参考資料

User's Guide

MSPM0 マイコン での Zephyr の導入

TEXAS INSTRUMENTS

概要

このガイドは、MSPM0 マイコン (MCU) を使用してオープンソースの Zephyr リアルタイム オペレーティング システム (RTOS) を使用するユーザー向けに、高レベルのエントリ ポイントを提供します。テキサス インスツルメンツ MSPM0 で利用可能なソフトウェア リソースを紹介し、MSPM0 デバイスの機能について説明し、OpenOCD、VSCode デバッグ、GNU デバッグ ツールの概要を示します。

このガイドの内容:

- Zephyr とは
- テキサス インスツルメンツの MSPM0 マイコンで Zephyr を使用する理由
- Zephyr 環境の設定方法
- Zephyr のアップストリーム リポジトリとテキサス インスツルメンツが管理しているダウンストリーム リポジトリの相違点
- コマンド ラインと Visual Studio Code を使用した Zephyr プロジェクトのデバッグの概要。
- MSPM0 LaunchPad™ でサンプルを実行する方法

目次

商標

1 Zephyr とは

1.1 リアルタイム オペレーティング システム (RTOS)

リアルタイム オペレーティング システムは、組み込みアプリケーションに適したコンピュータ オペレーティング システムの一種です。RTOS は、コマンドを実行する際の時間的制約を考慮して、オーバーヘッドが低く、小型で確定的になるように設計されています。

RTOS は、Linux などのオペレーティング システムよりも大幅に軽量でありながら、マルチタスク、スケジューリング、メモリ処理などの利点が多くあります。これらの機能により、RTOS は複雑なアプリケーションに最適でありながら、メモリ、電力、および計算能力にも優れています。

マルチタスク、スケジューリング、メモリ アーキテクチャの複雑さを理解するための優れたリソースは数多くありますが、ここでは取り上げません。ただし、Zephyr を使用して開発を開始し、動作させるには、以下の RTOS の概念を基本的に理解しておくことが重要です。

- マルチタスク / スレッド化: オペレーティング システムはカーネルを採用しています。これは、複数の「ユーザー」またはプログラムがプロセッサのコンピューティング リソースやメモリ リソースにアクセスできるようにするコア プロセスです。この同時処理はスレッド化によって実行され、実行中の各プログラムにスレッド (または Zephyr 内のタスク) が割り当てられます。これらのタスクは、スケジューリングと組み合わせると「同時に」実行できます。
- スケジューリング: 同時実行の錯覚を可能にするカーネルのコア コンポーネントはスケジューリングです。スケジューラは、タスクの実行中にタスクを複数回一時停止、再開、および切り換える機能を備えたオペレーティング システム内の基本的なコード ブロックです。つまり、あるタスクにダウンタイムがある場合、またはアクティブにコードを実行していない場合、そのタスクが完了するか、より優先度の高いタスクが開始されるまで、別のタスクがコンピューティングとメモリを制御できます。
- リアルタイムのオペレーション: 多くのアプリケーションでは、外部刺激に対するリアルタイムの応答が求められます。これが、Zephyr のような RTOS と、Linux のような一般的な非リアルタイム オペレーティング システムとの違いです。タスクが作成されると、割り込みと同様の優先度が割り当てられます。時間に制約のある応答を必要とするタスクには、時間に制約のないタスクよりも高い優先度が割り当てられます。これにより、作成後すぐにコンピューティングを使用できます。

1.2 オープンソース RTOS オプションとしての Zephyr

Zephyr はオープンソース RTOS であり、最近 10 年間にわたって組込み開発分野で人気が高まっています。Zephyr は、コミュニティで管理される RTOS であり、ライセンスのロイヤリティを必要としません。

