IEEE標準リアルタイムOS

「μT-Kernel 3.0」による最新組込みシステム開発

アジェンダ

- μT-Kernel 3.0とは
- 最新情報
  - Arm v8-Mアーキテクチャ、TrustZone対応
- μT-Kernel 3.0を使用するには
  - BSP(ボードサポートパッケージ)
  - マイコンボード向け開発環境の構築（実演）
- μT-Kernel 3.0プログラミング
- 質疑応答

μT-Kernel 3.0とは

µT-Kernel 3.0とは

- 小規模組込みシステム, IoTエッジノード向け高機能リアルタイムOS
- TRONプロジェクトの最新のリアルタイムOS
  - µITRONやT-KernelなどTRONのOSと高い互換性
- IEEE 2050-2018 国際標準仕様に準拠した軽量リアルタイムOS
  - IEEE 2050-2018のベースはµT-Kernel 2.0
- ベンダーニュートラルなオープンソースのリアルタイムOS
  - GitHubからソースコード公開
  - 改変可能、商利用含めて無償なT-License

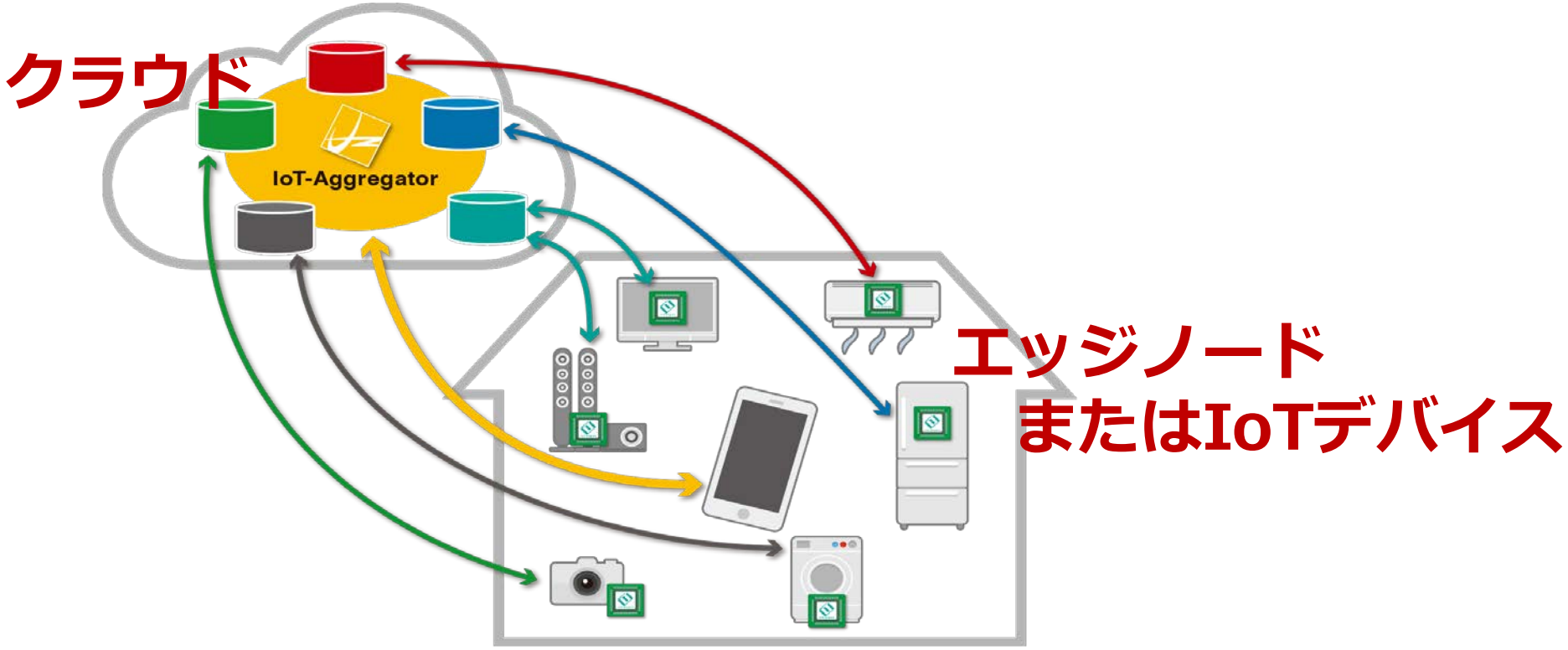
µT-Kernel 3.0

組込みシステムとは

- 家電製品や産業機械など様々な電子・電機機器に組み込まれて使われるコンピュータ・システム

IoTエッジノードとは

- IoT (Internet of Things: “モノ”のインターネット) のネットワークの末端である“モノ”
- ネットワークに接続された組込みシステム



リアルタイムOSとは

- 主に組込みシステムで使用されるOS
  - 機器に組み込まれ特定のアプリケーションを実行する
  - 機器の制御が主たる目的
- 汎用OSと比べた特徴
  - 高いリアルタイム性能
    - 人ではなく機器の制御が目的のため
  - 少ないリソースで動作
    - マイコン（マイクロコントローラ, MCU）で実行するため
- 様々な種類のリアルタイムOSが存在する
  - 標準規格としてIEEE 2050-2018がある
    - IEEE 2050-2018の仕様のベースはTRONのOS仕様

TRONプロジェクトのリアルタイムOS

T0	T1	T2	T3
1993 ITRON μITRON3.0 1999 ITRON μITRON4.0	2009 MP T-Kernel 2004 T-Kernel 2007 μT-Kernel	2011 T-Kernel 2.0 2013 μT-Kernel 2.0	2018 IEEE 2050-2018 2019 μT-Kernel 3.0

TRONのリアルタイムOSの機能比較

分類	機能	µITRON 4.0	IEEE2050-2018	µT-Kernel 3.0
タスク	タスク管理	○	○	○
	タスク付属同期	○	○	○
	タスク例外	○	○	○
同期・通信	セマフォ	○	○	○
	イベントフラグ	○	○	○
	メールボックス	○	○	○
	ミューテックス	○	○	○
	メッセージバッファ	○	○	○
メモリ管理	固定長・可変長メモリプール	○	○	○
	システムメモリ管理	×	○	○
時間管理	周期ハンドラ	○	○	○
	アラームハンドラ	○	○	○
ハードウェア管理	割込みハンドラ	○	○	○
	デバイス管理	×	○	○
	物理タイマ	×	○	○
機能拡張	サブシステム	×	×	○

