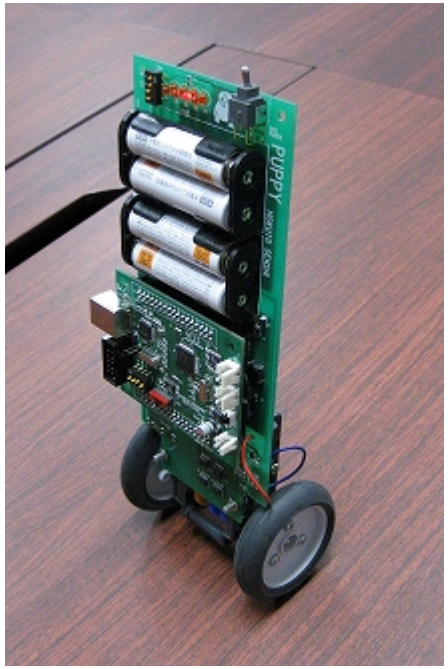
TECSの倒立制御実験機 PUPPY/PUPPY2 への適用

株式会社ヴィッツ
鵜飼 敬幸

TOPPERS

北斗電子 PUPPY / PUPPY2を用いた事例

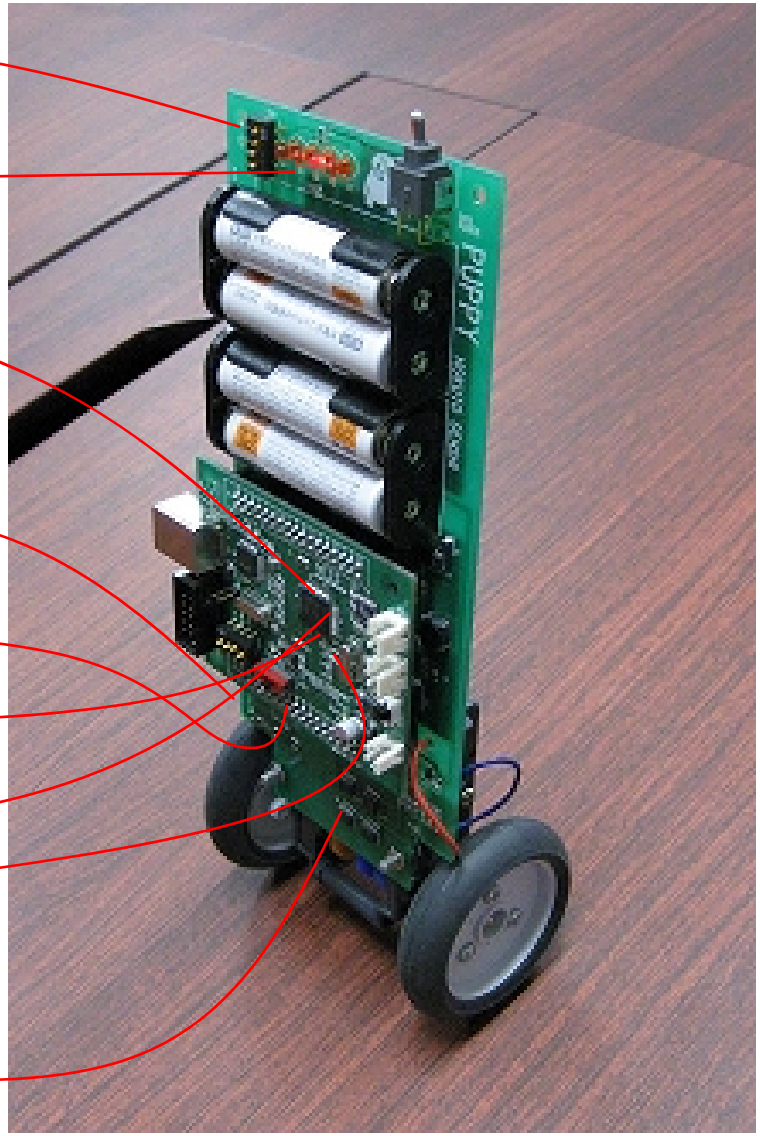
名古屋大学組込みシステム人材育成プログラム
NEP で利用するJSP対応PUPPYを対象



TOPPERSブースにてデモ展示中

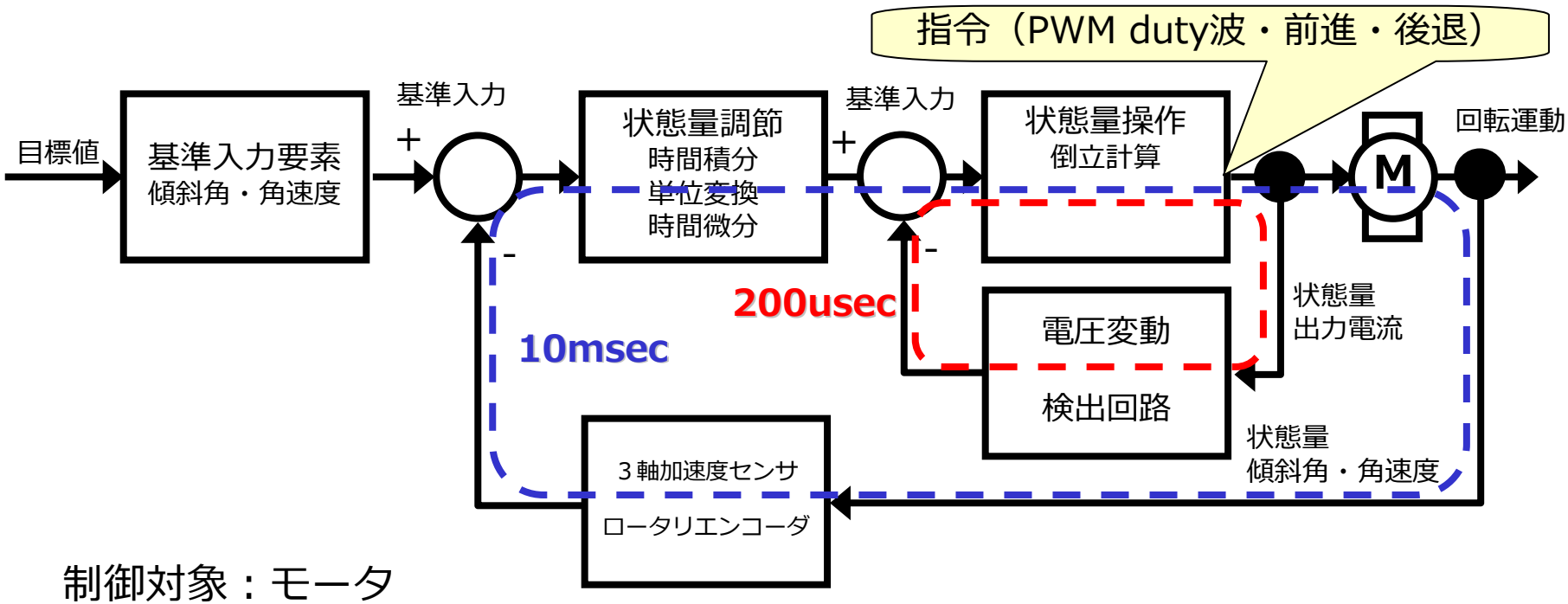
PUPPY / PUPPY2 の構成機能

- スイッチ入力
  - 制御方式の切り替えに使用します
- LED
  - 状態の目視確認に使用します
- ADコンバータ
  - アナロググランド計測
  - ジャイロセンサ値読み取り
  - モータ電流量計測センサ値読み取り
- ロータリーエンコーダ
  - モータの回転数を読み取ります
- ジャイロセンサ
  - 角速度の取得に用います
- PWM
  - モータの回転数制御に使用します
- 倒立計算
  - 制御機能の本体です
- タイマ
  - タスクの周期制御とPWM機能に用います
- モータ電流量計測センサ
  - 実際の出力値を読み取ります



