

TOPPERS基礎実装セミナー

(Raspberry Pi Pico(W)/Pico2(W)版:基本)

Appendix

TOPPERSプロジェクト
教育ワーキング・グループ

本教材の利用条件

NEXCESS基礎コース01 組込みソフトウェア開発技術の基礎

Copyright (C) 2006-2007 by 名古屋大学 組込みソフトウェア技術者人材養成プログラム

Copyright (C) 2006-2007 by 本田晋也

上記著作権者は、以下の(1)～(4)の条件を満たす場合に限り、本コンテンツ(本コンテンツを改変・翻訳したものを含む、以下同じ)を使用・複製・改変・翻訳・再配布(以下、利用と呼ぶ)することを無償で許諾する。

- (1) この枠内の著作権表記等が、そのままの形でコンテンツ中に含まれていること。
- (2) 本コンテンツを再配布する場合には、再配布の形態等を、以下のウェブサイトから報告すること。
<http://www.nces.is.nagoya-u.ac.jp/NEXCESS/REPORT/>
- (3) 本コンテンツを改変・翻訳する場合には、コンテンツを改変・翻訳した旨の記述を、コンテンツ中に含めること。また、改変・翻訳者の著作権表記等は、この枠内の著作権表記等とは別に行うこと。
- (4) 本コンテンツの利用により直接的または間接的に生じるいかなる損害からも、上記著作権者を免責すること。

※ 本コンテンツの一部は、文部科学省 科学技術振興調整費により、名古屋大学 組込みソフトウェア技術者人材養成プログラム(NEXCESS)の一環として作成しました。

※ 本コンテンツ中に記載されている商品名やサービス名などは、各社の商標または登録商標です。

本ドキュメントに関して

1. 著作権に関する表記

＜TOPPERS基礎実装セミナー(Paspberry Pi Pico(W)/Pico2(W)版:基本)アペンデックス＞

Copyright (C) 2006-2007 by 名古屋大学 組込みソフトウェア技術者人材養成プログラム

Copyright (C) 2006-2007 by 本田晋也 名古屋大学

Copyright (C) 2007-2025 by 竹内良輔 TOPPERSプロジェクト

Copyright (C) 2014-2015 by 田中裕貴 (株)リコー

上記著作権者は、以下の (1)～(3) の条件を満たす場合に限り、本ドキュメント（本ドキュメントを改変したものを含む。以下同じ）を使用・複製・改変・再配布（以下、利用と呼ぶ）することを無償で許諾する。

- (1) 本ドキュメントを利用する場合には、上記の著作権表示、この利用条件および以下の無保証規定が、そのままの形でドキュメント中に含まれていること。
- (2) 本ドキュメントを改変する場合には、ドキュメントを改変した旨の記述を、改変後のドキュメント中に含めること。
ただし、改変後のドキュメントがTOPPERSプロジェクト指定の開発成果物である場合には、この限りではない。
- (3) 本ドキュメントの利用により直接的または間接的に生じるいかなる損害からも、上記著作権者およびTOPPERSプロジェクトを免責すること。また、本ドキュメントのユーザまたはエンドユーザからのいかなる理由に基づく請求からも、上記著作権者およびTOPPERSプロジェクトを免責すること。

本ドキュメントは、無保証で提供されているものである。上記著作権者およびTOPPERSプロジェクトは、本ドキュメントに関して、特定の使用目的に対する適合性も含めて、いかなる保障もしない。また、本ドキュメントの利用により直接的または間接的に生じたいかなる損害に関しても、その責任を負わない。

2. 本ドキュメントに関するご意見・ご提言・ご感想・ご質問等がありましたら、TOPPERSプロジェクト事務局までE-Mailにてご連絡ください。
3. 本ドキュメントの内容は、内容の改善や適正化の目的で予告無く改定することがあります。

本ドキュメントでは、Microsoft社のClip Art Galleryコンテンツを使用しています。

TRONは”The Real-time Operating system Nucleus”の略称です。ITRONは”Industrial TRON”の略称です。
μITRONは”Micro Industrial TRON”の略称です。TOPPERS/JSPはToyohashi Open Platform for Embedded Real-Time System/Just Standard Profile Kernelの略称です。」

本ドキュメント中の商品名及び商標名は、各社の商標または登録商標です。

目次

基礎1 補足

- ・1日目 組込み技術 : 補足資料
- ・1日目 ハードウェアを使ったデバッグ環境
- ・2日目 割込みプログラミング : タイマ割込みプログラム
- ・2日目 割込みプログラミング : シリアルI/O割込みプログラム
- ・2日目 カップラーメンタイマの作成

基礎2 補足

- ・1日目 リアルタイムOSの基礎: 補足資料
- ・2日目 カップラーメンタイマ(リアルタイムOS版)の実装

基礎1 補足

1日目 組込み技術

組込み機器・組込みシステム

- 組込みシステムとは、各種の機器に組みこまれて、それを制御する組込みシステムのこと（一般のパソコン、コンピュータは、この範疇に含まれない）

➡ 明確な規定はない

- 組込みシステムは、機器の一部とみなされる

組込みシステムの範囲はさまざま

- パソコン、汎用コンピュータなどのコンピュータの形をしているものの以外は組込みシステム
- 狭い意味での組込みシステム（外見がコンピュータでないもの）をdeeply embedded systemと呼ぶ場合もある

組み込みシステムの適応機器の例

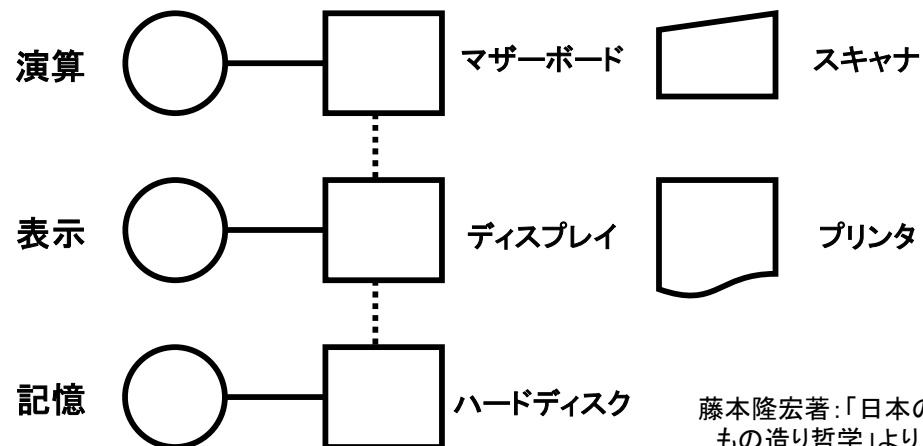
- 家電機器(電子レンジ、炊飯器、乾燥機、エアコン)
- AV機器(テレビ、ビデオ、デジタルビデオ、オーディオ機器)
- 娯楽／教育機器(ゲーム機、電子楽器、カラオケ、パチンコ)
- 個人用情報機器(PDA、電子手帳、カーナビ)
- パソコン周辺機器(プリンタ、スキャナ、ディスク、DVDドライブ)
- OA機器(コピー、FAX)
- 通信機器 端末(電話機、留守番電話、携帯電話)
- ネットワーク設備(交換機、PBX、ネットワークルータ、HUB)
- 運輸機器(自動車、信号機、鉄道車両／制御、航空機、船舶)
- 工業制御／FA機器(プラント機器、工作機器、工業用ロボット)
- 設備機器(ビル用照明、ビル用空調、ビル用電力システム、エレベータ)
- 医療機器／福祉機器(血圧計、心電計、レントゲン、CTスキャン)
- 宇宙／軍事(ロケット、人工衛星、ミサイル)
- その他の事務用機器(事務用データ端末、POS端末、自動販売機)
- その他の計測機器(シンクロスコープ、ICテスタ、電子メータ)

パソコンは組み合わせ(モジュラー)型製品の代表

- 製品アーキテクチャは組み合わせ(モジュラー)型と摺り合わせ(インテグラル)型があると言われています
- パソコンはいろいろな部品を組み合わせることにより、製品として成り立つ組み合わせ型製品の代表と言われています
- しかし、パソコンの部品は摺り合せ型、制御ソフトウェアは「パソコン型ソフトウェア」では開発できません