2 MSPM0 における Zephyr の利点

2.1 ベア メタルに対する利点

通常、単一のアプリケーションで複数のプロセスまたはタスクを実行するのは非常に困難な場合があり、ソフトウェア エンジニアによる複雑なメモリ管理が必要です。しかし、RTOS はカーネルを介してこのプロセスを簡素化します。一般的なベアメタル コードよりもオーバーヘッドが高くなりますが、Zephyr のような RTOS カーネルは、オーバーヘッドが比較的低く、コンパイル時にメモリが最適化されるため、メモリを重視するアプリケーションに最適です。Zephyr のもう一つの重要な検討事項は、コードの検証とセキュリティの容易さです。Zephyr 内のすべてのアプリケーション コードはハードウェア アブストラクション レイヤ上で動作しているため、検証がはるかに簡単になります。ソフトウェア開発者は、コードの基盤が健全であることを認識しており、アプリケーション コードのみに気を配ればよく、Zephyr のカーネルが提供する追加のデバッグ機能やカーネル機能により、デバッグが容易になる可能性があります。

2.2 MSPM0 の検討事項

Arm Cortex-M0+ マイコンの MSPM0 ファミリは、電力、サイズ、コストの面で優れています。このため、MSPM0 はアナログ、通信、ハウスキーピング機能を 1 つのパッケージに統合でき、ほとんどすべてのアプリケーションに適用できます。Zephyr の軽量で時間に制約のある特性に加えて、MSPM0 はさまざまな組込みプロジェクトに最適なオペレーティング システムになります。

Zephyr には以下の特性があり、電力、メモリ、MSPM0 の性能を最大限に引き出すのに役立ちます。

- 電源:
  - アイドル動作はティックレスで、低消費電力モードでは省エネルギーを実現します
  - タスクが実行されていないときは、マイコンはディープスリープ状態に移行できます
- メモリ:
  - Zephyr のカーネルは、数 KB の ROM と RAM に収容でき、MSPM0 のより小さなメモリ リソースでも適切に動作します
- 性能:
  - プリエンプティブ スケジューラは、予測可能なタスク レイテンシを提供します
  - 厳格なリアルタイム性能により、センサ フュージョンやモーター制御などのアプリケーションに最適です
  - タスクは必要なときにのみ実行されるため、スレッド化 / タスク処理により CPU の能力を最大限に活用できます

全体的に、Zephyr は、デバイスの効率性と確定的な動作を維持しながらマルチタスクを追加するスケーラブルな標準ベースの RTOS 環境を実現することで、MSPM0 ファミリのデバイスの低消費電力とコスト効率の利点を補完します。

2.3 一般的なアプリケーション

多くの場合、エンジニアは医療、産業、車載、ウェアラブルなど、時間の制約が厳しいアプリケーションで RTOS を使用します。Zephyr の利点を確認すると、このような状況でどのように役立つかが明確になります。時間に制約のある特性は、ユーザーの安全やアプリケーションの一貫性を保つために即座の応答が重要な医療、産業、車載アプリケーションに適しています。

Zephyr スタックには、TI とサードパーティーの多数のセンサが統合されており、設計者はこれらのセンサを手動でコーディングすることなく、包括的なシステムやプロトタイプを簡単に開発できます。また、Zephyr はオープンソースなので、更新されたセンサは頻繁に追加されます。バッテリ チャージャ、コネクティビティ ソリューション、環境センサなどが利用できるため、すぐに接続できます。

2.4 セキュリティの概要

これらすべてのアプリケーションにおいて、アプリケーションに関係なくセキュリティが重要な問題となっています。オペレーティング システムは抽象化の層を増やしてアプリケーション セキュリティの自然な層を提供しますが、Zephyr はシステム全体のセキュリティを改善するために追加の手順を実行します。組込みアプリケーションでのセキュリティの重要な要素はメモリの保護であり、ベア メタル アプリケーションでは、これは主にソフトウェア開発者に任されています。一方、Zephyr は分離性を提供するため、小さなコーディング ミスがシステム全体に影響を及ぼす可能性は低くなります。ただし、これは Zephyr が攻撃から完全に保護することを意味するものではなく、開発中は『安全なコーディング ガイドライン』に従って、Zephyr 内の攻撃ベクトルを最小限に抑えることが重要です。これは Zephyr のセキュリティ機能の概要ですが、Zephyr のセキュリティ機能の詳細については、Zephyr のドキュメントを参照してください。

3 Zephyr 開発環境の設定方法

3.1 一般的な設定

Zephyr のインストール方法は、ユーザーのオペレーティング システムによって異なります。このドキュメントでは、インストールは Ubuntu 22.04 LTS に表示されます。Windows および MacOS の場合は、ガイドを参照してください。ただし、インストールの大部分は Python と west を使用して実行されるため、オペレーティング システム間でのインストール プロセスの違いは大幅に少なくなります。

