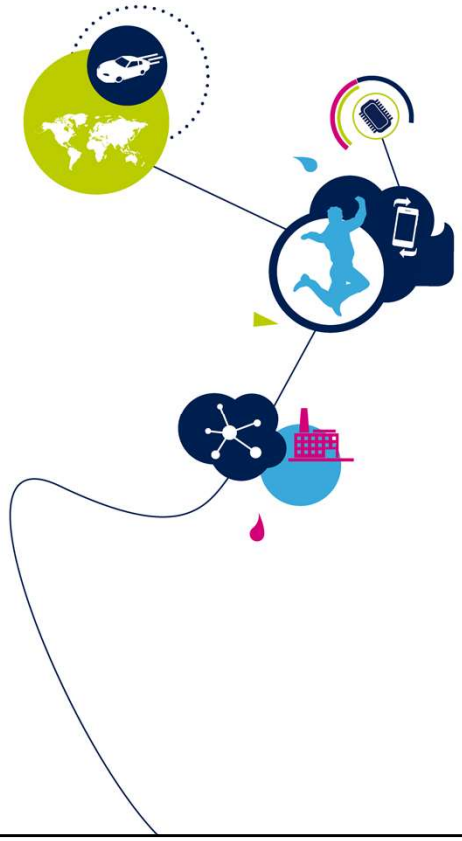
STM32MP1 OpenSTLinux ディストリビューション

OpenSTLinux ディストリビューションの概要
1.0 版



life.augmented

OpenSTLinux ディストリビューションの概要にようこそ。ここでは、このディストリビューションの主なコンポーネントおよびその提供方法について説明します。
このトレーニングでは、OpenSTLinux の基本ビルドシステムである OpenEmbedded の技術的な説明については掘り下げません。このレベルの情報については、コミュニティサイトを参照してください。

OpenSTLinux

- OpenSTLinux は STM32MPU 組込みソフトウェアパッケージ向けのコンセプト
  - コンセプト = 命名 + 関連する中心要素
- 中心要素
  - 標準カーネルインタフェースの使用（独自インタフェースなし）
  - オープンソースソフトウェアの使用
  - コミュニティへのリンク（アップストリーム）
  - 扱いやすさ
- OpenEmbedded ビルドプロセスをサポート
  - Yocto 互換性あり（目標は Yocto サーバでホストされたボードサポートパッケージ(BSP)を準備すること）