TOPPERS

倒立計算の流れ・ブロック線図（概略）



制御対象：モータ
制御モデル：ラグランジュの運動方程式（ラグランジュ関数）
検出器：
　　３軸加速度センサ（傾斜角・角速度）
　　電圧変動検出回路（電流量を電圧変動量で計算）
　　ロータリエンコーダ（フォトインタラプタによるモータ回転数取得）

※倒立計算式など、詳しくはPUPPYの付属資料を参照

TOPPERS

コンポーネント開発の環境

- TECS対応ASPカーネル使用
  - 先に紹介したコンポーネント化ASPによる組上げ
  - タスク、周期ハンドラなど、カーネルオブジェクト全てをコンポーネントとして利用している
- TECS仕様で利用している機能
  - 定数
  - シグニチャ
  - セルタイプ
  - 複合セル
  - 組上げ記述
  - Importによる複数のCDLファイル指定

TECSの動作環境

- ルネサス 統合開発環境 TM Ver 3.20
  - M16Cマイコン対応ツールチェインを使用
- 実行形式に対応したTECSジェネレータ使用
  - ExerbによりWindowsの実行ファイル化したもの

    参考:http://exerb.sourceforge.jp/
- CPPにM16C対応ツールチェインを使用
  - HEW対応ツールチェイン cpp30
    - ネイティブな環境（例：Cygwinなど）で動作するプログラムではTECSプリプロセッサも使用できる
    - 今回はマイコン専用のものを使用

ＰＵＰＰＹ・ＰＵＰＰＹ２
コンポーネント化

作業の遍歴

- それぞれ別のアプローチで並行作業した
  - PUPPY
    - TECS対応ASPを用いて全てコンポーネント化
  - PUPPY2
    - ASPをそのまま利用（非コンポーネント化）
    - 倒立制御のアプリ部分をコンポーネント化

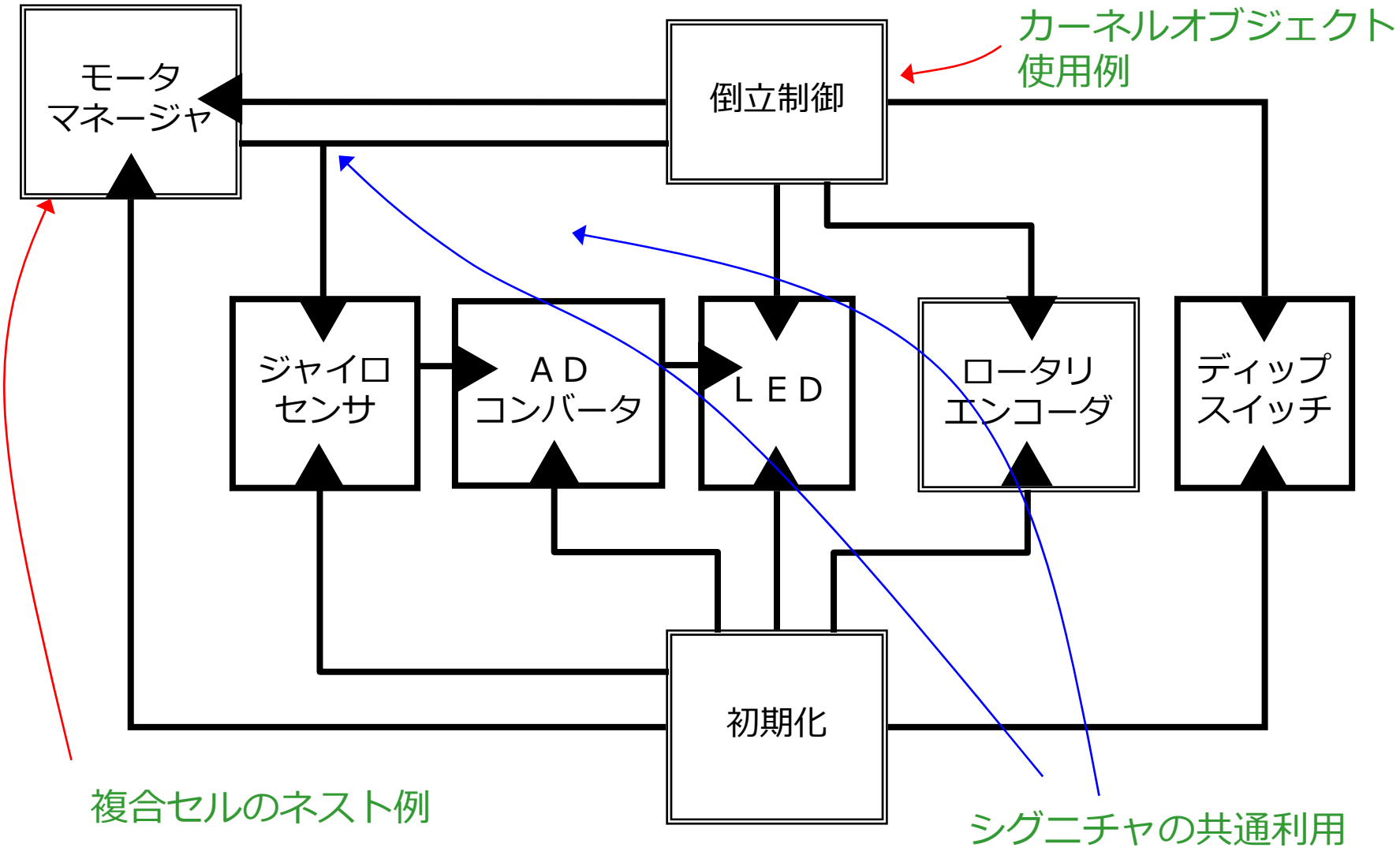
- 共通コンポーネント化を検討
  - 基本の制御フローは変わらない
  - プロダクトライン的な組み換えが検討できそう
  - モータやセンサの複製に機能を活かせる

PUPPY のフォルダ構成

- puppy
  - target ⟵ ターゲット階層
    - hsb16c29_tm ⟵ M16C TM環境
      - puppy1_project ⟵ ASP+TECS対応版
      - sample_project
  - tecs_kernel ⟵ TECS対応カーネルオブジェクト
  - その他 (arch, cfg… etc)
    - aspのフォルダ構成と同様

※tecsgen をTMのプロジェクト中では実行しない
→予め出力しておいたファイルを利用した
※TMのプロジェクトファイルによりビルドを設定した
→tecsgen が自動生成するMakefile は使わない

