

TOPPERS中級実装セミナー

(リアルタイムシステム構築偏:H8/3069F版)

3日目

TOPPERSプロジェクト
教育ワーキング・グループ

2005/12/17

TOPPERSプロジェクト認定

1



本ドキュメントに関して

■ 本ドキュメントの利用に関して以下の点にご留意ください

1. 著作権に関しての表記

＜TOPPERS中級実装セミナー（リアルタイムシステム構築編：H8-3069F版）3日目＞

Copyright (C) 2004-2005 by 竹内良輔 (株)リコー GJ事業部

Copyright (C) 2004-2005 by 森本亮太 (株)リコー MFP事業本部

Copyright (C) 2004-2005 by 中嶋 哲 (株)東芝情報システム株式会社

上記著作権者は、以下の (1)～(3) の条件を満たす場合に限り、本資料（本資料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) 本資料を利用する場合には、上記の著作権表示およびこの利用条件が、そのままの形で資料中に含まれていること。
- (2) 本資料を改変する場合には、資料を改変した旨の記述を、資料中に含めること。ただし、TOPPERSプロジェクトの活動の一環として本資料を改変する場合には、資料を改変した旨の記述を含める必要はない。
- (3) 本資料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。

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。 μ ITRONは”Micro Industrial TRON”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。

2005/12/17

TOPPERSプロジェクト認定

2



スケジュール

■ 3日目

1. RTOSを用いたリアルタイムシステム構築 1. 5時間
2. 実習課題の説明 0. 5時間
3. 開発環境の確認 0. 25時間
4. 話題沸騰ポット作成1 3. 25時間
 - 4-1. 作成のヒント1 (0. 5時間)
 - 4-2. 作成のヒント2 (0. 5時間)

■ 4日目

1. 話題沸騰ポット作成2 1. 5時間
2. レビュー 0. 5時間
 - 設計、実装の評価
3. リアルタイム性向上の手法 1. 0時間
4. タスク負荷の検証と対応 0. 5時間
5. 話題沸騰ポットの割込みの対応 2. 5時間
 - まとめ 0. 5時間

セミナー3, 4日の目標

- リアルタイムシステムの設計技術の習得。
- 「話題沸騰ポット」試作機のソフトウェア開発を通してリアルタイムシステム実装技術の習得を行う。
- リアルタイムスケジューリング理論の学習、割り込み制御を用い、リアルタイム性向上のための実装技術の習得を行う。

RTOSを用いたリアルタイム システム構築

1. RTOSを用いたリアルタイムシステム構築
2. 実習課題の説明
3. 開発環境の確認
4. 話題沸騰ポット作成1

RTOSを用いたリアルタイムシステム構築に関して説明を行います。

組込みシステムの復習1

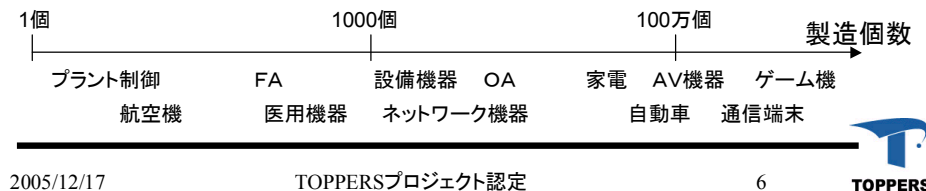
システムの多様性

多様であり、明確な技術教育がなされていない

- 組込みシステムは、システム規模・性質いずれの面からも極めて多様である
- 現状では、有効な分類指標が提案されていない
- 組込みシステム開発手法を左右する式

$$\text{トータルコスト} = \text{開発コスト} + \text{製造コスト} \times \text{製造個数}$$

さらにtime-to-marketの要因を加味する必要がある。
近年は大量生産においても開発コストは無視できない



組込みソフトウェア開発の特徴として以下のことが上げられます。

- ハードウェアに密着したプログラミング

システム毎にハードウェア構成や周辺デバイスが異なる為、ハードウェアを直接扱うプログラミングが必要となる。ノウハウの負う部分の開発が多く開発が難しいといえます。

- 開発環境とターゲット環境の分離

ターゲットシステム上で開発が出来るとは限りません。クロス開発環境やリモートデバック、シミュレーションによる開発等を考慮する必要があります。

また、システムを停止させずに検証やデバックが必要となるシステムもあります。

- ハードウェアとの協調設計・並行開発(の可能性)

ターゲットシステムが出来る前の検証やデバックも可能となりつつあります。

- 多様なプラットフォーム(ハードウェア、OS)

システムがハードウェアやOSに最適化されるケースが多くあります。

- 検証に掛かるコストが大きい

高い信頼性、タイミングに起因する非決定性、制御対象機器を動作させる必要があるため検証にコストが掛かります。

- 例外処理指向のソフトウェア設計が必要な場合もあります

- システム内のソフトウェアは信頼できるという前提が成り立ちます。

組込みシステムの復習2

組込みシステムの特性

4つの柱

- 専用化されたシステムである
システム全体がひとつの目的に専用化して設計される
- 厳しいリソース制約がある
コストダウン、低消費電力、軽量化など
- 高い信頼性を求められる
システムの誤動作が機器の誤動作に直結
システムの改修には高いコストがかかる
- リアルタイム性を必要とする
単に早ければよいわけではなく、制御対象の機器が定める時間要件に従って動作することが必要

➡ 組込み機器のほとんどがリアルタイムシステム

2005/12/17

TOPPERSプロジェクト認定

7



組込みシステムの多様性に伴い、いくつかの特性があります。

・専用化されたシステム

システム全体が1つの目的に専用化して設計されています。

・厳しいリソース制約

大量生産の場合、顕著ですが、厳しいコストダウンが要求されます。

低消費電力、動作環境(温度、実行環境等)、軽量化等特別な制約があります。

・高い信頼性

システムの誤動作が機器の誤動作に直結する 경우가多く、まったく保証しないというわけにはいきません。PL法の対象となる場合もあります。

システムの改修に高いコストが掛かる場合があります。

機器の信頼性に対するユーザの期待があります。

・リアルタイム性

制御対象の機器の定める時間要件に従って動作することが求められます。

これは、単に速ければ良いということではありません。

これを実現できるシステムがリアルタイムシステムです。

組込みソフトウェア開発の特性1

■ ハードウェアに密着したプログラミング

- ▼システム毎にハードウェア構成や周辺デバイスが異なるため、ハードウェアを直接扱うプログラミングが必要
- ▼ノウハウに負う部分が大きく、開発が難しい

■ 開発環境とターゲット環境の分離

- ▼ターゲットシステム上で開発できるとは限らない
→クロス開発環境やリモートデバッグが必要
- ▼システムを停止させずに検証/デバッグすることが必要なシステムもある

■ ハードウェアとの協調設計・平行開発(の可能性)

- ▼ターゲットシステムができる前の検証/デバッグ

■ 多様なプラットフォーム(ハードウェア、OS)

- ▼ハードウェアやOSがシステムに最適化されるため

2005/12/17

TOPPERSプロジェクト認定

8



組込みソフトウェア開発の特徴として以下の点が上げられます。

1) ハードウェアに密着したプログラミングです

システム毎に、ハードウェア構成や周辺デバイスが異なる、これらのハードウェアを直接制御するプログラムを開発する必要があります。プログラミング開発力ばかりではなく、ハードウェアの制御技術や、機器に対する特殊なノウハウも、プログラムの開発に必要となります。これらは、経験やノウハウに負う部分が大きく、開発が困難な場合が多くあります。

2) 開発環境とターゲットが環境が分離しています

多くの場合、システムの実行環境と開発環境が別です。プロダクトの運営のために、事前にクロス開発環境の準備、ビルドした実行形式をどのようにしてターゲット環境で動作、デバッグするかの準備を行っておかなければなりません。また、検証やデバッグに際しては、制御対象機器が非同期な動作を行ったり、マルチタスク環境では、タスクをいちいち停止させてデバッグしたのでは、制御対象機器が正常に動作しない等々、システムを停止させずに検証やデバッグを行うシステムもあります。

3) ハードウェアとの協調設計・平行開発を行う場合もあります

ターゲットシステムが完成する前に、組込みソフトウェアを開発しなければならないケースもあります。このような場合、シミュレーション環境で開発をおこなったり、ワークステーション上でハードウェアを擬似的に実行させ、ソフトウェア開発を行うケースもあります。ハードウェアの協調設計システムは、高価であり一般的には普及していません。

4) 多様なプラットフォーム

組込みシステムが多様であるため、個々のシステムに対応した多様なプラットフォーム(ここではハードウェアやOSをさす)での開発が必要となります。

組込みソフトウェア開発の特性2

- 検証にかかるコストが大きい
 - ▼高い信頼性が求められる
 - ▼タイミングに起因する非決定性
 - ▼制御対象機器を動作させる必要
 - 制御対象機器や環境を含めたシミュレーション
- 例外処理指向のソフトウェア設計が必要な場合も
 - ▼通常動作以外の処理が占める割合が高い
 - システム故障時の例外処理、診断処理
 - ▼例外処理が時間制約を守る上でネックになることも
 - 例外処理を最初から考慮したシステムが必要
- システム内のソフトウェアは信頼できるという前提
 - ▼デバッグが終われば、システム内の他のソフトウェアは信頼できるという前提が成り立つ

2005/12/17

TOPPERSプロジェクト認定

9



5) 検証にかかるコストが大きい

以下の要因によります。

- ・組込みシステムに高い信頼性が求められています
- ・制御対象機器やその実行環境に起因して種々のタイミングのイベントの発生を考慮しなければならず、非決定性の要因が多くあります
- ・制御対象機器を動作させないと検証ができない

6) 例外処理指向のソフトウェア設計が必要な場合もあります

一般的に、組込みシステムでは、通常動作以外の処理が占める割合が高くなります。例えば、システム故障時の例外処理、診断処理、それらに伴う特殊な設定やユーザ(サービスマン)インターフェイスなど。診断処理中は、異常動作時の動作判定のためにログをとる機能や動作を監視する処理等、システム負荷となるプログラムもあります。これらの要因を考慮して、システム設計を行う必要もあります。

7) システム内のソフトウェアは信頼できるという前提

システムのデバッグや検証が終われば、システム内の他のソフトウェアは信頼できるという前提が成り立ちます。

組込みソフトウェアの開発環境

2005/12/17

TOPPERSプロジェクト認定

10



組込みソフトウェアの開発環境について説明を行います。

組込みソフトウェアの開発環境

組込みソフトウェア開発に独特なツールについて紹介

■ クロス開発環境と開発ツール

▼ターゲットシステムと開発用のシステム(ホストシステム)が分離しているソフトウェア開発環境をクロス開発環境と呼ぶ

▼クロス開発環境のための開発ツール

- ・クロスコンパイラ
- ・クロスアセンブラ
- ・クロスリンカ
- ・リモートデバッガ・・・いろいろな方式

■ テスト／検証環境

- ・実機を用いないテスト／検証環境

2005/12/17

TOPPERSプロジェクト認定

11



・クロス開発環境

ターゲットシステムと開発用のシステム(ホストシステム)が分離しているソフトウェア開発環境をクロス開発環境と呼びます。

TOPPERS/JSPの開発環境はクロス開発環境です。GNUの場合以下の対応となります。

- | | |
|-----------|--|
| ・クロスコンパイラ | --- XXX-XXX-gcc XXX-XXX-g++
gccはC言語のコンパイラ、g++はC++言語のコンパイラです
各々言語から中間オブジェクト(*.o)を生成します。 |
| ・クロスアセンブラ | --- XXX-XXX-as
各CPUに対応したアセンブラ言語から中間オブジェクトを
生成します。 |
| ・クロスリンカ | --- XXX-XXX-ld
中間オブジェクトを実際のROM、RAMに再配置します。 |
| ・リモートデバッガ | --- GDB |

実機を用いない検証環境としては、TOPPERS/JSPのWindows版のデバイスシミュレータ、TORNADEのWindowsシミュレータ等があります。これらは実機を使わず組込みソフトウェアの検証が可能です。

組込みソフトウェア向けの開発ツールの特徴1 コンパイラの最適化

■ コードサイズ縮小のための最適化

▼コンパイラの最適化技法の中には、高速化とコードサイズ縮小が同時に達成できるものと、両者が矛盾するものがある

前者の例: 共通部分式除去

後者の例: ループ展開

▼コンパイラオプションで、高速化を重視するか、コードサイズ縮小を重視するか指定できるコンパイラもある

■ 消費電力削減のための最適化

▼研究テーマの段階

▼高速化のための最適化とおおよそ一致

2005/12/17

TOPPERSプロジェクト認定

12



組込みに使用され始めた当時のC言語は、アセンブラ言語に比べて、コンパイル後の実行サイズや実行スピードに問題がありました。その後、コンパイラは改善を重ねられ、最適化技法により、アセンブラ言語に近い実行サイズや実行スピードでコンパイルができるようになりました。最適化には高速化とコードサイズの縮小があります。この機能は同時に達成できるものと、両方の機能が矛盾するものがあります。

両方が達成できる例としては、

```
{
    int i, j;
    i = 10;
    j = 20;
    i = 30;
    ...
}
```

の記述があった場合、最初の「i=10;」は意味を持たないため、最適化の際、削除してコード生成します。この場合、無駄な実行コードが削除されるため、高速化とコードサイズ縮小が同時に達成できます。

矛盾する例としては

```
{
    int i;
    int exp[100];
    for(i = 0 ; i < 100 ; i++)
        exp[i] = 0;
}
```

の記述があった場合、exp[0] = 0 ; exp[1] = 0; ...exp[99]=0;のように最適化すれば、iを管理する必要がないため、高速化はできますが、コードサイズは大きくなります。

組込みソフトウェア向けの開発ツールの特徴2 最適化の落とし穴

■ 最適化によりメモリアクセスパターンが変わる可能性

→ volatile宣言(標準のC言語の機能)

■ 最適化レベルを上げるとデバッグが難しくなる

▼最適化により命令の順序が入れ替わるため、ソースプログラムとの対応をつけることが難しくなる

▼関数の途中で、引数の値が失われる場合も(デバッガが扱いに困る)

! ソフトウェアのテスト・デバッグ中は最適化レベルを下げておき、テストが終わると最適化レベルを上げる方法は、受け入れられない場合が多い

▼最適化レベルで時間的な振舞いは変わる

▼コンパイラの最適化にバグがあるかもしれない



2005/12/17

TOPPERSプロジェクト認定

13

TOPPERS

最適化にはいくつかの落とし穴がある。先ほどの最適化の例で

```
{
    unsigned char *port5 = (unsigned char *)H8_P5DR;
    *port5 = 0x02;
    soft_delay(100);
    *port5 = 0x03;
    ....
}
```

のような記述があった場合、ポート5に対して、0x02を書き込み、一定時間後に0x03を書き込みたいとしても、最適化により最初の0x02に書き込みが削除されてしまつては正常にハードウェアを動作させることはできません、この場合、volatile宣言を用いれば、この問題を回避できます。

最適化レベルを上げるとデバッグは難しくなります。最適化により、コンパイル時生成されるアセンブラコードの順序の入れ替えが行われ、ソースプログラムとの対応が難しくなります。また、関数の途中で参照されない引数の値が失われる場合もあり、引数値の表示機能をもつデバッガの機能を達成できなくなります。

テスト・デバック中は最適化レベルを下げておき(デバッグモードでコードを生成)、テストが終わると最適化をあげるという対応は受け入れられない場合が多い。それは以下の要因によります。

1) 最適化レベルで、プログラムの実行時間が変わると制御機器に対する実行タイミングが変わり、新たな障害となる場合があります。

2) コンパイラの最適化にバグがある、または、バグを誘発する可能性があります。

基礎知識: volatile宣言

- volatile宣言は値が不明が型で変更される可能性をもつことを示す (volatile=揮発性の)

```
char *device_register = (char *)0xfffe0020;
while ((*device_register & 0x01) == 0);
```



無限ループに落ちる！

```
volatile char *device_register = (volatile char *)0xfffe0020;
while ((*device_register & 0x01) == 0);
```

! volatile宣言は最適化を制限する

注意) ポインタのvolatile宣言

volatile char *device_register

→ device_registerの指す先がvolatile

char *volatile device_register

→ device_register自身がvolatile

2005/12/17

TOPPERSプロジェクト認定

14



• volatile宣言

宣言された値が不明の型で変更される可能性をもつことを示します。volatileは「不揮発性の」という意味です。device_registerが入力用のポートを指し、内容が0x01に変化するまで、待ちを入れたい場合、volatile宣言のないdevice_registerの記述では、最初に参照した値から変化しないと判定して、コンパイラの最適化によって、一度だけしか参照しないコードを生成する可能性があります。そのため、最初のアクセスのゼロビットのみを参照し、その値がゼロの場合、以降の参照はせず、永久待ちに入る場合があります。このような場合は、volatile宣言を行っておけば、コンパイラは最適化の対象からはずし、while文の参照を正しく行うため、device_registerのビット0の値が1に変化した時点で次にステップの移行します。

ポインタに対してvolatile宣言を行う場合、上記の例のように

- 1) char *の前に入れた場合は、ポインタが参照する先が不明の形で変更されることを示します。
- 2) char *の後に記述した場合は、ポインタ自体が不明の形で変更されることを示します。

組込みソフトウェア向け開発ツールの特徴3

■ インラインアセンブラ機能

- ▼C言語のソースファイルの中で、直接アセンブラの命令を記述するための機能
- ▼特殊なレジスタや命令を使う場合に必要

■ メモリアロケーションの指定

- ▼プログラム領域やデータ領域を、どのメモリ番地に置くかを細かく指定する機能が必要

← 性質の違うメモリを使いたい

例) 高速のSRAM, DRAM

バッテリーバックアップされたRAM

- ▼コンパイラ(またはリンカ)で、変数を置くセクション(セグメント)を指定する

- ▼リンカに対して、各セクションのリンク方法やその先頭番地を指定する

2005/12/17

TOPPERSプロジェクト認定

15



・インラインアセンブラ

インラインアセンブラは、C言語の記述の中に、直接アセンブラの記述を行う機能です。以下はH8のインラインアセンブラの例でconfig/h8/cpu_insn.h中の記述を抜き出したものです。Asm();の中にアセンブルコードを記述します。このような記述を行うとC言語のソースがCPU依存になりますので、気をつけましょう。

/*

* コンデションコードレジスタ(CCR)の現在値の読出し

*/

Inline UB

current_ccr(void)

{

 UB ccr;

 Asm("stc.b ccr,%0l" : "=r"(ccr));

 return(ccr);

}

・メモリアロケーション

組込みプログラムをROMやRAM上に正しく実行させるためには、プログラムの領域やデータ領域を細かく指定する機能が必要です。このセミナーで使用しているGNUの環境ではセクションを指定し、そのアドレスをリンカ(ld)によって再配置することにより、この機能を実現しています。

config/h8/akih8_3069f/debug.ldがデバック用に再配置するための指定ファイルとなります。本セミナーの実行環境はすべてRAM上なので、debug.ldは全てのアドレスをRAM上に設定しています。

組込みソフトウェア向け開発ツールの特徴4

■ ROM化可能コードの生成

▼プログラムをROMに置いて動作させるための機能

▼プログラムコードと定数データはROMに置く

定数データ: C言語const宣言された変数

文字列定数(””で囲んだ文字列)