リアルタイムOSの基本機能は変わらない

- μITRONからμT-Kernelへの移行も比較的簡単

- 何故リアルタイムOSの基本機能は変わらないのか？
  - 省資源を重視し付加的な機能はミドルウェアとして実現
    - 組込み製品として不要な機能はコスト増
  - 組込みシステムは製品寿命が長いことが多いので不要なバージョンアップをきらう
    - T-Kernelのキャッチコピーは「100年ソフト」だった

- では何故リアルタイムOSはバージョンアップする？

マイコンや開発環境は変化する

- マイコンは5年もすれば世代が変わる
  - 2000年代初め T-Kernel 1.0が開発された頃はハイエンドに32bitマイコンが使われだした
  - T-Kernel 2.0、μT-Kernel 2.0のリファレンスコードのターゲットはArm 11
- T-Kernel 1.0の開発環境は
  - MakeとPerlで多品種のマイコンに対応
  - 途中からEclipseに独自機能拡張で対応
  - 実はこの状況はμT-Kernel 2.0まで続いていた
- これらに対応するためにバージョンアップは必要

µT-Kernel 3.0の対応マイコン

- 組込みシステムの主だったCPUアーキテクチャに対応
  - リアルタイムOSが大きく依存するのはCPUコアのアーキテクチャ
- µT-Kernel 3.00.08.B0ではArm Cortex-M0+, M33, M85に対応

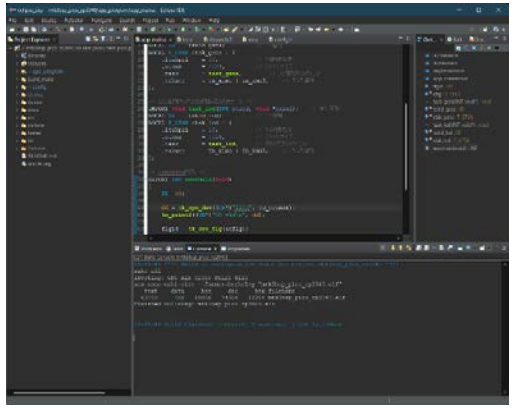
種別		対応マイコン
ARM	**Cortex-M**	**ARM Cortex-M3** TMPM367FDFG（東芝デバイス＆ストレージ製）
		ARM Cortex-M4 STM32L486（STマイクロエレクトロニクス製）
		ARM Cortex-M7 STM32H723（STマイクロエレクトロニクス製）
	Cortex-A	**ARM Cortex-A9** RZ/A2M（ルネサス エレクトロニクス製）
RX		**RXv2** RX231（ルネサス エレクトロニクス製）
		RXv2 RX65N（ルネサス エレクトロニクス製）

µT-Kernel 3.0 の開発環境

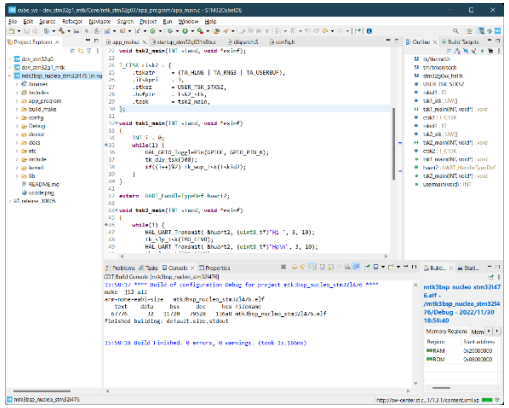
- µT-Kernel 3.0は開発環境への依存を極力排除
- ”基本的にはC言語処理系が動作すればよい”
  - µT-Kenrel 3.0をビルドするには
    - ターゲット・ハードウェアのシンボル指定のみ
      - 例：
        STM32L4 IoT-Engineならば　　_IOTE_STM32L4_
        Raspberry Pi Picoならば　　__PICO_RP2040__
    - インクルードファイルのパスも３つに絞る
      - /config
      - /include
      - /kernel/knlinc (カーネルのビルド用)
    - ソースコードは原則 C99（以降）
      - ただし一部ハードウェア依存のコードにC言語処理系(GCC)に依存はあり
      - アセンブラ
      - リンクのセクション指定

