# RX140 タッチキー評価キット
# RL78/G23 タッチキー評価キット
# RA2L1 タッチキー評価キット

## [ソフトウェア編]
## マニュアル

-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.1.0.2.0

ー目　次ー

# 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。

# 概要

本書は、
「RX140 タッチキー評価キット」
「RL78/G23 タッチキー評価キット」
「RA2L1 タッチキー評価キット」
付属 CD 内のサンプルプログラムについて解説を行うものです。

RX140, RL78/G23, RA2L1 マイコンはルネサスの新しいタッチキーユニットである CTSU2/CTSU2L を搭載しています。

本書では、キット付属の 2 種類のタッチキー基板、「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)」のキー読み取りのサンプルプログラムを示しています。

ルネサスエレクトロニクスでは、タッチキーユニット(CTSU, CTSU2)制御用のミドルウェアとして、QE-CapTouch をリリースしています。複雑なタッチキーユニットの初期設定や、タッチした際の閾値のチューニングや測定値のモニタ等が行える高機能なソフトとなりますが、QE-CapTouch が生成したプログラムコードは複雑で半分ブラックボックス化されている様なイメージです。

本書で解説するサンプルプログラムはキーパッドに触れた場合に、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。ゼロからタッチキーユニットのプログラムを作ってみたいという用途向けに、単純なプログラムコードとなっており、タッチキー制御の原理を理解するのに適していると考えます。

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、ルネサスエレクトロニクス製ツールである QE-CapTouch を使用して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。

本書とは別なマニュアルで、QE-CapTouch の使用例を記載していますので、そちらも併せて参照ください。

# 1. 本書で説明する範囲

## 1.1. CD フォルダ構成

| | | |
|---|---|---|
| SOURCE/<br>SOURCE_UTF8/ | ctsu2/ | タッチキー制御 |
| | intr/ | 割り込み設定（RA で使用、RX, RL78 では未使用） |
| | lcd_1602/ | LCD 制御 |
| | mcu_type/ | マイコンタイプ設定 |
| | sci/ | 通信(UART)設定 |
| | timer/ | タイマー処理 |

SOURCE, SOURCE_UTF8 フォルダ以下に、各種ソースコードが入っています。

※ SOURCE, SOURCE_UTF8 フォルダ以下のファイルは、文字コードが Shift-JIS(SOURCE), UTF-8(SOURCE_UTF8)の違いのみで、内容は同一です

開発環境として、CS+を使っている場合は、SOURCE 以下のファイルを使用して、e2studio を使っている場合は、SOURCE_UTF8 以下のファイルを使ってください。

| | | | |
|---|---|---|---|
| PROJECT/ | RX/ | RX140_CTSU2_SAMPLE/ | RX140 向け CS+プロジェクト |
| | RL78/ | RL78_G23_CTSU2_SAMPLE/ | RL78/G23 向け CS+プロジェクト |
| | | RL78_G23_CTSU2_SAMPLE_COM_DEBUG/ | RL78/G23 向け CS+プロジェクト（COM ポートデバッグ版） |
| | RA/ | RA2L1_CTSU2_SAMPLE.zip | RA2L1 向け e2studio プロジェクト |

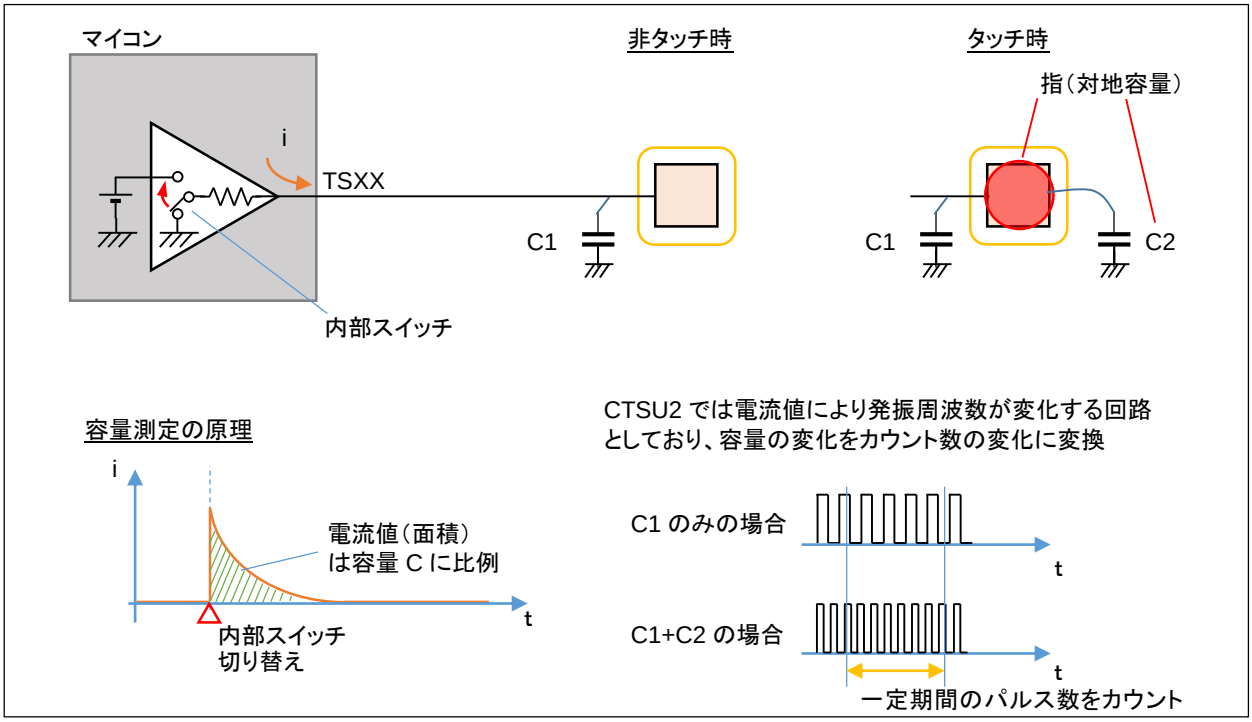
上記ソースコードを含むプロジェクトが、PROJECT フォルダに格納されています。

RX, RL78 は CS+のプロジェクトフォルダですので、PC の適当なフォルダにコピーして、「プロジェクト名.mtpj」ファイルをダブルクリックして、CS+を起動してください。RA は、e2studio のプロジェクトのアーカイブとなっていますので、ワークスペースにインポートしてください。

# 2. タッチキーの動作原理

## 2.1. 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）

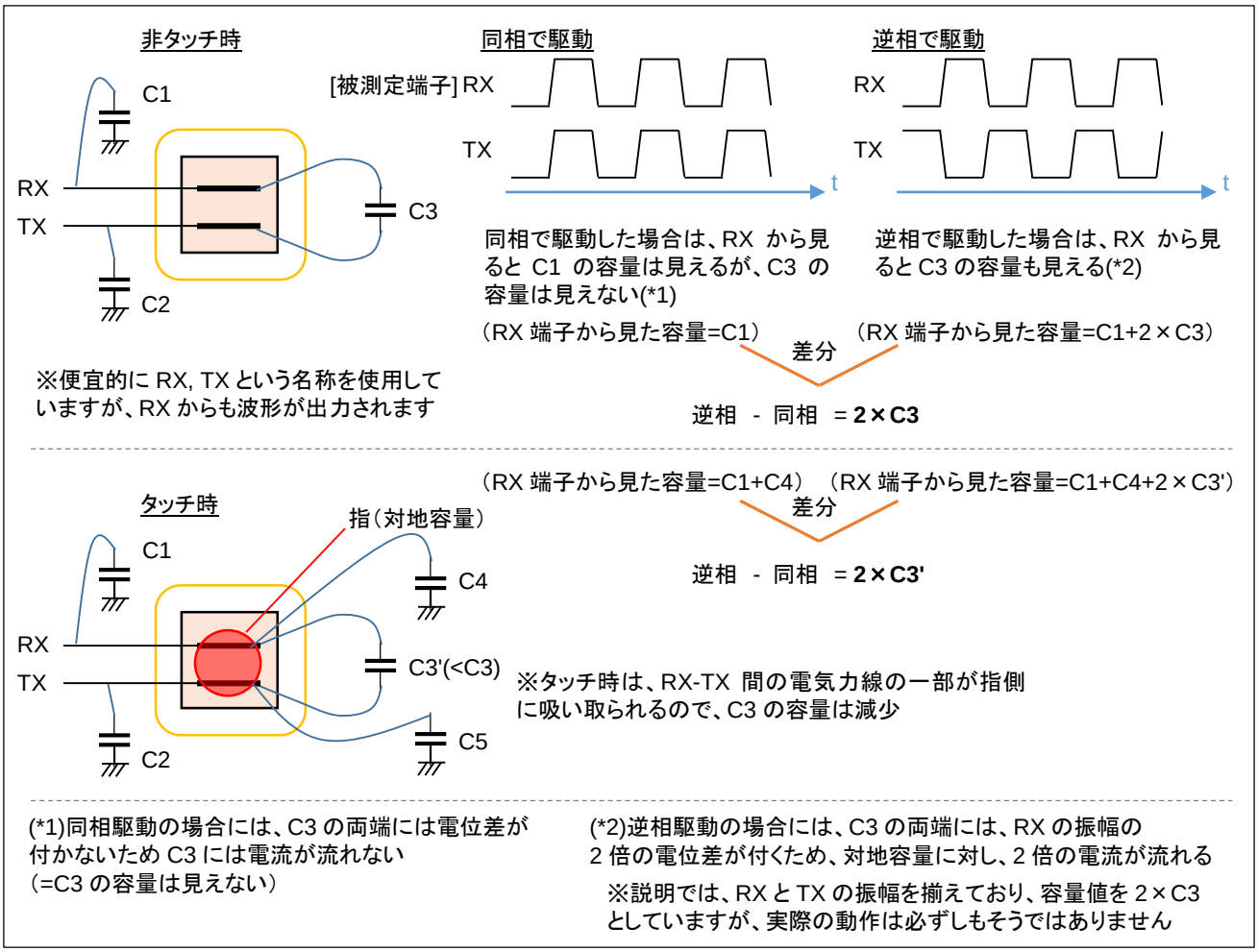
TSxx のラインを L→H に遷移させると、端子からは容量に比例した電流(i)が流れ出します。電流を測定する事により、TSxx ラインにつながっている容量の大きさを測る仕組みです（詳細は、マイコンのハードウェアマニュアルや、ルネサスエレクトロニクスより提供されている、CTSU 測定の原理の資料を参照ください）。容量の変化は、カウント数の変化という形で、数値に変換されます。カウント数が少ないときは、容量が小さく、カウント数が大きいときは容量も大きいといった具合です。

センサ値（カウント値）をプログラムで、定期的にモニタする事で、タッチした場合は容量値が増加し、結果的にセンサ値＝カウント値も増加するため、タッチの有無を検出する事ができます。

・非タッチ時の TSXX 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 TSXX 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

## 2.2. 相互容量タイプ

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。

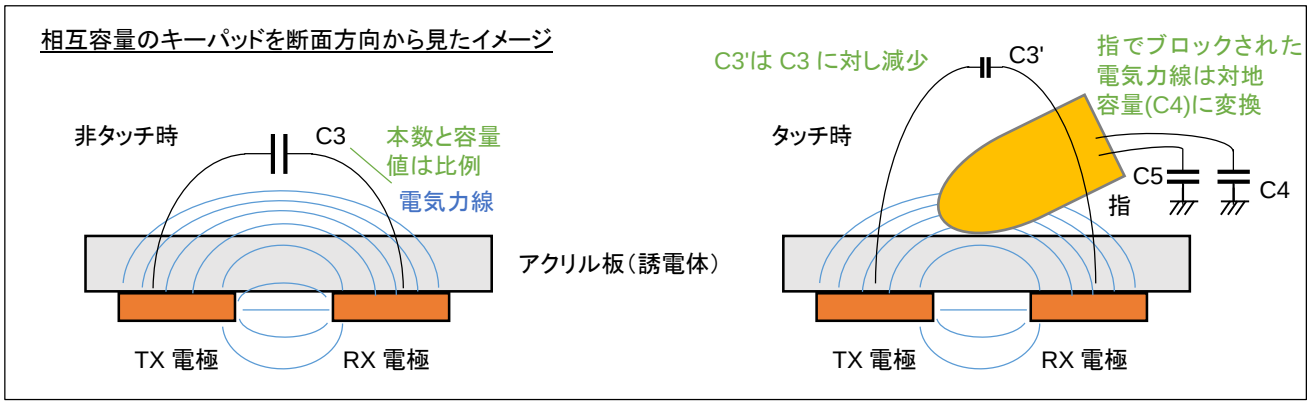



非タッチ時、RX, TX を同相で駆動すると、C3 両端の電位が等しくなり、C3 にはチャージ電流が流れません。C1 側にはチャージ電流が生じるため、RX 端子から流れ出る電流は、C1 に比例します。次に、RX と TX を逆相で駆動すると、C3 には RX の振幅の 2 倍の電圧がかかるため、C3 が対地容量の場合に対し RX 端子から流れ出る電流も 2 倍となります。結果的に C1+2×C3 の容量が見えます。

