

TOPPERS基礎実装セミナー

(GR-SAKURA版:ハードウェア設計)

基礎編:1日目

TOPPERSプロジェクト
教育ワーキング・グループ

2016/9/16

1



本ドキュメントに関して

1. 著作権に関しての表記

<TOPPERS基礎ハードウェア設計セミナー(GR-SAKURA版:基本)1日目>

Copyright (C) 2015-2016 by TOPPERSプロジェクト教育WG

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たいかなる損害から、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、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。

2016/9/16

2



スケジュール

■ 1日目

- | | |
|--------------------|-------|
| 1. はじめに | 0.2時間 |
| 2. 開発環境の説明 | 0.5時間 |
| 3. デバッガとLEDの点灯確認 | 1.5時間 |
| 昼休憩 | 1.0時間 |
| 4. ブザーとキーを動かそう | 1.0時間 |
| 5. LCDを動かそう | 1.0時間 |
| 休憩 | 0.5時間 |
| 6. カップラーメンタイマソフト説明 | 1.0時間 |
| 7. まとめ | 0.1時間 |

2016/9/16

3



はじめに

2016/9/16

4



ハードウェア知識の必要性

- 近年、ハードウェアロジックのSoC化、FPGAを使ったロジック設計により、ボード上に複雑な回路設計を行う設計が減ってきている
- 多機能なワンチップマイコンと中低速のシリアルインターフェイスで、いろいろな周辺モジュールを制御することができるようになった、しかも、ブレッドボード上の実装でも十分な品質で通信が可能となった
- 変わりに、SoCの複雑なペリフェラル制御をソフトウェア制御する要求が増えてきている
- Arduinoの普及でホビーとしての組込みが認知されてきた、また、Arduinoコネクタに新規の技術を乗せた種々のシールドが販売され、簡単に新しいハードウェアを試せるようになった

2016/9/16

5



ハードウェア知識の必要性

- 逆に、Arduinoシールドに対応したエバレーション・ボードが多く出されている
- Arduino IDE以外で、シールドを制御するためにハードウェア制御よりの知識が必要となる
- 回路設計者はドライバ等のソフトウェア設計技術が要求され、ソフトウェア設計者も品質の高いプログラム実装を行うためにハードウェアペリフェラルの設計知識が要求されている

まず、ブレッドボードを使ってハードウェア設計に
チャレンジしてみよう

2016/9/16

6



開発環境の説明

1. [マイコンボードの概要](#)
2. ハードウェア環境の構築
3. ソフトウェア環境の構築
4. フラッシュメモリへの書き込み

2016/9/16

7



マイコンボードの概要

マイコンボードと搭載プロセッサ及び
USBシリアルケーブルについて説明する

- マイコンボード
 - GR-SAKURA
 - 若松通商
- プロセッサ
 - RX63N (R5F563NBDDFP) 100pin
 - ルネサスエレクトロニクス
- USBシリアルケーブル(3.3V)
 - TTL-232R-3V3 (FTDI)

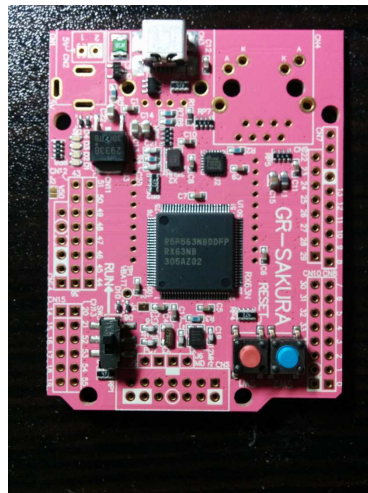
2016/9/16

8



マイコンボード:GR-SAKURA

- 若松通商が販売するマイコンボード
- ルネサスのRX63Nを搭載
- 拡張コネクタの配置はArduinoを意識
 - サービスピンが追加されている
- ハード仕様
 - <http://sakuraboard.net/gr-sakura.html>
- 回路図
 - http://sakuraboard.net/gr_sakura_schematic_color.pdf
- 販売
 - 若松通商
 - RSコンポーネンツ
 - aitando
 - コネクタの実装が異なる



2016/9/16

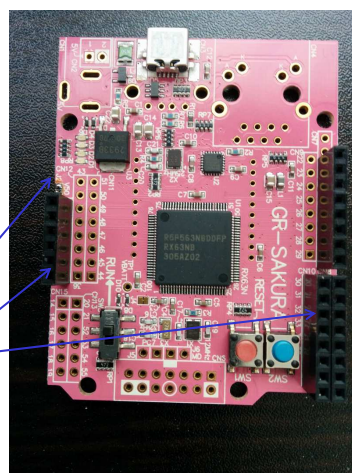
9



マイコンボード:GR-SAKURA

- GR-SAKURAは以下の追加工を行っている
 - USB電源を使用するためJ2を半田ジャンパー
 - USBシリアルケーブル、ブレッドボードと接続するためピンソケットを半田付け

J2
ピンソケット



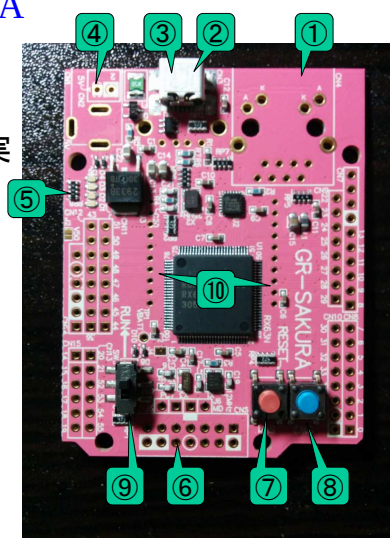
2016/9/16

10



マイコンボード:GR-SAKURA

1. イーサネットコネクタ(未実装)
2. USBターゲット接続コネクタ
3. USBホスト接続コネクタ(裏面未実装)
4. DC電源ジャック(未実装)5
5. LED×4個(D1～4)
6. E1接続コネクタ(パターンのみ)
7. リセットスイッチ(SW1)
8. ユーザースイッチ(SW2)
9. BOOTスイッチ(SW3)
10. Xbee(パターンのみ)



2016/9/16

11



プロセッサ(RX63N):概要

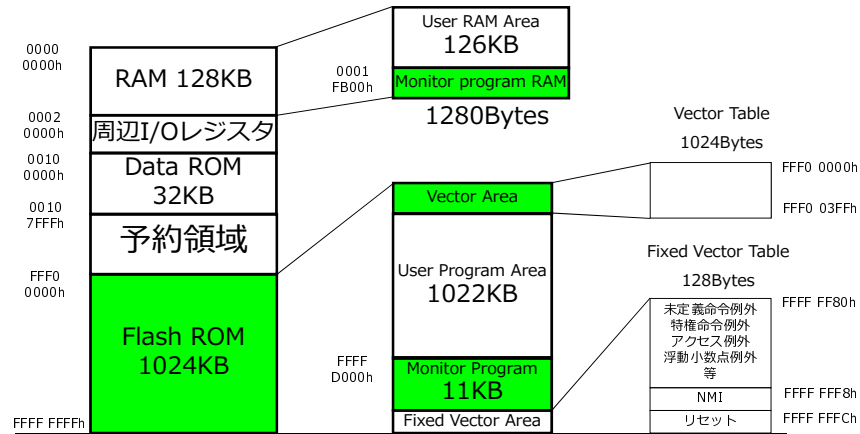
- ルネサスエレクトロニクス製32ビットマイクロコントローラ
- RX63N (R5F563NBDDFP) 100ピン
 - ROM : 1MB + 32KB RAM : 128KB
 - RX630シリーズのイーサネットコントローラ
- 周辺回路
 - 入出力ポート : 79本
 - タイマ : 16ビット×6チャンネル
8ビット×2チャンネル×2ユニット etc...
 - シリアルI/O : 9チャンネル
 - A/D : 10ビット×8チャンネル、12ビット×14チャンネル
 - CAN : 2チャンネル
 - DMAC : 4チャンネル

2016/9/16

12



プロセッサ(RX63N):メモリマップ



- ・ 周辺I/Oレジスタ: 周辺回路の制御レジスタ
- ・ モニタプログラム(後述)が使用するメモリ領域はユーザプログラムで使用しないこと

2016/9/16

13



プロセッサ(RX63N):レジスタ

- ・ 汎用レジスタ(16本)
- ・ 制御レジスタ(9本)
- ・ DSP機能命令関連で使用するアキュムレータ(1本)

ルネサスエレクトロニクス RX63Nグループ, RX631グループ
 ユーザーズマニュアルハードウェア編 Rev.1.00 2012.03 より抜粋

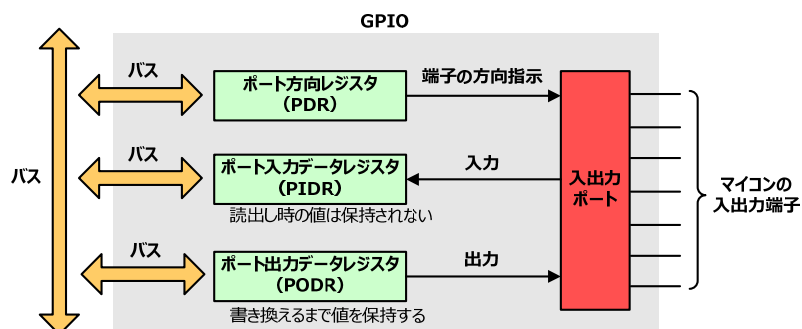


2016/9/16

14



GPIO (General Purpose Input / Output) の説明



- 汎用I/Oポート。LSI端子を介してデジタル信号の入出力を行なう。
- CPUと外部装置でデータの受け渡しをするために、プログラムで扱うデジタル値 (0/1) と信号 (LOW/HIGH) の変換を相互に実施する。

2016/9/16

15



GPIOの説明

- General Purpose Input / Output (汎用入出力)
- LSIの端子 (ポート) を介してデジタル信号の入出力を行なうデバイス

入力ポート

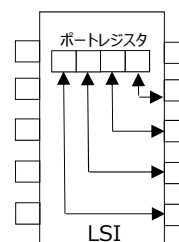
- ポートからの入力が、ポートレジスタに反映される

出力ポート

- ポートレジスタの設定内容が、ポートから出力される

コントロールレジスタ

- 各ポートを出力ポートとして使うか、入力ポートとして使うかなどを設定する
- ビット単位もしくは、あるグループ単位毎に設定可能



2016/9/16

16



開発環境の説明

1. マイコンボードの概要
2. ハードウェア環境の構築
3. ソフトウェア環境の構築
4. フラッシュメモリへの書き込み

2016/9/16

17



ハードウェア環境の構築: 梱包物

- ・ GR-SAKURA以外の必要機材は以下の通り



- ①ジャンパーワイヤー
- ②ジャンパーコード
- ③USBケーブル
- ④圧電ブザー
- ⑤タクトスイッチ
- ⑥LCD(I2C接続)
- ⑦USBシリアル変換機
- ⑧ブレッドボード
- ⑨抵抗、コンデンサ

2016/9/16

18



開発環境の説明

1. マイコンボードの概要
2. ハードウェア環境の構築
3. [ソフトウェア環境の構築](#)
4. フラッシュメモリへの書き込み

ソフトウェア環境の構築

- 統合開発環境
 - CS+ : 統合開発環境
 - RXシリアルデバッガ : リモートデバッガ
 - Renesas Flash Programmer : フラッシュ書込ツール
- ルネサスエレクトロニクスのウェブサイトからインストーラをダウンロードしてインストール
- 実習では無償評価版を使用
 - リンクサイズ128Kバイト以内の制限あり
- その他のソフトウェア
 - USB-シリアル変換ドライバ(FTDI)
 - テキストエディタ
 - PDFリーダー(Acrobat Reader等)

CS+のインストール

- インストーラを入手
 - http://japan.renesas.com/products/tools/ide/cubesuite_plus/downloads.jsp
 - 【無償評価版】統合開発環境 CS+ for CC Vx.xx.xx (一括ダウンロード版)をクリック
 - バージョンは時期によって異なる
 - CSPlus_CC_Package_Vxxxxxxx.exe をダウンロード
 - MyRenesas にログイン(登録が必要)
 - ダウンロードには同意が必要
 - ダウンロードしたファイルを実行
 - ファイルが解凍される

