［ET2016:ハードウェアトラック］

I2C通信の技術説明とドライバ作成方法

TOPPERSプロジェクト教育WG 主査

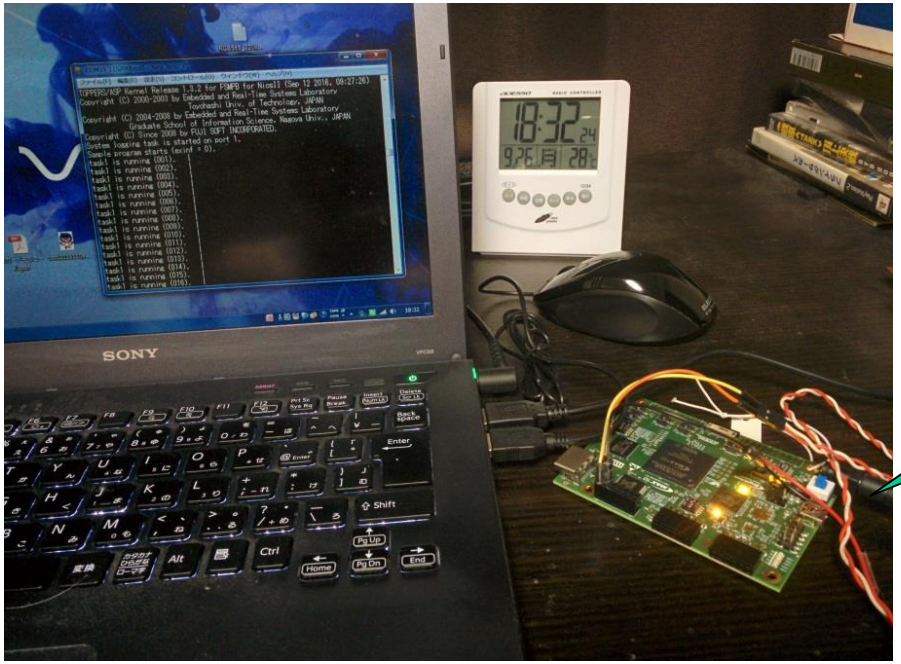
竹内良輔

TOPPERS

エレキ屋さんが、プログラム開発
ソフト屋さんが、ロジック合成

- エレキとソフトの垣根がなくなりつつある・・・
- 自宅のパソコンでFPGA開発：ALTREA(Intel)

  EK-10M50F484ボードでロジック合成



TOPPERS

1.シリアルデバイスI/F
2.I2C仕様(v2.1)
3.開発環境構築（ハード）
4.開発環境構築（ソフト）
5.デバイス・ドライバ内容紹介
6.各デバイス通信ソフトウェア

1.シリアルデバイスI/F

デバイスとのシリアル通信

- LCD、センサー、EEPROM等MPU外にあるデバイスとデータ交換を行う通信としてシリアル通信が主流となっている
  - 少ない外部ピンでペリフェラルと接続したい
  - SoCに関係なくペリフェラルを活用したい
- 現在、主流の周辺デバイスとのシリアル通信
  - UART(一対一の通信)
    - 二点式（ハードフローを使用の場合４点）、低速
  - I2C(Inter-Integrated Circuit)
    - 二点式、低速
  - SPI(Serial Peripheral Interface)
    - 四点式、中速

1.シリアルデバイスI/F
2.I2C仕様(v2.1)
3.開発環境構築（ハード）
4.開発環境構築（ソフト）
5.デバイス・ドライバ内容紹介
6.各デバイス通信ソフトウェア

2.I2C仕様(V2.1)

I2C通信(1):概要

- I2C通信のもとになる設計は1980年代の初めにフィリップス社(現NXP社)が内部バス用に設計したものである
- 2014年までに、いくつかのバージョンアップを行っているが、一般的に搭載されているものは、2001年に制定されたV2.1の仕様に準じている
  - 拡張仕様を使うと、接続できないSoCが多くなる
- 現在では、特許が失効しており、ロイヤリティなしで使用が可能となっている

I2C通信（1）：特徴V2.1

- シリアル・データ・ライン(SDA)とシリアル・クロック・ライン(SCL)の２本のバス・ラインのみで構成
- バスに接続される各デバイスはそれぞれ固有のアドレスをもち、それを元にソフトウェアによる各デバイスのアドレス指定が可能．また、デバイス間には、マスターとスレーブという関係が成立．マスターはマスター・トランスミッタまたはマスターレシーバとして機能
- 万一、複数のマスターが同時にデータ転送を開始しようとした場合でも、データ破壊を防ぐために衝突検出機能およびアービテーションを備えたマルチ・マスター・バス

I2C通信（2）：特徴V2.1

- 8ビットの双方向シリアル・データ転送を標準モードでは100kbit/s、ファースト・モードでは400kbit/s、ハイスピード(Hs)モードでは3.4Mbit/sで行うことが可能
- オンチップ・フィルタによりバス上のスパイクを防ぎ、データの信頼性を維持
- バス上の静電容量が400pF以下であれば、1つのバス上にICをいくつでも接続することが可能

I2C通信(1):仕様

- 一般的な用語の定義

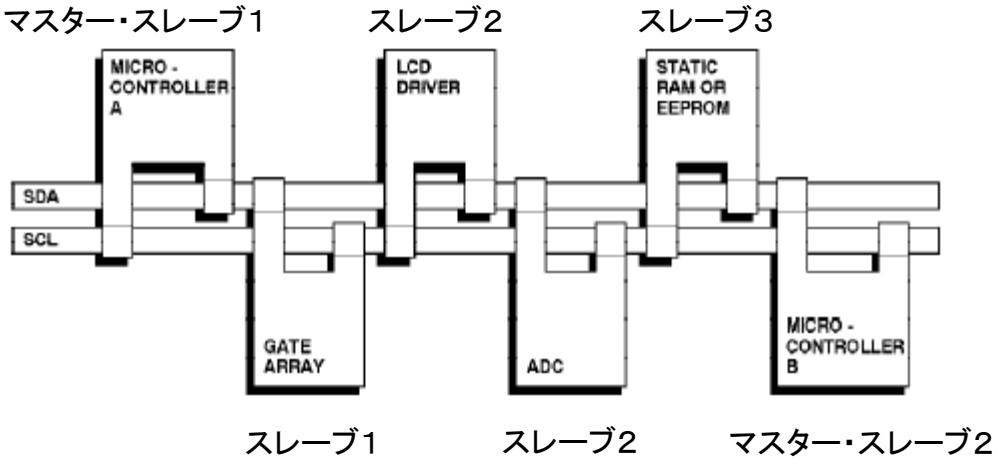
トランスミッタ	データをバスに送信するデバイス
レシーバ	データをバスから受信するデバイス
マスター	データ転送を開始し、クロック信号を生成し、データ転送を終了するデバイス
スレーブ	マスターからアドレス指定されるデバイス
マルチ・マスター	メッセージを失うことなく、複数のマスターが同時にバスをコントロールすること
アービトレーション	複数のマスターが同時にバスをコントロールしようとしたときに、1つのマスターだけがバスをコントロールできるようにし、さらに、メッセージが失われたり内容が変更されないようにする手段
同期化	複数のデバイスのクロック信号の周期をとるための手段

I2C通信(2):仕様

- 以下の図では、1つのバスに2つのマスターをもつ

マイクロコンピュータAがマイクロコンピュータBに送信する場合
①A(マスター)がB(スレーブ)にアドレスを指定
②A(マスター・トランスミッタ)がB(スレーブ・レシーバ)にデータを送信
③A(マスター)によってデータ送信を終了