注意) ポインタのconst宣言にはvolatile宣言と同じ注意

▼初期化されるデータは、ROMに置いて、プログラム実行前にRAMにコピーして使う

! データを配慮すべきアドレスと、実行時に参照すべきアドレスが異なる

→ スタートアップモジュールの役割

2005/12/17

TOPPERSプロジェクト認定

16



・ROM化のコード指定

実際の商品化を行うには、プログラムや定数データをROMに配置しなければなりません。そうしないと、毎回ダウンローダを用いてプログラム等をダウンロードする必要があります。ROMに配置し、RAMへの書き込み治具を用いて書き込みを行うことにより、ダウンロードせずに実行可能になります。しかし、以下の記述のような初期化変数がある場合、ROMに配置すると、配置後、データの書き換えができなくなります。

```
#define INITIAL_MODE 10
int Start_Mode = INITIAL_MODE;
```

このようなデータは、ROMに初期値を用意し、実行時(電源投入時、または、リセット時)その内容を実際に実行するデータのアドレスにコピーを行う必要があります。これは、スタートアップモジュールの機能です。このセミナーの教材のH8用TOPPERS/JSPのスタートアップモジュールはconfig/h8/Start.Sに記述されています。ちょうど、100行目あたりで、コピー処理を行っています。

組込みソフトウェアのデバッグ環境1 デバッガ(debugger)とは？

- バグ(bug)の発見を支援するツール
- 主な機能(ソースコードデバッガ)
 - ▼プログラムのロード
 - ▼プログラムの実行と停止
 - ▼プログラムの読み出し(対応するソースコードの表示)
 - ▼データの読出し、書換え(変数、メモリ、レジスタ)
 - ▼関数(またはサブルーチン)の呼出し
 - ▼ブレークポイント(実行ブレーク、アクセスブレーク)
 - ▼ステップ実行(ソースコードレベル、マシン語レベル)
 - ▼スタック状態の読出し、バックトレース

2005/12/17

TOPPERSプロジェクト認定

17



デバッガの主な機能は以下のとおりです。

1) プログラムのロード

開発システム上のプログラムをデバッガの通信機能を用いて、ターゲットシステムのRAM(時にはROM)にダウンロードします。

2) プログラムの実行と停止

ロードしたプログラムを指定の場所から、指定の場所まで実行します。

3) データの読出し、書換え

ターゲットシステム上のデータの読出しと書換えを行います。また、停止中のCPUレジスタの読出しや書換えを行います。

組込みソフトウェアのデバッグ環境2

■ ROMモニタ

- ▼ターゲットシステムのROMに入れておき、ターゲットシステムをシリアルインターフェイスなどで端末として接続し、使用するデバッガ
- ▼ソースコードレベルのデバッグ機能は持たない

■ リモートデバッガ

- ▼高度なデバッガ(ソースコードレベルのデバッグ機能など)をホストシステム側のデバッガで実現
- ▼ターゲットシステム上には、ターゲットシステムの状態を参照・設定するための最低限のソフトウェア(デバッグエージェント、デバッグモニタ)のみを実装
 - ・レジスタ／メモリの読出し／書込み
 - ・実行の開始／再開、1ステップ動作
 - ・プログラムの強制終了(リセット)

2005/12/17

TOPPERSプロジェクト認定

18



この実習で使用する苫小牧高専版 H8 簡易モニタのコマンドを以下に載せます。

- ld** モトローラ S レコード形式のデバッグモジュールをRAM にロードします。
- go [<addr>]** デバッグモジュールをアドレス <addr> から実行します。
 <addr> を省略すると、リセット例外処理ベクタアドレスに指定されている
 アドレスから実行します。
- dw [<addr> [<len>]]**
 アドレス <addr> から、長さ <len> ワード(2バイト)データを出力します。
 <addr> を省略すると、前回出力したデータの次から 256 バイト出力します。
 ただし、初めての実行の場合は、内蔵 RAM の開始アドレスから 256 バイト出力します。
- db [<addr> [<len>]]**
 アドレス <addr> から、長さ <len> バイトのデータを出力します。
 <addr> を省略すると、前回出力したデータの次から 256 バイト出力します。
 ただし、初めての実行の場合は、内蔵 RAM の開始アドレスから 256 バイト出力します。
- ew [<addr>]** アドレス <addr> から、ワードデータの書き込みモードになります。
 <addr> を省略すると、前回書き込んだ最後のアドレスの次から書き込みモードになります。
 書き込みモードでは、何も入力しないで改行したときは表示されているメモリへの書き込みは
 行いません。また、'!' のみ入力すると書き込みモードを終了します。
- fb [<addr> [<len>] [<init> [<inc> [<repeat>]]]]**
 アドレス <addr> から、長さ <len> バイトのデータを書き込みます。
 <addr> の省略値は内蔵 RAM の開始アドレスであり、<len> の省略値は 256 バイトです。
 また、<init> は書き込みを開始するデータパターンの初期値、<inc> は書き込みを
 開始するデータパターンの増加値、<repeat> は書き込みの繰り返し回数です。
- ?** 簡単なコマンドの書式を出力します。

組込みソフトウェアのデバッグ環境3

■ ICE (In-Circuit Emulator)

▼ボードのプロセッサを外し、代わりにICEをつなぐことにより(プロセッサのソケットにプローブを差す形が普通)、ICEがプロセッサをエミュレートして、実行を監視する

■ ROMエミュレータ

▼プロセッサではなく、ROMソケットにプロ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

■ 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

▼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監視できない

▼プロセッサの高速化により、プローブを差すと動作しなくなる／動作が不安定になる



2005/12/17

TOPPERSプロジェクト認定

19

TOPPERS

・ICE (In-Circuit Emulator)

ボードのプロセッサを外し、代わりにICEのプローブを接続することにより、プロセッサの処理をICEがエミュレートを行います。ICEの内部から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たり、変えることが可能になります。そのため、デバッガではできない、システムを停止しないデバッグ作業が可能になります。また、問題発生時点の(前)後の実行ログをとることにより、通常実行で発生する障害解析も可能になります。ユーザインターフェイスはホスト上で動作させるものが多いようです。プロセッサと基板の間を切り離すことにより、強力なデバッグ支援機能を実現しているとも見えます。

・ROMエミュレータ

プロセッサではなく、ROMソケットにプローブを差すことにより、プロセッサのインストラクション機能を監視するエミュレータです。プロセッサの動作を止めたり、変えるためにはNMIを使います。

・ICEの問題点

ICEは種々の問題があり、オンチップデバッグ機能に対応したICEが主流となりつつあります。ICEにはプロセッサ毎に開発しなければならないために高価になります。

→ROMエミュレータ

ワンチップ化やキャッシュにより、チップの外からバスが監視できない

→エバチップ (Evaluation Chip)

→オンチップデバッグ機能

プロセッサやバスの高速化により、プローブを差すと動作しなかったり、動作が不安定になる場合があります。

→オンチップデバッグ機能

組込みソフトウェアのデバッグ環境4

- 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
 - ▼オンチップデバッグ機能の標準化の試みも
EJTAG、nWIRE、NEXUSなど
- シミュレータ
 - ▼ハードウェアが完成する前にソフトウェアをテスト・デバッグしたい場合に広く用いられる
 - ▼シミュレーション速度と時間精度トレードオフに
 - 速度・精度・価格でいろいろな種類のシミュレータがある
 - ▼コシミュレーション (ハードウェアとソフトウェアを同時にシミュレーションするための技術)

2005/12/17

TOPPERSプロジェクト認定

20



・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、デバッグ時専用のチップ

・オンチップデバッグ機能

デバッグ支援回路がチップ内に実装されているもの、ICE (のサブセット) がチップ内に入っているイメージと考えてください。オンチップデバッグ機能の標準化の試みもあります。

・シミュレータ

ハードウェアが完成する前に、ソフトウェアを開発、デバッグしたい場合に広く用いられる方法です。シミュレータ速度と時間精度と価格をトレードオフに種々のシミュレータがあります。

- ・Cycle Accurateなシミュレータ (CAS、低速)
- ・命令セットレベルのシミュレータ (ISS、中速)
- ・C言語レベルのシミュレータ (高速、時間精度は無視)

コシミュレーションのように、ハードウェアとソフトウェアを同時にシミュレーションする技術もあります。

実質的なデバッグ方法

■ ログ出力によるデバッグ

- ▼print文やsyslog文を使い、実行ログをコンソールに表示して実行状態や順序を確認しながらデバッグする
- ▼タスクの実行を停止しないため、制御対象機器の実行手順に沿った確認ができる

■ アナライザを用いたデバッグ

- ▼ICE等の機器は、ハードウェアやハードウェアとのインターフェイス部分のデバッグを主な目的とする
 - ➡ デバッグ機能よりトレース機能を重視
- ▼障害発生環境で、アナライザをポートやメモリにつないで問題発生前後のアクセスやデータのログを取って解析を行う

2005/12/17

TOPPERSプロジェクト認定

21



・ログ出力によるデバッグ

マルチタスク環境で、デバッグを行う場合、対象のタスクをデバッガを使って停止してしまうと制御対象機器に対する制御順序に狂いが生じて、正しく制御できないケースが発生します。そこで、**syslog**文や**printf**文を用いて、実行ログを表示させデバッグを行う方法の方が、効率的にデバッグを行うケースが多々あります。特に、この方法の開発の有用な開発環境として以下のものがあげられます。

- 1) **LINUX**や**Windows**のような開発環境でシミュレーション開発を行い実行確認済みのソースが参照できるミドルウェアをターゲット環境で動作させていく場合、ログ出力を確認しながら動作確認を行うような開発手順があります。
- 2) 割込みや周期ハンドラのような非タスクコンテキストでは、デバッガでブレークを行えない場合があります。この場合、**syslog**文によるログによる確認は有用な手段です。

このセミナーの開発でも、ログ出力による開発が有用な手段となります。

組込みシステムのテスト環境1

■ テストにかかるコストが大きい理由

- ▼高い信頼性が求められること
- ▼タイミングに起因する非決定性が多いこと
 - ▼制御対象の機器の動作タイミングに依存した非決定性
 - ▼極めて再現性の低い障害が起こりうる(よく起こる)
- ▼制御対象の機器も「動作」させる必要があること

■ 動作させることの困難さ

- ▼機器を動作させるのに大きいコストがかかる
- ▼(そもそも)機器を動作させることができない
- ▼機器に異常動作をさせることができない
- ▼機器が完成していない

➡ **テスト(およびデバッグ)環境の構築が必要**



2005/12/17

TOPPERSプロジェクト認定

22

TOPPERS

・組込みシステムのテスト環境

組込みシステムのテストにかかるコストが大きい理由としては以下の要因が挙げられます。

1) 組込みシステムの誤動作が、機器の誤動作に直結する 경우가多く、機器の誤動作が人命にかかわる場合もあります。また、大量に販売されるものであれば、修繕に対するコストも高価となります。そのため、高い信頼性が求められる組込みシステムも多くあります。

2) 機器の動作のタイミングや種々の機器の複合した動作によって、予測できない制御を組込みシステムに求められる場合があります。このようなケースでは、再現性の低い障害が起こりうる場合も多く、検証方法として、ブラックボックス的なテストも必要となります。組込みシステムにはタイミングに起因する非決定要素が多くあります。

3) テストにあたっては、制御対象の機器も動作させる必要があります。このような機器が試作機レベルの場合、高価なことも多くあります。

組込みシステムでは、制御対象機器を「動作」させ、テストを行う必要があります。テスト(デバッグを含めて)を行う場合、テスト環境の計画的な構築が必要です。しかし、テストやデバッグ用に機器を動作させること自体が困難なケースがあります。

1) 機器を動作させること自体が大きなコストがかかる場合があります。

2) 種々な条件を整えないと、機器を動作させることが出来ない場合があります。

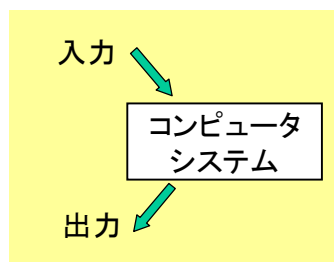
3) 機器に異常動作をさせるためには、機器自体の改造が必要な場合もあり、特殊な機器をテスト用に用意する必要がある場合があります。

4) 機器がまだ完成していない場合があります。

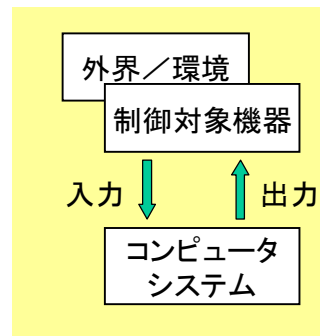
組込みシステムのテスト環境2

■ 制御対象の機器を「動作」させる必要

- ▼制御対象の機器も(機械的、電子的、物理的、化学的、…に)複雑なシステム
- ▼単に入力を与えて出力をチェックするだけではテストできない



普通のシステム



組込みのシステム

2005/12/17

TOPPERSプロジェクト認定

23



TOPPERS

エンタープライズ系のシステムでは、開発環境＝実行環境のケースが多く、このような場合、比較的簡単にデバッグ、テスト環境を構築しやすくなります。組込みシステムでは制御対象機器(これは外界や環境によって特別な動作をする場合がある)からの入力に従って、リアルタイムに出力データを機器に与える必要があります。制御対象機器も複雑なシステムである場合もあります。

リアルタイムOSを用いた システム設計の指針

2005/12/17

TOPPERSプロジェクト認定

24



リアルタイムOSを用いたソフトウェア開発1 上流開発

- 企画仕様書、要求仕様書など要求を元にシステムの分析を行う
- 分析の方法は、構造化分析、オブジェクト指向分析、会社独自の等方法等まちまち
- UMLに従ったモデルを作るケースが増えてきた

構造化分析では

イベントリスト
コンテキストダイアグラム
データ辞書
DFD0(基本分析)
非機能要件の調査

オブジェクト指向分析では 以下は 代表例

コンテキストダイアグラム
ユースケース、ユースケース記述
クラス図、オブジェクト図
状態遷移図、シーケンス図、イベントリスト

2005/12/17

TOPPERSプロジェクト認定

25

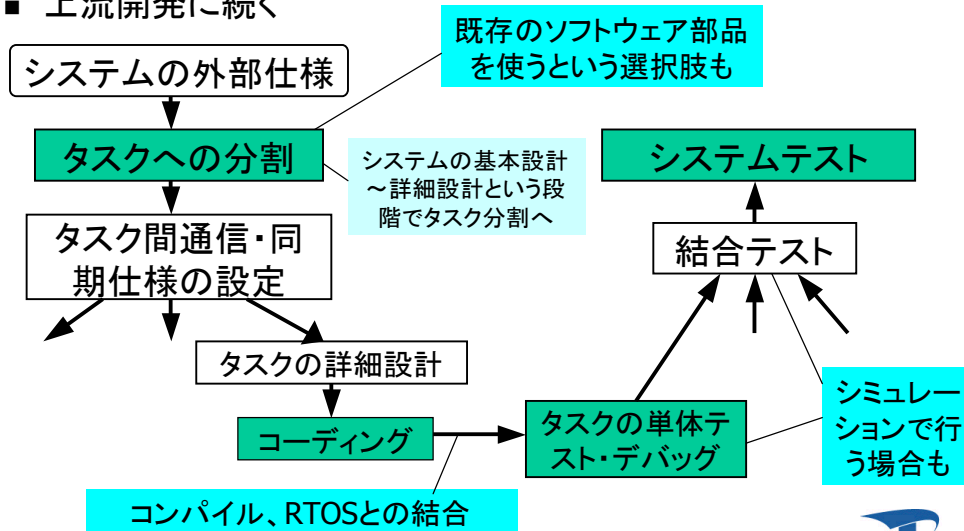


上流開発

企画仕様書や要求仕様書を元に、システムの分析を行います。これまでのセミナーでは構造化分析を用いて分析を行いましたが、実際の現場では、その会社、プロジェクトチーム等での独自の手法を含めて、いろいろな方法があります。上記の例では、構造化分析を行った場合の図や、オブジェクト指向分析でソフトウェアをモデルで表現する場合のUMLの図を挙げました。

リアルタイムOSを用いたソフトウェア開発2 典型的な設計フロー

■ 上流開発に続く



2005/12/17

TOPPERSプロジェクト認定

26

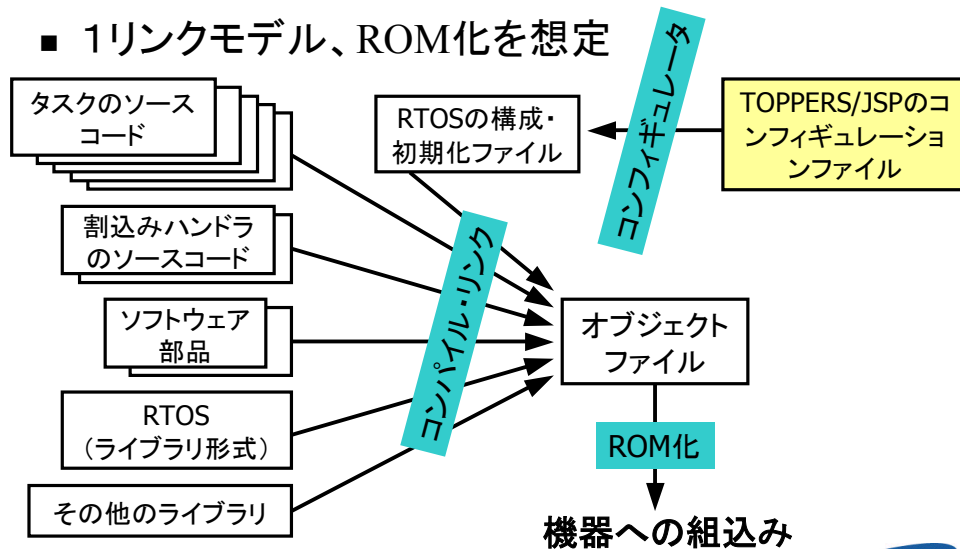


分析の結果を元に、実際に動作するプログラムの設定を行います。リアルタイムOSを用いる場合、システムをいくつかのタスクに分割する必要があります。分割の手法については、後の節で説明を行います。その後、タスク間の通信の仕様を決定します。ソフトウェア開発手法としてウォーターフォール型とスパイラル型があります。ウォーターフォール型は最初に決めた仕様でプログラムを作成し、最終テストまで行う方法ですが、スパイラル型は開発を幾つかのイテレーションに分けて行います。最初のイテレーションで基本となる重要な機能を満たすタスクを設計し、機能や性能を満たすことを確認した後、以後のイテレーションで機能拡張していく手法です。システムに不確定事項が多い場合や規模が大きい場合はスパイラル型が向くと言われています。

タスクの分割、通信手法の決定後、タスクごとに詳細設計、コーディングを行います。作成したタスクに対して単体テストを行い、それが終了後、複数のタスクを集めて結合テストを行います。システムテストは外部仕様書を元に個々の仕様の確認テストを行います。

リアルタイムOSを用いたソフトウェア開発3 ソースコードからROM化

■ 1リンクモデル、ROM化を想定



2005/12/17

TOPPERSプロジェクト認定

27



通常のリアルタイムOSを用いたシステムでは、1リンクモデルでビルドを行いROM化します。システム規模が多くなると、プロセス化したプログラムを必要に応じてロードして実行する場合がありますが、ここでは説明を行いません。