3.1.1 インストールの依存関係

本文書の作成時点では、Zephyr をインストールするための要件は 4 つあります。

- CMake バージョン 3.20.5 以降
- Python バージョン 3.10 以降
- デバイスツリー コンパイラ バージョン 1.4.6 以降
- Git

3.1.2 Python と Zephyr のセットアップ

Zephyr には Python との依存関係がかなりあり、これらを常に最新の状態に保つことが重要です。初めてセットアップする場合は、以下の手順を実行します。

1. 以下のコマンドを使用して、Ubuntu に依存関係をインストールします。

```
sudo apt install --no-install-recommends git cmake ninja-build gperf \
  ccache dfu-util device-tree-compiler wget python3-dev python3-venv python3-tk \
  xz-utils file make gcc gcc-multilib g++-multilib libsdl2-dev libmagic1
```

2. ホーム パスに新しい仮想環境を作成します。これは、すべての Zephyr の新規プロジェクトで実行する必要があります。この例では「zephyrproject」が使用されますが、将来のコマンドが更新されていれば、この名前は自由に変更できます。

```
python -m venv ~/zephyrproject/.venv #create script for virtual environment
source ~/zephyrproject/.venv/bin/activate #activate virtual environment
```

3. zephyrproject の west 側を初期化し、cd を挿入します。West update を実行します。多くのパッケージをインストールする必要があるため、これには約 15 分かかります。

```
pip install wheel
pip install west
west init ~/zephyrproject
cd ~/zephyrproject
west update
west zephyr-export
west packages pip --install
```

4. 最後に、Zephyr SDK をインストールします。

```
deactivate
cd ~
git clone https://github.com/openocd-org/openocd.git
sudo apt install libusb-1.0-0-dev libhidapi-dev
```

3.1.3 OpenOCD

Zephyr には、TI の LaunchPad をサポートしていない、Zephyr 0.17.4 として入手できる OpenOCD のバージョンが付属しています。そのため、TI MSP マイコンをサポートする新しいバージョンをインストールすることが重要です。

ここまでガイドに従ってきた場合は、ホーム ディレクトリに OpenOCD 用の新しいフォルダを作成する前に、まずこの時点まで使用していた仮想環境を非アクティブにすることが重要です。

```
deactivate

cd ~

git clone https://github.com/openocd-org/openocd.git

sudo apt install libusb-1.0-0-dev libhidapi-dev
```

これが完了したら、OpenOCD フォルダ内に OpenOCD をビルドできます。TI のハードウェア上にあるすべてのビルドでは、フラッシュ中にこのビルドを参照する必要があります。

```
cd <cloned_OPENOCD_dir>

git submodule update --init --recursive

cd jimtcl

./configure

make

sudo make install

cd .. #back in the cloned directory

sudo apt-get install libusb-1.0-0-dev

./bootstrap #when building from the git repository

./configure --enable-xds110 #optionally add any other debuggers as needed

make

sudo make install
```

これで、OpenOCD は次の手順に進む準備が整いました。

3.1.4 TI のダウンストリームの差別化

最後のステップは、TI のダウンストリーム Zephyr リポジトリをセットアップすることです。TI は、コミュニティによって管理されている Zephyr アップストリームの別のフォークを管理しています。これは TI ダウンストリーム、つまり TI が管理するフォークと呼ばれ、アップストリームにまだ含まれていない機能が含まれています。これには、追加のボード、ユーザー固有のアプリケーション、新しいドライバなどが含まれます。さらに、このフォークは TI によって積極的に管理されているため、ユーザーは E2E フォームを使用してサポートを受けることができます。アップストリームとダウンストリームの両方に存在するアイテムの場合は、アップストリーム フォークを使用することをお勧めします。

最初の手順では、分岐の現在のステータスをチェックして、現在のリモート ポインタの名前を確認します。

```
cd ~/zephyrproject/zephyr

git status

git remote -v
```

