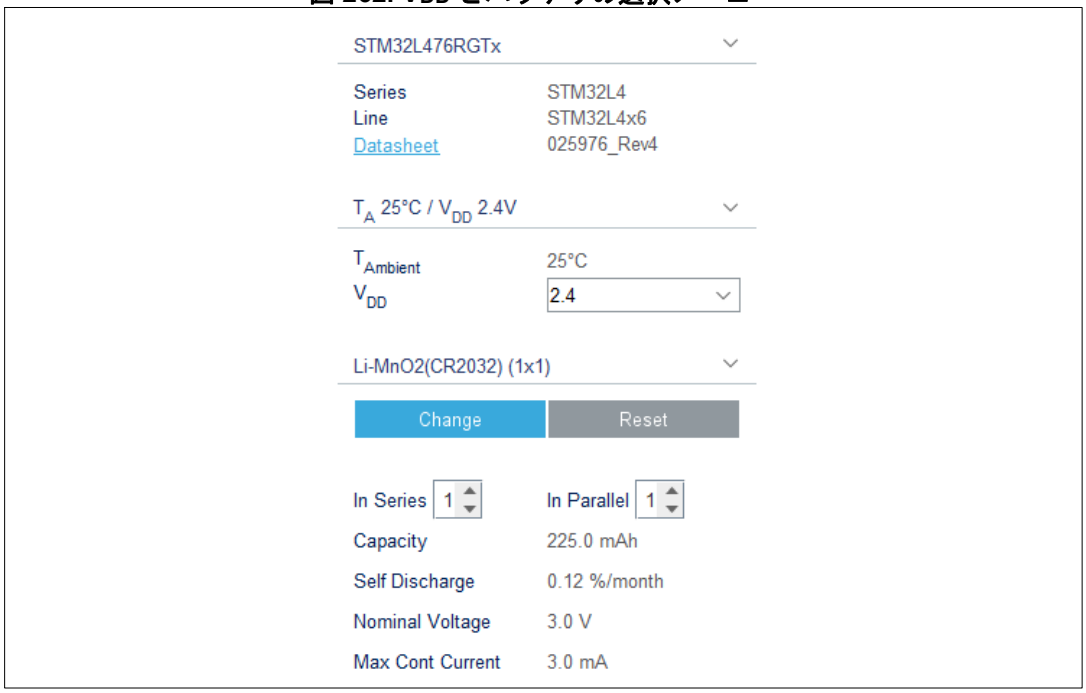



図 262. $V_{DD}$ とバッテリの選択メニュー



5. **[Transition checker]** を有効にして、シーケンスが有効であることを確認します（図 262 を参照）。このオプションを使用すると、STM32L476RG に実装された許容範囲のトランジションにこのシーケンスが従っていることを確認できます。
6. **[Add]** ボタンをクリックして、図 262 で記述したシーケンスに一致するステップを追加します。
   - デフォルトでは、これらのステップごとの所要時間は 1 ミリ秒です。なお、製品データシートに記載されているトランジション時間を使用してプリセットされているウェイクアップ・トランジションの時間は除きます（図 263 を参照）。
   - 消費電力が不明か無視できるペリフェラルは「*」記号で示されます（図 263 を参照）。

**図 263. [Sequence Table]**

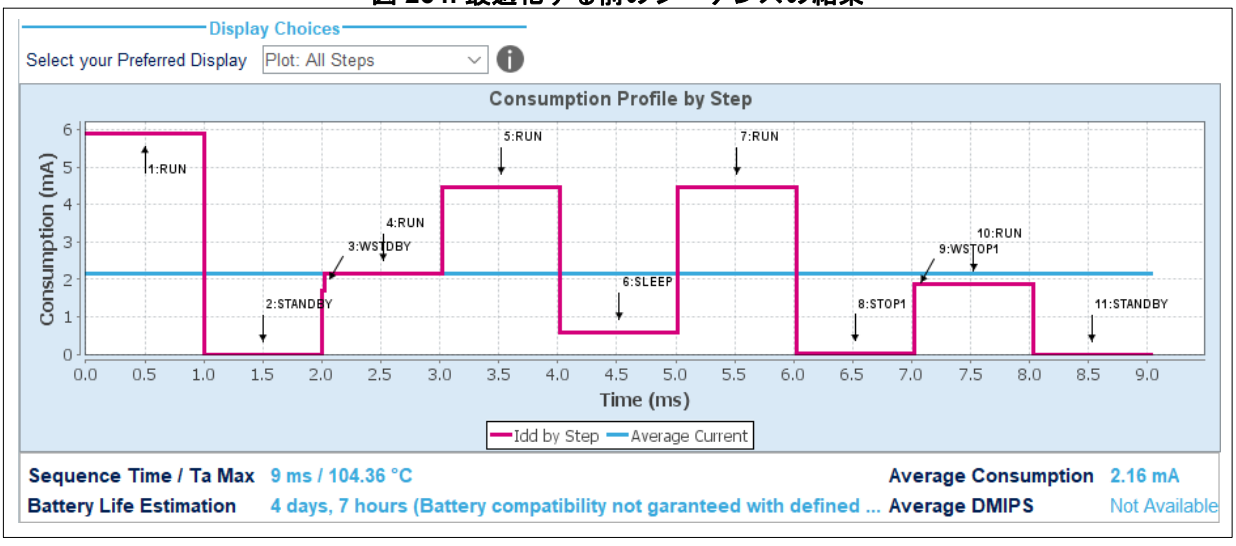
Sequence Table

| Step | Mode | Vdd | Range/Scale | ... | CPU/Bus Freq | Clock Config | Peripherals | Step Current | Duration |
|---|---|---|---|---|---|---|---|---|---|
| 1 | RUN | 2.4 | Range1-High | ... | 24 MHz | HSE | ADC1:fs_5_... | 5.9 mA | 1 ms |
| 2 | STANDBY | 2.4 | NoRange | ... | 0 Hz | LSI RTC | RTC* | 464 nA | 1 ms |
| 3 | WU_FROM_... | 2.4 | NoRange | ... | 4 MHz | MSI FAST | | 1.7 mA | 20.1 µs |
| 4 | RUN | 2.4 | Range1-High | ... | 16 MHz | HSE | RTC | 2.16 mA | 1 ms |
| 5 | RUN | 2.4 | Range2-Med... | ... | 16 MHz | HSE | ADC1:fs_5_... | 4.47 mA | 1 ms |
| 6 | SLEEP | 2.4 | Range2-Med... | ... | 16 MHz | HSE | | 589 µA | 1 ms |
| 7 | RUN | 2.4 | Range2-Med... | ... | 16 MHz | HSE | ADC1:fs_5_... | 4.47 mA | 1 ms |
| 8 | STOP1 | 2.4 | NoRange | ... | 0 Hz | ALL CLOCK... | USART1* | 6.65 µA | 1 ms |
| 9 | WU_FROM_... | 2.4 | NoRange | ... | 16 MHz | HSI16 | | 1.62 mA | 6.3 µs |
| 10 | RUN | 2.4 | Range2-Med... | ... | 16 MHz | HSE | RTC USART1 | 1.89 mA | 1 ms |
| 11 | STANDBY | 2.4 | NoRange | ... | 0 Hz | LSI RTC | | 464 nA | 1 ms |

7. **[Save]** ボタンをクリックして、このシーケンスを SequenceOne として保存します。

アプリケーションの消費電力プロファイルが作成されます。シーケンス全体の平均消費電流は 9 ミリ秒で 2.01 mA であり、バッテリ寿命はわずか 4 日であることが示されます（図 264 を参照）。

**図 264. 最適化する前のシーケンスの結果**



### 13.3.2 アプリケーションの消費電力の最適化

ここからさまざまな措置を講じて、合計消費電力とバッテリ寿命を最適化します。これらの措置は、上図のステップ1、4、5、6、7、8、および 10 で実行します。

次の図は、左側が当初のステップ、右側がさまざまな最適化で更新したステップを示します。

#### ステップ 1 （RUN）

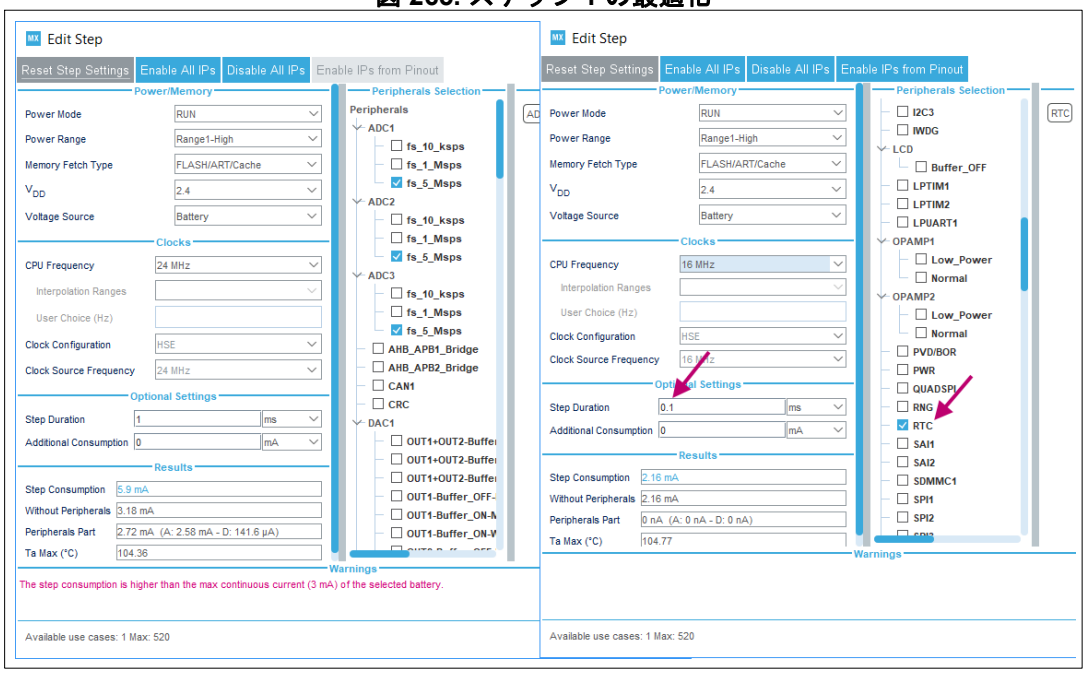
- 所見

  すべてのペリフェラルが有効になっていますが、アプリケーションでは RTC のみを必要としています。

- 措置
  - 動作周波数を低くします。
  - RTC ペリフェラルのみを有効にします。
  - 平均消費電流を削減するために、このモードを使用する時間を短縮します。
- 結果

  電流は 9.05 mA から 2.16 mA に減少しています（図 265 を参照）。

**図 265. ステップ 1 の最適化**




#### ステップ 4 （RUN、RTC）

- アクション

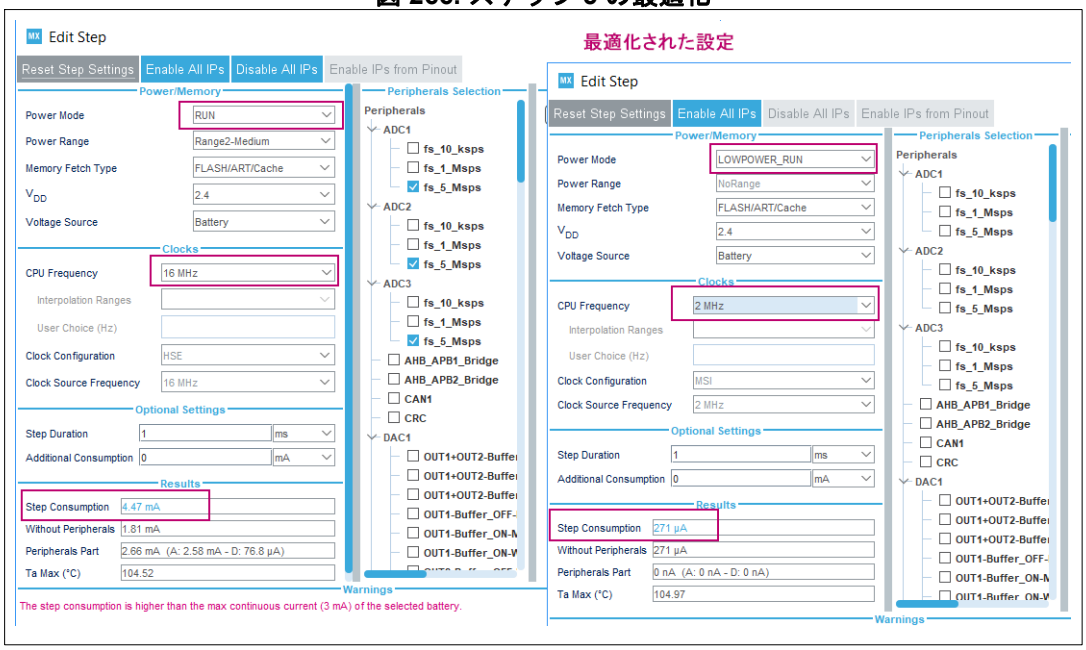
  このモードを使用する時間を 0.1 ミリ秒に短縮します。

### ステップ 5 （RUN、ADC、DMA、RTC）

- 措置
  - 省電力 RUN モードに切り替えます。
  - 動作周波数を低くします。
- 結果

  消費電流は 6.17 mA から 271 µA に減少しています（図 266 参照）。

**図 266. ステップ 5 の最適化**

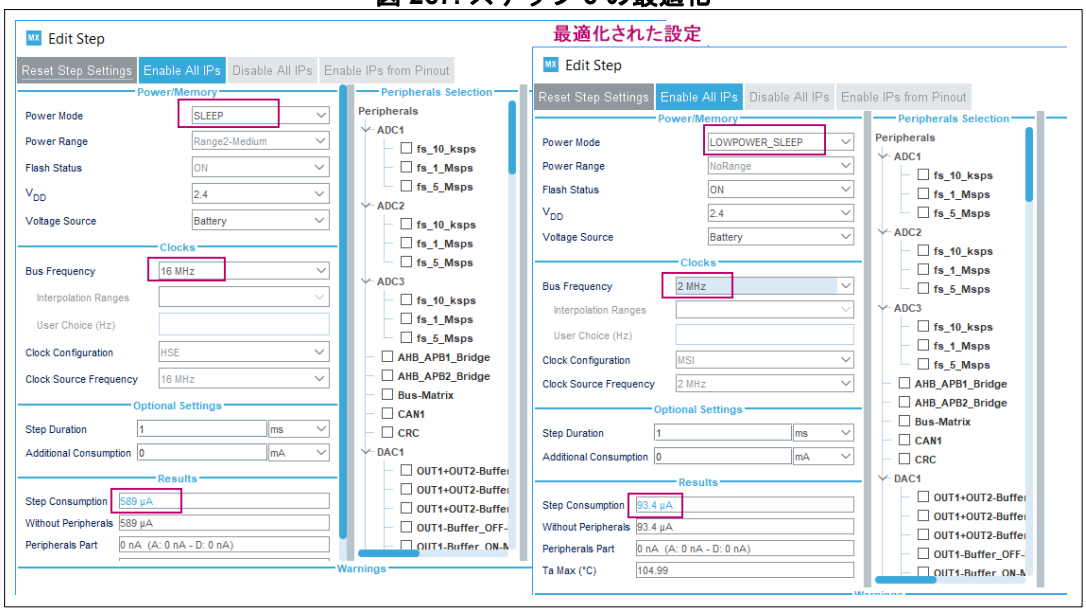


### ステップ 6 （SLEEP、DMA、ADC、RTC）

- 措置
  - 省電力 SLEEP モード（BAM モード）に切り替えます。
  - 動作周波数を 2 MHz に引き下げます。
- 結果

  消費電流は 703 µA から 93 µA に減少しています（図 267 参照）。

**図 267. ステップ 6 の最適化**

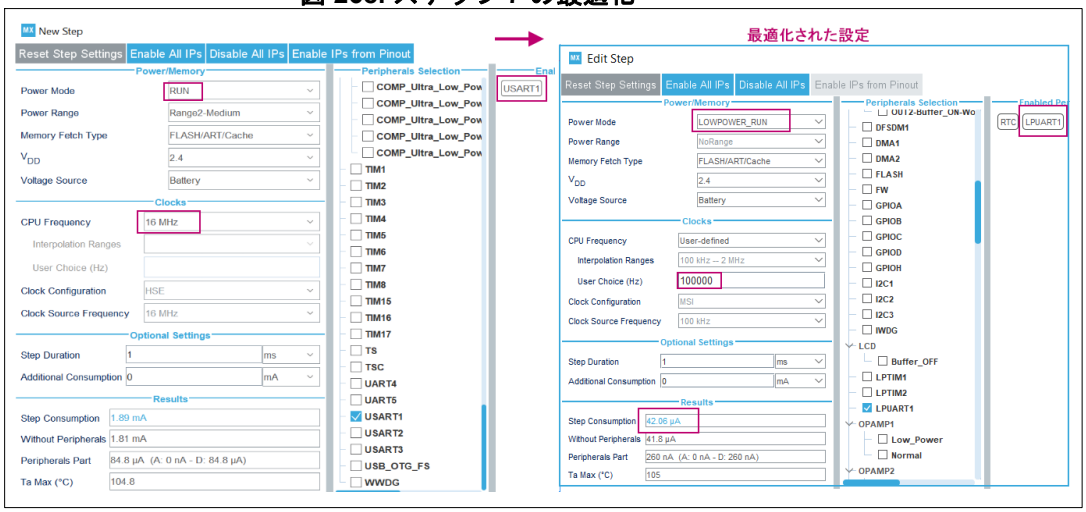


### ステップ 7 （RUN、DMA、RTC、USART）

- 措置
  - 省電力 RUN モードに切り替えます。
  - 電力効率の良い LPUART ペリフェラルを使用します。
  - 補完機能を使用して動作周波数を 1 MHz に引き下げます。
- 結果

  消費電流は 1.92 µA から 42 µA に減少しています（図 268 参照）。

**図 268. ステップ 7 の最適化**

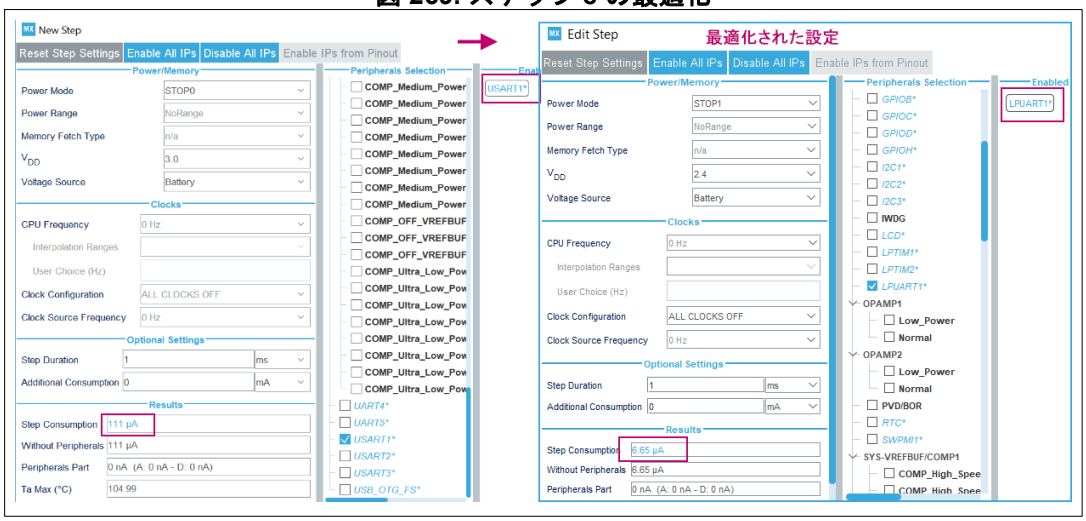


### ステップ 8 （STOP 0、USART）

- 措置
  - STOP 1 の省電力モードに切り替えます。
  - 電力効率の良い LPUART ペリフェラルを使用します。
- 結果

  消費電流が減少しています（図 269 参照）。

**図 269. ステップ 8 の最適化**

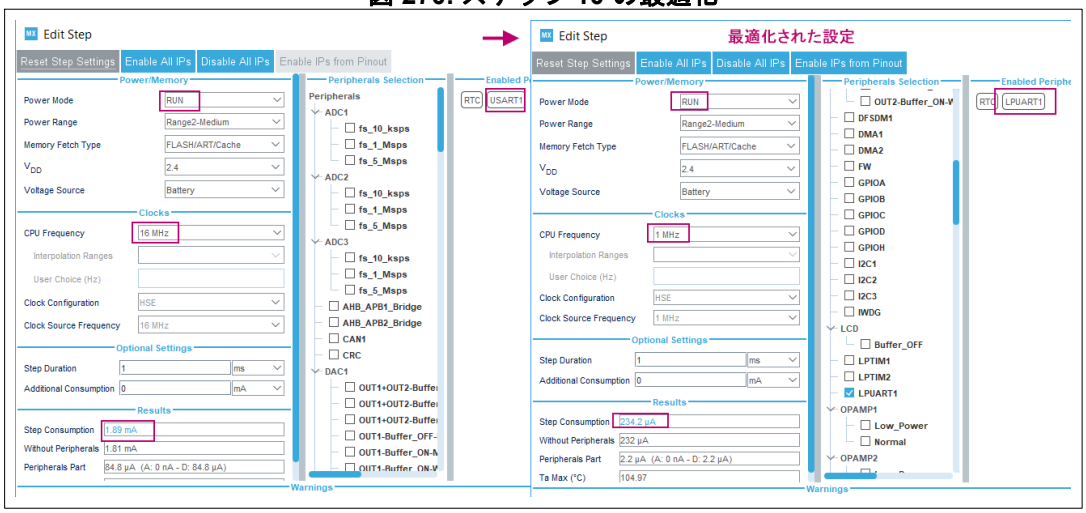



### ステップ 10 （RTC、USART）

- 措置
  - 電力効率の良い LPUART ペリフェラルを使用します。
  - 動作周波数を 1 MHz に引き下げます。
- 結果

  消費電流は 1.89 mA から 234 µA に減少しています（図 270 を参照）。

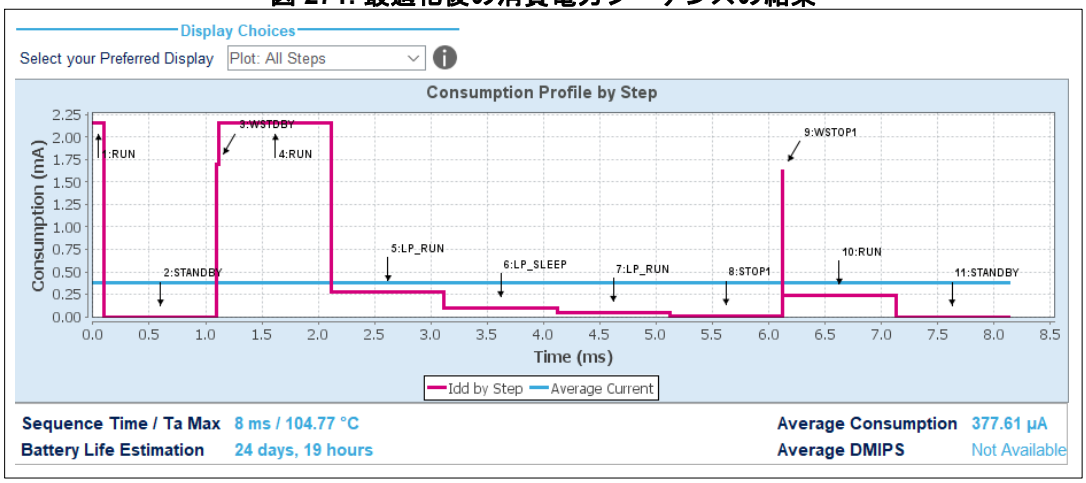
**図 270. ステップ 10 の最適化**



シーケンス全体の結果については、図 271 を参照してください。所要時間は 7 ミリ秒、バッテリ寿命は約 2 ヶ月、平均消費電流は 155.25 µA になっています。

**[Compare]** ボタンを使用して、SequenceOne.pcs として保存した当初の結果と現在の結果を比較します。

**図 271. 最適化後の消費電力シーケンスの結果**



# 14 チュートリアル 4 - STM32L053xx Nucleo ボードとの UART 通信の例

このチュートリアルは、STM32CubeMX を使用して、NUCLEO-L053R8 ボードで使用する UART シリアル通信アプリケーションを作成する方法を示すことを目的としています。

この例では、Windows PC が必要です。シリアルデータ通信およびマイクロコントローラ上でのファームウェアのダウンロードとデバッグの両方に ST-Link USB コネクタを使用します。ボードとコンピュータは、タイプ A から mini-B に変換するUSB ケーブルで接続する必要があります。USART2 ペリフェラルは PA2 ピンと PA3 ピンを使用します。これらは、ST-Link コネクタに接続されています。さらに、ST-Link の仮想 COM ポート経由で PC と通信するために USART2 が選択されています。仮想通信ポート経由でボードから受信したメッセージを表示するために、Tera Term などのシリアル通信クライアントを PC にインストールする必要があります。

## 14.1 チュートリアルの概要

チュートリアル 4 では、次のステップを実行します。

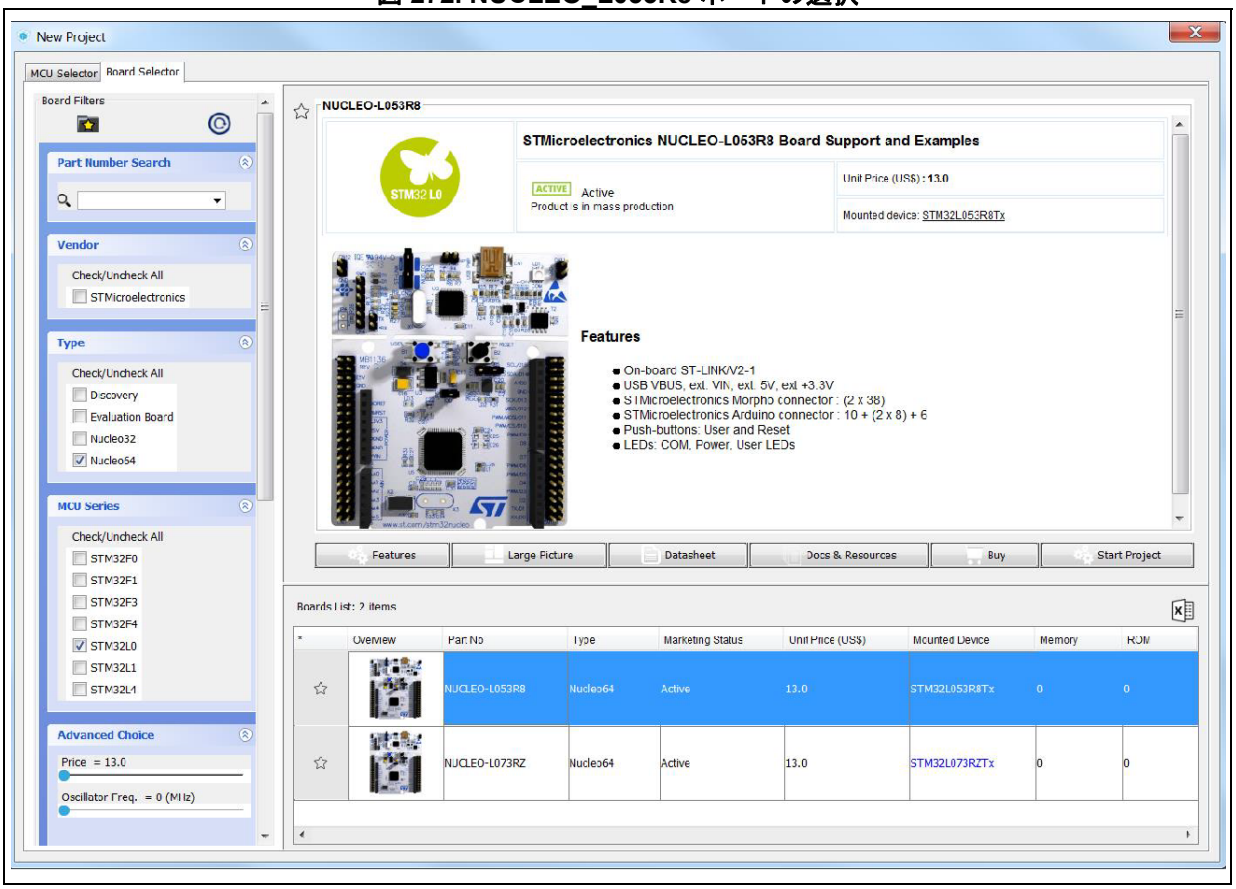
1. **[New Project]** メニューで NUCLEO-L053R8 ボードを選択
2. 必要な機能（デバッグ、USART、タイマ）を **[Pinout]** ビューで選択し、ペリフェラルの動作モードおよび関連する信号をピンに割当て
3. **[Clock Configuration]** ビューでマイクロコントローラのクロック·ツリーを設定
4. **[Configuration]** ビューでペリフェラルのパラメータを設定
5. **[Project Manager]** メニューでプロジェクトを設定し、そのプロジェクトを生成（初期化コードのみ）
6. UART 通信例に対応するユーザ·アプリケーション·コードでプロジェクトを更新
7. プロジェクトをコンパイルしてボード上で実行
8. PC 上でシリアル通信クライアントとして Tera Term ソフトウェアを設定
9. PC 上で結果を表示

## 14.2 新しい STM32CubeMX プロジェクトの作成と Nucleo ボードの選択

これを実行するには、次の手順に従います。

1. メイン·メニュー·バーから、**[File]** → **[New project]** を選択します。**[New Project]** ウィンドウが開きます。
2. **[Board selector]** タブに移動して、STM32L0 シリーズをフィルタで絞り込みます。
3. ［NUCLEO-L053R8］を選択して **[OK]** をクリックし、このボードを STM32CubeMX のユーザ·インタフェースにロードします（図 272 を参照）。

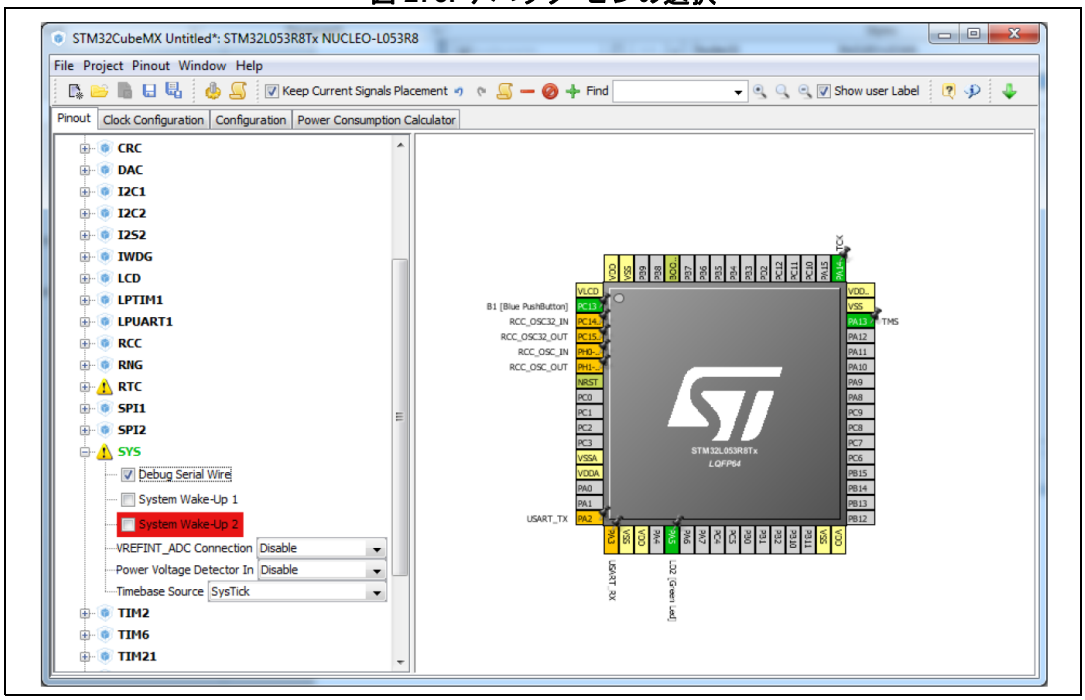
図 272. NUCLEO_L053R8 ボードの選択



## 14.3 ［Pinout］ビューでの機能の選択

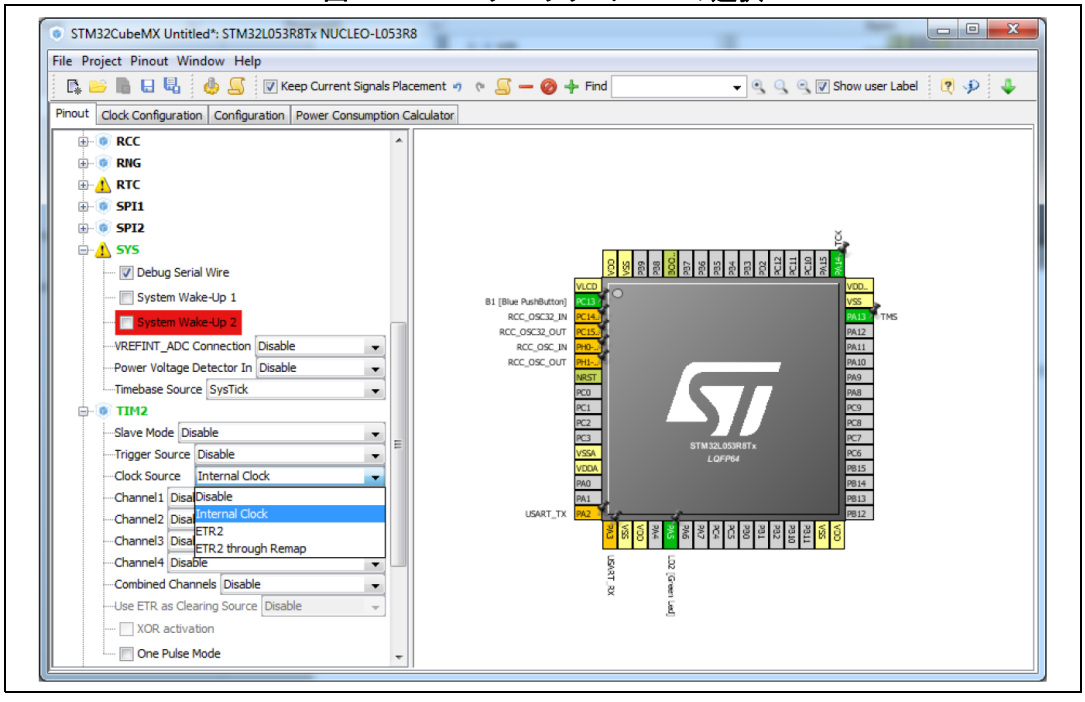
1. ［SYS］で［Debug Serial Wire］をチェックします（図 273 を参照）。

**図 273. デバッグ･ピンの選択**



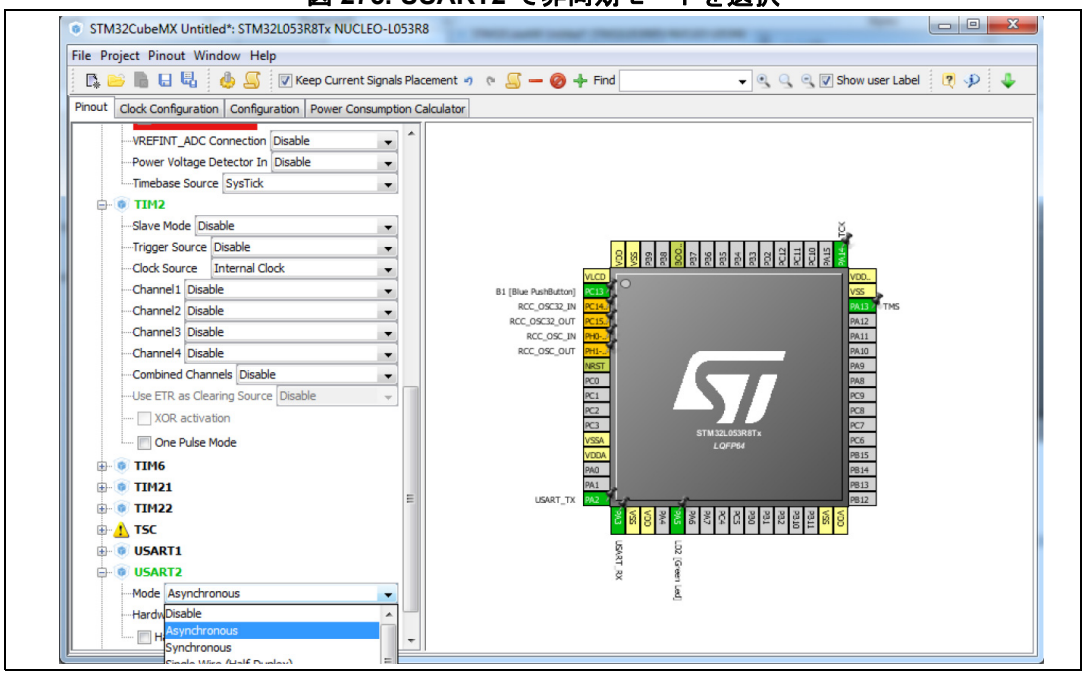

2. ［TIM2］ペリフェラルで［Clock Source］として［Internal Clock］を選択します（図 274 を参照）。

**図 274. TIM2 クロック･ソースの選択**



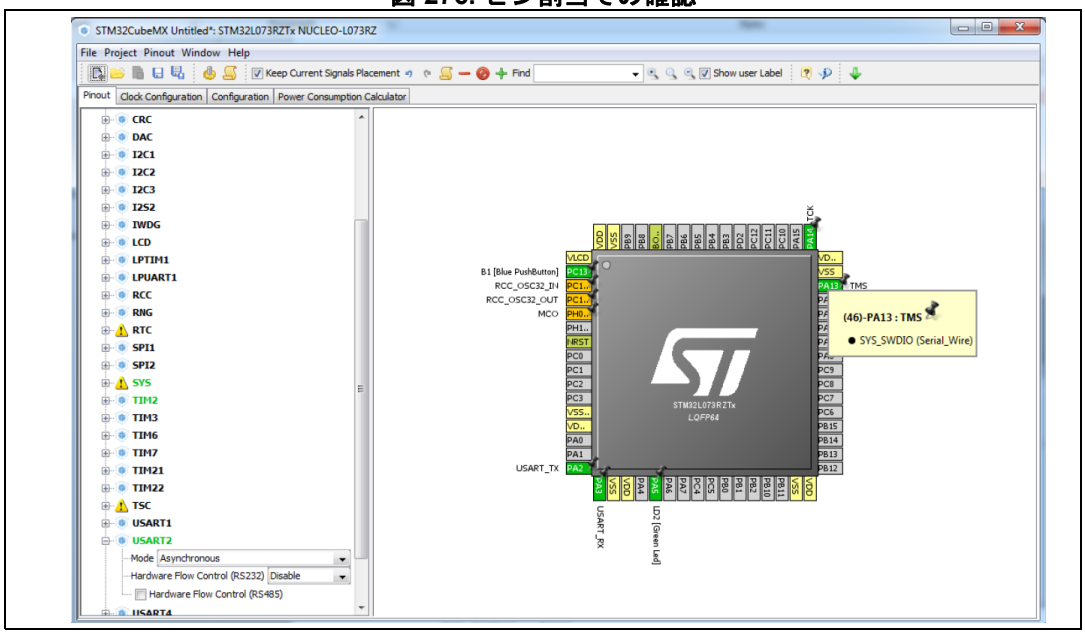
3. ［USART2］で［Asynchronous］モードを選択します（図 275 を参照）。

**図 275. USART2 で非同期モードを選択**




4. 次の各信号が正しくピンに割り当てられていることを確認します（図 276 を参照）。
   - SYS_SWDIO → PA13
   - TCK → PA14
   - USART_TX → PA2
   - USART_RX → PA3

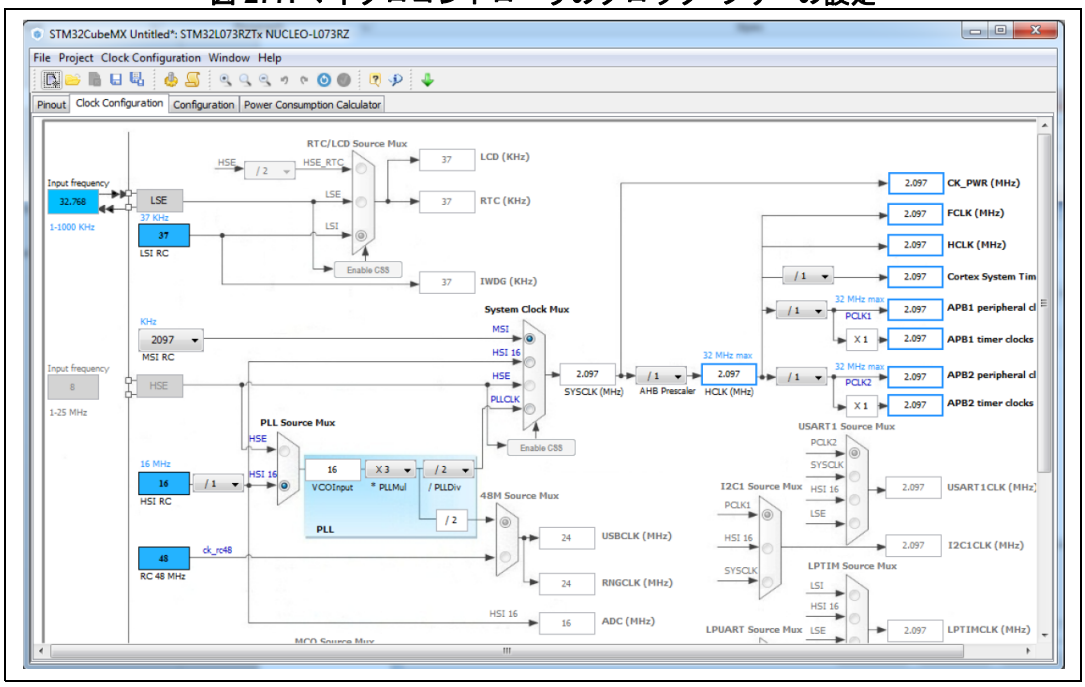
**図 276. ピン割当ての確認**



## 14.4 ［Clock Configuration］ビューでのマイクロコントローラのクロック･ツリーの設定

1. ［**Clock Configuration**］タブに移動しますが、MSI を入力クロックとして、2.097 MHz の HCLK を使用するために、このタブでの設定は変更しません（図 277 を参照）。

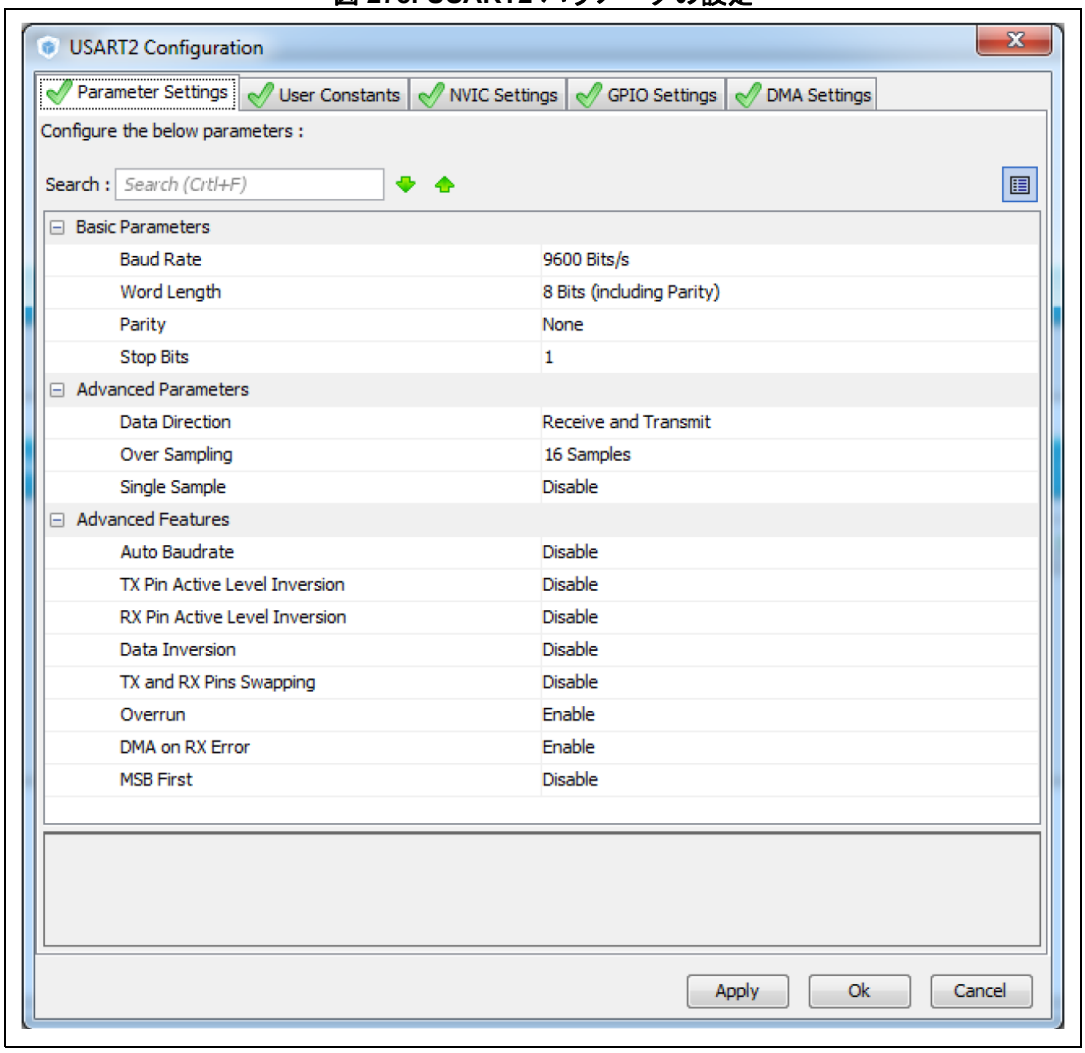
**図 277. マイクロコントローラのクロック･ツリーの設定**



## 14.5 ［Configuration］ビューでのペリフェラルのパラメータの設定

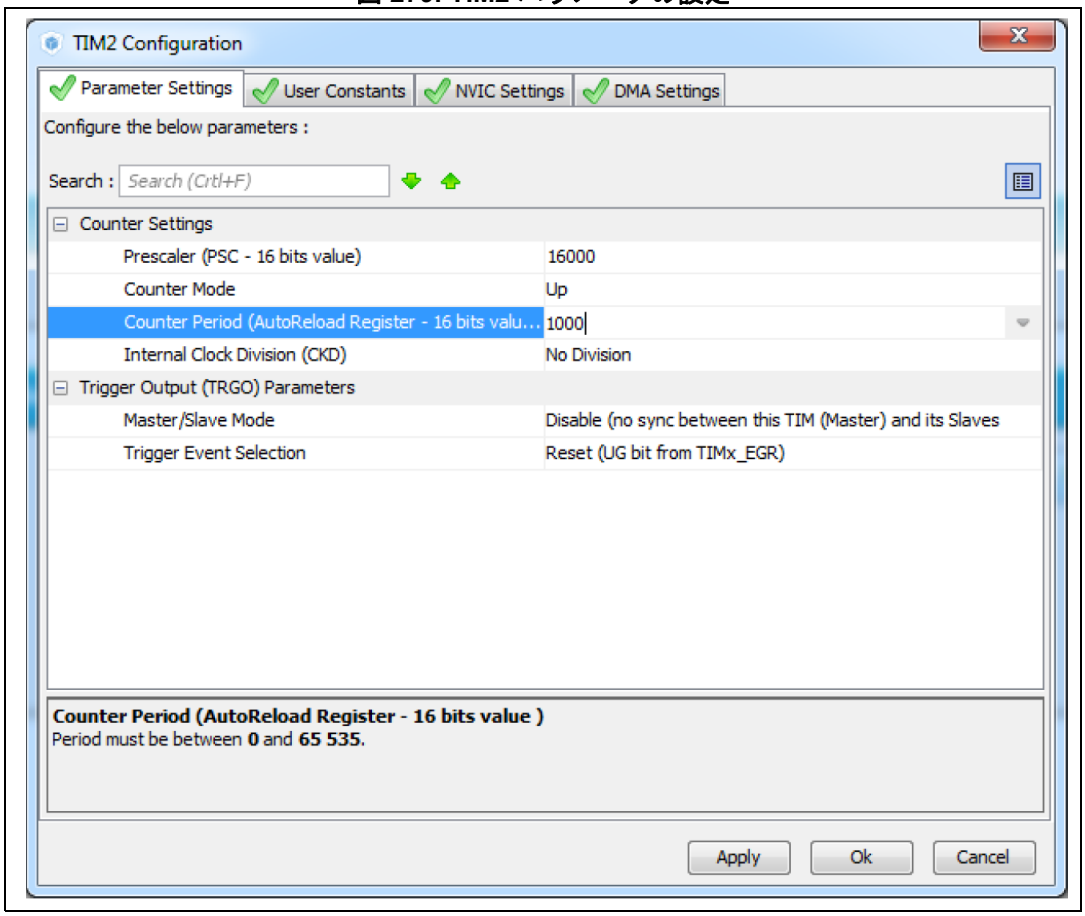
1. **[Configuration]** タブで **[USART2]** をクリックして、このペリフェラルの **[Parameter Settings]** ウィンドウを開き、［Baud Rate］を「9600」に設定します。［Data Direction］が［Receive and Transmit］に設定されていることを確認します（図 278 を参照）。
2. **［OK］**をクリックして変更を適用し、ウィンドウを閉じます。

**図 278. USART2 パラメータの設定**



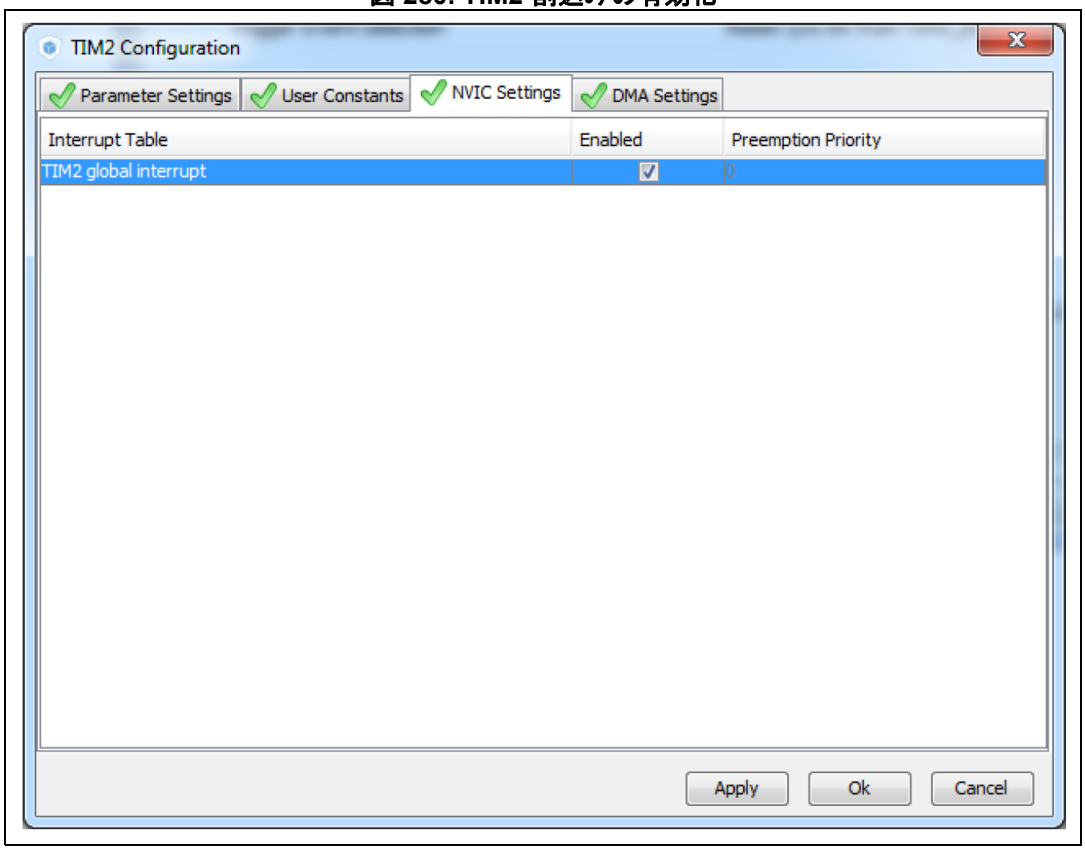
3. **[TIM2]** をクリックして、プリスケーラを 16000、ワード長を 8 bit、カウンタ周期を 1000 に変更します（図 279 を参照）。

