

TOPPERS中級実装セミナー

(基本編:H8/3069F版)

1日目

TOPPERSプロジェクト
教育ワーキング・グループ

2005/12/19

TOPPERSプロジェクト認定

1



本ドキュメントに関して

1. 著作権に関しての表記

＜TOPPERS中級実装セミナー（基本編：HS-3069F版）1日目＞

Copyright (C) 2005 by 高田広章 名古屋大学

Copyright (C) 2005 by 山本雅基 名古屋大学

Copyright (C) 2005 by 本田晋也 名古屋大学

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。
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。

スケジュール

■ 1日目

- | | |
|-------------------|-------|
| 1. 開発環境の確認 | 0.5時間 |
| 2. 基本プログラミング | 1.5時間 |
| 3. ITRON仕様の概要 | 0.5時間 |
| 4. ITRON API解説 | 2.5時間 |
| 5. カップラーメンタイマーの開発 | 1.0時間 |

■ 2日目

- | | |
|-----------------------|-------|
| 1. TCP/IPの概要 | 1.0時間 |
| 2. ITRON TCP/IP API仕様 | 1.0時間 |
| 3. TINET (TCP/IPスタック) | 0.5時間 |
| 4. TCPサーバープログラミング | 2.0時間 |
| 5. TCPクライアントプログラミング | 1.5時間 |

セミナー 1-2日目の目標

- AKI-H8/3069Fボードの設定と、TOPPERS/JSPカーネルの構築方法とサンプルプログラムの実行方法の習得
- マイコンボード上のデバイス进行操作するプログラムのプログラミング方法の習得
- ITRON仕様のAPIを用いたマルチタスクプログラミングの習得
- TCP/IPの基本的な概念の理解
- ITRON TCP/IP API仕様のAPIを用いたネットワークプログラミングの習得

開発環境の確認

1. [配布物の確認](#)
2. 開発対象の説明
3. 開発環境の説明

配布物の確認：テキスト、ボード、ケーブル

- パソコン(Windows-PC)
- テキスト
- PizzaFactory2
 - AKI-H8/3069Fボード + I/Oボード
 - ボード付属マニュアル・CD-ROM
 - 液晶基板
 - RS232Cケーブル
 - 10base-T クロスケーブル
 - AKI-H8/3069Fボード用電源

2005/12/19

TOPPERSプロジェクト認定

6



- ・パソコン 以下のツールが必要
PizzaFactory2,(CygwinでもOK)
(Cygwinで開発する場合は、AKI-H8 3069Fボード付属マニュアルとCD-ROMからインストールできます)
- エディタ
- 通信ソフト
- ・テキスト 本テキスト、 μ ITRON4.0仕様書抜粋
- ・マイコンボード AKI-H8/3069FとI/Oボード、液晶基盤を装着表示確認を行ってください。
- ・RS232Cケーブル
パソコンとボード間の通信を行います。
簡易モニタの表示やユーザープログラムのダウンロードに用います。
- ・10base-Tクロスケーブル
2日目からの実習でパソコンとのデータ通信用に使います。
- ・AKI-H8/3069Fボード用電源
ボードに電源を供給します。

配布物の確認：プログラムディレクトリ

- 開発プログラムはenvironment(プログラムディレクトリ)以下
- コンフィギュレータはWindowsで実行可能なexeが付属
- Windows以外の環境で使用する場合は、コンフィギュレータの再構築が必要
- ディレクトリ構成(environment以下)
 - jsp-1.4.1
 - TOPPERS/JSPカーネル 1.4.1+ TINET 1.2.3 ソースコード
 - 演習用プログラム(1日目から4日目共通)
 - tcp_server
 - Windows用のサーバープログラム
 - h8mon
 - H8用簡易モニタのソースコード

2005/12/19

TOPPERSプロジェクト認定

7



開発プログラムはディレクトリ「environment」以下にあります。サブディレクトリの内容を説明します。

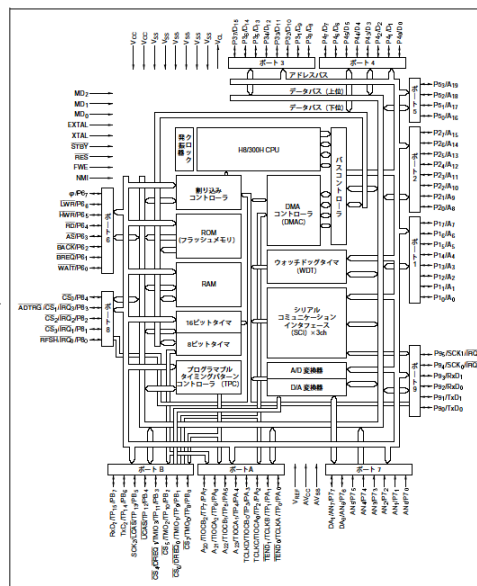
- jsp-1.4.1 TOPPERS/JSPカーネル1.4.1+TINET1.2.3ソースコードに
 演習用のプログラムを追加した開発環境
- h8mon AKIH8/3069Fに書き込むROMモニタのソースコード、書き込みフォーマット
 データ、ドキュメント
- tcp_server 2日目のセミナーで使用するパソコン上のTCPサーバープログラム
- netDevice 3, 4日目のセミナーで使用するネットデバイス用のサーバープログラム
- ExtBoard 3日目のセミナーで使用、
 カップラーメンシミュレータ: 拡張ボード上にボタンを3つ追加する
- potsim3069 3, 4日目のセミナーで使用
 話題沸騰ポットシミュレータAKIH8/3069F版
- DeviceManager
 Windows版デバイスマネージャーの実行形式
 netDeviceを使用するために使用

開発環境の確認

1. 配布物の確認
- [2. 開発対象の説明](#)
3. 開発環境の説明

開発対象：プロセッサ(H8/3069F)

- ルネサス製16bitマイコン
- 最大動作周波数：25MHz
- オンチップFLASH：512KB
- オンチップRAM：16KB
- 周辺回路
 - DMAコントローラ 4チャンネル
 - 16bitタイマ 3チャンネル
 - SCI 3チャンネル
 - 10bit A/D 8チャンネル
 - 8bit D/A 2チャンネル



2005/12/19

TOPPERSプロジェクト認定

9



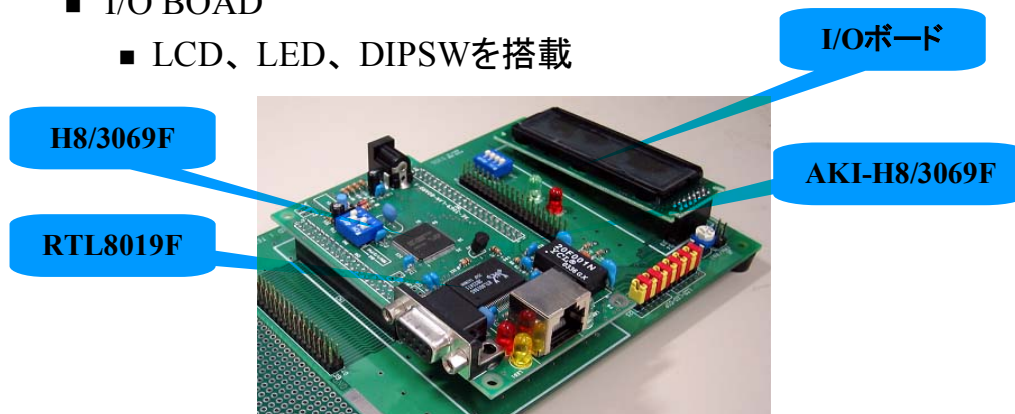
中級実装セミナーでは、ルネサス製の16ビットマイコンH8/3069Fを使います。

最大動作周期数は25MHzですが、AKIH8/3069Fボードでは20MHzで使用しています。オンチップ上に512KBのFLASHメモリと16KBのスタティックRAMを内蔵しています。AKIH8/3069Fボード上に外付けとして2MBのSDRAMが実装されています。本実習では、512KBのFLASHメモリに、H8用簡易モータを書き込み、このモータを使って、SDRAMにTOPPERS/JSPと開発したアプリケーションをダウンロードして、プログラムの検証を行います。

周辺回路は、基本的にパラレルI/Oポート以外は使用しません。

開発対象：ボード(AKI-H8/3069F)

- H8/3069F搭載
- NE2000互換ネットワークコントローラ、RTL8019AS搭載
- H8/3069F内蔵フラッシュROM用ライター回路内蔵
- I/O BOAD
 - LCD、LED、DIPSWを搭載



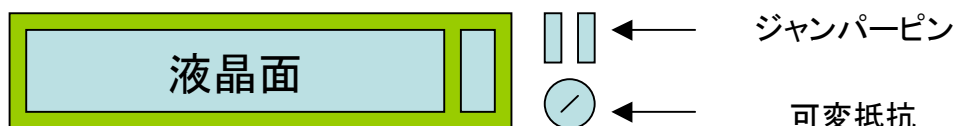
2005/12/19

TOPPERSプロジェクト認定

10



AKIH8/3069Fボード、I/Oボードの概観図です。I/Oボードには液晶基盤を装着します。液晶を正しく表示させるには、ジャンパーピンを装着し、可変抵抗を適切な値へ設定変更する必要があります。後の実習で確認を行います。

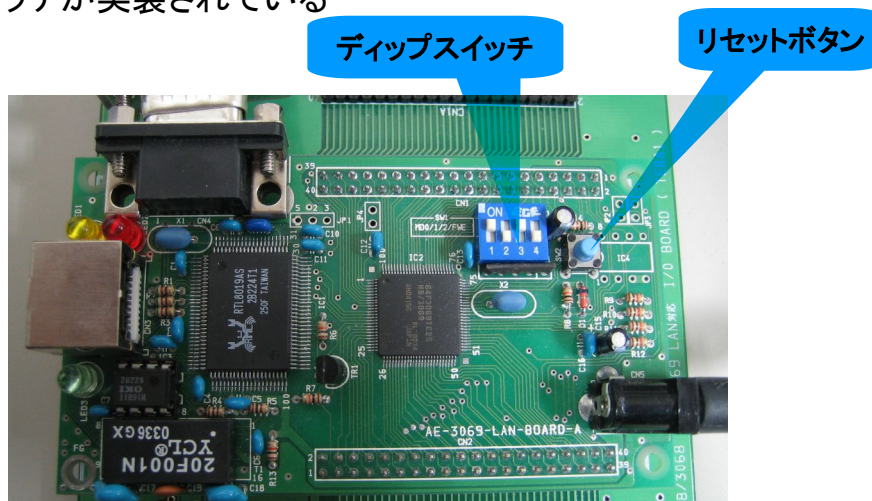


AKIH8/3069Fボードには、NE2000互換のRTL8019ASネットワークコントローラとLANソケット、シリアルドライバと9ピンのD-SUBコネクタが搭載されています。これらの制御はH8用TOPPERS-JSPとTINETが行います。

I/Oボード上には2つのLEDと4ビットのDIPSWと液晶基盤が搭載されています。

開発対象：リセットボタンとディップスイッチ

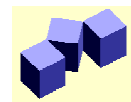
- ボード上には、リセットボタンとモード変更用のディップスイッチが実装されている



開発環境の確認

1. 配布物の確認
2. 開発対象の説明
- [3. 開発環境の説明](#)

開発環境 : PizzaFactory2



<http://support.toppers-open.org/>

- TOPPERSカーネル対応の開発環境のバイナリディストリビューションパッケージ
- TOPPERSカーネルをビルドするために必要なツール類およびカーネルのソースコードを収録
- Windows実行形式のインストーラとして提供。Cygwin環境と比較して容易にインストール可能
- コンパイラはCygwinベースではなく、MinGWベースのGCCを採用。Cygwinベースのコンパイラと比較して2倍以上高速
- ディスクトップ上のアイコン"PizzaFactory2CommandLine"を実行すると、コンソールが起動される

2005/12/19

TOPPERSプロジェクト認定

13



本実習では、PizzaFactory2の開発環境にてプログラム開発を行います。Cygwinに比べて、インストール、アンインストールが容易です。開発環境を含んだインストールが可能なので、GNUのインストールに不安な方に、お勧めします。また、Cygwinに比べて、上記の特徴があります。

開発環境 : GCC



- GNUプロジェクトにより開発されているオープンソースのコンパイラ
- GCCは「GNU Compiler Collection」の略であり、名前が示すように多くの言語(C、C++、Objective-C、FORTRAN、Java、Ada)をサポート
- 多くの種類のプロセッサをサポート
 - Alpha、ARM、AVR、H8、IA64、M32R、M68K、MIPS、SH、SPARC、V850
- GCCはアセンブラやリンカとしてbinutilsを呼び出す
 - アセンブラ(gas)、リンカ(ld)、オブジェクトダンプ(objdump)
- デバッガとしてはGNUプロジェクトより同じくオープンソースのソフトウェアとしてgdbが提供されている

2005/12/19

TOPPERSプロジェクト認定

14



PizzaFactory2では、GNU開発環境はバイナリデータとしてインストールされますが、Cygwin上で自分でインストールする場合は、かなりのノウハウが必要です。TOPPERS/JSPのドキュメント「docディレクトリ」中に、GNUのインストール方法を記述した「gnu_install.txt」ドキュメントがありますので参考にしてください。

開発環境 : Cygwin



- 一般的なGNU開発プログラムを含むUNIXのプログラムをWindows上で動作させるための環境
- Cygwinライブラリ(Cygwin.dll)によりUNIXのシステムコールを提供(バーチャルマシンではない)
- ほぼ全てがGPL/X11ライセンスのフリーソフトウェア
- Cygnus Solution社(現在はRed Hat社の一部)が開発
- インストールはCygwinのホームページからダウンロードできるインストーラを用いる
- インストール方法は書籍を参考のこと
 - Cygwin+CygwinJE-Windowsで動かすUNIX、佐藤 竜一、アスキー
 - Cygwin—Windowsで使えるUNIX環境、川井 義治、米田 聡、ソフトバンクパブリッシング

2005/12/19

TOPPERSプロジェクト認定

15



CygwinはTOPPERS/JSPのH8版の開発環境と同等です。オープンソフトを使って、安価に開発環境を構築したい方にお勧めします。基本的にCygwinのホームページからダウンロードしてインストールを行います。但し、環境設定や日本語化にノウハウが必要です。CD-ROM付の書籍を購入してインストールした方が、安全にインストールが可能です。

TOPPERS/JSPカーネル

JSP = Just Standard Profile

- TOPPERSプロジェクトで開発されている μ ITRON4.0仕様のスタンダードプロファイルに準拠したオープンソースのリアルタイムOS。最新版は Release 1.4.1
- 開発の目的
 - μ ITRON4.0仕様の評価、リファレンス実装
 - 研究・教育機関における研究・教育のプラットフォーム
 - ソフトウェア部品 (IP) 開発のプラットフォーム
 - 評価目的・プロトタイプ開発への利用
 - 実製品への適用
- ターゲット環境
 - SH1/2、SH3、H8、ARMv4、MIPS、PowerPC

2005/12/19

TOPPERSプロジェクト認定

16



本実習では、AKIH8/3069Fボード上で動作するRTOSとしてTOPPERS/JSPを使用します。TOPPERS/JSPは μ ITRON4.0仕様のスタンダードプロファイルに準拠したオープンソースのリアルタイムOSです。TOPPERS/JSPは以下の目的で開発されました。

- 1) μ ITRON4.0仕様の評価、リファレンス実装用に開発を行った。
- 2) 研究・教育機関における組み込み技術の研究・教育のプラットフォームを目指した。
- 3) 組み込みソフトウェアのソフトウェア部品 (IP) 開発のプラットフォームを目指した。
- 4) 評価目的・プロトタイプ開発への利用
- 5) 実製品への適用

TOPPERS/JSPは、多種多様なターゲット環境に対応していますが、今回はH8のAKIH8/3069Fボードをターゲットにしています。このターゲット依存部はソースコード上以下のディレクトリに記述されています。
jsp-1.4.1/config/h8及びjsp-1.4.1/config/h8/akih8_3069f

TOPPERS/JSPカーネルの特徴

- 読みやすく改造しやすいソースコード
 - 定量的な評価は難しいが、読みやすさには自信あり
 - 可能な限り #ifdef を追放
 - 様々な改造・拡張の実績あり
- 他のターゲットへのポーティングが容易な構造
 - 実行性能を落とさないプロセッサの抽象化
 - 新しいプロセッサへのポーティングを2日間で行った例
- 高い実行性能と小さいRAM使用量
 - **！** 大部分をC言語で記述したカーネルとしては
- LinuxおよびWindows上でのシミュレーション環境
 - プロトタイプ開発、教育用に最適
- オープンソースソフトウェアのみで開発環境まで

2005/12/19

TOPPERSプロジェクト認定

17



TOPPERS/JSPの特徴を以下に列記します。

1) 読みやすく改造しやすいソースコード

定量的な評価は困難ですが、読みやすさには自信があります。

可能な限り、条件コンパイル文(#ifや#ifdef等)を追放しました。

いろいろな改造や拡張の実績があります。μITRON4.0仕様の機能拡張・確認にもJSPを使用しています。

2) 他のターゲットへのポーティングが容易な構想

実行性能を落とさないプロセッサの抽象化を行っています。

新しいプロセッサへのポーティングを2日間で行った事例があります。

3) 高い実行性能と小さいRAM使用量

大部分をC言語で記述したリアルタイムカーネルとしては

驚くべき実行性能と少ないメモリ上での実行が可能です。

4) Linux及びWindows上でのシミュレーション環境

Linux及びWindows上でのシミュレーション環境があります。プロトタイプ開発や教育用に最適です。

5) オープンソースソフトウェアのみで開発環境まで

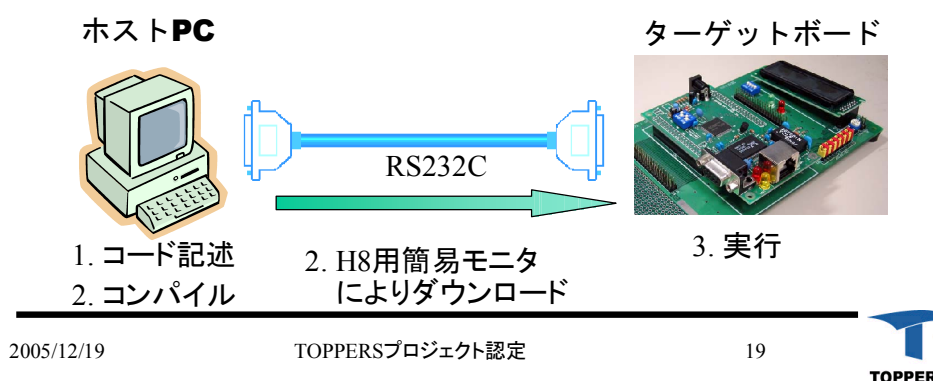
LinuxやCygwinの開発環境をそのまま使用できます。

基本プログラミング

1. [開発環境の構築](#)
2. サンプルプログラムの実行
3. 周辺デバイスプログラミング

開発環境の構築：開発環境の概要

- ターゲットボードは、内蔵フラッシュROMにH8用簡易モニタを書き込んでおき、このプログラムでブートする
- プログラムはホストPC上でクロスコンパイラであるH8用のGCCを用いてコンパイル
- コンパイルしたプログラムをH8用簡易モニタを使いRS232C経由でターゲットボードのRAMにコピーし、実行する



開発環境の概要について説明します。

開発したプログラムは、AKIH8/3069Fボード上のFLASH ROMに直接書き込んで実行させるのではなく、まず、FLASH ROMに「H8用簡易モニタ」を書き込んでおきます（実習ボードにはすでに書き込みが終了しています）。このモニタは電源投入により、RS232Cを通してパソコン上の通信ソフトにプロンプトを表示します。