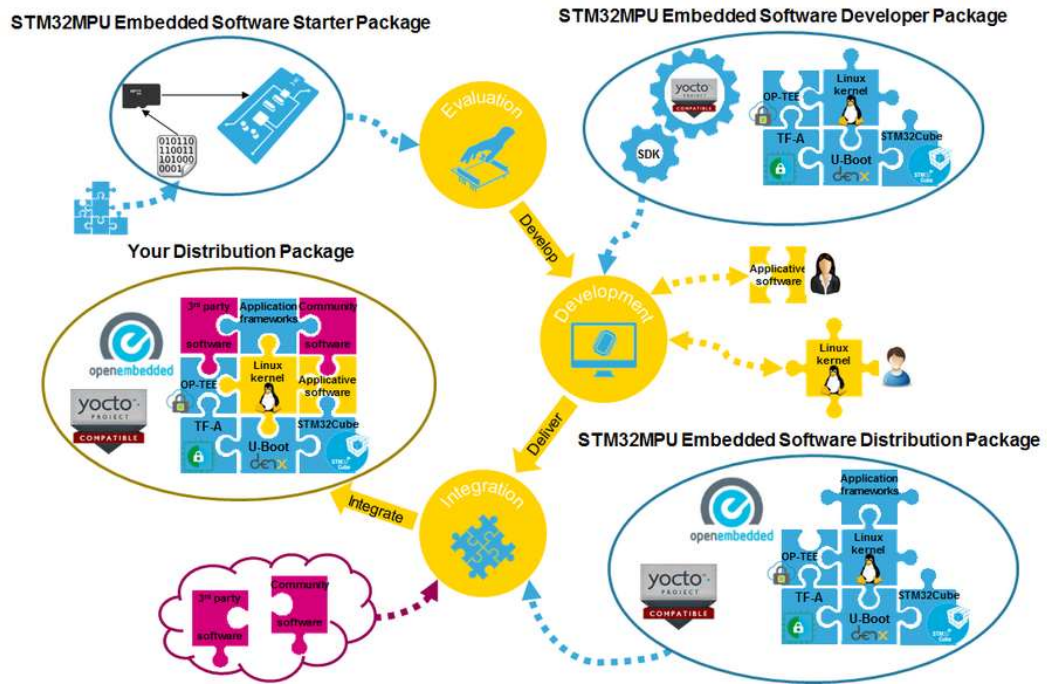
life.augmented

OpenSTLinux は STM32MPU 組込みソフトウェアパッケージ向けのコンセプトです。ここでのコンセプトとは、中心要素や価値に関連した命名を指します。
これらの中心要素には次のものがあります。

- 標準カーネルインタフェースの使用：目標は、最小限の独自パッチで kernel.org のカーネルバージョンを使用することです。
- オープンソースソフトウェアの使用：オープンソースは、OpenSTLinux の強力な基礎です。
- コミュニティへのリンク
  - 目標は、カーネルドライバだけでなく、ブートローダコードやビルドシステムも含め、可能な限り多くのソースコードをアップストリームすることです。
  - コミュニティによって、標準的な適合性、コードの品質、セキュリティ、長期サポート、また言うまでもなく開発コストの点で大きなメリットがもたらされます。
- 扱いやすさ
  - 最後の中心要素は、なんと言っても扱いやすいソリューションに対するニーズです。多様なニーズでカジュアルな開発者から産業デバイスメーカーまで幅広い顧客に対応します。したがって、目標は顧客が求めるものを構築できる拡張性があり、強力で扱いやすいソリューションです。

OpenEmbedded を OpenSTLinux ビルドプロセスとして選択しました。改めて申し上げますが、目標は標準に従い、より多くの要素をアップストリームすることです。STM32MPU BSP（ボードサポートパッケージ）は、Yocto コミュニティにアップストリームされます。

使用可能なパッケージ



使用可能なソフトウェアパッケージには次の 3 種類があります。

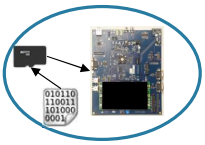
- スタータパッケージ
- 開発者パッケージ
- ディストリビューションパッケージ

これら 3 種類のパッケージはそれぞれ補完し合っており、製品開発サイクルを通じて別々のニーズに対応しています。

- プラットフォーム評価
- アプリケーションやカーネルの開発
- 統合および製品供給

パッケージと使用例

- パッケージ
  - スタータパッケージ　書込み可能イメージ
  - 開発者パッケージ　ソフトウェア開発キット（SDK）および BSP tar 形式ファイル
  - ディストリビューションパッケージ　OpenEmbedded ディストリビューションの全ソース
- 考え方
  - 想定しうるすべての顧客の使用例に対応
    - 発見 -> 試作 -> 新しいハードウェアの起動 -> ソフトウェアの製品化
  - 顧客は OpenSTLinux スタータパッケージでボードを受け取る
    - ボードの機能および性能を評価
  - 既存のアプリケーションを実行する場合 = スタータパッケージ
  - 独自のアプリケーションを開発する場合 = 開発者パッケージ
  - 独自のハードウェアを起動する場合 = 開発者パッケージで試作してディストリビューション
  - ソフトウェアの製品化 = ディストリビューションパッケージ