PUPPY コンポーネント全体図（CDL表記）



TOPPERS

PUPPY コンポーネント説明（概略）

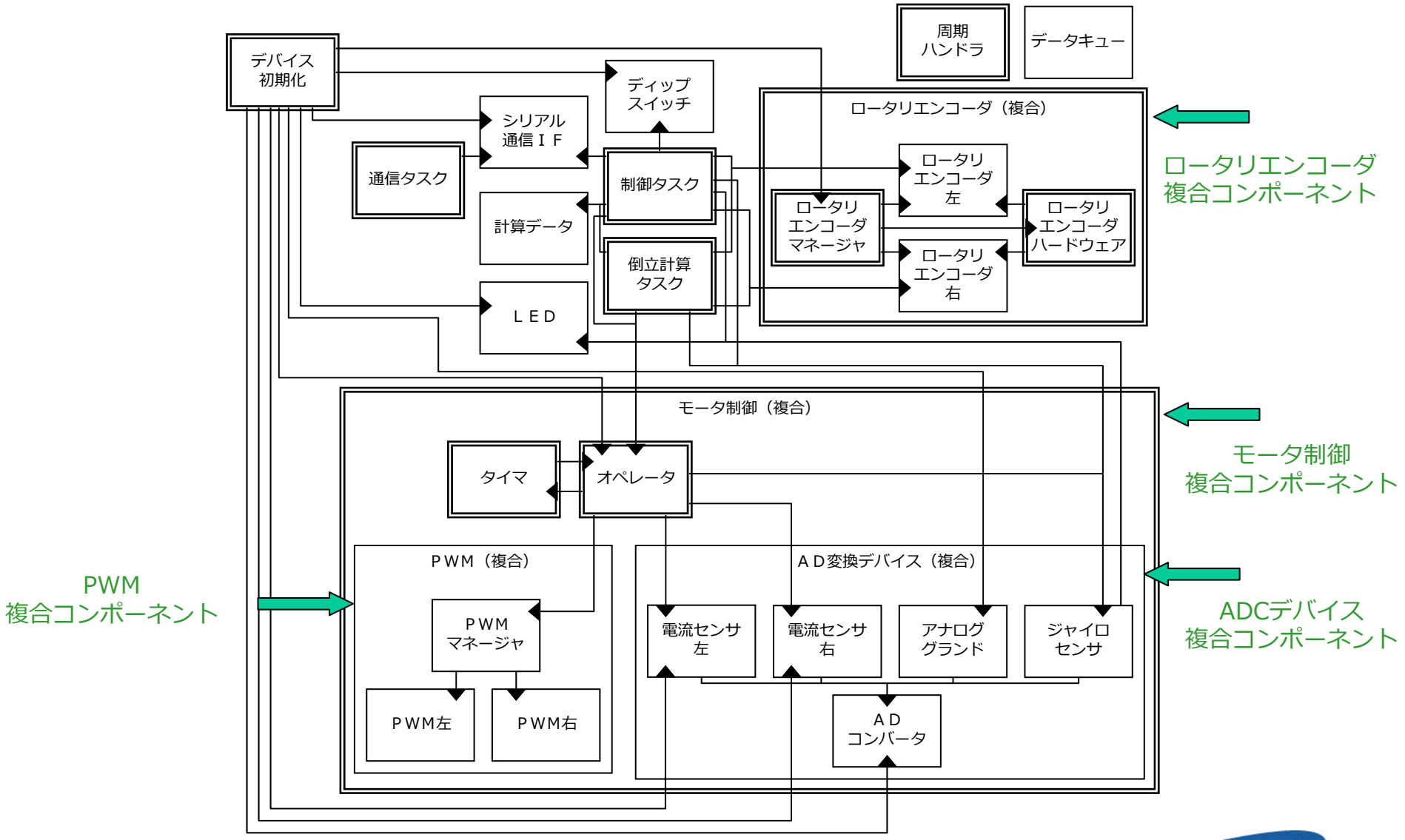
- 倒立制御
  - 倒立計算を行い、タスクと周期ハンドラを内包
- モータマネージャ
  - PWMとモータ制御用数値の定期更新
  - タイマコンポーネントによる周期ハンドラとＰＷＭを実装
- 初期化
  - 各種コンポーネント初期化
  - 各タスク調停のためのイベントと自身用のタスクを内包
- ＡＤコンバータ
  - AD変換デバイス機能
  - PUPPY起動直後のキャリブレーション期間確認のためにLEDを呼び出す
- イベントフラグ
  - 初期化終了タイミングと各タスク連携のために用意
- ロータリエンコーダ
  - フォトカプラによる回転数計測
  - カーネルオブジェクトの割込みを内包
- ディップスイッチ
  - 制御計算の方式切替と動作変更

PUPPY2 のフォルダ構成

- puppy2
  - asp+tecs
    - target ← ターゲット階層
      - hsb16c29_tm ← M16C TM環境
        - » puppy2 ← ASP対応版
        - » puppy2+tecs ← ASP+TECS対応版
        - » sample_project ← ASPサンプル
    - tecsgen
      - tecsgen.exe ← Exerb対応版 tecsgen
    - その他 (arch, cfg… etc)
      - aspのフォルダ構成と同様

