# EV3RTの内部構造と進んだ使い方

Beta 7

２０１７年６月１７日

名古屋大学大学院情報科学研究科
高田・本田研究室　博士課程３年
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
- 動作ローディング機能
  - Dynamic Module Loader
  - アプリケーションモジュール
  - アプリのローディングの流れ
- 進んだ使い方

# 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
- 動作ローディング機能
  - Dynamic Module Loader
  - アプリケーションモジュール
  - アプリのローディングの流れ
- 進んだ使い方

# EV3RTの二つの実行形式

- 動的ローディング形式
  - 普通のユーザアプリを開発する時に推奨
    - 特権モードのドライバAPI等ローレベルの機能は使用不可
  - アプリはローダで動的に実行できるモジュールとしてビルド
  - SDカード、Bluetoothからロードできる
    - USBのSDカードリーダ機能で実行中にアプリをアップロードできる
    - Beta7:「make upload」で簡単にBluetooth PAN経由でアップロード可能
- スタンドアローン形式
  - プラットフォーム開発等フル機能を使いたい場合に有用
    - 動的のアプリローダ自体もこの形式のアプリの一つである
  - アプリとベースシステムは一つのバイナリとしてビルド
  - EV3ブートローダが直接にサポートする実行形式（uImage）
    - SDカードのルートフォルダに入れることで動作可能

# 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
```

- ビルド方式はマクロで判定できる
  - 普通の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

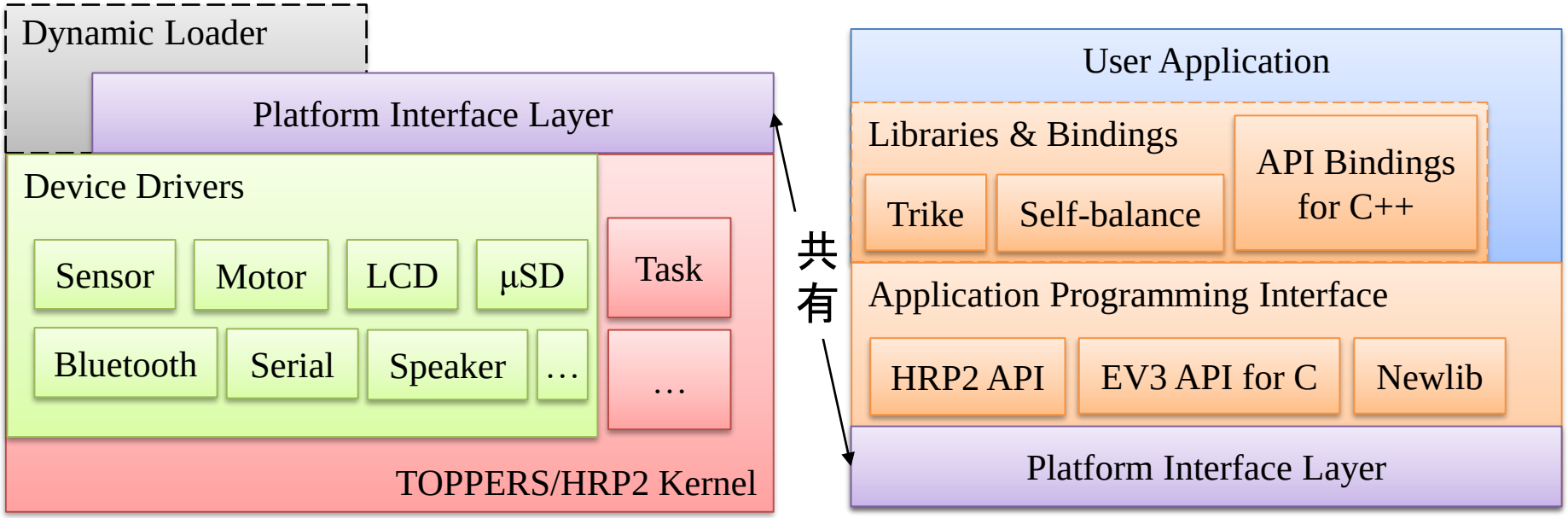
```
#if defined(BUILD_MODULE)
#include "module_cfg.h"
#else
#include "kernel_cfg.h"
#endif
```

