

TOPPERS基礎実装セミナー

(LPC2388版:基本)

基礎編:2日目

TOPPERSプロジェクト
教育ワーキング・グループ

2014/06/13

TOPPERSプロジェクト認定

1



本教材の利用条件

NEXCESS基礎コース01 組込みソフトウェア開発技術の基礎

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

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

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

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

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

2014/06/13

TOPPERSプロジェクト認定

2



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版:基本)2日目＞

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

Copyright (C) 2006-2007 by 本田晋也 名古屋大学

Copyright (C) 2007-2013 by 竹内良輔 (株)リコー

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

(1) 本ドキュメントを利用する場合には、上記の著作権表示、この利用条件および以下の無保証規定が、そのままの形でドキュメント中に含まれていること。

(2) 本ドキュメントを改変する場合には、ドキュメントを改変した旨の記述を、改変後のドキュメント中に含めること。ただし、改変後のドキュメントがTOPPERSプロジェクト指定の開発成果物である場合には、この限りではない。

(3) 本ドキュメントの利用により直接的または間接的に生じるいかなる損害からも、上記著作権者およびTOPPERSプロジェクトを免責すること。また、本ドキュメントのユーザまたはエンドユーザからのいかなる理由に基づく請求からも、上記著作権者およびTOPPERSプロジェクトを免責すること。

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

2. 本ドキュメントに関するご意見・ご提言・ご感想・ご質問等がありましたら、TOPPERSプロジェクト事務局までE-Mailにてご連絡ください。

3. 本ドキュメントの内容は、内容の改善や適正化の目的で予告無く改定することがあります。

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

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

本ドキュメント中の商品名及び商標名は、各社の商標または登録商標です。

2014/06/13

TOPPERSプロジェクト認定

3



スケジュール

■ 2日目

- | | |
|------------------------|-------|
| 1. メモリマップドレジスタの操作方法の確認 | 0.5時間 |
| 2. ポーリングプログラム | 3.0時間 |
| 3. 割込みプログラム | 2.0時間 |
| 4. まとめ | 0.5時間 |

2014/06/13

TOPPERSプロジェクト認定

4



概要

- マイコンボードを用いて組込みプログラミングの基礎を学ぶ
- メモリマップドレジスタの操作方法の確認
 - デバッガを用いてデバイスレジスタを操作する
- ポーリングプログラミング
 - LED, スイッチ, タイマ, シリアルI/Oを操作するプログラムの作成
- 割り込みプログラミング
 - スイッチ, タイマ, シリアルI/Oからの事象を割り込みにより扱うプログラムの作成
- カップラーメンタイマの作成
 - スイッチ, タイマによりカップラーメンタイマを実現

メモリマップドレジスタの操作方法の確認

1. [ROMモニタの操作](#)
2. [LEDの接続](#)
3. [LEDの操作](#)

メモリマップドレジスタの操作方法の確認：目的

LEDが接続されているプログラマブル入出力ポートを例に
メモリマップドレジスタの操作方法を学ぶ

- ROMモニタを用いると任意のアドレスのメモリの読み書きが可能
- ARM7の周辺機能の制御レジスタはメモリ空間に配置されている(メモリマップドレジスタ)
 - ROMモニタからLEDが接続されているポートを操作可能
- ROMモニタの E(L)コマンドによるメモリの読み書き

```

ARM7-LPC2388 Monitor v1.01
Copyright (C) 1999-2004 ESE Technology, Inc.
Copyright (C) 2000- ROMON CORP/ARM, LTD.
0: 40000000
40000000: 17136fcd: 12345678
40000004: c5803327..
0: 40000000
40000000: 12345678 c5803327 f754eeb4 3a821011
40000010: c481c5c0 70391f56 f1fc7f41 2b4475e3
40000020: ee5f745e 438100e4 c99255e0 a4251188
40000030: 7a21f5f4 343944e7 39f890c9 ca805cc0
40000040: f7136fcd c5803327 f754eeb4 3a821011
40000050: c481c5c0 70391f56 f1fc7f41 2b4475e3
40000060: ee5f745e 438100e4 c99255e0 a4251188
40000070: 7a21f5f4 343944e7 39f890c9 ca805cc0
40000080: 4381c5c0 e7ecce04 c50a0a4d 41d477ee
40000090: 5e544ec1 c28f2877 441804d1 c2f46033
400000a0: 91e42e57 87411b78 07188aeb f18fa88c
400000b0: 5726d061 ca3c90bc b7a0a059 63374588
400000c0: 4381c5c0 e7ecce04 c50a0a4d 41d477ee
400000d0: 1e5e4e09 8a2f2277 442e0271 c2f46033
400000e0: 97bf3b77 67890b7a 4f583afa f18fa88c
400000f0: 5779e161 da3490bd b7f0f059 63374582
0:
  
```

2014/06/13

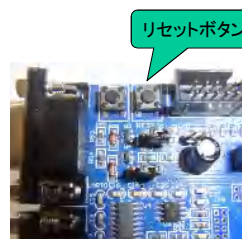
TOPPERSプロジェクト認定

7



ROMモニタの操作方法

- 開発環境の確認編の“ボードのセットアップ”に従ってマイコンボードとPCを接続する
- ROMモニタを起動する前は常に
 - J8、J9が書き込み側ではないか？
 - TeraTarmを立ち上げたか？
 - ボードの電源が入っているか？
 を確認して、一度リセットボタンを押す
- コマンドを使ってモニタを操作する
 - ? Enterで、使用可能なコマンドの一覧を表示する



2014/06/13

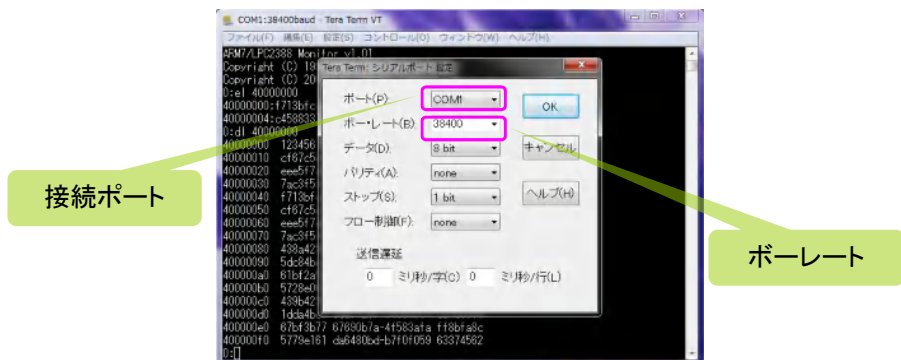
TOPPERSプロジェクト認定

8



ROMモニタの操作方法 : TeraTermシリアルポート

- ROMモニタはUARTを使用して、ボードに対して種々の設定や確認を行うことができます
- シリアルポート設定で設定を行います



2014/06/13

TOPPERSプロジェクト認定

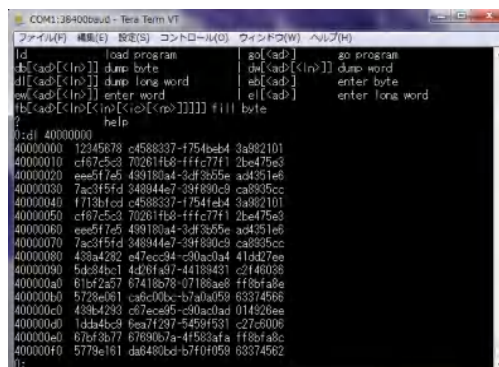
9



デバッガの操作方法 : 読み出し

- dl コマンドで4バイト単位のメモリDUMPができます
- db コマンドは1バイト単位、dw コマンドは2バイト単位

dl <address> return



2014/06/13

TOPPERSプロジェクト認定

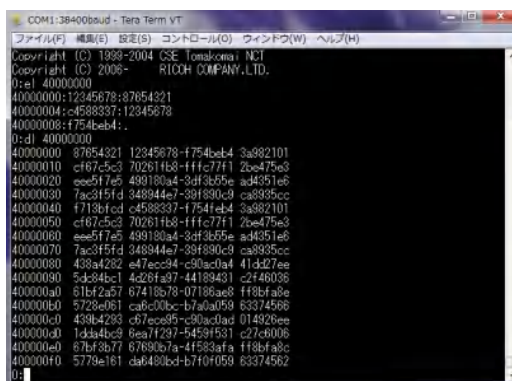
10



デバッガの操作方法：書き込み

- el コマンドにて4バイト単位でのデータ書き込みができます
- el アドレスで、データ入力モードに移行し、データを入力すると設定アドレスにデータを書き込みます
- ‘.’ returnにて、データ入力モードを終了します

RAM領域の
0x40000000番地と
0x40000004番地に
0x87654321と
0x12345678を書き込む



2014/06/13

TOPPERSプロジェクト認定

11



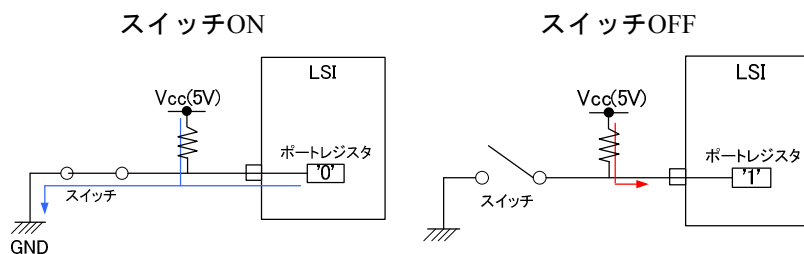
周辺デバイス：プルアップとプルダウン

- デジタル回路は電圧の高低(HiとLo)で'0'と'1'を判断する
例)電源電圧が5vの場合, Hiが4~5v, Loは0~1v
- LSIへの入力はHiかLoの電圧になるようにしなければならない

プルアップとプルダウン

- GPIO入力ポートとしてスイッチ等を接続する場合, 入力を電氣的に安定させるための回路構成

プルアップの例



2014/06/13

TOPPERSプロジェクト認定

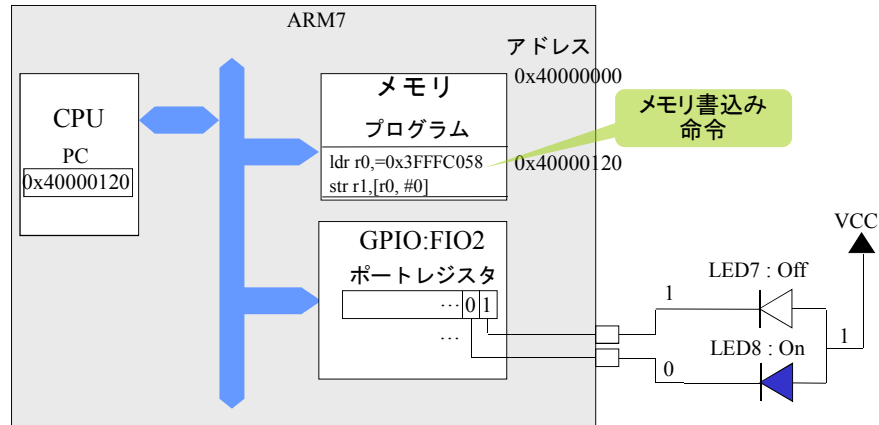
12



周辺デバイス : プロセッサとLEDとの接続の関係

LPC2388の例

–ポートレジスタの値が0でLEDが点灯する



2014/06/13

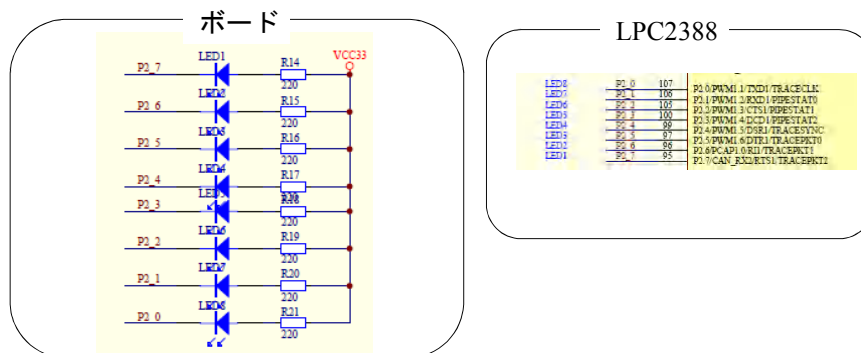
TOPPERSプロジェクト認定

13



LEDの接続 : マニュアルによる確認

- LEDはプルアップされている
- 直接、マイコンのプログラマブル入出力ポートに接続されている
 - マイコンから”0”を出力すると点灯する



2014/06/13

TOPPERSプロジェクト認定

14



LEDの接続：レジスタ構成

- LEDはポートFIO2に接続
 - ポート方向レジスタ：FIO2DIR 0x3FFFC040
 - ポートセットレジスタ：FIO2SET 0x3FFFC058
 - ポートクリアレジスタ：FIO2CLR 0x3FFFC05C
 - ポートレジスタをオン(1)にするためにFIO2SETの対応ビットを1
 - ポートレジスタをオフ(0)にするためにFIO2CLRの対応ビットを1にしなければならない

