

TOPPERS基礎3実装セミナー (LPC2388/ASP版) プラットフォーム編:1日目

TOPPERSプロジェクト
教育ワーキング・グループ

2013/09/16

TOPPERSプロジェクト認定

1



本ドキュメントに関して

1. 著作権に関する表記

<TOPPERS基礎実装セミナー(LPC2388/ASP版:基本3) 1日目>

Copyright (C) 2011-2013 by 竹内良輔 (株) リコー
Copyright (C) 2011 by 福井徹 (株) デンソークリエイティブ

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

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

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

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

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

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

本ドキュメント中の商品名及び商標名は、各社の商標または登録商標です。

2013/09/16

TOPPERSプロジェクト認定

2



スケジュール

■ 1日目

1. プラットフォーム構築	0.7時間
2. 開発環境のセットアップ	1.3時間
3. RTCのデバドラの説明とデバッグ	1.0時間
4. MCI/FAT/POSIXファイルシステム	1.0時間
5. SDファイルシステムの確認	1.5時間
6. まとめ	0.2時間

概要

- 本教材は組込みプラットフォームについて学ぶ教材です
- 組込みプラットフォームを構築するためにはドメイン、汎用OS、ハードウェア、開発環境等に深い知識が必要となります。本教材はあくまで基礎的な部分についての実習教材となります
- 設計作業で発生するドキュメント化等の作業やプロセスについて、本テキストでは解説を行いません
- 組込みプラットフォームは単純な構成では以下の4つから構成されます
 - API(Application Program Interface)
 - ミドルウェア
 - ソフトウェアデバッガ(タスクモニタ等)
 - RTOS
- 本教材ではAPIとミドルウェアについて学習を行います

概要

- APIに関しては、1日目に汎用OSで使用するソフトウェアライブラリ、POSIX参照したAPI、2日目にITRON TCPIP APIについて学習します
- アプリケーションの例として、簡単なUNIXのシェル機能の実現方法について解説します
- デバイスドライバとしては、RTC、MCI、DMA、SDカードインターフェイス、ETHERNETドライバについて作成方法を解説、実習します
- ミドルウェアに関しては、1日目にファイルシステムについて、2日目にTCPIPプロトコルスタックに関して学習します。本実習はミドルウェアの下位インターフェイスとハードウェアを連結するデバイスドライバの作成や検証方法について実習で学習を行います

概要

- 実際の組み込み業務として、種々のデバイスドライバを作成し、ミドルウェアと組み合わせて組み込み機器を正しく動作させることが、組み込みソフトウェアエンジニアの最終的な業務となります
- ハードウェアを理解してデバイスドライバを正しく動作させることはかなりスキルの高い作業です
- この教材はC言語と μ ITRON-RTOSで組み込み開発を体験した技術者を対象としています
- なお、セミナーの時間制約上デバイスドライバの講義は細部にわたる説明を行うことができません。より深い理解を行うために、講義内容と添付のソースを参考に事後学習を行うことをお勧めします

プラットフォーム構築

1. [プラットフォーム概要](#)
2. 仮想端末機器
3. プラットフォーム構築の手順

組込み規模と開発手法

- ・ 組込み機器適用機種はUSBメモリからプラント機器まで多機種に及ぶ
- ・ 開発規模により開発手法が異なる
- ・ 便宜上、小規模、中規模、大規模に分類する
- ・ この講座では中規模用のプラットフォーム構築技術について講義を行う

開発規模	小規模	中規模	大規模
総ステップ数 ^{注1}	2万行以下	2万行～50万行くらい	50万行以上
デバッグ方法	ICEが主流	ソフトデバッグやICE	シュミレータ等を使用しパソコン上で開発
ベース環境	RTOS等を使用しない	RTOSやミドルウェアでプラットフォームを構築	汎用OSを使用する場合もある
人数 ^{注1}	10人未満	100人未満	100人以上

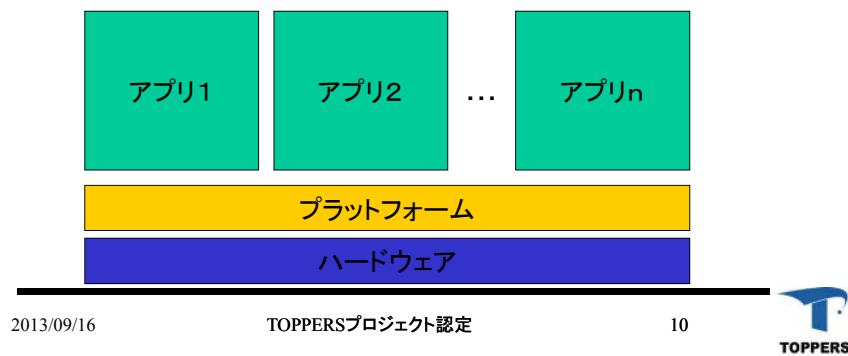
注1) 目安であり、プロジェクトによって異なる

組込みプラットフォーム技術者の必要性

- LINUXや汎用組込みプラットフォームを使ってシステム構築する
 - LINUXはオープンソースであるため、組込み用のハードウェアに特化した場合(特化しなくても)OSの改変やメンテナンス、開発環境のメンテナンスが必要となる。これらを外部に委託した場合コストメリットは低くなる
 - 市販の汎用組込みプラットフォームは開発環境を含んだ高価なものが多い、最終的なカスタマイズや不具合解析は使用者が行わなければならない
- いずれにしても、プラットフォームに特化したスキルを持った技術者は必要
- この講座はLINUXや他のプラットフォームを使用する場合でも必要なスキルを提供する

中規模組込みシステム構成

- 中規模組込みシステムを模式化すると、アプリケーションとプラットフォームで構成される
 - アプリケーション 機器の機能を実現するソフトウェア群
 - プラットフォーム 複雑な部分を隠蔽しアプリケーションにサービスを提供するソフトウェア群
- プラットフォームはハードウェアを隠蔽し、アプリケーションに最適化とポータビリティを与える



アプリケーションとプラットフォーム

- アプリケーションは機器の機能実現用のソフトウェア群
 - 多人数でより簡単に、効率よく開発できること
 - 組込み機器の仕様変更に従って、変更可能なこと
 - 大規模開発が可能なこと
- プラットフォームは汎用サービス用のソフトウェア群
 - 複雑部分を隠蔽するため小規模な方がよい
 - ハードウェアに依存した設計、作りこみが必要
 - アプリケーションに汎用的なサービスを提供
 - アプリケーションの改変に影響されない設計



理想的なプラットフォーム設計は、汎用OSの設計に観られる

組込みプラットフォームの設計指針

- 組込みプラットフォームには2つの側面がある
 - 汎用的なコンピュータ制御を行う側面
 - 組込み機器に特化した側面
- 汎用的なコンピュータ制御
 - 汎用OS設計思想が反映される
 - LINUX, Windows, MS-DOS, CPM等
 - ファイルシステムやネットワークは標準的な方がよい
 - 割込み、キャッシュ、ハードウェア制御等の隠蔽
- 組込み機器制御
 - 機器に特化したミドルウェアやハードウェアを効率よく制御する必要がある

組込みプラットフォームの構成

- 組込みプラットフォームは一般的には以下の部品で構成される
 - API (Application Program Interface)
 - ミドルウェア(デバイスドライバを含む)
 - ソフトウェアデバッガ(タスクモニタ)
 - RTOS
- APIはアプリケーションとのインターフェイス
 - APIはアプリケーションにサービスを提供する関数群
 - 機器固有のサービス化も必要となる
 - アプリケーションのポータビリティを向上させるためRTOSのサービスも隠蔽化することがある

API: 組込みプラットフォームの部品

- APIはApplication Program Interfaceの略でアプリケーションから使用できるプラットフォームの関数群。プラットフォームを特徴づける機能
- 通常は隠蔽化したAPIを作成する。種々の事情でRTOSやデバイスドライバのI/Fをそのまま使用する場合もある
- APIを用いてプラットフォームを使用する上での手続きや規約を定める
- 組込み用のAPIは以下の相反する特性を持つ
 - 組込み用のAPIは機器の特性に特化する
 - アプリケーションの変更に追従する汎用性を求める
- 最適なプラットフォーム設計には相反する特性を融合し最適化するノウハウが必要となる。これらは経験やスキルによる要因も多く、アプリケーション開発とは違った開発プロセスが必要となる

ミドルウェア:組込みプラットフォーム部品

- ミドルウェアとは、RTOSとアプリケーションの中間に位置し専門的な処理を行うソフトウェア群をさす
- ミドルウェアには、機器固有の専門機能を行うもの(専用ミドルウェア)と、多くの機器で汎用的に使用されるもの(汎用ミドルウェア)に分けられる
- 汎用ミドルウェアの例は以下のとおりで、オープンソースとして提供されるものもある
 - TCP/IPプロトコルスタック(TINET)
 - ファイルシステム(FatFs)
 - USBホスト・デバイス
- 専用ミドルウェアは企業秘密に属するものが多く、一般には公開されない

デバイスドライバ:組込みプラットフォーム部品

- ハードウェアとのインターフェイスを行う関数群
 - 開発にはハードウェアやIPに関する知識が必要
- ミドルウェア用のデバイスドライバ
 - ミドルウェアでハードウェアとのインターフェイスを規定しているものがある
 - 開発にはミドルウェアとハードウェアの両方の知識が必要となる
 - タスクと割り込みハンドラで構成される
 - USBデバイスが特に複雑
 - メーカーごとにIPの仕様が異なる
 - 割り込みハンドラが実装の大部分を占める
- 割り込み等はプラットフォームで隠蔽した方がよい

ソフトウェアデバッガ:組込みプラットフォーム部品

- 小規模なシステムではICEを使用する人が多い
- 多人数で組込み開発を行う場合、すべてのメンバー（特にアプリケーション開発者）にICEを配給できない
- 開発用、テスト時の問題解析用にソフトウェアデバッガをプラットフォームに実装した方が、問題解析しやすい
- ソフトウェアデバッガに求められる機能
 - 一般的なデバッグ機能
 - 機器固有の機能検証機能
 - 機器固有のモード設定機能
 - ストール時の問題解析機能
- プラットフォーム開発者はICE等のデバッグ機器は有用な開発手段となる

RTOS:組込みプラットフォーム部品

- 組込み機器のCPUと時間をオブジェクト化する
- マルチタスク機構により、複数のCPUを使用するようにアプリケーションを作成できる
 - 排他制御の問題が発生:プラットフォームで隠蔽した方がよい
 - リアルタイム性を保障するため、リエントラントな優先度ベーススケジューリングが多い
- 日本ではμITRON仕様のRTOSの使用が多い
 - RTOSについては、基礎2講座を参照
- 組込みシステムではRTOSのすべてのサービスコールを使い切る必要はない。機器の特性に合わせたサービスコールを使用した方がよい

ETロボコンTOPPERS/JSPプラットフォーム

- ETロボコンTOPPERS/JSPプラットフォームの構成
- ECROBOT (API)
 - アプリケーションへのアーキテクチャの提示
 - 共通メニューの表示
 - 電源、キーの管理
- Balancer Library (ミドルウェア)
 - 移動情報をモータ情報に変換
- デバイスドライバ
 - ハードウェアの管理
- リアルタイムOS (TOPPERS/JSP)
 - CPU、時間のオブジェクト化 (マルチタスク機能、タスク間通信等)
 - 実行環境の初期化、デバッグ環境

ECROBOT

Balancer Library

デバイスドライバ

TOPPERS/JSP
(リアルタイムOS)

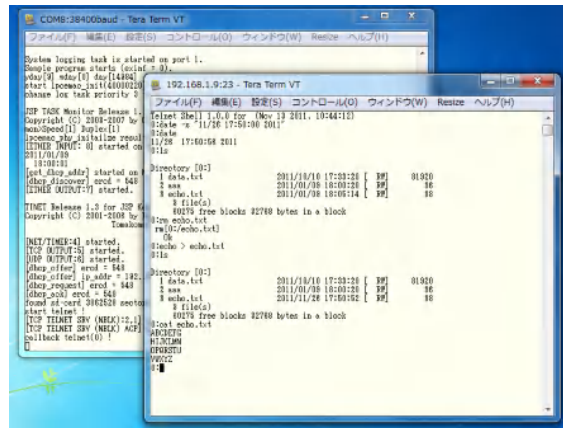
プラットフォーム構築

1. プラットフォーム概要
2. [仮想端末機器](#)
3. プラットフォーム構築の手順

教材の組み込みシステム

- 仮想端末を使用してSDカードの内容を検証するシステムを作成します

仮想端末はパソコンから
TeraTermを用いてTELNETにて
接続しUNIXライクなコマンドを用
いてSDカードの内容を確認でき
るものとします



仮想端末とは

- 仮想端末とは、コンピュータが高価だったころ、ホストコンピュータを端末(TTY)の形で、多くの人がシェアして使用するための機能です
- 本実習ではアプリケーションとして、組込みボード上に仮想端末アプリを構築します。そのために必要となるプラットフォームを想定し、実習上でプラットフォーム構築のための学習を行います



TELNET



TELNET

- TELNET (Telecommunication network)
- 汎用的な端末間通信を行う通信プロトコル
 - RFC854で規定
 - TCPの23ポートを使用する
- UNIXに複数の仮想端末を接続するために開発
- 仮想端末システムの構築には以下の環境が必要
 - Telnetクライアントプログラム: TeraTermを使用
 - Telnetサーバ
 - Telnetシェル: この講座のアプリケーション

POSIX

- POSIX (Portable Operating System Interface)
- IEEEで定められたUNIX OS等で使用するAPI
- UNIXアプリケーションソフトウェア開発用
- 規格の範囲は以下のとおり
 - カーネル用C言語インターフェイス
 - プロセス環境
 - ファイルとディレクトリ
 - システム環境
 - 開発環境等
- 本講義ではファイルシステムを使用するため、Telnetシェルで使いやすいファイル系にPOSIXを使用する

仮想端末システムのプラットフォームの機能

- LPC2388ボード上で仮想端末アプリを作成する上でプラットフォームに要求される機能
 - RTOS(TOPPERS/ASP/JSP)
 - TCP/IPプロトコルスタック(TINET)
 - ETHER-NETデバイスドライバ
 - ファイルシステム(FatFs)
 - MCIデバイスドライバ
 - SDカードインターフェイスドライバ
 - 日付時間の管理(RTC)

仮想端末システムのAPI

- 仮想端末、TELNETの機構はUNIXで構築されました
- UNIX用のAPI(POSIX)を想定したAPIを構築した方がアプリケーションの開発を行いやすい



POSIX準拠のインターフェイスをAPIとして構築する

プラットフォーム構築

1. プラットフォーム概要
2. 仮想端末機器
3. プラットフォーム構築の手順



開発手順

- 開発環境の構築
 - Cygwin上にGCC-ARMコンパイラを動作させる
- RTOSのポーティング
 - TOPPERS/JSP, TOPPERS/ASP
 - このセミナーでは実装済みとして検証のみ
- デバイスドライバの作成とテスト
 - RTC、MCI、SDカードI/F、ETHERNET
- ミドルウェアとの結合、ポーティング
 - ストレージマネージャ、FatFs、TINET
- POSIX-APIの作成
- TELNETサーバの作成
- 仮想端末アプリケーションの作成



開発環境の構築

- 次章で説明を行います
 - Cygwin-GNUの開発環境
 - デバッグツールを使ってFLASH-ROM書き込みを行う
 - ログ出力とタスクモニタを用いてデバッグ
- ROMモニタによるデバッグ
 - 本セミナーでは、講師のデモを行います
- タスクモニターの設定
 - 基礎講座で使用しているタスクモニタを使用します

RTOSのポーティング

- TOPPERS/JSP、TOPPERS/ASPをターゲットボードにポーティング
 - LPC2388はARM7をベースにしています
 - JSPではconfig/armv4をベースに修正
 - ASPではtarget/btc090_gccをベースに修正
- ARMは割込み機構がベンダ依存であるため注意
- 基本的な開発項目
 - 割込み機構の作成
 - システムタイマの作成
 - シリアルデバイスの作成
- 実行後SAMPLE1を用いて機能検証

デバイスドライバの作成とテスト

- 以下のデバイスドライバの作成とテストを行います
 - RTC:タスクモニタを使ってハードウェアの確認
 - MCI/SDカードインターフェイス、FAT
 - MCI/SDカードインターフェイス、FATについての講義
 - FAT用ドライバの作成とテスト:実習
 - TINET
 - ETHERデバイスドライバの説明
 - ETHERデバイスドライバの作成とTELNETアプリを使ったテスト

POSIX-APIとTELNETサーバの作成

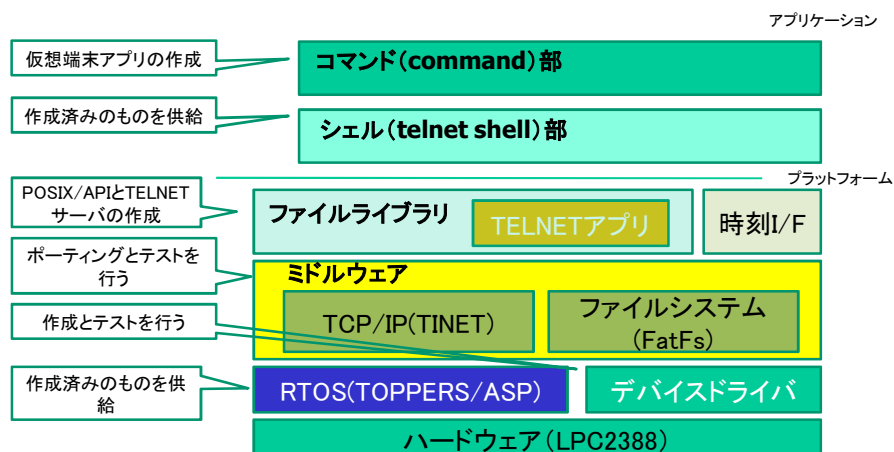
- ファイルライブラリとファイル用POSIX-API
 - 教材として用意したファイルライブラリとファイル系のPOSIX-API環境を使用する
 - ミドルウェアの形態となっているのでインストール手順について説明を行う
- TELNETサーバ
 - 教材として用意したTELNETサーバをTINETのテストプログラムとして使用します

