

TOPPERS初級実装セミナー

(JSP-1.4|OAKS16 mini|カップラーメンタイマー版)

1日目

TOPPERS プロジェクト

教育ワーキング・グループ

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 初級実装セミナー (JSP1.4/OAKS16-MINI/タイマー版)1日目>

Copyright (C) 2004 by 竹内良輔 (株)リコー プラットフォーム開発センター
Copyright (C) 2004 by 山本雅基 (株)デンソークリエイト

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) することをお断りします。

- (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。

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。

本セミナーの目標

- TOPPERS/JSP上へのアプリケーション・プログラム実装を通して、RTOSに対する実装技術を習得する。
- ハードウェアの初歩的なインターフェイスについての実習による理解を深める。
- SESSAME組込みソフトウェア技術者教育セミナーの復習による分析、設計、実装プロセスの理解を深める。

アプリケーションの実装に使用するボードはCPUとしてルネサステクノロジ製のM16Cを使用しています。この開発環境はWindows上動作する統合開発環境でコンパイル、リンクが行えます。まず、この環境での開発方法を習得します。生成された実行ファイルをターゲット・ボード上のフラッシュROMに書き込むことにより、きちんとした動作することを確認します。TOPPERS/JSPへのアプリケーションのポーティングは、サンプル・プログラムを参考にして統合開発環境上でのTOPPERS/JSPシステムの構築方法を習得します。その後、サンプル・プログラムを拡張していき、目的のシステムを作ってください。拡張に使用するRTOSの機能は、タスクの生成と起動、共有メモリを使ったタスク間の通信、イベント・フラグを用いたタスク間通信です。

ハードウェアのインターフェイスはdevice.cというソース・ファイルに収められたデバイスインターフェイスによって制御しますので、直接内容を理解する必要はありませんが、講座の中でタスク・モニターを用いてハードウェア・ポートを直接アクセスする内容が含まれています。このデバイスの記述はDIC仕様に基づいて記述されています。

SESSAMEの組込みソフトウェア初級セミナーでは、構造化設計を用いて、ターゲットシステムを、要求分析、設計、実装、テストという開発プロセスで開発することを勉強しました。教材を用いてこの開発プロセスを復習します。

スケジュール

■ 1日目

1. 開発環境の確認 0.5時間
2. 組込みとは、SESSAMEの復習 1.5時間
3. SESSAME版教材の動作確認 1時間
4. リアルタイムOSとは何か 1.5時間
RTOSの状態遷移、HW I/F
教材を用いた確認
5. カップラーメンタイマーの設計 1.5時間
設計を行う

■ 2日目

- カップラーメンタイマーの実装
- 実装を行う 1.5時間
- レビュー 0.5時間
- 設計、実装の評価
- 同期と通信 1時間
- タスク間通信について
- 同期、排他制御について
- イベントフラグを用いた改造 1.5時間
- まとめ、宿題 1時間

開発環境の確認

1. 開発環境の確認
2. 組込みとは、SESSAMEの復習
3. SESSAME版教材の動作確認
4. リアルタイムOSとは何か
5. カップラーメンタイマーの設計

まず、OAKS16-MINIの開発環境について習得しましょう。ここでは提供されている機材の確認、開発環境の4つのツールの確認、パソコン上のTOPPERS開発環境の理解、エディタを用いてソースの参照ができるかの確認、統合開発環境でのビルドができるかの確認を行きましょう

実習環境を確認してください

用意されている実習環境の確認

- パソコン
- ボード(OAKS16 MINI)
- RS232Cケーブル
- テキスト
- 箱

電源	1個
CD-ROM	1枚
フロッピー	1枚
説明書	1枚



RS232Cケーブル



Windows-PC



OAKS16 mini ボード



テキスト



箱



- ・パソコン
 - 以下の実装が必要
 - MITSUBISHI TOOL
 - エデッタ
 - 通信ソフト
- ・OAKS16 MINIボード
 - 組み立て済みのもの
- ・RS232Cケーブル
 - パソコンとボード間の通信を行います。
 - モニターの表示やボード上のフラッシュROMの書き込みに用います
- ・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
 - ボードの説明や開発環境が入ってます
- ・フロッピー
 - 教材の環境やソースや設定方法が入ったフロッピー
 - CD-ROMと合わせて、お家でも開発環境を構築できます
- ・説明書
 - OAKS16-MINIの説明書

開発環境を確認してください

Windowsの開発環境の確認

- スタート→すべてのプログラム→RENESAS TOOL
- Flash staterは直接実行です

TM V3.20	統合化開発環境、ここをベースにコンパイル、リンクを行う	使用する
NC30WA V4.00,V5.00	M16C用のC言語コンパイラ、アセンブラ、リンカー	使用する
KD30 V4.10	リモート・デバッガ、Monitor Programと通信して動作	確認のみ 使用
Flash stater	内蔵フラッシュメモリ書き込みツール	使用する

開発環境は4つのツールから構成されています。

TMは統合開発環境です。プロジェクト単位に管理を行います。Windows インターフェイスでコンパイル、リンクが可能です。

NC30WA、M16C用のC言語コンパイラ、アセンブラ、リンカーです。統合開発環境が無い場合は、DOS窓からのコマンドにより実行できます。

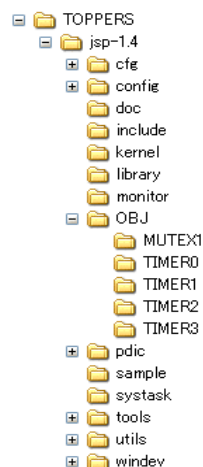
KD30はリモートデバッガです。この講習では使用しません。実行プログラムはフラッシュROMに書き込みますので、ボード上のモニターは上書きされ消滅します。CD-ROM上にモニターの実行形式ファイルがありますので、それを上書きすることにより、再び、リモート・デバッガを使用することができます。

Flash staterは、ボード上のフラッシュROMにプログラムを書きこむプログラムです。

開発ディレクトリの確認

開発プログラムや設定が入ったディレクトリ

- Windowsのエクスプローラから開発用のディレクトリを確認してください
- TMを起動して、OBJ\TIMER0ディレクトリ中のプロジェクトをオープンしてください
- Buildできるかの確認を行ってください
- エディタ - の確認を行ってください



パソコンの中の開発ディレクトリを確認してください。

スタート→すべてのプログラム→MITSUBISHI-TOOL→TM V.3.20→TMで

TMを起動してください。ファイルを開くからTOPPERS\OBJ\TIMER0ディレクトリ中のNonOsTimer0.tmkを開いてください。Project Editorを開くを押して、Project Editorを開いてください。Project Editor中のall→timer0.x30→timer0.r30を開きtimer0.cをダブルクリックしてtimer0.cを表示させてください。これが今回作成するカップラーメンタイマーのノンOS版のソースです。

リビルドを選択して、コンパイルとリンクを行ない、Builder画面で正常終了することを確認してください。timer0ディレクトリ中に、NonOsTimer0.motファイルが作成されたことを確認してください。このファイルが実行形式ファイルです。

ディレクトリについて

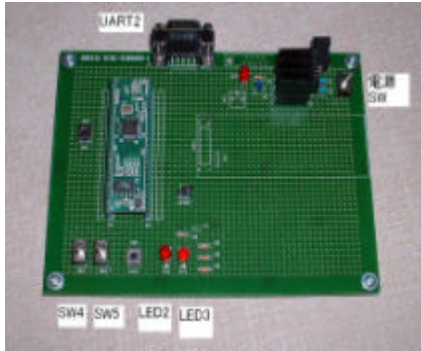
cfgはコンフィギュファイルをソースに変換したり、チェックしたりするプログラムが入っています。configは、TOPPERSのCPUやボードの対応部分のソースが格納されています。docはドキュメント類、includeはITRON4.0の宣言、kernelはリアルタイムカーネルのソース、libraryはカーネル用のライブラリのソース、monitorはタスクモニターのソース、pdicはデバイスドライバのソース、sampleはサンプルプログラム、systaskはタイマーやシリアル制御部やシステムログタスクのソースが収められています。toolsには使用する開発環境のようなプログラム、utilsは構築用のユーティリティ、windevにはWindows環境でのデバイスドライバのソースが収められています。OBJディレクトリの下に教材用のプログラムの構築環境があります。

OAKS16 mini

概要図と外部仕様

- RAM2KB、ROM64KB
2つのスイッチ、2つのLEDを制御できます

概要図



外部仕様

項目	内容
MCU	MCU:M30262F8GP 動作モード：シングルチップモード クロック周波数：メインクロック 20MHz
メモリ	内部メモリ RAM :2KB Flash ROM :64KB 外部メモリ なし

OAKS16 miniボードを確認してください。ボードのD-SUB9ピンとパソコンのD-SUB9ピンをRS232Cケーブルで接続してください。電源を電源コネクタの接続してください。

ターミナルの設定は以下の設定としてください。

19,200 baud

8bit

1stop bit

None Flow Control

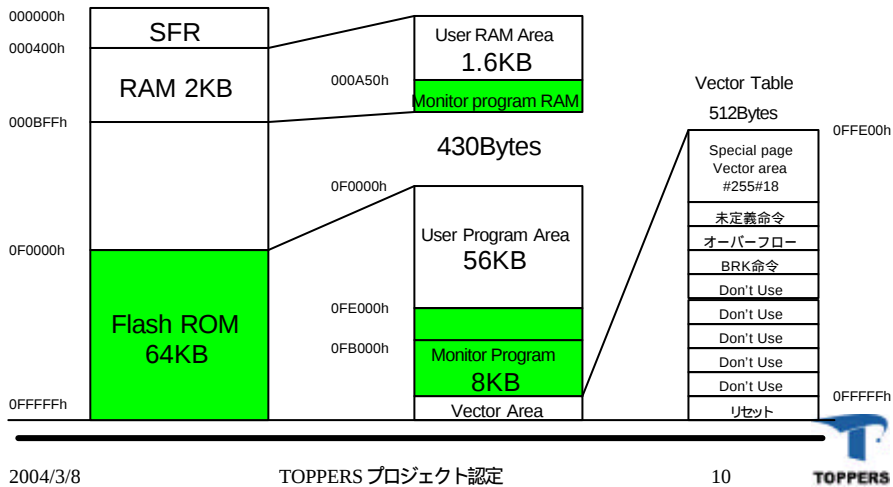
ボードが正常に動作することを確認するために、KD30デバッカの起動を行います。電源スイッチをオンにしてください。LED1が点灯することを確認してください。パソコンでKD30を起動します。スタート→すべてのプログラム→MITSUBISHI-TOOL→KD30 V3.10 Release1→KD30。Referを押して、M30262F8_DBC.msuを選択してください。PortとBaud Rateを設定し、OKを押して、KD30のデバックウインドウが表示することを確認してください。