Generic Name	Description	Access	Reset value ¹⁾	PORTn Register Address & Name
FIO2DIR	Fast GPIO Port Direction control register. This register individually controls the direction of each port pin.	R/W	0x0	FIO0DIR - 0x3FFF C000 FIO1DIR - 0x3FFF C020 FIO2DIR - 0x3FFF C040 FIO3DIR - 0x3FFF C060 FIO4DIR - 0x3FFF C080
FIO2SET	Fast Port Output Set register using FIO2MASK. This register controls the state of output pins. Writing 1s produces highs at the corresponding port pins. Writing 0s has no effect. Reading this register returns the current contents of the port output register. Only bits enabled by 0 in FIO2MASK can be altered.	R/W	0x0	FIO0SET - 0x3FFF C018 FIO1SET - 0x3FFF C038 FIO2SET - 0x3FFF C058 FIO3SET - 0x3FFF C078 FIO4SET - 0x3FFF C098
FIO2CLR	Fast Port Output Clear register using FIO2MASK0. This register controls the state of output pins. Writing 1s produces lows at the corresponding port pins. Writing 0s has no effect. Only bits enabled by 0 in FIO2MASK0 can be altered.	WO	0x0	FIO0CLR - 0x3FFF C01C FIO1CLR - 0x3FFF C03C FIO2CLR - 0x3FFF C05C FIO3CLR - 0x3FFF C07C FIO4CLR - 0x3FFF C09C

NXPセミコンダクタ
User manual Chapter10参照

2014/06/13

TOPPERSプロジェクト認定

15



LEDの接続：設定値

レジスタを操作して全てのLEDを点灯させる

- ポートを構成するレジスタとLEDとの接続を整理すると全てのLEDを点灯させるには以下の設定が必要
 - ポート方向レジスタ (0x3FFFC040) の7,6,5,4,3,2,1,0ビットに‘1’を書き込み出力モードにする
 - ポートレジスタ (0x3FFFC05C) の7,6,5,4,3,2,1,0ビットに‘1’を書き込みポートの出力を‘0’にする
- レジスタへ書き込む値
 - 0x3FFFC040 : 0x000000FF ("0000 0000 1111 1111")
 - 0x3FFFC05C : 0x000000FF ("0000 0000 1111 1111")

2014/06/13

TOPPERSプロジェクト認定

16



LEDの操作 : 任意のLEDの点灯

- 次のパターンでLEDを点灯させるためのポートレジスタへ書き込む値を考え, 実際に確認せよ

1. LED1:OFF LED2:OFF LED3:OFF LED4:OFF

2. LED1:ON LED2:ON LED3:OFF LED4:OFF

3. LED1:OFF LED2:ON LED3:ON LED4:OFF

4. LED1:OFF LED2:OFF LED3:ON LED4:ON

5. LED1:OFF LED2:ON LED3:OFF LED4:ON

2014/06/13

TOPPERSプロジェクト認定

17



LEDの操作 : 任意のLEDの点灯

1. LED1:OFF LED2:OFF LED3:OFF LED4:OFF

- SET:0xF0 : "1111 0000"

2. LED1:ON LED2:ON LED3:OFF LED4:OFF

- CLR:0xC0 : "1100 0000" SET:0x30 : "0011 0000"

3. LED1:OFF LED2:ON LED3:ON LED4:OFF

- CLR:0x60 : "0110 0000" SET:0x90 : "1001 0000"

4. LED1:OFF LED2:OFF LED3:ON LED4:ON

- CLR:0x30 : "0011 0000" SET:0xC0 : "1100 0000"

5. LED1:OFF LED2:ON LED3:OFF LED4:ON

- CLR:0x50 : "0101 0000" SET:0xA0 : "1010 0000"

2014/06/13

TOPPERSプロジェクト認定

18



ポーリングプログラム

1. LEDプログラム

・プロジェクトの作成方法

2. スイッチプログラム
3. タイマプログラム
4. シリアルI/Oプログラム

ポーリングプログラミング：目的

周辺デバイス进行操作するプログラムの書き方を学ぶ

- 初期設定, データの入出力, 事象の待ち方
 - スイッチのONやタイマのタイムアウトなどの外部事象はポーリングによって扱う
- プログラミング対象の周辺デバイス
 - LED(プロマブル入出力ポート出力)
 - スイッチ(プロマブル入出力ポート入力)
 - タイマ
 - シリアルI/O

ポーリングプログラミング : プログラムファイル

- プログラムファイルの置き場所
 - 教材ディレクトリ/program1
- プログラム一覧
 - led_shift : LEDシフト点灯プログラム
 - led_count : LEDカウント点灯プログラム
 - switch_push : PUSHスイッチプログラム
 - timer : タイマプログラム
 - uart : シリアルI/Oプログラム

LEDプログラム : led_shift 概要

- 次のパターンでLEDを点灯させる
 - LED全てOFF → LED1 ON → LED2 ON → LED3 ON → LED4 ON → LED全てOFF
- → ●○○○ → ○●○○ → ○○●○ → ○○○● → ●○○○
- 学習内容
 - プログラムの全体像の理解
 - ビット操作(セット, クリア)
 - デバイスレジスタ操作(様々な記法)
ポートからの出力
 - スタートアップルーチン, セクション
 - 構成ファイル
 - led_shift.c
 - start.S

led_shift : メイン関数

– 後述するスタートアップルーチンから呼び出される

LED接続ポート
の初期化

ウェイト

処理内容は
同一であるが
記法が異なる

```
void
main(void){
    led_init();
    for (;;) {
        led_out_no_macro(0x00);
        busy_wait();
        led_out_macro1(0x08);
        busy_wait();
        led_out_macro2(0x04);
        busy_wait();
        led_out_struct(0x02);
        busy_wait();
        led_out_sil(0x01);
        busy_wait();
    }
}
```

LED全消灯

LED1点灯

LED2点灯

LED3点灯

LED4点灯

2014/06/13

TOPPERSプロジェクト認定

23



led_shift : ポートレジスタへの書き込み

- LED点灯・消灯制御
 - FIO2のポートレジスタ(0x3FFFC058,0x3FFFC05C)への書き込み
 - 特定アドレスの読み書きが必要
- アセンブリコード

```
ldr    r0, =0x3fffc058
ldr    r1, =0x000000ff
書き込み : str    r1, [r0, #0]
読み込み : ldr    r2, [r0, #0]
```

- C言語
 - ポインタを使用
 - アドレスをポインタ変数にキャスト

```
書き込み : *((unsigned long *) (0x3fffc058)) = 0x000000ff;
読み込み : tmp = *((unsigned long *) (0x3fffc058));
```

2014/06/13

TOPPERSプロジェクト認定

24



led_shift : volatile修飾子

- volatile
 - ポインタを用いてレジスタをアクセスする場合は、最適化を抑制するため、常にvolatile指定をつける必要がある

```
書き込み : *((volatile unsigned long *) (0x3ffc058)) = 0x000000ff;
読み込み : tmp = *((volatile unsigned long *) (0x3ffc058));
```

- volatile修飾子の場所
 - 以下の記述でもコンパイルは通るが、最適化は抑制できないので注意が必要である(ポインタ変数がvolatile)
 - ポインタ変数が指し示す先をvolatileにしなければならない

```
tmp = *((unsigned long * volatile) (0x3ffc058));
```

2014/06/13

TOPPERSプロジェクト認定

25



led_shift : ポートレジスタ書き込み関数

- LED点灯・消灯制御
 - FIO2のポートレジスタ(0x3ffc058)への書き込み
 - アクセスサイズが4byteなので、unsigned long へのポインタへキャストして書き込む

```
void led_out(unsigned char led_data){
    unsigned long reg1 = ~led_data;
    unsigned long reg2 = led_data;
    *((volatile unsigned long *) 0x3ffc058) = reg1; /* 書き込み*/
    *((volatile unsigned long *) 0x3ffc05c) = reg2;
}
```

今回はこれで十分だが、
再利用性を考えると不十分なコード?

- 将来的にFIO2のLEDが接続されているビット以外のビットに他の回路が接続された場合、このコードを実行する発生する不具合は?

2014/06/13

TOPPERSプロジェクト認定

26



led_shift : 変更値以外の保存

- 再利用性を考えると操作対象のビット以外は変更するべきではない
 - 操作対象のビット以外は変更前の値を書き込む
- 変更前の値をどこから読むか?
 - ポートレジスタから?

通常のMapped IOの場合、ポートレジスタに直接データを書き込む、この場合、周辺回路のレジスタはメモリマッピングされているが、メモリとは異なり書き込んだ値が読み込めるとは限らないため注意が必要

LPC23xxのポートレジスタの仕様(users manual chapter 10から引用)

- FIOSET: Reading this register returns the current contents of the port output register. Only bits enabled by 0 in FIOMASK can be altered.

FIO2SETレジスタを読めば、FIO2MASKがイネーブルの場合、ポートの設定内容が読み込めます。

FIOのFIOSETレジスタは現在の出力値を読み込める仕様

- ポートレジスタが値を保持しない場合はグローバル変数に値を保持する

2014/06/13

TOPPERSプロジェクト認定

27



led_shift : 特定ビットのみの変更

- 変更前の値はポートレジスタから取得可能
- 取得した値のうち、LEDが接続されているビット7,6,5,4,3,2,1,0にのみ引数 led_data の値を反映させる
- 例えば..
 - ポートレジスタの値(LED3:ON): 0x0120 (“0000 0001 0010 0000”)
 - 引数led_data(LED4:ON) : 0x0010 (“0000 0000 0001 0000”)
 - 書き込みたい値は : 0x0110 (“0000 0001 0001 0000”)
- プログラムによる0x0110の生成手順
 1. ポートレジスタの値のビット7-0をクリア
(0x0100 (“0000 0001 0000 0000”))
 2. led_dataのビット7-0のみを取り出す
 3. 1.と2.で生成した値の論理和を生成
(“0000 0001 0000 0000”|“0000 0000 0001 0000”
= “0000 0001 0001 0000”)
- 特定ビットのクリアや特定ビットの取り出しが必要
 ➡ ビット演算により実現

2014/06/13

TOPPERSプロジェクト認定

28



led_shift : led_outの変更

- ポートレジスタのビット7-0以外は元の値を保持
LPC2388の場合、SET、CLRが別ポートであり、0を書き込んだビットは変化しないため、ビット演算の必要はない

```
void
led_out_no_macro(unsigned char led_data){
    unsigned long reg1, reg2;

    reg1 = ~led_data & 0xff;
    reg2 = led_data & 0xff;

    *((volatile unsigned long *)0x3fffc058) = reg1;
    *((volatile unsigned long *)0x3fffc05c) = reg2;
}
```

1. オン(1)にするビットを取り出す

2. オフ(0)にするビットを取り出す

3. FIOSETにオンにするデータを書き込む

4. FIOCLRにオフにするデータを書き込む

2014/06/13

TOPPERSプロジェクト認定

29



led_shift : マクロによる記述1

定数はマクロ(記号定数)で記述する

- レジスタのアドレスや接続ビットの値を直接プログラム中に記述すると可読性が悪くなる(マジックナンバー)
- レジスタ値や接続ビットの変更に対応が可能
- FIO2レジスタのマクロ定義

```
#define TADR_FIO2_SET    0x3FFFC058
#define TADR_FIO2_CLR    0x3FFFC05C
```

- FIO2のLEDの接続ビット定義

```
#define LED1    0x80    /* LED1ビット定義 */
#define LED2    0x40    /* LED2ビット定義 */
#define LED3    0x20    /* LED3ビット定義 */
#define LED4    0x10    /* LED4ビット定義 */
#define LED5    0x08    /* LED5ビット定義 */
#define LED6    0x04    /* LED6ビット定義 */
#define LED7    0x02    /* LED7ビット定義 */
#define LED8    0x01    /* LED8ビット定義 */
#define LED_MASK (LED1 | LED2 | LED3 | LED4 | LED5 | LED6 | LED7 | LED8)
```

2014/06/13

TOPPERSプロジェクト認定

30



led_shift : マクロによるled_outの書き換え1

- 何をしているプログラムなのか分かりやすくなる

```
void
led_out_macro1(unsigned char led_data){
    unsigned long reg1, reg2;

    reg1 = ~led_data & LED_MASK;
    reg2 = led_reg & LED_MASK;

    *((volatile unsigned long *)TADR_FIO2_SET) = reg1;
    *((volatile unsigned long *)TADR_FIO2_CLR) = reg2;
}
```

2014/06/13

TOPPERSプロジェクト認定

31



led_shift : マクロによる記述2

デバイスレジスタでは、ポインタへのキャストの部分も含めてマクロ化する場合が多い

- 型の指定間違いを防ぐ
 - デバイスレジスタにはアクセスサイズがある
- 記述が簡素になる
- マクロ化の方法や命名規則はプログラムやプロジェクト単位で一定化しておく
- FIO2レジスタ

```
#define TREG_FIO2_SET ((volatile unsigned long *)TADR_FIO2_SET)
#define TREG_FIO2_CLR ((volatile unsigned long *)TADR_FIO2_CLR)
```

- ポインタ部までマクロ化(プログラム例はない)

```
#define TREG_FIO2_SET *((volatile unsigned long *)TADR_FIO2_SET)
#define TREG_FIO2_CLR *((volatile unsigned long *)TADR_FIO2_CLR)
```

2014/06/13

TOPPERSプロジェクト認定

32



led_shift : マクロによるled_outの書き換え2

```
void
led_out_macro2(unsigned char led_data){
    unsigned long reg1, reg2;

    reg1 = ~led_data & LED_MASK;
    reg2 = led_data & ~LED_MASK;

    *TREG_FIO2_SET = reg1;
    *TREG_FIO2_CLR = reg2;
}
```