上記の図ではリアルタイムOSの1リンクモデルの場合を示します。詳細をTOPPERS/JSPの各開発例で説明します。タスクのソースコード、割込みハンドラのソースコード、ソフトウェア部品 (ライブラリの形もある)、TOPPERS/JSP (ライブラリの形が多い)、その他のライブラリ、OSオブジェクトの初期化や構成をソース化したものをコンパイル、リンクし、ROM化して機器に組込みます。TOPPERS/JSPではコンフィギュレーションファイルを作成すれば、コンフィギュレータ (コンフィギュレーションファイルをソースコードに変換するプログラム) を用いてOSオブジェクトの初期化や構成をもつソースコードに変換を行います。また、本実習のTOPPERS/JSPを用いたシステムではリンク後のオブジェクトをモトローラのSフォーマットに変換し、ROMモニタを用いてH8/3069Fのボード上のSRAMにダウンロードして実行します。ROM実行する場合は、同じくSフォーマットデータをワンチップCPU上のフラッシュROMへツールを使って書き込みを行って実行が可能です。

リアルタイムOSを用いたソフトウェア開発4

RTOSのコンフィギュレーション

- RTOSの構成や初期状態を決める手順
- どのような構成が変更できるかはRTOS毎.例えば,
 - ▼用いる/用いない機能
 - ▼各オブジェクト(タスクやセマフォ)などの最大数
 - ▼タスクの優先度の段階
 - ▼メモリの割付け/配置
 - ▼各オブジェクトの生成・定義情報
(オブジェクトを静的に生成する場合)
- コンフィギュレーションの機構
 - ▼コンフィギュレーションによりRTOSのコードを変える機構
 - ▼コンフィギュレーション結果は1つ(または複数)のソースコードに変換する方法

2005/12/17

TOPPERSプロジェクト認定

28



μ ITRON4.0仕様におけるコンフィギュレーション

- カーネルオブジェクトは静的に生成
 - オブジェクト生成情報の、コンフィギュレーションファイル中での記述を標準化(静的API)
- 静的APIの例


```
CRE_TSK ( tskid, { tskatr, exinf, task, itspri,
                  stksz, stk });
ATT_ISR ({ intatr, exinf, intno, isr });
```
- ソフトウェア部品の静的APIを混在して記述可能
- 利点
 - ▼ソフトウェアを別のカーネルへ移植するのが楽に
 - ▼ソフトウェア部品の組込みが容易に
 - ▼サービスコールと静的APIを個別に覚える必要がない

2005/12/17

TOPPERSプロジェクト認定

29



μ ITRON4.0のスタンダードプロファイルに準拠したリアルタイムOSでは、静的APIを記述したコンフィギュレーションファイルをコンフィギュレータを用いてソースコードに変換します。また、ITRON TCP/IP仕様でも、通信端点の生成用の静的APIがあり、プロトコルスタックのオブジェクトを生成します。

通常のコフィギュレーションは、メニュー形式でコンフィギュレーション設定を行います。そのため、OSのメーカーやCPUが変われば、設定をしないする必要があります。静的APIで記述したコンフィギュレーションは標準的な仕様ですので、メーカーやCPUに依存せず、再利用が可能です。また、RTOSの初期化やオブジェクトをC言語で直接記述するには、実装環境やリアルタイムOSの知識が必要です。静的APIを用いて記述すれば、このようなノウハウはコンフィギュレータ中に隠蔽され、コンフィギュレータの作成者が実装部分を作成します。そのため、リアルタイムOSの利用者は実装設計を行う必要はありません。

タスクへの分割指針1

タスクへの分割

- どのようなアプリケーションにも適用できる一般的な方法は存在しない
- 次のような要因を考慮に入れて決定すべき
 - ▼アプリケーションの機能と性能要件
 - ▼ソフトウェア部品として外部から導入する部分
 - ▼ソフトウェアの開発体制

リアルタイム性向上の為の分割指針

- ！ 論理的な処理手順と時間的な処理手順の分離により、時間制約を持ったプログラムの保守性が向上
- デッドラインの異なる処理は別タスクに
- (過負荷になる場合) 重要度が違う処理は別タスクに

2005/12/17

TOPPERSプロジェクト認定

30


 TOPPERS

組込みシステムは広い範囲の機器が含まれます。また、非機能要件や実装アーキテクチャによって、タスクの設定は大きく影響を受けます。そのため、どのようなアプリケーションにも対応できる一般的なタスク分割方法は存在しません。タスク分割の際には、以下の要因を考慮してください。

- 1) アプリケーションの機能と性能の要件
- 2) ソフトウェア部品として外部から導入する部分
- 3) ソフトウェアの開発体制

リアルタイムOSのタスク機能の特徴は、論理的な処理手順と時間的な処理手順を分離できることです。例えば、構造化設計やオブジェクト指向設計は凝縮度や結合度を最適化させることにより、モジュール化、オブジェクト化を行う手法であり、論理的な処理手順と時間的な処理手順を分離するメカニズムは含まれません。そのため、タスク分割とモジュールやオブジェクトはうまく対応しない面があります。リアルタイム性を向上させる為のタスク分割には以下の点に留意してください。

- 1) デッドラインの異なる処理は別タスクにする

デッドラインは、その機能に課せられた締め切り時刻であり、リアルタイムシステムでは、締め切り時間以内で処理を終えなければならない。デッドラインが異なる処理は別タスクにした方が、優先度の設定等でデッドラインの対応が行いやすい

- 2) (過負荷になる場合) 重要度が違う処理は別タスクとして分ける

タスク化により、重要度の違うタスクは、別の優先度で実行させることにより、過負荷対応でできる。

タスクへの分割指針2

モジュール化のための分割指針

！ソフトウェアの構造化・開発容易化のための分割

- 異なる機能モジュールや異なるデバイスの操作は別タスクに
- 異なる種類の処理は別タスクに
 - ▼I/Oタスクと内部処理タスクに分割
 - ▼システムの監視・保守・診断用のタスクを別タスクに
 - ▼例外時、故障時の処理を別タスク化する方法もある
- 再利用しやすい単位は単独のタスクに
- まとまった処理単位は同一のタスクに
 - ▼一つの状態遷移/一つの結果を出す処理/データの流が閉じた処理は同一タスク内で
 - ▼状態(モード)毎に別タスクにする方法もある→Cスベック

2005/12/17

TOPPERSプロジェクト認定

31



今度は、ソフトウェアの構造化、開発容易化の指針でタスク分割を考えます。ここでは、構造化分析、オブジェクト指向分析の結果が分割の要因となります。

1) 異なる機能モジュールや異なるデバイスの操作は別タスクに割り当てます。

2) 異なる種類の処理は別タスク化します。例としては以下の指針があります。

- ・I/Oタスクと内部処理タスクに分割します。
- ・システムの監視・保守・診断用のタスクは別タスクとして分けます。
- ・例外発生時、故障時の処理を別タスク化する方法もあります。別タスク化しない方法もあります。

3) 再利用しやすい単位は別タスクにした方が、機能アップ等が発生した場合、対応しやすい。

4) まとまった処理単位は同一のタスクとします。

- ・一つの状態遷移、一つの結果を出す処理、データの流が閉じた処理は同一のタスクに入れます
- ・状態(モード)毎に別タスクにする方法もあります。例として、構造化分析から構造化設計を行う場合、Cスベックという手法が用いられます。

タスクへの分割指針3

モジュール化のための分割指針(続き)

- 同一処理を複数並行して行いたい場合
 - ▼同一のプログラムコードを共有して、複数のタスクを動作させる方法
 - ▼データで切り分ける方法もある
- 開発者/グループ毎にタスクを分割する
- 起動イベント/起動タイミングの異なる処理
 - ▼起動イベント/タイミング毎にタスクを分ける方法
 - ▼起動周期毎にタスクを分ける方法は一般的
 - ▼起動イベント/タイミングが異なっても、まとまった処理単位は同一タスクとする方法も

設計上の留意事項

- 分割のしすぎはオーバーヘッドの増大につながるので注意

2005/12/17

TOPPERSプロジェクト認定

32



5) 同一処理を複数並行して行いたい場合

- ・同一のプログラムコードをタスク化し、別のデータを対応させ並列実行させる法があります
- ・上記の方法ではオーバーヘッドやメモリリソースを多く使用するため、別々のデータをひとつのタスクで状態遷移させる方法もあります。

6) 開発者やグループ毎にタスク分割した方が、開発が容易になります。

7) 起動イベントや起動タイミングの異なる処理で分割を行います。

- ・起動イベントやタイミング毎にタスクを分ける方法があります。
- ・起動周期毎にタスクを分割する方法は一般的です。
- ・起動イベントやタイミングが異なっても、まとまった処理単位は同一のタスクとする方法もあります。

注意:

システム上のタスクを分割しすぎると以下の問題が増大します。

- ・オーバーヘッドが増大
- ・メモリ効率も低下
- ・排他制御の複雑化

タスクの優先度の決定

優先度の決め方(明日の講義で詳細説明)

- 時間制約を満たせる範囲では、急ぐタスク(デッドラインの短いタスク)優先
- 一時的な過負荷時に時間制約を満たせない場合には、重要なタスク(時間制約を満たせなかった場合、被害が大きいタスク)を優先

時間制約を満たせるかをどう判断するか？

- リアルタイムスケジューリング理論が適用可能

一時的な過負荷時にはどうするか？

- 重要性の低いタスクをさぼる/精度を落とす
- 種々のテクニックがある(今までの内容と矛盾？)

2005/12/17

TOPPERSプロジェクト認定

33



タスクの優先度については、明日のセミナーで「リアルタイム性の向上手法」で説明を行います。

タスク間通信・同期の設計

共有メモリ vs メッセージ通信

- 基本的には、一方で実現できることは、もう片方でも実現できる
 - ▼一般的には、共有メモリの方がオーバーヘッドが小さい
 - ▼システム検証には、通信粒度が大きくRTOSレベルで監視できるメッセージ通信の方が扱いやすい
 - 例えば、問題の切り分けが容易
- 状況により、どちらかが有利/便利な状況がある
 - ▼情報の流れが1:nの場合(ブロードキャストを含む)や、情報が必要なタイミングが不定の場合には、共有メモリが有利
 - ▼情報のキューイングが必要な時は、メッセージ通信が便利

設計上の留意事項1

- 二つの方式を混在させて使うと、システム設計が複雑化し、見通しが悪くなる場合があるので注意
- セマフォ/ミューテックスによる排他制御には、デッドロックと優先度逆転に関して注意が必要
- メッセージ通信ではサーバタスクに処理依頼する場合にも、優先度逆転の問題が起こりうる

デッドロック

- 2つ以上のタスクが、互いに相手の処理が進むのを待つ状態
 - ▼典型的には、2つ以上のセマフォ/ミューテックスを、タスクによって異なる順序ロックする場合に発生
 - ロックする順序を決められれば解決

2005/12/17

TOPPERSプロジェクト認定

35



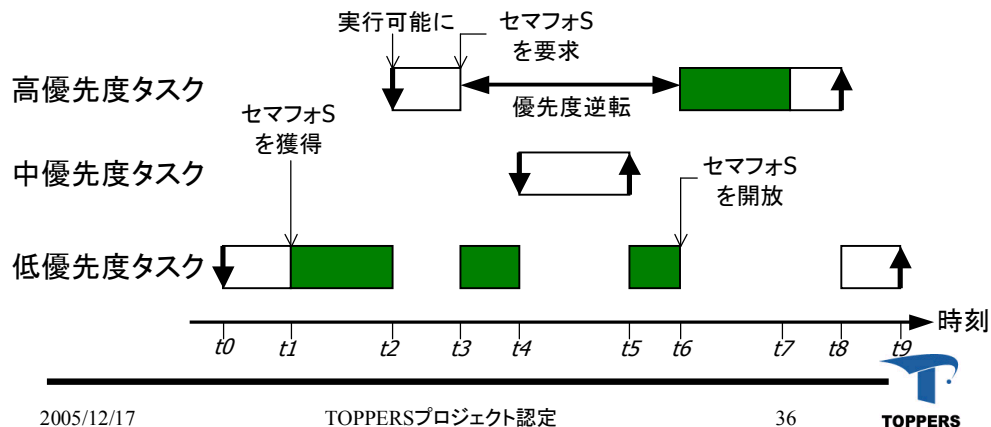
設計上の留意点は以下の3点

- 1) 二つの通信方式を混在させて使うと、システム設計が複雑化し、見通しが悪くなる場合があるので注意が必要です。
- 2) セマフォやミューテックスによる排他制御を行う場合は、デッドロックと優先度逆転の発生に注意が必要です。
- 3) メッセージ通信ではサーバタスクに処理依頼する場合にも、優先度逆転の問題が起こりえます。
 (例えば、優先度の高いタスクがデータキューにて優先度の低いサーバタスクに、処理依頼する場合、キューが満杯の場合、優先度の高いタスクは待ち状態となる)

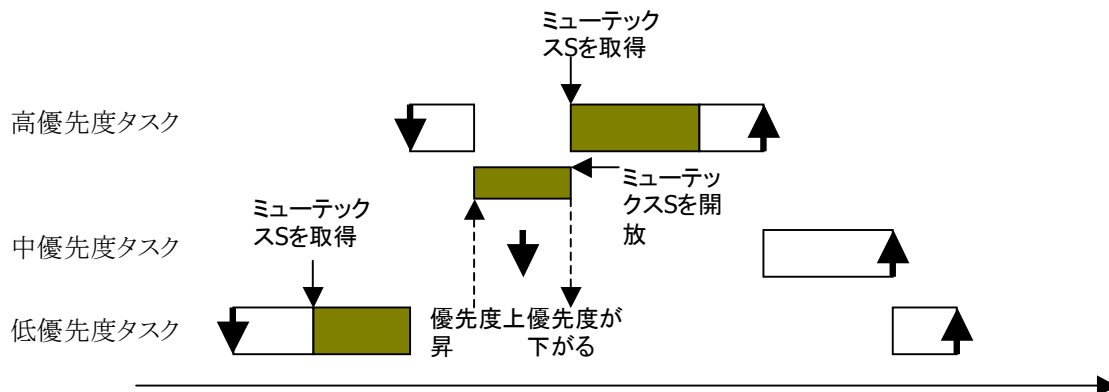
設計上の留意点2

優先度逆転(Priority Inversion)

- 高優先度タスクと低優先度タスクが資源を共有、資源に対する排他制御をセマフォSで実現している場合、低優先度タスクにより高優先度タスクの実行を待たされる



上記の操作を、セマフォではなく、優先度継承プロトコルを設定したミューテックスで行った場合、排他制御の動作が異なります。優先度継承プロトコルを設定すれば、低優先度タスクがロックしたミューテックスを高優先度タスクが待ち状態になった時点で低優先度タスクの優先度が高優先度タスクの優先度まで上げられるので、中優先度の実行終了を待つ必要がなくなります。



テスト(検証)とデバッグ

タスクの単体テスト

- 該当のタスクを含んだ最小構成でリンク
- 他のタスクの振舞いはシミュレーションで
- タスクの単体機能を確認する
- タスクの処理時間、スタックの使用率などを評価するなど

割込みハンドラの単体テスト

複数タスクの結合試験

- シミュレーションされているタスクを単体テスト済みのタスクに入れ替えていく
- タスク間のインターフェイスの確認
- 資源の競合のテスト(排他制御が正しいか?)
などなど

2005/12/17

TOPPERSプロジェクト認定

37



・テスト(検証)とデバッグ

タスクの単体テスト

単体テストを行う場合、対象のタスクの機能、入力に対して、どのような出力があるか、また、どのような振舞いを行うか等を項目としてまとめます。対象のタスクプログラムの入力や出力を確認できるプログラムを加えて、項目に従って、タスクの振舞いや出力の確認を行います。タスクの処理時間やスタックの使用率なども、あらかじめ測定しておきます。

割込みハンドラの単体テスト

割込みハンドラについても、タスクの単体テストと同様に、確認項目を決めてシミュレーション環境で単体テストを行います。

複数タスクの結合試験

結合試験では、重要な機能から確認を行い、付随的な機能を付加していきながら、全体の動作が正しく実行されるかの検証を行っていくのが一般的な検証手順です。そのために、重要な機能を実現するタスクを先に作成、検証し、付随的な機能のタスク等は、インターフェイスをもつダミーのプログラムで代行させたり、結合しなかったりして、まず、ビルドを完了させ結合機能テストを行います。このとき、タスク間のインターフェイスの確認や資源の競合テストも行います。付随的な機能をもつタスクの検証が終わり、ビルド中のダミープログラムと入れ替えたり、インターフェイスを追加し結合を行います。新たな機能を含んだ結合テストを行い、問題のある部分の修正を行いながら、最終仕様に向けたビルド、結合テストを繰り返していきます。

デバッグ環境のRTOSサポート機能

- RTOSをサポートするデバッグ環境(ソフトデバッガ、ICEなど)は次のような機能を持つ
 - ▼オブジェクト状態の読み出し
 - タスクやセマフォなどの状態を表示
 - ▼タスクのコンテキストの取り出し
 - 待ち状態のタスクのレジスタなどを表示
 - ▼システムコールの発行
 - システムコールによりオブジェクト状態を変更
 - ▼タスクを意識したブレークポイント
 - 指定したタスクが指定したアドレスを実行した時に、
該当のタスクのみを止める機能など
 - ▼RTOSの実行ログの取得と表示
 - タスク切替えなどの情報を記録し、後で表示

2005/12/17

TOPPERSプロジェクト認定

38



市販のリアルタイムOSではリアルタイムOSをサポートしたデバッグ環境が提供される場合が多いようです。この教材では、補助用にタスクモニタが提供されています。その内容に合わせて説明を行います。

1) オブジェクト状態の読み出し

タスクのみ参照可能: DISPLAY TASKコマンド

2) タスクのコンテキストの取り出し

レジスタの参照: DISPLAY REGISTERコマンド

3) システムコールの発行

act_tsk: TASK ACTIVATEコマンド

ter_tsk: TASK TERMINATEコマンド

sus_tsk: TASK SUSPENDコマンド

rsm_tsk: TASK RESUMEコマンド

chg_pri: TASK PRIORITYコマンド

4) タスク意識したブレークポイント

サポートしていません

5) RTOSの実行ログの取得と表示

サポートしていません

実習課題の説明

1. RTOSを用いたリアルタイムシステム構築
2. 実習課題の説明
3. 開発環境の確認
4. 話題沸騰ポット作成1

2005/12/17

TOPPERSプロジェクト認定

39



実習課題の説明を行います。

- 実習で設計、製作を行う「話題沸騰ポット(GOMA-1015型)」についての説明
- 実行環境は、**netDevice**を用いたシミュレーション環境を使用しますのでこの環境と制約事項についての説明を行います。

SESSAME: 話題沸騰ポットの実装 H8/3069を使用するためハード仕様に変更があります

- 話題沸騰ポットは(GOMA-1015型)要求仕様書: 第3版をベースとして試作機を作成します
- 試作ポットは製作しますが、専用LEDは製作せず、温度と水位とタイマー表示については、16文字×2行のLCDモジュールに表示することになります