```
#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
uImageとしてビルド

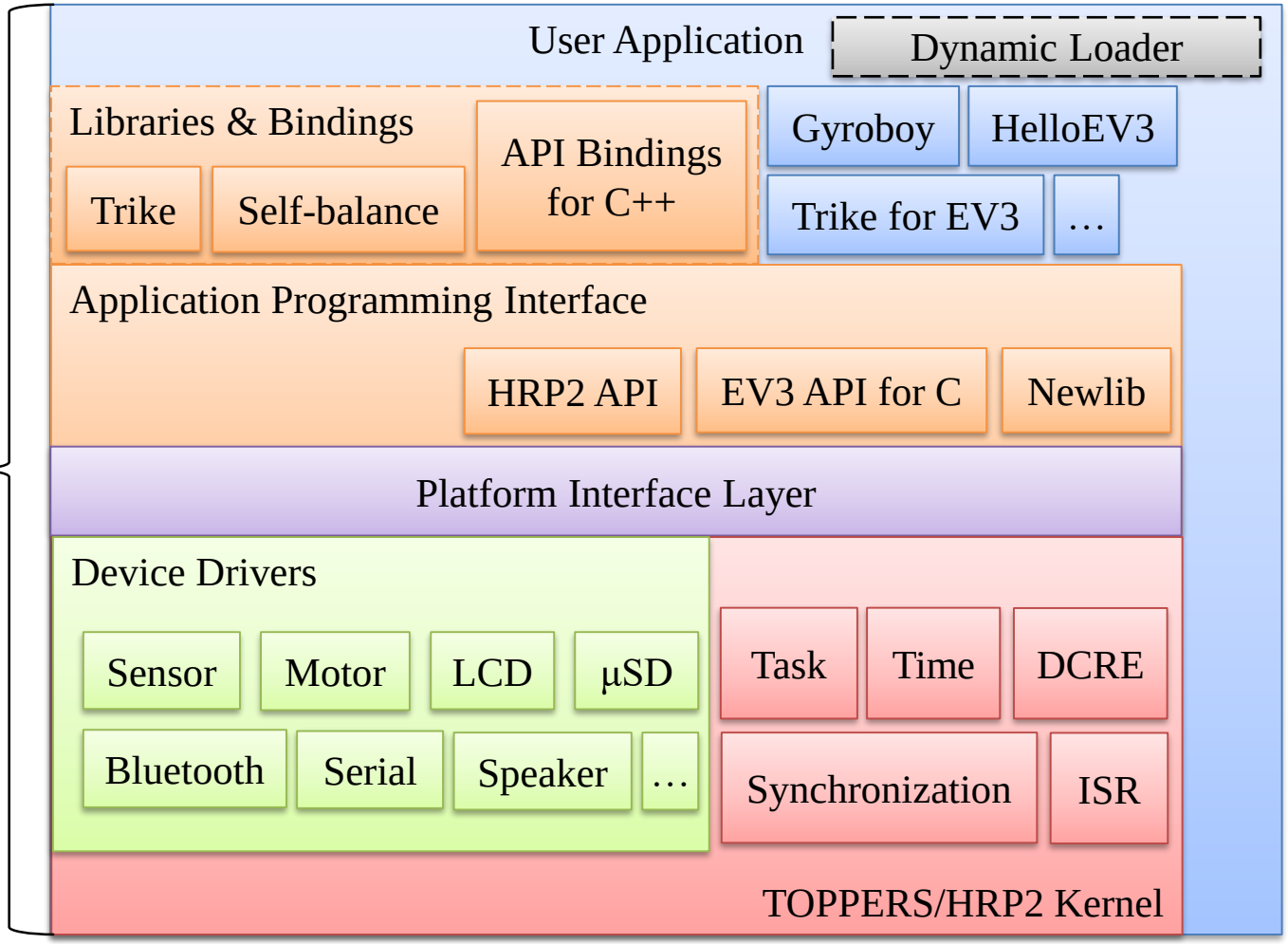
ユーザアプリ

非特権モード
専用ユーザドメイン
（TDOM_APP）
APIで開発

# 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



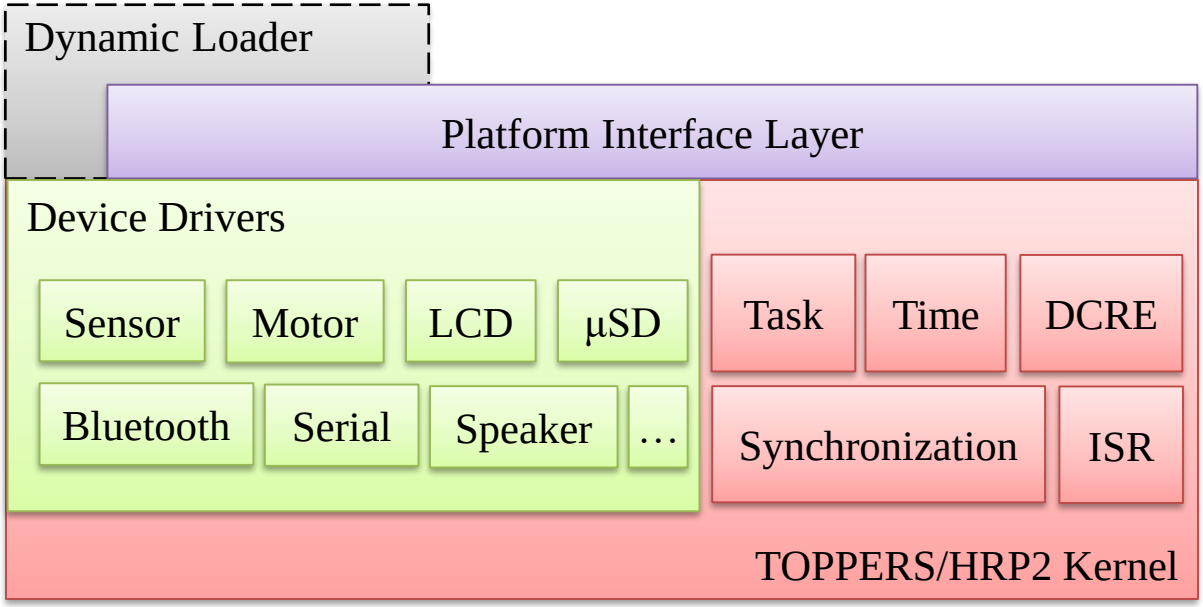

※開発者が自由に改造できるため、動的形式に基づいて解説

# 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
- 動作ローディング機能
  - Dynamic Module Loader
  - アプリケーションモジュール
  - アプリのローディングの流れ
- 進んだ使い方

# 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
- モジュール構成
  - HRP2 Kernel
  - 各種ドライバ
  - Platform I/F
  - ミドルウェア (e.g. Loader)



# EV3RTのベースシステム

- TOPPERS/HRP2カーネル
  - 静的リアルタイムオペレーティングシステム
    - 利用可能なリソースはcfgファイルで静的に定義
  - 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
    - ユーザアプリからドライバをアクセスするために必要
  - 動的生成機能拡張

# EV3RTのベースシステム

- HRP2カーネルの動的生成機能拡張（DCRE）
  - カーネルオブジェクトの動的生成をサポートするパッケージ

```
ER_ID tskid = acre_tsk(pk_ctsk); // C言語でタスクを生成
```

  - Object Poolパターン：動的に生成できる最大数が静的指定

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

  - Dynamic Loaderやドライバ等を開発するために導入
  - ユーザアプリの開発では直接に使えない
    - DCREはミドルウェアの開発向けの機能、細粒度の保護機能が無い
      - ✓生成できるタスクの属性を制限しない
      - ✓例：カーネルドメインのタスクを生成して保護機能をbypass
    - EV3RTではシステムタスク（カーネルドメイン）からしか利用できない
      - ✓スタンドアローン形式はシステムタスクも使えるため利用可能

# EV3RTのベースシステム

- 重要なデバイスドライバ・ミドルウェアの一覧（1/2）
  - カーネル機能とLow Level Driver
    - Linux Kernel Driver Interface互換レイヤ
    - SoC
    - 汎用入出力（GPIO）ポート
    - 高精度アラームハンドラ機能
  - EV3専用のデバイスドライバ
    - スピーカ
    - モータ
    - アナログ-デジタル変換（ADC）
    - Soft UARTポート
    - UARTセンサ
    - SDカードコントローラ
    - LCDスクリーン