- 1行での記述も可能

```
*TREG_FIO2_SET = ~led_data & LED_MASK;
```

2014/06/13

TOPPERSプロジェクト認定

33



led_shift : 構造体による記述

構造体を用いてデバイスレジスタのアドレスを指定する

- ビット単位の読み書きが容易に記述可能
 - コンパイラ依存であり, 利用できないコンパイラもある
- 記述方法もコンパイラによって異なる

```
/* ビット構造体 */
struct bit_def {
    unsigned char b0:1;
    unsigned char b1:1;
    unsigned char b2:1;
    unsigned char b3:1;
    unsigned char b4:1;
    unsigned char b5:1;
    unsigned char b6:1;
    unsigned char b7:1;
};
```

ビット
フィールド

```
/* ワード構造体 */
struct byte_def{
    struct bit_def bit0;
    struct bit_def bit1;
    struct bit_def bit2;
    struct bit_def bit3;
};
```

```
union word_def{
    struct byte_def byte;
    unsigned long word;
};
```

2014/06/13

TOPPERSプロジェクト認定

34



led_shift : 構造体による書き換え

バイトアクセス
による記述

ビットアクセス
による記述

```
void
led_out_struct(unsigned char led_data){
    union word_def reg1, reg2;

    reg1.word = 0;
    reg2.word = 0;
    reg1.byte.bit0.b7 = ((LED1 & led_data) == 0)? 1 : 0;
    reg2.byte.bit0.b7 = ((LED1 & led_data) != 0)? 1 : 0;
    reg1.byte.bit0.b6 = ((LED2 & led_data) == 0)? 1 : 0;
    reg2.byte.bit0.b6 = ((LED2 & led_data) != 0)? 1 : 0;
    reg1.byte.bit0.b5 = ((LED3 & led_data) == 0)? 1 : 0;
    :
    reg1.byte.bit0.b0 = ((LED8 & led_data) == 0)? 1 : 0;
    reg2.byte.bit0.b0 = ((LED8 & led_data) != 0)? 1 : 0;
    *TREG_FIO2_SET = reg1.word;
    *TREG_FIO2_CLR = reg2.word;
}
```

2014/06/13

TOPPERSプロジェクト認定

35



led_shift : デバイスアクセス関数による記述

デバイスへのアクセスを専用の関数により実現

- プログラムのハードウェア依存性が弱まる
- シミュレーション環境への対応が容易

例) デバイスドライバ設計ガイドラインのアクセスインタフェース

– バイトデータの読み込みと書き込み

```
unsigned long
sil_rew_mem(unsigned int addr){
    return *((volatile unsigned long *)addr);
}
```

```
void
sil_wrw_mem(unsigned int addr, unsigned long bdata){
    *((volatile unsigned long *)addr) = bdata;
}
```

2014/06/13

TOPPERSプロジェクト認定

36



led_shift : デバイスアクセス関数による書き換え

- アクセスサイズによって呼び出す関数が異なる

```
void
led_out_sil(unsigned char led_data){
    unsigned long reg1, reg2;

    reg1 = ~led_data & LED_MASK;
    reg2 = led_data & LED_MASK;

    sil_wrw_mem(TADR_FIO2_SET, reg1);
    sil_wrw_mem(TADR_FIO2_CLR, reg2);
}
```

2014/06/13

TOPPERSプロジェクト認定

37



led_shift : 初期化関数 led_init()

- ポートを出力方向に設定し, 全LEDを消灯
- LED接続ビット以外を変更しないようにする

```
void
led_init(void){
    unsigned long reg;
    /* 方向レジスタ出力設定 */
    reg = *((volatile unsigned long *)TADR_FIO2_DIR);
    reg = reg | LED_MASK;
    *((volatile unsigned char *)TADR_FIO2_DIR) = reg;
    /* 初期状態は消灯 */
    reg = LED_MASK;
    *((volatile unsigned long *)TADR_FIO2_SET) = reg;
}
```

7-0ビット
をセット

7-0ビット
をセット

2014/06/13

TOPPERSプロジェクト認定

38



led_shift : ビジーウェイト関数 busy_wait()

- forループにより任意時間待ちを生成
- ループ回数は1msをベース
- ループ変数はlong型
 - 1msをベースに32ビット長の時間に対応するため
 - 引数1に対して、1ナノ秒のソフト待ちを行うsil_dly_nseを用意

```
#define DELAY_LOOP (250*1000)

void
busy_wait(void){
    volatile long i;
    for(i = 0; i < DELAY_LOOP; i++)
        sil_dly_nse(1000);
}
```

Long(32bit)
指定

led_shift : start.S

アセンブリ言語で記述されたファイル

- 各プログラムで共通に使用できる
- セクションの配置
 - ROM実行の場合、ROM上に配置
 - RAM実行の場合、RAM上に配置
- ベクタテーブル
 - ROM実行の場合、すべてのプログラムを実行
 - RAM実行の場合、IRQ以外は使用しない
 - スタートアップの実行はROMモニタからの起動
- スタートアップルーチン
 - セクションの初期化
 - _main関数の呼び出し

led_shift : start.S セクションの配置 (ROM実行)

- リンクファイル: lpc2388_rom.ldで指定

```
; ----- RAM 領域 -----
ORG      40000000
SECTION  .data
SECTION  .bss
```

```
; ----- ROM 領域 -----
ORG      0x00000000
SECTION  .vector
SECTION  .text
SECTION  .data
```

セクション

- .vector : 割込み、エクセプション実行、書き換え不可
- .text : プログラム、書き換え不可データ
- .data : 初期値ありデータ
ROM化される場合、ROM上の初期値LOADADDR(.data)
RAM上の参照領域ADDR(.data)に分けられる
スタートアップルーチンでROM上のデータをRAM上にコピーする
- .bss : 初期値なしデータ
スタートアップルーチンでゼロに初期化

2014/06/13

TOPPERSプロジェクト認定

41



led_shift : start.S セクションの配置 (RAM実行)

- リンクファイル: lpc2388_ram.ldで指定

```
; ----- RAM 領域 -----
ORG      40000000
SECTION  .vector
SECTION  .text
SECTION  .data
SECTION  .bss
```

セクション

- すべてのデータはROMモニタのldコマンドでダウンロード
- .vector : 割込み、エクセプション実行、書き換え不可
- .text : プログラム、書き換え不可データ
- .data : 初期値ありデータ
RAM上にダウンロードされるためコピーに必要がない
- .bss : 初期値なしデータ
ROMモニタのgoコマンド実行後、ゼロに初期化

2014/06/13

TOPPERSプロジェクト認定

42



led_shift : start.S 固定ベクタ

- ARM7はリセット後、0x00000000から実行を始める
- ROM実行の場合、0x00000000にリセット処理(start)へのジャンプ命令を記載する
- RAM実行の場合、ROMモニタのgoコマンドが0x40000000の命令を実行する。ここにリセット処理(start)へのジャンプ命令を記載する

```
.section .vector,"a"
.code 32
.align 0
.global vector_table
vector_table:
    .ldr pc, reset_vector
    .ldr pc, undef_vector
    .ldr pc, swi_vector
    .ldr pc, prefetch_vector
    .ldr pc, data_abort_vector
    .ldr pc, reserved_vector
    .ldr pc, irq_vector
    .ldr pc, freq_vector
reset_vector:
    .long start
```

RESET時の実行アドレス

関数startのアドレス

2014/06/13

TOPPERSプロジェクト認定

43



led_shift : start.a30 スタートアップルーチン

- アセンブリ言語で記述
 - 設定しているレジスタをマニュアルでチェック
 - C言語と同様に定数はマクロ化

```
start:
    mov    r1, #(0x13|0xc0)    /* スーパーバイザモードへ移行 */
    msr    cpsr, r1
    ldr    r3, =STACKTOP      /* スーパーバイザモードのスタックポインタをセット */
    mov    sp, r3
    mov    r11, #0             /* armモードフレームポインタをゼロ */
    mov    r7, #0              /* thumbモードフレームポインタをゼロ */
    #if ROM_EXEC != 0
    ldr    r0, #TADR_SCB_BASE /* クロック設定 */
    mov    r1, #0xAA
    mov    r2, #0x55
    ldr    r3, =SCS_Val        /* main oscillatorをイネーブルに */
    str    r3, [r0, #TOFF_SCB_SCS]
    osc_loop:                  /* OSCの初期化待ち */
    ldr    r3, [r0, #TOFF_SCB_SCS]
    ands   r3, r3, #OSCSSTAT   /* OSCSTATの確認 */
    beq    osc_loop
```

2014/06/13

TOPPERSプロジェクト認定

44



led_shift : start.a30 セクションの初期化

```

/*----- BSS領域の初期化 -----*/
start_1:  ldr    r1, =__bss_start
          ldr    r2, =__bss_end
          cmp    r1, r2
          bhs    start_3
          mov    r0, #0
start_2:  str    r0, [r1], #4
          cmp    r1, r2
          blo    start_2
/*----- DATA領域の初期化 -----*/
start_3:  ldr    r1, =__idata_start
          ldr    r3, =__idata_end
          cmp    r1, r3
          bhs    start_5
          ldr    r2, =__data_start
start_4:  ldr    r0, [r1], #4
          str    r0, [r2], #4
          cmp    r1, r3
          blo    start_4

```

bss_startからbss_end
までをゼロクリア

__idata_startから__idata_end
までを__data_startにコピー

2014/06/13

TOPPERSプロジェクト認定

45



led_shift : start.S main関数の呼び出し

- ハードウェアの初期化, data, bssの初期化が終了後, _main関数を呼び出す

```

/****** メインルーチンへ *****/
start_5:  ldr    r0, =_main
          mov    lr, pc
          bx     r0          /* --> _main() */
start_6:  b      start_6

```

- main()という名前で関数を作成するとCのランタイムの初期化処理を自動的に呼び出すルーチンを関数内に入れるコンパイラもあるので注意
- GCCでは、それをさけるために、mainを_mainとしている

2014/06/13

TOPPERSプロジェクト認定

46



ポーリングプログラム

1. LEDプログラム
 - ・プロジェクトの作成方法
2. スイッチプログラム
3. タイマプログラム
4. シリアルI/Oプログラム

2014/06/13

TOPPERSプロジェクト認定

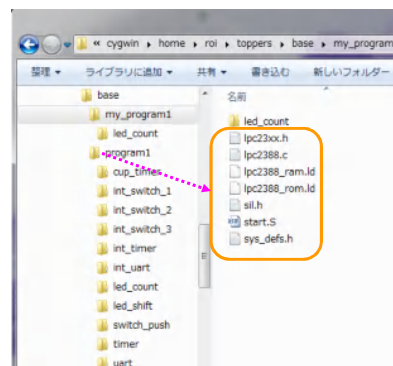
47



プロジェクトの作成方法

新規プロジェクトの作成方法を学習

- プロジェクトのためのディレクトリ (my_program1/led_count) を作成
 - 作成場所に制約はない
- 教材ディレクトリ /program1/led_shift にある以下のファイルをmy_program1フォルダにコピーする
 - lpc23xx.h
 - lpc2388.c
 - lpc2388_ram.ld
 - lpc2388_rom.ld
 - sil.h
 - start.S
 - sys_defs.h



2014/06/13

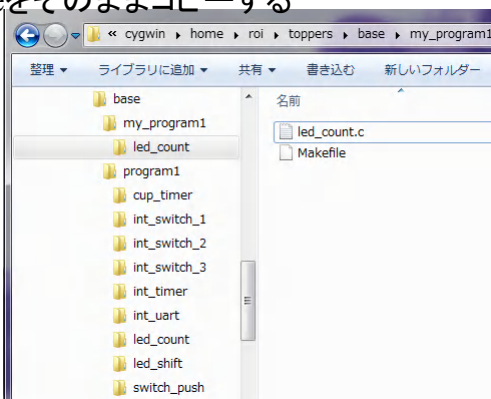
TOPPERSプロジェクト認定

48



プロジェクトの作成方法：C言語プログラム

- コンパイルするC言語プログラムを用意する
- program1/led_shift/led_shift.cの内容をコピーして、led_countのディレクトリに led_count.c として作成する
- Makefileをそのままコピーする



2014/06/13

TOPPERSプロジェクト認定

49



Makefileの書き換え

- コピーしたMakefileをエディタで開く
- 20行目のOBJNAMEをled_shiftからled_countに変更する

```
#
# オブジェクトファイル名の定義
#
OBJNAME = led_count
#ifdef OBJEXT
    OBJFILE = $(OBJNAME).$(OBJEXT)
#else
    OBJFILE = $(OBJNAME)
```

led_shiftからled_countに変更

2014/06/13

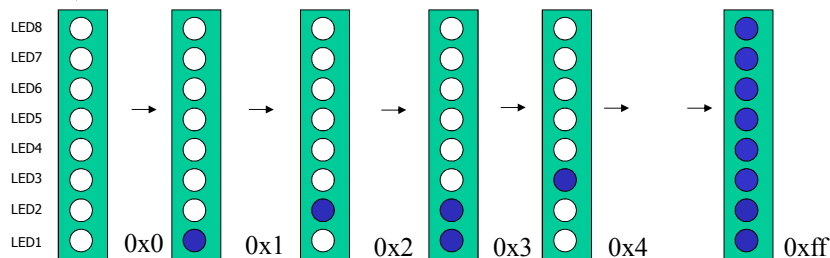
TOPPERSプロジェクト認定

50



LEDプログラム : led_count 概要

- LEDを2進数の表示器とみなし、一定時間毎に0x00から0xffまでを表示する(カウントアップ)