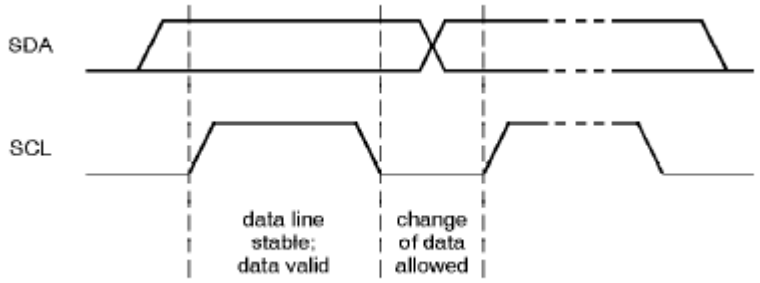
マイクロコンピュータAがマイクロコンピュータBから情報を受信する場合
①A(マスター)がB(スレーブ)にアドレスを指定
②B(スレーブ・トランスミッタ)がA(マスター・レシーバ)にデータを受信
③A(マスター)によってデータ受信を終了



TOPPERS

I2C通信（3）：仕様

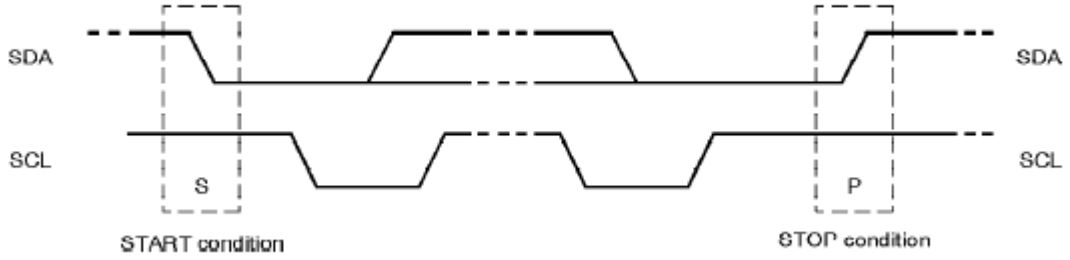
- データの有効性



クロックが"H"の間にはSDAラインの状態は一定でなければならない。データ・ラインが"H"と"L"の間で状態を変更できるのは、SCLラインのクロックが信号が"L"のときに限られる

- 「START」条件と「STOP」条件

I2Cバスの通信手順では「START」条件および「STOP」条件という固定の状態が発生する
SCLが"H"のときSDLラインが"H"から"L"に変化する状態を「START」条件と呼ぶ
SCLが"H"のときSDLラインが"L"から"H"に変化する状態を「STOP」条件と呼ぶ
「START」条件と「STOP」条件は常にマスターが生成する

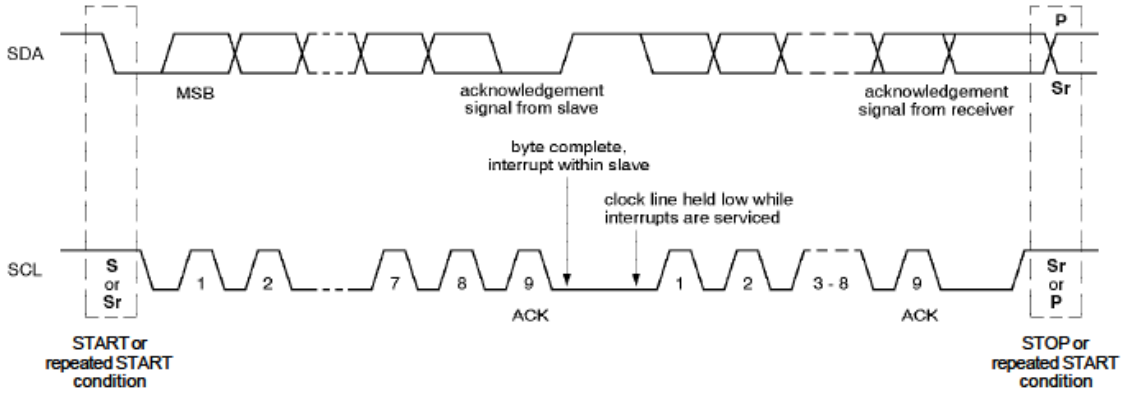


TOPPERS

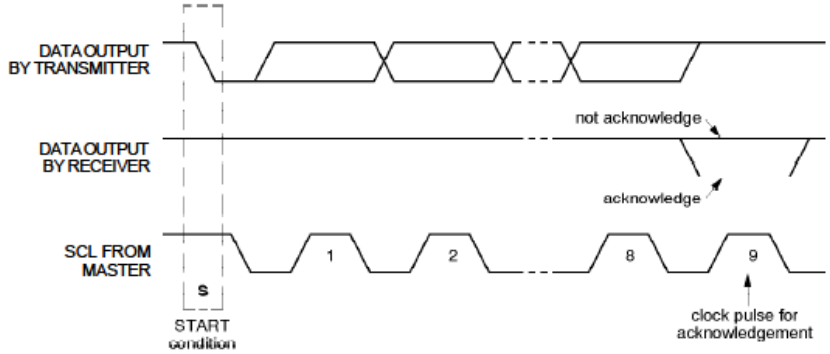
I2C通信(4):仕様

- バイトのフォーマット

SDAラインに出力されるバイトの長さは8ビットで、一回の転送に伝達できるバイト数には制限がない。各バイトの後ろにはアクノレッジ・ビットであり、データは最上位ビット(MSB)から順に送信する



- アクノレッジ

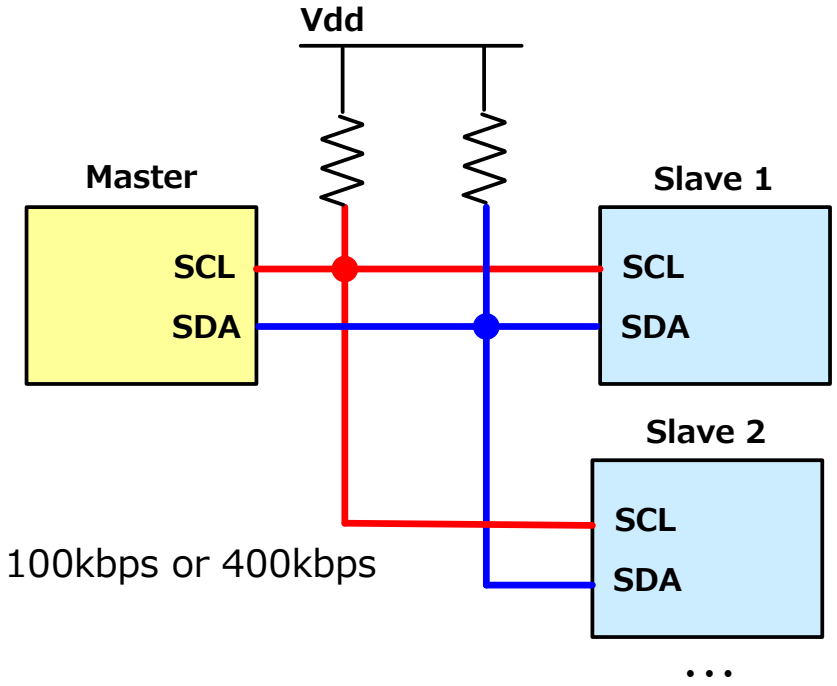


データ転送を行う場合、必ずアクノレッジが必要となる。アクノレッジのクロックはマスターが生成し、生成時トランスミッターはSDAラインを解放("H")する
レシーバはアクノレッジ・クロック・パルスが"H"のとき、SDLラインが"L"で安定するように設定する
レシーバは、スレーブアドレスを確認できなかった場合、データ受信できなかった場合、アクノレッジ・クロック・パルスに対してSDAラインを解放する