上記の手順に従うと、アップストリームがインストールされ、追跡されます。上記のコマンドの出力は、以下のようになります。

```
origin  https://github.com/zephyrproject-rtos/zephyr (fetch)
origin  https://github.com/zephyrproject-rtos/zephyr (push)
```

origin は非特異的な名前であるため、わかりやすくするために、アップストリームとダウンストリームを明示的に参照することをお勧めします。以下のコマンドはどちらも、TI のダウンストリームのソースを作成すると同時に、zephyr origin を upstream に変更します。

```
git remote rename origin upstream

git remote add downstream https://github.com/TexasInstruments/msp-zephyr.git
```

git remote -v を実行した後の予想される出力は、以下のようになります。

```
#expected output after git remote -v
downstream      https://github.com/TexasInstruments/msp-zephyr.git (fetch)
downstream      https://github.com/TexasInstruments/msp-zephyr.git (push)
upstream        https://github.com/zephyrproject-rtos/zephyr (fetch)
upstream        https://github.com/zephyrproject-rtos/zephyr (push)
```

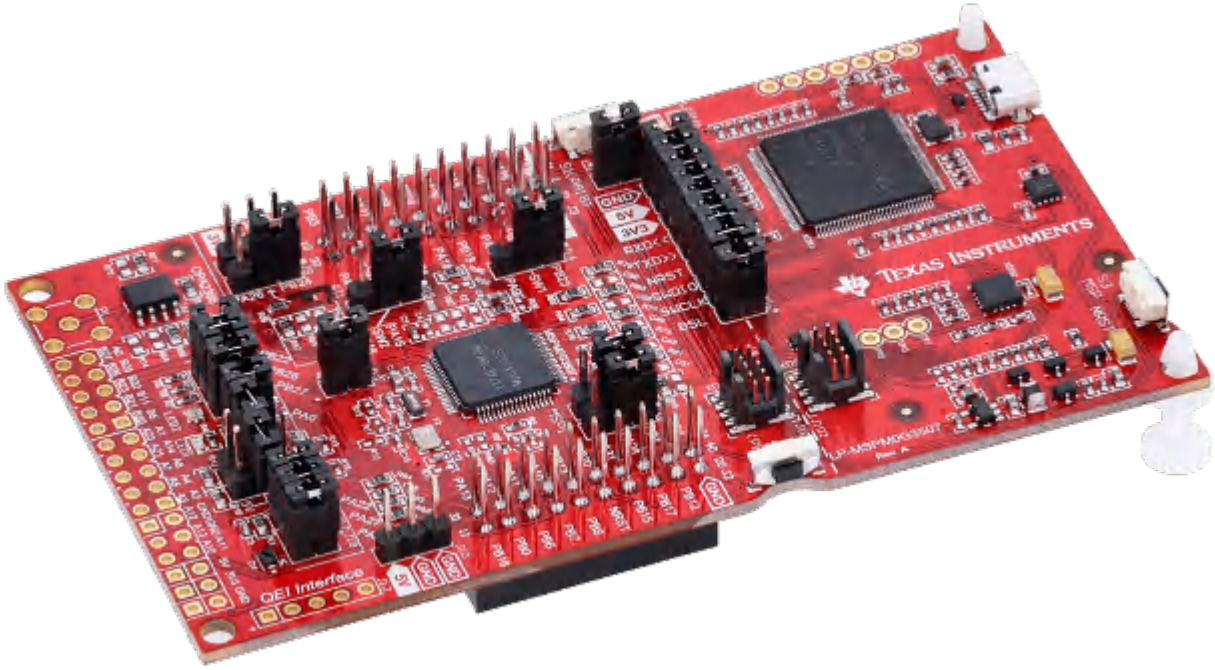
そこから、ダウンストリームへの更新には、ダウンストリーム ポインタを使用してアクセスできます。以下の例では、既存の分岐をチェック アウトし、新しいプロジェクトの開発時にバージョン管理用の新しいローカル分岐を作成します。

```
git fetch downstream # receives information about the downstream

git checkout downstream/stable # checks out remote downstream stable branch without
creating local branch

git checkout -b stable downstream/stable # creates new local branch called stable that
tracks the downstream stable branch
```