# EV3RTのベースシステム

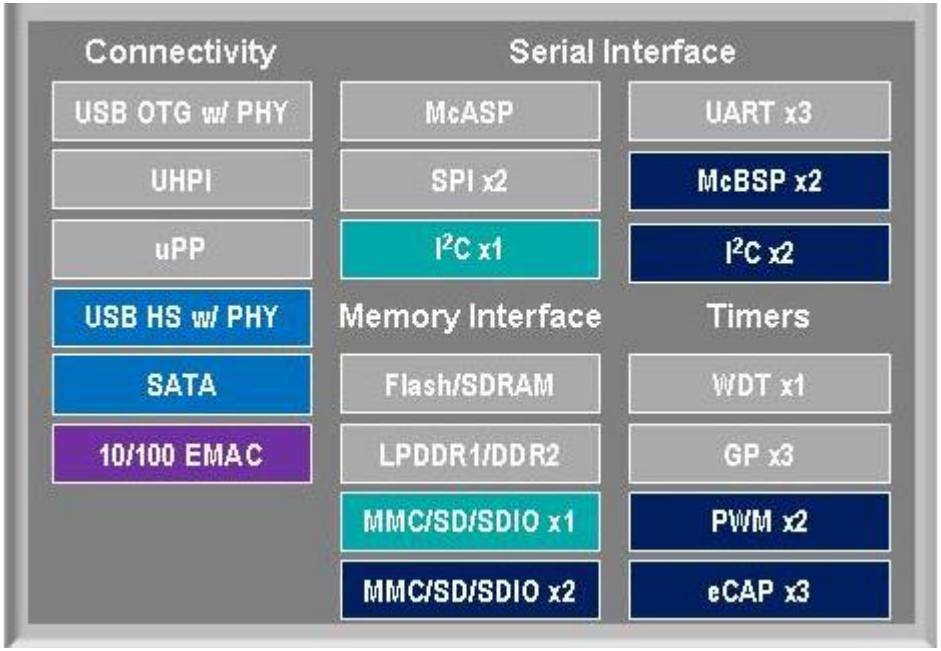
- 重要なデバイスドライバ・ミドルウェアの一覧（2/2）
  - 汎用ドライバ・ライブラリ・拡張機能
    - Stellaris USB Library
    - BTstack
    - FatFS
    - TLSFメモリアロケータ
    - Newlib
    - ユーザドメインのボタンハンドラ機能
    - ユーザドメインの周期ハンドラ機能
    - TCP/IP
  - その他
    - EV3RT Console機能
    - システム設定機能

# EV3RTのベースシステム

- Linux Kernel Driver Interface互換レイヤ
  - 一部のLinuxのドライバを再利用するために、TOPPERSカーネルAPIを使って簡易のLinux Kernel Driver Interface互換レイヤーを実装した
    - メモリ管理・アクセス機能
    - カーネルロギング機能
    - 同期機能（spin lock、mutex、semaphore）
    - 割込みハンドラ管理機能
    - 高精度カーネルタイマ（hrtimer）

# EV3RTのベースシステム

- SoCドライバ
  - EDMA、eCAP、UART等SoCの基本機能をサポート
  - 他のドライバの開発を支援するためのミドルウェア
  - StarterWare for ARM-based Sitara Processorsをベースとする
    - EV3のSoCメーカ（Texas Instruments）が提供したLow Level Driver
  - サブシステムの管理
  - Special Registersの定義



# EV3RTのベースシステム

- 汎用入出力（GPIO）ポートのドライバ
  - GPIOに関する操作をサポート
  - Pin定義とMultiplex機能

```
typedef enum {
  GP0_0,GP0_1,GP0_2,GP0_3,GP0_4,GP0_5, // ...
  GP1_0,GP1_1,GP1_2,GP1_3,GP1_4,GP1_5, // ...
  // ...
  GP8_0,GP8_1,GP8_2,GP8_3,GP8_4,GP8_5, // ...
  NO_OF_GPIOS,
  UART0_TXD,UART0_RXD,UART1_TXD,UART1_RXD,
  SPI0_MOSI,SPI0_MISO,SPI0_SCL,SPI0_CS,
  SPI1_MOSI,SPI1_MISO,SPI1_SCL,SPI1_CS,
  EPWM1A,EPWM1B,APWM0,APWM1,EPWM0B,AXR3,AXR4,
  UART2_TXD, UART2_RXD, UART2_CTS, UART2_RTS,
  ECAP2_APWM2,        // Bluetooth slow clock
  NO_OF_PINS
} PINNO;
extern void setup_pinmux(uint32_t pin);
```

  - Pin単位の割込みハンドラ登録とDispatch機能

```
void request_gpio_irq(int pin, uint32_t trigger, ISR isr, intptr_t exinf);
```

# EV3RTのベースシステム

- 高精度アラームハンドラ機能のドライバ
  - 第二世代のTOPPERSカーネル（HRP2）はタイマ管理機能はミリ秒単位しか対応していない
  - マイクロ秒精度で動作するドライバ（Analogセンサ等）が存在するため、より高精度なアラームハンドラ機能が必要
  - HRP2カーネルはタイムティックの設定をサポートする
    - タイムティック周期＝（TIC_NUME / TIC_DENO）ミリ秒
    - TIC_DENOを増やすことで，より高精度のタイムティックが得られる
  - 標準のアラームハンドラ機能を参考してタイムティック精度で動作するアラームハンドラ機能を実装した

```
acre_hires_alm(...) // 高精度アラームハンドラの生成
sta_hires_alm(...)  // 高精度アラームハンドラの動作開始
stp_hires_alm(...)  // 高精度アラームハンドラの動作停止
```

# EV3RTのベースシステム

- 高精度アラームハンドラ機能のティック周期の課題
  - 引数の単位はマイクロ秒で、実際の動作はティック精度
  - アラームの起動時刻はティック単位で切り捨てる
    - Tick: 100us, Almtim: 125us => 実際のAlmtim: 100us
    - 切り捨てられた場合はWarning Logを出力
  - ティック周期を全ての高精度アラームハンドラの相対起動時刻の最大公約数に設定しないと精確に動作できない

| 機能 | 起動周期 |
|---|---|
| サウンド再生 | 125 μs |
| Analogセンサ情報更新 | 200 μs |

- しかし、ティック周期が早すぎると、オーバヘッドがとても大きくなって、リアルタイム性に悪影響が出る。現時点は、サウンド再生機能の処理精度を犠牲にして100usに設定

# EV3RTのベースシステム

- スピーカのドライバ
  - サウンドの再生をサポート
  - 指定した周波数のノートとWAVファイル（8bit、モノラル）のデータストリームを再生できる
    - リアルタイム性を保証するために、データストリームはインメモリでないといけない
    - アプリはWAVファイルをメモリにロードしてから再生する
  - 音声の再生は排他的に行う
    - 現在再生中のサウンドが停止される
  - Linuxのドライバを再利用した