2005/12/17

TOPPERSプロジェクト認定

40



・話題沸騰ポット(GOMA-1015型)要求仕様書 第3版

GOMA-1015型はSESSAMEの教材として作成されたものです。

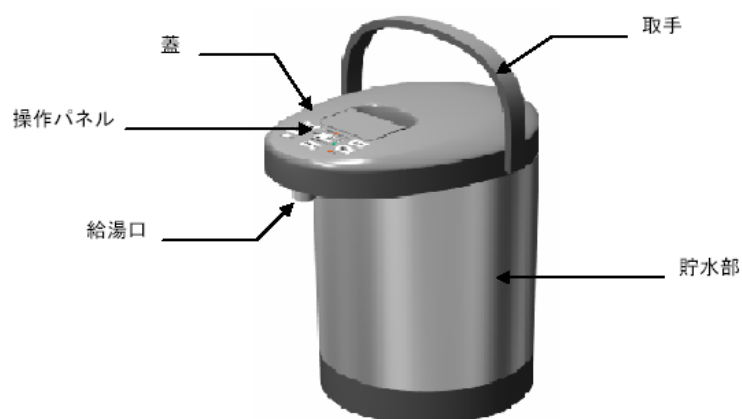
1.機能要件

今回設計する沸騰ポットは、ユーザに以下の機能を提供する家電製品です。

- ・ポット内の水を沸騰・保温する機能
- ・ポット内の水を給湯する機能
- ・ユーザが指定した時間がきたら、ブザーを鳴らして知らせるキッチンタイマ機能

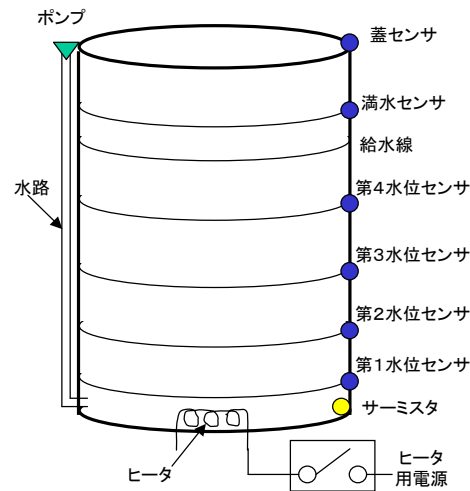
2.ポットの概観と名称

ここの部位の名称、機能については、一般的なものなので説明を省きます。



ハードウェアメカ部の構造 貯水部の構造を説明します

- 貯水部の構造は右図の通りです
- ヒータと電源供給部にスイッチがあります
- 細かい温度制御はヒータスイッチで行います
- 温度検知用のサーミスタは貯水部の底についており、ADコンバータにより温度に変換します



2005/12/17

TOPPERSプロジェクト認定

41



・各部位の用語説明

1) 満水センサ

水位が、このポットの許容上限を超えていないかどうかを検出します。このセンサがオンの時、水位が許容上限を超えていることになります。この場合ヒータを切らなければなりません。

2) 第n水位センサ

水位を検出します。各センサはオンの時、その位置より水位が高いことになります。

3) 蓋センサ

蓋が開いているかどうかを検出します。蓋が閉じている時にオンになります。

4) サーミスタ

ポット内の水温を検出します。温度は電圧で出力されます。この電圧をA/Dコンバータを用いて温度の数値に変換します。

5) ヒータ

ポット内の水を加熱します。

6) ヒータ用電源

ヒータへの電力を供給します。通常はオンですが、ヒータに異常が発生した場合にオフにして電力を遮断します。

7) 給水線

ユーザに、このポットに入れることができる水量の上限を知らせるための目印です。満水センサより若干下にあります。

8) ポンプ

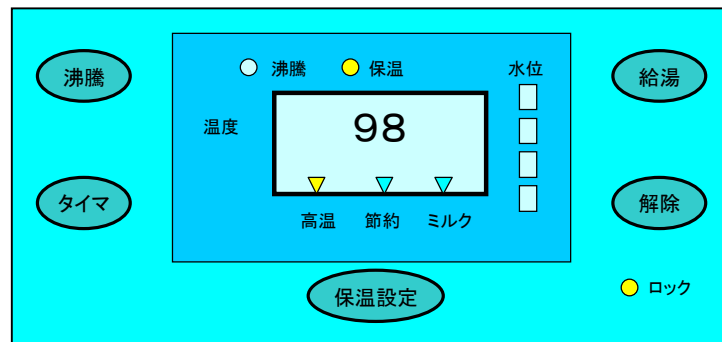
ポット内の水を吸い上げて、給水口から排出します。

9) 水路

ポンプによって吸い上げられる水の通路です。

操作パネル部1 試作ポット上のパネル

- 下図が試作ポット上の操作パネルです
- 温度の表示部と水位の表示部はハリボテで実際の表示はH8/3069Fボード上のLCDに表示されます



2005/12/17

TOPPERSプロジェクト認定

42



試作ポットのパネルは5つのプッシュボタンと7つのランプから構成されます。

・プッシュスイッチ

1) タイマボタン

このボタンを押すとタイマが起動し、1回押す毎に1分追加されます。

2) 保温設定ボタン

このボタンを押すと、保温モードを高温(98度保温)、節約(90度保温)、ミルク(60度保温)モードに設定します。1回押す毎に高温→節約→ミルク→高温とモードが変わります。

3) 解除ボタン

給湯口のロックと解除を行います。ロック中は、給湯ボタンを押しても水は出ません。ロック中に押すとロックは解除され、解除されている時に押すと給湯口をロックします。また、給湯中はロックできません。

4) 給湯ボタン

このボタンを押すと、ポンプを動作させて給湯口から水を排出します。押している間水は給湯を行い、ボタンから手を離すと給湯を停止します。

5) 沸騰ボタン

このボタンを押すと、ポット内の水を沸騰させてカルキ抜きを行います。沸騰中に押すと、沸騰を中断して保温状態になります。1回押す毎に沸騰→保温→沸騰と変わります。

・ランプ

1) 温度・モードランプ

高温、節約、ミルクの文字の上に▼のランプが点灯してモードを示します。

2) ロックランプ

給湯口がロックされているかどうかを表します。給湯口がロックされている時点灯します。

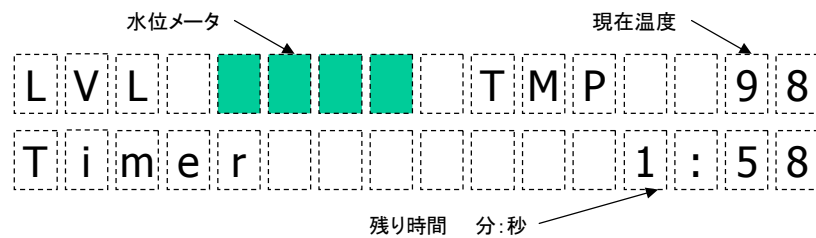
3) 沸騰、保温ランプ

沸騰ランプは沸騰時点灯、保温時消灯します。保温ランプはその逆の動作です。

操作パネル2

H8-3069Fボード上のLCD

- 16文字×2行LCDモジュールの1行目に、水位レベル(4レベル)、温度を表示します
- 2行目にキッチンタイマーの残りの分:秒を表示します
- タイマがオフの場合、2行目は表示しません



2005/12/17

TOPPERSプロジェクト認定

43



・LCDへの表示

1) 水位メータ

ポット内の水位を表示します。

水位メータと各水位センサの関係は以下の通りです。

8文字目←□→第4水位センサに対応

7文字目←□→第3水位センサに対応

6文字目←■→第2水位センサに対応

5文字目←■→第1水位センサに対応

2) 現在温度

現在の温度を表示します。

3) タイマ残り時間表示窓

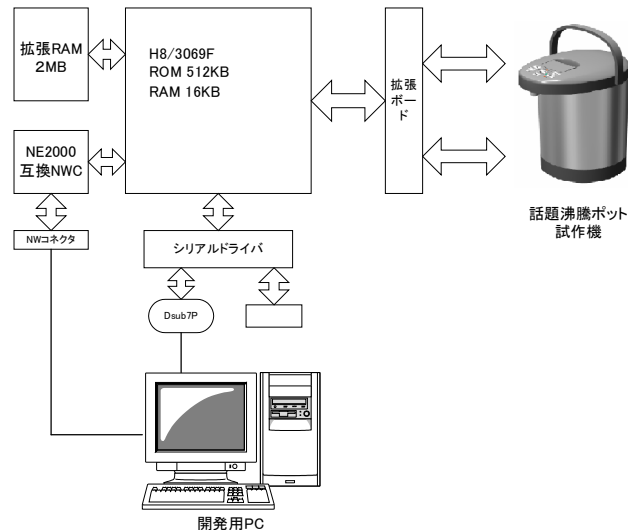
タイムアップまでの残り時間を分:秒で表示されます。

タイマがオフの場合は2行目は何も表示しません。

H8/3069Fの概要1

基本構成

- H8/300HをCPUコアにしたワンチップマイコンです
- 拡張RAMとネットワークコントローラが接続している



2005/12/17

TOPPERSプロジェクト認定

44



・H8/3069FはH8/300HをCPUコアとしたワンチップマイコンです。

- 1) フラッシュROM 512KB
- 2) RAM 16KB
- 3) 割込みコントローラ
- 4) 16ビットインテグレートドタイマーユニット
- 5) DMAコントローラ
- 6) リフレッシュコントローラ
- 7) ウォッチドックタイマ
- 8) シリアル コミュニケーション インターフェイス×2
- 9) A/D変換機
- 10) D/A変換機

・拡張RAM 2MB

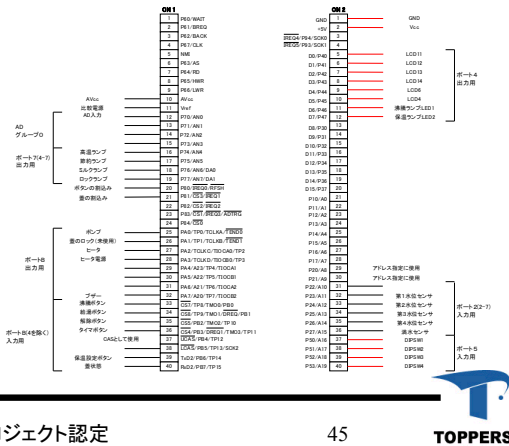
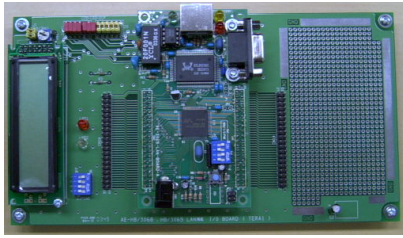
・NE2000互換ネットワークコントローラ、RTS8019ASを搭載

・フラッシュROMにTOPPERS/JSPプログラムがロード可能なROMモニタを書き込んでおきます。モニタが「開発用PC」のシリアルポートを用いて、ボードに接続しています。また、「開発用PC」とは、ETHERNETにて通信が可能です。「話題沸騰ポット試作機」とは、拡張ボードに接続している2つのチャンネルC N1とCN2を用いて試作機のデバイスとポートにて接続しています。

(実際は、話題沸騰ポットシミュレータとネットワークを介して接続している)

試作ポットとの接続 H8/3069Fボードとの接続

- 拡張ボードのCN1とCN2の空きポートに各デバイスのインターフェイスを割り当てます
- ポート4, 5は既存の拡張ボードをそのまま使用します



2005/12/17

TOPPERSプロジェクト認定

45



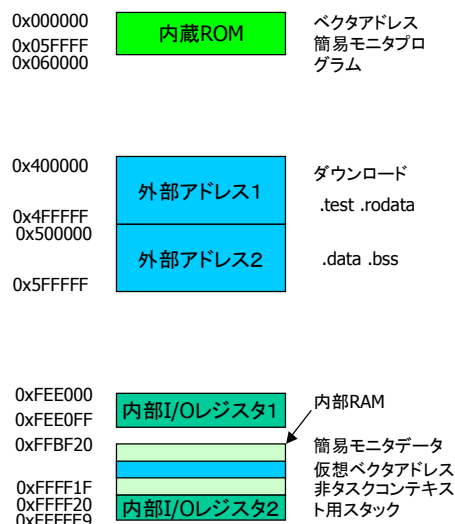
- ・拡張ボード上の2つのチャンネルCN1とCN2と「話題沸騰ポット試作機」の各デバイスとの接続は以下の通りです。
- 1) 割り込みとしてCN1の20, 21番のIREQ0とIREQ1を使用します。
 - 2) A/Dコンバータにて温度を取得するため、グループ0のAN0をセンサに接続し、サンプリング電圧をVrefから取得します。
 - 3) LCDは拡張ボードに付属のものをそのまま使用します。LCDの制御用にポート4を出力として使用します。
 - 4) ポンプ、ヒータ、ブザーの制御用にポートAを出力として使用します。
 - 5) 各ボタンや蓋の状態取り込み用にポートBを入力として使用します。
 - 6) ランプの表示用に、ポート4と7を出力用に使用します。
 - 7) ポート2は入力とし、各センサに接続します。
 - 8) 拡張ボードでDIP-SWに接続しているポート5はそのまま使用します。

H8/3069F概要2

CPU動作モードとアドレス空間

- 本開発のH8/3069Fは7つの動作モード中のモード5で実行します

- ROM上に簡易モニタが書き込まれており、左図の空色の部分にJSPを含むプログラムをロードし、実行することができます



2005/12/17

TOPPERSプロジェクト認定

46



H8/3069Fは外部RAMやLANドライバを持つ、これらを使用するためには、モード5の動作モード実行しなければなりません。これを使用するためにいくつかのポートは使用できなくなります。

本実習では、H8/3069FのフラッシュROM上に、苫小牧高専の阿部先生がつくられたROMモニタh8monを書き込んでいます。このモニタはH8用TOPPERS/JSPのデバッグモードでビルドしたモトローラSフォーマットの実行プログラムをダウンロードして実行することが可能です。ダウンロードしたプログラムは上記メモレイアウトの空色の部分に書き込まれます。この中には、**vector**セクションの割込みベクタも0xFFC000から0xFFC0FFに書き込まれ、仮想的に割込みベクタとして実行することができるため、フラッシュROMを書き換えずにTOPPERS/JSPを含むプログラムを実行することができます。

フラッシュROMには、保証書き込み回数がありデバッグのために何度もフラッシュROMに書き込みを行うと、新規に書き込めなくなる場合があります。

H8/3069F概要3

I/Oポート

- 動作モード5で実行する場合、いくつかのポートは使用できない
- H8/3069Fの拡張ボードはポート4、5で接続している
- 試作ポットはポート2、4、7、A、Bを使って接続する

ポート	概要	端子	接続
2	8ビットの入出力ポート、入力プルアップMOS内蔵	P17-P10 / A7-A0	入力、水位センサと接続
4	8ビットの入出力ポート、入力プルアップMOS内蔵	P47-P40 / D7-D0	出力、LCDと2つのランプと接続
5	4ビットの入出力ポート、入力プルアップMOS内蔵	P53-P50 / A19-A16	入力、DIP-SWと接続
7	8ビットの入出力ポート	P77/AN7 P76/AN6 P75-P70	出力、その他のランプ
A	8ビットの入出力ポート	PA7-PA0	出力、制御用
B	8ビットの入出力ポート	PB7-PB0	入力、ボタンと蓋

2005/12/17

TOPPERSプロジェクト認定

47

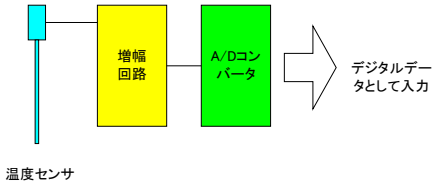


H8/3069Fの以下のポートはモード5、または、LANドライバを動作させるために使用しています。そのため、試作ポットと接続するために、これら以外のポートを使用しなければなりません。

- 1) ポート1
- 2) ポート2のビット0、1、2
- 3) ポート3
- 4) ポート5のビット4から7(無し)
- 5) ポート6
- 6) ポート8のビット2と3
- 7) ポート9
- 8) ポートBのビット4

H8/3069F概要4
A/Dコンバータ

- ポットの温度センサ値は増幅して、A/Dコンバータに取り込みます
- アナログ入力はAN₀からの電圧をADDRAレジスタに10ビットのデジタル値に変換します。このとき下3ビットは少数値となります



A/Dデータレジスタ: ADDRA

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
AD9	AD8	AD7	AD6	AD5	AD4	AD3	AD2	AD1	AD0	-	-	-	-	-	-
度数: 0度 ~ 127度							小数点								

H8/3069Fには10ビット8チャンネルのA/Dコンバータがあります。温度の取り込み用には、AN₀のアナログ入力のみ使用します。入力値はA/D変換後は、A/DデータレジスタA (ADDRA) に記憶されます。このレジスタを読み込めば温度として取り込みます。ここではADDRAで有効な10ビットの内上位7ビットを度数、下位3ビットをその小数点値として取り込みます。

しかし、実際はセンサの値をデジタル化した値がそのまま温度の比例値として取り出せることはありません。そのため、テーブル等を用いて変換を行います。この実習ではシミュレータを用いますので、センサからきちんとした温度が取り出せるものとしてプログラムを作成します。

接続ポート表1 出力ポート

- 試作ポットのハードウェア書き込み部に対応したポートの接続レイアウトは以下の通り

ポート	ビット	制御	ポート	ビット	制御
4	0-3	LCDデータビット	A	0	ポンプ:1でオン
	4	LCDRS信号		1	蓋のロック(未使用)
	5	LCDイネーブル信号		2	ヒータ:1でオン
	6	沸騰ランプ:1でオン		3	ヒータ電源:1でオン
	7	保温ランプ:1でオン		7	ブザー:1でオン
7	4	高温ランプ:1でオン			
	5	節約ランプ:1でオン			
	6	ミルクランプ:1でオン			
	7	ロックランプ:1でオン			

2005/12/17

TOPPERSプロジェクト認定

49



試作ポットと接続するためにポート4、ポート7、ポートAの3つのポートを使用しています。これらはデータ出力用に使用します。

ポート4の0から5ビットはLCDを制御するために使用します。ポート4はボード上の2つのLEDと試作ポットの沸騰ランプ、保温ランプに接続しています。ポート7は残り4つのランプに接続しています。

ランプは1をポートに1をセットすると点灯、0をセットすると消灯します。

ポートAは、デバイスを制御します。

ビット0はポンプの制御を行い、0のセットで停止、1のセットでポンプは給湯します。

ビット1は蓋のロックですが、本実習では使用しません。

ビット2はヒータの制御を行い、0のセットでオフ、1のセットでオンとなります。

ヒータ電源がオフの場合は、ヒータは動作しません。

ビット3はヒータ電源の制御で、0で電源カット、1で電源がオンとなります。

ビット7はブザーの制御で、1でブザーオン、0でブザーオフです。

接続ポート表2 入力ポート表

- 試作ポットのハードウェア読み込み部に対応したポートの接続レイアウトは以下の通り

ポート	ビット	制御	ポート	ビット	制御
2	3	第1水位センサ:0でオン	5	3	DIP-SW4:0でオン
	4	第2水位センサ:0でオン	B	0	沸騰ボタン:0でオン
	5	第3水位センサ:0でオン		1	給湯ボタン:0でオン
	6	第4水位センサ:0でオン		2	解除ボタン:0でオン
	7	満水センサ:0でオン		3	タイマボタン:0でオン
5	0	DIP-SW1:0でオン			
	1	DIP-SW2:0でオン		6	保温設定ボタン:0でオン
	2	DIP-SW3:0でオン		7	蓋:0で閉まる

2005/12/17

TOPPERSプロジェクト認定

50



ポート5は、DIP-SWの入力用に使用します。DIP-SW1はモニタの使用の有無に使用します。オンの場合、電源投入でモニタが起動します。オフの場合、モニタは起動せず、タスクID1のタスクを **activate** にします。