➡ 部品自体がコストアップするため

高機能化しなければ
ビジネスモデルが
成り立たない



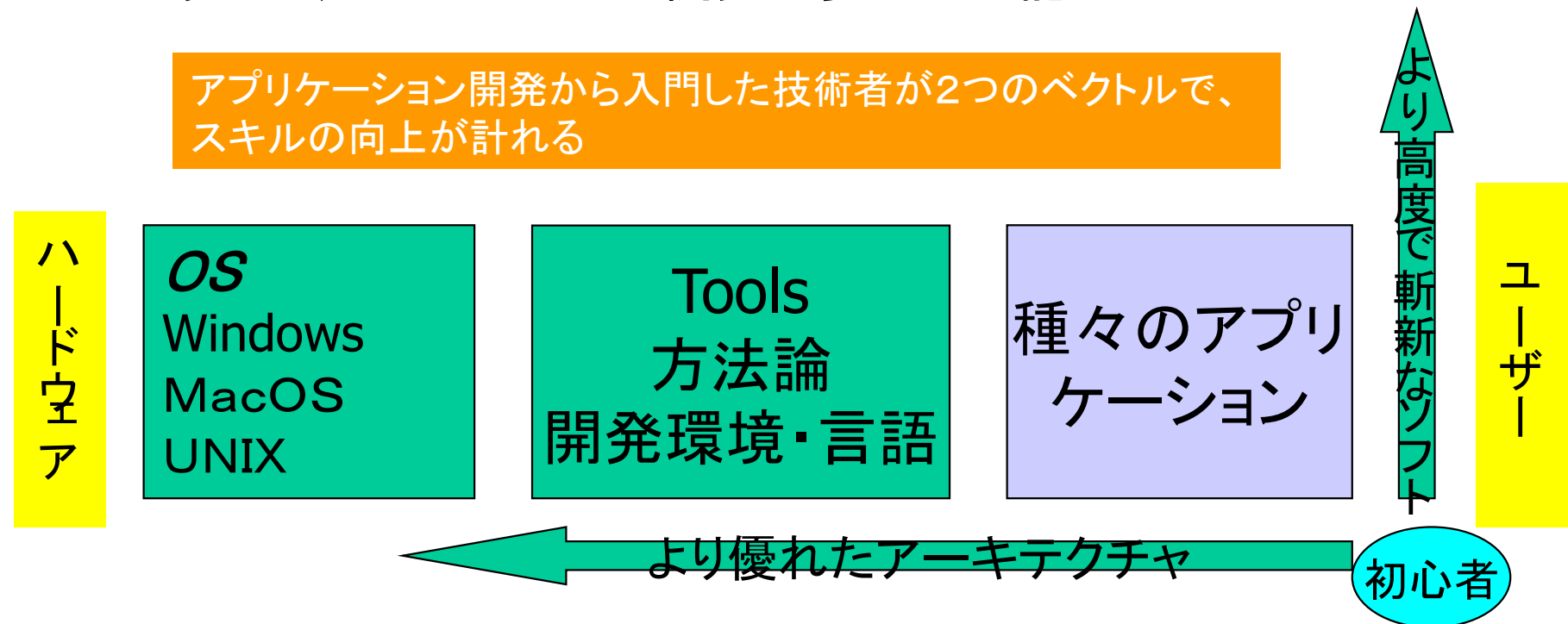
藤本隆宏著:「日本の
もの造り哲学」より

パソコンのアプリは常に高速化・高機能化。より自由に開発できるように大容量のメモリ・ストレージを求めます

➡ パソコンは常に高機能化しなければならない

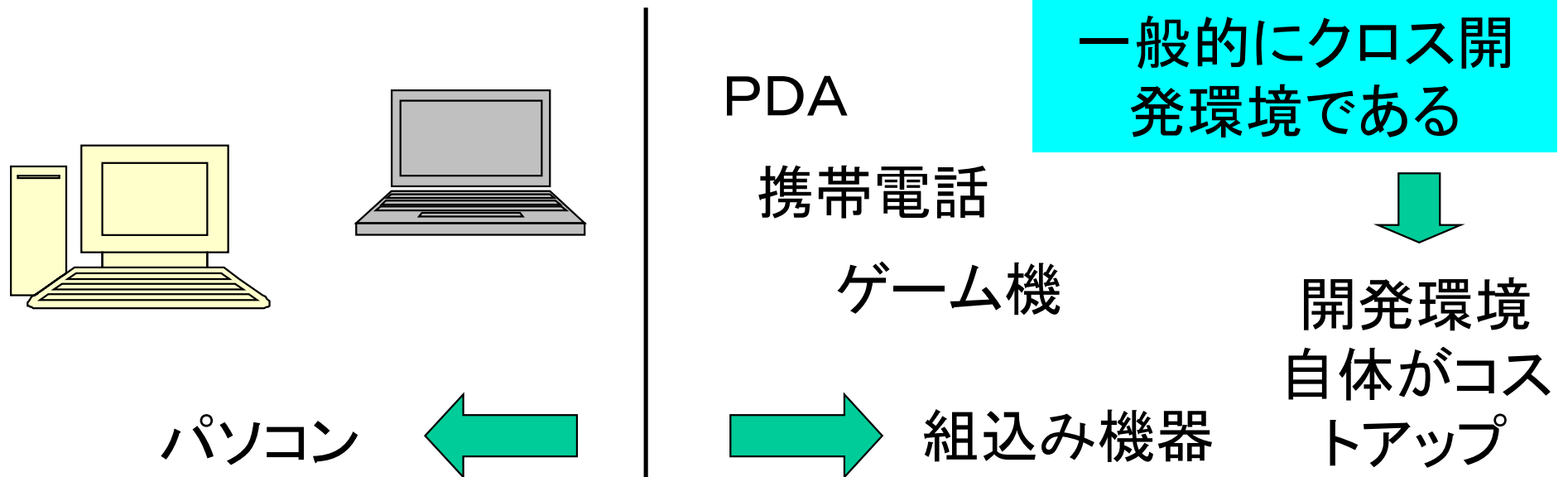
パソコンのビジネスモデル・開発モデル

- パソコン型ソフトウェア開発は、セルフ開発環境であるため、パソコン環境でアプリケーションの開発が可能
ソフトウェア開発者は、より高度な機能、斬新なアプリケーションを実機上で開発でき、スキルを向上可能である
プラットフォームの改善や新しいToolの開発が必要な場合は、能力さえあれば、OSやツールの開発に参加が可能

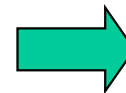


パソコンと目的がかぶる組み込み機器

- パソコンのアーキテクチャのまま、情報機器化するとモバイルまで、コストがアップし、性能は落ちる



「パソコン型ソフトウェア開発」
そのままでは、組み込みシステムは開発できない



性能・コスト
が必ず満た
せない

規模からみた組込みシステムの分類

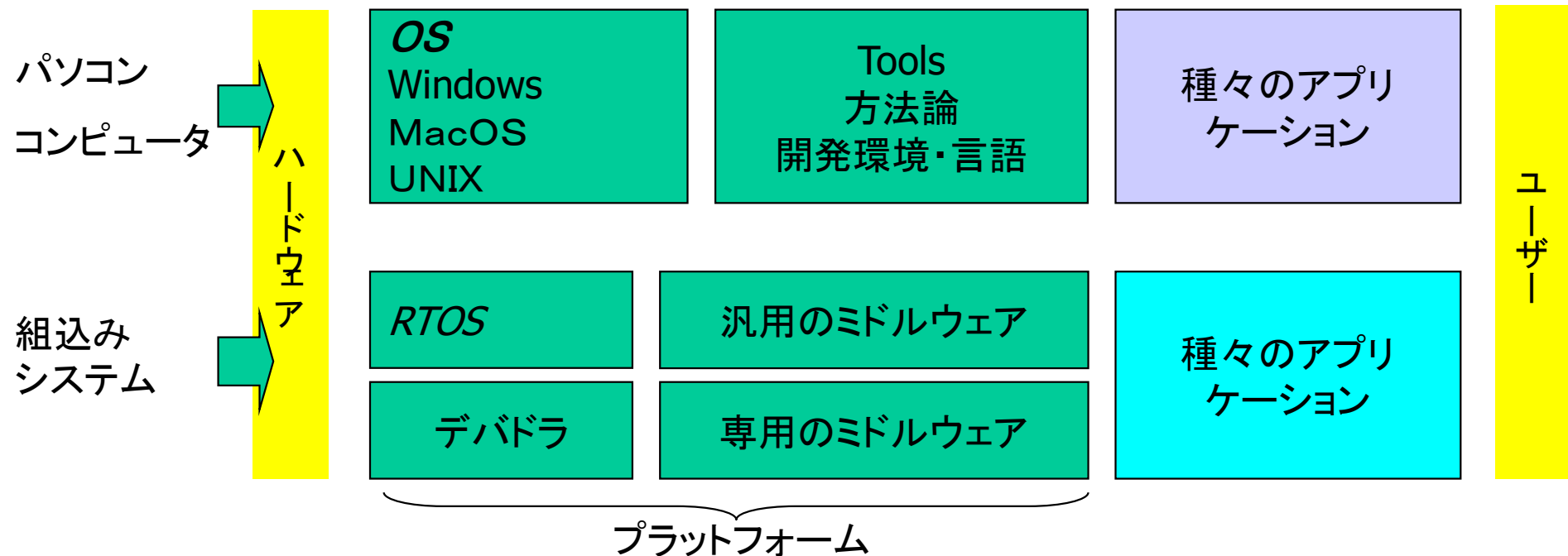
- 組込み機器の規模から3つの組込みシステムに分類する
 - (1) 小規模組込みシステム
ワンチップCPUで制御を行う小規模な組込みシステム、ROMやRAMを拡張することもある。RTOSを用いないシステムが多い。
 - (2) 中規模組込みシステム
32ビットクラスのCPUを使うシステム、ソフトウェア規模も大きくRTOSを用いてシステムの制御を行う
 - (3) 大規模組込みシステム
パソコンに近い規模の組込みシステム、実際にパソコンやワークステーションを制御用に用いる場合もある。

小規模組込みシステムの技術

- クロス開発となるため、開発環境を用意しないと開発自体ができない・・・(1)
 - 電源オンからアプリケーションプログラムが実行するまでのプログラムを自作する必要がある。割込み等も自分で実装(パソコンではOSが代行してくれる)
 - 簡単なプログラムミスでハングアップ(機器が未応答になる状態)や暴走が発生する。これを解決するためにICE等の特別な開発ツールを用いてプログラムの実行を監視したり、地道にチェックポイントにLEDを点ける等のデバッグテクニックが必要。・・・(2)
 - デバイス等がインテリジェント化し、内容を理解してデバイスドライバを作成しなければならない。開発にはハードウェアとソフトウェアの両方のスキルが必要。・・・(3)
- 家電機器、簡単なパソコン周辺機器、電子おもちゃ等

中規模組込みシステムの技術(1)