# EV3RTのベースシステム

- モータのドライバ
  - モータの制御をサポートするドライバ
  - LinuxのPWM制御モジュール（d_pwm）を再利用した
  - 以下の機能を持つ
    - 速度制御
    - ブレーキ制御
    - 二つのモータの同期運転（ステアリング等）
    - 回転制御（指定した角度や時間でモータを回す）
    - ロータリーエンコーダ（モータの回転角を測定）

# EV3RTのベースシステム

- アナログ-デジタル変換（ADC）のドライバ
  - ADコンバータ（SPI経由）からデータを取得する
    - Analogセンサ（タッチセンサ等）のデータ
    - I2Cセンサ（HiTechnic製加速度センサ等）のデータ
    - モータの電流
    - バッテリの電圧
    - センサのタイプ
  - Linuxのドライバを再利用した

# EV3RTのベースシステム

- Soft UARTポートのドライバ
  - EV3には4つ（ポート1～4）のUARTポートが搭載される
  - ポート1と2はNativeのUARTでポート3と4はSoCのPRUSS（Programmable Real-Time Unit Subsystem）という機構でソフトウェアでエミュレートしたもの
  - PRUSSはTI社独自の開発言語（アセンブラ風）を使ってとても理解し難いがLinux用のファームウェアが提供している
  - Linux用Soft UARTのファームウェアのLoaderを移植してSoft UARTポートのドライバを実装した

# EV3RTのベースシステム

- UARTセンサのドライバ
  - UARTで通信するセンサをサポート
  - EV3のUARTセンサは具体的なタイプ（超音波かカラー等）にかかわらず、LEGO社が定義した共通のプロトコルで通信
  - UARTセンサ通信プロトコルの機能
    - センサとポートのクロック・Baudrate同期
    - センサの情報の取得（モードの数、データの範囲等）
    - センサモードの切り替え（カラー＝＞カラーモード、反射光モード等）
    - センサデータの更新
  - Linuxのドライバを再利用した

# EV3RTのベースシステム

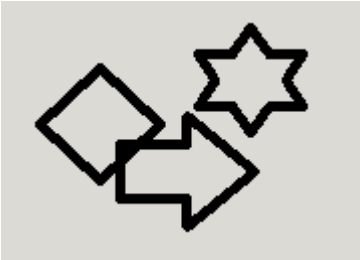
- SDカードコントローラのドライバ
  - EV3に搭載されたmicroSDカードスロットをサポート
  - EV3のSoCは二種類のSDカードを対応；SDXCは非対応

| 規格 | サイズ | Addressing Mode |
|---|---|---|
| SD | ～2GB | Byte単位 |
| SDHC（SD High Capacity） | 4GB～32GB | Block（512B）単位 |

  - Block単位のRead/Write操作を実装した
    - 現時点はSDHCだけ対応（SoCのマニュアルはSDHCベースのため）
    - SDカード（<=2GB）は将来サポートする予定
    - DMA方式で実装したので性能が良くてリアルタイム性に影響小さい

# EV3RTのベースシステム

- LCDスクリーンのドライバ
  - EV3のLCDコントローラ（ST7586）で画面表示
  - 複数のFramebufferを提供、互い干渉しないので開発は簡単
    - アプリ用Framebuffer：ユーザアプリが直接にアクセスできるVRAM
    - システム用Framebuffer：システムメニュー・Console等用のVRAM



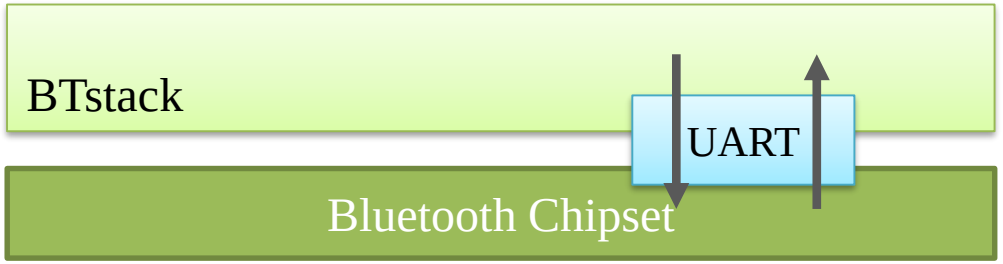
  - 指定した頻度で画面をリフレッシュする
    - デフォルトは25fps、DMAで処理するのでオーバヘッドは小さい
    - VRAMへの書き込みはLinuxのST7586ドライバを参考して実装した

# EV3RTのベースシステム

- Stellaris USB Library
  - TI社により提供した簡単なUSBスタック
    - TI社のSoCなら無料に使える
    - Mass Storage Class（MSC）のサンプルも入っている
  - MSCのサンプルをEV3RT用に改造した
    - 元々はRAMdisk → MMC/SDカードドライバと連携
    - Mac OS XのMSCドライバに対応
    - Buffer機能を実装することで速度を向上（e.g.書込 300KB/s→6MB/s）
    - EV3RT用fork：https://github.com/ev3rt-git/usblib
  - EV3のミニUSBポートに接続してSDカードリーダとして使える
    - ユーザアプリと同時に動作できない（排他制御）
    - ejectせずに取り外すと再起動しないと接続できないことがある