ポート2、ポートBは試作ポットの入力データとして使用します。ポート2のビット3から7は水位センサに対応しており、センサがオンの場合、ポートの値はゼロとなります。

ポートBはボタンと蓋の状態に対応しています。

ビット0は沸騰ボタンがオンでゼロ、オフで1となります。

ビット1は給湯ボタンがオンでゼロ、オフで1となります。

ビット2は解除ボタンがオンでゼロ、オフで1となります。

ビット3はタイマボタンがオンでゼロ、オフで1となります。

ビット4は保温設定ボタンがオンでゼロ、オフで1となります。

ビット7は蓋が開いた状態で1、閉めた状態でゼロとなります。

割込み設定

2つを試作ポットで使用しています

- H8/3069FのTOPPERS/JSPとTINETで8つの割込みを使用しています

番号	ラベル	割込み内容
13	IREQ1	ポットのボタンONまたはOFFで割込み
14	IREQ2	蓋の開閉で割込み
17	IREQ5	ネットワークドライバIF_EDで使用する割込み
24	IMIA1	TOPPERS/JSPのシステムタイマ割込み
52	ERI0	シリアルポート0のエラー割込み
53	RXI0	シリアルポート0のデータ受信割込み
54	TXI0	シリアルポート0のデータ送信割込み
56	ERI1	シリアルポート1のエラー割込み
57	RXI1	シリアルポート1のデータ受信割込み
58	TXI1	シリアルポート1のデータ送信割込み

2005/12/17

TOPPERSプロジェクト認定

51



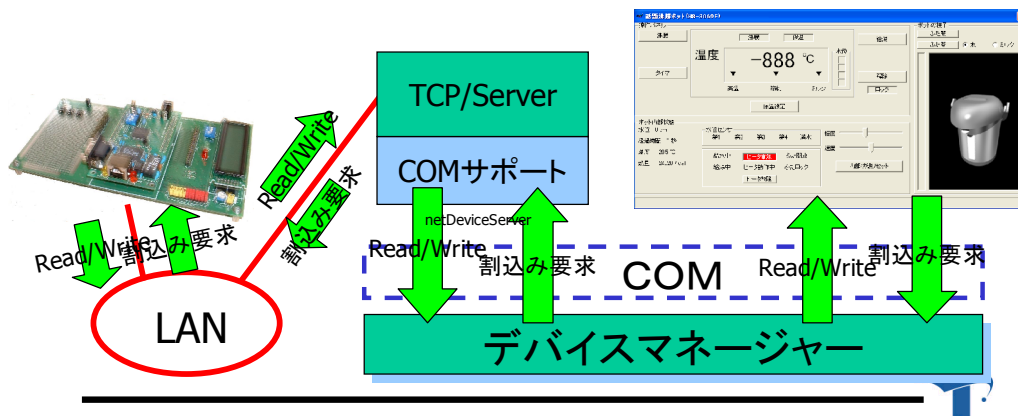
試作ポットには、IREQ1とIREQ2の2つの割込みが対応しています。5つのボタンのどれかが押された場合、または、離された場合にIREQ1割込みが発生します。また、蓋が開閉が発生した場合、IREQ2の割込みが発生します。

その他の割込みはH8/3069Fボードで対応している割込みです。

実習環境の説明

netDeviceServerを用いた開発の説明

- 開発はH8/3069Fを用います
- H8/3069Fボードに、話題沸騰用の拡張ハードウェアが実装されているように制御が可能です



2005/12/17

TOPPERSプロジェクト認定

52

TOPPERS

・netDevice

Windows版TOPPERS/JSP用のデバイスマネージャとネットワークサーバ(NetDeviceServer)をCOMを介して接続したものです。H8/3069FのTOPPERS/JSPで稼働するTINETを使って、このサーバに接続することにより、VB等で作成したシミュレータをH8/3069Fの拡張ハードウェアとして使用することができます。リアルタイムシステム開発実習では、このシステムを使用して話題沸騰ポットシミュレータをH8/3069Fの拡張ハードウェアとして使用し、話題沸騰ポットシステムの開発を行っていただきます。

netDeviceServerはTCPのネットワーク・サーバーとして起動しますが、マルチスレッド化していないため、クライアントはひとつです。

開発環境の説明

1. RTOSを用いたリアルタイムシステム構築
2. 実習課題の説明
3. 開発環境の確認
4. 話題沸騰ポット作成1

2005/12/17

TOPPERSプロジェクト認定

53



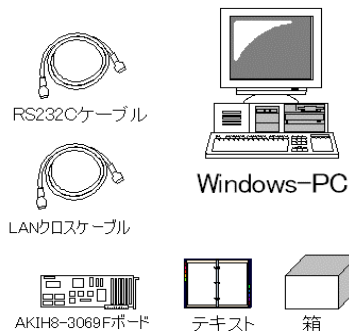
話題沸騰ポット試作版開発のための環境説明を行います。

AKIH8/3069Fのプッシュスイッチ付きに拡張ボードのシミュレーションを行う環境を用いて解説を行います。
このシミュレーション環境があれば、拡張ボードがなくてもカップラメンタイマのデバックが可能です。

開発環境を確認します 機器は接続していますか

- パソコン
- ボード(AKIH8/3069F)
- RS232Cケーブル
- LANクロスケーブル
- テキスト
- 箱

電源 1個
CD-ROM 1枚
等



2005/12/17

TOPPERSプロジェクト認定

54



- パソコン 以下の実装が必要
PizzaFactory2、または、Cygwinインストール済み
エディタ(TeraPat)
通信ソフト(TeraTerm)
- AKIH8/3069Fボード
組み立て済みのもの
ROMモニタがフラッシュROMに書き込み済み
(もなみソフトの組み立て済みボードは書き込み済みです)
- RS232Cケーブル
パソコンとボード間の通信を行います。
モニターの表示やプログラムのダウンロードを行います
- LANクロスケーブル
ボードとパソコン上のシミュレータとの通信をします
- テキスト 本テキスト
- 電源 ボードに電源を供給します。
- CD-ROM PizzaFactory2やJSPでの開発方法が書かれています
- 箱 AKIH8/3069Fボード、ケーブル、CD-ROMを入れる箱

開発環境の接続を確認します

1, 2日の環境と同じです

- PCのLANアダプタとAKIH8/3069FボードのLANアダプタをLANクロスケーブルで接続
- PCのRS232CコネクタとAKIH8/3069FボードのRS232CコネクタをRS232Cケーブルで接続
- TeraTermを立ち上げ、ボードの電源を入れます、モニタのプロンプトが出ることを確認します
- DOS窓を開き、ipconfigコマンドでLANのIPアドレスを確認します

IP=192.168.11.6となっていますか？

2005/12/17

TOPPERSプロジェクト認定

55



・LANケーブル

ボードとパソコンをLANで接続します。パソコンがサーバー、ボードがクライアントとなり、デバイスのアクセス情報を通信します。通信情報により、ボードのプログラムがパソコン上の拡張ボードシミュレータを制御します。クロスケーブルなので1対1の通信となります。

・RS232Cケーブル

ボードとパソコンをRS232Cシリアル手順で接続します。パソコン側は通信ソフトTeraTermにてボードのモニタプログラム(ROM用とタスク用)の制御と状態表示を行います。また、ROMモニタとTeraTermの通信機能を用いて、ビルドしたプログラムのダウンロードを行います。

・IPアドレス

パソコンにIPの設定を確認します。ケーブルが接続され電源が投入され、IPが設定されていればipconfigコマンドにより、設定IPの確認ができます。設定されていない場合は「コントロールパネル」の「ネットワーク接続」で設定してください。netDeviceはIPアドレスでサーバーパソコンを指定します。

確認環境の構築

netDevice版カップラーメンタイマを実行しましょう

- netDevice+AKIH8/3069Fボードを使って“カップラーメン・タイマ”を作
りましょう。
- 仕様は？

電源投入後、実行確認の為にLED3を1秒間隔で点滅。

スイッチ5オンでタイマー起動、オフでタイマー停止、タイムアウト時間は最初は1分

タイマー起動中、スイッチ4オンでタイムアウト時間を1分延長。2回オンでタイムアウト時間は3分。

タイマー動作中確認の為、10秒毎にLED2を点滅、タイムアウト時30秒間点滅して停止。



2005/12/17

TOPPERSプロジェクト認定

56



通常のタイマーはLCDによる時間表示とブザーで構成されています。AKIH8/3069F miniボードでは、LCDとLEDしかデバイスが無い為、LEDとプッシュスイッチをシミュレータで作成し、時間の通知とタイムアウトの通知を行うようにしています。

時間の通知に関しては、LED3を1秒間隔で点滅して通知を行います。タイマーの通知はLED2で行います。タイマーが起動すると10秒毎に点滅させます。タイムアウトの通知はLED2を15秒間点滅させて通知を行います。

タイマーの開始はスイッチ5をオン、タイマーの停止はタイマー5をオフで行います。開始時は1分のタイムアウトが設定され、スイッチ4をオンする毎に1分ずつタイムアウトを延長させます。

netDevice版カップラーメンタイマ1 カーネルライブラリの再構築

- 2日目はTINETに対応するためにMakefileに-DSUPPORT_ETHER定義を追加しました
- ここで、タスクモニターに対応するためにエディタを使って-DMON定義を追加し、再作成を行います
COPTS := \$(COPTS) -DMON -DSUPPORT_ETHER

```
$ cd ./OBJ/AKIH8_3069F/libkernel
$ make clean
$ make libkernel.a
```

2005/12/17

TOPPERSプロジェクト認定

57



タスクモニターのタスク時間を設定する機能を有効にするために、カーネルライブラリを再構築します。`./OBJ/AKIH8_3069F/libkernel`中のMakefileの101行目のCOPTSオプションに-DMONオプションを追加し、再構築を行います。

```
COPTS := $(COPTS) -DSUPPORT_ETHER
```

↓

```
COPTS := $(COPTS) -DMON -DSUPPORT_ETHER
```

再構築は以下のように行います。

```
$ make clean
```

```
$ make libkernel.a
```

netDevice版カップラーメンタイマ2 ビルドしましょう

- 開発環境TIMER3Nに移動します
- 以下の手順でjsp.srecをビルドします

```
$ cd ../TIMER3N
$ ls
akih8_device.c  Makefile  timer3.c  timer3.h  tinet_timer3.cfg
Akih8_lcddevice.c route_cfg.c timer3.cfg tinet_app_config.c
$ make tinet
$ make depend
$ make
```

2005/12/17

TOPPERSプロジェクト認定

58



参考用のカップラーメンタイマプログラムは、OBJ/AKIH8_3069F/TIMER3Nディレクトリ中に格納されています。PizzaFactory2(Cygwin)のコマンド入力で、このディレクトリに移動して、以下のコマンドでビルドを行います。

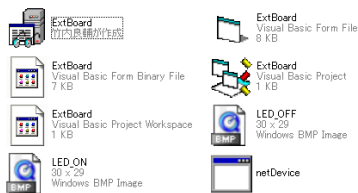
```
$ make tinet
$ make depend
$ make
```

ビルド後、jsp.srecは生成されます。これをダウンロードし、ボード上で実行します。

netDevice版カップラーメンタイマ3 シミュレータの立ち上げ

- スタート→話題沸騰ポット→拡張ボードにて、「ExtBoard」シミュレータを起動してください

ExtBoardは、AKIH8/3069Fの拡張ボードに3つのスイッチをつけたボードをシミュレーションします。拡張ボードがなくても実行可能です。



2005/12/17

TOPPERSプロジェクト認定

59



・AKIH8/3069F版拡張ボードシミュレータ

VB6で作成したAKIH8/3069Fの拡張ボードのシミュレーションを行うプログラムです。このプログラムはWindows版TOPPERS/JSPと共用ですので、Windows版のドライバを作成すればAKIH8/3069Fボードがなくても実行可能です。基本的にポート上からアクセスしているポートは以下の5つです。

H8P2DDR	出力	ポート2(H8P2DR)アクセスモード設定
H8P2PCR	出力	ポート2(H8P2DR)のプルアップ設定
H8P2DR	入力	プッシュボタンの入力ポート
H8P4DDR	出力	ポート4(H8P4DR)のアクセスモード設定
H8P4DR	出力	LEDの設定ポート、AKIH8/3069Fの拡張ボードのLEDと同等のポート設定となっており、netDeviceに接続しない場合はそのまま、拡張ボードのLEDの制御が行えます

実習環境の起動手順1

Deviceマネージャの確認とnetDeviceの起動

- TOPPERSシミュレータ用DeviceManagerのインストールを確認(ツールバーのアイコンを確認)
- DOS窓でExtBoardディレクトリに移動しnetDevice(サーバー)を起動します
- 直接ダブルクリックしても起動できます

```

C:\Documents and Settings\ksc\My Documents\WPIZZAFactory\TOPPERS\NetDevice
Microsoft Windows [Version 5.00.2195]
(C) Copyright 1985-2000 Microsoft Corp.

C:\Documents and Settings\ksc\My Documents\WPIZZAFactory\TOPPERS\NetDevice>ipconfig

Windows 2000 IP Configuration

Ethernet adapter ローカル エリア接続:

    Connection-specific DNS Suffix  . : 
    IP Address. . . . . : 192.168.11.100
    Subnet Mask . . . . . : 255.255.255.0
    Default Gateway . . . . . : 192.168.11.1

C:\Documents and Settings\ksc\My Documents\WPIZZAFactory\TOPPERS\NetDevice>
C:\WPIZZAFactory\TOPPERS\NetDevice>netDevice
NetDeviceServer Ready !!
  
```



← デバイスドライバアイコン

ボードの起動時、DIPS W1がオンになっていることを確認してください

2005/12/17

TOPPERSプロジェクト認定

60



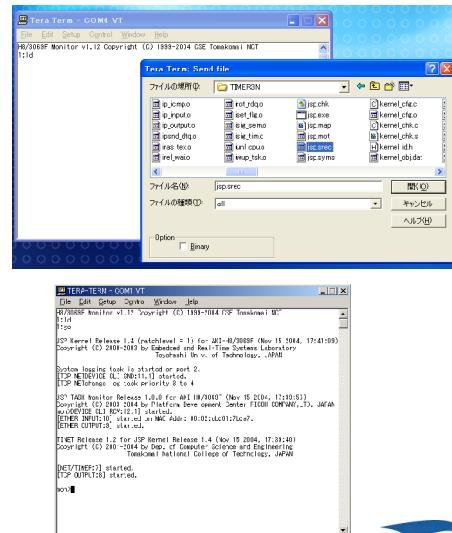
・実習環境

実習環境には、TOPPERS/JSP Windows版のDevice Managerを開発環境にインストールしておく必要があります。インストールしてあれば、H8版の話題沸騰ポットシミュレータを起動することにより、Device Managerアイコンを表示しますので確認できます。

まず、話題沸騰ポットと、netDeviceを起動してください。起動の順番はどちらが先でもかまいません。netDeviceの起動前に、DOS窓からipconfigコマンドの開発環境パソコンのIPアドレスを確認してください。netDeviceはネットワークサーバーとして動作しますので、ボードから話題沸騰ポットシミュレータに接続するために、サーバーのIPアドレスを指定しなければなりません。netDeviceが正常にサーバーとして起動すれば「NetDeviceServer Ready !!」の表示が出ます。

実習環境の起動手順2 プログラムのダウンロードとモニタの起動

- DIP-SW1はオンにしてください
- TeraTermでldコマンド入力後、jsp.srecをTeraTermに重ねるとダウンロードが始まります
- File→Send File ...で、jsp.srecを選択してもダウンロードできます
- ダウンロード後、goコマンドでタスクモニタが実行します



2005/12/17

TOPPERSプロジェクト認定

61



・H8ボードの起動と2つのモニタについて

H8ボードでは、苫小牧高専で作成したH8用のROMモニタがフラッシュROMに書き込まれています。また、netDeviceに簡単に接続するために、初級実装セミナーで使用したタスクモニタが必要となります。ROMモニタとタスクモニタはH8ボードに付属のRS232Cポートを使ってコマンドによって操作ができます。RS232Cが接続したパソコン上でTeraTermを以下の設定で起動します。

Baud rate: 38400bps
Data: 8bits
Parity: none
Stop: 1 bit
Flow control: none

RS232Cが接続していることと、DIP-SW1がオンとなっていることを確認して、H8ボードの電源を入れてください。

TeraTerm上に以下の表示が行われROMモニタが起動しました。

「H8/3069F Monitor v1.12 Copyright © 1999-2004 CSE Tomakomai NCT

1:

」

ビルドしたプログラムをダウンロードするには、ldコマンドを使用します。コマンド入力後、File→Send file...でSRECファイルを指定するとダウンロードが開始されます。ダウンロードが終了すると「1:」プロンプトが再表示されます。ダウンロードしたプログラムを実行するには、goコマンドを使用します。goコマンドにより、TOPPERS/JSP→TINET→タスクモニタが起動します。

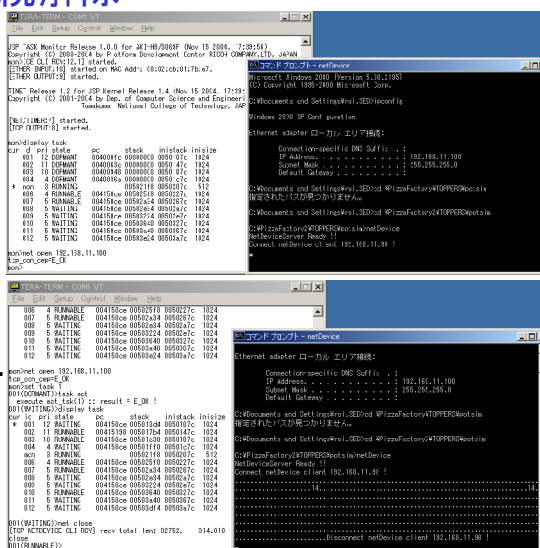
タスクモニタはH8のデバックボードのDIP-SWの1番がオンのとき起動するようにプログラムされていますので、DIP-SWの確認を行ってください。

タスクモニタのプロンプト「mon>」からdisplay task(return)で現状のタスクの状態を表示します。

DORMANT状態のタスクがユーザプログラムで、その他のタスクがTINET用タスクまたはモニタタスクです。

実習環境の起動手順3 netDeviceとの接続と接続解除

- タスクモニタ起動後、
net open <IPアドレス>で、netDeviceに
接続します
- net closeコマンドで
netDeviceとの接続
を解除します
- 非接続の状態でボー
ドをリセットし、新し
いプログラムのダウ
ンロードができます



2005/12/17

TOPPERSプロジェクト認定

62



・netDeviceとの接続

ユーザプログラムが起動すると、デバイスからの読み出し、書き込みを行います。そのため、ユーザタスクはタスクモニタから起動することをお勧めします。モニタ起動時は、自動的にnetDeviceに接続するようにはしていません。そのため、起動時にnet open <IPアドレス> (return)のコマンド入力により、netDeviceと接続する必要があります。接続した場合は「Connect netDevice client<IPアドレス> !」が表示されます。また、接続を解除するには、net close (return)のコマンド入力で解除が行われます。このとき、「Disconnect netDevice client <IPアドレス> !」の表示が出されます。net close入力後、サーバはすぐに非接続となりますが、TINETのクローズタイムアウトの時間の関係上H8ボードは数分の待ち状態となります。プログラムをダウンロードし直すならば、タイムアウトを待たずにリセットキーでリセットし、ROMモニタから再度のプログラムのダウンロードが可能です。netDeviceや話題沸騰ポットシミュレータを再起動する必要はありません。

netDeviceと接続せずにユーザタスクをスタートするとプログラムはH8の実ボードのポートを対象として動作を行います。そのため、期待したデータの入力が行われませんので、正しい動作はしませんが、プログラムが暴走することはありません。