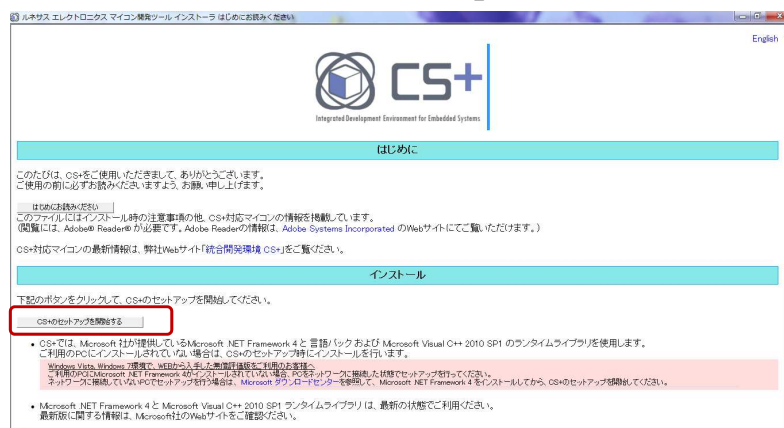
2016/9/16

21



CS+のインストール

- 「はじめに」の画面が開く(以下の説明はv3.01.00で説明)
 - 「CS+のセットアップを開始する」をクリック



2016/9/16

22



CS+のインストール

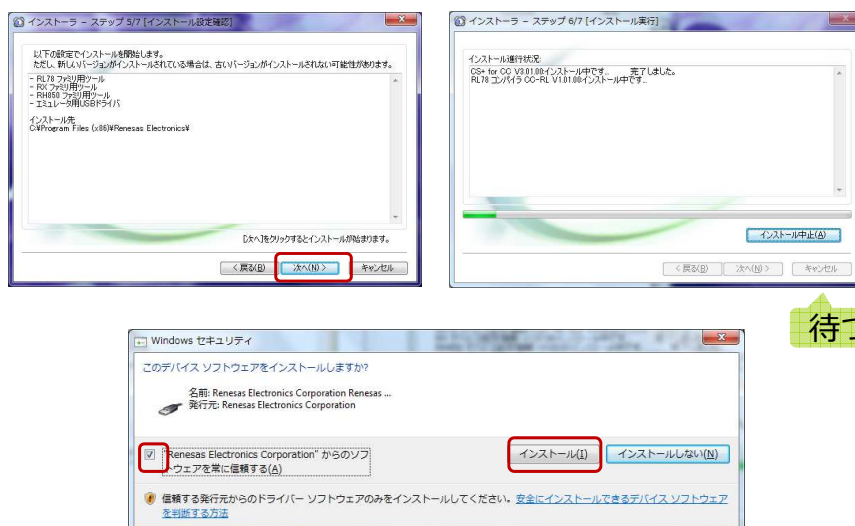


2016/9/16

23



CS+のインストール



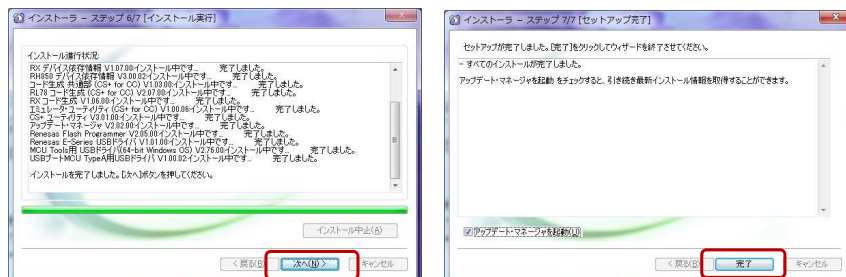
待つ

2016/9/16

24



CS+のインストール



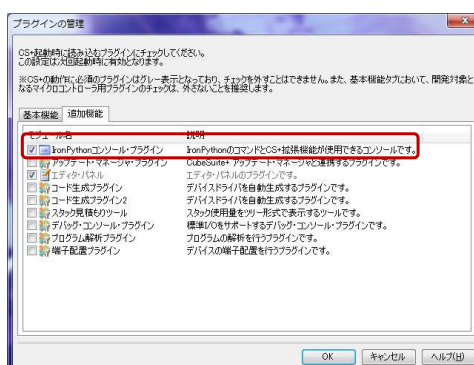
2016/9/16

25



CS+のインストール

- スタートメニューからCS+を起動する
- 「ツール」→「プラグインの管理」→「追加機能」タブを開く
- IronPythonコンソール・プラグイン」にチェックが入っている事を確認
- OKをクリック



2016/9/16

26



RXシリアルデバッガのインストール

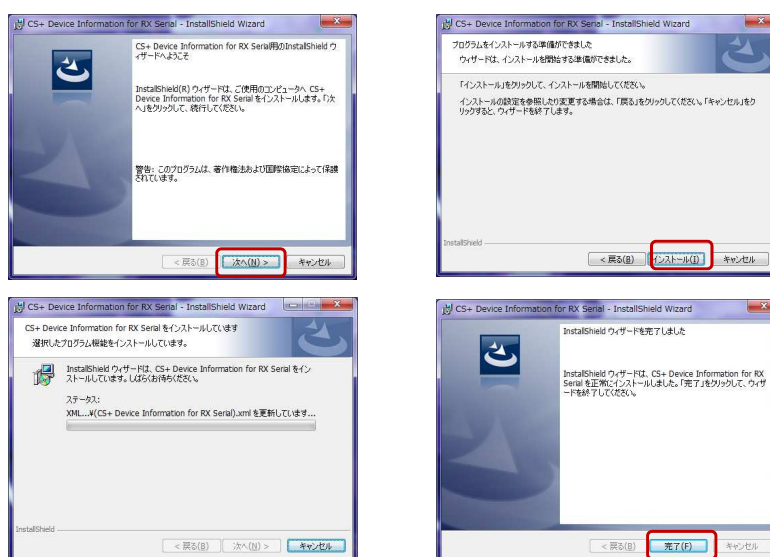
- ・ インストーラを入手
 - ・ http://japan.renesas.com/products/tools/emulation_debugging/monitor_debuggers/serial_debugger/downloads.jsp
 - ・ 「RXシリアルデバッガ」をクリック
 - ・ 「RX_Serial_Debugger_Vxxxx.zip」をダウンロード
 - ・ ダウンロードには同意が必要
 - ・ MyRenesasにログイン
 - ・ ダウンロードしたファイルを右クリックメニューで「すべて展開...」
 - ・ 展開されたフォルダにあるインストーラでインストール
「CP_DevInfo_RX_Serial_Vxxxxxx.exe」
 - ・ バージョンはCS++のバージョンに合わせる

2016/9/16

27



RXシリアルデバッガのインストール



2016/9/16

28



RXシリアルデバッガのインストール

- CS+を用いてシリアル経由でデバッグを行うためには、ボードにモニタプログラムを書き込んでおく必要がある
- 教材のGR-SAKURA/RX63Nは書込み済み
 - 電源ONでGR-SAKURAのD1, D2が点灯
- 書込み手順は「4.フラッシュメモリの書込み」を参照

2016/9/16

29



USB-シリアル変換ドライバのインストール

- 実習ではUSBシリアル変換ケーブル用いてPCと接続する(GR-SAKURAはデバッグを行わない場合接続)
 - 5V電源の供給
 - RXシリアルデバッガの通信
- USBシリアル変換ケーブルはFTDI社製のUSBシリアル変換機を使用しているため必要なドライバをインストールする

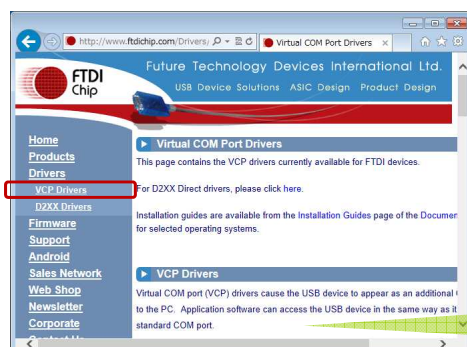
2016/9/16

30



USB-シリアル変換ドライバのインストール

- ドライバのダウンロード
 - FTDIのホームページ(<http://www.ftdichip.com/>)から、使用するホストPCのOSに合わせたVCPドライバをダウンロードする



下方にある一覧表から選択

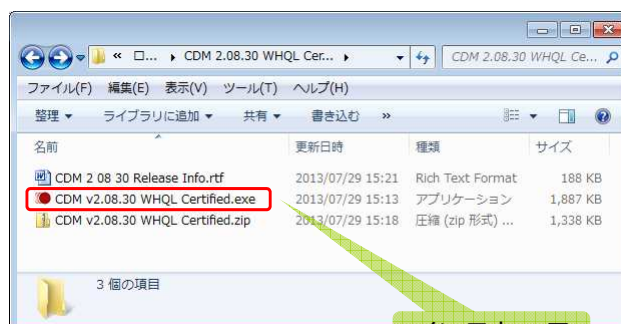
2016/9/16

31



USB-シリアル変換ドライバのインストール

- ダウンロードした圧縮ファイルを展開する
 - フォルダやファイル名はバージョンによって異なる
- USBシリアル変換ケーブルを抜いてインストーラをダブルクリック



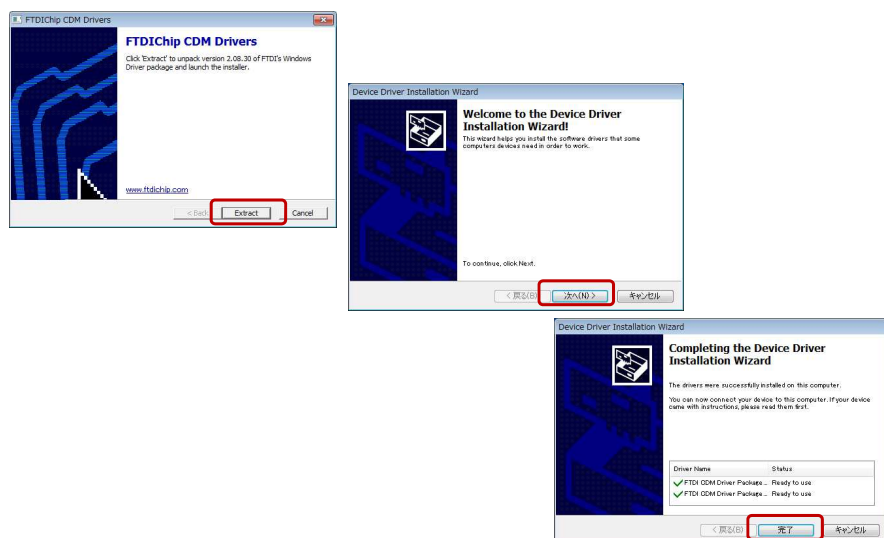
インストーラ

2016/9/16

32



USB-シリアル変換ドライバのインストール



2016/9/16

33



USB-シリアル変換ドライバのインストール

- COMポートの確認
 - “コンピューター”のメニューから“コントロールパネル”を開き“デバイスマネージャー”を起動する



2016/9/16

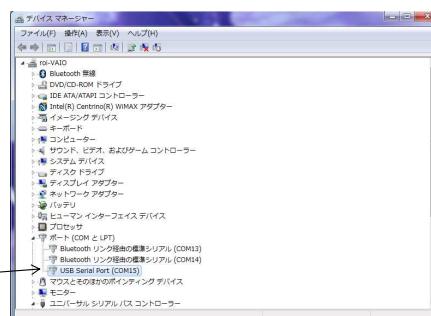
34



USB-シリアル変換ドライバのインストール

- “ポート(COMとLPT)”を開く
 - USBを接続
 - PCから認識音
 - USB Serial Port の項目にCOMポートが出現する
 - 番号は環境によって異なる
 - 番号をメモしておく
 - 後述のデバッガの設定が必要

COM番号



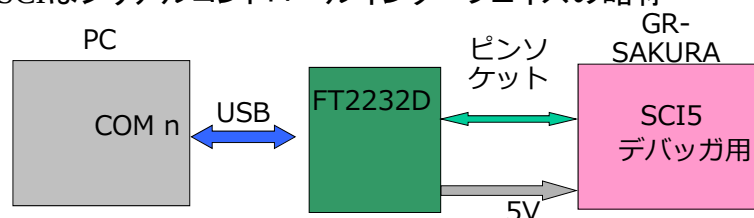
2016/9/16

35



USB-シリアル変換ドライバのインストール

- USBシリアル変換ケーブルをPCと接続するとGR-SAKURAに5V電源が供給される
- RX63NのUART(SCI5)もUSBシリアル変換ケーブルを介してPCと接続される
- RXシリアルデバッガ用モニタプログラムはSCI5を使用するようにカスタマイズしてある
- SCIはシリアルコントロールインターフェ이스の略称



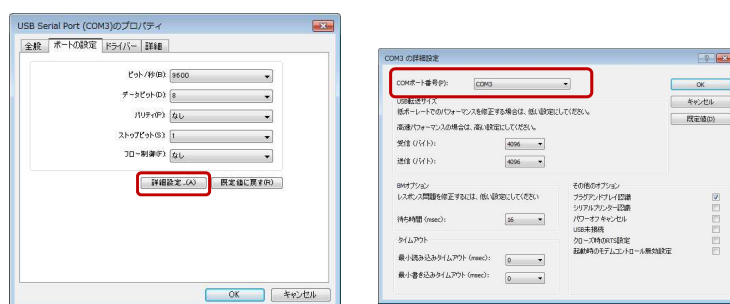
2016/9/16

36



USB-シリアル変換ドライバのインストール

- 割当てCOMポートの変更が必要な場合
 - USB Serial Port の項目で右クリックし, “プロパティ”を選択する
 - “ポートの設定” のタブを選択し, “詳細設定”をクリックする
 - “COMポート番号”で割り当てるCOMポートの変更が可能



2016/9/16

37



開発環境の説明

1. マイコンボードの概要
2. ハードウェア環境の構築
3. ソフトウェア環境の構築
4. フラッシュメモリへの書き込み

2016/9/16

38



フラッシュメモリへの書き込み

プログラムのフラッシュメモリへの書き込み方法を解説

- 教材のGR-SAKURA/RX63Nマイコンのフラッシュメモリには、サンプルプログラムが書き込まれて出荷されており、電源を入れるとサンプルプログラムが動作する
- モニタプログラムはPCで稼動するデバッガと通信を行い、ユーザープログラムのダウンロードと実行を実現する
 - ユーザープログラムもフラッシュメモリに書き込まれるが、デバッグを行う場合、デバッグモニタをフラッシュメモリに書き込んでおく必要がある、デバッグモニタで起動するとGR-SAKURAのD1とD2が点灯
- デバッガを介さずユーザープログラムを実行させるにはデバッガを消してユーザープログラムを直接フラッシュメモリに書き込む必要がある

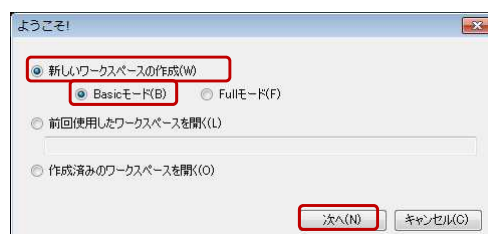
2016/9/16

39



フラッシュメモリへの書き込み

- スタート→Renesas Electronics Utilities→書き込みツール
→Renesas Flash Programmer を起動
 - CS+と一緒にインストールされている
- “新しいワークスペースの作成”, “Basicモード”
- 次へ