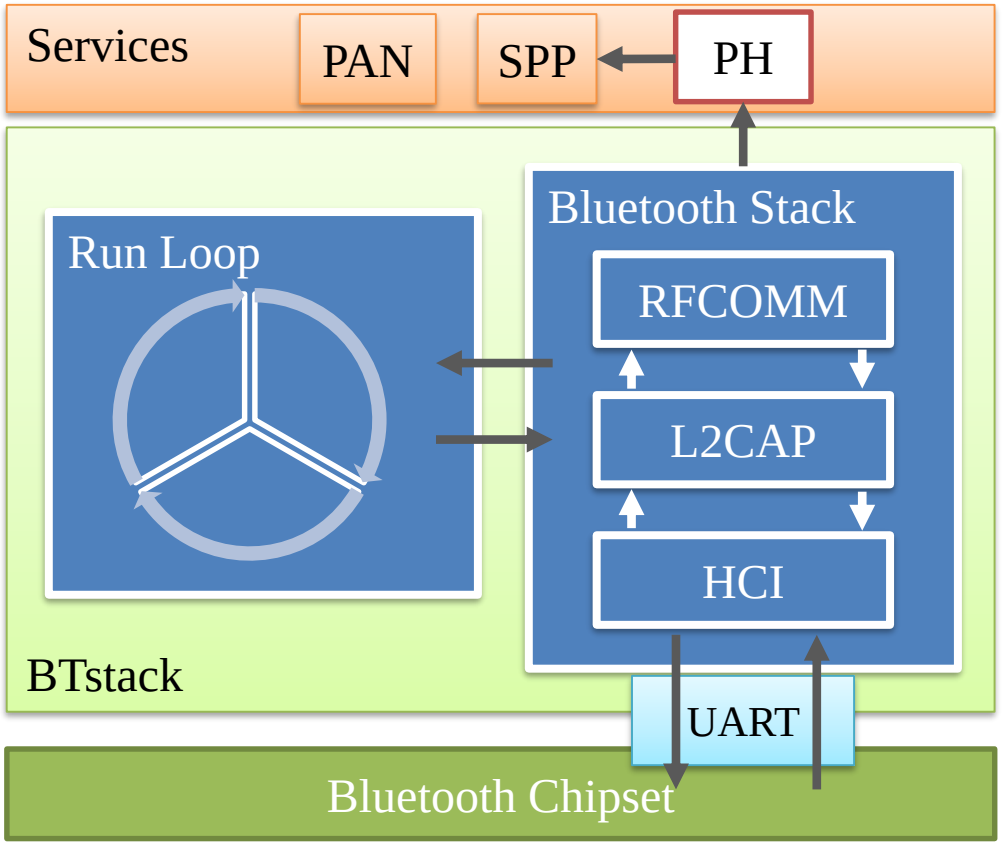
# EV3RTのベースシステム

- Bluetoothのドライバ
  - EV3に標準のBluetoothチップ（CC256x, v2.1 EDR）が搭載されているため、標準のBluetoothスタックドライバが必要
  - BTstackというオープンソースのスタックドライバを移植
    - 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もサポートする
    - EV3RT用fork：https://github.com/ev3rt-git/btstack
  - CC256xとSoCはUARTで通信してい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



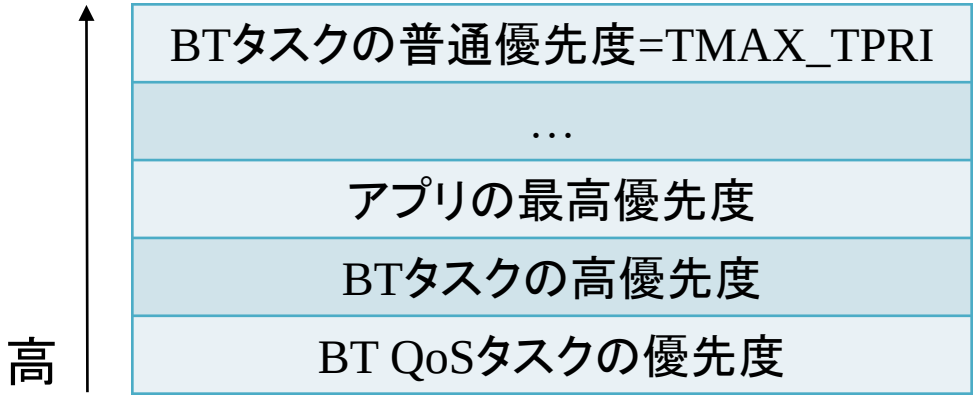
# EV3RTのベースシステム

- Bluetoothのドライバ
  - BTstackのAPIを使ってSPPとPANのサービスを実装した
    - Serial Port Profile
    - Personal Area Network
  - 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
  - 受送信データを処理するためにRun Loopが必要。Run Loop用のタスクを作成した



# EV3RTのベースシステム

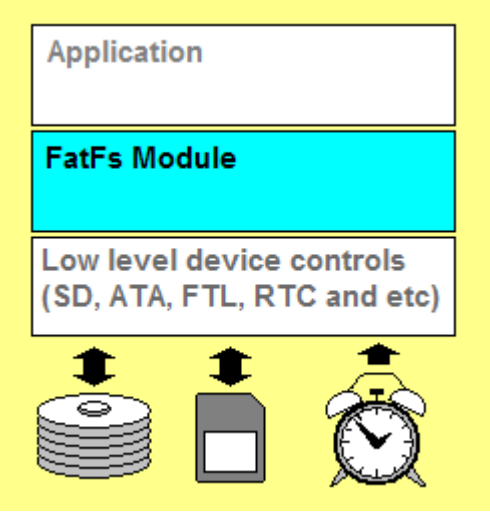
- Bluetooth通信のQoS確保
  - アプリより高い優先度で動作すると
    - 高速で通信する場合、アプリのリアルタイム性に悪影響
  - アプリより低い優先度で動作すると
    - スタベーションで接続が切れてしまう可能性がある
  - 接続を保護するために、QoSタスクで動的にBTタスクの優先度を上げる
    - 現時点のポリシー　20ミリ秒ごとに1ミリ、BTタスクをアプリより高い優先度に
      - ✓参考：倒立制御処理　5ミリ秒周期で動作、実行時間は1ミリ秒以内
    - 課題：より精確なKeep Aliveポリシーを使う



# EV3RTのベースシステム

- 汎用FATファイルシステムモジュール FatFS
  - 特徴
    - Windows互換FATファイル システムをサポート
    - ANSI C(C89)準拠で移植性が優れている
  - 下位レイヤインターフェース

disk_status - デバイスの状態取得
disk_initialize - デバイスの初期化
disk_read - データの読み出し
disk_write - データの書き込み
disk_ioctl - その他のデバイス制御
get_fattime - 日付・時刻の取得




  - disk系関数はSDカードのドライバを使って対応できる
  - get_fattime()は現時点で未対応

# EV3RTのベースシステム

- Two-Level Segregate Fit（TLSF）メモリアロケータ
  - リアルタイムシステム向けの動的メモリアロケータ
    - 動的生成機能拡張と標準Cライブラリのために導入
  - O(1)の計算量でmalloc/free、リアルタイム性高い
  - 複数のメモリプールが対応
    - ベースシステム（カーネル）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
    - Newlibのデフォルトのアロケータは一つのheapしか使えない
  - Fragmentationは少ない（平均的に<15%）
  - 排他制御はサポート