TOPPERS

I2C通信(1):接続例

- I2C(Inter-Integrated Circuit)は周辺デバイスとのシリアル通信方式

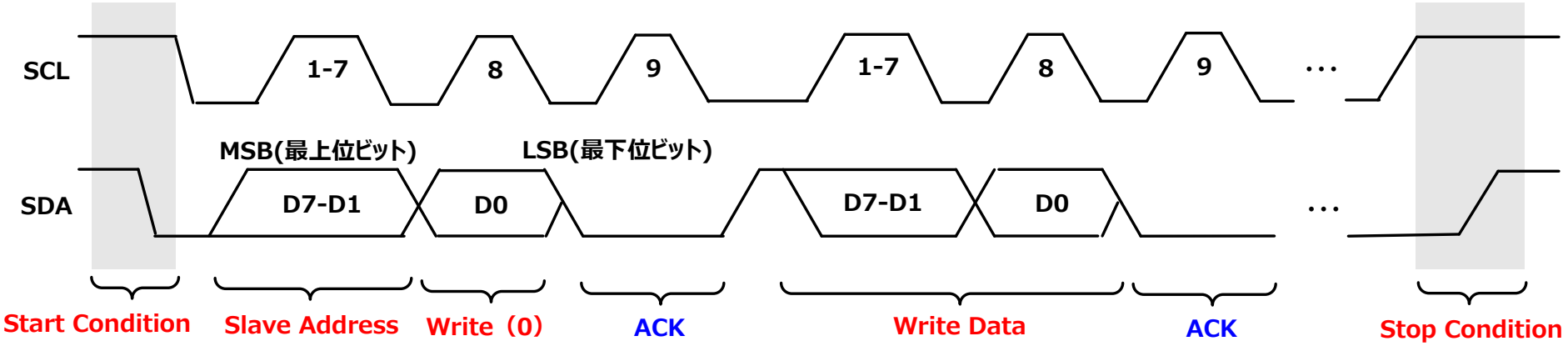


- クロック信号(SCL)とデータ信号(SDA)の2線で通信を実施する
- 各スレーブがスレーブアドレスを持ち、データの中にアドレスを含む
- 通信を開始するのは全てマスタ側でSCLを基準にデータがSDA上で転送
- スレーブアドレスが一致したデバイスのみ、送受信を継続する仕組み

TOPPERS

I2C通信（2）：マスタからスレーブへ送信の場合

スレーブアドレスが7bitの時のタイミングチャート



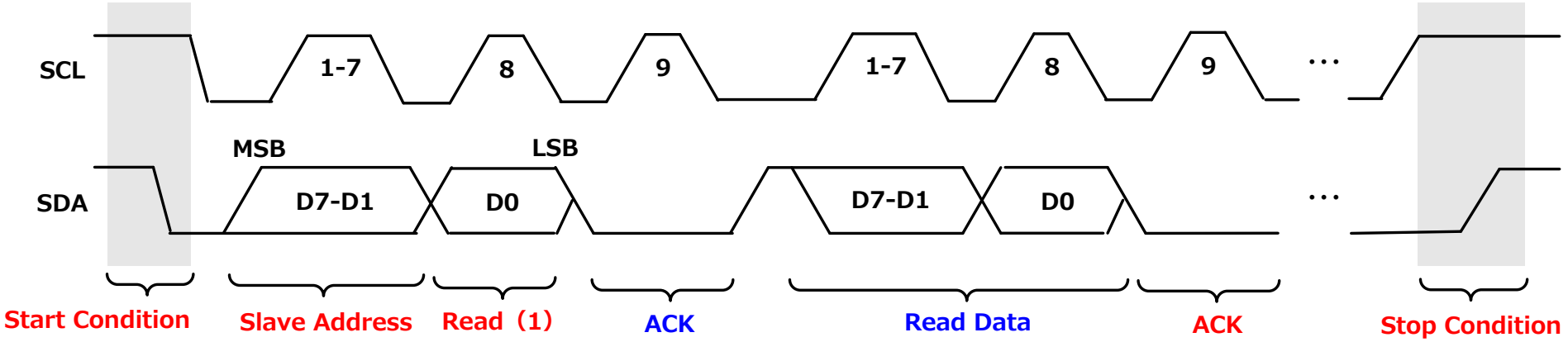
※赤：マスタ側、青：スレーブ側

- SCLがHighの間にマスタがSDAをLowとするとStart条件になる
- マスタはSCLがHighの間にSDAをHighにすることでStop条件となる（それぞれ通常のデータ時はSCLがLowの時しか変化しない）
- スレーブ側はデータの取り出しが完了するまでBUSYとしてSCLを強制的にLowとすることで、見かけ上クロックがなくなるので、マスタ側は次のデータ出力を待つことになる

TOPPERS

I2C通信(3):スレーブからマスタへ受信の場合

スレーブアドレスが7bitの時のタイミングチャート



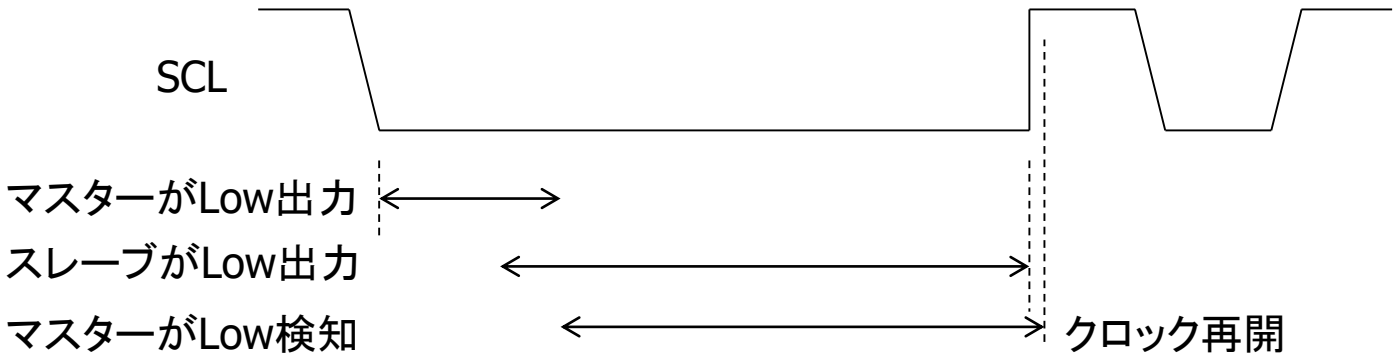
※赤：マスタ側、青：スレーブ側

- スレーブ側からのデータ入力に応じて、マスタ側はACKを自動返信する

TOPPERS

クロックストレッチ（ウエイト）

- スレーブがセンサーなどの場合、マスターへの送信で待ちを要求する場合がある
- この待ちは、マスターがSCLにLow出力するタイミングで、スレーブがSCLにLow出力することによりマスターに待ちを伝達する、マスター側はSCLを入力検知しHiになるまで動作待ちを行わなければならない
- マスター側にウエイト検知機能がない場合、ドライバを改造してソフト待ちを入れなければならない



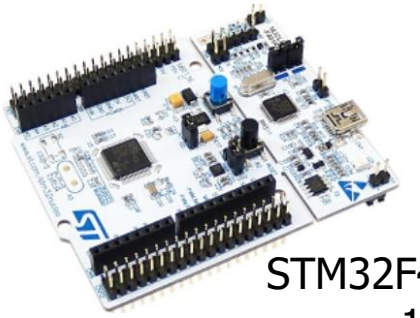
TOPPERS

1.シリアルデバイスI/F
2.I2C仕様(v2.1)
3.開発環境構築（ハード）
4.開発環境構築（ソフト）
5.デバイス・ドライバ内容紹介
6.各デバイス通信ソフトウェア