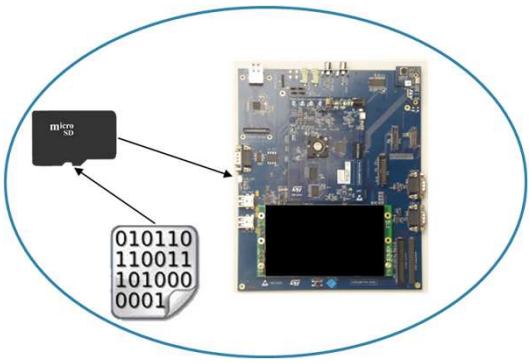


life.augmented

スタータパッケージを使用すれば、迅速かつ容易な方法で STM32 マイクロプロセッサ開発プラットフォームを立ち上げて実行できます。
開発者パッケージを使用すれば、STM32MPU 組込みソフトウェアディストリビューションの一部のソフトウェアの変更や、プラットフォームソフトウェアイメージへのアプリケーションの追加が可能です。
ディストリビューションパッケージを使用すれば、製品化という最終目標に沿って新しいディストリビューションを作成できます。
これら 3 種類のパッケージを組み合わせて、発見から製品化まで想定しうるすべての使用例に対応します。

スタータパッケージ

- STM32MPU 組込みソフトウェアスタータパッケージ
  - HW diversity flashlayout.zip とともに格納された ST image = Yocto ベースイメージ



life.augmented

STM32 マイクロプロセッサ開発プラットフォーム（STM32MP157C-EV1 評価ボードなど）および microSD カード（このプラットフォーム向けに配布されるソフトウェアイメージを保存）があれば、プラットフォームの検証が可能です。統合開発環境（IDE）やソフトウェア開発キット（SDK）は必要ありません。Linux® を実行する「マイクロ PC」として開発プラットフォームを使用するだけです。
このソフトウェアイメージには、STM32Cube MPU パッケージに関する例として提供されるファームウェアが含まれています（検討する開発プラットフォームに Arm® Cortex®-M プロセッサが含まれている場合）。

スタータパッケージの要点

- 顧客による使用可能なバイナリ：Yocto/OpenEmbedded イメージ（Weston）
  - プラットフォームの機能を実行
  - microSD カードでの使用またはボードへの直接書込みが可能
  - この時点では Yocto スタータパッケージは開発キットの前提条件であるのみ
- 成果物
  - 特定の Flash レイアウトでのバイナリセット
  - SD カードの元イメージを生成するスクリプト
  - microSD カード使用可能イメージ（別名 stimg）の全組み合わせ（バイナリ x Flash レイアウト）は設定が多すぎるためデフォルトで提供なし

life.augmented

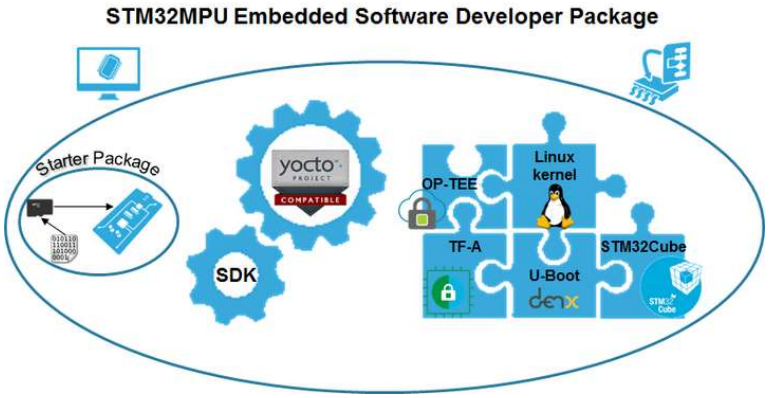
このパッケージで対応できる主な使用例にはどのようなものがあるでしょうか。

- STM32 マイクロプロセッサデバイスと開発プラットフォームの評価（内部ペリフェラルおよび性能など）
- 各 STM32MPU 組込みソフトウェアディストリビューションのスタータパッケージ内で提供される Cube M4 ファームウェアプロジェクトの実行（Arm Cortex-M プロセッサで使用可能なファームウェアなど）
- 異なる言語（シェル、Python など）での開発プラットフォームにおける簡単なユーザ事例の直接的な開発