**図 279. TIM2 パラメータの設定**



4. **［NVIC Settings］**タブで TIM2 グローバル割込みを有効にします（図 280 を参照）。

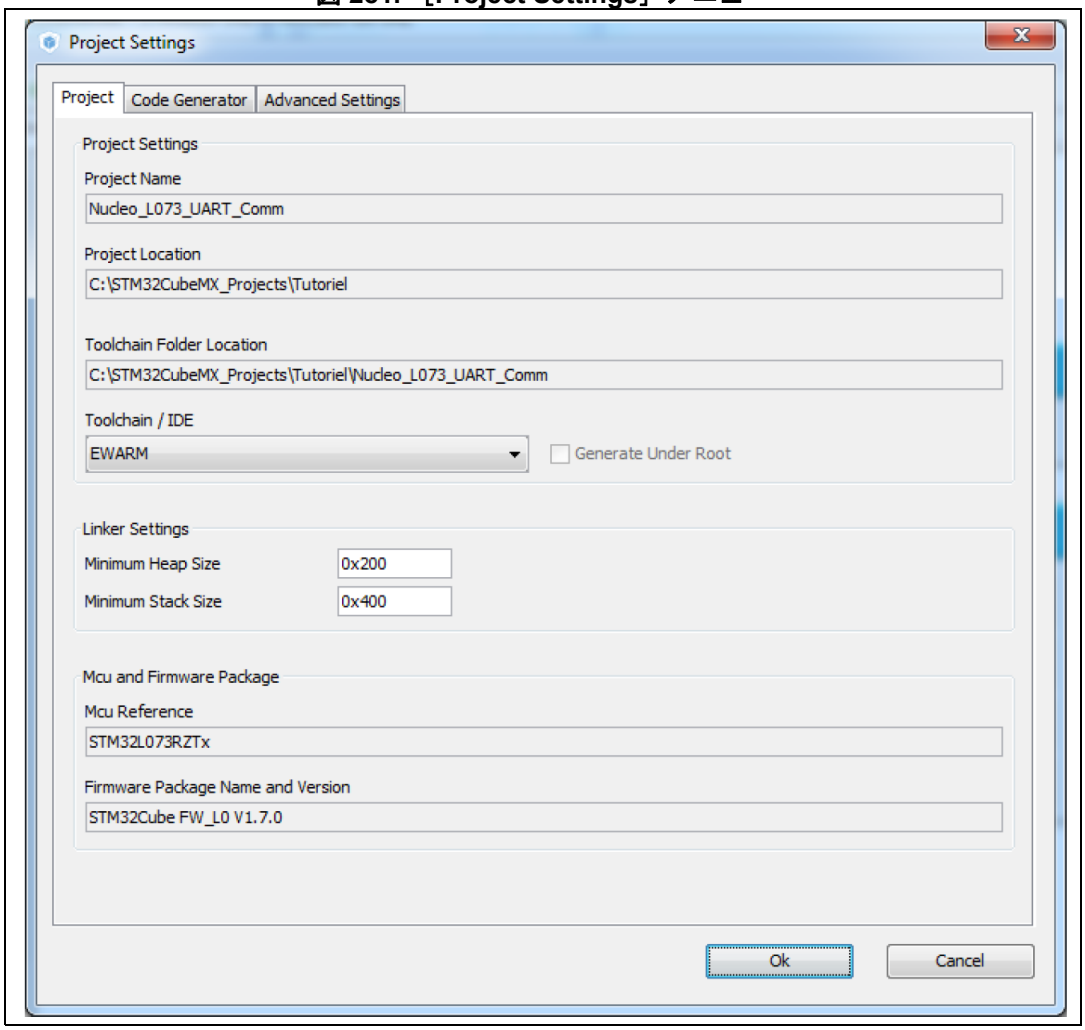
**図 280. TIM2 割込みの有効化**



## 14.6 プロジェクトの設定およびプロジェクトの生成

1. ［**Project Settings**］メニューで、プロジェクト名と転送先フォルダを指定し、EWARM IDE ツールチェーンを選択します（図 281 を参照）。

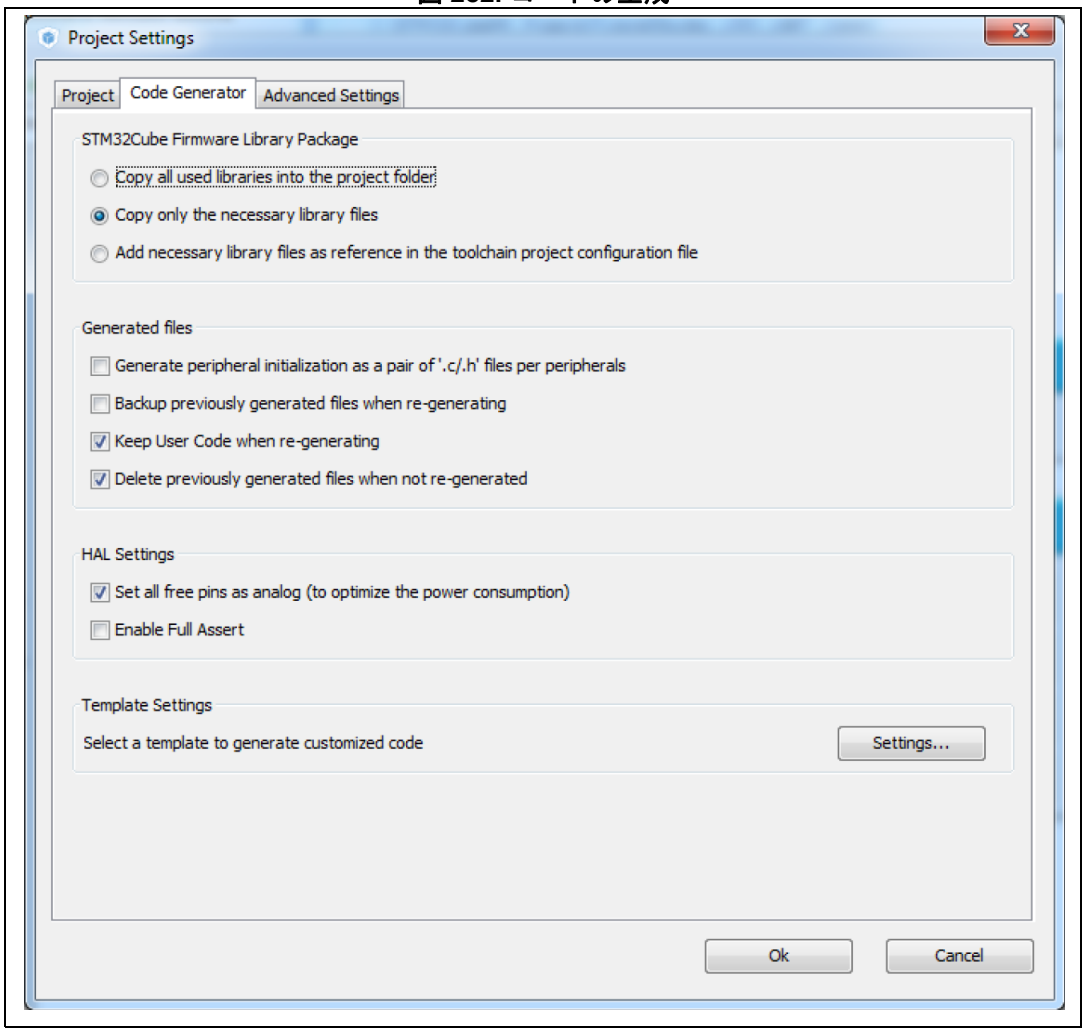
**図 281. ［Project Settings］メニュー**




ユーザの PC 上にファームウェアのパッケージ･バージョンが存在していない場合は、ファームウェア･パッケージのダウンロード進捗状況を示す進捗ウィンドウが表示されます。

2. ［**Code Generator**］タブで、図 282 に示すように、生成するコードを設定し、［**OK**］をクリックしてそのコードを生成します。

**図 282. コードの生成**




## 14.7 ユーザのアプリケーション･コードによるプロジェクトの更新

次のユーザ･コードを追加します。

```
/* USER CODE BEGIN 0 */
#include "stdio.h"
#include "string.h"
/* Buffer used for transmission and number of transmissions */
char aTxBuffer[1024];
int nbtime=1;
/* USER CODE END 0 */
```

main 関数で、次のようにタイマ･イベント生成関数を開始します。

```
/* USER CODE BEGIN 2 */
```

```
  /* Start Timer event generation */
    HAL_TIM_Base_Start_IT(&htim2);
  /* USER CODE END 2 */

/* USER CODE BEGIN 4 */
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim){
sprintf(aTxBuffer,"STM32CubeMX rocks %d times \t", ++nbtime);
HAL_UART_Transmit(&huart2,(uint8_t *) aTxBuffer, strlen(aTxBuffer), 5000);
}
/* USER CODE END 4 */
```

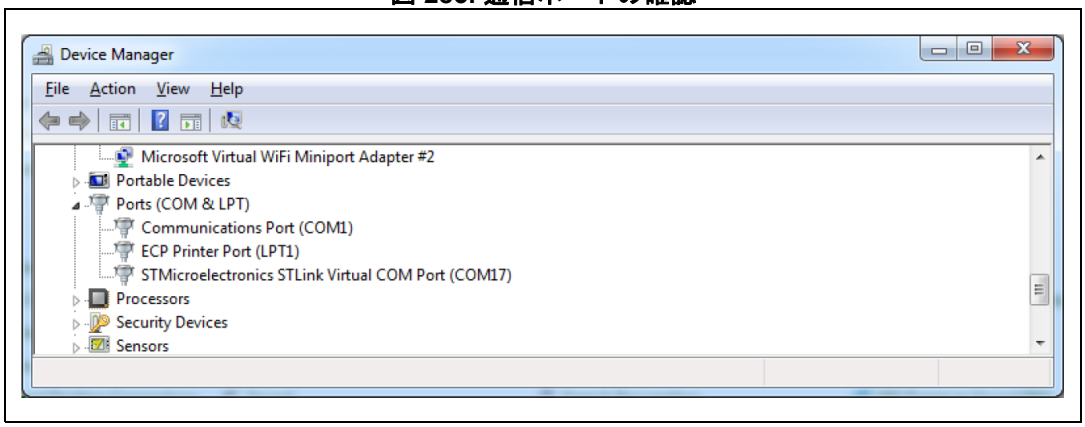
## 14.8 プロジェクトのコンパイルと実行

1. 都合の良い IDE でプロジェクトをコンパイルします。
2. それをボードにダウンロードします。
3. プログラムを実行します。

## 14.9 PC 上のシリアル通信クライアントとしての Tera Term ソフトウェアの設定

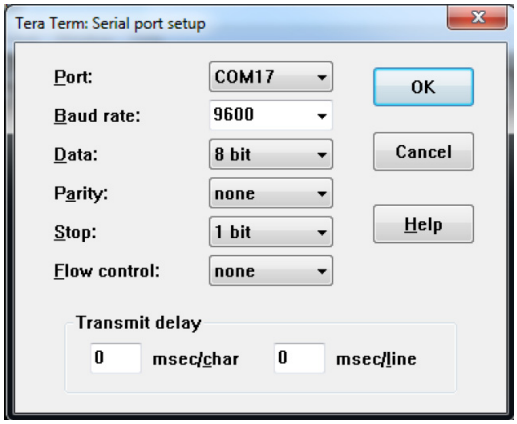
1. コンピュータで、STマイクロエレクトロニクスが使用する仮想通信ポートを［デバイス マネージャ］ウィンドウで確認します（図 283 を参照）。

**図 283. 通信ポートの確認**



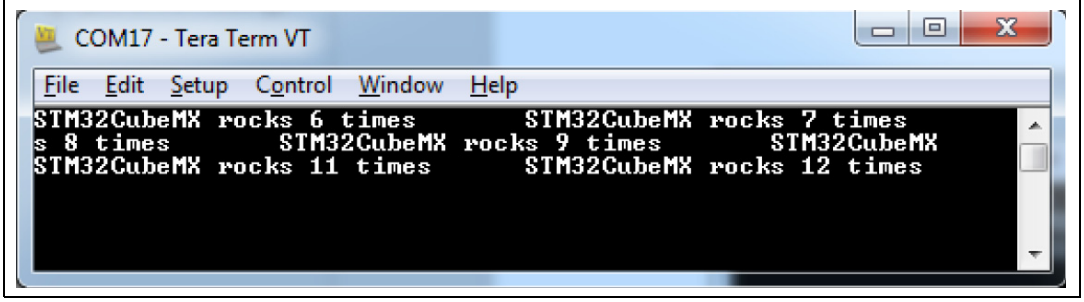
2. 関連の仮想通信ポートを接続するように Tera Term を設定するには、USART2 パラメータの設定に一致するようにマイクロコントローラで各種パラメータを調整します（図 284 を参照）。

**図 284. Tera Term のポート･パラメータの設定**




3. Tera Term のウィンドウには、数秒の間隔でボードからのメッセージが表示されます（図 285 を参照）。

**図 285. Tera Term のポート･パラメータの設定**



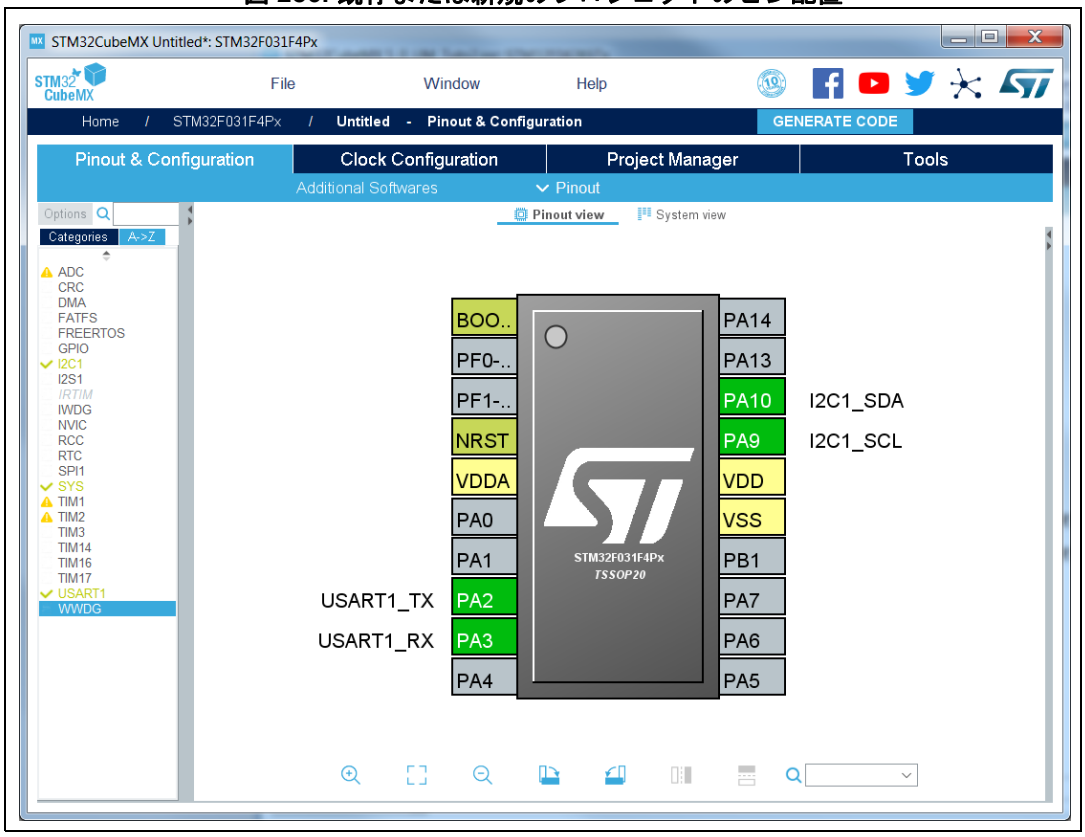
# 15 チュートリアル 5：互換性のあるマイクロコントローラへの現在のプロジェクト設定のエクスポート

**[Pinout]** メニューで **[List pinout compatible MCUs]** を選択すると、現在のプロジェクト設定と互換性のあるマイクロコントローラの一覧が STM32CubeMX によって取得され、新たに選択された互換マイクロコントローラに現在の設定をエクスポートすることが提案されます。

このチュートリアルでは、互換性のあるマイクロコントローラの一覧を表示して、現在のプロジェクト設定を互換マイクロコントローラにエクスポートする方法を紹介します。

1. 既存のプロジェクトを読み込むか、新たなプロジェクトを作成して保存します。

**図 286. 既存または新規のプロジェクトのピン配置**




2. **[Pinout]** メニューに移動して **[List pinout compatible MCUs]** を選択します。**[Pinout compatible]** ウィンドウがポップアップ表示されます（図 287 および図 288 を参照）。

   必要に応じ、検索条件とフィルタ·オプションを変更し、**[Search]** ボタンをクリックして検索プロセスを再実行します。

   色の濃さと［Comments］列によって、次のように検索に対する一致の程度が示されます。

   – 完全一致：このマイクロコントローラには、現在のプロジェクトに対する全面的な互換性があります（図 288 で例を参照）。

– ハードウェア互換の部分一致：ハードウェアの互換性は確保されている可能性がありますが、一部のピン名が保持されない可能性があります。目的のマイクロコントローラの上にマウス･カーソルを置くと、説明を記述したツールチップが表示されます（図 287 で例を参照）。

– ハードウェア非互換の部分一致：正確に同じピン位置に配置できないピンが存在するため、再マッピングが必要です。目的のマイクロコントローラの上にマウス･カーソルを置くと、説明を記述したツールチップが表示されます（図 288 で例を参照）。

**図 287. ピン配置互換マイクロコントローラの一覧表示- ハードウェア互換の部分一致**

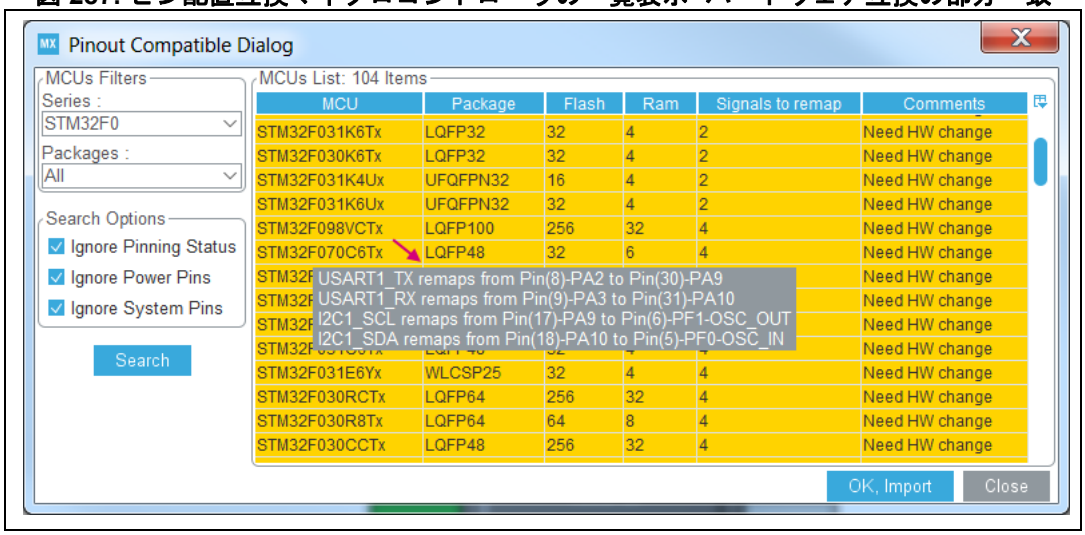
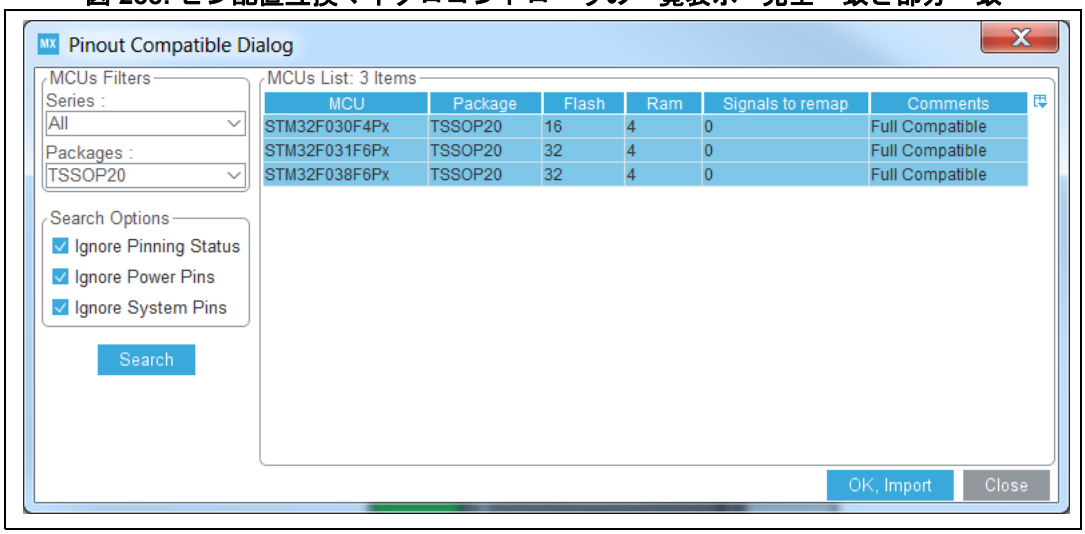
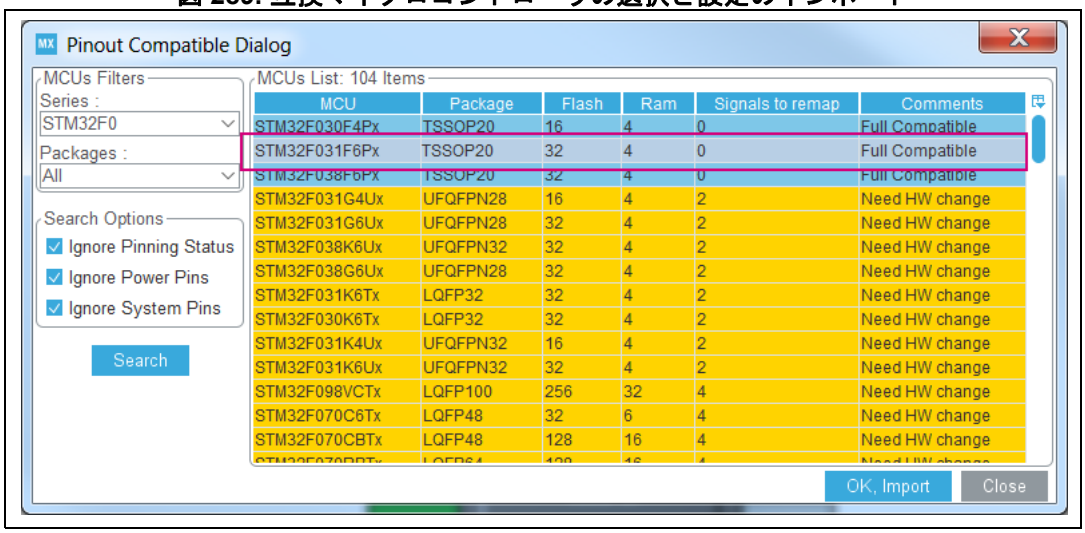



**図 288. ピン配置互換マイクロコントローラの一覧表示 - 完全一致と部分一致**



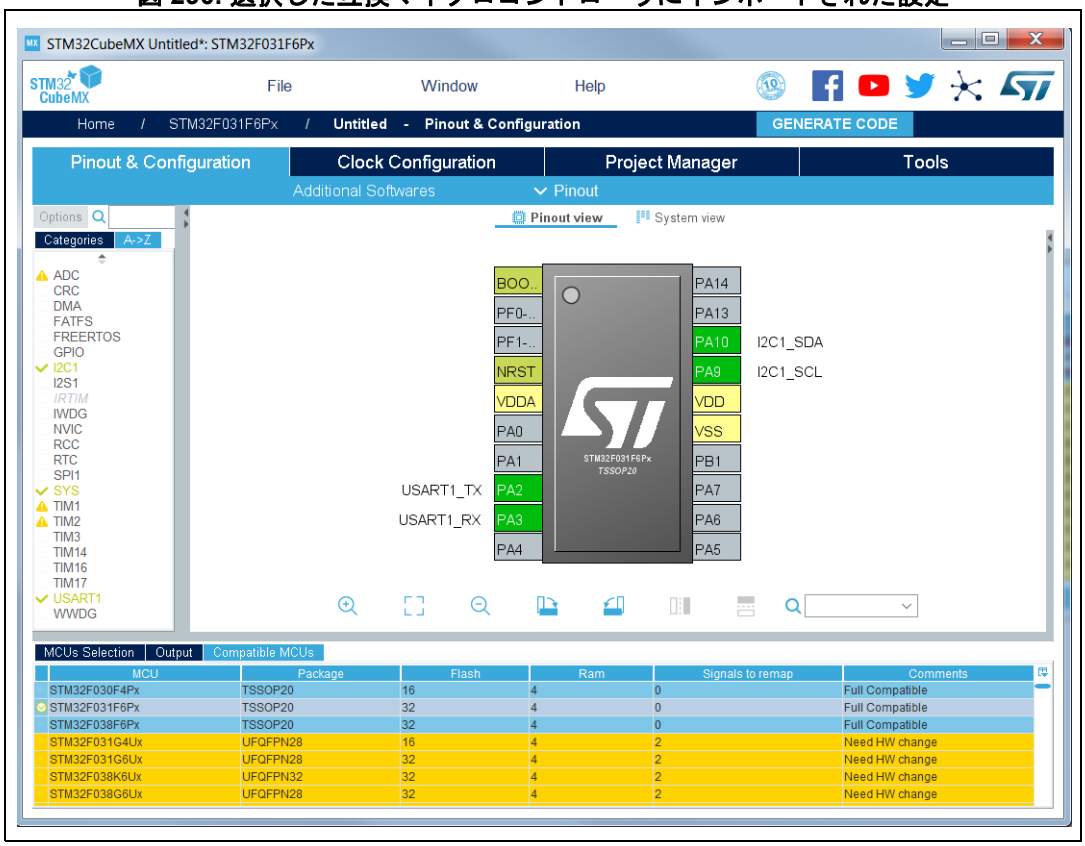
3. 現在の設定をインポートするマイクロコントローラを選択し、**[OK, Import]** をクリックします。

**図 289. 互換マイクロコントローラの選択と設定のインポート**




選択したマイクロコントローラで、その設定を使用できるようになります。

**図 290. 選択した互換マイクロコントローラにインポートされた設定**



4. 任意の時点で互換マイクロコントローラの一覧を表示するには、**[Window]** メニューで **[Outputs]** を選択します。

   現在の設定を別の互換マイクロコントローラに読み込むには、互換マイクロコントローラの一覧をダブルクリックします。

5. 検索条件に対する制約の一部を削除するには、次のような方法があります。
   - ピンの状況（ロックされたピン）を無視するには **[Ignore Pinning Status]** チェックボックスを選択します。
   - 電源ピンが考慮されないようにするには **[Ignore Power Pins]** を選択します。
   - システム･ピンが考慮されないようにするには **[Ignore System Pins]** を選択します。このチェックボックスの上にマウス･カーソルを置くと、現在のマイクロコントローラに存在するシステム･ピンを一覧記述したツールチップが表示されます。

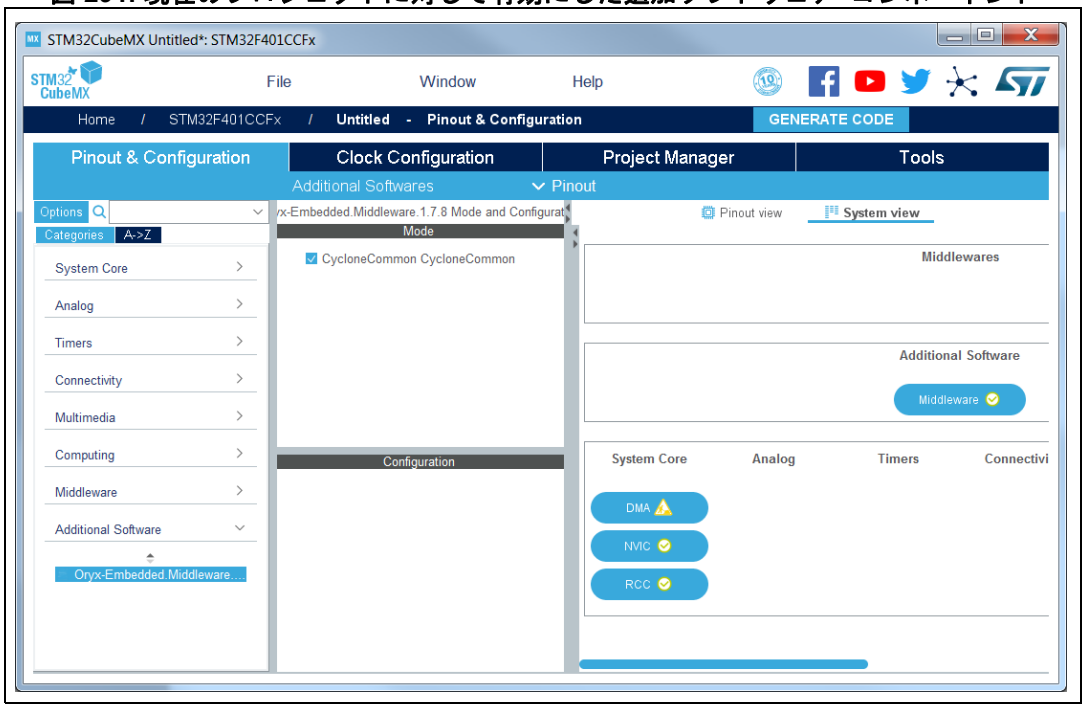
# 16 チュートリアル 6 ― ユーザのプロジェクトへの組込みソフトウェア･パッケージの追加

このチュートリアルでは、Oryx 組込みミドルウェア 1.7.8. パッケージを例として、STM32CubeMX プロジェクトにパッケージ･ソフトウェア･コンポーネントを追加する方法を紹介します。ここで取り上げるこのパッケージの使用方法を、STマイクロエレクトロニクス推奨の方法であると考えないでください。

プロジェクトに組込みソフトウェア･パッケージを追加するには、次の手順に従います。

1. http://www.oryx-embedded.com から入手できる .pdsc ファイルを使用して、Oryx 組込みミドルウェアの 1.7.8. パッケージをインストールします（セクション 3.4.5 : 組込みソフトウェア･パッケージのインストールを参照）。
2. **［New project］**を選択します。
3. **［MCU selector］**で［STM32F01CCFx］を選択します。
4. **［Pinout & Configuration］**ビューの**［Additional Software］**を選択して追加ソフトウェア･コンポーネントのウィンドウを開き、CycloneCommon バンドルの Compiler Support、RTOS Port/None、および Date Time Helper Routines の各ソフトウェア･コンポーネントを選択します（セクション 4.13 : ［Additional software component selection］ウィンドウを参照）。
5. **［OK］**をクリックして、選択したコンポーネントをツリー･ビューで表示します。チェックボックスをクリックし、現在のプロジェクトに対してそのソフトウェア･コンポーネントを有効にします（図 291 を参照）。

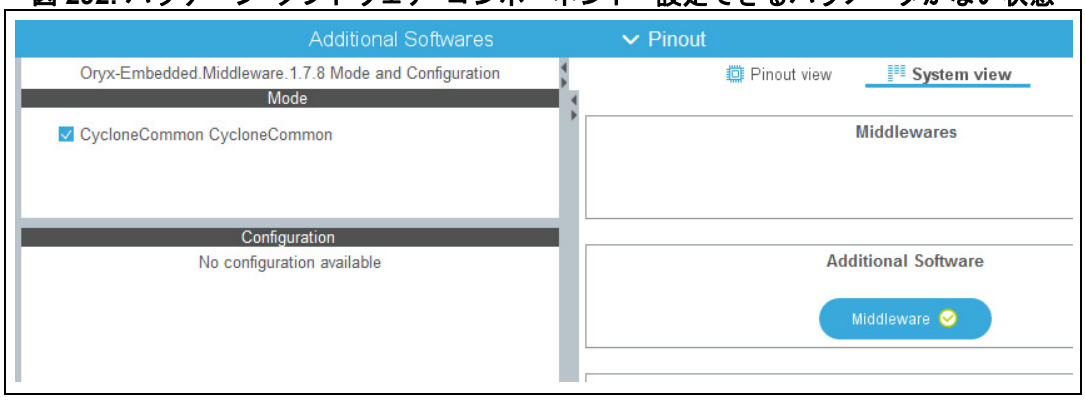
**図 291. 現在のプロジェクトに対して有効にした追加ソフトウェア･コンポーネント**




パッケージ名が緑色でハイライトされていれば、選択したソフトウェア･コンポーネントですべての条件が成立しています（真として解決されています）。解決されていない条件が1つでもあると、パッケージ名がオレンジ色で表示されます。

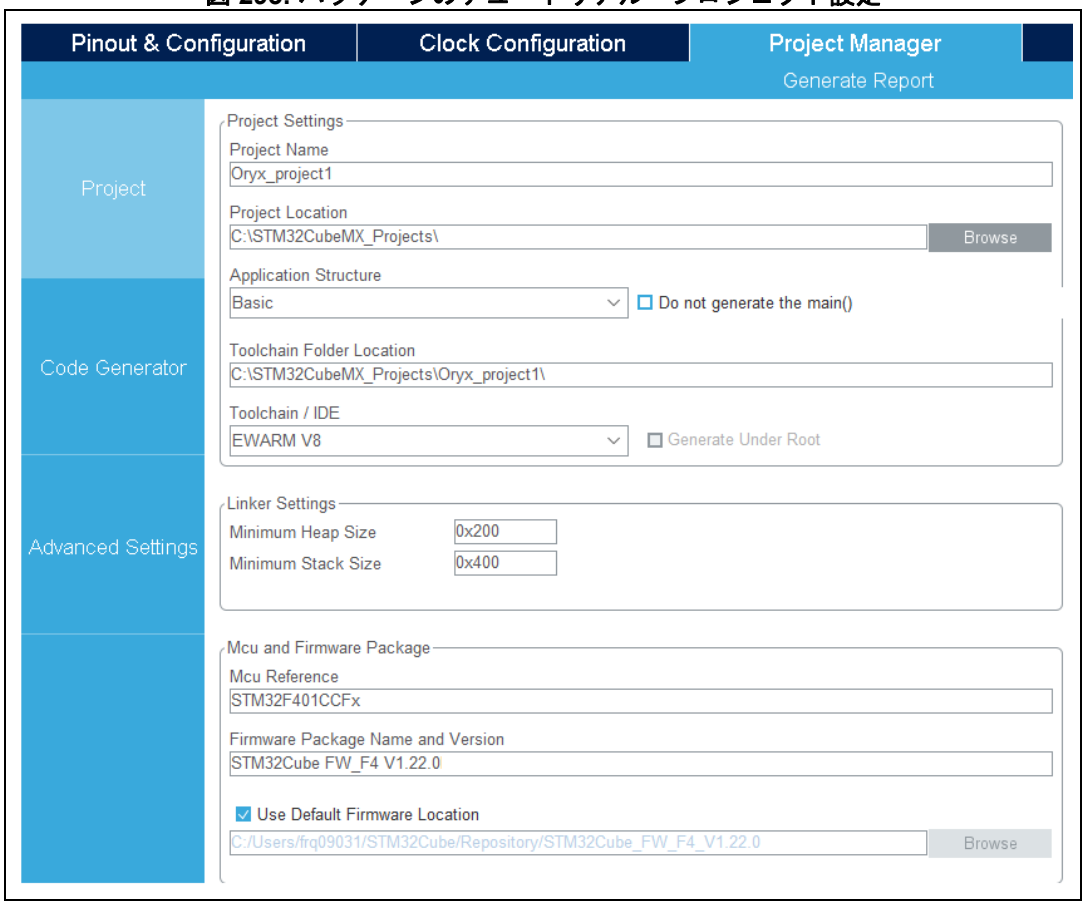
6. **［Configuration］**タブではどのパラメータも設定できないことを確認します（図 292 を参照）。

**図 292. パッケージ･ソフトウェア･コンポーネント - 設定できるパラメータがない状態**



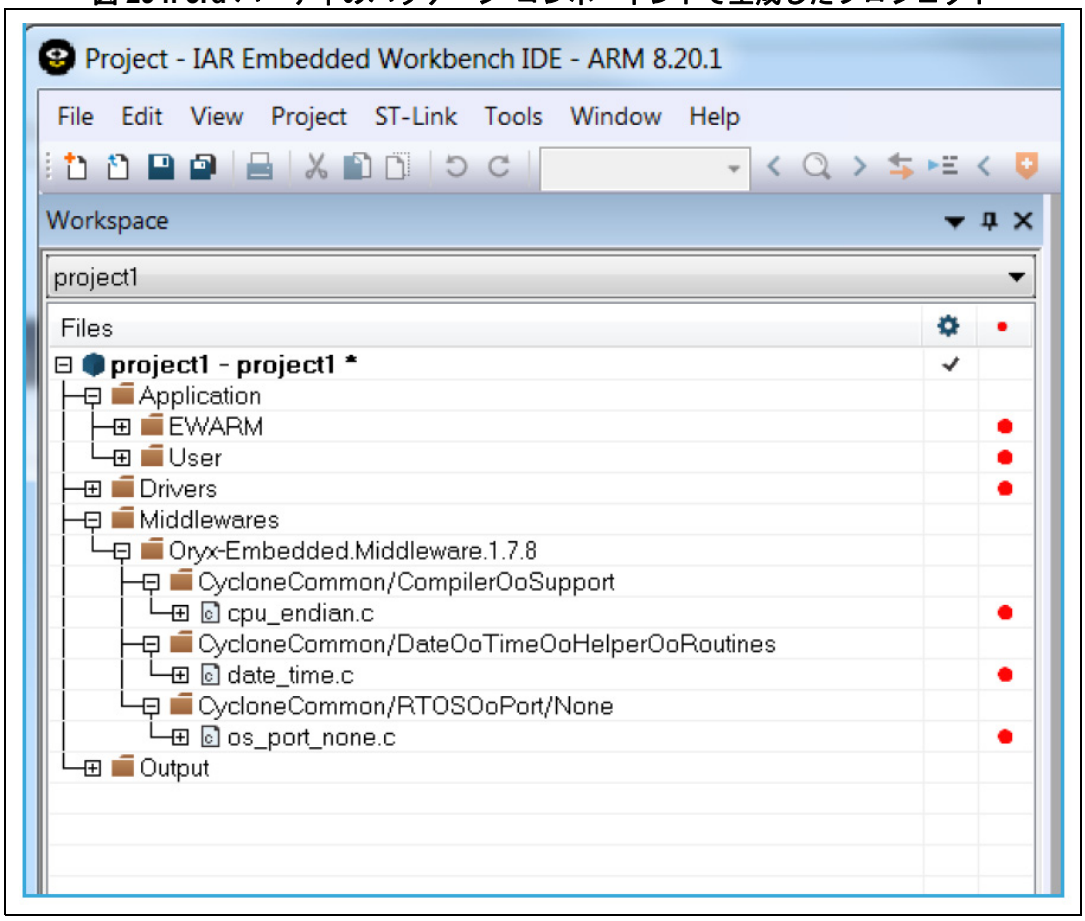

7. **［Project Manager］**の［Project］タブで、プロジェクトのパラメータを指定し（図 293 参照）、IDE として IAR™ EWARM を選択します。

**図 293. パッケージのチュートリアル - プロジェクト設定**



8. GENERATE CODE をクリックしてプロジェクトを生成します。STM32CubeF4 マイクロコントローラ･パッケージが STM32Cube リポジトリにない場合は、そのパッケージをダウンロードすることに同意します。
9. **[Open project]** をクリックします。生成したプロジェクトに Oryx ソフトウェア･コンポーネントが表示されます（図 294 参照）。

**図 294. 3rd パーティのパッケージ･コンポーネントで生成したプロジェクト**



# 17 チュートリアル 7 ー X-Cube-BLE1 ソフトウェア･パッケージの使用

このチュートリアルでは、X-Cube-BLE1 ソフトウェア･パッケージを使用して機能的なプロジェクトを実現する方法を紹介します。

このチュートリアルを実行するための前提条件を以下に挙げます。

- ハードウェア：NUCLEO-L053R8、X-NUCLEO-IDB05A1、およびミニ USB ケーブル（図 295 を参照）
- ツール：STM32CubeMX、IDE（Atollic®など、STM32CubeMX でサポートされている任意のツールチェーン）
- 組込みソフトウェア･パッケージ：STM32CubeL0（バージョン 1.10.0 以降）、X-Cube-BLE1 1.1.0（図 296 参照）
- モバイル･アプリケーション（図 297 参照）：ST マイクロエレクトロニクス BlueNRG アプリケーション（iOS® 用または Android™ 用）

**図 295. ハードウェアの前提条件**

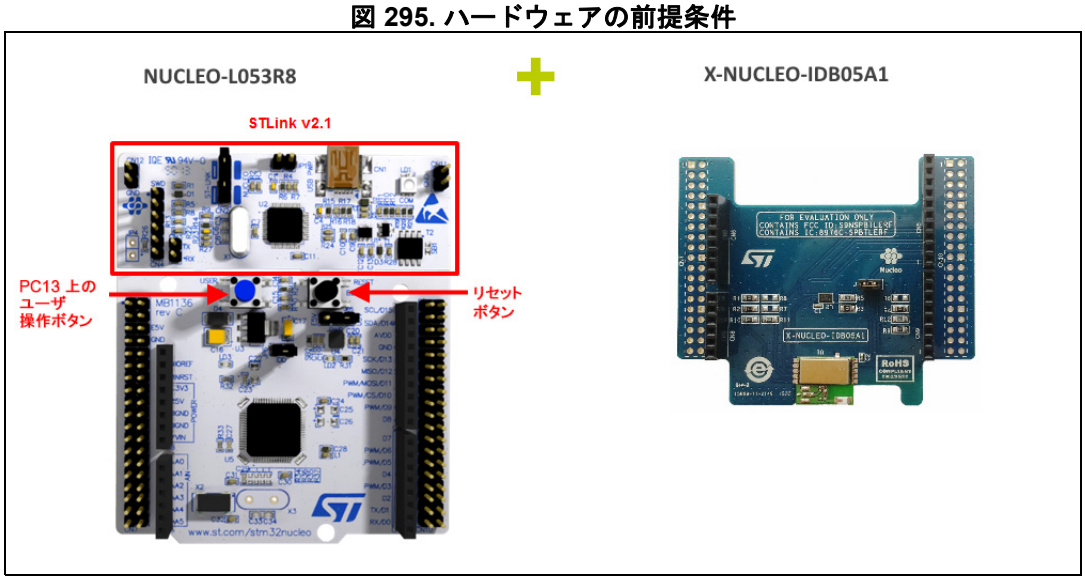


図 296. 組込みソフトウェア･パッケージ

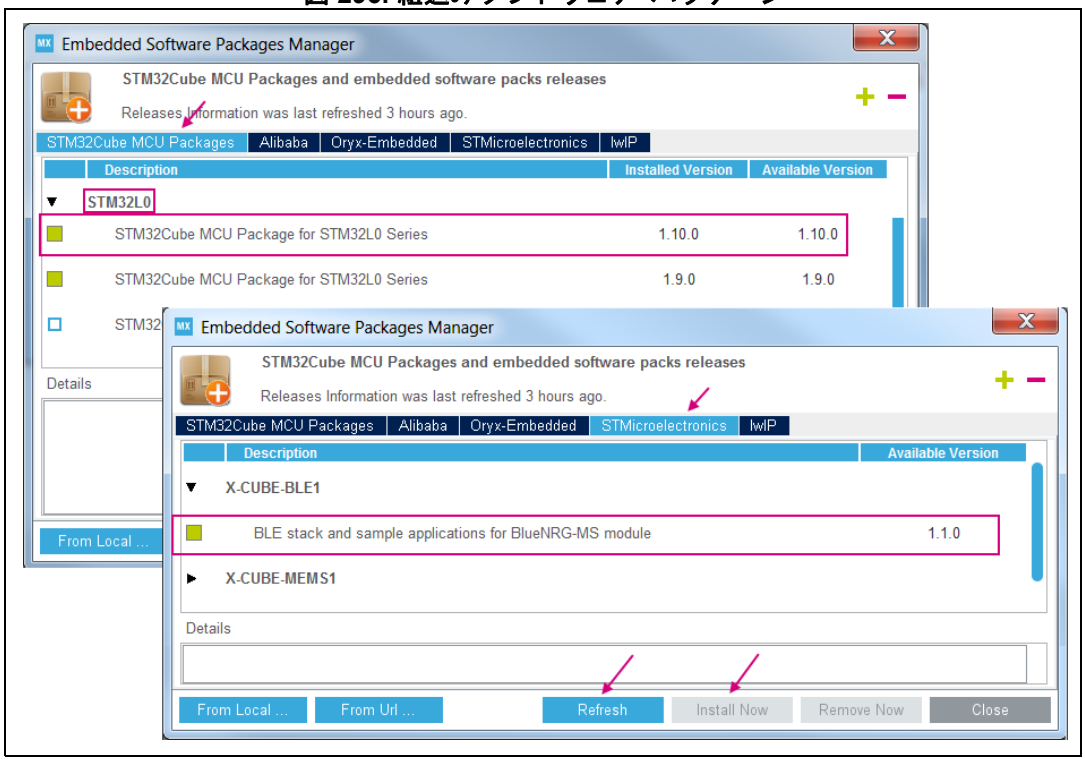
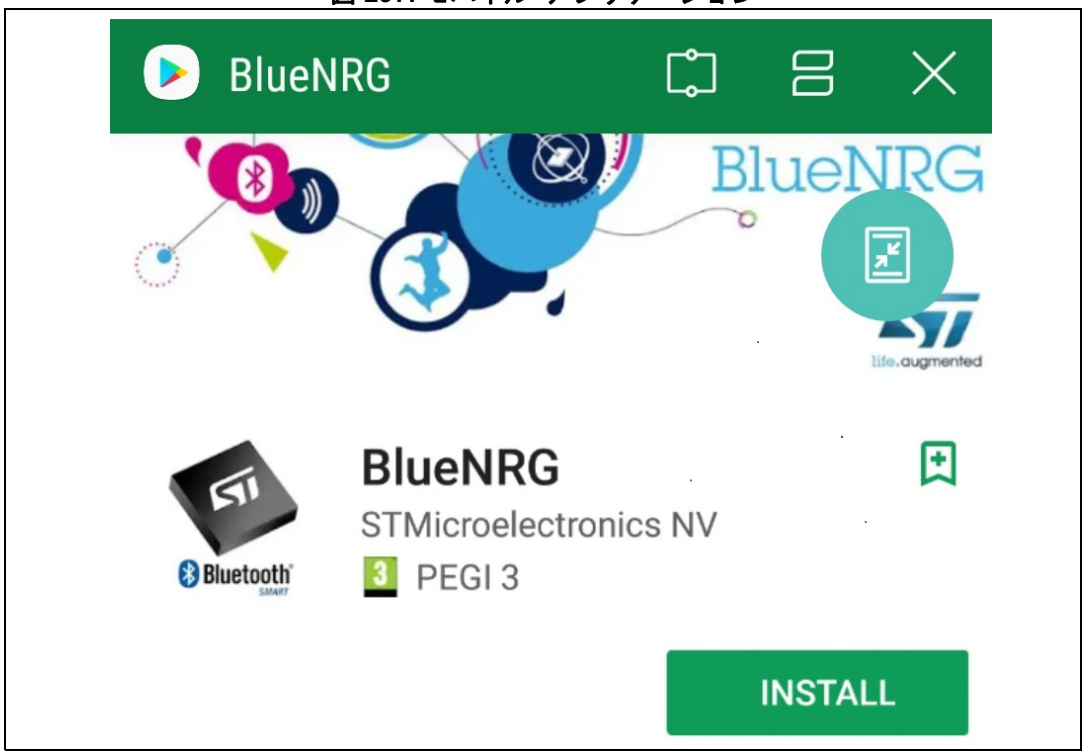


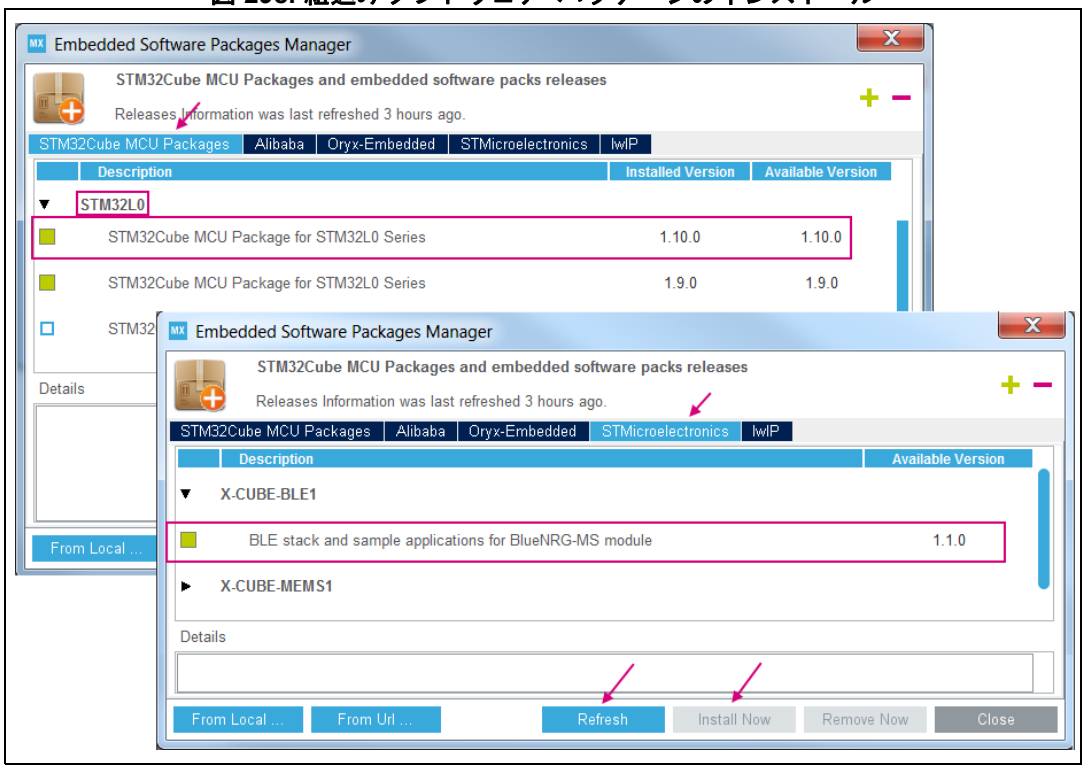

図 297. モバイル･アプリケーション



チュートリアルをインストールして実行するには次の手順に進みます。

1. STM32CubeMX のインターネット接続を確認します。
   a) ［**Help**］→［**Updater Settings**］を選択してアップデータのウィンドウを開きます。
   b) インターネット接続が設定済みで稼動していることを［**Connection**］タブで確認します。
2. 必要な組込みソフトウェア･パッケージをインストールします（図 298 を参照）。
   a) ［**Help**］→［**Manage Embedded software packages**］を選択して、［**Embedded Software Packages Manager**］ウィンドウを開きます。
   b) ［**Refresh**］ボタンをクリックして、利用できる最新のパッケージ･バージョンで一覧を更新します。
   c) ［**STM32Cube MCU Package**］タブを選択して、STM32CubeL0 のファームウェア･パッケージとしてバージョン 1.10.0 以降がインストールされていることを確認します（チェックボックスが緑色で表示されている必要があります）。そのようになっていない場合は、チェックボックスを選択して［**Install now**］をクリックします。
   d) ［**STMicroelectronics**］タブを選択して、X-Cube-BLE1 ソフトウェア･パッケージのバージョン 1.0.0 がインストールされていることを確認します（チェックボックスが緑色で表示されている必要があります）。そのようになっていない場合は、チェックボックスを選択して［**Install now**］をクリックします。