ビルドしたプログラムは、H8用簡易モニタと通信ソフトの転送機能を使って、AKIH8/3069Fボード上のSDRAMの領域にダウンロードします。ダウンロードが終了すると、モニタはプロンプト要求状態に復帰しますので、コマンドを使ってプログラムをボード上で実行させます。

開発者は、パソコン上の開発環境でTOPPERS/JSPで動作するプログラムを開発し、コンパイル→リンク→ダウンロードのフォーマットに変換して、実機で動作させます。なお、コンパイル、リンク、フォーマット変換は、**make**コマンドを用いて行います。

開発環境の構築 : H8用簡易モニタ

- ユーザープログラムをRS232C経由で、RAM にロードして実行するためのプログラム
- TOPPERSプロジェクトのホームページからダウンロード可能
- H8/3069Fの内蔵フラッシュ ROM の書込み回数には制限があるため、ユーザープログラムを毎回、内蔵フラッシュ ROM に書込んでデバッグするのは、あまりよい方法ではない
- 内蔵フラッシュ ROM には、H8 簡易モニタを書込んでおき、ユーザープログラムを H8/3069F の内蔵 RAM、または外部に増設した RAM にロードし、デバッグを行う
- メモリ内容の表示・変更機能も持つ
- 詳細はプログラムディレクトリの ./h8mon/readme.txtを参照のこと

2005/12/19

TOPPERSプロジェクト認定

20



H8簡易モニタについて説明します。

このプログラムはTOPPERSプロジェクトのホームページからダウンロード可能です。AKIH8/3069Fボードには、実行用に2MBの大きなメモリを持ちます。プログラムとデータをこの領域で実行が可能です。これに加えて、CPU内蔵のFLASH ROMには、書き込み回数に制限があり、何回もFLASH ROMに書き込みを行うと、書き込みエラーが発生することがあります。そこで、本実習の実行環境を「H8簡易モニタ」を用いて、ダウンロード実行する環境としました。モニタについての詳細は、./h8mon/readme.txtを参照してください。

開発環境の構築：モニタの書き込み



- **AKI-H8/3069F上のディップスイッチ**を設定し内蔵フラッシュROM書き込みモードに(DIP1:ON、DIP2:ON、DIP3:OFF、DIP4:ON)
- ターゲットボードとホストPCをRS232Cで接続し、ターゲットボードの電源をいれる(電源が入った瞬間LED3が一瞬点灯する)
- コンソールを開きプログラムディレクトリの ./h8mon/mon3069f に移動
- h8write.exe を使い内蔵フラッシュROMにmon3069.motを書き込む

```
$ cd ./h8mon/mon3069f
$ ls
3069.S* 3069.h* 3069.ld* Makefile* load3069.mot* mon3069.mot*
$h8write.exe -3069 -f20 mon3069.mot COM1
H8/3069F is ready! 2002/5/20 Yukio Mituiwa.
writing
ingore record
ingore record
.....
EEPROM Writing is succeeded.
```

接続している
COMポートを
指定

2005/12/19

TOPPERSプロジェクト認定

21



H8簡易モニタの書き込み手順を説明します。(本実習では、書き込みを行う必要はありません)

FLASH ROMへの書き込みはAKIH8/3069Fボードに添付されているCD-ROM中のh8write.exeプログラムを使用します。

- 1) 電源切った状態で、AKIH8/3069Fボード上のDIPSWをFLASH ROM書き込みモードに設定します。DIP-SWを1-ON、2-ON、3-OFF、4-ONに設定します。
- 2) RS232Cケーブルを接続して、ターゲットボードの電源を入れます。電源投入でLED3が一瞬点灯します。
- 3) DOS窓(または、PizzaFactory2、または、Cygwin)のコマンドラインより、./h8mon/mon3069fに移動して、以下のコマンドを実行します。

> h8write.exe -3069 -f20 mon3069.mot COM1(リターン)

このコマンドのCOM1はRS232CケーブルがCOM1に接続されていることを想定しての入力です。他のポートに接続されている場合は、そのポートを指定してください。

- 4) 書き込みが終了すると「EEPROM Writing is succeeded.」が表示されます。

開発環境の構築：モニタの動作確認



- 書き込みが終了したらターゲットボードの電源を切る
- AKI-H8/3069F上のディップスイッチを設定し、通常動作モード(モード5)に設定(DIP1:ON、DIP2:OFF、DIP3:ON、DIP4:OFF)
- 通信ソフト(TeraTerm等)をRS232Cを接続しているポートにスピード38400bps、データ8bit、パリティなし、ストップビット1bitで接続
- ターゲットボードの電源を入れ、ターミナルソフトウェアにH8用簡易モニタの起動ログが出力されるか確認する

?を入力

```
1:H8/3069F Monitor v1.12 Copyright (C) 1999-2004 CSE Tomakomai NCT
1:?
H8/3069F Monitor v1.12 Copyright (C) 1999-2004 CSE Tomakomai NCT
-----
ld          load program      | go[<ad>]    go program
dw[<ad>[<ln>]] dump word      | db[<ad>[<ln>]] dump byte
ew[<ad>]     enter word
fb[<ad>[<ln>[<in>[<ic>[<rp>]]]]] fill byte
?           help
1:
```

2005/12/19

TOPPERSプロジェクト認定

22



モニタの動作確認を行います。

- 1) 書き込みが終了したら、ターゲットボードの電源を切ってください。
- 2) AKIH8/3069FボードのDIP-SWを通常動作モード(モード5)に設定します。DIP-SWを1-ON、2-OFF、3-ON、4-OFFに設定する。
- 3) パソコン上で通信ソフト(TeraTerm等)を起動し、シリアルポートの設定を以下にしてください。

Baud rate:38400

Data:8bit

Parity:none

Stop:1bit

Flow control:XON/XOFF

- 4) ターゲットボードの電源を入れて、バーナー表示とプロンプトが表示されることを確認してください
- 5) ?コマンドで、ヘルプメニューが表示されます。

基本プログラミング

1. 開発環境の構築
- [2. サンプルプログラムの実行](#)
3. 周辺デバイスプログラミング

サンプルプログラムの実行 : sample1

- TOPPERS/JSPの配布パッケージにはカーネルの基本的な動作を確認するためのsample1と呼ばれるプログラムが含まれている
- sample1はコンソールからの入力を受け付けるメインタスクと、三つの並列実行されるタスクで構成される
- 並列実行されるタスクは、一定回数空ループを実行する度に、タスクが実行中であることをあらわすメッセージを表示する
- メインタスクは、コンソールからの入力を待ち、(待ちの間は、並列実行されるタスクが実行される)、入力に対応した処理を実行する
 - sample1のコンパイル、ダウンロード、実行によりTOPPERS/JSPカーネルの構築・実行方法を学ぶ

2005/12/19

TOPPERSプロジェクト認定

24



TOPPERS/JSPの配布パッケージ中に、カーネルの基本的な動作確認を行うためのプログラム **sample1**が含まれています。**sample1**は起動後4つのタスクを起動します。メインタスクはコンソールからの入力を解析し、他の3つのタスクにサービスコールを用いて動作の指示を行います。3つのタスクはコンソールから実行状態を表示します。

TOPPERS/JSPの実行確認用にこのプログラムを構築(ビルド)し、AKIH8/3069F上で実行してみましよう。

サンプルプログラムの実行 : カーネルライブラリ

- TOPPERS/JSPカーネルの構築方法には二つの方法がある
 - アプリケーションとカーネルを同時に構築
 - カーネルをライブラリとして構築しアプリケーションとリンク
- アプリケーションとカーネルを同時に構築すると、コンパイル時間やディスク容量が必要となるため、本演習ではカーネルをライブラリとして構築しアプリケーションとリンクする方法を採る
- プログラムディレクトリにディレクトリ ./jsp-1.4.1/OBJ/AKIH8_3069F/libkernel を作成し以下の手順でカーネルライブラリ (libkernel.a) を構築する

```
$ cd jsp-1.4.1/OBJ/AKIH8_3069F
$ mkdir libkernel
$ cd libkernel
$ ../../configure -C h8 -S akih8_3069f
$ make depend
$ make libkernel.a
```

TOPPERS/JSPのカーネルのビルド方法は2通りあります。ひとつは、毎回アプリケーション(例えば sample1)のビルド時、毎回カーネルもビルドし直す。もうひとつは、カーネルをライブラリ化しておき、アプリケーションのビルド時、ライブラリとしてリンクする。前者の場合、リビルドしたおした後、カーネル自体もコンパイル、リンクするため構築に時間が掛かります。一方、後者では、ビルド時間の短縮になりますが、カーネルにかかわる定義を変更した場合、カーネル自体を構築しなおさねばなりません。アプリケーションのビルド時自動ではやってくれません。このような管理を自分で行わねばなりません。

今回のセミナーでは、アプリケーションを何度も、構築しなおさねばなりません。そこで、カーネルをライブラリ化して構築を行うようにしましょう。このセミナーでは1日目、2日目、3、4日目でカーネルの設定条件が違います。必要に応じて、カーネルライブラリ (libkernel.a) を構築し直す必要があります。

カーネルライブラリは、jsp-1.4.1/OBJ/AKIH8_3069F/libkernelに構築します。libkernelディレクトリを作成し、そのディレクトリに移動して以下のコマンドにて構築を行います。

```
../../configure -C h8 -S akih8_3069f
```

```
make depend
```

```
make libkernel.a
```

サンプルプログラムの実行 : sample1の構築

- プログラムディレクトリにディレクトリ ./jsp-1.4.1/OBJ/AKIH8_3069F/sample1を作成configureを実行する
- configureを実行すると, サンプルプログラムを構築するためのファイルが生成される
 - Makefile : メイクファイル
 - sample1.c/h : サンプルプログラムのソースコード
 - sample1.cfg : サンプルプログラムのコンフィギュレーションファイル

```
$ cd jsp-1.4.1/OBJ/AKIH8_3069F
$ mkdir sample1
$ cd sample1
$ ../../configure -C h8 -S akih8_3069f
configure: Generating Makefile from ../sample/Makefile.
configure: Generating sample1.c from ../sample/sample1.c.
configure: Generating sample1.h from ../sample/sample1.h.
configure: Generating sample1.cfg from ../sample/sample1.cfg.
$ls
Makefile sample1.c sample1.cfg sample1.h
```

2005/12/19

TOPPERSプロジェクト認定

26



sample1をjsp-1.4.1/OBJ/AKIH8_3069F/sample1ディレクトリ上に構築します。sample1ディレクトリを作成し、このディレクトリに移動します。以下のコマンドを実行し、メイクファイルとソースファイルを構築します。

```
../../configure -C h8 -S akih8_3069f
```

configureを実行すると, サンプルプログラムを構築するためのファイルが生成される

Makefile	: メイクファイル
sample1.c/h	: サンプルプログラムのソースコード
sample1.cfg	: サンプルプログラムのコンフィギュレーションファイル

サンプルプログラムの実行 : sample1 の構築

- プログラムディレクトリにディレクトリ `./jsp-1.4.1/`
`/OBJ/AKIH8_3069F/sample1` を作成 `configure` を実行する

```
$ cd jsp-1.4.1/OBJ/AKIH8_3069F
$ mkdir sample1
$ cd sample1
$ ../../configure -C h8 -S akih8_3069f
```

- `./jsp-1.4.1/OBJ/AKIH8_3069F/sample1/` に作成された Makefile をエディタで開き81行目にある `KERNEL_LIB` に先ほどカーネルライブラリを作成したディレクトリを指定する

```
#
# カーネルライブラリ(libkernel.a)のディレクトリ名
# (カーネルライブラリも make 対象にする時は、空に定義する)
#
KERNEL_LIB = ../libkernel
```

- `make depend` 及び `make` を実行することで、コンパイル及びカーネルライブラリとのリンクを行い、`jsp.srec` の生成を確認

2005/12/19

TOPPERSプロジェクト認定

27



`sample1`を`jsp-1.4.1/OBJ/AKIH8_3069F/sample1`ディレクトリ上に構築します。`sample1`ディレクトリを作成し、このディレクトリに移動します。以下のコマンドを実行し、メイクファイルとソースファイルを構築します。

```
../../configure -C h8 -S akih8_3069f
```

カーネルライブラリを参照するために、`Makefile`の81行目の`KERNEL_LIB`にカーネルライブラリのベースアドレス(`../libkernel`)を設定するように、書き換えを行います。以下のコマンドで構築を行います。

```
make depend
```

```
make
```

サンプルプログラムの実行 : ボードのセットアップ

- AKI-H8/3069F上のディップスイッチを設定し、通常動作モード(モード5)に設定(DIP1:ON、DIP2:OFF、DIP3:ON、DIP4:OFF)
- ターミナルソフトウェア(TeraTerm等)をRS232Cを接続しているポートにスピード38400bps、データ8bit、パリティなし、ストップビット1bitで接続
- ターゲットボードの電源を入れ、ターミナルソフトウェアにH8用簡易モニタの起動ログが出力されるか確認する

?を入力

```
1:H8/3069F Monitor v1.12 Copyright (C) 1999-2004 CSE Tomakomai NCT
1:?
H8/3069F Monitor v1.12 Copyright (C) 1999-2004 CSE Tomakomai NCT
-----
ld          load program      | go[<ad>]   go program
dw[<ad>[<ln>]] dump word      | db[<ad>[<ln>]] dump byte
ew[<ad>]     enter word
fb[<ad>[<ln>[<in>[<inc>[<rp>]]]] fill byte
?           help
1:
```

2005/12/19

TOPPERSプロジェクト認定

28



ROMモニタのコマンドを以下に示します。

- 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。

サンプルプログラムの実行 : sample1 のダウンロード

- ターミナルソフトウェアを開き、H8用簡易モニタと接続
- H8用簡易モニタの `ld` コマンドを実行し、ターミナルソフトウェアのファイル転送機能により作成した `jsp.srec` をダウンロードする (TeraTerm の場合はドラッグ&ドロップでダウンロード可能)
- ダウンロード後、`go` コマンドによりカーネルをスタートさせる

ロード

実行

```

Tera Term - COM2 VT
File Edit Setup Control Window Help
l:
l: H8/3089F Monitor v1.12 Copyright (C) 1999-2004 CSE Tomakomai NCT
l: ld
l: go

JSP Kernel Release 1.4 (patchlevel = 1) for AKI-H8/3089F (Oct 16 2004, 02:45:02)
Copyright (C) 2000-2003 by Embedded and Real-Time Systems Laboratory
Toyohashi Univ. of Technology, JAPAN

System logging task is started on port 2.
Sample program starts (exinf = 0).
task1 is running (001).
task1 is running (002).
task1 is running (003).
task1 is running (004).
task1 is running (005).
task1 is running (006).
task1 is running (007).
  
```

2005/12/19

TOPPERSプロジェクト認定

29



sample1の操作はターミナルソフトウェアからコマンドの入力で行います。コマンドの表は以下の通りです。

'1':	以降のコマンドは TASK1 に対して行う。
'2':	以降のコマンドは TASK2 に対して行う。
'3':	以降のコマンドは TASK3 に対して行う。
'a':	タスクを act_tsk により起動する。
'A':	タスクに対する起動要求を can_act によりキャンセルする。
'e':	タスクに ext_tsk を呼び出させ、終了させる。
't':	タスクを ter_tsk により強制終了する。
'>':	タスクの優先度を HIGH_PRIORITY にする。
'=':	タスクの優先度を MID_PRIORITY にする。
'<':	タスクの優先度を LOW_PRIORITY にする。
'G':	タスクの優先度を get_pri で読み出す。
's':	タスクに slp_tsk を呼び出させ、起床待ちにさせる。
'S':	タスクに tslp_tsk(10秒) を呼び出させ、起床待ちにさせる。
'w':	タスクを wup_tsk により起床する。
'W':	タスクに対する起床要求を can_wup によりキャンセルする。
'l':	タスクを rel_wai により強制的に待ち解除にする。
'u':	タスクを sus_tsk により強制待ち状態にする。
'm':	タスクの強制待ち状態を rsm_tsk により解除する。
'M':	タスクの強制待ち状態を frsm_tsk により強制解除する。

サンプルプログラムの実行 : sample1 のコマンド

- ターミナルソフトウェアからコマンドを入力してsample1を制御してみましょう。
- ‘s’コマンドでタスク1を停止させると、表示がtask2の実行に変わります。
- ‘2’コマンドで対象をタスク2に変え、‘s’コマンドで停止させると、表示がtask3の実行に変わります。
- ‘3’コマンドで対象をタスク3に変え、‘s’コマンドで停止させると、3つのタスクが停止し表示が止まります。
- ‘1’コマンドで対象をタスク2に変え、‘w’コマンドで実行を再開させると、表示がtask1実行に変わります。
- 他のコマンドでも、試してみましょう。

2005/12/19

TOPPERSプロジェクト認定

30



'd' :	タスクに <code>dly_tsk</code> (10秒) を呼び出させ、時間経過待ちにさせる。
'x' :	タスクにパターン <code>0x0001</code> の例外処理を要求する。
'X' :	タスクにパターン <code>0x0002</code> の例外処理を要求する。
'y' :	タスクに <code>dis_tex</code> を呼び出させ、タスク例外を禁止する。
'Y' :	タスクに <code>ena_tex</code> を呼び出させ、タスク例外を許可する。
'r' :	三つの優先度のレディキューを回転させる。
'c' :	周期ハンドラを動作させる。
'C' :	周期ハンドラを停止させる。
'V' :	<code>vxget_tim</code> で性能評価用システム時刻を2回読む。
'v' :	発行したシステムコールを表示する(デフォルト)。

基本プログラミング

1. 開発環境の構築
2. プログラムのコンパイル及び実行
3. 周辺デバイスプログラミング

周辺デバイスプログラミング

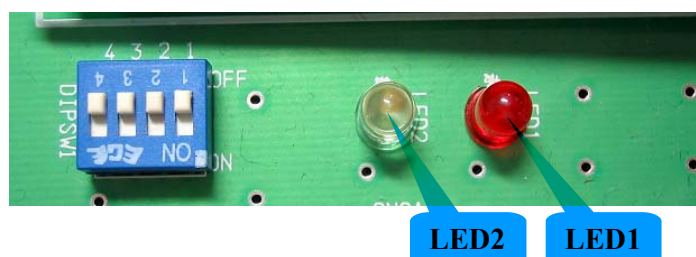
- sample1をベースに、I/Oボード上のLEDとDIPスイッチを操作するプログラムを作成する
- 以下の事項を学ぶ
 - マニュアルの読み方
 - TOPPERS/JSPカーネルのシステムインタフェースレイヤー(SIL)の使い方
- 完成プログラム
 - プログラムディレクトリの
./jsp-1.4.1/OBJ/AKIH8_3069F/device

sample1を修正してI/Oボード上のLEDとDIP-SWを操作するプログラムを作成します。周辺デバイスプログラム(device.hとdevice.c)は、./jsp-1.4.1/OBJ/AKIH8_3069F/deviceディレクトリ中にあります。これを利用してプログラムを修正します。

TOPPERS/JSPでは、トロン協会で策定した「デバイスドライバー設計ガイドライン」中のシステムインターフェース層(SIL)を用いて外部のポートアクセスを行っています。LEDやDIP-SWのハードウェアインターフェースとそのアクセス方法について、説明を行っていきます。

周辺デバイスプログラミング : LEDとDIP

- I/Oボード上には2つのLED(LED1とLED2)と4ビットのディップスイッチ(写真の右からDIP1、DIP2、DIP3、DIP4)が実装されている。
- LEDとDIPそれぞれがAKI-H8/3069Fにどのように接続されているかI/OボードのマニュアルとAKI-H8/3069Fのマニュアルを参照して確認しましょう。