各種開発環境で使用可能

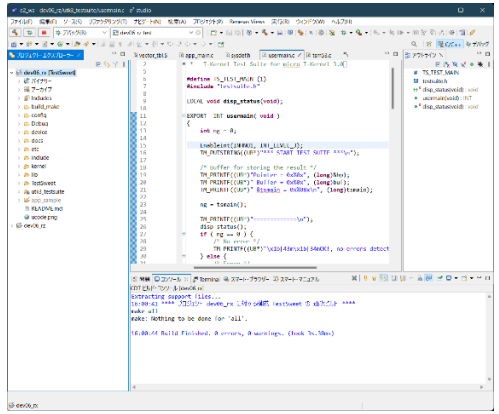
- Eclipse Embedded CDT



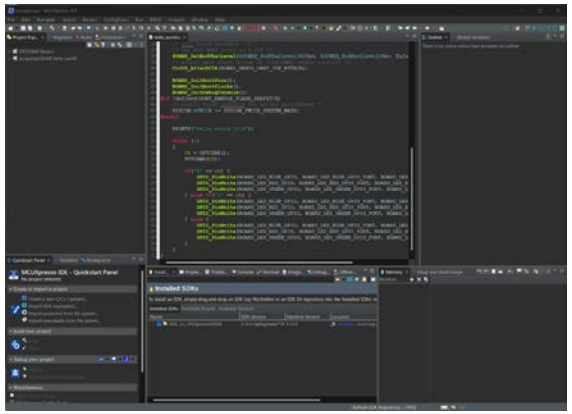
- STM32CubeIDE



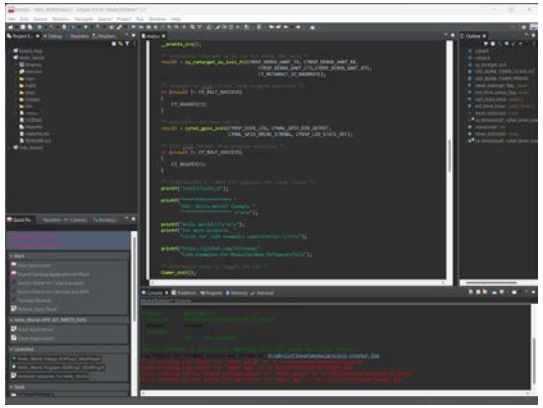
- e²Studio



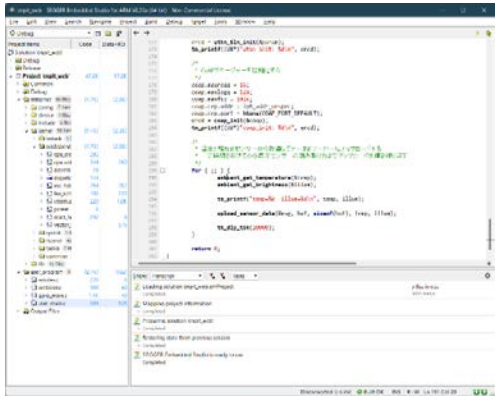
- MCUXpresso



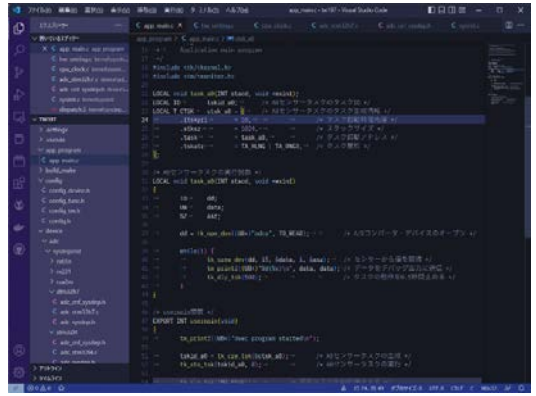
- ModusToolbox



- SEGGER Embedded studio



- Visual studio code



その他、マイコンのC言語処理系が動作するIDEであれば使用可能

µT-Kernel 3.0 最新情報

μT-Kernel 3.0 最新バージョン

- v3.00.07 2024年4月リリース
  - マイコンメーカ提供のコンフィギュレータ、ファームウェアなどとの親和性を強化
- v3.00.08.B0 2024.12月 βリリース
  - Armv8-Mアーキテクチャのマイコンに対応
    - ARM Cortex-M33, M85など
    - タスクのスタックオーバーフロー検出機能を実装
- TrustZone対応（2025年度後半予定）

マイコンメーカ提供のコンフィギュレータ、ファームウェアなどとの親和性を強化

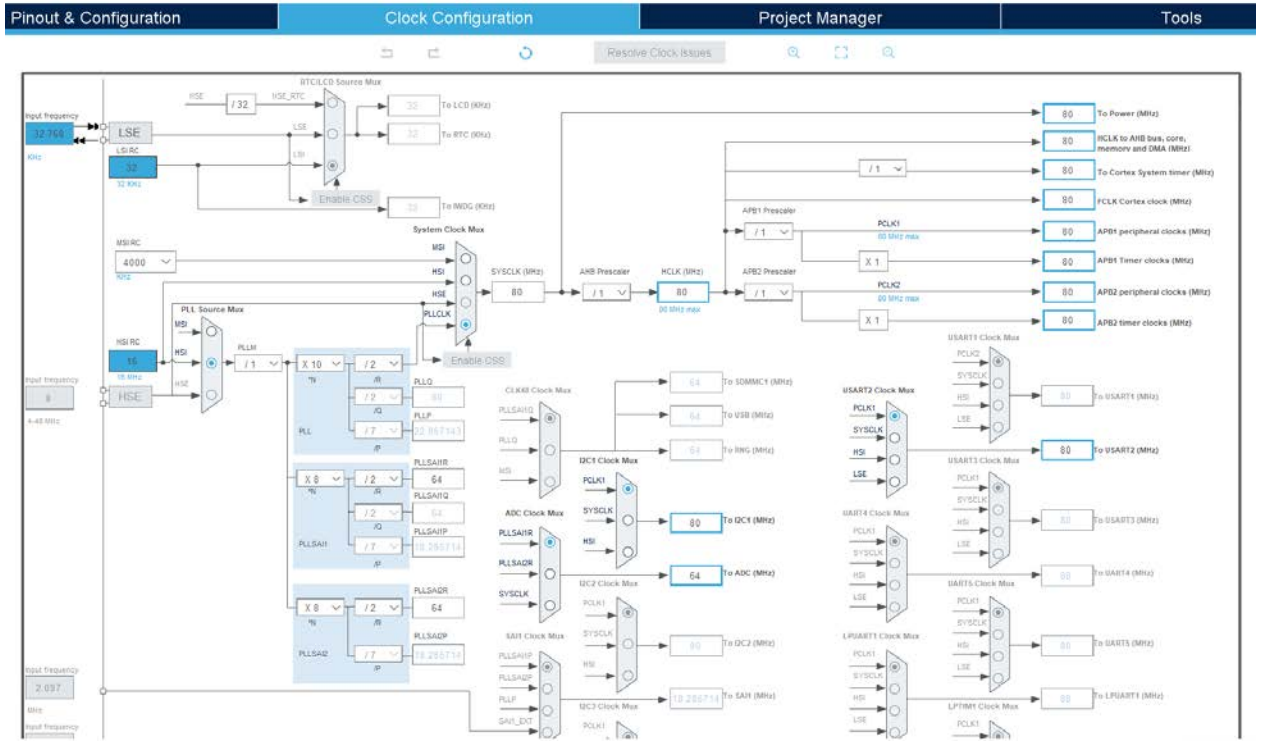
- μT-Kernel 3.0は特定の開発環境に依存しない(基本方針)
- 他のソフトウェアに依存しない完全なスタンドアローン開発を主としてきた
  - ブート処理からデバイスドライバ、ライブラリまで独立
  - もちろん、デバイスドライバ、ミドルウェア、ライブラリなどの移植は可能

- 組込み製品の開発はこれが求められていたのだが…
  - 他のソフトウェアを利用するとライセンス、品質保証の問題

複雑化するマイコンのソフト開発

- マイコンに内蔵されるペリフェラルの多機能化、高性能化に伴い、今やマイコンを起動させるだけでも大変
- 例えばマイコンのクロックを設定するにも、メーカ提供のツール無しでは大変