ここで、「逆相の測定値」から「同相の測定値」を差し引けば、C1 の影響はキャンセルされ、結果的には 2×C3 に相当する観測値が得られます。

※ここでは、RX と TX の振幅が等しいとして、原理を説明しています

タッチキーボードにタッチした際の容量の変化は以下の様になります。




非タッチ時は、2 つの電極間に電気力線が飛んでおり、この電気力線の本数と電極間の容量(C3)は比例します。

一方タッチ時、指はほぼ理想的な絶縁体である空気と比べると、導体と見なす事ができます。RX 電極と指の間の電気力線は、C4（対地容量）になります。（TX 電極と指の間の電気力線は、C5）

タッチ前に対して、RX 電極－TX 電極間の電気力線は減少します。その結果、RX-TX 間の容量(C3')はタッチ前(C3)に対して減少します。

タッチ時も、非タッチ時と同様に「逆相の測定値」から「同相の測定値」を差し引けば、2×C3'に相当する測定値が得られます。

C3 と C3'の差分で、非タッチ・タッチの判定を行う事ができます。

相互容量方式では、自己容量方式と異なりタッチした際、測定値（の演算結果）の減少が見られる事となります。

・非タッチ時の TX 同相駆動時の RX のセンサ値を取得(*1)
・非タッチ時の TX 逆相駆動時の RX 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

以上が、自己容量と相互容量で非タッチとタッチの判定を行う、基本的な考え方となります。

# 3. タッチキーユニット(CTSU2)の設定

タッチキーユニット(CTSU2)は、数多くのレジスタがあり、感度や動作マージンををトリミングできる様になっています。細かなレジスタ値のチューニングを行いたい場合は、QE-CapTouch に任せる事とし、本書で扱うのは、動作させる上で必須となる設定に留め、シンプルな構成とする事と致します。

## 3.1. 初期設定レジスタ

### 3.1.1. CTSUCRA

基本的な動作を決めるレジスタです。

RX : CTSU.CTSUCRA 32bit レジスタ
RA : R_CTSU->CTSUCRA 32bit レジスタ
（CTSUCRAL, CTSUCRH 16bit レジスタとしてもアクセス可）
（CTSUCRA0~3 8bit レジスタとしてもアクセス可）
RL78: CTSUCRAL, CTSUCRH 16bit レジスタ

レジスタの中身（ビットフィールド）は、マイコン種に拠らずほぼ同じですが、アクセス時の bit 数やアクセスする際のレジスタ名称はマイコン種に依存します。本サンプルプログラムでは、マクロ、関数を用意してどのマイコン種でも

CTSUCRA(INIT, 1);　　//レジスタ名(ビットフィールドシンボル，設定値)

の様に設定しています。（マイコン種に依存せず同じ構文としています）

| レジスタシンボル | 設定値 | 備考 |
|---|---|---|
| PUMPON(*1) | 0 | VCC を 5V で使用する場合<br>VCC を 4.5V 未満で使用する場合は 1 に設定する |
| TXVSEL[1:0]<br>(RX, RA) | 0b01 | 相互容量方式で送信用電源として VCC を使用 |
| TXVSEL2,TXVSEL<br>(RL78) | | |
| ATUNE2 | 0 | 電流計測レンジ 40uA<br>（20uA, 40uA, 80uA の内真ん中を選択） |
| ATUNE1 | 1 | |
| CLK | 0b00 | RX,RA:PCKLB=24MHz, RL78:FCLK=32MHz をクロックソースとして使用 |
| ATUNE0(*1) | 0 | 通常電圧動作モード<br>VCC を 2.4V 未満で使用する場合は低電圧モード 1 に設定する |
| MD1(*2) | 0 | 自己容量タイプ(S16A)基板を使用する場合：計測回数 1 回 |
| | 1 | 相互容量タイプ(D55A)基板を使用する場合：計測回数 2 回（同相・逆相） |
| MD0 | 1 | マルチスキャンモード（全 ch を通しで計測する） |
| LOAD[1:0] | 0b00 | 2.5uA 定電流負荷モード（通常計測用） |

| レジスタシンボル | 設定値 | 備考 |
|---|---|---|
| POSEL[1:0] | 0b00 | 非計測チャンネル：Low 出力 |
| SDPSEL | 0 | ランダムパルス |
| PCSEL<br>(RX,RA) | 0 | ブースト回路クロック選択：センサ駆動パルスを選択 |
| FCMODE<br>(RL78) | 0 | センサユニットクロックを周波数拡散クロックとして使用 |
| STCLK | 0x17(RX,RA)<br>0x1f(RL78) | 24MHz/((23+1)×2) = 500kHz 設定(RX, RA)<br>32MHz/((31+1)×2) = 500kHz 設定(RL78) |
| DCMODE | 0 | 通常モード（静電容量計測モード） |
| DCBACK | 0 | 電流計測フィードバック：TSCAP を選択 |

初期設定後、タッチキーユニットを動作させる際に、下記レジスタを設定します。

| レジスタシンボル | 設定値 | 備考 |
|---|---|---|
| CSW | 1 | TSCAP 端子有効 |
| PON | 1 | CTSU 計測電源起動 |

(*1)の設定はボードを 5V ではなく、低い電圧で使用する際に値を変えてください。
(*2)接続するタッチキーボード、タッチキー方式（自己容量、相互容量）によって選択してください。

### 3.1.2. CTSUCRB

| レジスタシンボル | 設定値 | 備考 |
|---|---|---|
| PRRATIO[3:0] | 0x3 | クロックパルス相制御周波数（推奨値設定) |
| PRMODE[1:0] | 0b10 | 基本パルス数：62 パルス |
| SOFF | 0 | スペクトラム拡散 ON |
| PROFF | 0 | 駆動パルス相制御に乱数を用いる |
| SST[7:0] | 0x10 | センサ安定待機時間 |
| SSMOD[2:0] | 0x0 | スペクトラム拡散変調周波数：125kHz |
| SSCNT[1:0] | 0b11 | SUCLK 周波数調整 |

### 3.1.3. CTSUCHACA/CTSUCHACB

計測を有効化する TSxx の端子を有効化するレジスタです。

・RX, RA

CTSUCHACA : TS0~TS31 の有効化指定
CTSUCHACB : TS32~TS63 の有効化指定

例えば、TS0, TS2 を有効化する場合は、CTSUCHACA=0x00000005（bit0 と bit2 を 1）とします。

・RL78

CTSUCHACAL : TS0~TS15 の有効化指定
CTSUCHACAH : TS16~TS31 の有効化指定
CTSUCHACBL : TS32~TS47 の有効化指定
CTSUCHACBH : TS48~TS63 の有効化指定

RL78 では、16bit レジスタで TS 端子の有効化を設定します。

### 3.1.4. CTSUCHTRCA/CTSUCHTRCA

相互容量方式では、TX と RX の端子を指定する必要があり、このレジスタで TX の端子を指定します。

・RX, RA

CTSUCHTRCA : TS0~TS31 の TX 端子指定
CTSUCHTRCB : TS32~TS63 の TX 端子指定

・RL78

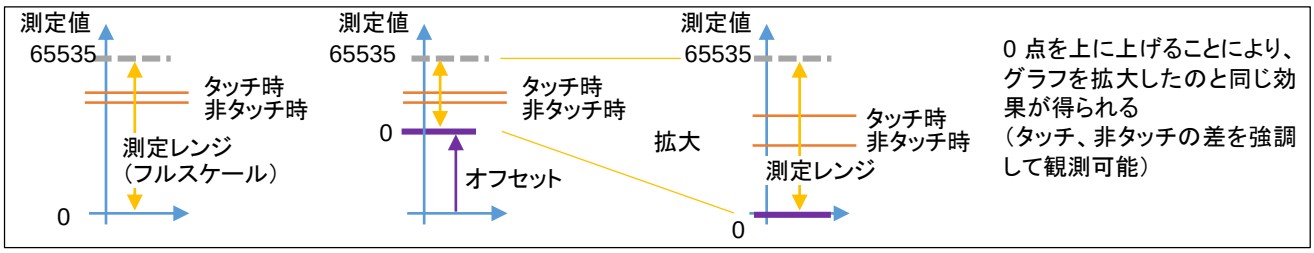
CTSUCHTRCAL : TS0~TS15 の TX 端子指定
CTSUCHTRCAH : TS16~TS31 の TX 端子指定
CTSUCHTRCBL : TS32~TS47 の TX 端子指定
CTSUCHTRCBH : TS48~TS63 の TX 端子指定

RX 端子指定 ＝ CTSUCHACx で有効化した端子 － CTSUCHTRCx で指定した端子
となります。

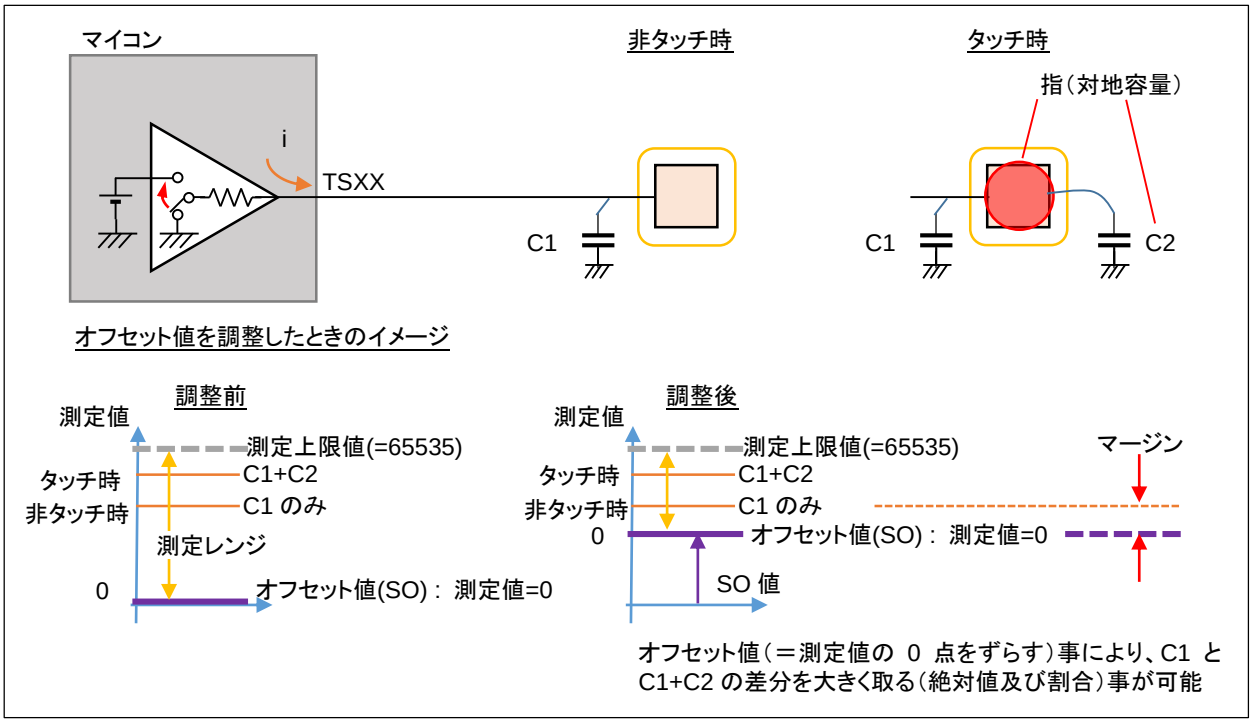
## 3.2. センサオフセット調整

CTSUSO.SO[9:0]で、センサー電流のオフセット値を調整します。

オフセット値は非タッチ時の容量（測定値）をキャンセルする意味で用いられます。

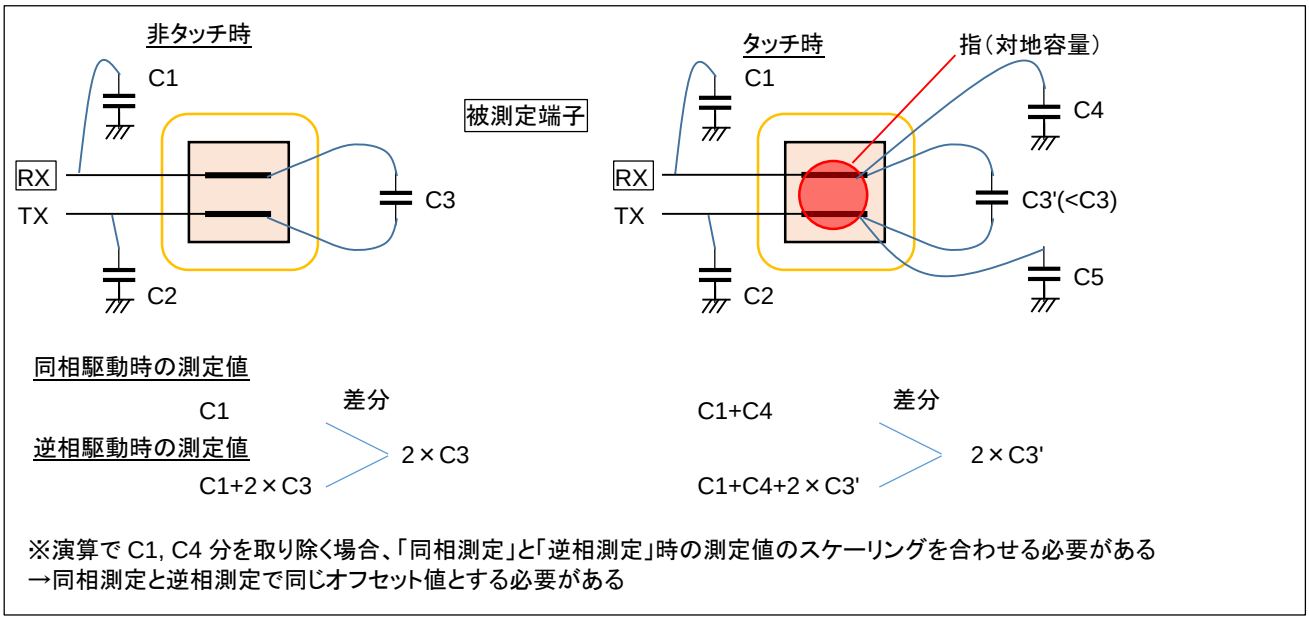


・自己容量の場合




適切なオフセット値（SO 値）を設定する事により、タッチ、非タッチの差分を測定レンジ内に大きく取る事が可能で、理想的には C1 分をオフセットでキャンセルしてしまえば良いという事となります。しかし、オフセットを大きく取りすぎた場合、C1 の値を超えたオフセットを設定すると、今度はタッチと非タッチ時の測定値の差分が減少する事となります。特性の変動も加味すると、SO 値は非タッチ時(C1)の測定値=0 となる値より多少（マージンを持たせた）小さい値とする必要があると考えます。