※TMのプロジェクトファイルにて構成
→tecsgen が自動生成するMakefile は使わない

PUPPY2 のコンポーネント図全体

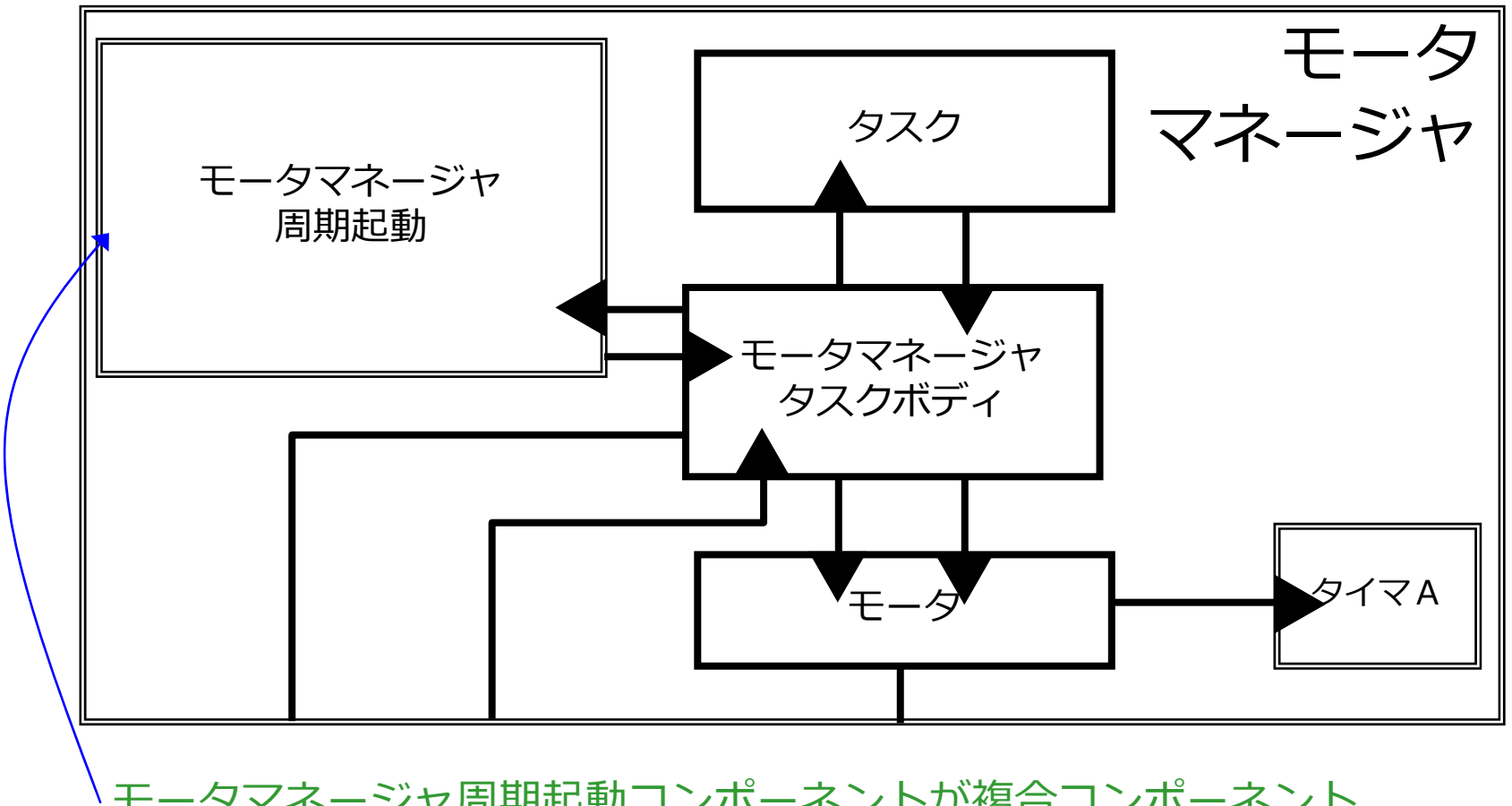


TOPPERS

PUPPY２ コンポーネント説明（概略）

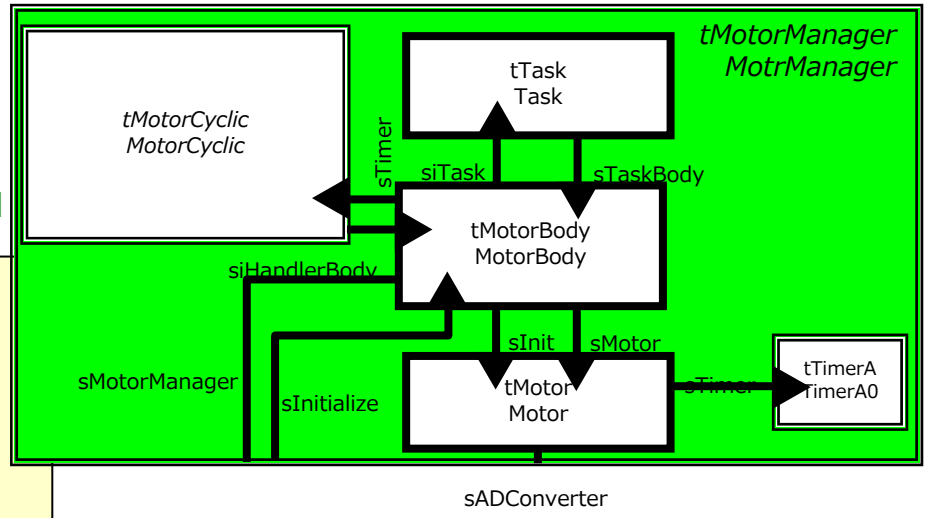
- ロータリエンコーダ複合コンポーネント
  - 左右２つのロータリエンコーダによる回転数を計測
  - 割込みを内包
- モータ制御複合コンポーネント
  - 左右２つのモータに対するＰＷＭ制御とモータ周辺のセンサ値更新
  - ＡＤ変換とＰＷＭ、および周期起動タイマを内包
- デバイス初期化コンポーネント
  - 起動直後のデバイス初期化
- 制御タスクコンポーネント
  - 各タスクとの連携と起動直後の動作決定
  - タスクを内包
- 通信タスクコンポーネント
  - シリアル通信からの指示を制御タスクと通信する
  - タスクを内包
- 倒立計算タスクコンポーネント
  - 各センサからの値を変換し制御入力値に変える
  - タスクを内包
- データキューコンポーネント
  - 制御タスクと通信タスクとの通信用
- 周期ハンドラコンポーネント
  - １０msecごとの倒立計算タスク起動
- etc（シリアルコントローラ、ＬＥＤ，ディップスイッチなどの非アクティブセル）

コンポーネント図　モータマネージャ(PUPPY)



モータマネージャ周期起動コンポーネントが複合コンポーネント
タイマＡと同様の構造を持つタイマＢを持つ

シグニチャ定義（抜粋）　モータマネージャ



target¥hsb16c29_tm¥puppy1_project¥tMotorManager.cdl

```
[active]
composite tMotorManager {
    entry sMotorManager eMotorManager;
    entry sInitialize eInitialize;

    call sADConverter       cADConverter;    /* A/Dコンバータ */

attr {
        /* タスク関連設定 */
        [omit] PRI taskPriority = C_EXP("LOW_PRIORITY");
        [omit] SIZE stackSize = C_EXP("STACK_SIZE");

        /* 割込みサービスルーチン */
        [omit] ATR   isrAttribute = C_EXP( "TA_NULL" );
        [omit] PRI   isrPriority = 1;
        [omit] INTNO interruptNumber = TINTNO_TB0;

        /* 割込み要求ライン */
        [omit] ATR   interruptAttribute = C_EXP( "TA_POSEDGE" );
        /*
         * モータ制御周期生成用タイマ(TB0)の優先度
         */
        [omit] PRI   interruptPriority = INTLVL_TB0;

        [omit] volatile uint8_t * const p7 = TADR_SFR_P7;
        [omit] volatile uint8_t * const pd7 = TADR_SFR_PD7;
```

複合コンポーネント組上げ記述

[omit] 指定：
C言語中では参照しない属性
factoryの記述のみ必要であり、
例えばcfgファイル生成に使用

ロータリーエンコーダ複合コンポーネント シグニチャ定義(PUPPY2)

target¥hsb16c29_tm¥puppy2+tecs¥tEncoderManager.cdl

```
signature sEncoderManager {
    /*
     *  初期化
     */
    void    initialize( void );
};
```

公開関数

target¥hsb16c29_tm¥puppy2+tecs¥RotaryEncoder.cdl

```
signature sRotaryEncoderInitialize{
    /*
     *  初期化
     */
    void initialize( void );
};

signature sRotaryEncoder {
    /*
     *  共有変数へのアクセス
     */
    int32_t    getCount( void );
};

signature sEncoderHW {
    /*
     *  割込み処理
     */
    void    phaseA( void );
    void    phaseB( void );
};
```