これを行うために、スタータパッケージには使用可能バイナリが含まれています。これらのバイナリは、小型で高速なグラフィカルコンポジッタの Weston を使用する ST OpenEmbedded イメージで生成されました。
バイナリは Flash ローダツールを使用して microSD カードにロードできます。
また、スクリプトからカスタムの microSD カードコンテンツも生成できます。ソフトウェア設定を生成するための組み合わせが膨大になるため、最も関連するセットのみを提供します。

開発者パッケージ

- st-image-weston をベースとした Yocto =
  - SDK（ツールチェーン + 同梱物）
  - ソース（コミュニティ tar 形式ファイル + ST パッチ + ST 設定）
    - カーネル
    - ブート（U-Boot、ATF）
  - STM32MP1-M4 Cube



life.augmented

開発者パッケージはスタータパッケージに加えて次のものが提供されます。

- ホスト PC でのクロス開発用のソフトウェア開発キット（SDK）
- ソースコードでの OpenSTLinux ディストリビューションのソフトウェア：
  - U-Boot
  - Trusted Firmware-A（TF-A）
  - Linux® カーネル
  - オプションでOpen source Trusted Execution Environment（OP-TEE）

- ソースコード のSTM32Cube MPU パッケージ（検討する開発プラットフォームに Arm Cortex-M プロセッサが含まれている場合）
- 初期化コードジェネレータ（STM32CubeMX）。このツールによって次のことが可能になります。
  - OpenSTLinux ディストリビューションのデバイスツリーの生成
  - STM32Cube MPU パッケージのペリフェラル初期化 C コードと IDE プロジェクト作成ファイルの生成
- Eclipse ベースの統合開発環境（IDE）- STM32-CoPro-MPU
  - STM32Cube ファームウェアプロジェクトは STM32-CoPro-MPU Eclipse プラグインでサポートされます。
  - STM32CubeMX で生成されたプロジェクトファイルはこの IDE で直接開くことができ、Cube ファームウェア開発を継続できます。
  - すべての操作はコマンドラインで実行できるため、IDE は OpenSTLinux 開発のオプションです。

開発者パッケージの要点

8

- 開発者パッケージ = SDK（ソフトウェア開発キット） + ソースコード
  - スタータパッケージイメージの使用
  - Weston イメージに基づく Yocto/OpenEmbedded SDK のみ提供
  - ソースコードの提供
    - カーネル、U-Boot、ATF、Optee（オプション）、STM32Cube
  - 事前コンパイル済みツールチェーン
- プロジェクトステージに応じたリリースモード
  - アルファ顧客
    - ソースコード（コミュニティの tar 形式ファイル） + パッチ
  - マスマーケット
    - ソースコード（コミュニティの tar 形式ファイル） + パッチ
    - Git（ST github） = コミュニティコンテンツ + アップストリーム保留中の全パッチ

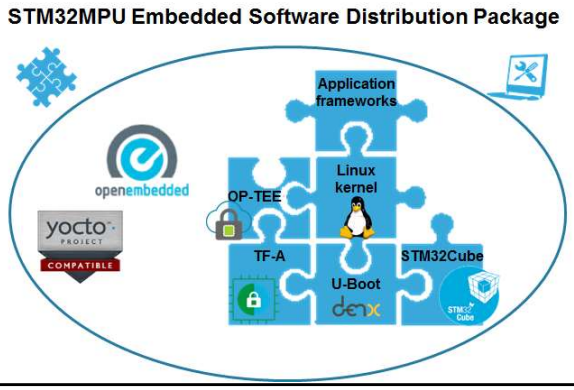
life.augmented

このパッケージで対応できる主な使用例にはどのようなものがあるでしょうか。

- ソースコードで配布されるソフトウェアの一部の変更および調整
  - コードの変更
  - 効率的な開発サイクル（コンパイル、ターゲットへのプッシュ、デバッグ）
  - 設定オプションの調整（Linux® カーネルの場合など）
  - マイクロプロセッサデバイスおよび開発プラットフォームのデバイスツリーファイルの適応
- Arm Cortex-M プロセッサで実行するファームウェアの開発
- 独自の Linux® アプリケーションや Linux® アプリケーションフレームワークの追加
- 独自の信頼できるアプリケーション（TA）の追加（OP-TEE がソフトウェアリリースの一部である場合）
- 独自の U-Boot アプリケーションの追加