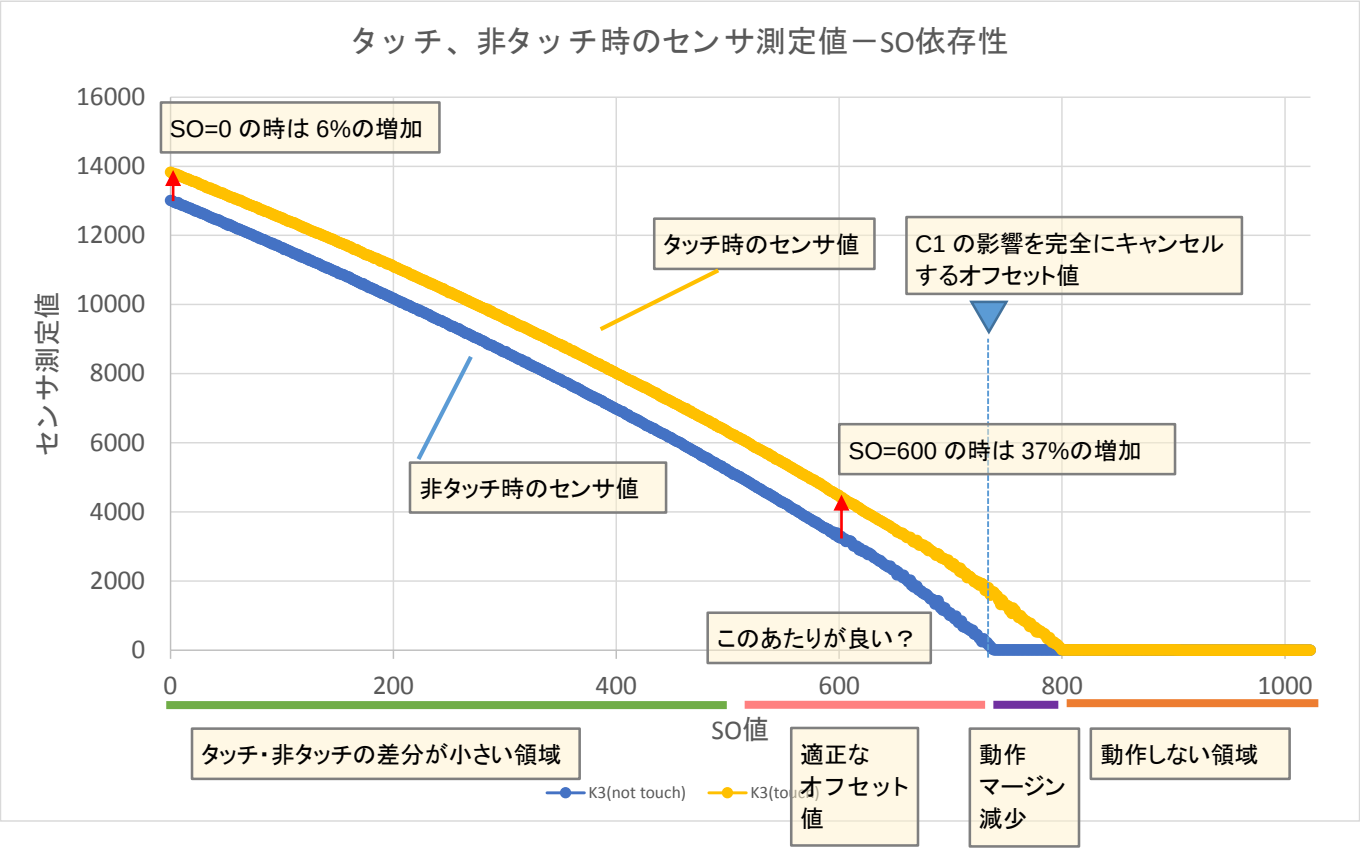
・相互容量の場合




相互容量の場合は、同相測定時の C1 に相当するオフセットをかければ良い。SO 値は自己容量と同じ考え方で同相測定値で決めれば良い、と考えます。

（逆相測定時のオフセット設定値は、同相測定で決めたオフセット値とする）

電流オフセット値(SO 値)と、センサ測定値(CTSUSC)の関係の実測例を示します。
SO 値（オフセット値）、10bit で 0~1023(0x3FF)の範囲で設定が可能です。




上記は電流オフセット値(SO)を振った際の、非タッチ時とタッチ時のセンサ測定値（CTSUSC レジスタ値、容量値に対応）をプロットしたものです。

SO 値と測定値に関して何点か抜き出して示します。

| SO 値 | 非タッチ時の測定値 | タッチ時の測定値 | 差分 | 備考 |
|---|---|---|---|---|
| 0 | 13008 | 13826 | 818 | |
| 200 | 10183 | 11118 | 935 | |
| 400 | 6976 | 8009 | 1033 | |
| 600 | 3267 | 4478 | 1211 | |
| 700 | 1069 | 2495 | 1426 | |
| 740 | 4 | 1631 | 1627 | 限界までオフセット |
| 800 | 5 | 5 | 0 | 測定値以上のオフセット |

SO 値の小さな領域では、タッチと非タッチ時の測定値の差分が大きくありません。SO 値を大きくしていくと、差分も増加します。しかし、一定の数値を超えると、今度は測定値が取れなく（測定結果が 0 に）なります。

測定値が取れる限界付近ですと、差分の数値も増加率も大きく取れますが、経時変化（電源電圧や、温度等の変化）があると、動作しなくなる（上記のグラフが全体的に左にシフトするケース）可能性も考えられます。
→限界までオフセットをかける訳ではなく、限界より多少左側の（SO 値が小さい）領域で使用

# 4. センサ値の測定

タッチ検出端子（マイコン TSxx）に接続されたタッチキー電極の容量値（に相当するセンサ測定値)は、

CTSU.CTSUSCNT.BIT.SC (RX)
CTSUSC (RL78)
R_CTSU->CTSUSC (RA)

を読み出す事で行えます。この、センサ値を格納するレジスタは 1 つしかありませんので、複数のチャネル(*1)の測定を行った場合は、順次読み出していく事となります。

(*1)自己容量タイプでは、タッチキー電極の数＝測定チャネル数となります。相互容量タイプでは、タッチキー電極の数×2（同相、逆相）＝測定チャネル数となります。

センサ値の測定の流れは以下の様になります。（ここでは、流れだけを示します。具体的な処理は後述。）

(1)CTSUCRA.STAT に 1 をセット（測定開始）
(2)CTSUWR 割り込みが入る
(3)1 番目の測定チャネルに対応する（測定チャネル毎のオフセット）SO 値を CTSUSO に書き込む
(4)CTSURD 割り込みが入る
(5)1 番目の測定チャネルのセンサ値 CTSUSC を読み出す
(6)(2)~(5)の処理を測定チャネル分行う
(7)CTSUFN 割り込みが入る

上記がマルチスキャンモード（測定対象のチャネルを全部測定する）の動作となります。処理を開始すると、所定の割り込みが入りますので、割り込み時にレジスタに値をセット、レジスタ値の読み出し処理を実行します。

なお、センサ値は、16bit(0~65535)の値を取ります。

# 5. タッチ判定を含めた全体の流れ

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。

## (1)CTSU2 初期化

・TSxx 端子設定
・TSCAP 端子設定
・CTSU レジスタ設定

TSxx 端子は、タッチキー電極につながる、マイコンの端子設定です。TACAP は、タッチキー機能を使う上で必要な LPF（ローパスフィルタ）に使う端子で、TSCAP 端子には 10nF(0.01uF)のコンデンサ（対地）を接続し、その端子を TSCAP に割り当てる必要があります。

## (2)電流オフセット値の設定

・CTSUSO.SO レジスタ値の最適化

## (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。

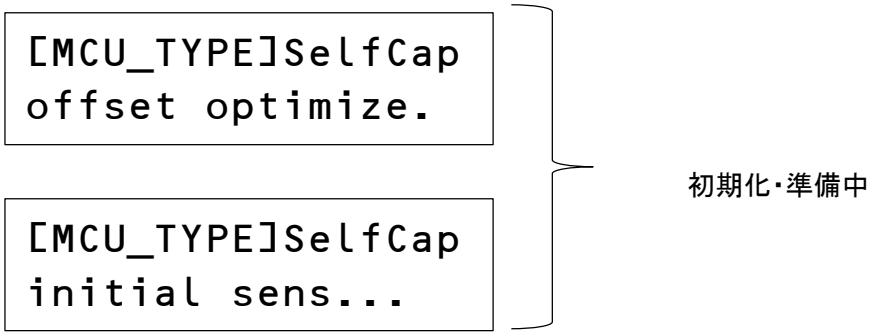
(4)でタッチ時のセンサ値を取得できれば、判定の閾値を非タッチ時とタッチ時のセンサ値の中間に設定可能です（＝動作マージンを大きく取れると考えます）。但し、実際のアプリケーションでは、非タッチ時のセンサ値は初期化の時（電源投入時）に測定すれば良いと思いますが、タッチした時の基準となるセンサ値を取得するのは出来ないケースの方が多いのではないかと思われます。（電源投入後、一通りキーにタッチしないと使えないシステムは非常に使い勝手が悪いと思います。）

設計したボードで、非タッチ、タッチ時のセンサ値の差分を観測し、その値から判定の閾値を決めるというのが、一般的な手法になるかと考えます。

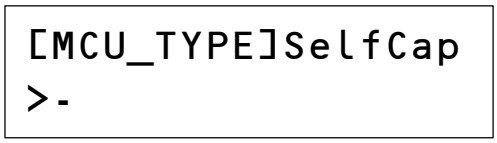
# 6. サンプルプログラムの実行例

## 6.1. 自己容量キーパッド(S16A)

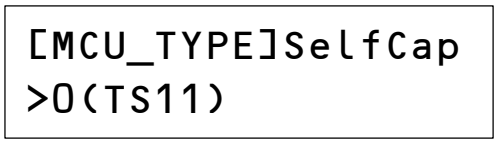
プログラムを起動すると、LCD 画面に




上記表示出力後、2 行目が－の表示となります。




この時点で、初期化は終わっていますので、キーパッドに触れてみてください。




触れたキーパッドの番号(0~9,A~F)と、カッコ内にキーパッドにつながっている TXxx の端子名が表示されます。
※RA では 0~9 が有効、A~F は無効です

LCD には、タッチしているキーの情報のみ表示されます。

シリアル端末（11,5200bps）は、以下の表示が出力されます。

```
Copyright (C) 2021 HokutoDenshi. All Rights Reserved.

HSBRL78G23-100 CTSU(TouchKey) Self Cap Sample program.

Please do NOT touch key pad!

offset optimize & no-touch data scanning...
SO = 20
 （中略）
SO = 300
  4(TS23)  SO setting value = 315
SO = 320
  3(TS25)  SO setting value = 324
  9(TS22)  SO setting value = 327
  B(TS29)  SO setting value = 327
  8(TS24)  SO setting value = 334
SO = 340
  2(TS30)  SO setting value = 348
  C(TS27)  SO setting value = 350
  7(TS31)  SO setting value = 351
 （中略）
SO = 540
  D(TS26)  SO setting value = 542

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
  h : help
>
```

