EV3RTの内部構造と進んだ使い方

Beta 7-2

２０１９年６月１５日

名古屋大学　大学院情報学研究科
附属組込みシステム研究センター
李　奕驍

発表内容

- EV3RTの内部構造の解説
  - アーキテクチャの概要
  - ベースシステムの解説
  - ユーザアプリケーションとSDK
- 動的ローディング機能
  - Dynamic Module Loader
  - ユーザアプリのモジュール
  - ローディング操作に必要な処理
- 進んだ使い方
  - スタンドアローン形式
  - EV3RT/HRP3と新しい機能

発表内容

- EV3RTの内部構造の解説
  - アーキテクチャの概要
  - ベースシステムの解説
  - ユーザアプリケーションとSDK
- 動的ローディング機能
  - Dynamic Module Loader
  - ユーザアプリのモジュール
  - ローディング操作に必要な処理
- 進んだ使い方
  - スタンドアローン形式
  - EV3RT/HRP3と新しい機能

EV3RTの二つの実行形式

- スタンドアローン形式
  - ブートイメージ（uImage）＝ベースシステム＋ユーザアプリ
    - アプリを更新するたびに、SDカード書き換えてEV3を再起動する
  - プラットフォームの（特権モードを含む）全機能が使える
    - 動的ローディング形式のアプリローダ自体はこの形式のアプリの一つ

- 動的ローディング形式
  - ブートイメージ（uImage）＝ベースシステム＋アプリローダ
    - 起動後、ローダでアプリをBluetoothやSDカードから実行する
    - 複数回ロード可能で、アプリ更新の時、再起動が不要
      - ✓Bluetooth再接続等の手間が省ける
  - 普通のユーザアプリを開発する時に推奨
    - 制限：非特権モードで動作、指定のAPIで開発

EV3RTの二つの実行形式

- ビルド方式はmakeコマンドの引数で指定

```
Dynamic:      make app=<folder name> → Loader用Module
Standalone:   make img=<folder name> → U-Boot用uImage
```

- ソースコードの中でビルド方式をマクロで判定できる
  - ユーザアプリの開発者は意識しなくても良い
  - プラットフォームを改造・拡張するときに有用
    - e.g. ev3api.h, app_common.cfg

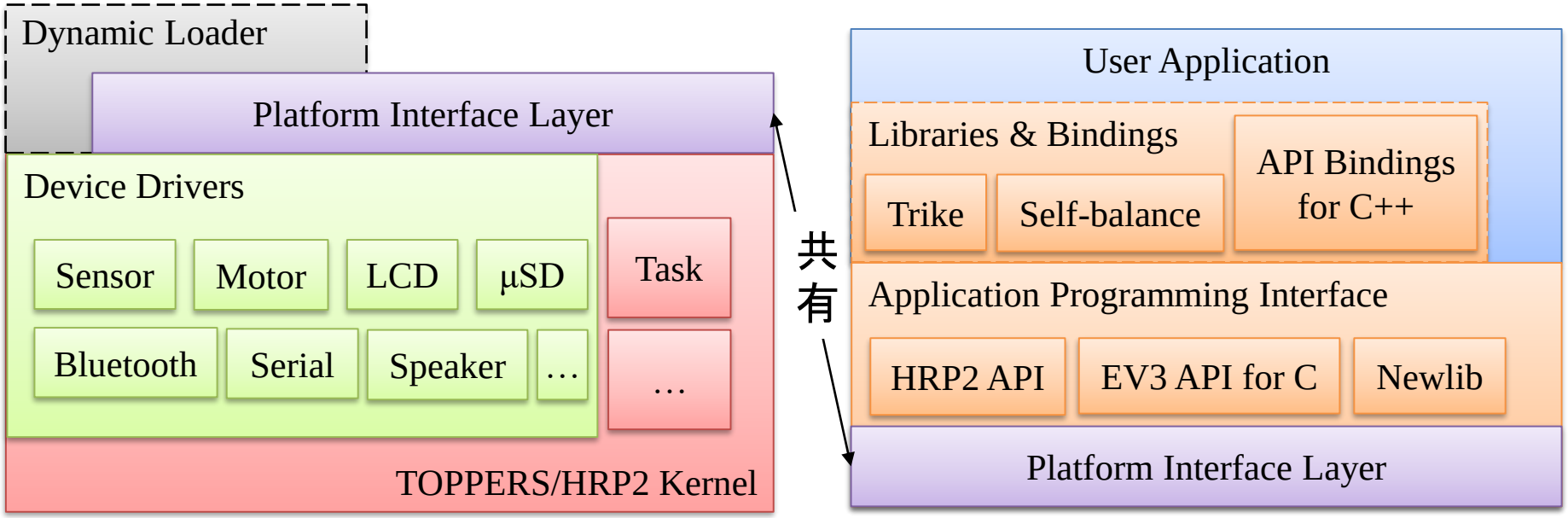
```
#if defined(BUILD_MODULE)
#include "module_cfg.h"
#else
#include "kernel_cfg.h"
#endif
```

```
#if !defined(BUILD_MODULE)
INCLUDE("ev3.cfg");
#endif

INCLUDE("ev3api.cfg");
```