STM32CubeIDEのクロック設定画面

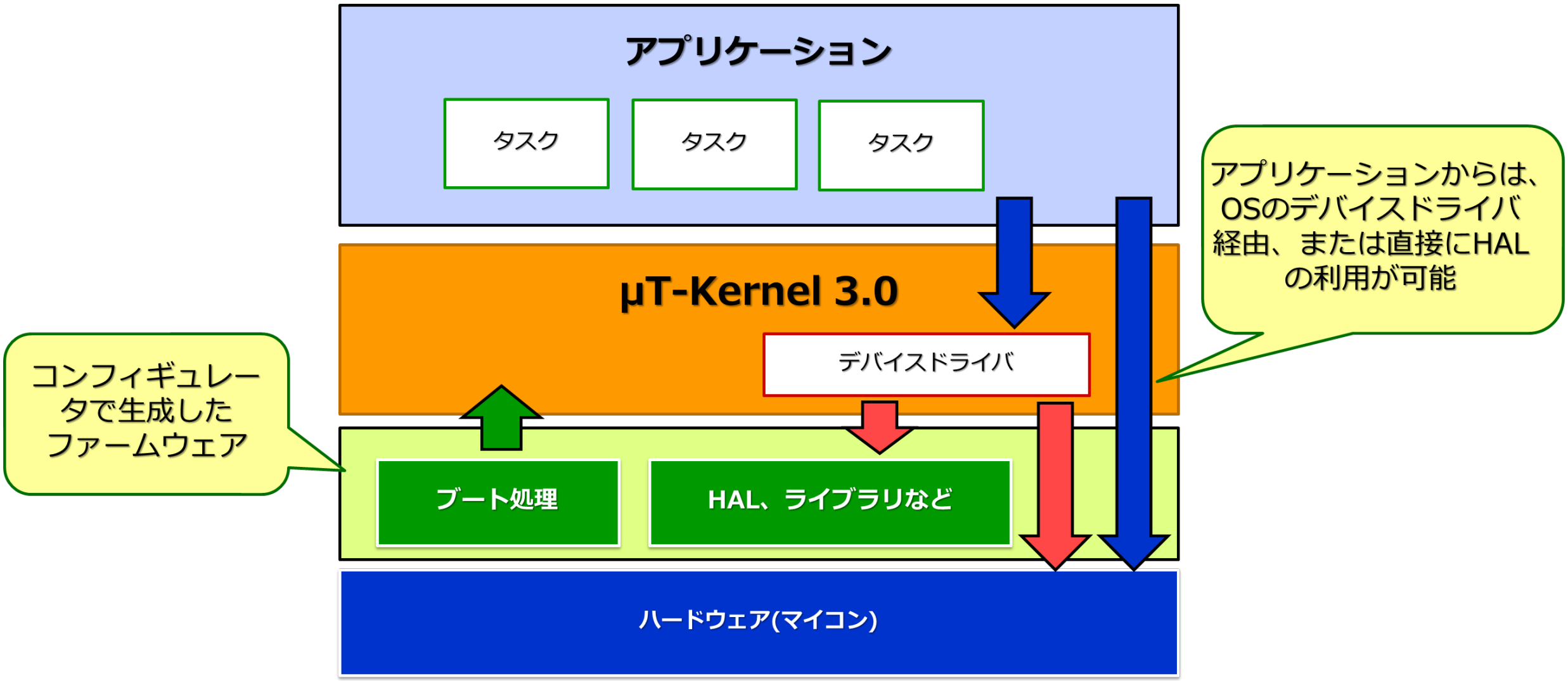


メーカ提供のツールを使った開発

- IDEやコンフィギュレータで各種設定を行いプロジェクト(ソースコード)を自動生成
  - クロック設定
  - 端子設定
  - ペリフェラル設定
- メーカからHAL (Hardware Abstraction Layer)、ライブラリ、ミドルウェアなどが提供
  - アプリケーションの開発にすぐに取り掛かれる

- μT-Kernel 3.0もメーカ提供のツール、プラットフォームで簡単に使用したい　→　v3.00.07で対応

µT-Kernel 3.00.07の構成例



µT-Kernel 3.00.07
プロジェクトへの組込み手順

① コンフィギュレータやIDEでプロジェクトの作成

② ハードウェアのコンフィギュレーション
- クロック設定、ペリフェラル設定、端子設定など

③ µT-Kernel 3.0 のソースコードのコピー

④ µT-Kernel 3.0 の簡単な設定
- ターゲットシンボル定義、インクルードパス設定、エントリ関数の呼び出し

⑤ ビルドして完成

後ほど実演します

Armv8-Mアーキテクチャ

- Arm Cortex-Mの新しいアーキテクチャ
  - Armからの発表は2015年
  - ここ数年で各メーカから対応マイコンが販売されはじめた
- Armv8-Mマイコンの分類
  - ベースライン： Cortex-M23
    - Cortex-M0+の上位
  - メインライン： Cortex-M33, M55, M85
    - Cortex-M3/M4/M7の上位
- µT-Kernel 3.00.08で対応

µT-Kernel 3.0のArmv8-M対応状況

- 対応マイコン(v3.00.08.B0)
  - ルネサス エレクトロニクス **RA8D1, RA8M1** (Cortex-M85)
  - NXP **MCX N947, LPC55S69** (Cortex-M33)
  - STマイクロエレクトロニクス **STM32N657** (Cortex-M33) 開発中
- スタックオーバーフロー検出機能実装
  - Armv8-Mで実装された機能
- TrustZone対応（開発中）
  - 現状はTrustZone無効状態で動作

