TOPPERS基礎実装セミナー

（STM32F401/446 Nucleo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

TOPPERS

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

※ 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．

本ドキュメントに関して

1. 著作権に関しての表記

<TOPPERS基礎実装セミナー(STM32F4-01/446 Nucleo-64版:基本)2日目>

---

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。

TOPPERS

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

概要

- マイコンボードを用いて
  組込みプログラム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

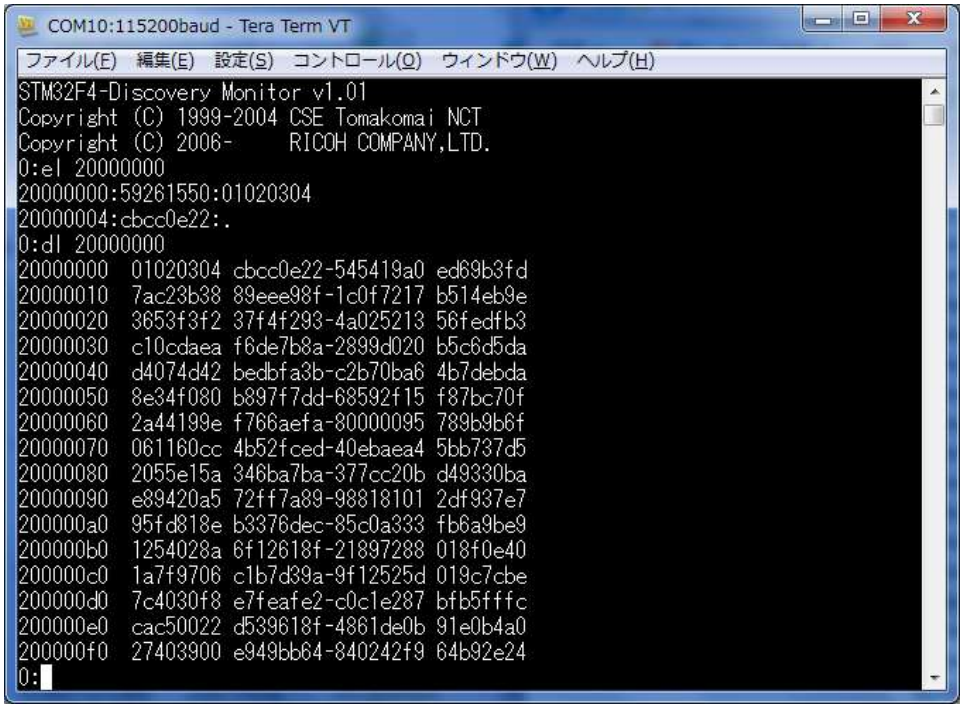
1. ROMモニタの操作
2. LEDの接続
3. LEDの操作

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
- Cortex-M4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



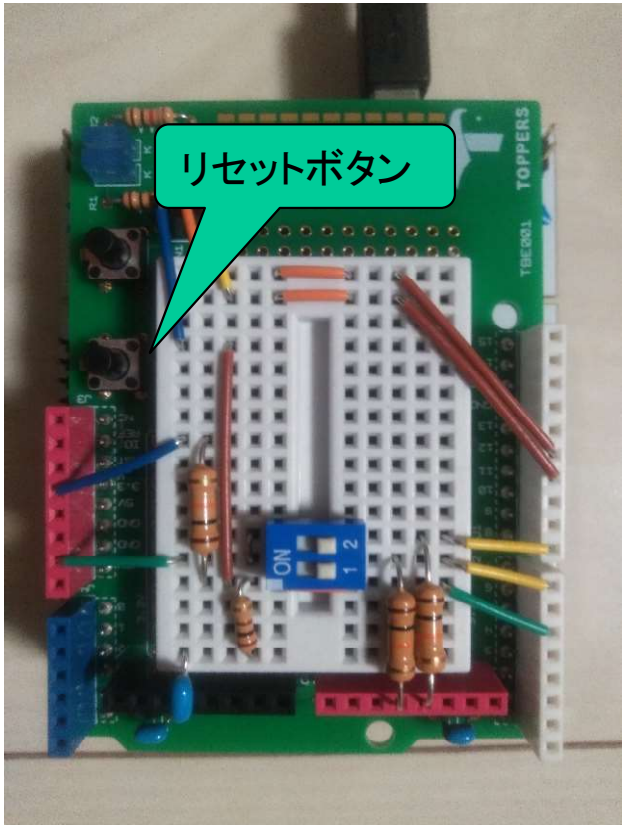
TOPPERS

ROMモニタの操作方法

- ST-LINK USBをケーブルでパソコンに接続
- デバイスマネージャでCOMポートを確認
  - TeraTermを立ち上げ
  - TeraTermを設定