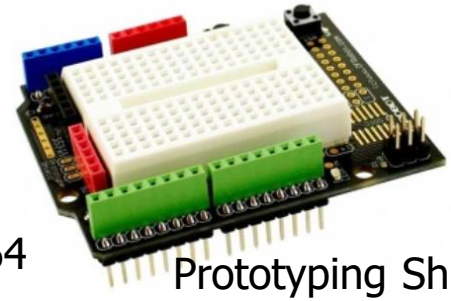
3.開発環境構築（ハード）

PCBを作成しなくても、ソフト開発は可能

- 一般的な開発では、PCB作成前のソフト開発はシミュレーション開発が主流
- I2Cは低クロックのデバイス制御なので、ブレッド・ボードを活用すればPCB作成前から本格的なソフトウェア開発が可能



STM32F401 nucleo64
1500円



Prototyping Shield
1100円

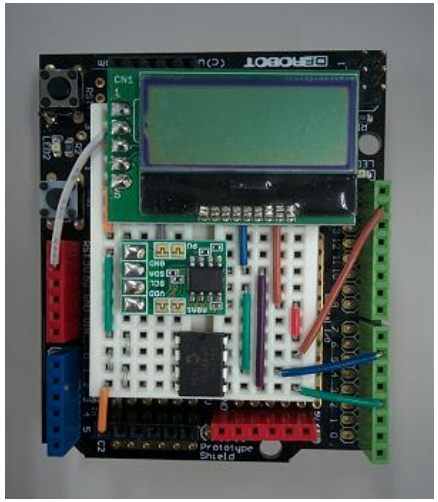
I2C DEVICE

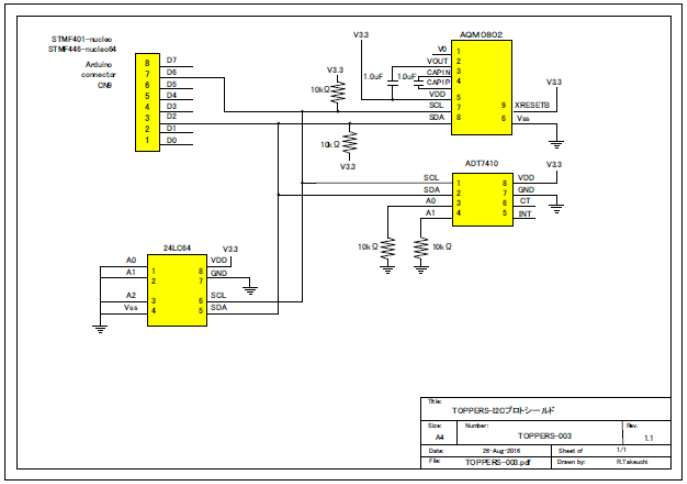
Arduino Uno Rev3
2940円

Arduinoはドライバをライブラリで提供

I2C (Prototyping) Shield

- Prototyping Shieldを用いて、I2Cに対応したキャラクタLCD、温度センサー、EEPROMを配置してI2Cの評価を行う
- サンプルプログラムでは、温度センサーから読み取った温度を1秒単位にLCDに表示を行う。EEPROMはreadプログラムを用意
- Prototyping ShieldがD14,D15ピンに対応していないため、拡張用にI2Cのインターフェイスをもつ、401nucleo-64,446 nucleo-64が実行対象であるが、D14,15ピンに対応しているシールドを使用すればArduinoコネクタをもつすべてのボードで対応可能

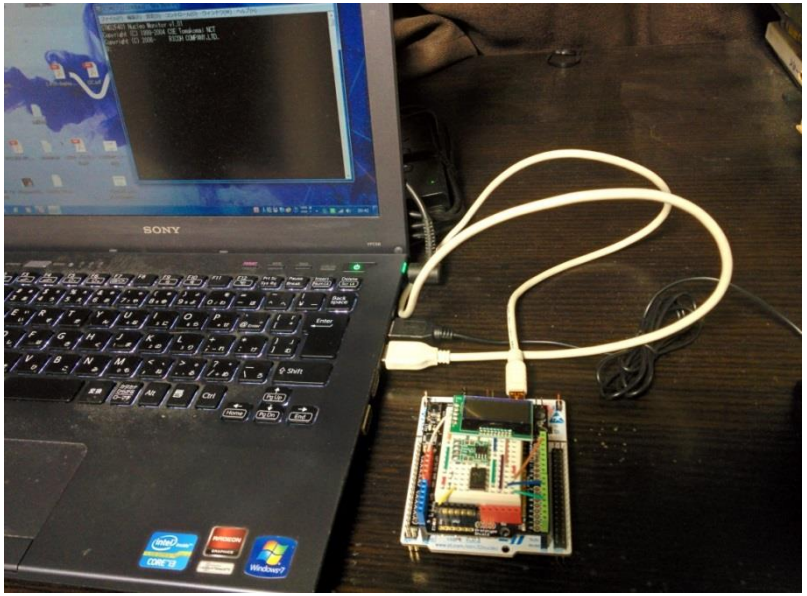




対応ボード
STM32F401 nucleo-64
STM32F446 nucleo-64

NucleoとI2C (Prototyping) Shieldを接続

- STM32F401 NucleoとI2C (Prototyping) Shield をドッキングして、USB接続すると、ST-LINKV2と仮想COMが使用可能となる



1.シリアルデバイスI/F
2.I2C仕様(v2.1)
3.開発環境構築（ハード）
4.開発環境構築（ソフト）
5.デバイス・ドライバ内容紹介
6.各デバイス通信ソフトウェア

4.開発環境構築（ソフト）

ソフトウェア開発環境

- ソフトウェア開発環境としてTOPPERS BASE PLATFORMV1.1 for STM32F4xxを使用
- Windows上の開発環境としてオープンソースを組み合わせた環境を使用
  - Bash環境としてCygwinまたはMinGW等を使用
  - ARMコンパイラとしてGNU-ARMを使用
  - RTOSはTOPPERS/ASP
  - STM32F401 Nucleoのデバイスドライバをオープンソースで提供
- 開発環境のインストール方法は「基礎セミナーの開発環境編」に記載

TOPPERS BASE PLATFORM V1.1

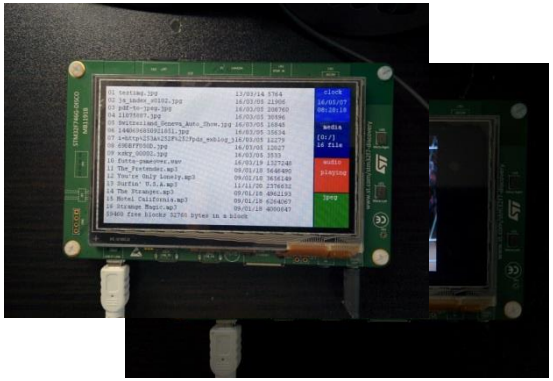
- ASP-1.9.2上に5つのレイアでプラットフォームを構築

BASE PLATFROM V1.1		STM32F4xx	STM32F7xx	detail	contents
Device driver	Base driver	gpio	gpio		新基礎2
		dma	dma		
		(timer)	(timer)		
		uart	uart		
	Standard driver	i2c	i2c	CLCD/senser	新基礎3 Reference Manual
		spi	spi	GLCD/SDcard	
		adc	adc	Joy stick	
		usb OTG	usb OTG	MSC/HID	
	M7F depend driver		ltdc	GLCD	Application Manual Reference Manual
			tp		
			rtc	Clock	
			sdmmc	SDcard	
			audio		
api middleware	File System	fatfs	fatfs	FAT	新基礎3
		file library	file library	C language file	
		stdio	stdio	STDIO	
	(Open source) library		STM-BSP	GUI	
			libjpeg	JPEG	
			libmad	MP3	