**図 298. 組込みソフトウェア･パッケージのインストール**




3. 新しいプロジェクトを開始します。
   a) ［**New Project**］を選択して新規プロジェクトのウィンドウを開きます。
   b) ［**Board selector**］タブを選択します。
   c) ボード･タイプとして Nucleo64、マイクロコントローラのシリーズとして STM32L0 をそれぞれ選択します。

d) 表示されたボードの一覧から NUCLEO-L053R8 を選択します（図 299 を参照）。

e) すべてのペリフェラルをそれぞれのデフォルト･モードに初期化するかどうかを確認するダイアログ･ボックスが表示されるので［**No**］を選択します（図 300 を参照）。

**図 299. 新規プロジェクトの開始 - NUCLEO-L053R8 ボードの選択**

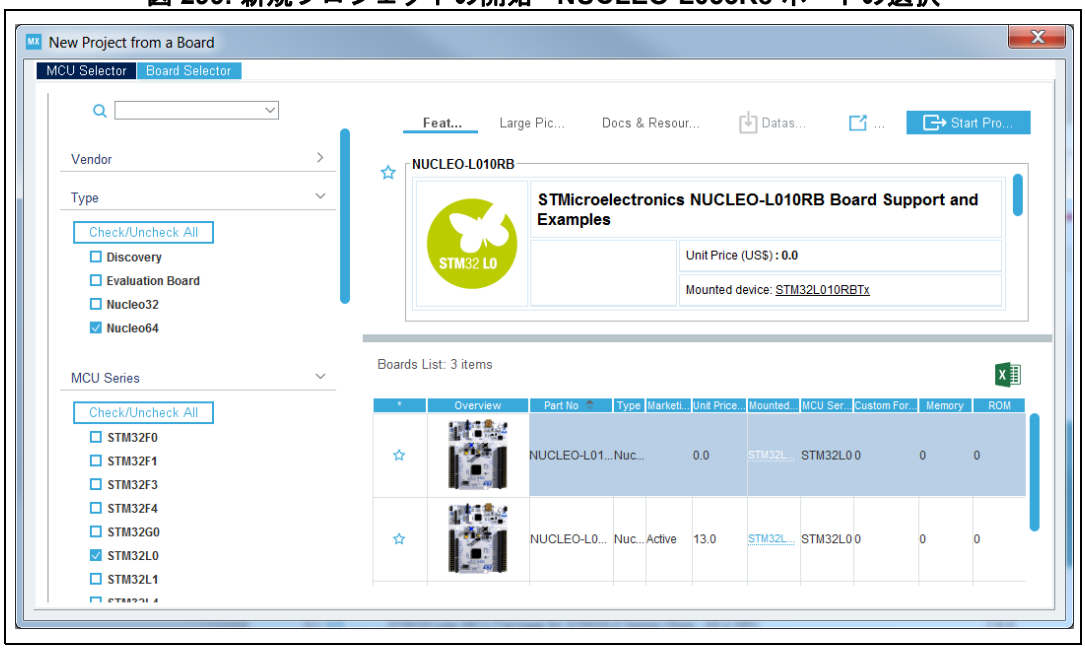
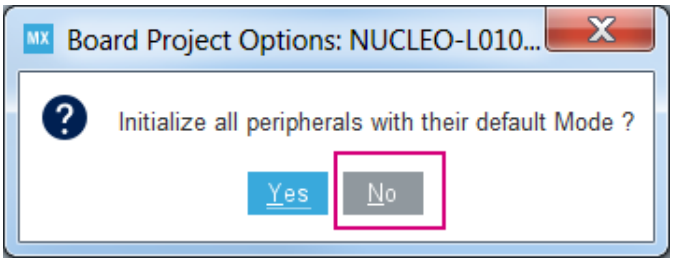



**図 300. 新規プロジェクトの開始 - すべてのペリフェラルの初期化**




4. 次の手順でプロジェクトに X-Cube-BLE1 コンポーネントを追加します。

a) ［**Pinout & Configuration**］ビューの［**Additional Software**］をクリックして、［**Additional Software component Selection**］ウィンドウを開きます。

b) 関連するコンポーネントを選択します（図 301 参照）。

［Application］グループにはアプリケーションの一覧が表示されます。これらの C ファイルによって、Process() 関数であるアプリケーション･ループが実装されます。［Application］グループから［**SensorDemo**］を選択します。

**Controller** コンポーネントと **Utils** コンポーネントを選択します。

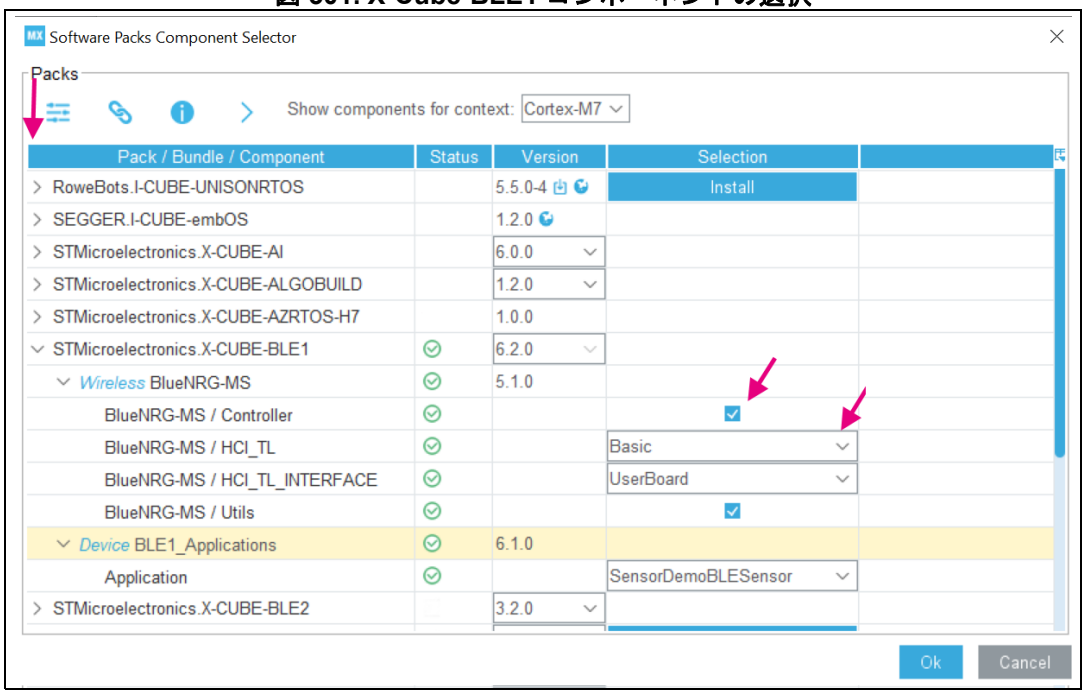
**HCI_TL** コンポーネントに対するバリアントとして［**Basic**］を選択します。Basic バリアントを選択すると、HCI_TL API の ST マイクロエレクトロニクス実装が提供されますが、テンプレートのオプションを選択した場合は、ユーザ側で独自のコードを実装する必要があります。

**HCI_TL_INTERFACE** コンポーネントに対するバリアントとして［**UserBoard**］を選択します。［UserBoard］のオプションを選択すると <ボード名>_bus.c ファイルが生成されます。このチュートリアルでは、このファイルは nucleo_l053r8_bus.c です。テンプレートのオプションを選択した場合は custom_bus.c ファイルが生成され、ユーザ側で独自の実装を用意する必要があります。

ソフトウェア･コンポーネントの詳細については、X-Cube-BLE1 パッケージのドキュメントを参照してください。

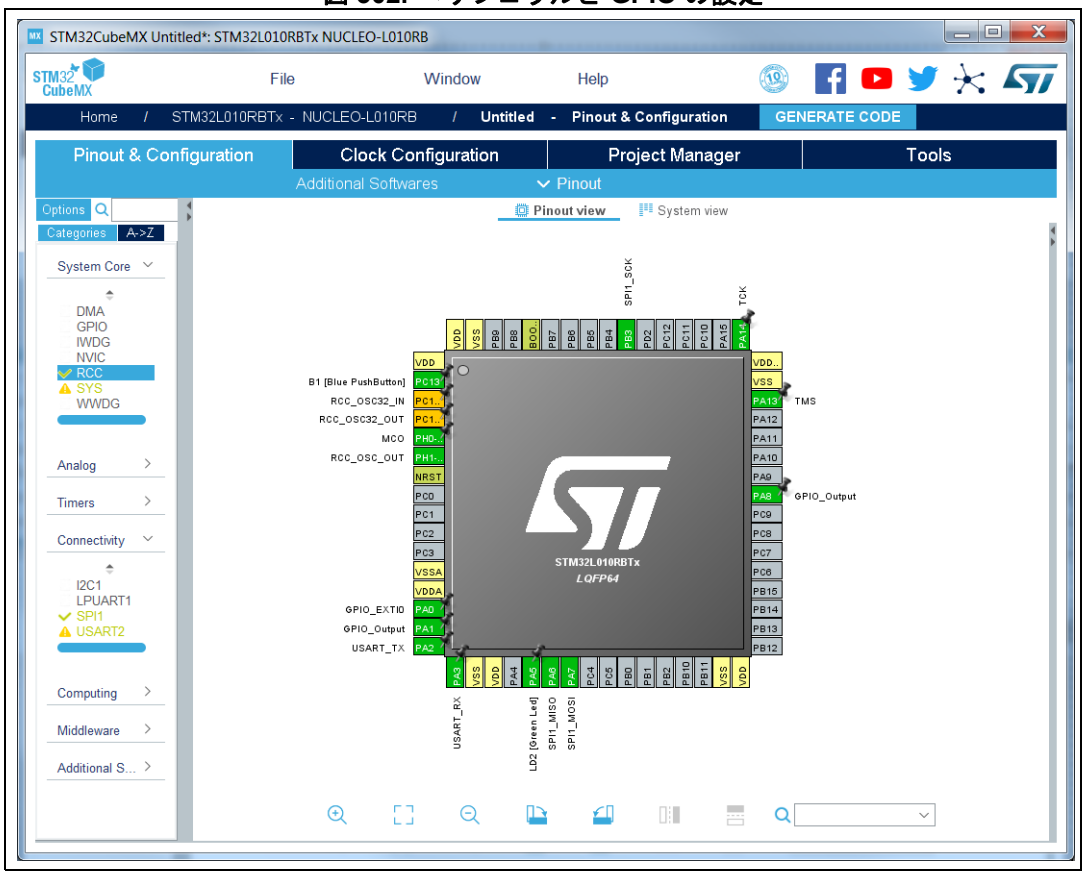
c) ［**OK**］をクリックしてプロジェクトに選択内容を適用し、ウィンドウを閉じます。この適用に応じて、左側パネルの［**Additional Software**］セクションが更新されます。

**図 301. X-Cube-BLE1 コンポーネントの選択**




5. 次の手順により、［**Pinout**］タブでペリフェラルと GPIO を有効にします（図 302 を参照）。
   a) ［**USART2**］のモードを［**Asynchronous**］に設定します。
   b) ［**SPI1**］のモードを［**Full-duplex master**］に設定します。
   c) 次の各ピンを左クリックして、必要な GPIO 設定を指定します。
      **PA0**：GPIO_EXTI0
      **PA1**：GPIO_Output
      **PA8**：GPIO_Output
   d) ［**SYS**］ペリフェラルで［**Debug Serial Wire**］を有効にします。

**図 302. ペリフェラルと GPIO の設定**




6. 次の手順により、**[Configuration]** タブで各ペリフェラルを設定します。

   a) **[System]** セクションの **[NVIC]** ボタンをクリックして **[NVIC configuration]** ウィンドウを開きます。[EXTI line 0 and line 1 interrupts] を有効にして **[OK]** をクリックします（図 303 を参照）。

   b) **[Connectivity]** セクションの **[SPI]** ボタンをクリックして **[SPI configuration]** ウィンドウを開きます。データ･サイズが 8 bit、プリスケーラ値が 16 に設定されていることを確認します。これにより、このプリスケーラ値で分周された HCLK が 8 MHz 以下になります。

   c) **[Connectivity]** セクションで **[USART2]** をクリックして **[Configuration]** ウィンドウを開き、次の各パラメータ設定を確認します。

   **[Parameter Settings]** での設定：

   ボーレート：115200 bit/s

   ワード長：8 bit（パリティを含む）

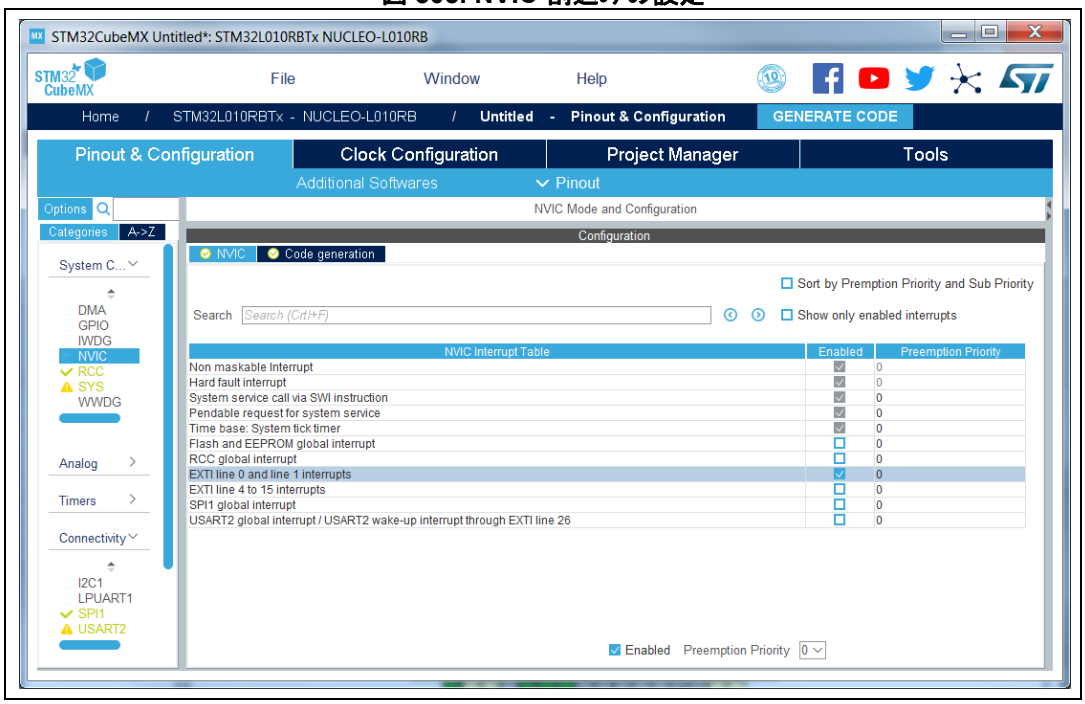
   パリティ：なし

   ストップ･ビット：1

   **[GPIO Settings]** での設定：

   ユーザ･ラベル：USART_TX および USART_RX

**図 303. NVIC 割込みの設定**




7. **［Pinout & Configuration］**ビューで X-Cube-BLE1 パッケージ･コンポーネントを有効化し、設定します。

   a) 左側パネルのパッケージ項目をクリックすると［Mode］タブと［Configuration］タブが表示されます。

   b) ［Mode］パネルのチェックボックスをクリックして X-Cube-BLE1 を有効化すると、設定するパラメータが［Configuration］パネルに表示されます。橙色の三角形は、未設定のパラメータがあることを表しています。すべてのパラメータを適切に設定すると、緑色のチェック･マークに変化します（図 304 参照）。

   c) ［Parameter Settings］タブは変更しません。

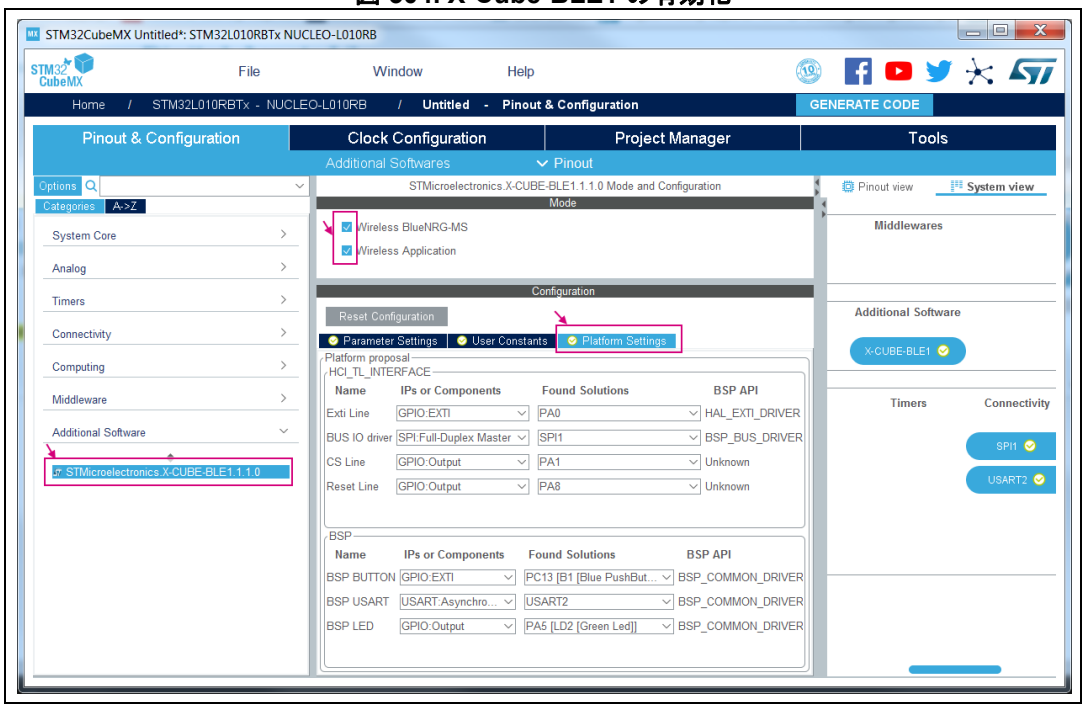
   d) ［Platform Settings］タブに移動して、図 304 と 表 23 に示したようにハードウェア･リソースとの接続を設定します。

**表 23. ハードウェア･リソースとの接続**

| 名前 | IPまたはコンポーネント | 検出されたソリューション |
|---|---|---|
| BUS IO ドライバ | 全二重マスタ･モードの SPI | SPI1 |
| EXTI ライン | GPIO:EXTI | PA0 |
| CS ライン | GPIO 出力 | PA1 |
| リセット･ライン | GPIO 出力 | PA8 |
| BSP LED | GPIO 出力 | PA5 |
| BSP ボタン | GPIO:EXTI | PC13 |
| BSP USART | 非同期モードの USART | USART2 |

アイコンが  に変わることを確認します。**[OK]** をクリックして **[Configuration]** ウィンドウを閉じます。

図 304. X-Cube-BLE1 の有効化




8. SensorDemo プロジェクトを生成します。

   a) GENERATE CODE をクリックしてコードを生成します。プロジェクトを保存していない場合は **[Project Settings]** ウィンドウが開きます。

   b) プロジェクト設定が適切に完了したら、GENERATE CODE をクリックしてコードを生成します（図 305 参照）。この生成が完了すると、[Open Folder] ボタンでプロジェクトのフォルダを開くか、[Open Project] ボタンでプロジェクトを IDE ツールチェーンで開くか選択することを求めるダイアログ･ウィンドウが表示されます。**[Open Project]** を選択します（図 306 を参照）。

   c) .cproject ファイルを Atollic® TrueStudio® に関連付けると、**[Open Project]** をクリックしたときに TrueStudio® が自動的に起動します。TrueStudio の起動ウィンドウで、ワークスペースを作成するか既存のワークスペースを選択して（図 307 参照）、**[OK]** をクリックします。STM32CubeMX で生成したプロジェクトが、TrueStudio® の **[Project Explorer]** パネルに表示されます（図 308 参照）。

図 305. SensorDemo プロジェクトの設定

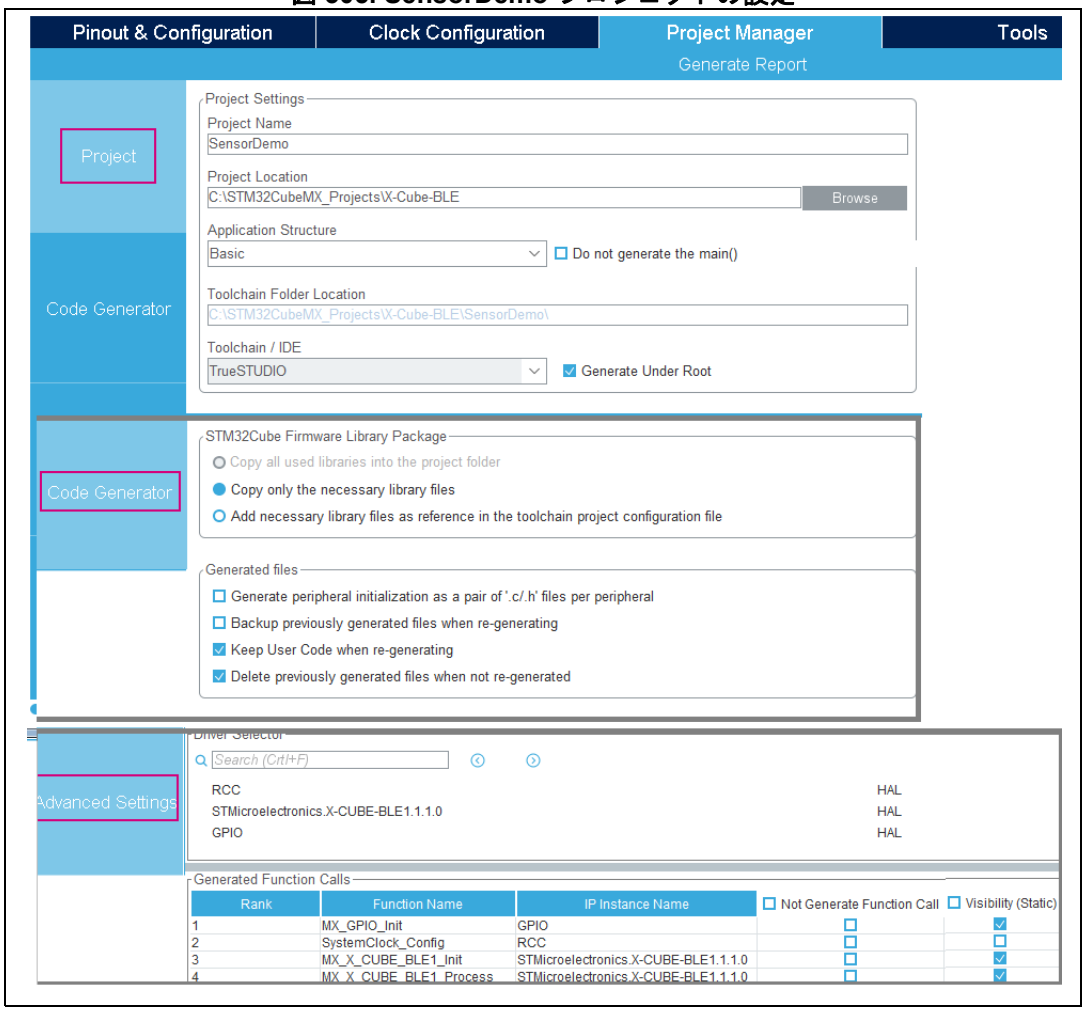



図 306. IDE ツールチェーンで SensorDemo プロジェクトを開く

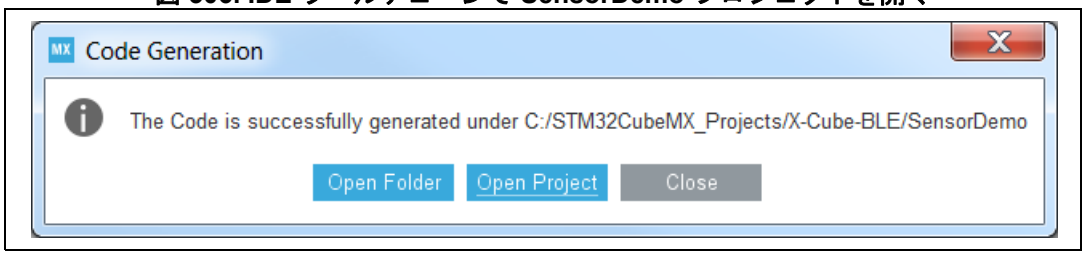


図 307. Atollic® TrueStudio® での SensorDemo プロジェクトの起動

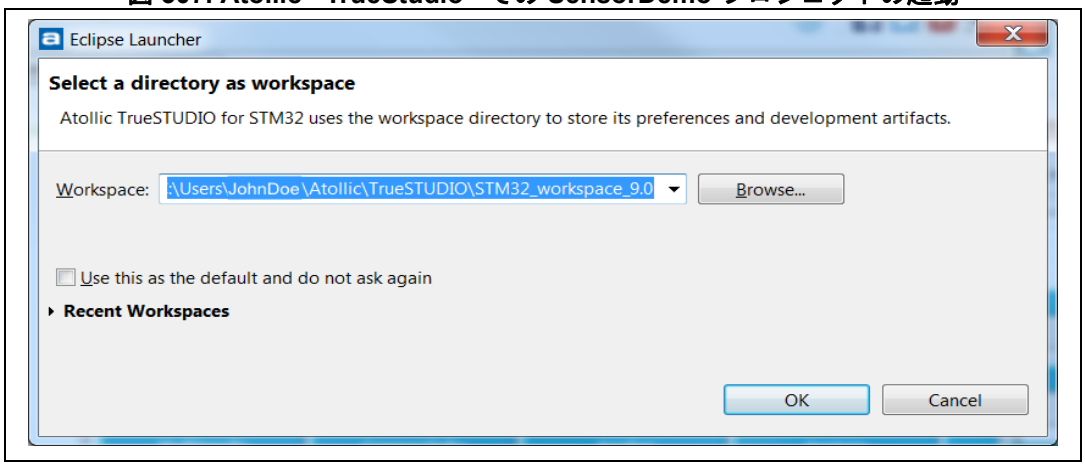
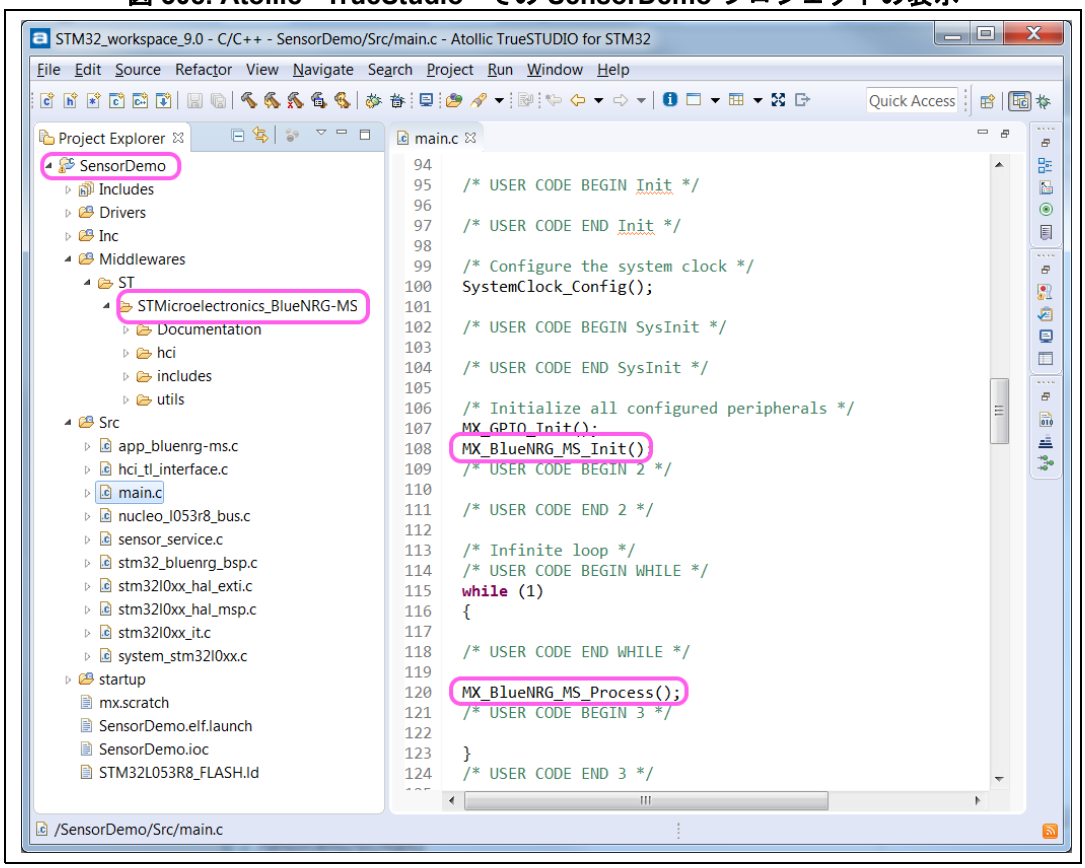



図 308. Atollic® TrueStudio® での SensorDemo プロジェクトの表示



9. 次の手順により、Atollic® TrueStudio® で SensorDemo アプリケーションをビルドして起動します。

   a) プロジェクトのプロパティを設定します（図 309 を参照）。

   **[Project Explorer]**パネルでプロジェクト名（SensorDemo）を右クリックし、**[Properties]**を選択して**［Properties］**ウィンドウを開きます。

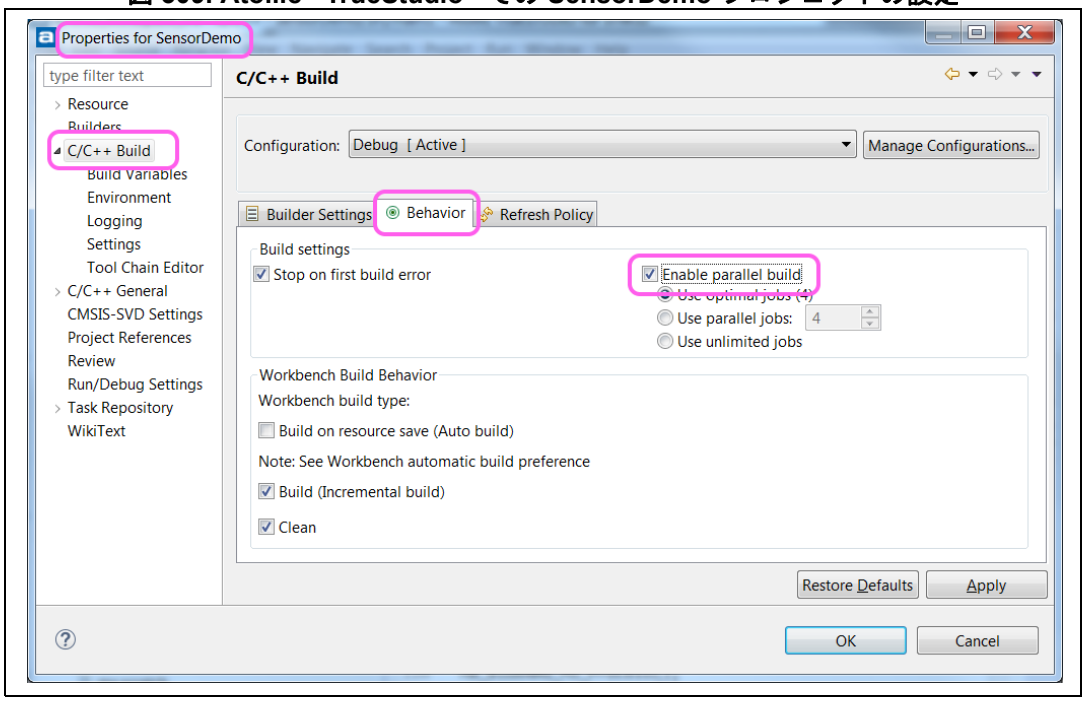
   **[C/C++ Build]**を選択し、**[Behavior]**タブで並行ビルドを有効にしてビルド･プロセスの高速化を図ります。

   b) ビルドのアイコン  をクリックしてプロジェクトをビルドします。

   c) Nucleo ボードの ST リンク･コネクタに USB ケーブルでコンピュータを接続します。

   d) **［Run］**メニューで  をクリックし、ボード上でプロジェクトを実行します。

**図 309. Atollic® TrueStudio® での SensorDemo プロジェクトの設定**



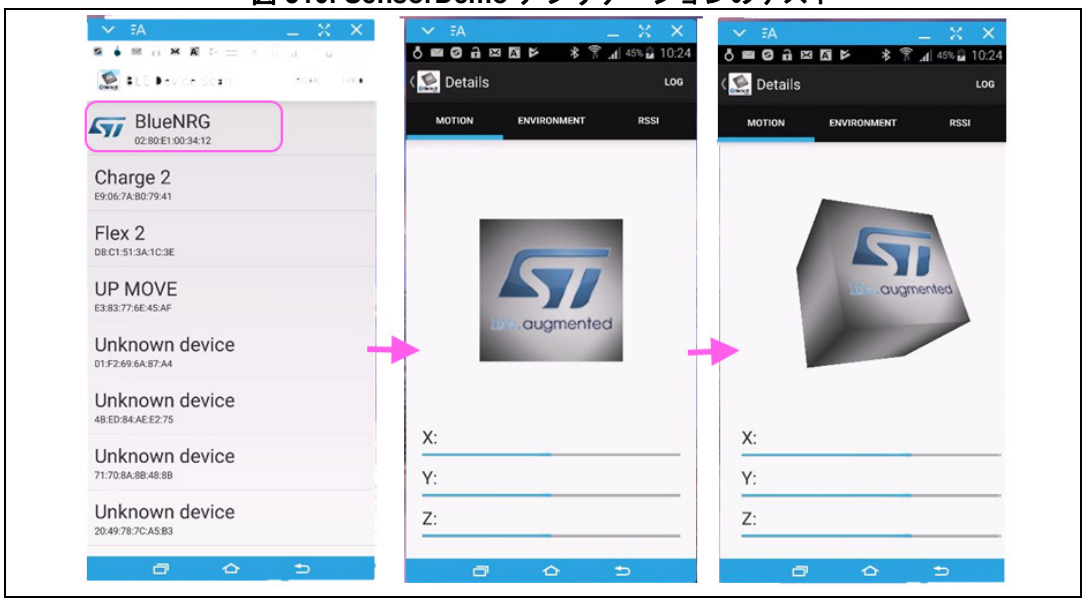

10. 次の手順により、スマートフォン･アプリケーション上で BlueNRG を起動することで、STM32 SensorDemo アプリケーションをテストします。

   e) 近くにあるデバイスをスキャンします。

   f) BlueNRG デバイスを選択します。

   g) このハードウェアには MEMS 検出素子がないので、**青色のボタン**を押して MEMS データをシミュレートします。このボタンを押すたびに ST キューブが一定の値で回転します（図 310 参照）。

図 310. SensorDemo アプリケーションのテスト



# 18 LPBAM プロジェクトの作成

## 18.1 LPBAM の概要

注意 : LPBAM モードの概要と使用方法について学ぶには、www.st.com より入手可能な LPBAM のアプリケーション·ノートと、SMT32Cube ファームウェア·パッケージの Utilities フォルダに保存されている LPBAM ユーティリティの入門ガイドを読むことをお勧めします。

### 18.1.1 LPBAM 動作モード

LPBAM は Low Power Background Autonomous Mode（低消費電力バックグラウンド自律モード）の略です。これは、電力モードに関係なく、ソフトウェアを一切実行せずに、ペリフェラルが自律的に機能する動作モードです。STM32 製品に組み込まれたハードウェア·サブシステムによってこの動作モードが実現しています。Linked-List モードでの DMA 転送を使用してさまざまな動作を連結することで、有用な機能（ペリフェラルの設定や転送）を構築できます。必要に応じて、非同期のイベントや割込みを生成することも可能です。LPBAM 動作モードでは、CPU の介入を一切必要としません。その結果、LPBAM サブシステムのメカニズムを使用することで、消費電力の最適化と CPU 負荷の軽減という 2 つの大きな利点が得られます。

### 18.1.2 LPBAM ファームウェア

LPBAM ファームウェアは、LPBAM アプリケーションを容易に作成できるように設計されています。LPBAM ユーティリティはモジュール方式のドライバ群であり、STM32Cube ファームウェア·パッケージの Utilities フォルダに置かれています。各モジュールは、アプリケーションのシナリオの構築に必要な API を実装する、C ファイルのペアとして提供されています。各モジュールで、特定のペリフェラルに対する設定可能性とデータ転送を管理します。LPBAM ユーティリティの設計では、LPBAM サブシステムのメカニズムをサポートするあらゆる STM32 デバイスとの互換性を、設定モジュールを通じて実現しています。そのために、アプリケーションのニーズに合わせた設定ファイル stm32_lpbam_conf.h が必要です。このユーティリティには、単一のアプリケーション·エントリ·ポイントである stm32_lpbam.h があるので、このファイルをプロジェクトにインクルードする必要があります。

### 18.1.3 サポート対象のシリーズ

LPBAM ファームウェアは、TrustZone をアクティブ化しているかどうかにかかわらず、STM32U575/585、STM32U595/5A5、STM32U599/5A9 の各製品に対応しています。

STM32CubeMX 6.5.0 からは、STM32U575/585 製品ライン上で TrustZone をアクティブ化していないプロジェクトに LPBAM を導入しています。これにより、STM32CubeMX の［LPBAM Scenario & Configuration］ビューでプロジェクトの LPBAM アプリケーションを作成し、対応するコードを生成できます。生成した C プロジェクトによって LPBAM ファームウェアが組み込まれます。

### 18.1.4 LPBAM の設計

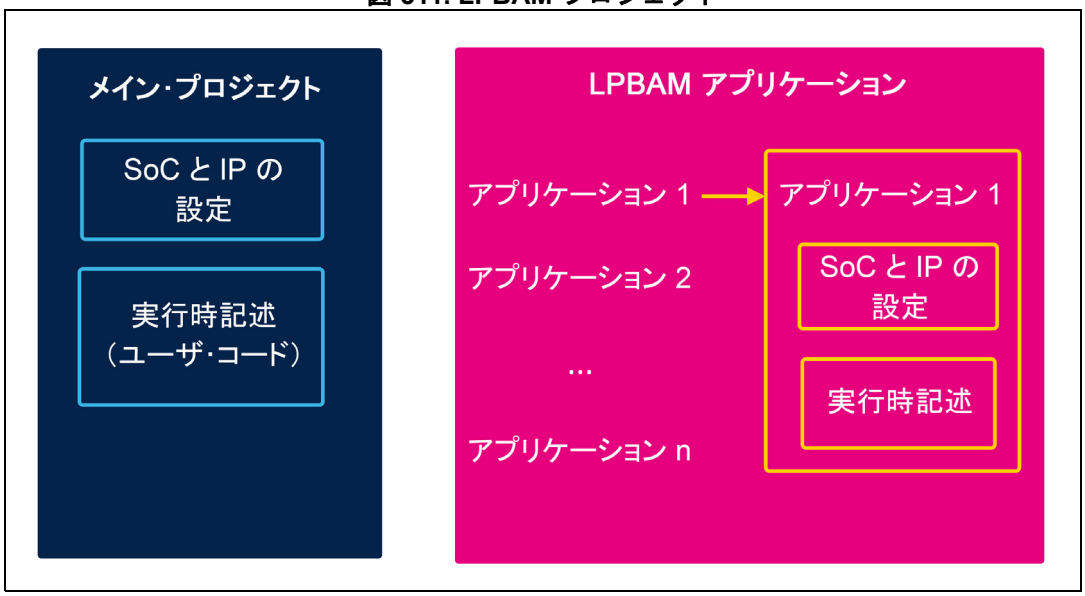
消費電力を削減し、CPU に対する負荷を軽減するために、LPBAM 転送の使用をお勧めします。

- LPBAM のメカニズムは、SmartRun ドメインで ADC4、COMP1/2、DAC1、I2C3、LPDMA1、LPGPIO、LPTIM1/2/3、LPUART1、OPAMP1/2、SPI3、VREFBUF の各ペリフェラル群をサポートします。
- SmartRun ドメインでの LPDMA 実装に従い、LPBAM には SRAM4 のみへのアクセス権があります。
- LPBAM メカニズムの実装は、STOP2 モードまで自律動作が可能です。
- 消費電力が最小限になるように、システムによる電源使用量、システム·クロック、自律性があるペリフェラルのカーネル·クロックを設定できます。

### 18.1.5 STM32CubeMX での LPBAM プロジェクトのサポート

LPBAM プロジェクトは、メイン·プロジェクトと 1 つ以上の LPBAM アプリケーションから構成されます。

**図 311. LPBAM プロジェクト**



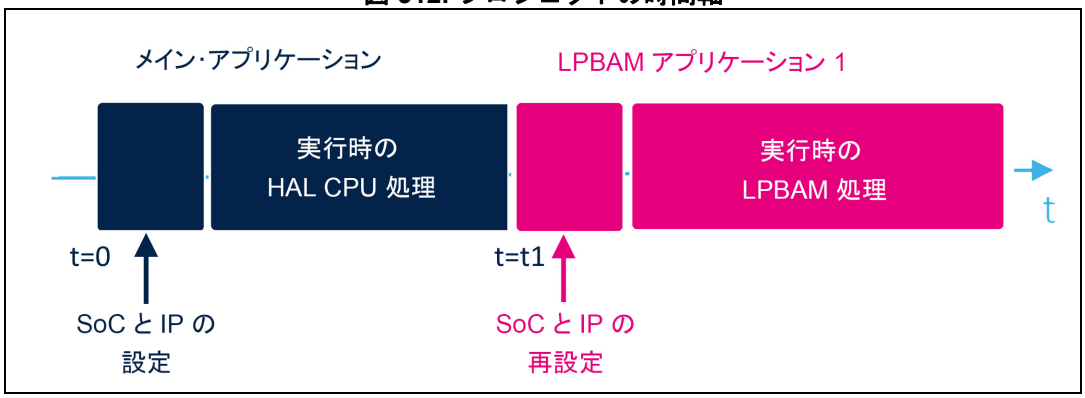

「メイン·プロジェクト」には初期化時に適用される「SoC と IP の設定」とメイン·アプリケーションの実行時の動作記述があります。STM32CubeMX では「SoC と IP の設定」の部分を記述できます。

LPBAM アプリケーションごとに「SoC と IP の設定」と実行時記述があります。STM32CubeMX では両方を記述できます。

「SoC と IP の設定」として STM32CubeMX で生成するコードでは、メイン·プロジェクトと LPBAM アプリケーションの両方で STM32Cube の HAL API と LL API（またはそのいずれか）が使用されます。LPBAM アプリケーションの実行時向けに生成されるコードでは、LPBAM ファームウェア API が使用されます。

図 312 は、メイン·アプリケーションと 1 つの LPBAM アプリケーションで構成した簡単な LPBAM プロジェクトで、実行時に可能な動作の例を示しています。

図 312. プロジェクトの時間軸




## 18.2 LPBAM プロジェクトの作成

### 18.2.1 LPBAM 機能の使用可能性

LPBAM 機能を備えたプロジェクトを開くと、ユーザ·インタフェースに専用のエントリが表示されます（図 313 参照）。この機能の使用は任意であり、使用しなくても、生成されるプロジェクトには何の影響もありません。

図 313. LPBAM 機能を備えたプロジェクト


### 18.2.2 LPBAM プロジェクトの記述

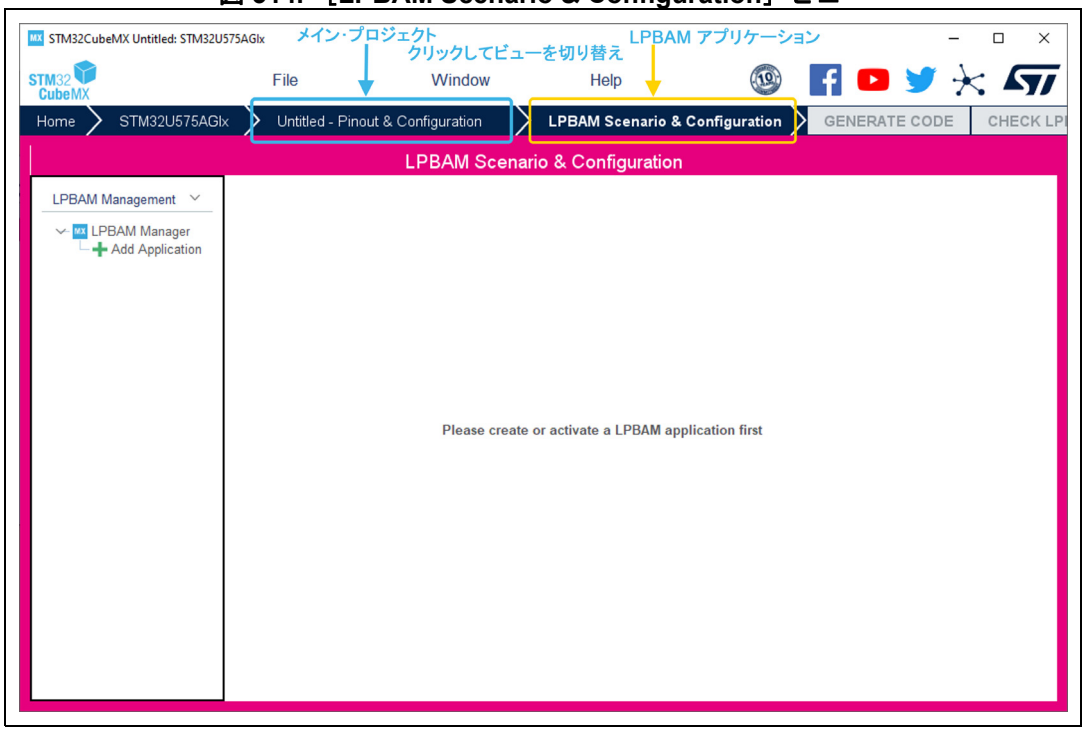
STM32CubeMX の LPBAM プロジェクトの作成では、STM32CubeMX のメイン·プロジェクト·ページを使用してメイン·プロジェクトを記述し、専用の［LPBAM Scenario & Configuration］ページを使用して LPBAM アプリケーションを 1 つ以上記述します。

STM32CubeMX 6.5 以降で次の手順を実行します。

- STM32U575/585　製品ラインのマイクロコントローラまたはボードの品名を選択することでプロジェクトを作成します。
- プロジェクトで TrustZone をアクティブ化しないでください。
- LPBAM 専用ページを表示するには、［LPBAM Scenario & Configuration］リボンをクリックします。

LPBAM のコンテキストがピンク色の枠でハイライトされます。メイン·プロジェクトの設定と［LPBAM Scenario & Configuration］を切り換えるには、該当のリボンをクリックします。

**図 314.［LPBAM Scenario & Configuration］ビュー**




## 18.2.3 プロジェクトでの LPBAM アプリケーションの管理

［LPBAM Scenario & Configuration］ビューでは、まず LPBAM アプリケーションを追加する必要があります。

左側パネルの［LPBAM Manager］セクションで、LPBAM アプリケーションの追加、削除、名前の変更、切り換えが可能です。

最初の LPBAM アプリケーションを追加するには［Add Application］をクリックします。

- デフォルトの名前をそのまま使用する場合は、名前が "LpbamApp1" のアプリケーションが作成されます。
- LpbamApp1 の最初のキュー "Queue1" が作成されます。
- LpbamApp1 の記述に必要な設定ビュー（LPBAM のシナリオ、ピン配置と IP、クロック）が用意されます。

別のキューを新たに追加するには［Add Queue］をクリックします。

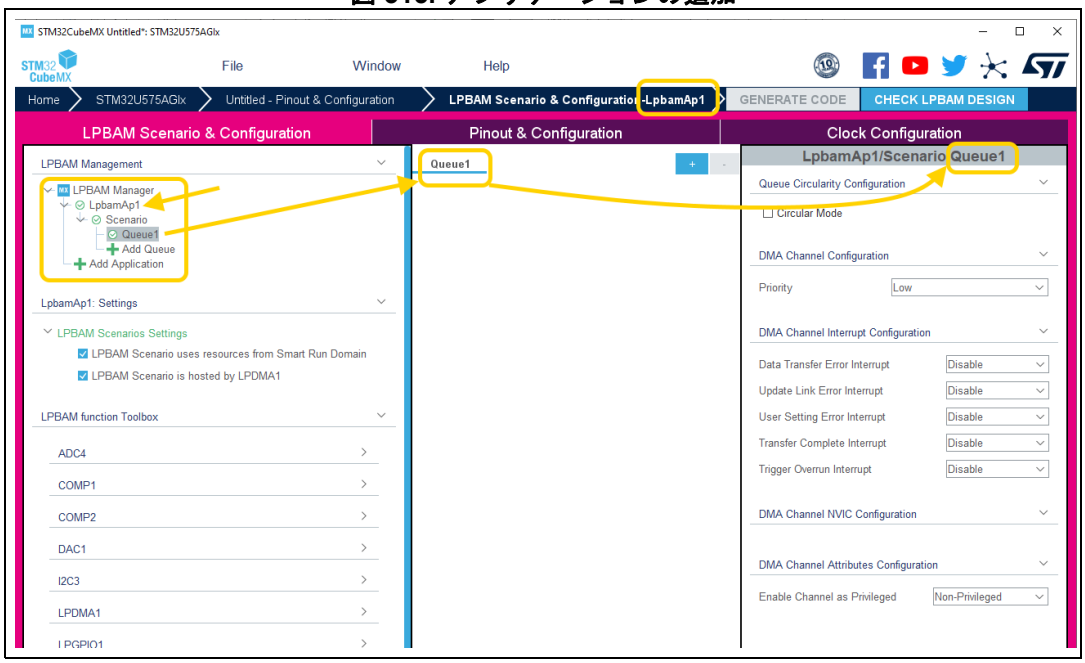
アプリケーション（またはキュー）を削除するには、アプリケーション（またはキュー）名を右クリックして［Delete］を選択します。

アプリケーション（またはキュー）の名前を変更するには、アプリケーション（またはキュー）名を右クリックして［Rename］を選択します。生成されるプロジェクトでは、このアプリケーション名が使用されることに注意してください。

LPBAM アプリケーションを切り換えるには、目的のアプリケーション名をクリックします。選択したアプリケーションの LPBAM パネルが読み込まれます。

LPBAM アプリケーションでキューを切り換えるには、目的のキュー名をクリックします。中央と右側のパネルが更新され、選択したキューとその設定が表示されます。

**図 315. アプリケーションの追加**




## 18.3 LPBAM アプリケーションの記述

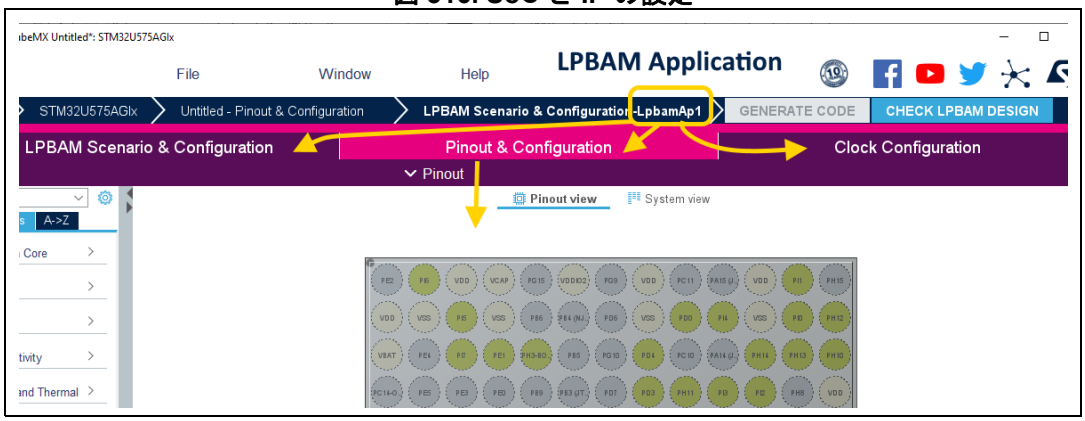
### 18.3.1 概要（SoC と IP の設定、実行時のシナリオ）

LPBAM アプリケーションの作成では、標準の STM32CubeMX プロジェクトと同様に SoC と IP の設定を記述し、アプリケーションの実行時部分を記述します。

#### SoC と IP の設定

LPBAM アプリケーションのコンテキストで IP と SoC を設定するには、LPBAM アプリケーションに用意されている［Pinout & Configuration］と［Clock Configuration］を使用します。