- パソコンほどソフトウェア規模はないが、全てのソフトウェア開発、品質保証を行わなければならない
- コンピュータシステムのOS、Toolにあたる部分がプラットフォームと呼ばれ、設計の良し悪しが組込み機器の品質・性能・コストに大きく影響する。設計は企業秘密にあたり公開はされない・・・(4)



中規模組込みシステムの技術(2)

- 専用ミドルウェアは機種独自の機能性能を達成するソフトウェアである。ハードウェアと協業して機能を達成する場合が多い。企業秘密にあたる部分で一般には公開されない・・・(5)
- 汎用ミドルウェアはプロトコルスタックやファイルシステムのような社外から導入するソフトウェアである。機能を取り込んだり、評価を行うために、やはり、ノウハウが必要。テストや最悪市場障害発生時等、社内の対応ができるようにする必要がある。

プリンタシステムの専用ミドルウェアは、画像処理やデータレンダリングや変換処理を行うプログラムが、これにあたる。

ゲーム機、プリンタやFAX、カーナビ、デジカメ等

大規模組込みシステムの技術

- パソコンやワークステーションのシステムをそのまま使用する組込みシステム
プラント制御、航空機、銀行ATM等
- 機器の規模が大きくなればなるほど、パソコン型ソフトウェア開発を、そのまま使用したほうが開発効率が良い、それでも、機器の非機能要件によっては、中規模組込みシステムの延長のアーキテクチャを採用するものも多い。
携帯電話等
- 組込みLinuxは中間型と言える
- マルチコアプロセッサを使用するために、Linuxをベースにするシステムが増えつつある

組込み開発形態からまとめスキルと問題点

- パソコン型ソフトウェア開発の熟練したアプリ開発者でも、組込みシステムを円滑に開発するのは難しい
理由は(1)～(5)までにある

(1)クロス開発であるために環境を整えないと開発体験ができない・・・本基礎セミナーの目標

(2)パソコン型ソフトウェア開発に比べ、安全性の面でアプリケーションの開発が難しい

(3)ソフトウェアとハードウェアの両方のスキルをもった技術者が必要

(4)適切なプラットフォームの開発には熟練と経験が必要

(5)企業秘密に属する技術は企業に入らないと教育ができない

組み込みソフトウェア教育の問題点

- 前述の通り、パソコン型ソフトウェア開発の教育を、そのまま組み込みソフトウェア教育に適用しようとしても、必ず、不足が発生する
- パソコンのプラットフォームにあたる部分の開発を日本では行っていないため技術補完がされない。また、学校等の長期のカリキュラムでの教育が必要である。
- 教育の形で解決できない技術取得があり、職場の先輩等のコミュニケーションがその代用を果たすケースが多い(新人技術者の一番重要なスキルかもしれない)。また、企業の組織自体も、この点を考慮しない組織を構築すると円滑に機能しない場合がある。(パソコン型ソフトウェア開発は考慮しなくても開発が行えるプロセスを持つ)

基礎1 補足

ハードウェアを使ったデバッグ環境

ICE (In-Circuit Emulator)

- ターゲットボード上のプロセッサを外し、代わりにプロセッサをエミュレーションするICEを接続する
- プロセッサの動きをリアルタイムに監視し、必要なタイミングで動作を止める/変えることができる
- “フルICE”とも呼ばれる
 - ➡ システムを停止させないデバッグ作業を支援
 - ユーザーインタフェース部分はホスト上で動作させる



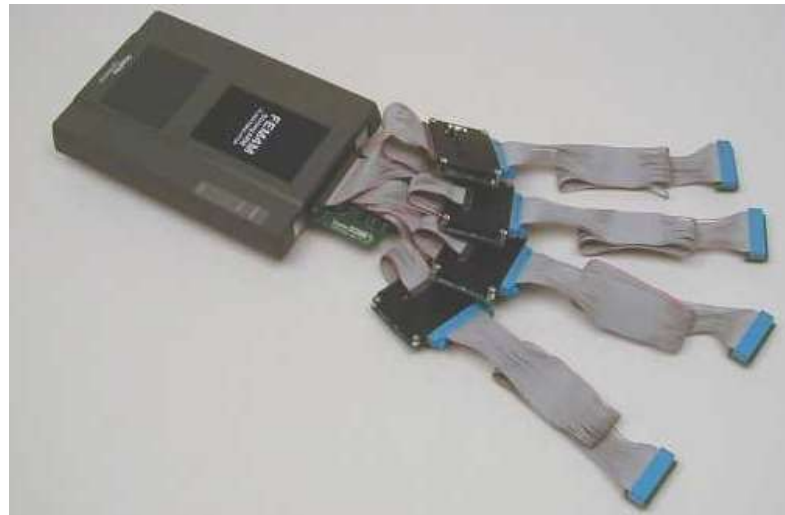
ICEの問題点

- プロセッサ毎に開発しなければならないため高価
➡ ROMエミュレータ
- ワンチップ化やキャッシュにより, チップの外部からバスが観測できない
➡ エバチップ (Evaluation Chip)
➡ オンチップデバッグ機能
- プロセッサの高速化により, ブループを指すと動作しなくなる/動作が不安定になる
➡ オンチップデバッグ機能

ROMエミュレータ/エバチップ

ROMエミュレータ

- プロセッサではなく, ROMソケットにプルーブを差すことでプロセッサの動作を監視
- プロセッサの動作を止める/変えるためには, NMIを使う



エバチップ (Evaluation Chip)

- プロセッサ内部のバスを外部に引き出した, デバッグ時専用のプロセッサ

オンチップデバッグ機能

- デバッグ支援回路をチップ内に実装する
- 外部のインターフェースユニットを経由してホストパソコンから制御
- ICE(のサブセット)がチップ内に入っているイメージ



基礎1 補足

2日目 割込みプログラミング : シリアルI/O割込みプログラム

セミナーでは時間制約があり、実習できなかったコンテンツを紹介します。

本章は、本Appendix「宿題：タイマ割込みプログラム」まで学習後、実施して下さい。

シリアルI/O割込みプログラム : int_uart

- シリアル受信割込みを用いて, ポーリングプログラム uart と同様の機能を実現する
- 学習内容
 - シリアルI/O割込みの扱い方
- 作成方法
 - ポーリングプログラム uart をベースに作成

int_uart : 作成

プログラム int_uart を作成する

- 手順
 - 割込み要因ハードウェアの初期化 (UART2)
 - ポーリングプログラム uart を流用
 - 割込み制御レジスタの初期化
 - 割込み優先度の設定
 - 受信割込みと送信割込みは独立
 - 割込みハンドラ
 - データをシリアルI/Oから受信して, LEDをカウントアップし, 受信したデータをシリアルI/Oに送信する

int_uart : 割り込み制御レジスタの初期化

- int_timer で使った init_int 関数を使用
 - INT_VECTOR_UART0 は rp2040.h と rp2350.h で定義
- uart の初期化関数に割り込み設定を追加

```
void
uart0_init(void){
    :
    /*
     * 受信割り込み設定
     */
    *TREG_UART0_UARTIMSC = UART_UARTIMSC_RXIM;
    *TREG_UART0_UARTIFLS = 0;
#ifdef __riscv
    hazard3_set_vector(INT_VECTOR_UART0, UART0_IRQHandler);
#endif
    init_int(INT_VECTOR_UART0, 2, ENABLE);
}
```

割り込み設定を追加

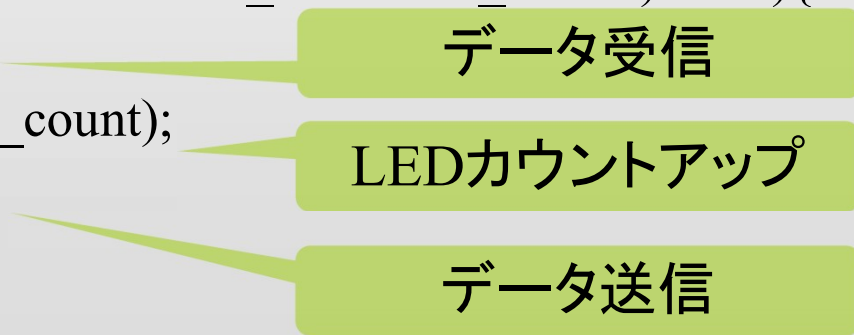
int_uart : 割り込みハンドラ

- LEDをカウントアップし, データを受信して, データを送信

```
uint32_t led_count;

void
UART0_IRQHandler(void){
    uint32_t imsc = *TREG_UART0_UARTIMSC;
    uint32_t ufr = *TREG_UART0_UARTFR;
    uint8_t c;

    if ((imsc & UART_UARTIMSC_RXIM) != 0
        && (ufr & UART_UARTFR_RXFE) == 0){
        c = uart0_pol_getc();
        led_out_bdigit(++led_count);
        uart0_pol_putc(c);
    }
}
```



int_uart: メイン関数

- LEDの初期化後にUART0の初期設定を行う

```
void
_main(void){
    led_init();
    uart0_init();
    led_count = 0;

    for(;;){
    }
}
```

基礎1 補足

2日目 カップラーメンタイマの作成

セミナーでは時間制約があり、実習できなかったコンテンツを紹介します。
本章は、本Appendix「シリアルI/O割込みプログラム」まで学習後、
実施して下さい。

カップラーメンタイマの作成

1. 仕様
2. プログラムの作成
 - 1) Step1
 - 2) Step2
 - 3) Step3
 - 4) Step4
 - 5) Step5
 - 6) Step6