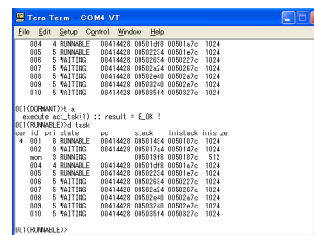
システム構築上の制約

ネットワークにデバイスを代行させることによる制約

- 非タスクコンテキスト上でのネットワーク上のポートアクセスはできません

これはリアルタイムシステム設計上の大きな制約となります

- TCP/IPやクライアントアプリを多くのタスクで実行させるため、実際のシステムよりオーバーヘッドが発生します



カプラーメンタシステムではネットワークに6つのタスクを使用している



2005/12/17

TOPPERSプロジェクト認定

63

TOPPERS

・非タスクコンテキストでのポートアクセス

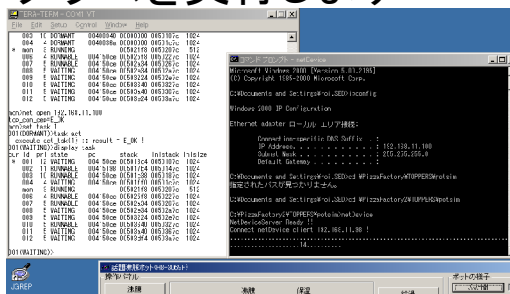
Windows版TOPPERS/JSPでは、タスクコンテキストも非タスクコンテキストもWindowsのスレッドとして実行します。そのため、どちらの場合でもCOMを介してプロセス間通信を行い、デバイスのアクセスが可能です。netDeviceを使用したシステムでは、デバイスのアクセスはTCP/IPのプロトコルスタックを使用してポートのアクセスを行うため、非タスクコンテキストからのポートアクセスはできません。非タスクコンテキストからnetDeviceに対するアクセスを行うと「エラーログ」を発行し、実デバイスへのアクセスを実行します。そのため、正しくシミュレータへのアクセスはできません。

リアルタイムOSでは割込み等の非タスクコンテキストはタスクより、優先して実行されます。そのため、割込み中にネットワークへのアクセスを行っても、割込みが終了した後にネットワーク用のタスクが実行されるため、正しくポートへのアクセスはできません。また、割込みプログラムを擬似的にタスクで実行したとしても、全体としてシステムが複雑となり、教育的なシステムに適さないと判断しました。この制約は、システム設計上大きな制約となります。

実習プログラムの実行 プログラムの実行とエラー時の対応

- netDeviceとの接続後、set task <ID>コマンドで起動タスクに移行し、task activateコマンドでタスクを起動してください。シミュレータをデバイスとしてプログラムを実行します

予期せぬエラー発生場合は、全てのプログラムを終了してください



2005/12/17

TOPPERSプロジェクト認定

64



・ユーザタスクの実行

話題沸騰ポットのサンプル開発環境POT1とPOT2はタスク1が他のタスクを起動するように作成してあります。そのため、モニタのtaskコマンドを使用して、タスク1を起動すれば、他のユーザタスクは起動します。ユーザタスクの実行中にnet close (return)コマンドにてnetDeviceと非接続にしても問題はありません。

プログラムを再ダウンロードするためにnetDeviceと接続中にH8ボードにリセットキーにてボードをリセットしてしまうとnetDeviceはリセットを検知できません。この場合、以下の対応ができます。

- 1) netDeviceを再起動する
- 2) LANケーブルをはずす
- 3) そのまま、ダウンロードしてgoコマンドでタスクモニタを起動してnet openしなおす

但し、3)については再動作可能ですが、保証はしていません。

・その他

再実行時、話題沸騰ポットの「内部状態リセット」ボタンにてシミュレータのリセットを行ってください。

その他、シミュレータやnetDeviceの状態がおかしい場合は、各々のプログラムを再起動してください。

netDeviceの制約事項で非タスクコンテキスト中で、netDevice上のポートのアクセスはできません。

そのため、割込み後、タスクからボタン等をポーリングで読み出すとデレィは発生し誤った読み出しを行うことがあります。シミュレータのボタン等の押下はゆっくりと行ってください。

話題沸騰ポット作成1

1. RTOSを用いたリアルタイムシステム構築
2. 実習課題の説明
3. 開発環境の説明
3. 話題沸騰ポット作成1

2005/12/17

TOPPERSプロジェクト認定

65



話題沸騰ポットシステムを作成しましょう。

話題沸騰ポットとは1 仕様書は読みましたか？

- 「話題沸騰ポット(GOMA-1015型)」仕様書は
SESSAMEのホームページ
からダウンロードできます
<http://www.sesame.jp/>
- 本教材は第3版を対象に
作成しました
- 事前学習で、仕様書を読ん
できましたか？



話題沸騰ポットとは2

カタログ仕様

■「話題沸騰ポット(GOMA-1015型)試作機」のカタログ使用を簡単に説明します

ポット内の水を沸騰・保温する機能

沸騰設定、蓋の閉じた後、3分間の沸騰を行います

ポット内の水を保温する3つの保温モード

沸騰後、高温(98度)、節約(90度)、ミルク(60度)の保温を行えます

給湯のロック機能

給湯ボタンで簡単給湯、ロックボタンで給湯のロックもでき安全

水温と水位の表示

いつでも、水温と水の量が表示されていますので確認が容易です

指定時間にブザーを鳴らす、キッチンタイマ機能

1分毎に時間を設定、ブザーで通知します

2005/12/17

TOPPERSプロジェクト認定

67



「話題沸騰ポット(GOMA-1015型)」要求仕様書から作成した電子ポットの4つの特徴をカタログ化しました。

話題沸騰ポット試作機のカタログ仕様は、一般に目にする電子ポットとほぼ一緒です。

1. ポット内の水を沸騰・保温する機能

沸騰ボタンの押下、蓋を閉じた時、自動的に沸騰モードに移行し3分間沸騰状態を続けカルキ抜きを行います。

2. ポット内の水を保温する3つの保温モード

高温(98度)、節約(90度)、ミルク(60度)の保温温度を選べます。沸騰中、保温ボタンの押下で保温モードに移行します。

3. 給湯のロック機能

給湯ボタンの押下で、ポンプにより自動的に給湯が行えます。また、給湯にはロック機能があり、ロックを行っておけば、お子様が誤って給湯ボタンを押しても、熱湯が出ることがなく安心です。

4. 水温と水位の表示

水温と水位が常時表示されますので、必要な温度の確認と、水が足りない場合すぐにわかります。

5. 指定時間にブザーを鳴らす、キッチンタイマ機能

キッチンタイマ機能がついていますので「カップラーメンタイマ」がなくても、カップラーメンの出来ごろ時間がすぐにわかります。

話題沸騰ポットの構造化分析1

イベントリスト(温度の制御以外)

イベント	スティミュラス	アクション	レスポンス
タイマを開始する	・時間	(1)タイマを起動する (2)残り時間なしでユーザに通知を行う	(1)ブザーオンと残り時間の表示 (2)ブザーオンと残り時間の表示
タイマ時間追加	・時間	現在の残り時間に指定時間を追加	・ブザーオン
保温モード表示	・保温モード	・保温モードを設定する ・温度制御を変更する	・ブザーオン ・モード表示
給湯口のロック	(なし)	・給湯口をロックする	・ブザーオン ・ロックランプ点灯
給湯口のロック解除	(なし)	・給湯口のロックを解除する	・ブザーオン ・ロックランプ消灯
給湯開始	(なし)	・ポット内の水を給湯する	・給湯口から排水(ポンプオン)
給湯終了	(なし)	・ポット内の水の給湯を停止する	・給湯口からの排水を停止(ポンプオフ)
水位の変化	・水位	・表示を行う	・水位表示を更新

2005/12/17

TOPPERSプロジェクト認定

68



イベントリスト(温度制御以外)

・状態を持つ機能(4つ)

タイマの残り時間の管理機能

ブザーのオン時間:ボタンの押下1秒、タイムアウト時3秒、エラー時30秒

水位の監視時間:200MS

ボタンと蓋のポーリング時間:50MS

・タイマ処理

タイマボタン押下:タイマ起動中、残り時間を60秒追加:1秒ブザー。

その他、60秒タイマを起動中:1秒ブザー。

タイマ残り時間終了:3秒ブザー。

タイマ起動中はLCDに残り時間を表示

・給湯処理

給湯ボタン押下:以下の状態以外はポンプをオン。

ロックされているとき、蓋が開いているとき

・保温モード処理

保温ボタン押下:1秒ブザー、押すたびに高温→節約→ミルク→高温・・・の順番にモードを変更
対応したランプを表示する、沸騰中なら保温モードに移行します

話題沸騰ポットの構造化分析2

イベントリスト(温度の制御関係)

イベント	スティミュラス	アクション	レスポンス
沸騰開始	(なし)	(1)水を沸騰させる (2)沸騰終了で水を保温状態にする	(1)・沸騰ランプオン ・保温ランプオフ ・ヒーターオン (2)・沸騰ランプオフ ・保温ランプオン
沸騰中断	(なし)	・水を保温状態にする	・沸騰ランプオフ ・保温ランプオン
満水	・水位	・温度制御を停止する	・沸騰保温ランプオフ ・ヒーターオフ
水なし	・水位	・温度制御を停止する	・沸騰保温ランプオフ ・ヒーターオフ
蓋を閉じる	(なし)	・温度制御可能ならば沸騰を開始する	・沸騰ランプオン ・保温ランプオフ ・ヒーターオフ
蓋を開ける	(なし)	・温度制御を停止する	・沸騰保温ランプオフ ・ヒーターオフ
温度異常	(なし)	・温度制御を停止する ・ブザーを鳴らし警告する	・ブザーオン ・ヒーターオフ

2005/12/17

TOPPERSプロジェクト認定

69



イベントリスト(温度の制御)

・状態を持つ機能(2つ)

サーミスタの検知、目標温度に合わせヒータをオン、オフ:100MS周期

温度エラー状態の検知(高温異常、温度上がらずエラー):100MS周期

・温度の制御

起動時、ヒータ電源を投入する

蓋の閉じた時、沸騰ボタンの押下時、沸騰モードに移行し3分間沸騰する。

その後、保温モードに移行する。

以下の場合、ヒータをオフする。

蓋が開いた時、水位が空、水位が満水

高温異常、温度上がらず異常が発生した場合、ヒータとヒータ電源をオフする。

・温度エラーの検知

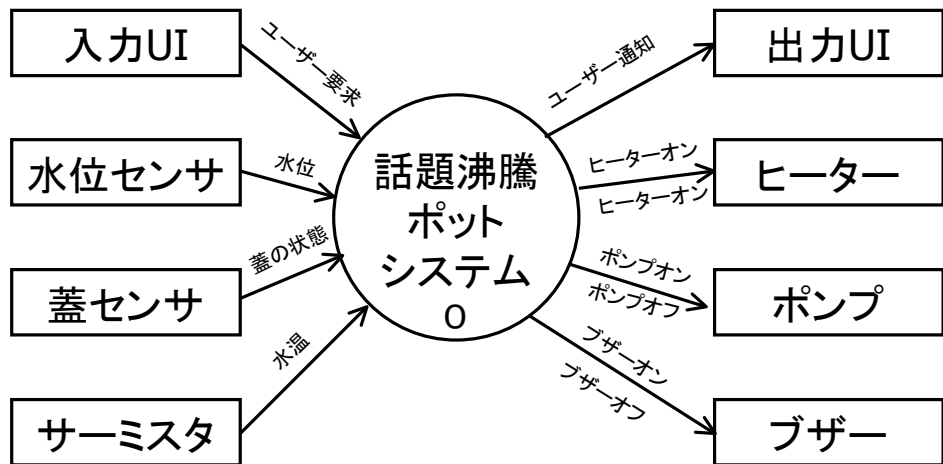
温度上がらず異常:目標温度より5度以上低い状態で、水温が60秒間変化しない場合

高温異常:水温が110度を越えた

・開発の注意

netDeviceの通信制御で50%位のCPU負荷となっています。周期タスクの数が4つを越えると、デッドラインを越えるケースが発生します。リアルタイム性を保障するためにタスクの数を制限しましょう。

話題沸騰ポットの構造化分析3 コンテキストダイアグラム



2005/12/17

TOPPERSプロジェクト認定

70



話題沸騰ポットのコンテキストダイアグラムを作成しました。

コンテキストダイアグラムは、システムとシステムとの情報の交換を行う実体との境界を切り分け、何を分析対象とするか明確にします。

・システム

分析対象のシステム。ここでは話題沸騰ポットシステム

・ターミネータ

他のシステム、ハードウェアなど分析対象とデータ送受信を行う実体。

UI、水位センサ、蓋センサ、サーミスタ、ヒーター、ポンプ、ブザー。

・データフロー

流れていくデータの名前と流れる方向を表す。

ヒーターオン、ヒーターオフ、水位など。

入力と出力UI(ユーザインターフェイス)の詳細内容を以下に示します。

・入力UI

給湯ボタン、解除ボタン、沸騰ボタン、タイマボタン
保温設定ボタン

・出力UI

ランプ: 沸騰、保温、高温、節約、ミルク、ロック
LCD(16×2): 温度表示、水位表示、時間表示

話題沸騰ポットの構造化分析4

データ辞書1

ユーザー要求 = [保温モード | 給湯口のロック | 給湯口のロック解除 | 給湯 | 沸騰 | タイマ時間]
 ユーザー通知 = [保温モード表示 | 温度表示 | 沸騰ランプ点灯 | 沸騰ランプ消灯 | 保温ランプ点灯
 | 保温ランプ消灯 | ロックランプ点灯 | ロックランプ消灯 | 水位表示 | タイマ時間表示]
 タイマ時間表示 = 整数、単位:分、下限:0
 保温モード = [高温モード | 節約モード | ミルクモード]
 温度異常 = [高温異常 | 温度上がらず異常]
 水位 = [0 | 1 | 2 | 3 | 4 | 満水 | 異常]、単位:なし
 水温 = 整数、単位:℃
 蓋の状態 = [開 | 閉]
 給湯 = [開始 | 停止]
 沸騰 = [開始 | 停止]、時間:3分
 保温 = [開始 | 停止]
 高温モード = 沸騰とほとんど同じように使うための保温モード
 節約モード = 消費電力を節約するためのモード
 ミルクモード = 乳児の粉ミルク調乳用として使うためのモード

2005/12/17

TOPPERSプロジェクト認定

71



話題沸騰ポットで使用するデータをデータ辞書に定義しました。

ここでは、BNF記法(Backus-Naur form)で各データの構造を記載しました。

例)

ユーザー要求 = [保温モード | 給湯口のロック | 給湯口のロック解除 | 給湯 | 沸騰 | タイマ時間]

⇒ユーザー要求とは、保温モード、給湯口のロック、給湯口のロック解除、給湯、沸騰、タイマ時間のいずれかです。

話題沸騰ポットの構造化分析5

データ辞書2

保温温度 = [98 | 90 | 60]、単位: °C

温度制御方式 = [PID制御方式 | 温度制御テーブル方式 | 目標温度ON/OFF制御方式]、
監視時間: 100MSごと

高温異常温度 = [110]、単位: °C

高温異常仕様 = 高温異常温度 + 検出周期

温度上がり異常仕様 = 目標温度との差異 + 前回検出温度との差異 + 検出周期

現在保温モード = 保温モード

現在水温 = 水温

現在水位 = 水位

現在経過時間 = 整数、単位: 秒、下限: 0

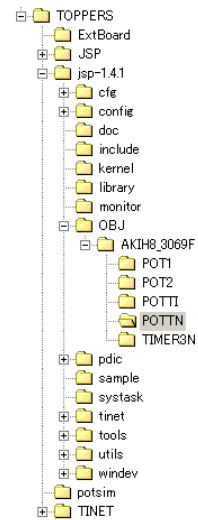
沸騰温度仕様 = 沸騰温度 + 沸騰加熱時間

沸騰温度 = 整数、単位: °C

ロック = [ロック | ロック解除]

話題沸騰ポット開発ワークスペース1 POT1とPOT2

- 開発ディレクトリを確認してください
- POT1はドライバのみの環境で全てのアプリケーションプログラムを作成します
- POT2はドライバ+タイマ機能がすでに実装済みです。温度の制御機能のみを開発してください
- どちらかを選んで開発してください



2005/12/17

TOPPERSプロジェクト認定

73

TOPPERS

話題沸騰ポット試作機を開発するソフトウェア環境はPOT1とPOT2の2つのディレクトリがあります。どちらかを選んで開発してください。

•POT1は先に説明したドライバのソースのみ準備されています。話題沸騰ポットを制御するアプリケーションプログラム(いくつかのタスク)は自分で作成してください。但し、デバイスの初期化のために`event_task`は用意されています。このタスクの起動時にデバイスの初期化`init_pot()`を行っています。デバイスの初期化は`netDevice`がコネクした後に実行しなければならないため、`ATT_INI`等のサービスコールでは初期化はできません。

•POT2はドライバのソースと温度制御以外のイベントリストの機能は実装済みです。POT2を選択した場合は、温度の制御のみを開発してください。

話題沸騰ポットのモード設定を行う`event_task`は出力ユーザインターフェイスに対する設定のみを行っていますので、作成した温度の制御機能とこのタスクとの通信機能は自分で作成し`event_task`を変更してください。

話題沸騰ポット開発ワークスペース2 POT1とPOT2のファイルの説明

- pot1.c/pot1.h (pot2.c/pot2.h)
▲メインのソースとインクルードです、ここに実装を追加してってください
- pot1.cfg (pot2.cfg)
▲ポットのコンフィギュレーションファイル
▲タスク定義、セマフォ定義等を記述します
- akih8pot_device.c/akih8pot_device.h
▲Windows上のポットシミュレータに関するデバイスドライバ(netDevice用のドライバ)
- akih8lcd_device.c
▲AKIH8/3069F上のLCDに関するデバイスドライバ(拡張ボード用のドライバ)

2005/12/17

TOPPERSプロジェクト認定

74



jsp-1.4-1

cfg	カーネルコンフィギュレータ コンフィギュファイルをソースに変換したり、チェックしたりするプログラムが入っています。
config	ターゲット依存部
doc	ドキュメント
include	共通ヘッダファイル μ ITRON4.0関連の宣言が入っています。
kernel	カーネルソースファイル
library	サポートライブラリソースファイル
monitor	タスクモニタソースファイル(話題沸騰ポット用)
OBJ	教材用のプログラム開発環境(話題沸騰ポット用)
pdic	PDIC(デバイスドライバのOS非依存部)
sample	TOPPERS/JSPサンプルプログラム
systask	システムサービスソースファイル
tinnet	TINET(TCP/IPプロトコルスタック)関連ファイル
tools	開発環境依存ディレクトリ
utils	ユーティリティ
windev	Windowsデバイスマネージャ
potsim	VisualBasicプロジェクト(話題沸騰ポット用)
extborad	VisualBasicプロジェクト(カップラーメンタイマ用)

akih8pot_device.cはnetDeviceにアクセスするドライバを集めたソースファイルです。このソースはSILインターフェイスではハードウェアにアクセスしますが、`#include <monsil.h>`にて、タスクモニタ用のSILインターフェイスにアクセスを行います。モニタ用のSILはnetDeviceに接続時はnetDeviceサーバをアクセスし、非接続時は実機のハードウェアにアクセスします。

akih8lcd_device.cは実機のドライバ用のソースファイルで、実機用のSILインターフェイス用のインクルードファイル<sil.h>をインクルードしています。