図 316. SoC と IP の設定

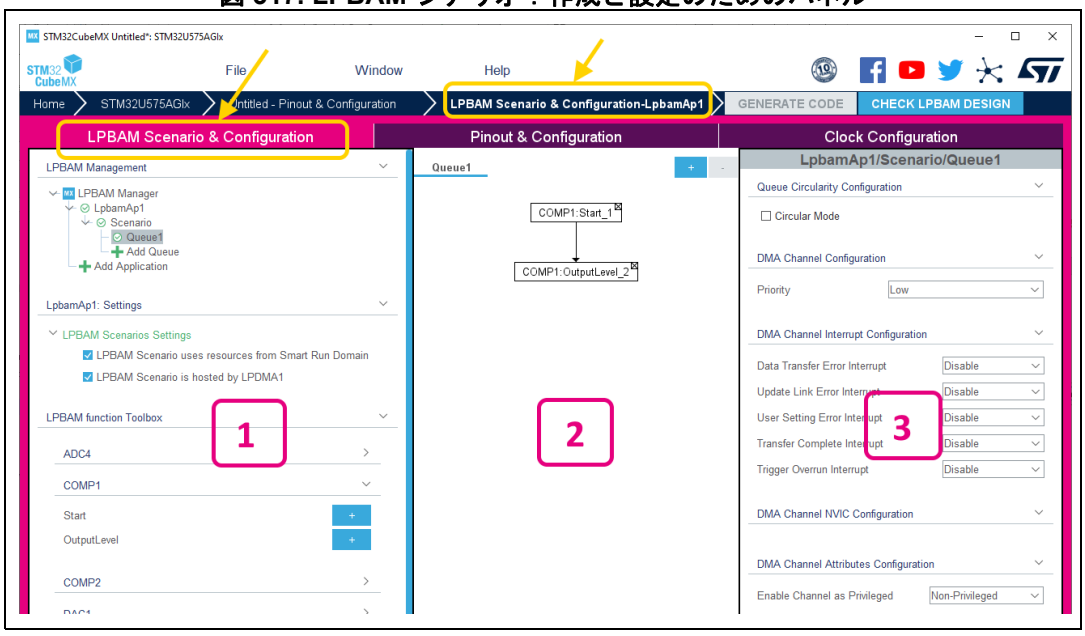



## 実行時部分の記述（シナリオ）

標準の STM32CubeMX プロジェクトでは、STM32Cube の HAL ドライバ API または LL ドライバ API（HAL_COMP_Start、HAL_TIM_Start、HAL_TIM_Stop など）を使用してメイン･アプリケーションの実行時の動作を管理するコードを追加する必要があります。

LPBAM アプリケーション向けに、実行時部分の記述（シナリオ）を作成する［LPBAM Scenario & Configuration］パネルが用意されています。図 317 に示すように、このパネルは 3 つの部分で構成されます。

図 317. LPBAM シナリオ：作成と設定のためのパネル




注： **LPBAM アプリケーションは LPBAM ファームウェア API を使用し、連結した DMA 転送で構成します。**

LPBAM アプリケーションのコンテキストでは、第 1 のパネルを次の目的で使用します。

- アプリケーションのキューの管理
- ノードの参照と STM32CubeMX のユーザ·インタフェースで現在選択しているキューへのノードの追加
- アプリケーション固有の設定。STM32U5 シリーズで LPBAM を使用する場合、これらの設定は変更も無効化もできません。

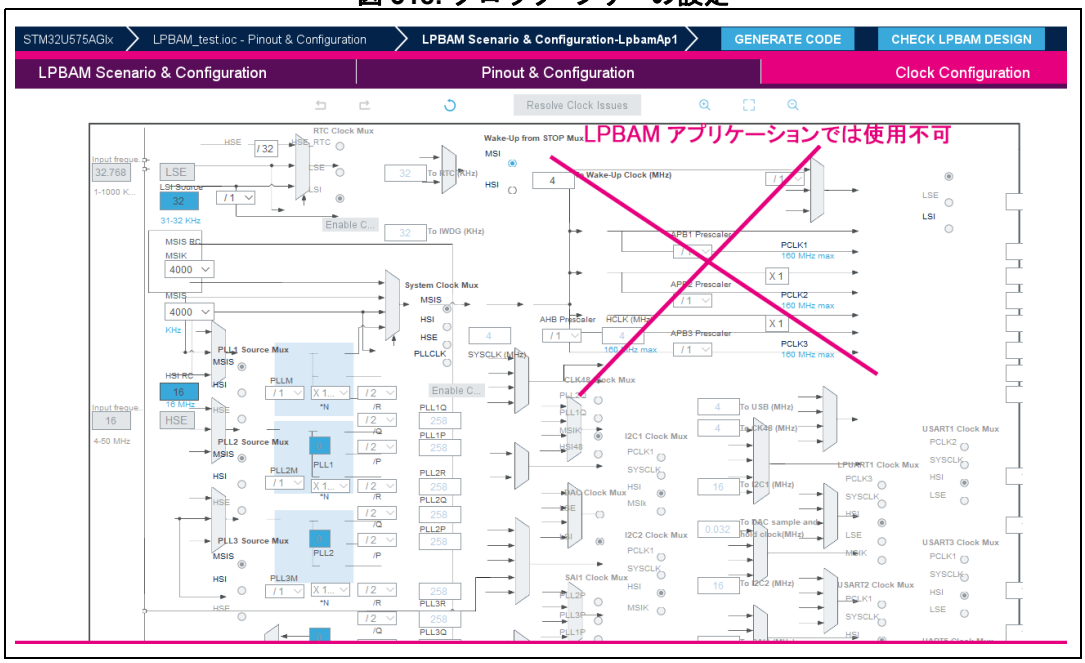
第 2 のパネルには、LPBAM アプリケーションで現在選択しているキューのダイアグラムが表示されます。

第 3 のパネルでは、キュー（キュー名をクリックした場合）またはノード（ダイアグラム上でノードを選択した場合）のいずれかを設定できます。

### 18.3.2 SoC および IP：クロックの設定

LPBAM サブシステムは、STOP2 モード以上で動作し、SmartRun ドメインの IP のみをサポートします。したがって、LPBAM のコンテキストで設定できるクロック·ツリーはその一部にとどまります。STM32CubeMX でクロック·ツリーを設定する方法の詳細は、セクション 4.8：を参照してください。

**図 318. クロック·ツリーの設定**




### 18.3.3 SoC および IP：IP の設定

LPBAM のコンテキストでは、SmartRun ドメインの IP のみを使用できます。

ほとんどの IP はメイン·プロジェクトと同様に設定できます。ただし、追加の設定が必要になる IP もあります。たとえば、LPBAM コンテキストで IP の内部割り込みを使用できる場合、専用の設定タブが表示されます。

図 319. 使用可能な IP

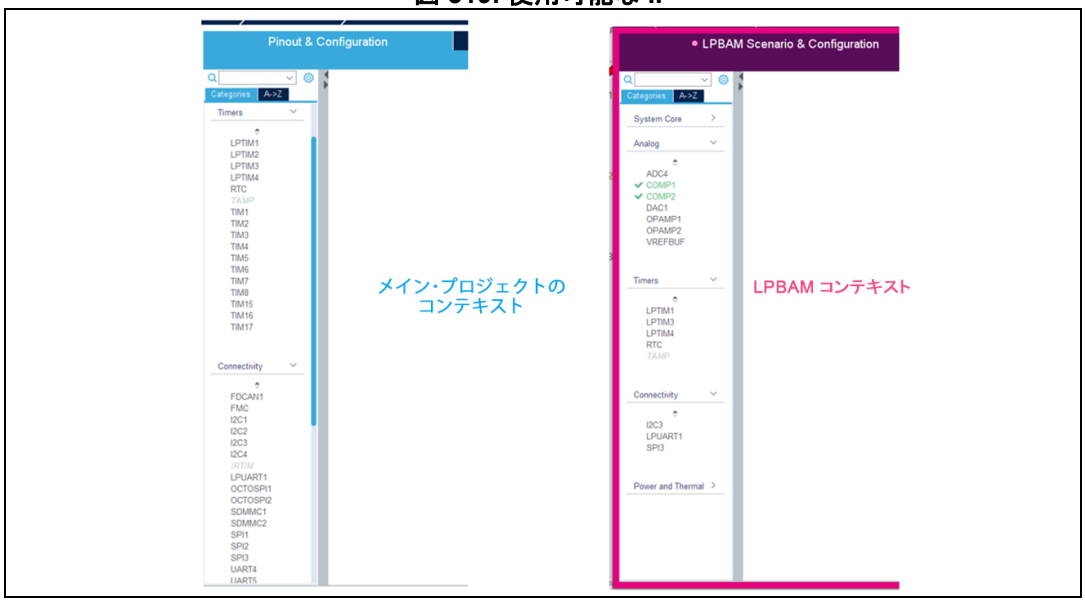
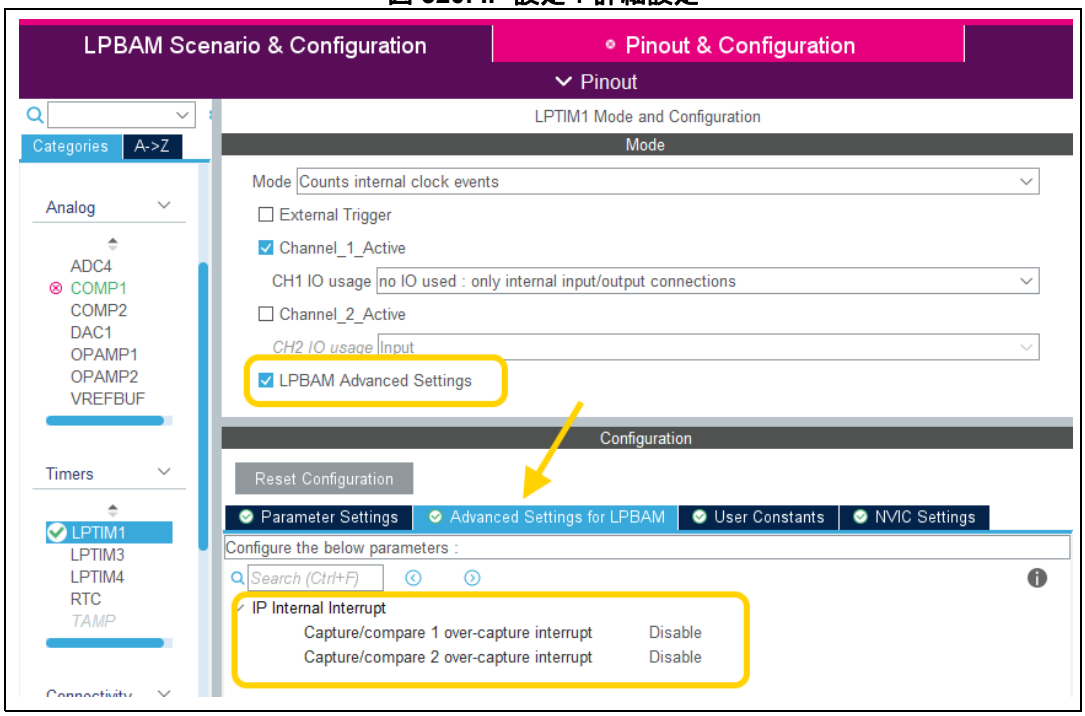



図 320. IP 設定：詳細設定




LPBAM で実行時に使用される IP は、すべて［Pinout & Configuration］ビューで設定する必要があります。それらの設定には、LPBAM シナリオとの整合性が必要です。

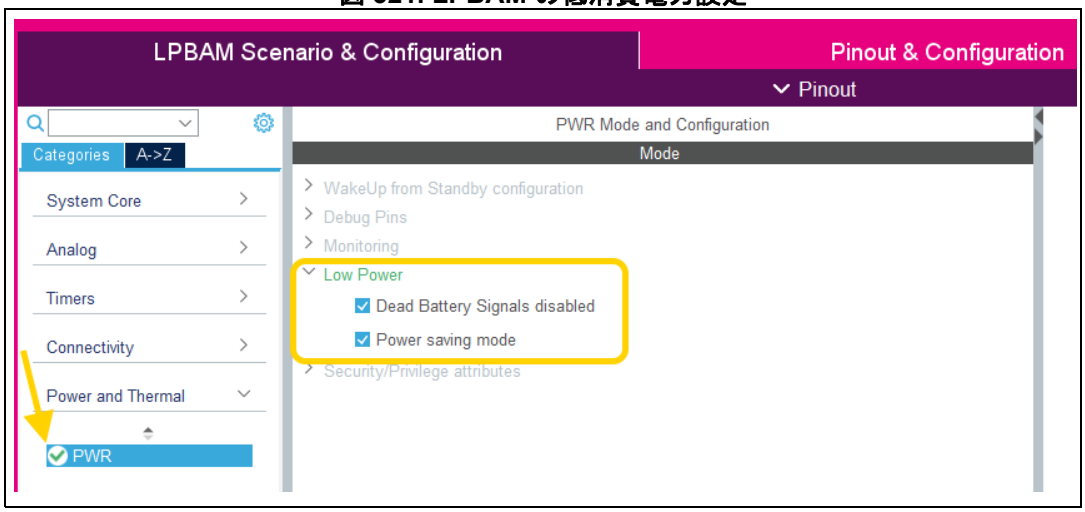
ユーザ・インタフェース右上の［Check LPBAM Design］をクリックすると、使用されている IP のうち LPBAM アプリケーションで設定されていない各 IP について、LPBAM 出力ウィンドウに警告が表示されます。

**警告：** **［Check LPBAM Design］では、IP が［Pinout & Configuration］ビューで設定されているかどうかのみが確認されます。シナリオで使用されている LPBAM API と HAL 設定が整合しているかどうかは確認されません。**

### 18.3.4 SoC および IP：低消費電力の設定

STM32CubeMX6.5 以降、プロジェクトの低消費電力設定を調整できるようになりました。これらの設定（PWR IP の下にあります）は、LPBAM アプリケーションの電力消費を最小限に抑えるうえできわめて重要です。

**図 321. LPBAM の低消費電力設定**




### 18.3.5 LPBAM シナリオ：キューの管理

LPBAM シナリオは 1 つ以上のキューから構成され、各キューには 1 つ以上のノードがあります。中央パネルは、LPBAM アプリケーションのシナリオを記述します。キュー名をクリックすると中央パネルにキュー･ダイアグラム、右側パネルにキューの設定が表示されます。選択したキューの名前には青色の下線が表示されます。

キューを追加するには、そのパネルの［+］ボタンをクリックするか、左側パネルの LPBAM 管理セクションにある［Add Queues］をクリックします。

- STM32U5 シリーズではキューの最大数は 4 です。これは LPDMA1 チャネル数による制約です。
- プロジェクトに LPBAM アプリケーションを追加すると、そのアプリケーション用に空のキューが 1 つ自動的に作成されます。

**警告：** **複数のキューを持つ LPBAM アプリケーションの場合、実行時のキューどうしの同期が STM32CubeMX では管理されません。アプリケーション実行時の各種キューの開始は、最終的なアプリケーションをアセンブルする際にユーザ側で管理する必要があります。**

［LPBAM Management］セクションでは、次のようにキューの削除と名前の変更が可能です。

- キューを削除するには、キュー名を右クリックして［Delete］を選択します。
- キューの名前を変更するには、キュー名を右クリックして［Rename］を選択します。
- LPBAM アプリケーションでキューを切り換えるには、目的のキュー名をクリックします。中央と右側のパネルが更新され、選択したキューとその設定が表示されます。

### 18.3.6 キューの記述：ノードの管理

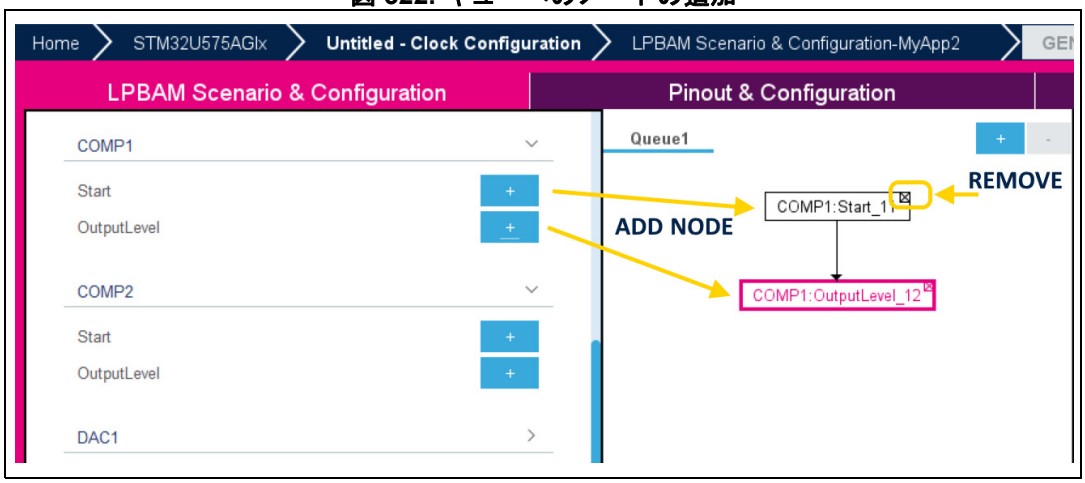
キューの記述は、時系列に並べられた機能ノードのシーケンスから構成されます。シーケンスは中央パネルにダイアグラムとして表示され、キューの設定が右側パネルに表示されます。

キューにノードを追加するには、次の手順を実行します。

- 更新するキューの名前をクリックします。
- 左側パネルの［LPBAM function Toolbox］で、ノードの作成に使用できる IP と関数（LPBAM ファームウェア API）のリストを参照します。
- IP 名をクリックすると、使用できる関数のリストが展開表示されます。
- 関数名の横の「+」符号をクリックすると、その関数がキューにノードとして追加され、中央パネルのキュー･ダイアグラムが更新されます。
- 例：LpbamAp1 の Queue1 で COMP1 が開始され、COMP1 出力にデータが転送されます（図 322 参照）。

ダイアグラムからノードを削除するには、ノードの右上隅にある×印をクリックします。

**図 322. キューへのノードの追加**

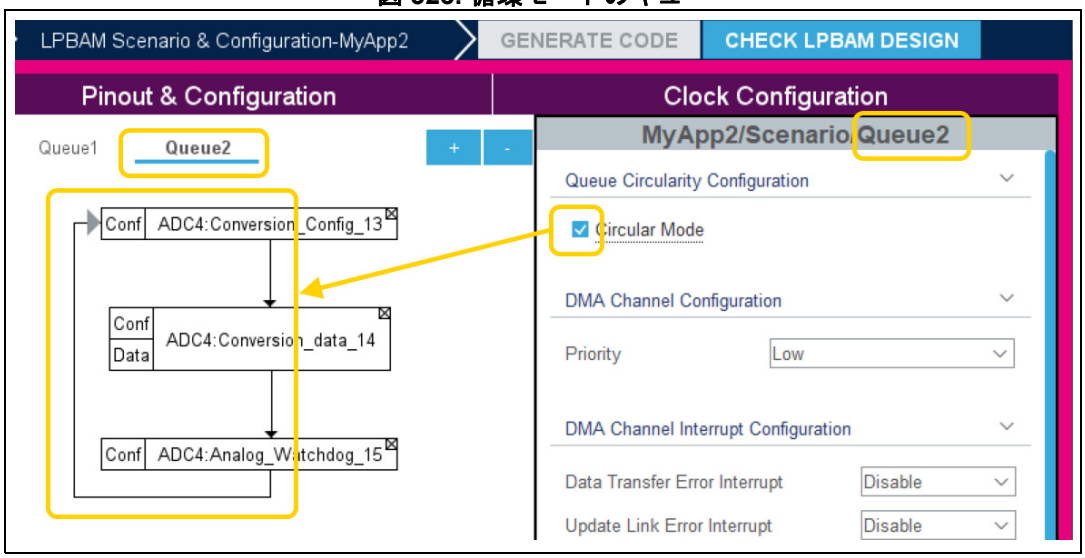


### 18.3.7 キューの記述：循環モードのキューの設定

STM32CubeMX では循環キューを設計できます。

- 中央パネルでキュー名をクリックして、設定するキューを選択します。右側のパネルにキューの設定が表示されます。
- キューを循環モードに設定するには、［Circular Mode］チェックボックスをクリックします。デフォルトでは、キューのループは先頭のノードに戻ります（図 323 参照）。
- 先頭以外のノードに戻すには、ダイアグラムで矢印の端をクリックして、目的のノードまでドラッグします。
- ループを削除するには、［Circular Mode］のチェックを外します。

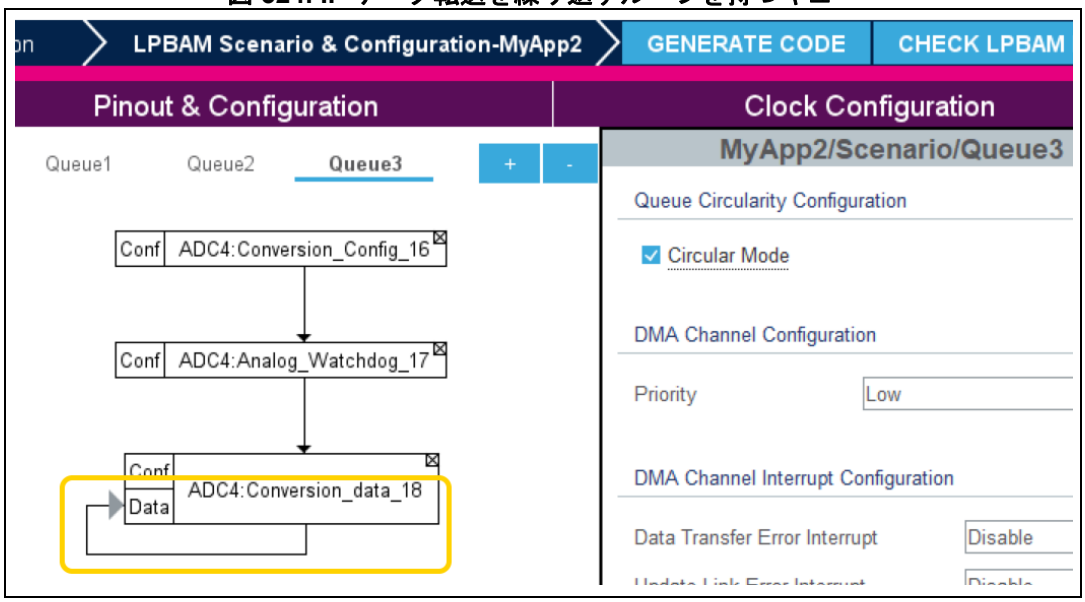
**図 323. 循環モードのキュー**




関数によっては、IP を設定してからデータ転送を扱うものがあります。循環モードのループは、関数の設定部分（Conf）またはデータ部分（Data）に接続できます。

図 324 に例を示します。このキューを実行すると、1 番目から 3 番目までのノードの設定（Conf）は 1 回だけ実行され、データ転送部分はループとして繰り返されます。

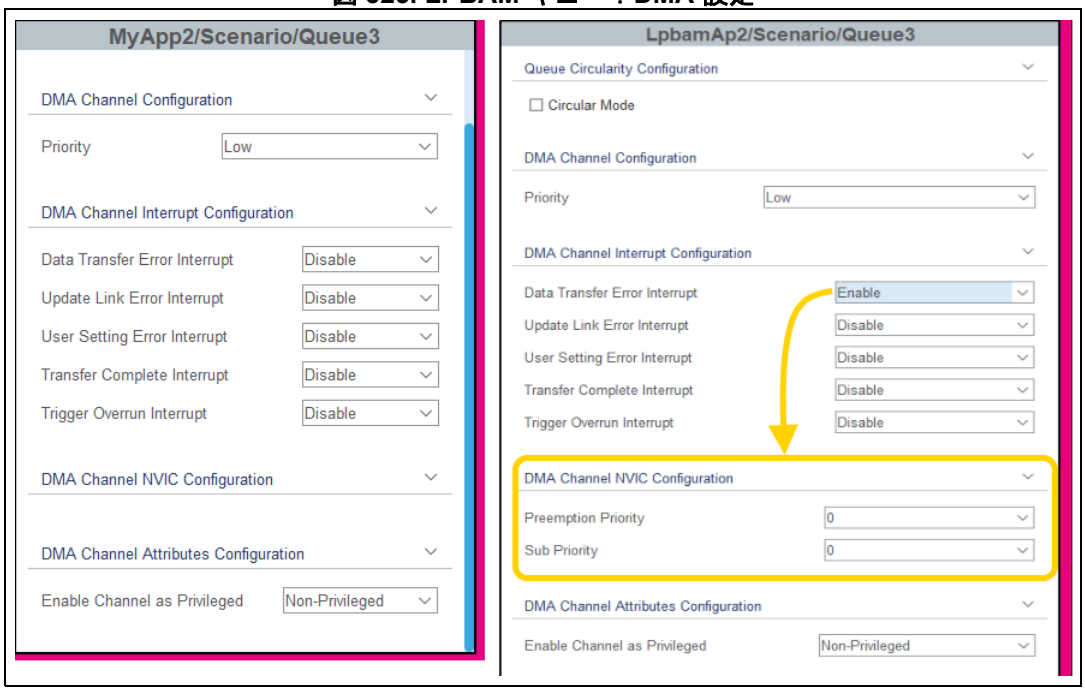
図 324. IP データ転送を繰り返すループを持つキュー




## 18.3.8 キューの記述：キューをホストする DMA チャネルの設定

LPBAM のキューの実行は、連結された LPDMA 転送で構成されます。アプリケーションのニーズに応じて、キューの実行をホストする DMA を設定する必要があります（図 325 参照）。

図 325. LPBAM キュー：DMA 設定



### 基本設定

中央パネルでキュー名をクリックして、設定対象のキューを選択します。そのキューをホストするDMA チャネルの設定が右側パネルに表示されます。

通常なら使用できる DMA チャネルの設定の一部が、このユーザ·インタフェースには表示されないことに注意してください。それらの設定は、STM32CubeMX または LPBAM ドライバによって直接管理されるためです。

### DMA チャネルの NVIC 設定

NVIC 設定は、DMA チャネル割込みが 1 つ有効な場合にのみ使用できます（図 325 の右側パネル参照）。LPBAM コンテキストでの横取り優先順位とサブ優先順位の範囲は、プロジェクト（LPBAM アプリケーションを伴うメイン·プロジェクト）全体に設定された NVIC 優先順位グループで決まります。

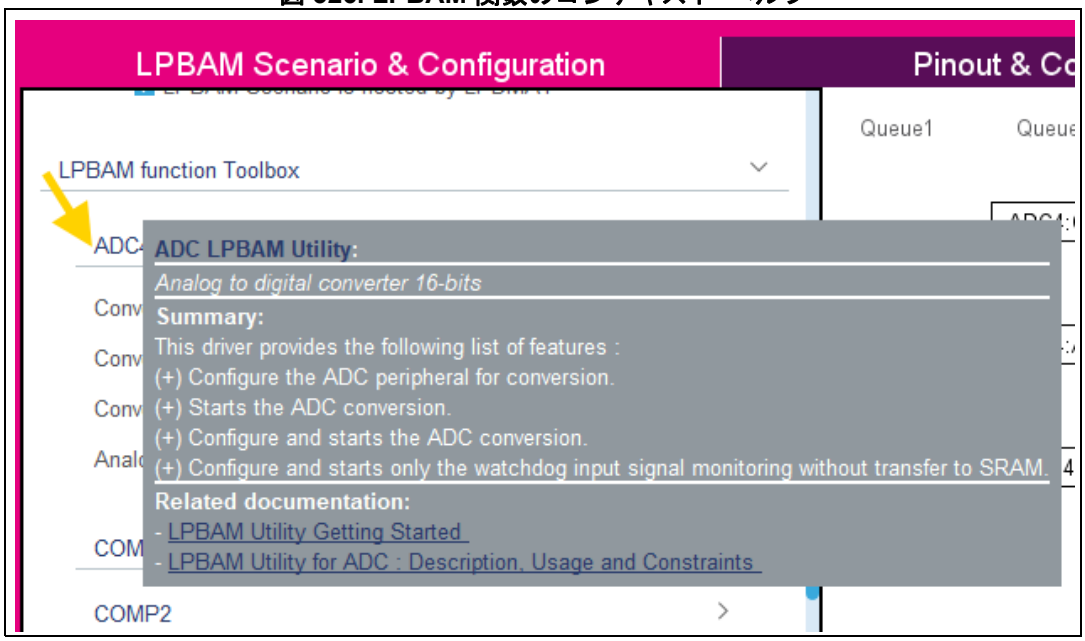
**警告： メイン·プロジェクトの［Pinout & Configuration］ビューで NVIC 優先順位グループを変更した後は、必ず LPBAM コンテキストの横取り優先順位とサブ優先順位を確認してください。**

## 18.3.9 ノードの記述：コンテキスト·ヘルプとドキュメントへのアクセス

STM32CubeMX には、コンテキスト·ヘルプと LPBAM 関数に関する参照ドキュメントへのリンクが用意されているので、関数の選択手順を確認できます。

- 左側パネルの［LPBAM function Toolbox］で IP 名の上にマウス·カーソルを置くと、参照ドキュメントへのリンクを記述したコンテキスト·ヘルプが表示されます（図 326 参照）。
- キューにノードを配置する方法や使用できる機能と制約などを把握するために、LPBAM のグローバル·ドキュメントと IP の「説明、使用法、制約」を十分に理解することをお勧めします。LPBAM のメカニズムに起因する制約がいくつかあります。これらの制約は、IP 自体または HAL の制約によるものではありません。

**図 326. LPBAM 関数のコンテキスト·ヘルプ**

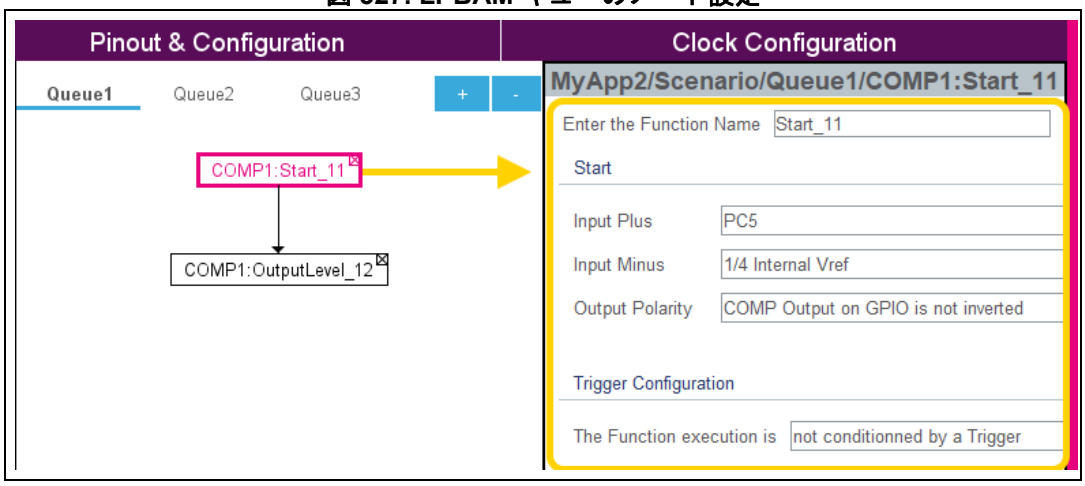


### 18.3.10 ノードの記述：ノード･パラメータの設定

［LPBAM Function Toolbox］で関数を選択してキューに追加すると、その関数を設定できます。中央パネルでノードをクリックして選択します。ノードの関数がピンクでハイライトされ、その設定が右側パネルに表示されます（図 327 参照）。

ここに示した例では、LPBAM の COMP1_Start 関数の "Start" パラメータが表示されています。HAL ドライバでも、COMP IP の設定でこれと同じパラメータ名が使用されます。前述のとおり、LPBAM ファームウェアは HAL ドライバではありません。しかし、IP が一意であることから、LPBAM ドライバは、HAL ドライバで使用されている IP パラメータと同じ名前を可能な限り使用するように設計されています。

**図 327. LPBAM キューのノード設定**




**警告： LPBAM IP 関数は、［Pinout & Configuration］ビューで適切に設定された、IP のハードウェア･リソースにアクセスします。**

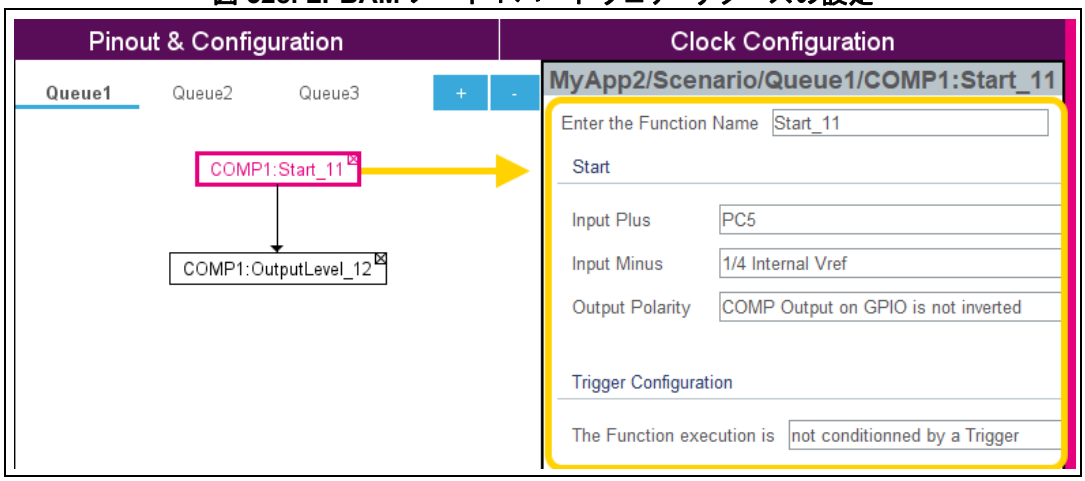
GPIO などのハードウェア･リソースにパラメータを設定する場合、そのリソースを［Pinout & Configuration］ビューで設定する必要があります。

図 327 に示した例では、COMP の［Input Plus］として PC5 を設定しています。PC5 が［Pinout & Configuration］ビューで設定されていないと、生成された LPBAM アプリケーションは "Input Plus" に "Null 信号" を受信する可能性があり、その場合は機能しません。

この問題を解決するには、次の手順を実行します。

- ［Pinout & Configuration］ビューに移動します。
- 検索フィールドを使用して PC5 を検索します。
- PC5 ピンを右クリックして［COMP_Inp］を選択します（図 328 参照）。

図 328. LPBAM ノード：ハードウェア･リソースの設定




タイマを使用して PWM 信号を生成する例も作成できます。HAL ドライバでは、タイマ･チャネルを出力として設定する必要があります。これは、LPBAM ファームウェアを使用する場合も同様です。

注： **IP の初期設定に関するすべての制約は、LPBAM ファームウェアのドキュメントに記載されています。設定漏れを検出するには STM32CubeMX の［LPBAM Design Check］機能を使用します（該当のセクションを参照してください）。**

### 18.3.11 ノードの記述：トリガの設定

すべての IP と関数で、LPBAM ファームウェアにより、ハードウェア信号を使用してノードをトリガできます。STM32CubeMX では、ノード設定パネルでそのようなトリガを設定できます。デフォルトでは、ノード実行はトリガされません。トリガを有効にすると、使用できるトリガ信号がすべて一覧表示されます。

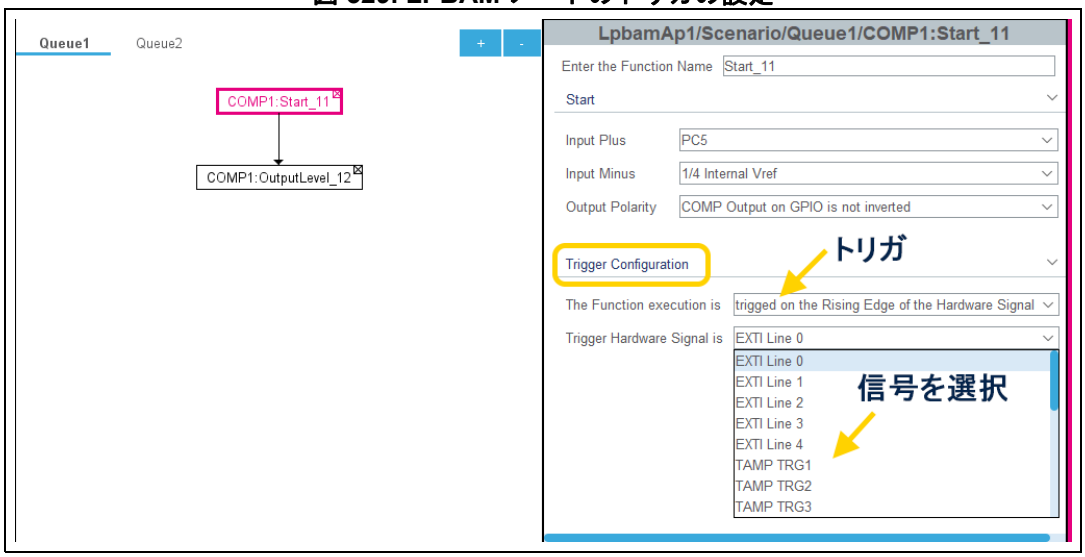
---

**警告： トリガはユーザの責任で適切に設定する必要があります。**
**STM32CubeMX では設定エラーが確認されません。**

---

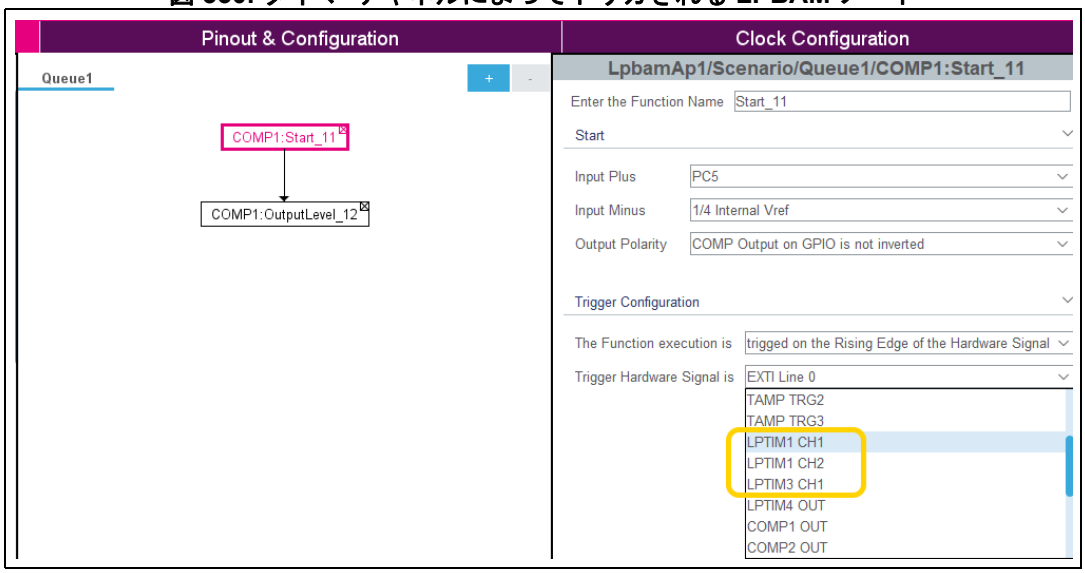
たとえば、COMP 関数の "Start" で（図 329 参照）、ハードウェア信号の立ち上がりエッジで関数の実行がトリガされるように選択し、表示されているハードウェア信号の一覧から目的の信号を選択します。

図 329. LPBAM ノードのトリガの設定




ノードが LPTIM1_CH1 を扱う関数であれば、LPTIM1_CH1 をトリガとして選択できます（図 330 参照）。

図 330. タイマ･チャネルによってトリガされる LPBAM ノード

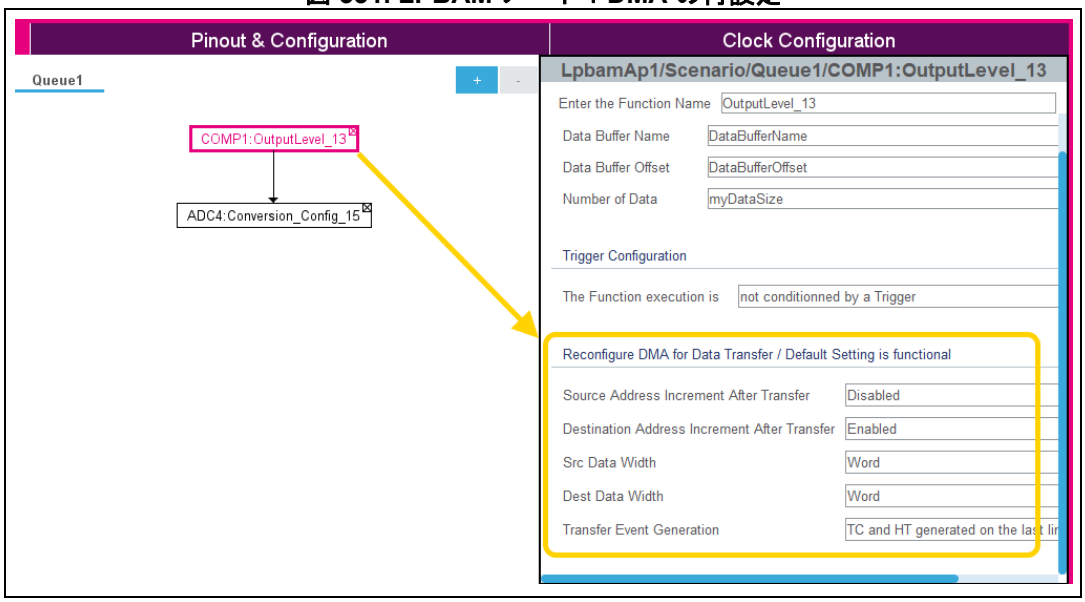



## 18.3.12 ノードの記述：データ転送用の DMA の再設定

データ転送を扱う関数（データ転送を伴い、名前の末尾が _Config ではない、すべての関数）を設定したノードには、固有の設定セクション［Reconfigure DMA for Data Transfer］があります（図 331 参照）。

各 DMA データ転送は、データ･サイズ、バッファ･アドレス、アドレス･インクリメントをはじめとする固有設定に基づいて実行されます。DMA は、デフォルト設定のままで機能します。

図 331. LPBAM ノード：DMA の再設定

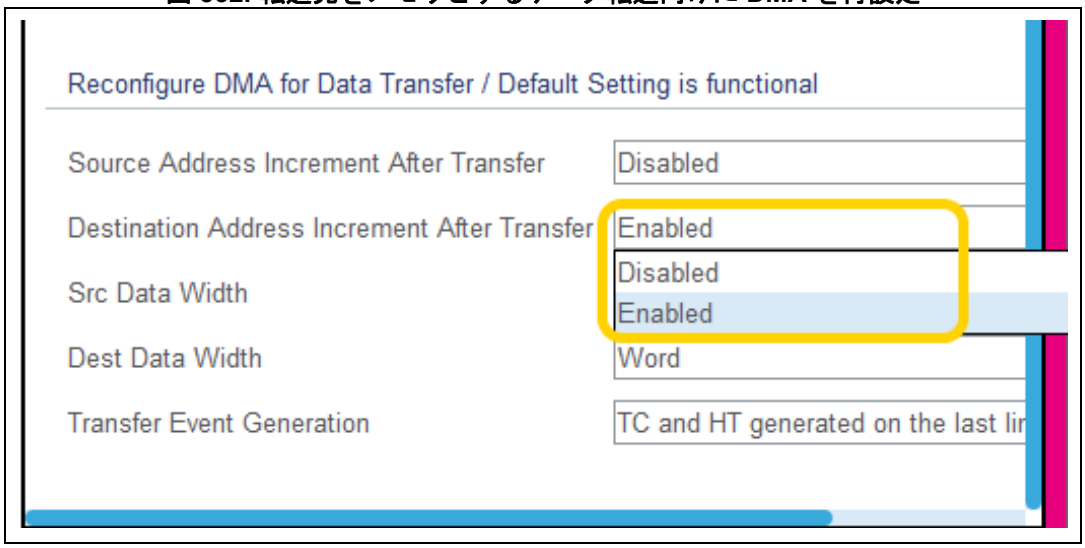



DMA 設定の変更の可否は IP と関数によって決まります。

たとえば、"COMP Output Level" の場合は次のようになります。

- 転送されるデータは出力データであり、レジスタ IP からメモリに転送されます。IP のデータ・レジスタを参照する転送元アドレスはインクリメントされません。STM32CubeMX のユーザ・インタフェースには、"Source address increment after transfer"パラメータを有効化できないことが示されます。
- メモリに転送されるデータは、同じメモリ・アドレス、またはテーブルに保存できます。テーブルに保存する場合、"Destination address increment after transfer" は無効でも有効でもかまいません（図 332 参照）。

図 332. 転送先をメモリとするデータ転送向けに DMA を再設定



## 18.4 LPBAM 設計の確認

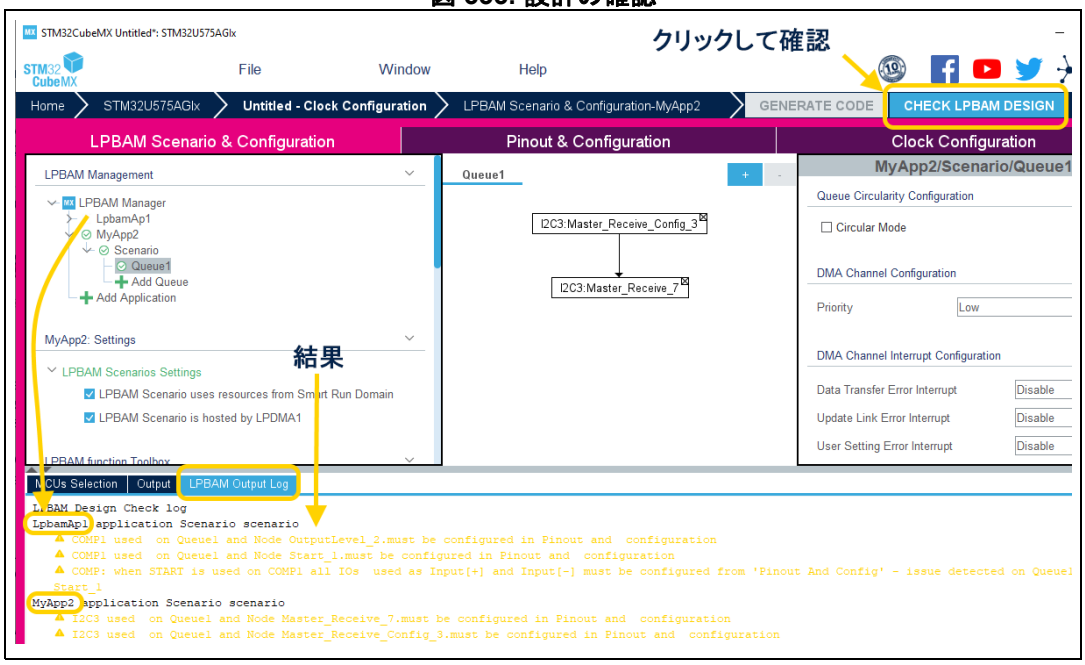
STM32CubeMX には、以下の項目を検出することによって、LPBAM 設計の一貫性と完全性を確認する機能があります。

- ノードに選択された IP の LPBAM 関数と、それに対応する IP 設定との不整合
- 誤ったキュー設計（ノードのシーケンスが無効）

プロジェクトで現在使用可能なすべての LPBAM アプリケーションを確認するには［CHECK LPBAM DESIGN］をクリックします。結果は、LPBAM 出力ログ･ウィンドウに表示されます（図 333 参照）。

**注：** **LPBAM 設計に関して警告が表示されても、プロジェクトの C コードは生成できます。**

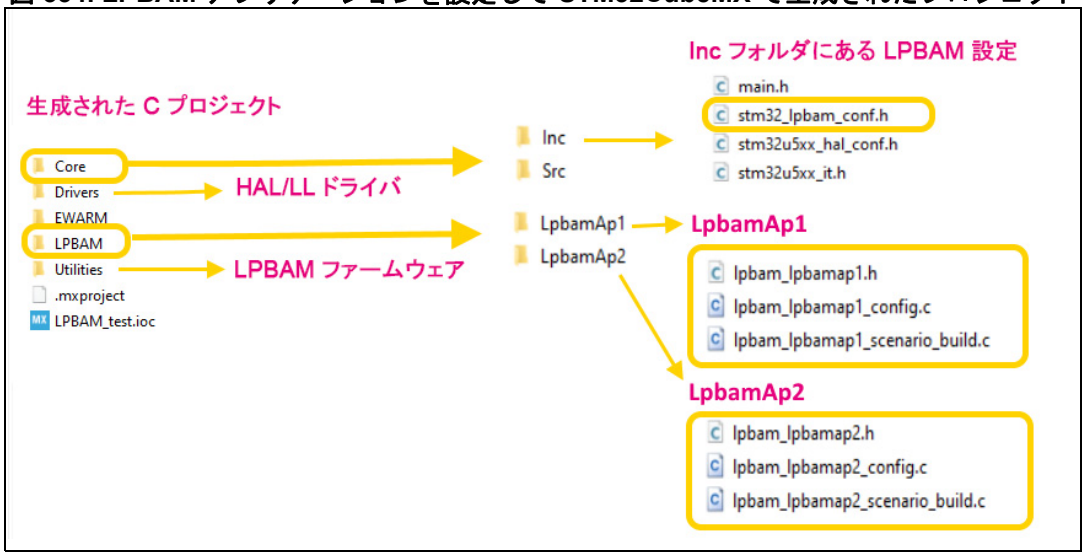
**図 333. 設計の確認**



## 18.5 LPBAM アプリケーションを設定したプロジェクトの生成

メイン・プロジェクト・ビューの［Generate Code］をクリックします。図 334 の例にあるように、生成されたプロジェクトには、メイン・プロジェクトのファイルとフォルダのほか、stm32_lpbam_conf.h ファイル、LPBAM アプリケーションの設定コードを保存する LPBAM 専用のフォルダ、LPBAM ユーティリティ・ファームウェアを保存するユーティリティ フォルダが表示されます。

**図 334. LPBAM アプリケーションを設定して STM32CubeMX で生成されたプロジェクト**




STM32CubeMX は、以下を生成します。

- Core/Inc フォルダに保存された stm32_lpbam_conf.h ファイル。このファイルは、LPBAM アプリケーションで有効化される LPBAM モジュールのすべてを定義し、LPBAM ユーティリティ・ファームウェアで使用されます。
- LPBAM フォルダに保存された、LPBAM アプリケーションとそのシナリオで使用されるコード。lpbam_<アプリケーション名>.h ファイルは、メイン・プロジェクトで呼び出す関数のプロトタイプを提供します。これらの関数は、アプリケーションの初期化、シナリオの構築と初期化、シナリオと DMA との接続、シナリオの開始と停止、シナリオとのリンクの解除、シナリオの初期化解除を実行します。

たとえば、LpbamAp1 アプリケーションの場合、STM32CubeMX は次のような関数を生成します。

```
/* LpbamAp1 application initialization */
void MX_LpbamAp1_Init(void);

/* LpbamAp1 application - scenario initialization */
void MX_LpbamAp1_Scenario_Init(void);

/* LpbamAp1 application - scenario build */
void MX_LpbamAp1_Scenario_Build(void);

/* LpbamAp1 application - scenario link */
void MX_LpbamAp1_Scenario_Link(DMA_HandleTypeDef *hdma);

/* LpbamAp1 application - scenario start */
```

```
void MX_LpbamAp1_Scenario_Start(DMA_HandleTypeDef *hdma);

/* LpbamAp1 application - scenario stop */
void MX_LpbamAp1_Scenario_Stop(DMA_HandleTypeDef *hdma);

/* LpbamAp1 application - scenario unlink */
void MX_LpbamAp1_Scenario_UnLink(DMA_HandleTypeDef *hdma);

/* LpbamAp1 application - scenario de-initialization */
void MX_LpbamAp1_Scenario_DeInit(void);
```

# 19 よくある質問

## 19.1 STM32CubeMX からのダウンロード中にネットワーク接続エラーが発生します。

ダウンロード中にネットワーク接続の問題が発生した場合は、［Help］→［Updater Settings］を選択して接続状態を確認してください。

［Check Connection］ボタンに緑色のチェック･マークが表示されている場合は（接続中）、それをクリックします。ボタンが赤紫色の×印（接続失敗）に変化します。使用中のネットワーク設定に適合するようにパラメータを調整して、再度［Check Connection］ボタンをクリックします。緑色のチェック･マークが表示されれば（接続が復帰）、ダウンロードを継続できます。

## 19.2 インターネットにアクセスするログインを変更して以来、一部のソフトウェア･パッケージが使用できなくなったようです。

回避策は、ユーザ･ディレクトリから stm32cubemx フォルダを削除して、STM32CubeMX を再起動することです。この場合、［From URL］ボタンを使用してインストールしたソフトウェア･パッケージをすべて再インストールする必要があります。

［Help］→［Updater Settings］を選択して、以下の手順を実行します。

- **［Updater Settings］**タブでリポジトリ･パスの設定を確認します。
- プロキシを使用している場合は、**［Connection Parameters］**タブにログインとパスワードの情報を入力します。

## 19.3 デュアルコンテキストの製品で、特定のコンテキストでは使用できないペリフェラルまたはミドルウェアがあるのはなぜですか。

ペリフェラルとミドルウェアの中には、それと同じコンテキストで別のペリフェラルやミドルウェアの有効化を必要とするものがあるからです。

たとえば、LwIP ミドルウェアでは、それと同じコンテキストで ETH ペリフェラルを有効にする必要があります。

## 19.4 ピン配置の設定パネルで、新しいペリフェラルを追加すると、STM32CubeMX によって移動される機能があるのはなぜですか。

Keep Current Signals Placement を選択解除している可能性があります。その場合は、配置を最適化するために再マッピングが自動的に実行されます。

## 19.5 手動で機能を強制的に再マッピングするにはどうすればいいですか。

**手動再マッピング**の機能を使用します。

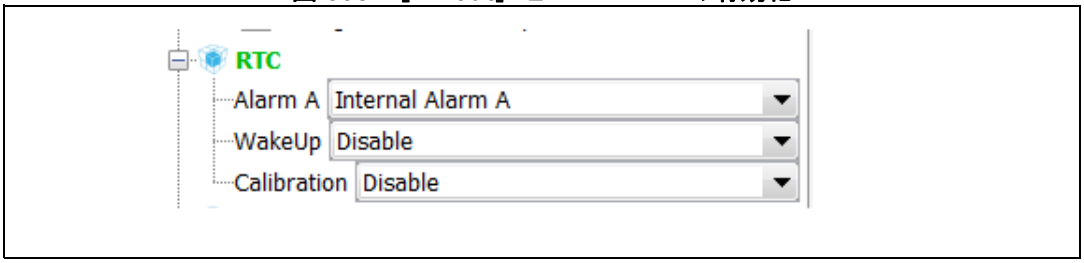
## 19.6 ［Pinout］ビューで、ハイライトが黄色のピンと薄い緑色のピンがあるのはなぜですか。機能を変更できないピンがあるのはなぜですか（それらのピンをクリックしても何も変化しません）。

これらのピンは、ペリフェラルの信号に使用できない固有のピンです（電源や BOOT など）。

## 19.7 ［Clock tree］ビューで RTC マルチプレクサが非アクティブのままになるのはなぜですか。

RTC マルチプレクサを有効にするには、次に示すように［**Pinout**］ビューで RTC ペリフェラルを有効にする必要があります。

**図 335. ［Pinout］ビュー - RTC の有効化**




## 19.8 クロック･ソースとして LSE と HSE を選択して周波数を変更するにはどうすればいいですか。

**[Pinout]** ビューで RCC を適宜設定すると、LSE クロックと HSE クロックがアクティブになります。例を図 336 に示します。これでこれらのクロック･ソースの周波数を変更し、外部ソースを選択できます（図 337 を参照）。

**図 336. ［Pinout］ビュー - LSE クロックと HSE クロックの有効化**

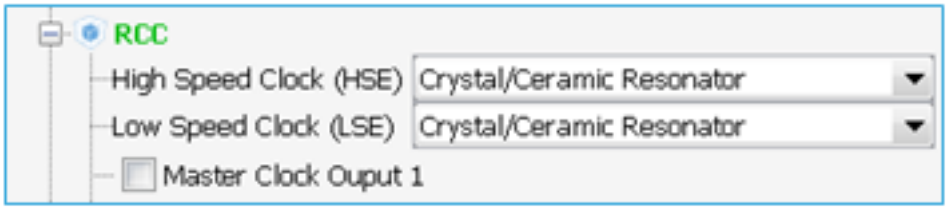
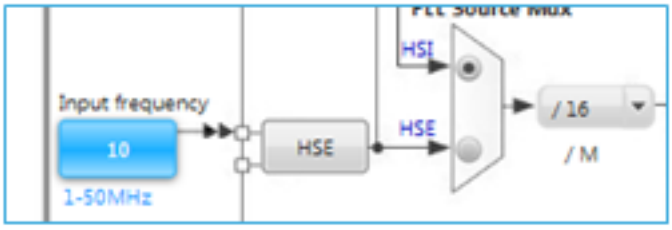



**図 337. ［Pinout］ビュー - LSE/HSE のクロック周波数の設定**



## 19.9 STM32CubeMX で、PC13、PC14、PC15、PI8 のいずれかを出力として設定済みであっても、これらを出力として設定できないのはなぜですか。

STM32CubeMX では次の制約を実装しています。この制約は、リファレンス･マニュアルで出力電圧特性に関する表に脚注として記述されています。

“PC13、PC14、PC15、および PI8 は電源スイッチを経由して接続されます。このスイッチに流すことができる電流には上限があるので（3 mA）、GPIO の PC13、PC14、PC15、および PI8 を出力モードとした場合は、最大負荷 30 pF で最大周波数が 2 MHz を超えないこと、およびこれらの I/O を電流源として使用しないこと（LED の駆動など）という使用上の制限があります。”

## 19.10 Ethernet 設定：DP83848 や LAN8742A を指定できない場合があるのはなぜですか。

STM32CubeMX ではほとんどのシリーズで、選択した Ethernet のモードに応じて、使用できる PHY コンポーネント･ドライバの一覧が次のように調整されます。

- Ethernet MII モードを選択すると、DP83848 コンポーネント･ドライバまたは "ユーザ PHY" を選択できます。
- Ethernet RMII モードを選択すると、LAN8742A コンポーネント･ドライバまたは "ユーザ PHY" を選択できます。

"ユーザ PHY" を選択した場合は、使用するコンポーネント･ドライバをプロジェクトで手動指定する必要があります。

注： **STM32H7 シリーズの場合、PHY は外部コンポーネントとして認識されるので、Ethernet ペリフェラルの設定では指定されなくなります。LwIP の［Platform Settings］タブで PHY を選択できます。なお、STM32H7 ファームウェア･パッケージでは、すべての STM32H7 の評価ボードと Nucleo ボードで使用できる LAN8742A コンポーネントのドライバ･コードのみが用意されているので、ユーザ･インタフェースでは "ユーザ PHY" または LAN8742 のみを選択できます。**

**LAN8742 を選択すると、生成したプロジェクトに BSP ドライバ･コードがコピーされます。**

## 19.11 どうすれば STM32CubeMX で生成するプロジェクトで MX_DMA_Init の呼び出しランクを固定できますか。

DMA を使用する場合、MX_DMA_Init は必ず他のあらゆる HAL_***_Init（*** は DMA 初期化コードに対するハードウェア依存性を持つペリフェラルの名前）よりも前に呼び出す必要があります。

STM32CubeMX バージョン 6.3.0（STM32CubeIDE バージョン 1.7.0）では、初期化関数の呼び出しが誤った順序で生成される不具合が発生しました。

この問題は、新規作成されるプロジェクトに対しては STM32CubeMX 6.4.0 で対策されました。しかし、STM32CubeMX 6.3.0 で生成したプロジェクトの .ioc ファイルに保存された呼び出し順序は、そのファイルを STM32CubeMX 6.4.0 で開いて再生成しても引き続き不具合の原因となります。

この問題を修正するには、保存済みの .ioc ファイル（6.3.0 バージョンで作成済みのファイル）を、使用中のマシンにインストールされている任意のテキスト･エディタで開き、次の行を削除します。ProjectManager.functionlistsort=....