シールドの組み合わせで種々の開発が可能

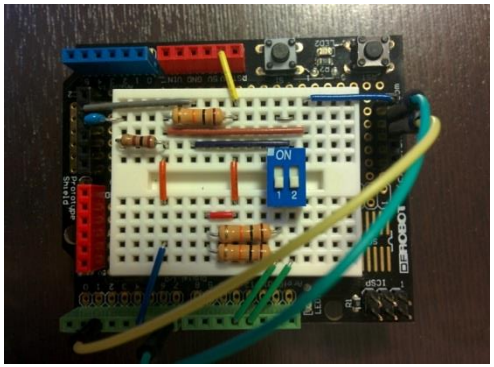
- インターフェイス2016年12月号に紹介

TOPPERSブースでデモ中



SD-Card Player

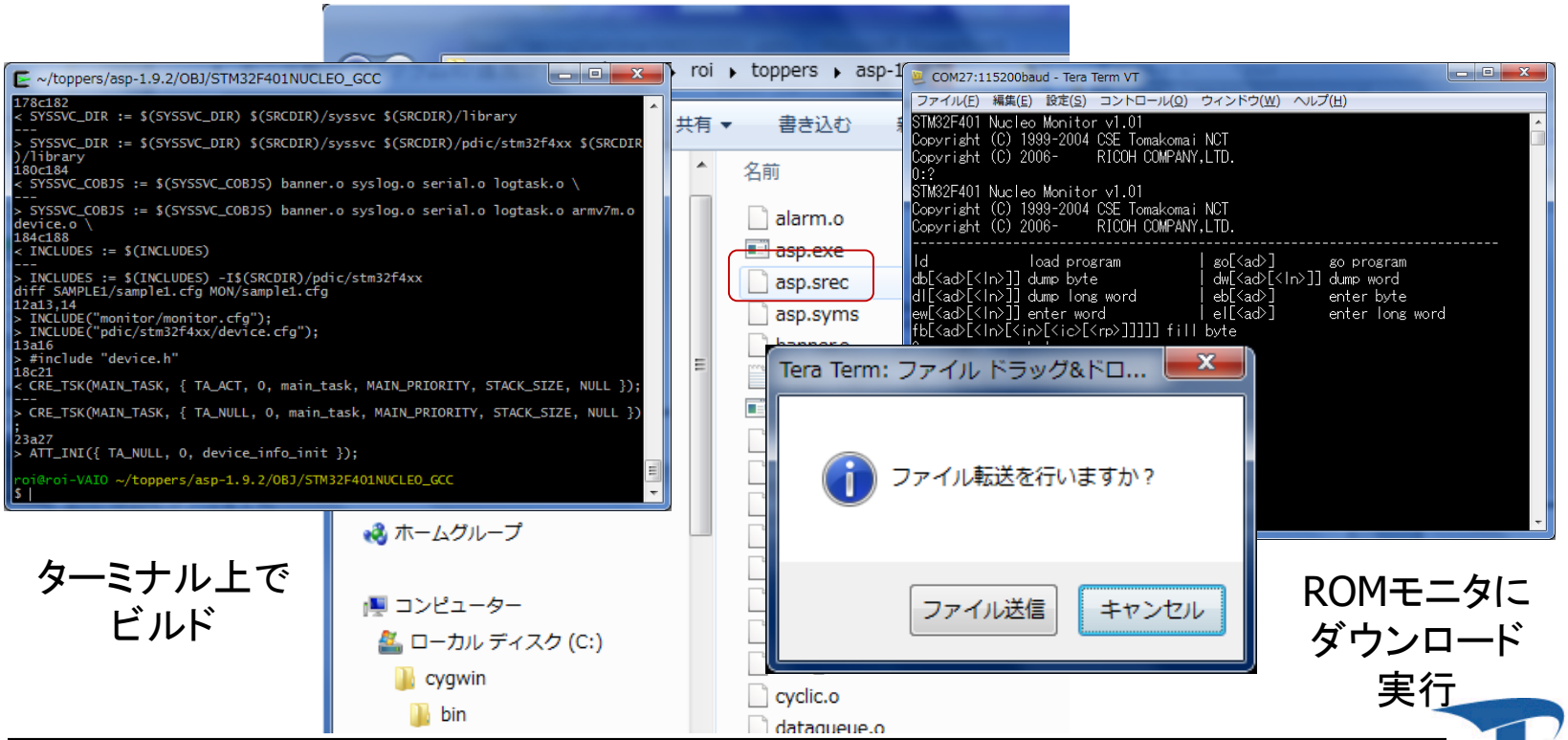
Adafruit1.8” TFTシールド



セミナー教材用シールド

ターミナル画面で、Bashコマンドでビルド

- ターミナル上でデバイスドライバ＋テストプログラムをビルド（asp.srecができる）これを、ダウンロード後実行



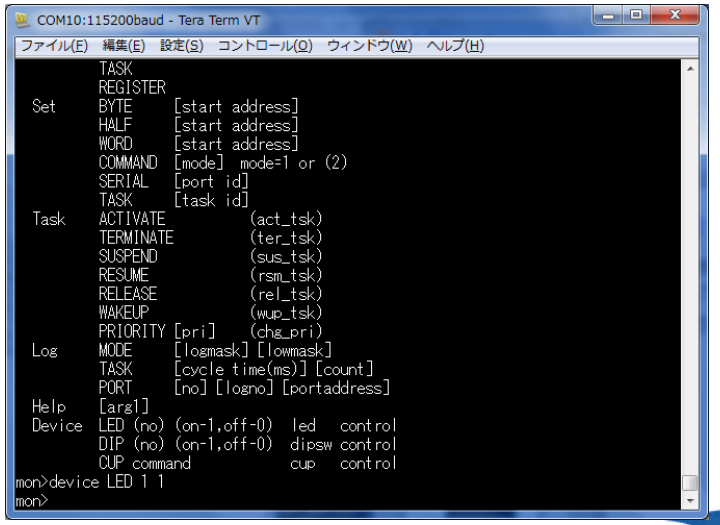
ターミナル上でビルド

ROMモニタにダウンロード実行

syslog文やprintfでログ表示、テストコマンド拡張

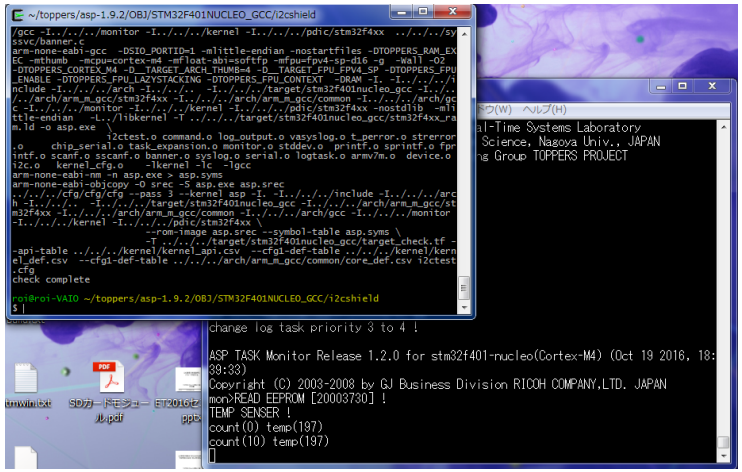
- オーソドックスなデバッグ手法
  - UNIX等のプログラム開発と同じ
- プログラム中で変数の値を表示させたり、パスの確認のために使用
- テスト用のプログラムのコマンド実行

