RA6M2-144 タッチキー評価キット
RA6M1-100 タッチキー評価キット
[ソフトウェア編]
取扱説明書

ルネサス エレクトロニクス社　RA6M2/RA6M1 搭載　HSB シリーズマイコンボード向け評価キット

-本書を必ずよく読み、ご理解された上でご利用ください

株式会社北斗電子

REV.1.0.1.0

ー目　次ー

注意事項 **1**

安全上のご注意 **2**

概要 **4**

サンプルプログラム CD **5**

1.　サンプルプログラムのビルドと書き込み **7**

2.　サンプルプログラムの動作 **32**

2.1.　自己容量タイプタッチキー基板 32
2.2.　相互容量タイプタッチキー基板 39

3.　タッチキー機能(CTSU)使用のフロー **48**

3.1.　全体の流れ 48
3.2.　電流オフセットレジスタ値の決め方 49
3.3.　初期センサ値の取得 52
3.4.　タッチ閾値の最適化（オプション） 52
3.5.　タッチ判定 52

4.　サンプルプログラムの解説 **53**

4.1.　ソースファイル構成 53
4.2.　プログラムのフロー 55
4.3.　CTSU 処理の流れ 59
4.4.　電流オフセットの最適化 61
4.5.　タッチの判定 62
4.5.1.　自己容量タイプ 62
4.5.1.　相互容量タイプ 63

5.　サンプルプログラムの関数 **65**

5.1.　ctsu 65
5.2.　common 70
5.3.　intr 71
5.4.　lcd_1602 72
5.5.　rtc 75
5.6.　sci 77
5.7.　timer 82

6.　サンプルプログラムの変数、定数値 **84**

6.1.　CTSU で使用しているグローバル変数 84
6.2.　その他で使用しているグローバル変数 87
6.3.　CTSU で使用している定数値 88
6.4.　その他で使用している定数値（抜粋） 91

7.　備考 **92**

7.1. ボード上の LED 92
取扱説明書改定記録 93
お問合せ窓口 93

注意事項

本書を必ずよく読み、ご理解された上でご利用ください

【ご利用にあたって】

1. 本製品をご利用になる前には必ず取扱説明書をよく読んで下さい。また、本書は必ず保管し、使用上不明な点がある場合は再読し、よく理解して使用して下さい。
2. 本書は株式会社北斗電子製マイコンボードの使用方法について説明するものであり、ユーザシステムは対象ではありません。
3. 本書及び製品は著作権及び工業所有権によって保護されており、全ての権利は弊社に帰属します。本書の無断複写・複製・転載はできません。
4. 弊社のマイコンボードの仕様は全て使用しているマイコンの仕様に準じております。マイコンの仕様に関しましては製造元にお問い合わせ下さい。弊社製品のデザイン・機能・仕様は性能や安全性の向上を目的に、予告無しに変更することがあります。また価格を変更する場合や本書の図は実物と異なる場合もありますので、御了承下さい。
5. 本製品のご使用にあたっては、十分に評価の上ご使用下さい。
6. 未実装の部品に関してはサポート対象外です。お客様の責任においてご使用下さい。

【限定保証】

1. 弊社は本製品が頒布されているご利用条件に従って製造されたもので、本書に記載された動作を保証致します。
2. 本製品の保証期間は購入戴いた日から1年間です。

【保証規定】

保証期間内でも次のような場合は保証対象外となり有料修理となります

1. 火災・地震・第三者による行為その他の事故により本製品に不具合が生じた場合
2. お客様の故意・過失・誤用・異常な条件でのご利用で本製品に不具合が生じた場合
3. 本製品及び付属品のご利用方法に起因した損害が発生した場合
4. お客様によって本製品及び付属品へ改造・修理がなされた場合

【免責事項】

弊社は特定の目的･用途に関する保証や特許権侵害に対する保証等、本保証条件以外のものは明示・黙示に拘わらず一切の保証は致し兼ねます。また、直接的・間接的損害金もしくは欠陥製品や製品の使用方法に起因する損失金・費用には一切責任を負いません。損害の発生についてあらかじめ知らされていた場合でも保証は致し兼ねます。
ただし、明示的に保証責任または担保責任を負う場合でも、その理由のいかんを問わず、累積的な損害賠償責任は、弊社が受領した対価を上限とします。本製品は「現状」で販売されているものであり、使用に際してはお客様がその結果に一切の責任を負うものとします。弊社は使用または使用不能から生ずる損害に関して一切責任を負いません。
保証は最初の購入者であるお客様ご本人にのみ適用され、お客様が転売された第三者には適用されません。よって転売による第三者またはその為になすお客様からのいかなる請求についても責任を負いません。
本製品を使った二次製品の保証は致し兼ねます。

安全上のご注意

製品を安全にお使いいただくための項目を次のように記載しています。絵表示の意味をよく理解した上でお読み下さい。

表記の意味

警告 取扱を誤った場合、人が死亡または重傷を負う危険が切迫して生じる可能性がある事が想定される

注意 取扱を誤った場合、人が軽傷を負う可能性又は、物的損害のみを引き起こすが可能性がある事が想定される

絵記号の意味

	**一般指示** 使用者に対して指示に基づく行為を強制するものを示します		**一般禁止** 一般的な禁止事項を示します
	**電源プラグを抜く** 使用者に対して電源プラグをコンセントから抜くように指示します		**一般注意** 一般的な注意を示しています

警告

以下の警告に反する操作をされた場合、本製品及びユーザシステムの破壊・発煙・発火の危険があります。マイコン内蔵プログラムを破壊する場合もあります。

1. 本製品及びユーザシステムに電源が入ったままケーブルの抜き差しを行わないでください。
2. 本製品及びユーザシステムに電源が入ったままで、ユーザシステム上に実装されたマイコンまたはIC等の抜き差しを行わないでください。
3. 本製品及びユーザシステムは規定の電圧範囲でご利用ください。
4. 本製品及びユーザシステムは、コネクタのピン番号及びユーザシステム上のマイコンとの接続を確認の上正しく扱ってください。

発煙・異音・異臭にお気付きの際はすぐに使用を中止してください。

電源がある場合は電源を切って、コンセントから電源プラグを抜いてください。そのままご使用すると火災や感電の原因になります。

⚠注意

以下のことをされると故障の原因となる場合があります。

1. 静電気が流れ、部品が破壊される恐れがありますので、ボード製品のコネクタ部分や部品面には直接手を触れないでください。
2. 次の様な場所での使用、保管をしないでください。

   ホコリが多い場所、長時間直射日光があたる場所、不安定な場所、衝撃や振動が加わる場所、落下の可能性がある場所、水分や湿気の多い場所、磁気を発するものの近く
3. 落としたり、衝撃を与えたり、重いものを乗せないでください。
4. 製品の上に水などの液体や、クリップなどの金属を置かないでください。
5. 製品の傍で飲食や喫煙をしないでください。

ボード製品では、裏面にハンダ付けの跡があり、尖っている場合があります。

取り付け、取り外しの際は製品の両端を持ってください。裏面のハンダ付け跡で、誤って手など怪我をする場合があります。

CD メディア、フロッピーディスク付属の製品では、故障に備えてバックアップ(複製)をお取りください。

製品をご使用中にデータなどが消失した場合、データなどの保証は一切致しかねます。

アクセスランプがある製品では、アクセスランプが点灯中に電源を切ったり、パソコンをリセットをしないでください。

製品の故障の原因となったり、データが消失する恐れがあります。

本製品は、医療、航空宇宙、原子力、輸送などの人命に関わる機器やシステム及び高度な信頼性を必要とする設備や機器などに用いられる事を目的として、設計及び製造されておりません。

医療、航空宇宙、原子力、輸送などの設備や機器、システムなどに本製品を使用され、本製品の故障により、人身や火災事故、社会的な損害などが生じても、弊社では責任を負いかねます。お客様ご自身にて対策を期されるようご注意ください。

概要

本書は、フラッシュメモリ内蔵のルネサス エレクトロニクス製RA6M2/RA6M1マイコン搭載ボードを使用したタッチキーの評価キットのサンプルソフトウェアを説明する資料となります。

タッチキーのプログラムを行う際、ルネサスエレクトロニクス製のツールとして、QE for touch を使用する方法もあり、そちらは別マニュアルで解説を行っています。本マニュアルでは、「自己容量タイプタッチキー基板(S16A)」及び「相互容量タイプタッチキー基板(D55A)」のキー読み取りの当社で作成したサンプルプログラムを解説していますので、CDに含まれるソースと本マニュアルを相互に参照頂きたく。

本書で解説するサンプルプログラムは触れているキーを検出する部分のエッセンスを抜き出したものとなっています。そのため、単純なコードとなっており、ルネサス RA マイコンのタッチキー機能(CTSU)の基本的な動作をモニタするのに適しているかと考えます。

実際のタッチキーアプリケーションを組む際には、ノイズや特性の経時変化等を考慮する必要がある場合がありますので、QE ベースでプログラムを作成するか、タッチキーの仕組みを理解した上で、お客様のシステムに応じたプログラム設計をお願い致します。

サンプルプログラム CD

・自己容量タッチキーサンプルプログラム（自己容量タッチキー基板(S16A)向け）
・相互容量タッチキーサンプルプログラム（相互容量タッチキー基板(D55A)向け）

ーソースファイルー

(1)マイコンボード毎の設定

フォルダ		内容
SOURCE¥RA_CTSU_RA6M2¥	src¥settings¥	RA6M2 向け 設定ファイルフォルダ
SOURCE¥RA_CTSU_RA6M1¥	src¥settings¥	RA6M1 向け 設定ファイルフォルダ

(2)自己容量/相互容量タッチキープログラムソース

フォルダ		内容
SOURCE¥RA_CTSU_SELF¥	src¥ctsu	自己容量タイプタッチキー制御
SOURCE¥RA_CTSU_MUTUAL¥	src¥ctsu	相互容量タイプタッチキー制御

(3)共通ソース

フォルダ		内容
SOURCE¥RA_CTSU¥	src¥common	共通定義フォルダ
	src¥intr	割り込み設定
	srclcd_1602	LCD 制御
	src¥sci	SCI(UART)ドライバ
	src¥timer	タイマドライバ
	src¥hal_entry.c	エントリ関数サンプル

(1)は、どちらか（マイコンボードに合わせて選択）を使用
(2)も、どちらか（タッチキー基板に応じて）を使用
(3)は、そのまま使用
となります。

本キットは、ソースファイルの形で構成されていますので、RA マイコン向けのどの開発環境で動作させる事は可能かと考えますが、開発環境は e2studio を想定しています。

サンプルプログラムは、e2studio(V7.8.0) + GNU Tools for ARM Embedded Processors (arm-none-eabi-gcc) 2019-q4 + FSP v1.1.0 で動作を確認しています。

ーバイナリファイルー

コンパイル、リンク後の mot ファイル(srec)を格納しているフォルダです。マイコンボードに書き込んで動作確認を行う用途等に使用してください。

フォルダ		内容
BINARY¥	RA6M2_CTSU_SELF.srec RA6M2_CTSU_MUTUAL.srec RA6M1_CTSU_SELF.srec RA6M1_CTSU_MUTUAL.srec	RA6M2 自己容量 RA6M2 相互容量 RA6M1 自己容量 RA6M1 相互容量

ーアーカイブファイルー

プロジェクトを、zip ファイルに固めたものです。e2studio のワークスペースにインポートする事が可能です。

フォルダ		内容
ARCHIVE¥	RA6M2_CTSU_SELF.zip RA6M2_CTSU_MUTUAL.zip RA6M1_CTSU_SELF.zip RA6M1_CTSU_MUTUAL.zip	RA6M2 自己容量 RA6M2 相互容量 RA6M1 自己容量 RA6M1 相互容量

※QE CapTouch 導入編　マニュアルで説明している部分です

フォルダ		内容
ARCHIVE_QE¥	RA6M2_CTSU_SELF_QE.zip RA6M2_CTSU_MUTUAL_QE.zip	RA6M2 自己容量 RA6M2 相互容量

※RA6M1(HSBRA6M1F100)に適用する場合は、Device の変更を行ってください

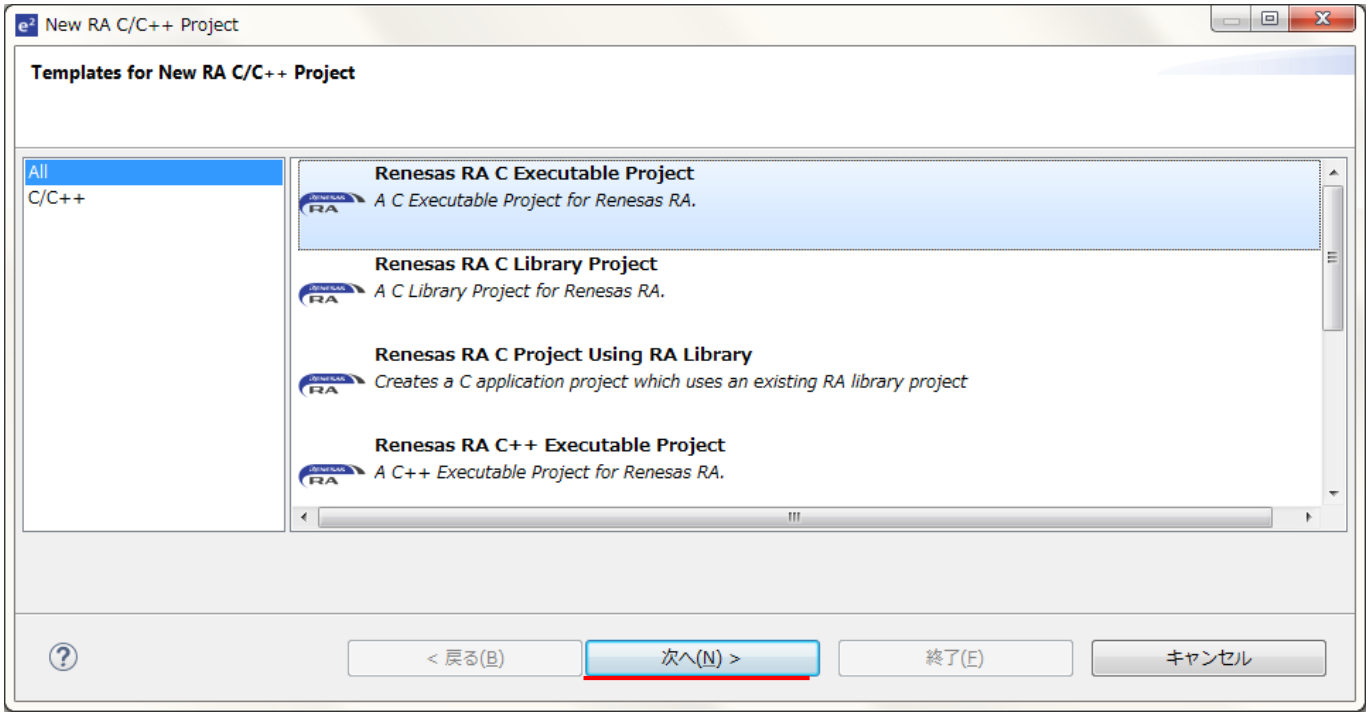
ードキュメントー

本書を含むマニュアル類を格納しているフォルダです。

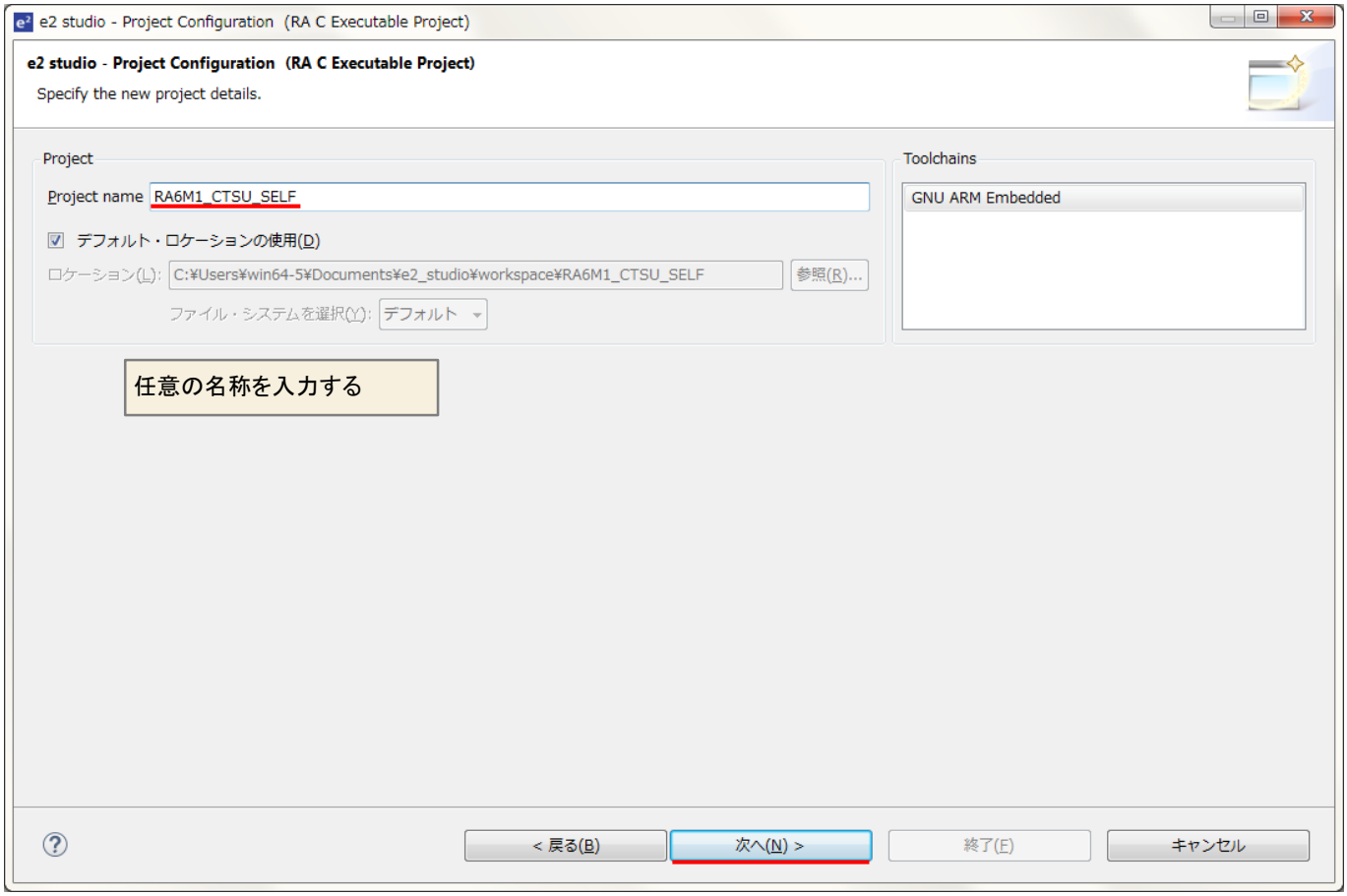
フォルダ		内容
MANUAL¥		マニュアル類

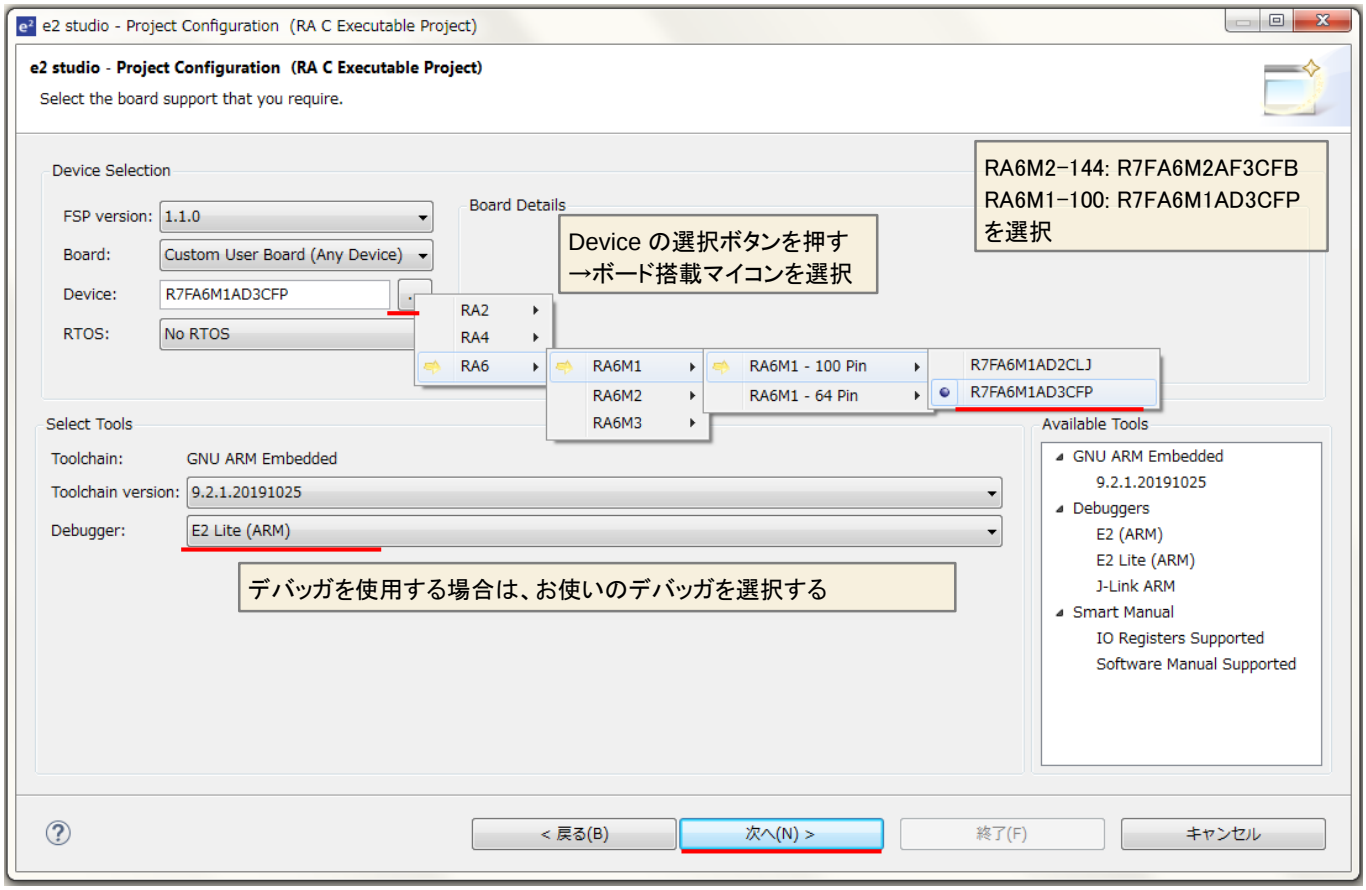
1. サンプルプログラムのビルドと書き込み

(1)マイコンボード（搭載マイコン種）に応じたプロジェクトを作成

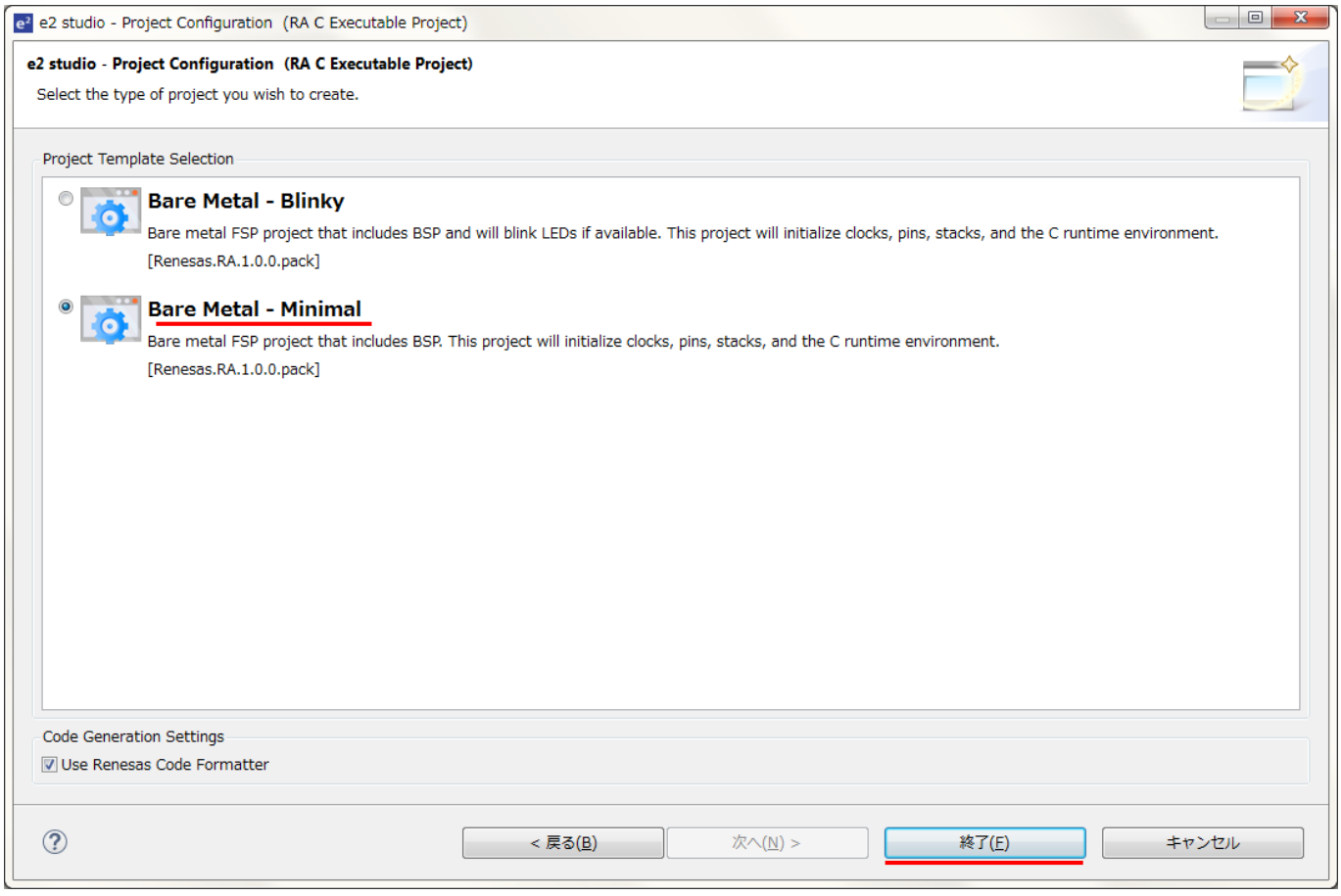


新規プロジェクト作成で、Renesas RA C Executable project を選択



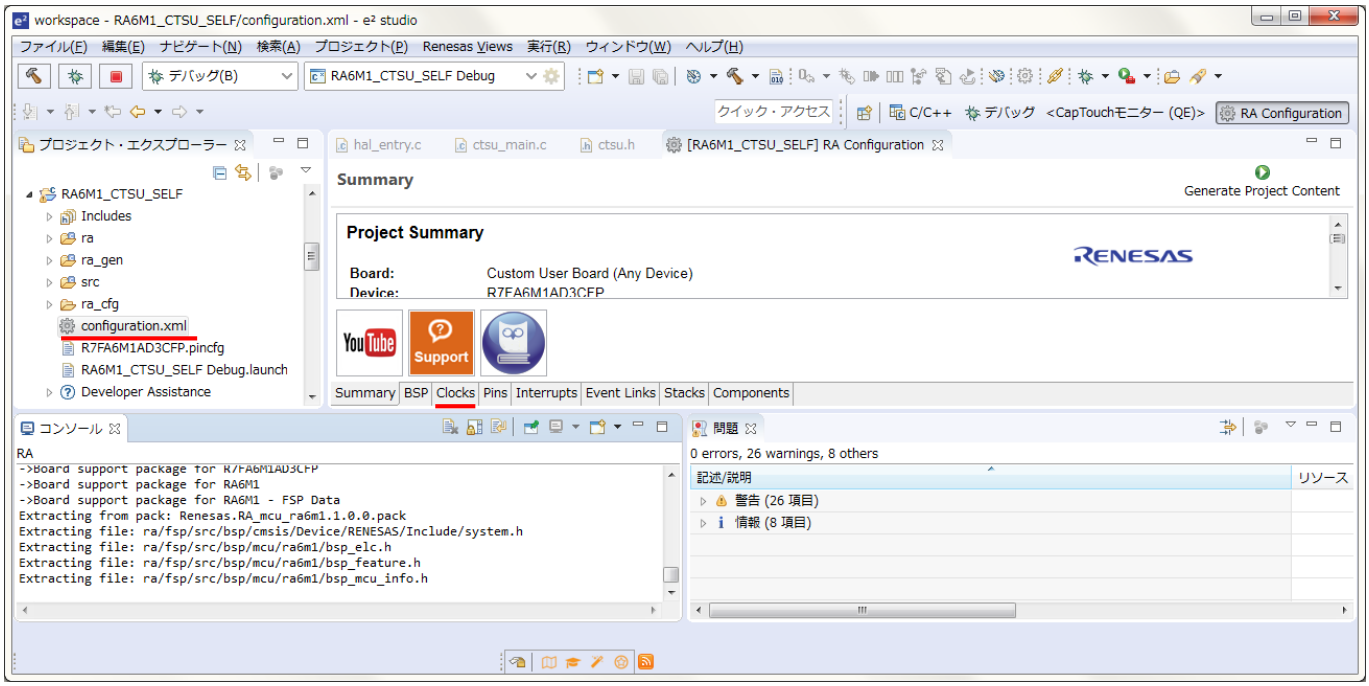


使用するマイコンボードに搭載されているマイコンを選択する。(デバッガを使用する場合は、デバッガを選択する)