ディストリビューションパッケージ

- Yocto ベース -（Git コミュニティの ST レイヤ + ST パッチ）=
  - oe-manifest
  - meta-st-stm32mp
  - meta-st-openstlinux
  - STM32MP1-M4 Cube
  - meta-st-custo（顧客ごとの付加に応じてカスタマイズ）



このパッケージを使用すれば、製品化という最終目標に沿って新しい Linux ディストリビューションを作成できます。
これには、STM32MPU 組込みソフトウェアディストリビューションの全ソフトウェアのソースコード（アプリケーションフレームワークを含む）、および OpenEmbedded に基づくビルドフレームワーク（別名ディストリビューションビルダ）が含まれています。このパッケージを動作させるには、ホスト PC が必要です（Linux® のホスト PC を強く推奨）。
ここでは一部 OpenEmbedded の基本用語を使用しています（レイヤ、メタデータ、付加）。これらの用語については、以降のスライドで説明します。

ディストリビューションパッケージの要点

- ディストリビューションパッケージ
  - Yocto/OpenEmbedded 環境
    - OpenSTLinux BSP（カーネル、ST ドライバ）
    - OpenSTLinux アプリケーションフレームワーク（Weston、GStreamer など）
  - 開発者 git を指定する OpenEmbedded レシピ（SRC_URI）
  - すべてコンパイル（ツールチェーン、カーネル、イメージ）
- カーネルバージョン
  - アルファステージ
    - 最新の安定したカーネル + アップストリーム保留中の全パッチ
  - マスマーケット
    - アルファステージ（例として提供）
    - LTS カーネル + アップストリーム保留中の全パッチ（デフォルト設定）

life.augmented

このパッケージで対応できる主な使用例にはどのようなものがあるでしょうか。

- 独自の Linux® ディストリビューションの作成
  - システム設定オプション（メモリサイズやデバッグオプションなど）の調整
  - チーム内での全開発内容（ソフトウェアベースライン）の共有
  - 最終版ソフトウェアの製品化準備
- このディストリビューションで次のソフトウェアを統合：
  - 開発者パッケージで開発した内容
  - オープンソースコミュニティまたはサードパーティ由来のソフトウェア
  - 独自のディストリビューションへの Cube M4 ファームウェアプロジェクトの統合（バイナリまたはソースコード）
- 独自のスタータパッケージイメージの生成
- 独自の開発者パッケージ SDK の生成

一般的に、ディストリビューションパッケージを使用して、プロジェクト Linux® ディストリビューションの作成、プロジェクト開発者パッケージの生成を行うことはほとんどありません。ほとんどの開発者は、軽量かつ効率的な環境（短い開発サイクル）であるため、開発ではこの開発者パッケージを使用します。

OpenEmbedded

- 2010 年に Linux Foundation が着手し、現在もその一員である Richard Purdie が管理しているプロジェクト
- Linux ベースのクロスコンパイルフレームワーク
- オープンソース（ただし、独自コードのビルドに使用可能）
- ソフトウェア設定管理用の git がベース

life.augmented

OpenEmbedded は、2010 年に Linux Foundation が着手し、現在もその一員である Richard Purdie が管理しているプロジェクトです。
OpenEmbedded は、Linux ベースのクロスコンパイルフレームワークです。クロスコンパイルとは、あるハードウェアアーキテクチャとオペレーティングシステムを搭載したマシンで、別のハードウェアアーキテクチャやオペレーティングシステム用にプログラムをコンパイルできることです。
OpenEmbedded はオープンソースですが、独自コードのビルドに使用できます。
これは、ソフトウェア設定管理用の git に基づいています。