2005/12/19

TOPPERSプロジェクト認定

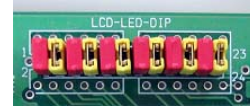
33



I/Oボード上には、2つのLED(LED1とLED2)と4ビットのディップスイッチ (DIP-SW) が実装されています。LEDとDIP-SWはI/Oボード横の10ピンのジャンパーピン(LCD-LED-DIP)を介して、CPUボードのCN2Aコネクタに接続しています。

周辺デバイスプログラム：LEDとDIPの接続

- LEDとDIPはそれぞれLCD-LED-DIPに接続されている
- LCD-LED-DIPはジャンパピンにより隣りあったピンを接続可能
- LEDとDIPが接続されているピンの隣りのピンはCN2Aに接続されている
- CN2Aは、AKI-H8/3069Fのポートに接続されている



	LED-LCD-DIP			CN2A	H8/3069f
LED1	21	-	22	11	D6/P46
LED2	23	-	24	12	D7/P47
DIP1	13	-	14	37	P50/A16
DIP2	15	-	16	38	P51/A17
DIP3	17	-	18	39	P52/A18
DIP4	19	-	20	40	P53/A19

2005/12/19

TOPPERSプロジェクト認定

34



LEDとDIP-SWはジャンパピン(LCD-LED-DIP)とコネクタCN2Aを介して、CPUボード上の3069F-CPUの平行ポートに接続しています。ジャンパピンが接続していなければ、LEDの制御を行ったり、DIP-SWの状態を取り込むことはできません。

LEDとDIP-SWとCPUとの接続は以下の通り。

LED1	ポート4の6ビット
LED2	ポート4の7ビット
DIP1	ポート5の0ビット
DIP2	ポート5の1ビット
DIP3	ポート5の2ビット
DIP4	ポート5の3ビット

周辺デバイスプログラム：LEDの接続先ポート

- LEDは、I/Oポート4のビット6と7に接続されている
- このポートは、表記から(D6/P46)他の機能との排他になっている
- H8/3069FのマニュアルからI/Oポート4に関連する説明を参照(P328)
 - データバス兼用の8bit入出力ポート
 - モード1～5(拡張モード)のときはバス幅コントロールレジスタ(ABWCR)によりエリア0～7の全てを8ビットアクセス空間に設定すると8ビットバスモードになり、ポート4は入出力ポートとなる
 - モード7の場合はポート4は入出力ポートとなる
- プロセッサの動作モードをマニュアルでチェック
 - AKI-H8/3069Fではモード5で使用
- モード5のため、バス幅コントロールレジスタ(ABWCR)を設定して8ビットバスモードにする必要がある
- ABWCRをマニュアルで確認(P162)
 - アドレス 0xee020、8bitレジスタ。'1'を書き込むと8ビットアクセス空間に設定

2005/12/19

TOPPERSプロジェクト認定

35



LEDに接続しているI/Oポート4は実行モードにより設定が違い、シングルチップモード:モード7では入出力ポートに固定となります。拡張モード:モード1～5では、8ビットバスモードでは入出力ポート、16ビットバスモードではデータ入出力端子となります。バス幅はバス幅コントロールレジスタ(ABWCR)レジスタにより設定されます。このレジスタではエリア0から7までのバス幅の設定が行えます。AKIH8/3069Fでは、モード5で外部メモリを8ビットバスモードで使用しており、デフォルトは8ビットバスモードなのであえて設定を行う必要はなく、ポート4は入出力ポートとして使用可能です。

本実習では、`device.c`中の`initial_led`関数でABWCRレジスタの初期化を行っています。

周辺デバイスプログラム：LEDの接続先ポート

- I/Oポート4の構成レジスタを確認(P329)
 - P4DDR(0xee003)
 - ポート4データディレクションレジスタ(1で出力ポート)
 - P4DR : 0xfffd3
 - ポート4データレジスタ
 - P4PCR : 0xee03e
 - ポート4入力プルアップMOSコントロールレジスタ
 - (P4DDRが0で1にするとMOSがON)
- 以上よりLEDポートの初期化と操作は以下のようになる
 - 初期化
 - ABWCR(0xee020) に 0xff を書き込み8ビットバスモードに設定
 - P4DDR(0xee003)のビット6と7に'1'を書き込み出力に設定
 - LED操作
 - P4DRのビット6と7にデータを書き込む

2005/12/19

TOPPERSプロジェクト認定

36



I/Oポート4は3つのレジスタによって構成されます。

P4DDR(ポート4データディレクションレジスタ)

8ビットの書き込み専用レジスタで、ポート4の各端子の入出力をビットごとに設定できます。各ビットは1で出力ポート、0で入力ポートとなります。

P4DR(ポート4データレジスタ)

8ビットのリード/ライト可能なレジスタで、ポート4の出力データを格納します。ポート4が出力ポートとして機能する場合、本レジスタの値が出力されます。また、このレジスタをリードすると、P4DDRが0のビットは端子のロジックレベルが読み出され、1のビットは、P4DRの値が読み出されます。

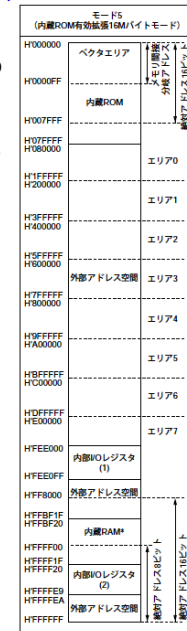
P4PCR(ポート4入力プルアップMOSコントロールレジスタ)

P4PCRは8ビットのリード/ライト可能なレジスタで、ポート4に内蔵した入力プルアップMOSをビットごとに制御します。モード1～5(拡張モード)の8ビットバスモード時とモード7(シングルチップモード)時、P4DDRを0にクリアした(入力ポートの)状態で、P4PCRを1にセットすると入力プルアップMOSはONします。

初期化関数(initial_led)でABWCRとP4DDRの初期化を行い。LED操作関数(set_led)でP4DRの6と7ビットの設定により、LEDの点灯、消灯を行います。

周辺デバイスプログラム：モードとアドレス

- H8/3069Fはモード5の場合、アドレス幅は24ビットとなる
- 一方周辺デバイスのアドレスは20ビットで書かれている
 - ✓ ex. ABWCR : 0xee020
- そのため、周辺デバイスのアクセスの際には、上位4ビットには0xfを付けなければならない
 - ✓ マニュアルの作りが不親切な気が...



2005/12/19

TOPPERSプロジェクト認定

37



H8/3069Fのモード5 (H8/3069F版TOPPERS/JSPが対応しているモード) では、アドレス幅は24ビットとなります。I/Oポート等のレジスタや内蔵RAMは0xFEE000番地以降の領域に配置されますので、ABWCRレジスタは0xFEE020番地に配置されます。

周辺デバイスプログラム：システムインターフェースレイヤー

- TOPPERS/JSPカーネルは、汎用OSと異なり、ユーザーモードとカーネルモードの区別がない。そのため、ユーザープログラムから直接デバイスを操作可能である
- TOPPERS/JSPカーネルは、システムインターフェースレイヤー(SIL)の一部をサポートしている
- SILは、デバイスドライバのポータビリティを向上させるために策定されているITRONデバイスドライバ設計ガイドラインで提案されている
- SILを用いるプログラムは `s_services.h` をインクルードする必要がある
- 今回用いるのはデバイスレジスタへのアクセス関数群

VB sil_reb_mem(VP mem)	mem のアドレスから8ビット単位でデータを読む
void sil_wrb_mem(VP mem, VB data)	memアドレスにdataを8ビット単位で書き込む
VH sil_reh_mem(VP mem)	mem のアドレスから16ビット単位でデータを読む
void sil_wrh_mem(VP mem, VH data)	memアドレスにdataを16ビット単位で書き込む
VW sil_rew_mem(VP mem)	mem のアドレスから32ビット単位でデータを読む
void sil_rww_mem(VP mem, VW data)	memアドレスにdataを32ビット単位で書き込む

2005/12/19

TOPPERSプロジェクト認定

38



「デバイスドライバ設計ガイドライン」では、デバイスドライバを3つのコンポーネントに分けています。

GDIC (General Device Interface Component)

PDIC (Primitive Device Interface Component)

SIL (System Interface Layer)

TOPPERS/JSPでは、デバイスとのアクセスインターフェースであるSILの仕様に従ってデバイスのアクセスを行う。具体的には、以下のアクセス関数を持っています。

VB sil_reb_mem(VP mem);

memのアドレスから8ビット単位でデータを読み込む

void sil_wrb_mem(VP mem, VB data);

memアドレスにdataを8ビット単位で書き込む

VH sil_reh_mem(VP mem);

memのアドレスから16ビット単位でデータを読み込む

void sil_wrh_mem(VP mem, VH data);

memアドレスにdataを16ビット単位で書き込む

VW sil_rew_mem(VP mem);

memのアドレスから32ビット単位でデータを読み込む

void sil_rww_mem(VP mem, VW data);

memアドレスにdataを32ビット単位で書き込む

周辺デバイスプログラム：プロジェクトへのファイル追加

- sample1に device.h device.c を追加し、LED操作関数を記述する
- プロジェクトにファイルを追加した場合には、そのファイルをコンパイルするようMakefileのUTASK_COBJに.oファイルを追加する。

```

ifdef USE_CXX
    UTASK_CXXOBS = $(UNAME).o
    UTASK_COBS =
else
    UTASK_COBS = $(UNAME).o device.o
endif

```

C++をサポート
する場合はこ
ちらに追加

- プロジェクトにファイルを追加した場合は依存関係を再構築するため、以下のコマンドを実行すること

```

$ make realclean
$ make depend

```

まず、sampe1ディレクトリに、deviceディレクトリからdevice.hとdevice.cをコピーします。

次に、Makefileにdevice.cをコンパイル、リンクするように指定を追加します。具体的には、Makefile中のUTASK_COBJに、device.cのコンパイル結果であるdevice.oを追加します。

周辺デバイスプログラム：ヘッダーファイル

- プログラムは外部関数として次の二つの関数を作成する

```
extern void initial_led(void);  
extern void set_led(LED led, CONDITION req);
```

- LEDとCONDITIONは列挙型として定義する

```
typedef enum{ LED1, LED2 }LED;  
typedef enum{ON, OFF }CONDITION;
```

sample1.cからLEDの初期化とLED操作関数が読み出されるように、関数のexternal宣言と引数として用いるenumを定義します。この記述はdevice.hに記述されていますので、sample1.cからdevice.hをインクルードするように書き換えればOKです。

周辺デバイスプログラム : sample1.cの変更

- sample1.c には main_task のスタート時に initial_led() を呼び出し LED を初期化するコードを追加
- '4' を入力すると LED1 を点灯
- '5' を入力すると LED1 を消灯
- '6' を入力すると LED2 を点灯
- '7' を入力すると LED2 を消灯

```
case '3':
    tskno = 3;
    tskid = TASK3;
    break;
case '4':
    set_led(LED1, ON);
    break;
case '5':
    set_led(LED1, OFF);
    break;
case '6':
    set_led(LED2, ON);
    break;
case '7':
    set_led(LED2, OFF);
    break;
case 'a':
```

sample1.cの本文を改造します。main_taskのスタート時にデバイスの初期化を行うように、main_task関数のはじめに、initial_led関数を呼び出すように修正します。次に、LEDに対する設定を行うために、メインループ中のswitch文にcase '4' から '7' を追加し、各処理に対応したLED操作関数を追記します。

周辺デバイスプログラム：プログラム例(1/2)

■ マクロ定義とグローバル変数

```
#define ABWCR      0xfe020 /* バス幅コントロールレジスタ */
#define P4DDR      0xfe003 /* データディレクションレジスタ */
#define P4DR       0xffffd3 /* データレジスタ */

#define P4DDR_B6   0x40 /* LED1接続ポートのビット */
#define P4DDR_B7   0x80 /* LED2接続ポートのビット */

unsigned char cled; /* LEDの設定状態を保存 */
```

■ 初期化関数

```
void
initial_led(){
    cled = 0x00;
    /* 8ビットバスモード */
    sil_wrb_mem((VP)ABWCR, 0xff);
    /* P4のビット6とビット7を出力ポートに */
    sil_wrb_mem((VP)P4DDR, sil_reb_mem((VP)P4DDR) | (P4DDR_B6 | P4DDR_B7));
    /* P4の出力データ初期化(初期状態は消灯) */
    sil_wrb_mem((VP)P4DR, sil_reb_mem((VP)P4DR) & ~(P4DDR_B6|P4DDR_B7));
}
```

2005/12/19

TOPPERSプロジェクト認定

42



P4DDRポートの初期化手順について、上記の記述ではP4DDRポート読み出したあとに、出力ビットを指定して再書き込みを行っています。P4DDRポートはWrite onlyポートのため、読み出しを行うとルネサス社のマニュアルでは1として読み出されます。そのためP4ポート全てが出力ポートとして設定されます。I/Oボード上はP4ポートの他のビットはLCD用の出力に設定されているため、動作上は問題ありません。P4DDRポートの初期値はLCDの設定を含めて以下の記述の方が適切です。

```
#define P4DDR_LCD      0xfc
sil_wrb_mem((VP)P4DDR, (P4DDR_B6|P4DDR_B7|P4DDR_LCD));
```

周辺デバイスプログラム : プログラム例 (2/2)

■ LED接続ポートの書き込み

```
void
led_out(unsigned char led_data){
    sil_wrb_mem((VP)P4DR,
                sil_reb_mem((VP)P4DR) & ~(P4DDR_B6 | P4DDR_B7)
                |(led_data & (P4DDR_B6 | P4DDR_B7)));
}
```

■ 出力関数

```
void
set_led(LED led、CONDITION req){
    unsigned char rled = cled;
    switch(led){
        case LED1:
            if (req == OFF)
                rled &= ~P4DDR_B6;
            else
                rled |= P4DDR_B6;
            break;
    }
```

```
        case LED2:
            if (req == OFF)
                rled &= ~P4DDR_B7;
            else
                rled |= P4DDR_B7;
            break;
        default:
            break;
    }
    if (cled != rled) {
        cled = rled;
        led_out(cled);
    }
}
```

周辺デバイスプログラム：DIPスイッチ

- ‘8’を押すことによって、DIPスイッチの状態をコンソールに出力するように変更
- DIPスイッチは、ポート5に接続
 - DIP1 : P50/ DIP2 : P51/ DIP3 : P52/ DIP4 : P53
- ポート5構成レジスタ
 - P5DDR : 0xfe004 : データディレクションレジスタ(1で出力ポート)
 - P5DR : 0xffffd4 : ポート5データレジスタ
 - P5PCR : 0xfe03f : ポート5入力プルアップ
- モード5ではポート5はP5DDRの設定によりアドレスバスまたは入力ポートになる
 - P5DDRを0x00にして入力モードにするとポートが有効に
 - プルアップはONに

2005/12/19

TOPPERSプロジェクト認定

44



DIP-SWの1から4ピンは、ポート5の0ビットから3ビットに接続しています。ポート5はモード1から4の場合、4ビットのアドレス出力端子となります。モード7の場合、4ビットの入出力ポートとなります。モード5(本設定)の場合、4ビットの出力端子と4ビットの入力ポートの兼用となり、P5DDRレジスタの設定により切り替えが可能です。

P5DDR(ポート5データディレクションレジスタ)

8ビットの書き込み専用レジスタで、ポート5の各端子の入出力をビットごとに設定できます。各ビットは1で出力ポートまたはアドレス出力端子、0で入力ポートとなります。ビット7～4はリザーブビットで1に固定です。

P5DR(ポート5データレジスタ)

8ビットのリード／ライト可能なレジスタで、ポート5の出力データを格納します。ポート5が出力ポートとして機能する場合、本レジスタの値が出力されます。また、このレジスタをリードすると、P5DDRが0のビットは端子のロジックレベルが読み出され、1のビットは、P5DRの値が読み出されます。ビット7～4はリザーブビットで1に固定です。

P5PCR(ポート5入力プルアップMOSコントロールレジスタ)

P5PCRは8ビットのリード／ライト可能なレジスタで、ポート5に内蔵した入力プルアップMOSをビットごとに制御します。モード5(拡張モード)とモード7(シングルチップモード)時、P5DDRを0にクリアした(入力ポートの)状態でP5PCRを1にセットすると入力プルアップMOSはONします。ビット7～4はリザーブビットで1に固定です。本ハードウェアではプルアップを行う必要があります。

周辺デバイスプログラム：インタフェース(1/2)

- プログラムは外部関数として次の二つの関数を作成する

```
extern void initial_switch(void);  
extern CONDITION get_switch(SWITCH sw);
```

- SWITCHとCONDITIONは列挙型として定義する

```
typedef enum{  
    ON,  
    OFF  
}CONDITION;
```

```
typedef enum{  
    SW1,  
    SW2,  
    SW3,  
    SW4  
}SWITCH;
```

device.hインクルードファイル中にスイッチ用の定義を行っています。

initial_switch関数は、スイッチデバイスの初期化を行います。

get_switch関数は、DIP-SWの状態を取り込みます。引数として**SWITCH**タイプの4つのどれかを指定すると、戻り値として**CONDITION**タイプの状態を返します。

周辺デバイスプログラム：インタフェース (2/2)

- sample1.c ではmain_taskの実行がスタートしたら initial_switch()を呼び出しDIPを初期化するコードを追加
- vmsk_log関数の第一引数をLOG_UPTO(LOG_NOTICE)に変更するとDIPの各ビットの状態をコンソールに出力

```
if (get_switch(SW1) == ON) {
    syslog(LOG_NOTICE, "SW1 is ON");
}
else {
    syslog(LOG_NOTICE, "SW1 is OFF");
}

if (get_switch(SW2) == ON) {
    syslog(LOG_NOTICE, "SW2 is ON");
}
else {
    syslog(LOG_NOTICE, "SW2 is OFF");
}
.....
```

sample1.cでは、main_task関数を改造しinitial_switch関数を追加してスイッチの初期化を行います。get_switch関数を用いて、スイッチの状態をログ表示するように改造します。

周辺デバイスプログラム：プログラム例(1/2)

■ マクロ定義

```
#define P5DDR      0xfe004 /* ポート5ディレクションレジスタ */
#define P5DR       0xffffd4 /* ポート5データレジスタ */
#define P5PCR      0xfe03f /* ポート5プルアップレジスタ */
#define P5DR_B0    0x01 /* DIP1接続ポートのビット */
#define P5DR_B1    0x02 /* DIP2接続ポートのビット */
#define P5DR_B2    0x04 /* DIP3接続ポートのビット */
#define P5DR_B3    0x08 /* DIP4接続ポートのビット */

#define P5PCR_PULLUP 0x0f
```

■ 初期化関数

```
void
initial_switch(){
    sil_wrb_mem((VP)P5DDR, 0x00); /* 入力ポート */
    sil_wrb_mem((VP)P5PCR, P5PCR_PULLUP); /* PULL UP */
}
```

2005/12/19

TOPPERSプロジェクト認定

47



DIP-SW対応のポートのマクロ定義が**device.c**ファイル中に記載されています。DIP-SWを入力として取り込むためには、プルアップ設定としなければなりません。この中の**P5PCR_PULLUP**の定義は、プルアップ対象の4つのビット定義するためのものです。

initial_switch関数は、DIP-SW参照のための設定を行っています。ポート5の4ビットを以下の関数で入力設定とします。

```
sil_wrb_mem((VP)P5DDR, 0x00);
```