EV3RTのアーキテクチャ

- 動的形式のアーキテクチャは2つの部分に分けられる



ベースシステム（+ローダ）

主に特権モード
デバイスドライバ・ミドルウェア
HRP2カーネル

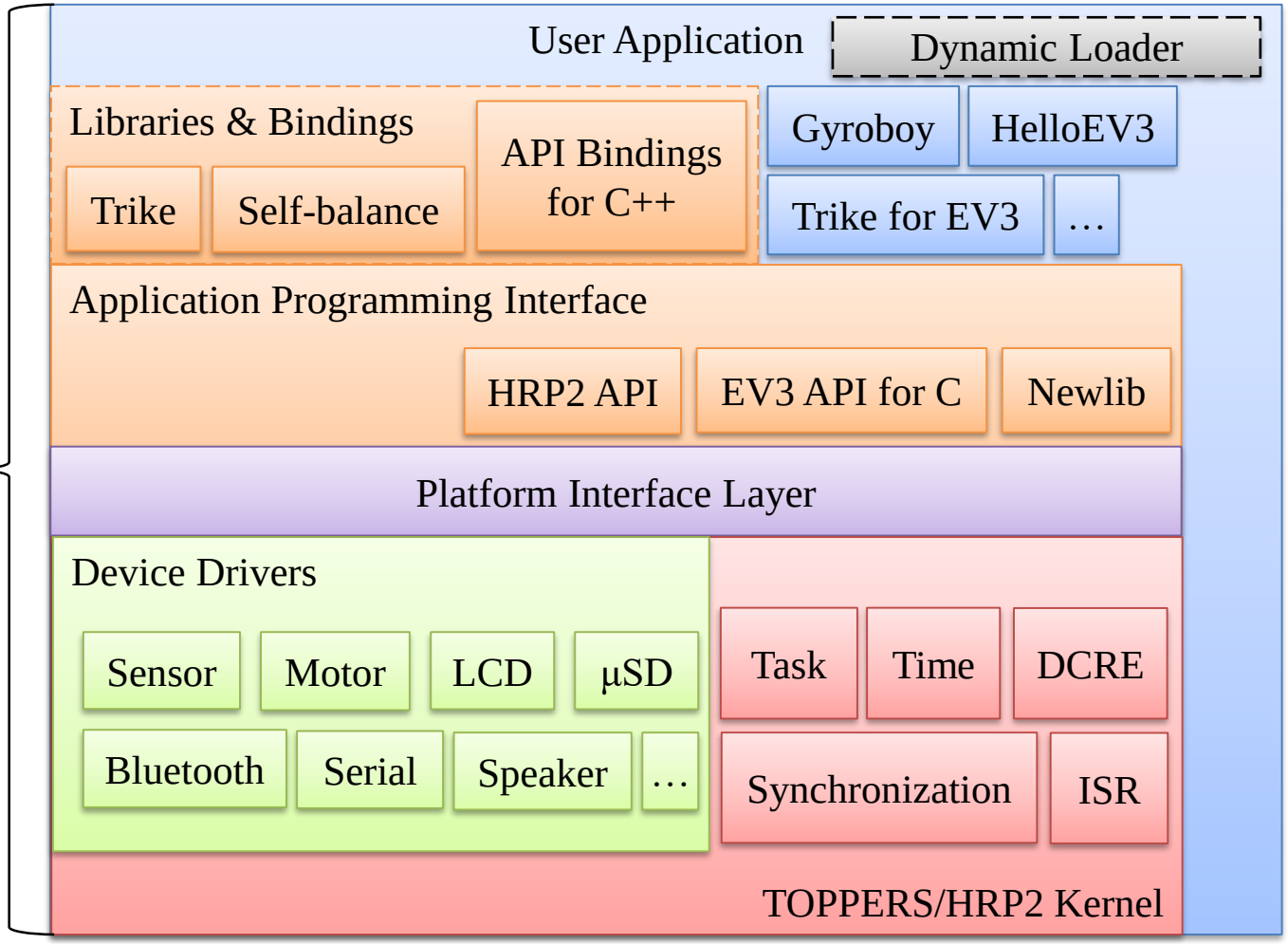
ユーザアプリ

非特権モード
ハイレベルAPI
専用ユーザドメイン
（TDOM_APP）

EV3RTのアーキテクチャ

- スタンドアローン形式の場合のアーキテクチャ

任意のドメイン
特権/非特権
全機能利用可能



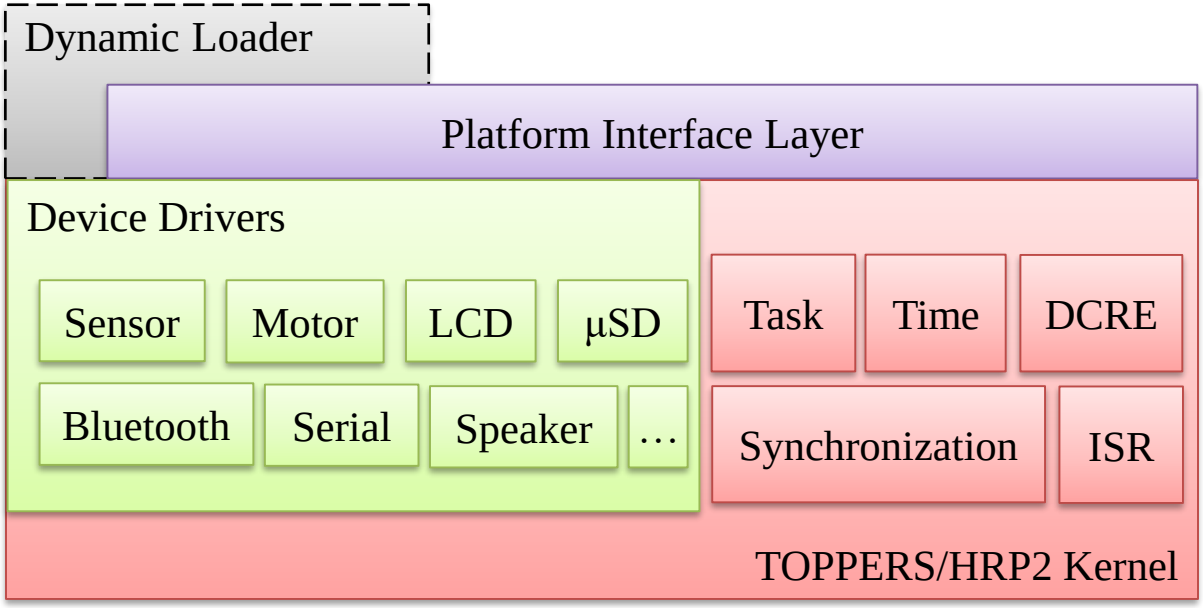
※開発者が自由に改造できるため、動的形式に基づいて解説

発表内容

- EV3RTの内部構造の解説
  - アーキテクチャの概要
  - ベースシステムの解説
  - ユーザアプリケーションとSDK
- 動的ローディング機能
  - Dynamic Module Loader
  - ユーザアプリのモジュール
  - ローディング操作に必要な処理
- 進んだ使い方
  - スタンドアローン形式
  - EV3RT/HRP3と新しい機能

EV3RTのベースシステム

- ユーザアプリの実行をサポートするためのサービスを提供するシステムソフトウェア
- 構成
  - RTOSカーネル
  - 各種ドライバ
  - Platform I/F
  - ミドルウェア
    (e.g. Loader)



TOPPERS/HRP2カーネル

- 特徴
  - 静的RTOS
    - カーネルオブジェクト等は静的APIでcfgファイルに定義
      - ✓例：タスク、セマフォ…
    - 必要なリソースはコンパイル時決定
  - 保護機能
    - メモリアクセスの保護
    - カーネルオブジェクトのアクセス保護
  - 拡張サービスコール機能
    - 例：ユーザアプリから特権モードのドライバを安全にアクセスする
  - 動的生成機能拡張（DCRE、Dynamic CReation Extension）

TOPPERS/HRP2カーネル

- HRP2カーネルの動的生成機能拡張（DCRE）
  - C言語でのカーネルオブジェクトの動的生成が可能に

```
ER_ID tskid = acre_tsk(pk_ctsk); // タスクを生成
```

  - Cfgで動的に生成できるリソースの最大数を静的指定・確保

```
AID_TSK(32); // 最大32個のタスクが生成できる
```

  - 細粒度のアクセス制御はできない
    - 任意のドメインに属するオブジェクトを生成できる
      - ✓例：ユーザアプリがカーネルのタスクを作成して、保護機能回避
    - EV3RTでは、ユーザアプリからDCREを使用できない
  - スタンドアローン形式でカーネルドメインから使用できる
    - 例：ローダがユーザアプリのオブジェクトを生成する