今回使用するデバイスは、SW4,SW5のキーとLED2、LED3のLEDです。

OAKS16 mini

メモリマップ

■ 本講習では、Monitor Programは使用しません



OAKS16 miniのメモリマップです。RAMは2KB、ROMは64KBです。KD30の実行のためにモニタープログラムが実装されています。このモニタープログラムはROMを8KB、RAMを430バイト使用します。M16C版TOPPERS/JSPでは、モニタープログラムを使用しません。そのため、64KBのROMと2KBのRAMをすべて使用する形となります。デバッカがない代わりに、M16C版TOPPERS/JSPでは、タスク・モニターを実装してあります。このモニターはタスクのひとつとして動作し、他のタスクの状態管理、状態監視やメモリの表示、設定等が行えます。

SFRはM16C-CPUコアの中に内蔵されたハードウェア部とのインターフェイスを行うポート領域です。

OAKS16 mini回路図

回路図はOAKS16MINIのCD-ROMを参照してください

OAKS16 miniの回路図です。

SW4、SW5はCPU側ではP81、P80ピンの接続されています。このピンはポート8の1ビットと0ビットに対応しています。ポート8は入力ポートとして使用します。LED2、LED3はCPU側のP75、P74ピンに接続されています。このピンはポート7の5ビットと4ビットに対応しています。ポート7は出力ポートとして使用します。

OAKS16を使用の場合は、SW2,SW3をP81、P80ピンに、LED2、LED3をP75、P74に接続するような改造が必要です。

組込みとは、 SESSAMEの復習

- 1 .開発環境の確認
- 2 .組込みとは、SESSAMEの復習
- 3 .SESSAME版教材の動作確認
- 4 .リアルタイムOSとは何か
- 5 .カップラーメンタイマーの設計

ここでは、組み込みシステム全般について勉強を行います。また、この講習で開発する「カップ・ラーメンタイマー」の仕様について確認を行います。

「カップラーメン・タイマー」をSESSAME初級で学んだ、構造化設計を用いて、要求分析、設計を行っていきましょう。NonOsTimer0を実際にボードに書き込んで、いろいろなテストを行っていきましょう。

組込みシステムとは？

対象システムの確認

- 各種の機器に組み込まれて、この制御を行うコンピュータ機器のこと

組み込まれて」のところが曖昧性があり、実際は人により組み込みシステムとする範囲が異なる

境界例) プラント制御、PDA、ゲーム機

- ここでは「組込みシステム」を広い意味に捉える
極端に言えば、パソコン、メインフレームなどの汎用コンピュータ以外は組込みシステム

ある目的に専用化されたシステムと言い換えることもできる

ここで捉える組込みシステムは汎用コンピュータ以外は組込みシステムとします。
狭い意味での組み込みシステム (外見がコンピュータでないもの) を deeply embedded system と呼ぶ場合もあります。

組込みシステムの例としては、以下のものがあります。

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

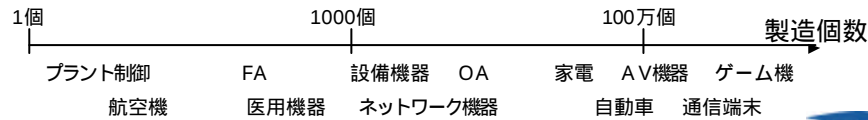
組込みシステムの多様性

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



2004/3/8

TOPPERS プロジェクト認定

14



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る。

・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い。

・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る。

・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つ

組込みシステム開発の最近の動向

組込み開発における種々の変化

■ 組込みシステムの適応範囲が拡大

Digital consumer(携帯電話、デジタル機器、ITS、・・・)

■ 従来からの組込みシステムの大規模化・複雑化

機器の複合化、デジタル化、ネットワーク化

ユーザインターフェイスの高度化

コンピュータ制御による高機能化・高付加価値化

■ 開発期間(time-to-market)の短縮コストダウン

システムの品質・信頼性の確保が大きな問題に

■ ソフトウェアとハードウェアの境界の流動化

2004/3/8

TOPPERS プロジェクト認定

15



このような動向の中で、設計資産の再利用が注目されています。

設計資産の再利用

ソフトウェア設計とハードウェア設計に共通の課題として、設計資産の有効な再利用が不可欠となってきています。この方向性として2つの傾向があります。

ひとつは自社で過去に開発した設計資産の再利用があります。開発自体をうまくやっていかないと、どんどん複雑な構造(レガシーコード化)になってしまいます。この対応として、設計資産のナレッジ化(モデル化)を行ったりリファクタリング(再利用の為に再構成)を行っていく必要があります。

もうひとつは他社からソフトウェア部品/IPの形で設計資産を購入する場合があります。これは、すべてのソフトウェアを自社内で開発しては開発期間短縮の要求に応えられませんし、また、従来とは異なる種々の技術がシステムとして要求されるケースが増えつつありますし、自社のみの開発では大きな投資が必要となるからです。この対応として、組込みシステム開発のオープン化が求められています。

設計資産の再利用の阻害要因

阻害要因としては、過剰なまでの多様性が挙げられます。そのジャンルはハードウェア、プロセッサ、OS、ミドルウェア、ネットワーク方式等です。これらには、囲い込みによる淘汰が必要となります。また、テストに掛かるコストが大きいことも阻害要因のひとつです。

組込みシステムの特徴

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

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る。

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る。

・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。

まず、SESSAME初級セミナーの復習

教材を用いた復習

■ SESSAME組込みソフトウェア技術者セミナー

組込みソフトウェア開発では「話題沸騰ポット」や「鹿脅し」の事例にて、組込み向けの構造化設計手法について、勉強しました。

要求分析→設計→実装→テスト

このセミナーでは、まず、教材を用いて、組込み構造化設計を解説してみましよう。

SESSAME組込みソフトウェア技術者セミナーについて、

組込みソフトウェア管理者・技術者育成研究会 (SESSAME:Society of Embedded Software Skill Acquisition for Managers and Engineers)は組込みソフトウェア技術者や管理者を育成する為のカリキュラムの整備、そしてその基となる方法論・ツールの開発に関する研究を行っている団体です。

初級技術者セミナーでは、分析・設計・プログラミング・テストに絞り、まず、実装を考える前に、組込みシステムの分析、設計を行うという観点から始め、分析、設計、テストの為の方法論の紹介を行っています。

OAKS16 MINIで何を作るか？

カップラーメンの食べごろを知らせるタイマー

- OAKS16 MINIを使用して “カップラーメン・タイマー” 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に、10秒毎にLED2を点滅、タイムアウト時 30秒間点滅して停止。



開発を始める前に

2004/3/8

TOPPERS プロジェクト認定

18



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。OAKS16 miniボードでは、LEDとスイッチしかデバイスが無い為、LEDを使って時間と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30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分づつタイムアウトを延長させます。

まずはSESSAME初級セミナーの復習

■ 組込み開発は要求分析から

イベントリスト

データ・ディクショナリ

コンテキスト・ダイアグラム

フローダイアグラム

プロセス仕様書

要求分析

設計

実装

テスト

ストラクチャチャート

モジュール仕様書

組込みシステムは、前述の通り 高い信頼性、リアルタイム性を要求されるシステムといえます。それに反して、近年のシステムの大規模化に伴い、品質の低下、保守性の確保が問題となっています。これらの問題の解決と共に、システムの可視化を目指して、要求分析からシステム構築を始めましょう。

要求分析は、開発対象システムの本質理解、明確化する作業です。どうやって仕様を満たして、作り上げていくかではなく、まず、求められているシステムはどのようなものを、考察、整理することから分析を行いましょう（Whatの視点）

SESSAMEの要求分析は、構造化分析からはじめ、設計に移る過程で、構造化設計に移行します。構造化設計は分析の結果を、どのように作るかに視点をおきます。（Whatの視点からHowの視点への移行）。

構造化分析は以下の開発手順で行われます。

1. イベントリスト
2. コンスタント・ダイアグラム
3. フローダイアグラム
4. プロセス仕様書

必要に応じて、データ・ディクショナリを作成します。

要求分析 1

イベントリスト、データ・ディクショナリ

■ イベントリスト

イベント：システム外で発生しシステムに影響を及ぼす事象

ステミュラス：イベントの発生を伝える手段

アクション：イベントの発生時にシステムで行う機能

レスポンス：システム外への応答

イベント	ステミュラス	アクション	レスポンス
電源オン	動作開始	現在時間を知らせる	LED3を1秒ごとに点滅
タイマー開始	SW5 オン	タイマーを起動 ユーザーへ起動の通知 タイムアウト時ユーザーに通知	LED2を10秒毎に点滅させる タイムアウト時 LED2を30秒点滅
タイマー時間追加	SW4 オン	現在のタイムアウト時間を分延長する	なし
タイマー停止	SW5 オフ	タイマーを停止する	リセット状態へ
電源オフ	動作停止		

■ データ・ディクショナリ

初期状態	LED3を1秒毎に点滅させることにより、システムの稼働状態をユーザーに伝える。
タイムアウト時間	初期タイムアウト時間、分、タイマー時間追加毎に1分ずつ延長する。
タイムアウト表示	タイマー実行中は10秒毎にLED2の点滅、タイムアウト時30秒間LED2を点滅する。

2004/3/8

TOPPERS プロジェクト認定

20



イベントリストは外部から発生する入力に対応して、システムがどのように対処するかを明確化したリストです。このリストは4つの項目に分けて作ります。イベントリストを作ることで、カップラーメン・タイマーの動作がより明確になります。