OpenEmbedded の開発エコシステム

- Yocto、Poky、OpenEmbedded について話す際に、混乱が生じる場合があります。
  - OpenEmbedded:
    - 組込み Linux 向けのビルドフレームワーク
    - コミュニティで管理
    - Poky のソースバージョン
    - 主に ARM プラットフォーム向けに強化されたセットアップ
  - Yocto
    - OpenEmbedded ビルドシステムを使用するプロジェクト
  - Poky
    - Poky は Yocto Project のリファレンスシステムであり、Yocto Project のツールと、動作例一式として機能するメタデータをまとめたものです。Poky は OpenEmbedded Core を使用します。
    - Poky は Intel によって管理されています。セットアップは主に Intel プラットフォーム向けに強化されています。
- 一部のプロジェクトは Yocto ベースで動作し、その他は Poky ベースで動作しますが、最終的にはすべて互換性があります。

life.augmented

Yocto、Poky、OpenEmbedded について話す際に、混乱が生じる場合があります。

- OpenEmbedded:
  - 組込み Linux 向けのビルドフレームワークです。
  - コミュニティで管理されています。
  - Poky のソースバージョンです。
  - 主に ARM プラットフォーム向けに強化されています。
- Yocto
  - OpenEmbedded ビルドシステムを使用するプロジェクトです。
- Poky
  - Yocto Project のリファレンスシステムであり、Yocto Project のツールと、動作例一式として機能するメタデータをまとめたものです。Poky は OpenEmbedded Core を使用します。
  - Poky は Intel によって管理されています。このセットアップは主に Intel プラットフォーム向けに強化されています。

一部のプロジェクトは Yocto ベースで動作し、その他は Poky ベースで動作しますが、最終的にはすべて互換性があります。

OpenEmbedded の操作

- ソースコードのダウンロード
- パッチ適用
- クロスコンパイル
- パッケージ管理

OpenEmbedded は強力ですが複雑なビルドフレームワークと考えられます。わかりやすくするために、コンパイル中の OpenEmbedded の主な操作をまとめます。
以下のほとんどの操作が、レシピと呼ばれるファイルで定義されています。

- さまざまなレポジトリからソースコードをダウンロードします。
- パッチをソースコードに適用します。
- この変更済みのソースコードをクロスコンパイルして、実効可能イメージを準備します。
- 最終的に、異なる可能な形式でパッケージを作成します。OpenEmbedded フレームワークにより、これらのパッケージをPCの Ubuntu や Debian とまったく同じ方法でインストールできます。例として、PC と同様にターゲット上で apt-get install も実行できます。OpenEmbedded は Linux ディストリビューションのメーカーであると覚えてください。