Linuxデバイスドライバの再利用

- 課題：センサ、モータ等EV3専用のデバイスのサポート
  - 機能向上に伴って、ドライバの規模が膨大に
  - 詳細仕様に関するドキュメントは公開されていない
    - Linuxベースの標準環境（lms2012）のソースコードはある
  - ゼロから実装するのは時間的に無理

- 対応：Linuxのデバイスドライバを再利用
  - 専用デバイスドライバはLinux独自の機能をほぼ使ってない
    - 殆どのOSカーネルに存在している基本機能で開発されている
  - 簡易のLinux Kernel Driver Interface互換レイヤを作成
    - TOPPERSカーネルで一部のLinuxカーネル内の機能を実装
      - ✓例：メモリ管理、タスク同期、割込みハンドラ、タイマ

Linuxデバイスドライバの再利用

- 互換レイヤを使ってLinuxドライバを導入
  - PWM制御モジュールの例

```
// Include the common part for a driver
#include "driver_common.h"
```

共通なヘッダファイル
Linux Kernel Driver Interface

```
// Hacks for this module
#define InitGpio PWM_InitGpio
static void SetGpioRisingIrq(...) {
...
}
```

モジュール毎のHacks
名前衝突を回避するマクロ定義
モジュール内の関数の再実装
（GPIO割込みの対応等）…

```
// Include the source file to reuse
#include "d_pwm.c"
```

Linuxドライバの再利用
保守性を向上するため、原則として、コードは変更しない
不要な部分をコメントアウト

```
// Init
static void initialize(...) {
...
    ModuleInit();
...
}
```

初期化関数の作成
モジュールの初期化関数をEV3RT起動時に呼び出す

Linuxデバイスドライバの再利用

- 再利用できたデバイスドライバ
  - LCD
  - PWM制御モジュール（モータ）
  - Analog入出力制御モジュール（Analogセンサとバッテリー）
  - UARTセンサ
  - Soft UARTポート
  - Soft I2Cポート
  - スピーカ

32,183行 ⇒ 1,033行

95%以上の作業量削減

- オーバヘッド評価（モータドライバの場合）

Function	Average time	Maximum
ev3_motor_set_speed()	4.54 us	40 us
ev3_motor_set_speed()[Linux]	596 us	3424 us

ユーザアプリのボタンハンドラ機能

- EV3RTにボタンハンドラを登録するAPIがある
  - ボタンがクリックされたら、ユーザアプリの関数を呼び出す
- 課題：クリックイベントの処理コンテキスト
  - クリックイベントのトリガーはGPIO割込み
    - 割込み処理からユーザアプリの関数を呼び出すと保護が効かない
- 対応：ボタンハンドラ用のイベントフラグとタスクを用意
  - ボタンのGPIO割込みでイベントフラグをセット
    - 例：iset_flg(current_button_flag, 1 << BRICK_BUTTON_BACK)
  - ユーザドメインのタスクでフラグをチェックしてハンドラを実行
    - wai_flg(BTN_CLICK_FLG, …) → handlers[i](exinfs[i])
    - ハンドラ実行時、保護機能が適用されている
    - ボタンハンドラ用タスクの優先度はユーザアプリよりも高い
      - ✓処理が長いハンドラ関数を使う時は要注意

ユーザアプリの周期ハンドラ機能

- 課題：HRP2の標準の周期ハンドラは割込みで処理
  - ボタンハンドラと違って数や定義は固定されていない
    - ボタンハンドラと同じ手法で対応できない
- 対応：アプリのハンドラ毎に標準ハンドラとタスク生成
  - HRP2の標準周期ハンドラで生成したユーザタスクを起動
  - 生成したタスクでアプリから登録したハンドラ関数を実行
    - 標準の周期ハンドラと違って、タスクコンテキストで実行
    - 優先度はユーザタスクより高い
- ユーザアプリの周期ハンドラ用API

```
EV3_CRE_CYC(...) // 静的API、パラメータはCRE_CYCと同じ
ev3_sta_cyc(...) // ユーザドメイン周期ハンドラの動作開始
ev3_stp_cyc(...) // ユーザドメイン周期ハンドラの動作停止
```

ユーザアプリの周期ハンドラ機能

- 生成について
  - 静的API定義は動的に生成するコードに変換される

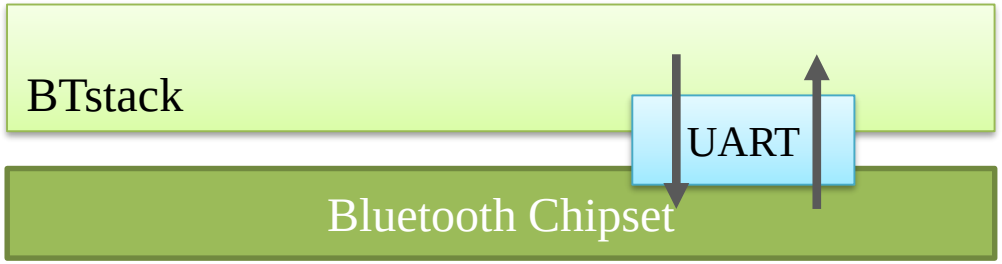
```
EV3_CRE_CYC(EV3_CYC1,{TA_STA,0,test_ev3_cychdr,500,0});
```

```
void _initialize_ev3api_cyc() {
    ...
    pk_ccyc.cycatr = TA_STA; pk_ccyc.exinf = 0;
    pk_ccyc.cychdr = test_ev3_cychdr;
    pk_ccyc.cyctim = 500; pk_ccyc.cycphs = 0;
    ercd = _ev3_acre_cyc(&pk_ccyc);//動的生成の拡張SVC
    _ev3api_id_EV3_CYC1 = ercd; }
```