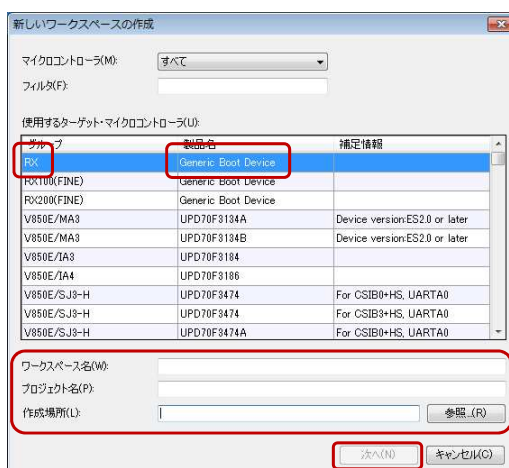
2016/9/16

40



フラッシュメモリへの書き込み

- “RX”, “Generic Boot Device” を選択
- 作成場所, ワークスペース名, プロジェクト名を指定
- 次へ



2016/9/16

41



フラッシュメモリへの書き込み

- “使用ツール”は“USB Direct”を選択
- 次へ



- GR-SAKURAの準備
 - シールドとGR-SAKURAを分離
 - GR-SAKURAのモードスイッチ (SW3)を下側(RUNの逆)に切り替え



2016/9/16

42

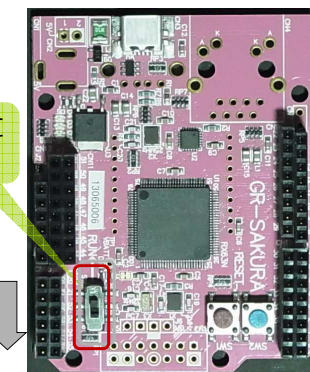


フラッシュメモリへの書き込み

- GR-SAKURAのUSBとPCを接続



USBケーブル



- OKをクリック



下側に
する

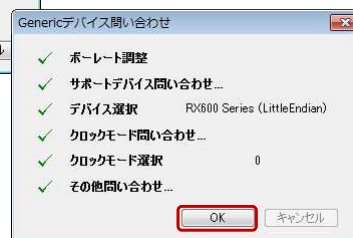
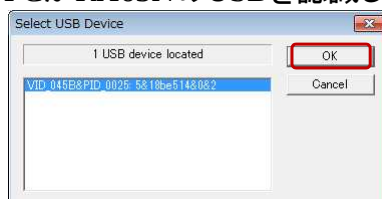
2016/9/16

43



フラッシュメモリへの書き込み

- PCがRX63NのUSBを認識したらOK

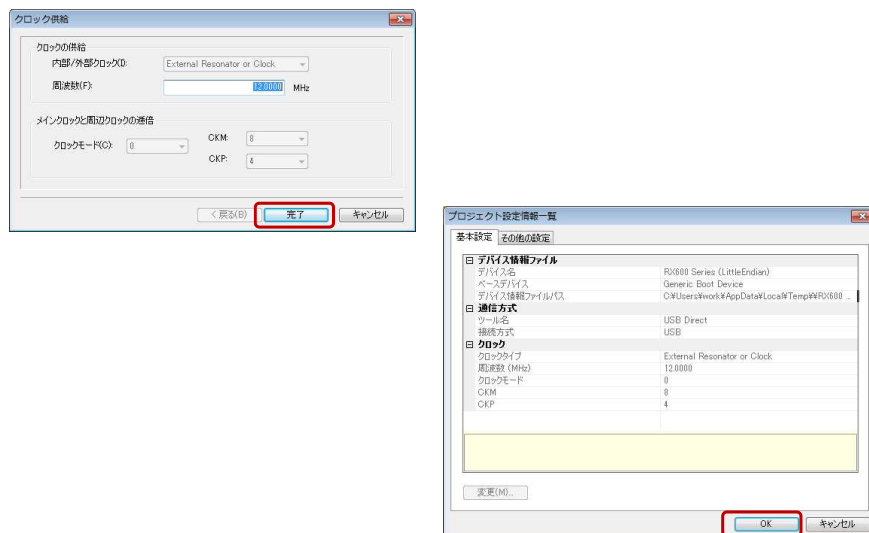


2016/9/16

44



フラッシュメモリへの書き込み: デバugg



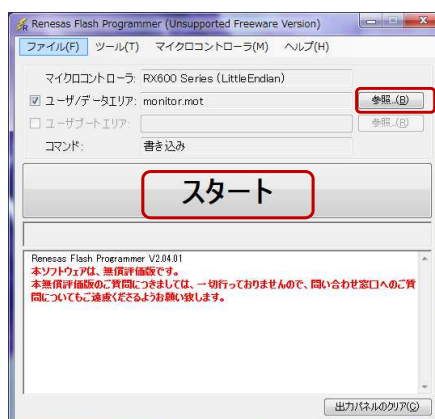
2016/9/16

45



フラッシュメモリへの書き込み

- 「参照」をクリックしてファイルを指定
 - program¥Serial_Debugger¥RX63X_sci5¥LoadModule¥monitor.mot
- 書き込みスタート



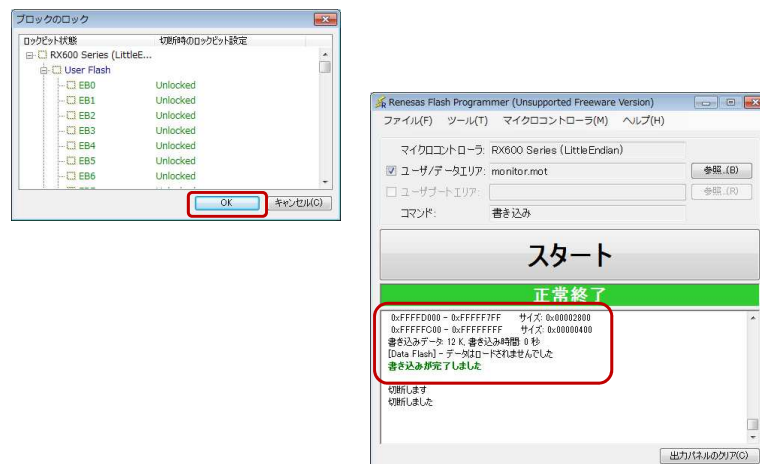
2016/9/16

46



フラッシュメモリへの書き込み

- 書き込み成功



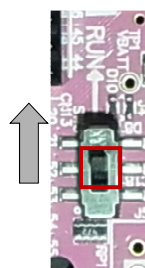
2016/9/16

47



フラッシュメモリへの書き込み: デバッグ

- USBケーブルを抜く
- GR-SAKURAのSW3をRUN(上側)にする



2016/9/16

48



デバuggとLEDの点灯確認

1. シリアルデバugg
2. 開発環境のビルドと実行
3. LED点灯プログラムの確認

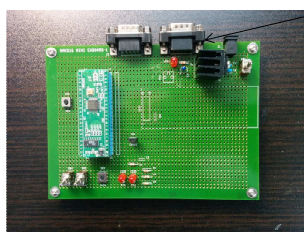
2016/9/16

49

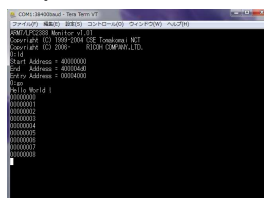


シリアルデバugg:RS232C

- 組み込みデバuggの基本的な手法
 - シリアル通信デバugg
 - ICEによるデバugg
- 10年前は安価なボードでもRS232Cコネクタをもち、モニタの実装やログ出力でデバuggを行ってきた



RS232Cポート



M16Cボード(5000円程度)デバugg用にRS232Cポートを持っていた

RS232Cを使ったログ出力

2016/9/16

50



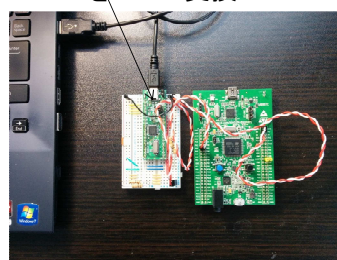
シリアルデバッガ: USB-シリアル出力

- パソコンからRS232Cポートがなくなり、USBシリアル変換機にてRS232Cポートに対応した
- ボード上、または、拡張ボードにUSBシリアル変換LSIを乗せシリアル通信手順が温存されようになった

USBシリアル変換機、変換機にはRS232Cポートが付いている



USBシリアル変換ボード、RS232Cポートはなく、直接ターゲットボードのUARTをUSBに変換



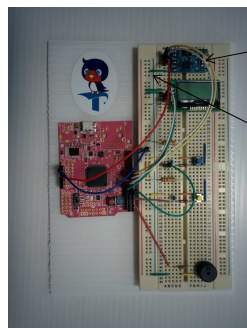
2016/9/16

51



シリアルデバッガ: USBシリアル変換ケーブル

- USBシリアル変換機と、ボード上のUART通信ポートと接続することでシリアルデバッグを可能にする
- ICEを使う場合、デバッグ手順がICEツールに依存した方法しか取れないが、シリアル通信を使用すればユーザーがデバッグ機能を拡張していける



USBシリアル機

ピンストラップ上の信号とケーブル上のピンをピンソケットで接続することでシリアル通信が可能になる

2016/9/16

52



シリアルデバッグ: USBシリアル変換ケーブル

- USBシリアル変換機の裏面のDIPピンに信号名が記載されている、表面はUSBケーブルをつなぐ
- CTS/RTSはRS232Cに対応してフロー制御用の信号であるため、接続を行う必要はない
- 他の信号はGR-SAKURAの適切なピンソケットに接続が必要

TTL-232Rピン	GR-SAKURAコネクタ
GND	GND:ブレッドボードのGND
CTS	
+5V	+5V (J2をショートさせた場合)
TXD	PC3
RXD	PC2
RTS	



表面



裏面に信号名を記載

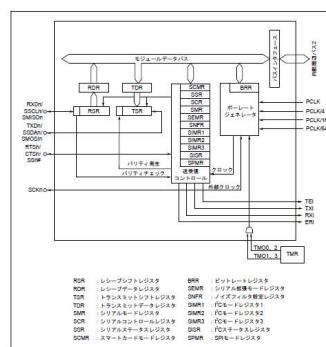
2016/9/16

53



シリアルデバッグ: データブックの確認

- シリアルデバッグとしてシリアル通信を行うためにはRX63NのSCI5の通信ポートと接続する必要がある
- RX63Nのデータブックからシリアルコミュニケーションインターフェイス(SCI) P.1343を参照する
- P.1347のSC5, SC6のブロック図とP.1349の表からTXD5, RXD5がデバッグと通信するシリアル信号であることがわかる



SCI5	SK5	入出力	SCI5のクロック入出力端子
	RXD5	入力	SCI5の受信データ入出力端子
	TXD5	出力	SCI5の送信データ入出力端子
	CTS5#/RTS5#	入出力	SCI5送受信開始制御用入出力端子
SCI6	SK6	入出力	SCI6のクロック入出力端子
	RXD6	入力	SCI6の受信データ入出力端子
	TXD6	出力	SCI6の送信データ入出力端子
	CTS6#/RTS6#	入出力	SCI6送受信開始制御用入出力端子

2016/9/16

54



シリアルデバッガ: データブックの確認

- データブックのP.119～P.123に100ピンLQFPの機能別端子一覧表が掲載されている
- TXD5/RXD5を探すとP.121の49(PC3)ピンがTXD5に、50(PC2)ピンがRXD5信号に接続していることがわかる

RX63N グループ, RX631 グループ

1. 概要

表 1.10 機能別端子一覧 (100ピンLQFP) (3 / 5)

ピン番号 100ピン LQFP	電源 クロック システム制御	I/Oポート	バス EXDMAC	タイマ (MTU, TPU, TMR, PPG, RTC, PGE)	通信 (ETHERC, SCI, SCIg, RSPI, RIIC, CAN, IEB, USB)	割り込み	S12AD, AD, DA
45		PC7	A23/CS0#	MTIOCS3A/ MTCCLKB/ TMC02/PO31	TXD5/SMOSI/ SSDA8/MSOA/ ET_COL	IRQ14	
46		PC6	A22/CS1#	MTIOCS3C/ MTCCLKA/ TMC02/PO30	RXD5/SMOSB/ SSCL8/MOSIA/ ET_ETXD3	IRQ13	
47		PC5	A21/CS2#/ WAIT#	MTIOCSB/ MTCCLKD/ TMR12/PO29	SCK3/RSPCKA/ ET_ETXD2		
48		PC4	A20/CS3#	MTIOCS3D/ MTCCLKG/ TMC01/PO25/ POED#	SCK5/CTS8#/ RTS8#/SS8#/ SSLA0/ET_TX_CLK		
49		PC3	A19	MTIOC4D/ TCLKB/PO24	TXD5/SMOSI/ SSDA5/ETXD/ ET_TX_ER		
50		PC2	A18	MTIOC4B/ TCLKA/PO21	RXD5/SMOSB/ SSCL8/SLA3/ IERXD/ET_RX_DV		