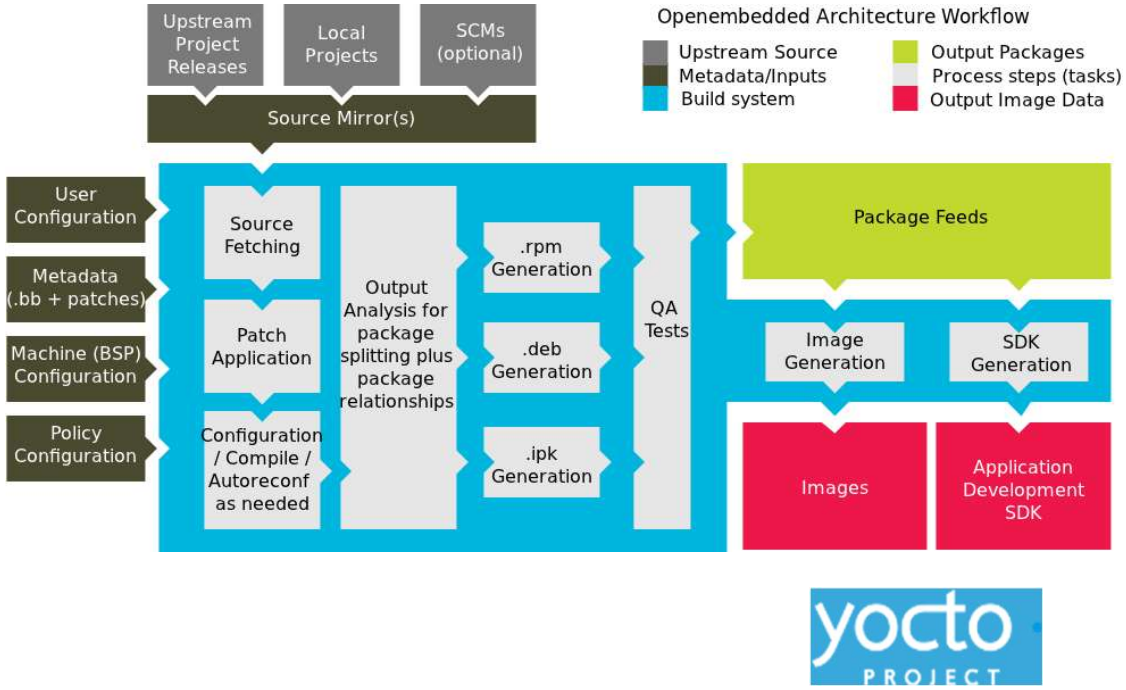
OpenEmbedded の生成物

- バイナリパッケージ
- Linux ベースシステムイメージ
- ツールチェーン
- SDK（ソフトウェア開発キット）

実際、OpenEmbedded はバイナリー式を生成します。

- 先に説明したバイナリパッケージを含みます。
- Linux ベースシステムイメージ：カーネルイメージ、デバイスツリー、コンパイルした Linux システムをデバイスで実行するのに必要なものすべてを含みます。
- ツールチェーン：ソースコードをクロスコンパイルする前に OpenEmbedded がツールチェーンをコンパイルします。これがコンパイルに膨大な時間がかかる可能性がある理由です。しかし、非常に強力です。コンパイルする前にダウンロードする適切なツールチェーンを見つけなければならない場合を想像してみましょう。
- ソフトウェア開発キット：ツールチェーンを含むこのキットは、すぐに使用可能な SDK です。シンプルな開発の場合、Linux ディストリビューションのフレームが定義されると、作業する上で SDK は最高のツールとなります。

コンパイルフロー



Yocto Project サイトから引用したこの図に、コンパイルフローがまとめられています。
ソースコードは、別々の場所からダウンロードされます。パッチが適用され、ソースコードがクロスコンパイルされ、パッケージが生成され、イメージが生成され、SDK が生成されます。
さらに、QA テストを実行できます。

コンセプト：レイヤ、ディストリビューション、マシン、イメージ（1/2）

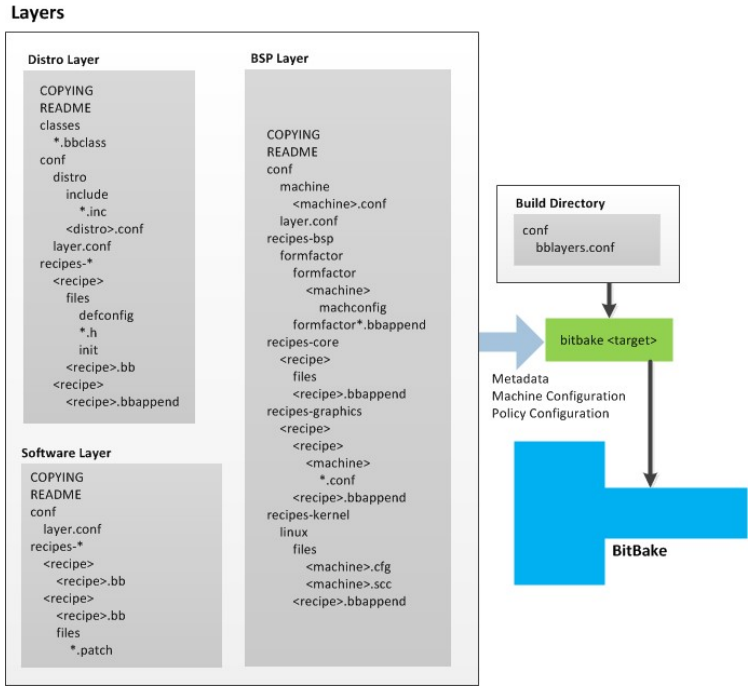
- レイヤ：
  - ソフトウェア、マシン、ディストリビューションメタデータを提供
- マシン：
  - マシンメタデータ（ハードウェアの設定）
- ディストリビューション：
  - ディストリビューションメタデータ（ソフトウェアの設定）
  - ハードウェアに関連する機能：WiFi、ネットワーク、Bluetooth など
  - ソフトウェアに関連する機能：初期化、ソフトウェアの指定の実装、SDK 名
- イメージ：
  - ボードに書き込めるイメージ上に存在するソフトウェアのセット
  - 一部のソフトウェアはディストリビューション機能スイッチによって条件付け可能