- 学習内容
 - プロジェクトの作成方法
 - 出力値のエンコード方法
- 作成方法
 - led_shiftをベースに作成

2014/06/13

TOPPERSプロジェクト認定

51



led_count : メイン関数

- 無限ループで一定時間毎に変数を引数にLEDの表示を更新する関数を呼び出し、変数をインクリメントする
- led_out_bdigit()は引数の下位8bitをLEDに反映させる

```
void
main(void){
    unsigned char count = 0;

    led_init();

    for (;;) {
        led_out_bdigit(count++);
        busy_wait();
    }
}
```

ハードウェア
初期化

LED設定

2014/06/13

TOPPERSプロジェクト認定

52



led_count : 作成

プログラム led_count を作成せよ

- 手順
 - led_out_bdigit()を作成
 - 引数の下位4bitをそのままポートに書き込むと正しく表示されないため(LED1はビット7に接続), 引数見て, 対応するビットをONにする必要がある
 - LED接続ポートへの書き込みは, led_out_xxx() 関数を用いる

2014/06/13

TOPPERSプロジェクト認定

53



led_count : led_out_bdigit()

- 引数の各ビットをチェックし, 0ビット目が'1'ならLED1をON, 1ビット目が'1'ならLED2をONとなるようにled_dataを設定する
- led_dataの初期値は, 全てのLED OFF とする0x00|しておく

```

void
led_out_bdigit(unsigned char data){
    unsigned char led_data = 0x00;
    if ((data & 0x01) == 0x01){
        led_data = led_data | LED1;
    }
    if ((data & 0x02) == 0x02){
        led_data = led_data | LED2;
    }
    ...省略...
    led_out(led_data);
}

```

初期値

LED1設定

LED2設定

LED設定
led_out以下の5つの関数から選ぶ
①led_out_no_macro
②led_out_macro1
③led_out_macro2
④led_out_struct
⑤led_out_sil

2014/06/13

TOPPERSプロジェクト認定

54



デバッグ表示の記載例

- 正しくLEDが表示されない場合
- led_out_bdigitの引数が正しくLEDの設定値になっているか確認のためにsys_printf/sys_puthex文を入れてデバッグする(make DEBUG=1でビルド)

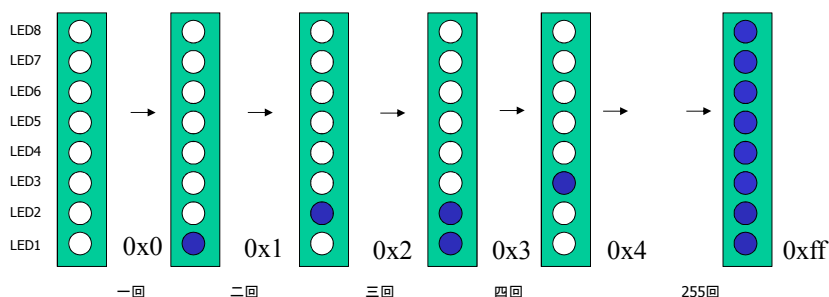
```
void
led_out_bdigit(unsigned char data){
    ...省略...
    led_out(led_data);
    sys_printf("led_out_bdigit data["); sys_puthex(data);
    sys_printf("] led_data ["); sys_puthex(led_data);
    sys_printf("¥n");
}
```

ポーリングプログラム

1. LEDプログラム
 - ・プロジェクトの作成方法
2. [スイッチプログラム](#)
3. タイマプログラム
4. シリアルI/Oプログラム

スイッチプログラム：switch_push 概要

- SW2の押下回数を、LEDにバイナリ表示する



- 学習内容
 - ポートからの読み込み
- 作成方法
 - led_countコピーしてベースに作成(次のスライドで説明)

2014/06/13

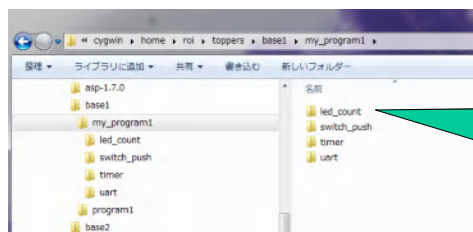
TOPPERSプロジェクト認定

57



led_countディレクトリのコピー

- 前の演習で作成したmy_program1中のled_countディレクトリのプログラムを元に以下の演習を行います
 - switch_push(この演習)
 - timer(ポーリングタイマ)
 - uart(ポーリングUART)
- led_countをmy_program1中に3つコピーしてそれぞれのディレクトリ名を、上記の3つに変更します



led_countディレクトリを3つコピーして作り、名称をswitch_push/timer/uartとする

2014/06/13

TOPPERSプロジェクト認定

58



switch_push : 作成

プログラム switch_push を作成

- 手順
 - マニュアルによるプッシュスイッチの接続の確認
 - マクロの定義
 - 初期化関数 switch_push_init() の作成
 - ポートを入力モードへ
 - プッシュスイッチ状態取得関数 switch_push_sense() の作成
 - ポートの読み込み
 - メイン関数
 - スイッチの変化を捉える (OFFからONへ)

2014/06/13

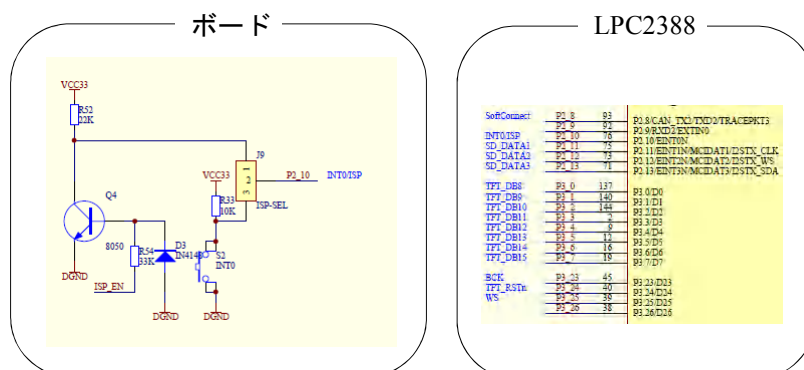
TOPPERSプロジェクト認定

59



switch_push : プッシュスイッチの接続

- マニュアルにより確認する
- マイコンのFIO2:10/INT0ポートに接続されている
- ONで電流が流れ、FIO2:10にSW2の状態が取り込める
 - マイコンからはONで'0'が, OFFで'1'が読み込める



2014/06/13

TOPPERSプロジェクト認定

60



switch_push : プッシュスイッチの接続ポート

- ポートFIO2に接続
 - SW10: ビット5 : P15
- ポートFIO2の構成レジスタ
 - ポート方向レジスタ : 0x3FFFC040
 - ポート読み込みレジスタ : 0x3FFFC054
- 初期化
 - ポート方向レジスタのビット10を入力モードへ(初期値)

Generic Name	Description	Access	Reset value ¹⁾	PORTn Register Address & Name
FIODIR	Fast GPIO Port Direction control register. This register individually controls the direction of each port pin.	R/W	0x0	FIO0DIR - 0x3FFF C000 FIO1DIR - 0x3FFF C020 FIO2DIR - 0x3FFF C040 FIO3DIR - 0x3FFF C060 FIO4DIR - 0x3FFF C080
FIOPIN	Fast Port Pin value register using FIOMASK. The current state of digital port pins can be read from this register, regardless of pin direction or alternate function selection (as long as pins are not configured as an input to ADC). The value read is masked by ANDing with inverted FIOMASK. Writing to this register places corresponding values in all bits enabled by zeros in FIOMASK. Important: If a FIOPIN register is read, its bit(s) masked with 1 in the FIOMASK register will be set to 0 regardless of the physical pin state.	R/W	0x0	FIO0PIN - 0x3FFF C014 FIO1PIN - 0x3FFF C034 FIO2PIN - 0x3FFF C054 FIO3PIN - 0x3FFF C074 FIO4PIN - 0x3FFF C094

2014/06/13

TOPPERSプロジェクト認定

61



switch_push : マクロ定義

- ポインタ型へのキャストまで含めてマクロ化する(方針)
- FIO2関連のレジスタのマクロ定義

```
#define TADR_FIO2_DIR 0x3FFFC040
#define TADR_FIO2_PIN 0x3FFFC054
#define TREG_FIO2_DIR ((volatile unsigned long *)TADR_FIO2_DIR)
#define TREG_FIO2_PIN ((volatile unsigned long *)TADR_FIO2_PIN)
```

- FIO2ポートのプッシュスイッチの接続ビット定義

```
#define PSW2 0x400
#define PSW_MASK (PSW2)
```

2014/06/13

TOPPERSプロジェクト認定

62



switch_push : 初期化関数

- FIO2ポートのビット10を入力モードへ
 - FIO2ポート方向レジスタのビット10を'0'に設定する

```
void
switch_push_init(void){
    unsigned long reg;

    reg = *TREG_FIO2_DIR;
    reg = reg & ~PSW_MASK;
    *TREG_FIO2_DIR = reg;
}
```

ビット10のみ'0'に
設定

switch_push : プッシュスイッチ状態取得関数

- switch_push_sense()
- FIO2ポートレジスタのビット10の値を反転し取得

```
unsigned long
switch_push_sense(void){
    unsigned long reg;

    reg = *TREG_FIO2_PIN;
    reg = reg & PSW_MASK;
    return (reg ^ PSW_MASK) & PSW_MASK;
}
```

ビット10のビットを反転後
ビット10のみ有効に

switch_push : main()関数

- カウント用変数led_countの用意
- ハードウェアの初期化
- 無限ループ内でプッシュスイッチの状態をチェックしてSW9がOFFからONに変化した場合は, led_countをインクリメントし, SW8がOFFからONに変化した場合は, led_countをデクリメントする
- 状態の変化の捉え方
 - プッシュスイッチの以前の状態を保存して比較
 - ローカル変数を用意して前回の状態取得時の状態を保存する
- スwitchの状態(ON,OFF)はマクロ化する

```
#define SW_ON    1
#define SW_OFF   0
```

2014/06/13

TOPPERSプロジェクト認定

65



switch_push : main()関数

```
void _main(void){
    unsigned char led_data;
    unsigned long sw_data;
    unsigned char sw2_state = SW_OFF;
    led_init();
    switch_push_init();
    for (;;) {
        sw_data = switch_push_sense();
        if(((sw_data & PSW2) == PSW2) && (sw2_state == SW_OFF)){
            led_data++;
        }
        sw2_state = ((sw_data & PSW2) == PSW2)? SW_ON : SW_OFF;
        led_out_bdigit(led_data);
    }
}
```

SW2状態保存用変数

プッシュスイッチ状態取得

SW2がOFFからONになったか

SW2の状態を保存

LED設定

2014/06/13

TOPPERSプロジェクト認定

66



ポーリングプログラム

1. LEDプログラム
 - ・プロジェクトの作成方法
2. スイッチプログラム
3. タイマプログラム
4. シリアルI/Oプログラム

2014/06/13

TOPPERSプロジェクト認定

67



タイマプログラム : timer 概要

- ハードウェアタイマにより正確に1秒間隔でLEDの点灯をカウントアップする
- 学習内容
 - ハードウェアタイマの使い方
 - カウントソース, カウント値の決定方法
- 作成方法
 - led_count よりコピーしたtimerをベースに作成

2014/06/13

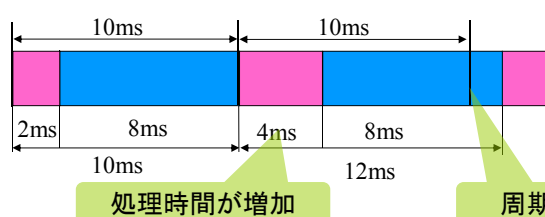
TOPPERSプロジェクト認定

68



timer : ハードウェアタイマによる時間生成

- led_countでは, 時間生成にbusy_wait()関数を使用
 - for()ループにより一定時間を生成
- ループでは正確な時間が生成できない
 - そもそもループでは正確な時間は作れない
 - 正確な周期処理を実現が困難
 - 周期 = 周期処理の実行時間 + ループの実行時間
 - 周期処理の実行時間が変わると, それに合わせてループの長さを変更する必要がある



2014/06/13

TOPPERSプロジェクト認定

69



timer : ハードウェアタイマの概要

- LPC2388のハードウェアタイマ
 - 32ビットタイマを4個
 - タイマモード, 4つのマッチレジスタ, カウント方法やクロックソースはコントロールレジスタで設定可能
- タイマ0を使用
 - タイマモードを使用
 - 内部カウントソースによりカウントアップ
 - マッチカウンタと比較
 - マッチ後、カウンタリセット
 - メインクロック48MHzを1/4に分周(デフォルト)
 - 周辺クロック選択レジスタで変更可能

2014/06/13

TOPPERSプロジェクト認定

70



timer : レジスタ

- ・ タイムアウトは割込み制御レジスタのMR0ビットにより通知
 - タイマ側のレジスタのビットがセットされる場合が多い

名前	アドレス	説明
T0IR	0xE0004000	タイマ0割込み制御レジスタ
T0TCR	0xE0004004	タイマ0コントロールレジスタ
T0TC	0xE0004008	タイマ0カウンター
T0PR	0xE000400C	タイマ0プリスケールレジスタ
T0MCR	0xE0004014	タイマ0マッチコントロールレジスタ
T0MR0	0xE0004018	マッチレジスタ0
T0CTCR	0xE0004028	チャプターコントロールレジスタ

6.1 Interrupt Register (T[0/1/2/3]IR - 0xE000 4000, 0xE000 8000, 0xE007 0000, 0xE007 4000)

The Interrupt Register consists of four bits for the match interrupts and four bits for the capture interrupts. If an interrupt is generated then the corresponding bit in the IR will be high. Otherwise, the bit will be low. Writing a logic one to the corresponding IR bit will reset the interrupt. Writing a zero has no effect.