DIP-SWの対象のビットを以下の関数でプルアップします。

```
sil_wrb_mem((VP)P5PCR, P5PCR_PULLUP);
```

周辺デバイスプログラム：スイッチ状態の取得

■ ポート状態の取得

```
unsigned char
sense_switch(void){
    return(sil_reb_mem((VP)(P5DR))&(P5DR_B0|P5DR_B1|P5DR_B2|P5DR_B3));
}
```

■ スイッチ状態の取り出し

```
CONDITION
get_switch(SWITCH sw){
    unsigned char key;
    CONDITION result = OFF;

    key = sense_switch();
    switch(sw){
    case SW1:
        if((key & P5DR_B0) == 0)
            result = ON;
        break;
    case SW2:
        if((key & P5DR_B1) == 0)
            result = ON;
        break;
```

```
        case SW3:
            if((key & P5DR_B2) == 0)
                result = ON;
            break;
        case SW4:
            if((key & P5DR_B3) == 0)
                result = ON;
            break;
        default:
            break;
    }
    return result;
}
```

sense_switch関数はDIP-SWの対応するポート5の内容を読み込みます。B3からB0の対応のビットがゼロの時ONであり、1の時OFFの状態です。

get_switch関数は、引数で指定されたDIP-SWの現状の状態を**sense_switch**関数を使ってセンスし、戻り値としてONの状態ではONを、OFFの状態ではOFFを返す関数です。

ITRON仕様の概要

2005/12/19

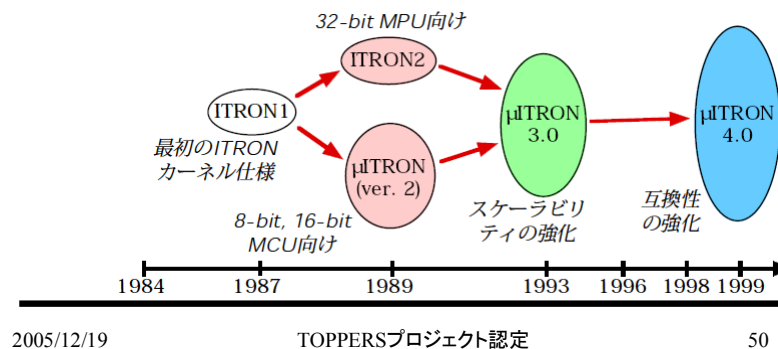
TOPPERSプロジェクト認定

49



ITRON仕様

- ITRONプロジェクト
 - TRONプロジェクトのサブプロジェクトの1つ
 - ITRON仕様
 - ITRONプロジェクトで策定されたリアルタイムカーネル仕様
 - これまでに4世代のITRON仕様に策定・公表
 - 現世代のリアルタイムカーネル技術の範囲では、完成度の高い仕様に
- ➡組込みシステム開発のオープン化に対する基盤



トロン(TRON: The Real-time Operating system Nucleus)は、理想的なコンピュータアーキテクチャの構築を目的として、1984年に東京大学の坂村健博士によって開始されたプロジェクトです。産業界と大学の協力のもとに、まったく新しいコンピュータの体系の実現を目指しています。トロンプロジェクト推進を行うに際して、実質的ないくつかのドメインを対象としてサブプロジェクト単位に、具体的な仕様の標準化を行ってきました。ITRON(組込みシステム用のリアルタイムOS仕様とその関連仕様)、BTRON(パソコンやワークステーション用のOS仕様とその関連仕様)、CTRON(通信制御や情報処理を目的としたOSインターフェース仕様)、トロン電子機器HMI(各種の電子機器のヒューマンマシンインターフェースの標準ガイドライン)等のサブプロジェクトです。

最初のITRON仕様は、1987年にITRON1仕様という形でまとめられました。ITRON1仕様に従っていくつかのリアルタイムカーネルが開発・応用され、仕様の適応性の検証に重要な役割を果たしました。その後、8～16ビットのMCUに適応するための機能を絞り込んだμITRON仕様(Ver.2.0)、逆に大規模な32ビットのプロセッサに適応するためのITRON2.0仕様の検討を進め、共に1989年に仕様を公開しました。μITRON仕様は限られたリソースのMCUでも実用的な性能を発揮できたことから多くのMCUに実装され、多くの組込み機器に応用されました。μITRON仕様が広域な分野に応用されるにつれて要求が明確化されたことと、μITRON仕様が32ビットMCUに応用され始め、8ビットから32ビットまでの各規模のMCUに適応できるスケーラビリティを持った仕様の要求が高まったことから1993年にμITRON3.0仕様として策定しなおし公開を行いました。

その後、ITRON TCP/IP API仕様やJTRON仕様の策定を経て、μITRON3.0仕様の見直しが指摘され、1999年にμITRON4.0仕様が公開されました。μITRON4.0仕様は組込みシステム開発のオープン化に対する基盤となるべく構築された仕様です。

ITRON仕様の特徴

- OSの小型軽量化が可能
 - ワンチップマイコンにも適用可能
- 仕様の理解が容易
 - 技術者教育のための標準化の側面を重視
- 完全にオープンな標準仕様
 - ロイヤリティなしで実装することができる
- 多種多様なプロセッサ用に実装できる/されている
 - 8bitワンチップマイコンから32bit RISCマイコンまで
 - 異なるプロセッサへの移行が容易に
- 多くの機器で使用実績がある
 - 組み込みシステム分野で最も広く使われているOS仕様
- 多くのメーカ/ベンダがサポート



2005/12/19

TOPPERSプロジェクト認定

51



ITRON仕様の特徴について列記します。

- 1) OSの小型軽量化が可能であり、ワンチップマイコンでも、実質的な性能を発揮することができます。
- 2) 仕様の理解が容易です。これは、技術者の教育のための標準化の側面を重視して仕様策定した結果です。
- 3) 完全にオープンな標準仕様であり、ロイヤリティなしで実装することができます。
- 4) 8ビットワンチップマイコンから32ビットのRISCプロセッサまで、多種多様なプロセッサに対して実装可能であり、すでに、実装されている。そのため、異なるプロセッサへの移行は容易である。
- 5) 多くの組み込み機器で使用実績がある。
- 6) 多くのメーカやベンダでサポートしている。

ITRON仕様のカーネルの機能(μ ITRON4.0仕様)

- | | |
|---|--|
| <ul style="list-style-type: none"> ■ カーネルの機能 <ul style="list-style-type: none"> – タスク管理機能 – タスク付属同期機能 – タスク例外処理機能 – 同期・通信機能 – 拡張同期・通信機能 – メモリプール機能 – 時間管理機能 – システム状態管理機能 – 割り込み管理機能 – サービスコール管理機能 | <ul style="list-style-type: none"> ■ サービスコール数 <ul style="list-style-type: none"> – フルセット <ul style="list-style-type: none"> • サービスコール : 166 • 静的API : 21 – スタンダードプロファイル <ul style="list-style-type: none"> • サービスコール : 70 • 静的API : 11 – 自動車制御用プロファイル <ul style="list-style-type: none"> • サービスコール : 43 • 静的API : 8 – 最小セット |
|---|--|



! I/O操作のための機能は規定されていない

2005/12/19

TOPPERSプロジェクト認定

52



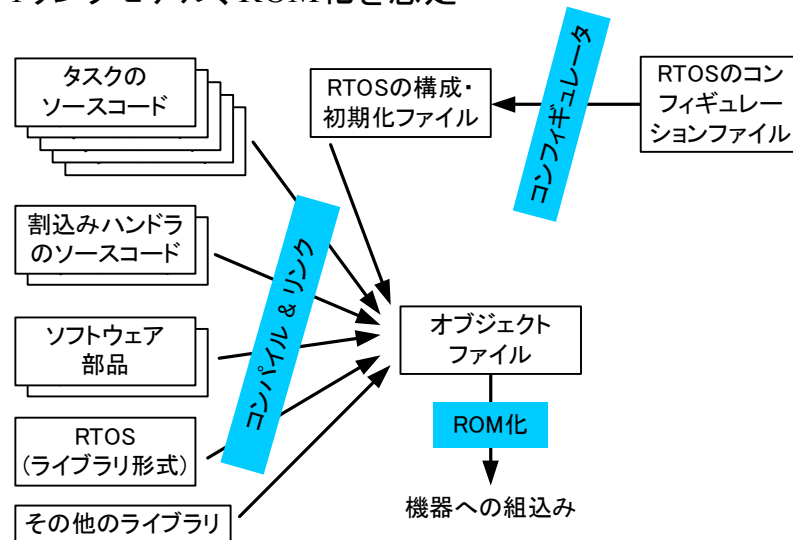
μ ITRON4.0仕様について説明を行います。カーネルの機能は以下のものがあります。

- 1) タスク管理機能: マルチタスク機能を管理する基本的な機能
- 2) タスク付属同期機能: タスク機能に用意された基本的な同期機能
- 3) タスク例外処理機能: タスクに発生した例外事象の処理をタスクコンテキストで実行するための機能
- 4) 同期・通信機能: タスクとは独立したオブジェクトにより、タスク間の同期・通信を行う機能
- 5) 拡張同期・通信機能: タスクとは独立したオブジェクトにより、高度なタスク間の同期・通信を行う機能
- 6) メモリプール機能: ソフトウェアによって動的なメモリ管理を行う機能
- 7) 時間管理機能: 時間に依存した処理を行う機能
- 8) システム状態管理機能: システムの状態を変更・参照する機能
- 9) 割り込み管理機能: 外部割り込みによって起動される割り込みハンドラおよび割り込みサービスルーチンを管理するための機能
- 10) サービス管理機能: 拡張サービスコールの定義と呼び出しを行うための機能

μ ITRON4.0仕様には、より良いスケーラビリティを実現するために、フルセットと最小セットの間に、スタンダードプロファイルと自動車制御用プロファイルを設けている。

RTOSを用いた組み込みソフトウェアの構築方法

■ 1リンクモデル、ROM化を想定



2005/12/19

TOPPERSプロジェクト認定

53



RTOSを用いた組み込みソフトウェアの構築方法を説明します。基本的に1リンクモデルとし、ROM化を想定した方法について説明を行います。

ユーザープログラムは、タスクのソースコードとして作成します。割り込みに関する処理は割り込みハンドラのソースコードとして記述します。これらのプログラムに、ソフトウェア部品とライブラリ化されたRTOSとその他のライブラリを加え、RTOSのコンフィギュレーションファイルから生成されるRTOSの構成・初期化ファイルをコンパイル・リンクしてオブジェクトファイル化しROM化して機器に組み込みを行います。

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、
 - 用いる機能/用いない機能
 - 各オブジェクト(タスクやセマフォ)などの最大値
 - タスク優先度の段数
 - **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のコードを変える方法(条件コンパイルなどを使う)
 - コンフィギュレーション結果を1つ(または複数)のソースコードに生成する方法

2005/12/19

TOPPERSプロジェクト認定

54



RTOSの構成や初期状態を決める手順をRTOSのコンフィギュレーションと言います。どのような構成にするか、また、初期状態にするかはRTOS毎に異なります。例えば、RTOSとして用いる機能や用いない機能を変更したり、オブジェクトの最大数を設定したり、優先度の段階を設定したり、静的にオブジェクトを生成を行う場合は生成定義情報を設定することができます。

コンフィギュレーションによりRTOSのコードに変更を加え、コンフィギュレーション結果をひとつまたは複数のソースコードとして生成する機構をコンフィギュレーション機構と呼びます。

μITRON4.0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の中での記述方法を標準化(静的API)
- 静的APIの例

```
CRE_TSK(tskid, {tskatr, exinf, task, itskpri, stksz, stk});  
ATT_ISR({intatr, exinf, intno, isr});
```

- コンフィギュレータは静的APIを解析して、カーネルの内部情報を生成する

μITRON4.0仕様では、カーネルオブジェクトを静的に生成することが出来ます。コンフィギュレーションファイルを作成し、このファイル中にオブジェクトの生成情報を静的APIで記述しておくことにより、コンフィギュレータがソースコードを自動生成してくれます。

静的APIの例としては、CRE_TSKは静的に生成するタスクオブジェクトの属性を指定するものです。

マルチタスク機構

■ スケジューリング方式

- μ ITRON4.0では、プリエンプティブな優先度ベーススケジューリングによるマルチタスク機構をサポート
- 同優先度のタスクはFCFS(First Come First Served)方式でスケジューリング。同優先度のタスク内での実行順序を変更する機能を用意。これを用いて、同優先度内でのラウンドロビンスケジューリングも実現可能
- 優先度付けは静的が基本だが、優先度を動的に変更する機能も用意

■ タスク状態

- 実行、実行可能、休止、待ち、強制待ち、二重待ち、未登録の7つの状態をサポート

2005/12/19

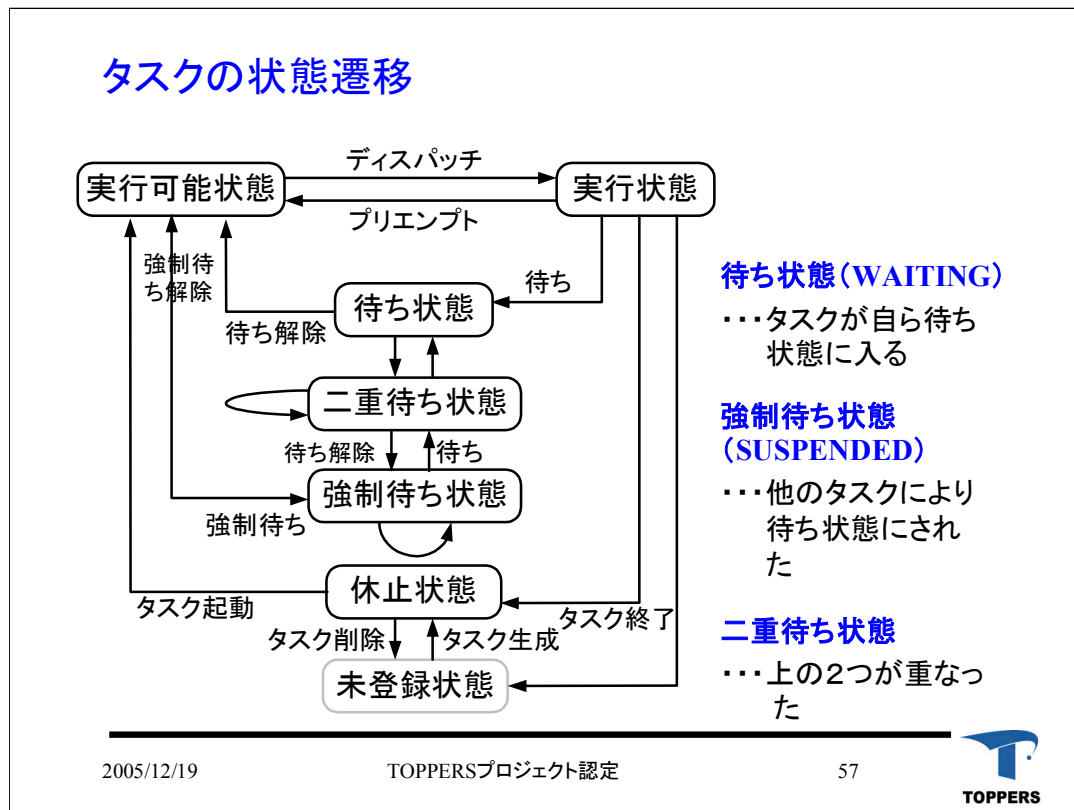
TOPPERSプロジェクト認定

56



μ ITRON4.0仕様のマルチタスク機構について説明を行います。タスクのスケジューリング方式はプリエンプティブな優先度ベーススケジューリングです。同一優先度のタスクはキューイングされ、FCFS (First Come First Served) 方式で実行します。同一優先度でもキューイング順番を回転させる機能 (rot_rdq サービスコール) により実行順序の変更が可能です。優先度についても優先度変更機能 (chg_pri サービスコール) により動的に変更が可能です。

タスクの状態は、実行、実行可能、休止、待ち、強制待ち、二重待ち、未登録の7の状態をサポートしています。



タスクの7つの状態

1) 実行状態 (RUNNING)

現在そのタスクが実行中であるという状態。但し、非タスクコンテキストを実行している場合は、基本的に非タスクコンテキストの実行開始前に実行していたタスクを実行状態にあるものとする。

2) 実行可能状態 (READY)

そのタスクは実行可能であるが、そのタスクより優先順位の高いタスクが実行中であるために、そのタスクが実行できない状態。

3) 待ち状態 (WAITING)

何らかの条件が整うまで自タスクの実行を中断するサービスコールを呼び出されたことにより、実行が中断された状態

4) 強制待ち状態 (SUSPENDED)

他のタスクによって、強制的に実行を中断させられた状態、但し μ ITRON4.0仕様では、自タスクを強制待ち状態にすることも出来る。

5) 二重待ち状態 (WAITING-SUSPENDED)

待ち状態と強制待ち状態が重なった状態、待ち状態のタスクに対して、強制待ち状態への移行が行われると、二重待ち状態に移行する

6) 休止状態 (DORMANT)

タスクがまだ起動されていないか、実行を終了した状態

7) 未登録状態 (NON-EXISTENT)

タスクがまだ生成されていないか、削除された後のシステムに登録されていない仮想的に状態

ITRON API解説

1. [タスク関連のAPI](#)
2. プログラムによる確認
3. 同期付属機能
4. プログラムによる同期付属機能の確認

ITRON仕様のサービス・コール(API)

■ サービス・コール名称のガイドライン

例) **act**_tsk

操作対象を表す。この場合はタスク(task)
操作方法を表す。この場合は起動(activate)

- サービス・コールが成功するとE_OKが戻り値として返される

■ 割込みハンドラ(非タスクコンテキスト)から呼び出せるサービス・コールは"i"で始まる名称として区別

例) **iact**_tsk

■ 割込みハンドラから呼び出せないAPIがある

- 待ち状態に入るAPI

サービス・コールとは、タスクなどのアプリケーション・プログラムからリアルタイム・カーネルに要求を出すために使うインターフェースで、一般にはAPIと呼ばれます。 μ ITRON3.0仕様までは「システム・コール」と呼ばれていました。ITRON仕様のサービス・コールの名称にはガイドラインがあり、前半が操作方法を現し、後半は操作対象を表します。サービスコール「act_tsk」では、actは起動を表すactivateの省略、tskはタスクの省略です。サービス・コールが正常終了するとE_OKが戻り値として返されます。

割込みハンドラ(非タスクコンテキスト)から呼び出されるサービス・コールは"i"で始まる名称として区別されます。非タスクコンテキストは、待ち状態がないため、待ち状態を要求するサービス・コールは呼び出せません。

静的APIの詳細

CRE_TSK	タスク生成 (静的API)	【スタンダード】
cre_tsk	タスク生成	【スタンダード外】

【静的API】

CRE_TSK (ID tskid, {ATR tskatr, VP_INT exinf, FP task,
PRI itskpri, SIZE stksz, VP stk})

【C言語API】

ER ercd = cre_tsk(ID tskid, T_CTSK *pk_ctsk)

【パラメータ】

ID	tskid	生成対象のタスクのID番号
T_CTSK	*pk_ctsk	タスク生成情報を入れたパケット
ATR	tskatr	タスク属性 (記述言語、初期状態)
VP_INT	exinf	タスクの拡張情報 (タスクへのパラメータ)
FP	task	タスクの起動番地
PRI	itskpri	タスクの起動優先度
SIZE	stksz	タスクのスタック領域のサイズ (バイト数)
VP	stk	タスクのスタック領域の先頭番地