仮想端末アプリケーションの作成

- プラットフォーム上に仮想端末アプリケーションを作成します
 - アプリケーション用のワークスペースを作成する
 - ワンリンクモデルでの実行形式の作成
 - プラットフォーム部とアプリケーション部の結合
 - 例題アプリを参考に仮想端末コマンドを追加する

このセミナーでの学習内容

- 以下に仮想端末システムのブロック図と学習内容を示します



開発環境のセットアップ

1. [GNU環境の構築](#)
2. ツール環境の構築
3. RTOSのセットアップ
4. SAMPLE1の実行
5. ハードウェアの確認



組み込みソフトウェア開発に使われるプログラミング言語

- C言語
 - ハードウェアを直接操作するプログラミングが可能であるため、組み込みソフトウェア開発では、最も使われている
- アセンブリ言語
 - DSPなどの特殊なプロセッサで使われる場面が多い
 - コンパイラが扱えない特殊命令を直接記述して、性能を出す
- C++言語
 - 利用は広がっているが、まだ限定的
 - オーバーヘッドが大きい
 - どのような実行コードになるか見えにくい



本教材のプログラムを開発するための開発環境

- 教材ボードはARM7プロセッサが使用されている
 - ARM用コンパイラとアセンブラが必要
 - GNUのARM用コンパイラを使用する
- GNUの実行にはUNIX環境が必要
 - Windows上で動作するUNIX環境: Cygwinを使用する
- 教材ボード用の開発環境を構築するため必要なこと
 - CygwinとARM-GNUコンパイラのインストール
 - Cygwin(UNIX)の開発手法について学ぶ



DOS窓のようなCygwinのコマンド環境

ここでb-shell(bash)が実行できる



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

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

2013/09/16

TOPPERSプロジェクト認定

39



C言語のツールチェーン

C言語コードを実行コードに変換するためのツール群

コンパイルドライバ

- 実行コードを生成するまでの一連の処理を実行
 - プリプロセッサ, コンパイラ, アセンブラ, リンカを呼び出す

プリプロセッサ

- #includeやマクロを展開

コンパイラ

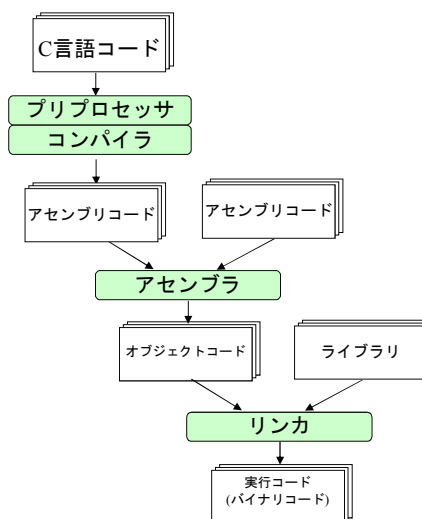
- プリプロセスされたC言語コードをアセンブリコードへ変換

アセンブラ

- アセンブリコードをオブジェクトコード(機械語プログラム)に変換

リンカ

- 複数のオブジェクトコードとライブラリをリンクし実行コードを生成する



2013/09/16

TOPPERSプロジェクト認定

40



Cygwinをインストールしよう1

- CygwinのサイトからCygwin 1.5.x以降のバージョンをダウンロードし、Windows-XPまたはWindows-7にインストールします(makeのバージョンは3.81以降をお勧め)

Cygwinサイト → <http://www.cygwin.com/>

注意 トラブル回避のために、すでにCygwinをインストール済みの方は、バージョンの確認をお願いします

Bash上で、`uname -a`(return)がCygwinのバージョン問い合わせ、
`make -ver`(return)がmakeのバージョン表示です



2013/09/16

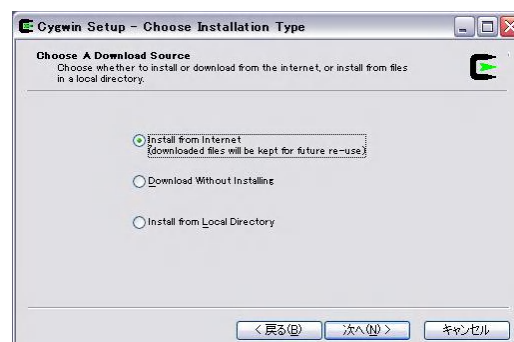
TOPPERSプロジェクト認定

41



Cygwinをインストールしよう2

- setup.exeを起動して、インターネット経由のダウンロードインストールまたはダウンロード後インストールのどちらかを選択できます



2013/09/16

TOPPERSプロジェクト認定

42



Cygwinをインストールしよう3

- マルチバイト文字およびスペースを含まないディレクトリにインストールします(例: C:\cygwin)



2013/09/16

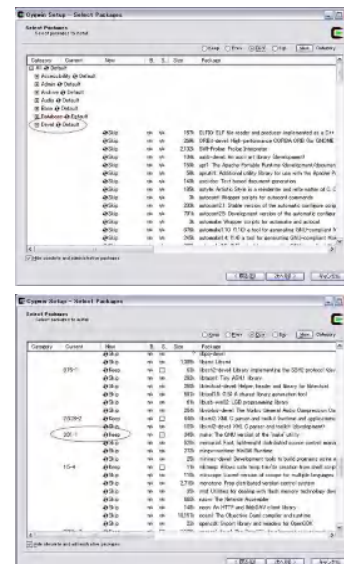
TOPPERSプロジェクト認定

43



Cygwinをインストールしよう4

- makeのバージョンは3.81-1を選択します
- makeコマンドはmakeファイルの記述に従って、ビルドの手順を指定するコマンドです



2013/09/16

TOPPERSプロジェクト認定

44



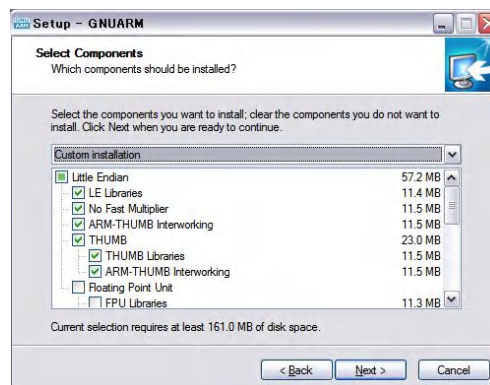
ARM-GCCをインストールしよう1

- GCCのインストールは、GCCのソースコードをダウンロードして、インストールが可能ですが、手順が複雑なため、ここではバイナリインストールを行います
- 指定の(bu-2.16.1 gcc-4.0.2-c-c++ nl-1.14.0 gi-6.4.exe)をダウンロードしてください



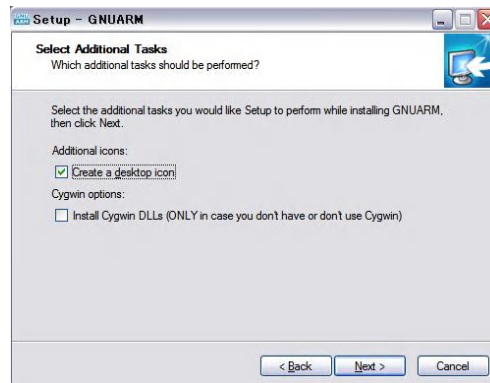
ARM-GCCをインストールしよう2

- インストールディレクトリはUNIX風に
C:\cygwin\usr\local
にインストールします
- 教材ボードで使われているARM7(NXP AM7TDMI)にはlittle Endian, Floating Point Unitなし, THUMBコードの対応を行うために右記のダイアログの設定としてください



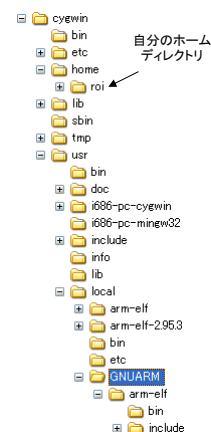
ARM-GCCをインストールしよう3

- Cygwinはインストール済みのため、“install Cygwin DLLs...”は選択しないでください
- インストール終了時に、GNU ARMインストールディレクトリに対するWindows環境変数(パス)登録を確認されますが、パスを登録する必要はありません



ARM-GCCをインストールしよう4

- 自分のHOMEディレクトリの.bash_profileの内容を修正します
- .bash_profile中のexportコマンドにインストールしたARMGCCのコマンドパス(/usr/local/GNUARM/bin)を追加します
- Cygwinの起動後コンパイラ等のコマンドが使用できるようになります
 - arm-elf-gcc コンパイラ
 - arm-elf-ld リンカ
 - arm-elf-ar ライブラリアン



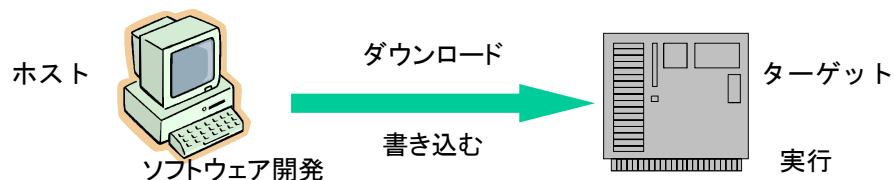
```
export PATH=${PATH}:/usr/local/GNUARM/bin
```

開発環境のセットアップ

1. GNU環境の構築
2. [ツール環境の構築](#)
3. RTOSのセットアップ
4. SAMPLE1の実行
5. ハードウェアの確認

書き込みツールのインストール

- LPC2388にプログラムを書き込むためにパソコンに書き込みツールをインストールする
 - Flash Magic
- Flash MagicはLPC2388内のFlashROMにプログラムを書き込むためのツールです



Flash Magicのインストール1

- FlashMagic.exeを実行する
- 「Next」を押すと、ライセンス画面が表示される。
- 「I accept the agreement」を選択して、「Next」ボタンを押す



2013/09/16

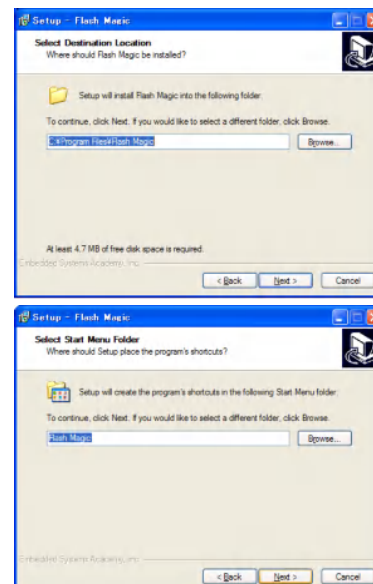
TOPPERSプロジェクト認定

51



Flash Magicのインストール2

- インストール先を変更せず、「Next」を押す
- メニュー・フォルダを変更せず、「Next」を押す



2013/09/16

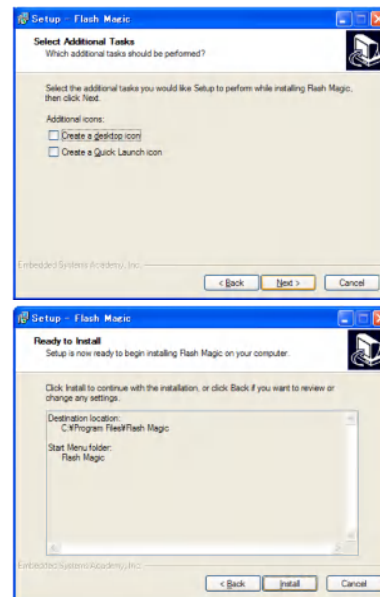
TOPPERSプロジェクト認定

52



Flash Magicのインストール3

- ICONを追加せず、「Next」を押す
- インストール確認画面で「Install」を押す



2013/09/16

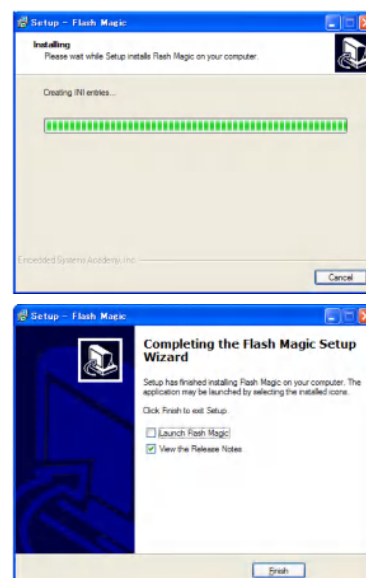
TOPPERSプロジェクト認定

53



Flash Magicのインストール4

- インストール終了画面で「Finish」を押すとインストールが終了します



2013/09/16

TOPPERSプロジェクト認定

54



開発環境のセットアップ

1. GNU環境の構築
2. ツール環境の構築
3. [RTOSのセットアップ](#)
4. SAMPLE1の実行
5. ハードウェアの確認

2013/09/16

TOPPERSプロジェクト認定

55



TOPPERS/ASPのダウンロード

- TOPPERSプロジェクトのWebからasp-1.7.0の個別パッケージをダウンロードします
- 同様にARMアーキテクチャ・GCC依存部パッケージ asp_arch_arm_gcc-1.7.0をダウンロードします

TOPPERS/ASPカーネル 個別パッケージのダウンロード

TOPPERS/ASPカーネルの最新リリースの個別パッケージを配布しています。各ターゲットカーネル非依存パッケージ、依存部パッケージ（アーキテクチャ依存部、ターゲットビルドライブラリ依存部(PDCL)）パッケージを個別にダウンロードできます。

ターゲットごとに必要なソースコードを一つにまとめた簡易パッケージは、[こちら](#)からります。

一般公開以前のバージョン(Release 1.3.0以前)に付いた個別パッケージは、登録品としての扱いですが、[こちら](#)からダウンロードできます。

ターゲット非依存部

TOPPERS/ASPカーネル ターゲット非依存部パッケージ (担当: 名古屋大学)		
パッケージ	リリース日	
asp-1.3.0.tar.gz	2012-12-26	
asp-1.7.0.tar.gz	2011-05-09	

ARMアーキテクチャ・GCC依存部パッケージ (担当: 名古屋大学)			
パッケージ	ディレクトリ	対応する非依存部のバージョン	リリース日
asp_arch_arm_gcc-1.7.0.tar.gz	arch/arm_gcc/common target/at91skyeye_gcc	1.7.*	2011-05-19
asp_arch_arm_gcc-1.4.0.tar.gz	arch/arm_gcc/common arch/arm_gcc/at91sam7s target/at91skyeye_gcc target/bic090_gcc	1.4.*	2009-05-18
asp_arch_arm_gcc-1.3.2.tar.gz	arch/arm_gcc/common arch/arm_gcc/at91sam7s target/at91skyeye_gcc target/bic090_gcc	1.3.*	2008-08-27

2013/09/16

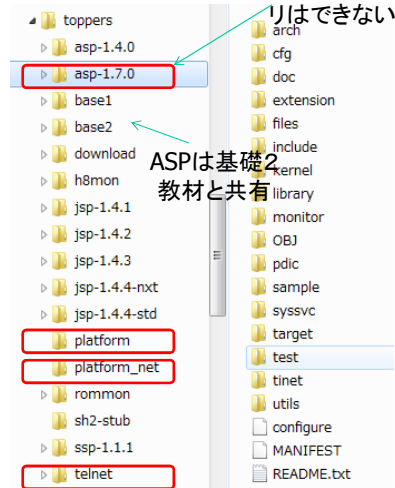
TOPPERSプロジェクト認定

56



TOPPERS/ASPを構築するディレクトリの位置

- TOPPERS/ASPの開発ディレクトリは基礎2、基礎3で共通に使用します
- 基礎2でASPをポーティング済の場合、解凍の作業を行う必要はありません
- 基礎1,2のディレクトリと並列の位置にasp-1.7.0のディレクトリを作成してください
- base3から以下のファイルをコピー解凍してください
 - platform/platform_net/telnet



TOPPERS/ASPのセットアップ

- 開発用のディレクトリにasp-1.7.0.tar.gz、asp_arch_gcc-1.7.0.tar.gzと、添付の開発環境にasp-1.7.0-030913.tar.gzを置きます
- asp-1.7.0.tar.gzとasp_arch_arm_gcc-1.7.0.tar.gzの解凍後、解凍されたaspディレクトリ名をasp-1.7.0に変更します
- その後、asp-1.7.0-030913.tar.gzを解凍します

```
$ ls
asp_arch_arm_gcc-1.7.0.tar.gz asp-1.7.0.tar.gz asp-1.7.0-070812.tar.gz
$ tar zxvf asp-1.7.0.tar.gz
$ tar zxvf asp_arch_arm_gcc-1.7.0.tar.gz
$ mv asp asp-1.7.0
$ tar zxvf asp-1.7.0-030912.tar.gz
```

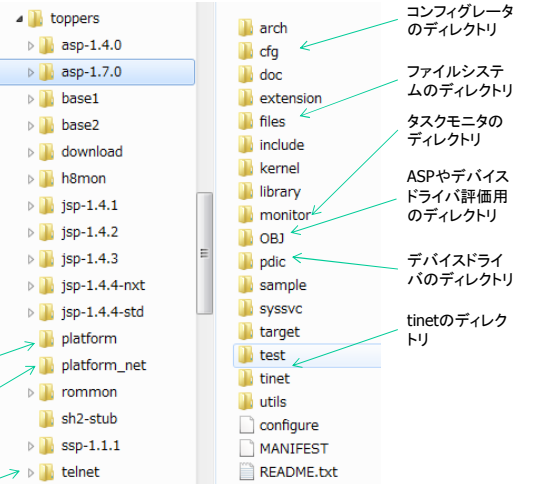
TOPPERS/ASPワークスペースの説明

- asp-1.7.0にプラットフォームに必要なソフトウェア部品やテスト環境が含まれます
- アプリはaspやjspを選択できるように並列したディレクトリに配置します