1. イベント システム外で発生し、システムに影響を及ぼす事象
システムが発生を制限できない事象ということもできます。
2. ステミュラス イベントが発生したことを伝える情報入力
3. アクション イベントが発生したときに、システムが果たすべき機能
4. レスポンス イベントが発生したときの外部への応答

カップラーメン・タイマーでは4つのイベントに対して、イベントリストを作成しました。

データ・ディクショナリは分析や設計で用いる用語を解説を加えて登録したものです。用語は、誰にでも分かる言葉を用いて、開発作業で統一して用いることが重要です。

注意：ここでは、実装の講義で語句が統一されるように、ステミュラスに、実装依存の名称を使用していますが、実際の要求分析では以下の例のように分析用の名称を使います。

SW5	→	タイマー起動スイッチ
SW4	→	タイマー延長スイッチ

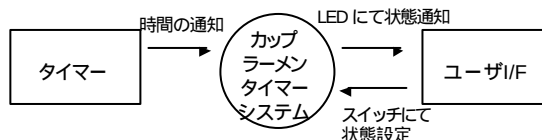
要求分析 2

コンテキスト・ダイアグラム、フローダイアグラム

■ コンテキスト・ダイアグラム

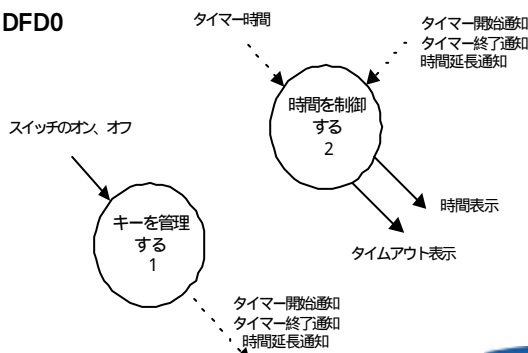
システムと外部のターミネータの関係を示す図

フローダイアグラムの最上位層



DFD0

■ フローダイアグラム



コンテキスト・ダイアグラムはUMLでも用いられる表記法です。システムと外部のターミネータとの関連を表す図です。カップラーメンタイマーシステムはターミネーターとして、ユーザI/F（ユーザ）とタイマーを上げていますが、タイマーはシステムに含まれると考えるならばターミネータとする必要はありません。

フローダイアグラムの記述として、データフロー・ダイアグラム(DFD)を用います。データフロー・ダイアグラムは丸の部分（プロセス、変換）とそれに向かう矢印（入力データ）とそれから出る矢印（出力データ）で構成されます。丸の部分は入力データに対して何らかの変換を行うものと考えられます。カップラーメン・タイマーシステムでは2つのプロセスに分解されます。

1. プロセス1: キーを管理するプロセスです。スイッチのオンオフより、タイマー開始通知、タイマー終了通知、時間延長通知の3つの出力データに変換させます。

2. プロセス2: 時間を制御するプロセスです。タイマー時間と3つのキーの要求通知より、時間表示とタイムアウト表示の2つの出力データに変換を行います。

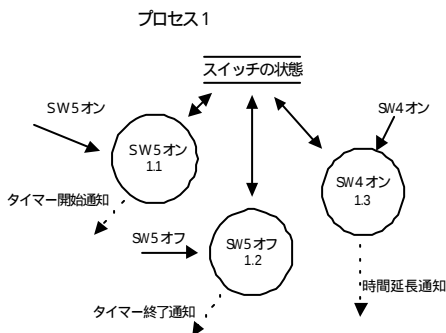
これらのデータフロー・ダイアグラムはシステムから抽出された最初のフローダイアグラムなのでDFD0と呼ばれます。

要求分析 3

フローダイアグラムの詳細化

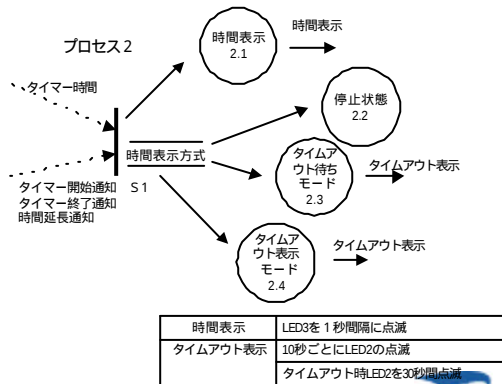
■ プロセス 1 の詳細化

キーの管理機能をより詳細に
ダイアグラム化する



■ プロセス 2 の詳細化

時間の表示機能をより詳細
にダイアグラム化する



DFD0の2つのプロセスをより詳細なデータフローダイアグラムに分析しましょう

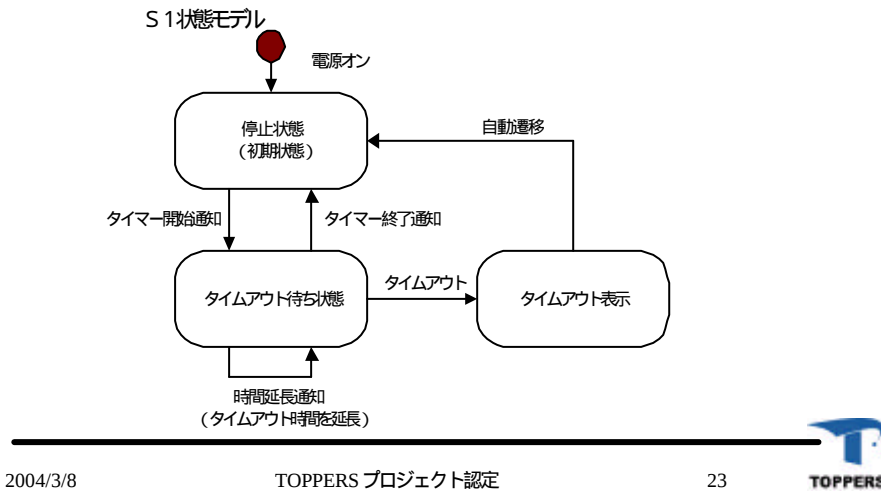
スイッチの状態や時間表示方法は状態を持ちます。状態については状態モデルを用いて詳細化しましょう。プロセス1は詳細化によって、SW5 オン1.1、SW5 オフ1.2、SW4 オン1.3の3つのプロセスに分解されました。プロセス2は詳細化により、時間表示2.1、停止状態2.2、タイムアウト待ちモード2.3、タイムアウト表示モード2.4の4つのプロセスに分解されました。

タイムアウト表示には、10秒後との経過表示とタイムアウト時の表示の2つに分けられます。

要求分析 4

時間表示方式を状態モデル化 (S1)

■ S1は3つの状態をもつ



時間表示方式を状態モデル化してみました。時間表示 2.1 は時間表示方式には関係なく常に表示されますので、状態モデルには表されません。タイマーの機能としては、停止状態、タイムアウト待ち状態、タイムアウト表示の 3 つの状態に遷移します。

要求分析 5 プロセス仕様書 (2.2-2.4)

MTを最大時間、CTを現在時間とします。

■ 2.2初期状態

1. タイムアウト時間を初期化する

MT = 0, CT = 0

制御周期ごとに

1. タイムアウト待ち表示チェック

CTが10秒の倍数ならば

2秒間のタイムアウト表示を要求する

■ 2.3タイムアウト待ちモード

初回

1. タイムアウト時間の設定

MT = デフォルトタイムアウト時間

2. 現在時間を更新する

CT += Δ

時間延長通知ごとに

1. タイムアウト時間の延長

MT += 1分

■ 2.4タイムアウト表示モード

1. 30秒間のタイムアウト表示を要求する

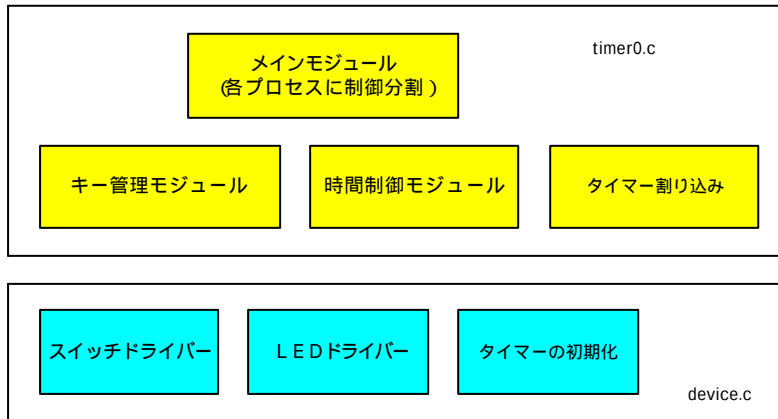
分析したプロセスに関して、プロセス仕様書を作成しましょう。プロセス仕様書は入力から出力へどのような変換が行われるかを記述するものです。記述方法は、適切に記述できれば何でもかまいません。この例については文章で記述しましたが、前述のように状態モデルでも、状態遷移図でもかまいません。

機能については、DFDのプロセスを用いて自然に抽出されます。次に制御モジュールの抽出を行います。上記の例では、経過時間監視を時間の監視を行うモジュールとして作成しました。最後に外部の入出力モジュールとして追加します。ここでは、タイマー、スイッチ、LEDを入出力モジュールとして追加しました。

設計 2

構造図をモジュール単位に分類

■ プロセス主体の部分とデバイス部に分類



次に、構造図をモジュール単位に分割します。ここでは、DFDのプロセスに関するモジュールと入出力モジュールに関する部分の2つのモジュールに分割しました。

モジュール分割に関しては種々の吟味が必要です。この内容については、レビューの時に、詳細を記述します。

設計 3

モジュール仕様設計

名称	目的	引数	戻り値	ソース
メイン	キー管理と、表示制御に一定時間で制御を与える	なし	なし	timer0.c
キー管理	キーの状態を管理し時間制御に伝える	なし	なし	
時間制御	時間ごとのLED表示を行う	なし	なし	
タイマー初期化	タイマーHWの初期化	なし	なし	device.c
スイッチ初期化	スイッチHWの初期化	なし	なし	
LED初期化	LEDHWの初期化	なし	なし	
スイッチ読み込み	スイッチの状態を読み込む	番号	状態	
LED設定	LEDに点灯、消灯を設定	番号 状態	なし	
タイマー割り込み	一定時間での割り込みにより、現在時間を管理する	なし	なし	timer0.c