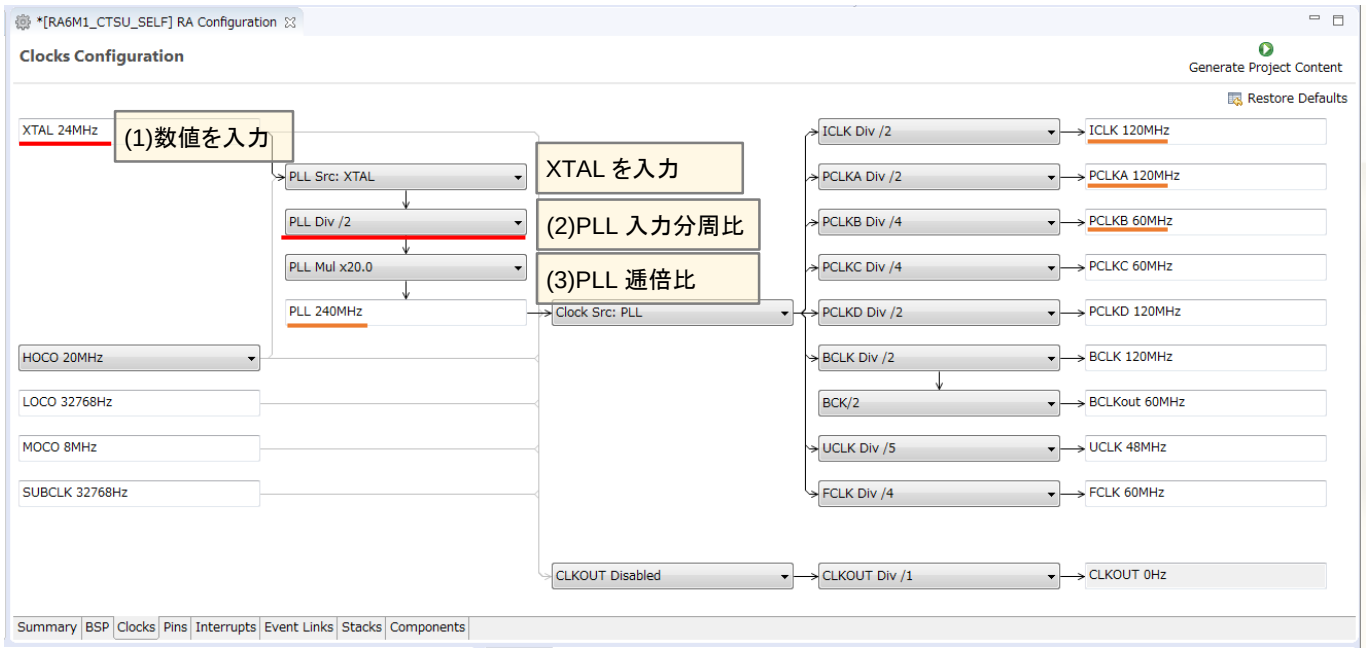


Bare Metal - Minimal 側を選択。終了。

(2)クロックの設定



Configuration.xml の Clocks タブを開く



※「RA6M2-144 タッチキー評価キット」、「RA6M1-100 タッチキー評価キット」で共通

(1)XTAL 24,000,000 (24MHz)を入力（ボード搭載クロック値）

(2)PLL Div/2 を選択　PLL 入力分周比 1/2

（他はデフォルト値, (3)の PLL 逓倍比は x20.0 を選択）

PLL 240MHz, ICLK,PCLKA 120MHz, PCLKB 60MHz となれば問題ありません。

（クロック設定は、マイコンのスペック内であれば選択は自由ですが、プログラム上、上記周波数を前提にしている部分があります。）

GUI でクロック設定後、Create Project Content ボタンを押すと、クロック設定のソースコードが自動生成されますが、この後割り込み設定を行い、最後にコード生成を実行させますので、ここではコード生成はせず先に進んで良いです。

(3)ソースをプロジェクトに追加（コピー）

プログラムは、ソースファイルの形で CD 内に格納されています。

SOURCE¥RA_CTSU_*xxx*¥src¥settings　　…(a)
（*xxx* は、マイコンタイプ名で分かれ"RA6M2"または、"RA6M1"です）

SOURCE¥RA_CTSU_*yyy*¥src¥ctsu　　…(b)
（*yyy* は、"SELF"（自己容量）または、"MUTUAL"（相互容量）です）
接続するタッチキー基板（S16A のときは"SELF"、D55A のときは"MUTUAL"）に合わせて選択してください。

SOURCE¥RA_CANST¥source¥*[folders]*　　…(c)
（[folders]は、"common", "intr", "lcd_1602", "rtc", "sci", "timer"です）

SOURCE¥RA_CANST¥source¥hal_entry.c　　…(d)

上記(a)(b)(c)フォルダを、作成したプロジェクトのソースフォルダ(src)以下にコピーしてください。
上記(d)を、src 以下の hal_entry.c に上書きするか、(d)を参照して、hal_entry.c を書き換えてください。

hal_entry.c（抜粋、コメントは一部削除）

```
#include "hal_data.h"

#include "intr/intr.h"
#include "ctsu/ctsu.h"

FSP_CPP_HEADER
void R_BSP_WarmStart(bsp_warm_start_event_t event);
FSP_CPP_FOOTER

void hal_entry(void) {
    /* TODO: add your own code here */

    intr_init();
    ctsu_main();
}

void R_BSP_WarmStart(bsp_warm_start_event_t event) {
    if (BSP_WARM_START_RESET == event) {
#if BSP_FEATURE_FLASH_LP_VERSION != 0

        R_FACI_LP->DFLCTL = 1U;

#endif
    }

if (BSP_WARM_START_POST_C == event) {
    /* C runtime environment and system clocks are setup. */

    /* Configure pins. */
    R_IOPORT_Open(&g_ioport_ctrl, &g_bsp_pin_cfg);
    }
}
```

追加

追加

hal_entry.c に追加が必要な部分を赤字で示します。

intr_init() は、割り込みの初期化。ctsu_main()は、タッチキーのメイン処理ルーチンとなります。

例えば、e2studio 上で、ターゲットマイコンが RA6M1 で、自己容量タイプのタッチキー基板を使用する "RA6M1_CTSU_SELF"というプロジェクトを作成したとします。その場合、ワークスペースフォルダ([Workspace])以下

[Workspace]¥RA6M1_CTSU_SELF¥src¥

上記フォルダ内に、

SOURCE¥RA_CTSU_RA6M1¥src¥settings¥ （ターゲットマイコン毎に異なる）
SOURCE¥RA_CANST_SELF¥src¥ctsu¥ （タッチキー基板毎に異なる）
SOURCE¥RA_CTSU¥src¥[folders]¥
SOURCE¥RA_CTSU¥src¥hal_entry.c

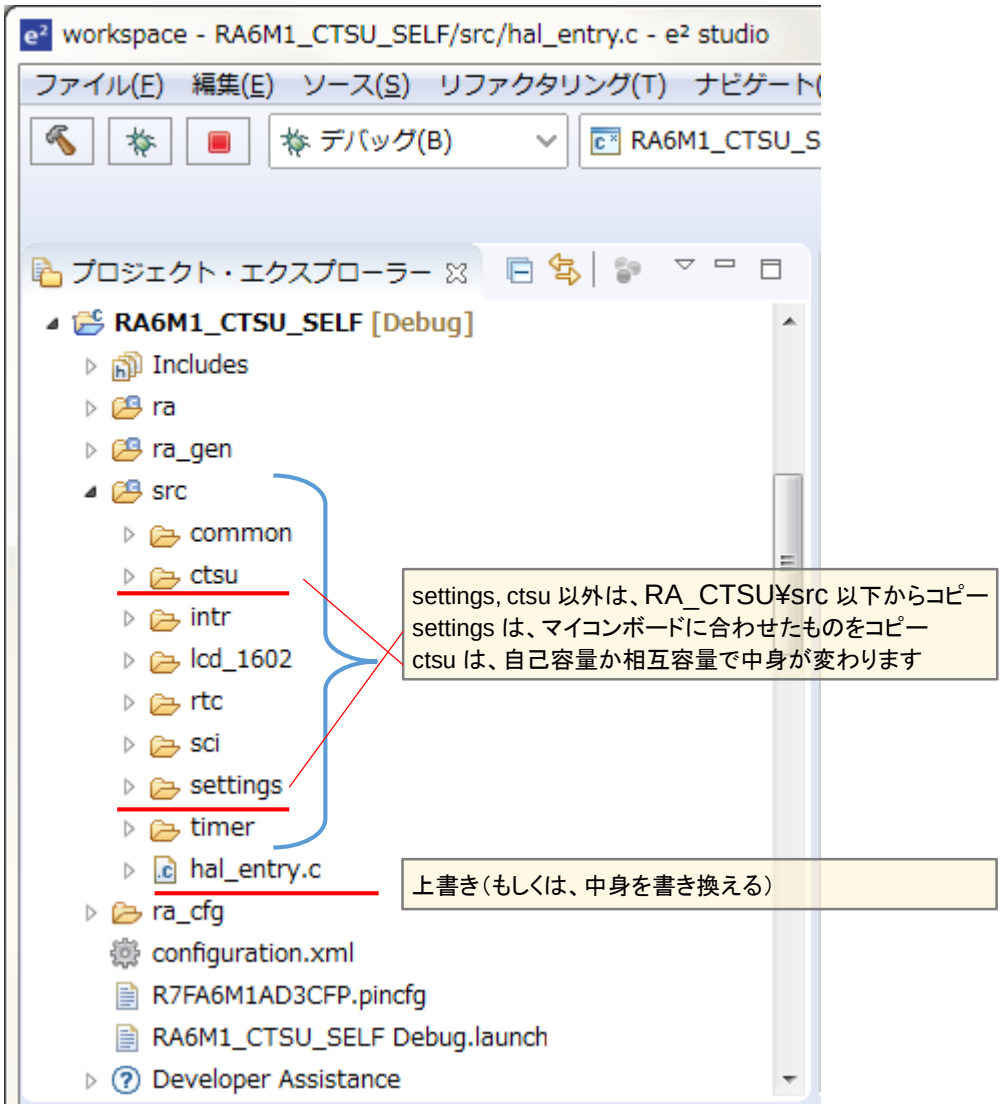
をコピーしてください。

コピー後は、

[Workspace]¥RA6M1_CTSU_SELF¥src¥common	(c)からコピー
[Workspace]¥RA6M1_CTSU_SELF¥src¥ctsu	(b)からコピー（タッチキー基板毎に異なる）
[Workspace]¥RA6M1_CTSU_SELF¥src¥intr	(c)からコピー
[Workspace]¥RA6M1_CTSU_SELF¥src¥lcd_1602	(c)からコピー
[Workspace]¥RA6M1_CTSU_SELF¥src¥rtc	(c)からコピー
[Workspace]¥RA6M1_CTSU_SELF¥src¥sci	(c)からコピー
[Workspace]¥RA6M1_CTSU_SELF¥src¥settings	(a)からコピー（マイコンボード毎に異なる）
[Workspace]¥RA6M1_CTSU_SELF¥src¥timer	(c)からコピー
[Workspace]¥RA6M1_CTSU_SELF¥src¥hal_entry.c	(d)からコピー（または(d)を参照して書き換える）

上記の様なフォルダ構成となります。（RA6M1_CTSU_SELF は、作成したプロジェクト名）

settings 以下には、マイコンボード毎の設定が入ります。(a)
ctsu 以下は、自己容量と相互容量で変わります。(b)
その他は、共通です。(c)(d)



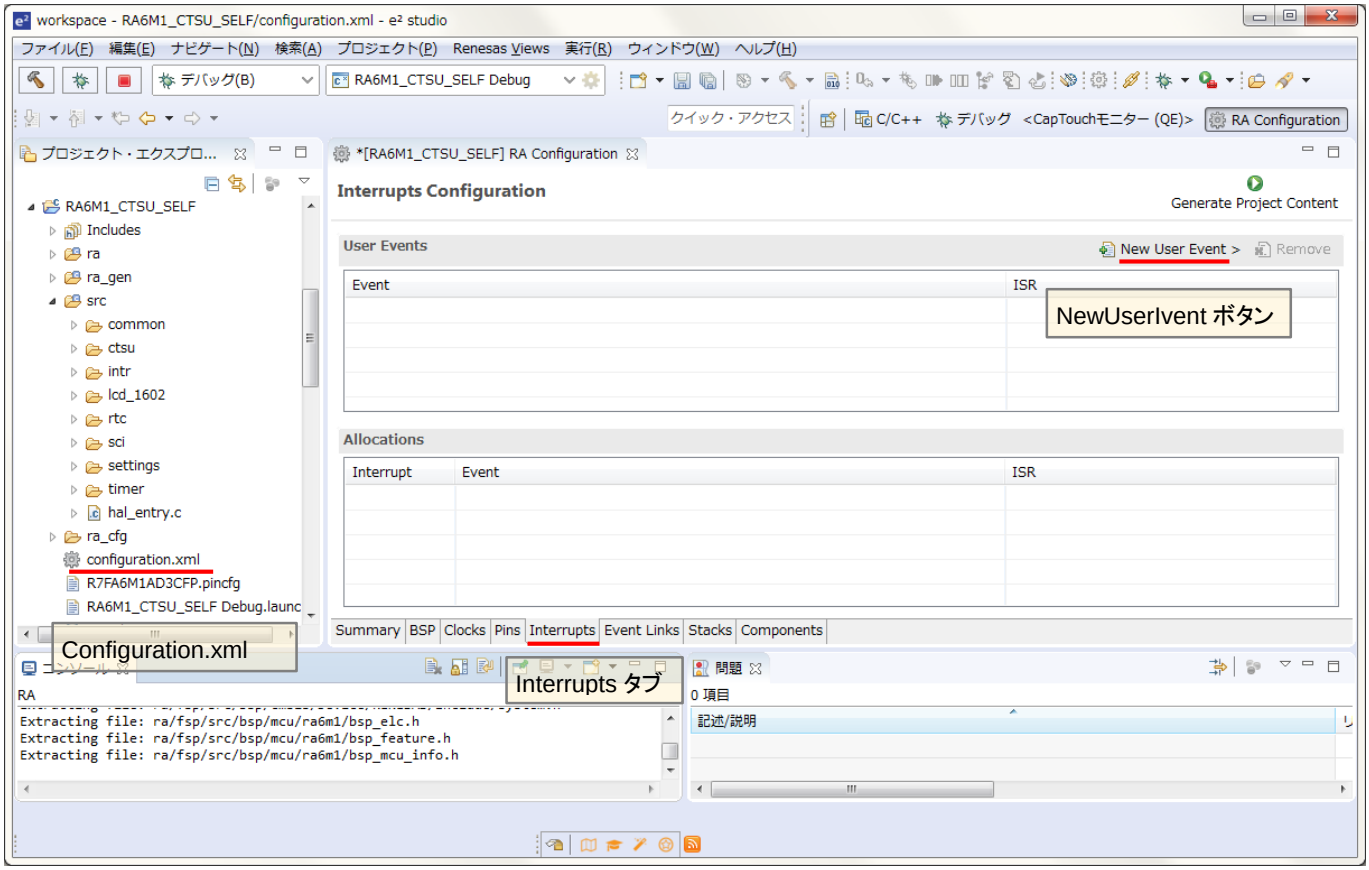
コピー後のフォルダは上記の様になります。I

(4)割り込み設定

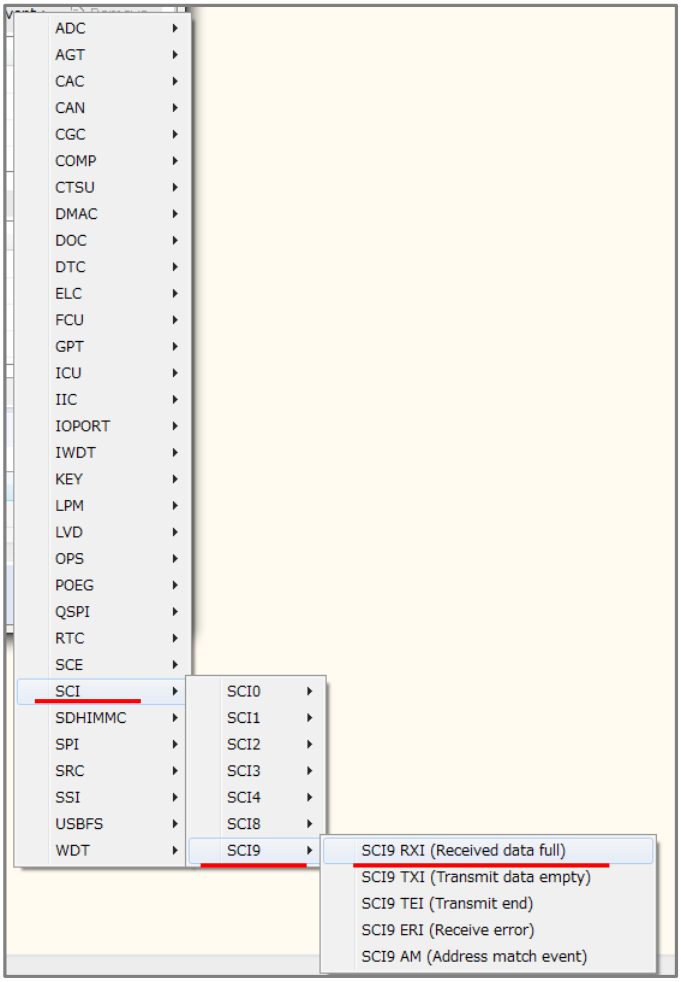
RA を e2studio+FSP(*1)の組み合わせで使用する際は、割り込みが GUI 上で設定できる様になっています。

(*1)ルネサスエレクトロニクスから提供されている RA 向けの Free Software Package

e2studio のプロジェクト画面を示します。

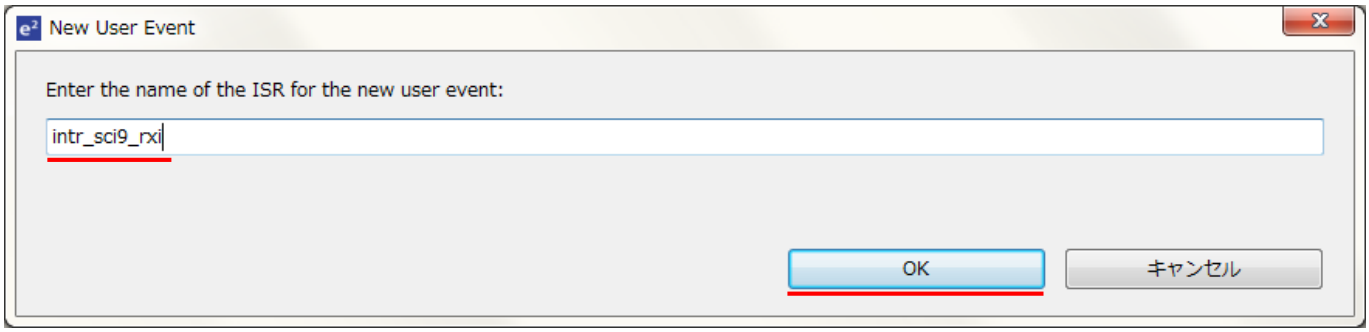


設定(Configuration.xml)の、Interrupt タブを開き、NewUserIvent を押してください



SCI - SCI9 - SCI9 RXI

を選択してください。

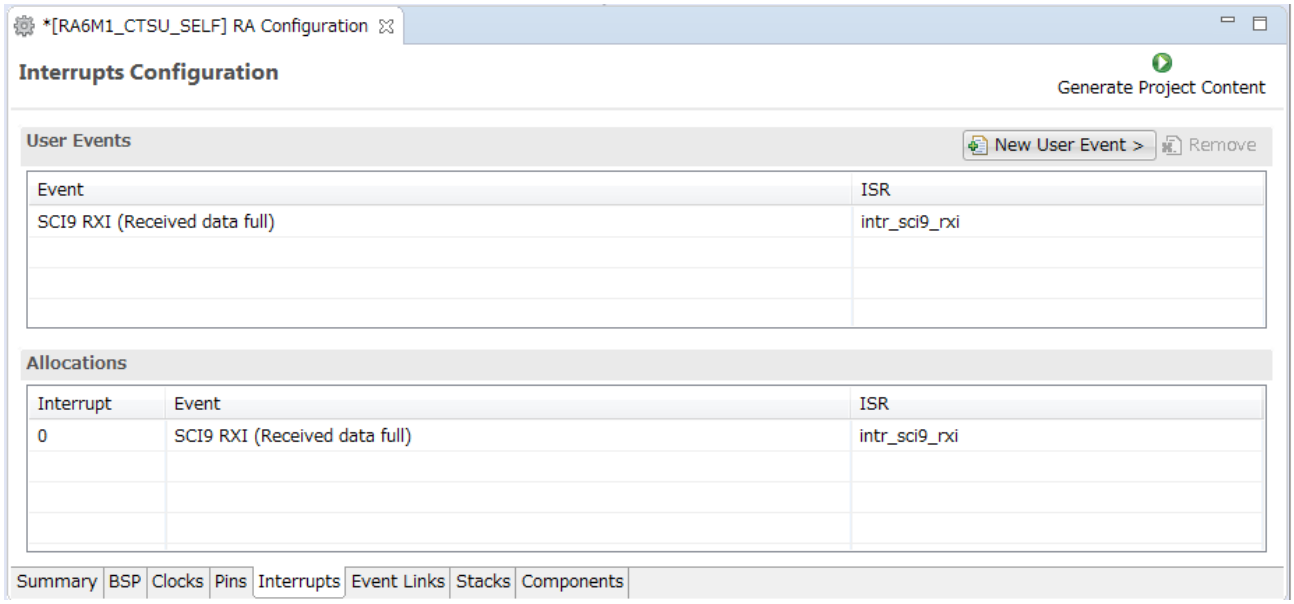


名称を入力するダイアログボックスが出ますので、

intr_sci9_rxi

と入力してください。OK を押す。

※この名称はソースファイル内の割り込み関数の関数名と対応していますので、別な名称とした場合は、ソースの一部変更が必要です

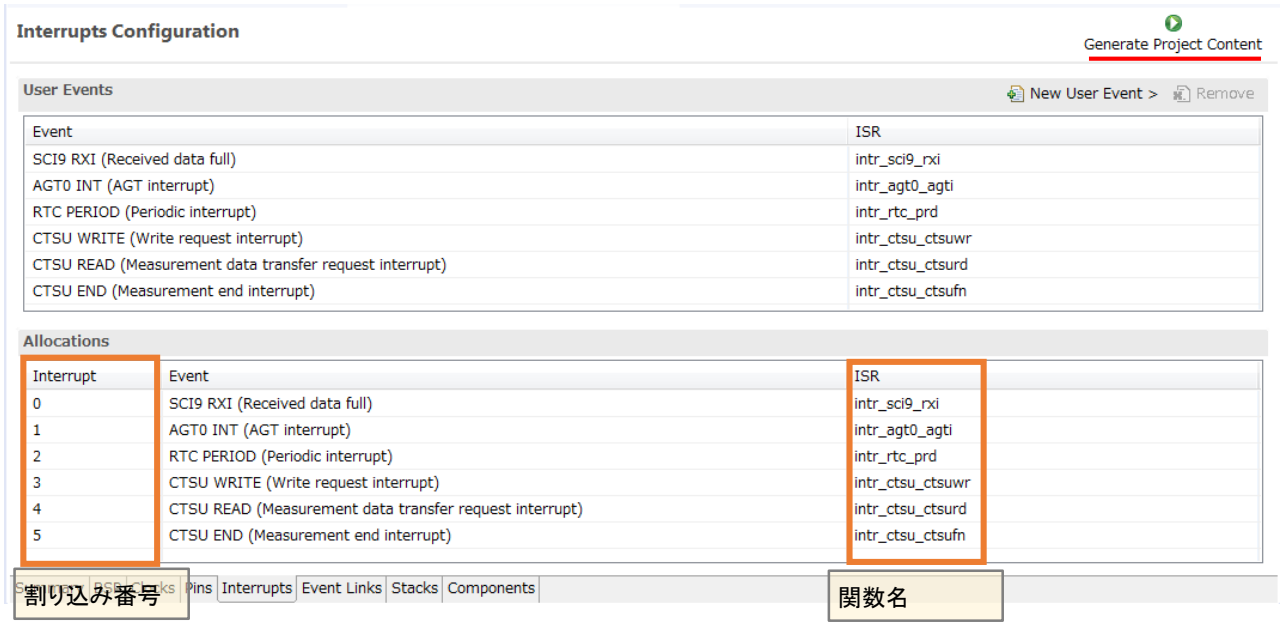


上記操作を行うと、割り込み設定に、SCI9-RXI 割り込みが、intr_sci9_rxi という名称（関数名に対応）で追加されます。

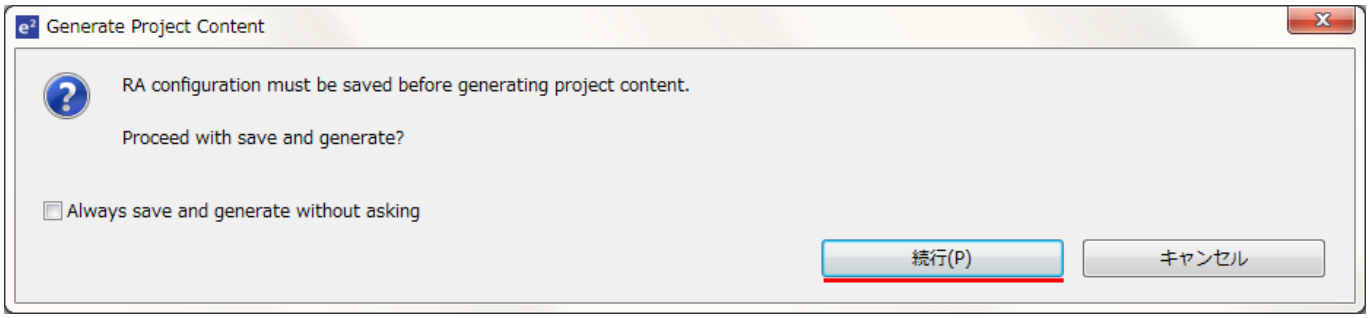
同様に、

グループ	モジュール	種別	定義名
AGT	AGT0	AGT0 INT	intr_agt0_agti
RTC		RTC PERIOD	intr_rtc_prd
CTSU		CTSU WRITE	intr_ctsu_ctsuwr
CTSU		CTSU READ	intr_ctsu_ctsurd
CTSU		CTSU END	intr_ctsu_ctsufn

AGT と RTC と CTSU の割り込みを追加してください。（表の上からの順番で）



追加後は、上記の様になります。Create Project Content を押してください。（コードが自動生成されます）



上記の画面が出た場合は、「続行」してください。

・割り込み番号定義

割り込み番号(*1)	割り込み種別	定義名(*2)
0	SCI9 RXI	intr_sci9_rxi
1	AGT0 INT	intr_agt0_agti
2	RTC PERIOD	intr_rtc_prd
3	CTSU WRITE	intr_ctsu_ctsuwr
4	CTSU READ	intr_ctsu_ctsurd
5	CTSU END	intr_ctsu_ctsufn

割り込み番号と割り込みの種別、定義名は上記となる様に設定してください。

(*1)が上記と異なる場合は、src¥common¥board_settings.h 内の定義番号を変更してください。

board_settings.h

```
//割り込み番号（IELSR の番号）
#define INTR_SCI9_RXI    0
#define INTR_AGT0_AGTI   1
#define INTR_RTC_PRD     2
#define INTR_CTSU_CTSUWR 3
#define INTR_CTSU_CTSURD 4
#define INTR_CTSU_CTSUFN 5
```

board_settings.h 内には、割り込み番号の定義（プログラム内や割り込みの初期化時に使用）がなされていますので、e2studio FSP の割り込み設定（番号）が上記と異なる場合は、board_settings.h 内の定義値（数値）を FSP の設定に従い変更してください。