マネージャ接続関数

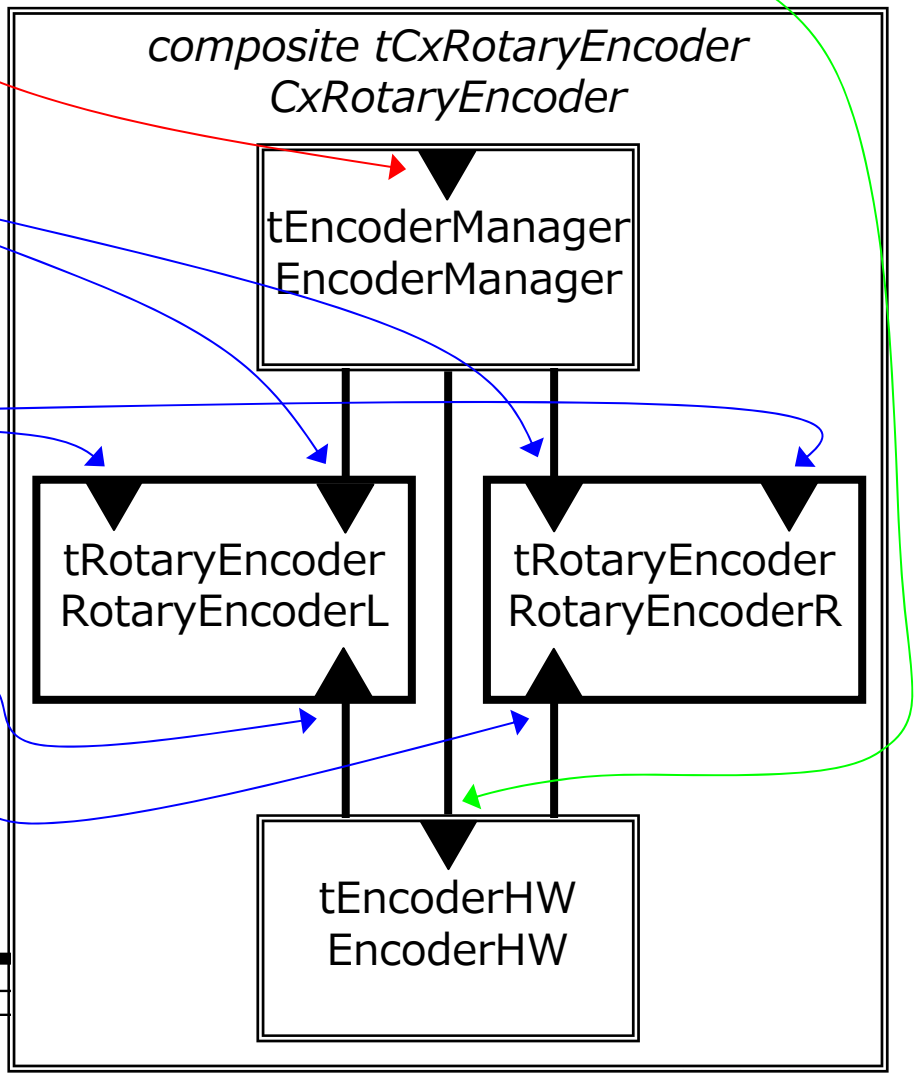
公開関数

依存部向け関数

target¥hsb16c29_tm¥puppy2+tecs¥EncoderHW.cdl

```
signature sEncoderHWInitialize {
    /*
     *  割込みエッジ初期化
     */
    void    initialize( void );
};
```

マネージャ接続関数



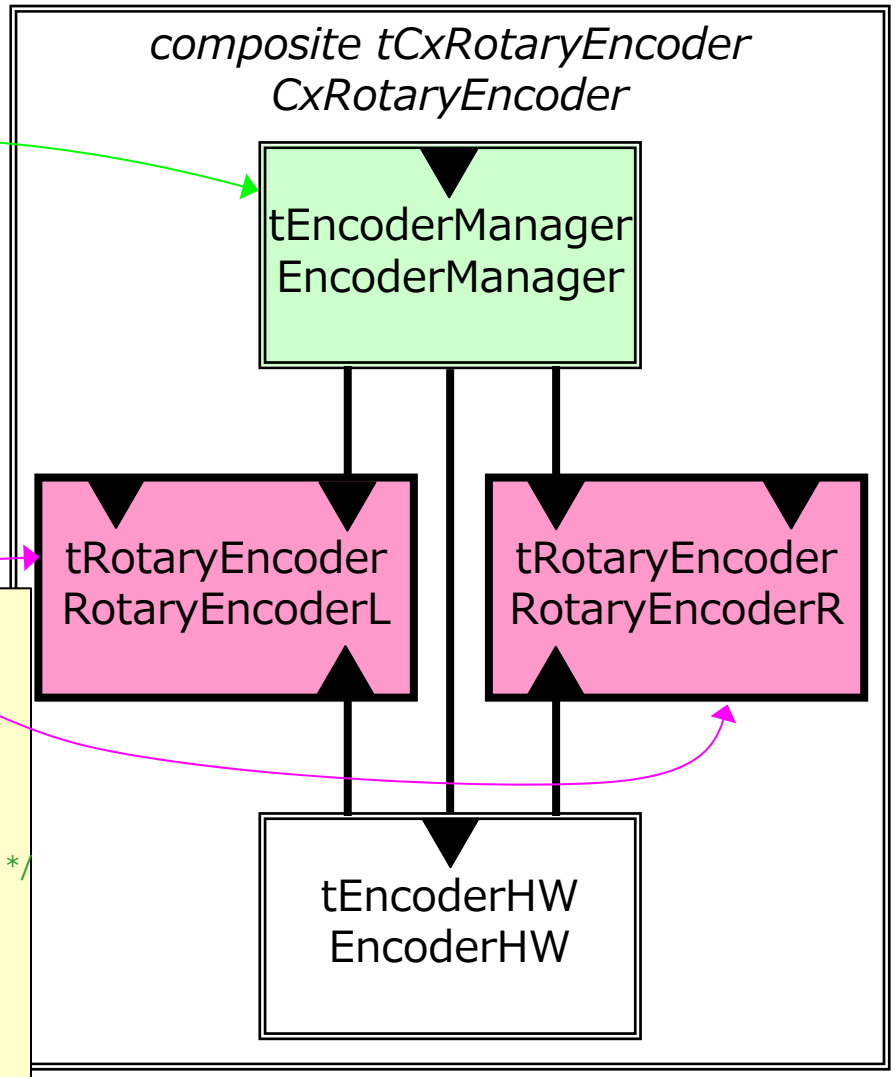
ロータリーエンコーダ複合コンポーネント
セルタイプ定義（抜粋）

target¥hsb16c29_tm¥puppy2+tecs¥EncoderManager.cdl

```
[active]
celltype tEncoderManager {
    entry    sEncoderManager eEncoderManager;
    call     sEncoderHWInitialize cEncoderHWInitialize;
    call     sRotaryEncoderInitialize cRotaryEncoderInitialize;
    call     sRotaryEncoderInitialize cRotaryEncoderInitialize;

    attr {
        /* 割込み要因選択レジスタアドレス */
        volatile uint8_t* ifsr = BREG_SFR_IFSR;
    };
};
```