マイコンボード種
（キットにより変わります）

表示される数値はマイコンボード毎に異なります
（オフセット最適化の過程）

テストメニュー

キーボードからコマンドの入力が可能です。

〇m コマンド（測定値のモニタ）

現在の測定値（デフォルトでは、2 秒毎に表示を更新）を表示します。

※RA では 0~9 までの 10 個の測定値が表示されます

```
sens value monitor(press any key to exit)>
key :    0(TS11)  1(TS32)  2(TS30)  3(TS25)  4(TS23)  5(TS10)  6(TS33)  7(TS31)  8(TS24)  9(TS22)  A(TS21)  B(TS29)  C(TS27)  D(TS26)  E(TS20)  F(TS28)
initial :  5860     5378     5260     5061     4854     6130     5501     5295     5182     4968     5346     5131     5343     6856     5483     5426

sens value :
         5952    5435    5246    4969    4742    6284    5422    5340    5026    4973    5378    5111    5337    6887    5469    5506
         7786*   5448    5364    5069    4827    6257    5555    5298    5149    5022    5321    5153    5284    6874    5464    5408
         5962    5410    5260    4936    4740    6164    5440    5280    5168    5011    5276    5140    5359    6727    5515    5401
         5995    7117*   5258    5084    4938    6355    5642    5298    5156    5032    5409    5189    5204    6907    5517    5406
         5780    5259    5285    5114    4805    6172    5510    5207    5092    5014    5308    5065    5336    6793    5426    5445
         5992    5501    6956*   6720*   5028    6243    6020    5757    5313    4984    5362    5138    5280    6813    5643    5525
         6074    5433    7074*   6805*   4912    6226    5926    5760    5313    5107    5456    5213    5435    6707    5640    5479
```

表示の左側のみ拡大

```
sens value monitor(press any key to exit)>
key :     0(TS11)  1(TS32)  2(TS30)  3(TS25)  4(TS23)  5(TS10)
nitial :  5860     5378     5260     5061     4854     6130

sens value :
          5952     5435     5246     4969     4742     6284
          7786*    5448     5364     5069     4827     6257
          5962     5410     5260     4936     4740     6164
          5995     7117*    5258     5084     4938     6355
          5780     5259     5285     5114     4805     6172
          5992     5501     6956*    6720*    5028     6243
          6074     5433     7074*    6805*    4912     6226
```

キーパッド毎の初期
（タッチしていないときの）測定値

キーパッド 0 にタッチ

キーパッド 1 にタッチ

キーパッド 2 と 3 にタッチ

時系列（2 秒毎に 1 行追加）

・タッチしているキーの測定値が増加します
・増加値が、一定以上の場合は、数値の右に(*)が付きます（タッチしているキーである印）
・複数のキーにタッチした場合でも、タッチの判定がなされます（LCD に表示されるのは代表値 1 つです）
・キーボードから何かキーを押すと表示は終了します

〇t コマンド（閾値のチューニング）

```
threshold value optimize...

Please touch key pad -> 0(TS11)
& PC keyboard enter

touch data scaning...

 key scan end.

  key pad(TS ch) = 0(TS11)

  touch(ave) = 7918 no-touch(ave) = 5860

Please touch key pad -> 1(TS32)
& PC keyboard enter

…(中略)

Please touch key pad -> F(TS28)
& PC keyboard enter

touch data scaning...

 key scan end.

  key pad(TS ch) = F(TS28)

  touch(ave) = 7653 no-touch(ave) = 5426

 tuning end.
```

キーパッド 0 にタッチして
キーボードのリターンキーを押してください

タッチした時の測定値=7918
非タッチ時の測定値=5860 です

次はキーパッド 1 をタッチして
同様にリターンを押す

最後にキーパッド F をタッチして
同様にリターンを押す

全てのキーの閾値の最適化終了

タッチ判定の閾値を最適化するコマンドです。

※キーにタッチし、キーボードのリターンキーを押した後次の「Please touch key pad」の表示が出るまではタッチを続けてください

※ESC キーの入力か、CTRL-C の入力で処理を中断できます

タッチした時の測定値（256 回測定の平均）と非タッチ時の測定値（起動時に測定）の真ん中に閾値を設定します。

閾値の最適化前は、閾値はどのキーも同じ値となっていますが、このコマンドを使用すると実際にタッチした時の測定値を元に閾値を設定します。

〇p コマンド（初期値と閾値の表示）

```
print initial reference value>

TSxx   :   SO :  Sens : threshold
0(TS11):  410 :  5860 : 1.176
1(TS32):  360 :  5378 : 1.160
2(TS30):  348 :  5260 : 1.165
3(TS25):  324 :  5061 : 1.170
4(TS23):  315 :  4854 : 1.189
5(TS10):  429 :  6130 : 1.163
6(TS33):  364 :  5501 : 1.193
7(TS31):  351 :  5295 : 1.208
8(TS24):  334 :  5182 : 1.221
9(TS22):  327 :  4968 : 1.234
A(TS21):  357 :  5346 : 1.176
B(TS29):  327 :  5131 : 1.180
C(TS27):  350 :  5343 : 1.176
D(TS26):  542 :  6856 : 1.159
E(TS20):  364 :  5483 : 1.198
F(TS28):  366 :  5426 : 1.205
```

・SO（オフセット値）
・Sens（非タッチ時の測定値）
・threshold（判定閾値）
の表示を行います。

上記表示例は、t コマンドで閾値の最適化を行った場合です。（閾値の最適化を行わない場合は、threshold 値は一律 1.100(+10%)です。）

〇s コマンド（最大－最小、偏差の表示）

```
sens statistics(measure 5 sec)>

sens value statistics data

key(TSxx) :   Ave (   Max -   Min , 3sigma )
0(TS11)   :  5908 (  6027 -  5768 , 177 )
1(TS32)   :  5410 (  5555 -  5208 , 210 )
2(TS30)   :  5972 (  7118 -  5115 , 2577 )
3(TS25)   :  5115 (  5288 -  4932 , 211 )
4(TS23)   :  4866 (  4985 -  4735 , 150 )
5(TS10)   :  7769 (  7915 -  7644 , 175 )
6(TS33)   :  5722 (  5913 -  5506 , 309 )
7(TS31)   :  5383 (  5561 -  5198 , 274 )
8(TS24)   :  5217 (  5315 -  5090 , 147 )
9(TS22)   :  4993 (  5094 -  4843 , 169 )
A(TS21)   :  5355 (  5471 -  5235 , 164 )
B(TS29)   :  5127 (  5230 -  4992 , 151 )
C(TS27)   :  5326 (  5443 -  5165 , 171 )
D(TS26)   :  6920 (  7102 -  6701 , 262 )
E(TS20)   :  5481 (  5594 -  5348 , 162 )
F(TS28)   :  5469 (  5578 -  5229 , 181 )
```

2 のキーは測定期間の半分程度の時間タッチ
→最大－最小の差分や 3σ 値が大きい

5 のキーは測定期間タッチ
→測定値は大きな値となっているが測定値のばらつき（3σ 値）はそれ程大きくない

5 秒間測定を行い、測定期間内の最大－最小値と 3 シグマ（偏差×3）を表示します。

ctsu2/ctsu2.h
//統計データを取得する数
#define CTSU_SENS_INDEX_MAX　(100)　//5 秒間（データサンプリング周期 20Hz×5 秒=100 個）

測定時間（5 秒）は、上記定数で変更できますが、マイコンの搭載 RAM容量により上限が決まります。
（16ch×100（測定値個数）×2bytes（測定値は 2 バイト=3200bytes）
※CTSU_SENS_INDEX_MAX は、相互容量の統計でも同じ定数が使われています
（相互容量の場合は測定チャネルは 50ch になりますので、RAM 容量の計算時考慮してください）

〇i コマンド（再初期化）

```
re-initialize>
Please do NOT touch key pad!

offset optimize & no-touch data scanning...
SO = 20
 （中略）
SO = 540
  D(TS26)  SO setting value = 543
initial sens value measure...
initialize finished.
```

「オフセット値最適化」「初期（非タッチ時の）測定値取得」「閾値を初期値に設定」を行います。起動後の初期化と同じ処理です。

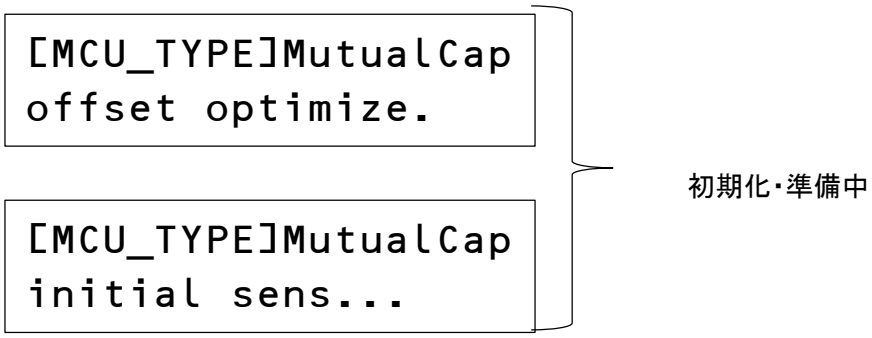
○h コマンド（ヘルプ表示）

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

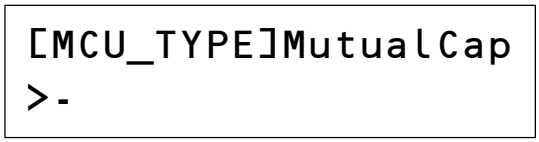
コマンドリストを表示します。

## 6.2. 相互容量キーパッド(D55A)

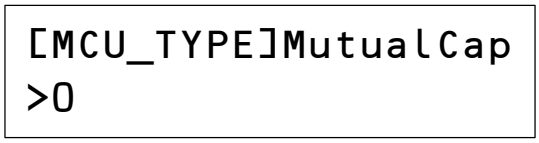
プログラムを起動すると、LCD 画面に




上記表示出力後、2 行目がーの表示となります。




この時点で、初期化は終わっていますので、キーパッドに触れてみてください。




触れたキーパッドの番号(0~24)が表示されます。

LCD には、タッチしているキーの情報のみ表示されます。

シリアル端末（11,5200bps）は、以下の表示が出力されます。

```
Copyright (C) 2021 HokutoDenshi. All Rights Reserved.

HSBRL78G23-100 CTSU(TouchKey) Mutual Cap Sample program.

Please do NOT touch key pad!

offset optimize & no-touch data scanning...
SO = 20
 （中略）
SO = 200
  key = 22  SO setting value = 202
SO = 220
SO = 240
  key = 17  SO setting value = 258
SO = 260
  key = 23  SO setting value = 266
 （中略）
SO = 400
  key = 5  SO setting value = 403
offset optimize finished.

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
  h : help
>
```

マイコンボード種
（キットにより変わります）

表示される数値はマイコンボード毎に異なります
（オフセット最適化の過程）

テストメニュー

キーボードからコマンドの入力が可能です。

〇m コマンド（測定値のモニタ）

現在の測定値（デフォルトでは、2 秒毎に表示を更新）を表示します。

```
--
ch/key        : init(1) init(2)  init_diff : sens(1) sens(2)  sens_diff : rate
ch= 0, key=20 :    5759   19835   ( 14076) :    5953   19760   ( 13807) : 0.981
ch= 1, key=21 :    5817   20649   ( 14832) :    5837   20755   ( 14918) : 1.006
ch= 2, key=24 :    5507   21869   ( 16362) :    5581   21808   ( 16227) : 0.992
ch= 3, key=23 :    5577   20790   ( 15213) :    5605   20767   ( 15162) : 0.997
ch= 4, key=22 :    5697   20520   ( 14823) :    5637   20630   ( 14993) : 1.011
ch= 5, key=15 :    5796   20444   ( 14648) :    5898   20282   ( 14384) : 0.982
ch= 6, key=16 :    5721   22142   ( 16421) :    5718   22268   ( 16550) : 1.008
ch= 7, key=19 :    5450   23766   ( 18316) :    5504   23846   ( 18342) : 1.001
ch= 8, key=18 :    5617   22489   ( 16872) :    5673   22588   ( 16915) : 1.003
ch= 9, key=17 :    5749   22323   ( 16574) :    5848   22423   ( 16575) : 1.000
ch=10, key=10 :    5794   20653   ( 14859) :    5981   20710   ( 14729) : 0.991
ch=11, key=11 :    5482   22392   ( 16910) :    5576   22387   ( 16811) : 0.994
ch=12, key=14 :    5384   24504   ( 19120) :    5411   24539   ( 19128) : 1.000
ch=13, key=13 :    5475   23198   ( 17723) :    5482   23330   ( 17848) : 1.007
ch=14, key=12 :    5498   22480   ( 16982) :    5661   22452   ( 16791) : 0.989
ch=15, key= 5 :    5636   20861   ( 15225) :    6029   20710   ( 14681) : 0.964
ch=16, key= 6 :    5454   22519   ( 17065) :    5558   22644   ( 17086) : 1.001
ch=17, key= 9 :    4239   32334   ( 28095) :    4383   32356   ( 27973) : 0.996
ch=18, key= 8 :    5430   24084   ( 18654) :    5499   24131   ( 18632) : 0.999
ch=19, key= 7 :    5477   23109   ( 17632) :    5545   23303   ( 17758) : 1.007
ch=20, key= 0 :    5726   22145   ( 16419) :    7584   21211   ( 13627) : 0.830*
ch=21, key= 1 :    5377   23650   ( 18273) :    5885   23800   ( 17915) : 0.980
ch=22, key= 4 :    3556   36345   ( 32789) :    3920   36334   ( 32414) : 0.989
ch=23, key= 3 :    4377   32322   ( 27945) :    4755   32120   ( 27365) : 0.979
ch=24, key= 2 :    5363   24344   ( 18981) :    5699   24232   ( 18533) : 0.976
```