変更を保存したら、CubeMX バージョン 6.4.0（STM32CubeIDE バージョン 1.8.0）以降を使用して、その .ioc ファイルを再度開きます。［Project Manager］ビュー→［Advanced Settings］タブで初期化関数が正しい順序で配置されていることを確認し、プロジェクトを再生成します。

# 付録 A STM32CubeMX のピン割当てルール

STM32CubeMXでは、次のピン割当てルールが実装されています。

- ルール 1：ブロックの整合性
- ルール 2：ブロックの相互依存性
- ルール 3：1 つのブロックを 1 つのペリフェラルとするモード
- ルール 4：ブロックの再マッピング（STM32F10x のみ）
- ルール 5：機能の再マッピング
- ルール 6：ブロックのシフト（STM32F10x のみ）
- ルール 7：ペリフェラル･モードの設定またはクリア
- ルール 8：機能の個別マッピング（[Keep Current Signals Placement] をチェックしていない場合）
- ルール 9：GPIO 信号マッピング

## A.1 ブロックの整合性

ピン･シグナルを設定すると（対応するペリフェラル･モードに関する曖昧さがない場合）、このモードで必要なピンとシグナルとの組合せがすべてマッピングされ、ピンが緑色で表示されます（曖昧さがある場合、設定したピンはオレンジ色で表示されます）。

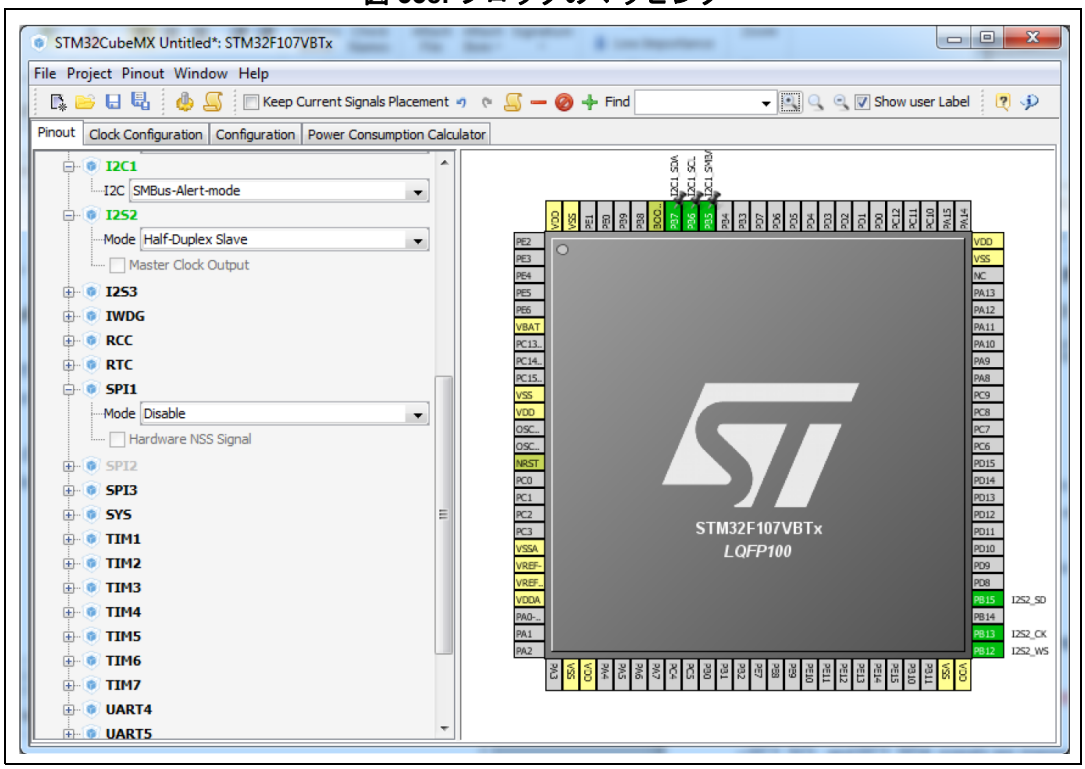
ピンのシグナルをクリアすると、このモードで必要なピンとシグナルの組合せはすべて同時にマッピングが解除され、ピンはグレー表示に戻ります。

### STM32F107x マイクロコントローラによるブロック･マッピングの例

PB5 に I2C1_SMBA 機能を割り当てると、STM32CubeMX によってピンとモードが次のように設定されます。

- I2C1_SCL 信号は PB6 ピン、I2C1_SDA 信号は PB7 ピンにそれぞれマッピングされます（図 338 を参照）。
- I2C1 ペリフェラル･モードは SMBus-Alert モードに設定されます。

図 338. ブロックのマッピング

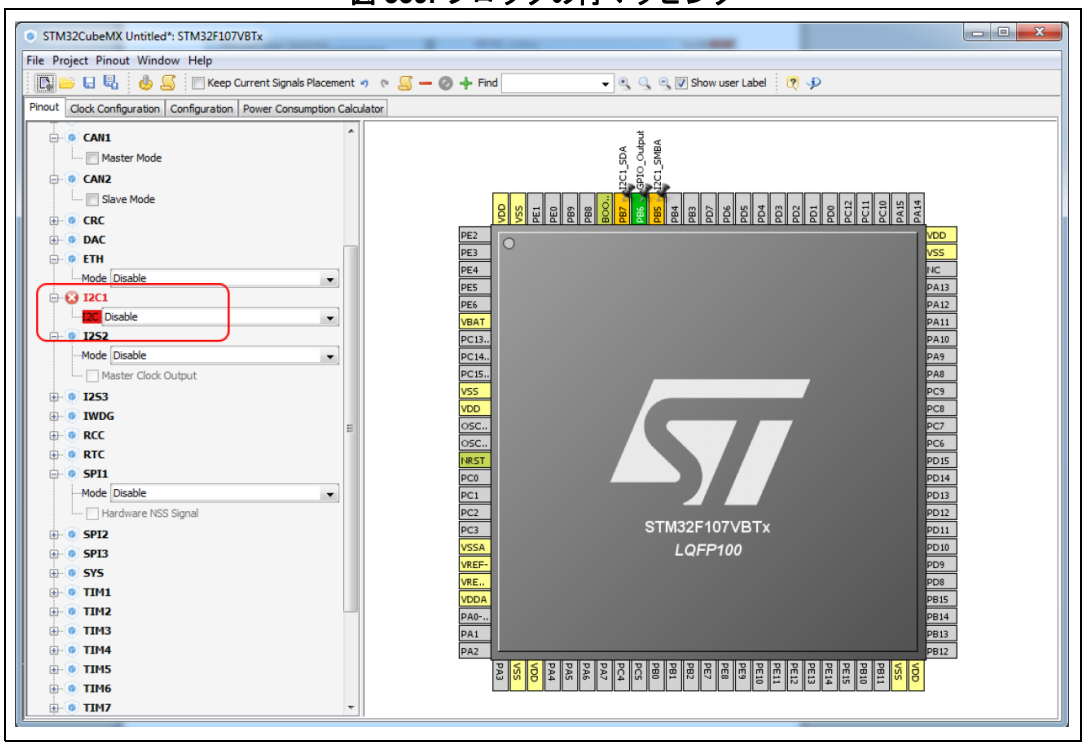



## STM32F107x マイクロコントローラによるブロック再マッピングの例

PB6 に GPIO_Output を割り当てると、STM32CubeMX によって I2C1 SMBus-Alert ペリフェラル·モードがペリフェラル·ツリー·ビューで自動的に無効になり、他の I2C1 ピン（PB5 および PB7）が次のように更新されます。

- 信号が固定されていないピンは、設定がリセットされます（ピンがグレー表示になります）。
- シグナルが固定されているピンは、ピンに割り当てられたペリフェラル·シグナルが保持されますが、ペリフェラル·モードと一致しなくなるのでピンのハイライトはオレンジ色になります（図 339 を参照）。

図 339. ブロックの再マッピング




I²C ペリフェラル·モードに替わるソリューションを STM32CubeMX で検出するには、I2C1 ピンの信号固定を解除し、ペリフェラル·ツリー·ビューから I2C1 モードを選択する必要があります（図 340 および図 341 を参照）。

図 340. ブロックの再マッピング - 例 1

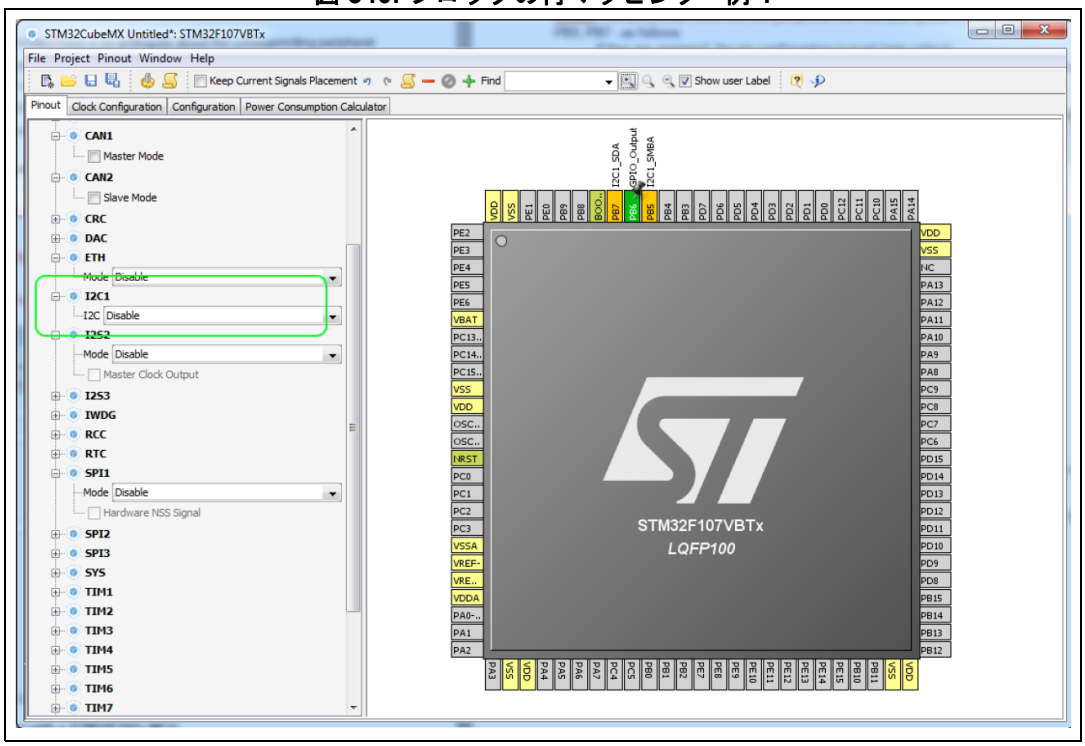
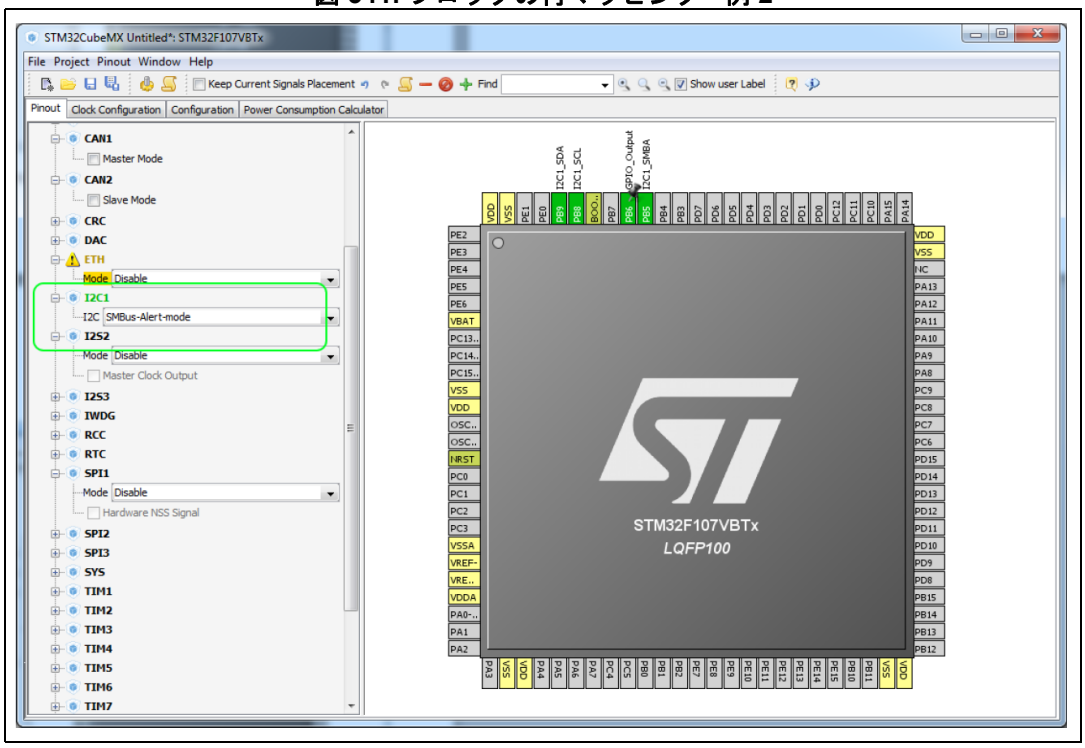



図 341. ブロックの再マッピング - 例 2



## A.2 ブロックの相互依存性

**[Pinout]** ビューでは、複数のピンの代替機能として同じ信号が表示されることがあります。一方、信号のマッピングは 1 回のみ可能です。

その結果、STM32F1 マイクロコントローラでは、同じペリフェラル･モードでピンの 2 つのブロックを同時に選択することはできません。いずれかのブロックでブロックとシグナルの組合わせを選択すると、他のブロックはクリアされます。

### STM32F107x マイクロコントローラの全二重マスタ･モードでの SPI のブロック再マッピングの例

ツリー･ビューで SPI1 全二重マスタ･モードを選択すると、対応する SPI 信号がデフォルトで PB3、PB4、および PB5 の各ピンに割り当てられます（図 342 を参照）。

現在 PB4 に割り当てられている SPI1_MISO 機能を PA6 に割り当てると、STM32CubeMX により、SPI1_MISO 機能から PB4 ピンがクリアされるほか、このブロックに設定されている他のピンもすべてクリアされます。それらに対応する SPI1 機能は、PB4 ピンと同じブロックにある関連のピンに移動します（図 343 を参照）。

（この操作を実行するには、Ctrl キーを押したままで PB4 をクリックし、PA6 の代替機能が青色で表示されたら、その信号を PA6 ピンにドラッグしてドロップします）

**図 342. ブロックの相互依存性 ー PB3、PB4、PB5に割り当てられた SPI シグナル**

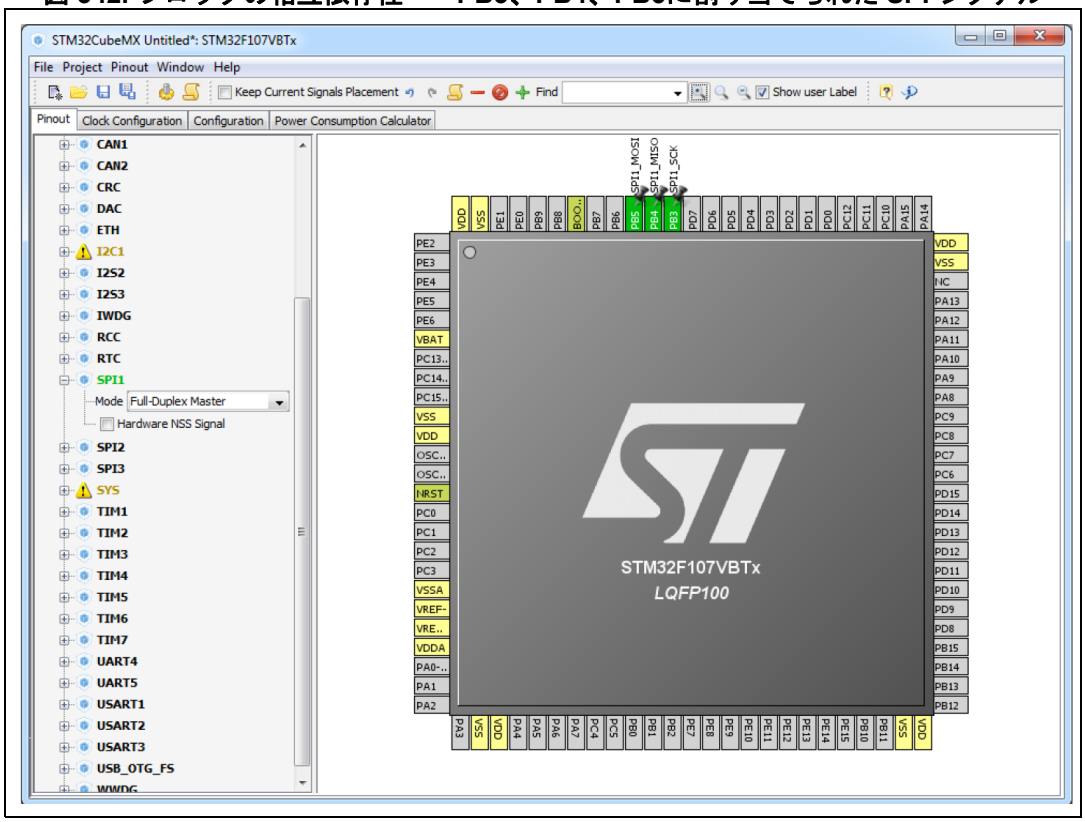
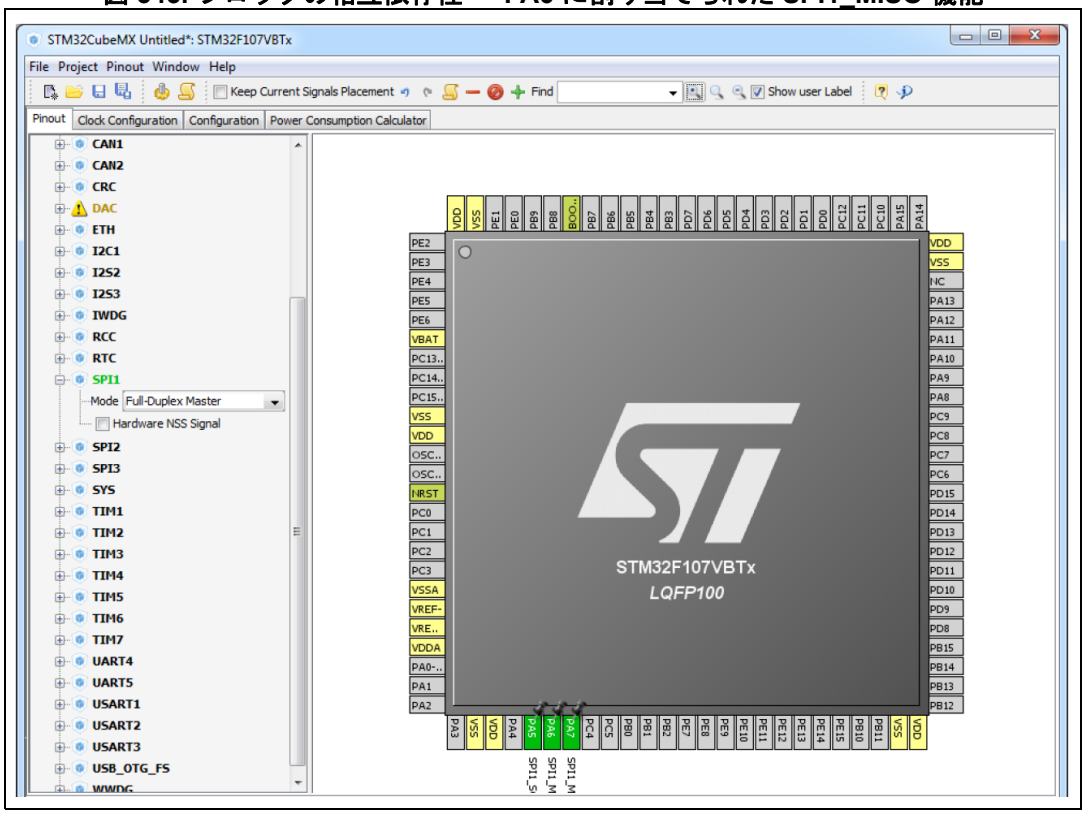


図 343. ブロックの相互依存性 ー PA6 に割り当てられた SPI1_MISO 機能



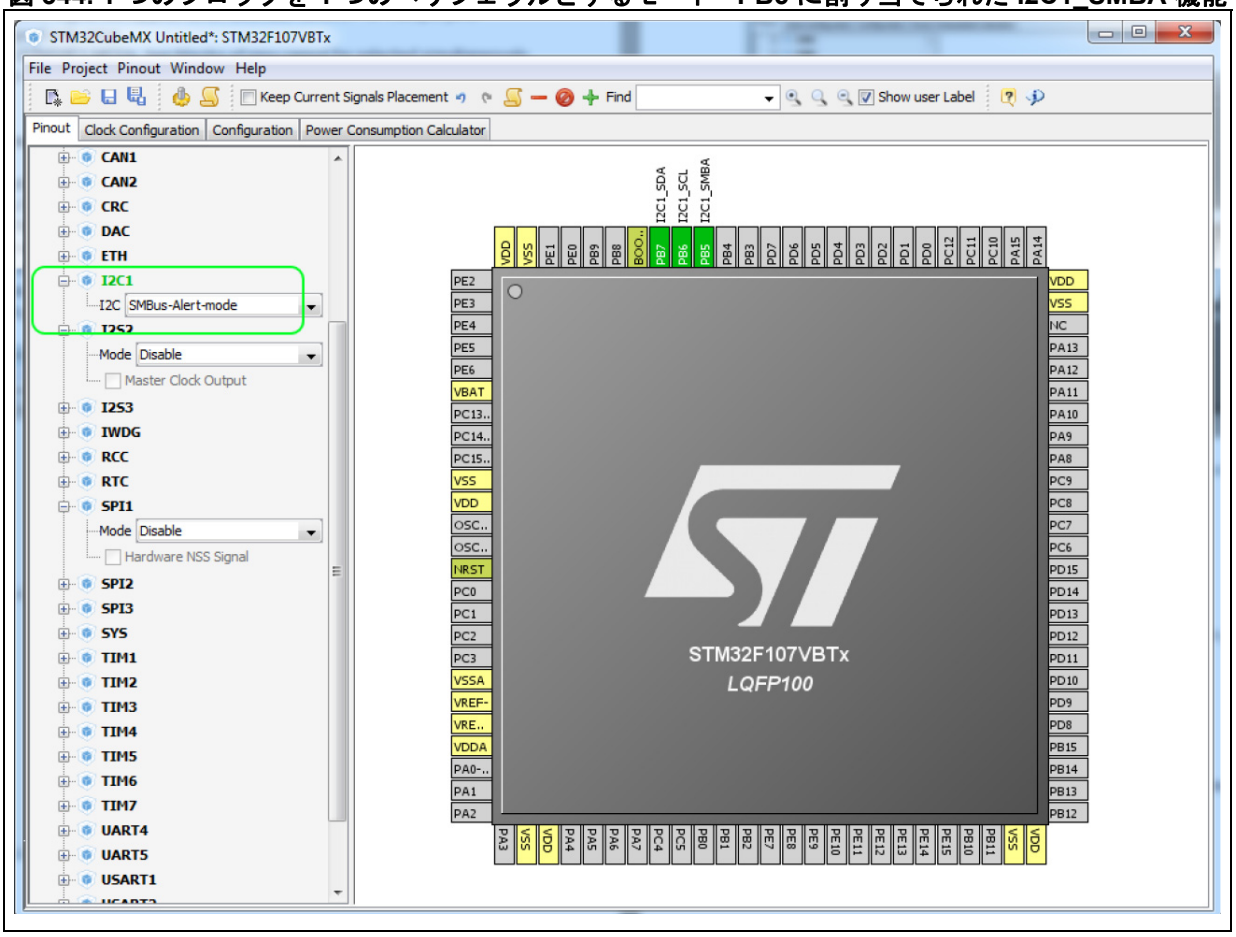
## A.3 1 つのブロックを 1 つのペリフェラルとするモード

[Pinout] ビューでピンのブロックをすべて設定すると（ピンが緑色で表示された状態）、それに関連したペリフェラル·モードがペリフェラル·ビューで自動的に設定されます。

### STM32F107x マイクロコントローラの例

PB5 に I2C1_SMBA 機能を割り当てると、I2C1 ペリフェラルが自動的に SMBus-Alert モードに設定されます（図 344 でペリフェラル·ツリーを参照）。

**図 344. 1 つのブロックを 1 つのペリフェラルとするモード - PB5 に割り当てられた I2C1_SMBA 機能**




## A.4 ブロックの再マッピング（STM32F10x のみ）

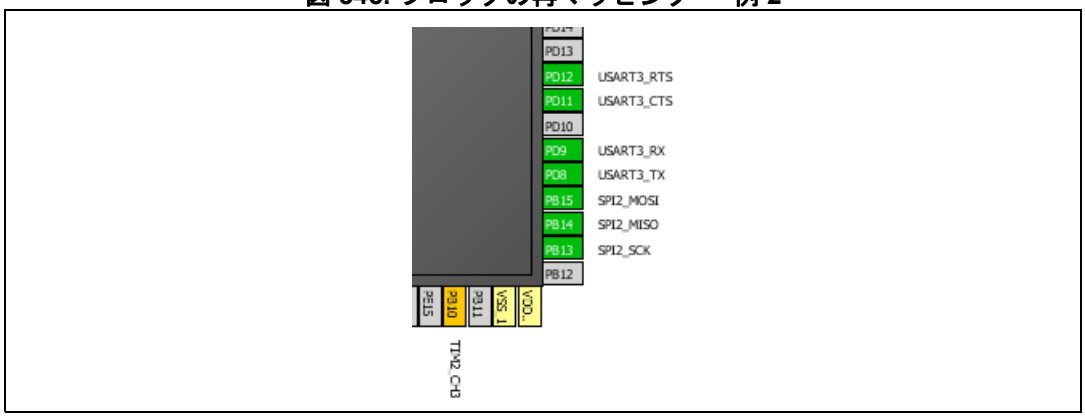
ペリフェラル·モードを設定するには、STM32CubeMX でピンのブロックを選択し、このブロックのピンに各モード信号を割り当てます。この目的で、モードをマッピングできる最初の空きブロックが検索されます。

ペリフェラル·モードを設定するときに、デフォルトのブロックにすでに使用されているピンが 1 本でもあると、STM32CubeMX によって代わりのブロックの検出が試みられます。そのようなブロックが見つからないと、別の順序で機能が選択されるか、Keep Current Signals Placement のチェックが外され、ソリューションが見つかるようにすべてのブロックが再マッピングされます。

### 例

USART3 のデフォルトのブロックである PB14 に SPI2_MISO 機能がすでに割り当てられているので、STM32CubeMX によって（PD8-PD9-PD11-PD12）ブロックに USART3 の hardware-flow-control モードが再マッピングされます（図 345 を参照）。

**図 345. ブロックの再マッピング ー 例 2**



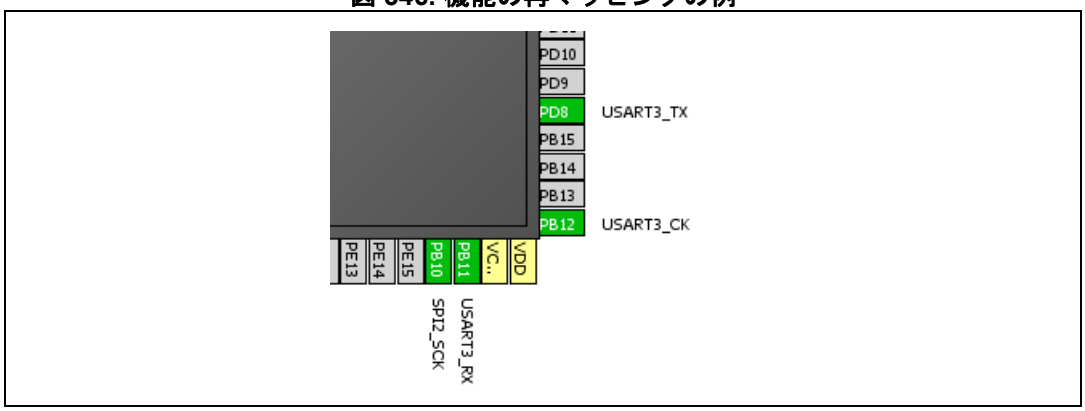

## A.5 機能の再マッピング

ペリフェラル·モードを設定するために、STM32CubeMX ではモードの各信号をピンに割り当てます。この目的で、信号をマッピングできる最初の空きピンが検索されます。

### STM32F415x を使用した例

USART3 を同期モードに設定すると、USART3_TX 信号のデフォルトの PB10 ピンがすでに SPI で使用されたことが STM32CubeMX で検出されます。その結果、機能が PD8 に再マッピングされます（図 346 を参照）。

**図 346. 機能の再マッピングの例**



## A.6 ブロックのシフト（STM32F10x で［Keep Current Signals Placement］がチェックされていない場合のみ）

ブロックをマッピングできず、使用されていない代替のソリューションがない場合、STM32CubeMX では、共有ピンの影響を受けるすべてのペリフェラル·モードを再マッピングすることでピンを解放しようとします。

### 例

［Keep Current Signals Placement］を有効にした状態で、USART3 の同期モードを先に設定した場合は、非同期のデフォルト·ブロック（PB10 ~ PB11）がマッピングされ、Ethernet は赤色表示となって利用できなくなります（図 347 を参照）。

Keep Current Signals Placement をチェックしないと、STM32CubeMX ではブロックをシフトし、Ethernet MII モードで使用できるようにブロックを解放できます（ 図 348を参照。）

**図 347. ブロックのシフトを適用しない場合**

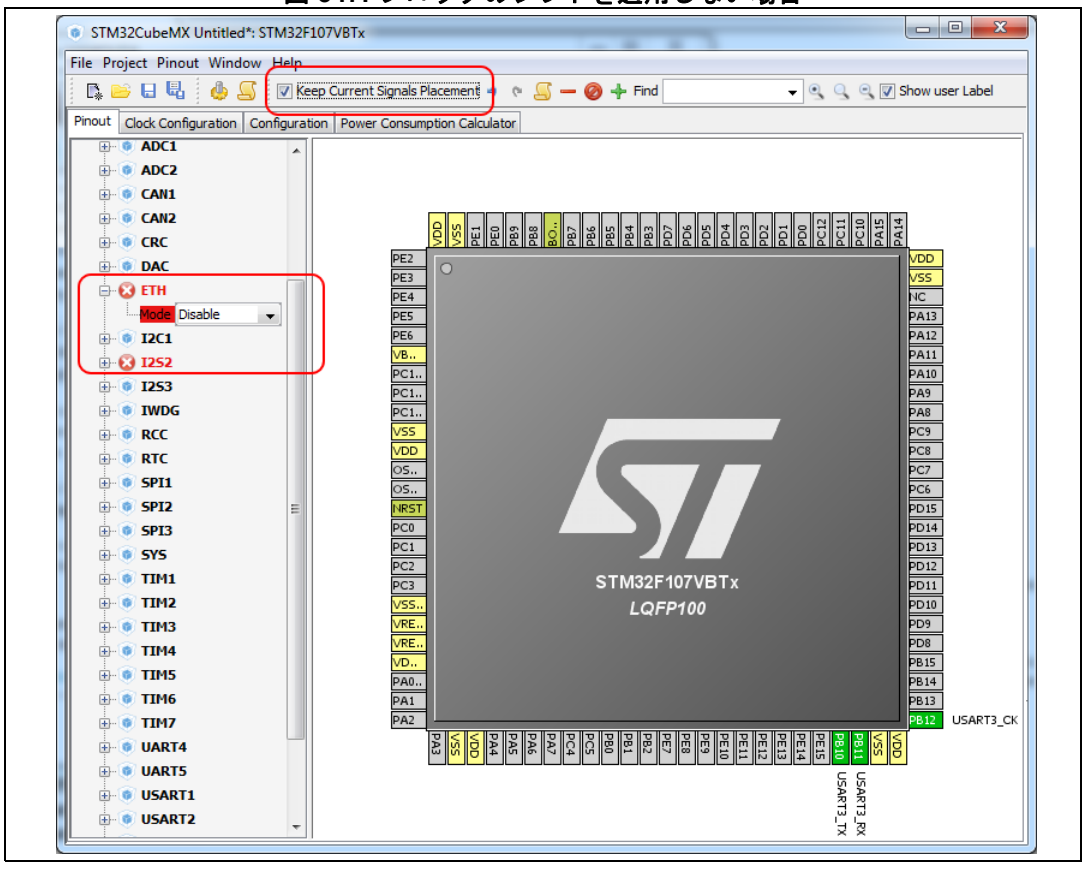
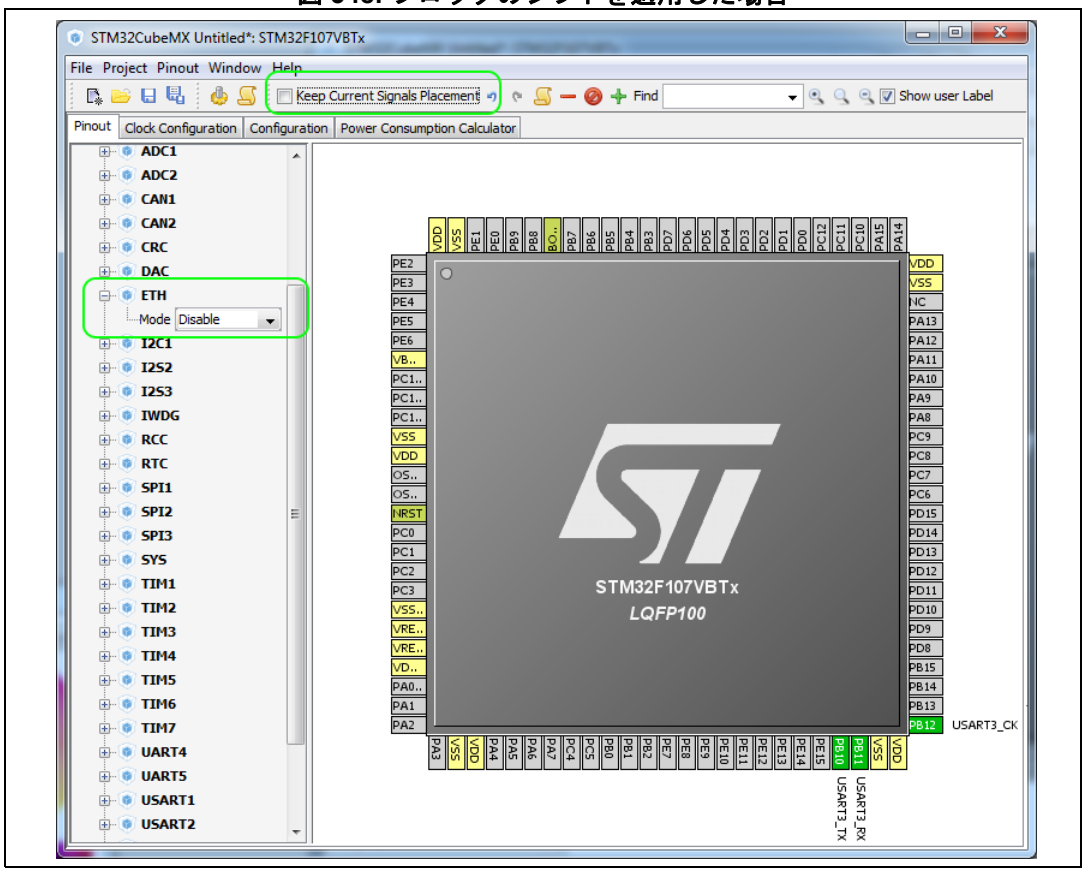


図 348. ブロックのシフトを適用した場合




## A.7 ペリフェラル·モードの設定およびクリア

[Peripherals] パネルと **[Pinout]** ビューはリンクしていて、ペリフェラル·モードを設定またはクリアすると、それに対応するピン機能が設定またはクリアされます。

## A.8 機能の個別マッピング

STM32CubeMX では、ある機能（ペリフェラル·モードの設定なし）に手動ですでに割り当てられているピンを使用する必要があると、Keep Current Signals Placement がチェックされていない場合にのみ、その機能を別のピンに移動し、機能にピンが固定されていない状態にすることができます（ピンのアイコンが表示されない状態）。

## A.9 GPIO 信号マッピング

I/O 信号ピン（GPIO_Input、GPIO_Output、GPIO_Analog）は、**[Pinout]** ビューから手動で割り当てるか、**[Pinout]** メニューから自動的に割り当てることができます。このようなピンは、他の信号に自動的には割り当てることができなくなります。STM32CubeMX では、このピンの I/O 信号を他のピンに移動しないので、信号の自動配置では、このピンが考慮されません。

手動操作であれば、このピンを他の信号に割り当てることも、状態をリセットすることもできます。

# 付録 B STM32CubeMXC コード生成の設計上の選択肢と制限事項

## B.1 STM32CubeMX で生成した C コードとユーザ･セクション

STM32CubeMX で生成した C コードには、次に示すようなユーザ･セクションが用意されます。これらを使用することで、ユーザ C コードの挿入や、次回の C コード生成に備えたユーザ C コードの保持が可能です。

ユーザ･セクションは削除も名前変更もできません。STM32CubeMX で定義したユーザ･セクションのみが保持されます。ユーザが作成したセクションは、次回の C コード生成で無視され、失われます。

```
 /* USER CODE BEGIN 0 */
(..)
/* USER CODE END 0 */
```

注： **STM32CubeMX では、いくつかのユーザ･セクションに C コードを生成できます。このセクションで廃止扱いになる部分をクリーンアップするかどうかは、ユーザの判断となります。たとえば、main 関数で while(1) ループを次のようにユーザ･セクション内部に配置するとします。**

```
/* Infinite loop */
  /* USER CODE BEGIN WHILE */
  while (1)
  {
  /* USER CODE END WHILE */

  /* USER CODE BEGIN 3 */
  }
/* USER CODE END 3 */
```

## B.2 ペリフェラルを初期化するための STM32CubeMX の設計上の選択肢

STM32CubeMX では、次のように MX_ プレフィックスで容易に特定できる、ペリフェラルの _Init 関数が生成されます。

```
static void MX_GPIO_Init(void);
static void MX_<Peripheral Instance Name>_Init(void);
static void MX_I2S2_Init(void);
```

ユーザが選択したペリフェラルのインスタンスごとに MX_<ペリフェラルのインスタンス名>_Init 関数が存在します（たとえば、MX_I2S2_Init）。この関数は、HAL ドライバの初期化（HAL_I2S_Init など）とこの関数の実際の呼び出しに必要となる該当のハンドル構造（たとえば、I2S の 2 番目のインスタンスであれば &hi2s2）を初期化します。

```
void MX_I2S2_Init(void)
{
 hi2s2.Instance = SPI2;
  hi2s2.Init.Mode = I2S_MODE_MASTER_TX;
  hi2s2.Init.Standard = I2S_STANDARD_PHILLIPS;
  hi2s2.Init.DataFormat = I2S_DATAFORMAT_16B;
```

```
  hi2s2.Init.MCLKOutput = I2S_MCLKOUTPUT_DISABLE;
  hi2s2.Init.AudioFreq = I2S_AUDIOFREQ_192K;
  hi2s2.Init.CPOL = I2S_CPOL_LOW;
  hi2s2.Init.ClockSource = I2S_CLOCK_PLL;
  hi2s2.Init.FullDuplexMode = I2S_FULLDUPLEXMODE_ENABLE;
  HAL_I2S_Init(&hi2s2);
}
```

デフォルトでは、ペリフェラルの初期化は main.c で実行されます。ミドルウェア･モードで使用するペリフェラルは、そのミドルウェアの対応する .c ファイルで初期化できます。

カスタマイズした HAL_<ペリフェラル名>_MspInit()関数は、選択したペリフェラルの低レベル･ハードウェア（GPIO、CLOCK）を設定する stm32f4xx_hal_msp.c ファイルに作成されます。

## B.3 ミドルウェアを初期化するための STM32CubeMX の設計上の選択肢と制限事項

### B.3.1 概要

STM32CubeMX では、ミドルウェア･スタックのネイティブ･ファイルに C のユーザ･コードを挿入することはできませんが、用途によっては LwIP などのスタックでこのようなコードの挿入が必要になることがあります。

STM32CubeMX では、次のように MX_ プレフィックスで容易に特定できる、ミドルウェアの _Init 関数が生成されます。

```
MX_LWIP_Init(); // lwip.h ファイルで定義
MX_USB_HOST_Init(); // usb_host.h ファイルで定義
MX_FATFS_Init(); // fatfs.h ファイルで定義
```

なお、次の例外もあります。

- .c ファイルと .h ファイルのペアで Init 関数を生成することをユーザが［Project Settings］ウィンドウで選択していない限り、FreeRTOS については Init 関数が生成されません。代わりに、main.c ファイルで StartDefaultTask 関数が定義され、CMSIS-RTOS のネイティブ関数（osKernelStart）が main 関数で呼び出されます。
- FreeRTOS が有効であれば、使用している他のミドルウェアの Init 関数は、main.c ファイルの StartDefaultTask 関数から呼び出されます。

  例：

```
void StartDefaultTask(void const * argument)
{
   /* init code for FATFS */
   MX_FATFS_Init();
   /* init code for LWIP */
   MX_LWIP_Init();
  /* init code for USB_HOST */
  MX_USB_HOST_Init();
  /* USER CODE BEGIN 5 */
  /* Infinite loop */
  for(;;)
```

```
  {
    osDelay(1);
  }
  /* USER CODE END 5 */
}
```

### B.3.2 USB ホスト

USB ペリフェラルは、usbh_conf.c ファイルのミドルウェア初期化 C コードで初期化され、USB スタックは、usb_host.c ファイルで初期化されます。

USB ホストのミドルウェアを使用する場合、生成した usb_host.c ファイルへの USBH_UserProcess コールバック関数の実装はユーザ側で対応する必要があります。

STM32CubeMX のユーザ･インタフェースでは、クラスを 1 つだけ登録できるほか、アプリケーションで複数のクラスを動的に切り替える必要がある場合はそのすべてのクラスを登録することもできます。

### B.3.3 USB デバイス

USB ペリフェラルは、usbd_conf.c ファイルのミドルウェア初期化 C コードで初期化され、USB スタックは、usb_device.c ファイルで初期化されます。

USB VID、PID、および文字列の標準的な記述子は STM32CubeMX のユーザ･インタフェースで設定し、usbd_desc.c 生成ファイルで使用できます。他の標準的な記述子（設定、インタフェース）は、USB 複合デバイスのサポートを防止する同じファイルにハードコードされています。

USB デバイスのミドルウェアを使用する場合、すべてのデバイス･クラスについて usbd_<クラス名>_if.c クラス･インタフェース･ファイルに該当の関数を実装する作業はユーザ側で対応する必要があります。

USB MTP クラスと CCID クラスはサポートされていません。

### B.3.4 FatFs

FatFs は、小規模な組込みシステムに適した汎用的な FAT/exFAT ファイル･システム･ソリューションです。

FatFs の設定は、ffconf.h 生成ファイルにあります。

FatFs SD カード･モードでの SDIO ペリフェラルの初期化、および FatFs 外部 SDRAM モードと FatFs 外部 SRAM モードでの FMC ペリフェラルの初期化は、main.c ファイルに保持されます。

ファイルによっては、ユーザのボード仕様に適合するようにユーザ側で修正が必要です。STM32Cube 組込みソフトウェア･パッケージの BSP が使用できる例を次に挙げます。

- FatFs SD カード･モードを使用する場合の bsp_driver_sd.c/.h 生成ファイル
- FatFs 外部 SRAM モードを使用する場合の bsp_driver_sram.c/.h 生成ファイル
- FatFs 外部 SDRAM モードを使用する場合の bsp_driver_sdram.c/.h 生成ファイル

複数ドライブの FatFs がサポートされているので、アプリケーションで複数の論理ドライブを使用できます（外部 SDRAM、外部 SRAM、SD カード、USB ディスク、ユーザ定義）。一方、1 つの論理ドライブの複数インスタンスを使用することはできません（USB ホストの 2 つのインスタンスや複数の RAM ディスクを使用する FatFs など）。

NOR Flash･メモリと NAND Flash･メモリはサポートされていません。この場合は、FatFs のユーザ定義モードを選択し、生成した user_diskio.c ドライバ･ファイルを更新して、ミドルウェアと選択したペリフェラルの間にインタフェースを実装する必要があります。