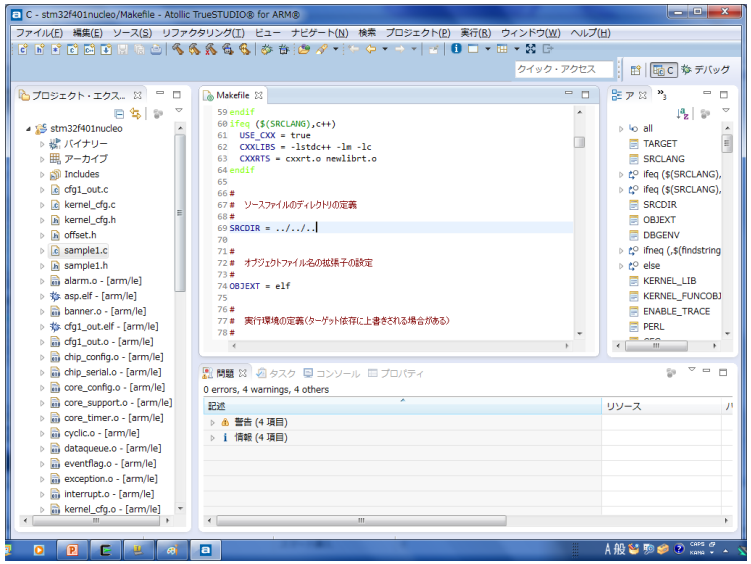
波形測定やハードテストを用意にソフトサポート



統合開発環境に変更が可能

- b-shell(bash)ベースの開発環境となる
- MakefileプロジェクトはTureSTDIOで簡単にプロジェクトとして変換できるため、統合環境化が可能





TOPPERS

対応ボード一覧

- TOPPERS BASE PLATFORMV1.1は6つのSTMボードに対応する
- Arduinoコネクタに対応して4つのシールドに対応して、評価、実験を行うことができる。（商品として発売されるボードは2つ）
- USBコネクタが付いたボードはUSB-OTGに対応して、MSCやHIDの評価、実験が可能
- STM32F746 Discoveryボードでは、LCD、オーディオ、SDMMC、USB-OTGに対応している

▲はD15,14ピンに接続した場合

	基礎2	LCDシールド 新基礎3	I2Cシールド	BLEシールド	EPCシールド 開発中	USB-OTG
STM32F401RE nucleo	○	○	○	○	○	×
STM32F4 Discovery	○	-	-	-	-	○
STM32F746 Discovery	-	-	×	×	○	○
STM32F446RE nucleo-64	○	○	○	○	○	×
STM32F446ZE nucleo-144	○	○	▲	×	○	○
STM32F746ZG nucleo-144	○	○	▲	×	○	○

1.シリアルデバイスI/F
2.I2C仕様(v2.1)
3.開発環境構築（ハード）
4.開発環境構築（ソフト）
5.デバイス・ドライバ内容紹介
6.各デバイス通信ソフトウェア

5.デバイス・ドライバ内容紹介

テストプログラムとデバイスドライバ

- テストプログラム

  目的:デバイスのハードウェア機能確認のためのプログラム

  – すべてのハード機能が評価できるように作成する
  – セルフテスト設定もある

- デバイスドライバ

  目的:アプリからデバイスを操作するためのプログラム

  – アプリで使用する機能をAPIに従って操作する
  – システム的に高品質、高速になる機能を用いて作成
  – 入力、出力のパラメータ検証が必要
  – 問題発生時、解析ができるように作成する
  – RTOSやハードウェアが変わっても使用できるよう考慮する

I2CドライバAPI

- ポートIDよりハンドラを取出し操作
- 割込みを使用するが、I/F上は隠蔽する
- スレーブは通常使用しないので説明しない

関数	型	引数	機能
i2c_init	I2C_Handler *	ID portid I2C_Init_t* hi2c	指定ポートIDのI2Cペリフェラルを初期化、ハンドラへのポインタを返す
i2c_deinit	ER	I2C_Handler* hi2c	I2Cを未初期化状態に戻す
i2c_slavewrite	ER	I2C_Handler* hi2c uint8_t* pData uint16_t Size	スレーブモードのデータ送信
i2c_slaveread	ER	I2C_Handler* hi2c uint8_t* pData uint16_t Size	スレーブモードのデータ受信

I2CドライバAPI

- デバイスにより、アドレスを指定するものとしないものがある。MemAddSizeの設定により両方のシーケンスをサポートする

関数名	型	引数	機能
i2c_memwrite	ER	I2C_Handler* hi2c uint16_t DevAddr uint16_t MemAddr uint16_t MemAddSize uint8_t* pData uint16_t Size	マスターモードのデータ送信、MemAddSizeにゼロを設定するとアドレス設定を行わない
i2c_memread	ER	I2C_Handler* hi2c uint16_t DevAddr uint16_t MemAddr uint16_t MemAddSize uint8_t* pData uint16_t Size	マスターモードのデータ受信、MemAddSizeにゼロを設定するとアドレス設定を行わない

I2C：マスター送信ドライバ（1）

- ドライバの型はμITRON仕様に従いER型を使用している
- パラメータエラーはE_PAR、状態エラーはE_OBJとしている

```
ER
i2c_memwrite(I2C_Handle_t *hi2c, uint16_t DevAddress, uint16_t MemAddress,
                        uint16_t MemAddSize, uint8_t *pData, uint16_t Size)
{
    ER ercd = E_OK;
    if(pData == NULL || Size == 0)
        return  E_PAR;
    if(hi2c->Init.AddressingMode != I2C_ADDRESSINGMODE_7BIT && MemAddSize != 0)
        return  E_PAR;
    if(hi2c->status != I2C_STATUS_READY)
        return E_OBJ;
    if(i2c_busreadywait(hi2c, I2C_TIMEOUT_BUSY_FLAG) != E_OK)
        return E_OBJ;
    /*
     * 書き込みロック
     */
    if(hi2c->Init.semlock != 0)
        wai_sem(hi2c->Init.semlock);
```

パラメータのデータチェック

ハンドラ毎にセマフォで排他制御を設定

TOPPERS

I2C：マスター送信ドライバ（2）

```
    /*
     * Acknowledge/PEC Position禁止
     */
    sil_andw_mem((uint32_t *)(hi2c->base+TOFF_I2C_CR1), I2C_CR1_POS);
    hi2c->status = I2C_STATUS_BUSY_TX;
    hi2c->ErrorCode = I2C_ERROR_NONE;
    hi2c->pBuffPtr = pData;
    hi2c->XferSize = Size;
    hi2c->XferCount = Size;
    /* スレーブアドレスと書込みアドレスの設定 */
    if((ercd = i2c_masterwriteaddress(hi2c, DevAddress, MemAddress, MemAddSize, I2C_TIMEOUT_FLAG)) != E_OK){
       if(hi2c->Init.semlock != 0)
           sig_sem(hi2c->Init.semlock);
           return ercd;
    }
    /*
     * 書き込み割込み許可
     */
    sil_orw_mem((uint32_t *)(hi2c->base+TOFF_I2C_CR2), (I2C_CR2_ITEVTEN | I2C_CR2_ITBUFEN | I2C_CR2_ITERREN));
    ercd = i2c_transwait(hi2c, 500);
    /*
     * 書き込みロック解除
     */
     if(hi2c->Init.semlock != 0)
         sig_sem(hi2c->Init.semlock);
     return ercd;
}
```