モジュール分割に基づいて、モジュール仕様書を作成します。モジュール仕様書はインターフェイス仕様書と機能仕様書に分けられます。このスライドの仕様書はインターフェイス仕様書です。カップラーメン・タイマー自体が大きなシステムではないので、目的からスペックを簡単に類推できることもあり、ここでは機能仕様書は割愛します。2つの仕様書に関して簡単に説明を行います。

1. インターフェイス仕様書

そのモジュールは何をすべきかの観点で記述します。特に入出力を明確にすることが必要です。スライドの例のように、名称、目的、引数、戻り値を項目別に記述しましょう

2. 機能仕様書

そのモジュールの機能をどのように実現するか観点で記述します。

例えば、キー管理の場合、

- (1) SW 5 オンならばタイマー開始を通知
- (2) SW 5 オフならばタイマー停止を通知
- (3) SW 4 オンならば時間延長を通知

のように記述する。

設計 4

非機能要件を明確化

- 時間の取り込み
タイマー機能、割り込みにて対応
- デバイスインターフェイス
スイッチは最小2つポーリングにてセンス
LEDは最小2つポート設定により点灯、消灯
- 製品対応要求
ROM 64KB以内、RAM 2KB以内
- エラー処理メカニズム
特に対応しない

機能として表されない部分を明確化する必要があります。カップラーメンタイマーでは、ハードウェアに依存した部分も機能要件にいった設計を行っていますが。本来実装に依存する部分を分離して分析、設計を行うと、より実装に依存しない再利用性の高い分析、設計が可能となります。設計を実装化しやすいように非機能要件を明確化する必要があります。

非機能要件としては以下のものが上げられます。

1. 時間的制約

応答時間

システムの分類としては、ハードリアルタイム、ソフトリアルタイム

2. リソースの制約

ROM / RAMサイズ

CPUのパワー

3. システムの制約

開発環境、工場検査、市場でのメンテナンス等

4. システムの安全度

フォールトトレランス、過負荷、フェイルセーフ

実装 1

既に、実装されたものを参照してください

- 先ほど、実装例がTIMER0が収められている
- 「SESSAME版教材の動作確認」にて詳細な説明します

start.a30	電源投入時のハードウェアの初期化、割り込みベクトルの設定
device.c	ボードに依存しないスイッチやLEDの制御インターフェイス
timer0.c	メインルーチン、2つのプロセス、割り込みを実装したソースファイル

1 .timer0.c

```
void main(void);
```

ハードウェアの起動後実行される関数、各デバイスの初期化後、250ms毎に、プロセス1 (switch_process)とプロセス2 (timer_process)を呼び出す。

```
void swith_process(void);
```

SW4、SW5が変化した場合、共有メモリ領域に通知情報を書き込む

```
void timer_process(void);
```

共有メモリが設定されている場合、タイマーの状態遷移を行う 状態遷移とは関係なく、1秒単位ごとにLED3を点滅する。

```
void timer_handler_entry(void);
```

タイマー割り込みプログラム。1MS単位に割り込みが発生しbase_timeをインクリメントする。

2 .device.c

```
void initial_timer(void);
```

```
void initial_key(void);
```

```
void initial_led(void);
```

各デバイスの初期化を行う

```
Int get_key(int sw);
```

swに対応したキーの状態を取り込み、ONでオン、OFFでオフ。

```
void set_led(int led, int req);
```

ledをreq状態にする。reqはONは点灯、OFFは消灯。

テスト1

テストとは

■ テストの目的は？

可能な限り品質保証できる範囲を広げる
効果的に不具合を発見する

■ テストの仕方は？

・網羅型

全体をくまなく一通り

制御パステスト、機能網羅テスト、状態網羅テストなど

・ピンポイント型

不具合の多そうなところを狙う

境界値テスト、ストレステスト

過去の経験から

不具合の分析を行って、弱点を把握する

不具合の影響はとんでもなく大きい

- ・300万行のうち1行余計だったせいで、AT&Tは11億ドルの損害

エラー回復コードがバイパスしたせいで、

長距離電話が9時間に渡り話中になった

- ・ブレーキシステムのバグで、GMにリコール

停車距離が15～20m伸びてしまった

350万台がリコール、数百万ドルの損害

- ・ARIANE 5ロケットが爆発

オーバーフローが原因

4億ドルの損害

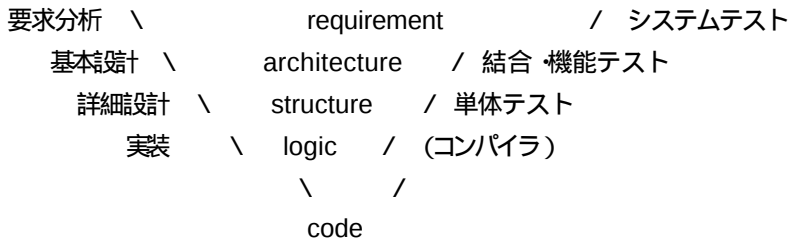
テスト2

テストに対する意識

- テストの仕方の考慮 (網羅型、ピンポイント型)
- テストとは
 - ・知恵を絞って設計すべし
 - ・不具合を見つけて初めてテストは成功
 - ・いろいろなテスト手法があり、目的に応じたテストが必要
- テストが上手になると、不具合の少ない開発が可能になる
- テスト意識して開発することで始めから不具合の少ないソフトウェアを作ろう

テストのやり方

組み込みテストのフェーズ :V字モデル



SESSAME版教材の 動作確認

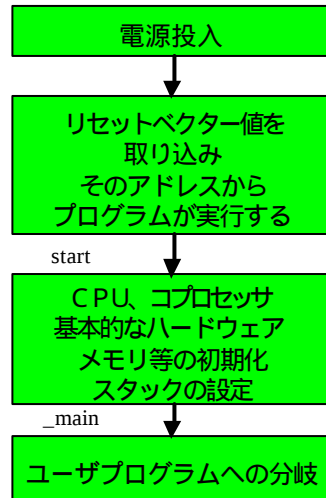
- 1 .開発環境の確認
- 2 .組込みとは、SESSAMEの復習
- 3 .SESSAME版教材の動作確認
- 4 .リアルタイムOSとは何か
- 5 .カップラーメンタイマーの設計

教材を用いて、仕様や実装の確認、ビルド、簡単なテストを行ってみよう

実装知識 1

スタートアップ

- 汎用OSでは、OSがmain()に実行権を与えてくれる
- OSがない場合は、自分でmain()に実行権を与えるプログラムを作らなければならない
- この処理をスタートアップという



```

/*
 * カップラーメンタイマーのメイン関数
 * ハードウェアとデータの初期化後
 * switch_process とtimer_process を250ms 毎に
 * 起動する .
 */
void
main(void)
{
    unsigned long timer250;

    initial_key();
    initial_led();
    initial_timer();
    base_time = 0;
    event = 0;
    timer250 = 250;

    _asm( "%iFSET I");
    sw4 = get_key(SW4);
    sw5 = get_key(SW5);
    timer_state = STATE_STOP_TIMER;
    sec = 0;
    alaem = 0;
    led2 = OFF;
    led3 = OFF;
    for(;;){
        if(timer250 <= base_time){
            switch_process();
            timer_process();
            timer250 += 250;
        }
    }
}

```

```

/* スイッチの初期化 */
/* LEDの初期化 */
/* タイマーの初期化 */

/* 割り込み許可 */
/* 現在のSW4の取り込み */
/* 現在のSW5の取り込み */
/* timer_process初期化状態 */

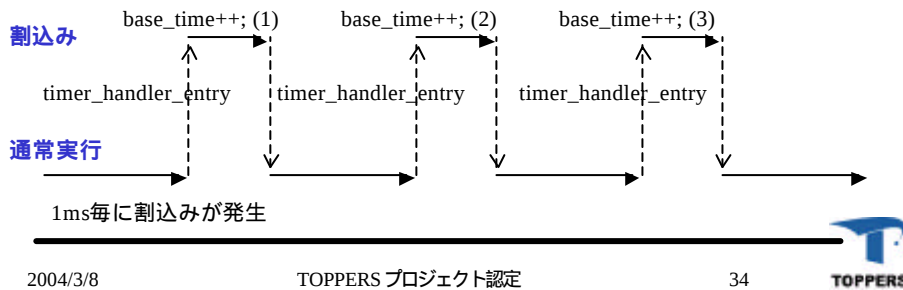
/* 永久ループ */

```

実装知識 2

割込み

- ハードウェアがCPUに対して、特別な動作 (timer_handler_entry) を要求したい場合、割込みを発生させることができる
- timer_handler_entryを割込みベクトルテーブルにセットし、ハードウェアに割込みの設定する



```
/*
 * スイッチプロセス
 */
void
switch_process(void)
{
    unsigned char sw;

    sw = get_key(SW5);
    if(sw5 != sw){
        if(sw == ON)
            event |= EVT_TIMER_START;
        else
            event |= EVT_TIMER_STOP;
        sw5 = sw;
    }
    sw = get_key(SW4);
    if(sw4 != sw){
        if(sw == ON)
            event |= EVT_TIMER_COUNT;
        sw4 = sw;
    }
}

/*
 * タイマーA0割込み関数
 * 1ms単位に起動される
 * base_timeは起動からのミリ秒のカウントを持つ
 */
void
timer_handler_entry(void)
{
    base_time++;
}
```