・タッチしているキーの測定値（の演算結果）が減少します

→initial_diff に対して sens_diff が減少

・減少値が、一定以上の場合は、数値の右に(*)が付きます（タッチしているキーである印）

・キーボードから何かキーを押すと表示は終了します

○t コマンド（閾値のチューニング）

※（自己容量に比べ）多少長い時間
タッチしたままとする必要があります

```
threshold value optimize...

Please touch key pad ->0
& PC keyboard enter

touch data scaning... ch=20

 key scan end.

  touch(ave) = 7684,21215 no-touch(ave) = 5726,22145  touch(diff) = 13531  no-touch(diff) = 16419


Please touch key pad ->1
& PC keyboard enter

 …（中略）

Please touch key pad ->24
& PC keyboard enter

touch data scaning... ch=2

 key scan end.

  touch(ave) = 8083,21016 no-touch(ave) = 5507,21869  touch(diff) = 12933  no-touch(diff) = 16362


 tuning end.
```

キーパッド 0 にタッチして
キーボードのリターンキーを押してください

タッチした時の逆相と同相の差分=13531
非タッチ時の差分=16419 です

次はキーパッド 1 をタッチして
同様にリターンを押す

最後にキーパッド 24 をタッチして
同様にリターンを押す

全てのキーの閾値の最適化終了

タッチ判定の閾値を最適化するコマンドです。

※キーにタッチし、キーボードのリターンキーを押した後次の「Please touch key pad」の表示が出るまではタッチを続けてください

※ESC キーの入力か、CTRL-C の入力で処理を中断できます

タッチした時の測定値（256 回測定の平均）と非タッチ時の測定値（起動時に測定）の真ん中に閾値を設定します。

閾値の最適化前は、閾値はどのキーも同じ値となっていますが、このコマンドを使用すると実際にタッチした時の測定値を元に閾値を設定します。

○p コマンド（初期値と閾値の表示）

```
print initial reference value>

ch/key        :   SO : sens(1) sens(2) (  diff) : threshold
ch= 0, key=20 :  393 :    5759   19835 ( 14076) : 0.892
ch= 1, key=21 :  385 :    5817   20649 ( 14832) : 0.907
ch= 2, key=24 :  373 :    5507   21869 ( 16362) : 0.895
ch= 3, key=23 :  383 :    5577   20790 ( 15213) : 0.905
ch= 4, key=22 :  387 :    5697   20520 ( 14823) : 0.902
ch= 5, key=15 :  403 :    5796   20444 ( 14648) : 0.905
ch= 6, key=16 :  387 :    5721   22142 ( 16421) : 0.918
ch= 7, key=19 :  370 :    5450   23766 ( 18316) : 0.913
ch= 8, key=18 :  382 :    5617   22489 ( 16872) : 0.917
ch= 9, key=17 :  385 :    5749   22323 ( 16574) : 0.922
ch=10, key=10 :  391 :    5794   20653 ( 14859) : 0.904
ch=11, key=11 :  374 :    5482   22392 ( 16910) : 0.912
ch=12, key=14 :  353 :    5384   24504 ( 19120) : 0.910
ch=13, key=13 :  366 :    5475   23198 ( 17723) : 0.912
ch=14, key=12 :  373 :    5498   22480 ( 16982) : 0.914
ch=15, key= 5 :  386 :    5636   20861 ( 15225) : 0.906
ch=16, key= 6 :  369 :    5454   22519 ( 17065) : 0.912
ch=17, key= 9 :  258 :    4239   32334 ( 28095) : 0.944
ch=18, key= 8 :  353 :    5430   24084 ( 18654) : 0.923
ch=19, key= 7 :  363 :    5477   23109 ( 17632) : 0.921
ch=20, key= 0 :  380 :    5726   22145 ( 16419) : 0.912
ch=21, key= 1 :  367 :    5377   23650 ( 18273) : 0.921
ch=22, key= 4 :  202 :    3556   36345 ( 32789) : 0.954
ch=23, key= 3 :  266 :    4377   32322 ( 27945) : 0.948
ch=24, key= 2 :  360 :    5363   24344 ( 18981) : 0.927
```

・SO（オフセット値）
・Sens（非タッチ時の測定値）
・threshold（判定閾値）
の表示を行います。

上記表示例は、t コマンドで閾値の最適化を行った場合です。（閾値の最適化を行わない場合は、threshold 値は一律 0.950 です。）

## ○s コマンド（最大－最小、偏差の表示）

```
sens statistics(measure 5 sec)>

sens value statistics data

ch/key       :   same phase                    reverse phase                  diff
             :   Ave (   Max -   Min, 3sigma ) :   Ave (   Max -   Min, 3sigma ) :   Ave (   Max -   Min, 3sigma )
ch= 0, key=20 :  5770 (  5877 -  5621,    143 ) : 19582 ( 19780 - 19257,    269 ) : 13811 ( 14070 - 13442,    333 )
ch= 1, key=21 :  5802 (  5935 -  5654,    169 ) : 20429 ( 20659 - 20201,    266 ) : 14627 ( 14912 - 14368,    342 )
ch= 2, key=24 :  5438 (  5571 -  5312,    158 ) : 21706 ( 21882 - 21455,    270 ) : 16268 ( 16490 - 15912,    312 )
ch= 3, key=23 :  5521 (  5634 -  5366,    188 ) : 20587 ( 20764 - 20345,    248 ) : 15066 ( 15351 - 14751,    332 )
ch= 4, key=22 :  5638 (  5759 -  5488,    173 ) : 20326 ( 20538 - 20122,    230 ) : 14687 ( 14904 - 14457,    285 )
ch= 5, key=15 :  5792 (  5926 -  5617,    178 ) : 20239 ( 20416 - 19922,    262 ) : 14447 ( 14682 - 14144,    325 )
ch= 6, key=16 :  5670 (  5807 -  5503,    198 ) : 21969 ( 22311 - 21719,    301 ) : 16298 ( 16617 - 15976,    358 )
ch= 7, key=19 :  5419 (  5553 -  5273,    167 ) : 23584 ( 23821 - 23290,    348 ) : 18164 ( 18433 - 17849,    423 )
ch= 8, key=18 :  5558 (  5704 -  5443,    176 ) : 22338 ( 22558 - 22077,    284 ) : 16779 ( 17033 - 16410,    367 )
ch= 9, key=17 :  5699 (  5841 -  5561,    170 ) : 22160 ( 22328 - 21942,    266 ) : 16460 ( 16714 - 16196,    335 )
ch=10, key=10 :  5812 (  5934 -  5683,    161 ) : 20493 ( 20699 - 20303,    226 ) : 14681 ( 14900 - 14479,    283 )
ch=11, key=11 :  5468 (  5639 -  5284,    200 ) : 22253 ( 22470 - 22004,    290 ) : 16784 ( 17064 - 16469,    373 )
ch=12, key=14 :  5379 (  5508 -  5193,    168 ) : 24354 ( 24632 - 24033,    354 ) : 18975 ( 19304 - 18560,    416 )
ch=13, key=13 :  5466 (  5578 -  5329,    160 ) : 23065 ( 23274 - 22837,    297 ) : 17599 ( 17908 - 17320,    338 )
ch=14, key=12 :  5457 (  5595 -  5291,    186 ) : 22360 ( 22556 - 22133,    295 ) : 16902 ( 17129 - 16625,    348 )
ch=15, key= 5 :  5846 (  5987 -  5671,    195 ) : 20585 ( 20752 - 20402,    249 ) : 14738 ( 15017 - 14451,    343 )
ch=16, key= 6 :  5455 (  5612 -  5271,    193 ) : 22376 ( 22650 - 22156,    311 ) : 16921 ( 17245 - 16624,    376 )
ch=17, key= 9 :  4277 (  4380 -  4179,    127 ) : 32099 ( 32517 - 31529,    577 ) : 27822 ( 28238 - 27262,    574 )
ch=18, key= 8 :  5417 (  5562 -  5262,    173 ) : 23981 ( 24207 - 23682,    332 ) : 18564 ( 18875 - 18142,    415 )
ch=19, key= 7 :  5475 (  5653 -  5276,    204 ) : 22973 ( 23207 - 22611,    335 ) : 17498 ( 17799 - 16958,    422 )
ch=20, key= 0 :  7642 (  7815 -  7474,    207 ) : 21000 ( 21208 - 20655,    300 ) : 13358 ( 13688 - 12972,    405 )
ch=21, key= 1 :  5791 (  5895 -  5640,    175 ) : 23497 ( 23734 - 23222,    322 ) : 17705 ( 18094 - 17335,    417 )
ch=22, key= 4 :  3811 (  3919 -  3680,    124 ) : 36209 ( 36796 - 35487,    906 ) : 32398 ( 32957 - 31691,    937 )
ch=23, key= 3 :  4627 (  4717 -  4505,    137 ) : 32176 ( 32640 - 31642,    613 ) : 27549 ( 28022 - 26991,    620 )
ch=24, key= 2 :  5582 (  5732 -  5416,    165 ) : 24281 ( 24509 - 23872,    370 ) : 18698 ( 18943 - 18149,    413 )
```

この表示例はキーパッド 0 に触れている条件です

5 秒間測定を行い、測定期間内の

・同相測定時の最大－最小値と 3 シグマ（偏差×3）
・逆相測定時の最大－最小値と 3 シグマ（偏差×3）
・逆相－同相時の演算結果の最大－最小値と 3 シグマ（偏差×3）

を表示します。

ctsu2/ctsu2.h
//統計データを取得する数
#define CTSU_SENS_INDEX_MAX　(100)　//5 秒間（データサンプリング周期 20Hz×5 秒=100 個）

測定時間（5 秒）は、上記定数で変更できますが、マイコンの搭載 RAM容量により上限が決まります。
（50ch×100（測定値個数）×2bytes（測定値は 2 バイト=約 10kbytes）

〇i コマンド（再初期化）

```
re-initialize>

Please do NOT touch key pad!

offset optimize & no-touch data scanning...
SO = 20 （中略）
SO = 400
  key = 5  SO setting value = 401
offset optimize finished.
initial sens value measure...
initialize finished.
```

「オフセット値最適化」「初期（非タッチ時の）測定値取得」「閾値を初期値に設定」を行います。起動後の初期化と同じ処理です。

〇h コマンド（ヘルプ表示）

コマンドリストの表示。

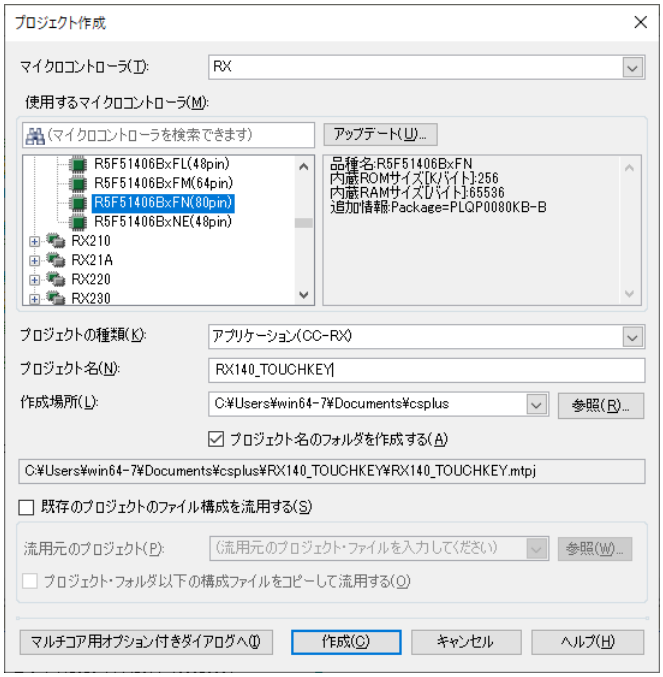
# 7. 開発環境の構築

ここでは、CS+の環境に本書で説明するソースコードを動かせる環境を構築する方法を示します。なお、CD 内には、既に作成済みの CS+プロジェクトフォルダが格納されていますので、本章はプロジェクトを最初から作成したいという方向けの説明となります。

## 7.1. RX

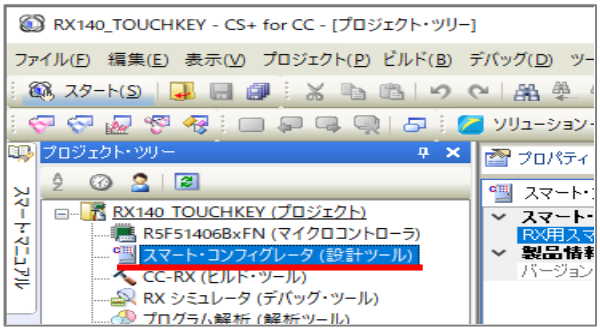
CS+ for CC と、RX スマート・コンフィグレータを使用しますので、これらのツールを使用する環境が整っていない場合は、予めインストールして頂きたく。