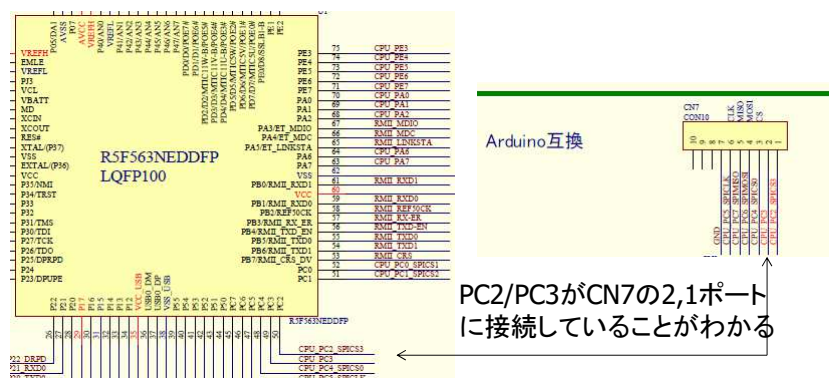
2016/9/16

55



シリアルデバッガ: 回路図の確認

- 回路図で確認する
- PC2がCN7の1番、PC3がCN7の2番に接続している



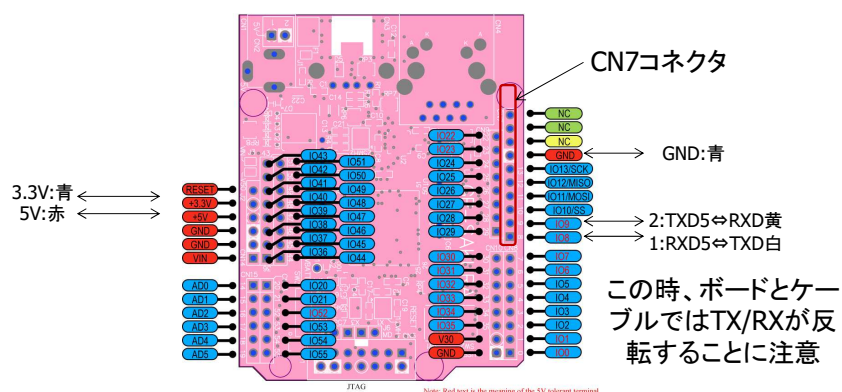
2016/9/16

56



シリアルデバッグ:ピンケーブルの接続

- GR-SAKURAのボード図上の以下の接続を行えばよい
 - 5V:赤 ⇔ 5V GND青 ⇔ GND
 - RXD:黄 ⇔ IO9 TXD白 ⇔ IO8



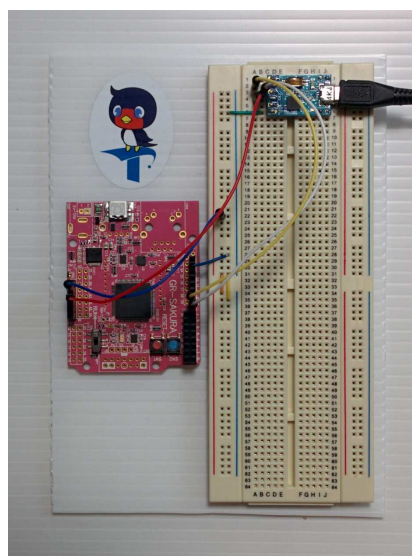
2016/9/16

57



シリアルデバッグ:ピンケーブルで接続

- USBシリアル変換機、USBケーブルとGR-SAKURAのピンコネクタをピンケーブルで接続する



2016/9/16

58



デバuggaとLEDの点灯確認

1. シリアルデバugga
2. [開発環境のビルドと実行](#)
3. LED点灯プログラムの確認

2016/9/16

59



ビルドと実行: 開発環境の起動

開発環境の動作確認のため,
サンプルプログラムをコンパイルする

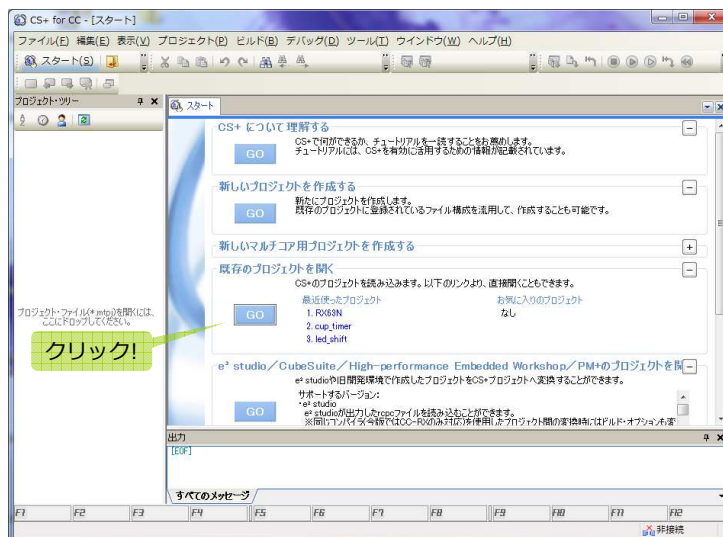
- スタートメニューからCS+を起動
 - デスクトップにショートカットを作成するとよい
 - 起動したら「ヘルプ」→「バージョン情報」を開いてインストールしたバージョンであることを確認

2016/9/16

60



ビルドと実行: 開発環境の起動



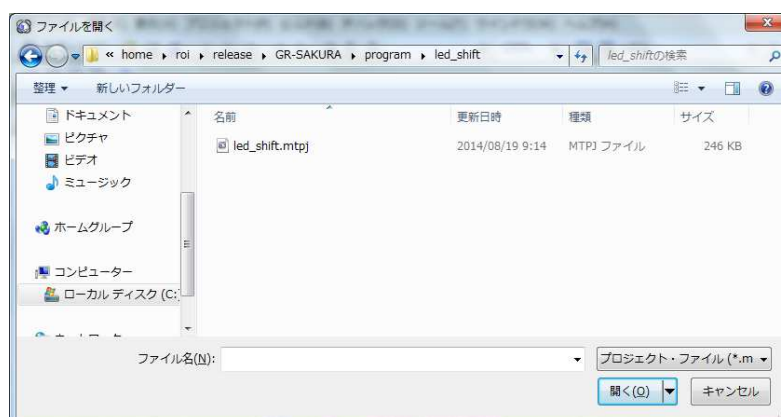
2016/9/16

61



ビルドと実行: 開発環境の起動

- ・ 実習用のプロジェクトファイルを選択
 - program¥led_shift¥led_shift.mtpj

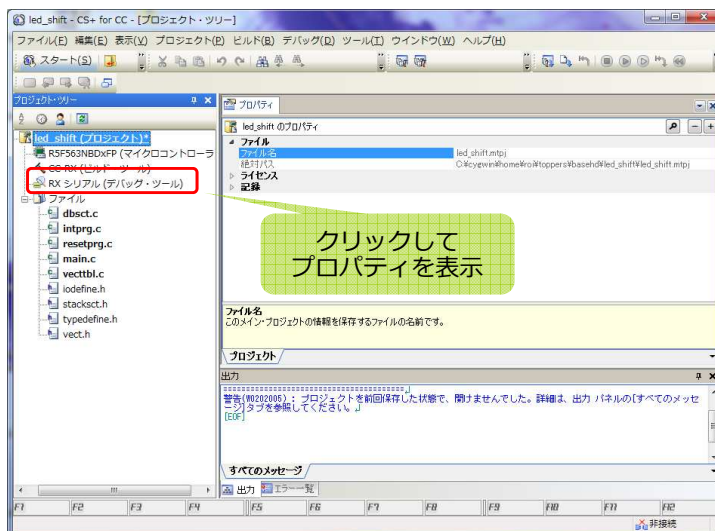


2016/9/16

62



ビルドと実行: 開発環境の起動

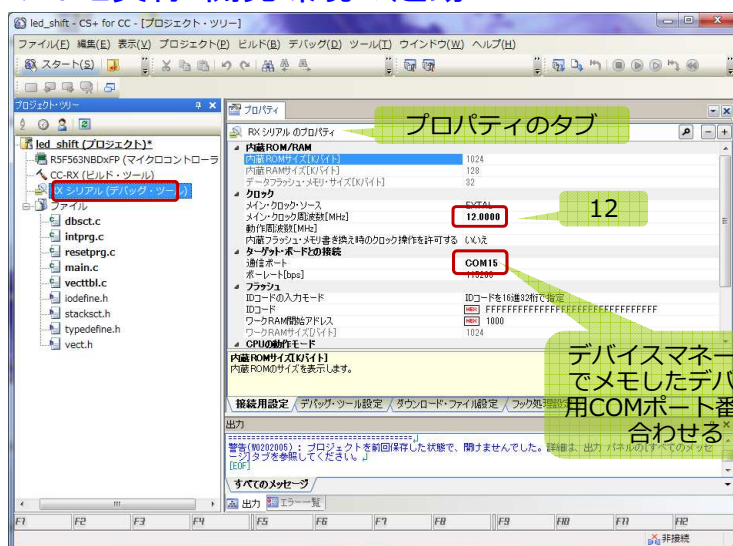


2016/9/16

63



ビルドと実行: 開発環境の起動



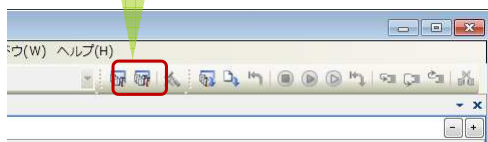
2016/9/16

64



ビルドと実行:リビルド

- 「ビルド」メニューの「リビルド・プロジェクト」を選択
- または **リビルド**



- ビルド成功



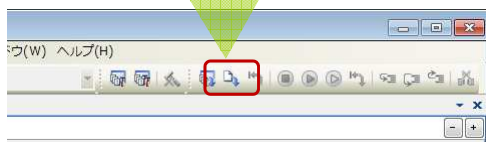
2016/9/16

65



ビルドと実行:ダウンロード

- 「デバッグ」メニューの「デバッグ・ツールへダウンロード」を選択
- または **デバッグツールへプログラムをダウンロード**

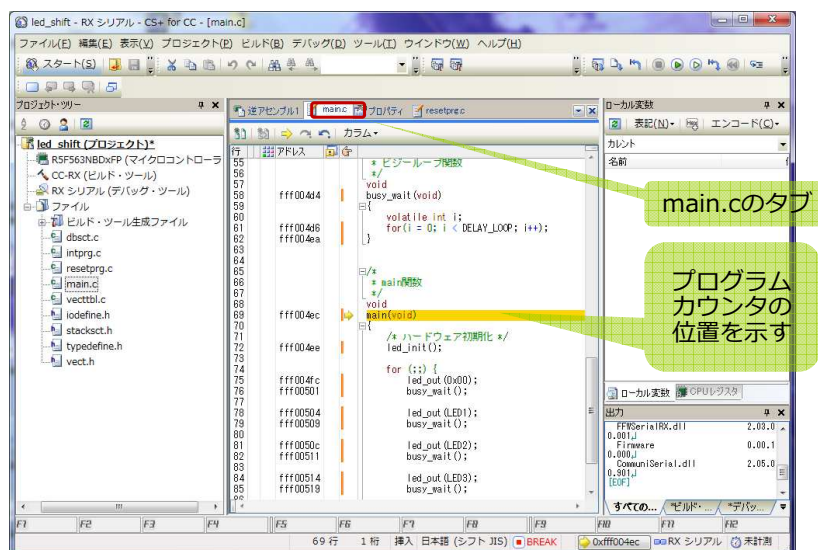


2016/9/16

66



ビルドと実行: シリアルデバグ



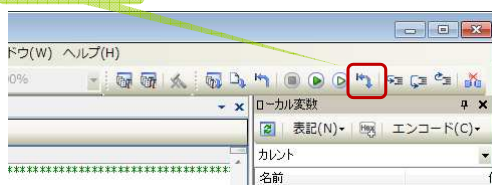
2016/9/16

67

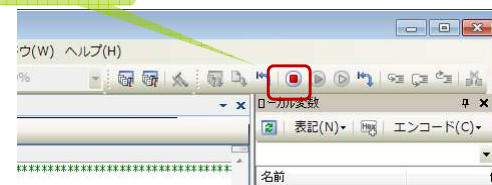


ビルドと実行: 実行と停止

- 「デバッグ」メニューの「リスタート」を選択
- または **クリック**



- 「デバッグ」メニューの「停止」を選択
- または **クリック**



2016/9/16

68



ビルドと実行:デバッグ機能

- デバッグコマンド
 - CPUリセット
 - CPUリセット後実行(リスタート)
 - 停止
 - 実行
 - ステップ・イン実行
 - ステップ・オーバー実行
 - 現在の関数からリターン
 - デバッグ・ツールへ接続
 - ビルド&デバッグ・ツールへダウンロード
 - デバッグ・ツールから切断

2016/9/16

69



デバッガとターゲットが接続できない場合

- デバッガとターゲットの接続が出来ない場合は、以下の事項をチェックし、接続を再実行する
 - シールドの電源LED(POWER)の点灯
 - GR-SAKURAのD1,D2の点灯
 - GR-SAKURAのSW3の設定(RUN側)
 - リセットを押す
 - デバッグ・ツール・プロパティ画面の通信ポートCOM番号がデバ
イスマネージャー画面でメモしたCOM番号と一致しているか
 - GR-SAKURAとシールドを分離して付け直した場合はピンの接
触不良もある←再度付け直す
- GR-SAKURAのD1,D2が点灯しない場合はRX63Nにデ
バッガ用モニタプログラムが書き込まれていない可能性
がある
 - "フラッシュメモリ書込み" に従ってモニタを復帰

2016/9/16

70



デバuggとLEDの点灯確認

1. シリアルデバugg
2. 開発環境のビルドと実行
3. LED点灯プログラムの確認

2016/9/16

71



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
 - LEDデバイスの確認
 - LEDプログラムの確認
 - スタートアップルーチン

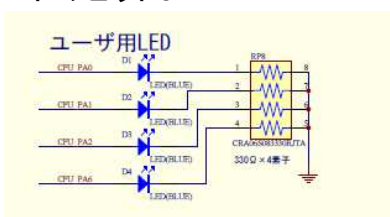
2016/9/16

72



LEDデバイスの確認

- GR-SAKURAの回路図からLEDの接続ポートを確認する
- 「USB回路と周辺コンポーネント」のページからユーザ用LEDを見つけると以下の対応がわかる
 - LED0⇔PA0: GPIO-Aポートのビット0
 - LED1⇔PA1: GPIO-Aポートのビット1
 - LED2⇔PA2: GPIO-Aポートのビット2
 - LED3⇔PA6: GPIO-Aポートのビット6



2016/9/16

73



LEDプログラム:led_shift概要

- | | |
|------------------|----------------------|
| • 構成ファイル | |
| • dbstc.c | セクションの定義 |
| • hwsetup.c | ハードウェア初期化処理 |
| • intprg.c | ベクタ関数 |
| • iodef.h | IOレジスタ定義 |
| • led_shift.mtpj | CubeSuite+プロジェクトファイル |
| • main.c | メイン関数 |
| • resetprg.c | スタートアップルーチン |
| • stacksct.h | スタックサイズ定義 |
| • typedef.h | 変数型定義 |
| • vect.h | ベクタ関数のヘッダ |
| • vecttbl.c | 固定ベクタテーブル |

2016/9/16

74



led_shift: main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

```
void
main(void){
    led_init();
    for (;;) {
        led_out(0x00);
        busy_wait();
        led_out(LED1);
        busy_wait();
        led_out(LED2);
        busy_wait();
        led_out(LED3);
        busy_wait();
        led_out(LED4);
        busy_wait();
    }
}
```

LED全消灯

LED1点灯

LED2点灯

LED3点灯

LED4点灯

2016/9/16

75



led_shift: ポートレジスタ書き込み関数

- LED点灯・消灯制御
 - ポートAデータレジスタへの書き込み
 - ポートのアドレス、型は”iodefine.h”ファイルで定義されているのでそのまま使用

```
/*
 * ポートAのLEDビット
 */
#define LED1      0x01
#define LED2      0x02
#define LED3      0x04
#define LED4      0x40
#define LED_MASK (LED1 | LED2 | LED3 | LED4)
void
led_out(unsigned char led_data)
{
    PORTA.PODR.BYTE = ((PORTA.PODR.BYTE & ~LED_MASK) | (led_data & LED_MASK));
}
```