(*2)が上記と異なる場合は、ソース内の関数名を(*2)に合わせて修正する必要があります。

intr_sci9_rxi

→src¥sci¥sci.c 内に、intr_sci9_rxi() という関数が定義されていますので、（FSP の設定を前出の名称から変更した場合は）FSP の設定と合わせる様にしてください。

intr_agt0_agti

→src¥timer¥agt.c 内に、intr_agt0_agti() という関数が定義されていますので、FSP の設定と合わせる様にしてください。

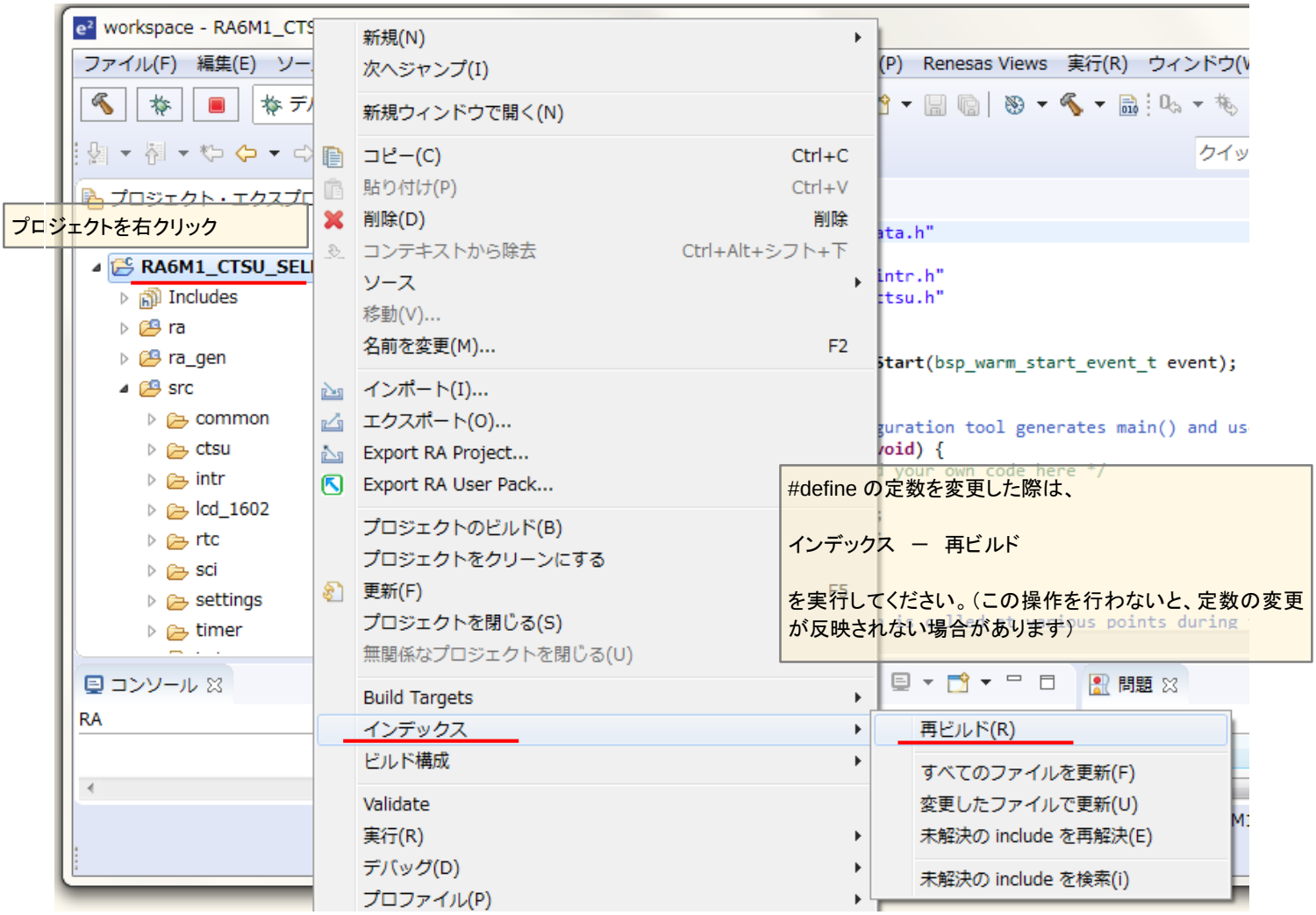
intr_rtc_prd

→src¥rtc¥rtc.c 内に、intr_rtc_prd() という関数が定義されていますので、FSP の設定と合わせる様にしてください。

intr_ctsu_ctsuwr, intr_ctsu_ctsurd, intr_ctsu_ctsufn

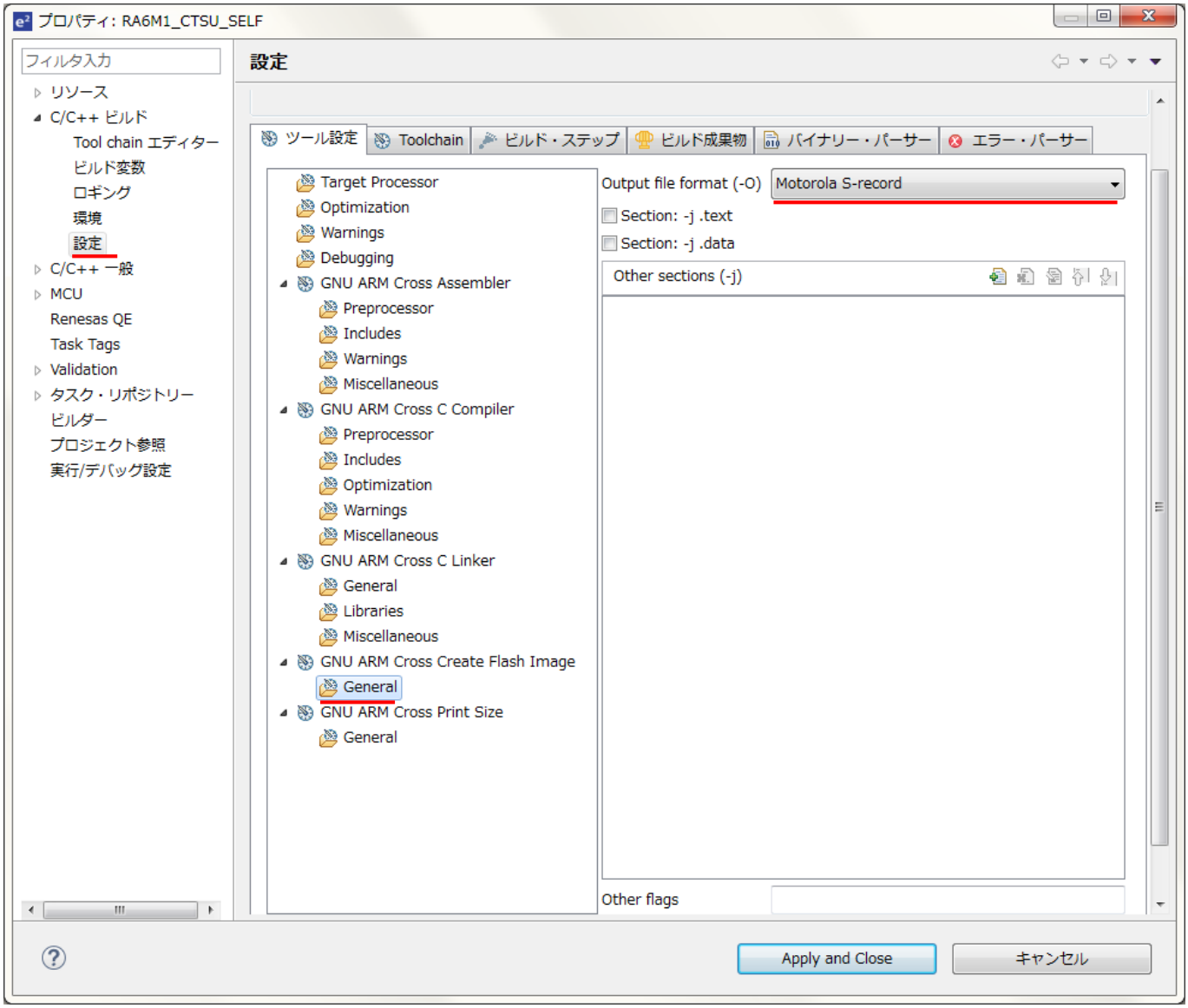
→src¥ctsu¥ctsu_intr.c 内に、intr_ctsu_ctsuwr(), intr_ctsu_ctsurd(), intr_ctsu_ctsufn()
という関数が定義されていますので、FSP の設定と合わせる様にしてください。

(5)ビルド



トンカチのアイコンをクリックすると、プログラムがビルドされます。

プロジェクトを右クリックして、プロパティを開き、



GNU ARM Cross Create Flash Image　ー　General

Output file format

で、生成されるファイル（モトローラ-S(mot)か Intel hex か）を変更可能です。

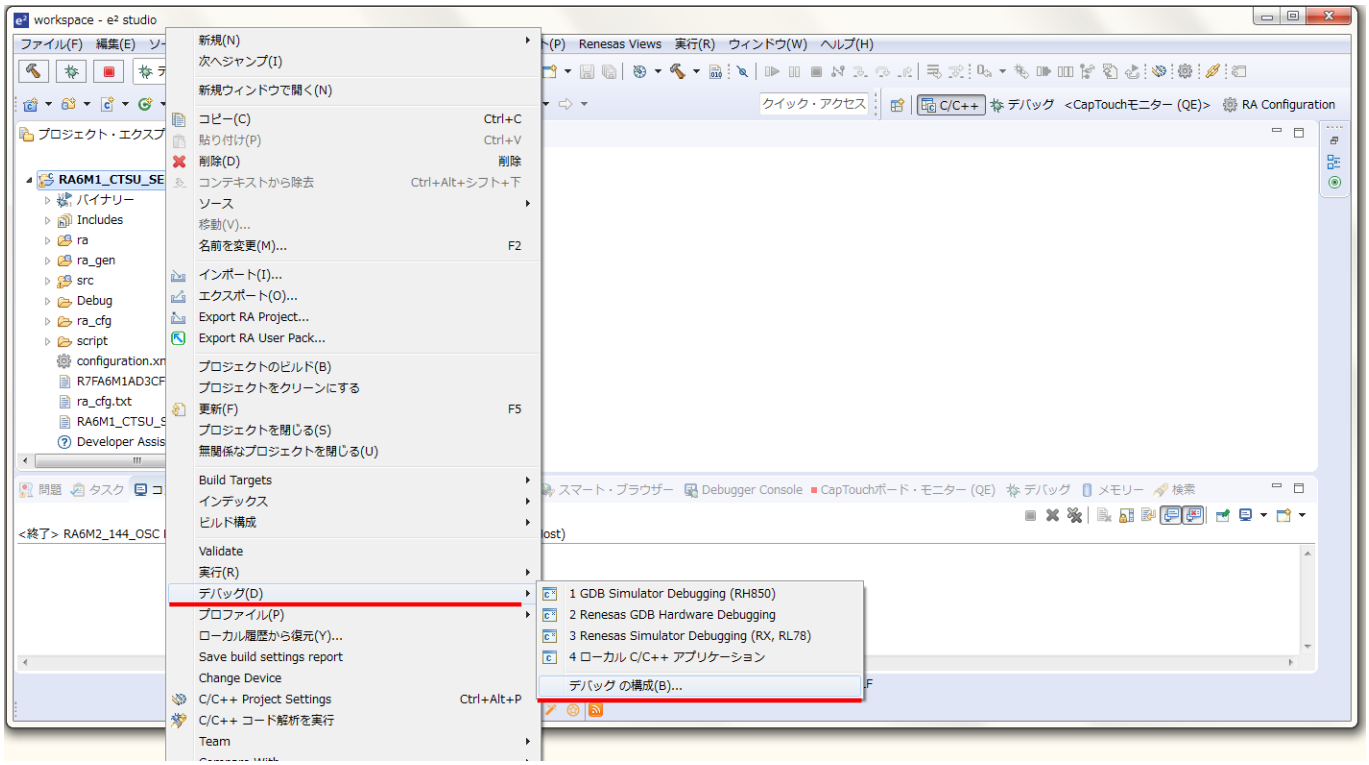
ビルド結果は、（デフォルトから変更しなければ）

[Workspace]¥*RA6M1_CTSU_SELF*¥Debug¥*RA6M1_CTSU_SELF*.srec

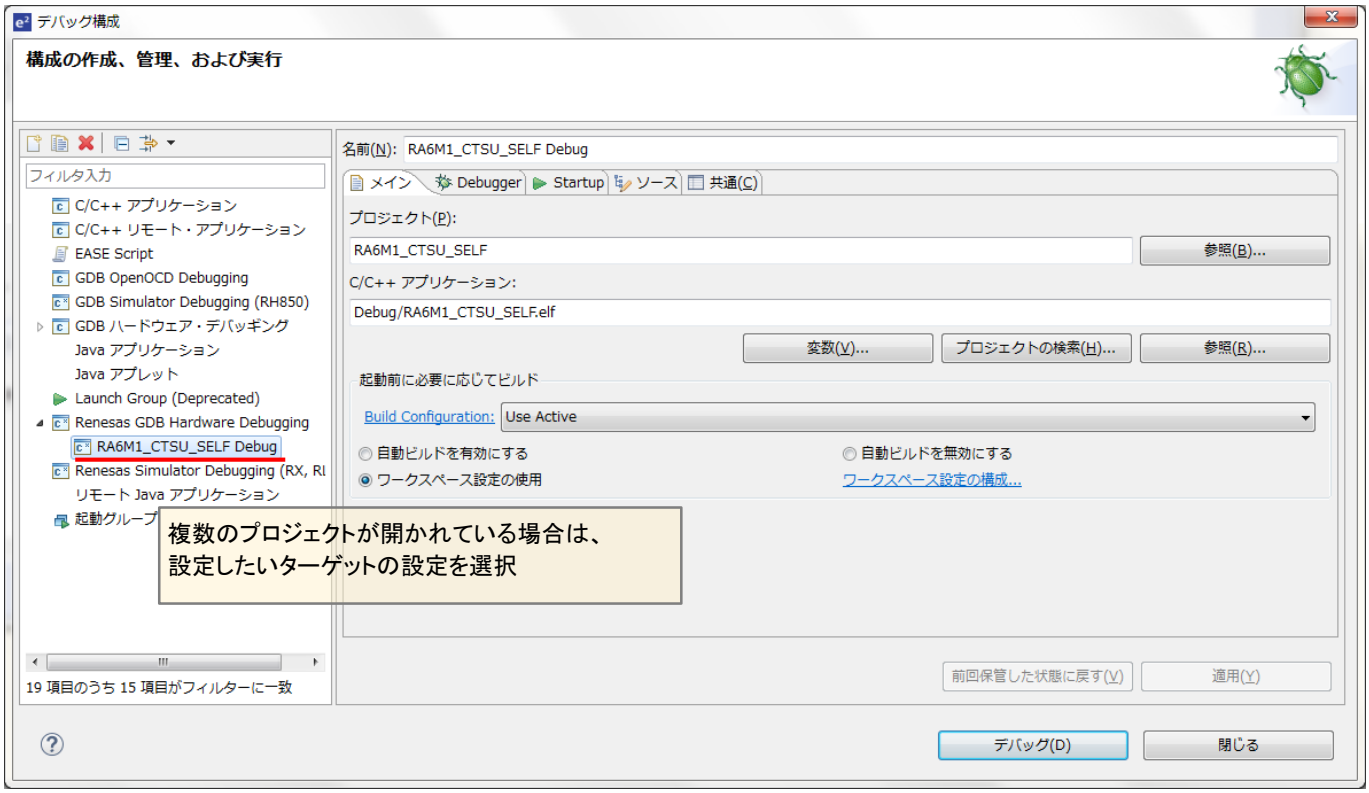
ワークスペースフォルダの下のプロジェクト名フォルダの下の Debug フォルダ内に、「プロジェクト名.srec」というファイルで生成されます。このファイルをマイコンボードに書き込んでください。

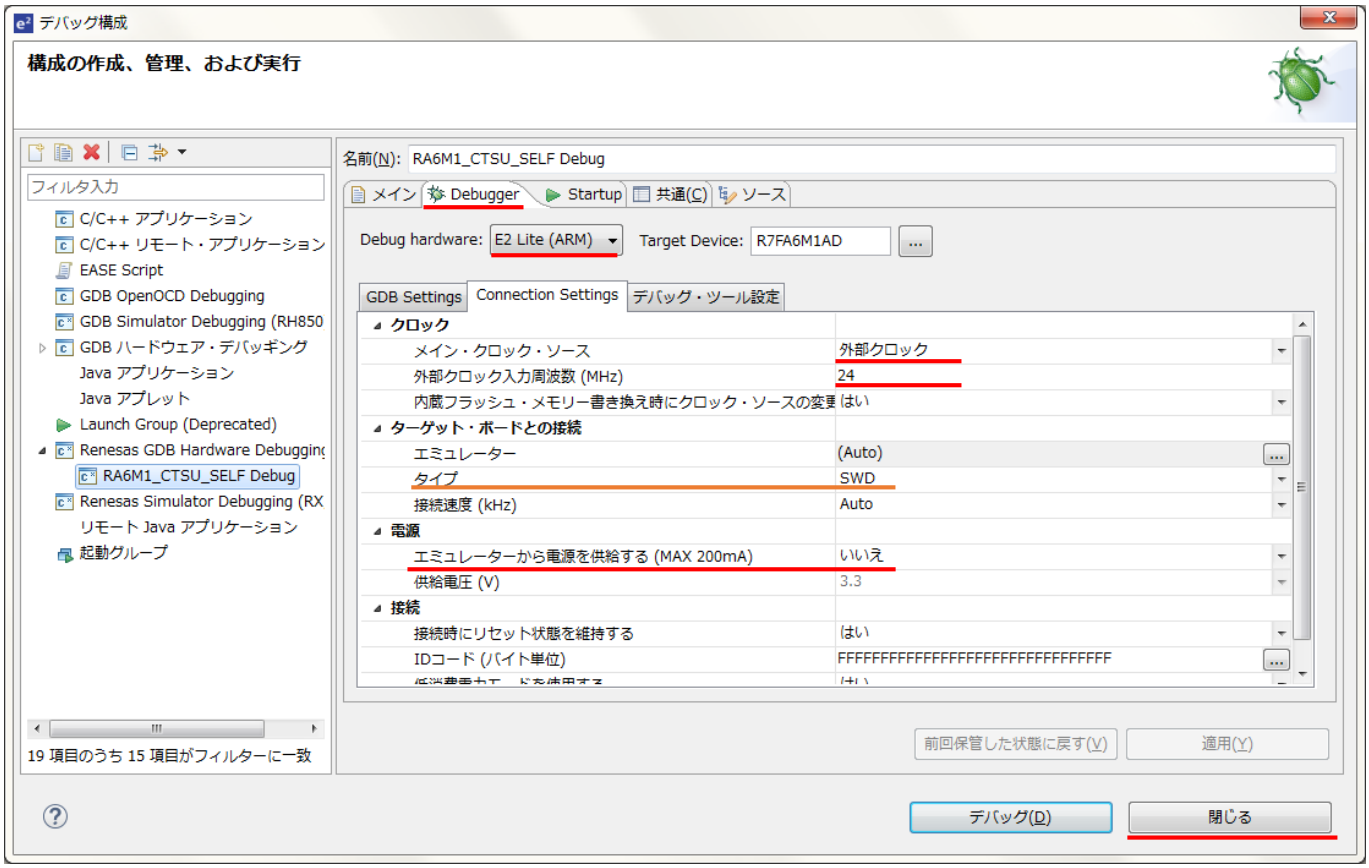
※RA のデフォルトは拡張子".srec"である事に注意ください（中身は、.mot ファイルと同じ形式です）

(6)デバッガ接続（オプション）



プロジェクトを右クリック、デバッグ ー でバックの構成





Debugger タブ

Debug hardware　お手持ちのデバッガを選択

メインクロックソース 「外部クロック」

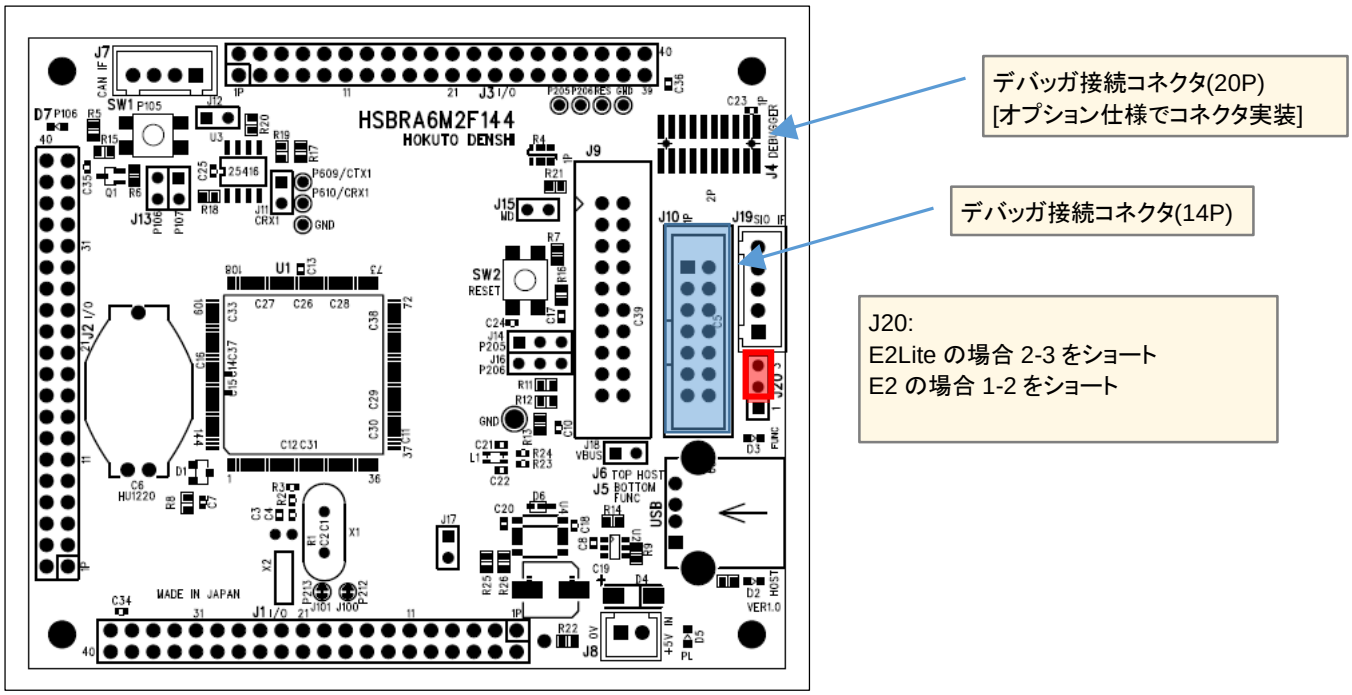
外部クロック入力周波数(MHz) 「24」

エミュレータから電源を供給する(MAX 200mA) **「いいえ」** ※はいとすると接続エラーとなります

エミュレータの　タイプ　は、E2Lite の場合は「SWD」、E2 の場合は「JTAG」「SWD」のどちらかとなります。

デバッグボタンを押すと、デバッガ接続、閉じるを押すと設定保存となります。

E2, E2Lite の場合デバッガをマイコンボードの 14P コネクタ（または、USB-ADAPTER-RX14 の 14P コネクタ）に接続。デバッガ選択ジャンパを、接続するデバッガに応じて選択。

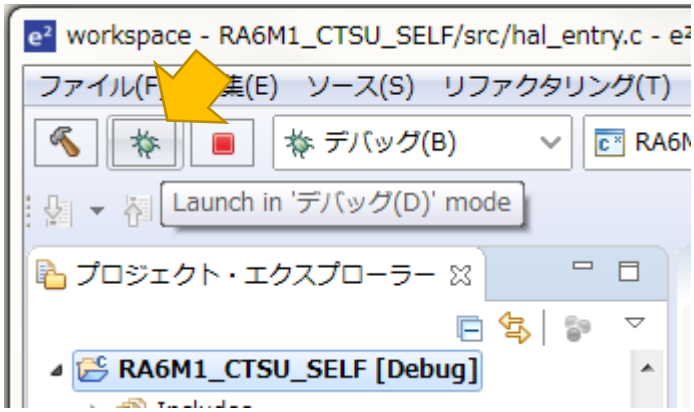


※HSBRA6M1F100 の場合は、J20→J16 となりますが、ジャンパの場所は同じです

※オプションの 20P コネクタを使用する場合は、ジャンパ設定は不要です

ビルド後、

デバッガ接続

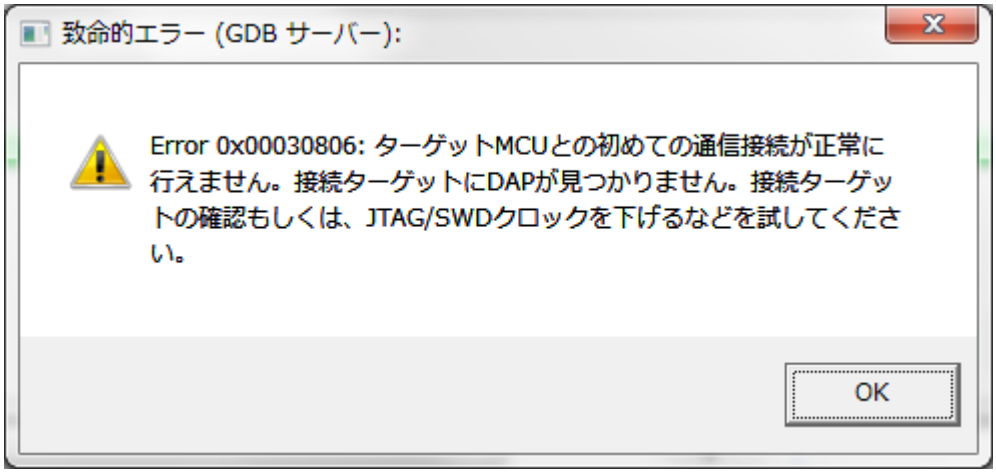


虫のアイコンをクリック

で、マイコンボードに対して、プログラムの書き込み（及びデバッグ）を行う事ができます。

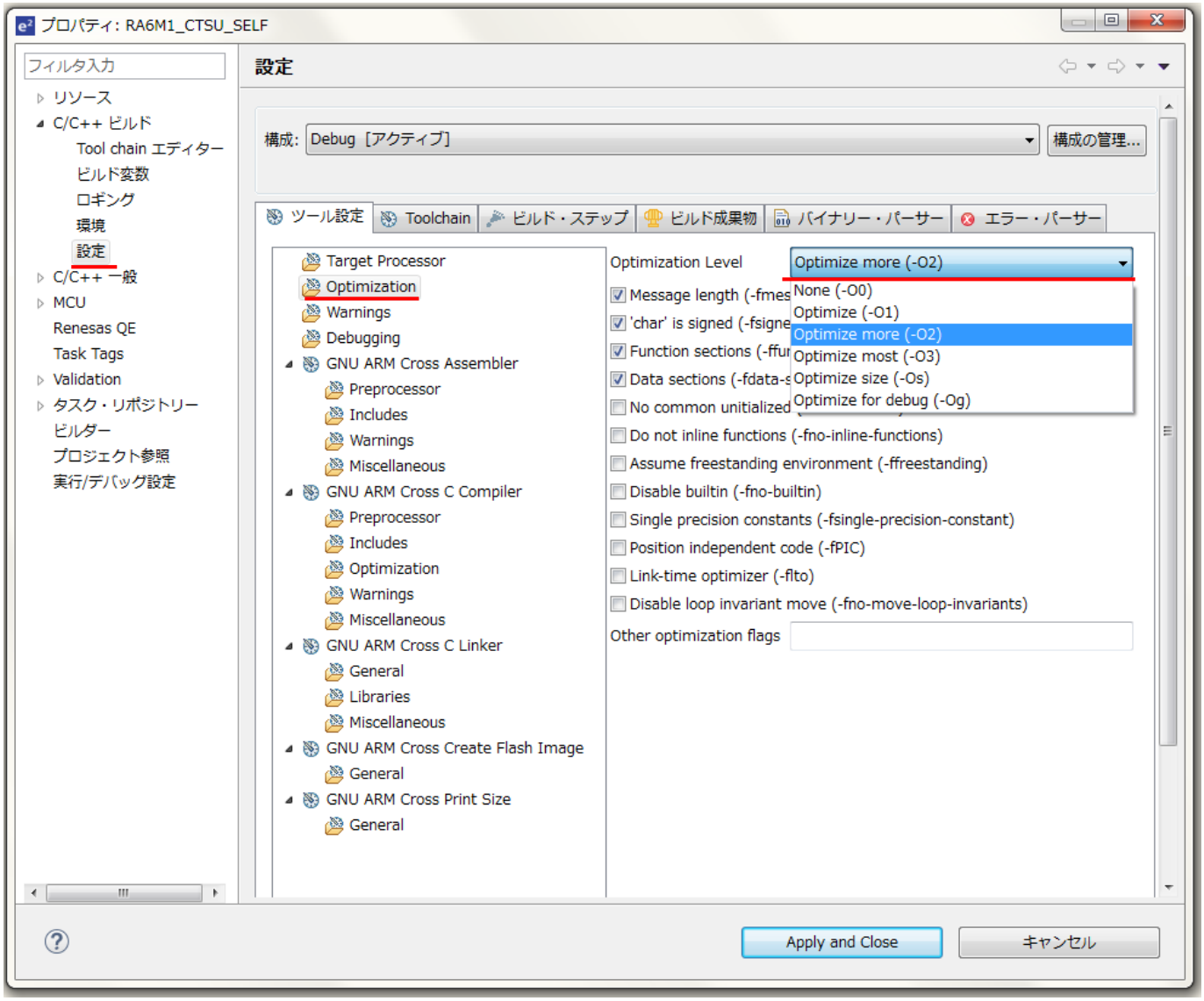
デバッガをお持ちであれば、RenesasFlashProgrammer を使用した書き込みより、こちらの方が手早く行えるかと考えます。

[参考]