・CS+でプロジェクトを作成

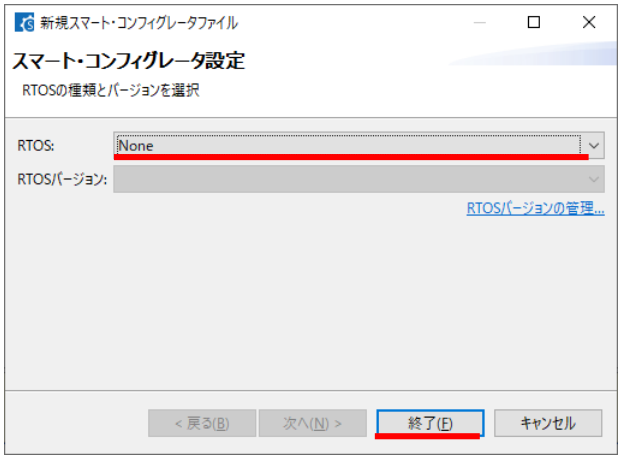



マイクロコントローラとして、R5F51406BxFN を選択してください。プロジェクト名は任意の名称を入力してください。

・スマート・コンフィグレータを起動

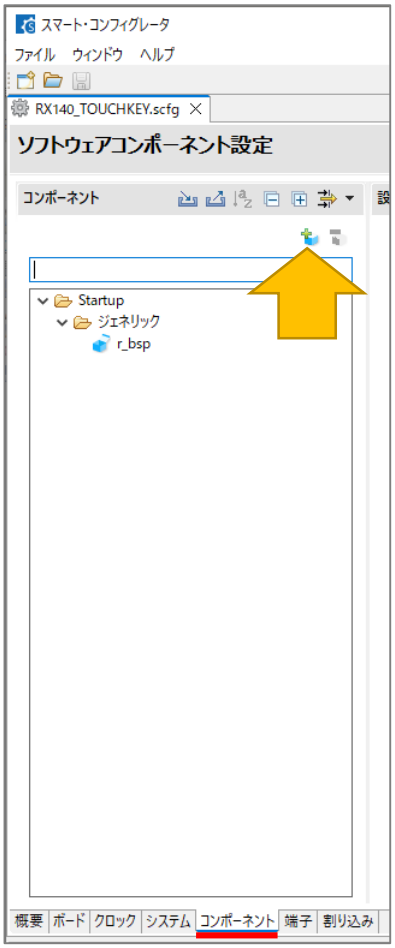



スマート・コンフィグレータ（設計ツール）をダブルクリックで起動

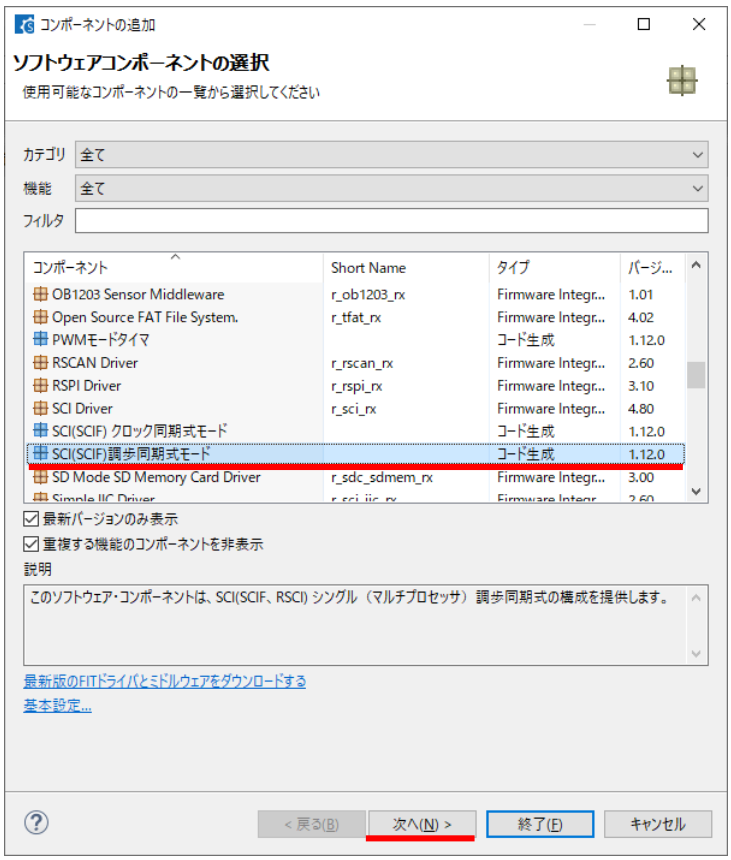



RTOS: None を選択
「終了」

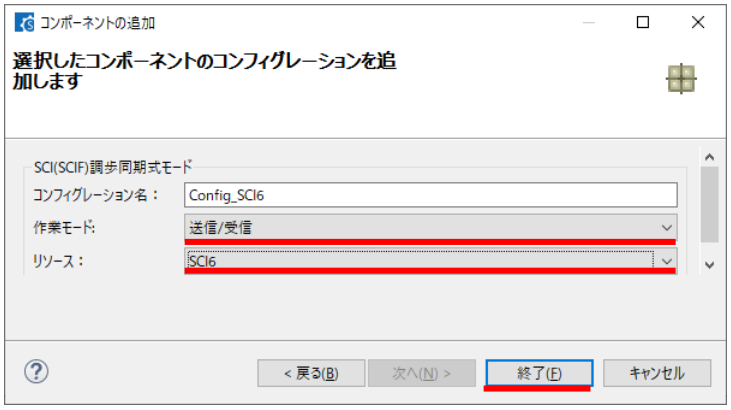
・スマートコンフィグレータで SCI のコンポーネントを追加する



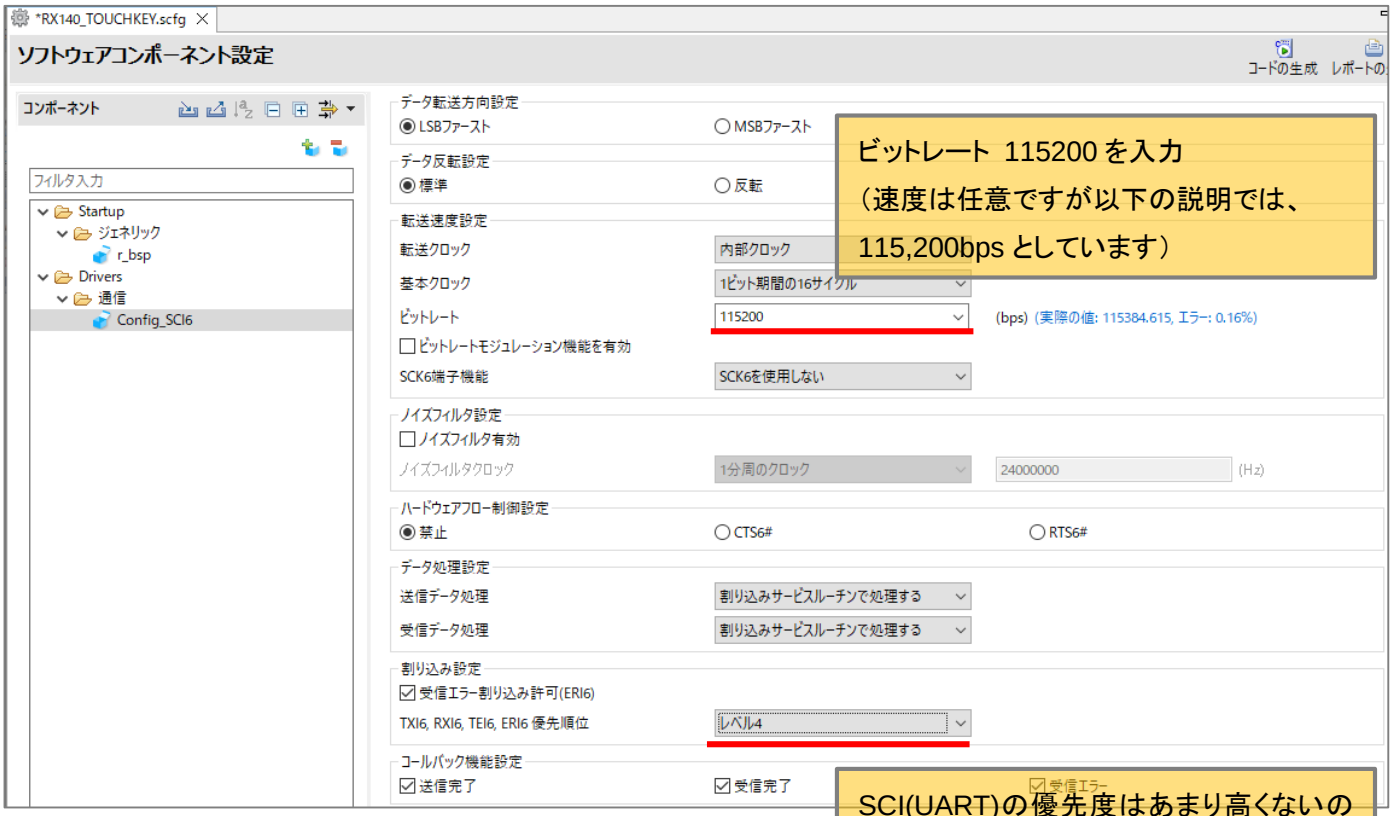

「コンポーネント」タブを選択
コンポーネント追加アイコンを押す




SCI(SCIF)調歩同期式モードを選択
「次へ」




送信/受信
SCI6 を選択
「終了」

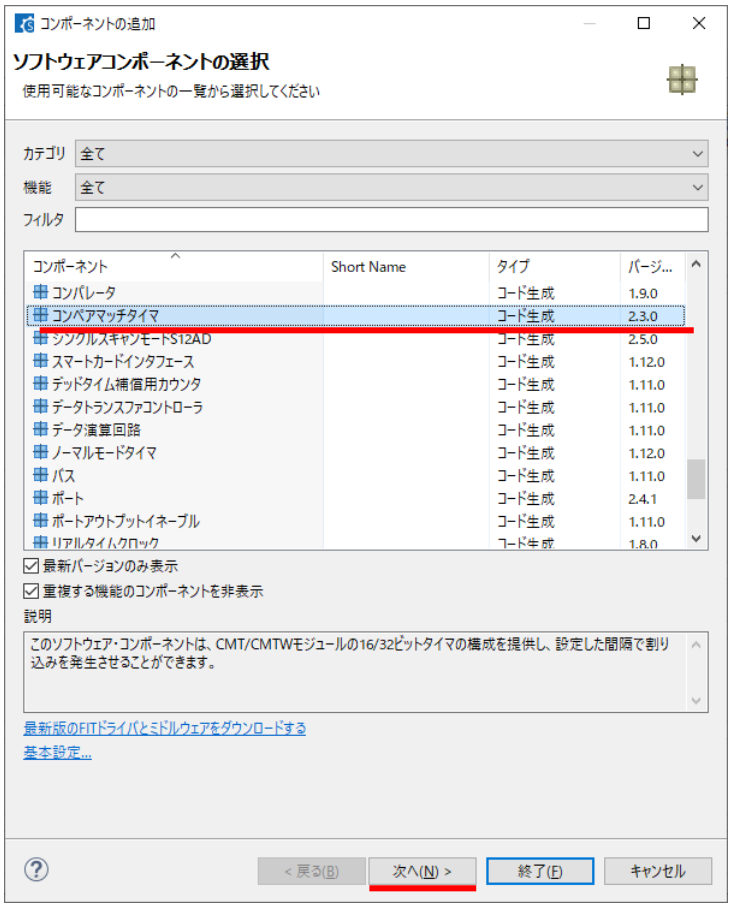



ビットレート 115200 を入力
（速度は任意ですが以下の説明では、
115,200bps としています）

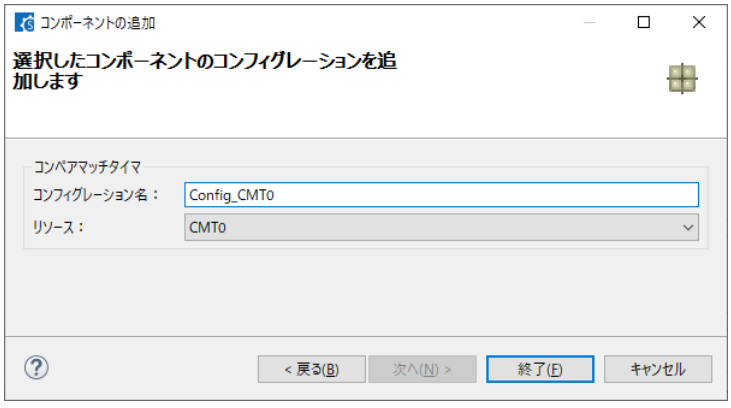
SCI(UART)の優先度はあまり高くないの
でここでは、レベル 4 に設定します
（設定は任意です）