4 MSPM0 LaunchPad でサンプルを実行する方法

4.1 MSPM0 Launchpad

MSPM0 ファミリのデバイスには、Launchpad と呼ばれる評価基板があり、これを使用して Zephyr の試験とプロトタイプ製作を開始できます。LP-MSPM0G3507 を以下に示します。これは、以下のチュートリアルで使用します。



4.2 MSPM0 Launchpad でプロジェクトを実行する

以下のセクションは、Zephyr ドキュメントの入門ガイドに基づいています。ただし、重要な変更がある場合は、ここで説明します。

4.2.1 Blinky を実行する

ファイル システムが適切にセットアップされたら、USB 経由で LaunchPad を接続します。仮想環境で、仮想環境に再度入り、/ zephyr フォルダに cd を挿入して、以下を実行します。

```
cd ~/zephyrproject/zephyr
```

```
west build -p always -b lp_mspm0g3507 -d out samples/basic/blinky
```

```
west flash -d out --openocd ~/openocd/src/openocd --openocd-search ~/openocd/tcl
```

これは、以前にインストールされた OpenOCD バージョンを使用してボードをビルドし、フラッシュします。

*現在サポートされているボードを確認するには、**west build** を実行し、**LaunchPad** に対応するボード名を使用します。**LP-MSPM0G3507 LaunchPad™** の場合、ボード名は **lp_mspm0g3507** です。*

フラッシュが完了すると、ハードウェアの再起動時に LED が定期的に点滅します。フラッシュ時には、ボードはデバッグモードに入り、west debug コマンドまたはセクション 5.2.3「プロジェクトのデバッグ」で説明されているその他のデバッグツールを使用して west 経由でデバッグすることができます。

4.2.2 より複雑なサンプルを実行する

他の MSPM0 のペリフェラルを対象としたより複雑なサンプルも用意されており、ユーザーはこれらの構築済みサンプルを使用して迅速にプロトタイプを製作したり、それらを変更してより大規模で複雑なプロジェクトの出発点として使用したりできます。

Blinky と同様に、これらのサンプルをすぐに実行するには、west ビルドでプロジェクトをビルドし、west フラッシュを使用してボードにフラッシュできます。Zephyr は、さまざまなボード向けに製作された多様なサンプルが付属しています。Zephyr のポータブル設計により、MSPM0 に簡単に移植できる多くのものが存在します。これらのサンプルは、複雑さの程度がさまざまですが、samples / フォルダ内にあります。

4.3 プロジェクトのデバッグ

4.3.1 コマンド ライン付き GNU Debugger (GDB)

GDB は、XDS-110 エミュレータとのインターフェイスを確立して、Zephyr プロジェクトをデバッグするために使用できるコマンド ライン インターフェイスです。CCS と同様に、Zephyr プロジェクトのビルド時に作成された zephyr.elf ファイルは、GDB を使用してデバッグできます。

west デバッグには複数のオプションがありますが、『Zephyr Project の west デバッグ ガイド』を参照して、使用可能な特定のランナーやデバッグ ツールを確認することをお勧めします。

4.3.2 Visual Studio Code (VSCode) 環境を設定する

Visual Studio Code は、適切な拡張機能を使用して組込みプロジェクトをデバッグする一般的な方法です。

拡張機能「Cortex-Debug」により、Visual Studio Code 内で視覚的なデバッグが可能になります。VSCode 内の「extensions menu」(拡張機能メニュー) から拡張機能をインストールした後、launch.json ファイルが MSPM0 Zephyr Projects をサポートするように設定する必要があります。その後、環境は Zephyr プロジェクトのビルド、フラッシュ、デバッグを行う準備ができます。

```
{
    "version": "2.0.0",
    "configurations": [
        {
            "name": "Zephyr Debug",
            "executable":"<absolute_path_to>/zephyrproject/zephyr/out/zephyr/zephyr.elf,
            "request": "launch",
            "type": "cortex-debug",
            "runToEntryPoint": "main",
            "servertype": "external",
            "gdbPath": "<absolute_path_to>/<path_to_zephyr_sdk>/arm-zephyr-eabi/bin/arm-zephyr-eabi-gdb",
            "gdbTarget": "localhost:3333",
            "device": "MSPM0G3507"
        },
    ]
}
```