作成したplatformのディレクトリ

改造中のplatformのディレクトリ

作成したアプリケーションのディレクトリ



注: このディレクトリ図はいろいろがアプリを併用しているため解凍したものとの多少の差異があります

2013/09/16

TOPPERSプロジェクト認定

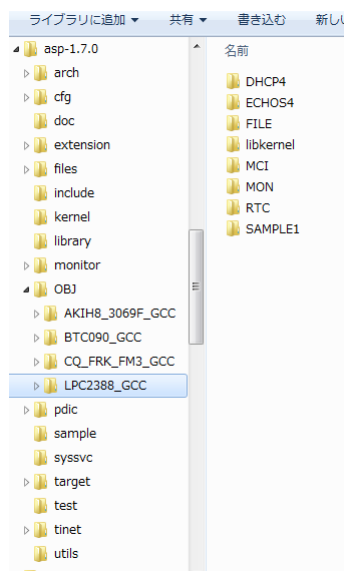
59



ASP評価用ディレクトリの解説

- デバイスドライバ評価用の構成は以下の通りです
- libkernelは、この後作成します

ディレクトリ	内容
DHCP4	DHCPアプリ評価プログラム
ECHOS4	エコーサーバー評価プログラム
FILE	ファイルシステム評価プログラム
libkernel	カーネルライブラリ
MCI	MCIデバイスドライバ評価プログラム
MON	sample1+タスクモニタ
RTC	RTCデバイスドライバ評価プログラム
SAMPLE1	ASP評価プログラム



2013/09/16

TOPPERSプロジェクト認定

60



ソースコードについての注意

- ASPのカーネルソース、教材のプログラムの漢字コードはEUC-JP、改行コードはUNIXと互換のLFとなっています
- Cygwinの環境で参照、修正を行う場合は、UNIX互換漢字コード、改行コード対応のエディタを使用してください



漢字コード、改行コードを自動変換する
エディタ: TeraPad

コンフィギュレータの構築

- ASPは静的APIをサポートしているため、静的APIをプログラミング言語に変換するコンフィギュレータが用意されています
- Cygwinでは、cfgの下でMakefileを用いてコンフィギュレータのビルドを行う必要があります
- asp-1.7.0上にRelease 1.8.0をダウンロードし、解凍するとcfgディレクトリが作成される

TOPPERS新世代カーネル用コンフィギュレータ

TOPPERS新世代カーネル用コンフィギュレータの最新リリースを配布しています。コンフィギュレータ自体についてはこちらを、構築方法等についてはアーカイブに含まれる README.txtをご覧ください。

リリース名	タイプ	サイズ	リリース日
コンフィギュレータ Release 1.9.1	tar.gz(EUC-LF)	139KB	2013-04-05
コンフィギュレータ Release 1.9.1(Windows用バイナリ)	zip	10.8MB	2013-04-05
Release 1.9.0Cの通り			

リリース名	タイプ	サイズ	リリース日
コンフィギュレータ Release 1.8.0	tar.gz(EUC-LF)	91KB	2012-05-09
コンフィギュレータ Release 1.8.0(Windows用バイナリ)	zip	1.0MB	2012-05-09
コンフィギュレータ Release 1.8.0(Cygwin用バイナリ)	tar.gz	450KB	2012-05-09

コンフィギュレータのビルド

- cfg-1.8.0.tar.gzを解凍し、解凍したディレクトリに移動してビルドを行う
- 解凍、ビルド後コンフィギュレータcfg.exeがcfg/cfgのしたにビルドされる
- Cygwinが古い場合、ビルドでエラーが発生する場合があります。この場合、Windows用バイナリをダウンロードして、cfg/cfg/cfg.exeに置けばよい

```
$ cd asp-1.7.0
$ tar zxvf cfg-1.8.0.tar.gz
$ cd cfg
$ ./configure
$ make
```

カーネルライブラリのビルド

- OBJ/LPC2388_GCC環境用にカーネルライブラリを作成する
- これにより、プログラムの開発時、毎回カーネルライブラリをコンパイルしなくてよくなる
- configureコマンド実行後は、4つのファイルが作成される
 - Makefile/sample1.cfg/sample1.h/sample1.c

```
$ cd ../OBJ/LPC2388_GCC
$ mkdir libkernel
$ cd libkernel
$ ../../configure -T lpc2388_gcc -f
$ make depend
$ make libkernel.a
```

カーネルライブラリの指定

- make kernel.aによりlibkernel上のカーネルライブラリ(kernel.a)が作成される
- 各アプリからカーネルライブラリを指定するためにMakefileのLIB_KERNELのカーネルライブラリのディレクトリを指定する必要がある
- configureでsample1を生成した場合は、KERNEL_LIBの設定は空になっている

```
78 #
79 # カーネルライブラリ(libkernel.a)のディレクトリ名
80 # (カーネルライブラリもmake対象にする時は、空に定義する)
81 #
82 KERNEL_LIB = ../libkernel
```

SAMPLE1のビルド

- ワークスペースで、SAMPLE1プログラムのビルドを行う
- LPC2388用のTOPPERS/ASPはDBGENV環境スイッチの指定で実行環境を変えられます
 - DBGENV=ROMまたはなしでROM実行版
 - DBGENV=RAMでRAM実行版
- 今回はRAM実行版を作成します

```
$ cd ../SAMPLE1
$ make depend DBGENV=RAM
$ make DBGENV=RAM
```

開発環境のセットアップ

1. GNU環境の構築
2. ツール環境の構築
3. RTOSのセットアップ
4. [SAMPLE1の実行](#)
5. ハードウェアの確認

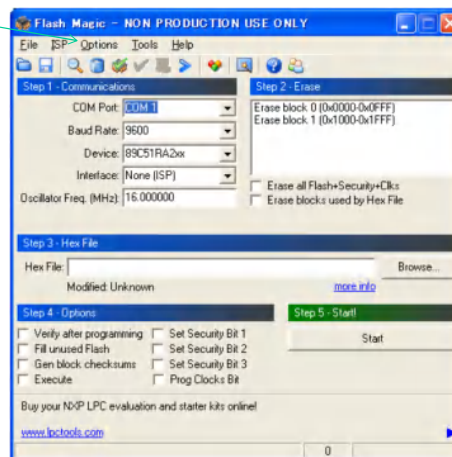
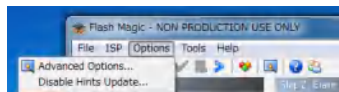
- 基礎1, 2で使ったROMモニタを用いてRAM実行でSAMPLE1を実行する
- 次の章のRTCもRAM実行で確認を行うが、FILEシステム以降は、メモリの関係からROM実行版で検証を行う

ROMモニタをボードに書き込む

- Windowsのスタートメニューから「Flash Magic」を起動する

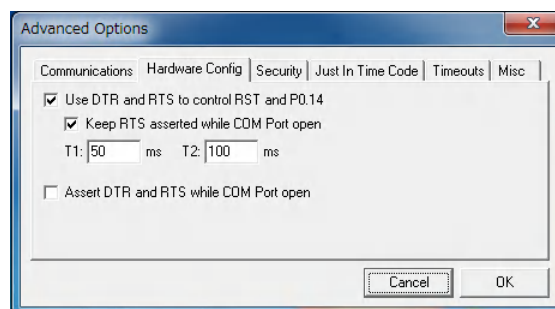
「Option」メニュー

最初に起動した時は、
左の画面が表示されます
「Option」⇒「Advanced
Options」を選択してくだ
さい



Flash Magicの初期設定

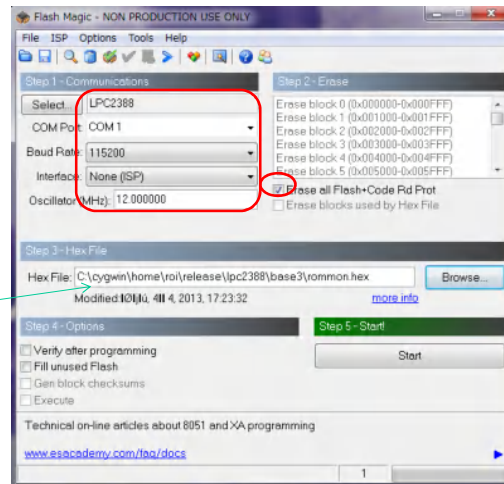
- Advanced Optionsメニュー中の「Hardware Config」メニューを表示して画面の通りの設定を行う



rommon.hexを選択する

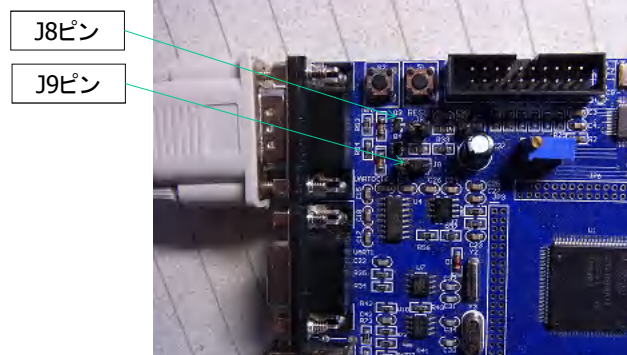
- メイン画面のSTEP1/2/3を設定する
- STEP3では
../base3/rommon.hexを
選択する

rommon.hex
ファイルの選
択

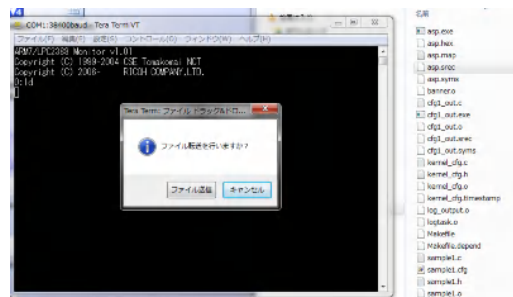


Flash Magicでプログラムを書き込む

- LPC2388ボードのUART0ポートにRS232Cケーブルを接続し、J8ピンを接続、J9ピンの左の2つのピンを接続する
- ボードの電源を入れてメニューの「Start」ボタンを押すと書き込みが始まります



- 書き込み後、電源オフし、J8、J9ピンを元に戻す
- Tera Termを立ち上げ、電源をオンするとROMモニタのプロンプトが表示される
- asp.srecをTera Termにドラックすると転送開始のダイアログがでるので「ファイル転送」を押す



2013/09/16

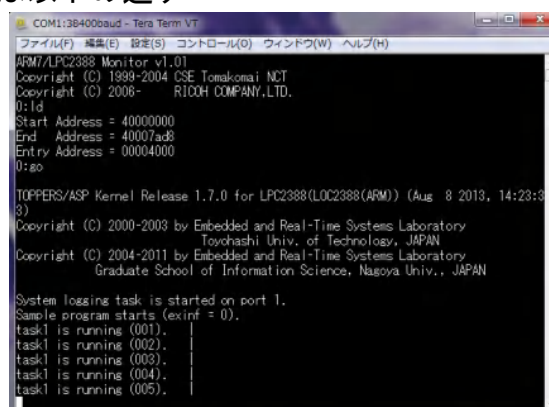
TOPPERSプロジェクト認定

73



SAMPLE1を実行する

- ダウンロード終了後、goコマンドで実行するとSAMPLE1が実行する
- TeraTermの設定は以下の通り
 - 38400baud
 - 8bit
 - Non parity
 - Stop bit1
 - None flow



2013/09/16

TOPPERSプロジェクト認定

74



開発環境のセットアップ

1. GNU環境の構築
2. ツール環境の構築
3. RTOSのセットアップ
4. SAMPLE1の実行
5. ハードウェアの確認

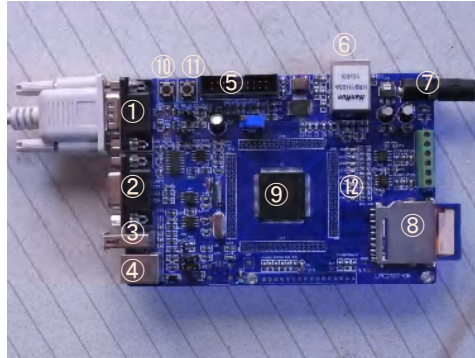
LPC2388ボードの解説

- マイコンボード:LPC2388
 - (株)日昇テクノロジー製
 - <http://www.csun.co.jp>
- CPUコアとしてARM7を使用
- 電源は外部より5Vを供給
- オンチップ中にメモリをもつ
 - 512KB-Flash ROM
 - 64KB-SRAM

AHB Peripherals	0xF0000000
APB Peripherals	0xE0000000
	0x81010000
64KB External Memory Bank1	0x81000000
	0x80010000
64KB External Memory Bank0	0x80000000
BOOT FLASH ROM	
16KB EtherNet RAM	0x7FE00000
16KB USB RAM	0x7FD00000
	0x40010000
64KB SRAM	0x40000000
	0x00800000
512KB Flash ROM	0x00000000

LPC2388ボードの説明

- 1.UART0コネクタ
- 2.UART1コネクタ
- 3.USBホスト
- 4.USBデバイス
- 5.JTAG
- 6.10/100ETHER-LAN
- 7.5V電源コネクタ
- 8.SDソケット
- 9.LPC2388
- 10.USERスイッチ
- 11.リセットスイッチ
- 12.LED



2013/09/16

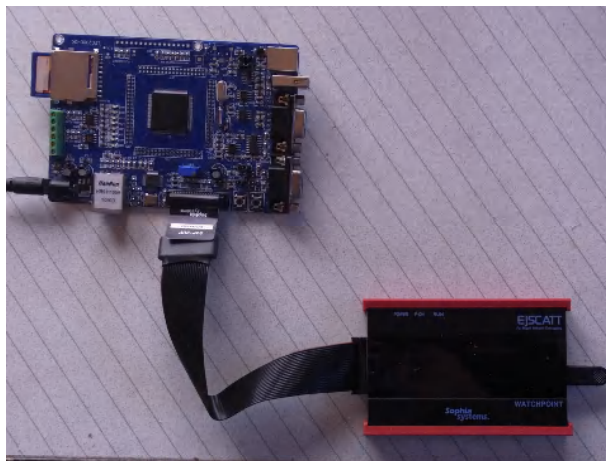
TOPPERSプロジェクト認定

77



JTAG-ICEの使用

- デバイスドライバのデバッグにはJ-TAG-ICEが有用です



2013/09/16

TOPPERSプロジェクト認定

78

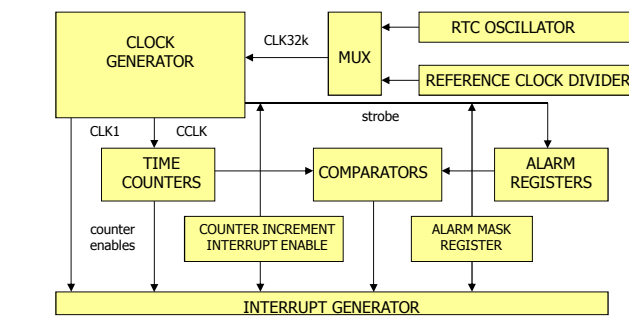


RTCデバドラの説明

1. RTCハードウェア
2. インターフェイス関数
3. タスクモニタを使ったデバッグ

ハードウェア仕様

- RTC (Real Time Clock) 回路の説明はLPC23XX_umの第26章: LPC23XX Real Time Clock(RTC) and battery RAMに記載
- ボードからの3.3Vのバッテリーバックアップ電源がないため電源OFF時の日時のバックアップはできない



RTC制御レジスタ

- RTCはカレンダー機能と割込みによる種々のアラーム機能をもつ
- LPC23xxのRTCレジスタは以下のとおり
- 現在日時を設定すると、電源供給中は日時が更新される
- 日時の更新時、アラーム日時の設定と一致時割込みが発生する

name	address	access	width	discription
ILR	0xE0024000	R/W	2	RTC用割込みステータスレジスタ
CCR	0xE0024008	R/W	4	クロック設定レジスタ
CIIR	0xE002400C	R/W	8	カウンタ加算割込みレジスタ
AMR	0xE0024010	R/W	8	アラームマスクレジスタ
SEC	0xE0024020	R/W	6	秒カウンタ
MIN	0xE0024024	R/W	6	分
HOUR	0xE0024028	R/W	5	時
DOM	0xE002402C	R/W	5	月中の日
DOW	0xE0024030	R/W	3	週中の日
DOY	0xE0024034	R/W	9	年中の日
MONTH	0xE0024038	R/W	4	月
YEAR	0xE002403C	R/W	12	年
CISS	0xE0024040	R/W	8	Counter Interrupt select mask for Sub-Second interrupt
ALSEC	0xE0024060	R/W	6	アラーム設定の秒
ALMIN	0xE0024064	R/W	6	アラーム設定の分
ALHOUR	0xE0024068	R/W	5	アラーム設定の時間
ALDOM	0xE002406C	R/W	5	アラーム設定の月中の日
ALDOW	0xE0024070	R/W	3	アラーム設定の週中の日
ALDOY	0xE0024074	R/W	9	アラーム設定の年中の日
ALMON	0xE0024078	R/W	4	アラーム設定の月
ALYEAR	0xE002407C	R/W	12	アラーム設定の年
PREINT	0xE0024080	R/W	13	割込み用プレスケール設定
PREFRAC	0xE0024084	R/W	15	フラクション用プレスケール設定

参照、設定しないレジスタは削除しています

RTCデバドラの説明

1. RTCハードウェア
2. インターフェイス関数
3. タスクモニタを使ったデバッグ

RTCデバイスドライバのインターフェイス

- RTCデバイスドライバはPDICに準じた記載で記述
- デバイスドライバとして、以下の8つの関数を用意する
- 時間の受け渡しのためにtm2という構造体を用意する

書式	機能	返り値
void rtc_isr0(void);	RTC割り込みハンドラ	なし
void rtc_init(VP_INT exinf);	RTC初期化関数	なし
ER rtc_start(VP_INT func);	RTCスタート設定	E_OK
ER rtc_terminate(void);	RTC終了設定	E_OK
ER set_time(struct tm2 *pt);	日時設定	E_OK/E_PAR
ER rtc_set_alarm(struct tm2 *pt);	アラート設定	E_OK/E_PAR
ER rtc_get_time(struct tm2 *pt);	日時取得	E_OK/E_PAR
ER rtc_set_event(struct tm2 *pt);	インクリメント割り込み設定	E_OK/E_PAR

RTCデバイスドライバ解説:tm2構造体

- RTCDライバとデータ交換する構造体tm2を定義
- 本ドライバはSIL仕様を使用しているため、対応のレジスタとSILの項目定義を表に列記

```
#define INHNO_RTC    INTNO_RTC
#define BASE_YEAR    1970
#define IMSEC        (1<<0)    /* Interrupt for Seconds */
:
#define _MACRO_ONLY
/*
 * 日時設定用の構造体を定義
 */
struct tm2 {
    int    tm_sec;        /* 秒 */
    int    tm_min;        /* 分 */
    int    tm_hour;       /* 時 */
    int    tm_mday;       /* 月中の日 */
    int    tm_mon;        /* 月 */
    int    tm_year;       /* 年 */
    int    tm_wday;       /* 曜日 */
    int    tm_yday;       /* 年中の日 */
    int    tm_isdst;
};
```