  - アプリの初期化タスクで呼び出される ⇑

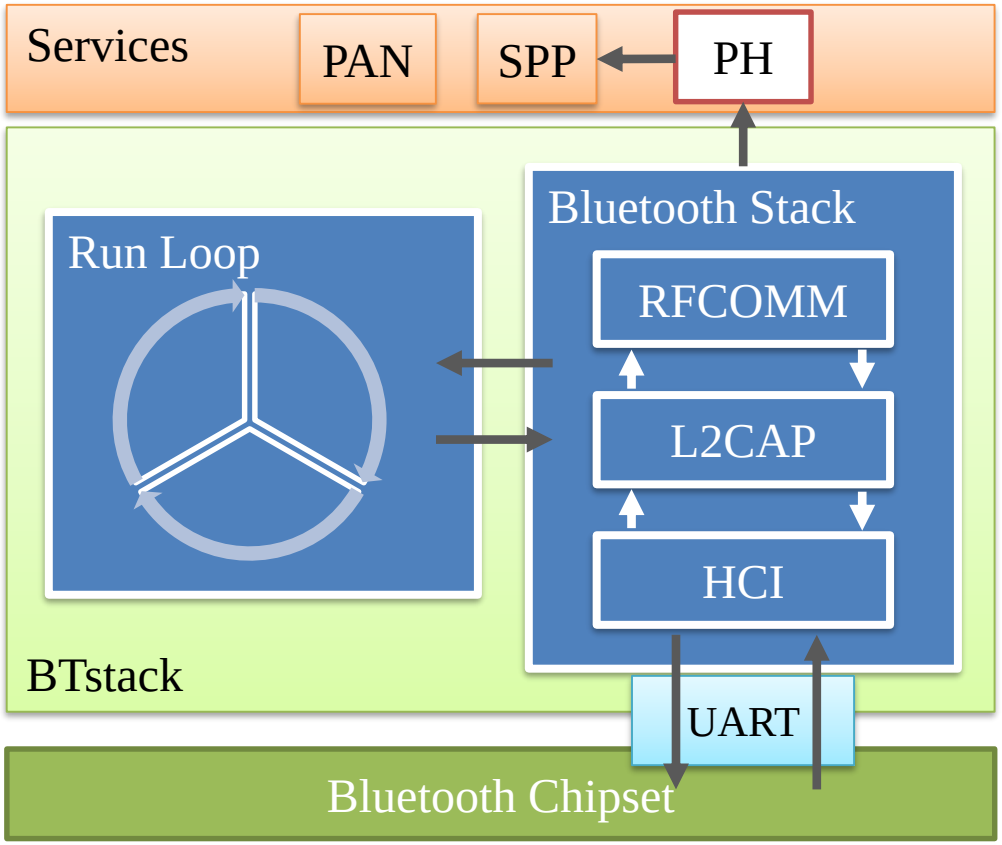
Bluetooth通信機能

- BTstackというオープンソースのスタックドライバを使用
  - 特徴
    - 軽量で組込み向け、OSなしでも動作できる
    - 機能が豊富、Bluetooth SIGにより認証済み
    - 対応するチップが多い、EV3のCC256x（v2.1 EDR）もサポートする
    - EV3RT用fork： https://github.com/ev3rt-git/btstack
  - CC256xとSoCはUARTで通信するので移植はとても簡単
    - UART操作関数でBTstackのHCIインタフェースを実現したら良い
    - UARTドライバの実装にDMAを使って性能向上、CPU負荷を軽減



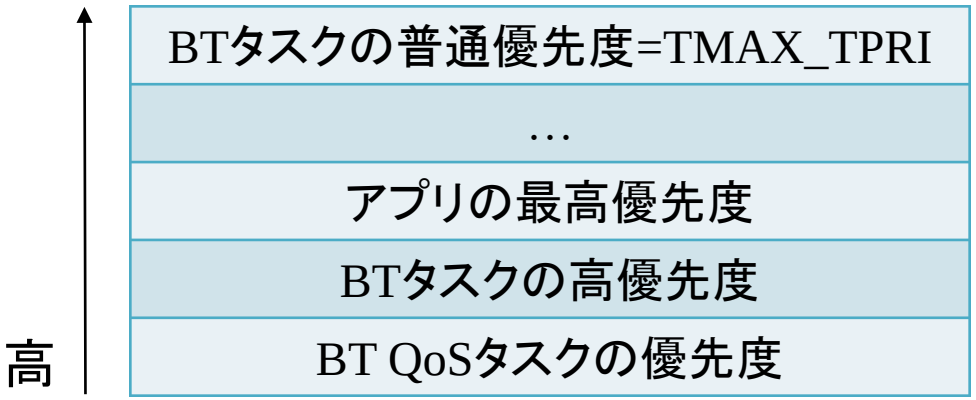
Bluetooth通信機能

- BTstackのAPIを使ってサービスを実装
  - SPP（Serial Port Profile）とPAN（Personal Area Network）対応
  - SPPの仮想シリアルはTOPPERS SIOドライバでも使えるようにした。接続された時にopen、切れたらcloseされる
  - 受送信データを処理するためにRun Loopが必要。Run Loop用のBTタスクを作成した



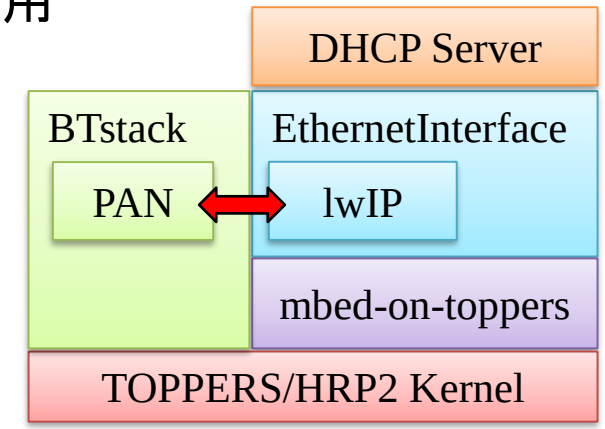
Bluetooth通信機能

- Bluetooth通信のQoS確保
  - 課題：BTタスクの優先度
    - アプリより高い優先度で動作すると
      - ✓高速で通信する場合、アプリのリアルタイム性に悪影響
    - アプリより低い優先度で動作すると
      - ✓スタベーション（例：無限ループ）で接続が切れてしまう可能性がある
  - 接続を保護するために、QoSタスクで周期的にBTタスクの優先度を変更
    - 現時点のポリシー　20ミリ秒ごとに1ミリ、BTタスクをアプリより高い優先度で実行させる
      - ✓参考：倒立制御処理　5ミリ秒周期で動作、実行時間は1ミリ秒以内



Bluetooth通信機能

- Bluetooth PANでネットワーク通信（TCP/IP）を対応
  - HTTP経由のアプリ更新に必要な機能
    - Beta 7以前はSPPのみ対応、アップロードの時、毎回Tera Term等からファイルやSPPの番号（環境依存）を指定する必要がある
    - Beta 7からPAN対応、「make upload」だけで簡単にアップロードできる
  - 課題：TOPPERS/HRP2用DHCPサーバのミドルウェアがない
    - EV3とPCのIPアドレスを自動設定するために必要
  - ARM mbed OS互換レイヤ（mbed-on-toppers）で対応
    - TCP/IPスタック（lwIP）とDHCPサーバを使用
    - BTstack PANとlwIPのbridgeを実装
  - 現時点、PANはアプリ更新専用
    - ユーザアプリから使用できない



その他の主なデバイスドライバ・ミドルウェア

- Console機能（drivers/console）
  - LCDでの操作メニューとログ表示

- ファイルシステムとSDカード（drivers/fatfs）
  - 汎用FATファイルシステムモジュール FatFS
    - Windows互換FATファイルシステムをサポート
    - 下位レイヤインターフェースを実装して、SDカードを操作できる

その他の主なデバイスドライバ・ミドルウェア

- USB Mass Storage Class（drivers/usbmsc）
  - SoCメーカが提供したUSBスタックTI StarterWare USBで開発
  - EV3のMini USBをPCに接続、SDカードリーダとして使える

- 設定ファイル機能（platform/src/ev3rt_config.c）
  - 一部の設定（例：Bluetooth PIN）をrc.conf.iniで変更できる
    - ベースシステムのリコンパイルが不要
  - INIファイルの解析にminIniというライブラリを使用