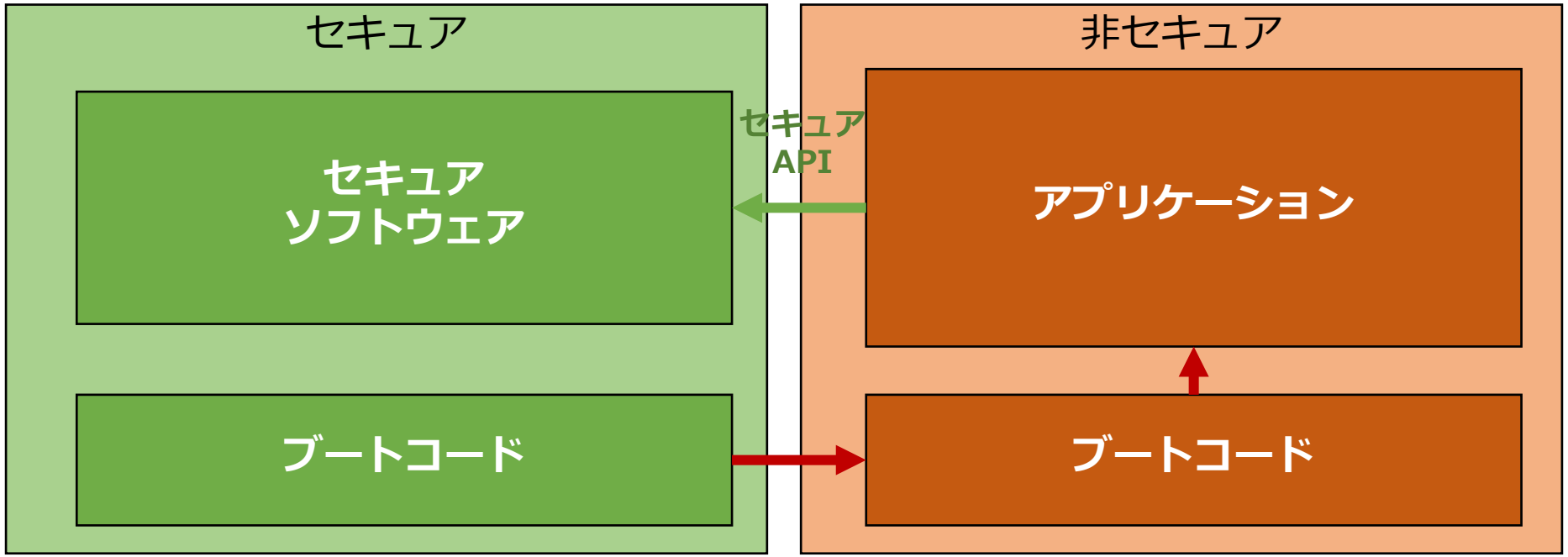
スタックオーバーフロー検出機能

- Armv8-Mのスタックポインタモニタ機能：
  実行時にスタックオーバーフローを検出し例外を発生
- µT-Kernel 3.0はこの機能を使用しタスク毎にスタックのオーバーフロー検出が可能

  - ルネサスRAマイコンはArmv7-Mでも独自のスタックオーバーフロー検出機能がある　→　µT-Kernel 3.0は対応

TrustZoneとは

- Armのセキュリティ技術
  - Cortex-MではArmv8-Mから搭載
- マイコン上のソフトウェアをセキュアと非セキュアに分離
  - 非セキュアのソフトウェアからセキュアの情報はアクセスできない

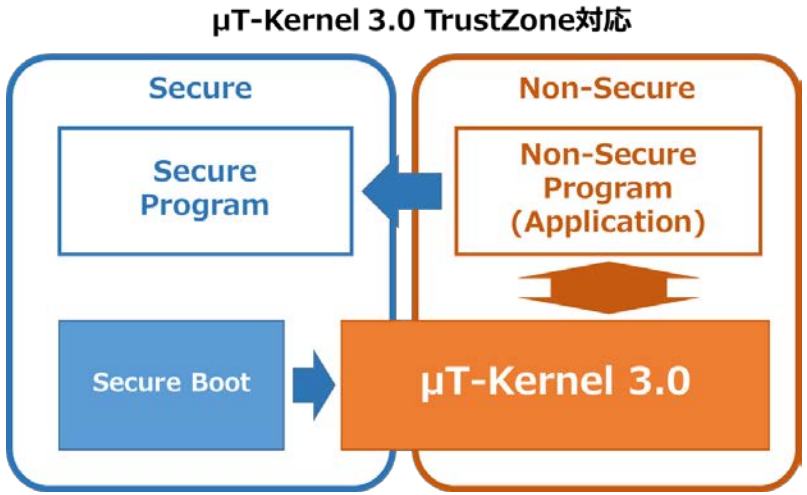


TrustZoneと特権モード

- 従来の特権/非特権(ユーザ)と共存
  - セキュアの特権/非特権、非セキュアの特権/非特権、
- 特権はSafety、TrostZoneはSecurity と考えることもできる

μT-Kernel 3.0のTrustZone対応 (開発中)

- OS、アプリケーションは基本的に非セキュアで実行
  - ArmのPSA(Platform Security Architecture)の考え方
- アプリケーションのタスクは必要に応じてセキュア・ソフトウェアを呼び出せる
  - TrustZoneのセキュアAPI
- セキュア・ソフトウェアからのOSのAPI実行は不可
- セキュア・ソフトウェア実行中でもリアルタイム性は確保
  - 割込みや時間管理によるディスパッチはセキュア実行中でも有効
  - このため一部OS機能はセキュアで実行される



μT-Kernel 3.0を使用するには

BSP(ボードサポートパッケージ)

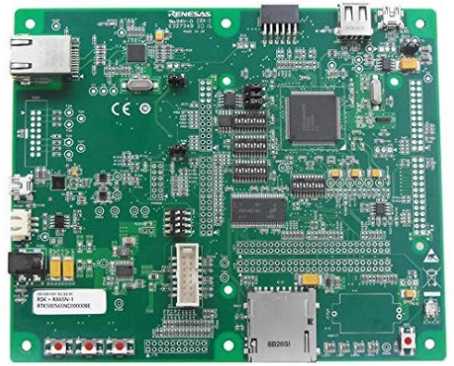
- μT-Kernel 3.0の入門や学習、評価用に市販マイコン・ボードですぐに使用可能なソフトウェア・パッケージ
  - OS,基本的なデバイスドライバ（A/Dコンバータ、I²C）、デバッグ用シリアル通信が実装済み

- BSPの種類
  - BSP
    - 他のソフトウェアを利用せずにスタンドアローンで実行可能
  - BSP2
    - マイコンメーカ提供のプラットフォーム上で実行