target¥hsb16c29_tm¥puppy2+tecs¥RotaryEncoder.cdl

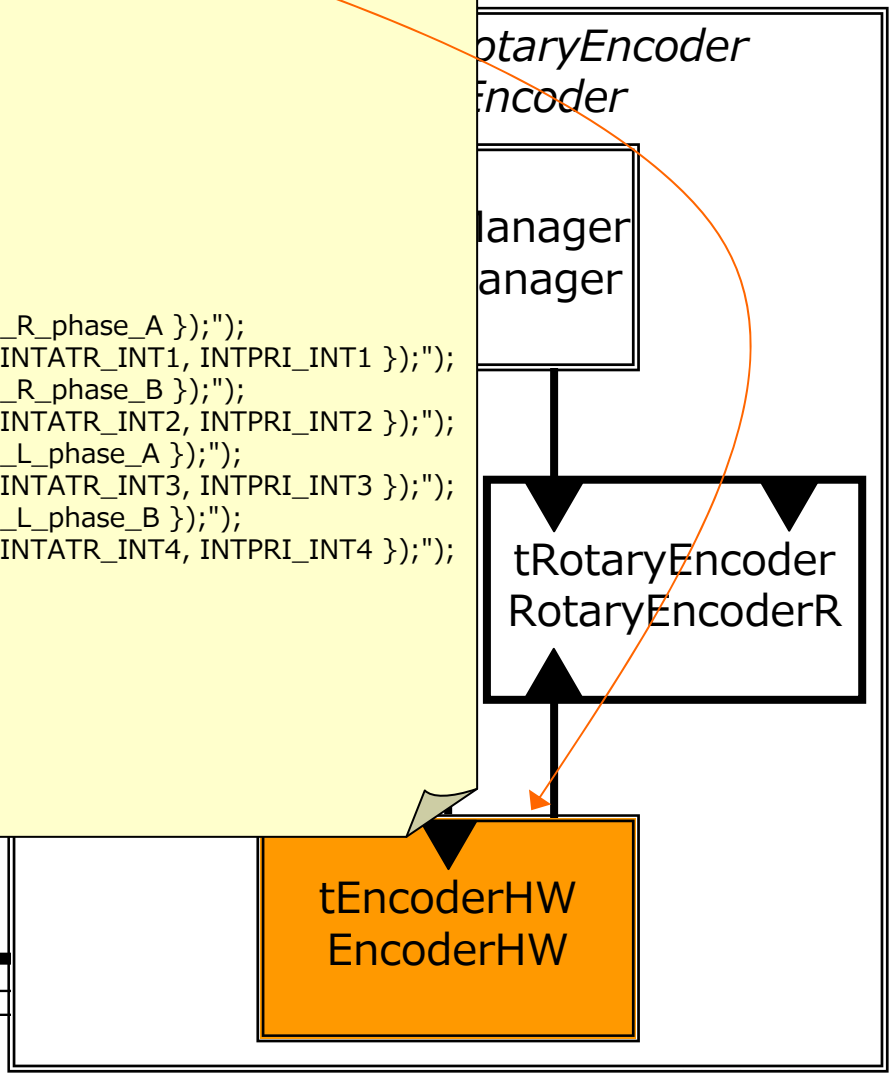
```
celltype tRotaryEncoder {
    entry    sRotaryEncoderInitialize eRotaryEncoderInitialize;
    entry    sRotaryEncoder eRotaryEncoder;
    entry    sEncoderHW eEncoderHW;

    attr {
        uint16_t    interruptPortA;       /* A相割込みポート */
        uint16_t    interruptPortB;       /* B相割込みポート */
        volatile uint8_t* interruptPort;  /* 割込みポートレジスタアドレス */
    };

    var {
        uint32_t count = 0;/* ロータリエンコーダのカウント値を格納 */
    };
};
```



ロータリーエンコーダ複合コンポーネント セルタイプ定義（抜粋）

target¥hsb16c29_tm¥puppy2+tecs¥tEncoderHW.cdl

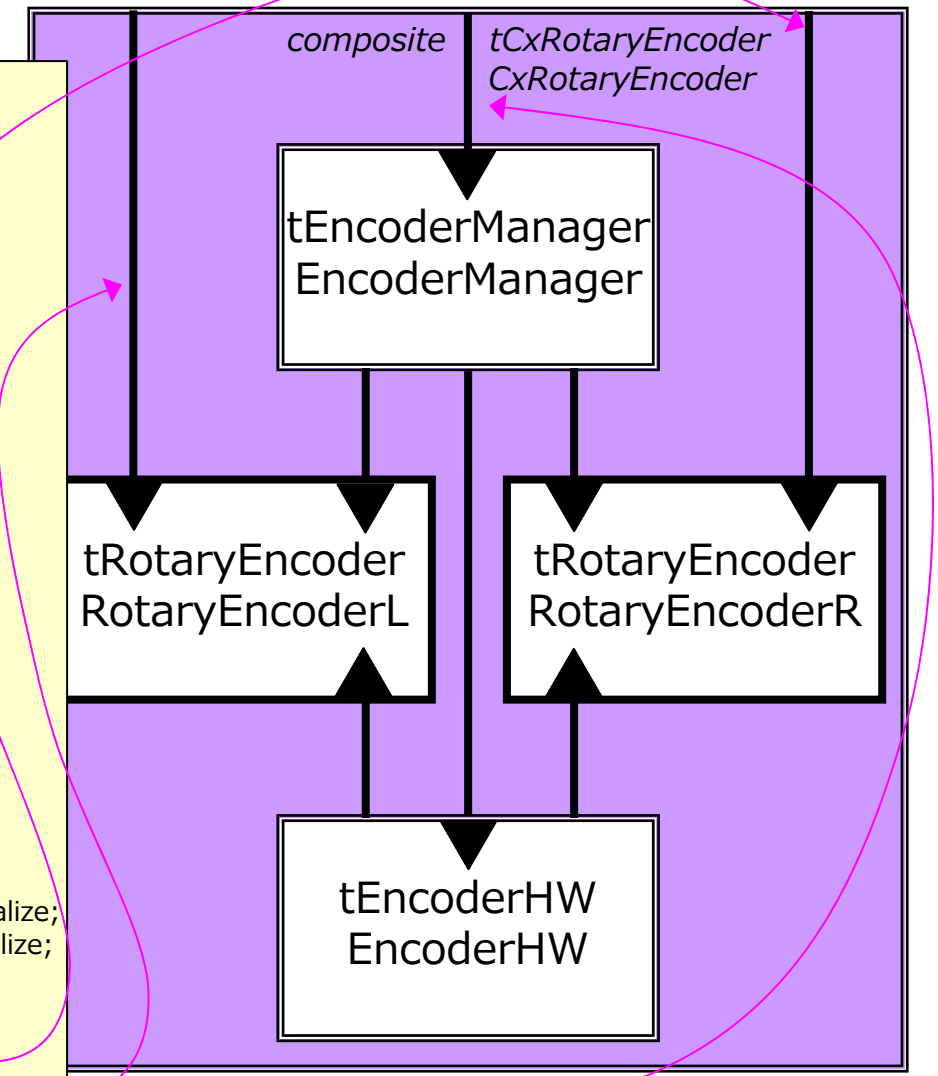
```
[active]
celltype tEncoderHW {
    entry    sEncoderHWInitialize eEncoderHWInitialize;
    call     sEncoderHW cEncoderHWR;
    call     sEncoderHW cEncoderHWL;

    attr {
        volatile uint8_t* intic1 = BREG_SFR_INT1IC;
        volatile uint8_t* intic2 = BREG_SFR_INT2IC;
        volatile uint8_t* intic3 = BREG_SFR_INT3IC;
        volatile uint8_t* intic4 = BREG_SFR_INT4IC;
    };

    factory {
        write("EncoderHW.cfg", "DEF_INH(TINHNO_INT1, { TA_NULL, inh_R_phase_A });");
        write("EncoderHW.cfg", "CFG_INT(TINTNO_INT1, { TA_ENAINT | INTATR_INT1, INTPRI_INT1 });");
        write("EncoderHW.cfg", "DEF_INH(TINHNO_INT2, { TA_NULL, inh_R_phase_B });");
        write("EncoderHW.cfg", "CFG_INT(TINTNO_INT2, { TA_ENAINT | INTATR_INT2, INTPRI_INT2 });");
        write("EncoderHW.cfg", "DEF_INH(TINHNO_INT3, { TA_NULL, inh_L_phase_A });");
        write("EncoderHW.cfg", "CFG_INT(TINTNO_INT3, { TA_ENAINT | INTATR_INT3, INTPRI_INT3 });");
        write("EncoderHW.cfg", "DEF_INH(TINHNO_INT4, { TA_NULL, inh_L_phase_B });");
        write("EncoderHW.cfg", "CFG_INT(TINTNO_INT4, { TA_ENAINT | INTATR_INT4, INTPRI_INT4 });");
    };
    FACTORY {
        write("EncoderHW.cfg", "#include ¥"tEncoderHW_tecsgen.h¥"");
        write("EncoderHW.cfg", "#include ¥"tEncoderHW.h¥"");
        write("asp_prog.cfg", "INCLUDE(¥"EncoderHW.cfg¥")");
    };
};
```