- バッテリ残量警告
  - 周期ハンドラEV3_BATTERY_MONITOR_CYCで電圧をチェック。低下の時、警告メッセージ出力＋LEDを赤に設定

その他の主なデバイスドライバ・ミドルウェア

- プラットフォーム初期化（platform/src/ev3main.c）
  - デバイスドライバ等を適切な順序で初期化する
  - EV3RTの起動バナーを表示

- TLSFメモリアロケータ（ev3_gcc/TLSF-2.4.6）
  - リアルタイムシステム向けの動的メモリアロケータ
    - O(1)の計算量でmalloc/free
  - 動的生成機能拡張と標準Cライブラリのために導入
  - 複数のメモリプールを対応
    - ベースシステム用メモリプールとアプリ用メモリプールを別々に管理できるのでHRP2のメモリ保護機能との相性が良い

Platform Interface Layer

- ユーザアプリからベースシステムのサービスを安全にアクセスするためのインターフェース
  - 共有メモリ（brickinfo_t）と拡張サービスコールから構成

- 保守性向上のために、比較的に安定している
  - ABI（Application Binary Interface）の概念に近い
  - PILにバージョン番号がある
    - ベースシステムは同じPILバージョンを持つアプリのみサポートする
      - ✓Beta 6-2（PIL Ver.5）のアプリはBeta 7（PIL Ver. 6）で実行できない
    - 同じPIL番号のベースシステム間はアプリのバイナリ互換性を持つ
      - ✓Bug fixの場合、アプリをリコンパイル必要がない e.g. β6-2→β6-3

Platform Interface Layer

- 共有メモリ(brickinfo_t)のインタフェース
  - アプリから直接アクセス可能のメモリ領域ポインタの集合
  - ベースシステムはドライバの初期化でポインタ値を設定
  - ユーザアプリは起動時、SVC _fetch_brick_info()でベースシステムから取得
  - オーバヘッドは拡張サービスコールよりずっと小さい

```
typedef struct {
    uart_data_t      *uart_sensors;
    analog_data_t    *analog_sensors;
    i2c_data_t       *i2c_sensors;
    motor_data_t     *motor_data;
    uint8_t          *motor_ready;
    volatile bool_t  *button_pressed;
    bitmap_t         *lcd_screen;
    font_t           *font_w6h8;
    font_t           *font_w10h16;
    void             *app_heap;
    int16_t          *motor_current;
    int16_t          *battery_current;
    int16_t          *battery_voltage;
    int16_t          *battery_temp;
} brickinfo_t;
```

Platform Interface Layer

- 拡張サービスコールのインタフェース
  - モータ制御等共有メモリだけでは実現できない機能がある
    - 特権モードでコードを実行する必要がある
  - ベースシステムに拡張サービスコールを生成・実装

```
#define TFN_FETCH_BRICK_INFO (24)
ER_UINT extsvc_fetch_brick_info(intptr_t p_brickinfo, intptr_t par2,
        intptr_t par3, intptr_t par4, intptr_t par5, ID cdmid) {
    return _fetch_brick_info((brickinfo_t*)p_brickinfo, cdmid);
}
// DEF_SVC(TFN_FETCH_BRICK_INFO, { TA_NULL, extsvc_fetch_brick_info, 1024 });
```

  - PILにアプリから拡張サービスコールを呼び出す関数を定義

```
static inline ER fetch_brick_info(brickinfo_t *p_brickinfo) {
    ER_UINT ercd = cal_svc(TFN_FETCH_BRICK_INFO, (intptr_t)p_brickinfo, (intptr_t)0, (intptr_t)0, 0, 0);
    return ercd;
}
```

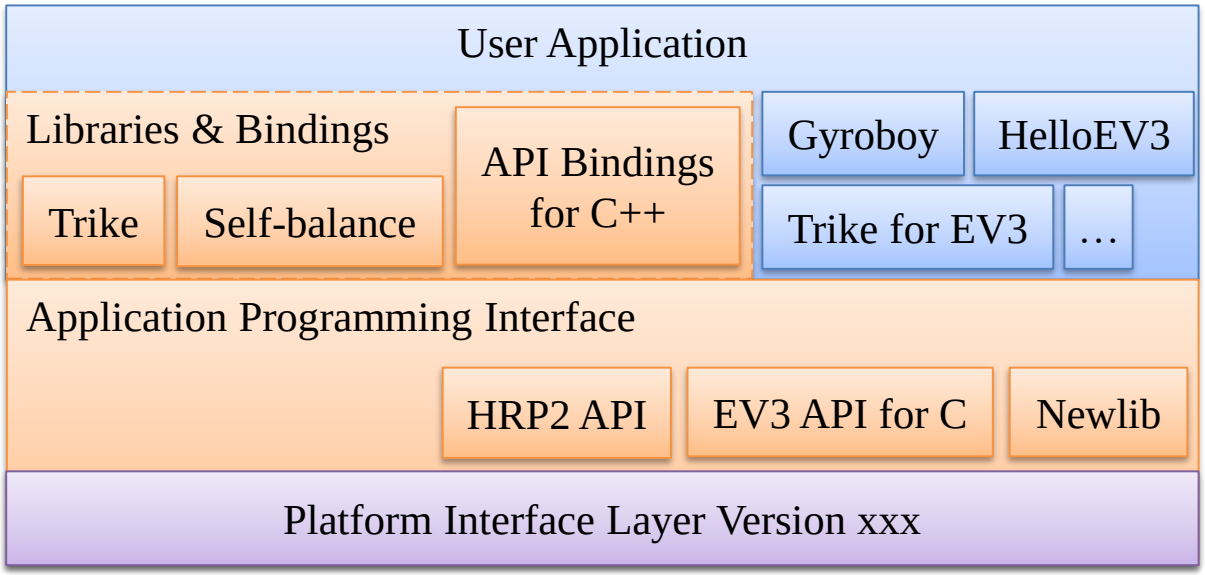
  - ユーザアプリのコンパイルはベースシステムに依存しない
    - ✓PILと一緒に独立したモジュールとしてビルド可能

発表内容

- EV3RTの内部構造の解説
  - アーキテクチャの概要
  - ベースシステムの解説
  - ユーザアプリケーションとSDK
- 動的ローディング機能
  - Dynamic Module Loader
  - ユーザアプリのモジュール
  - ローディング操作に必要な処理
- 進んだ使い方
  - スタンドアローン形式
  - EV3RT/HRP3と新しい機能

ユーザアプリケーションとSDK

- ユーザアプリ：Mindstorms EV3を制御するプログラム
  - EV3RTが提供するソフトウェア開発キット（SDK）で実装
  - 特定のユーザドメイン（TDOM_APP）で動作
  - 動的ローディング用のアプリモジュールとしてビルド可能
- ソフトウェア開発キット（SDK）の構成
  - API
  - 静的ライブラリ
  - サンプル（& Workspace）