実装知識 3

ソース・コードを検証しよう

- プロセス1 (キーの管理) をswitch_process()として実装、プロセス2 (時間の管理) をtimer_process()として実装した
- 2つのプロセスはmain()より 250msごとにコールされ、実行する
- base_timeは割込み機能により 1ms間隔でインクリメントされる。main()はbase_timeをポーリングして監視する

```
/*
 * タイマ用プロセス
 */
void
timer_process(void)
{
    if(event != 0){
        if(event & EVT_TIMER_START){
            timer_state = STATE_ACT_TIMER;
            current_time = 0;
            max_time = 60*1000;
        }
        if(event & EVT_TIMER_STOP){
            timer_state = STATE_STOP_TIMER;
            alarm = 0;
        }
        if(event & EVT_TIMER_COUNT){
            if(timer_state == STATE_ACT_TIMER){
                max_time += 60*1000;
            }
        }
        event = 0;
    }
    switch(timer_state){
        case STATE_ACT_TIMER:
            if(current_time >= max_time){
                alarm = 15*4;
                timer_state = STATE_TIMEOUT;
            }
            else if((current_time % (10*1000)) == 0){
                alarm = 1*4;
            }
            current_time += 250;
            break;
```

考察

割り込みとスケジューリングによる実装

- 小さな開発ならばすっきりまとまっている
例題が簡単なのでうまく実装しているが
複雑なシステムではモジュール化が難しそう
ハードウェア依存部の実装は良くわからない
- 実際のシステム負荷はわからない
プログラムがどれだけシステムに負荷があるかわからない
→ リアルタイム性の保証

```
/*
 * 前ページからの続き
 */
case STATE_TIMEOUT:                                /* タイムアウト状態 */
    timer_state = STATE_STOP_TIMER;
    break;
default:                                             /* タイマー停止中状態 */
    break;
}
if(++sec > 3){                                       /* 時間の表示 :1秒経過 */
    led3 ^= ON;                                     /* LED3 のオン、オフを切り替える */
    sec = 0;
}
if(alaem > 0){                                       /* 警告の要求 :250ms経過 */
    led2 ^= ON;                                     /* LED2 のオン、オフを切り替える */
    alaem--;
}
else                                                /* 警告の要求がなくLED2がオンなら消灯 */
    led2 = OFF;

set_led(LED2, led2);                               /* LED2の設定 */
set_led(LED3, led3);                               /* LED3の設定 */
}
```

実装 テスト

実際にFlash ROMに書き込んで実行してみよう

1. ボードのCNVSSをショートさせた状態で電源をオン
2. MTOOL → flashsta → FlashSta.exeを直接実行しNonOsTimer0.motをFlash ROMに書き込みます
3. 終了後、電源をオフし、CNVSSのジャンパーピンをはずします

FlashStaの実行

マイコンピュータからMYTOOL¥flashstaデレクトリに移動します。

FlashSta.exeをダブルクリックして実行します。

Select Programメニューが表示されたら、Port を選択してOK を押してください。

ID Checkメニューが表示されますので、Referを押してください。

ファイルを開くメニューが表示されたら、TOPPERS¥OBJ¥TIMER0 デレクトリのNonOsTimer0.mot を開いてください。

ID CheckメニューにFilePathやIDが表示されますので、OKを押してください。

M16C Flash Startメニューが表示されますのでEraseを押してFlash ROMのクリアを行います。

Erase終了後、M16C Flash Startメニューより、Programを押してNonOsTimer0の書き込みを行います。

Program終了後、Exitを押してFlashStaを終了させてください。

テストや動作確認をしましょう

ピンポイント型テストをしましょう

1. 電源をオンしてLED3の点滅を確認します
2. スイッチ5をオン、オフしてタイマーが起動することを確認します
3. 一分で、タイムアップを知らせることを確認します
4. スイッチ4をオン、オフしてタイムアップ時間が延長されることを確認します
5. 自由にテストをしてみてください

電源オンで、カップラーメン・タイマーは起動します。

2つのスイッチを使って、仕様通りにタイマーが動作することを確認しましょう

リアルタイムOSとは何か

1. 開発環境の確認
2. 組込みとは、SESSAMEの復習
3. SESSAME版教材の動作確認
4. リアルタイムOSとは何か
5. カップラーメンタイマーの設計

ここでは TOPPERS :ITRON4.0のようなRTOSを用いて、システム設計を行う意味について説明を行います。続いて、ITRON4.0仕様の簡単な説明とタスクとタスクスケジューリングについての説明を行います。

その後、ボードを用いて、TOPPERS:ITRON4.0上のサンプルプログラムtimer1の実行とタスクモニターの動作確認、モニターを用いたデバイスの制御の実験を行います。

μITRON4.0仕様書Ver 4.01.00は以下のサイトからPDFでダウンロード可能です。

<http://www.assoc.tron.org./jpn/document.html>

オペレーティングシステム(OS)の役割

OSには明確な定義がない

- コンピュータの持つ資源を抽象化・仮想化・多様化したもの

アプリケーションから資源へのアクセスを容易に

アプリケーションのポータビリティを向上させる

- コンピュータの持つ資源を安全・効率的に一括管理したもの

複数のアプリケーションが同時にアクセスできるなど、資源の有効利用を可能にする

資源に対するアクセス権の管理を行い安全性を向上

オペレーティングシステムの主目的は、コンピュータシステムの効率的運用管理であり、これを追い求めることにより、進化してきた。それにもハードウェアの急速な進歩に対応するために、ソフトウェアがハードウェアに依存しない手段、また、ユーザの使い勝手を向上させる手段としても、大きな意味を持つ。

オペレーティングシステムの一部として、ファイルシステムは、ディスクというハードウェア資源を仮想化、多重化すると共に、安全・効率的に一括管理したシステムといえる。

リアルタイムOS (RTOS)とは？

文字通りリアルタイムシステム構築のためのOS

何をもってリアルタイムシステム構築のためといふかはOSによって立場が異なる

- 予測可能性を持つ
OSの各サービス時間があらかじめわかっている
- リアルタイムシステム向けの機能を持つ
優先度継承や優先度上限プロトコルのサポート
- 時間制約を管理（一部の研究ベースで対応）
OSが各処理の時間制約を考慮してスケジューリング
- 高速に応答

リアルタイムシステムの定義は一通りでない。定義としては「処理結果の正しさが、出力される結果値の正しさに加えて、結果を出す時刻にも依存するようなシステム」を言います。リアルタイムシステムは単に速い応答を求められるシステムという意味ではありません。多くの組み込みシステムはリアルタイムシステムです。この中には、一部の処理にリアルタイム性が求められるものも含まれます。リアルタイムシステムの中で、クリティカルなリアルタイム性を求められるシステムをハードリアルタイムシステムと言います。

上記の要件の中で、商用のリアルタイムOSの多くは、予測可能性を持ちますが、OSにより、そのレベルはまちまちです。また、リアルタイムシステムを構築する為のいろいろな機能をサポートしています。時間制約の管理はまだ、研究レベルです。

同期処理中、優先度の低いタスクが優先度の高いタスクを待たせてしまう状況がおきえる、高優先度タスクがクリティカルな処理を行う場合、問題となる。その解決として優先度継承などの対応があります。

・優先度継承 (Basic Priority Inheritance Protocol)

高優先度のタスクを待たせている低優先度のタスクに、高い優先度を持たせる方式

・優先度上限プロトコル (Priority Ceiling Protocol)

クリティカルなセクションに入る時に、chain blockingが起こらないように、早めにブロッキングを行う方式

リアルタイムOSの構成

リアルタイムカーネル

- RTOSの中心になるモジュール
- どのようなシステムにも共通する資源の扱いが中心
プロセッサ、メモリ、タイマ、...
- 保護機能を持たないものが多い
- リアルタイムモニタ、リアルタイムエグゼクティブと呼ばれることもある

リアルタイムカーネルのみ用いる場合は
「RTOS = リアルタイムカーネル」

OSの機能面から見た組込みシステムの特性としては以下の2つ

・保護の為の機能は必須ではない

組込みソフトウェアは、組込み対象の機器を制御することのみを目的に設計されます。これはソフトウェア自体が機器に固定されているということです。従って、デバック、テストが終われば、アプリケーションソフトウェアは信頼できるという前提が成り立ちますので、必ずしも保護の為の機能は必要としません。

・必ずしもサポートしなければならないI/Oはない

操作に時間のかかる資源をカーネル上に実現した方がよいという考えがあります。I/O装置はプロセッサやメモリに比べ低速なものであり、カーネル上には最低限のものしか置かれません。また、多くの組込み機器で共通に持つI/O機器はありませし、複数のアプリケーションで同一のI/O装置を共有する状況も少ない為です。但し、最近、汎用システムと近い性質を持った組込みシステムも多くなってきています。(PDAや携帯電話等)

リアルタイムOSの主な機能

■ リアルタイムカーネルの機能

- ・マルチタスク機構 → プロセッサの仮想化・多重化
- ・タスク間通信・同期機構
- ・割り込み管理 / 処理、例外管理 / 処理
- ・時間同期 / 管理 → タイマの仮想化・多重化
- ・メモリ管理 → メモリの仮想化・多重化
- ・システム管理 など

■ リアルタイムカーネル外の機能

- ・入出管理機能
- ・通信・ネットワーク機能
- ・ユーザインターフェイス機能 (GUIなど)
- ・プログラムモジュール管理機能 などなど

マルチタスク機構の説明

タスクとは、プログラムの並行実行の単位です。1つのタスク中のプログラムは逐次的に実行されます。また、異なるタスクのプログラムは並行して実行されます。これは、プロセッサを仮想化・多重化しているといえます。マルチタスク機構とは、複数のタスクを擬似的に並行実行するための機構です。もちろん、シングルプロセッサでは、同時に実行できるタスクは1つです。実際には、複数のタスクが同時に実行しているかのように見せているに過ぎません。

時間同期 / 管理の説明

ITRON4.0では、相対時間によるイベント発生とシステム時刻を管理する。各タスクは別々の相対時間を用いたイベントにより動作することが可能であり、タイマを仮想化しているといえる。

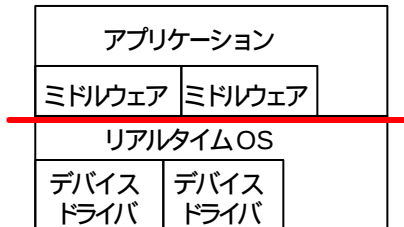
メモリ管理

UnixやWindowsのような汎用OSでは仮想メモリ管理により、ユーザプログラムは個々のメモリ空間で実行する。メモリ保護機能を持たないITRONでも、メモリプール機能により動的にメモリ領域の管理を行うことができる。