話題沸騰ポット開発ワークスペース3 POT1とPOT2のファイルの説明

- tinet_app_config.h
 - ▲ネットワークインターフェイスに関する定義
- tinet_pot1.cfg (tinnet_pot2.cfg)
 - ▲TINETコンフィギュレーションファイル
- route_cfg.c
 - ▲ルーティング表

2005/12/17

TOPPERSプロジェクト認定

75



・pot1.c / pot1.h (pot2.c / pot2.h)

最初はイベントタスクのメイン関数だけ実装されています。

ここに他のタスクのメイン関数など、実装を追加してってください。

pot2は、ここにタイマ機能、ボタンイベントの参照機能が実装されています。

・pot1.cfg (pot2.cfg)

ポットのコンフィギュレーションファイル

タスク定義、セマフォ定義等はここに記述します。

・akih8pot_device.c / akih8pot_device.h

Windows上のポットシミュレータに関するデバイスドライバ

この関数を利用すると、LAN経由でポットシミュレータのスイッチの状態、ポットの温度、ポットの水位などを操作することができます。

・akih8lcd_device.c

H8/3069Fボード上のLCDに関するデバイスドライバ

この関数を利用すると、直接LCDを操作することができます。

・tinnet_app_config.h

ネットワークインタフェースに関する定義。

・tinnet_pot1.cfg (tinnet_pot2.cfg)

TINETコンフィギュレーションファイル

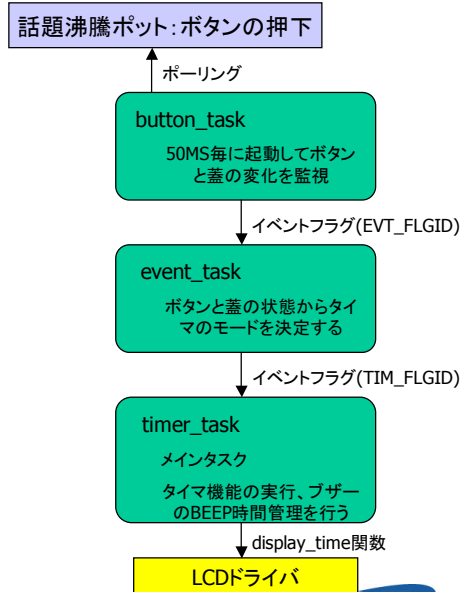
・route_cfg.c

ルーティング表。

話題沸騰ポット開発ワークスペース4

POT2のタイマ機能

- POT2では、ボタンの押下を監視するボタンタスク(button_task)、ボタンの解析を行うイベントタスク(event_task)、タイマとブザーの処理を行うタイマタスク(timer_task)が実装済です
- 3つのタスクはイベントフラグを用いて通信しています



2005/12/17

TOPPERSプロジェクト認定

76



POT2では、ドライバ以外に3つのタスクがサンプル実装されています。

1) button_task

話題沸騰ポットのボタンと蓋の状態を監視するタスク50msの周期で起動し、ボタンと蓋の状態に変化があった場合、event_taskにイベントフラグ(EVT_FLGID)を用いて通知します。

2) event_task

話題沸騰ポットのモードの決定を行います。タイマーの動作要求またはブザーのオン要求があった場合は、timer_taskにイベントフラグ(TIM_FLGID)を用いて実行要求を行います。ポットの温度制御を行いたい場合は、そのタスクの作成とモード設定用の通信機能を拡張する必要があります。

3) timer_task

200msの周期で起動し、タイマー機能、ブザーのオンオフ機能を実行します。

話題沸騰ポット開発済みファームウェア1 ドライバの説明

- ドライバは作成済みで、`akih8pot_device.c`と
`akih8lcd_device.c`の2つのソースに納めてあります
- 入出力のパラメータ設定は`akih8pot_device.h`で定義
してあります
- ドライバは`initial_port`関数による初期化が必要です

関数	入力	出力	機能
<code>initial_port</code>	なし	なし	LEDとLCDの初期化、ポート7とA の初期化、A/Dコンバータの初期 化を行う

2005/12/17

TOPPERSプロジェクト認定

77



ドライバを使用するには、`initial_port()`関数による初期化が必要です。実際のポートの初期化は
`ATT_INI`サービスコールにより、タスクに起動前に行えます。しかし、この開発環境ではネットワークを経由
して初期化が必要なものがあるため、タスクの起動後に行います。

ドライバ、`akih8lcd_device.c`は、拡張ボード上のLCDを制御する関数をまとめたものです。また、
`akih8_device.c`はアプリケーションから呼び出す関数を集めたものです。これらの関数には、`netDevice`
をアクセスして話題沸騰シミュレータと通信するものと、`akih8lcd_device.c`の関数を呼び出してLCD等を
アクセスするものがあります。

話題沸騰ポット開発済みファームウェア2 入力ドライバ

関数	入力	出力	機能
get_switch	なし	ポートBのビット設定	ボタンと蓋の状態を取り出す、1でオン、0でオフ
get_temp	なし	温度 0.125度単位	1/8度単位の温度を取り出す、1度は8
get_level	なし	水位 5ビット	水位を5ビットで取り出す、4ビット目がオンで満水
get_simtime	なし	シミュレーション時間0.1秒単位	話題沸騰シミュレータの時間を0.1秒単位で取り出す

2005/12/17

TOPPERSプロジェクト認定

78



入力用のドライバは上記の4つの関数があります。

get_switch関数は、話題沸騰シミュレータ上のボタンと蓋の状態を取り出します。ポートBのビット設定は以下の通りです。

ポートB設定(IN)

```
#define PB_SWBOIL      0x01      /* SW BOILビット定義 */
#define PB_SWPUMP      0x02      /* SW PUMPビット定義 */
#define PB_SWUNLOCK    0x04      /* SW UNLOCKビット定義 */
#define PB_SWTIMER     0x08      /* SW TIMERビット定義 */
#define PB_SWMODE      0x40      /* SW MODEビット定義 */
#define PB_COVER       0x80      /* COVERビット定義 */
```

ボタンの場合、対応のビット0でオフ、1でオン、蓋の場合、0で開いた状態、1で閉じた状態です。

get_temp関数は、話題沸騰ポットシミュレータの現在の水温を取り出します。1が0.0125度に対応し1度が8、2度が16となります。

get_level関数は、話題沸騰ポットシミュレータの現在の水位を取り出します。0から4までの5ビットがセンサーが検知した値に対応しており、オールビット0ならば水なし、オールビット1ならば満水です。

get_simtime関数は、話題沸騰ポットシミュレータの現在時間を取り出します。0.1秒単位です。TOPPERS/JSPでもシステム時間を持つため、アプリケーションではシステム時間を使用してもかまいません。話題沸騰ポットシミュレータは時間バーがあり、時間速度の設定が可能です。時間バーに対応させるためには、**get_simtime**を使用した方がよいでしょう。

話題沸騰ポット開発済みファームウェア3 出力ドライバ

関数	入力	出力	機能
set_led	PotLedID req	なし	ランプ(LED)の点灯、消灯を設定する
set_control	PotCtlID req	なし	ポンプ、ロック、ヒータ、ヒータ電源、ブザーを設定する
display_temp	temp level	なし	温度と水位をLCDに表示する
display_time	time	なし	タイマ時間を1秒単位に表示する

2005/12/17

TOPPERSプロジェクト認定

79



出力ドライバは上記の4つの関数があります。

set_led関数はランプのオンオフをreqにより設定します。Req=ONで点灯、OFFで消灯です。ランプの指定は以下のenumの指定に従います。(LED_LEVELのReqは表示ビット値です)

```
typedef enum potledid{
```

```
    LED_BOIL=0,           /* BOIL ID */           沸騰ランプ
    LED_KEEPWARM,         /* KEEPWARM ID */       保温ランプ
    LED_HIGH,             /* HIGH ID */           高温ランプ
    LED_ECO,              /* ECO ID */            節約ランプ
    LED_MILK,             /* MILK ID */           ミルクランプ
    LED_LOCK,             /* LOCK ID */           ロックランプ
    LED_LEVEL,            /* LEVEL ID */          レベルランプ(未使用)
    NUM_LED               /* Number of LED ID */
}
```

```
} PotLedID;
```

set_control関数はreqの値によりデバイスの制御を行います。req=ONで起動、OFFで停止です。制御できるデバイスは以下のenumの指定に従います。

```
typedef enum potctlid{
```

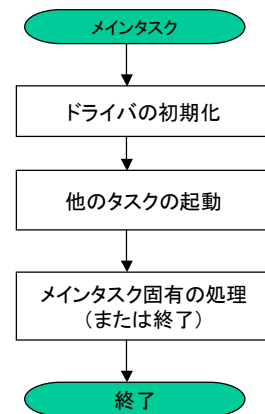
```
    CTL_PUMP,             /* PUMP ID */           ポンプ
    CTL_LOCK,             /* LOCK ID */           ロック(未使用)
    CTL_HEATER,           /* HEATER ID */         ヒータ
    CTL_HEATPOWER,        /* HEATPOWER ID */      ヒータ電源
    CTL_BUZZER,           /* BUZZER ID */         ブザー
    NUM_CTL               /* Number of CTL ID */
}
```

```
} PotCtlID;
```

display_temp関数と**display_time**関数は設定値をLCDに表示します。LCDの細かい制御はドライバで行います。また、time=0の場合、時間は未表示となります。

タイマのメインとしての機能 タスクの起動順序

- POT2では、timer_taskをメインタスクとして、デバイスの初期化、タスクオブジェクトの初期化、他のタスクの起動を行っています
- netDeviceのコネクト後、他のタスクからポートアクセスできるよう、メインタスクから他のタスクを起動するように設計してください



2005/12/17

TOPPERSプロジェクト認定

80



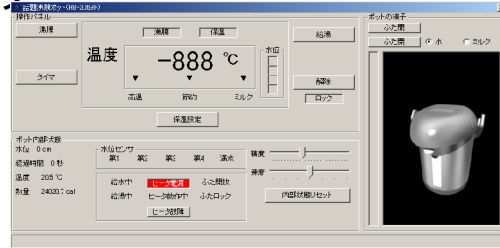
POT2のタイマタスクはメインタスクを兼ねており、システムの初期化を行っています。もちろん、初期化タスクのようなタスクを作り、初期化と他のタスクの起動を行ったあと、終了してもかまいません。しかし、TOPPERS/JSPのように各ソースを静的に設定するRTOSでは、初期化タスクの為のスタック領域は終了後に開放されないため、終了せずにメインタスクの形で他の機能を実行した方がメモリを有効に活用できます。

このシステムでは、タスクモニタが、まず起動します。タスクモニタは`akih8_device.c`中の`need_monitor()`関数を呼び出します。`need_monitor`関数は拡張ボード上のDIP-SWを参照し1がONの場合、TRUEを返します。TRUEが返ればモニタはそのまま継続実行します。DIP-SWの1がOFFの場合、TIMER_TASKを起動して、FALSEを返します。FALSEが返れば、モニタはタスク終了しますのでモニタタスクなしの実行ができます。`need_monitor`関数の書き換えにより最初に起動するタスクのコンフィギュレーションができます。但し、モニタが起動しない場合、netDeviceとの接続はされませんので、話題沸騰ポットシミュレーションの制御ができませんので気をつけてください。

実習を開始してください

- ExtBoardシミュレータを終了させ、話題沸騰ポットシミュレータ(H8/3069F版)を起動してください

- netDeviceサーバはそのまま使用できます



- 実習の開始後、開発のヒントがあります、実習の参考になしてください

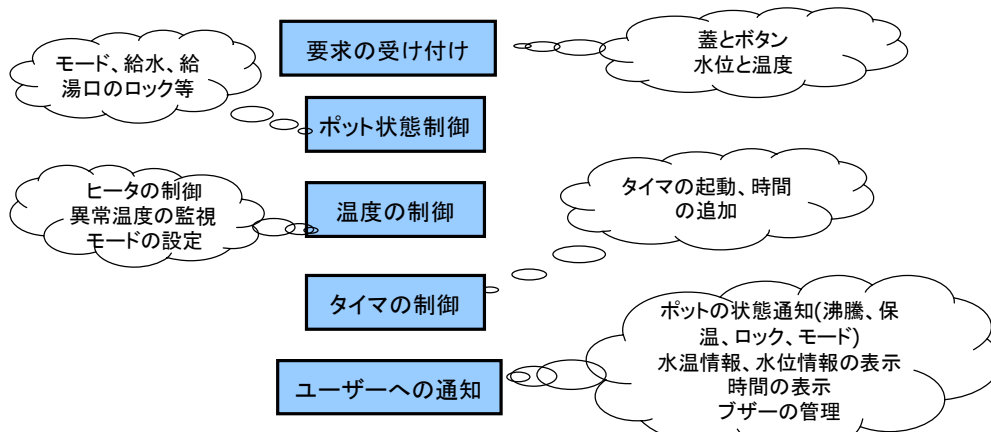
作成のヒント1

1. RTOSを用いたリアルタイムシステム構築
2. 実習課題の説明
3. 開発環境の確認
4. 話題沸騰ポット作成1

作成のヒントを説明します。

タスク分割の指針 やるべき処理を上げてみましょう

■ ポットの機能を整理してみると



何をいつ、どのような順番で行うか複雑になっていませんか？

2005/12/17

TOPPERSプロジェクト認定

83



話題沸騰ポットの機能を整理していくと、以下の5つにまとまります。

- 1) 要求の受付: 蓋とボタンの状態の受付部
- 2) ポット状態制御: 蓋とボタンからポットのモード(沸騰、3つの保温モード)を決定する
給湯、給湯のロックを決定する
タイマに対する動作指示等
- 3) 温度の制御: ポットのモード、現在温度、現在の水温からターゲット温度を決定し
ヒータの制御を行う
異常温度を検出し、異常時、エラーの通知とヒータを停止する
- 4) タイマの制御: タイマの動作要求があった後、残り時間の計算、表示、残り時間の終了後、ブザー要求を行う等の制御を行う
- 5) ユーザへの通知:
ポットの状態通知: 沸騰、保温、ロック等
現在の水温、水位の表示
時間の表示、ブザーの管理

タスク分割を行う1 タスク分割の指針

■ 何に着目してタスクを分割すれば良いのか？

【他の処理に比べ重要である処理】

- ・ 終了する期限(デッドライン)が決められているもの。
→ 温度の制御(温度の監視、ヒータの制御)

【外部からのイベントを受け取るもの】

- ・ デバイスの監視処理
→ 外部の状態が変化した時だけイベントをもらいたい
→ 蓋とボタンの監視

【並行に実行することで、効率の向上が見込めるもの】

- ・ 異なるデバイスの制御
→ ポットの状態制御、温度の制御、タイマの制御

上の内容でタスク分割できそうだ。

2005/12/17

TOPPERSプロジェクト認定

84



タスクの分割指針(あくまでサンプル分割です)

- ・他の処理に比べ重要である処理:ポットとして動作する基本機能

終了する期限(デッドライン)が決められているもの

温度の制御→ヒータの制御(100MS周期):100MS以内にターゲット温度を決定し

ヒータのオン、オフを行う(従って、ヒータの制御周期は100MS単位となる)

→温度の監視(100MS周期):100MS周期で温度の異常を検出する

正しく行われないと、火事になる場合もあり、特に重要

- ・外部からイベントを受け取るもの

デバイスの管理機能

→蓋とボタンの状態監視:蓋とボタンの変化を監視し、変化があれば、
他のタスク等に伝達する必要がある。

レスポンスを考慮して50MS周期

- ・並行に実行することで、効率の向上が見込めるもの

異なるデバイスの制御

→ポットの状態制御:沸騰、保温、給湯、給湯禁止等の判定

タイマ起動、タイマ延長等

→温度の制御:うえを参照

→タイマの制御:残り時間の制御

→ブザーの制御:ブザーを一定時間鳴らす

タスクを分割を行う2

タスクの分割例

■ 実際にタスク分割してみました

タスク名	タスク概要	優先度
EVENT_TASK	ポットの状態(沸騰中、保温中等)の制御。 状態の通知。	中
HEATER_TASK	水位および、水温からヒータのON,OFF制御を行う。	高
ERROR_TASK	水温を監視し、温度の異常が発生していないか監視する。	高
TIMER_TASK	・キッチンタイマの制御を行う。 ・ブザーの制御	低
BUTTON_TASK	ユーザーからの操作(ボタン押下、蓋の開閉)を監視する。	中

今回の例では、POT2のタイマタスクを参考にしてTIMER_TASK内にブザーの制御が組み込まれている。
ブザーの制御は別タスクへと分けるとより分かりやすい。

2005/12/17

TOPPERSプロジェクト認定

85



サンプルタスク分割

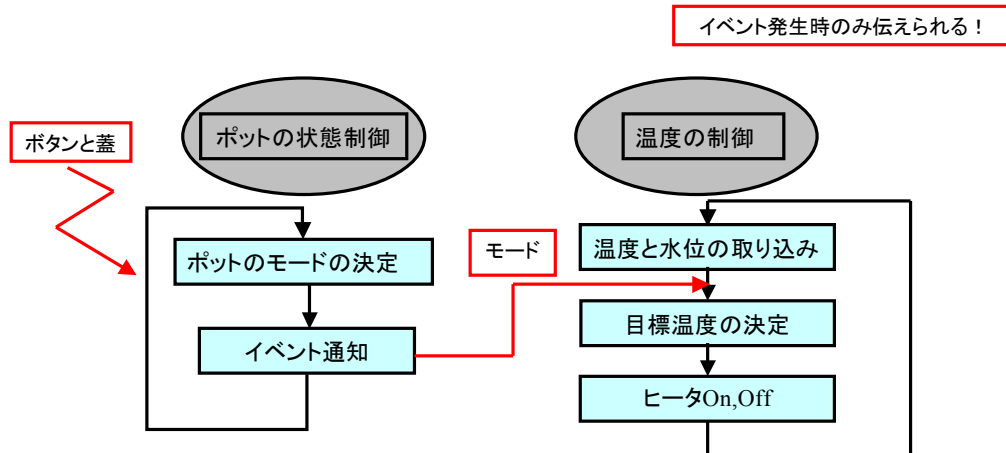
サンプルとしてタスク分割してみました

1. EVENT_TASK: pot2.cにevent_taskとして実装されている。HEATER_TASKと通信を行うための拡張が必要
2. HEATER_TASK: 未実装
3. ERROR_TASK: 未実装
4. TIMER_TASK: pot2.cにtimer_taskとして実装済み、タイマの機能とブザーの機能をもつ
5. BUTTON_TASK: pot2.cにbutton_taskとして実装済み

タスク分割を行う3

タスク分割の指針

- ポットの状態制御タスク(EVENT_TASK)と温度の制御タスク(HEATER_TASK)を図にしてみました



2005/12/17

TOPPERSプロジェクト認定

86



EVENT_TASKとHEATER_TASK間の通信を考えます。

HEATER_TASKは100MSごとに水温と水位を取り込み目標温度を決定し、100MS周期のヒータのオン、オフを決定します。従って、水位(満タン、水なし)のヒータの停止機能はHEATER_TASKが判定を行います。EVENT_TASKは蓋とボタンの状態からモードを決定し、HEATER_TASKにかかわる制御をイベントフラグで伝達すればよいので伝達イベントの例として以下のものを考えます。

- 1) HEAT_CHECK: 状態の変化(種々のポットの状態変化があった)
- 2) HEAT_BOIL + HEAT_BOILCHG: 沸騰要求
HEAT_BOILCHGのみ: 沸騰要求の変化、沸騰中は保温へ
- 3) HEAT_ERROR: 温度異常発生