2016/9/16

76



led_shift: 初期化関数led_init

- ポートを汎用入出力の出力方向に設定し, 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)
{
    /* 方向レジスタ出力設定 */
    PORTA.PDR.BYTE = PORTA.PDR.BYTE | LED_MASK;

    /* 初期状態は消灯とする */
    PORTA.PODR.BYTE = PORTA.PODR.BYTE & ~LED_MASK;

    /* 汎用入出力ポートに設定 */
    PORTA.PMR.BYTE = PORTA.PMR.BYTE & ~LED_MASK;
}
```

6,2,1,0ビット
をセット

6,2,1,0ビット
をクリア

6,2,1,0ビット
をクリア

2016/9/16

77



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forループ1回の実行時間から求める
 - 実際には適当にためして決定する

```
#define DELAY_LOOP 0x800000

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

最適化を抑制するため
volatile修飾子を付ける

2016/9/16

78



led_shift: resetprg.c

- PowerON_Reset
 - リセットで最初に実行される「スタートアップルーチン」
 - CPU制御レジスタの設定
 - 組み込み関数を使用している(machine.hをインクルード)
 - _INITISCTはRAMを初期化するライブラリ関数
 - main関数の呼び出し

2016/9/16

79



led_shift : resetprg.c PowerON_Reset関数

- リセットで最初に実行される「スタートアップルーチン」
- 制御レジスタの設定と変数の初期化を行いmain関数を実行する
- C言語では出来ないCPU制御レジスタの設定を記述可能とする「組み込み関数」を使用している
- set_psw(PSW_init)により割り込み許可フラグをセットしている

```
#include <machine.h>
~中略~
#define PSW_init 0x00010000 // PSW bit pattern
#define FPSW_init 0x00000000 // FPSW bit base pattern
void PowerON_Reset(void)
{
    set_intb(__sectop("C$VECT"));
    set_fpsw(FPSW_init | _ROUND | _DENOM);
    _INITISCT();
    // HardwareSetup(); // Use Hardware Setup
    nop();
    set_psw(PSW_init); // Set Ubit & Ibit for PSW

    main();
    brk();
}
```

FLASHROM実行
の場合はこの関
数の実行が必要

main関数を
呼び出す

2016/9/16

80



led_shift : resetprg.c 組み込み関数の利用

- コンパイラで用意された専用の関数
 - CubeSuite+ヘルプ3.2.6参照
- アセンブラ命令の一部をCの関数で記述できる
- machine.hのインクルードが必要
 - set_intb : INTB(割込みテーブルレジスタ)の設定
 - set_fpsw : FPSW(浮動小数点ステータスワードレジスタ)の設定
 - HardwareSetup : ハードウェアの初期化
 - nop : NOP命令
 - set_psw : PSW(ステータスワードレジスタ)の設定
 - brk : BRK命令例外

2016/9/16

81



led_shift : vecttbl.c 固定ベクタ

- RXはリセット後, 0xFFFFFD0から始まる固定ベクタの0xFFFFF0Cに置かれた番地から実行を開始する
- ベクタテーブルはC言語の配列として作成
- pragma命令によりセクション名を与える
- セクションはリンカのセクション設定にアドレスと共に配置
- リセット後, PowerON_Reset(スタートアップルーチン)にジャンプ

関数
PowerON_Reset
のアドレス

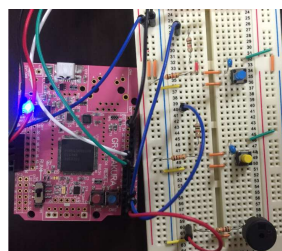
```
#pragma section C FIXEDVECT
void (*const Fixed_Vectors[])(void) = {
    Excep_SuperVisorInst,
    Excep_AccessInst,
    Dummy,
    Excep_UndefinedInst,
    Dummy,
    Excep_FloatingPoint,
    Dummy,
    Dummy,
    Dummy,
    Dummy,
    NonMaskableInterrupt,
    /*(void*)*/ PowerON_Reset_PC
};
```

2016/9/16

82



ブザーとキーを動かそう



2016/9/16

83



ブザー/キープログラム: switch_push概要

- キーを押すと同時に、ブザーを鳴らしてLEDを点灯させる

① キー(SW1)を押下 (ブザーを鳴らしてLEDをカウントアップ)
 ○○○○ → ○○○● → ○○●○ → ○○●● …

② キー(SW2)を押下 (ブザーを鳴らしてLEDをカウントダウン)
 ○○○○ → ●●●● → ●●●○ → ●●○● …

- 学習内容
 - ブレッドボードの実装方法
 - 回路図 (ブザー/キーデバイス) の確認
 - GPIOの確認
 - switch_pushプログラムの確認

2016/9/16

84



ブザーとキーを動かそう

1. [ブレッドボードの実装方法](#)
2. 回路図(ブザー/キー)の確認
3. GPIOの確認
4. switch_pushプログラムの確認

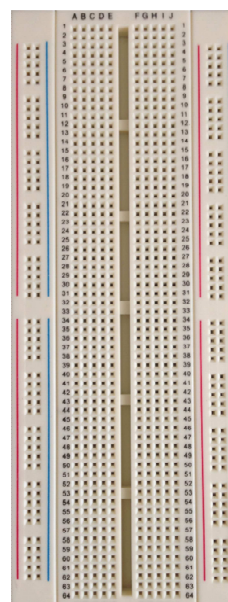
2016/9/16

85



ブレッドボードの説明(1)

- ブレッドボードとは、**半田を使用せずに**各種電子部品やジャンパ線をボードの穴に差し込むだけで電子回路を手軽に実現できる基板



2016/9/16

86

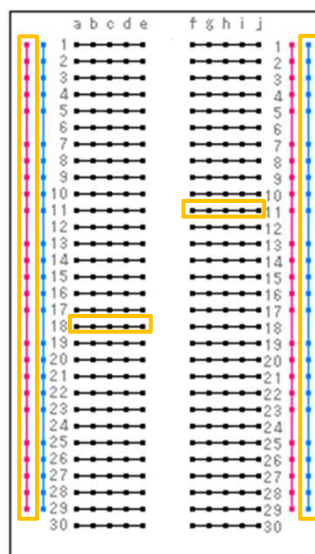


ブレッドボードの説明(2)

それぞれのボードの穴の接続は
予め決まっている
→穴を線で繋いだ部分が導通
※黒は横方向、赤青は縦方向

赤青4本の線は電源として使用
・赤:プラス側 (3,5V...)
・青:マイナス側 (0V,GND)

※本演習では、SAKURAボード側
から電源を取っていきます



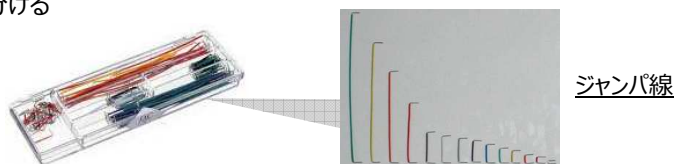
2016/9/16

87

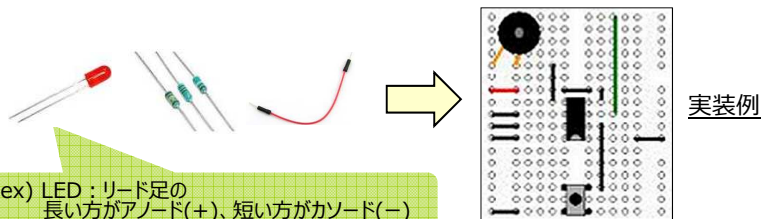


ブレッドボードの説明(3)

- ジャンパ線は長さごとに色が違うものが用意されているため、接続距離によって使い分ける



- 穴同士をジャンパ線や電子部品で繋ぐことで電子回路を実現する
- 電子部品によっては、方向性があるものもあるので注意



ex) LED : リード足の
長い方がアノード(+), 短い方がカソード(-)

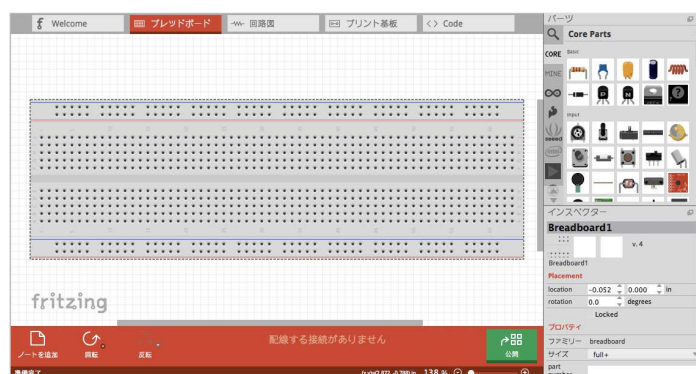
2016/9/16

88



ブレッドボードの説明(4)

- ブレッドボードの回路作図ツール : **fritzing**
<http://www.fritzing.org/>



- 素子やジャンパ線を選択して、自由にブレッドボード上に配置できます
- ドラッグ&ドロップで直感的に操作可能です

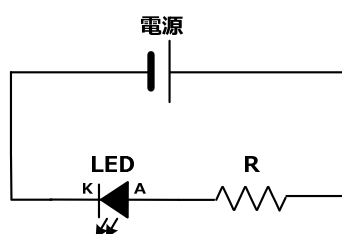
2016/9/16

89



ブレッドボードの説明(5): 例1

- 例1) 電源ONにより、LEDが点灯する電子回路



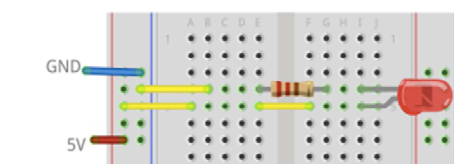
オームの法則
 $V = R \cdot I$

2016/9/16

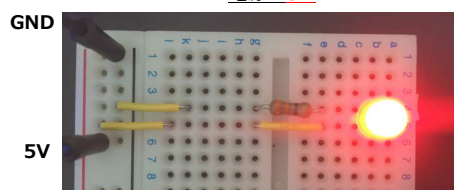
90



ブレッドボードの説明(6):例1 実装例



電源: ON



※ブレッドボードの縦4,5をLEDで接続

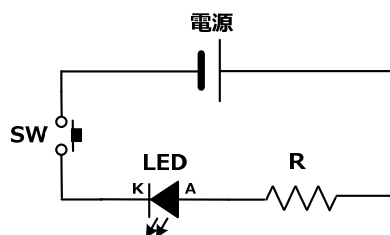
2016/9/16

91



ブレッドボードの説明(7):例2

- 例2) 電源ON後、キー(SW)押下によりLEDが点灯する電子回路

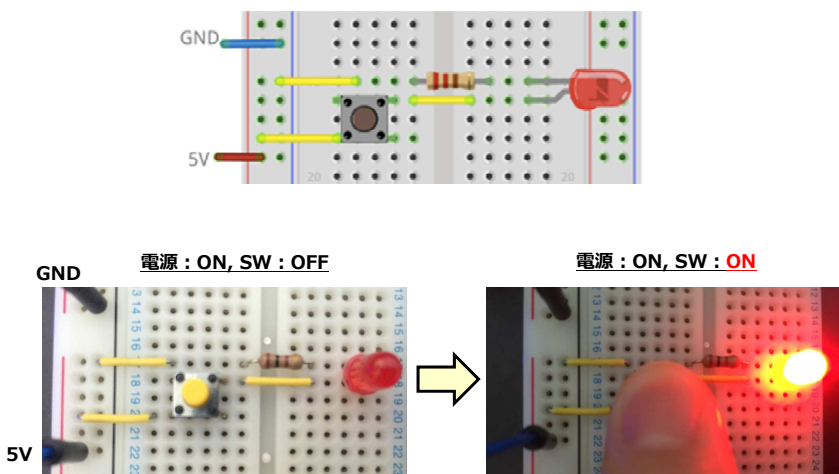


2016/9/16

92



ブレッドボードの説明(8):例2 実装例



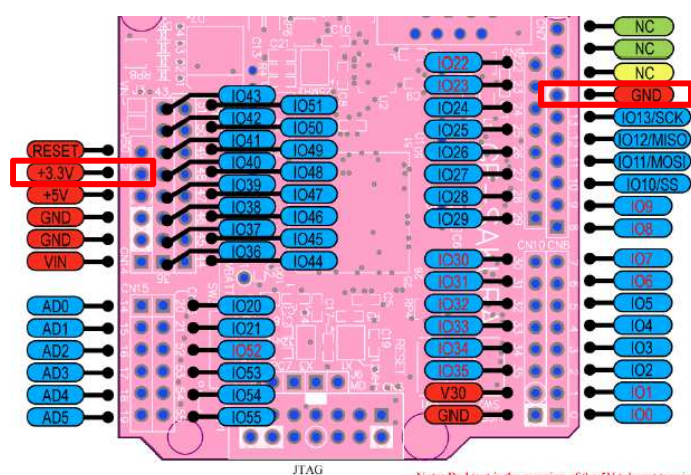
2016/9/16

93



SAKURAボードの電源/GNDの供給について

- GR_Family_AdvanceInformation_Brief_rev002.pdf (P.2) より
下記ピンに接続して、電源 (3.3V) とGNDを供給して下さい。



2016/9/16

94



ブザーとキーを動かそう

1. ブレッドボードの実装方法
2. [回路図\(ブザー/キー\)の確認](#)
3. GPIOの確認
4. switch_pushプログラムの確認

2016/9/16

95

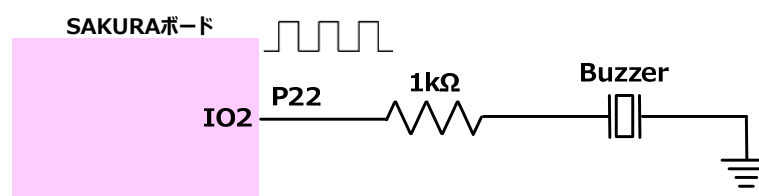


回路図(ブザー)の説明(1)

- 発信器が内蔵された電子部品でパルス（PWMなど）を入れることで音を出す
- パルスの波形を変更することで、音調を変更することが可能



- ブザーの回路図



2016/9/16

96



回路図(ブザー)の説明(2)