µT-Kernel 3.0 BSP

- 他のソフトウェアを使用せずスタンドアローンで実行
  - 2021年からリリース

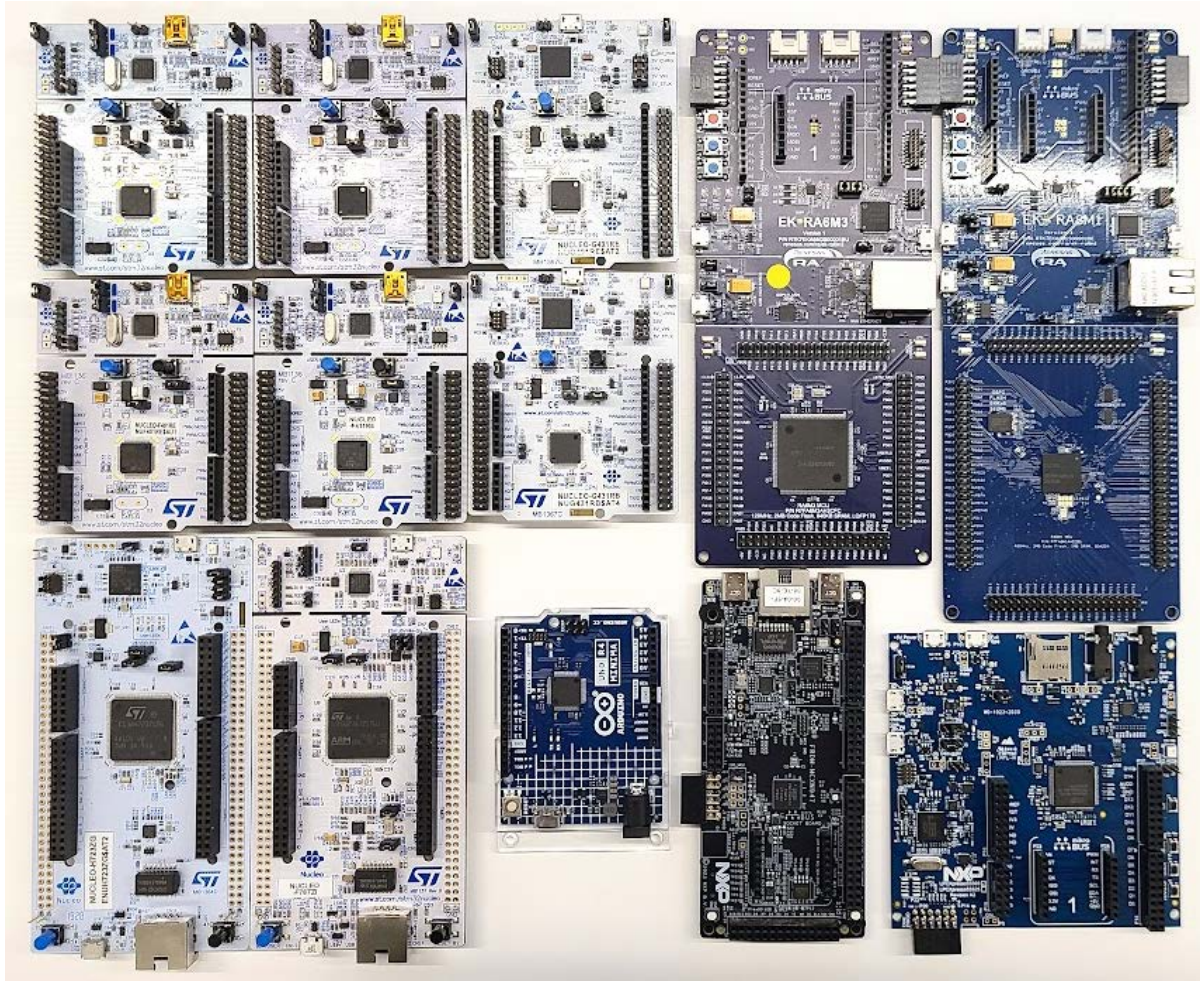
STM32L476 Nucleo-64	STM32H723 Nucleo-144	RX65N Renesas Target Board	RX65N Renesas Starter Kit+	Raspberry Pi Pico
ARM Cortex-M4	ARM Cortex-M7	RXv2	RXv2	ARM Cortex-M0+



µT-Kernel 3.0 BSP2

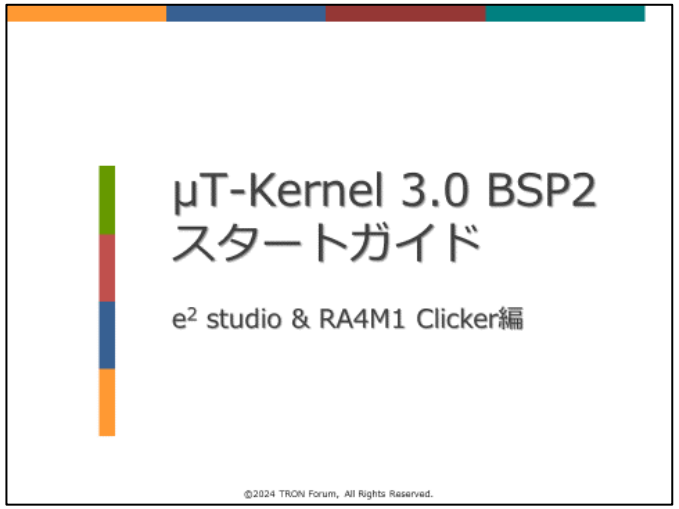
- ルネサス エレクトロニクス
  - EK-R8MD1
  - EK-RA8M1
  - EK-RA6M3
  - Clicker RA4M1
  - Arduino UNO R4
- STマイクロエレクトロニクス
  - STM32L476 Nucleo-64
  - STM32F401 Nucleo-64　STM32F411 Nucleo-64
  - STM32F446 Nucleo-64
  - STM32G431 Nucleo-64　STM32G491 Nucleo-64
  - STM32F767 Nucleo-144
  - STM32H723 Nucleo-144
  - STM32N6570-DK (開発中)
- NXP
  - FRDM-MCXN947
  - LPC55S69-EVK
- インフィニオン
  - EVK-XMC7200

その他、開発中



µT-Kenrel 3.0 BSP2 サンプル・プロジェクト

- IDEからインポートするだけですぐに使える完成したプロジェクトをGitHubからリリース
  - 基本的なデバイスドライバも組込み済み（I²C、A/Dコンバータなど）
- 対応マイコンボード
  - EK-RA8M1 (ルネサス エレクトロニクス)
  - RA4M1 Clicker (ルネサス エレクトロニクス)
  - NUCLEO-H723ZG (STマイクロエレクトロニクス)
  - FRDM-MCXN947 (NXP)
  - KIT_XMC72_EVK (インフィニオン)
- 各ボードについてスタートガイドも公開