# EV3RTのベースシステム

- Newlib Support Package
  - BTstack、FatFS、TLSF等のドライバはそれぞれのAPIを持っている。標準Cライブラリ（Newlib）でこれらのドライバの機能を吸収して開発者の学習コストを削減できる
  - TLSFアロケータで動的メモリ管理のシステムコールを実現
    - _malloc_r/_free_r/_calloc_r/_realloc_rを実装してNewlib本来のアロケータをoverrideした
  - FatFSでファイル操作機能をサポート
    - FatFSはFile Descriptorを使ってないため、NewlibのFile DescriptorでFatFSのデータ構造（FIL）を管理する機能を実装した
    - fopen/read/write等の関数でSDカードを操作できるようになった
  - Bluetooth通信のためのFD（SIO_BT_FILENO）を作成
    - 通信用の特殊なFD、fprintf/fscanf等で簡単にBluetooth機能を使える

# EV3RTのベースシステム

- ユーザドメインのボタンハンドラ機能
  - ボタンハンドラはGPIOの割込みで起動するが、割込み処理の中から直接に呼び出されると保護機能が適用できない
  - この課題ためにボタンハンドラ用の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
      - ✓iset_flg(current_button_flag, 1 << BRICK_BUTTON_BACK)
    - ユーザドメインのタスクはフラグをチェックしてハンドラを呼び出す
      - ✓wai_flg(BTN_CLICK_FLG, …) => handlers[i](exinfs[i])
    - ボタンハンドラのタスク優先度（TMIN_APP_TPRI-1）はアプリの全てのタスクよりも高いため、長い処理を行う時は要注意

# EV3RTのベースシステム

- ユーザドメインの周期ハンドラ機能
  - TOPPERSの標準の周期ハンドラはボタンハンドラと同様に割込み処理から呼び出されるので保護機能が使えない
  - 周期ハンドラの定義や数はボタンハンドラとは違って固定されていないので、同じ手法で対応できない
  - アプリの周期ハンドラ毎にTOPPERSの周期ハンドラと実行用タスクを生成して、TOPPERSの周期ハンドラでタスクを起動
    - 優先度はボタンハンドラと同じTMIN_APP_TPRI-1、長い処理を行う時、ハンドラからact_tskでアプリの普通のタスクを起動したほうが良い
  - 提供する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
```

# EV3RTのベースシステム

- ユーザドメインの周期ハンドラ機能
  - 静的APIの定義からユーザドメインの周期ハンドラを動的に生成する関数が生成される

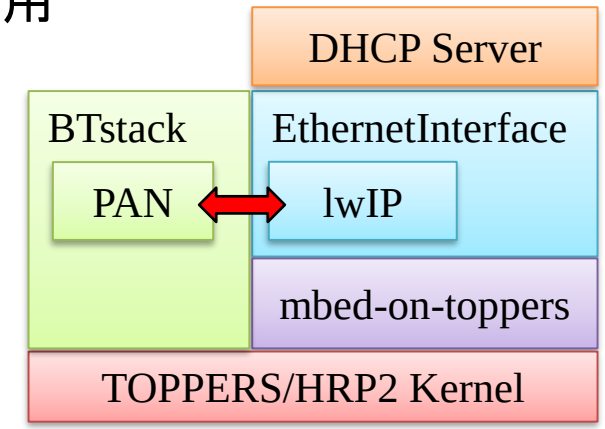
```
EV3_CRE_CYC(EV3_CYC1,{TA_STA,0,test_ev3_cychdr,500,0});
```

```
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);
    _ev3api_id_EV3_CYC1 = ercd; }
```

  - アプリの初期化タスクで呼び出される ⇑

# EV3RTのベースシステム

- TCP/IP対応
  - HTTP（Bluetooth PAN）でのアプリ更新のために必要
    - Beta 7以前はBluetooth 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では「make upload」コマンド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
  - ソースコードはパッケージにない
    - サイズが大きいので…
    - https://github.com/ev3rt-git



# EV3RTのベースシステム

- EV3RT Console機能
  - EV3本体で直接にEV3を制御できる
  - Backボタンの長押しで呼び出される
  - タスクログはここで見られる
    - syslog,printfの出力はパソコンがなくても確認できる
  - Loader機能とPower offが使える

# EV3RTのベースシステム

- システム設定機能
  - INIファイル（/ev3rt/etc/rc.conf.ini）でシステムを設定できる
  - INIファイルを解析するためにMinIniモジュールを導入
  - 現時点で設定可能な項目
    - [Debug] DefaultPort
      - ✓デフォルトのログ出力ポート：LCD、UART（Port1）、BT
    - [Sensors] DisablePort1
      - ✓センサポート1を無効化するとUART（シリアル）ポートとして使える
    - [Bluetooth] LocalName / PinCode
      - ✓Bluetoothのデバイス名とPINコード
    - [Bluetooth] DisablePAN / IPAddress
      - ✓Bluetooth PAN機能の無効化、EV3本体のIPアドレス
    - [USB] AutoTerminateApp
      - ✓USBケーブルを接続する時、自動的にアプリを終了するか

# EV3RTのベースシステム

- Platform Interface Layer
  - 保守性を向上するために、ユーザアプリからアクセスすることがあるドライバに対してある程度安定したI/Fが必要
  - PILはユーザアプリとベースシステム間のI/Fを定義する
    - 共有メモリ（brickinfo_t）
    - 拡張サービスコール
  - PILにバージョン番号がある
    - ベースシステムは同じPILバージョンのアプリしかロードできない
      - ✓Beta 6-2（PIL Ver.5）とBeta 7（PIL Ver. 6）
    - 同じPIL番号のベースシステム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

# EV3RTのベースシステム

- 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のポインタの集合
    - ベースシステムはドライバの初期化で該当のポインタをセット
    - SVC _fetch_brick_info()でベースシステムのbrickinfo_tを取得できる
    - 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;
```

# EV3RTのベースシステム

- Platform Interface Layer
  - 拡張サービスコールのインタフェース
    - ベースシステムにドライバStubのプロトタイプを定義

```
extern ER _fetch_brick_info(brickinfo_t *p_brickinfo, ID cdmid);
```

    - ベースシステムに拡張サービスコールを生成

```
#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 });
```

    - ユーザアプリに拡張サービスコールを呼び出す関数を提供

```
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;
}
```