回路図を元にブレッドボード上へ実装してみましょう

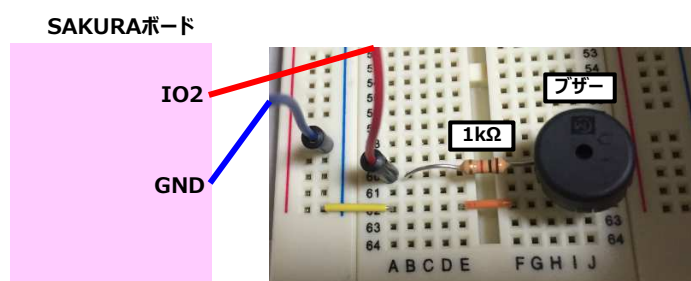
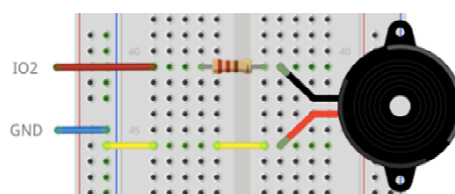
- ・まずは"fritzing"でパーツ配置
- ・次に実際にブレッドボード上へ実装してみましょう

2016/9/16

97



回路図(ブザー)の説明(3):実装例



2016/9/16

98

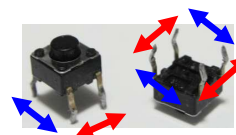


回路図(キー)の説明(1)

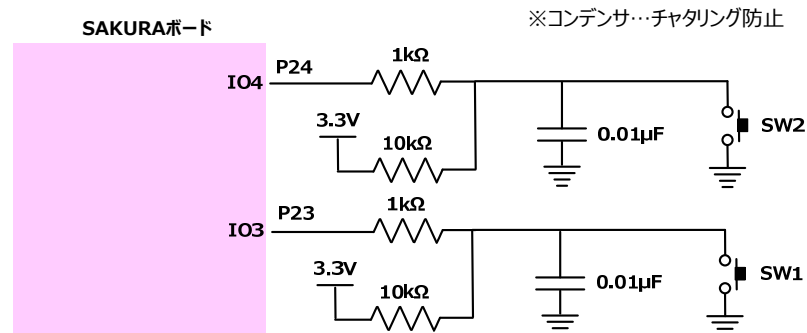
- キーを押すことでON/OFFが切り替わる

※初期状態では青矢印の方向が導通している

※スイッチを押すことで、赤矢印の方向に導通する



- キーの回路図



2016/9/16

99



回路図(キー)の説明(2)

回路図を元にブレッドボード上へ実装してみましょう

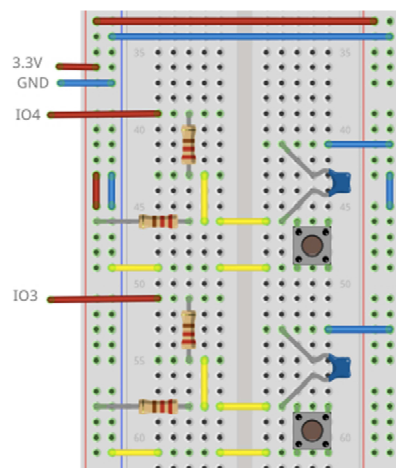
- まずは“fritzing”でパーツ配置
- 次に実際にブレッドボード上に実装してみましょう

2016/9/16

100



回路図(キー)の説明(3):実装例

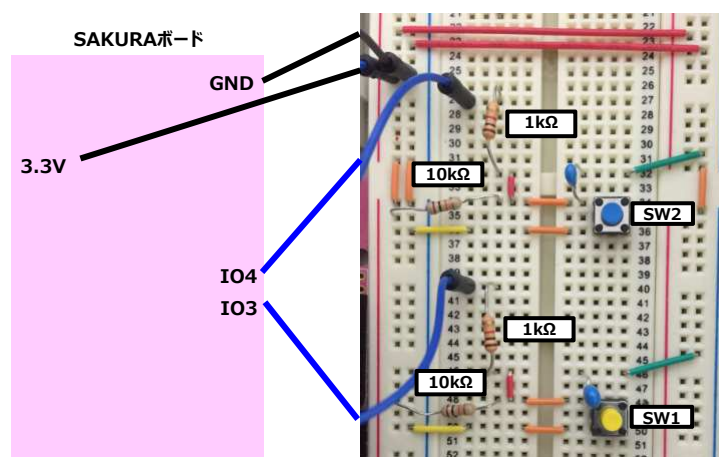


2016/9/16

101



回路図(キー)の説明(4):実装例



2016/9/16

102



ブザーとキーを動かそう

1. ブレッドボードの実装方法
2. 回路図(ブザー/キー)の確認
3. [GPIOの確認](#)
4. switch_pushプログラムの確認

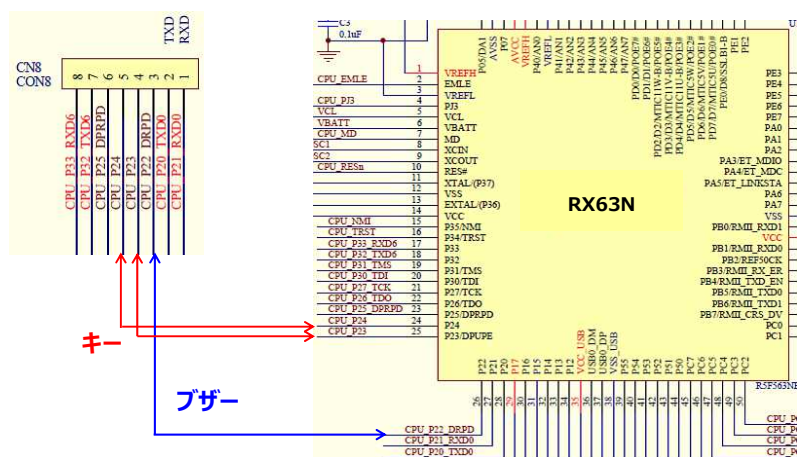
2016/9/16

103



ブザー/キー:GPIOポートの割当て

- ブザー (CN8の3番) ⇔ **P22** : GPIO-2ポートのビット2
- キー (CN8の4,5番) ⇔ **P23,24** : GPIO-2ポートのビット3,4



2016/9/16

104



ブザー/キー: GPIOのプログラム設定

・各デバイスのGPIO設定を実施する

ブザーのGPIO設定

switch_push/device.c

```
/*
 * BUZZER初期化
 */
void buzzer_init(void)
{
    /* 出力ポートに設定 */
    PORT2.PDR.BIT.B2 = 1;

    /* 初期状態はOFFとする */
    PORT2.PODR.BIT.B2 = 1;

    /* 汎用入出力ポートに設定 */
    PORT2.PMR.BIT.B2 = 0;
}
```

※PODRレジスタ (0:Low,1:High)

キーのGPIO設定

switch_push/device.h

```
#define PUSH1    0x08
#define PUSH2    0x10
#define PUSH_MASK (PUSH1 | PUSH2)
```

switch_push/device.c

```
/*
 * PUSHスイッチ接続ポート初期化
 */
void switch_push_init(void)
{
    /* 入力ポートに設定 */
    PORT2.PDR.BYTE = PORT2.PDR.BYTE & ~PUSH_MASK;

    /* 汎用入出力ポートに設定 */
    PORT2.PMR.BYTE = PORT2.PMR.BYTE & ~PUSH_MASK;
}
```

2016/9/16

105



ブザーとキーを動かそう

1. ブレッドボードの実装方法
2. 回路図(ブザー/キー)の確認
3. GPIOの確認
4. [switch_pushプログラムの確認](#)

2016/9/16

106



switch_pushプログラム: 概要

- 構成ファイル
 - dbst.c セクションの定義
 - device.c デバイス (LED/キー/ブザー/タイマ) 操作関数
 - hwsetup.c ハードウェア初期化処理
 - intprg.c ベクタ関数
 - iodefne.h IOレジスタ定義
 - main.c メイン関数
 - resetprg.c スタートアップルーチン
 - stacksct.h スタックサイズ定義
 - switch_push.mtpj CubeSuite+プロジェクトファイル
 - typedefine.h 変数型定義
 - vect.h ベクタ関数のヘッダ
 - vecttbl.c 固定ベクタテーブル

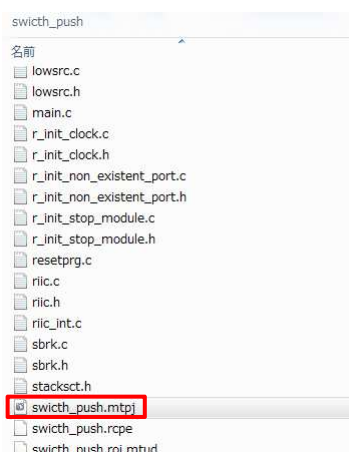
2016/9/16

107



switch_pushプログラム: 開発環境の起動

- 実習用のプロジェクトファイルを選択
 - program¥switch_push¥switch_push.mtpj



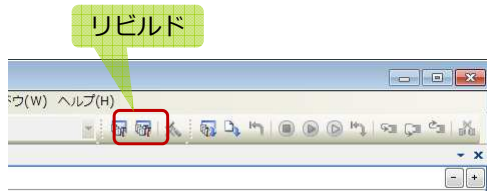
2016/9/16

108

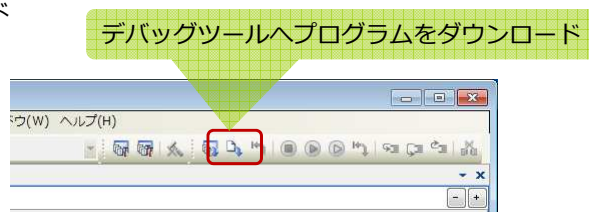


switch_pushプログラム:ビルドと実行

- RXシリアルの「プロパティ」にてデバッグ用COMポートに設定を合わせる
- リビルド



- ダウンロード



※詳細な実行方法はled_shiftの項を参照

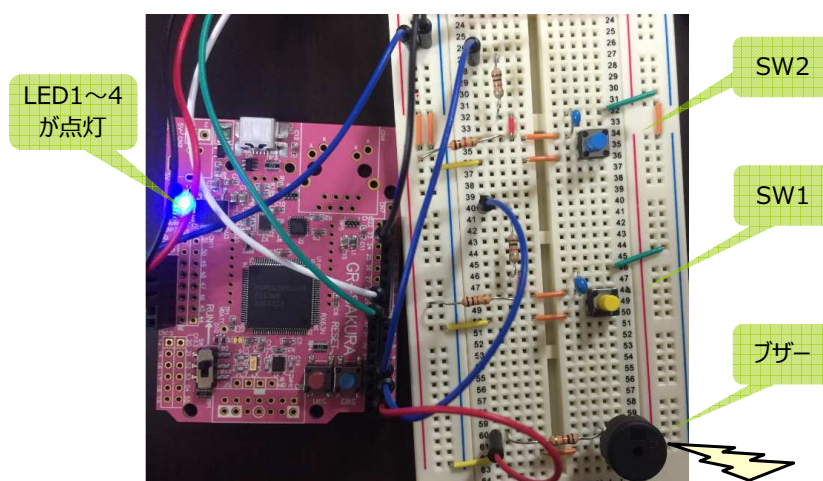
2016/9/16

109



switch_pushプログラム:実行

- キー(SW1/SW2)を押下するごとにブザーが鳴り、LEDが点灯
 ...SW1 : カウントアップ、SW2 : カウントダウン



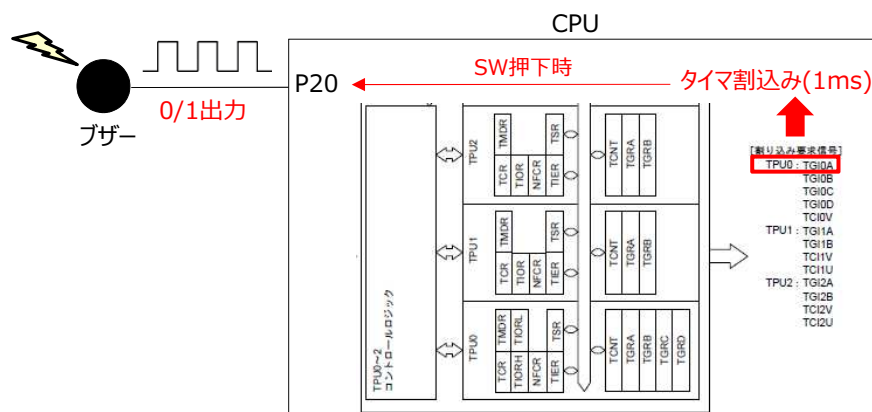
2016/9/16

110



switch_pushプログラム:ブザー制御の実現

- 今回のブザー制御には、**16ビットタイマパルスユニット機能（TPU0）**を使用
- 出力（0/1）を切替えながら、パルスを作り出す
→1msごとにタイマ割込みを発生させて処理する