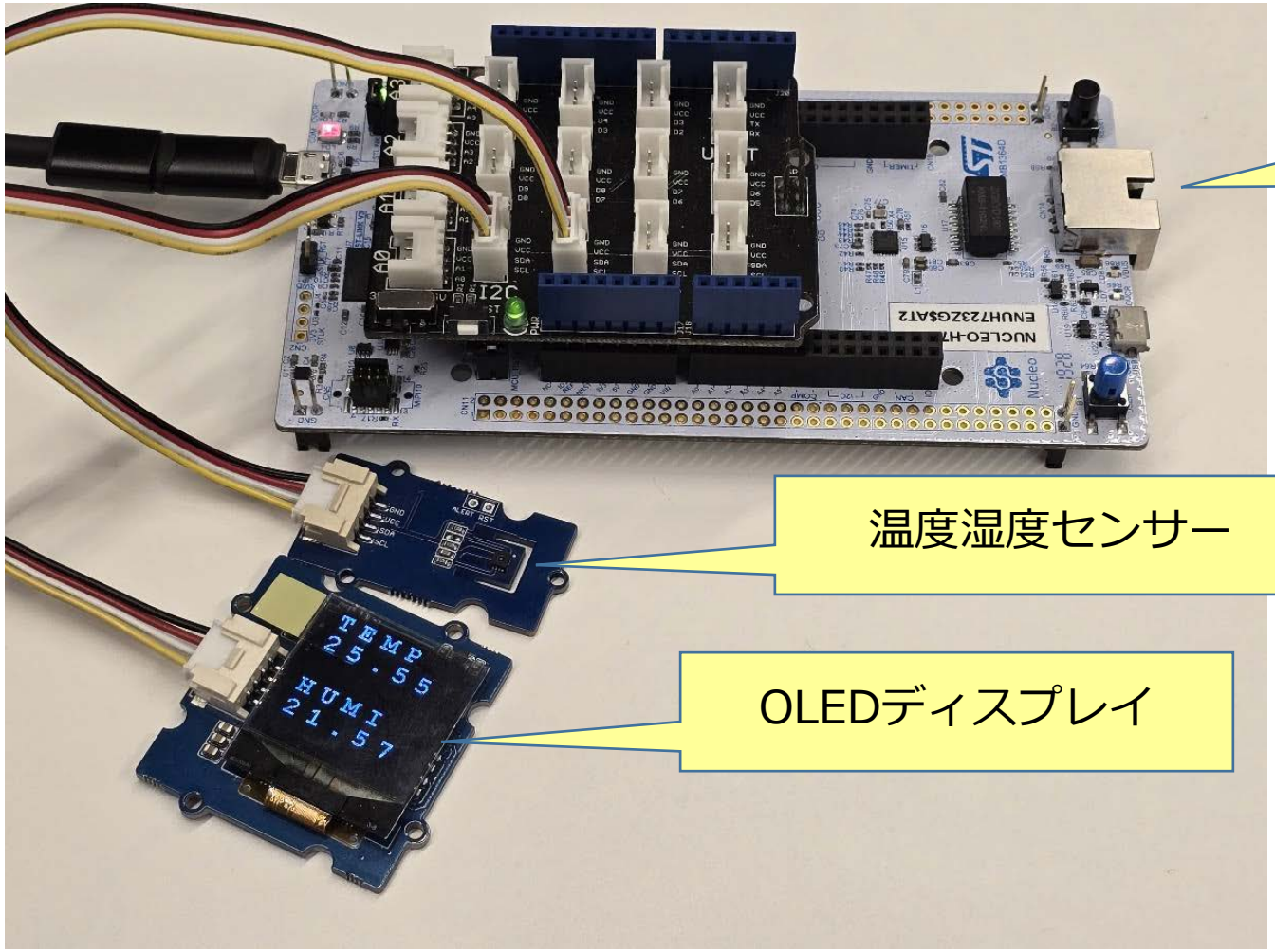
μT-Kernel 3.0関連の入手先

- μT-Kernel 3.0
  https://github.com/tron-forum/mtkernel_3
- μT-Kernel 3.0 BSP
  https://github.com/tron-forum/mtk3_bsp
- μT-Kernel 3.0 BSP2
  https://github.com/tron-forum/mtk3_bsp2
- μT-Kernel 3.0 サンプル・プロジェクト
  https://github.com/tron-forum/mtk3bsp2_samples

実演：BSP2による開発環境のセットアップ

- IDEで新規にプロジェクトを生成し、µT-Kernel 3.0 BSP2を組み込んで、アプリケーションを実行するまでを実演します
- 使用するマイコンボードおよび開発環境
  - STM32H723 Nucleo (STマイクロエレクトロニクス)
  - STM32CubeIDE
  - 他のマイコンボードでも手順は同じです
- 実行するアプリケーション
  - I²C接続のOLEDディスプレイ、温度湿度センサを制御し、センサー値をディスプレイに表示します