カップラーメンタイマの仕様

- 概要

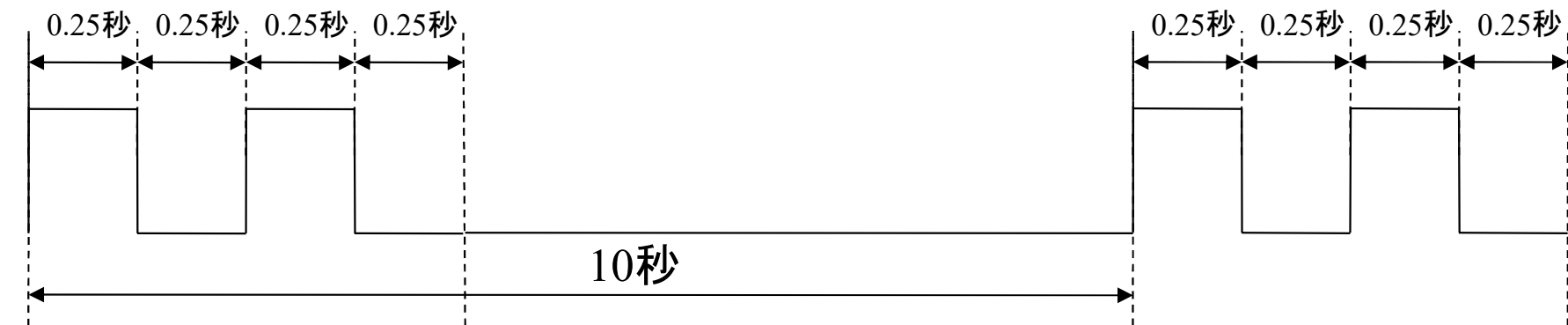
- タイマを起動してから、設定した時間が経過したことを知らせる
- 設定時間はDIPSWの設定に従う
 - 00-30秒、01-1分、10-2分、11-3分

- スイッチとLEDの使い方

- LED1 : 電源がONであること
(プログラムが動いていること)を表示
- LED2 : SW1が押されたことを表示
- : タイマが動いていることを表示
- : タイムアウト時30秒間点滅
- SW1 : タイマを起動・停止

カップラーメンタイマの仕様

- 外部仕様
 - 電源をONの間(プログラムが動いている間)、LED1の点灯・消灯を1秒間隔で繰り返す
 - SW1がONになると設定時間を30秒にして、タイマを起動する
 - SW1がOFFになると、タイマを停止する
 - タイマが動いている間にSW1がONになると、ONの間LED2を点灯させ、設定時間を30秒延長する
 - タイマが動いている間は、10秒間隔で、LED2を2回点滅させる。LED2の2回点滅は、点灯・消灯を0.25秒間隔で2回繰り返すことで行う
 - 設定時間が経過すると、LED2を15秒間点滅させた後、消灯する。LED2の点滅は、点灯・消灯を0.25秒間隔で繰り返すことで行う



カップラーメンタイマの作成

1. 仕様
2. プログラムの作成
 - 1) Step1
 - 2) Step2
 - 3) Step3
 - 4) Step4
 - 5) Step5
 - 6) Step6

カップラーメンタイマプログラム `cup_timer`

- カップラーメンタイマーの仕様を実現するプログラムを作成する
- プログラムの構造や処理の流れについては、構造化詳細設計演習の結果を用いる
- ハードウェアの操作に関してはこれまでに学習した内容を用いる
 - LEDの点灯
 - スイッチ（ディップ、プッシュスイッチ）のスキャン
 - タイマによる時間生成
 - 割込みによる時間通知

作成の手順

5つのステップに分けて順に進める

- Step1 : cup_timer_step1
 - デバイスドライバの作成
- Step2 : cup_timer_step2
 - タイマによる基本時間(0.25秒)の作成
- Step3 : cup_timer_step3
 - メイン関数でのLED1の1秒間隔の点灯の実現
- Step4 : cup_timer_step4
 - スイッチプロセスの作成
- Step5 : cup_timer_step5
 - タイマプロセスの状態遷移判定と各状態の大枠作成
 - LED4の点灯の実現
- Step6 : cup_timer
 - タイマプロセスの状態遷移判定と各状態の詳細

カップラーメンタイマの作成

1. 仕様
2. プログラムの作成
 - 1) Step1
 - 2) Step2
 - 3) Step3
 - 4) Step4
 - 5) Step5
 - 6) Step6

Step1 : デバイスドライバの作成(device.h/device.c)

LED, スイッチ(プッシュ), タイマの操作関数を作成

- LED, ディップスイッチ, プッシュスイッチ
 - ポーリングプログラムの初期化関数と操作関数を流用
 - LED1,2を個別にON,OFFする関数を追加

```
void set_led_state(uint32_t led, uint8_t state);
```

- SW1の状態を個別に読み込む関数を追加

```
uint8_t get_sw1_state(void);
```

- ON,OFF状態を示すマクロを定義

```
#define ON          1    /* LEDやスイッチON状態  */  
#define OFF         0    /* LEDやスイッチOFF状態 */
```

Step1: デバイスドライバの作成(device.h/device.c)

- UART0の制御
 - UART0から受信データを取り出す

```
uint8_t uart2_pol_getc(void);
```

- UART0にデータを送信する

```
void uart2_pol_putc(uint8_t c);
```

Step1 : デバイスドライバの作成

- タイマ
 - タイマ割込みプログラムの初期化関数を流用
 - 初期設定後, スタートさせ周期的に割込みハンドラを実行する
 - 1msecでタイムアウトするようにカウント値を変更する
 - タイマ割込みプログラムでは100msecで設定

Step1 : プログラム例 LEDの個別ON,OFF関数

- LEDの状態保存関数

```
uint32_t LedState;
```

- LEDの個別ON,OFF関数

```
void  
set_led_state(uint32_t led, uint8_t state){  
    if (state == ON) {  
        LedState = LedState | led;  
    } else {  
        LedState = LedState & ~led;  
    }  
    led_out(LedState);  
}
```

Step1 : プログラム例 スイッチの個別読み込み関数

- 必要な定数と変数の定義

```
#define TIMER_COUNT 1000                /* 1msec */  
#define ALARMNO      0  
int switch_value = 0;  
unsigned int wrap_time = TIMER_COUNT;  
void (*timer_func)(void) = 0;  
static void (*gpio_func[NUM_BANK0_GPIO])(void) = { 0 };
```

- スイッチの押下ごとの状態に変えてON/OFFを取り出す

```
uint8_t  
get_sw1_state(void){  
    if ((switch_value & 1) == 0)  
        return OFF;  
    else  
        return ON;  
}
```

Step1 : プログラム例 タイマハンドラ

- タイムアップ時、コールバック関数(timer_func)を読み出し

```
void
Timer0_IRQHandler(void){
    uint64_t delay_time;
    uint32_t ints = sil_rew_mem((uint32_t *)(TADR_TIMER_BASE+TOFF_TIMER_INTS));
    /*
     * 割込みクリア
     */
    sil_wrw_mem((uint32_t *)(TADR_TIMER_BASE+TOFF_TIMER_INTR), ints);
    /*
     * リロード時間設定
     */
    delay_time = sil_rew_mem((uint32_t *)(TADR_TIMER_BASE
                                         +TOFF_TIMER_TIMERAWL)) + wrap_time;
    sil_wrw_mem((uint32_t *)(TADR_TIMER_BASE
                             +TOFF_TIMER_ALARM0+ALARMNO*4), (uint32_t)delay_time);
    if(timer_func != 0)
        timer_func();
}
```



カップラーメンタイマの作成

1. 仕様
2. プログラムの作成
 - 1) Step1
 - 2) [Step2](#)
 - 3) Step3
 - 4) Step4
 - 5) Step5
 - 6) Step6

Step2 :タイマによる基本時間(0.25秒)の作成

- グローバル変数
 - 1msec毎にインクリメントする変数(BaseTime)を用意
- タイマ割込みハンドラ
 - 1msec周期で実行され, BaseTimeをインクリメント
- メイン関数
 - BaseTimeを用いて250msec周期の処理を実現
 - 各ハードウェアの初期化と割込みの許可
 - 無限ループ内でBaseTimeをチェックする

Step2 : プログラム例 タイマ割込みハンドラ

- グローバル変数 BaseTime
 - unsigned long (32bit)とする

```
volatile unsigned long BaseTime;
```

- タイマ割込みハンドラ
 - ドライバで設定、コールバック関数で呼び出す

```
static void base_time_handler(void){  
    BaseTime++;  
}
```

コールバック関数から呼び出されるハンドラ

Step2 : プログラム例 メイン関数

- 各ハードウェアの初期化
 - メイン関数の先頭で実行

```
led_init();           /* LED初期化           */
switch_push_init(14); /* PUSHスイッチ初期化  */
timer_init();         /* タイマ初期化         */
uart_init();          /* UARTの初期化 */
switch_dip_init();    /* DIPSWの初期化 */
```

Step2 : プログラム例 メイン関数

- 250msec周期の処理

```
#define TIMER_GRANULE 250    /* 時間単位      */

void
_main(void){
    unsigned long timer250 = TIMER_GRANULE;
    ....
    BaseTime = 0;
    Event = 0;
    Sw1 = get_sw1_state();
    for (;;) {
        if (timer250 <= BaseTime) { /* 250m周期で実行 */
            /* 現在時刻の250msec後を設定 */
            timer250 += TIMER_GRANULE;
        }
    }
}
```

タイマ割込みハンドラで1ms周期でインクリメントされる

カップラーメンタイマの作成

1. 仕様
2. プログラムの作成
 - 1) Step1
 - 2) Step2
 - 3) [Step3](#)
 - 4) Step4
 - 5) Step5
 - 6) Step6