アーキテクチャによるリアルタイムOSの分類

汎用OS型とリアルタイムカーネル型

■ 汎用OS型



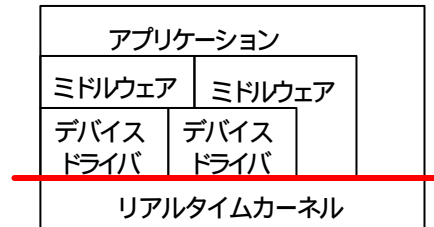
OSの機能は豊富

サイズは比較的大きい

一般に応答時間は遅い

デバイスドライバは別のAPIで作成

■ リアルタイムカーネル型



カーネルの機能は限定

カーネルサイズは小さい

一般に応答時間は速い

デバイスドライバとアプリケーションは同じAPI

OSの役割

- ・コンピュータの持つ資源を抽象化・仮想化・多重化する

これにより、アプリケーションから資源へのアクセスを容易にし、アプリケーションからのポータビリティを向上します。

- ・コンピュータの持つ資源を安全・効率的に一括管理する

複数のアプリケーションが同時に資源にアクセスできるなど、資源の有効利用を可能にします。資源に対するアクセス権の管理を行い安全性が向上します。

どの範囲のソフトウェアをOSとするかの、明確な定義はありません。例えばウインドウズシステムのような大きなソフトウェアも、役割から考えればOSの一部といえます。しかし、組込みシステム用のリアルタイムカーネルと比べれば、規模や機能が大きく違います。

リアルタイムOSを使用する利点は？

ソフトウェアの構造化が容易に

■ ソフトウェアの構造化が容易となる

構造化による生産性、保守性、信頼性の向上

(構造化設計手法のモジュール化とはアプローチが異なる)

■ 時間制約を持った多重処理システムの構築を容易に

➡ ソフトウェア部品を用いたソフトウェア開発の基盤となる

■ ハードウェア (特にプロセッサ) の違いを隠蔽

ハードウェアの詳細を知らなくても、上位のソフトウェアの開発が可能に。

リアルタイムOSを使用するメリットの詳細

・タスクを用いたソフトウェアの構造化

独立した処理の流れを独立したタスクに割り当てることにより、同一のプロセッサで複数のサブシステムの処理が行えたり、別々の入出力装置やイベントに対する処理などが可能になります。これにより、論理的な処理の順序と時間的な処理の順序を分離することができます。タスクの単位でプログラムは論理的な処理の順序で記述し、時間的な処理順序に従って実行が可能となります。

・時間制約を持った多重処理システムの構築を容易に

論理的な処理順序と時間的な処理順位を分離することにより、時間制約を持ったソフトウェアの保守性・再利用性が向上します。そのため、ソフトウェア部品を用いたソフトウェア開発 (コンポーネントベース開発) の基盤となります。

・ハードウェア (特にプロセッサ) の違いを隠蔽

ハードウェアの詳細を知らなくても、アプリケーションの構築が可能です。プロジェクトにまったく必要ないということではありません。「リアルタイムカーネルは、プロセッサのデバイスドライバ」と言われます。

リアルタイムOS使用の欠点は？

やはり、OSの中身を知る技術者は必要

- OSのオーバーヘッドによる実行性能の低下
- OS部、または、タスク使用によるメモリの消費

ITRON、TOPPERS/JSPはメモリ消費の少ないリアルタイムOSのひとつです。

- 障害解析時、OS内部に絡む解析が必要となる場合がある



OSの内部を知る技術者も
プロダクト内に必要

リアルタイムOSを使用するデメリットと回避策 (詳細)

- ・ OSのオーバーヘッドによる実行性能の低下
- ・ OSによるメモリの消費

OSを使用するメリットと引き換えに、メモリ効率は低下します。但し、実装技術の進化でメモリサイズはかなり小さくなっています。例として、今回教材のOAKS16 mini版のRAMサイズは2KBで3つのタスクが動作しています。

- ・ 非決定性が増すことによるデバック・テストの困難性

行儀のよい、タスク間同期・通信の困難性が必要です。モデルベース設計やシミュレーション等のソフトウェアの可視化を促進する技術等の対応も有効でしょう

- ・ OSのブラックボックス化による解析の困難性

リアルタイムのOSの内部を理解しましょう。OSのモニタリング等、種々の開発ツールの利用でかなりの改善が可能です。

μ IRON仕様カーネルの機能

(μ IRON4.0仕様)

■ カーネルの機能

- 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

- システム構成管理機能

■ サービスコール数

●フルセット

サービスコール: 166 (13*)
静的API: 21

●スタンダードプロファイル

サービスコール: 70 (13*)
静的API: 11

●自動車制御用プロファイル

サービスコール: 43 (1*)
静的API: 8

●最小セット

*カッコ内は非タスクコンテキスト専用
サービスコール (内数)

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カーネルに対して要求を出す為に使うインターフェイスです。一般的なカーネルの機能について説明を行います。

・タスク管理機能

タスクの状態を直接的に操作 / 参照する為の機能です。タスクの生成と削除、起動と終了、タスクに対する起動要求をキャンセル機能、タスクの優先度の変更や参照の機能が含まれる。

・タスク付属同期機能

タスクの状態を直接的に操作することによって同期を行うための機能です。タスクを起床待ちにする機能とそこから起床する機能、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、強制待ち状態へ移行する機能、自タスクの実行を遅延する機能が含まれる。

・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、タスクのコンテキストで行う為の機能です。タスク例外処理ルーチンを定義する機能、タスク例外処理を要求する機能、タスク例外処理を禁止・許可する機能、タスク例外処理に関する状態を参照する機能が含まれる。

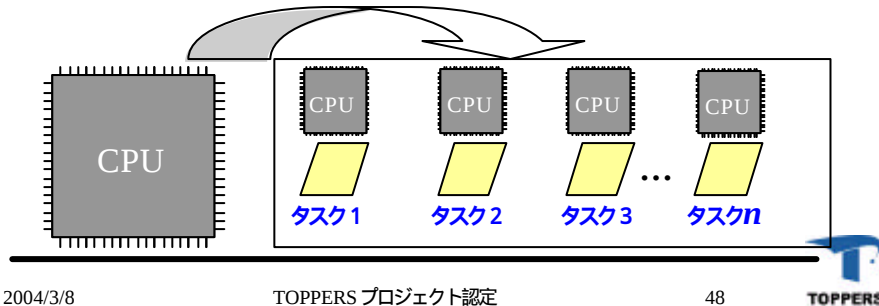
・同期・通信機能

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・通信を行う為の機能です。セマフォ、イベントフラグ、データキュー、メールボックス等の機能があります。

タスクとは

タスクを使って、小さく考える

- タスクとはプログラムの並行実行の単位
- プロセッサを仮想化・多重化したもの
- タスクの恩恵
 - 各タスクが専用CPUを持つかのように見える
 - 大きな問題を小さく分割して考える



タスクは目的を持ったプログラムの実行単位です。ひとつのタスクの中のプログラムは逐次的に実行されます。異なるタスクのプログラムは疑似的に並行して実行されます。リアルタイムOSはマルチタスク機構をサポートしています。マルチタスク機構とは複数のタスクを疑似並行に実行するための機構です。シングルプロセッサでは、実際に同時に実行できるタスクは1つのみです。複数のタスクが同時に実行しているように見せる機能です。

タスクの状態遷移を図解化する



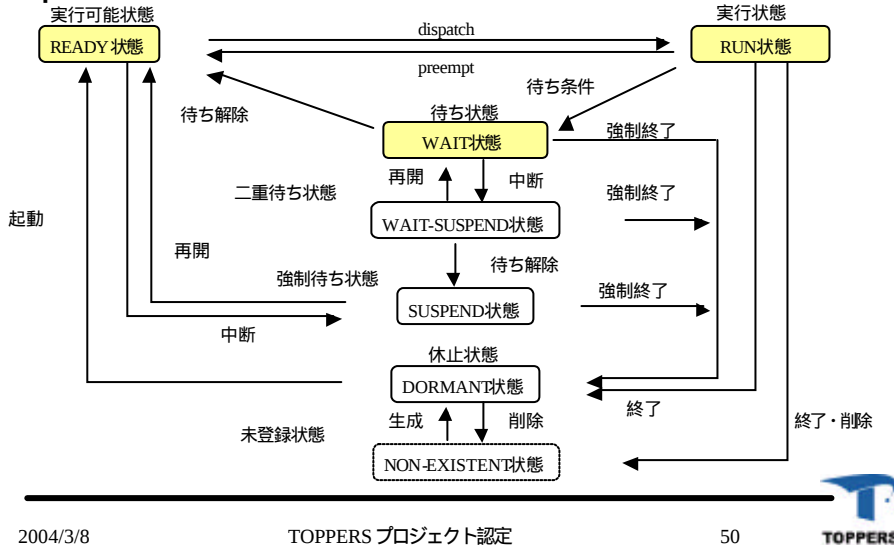
なぜ複数のタスクが必要なのでしょう？リアルタイムOSの場合、汎用システムのマルチタスク (TSS) とは動機が異なります。リアルタイムOSでは、独立した処理の流れを独立したタスクに割り当てることにより、同一のプロセッサが複数のサブシステムを処理できるようになり、別々の入出力装置やイベントに対する処理が可能にすることを目的にしています。

これはどのように実現されるのでしょうか？システムを論理的な処理の順序と時間的な処理の順序に分離します。それを論理的な処理順序でプログラムとして記述し、時間的な処理順序に従って実行していきます。この組み換えにより、プログラムの保守性・再利用性が向上し、外部で開発されたソフトウェアの部品の導入を容易にします。

タスクの状態遷移

ある瞬間、実行中のタスクはひとつだけ

■ μ IRON4.0仕様のタスク状態遷移



タスクの状態

(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る間は、特別に規定される場合を除いて、非タスクコンテキストの実行を開始する前に実行状態のタスクをさす。