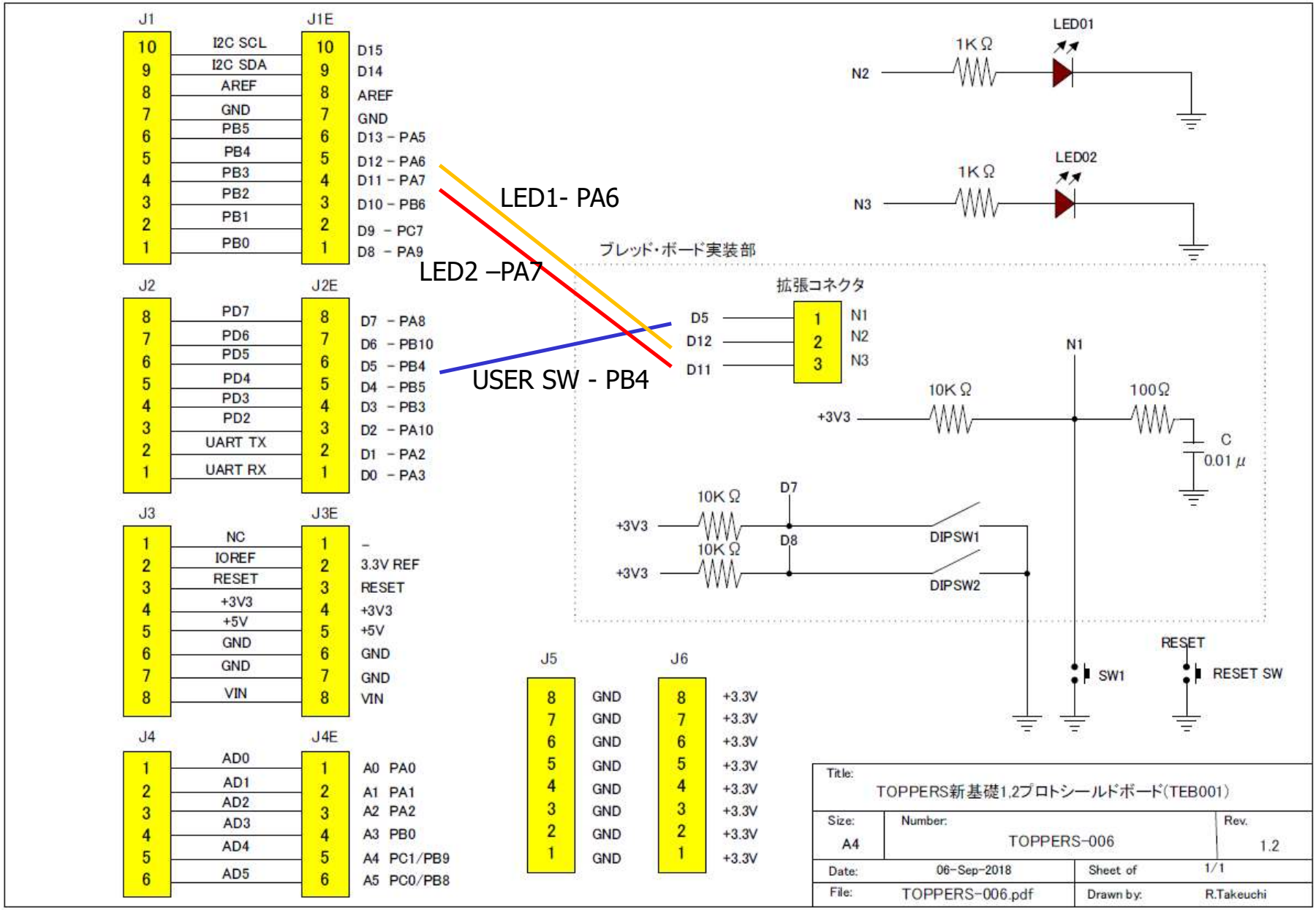
  を確認して、一度リセットボタンを押す

- 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



TOPPERS

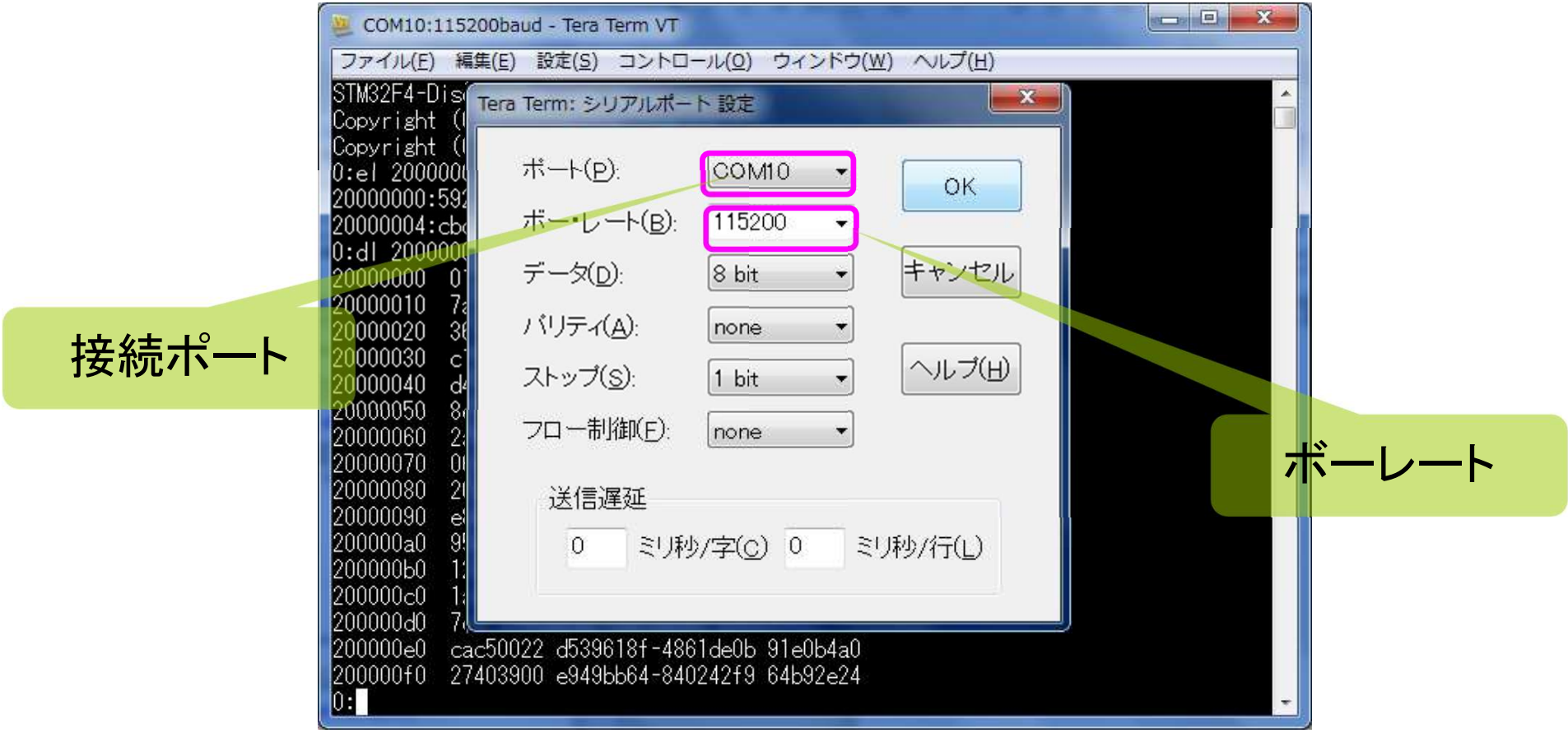
プロトタイピング・シールド回路図



TOPPERS

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

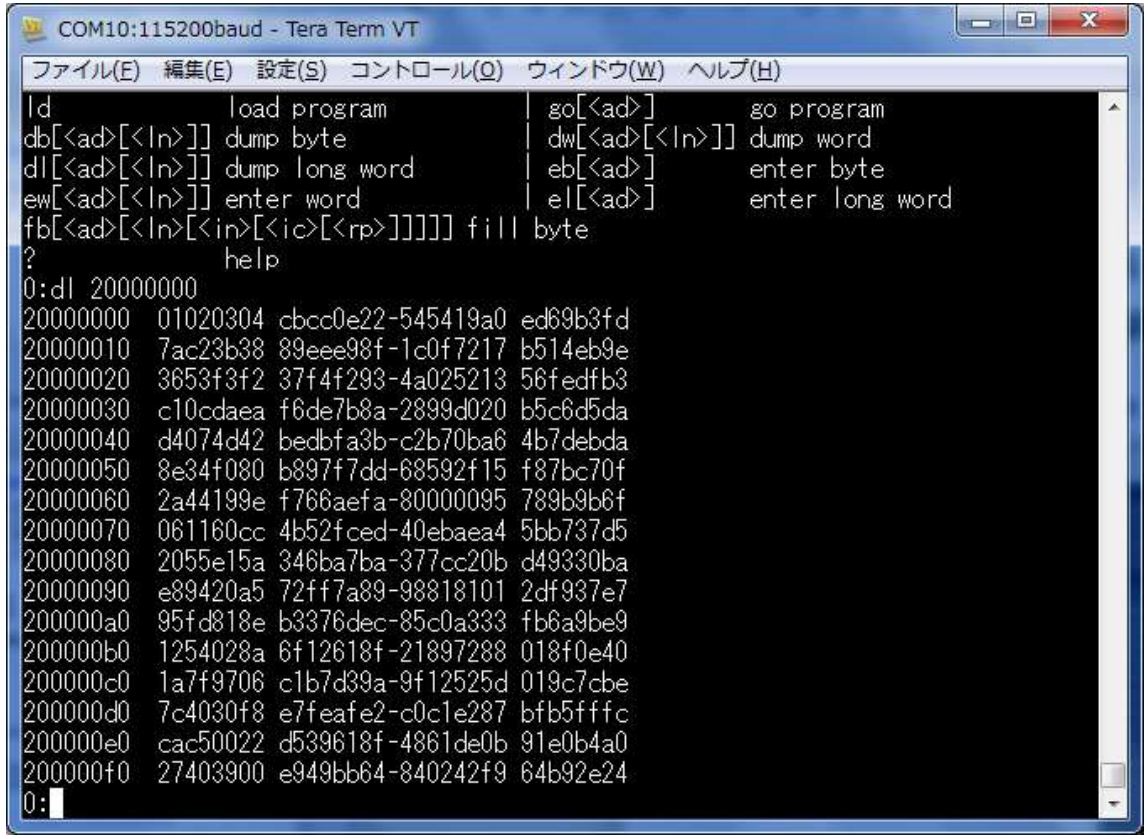


TOPPERS

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> enter

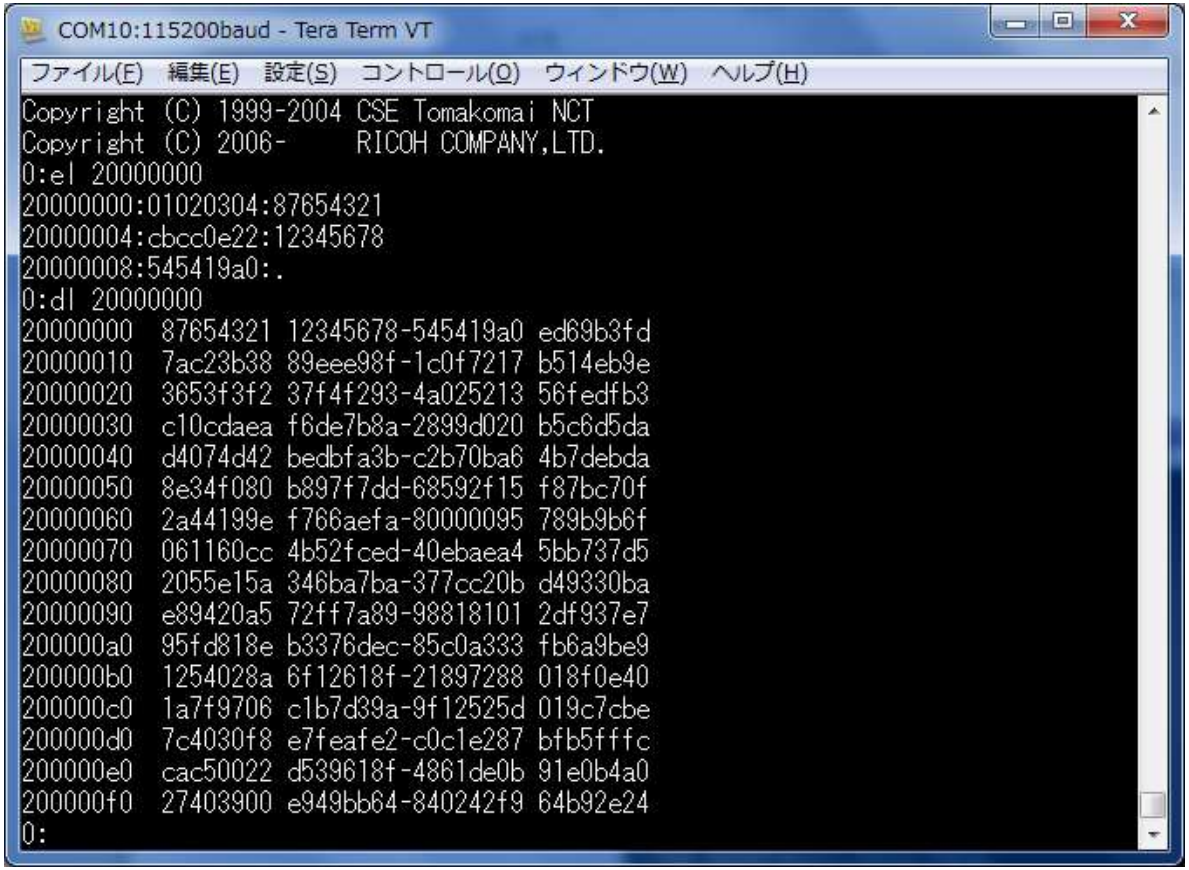


TOPPERS

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
- '.' enterにて、データ入力モードを終了します

RAM領域の
0x20000000番地と
0x20000004番地に
0x87654321と
0x12345678を書き込む



TOPPERS

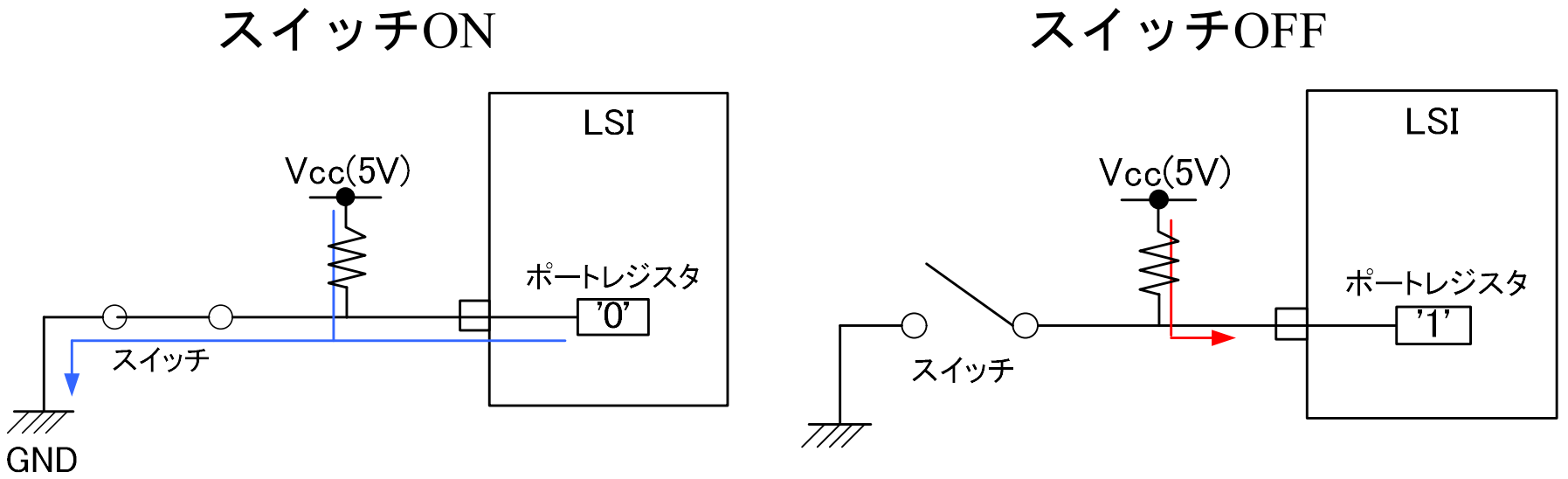
周辺デバイス：プルアップとプルダウン

- 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，入力を電気的に安定させるための回路構成

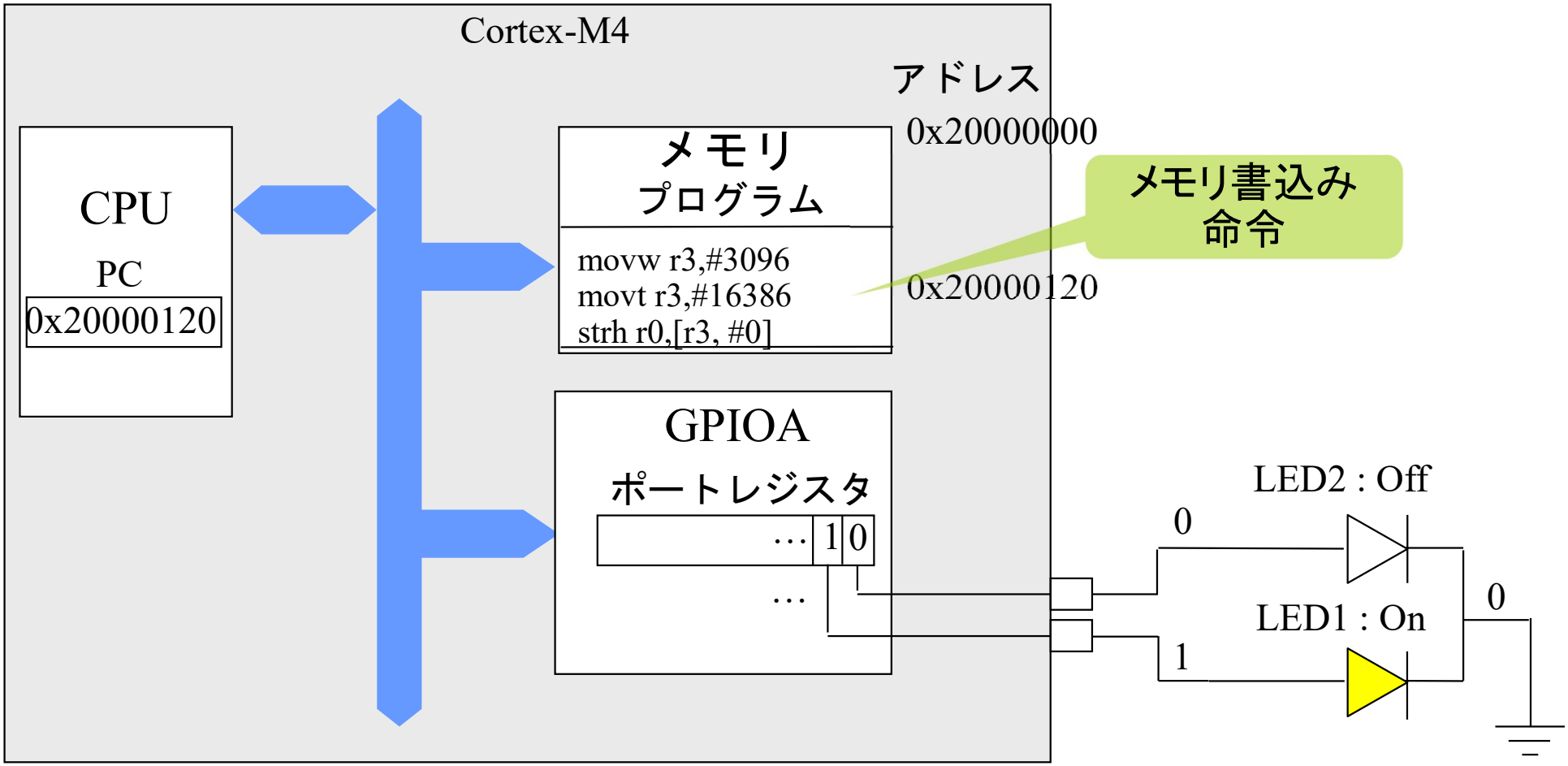
プルアップの例



周辺デバイス：プロセッサとLEDとの接続の関係

プロトタイピング・シールドの例

– ポートレジスタの値が1でLEDが点灯する

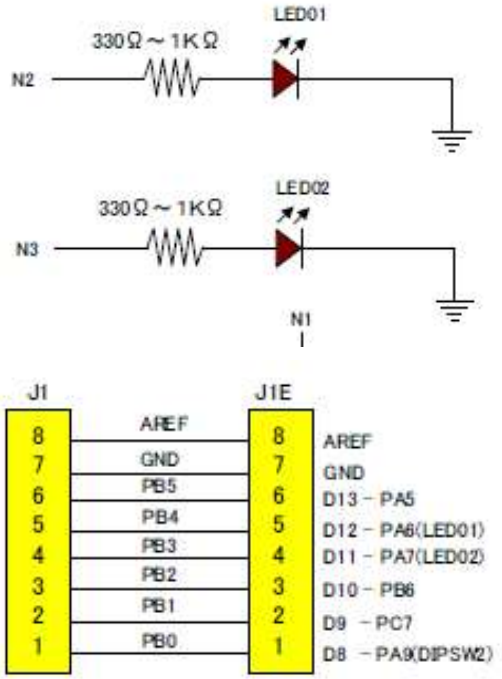


TOPPERS

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”1”を出力すると点灯する

ボード



Arduinoコネクタ

Right connectors				
CN5 digital	10	D15	PB8	I2C1_SCL
	9	D14	PB9	I2C1_SDA
	8	AREF	-	AVDD
	7	GND	-	Ground
	6	D13	PA5	SPI1_SCK
	5	D12	PA6	SPI1_MISO
	4	D11	PA7	TIM1_CH1N or SPI1_MOSI
	3	D10	PB6	TIM4_CH1 or SPI1_CS
	2	D9	PC7	TIM3_CH2
	1	D8	PA9	-

LED1-PA6に接続 LED2-PA7に接続

LEDの接続：レジスタ構成

- LEDはポートGPIOAに接続：初期化用
  - モードレジスタ：GPIOA_MODER　0x40020000
  - スピードレジスタ：GPIOA_OSPEEDR　0x40020008
  - 出力タイプレジスタ：GPIOA_OTYPER　0x40020004
  - プルアップダウンレジスタ：GPIOA_PUPUDR　0x4002000C

Table 39. GPIO register map and reset values

Offset	Register	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0x00	GPIOx_MODER (where x = C..I/J/K)	MODER15[1:0]		MODER14[1:0]		MODER13[1:0]		MODER12[1:0]		MODER11[1:0]		MODER10[1:0]		MODER9[1:0]		MODER8[1:0]		MODER7[1:0]		MODER6[1:0]		MODER5[1:0]		MODER4[1:0]		MODER3[1:0]		MODER2[1:0]		MODER1[1:0]		MODER0[1:0]	
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x04	GPIOx_ OTYPER (where x = A..I/J/K)	Reserved																OT15	OT14	OT13	OT12	OT11	OT10	OT9	OT8	OT7	OT6	OT5	OT4	OT3	OT2	OT1	OT0
	Reset value																	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x08	GPIOx_ OSPEEDER (where x = A..I/J/K except B)	OSPEEDR15[1:0]		OSPEEDR14[1:0]		OSPEEDR13[1:0]		OSPEEDR12[1:0]		OSPEEDR11[1:0]		OSPEEDR10[1:0]		OSPEEDR9[1:0]		OSPEEDR8[1:0]		OSPEEDR7[1:0]		OSPEEDR6[1:0]		OSPEEDR5[1:0]		OSPEEDR4[1:0]		OSPEEDR3[1:0]		OSPEEDR2[1:0]		OSPEEDR1[1:0]		OSPEEDR0[1:0]	
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x0C	GPIOx_PUPDR (where x = C..I/J/K)	PUPDR15[1:0]		PUPDR14[1:0]		PUPDR13[1:0]		PUPDR12[1:0]		PUPDR11[1:0]		PUPDR10[1:0]		PUPDR9[1:0]		PUPDR8[1:0]		PUPDR7[1:0]		PUPDR6[1:0]		PUPDR5[1:0]		PUPDR4[1:0]		PUPDR3[1:0]		PUPDR2[1:0]		PUPDR1[1:0]		PUPDR0[1:0]	
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

TOPPERS

LEDの接続:レジスタ構成

- データ設定レジスタ
  - インプットデータレジスタ　　0x40020010
  - アウトプットデータレジスタ　　0x40020014
  - ビットセットリセットレジスタ　　0x40020018
- GPIOAに16ビットの対応ピンがある:BSRRを使用
  - 1を書き込んだビットをリセット:0、セット:1に変更
  - ビットセット　　0x40020018:16ビット
  - ビットリセット　　0x4002001A:16ビット

Table 39. GPIO register map and reset values (continued)

Offset	Register	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0x0C	GPIOx_PUPDR (where x = C..I/J/K)	PUPDR15[1:0]		PUPDR14[1:0]		PUPDR13[1:0]		PUPDR12[1:0]		PUPDR11[1:0]		PUPDR10[1:0]		PUPDR9[1:0]		PUPDR8[1:0]		PUPDR7[1:0]		PUPDR6[1:0]		PUPDR5[1:0]		PUPDR4[1:0]		PUPDR3[1:0]		PUPDR2[1:0]		PUPDR1[1:0]		PUPDR0[1:0]	
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x10	GPIOx_IDR (where x = A..I/J/K)	Reserved																IDR15	IDR14	IDR13	IDR12	IDR11	IDR10	IDR9	IDR8	IDR7	IDR6	IDR5	IDR4	IDR3	IDR2	IDR1	IDR0
	Reset value																	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
0x14	GPIOx_ODR (where x = A..I/J/K)	Reserved																ODR15	ODR14	ODR13	ODR12	ODR11	ODR10	ODR9	ODR8	ODR7	ODR6	ODR5	ODR4	ODR3	ODR2	ODR1	ODR0
	Reset value																	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x18	GPIOx_BSRR (where x = A..I/J/K)	BR15	BR14	BR13	BR12	BR11	BR10	BR9	BR8	BR7	BR6	BR5	BR4	BR3	BR2	BR1	BR0	BS15	BS14	BS13	BS12	BS11	BS10	BS9	BS8	BS7	BS6	BS5	BS4	BS3	BS2	BS1	BS0
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

TOPPERS

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
  - GPIOAのクロック有効化(0x40023830)の0対応ビットを1
  - モードレジスタ（0x40020000）の7,6対応ビットに01:General purpose output modeを設定
  - 出力タイプレジスタ（0x40020004）の7,6に0:output push pull
  - スピードレジスタ(0x40020008)の7,6対応ビットに10:fast speed
  - プルアップダウン(0x400200C)の7,6対応ビットに01:pull up
- レジスタへ書き込む値（?は変更させない）
  - 0x40023830: 0x00000001(元の値：0x00000001)→変更不要
  - 0x40020000 : 0x????5???（“???? ???? 0101 ???? ???? ????"）
  - 0x40020004 : 0x??????0?(“???? ???? ???? ???? 00?? ????”)
  - 0x40020008 : 0x????A???（“???????? 1010 ???? ???? ????"）
  - 0x4002000C: 0x????5???(“???? ???? 0101 ???? ???? ????”)