Step3 :メイン関数でのLED1の1秒間隔の点灯の実現

- 250msec周期を用いて1秒毎の処理を実現
 - 4回の実行で1秒
- 1秒毎の処理
 - LED1の状態を反転させる
- メイン関数内のローカル変数
 - LED1の点灯状態を保持する変数(led1)
 - 1秒を生成するための変数(sec)

Step3 : プログラム例 メイン関数

- 4回実行されるとLED1を反転させる

```
#define TO_SEC      4    /* 1秒間の呼び出し回数      */
void _main(void){
    unsigned char sec = 0;
    unsigned char led1 = OFF;
    ....
    if (timer250 <= BaseTime) {
        timer250 += TIMER_GRANULE;
        if (++sec == TO_SEC) {
            if (led1 == ON) {
                led1 = OFF;
            } else {
                led1 = ON;
            }
            sec = 0;
            set_led_state(LED1, led1);
        }
    }
}
```

4回実行

LED1反転

LED1設定

カップラーメンタイマの作成

1. 仕様
2. プログラムの作成
 - 1) Step1
 - 2) Step2
 - 3) Step3
 - 4) [Step4](#)
 - 5) Step5
 - 6) Step6

Step4 :スイッチプロセスの作成

- イベントの定義とビット割り当て
 - 開始 : EVT_TIMER_START 0x01
 - 停止 : EVT_TIMER_STOP 0x02
 - 時間アップ : EVT_TIMER_COUNT 0x04
- グローバル変数
 - スwitchの状態を保持するための変数(Sw1)
 - イベント通知用変数(Event)
- スイッチプロセス
 - Sw1,UART0の変化をチェックし, イベント通知用変数をセットする
 - UART0の状態に応じて, LED2を点灯・消灯させる

Step4 : プログラム例 イベント定義とグローバル変数

- イベント定義

```
#define EVT_TIMER_START    0x01
#define EVT_TIMER_STOP     0x02
#define EVT_TIMER_COUNT    0x04
```

- グローバル変数

- メイン関数で初期化

```
unsigned char Event;
unsigned char Sw1;
```

```
Event = 0;
Sw1 = get_sw1_state();    /* 現在のSW1の取り込み */
```

Step4 : プログラム例 SW1の状態取得

- UART0の受信データによりイベントを発生させる

```
void  
switch_process(void){  
    unsigned char sw;  
    sw = uart_pol_getc();  
    if (sw == 'S' || sw == 's') {  
        Event |= EVT_TIMER_START;  
        uart_pol_putc('S');  
    }  
    else if (sw == 'E' || sw == 'e') {  
        Event |= EVT_TIMER_STOP;  
        uart_pol_putc('E');  
    }  
    else if (sw == 'U' || sw == 'u') {  
        Event |= EVT_TIMER_COUNT;  
        uart_pol_putc('U');  
    }  
}
```

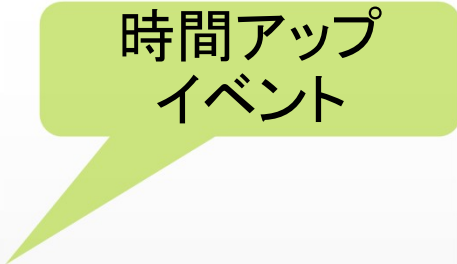
開始イベント

終了イベント

Step4 : プログラム例 UART0の受信状態取得

- SW1の状態をチェックし, 状態が変化していれば, イベント変数に対応するイベントをセットする

```
sw = get_sw1_state();
if (Sw1 != sw) {
    if (sw == ON) {
        uart_pol_putc('s');
        Event |= EVT_TIMER_START;
    }
    else {
        uart_pol_putc('e');
        Event |= EVT_TIMER_STOP;
    }
    Sw1 = sw;
}
```



カップラーメンタイマの作成

1. 仕様
2. プログラムの作成
 - 1) Step1
 - 2) Step2
 - 3) Step3
 - 4) Step4
 - 5) [Step5](#)
 - 6) Step6

Step5 : タイマプロセスの大枠とLED2点灯の実現

タイマプロセスの状態遷移判定と各状態の大枠作成
点滅用LED2(LED4と呼称)の点灯の実現

- タイマプロセスの状態を示す列挙型 (TIMER_MODE)
 - 計時終了 : STOP_TIMER_MODE
 - 計時中 : ACT_TIMER_MODE
 - タイムアウト : TIMEOUT_MODE
- タイマプロセスのローカル変数 (static変数)
 - タイムアウト時間を保持する変数 (max_time)
 - 現在の計時時間を保持する変数 (current_time)
 - カップラーメンタイマのモードを保持する変数 (timer_mode)
 - アラームのタイミング生成用変数 (alarm)
 - LED2の点滅状態を保持する変数 (led4)

Step5 : 列挙型とローカル変数

- タイマプロセスの状態を示す列挙型 (TIMER_MODE)

```
enum TIMER_MODE {  
    STOP_TIMER_MODE, /* 計時終了モード */  
    ACT_TIMER_MODE, /* 計時中モード */  
    TIMEOUT_MODE /* タイムアウトモード */  
};
```

- タイマプロセスのローカル変数 (static変数)
 - タイマプロセスは250msec周期で呼び出され、終了するため変数の値を保存するために static変数とする

```
static unsigned long max_time;  
static unsigned long current_time;  
static unsigned char timer_mode = STOP_TIMER_MODE;  
static unsigned char alarm = 0;  
static unsigned char led4 = OFF;
```

Step5 : 定数マクロの定義

- 各定数をマクロで定義する
 - タイムアウト時間の初期値
 - 時間アップ(延長)単位
 - タイムアウト時のLED4の点滅時間
 - カップラーメンタイマ動作通知のためのLED4点滅のインターバル
 - カップラーメンタイマ動作通知のためのLED4点滅の点滅時間

```
#define INIT_TIME          30  /* スタート時のタイムアウト時間 */
#define TIME_UNIT          30  /* 時間アップ単位 */
#define TIMEOUT_BLINK     15  /* タイムアウト時の点滅時間 */
#define ACT_INTERVAL      10  /* タイマ動作通知LEDのインターバル */
#define ACT_ON_TIME        1  /* タイマ動作通知LEDの点灯時間 */
```

Step5 : プログラム例 タイマプロセス状態遷移判定

- イベント通知用変数を参照してカップラーメンタイマのモード変数(状態)を設定する

```
/* switch_processより通知 */
if (Event != 0) {
    if (Event & EVT_TIMER_START) {    /* 開始 */
        timer_mode = ACT_TIMER_MODE;
    }
    if (Event & EVT_TIMER_STOP) {    /* 停止 */
        timer_mode = STOP_TIMER_MODE;
    }
    if (Event & EVT_TIMER_COUNT) {    /* 時間アップ */
        if (timer_mode == ACT_TIMER_MODE) {

        }
    }
    Event = 0;
}
```

Step5 : プログラム例 各状態毎の処理

- timer_modeによりモードを判定して実行

```
switch (timer_mode) {  
    case ACT_TIMER_MODE: /* 計時中モード */  
        if(current_time >= max_time){ /* タイムアウト */  
        }else if((current_time % (ACT_INTERVAL*1000)) == 0){  
        }  
        break;  
    case TIMEOUT_MODE: /* タイムアウトモード */  
        /* 15秒間点滅したらタイマー停止状態に */  
        if (current_time >= (TIMEOUT_BLINK * TO_SEC)){  
        }  
        break;  
    case STOP_TIMER_MODE: /* 計時終了モード */  
        break;  
    default:  
        break;  
}
```

Step5 : プログラム例 LED4の点灯

- LED4のタイミング生成用変数(alarm)が0以上なら点灯させる

```
if (alarm > 0) {          /* LED4点灯要求 : 250ms経過 */
    if(led4 == ON) {
        led4 = OFF;
    } else {
        led4 = ON;
    }
    alarm--;
} else {                  /* 点灯要求がなくLED4がオンなら消灯 */
    led4 = OFF;
}
if((led2 ^ led4) == 0)
    set_led_state(LED2, OFF); /* LED2消灯 */
else
    set_led_state(LED2, ON);  /* LED2点灯 */
```

カップラーメンタイマの作成

1. 仕様
2. プログラムの作成
 - 1) Step1
 - 2) Step2
 - 3) Step3
 - 4) Step4
 - 5) Step5
 - 6) Step6

Step6 : タイマプロセスの状態遷移判定と各状態の詳細

- 状態遷移判定での処理
 - 開始
 - current_time を0で初期化
 - max_time を設定し30秒でタイムアウトするよう設定
 - 停止
 - alarm を0にしてLED4の点灯を停止
 - 時間アップ
 - 計時中モードであれば max_time を30秒延長

Step6 : タイマプロセスの状態遷移判定と各状態の詳細

- 各状態(モード)での処理
 - 計時中モード
 - タイムアウトしていればモードをタイムアウトモードに移行させる
 - 10秒間隔のLED4の点灯が必要かチェックし, 必要ならLED4を点灯させるため alarm を設定する
 - タイムアウトモード
 - 15秒経過したらLED4の点灯を停止し, 計時終了モードに移行する
 - 計時終了モード
 - 特になし

Step6 : プログラム例 状態遷移判定の詳細

- 状態遷移と共に変数を設定する

```
if (Event & EVT_TIMER_START) {    /* 開始 */
    timer_mode = ACT_TIMER_MODE;
    current_time = 0;
    max_time = INIT_TIME*1000;
}
if (Event & EVT_TIMER_STOP) {    /* 停止 */
    timer_mode = STOP_TIMER_MODE;
    alarm = 0;
}
if (Event & EVT_TIMER_COUNT) {    /* 時間アップ */
    if (timer_mode == ACT_TIMER_MODE) {
        max_time += TIME_UNIT*1000;
    }
}
```