(2)実行可能状態(READY)

そのタスクの実行する条件は整っているが、そのタスクより優先順位が高いタスクが実行中であるために、そのタスクを実行できない状態をさす。

(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した為に実行が中断された状態を指す。

(4)強制待ち状態(SUSPENDED)

他のタスク等により、強制的に実行を中断させられた状態を指す。

(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

(6)休止状態(DORMANT)

タスクがまだ起動していないか、実行を終了した後の状態を指す。この状態の場合、実行状態を表現する情報は保存されない。

(7)未登録状態(NON-EXISTENT)

タスクが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状態

基本的な状態遷移のサービスコール

タスク管理機能、タスク付属同期機能

■ タスク管理機能

CRE_TSK	タスクの生成 (静)
act_tsk, iact_tsk	タスクの起動
can_act	タスク起動要求の キャンセル
ext_tsk	自タスクの終了
ter_tsk	タスクの強制終了
chg_pri	タスク優先度の 変更
get_pri	タスク優先度の 参照

■ タスク付属同期機能

slp_tsk	起床待ち
tslp_tsk	起床待ち (タイムアウトあり)
wup_tsk, iwup_tsk	タスクの起床
can_wup	タスク起床要求の キャンセル
rel_wai, irel_wai	待ち状態強制解除
sus_tsk	強制待ち状態への移行
rsm_tsk	強制待ち状態から再開
frsm_tsk	強制待ち状態からの 強制再開
dly_tsk	自タスクの遅延

静的API

μITRON4.0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が標準化され、静的APIが規定されました。このスライドで大文字記述しているものが静的APIです。静的APIはコンフィギュレーション・ファイルというファイルに記述しておき、コンフィギュレータ (TOPPERS¥cfg¥cfg.exe) を用いてインクルード・ファイル (kernel_id.h) とC言語ファイル (kernel_cfg.c) を生成します。その他のサービスコールはC言語のインターフェイスで記述されています。CRE_TSKを例として示します。

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

【パラメータ】

ID tskid	:タスクのID番号
ATR tskatr	:タスク属性((TA_HLNG TA_ASM) [TA_ACT])
VP_INT exinf	:タスクの拡張情報
FP task	:タスクの起動番地
PRI itskpri	:タスクの起動時優先度
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

マルチタスク機構の構成要素

マルチタスクを実現するもの

- マルチタスク機構とは、複数のタスクを疑似実行させるための仕組み
実際にはひとつしか実行していないタスクを同時に実行しているよかのように見せる
- ディスパッチ
(ディスパッチャ : 切り替えモジュール)
- スケジューリング
(スケジューラ : タスク決定モジュール)
- スケジューリングアルゴリズム

マルチタスク機構の構成要素

・デスパッチ (タスクディスパッチ、タスクスイッチ)

プロセッサが実行するタスクを切り替えることを指します。デスパッチを実現するモジュールをデスパッチャと呼びます。

・スケジューリング (タスクスケジューリング)

どの時間にどのタスクを実行するかを決定することを指します。多くのリアルタイムOSでは、次に実行するタスクを決定する処理を指します。スケジューリングを実現するモジュールをスケジューラと呼びます。

・スケジューリングアルゴリズム

次に実行するタスクを決定する方式を指します。TOPPERS/JSPを含め、ほとんどのリアルタイムOSがプロエンプティブな優先度ベースアルゴリズムをサポートしています。もちろん、他のスケジューリングアルゴリズムをもつリアルタイムOSもあります。

ITRONのスケジューリングアルゴリズム

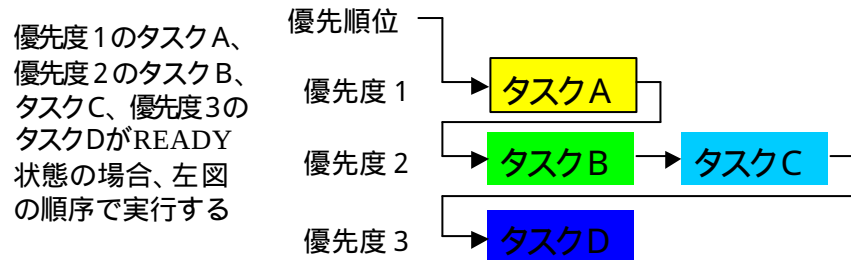
プリエンプティブな優先度ベーススケジューリング

■ 優先度ベーススケジューリング

最も優先度の高いタスクが実行される

優先度の高いタスクが実行できなくなるまで、優先度の低いタスクは実行されない

同一優先度タスク間ではFCFS(First Come First Served)



優先度ベーススケジューリング

イベント駆動スケジューリングの代表的なリアルタイムスケジューリング方式です。実行が必要な処理のなかで、最も高い優先度を持つものを実行します。優先度の高い処理の実行が完了する（または別の理由で実行できなくなった場合）まで、優先度の低い処理は実行されません。プリエンプティブとノンプリエンプティブな実装が可能です。優先度の設定については、静的な優先度割付けと動的な優先度割付け、動的な優先度変更が行えます。

ITRONの優先度は1以上の数字で表され、数字の値が小さい方が優先度が高い設定となります。

教材を用いた確認

教材の動作確認にタスク・モニターを使います。

タスク・モニターはリアルタイムOSで提供される機能のひとつである「論理的な処理との順序と時間的な処理の順序の分離」のひとつの例です。このような、機能をノンOSのシステムに組み込もうとすると、メインプログラムや各々のプロセスに、いろいろな仕掛けを入れなければなりません。

タスクモニターで実行できる機能はヘルプコマンドで、一覧表を表示することができます。

>help ←

タスクの遷移状態を確認してみよう

教材を使った確認

1. TIMER1デレクトリ中のプロジェクト (Jsp14Timerm)をBuildします
 2. timer1は1秒毎にLEDを点滅させるタスクと、SWの状態をLEDに表示するタスクと、タスクモニターの3つのタスクが動作するプロジェクトです
 3. Jsp14Timer1m.motをFlash ROMに書き込み、ターミナルを起動させてボードに電源を投入するとタスクモニターが立ち上がります
- タスクモニターとは、タスクとして実行し、他のタスクの管理や監視をするプログラムです

1.timer1.cfg

timer1プロジェクトのコンフィギュレーションファイル、entry_task、timer_task、タスクモニター用タスクの生成、シリアルドライバの為に割込みの設定、JSPで使用するタイマーの設定を行います。

2.timer1.c

```
void timer_task(VP_INT exinf);
```

タイマータスク、1秒単位にLED3を点滅させます。

```
void entry_task(VP_INT exinf);
```

キーとLEDのデバイスの初期化を行います。タイマーはTOPPERS/JSPで管理している為、初期化の必要はありません。その後、500ミリ秒毎に起動スイッチ5がオンならばLED2をオン、オフならばLED2をオフします。

タスクの遷移状態を確認してみよう

タスクモニターを起動します

- ターミナルソフトを起動し、
ボードの電源をオン
- >display task ←
タスクの状態の確認
- >set task 1 ←
対象タスクの選択
- >task activate ←
タスクの状態設定



タスクモニターはプロンプトに対するコマンドの設定によって動作します。コマンドの書式は [cmd] [(type)] [(parameter)][return] です。type と parameter は省略可能な場合があります。[return] は Entry キーの押下を示します。cmd と type は大文字小文字の区別はなく、最初の一文字だけで認識が可能です。スライドの例で

>display task ←

は

>d t ←

で実行可能です。

タスクの実行状態の確認

タスクの停止、再開、タスク負荷

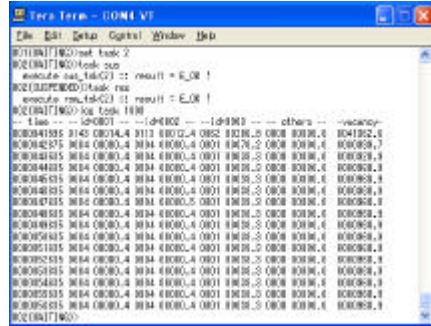
- タスク 2 に切り替えて suspend 状態に切り替えます

```
>set task 2 ←
```

$$\triangleright \text{task suspend} \leftarrow$$

- 再開し、ログタスク機能でシステムの負荷を見ます

```
>task resume ←
```

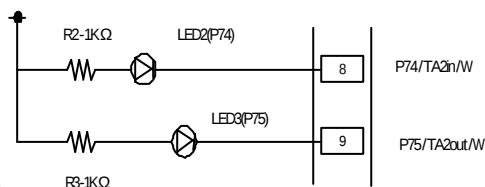
$$>\log \text{ task } 1000 \leftarrow$$


タスク状態を、SUSPEND状態に切り替えることにより、LED3は点滅しません。

LOG TASKにより、タスクの実行状態を監視できます。1000は1000msを示し、監視のタイミングを設定します。つまり、1秒(1000ms)中のタスクの実行回数と実行時間を(ms)を表示します。timer1では、タスク1(entry_task)も、タスク2(timer_task)も、1秒間に1ms以内の実行となりますので、システムの負荷はほとんどないということがわかります。但し、タイマー割込みはvacancyの中に含まれますので実際にどれくらいの負荷があるかは、このタスクモニターでは検出することはできません。

どのようなハードウェアで制御するのか

- LEDはSFRのポート7で制御します
- ポート7は出力用に設定されています
- LED2はポート7の5ビット、LED3は4ビットに接続されています
- LEDはプルアップされていますので、0“L”レベルに設定すると電流が流れて点灯します



タスクモニターのコマンド一覧

Display BYTE [start address] [endaddress] ← $\times \text{E!DUMP}(\text{byte})$

Display HELF [start address] [endaddress] ← $\times \text{HELDUMP}(\text{half word})$

Display WORD [start address] [endaddress] ← $\times \text{EIDUMP}(\text{word})$

Display TASK ← タスク状態の表示

Display REGISTER ← 待ち状態のレジスタ表示

Set BYTE [start address] ← メモリ設定(byte)

Set HELF [start address] ← メモリ設定(helf word)

Set WORD [start address] ← メモリ設定(word)