Loadable Application Module

Application Programming Interface

- TOPPERS/HRP2カーネルAPI
  - 静的APIとサービスコール
  - RTOS機能（タスク管理、同期・通信、タイマ等）
- 標準Cライブラリ（Newlib）
  - 動的メモリ管理、ファイル操作、入出力関数等
- EV3RT C言語 API
  - モータ、センサ、スピーカ、LCD等EV3RT独自の機能
  - Platform Interface Layerを用いて実装
  - ソースコードはsdk/common/ev3apiにある

Application Programming Interface

- 標準Cライブラリ（Newlib）
  - 方針：各種ミドルウェアのAPIの違いを吸収
    - メリット：開発者の学習コストを削減できる
    - デメリット：ユーザアプリはミドルウェアのフル機能を使えない
  - TLSFアロケータ → malloc等メモリ管理の標準関数
  - FatFS/SDカード → fopen/read等ファイルI/Oの標準関数
  - Bluetooth SPP → 通信用のファイル（SIO_BT_FILENO）
    - fprintf/fscanf等を使って簡単に仮想シリアルポートで通信できる
    - TOPPERSのシリアル関数でも通信可能（sio_opn_por等）

静的ライブラリ

- アプリが共通して利用したい機能をライブラリで拡張
  - EV3RTはライブラリを簡単に開発・導入できる機能を持つ
    - ライブラリのプロジェクトフォルダはsdk/common/libの下に

```
sdk/common/library/libcpp-ev3   ETロボコン用C++ライブラリ
sdk/common/library/libcpp-test  サンプルライブラリ
```

    - アプリのMakefile（ev3way-cpp/Makefile.incの例）で導入できる

```
...
# Include libraries
include $(EV3RT_SDK_LIB_DIR)/libcpp-ev3/Makefile
...
```

静的ライブラリ

- 静的ライブラリの開発
  - ライブラリ・プロジェクト・フォルダの構成

```
Makefile                          ライブラリのMakefile
src/*                             ライブラリのソースコード
include/*                         アプリが使えるヘッダファイル
<libname>-[loadable|standalone].a 静的lib（オプション）
```

  - ライブラリのMakefile

```
THIS_LIB_NAME := <libname>          # ライブラリの名前
THIS_LIB_OBJS := c-1.o c-2.o        # C言語オブジェクト
THIS_LIB_CXXOBJS := cpp-1.o cpp-2.o # C++オブジェクト
include $(EV3RT_SDK_LIB_DIR)/../Makefile.slib
```

  - 静的ライブラリ（.a）が無い場合、アプリのビルドで自動生成

ユーザアプリケーション

- 実行時の主なタスク構成

優先度	タスク	周期	実行時間
3	Bluetooth（高）	20ms	<1ms
4	LCDリフレッシュ	40ms	<1ms
TMIN_APP_TPRI - 1	ユーザ周期ハンドラ	User-defined	User-defined
>= TMIN_APP_TPRI	ユーザ アプリケーション		
TMAX_TPRI	Bluetooth（低）		

  – 汎用OSよりBackground Tasksが少なくリアルタイム性が良い

ユーザアプリケーション

- 初期化タスクでアプリの実行に必要な処理を行う

```
void _app_init_task(intptr_t unused) {
    brick_misc_command(CMD_BUSY_WAIT_INIT, 0);

    ER ercd = fetch_brick_info(&_global_ev3_brick_info);

    _initialize_ev3api_lcd();
    _initialize_ev3api_motor();
    _initialize_ev3api_newlib();
    _initialize_ev3api_sensor();

    // Call C++ global constructors
    extern func_ptr __init_array_start[0], __init_array_end[0];
    for (func_ptr* func = __init_array_start; func != __init_array_end; func++)
        (*func)();

    // EV3_CRE_CYC
    _initialize_ev3api_cyc();
}
```

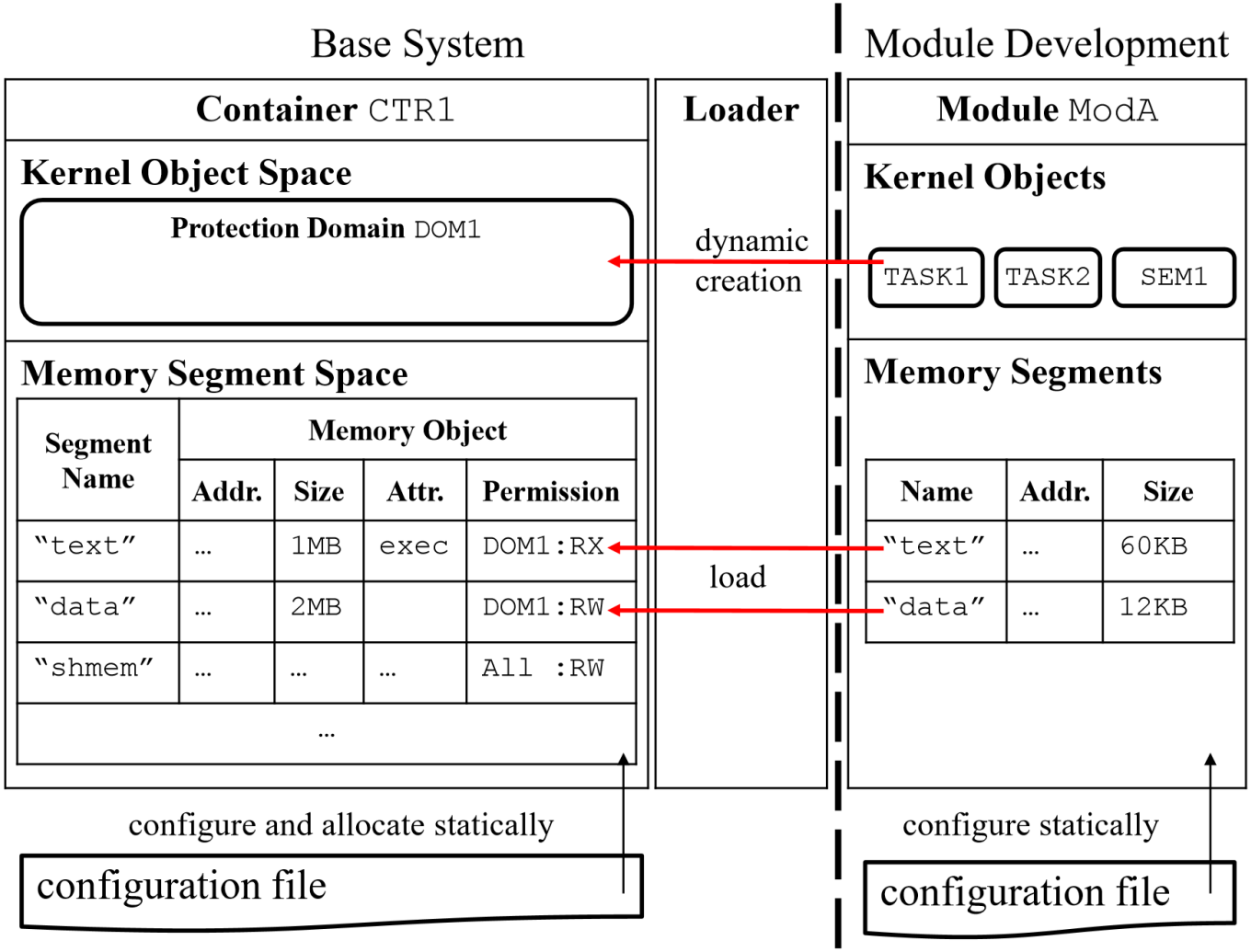
  - SDK内部のタスク、ユーザで作成・定義する必要がない
  - 注意：C++ Global Constructorは待ちになるコードを使えない