デバッガ接続時、上記のエラーが出た場合は、HSBRA6M2F144:J20, HSBRA6M1F100:J16 のジャンパを確認してください。

デバッグ時は、プロジェクトを右クリックして、プロパティを開き、



Optimization

Optimize Level

-O2（デフォルト）から(None) -O0
を選択した方がデバッグがやり易い場合もありますので、必要に応じて変更してください。

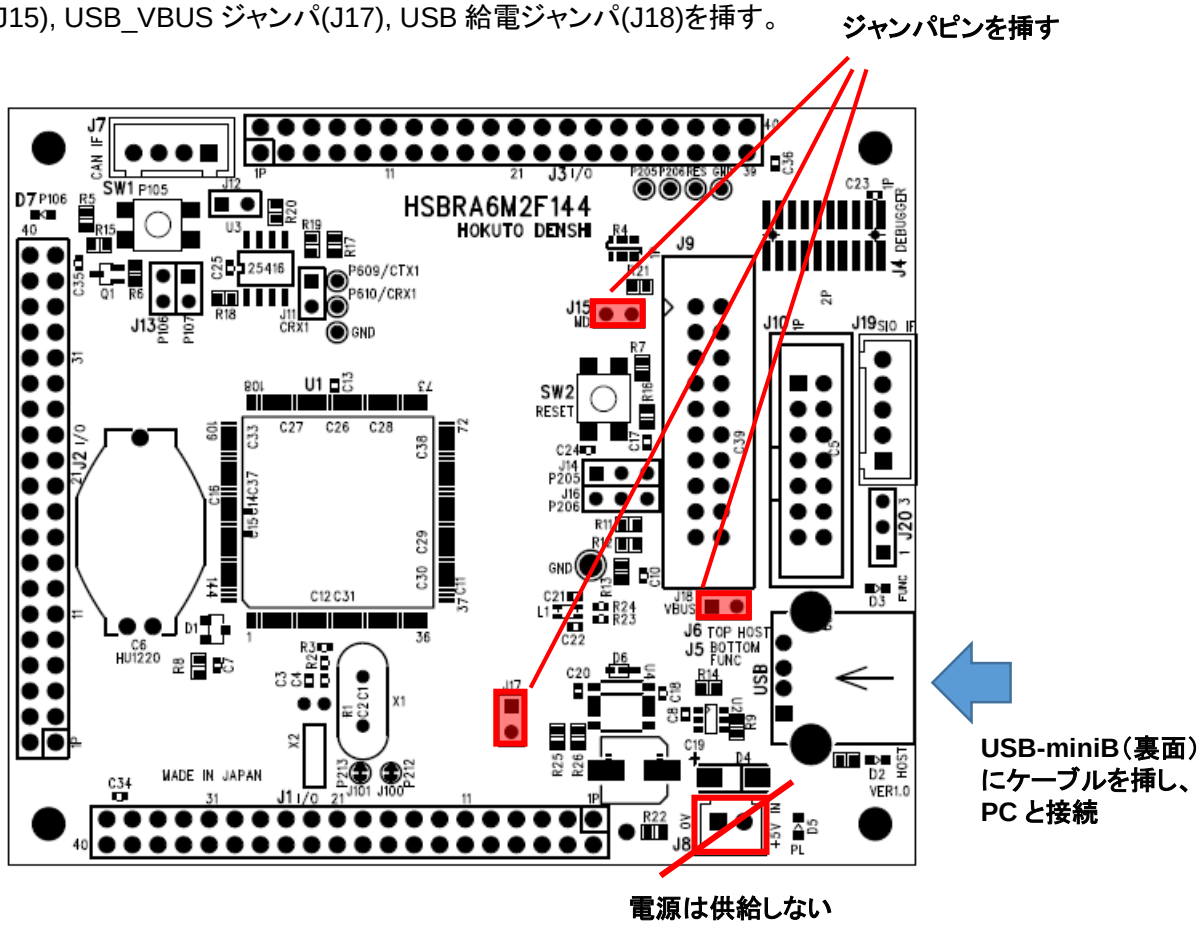
(7)プログラムの書き込み

ビルドしたプログラムをマイコンボードに書き込みを行って、プログラムを実行してください。デバッガ使用時は、デバッガ接続、プログラムをダウンロードした段階で、プログラムの書き込みは終了していますので、本ステップは不要です。プログラムの書き込みには、RenesasFlashProgrammer（以下 RFP）を使用しますので、予めダウンロード、インストールを行っておいてください。

※プログラムの書き込みは、USB-ADAPTER-RX14 を使う方法や、デバッガを使う方法等、何種類かあります。ここでは、USB ケーブルのみで書き込みを行う手順を示します。

(7-1)接続

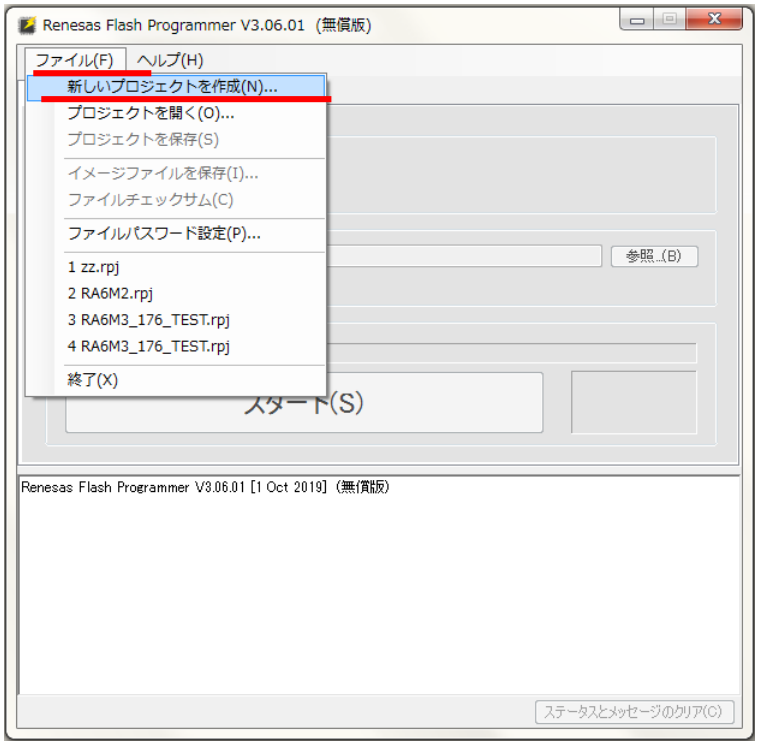
MD ジャンパ(J15), USB_VBUS ジャンパ(J17), USB 給電ジャンパ(J18)を挿す。



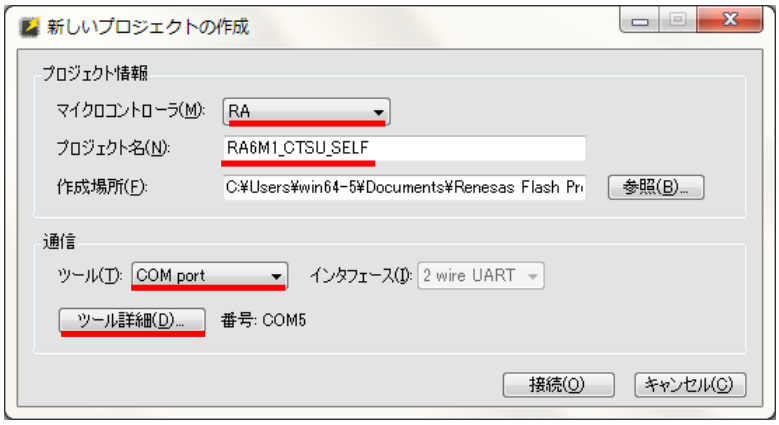
USB-miniB コネクタ（J5, 裏面）にケーブル（キット付属）を挿し、PC と接続する。
※ケーブルを挿した段階で PL(D5)が点灯し、ボードの電源が入るはずです。

※上記ボードレイアウトは、HSBRA6M2F144 のものですが、HSBRA6M1F100 でもジャンパの位置等は同一です

(7-2)RFP の起動とマイコンボードへの接続



RFP を起動し、
ファイルー新しいプロジェクトを作成



・マイクロコントローラ
RA
を選択する。

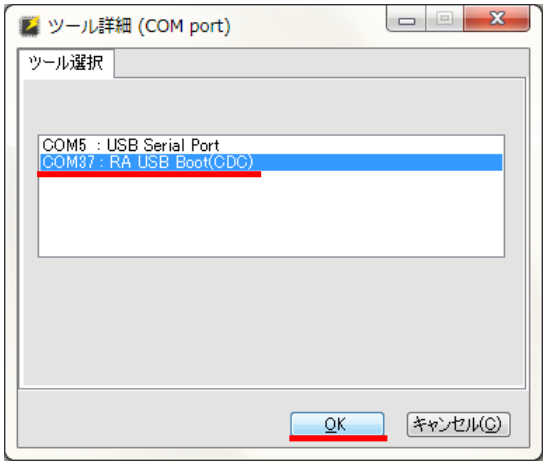
・プロジェクト名
任意の名称を入力

・ツール

COM port

に変更してください

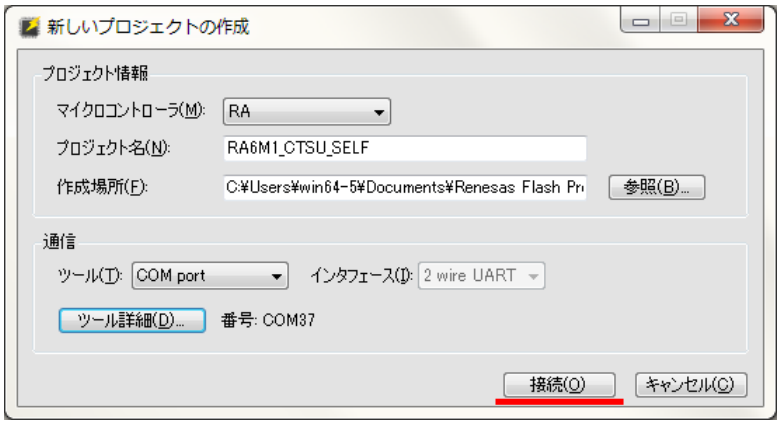
・ツール詳細



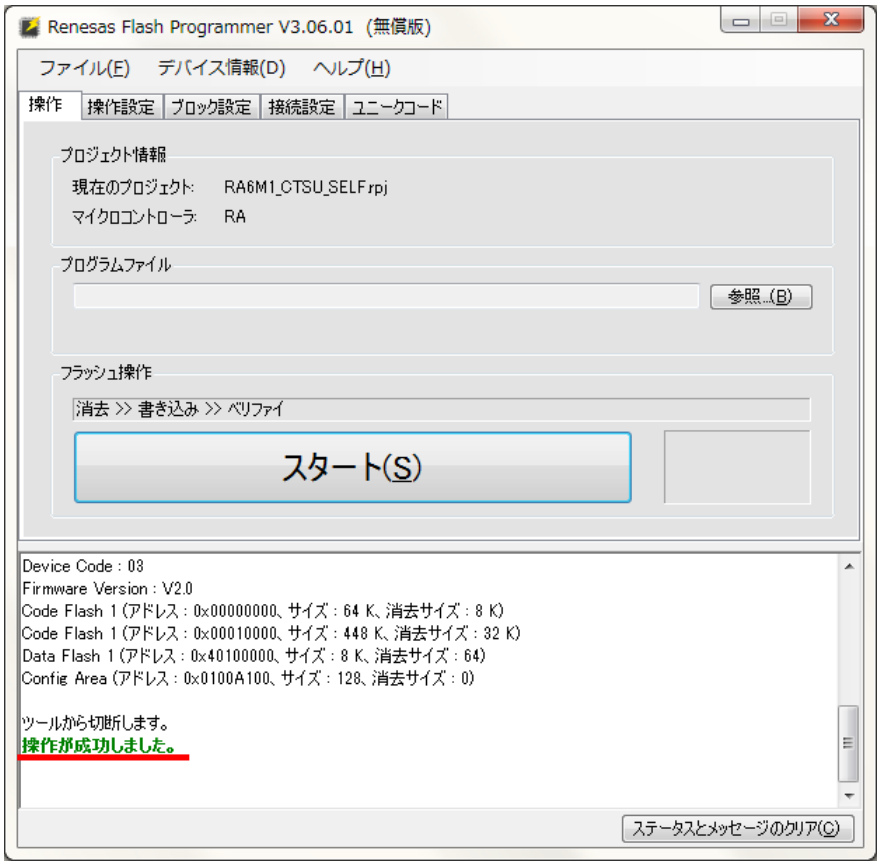
COM*nn* RA USB Boot(CDC)

を選択してください（*nn* の部分は、環境によって異なります）

上記 COM ポートが見えない場合は、マイコンボードのリセットボタンを押してください。（それでも見えない場合は、接続を確認してください）

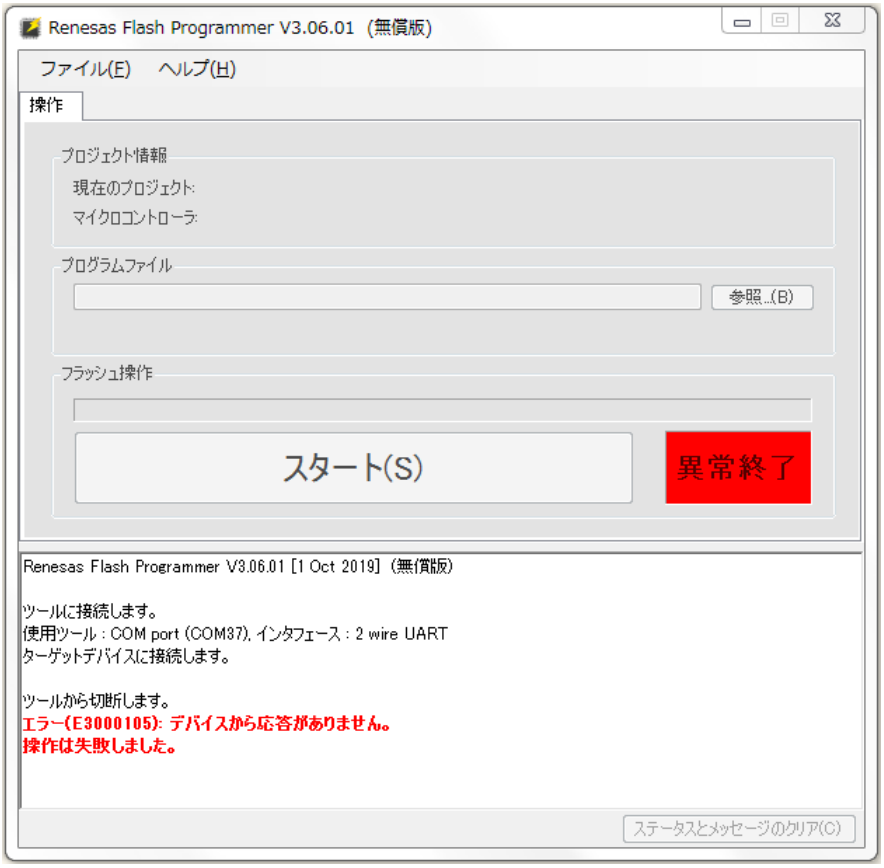


接続を押す



接続が成功しました、という表示となれば OK です。

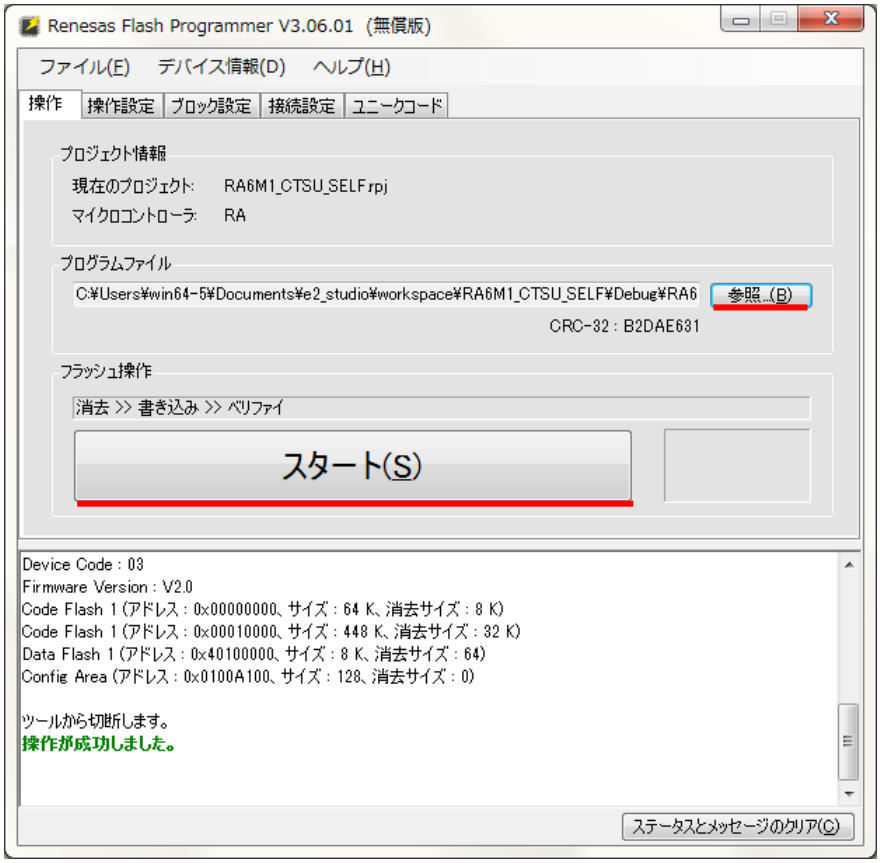
－エラーの場合－



上記の様なエラーとなった場合は、ボード上のリセットスイッチ(SW2)を押して、再度「新しいプロジェクトの作成ウィンドウの"接続"ボタン」を押してください。

なお、「**ツールとの接続に失敗しました**」というエラーの場合は、当該 COM ポート（上記例では、COM37）で端末が開いていないかを確認してください。端末が開いている場合は、その端末を閉じてください。

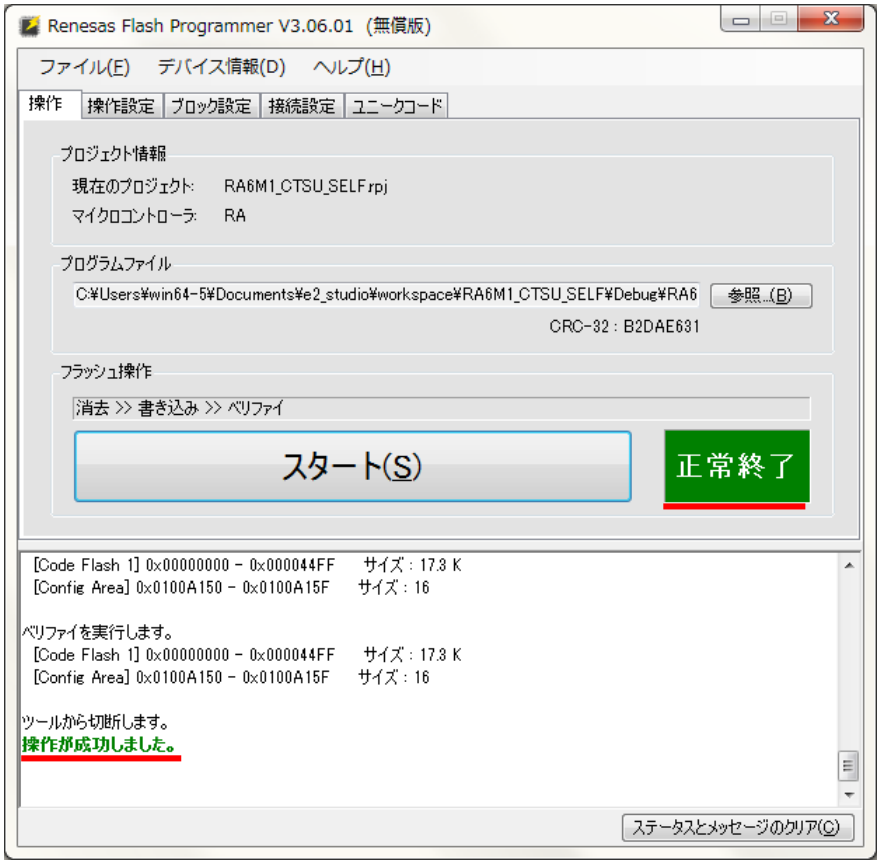
(7-3)プログラムの書き込み



参照ボタンを押し、書き込む MOT ファイル(.srec)を選択してください。
（[Workspace folder]¥*RA6M1_CTSU_SELF*¥Debug¥*RA6M1_CTSU_SELF*.srec）

マイコンボード上のリセットスイッチ(SW2)を押してください。（もしくは、USB ケーブルを一旦抜いて挿し直してください）

スタートボタンを押してください。



正常終了、「操作が成功しました」と表示されれば OK です。

マイコンボードにプログラムの書き込みは成功しています。

エラー(「デバイスから反応がありません」)となった場合は、ボード上のリセットスイッチ(SW2)を押して、再度スタートボタン」を押してください。

2. サンプルプログラムの動作

2.1. 自己容量タイプタッチキー基板

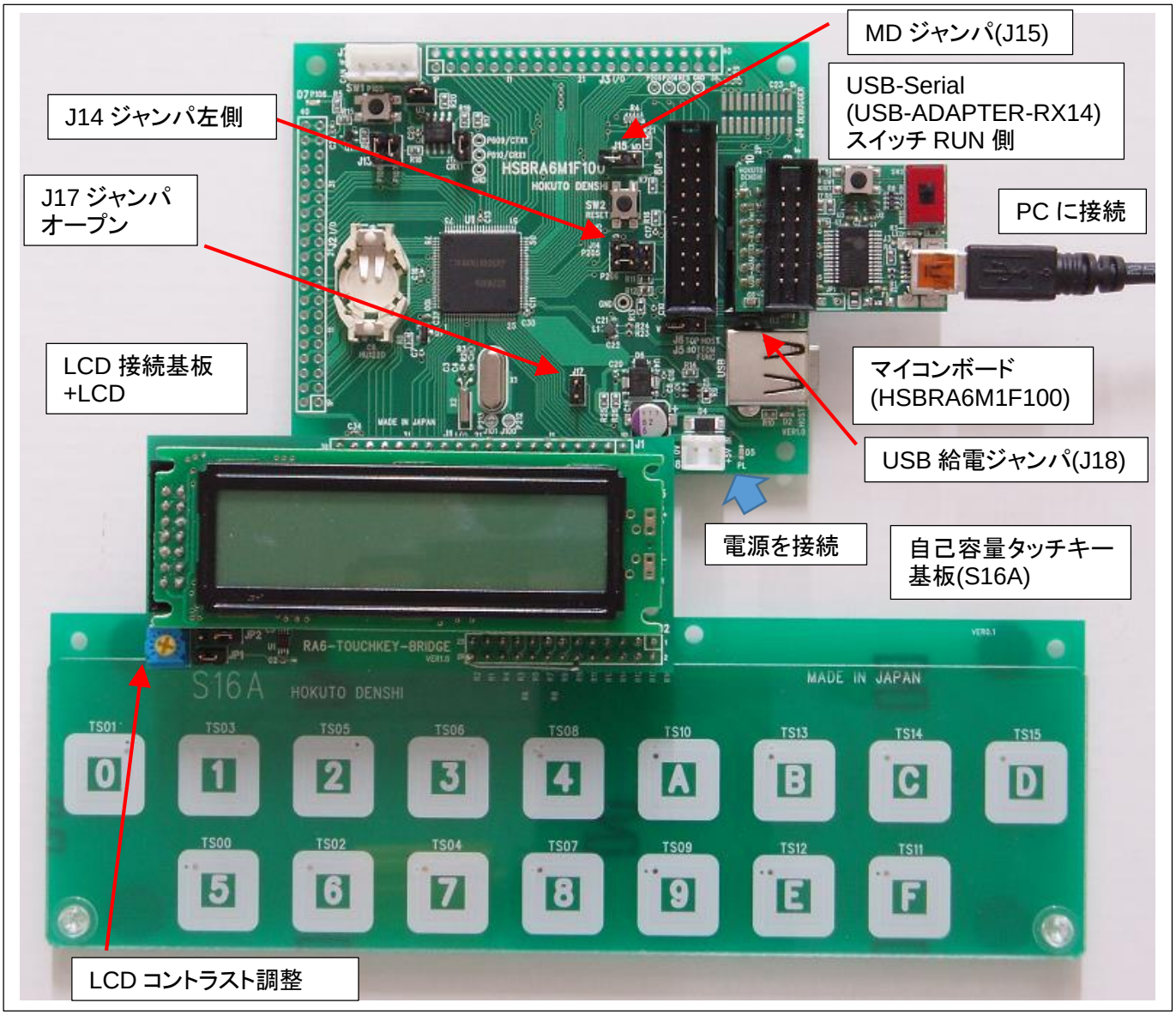


図 2-1 接続例(自己容量タイプタッチキー基板)

書き込みを行った設定に対し、MD ジャンパ(J15), USB_VBUS ジャンパ(J17), USB 給電ジャンパ(J18)を抜いてください。(上の写真では、ジャンパピンをずらして挿しています)

J14 P205 のジャンパは左側がショートとなる様に設定してください。

マイコンボードの電源コネクタ(J8)に、電源(+5V)を接続してください。

以下、自己容量タイプのプログラムが書かれている場合の動作を説明します。

LCD 画面に、文字が表示されるはずですので、見易くなるよう LCD コントラスト調整の可変抵抗を調整してください。

LCD 画面に以下の表示が出力されます。

HSBRA6 Self Cap offset otimize.	初期化中ですので、キーには触れないでください
HSBRA6 Self Cap >-	キー入力待ちですので、キーパッドに触れてください

キーパッド 0 に触れた場合

HSBRA6 Self Cap >0(TS11)	キーに触れている間、触れているキーが表示されます

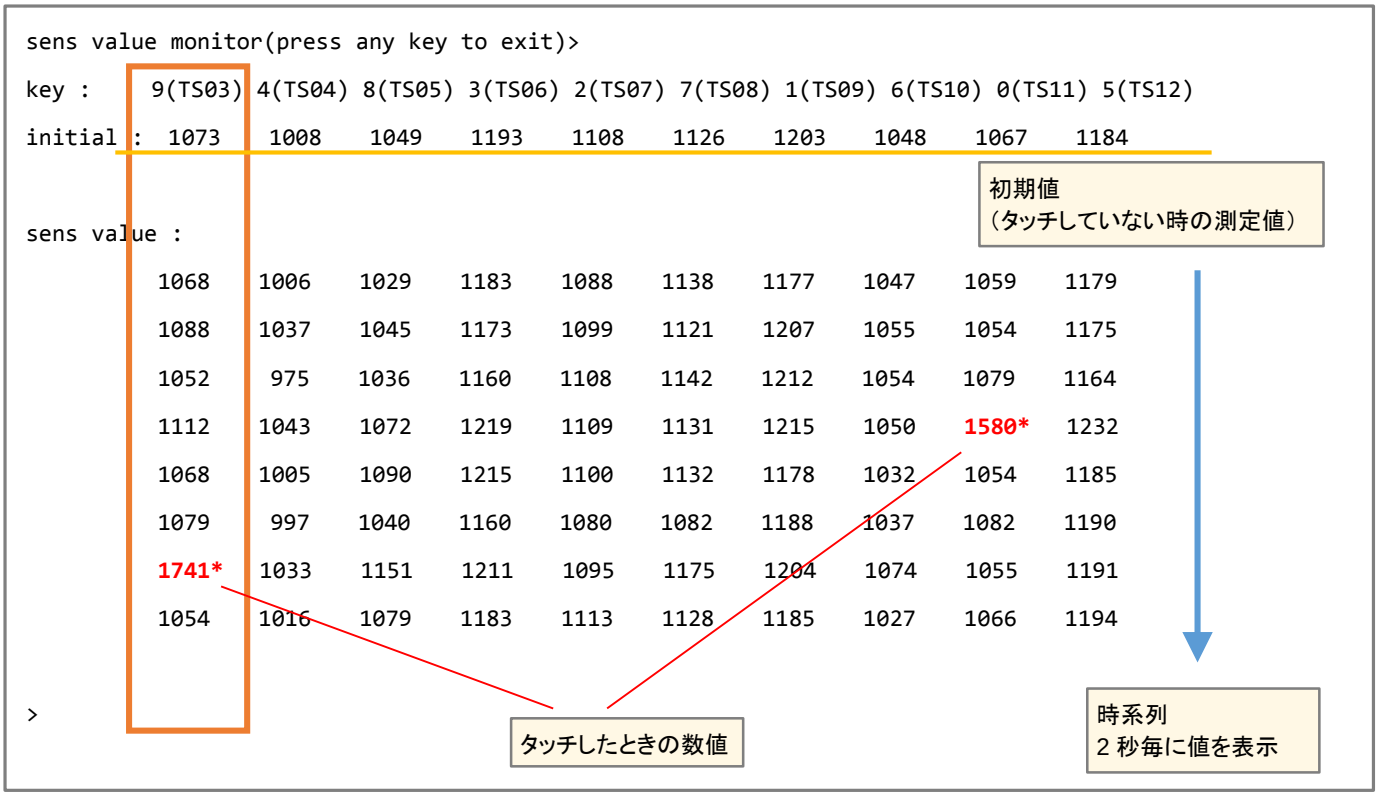
USB-ADAPTER-RX14 と PC を接続し、仮想端末（teraterm や putty 等）を、115,200bps で開いてください。

```
Copyright (C) 2020 HokutoDenshi. All Rights Reserved
HSBRA6xxxx Self Cap touchkey sample program boot!

Please do NOT touch key pad!

offset optimize & no-touch data scaning...

（中略）

initial sens value measure...

initialize finished.

TEST MENU:
  m : sens value monitor
  t : tuning key parameter
  p : print initial reference value
  s : statistics data print
  i : re-initialize
>
```

端末には、上記メッセージが表示されます。キーボードからコマンドを入力する事により、数値のモニタ等が行えます。

・m コマンド（数値モニタ）

キーボードから、小文字"m"を入力すると、以下の表示となります。



2 秒毎に、そのときの測定値が表示されます。

key: 9(TS03) は、タッチキー基板の"9", マイコンの信号は TS03 です。
（並び順が、TSxx になっており、キーパッドの順番でない事に注意願います）

initial: 初期（キーにタッチしていないとき）の測定値です

sens value: 現時点の測定値です。2 秒毎に値が表示されます。数値の後に(*)が付いているものは、タッチしていると判断されているキーです。

測定値は、電極の容量に対応しています。指でキーパッドに触れると、容量値が増加し、測定値が増加します。

キーボードから何か適当なキーを入力すると、表示は終了します。

・t コマンド（閾値のチューニング）

キーを押したかどうかの判定は、初期値（タッチしていない場合の測定値）から 20%値が増えた場合としています。但し、この閾値は暫定値で、実際にタッチを行った際の増分が反映されていません。t コマンドは、実際にタッチしたときの値を測定し、タッチ／非タッチの閾値を最適化します。

キーボードから、小文字"t"を入力すると、以下の表示となります。

```
threshold value optimize...

Please touch key pad ->0
& PC keyboard enter

touch data scaning...

 key scan end.

  key pad(TS ch) = 0(TS11)

  touch(ave) = 1671 no-touch(ave) = 1067


Please touch key pad ->1
& PC keyboard enter

[中略]

Please touch key pad ->9
& PC keyboard enter

touch data scaning...

 key scan end.

  key pad(TS ch) = 9(TS03)

  touch(ave) = 1673 no-touch(ave) = 1073


 tuning end.


>
```