Table 473. Interrupt Register (T[0/1/2/3]IR - addresses 0xE000 4000, 0xE000 8000, 0xE007 0000, 0xE007 4000) bit description

Bit	Symbol	Description	Reset Value
0	MR0 Interrupt	Interrupt flag for match channel 0	0
1	MR1 Interrupt	Interrupt flag for match channel 1	0
2	MR2 Interrupt	Interrupt flag for match channel 2	0
3	MR3 Interrupt	Interrupt flag for match channel 3	0
4	CR0 Interrupt	Interrupt flag for capture channel 0 event	0
5	CR1 Interrupt	Interrupt flag for capture channel 1 event	0
6	-	Reserved	0
7	-	Reserved	0

timer : タイマ操作の流れ

- ・ タイマ0モードを設定
 - 停止とリセット、割込みクリア、プリスケールと割込みを設定
- ・ タイマ0:MR0レジスタにカウント値を設定
 - カウントソースと計時したい時間から求める
- ・ マッチコントロール設定、カウント開始フラグを設定
- ・ タイマ0割込み制御レジスタをチェックして、タイムアウトの通知を待つ

timer : 作成

プログラム timer を作成

- 手順
 - カウントソースの決定
 - デフォルト1/4分周を使用
 - カウント値の決定
 - 1秒は直接計時不可能なため 100msecを10回
 - プログラミング
 - マクロの定義
 - 初期化関数
 - 計時関数
 - メイン関数

2014/06/13

TOPPERSプロジェクト認定

73



timer : カウントソースの選択

カウントソースを高速にすると精度は高くなるが測定可能な最大時間が小さくなり、カウントソースを低速にするとその逆となる

- カウント値とカウントソースの決定
 - カウント値を保持するタイマ0レジスタは32bit
 - 0から4264967595までを設定可能
 - カウントソース
 - 1/4(00),1/1(01),1/2(10),1/8(11)から設定可能
 - メインクロックが48MHzなので,
 - 1/4 : 48MHz/4 = 12MHz
 - 1/2: 48MHz/2 = 24MHz
 - 各ソースで測定可能な最大値
 - 1/4 : $1 / 12\text{MHz} * 4264967595 = 355.414 * 10^{-6} = 355.4\text{msec}$
 - 1/2 : $1 / 24\text{MHz} * 4264967595 = 107.707 * 10^{-6} = 107.7\text{msec}$
 - 1/8 : $1 / 6\text{MHz} * 4264967595 = 701.83 * 10^{-6} = 701.8\text{msec}$

2014/06/13

TOPPERSプロジェクト認定

74



timer : カウント値の決定

- 1/8でも1秒のカウントは不可能:デフォルトの1/4を使用
 - 100msecを10回計測して1秒を生成する
 - カウント値を計算
 - $(100\text{msec} / (1/12\text{MHz})) - 1 = 1199999$
- その他のレジスタの設定値
 - T0TCRの0ビット目をセットするとカウントをスタート
 - タイムアウトするとT0IRのビット0(0x01)が‘1’にセットされる
 - T0IRのビット0から‘1’を読み込んだ後は‘0’にクリアする
 - (0x00000001を書き込んでクリア)

名前	アドレス	設定値	説明
T0MR	0xE0004018	0x00124F7F	マッチレジスタ0,クロックソースは1/4
T0MCR	0xE0004014	0x00000003	MR0のリセットと割込みを設定
T0TCR	0xE0004004	0x00000001	タイマスタート(0x02で停止)

2014/06/13

TOPPERSプロジェクト認定

75



timer : マクロの定義

- タイマ関連のレジスタ(サイズは4バイト)

```

/* タイマ0割込みレジスタ */
#define TREG_TMR0_IR ((volatile unsigned long *)0xE0004000)
/* タイマ0コントロールレジスタ */
#define TREG_TMR0_TCR ((volatile unsigned long *)0xE0004004)
/* タイマ0プリスケールレジスタ */
#define TREG_TMR0_PR ((volatile unsigned long *)0xE000400C)
/* タイマ0マッチコントロールレジスタ */
#define TREG_TMR0_MCR ((volatile unsigned long *)0xE0004014)
/* タイマ0マッチレジスタ0 */
#define TREG_TMR0_MR0 ((volatile unsigned long *)0xE0004018)
#define TREG_TMR0_CTCT ((volatile unsigned long *)0xE0004070) /* カウンタ制御 */

```

- タイマ関連の定義

```

#define TIMER_COUNT ((12000*100)-1) /* 1/4分周で100msec */
#define COUNTER_START 0x00000001 /* タイマスタート */
#define COUNTER_RESET 0x00000002 /* タイマリセット */
/* タイマ0MR0設定とIR参照 */
#define MATCH_CTL_OINT 0x00000001 /* MR0割込み要求 */
#define MATCH_CTL_ORST 0x00000002 /* MR0リセット設定 */
#define MATCH_CTL_OSTP 0x00000004 /* MR0停止要求 */
#define MR0_INTERRUPT 0x00000001 /* MR0割込み */

```

2014/06/13

TOPPERSプロジェクト認定

76



timer : 初期化関数

- タイマーのモードを設定する
 - 停止させる必要があるかはタイマによって異なる
 - LPC2388はマニュアルに特に記載がない

```
void
timer0_init(void){
    *TREG_TMR0_TCR = COUNTER_RESET;
    *TREG_TMR0_IR  = MR0_INTERRUPT;
    *TREG_TMR0_CTCR= 0x00000000;
    *TREG_TMR0_PR  = 0x00000000;
}
```

タイマ停止

割込みクリア

モード設定
(デフォルト:エッジ設定)プリスケール設定
(デフォルト:1/4分周)

2014/06/13

TOPPERSプロジェクト認定

77



timer : 計時関数

- 引数で指定された回数タイマで計時する
 - オートリロードのため、スタートさせると後は自動的に計時を繰り返す

```
void
timer0_wait(int cnt){
    int lp_cnt;
    *TREG_TMR0_TCR = COUNTER_RESET;
    *TREG_TMR0_MR0 = TIMER_COUNT;
    *TREG_TMR0_MCR = (MATCH_CTL_0INT | MATCH_CTL_0RST);
    *TREG_TMR0_TCR = COUNTER_START;
    for(lp_cnt = 0; lp_cnt < cnt; lp_cnt++){
        while((*TREG_TMR0_IR & MR0_INTERRUPT) == 0);
        *TREG_TMR0_IR = MR0_INTERRUPT;
    }
    *TREG_TMR0_TCR = COUNTER_RESET;
}
```

タイマ停止

カウント値設定

マッチ制御設定

タイマスタート

タイムアウト待ち

割込みフラグクリア

タイマ停止

2014/06/13

TOPPERSプロジェクト認定

78



timer : メイン関数

- led_countのメイン関数を変更
 - busy_wait() を timer0_wait()に変更

タイマの初期化

LEDの出力

```
void
main(void){
    int count = 0;
    led_init();
    timer0_init();
    for (;;) {
        led_out_bdigit(count++);
        timer0_wait(10);
    }
}
```

LEDの初期化

タイマにより待つ

2014/06/13

TOPPERSプロジェクト認定

79



ポーリングプログラム

1. LEDプログラム
 - ・プロジェクトの作成方法
2. スイッチプログラム
3. タイマプログラム
4. シリアルI/Oプログラム

2014/06/13

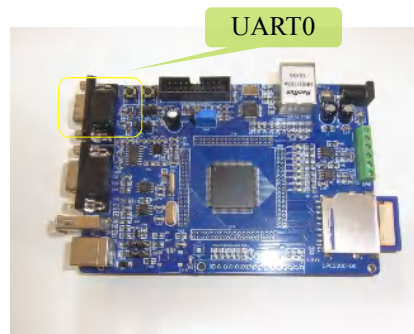
TOPPERSプロジェクト認定

80



シリアルI/Oプログラム : uart 概要

- シリアルI/Oからデータを受信すると, LEDの点灯をカウントアップし, 受け取ったデータをシリアルI/Oに送信する (ポーリング)
- 学習内容
 - シリアルI/Oの使い方
- 作成方法
 - led_countよりコピーした uartをベースに作成
- ボードの設定
 - UART0とPCのシリアルポートを接続する



2014/06/13

TOPPERSプロジェクト認定

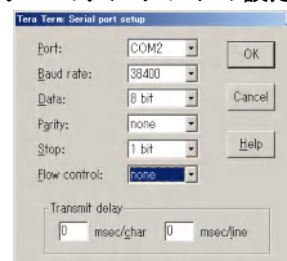
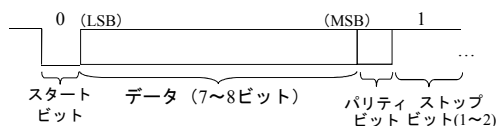
81



uart : シリアルI/Oのフォーマット

- 通信する双方で同じ設定にする必要がある
 - ボーレート(BPS) : 9600, 14400, 38400, 57600, 115200
 - データサイズ : 7bit, 8bit
 - パリティ : なし, even, odd
 - ストップビット : 1bit, 2bit
 - フローコントロール : なし, Xon/Xoff, hardware
- ターミナルソフトの設定例

データフォーマット



2014/06/13

TOPPERSプロジェクト認定

82



uart : 送受信フォーマットとレジスタ

- フォーマット
 - ボーレート(BPS) : 38400, データサイズ : 8bit, パリティ: なし, ストップビット: 1bit, フローコントロール : なし
- 使用チャネル
 - UART0を使用
 - UARTモードで使用, カウントソースは(48Mhzの1/4分周)

名前	アドレス	説明
PINSEL0	0xE002C000	ピンセレクトレジスタ0
UART0_RBR	0xE000C000	受信バッファレジスタ
UART0_THR	0xE000C000	送信ホールドレジスタ
UART0_DLL	0xE000C000	送受信デバイザーラッチLSB
UART0_DLM	0xE000C004	送受信デバイザーラッチMSB
UART0_IER	0xE000C004	送受信割込みイネーブルレジスタ
UART0_LCR	0xE000C00C	送受信ライン制御レジスタ
UART0_LSR	0xE000C014	送受信ラインステータスレジスタ

2014/06/13

TOPPERSプロジェクト認定

83



uart : 作成

プログラム uart を作成

- 手順
 - 転送速度レジスタの設定値の計算
 - ボーレートが38400bpsになるよう設定
 - その他のレジスタの設定値の決定
 - プログラミング
 - マクロの定義
 - 初期化関数
 - 受信関数
 - 送信関数
 - メイン関数

2014/06/13

TOPPERSプロジェクト認定

84



uart : 転送速度レジスタの設定値の計算

- 転送速度は 38400bps
- カウントソースは $f(12\text{Mhz}:48\text{Mhz}/4)$ を使用
- 転送速度レジスタの設定値の計算式(UARTモード)
 - $f/(16(n+1))$
 - $12\text{MHz}/(16(n+1)) = 38400$
- $n = 19$ にすると
 - $12\text{Mhz}/(16(19+1)) = 39473$
 - 誤差が数%以内なら動作

2014/06/13

TOPPERSプロジェクト認定

85



uart : その他のレジスタの設定値

フォーマットに従って設定値を決定する

- PINSEL0にて、GPIOをTX0:RX0に変更
- 送受信ライン制御レジスタ : 0x00000003
 - UARTモード転送データ長8ビット, 内部クロック,
 - 1ストップビット, パリティ禁止
- 送受信FIFO制御レジスタ : FIFOは使用しない
- クロックはデフォルトクロック(1/4分周)を使用するため設定は行わない
- 転送速度設定(DLM:DLL)を設定する場合は送受信ライン制御レジスタのDLAB(ビット7)を1にして設定を行う

2014/06/13

TOPPERSプロジェクト認定

86



uart : マクロ定義1 : UART0関連のレジスタ

```

/* PINSELECT#0 レジスタ */
#define TREG_PINSEL0 ((volatile unsigned long *)0xE002C000)
/* 送信バッファレジスタ */
#define TREG_UART0_THR ((volatile unsigned long *)0xE000C000)
/* 受信バッファレジスタ */
#define TREG_UART0_RBR ((volatile unsigned long *)0xE000C000)
/* UART0 デバイザーラッチLSB */
#define TREG_UART0_DLL ((volatile unsigned long *)0xE000C000)
/* UART0 デバイザーラッチMSB */
#define TREG_UART0_DLM ((volatile unsigned long *)0xE000C004)
/* 送受信割込み制御レジスタ0 */
#define TREG_UART0_IER ((volatile unsigned long *)0xE000C004)
/* 送受信ライン制御レジスタ0 */
#define TREG_UART0_LCR ((volatile unsigned long *)0xE000C00C)
/* 送受信ラインステータスレジスタ0 */
#define TREG_UART0_LSR ((volatile unsigned long *)0xE000C014)

```

2014/06/13

TOPPERSプロジェクト認定

87



uart : マクロ定義2 : レジスタの設定値

- 定めた設定値でマクロを定義

```

#define DEFAULT_PCLK 12000000 /* UART基本クロック */
#define LCR_VAL 0x03 /* 送受信制御レジスタ0の設定値 */
#define LCR_DLAB 0x80 /* DLABビットの定義 */