・コンペアマッチタイマの追加

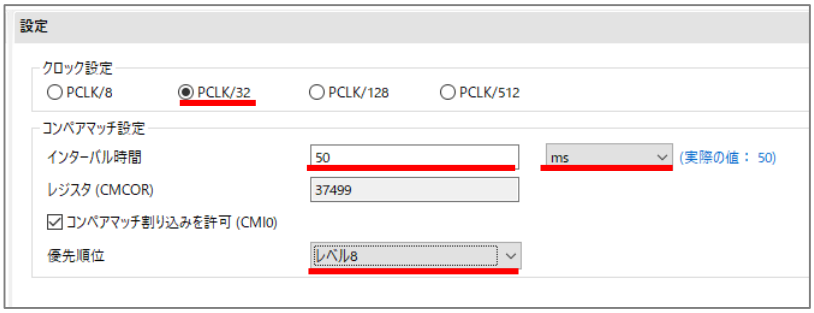
同様にコンペアマッチタイマ(CMT0)を追加します。




コンペアマッチタイマ
「次へ」

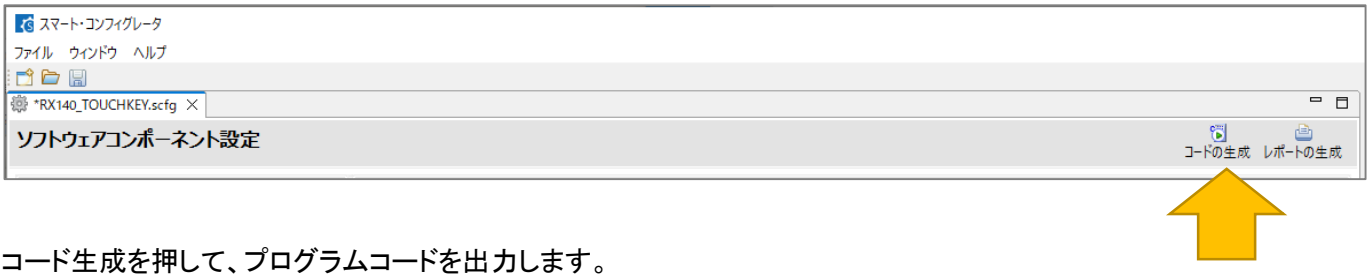



リソース CMT0
「終了」




時間は 50ms
割り込み優先度は 8 に設定します。
（時間も割り込み優先度も設定値は任意）

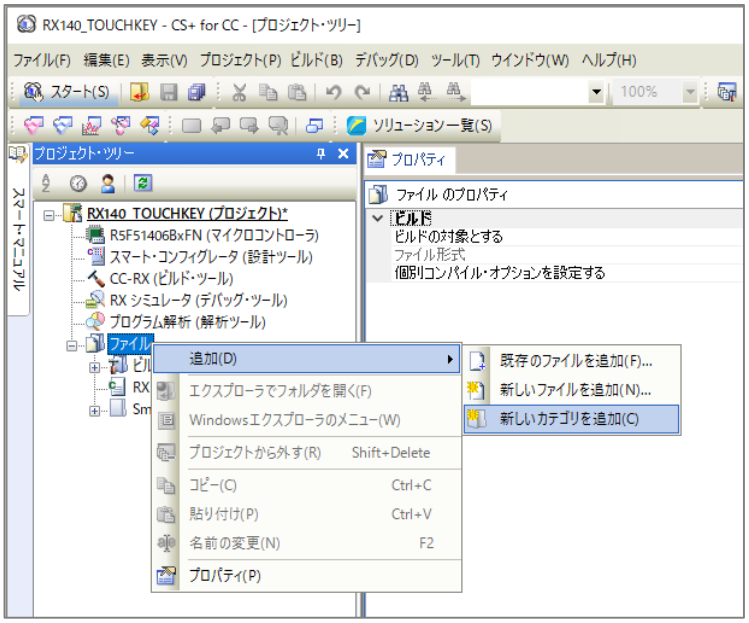
ここで設定した時間が測定周期（50ms であれば、1 秒間に 20 回）となります。




コード生成を押して、プログラムコードを出力します。

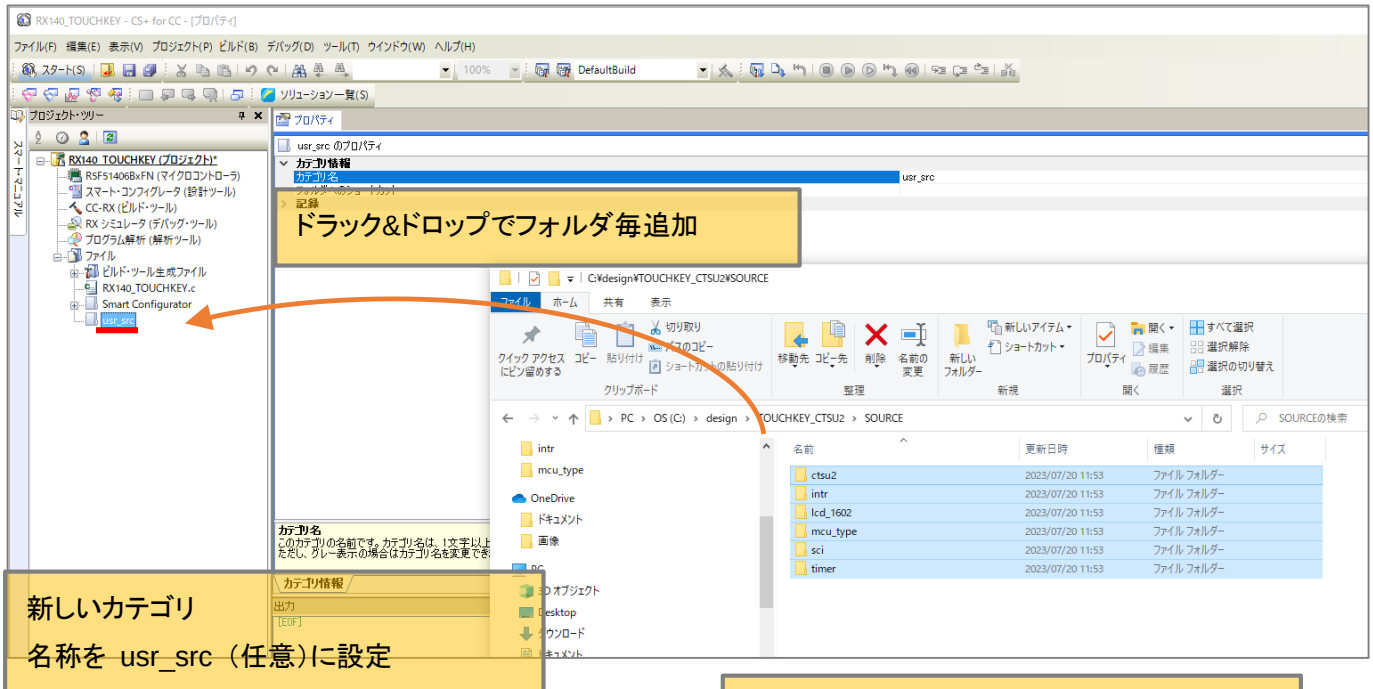

上記メッセージが表示された場合は「はい」を押してください。

・ソースファイルをプロジェクトに追加



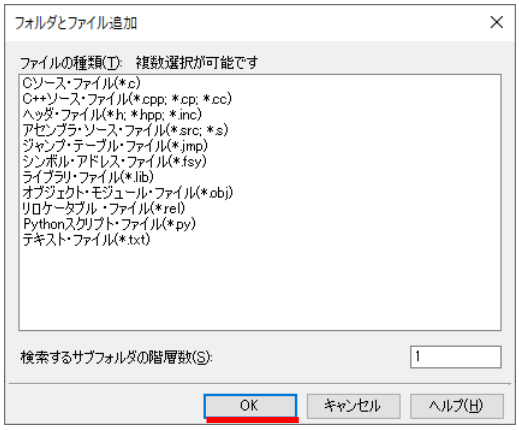

ファイル（右クリック）ー追加ー新しいカテゴリを追加

CS+のプロジェクトツリーで、新しいカテゴリを追加する。（ここでは、usr_src という名称で追加する（名称は任意））

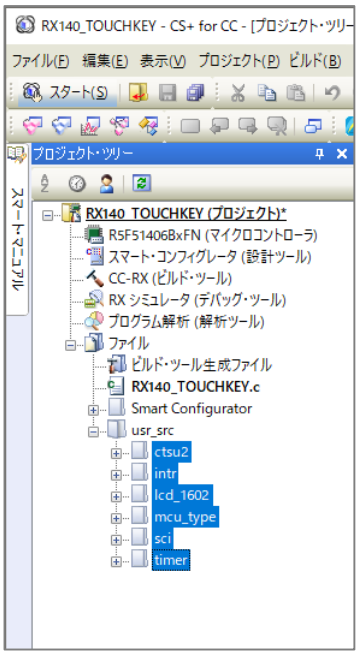



CD の SOURCE を PC の適当なフォルダにコピー

※CD のフォルダを直接 usr_src 以下にドラッグ&ドロップすると、CD を抜いたときソースファイルが見えなくなりますので、ソースファイルは PC のストレージ上にコピーしてください

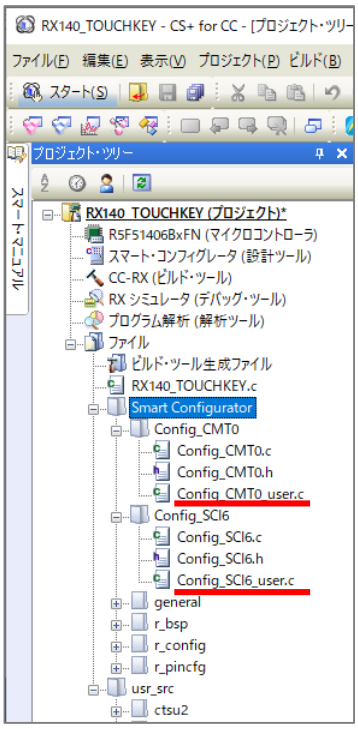



OK を押す




プロジェクトツリーにソース一式がぶら下がる形となれば OK です。

（上記の手順以外でも、CD の SOURCE フォルダ以下が、プロジェクトに追加できれば、どのような手法でも構いません）




Config_CMT0_user.c
Config_SCI6_user.c
をダブルクリックして、ファイルを開いて書き換えます

次に、CMT と SCI のコード生成されたファイルに、プログラムコードを追加します。

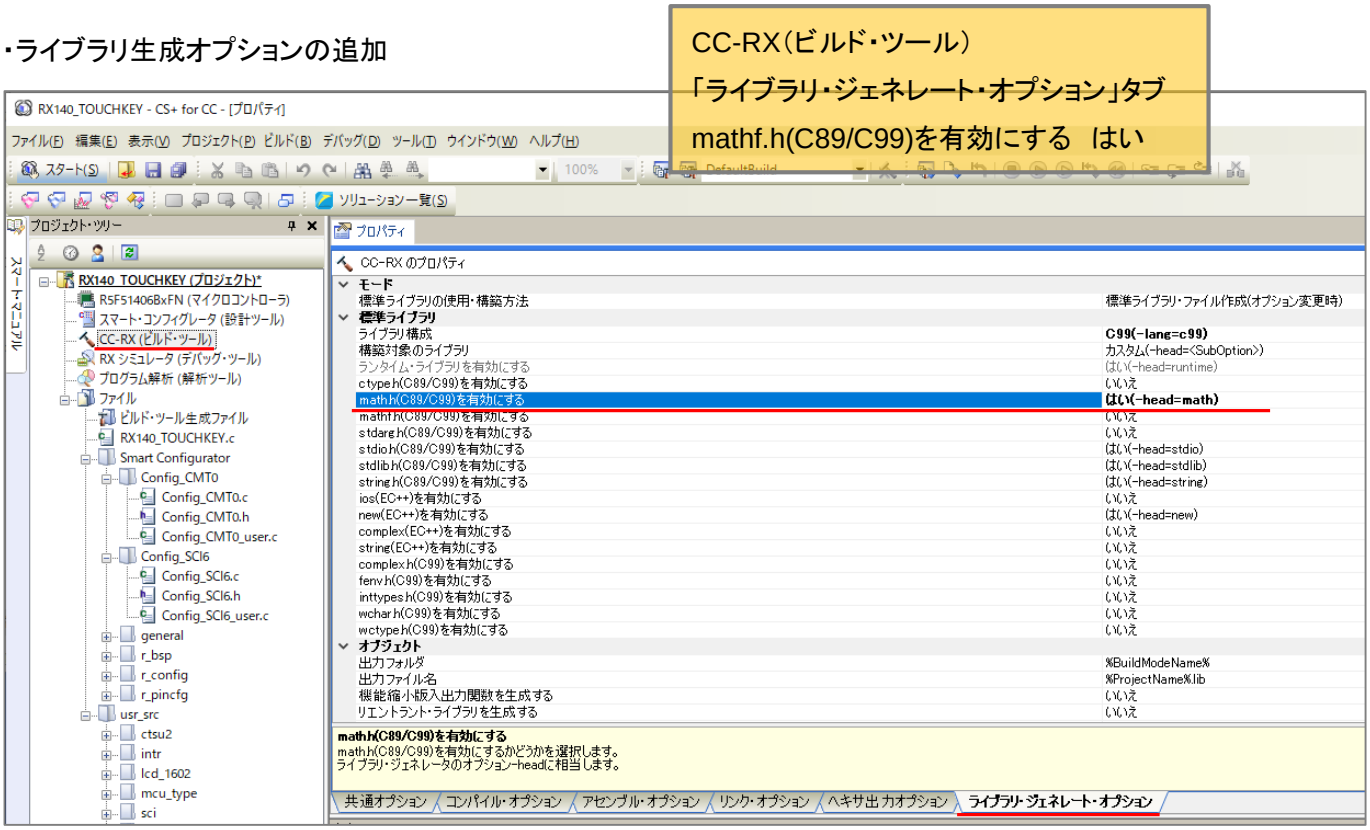
・Config_CMT0_user.c の追加（赤字の部分を追加）

```
/***********************************************************************************************************
**************
Includes
***********************************************************************************************************
**************/
#include "r_cg_macrodriver.h"
#include "Config_CMT0.h"
/* Start user code for include. Do not edit comment generated here */
#include "timer.h"
/* End user code. Do not edit comment generated here */
#include "r_cg_userdefine.h"

（中略）
#if FAST_INTERRUPT_VECTOR == VECT_CMT0_CMI0
#pragma interrupt r_Config_CMT0_cmi0_interrupt(vect=VECT(CMT0,CMI0),fint)
#else
#pragma interrupt r_Config_CMT0_cmi0_interrupt(vect=VECT(CMT0,CMI0))
#endif
static void r_Config_CMT0_cmi0_interrupt(void)
{
    /* Start user code for r_Config_CMT0_cmi0_interrupt. Do not edit comment generated here */
    periodic_process();
    /* End user code. Do not edit comment generated here */
}
```