TOPPERS

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. LED2:OFF LED1:OFF

2. LED2:ON LED1:ON

3. LED2:OFF LED1:ON

4. LED2:ON LED1:OFF

LEDの操作 : 任意のLEDの点灯

1. LED2:OFF LED1:OFF
   - RESET(0x4002001a):0x00c0 : “0000 0000 1100 0000"
2. LED2:ON LED1:ON
   - SET(0x40020018):0x00c0 : “0000 0000 1100 0000“
3. LED2:OFF LED1:ON
   - SET(0x40020018):0x0040 : "0000 0000 0100 0000“
   - RESET(0x4002001a):0x0080 : “0000 0000 1000 0000”
4. LED2:ON LED1:OFF
   - SET(0x40020018):0x0080 : “0000 0000 1000 0000“
   - RESET(0x4002001a):0x0040 : “0000 0000 0100 0000”

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

周辺デバイスを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

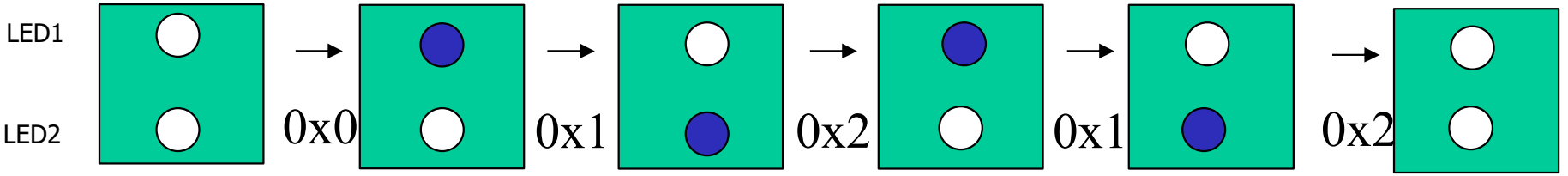
ポーリングプログラミング：プログラムファイル

- プログラムファイルの置き場所
  - 教材ディレクトリ/base1/program
  - MSYS2のホームディレクトリにtoppersを作り、base1をtoppersの下にコピーしてください
- 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_shift以外は、回答となるため、プログラムは用意しない

LEDプログラム : led_shift 概要

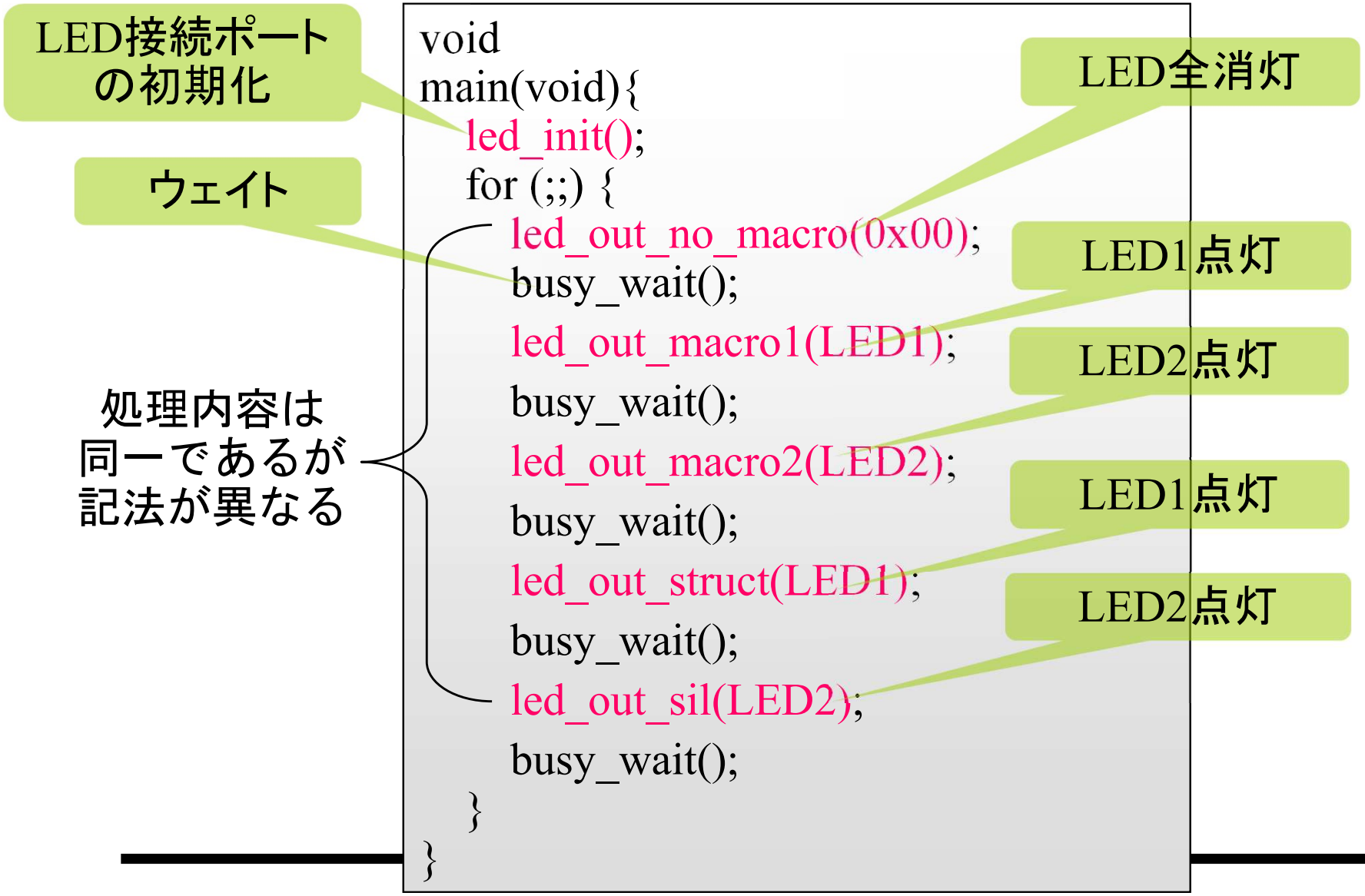
- 次のパターンでLEDを点灯させる
  - LED全てOFF → LED1 ON → LED2 ON → LED1 ON → LED2 ON → LED全てOFF



- 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/startup_stm32f4xx.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_shift : ポートレジスタへの書き込み

- LED点灯・消灯制御
  - GPIOAのBSRRレジスタ（0x40020018,0x4002001A）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

```
                movw   r2,  #26  /* 0x01a */
                movt    r2,  #16386 /* 0x4002 */
                mov.w  r1,  #192 /* 0x00c0 */
書き込み   :  strh      r1,  [r2, #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 short *)(0x4002001a)) = 0x00c0;
読み込み    :  tmp = *((unsigned long *)(0x40020014));
```

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指定をつける必要ある

```
書き込み　: *((volatile unsigned short *)(0x4002001a)) = 0x00c0;
読み込み　:  tmp = *((volatile unsigned long *)(0x40020014));
```

- 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 )(0x40020014));
```


TOPPERS

led_shift : ポートレジスタ書き込み関数

- LED点灯・消灯制御
  - GPIOAのBSRRレジスタ（0x40020018）への書き込み
  - アクセスサイズが2byte×2なので，unsigned short 型のポインタへキャストして書き込む

```
void led_out(unsigned short led_data){
    unsigned short reg1 = ~led_data;
    unsigned short reg2 = led_data;
    *((volatile unsigned short *) 0x4002001A) = reg1;    /* 書き込み*/
    *((volatile unsigned short *) 0x40020018) = 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GPIOA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、このコードの実行により発生する不具合は?

TOPPERS

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Memory 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

STM32F4xxのポートレジスタの仕様(users manual chapter 10から引用)

- 設定を直接参照する場合は、GPIO port output data registerを使用します
- このレジスタは、設定値の保存や読み出しができます

- 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

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(LED1:ON): 0x4160 (“0100 0001 0110 0000")
  - 引数led_data(LED2:ON) : 0x0080 (“0000 0000 1000 0000")
  - 書き込みたい値は : 0x41A0 (“0100 0001 1010 0000")
- プログラムによる0x41A0の生成手順
  1. ポートレジスタの値のビット7,6をクリア
     （0x4120（“0100 0001 0010 0000”））
  2. led_dataのビット7,6のみを取り出す
  3. 1.と2.で生成した値の論理和を生成
     （“0100 0001 0010 0000”|“0000 0000 1000 0000”
     = “0100 0001 1010 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

led_shift : led_outの変更

- ポートレジスタのビット7,6以外は元の値を保持

STM32F4xxの場合、SET、RESET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 short led_data){
    unsigned short reg1, reg2;

    reg1 = ~led_data & 0x00c0;
    reg2 = led_data & 0x00c0;

    *((volatile unsigned short *)0x4002001A) = reg1;
    *((volatile unsigned short *)0x40020018) = reg2;
}
```

1. オフ(0)にするビットを取り出す

2. オン(1)にするビットを取り出す

3. BSSRにオンにするデータを書き込む

4. BSRRにオフにするデータを書き込む

TOPPERS

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

- GPIOAレジスタのマクロ定義

```
#define   TADR_GPIOA_CLR      0x4002001A
#define   TADR_GPIOA_SET      0x40020018
```

- GPIOAのLEDの接続ビット定義

```
#define PINPOSITION6   6
#define PINPOSITION7   7

#define   LED1          (1<<PINPOSITION6)          /* LED1ビット定義 */
#define   LED2          (1<<PINPOSITION7)          /* LED2ビット定義 */
#define   LED_MASK      (LED1 | LED2)
```


TOPPERS

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 short led_data){
    unsigned short reg1, reg2;

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

   *((volatile unsigned short *)TADR_GPIOA_CLR) = reg1;
   *((volatile unsigned short *)TADR_GPIOA_SET) = reg2;
}
```

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

- GPIOAレジスタ

```
#define TREG_GPIOA_CLR   ((volatile unsigned short*)TADR_GPIOA_CLR)
#define TREG_GPIOA_SET  ((volatile unsigned short*)TADR_GPIOA_SET)
```

- ポインタ部までマクロ化(プログラム例はない)

```
#define TREG_GPIOA_CLR *((volatile unsigned short*)TADR_GPIOA_CLR)
#define TREG_GPIOA_SET *((volatile unsigned short*)TADR_GPIOA_SET)
```

led_shift : マクロによるled_outの書き換え2

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

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

   *TREG_GPIOA_CLR = reg1;
   *TREG_GPIOA_SET = reg2;
}
```

- 1行での記述も可能

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

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 {
  char  b0:1;
  char  b1:1;
  char  b2:1;
  char  b3:1;
  char  b4:1;
  char  b5:1;
  char  b6:1;
  char  b7:1;
};
```

ビット
フィールド

```
struct byte_def{
    struct bit_def bit0;
    struct bit_def bit1;
};
```

```
/* ワード構造体 */
union half_def{
    struct byte_def  byte;
    unsigned short half;
};
```


TOPPERS

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 short led_data){
    union half_def  reg1, reg2;

    reg1.half = 0;
    reg2.half = 0;
    reg1.byte.bit0.b7 = ((LED2 & led_data) == 0)? 1 : 0;
    reg2.byte.bit0.b7 = ((LED2 & led_data) != 0)? 1 : 0;
    reg1.byte.bit0.b6 = ((LED1 & led_data) == 0)? 1 : 0;
    reg2.byte.bit0.b6 = ((LED1 & led_data) != 0)? 1 : 0;
    *TREG_GPIOA_CLR = reg1.half;
    *TREG_GPIOA_SET = reg2.half;
}
```


TOPPERS

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

- ハーフデータの読み込みと書き込み

```
unsigned short
sil_reh_mem(unsigned int addr){
    return *((volatile unsigned short *)addr);
}
```

```
void
sil_wrh_mem(unsigned int addr, unsigned short bdata){
    *((volatile unsigned short *)addr) = bdata;
}
```


TOPPERS

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 short led_data){
    unsigned short reg1, reg2;

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

    sil_wrh_mem(TADR_GPIOA_CLR, reg1);
    sil_wrh_mem(TADR_GPIOA_SET, reg2);
}
```

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;
    unsigned char pinpos;
    *TREG_RCC_AHB1ENR |= RCC_AHB1ENR_GPIOAEN;
    for(pinpos = PINPOSITION6 ; pinpos <= PINPOSITION7 ; pinpos++){
        reg = *TREG_GPIOA_MODER & ~(3 << (pinpos *2));
        *TREG_GPIOA_MODER = reg | (GPIO_Mode_OUT << (pinpos * 2));
        reg = *TREG_GPIOA_OSPEEDR & ~(3 << (pinpos * 2));
        *TREG_GPIOA_OSPEEDR = reg | (GPIO_Speed_50MHz << (pinpos *2));
        *TREG_GPIOA_OTYPER |= (GPIO_Otype_PP << pinpos);
        reg = *TREG_GPIOA_PUPDR & ~(3 << (pinpos * 2));
       *TREG_GPIOA_PUPDR = reg |  (GPIO_PuPd_UP << (pinpos * 2));
       *TREG_GPIOA_CLR = 1 << pinpos;
}
```

GPIOAのクロック スタート

LEDの位置で LOOP

TOPPERS

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

TOPPERS

led_shift : startup_stm32f4xx.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、ROM上に配置
  - RAM実行の場合、RAM上に配置
    - ベクタテーブルの設定アドレスを切り替え
- 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 : startup_stm32f4xx.S（ROM実行）

- リンクファイル：stm32f4xx_rom.ldで指定

```
; ------- RAM 領域 -------------
    ORG          0x20000000
    SECTION  .data
    SECTION  .bss
```

```
; ------- ROM 領域 -------------
    ORG          0x08000000
    SECTION  .vector
    SECTION  .text
    SECTION  .data
```

セクション

- .vector　　:割込みベクタ、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

TOPPERS

led_shift : startup_stm32f4xx.S（RAM実行）

- リンクファイルstm32f4xx_ram.ldで指定

```
; ------- RAM 領域 -------------
    ORG          0x20000000
    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　　:割込みベクタ、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上にダウンロードするため、ROM領域からの.dataのコピーは不要
- .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

led_shift : startup_stm32f4xx.S 固定ベクタ

- Cortex-M4はリセット後，0x08000004番地のベクタから実行を始める
- ROM実行の場合、0x08000004にリセット関数(Reset_Handler)へのポインタを記載する
- 0x0800000番地にリセット時のmspのアドレスを置く

```
g_pfnVectors:
  .word  _estack
  .word  Reset_Handler
  .word  NMI_Handler
  .word  HardFault_Handler
  .word  MemManage_Handler
  .word  BusFault_Handler
  .word  UsageFault_Handler
  .word  0
  .word  0
  .word  0
  .word  0
  .word  SVC_Handler
  .word  DebugMon_Handler
```

RESET時のスタックポインタ

リセット時の関数ポインタ

TOPPERS

startup_stm32f4xx.S 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

```
 Reset_Handler:
/* Copy the data segment initializers from flash to SRAM */
  movs  r1, #0
  b  LoopCopyDataInit

CopyDataInit:
  ldr  r3, =_sidata
  ldr  r3, [r3, r1]
  str  r3, [r0, r1]
  adds  r1, r1, #4

LoopCopyDataInit:
  ldr  r0, =_sdata
  ldr  r3, =_edata
  adds  r2, r0, r1
  cmp  r2, r3
  bcc  CopyDataInit
```

_sdataから_edataまでに
_sidataをコピー

TOPPERS

startup_stm32f4xx.S セクションの初期化

```
 ldr  r2, =_sbss
  b  LoopFillZerobss
/* Zero fill the bss segment. */
FillZerobss:
  movs  r3, #0
  str  r3, [r2], #4

LoopFillZerobss:
  ldr  r3, = _ebss
  cmp  r2, r3
  bcc  FillZerobss

/* Call the clock system intitialization function.*/
  ldr    r0, =hardware_init_hook
  cmp    r0, #0x00
  beq    start_0
  blx    r0
```