Step6 : プログラム例 各状態の詳細

- 計時中モード

```
case ACT_TIMER_MODE:
    if (current_time >= max_time) {
        /* タイムアウト */
        alarm = TIMEOUT_BLINK*TO_SEC;
        timer_mode = TIMEOUT_MODE;
        current_time = 0;
    } else if ((current_time % (ACT_INTERVAL*1000)) == 0) {
        /* 動作通知 */
        alarm = ACT_ON_TIME*TO_SEC;
    }
    current_time += TIMER_GRANULE; /* タイムアップ */
    break;
```

Step6 : プログラム例 各状態の詳細

- タイムアウトモード

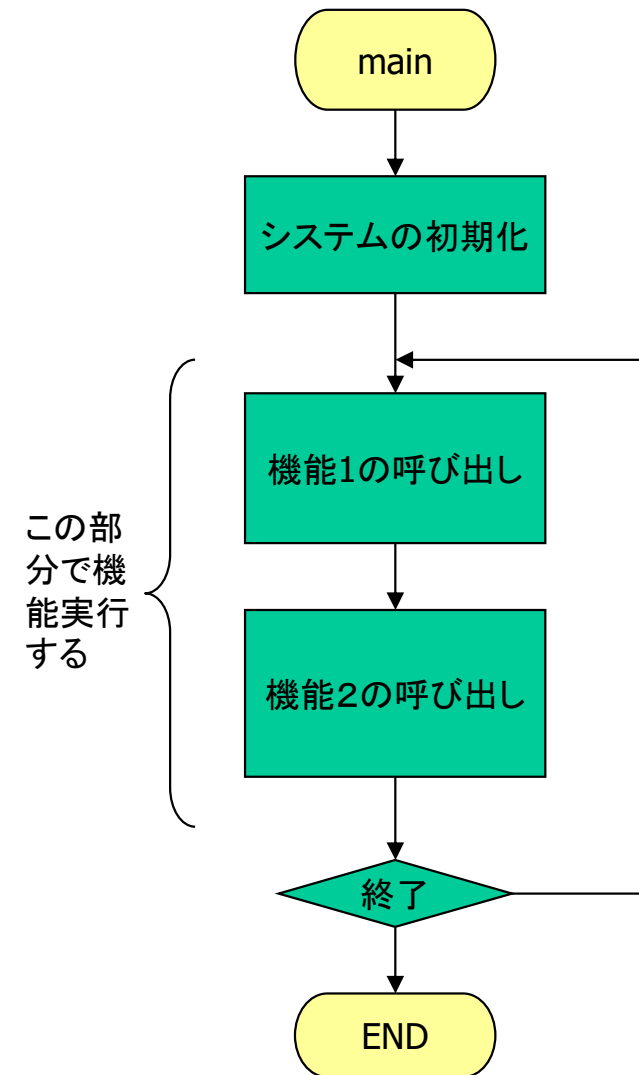
```
case TIMEOUT_MODE:
    /* 15秒間点滅したら計時終了モードに */
    if (current_time >= ((TIMEOUT_BLINK * TO_SEC)
                        * TIMER_GRANULE)) {
        timer_mode = STOP_TIMER_MODE;
    }
    current_time += TIMER_GRANULE; /* タイムアップ */
    break;
```

基礎2 補足

リアルタイムOSの基礎：補足資料

小規模組込み開発の開発手法

- 小規模組込み開発では、main関数中のループ部からシステムの制御に必要な関数を周期的に呼び出すことにより、制御を行うのが一般的な方法です
- 1, 2人の開発者でワンチップCPUに収まるソースコード一万行程度までのプログラムならば、十分な作りこみが可能です
- しかし、システムの規模が大きくなると種々の問題が発生します



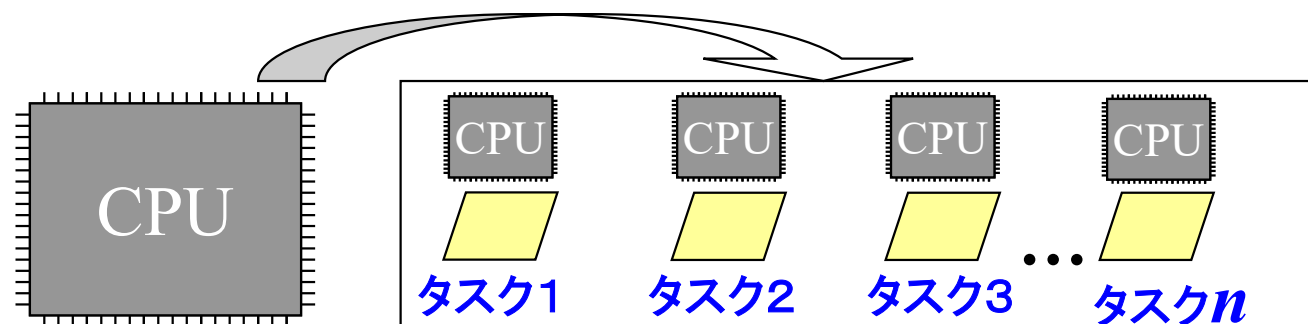
小規模組込み開発の限界

- 小規模システムでは小規模な機器管理が中心です。開発規模が大きくなると、複雑なUI部、通信機能、ファイルシステムの管理等種々のノウハウを持った開発者が分業してシステムを開発する必要があります
- 多くの機能を組み合わせた場合、main関数からの関数コールではハードリアルタイム性を保持するのが難しくなる
 - ある機能で100msの待ちが発生した場合、main関数からの関数呼び出しでは、その機能中でソフトウェアディレイで待つしかなく、不要なCPUの占有が発生する
 - ソフトウェアディレイ中に他の機能を動かそうとするとシステムが複雑化し、メンテナンス性が落ちる

➡ リアルタイムOSを用いれば問題解決可能

中規模組込みシステムとリアルタイムOSの必要性

- リアルタイムOSを用いることにより、個々の機能を独立したタスクとして動作させることが可能となります
 - リアルタイムOSの大きな特徴はマルチタスク機能
- マルチタスク機能とは
 - 複数のタスクを擬似並列に実行するための機能
 - 1CPUのシステムでは、実際には同時に実行できるのは1つのタスクのみです
 - あたかも複数のタスクを同時に実行しているようにみせる機能です



中規模組込み開発とシステム管理者

- 中規模組込みシステムではプラットフォーム部を先行開発しサブシステムを追加していく方式が一般的です、また、商品化後継続的に機能アップを行う必要があります
- システム管理者はプラットフォーム部とサブシステム間のアーキテクチャを決め、サブシステム間のインターフェイスを明確化する必要があります
- 分業開発では、予測外の問題で開発遅延が発生することが多々あります、これらの問題解決もシステム管理者の仕事です
 - 開発時、遅れたサブシステムをスタブ化してシステム開発の遅れを回避
 - 問題発生時、原因がどの部署に起因するかを解析

リアルタイムOSの内容を理解する技術者の必要性

- 中規模組込みシステムでは、リアルタイムOSに熟達した技術者を社内または社外の協力会社に用意する必要があります

システム開発中にアーキテクチャに関する問題解決が必要となるケースが多く発生する

- 中規模システムでは、汎用ミドルウェアの導入やデバイスドライバの導入が必要となります、この場合、リアルタイムOSに対して最適化を行う必要があります
- アプリケーションの障害でも、リアルタイムOS中でエクセプションが発生することがあります
 - ソースコードのないリアルタイムOSでは解析が困難
- 省エネ対応のためにリアルタイムOSの改造やSoC化によって何度も、検証用のデバイスドライバを書き換える必要があります

大規模組み込みシステムとの比較

- UNIXやWindowsの汎用OSを組み込みシステムとして用いる場合とリアルタイムOSを用いる場合の比較を行いましょう
- 商品は品質、機能、コストを最適にすることにより商品性が向上します
- 汎用OSはシステムの安全性を向上させる点からは有用ですが、OSの実行管理のために多くのCPUパワーやメモリが必要となり性能やコストの面で問題があります
- 汎用OS自体の工数が多いため、それを最適化し、管理するためそれなりの技術者も必要となる(リアルタイムOSより複雑でノウハウは高い)

簡単な例として、パソコンの周辺機の管理に、パソコンと同等のシステムを用いて行うのは無理があります

基礎2 補足

2日目 カップラーメンタイマの実装(リアルタイムOS版)

セミナーでは時間制約があり、実習できなかったコンテンツを紹介します。
本章は、テキスト2日目までの内容を全て学習後、実施して下さい。

カップラーメンタイマーの実装

1. 仕様
2. 設計
3. 実装

カップラーメンタイマの仕様

- 概要
 - タイマを起動してから、設定した時間が経過したことを知らせる
 - 設定時間はDIPSWの設定による
 - 00:30秒、01:1分、10:2分、11:3分
- スイッチとLEDの使い方
 - LED1 : 電源がONであること
(プログラムが動いていること)を表示
 - LED2 : 時間延長されたことを表示
: タイマが動いていることを表示
 - SW1または'S'押下 : タイマを起動
 - SW1または'E'押下 : タイマを停止
 - 'U'押下 : 設定時間を30秒延長