発表内容

- EV3RTの内部構造の解説
  - アーキテクチャの概要
  - ベースシステムの解説
  - ユーザアプリケーションとSDK
- 動的ローディング機能
  - Dynamic Module Loader
  - ユーザアプリのモジュール
  - ローディング操作に必要な処理
- 進んだ使い方
  - スタンドアローン形式
  - EV3RT/HRP3と新しい機能

動的ローディング機能

- Dynamic Module Loaderの概略図



Dynamic Module Loader

- 静的OS用のモジュールを動的に追加・削除する機能
- モジュール：EV3RTの場合はユーザアプリ
  - 静的APIで開発　→　必要なリソースはコンパイル時に決定
  - 全てのカーネルオブジェクトは同じ保護ドメインに属する
    - 汎用OSのプロセスに似ている
- コンテナ：モジュール実行用、ベースシステムで定義
  - モジュールが使える最大リソース（タスク等）を静的に確保
  - モジュールの保護機能（メモリアクセス権等）を静的に設定
- ローダ：DCREを使って、モジュールの実行ファイルを指定のコンテナに展開して、動作を開始する

ユーザアプリのモジュール

- 実行ファイルの形式
  - 独立のELFファイル
    - ベースシステムのコードに依存しない → 動的シンボル解決は不要
    - PIL、API、ライブラリは静的にリンク → 動的リンク機構は不要
    - 単一のドメイン（TDOM_APP）に属するため、コードとデータのメモリ保護設定は単純で、textセクションとdataセクションがあれば十分
  - Module Configuration TableをELFファイルに格納する
    - モジュールの静的API（cfgファイル）のデータ構造
    - ローダでモジュールのオブジェクトを動的に生成するために必要
  - 完全位置独立コード（-mno-pic-data-is-text-relative）でビルド
    - ローディング用メモリ領域はベースシステムの設定により変化する
  - 課題：実際のオブジェクトIDの取得
    - ユーザアプリに使うIDの実際の値はロードしないと分からない

Module Configuration Table

- Module Configuration Entryの配列

```
typedef struct {
    FN sfncd; const void *argument; void *retvalptr;
} MOD_CFG_ENTRY;
```

  - sfncd：静的APIの機能コード　e.g. TSFN_CRE_TSK/SEM/FLG/MTX
  - argument：引数へのポインタ、生成情報のパケットの場合が多い
  - retvalptr：　動的生成の関数の戻り値を格納する変数へのポインタ

– 静的API　CRE_TSKの例：

```
CRE_TSK(APP_INIT_TASK,
        { TA_ACT, 0, _app_init_task, TPRI_APP_INIT_TASK, STACK_SIZE, NULL });
```

```
ID _module_id_APP_INIT_TASK __attribute__((section (".module.text")));
static const T_CTSK _module_ctsk_tab[] = {
    { TA_ACT, 0, _app_init_task, TPRI_APP_INIT_TASK, ROUND_STK_T(STACK_SIZE), ... },
};
const MOD_CFG_ENTRY _module_cfg_tab[] = {
    { TSFN_CRE_TSK, &_module_ctsk_tab[0], &_module_id_APP_INIT_TASK },
};
```

オブジェクトIDの生成と取得

- モジュール開発環境とローダ内部処理の流れ
  1. オブジェクトIDのマクロと格納するための変数を定義

```
extern ID _module_id_APP_INIT_TASK;
#define APP_INIT_TASK ((const ID)(_module_id_APP_INIT_TASK))
```

  2. ID用変数のポインタをMod Cfg Entryの戻り値として登録

```
const MOD_CFG_ENTRY _module_cfg_tab[] = {
    { TSFN_CRE_TSK, &_module_ctsk_tab[0], &_module_id_APP_INIT_TASK },
};
```

  3. ローダが動的生成関数（acre_tsk等）を呼び出してオブジェクトIDをMod Cfg Entryの戻り値として設定
  4. ユーザアプリは変数のマクロでIDを参照しているため、実際の値を使用することになる

アプリローダ

- ローディング操作の流れ
  0. コンテナがモジュールのセクションを格納する領域を用意

```
static STK_T app_text_mempool[COUNT_STK_T(TMAX_APP_TEXT_SIZE)]
        __attribute__((section(".app_text_mempool"),nocommon));
static STK_T app_data_mempool[COUNT_STK_T(TMAX_APP_DATA_SIZE)]
        __attribute__((section(".app_data_mempool"),nocommon));
```

  1. ELFファイルのセクションを上記の領域にコピー
  2. 完全位置独立コードのrelocationを行う
  3. Module Cfg Tableを参照して順次にオブジェクトを生成
  4. 生成したオブジェクトのIDをretvalptrに格納
  5. 全てのオブジェクトが生成されたら、ロード完了
  6. 生成に失敗するオブジェクトがあったら、各retvalptrを参考して今まで生成した全てのオブジェクトを削除

※Unloadは生成したタスクを強制終了してオブジェクトを削除

発表内容

- EV3RTの内部構造の解説
  - アーキテクチャの概要
  - ベースシステムの解説
  - ユーザアプリケーションとSDK
- 動的ローディング機能
  - Dynamic Module Loader
  - ユーザアプリのモジュール
  - ローディング操作に必要な処理
- 進んだ使い方
  - スタンドアローン形式
  - EV3RT/HRP3と新しい機能

スタンドアローン形式

- より高度な開発ができる、例：
  - プラットフォームの開発・拡張
    - application loader
    - mruby on ev3rt+tecs
  - ベースシステムのカスタマイズ
  - デバイスドライバ・ミドルウェアの直接アクセス

スタンドアローン形式

- ベースシステムのカスタマイズ
  - 動的ローディングの制限を変更する時はリビルトが必要
    - 静的に生成されたリソースの情報（ページテーブル等）が変わった
    - 設定ファイルを書き換えるだけで対応できない

```
$ cd base-workspace; make app=loader
```