_sbssから_ebss
までをゼロクリア

hardware_init_hookがリンクされていれば実行

TOPPERS

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

```
/* Call the application's entry point.*/
start_0:
  bl  _main
  bx  lr
.size  Reset_Handler, .-Reset_Handler
```

- 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

ポーリングプログラム

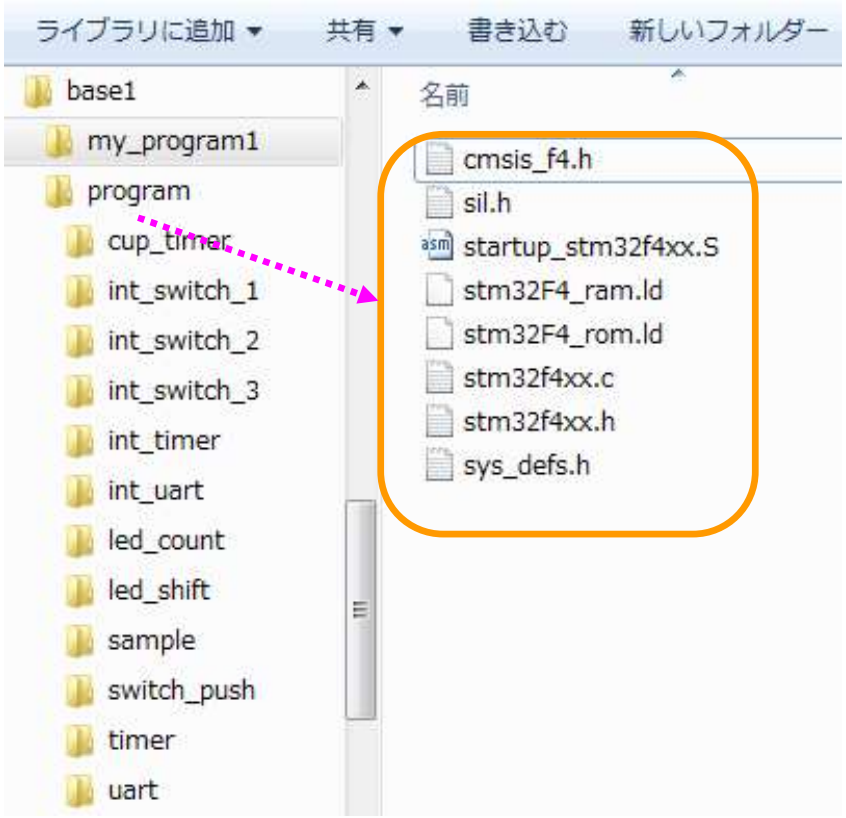
1. LEDプログラム
   - プロジェクトの作成方法
2. スイッチプログラム
3. タイマプログラム
4. シリアルI/Oプログラム

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

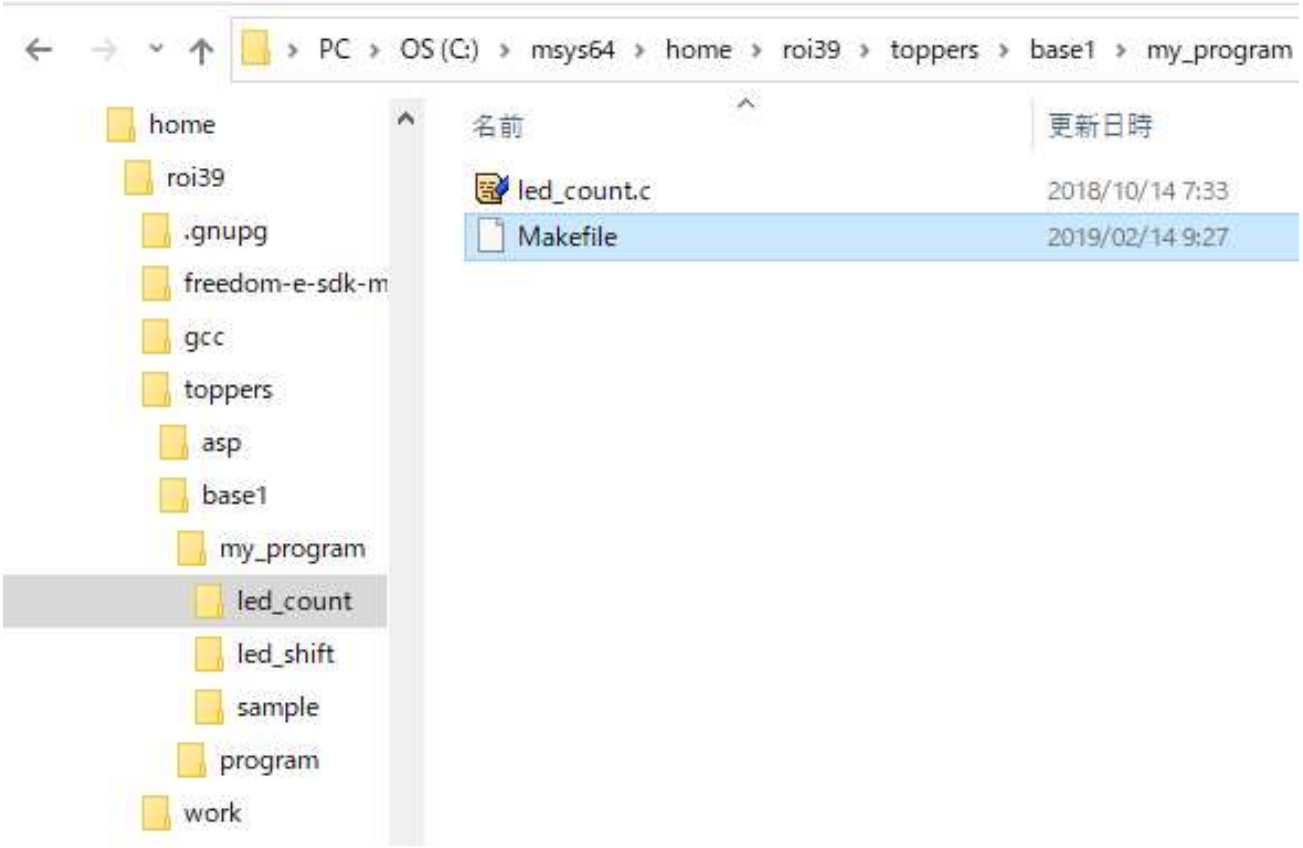
- プロジェクトのためのディレクトリ（my_program/led_count）を作成
  - 作成場所に制約はない
- 教材ディレクトリ /programの直下 にある以下のファイルをmy_program フォルダにコピーする

cmsis_f4.h
sil.h
startup_stm32f4xx.S
stm32f4_ram.ld
stm32f4_rom.ld
stm32f4xx.c
stm32f4xx.h　sys_defs.h



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
- program/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



TOPPERS

Makefileの書き換え

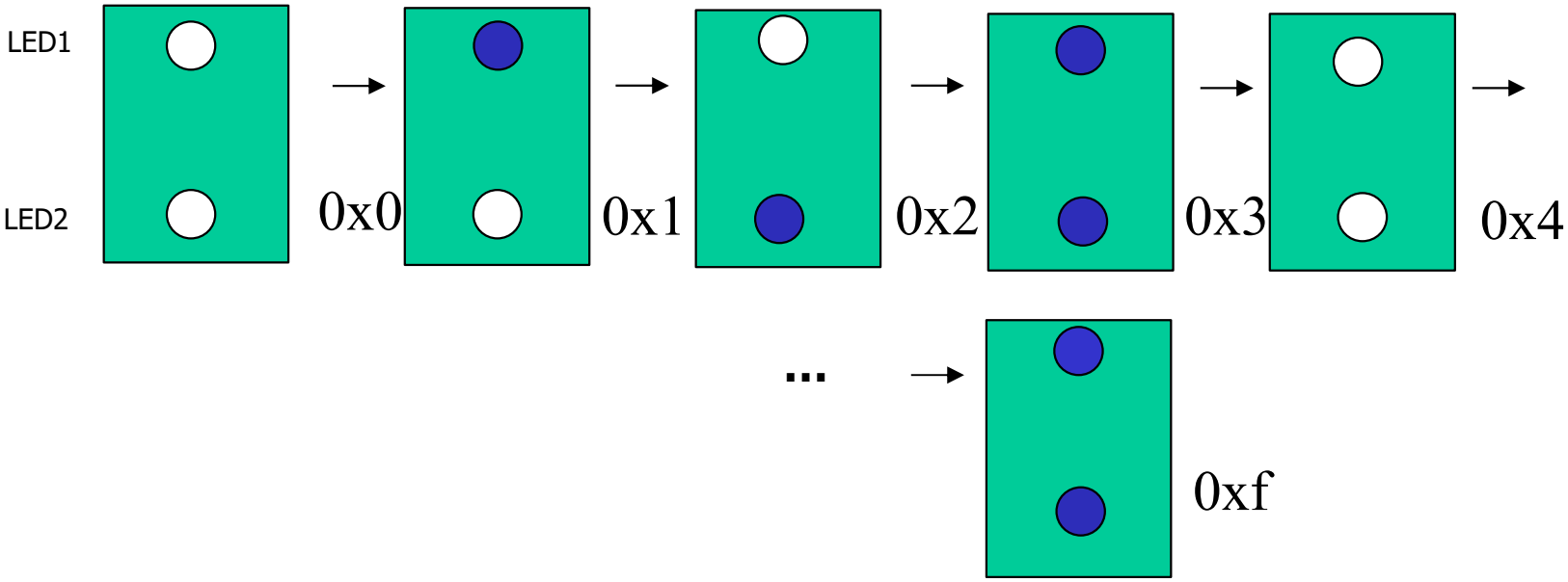
- コピーしたMakefileをエデッタで開く
- １２行目のOBJNAMEをled_shiftからled_countに変更する


```
#
#  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)
```


TOPPERS

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0x0fまで表示する（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

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2bit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

TOPPERS

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2bit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6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

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にしておく

```
void
led_out_bdigit(unsigned char data){
    unsigned short 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

TOPPERS

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）

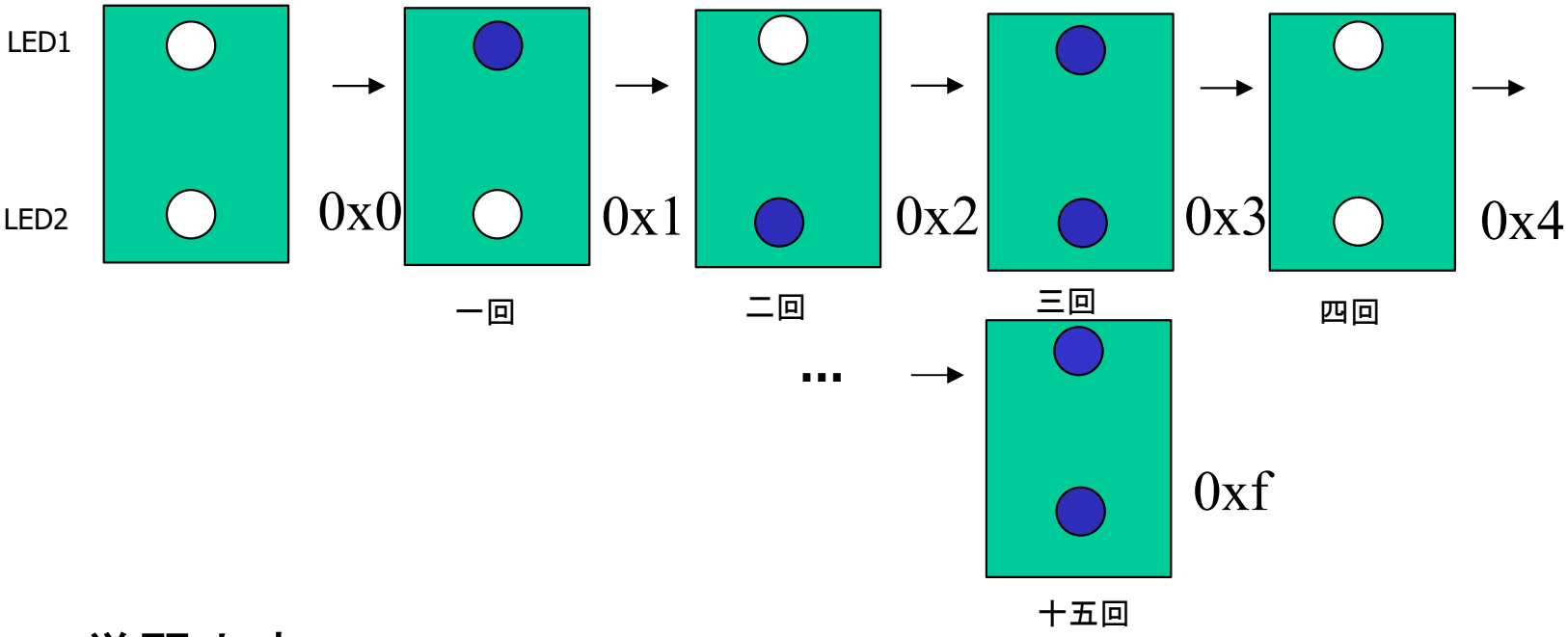
```
void
led_out_bdigit(unsigned char data){
    ...省略...
    led_out(led_data);
    sys_printf("led_out_bdight data[%02x]¥n", led_data);
}
```

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

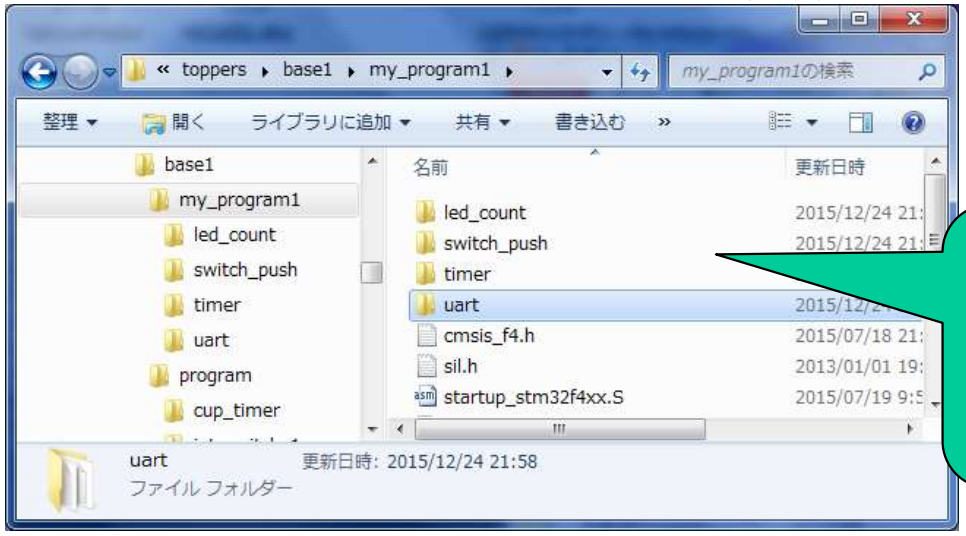
- SW1の押下回数を、LEDにバイナリ表示する



- 学習内容
  - ポートからの読み込み
- 作成方法
  - led_countをmy_programにコピーし、ディレクトリ名を書き換えて作成（次のスライドで説明）

led_countデレクトリのコピー

- 前の演習で作成したmy_program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)
- lcd_countをmy_program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

TOPPERS

switch_push : 作成

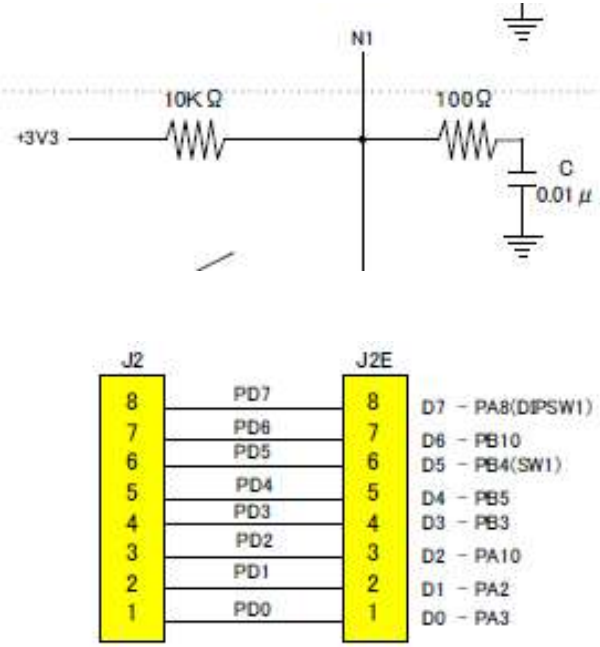
プログラム switch_push を作成

- 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
    - ポートの読込み
  - メイン関数
    - スイッチの変化を捉える(OFFからONへ)

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GPIOB:0/INT4ポートに接続されている
- ONで電流が流れ、GPIOB:0にSW1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

ボード



arduinoコネクタ

CN9 digital	8	D7	PA8	-
	7	D6	PB10	TIM2_CH3
	6	D5	PB4	TIM3_CH1
	5	D4	PB5	-
	4	D3	PB3	TIM2_CH2
	3	D2	PA10	-
	2	D1	PA2	USART2_TX
	1	D0	PA3	USART2_RX

SW1はPB4ポートに接続している

switch_push : プッシュスイッチの接続ポート

- ポートGPIOBに接続
  - SW1: ビット4 : PB4
- ポートGPIOBの構成レジスタ
  - モードレジスタ : 0x40020400
  - プルアップダウンレジスタ : 0x4002040C
  - ポート読み込みレジスタ : 0x40020410
- 初期化
  - ポート方向レジスタのビット0を入力モードへ（初期値）
- スイッチ状態の取得
  - ポートB4レジスタを読み込む

Table 39. GPIO register map and reset values (continued)

Offset	Register	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0x0C	GPIOx_PUPDR (where x = C..I/J/K)	PUPDR15[1:0]		PUPDR14[1:0]		PUPDR13[1:0]		PUPDR12[1:0]		PUPDR11[1:0]		PUPDR10[1:0]		PUPDR9[1:0]		PUPDR8[1:0]		PUPDR7[1:0]		PUPDR6[1:0]		PUPDR5[1:0]		PUPDR4[1:0]		PUPDR3[1:0]		PUPDR2[1:0]		PUPDR1[1:0]		PUPDR0[1:0]	
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x10	GPIOx_IDR (where x = A..I/J/K)	Reserved																IDR15	IDR14	IDR13	IDR12	IDR11	IDR10	IDR9	IDR8	IDR7	IDR6	IDR5	IDR4	IDR3	IDR2	IDR1	IDR0
	Reset value																	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x

TOPPERS

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）

- GPIOB関連のレジスタのマクロ定義（定義は自動でインクルード）