この表示の後、キーパッド"0"に触れ、キーボードの[Enter]を押す

この表示の後、キーパッドから指を離して OK です

タッチ時と非タッチ時の測定値が表示されます

以下、1~9 まで同様に、キーパッドに触れ[Enter]を押す

全てのキーのチューニング（閾値）の更新が終わりました

t コマンド実行前は、一律 20%増加点を閾値としていますが、t コマンド実行後はタッチ／非タッチの中間を閾値に設定します。

※t コマンド実行後でも見かけ上の動作は変わりませんが、タッチ判定を行う閾値は変更されています

・p コマンド（初期値の表示）

タッチキーの動作を左右するパラメータは、起動時に値を最適化しています（詳細は後述）。その値と、現在の閾値を表示します。

キーボードから、小文字"p"を入力すると、以下の表示となります。

```
print initial reference value>

TSxx   : CTSURICOA, CTSUSO :  Sens  :  threshold
9(TS03):        45,    350 :  1073  : 1.279
4(TS04):        45,    341 :  1008  : 1.250
8(TS05):        45,    379 :  1049  : 1.269
3(TS06):        50,    367 :  1193  : 1.202
2(TS07):        50,    388 :  1108  : 1.212
7(TS08):        50,    389 :  1126  : 1.250
1(TS09):        50,    396 :  1203  : 1.196
6(TS10):        45,    407 :  1048  : 1.262
0(TS11):        45,    378 :  1067  : 1.283
5(TS12):        50,    403 :  1184  : 1.248

>
```

CTSURICOA:　リファレンス ICO の入力電流(*1)
CTSUSO:　電流オフセット量(*1)
Sens:　非タッチ時の測定値（m コマンドの initial と同じ値）
threshold:　判定閾値（t コマンド実行前は、1.2 です）

(*1)起動時にプログラムで最適化されています。詳細は後述しますが、タッチキー機能(CTSU)を使用する上で、感度や測定誤差に関わるパラメータです。

・s コマンド（統計データの表示）

30 秒間測定を行い、最大値やばらつき等を表示します（値の表示は、"s"を入力してから 30 秒後となります）。

キーボードから、小文字"s"を入力すると、以下の表示となります。

```
sens statistics(measure 30 sec)>

sens value statistics data

key(TSxx) :   Ave (   Max -   Min , 3sigma )
9(TS03)   :  1083 (  1136 -  1034 , 49 )
4(TS04)   :  1016 (  1078 -   961 , 51 )
8(TS05)   :  1066 (  1160 -   998 , 57 )
3(TS06)   :  1202 (  1263 -  1138 , 53 )
2(TS07)   :  1113 (  1177 -  1043 , 52 )
7(TS08)   :  1172 (  1762 -  1083 , 379 )
1(TS09)   :  1208 (  1290 -  1149 , 52 )
6(TS10)   :  1085 (  1165 -  1024 , 60 )
0(TS11)   :  1070 (  1127 -  1008 , 52 )
5(TS12)   :  1781 (  1843 -  1724 , 52 )

>
```

ここで 30 秒待ちが入ります
30 秒後に以下を表示

1 回（1 秒程度）タッチしたキー

30 秒間タッチしていたキー

Ave は 30 秒間の平均値。Max, Min は最大、最小値。3sigma は、偏差の 3 倍です（いわゆる 3σ）。

上記は

5 のキー　30 秒間タッチ
7 のキー　1 秒程度タッチ
他のキー　非タッチ

の場合の、表示例です。

5 のキーは、測定値の平均は 1781（"p","m"コマンドで取得できる非タッチ時の値）の 1184 より、大きな値となっています。但し、30 秒の測定中ずっとタッチしていたため、測定値のばらつき（3σ値）はタッチしていない他のキーに対し大きくありません。

7 のキーは、最大－最小の差分が大きく、3σ値も大きな値が出ています。これは、「タッチ」「非タッチ」が混在する測定値の統計のためです。

このコマンドは、「タッチ」または「非タッチ」時に、どの程度測定値にばらつきが出るのかをモニタする事が可能です。

なお、測定値は 1/64 秒毎にサンプリングされています（30 秒で、1920 データ）。

・i コマンド（再初期化）

起動時の処理化を再度行います。（マイコンをリセットした場合と等価です）

キーボードから、小文字"i"を入力すると、以下の表示となります。

```
re-initialize(please do not touch key)>
ref current, current offset optimize.

CTSURICOA=0

CTSURICOA=5

CTSURICOA=10

CTSURICOA=15

CTSURICOA=20

CTSURICOA=25

CTSURICOA=30

CTSURICOA=35

CTSURICOA=40
  4(TS04) calibration finish.

CTSURICOA=45
  9(TS03) calibration finish.
  8(TS05) calibration finish.
  3(TS06) calibration finish.
  2(TS07) calibration finish.
  6(TS10) calibration finish.
  5(TS12) calibration finish.

CTSURICOA=50
  7(TS08) calibration finish.
  1(TS09) calibration finish.
  0(TS11) calibration finish.
initial sens value measure...
initialize finished.


>
```

再初期化中は、キーパッドにタッチしないでください

CTSURICOA の値を増分させながら調整値を検索（詳細は後述）

キー4 に関しては、調整終了

全てのキーで調整終了するとコマンド受付状態に戻ります

"i"コマンド実行後は、閾値も初期値に戻りますので、必要に応じて再度"t"コマンドで調整してください。

"i"コマンド実行前後の"p"コマンドのモニタ値から、経時変化や温度や環境変化による初期値の違いを確認することができます。

2.2. 相互容量タイプタッチキー基板

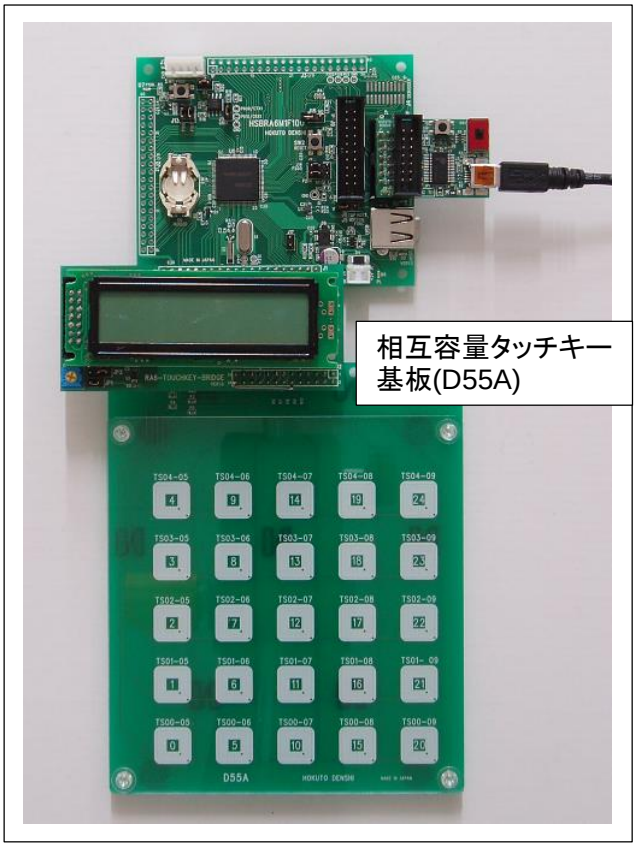


図 2-2 接続例（相互容量タイプタッチキー基板）

タッチキー基板を相互容量タイプ(D55A)に差し替える他は、自己容量タイプのタッチキー基板を使用する場合と変わりません。

マイコンボードに書き込むプログラムは、

SOURCE¥RA_CTSU_MUTUAL¥source¥ctsu

を含むプロジェクトで生成したものとしてください。（自己容量タイプとは異なるプログラムとなります）

マイコンボードに電源を投入して起動すると、LCD 画面に以下の表示が出力されます。

HSBRA6 MutualCap offset otimize.	初期化中ですので、キーには触れないでください
HSBRA6 MutualCap >-	キー入力待ちですので、キーパッドに触れてください

キーパッド 0 に触れた場合

HSBRA6 MutualCap 0	キーに触れている間、触れているキーが表示されます

USB-ADAPTER-RX14 と PC を接続し、仮想端末を開くと以下のメッセージが表示されます。

```
Copyright (C) 2020 HokutoDenshi. All Rights Reserved
HSBRAxxxx Mutual Cap touchkey sample program boot!

Please do NOT touch key pad!

offset optimize & no-touch data scaning...

initial sens value measure...

initialize finished.

TEST MENU:
  m : sens value monitor
  t : tuning key parameter
  p : print initial reference value
  s : statistics data print
  i : re-initialize
>
```

端末には、上記メッセージが表示されます。キーボードからコマンドを入力する事により、数値のモニタ等が行えます。

・m コマンド（数値モニタ）

キーボードから、小文字"m"を入力すると、以下の表示となります。

```
sens value monitor(press any key to exit)>
--
ch/key        : init(1) init(2) init_diff : sens(1) sens(2) sens_diff : rate
ch= 0, key=24 :    1533    5705   ( 4172) :    1467    5663   ( 4211) : 1.009
ch= 1, key=23 :    1539    5298   ( 3759) :    1474    5283   ( 3790) : 1.008
ch= 2, key=22 :    1537    5267   ( 3730) :    1529    5188   ( 3683) : 0.987
ch= 3, key=21 :    1538    5340   ( 3802) :    1544    5287   ( 3792) : 0.997
ch= 4, key=20 :    1512    5134   ( 3622) :    1492    5085   ( 3602) : 0.994
ch= 5, key=19 :    1531    6089   ( 4558) :    1471    6070   ( 4573) : 1.003
ch= 6, key=18 :    1554    5699   ( 4145) :    1465    5628   ( 4175) : 1.007
ch= 7, key=17 :    1537    5678   ( 4141) :    1485    5614   ( 4133) : 0.998
ch= 8, key=16 :    1545    5780   ( 4235) :    1505    5732   ( 4186) : 0.988
ch= 9, key=15 :    1517    5366   ( 3849) :    1484    5324   ( 3778) : 0.981
ch=10, key=14 :    1541    7692   ( 6151) :    1478    7653   ( 6176) : 1.004
ch=11, key=13 :    1548    5928   ( 4380) :    1508    5909   ( 4440) : 1.013
ch=12, key=12 :    1535    5838   ( 4303) :    1523    5790   ( 4304) : 1.000
ch=13, key=11 :    1531    5707   ( 4176) :    1496    5662   ( 4139) : 0.991
ch=14, key=10 :    1558    5235   ( 3677) :    1551    5190   ( 3664) : 0.996
ch=15, key= 9 :    1531    8570   ( 7039) :    1489    8561   ( 7118) : 1.011
ch=16, key= 8 :    1579    5939   ( 4360) :    1518    5904   ( 4429) : 1.015
ch=17, key= 7 :    1545    5876   ( 4331) :    1496    5783   ( 4316) : 0.996
ch=18, key= 6 :    1536    5716   ( 4180) :    1499    5691   ( 4155) : 0.994
ch=19, key= 5 :    1528    5267   ( 3739) :    1565    5184   ( 3638) : 0.972
ch=20, key= 4 :    1582   10098   ( 8516) :    1595   10110   ( 8519) : 1.000
ch=21, key= 3 :    1545    8633   ( 7088) :    1524    8610   ( 7112) : 1.003
ch=22, key= 2 :    1501    7680   ( 6179) :    1518    7680   ( 6151) : 0.995
ch=23, key= 1 :    1512    6084   ( 4572) :    1565    6037   ( 4437) : 0.970
ch=24, key= 0 :    1532    5709   ( 4177) :    1897    5416   ( 3451) : 0.826*

>
```

2 秒毎に値を表示

初期値
（タッチしていない時の測定値）

現在の測定値

タッチしたときの表示例

2 秒毎に、そのときの測定値が表示されます。

ch/key:　ch=24, key=0 は、タッチキー基板の"0", マイコンの測定チャネル番号は 24（TS12→TS07 の交点）です。（TSxx の若い順にチャネル番号が割り振られ、キーパッド(key)の順番とチャネル番号(ch)は逆の関係となります）

init(1):　初期（キーにタッチしていないとき）の同相の測定値です
init(2):　初期（キーにタッチしていないとき）の逆相の測定値です
init_diff:　初期の逆相と同相の測定値の差分です

sens(1):　現在の同相の測定値です
sens(2):　現在の逆相の測定値です
sens_diff:　現在の逆相と同相の測定値の差分です

rate:　現在の差分 / 初期値の差分 （割合）です。数値の後に(*)が付いているものは、タッチしていると判断されているキーです。

測定値は、電極の容量に対応しています。指でキーパッドに触れると、同相の測定値は増加し、逆相の測定値は減少します。判定は、同相と逆相の差分がどの程度変化したかで見ています。

キーボードから何か適当なキーを入力すると、表示は終了します。

・t コマンド（閾値のチューニング）

キーを押したかどうかの判定は、初期値（タッチしていない場合の測定値）から、差分が 5%値が減った場合としています。但し、この閾値は暫定値で、実際にタッチを行った際の増分が反映されていません。t コマンドは、実際にタッチしたときの値を測定し、タッチ／非タッチの閾値を最適化します。

キーボードから、小文字"t"を入力すると、以下の表示となります。

```
threshold value optimize...

Please touch key pad ->0
& PC keyboard enter

touch data scaning...

 key scan end.

  touch(ave) = 1939,5408 no-touch(ave) = 1532,5709  touch(diff) = 3469  no-touch(diff) = 4177


Please touch key pad ->1
& PC keyboard enter

（中略）

Please touch key pad ->24
& PC keyboard enter

touch data scaning...

 key scan end.

  touch(ave) = 2043,5529 no-touch(ave) = 1533,5705  touch(diff) = 3486  no-touch(diff) = 4172


 tuning end.

>
```

この表示の後、キーパッド"0"に触れ、
キーボードの[Enter]を押す

この表示の後、キーパッドから指を離して OK です

タッチ時と非タッチ時の測定値が表示されます

以下、1~24 まで同様に、キーパッドに触れ
[Enter]を押す

全てのキーのチューニング
（閾値）の更新が終わりました

t コマンド実行前は、一律 5%減少点を閾値としていますが、t コマンド実行後はタッチ／非タッチの中間を閾値に設定します。

・p コマンド（初期値の表示）

タッチキーの動作を左右するパラメータは、起動時に値を最適化しています（詳細は後述）。その値と、現在の閾値を表示します。

キーボードから、小文字"p"を入力すると、以下の表示となります。

```
print initial reference value>

ch/key        : CTSURICOA  CTSUSO  : sens(1) sens(2)  (diff)  : threshold
ch= 0, key=24 :         65     409     1533    5705 ( 4172) : 0.917
ch= 1, key=23 :         65     420     1539    5298 ( 3759) : 0.911
ch= 2, key=22 :         65     421     1537    5267 ( 3730) : 0.911
ch= 3, key=21 :         65     419     1538    5340 ( 3802) : 0.914
ch= 4, key=20 :         65     425     1512    5134 ( 3622) : 0.908
ch= 5, key=19 :         65     405     1531    6089 ( 4558) : 0.922
ch= 6, key=18 :         65     416     1554    5699 ( 4145) : 0.926
ch= 7, key=17 :         65     417     1537    5678 ( 4141) : 0.931
ch= 8, key=16 :         65     414     1545    5780 ( 4235) : 0.926
ch= 9, key=15 :         65     426     1517    5366 ( 3849) : 0.918
ch=10, key=14 :         65     363     1541    7692 ( 6151) : 0.945
ch=11, key=13 :         65     419     1548    5928 ( 4380) : 0.924
ch=12, key=12 :         65     422     1535    5838 ( 4303) : 0.927
ch=13, key=11 :         65     426     1531    5707 ( 4176) : 0.922
ch=14, key=10 :         65     439     1558    5235 ( 3677) : 0.920
ch=15, key= 9 :         65     329     1531    8570 ( 7039) : 0.961
ch=16, key= 8 :         65     419     1579    5939 ( 4360) : 0.930
ch=17, key= 7 :         65     422     1545    5876 ( 4331) : 0.930
ch=18, key= 6 :         65     427     1536    5716 ( 4180) : 0.924
ch=19, key= 5 :         65     440     1528    5267 ( 3739) : 0.921
ch=20, key= 4 :         65     234     1582   10098 ( 8516) : 0.972
ch=21, key= 3 :         65     327     1545    8633 ( 7088) : 0.962
ch=22, key= 2 :         65     368     1501    7680 ( 6179) : 0.948
ch=23, key= 1 :         65     419     1512    6084 ( 4572) : 0.924
ch=24, key= 0 :         65     430     1532    5709 ( 4177) : 0.915

>
```

CTSURICOA:　リファレンス ICO の入力電流(*1)

CTSUSO:　電流オフセット量(*2)

sens(1), sens(2), diff:　非タッチ時の同相、逆相の測定値とその差分（m コマンドの init と同じ値）

threshold:　判定閾値（t コマンド実行前は、0.95 です）

(*1)相互容量のプログラムでは、初期は固定値(65)が設定されています。後述の"i"コマンドで最適化されます。

(*2)起動時にプログラムで最適化されています。詳細は後述しますが、タッチキー機能(CTSU)を使用する上で、感度や測定誤差関わるパラメータです。

・s コマンド（統計データの表示）

30 秒間測定を行い、最大値やばらつき等を表示します（値の表示は、"s"を入力してから 30 秒後となります）。

キーボードから、小文字"s"を入力すると、以下の表示となります。

```
sens statistics(measure 30 sec)>

sens value statistics data

ch/key       :   same phase                      opposite phase                  diff
             :   Ave (   Max -   Min, 3sigma ) :   Ave (   Max -   Min, 3sigma ) :   Ave (   Max -   Min, 3sigma )
ch= 0, key=24 :  1511 (  1566 -  1459,     50 ) :  5707 (  5765 -  5652,     47 ) :  4195 (  4277 -  4111,     69 )
ch= 1, key=23 :  1534 (  1599 -  1462,     57 ) :  5296 (  5358 -  5220,     48 ) :  3761 (  3863 -  3664,     71 )
ch= 2, key=22 :  1520 (  1582 -  1455,     54 ) :  5196 (  5255 -  5138,     48 ) :  3676 (  3751 -  3573,     68 )
ch= 3, key=21 :  1512 (  1584 -  1454,     54 ) :  5276 (  5325 -  5215,     47 ) :  3764 (  3839 -  3686,     67 )
ch= 4, key=20 :  1551 (  1610 -  1478,     56 ) :  5101 (  5156 -  5029,     47 ) :  3549 (  3640 -  3466,     70 )
ch= 5, key=19 :  1548 (  1607 -  1484,     55 ) :  6117 (  6175 -  6068,     45 ) :  4568 (  4655 -  4490,     69 )
ch= 6, key=18 :  1591 (  1656 -  1523,     57 ) :  5746 (  5801 -  5698,     47 ) :  4155 (  4249 -  4059,     70 )
ch= 7, key=17 :  1528 (  1595 -  1467,     58 ) :  5649 (  5698 -  5592,     46 ) :  4120 (  4203 -  4042,     70 )
ch= 8, key=16 :  1565 (  1628 -  1487,     57 ) :  5767 (  5815 -  5709,     47 ) :  4202 (  4295 -  4130,     68 )
ch= 9, key=15 :  1570 (  1631 -  1505,     57 ) :  5363 (  5417 -  5313,     47 ) :  3792 (  3864 -  3726,     69 )
ch=10, key=14 :  1611 (  1678 -  1536,     54 ) :  7750 (  7796 -  7692,     42 ) :  6139 (  6219 -  6072,     67 )
ch=11, key=13 :  1572 (  1649 -  1491,     69 ) :  5968 (  6024 -  5913,     47 ) :  4396 (  4488 -  4297,     79 )
ch=12, key=12 :  1558 (  1627 -  1491,     59 ) :  5845 (  5901 -  5789,     46 ) :  4286 (  4386 -  4210,     71 )
ch=13, key=11 :  1542 (  1603 -  1474,     59 ) :  5687 (  5732 -  5625,     46 ) :  4145 (  4229 -  4070,     71 )
ch=14, key=10 :  1588 (  1657 -  1522,     58 ) :  5199 (  5256 -  5141,     49 ) :  3610 (  3699 -  3516,     73 )
ch=15, key= 9 :  1639 (  1710 -  1581,     51 ) :  8636 (  8676 -  8597,     39 ) :  6996 (  7072 -  6930,     64 )
ch=16, key= 8 :  1645 (  1723 -  1555,     77 ) :  5998 (  6058 -  5926,     45 ) :  4352 (  4440 -  4247,     84 )
ch=17, key= 7 :  1604 (  1665 -  1539,     57 ) :  5904 (  5954 -  5856,     46 ) :  4300 (  4374 -  4232,     71 )
ch=18, key= 6 :  1711 (  1774 -  1641,     57 ) :  5758 (  5805 -  5708,     46 ) :  4047 (  4136 -  3963,     69 )
ch=19, key= 5 :  1981 (  2037 -  1906,     54 ) :  5104 (  5156 -  5053,     49 ) :  3123 (  3202 -  3049,     67 )
ch=20, key= 4 :  1610 (  1662 -  1553,     52 ) : 10113 ( 10143 - 10082,     25 ) :  8503 (  8560 -  8443,     56 )
ch=21, key= 3 :  1557 (  1640 -  1464,     83 ) :  8641 (  8684 -  8595,     40 ) :  7084 (  7188 -  6972,     93 )
ch=22, key= 2 :  1542 (  1622 -  1457,     76 ) :  7688 (  7743 -  7639,     44 ) :  6146 (  6239 -  6066,     86 )
ch=23, key= 1 :  1597 (  1669 -  1518,     71 ) :  6077 (  6126 -  6027,     46 ) :  4480 (  4575 -  4395,     80 )
ch=24, key= 0 :  1676 (  1741 -  1601,     58 ) :  5618 (  5671 -  5565,     48 ) :  3942 (  4015 -  3872,     73 )

>
```

ここで 30 秒待ちが入ります
30 秒後に以下を表示

key=5　30 秒間タッチしていたキー

same phase は、同相の測定値。oppsite phase は、逆相の測定値。diff は差分です。
Ave は 30 秒間の平均値。Max, Min は最大、最小値。3sigma は、偏差の 3 倍です（いわゆる 3σ）。

上記は

5 のキー　30 秒間タッチ
他のキー　非タッチ

の場合の、表示例です。

5 のキーは、差分の平均は 3123（"p","m"コマンドで取得できる非タッチ時の値）の 3739 より、小さな値となっています。但し、30 秒の測定中ずっとタッチしていたため、測定値のばらつき（3σ値）はタッチしていない他のキーに対し大きくありません。（なお、相互容量タイプの場合は、押しているキー以外の値も変動します）

このコマンドは、「タッチ」または「非タッチ」時に、どの程度測定値にばらつきが出るのかをモニタする事が可能です。

なお、測定値は 1/64 秒毎にサンプリングされています（30 秒で、1920 データ）。

・i コマンド（再初期化）

起動時の処理化を再度行います。（本コマンドでは、CTSURICOA 値の最適も行いますので、通常の起動に比べ時間が掛かります。※相互容量タイプでは、起動時は CTSURICOA=65 に設定しています。）

キーボードから、小文字"i"を入力すると、以下の表示となります。

```
re-initialize(please do not touch key)>

Please do NOT touch key pad!

offset optimize & no-touch data scaning...

CTSURICOA=0

CTSURICOA=5

CTSURICOA=10

CTSURICOA=15

CTSURICOA=20

CTSURICOA=25

CTSURICOA=30

CTSURICOA=35

CTSURICOA=40

CTSURICOA=45
  ch = 0 calibration finish.
  ch = 20 calibration finish.

CTSURICOA=50
  ch = 7 calibration finish.
  ch = 15 calibration finish.
```

再初期化中は、キーパッドにタッチしないでください

CTSURICOA の値を増分させながら
調整値を検索（詳細は後述）

ch0,20 に関しては、調整終了

```
CTSURICOA=55
  ch = 1 calibration finish.
  ch = 2 calibration finish.
  ch = 3 calibration finish.
  ch = 4 calibration finish.
  ch = 5 calibration finish.
  ch = 6 calibration finish.
  ch = 8 calibration finish.
  ch = 9 calibration finish.
  ch = 10 calibration finish.
  ch = 12 calibration finish.
  ch = 14 calibration finish.
  ch = 17 calibration finish.
  ch = 18 calibration finish.
  ch = 19 calibration finish.
  ch = 24 calibration finish.

CTSURICOA=60
  ch = 13 calibration finish.

CTSURICOA=65
  ch = 11 calibration finish.
  ch = 16 calibration finish.
  ch = 21 calibration finish.
  ch = 22 calibration finish.
  ch = 23 calibration finish.

initial sens value measure...

initialize finished.


>
```

全ての ch で調整終了するとコマンド受付状態に戻ります

"i"コマンド実行前後の"p"コマンドのモニタ値から、経時変化や温度や環境変化による初期値の違いを確認することができます。

"i"コマンド後の、"p"コマンドの実行例。

```
print initial reference value>

ch/key       : CTSURICOA  CTSUSO : sens(1) sens(2)  (diff) : threshold
ch= 0, key=24 :        45     430 :    1056    5333 ( 4277) : 0.950
ch= 1, key=23 :        55     430 :    1305    5117 ( 3812) : 0.950
ch= 2, key=22 :        55     432 :    1325    5043 ( 3718) : 0.950
ch= 3, key=21 :        55     429 :    1324    5139 ( 3815) : 0.950
ch= 4, key=20 :        55     434 :    1322    4968 ( 3646) : 0.950
ch= 5, key=19 :        55     414 :    1311    5931 ( 4620) : 0.950
ch= 6, key=18 :        55     425 :    1316    5527 ( 4211) : 0.950
ch= 7, key=17 :        50     434 :    1145    5351 ( 4206) : 0.950
ch= 8, key=16 :        55     424 :    1305    5568 ( 4263) : 0.950
ch= 9, key=15 :        55     434 :    1337    5213 ( 3876) : 0.950
ch=10, key=14 :        55     370 :    1367    7593 ( 6226) : 0.950
ch=11, key=13 :        65     418 :    1552    5934 ( 4382) : 0.950
ch=12, key=12 :        55     431 :    1335    5661 ( 4326) : 0.950
ch=13, key=11 :        60     431 :    1432    5593 ( 4161) : 0.950
ch=14, key=10 :        55     449 :    1306    5024 ( 3718) : 0.950
ch=15, key= 9 :        50     342 :    1195    8413 ( 7218) : 0.950
ch=16, key= 8 :        65     421 :    1500    5883 ( 4383) : 0.950
ch=17, key= 7 :        55     431 :    1328    5690 ( 4362) : 0.950
ch=18, key= 6 :        55     436 :    1330    5517 ( 4187) : 0.950
ch=19, key= 5 :        55     448 :    1313    5083 ( 3770) : 0.950
ch=20, key= 4 :        45     254 :    1086    9988 ( 8902) : 0.950
ch=21, key= 3 :        65     324 :    1598    8652 ( 7054) : 0.950
ch=22, key= 2 :        65     366 :    1525    7675 ( 6150) : 0.950
ch=23, key= 1 :        65     418 :    1538    6056 ( 4518) : 0.950
ch=24, key= 0 :        55     437 :    1354    5544 ( 4190) : 0.950
```

CTSURICOA の値が、"i"コマンド実行前は初期値(65)固定ですが、"i"コマンド実行後は最適化された値となっています。(CTSURICOA, CTSUSO 値の選び方は、詳細を後述します)

なお、"i"コマンドの実行は 1 分程度(最適化値に依存)掛かります。

この表示のケースでは、閾値(threshold)は、"t"コマンドでの最適化未実行です。

以上が、自己容量と相互容量のタッチキーのサンプルプログラムの動作となります。

LCD 画面には、そのときタッチしているキーがリアルタイムで表示され、仮想端末では、数値の詳細なモニタ等が行えます。

3. タッチキー機能(CTSU)使用のフロー

タッチキー機能(CTSU)を使用する基本的なフローを示します。

3.1. 全体の流れ

(1)CTSU 初期化

・TSCAP 端子設定
・TSxx 端子設定
・CTSU レジスタ設定

(2)リファレンス ICO 入力電流調整

・CTSUSO1.CTSURICOA レジスタ値の最適化

(2')非タッチ時の電流オフセット調整

・CTSUSO0.CTSUSO レジスタ値の最適化

※(2)(2')は何度か繰り返し

(3)非タッチ時のセンサ値取得

・非タッチ時のセンサ値（基準値）取得

(4)タッチ閾値の最適化（省略可）

・タッチ時のセンサ値（基準値）取得

(5)タッチ判定

・センサ値を取得して非タッチ時の基準値と比較してタッチ／非タッチの判定を行う。

RA マイコンのタッチキー(CTSU)は、容量を測定して、センサ値（数値）に変換しています。センサ値は「リファレンス ICO 入力電流」「電流オフセット」の設定により変化するため、上記(2)(2')の部分で適切な設定値を求める事が測定精度に関わってきます。