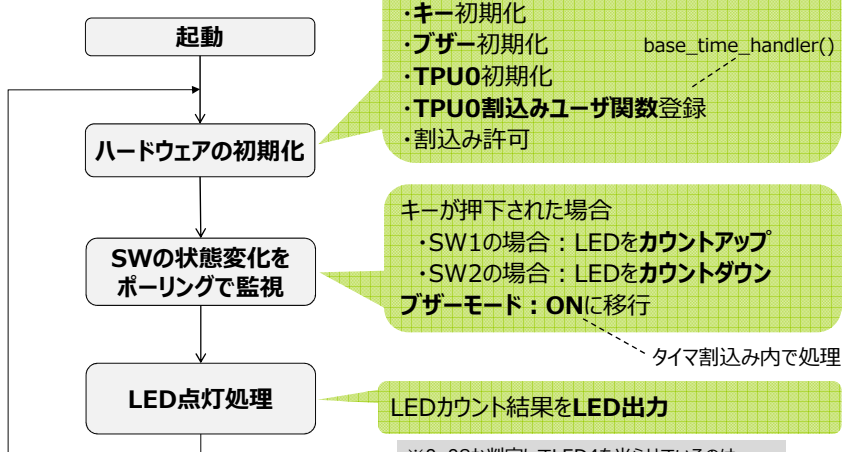
2016/9/16

111



switch_pushプログラムの説明(1):メイン関数

switch push/main.c



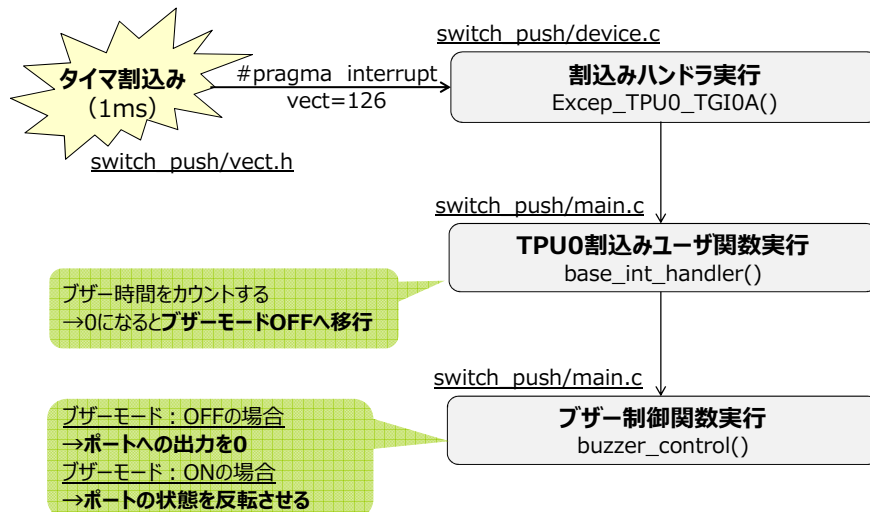
※0x08か判定してLED4を光らせているのは
0x1,0x2,0x4,0x40という仕様になっているため

2016/9/16

112



switch_pushプログラムの説明(2):タイマ割り込み

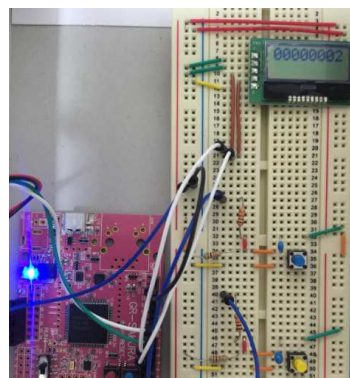


2016/9/16

113



LCDを動かそう



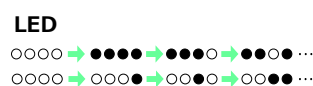
2016/9/16

114



LCDプログラム:lcd_test概要

- プログラム実行後、LCD上に"00000000"が表示される。
 - ①SW1を押下すると、"1"カウントダウンする。LEDも2進表示で点灯する。
 - ②SW2を押下すると、"1"カウントアップする。LEDも2進表示で点灯する。
- 学習内容
 - 回路図（LCD）の確認
 - I2C通信について
 - lcd_testプログラムの確認



2016/9/16

115



LCDを動かそう

- [1. 回路図\(LCD\)の確認](#)
- I2C通信について
- lcd_testプログラムの確認

2016/9/16

116

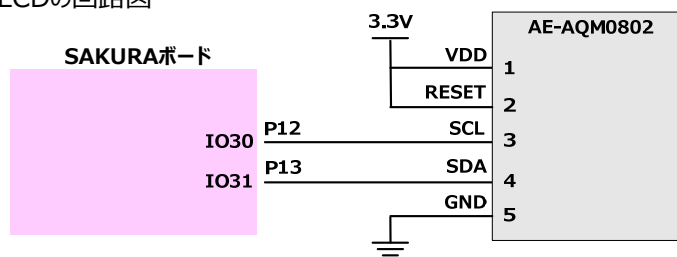


回路図(LCD)の説明(1)

- LCD (Liquid crystal display) とは、画面表示装置である。
- 種類はキャラクタ/グラフィック/タッチパネル式など様々である。



- LCDの回路図



2016/9/16

117



回路図(LCD)の説明(2)

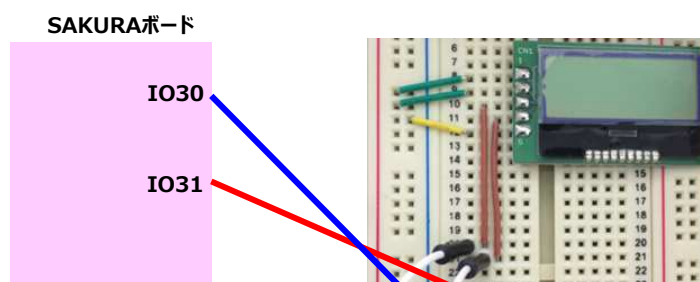
回路図を元にブレッドボード上へ実装してみましょう

2016/9/16

118



回路図(LCD)の説明(3):実装例



2016/9/16

119



LCDを動かそう

1. 回路図(LCD)の確認
2. [I2C通信について](#)
3. lcd_testプログラムの確認

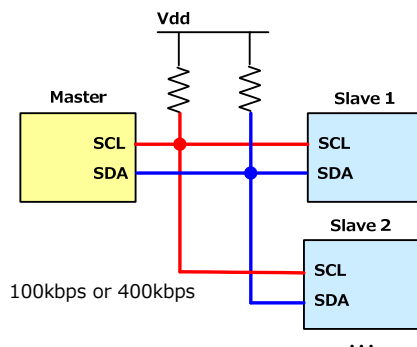
2016/9/16

120



I2C通信(1):概要

- 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

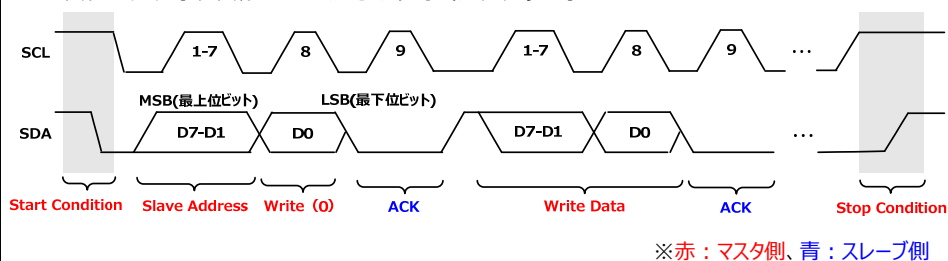
2016/9/16

121



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



- SCLがHighの間にマスタがSDAをLowとするとStart Conditionになる
- マスタはSCLがHighの間にSDAをHighにすることでStop Condition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

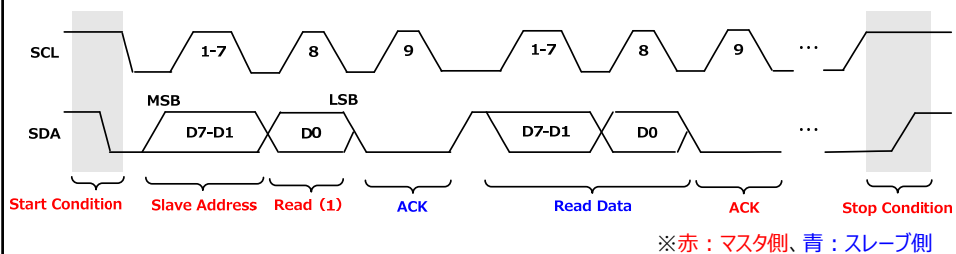
2016/9/16

122



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



- 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

2016/9/16

123



LCDを動かそう

1. 回路図(LCD)の確認
2. I2C通信について
3. [lcd testプログラムの確認](#)

2016/9/16

124



lcd_testプログラム: 概要

- 構成ファイル
 - dbstc.c セクションの定義
 - **device.c** **デバイス（LED/キー/ブザー/LCD）操作関数**
 - hwsetup.c ハードウェア初期化処理
 - intprg.c ベクタ関数
 - iodefne.h IOレジスタ定義
 - main.c メイン関数
 - resetprg.c スタートアップルーチン
 - stacksct.h スタックサイズ定義
 - switch_push.mtpj CubeSuite+プロジェクトファイル
 - typedefine.h 変数型定義
 - vect.h ベクタ関数のヘッダ
 - vecttbl.c 固定ベクタテーブル

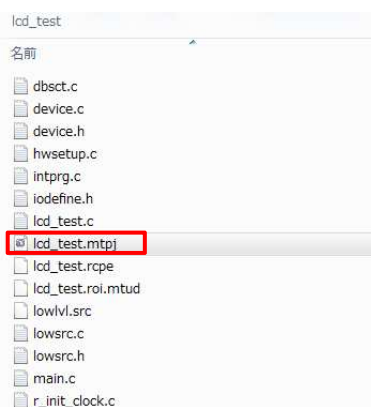
2016/9/16

125



lcd_testプログラム: 開発環境の起動

- 実習用のプロジェクトファイルを選択
 - program¥lcd_test¥lcd_test.mtpj



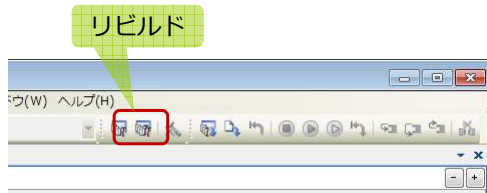
2016/9/16

126

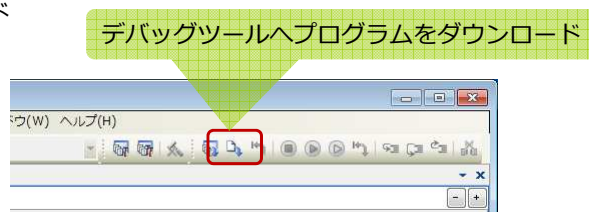


lcd_testプログラム:ビルド

- RXシリアルの「プロパティ」にてデバッガ用COMポートに設定を合わせる
- リビルド



- ダウンロード



※詳細な実行方法はled_shiftの項を参照

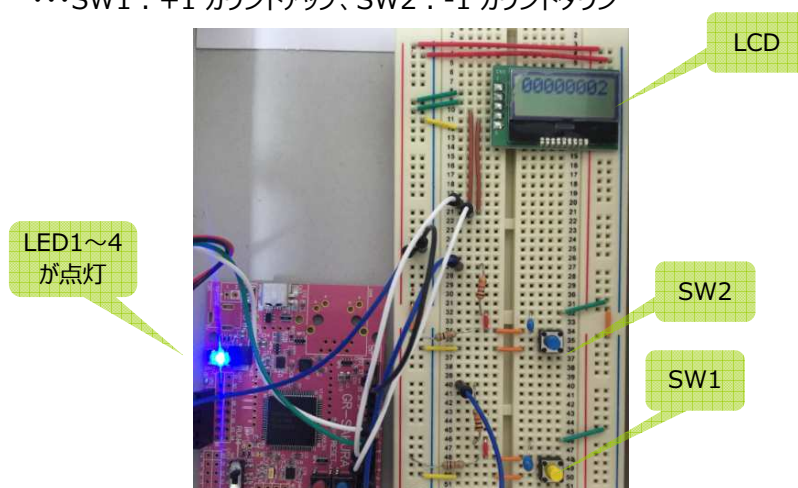
2016/9/16

127



lcd_testプログラム:実行

- キー(SW1/SW2)を押下するごとにLCD上で演算、LEDも点灯
...SW1 : +1 カウントアップ、SW2 : -1 カウントダウン



2016/9/16

128



lcd_testプログラムの説明: メイン関数

lcd_test/main.c



2016/9/16

129



カップラーメンタイマソフト説明

2016/9/16

130



カップラーメンタイマの仕様

- 概要
 - － カップラーメンの完成を知らせるタイマ
 - － 30秒刻みで、お知らせ時間を設定
 - － タイマをスタートしてから、設定した時間が経過したことブザーでLEDで知らせる
- スイッチとLEDの使い方
 - － LED3 : 電源がONであること、(プログラムが動いていること)を表示
 - － LED2 : SW1が押されたことを表示
 - － LED1 : タイマが動いていることを表示
 - － SW2 : タイマを起動・停止
 - － SW1 : 設定時間を30秒延長
タイマー停止中はシステム時間の表示切替

2016/9/16

131



カップラーメンタイマの詳細仕様

- － 電源をONの間(プログラムが動いている間)、LED3の点灯・消灯を1秒間隔で繰り返す
- － SW2がONになると設定時間を30秒にして、タイマを起動する
- － SW2が再度ONになると、タイマを停止する
- － タイマが動いている間にSW1がONになると、ONの間LED2を点灯させ、設定時間を30秒延長する
- － タイマが動いている間は、LED1を点灯する
- － 設定時間が経過すると、LED1を0.25秒間隔で点滅させて、15秒後に消灯する。その間ブザーを鳴らす。

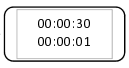
2016/9/16

132



カップラーメンタイマの拡張仕様

LCD表示を追加する

- LCDにタイマ停止の状態を表示する 
- タイマが動いている間はタイマ設定時間と経過時間を表示する 
- タイマ終了を表示で知らせる。 
- タイマ停止の状態でSW1がONになると、ONの間LED2を点灯させ、カウンターの表示とREADY表示を切り替える

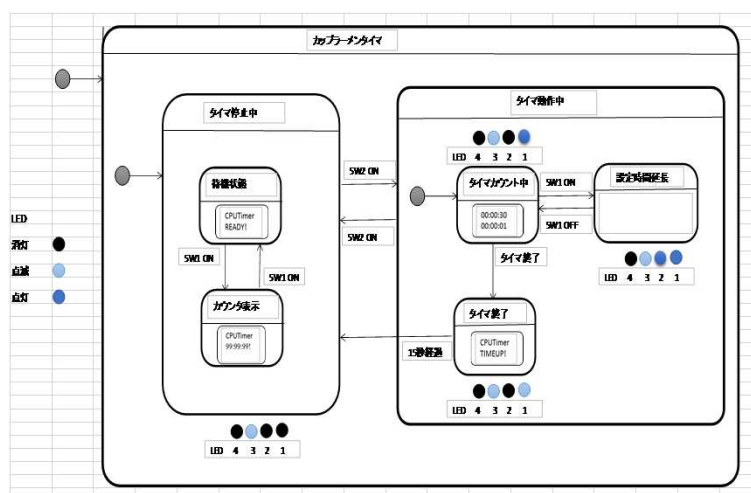


2016/9/16

133



カップラーメンタイマの状態遷移図



2016/9/16

134



カップラーメンタイマプログラム

cup_timer

- ハードウェアの操作に関してはこれまでに学習した内容を用いる
 - ブザーとキーを動かそう
 - ハードウェアの初期化
 - LED点灯と消灯
 - SWの状態変化を監視
 - ブザーを鳴らす
 - タイマー割込みによる時間をカウントする
 - LCDを動かそう
 - ハードウェアの初期化
 - LCD表示処理