```
#define TADR_RCC_APB2ENR   0x40023844
#define TADR_GPIOB_MODER  0x40020400
#define TADR_GPIOB_PUPDR    0x4002040C
#define TADR_GPIOB_IDR        0x40020410
#define TREG_RCC_APB2ENR  ((volatile unsigned long *)(TADR_RCC_APB2ENR))
#define TREG_GPIOB_MODER ((volatile unsigned long *)(TADR_GPIOB_MODER))
#define TREG_GPIOB_PUPDR   ((volatile unsigned long *)(TADR_GPIOB_PUPDR))
#define TREG_GPIOB_IDR       ((volatile unsigned long *)(TADR_GPIOB_IDR))
```

- GPIOBポートのプッシュスイッチの接続ビット定義

```
#define PINPOSITION4  4
#define PSW1              0x0010
#define PSW_MASK    (PSW1)
#define PSW1_PINNO   PINPOSITION4
```

switch_push : 初期化関数

- GPIOBポートのビット0を入力モードへ
  - GPIOBポート方向レジスタのビット4を’00’に設定する

```
void
switch_push_init(void){
    unsigned long reg;
    unsigned int pinpos = PSW1_PINNO;
    *TREG_RCC_AHB1ENR |= RCC_AHB1ENR_GPIOBEN;
    *TREG_RCC_APB2ENR |= RCC_APB2ENR_SYSCFGEN;
    reg = *TREG_GPIOB_MODER & ~(3 << (pinpos*2));
    *TREG_GPIOB_MODER = reg | (GPIO_Mode_IN << (pinpos *2));
    reg = *TREG_GPIOB_OSPEEDR & ~(3 << (pinpos * 2));
    *TREG_GPIOB_OSPEEDR = reg | (GPIO_Speed_2MHz << (pinpos * 2));
    reg = *TREG_GPIOB_PUPDR & ~(3 << (pinpos*2));
    *TREG_GPIOB_PUPDR = reg | (GPIO_PuPd_NOPULL << (pinpos *2));
}
```

ビット0のみ’00’に設定

TOPPERS

switch_push : プッシュスイッチ状態取得関数

- switch_push_sense()
- GPIOBのビット4の値を取得

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

    reg = *TREG_GPIOB_IDR ^ PSW1;
    return reg  & PSW_MASK;
}
```

ビット4のみ有効に

TOPPERS

switch_push : main()関数

- カウント用変数led_data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SW1がOFFからONに変化した場合は，led_dataをインクリメントする
- 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
- スイッチの状態（ON,OFF）はマクロ化する

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

switch_push : main()関数

```
void  _main(void){
    unsigned char led_data = 0;
    unsigned long sw_data;
    unsigned char sw1_state = SW_OFF;
    led_init();
    switch_push_init();
    for (;;) {
        sw_data = switch_push_sense();
        if(((sw_data & PSW1) == PSW1) && (sw1_state == SW_OFF)){
            led_data++;
        }
       sw1_state = ((sw_data & PSW1) == PSW1)? SW_ON : SW_OFF;
       led_out_bdigit(led_data);
   }
}
```

SW1状態保存用変数

プッシュスイッチ状態取得

SW1がOFFからONになったか

SW1の状態を保存

LED設定

TOPPERS

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

タイマプログラム：timer 概要

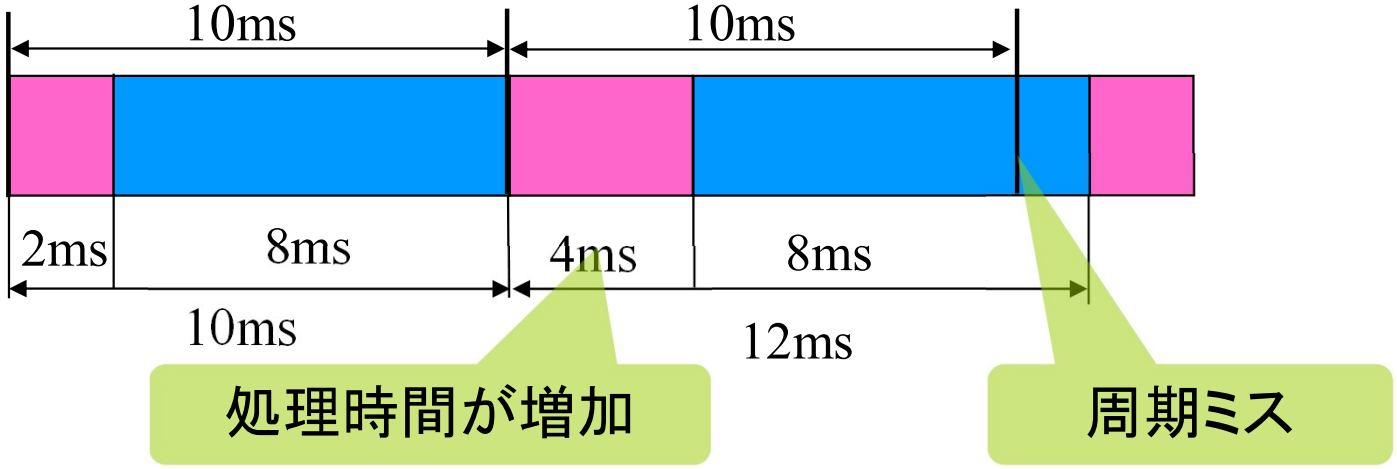
- ハードウェアタイマにより正確に1秒間隔でLEDの点灯をカウントアップする

- 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 をコピーし、ディレクトリ名をtimerに書き換えて作成

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



timer : ハードウェアタイマの概要

- stm32F401のハードウェアタイマ
  - ３２ビットタイマを7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4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84MHzをタイマクロックとして使用
  - 周辺クロック選択レジスタで変更可能

timer : レジスタ

- タイムアウトはステータスレジスタのUIF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
TIM4_CR1	0x40000800	コントロール・レジスタ1
TIM4_SR	0x40000810	ステータス・レジスタ
TIM4_EGR	0x40000814	タイマ0カウンター
TIM4_PSC	0x40000828	プリスケールレジスタ
TIM4_ARR	0x4000082C	オートリロード・レジスタ
RCC_APB1 ENR	0x40023840	APB1クロック・イネーブル・レジスタ
RCC_AHB1 ENR	0x40023830	AHB1クロック・イネーブル・レジスタ

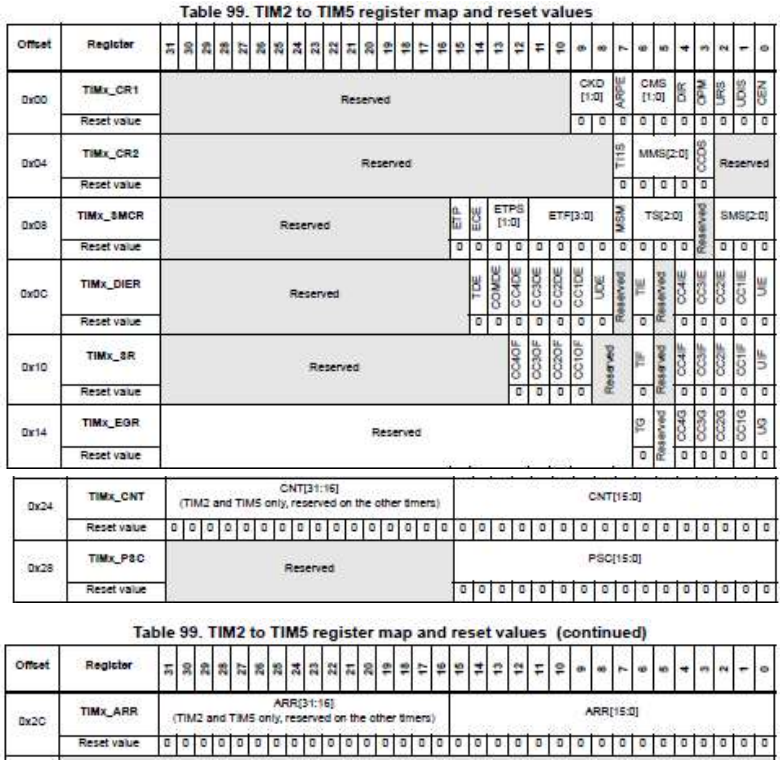


Table 99. TIM2 to TIM5 register map and reset values

Offset	Register	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0x00	TIMx_CR1	Reserved																						CKD [1:0]		ARPE	CMS [1:0]		DIR	OPM	URS	UDIS	CEN
	Reset value																							0	0	0	0	0	0	0	0	0	0
0x04	TIMx_CR2	Reserved																								TI1S	MMS[2:0]			CCDS	Reserved		
	Reset value																									0	0	0	0	0			
0x08	TIMx_SMCR	Reserved																ETP	ECE	ETPS [1:0]			ETF[3:0]			MSM	TS[2:0]			Reserved	SMS[2:0]		
	Reset value																	0	0	0	0	0	0	0	0	0	0	0	0		0	0	0
0x0C	TIMx_DIER	Reserved																TDE	COMDE	CC4DE	CC3DE	CC2DE	CC1DE	UDE	Reserved	TIE	Reserved	CC4IE	CC3IE	CC2IE	CC1IE	UIE	
	Reset value																	0	0	0	0	0	0	0		0		0	0	0	0	0	
0x10	TIMx_SR	Reserved																		CC4OF	CC3OF	CC2OF	CC1OF	Reserved		TIF	Reserved	CC4IF	CC3IF	CC2IF	CC1IF	UIF	
	Reset value																			0	0	0	0			0		0	0	0	0	0	
0x14	TIMx_EGR	Reserved																								TG	Reserved	CC4G	CC3G	CC2G	CC1G	UG	
	Reset value																									0		0	0	0	0	0	
0x24	TIMx_CNT	CNT[31:16] (TIM2 and TIM5 only, reserved on the other timers)																CNT[15:0]															
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x28	TIMx_PSC	Reserved																PSC[15:0]															
	Reset value																	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 99. TIM2 to TIM5 register map and reset values (continued)

Offset	Register	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0x2C	TIMx_ARR	ARR[31:16] (TIM2 and TIM5 only, reserved on the other timers)																ARR[15:0]															
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

TOPPERS

timer : タイマ操作の流れ

- タイマ4モードを設定
  - クロックを有効化
  - 停止とリセット、分周比
- リロードカウンタ、プリスケール値を設定
  - カウントソースと計時したい時間から求める
- リロードモード設定

- ステータスレジスタをクリアし、タイマ4を起動し、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1/1分周を使用
  - カウント値の決定
    - リロードカウンタを4000として1秒のタイムアウトを作成
  - 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

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
  - クロックがリロードカウンタ分、発生するとカウントアップする
  - カウンタ値がプリスケールにマッチするとタイムアウト
  - カウントソース：システムクロック/1
    - 1/1(00),1/2(01),1/4(10)から設定可能
    - STM32F401はシステムクロックが84MHzなので，カウントソースは84MHz
      - 1/1: 84MHz/1 ＝84MHz
      - 1/2: 84MHz/2 ＝42MHz
      - 1/4: 84MHz/4 ＝ 10.5MHz
    - STM32F446は180MHzなので、180MHz
  - カウンタ値が16ビットに収まるようリロードカウンタを4000に設定
    - 84MHzで1秒を計測するには84000000/65536≒1281
    - 180MHzで1秒を計測するには90000000/65536≒2747
    - リロードカウンタで4000にすれば16ビットに収まる

timer : カウント値の決定

- 分周を1/1、リロードカウンタ4000（1秒）
  - プリスケール設定値：84,000,000/4000 – 1 = 20999
  - (STM32F446では180,000,000/4000-1=44999)
  - リロードカウンタ設定値：4000
- その他のレジスタの設定値
  - TM4_CR1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TIM4_SRのビット0(0x01)が‘1’にセットされる
  - TIM4_SRのビット0から‘0’を読み込んだ後は‘0’にクリアする
    - (0x0000を書き込んでクリア)

名前	アドレス	設定値	説明
TM4_ARR	0x4000082C	0x0F9F	リロードカウンタ設定
TM4_PSC	0x40000828	0x5207	プリスケール設定
TM4_CR1	0x40000800	0x0089	タイマスタート(0x0088で停止)

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: 自動でインクルード)

```
/* TIMER用 RCCレジスタ */
#define TREG_RCC_APB1ENR ((volatile unsigned long *)0x40023840)
/* タイマ4コントロール１レジスタ */
#define TREG_TIM4_CR1    ((volatile unsigned short *)0x40000800)
/* タイマ4ステータスレジスタ */
#define TREG_TIM4_SR     ((volatile unsigned short *)0x40000810)
/* タイマ4イベントジェネレーションレジスタ */
#define TREG_TIM4_EGR   ((volatile unsigned short *)0x40000814)
/* タイマ4プリスケールレジスタ       */
#define TREG_TIM4_PSC   ((volatile unsigned short *)0x40000828)
/* タイマ4オートリロードレジスタ   */
#define TREG_TIM4_ARR  ((volatile unsigned short *)0x4000082C)
```