  - カスタマイズ可能な項目はev3_gcc/ev3.hにまとめてある

```
#define TMAX_APP_TSK_NUM     (32)
#define TMAX_APP_SEM_NUM     (16)
#define TMAX_APP_FLG_NUM     (16)
#define TMAX_APP_DTQ_NUM     (16)
…
#define TMAX_APP_TEXT_SIZE   (1024 * 1024)
#define TMAX_APP_DATA_SIZE   (1024 * 1024)
#define TMAX_APP_BINARY_SIZE (1024 * 1024)
```

e.g.
Application
Moduleの
リソース制限

スタンドアローン形式

- デバイスドライバ・ミドルウェアへの直接アクセス
  - アプリ開発の利便性を向上するために、BluetoothやFatFSのようなミドルウェアは標準Cライブラリ経由で使う
    - 学習コストを削減できたが、フル機能は使用できない
  - 例：標準BluetoothスタックのBTstackはSPP、PANの他
    - Bluetooth HID => キーボード、ゲームコントローラ等
    - Bluetooth OBEX => ファイル転送サービス
      等様々なプロトコルも対応
  - 例：lwIPでTCP/IP通信

EV3RT/HRP3

- TOPPERS第3世代カーネルHRP3を使用するEV3RT
  - テストリリース: https://github.com/ev3rt-git/ev3rt-hrp3
  - 現時点、HRP2版と同等の機能を全て対応している
  - 今年度のETロボコンの公式プラットフォームではない
  - HRP3カーネルに色々な機能が追加されたが、開発者にとって一番大きい変更は高分解能(μs単位)時間管理
    - 1ms sleepの場合:HRP2でtslp_tsk(1) → HRP3でtslp_tsk(1000)

- 今後の方針
  - HRP2版:不具合の修正を行う
  - HRP3版:新しい機能の追加等

EV3RT/HRP3の新しい機能

- SPPマスタ機能
  - 従来のEV3RTはSPPサービスを提供する側
    - 例：PCからEV3に接続して、Bluetoothでシリアル通信を行う
  - SPPマスタは他のSPPサービスへ接続する機能
  - 関係するAPI

```
// 既存のSPPマスタによる接続をリセットする
ER ev3_spp_master_reset();
// SPPマスタによる接続を開始する
ER ev3_spp_master_connect(addr, pin, service);
// SPPマスタとリモートサービスの接続状態をチェックする
bool_t ev3_spp_master_is_connected();
// SPPマスタで通信用のファイルをオープンする
FILE* ev3_serial_open_file(EV3_SERIAL_SPP_MASTER);
```

EV3RT/HRP3の新しい機能

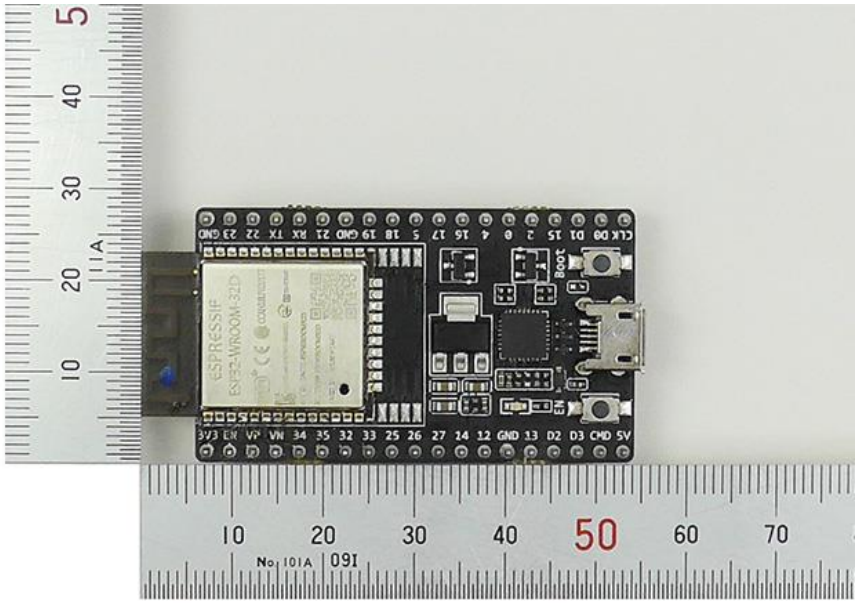
- SPPマスタ機能でEV3間通信
  - サンプルアプリ（sdk/workspace/test-spp-master）を提供
    - HRP2版もSPPマスタのアプリを実験的に提供している
    - HRP3版は更に汎用化・安定したサポート
  - マスタとして他のEV3へ接続

```
FILE *bt_slave = ev3_serial_open_file(EV3_SERIAL_SPP_MASTER);
ev3_spp_master_reset();
ev3_spp_master_connect(remote_addr, remote_pincode, "Serial Port");
while (!ev3_spp_master_is_connected()) {
    dly_tsk(100U * 1000U);
}
fprintf(bt_slave, "MOTOR %d\n", power10 / 10);
```

    - 通信はEV3↔PCと同様にファイル操作で行う
    - EV3のBluetoothのMACアドレスは起動時のログから確認できる

EV3RT/HRP3の新しい機能

- SPPマスタ機能でESP32系ボードと通信
  - ESP32：IoT開発で流行っている開発ボードシリーズ
    - Wi-FiとBluetooth内蔵、低コストで入手可能
    - USBで給電・UART通信・プログラムをダウンロードできる
    - 例：ESP32-DevKitC-32D
      - ✓1,500円程度
      - ✓サイズ小さい（5.5cm x 3cm）



図はhttp://akizukidenshi.com/catalog/g/gM-13628/により

EV3RT/HRP3の新しい機能

- SPPマスタ機能でESP32系ボードと通信
  - ESP32はATコマンドをサポートし、シリアルでWiFi通信可能
    - 例えば、シリアルで「AT+CWLAP\r\n」コマンドを送信すると接続範囲内のアクセスポイントのリストが出力される

```
+CWLAP:(3,"AP1",-37,"ab:cd:ef:65:43:31",11)
+CWLAP:(3,"AP2",-37,"ab:cd:ef:65:43:32",11)
OK
```

    - ATコマンドの詳細は「ESP32 AT Instruction Set and Examples」（https://www.espressif.com/sites/default/files/documentation/esp32_at_instruction_set_and_examples_en.pdf ）
  - ESP32標準（公式）のATコマンドアプリは物理シリアルのみ
    - EV3RTと通信するために、Bluetooth SPPでも使えるように拡張した
    - ほぼ全ての（ダウンロード用のUSBポートがある）ESP32ボードに対応
    - https://github.com/ev3rt-git/esp32-at-via-bluetooth

EV3RT/HRP3の新しい機能

- SPPマスタ機能でESP32系ボードと通信
  - ESP32に接続して、ATコマンドで通信するサンプルを提供
    - sdk/workspace/test-esp32-spp
  - EV3のUSBに接続して、WiFiドングル的な使い方が可能
    - 同じネットワーク内の複数のEV3・PC間で通信
    - インターネットへの接続
  - ただし、ATコマンドの解析等はやや複雑である
    - EV3RTとして、更に使いやすい機能を近日提供する予定
      - ✓指定したWiFiへの接続はプラットフォーム側で行う
      - ✓ユーザはATコマンドを意識せず、socket風で通信
      - ✓WiFi経由のアプリ更新等も検討している

ご清聴ありがとうございました