ロータリーエンコーダ複合コンポーネント セルタイプ定義（抜粋）

target¥hsb16c29_tm¥puppy2+tecs¥EncoderHW.cdl

```
[active]
composite tCxRotaryEncoder {

    entry sRotaryEncoder eRotaryEncoderL;
    entry sRotaryEncoder eRotaryEncoderR;
    entry sEncoderManager eEncoderManager;

    cell tRotaryEncoder RotaryEncoderR {
        interruptPortA    = INT_PORT_R_INT1;
        interruptPortB    = INT_PORT_R_INT2;
        interruptPort     = BREG_SFR_P8;
    };

    cell tRotaryEncoder RotaryEncoderL {
        interruptPortA    = INT_PORT_L_INT3;
        interruptPortB    = INT_PORT_L_INT4;
        interruptPort     = BREG_SFR_P1;
    };

    cell tEncoderHW EncoderHW {
        cEncoderHWR = RotaryEncoderR.eEncoderHW;
        cEncoderHWL = RotaryEncoderL.eEncoderHW;
    };

    cell tEncoderManager EncoderManager {
        cEncoderHWInitialize = EncoderHW.eEncoderHWInitialize;
        cRotaryEncoderInitializeR = RotaryEncoderR.eRotaryEncoderInitialize;
        cRotaryEncoderInitializeL = RotaryEncoderL.eRotaryEncoderInitialize;
    };

    eRotaryEncoderR => RotaryEncoderR.eRotaryEncoder;
    eRotaryEncoderL => RotaryEncoderL.eRotaryEncoder;
    eEncoderManager => EncoderManager.eEncoderManager;
};
```