```

- 送受信ラインステータスレジスタ0のビット定義

```

#define US_TXEMPTY 0x20 /* 送信バッファエンプティ */
#define US_FE 0x08 /* 受信フレーミングエラー */
#define US_PE 0x04 /* 受信パリティエラー */
#define US_OE 0x02 /* 受信オーバーランエラー */
#define US_RXRDY 0x01 /* 受信バッファあり */

```

2014/06/13

TOPPERSプロジェクト認定

88



uart : 初期化関数 uart0_init

- 各制御レジスタを初期化し, 送受信を許可する

```
void
uart0_init(void){
    unsigned long tmp;
    /* RX0/TX0の設定 */
    tmp = *TREG_PINSEL0;
    tmp &= ~0x000000f0;
    tmp |= 0x00000050;
    *TREG_PINSEL0 = tmp;
    /* 送受信モードの設定 */
    *TREG_UART0_LCR = (LCR_DLAB | LCR_VAL);
    tmp = (DEFAULT_PCLK * 10) / 16 / DEFAULT_SPEED;
    tmp = (tmp+5) / 10;
    *TREG_UART0_DLL = tmp & 0xff;
    *TREG_UART0_DLM = tmp >> 8;
    *TREG_UART0_LCR = LCR_VAL;
}
```

DEFAULT_SPEEDは
sys_defs.hで定義

2014/06/13

TOPPERSプロジェクト認定

89



uart : 受信関数

- データを受信すると, 受信制御レジスタのRIビット(受信完了フラグ)がセットされる
- ポーリングでセットされるのを待つ

```
unsigned char
uart0_pol_getc(void){
    /* 受信バッファにデータが入るのを待つ */
    while((*TREG_UART0_LSR & US_RXRDY) == 0);

    /* データ受信 */
    return (unsigned char)(*TREG_UART0_RBR);
}
```

2014/06/13

TOPPERSプロジェクト認定

90



uart : 送信関数

- 送信データを書き込める状態であれば, 受信制御レジスタのTI(送信バッファ空フラグ)がセットされる
- ポーリングでセットされるのを待つ

```
void
uart0_pol_putc(unsigned char c){
    /* 送信バッファが空くのを待つ */
    while((*TREG_UART0_LSR & US_TXEMPTY) == 0);

    /* データ送信 */
    *TREG_UART0_THR = c;
}
```

2014/06/13

TOPPERSプロジェクト認定

91



uart : メイン関数

- 受信関数でデータを受信後, LEDをカウントアップし, 送信関数で受信したデータを送信する

```
void
main(void){
    unsigned char count = 0;
    unsigned char c;

    led_init();
    uart0_init();

    for (;;) {
        c = uart0_pol_getc();
        led_out_bdigit(++count);
        uart0_pol_putc(c);
    }
}
```

データ受信

LEDカウントアップ

データ送信

2014/06/13

TOPPERSプロジェクト認定

92



割込みプログラミング

1. [LPC2388の割込みアーキテクチャ](#)
2. スイッチ割込みプログラム

2014/06/13

TOPPERSプロジェクト認定

93



目的

割込みを用いたプログラミングを学ぶ

- 割込みによる事象待ちのために必要な知識
 - 割込みハンドラ
 - 割込みベクタテーブル
 - 割込み関連のレジスタ
 - 割込み禁止・許可
 - 割込み優先度
- プログラム対象の周辺デバイス
 - プッシュスイッチ
 - タイマ
 - シリアルI/O

2014/06/13

TOPPERSプロジェクト認定

94



割込みプログラミング：プログラムファイル

- プログラムファイルの置き場所
 - 教材ディレクトリ¥program1
- プログラム一覧
 - int_switch_1 : PUSHスイッチ割込みプログラム1
(単一割込み)
 - int_switch_2 : PUSHスイッチ割込みプログラム2
(割込み禁止/割込み応答時間)
 - int_switch_3 : PUSHスイッチ割込みプログラム3
(複数割込み/優先度の設定)
 - int_timer : タイマ割込みプログラム
 - int_uart : シリアルI/O割込みプログラム

2014/06/13

TOPPERSプロジェクト認定

95



ARM7の割込みアーキテクチャ

割込み、エクセプションの種別で0番地からの7のアドレスから
実行、実行モード、スタック等の設定はソフトウェアで記述

- 割込み許可等はCPSRレジスタ中にある
 - 割込み禁止はA,I,Fビット
- 割込みにより、実行モードが変化すると、CPSRレジスタの内容はSPCR_xxxレジスタに退避される。割込み前の実行番地+4は変化した実行モードに対応したlr(R14)レジスタに退避
- 細かい割込み機構はSoCメーカーによって異なる
 - LPCでは割込み番号管理、レベル設定
 - 割込みハンドラの取出し等の機能を持つ
 - エッジ、レベル設定はEXTINTのみ
(割込みデバイスより設定は規定される)

2014/06/13

TOPPERSプロジェクト認定

96



LPC2388割り込みアーキテクチャ: VIC

周辺機能の割り込み用割り込み
ハンドラのアドレスを登録

- VIC (Vectored interrupt Controller)
 - ←LPC23XXユーザーズ
マニュアル: Chapter 6
- 32のIRQ/FIQ割り込みの制御
 - 割り込み番号に対応した
ハンドラアドレスと割り
込みレベルを設定、割り
込みが発生すると対応の
VICAddress取り出せる

Table 76. VIC register map

Name	Description	Access	Reset value ⁽¹⁾	Address
VICIRQStatus	IRQ Status Register. This register reads out the state of those interrupt requests that are enabled and classified as IRQ.	RO	0	0xFFFFF000
VICFIQStatus	FIQ Status Register. This register reads out the state of those interrupt requests that are enabled and classified as FIQ.	RO	0	0xFFFFF004
VICRawINTR	Raw Interrupt Status Register. This register reads out the state of the 32 interrupt requests / software interrupts, regardless of enabling or classification.	RO	-	0xFFFFF006
VICIntSelect	Interrupt Select Register. This register classifies each of the 32 interrupt requests as contributing to FIQ or IRQ.	R/W	0	0xFFFFF00C
VICIntEnable	Interrupt Enable Register. This register controls which of the 32 interrupt requests and software interrupts are enabled to contribute to FIQ or IRQ.	R/W	0	0xFFFFF010
VICIntEnClr	Interrupt Enable Clear Register. This register allows software to clear one or more bits in the Interrupt Enable register.	WO	-	0xFFFFF014
VICSoftInt	Software Interrupt Register. The contents of this register are OR'ed with the 32 interrupt requests from various peripheral functions.	R/W	0	0xFFFFF018
VICSoftIntClr	Software Interrupt Clear Register. This register allows software to clear one or more bits in the Software Interrupt register.	WO	-	0xFFFFF01C
VICProtection	Protection enable register. This register allows limiting access to the VIC registers by software running in privileged mode.	R/W	0	0xFFFFF020
VICSWPriorityMask	Software Priority Mask Register. Allows masking individual interrupt priority levels in any combination.	R/W	0xFFFF	0xFFFFF024
VICVectAddr0	Vector address 0 register. Vector Address Registers 0-31 hold the addresses of the Interrupt Service routines (ISRs) for the 32 vectored IRQ calls.	R/W	0	0xFFFFF100

Table 77. VIC register map

Name	Description	Access	Reset value ⁽¹⁾	Address
VICVectPriority05	Vector priority 25 register.	R/W	0x0	0xFFFFF268
VICVectPriority27	Vector priority 27 register.	R/W	0x0	0xFFFFF26C
VICVectPriority28	Vector priority 28 register.	R/W	0x0	0xFFFFF270
VICVectPriority29	Vector priority 29 register.	R/W	0x0	0xFFFFF274
VICVectPriority30	Vector priority 30 register.	R/W	0x0	0xFFFFF278
VICVectPriority31	Vector priority 31 register.	R/W	0x0	0xFFFFF27C
VICAddress	Vector address register. When an IRQ interrupt occurs, the Vector Address Register holds the address of the currently active interrupt.	R/W	0	0xFFFFF100

⁽¹⁾ Reset value reflects the data stored in user table only. It does not include reserved bits content.

2014/06/13

TOPPERSプロジェクト認定

97



LPC2388割り込みアーキテクチャ: VIC

- Interrupt sourcesに対応したVICVectAddr#nに割り込みハンドラアドレス、VICVectPriority#nに優先順位を設定
 - 優先順位は0~15の値で、15が最下位割り込みレベル
- VICIntEnableのinterrupt sources対応ビットを1にすると割り込み許可
- VICSoftClrのinterrupt sources対応ビットを1にすると割り込みソフトクリアする

Table 87. Interrupt sources bit allocation table

Bit	31	30	29	28	27	26	25	24
Symbol	I2S	I2C2	UART3	UART2	TIMER3	TIMER2	GPDMA	SD/MMC
Bit	23	22	21	20	19	18	17	16
Symbol	CAN1&2	USB	Ethernet	BOD	I2C1	AD0	EINT3	EINT2
Bit	15	14	13	12	11	10	9	8
Symbol	EINT1	EINT0	RTC	PLL	SSP1	SPI/SSP0	I2C0	PWM1
Bit	7	6	5	4	3	2	1	0
Symbol	UART1	UART0	TIMER1	TIMER0	ARMCORE1	ARMCORE0	-	WDT

2014/06/13

TOPPERSプロジェクト認定

98



ARM7の割込みアーキテクチャ：割込みシーケンス

- プロセッサによる割込み処理シーケンス (IRQモード)
 1. 割込みを受け付ける
 2. CPSRレジスタの内容をSPSR_irqにコピー
 3. CPSRのIビットを1 (IRQ割込み禁止)
 4. 実行再開アドレス+4をIRQモードのlrレジスタにコピー
 5. 0x00000018番地からIRQモードで実行

ハードウェアが何をしてくれて何をしてくれないかを明確にする

- CISCプロセッサでは割込みベクトル管理を行い割込みハンドラの実行をハードウェアでサポートしてくれる
- ARMのような、RISCプロセッサの多くは5までの処理しか行わず、残りはプログラム(ソフトウェア)で処理しなければならない

LPC2388の割込みアーキテクチャ

- 0x00000018番地からIRQ_Handlerにジャンプする、割込みハンドラを実行するプログラムはアセンブラで記述する

```

IRQ_Handler:
    sub    lr,lr,#4      /* 再開アドレスを計算 */
    push   {lr}          /* 再開アドレスをIRQ-stackにスタックする */
    mrs    lr,sp,sp,lr   /* IRQ spsrをlrレジスタに取り出す */
    push   sp!,{r0-r4,r12,lr} /* 割込みハンドラで使用するレジスタをセーブ */
    msr    cpsr_c,#(0x13|0x80) /* 割込み禁止スーパーバイザモードに変更 */
    and    r1,sp,#4      /* スタックを8のアラインに補正 */
    sub    sp,sp,r1
    push   {r1,lr}       /* 補正值とlrレジスタをセーブ */
    ldr     r3,=TADR_VIC_BASE /* ハンドラアドレスを取り出す */
    ldr     r0,[r3,#TOFF_VIC_ADDR]
    cmp    r0,#0x00      /* ハンドラアドレスの有効判定 */
    bne    IRQ_callhandler /* 有効の場合スキップ */
    ldr     r0,reserved_handler /* 無限ループルーチンを設定 */
IRQ_callhandler:
    mrs    r2,cpsr       /* 割込み許可 */
  
```

LPC2388の割込みアーキテクチャ

- 割込みハンドラは割込み解除状態のスーパーバイザモードで実行する(上位の割込みが可能になる)

```

and r2,r2,0x80 /* 割込み許可続き */
msr cpsr,r2
mov lr,pc /* リターンアドレスをlrにセット */
bx r0 /* 割込みハンドラにジャンプ */
msr r2,cpsr /* 割込み禁止 */
orr r2,r2,#0x80
msr cpsr,r2
pop {r1,lr} /* スタック補正值とlrレジスタを復帰 */
add sp,sp,r1 /* スーパーバイザスタックをもとに戻す */
ldr r3,=TADR_VIC_BASE /* VICの割込みクリア */
ldr r0,=0x00000000
str r0,[r3,#TOFF_VIC_VECTADDR]
msr cpsr_c,#(0x12|0x80) /* IRQモードに移行 */
pop {r0-r4,r12,lr} /* レジスタを復帰 */
msr spsr_cxsf,lr /* SPSRを復帰する */
ldmfd sp!,{pc}^ /* もとのプログラムに復帰 */

```

この区間で割込み
ハンドラが呼び出される

割込みプログラミング

1. LPC2388の割込みアーキテクチャ
2. [スイッチ割込みプログラム](#)

スイッチ割込みプログラム1 : int_switch_1 概要

- SW2を押すと発生する割込み (EXTINT0) により, LEDを
カウントアップする
- 学習内容
 - 割り込みプログラムの全体像の理解
 - 割込みベクタテーブル
 - 割込み制御レジスタ
 - 割込みハンドラ
- 動作確認
 - 作成済みのプログラムを実行し動作を確認する

2014/06/13

TOPPERSプロジェクト認定

103



int_switch_1 : 割込みプログラムの全体像

- 割込みをプログラムの視点から見ると
 - メイン関数を実行中に割込みが入ると、一旦メイン関数中の処理が中断され、割込みに対応した関数 (割込みハンドラ) が実行される
 - 割込みハンドラの実行が終了すると、メイン関数の処理が再開される
- プログラムで必要となる処理
 - 割込み要因ハードウェアの初期化
 - 割込みハンドラの用意
 - 割込みベクタテーブルの用意
 - 割込み関連の初期化

2014/06/13

TOPPERSプロジェクト認定

104



int_switch_1 : 割込み要因ハードウェア

プッシュスイッチ(SW2)を割込み要因として用いる

- 接続
 - FIO2入出力ポートのビット10(P2:10)に接続
 - P2:10は外部割込み0(EXTINT0)とピンを共有
 - ボタンを押すと‘0’が入力される
- 割込み設定には、GPIO-INTとEXTINTがある
 - 今回は、EXTINTを用いて割込み設定する
- 割込み番号、優先度(VICを参照)
 - 割込み番号EINT0:14番に設定済
 - 割込みハンドラ、レベルを設定
- 外部割込みのモード設定:ユーザーマニュアルCapter3参照
 - PINをEINT0に設定、モードを立下りエッジに設定する

2014/06/13

TOPPERSプロジェクト認定

105



int_switch_1 : 割込み要因ハードウェアの初期化

ポート関連のレジスタと
割込み制御関連のレジスタの初期化が必要

- ポート関連レジスタの初期化
 - ポーリングプログラミング switch_push の初期化関数を流用
- 割込み制御関連のレジスタ
 - 割込み制御レジスタ
 - 割込みレベルや割込みエッジの設定
 - 割込み要因選択レジスタ
 - 割込み極性切り替え(両エッジか片エッジか)

2014/06/13

TOPPERSプロジェクト認定

106



switch_push : プッシュスイッチの接続ポート

- PINSELECT4にて、P2:10の接続をEINT0に変更
- EXTINT/EXTMODE/EXTPOLARレジスタに割込みモード設定を行う

2. Pin description

Table 3-15 shows pins that are associated with System Control block functions.