### B.3.5 FreeRTOS

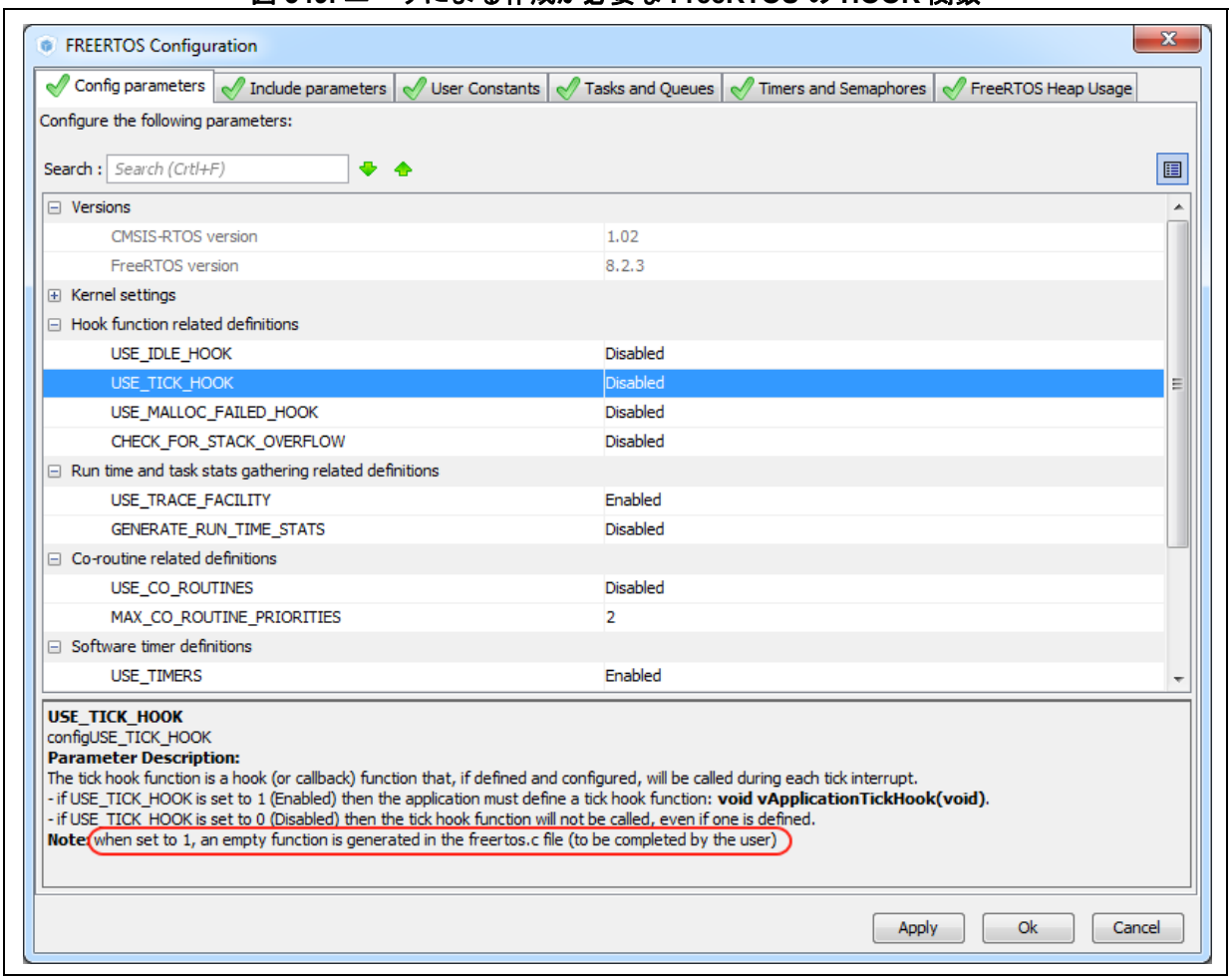
FreeRTOS は、マイクロコントローラに適した無償のリアルタイム組込みオペレーティング･システムです。

FreeRTOS の設定は、FreeRTOSConfig.h 生成ファイルにあります。

FreeRTOS を有効にすると、選択した他のすべてのミドルウェア･モード（LwIP、FatFs、USB など）は、main.c ファイルの同じ FreeRTOS スレッドで初期化されます。

GENERATE_RUN_TIME_STATS、CHECK_FOR_STACK_OVERFLOW、USE_IDLE_HOOK、USE_TICK_HOOK、および USE_MALLOC_FAILED_HOOK の各パラメータをアクティブにすると、空の関数を記述した freertos.c ファイルが STM32CubeMX によって生成されます。この空の関数は、ユーザが実装する必要があります。この点は、ツールチップでハイライトされます（図 349 を参照）。

**図 349. ユーザによる作成が必要な FreeRTOS の HOOK 関数**



### B.3.6 LwIP

LwIP は、TCP/IP プロトコル·スイートの小規模な独自実装です。RAM の使用量が少ないので、RAM の空き容量が数十 KB の組込みシステムに適しています。

LwIP の初期化関数は lwip.c で定義され、LwIP の設定は lwipopts.h 生成ファイルにあります。

STM32CubeMX では、Ethernet での LwIP のみがサポートされています。Ethernet ペリフェラルは、ミドルウェアの初期化 C コードで初期化されます。

STM32CubeMX では、スタックのネイティブ·ファイルにユーザの C コードを挿入することはできません。一方、LwIP の用途によっては、スタックのネイティブ·ファイル（cc.h, mib2.c など）の修正が必要になります。ユーザによる修正は、次回の STM32CubeMX の生成で失われるので、バックアップを作成しておく必要があります。

LwIP のリリース 1.5 より、STM32CubeMX の LwIP では IPv6 をサポートしています（図 351 を参照）。

静的 IP アドレスを設定するには、DHCP を無効にする必要があります。

**図 350. LwIP 1.4.1 の設定**

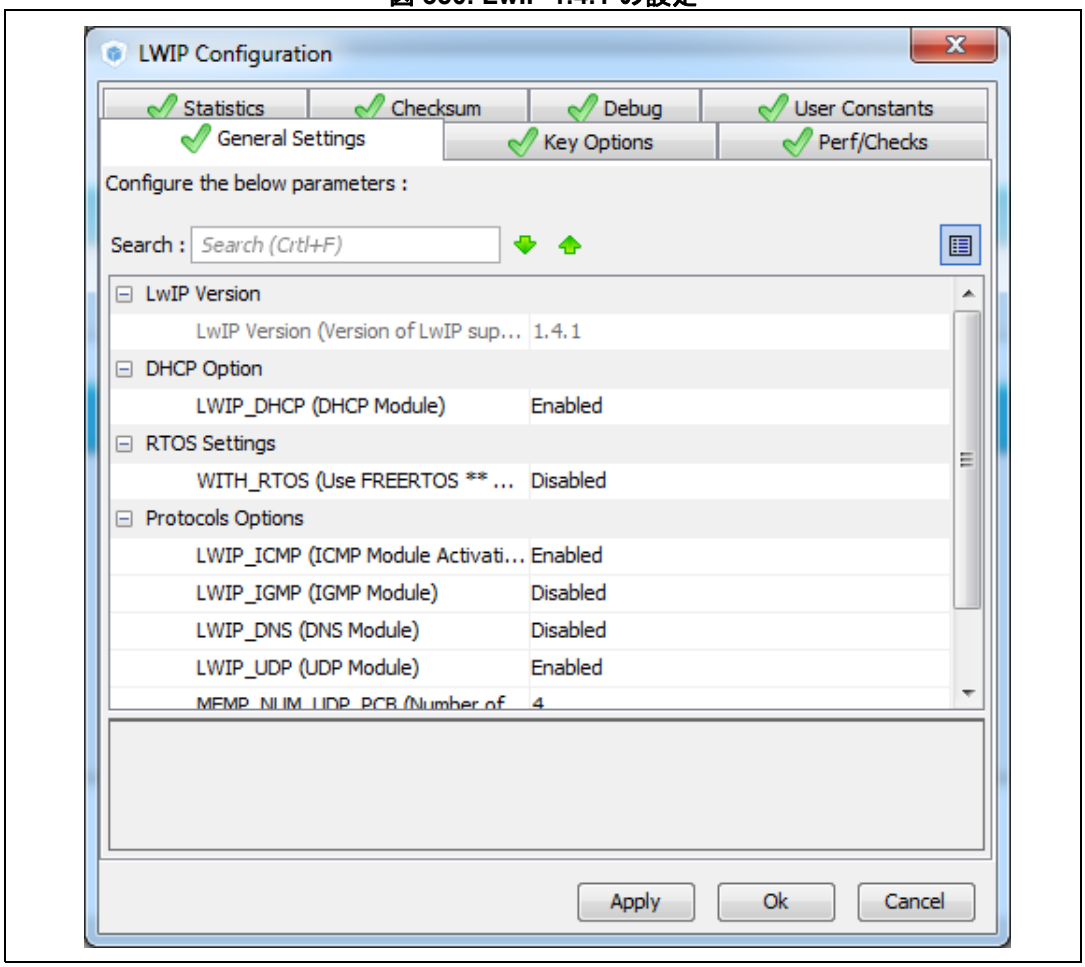
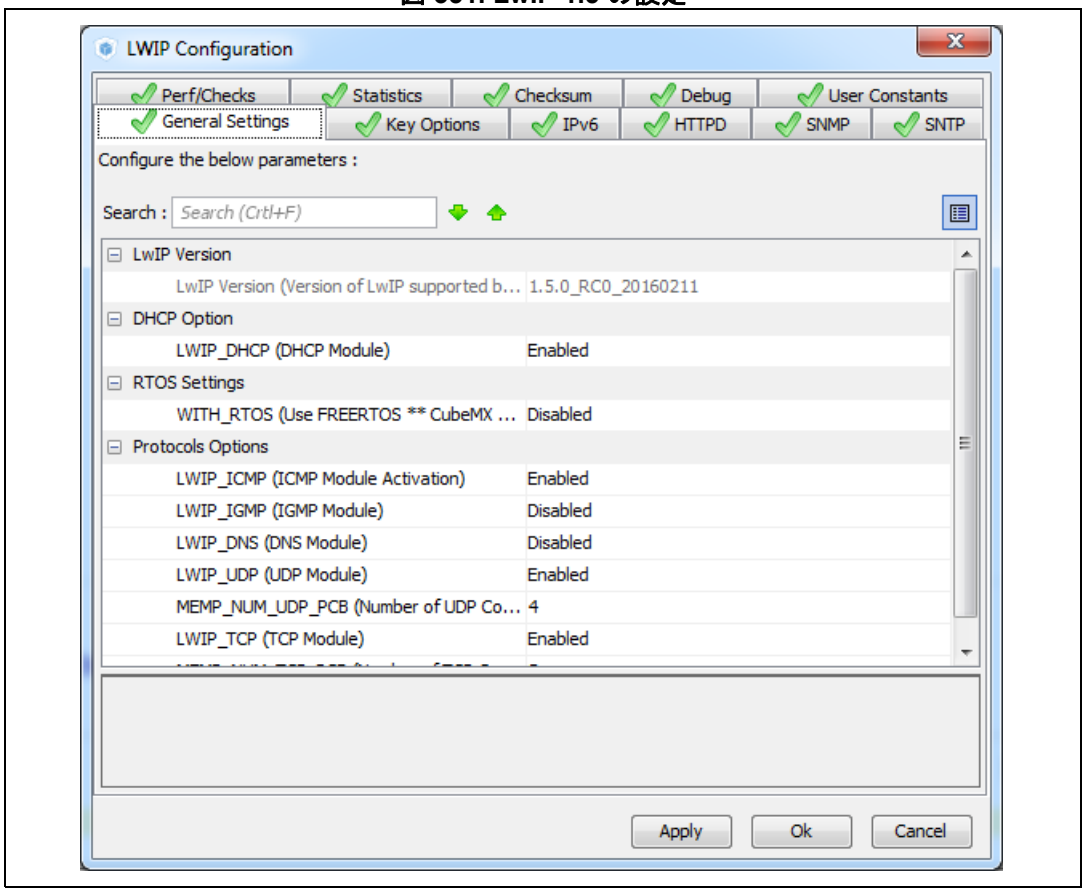


図 351. LwIP 1.5 の設定




特定のパラメータ（デフォルトでは無効）を有効にしておくと、STM32CubeMX で生成した C コードにコンパイル･エラーが報告されます。スタックのパッチ（インターネットからダウンロード）またはユーザ C コードで問題を修正する必要があります。次の各パラメータによってエラーが報告されます。

- MEM_USE_POOLS : lwipopts.h または cc.h（スタック･ファイル）に追加するユーザ C コード。
- PPP_SUPPORT、PPPOE_SUPPORT：必須のユーザ C コード。
- MEMP_OVERFLOW_CHECK に 0 よりも大きい値を指定した場合の MEMP_SEPARATE_POOLS：スタックの必須パッチ
- 有効にした MEM_LIBC_MALLOC および MEM_LIBC_RTOS：スタックの必須パッチ
- LWIP_EVENT_API：スタックの必須パッチ

STM32CubeMX では、netconn API と sockets API で LwIP を使用するために FreeRTOS を有効にする必要があります。これらの API ではスレッドを使用する必要があることから、必然的にオペレーティング･システムを使用する必要があります。FreeRTOS が機能していない場合、使用できる API は、LwIP のイベントで動作する Raw API のみとなります。

### B.3.7 Libjpeg

Libjpeg は、JPEG ファイルを読み書きするために広く使用されている C ライブラリです。STM32CubeF7、STM32CubeH7、STM32CubeF2、および STM32CubeF4 の各組込みソフトウェア・パッケージに付属して提供されています。

STM32CubeMX では次のファイルが生成されます。これらのファイルの内容は、STM32CubeMX のユーザ・インタフェースで設定できます。

- **libjpeg.c/.h**

  MX_LIBJPEG_Init() 初期化関数が libjpeg.c ファイルに生成されます。このファイルは空です。アプリケーションで必要なコードと libjpeg 関数の呼び出しは、ユーザがユーザ・セクションに入力する必要があります。

- **jdata_conf.c**

  このファイルは、データ・ストリーム管理タイプとして FatFs を選択している場合にのみ生成されます。

- **jdata_conf.h**

  このファイルの内容は、選択したデータ・ストリーム管理タイプに応じて調整されます。

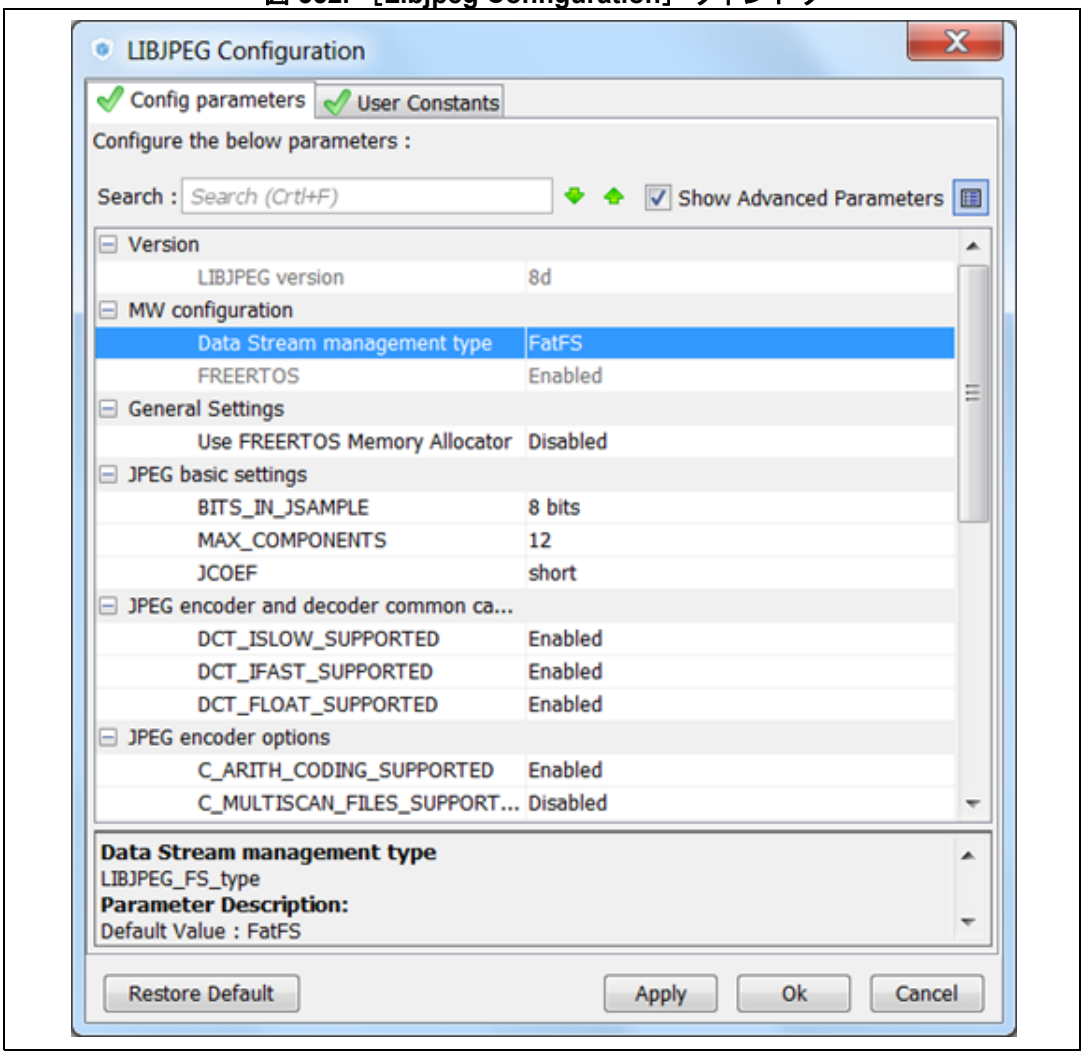
- **jconfig.h**

  このファイルは STM32CubeMX で生成されますが、設定はできません。

- **jmorecfg.h**

  このファイルに記述された define ステートメントの一部は、STM32CubeMX の［Libjpeg Configuration］メニューで変更できます。

図 352. ［Libjpeg Configuration］ウィンドウ




## B.3.8 Mbed TLS

Mbed TLS は、組込み製品に暗号機能を追加する C ライブラリです。このライブラリは、SSL（Secure Socket Layer）プロトコルと TLS（Transport Layer Security）プロトコルを扱います。これらのプロトコルは、安全性が低いネットワークで接続された 2 者の間に、暗号化と認証を導入した安全な接続を実現します。Mbed TLS は、API が直感的であることと作成するコードの量が最小限ですむことを特徴としています。詳細については https://tls.mbed.org/ を参照してください。

Mbed TLS は、STM32CubeF2、STM32CubeF4、STM32CubeF7、および STM32CubeH7 の各組込みソフトウェア･パッケージに付属して提供されています。

Mbed TLS は LwIP スタックがなくても動作します（図 353 : LwIP を使用しない場合の Mbed TLSを参照）。

LwIP を使用する場合は、FreeRTOS も有効にする必要があります（ 図 354：LwIP と FreeRTOS を使用した場合の Mbed TLSを参照）。

STM32CubeMX では、次のファイルが生成されます。これらのファイルの内容は、STM32CubeMX のユーザ·インタフェースまたはコード自体のユーザ·セクションを使用してユーザ側で変更できます（図 355：［MBED TLS Configuration］ウィンドウを参照）。

- mbedtls_config.h
- mbedtls.h
- net_sockets.c（LwIP が有効な場合にのみ生成）
- mbedtls.c

**図 353. LwIP を使用しない場合の Mbed TLS**

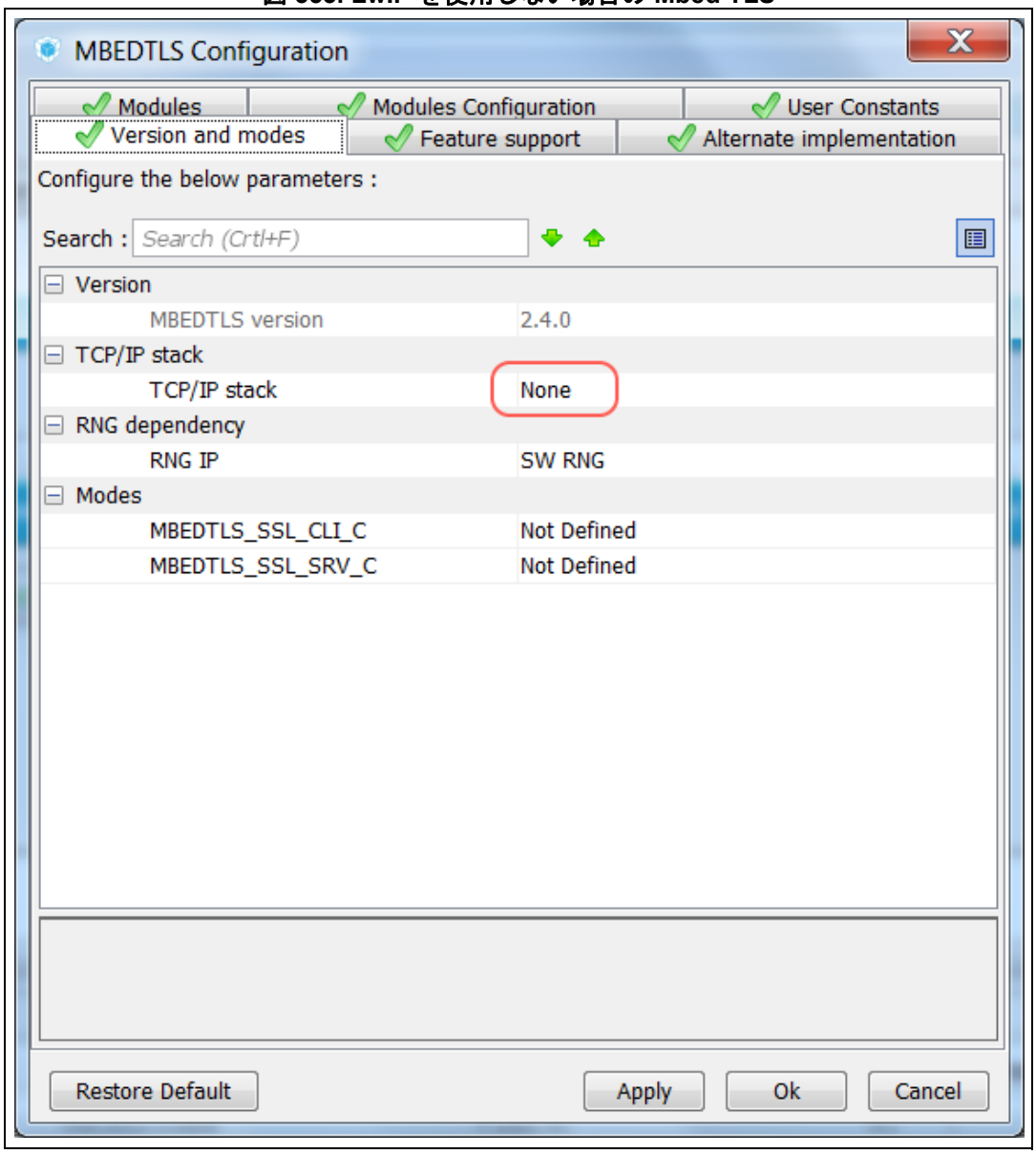


図 354. LwIP と FreeRTOS を使用した場合の Mbed TLS

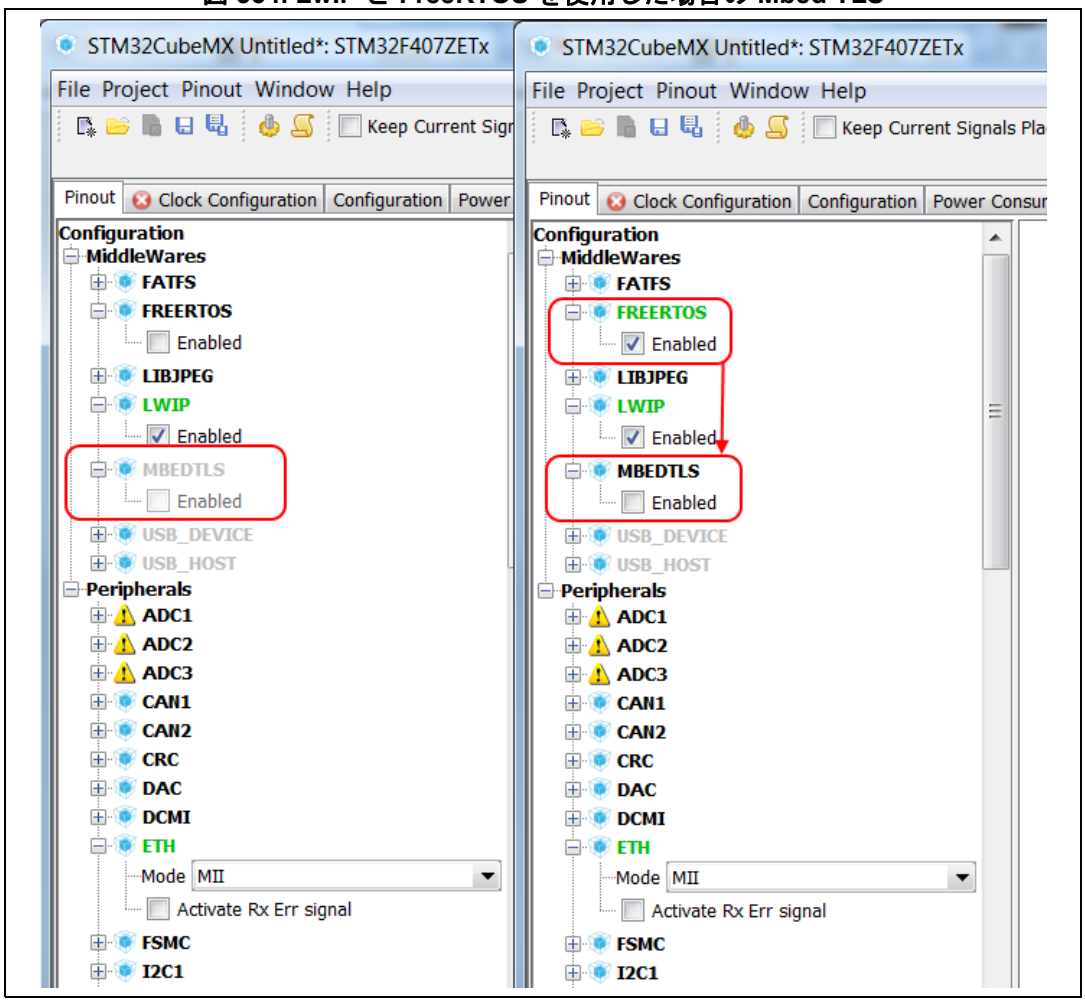
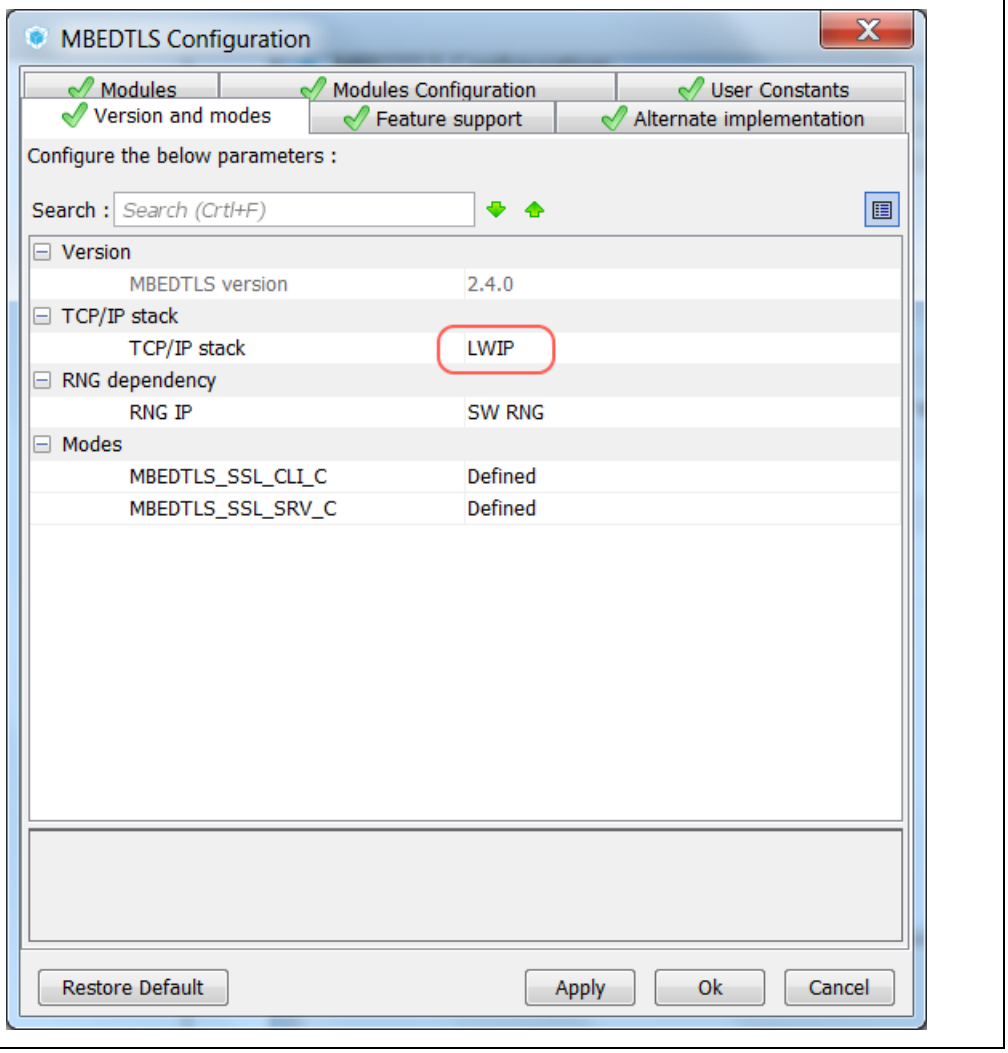


図 355. ［MBED TLS Configuration］ウィンドウ




## B.3.9 TouchSensing

STM32 TouchSensing ライブラリは、従来の電気機械式スイッチを STM32 マイクロコントローラによる容量センサに置き換えることにより、高度なヒューマン･インタフェースを作成できるようにする C ライブラリです。

このライブラリでは、タッチセンシング方式のペリフェラルをマイクロコントローラ上に設定する必要があります。

STM32CubeMX では次のファイルが生成されます。これらのファイルの内容は、STM32CubeMX のユーザ･インタフェースまたはコード自体のユーザ･セクションを使用してユーザ側で変更できます（図 356 : TouchSensing ペリフェラルの有効化、図 357 : タッチセンシング方式センサの選択パネル、および図 358 :［TOUCHSENSING Configuration］パネルを参照）。

- touchsensing.c/.h
- tsl_user.c/.h
- tsl_conf.h

図 356. TouchSensing ペリフェラルの有効化

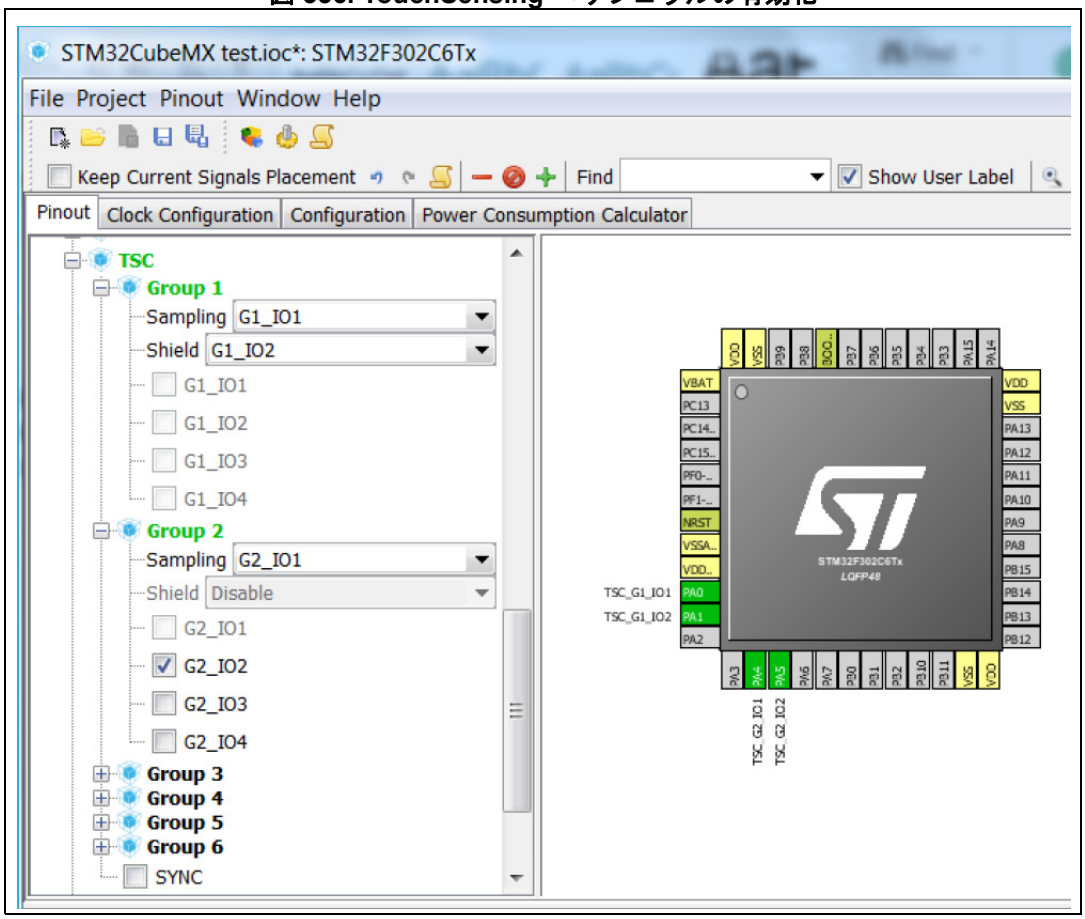


図 357. タッチセンシング方式センサの選択パネル

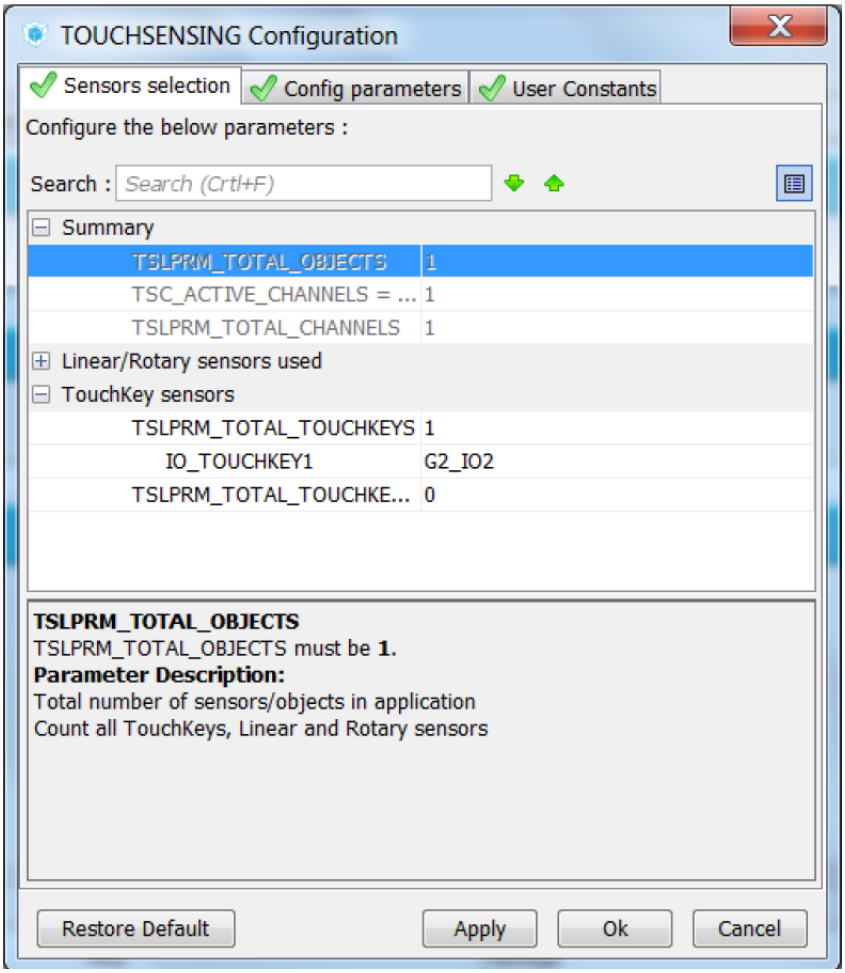
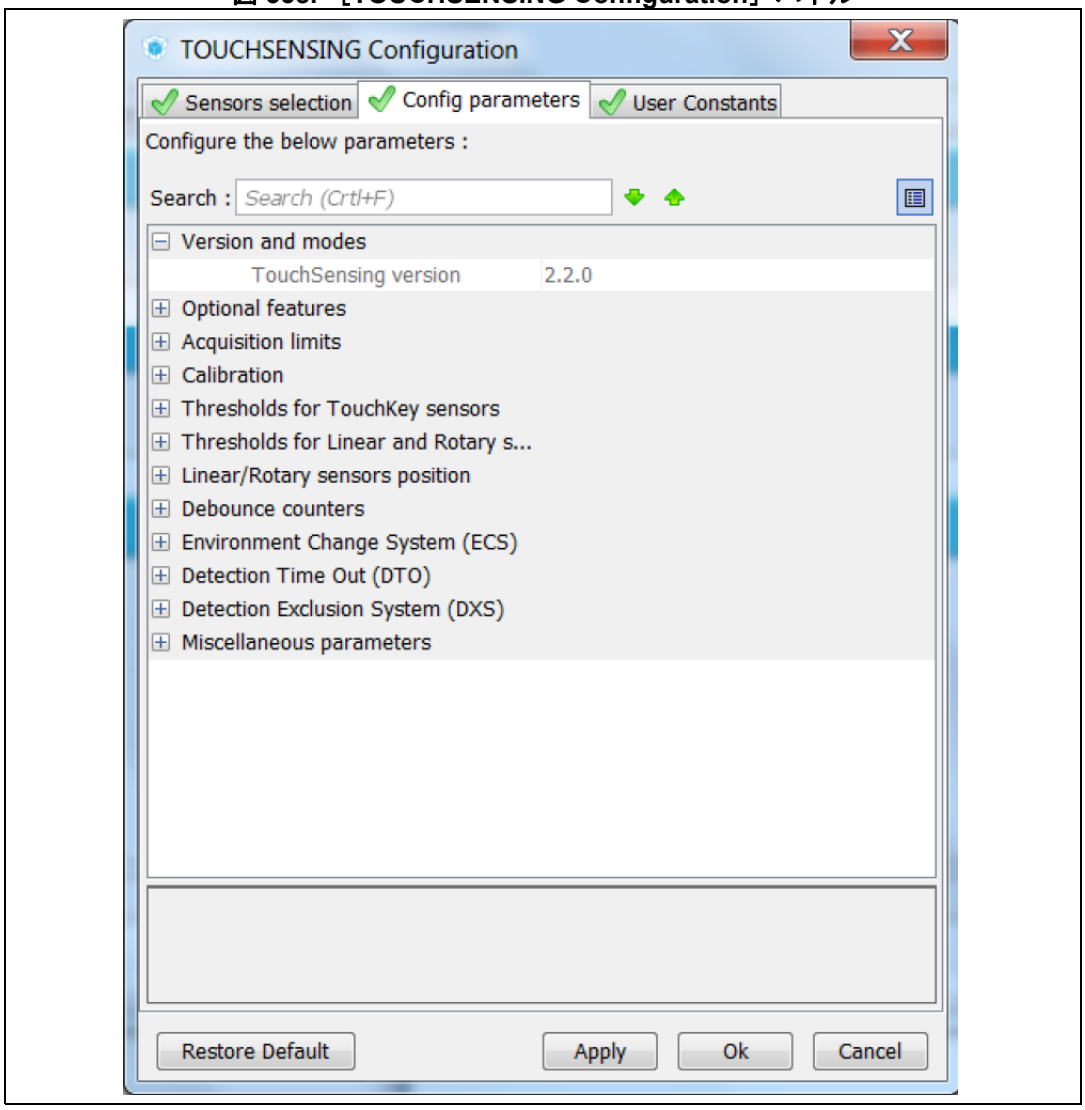


図 358. ［TOUCHSENSING Configuration］パネル




## B.3.10 PDM2PCM

PDM2PCM ライブラリは、パルス密度変調（PDM）データ出力を 16 bit のパルス符号変調（PCM）形式に変換するための C ライブラリです。このライブラリでは、CRC ペリフェラルを有効にする必要があります。

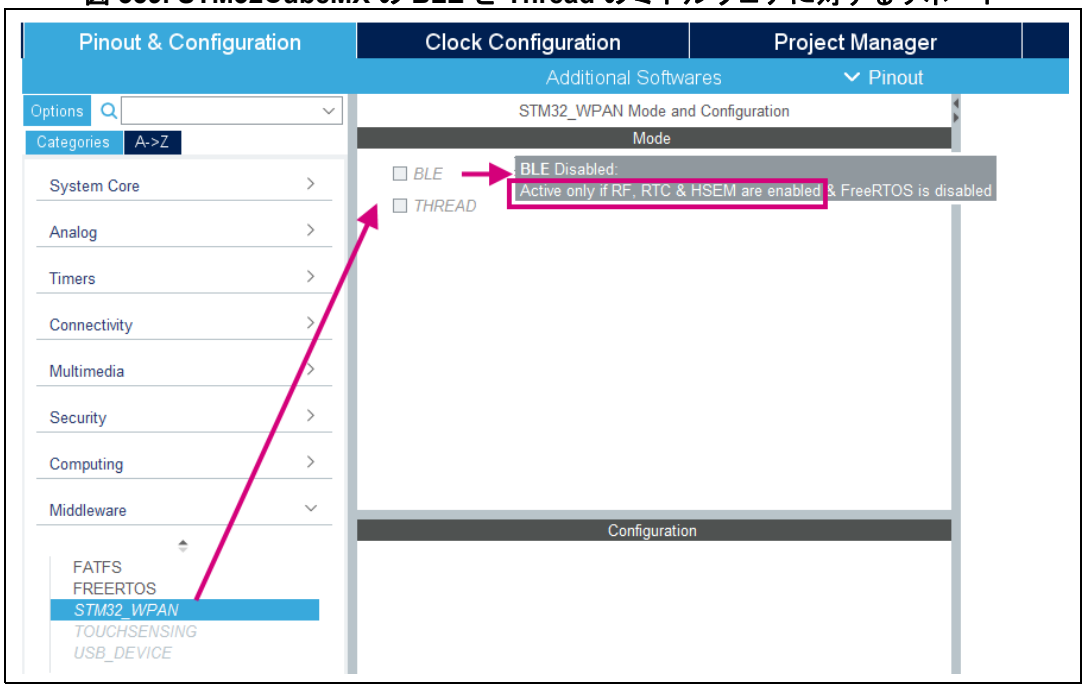
STM32CubeMX では次のファイルが生成されます。このファイルの内容は、STM32CubeMX のユーザ･インタフェースまたはコード自体のユーザ･セクションを使用してユーザ側で変更できます。

- pdm2pcm.h/.c

### B.3.11 STM32WPAN BLE/THREAD（STM32WB シリーズのみ）

STM32WPAN の BLE と Thread のミドルウェアを STM32CubeMX で使用できるようになりました。

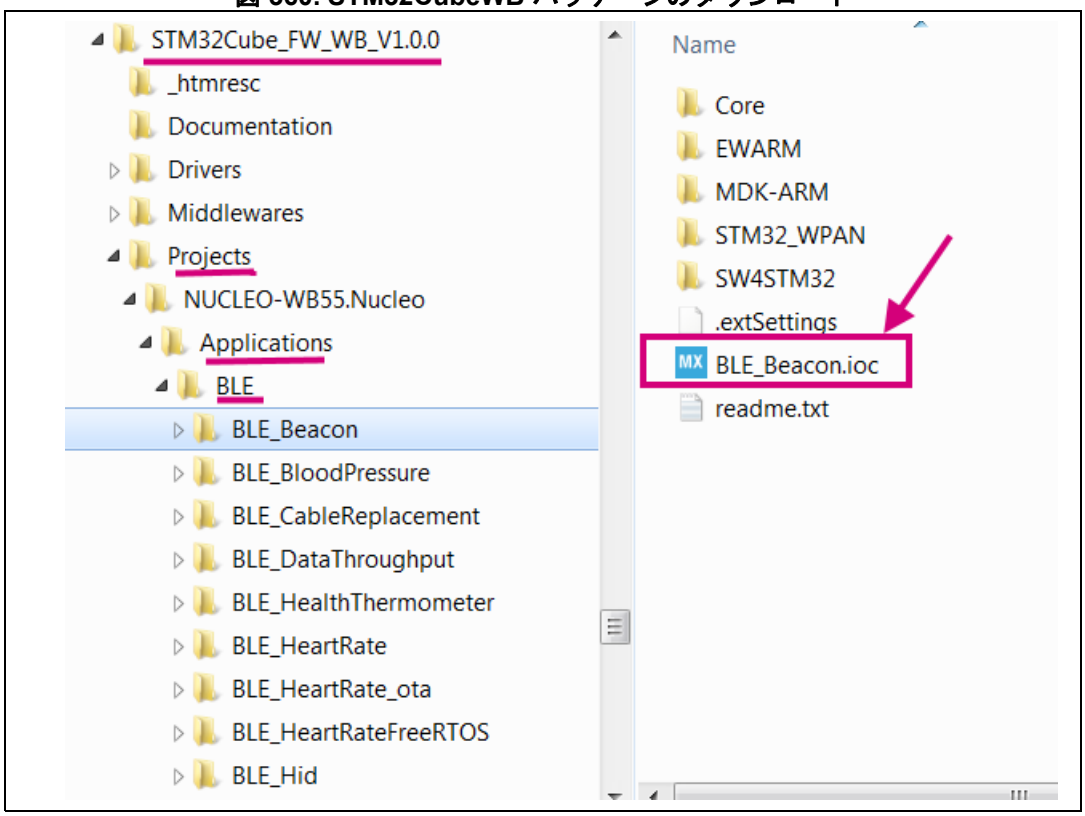
**図 359. STM32CubeMX の BLE と Thread のミドルウェアに対するサポート**




特定のプロジェクトでは両者を同時に使用することはできません。また FreeRTOS を使用した設定には未対応です。

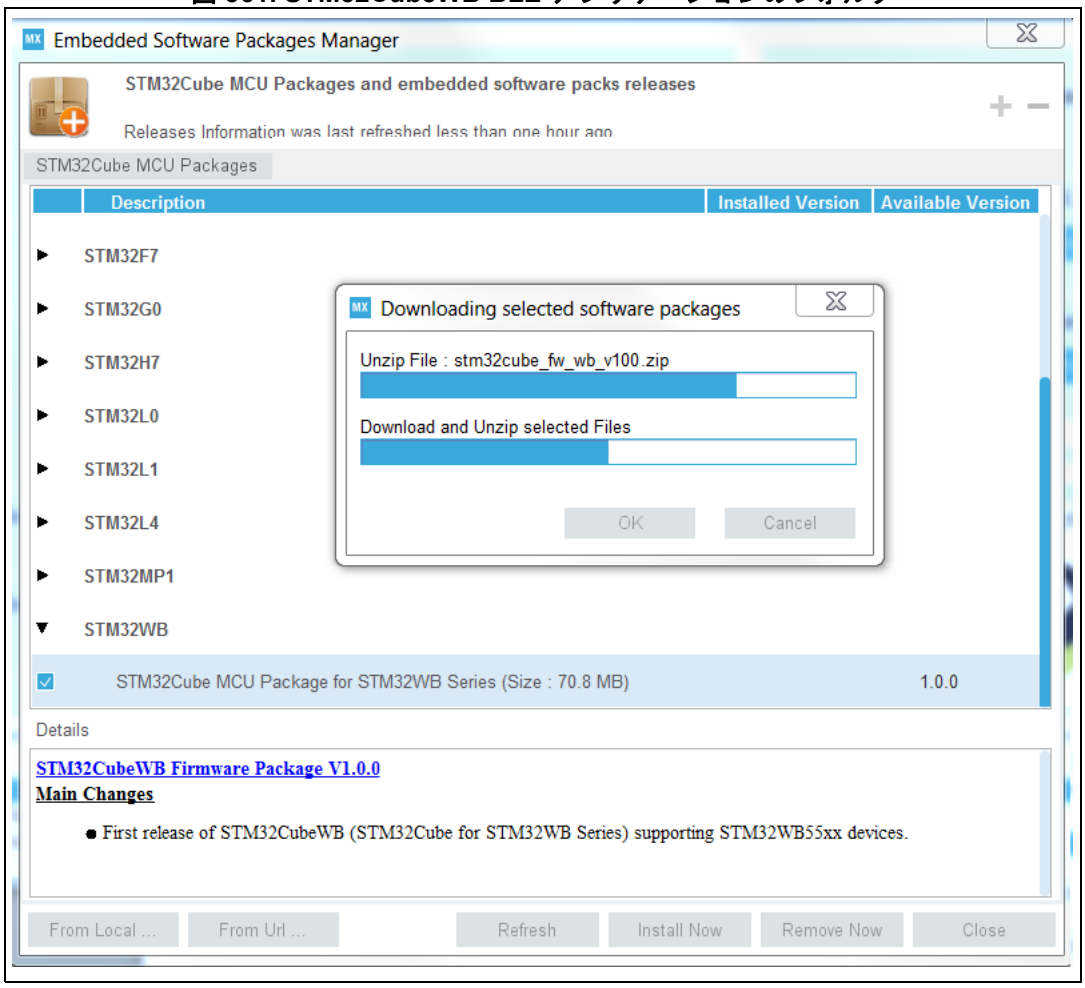
STM32CubeMX によって生成されるアプリケーション・プロジェクトは、STM32CubeWB マイクロコントローラ・パッケージのプロジェクト・フォルダにあります。

**図 360. STM32CubeWB パッケージのダウンロード**



このパッケージは、セクション 3.4.3 : STM32 マイクロコントローラ·パッケージのインストール で説明している標準的な手順に従うことで STM32CubeMX を介してインストールできます。

**図 361. STM32CubeWB BLE アプリケーションのフォルダ**




## BLE の設定

BLE を有効化するには、あらかじめいくつかのペリフェラル（RTC、HSEM、RF）をアクティブ化しておく必要があります。

つづいて、アプリケーションの種類として、トランスペアレント·モード、サーバ·プロファイル、ルータ·プロファイル、クライアント·プロファイルのいずれかを選択する必要があります。

最後に、このアプリケーションの種類に関連するモードとその他のパラメータを設定します。

注 : **BLE のトランスペアレント·モードと、すべての Thread アプリケーションでは、USART ペリフェラルまたは LPUART ペリフェラルの設定も必要になります。**