ハンドラ番号を割り込み番号で定義

年の基準となる年を定義

構造体項目とレジスタ、SIL定義の表

項目	レジスタ	SILのレジスタ定義
tm_sec	SEC	TOFF_RTC_SEC
tm_min	MIN	TOFF_RTC_MIN
tm_hour	HOUR	TOFF_RTC_HOUR
tm_mday	DOM	TOFF_RTC_DOM
tm_mon	MONTH	TOFF_RTC_MONTH
tm_year	YEAR	TOFF_RTC_YEAR
tm_wday	DOW	TOFF_RTC_DOW
tm_yday	DOY	TOFF_RTC_DOY

RTCDライバの実装関数の説明

- デバイスドライバ名の`lpc23xx_`の名称はリネームインクルードファイル(`board_config.h`)によって削除される
- これにより、ユーザーは標準的な名称を使用することができる

board_config.hでの名称変更

書式	インターフェイス上の名称
ER lpc23xx_rtc_start(VP_INT func);	rtc_start
ER lpc23xx_rtc_terminate(void);	rtc_terminate
ER lpc23xx_rtc_set_time(struct tm2 *pt);	rtc_set_time
ER lpc23xx_rtc_set_alarm(struct tm2 *pt);	rtc_set_alarm
ER lpc23xx_rtc_get_time(struct tm2 *pt);	rtc_get_time
ER lpc23xx_rtc_set_event(struct tm2 *pt);	rtc_set_event

```
/*
 * RTCに関する関数定義
 */
#define rtc_start      lpc23xx_rtc_start
#define rtc_terminate  lpc23xx_rtc_terminate
#define rtc_set_time   lpc23xx_rtc_set_time
#define rtc_set_alarm  lpc23xx_rtc_set_alarm
#define rtc_get_time   lpc23xx_rtc_get_time
#define rtc_set_event  lpc23xx_rtc_set_event
```

RTCDライバの実装

- RTCDライバの保管場所
 - RTOSディレクトリ/pdic
- 実装ファイル
 - `rtc.cfg`: RTCデバイスドライバが使用するRTOSオブジェクトを定義するコンフィギュレーションファイル
 - `lpc23xx_rtc.h`: RTCデバイスドライバインクルードファイル
 - `lpc23xx_rtc.c`: RTCデバイスドライバソースファイル
- RTCレジスタ等の定義は以下のインクルード設定にある
 - RTOSディレクトリ/arch/arm_gcc/lpc23xx/lpc23xx.h
- 実作業では設計ドキュメント(仕様書)を用意する必要があるが、本学習では記載しない

RTCデバイスドライバ解説: rtc.cfg

- RTCで使用するRTOSオブジェクトの定義を行う
 - 初期化関数定義: rtc_init
 - 割り込みハンドラ定義: rtc_isr0
 - 排他制御用にセマフォを定義

```

/*
 * RTCデバイスドライバのコンフィギュレーションファイル
 */
#include "lpc23xx_rtc.h"

ATT_INI({ TA_HLNG, 0, rtc_init });
CRE_SEM(RTCSEM, { TA_TPRI, 1, 1 });
DEF_INH(INHNO_RTC, { TA_HLNG, rtc_isr0 });
CFG_INT(INTNO_RTC, { TA_ENAINT|INTATR_RTC, INTPRI_RTC });

```

初期化ルーチンの追加静的APIにてRTC用初期化関数を設定

排他制御用のセマフォをRTCSEMで設定

割り込みハンドラの定義静的APIにて、アラーム用の割り込みハンドラを定義

2013/09/16

TOPPERSプロジェクト認定

87



RTCデバイスドライバの説明: lpc23xx_rtc.c: 2

- 割り込みハンドラでは、RTCの割り込みが発生するとステータスを読み取り、alarm_funcが設定されていれば、ステートを引き数にてコールバックを行う

```

static void (*alarm_func)(int event);
/*
 * RTC用割り込みハンドラ
 */
void rtc_isr0(void)
{
    uint32_t status = sil_rew_mem((VP)(TADR_RTC_BASE+TOFF_RTC_ILR));
    i_begin_int(INTNO_RTC);
    sil_wrw_mem((VP)(TADR_RTC_BASE+TOFF_RTC_ILR), status);
    if(alarm_func != NULL)
        alarm_func(status);
    i_end_int(INTNO_RTC);
}

/*
 * RTC用初期化
 */
void rtc_init(VP_INT exinf)
{
    alarm_func = NULL;
    sil_wrw_mem((VP)(TADR_RTC_BASE+TOFF_RTC_AMR), 0x000000FF);
    sil_wrw_mem((VP)(TADR_RTC_BASE+TOFF_RTC_CIR), 0x00000000);
    :
}

```

割り込み要因を読み出す

割り込み要因を書き込み割り込みクリア

コールバック関数を読み出す

コールバック関数を初期化

ハードウェアの初期化

2013/09/16

TOPPERSプロジェクト認定

88



RTCデバイスドライバの説明:lpc23xx_rtc.c:3

- RTCを起動/終了するための2つの関数を以下に示す

```

/*
 * RTC起動関数
 * 引数として割込み発生時のコールバック関数を渡す
 * 不要な場合は、NULLを渡せばよい
 */
ER lpc23xx_rtc_start(VP_INT func)
{
    UW ccr = sil_rew_mem((VP)(TADR_RTC_BASE+TOFF_RTC_CCR));

    sil_wrw_mem((VP)(TADR_RTC_BASE+TOFF_RTC_CCR), ccr | CCR_CLKEN);
    sil_wrw_mem((VP)(TADR_RTC_BASE+TOFF_RTC_ILR), ILR_RTCCIF|ILR_RTCALF);
    alarm_func = func;
    return E_OK;
}

/*
 * RTC停止関数
 */
void lpc23xx_rtc_terminate(void);
{
    alarm_func = NULL;
    sil_wrw_mem((VP)(TADR_RTC_BASE+TOFF_RTC_CCR), sil_rew_mem((VP)(TADR_RTC_BASE+TOFF_RTC_CCR)) & ~CCR_CLKEN);
    return E_OK;
}

```

RTC用のクロックをイネーブルに

割込みを許可に

割込み用のコールバック関数の設定

2013/09/16

TOPPERSプロジェクト認定

89



RTCデバイスドライバの説明:rtc_set_time

- RTCへの日時設定関数は、POSIXの日時設定時に使用するtm構造体を使用する
- tm構造体の日時をRTCレジスタに設定を行う
- 年はtm構造体の指定が1970を0としているため
BASE_YEARを足す

```

/*
 * RTCの時刻設定関数
 * 時刻の設定はPOSIXのtm構造体を使用する
 * POSIXのインクルードファイルがない場合を考慮し、同一項目のtm2をドライバとして定義する
 */
ER lpc23xx_rtc_set_time(struct tm2 *pt)
{
    if(pt == NULL)
        return E_PAR;
    _syscall(wai_sem(RTCSEM));
    sil_wrw_mem((VP)(TADR_RTC_BASE+TOFF_RTC_SEC), pt->tm_sec);
    sil_wrw_mem((VP)(TADR_RTC_BASE+TOFF_RTC_MIN), pt->tm_min);
    sil_wrw_mem((VP)(TADR_RTC_BASE+TOFF_RTC_HOUR), pt->tm_hour);
    sil_wrw_mem((VP)(TADR_RTC_BASE+TOFF_RTC_DOM), pt->tm_mday);
    sil_wrw_mem((VP)(TADR_RTC_BASE+TOFF_RTC_DOW), pt->tm_wday);
    sil_wrw_mem((VP)(TADR_RTC_BASE+TOFF_RTC_DOY), pt->tm_yday);
    sil_wrw_mem((VP)(TADR_RTC_BASE+TOFF_RTC_MONTH), pt->tm_mon);
    sil_wrw_mem((VP)(TADR_RTC_BASE+TOFF_RTC_YEAR), pt->tm_year+BASE_YEAR);
    _syscall(sig_sem(RTCSEM));
    return E_OK;
}

```

実装上はPOSIXのtm構造体と区別するためtm2構造体を作成した

2013/09/16

TOPPERSプロジェクト認定

90



RTCデバイスドライバの説明: rtc_get_time

- RTCからの日時の取り出し関数はRTCレジスタから読み出した日時をtm2構造体にセットする
- 設定も取得も2つのタスクから呼び出された場合を考慮しセマフォを使って排他制御する

```

/*
 * RTCの時刻取り出し関数
 * 時刻の設定はPOSIXのtm構造体を使用する
 * POSIXのインクルードファイルがない場合を考慮し、同一項目のtm2をドライバとして定義する
 */
ER lpc23xx_rtc_get_time(struct tm2 *pt)
{
    if(pt == NULL)
        return E_PAR;
    _syscall(wai_sem(RTCSEM));
    pt->tm_sec = sil_rew_mem((VP)(TADR_RTC_BASE+TOFF_RTC_SEC));
    pt->tm_min = sil_rew_mem((VP)(TADR_RTC_BASE+TOFF_RTC_MIN));
    pt->tm_hour = sil_rew_mem((VP)(TADR_RTC_BASE+TOFF_RTC_HOUR));
    pt->tm_mday = sil_rew_mem((VP)(TADR_RTC_BASE+TOFF_RTC_DOM));
    pt->tm_wday = sil_rew_mem((VP)(TADR_RTC_BASE+TOFF_RTC_DOW));
    pt->tm_yday = sil_rew_mem((VP)(TADR_RTC_BASE+TOFF_RTC_DOY));
    pt->tm_mon = sil_rew_mem((VP)(TADR_RTC_BASE+TOFF_RTC_MONTH));
    pt->tm_year = sil_rew_mem((VP)(TADR_RTC_BASE+TOFF_RTC_YEAR)-BASE_YEAR);
    _syscall(sig_sem(RTCSEM));
    return E_OK;
}

```

RTCデバイスドライバの説明: rtc_set_alarm

- アラームの設定はtm構造体に有効な値が設定されている場合、その値をアラームレジスタに設定し割込みを許可する
- インクリメント割込みも同様であるため解説は行わない

```

/*
 * RTCのアラーム時刻設定関数
 * 時刻の設定はPOSIXのtm構造体を使用する
 * POSIXのインクルードファイルがない場合を考慮し、同一項目のtm2をドライバとして定義する
 */
ER lpc23xx_rtc_set_alarm(struct tm2 *pt)
{
    UWP alarm_mask = 0;
    if(pt == NULL)
        return E_PAR;
    _syscall(wai_sem(RTCSEM));
    if(pt->tm_sec >= 0)
        sil_wrw_mem((VP)(TADR_RTC_BASE+TOFF_RTC_ALSEC), pt->tm_sec);
    else
        alarm_mask |= AMRSEC;
    if(pt->tm_min >= 0)
        sil_wrw_mem((VP)(TADR_RTC_BASE+TOFF_RTC_ALMIN), pt->tm_min);
    else
        alarm_mask |= AMRMIN;
    if(pt->tm_hour >= 0)
        sil_wrw_mem((VP)(TADR_RTC_BASE+TOFF_RTC_ALHOUR), pt->tm_hour);
    else
        alarm_mask |= AMHOUR;
    if(pt->tm_mday > 0)
        sil_wrw_mem((VP)(TADR_RTC_BASE+TOFF_RTC_ALDOM), pt->tm_mday);
    else
        alarm_mask |= AMDOM;
    if(pt->tm_wday >= 0)
        sil_wrw_mem((VP)(TADR_RTC_BASE+TOFF_RTC_ALDOW), pt->tm_wday);
    else
        alarm_mask |= AMDOW;
    sil_wrw_mem((VP)(TADR_RTC_BASE+TOFF_RTC_AMR), alarm_mask);
    return E_OK;
}

```

RTCデバイスドライバの説明: rtc_set_event

- tm2構造体の各項目に有効値が設定されている場合、インクリメント割り込み有効ビットをオンにする

```
/*
 * インクリメント割り込み設定項目設定
 *
 * 時刻の設定はPONIXのtm構造体を使用する
 * 比較対象外の項目はマイナス値を設定することにより、比較しないように設定。
 */
ER ipc23xx_rtc_set_event(struct tm2 *pt)
{
    UW sint = 0;
    if(pt == NULL)
        return E_PAR;
    _syscall(wai_sem(RTCSEM));
    if(pt->tm_sec >= 0)
        sint |= IMSEC;
    if(pt->tm_min >= 0)
        sint |= IMMIN;
    if(pt->tm_hour >= 0)
        sint |= IMHOUR;
    if(pt->tm_mday >= 0)
        sint |= IMDOM;
    if(pt->tm_wday >= 0)
        sint |= IMDOW;
    if(pt->tm_yday >= 0)
        sint |= IMDOY;
    if(pt->tm_mon >= 0)
        sint |= IMMON;
    if(pt->tm_year >= 0)
        sint |= IMYEAR;
    sil_wrw_mem((VP)(TADR_RTC_BASE+TOFF_RTC_CIRR), sint);
    _syscall(sig_sem(RTCSEM));
    return E_OK;
}
```

RTCデバドラの説明

1. RTCハードウェア
2. インターフェイス関数
3. タスクモニタを使ったデバッグ

検証プログラムの作成

- RTCデバイスドライバを効率よくテストするために検証用のプログラムを作成する
- 実作業では検証用のドキュメント(仕様書)を作成する必要があるが、本学習では記載しない
- 以下の2つの検証プログラムを用意する
 - テストプログラム
 - タスクモニタを用いたコマンド設定プログラム
- デバイスドライバの検証にはICE等のデバッグツールを使用するケースが多い、本学習ではタスクモニタを使ったデバッグの説明を行う

テストプログラム

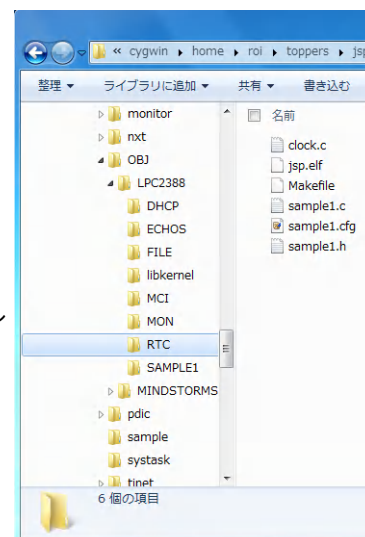
- ファイルテストプログラムの置き場所
 - RTOSディレクトリ/OBJ/LPC2388/RTC
RTCデバイスドライバをアクセスし日時の設定、時間経過ごとの日時の表示、アラーム、インクリメント割込みの設定実行を行う
- RTCデバイスドライバの置き場
 - RTOSディレクトリ/pdic/lpc23xx
- 実際の開発では、RTCのすべての機能評価を行う必要がありますが、時間の関係でアラームに関しては分の変化のみ、インクリメントについては年と分の変化のみのテストプログラムにしています
- テスト用タスクをACTIVEにすれば、何度でもテストが可能

コマンドテスト

- タスクモニタのPIPE機能(基礎2講座を参照)を用いて日時の設定コマンドを追加します
 - PIPE DATE 年:4桁 月:2桁 日:2桁
 - PIPE TIME 時:2桁 分:2桁 秒:2桁
 - PIPE CLOCK
- DATEで日にちの設定、TIMEで時間の設定、CLOCKで日時の表示を行う
- コマンドテストは有効になっていない

RTCテストプログラム解説: sample1.h

- RTCの基本テストを行う
- 学習内容
 - テスト内容の理解
 - ICEの使い方の学習
 - デバイスドライバの理解
- 構成ファイル
 - sample1.h インクルードファイル
 - sample1.c テストファイル本体
 - clock.c 日時設定
 - sample1.cfg コンフィギュレーションファイル



RTCテストプログラム概要:テスト内容

- main_taskにてRTCのテストを行います
- 2度目以降のmain_taskタスク起動は初期化処理を行わない
- 初期化処理ではデフォルト日時を2011/7/23 00:00:00に設定する
- RTCのコールバック関数を設定する(alarm_event)
- テスト部では2分後にアラームを設定する
- インクリメント割込みを分と年に設定する
- 2秒毎にRTCの日時とalarm_eventの発生を表示する、これを101回繰り返す
- テストの終了日時を表示して、タスク終了する

sample1解説:コンフィグレーションファイル

- RTC用のOSリソースをrtc.cfgを用いて定義する

```

/*
 * サンプルプログラム(1)のシステムコンフィギュレーションファイル
 */
INCLUDE("target_timer.cfg");
INCLUDE("syssvc/syslog.cfg");
INCLUDE("syssvc/banner.cfg");
INCLUDE("syssvc/serial.cfg");
INCLUDE("syssvc/logtask.cfg");
INCLUDE("pdic/lpc23xx/rtc.cfg");
INCLUDE("monitor/monitor.cfg");

Include "sample1.h"
CRE_TSK(MAIN_TASK, { TA_HLNG|TA_ACT, 0, main_task,
                     MAIN_PRIORITY, STACK_SIZE, NULL });

```

sample1解説:sample1.c:1

• RTCのコールバック関数alarm_eventの記載

```
static bool_t alarm_on;
static bool_t first_time = true;
:
static alarm_event(int event)
{
    alarm_on = true;
    iwup_tsk(MAIN_TASK);
    syslog_1(LOG_NOTICE, "alarmevent[%08x] !", event);
}
void main_task(intptr_t exinf)
{
    SYSTIM stime;
    int i;
    struct tm timedate;
    struct tm2 timedate2;

    if(first_time == FALSE){
        syscall(rtc_get_time((struct tm2 *)&timedate2); /* 時間の取得 */
        goto start_test;
    }
    :
start_test:
```