2005/12/19

TOPPERSプロジェクト認定

60



μ ITRON4.0仕様では、システムを構成するオブジェクトを静的に生成するための方法 (コンフィギュレータ) が標準化され、静的APIが規定されました。静的APIとは、コンフィギュレーション・ファイル内に記述する。オブジェクトの生成内容を示すAPIです。C言語APIのサービス・コールの名称は小文字で記述しますが、静的APIの名称は大文字で記述します。パラメータはC言語APIと基本的に同じですが、C言語APIで構造体として指定するパラメータは、静的APIではそのままの形で記述します。

タスク管理機能

タスクの状態を直接的に操作するための機能

■ タスク管理機能のサービス・コール

cre_tsk	タスクの生成
del_tsk	タスクの削除
act_tsk、iact_tsk	タスクの起動
can_act	タスクの起動要求の無効化
ext_tsk	自タスクの終了
ter_tsk	タスクの強制終了
chg_pri	タスクの優先度の変更
get_pri	タスクの優先度の参照
ref_tsk	タスクの状態参照

■ タスクの起動要求(act_tsk)のキューイングとその無効化(can_act)

2005/12/19

TOPPERSプロジェクト認定

61



タスク管理機能は、タスクの状態を直接的に操作・参照するための機能です。タスクを生成・削除する機能、タスクを起動・終了する機能、タスクに対する起動要求をキャンセルする機能、タスクの優先度を変更する機能、タスクの状態を参照する機能が含まれます。タスクはID番号で識別されるオブジェクトです。タスクのID番号をタスクIDと呼びます。

タスクは、実行順序を制御するために、ベース優先度と現在優先度を持つ、単にタスクの優先度といった場合には、タスクの現在優先度を示します。タスクのベース優先度は、タスクの起動時にタスクの起動時優先度に初期化されます。タスクに対する起動要求はキューイングされます。つまり、すでに起動されたタスクを再起動しようとする、そのタスクを起動要求があったという記録が残り、後でそのタスクが終了した時に、タスクを自動的に再起動します。ただし、起動コードを指定してタスクを起動するサービスコール(sta_tsk)は、起動要求はキューイングされません。

リアルタイム・カーネルは、タスクの終了時に、ミューテックスのロック解除を除いては、タスクが取得した資源(セマフォ資源、メモリブロックなど)を開放する処理を行いません。これらの資源の解放はアプリケーションで行わなければなりません。

タスク付属同期機能

タスクを直接操作して同期を実現するための機能

■ タスク付属同期機能のサービス・コール

slp_tsk、tslp_tsk	自タスクを起床待ち状態に
wup_tsk、iwup_tsk	他タスクの起床
can_wup	起床要求の無効化
rel_wai、irel_wai	タスクの待ち状態の強制解除
sus_tsk	タスクを強制待ちに
rsm_tsk、frsm_tsk	タスクを強制待ちからの再開
dly_tsk	自タスクを遅延

- タスクの起床要求(wup_tsk)のキューイングとその無効化(can_wup)
- タイムアウト付きの起床待ち(tslp_tsk)と自タスクの遅延(dly_tsk)

2005/12/19

TOPPERSプロジェクト認定

62



タスク付属同期機能は、タスクの状態を直接に操作することにより同期を行う為の機能です。タスクを起床待ちにする機能とそこから起床する機能、タスクの起床要求をキャンセルする機能、タスクの待ち状態を強制解除する機能、タスクを強制待ち状態へ移行する機能とそこから再開する機能、自タスクの実行を遅延する機能が含まれています。

タスクに対する起床機能は、キューイングされます。つまり、起床待ち状態でないタスクを起床しようとする、そのタスクを起床要求が記録として残り、後でそのタスクが起床待ちに移行しようとした時に、タスクを起床待ちにしません。

タスクに対する強制待ち要求は、ネストされます。つまり、すでに強制待ち状態(二重待ち状態を含む)になっているタスクを再度強制待ち状態に移行させようとする、そのタスクを強制待ち状態に移行させようとしたという記録が残り、後でそのタスクを強制待ち状態(二重待ち状態を含む)から再開させようとした時に、強制待ちからの再開を行いません。

時間管理機能：概要

- システム時刻管理
 - システム時刻(カーネルが管理する現在時刻)を管理するための機能
- タイムイベントハンドラ
 - 時間の経過をきっかけに呼び出されるルーチン
 - 以下のタイマハンドラを用意
 - ・ 周期ハンドラ(周期的に呼ばれる)
 - ・ アラームハンドラ(指定した時刻に呼ばれる)
 - ・ オーバーランハンドラ(オーバーラン時に呼ばれる)
- 時間同期のためのその他の機能
 - タスクの遅延(タスクを一定時間待たせる)
 - 待ちに入るサービス・コールのタイムアウト機能

2005/12/19

TOPPERSプロジェクト認定

63



時間管理機能は、時間に依存した処理を行う為の機能です。システム時刻管理、周期ハンドラ、アラームハンドラ、オーバーランハンドラの各機能が含まれます。周期ハンドラ、アラームハンドラ、オーバーラン・ハンドラを総称してタイムイベントハンドラと呼びます。

・システム時刻管理

システム時刻管理機能は、システム時刻を操作するための機能です。システム時刻を設定・参照する機能、タイムスティックを供給してシステム時刻を更新する機能が含まれます。

システム時刻は、システム初期化時に0に初期化します。以降、アプリケーションによってタイムスティックを供給するサービスコール(`isig_tim`)が呼び出される毎に更新します。`isig_tim`を呼びだれる度にシステム時刻をどれだけ更新するかは、実装依存になります。但し、システム時刻を更新する機能をカーネル内部に持つことも可能で、その場合には`isig_tim`をサポートする必要はありません。

・タイムイベントハンドラ

周期ハンドラは、一定周期で起動されるタイムイベントハンドラです。周期ハンドラ機能には、周期ハンドラを生成・削除する機能、周期ハンドラの動作の開始・停止する機能、周期ハンドラの状態を参照する機能が含まれます。周期ハンドラはID番号で識別されるオブジェクトである。周期ハンドラのID番号を周期ハンドラIDと呼びます。周期ハンドラの起動周期と起動位相は、周期ハンドラの生成時に、周期ハンドラ毎に設定することができます。カーネルは、周期ハンドラの操作時に、設定された起動周期と起動位相から、周期ハンドラを次に起動すべき時刻を決定します。周期ハンドラの生成時には、周期ハンドラの生成時には、周期ハンドラを生成した時刻に起動位相を加えた時刻を、次に起動すべき時刻とします。周期ハンドラを起動すべき時刻になると、その周期ハンドラの拡張情報(`exinf`)をパラメータとして、周期ハンドラを起動します。また、このとき、周期ハンドラの起動すべき時刻に起動周期を加えた時刻を、次に起動すべき時刻を決定し直す場合があります。周期ハンドラの起動位相は、起動周期以下であることを原則とします。起動位相に起動周期よりも長い時間が指定された場合の動作は実装依存となります。周期ハンドラは、動作している状態か、動作していない状態かのいずれかの状態を取ります。周期ハンドラが動作していない状態の時は、周期ハンドラを起動すべき時刻になっても周期ハンドラを起動せず、次に起動すべき時刻の決定のみ行います。周期ハンドラの動作を開始するサービスコール(`sta_cyc`)が呼び出されると、周期ハンドラを動作している状態に移行させ、必要なら周期ハンドラを次に起動すべき時刻を決定しなおします。周期ハンドラの動作を停止させるサービスコール(`stp_cyc`)が呼び出されると、周期ハンドラを動作していない状態に移行させます。周期ハンドラを生成した後どちらの状態になるかは、周期ハンドラの属性によって決めることができます。

時間管理機能：システムコール

■ システム時刻管理のためのサービス・コール

set_tim	システムクロックの設定
get_tim	システムクロックの読み出し
isig_tim	タイムティック供給

■ 周期ハンドラ操作のためのサービス・コール

cre_cyc	周期ハンドラ生成
del_cyc	周期ハンドラの削除
sta_cyc	周期ハンドラの起動
stp_cyc	周期ハンドラ停止

2005/12/19

TOPPERSプロジェクト認定

64



アラームハンドラは、指定した時刻に起動されるタイムイベントハンドラです。アラームハンドラ機能には、アラームハンドラを生成・削除する機能、アラームハンドラの動作を開始・停止する機能、アラームハンドラの状態を参照する機能が含まれています。アラームハンドラはID番号で識別されるオブジェクトです。アラームハンドラのID番号をアラームハンドラIDと呼びます。

アラームハンドラを起動する時刻(これをアラームハンドラの起動時刻と呼びます)は、アラームハンドラ毎に設定することができます。アラームハンドラの起動時刻になると、そのアラームハンドラの拡張情報(exinf)をパラメータとして、アラームハンドラを起動します。アラームハンドラの生成直後には、アラームハンドラの起動時刻は設定されておらず、アラームハンドラの動作は停止状態です。アラームハンドラの動作を開始するサービスコール(sta_arm)が呼び出されると、アラームハンドラの起動時刻を、サービスコールが呼び出された時刻から指定された相対時刻後に設定します。アラームハンドラの動作を停止するサービスコール(stp_arm)が呼び出されると、アラームハンドラの起動時刻を解除しアラームハンドラの動作を停止します。

オーバランハンドラは、タスクが設定された時間を越えてプロセッサを使用した場合に起動されるタイムイベントハンドラです。オーバランハンドラ機能には、オーバランハンドラを定義する機能、オーバランハンドラの動作を開始・停止する機能、オーバランハンドラの状態を参照する機能が含まれます。

オーバランハンドラの起動条件として設定される時間(これをタスクの上限プロセッサ時間と呼びます)は、タスクごとに設定ができます。カーネルは上限プロセッサ時間が設定されたタスクに対して、上限プロセッサ時間が設定されて以降にそのタスクが使用した累積プロセッサ時間(これをタスクの使用プロセッサ時間と呼びます)を管理し、タスクの使用プロセッサ時間が上限プロセッサ時間を越えると、オーバランハンドラを起動します。システムに定義できるオーバランハンドラは1つだけであるため、オーバランハンドラには、起動の原因となったタスクのID番号(tsikid)と拡張情報(exinf)をパラメータとして渡します。タスクの生成直後には、タスクの上限プロセッサ時間は設定されていません。オーバランハンドラの動作を開始するサービスコール(sta_ovr)が呼び出されると、指定されたタスクの上限プロセッサ時間の設定を解除する。タスクの上限プロセッサ時間の設定は、そのタスクが原因となってオーバランハンドラを起動する時や、そのタスクが終了する時にも解除を行う。タスクが使用したプロセッサ時間には、少なくとも、そのタスク(タスク例外処理ルーチンを含む)とたそのタスクから呼び出したサービスコールの実行時間が含まれる。逆に、他のタスク(タスク例外処理ルーチンを含む)と他のタスクから呼び出したサービスコール実行時間は含まれません。

システム状態管理機能

システム状態を参照/変更するための機能

■ システム状態管理機能のサービス・コール

rot_rdq, irot_rdq	タスクの実行順序の回転
get_tid, iget_tid	実行状態のタスクのIDの取り出し
loc_cpu, iloc_cpu	CPUロック状態に
unl_cpu_iunl_cpu	CPUロック状態の解除
dis_dsp	ディスパッチ禁止
ena_dsp	ディスパッチ許可
sns_ctx	非タスクコンテキスト実行中か?
sns_loc	CPUロック状態か?
snd_dsp	ディスパッチ禁止状態か?
sns_dpn	ディスパッチ保留状態か?

システム状態管理機能は、システムの状態を変更・参照するための機能です。タスクの優先順位を回転する機能、実行状態のタスクIDを参照する機能、CPUロック状態への移行・解除を行う機能、タスクのディスパッチを禁止・解除する機能、コンテキストやシステム状態を参照する機能などがあります。

割込み処理と割込み管理機能

- 割込み処理(外部割込み)
 - 割込みハンドラをアプリケーション側で記述できる
 - 割込みが発生すると、カーネル内の出入り口処理を経由して、割込みハンドラが呼び出される
 - 割込みハンドラの中でサービス・コールを呼び出して、タスクの起動/起床などが可能
 - 割込みハンドラ処理はタスクよりも優先
 - ・ ディスパッチが遅延する

- 割込み管理機能のサービス・コール

`def_int` 割込みハンドラの定義

- この他に、割込みを個別に禁止/許可する機能など

2005/12/19

TOPPERSプロジェクト認定

66



割込み管理機能は、外部割込みによって起動される割込みハンドラおよび割込みサービスルーチンを管理する為の機能です。割込みハンドラを定義する機能、割込みサービスルーチンを生成・削除する機能、割込みサービスルーチンの状態を参照する機能、割込みを禁止・許可する機能、割込みマスクを変更・参照する機能が含まれます。割込みサービスルーチンはID番号で識別されるオブジェクトである。割込みサービスルーチンのID番号は割込みサービスルーチンIDと呼びます。

割込み管理機能では、以下のデータ型を持ちます。

INHNO 割込みハンドラ番号

INTNO 割込み番号

IXXXX 割込みマスク

割込みマスクのデータ型の一部のXXXXは、実装定義でターゲットプロセッサのアーキテクチャに応じた適切な文字列に定めます。割込みハンドラの記述形式は実装依存です。割込みサービスルーチンを起動する際には、その割込みサービスルーチンの拡張情報(exinf)をパラメータとして渡します。割込みサービスルーチンのC言語による記述形式は以下の通りです。

```
void isr (VP_INT exinf)
{
    割込みサービスルーチン本体
}
```

ITRON API解説

1. タスク関連のAPI
- [2. プログラムによる確認](#)
3. 同期付属機能
4. 同期付属機能プログラミング

サンプルによる確認

sample1を題材にしてJSPカーネルの
コンフィギュレーション方法、スケジューリング及び
タスク関連のAPIの使い方を学ぶ

- プログラムのディレクトリの ./jsp-1.4.1/OBJ/AKIH8_3069F/sample1に移動
- make realcleanを実行して初期状態に戻しファイルを確認
- make dependを実行して何が実行され何が生成されるのか確認

```
$ cd ./jsp-1.4.1/OBJ/AKIH8_3069F/sample1
$ make realclean
$ ls
Makefile* sample1.c sample1.cfg sample1.h
$ make depend
h8300-hms-gcc -E -I -I../include -I../config/h8/akih8_3069f -I../config/h8
ig/h8 -DCPU_CLOCK=20000000 -DLABEL_ASM -DAKI_MONITOR
-x c-header sample1.cfg > tmpfile1
../config/cfg -s tmpfile1 -c -obj h8 -system akih8_3069f
rm -f tmpfile1
Generating Makefile.depend.
$ls
Makefile* kernel_cfg.c kernel_id.h sample1.c sample1.h
Makefile.depend kernel_chk.c kernel_obj.dat sample1.cfg
```

2005/12/19

TOPPERSプロジェクト認定

68



sample1を題材にして、JSPカーネルを用いた構築の手順を確認します。sample1ディレクトリ上で以下のコマンドを実行します。

make realclean	構築ファイルのクリア
ls	初期状態のディレクトリを確認
make depend	依存関係の構築、カーネル関係のコンフィギュレータの実行
ls	ファイルの確認

以下のファイルが新たに生成されています。

Makefile.depend、kernel_cfg.c、kernel_id.h、kernel_chk.c、kernel_obj.dat

Makefile.dependはMakefileが生成したソースとインクルードファイルの依存関係を記述したファイルです。その他のファイルはコンフィギュレータにより、生成されたファイルです。

コンフィギュレーション：生成ファイル

- H8用のGCCを -E オプションで実行。-E オプションはプリプロセッサのみを実行する
- sample1.cfg をプリプロセッサに通して tmpfile1 にリダイレクト

```
h8300-hms-gcc -E -I. -I../include -I../config/h8/akih8_3069f -I../conf
ig/h8 -DCPU_CLOCK=20000000 -DLABEL_ASM -DAKI_MONITOR
-x c-header sample1.cfg > tmpfile1
```

- 生成したtmpfile1に対してコンフィギュレータ(cfg)を実行する

```
../cfg/cfg -s tmpfile1 -c -obj -cpu h8 -system akih8_3069f
```

- コンフィギュレータは以下のファイルを生成する、コンフィギュレーションに関するものは kernel_cfg.c と kernel_id.h の二つのファイル

```
kernel_cfg.c kernel_id.h kernel_chk.c kernel_obj.dat
```

TOPPERS/JSPでのコンフィギュレーションでは、どのようなことが行われるのでしょうか？まず、コンフィギュレーションファイル(sample1.cfg)をプリプロセッサを通してtmpfile1ファイルに変換します。プリプロセッサは、インクルードファイル中の定義を実数に変換し、コンフィギュレーションしやすいデータ形式に変換してくれます。tmpfile1ファイルを対象に、コンフィギュレータ(jsp-1.4.1/cfg/cfg)がC言語でコンパイル可能なインクルードファイル(kernel_id.h)とソースファイル(kernel_cfg.c)を生成します。kernel_id.hはカーネルオブジェクトの定義用のインクルードファイルであり、kernel_cfg.cはカーネルオブジェクトのデータ領域定義やテーブル、初期化関数のソースファイルです。

コンフィギュレーション : sample1のコンフィギュレーションファイル

- コンフィギュレータはC言語記述を読み込めないため、`_MACRO_ONLY`を定義。C言語記述はこのマクロ中に書く。
- sample1のコンフィギュレーションファイル(sample1.cfg)では、タスクを4つと周期ハンドラを1つを静的APIで生成している
- 最後の3行でインクルードしているのはシステムクロックドライバ、リアルタイムインターフェースドライバ、システムログタスクのコンフィギュレーションファイル

```
#define _MACRO_ONLY
#include "sample1.h"
INCLUDE("%sample1.h%");
CRE_TSK(TASK1, { TA_HLNG, (VP_INT) 1, task, MID_PRIORITY, STACK_SIZE, NULL });
CRE_TSK(TASK2, { TA_HLNG, (VP_INT) 2, task, MID_PRIORITY, STACK_SIZE, NULL });
CRE_TSK(TASK3, { TA_HLNG, (VP_INT) 3, task, MID_PRIORITY, STACK_SIZE, NULL });
CRE_TSK(MAIN_TASK, { TA_HLNG|TA_ACT, 0, main_task, MAIN_PRIORITY,
                        STACK_SIZE, NULL });

....
CRE_CYC(CYCHDR1, { TA_HLNG, 0, cyclic_handler, 2000, 0 });

#include "../systask/timer.cfg"
#include "../systask/serial.cfg"
#include "../systask/logtask.cfg"
```

2005/12/19

TOPPERSプロジェクト認定

70



sample1.cfgを対象にコンフィギュレーションファイルの説明を行います。コンフィギュレータは、C言語のプロファイルのような関数記述を解析の対象としていないため、コンフィギュレーションファイルのインクルードファイル中にこのような記述がある場合、削除された方が正しく変換されます。コンフィギュレーションファイルの先頭で、`_MACRO_ONLY`を定義し、プリプロセッサを通すことにより、`#ifndef`定義で囲まれたC言語記述を削除します。sample1.cfgでは4つのタスクとひとつの周期ハンドラを定義しています。

sample1.cfg中の最後のインクルードファイルはシステムで使用するドライバやカーネルの拡張機能用のコンフィギュレーションファイルです。

timer.cfg	システムタイマーのドライバのコンフィギュレーションファイル
serial.cfg	TOPPERS/JSPで用いるシリアルドライバのコンフィギュレーションファイル
logtask.cfg	syslog用のコンフィギュレーションファイル

コンフィギュレーション : kernel_id.h

- kernel_id.h には、オブジェクトのID自動割付け結果が定義されている
- ユーザープログラムではAPIで用いるIDをマクロで記述しておき、このファイルをインクルードする