図 362. BLE のサーバ・プロファイルの選択

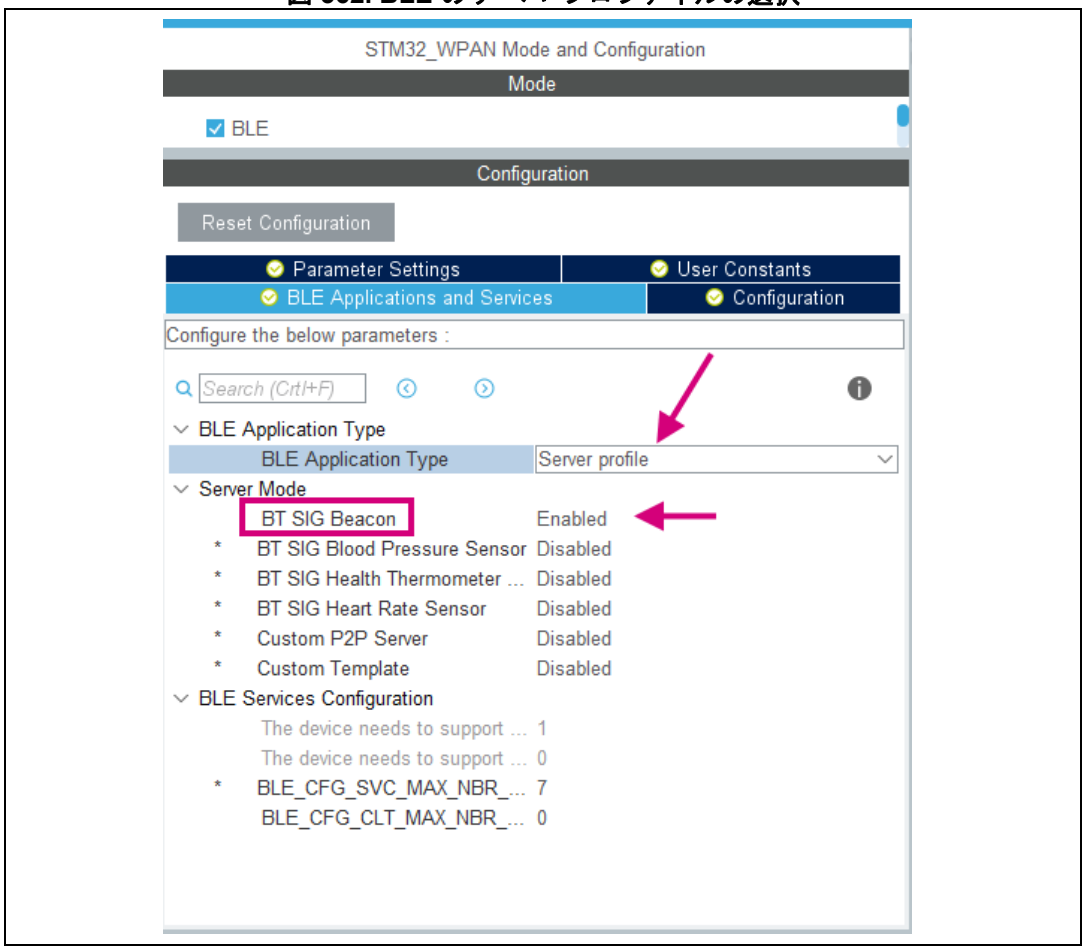
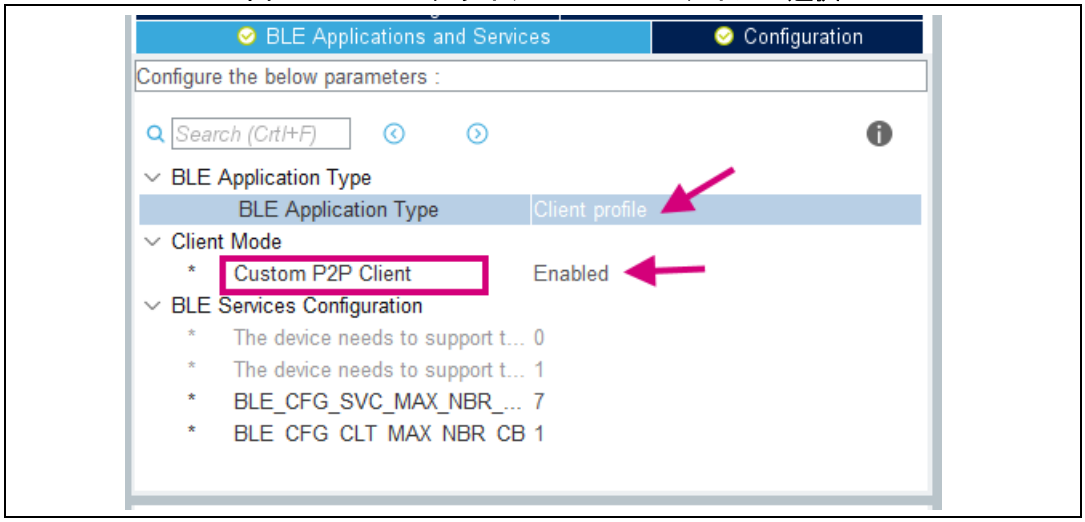



図 363. BLE のクライアント・プロファイルの選択

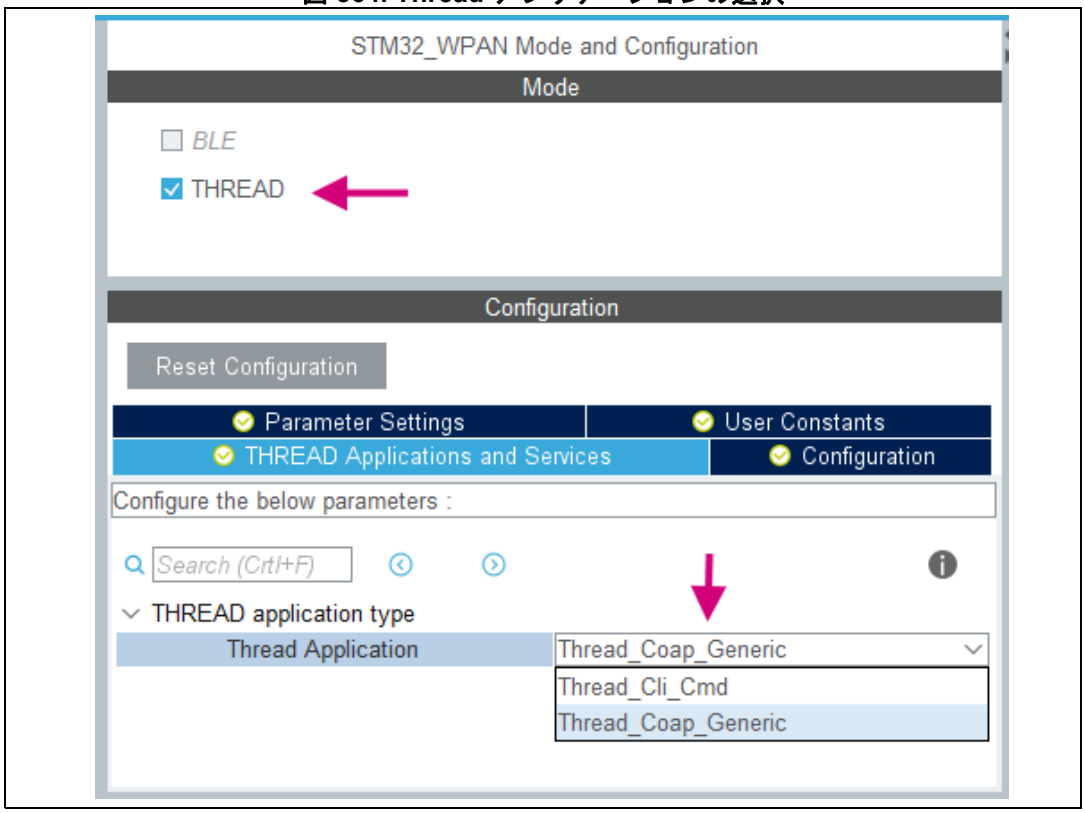


### Thread の設定

Thread を有効化するには、あらかじめいくつかのペリフェラル（RTC、HSEM、RF）をアクティブ化しておく必要があります。

つづいて、アプリケーションの種類を選択し、関連するパラメータを設定します。

**図 364. Thread アプリケーションの選択**




## B.3.12 OpenAmp と RESMGR_UTILITY（STM32MP1 シリーズおよび STM32H7 デュアルコア製品ライン）

デュアルコア製品に、複数コア間の連係動作を可能とする新しいソフトウェアとハードウェアが導入されました。

- STM32MP1 シリーズのみ：2 つのプロセッサ･インスタンス間のデータ交換に使用する IPCC（Inter-Processor communication controller ：プロセッサ間通信コントローラ）は、共有メモリ･バッファがマイクロコントローラの SRAM に割り当てられ、各プロセッサが固有のレジスタ･バンクと割込みを備えていることを前提としています。
- STM32MP1 シリーズのみ：Cortex-A コアと Cortex-M コア間の相互通信に使用される OpenAMP ミドルウェアは、RPMsg メッセージング･プロトコルを実装します（図 365 参照）。
- システム･リソース管理用のリソース･マネージャ･ライブラリ（RESMGR_UTILITY）：マルチプロセッサのデバイスでは、複数のコアで別々のファームウェアを実行できます（図 366 参照）。これは、あるペリフェラルがすでに使用されていることを認識できないコアが、そのペリフェラルの使用を試みる状況が発生する可能性を示唆しています。リソース管理ライブラリの役割は、ペリフェラルをそれ専用のコアに割り当て、さらにそのペリフェラルを動作させるために使用するシステム･リソースの設定方法を提供することです（図 367 参照）。

図 365. STM32MP1 デバイスの OpenAmp の有効化

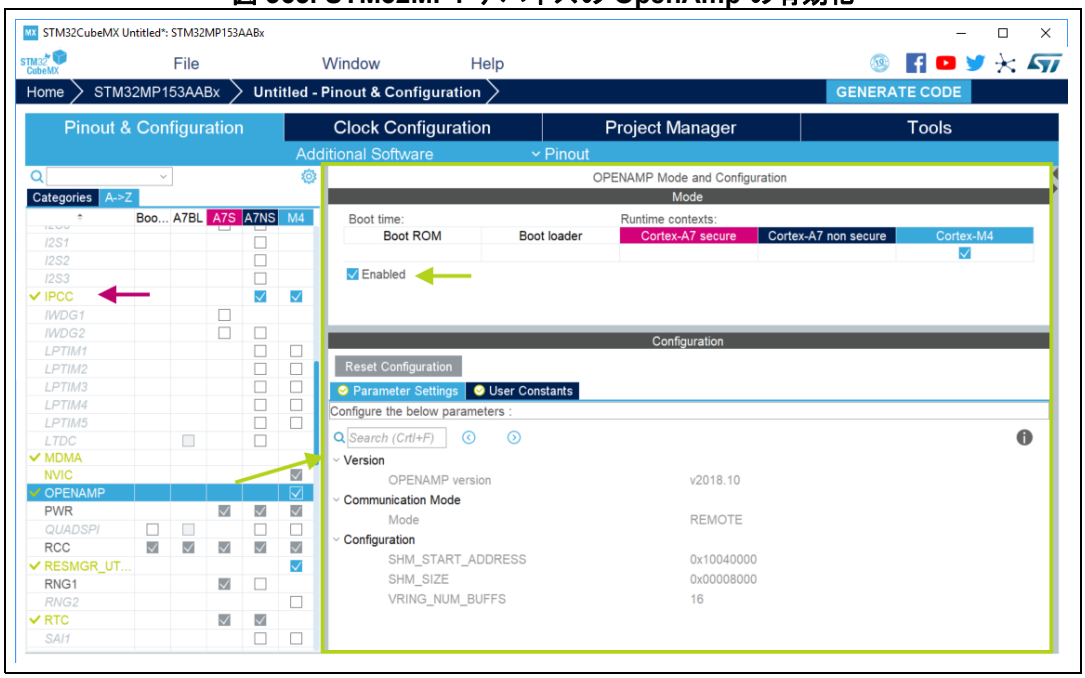



図 366. STM32MP1 デバイスのリソース･マネージャの有効化

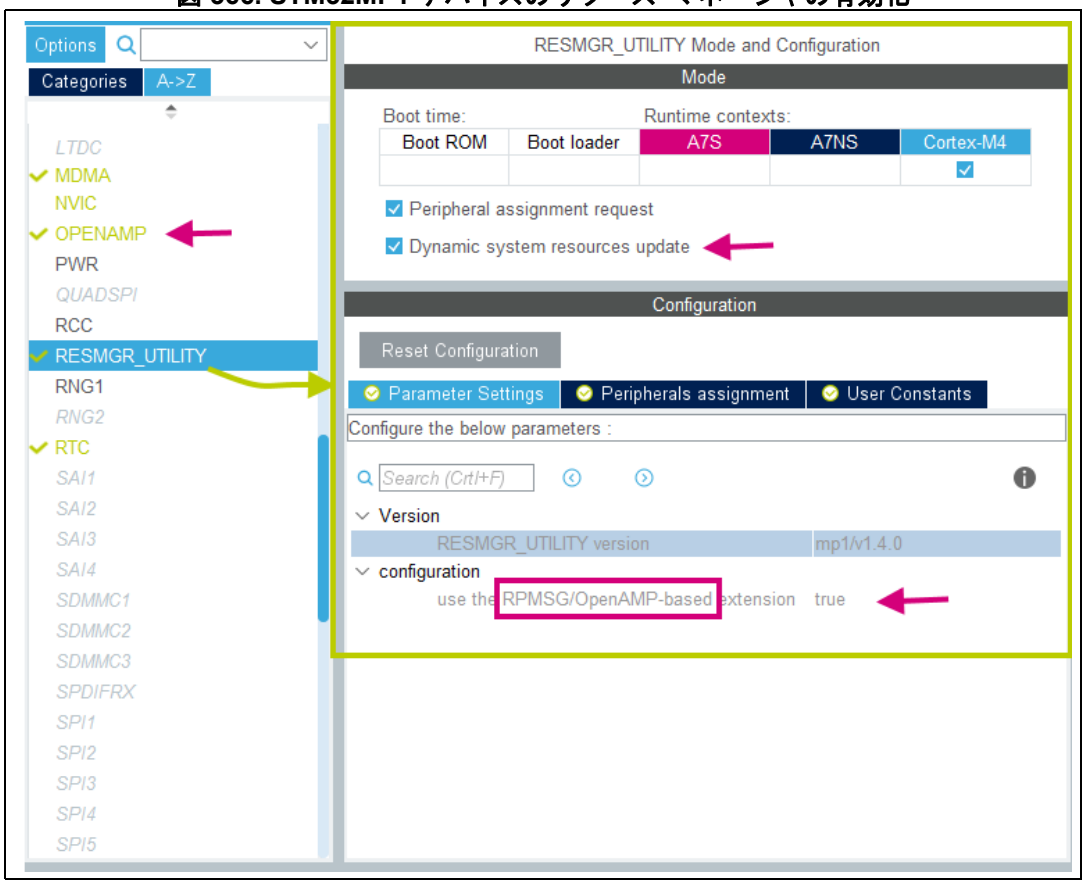
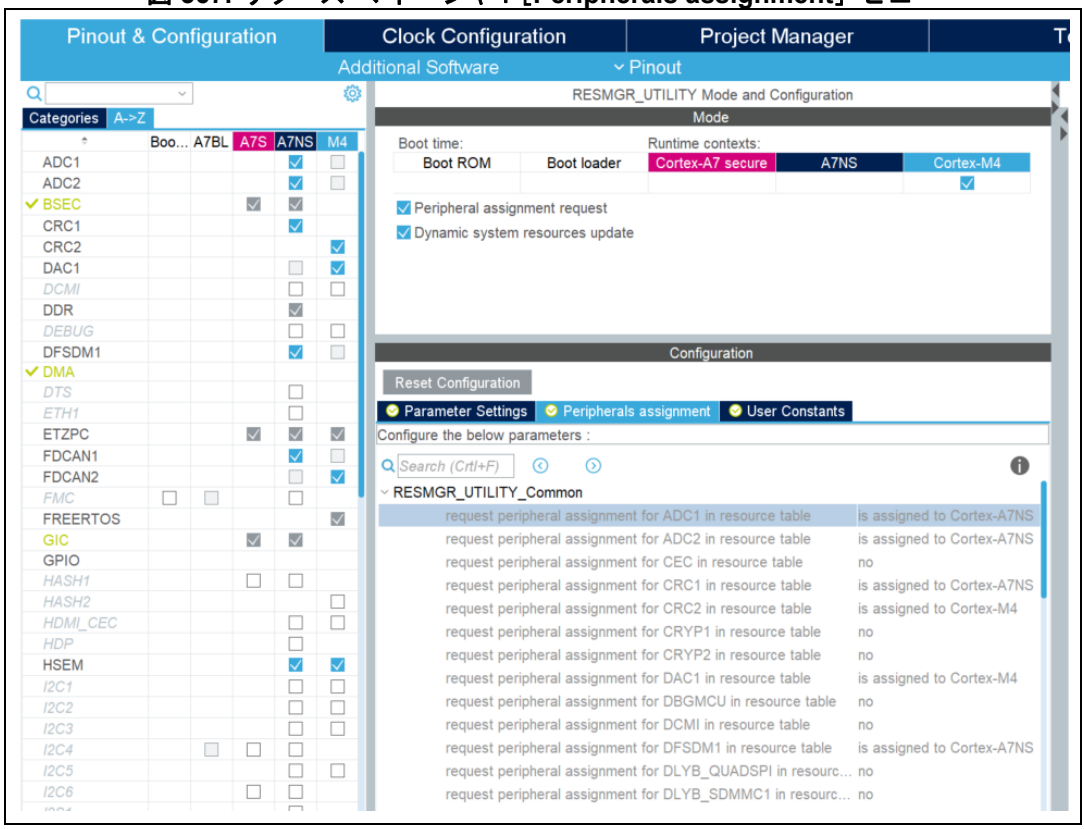


図 367. リソース･マネージャ：[Peripherals assignment] ビュー




詳細については、https://wiki.st.com/stm32mpu で STM32MP1 に関する wiki を参照してください。

# 付録 C STM32 マイクロコントローラの命名規則

STM32 マイクロコントローラの部品番号は、以下の命名規則に従って体系化されています。

- デバイスのサブファミリ

  部品番号が大きいほど、多くの機能を備えています。

  たとえば、STM32L0 ラインにはサブファミリとして STM32L051、STM32L052、STM32L053、STM32L061、STM32L062、STM32L063 がありますが、部品番号が STM32L06x の製品が備える AES を、STM32L05x の製品は備えていません。

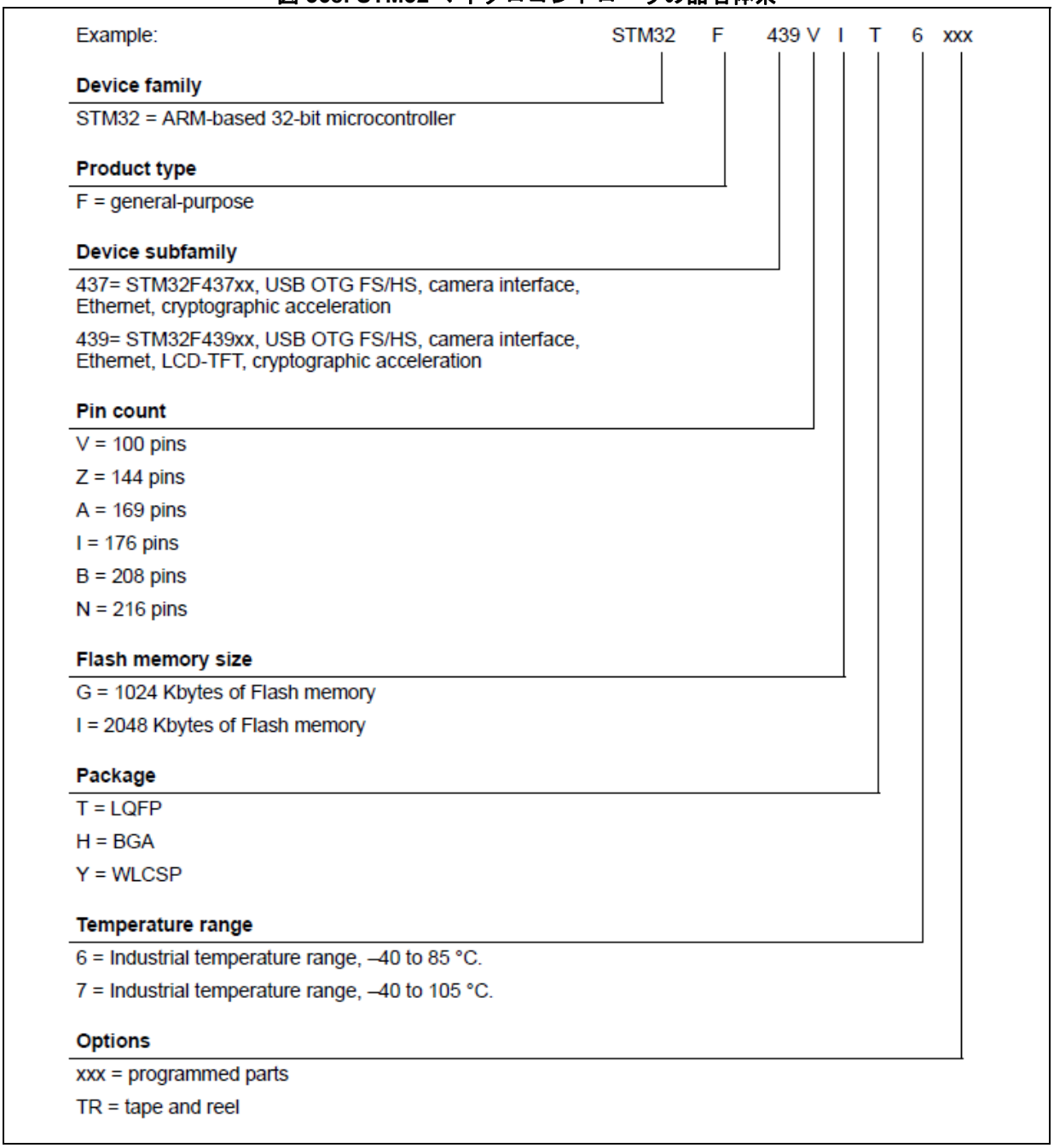
  末尾の桁は機能レベルを示します。上記の例では次のようになります。

  – 1 = アクセスライン
  – 2 = USB 搭載
  – 3 = USB と LCD を搭載
- ピン数
  – F = 20 ピン
  – G = 28 ピン
  – K = 32 ピン
  – T = 36 ピン
  – S = 44 ピン
  – C = 48 ピン
  – R = 64 ピンまたは 66 ピン
  – M = 80 ピン
  – O = 90 ピン
  – V = 100 ピン
  – Q = 132 ピン（STM32L162QDH6 など）
  – Z = 144 ピン
  – I = 176（+25）ピン
  – B = 208 ピン（STM32F429BIT6 など）
  – N = 216 ピン
- Flash･メモリ･サイズ
  – 4 = 16 KB の Flash メモリ
  – 6 = 32 KB の Flash メモリ
  – 8 = 64 KB の Flash メモリ
  – B = 128 KB の Flash メモリ
  – C = 256 KB の Flash メモリ
  – D = 384 KB の Flash メモリ
  – E = 512 KB の Flash メモリ
  – F = 768 KB の Flash メモリ
  – G = 1024 KB の Flash メモリ
  – I = 2048 KB の Flash メモリ
- パッケージ
  – B = SDIP
  – H = BGA

- M = SO
- P = TSSOP
- T = LQFP
- U = VFQFPN
- Y = WLCSP

図 368 に、STM32 マイクロコントローラの部品番号体系の例を示します。

**図 368. STM32 マイクロコントローラの品名体系**



# 付録 D STM32 マイクロコントローラの消費電力パラメータ

このセクションでは、STM32CubeMX の消費電力計算機能の使用方法を簡単に説明します。

マイクロコントローラの消費電力は、チップ·サイズ、電源電圧、クロック周波数、動作モードによって変化します。組込みアプリケーションでは、高速処理が不要なときにクロック周波数を低くすること、および最適な動作モードと電源電圧レンジを選択することで、STM32 マイクロコントローラの消費電力を最適化できます。STM32 の電力モードと電源電圧レンジについて以下で説明します。

## D.1 電力モード

STM32 マイクロコントローラは、さまざまな電力モードに対応しています（詳細は STM32 マイクロコントローラのデータシートを参照してください）。

### D.1.1 STM32L1 シリーズ

STM32L1 マイクロコントローラは最大で 6 種類の電力モードを備え、そのうちの 5 種類は省電力モードです。

- **RUN** モード

  HSE/HSI クロック·ソースを使用して最高のパフォーマンスが得られます。CPU は最高で 32 MHz のクロックで動作し、電圧レギュレータが有効になります。

- **SLEEP** モード

  HSE または HSI をシステム·クロック·ソースとして使用します。電圧レギュレータが有効で、CPU は停止します。すべてのペリフェラルは動作を継続し、割込みやイベントが発生したときは CPU をウェイクアップできます。

- **省電力 RUN**モード

  最低クロック周波数（131 kHz）に設定したマルチスピード内部（MSI）RC オシレータと省電力モードの内部レギュレータを使用します。クロック周波数と有効化できるペリフェラルの数に制限があります。

- **省電力 SLEEP** モード

  このモードは、SLEEP モードを開始することによって実現します。内部電圧レギュレータは省電力モードで動作します。クロック周波数と有効化できるペリフェラルの数に制限があります。典型的な例として、32 kHz で動作するタイマがあります。

  イベントや割込みによってウェイクアップがトリガされると、システムはレギュレータをオンにした実行モードに復帰します。

- **STOP** モード

  RAM とレジスタの内容を保持しながら、消費電力が最小になります。クロックは停止します。32 kHz の LSE と 37 kHz の LSI を使用することでリアルタイム·クロック（RTC）をバックアップできます。有効化できるペリフェラルの数に制限があります。電圧レギュレータは省電力モードで動作します。

  デバイスは、どの EXTI ラインによっても STOP モードからウェイクアップできます。

- **STANDBY** モード

  消費電力が最小になります。内部電圧レギュレータがオフになるので、$V_{CORE}$ ドメイン全体がオフになります。クロックは停止しますが、32 kHz の LSE と 37 kHz の LSI を使用することでリアルタイム·クロック（RTC）を保持できます。スタンバイ回路にあるレジスタを除き、RAM

とレジスタの内容は失われます。有効化できるペリフェラルの数は、STOP モードの場合よりも少なくなります。

デバイスは、リセット動作、3 本ある WKUP ピンのいずれかでの立ち上がりエッジ、または RTC イベント（RTC がオンの場合）の発生によって STANDBY モードを終了します。

注： **STOP モードまたは STANDBY モードを終了して RUN モードを開始した STM32L1 マイクロコントローラは、MSI オシレータをクロック･ソースとして使用する状態を経由します。この移行によって、全体の消費電力が大幅に増加することがあります。このことから、消費電力計算機能では、2 種類の移行ステップとしてWU_FROM_STOP と WU_FROM_STANDBY を導入しています。これらのステップでは、クロックが自動的に MSI に設定されます。**

### D.1.2 STM32F4 シリーズ

STM32F4 マイクロコントローラは合計で 5 種類の電力モードを備え、そのうちの 4 種類は省電力モードです。

- **RUN** モード

  電源をオンにしたときまたはシステムをリセットしたときのデフォルトのモードです。HSE/HSI クロック･ソースを使用して最高のパフォーマンスが得られます。CPU は、選択した電力スケールに応じて最高の周波数で動作します。

- **SLEEP** モード

  CPU のみが停止します。すべてのペリフェラルは動作を継続し、割込みやイベントが発生したときは CPU をウェイクアップできます。SLEEP モードになる前に設定されていたクロックがクロック･ソースになります。

- **STOP** モード

  RC オシレータをクロック･ソースとして使用することにより、消費電力がきわめて少なくなります。CPU とペリフェラル同様に、1.2 V ドメインのクロックもすべて停止します。PLL、HSI RC、HSE のクリスタル･オシレータはディスエーブルになります。レジスタと内部 SRAM の内容は保持されます。

  電圧レギュレータは、通常のメイン･レギュレータ･モード（MR）と省電力レギュレータ･モード（LPR）のどちらにも設定できます。省電力レギュレータ･モードのレギュレータを選択すると、ウェイクアップに要する時間が長くなります。

  Flash･メモリは、高速なウェイクアップが可能な STOP モードに置くことができるほか、ウェイクアップに時間を要するものの、消費電力が少なくなるディープ･パワーダウンとすることもできます。

  STOP モードには次のサブモードがあります。

  – **ノーマル･モードで停止（デフォルト･モード）**

    1.2 V ドメインは通常リーク電流モードに保持され、最小 V12 電圧は 1.08 V です。

  – **アンダードライブ･モードで停止**

    1.2 V ドメインは低リーク電流モードに保持され、V12 電圧は 1.08 V 未満になります。メイン･モードまたは省電力モードのレギュレータは、アンダードライブ･モードまたは省電力モードになります。Flash･メモリは、ディープ･パワーダウン･モードにする必要があります。ウェイクアップ時間は、ノーマル･モードよりも約 100 マイクロ秒長くなります。

- **STANDBY** モード

  RC オシレータをクロック･ソースとして使用することにより、消費電力がきわめて少なくなります。内部電圧レギュレータがオフになるので、1.2 V ドメイン全体がオフになります。CPU とペリフェラルは停止します。PLL、HSI RC、HSE のクリスタル･オシレータはディスエーブルになります。バックアップ･ドメインのレジスタと選択した 4 バイト･バックアップ SRAM を除き、SRAM とレジスタの内容は失われます。RTC オシレータと LSE オシレータの各ブロックのみがオンになります。外部リセット（NRST ピン）、IWDG リセット、WKUP ピンでのパルスの

立ち上がりエッジ、RTC のアラーム/ウェイクアップ/タンパー /タイム･スタンプ･イベントのいずれかが発生すると、デバイスは STANDBY モードを終了します。

- **$V_{BAT}$ 動作**

  この動作では、STANDBY モードよりも消費電力を大幅に低減できます。バックアップ･ドメインに電源を供給する $V_{BAT}$ ピンをバッテリなどの別の電源に接続すると、このモードを使用できます。$V_{BAT}$ ドメイン（RTC レジスタ、RTC バックアップ･レジスタ、バックアップ SRAM）は保持され、RTC オシレータと LSE オシレータの各ブロックはオンになります。STANDBY モードとの主な相違は、外部割込みと RTC アラーム/イベントがあっても、デバイスが $V_{BAT}$ 動作を終了しないことです。$V_{DD}$ が最小閾値まで上昇すると、この動作が終了します。

### D.1.3 STM32L0 シリーズ

STM32L0 マイクロコントローラには、最大で 8 種類の電力モードがあります。そのうちの 7 種類は、低消費電力、起動時間の短縮、利用可能なウェイクアップ･リソースとの最適なバランスを実現する省電力モードです。

- **RUN モード**

  HSE/HSI クロック･ソースを使用して最高のパフォーマンスが得られます。CPU は最高で 32 MHz のクロックで動作でき、電圧レギュレータが有効になります。

- **SLEEP モード**

  HSE または HSI をシステム･クロック･ソースとして使用します。電圧レギュレータが有効で、CPU のみが停止します。すべてのペリフェラルは動作を継続し、割込みやイベントが発生したときは CPU をウェイクアップできます。

- **低電力 RUN モード**

  内部レギュレータを省電力モードで使用し、マルチスピード内部（MSI）RC オシレータを最低クロック周波数（131 kHz）に設定します。省電力 RUN モードでは、クロック周波数と有効化できるペリフェラルの数の両方に制限があります。

- **省電力 SLEEP モード**

  内部電圧レギュレータを省電力モードにして SLEEP モードを開始することでこのモードを実現します。クロック周波数と有効化できるペリフェラルの数の両方に制限があります。イベントや割込みがあると、レギュレータがオンになって RUN モードに復帰できます。

- **RTC を使用した STOP モード**

  この STOP モードでは、RAM レジスタの内容とリアルタイム･クロックを保持した条件下で最小の消費電力を実現します。電圧レギュレータは省電力モードで動作します。LSE や LSI は動作を継続します。$V_{CORE}$ ドメインのすべてのクロックが停止し、PLL、MSI RC、HSEクリスタル、および HSI RC オシレータがディスエーブルになります。

  ウェイクアップ機能を備えたペリフェラルの中には、STOP モードでも HSI RC を有効にしていて、ウェイクアップ条件を検出できるものがあります。どの EXTI ラインによってもデバイスは STOP モードから 3.5 マイクロ秒以内にウェイクアップでき、プロセッサは割込みの生成やコードへの復帰が可能です。

- **RTC を使用しない STOP モード**

  RTC クロックを停止する点を除き、“RTC を使用した STOP モード”と同じです。

- **RTC を使用した STANDBY モード**

  この STANDBY モードでは、リアルタイム･クロックが動作している条件下で最小の消費電力を実現します。内部電圧レギュレータがオフになるので、$V_{CORE}$ ドメイン全体がオフになります。

PLL、MSI RC、HSE クリスタル、および HSI RC オシレータも電源オフになります。LSE や LSI は動作を継続します。

STANDBY モードになると、スタンバイ回路（ウェイクアップ･ロジック、IWDG、RTC、LSI、LSE クリスタル 32 kHz オシレータ、RCC_CSR レジスタ）のレジスタを除き、RAM とレジスタの内容は失われます。

外部リセット（NRST ピン）、IWDG リセット、3 本ある WKUP ピンのいずれかでの立ち上がりエッジ、RTC アラーム（アラーム A またはアラーム B）、

RTC タンパー･イベント、RTC タイムスタンプ･イベント、または RTC ウェイクアップ･イベントが発生すると、デバイスは 60 マイクロ秒以内に STANDBY モードを終了します。

- **RTC を使用しない STANDBY モード**

  RTC の LSE クロックと LSI クロックを停止する点を除き、RTC を使用した STANDBY モードと同じです。

  外部リセット（NRST ピン）または 3 本ある WKUP ピンのいずれかでの立ち上がりエッジが発生すると、デバイスは 60 マイクロ秒以内に STANDBY モードを終了します。

**注：** **STOP モードまたは STANDBY モードを開始しても、RTC、IWDG、および対応するクロック･ソースは自動的には停止しません。STOP モードを開始しても、LCD は自動的には停止しません。**

## D.2 消費電力レンジ

STM32 マイクロコントローラでは、ダイナミック電圧スケーリング機能を使用することで、消費電力をさらに最適化できます。このスケーリング機能では、電圧レンジ（STM32L1 と STM32L0）または電圧スケール（STM32 F4）を選択することで、ロジック（CPU、デジタル･ペリフェラル、SRAM、およびFlash･メモリ）に供給するメイン内部レギュレータ出力電圧 V12 をソフトウェアで調整できます。

消費電力レンジの定義を以下に示します（詳細は STM32 マイクロコントローラのデータシートを参照してください）。

### D.2.1 STM32L1 シリーズが備える 3 種類の $V_{CORE}$ レンジ

- CPU の最大クロック周波数を 32 MHz とした高性能の**レンジ 1**（$V_{DD}$ のレンジを 2.0 ～ 3.6 V に制限）

  $V_{DD}$ 入力電圧が 2.0 V を超えていれば、電圧レギュレータから 1.8 V の電圧（typ）が出力されます。Flash･プログラムと消去動作を実行できます。
- CPU の最大クロック周波数を 16 MHz とした中間性能の**レンジ 2**（$V_{DD}$ のレンジ全体を使用）

  1.5 V で、Flash メモリは機能しますが、読出しアクセス時間は中程度です。Flash プログラムと消去動作を実行できます。
- CPU の最大クロック周波数を 4 MHz（マルチスピード内部 RC オシレータのクロック･ソースでのみ生成）とした低性能の**レンジ 3**（$V_{DD}$ のレンジ全体を使用）

  1.2 V で、Flash メモリは機能しますが、読出しアクセス時間は長くなります。Flash プログラムと消去動作は実行できなくなります。

### D.2.2 STM32F4 シリーズが備えるいくつかの $V_{CORE}$ スケール

このスケールは、PLL がオフで HSI または HSE をシステム･クロック･ソースとして選択している場合にのみ変更できます。

- **スケール 1**（V12 の電圧レンジを 1.26 ~ 1.40 V に制限）。リセット時のデフォルト･モード
  HCLK 周波数レンジ = 144 ~ 168 MHz（オーバードライブで 180 MHz）。
  リセット時のデフォルト･モードです。
- **スケール 2**（V12 の電圧レンジを 1.20 ~ 1.32 V に制限）
  HCLK 周波数レンジは最大 144 MHz （オーバードライブで 168 MHz）
- **スケール 3**（V12 の電圧レンジを 1.08 ~ 1.20 V に制限）。STOP モードを終了したときのデフォルト･モード
  HCLK 周波数は 120 MHz 以下。

電圧スケールは、周波数 $f_{HCLK}$ に基づいて次のように調整されます。

- **STM32F429x/39x マイクロコントローラ**：
  - **スケール 1**：最大 168 MHz（オーバードライブで最大 180 MHz）
  - **スケール 2:** 120 ~ 144 MHz（オーバードライブで最大 168 MHz）
  - **スケール 3:** 最大 120 MHz
- **STM32F401x マイクロコントローラ**：
  スケール 1 なし
  - **スケール 2:** 60 ~ 84 MHz
  - **スケール 3:** 最大 60 MHz
- **STM32F40x/41x マイクロコントローラ**：
  - **スケール 1:** 最大 168 MHz
  - **スケール 2:** 最大 144 MHz

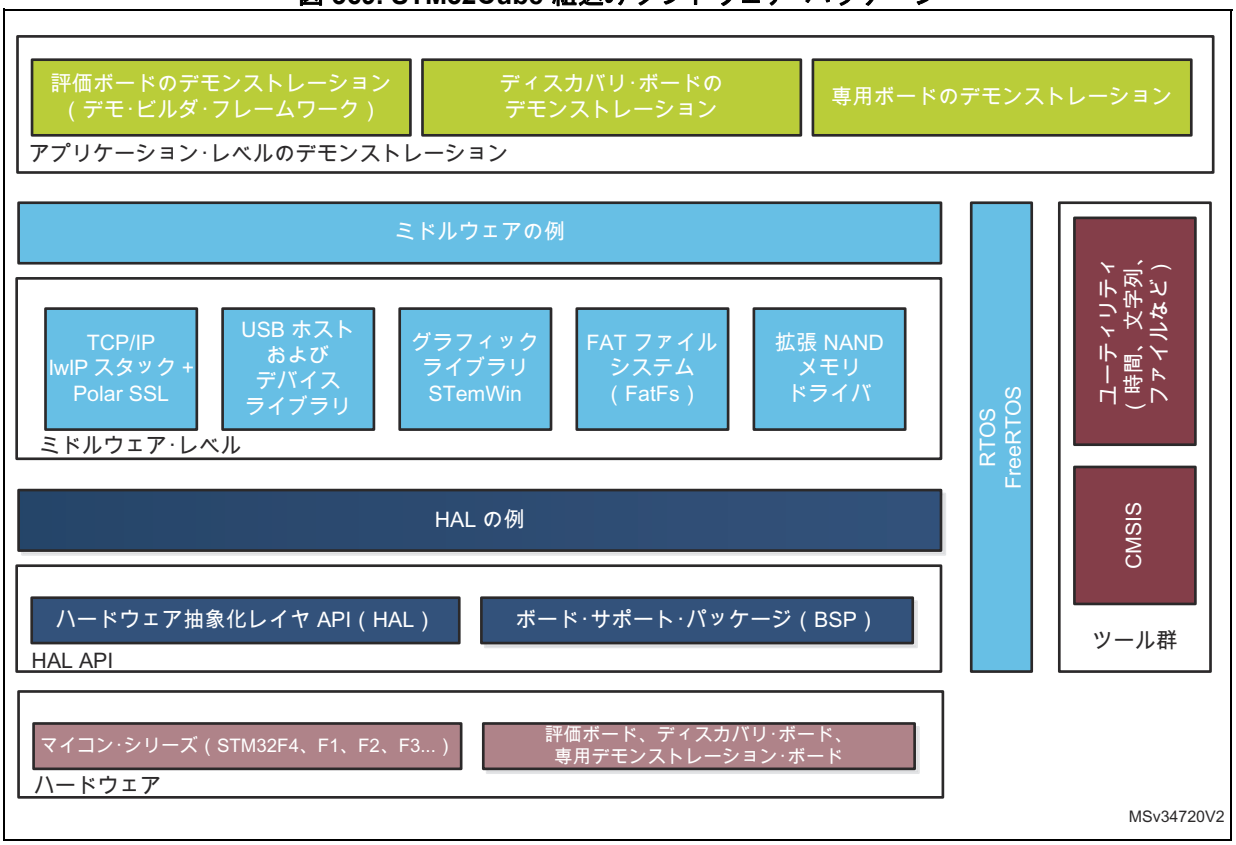
### D.2.3 STM32L0 シリーズが備える 3 種類の $V_{CORE}$ レンジ

- CPU の最大クロック周波数を 32 MHz としたレンジ 1（$V_{DD}$ のレンジを 1.71 ~ 3.6 V に制限）
- CPU の最大クロック周波数を 16 MHz としたレンジ 2（$V_{DD}$ のレンジ全体を使用）
- CPU の最大クロック周波数を 4.2 MHz に制限したレンジ 3（$V_{DD}$ のレンジ全体を使用）

# 付録 E STM32Cube 組込みソフトウェア･パッケージ

組込みソフトウェア･パッケージは、STM32CubeMX の C コード生成機能とともに、STM32Cube イニシアチブの構成要素です（DB2164 データシートを参照）。これらのパッケージは、低レベルのハードウェア抽象化レイヤ（HAL）を用意しています。この HAL は、ST マイクロエレクトロニクス製ボード上で動作する数多くのサンプルと使用することで、マイクロコントローラのハードウェアを対象としています（図 369 を参照）。これらのコンポーネントは、STM32 シリーズ全体にわたって容易に移植可能です。このパッケージには、STM32CubeMX で生成した C コードとの全面的な互換性があります。

**図 369. STM32Cube 組込みソフトウェア･パッケージ**




**注：** **STM32CubeF0、STM32CubeF1、STM32CubeF2、STM32CubeF3、STM32CubeF4、STM32CubeL0、および STM32CubeL1 の各組込みソフトウェア･パッケージは st.com で入手可能です。これらのパッケージは、STM32Cube リリース v1.1 に基づいており（他のシリーズも追って導入の予定です）、初期化 C コードの生成に STM32CubeMX で使用する組込みソフトウェア･ライブラリを備えています。**

**STM32 のアプリケーション開発に着手するには、STM32CubeMX を使用して、初期化 C コードとパッケージに用意されているサンプルを生成します。**

# 改版履歴

表 24. 文書改版履歴

| 日付 | 版 | STM32CubeMX リリース番号 | 変更内容 |
|---|---|---|---|
| 2014 年 2 月 17 日 | 1 | 4.1 | 初版発行 |
| 2014 年 4 月 4 日 | 2 | 4.2 | STM32CubeF2 および STM32F2 シリーズのサポートを、表紙、セクション 2.2 : 主な機能、セクション 5.14.1: ペリフェラルとミドルウェアの［Configuration］ウィンドウ、付録 E : STM32Cube 組込みソフトウェア･パッケージに追加。<br>セクション 11.1 : 新しい STM32CubeMX プロジェクトの作成、セクション 11.2 : マイクロコントローラのピン配置の設定、セクション 11.6 : マイクロコントローラの初期化パラメータの設定を更新。<br>GPIO 初期化 C コードの生成に関する節をセクション 8:チュートリアル 3 ー GPIO 初期化 C コードの生成（STM32F1 シリーズのみ）に移動して、内容を更新。<br>セクション 18.6 : "Java 8 update 45" 以降のバージョンの JRE をインストールするときに "Java 8 update 45" エラーが発生する理由を追加。 |
| 2014 年 4 月 24 日 | 3 | 4.3 | STM32CubeL0 および STM32L0 シリーズのサポートを、表紙、セクション 2.2 : 主な機能、セクション 2.3 : ルールと制限事項、セクション 5.14.1: ペリフェラルとミドルウェアの［Configuration］ウィンドウに追加。<br>ボードの選択を、表 13: ［File］メニューの機能、セクション 5.7.3: ［Pinout］メニュー、セクション 4.2 : [New Project] ウィンドウに追加。表 15 : ［Pinout］メニューを更新。<br>図 130 : [Power Consumption Calculator] のデフォルト･ビューを更新し、セクション 5.2.1 : 消費電力シーケンスの構築にバッテリの選択を追加。<br>セクション 5.2 : ［Power Consumption Calculator］ビューの注を更新<br>セクション 11.1 : 新しい STM32CubeMX プロジェクトの作成を更新。<br>セクション 19.7 : [Clock tree] ビューで RTC マルチプレクサが非アクティブのままになるのはなぜですか。、セクション 19.8: クロック･ソースとして LSE と HSE を選択して周波数を変更するにはどうすればいいですか。、およびセクション 19.9 : STM32CubeMX で、PC13、PC14、PC15、PI8 のいずれかを出力として設定済みであっても、これらを出力として 設定できないのはなぜですか。を追加。 |

表 24. 文書改版履歴 （続き）

| 日付 | 版 | STM32CubeMX リリース番号 | 変更内容 |
|---|---|---|---|
| 2014 年 6 月 19 日 | 4 | 4.4 | STM32CubeF0、STM32CubeF3、STM32F0、および STM32F3 シリーズのサポートを表紙、セクション 2.2 : 主な機能、セクション 2.3 : ルールと制限事項に追記。<br>ボード選択機能とピンのロック機能をセクション 2.2 : 主な機能、表 2 : [Home] ページのショートカット、セクション 4.2 : [New Project] ウィンドウ、セクション 5.7: ツールバーとメニュー、セクション 4.11 : [Set unused GPIOs]/[Reset used GPIOs]ウィンドウ、セクション 4.9: [Project Manager] ビュー、セクション 5.15: [Pinout] ビューに追加。セクション 5.15.1 : ピンへの信号の固定と信号へのラベルの割当てを追加。<br>セクション 5.16: [Configuration] ビュー、セクション 4.8 : [Clock Configuration] ビュー、セクション 5.2 : [Power Consumption Calculator] ビューを更新。<br>図 39 : マイクロコントローラを選択したときの STM32CubeMX のメイン・ウィンドウ、図 101 : [Project Settings] ウィンドウ、図 128 : [About] ウィンドウ、図 140: STM32CubeMX [Pinout] ビュー、図 120: [Chip] ビュー、図 130 : [Power Consumption Calculator] のデフォルト・ビュー、図 131 : バッテリの選択、図 87: 消費電力シーケンスの構築、図 133 : 消費電力のシーケンス : [New Step] のデフォルト・ビュー、図 140 : シーケンス構築後の [Power Consumption Calculator] ビュー、図 141 : [Sequence Table] の管理機能、図 88: PCC の [Edit Step] ウィンドウ、図 83: 消費電力シーケンス : 設定した新しいステップ（STM32F4 の例）、図 138 : [Pinout] ビューで選択された ADC, 図 139 : [Power Consumption Calculator] のステップ設定ウィンドウ : [Import Pinout] を使用して有効にした ADC、図 143 : 結果エリアの表記、図 100: ペリフェラルの消費電力に関するツールチップ、図 261 : 消費電力計算の例、図 155: [Sequence Table]、図 156: 消費電力の計算結果を更新。<br>図 142: STM32CubeMX [Configuration] ビュー、図 39: STM32CubeMX の設定ビュー - STM32F1 シリーズのタイトルを更新。<br>セクション 5.2: [Power Consumption Calculator] ビューに STM32L1 を追加。<br>PCC パネルを使用した新しいステップの追加の図をセクション 8.1.1 ステップの追加から削除。シーケンスへの新しいステップの追加の図をセクション 5.2.2 : 消費電力シーケンスでのステップの設定から削除。<br>セクション 8.2 : 結果のレビューを更新。<br>付録B.3.4 : FatFsおよび付録 D : STM32 マイクロコントローラの消費電力パラメータを更新。付録D.1.3 : STM32L0 シリーズおよびD.2.3 : STM32L0 シリーズが備える 3 種類の VCORE レンジを追加。 |

表 24. 文書改版履歴 （続き）

| 日付 | 版 | STM32CubeMX リリース番号 | 変更内容 |
|---|---|---|---|
| 2014 年 9 月 19 日 | 5 | 4.5 | STM32CubeL1 シリーズのサポートを表紙、セクション 2.2 : 主な機能、セクション 2.3 : ルールと制限事項に追記。<br>セクション 3.2.3 : STM32CubeMX のスタンドアロン･バージョンのアンインストールを更新。<br>オフライン更新をセクション 3.4 : STM32CubeMX による更新の取得に追加し、図 8 : ［Embedded Software Packages Manager］ウィンドウおよびセクション 3.4.3 : STM32 マイクロコントローラ･パッケージのインストールの内容を修正。<br>セクション 4 : STM32CubeMX のユーザ･インタフェースの概要、表 2 : ［Home］ページのショートカット、およびセクション 4.2 : ［New Project］ウィンドウを更新。<br>図 31 : ［New Project］ウィンドウ - ［Board Selector］を追加。<br>図 109 : ［Project Settings］-［Code Generator］を更新。<br>セクション 4.9 : ［Project Manager］ビューのステップ 3 を変更。<br>図 39 : STM32CubeMX の設定ビュー ー STM32F1 シリーズを更新。<br>セクション 5.14.1: ペリフェラルとミドルウェアの［Configuration］ウィンドウに STM32L1 を追加。<br>図 63 : GPIO 設定ウィンドウ - GPIO の選択、セクション 4.4.12 : GPIO設定ウィンドウ、および図 68 : DMA の MemToMem 設定を更新。<br>セクション 4.8 : ［Clock Configuration］ビュー の概要を更新。セクション 4.8.1 : クロック･ツリーの設定機能、セクション 4.8.3 : 推奨事項、セクション 5.2 : ［Power Consumption Calculator］ビュー、図 133 : 消費電力のシーケンス : ［New Step］のデフォルト･ビュー、図 140 : シーケンス構築後の［Power Consumption Calculator］ビュー、図 83 : 消費電力シーケンス : 設定した新しいステップ（STM32F4 の例）、および図 139 : ［Power Consumption Calculator］のステップ設定ウィンドウ : ［Import Pinout］を使用して有効にした ADCを更新。図 142 : 消費電力 : ペリフェラルの消費電力チャートを追加、図 100: ペリフェラルの消費電力に関するツールチップを更新。セクション 5.2.4 : 消費電力シーケンス･ステップ･パラメータの用語を更新。<br>セクション 6 : STM32CubeMX の C コード生成の概要を更新。<br>セクション 11.1 : 新しい STM32CubeMX プロジェクトの作成 および セクション 11.2 : マイクロコントローラのピン配置の設定を更新。<br>セクション 12 : チュートリアル 2 - STM32429I-EVAL 評価 ボードを使用した、SD カード上の FatFs の例を追加し、セクション 8 : チュートリアル 3 ー GPIO 初期化 C コードの生成（STM32F1 シリーズのみ）を更新。<br>セクション 5.2.2 : 消費電力シーケンスでのステップの設定を更新。 |

表 24. 文書改版履歴 （続き）

| 日付 | 版 | STM32CubeMX リリース番号 | 変更内容 |
|---|---|---|---|
| 2015 年 1 月 19 日 | 6 | 4.6 | すべての STM32 シリーズで、完全なプロジェクトの生成、消費電力の計算、およびクロック･ツリーの設定が可能。<br>セクション 2.2 : 主な機能 および セクション 2.3 : ルールと制限事項を更新。<br>セクション 3.1.3 : ソフトウェア要件で、Eclipse IDE を更新。<br>図 6 : [Updater Settings] ウィンドウ、図 8 : [Embedded Software Packages Manager] ウィンドウ、および図 31 : [New Project] ウィンドウ - [Board Selector] を更新。セクション 4.9 : [Project Manager] ビューおよびセクション 4.12 : [Update Manager] ウィンドウを更新。<br>図 128 : [About] ウィンドウを更新。<br>STM32CubeMX の設定ビュー ー STM32F1 シリーズの図を削除。<br>表 17: STM32CubeMX チップ･ビュー - アイコンとカラー･スキームを更新。<br>セクション 5.14.1: ペリフェラルとミドルウェアの [Configuration] ウィンドウを更新。<br>図 66 : 新しい DMA リクエストの追加 および 図 68 : DMA の MemToMem 設定を更新。<br>セクション 4.8.1 : クロック･ツリーの設定機能を更新。<br>図 131 : バッテリの選択、図 87 : 消費電力シーケンスの構築、図 88: PCC の [Edit Step] ウィンドウを更新。<br>セクション 6.3 : カスタム･コードの生成を追加。<br>図 215 : クロック･ツリー･ビュー および 図 220 : [Pinout & Configuration] ビューを更新。<br>セクション 11.6.2 : ペリフェラルの設定 の ペリフェラル設定シーケンスおよび図 222 : タイマ 3 の設定ウィンドウを更新。<br>チュートリアル 3 : GPIO 初期化 C コードの生成（STM32F1 シリーズのみ）を削除。<br>図 226 : GPIO モード設定を更新。<br>図 261 : 消費電力計算の例および図 155 : シーケンス･テーブルを更新。<br>付録A.1 : ブロックの整合性、A.2 : ブロックの相互依存性、およびA.3 : 1 つのブロックを 1 つのペリフェラルとするモードを更新。<br>付録A.4 : ブロックの再マッピング（STM32F10x のみ）: セクション : 例を更新。<br>付録A.6 : ブロックのシフト（STM32F10x で [Keep Current Signals Placement] がチェックされていない場合のみ）: セクション : 例を更新<br>付録A.8 : 機能の個別マッピングを更新。<br>付録B.3.1 : 概要を更新。<br>付録D.1.3 : STM32L0 シリーズを更新。 |