Pin name	Pin direction	Pin description
EINT0	Input	External Interrupt Input 0 - An active low/high level or falling/rising edge general purpose interrupt input. This pin may be used to wake up the processor from life or power down modes.
EINT1	Input	External Interrupt Input 1 - See the EINT0 description above.
EINT2	Input	External Interrupt Input 2 - See the EINT0 description above.
EINT3	Input	External Interrupt Input 3 - See the EINT0 description above.
RESET	Input	External Reset Input - A LOW on this pin resets the chip, causing I/O ports and peripherals to take on their default states, and the processor to begin execution at address 0x0000 0000.

3. Register description

All registers, regardless of size, are on word address boundaries. Details of the registers appear in the description of each function.

Table 16. Summary of system control registers

Name	Description	Access	Reset value	Address
EXTINT	External Interrupt Flag Register	R/W	0x00	0xE01F C140
EXTMODE	External Interrupt Mode register	R/W	0x00	0xE01F C148
EXTPOLAR	External Interrupt Polarity Register	R/W	0x00	0xE01F C14C

Table 19. External Interrupt Flag register (EXTINT - address 0xE01F C140) bit description

Bit	Symbol	Description	Reset value
0	EINT0	In level-sensitive mode, this bit is set if the EINT0 function is selected for its pin, and the pin is in its active state. In edge-sensitive mode, this bit is set if the EINT0 function is selected for its pin, and the selected edge occurs on the pin. This bit is cleared by writing a one to it, except in level sensitive mode when the pin is in its active state [1].	0
1	EINT1	In level-sensitive mode, this bit is set if the EINT1 function is selected for its pin, and the pin is in its active state. In edge-sensitive mode, this bit is set if the EINT1 function is selected for its pin, and the selected edge occurs on the pin. This bit is cleared by writing a one to it, except in level sensitive mode when the pin is in its active state [1].	0
2	EINT2	In level-sensitive mode, this bit is set if the EINT2 function is selected for its pin, and the pin is in its active state. In edge-sensitive mode, this bit is set if the EINT2 function is selected for its pin, and the selected edge occurs on the pin. This bit is cleared by writing a one to it, except in level sensitive mode when the pin is in its active state [1].	0
3	EINT3	In level-sensitive mode, this bit is set if the EINT3 function is selected for its pin, and the pin is in its active state. In edge-sensitive mode, this bit is set if the EINT3 function is selected for its pin, and the selected edge occurs on the pin. This bit is cleared by writing a one to it, except in level sensitive mode when the pin is in its active state [1].	0
7:4	-	Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA

[1] Example: e.g. if the EINT0 is selected to be low level sensitive and low level is present on corresponding pin, this bit can not be cleared; this bit can be cleared only when signal on the pin becomes high.

2014/06/13

TOPPERSプロジェクト認定

107



swieth_push: EXT割込み設定

- EXTMODE0: エッジ割込み(1)
- EXTPOLAR0: フォーリングエッジ設定(0)
- EXTINT: ビット0に1を書き込み割込みクリア

Table 20. External Interrupt Mode register (EXTMODE - address 0xE01F C148) bit description

Bit	Symbol	Value	Description	Reset value
0	EXTMODE0	0	Level-sensitivity is selected for EINT0.	0
		1	EINT0 is edge sensitive.	
1	EXTMODE1	0	Level-sensitivity is selected for EINT1.	0
		1	EINT1 is edge sensitive.	
2	EXTMODE2	0	Level-sensitivity is selected for EINT2.	0
		1	EINT2 is edge sensitive.	
3	EXTMODE3	0	Level-sensitivity is selected for EINT3.	0
		1	EINT3 is edge sensitive.	
7:4	-	-	Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA

Table 21. External Interrupt Polarity register (EXTPOLAR - address 0xE01F C14C) bit description

Bit	Symbol	Value	Description	Reset value
0	EXTPOLAR0	0	EINT0 is low-active or falling-edge sensitive (depending on EXTMODE0).	0
		1	EINT0 is high-active or rising-edge sensitive (depending on EXTMODE0).	
1	EXTPOLAR1	0	EINT1 is low-active or falling-edge sensitive (depending on EXTMODE1).	0
		1	EINT1 is high-active or rising-edge sensitive (depending on EXTMODE1).	
2	EXTPOLAR2	0	EINT2 is low-active or falling-edge sensitive (depending on EXTMODE2).	0
		1	EINT2 is high-active or rising-edge sensitive (depending on EXTMODE2).	

2014/06/13

TOPPERSプロジェクト認定

108



int_switch_1 : VIC制御関連のレジスタの定義

- 割込み制御レジスタ

```
#define TREG_VICINTEN    ((volatile unsigned long *)0xFFFFF010)
#define TREG_VICSINTCLR ((volatile unsigned long *)0xFFFFF01C)
#define TREG_VICVCTADDR ((volatile unsigned long *)0xFFFFF100)
#define TREG_VICVCTPRIO ((volatile unsigned long *)0xFFFFF200)
```

- PINSELECT4レジスタ

```
#define TREG_PINSELECT4 ((volatile unsigned long *)0xE002C010)
```

- 割込み設定

```
#define INTNOEINT0  14  lpc23xx.hにて定義
#define INT0_INT_LVL 0x07
```

2014/06/13

TOPPERSプロジェクト認定

109



int_switch_1 : 外部割込み関係レジスタ定義

- 外部割込み制御レジスタ

```
#define TREG_EXTINT    ((volatile unsigned long *)0xE01FC140)
#define TREG_EXTMODE ((volatile unsigned long *)0xE01FC148)
#define TREG_EXTPOLAR ((volatile unsigned long *)0xE01FC14C)
```

- 割込み制御クリアビット定義

```
#define EINT0  0x00000001  /* EINT0割込みクリアビット */
#define EINT1  0x00000002  /* EINT1割込みクリアビット */
#define EINT2  0x00000004  /* EINT2割込みクリアビット */
#define EINT3  0x00000008  /* EINT3/GPIoint割込みクリアビット */
```

- 割込み要因選択レジスタビット定義

```
#define EINT0_EDGE 0x00000001 /* EINT0エッジ */
#define EINT1_EDGE 0x00000002 /* EINT1エッジ */
#define EINT2_EDGE 0x00000004 /* EINT2エッジ */
#define EINT3_EDGE 0x00000008 /* EINT3エッジ */
```

2014/06/13

TOPPERSプロジェクト認定

110



int_switch_1 : 割り込み制御関連のレジスタの初期化

- VICに割り込み設定を行うint_init関数を作成する
 - 優先度を引数とし, VICの割り込み制御レジスタと割り込み要因選択レジスタを初期化する
 - VICの割り込みクリア、割り込み有効設定を行う

```
void
int_init(unsigned int no, unsigned lvl, void *func){
    /* 割り込みハンドラアドレスを設定*/
    *(TREG_VICVCTADDR+no) = (unsigned long)func;
    /* 割り込み優先度を設定 */
    *(TREG_VICVCTPRIO+no) = (unsigned long)lvl;
    *TREG_VICINTCLR = 1<<no; /* 割り込みクリア */
    *TREG_VICINTEN = 1<<no; /* 割り込みを有効に */
}
```

2014/06/13

TOPPERSプロジェクト認定

111



int_switch_1 : 割り込み許可禁止設定

- start.Sにて、割り込み許可禁止関数を記載
 - Enable(); 割り込み許可
 - Disable(); 割り込み禁止

```
Enable:
    mrs r0,CPSR
    and r0,r0,#~0x80 /* I bit reset */
    msr CPSR,r0
    bx lr
Disable:
    mrs r0,CPSR
    orr r0,r0,#0x80 /* I bit set */
    msr CPSR,r0
    bx lr
```

2014/06/13

TOPPERSプロジェクト認定

112



int_switch_1: SW2用の割込み設定

- ポーリング型のSWITCH初期化関数を割込み型に改造
- プッシュボタンを押した際に割込みが入るよう立ち下がりエッジかつ片エッジを選択
- PINSELECTをEINT0に設定

```
void
switch_push_init(unsigned int no, unsigned int lvl, void *func){
    unsigned long reg;
    reg = *TREG_PINSEL4;
    reg &= ~0x00300000;
    reg |= 0x00100000;
    *TREG_PINSELECT4 = reg;
    int_init(no, lvl, func);
    *TREG_EXTMODE = EINT_EDGE0; /* EINT0エッジ設定 */
    *TREG_EXTPOLAR = 0; /* EINT0立下りエッジ */
    *TREG_EXTINT = EINT0; /* 割込みクリア */
}
```

引数に割込み条件を追加する

2014/06/13

TOPPERSプロジェクト認定

113



int_switch_1: 割込みハンドラ

- グローバル変数をカウントアップしてLEDに出力

```
void
int0_int_handler(void){
    *TREG_EXTINT = EINT0; /* 割込みクリア */
    /* LEDをカウントアップ */
    led_out_bdigit(++led_count);
}
```

- C言語関数を割込みハンドラとする場合の問題点
 - 単に関数を割込みハンドラとすると正しく動作しない
 - 割込みハンドラの出入り口では、元の処理の保存と復帰を行う必要がある

2014/06/13

TOPPERSプロジェクト認定

114



int_switch_1 : 割込みハンドラ #pragma等指定

- 割込みハンドラの出入り口処理
 - 元の処理に戻るための必要な情報(コンテキスト)は残す必要がある(スタックに)
 - 割込みハンドラから戻る場合は関数からのリターンとは異なる命令で戻る必要がある
- コンパイラによる出入り口処理の付加
 - 関数が割込みハンドラであることを通知すると, 割込みハンドラ用の出入り口処理を付加したアセンブリコードを生成可能なコンパイラが存在する
 - コンパイラへは一般的に#pragmaで通知する
 - ARM-KEILではALIAS設定

```
void int0_int_handler(void) ALIAS(IntDefaultHandler)
```

2014/06/13

TOPPERSプロジェクト認定

115



int_switch_1 : メインルーチン

- 初期化後, 割込みを許可して無限ループで割込みを待つ
- 割込みの許可はC言語では直接記述できない(プロセッサ毎に命令が異なる)ため, インラインアセンブラによりアセンブル言語による記述する
 - 記述方法はコンパイラによって異なるので注意が必要

```
void
_main(void){
    led_init();
    switch_push_init(INT0_EINT0, INT0_INT_LVL, int0_int_handler);
    led_count = 0;
    Enable();
    for (;;) {
    }
}
```

割込み許可関数

何も行わない
無限ループ

2014/06/13

TOPPERSプロジェクト認定

116



スイッチ割込みプログラム2 : int_switch_2 概要

- メイン関数でLEDを一定周期でカウントアップし, SW2 (EINT0)が入力されるとカウント値をクリアする
- 学習内容
 - メインルーチンと割込みハンドラ間でのデータの共有
 - 割込み禁止の必要性(クリティカルセクション)
- 動作確認
 - 作成済みのプログラムを実行し動作を確認する

2014/06/13

TOPPERSプロジェクト認定

117



int_switch_2 : データ共有と割込みハンドラ

メインルーチンと割込みハンドラ間でのデータの共有は,
グローバル変数(共有変数)で実現する

- LEDのカウントデータはグローバル変数とする

```
unsigned char led_data;
```

- 割込みハンドラではカウントデータをクリアする

```
void
int0_int_handler(void){
    *TREG_EXTINT = EINT0;
    led_data = 0;
    led_out_bdigit(led_data);
}
```

外部割込み要因のクリア

データのクリア

2014/06/13

TOPPERSプロジェクト認定

118



int_switch_2 : メイン関数

- led_dataのカウントアップは外部関数で行う
- 外部関数の戻り値(インクリメントした値)を led_data に入れる

```
void
_main(void){
    led_init();
    switch_push_init();
    int_init(INT0_ENT0, INT_0_INT_LVL, int0_int_handler);
    led_data = 0;
    Enable();
    for (;;) {
        led_data = inc_data(led_data);
        led_out_bdigit(led_data);
        busy_wait();
    }
}
```