- タイマ関連の定義

```
#define TIMER_COUNT        4000          /* プリスケール値 */
#define TIM_CR1_CEN        0x0001        /* タイマスタート */
#define TIM_CR1_OPM        0x0008        /*  ワンプラスモード */
#define TIM_CR1_DIR        0x0010        /* ディレクション */
#define TIM_CR1_CMS        0x0060        /* CMS */
#define TIM_CR1_ARPE       0x0080        /* オートリロードイネーブル */
#define TIM_CR1_CKD        0x0300        /* CKD */
/* ステータスレジスタ設定 */
#define TIM_SR_UIF         0x0001        /* アップデート要求 */
```


TOPPERS

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
  - STM32F401はマニュアルに特に記載がない

```
void
timer4_init(void){
    unsigned short PrescalerValue = 0;
    unsigned short tmpcr1 = *TREG_TIM4_CR1;
    /* TIM4クロックを有効 */
    *TREG_RCC_APB1ENR |= RCC_APB1ENR_TIM4EN;
    /* カウンタモード、クロック分周設定 */
    *TREG_TIM4_CR1 = tmpcr1 & ~(TIM_CR1_DIR | TIM_CR1_CKD);
    PrescalerValue = ((SYSTEMCORECLOCK)/TIMER_COUNT)-1;
    /* プレスケール値の設定 */
    *TREG_TIM4_ARR= TIMER_COUNT;
    *TREG_TIM4_PSC= PrescalerValue;
    *TREG_TIM4_EGR=   0x0001;
}
```

クロックイネーブル

モード設定

プリスケール値設定

TOPPERS

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
timer4_wait(int cnt){
    while(cnt > 0){
      /* タイマ停止 */
      *TREG_TIM4_CR1 &= ~TIM_CR1_CEN;
      /* モード再設定 */
      *TREG_TIM4_CR1 |= TIM_CR1_ARPE | TIM_CR1_OPM;
      /* タイマーステータスリセット */
     *TREG_TIM4_SR = 0x0000;
      /* カウントスタート */
      *TREG_TIM4_CR1 |= TIM_CR1_CEN;
      while((*TREG_TIM4_SR & TIM_SR_UIF) == 0){
      }
      cnt--;
   }
   *TREG_TIM4_CR1 &=  ~TIM_CR1_CEN;
}
```

タイマ停止

モード設定

割込みフラグクリア

タイマスタート

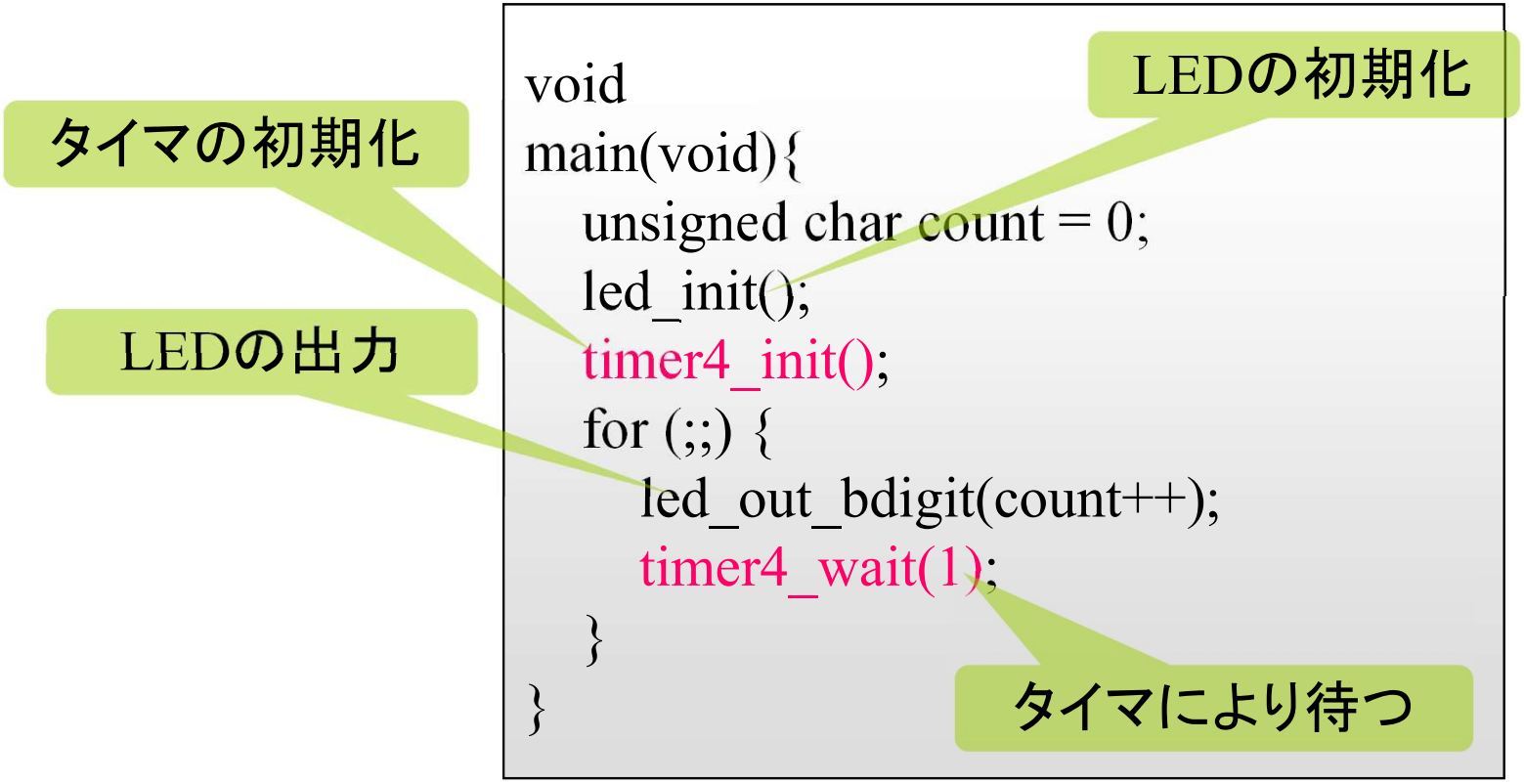
タイムアウト待ち

タイマ停止

TOPPERS

timer : メイン関数

- led_countのメイン関数を変更
  - busy_wait() を timer4_wait()に変更




```
void
main(void){
    unsigned char count = 0;
    led_init();
    timer4_init();
    for (;;) {
        led_out_bdigit(count++);
        timer4_wait(1);
    }
}
```

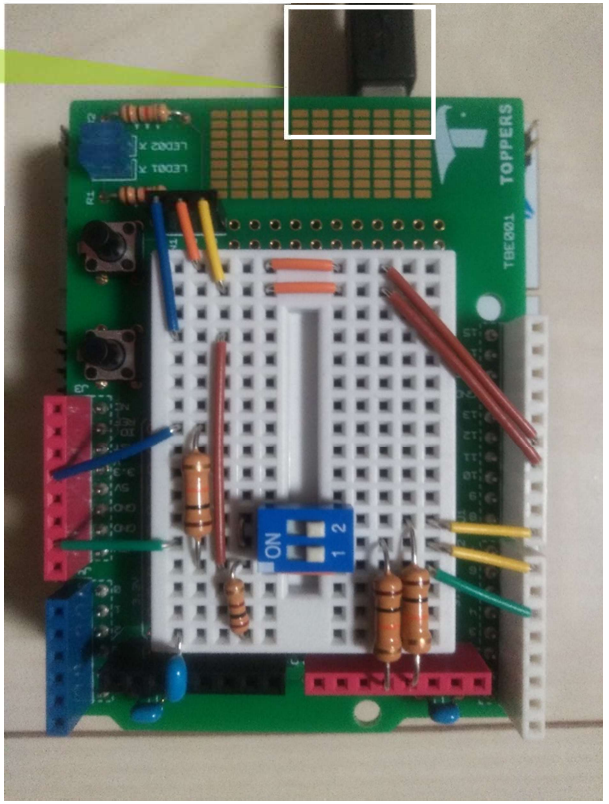
ポーリングプログラム

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

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）

USART2 TX/RX
仮想COMでUSB
に対応

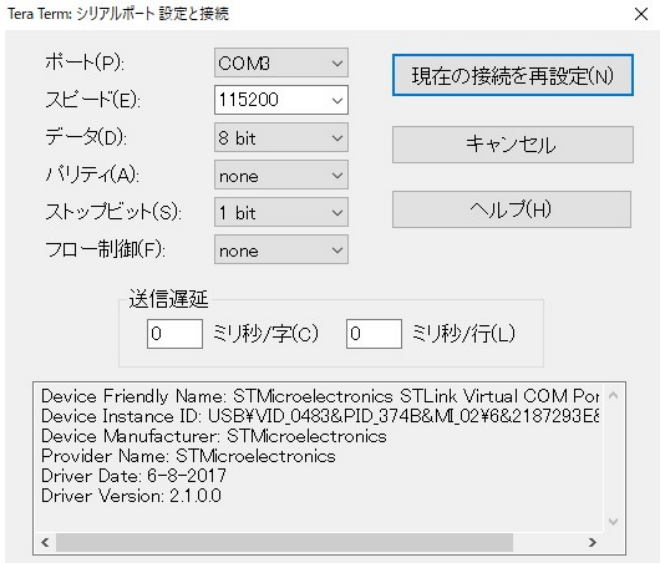


- 学習内容
  - シリアルI/Oの使い方
- 作成方法
  - led_countをコピーし、
    ディレクトリ名をuartに書き換えて
    作成
- ボードの設定
  - USART2のTX/RXをST-LINK仮想COMに接続する

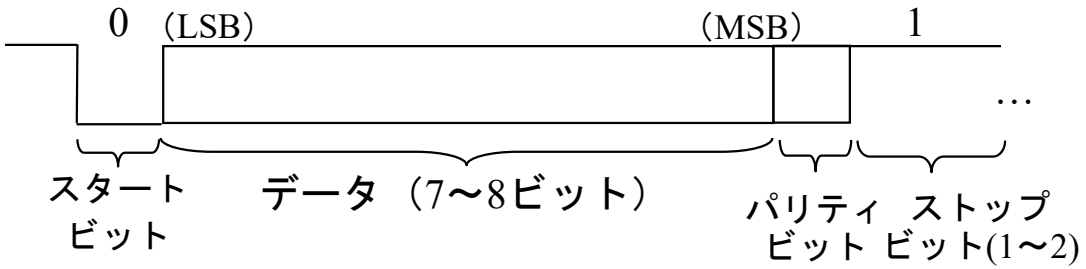
uart : シリアルI/Oのフォーマット

- 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



TOPPERS

uart:ピン・クロック設定

- UARTとGPIOA(TX/RX)のクロック設定
  - AHB1 : GPIOAクロックイネーブル
  - APB1 : USART2クロックイネーブル
- TX/RXピンのGPIOピン設定
  - TX/RXピン対応のGPIOAのピン設定

名前	アドレス	説明
TREG_RCC_AHB1ENR	0x40023830	AHB1クロック・イネーブル
TREG_RCC_APB1ENR	0x40023840	APB1クロック・イネーブル
TREG_GPIOA_MODER	0x40020000	GPIOAのモードレジスタ
TREG_GPIOA_OTYPER	0x40020004	GPIOAのアウトプットタイプレジスタ
TREG_GPIOA_OSPEEDR	0x40020008	GPIOAのスピードレジスタ
TREG_GPIOA_PUPDR	0x4002000C	GPIOAのプルアップダウンレジスタ
TREG_GPIOA_AFR0	0x40020020	GPIOAアルタネートファンクションレジスタ

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）：115200，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
  - UART2を使用
  - UARTモードで使用，カウントソースはPCLK1（82Mhzの1/2分周）

名前	アドレス	説明
TREG_USART2_SR	0x40004400	ステータスレジスタ
TREG_USART2_DR	0x40004404	データレジスタ
TREG_USART2_BRR	0x40004408	ボーレートレジスタ
TREG_USART2_CR1	0x4000440C	コントロールレジスタ1
TREG_USART2_CR2	0x40004410	コントロールレジスタ2
TREG_USART2_CR3	0x40004414	コントローラレジスタ3

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115200bps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

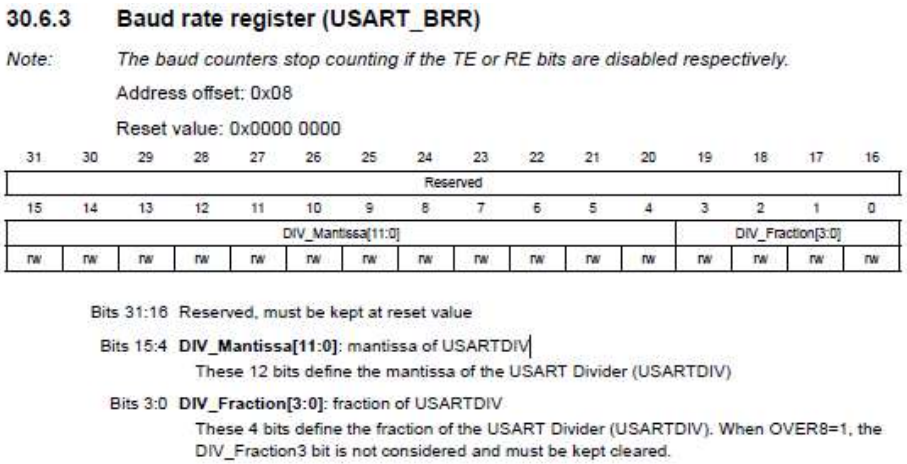
uart:クロック設定、TX/RX設定

クロック設定とTX/RXのピン設定を行う

- APB1ペリフェラルのUSART2クロックを有効に
- USART2のTX/RXはGPIOAの2ピン、3ピンに対応
  - GPIOAのクロックを有効に
- TX/RX対応のGPIOAの2ピン、3ピンを以下に設定
  - MODERを10:アルタネートファンクションモード
  - OSPEEDRを10:Fast Speed
  - OTYPERを0:Output push-pull
  - PUPDRを01:Pull-UP
  - AFRを0111:AF7

uart : 転送速度レジスタの設定値の計算

- 転送速度は 115200bps
- カウントソースはf(42Mhz:84Mhz/2)を使用
- BRRにUSARTDIV(小数点2ケタ)を設定
  - USARTDIV = f / (0.16 × bps)
- USARTDIV = ((25 * 42MHz) / (4*115200))
- DIV_Masntissa = USARTDIV / 100
- DIV_Fraction = USARTDIV – (DIV_Masntissa * 100)
- DIV_Fraction = (((DIV_Fraction * 16) + 50) / 100) & 0xf



30.6.3 Baud rate register (USART_BRR)

Note: The baud counters stop counting if the TE or RE bits are disabled respectively.

Address offset: 0x08

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DIV_Mantissa[11:0]												DIV_Fraction[3:0]			
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:16 Reserved, must be kept at reset value

Bits 15:4 **DIV_Mantissa[11:0]**: mantissa of USARTDIV
These 12 bits define the mantissa of the USART Divider (USARTDIV)

Bits 3:0 **DIV_Fraction[3:0]**: fraction of USARTDIV
These 4 bits define the fraction of the USART Divider (USARTDIV). When OVER8=1, the DIV_Fraction3 bit is not considered and must be kept cleared.

TOPPERS

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

- コントロールレジスタ2 : 1ストップビット
  - STOP[13:12]

- コントロールレジスタ1 : 以下の設定
  - UARTモード転送データ長8ビット:M[12]
  - TX/RXモード:TE|RE[4:3]
  - パリティ禁止:PCE|PS[10:9]

- コントロールレジスタ3 : ハードウェアフロー設定
  - CTSE:RTSE[9:8]

uart : マクロ定義1 : UART2関連のレジスタ