もう少しコンセプトを説明します。改めて、基本用語を挙げていきます。
レイヤ、マシン、ディストリビューション、イメージは、OpenEmbedded の世界では常用される言葉です。実際、これらのコンテンツは限られた人員（通常はインテグレータ）によって定義されます。
レイヤは、フォルダとみなすことができ、名前が « meta » で始まります。これには、レシピと設定ファイルが含まれています。レイヤは全体的なセットを記述する情報を含む形で整理されています。たとえば、BSP はレイヤで定義されます。QT などの他のフレームワークも特定のレイヤに追加できます。

一方、その結果作成される製品ソフトウェアは、それぞれが異なるレベルの情報を有するマシン、ディストリビューション、イメージの組み合わせで定義されます。どういうことでしょうか。1 か所ですべての情報を探すほうが簡単なのではないでしょうか。確かにそうかもしれませんが、ソフトウェアをこれらの 3 つのレベルに分けることで、多様に組み合わせることができます。
ソフトウェアコンテンツは好きですが、別のプラットフォーム用にコンパイルしたいとも考えているとします。簡単にできます。
ソフトウェアパッケージをディストリビューションに追加したいと考えているとします。これも簡単です。
ソフトウェアを別々に設定したいと考えているとします。これも簡単です。

レイヤ：
　　ソフトウェア、マシン、ディストリビューションメタデータを提供
マシン：
　　マシンメタデータ（ハードウェアの設定）
ディストリビューション：
　　ディストリビューションメタデータ（ソフトウェアの設定）
　　ハードウェアに関連する機能：WiFi、ネットワーク、Bluetooth など
　　ソフトウェアに関連する機能：初期化、ソフトウェアの指定の実装、SDK 名
イメージ：
　　ボードに書き込めるイメージ上に存在するソフトウェアのセット
　　一部のソフトウェアはディストリビューション機能スイッチによって条件付け可能

コンセプト：レイヤ、ディストリビューション、マシン、イメージ（2/2）



これは、レイヤと想定するコンテンツを示した図です。
各フォルダは、設定ファイルとレシピを持つ同様の構造になっています。ディストリビューション、イメージ（図のソフトウェアレイヤ）、マシン（図の BSP レイヤ）をアセンブルするには、BitBake コマンドを使用します。これは OpenEmbedded フレームワークでコンパイルするためのコマンドです。
ディストリビューション、マシン、イメージのレイヤの組み合わせは、OpenEmbedded 環境が初期化されるときに決まります。

レイヤとイメージ

これらのコンセプトすべてを、OpenSTLinux 向けに実際に実装する内容につなげてみましょう。
上の図は ST が作成したレイヤを示しています。

meta-st-stm32mp には BSP 定義が含まれます。正確には 3 つの BSP です。汎用的な stm32mp1 は、1 回のコンパイルで選択したすべての組み合わせを生成します。たとえば、テスト用に役立ちます。さらに、2 つの専用マシン、stm32mp1-eval と stm32mp1-disco（EVAL ボードと DISCO ボード）があります。

次に、meta-st-openstlinux です。ここでは、ディストリビューションとイメージの両方の定義をホストします。

白文字は公式に提供されているものです。イタリック体の緑色の文字は、公式サポートがない例として提供される設定です。

役立つリンク

- OpenEmbedded Wiki: http://www.openembedded.org/wiki/Main_Page
- Yocto project: https://www.yoctoproject.org/

こちらは、OpenSTLinux ディストリビューションの配置に役立つリンクです。