カップラーメンタイマの仕様

- 外部仕様

- 電源をONの間(プログラムが動いている間)、LED1の点灯・消灯を1秒間隔で繰り返す
- SW2がONになると設定時間を30秒にして、タイマを起動する
- SW2がOFFになると、タイマを停止する
- タイマが動いている間に'U'コマンドにて、ONの間LED2を点灯させ、設定時間を30秒延長する
- タイマが動いている間は、10秒間隔で、LED4を2回点滅させる。LED2の2回点滅は、点灯・消灯を0.25秒間隔で2回繰り返すことで行う
- 設定時間が経過すると、LED2を15秒間点滅させた後、消灯する。LED2の点滅は、点灯・消灯を0.25秒間隔で繰り返すことで行う



カップラーメンタイマの仕様

- デバッグ仕様
 - PIPE TIMERコマンドを追加し、タイマーが実行中は設定時間の残り時間を表示する

カップラーメンタイマーの実装

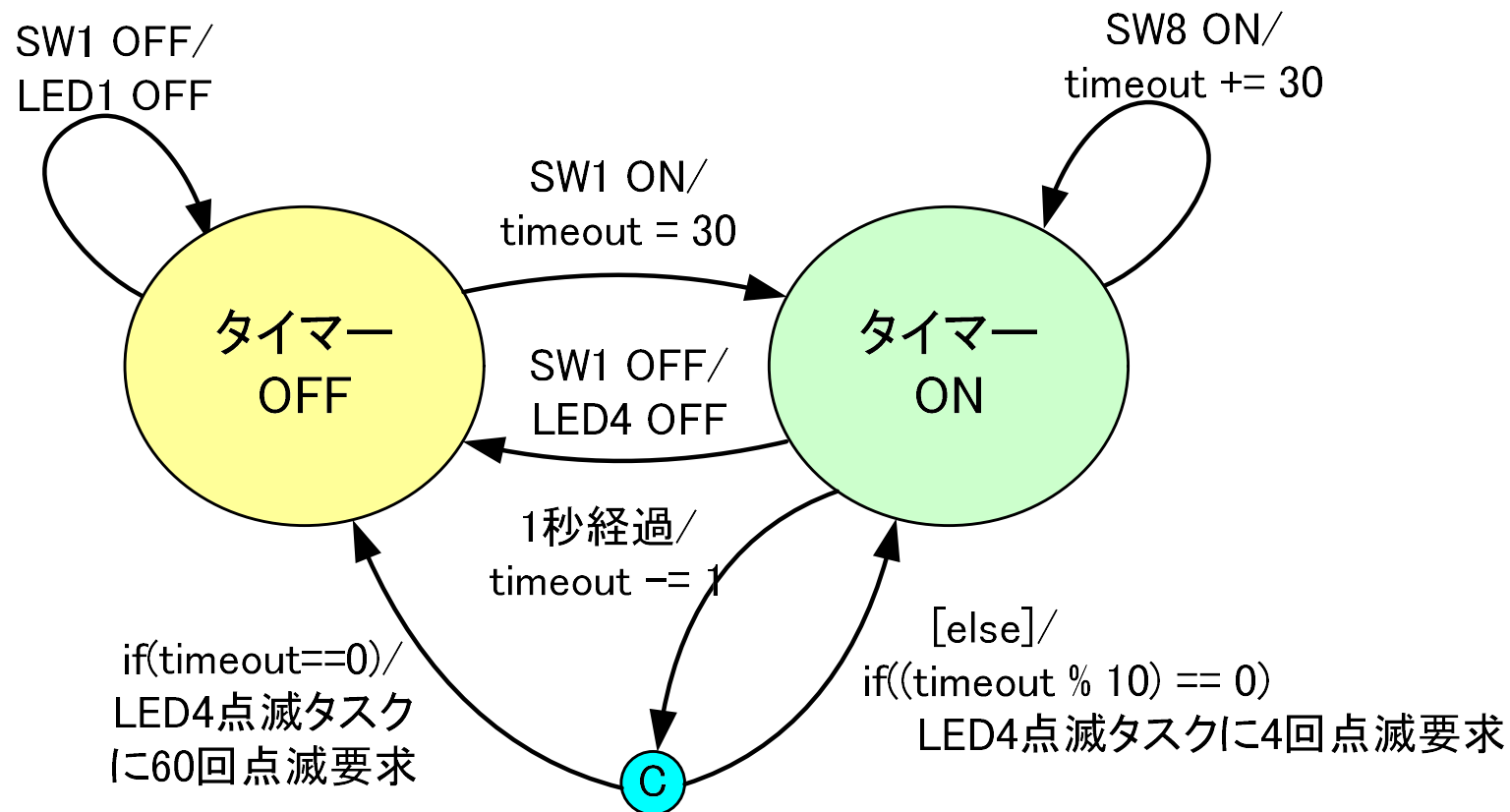
1. 仕様
2. 設計
3. 実装

設計：基本設計方針

- タスク
 - タイマーを管理するタイマータスクと, LED2の点滅を行うLED2点滅タスクの2つ
- 周期ハンドラ
 - スイッチの状態取得は周期ハンドラで行う
 - 各種時間生成も周期ハンドラで行う
- イベント通知
 - 周期ハンドラ-タスク間、タスク間のイベント通知はイベントフラグにより行う

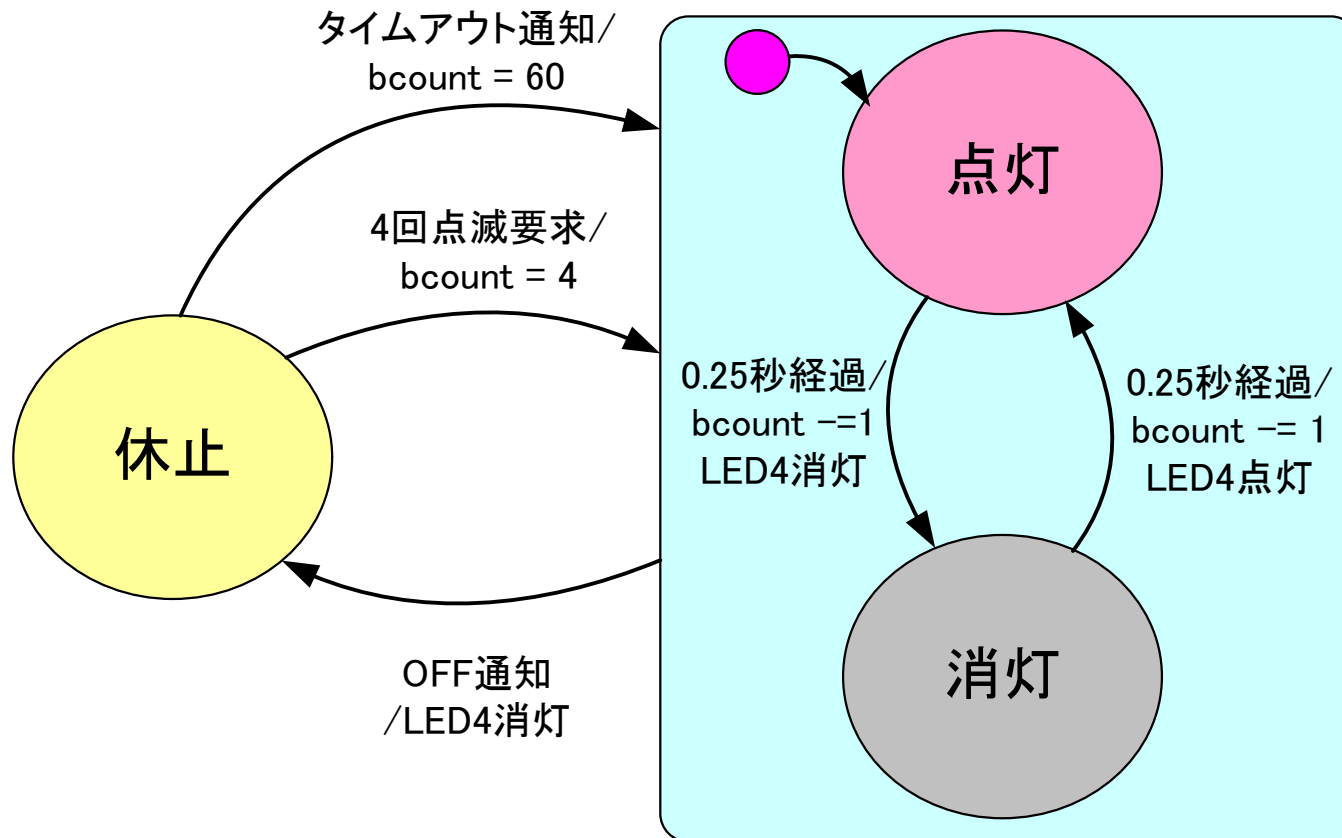
設計：タイマータスク

- イベント
 - SW1 ON, SW1 OFF, SW8がON, 1秒経過
- 状態遷移図



設計 : LED4点滅タスク

- イベント
 - 4回点滅要求, タイムアウト通知, OFF通知, 0.25秒経過
- 状態遷移図



設計：周期ハンドラと初期化ルーチン

- 周期ハンドラ
 - LED1点滅ハンドラ : 1秒周期
 - LED4点滅時間生成ハンドラ : 250m周期
 - スイッチ状態取得ハンドラ : 10m周期
 - 基本時間生成ハンドラ : 1秒周期
- 初期化ルーチン
 - ハードウェアを初期化
 - スイッチの初期状態を初期化
 - ブザーの初期状態を初期化

カップラーメンタイマーの実装

1. 仕様
2. 設計
3. 実装

実装 : マクロ1

- 実装済みのコード
 - program2/cup_timer
- タイマータスクイベント割付

#define TIMER_START	0x01	/* START 通知 */
#define TIMER_STOP	0x02	/* STOP 通知 */
#define TIMER_UPDATE	0x04	/* UPDATE 通知 */
#define TIMER_BASE_TIME	0x08	/* 1秒経過 通知 */
#define TIMER_ALL	0x0f	/* イベントビットのOR */