コールバック関数を定義

タスク再起動の場合、初期化処理をスキップするように記載

2013/09/16

TOPPERSプロジェクト認定

101



sample1解説:sample1.c:2

• 初期化として仮の日時を設定する

```
/* POSIX互換関数のテスト */
time = 0;
timedate.tm_sec = 0;
timedate.tm_min = 0;
timedate.tm_hour = 0;
timedate.tm_mday = 23;
timedate.tm_mon = 7;
timedate.tm_year = 2011-1970;
timedate.tm_isdst = 0;
tme = mktime(&timedate);
gmtime_r(&time, &timedate);
syslog_4(LOG_NOTICE, "time_t[%d] %04d/%02d/%02d", time, timedate.tm_year+1970, timedate.tm_mon, timedate.tm_day);
syslog_5(LOG_NOTICE, "y[%d][%d] %02d/%02d/%02d", timedate.tm_yday, timedate.tm_wday, timedate.tm_hour,...
/* RTCのテスト */
syscall(rtc_set_time((struct tm2 *)&timedate)); /* 時間の設定 */
syslog_0(LOG_NOTICE, "set current date/time");
first_time = FALSE;

sart_test:
timedate.tm_min += 2;
:
syscall(rtc_set_alarm((struct tm2 *)&timedate)); /* アラームの設定 */
syslog_0(LOG_NOTICE, "set alarm date/time");
```

仮の時間を2011/7/23 23:00:00に設定する

アラームを2分後に設定

2013/09/16

TOPPERSプロジェクト認定

102



sample1解説: sample1.c:3

- イベントの設定、2秒×101回日時表示を繰り返す

```

timedate.tm_sec = -1;
timedate.tm_min = 0;
timedate.tm_year = 0;
syscall(rtc_set_event((struct tm *)&timedate);
alarm_on = FALSE;
syscall(rtc_start(alarm_event));
syslog_0(LOG_NOTICE, "RTC start");
for(i = 0; i < 101; i++){
    syscall(rtc_get_time((struct tm2 *)&timedate));
    get_tm(&stime);
    syslog_3(LOG_NOTICE, "%04d:%02d:%02d", timedate.tm_year+1970, timedate.tm_mon, timedate.tm_mday);
    syslog_3(LOG_NOTICE, " %02d:%02d:%02d", timedate.tm_hour, timedate.tm_min, timedate.tm_sec);
    tslp_tsk(2000);
    alarm_on = FALSE;
}
syscall(rtc_get_time((struct tm2 *)&timedate)); /* 終了時刻の取り出し */
syslog_3(LOG_NOTICE, "%04d:%02d:%02d", timedate.tm_year+1970, timedate.tm_mon, timedate.tm_mday);
syslog_3(LOG_NOTICE, " %02d:%02d:%02d", timedate.tm_hour, timedate.tm_min, timedate.tm_sec);

syslog(LOG_NOTICE, "Sample program ends.");
dly_tsk(1000);
}

```

分と年にインクリメント割込みを設定

日時の表示を2秒ごとに繰り返す

デバイスドライバとタスクモニタの設定

- デバイスドライバのパス、ソースは設定済み
- タスクモニタのメイクインクルード、ARM7依存プログラムは設定済み

```

# モデルウェアの Makefile のインクルード
#
include $(SRCDIR)/monitor/Makefile.config
#
# システムサービスに関する定義
#
SYSSVC_DIR := $(SYSSVC_DIR) $(SRCDIR)/sysvc $(SRCDIR)/pdic/lpc23xx $(SRCDIR)/library
SYSSVC_ASMOBS := $(SYSSVC_ASMOBS)
SYSSVC_COBJS := $(SYSSVC_COBJS) banner.o syslog.o serial.o logtask.o ¥
log_output.o vasylog.o t_perror.o strerror.o ¥
lpc23xx_rtc.o armv4.o $(CXXRTS)
SYSSVC_FLAGS := $(SYSSVC_FLAGS)
SYSSVC_LIBS := $(SYSSVC_LIBS)
INCLUDES := $(INCLUDES) -I$(SRCDIR)/pdic/lpc23xx

```

タスクモニタは設定済み

ARM7依存プログラム

RTCデバイスドライバパス等

RTCデバイスドライバ

コマンドテストを有効にする

- Makefileを修正しPIPEコマンドの追加とclock.cを追加

```
# 共通コンパイルオプションの定義
#
COPTS := $(COPTS) -g
ifndef OMIT_WARNING_ALL
COPTS := $(COPTS) -Wall
endif
ifndef OMIT_OPTIMIZATION
COPTS := $(COPTS) -O2
endif
CDEFS := $(CDEFS) -DSUPPORT_PIPE
INCLUDES := -I. -I$(SRCDIR)/include -I$(SRCDIR)/arch -I$(SRCDIR) $(INCLUDES)
LDFLAGS := $(LDFLAGS)
LIBS := $(LIBS) -lc $(CXXLIBS)
CFLAGS = $(COPTS) $(CDEFS) $(INCLUDES)

#
# アプリケーションプログラムに関する定義
#
APPLNAME = sample1
APPLDIR =
APPL_CFG = $(APPLNAME).cfg

APPL_DIR = $(APPLDIR) $(SRCDIR)/library
APPL_ASMOBS =
ifdef USE_CXX
APPL_CXXOBS = $(APPLNAME).o
APPL_COBS = clock.o
else
APPL_COBS = $(APPLNAME).o clock.o
endif
```

SUPPORT_PIPEの定義
を追加してください

時刻管理プログラムを追加

2013/09/16

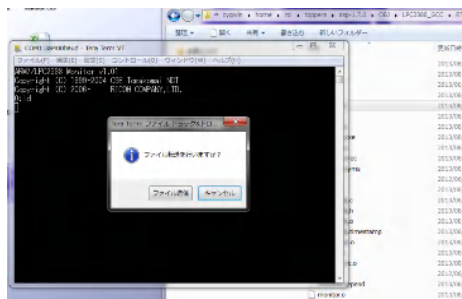
TOPPERSプロジェクト認定

105



ビルドとダウンロード

- RTCテストプログラムをビルド
 - OBJ/LP2388/RTCに移動して
 - make depend DBGENV=RAM
 - make DBGENV=RAM
- ROMモニタを用いて実行形式(.srec)をダウンロード



2013/09/16

TOPPERSプロジェクト認定

106



RTCレジスタのアドレス

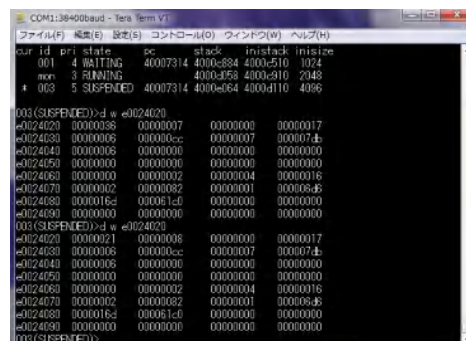
- RTCレジスタをDUMPLして確認する
- マップドIOのRTCのレジスタアドレスを以下の表のとおり

RTC_SEC	秒:HEX	0xE0024020
RTC_MIN	分:HEX	0xE0024024
RTC_HOUR	時間:HEX	0xE0024028
RTC_DOM	月中の日:HEX	0xE002402C
RTC_DOW	週中の日:HEX	0xE0024030
RTC_DOY	年中の日:HEX	0xE0024034
RTC_MONTH	月:HEX	0xE0024038
RTC_YEAR	年:HEX	0xE002403C

RTCレジスタのDUMP

- DISPLAY WORD E0024020コマンドでRTCのレジスタをDUMPする
- 表示の内容からRTCの時刻が読み取れる

0xE0024020 0x06⇒6秒
 0xE0024024 0x07⇒7分
 0xE0024028 0x00⇒0時
 0xE002402C 0x17⇒23日
 0xE0024038 0x07⇒7月
 0xE002403C 0x7db⇒2011年



```

COM1:38400baud - Tera Term V1
ファイル(F) 編集(E) 設定(S) コントロール(C) ウィンドウ(W) ヘルプ(H)
cur id pri state pc stack inistack inisize
001 4 WAITING 40007314 4000c884 4000c510 1024
non 3 RUNNING 4000c483 4000c410 2048
+ 002 5 SUSPENDED 40007314 4000cd64 4000cd10 4096
[003 (SUSPENDED)]> d w e0024020
e0024020 00000006 00000007 00000000 00000017
e0024024 00000000 00000000 00000007 00000007
e0024028 00000000 00000000 00000000 00000000
e002402c 00000000 00000000 00000000 00000000
e0024030 00000000 00000000 00000000 00000000
e0024034 00000000 00000000 00000000 00000000
e0024038 00000000 00000000 00000000 00000000
e002403c 00000000 00000000 00000000 00000000
[003 (SUSPENDED)]> d w e0024020
e0024020 00000021 00000008 00000000 00000017
e0024024 00000000 00000000 00000007 00000007
e0024028 00000000 00000000 00000000 00000000
e002402c 00000000 00000000 00000000 00000000
e0024030 00000000 00000000 00000000 00000000
e0024034 00000000 00000000 00000000 00000000
e0024038 00000000 00000000 00000000 00000000
e002403c 00000000 00000000 00000000 00000000
[003 (SUSPENDED)]>
  
```

タスクモニタを用いて時間設定と再開

- PIPE TIME/PIPE DATEコマンドで現在の時刻を設定
- TASK RESUMEコマンドでテストタスクを再開する
- 表示時間が設定時刻になっていることを確認する

```

COM1:38400baud - Tera Term VT
ファイル(F) 編集(E) 設定(S) コントロール(C) ウィンドウ(W) ヘルプ(H)
003(WAITING)>t s
execute sus_task(3) :: result = E_OK !
003(SUSPENDED)>pipe help
DATE (year #) (month #) (day #)
TIME (hour #) (min #) (sec #)
CLOCK DATE:TIME
003(SUSPENDED)>pipe date 2013 8 8
2013/08/08
003(SUSPENDED)>pipe time 16 0 0
16:00:00
003(SUSPENDED)>task res
execute rsn_task(3) :: result = E_OK !
003(WAITING)>log mode 5
set logmask=LOG_NOTICE logmask=LOG_EMERG !
003(WAITING)>2013/08/08
16:00:16
count[13] alarm[0] systime[63647]
2013/08/08
16:00:18
count[14] alarm[0] systime[65648]
2013/08/08
16:00:20
count[15] alarm[0] systime[67649]

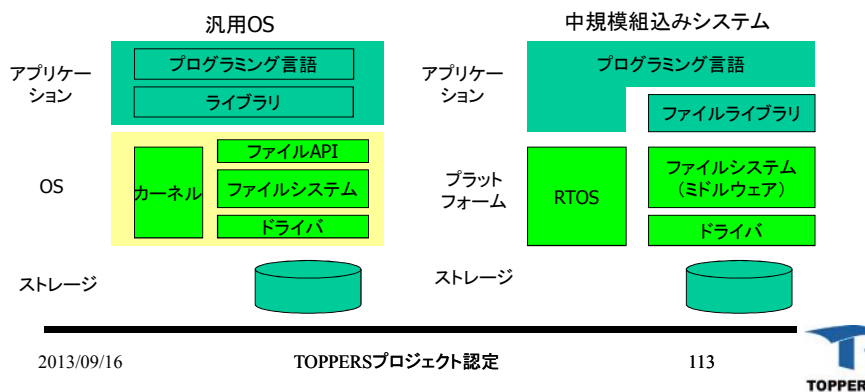
```

MCI/FAT/POSIXファイルシステム

1. 全体構成
2. POSIXファイルシステム
3. FatFs
4. SDメモ리카ードアクセス
5. MCIハードウェア

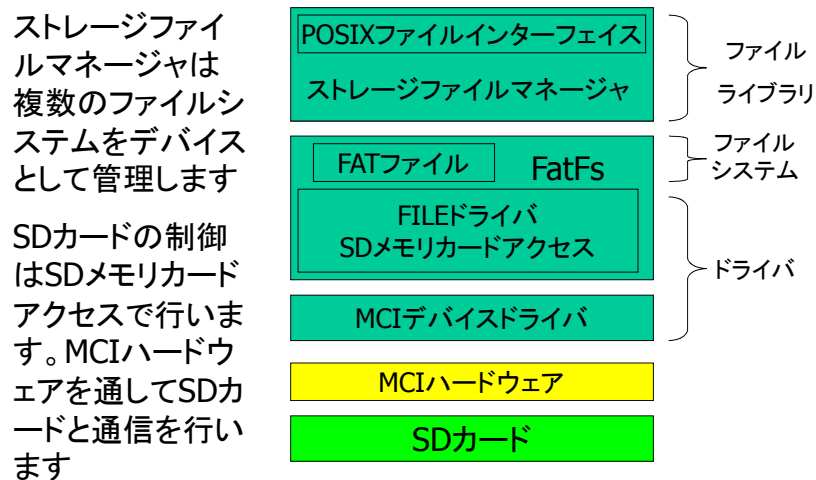
汎用OS用とのファイルシステムの違い

- LINUXのような汎用OSではファイルシステムはOSの機能として組み込まれる
 - 汎用OSごとのファイルAPIをもつ
- 組み込みシステムではファイルシステムはミドルウェアとして取り込むため種々のAPIとなる



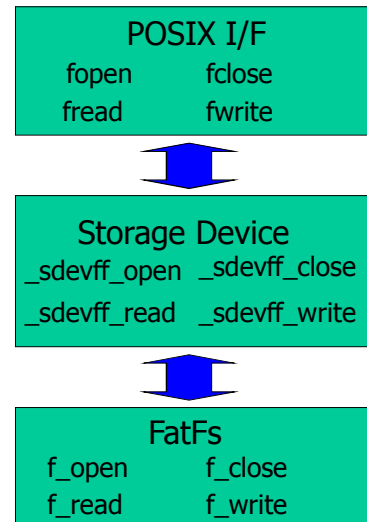
ファイルシステムの構成

- プラットフォーム内のファイルシステムの構成



ストレージファイルマネージャ

- ストレージファイルマネージャは教材として作成したデバイスドライバ管理プログラムです
- アプリケーションに対してPOSIXファイルインターフェイスを供給する
- POSIXファイルインターフェイスからのアクセスをデバイス毎のコールバック関数を通して各ファイルアクセスインターフェイスに変換する



FatFs: FATファイルシステム

- FatFsは赤松武志さんの作成によるFATファイルシステムモジュールです
- フリーソフトウェアとして公開
- FATファイル関数をストレージデバイスの読み出し、書き込み等のデバイスインターフェイスに変換
- FatFsでは、`diskio.c`というソースファイル中にデバイスインターフェイス関数を記載している
- この教材では、`diskio.c`のデバイスインターフェイス関数を作成する実習を行う
- `diskio.c`をSDメモリーカードアクセス用のデバイスインターフェイス関数に書き換えることによって、SDカードをFATファイルとしてアクセスすることができる

SDメモ리카ードアクセス

- SDメモ리카ードは1999年に松下電器産業を中心に共同規格が発表され、2000年からSDカードアソシエーションが設立され規格の管理を行っている
- SDメモ리카ードアクセスはSDメモ리카ードを読み出し、書き込みを行うためのインターフェイス
- SDメモ리카ード規格よりメモリ容量とアクセススピードをアップしたSDHCメモ리카ード規格、SDXCメモ리카ード規格がある
- SDIOという規格が追加されメモ리카ード以外のI/Oアクセスを行うことが可能
- SDメモ리카ードアクセスはSDメモ리카ードに対するCMDとREPLYとSTATUSにより通信を行う

MCI

- MCI(Multimedia Card Interface)はマルチメディアカードやSDメモ리카ードを制御するモジュールです
- MCIの機能
 - MMC及びSDメモ리카ードのホストコントロール機能
 - SDカードの4ビット・バスをサポート
 - DMAによるデータ転送機能
- データ転送はDMA方式とポーリング方式が使用できるが、ARM7CPUによるポーリング転送ではSDカードの転送速度に追従できないため、本システムではDMA転送を使用する

MCI/FAT/POSIXファイルシステム

1. 全体構成
2. [POSIXファイルシステム](#)
3. FatFs
4. SDメモ리카ードアクセス
5. MCIハードウェア

ファイル関数の選択

- プラットフォームのファイルAPIをPOSIX準拠とする
- ファイルライブラリで、よく使用される関数を選択する
 - C言語標準のもの(C89、C99)
 - POSIX.1-2001でよく使われるもの
 - LINUX固有であるが、必要なもの
- C89,C99はISOで定めたC言語標準
 - C89は1989年、C99は1999年の発行でC99は2000年5月にANSIとして受理されている
- アプリケーションからファイルを操作する場合、これらの関数を使用する

C言語標準の関数1

書式	機能	返回值
FILE *fopen(const char *path, const char *mode);	pathで指定された名前のファイルを開きストリームと結びつける	成功するとFILE型のポインタを返す。失敗するとNULL
int fclose(FILE *fp);	fpで指定されるストリームをフラッシュ後、クローズする	正常終了すると0、エラー時EOFを返す
size_t fread(void *ptr, size_t size, size_t nmemb, FILE *fp);	fpで指定されるストリームからnmemb個のデータを読みptrに格納する。個々のデータはsizeバイトの長さ	成功した場合要素の個数、エラーの場合少ない数
size_t fwrite(void *ptr, size_t size, size_t nmemb, FILE *fp);	ptrからnmemb個のデータをfpで指定されたストリームに書き込む。個々のデータはsizeバイトの長さ	成功した場合要素の個数、エラーの場合少ない数

2013/09/16

TOPPERSプロジェクト認定

121



C言語の関数2

書式	機能	返回值
int fputc(int c, FILE *fp);	キャラクタcをfpストリームに書き込む	成功した場合c、エラー時EOF
int fputs(const char *s, FILE *fp);	文字列sをfpストリームに書き込む	成功した場合負でない値、エラー時EOF
int putc(int c, FILE *fp);	マクロで実装されている可能性ありという点を除きfputcと同じ	成功した場合c、エラー時EOF
int fgetc(FILE *fp);	fpから次の文字を読み込む、ファイルの終わリエラー時EOF	機能を参照
char *fgets(char *s, int size, FILE *fp);	fpから最大size-1個の文字をsバッファに読み込む	エラー時NULLを返す
int fflush(FILE *fp);	fpストリームのユーザ領域のバッファをフラッシュする	成功時0、エラー時EOFを返す