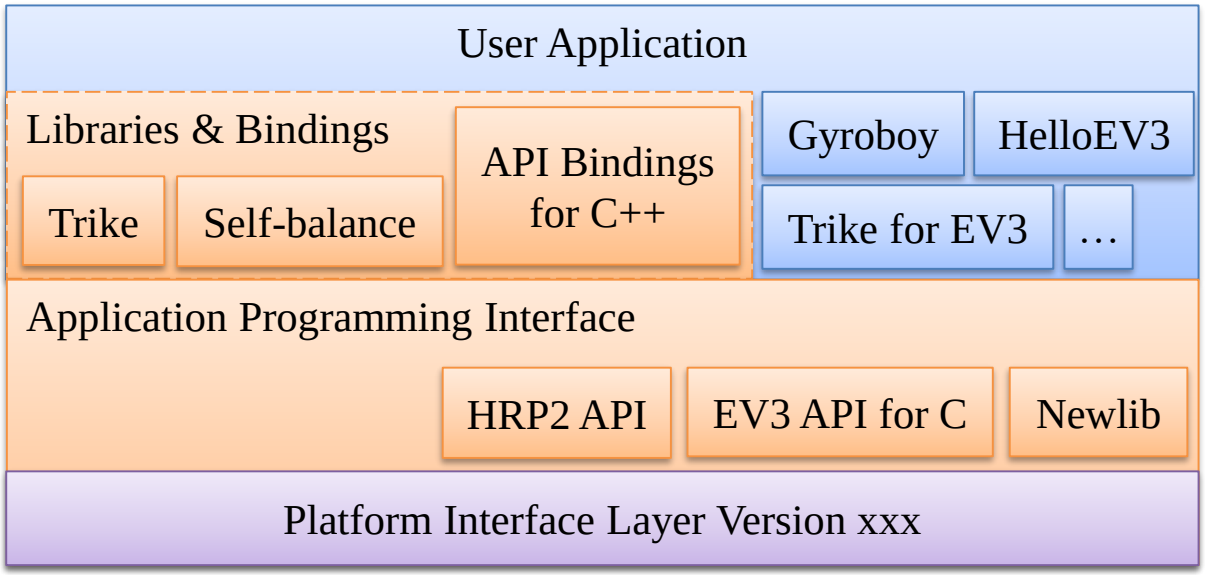
    - ユーザアプリのコンパイルはベースシステムのソースに依存しない

# 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**
- 動作ローディング機能
  - Dynamic Module Loader
  - アプリケーションモジュール
  - アプリのローディングの流れ
- 進んだ使い方

# EV3RTの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で作成できる
  - 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

# EV3RTのユーザアプリケーションとSDK

- Application Programming Interface
  - TOPPERS/HRP2カーネル静的API
    - cfgファイルに記述して、カーネルオブジェクトを静的に定義・生成
    - 保護ドメインはTDOM_APPか無所属でないといけない [Dynamic]
  - TOPPERS/HRP2カーネルAPI（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（Bluetooth通信含む）、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

# EV3RTのユーザアプリケーションとSDK

- 静的ライブラリ
  - アプリの開発ではAPIの他にオプションのライブラリも使える
  - 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
...
```

# EV3RTのユーザアプリケーションとSDK

- 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

# EV3RTのユーザアプリケーションとSDK

- 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 |
|---|---|---|---|
| 1 | システム初期化 | once | / |
| 2 | USB | / | / |
| 3 | Bluetooth（高） | 20ms | <1ms |
| 4 | LCDリフレッシュ | 40ms | <1ms |
| 6 | BUSY | / | / |
| 7 | アプリ初期化 | once | / |
| >= TMIN_APP_TPRI | ユーザ アプリケーション | | |
| TMAX_TPRI | Bluetooth（低） | | |

–BUSYタスクは単純な無限ループ
  - システム初期化の途中でアプリのタスクの実行を一時停止できる

# EV3RTのユーザアプリケーションとSDK

- ユーザアプリケーションの初期化タスク（一部省略）

```
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();
}
```

  - C++ Global Constructorには待ち状態になるコードを使っては行けない

# 発表内容

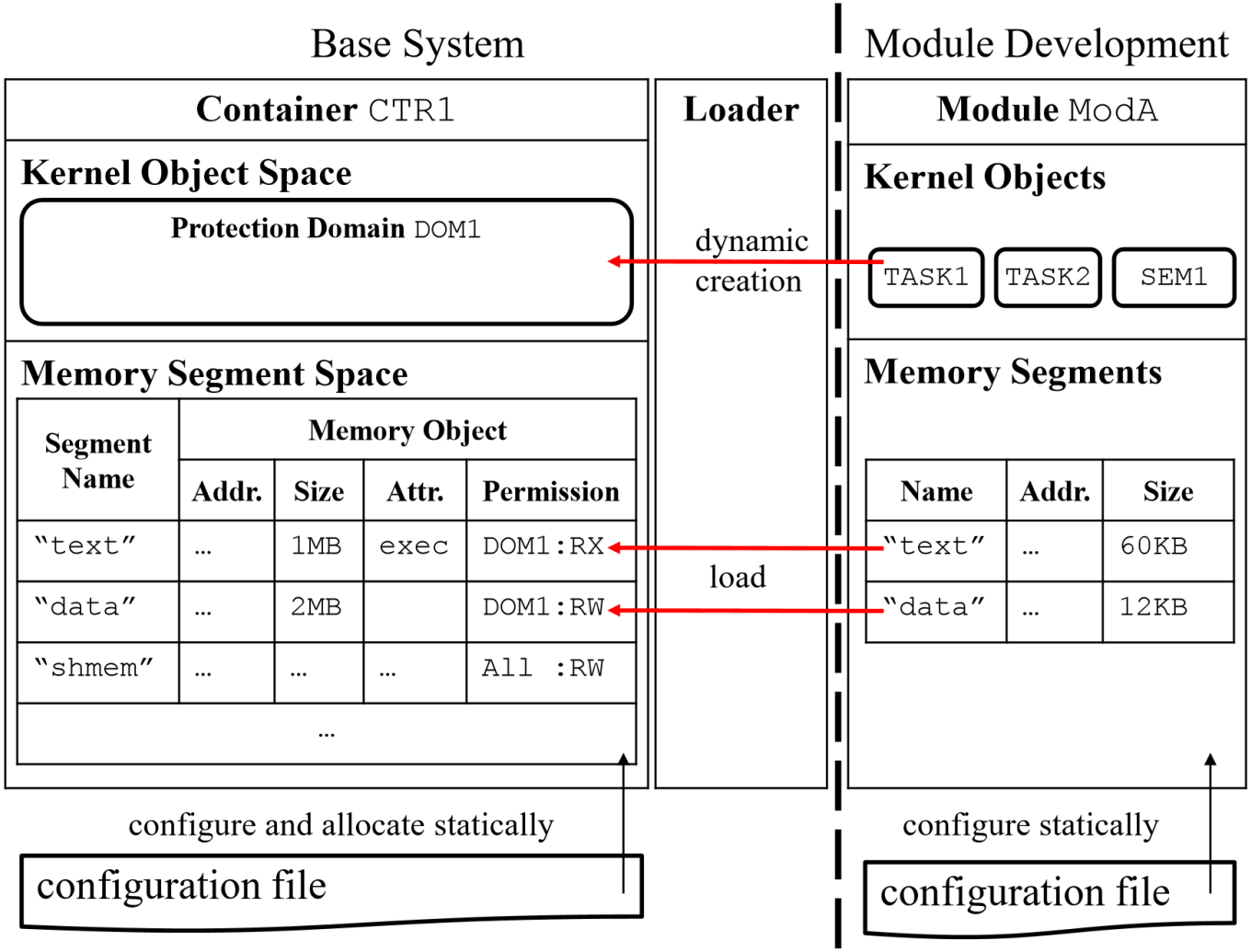
- EV3RTの内部構造の解説
  - アーキテクチャの概要
  - ベースシステムの解説
  - ユーザアプリケーションとSDK
- **動作ローディング機能**
  - Dynamic Module Loader
  - アプリケーションモジュール
  - アプリのローディングの流れ
- 進んだ使い方

# 動的ローディング機能

- Dynamic Module Loader
  - 静的OS（HRP2）のために開発されたモジュールを（EV3RTのベースシステムに）動的に追加・削除するための機能
  - モジュールの保護設定とリソースはコンパイル時に決まる
    - cfgファイルに記入した静的APIから生成
    - HRP2 DCRE（動的生成機能拡張）のAPIは使用できない
    - モジュール毎には一つの専用のユーザドメインしか使えない
      - ✓汎用OSのプロセスに似た概念
  - ベースシステムはモジュール実行用のコンテナを用意
    - 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
  - ローディング：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、動作を開始すること

# 動的ローディング機能

- Dynamic Module Loaderの概略図



# 動的ローディング機能

- 動的ローディング用アプリケーションモジュール
  - 独立のELFファイルとしてビルド
    - 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
  - cfgファイルの内容をELFファイルに格納する
    - 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
    - Module Configuration Tableのコードをコンパイル時に生成する
  - ローダからオブジェクトのIDを取得する方法が必要
    - コンパイル時はunknown
  - ローディング用メモリ領域のアドレスは固定ではない
    - ベースシステムの設定等により変化
    - 完全位置独立コード（-mno-pic-data-is-text-relative）で対応

# 動的ローディング機能

- cfgファイルを格納するModule Configuration Table
  - Module Configuration Entryの配列

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

    - sfncd：静的機能コード　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 },
};
```