LEDのカウント
アップ

2014/06/13

TOPPERSプロジェクト認定

119



int_switch_2 : カウントアップ関数

- 引数の値をインクリメントして返す
- busy_wait()により故意に時間を消費している
 - 関数の実行によりある一定の処理時間が必要であるという想定を模している
 - 外部関数がライブラリで提供されていると処理時間は分からない場合がある
 - busy_wait()は以前のプログラムの2倍の時間待つ

```
unsigned char
inc_data(unsigned char data){
    busy_wait();
    busy_wait();
    return (data + 1);
}
```

ダミーウェイト

2014/06/13

TOPPERSプロジェクト認定

120



int_switch_2 : プログラムの問題点

前述のプログラムには問題があり正しく動作しない

- SW2を押して割込みハンドラでLEDをクリアしても, 次のカウントのタイミングで'1'ではなく, 前のカウント値をインクリメントした値が表示される

実際に動作させて問題点を確認する

- メイン関数ではグローバル変数 led_data を引数に inc_data() を呼び出し, 戻り値を led_data に戻している
 ➡ グローバル変数の値を読み込んでから書き戻すまでに時間が必要

2014/06/13

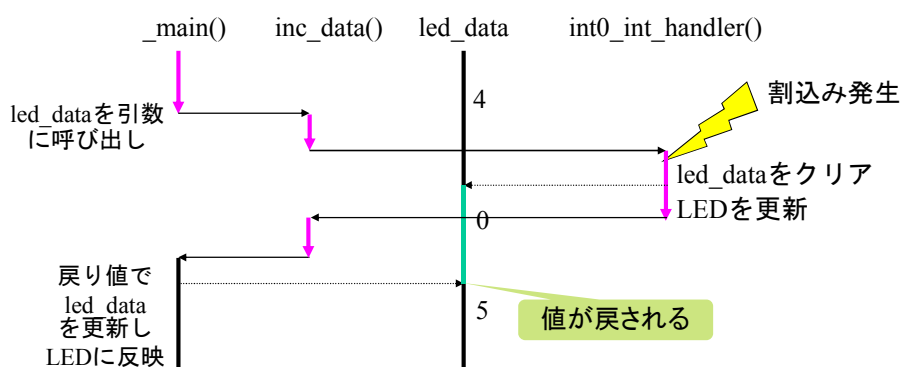
TOPPERSプロジェクト認定

121



int_switch_2 : led_dataの更新

- _main()関数でled_dataを読み込んでinc_data()を実行している間に, 割込みが発生し, 割込みハンドラでled_dataをクリアすると, その後_main()関数によって上書きされてしまう



2014/06/13

TOPPERSプロジェクト認定

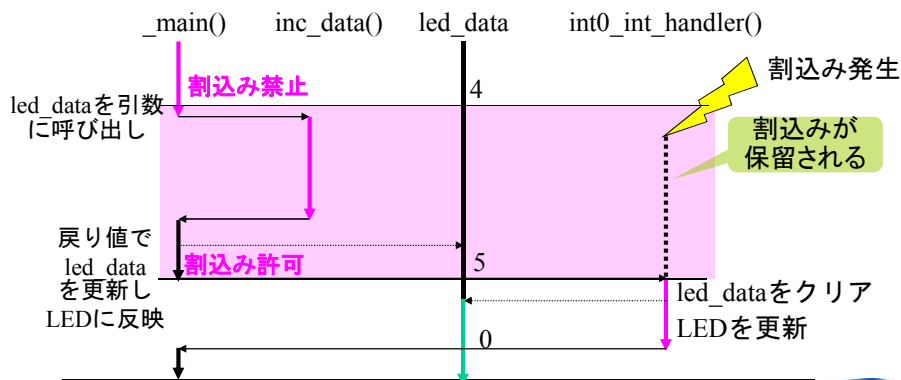
122



int_switch_2 : 割込みの禁止

- 共有データ(led_data)を排他的に扱えば問題は発生しない
- _main()関数でled_dataを更新する間はSW2の割込みがプロセッサによって受け付けられないようにすればよい

➡ 割込み禁止により実現



2014/06/13

TOPPERSプロジェクト認定

123



int_switch_2 : 割込みの禁止 プログラムでの実現

- 割込み禁止・許可命令で実現する
 - プロセッサ毎に実現方法は異なる
 - 割込みマスクで実現する方法も(後述)

割り込み禁止

```
void
_main(void){
  ...省略...
  for (;;) {
    Disable();
    led_data = inc_data(led_data);
    led_out_bdigit(led_data);
    Enable();
    busy_wait();
  }
}
```

割り込みが発生しても受け付けられず、割り込みハンドラと排他的に実行される

2014/06/13

TOPPERSプロジェクト認定

124



int_switch_2 : クリティカルセクション

排他的に動作する(しなければならない)区間

- 実現方法
 - メイン関数と割込みハンドラ間や割込みハンドラ間は、割込み禁止・許可命令を組み合わせることで実現
- クリティカルセクションの実行時間
 - 長くなると、割込みが発生してから応答するまでの時間(割り込み応答時間)が長くなり、リアルタイムシステムでは問題となる場合がある
 - 修正したプログラムでは、スイッチを押してからクリアされるまでのタイムラグが発生する

2014/06/13

TOPPERSプロジェクト認定

125



スイッチ割込みプログラム3 : int_switch_3 概要

- SW2を押すと発生する割込み(EINT0)により、LEDをカウントアップし、UART0のデータ受信で発生する割込み(INT_UART0)によりLEDをカウントダウンする
- 学習内容
 - 複数の割込みの扱い
 - 割込み関連のマニュアル読み
 - 割込み優先度
- 作成方法
 - int_switch_1 をベースに開発

2014/06/13

TOPPERSプロジェクト認定

126



int_switch_3 : 作成

プログラム int_switch_3 を作成

- ポートの初期化, SW2の割込み関連の初期化は流用する
- 手順
 - 割込み要因ハードウェア (INT_UART0) の整理
 - 割込み関連のレジスタの整理
 - 割込みの初期化はint_init()関数の設定をUART0用の引数に変えればよい, 加えてUART0の受信割込みを許可しなければならない (INT3)と比較して初期化項目が多い
 - プログラミング
 - 割込み関連のレジスタの定義
 - 割込み関連の初期化
 - UART0のデバイスの初期化
 - UART0割込みハンドラ
 - 割込みベクタテーブルへの登録

2014/06/13

TOPPERSプロジェクト認定

127



int_switch_3 : 割込み関連のハードウェア整理

- 割込み要因ハードウェア (UART0) の整理
 - 送信ラインTX0はGPIO0:2, 受信ラインRX0はGPIO0:3の共有ラインに接続
 - PINSEL0をTX0:RX0に変更する必要がある
 - その後, UART0デバイスの初期化を行う
- 割込み関連のレジスタの整理 (VIC ユーザーマニュアル:Chapter6)
 - 割込み優先度
 - 割込み制御レジスタ(VICVCTPRIO6)により設定
 - 割込み要因選択レジスタ
 - UA0_IERレジスタのERXI(ビット0)を'1'に設定
 - VICの割込み要因レジスタの設定
 - 割込みベクタ
 - 割込みアドレスレジスタ(VICADDR6)にハンドラアドレスを設定

2014/06/13

TOPPERSプロジェクト認定

128



int_switch_3 : 割込み関連のレジスタの定義

- 割込み制御レジスタ
 - 割込みレジスタの設定はint_init関数を使用

- PINSELECT4レジスタ(uartの実習で使用)

```
#define TREG_PINSELECT4 ((volatile unsigned long *)0xE002C010)
```

- 割込み設定

```
#define INTNO_UART0 6    lpc23xx.hにて定義
#define UART0_INT_LVL 0x04
```

2014/06/13

TOPPERSプロジェクト認定

129



UART0の初期設定と割込み

- ポーリングのUARTの設定と同様
- 初期化関数uart0_init()はそのまま使用する
- UART0の受信割込み関数を作成し、関数内でLEDの表示と受信データの取出しと、データのエコー送信を行う
 - 受信はuart0_pol_getc関数が使用可能
 - 送信はuart0_pol_putc関数が使用可能
- UART0割込み設定レジスタ

```
#define TREG_UART0_IER ((volatile unsigned long *)0xE000C004)
```

- 割込み設定

```
#define US_ETXI 0x02    送信割込み設定
#define US_ERXI 0x01    受信割込み設定
```

2014/06/13

TOPPERSプロジェクト認定

130



int_switch_3 : 割り込み制御関連のレジスタの初期化

- 割り込み番号、優先度、割り込みハンドラアドレスを引数とし、割り込みレジスタと割り込み要因選択レジスタを初期化する
 - VICの設定はint_init()関数を使用する
- 受信割り込みを有効にするようにUART0の割り込み要因レジスタをセットする

```
void
uart0_rx_int_init(int no, unsigned char lvl, void *func){
    int_init(no, lvl, func);
    *TREG_UART0_IER = US_ERXI;    /* 割り込みを有効化 */
}
```

2014/06/13

TOPPERSプロジェクト認定

131



int_switch_3 : 割り込みハンドラとベクターテーブルへの登録

- グローバル変数(led_count)をデクリメントしてLEDに出力
- 受信文字をエコーバックする
- SW2の割り込みハンドラとの排他制御

```
void
uart0_rx_int_handler(void){
    unsigned char c;
    /* LEDをカウントダウン */
    led_out_bdigit(--led_data);
    c = uart0_pol_getc();
    uart0_pol_putc(c);
}
```

2014/06/13

TOPPERSプロジェクト認定

132



int_switch_3 : 割込み優先度

割込み優先度を学ぶ

- 割込みを抑止する方法として割込み優先度を用いる
 - 割込み要因毎に優先度を設定し、プロセッサの優先度より高い割込みのみを受付ける
 - 割込み優先度の概念がなく、割込みを個別に禁止・許可する機能しかもたないプロセッサも存在
- 優先度を適切に設定すると特定の割込みを禁止できる
 - 重要な割り込みのみを受付ける
 - 多重割込みとの組み合わせ
- 割込み優先度の設定
 - プロセッサの特殊レジスタ
 - 割込みコントローラで設定

2014/06/13

TOPPERSプロジェクト認定

133



int_switch_3 : LPC2388の割込み優先度

- 割込み要因の優先度
 - 各割込み要因は、割込み制御レジスタにより割込み優先度を0～15の範囲で設定可能
 - 割込み関連初期化関数の引数で設定
- VICの割込み優先度
 - 各割込みに対してVICVCTPRIONレジスタにセット
 - VICの割込み発生し、クリアされない限り、その割込み以下の優先度の割込みが禁止される
- int_switch_3では、uartの割込みが6、SW2の割込みが7に設定しているためSW2の割込み中に、uartの割込みを受け付ける可能性がある

2014/06/13

TOPPERSプロジェクト認定

134



int_switch_3 : 割り込み優先度 メイン関数

- sw2,uart0の初期化、割り込み設定後、割り込みを許可する

```
void
_main(void){
    led_init();
    switch_push_init();
    int_init(INTNO_EINT0, INT0_INT_LVL, int0_int_handler);
    uart0_init();          /* UART0の初期化 */
    uart0_rx_int_init(INTNO_UART0, UART0_INT_LVL, uart0_rx_int_handler);
    led_data = 0;

    Enable();              /* 割り込み許可 */

    for (;;) {
    }
}
```

2014/06/13

TOPPERSプロジェクト認定

135



タイマとシリアルI/Oの割り込み対応

- Appendixとして2つのデバイスの割り込み演習を用意しました
 - タイマを割り込みで動作させるint_timer
 - シリアルI/Oを割り込みで動作させるint_uart
- Appendixのテキストを参考に演習を行ってください

2014/06/13

TOPPERSプロジェクト認定

136



まとめ

小規模組込み開発のまとめ

- 小規模な組込み開発では少数の開発者でファームウェア、ソフトウェアの開発を行うため、ハードウェアの知識や開発環境を構築する知識も必要となる
- 開発言語はC言語を用いることが多いが、C言語で記述できない部分は、アセンブラ言語で開発する必要がある
- 開発環境はプロセッサメーカーの開発環境を用いることが多く、開発前にノウハウの蓄積が必要
- ハードウェアの制御は、ポーリングを用いるのもの、割り込みを用いて制御を行うものがあり、適切な方式を選択する必要がある
- 割り込み制御はプロセッサ毎に異なる。特別な設定を行う必要があり、処理を熟知して開発を行う必要がある

小規模組込み開発のまとめ

- アプリケーション部分の開発については、オブジェクト指向やモデル設計を用いれば、可視的なソフトウェア開発ができる。
- このような開発は実行ROM、RAMサイズの増加を伴うことが多いので注意が必要
- NPO法人 組込みソフトウェア管理者・技術者育成研究会では、いろいろは組込みソフトウェア開発手法の紹介を行ってますので、組込み開発プロセスの参考にしてください。 <http://www.sesame.jp/>

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

- メモリマップドレジスタの操作方法の確認
 - 本田 晋也(名古屋大学)
- ポーリングプログラミング
 - 本田 晋也(名古屋大学)
- 割込みプログラミング
 - 本田 晋也(名古屋大学)
- おまけ:カップラーメンタイマの作成
 - 本田 晋也(名古屋大学)
- LPC2388用のプログラム作成、テキストの修正
 - 竹内良輔((株)リコー)