2013/09/16

TOPPERSプロジェクト認定

122



C言語の関数3

- fpで指定するストリームはファイルを指すとは限らない、標準入出力の場合以下のストリームを使用する
 - stdin 標準入力用のストリーム
 - stdout 標準出力用のストリーム
 - stderr エラー出力用のストリーム
- ストリームを使用しないC言語関数を以下に示す

書式	機能	返回值
int rename(const char *oldpath, const char *newpath);	ファイルの名前を変更し、必要ならばディレクトリ間の移動を行う	成功した場合0、エラーの場合-1を返す
int remove(const char *pathname);	ファイルシステムからファイル名を削除する	成功した場合0、エラーの場合-1を返す

POSIX.1-2001の関数1

書式	機能	返回值
int open(const char *pathname, int flags);	pathnameに対応のファイルディスクリプタを返す	エラーの場合、-1
int close(int fd);	ファイルディスクリプタをクローズする	正常時0、エラー時-1
ssize_t read(int fd, void *buf, int count);	ファイルディスクリプタから最大countバイトをbufに読み込む	読み込みサイズ、0で終端、-1でエラー
ssize_t write(int fd, void *buf, int count);	bufからファイルディスクリプタに最大countバイトを書き込む	書き込みサイズ、-1でエラー
int stat(const char *path, struct stat *buf); int fstat(int fd, struct stat *buf); int lstat(const char *path, struct stat *buf);	ファイルについての情報を返す	成功時0、エラー時-1

POSIX.1-2001の関数2

書式	機能	返り値
int access(const char *pathname, int mode);	pathnameにアクセスできるかをチェック	成功時0、その他は-1
int mkdir(const char *path, mode_t mode);	ディレクトリを作成する	成功時0、その他は-1
int rmdir(const char *pathname);	空のディレクトリを削除する	成功時0、その他は-1
int chmod(const char *path, mode_t mode);	ファイルのアクセス許可を変更する	成功時0、失敗時は-1
int opendir(const char *name);	nameに対応したディレクトリストリームをオープンし、ストリームを返す	成功時ストリームのポインタ、エラー時NULL
int closedir(dir *dirp);	ディレクトリストリームをクローズする	成功時0、失敗時は-1

LINUX固有の関数

- LINUX固有であるが、ディレクトリ表示等に必要なため追加した関数は以下のとおり

書式	機能	返り値
int readdir(unsigned int fd, struct dirent *dirp, unsigned int count);	ファイルディスクリプタfdが参照しているディレクトリからdirent構造体を読み、dirpで指されたバッファに格納する	成功すれば1、エラーならば-1を返す
int statfs(const char *path, struct statfs *buf);	マウントされたファイルシステムについての情報を返す	成功すれば0、エラーならば-1

MCI/FAT/POSIXファイルシステム

1. 全体構成
2. POSIXファイルシステム
3. [FatFs](#)
4. SDメモ리카ードアクセス
5. MCIハードウェア



FATファイルシステム

- FATファイルシステムはマイコン用のBasic言語のファイル管理用に開発され、MS-DOSに使用されて普及した
- FATとは、ファイル管理方式File Allocation Tableがそのまま名称となった
- ストレージの大容量化によりFAT12/FAT16/FAT32の3つのサブタイプがある

予約領域	ボリュームの管理領域、BPBやブートセクタが置かれる
FAT領域	File Allocation Tableを置く領域
ルートディレクトリ領域	ルートのディレクトリデータを置く領域、FAT32では存在しない
データ領域	実際のファイルデータがクラスタ単位に置かれる領域

FATファイルシステムのストレージ構成



FATファイルシステムのアーキテクチャ

- ストレージはセクタと呼ばれる最小ブロック単位にアクセスされる
 - 一般的なセクタは512バイトが多い
 - ストレージの先頭セクタを0として昇順に割り付けられる（あくまでアクセス単位で、物理的な位置ではない）
- いくつかのセクタをまとめてクラスタという単位でデータを扱う
- FATはクラスタを管理するテーブル
 - FATの12,16,32は管理できるクラスタのビット数
最大管理できるボリューム要領がきまる
 - クラスタ番号やデータはリトルエンディアン管理

BPBによるボリューム管理

- 予約領域の先頭をブートセクタと呼ぶ
 - 初期にMS-DOSやBasic言語の起動情報が書かれた
 - MSDOS Ver2からBPBというメディア管理情報を置く
- BPBはBIOS Parameter Blockの略
 - FAT12/16とFAT32は別の専用フィールドを持つ

共通領域のBPB定義

フィールド名	オフセット	サイズ	機能
BS_jmpBoot	0	3	ブート用ジャンプ命令
BS_OEMName	3	8	システム名称
BPB_BytsPerSec	11	2	セクタ当りのバイト数
BPB_SecPerClus	13	1	クラスタ当りのセクタ数
BPB_RsvSecCnt	14	2	予約領域のセクタ数
BPB_NumFATs	16	1	FATの数
BPB_RootEntCnt	17	2	ルートディレクトリのサイズ
BPB_TotSec16	19	2	ボリュームの総セクタ数1
BPB_Media	21	1	メディア・ディスクリプタ・バイト数
BPB_FATSz16	22	2	1個のFATが占めるセクタ数
BPB_SecPerTrk	24	2	トラック当りの物理セクタ数
BPB_NumHeads	26	2	物理ヘッド数
BPB_HiddSec	28	4	隠れた物理セクタ
BPB_TotSec32	32	4	ボリュームの総セクタ数2

FAT12/16用のBPB定義

フィールド名	オフセット	サイズ	機能
BS_DrvNum	36	1	物理ドライブ番号
BS_Reserved1	37	1	予約
BS_BootSig	38	1	拡張ブート・シグネチャ
BS_VolID	39	4	ボリューム・シリアル番号
BS_VolLab	43	11	ボリュームラベル文字列
BS_FilSysType	54	8	FATタイプの文字列

ディレクトリ・エントリ

- FATにはショートネームとロングネームの2つの形式のディレクトリ形式がある
- 基本はショートネーム用のディレクトリ・エントリ構造体(SFN)でファイル管理を行う

フィールド名	オフセット	サイズ	機能
DIR_Name	0	11	ファイル名
DIR_Attr	11	1	属性
DIR_NTRes	12	1	NTタイプ
DIR_CrtTimeTenth	13	1	ファイル作成時刻0.1秒
DIR_CrtTime	14	2	ファイル作成時刻
DIR_CrtDate	16	2	ファイル作成日付
DIR_LstAccDate	18	2	アクセス日付
DIR_FstClusHI	20	2	開始クラスタ番号HI
DIR_WrtTime	22	2	書き込み時刻
DIR_WrtDate	24	2	書き込み日付
DIR_FstClusLO	26	2	開始クラスタ番号LOW
DIR_File	28	4	ファイル・サイズ

未使用ディレクトリはDIR_Name[0]を0xE5をセット

先頭データのクラスタ番号はDIR_FstClusHI/LOにセット

クラスタ・チェーン

- FATエントリの値はクラスタと1対1の対応をもつ
- FATはファイルのクラスタ・チェーンを管理する

FATエントリとクラスタの対応

FAT[0]	クラスタ0
FAT[1]	クラスタ1
FAT[2]	クラスタ2
FAT[3]	クラスタ3
FAT[4]	クラスタ4
...	...

クラスタ0, 1がデータ領域である場合対応を持たない

ファイルのクラスタ・チェーン

ファイル名	先頭クラスタ番号
FILE1.TXT	3

FILE1.TXTはデータ領域としてクラスタ3,4,5,9,10を持つ1クラスタが4KBとすると20KBのファイル領域を持つ

FAT	値
FAT[2]	0
FAT[3]	4
FAT[4]	5
FAT[5]	9
FAT[6]	7
FAT[7]	8
FAT[8]	EOC
FAT[9]	10
FAT[10]	EOC
FAT[11]	0
FAT[12]	0

FatFsの概要

- FatFsはフリーソフトウェアとして公開されている
- FatFsは以下の特徴を持つ
 - Windows互換FATファイル・システム
 - プラットフォームに依存しない
 - コンパクトなコード・サイズとRAM使用量
 - 複数のボリュームに対応(物理ドライブ・区画)
 - 複数のANSI/OEMコードページに対応
 - ロングファイルネームにも対応
(Unicode APIも選択可能)
 - RTOSに対応
 - 記述はC89に準拠

FatFsのAPI関数

- FatFsの上位レイヤへのインターフェイスは以下のとおり

API関数	機能
f_mount	ワーク・エリアの登録削除
f_open	ファイルのオープン・作成
f_close	ファイルのクローズ
f_read	ファイルの読み出し
f_write	ファイルの書き込み
f_sync	キャッシュデータのフラッシュ
f_lseek	リード/ライト・ポインタの移動
f_opendir	ディレクトリのオープン
f_readdir	ディレクトリの読み出し
f_stat	ファイル情報の取得
f_getfree	ボリュームの空き領域の取得
f_truncate	ファイルの切り詰め
f_unlink	ファイルやディレクトリの削除

API関数	機能
f_mkdir	ディレクトリの作成
f_chmod	アトリビュートの変更
f_utime	タイムスタンプの変更
f_rename	名前の変更
f_chdir	カレント・ディレクトリの変更
f_chdrive	カレント・ドライブの変更
f_mkfs	ボリュームのフォーマット
f_forward	ファイル・データをストリームに転送
f_putc	文字の書き込み
f_puts	文字列の書き込み
f_printf	書式付き文字列の書き込み
f_gets	文字列の読み込み

 本教材では未使用

FatFsの下位レイヤ・インターフェイス

- FatFsがボリュームにアクセスするために、下位レイヤ・インターフェイスを用意する必要がある
- 要求される下位インターフェイスはオプション指定に従う

下位レイヤ関数	必要となる条件	機能
disk_initialize	常時	ドライブの初期化
disk_status		ディスクステータス取得
disk_read		セクタの読み込み
disk_write	_FS_READONLY == 0	セクタの書き込み
get_fattime		時刻の取得
disk_ioctl(CTRL_SYNC)		書き込み時の同期
disk_ioctl(GET_SECTOR_COUNT)	_USE_MKFS == 1	セクタサイズ取得
disk_ioctl(GET_SECTOR_SIZE)	_MAX_SS >= 1024	総セクタ数取得
disk_ioctl(GET_BLOCK_SIZE)	_USE_MKFS == 1	消去ブロックサイズ取得

 は本教材では使用しない

POSIX-APIとの連結

- POSIX準拠の関数はStorageDeviceFileFunc_tの関数テーブルを用いて、FatFsの上位レイヤ・インターフェイスを呼び出す構成とした(ソースファイルstdio.c/stdfile.c)

```

/*
 * POSIXファイルインターフェイス用関数テーブル定義
 */
typedef struct StorageDeviceFileFunc {
    void (*_sdevff_opendir)(const char *name);
    int (*_sdevff_closedir)(void *dir);
    int (*_sdevff_readdir)(void *dir, void *dirent2);
    int (*_sdevff_mkdir)(const char *name);
    int (*_sdevff_rmdir)(const char *name);
    int (*_sdevff_unlink)(const char *name);
    int (*_sdevff_rename)(const char *oname, const char *nname);
    int (*_sdevff_chmod)(const char *name, int mode);
    int (*_sdevff_stat)(const char *name, struct stat *buf);
    int (*_sdevff_statfs)(const char *name, void *status);
    int (*_sdevff_open)(const char *name, int flags);
    int (*_sdevff_close)(int fd);
    int (*_sdevff_fstat)(int fd, struct stat *buf);
    off_t (*_sdevff_lseek)(int fd, off_t offset, int whence);
    long (*_sdevff_read)(int fd, void *buf, long count);
    long (*_sdevff_write)(int fd, const void *buf, long count);
    void (*_sdevff_mmap)(void *start, size_t length, int prot, int flags, int fd, off_t offset);
} StorageDeviceFileFunc_t;

```

MCI/FAT/POSIXファイルシステム

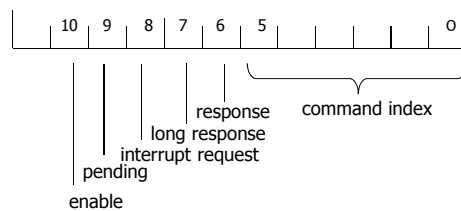
1. 全体構成
2. POSIXファイルシステム
3. FatFs
4. [SDメモ리카ードアクセス](#)
5. MCIハードウェア

SDカードインターフェイス

- SDカード(Secure Digital Memory Card)を制御するには、MCI等の通信回路を介して、コマンドにて機能要求、レスポンスの要求を行い、目的に添って制御を行う必要がある
- SD Specifications Physical Layer Specification Version 3.00で41つのCMD、7つのACMDが制定されている
 - ACMDはCommand Indexが6ビットしかないため、CMD55(APP_CMD)により拡張したコマンドである
- CMDの実行結果は制御方式により異なる、MCIではMCIStatusレジスタの22ビットのステータスビットで実行結果を示す

コマンド

- CMD: commnd index 以外はコントローラ依存
 - 11ビットのCPSM(Command Path State Machine)制御



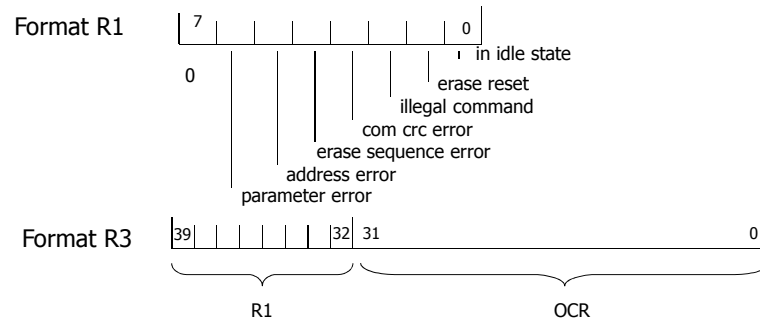
- ARGUMENTS
 - CMDの種別により32ビットの引数をもつ

よく使うコマンド

command index	argument	response	abbreviation
CMD0		-	GO_IDLE_STATE
CMD2		R2	ALL_SEND_CID
CMD3		R6	SEND_RELATIVE_ADDR
CMD7	RCA	R1b	SELECT/DESELECT CARD
CMD8	VHS/check pattern	R7	SEND_IF_COND
CMD9	RCA	R9	SEND_CSD
CMD10	RCA	R2	SEND_CID
CMD13		R1	SEND_STATUS
CMD16	block length	R1	SET_BLOCKLEN
CMD17	data address	R1	READ_SINGLE_BLOCK
CMD18	data address	R1	READ_MULTIPLE_BLOCK
CMD24	data address	R1	WRITE_BLOCK
CMD25	data address	R1	WRITE_MULTIPLE_BLOCK
ACMD6	bus width	R1	SET_BUS_WIDTH
ACMD41	HCS/XPC/S18R/Vdd	R3	SD_SEND_OP_COND

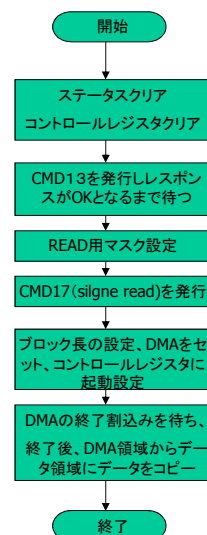
レスポンス

- コマンドの設定により最大127ビットのレスポンスを取得する



ポーリング・シングル・リードの例

- SDカードからのDMAを使用したシングルブロック・リードのフローチャートを左図に示す
- CMD13はカードのステータスをレスポンスとして読み込むコマンドです
- SDカードへの操作はコマンドによる設定にて行う

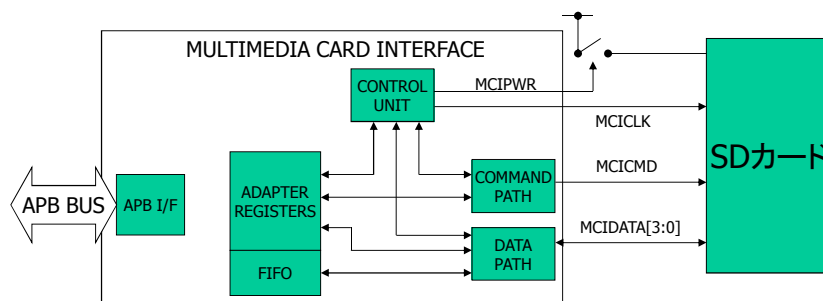


MCI/FAT/POSIXファイルシステム

1. 全体構成
2. POSIXファイルシステム
3. FatFs
4. SDメモ리카ードアクセス
5. MCIハードウェア

ハードウェア仕様

- MCIのハードウェアの説明はLPC23XX_umの第20章：
LPC23XX SD/MMC Interface記載
- SD (Secure Digital) カードとは、MCI回路を通して通信を行う



MCI制御レジスタ

- MCI制御レジスタは32ビットアクセス、リトルエンディアン
- 初期値はすべてゼロ
- 詳細はユーザーズマニュアルを参照のこと

name	address	access	width	discription
MCIPower	0xE008C000	R/W	8	電源と信号の駆動制御
MCI Clock	0xE008C004	R/W	12	クロック出力とバス幅指定
MCIArgument	0xE008C008	R/W	32	コマンドの引数設定
MMCCommand	0xE008C00C	R/W	11	コマンドレジスタ
MCIRespCmd	0xE008C010	RO	6	受信コマンドレスポンスの保持
MCIResResponse0	0xE008C014	RO	32	受信レスポンス保持0
MCIResResponse1	0xE008C018	RO	32	受信レスポンス保持1
MCIResResponse2	0xE008C01C	RO	32	受信レスポンス保持2
MCIResResponse3	0xE008C020	RO	31	受信レスポンス保持3
MCIDataTimer	0xE008C024	R/W	32	データタイムアウト時間設定
MCIDataLength	0xE008C028	R/W	16	転送バイト数設定
MCIDataCtrl	0xE008C02C	R/W	8	データ制御設定
MCIDataCnt	0xE008C030	R/W	16	データ転送バイト数保持
MCIStatus	0xE008C034	RO	22	MCIステータスレジスタ
MCI Clear	0xE008C038	WO	11	MCIステータスクリア設定
MCI Mask0	0xE008C03C	R/W	22	割込みマスク設定
MCI FifoCnt	0xE008C048	RO	15	FIFO用カウンタ
MCI FIFO	0xE008C080 to 0xE008C0BC	R/W	32	データFIFO読み書きポート