ハンドラに通信データをセットする
割込み内で参照

次のページのサブルーチンへ

スレーブアドレスとデバイスアドレスをセットする

データの転送は割込みサービスルーチンで（3ページ先）

割込み設定と終了待ち

排他制御解除

I2C：マスター送信ドライバ（3）

- スレーブアドレスとデバイスアドレスを設定する関数として使用するi2c_mesaterwriteaddressを解説する

```
static ER
i2c_masterwriteaddress(I2C_Handle_t *hi2c, uint16_t DevAddress, uint16_t MemAddress, uint16_t MemAddSize, uint32_t Timeout)
{
   ER ercd = E_OK;
   /*
    * スタート設定と設定待ち
    */
   sil_orw_mem((uint32_t *)(hi2c->base+TOFF_I2C_CR1), I2C_CR1_START);
   if(i2c_sr1flagsetwait(hi2c, I2C_SR1_SB, Timeout) != E_OK){
      return E_TMOUT;
   }
   /*
    * デバイスアドレス設定と設定待ち
    */
   if(hi2c->Init.AddressingMode == I2C_ADDRESSINGMODE_7BIT){
      sil_wrw_mem((uint32_t *)(hi2c->base+TOFF_I2C_DR), I2C_7BIT_ADD_WRITE(DevAddress));
   }
```

スタート状態設定

7ビットスレーブアドレス設定

TOPPERS

I2C：マスター送信ドライバ（4）

```
else{
    sil_wrw_mem((uint32_t *)(hi2c->base+TOFF_I2C_DR), I2C_10BIT_HEADWRITE(DevAddress));
    if((ercd = i2c_addrflagwait(hi2c, I2C_SR1_ADD10, Timeout)) != E_OK){
        return ercd;
    }
    /*
     * 10ビットアドレス設定と設定待ち
     */
    sil_wrw_mem((uint32_t *)(hi2c->base+TOFF_I2C_DR), I2C_10BIT_ADDRESS(DevAddress));
}
/*
 * アドレスフラグ設定待ち
 */
if((ercd = i2c_addrflagwait(hi2c, I2C_SR1_ADDR, Timeout)) != E_OK){
    return ercd;
}
/*
 * アドレスフラグクリア
 */
i2c_clear_addr(hi2c);
/*
 * メモリアドレス設定なしなら正常終了
 */
if(MemAddSize == 0)
    return E_OK;
```

10ビットスレーブアドレス設定

スレーブアドレス設定終了待ち

アドレス設定クリア

デバイスアドレスを設定しない場合はここで終了

I2C：マスター送信ドライバ（5）

- デバイスアドレスがある場合は8ビット、または、16ビットのデバイスアドレスをセットする

```
    /*
     * TXEフラグ設定まで待ち
     */
    if(i2c_sr1flagsetwait(hi2c, I2C_SR1_TXE, Timeout) != E_OK){
        return E_TMOUT;
    }
    /*
     * メモリアドレス設定と設定待ち
     */
     if(MemAddSize == I2C_MEMADD_SIZE_8BIT){
         sil_wrw_mem((uint32_t *)(hi2c->base+TOFF_I2C_DR), I2C_MEM_ADD_LSB(MemAddress));
     }
     else{              /* 16ビットケース */
         sil_wrw_mem((uint32_t *)(hi2c->base+TOFF_I2C_DR), I2C_MEM_ADD_MSB(MemAddress));
         if(i2c_sr1flagsetwait(hi2c, I2C_SR1_TXE, Timeout) != E_OK){
             return E_TMOUT;
         }
         sil_wrw_mem((uint32_t *)(hi2c->base+TOFF_I2C_DR), I2C_MEM_ADD_LSB(MemAddress));
     }
    return E_OK;
}
```

送信終了待ち

デバイスアドレスを設定

I2C:割込みハンドラ(1)

- 送信の場合は、スレーブアドレスまたはデバイスアドレス転送終了後TXE割込みが発生し、割込み中で転送データ設定を行う

```
void
i2c_ev_handler(I2C_Handle_t *hi2c)
{
   uint32_t sr1 = sil_rew_mem((uint32_t *)(hi2c->base+TOFF_I2C_SR1));
   uint32_t sr2 = sil_rew_mem((uint32_t *)(hi2c->base+TOFF_I2C_SR2));
   uint32_t cr2 = sil_rew_mem((uint32_t *)(hi2c->base+TOFF_I2C_CR2));
   volatile uint32_t tmp;
   /*
    * マスターモード
    */
   if((sr2 & I2C_SR2_MSL) != 0){
      if((sr2 & I2C_SR2_TRA) != 0){          /* 送信モード */
         if(((sr1 & I2C_SR1_TXE) != 0) && ((cr2 & I2C_CR2_ITBUFEN) != 0) && ((sr1 & I2C_SR1_BTF) == 0)){
            i2C_MasterTransmit_TXE(hi2c);
         }
         else if(((sr1 & I2C_SR1_BTF) != 0) && ((cr2 & I2C_CR2_ITEVTEN) != 0)){
            i2c_mastertrans_BTF(hi2c);
         }
      }
      else{                                  /* 受信モード */
```

データ送信
サブルーチン

データ送信終了処理
サブルーチン

I2C:割込みハンドラ(2)

```
static void
i2C_MasterTransmit_TXE(I2C_Handle_t *hi2c)
{
   /* １バイト送信 */
   sil_wrw_mem((uint32_t *)(hi2c->base+TOFF_I2C_DR), (uint32_t)(*hi2c->pBuffPtr++));
   hi2c->XferCount--;

   if(hi2c->XferCount == 0){ /* 停止 */
      sil_andw_mem((uint32_t *)(hi2c->base+TOFF_I2C_CR2), I2C_CR2_ITBUFEN);
   }
}
```

送信エンプティ割込み

1バイトづつデータ送信

```
static void
i2c_mastertrans_BTF(I2C_Handle_t *hi2c)
{
   if(hi2c->XferCount != 0){  /* データ送信 */
      sil_wrw_mem((uint32_t *)(hi2c->base+TOFF_I2C_DR), (uint32_t)(*hi2c->pBuffPtr++));
      hi2c->XferCount--;
   }
   else{              /* 送信終了 */
       sil_andw_mem((uint32_t *)(hi2c->base+TOFF_I2C_CR2),
                    (I2C_CR2_ITEVTEN | I2C_CR2_ITBUFEN | I2C_CR2_ITERREN));
       sil_orw_mem((uint32_t *)(hi2c->base+TOFF_I2C_CR1), I2C_CR1_STOP);
       hi2c->status = I2C_STATUS_READY;
       if(hi2c->writecallback != NULL)
          hi2c->writecallback(hi2c);
   }
}
```

転送終了割込み

データ送信フェールセーフ

データ送信要求リセットストップ条件設定

1.シリアルデバイスI/F
2.I2C仕様(v2.1)
3.開発環境構築（ハード）
4.開発環境構築（ソフト）
5.デバイス・ドライバ内容紹介
6.各デバイス通信ソフトウェア

6.各デバイス通信ソフトウェア

デバイス通信プロトコル

- I2Cは制御やデータの通信路であり、デバイス毎に通信手順やデータは異なる
- ここでは、IoT関連ということで、温度センサー：ADT7410との通信について解説する
- ADT7410は14個のレジスタをもち、I2C通信を用いてレジスタアクセスが可能である

レジスタへの書込みは、1バイトアドレス指定I2C送信を使う
レジスタの読込みは、1バイトアドレス指定I2C受信を使う

Table 6. ADT7410 Registers

Register Address	Description	Power-On Default
0x00	Temperature value most significant byte	0x00
0x01	Temperature value least significant byte	0x00
0x02	Status	0x00
0x03	Configuration	0x00
0x04	T_{HIGH} setpoint most significant byte	0x20 (64°C)
0x05	T_{HIGH} setpoint least significant byte	0x00 (64°C)
0x06	T_{LOW} setpoint most significant byte	0x05 (10°C)
0x07	T_{LOW} setpoint least significant byte	0x00 (10°C)
0x08	T_{CRIT} setpoint most significant byte	0x49 (147°C)
0x09	T_{CRIT} setpoint least significant byte	0x80 (147°C)
0x0A	T_{HYST} setpoint	0x05 (5°C)
0x0B	ID	0xCX
0x2F	Software reset	0xXX