本サンプルプログラムで採用している、(2)(2')の部分の手順、考え方を以下に示します。

3.2. 電流オフセットレジスタ値の決め方

リファレンス ICO 電流設定(CTSURICOA)と、センサ値(CTSUSC)の関係の実測例を示します。

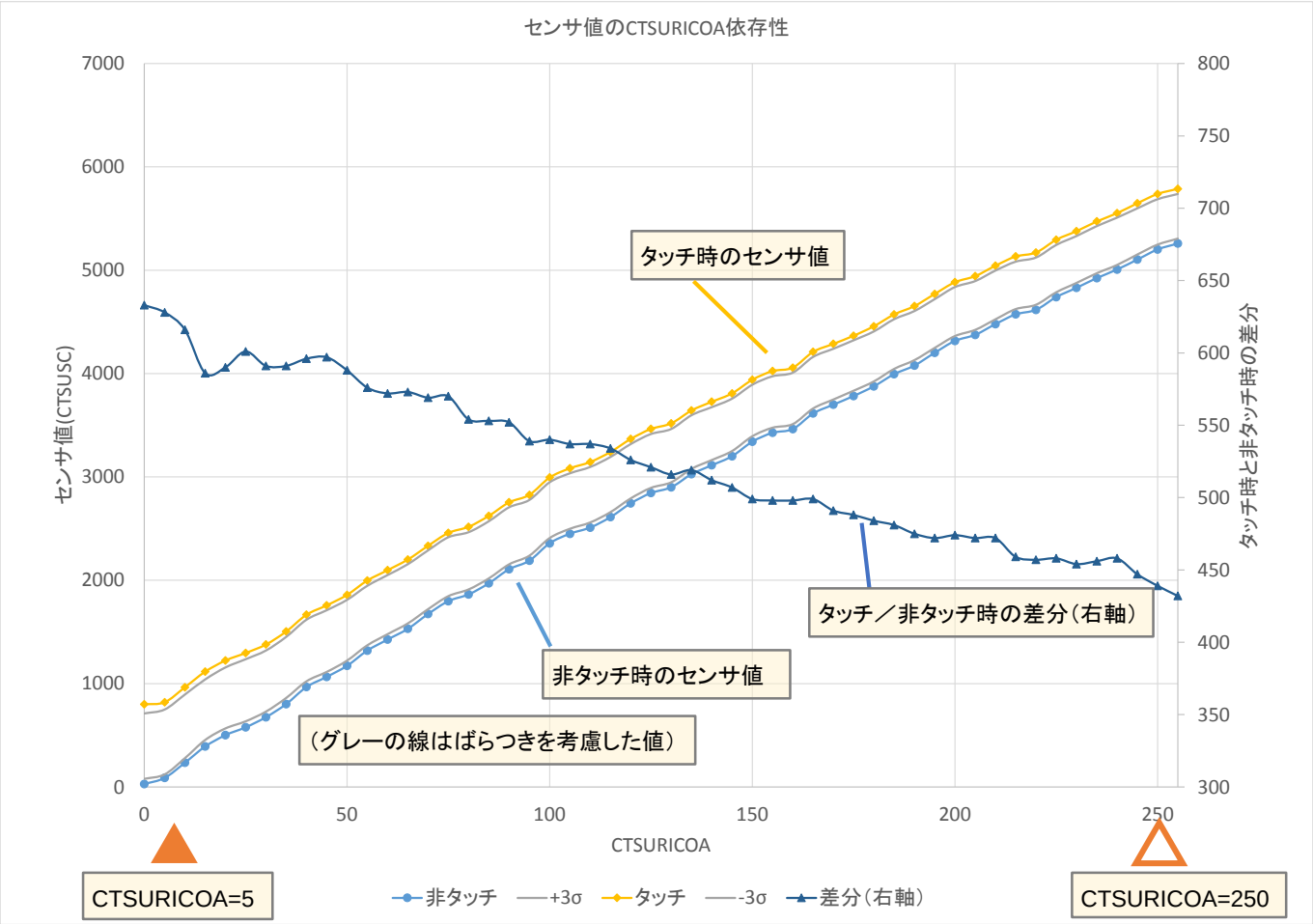


図 3-1 センサ値 CTSURICOA 依存性（自己容量）

CTSURICOA 値（0~255 の範囲で設定可能）を振った際の、非タッチ時とタッチ時のセンサ値（CTSUSC レジスタ値、容量値に対応）をプロットしたものです。同一の CTSURICOA 値に対し、測定は複数回行っており、測定値のばらつきの偏差（3σ）も取得してプロットしています。差分は（タッチ時の平均値-3σ）と（非タッチ時の平均+3σ）の差分です。

タッチ時と、非タッチ時の差分は、右肩下がりの傾向がありますで、グラフの左側を選んだ方が差が検出し易いという事が言えるかと考えます。

	CTSURICOA=5	CTSURICOA=250
非タッチ時の平均 （3σ値）	87 (34)	5201 (47)
タッチ時の平均 （3σ値）	819 (70)	5738 (51)

グラフ内の数値を抜き出したものを示します。CTSURICOA=5 のセンサ値は、非タッチ時で測定値 87 に対し、3σ値が 34 となっており、測定値に対し 40%の変動となっています。また、タッチ時で、同 8.5%の変動があります。

それに対し、CTSURICOA=250 の場合は、測定値に対しての変動は 3σ値で非タッチ、タッチ時共 1%未満となっています。CTSURICOA が小さな値の場合は、測定値に対するばらつきの比率が高い事はポイントになってくるかと考えます。よって、測定値に対するばらつきが許容できる範囲でグラフの左側の方の CTSURICOA 値を適用するのが良いと考えます。

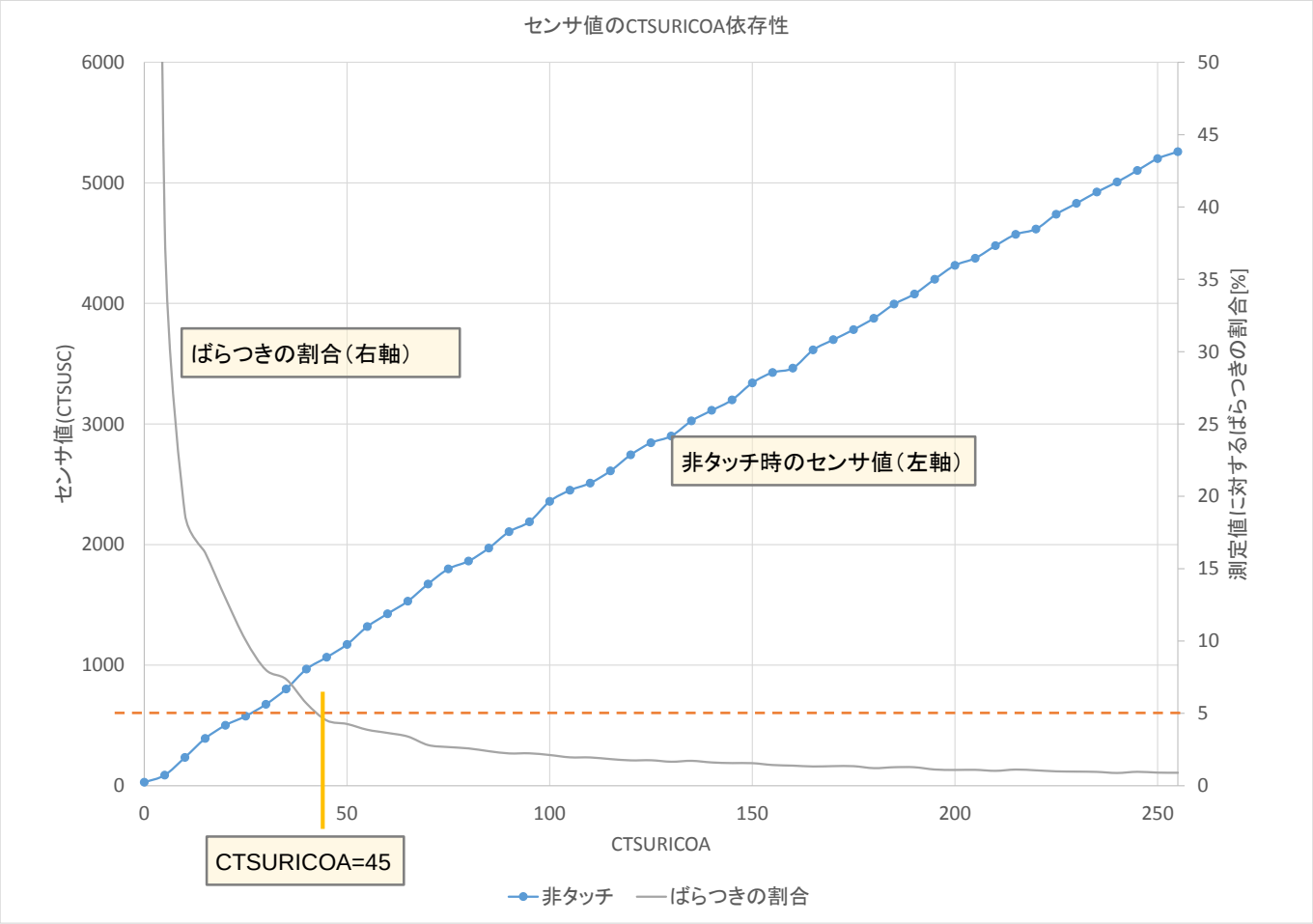


図 3-2 センサ値のばらつきの CTSURICOA 依存性（自己容量）

非タッチ時の測定値に対し、3σのばらつきの割合をプロットしたものを示します。仮に、このばらつきが、5%以下になる点を選ぶとすると、CTSURICOA=45 となります。

CTSURICOA=45 の点は、非タッチ時の測定値のばらつきが一定以下であり、かつタッチ時／非タッチ時の差分が大きく取れる点と言えるかと考えます。

本サンプルプログラムでは、
src¥ctsu¥ ctsu_offset_value.h

```
//非タッチ時のばらつき　3σが非タッチセンス値の5%未満である事
#define CTSU_NO_TOUCH_SCATTERING (0.05)
```

上記ファイル内でこの割合値を定義しており、初期値は 0.05(5%)です。

次に、電流オフセット調整(CTSUSO)の値の決め方ですが、下記の流れとなります。

・リファレンス ICO 電流値(CTSURICOA)を決める
・リファレンスカウンタ(CTSURC)値を取得
・CTSUSO を変えセンサ値(CTSUSC)を取得
・センサ値がリファレンス値を上回らない CTSUSO の値を探す

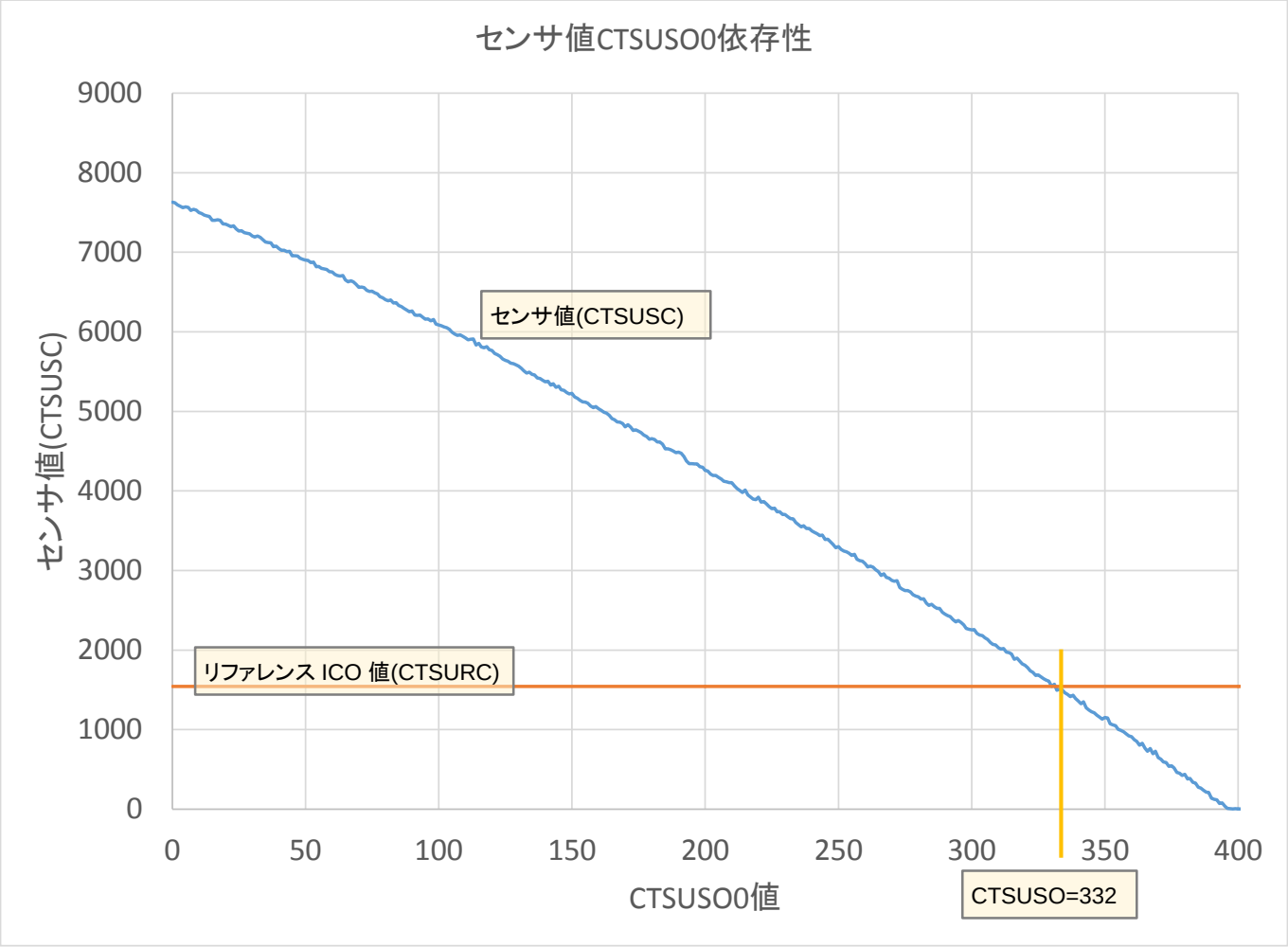


図 3-3 センサ値の CTSUSO 依存性

CTSUSO を増やしていくと、センサ値(CTSUSC)は、減少します。この値が、リファレンス ICO 値(CTSURC)を超えない範囲で、かつ、リファレンス ICO 値に近い値の方がダイナミックレンジを大きく取れるので、CTSUSO はこの例では CTSUSO=332 が求める値となります。

本サンプルプログラムでは、

(1)リファレンス ICO 電流値(CTSURICOA)に仮値(0)を設定
(2)リファレンスカウンタ(CTSURC)値を取得
(3)CTSUSO を変えセンサ値(CTSUSC)を取得
(4)センサ値がリファレンス値を上回らない CTSUSO の値を探す
(5)非タッチ時のセンサ値を複数回取得して平均とばらつきを算出
(6)センサ値のばらつきが一定以下→調整終了
(7)リファレンス ICO 電流値(CTSURICOA)を増分、(1)に戻る

上記のフローとしています。

3.3. 初期センサ値の取得

電流調整レジスタ値を決めたら、次に非タッチ時のセンサ値(CTSUSC)を取得します。

起動時の初期化としては、ここまでとなります。

3.4. タッチ閾値の最適化（オプション）

本サンプルプログラムは、閾値の最適化を行わなくても動作させることができますが、実際のタッチ時のセンサ値を取得し、非タッチ／タッチ時の中間のセンサ値に閾値を設定する事により、動作マージンの向上を図ることができます。

但し、実際のタッチキーアプリケーションでは、起動後に毎回キーにタッチして閾値を調整する事は難しいかと思います。

3.5. タッチ判定

センサ値(CTSUSC)を取得して、非タッチ時のセンサ値(CTSUSC)と比較を行い、タッチの判定を行います。

4. サンプルプログラムの解説

4.1. ソースファイル構成

1 章の手順でプロジェクトを作成すると、下記のソースファイル構成となるはずです。

[Workspace]¥*ProjectName*¥src¥common	共通定義ファイル等
[Workspace]¥*ProjectName*¥src¥ctsu	タッチキー制御プログラム（自己容量と相互容量で内容異なる）
[Workspace]¥*ProjectName*¥src¥intr	割り込み定義
[Workspace]¥*ProjectName*¥src¥lcd_1602	キャラクタ LCD 制御
[Workspace]¥*ProjectName*¥src¥rtc	リアルタイムクロック
[Workspace]¥*ProjectName*¥src¥sci	UART
[Workspace]¥*ProjectName*¥src¥settings	マイコンボード固有設定（マイコンボード毎に異なる）
[Workspace]¥*ProjectName*¥src¥timer	タイマ処理
[Workspace]¥*ProjectName*¥src¥hal_entry.c	ユーザプログラム開始関数

それぞれのフォルダ内のファイルを以下に示します。

・common

board_setting.h	マイコンボード固有設定定義ファイル
common.c	端子設定関数等
common.h	共通ヘッダ

・ctsu

本フォルダは、「自己容量」と「相互容量」でファイル構成が異なります。

－自己容量タッチキー向けプロジェクト－

ctsu.h	CTSU ヘッダ
ctsu_intr.c	CTSU 割り込み処理関数
ctsu_main.c	メイン処理（メイン関数）
ctsu_offset_value.h	オフセット値定義ファイル
ctsu_pin_def.h	端子設定ファイル
ctsu_self_cap.c	自己容量タッチキープログラム

－相互容量タッチキー向けプロジェクト－

ctsu.h	CTSU ヘッダ
ctsu_intr.c	CTSU 割り込み処理関数
ctsu_main.c	メイン処理（メイン関数）
ctsu_offset_value.h	オフセット値定義ファイル
ctsu_pin_def.h	端子設定ファイル
ctsu_mutual_cap.c	相互容量タッチキープログラム

・intr

intr.c	割り込み関数
intr.h	割り込みヘッダ

・lcd_1602

lcd_1602.c	キャラクタ LCD 制御プログラム
lcd_1602.h	キャラクタ LCD ヘッダ
lcd_1602_pin_def.c	端子設定ファイル

・rtc

rtc.c	リアルタイムクロック向けプログラム
rtc.h	リアルタイムクロックヘッダ

・sci

sci.c	UART 関連プログラム
sci.h	UART ヘッダ

・settings

※本フォルダは、マイコンボード毎に、ファイルの中身が異なります。

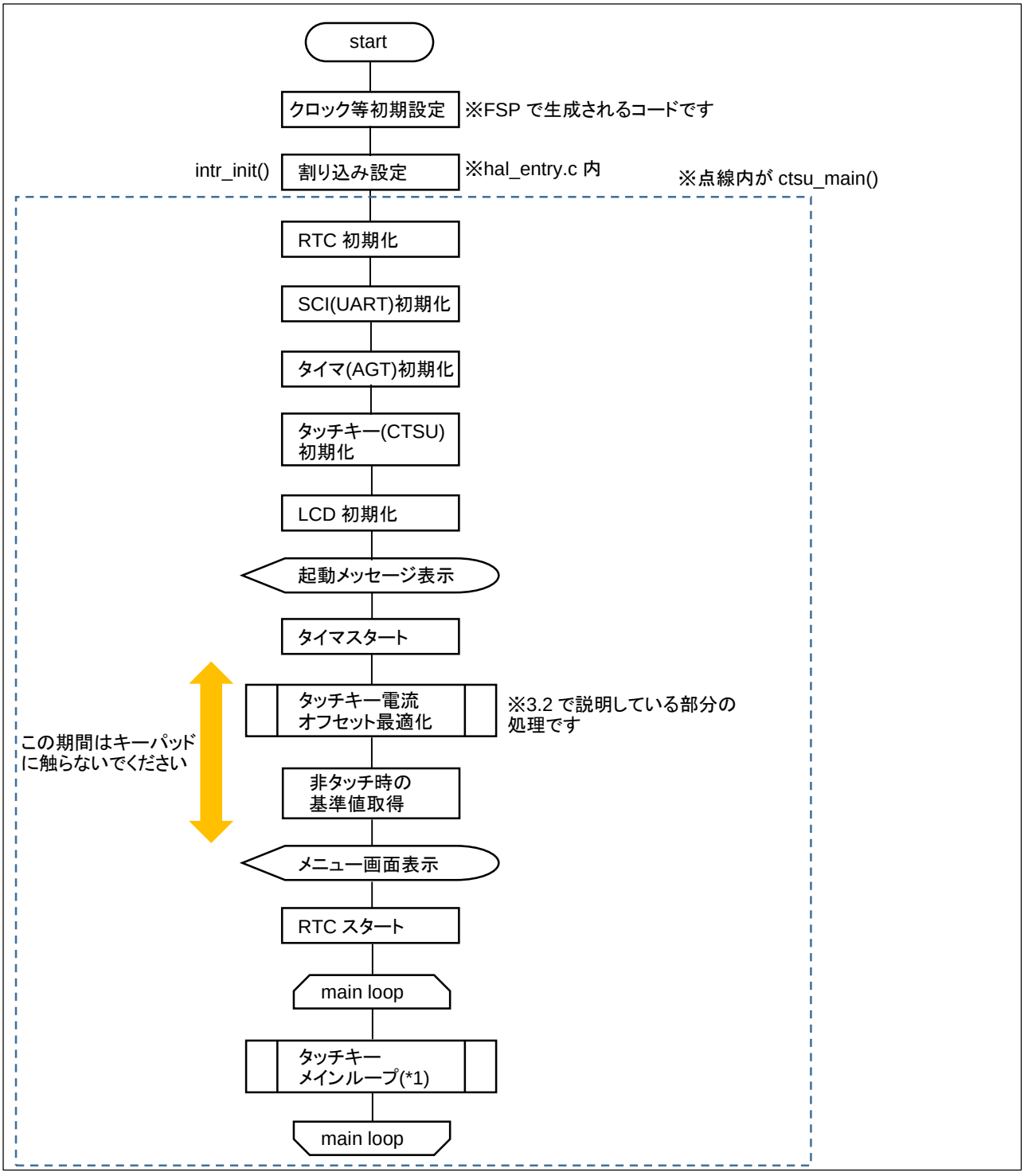
board.h	ボード種定義ファイル

・timer

agt.c	タイマ定期処理プログラム
agt.h	タイマヘッダ

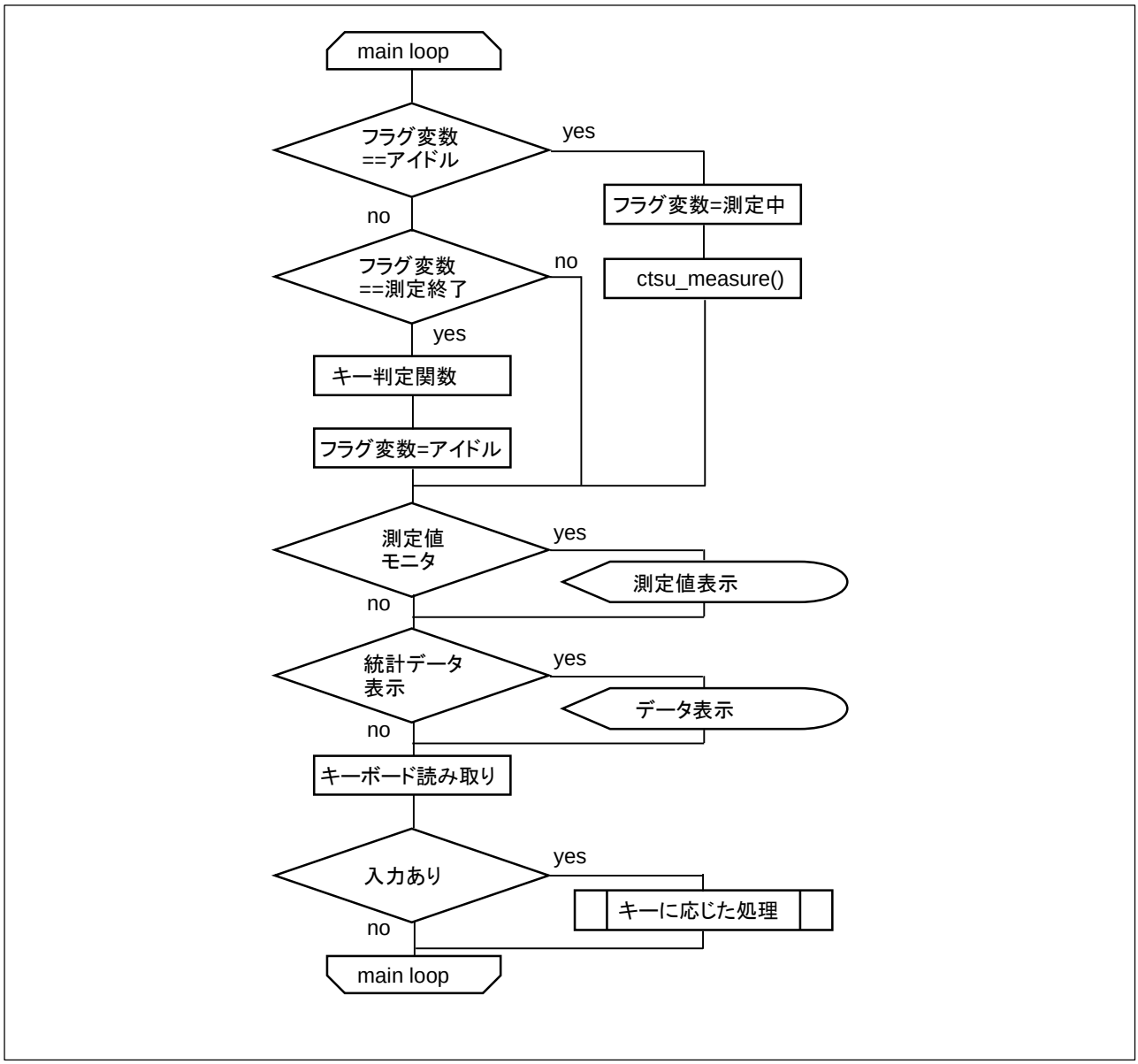
4.2. プログラムのフロー

・全体フロー[ctsu¥ctsu_main.c, ctsu_main()]



各種初期化を行った後、メインループ（無限ループ）を実行します。

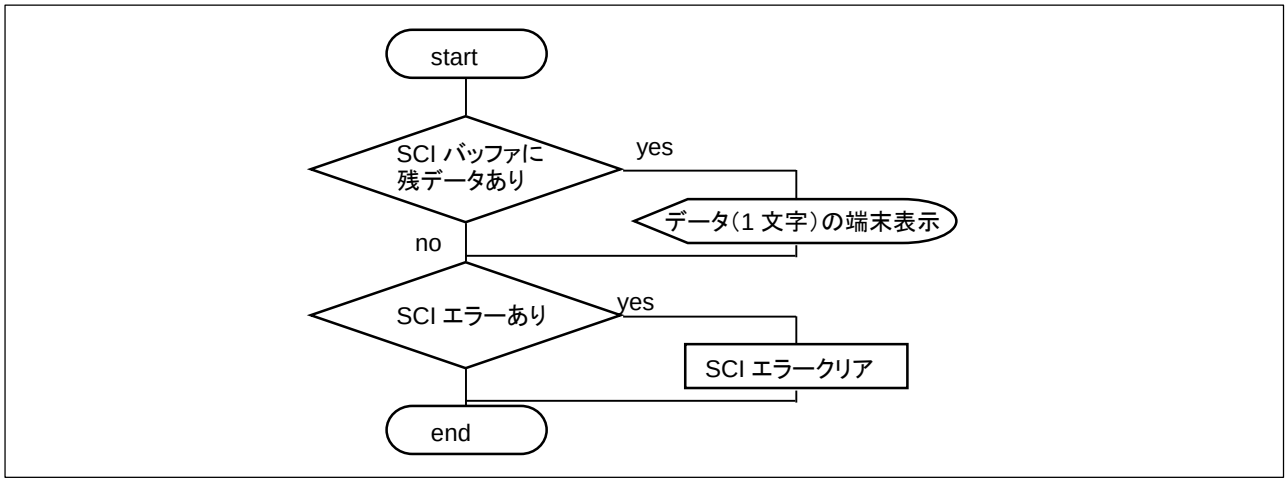
・メインループ(*1)