実演：BSP2による開発環境のセットアップ



STM32H723 Nucleo

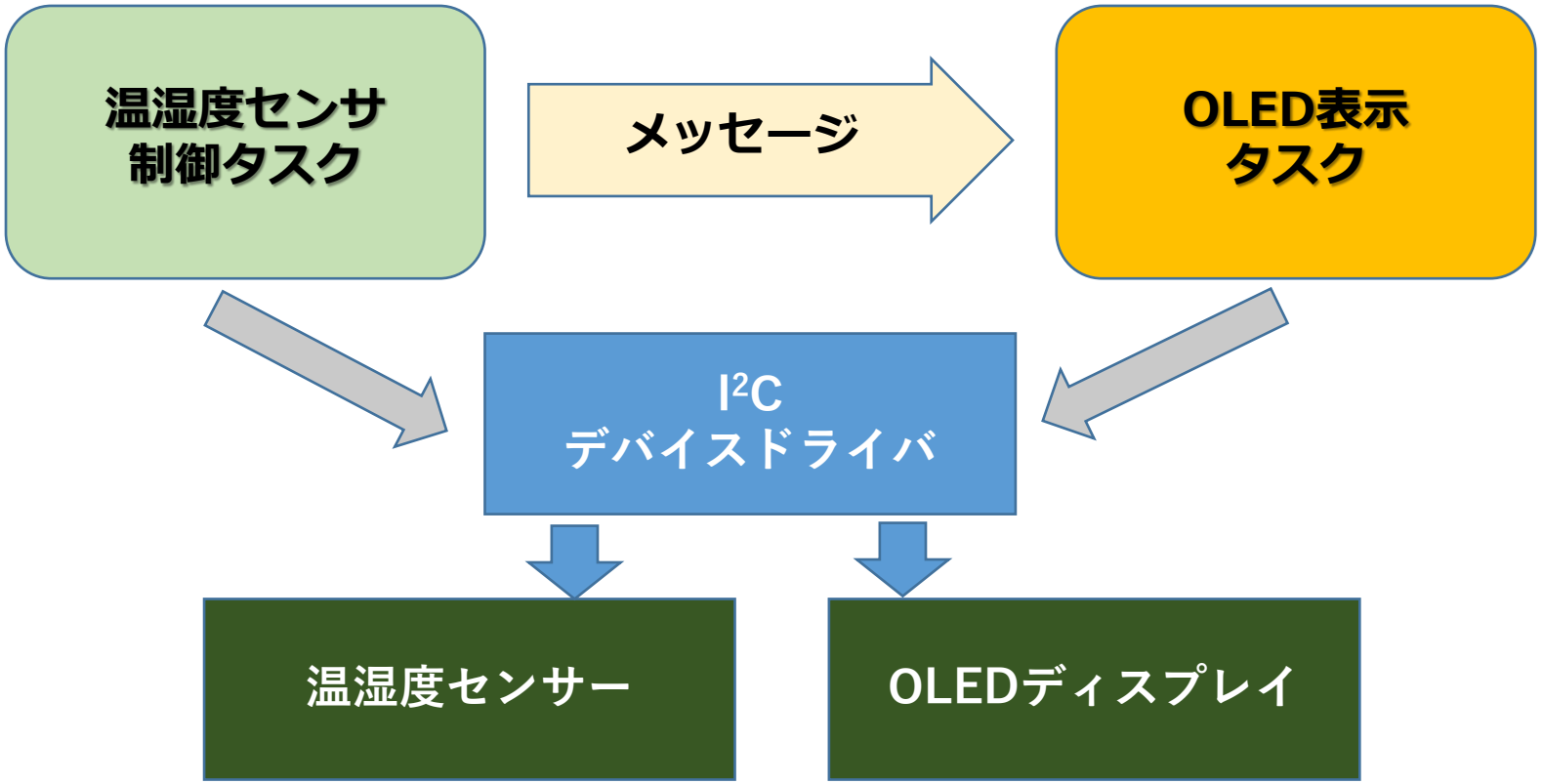
温度湿度センサー

OLEDディスプレイ

μT-Kernel 3.0 プログラミング

今回実行したアプリケーション

- 温湿度センサー制御タスクとOLCD表示タスクがメッセージ通信(メッセージバッファ)で連携して動作



アプリケーションの起動処理

Usermain関数（アプリケーション最初に実行される関数）

```
EXPORT INT usermain(void)
{
        ID      dd_i2c;

        dd_i2c = tk_opn_dev( (UB*)I2C_DEVNM, TD_UPDATE);        // I2Cドライバオープン

        mbfid_sns = tk_cre_mbf(&cmbf_sns);    // メッセージバッファの生成

        tskid_sns = tk_cre_tsk(&ctsk_sns);    // センサー制御タスクの生成
        tk_sta_tsk(tskid_sns, (INT)dd_i2c);   // センサー制御タスクの実行

        tskid_oled = tk_cre_tsk(&ctsk_oled); // OLED表示タスクの生成
        tk_sta_tsk(tskid_oled, (INT)dd_i2c); // OLED表示タスクの実行

        tk_slp_tsk(TMO_FEVR);                 // 起床待ち
        return 0;                             // ここは実行されません
}
```

タスク生成情報 (センサー制御タスク)

```
LOCAL T_CTSK ctsk_sns = {
        .itskpri = 10, .stksz = 1024, .task = task_sns, .tskatr = TA_HLNG | TA_RNG3
};
```

温度湿度センサー制御タスク

温度湿度センサー制御タスクの制御関数

```
LOCAL void task_sns(INT stacd, void *exinf)
{
        ID                  dd_i2c = (ID)stacd;
        T_SNS_DATA          sns_data;
        UB                  cmd[2], data[6];
        SZ                  asz;
        ER                  err;

        while(1) {
                cmd[0] = 0x2C; cmd[1] = 0x06;
                err = tk_swri_dev(dd_i2c, I2C_SADR, cmd, sizeof(cmd), &asz);     // コマンド送信
                err = tk_srea_dev(dd_i2c, I2C_SADR, data, sizeof(data), &asz);   // データ受信

                /* 温度・湿度を計算 */
                UW work = ((UW)data[0]<<8) | data[1];
                sns_data.temp = (work*17500 >> 16) - 4500;
                work = ((UW)data[3]<<8) | data[4];
                sns_data.humi = work*10000 >> 16;

                tk_snd_mbf(mbfid_sns, &sns_data, sizeof(sns_data), TMO_FEVR);    // メッセージ送信
                tk_dly_tsk(500);                                                 // 500ミリ秒待ち
        }
        tk_ext_tsk();
}
```

OLED表示タスク

OLED表示タスクの制御関数

```
LOCAL void task_oled(INT stacd, void *exinf)
{
        ID          dd_i2c = (ID)stacd;
        T_SNS_DATA  sns_data;
        ER          err;

        err = oled_init(dd_i2c);                    // OLED 初期化
        oled_onoff(dd_i2c, 1);                      // OLED オン
        err = oled_fill_screen(dd_i2c, 0x00);       // OLED 画面クリア

        while(1) {
                static UB           msg[128];
                INT                 work;

                err = tk_rcv_mbf(mbfid_sns, &sns_data, TMO_FEVR);      // メッセージ受信

                oled_fnt24_prints(dd_i2c, 0, 0, (UB*)“TEMP”);          // OLED文字列表示
                work = sns_data.temp /100;
                sprintf((char*)msg, "%2d.%2d", work, (int)(sns_data.temp - work*100));
                oled_fnt24_prints(dd_i2c, 0, 1, msg);                  // OLED文字列表示
                oled_fnt24_prints(dd_i2c, 0, 3, (UB*)“HUMI”);          // OLED文字列表示
                work = sns_data.humi /100;
                sprintf((char*)msg, "%2d.%2d", work, (int)(sns_data.humi - work*100));
                oled_fnt24_prints(dd_i2c, 0, 4, msg);                  // OLED文字列表示
        }
        tk_ext_tsk();
}
```

µT-Kernel 3.0 BSP2を利用したメリット

- 今回のアプリケーションプログラムは、 µT-Kernel 3.0 BSP2の全てのマイコンボードで実行できます
  - µT-Kernel 3.0のデバイス管理機能により、ペリフェラルの操作が標準化
  - 変更点はデバイス名のみ
- 今回使用しているデバイスドライバはメーカ提供のHALを使用して実装されています
  - IDEやコンフィギュレータから設定の変更が可能

μT-Kernel 3.0の
セミナー、技術情報

μT-Kernel 3.0 セミナー

- トロンフォーラムではμT-Kernel 3.0の各種セミナーを実施しています
- 3月26日【実習】μT-Kernel 3.0 組込みプログラミング入門 (RAマイコン・BSP2編)
  - INIAD(東洋大学情報連携学部) 東洋大学赤羽台キャンパス
  - 市販のマイコンボードを使用した１日のハンズオン実習
  - RA4M1 Clicker (Arm Cortex-M4搭載)を使用
- 詳細、申し込みは
  - https://www.tron.org/ja/2025/02/seminar0326/

µT-Kernel 3.0 解説書

- リアルタイムOSの基礎から
  デバイスドライバの開発、
  OSの移植まで説明
- µT-Kernel 3.0 BSPを使用し、市販のマイコンボードでプログラムを作成し実行
- 実習セミナーのテキストに使用

- 以下からダイジェスト版を公開
  https://www.tron.org/ja/page-722/rtos01/

µT-Kernel 技術情報サイト

- µT-Kernel 3.0に関する各種情報を掲載しています

  https://www.tron.org/mt-kernel3/

TRONプログラミングコンテスト2025

- マイコンメーカの協力のもとμT-Kernel 3.0のプログラミングコンテストを開催
  - 各メーカのマイコンボードとμT-Kernel 3.0 BSP2を使用
- https://www.tron.org/ja/programming_contest-2025/