TOPPERS

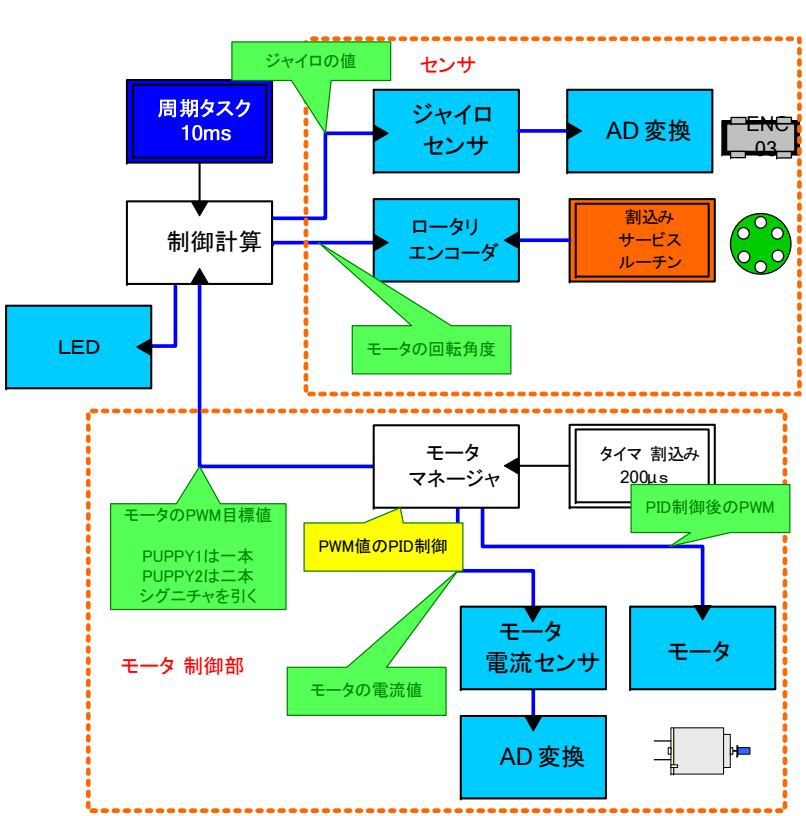
統合後のコンポーネント図の比較

色分け

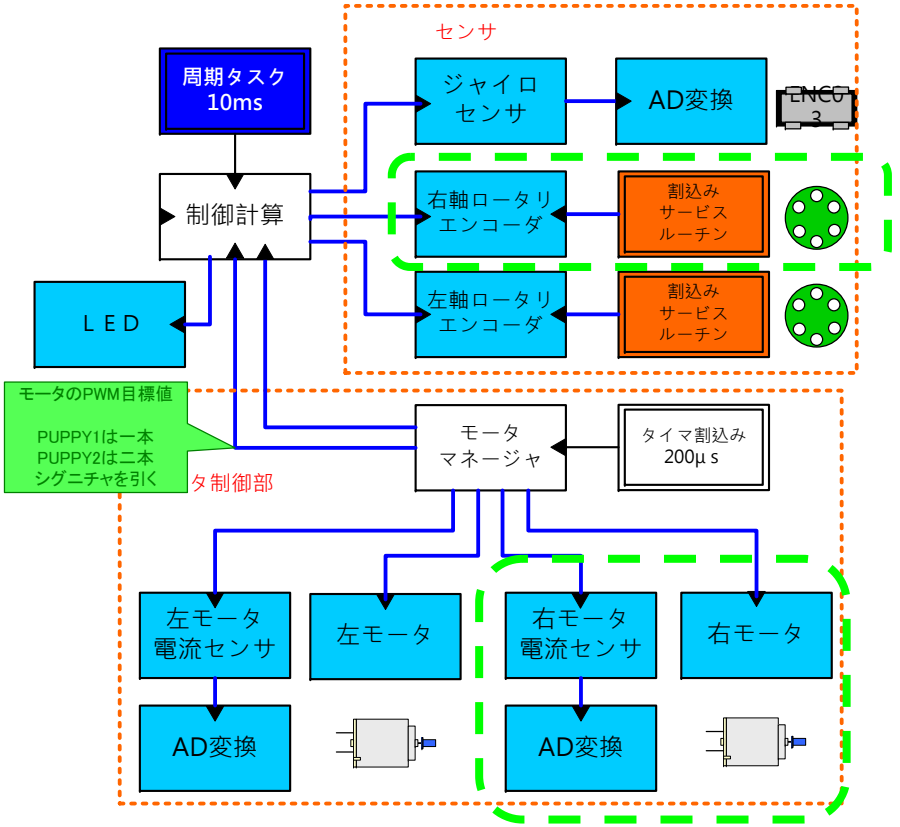
セルタイプが共通なセル

ユニークなセル

- 大部分のセルタイプが同一
- モデルの見通しが向上



PUPPY



PUPPY2

※初期化は省略している

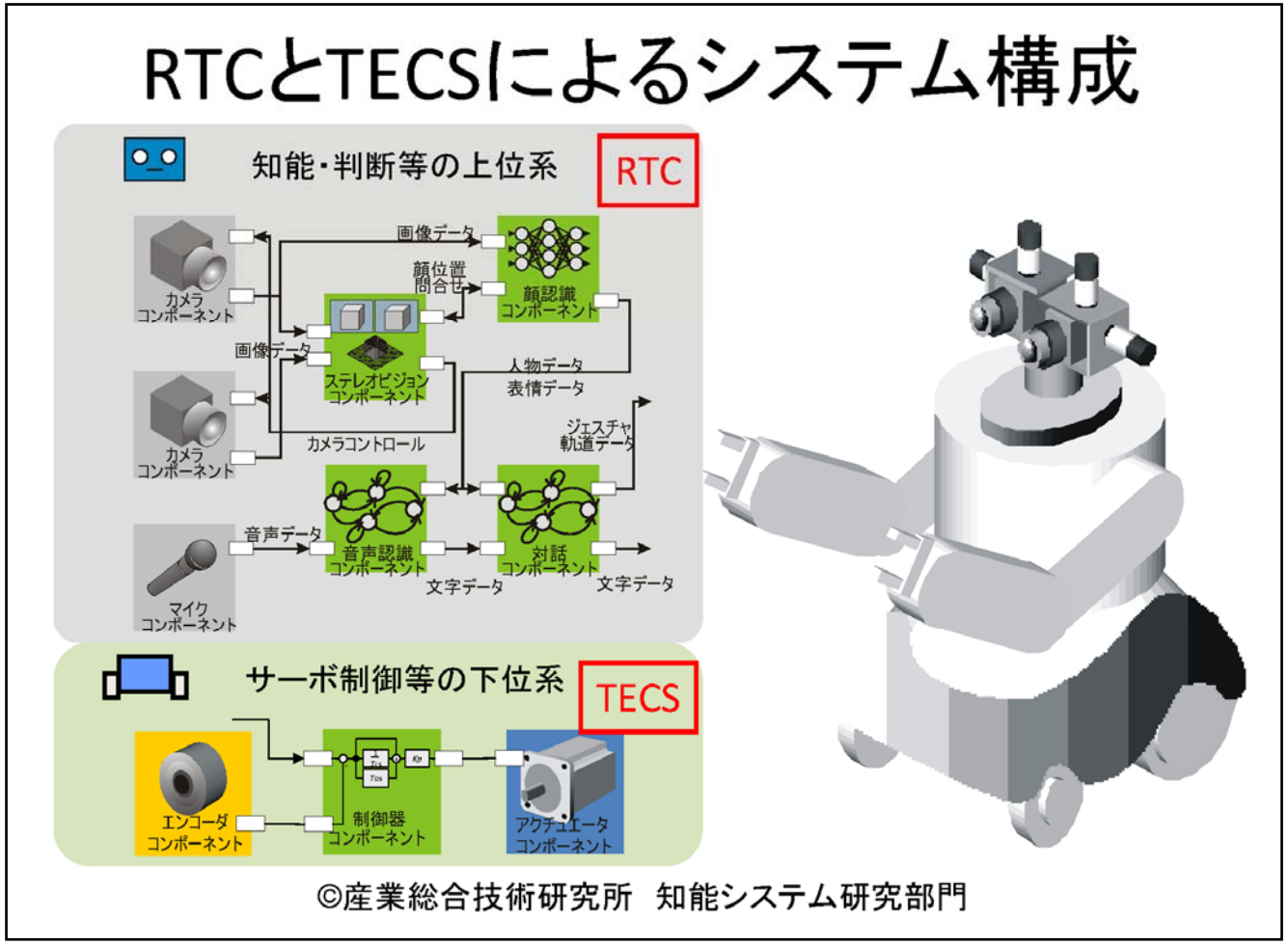
TOPPERS

まとめ

- コンポーネント同士の組上げからソースコードの生成まで一貫して記述できた
  - システム全体の記述までを想定のコンポーネント化技術
    - RPCによる通信の記述や、プラグインによる開発工程の支援により、システム設計からデバッグまで一貫して使用できる
  - 設計図とソースコードとの乖離を未然に抑止する
    - 設計者と実装者間において、矛盾ない情報共有を支援する
- 部品の組み換えと再利用を実現できた
  - セルタイプの概念でコンポーネント流通を実現している
- TECSを利用してシステム内部の表現もできた
  - 開発ターゲット内部のモデルの見通しをよくする

TECSの応用事例

RTC=ロボットシステム開発に資する共通プラットフォーム仕様(RTミドルウェア)のコンポーネント実装の一つとして産総研が開発した Open-RTM-aist が存在する
(URL http://www.is.aist.go.jp/rt/OpenRTM-aist/html/FrontPage.html)



TOPPERS

TECSの応用事例

RTC=ロボットシステム開発に資する共通プラットフォーム仕様(RTミドルウェア)のコンポーネント実装の一つとして産総研が開発した Open-RTM-aist が存在する
(URL http://www.is.aist.go.jp/rt/OpenRTM-aist/html/FrontPage.html)

RTC+TECS

- 下位系(サーボ等)も細粒度コンポーネント指向開発可能
  - センサ・コントローラ・アクチュエータといったレベルでのコンポーネント分割も可能
  - ディペンダビリティ向上
  - 組込みMPUに搭載可能
- 上位系はRTCによるコンポーネント指向開発
  - 動的結合による柔軟なシステム構成が可能に
  - 多様なOS・言語で開発が可能
  - バイナリレベルでの再利用性

©産業総合技術研究所　知能システム研究部門

TOPPERS

コンポーネント化 TIPS

コンポーネント化 TIPS

- セルタイプに機能がわかる名前をつけよう
  - ユースケースから考えられる名前をつけることで、コンポーネント図の見通しがよくなります
    - 反例）センサー系はADCコンポーネントに入れ込んだ
      →ジャイロや電流センサはどこに・・
    - 改善）ADCを呼び出す各種コンポーネントを命名して配置する
      その後でセンサー系複合コンポーネントにしてもよい

！汎用的なコンポーネントの設計に繋がる
→機能を隠蔽するのでなく部品の独立性を保たせる
・機能はある程度まで粒度を小さく分割する
→部品のポータビリティを確保できる

コンポーネント化 TIPS

- 定数はできるだけ使用する場所で定義しよう
  - CソースとCDLの記載とは分離して検討できます
  - 大域定数は global_tecsgen.h に集約されます
  - 定義と使用箇所が組みになり保守性が上がります

！C++の利点をローコストで利用できる
→カプセル化が擬似的に使える
→移植時の定数の再定義に利用できる
global_tecsgen.h の出力を代用する

コンポーネント化 TIPS

- できるだけセルタイプを再利用しよう
  - チャンネル・IDの違いがあるのみで同じインターフェースが提供されるような場合にはセルタイプが活用できます
  - 属性の初期値はセル生成時かセルタイプ宣言時を選べます
  - レジスタのアドレスなども属性にできます

！コンポーネントの複製の手間を省ける
→複合コンポーネントごと複製することで、
全く同じコードを全て用意する手間が省ける
→初期値の設定段階を選択できるので、
ターゲットやアプリケーションに依存した定数等
を含まない抽象度の高い部品が書ける

コンポーネント化 TIPS

- CELLCBを活用しよう
  - CELLCB （セルコンテキストブロック）
    - セルタイプの実体となるセルの識別に使用されます
    - シグニチャや属性の短縮形が使えます

！保守性の向上を狙った設計を容易にする
→APIの引数にCELLCBを渡すことにより
関数の汎用化を促進できる