# 動的ローディング機能

- オブジェクトID生成の流れ
  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. オブジェクトIDの変数を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_yyy()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. これで、オブジェクトIDのマクロは実際の値になる

# 動的ローディング機能

- ローダのロード操作の流れ
  0. 動的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
- 動作ローディング機能
  - Dynamic Module Loader
  - アプリケーションモジュール
  - アプリのローディングの流れ
- **進んだ使い方**

# EV3RTの進んだ使い方

- スタンドアローン形式で、より高度な開発ができる
  - 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
  - TOPPERS/HRP2カーネルの全機能が自由に利用できる
  - デバイスドライバ・ミドルウェアの直接アクセス

# EV3RTの進んだ使い方

- 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
    - 静的に生成されたページテーブルの情報が変わった
    - 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.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

# EV3RTの進んだ使い方

- HRP2カーネルの全機能が自由に利用できる
  - 動的ローディング形式では、アプリの障害からLoaderとベースシステムを守るために、アプリには幾つかの制限が存在
    - 全てのオブジェクトは単一のユーザドメイン（TDOM_APP）に属する
    - 使用可能な最高優先度（TMIN_APP_PRI）が存在する
  - スタンドアローン形式では、上記の制限がなくて、cfgファイルでアプリケーションを自由に設計できる
    - メリット
      - ✓HPR2の全ての機能（動的生成機能拡張を含む）が使える
    - デメリット
      - ✓Loaderが使えなくなって、アプリの更新に手間がかかる
      - ✓カーネルドメインを利用する場合、デバッグは難しくなる

# EV3RTの進んだ使い方

- デバイスドライバ・ミドルウェアの直接アクセス
  - アプリ開発の利便性を向上するために、BluetoothやFatFSのドライバ機能が標準Cライブラリ経由で使う
  - 学習コストがなくなった反面、フル機能も使えなくなった
  - 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またはmbedのEthernetInterfaceでTCP/IP通信
    - アプリのアップロードでしか使用していないが、lwIPのAPI等を使えばPCとのsocket通信ができ、PC経由でインターネット利用も実現可能

# EV3RTの進んだ使い方

- プラットフォーム拡張の例：スクリーンショット機能追加
  - EV3RTのチュートリアルを作る時、LCDの画面を撮りたい
    - ユーザアプリは自分の画面しかアクセスできない
      - ✓Consoleの画面を撮ることができない
      - ✓複数のアプリを撮りたい場合、アプリ毎に修正が必要
  - スタンドアロン形式でシステムタスクを追加
    - 特権モードで動作、LCDのドライバに直接にアクセスできる
    - lcd_dri.h→on_display_fbの内容を周期的にSDカードに保存
      - ✓ボタンハンドラ方式だと既存の機能と干渉しやすい
      - ✓Newlibのファイル操作関数（fopen等）は非特権用のため、fatfs_dri.hをincludeして直接にFatFSの関数（f_open等）を使用

# EV3RTの進んだ使い方

- EV3間通信機能はBeta 7で実験的にサポートした
  - アプリ「test-spp-master」と「test-spp-slave」を提供
    - masterにslaveのEV3のBT Addr.とPin Codeを指定して接続を行う
      - ✓「 spp_master_test_connect_ev3(remote_addr, remote_pincode); 」
      - ✓BT Addr.はEV3RTの起動時のバナーで確認できる
    - SPPベースで、PC間との通信と同様にファイル操作で行う
      - ✓Master
        - »「 FILE *bt_slave = spp_master_test_open_file(); 」
        - »「 fprintf(bt_slave, "MOTOR %d\n", power10 / 10); 」
      - ✓Slave
        - »「 FILE *bt = ev3_serial_open_file(EV3_SERIAL_BT); 」
        - »「 sscanf(buf, "MOTOR %d", &power); 」

ご清聴ありがとうございました