- LED4点滅タスクイベント割付

#define BLINK_ACTIVE	0x01	/* 4回点滅要求 通知 */
#define BLINK_TIMEOUT	0x02	/* タイムアウト 通知 */
#define BLINK_BLINK	0x04	/* 0.25秒経過 通知 */
#define BLINK_OFF	0x08	/* OFF 通知 */
#define BLINK_ALL	0x0f	/* イベントビットのOR */

実装：マクロ2 ヘッダーファイル

- 各時間間隔の定義

```
#define LED1_BLINK_INTERVAL 1000 /* LED1の点滅間隔 */  
#define LED2_BLINK_INTERVAL 250 /* LED2の点滅間隔 */  
#define SW_SCAN_INTERVAL 10 /* SWのスキャン間隔 */  
#define BASE_TIME_INTERVAL 1000 /* 基本時間間隔 */
```

- その他の定義

```
#define INIT_TIME 30 /* スタート時のタイムアウト時間 */  
#define EXTRA_UNIT 30 /* 時間延長単位 */  
#define TIMEOUT_BLINK 60 /* タイムアウト時の点滅回数 */  
#define ACT_INTERVAL 10 /* タイマ動作通知LEDのインターバル */  
#define ACT_BLINK_TIME 4 /* タイマ動作通知LEDの点滅回数 */
```

実装 : タイマータスク

- タイマーOFF状態
 - TIMER_START, TIMER_STOP イベント受け取る

```
W timer;
B def_led;
void
timer_task(intptr_t exinf){
    FLGPTN flgptn;
    UW dip_value;
    for (;;) {
        timer = 0;
        do {
            set_led_state(LED02, OFF);
            def_led = OFF;
            wai_flg(FLG_TIMER, TIMER_ALL, TWF_ORW, &flgptn);
            if ((flgptn & TIMER_START) != 0) {
                dip_value = switch_dip_sense() & 3;
                if(dip_value == 0)
                    timer = INIT_TIME;
                else
                    timer = INIT_TIME = dip_value * 2;
            }
            if ((flgptn & TIMER_STOP) != 0) {
                set_flg(FLG_BLINK, BLINK_OFF);
            }
        } while (timer == 0);
    }
}
```

実装：タイマーON状態

- TIMER_STOPイベントを受け取ると、ブリンクタスクにブリンクOFFを通知
- TIMER_UPDATE で時間延長
- 1秒経過イベントで時間をデクリメントし、ブリンクタスクにタイムアウトで60回点滅イベントを送り、10秒で4回点滅イベントを送る

```
sta_cyc(BASE_TIME_HANDLER);
while (timer > 0) {
    set_led_state(LED02, ON);
    def_led = ON;
    wai_flg(FLG_TIMER, TIMER_ALL, TWF_ORW, &flgpfn);
    if ((flgpfn & TIMER_STOP) != 0) {
        set_flg(FLG_BLINK, BLINK_OFF);
        timer = 0;
    } else {
        if ((flgpfn & TIMER_UPDATE) != 0)
            timer = timer + EXTRA_UNIT;
        if ((flgpfn & TIMER_BASE_TIME) != 0) {
            timer = timer - 1;
            if (timer == 0)
                set_flg(FLG_BLINK, BLINK_TIMEOUT);
            else if ((timer % ACT_INTERVAL) == 0)
                set_flg(FLG_BLINK, BLINK_ACTIVE);
        }
    }
}
stp_cyc(BASE_TIME_HANDLER);
}
```

実装：LED4点滅タスク

- タイマータスクからのイベントを待つ
- 60回点滅イベントもしくは、10秒で4回点滅イベントを受け取ると bcount をそれぞれ 60, 4 に設定してループを抜ける

```
void blink_task(intptr_t exinf) {
    W bcount;
    FLGPTN flgptn;
    UB sta_led = OFF;
    for (;;) {
        bcount = 0;
        do {
            wai_flg(FLG_BLINK, BLINK_ALL, TWF_ORW, &flgptn);
            if ((flgptn & BLINK_ACTIVE) != 0) {
                bcount = ACT_BLINK_TIME;
            }
            if ((flgptn & BLINK_TIMEOUT) != 0) {
                bcount = TIMEOUT_BLINK;
            }
        } while (bcount == 0);
    }
}
```

実装：ブリンクタスク

- 点滅時間生成ハンドラを実行し, 送られてくるイベントに合わせてLEDの点滅を bcount 回繰り返す

```
sta_cyc(LED4_BLINK_TIME_HANDLER);
while (bcount > 0) {
    wai_flg(FLG_BLINK, BLINK_ALL, TWF_ORW, &flgptn);
    if ((flgptn & BLINK_BLINK) != 0) {
        if(sta_led == ON)
            sta_led = OFF;
        else
            sta_led = ON;
        if((sta_led ^ def_led) == 0)
            set_led_state(LED02, OFF);
        else
            set_led_state(LED02, ON);
        bcount--;
    }
    if ((flgptn & BLINK_OFF) != 0) {
        bcount = 0;
    }
}
stp_cyc(LED4_BLINK_TIME_HANDLER);
sta_led = OFF;
if(set_led ^ def_led) == 0)
    set_led_state(LED02, OFF);
else
    set_led_state(LED02, ON);
}
```

実装：周期ハンドラ

- 基本時間生成ハンドラ
 - 1秒周期で1秒経過イベントを通知

```
void  
base_time_handler(void){  
    SVC_ERROR(iset_flg(FLG_TIMER, TIMER_BASE_TIME));  
}
```

- LED2点滅時間生成ハンドラ
 - 250m周期で0.25秒経過イベントを通知

```
void  
led2_blink_time_handler(void){  
    SVC_ERROR(iset_flg(FLG_BLINK, BLINK_BLINK));  
}
```

実装：スイッチ状態取得ハンドラ

- 実装：コマンド確認取得ハンドラ

```
void
sw_scan_handler(void){
    unsigned char sw;
    FLGPTN flgptn = 0;

    set_led_state(LED02, OFF);
    sw = get_cup_command();
    if (sw == 'S' || sw == 's') {
        flgptn |= TIMER_START;
    }
    else if (sw == 'E' || sw == 'e') {
        flgptn |= TIMER_STOP;
    }
    else if (sw == 'U' || sw == 'u'){
        flgptn |= TIMER_UPDATE;
    }
    if (flgptn != 0x00)
        SVC_PERROR (iset_flg(FLG_TIMER, flgptn));
}
```

実装: スイッチ状態取得ハンドラ

- スイッチ状態取得ハンドラ
 - SW割込みから呼び出す割込みハンドラ(コールバック関数)

```
void
sw_handle(void)
{
    if(timer > 0) {
        SVC_PERROR(iset_flg(FLG_TIMER, TIMER_STOP));
    }
    else {
        SVC_PERROR(iset_flg(FLG_TIMER, TIMER_START));
    }
}
```

実装：LED1点滅ハンドラ

- 1秒周期でLED1を点滅

```
void  
led1_blink_handler(void)  
{  
    static UB sta_led = OFF;  
    if (sta_led == ON) {  
        sta_led = OFF;  
    } else {  
        sta_led = ON;  
    }  
    set_led1_state(sta_led);  
}
```

実装：コンフィギュレーションファイル

- デバイスの初期化

```
#if BOARDNO == 0  
INCLUDE("pdic/rp2040/device.cfg");  
#else  
INCLUDE("pdic/rp2350/device.cfg");  
#endif
```

- タスク

```
CRE_TSK(MAIN_TASK, { TA_ACT, 0, timer_task,  
                     DEFAULT_PRIORITY, STACK_SIZE, NULL });  
CRE_TSK(BLINK_TASK, { TA_ACT, 0, blink_task,  
                     DEFAULT_PRIORITY, STACK_SIZE, NULL });
```

実装：コンフィギュレーションファイル

- 周期ハンドラ

```
CRE_CYC(LED1_BLINK_HANDLER, {TA_STA, 0,  
    led1_blink_handler, LED1_BLINK_INTERVAL, 0});  
CRE_CYC(LED2_BLINK_TIME_HANDLER, {TA_NULL, 0,  
    led2_blink_time_handler, LED4_BLINK_INTERVAL, 0});  
CRE_CYC(SW_SCAN_HANDLER, {TA_STA, 0,  
    sw_scan_handler, SW_SCAN_INTERVAL, 0});  
CRE_CYC(BASE_TIME_HANDLER, {TA_NULL, 0,  
    base_time_handler, BASE_TIME_INTERVAL, 0});
```

- イベントフラグ

```
CRE_FLG(FLG_TIMER, {(TA_TFIFO|TA_CLR), 0});  
CRE_FLG(FLG_BLINK, {(TA_TFIFO|TA_CLR), 0});
```

実装：コンフィギュレーションファイル

- 初期化関数
 - シールドのHW初期化：sheild_init
 - SW割込み用コールバック関数設定：setup_sw2_func
 - デバイス用のタスクモニタコマンド設定：device_info_init

```
ATT_INI({ TA_NULL, 0, sheild_init });  
ATT_INI({ TA_NULL, sw_handle, setup_sw2_func });  
ATT_INI({ TA_NULL, 0, device_info_init });
```

実装: コマンドの取得

- PIPEコマンドを拡張してCUP拡張コマンドの後のキャラクタをget_cup_command関数で取り出すプログラムを拡張する
 - PIPE CUP S(return) スタートコマンド
 - PIPE CUP E(return) ストップコマンド
 - PIPE CUP U(return) アップデートコマンド