- kernel_id.h

```
#define CYCHDR1 1
#define LOGTASK 5
#define MAIN_TASK 4
#define SERIAL_RCV_SEM1 1
#define SERIAL_RCV_SEM2 3
#define SERIAL_SND_SEM1 2
#define SERIAL_SND_SEM2 4
#define TASK1 1
#define TASK2 2
#define TASK3 3
```

- コンフィギュレーションファイル

```
CRE_TSK(TASK1, { TA_HLNG, ...
CRE_TSK(TASK2, { TA_HLNG, ...
CRE_TSK(TASK3, { TA_HLNG, ...
CRE_TSK(MAIN_TASK, { TA_HLNG ...
```

- ユーザープログラム

```
/*
 * タスクの起動
 */
act_tsk(TASK1);
act_tsk(TASK2);
act_tsk(TASK3);
```

2005/12/19

TOPPERSプロジェクト認定

71



kernel_id.hは、カーネルオブジェクトのIDの自動割付け結果が定義されています。上記の例ではコンフィギュレーションファイル中のCRE_TSK静的APIの宣言に数値の割り当てが行われています。

TASK1には1、TASK2には2、TASK3には3、MAIN_TASKには4になっています。LOGTASKの5はlogtask.cfgで定義されているロギングタスクのようなタスクIDです。周期ハンドラ用のIDであるCYCHDR1にも数値が割り当てられています。これらの割り当てはkernel_id.hをインクルードすることにより、ユーザープログラムでも、実数としてコンパイルすることができるようになります。

コンフィギュレーション : kernel_cfg.c

- カーネル構成・初期化ファイル
- 静的APIによる生成情報から作成された、カーネルの内部データや初期化ルーチン
- ユーザープログラム同様にコンパイル・リンクする

```
#define TNUM_TSKID 5
const ID _kernel_tmax_tskid = (TMIN_TSKID + TNUM_TSKID - 1);
static __STK_UNIT __stack_TASK1[_TCOUNT_STK_UNIT(1024)];
static __STK_UNIT __stack_TASK2[_TCOUNT_STK_UNIT(1024)];
static __STK_UNIT __stack_TASK3[_TCOUNT_STK_UNIT(1024)];
static __STK_UNIT __stack_MAIN_TASK[_TCOUNT_STK_UNIT(1024)];
static __STK_UNIT __stack_LOGTASK[_TCOUNT_STK_UNIT(LOGTASK_STACK_SIZE)];
const TINIB _kernel_tinib_table[TNUM_TSKID] = {
    {0, (VP_INT)((VP_INT) 1), (FP)(task), INT_PRIORITY(10), ...},
    {0, (VP_INT)((VP_INT) 2), (FP)(task), INT_PRIORITY(10), ...},
    {0, (VP_INT)((VP_INT) 3), (FP)(task), INT_PRIORITY(10), ...},
    {0x00u | 0x02u, (VP_INT)(0), (FP)(main_task), INT_PRIORITY(5), ...},
    {0x00u | 0x02u, (VP_INT)((VP_INT) 2), (FP)(logtask), ..}
};
const ID _kernel_torder_table[TNUM_TSKID] = {1,2,3,4,5};
TCB _kernel_tcb_table[TNUM_TSKID];
```

2005/12/19

TOPPERSプロジェクト認定

72



kernel_cfg.cはカーネル構成・初期化ファイルです。コンフィギュレーションファイルから生成したカーネルの内部データ、カーネルオブジェクトのデータ定義や初期化ルーチン等の記述が生成されています。このファイルはユーザープログラムと同時にコンパイル・リンクされます。

上記の例では、**_kernel_tmax_tskid**はタスクIDの最大数のコンスタントデータ、**_stack_TASK1**は並行実行されるタスク(task)のタスクID1 (TASK1)用のスタック領域、**_kernel_tinib_table**はTCB(タスク・コントロール・ブロック)の初期化テーブル、**_kernel_tcb_table**はTCBのデータ領域です。

静的API : 実行状態でのタスク生成

- ディレクトリ `./jsp-1.4.1/OBJ/AKIH8_3069F/task_api/` を作成し、`configure` を実行して `sample1` のプロジェクトを作成
- 並列実行するTASK1、TASK2、TASK3は休止状態で生成され、メインタスクで `act_tsk()` を実行することにより実行可能状態にしている
- `sample1` を以下のように変更する
 - メインタスクでは `act_tsk()` を呼び出さない
 - 静的APIを変更して三つのタスクを実行可能状態で生成する
- また、並列実行するタスクの数を1つ増やしてみましょう

2005/12/19

TOPPERSプロジェクト認定

73



`sample1`を改造してみましょう。`task_api`ディレクトリを作成し、`sample1`プロジェクトを生成します。このプロジェクトのプログラムを以下のように修正します。

1) 現状のプログラムでは、TASK1、TASK2、TASK3の3つの並列実行タスクは休止状態で生成され、メインタスク中の`act_tsk()`サービスコールにより実行状態に移行させています。メインタスクとコンフィギュレーションファイルを改造してメインタスクで`act_tsk()`を実行しなくてもよいように、3つのタスクが実行状態で生成されるように修正しましょう。

2) 並列実行するタスクの数を3つから4つに増やしましょう。

静的API: 実行状態でのタスク生成

- CRE_TSKのタスク属性にTA_ACTを指定すると、対象タスクを実行可能状態に移行させ、タスクの生成時に行うべき処理に加えて、タスクの起動時に行うべき処理を行う。タスクの状態は、実行可能状態となる。

```
CRE_TSK(TASK1, { TA_HLNG|TA_ACT, (VP_INT) 1,
                 task, MID_PRIORITY, STACK_SIZE, NULL });
CRE_TSK(TASK2, { TA_HLNG|TA_ACT, (VP_INT) 2,
                 task, MID_PRIORITY, STACK_SIZE, NULL });
CRE_TSK(TASK3, { TA_HLNG|TA_ACT, (VP_INT) 3,
                 task, MID_PRIORITY, STACK_SIZE, NULL });
```

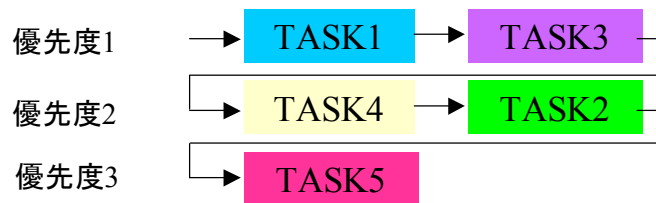
並列実行タスクを自動実行させるには、コンフィギュレーションファイル中のCRE_TSK静的APIの属性にTA_ACTを追加すると、生成時自動実行します。

並列実行タスクの数を増やすには、コンフィギュレーションファイルにTASK4用のCRE_TSK宣言を追加します。これだけでは、メインタスクからTASK4に対する実行要求が行えません。sample1.cファイルの改造が必要です。messageの領域を4つに増やし、main_task関数にact_tsk(TASK4);の追加、switch文中に以下の記述を追加します。

```
case '4':
    tskno = 4;
    tskid = TASK4;
    break;
```

スケジューリング：レディキュー

- 実行可能状態のタスクを管理するためのOS内のキュー
- 優先度毎のキューが優先度の順に繋がっている
 - 同一優先度のキューは起動した順に接続されている
- OSはレディーキューの先頭のタスクを実行する(実行状態にする)



2005/12/19

TOPPERSプロジェクト認定

75



レディキューはRTOS内で、実行可能状態のタスクを管理するためのキューです。レディキューは優先度毎のキューが優先度準で接続されています。

RTOSはスケジューリングの際、レディキューをチェックして、先頭のタスクを実行する。

スケジューリング：

- sample1で並列実行される3つのタスクは、優先度 MID_PRIORITY(10) の初期優先度で実行される
- sample1を動作させ、コンソールからのコマンド入力を行い以下を確認する
 - 起動時のMID_PRIORITYのレディキューの状態
 - ‘<’コマンドにて、TASK1の優先度をLOW_PRIORITY(11)に下げた場合のMID_PRIORITYのレディキューの状態
 - この状態で‘r’コマンドにて一度レディキューを回転させる
 - ‘=’コマンドにて、TASK1の優先度をMID_PRIORITYに戻した場合のMID_PRIORITYのレディキューの状態

2005/12/19

TOPPERSプロジェクト認定

76



sample1の実行後、コマンドの対象がTASK1になっている状態で、‘<’コマンドを入力してTASK1の優先度をLOW_PRIORITY(11)に設定します(‘<’コマンドはchi_pri(tskid, LOW_PRIORITY)サービスコールを発行する)。ログの表示はどのように変化するでしょうか。

その後、‘r’コマンドを入力してレディキューの回転を行います(‘r’コマンドはHIGH_PRIORITY、MID_PRIORITY、LOW_PRIORITYの3つのタスクレベルに対してrot_rdqサービスコールを発行してレディキューの回転を要求する)。ログ表示はどのように変わるでしょうか？

今度は、‘=’コマンドを入力してTASK1の優先度をMID_PRIORITY(10)に設定します(‘=’コマンドはchi_pri(tskid, MID_PRIORITY)サービスコールを発行する)。ログの表示はどのように変化するでしょうか。

スケジューリング：結果

■ 起動時のMID_PRIORITYのレディキューの状態



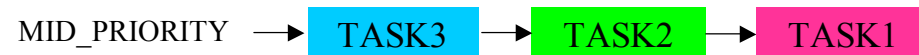
■ TASK1の優先度をLOW_PRIORITY(11)に下げた場合



■ レディキューを一度回す



■ TASK1の優先度を戻す



2005/12/19

TOPPERSプロジェクト認定

77



結果の検証を行います。sample1の起動直後はTASK1が実行するため、task1のログ表示が行われています。

1) '←'コマンドを入力すると、TASK1の優先度はLOW_PRIORITYとなるため、MID_PRIORITYのレディキューの先頭のTASK2が実行状態となります。従って、ログ表示はtask2に変わります。

2) 'r'コマンドを入力すると、MID_PRIORITYのレディキューが回転して、レディキューの先頭がTASK3に変わります。従って、ログ表示はtask3に変わります。

3) '='コマンドを入力すると、TASK1の優先度がMID_PRIORITYに変更され、MID_PRIORITYのレディキューの最後にTASK1がつながれます。そのため、ログ表示は変わりません。この時、'r'コマンドを2度入力すると、TASK1がレディキューの先頭になりますので、表示はtask1に変わります。

起動要求のキューイング

- タスクへの起動要求、起床要求、強制待ち要求はキューイングされる
- sample1を用いて起動要求のキューイングを確認する
 - ‘a’コマンドにて起動要求API `act_tsk()`をTASK1へ何回発行可能か?
 - 起動要求がキューイングされている状態でTASK1を終了’t’コマンド(`ter_tsk()`の発行)もしくは’e’コマンド(`ext_tsk()`の発行)により休止させると、TASK1はどのように振る舞うか?

sample1の起動後、‘a’コマンド(対応のタスクに対して`act_tsk`サービスコールを発行する)を入力します。TASK1が実行中でも1度はエラーとならないが、2度目はE_QOVRエラーが発生して受け付けません。TOPPERS/JSPでは起動中でも、起動要求は1度はキューイングされます。この状態で’t’コマンド(`ter_tsk`サービスコールの発行)または’e’コマンド(`ext_tsk`サービスコールの発行)にてTASK1を休止させても、自動的に起床状態に復帰します。

ITRON API解説

1. タスク関連のAPI
2. プログラムによる確認
- [3. 同期付属機能](#)
4. 同期付属機能プログラミング

タスク間の通信と同期

- タスク間の通信・同期の必要性
 - タスクが協調して動作するためには、タスク間でデータをやりとりすること(=タスク間通信)が必要
 - タスク間通信の際には、タスク同士で動作タイミングをあわせること(=タスク間同期)が必要
 - 複数のタスクが同一の資源を取り合う場合にも、タスク間の同期が必要
- タスク間通信・同期のタイプ

タスクを仮想化されたプロセッサと捉えるなら、タスク間の通信・同期はプロセッサ間の通信・同期を仮想化したもの

共有メモリによる通信 or メッセージによる通信

共有メモリによる通信

複数のタスクからアクセスできるメモリ領域上に
受け渡しするデータ(共有データ)を置く

– C言語のグローバル変数による実現

- 共有データを読み書きする際には、RTOSの機能を用いて、排他制御することが必要

– 割り込み/ディスパッチの禁止

– セマフォ/ミューテックス

 **デッドロックと優先度逆転に注意**

– タスクの優先度の変更

- 共有データを速やかに処理させたい場合には、事象の発生を知らせる機能を用いる

– タスクの起動/起床

– イベントフラグ/Condition Variable(条件変数)

実際のシステム設計で、タスク間通信を機能を構築する為のノウハウを説明します。通信方式は、大別して「共有メモリによる通信」と「メッセージによる通信」に分けることができます。

「共有メモリによる通信」では、タスク間の受け渡しデータをグローバルメモリ領域やシェアードメモリ上に確保します。この領域上に受け渡しデータを読み書きすることにより、タスク間通信を行います。各タスクや割り込みは非同期に動作しますので、あるタスクがデータを書き終わらないうちに、別のタスクがデータを読み取ると間違った情報を受け取ってしまう可能性があります。そこで、セマフォなどを使って排他制御することによって、このような状態が発生することを防ぐことができます。

また、事象が変化した場合、すぐに対応を行う必要がある場合はタスクの起動や起床、イベントフラグなどを使って受け取り側タスクをすぐに動作させる必要があります。

メッセージによる通信

RTOSが持つメッセージ通信のための機能を用いて、
タスク間でメッセージを受け渡しする

- メッセージ通信機能のタイプ
 - コネクションあり or なし、単方向 or 双方向、
1対1(送信できるタスクが固定) or n対1 or n対n
 - 通信相手の指定方法
 - 通信オブジェクトを指定 or 相手タスクを指定 or ……
 - 同期メッセージ通信 or 非同期メッセージ通信
 - (非同期通信で)バッファにメッセージが無い場合の振る舞い
 - 待つ(ブロッキング) or 待たない(ノンブロッキング)
 - (非同期通信で)バッファがフルの時の振る舞い
 - 待つ(ブロッキング) or 待たない(ノンブロッキング)
or 上書きする

2005/12/19

TOPPERSプロジェクト認定

82



「メッセージによる通信」では、メールボックスやデータキューなどのメッセージ機能を用いて構築します。メッセージの内容はシステム設定時によく吟味する必要があります。メッセージ機構は、排他制御をキチンと行えば自作することもできます。システム全体を考慮して最適な方法を構築してください。

メッセージ通信機能のタイプ(続き)

- メッセージが固定長 or 可変長(任意長)
- (可変長の時に)パケットの単位がある or ない
- ポインタを渡す or コピーする
- (非同期通信で)メッセージのキューイング順序
 - FIFO順 or 優先度順
- 受信/送信待ちタスクのキューイング順序
 - FIFO順 or 優先度順
- その他特殊な機能
 - マルチキャスト/ブロードキャスト
 - 状態メッセージ(OSEK/VDK仕様の unqueued message)

RTOSの持つメッセージ通信機能(複数の機能を持つ場合も多い)の性質をよく理解することが必要

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、通信の粒度が大きく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には、メッセージ通信が便利

2005/12/19

TOPPERSプロジェクト認定

84



共有メモリとメッセージ通信による通信同期の設計を比較してみましょう。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ます。しかし、方式により、得手不得手があります。通信によるオーバーヘッドを比較すると、きちんと設計すれば、共有メモリ方式の方がオーバーヘッドが小さくなります。また、割込みルーチンなどの非タスクコンテキストを用いてタスクに対して、要求を伝える設計をしたい場合も、共有メモリ方式の方が有利です。しかし、RTOSの機能を用いてシステム中の通信を監視しやすいシステムを構築したい場合は、メッセージ方式の方が構築しやすくなります。

通信同期の設計要求によって、片方の方式では達成しづらい場合もあります。例えば、情報の流れが1対nの場合や情報の伝達タイミングが不定の場合は、共有メモリ方式の方が、システム設計が容易になります。しかし、伝達情報（パラメータ）をキューイングして伝達することが必要なシステムでは、メッセージ方式の方が、格段にシステム設計しやすくなります。

システムの要求を考慮して適切な方式を選択する必要があります。しかし、同一システムに多くの通信手法を混在してしまうと、システムが複雑になりすぎて、拡張や障害対応が難しくなりますので、よく考慮してください。

ITRONの同期・通信機能

- μ ITRON4.0では、タスク間同期・通信のために以下の機能を用意
- (主に) 共有メモリによる通信のための機能
 - セマフォ(排他制御など)
 - イベントフラグ(事象通知)
 - ミューテックス(排他制御)
- メッセージ通信のための機能
 - データキュー(同期・非同期, 1ワードのメッセージ)
 - メールボックス(非同期, ポインタを送受信)
 - メッセージバッファ(同期・非同期, 可変長メッセージ)
 - ランデブ(同期、双方向に通信, 可変長メッセージ)

2005/12/19

TOPPERSプロジェクト認定

85



μ ITRONでは、タスク間同期・通信のために以下の機能を用意しています。

・共有メモリによる通信、排他制御のための機能

- セマフォ CRE_SEM、cre_sem、acre_sem、del_sem、sig_sem、isem_sig、wai_sem
pol_sem、twai_sem、ref_sem
- イベントフラグ CRE_FLG、cre_flg、acre_flg、del_flg、set_flg、iset_flg、clr_flg、wai_flg
pol_flg、twai_flg、ref_flg
- ミューテックス CRE_MTX、cre_mtx、acre_mtx、del_mtx、loc_mtx、ploc_mtx、tloc_mtx
unl_mtx、ref_mtx

・メッセージ通信のための機能

- データキュー CRE_DTQ、cre_dtq、acre_dtq、del_dtq、snd_dtq、psnd_dtq、ipsnd_dtq
tsnd_dtq、fsnd_dtq、ifsnd_dtq、rcv_dtq、prcv_dtq、trcv_dtq、ref_dtq
- メールボックス CRE_MBX、cre_mbx、acre_mbx、del_mbx、snd_mbx、rcv_mbx、prcv_dtq
trcv_dtq、ref_mbx
- メッセージバッファ
CRE_MBF、cre_mbf、acre_mbf、del_mbf、snd_mbf、psnd_mbf、tsnd_mbf
rcv_mbf、prcv_mbf、trcv_mbf、ref_mbf
- ランデブ CRE_POR、cre_por、acre_por、del_por、cal_por、tcal_por、acp_por
pacp_por、tacp_por、fwd_por、rpl_rdv、ref_por、ref_rdv

セマフォ、イベントフラグ、データキュー、メールボックスはスタンダードプロファイルとして用意しています。

ITRON同期・通信機能の比較

	セマフォ	イベントフラグ	メールボックス	データキュー
機能	排他制御・同期	イベント通知	同期・通信	同期・通信
情報量	少	中	多	多
オーバーヘッド	小	小	中～大	大
キューイングされるオブジェクト	資源待ちタスク	イベント待ちタスク	受信タスク メッセージ	送信タスク 受信タスク メッセージ
サービスコール	CRE_SEM, sig_sem, wai_sem, twai_sem, pol_sem	CRE_FLG, set_flg, clr_flg, wai_flg, pol_flg, twai_flg	CRE_MBX, snd_mbx, rcv_mbx, prcv_mbx, trcv_mbx	CRE_DTQ, snd_dtq, psnd_dtq, tsnd_dtq, fsnd_dtq, rcv_dtq, prcv_dtq, trcv_dtq