Set COMMAND [mode] ← コマンド設定の変更

Set SERIAL [port id] ← [表示コンソールの変更](#)

Set TASK [task id] ← タスクの選択

Task ACTIVATE ← ack tskの実行

Task TERMINATE ← ter tskの実行

Task SUSPEND← sus tskの実行

Task RESUME ← rsm tskの実行

Log MODE [logmask] [lowmask] ← ログ表示モードの変更

Log TASK [cycle time(ms)] ← タスクの実行状態の表示

Log PORT [no] [logno] [portaddress] ← ポートアクセスのログ表示

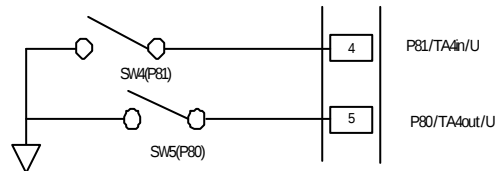
スイッチをセンスしてみよう

どのようなハードウェアで制御するのか

- スイッチはSFRのポート8に接続されています
- ポート8は入力用に設定されています
- スイッチ4はポート8の1ビット、スイッチ5はポート8の0ビットに接続されています
- スイッチはグランドに接続されていますのでスイッチオンで0“L”レベルに変化します

ポートP8レジスタ

ビット番号	ビット名	機能	初期値
P8.0	ポートP8ビット0	入力ポートに接続した外部デバイスで発生するビットを監視し、検出レベルが指定される。	100
P8.1	ポートP8ビット1	検出レベルが指定された外部デバイスで発生するビットを監視し、検出レベルが指定される。	100
P8.2	ポートP8ビット2	検出レベルが指定された外部デバイスで発生するビットを監視し、検出レベルが指定される。	100
P8.3	ポートP8ビット3	検出レベルが指定された外部デバイスで発生するビットを監視し、検出レベルが指定される。	100
P8.4	ポートP8ビット4	検出レベルが指定された外部デバイスで発生するビットを監視し、検出レベルが指定される。	100
P8.5	ポートP8ビット5	検出レベルが指定された外部デバイスで発生するビットを監視し、検出レベルが指定される。	100
P8.6	ポートP8ビット6	検出レベルが指定された外部デバイスで発生するビットを監視し、検出レベルが指定される。	100
P8.7	ポートP8ビット7	検出レベルが指定された外部デバイスで発生するビットを監視し、検出レベルが指定される。	100



2004/3/8

TOPPERS プロジェクト認定

60



この教材では、ハードウェアとのインターフェイスはデバイス・インターフェイスによって隠蔽されています。そのため、直接ハードウェアの設定を行う必要はありません。ポート8は、TADR_SFR_P8で表されるアドレス0x3f0です。

/*

* スイッチの状態を取り出す

* arg1:SW1またはSW2

*/

int get_key(int sw)

{

volatile UB key;

int result = OFF;

key = sil_reb_mem((VP)TADR_SFR_P8);

switch(sw){

case SW4:

if((key & P8_SW2) == 0)

result = ON;

break;

case SW5:

if((key & P8_SW1) == 0)

result = ON;

break;

default:

break;

}

return result;

}

モニターを使ってSFRを制御

モニターにて、ポートを直接確認

- ポート7のアドレスは3EDh、ポート8のアドレスは3F0h
- >set 3ed ← でポート7に16進を設定し、LEDの点灯、消灯を確認します
- スイッチをオン、オフ後、>display 3f0 ← でポート8の内容を表示し、スイッチを確認します



2004/3/8

TOPPERS プロジェクト認定

61



ポート8は、TADR_SFR_P7で表されるアドレス0x3edです。

/*

* LEDの設定を行う

* arg1:LED1またはLED2

* arg2:ONまたはOFF

*/

void set_led(int led, int req)

{

volatile UB rled = cled;

switch(led){

case LED2:

rled &= ~P7_LED2;

if(req == OFF)

rled |= P7_LED2;

break;

case LED3:

rled &= ~P7_LED1;

if(req == OFF)

rled |= P7_LED1;

break;

default:

break;

}

if(cled != rled){

cled = rled;

sil_wrb_mem((VP)TADR_SFR_P7, cled);

}

}

カップラーメンタイマーの設計

1. 開発環境の確認
2. 組込みとは、SESSAMEの復習
3. SESSAME版教材の動作確認
4. リアルタイムOSとは何か
5. カップラーメンタイマーの設計

カップラーメンタイマーのTOPPERS/JSP版を作成してみましょう

この開発環境では、デバックが使用できませんので、タスクモニターやシステムログのダンプ機能を使ってデバックしましょう

TOPPERS/JSPを使用したタイマーの設計 1

サンプルプログラムの紹介

- 2つのタスクとモニタータスクの構成
- ENTRY_TASK: SW 5の状態をLED2に対応させる
- TIMER_TASK: LED3を1秒毎に点滅させる
- ENTRY_TASKが起動させれば、act_tskによってTIMER_TASKを実行する
- デバックの方法は、syslog_n関数を使って現状の状態をログで表示。または、タスクの状態をモニターで確認

システム・ログは、ログを要求したタスクから直接表示を行うのではなく、ログテーブルに蓄積してからLOGTASKにて、表示を行う。非タスクコンテキスト中からの表示も可能です。また、LOGTASKの優先度を下げることにより、通常のタスクの実行を阻害せず。ログ表示を行うことが出来ます。但し、LOGTASKに実行権が長期にわたり与えられない場合、ログメッセージをロストする場合があります。

TOPPERS/JSPを使用したタイマーの設計 2

コンフィギュ・ファイルの説明 (timer1m.cfg)

■ CRE_TSKでタスクの定義

```
CRE_TSK(ENTRY_TASK, { TA_HLNG, 0,
entry_task,DEFAULT_MAIN_PRIORITY,STACK_SIZE, NULL});
CRE_TSK(TIMER_TASK, { TA_HLNG, (VP_INT) 0, timer_task,
TIMER_PRIORITY, STACK_SIZE, NULL });
```

- Serial.cfgにてシリアルドライバの設定、ドライバの初期化と割り込みの設定
- monitor.cfgにてモニタータスクの設定
- timer.cfgにてタイマーの割り込みとタイム・サービスの設定

```
#define _MACRO_ONLY
```

```
#include "../timer1.h"
```

```
INCLUDE("¥timer1.h¥");
```

```
CRE_TSK(ENTRY_TASK, { TA_HLNG, 0, entry_task, DEFAULT_MAIN_PRIORITY,
STACK_SIZE, NULL});
```

```
CRE_TSK(TIMER_TASK, { TA_HLNG, (VP_INT) 0, timer_task, TIMER_PRIORITY, STACK_SIZE,
NULL });
```

```
#ifdef CPUEXC1
```

```
DEF_EXC(CPUEXC1, { TA_HLNG, cpuexc_handler });
```

```
#endif /* CPUEXC1 */
```

```
/* イレギュラー割り込みの設定 */
```

```
DEF_INH(INT_IRG, { TA_HLNG, irregular_int_handler });
```

```
DEF_EXC(EXC_IRG, { TA_HLNG, irregular_ext_handler });
```

```
#include "../MONITOR/monitor.cfg"
```

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

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

TOPPERS/JSPを使用したタイマーの設計3 カップラーメン・タイマー化の考察 (ENTRY_TASK)

- ハードウェアの初期化(initial_key, initial_led)
- 500ms毎に、タスクを起動する。それ以外は待ち状態(WAIT状態)。tslp_tsk()にてタスク状態の切り替え
- SW5の状態を取り込み(get_key())、変化すればLED2の状態を書き換える(set_led())
- sw = get_key(SW5);
SW5の状態を取り出す。取り出した状態はON、または、OFF、詳細はdevice.cを参照

```
/*
 * エントリタスク
 */
void entry_task(VP_INT exinf)
{
    UB sw5, sw;
    UB led2 = OFF;

    syslog_1(LOG_INFO, "Sample entry task starts (exinf = %d).", exinf);
    initial_key();           /* キーの初期化 */
    initial_led();           /* LEDの初期化 */
    sw5 = get_key(SW5);       /* 現在のSW5の取り込み */
    act_tsk(TIMER_TASK);     /* タイマータスクの起動 */
    for(;;){
        tslp_tsk(500);
        sw = get_key(SW5);
        if(sw5 != sw){
            syslog_1(LOG_INFO, "Change switch5 = 0x%x.", sw);
            if(sw == ON)
                led2 = ON;
            else
                led2 = OFF;
            sw5 = sw;
        }
        set_led(LED2, led2); /* LED2の設定 */
    }
}
```

TOPPERS/JSPを使用したタイマーの設計4 カップラーメン・タイマー化の考察 (TIMER_TASK)

- 現在時間を取り出し(get_tim(¤t_time);)、その時間に対して、250ms単位にタスクを起動する(tslp_tsk(base_time - current_time));
- base_time-current_timeが負の値となる場合？
- 1秒毎に、LED3の状態を切り替える
- set_led(LED3, led3);
LED3の表示状態を切り替える、led3は点灯(ON)、消灯(OFF)

```
/*
 * タイマ用タスク
 */
void
timer_task(VP_INT exinf)
{
    SYSTIM base_time, current_time;
    TMO    tmout = 0;
    UB     led3 = OFF;

    syslog_1(LOG_INFO, "Sample1 timer task starts (exinf = %d).", exinf);
    get_tim(&base_time);          /* 現在時間の取り出し*/
    for(;;){
        tslp_tsk(tmout);          /* TICK待ち */

                                   /* 奇数秒でLED3オン*/
        led3 = (base_time / T_1SEC) & ON;
        set_led(LED3, led3);      /* LED3の設定 */
        base_time += T_TICK;
        get_tim(&current_time);
        tmout = base_time - current_time;
        if(tmout < 0)
            tmout = 0;
    }
}
```

TOPPERS/JSPを使用したタイマーの設計 5

開発環境の確認

- 開発環境はTIMER2のディレクトリ上で行います
- TIMER2のディレクトリには、TIMER1と同じプログラムが用意されています
- プロジェクトファイルをJsp14Timer2mに切り替えます
- 開発終了後、どのような改造を行ったか確認してください