メインループでは、フラグ変数のセットと、キーボードの読み取り処理等を行います。

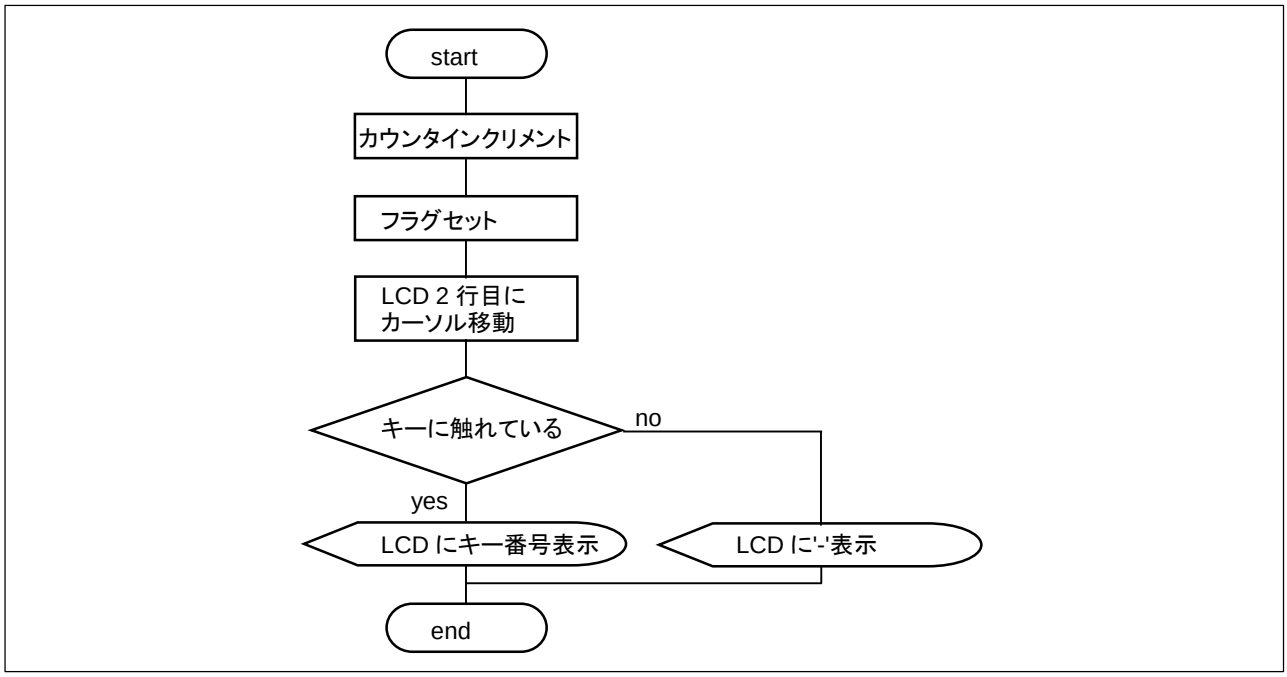
・割り込み関数

ータイマ関数（100us 毎）[¥timer¥agt.c, intr_agt0_agti()]ー



AGT タイマを使い、100us 毎に実行される割り込み関数です。SCI(UART)は、表示時にバッファに格納し、本関数で表示処理が行われます。

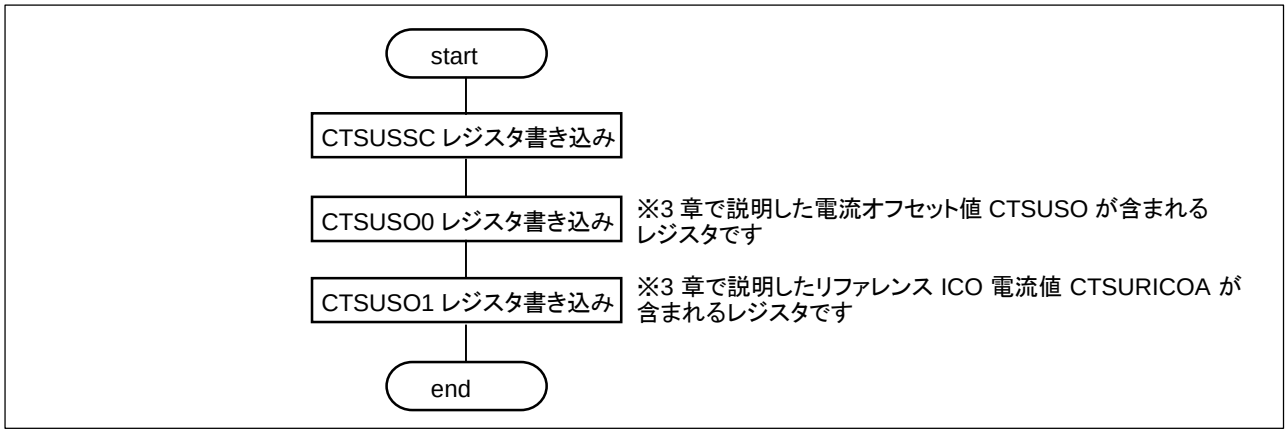
ーRTC 周期割り込み（1/64s 毎）[¥rtc¥rtc.c, intr_rtc_prd()]ー



RTC の周期割り込みは、LCD に表示の更新に使用しています。メインループの「キー判定関数」の判定結果を、1/64 秒毎に、LCD 画面に反映させます。

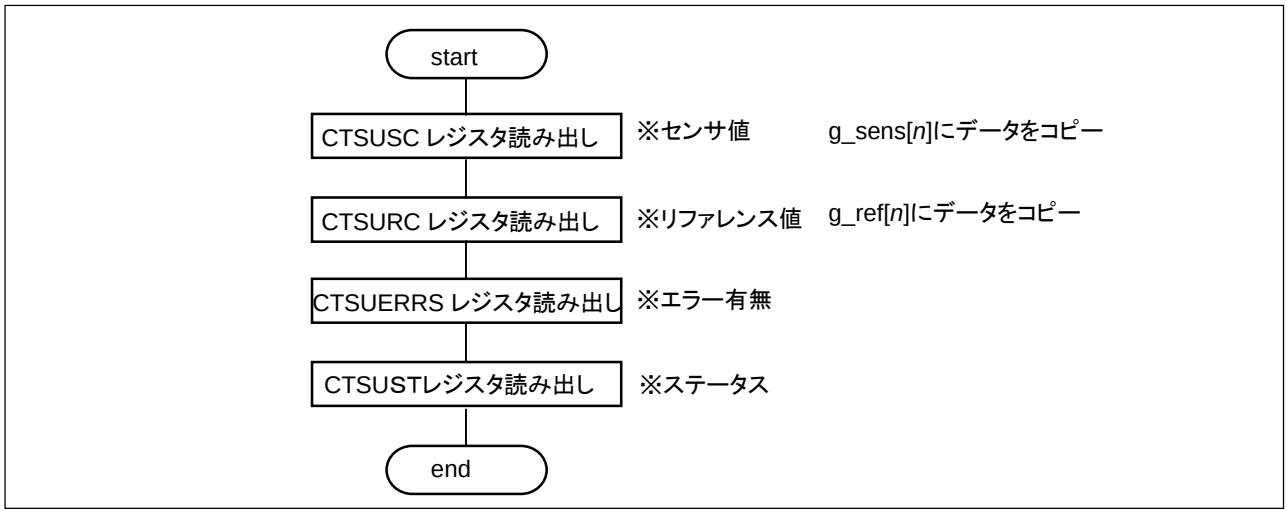
※本サンプルプログラムでは、メインクロックをベースとした AGT タイマと、サブクロック(32.768kHz)をベースとした、RTC タイマを使用していますが、タイマを使うサンプルとして 2 種類の異なるタイマを使用しています。
（CTSU 使用時に、原理的に 2 種のタイマを使用しなければならないという訳ではありません）

－CTSU WRITE 割り込み[¥ctsu¥ctsu_intr.c, intr_ctsu_ctsuwr()]－



CTSU WRITE 割り込みでは、電流設定値等のレジスタにデータをセットする処理を行っています。CTSU の処理をスタートさせると、この割り込みが入りますので、そのタイミングで上記 3 レジスタに値を書き込みます。
（CTSUSO1 レジスタに値が書き込まれると、実際のセンサ値の測定に移ります）

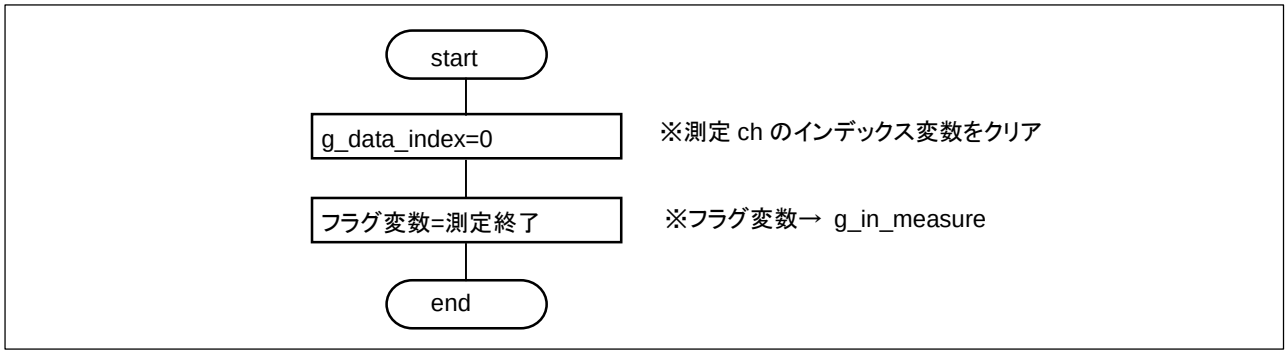
－CTSU READ 割り込み[¥ctsu¥ctsu_intr.c, intr_ctsu_ctsurd()]－



CTSU RD 割り込みは、測定が終わった段階で呼び出される割り込みです。測定値を、グローバル変数にコピーする処理を行っています。

※CTSU WRITE, CTSU READ 割り込みでは、（処理が判り易い様に）直接レジスタへの書き込み、読み出しを行っていますが、ここで使用しているレジスタアクセスは DTC が使用可能です。（実際のアプリケーションプログラムでは、DTC の使用もご検討ください。）

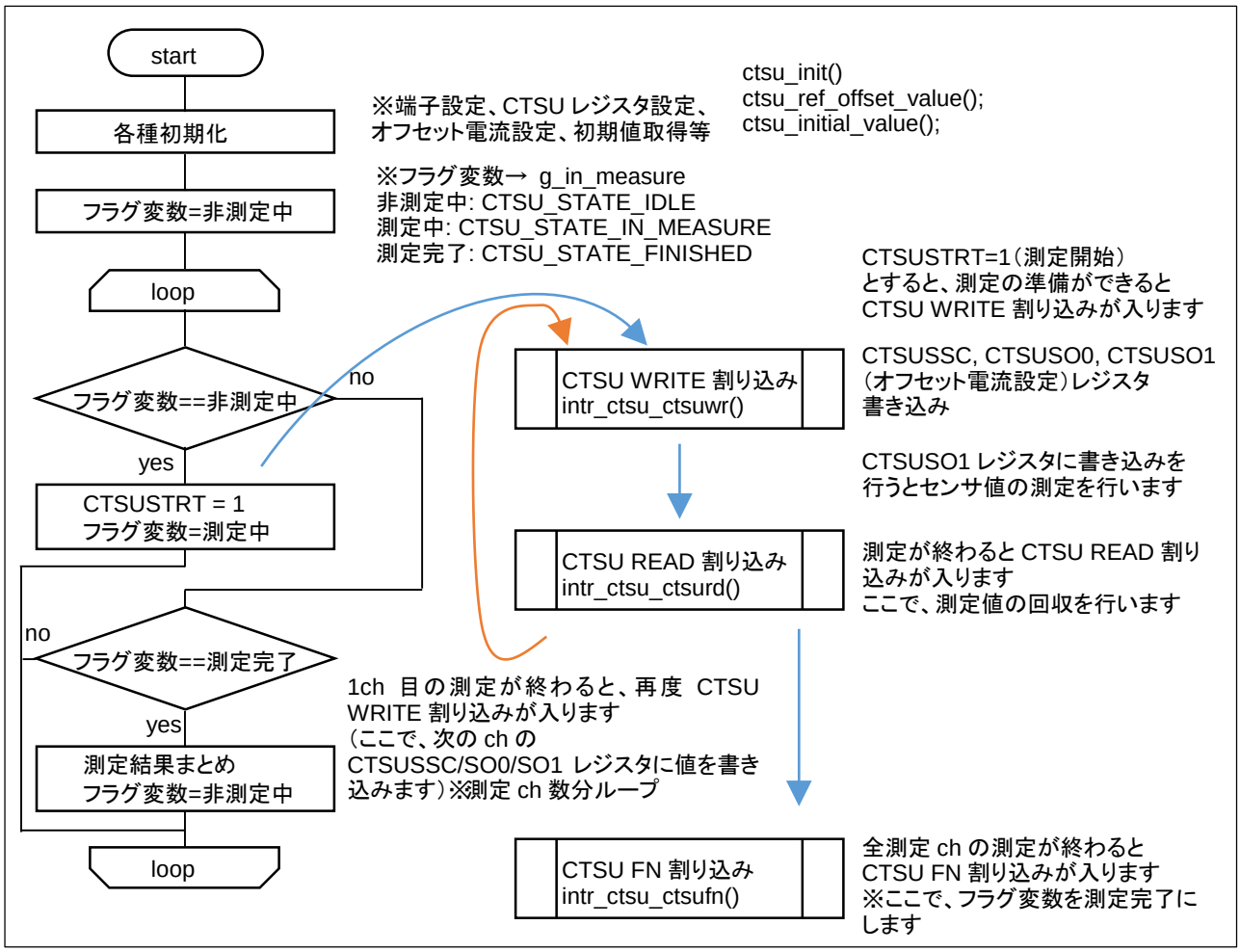
ーCTSU FN 割り込み[¥ctsu¥ctsu_intr.c, intr_ctsu_ctsufn()]ー



CTSU FN 割り込みは、全チャネルの測定が終わった段階で呼び出される割り込みです。インデックス変数やフラグ変数の設定を行っています。

4.3. CTSU 処理の流れ

タッチキー機能(CTSU)は、以下の手順で使用します。



CTSU の処理は基本的には、上記の流れとなります。

測定開始(CTSUSTRT=1)とすると、CTSU マクロは（内部で準備をした後）CTSU WRITE 割り込みを掛けます。ユーザプログラム側で、CTSU WRITE 割り込みルーチン内では、CTSUSSC, CTSUSO0（リファレンス ICO 電流設定(CTSURICOA)を含むレジスタ）, CTSUSO1（電流オフセット値 CTSUSO を含むレジスタ）に、値を書き込みます。

※CTSUSO0, CTSUSO1 設定値は、測定 ch 毎に最適化されている事が望ましいです

CTSUSO1 レジスタに値を書き込むと、（端子容量の）センサ値が測定され、測定が終わると、CTSU READ 割り込みが入ります。ここでは、センサ値の測定値(CUSUSC)等のレジスタを読み出します。

※センサ値が格納されるレジスタ(CTSUSC)は、1 つしかありませんので、CTSU READ 割り込みの時点で保存する必要があります
（例えば、A/D コンバータの結果を格納するレジスタは、ch 毎に別になっていますが、CTSU は 1 つしかありません）

CTSURC（リファレンスカウンタ値）レジスタ読み出し後に、CTSU マクロは次の測定 ch の準備に入り、準備ができると、再度 CTSU WRITE 割り込みを掛けます。

2ch 目の CTSUSSC, CTSUSO0, CTSUSO1 値を書き込むと、2ch 目の測定後 CTSU READ 割り込みが入り、同様の処理が、測定 ch 分続きます。

※測定 ch は、予め測定対象と設定している ch 数分です

最後の測定 ch まで一連の処理が終わると、CTSU FN 割り込みが入ります。

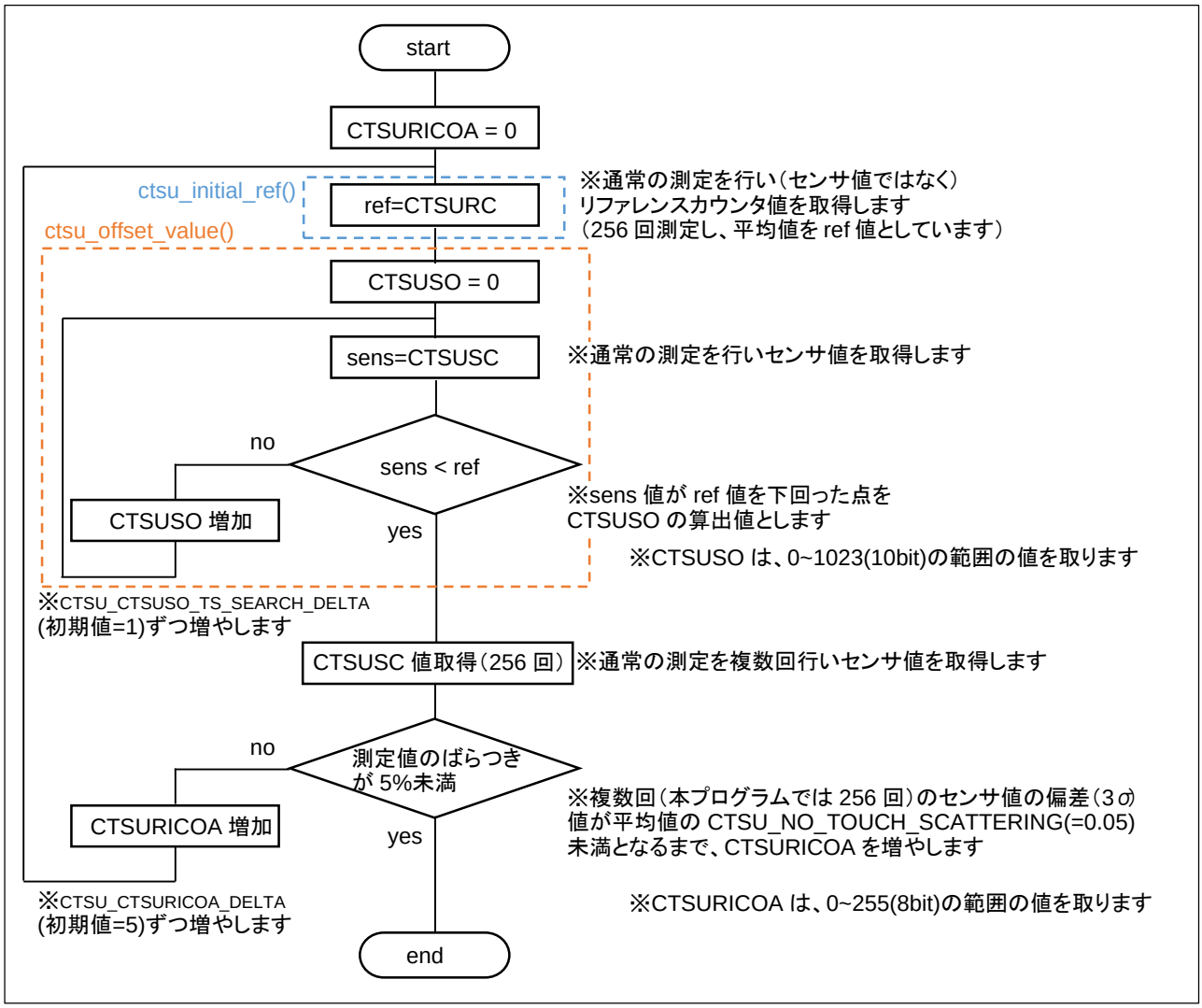
ここで、全 ch の結果まとめや、フラグレジスタのクリア等必要な処理を行います。

※本サンプルプログラムでは、全 ch の測定が終わると、フラグをクリアし、無限ループ内で次の測定を開始するようになっていますが、CTSUSTRT=1 のセットは、例えばタイマで一定時間毎に行う等、タイミングは自由です

4.4. 電流オフセットの最適化

3.2 章で説明している
CTSUSO0.CTSUSO　　リファレンス電流
CTSUSO1.CTSURICOA　　電流オフセット
値の最適化のフローを示します。

－電流オフセットの最適化 ctsu_ref_offset_value()－



上記フローチャートでは記載を省略していますが、測定 ch 毎に CTSUSO, CTSURICOA 値を算出しています。

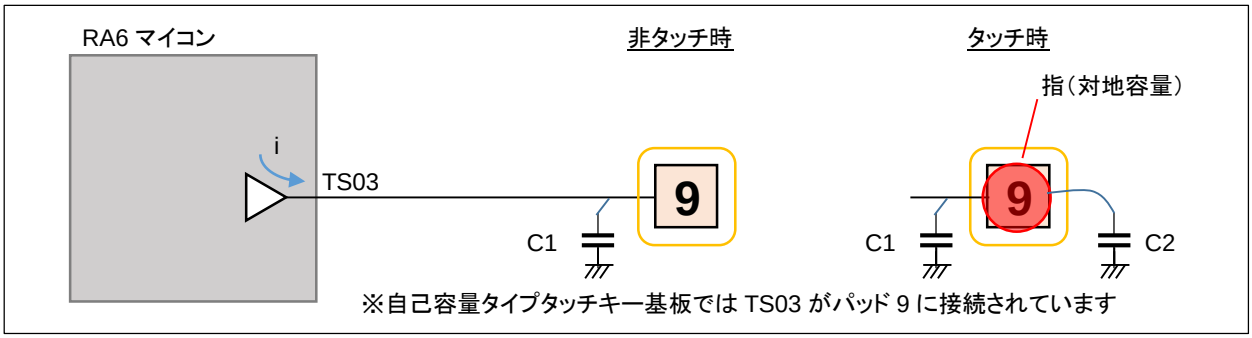
CTSURICOA 増分ループ内に、CTSUSO 増分ループが入れ子になる形で、2 値を増やしながら最適値を算出しています。

※上記は、本サンプルプログラムで採用しているアルゴリズムです。実際のタッチキーアプリケーションでは、初期化に掛けられる時間等を勘案して、アルゴリズムを決めて頂きたく。

4.5. タッチの判定

4.5.1. 自己容量タイプ

自己容量タイプのタッチ判定は単純です。



自己容量は、マイコンの端子(TSxx)に 1:1 でキーパッドが接続される形となります。
（キーパッドの数=測定 ch 数となります）

TS のラインを L→H に遷移させると、端子からは容量に比例した電流(i)が流れ出します。電流を測定する事により、TS ラインにつながっている容量の大きさを測る仕組みです（詳細は、マイコンのハードウェアマニュアルや、ルネサスエレクトロニクスより提供されている、CTSU 測定の原理の資料を参照ください）。

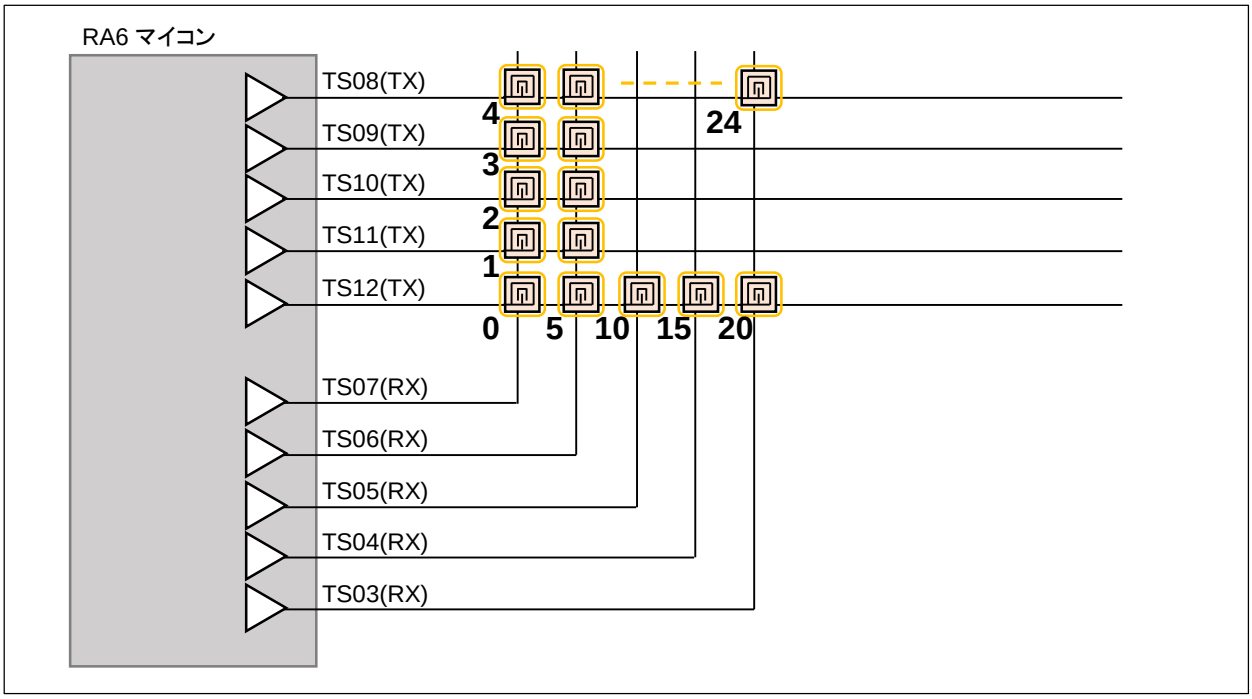
・非タッチ時の TS03 のセンサ値(CTSUSC)を取得（C1 に相当する値）
・タッチ判定の閾値を決める(*1)
・タッチ時は TS03 のノードの容量値が増加＝センサ値が増加する（C1+C2 に相当する値）
・センサ値が閾値を超えていたら「タッチしている」と判断する

(*1)判定の閾値(g_touch_threshold)は、初期値は 1.2（20%以上センサ値が増加した場合）としています
サンプルプログラムでは、"t"コマンドで、実際にタッチした際のセンサ値を元に閾値を最適化できる様になっています
閾値は、「割合で判断する」手法と「絶対値で判断する」手法があるかと考えますが、本サンプルプログラムでは前者としています。

※実際のタッチキーアプリケーションでは、
・起動後に実際にタッチして、閾値の最適化が可能
・装置組み立て後に、装置毎に閾値の最適化が可能
・予め規定値を代入
等、用途に応じて閾値の決め方は変わるかと考えます。

4.5.1. 相互容量タイプ

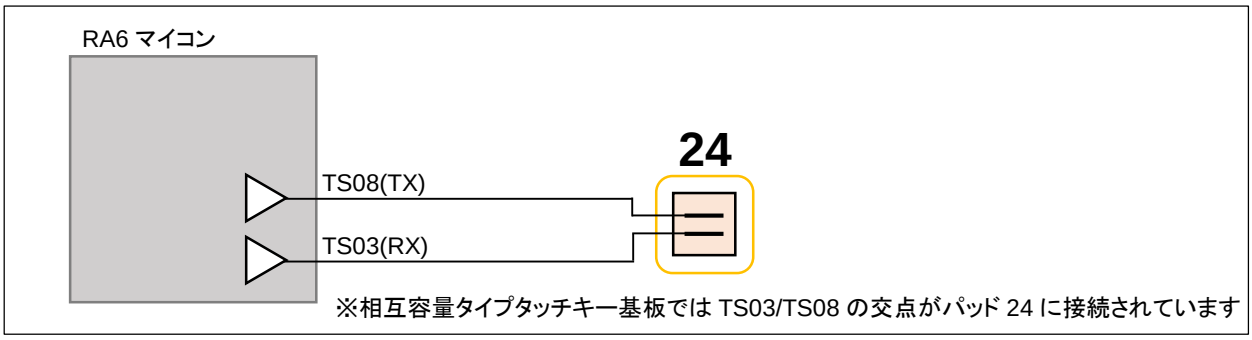
相互容量タイプのタッチ判定は以下の様になります



相互容量は、マトリックス接続の TX と RX の交点のタッチ／非タッチを判定する形となります。キット付属の相互容量タッチキー基板(D55A)は、5x5(TX:5, RX:5)の 25 キーで構成されています。

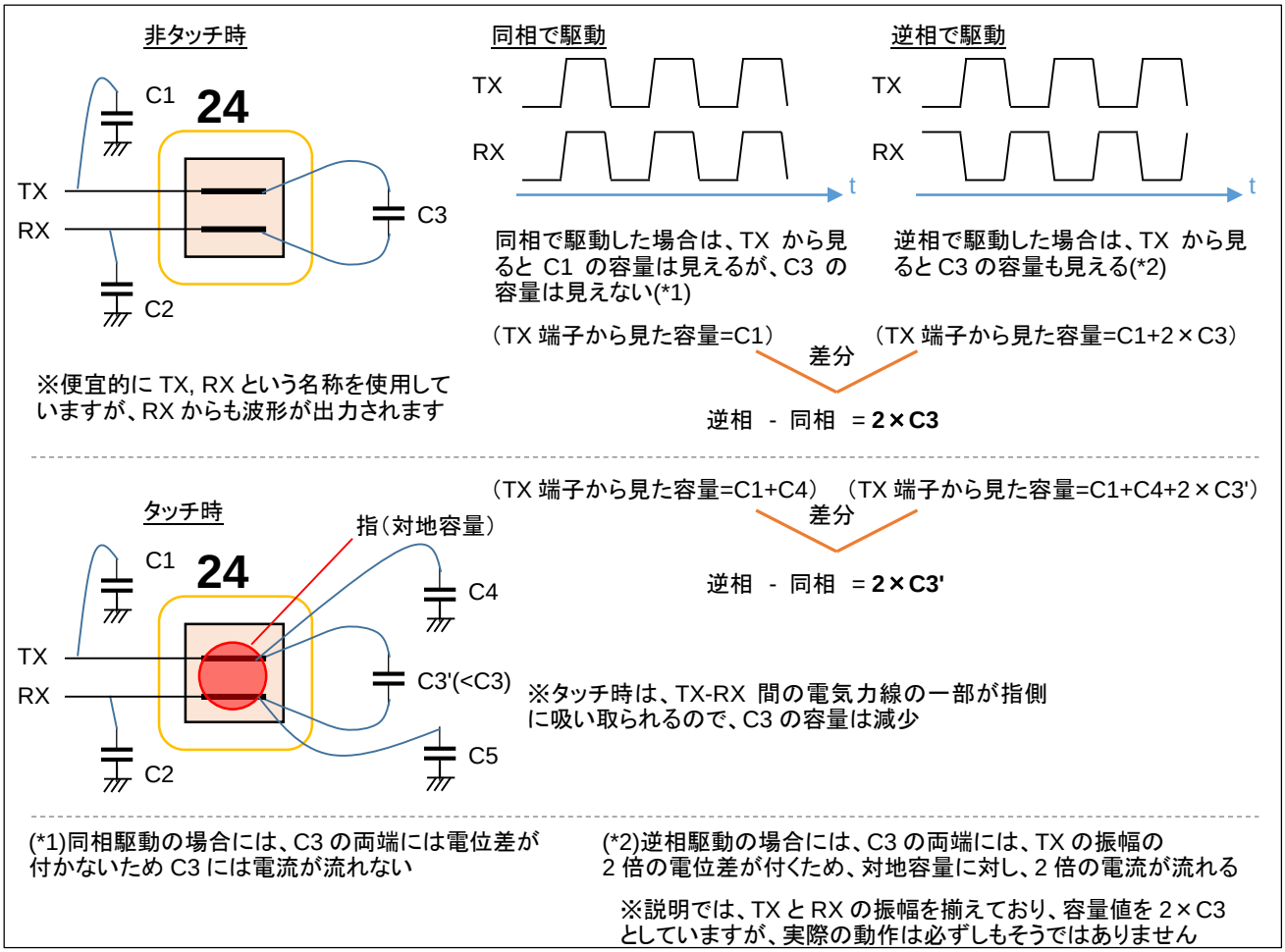
TS12 のラインが、0, 5, 10, 15, 20 のキーパッドに、TS07 のラインが 0, 1, 2, 3, 4 のキーパッドにつながっています。例えば 24 のキーパッドは、TS08 と TS03 の交点となります。