```
#define TADR_RCC_AHB1ENR    0x40023830    /* RCC AHB1ENRレジスタ */
#define TADR_RCC_APB2ENR    0x40023844    /* RCC APB2ENレジスタ */
#define TADR_GPIOA_MODER    0x40020000    /* AポートMODEレジスタ */
    :
/* RCC レジスタ */
#define TREG_RCC_AHB1ENR  ((volatile unsigned long *) (TADR_RCC_AHB1ENR))
#define TREG_RCC_APB1ENR   ((volatile unsigned long *) (TADR_RCC_APB2ENR))
 /* GPIOAバッファレジスタ */
#define TREG_GPIOA_MODER   ((volatile unsigned long *) (TADR_GPIOA_MODER))
#define TREG_RCC_AHB1ENR  ((volatile unsigned long *) (TADR_RCC_AHB1ENR))
#define TREG_GPIOA_OSPEEDR   ((volatile unsigned long *) (TADR_GPIOA_OSPEEDR))
#define TREG_GPIOA_OTYPER  ((volatile unsigned long *) (TADR_GPIOA_OTYPER))
#define TREG_GPIOA_PUPDR   ((volatile unsigned long *) (TADR_GPIOA_PUPDR))
#define TREG_GPIOA_AFR0  ((volatile unsigned long *) (TADR_GPIOA_AFR0))
/* USART2レジスタ  */
#define TREG_USART2_SR    ((volatile unsigned short *) (TADR_USART2_SR))
#define TREG_USART2_DR   ((volatile unsigned short *) (TADR_USART2_DR))
#define TREG_USART2_BRR ((volatile unsigned short *) (TADR_USART2_BRR))
#define TREG_USART2_CR1  ((volatile unsigned short *) (TADR_USART2_CR1))
#define TREG_USART2_CR2  ((volatile unsigned short *) (TADR_USART2_CR2))
#define TREG_USART2_CR3  ((volatile unsigned short *) (TADR_USART2_CR3))
```


TOPPERS

uart : マクロ定義2 : レジスタの設定値

- ビット定義はstm32f4xx.hに定義

```
#define TX2_PINPOS              2          /* TX-PIN POSITION */
#define RX2_PINPOS              3          /* RX-PIN POSITION */
#define CR1_CLEAR_MASK (USART_CR1_M | USART_CR1_PCE
              | USART_CR1_PS | USART_CR1_TE | USART_CR1_RE)
#define CR3_CLEAR_MASK (USART_CR3_RTSE | USART_CR3_CTSE)
#define GPIO_AF_USART2     0x07     /* USART2 Alternate function */
```

- USART初期化定義

```
#define USART_StopBits_1  0x00 /* ストップビット１  */
#define USART_WordLength_8b 0x00 /* Data 8bit */
#define USART_Parity_No   0x00 /* No Parity */
#define USART_Mode_Rx    USART_CR1_RE
#define USART_Mode_Tx    USART_CR1_TE
#define USART_HardwareFlowControl_None 0x00
```

uart : 初期化設定関数

- RCC設定とTXピン設定

```
void
uart2_init(void){
    unsigned int tmpreg, apbclock, integerdivider, fractionaldivider;
    /* RCC設定 */
    *TREG_RCC_APB1ENR |= RCC_APB1ENR_USART2EN;
    *TREG_RCC_AHB1ENR |=  RCC_AHB1ENR_GPIOAEN;
     /* TX-PIN設定 */
    *TREG_GPIOA_MODER &= ~(GPIO_MODER_MODER2 <<(TX2_PINPOS*2));
    *TREG_GPIOA_MODER |= GPIO_Mode_AF<< (TX2_PINPOS*2);
    *TREG_GPIOA_OSPEEDR &= ~(GPIO_OSPEEDER_OSPEEDR2<<(TX2_PINPOS*2));
    *TREG_GPIOA_OSPEEDR |= GPIO_Speed_50MHz<<(TX2_PINPOS*2);
    *TREG_GPIOA_OTYPER &= ~(GPIO_OTYPER_OT << TX2_PINPOS);
    *TREG_GPIOA_OTYPER |= GPIO_OType_PP << TX2_PINPOS;
    *TREG_GPIOA_PUPDR &= ~(GPIO_PUPDR_PUPDR2 << (TX2_PINPOS*2));
    *TREG_GPIOA_PUPDR |= GPIO_PuPd_UP<< (TX2_PINPOS*2);
```


TOPPERS

uart : 初期化設定関数

- RXピン設定とアルティネイトファンクション設定

```
 /* RX-PIN設定 */
*TREG_GPIOA_MODER &= ~(GPIO_MODER_MODER2 <<(RX2_PINPOS*2));
*TREG_GPIOA_MODER |= GPIO_Mode_AF<< (RX2_PINPOS*2);
*TREG_GPIOA_OSPEEDR &= ~(GPIO_OSPEEDER_OSPEEDR2<<(RX2_PINPOS*2));
*TREG_GPIOA_OSPEEDR |= GPIO_Speed_50MHz<<(RX2_PINPOS*2);
*TREG_GPIOA_OTYPER &= ~(GPIO_OTYPER_OT << RX2_PINPOS);
*TREG_GPIOA_OTYPER |= GPIO_OType_PP << RX2_PINPOS;
*TREG_GPIOA_PUPDR &= ~(GPIO_PUPDR_PUPDR2 << (RX2_PINPOS*2));
*TREG_GPIOA_PUPDR |= GPIO_PuPd_UP<< (RX2_PINPOS*2);
 /* UART AF設定 */
*TREG_GPIOA_AFR0 &= ~(0xF << ((TX2_PINPOS & 0x07) * 4));
*TREG_GPIOA_AFR0 |= GPIO_AF_USART2 << ((TX2_PINPOS & 0x07) * 4);
*TREG_GPIOA_AFR0 &= ~(0xF << ((RX2_PINPOS & 0x07) * 4));
 *TREG_GPIOA_AFR0 |= GPIO_AF_USART2 << ((RX2_PINPOS & 0x07) * 4);
```


TOPPERS

uart : 初期化関数 uart2_init

- 各制御レジスタを初期化し，送受信を許可する

```
    *TREG_USART2_CR2 &= ~USART_CR2_STOP;
    *TREG_USART2_CR2 |= USART_StopBits_1;
    *TREG_USART2_CR1 &= ~CR1_CLEAR_MASK;
    *TREG_USART2_CR1 |= USART_WordLength_8b | USART_Parity_No
                      | USART_Mode_Rx | USART_Mode_Tx;
    *TREG_USART2_CR3 &=  ~CR3_CLEAR_MASK;
    *TREG_USART2_CR3 |= USART_HardwareFlowControl_None;
    /* BRR設定 */
    apbclock = PCLK1;
    integerdivider = (25 * apbclock) / (4 * DEFAULT_SPEED);
    tmpreg = (integerdivider / 100) << 4;
    fractionaldivider = integerdivider - (100 * (tmpreg >> 4));
    tmpreg |= (((fractionaldivider * 16) + 50) / 100) & 0xf;
    *TREG_USART2_BRR = tmpreg;
    *TREG_USART2_CR1 |= USART_CR1_RXNEIE;
    *TREG_USART2_CR1 |= USART_CR1_UE;
}
```

DEFAILT_SPEEDと
PCLK1は
sys_defs.hで定義

TOPPERS

uart : 受信関数

- データを受信すると，受信制御レジスタのRXNE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
uart2_pol_getc(void){
    /* 受信バッファにデータが入るのを待つ */
   while((*TREG_USART2_SR & USART_SR_RXNE) == 0);

   /* データ受信 */
   return (unsigned char)(*TREG_USART2_DR);
}
```

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TC（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
uart2_pol_putc(unsigned char c){
    /* 送信バッファが空くのを待つ */
    while((*TREG_USART2_SR & USART_SR_TC) == 0);

    /* データ送信 */
    *TREG_USART2_DR = c;
}
```

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();
    uart2_init();

    for (;;) {
        c = uart2_pol_getc();
        led_out_bdigit(++count);
        uart2_pol_putc(c);
    }
}
```

データ受信

LEDカウントアップ

データ送信

TOPPERS

割込みプログラミング

1. Cortex-Mの割込みアーキテクチャ
2. スイッチ割込みプログラム
3. 割込みタイマ

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

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/base1/program

- 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

Cortex-Mの割込みアーキテクチャ

Cortex-Mはネスト型割込みコントローラ(NVIC)を標準搭載

- NVICは最大240の割込みをサポート
- 各割込みには最大256レベルの優先度を動的に設定可能
- NVICへの完全なアクセスは特権モードから可能
- レベル割込みとパルス割込み
  - レベル割込みは、デバイスにアクセスするISRによってクリアさせるまで、アサートされ続ける
  - パルス割込みが、エッジモデルの一種で、パルスのサンプリングは非同期ではなく、Cortex-MのクロックFCLKの立ち上がりでエッジされることを保証する必要がある

NVICレジスタ

NVICレジスタ設定に割込みの有効無効、割込み優先度を設定する

- NVIC_ISER割込みの有効化
- NVIC_ICER割込みの無効化
- NVIC_IPR割込み優先度設定

アドレス	名前	タイプ	リセット時の値	説明
0xE000E004	ICTR	RO	-	*「割り込みコントローラタイプレジスタ、ICTR」*
0xE000E100～0xE000E11C	NVIC_ISER0～NVIC_ISER7	RW	0x00000000	割り込みイネーブルセット レジスタ
0xE000E180～0xE000E19C	NVIC_ICER0～NVIC_ICER7	RW	0x00000000	割り込みイネーブルクリア レジスタ
0xE000E200～0xE000E21C	NVIC_ISPR0～NVIC_ISPR7	RW	0x00000000	割り込み保留セットレジスタ
0xE000E280～0xE000E29C	NVIC_ICPR0～NVIC_ICPR7	RW	0x00000000	割り込み保留クリアレジスタ
0xE000E300～0xE000E31C	NVIC_IABR0～NVIC_IABR7	RO	0x00000000	割り込みアクティブビット レジスタ
0xE000E400～0xE000E4EC	NVIC_IPR0～NVIC_IPR59	RW	0x00000000	割り込み優先度レジスタ

TOPPERS

Cortex-Mの割込みアーキテクチャ

- IRQ割込みはstartup_stm32f4xx.Sにベクタテーブルがすでに定義されている

```
.section  .isr_vector,"a",%progbits
 .type  g_pfnVectors, %object
 .size  g_pfnVectors, .-g_pfnVectors
g_pfnVectors:
.word  _estack
.word  Reset_Handler
.word  NMI_Handler
.word  HardFault_Handler
 :
  /* External Interrupts */
.word     WWDG_IRQHandler            /* Window WatchDog              */
.word     PVD_IRQHandler             /* PVD through EXTI Line detection */
.word     TAMP_STAMP_IRQHandler      /* Tamper and TimeStamps through the EXTI line */
.word     RTC_WKUP_IRQHandler        /* RTC Wakeup through the EXTI line */
.word     FLASH_IRQHandler           /* FLASH                        */
 :
.word     EXTI4_IRQHandler           /* EXTI Line4                   */
```


TOPPERS

Cortex-Mの割込みアーキテクチャ

- EXTI4用の割込みハンドラ（EXTI4_IRQHandler）はstartup_stm32f4xx.S内で.weak設定で定義されており、Default_Handlerにリンクしている
- この定義は仮定義で、globalにEXTI4_Handler関数をリンクすれば、その関数が正しい関数として扱われる

```
.weak      RTC_WKUP_IRQHandler
.thumb_set RTC_WKUP_IRQHandler,Default_Handler

.weak      FLASH_IRQHandler
.thumb_set FLASH_IRQHandler,Default_Handler

.weak      RCC_IRQHandler
.thumb_set RCC_IRQHandler,Default_Handler

.weak      EXTI4_IRQHandler
.thumb_set EXTI4_IRQHandler,Default_Handler
```

割込みプログラミング

1. Cortex-Mの割込みアーキテクチャ
2. スイッチ割込みプログラム
3. 割込みタイマ

スイッチ割込みプログラム1：int_switch_1 概要

- SW1を押すと発生する割込み（EXTI4）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
  - switch_pushをコピーして、ディレクトリ名をint_switch_1に修正して作成する

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

int_switch_1 : 割込み要因ハードウェア

プッシュスイッチ（SW1）を割込み要因として用いる

- 接続
  - GPIOB入出力ポートのビット4（PB4）に接続
  - PB4は外部割込み4（EXTI4）とピンを共有
  - ボタンを押すとゼロ(LOW)が入力される
- 割込み番号、優先度（NVIC_IPRを参照）
  - 割込み番号EXTI4：10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Capter12.2.1参照
  - PINをEXTI4に設定、モードを立上がりエッジに設定する
    - キーを押し、離した時点で割込みが発生

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）

int_switch_1 : EXTI割込み許可

- EXTI_IMR：割込みマスクレジスタ
- EXTI_EMR：イベントマスクレジスタ
  - EXTI4（ビット4）のイベントクリア割込みの許可

12.3.1 Interrupt mask register (EXTI_IMR)

Address offset: 0x00

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved									MR22	MR21	MR20	MR19	MR18	MR17	MR16
									rw	rw	rw	rw	rw	rw	rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
MR15	MR14	MR13	MR12	MR11	MR10	MR9	MR8	MR7	MR6	MR5	MR4	MR3	MR2	MR1	MR0
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:23 Reserved, must be kept at reset value.

Bits 22:0 **MRx:** Interrupt mask on line x

0: Interrupt request from line x is masked

1: Interrupt request from line x is not masked

12.3.2 Event mask register (EXTI_EMR)

Address offset: 0x04

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved									MR22	MR21	MR20	MR19	MR18	MR17	MR16
									rw	rw	rw	rw	rw	rw	rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
MR15	MR14	MR13	MR12	MR11	MR10	MR9	MR8	MR7	MR6	MR5	MR4	MR3	MR2	MR1	MR0
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:23 Reserved, must be kept at reset value.

Bits 22:0 **MRx:** Event mask on line x

0: Event request from line x is masked

1: Event request from line x is not masked

TOPPERS

int_swicth_1：EXTI割込み設定

- EXTI_RTSR：ライジング・トリガ・選択レジスタ
- EXTI_FTSR：フォーリング・トリガ・選択レジスタ
  - EXTI4（ビット4）のライジング・トリガ設定

12.3.3 Rising trigger selection register (EXTI_RTSR)

Address offset: 0x08
Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved									TR22	TR21	TR20	TR19	TR18	TR17	TR16
									rw	rw	rw	rw	rw	rw	rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
TR15	TR14	TR13	TR12	TR11	TR10	TR9	TR8	TR7	TR6	TR5	TR4	TR3	TR2	TR1	TR0
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:23 Reserved, must be kept at reset value.

Bits 22:0 **TRx:** Rising trigger event configuration bit of line x
0: Rising trigger disabled (for Event and Interrupt) for input line
1: Rising trigger enabled (for Event and Interrupt) for input line

Note: The external wakeup lines are edge triggered, no glitch must be generated on these lines. If a rising edge occurs on the external interrupt line while writing to the EXTI_RTSR register, the pending bit is be set.

Rising and falling edge triggers can be set for the same interrupt line. In this configuration, both generate a trigger condition.

12.3.4 Falling trigger selection register (EXTI_FTSR)

Address offset: 0x0C
Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved									TR22	TR21	TR20	TR19	TR18	TR17	TR16
									rw	rw	rw	rw	rw	rw	rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
TR15	TR14	TR13	TR12	TR11	TR10	TR9	TR8	TR7	TR6	TR5	TR4	TR3	TR2	TR1	TR0
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:23 Reserved, must be kept at reset value.