4.3.3 VSCode で Cortex-Debug を使用してデバッグする

VSCode で、ボードの .ccxml (ほとんどの例の「ターゲット構成」セクションにあります) を使用してボードに接続し、それを使用してプロジェクトなしのデバッグを開始します。そこから、west ビルド コマンドを使用するときに west によって生成される zephyr.elf を使用してシンボルをロードできます。

セクション 4.3.2 に示すように、VSCode 環境が debug-cortex で設定されていると仮定すると、以下のコマンドを使用してデバッグ サーバーを起動し、VSCode 内でデバッグを開始することができます。

```
west debugserver -d out --openocd ~/openocd/src/openocd \
--openocd-search ~/openocd/tcl
```

4.4 独自のプロジェクトを作成する

新規プロジェクトは、既存の例から構築することも、zephyrproject/zephyrfolder (リポジトリ アプリ)、zephyrproject / フォルダ自体 (ワークスペース アプリ)、または以前に作成したフォルダの外側に新規フォルダを作成することによって作ることもできます (独立型アプリ)。詳細については、Zephyr のドキュメントでこれらのさまざまなアプリケーション タイプを参照してください。

独自のプロジェクトを作成する場合、Zephyrproject のアプリケーション開発セクションに知っておくべき情報がすべて記載されています。一般的に、TI または Zephyr が提供したサンプルのいずれかをベースにプロジェクトを作成する方が、プロジェクトを作成するためのより簡単な方法です。これらのサンプルには、必要な構成ファイルがデフォルトで含まれています。つまり、オーバーレイや prj.conf ファイルなどの微調整が少なくなります。

5 参考資料

- RTOS の基本ガイド
- LP-MSPM0G3507 LaunchPad
- Zephyr ドキュメント 入門ガイド
- Cortex-Debug VSCode 拡張機能

6 E2E

TI 製品のサポートについては、E2E サポート フォームを使用してよくある質問を参照し、新しい質問を投稿してください。

7 改訂履歴

資料番号末尾の英字は改訂を表しています。その改訂履歴は英語版に準じています。

日付	改訂	注
2026 年 2 月	*	初版リリース

重要なお知らせと免責事項

TI は、技術データと信頼性データ (データシートを含みます)、設計リソース (リファレンス デザインを含みます)、アプリケーションや設計に関する各種アドバイス、Web ツール、安全性情報、その他のリソースを、欠陥が存在する可能性のある「現状のまま」提供しており、商品性および特定目的に対する適合性の黙示保証、第三者の知的財産権の非侵害保証を含むいかなる保証も、明示的または黙示的にかかわらず拒否します。

これらのリソースは、TI 製品を使用する設計の経験を積んだ開発者への提供を意図したものです。(1) お客様のアプリケーションに適した TI 製品の選定、(2) お客様のアプリケーションの設計、検証、試験、(3) お客様のアプリケーションに該当する各種規格や、その他のあらゆる安全性、セキュリティ、規制、または他の要件への確実な適合に関する責任を、お客様のみが単独で負うものとします。

上記の各種リソースは、予告なく変更される可能性があります。これらのリソースは、リソースで説明されている TI 製品を使用するアプリケーションの開発の目的でのみ、TI はその使用をお客様に許諾します。これらのリソースに関して、他の目的で複製することや掲載することは禁止されています。TI や第三者の知的財産権のライセンスが付与されている訳ではありません。お客様は、これらのリソースを自身で使用した結果発生するあらゆる申し立て、損害、費用、損失、責任について、TI およびその代理人を完全に補償するものとし、TI は一切の責任を拒否します。

TI の製品は、 TI の販売条件 、 TI の総合的な品質ガイドライン 、 ti.com または TI 製品などに関連して提供される他の適用条件に従い提供されます。TI がこれらのリソースを提供することは、適用される TI の保証または他の保証の放棄の拡大や変更を意味するものではありません。 TI がカスタム、またはカスタマー仕様として明示的に指定していない限り、TI の製品は標準的なカタログに掲載される汎用機器です。

お客様がいかなる追加条項または代替条項を提案する場合も、TI はそれらに異議を唱え、拒否します。

最終更新日：2025 年 10 月