キーパッド 24 の部分を抜き出したものが、下記です。



キーパッド 24 の容量を測定する際は、TS08(TX)と TS03(RX)の 2 線間の容量を以下の手法で測定します。

TX, RX を「同相で駆動」「逆相で駆動」の 2 回測定する事により、TX, RX 間の容量を測ります。



逆相での測定値から同相での測定値を差し引く事により、TX-RX 間の容量に相当する測定値が得られます。タッチ時は、TX-RX 間の容量値は減少するので、数値の減少でタッチ判定を行います。

・非タッチ時の（TS03(RX)同相駆動時の）TS08(TX)のセンサ値を取得(*1)
・非タッチ時の（TS03(RX)逆相駆動時の）TS08(TX)のセンサ値を取得(*2)
・(*2)-(*1)の値（逆相、同相の差分値）を求め、非タッチ時の基準値とする
・タッチ判定の閾値を決める(*3)
・タッチ時は逆相と同相の差分値が減少
・差分値が閾値を下回っていたら「タッチしている」と判断する

(*3)判定の閾値(g_touch_threshold)は、初期値は 0.95（5%以上センサ値が減少した場合）としています
サンプルプログラムでは、"t"コマンドで、実際にタッチした際のセンサ値を元に閾値を最適化できる様になっています
閾値は、「割合で判断する」手法と「絶対値で判断する」手法があるかと考えますが、本サンプルプログラムでは前者としています。

相互容量タイプでは、キーが 25 ある場合、同相と、逆相で 50ch の測定を行います。

5. サンプルプログラムの関数

サンプルプログラムで使用している関数に関して示します。
章名は、フォルダ名に対応しています（5.1 ctsu　は、ソースの ctsu フォルダ内の関数です）

5.1. ctsu

ctsu_init

概要：初期化関数
宣言：
　　void ctsu_init(void);
説明：
　・変数初期化
　・TSCAP 端子放電
　・TS 端子設定
　・CTSU レジスタの設定
を行います
引数：
　　なし
戻り値：
　　なし

ctsu_measure

概要：測定指示関数
宣言：
　　void ctsu_measure(void);
説明：
　・測定の開始(CTSUSTRT=1)
を行います
引数：
　　なし
戻り値：
　　なし

ctsu_set_param

概要:パラメータの設定関数
宣言:
　　void ctsu_set_param(void);
説明:
　・測定 ch 毎の CTSUSSC, CTSUSO0, CTSUSO1 レジスタ値のセット
を行います
引数:
　　なし
戻り値:
　　なし

ctsu_read

概要:測定値の読み出し関数
宣言:
　　void ctsu_read(void);
説明:
　・センサ値(CTSUSC), リファレンス値(CTSURC), エラー, ステータス, レジスタ値をグローバル変数へコピー
を行います
引数:
　　なし
戻り値:
　　なし

ctsu_ref_offset_value

概要:電流オフセット設定の最適化関数
宣言:
　　void ctsu_ref_offset_value(void);
説明:
　・CTSUSO0(CTSUSO), CTSUSO1(CTSURICOA)値の最適値の算出
を行います。
引数:
　　なし
戻り値:
　　なし

ctsu_offset_value

概要:電流オフセット値の最適化関数
宣言:
　　void ctsu_offset_value(void)
説明:
　・CTSUSO0(CTSUSO)の最適値の算出
を行います。
引数:
　　なし
戻り値:
　　なし
補足:
　ctsu_ref_offset_value()から呼び出される関数です。

ctsu_ref_initial_ref

概要:リファレンスカウンタ初期値取得
宣言:
　　void ctsu_initial_ref(void)
説明:
　・リファレンス値(CTSURC)の初期値の算出
を行います。
引数:
　　なし
戻り値:
　　なし

ctsu_ref_initial_value

概要:センサ初期値取得
宣言:
　　void ctsu_initial_value(void);
説明:
　・非タッチ時のセンサ値(CTSUSC)の取得
を行います。
引数:
　　なし
戻り値:
　　なし

ctsu_result

概要:タッチ結果取得関数

宣言:

unsigned char ctsu_result(unsigned char *key_state);

説明:

・タッチ結果の判定

を行います。

引数:

*key_state: 押しているキーが格納されます

unsigned char 型の key_state[0] ~ key_state[9]が、押しているキー0x01, [自己容量]

unsigned char 型の key_state[0] ~ key_state[24]が、押しているキー0x01, [相互容量]

押されていないキー0x00 となります。キー数(9[自己容量])(25[相互容量])以上の配列のポインタを引数に指定してください。

戻り値:

タッチ判定されている中で、一番容量の変動が大きなキー

補足:

複数のキーが押されている場合、key_state には、押されているキーの情報が入ります。

戻り値は、一番しっかり押しているキーとなります。

ctsu_row_column [相互容量のみ]

概要:行列分割関数

宣言:

void ctsu_row_column(unsigned char num, unsigned char *result);

説明:

・行列の分割

を行います。

引数:

num: キー番号

*result: 分割後の行、列番号, result[0]が行番号, result[1]が列番号を返します。

戻り値:

なし

intr_ctsu_ctsuwr

概要:CTSU WRITE 割り込み関数
宣言:
　　intr_ctsu_ctsuwr(void)
説明:
　・ctsu_set_param()（測定 ch 毎の CTSUSO0, CTSUSO1 設定）の呼び出し）
を行います

intr_ctsu_ctsurd

概要:CTSU READ 割り込み関数
宣言:
　　intr_ctsu_ctsurd(void)
説明:
　・ctsu_read()（測定結果のコピー）の呼び出し
を行います

intr_ctsu_ctsufn

概要:CTSU FN 割り込み関数
宣言:
　　intr_ctsu_ctsufn(void)
説明:
　・測定 ch を示すインデックス変数のクリア（0 代入）
　・測定中を示すフラグ変数を「測定終了」のステータス変更
を行います

5.2. common

pfs_set

概要:PFS 設定関数
宣言:
　　int pfs_set(char* term, unsigned short psel);
説明:
　・端子の PFS 設定
を行います
引数:
　　term: 端子名（P100 等）
　　psel: PFS.PSEL 設定値（0x0C 等）
戻り値:
　　0: 正常終了
　　1: 引数エラー

pmr_set

概要:PMR 設定関数
宣言:
　　int pmr_set(char* term);
説明:
　・端子の PMR 設定（周辺機器割り当て）
を行います
引数:
　　term: 端子名（P100 等）
戻り値:
　　0: 正常終了
　　1: 引数エラー

pmr_clear

概要:PMR 設定関数
宣言:
　　int pmr_clear(char* term)
説明:
　・端子の PMR 設定解除（周辺機器割り当てから開放）
を行います
引数:
　　term: 端子名（P100 等）
戻り値:
　　0: 正常終了
　　1: 引数エラー

5.3. intr

intr_init

概要:割り込み設定関数
宣言:
　　void intr_init(void);
説明:
　・割り込みの初期設定（NVIC_ISER, NVIC_IPR レジスタ設定）
を行います
引数:
　　なし
戻り値:
　　なし

5.4. lcd_1602

lcd_init

概要:LCD 初期化関数
宣言:
　　void lcd_init(void);
説明:
　・LCD 制御の初期化
を行います
引数:
　　なし
戻り値:
　　なし

lcd_cmd

概要:LCD コマンド送信関数
宣言:
　　void lcd_cmd(unsigned char c);
説明:
　・LCD へのコマンドの送信
を行います
引数:
　　c: 送信コード
戻り値:
　　なし

lcd_hs1

概要:LCD 1 行目にカーソルを設定する関数
宣言:
　　void lcd_hs1(void);
説明:
　・LCD のカーソルを 1 行目に移動させる処理
を行います
引数:
　　なし
戻り値:
　　なし

lcd_hs2

概要:LCD 2 行目にカーソルを設定する関数
宣言:
　　void lcd_hs2(void);
説明:
　・LCD のカーソルを 2 行目に移動させる処理
を行います
引数:
　　なし
戻り値:
　　なし

lcd_clear

概要:LCD 画面のクリアを行う関数
宣言:
　　void lcd_clear(void);
説明:
　・LCD の画面消去
を行います
引数:
　　なし
戻り値:
　　なし

lcd_write_char

概要:LCD 1 文字表示関数
宣言:
　　void lcd_write_char(unsigned char c);
説明:
　・LCD に 1 文字表示させる処理
を行います
引数:
　　c: 文字コード
戻り値:
　　なし

lcd_write_hex

概要:LCD hex 表示関数
宣言:
　　void lcd_write_hex(unsigned char c);
説明:
　・LCD に hex で 1 バイト表示させる処理（0xAB 等）
を行います
引数:
　　c: 数値
戻り値:
　　なし

lcd_write_byte_int

概要:LCD 数値表示関数
宣言:
　　void lcd_write_byte_int(unsigned char num);
説明:
　・LCD に 1 バイトの数値表示させる処理（123 等）
を行います
引数:
　　num: 数値
戻り値:
　　なし

lcd_write_short_int

概要:LCD 数値表示関数
宣言:
　　void lcd_write_short_int(unsigned short num);
説明:
　・LCD に 2 バイトの数値表示させる処理（12345 等）
を行います
引数:
　　num: 数値
戻り値:
　　なし

lcd_write_str

概要:LCD 文字列表示関数
宣言:
　　void lcd_write_str(char *str);
説明:
　・LCD に文字列表示
を行います
引数:
　　*str: 文字列
戻り値:
　　なし

5.5. rtc

rtc_init

概要:RTC 初期化関数
宣言:
　　void rtc_init(void);
説明:
　・RTC の初期化
を行います
引数:
　　なし
戻り値:
　　なし

rtc_start

概要:RTC 開始関数
宣言:
　　rtc_start(void);
説明:
　・RTC のカウント開始
を行います
引数:
　　なし
戻り値:
　　なし

rtc_stop

概要:RTC 停止関数
宣言:
　　rtc_stop(void);
説明:
　・RTC のカウント停止
を行います
引数:
　　なし
戻り値:
　　なし

ー割り込み関数ー

intr_rtc_prd

概要:RTC 周期割り込み
宣言:
　　void intr_rtc_prd(void);
説明:
　・RTC 周期割り込み（1/64 秒間隔で割り込み）
を行います

5.6. sci

sciInit

概要:SCI 初期化関数
宣言:
　　void sciInit(void);
説明:
　・SCI の初期化
を行います
引数:
　　なし
戻り値:
　　なし

sciRead

概要:SCI 読み取り関数
宣言:
　　char sciRead(void);
説明:
　・SCI（キーボード）から 1 文字読み出し
を行います
引数:
　　なし
戻り値:
　　文字コード（0xff のときは、キー入力なし）

sciWrite, sciWriteBuf

概要:SCI 1 文字出力関数
宣言:
　　void sciWrite(unsigned char Buffer);
　　void sciWriteBuf(unsigned char Buffer);
説明:
　・SCI（端末）に 1 文字出力
を行います
引数:
　　Buffer: 文字コード
戻り値:
　　なし

補足:
Buf なしの関数(sciWrite)は、文字出力が終わるまで、関数から処理が戻りません。
Buf 付きの関数は、関数内では sci の出力バッファへのコピーのみを行い、文字出力は、100us 毎の AGT 割り込み内で行います。(AGT タイマ開始前は、Buf なし版を使用してください)

sciPrint, sciPrintBuf

概要:SCI 1 文字列出力関数
宣言:
void sciPrint(char *Buffer);
void sciPrintBuf(char *Buffer);
説明:
・SCI(端末)に文字列出力
を行います
引数:
*Buffer: 文字列のポインタ
戻り値:
なし
補足:
Buf の有無は、sciWrite と同様です。SCI への文字出力は、115,200bps では 1 文字あたり 90us 程度掛かります。マイコンの動作クロック(120MHz->8.33ns)に対して非常に低速で、sciPrint 処理で長時間処理が止まるため、Buf 付を用意しています。sci の出力バッファは 2048 文字分確保しています(増やす事は可能です)。短期間に、Buf 付き関数を用いて大量の文字列を出力するとバッファが溢れるため、その様な場合は出力タイミングを調整するか、Buf なし版を使ってください。
(Buf あり版→Buf なし版に切り替える際は、バッファが空になるための 2048x100us=0.2 秒程度のウェイトを入れてください)

sciPrintByte, sciPrintByteBuf

概要:SCI 1 バイト出力関数
宣言:
void sciPrintByte(unsigned char dmy);
void sciPrintByteBuf(unsigned char dmy);
説明:
・SCI(端末)に 1 バイトを hex 表記出力(0x12 等)
を行います
引数:
dmy: 数値
戻り値:
なし

sciPrintWord, sciPrintWordBuf

概要:SCI 2 バイト出力関数
宣言:
　　void sciPrintWord(unsigned short dmy);
　　void sciPrintWordBuf(unsigned short dmy);
説明:
　・SCI（端末）に 2 バイトを hex 表記出力（0x1234 等）
を行います
引数:
　　dmy: 数値
戻り値:
　　なし

sciPrintDword, sciPrintDwordBuf

概要:SCI 4 バイト出力関数
宣言:
　　void sciPrintDword(unsigned long dmy);
　　void sciPrintDwordBuf(unsigned long dmy);
説明:
　・SCI（端末）に 4 バイトを hex 表記出力（0x12345678 等）
を行います
引数:
　　dmy: 数値
戻り値:
　　なし

sciPrintByteInt, sciPrintByteIntBuf

概要:SCI 1 バイト出力関数
宣言:
　　void sciPrintByteInt(unsigned char dmy);
　　void sciPrintByteIntBuf(unsigned char dmy);
説明:
　・SCI（端末）に 1 バイトを数値表記出力（123 等）
を行います
引数:
　　dmy: 数値
戻り値:
　　なし

sciPrintShortInt, sciPrintShortIntBuf

概要:SCI 2 バイト出力関数
宣言:
　　void sciPrintShortInt(unsigned short dmy);
　　void sciPrintShortIntBuf(unsigned short dmy);
説明:
　・SCI（端末）に 2 バイトを数値表記出力（12345 等）
を行います
引数:
　　dmy: 数値
戻り値:
　　なし

sciPrintByteSignedInt, sciPrintByteSignedIntBuf

概要:SCI 1 バイト符号付き出力関数
宣言:
　　void sciPrintByteSignedInt(char dmy);
　　void sciPrintByteSignedIntBuf(char dmy);
説明:
　・SCI（端末）に 1 バイトを符号付き数値表記出力（-123 等）
を行います
引数:
　　dmy: 数値
戻り値:
　　なし

sciPrintShortSignedInt, sciPrintShortSignedIntBuf

概要:SCI 2 バイト符号付き出力関数
宣言:
　　void sciPrintShortSignedInt(short dmy);
　　void sciPrintShortSignedIntBuf(short dmy);
説明:
　・SCI（端末）に 2 バイトを符号付き数値表記出力（-12345 等）
を行います
引数:
　　dmy: 数値
戻り値:
　　なし

－割り込み関数－

intr_sci9_rxi

概要:SCI RXI 割り込み
宣言:
　　void intr_sci9_rxi(void)
説明:
　・SCI（キーボード）に入力があった際データの読み出しと、フラグ変数のセット
を行います

5.7. timer

agt0_init

概要:AGT0 初期化関数
宣言:
　　void agt0_init(void);
説明:
　・AGT0 の初期化
を行います
引数:
　　なし
戻り値:
　　なし
補足:
　本関数では、100us 毎の周期割り込み設定となっています。

agt0_start

概要:AGT0 開始関数
宣言:
　　void agt0_start(void);
説明:
　・AGT0 のカウント開始
を行います
引数:
　　なし
戻り値:
　　なし

agt0_stop

概要:AGT0 停止関数
宣言:
　　void agt0_stop(void);
説明:
　・AGT0 のカウント停止
を行います
引数:
　　なし
戻り値:
　　なし

ー割り込み関数ー

intr_agt0_agti

概要:AGT0 割り込み関数
宣言:
　　void intr_agt0_agti(void);
説明:
　・100us 毎に SCI バッファの文字出力
を行います

6. サンプルプログラムの変数、定数値

サンプルプログラムで使用している変数に関して示します。

6.1. CTSU で使用しているグローバル変数

unsigned char g_key

現在（一番強く）押されているキー番号を示す変数。（0xff の時は、どのキーも押されていないと判断）
ctsu_result()の戻り値を g_key に代入。メインループ内で随時更新。

unsigned char g_key_state[ ]

現在押されているキーの情報を示す変数。
キー数（自己容量は 10，相互容量は 25）の配列変数となっており、押しているキーは 1，押されていないキーは 0 が格納されます。g_key 同様に、メインループ内で随時更新されます。

unsigned char g_sens[ ]

センサ値が格納される変数です。測定 ch 数（自己容量は 10,. 相互容量は 50）の配列変数で、メインループ内で随時更新されます。

unsigned char g_ref[ ]

リファレンス値が格納される変数です。測定 ch 数（自己容量は 10,. 相互容量は 50）の配列変数で、メインループ内で随時更新されます。

unsigned char g_err

CTSU 測定のエラーフラグが格納される変数です。測定毎に、その時点のエラーを示すレジスタ値と OR を取っていますので、過去に出たエラーを示しています。0 であれば、エラーが出ていないことを示します。

long g_initial_ref[ ]

初期リファレンス値で、CTSUSO 値の最適化の際参照されます。256 回測定した平均値が格納されています。（256 回の平均を計算するために、long 型で定義していますが、最終的には 2 バイト(0~65535)内の値となります）測定 ch 数の配列です。

long g_initial_sens[ ]

初期センサ値で、非タッチ時の基準値として使用されます。256 回測定した平均値が格納されています。(256 回の平均を計算するために、long 型で定義していますが、最終的には 2 バイト(0~65535)内の値となります)測定 ch 数の配列です。

unsigned char g_status[ ]

測定 ch 毎のステータスが格納される変数です。メインループ内で随時更新されます。

unsigned short g_diff[ ] [相互容量タイプのみ]

逆相－同相のセンサ値が格納される変数です。キー数(相互容量:25)の配列変数で、メインループ内で随時更新されます。

unsigned short g_initial_diff[ ] [相互容量タイプのみ]

逆相－同相の初期センサ値で、非タッチ時の基準値として使用されます。キー数配列変数です
(g_initial_sens(256 回の測定の平均値)から計算されます)

unsigned short g_ctsu_ctsuso1_fix[ ]

CTSUSO1(CTSURICOA)の最適値検索済みを示す変数です。キー数分の配列です。

※相互容量では、測定 ch は 50 ですが、オフセット電流(CTSUSO0, CTSUSO1)は、同相 ch の測定で最適化を行い(25ch 分)、逆相 ch の設定値は、同相 ch と同じ値を適用しています。

unsigned short g_ctsu_ctsuso0[ ]

測定 ch 毎の、CTSUSO0 レジスタ設定値を保存している変数です。CTSU WRITE 割り込みの際に、本変数がレジスタにコピーされます。測定 ch 数分の配列です。

unsigned short g_ctsu_ctsuso1[ ]

測定 ch 毎の、CTSUSO1 レジスタ設定値を保存している変数です。CTSU WRITE 割り込みの際に、本変数がレジスタにコピーされます。測定 ch 数分の配列です。

unsigned short g_data_index

測定 ch のインデックス変数です。CTSU READ 割り込み（測定完了）時に、インクリメントされ、CTSU FN 割り込み（全 ch の測定終了）時にクリアされます。

double g_touch_threshold[ ]

タッチ、非タッチの閾値を示す変数です。自己容量では、初期値 1.2、相互容量では 0.95 で、"t"コマンドで閾値の最適化を行うと更新されます。キー数分の配列です。

volatile unsigned char g_in_measure

測定中を示すフラグ変数です。
CTSU_STATE_IDLE(0) : 測定中ではない
CTSU_STATE_IN_MEASURE(1) : 測定中
CTSU_STATE_FINISHED(2) : 1 セット（全測定 ch）の測定完了

volatile unsigned short g_initialize_finished

初期化完了を示すフラグ変数です。
CTSU_INITIAL_UNFINISHED(0) : 初期化が終わっていない
CTSU_INITIAL_FINISHED(1) : 初期化済み

const unsigned char g_ctsu_valid_terminal[] = {TS03, TS04, TS05, TS06, TS07, TS08, TS09, TS10, TS11, TS12, TS13};

CTSU 機能で使用する端子の設定。

const char* g_ctsu_terminal[] = {"P407", "P408", "P409", "P410", "P411", "P412", "P413", "P414", "P415", "P708"};

CTSU 機能で使用する端子名の設定。

const char* g_ctsu_s16_str[] = {"9(TS03)", "4(TS04)", "8(TS05)", "3(TS06)", "2(TS07)", "7(TS08)", "1(TS09)", "6(TS10)", "0(TS11)", "5(TS12)", "NC", "NC", "NC", "NC", "NC", "NC"};

[自己容量のみ]自己容量タイプタッチキー基板(S16A)のキーパッドと TS??の関係（メッセージ表示等で使用）

const unsigned short g_ctsu_s16_index[] = {8, 6, 4, 3, 1, 9, 7, 5, 2, 0}; //keypad 0(TS11)->ch8, 計測 ch と keypad 番号の関係

[自己容量のみ]TSch(TS??)とキーパッド番号の関係。

const unsigned char g_ctsu_mutual_tx_terminal[] = {TS08, TS09, TS10, TS11, TS12};

[相互容量のみ]相互容量で使用する TX の端子定義。

6.2. その他で使用しているグローバル変数

unsigned long g_rtc_counter

RTC の周期割り込み（1/64 秒）毎にインクリメントされる変数です。1 秒に 1 回行いたい処理等で使用する事を想定しています。本サンプルプログラムでは、"m"コマンドの 2 秒毎に、測定値を表示する用途で使用しています。

unsigned long g_rtc_flag

RTC の周期割り込み（1/64 秒）毎にセット（1 を代入）される変数です。1/64 秒毎に行いたい処理で使用する事を想定しています。本サンプルプログラムでは、"s"コマンドで、1/64 秒毎に測定値を保存する用途で使用しています。

unsigned char g_sci_buf[ ]

SCI の出力のバッファリングに使用している変数です。sciWriteBuf 等の SCI 出力関数では、本バッファへのコピーのみ行っています。2048 の配列です（サイズの変更は問題ありません）。

unsigned short g_sci_buf_write_p

g_sci_buf[ ]用の書き込み箇所を示すインデックス（ポインタ）変数です。g_sci_buf[ ]は、リングバッファとして構成されており、書き込み済み位置を本変数で示します。

unsigned short g_sci_buf_read_p

g_sci_buf[ ]用の読み出し箇所を示すインデックス（ポインタ）変数です。本変数で示す位置までは、端末に表示済みです。

unsigned long g_rxi_flag

SCI 入力（キーボード）が合った場合にセットされるフラグ変数。

unsigned char g_rdr

SCI 入力（キーボード）が合った場合に、入力データが格納される変数。

6.3. CTSU で使用している定数値

```
#define CTSU_STATE_IDLE (0)                 //非測定中（測定開始待ち）
#define CTSU_STATE_IN_MEASURE (1)           //測定中
#define CTSU_STATE_FINISHED (2)             //全チャネルの測定が終了
```

フラグ変数 g_in_measure の状態を示す定数値。メインループ内では、CTSU_STATE_FINISHED になったタイミングで、キーの判定。CTSU_STATE_IDLE であれば、測定開始指示。CTSU_STATE_IN_MEASURE であれば何もお行いません。

```
#define CTSU_INITIAL_UNFINISHED (0)         //初期化完了前
#define CTSU_INITIAL_FINISHED (1)           //初期化完了後
```

初期化終了状態を示す定数値。

```
#define TS00 (0)
…
#define TS17 (17)
```

キー等を、TS??の名称でアクセスするための定義値。

#define TS_MAX (18) //RA6

TS 端子最大数。

//有効端子（TS03 - TS12 有効）
#define CTSU_VALID_TERMINALS (10)

TS??で使用している端子数。（自己容量、相互容量とも 10 端子を使用）

//TS03 - TS12(keypad 0~9)
#define CTSU_CH_NUM (10) //[自己容量]

//5x5 マトリックス，同相，逆相
#define CTSU_CH_NUM (50) //[相互容量]

測定 ch 数。自己容量の場合は、=キーパッド数となり、相互要領の場合は、キーパッド数×2（同相、逆相）となります。

//TX 端子数
#define CTSU_TX_TERMINALS (5) //[相互容量]

[相互容量のみ]マトリックスキーを構成する、TX 端子数。

//5x5 マトリックス
#define CTSU_KEY_NUM (25) //[相互容量]

[相互容量のみ]キー数。

//マトリックスの列数
#define CTSU_COLUMN_NUM (5) //[相互容量]

[相互容量のみ]マトリックスキーを構成する列数。

//モニタ表示の頻度（n 秒に 1 回）
#define CTSU_MONITOR_PERIOD (2)

"m"コマンドでの表示更新頻度。

```
//リファレンス ICO の電流オフセット調整を省略する
//#define CTSU_REF_OFFSET_OMIT     //[自己容量]
#define CTSU_REF_OFFSET_OMIT     //[相互容量]
```

本定数定義時は、起動後のリファレンス電流(CTSUSO1 CTSURICOA)は、固定値とします。(電流オフセット、CTSUSO0.CTSUSO レジスタの最適化は行います)。

※自己容量は 10 キーに対し、相互容量は 25 キーと多くなるためデフォルトでは、リファレンス電流の最適化は自己容量タイプのみで行っています

```
#define DEBUG_PRINT
```

定義時は端末(仮想 COM ポート)に対するメッセージの表示が多くなります。

```
#define CTSU_TSCAP "P205"
#define TSCAP_PDR R_PORT2->PDR_b.PDR5         //P205
#define TSCAP_PODR R_PORT2->PODR_b.PODR5      //P205
```

TSCAP 端子の定義です。

```
//計測回数=1 回(6bit)
#define CTSU_CTSUSNUM (0)
```

1 回の測定での、測定 ch 毎の計測回数です(初期値は 1 回)。

```
//センサ ICO 入力電流オフセット調整(10bit, 0-1023), CTSUSO 検索刻み(0~1023 を 1 刻みで検索)
#define CTSU_CTSUSO_TS_SEARCH_DELTA (1)
```

電流オフセット調整時、CTSUSO をこの値(初期値 1)ずつ増やしながら最適値を検索します。初期化に掛かる時間を節約したいときはこの数値を増やしても問題ありません。

```
//CTSUSO1 設定
//ゲイン=100%(2bit)
#define CTSU_CTSUICOG (0)
```

電流ゲイン調整値(初期値は 0: 100%)です。

//ベースクロック設定=4 分周(5bit)(PCLKB=60MHz, 60MHz/4(CTSUCLK)/4(CTSUSDPA)=3.75MHz)
#define CTSU_CTSUSDPA (1)

動作クロック分周比です。初期値は 1: 3.75Mz 設定です。

//リファレンス ICO 電流調整(8bit, 0-255), デフォルト値
#define CTSU_CTSURICOA (65)

CTSU_REF_OFFSET_OMIT を定義して、CTSURICOA の最適化を行わない場合の設定値です。CTSU_REF_OFFSET_OMIT 未定義の場合は使用されません。

//リファレンス ICO 電流調整刻み（0-255 を 5 刻みで検索）
#define CTSU_CTSURICOA_DELTA (5)

CTSURICOA の最適値検索時の検索増分値です。CTSURICOA のループを回すのは、比較的時間が掛かるので、初期値は 5 ずつ増やす事としています。

//非タッチ時のばらつき 3σが非タッチセンス値の 5%未満である事
#define CTSU_NO_TOUCH_SCATTERING (0.05)

電流オフセット値最適化時、測定値のばらつきが 5%(0.05)未満となる点を探します。この値を大きくすると、CTSURICOA は小さな値。逆に、小さくすると、CTSURICOA は大きな値となります。CTSU_REF_OFFSET_OMIT 未定義の場合は使用されません。

//判定閾値（初期値）
#define CTSU_DEFAULT_THRESHOLD (1.2) //[自己容量]
#define CTSU_DEFAULT_THRESHOLD (0.95) //[相互容量]

"t"コマンドで、閾値を最適化する前のデフォルトの閾値です。

6.4. その他で使用している定数値（抜粋）

#define SCI_BUF_SIZE 2048

SCI の出力バッファサイズです。初期値は、2048 文字（バイト）です。

7. 備考

7.1. ボード上の LED

ボードには、モニタ LED D7(P106)が搭載されています。タッチキーサンプルプログラム実行時はこの LED は点滅しています。これは 1 セットの測定（自己容量:10ch, 相互容量:50ch）の測定終了毎（ctsu_result()実行のタイミング）で、LED の点灯、消灯を切り替えています。P106 の端子(J2-27)をモニタすると、1 回の測定にどの程度時間が掛かっているのかがモニタ可能です。

実際にモニタすると、自己容量タイプでは、1 回の測定に 850us 程度。相互容量タイプでは、4.3ms 程度掛かっています。

取扱説明書改定記録

バージョン	発行日	ページ	改定内容
REV.1.0.0.0	2020.6.4	—	初版発行
REV.1.0.1.0	2020.6.24	P64,P68	誤記訂正

お問合せ窓口

最新情報については弊社ホームページをご活用ください。

ご不明点は弊社サポート窓口までお問合せください。

株式会社 北斗電子

〒060-0042 札幌市中央区大通西 16 丁目 3 番地 7

TEL 011-640-8800　FAX 011-640-8801

e-mail：support@hokutodenshi.co.jp (サポート用)、order@hokutodenshi.co.jp (ご注文用)

URL:http://www.hokutodenshi.co.jp

ルネサス エレクトロニクス RA6M2/RA6M1 搭載
HSB シリーズマイコンボード向け評価キット

RA6M2-144 タッチキー評価キット
RA6M1-100 タッチキー評価キット
[ソフトウェア編]
取扱説明書

株式会社 北斗電子

 2020 年 6 月 24 日改訂 REV.1.0.1.0 (200624)