2016/9/16

135



作成の手順

- Step1 :
 - デバイスドライバの作成
- Step2 :
 - タイマによる基本時間(0.25秒)の作成
- Step3 :
 - メイン関数でのLED3の1秒間隔の点灯の実現
- Step4 :
 - スイッチプロセスの作成
- Step5 :
 - タイマプロセスの状態遷移判定と各状態の大枠作成
- Step6 :
 - タイマプロセスの状態遷移判定と各状態の詳細
- Step7 :
 - 拡張仕様(LCD表示)の作成

2016/9/16

136



Step1 : デバイスドライバの作成

LED, スイッチ, ブザー、タイマ、LCDの操作関数を作成

switch_push の初期化関数と操作関数を流用

device.c device.h

- LED1,2,3,4を個別にON,OFFする関数

```
void set_led_state(unsigned char led, unsigned char state)
```

- プッシュSWの状態を個別に読み込む関数

```
unsigned char get_push_state(unsigned char sw)
```

- ブザーモードの制御

```
void set_buzzer_mode(unsigned char mode)
```

2016/9/16

137



Step1 : デバイスドライバの作成

LED, スイッチ, ブザー、タイマ、LCDの操作関数を作成

switch_push の初期化関数と操作関数を流用

device.c device.h

- LCDのロケート設定

```
void LCD_locate(int x, int y)
```

- LCDのデータ送信

```
void LCD_puts(char *buffer)
```

2016/9/16

138



Step2 :タイマによる基本時間(0.25秒)の作成

- グローバル変数
 - 1msec毎にインクリメントする変数(BaseTime)を用意
- タイマ割込みハンドラ
 - 1msec周期で実行され, BaseTimeをインクリメント
- メイン関数
 - BaseTimeを用いて250msec周期の処理を実現
 - 各ハードウェアの初期化と割込みの許可
 - 無限ループ内でBaseTimeをチェックする

2016/9/16

139



Step2 : プログラム例 タイマ割込みハンドラ

- グローバル変数 BaseTime
 - unsigned int (32bit)とする

```
unsigned int BaseTime;
```

- タイマ割込み
 - タイマ割り込み初期化

```
tpu0_int_init(base_time_handler, TPU0_INT_LVL);
```

```
static void base_time_handler(void){
    BaseTime++;
}
```

コールバック関数から呼び出されるハンドラ

2016/9/16

140



Step2 : プログラム例 メイン関数

- 250msec周期の処理

```
#define TIMER_GRANULE 250  /* 時間単位 */

void
_main(void){
    unsigned long timer250 = TIMER_GRANULE;
    .....
    BaseTime = 0;

    for (;;) {
        if (timer250 <= BaseTime) { /* 250m周期で実行 */
            /* 現在時刻の250msec後を設定 */
            timer250 += TIMER_GRANULE;
        }
    }
}
```

タイマ割込み
ハンドラで1ms
周期でインクリ
メントされる

2016/9/16

141



Step3 :メイン関数でのLED3の1秒間隔の点灯の実現

- 250msec周期を用いて1秒毎の処理を実現
 - 4回の実行で1秒
- 1秒毎の処理
 - LED3の状態を反転させる
- メイン関数内のローカル変数
 - LED3の点灯状態を保持する変数(led3)
 - 1秒を生成するための変数(sec)

2016/9/16

142



Step3 : プログラム例 メイン関数

- 4回実行されるとLED1を反転させる

```
#define TO_SEC      4    /* 1秒間の呼び出し回数 */
void _main(void){
    unsigned char sec = 0;
    unsigned char led3 = OFF;
    ....
    if (timer250 <= BaseTime) {
        timer250 += TIMER_GRANULE;
        if (++sec == TO_SEC) {
            if (led3 == ON) {
                led3 = OFF;
            } else {
                led3 = ON;
            }
            sec = 0;
            set_led_state(LED3, led3);
        }
    }
}
```

4回実行

LED3反転

LED3設定

2016/9/16

143



Step4 : スイッチプロセスの作成

- イベントの定義とビット割り当て
 - 開始 : EVT_TIMER_TRIG 0x01
 - 時間アップ : EVT_TIMER_COUNT 0x04
- グローバル変数
 - スイッチの状態を保持するための変数 (Push1, Push2)
 - イベント通知用変数 (Event)
- スイッチプロセス
 - SW1, SW2の変化をチェックし、イベント通知用変数をセットする
 - SW1の状態に応じて、LED2を点灯・消灯させる

2016/9/16

144



Step4 : プログラム例 SW2の状態取得

- SW2の状態をチェックし, 状態が変化していれば, イベント変数に対応するイベントをセットする

```
void
switch_process(void){
    unsigned char sw;
    sw = get_push_state(PUSH2);
    if (Push2 != sw) {
        if (sw == ON) {
            Event |= EVT_TIMER_TRIG;
        }
        Push2 = sw;
    }
}
```

開始イベント

2016/9/16

145



Step4 : プログラム例 SW1の状態取得

- SW1の状態をチェックし, 状態が変化していれば, イベント変数に対応するイベントをセットする。
- SW1がONの間LED2を点灯する

```
sw = get_push_state(PUSH1);
if (Push1 != sw) {
    if (sw == ON) {
        Event |= EVT_TIMER_COUNT;
        set_led_state(LED2, ON);
    } else {
        set_led_state(LED2, OFF);
    }
    Push1 = sw;
}
```

時間UPイベント

LED2:ON

LED2:OFF

2016/9/16

146



Step5 : タイマプロセスの大枠とLED1点灯の実現

タイマプロセスの状態遷移判定と各状態の大枠作成
LED1の点灯の実現

- タイマプロセスの状態を示す列挙型 (TIMER_MODE)
 - 計時終了 : STOP_TIMER_MODE
 - 計時中 : ACT_TIMER_MODE
 - タイムアウト : TIMEOUT_MODE
- タイマプロセスのローカル変数 (static変数)
 - タイムアウト時間を保持する変数 (maximum_time)
 - 現在の計時時間を保持する変数 (current_time)
 - カップラメントイマのモードを保持する変数 (timer_mode)
 - LED1のタイミング生成用変数 (alarm)
 - LED1の点灯状態を保持する変数 (led1)

2016/9/16

147



Step5 : 列挙型とローカル変数

- タイマプロセスの状態を示す列挙型 (TIMER_MODE)

```
enum TIMER_MODE {
    STOP_TIMER_MODE, /* 計時終了モード */
    ACT_TIMER_MODE,  /* 計時中モード */
    TIMEOUT_MODE     /* タイムアウトモード */
};
```

- タイマプロセスのローカル変数 (static変数)
 - タイマプロセスは250msec周期で呼び出され、終了するため変数の値を保存するために static変数とする

```
static unsigned long maximum_time;
static unsigned long current_time;
static unsigned char timer_mode = STOP_TIMER_MODE;
static unsigned char alarm = 0;
static unsigned char led1 = OFF;
```

2016/9/16

148



Step5 : 定数マクロの定義

- 各定数をマクロで定義する
 - タイムアウト時間の初期値
 - 時間アップ(延長)単位
 - タイムアウト時のLED1の点滅時間

```
#define INIT_TIME      30  /* スタート時のタイムアウト時間 */
#define TIME_UNIT      30  /* 時間アップ単位 */
#define TIMEOUT_BLINK  15  /* タイムアウト時の点滅時間 */
```

2016/9/16

149



Step5 : プログラム例 タイマプロセス状態遷移判定

- イベント通知用変数を参照してカップラーメンタイマのモード変数(状態)を設定する

```
if (Event != 0) {          /* switch_processより通知 */
    if (Event & EVT_TIMER_TRIG) {
        if(timer_mode == STOP_TIMER_MODE){
            timer_mode = ACT_TIMER_MODE; /* 開始 */
        }else{
            timer_mode = STOP_TIMER_MODE; /* 停止 */
        }
    }
    if (Event & EVT_TIMER_COUNT) { /* 時間アップ */
        if (timer_mode == ACT_TIMER_MODE) {

        }
    }
    Event = 0;
}
```

2016/9/16

150



Step5 : プログラム例 各状態毎の処理

- timer_modeによりモードを判定して実行

```
switch (timer_mode) {
  case ACT_TIMER_MODE: /* 計時中モード */
    if(current_time >= maximum_time){ /* タイムアウト */
      timer_mode = TIMEOUT_MODE;
    }
    break;
  case TIMEOUT_MODE: /* タイムアウトモード */
    /* 15秒間点滅したらタイマー停止状態に */

    break;
  case STOP_TIMER_MODE: /* 計時終了モード */
    break;
  default:
    break;
}
```

2016/9/16

151



Step5 : プログラム例 LED1の点灯

- LED1のタイミング生成用変数(alarm)が0以上なら点滅させる

alarm = TIMEOUT_BLINK*TO_SEC;

```
if (alarm > 0) { /* LED1点灯要求 : 250ms経過 */
  if(led1 == ON) {
    led1 = OFF;
  } else {
    led1 = ON;
  }
  alarm--;
} else
  timer_mode = STOP_TIMER_MODE;
}
set_led_state(LED1, led1); /* LED1の設定 */
```

2016/9/16

152



Step6 : タイマプロセスの状態遷移判定と各状態の詳細

- 状態遷移判定での処理
 - 開始
 - current_time を0で初期化
 - maximum_time を設定し30秒でタイムアウトするよう設定
 - 停止
 - alarm を0にしてLED1を消灯
 - 時間アップ
 - 計時中モードであれば maximum_time を30秒延長

2016/9/16

153



Step6 : タイマプロセスの状態遷移判定と各状態の詳細

- 各状態(モード)での処理
 - 計時中モード
 - タイムアウトしていればモードをタイムアウトモードに移行させる
 - LED1を点灯させる
 - ブザーをONにする
 - タイムアウトモード
 - 15秒経過したらLED1の点滅を停止し, 計時終了モードに移行する
 - 計時終了モード
 - LED1を消灯させる
 - ブザーをOFFにする

2016/9/16

154



Step6 : プログラム例 状態遷移判定の詳細

- 状態遷移と共に変数を設定する

```

if (Event & EVT_TIMER_TRIG){
    if(timer_mode == STOP_TIMER_MODE){
        timer_mode = ACT_TIMER_MODE;
        current_time = 0;
        maximum_time = INIT_TIME * TO_SEC;
    }else{
        timer_mode = STOP_TIMER_MODE;
    }
}
if (Event & EVT_TIMER_COUNT) {    /* 時間アップ */
    if (timer_mode == ACT_TIMER_MODE) {
        maximum_time += TIME_UNIT * TO_SEC;
    }
}

```

2016/9/16

155



Step6 : プログラム例 各状態の詳細

- 計時中モード

```

case ACT_TIMER_MODE:
    if (current_time >= maximum_time) {
        /* タイムアウト */
        alarm = TIMEOUT_BLINK*TO_SEC;
        timer_mode = TIMEOUT_MODE;
    }
    current_time++;
    led1 = ON;
    break;

```

2016/9/16

156



Step6 : プログラム例 各状態の詳細

- タイムアウトモード

```
case TIMEOUT_MODE:
    if(alarm > 0){
        if(led1 == ON)
            led1 = OFF;
        else
            led1 = ON;
        buzzer_mode = ON;
        alarm--;
    }else
        timer_mode = STOP_TIMER_MODE;
break;
```

2016/9/16

157



Step7 : 拡張仕様(LCD表示)の作成

- LCD表示文字と変数の定義

```
static const char title1[] = "CPUTimer";
static const char title2[] = "READY! ";
static const char title3[] = "TIMEUP! ";

char lcd_buf[2][12];
```

2016/9/16

158



Step7 : 拡張仕様(LCD表示)の作成

- 各状態(モード)での処理
 - 計時中モード
 - 設定されたタイムアウト時間を表示する
 - 計時中の時間を表示する
 - タイムアウトモード
 - title1とtitle3 を表示する
 - 計時終了モード
 - title1とtitle2 を表示する
 - SW1がONになるtitle2を、カウンター表示に切り替える

2016/9/16

159



Step7 : プログラム例

- 各状態(モード)での処理

```
case ACT_TIMER_MODE:
    set_time(&lcd_buf[0][0], maximum_time);
    set_time(&lcd_buf[1][0], current_time);

case TIMEOUT_MODE:
    strcpy(&lcd_buf[0][0], title1);
    strcpy(&lcd_buf[1][0], title3);

case STOP_TIMER_MODE:
    strcpy(&lcd_buf[0][0], title1);
    if(base_disp_mode == OFF)
        strcpy(&lcd_buf[1][0], title2);
    else
        set_time(&lcd_buf[1][0], BaseTime/TIMER_GRANULE);
```

2016/9/16

160



Step7 :プログラム例

- timer_processにLCD表示処理を追加

```
LCD_locate(0, 0);  
LCD_puts(&lcd_buf[0][0]);  
LCD_locate(0, 1);  
LCD_puts(&lcd_buf[1][0]);
```

2016/9/16

161



Step7 :プログラム例

- 時間を時:分:秒の文字列に変換する

```
void set_time(char *buf, long time)  
{  
    int wk = time/TO_SEC;  
    set_value(&buf[6], wk % 60);  
    wk = wk / 60;  
    buf[5] = ':';  
    set_value(&buf[3], wk % 60);  
    wk = wk / 60;  
    buf[2] = ':';  
    set_value(&buf[0], wk);  
}
```

2016/9/16

162



まとめ

2016/9/16

163



著者リスト

- はじめに
- 開発環境の説明
- デバッガとLCDの点灯確認
 - 竹内 良輔((株)リコー)
- ブザーを動かそう
- LCDを動かそう
 - 田中裕貴((株)リコー)
- カップラーメンタイマソフト説明
 - 森田浩((株)アドバンスドデータコントロール)

2016/9/16

164