表 24. 文書改版履歴 （続き）

| 日付 | 版 | STM32CubeMX リリース番号 | 変更内容 |
|---|---|---|---|
| 2015 年 3 月 19 日 | 7 | 4.7 | セクション 2.2 : 主な機能 : STM32F1 シリーズのピン配置初期化 C コードの生成を削除、完全なプロジェクトの生成を更新。<br>図 8 : ［Embedded Software Packages Manager］ウィンドウ、図 31 : ［New Project］ウィンドウ - ［Board Selector］を更新。<br>セクション 4.9 : ［Project Manager］ビューの IDE リストを更新し、図 101 : ［Project Settings］ウィンドウの内容を修正。<br>セクション 4.8.1 : クロック･ツリーの設定機能を更新。図 97 : STM32F469NIHx のクロック･ツリー設定ビューを更新。<br>セクション 5.2 : ［Power Consumption Calculator］ビュー : トランジション･チェッカー･オプションを追加。図 130 : ［Power Consumption Calculator］のデフォルト･ビュー、図 131 : バッテリの選択、図 87: 消費電力シーケンスの構築を更新。図 134 : 設定済みのシーケンスに対するトランジション･チェッカー･ オプションの使用 - すべてのトランジションが有効な場合、図 135 : 設定済みのシーケンスに対するトランジション･チェッカー･ オプションの使用 - 1 つ以上のトランジションが無効な場合、および図 136 : トランジション･チェッカー･オプション - ログの表示を追加。図 140 : シーケンス構築後の［Power Consumption Calculator］ビューを更新。セクション ：シーケンス･ステップの管理、セクション ：シーケンス全体の管理（ロード、保存、および比較）を更新。図 88 : PCC の［Edit Step］ウィンドウおよび図 143 : 結果エリアの表記を更新。<br>図 261 : 消費電力計算の例、図 155 : ［Sequence Table］、図 156 : 消費電力の計算結果および図 158 : 消費電力結果 ー IP 消費チャートを更新。<br>付録B.3.1 : 概要およびB.3.5 : FreeRTOSを更新。 |
| 2015 年 5 月 28 日 | 8 | 4.8 | セクション 3.2.2 : コマンドラインからの STM32CubeMX のインストールおよびセクション 3.3.2: コマンドライン･モードでの STM32CubeMX の実行を追加。 |
| 2015 年 7 月 09 日 | 9 | 4.9 | STLM32F7 および STM32L4 のマイクロコントローラ･シリーズを追加。<br>プロジェクトのインポート機能を追加。表 13: ［File］メニューの機能にインポート機能を追加。セクション 4.10 : ［Import Project］ウィンドウを追加。図 133 : 消費電力のシーケンス : ［New Step］のデフォルト･ビュー、図 88 : PCC の［Edit Step］ウィンドウ、図 83 : 消費電力シーケンス : 設定した新しいステップ（STM32F4 の例）、図 139 : ［Power Consumption Calculator］のステップ設定ウィンドウ : ［Import Pinout］を使用して有効にした ADC、および図 87: ペリフェラルの消費電力に関するツールチップを更新。<br>セクション 3.3.2 : コマンドライン･モードでの STM32CubeMX の実行で、STM32CubeMX を実行するコマンドラインを更新。<br>セクション 5.16: ［Configuration］ビューの注釈を更新。<br>新しいクロック･ツリー設定機能をセクション 4.8.1 : に追加。<br>図 228 : ミドルウェアのツールチップを更新。<br>付録B.1 : STM32CubeMX で生成した C コードとユーザ･セクションのコード例の内容を修正。<br>付録B.3.1 : 概要を更新。<br>付録B.3.4 : FatFsで、生成した .h ファイルを更新。 |

表 24. 文書改版履歴 （続き）

| 日付 | 版 | STM32CubeMX リリース番号 | 変更内容 |
|---|---|---|---|
| 2015 年 8 月 27 日 | 10 | 4.10 | セクション ：概要の UM1742 を UM1940 に変更。<br>セクション 3.3.2 : コマンドライン･モードでの STM32CubeMX の実行で、コマンドライン･モードでSTM32CubeMX を実行するコマンド･ラインを更新。表 1 : コマンドラインの要約の内容を修正。<br>セクション 4.2 : ［New Project］ウィンドウのボード選択を更新。<br>セクション 5.16: ［Configuration］ビューの概要を更新。セクション 5.14.1: ペリフェラルとミドルウェアの［Configuration］ウィンドウ、セクション 4.4.12 : GPIO設定ウィンドウ、セクション 4.4.13 : DMA 設定ウィンドウを更新。セクション 4.4.11 : ［User Constants］設定ウィンドウを追加。<br>セクション 4.8 : ［Clock Configuration］ビューを更新し、パスのリバース機能を追加。<br>セクション 11.1 : 新しい STM32CubeMX プロジェクトの作成、セクション 11.5 : マイクロコントローラのクロック･ツリーの設定、セクション 11.6 : マイクロコントローラの初期化パラメータの設定、セクション 11.7.2 : ファームウェア･パッケージのダウンロードと C コードの生成、セクション 11.8 : C コード･プロジェクトのビルドと更新を更新。セクション 11.9 : 別のマイクロコントローラへの切替えを追加。<br>セクション 12 : チュートリアル 2 - STM32429I-EVAL 評価 ボードを使用した、SD カード上の FatFs の例を更新し、STM32F429I-EVAL を STM32429I-EVAL. に置き換え。 |
| 2015 年 10 月 16 日 | 11 | 4.11 | 図 8 : ［Embedded Software Packages Manager］ウィンドウ および セクション 3.4.7 : 更新の確認を更新。<br>セクション 4.4.11 : ［User Constants］設定ウィンドウでサポートされる文字列定数。<br>セクション 4.8 : ［Clock Configuration］ビューを更新。<br>セクション 5.2 : ［Power Consumption Calculator］ビューを更新。<br>図 261 : 消費電力計算の例の内容を修正。<br>セクション 13 : チュートリアル 3 - 消費電力計算機能の使用による 組込みアプリケーションの消費電力の最適化などを更新。<br>Eclipse Mars をセクション 3.1.3 : ソフトウェア要件に追加。 |
| 2015 年 12 月 3 日 | 12 | 4.12 | ［Project Settings］メニューでコード生成オプションをサポート。<br>セクション 3.1.3 : ソフトウェア要件を更新。<br>セクション 4.10 : ［Import Project］ウィンドウにプロジェクト設定を追加。図 114 : プロジェクトの自動インポートを更新、プロジェクトの手動インポートの内容を修正、図 115 : プロジェクトの手動インポートおよび図 116 : ［Import Project］メニュー - インポートの試行で発生したエラーを更新、インポート･シーケンスの 3 番目の手順を修正。<br>図 83: エラーが発生しているクロック･ツリー設定ビューを更新。<br>セクション 6.1:HAL ドライバのみによる STM32Cube のコード生成（デフォルト･モード）に mxconstants.h を追加。<br>図 261 : 消費電力計算の例を図 270 : ステップ 10 の最適化に更新。<br>図 271 : 最適化後の消費電力シーケンスの結果を更新。 |

表 24. 文書改版履歴 （続き）

| 日付 | 版 | STM32CubeMX リリース番号 | 変更内容 |
|---|---|---|---|
| 2016 年 2 月 3 日 | 13 | 4.13 | セクション 2.2 : 主な機能を更新。<br>– .ioc ファイルに関連する情報。<br>– クロックツリーの設定<br>– STM32CubeMXと STM32Cube の自動更新。<br>セクション 2.3 : ルールと制限事項で、STM32CubeMX での C コード生成に関連する制限事項を更新。<br>セクション 3.1.1 : サポート対象のオペレーティング･システムとアーキテクチャに Linux を追加。セクション 3.1.3 : ソフトウェア要件で Java Run Time Environment のリリース番号を更新。<br>セクション 3.2.1:STM32CubeMXスタンドアロン･バージョンのインストール、セクション 3.2.3:STM32CubeMX のスタンドアロン･バージョンのアンインストール、セクション 3.3.1: STM32CubeMX プラグインのインストール･パッケージのダウンロードを更新。<br>セクション 3.3.1 : スタンドアロン･アプリケーションとしての STM32CubeMX の実行を更新。<br>セクション 4.9 : [Project Manager] ビュー および セクション 4.12 : [Update Manager] ウィンドウを更新。<br>セクション 5.15.1 : ピンへの信号の固定と信号へのラベルの割当てを更新。<br>セクション 4.4.16 : HAL タイムベース･ソースの設定を追加<br>図 143: [Configuration] ウィンドウで GPIO、DMA、および NVIC の設定を表示するタブ（STM32F4 シリーズ）を更新。<br>出力モードの GPIO 設定に関連する注をセクション 4.4.12 : GPIO設定ウィンドウに追加、図 63 : GPIO 設定ウィンドウ - GPIO の選択を更新。<br>図 130 : [Power Consumption Calculator] のデフォルト･ビュー、図 86: 消費電力シーケンスの構築、図 132 : ステップ管理機能、図 134 : 設定済みのシーケンスに対するトランジション･チェッカー･ オプションの使用 - すべてのトランジションが有効な場合、図 135 : 設定済みのシーケンスに対するトランジション･チェッカー･ オプションの使用 - 1 つ以上のトランジションが無効な場合 を変更。<br>[Import Pinout] ボタンのアイコンをセクション ：ピン配置のインポートに追加。<br>セクション ：すべてのペリフェラルの選択または選択解除を追加。図 140 : シーケンス構築後の [Power Consumption Calculator] ビューの内容を修正。セクション ：シーケンス全体の管理（ロード、保存、および比較）を更新。図 143 : 結果エリアの表記、図 100: ペリフェラルの消費電力に関するツールチップを更新。<br>図 261 : 消費電力計算の例 および 図 263 : [Sequence Table] を更新。<br>セクション 6.3 : カスタム･コードの生成を更新。<br>セクション 11.1 : 新しい STM32CubeMX プロジェクトの作成の図 207 : [MCUs Selection] を表示した [Pinout] ビューおよび図 208 : [MCUs Selection] ウィンドウを非表示にした [Pinout] ビューを更新。<br>セクション 11.6.2 : ペリフェラルの設定を更新。<br>セクション 11.7.1 : プロジェクト･オプションの設定の図 233 : [Project Settings] とツールチェーンの選択および図 234 : [Project Manager] メニュー - [Code Generator] タブ、セクション 11.7.2 : ファームウェア･パッケージのダウンロードと C コードの生成の図 235: ファームウェア･パッケージが見つからないことを示す警告メッセージを更新。 |

表 24. 文書改版履歴 （続き）

| 日付 | 版 | STM32CubeMX リリース番号 | 変更内容 |
|---|---|---|---|
| 2016 年 3 月 15 日 | 14 | 4.14 | STM32CubeMX のリリース番号を 4.14.0 にアップグレード。<br>以前に保存したプロジェクトのインポートとテンプレートからのユーザ･ファイルの生成をセクション 2.2 : 主な機能に追加。<br>セクション 3.1.1 : サポート対象のオペレーティング･システムとアーキテクチャ、セクション 3.2.1 : STM32CubeMXスタンドアロン･バージョンのインストール、セクション 3.2.3 : STM32CubeMX のスタンドアロン･バージョンのアンインストール、セクション 3.4.3: Eclipse IDE での STM32CubeMX プラグインの実行に MacOS を追加。<br>テンプレートからユーザ･ファイルを生成するコマンドラインをセクション 3.3.2 : コマンドライン･モードでの STM32CubeMX の実行に追加。<br>セクション 3.4.2 : アップデータの設定で、新しいライブラリ･インストール･シーケンスを更新。<br>セクション 5.7.3: ［Pinout］メニューの図 107: ［Pinout］メニュー（［Pinout］タブを選択している場合）、図 108: ［Pinout］メニュー（［Pinout］タブを選択していない場合）を更新。<br>表 16: ［Window ］メニューを変更。<br>セクション 5.7: ［Output］ウィンドウを更新。<br>図 101 : [Project Settings] ウィンドウ および セクション 4.9.1 : [Project] タブを更新。<br>セクション 4.4.16 : HAL タイムベース･ソースの設定の図 81 : HAL タイムベースとして SysTick を使用して FreeRTOS を使用しない場合の NVIC 設定および図 82 : FreeRTOS および HAL タイムベースとして SysTick を使用する場合の NVIC 設定を更新。<br>セクション 4.4.11 : ［User Constants］設定ウィンドウの図 54 : ［User Constants］タブおよび図 55 : 生成された main.h からの抜粋を更新。<br>セクション 4.4.12 : GPIO設定ウィンドウ : 図 63 : GPIO 設定ウィンドウ - GPIO の選択、図 64 : ペリフェラル別にグループ化した GPIO 設定、および図 65 : 複数のピンの設定を更新。<br>セクション 4.4.14 : NVIC 設定ウィンドウを更新。 |
| 2016 年 5 月 18 日 | 15 | 4.15 | 同じシリーズのマイクロコントローラどうしに限定されていたプロジェクトのインポート機能の制限を撤廃（セクション 2.2 : 主な機能、セクション 5.7.1: [File] メニュー、およびセクション 4.10 : [Import Project] ウィンドウを参照）。<br>セクション 3.3.2 : コマンドライン･モードでの STM32CubeMX の実行でコマンドラインを更新。<br>表 1 : コマンドラインの要約 : set dest_path <パス> の例同様に、設定コマンドに関連するすべての例の内容を修正。<br>表 13: ［File］メニューの機能に［Load Project］メニューについての注意/注を追加。<br>表 14: ［Project］メニューの［Generate Code］メニューの説明を更新。<br>表 15: ［Pinout］メニューの［Set unused GPIOs］メニューを更新。<br>FreeRTOS が有効な場合の説明をセクション :［NVIC］タブ･ビューを使用した割込みの有効化に追加。<br>セクション 4.4.15 :［FreeRTOS Configuration］パネルを追加。<br>付録B.3.5 : FreeRTOSおよびB.3.6 : LwIPを更新。 |

表 24. 文書改版履歴 （続き）

| 日付 | 版 | STM32CubeMX リリース番号 | 変更内容 |
|---|---|---|---|
| 2016 年 9 月 23 日 | 16 | 4.17 | ドキュメント全体で mxconstants.h を main.h に置き換え。<br>概要、セクション 3.1.1 : サポート対象のオペレーティング･システムとアーキテクチャおよびセクション 3.1.3 : ソフトウェア要件を更新。<br>セクション 3.4.4 : STM32 マイクロコントローラ･パッケージのパッチのインストールを追加。<br>表 2 : [Home] ページのショートカットで、プロジェクトの読込みに関する説明を更新。<br>表 15: [Pinout] メニューのピン配置のクリア機能を更新。<br>セクション 4.9.3 : [Advanced Settings] タブを更新して Low Layer ドライバを追加。<br>表 17: ペリフェラルとミドルウェアの [Configuration] ウィンドウに表示されるボタンとツールチップに [No check] オプションと [Decimal and hexadecimal check] オプションを追加。<br>セクション : [Tasks and Queues] タブおよび図 78 : FreeRTOS によるヒープ使用量を更新。<br>図 63 : GPIO 設定ウィンドウ - GPIO の選択を更新。<br>ドキュメント全体で、PCC を消費電力計算機能に置き換え。<br>セクション 6.2:Low Layer ドライバを使用する STM32Cube コードの生成を追加し、表 20 : LL と HAL の比較 : STM32CubeMX によるソースファイルの生成と表 21 : LL と HAL の比較 : STM32CubeMX による関数と関数呼出しの生成を更新。<br>図 335 : [Pinout] ビュー - RTC の有効化を更新。<br>セクション 14 : チュートリアル 4 - STM32L053xx Nucleo ボードとの UART 通信の例を追加。<br>STM32CubeMX のリリース番号とドキュメント版との対応関係を追記。 |
| 2016 年 11 月 21 日 | 17 | 4.18 | セクション 3.1.3 : ソフトウェア要件で、Windows XP を削除し、Windows 10 を追加。<br>セクション 3.2.3 : STM32CubeMX のスタンドアロン･バージョンのアンインストールを更新。<br>setDriver コマンド･ラインを表 1 : コマンドラインの要約に追加。<br>次のように [List pinout compatible MCUs] 機能を追加。<br>– 表 15 : [Pinout] メニューを更新。<br>– セクション 15 : チュートリアル 5 : 互換性のあるマイクロコントローラへの現在のプロジェクト設定のエクスポートを追加<br>セクション 4.9.1 : [Project] タブおよび図 101 : [Project Settings] ウィンドウにファームウェアの場所を選択するオプションを追加。<br>次のように [Restore Default] 機能を追加。<br>– 表 8 : ペリフェラルとミドルウェアの [Configuration] ウィンドウに表示される ボタンとツールチップを更新。<br>– 図 56 : ペリフェラル･パラメータ設定で使用している定数を更新。 |
| 2017 年 1 月 12 日 | 18 | 4.19 | 同じシリーズに属するマイクロコントローラどうしでのみ可能だったプロジェクトのインポートで、シリーズの制限を撤廃 : 概要、図 114 : プロジェクトの自動インポート、図 115: プロジェクトの手動インポート、図 116: [Import Project] メニュー - インポートの試行で発生したエラー、および図 117 : [Import Project] メニュー - 調整を経て正常に終了したインポート。<br>付録B.3.4 : FatFs、B.3.5 : FreeRTOS、およびB.3.6 : LwIPの内容を修正。付録B.3.7 : Libjpegを追加。 |

表 24. 文書改版履歴 （続き）

| 日付 | 版 | STM32CubeMX リリース番号 | 変更内容 |
|---|---|---|---|
| 2017 年 3 月 2 日 | 19 | 4.20 | 表 17：STM32CubeMX チップ･ビュー - アイコンとカラー･スキーム：<br>– 代替機能例のリストを更新。<br>– ピンへのマッピング機能に対応する例と説明を更新。<br>– 同じピンを共有するアナログ信号の例と説明を追加。<br>図 87: ペリフェラルの［Configuration］ウィンドウ（STM32F4 シリーズ）、図 54：［User Constants］タブ、図 60：ペリフェラル設定で使用されているユーザ定数の削除 - ペリフェラル設定への影響、図 61：ユーザ定数リストを名前で検索、図 62：ユーザ定数リストを値で検索を更新。<br>セクション 5.2.6：SMPS 機能を追加。<br>セクション 6.4：C プロジェクト生成の追加設定を追加。<br>付録 B.3.7：Libjpegで、Libjpeg などのパッケージのリストに STM32CubeF4 を追加。 |
| 2017 年 5 月 5 日 | 20 | 4.21 | セクション 1：STM32Cube の概要の軽微な修正。<br>図 27：［New Project］ウィンドウ -［MCU Selector］および 図 101：［Project Settings］ウィンドウを更新。<br>セクション 4.9.1：［Project］タブでプロジェクトの設定の説明を更新。<br>図 112：［Advanced Settings］ウィンドウを更新。<br>付録B.3.7：Libjpegで、Libjpeg が組み込まれたソフトウェア･パッケージのリストに STM32CubeF2 および STM32CubeH7 を追加。<br>図 369：STM32Cube 組込みソフトウェア･パッケージのルック･アンド･フィールを修正。 |
| 2017 年 7 月 6 日 | 21 | 4.22 | サポート対象の STM32 シリーズのリストに STM32H7 を追加。<br>セクション 3.4：STM32CubeMX による更新の取得にマイクロコントローラのデータとドキュメント更新機能を追加し、図 6：［Updater Settings］ウィンドウを更新。<br>近傍のマイクロコントローラを識別する機能をセクション 4.2：［New Project］ウィンドウに追加、図 27：［New Project］ウィンドウ -［MCU Selector］を更新、図 29：［New Project］ウィンドウ - 指定の機能に近い機能を有するマイクロコントローラの一覧および図 30：［New Project］ウィンドウ - 指定の機能に近い機能を有する マイクロコントローラの一覧表示を追加、図 206: マイクロコントローラの選択を更新。<br>図 39：マイクロコントローラを選択したときの STM32CubeMX のメイン･ウィンドウを更新。<br>表 15: ［Pinout］メニューに［Rotate clockwise］と［Counter clockwise］および上面ビューと下面ビューを追加。<br>セクション 4.1.4：ソーシャル･プラットフォームへのリンクを追加。<br>図 151：ステップごとの SMPS モードの設定を更新。<br>セクション 6.2:Low Layer ドライバを使用する STM32Cube コードの生成を更新。<br>図 233：［Project Settings］とツールチェーンの選択を更新。 |

表 24. 文書改版履歴 （続き）

| 日付 | 版 | STM32CubeMX リリース番号 | 変更内容 |
|---|---|---|---|
| 2017 年 9 月 5 日 | 22 | 4.22.1 | 概要、セクション 5.2 : ［Power Consumption Calculator］ビュー、およびセクション 6.2 : Low Layer ドライバを使用する STM32Cube コードの生成にSTM32L4+ シリーズを追加。<br>Mac OS 上で STM32CubeMX を実行するためのガイドラインをセクション 3.3.1 : スタンドアロン･アプリケーションとしての STM32CubeMX の実行に追加。セクション 3.4.3: Eclipse IDE での STM32CubeMX プラグインの実行から MacOS を削除。<br>セクション 19.10 : Ethernet 設定 : DP83848 や LAN8742A を指定できない場合があるのはなぜですか。を追加 |
| 2017 年 10 月 18 日 | 23 | 4.23 | セクション 1 : 一般情報を追加。<br>セクション 4.2 : ［New Project］ウィンドウで、**［Display close］**ボタンを**［Display similar items］**に名前変更。<br>**［Refresh Data］**メニューと**［Docs & Resources］**メニューをセクション 5.7.5: ［Help］メニューに追加。<br>STM32F2、STM32F4、および STM32F7 の各シリーズをセクション 6.2 : Low Layer ドライバを使用する STM32Cube コードの生成に追加。<br>付録B.3.8 : Mbed TLSを追加。<br>ユーザ･マニュアル第 22 版に対応する STM32CubeMX のリリース番号を更新。 |

表 24. 文書改版履歴 （続き）

| 日付 | 版 | STM32CubeMX リリース番号 | 変更内容 |
|---|---|---|---|
| 2018 年 1 月 16 日 | 24 | 4.24 | “STM32Cube ファームウェア･パッケージ”を“STM32Cube マイクロコントローラ･パッケージ”に置き換え。<br>セクション 1 : STM32Cube の概要を更新。<br>セクション 3.1.1 : サポート対象のオペレーティング･システムとアーキテクチャで、Mac OS を更新。セクション 3.1.3 : ソフトウェア要件で、Eclipse の要件を更新。<br>セクション 3.4 : STM32CubeMX による更新の取得 :<br>– セクションの概要を更新。<br>– 図 13 : ［Connection Parameters］タブ - ［No proxy］を更新。<br>– セクション 3.4.3 : を“STM32 マイクロコントローラ･パッケージのインストール”に名前変更して更新。<br>– セクション 3.4.4 : を“STM32 マイクロコントローラ･パッケージのパッチのインストール”に名前変更。<br>– セクション 3.4.5 : 組込みソフトウェア･パッケージのインストールを追加<br>– セクション 3.4.7 : 更新の確認を更新。<br>図 31 : ［New Project］ウィンドウ - ［Board Selector］を更新。<br>図 40 : ボードを選択したときの STM32CubeMX のメイン･ウィンドウ（ペリフェラルの初期化なし）および概要の文章を更新。<br>図 41 : ボードを選択したときの STM32CubeMX のメイン･ウィンドウ （デフォルト設定でペリフェラルを初期化）および概要の文章を更新。<br>［Select additional software components］メニューを表 14: ［Project］メニューに追加。<br>表 17: ［Help］メニューの［Install new libraries］メニューを［Manage embedded software packages］に名前変更し、対応する説明を更新。<br>セクション 3.4.6 : インストール済みの組込みソフトウェア･パッケージの削除を更新。<br>セクション 4.12 : ［Update Manager］ウィンドウを更新。<br>セクション 4.13 : ［Additional software component selection］ウィンドウを追加。<br>ピン･スタッキング機能を表 17: STM32CubeMX チップ･ビュー - アイコンとカラー･スキームに追加。<br>セクション 6.2 : Low Layer ドライバを使用する STM32Cube コードの生成 : ローレベル･ドライバをサポートする製品シリーズのリストに STM32F0、STM32F3、STM32L0 を追加。<br>セクション 12 : チュートリアル 2 - STM32429I-EVAL 評価 ボードを使用した、SD カード上の FatFs の例 : 図 253 : ボードの選択を更新し、プロジェクトを生成してチュートリアル 2 を実行するシーケンスのステップ 6 を修正。<br>セクション 14 : チュートリアル 4 - STM32L053xx Nucleo ボードとの UART 通信の例 : 図 272 : NUCLEO_L053R8 ボードの選択を更新。<br>セクション 16:チュートリアル 6 ー ユーザのプロジェクトへの組込みソフトウェア･パッケージの追加を追加。<br>付録B.3.9 : TouchSensingおよびB.3.10 : PDM2PCMを追加。<br>セクション 4.4.14 : NVIC 設定ウィンドウ/割込みのデフォルトの初期化シーケンス : 割込みを有効にするコードに対応する色を緑色から黒の太字に変更。 |

表 24. 文書改版履歴 （続き）

| 日付 | 版 | STM32CubeMX リリース番号 | 変更内容 |
|---|---|---|---|
| 2018 年 3 月 7 日 | 25 | 4.25 | 概要、セクション 1 : STM32Cube の概要、セクション 2.3 : ルールと制限事項、セクション 3.2.1 : STM32CubeMXスタンドアロン·バージョンのインストール、セクション 4 : STM32CubeMXのユーザ·インタフェース、セクション 4.9.1 : ［Project］タブ、セクション 5.13.1: ペリフェラルとミドルウェアのツリーのパネルを更新。<br>ドキュメント全体で文章を軽微に編集。<br>表 13: ［File］メニューの機能、表 12 : 電源オーバードライブと HCLK 周波数の関係を更新。<br>図 27 : ［New Project］ウィンドウ - ［MCU Selector］、図 27: マイクロコントローラ·セレクタで有効にしたグラフィック機能による選択、図 101 : ［Project Settings］ウィンドウ、図 106 : 異なるファームウェア保存場所の選択、図 77: STemWin フレームワークの有効化、図 116: グラフィックの ［Configuration］ビュー、図 336 : ［Pinout］ビュー - LSE クロックと HSE クロックの有効化、図 337 : ［Pinout］ビュー - LSE/HSE のクロック周波数の設定を更新。<br>Excel へのエクスポート機能、優先マイクロコントローラの表示機能、セクション 4.4.16: グラフィック·フレームワークとグラフィック·シミュレータを追加。<br>セクション 17: チュートリアル 8 - STemWin グラフィック·フレームワークの使用、セクション 18: チュートリアル 9: STM32CubeMX グラフィック·シミュレータの使用および、それらのサブセクションを追加。<br>セクション B.3.11: グラフィックを追加。 |

表 24. 文書改版履歴 （続き）

| 日付 | 版 | STM32CubeMX リリース番号 | 変更内容 |
|---|---|---|---|
| 2018 年 9 月 5 日 | 26 | 4.27 | 表紙の STM32Cube ロゴを更新。<br>ドキュメント全体で STMCube™ を STM32Cube™ に置き換え。セクション 1 : STM32Cube の概要を更新。<br>図 1 : STM32CubeMX による C コード生成フローの概要を更新。<br>セクション 2.2 : 主な機能を更新して、グラフィック･シミュレータ、CMSIS-Packフォーマットでの組込みソフトウェア･パッケージのサポート、コンテキスト･ヘルプを新機能として追加。<br>セクション 3.4 : のタイトルを“STM32CubeMXによる更新の取得”に変更。[Connection Parameters] タブ - [No proxy] および [Connection Parameters] タブ - ［Use System proxy parameters］の各図を抑制。図 9 : 組込みソフトウェア･パッケージの管理 - ［Help］メニューを更新。<br>セクション 3.4.5 : 組込みソフトウェア･パッケージのインストールで、組込みソフトウェア･パッケージのインストール･シーケンスのステップ 3f を更新し、図 14 : ライセンス契約への同意を追加。<br>セクション 4.2 : [New Project] ウィンドウ : 図 27 : [New Project] ウィンドウ - ［MCU Selector］、図 28 : 優先マイクロコントローラの設定、および図 31 : [New Project] ウィンドウ - ［Board Selector］を更新。<br>セクション 5.7.1: ［File］メニュー : 表 13: ［File］メニューの機能に、新規プロジェクトについての注意/注を追加。図 107: [Pinout] メニュー（[Pinout] タブを選択している場合）、図 108: ［Pinout］メニュー（[Pinout] タブを選択していない場合）を更新。<br>セクション 4.9 : [Project Manager] ビュー :<br>– プロジェクトの保存（ステップ 3）に関連する注を追加。<br>– 図 101 : [Project Settings] ウィンドウを更新。<br>– セクション 4.9.1 : [Project] タブおよび図 106 : 異なるファームウェア保存場所の選択を更新。<br>セクション 4.13.4 : [Component dependencies] パネル、コンテキスト･ヘルプ、セクション 10 : CMSIS-Pack 規格に基づく追加ソフトウェア･ コンポーネントのサポートおよび セクション 17 : チュートリアル 7 ー X-Cube-BLE1 ソフトウェア･パッケージの使用 を追加。 |
| 2018 年 11 月 12 日 | 27 | 4.28 | セクション 3.4.3 : STM32 マイクロコントローラ･パッケージのインストール、セクション 3.4.5 : 組込みソフトウェア･パッケージのインストール、セクション 3.4.6 : インストール済みの組込みソフトウェア･パッケージの削除、セクション 3.4.7 : 更新の確認および含まれる図を更新。<br>セクション 4 : STM32CubeMXのユーザ･インタフェースおよびそのサブセクション、図、表を更新。<br>セクション 10 : CMSIS-Pack 規格に基づく追加ソフトウェア･ コンポーネントのサポート、セクション 11.6.1 ~ 11.6.5、セクション 11.7.1 : プロジェクト･オプションの設定、セクション 11.7.2 : ファームウェア･パッケージのダウンロードと C コードの生成、セクション 11.8 : C コード･プロジェクトのビルドと更新、セクション 11.9 : 別のマイクロコントローラへの切替え、セクション 12 : チュートリアル 2 - STM32429I-EVAL 評価 ボードを使用した、SD カード上の FatFs の例および含まれる図、セクション 15 : チュートリアル 5 : 互換性のあるマイクロコントローラへの現在のプロジェクト設定のエクスポートおよび含まれる図、セクション 16 : チュートリアル 6 ー ユーザのプロジェクトへの組込みソフトウェア･パッケージの追加、セクション 17 : チュートリアル 7 ー X-Cube-BLE1 ソフトウェア･パッケージの使用を更新。 |

表 24. 文書改版履歴 （続き）

| 日付 | 版 | STM32CubeMX リリース番号 | 変更内容 |
|---|---|---|---|
| 2018 年 11 月 12 日 | 27 （続き） | 5.0 | セクション 19: チュートリアル 10: ST-TouchGFX フレームワークの使用およびそのサブセクションを追加。<br>表 21 : LL と HAL の比較 : STM32CubeMX による関数と関数呼出しの生成を更新。<br>旧版の図 164: CMSIS-Pack ソフトウェア･コンポーネントの有効化と設定、図 192: FatFs ペリフェラル･インスタンス、図 213: プロジェクトの［Import Status］、図 254: ユーザ設定としてソフトウェア･コンポーネントの選択を保存、図 268: X-Cube-BLE1 の設定を削除。<br>図 1 : STM32CubeMX による C コード生成フローの概要、図 4 : STM32Cube のインストール･ウィザード、図 7: STM32CubeMX のパースペクティブを閉じる、図 9: Eclipse プラグインの起動、図 10: STM32CubeMX パースペクティブ、図 144 : ペリフェラル全体の消費電力、図 177 : define ステートメントを生成するユーザ定数、図 203 : CMSIS-Pack ソフトウェア･コンポーネントの選択、図 204 : CMSIS-Pack ソフトウェア･コンポーネントの有効化と設定、図 205 : CMSIS-Pack ソフトウェア･コンポーネントから生成されたプロジェクト、図 206 : マイクロコントローラの選択、図 207 : ［MCUs Selection］を表示した［Pinout］ビュー、図 208 : ［MCUs Selection］ウィンドウを非表示にした［Pinout］ビュー、図 210 : タイマ設定、図 211 : 簡単なピン配置設定、図 212 : ［Save Project As］ウィンドウ、図 213 : プロジェクト･レポートの生成 - 新しいプロジェクトの作成、図 214 : プロジェクト･レポートの生成 - プロジェクトを問題なく作成、図 215 : クロック･ツリー･ビュー、図 220 : ［Pinout & Configuration］ビュー、図 221 : 設定パラメータが存在しないペリフェラルとミドルウェアの場合、図 222: タイマ 3 の設定ウィンドウ、図 223: タイマ 3 の設定、図 224 : タイマ 3 の割込みの有効化、図 225 : GPIO 設定のカラー･スキームとツールチップ、図 226 : GPIO モード設定、図 227 : DMA パラメータ設定ウィンドウ、図 228 : ミドルウェアのツールチップ、図 229 : USB ホストの設定、図 229 : USB ホストの設定、図 230 : FatFs over USB モードの有効化、図 231 : FatFs と USB を有効にした［System］ビュー、図 232 : FatFs の define ステートメント、図 233 : ［Project Settings］とツールチェーンの選択、図 234 : ［Project Manager］メニュー - ［Code Generator］タブ、図 235 : ファームウェア･パッケージが見つからないことを示す警告メッセージ、図 237 : ダウンロードするためのアップデータ設定、図 238 : 接続が確立した状態の［Updater Settings］、図 239 : ファームウェア･パッケージのダウンロード、図 240 : ファームウェア･パッケージの解凍、図 241 : C コード生成の完了メッセージ、図 251 : ［Import Project］メニュー、図 281 : ［Project Settings］メニュー、図 291 : 現在のプロジェクトに対して有効にした追加ソフトウェア･コンポーネント、図 292 : パッケージ･ソフトウェア･コンポーネント - 設定できるパラメータがない状態、図 293 : パッケージのチュートリアル - プロジェクト設定、図 296 : 組込みソフトウェア･パッケージ、図 298 : 組込みソフトウェア･パッケージのインストール、図 299 : 新規プロジェクトの開始 - NUCLEO-L053R8 ボードの選択、図 300 : 新規プロジェクトの開始 - すべてのペリフェラルの初期化、図 301 : X-Cube-BLE1 コンポーネントの選択、図 302 : ペリフェラルと GPIO の設定、図 303 : NVIC 割込みの設定、図 304 : X-Cube-BLE1 の有効化、図 304 : X-Cube-BLE1 の有効化、図 305 : SensorDemo プロジェクトの設定、図 312: グラフィック･シミュレータのユーザ･インタフェースを更新。 |

表 24. 文書改版履歴 （続き）

| 日付 | 版 | STM32CubeMX リリース番号 | 変更内容 |
|---|---|---|---|
| 2019 年 2 月 19 日 | 28 | 5.0 | 概要、セクション 1 : STM32Cube の概要、セクション 2.2 : 主な機能、セクション 3.1.3 : ソフトウェア要件、セクション 3.4.3 : STM32 マイクロコントローラ·パッケージのインストール、セクション 4 : STM32CubeMXのユーザ·インタフェース、ピン競合の解決、セクション 4.4.10 : コンポーネント設定パネル、セクション 4.8 : ［Clock Configuration］ビュー、セクション 4.9 : ［Project Manager］ビュー、セクション 4.9.1 : ［Project］タブ、セクション 4.9.3 : ［Advanced Settings］タブ、トランジション·チェッカーの使用、セクション 9.2 : STM32CubeMX のデバイス·ツリー生成、セクション 6.3.2 : ユーザ·テンプレートの保存と選択、.extSettings ファイルの例と生成された結果、セクション 11.6.4 : DMA の設定を更新。<br>セクション 4.5 : STM32MP1 シリーズの［Pinout & Configuration］ビュー、セクション 4.5.2 : ブート段階の設定、セクション 5 : STM32CubeMX の各種ツール、セクション 9 : デバイス·ツリーの生成（STM32MP1 シリーズのみ）、セクション B.3.11 : STM32WPAN BLE/THREAD（STM32WB シリーズのみ）、セクション B.3.12 : OpenAmp と RESMGR_UTILITY（STM32MP1 シリーズおよび STM32H7 デュアルコア製品ライン）およびそのサブセクションを追加。<br>旧版のセクション 1 : 一般情報を削除。<br>表 2 : ［Home］ページのショートカット、表 5 : コンポーネント·リスト、モード·アイコンとカラー·スキーム、表 6 : ピン配置のメニューとショートカットおよび表 9 : ［Clock Configuration］ビューのウィジェットのタイトルを更新。<br>図 101 : ［Project Settings］ウィンドウ、図 102 : プロジェクト·フォルダ、図 106 : 異なるファームウェア保存場所の選択、図 114 : プロジェクトの自動インポート、図 115 : プロジェクトの手動インポート、図 116 : ［Import Project］メニュー - インポートの試行で発生したエラー、図 117 : ［Import Project］メニュー - 調整を経て正常に終了したインポート、図 118 : ［Set unused GPIOs］ウィンドウ、図 119 : ［Reset used GPIOs］ウィンドウ、図 128 : ［About］ウィンドウ、図 198 : STM32CubeMX で生成された DTS – 抜粋 3、図 203 : CMSIS-Pack ソフトウェア·コンポーネントの選択、図 204 : CMSIS-Pack ソフトウェア·コンポーネントの有効化と設定、図 258 : FATFS のチュートリアル - プロジェクト設定、図 259 : C コード生成の完了メッセージを更新。 |
| 2019 年 4 月 16 日 | 29 | 5.1 | 概要、セクション 3.1.3 : ソフトウェア要件、セクション 4.2 : ［New Project］ウィンドウ、指定の機能に近い機能を有するマイクロコントローラの表示、外部クロック·ソース、ピン配置のインポート、すべてのペリフェラルの選択または選択解除、セクション 4.5 : STM32MP1 シリーズの［Pinout & Configuration］ビュー、セクション 4.13 : ［Additional software component selection］ウィンドウ、セクション 5.3.1 : DDR 設定、セクション 6.2 : Low Layer ドライバを使用する STM32Cube コードの生成、BLE の設定、セクション B.3.12 : OpenAmp と RESMGR_UTILITY（STM32MP1 シリーズおよび STM32H7 デュアルコア製品ライン）を更新。<br>セクション 4.2.1 : ［MCU Selector］、セクション 4.2.2 : ［Board Selector］、セクション 4.2.4 : ［Cross Selector］、セクション 4.6 : STM32H7 デュアルコア製品ラインの［Pinout & Configuration］ビュー、セクション 5.2.9 : サンプル機能（STM32MP1 と STM32H7 デュアルコアのみ）、セクション 7 : デュアルコア·マイクロコントローラのコード生成（STM32H7 デュアルコア製品ラインのみ）を追加。<br>旧版のセクション 3.3: STM32CubeMX のプラグイン·バージョンのインストールおよびそのサブセクション、旧版のセクション 3.4.3: Eclipse IDE での STM32CubeMX プラグインの実行を削除。 |

表 24. 文書改版履歴 （続き）

| 日付 | 版 | STM32CubeMX リリース番号 | 変更内容 |
|---|---|---|---|
| 2019 年 4 月 16 日 | 29 （続き） | 5.1 | 表 3 : ［Window］ メニューを更新。<br>図 27 ~ 31、図 112 : ［Advanced Settings］ ウィンドウ、図 130 ~ 137、139 ~ 142、144 ~ 153、図 233 : ［Project Settings］ とツールチェーンの選択、261 ~ 271 を更新。<br>図 24 : ［New Project］ ウィンドウのショートカット、図 85 : STM32MP1 シリーズ : GPIO の割当てオプション、図 367 : リソース·マネージャ : ［Peripherals assignment］ ビューおよび 図 369 : STM32Cube 組込みソフトウェア·パッケージ を追加。 |
| 2019 年 10 月 01 日 | 30 | 5.2 | 概要を更新。セクション 2.2 : 主な機能、セクション 3.3.2 : コマンドライン·モードでの STM32CubeMX の実行、品名選択、セクション 4.13 : ［Additional software component selection］ ウィンドウ、セクション 4.13.1 : ソフトウェア·コンポーネントの概要、セクション 4.13.2 : ［Filters］ パネル、セクション 4.13.3 : ［Packs］ パネル、セクション 4.13.4 : ［Component dependencies］ パネル、セクション 4.13.6 : 追加ソフトウェア·コンポーネントのツリー表示の更新、セクション 5.2 : ［Power Consumption Calculator］ ビュー、セクション 6.2 : Low Layer ドライバを使用する STM32Cube コードの生成。<br>表 1 : コマンドラインの要約、表 6 : ピン配置のメニューとショートカット、表 16 : ［Additional Software］ ウィンドウ - ［Packs］ パネルのアイコン および 表 17 : ［Component dependencies］ パネルのコンテキスト·ヘルプを更新。<br>図 20 : STM32CubeMX ［Home］ ページ、図 125 : 追加のソフトウェア·コンポーネントの選択、図 126 : 追加のソフトウェア·コンポーネント - 更新後のツリー表示、図 203 : CMSIS-Pack ソフトウェア·コンポーネントの選択、および図 301 : X-Cube-BLE1 コンポーネントの選択を更新。<br>セクション 4.4.8 : マルチボンディング·パッケージのピン配置およびセクション 4.13.5 : ［Details and warnings］ パネルを追加。<br>表 15 : ［Additional Software］ ウィンドウ - ［Packs］ パネルに表示される列を追加 |

表 24. 文書改版履歴 （続き）

| 日付 | 版 | STM32CubeMX リリース番号 | 変更内容 |
|---|---|---|---|
| 2019 年 12 月 13 日 | 31 | 5.4 | 概要、セクション 1 : STM32Cube の概要、セクション 4.2 : [New Project] ウィンドウ、新規プロジェクトに使用するマイクロコントローラまたは MPU の選択、およびセクション 11.7.1 : プロジェクト·オプションの設定を更新。<br>セクション 4.7 : [Pinout & Configuration] ビューでのセキュリティの有効化（STM32L5 シリーズと STM32U5 シリーズのみ）およびそのサブセクション、セクション 4.8.2 : クロック·リソースのセキュリティ保護（STM32L5 シリーズのみ）、セクション 8 : TrustZone® を有効にしたコード生成（STM32L5 シリーズのみ）を追加。<br>旧版のセクション 4.4.16: グラフィック·フレームワークとグラフィック·シミュレータ、セクション 17: チュートリアル 8 - STemWin グラフィック·フレームワークの使用、セクション 18: チュートリアル 9: STM32CubeMX グラフィック·シミュレータの使用、セクション 19: チュートリアル 10: ST-TouchGFX フレームワークの使用、セクション B.3.11: グラフィックを削除。<br>ドキュメント全体で文章を軽微に編集。<br>表 1 : コマンドラインの要約を更新。<br>図 48 : [Pinout] ビュー : マルチボンディングを伴うマイクロコントローラ、図 49 : [Pinout] ビュー : 拡張モードによるマルチボンディング、図 85 : STM32MP1 シリーズ : GPIO の割当てオプション、図 101 : [Project Settings] ウィンドウ、図 162 : DDR スイート - ターゲットへの接続、図 163 : DDR スイート - 接続されたターゲット、図 164 : DDR 動作ログ、図 165 : DDR 対話ログ、図 166 : DDR レジスタの読込み、図 167 : U-Boot SPL での DDR テスト·リスト、図 168 : DDR テスト·スイートの結果、図 169 : DDR のテスト履歴、図 170 : DDR のチューニングの前提条件、図 171 : DDR のチューニングの手順、図 172 : ビットのデスキュー、図 173 : [Eye Centering] パネル、図 174 : DDR のチューニング - 設定への保存、図 195 : STM32CubeIDE ツールチェーンのプロジェクト設定、図 233 : [Project Settings] とツールチェーンの選択を更新。<br>図 25 : STM32L5 シリーズの Arm® TrustZone® の有効化を追加。 |

表 24. 文書改版履歴 （続き）

| 日付 | 版 | STM32CubeMX リリース番号 | 変更内容 |
|---|---|---|---|
| 2020 年 7 月 10 日 | 32 | 6.0 | セクション 2.2 : 主な機能、セクション 3.1.1 : サポート対象のオペレーティング･システムとアーキテクチャ、セクション 3.1.3 : ソフトウェア要件、セクション 3.2.1 : STM32CubeMXスタンドアロン･バージョンのインストール、セクション 3.4 : STM32CubeMX による更新の取得、セクション 3.4.5 : 組込みソフトウェア･パッケージのインストール、セクション 4.2 : [New Project] ウィンドウ、Excel へのエクスポート機能、セクション 4.4 : [Pinout & Configuration] ビュー、セクション 4.9.3 : [Advanced Settings] タブ、セクション 18.6 : "Java 8 update 45" 以降のバージョンの JRE をインストールするときに "Java 8 update 45" エラーが発生する理由を追加。<br>セクション 4.2.3 : [Example Selector]、セクション 5.1 : 外部ツール、セクション 19.2 : インターネットにアクセスするログインを変更して以来、一部のソフトウェア･パッケージが使用できなくなったようです。および セクション 19.3 : デュアルコンテキストの製品で、特定のコンテキストでは使用できないペリフェラルまたはミドルウェアがあるのはなぜですか。を追加。<br>旧版のグラフィック機能の基準に基づくマイクロコントローラの選定を削除。<br>表 4 : [Help] メニューのショートカット および 表 14 : [Additional Software] ウィンドウ - フィルタのアイコンを更新。<br>図 20 : STM32CubeMX [Home] ページ、図 24 : [New Project] ウィンドウのショートカット、図 31 : [New Project] ウィンドウ - [Board Selector] 、図 34 : [Cross Selector] - データ更新の前提要件、図 112 : [Advanced Settings] ウィンドウ、図 122 : [Additional Software] ウィンドウ、図 200 : Linux カーネルのデバイス･ツリー生成、図 201 : STM32CubeMX による U-boot 用のデバイス･ツリー生成、図 202 : STM32CubeMX による TF-A 用のデバイス･ツリー生成、図 301 : X-Cube-BLE1 コンポーネントの選択、図 306 : Java コントロール･パネルを更新。 |
| 2020 年 11 月 10 日 | 33 | 6.1 | 概要、セクション 3.1.3 : ソフトウェア要件、セクション 3.4.7 : 更新の確認、セクション 4.13.3 : [Packs] パネル、セクション 5.1 : 外部ツール、セクション 12 : チュートリアル 2 - STM32429I-EVAL 評価 ボードを使用した、SD カード上の FatFs の例セクション 18.6 : "Java 8 update 45" 以降のバージョンの JRE をインストールするときに "Java 8 update 45" エラーが発生する理由を更新。<br>一部のペリフェラルまたはミドルウェアの初期化コードを生成しない選択を追加。<br>表 1 : コマンドラインの要約を更新。<br>図 19 : [Help] メニュー : 更新のチェック、図 20 : STM32CubeMX [Home] ページ、図 112 : [Advanced Settings] ウィンドウ、図 122 : [Additional Software] ウィンドウ、図 129 : ST ツール、図 254 : SDIO ペリフェラルの設定を更新。 |

表 24. 文書改版履歴 （続き）

| 日付 | 版 | STM32CubeMX リリース番号 | 変更内容 |
|---|---|---|---|
| 2021 年 2 月 12 日 | 34 | 6.2 | セクション 3.1.1 : サポート対象のオペレーティング･システムとアーキテクチャ、セクション 3.1.3 : ソフトウェア要件、セクション 3.2.1 : STM32CubeMXスタンドアロン･バージョンのインストール、セクション 3.2.2 : コマンドラインからの STM32CubeMX のインストール、セクション 3.2.3 : STM32CubeMX のスタンドアロン･バージョンのアンインストール、セクション 3.3.2 : コマンドライン･モードでの STM32CubeMX の実行、セクション 3.4.7 : 更新の確認の警告 : 、セクション 4.1 : [Home] ページ、セクション 4.13 : [Additional software component selection] ウィンドウ、セクション 4.13.2 : [Filters] パネル、セクション 4.13.3 : [Packs] パネル、セクション 4.13.4 : [Component dependencies] パネル、セクション 4.13.5 : [Details and warnings] パネル、セクション 12 : チュートリアル 2 - STM32429I-EVAL 評価 ボードを使用した、SD カード上の FatFs の例を更新。<br>表 6 : ピン配置のメニューとショートカットを更新。<br>図 2 : macOS のフル･ディスク･アクセスおよび図 123 : コンポーネントの依存関係のパネルを追加。<br>図 20 : STM32CubeMX [Home] ページ、図 25 : STM32L5 シリーズの Arm® TrustZone® の有効化、図 122 : [Additional Software] ウィンドウを更新。<br>旧版の図 5: コマンドラインからの自動インストール、セクション 18.6: "Java 8 update 45" 以降のバージョンの JRE をインストールするときに "Java 8 update 45" エラーが発生する理由を削除。 |
| 2021 年 6 月 22 日 | 35 | 6.3 | セクション 3.1.1 : サポート対象のオペレーティング･システムとアーキテクチャ、セクション 3.1.3 : ソフトウェア要件、セクション 4.2 : [New Project] ウィンドウ、セクション 4.3 : プロジェクト･ページ、セクション 4.4.5 : [Pinout] ビューの高度な操作方法、セクション 4.7 : [Pinout & Configuration] ビューでのセキュリティの有効化 （STM32L5 シリーズと STM32U5 シリーズのみ）、セクション 12 : チュートリアル 2 - STM32429I-EVAL 評価 ボードを使用した、SD カード上の FatFs の例のコードを更新。<br>図 26 : 選択機能によって表示されるビューの調整 およびセクション 19.1 : STM32CubeMX からのダウンロード中にネットワーク接続エラーが発生します。を追加<br>表 1 : コマンドラインの要約、表 16 : [Additional Software] ウィンドウ - [Packs] パネルのアイコン、表 17 : [Component dependencies] パネルのコンテキスト･ヘルプを更新。<br>図 203 : CMSIS-Pack ソフトウェア･コンポーネントの選択 および 図 301 : X-Cube-BLE1 コンポーネントの選択を更新。 |
| 2021 年 11 月 05 日 | 36 | 6.4 | セクション 2.2 : 主な機能、セクション 3.3.1 : スタンドアロン･アプリケーションとしての STM32CubeMX の実行、セクション 3.4 : STM32CubeMX による更新の取得、セクション 4.2 : [New Project] ウィンドウ、[NVIC] タブ･ビューを使用した割込みの有効化、セクション 4.7 : [Pinout & Configuration] ビューでのセキュリティの有効化 （STM32L5 シリーズと STM32U5 シリーズのみ）、セクション 4.9.1 : [Project] タブ、およびセクション 5.2.7 : BLE および ZigBee のサポート（STM32WB シリーズのみ）を更新。<br>セクション 3.4.1 : プロキシ･サーバのもとでの STM32CubeMX の実行およびセクション 5.2.8 : Sub-GHz のサポート（STM32WL シリーズのみ）を追加。<br>図 69 : NVIC 設定タブ - FreeRTOS が無効な場合を更新。 |

表 24. 文書改版履歴 （続き）

| 日付 | 版 | STM32CubeMX リリース番号 | 変更内容 |
|---|---|---|---|
| 2022 年 2 月 18 日 | 37 | 6.5 | 概要 および セクション 3.1.1 : サポート対象のオペレーティング・システムとアーキテクチャを更新。<br>セクション 18 : LPBAM プロジェクトの作成およびそのサブセクション、セクション 19.11 : どうすれば STM32CubeMX で生成するプロジェクトで MX_DMA_Init の呼び出しランクを固定できますか。を追加。<br>ドキュメント全体で文章を軽微に編集。 |

表 25. 日本語版文書改版履歴

| 日付 | 版 | STM32CubeMX リリース番号 | 変更内容 |
|---|---|---|---|
| 2022 年 6 月 | 1 | | 日本語版　初版発行 |

**重要なお知らせ（よくお読み下さい）**

STMicroelectronics NV およびその子会社（以下、ST）は、ST製品及び本書の内容をいつでも予告なく変更、修正、改善、改定及び改良する権利を留保します。購入される方は、発注前にST製品に関する最新の関連情報を必ず入手してください。ST 製品は、注文請書発行時点で有効なST の販売条件に従って販売されます。

ST製品の選択並びに使用については購入される方が全ての責任を負うものとします。購入される方の製品上の操作や設計に関してST は一切の責任を負いません。

明示又は黙示を問わず、STは本書においていかなる知的財産権の実施権も許諾致しません。

本書で説明されている情報とは異なる条件でST 製品が再販された場合、その製品についてSTが与えたいかなる保証も無効となります。

ST およびST ロゴはSTMicroelectronics の商標です。STの登録商標についてはSTウェブサイトをご覧ください。www.st.com/trademarks その他の製品またはサービスの名称は、それぞれの所有者に帰属します。

本書の情報は本書の以前のバージョンで提供された全ての情報に優先し、これに代わるものです。

この資料は、STMicroelectronics NV 並びにその子会社(以下ST)が英文で記述した資料（以下、「正規英語版資料」）を、皆様のご理解の一助として頂くためにSTマイクロエレクトロニクス㈱が英文から和文へ翻訳して作成したものです。この資料は現行の正規英語版資料の近時の更新に対応していない場合があります。この資料は、あくまでも正規英語版資料をご理解頂くための補助的参考資料のみにご利用下さい。この資料で説明される製品のご検討及びご採用にあたりましては、必ず最新の正規英語版資料を事前にご確認下さい。ST及びSTマイクロエレクトロニクス㈱は、現行の正規英語版資料の更新により製品に関する最新の情報を提供しているにも関わらず、当該英語版資料に対応した更新がなされていないこの資料の情報に基づいて発生した問題や障害などにつきましては如何なる責任も負いません。