ADT7410レジスタアクセス

- ADT7410のレジスタはEEPROMと同様に1バイトのアドレス指定、READ/WRITEでアクセス可能である
- 最低限のアクセスとして、レジスタ2が状態値、レジスタ0，1がハイ、ローの温度値となる

Table 10. Status Register (Register Address 0x02)

Bit	Default Value	Type	Name	Description
[3:0]	0000	R	Unused	Reads back 0.
4	0	R	T_{LOW}	Set this bit to 1 when the temperature goes below the T_{LOW} temperature limit, and if comparator mode is selected through the configuration register, Register Address 0x03[4]. The bit clears to 0 when the status register is read and/or when the temperature measured goes back above the limit set in the setpoint $T_{LOW} + T_{HYST}$ registers.
5	0	R	T_{HIGH}	Set this bit to 1 when the temperature goes above the T_{HIGH} temperature limit, and if comparator mode is selected through the configuration register, Register Address 0x03[4]. The bit clears to 0 when the status register is read and/or when the temperature measured goes back below the limit set in the setpoint $T_{HIGH} - T_{HYST}$ registers.
6	0	R	T_{CRIT}	Set this bit to 1 when the temperature goes above the T_{CRIT} temperature limit, and if comparator mode is selected through the configuration register, Register Address 0x03[4]. This bit clears to 0 when the status register is read and/or when the temperature measured goes back below the limit set in the setpoint $T_{CRIT} - T_{HYST}$ registers.
7	1	R	$\overline{RDY}$	This bit goes low when the temperature conversion result is written into the temperature value register. It is reset to 1 when the temperature value register is read. In one-shot and 1 SPS modes, this bit is reset after a write to the one-shot bits.

ステータスレジスタのビット7がREADYビットとなるため、READYを待って温度を取得

温度値の取得

- 温度はレジスタ0、1から取得：2つのレジスタを合わせた16値のうち、B14-B4までが、0.0625単位の度数となる。小数点1桁の度数を取り出すには

$$\text{小数点1桁の温度} = \text{レジスタ0：1} \times \frac{1}{8} \times \frac{625}{1000} = \frac{1}{12.8} \fallingdotseq \frac{1}{13}$$

Table 8. Temperature Value MSB Register (Register Address 0x00)

Bit	Default Value	Type	Name	Description
[14:8]	0000000	R	Temp	Temperature value in twos complement format
15	0	R	Sign	Sign bit, indicates if the temperature value is negative or positive

Table 9. Temperature Value LSB Register (Register Address 0x01)

Bit	Default Value	Type	Name	Description
0	0	R	T_{LOW} flag/LSB0	Flags a T_{LOW} event if the configuration register, Register Address 0x03[7] = 0 (13-bit resolution), and if comparator mode is selected through the configuration register, Register Address 0x03[4]. When the temperature value is below T_{LOW}, this bit it set to 1. Contains the Least Significant Bit 0 of the 15-bit temperature value if the configuration register, Register Address 0x03[7] = 1 (16-bit resolution).
1	0	R	T_{HIGH} flag/LSB1	Flags a T_{HIGH} event if the configuration register, Register Address 0x03[7] = 0 (13-bit resolution), and if comparator mode is selected through the configuration register, Register Address 0x03[4]. When the temperature value is above T_{HIGH}, this bit it set to 1. Contains the Least Significant Bit 1 of the 15-bit temperature value if the configuration register, Register Address 0x03[7] = 1 (16-bit resolution).
2	0	R	T_{CRIT} flag/LSB2	Flags a T_{CRIT} event if the configuration register, Register Address 0x03[7] = 0 (13-bit resolution), and if comparator mode is selected through the configuration register, Register Address 0x03[4]. When the temperature value exceeds T_{CRIT}, this bit it set to 1. Contains the Least Significant Bit 2 of the 15-bit temperature value if the configuration register, Register Address 0x03[7] = 1 (16-bit resolution).
[7:3]	00000	R	Temp	Temperature value in twos complement format.

温度センサーREADY待ち

```
void main_task(intptr_t exinf)
{
   I2C_Init_t i2c_initd;
   I2C_Handle_t *hi2c;
   ER_UINT         ercd;
   i2c_initd.ClockSpeed      = 100000;
   i2c_initd.DutyCycle       = I2C_DUTYCYCLE_16_9;
   i2c_initd.AddressingMode  = I2C_ADDRESSINGMODE_7BIT;
   i2c_initd.DualAddressMode = I2C_DUALADDRESS_DISABLE;
   i2c_initd.GeneralCallMode = I2C_GENERALCALL_DISABLE;
   i2c_initd.NoStretchMode   = I2C_NOSTRETCH_DISABLE;
   i2c_initd.OwnAddress1     = 0;
   i2c_initd.OwnAddress2     = 0;
   i2c_initd.semid           = I2CTRS_SEM;
   i2c_initd.semlock         = I2CLOC_SEM;
   if((hi2c = i2c_init(I2C_PORTID, &i2c_initd)) == NULL){
      syslog_0(LOG_ERROR, "## I2C ERROR(1) ##");
   }
   do{
      if(i2c_memread(hi2c, (uint16_t)ADT7410_ADDR, 0x02, 1, (uint8_t*)aRxBuffer, 1)!= E_OK){
         syslog_1(LOG_ERROR, "## I2C ERROR(13)[%08x] ##", hi2c->ErrorCode);
         return;
      }
      if(I2C_TransWait(hi2c, 500) != E_OK)
         return;
      temp = aRxBuffer[0];
   }while((temp & 0x80) != 0);
```

I2Cドライバの初期化

レジスタ2からデータ読み取り
I2Cアドレス2指定受信

ADT7410のREADY待ち

温度データ取得

- 温度データを取出し、小数点１桁の温度データへ

```
while(count < 1000){
  if(i2c_memread(hi2c, (uint16_t)ADT7410_ADDR, 0x00, 1, (uint8_t*)aRxBuffer, 1)!= E_OK){
    /* I2C RECEIVE ERROR */
    syslog_1(LOG_ERROR, "## I2C ERROR(14)[%08x] ##", hi2c->ErrorCode);
    return;
  }
  if(I2C_TransWait(hi2c, 500) != E_OK)
    return;
  temp = aRxBuffer[0] << 8;
  if(i2c_memread(hi2c, (uint16_t)ADT7410_ADDR, 0x01, 1, (uint8_t*)aRxBuffer, 1)!= E_OK){
    /* I2C RECEIVE ERROR */
    syslog_1(LOG_ERROR, "## I2C ERROR(15)[%08x] ##", hi2c->ErrorCode);
    return;
  }
  if(I2C_TransWait(hi2c, 500) != E_OK)
    return;
  temp += aRxBuffer[0];
  temp /= 13;
```

レジスタ0を取出し、tempのb15-b8にセット
I2Cアドレス0指定受信

レジスタ1を取出し、tempのb7-b4にセット
I2Cアドレス１指定受信

まとめ

- シリアル通信のデバイスが入手できるようになった
  - 接点が少なく、小規模設計が可能
- I2Cは複数デバイスをバス結合できるので、さらに優位
- クロックが低速なため、DIPでも十分動作
  - PCBを作らなくても、ドライバソフト開発、評価が可能
- TOPPERS BASE PLATFORMのような、ソフトウェアプラットフォームを活用すれば、ソースコードを参考にしながら、デバイスドライバ開発、テスト環境開発が可能

参考文献

- I2Cバス仕様書　バージョン2.1

  Philips Semiconductors
- ATD7410 Data Sheet

  ANALOG DEVICES
- TOPPERS BASE PLATFORM
  リファレンス・マニュアル

  TOPPERSプロジェクト 教育WG
- TOPPERSハードウェア設計セミナーテキスト