2005/12/19

TOPPERSプロジェクト認定

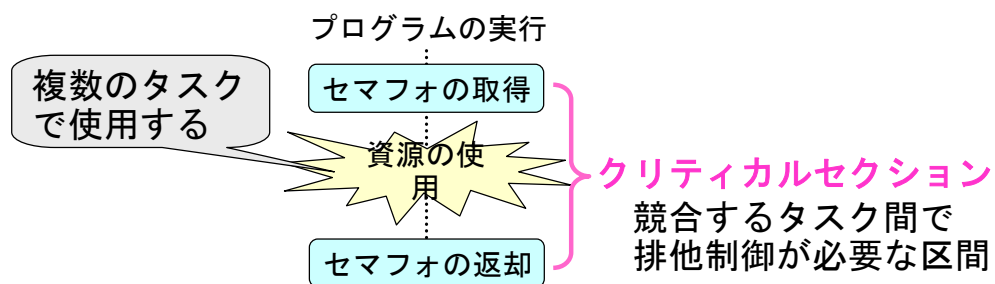
86



システム設計を行う場合、同じシステムに種々の通信・同期機能を混在して使用すると、システムが複雑になってしまう為、特別な事情がある場合を除いて避けた方が良いでしょう。なるべくシステム自体がシンプルとなるように、通信・同期機能を選択することをお勧めします。

セマフォ(Semaphore)

- 使用されていない資源の有無や数量をセマフォの資源数として管理することにより排他制御を実現
- 資源を使用する前にセマフォ資源を取得し、資源を使用した後にセマフォ資源を返却する。
- 資源が全て使用されていれば、セマフォ資源獲得の時点で待ち状態になる。



2005/12/19

TOPPERSプロジェクト認定

87



セマフォは、使用されていない資源の有無や数量をカウンタで表現することにより、その資源を使用する際の排他制御や同期を行うオブジェクトです。資源を解放する場合、セマフォのカウンタを+1し、資源を取得する場合、セマフォのカウンタを-1します。

セマフォはこのカウンタと資源の獲得待ちのタスクの待ち行列で構成されています。資源の獲得を繰り返すカウンタがゼロになると、使用されていない資源が無い為、資源の取得を要求したタスクはセマフォ資源の待ち行列につながれ、待ち状態に入ります。

セマフォには資源の初期値と、資源に対して資源が返却され過ぎるのを防ぐ為に最大資源数が設定できます。最大資源数を越えて資源の返却を行おうとした場合、エラーが発生します。セマフォIDは個々のセマフォを識別するためのIDです。

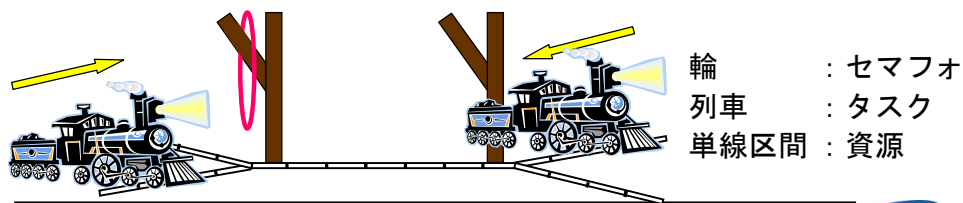
セマフォ(Semaphore) : システムコール&語源

- システムコール

cre_sem	セマフォの生成
del_sem	セマフォの削除
sig_sem, isig_sem	セマフォ資源の返却
wai_sem, pol_sem, twai_sem	セマフォ資源獲得

- 語源

鉄道の単線区間で進入制御に用いていた「輪」.
単線区間に入る場合は, この輪(セマフォ)を取る



2005/12/19

TOPPERSプロジェクト認定

88



μ ITRON4.0仕様のセマフォのサービス・コールは以下の通りです。

1. セマフォの生成

```
CRE_SEM(ID semid, {ATR sematr, UINT isemcnt, UINT maxsem});
```

ID semid セマフォID番号

ATR sematr セマフォ属性(TA_FIFO || TA_TPRI)

UINT isemcnt セマフォ資源数の初期値

UNIT maxsem セマフォの最大資源数

2. セマフォ資源の返却

```
ER ercd = sig_sem(ID semid); ER ercd = iseg_sem(ID semid);
```

ID semid 資源返却対象のセマフォ番号

3. セマフォ資源の取得

```
ER ercd = wai_sem(ID semid); ER ercd = pol_sem(ID semid);
```

```
ER ercd = twai_sem(ID semid, TMO tmout);
```

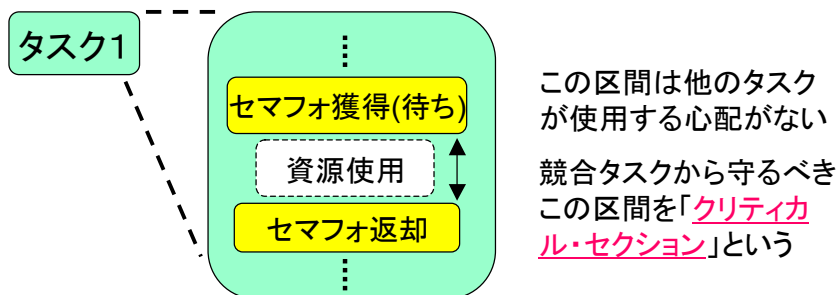
ID semid 資源獲得対象のセマフォ番号

TMO tmout タイムアウト指定 (twai_semのみ)

セマフォ(Semaphore) : 使い方

- 使用されていない資源の有無や数量を数値(セマフォの資源数)で管理することにより排他制御を実現
- システムコール

cre_sem	セマフォの生成
del_sem	セマフォの削除
sig_sem, isig_sem	セマフォ資源の返却
wai_sem, pol_sem, twai_sem	セマフォ資源獲得



2005/12/19

TOPPERSプロジェクト認定

89



μ ITRON4.0仕様のセマフォのサービス・コールは以下の通りです。

1. セマフォの生成

```
CRE_SEM(ID semid, {ATR sematr, UINT isemcnt, UINT maxsem});
```

ID semid	セマフォID番号
ATR sematr	セマフォ属性(TA_FIFO TA_TPRI)
UINT isemcnt	セマフォ資源数の初期値
UNIT maxsem	セマフォの最大資源数

2. セマフォ資源の返却

```
ER ercd = sig_sem(ID semid); ER ercd = iseg_sem(ID semid);
```

ID semid	資源返却対象のセマフォ番号
----------	---------------

3. セマフォ資源の取得

```
ER ercd = wai_sem(ID semid); ER ercd = pol_sem(ID semid);
```

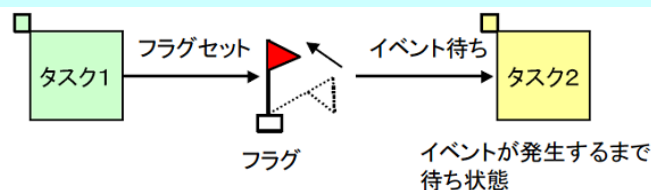
```
ER ercd = twai_sem(ID semid, TMO tmout);
```

ID semid	資源獲得対象のセマフォ番号
TMO tmout	タイムアウト指定 (twai_semのみ)

イベントフラグ (EventFlag)

- タスク間でイベントの発生を伝えることで同期するための機構。1つのイベントフラグは、複数のフラグで構成
- wai_flgでは、指定したフラグの全てがセットされたか、いずれかがセットされたかを待ち合わせ可能
- システムコール

cre_flg	イベントフラグの生成
del_flg	イベントフラグの削除
set_flg, iset_flg	イベントフラグのセット
clr_flg	イベントフラグのクリア
wai_flg, pol_flg, twai_flg	イベントフラグ待ち



2005/12/19

TOPPERSプロジェクト認定

90



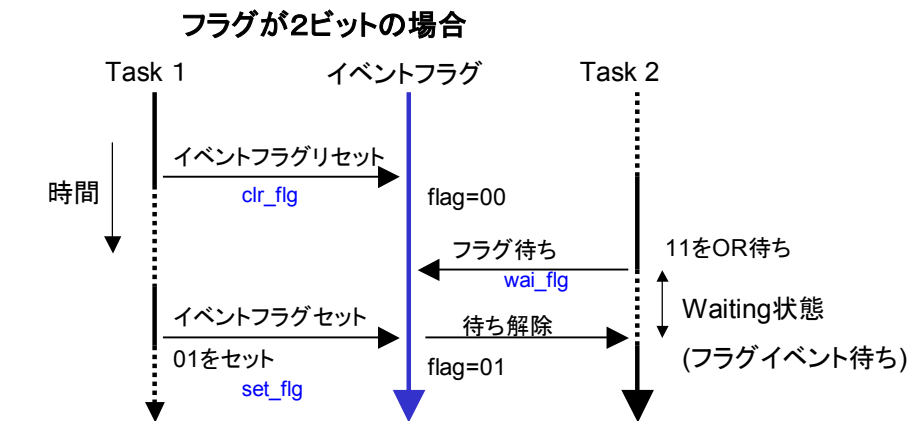
イベントフラグは、イベントの有無をビットごとのフラグで表すことにより、タスク間の同期を行うためのオブジェクトです。

イベントフラグには、対応するイベントの有無をビット毎に表現するビットパターンと、そのイベントフラグの設定を待つタスクの待ち行列で構成されています。イベントを通知する側は、イベントフラグのビットパターンを指定をしたビットをセットまたはクリアすることが可能です。一方、イベントを待つ側は、イベントフラグのビットパターンの指定したビットのすべてまたは一部がセットされるまで、タスクをイベント待ち状態にすることができます。イベント待ち状態を要求したタスクは、要求したイベントの待ち行列につながれます。

イベントフラグ機能には、イベントフラグを生成、削除する機能、イベントフラグをセット、リセットする機能、イベントフラグを設定を待つ機能、イベントフラグの状態を参照する機能で構成されています。

イベントフラグ(EventFlag) : タスクの実行状態

- イベントの有無をフラグのビットパターンで通知する



2005/12/19

TOPPERSプロジェクト認定

91



μ ITRON4.0のイベントフラグ機能のサービス・コールは以下の通りです。

1. イベントフラグの生成

CRE_FLG(ID flgid, {ATR flgatr, FLGPTN iflgptn});

ID flgid イベントフラグのID番号

ATR flgatr イベントフラグ属性((TA_TFIFO||TA_PRI) |
(TA_WSGL||TA_WMUL) [TA_CLR])

FLGPTN iflgptn イベントフラグのビットパターンの初期値

2. イベントフラグのセット

ER ercd = set_flg(ID flgid, FLGPTN setptn);

ER ercd = iset_flg(ID flgid, FLGPTN setptn);

3. イベントフラグのクリア

ER ercd = clr_flg(ID flgid, FLGPTN clrptn);

4. イベントフラグ待ち

ER ercd = wai_flg(ID flgid, FLGPTN waiptn, MODE wfmode, FLGPTN *p_flgptn);

ER ercd = pol_flg(ID flgid, FLGPTN waiptn, MODE wfmode, FLGPTN *p_flgptn);

ER ercd = twai_flg(ID flgid, FLGPTN waiptn, MODE wfmode, FLGPTN *p_flgptn, TMO tmout);

ITRON API解説

1. タスク関連のAPI
2. プログラムによる確認
3. 同期付属機能
4. [同期付属機能プログラミング](#)

同期付属機能プログラミング

sample1をベースに同期付属機能の
イベントフラグのプログラミングを学ぶ

- プログラムディレクトリ ./jsp-1.4.1
/OBJ/AKIH8_3069F/sync_api を作成し、configure を使い
sample1 のプロジェクトを生成
- メインタスク(main_task)から並列実行されるタスク(task)
への指令をイベントフラグによる実現に変更
- 並列実行するタスクはfor()によるループはやめて
twai_flg() によりイベントフラグの更新を1秒間待つ
- ポイント
 - イベントフラグの生成
 - ビット割り当てと and/or待ち
 - 各タスクが自分のイベントフラグIDを抽出する方法

2005/12/19

TOPPERSプロジェクト認定

93



sample1をベースに同期付属機能プログラムを作成します。

変更内容は以下の通りです。

- 1) 現行のプログラムでは、メインタスク(main_task)から並行実行されるタスク(task)への指令はmessage配列によって受け渡されます。この受け渡しをイベントフラグを用いて行うように修正してください。
- 2) 並行実行されるタスク(task)は、イベントフラグ待ちに修正します。加えて、現状for文によるソフトデレイ待ちをtwai_flgサービスコールによる1秒タイムアウト待ちに修正します。

ポイントとしては以下の通りです。

- 1) コンフィギュレーションファイルを修正して、イベントフラグを生成します。
- 2) タスク間で交換するイベントフラグのビット割り当てきめ、OR待ちにするか、AND待ちにするかを考えましょう。
- 3) イベントフラグIDを決め、各々のタスクでこの名前をどのように共有するかを考えてみましょう。

同期付属機能プログラミング

■ イベントフラグの生成(sample1.cfg)

- イベントフラグ属性にTA_CLRを指定し、タスクをイベントフラグの待ち状態から待ち解除する時に、イベントフラグのビットパターンの全てのビットをクリアする

```
CRE_FLG(FLG_TASK1, {(TA_TFIFO|TA_CLR), 0});
CRE_FLG(FLG_TASK2, {(TA_TFIFO|TA_CLR), 0});
CRE_FLG(FLG_TASK3, {(TA_TFIFO|TA_CLR), 0});
```

■ ビット割り付け

```
#define FLG_PTN_e 0x01
#define FLG_PTN_s 0x02
#define FLG_PTN_S 0x04
#define FLG_PTN_d 0x10
```

```
#define FLG_PTN_y 0x20
#define FLG_PTN_Y 0x40
#define FLG_PTN_z 0x80
#define FLG_PTN_Z 0x100
#define FLG_PTN_ALL 0xff
```

2005/12/19

TOPPERSプロジェクト認定

94



sample1.cfgコンフィギュレーションファイルを修正して、3つの並列実行タスク(task)用のイベントフラグの生成をCRE_FLG静的APIを用いて追加します。CRE_FLGの属性、TA_TFIFOはC言語インターフェースの指定、TA_CLRは並列実行タスクがイベントフラグの待ち状態から解除される時にイベントフラグのビットパターンをゼロクリアする指定です。

イベントフラグは8つのビットを定義します。この設定は3つのフラグで共通です。

- 1) FLG_PTN_e 'e'コマンドが指定された
- 2) FLG_PTN_s 's'コマンドが指定された
- 3) FLG_PTN_S 'S'コマンドが指定された
- 4) FLG_PTN_d 'd'コマンドが指定された
- 5) FLG_PTN_y 'y'コマンドが指定された
- 6) FLG_PTN_Y 'Y'コマンドが指定された
- 7) FLG_PTN_z 'z'コマンドが指定された
- 8) FLG_PTN_Z 'Z'コマンドが指定された

twai_flgの待ち状態を指定するためにFLG_PTN_ALLを定義します。

同期付属機能プログラミング

- フラグIDの取り出しはフラグIDをセットした配列を静的な変数として用意する

```
const ID flgid[] = {FLG_TASK1, FLG_TASK2, FLG_TASK3};
```

- twai_flg()によりフラグを取り出す
 - ・ 待ちビットパターンは、全てのフラグをORしたFLG_PTN_ALLでの待ちとする
 - ・ 待ちモードはOR待ち(TWF_ORW)とし、いずれかのビットが設定されると起床する

```
FLGPTN flgptn;
:
flgptn = 0;
twai_flg(flgid[tskno-1], FLG_PTN_ALL, TWF_ORW, &flgptn,
        FLG_WAIT_TIME);
switch (flgptn) {
    case FLG_PTN_e:
```

2005/12/19

TOPPERSプロジェクト認定

95



並列実行タスクに対応したフラグIDに対応した静的な配列を定義します。この配列はメインタスクと並列実行タスクにて、対象のフラグIDを選択するために使用します。FLG_WAIT_TIMEは1秒を指定するために1000を指定します。

並列実行タスクでは、自動変数としてFLGPTN flgptnを定義します。while(1)文内にイベントフラグの待ち状態をtwai_flgサービスコールを追加します。待ちビットパターンは、全てのフラグをORしたFLG_PTN_ALLで待ちます。セットされたイベントビットはflgptn変数にセットされます。コマンドの解析はflgptn変数を用いて行うように修正します。同時にセットされた場合を考えて、switch文による解析より、if文によるビット解析の方が一般的です。このシステムでは、同時設定はありえないので、switch文でもかまいません。

同期付属機能プログラミング

■ メインタスク

- コンソールからデータを受け取りその種類に応じてイベントフラグをセットする
- セットする前にイベントフラグをclr_flgでクリアする

```
switch(c){  
case 'e':  
    syscall(clr_flg(flagid [tskno-1], 0x00));  
    syscall(set_flg(flagid[tskno-1], FLG_PTN_e));  
    break;  
case 's':  
    syscall(clr_flg(flagid [tskno-1], 0x00));  
    syscall(set_flg(flagid[tskno-1], FLG_PTN_s));  
    break;  
.....
```

メインタスクでは、コマンドをset_flgサービスコールに変えて、並列実行タスクに渡します。

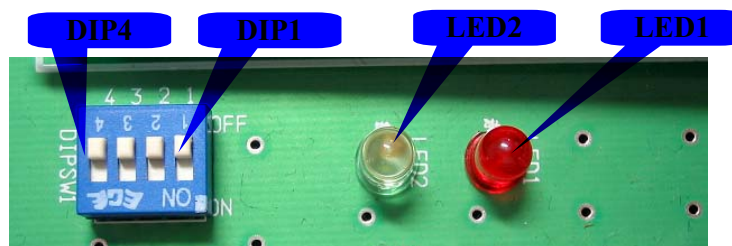
syscallマクロについては、./jsp-1.4.1/include/t_services.hを参照してください。サービスコールの返り値がマイナス(エラー)の場合、エラーの内容を表示する機能を持ちます。

カップラーメンタイマーの開発

- [1. カップラーメンタイマー仕様](#)
2. 設計方針
3. 設計例
4. コーディング例

カップラーメンタイマーの仕様

- 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30秒刻みで設定可能
- 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 : タイマが動いていることを表示
 - DIP1 : タイマを起動・停止
 - DIP4 : 設定時間を30秒延長



2005/12/19

TOPPERSプロジェクト認定

98



カップラーメンタイマーの作成

・簡単な仕様

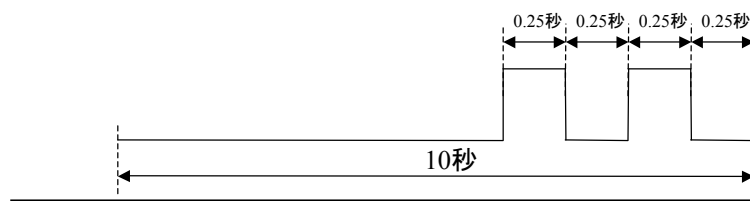
4つのDIP-SWのうち、DIP1とDIP4を使用します。DIP1をONで30秒タイマーを起動し、OFFでタイマーを終了します。タイマー起動中、DIP4をONにするたびに、タイムアップ時間を30秒延長します。電源投入後はLED1を2秒周期(1秒点灯、1秒消灯)します。タイマー起動中は10秒おきにLED2を0.5秒周期(250ms点灯、250ms消灯)で1秒間点滅します。また、タイムアップ時はLED2を0.5秒周期で15秒間点滅します。

・スイッチとLEDの機能

- 1) DIP1-ONでタイマー起動、OFFでタイマー終了
- 2) DIP4-タイマー起動中、ONを行うたびにタイムアウト時間を30秒延長する
- 3) LED1-タイマー起動中は2秒周期で点滅する
- 4) LED2-タイマー起動中、10秒単位に1秒間インターバル点滅する。タイムアウト時、15秒間インターバル点滅する。