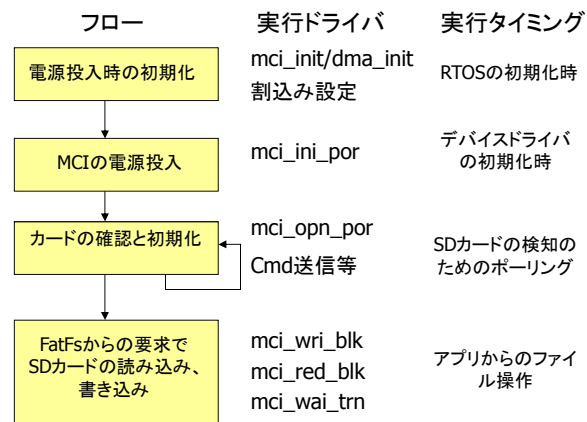
MCI制御関数

- MCIを制御する関数として以下を用意した

ドライバ関数	機能	備考
mci_init	MCIハードウェアの初期化関数	ATT_INI
lpc23xx_mci_set_mask	割込みマスク設定	
lpc23xx_mci_set_clock	クロックの設定	
lpc23xx_mci_snd_acmd	ACOMMANDの送信	
lpc23xx_mci_snd_cmd	COMMANDの送信	
lpc23xx_mci_get_resp	レスポンスの取得	
lpc23xx_mci_ini_por	MCIドライバの初期化	管理ブロック作成
lpc23xx_mci_opn_por	MCIドライバのオープン	カード認証
lpc23xx_mci_cls_por	MCIドライバのクローズ	
lpc23xx_mci_wri_blk	MCIデバイスへのデータ書き込み	
lpc23xx_mci_rea_blk	MCIデバイスからのデータ読み込み	
lpc23xx_mci_wai_trn	MCIデバイスの転送待ち	

制御フロー

- 大まかな制御フローでは、4つの操作に分けられる
- ハードウェア制限でカードの抜き差し制御は動作しない



2013/09/16

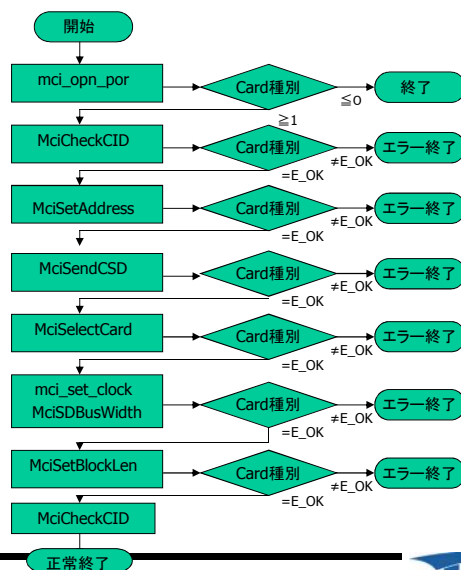
TOPPERSプロジェクト認定

147



カードの確認

- mci_opn_porがカード種別を返さない場合はカードが挿入されていないとして終了、再リトライとなる
- カード設定や判定が不良の場合エラーとする
- すべて正常の場合 f_mountしてFatFsにて使用可能となる



2013/09/16

TOPPERSプロジェクト認定

148



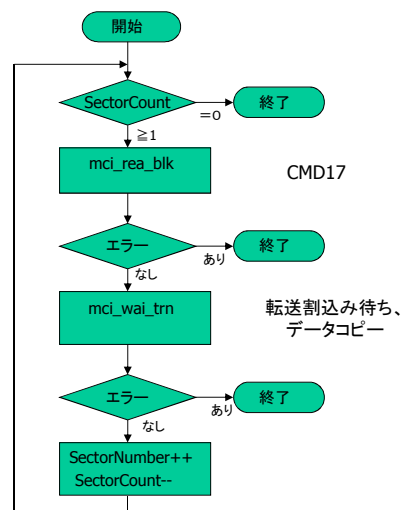
データ転送方式

- LPC23XXでは2つの転送方式が可能
 - CPUによるFIFOのポーリング
 - GPDMAを用いたDMA転送
- ROM実行でのCPUのポーリングでは転送速度に追従しない場合があり、DMA転送方式で実装した
- GPDMAを制御するために以下のドライバを用意した

ドライバ関数	機能	備考
dma_init	GPDMAハードウェアの初期化関数	ATT_INI
lpc23xx_set_trn	DMA転送パラメータ設定	
lpc23xx_set_cfg	GPDMAのコンフィギュレーション設定	

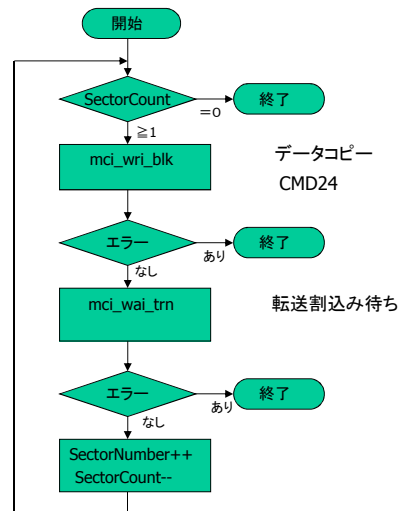
データ・リード

- データの受信はシングル・セクタ・リード(CMD17)を使用
- 転送待ちはGPDMAの完了割込みにて認識
- 転送データをGPDMA領域からDRAMにコピーを行う



データ・ライト

- 転送データは転送前にDMA領域からGPDMA領域にコピー
- データの送信はシングル・セクタ・ライト(CMD24)を使用
- 転送待ちはGPDMAの完了割り込みにて認識



SDファイルシステムの確認

1. [MCI/SDドライバの確認](#)
2. FAT-FSデバイスIOの確認

ファイルテストを行うテストプログラム

- MCIテストプログラム
 - MCIドライバ、DMAドライバを使ってSDメモリ・カードと通信しセクタをREAD/WRITEする
 - MCIテストプログラムを用いてセクタを読み込み、BPB、FAT、ディレクトリを確認する
- ファイルテストプログラム
 - C言語ファイル関数を用いてSDメモリ・カードにマウントし、ファイルの作成、読み書きテスト等を行う
 - C言語ファイル関数はストレージファイルマネージャを経由してFatFsの上位インターフェイスに変換する
 - 別のファイルシステムを使用する場合、ストレージファイルマネージャの変換部を作成すれば、テストプログラム(アプリ)は、そのまま使用可能

MCIテストプログラム

- MCIテストプログラムの置き場所
 - RTOSディレクトリ/OBJ/LPC2388/MCI
MCIテストプログラムがMCI用コマンドプログラムやMCIドライバ、DMAドライバを制御してSDメモリ・カードのREAD/WRITEとデータ表示を行う
- MCIコマンドプログラムの置き場
 - RTOSディレクトリ/files
- MCIドライバの置き場所
 - RTOSディレクトリ/pdic/lpc23xx
- DMAドライバの置き場所
 - RTOSディレクトリ/pdic/lpc23xx

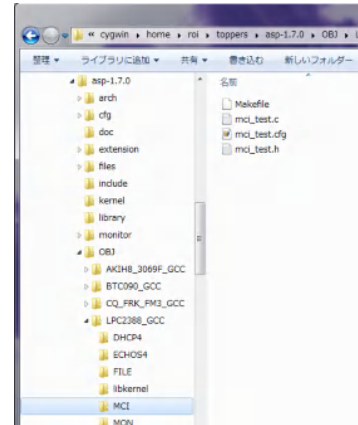
MCIテストプログラムの概要:mci_test解説

- コマンドを用いてセクタのREAD/WRITEを行う
- 学習内容

- ドライバI/Fの理解
- SDメモ리카ードとの通信手順を理解
- FATで使用するセクタの確認を行う

- 構成ファイル

- mci_test.h インクルードファイル
- mci_test.c テストファイル本体
- mci_test.cfg コンフィギュレーションファイル



asp-1.7.0用ディレクトリ

mci_test解説:MCIドライバ

- MCIを制御するドライバ
 - pdic/lpc23xxディレクトリに置かれる
 - mci.cfg MCIドライバが使用するRTOSリソースを定義
 - mci.h MCIのデータ定義
 - lpc23xx_mci.h MCIドライバのパラメータ、関数定義
 - lpc23xx_mci.c MCIドライバの関数
- MCIのブロックREADに使用するのは以下の2つの関数
 - ER mci_rea_blk(MCIPCB *pmci, B *buf, W blockNum);
 - ブロックREAD開始設定、
 - ER mci_wai_trn(MCIPCB *pmci, W timeout);
 - ブロック転送終了待ち
 - リネーム設定で最初のlpc23xx_はなくなる

mci_test解説: DMAドライバ

- DMAを制御するドライバ
 - pdic/lpc23xxディレクトリに置かれる
 - lpc23xx_gpdma.h DMAのパラメータ、関数定義
 - lpc23xx_gpdma.c DMAドライバ関数
- RTOSリソースはMCIのコンフィギュレーションファイルで定義
- DMAドライバはMCIドライバから制御するため、ユーザは直接実行する必要なし
- 関数
 - void dma_init(VP_INT exinf);
 - void dma_isr0(void);
 - ER set_trn(UB cnum, UB mode, UW src, UW dst, UW size);
 - ER set_cfg(UB cnum, UB sp, UW dp, UB mode);



mci_test解説: MCIコマンドプログラム

- カード確認用のコマンド関数をfiles/mcicmd.hとfiles/mcicmd.cに用意した
- この関数群はMCIドライバのコマンドとレスポンス関数を使用してカードの確認を行う
- カード確認に必要な関数は以下の6関数

関数名	処理内容	使用コマンド
MciSDBusWidth	SDカードのバス幅の設定	ACMD6
MciCheckCID	CIDを取得する	CMD2
MciSetAddress	相対アドレス(RCA)の取得	CMD3
MciSendCSD	CSD(Card Specific Data)の取得	CMD9
MciSelectCard	Select Cardの送信	CMD7
MciSetBlockLen	ブロック長の設定	CMD16



mci_test解説: Makefile

- タスクモニタ用のSAMPLE1を改造して使用
- コンパイルオプションとシステムサービスに関する定義にMCIコマンド、MCIドライバ、DMAドライバを追加

```
#
# 共通コンパイルオプションの定義
#
COPTS := $(COPTS) -g
ifndef OMIT_WARNING_ALL
COPTS := $(COPTS) -Wall
endif
ifndef OMIT_OPTIMIZATION
COPTS := $(COPTS) -O2
endif
CDEFS := $(CDEFS) -DSUPPORT_PIPE
INCLUDES := -I. -I$(SRCDIR)/include -I$(SRCDIR)/arch -I$(SRCDIR) -I$(SRCDIR)/files -I$(SRCDIR)/pdic/lpc23xx $(INCLUDES)
LDFLAGS := $(LDFLAGS)
LIBS := $(LIBS) -lc $(CXXLIBS)
CFLAGS = $(COPTS) $(CDEFS) $(INCLUDES)
:
#
# システムサービスに関する定義
#
SYSSVC_DIR := $(SYSSVC_DIR) $(SRCDIR)/syssvc $(SRCDIR)/files $(SRCDIR)/pdic/lpc23xx $(SRCDIR)/library
SYSSVC_ASMOBS := $(SYSSVC_ASMOBS)
SYSSVC_COBJS := $(SYSSVC_COBJS) banner.o syslog.o serial.o logtask.o ¥
log_output.o vasylog.o t_perror.o strerror.o ¥
mccmd.o lpc23xx_mci.o lpc23xx_gpdma.o armv4.o $(CXXRTS)
```

mci_test解説: コマンド処理部(mci_test.c)

- タスクモニタのPIPEコマンドを用いて、セクタREAD、WRITEコマンドを追加する
- セクタ番号でセクタの読み出し表示、書き込みを行う

```
/*
 * パイプコマンドテーブル
 */
static const struct SUBCOMMAND_TABLE const mon_pipe[] = {
    {"READ", PIPE_READ}, /* 1ブロックREAD */
    {"WRITE", PIPE_WRITE}, /* 1ブロックWRITE */
    {"EXIT", PIPE_EXIT}, /* 終了 */
    {"HELP", PIPE_HELP}
};

/*
 * PIPEコマンドのディスパッチ
 */
INT pipe_command(B *command)
{
    int point=0;
    char cmd=1;
    int no, count, arg1;
    char reg;
    count = sizeof(mon_pipe) / sizeof(struct SUBCOMMAND_TABLE);
    if(command[point]){
        for(no = 0; no < count; no++){
            :
        }
    }
}
```

mci_test解説:カードの確認(mic_test.c)

- テストプログラムはmain_taskで実行する
- MCIデバイスの初期化、オープンの後、カード確認コマンドを実行してカードの認識を行う

```

/*
 * メインタスク
 */
void main_task(VP_INT exinf)
{
    ER_UINT  ercd;
    MCIPCB   *pmci;
    :
    pmci = mci_ini_por();
    if(pmci == NULL){
        syslog_0(LOG_NOTICE, "mci_ini_pro ERROR !!");
        slp_tsk();
        /* fatal error */
    }
    MCI_CardType = mci_opn_por(pmci);
    syslog_2(LOG_NOTICE, "Open [%08x][%d]!", pmci, MCI_CardType);
    if(MCI_CardType <= 0){
        syslog_0(LOG_NOTICE, "mci_opn_por ERROR !!");
        slp_tsk();
        /* fatal error */
    }
    if(MCICheckCID(pmci) != E_OK){
        syslog_0(LOG_NOTICE, "MCI_Check_CID ERROR ##");
        slp_tsk();
        /* fatal error */
    }
}

```

2013/09/16

TOPPERSプロジェクト認定

161



mci_test解説:READの実行(mci_test.c)

- セクタのREAD/WRITEを行うプログラムはREAD/WRITEの実行の後にある
- 青字の部分にプログラムを記載してテストを行う

```

/*
 * READ/WRITEの実行
 */
while(reqcmd != PIPE_EXIT){
    while(actcount == 0){
        dly_tsk(100);
    }
    if(reqcmd == PIPE_READ){
        printf("\nread sector [%d][%08x]\n", sector, actaddr);

        ここに、sector番号からREADしたデータをactaddrにセットする関数を記載

        for(i = 0; i < BLOCK_LENGTH; i += 16){
            if((i % 16) == 0){
                printf("\n%04x ", i);
            }
            for(j = 0; j < 16; j++){
                printf("%02x ", (UB)actaddr[i+j]);
            }
            printf("\n");
            for(j = 0; j < 16; j++){
                if(actaddr[i+j] < ' '){
                    printf(".");
                }
                else{
                    tbuf[0] = actaddr[i+j];
                }
            }
        }
    }
}

```

2013/09/16

TOPPERSプロジェクト認定

162



mci_test解説: コンフィギュレーションファイル

- 静的APIを記述
- DMAドライバとMCIDライバの初期化、割込みを設定
 - プログラムでの初期化、割込みの設定は不要
- テスト用のプログラムをタスクとして起動

```

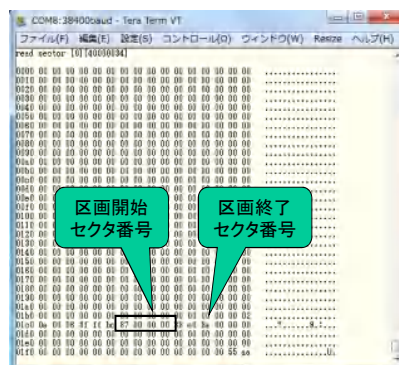
INCLUDE("target_timer.cfg");
INCLUDE("syssvc/syslog.cfg");
INCLUDE("syssvc/banner.cfg");
INCLUDE("syssvc/serial.cfg");
INCLUDE("syssvc/logtask.cfg");
INCLUDE("pdic/lpc23xx/mci.cfg");
INCLUDE("monitor/monitor.cfg");
#include "mci_test.h"
CRE_TSK(MAIN_TASK, { TA_NULL|TA_ACT, 0, main_task,
                    MAIN_PRIORITY, STACK_SIZE, NULL });

```

MCIDライバと
DMAドライバの
コンフィギュレーション
設定

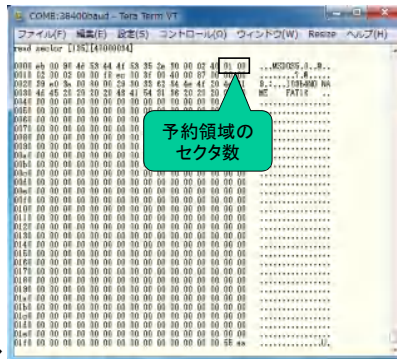
演習: パーティション

- 2GB・SDメモリ・カードの
0セクタにパーティション
設定されている
- 0x1be~0x1cdが最初区
画エントリー
- 区画開始セクタ番号:
LBA(0x1c6~0x1c9)
– 0x00000087 = 135
- FAT領域の先頭が135セ
クタとなる



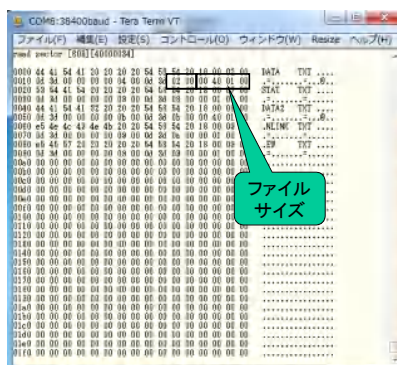
演習: BPBの確認

- 135セクタをDUMPする
- 予約領域にBPBがある
- 予約領域のセクタ数が1なので、FAT領域は136セクタにある
 - $135 + 1 = 136$
- BPBを確認する
 - BPB_NumFATs = 2
 - BPB_FATSz16 = 236
 - BPB_SecPerClst = 2
 - BPB_RootEntCnt = 2
 - BPB_TotSec32 = 3,858,489
- FATの大きさは472セクタ、ルートディレクトリは608セクタ = $136 + 472$ にある