Bits 22:0 **TRx:** Falling trigger event configuration bit of line x
0: Falling trigger disabled (for Event and Interrupt) for input line
1: Falling trigger enabled (for Event and Interrupt) for input line.

int_switch_1 : NVIC制御関連のレジスタの定義

- 割込み制御レジスタ（自動でインクルード）

```
#define TREG_SCB_AIRCR  ((volatile unsigned long *)0xE00ED0C)
#define TREG_NVIC_ISER  ((volatile unsigned long *)0xE000E100)
#define TREG_NVIC_ICER  ((volatile unsigned long *)0xE000E180)
#define TREG_NVIC_IP    ((volatile unsigned char *)0xE000E400)
```

- EXTIレジスタ（自動でインクルード）

```
#define TREG_SYSCFG_EXTICR ((volatile unsigned long *)(0x40013808)
#define TREG_EXTI_IMR   ((volatile unsigned long *)(0x40013C00)
#define TREG_EXTI_EMR   ((volatile unsigned long *)(0x40013C04)
#define TREG_EXTI_RTSR  ((volatile unsigned long *)(0x40013C08)
#define TREG_EXTI_FTSR  ((volatile unsigned long *)(0x40013C0C)
#define TREG_EXTI_PR    ((volatile unsigned long *)(0x40013C14)
```


TOPPERS

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 EXTI_PortSourceGPIOB  (0x01)
```

- 割込み設定定義

```
typedef  enum {DISABLE = 0, ENABLE = !DISABLE} FunctionState;

#define INT4_INT_LVL    0x0F
```

int_switch_1 : 割込み制御関連のレジスタの初期化

- NVICに割込み設定を行うinit_int関数を作成する

```
void
init_int(unsigned char no, unsigned char mpri, unsigned char spri, FunctionalState active){
    unsigned char tmppriority = 0x00, tmpsub = 0x0F, tmpgroup;
    if(active != DISABLE){
        tmpgroup =  (*TREG_SCB_AIRCR & 0x700) >> 8;
        tmpsub = (tmpgroup + __NVIC_PRIO_BITS) < 7 ? 0
                           : (tmpgroup - 7) + __NVIC_PRIO_BITS;
        tmppriority = mpri << tmpsub;
        tmppriority |= (spri & ((1 << tmpsub)-1));
        tmppriority = tmppriority << (8 - __NVIC_PRIO_BITS);
        TREG_NVIC_IP[no] = tmppriority;
        TREG_NVIC_ISER[no>>5] = 1<<(no & 0x1f);    /* 割込みを有効に */
    }
    else{
       TREG_NVIC_ICER[no>>5] = 1 << (no & 0x1f);   /* 割込みを無効に */
    }
}
```


TOPPERS

int_switch_1：SW1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

```
void
switch_push_init(unsigned int no, unsigned int lvl){
    unsigned long reg;
    unsigned int pinpos = PSW1_PINNO;
    *TREG_RCC_AHB1ENR |= RCC_AHB1ENR_GPIOBEN;
     :
    reg =  0x04 * (EXTI_PinSource0 & 0x03);
    TREG_SYSCFG_EXTICR[pinpos >> 0x02] &= ~(0x0F << reg);
    TREG_SYSCFG_EXTICR[pinpos >> 0x02] |= EXTI_PortSourceGPIOB << reg);
    *TREG_EXTI_IMR &= ~(1<<pinpos);
    *TREG_EXTI_EMR &= ~(1<<pinpos);
    *TREG_EXTI_IMR |= 1<<pinpos;
    *TREG_EXTI_RTSR &= ~(1<<pinpos);
    *TREG_EXTI_FTSR &= ~(1<<pinpos);
    *TREG_EXTI_RTSR |= 1<<pinpos;
    init_int(no, lvl, 0x0F, ENABLE);
}
```

引数に割込み条件を追加する

割込み許可

ライジング・エッジ

TOPPERS

int_switch_1 : データ共有用グローバル変数

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_count;
```

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
EXTI4_IRQHandler(void){
    unsigned long enablestatus = *TREG_EXTI_IMR & (1<<PSW1_PINNO);
    if((*TREG_EXTI_PR & (1<<PSW1_PINNO)) != 0 && enablestatus != 0){
        /* LEDをカウントアップ */
        led_out_bdigit(++led_count);
        *TREG_EXTI_PR = (1<<PSW1_PINNO);
    }
}
```

- 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

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(INTNO_EXTI4, INT4_INT_LVL);
    led_count = 0;
    for (;;) {
    }
}
```

何も行わない
無限ループ

TOPPERS

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，SW1（EXTI4）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

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_count;
```

- 割込みハンドラではカウントデータをクリアする

```
void
EXTI4_IRQHandler(void){
    unsigned long enablestatus = *TREG_EXTI_IMR & (1<<PSW1_PINNO);
    if((*TREG_EXTI_PR & (1<<PSW1_PINNO)) != 0 && enablestatus != 0){
        /* LEDをクリア */
        led_count =  0;
        led_out_bdigit(led_count);
        *TREG_EXTI_PR = 1<<PSW1_PINNO;
    }
}
```

データのクリア

外部割込み要因のクリア

TOPPERS

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(INTNO_EXTI4, INT4_INT_LVL);

    led_count = 0;

    for (;;) {
        led_count = inc_data(led_count);
        led_out_bdigit(led_count);
        busy_wait();
    }
}
```

LEDのカウントアップ

TOPPERS

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
  - 外部関数がライブリ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

TOPPERS

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

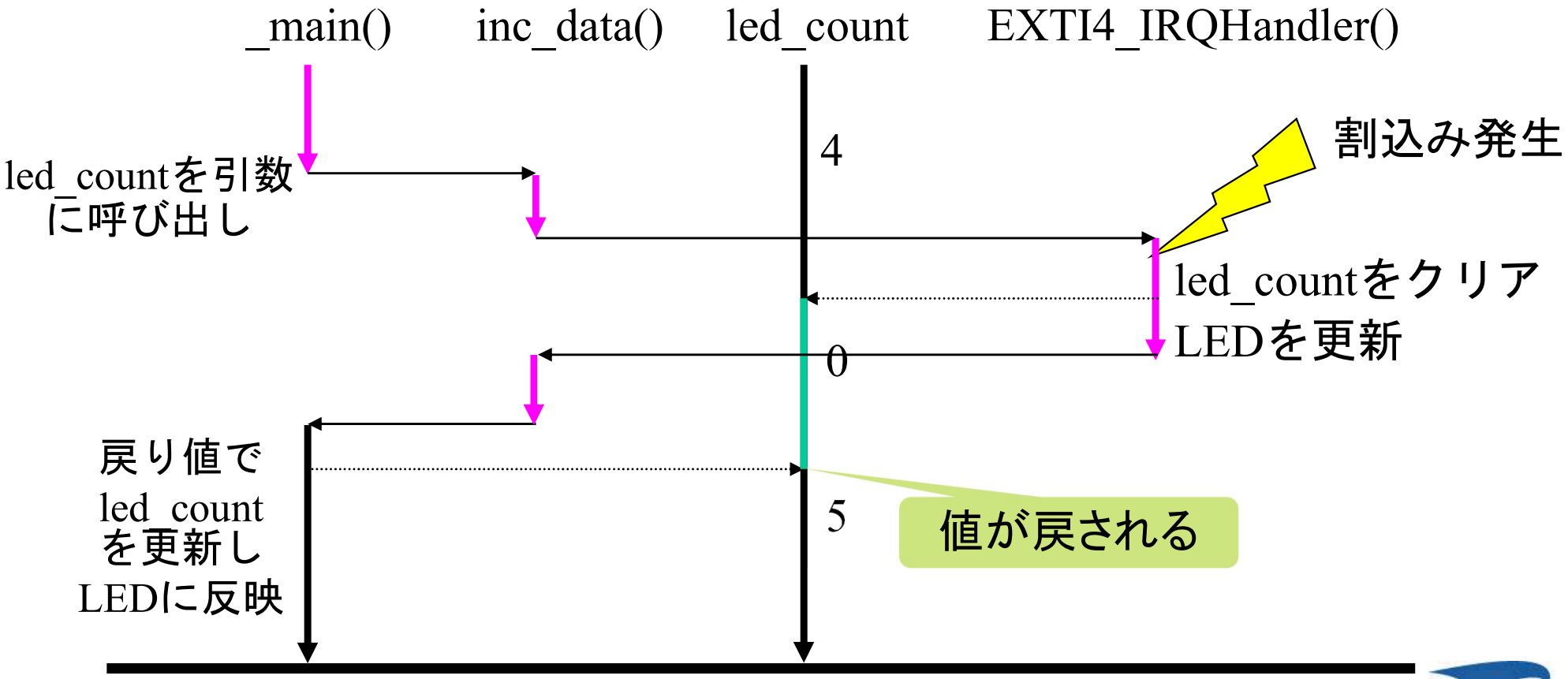
- SW1を押して割込みハンドラでLEDをクリアしても，次のカウントのタイミングで’1’ではなく，前のカウント値をインクリメントした値が表示される

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

int_switch_2 : led_countの更新

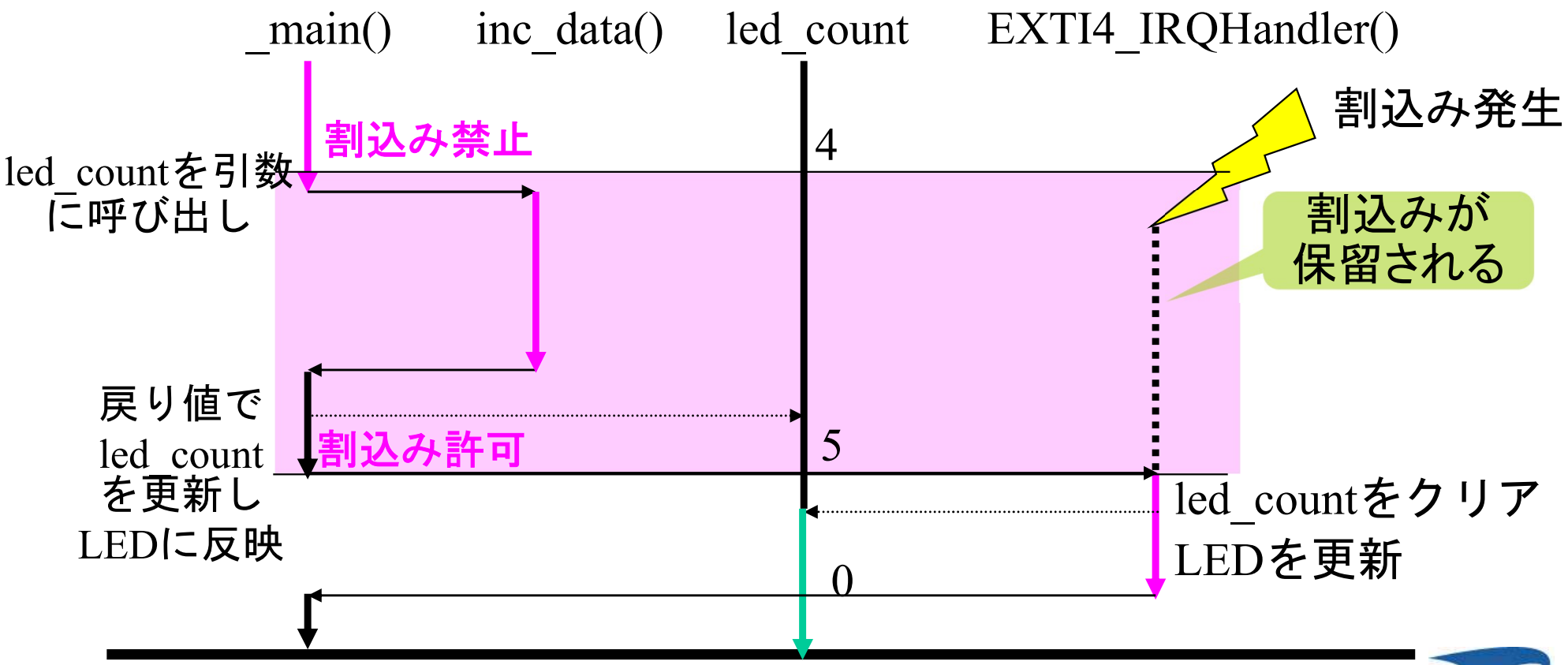
- _main()関数でled_countを読み込んでinc_data()を実行している間に，割込みが発生し，割込みハンドラでled_countをクリアすると，その後_main()関数によって上書きされてしまう



int_switch_2 : 割込みの禁止

- 共有データ(led_count)を排他的に扱えば問題は発生しない
- _main()関数でled_countを更新する間はSW1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



TOPPERS

int_switch_2：割込み許可禁止設定

- 割込み許可禁止関数を追加
  - Enable();　　割込み許可
  - Disable();　　割込み禁止

```
void Enable(void)
{
    asm volatile(“cpsie i”);
}

void Disable(void)
{
    asm  volatile(“cpsid i”);
}
```

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）

割込み禁止

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

割込みが発生しても受け付けられず，割込みハンドラと排他的に実行される

TOPPERS

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て実現
- 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

割込みプログラミング

1. Cortex-Mの割込みアーキテクチャ
2. スイッチ割込みプログラム
3. 割込みタイマ

タイマ割込みプログラム : int_timer 概要

- タイマ4の割込みを用いて1秒間隔でLEDをカウントアップする

- 学習内容
  - タイマ割込みの扱い方

- 作成方法
  - ポーリングプログラム timer をコピーして、ディレクトリ名をint_timerに修正して作成する

int_timer : 作成

プログラム int_timer を作成する

- 手順
  - 割込み要因ハードウエアの初期化（タイマ4）
    - カウントソース，カウント値を適切に設定してスタートさせる
  - 割込み制御レジスタの初期化
    - 割込み優先度の設定
  - 割込みハンドラ
    - LEDをカウントアップする

int_timer: init_int関数の取り込み

- int_switch_1で作成したinit_init関数をint_timer.cに取り込む
- FunctionalState定義も取り込みを行う

```
typedef enum {DISABLE = 0, ENABLE = !DISABLE} FunctionalState;
```

int_timer: NVIC制御関連のレジスタの定義

- 割込み制御レジスタ（自動でインクルード）

```
#define TREG_SCB_AIRCR  ((volatile unsigned long *)0xE00ED0C)
#define TREG_NVIC_ISER  ((volatile unsigned long *)0xE000E100)
#define TREG_NVIC_ICER  ((volatile unsigned long *)0xE000E180)
#define TREG_NVIC_IP    ((volatile unsigned char *)0xE000E400)
```

- TIMER4割込みレジスタ

```
#define TREG_TIM4_DIER   ((volatile unsigned short *)(0x4000080C)
```

- TIM4_DIER定義

```
#define TIM_DIER_UIE   0x0001   /* update interrupt enable */
```

int_timer : TIMER4ハードウェアの初期化

- オートリロード設定を行えば，一度スタートさせれば，設定周期で割込みが発生する

```
void
timer4_init(void){
    unsigned short PrescalerValue = 0;
    :
    *TREG_TIM4_EGR = 0x0001;
    /* Enable TIM4 Preload register on ARR */
    *TREG_TIM4_CR1 |= TIM_CR1_ARPE;
    *TREG_TIM4_SR = 0x0000;
    /* カウントスタート */
    *TREG_TIM4_CR1 |= TIM_CR1_CEN;
    /*割込み設定 */
    *TREG_TIM4_DIER |= TIM_DIER_UIE;
    init_int(INTNO_TIM4, 0x0E, 0x0F, ENABLE);
}
```

オートリロード設定

TOPPERS

int_timer : 割込みハンドラ

- 呼び出されると，LEDをカウントアップ
- led_countをグローバス変数に修正

```
unsigned char led_count;

void
TIM4_IRQHandler(void){
    /* 割込みクリア */
    *TREG_TIM4_SR = 0x0000;
    led_out_bdigit(led_count++);
}
```

int_timer：MAIN関数

- メイン関数ではled_countをグローバル変数に修正を行った対応

```
void
_main(void){
    led_init();
    timer4_init();

    led_count = 0;

    for(;;){
    }
}
```

シリアルI/Oの割込み対応

- Appendexとしてデバイスの割込み演習を用意しました
  - シリアルI/Oを割込みで動作させるint_uart
- program/int_uartを参考にしてください

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.sessame.jp/

基礎1実習通信講座終了の方へ

- my_programには、以下のディレクトリができ正しくプログラムが実行されているはずです
- my_programをzip圧縮して、TOPPERS事務局に送り、講座終了判定を行ってください

実習済プログラム	
int_switch_1	: 割込みによるプッシュスイッチプログラム
int_switch_2	: 割込みプッシュスイッチクリティカルセクション検証
int_timer	: タイマー割込みLEDカウントプログラム
led_count	: ソフトウェアタイマーLEDカウントプログラム
led_shift	: LEDシフトプログラム（コピー）
sample	: サンプルプログラム（コピー）
switch_push	: ポーリングによるスイッチ検知
timer	: ハードウェアタイマーLEDカウントプログラム
uart	: ポーリング方式UART通信

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）
- STM32F401-Nucleo用のプログラム作成、テキストの修正
  - 竹内良輔（（株）リコー）