プログラムの記述

コンフィギュレーションファイルの記述

- タスクの生成記述をコンフィギュレーションファイルに追加します

温度の制御タスク	ヒータを使用し、水の温度制御を行う。
ポットの状態制御タスク	ポットのシステムに要求されるイベントを監視する。



ソースコードではどのように表現されるのか？

po1.cfgまたはpot2.cfgの中にタスクを生成することを宣言

```
CRE_TSK
( HEATER_TASK , {TA_HLNG , (VP_INT) 0 , heater_task , HEATER_PRIORITY , STACK_SIZE , NULL })

CRE_TSK
( EVENT_TASK , { TA_HLNG , 0 , event_task , EVENT_PRIORITY , STACK_SIZE , NULL })
```

```
CRE_TSK
( タスク名 , {TA_HLNG , (VP_INT) 0 , heater_task , タスクの優先度 , STACK_SIZE , NULL })
```

2005/12/17

TOPPERSプロジェクト認定

87



HEATER_TASKを新規作成する場合は、タスク関数の作成とコンフィギュレーションファイルへCRE_TASK静的APIを使つての登録が必要です。

ソースファイルを別にする場合は、これに加えて、Makefileの修正が必要です。例えば、heater.cという新規のソースファイルとして追加する場合は、180行目のUTASK_COBJSにオブジェクト名で追加してください。

```
UTASK_DIR = $(SRCDIR)/library:$(NETAPP_DIR)
```

```
UTASK_ASMOBS =
```

```
UTASK_COBJS = $(UNAME).o akih8pot_device.o akih8lcd_device.o $(NETAPP_COBJS)
```

```
UTASK_CFLAGS =
```

```
UTASK_LIBS =
```

↓

```
UTASK_DIR = $(SRCDIR)/library:$(NETAPP_DIR)
```

```
UTASK_ASMOBS =
```

```
UTASK_COBJS = $(UNAME).o heater.o akih8pot_device.o akih8lcd_device.o $(NETAPP_COBJS)
```

```
UTASK_CFLAGS =
```

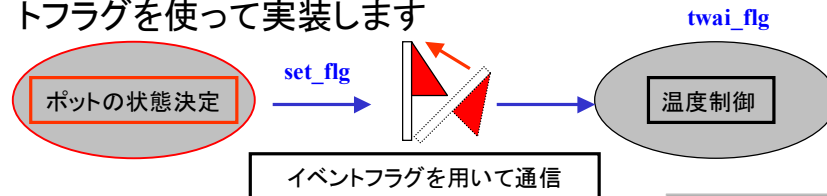
```
UTASK_LIBS =
```

その後、make dependとmakeをします。

タスク間の通信

イベントフラグを使ったタスク間通信①

- ポットの状態タスクと温度の制御タスク間の通信をイベントフラグを使って実装します



//フラグの作成(**.cfgファイルに一度記述すればOK)

```
CRE_FLG ( HEAT_FLGID, { TA_TFIFO | TA_WSGL }, 0 );
```

```
CRE_FLG ( イベントの種類 , { ( フラグの属性 ), フラグのビットパターン初期値 } );
```

//ポットの状態タスクから温度制御タスクへの通知(沸騰モードへの遷移を伝える場合)

```
set_flg ( HEAT_FLGID, HEAT_BOIL );
```

```
set_flg ( フラグのID , 伝えたい情報 );
```

//温度制御側でフラグが立つのを待つ。

```
twai_flg ( HEAT_FLGID, EVT_HEAT, TWF_ORW, &flgptn, HEATER_CYCLE );
```

```
twai_flg ( フラグのID , イベントの種類, 待ちモード, &flgptn, タイムアウト周期 );
```

詳しくは μ ITRON の仕様書を参考

2005/12/17

TOPPERSプロジェクト認定

88



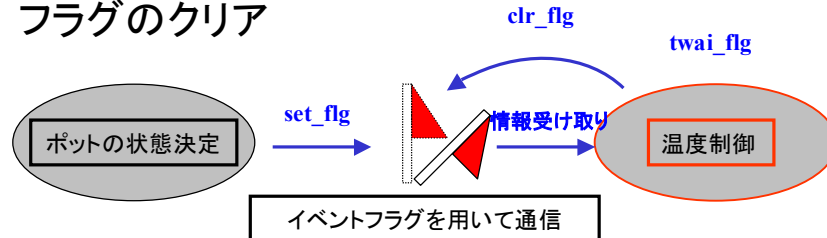
/* 沸騰ボタン押下 */

```
if(!(event & BUTTON_BOIL) && (sw & BUTTON_BOIL)){
    set_flg(TIM_FLGID, TIM_1SEC_BUZZER);
    set_flg(HEAT_FLGID, HEAT_BOIL);
}
```

タスク間の通信

イベントフラグを使ったタスク間通信②

■ フラグのクリア



//温度制御側で受け取ったフラグをクリアする。

```
clr_flg ( HEAT_FLGID, ~flgptn );
clr_flg ( フラグのID, クリアするビットパターン );
```

2005/12/17

TOPPERSプロジェクト認定

89



```
ercd = twai_flg(HEAT_FLGID, EVT_HEAT, TWF_ORW, &flgptn, HEATER_CYCLE);
if(ercd == E_OK && state != STATE_ERROR_HEATER){
    clr_flg(HEAT_FLGID, ~flgptn);
    if(flgptn & HEAT_ERROR){
```

set_flgにてflgptnの該当ビットを立てる。

HEAT_BOILの場合defineの定義からflgptn = 0x02 (0000 0010)

よってclr_flgにおけるクリアのビットパターンは下のようになる。

~flgptn = 1111 1101

flgptnと~flgptnの論理積により、HEAT_BOILのビットがクリアされる。

タスク間の通信

イベントフラグを使ったタスク間通信③

■ 2つのタスク間通信の実装例

//イベントタスク側

フラグ待ち

```
for(;;){
    ercd = wai_flg(EVT_FLGID, EVT_EVENT, TWF_ORW, &flgptn);
    //他のイベントによる処理
    . . . . .
    //沸騰ボタン押下
    set_flg(HEAT_FLGID, HEAT_BOIL);
}
```

//ヒータタスク側

フラグ待ち

```
for(;;){
    ercd = twai_flg(HEAT_FLGID, EVT_HEAT, TWF_ORW, &flgptn, HEATER_CYCLE);
    if(ercd == E_OK && state != STATE_ERROR_HEATER){
        clr_flg(HEAT_FLGID, ~flgptn);
        . . . . .
    }
}
```

フラグクリア

作成のヒント2

1. RTOSを用いたリアルタイムシステム構築
2. 実習課題の説明
3. 開発環境の確認
4. 話題沸騰ポット作成1

温度の制御タスクの実装例1

名称はヒータタスク(heater_task)

■ タスクとソースコードの対応

ヒータタスク	ヒータを使用し、水の温度制御を行う。	
--------	--------------------	--

ヒータタスク内に実際の処理を記述する。

```
void heater_task(VP exinf)
{
    set_control(CTL_HEATPOWER, ON);          //ヒータ電源オン
    for (;;) {                                //無限のループ
        モードの取り込み、または、監視時間のタイムアウト
        モードと水位から温度の決定
        現在温度の比較
        ヒータの加熱、加熱停止
        温度と水位の表示
    }
}
```

いきなりソースコードを書き始めるのではなく、状態遷移図、フローチャート等の図を元に実装しましょう。

2005/12/17

TOPPERSプロジェクト認定

92

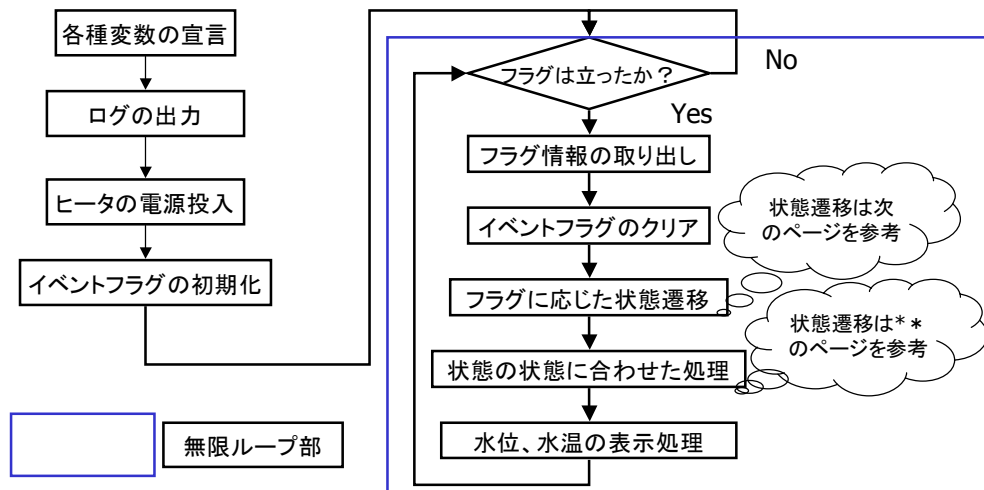


実際、ソースコードしか残っていない場合、その後の仕様変更や障害発生時に保守が大変になる場合があります。

(本人の意図が読み取れない場合も多い。)

そのため、開発の成果物は初めて担当する方でもメンテナンスが行いやすいように、設計のドキュメントを残すこと、およびドキュメントの更新が重要になってきます。

温度の制御タスクの実装例2 フローチャートで書くと



2005/12/17

TOPPERSプロジェクト認定

93



< 通信用のイベントフラグの例 >

```

/*
 * HEAT_FLGIDのビット定義
 */
#define HEAT_CHECK      0x01      /* 状態変更 */
#define HEAT_BOIL       0x02      /* 沸騰指定 */
#define HEAT_BOILCHG    0x04      /* 沸騰ボタン */
#define HEAT_ERROR      0x10      /* エラー発生 */

#define EVT_HEAT (HEAT_CHECK|HEAT_BOIL|HEAT_BOILCHG|HEAT_ERROR)
  
```

< 状態のenum設定の例 >

```

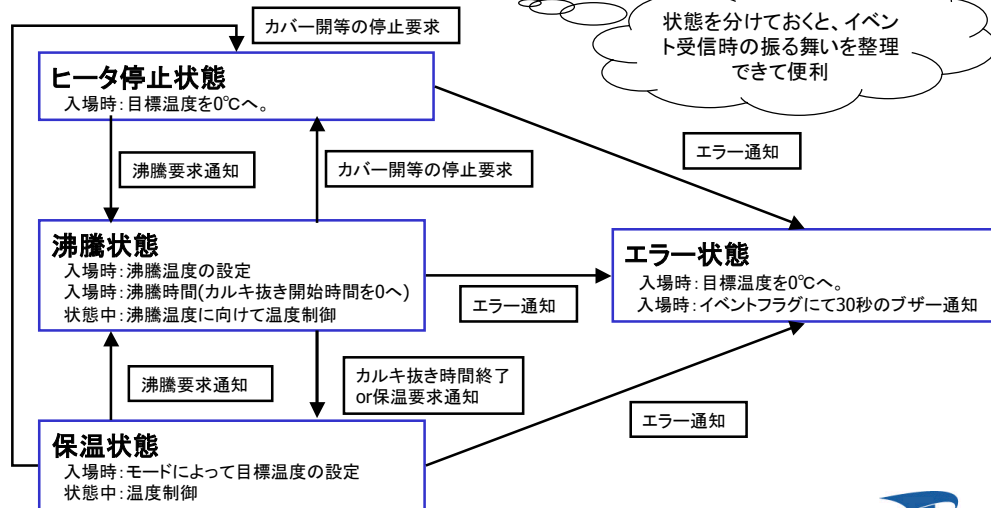
/*
 * ヒータタスクの状態
 */
typedef enum heat_state {
    STATE_STOP_HEATER,      /* ヒータ停止状態 */
    STATE_BOIL_HEATER,      /* 沸騰状態 */
    STATE_KEEP_HEATER,      /* 保温状態 */
    STATE_ERROR_HEATER      /* エラー状態 */
} HEAT_STATE;

void heater_task(VP_INT exinf)
{
    SYSTEM    boil_time;      システム時間型、沸騰時間として使用
    ER        ercd;           エラー情報型、エラー情報(イベントフラグ使用時に使う)
    FLGPTN    flgptn;         フラグパターン型、受け取ったイベント値を保持
    HEAT_STATE state = STATE_STOP_HEATER;  enum型、heater_taskの状態を保持する関数
    H         oldtemp;         100ms前の温度(1度=8)を保持
    UB        potlevel, oldlevel;  現在の水位と100ms前の水位を保持する変数
    int        count;          ループの回数
  
```

温度の制御タスクの実装例3

温度の制御タスクの状態

■ この実装例では4つの状態を設定



2005/12/17

TOPPERSプロジェクト認定

94



< サンプルリスト続き1 >

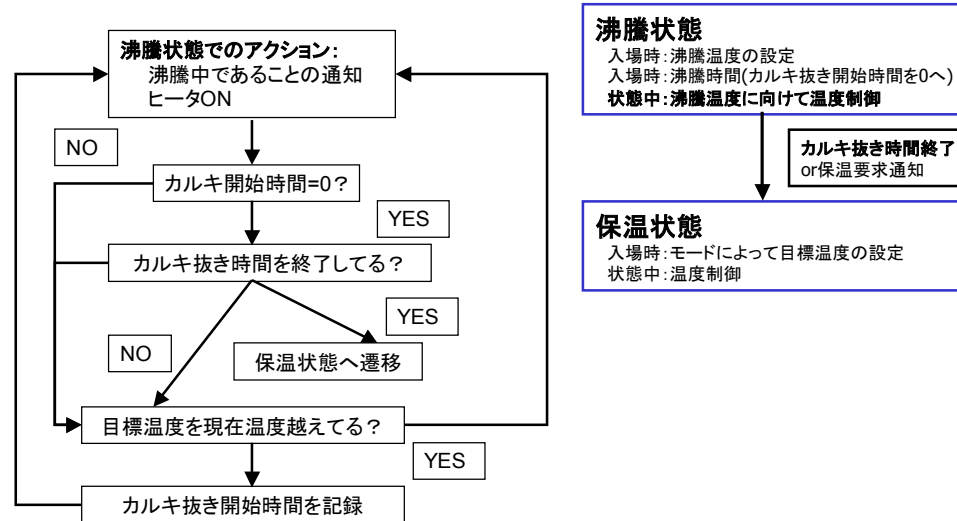
```

syslog(LOG_INFO, "Pot heater task starts (exinf = %d).", exinf);
pottemp = get_temp(); /* 初期状態の温度設定 */
oldtemp = pottemp - 1;
potlevel = get_level(); /* 初期状態の水位の設定 */
oldlevel = potlevel - 1;
count = 0; /* カウンタの初期化 */
set_control(CTL_HEATPOWER, ON); /* ヒータ電源投入 */
clr_flg(HEAT_FLGID, 0);
for(;;){
    ercd = twai_flg(HEAT_FLGID, EVT_HEAT, TWF_ORW, &flgptn, HEATER_CYCLE);
    if(ercd == E_OK && state != STATE_ERROR_HEATER){
        clr_flg(HEAT_FLGID, ~flgptn);
        //以下が状態遷移箇所
        if(flgptn & HEAT_ERROR){
            state = STATE_ERROR_HEATER;
            targettemp = 0;
            set_flg(TIM_FLGID, TIM_30SEC_BUZZER);
        }
        else if(potlevel > 15 || potlevel == 0 || (potstate & COVER_OPEN)){
            state = STATE_STOP_HEATER;
            targettemp = 0;
        }
        else if(state != STATE_BOIL_HEATER){
            int i;
            i = (potstate & MODE_MASK) - 1;
            if((flgptn & HEAT_BOIL)){
                targettemp = temp_table[i][0];
                state = STATE_BOIL_HEATER;
                boil_time = 0;
            }
            else{
                targettemp = temp_table[i][1];
                state = STATE_KEEP_HEATER;
            }
        }
    }
}
}

```

温度の制御タスクの実装例4

沸騰状態



2005/12/17

TOPPERSプロジェクト認定

95



<サンプルソースリスト続き2>

```

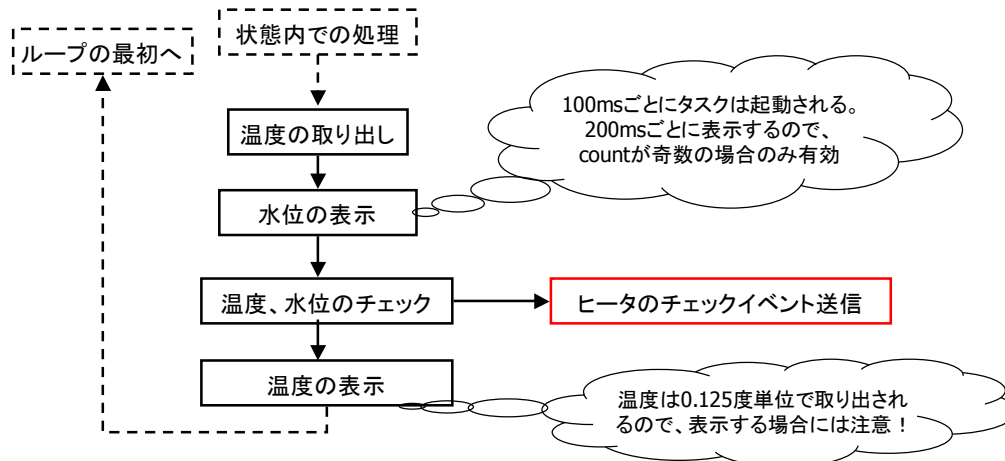
switch(state){
case STATE_STOP_HEATER:                                /* ヒータ停止状態 */
    set_control(CTL_HEATER, OFF);
    set_led(LED_BOIL, OFF);
    set_led(LED_KEEPWARM, OFF);
    break;
case STATE_BOIL_HEATER:                                /* 沸騰状態 */
    set_led(LED_BOIL, ON);
    set_led(LED_KEEPWARM, OFF);
    set_control(CTL_HEATER, ON);
    if(boil_time){
        if((get_simtime() - boil_time) >= 30*60){
            state = STATE_KEEP_HEATER;
            set_flg(HEAT_FLGID, HEAT_CHECK);
        }
    }
    else if(pottemp >= targettemp)
        boil_time = get_simtime();
    break;
case STATE_KEEP_HEATER:                                /* 保温状態 */
    set_led(LED_BOIL, OFF);
    set_led(LED_KEEPWARM, ON);
    if(pottemp > targettemp)
        set_control(CTL_HEATER, OFF);
    else if(pottemp < targettemp)
        set_control(CTL_HEATER, ON);
    break;
}

```

温度の制御タスクの実装例5

状態によらない処理

■ 温度の水位の表示



2005/12/17

TOPPERSプロジェクト認定

96



<サンプルソースリスト続き3>

```

default:
    set_control(CTL_HEATER, OFF);
    set_control(CTL_HEATPOWER, OFF);
    set_led(LED_BOIL, OFF);
    set_led(LED_KEEPWARM, OFF);
    break;
}

/* エラー状態 */

pottemp = get_temp();
count++;
/* 温度の取り出し */
/* カウントアップ */
/* 200ms毎の処理 */
if(count & 1){
    potlevel = get_level();
    set_led(LED_LEVEL, potlevel);
}

if(pottemp != oldtemp || potlevel != oldlevel){
    set_flg(HEAT_FLGID, HEAT_CHECK);
    display_temp((pottemp+5)>>3, potlevel);
    oldtemp = pottemp;
    oldlevel = potlevel;
}
}
}

```