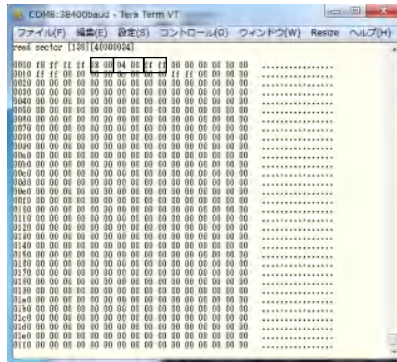
演習: ディレクトリ領域の確認

- 先にルートディレクトリをDUMPする
- ひとつのディレクトリは32バイトのエントリで構成される
- 最初のディレクトリはDATA.TXTは81,920バイト = $0x14000$ で、最初のクラスタは2クラスタを指す
- ルートセクタのクラスタ数が2なので、データ領域の先頭が2クラスタ目となる



演習:FAT領域の確認

- 2GB・SDメモ리카ードは
FATサイズ=2バイト(16
ビット)
- DATA.TXTは3, 4, 5の
3クラスタを占有する(クラ
スタ・チェーン)
- 1クラスタ=64セクタ、1セ
クタ=512バイト
- DATA.TXTは98,304バイ
ト(≥81,920)の領域をも
つ



SDファイルシステムの確認

1. MCI/SDドライバの確認
2. [FAT-FSデバイスIOの確認](#)

ファイルテストプログラム

- ファイルテストプログラムの置き場所
 - RTOSディレクトリ/OBJ/LPC2388/FILE
C言語ファイル関数を用いてSDメモ리카ード上のファイル実行テストを行う
- ファイルライブラリの置き場
 - RTOSディレクトリ/files
- FatFsとSDカードデバイスドライバの置き場
 - RTOSディレクトリ/files/ff
- テストプログラムはFatFs for TOPPERSをC言語ファイル関数用に書き換えたものです
- SDカードデバイスドライバを完成し、ファイルテストにて正しく動作することを確認する

2013/09/16

TOPPERSプロジェクト認定

169



ファイルテストプログラムの概要: sample1解説

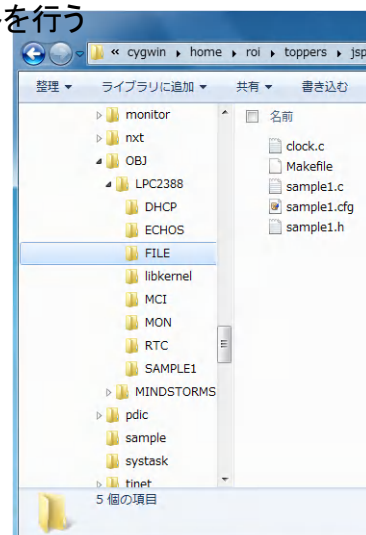
- ファイルシステムの基本テストを行う

- 学習内容

- C言語ファイル関数の理解
- FatFs上位インターフェースの理解
- デバイスドライバの作成

- 構成ファイル

- sample1.h インクルードファイル
- sample1.c テストファイル本体
- clock.c 日時設定
- sample1.cfg コンフィギュレーションファイル



2013/09/16

TOPPERSプロジェクト認定

170



ファイルテストプログラム概要: テスト内容

- C言語ファイル関数を用いてファイルのアクセステストを行う、以下の関数をsample1.cのmain_task関数で実行

テスト関数	テスト内容
test_read_write	data.txtを作成し固定データを書き込み、また読み込んでデータを 確認する
test_seek	data.txtをオープンして最後のクラスタの先頭にシークし、最後の クラスタデータを確認する
test_mkdir	サブディレクトリsub1,sub1/sub2,sub1/sub2/sub3を作成する
test_dir	SDカード中のファイルとサブディレクトリを表示する
test_stat	stat.txtを作成し、モードの変更、ファイルの状態の表示を行う
test_rename	old.txtを作成し、new.txtに改名しstat関数を使って確認を行う
test_unlink	unlink.txtを作成後、削除してstat関数で削除を確認する
test_read_write2	data2.txtを作成し固定データを書き込み、また読み込んでデータ を確認する。その後ファイルを削除
test_getfree	SDカードの空きサイズを表示する

ファイルテストプログラム概要: 関数の変換

- テストプログラムで使用したファイル関数
- テストプログラムはファイルライブラリを呼び出しファイルライブラリ関数はFatFsの上位インターフェイスに変換する

テスト関数	ファイルライブラリ関数	FatFs上位インターフェイス
test_read_write	fopen,fread,fwrite,fclose	f_open,f_read,f_write,f_close
test_seek	fopen,fseek,fread,fwrite,fclose	f_open,f_seek,f_read,f_write,f_close
test_mkdir	mkdir,opendir,closedir	f_opendir,f_mkdir
test_dir	opendir,closedir,readdir,rmdir	f_opendir,f_readdir,f_unlink
test_stat	fopen,fclose,stat,chmod,unlink	f_open,f_close,f_stat,f_chmod,f_unlink
test_rename	fopen,fwrite,fclose,stat,rename	f_open,f_write,f_close,f_stat,f_rename
test_unlink	unlink,stat	f_unlink,f_stat
test_read_write2	fopen,fread,fwrite,fclose,unlink	f_open,f_read,f_write,f_close,f_unlink
test_getfree	statfs	f_statfs

ファイルライブラリの説明

- filesディレクトリ中にファイルライブラリを用意した
 - files/stdfile.c: C言語ファイルライブラリ関数
 - files/stdio.c: POSIXファイル関数群
 - files/storagedevice.cfg: ストレージファイルマネージャのRTOSオブジェクト定義
 - files/storagedevice.h: ストレージファイルマネージャのインクルードファイル
 - files/storagedevice.c: ストレージファイルマネージャのソースファイル
- ファイルテストプログラムを標準のファイル関数で作成することができる

FatFsと下位インターフェイスの説明

- files/ffディレクトリ中にFatFsのプログラムを用意した
 - fatfs.cfg: FatFsが使用するOSオブジェクトを定義したコンフィギュファイル
 - diskio.h: FatFsの下位インターフェイス用インクルードファイル
 - ff.h: FatFs用FAT定義インターフェイスインクルードファイル(上位インターフェイスを定義)
 - ff.c: FatFs FATファイルシステムソースファイル
 - integer.h: FatFs用変数型定義
 - Makefile.config: TOPPERS/JSP用メイクファイル定義
 - sddiskio.c: 下位インターフェイスソースファイル
- sddisk.cをSDメモリファイルドライバに対応させることによりSDメモリファイルをFATファイルとしてアクセスすることができる

sample1解説: Makefile

- 必要なミドルウェアとデバイスドライバの取り込み(修正済)

```
#
# 共通コンパイルオプションの定義
#
COPTS := $(COPTS) -g
ifndef OMIT_WARNING_ALL
COPTS := $(COPTS) -Wall
endif
ifndef OMIT_OPTIMIZATION
COPTS := $(COPTS) -O2
endif
COPTS := $(COPTS) -DSUPPORT_VOLUME -DSUPPORT_PIPE -DSDEV_SENSE_ONETIME
INCLUDES := -I -I$(SRCDIR)/include -I$(SRCDIR)/arch -I$(SRCDIR) -I$(SRCDIR)/pdic/lpc23xx $(INCLUDES)
LDLAGS := $(LDLAGS)
LIBS := $(LIBS) -lc -lgcc $(CXXLIBS)
CFLAGS = $(COPTS) $(CDEFS) $(INCLUDES)

#
# アプリケーションプログラムに関する定義
#
APPLNAME = sample1
APPLDIR =
APPL_CFG = $(APPLNAME).cfg
APPL_DIR = $(APPLDIR) $(SRCDIR)/library
APPL_ASMOBS =
ifdef USE_CXX
APPL_CXXOBS = $(APPLNAME).o
APPL_COBJS = clock.o
else
APPL_COBJS = $(APPLNAME).o clock.o
endif
```

タスクモニタの機能拡張

時刻設定プログラム
を追加

2013/09/16

TOPPERSプロジェクト認定

175



Sample1解説: Makefile

- タスクモニタ、FatFs、ファイルライブラリをメイクファイルのインクルードで追加(修正済)
- デバイスドライバをSYSSVC_COBJSに追加(修正済)

```
# ミドルウェアの Makefile のインクルード
#
include $(SRCDIR)/monitor/Makefile.config
include $(SRCDIR)/files/ff/Makefile.config
include $(SRCDIR)/files/Makefile.config

#
# システムサービスに関する定義
#
SYSSVC_DIR := $(SYSSVC_DIR) $(SRCDIR)/syssvc $(SRCDIR)/files $(SRCDIR)/pdic/lpc23xx $(SRCDIR)/library
SYSSVC_ASMOBS := $(SYSSVC_ASMOBS)
SYSSVC_COBJS := $(SYSSVC_COBJS) banner.o syslog.o serial.o logtask.o ¥
log_output.o vasylog.o t_perror.o strerror.o ¥
mcicmd.o lpc23xx_mci.o lpc23xx_gpdma.o lpc23xx_rtc.o armv4.o $(CXXRTS)
SYSSVC_CFLAGS := $(SYSSVC_CFLAGS)
SYSSVC_LIBS := $(SYSSVC_LIBS)
INCLUDES := $(INCLUDES)
```

ミドルウェアを追加

デバイスドライバの
取り込み

デバイスドライバの
取り込み

2013/09/16

TOPPERSプロジェクト認定

176



sample1 解説: コンフィギュレーションファイル

- RTC,DMA,MCIのドライバ用のOSリソースの定義
- ファイルライブラリとFatFsのOSリソースの定義

```
INCLUDE("target_timer.cfg");
INCLUDE("syssvc/syslog.cfg");
INCLUDE("syssvc/banner.cfg");
INCLUDE("syssvc/serial.cfg");
INCLUDE("syssvc/logtask.cfg");
INCLUDE("pdic/lpc23xx/mci.cfg");
INCLUDE("pdic/lpc23xx/rtc.cfg");
INCLUDE("files/storagedevice.cfg");
INCLUDE("files/ff/fatfs.cfg");
INCLUDE("monitor/monitor.cfg");
#include "sample1.h"
CRE_TSK(MAIN_TASK, { TA_NULL|TA_ACT, 0, main_task,
                    MAIN_PRIORITY, STACK_SIZE, NULL });
```

2013/09/16

TOPPERSプロジェクト認定

177



sddisk.cの説明(ストレージファイルマネージャ)

- files/ff/sddiskio.cを開いて確認してください
- ストレージファイルマネージャとの関連設定を説明します
- _sd_initを初期化関数

```
static const StorageDeviceFunc_t fatfsSDeviceFunc = {
    sdc_card_sense,
    :
};
int _sd_init(void)
{
    StorageDevice_t *psdev;
    int result;
    result = SDMSSetupDevice(SDCARD_DEVNO, &psdev);
    assert(result == E_OK);
    psdev->pdevf = (StorageDeviceFunc_t *)&fatfsSDeviceFunc;
    psdev->pdevff = (StorageDeviceFileFunc_t *)&fatfsSDeviceFileFunc;
    psdev->sdev_sectsize = 512;
    psdev->sdev_port = SDCARD_PORTID;
    psdev->sdev_local[1] = mci_ini_port();
    assert(psdev->sdev_local[1] != NULL);
    psdev->sdev_attribute = SDEV_INSERTCHK | SDEV_CHECKREMOVE;
    return 1;
}
```

下位レベル関数としてSDメモリ
カードのセンス関数を定義

デバイス番号に対応し
たデバイス構造体を取
り出す

下位レベル関数テーブルを
デバイス構造体に設定

上位レベル関数テーブル
をデバイス構造体に設定

MCIデバイスをオープンしてMCIデバ
イス構造体を_sdev_local[1]にセット

2013/09/16

TOPPERSプロジェクト認定

178



sddisk.cの説明(FatFs用下位インターフェイス関数)

- FatFsの下位インターフェイス関数を実装している
- デバイス番号からストレージデバイス構造体を取り出し、
_sdev_local[1]からmciのデバイス構造体をキーとしてMCI
デバイスドライバにアクセスする

I/F関数	戻り値	下位インターフェイス関数の処理
disk_initialize	DISK_STATUS	ディスクドライブの初期化を行う。戻り値はSA_NODISK: メディアがない、SA_NOINIT: 初期化が終了していない、SA_PROTECT: メディアが書き込み禁止
disk_status	DISK_STATUS	ディスクドライバの状態を取得
disk_read	DRESULT	ディスクからの読み出し
disk_write	DRESULT	ディスクへの書き込み
disk_ioctl	DRESULT	IO制御関数
get_fattime	DWORD	日付、時間の取得、SDeviceHead._get_datetime関数から取得する

2013/09/16

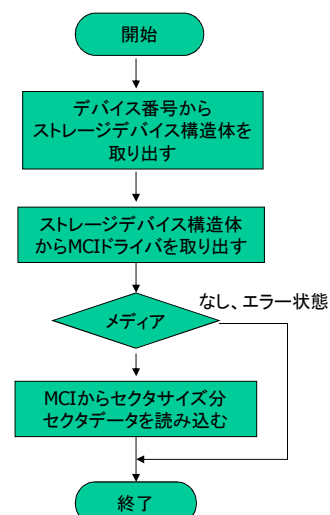
TOPPERSプロジェクト認定

179



演習:SDメモリカードREAD関数を作成

- sddisk.cのdisk_read関数を完成させて、FATファイルシステムが動作するようにしてください
- ストレージデバイス構造体からMCIドライバを取り出し、MCIデバイスドライバ関数を使ってSDメモリカードから指定のセクタ分READを行います
- READの手順はMCIテストプログラムを参照してください
- テストプログラムを実行して正常実行を確認してください



2013/09/16

TOPPERSプロジェクト認定

180



演習: SDメモリカードREAD関数を完成

- 以下のdisk_read関数を完成させてください

```

/*
 * ディスクからの読み込み
 * 引数
 *   BYTE Drive          物理ドライブ番号
 *   BYTE* Buffer         読み出しバッファへのポインタ
 *   DWORD SectorNumber  読み出し開始セクタ番号
 *   BYTE SectorCount    読み出しセクタ数
 */
DRESULT
disk_read(BYTE Drive, BYTE* Buffer, DWORD SectorNumber, BYTE SectorCount)
{
    StorageDevice_t *psdev = SDMGetStorageDevice(Drive);
    MCIPCB          *pmci;
    int             i;
/*
 * ここにセクタ読み込みプログラムを記述してください
 */
    return RES_OK;
}

```

FatFsへのファイル時間の管理

- FatFsにてファイルに日時を設定するには、下位インターフェイスのget_fattime関数に現在の日時(ローカル時間)を返す必要があります
- get_fattime関数の戻り値はDWORD型でビット単位で日時の設定を行います

get_fattimeの戻り値

フィールド	意味
bit[31:25]	年(0..127,0=1980)
bit[24:21]	月(1..12)
bit[20:16]	日(1..31)
bit[15:11]	時(0..23)
bit[10:5]	分(0..59)
bit[4:0]	秒/2(0..29)

現状のget_fatime関数の実装

- sddisk.c中のget_fatimeの実装は、現在の日時の取り出しをデバイスのヘッダ構造体中の_get_datetime関数を呼び出して行っている

```
DWORD get_fatime(void)
{
    SYSTIM system;
    DWORD fdate = ((1980-1980)<<25)+(1<<21)+(1<<16);
    DWORD ftime;
    if(SDeviceHead._get_datetime != 0)
        return (DWORD)SDeviceHead._get_datetime();
    else{
        get_tim(&system);
        systim = (systim+1000L) / 2000L;
        ftime = systim % 30;
        ftime += ((systim/30) % 60) << 5;
        ftime += ((systim/(60*30)) << 11);
        return (fdate+ftime);
    }
}
```

2013/09/16

TOPPERSプロジェクト認定

183



演習: FatFs用時間取得関数を作成

- sample1.cを改造してストレージデバイス構造体の時間取り出し関数をRTCを使って実装し、ファイルシステムで正しくファイル時間が設定できるようにしてください
- テストプログラムの最初でSDeviceHead._get_datetimeに設定するFAT用の日時取り出し関数get_fat_time関数を完成させてください
- 確認はファイルテストを実行しタスクモニタのディレクトリ表示機能で時間の確認をしてください

```
struct tm2 timedate;
:
timedate.tm_year = 2011-1970;
timedate.im_isdst = 0;
mktime((struct tm *)&timedate);
rtc_set_time(&timedate);
rtc_start(NULL);
SDeviceHead._get_datetime = get_fat_time;
```

2013/09/16

TOPPERSプロジェクト認定

184



演習：FatFs用時間取得関数の完成

- FILE/sample.cのget_fat_time関数を完成させてください
- FATファイルシステムで使用する時刻は、tm構造体の時刻をFatFsへのファイル時間管理のDWORD値に変換します

```
static int get_fat_time(void)
{
    int   reqtime = 0x00210000;
    /*
    ここに時刻を取り出すプログラムを記述してください
    */

    return reqtime;
}
```

まとめ

基礎3実習1日目のまとめ

- Cygwinを用いたGCCのARM開発環境の構築
- RTCドライバの作成
- MCIドライバの作成
- SDカード用ファイルシステムのドライバの作成
- ファイルライブラリを用いたファイル検証プログラムの確認
 - ファイルライブラリを用いることで、ファイルシステムに依存しないファイルの検証を行うことができます
 - また、アプリケーションもファイルシステムの依存性を排除できます

参考文献

- LPC23XX User manual(UM10211)
- LPC2388 Data Book
- MCI/SDカードの章ではInterface誌2009年6月号の赤松武志さんの記事を参照しました
- FatFsの章ではInterface誌2010年9月号の赤松武志さんの記事を参照しました

謝辞

本教材の開発において、NPO法人組込みソフトウェア管理者・技術者育成研究会およびNPO法人TOPPERSプロジェクトの作成した以下の教材を参考としました。

- OpenSESSAME Seminar組込みソフトウェア技術者・管理者向けセミナー初級者向けテキスト第5版
- TOPPERS初級実装セミナー教材

両団体および上記教材の作者の皆様へ感謝します。

本教材の開発において、NEXCESS初級コースおよび指導者養成コースの受講者の皆様のコメントを参考としました。両コースの受講者の皆様へ感謝します。

著者リスト

- プラットフォームの構築
 - 竹内 良輔((株)リコー)
 - 福井徹((株)デンソークリエイト)
- その他
 - 竹内 良輔((株)リコー)

教材に関するご意見は、
secretariat@toppers.jp
までお寄せください。