カップラーメンタイマーの仕様

- 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SW4がONになると、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



2005/12/19

TOPPERSプロジェクト認定

99



イベントリストを作成します。

イベント	ステミュ ラス	アクション	レスポンス
電源投入	動作開 始	デバイスの初期化	タイマー動作準備完了、 LED1を2秒周期で点滅 する
タイマー起動	DIP1オ ン	タイムアウト時間30秒 でタイマー起動	LED2を10秒毎に1秒毎 にインターバル点滅、タイ ムアウト時15秒間インター バル点滅
タイマー時間 追加	DIP4オ ン	タイムアウト時間を30 秒延長	なし
タイマー終了 タイマー動作準備完了状態	DIP1オ フ	タイマー停止 ハードウェアとソフトウェアの初期化が終了 可能な状態	タイマー動作準備完了状 態へいつでも、タイマーの起 動が可能
タイマータイム アウト	動作停 止	最初のタイムアウト時間は30秒、タイマー時間追加毎に30秒時間を 追加する	

カップラーメンタイマーの開発

1. カップラーメンタイマー仕様
- [2. 設計方針](#)
3. 設計例
4. コーディング例

設計方針

可能な限りRTOSの機能を用いて設計する

- タスクは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
- 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
- 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
- 基本プログラムで作成したデバイス操作プログラムを流用

可能な限りRTOSの機能を用いてカップラーメンタイマーの設計を行う。

タスクは以下の2つを用意します。

- 1) タイマー用タスク(TIMER_TASK)、タイマーの管理を行う
- 2) 点滅制御用タスク(BLINK_TASK)、LED2のインターバル点滅制御を行う

4つの周期ハンドラを用意する

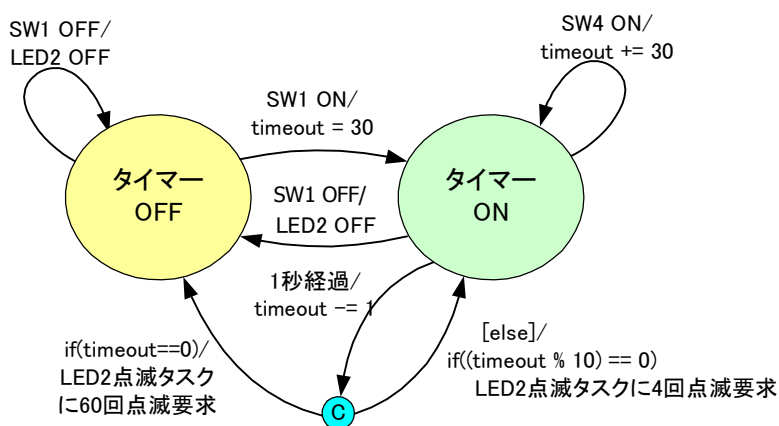
- 1) LED1点滅周期ハンドラ: LED1の点滅を制御する
- 2) LED2点滅周期ハンドラ: LED2の点滅を制御する
- 3) スイッチ用周期ハンドラ: DIP-SWの状態変化を検知する
- 4) タイマー用周期ハンドラ: タイマーの基本周期を発行する

カップラーメンタイマーの開発

1. カップラーメンタイマー仕様
2. 設計方針
- [3. 設計例](#)
4. コーディング例

設計例：タイマータスク

- 状態遷移図
- イベント
 - SW1 ON、SW1 OFF、SW4がON、1秒経過



2005/12/19

TOPPERSプロジェクト認定

103



タイマータスクは2つの状態を持ちます。

1) タイマーOFF状態 (タイマー動作準備完了状態)

タイマーOFF状態は、DIP-SW1の状態変化により状態変化が発生し、DIP-SW1がONの時、タイマーON状態に遷移します。

2) タイマーON状態

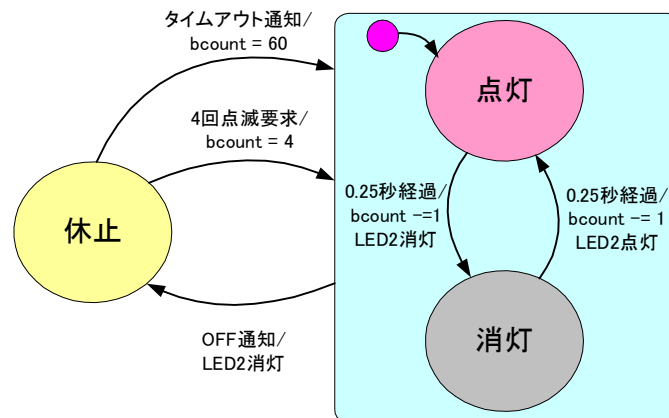
タイマーON状態は、以下の2つの変化によりタイマーOFF状態に遷移します。

- DIP-SW1がOFFに変化した時
- タイムアウトが発生した時

タイムアウト判定のために、タイマー周期ハンドラから10秒毎に通知をもらい、タイムアウト値を減算する。また、10秒毎、タイムアウト時に、インターバル点減の指示を行う。

設計例：LED2点滅タスク

- 状態遷移図
- 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



2005/12/19

TOPPERSプロジェクト認定

104



LED2は複雑なインターバル点滅を行う為、LED2点滅タスクとしてタスク管理します。LED2点滅タスクは休止状態と動作状態の2つの状態を持ちます動作状態ではLED2の点灯状態と消灯状態の2つの状態を持ちます。

休止状態から動作状態への遷移は、タイマータスクからのインターバル点滅要求によります。動作状態から休止状態への遷移は、インターバル時間の終了時に発生します。

点灯状態と消灯状態間の遷移は、LED2点滅周期ハンドラからの時間経過(250ms)通知により発生します。

設計例：周期ハンドラ

■ 周期ハンドラ

- LED1点滅周期ハンドラ : 1秒周期
- LED2点滅周期ハンドラ : 250ms周期
- スイッチ用周期ハンドラ : 10ms周期
- タイマー用周期ハンドラ : 1秒周期

■ 初期化ルーチン

- ハードウェアを初期化

2005/12/19

TOPPERSプロジェクト認定

105



4つの周期ハンドラを用意する

- 1) LED1点滅周期ハンドラ(LED1_BLINK_HANDLER) : 1秒周期
- 2) LED2点滅周期ハンドラ(LED2_BLINK_TIME_HANDLER) : 250ms周期
- 3) スイッチ用周期ハンドラ(SW_SCAN_HANDLER) : 10ms周期
- 4) タイマー用周期ハンドラ(BASE_TIME_HANDLER) : 1秒周期

ハードウェア初期化用の関数を用意する

初期化関数(init_cup_timer)は、LEDとDIP-SWの初期化を行う。

カップラーメンタイマーの開発

1. カップラーメンタイマー仕様
2. 設計方針
3. 設計例
4. [コーディング例](#)

コーディング例

- 実装されたコードは以下の場所にあります
 - プログラムディレクトリ ./jsp-1.4.1/OBJ/AKIH8_3069F/CUP_TIMER

- タイマータスクイベント割付

```
#define TIMER_SW1_ON      0x01  /* SW1 ON 通知 */
#define TIMER_SW1_OFF    0x02  /* SW1 OFF 通知 */
#define TIMER_SW4_ON      0x04  /* SW4 ON 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 */
```

- LED2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 */
```

2005/12/19

TOPPERSプロジェクト認定

107



2つのタスクへの状態遷移要求は、イベントフラグで設計します。

タイマータスクのイベントフラグ: FLG_TIMER

4つのフラグを使用します。

- 1) TIMER_SW1_ON: DIP-SW1がONに変化した(スイッチ用周期ハンドラから)
- 2) TIMER_SW1_OFF: DIP-SW1がOFFに変化した(スイッチ用周期ハンドラから)
- 3) TIMER_SW4_ON: DIP-SW4がONに変化した(スイッチ用周期ハンドラから)
- 4) TIMER_BASE_TIME: タイマーにて1秒の経過した(タイマー用周期ハンドラから)

LED2点滅タスクのイベントフラグ: FLG_BLINK

4つのフラグを使用します。

- 1) BLINK_ACTIVE: 1秒インターバル点滅を要求(タイマータスクから)
- 2) BLINK_TIMEOUT: 15秒インターバル点滅を要求(タイマータスクから)
- 3) BLINK_BLINK: 点灯、消灯の遷移を要求(LED2点滅周期ハンドラから)
- 4) BLINK_OFF: 点滅の終了を要求(タイマータスクから)

コーディング例 : タイマータスク

- タイマーOFF状態
 - SW1 ON、SW1 OFF イベント受け取る
 - SW1_OFF イベントを受け取ると、LED2タスクに BLINK_OFF を通知

```
void
timer_task(VP_INT exinf){
    int timer;
    FLGPTN flgptn;
    for (;;) {
        timer = 0;
        do {
            wai_flg(FLG_TIMER, TIMER_ALL, TWF_ORW, &flgptn);
            if ((flgptn & TIMER_SW1_ON) != 0) {
                timer = INIT_TIME;
            }
            if ((flgptn & TIMER_SW1_OFF) != 0) {
                set_flg(FLG_BLINK, BLINK_OFF);
            }
        } while (timer == 0);
    }
}
```

2005/12/19

TOPPERSプロジェクト認定

108



タイマータスク中のタイマーOFF状態は、自動変数のtimerの値がゼロの状態です。この状態は、TIMER_SW1_ONとTIMER_SW1_OFFの2つのイベントのみ、処理を行います。TIMER_SW1_ONを受信した場合は、timerの値をINIT_TIME (30秒) にセットすることにより、タイマーON状態へ遷移要求します。TIMER_SW1_OFFを受信した場合は、LED2周期タスクへ休止状態へ遷移を要求します。

コーディング例：タイマーON状態

- SW1 OFF イベントを受け取ると、LED2タスクに BLINK OFF を通知
- SW4 ON で時間延長
- 1秒経過イベントで時間をデクリメントし、0秒でLED2タスクにタイムアウトで60回点滅イベントを送り、10秒間隔で4回点滅イベントを送る

```

sta_cyc(BASE_TIME_HANDLER);
while (timer > 0) {
    wai_flg(FLG_TIMER, TIMER_ALL, TWF_ORW, &flgptn);
    if ((flgptn & TIMER_SW1_OFF) != 0) {
        set_flg(FLG_BLINK, BLINK_OFF);
        timer = 0;
    } else {
        if ((flgptn & TIMER_SW4_ON) != 0)
            timer = timer + EXTRA_UNIT;
        if ((flgptn & 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);

```

2005/12/19

TOPPERSプロジェクト認定

109



タイマーON状態では、タイマー用周期ハンドラをアクティブ状態に移行させ、1秒単位に **TIMER_BASE_TIME** を発行させます。このイベントにより、自動変数 **timer** の値を減算し、ゼロとなった時点でタイムアウトで、タイマーOFF状態に変化します。**timer** の値がゼロになった場合、15秒のインターバル点滅を **BLINK_TIMEOUT** フラグで、10秒毎に、1秒のインターバル点滅を **BLINK_ACTIVE** フラグで行います。

タイマーON状態で、**TIMER_SW1_OFF** イベントを受信すると、**timer** の値をゼロにして、タイマーOFF状態に移行します。また、**TIMER_SW4_ON** イベントを受信すると、**timer** の値に **EXTRA_UNIT** (30秒) を加算します。

コーディング例：初期化ルーチン

- ハードウェアの初期化
- スwitchの初期状態を保存

```
void  
init_cup_timer(void){  
    initial_switch(); /* スwitchの初期化 */  
    initial_led();    /* LEDの初期化 */  
  
    sw4 = get_switch(SW4); /* 現在のSW4の取り込み */  
    sw1 = get_switch(SW1); /* 現在のSW5の取り込み */  
}
```

DIP-SWとLEDとのインターフェースはdevice.hとdevice.cをそのまま使用します。ハードウェアの初期化関数は、スウィッチの初期化(initial_switch関数)とLEDの初期化(initial_led関数)の呼び出しを行います。また、現状のDIP-SWの状態を取り込むために、get_switch関数で取り込みを行います。Init_cup_timerの実行は、ATT_INI静的APIで行います。

コーディング例：周期ハンドラ

- タイマー用周期ハンドラ
 - 1秒周期で1秒経過イベントを通知

```
void
base_time_handler(void){
    syscall(isset_flg(FLG_TIMER, TIMER_BASE_TIME));
}
```

- LED2点滅周期ハンドラ
 - 250m周期で0.25秒経過イベントを通知

```
void
led2_blink_time_handler(void){
    syscall(isset_flg(FLG_BLINK, BLINK_BLINK));
}
```

タイマー用周期ハンドラは起動されるたびに、タイマータスクにTIMER_BASE_TIMEフラグをセットします。周期ハンドラは非タスクコンテキストで実行されるため、set_flgサービスコールではなく、isset_flgサービスコールを使用します。実行周期はCRE_CYC静的APIにて指定します。起動、終了要求はタイマータスクにて行います。

LED2点滅周期ハンドラは起動されるたびに、LED2点滅タスクにBLINK_BLINKフラグをセットします。実行周期はCRE_CYC静的APIにて指定します。起動、終了要求はLED2点滅タスクにて行います。

コーディング例：スイッチ用周期ハンドラ

－ スwitchの状態が変化した場合にイベントを通知

```
void
sw_scan_handler(void){
    CONDITION sw;
    FLGPTN  flgptn = 0;
    sw = get_switch(SW1);
    if (sw1 != sw) {
        if (sw == ON) {
            flgptn |= TIMER_SW1_ON;
        } else {
            flgptn |= TIMER_SW1_OFF;
        }
        sw1 = sw;
    }
}
```

```
sw = get_switch(SW4);
if (sw4 != sw) {
    if (sw == ON)
        flgptn |= TIMER_SW4_ON;
    sw4 = sw;
}
if (flgptn != 0x00)
    syscall(iset_flg(FLG_TIMER, flgptn));
}
```

2005/12/19

TOPPERSプロジェクト認定

112



スイッチ用周期ハンドラは、DIP－SWの状態を監視する周期ハンドラです。DIP－SW1とDIP－SW4の状態を監視し、変化があった場合、以下のフラグのセットを行います。

- TIMER_SW1_ON
- TIMER_SW1_OFF
- TIMER_SW4_ON

ドライバの初期化は、ATT_INIにて行われていますので、sw1とsw4の初期値はすでに設定されています。そのため、この周期ハンドラはシステムの起動時、起動状態になってもかまいません。CRE_CYC静的APIでアクティブ属性で実行してください。プログラムは、現状のDIP－SWの状態を読み込み、sw1とsw4に比較して、変化していればiset_flgサービスコールでフラグのセットを行います。

コーディング例 : LED1点滅周期ハンドラ

– 1秒周期でLED1を点滅

```
void  
led1_blink_handler(void){  
    static CONDITION sta_led = OFF;  
    if (sta_led == ON) {  
        sta_led = OFF;  
    } else {  
        sta_led = ON;  
    }  
    set_led(LED1, sta_led);  
}
```

LED1点滅周期ハンドラは、LED1を2秒周期で点滅する周期ハンドラです。LEDは初期化ルーチンですでに初期化済みで、タイマー起動可能状態では点滅を行っているため、この周期ハンドラもシステム起動時に実行可能な設定を行ってください。

コーディング例 : コンフィギュレーションファイル

- タスク2個、周期ハンドラ4個、イベントフラグ2個
- 初期化ルーチン

```

CRE_TSK(TIMER_TASK, { TA_HLNG|TA_ACT, (VP_INT) 0, timer_task,
    DEFAULT_PRIORITY, STACK_SIZE, NULL });
CRE_TSK(BLINK_TASK, { TA_HLNG|TA_ACT, (VP_INT) 0, blink_task,
    DEFAULT_PRIORITY, STACK_SIZE, NULL });
CRE_CYC(LED1_BLINK_HANDLER, {TA_HLNG|TA_STA, (VP_INT)0,
    led1_blink_handler, LED1_BLINK_INTERVAL, 0});
CRE_CYC(LED2_BLINK_TIME_HANDLER, {TA_HLNG, (VP_INT)0,
    led2_blink_time_handler, LED2_BLINK_INTERVAL, 0});
CRE_CYC(SW_SCAN_HANDLER, {TA_HLNG|TA_STA, (VP_INT)0,
    sw_scan_handler, SW_SCAN_INTERVAL, 0});
CRE_CYC(BASE_TIME_HANDLER, {TA_HLNG, (VP_INT)0,
    base_time_handler, BASE_TIME_INTERVAL, 0});
CRE_FLG(FLG_TIMER, {(TA_TFIFO|TA_CLR), 0});
CRE_FLG(FLG_BLINK, {(TA_TFIFO|TA_CLR), 0});
ATT_INI({TA_HLNG, 0, init_cup_timer});

```

2005/12/19

TOPPERSプロジェクト認定

114



2つのタスク、4つの周期ハンドラ、2つのイベントフラグの生成、初期化ルーチンの設定をコンフィギュレーションに記述を追加します。

CRE_TSKやCRE_CYCの属性、TA_HLNGはC言語のインターフェース記述を指定し、TA_ACTはシステムの起動時、タスクや周期ハンドラをアクティブ状態に移行させるように指定します。CRE_FLGの属性、TA_TFIFOはイベントフラグを待つタスクの待ち行列の実行順位をFIFO順に行うことをしています(このシステムでは実行タスクは各々ひとつしかない、加えて、TA_WMULにて複数タスク待ちを許可していないため、ここでは意味はない)。TA_CLRはタスクをイベントフラグ待ち状態から、実行状態に移行する時点でイベントフラグの全てのビットパターンをゼロクリアするように指定します。この設定があれば、wai_flgサービスコールの実行後、clr_flgサービスコールにてフラグのクリアを行う必要がなくなります。

ATT_INI静的APIで初期化指定を行えば、システムの起動前に初期化関数が実行されます。

まとめ

2005/12/19

TOPPERSプロジェクト認定

115



まとめ

以下の事項について学習

- AKI-H8/3069FとTOPPERS/JSPカーネルの構築方法
- 周辺デバイスのプログラミング方法
- ITRON仕様
- ITRON APIを用いたプログラミング
 - タスク関連のAPI
 - 同期付属機能
- カップラーメンタイマー
 - 設計
 - コーディング

2日目はTINETを用いたTCP/IPプログラミングです

付録：開発環境の構築方法(1/2)

- PizzaFactory2を使用する場合
 - 付属CD-ROMからインストール
- Cygwinを使用する場合
 - H8用のGCCをソースから構築もしくはバイナリをダウンロードしてインストール. 3.3.2で動作確認
- Tera Term Proのインストール
 - Webサイトからバイナリをダウンロードしてインストール
<http://hp.vector.co.jp/authors/VA002416/>

付録：開発環境の構築方法(2/2)

- ツール及びソースコードのインストール
 - ホームディレクトリ以下にnecess_moddle_2005.lzhを展開すると作成されるenvironmentを置く
 - PizzaFactoryのホームディレクトリは
C:\Program Files\MonamiSoftware\PizzaFactory2Edu\home\xxx
最後のxxxはログイン名, home以下はPizzaFactory2のCommandLineを実行すると作成される
 - enviroment\DeviceManager\RegDeviceManager.batを実行する. 二回ダイアログが表示されるのでOKを押す
 - 各プログラムのショートカットをデスクトップ上に置く

教材の作成

参考資料

- 「H8/300Hシリーズ プログラミングマニュアル」
ルネサステクノロジ

- 「H8/3069 F-ZTAT ハードウェアマニュアル」
ルネサステクノロジ