・Config_SCI6_user.c の追加（赤字の部分を追加）

```
/***********************************************************************************************************
***************
Includes
***********************************************************************************************************
*************/
#include "r_cg_macrodriver.h"
#include "Config_SCI6.h"
/* Start user code for include. Do not edit comment generated here */
#include "sci.h"
/* End user code. Do not edit comment generated here */
#include "r_cg_userdefine.h"
 （中略）

static void r_Config_SCI6_callback_transmitend(void)
{
    /* Start user code for r_Config_SCI6_callback_transmitend. Do not edit comment generated here */
    intr_sci_send_end();
    /* End user code. Do not edit comment generated here */
}
 （中略）

static void r_Config_SCI6_callback_receiveend(void)
{
    /* Start user code for r_Config_SCI6_callback_receiveend. Do not edit comment generated here */
    intr_sci_receive_end();
    /* End user code. Do not edit comment generated here */
}
 （中略）

static void r_Config_SCI6_callback_receiveerror(void)
{
    /* Start user code for r_Config_SCI6_callback_receiveerror. Do not edit comment generated here */
    intr_sci_receive_error();
    /* End user code. Do not edit comment generated here */
}
```

・ライブラリ生成オプションの追加

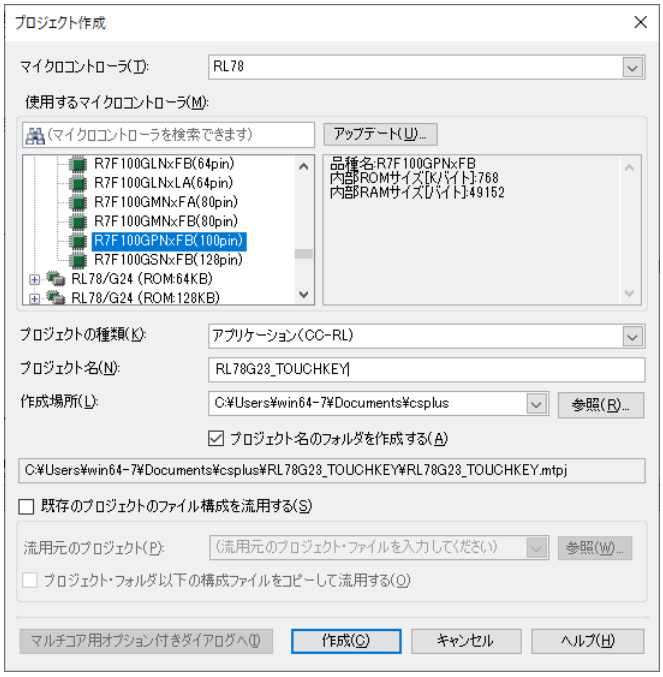



RX に固有の設定は以上です。続きは、8 章を参照してください。

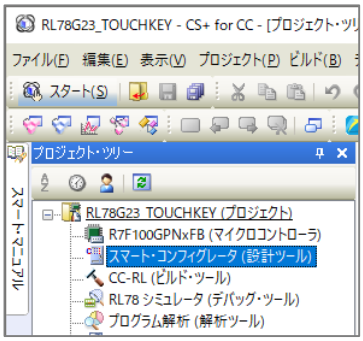
## 7.2. RL78

CS+ for CC と、RL78 スマートコンフィグレータを使用しますので、これらのツールを使用する環境が整っていない場合は、予めインストールして頂きたく。

・CS+でプロジェクトを作成

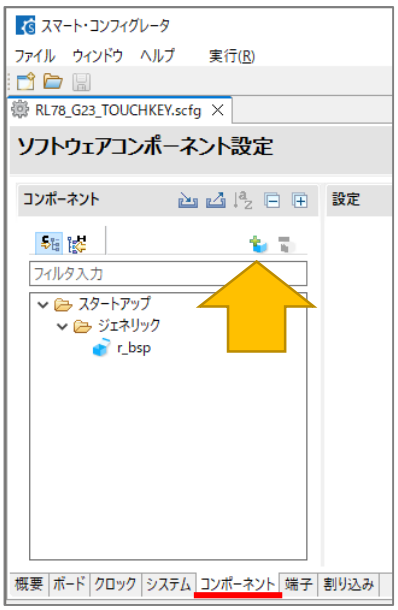



マイクロコントローラとして、R7F100PGNxFB を選択してください。プロジェクト名は任意の名称を入力してください。

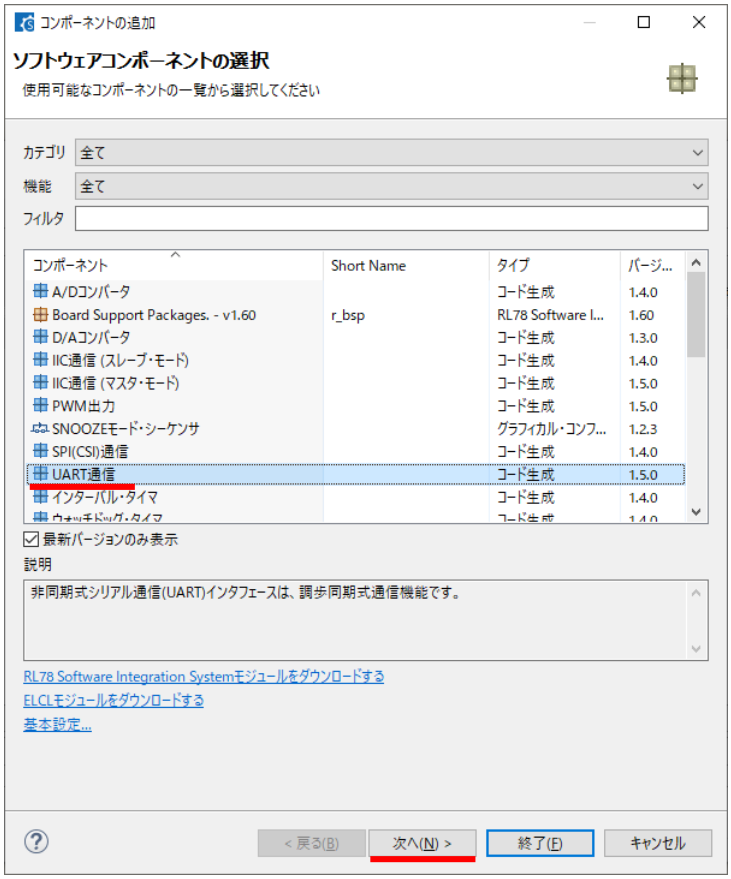



スマート・コンフィグレータ（設計ツール）をダブルクリックで起動

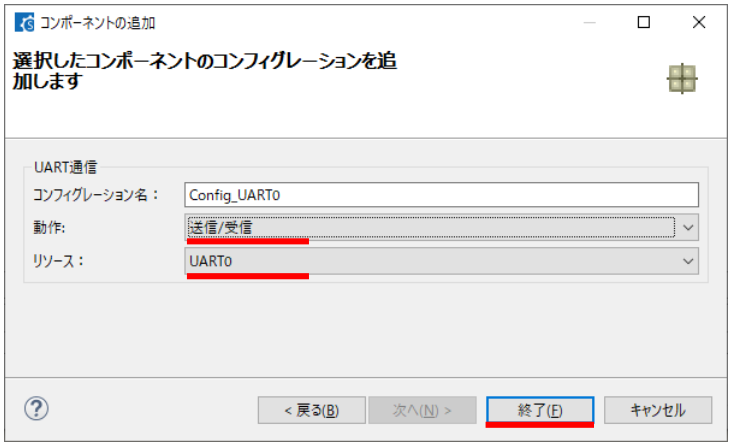
・スマートコンフィグレータで UART(SCI)のコンポーネントを追加する




「コンポーネント」タブを選択
コンポーネント追加アイコンを押す




UART 通信を選択
次へ




送信/受信を選択
終了

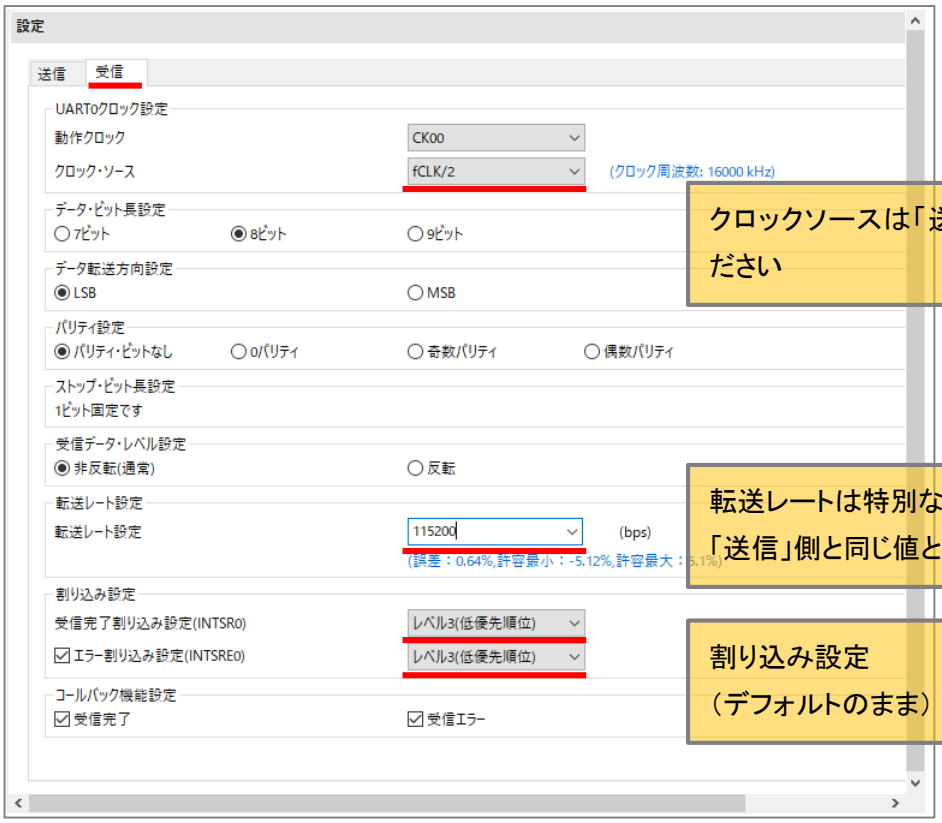
※リソースですが、
・E2, E2Lite のエミュレータを持っている、デバッガは使用しない場合： UART0 を選択
→ボード搭載の USB-Serial インタフェースで PC と通信を行います
・COM ポートデバッグを使用する場合： UARTA1 を選択
→ボード搭載の USB-Serial インタフェースはデバッガとして使用されるので、別な UART チャンネルを選択してください（USB-Serial 変換機器を別途拡張 I/O 端子に接続する必要があります）UARTA1 を選択した場合、LCD 接続基板上のコネクタ経由で PC と接続可能です


fCLK=32MHz のとき、115200bps 設定を行う場合は、クロックソースを fCLK/2 を選択してください

転送レート 115200 を入力
（速度は任意ですが以下の説明では、115,200bps としています）

割り込み設定
（デフォルトのまま）

※RL78 は、「送信」と「受信」で速度設定が別々となりますので、「受信」タブでも同様の速度設定としてください



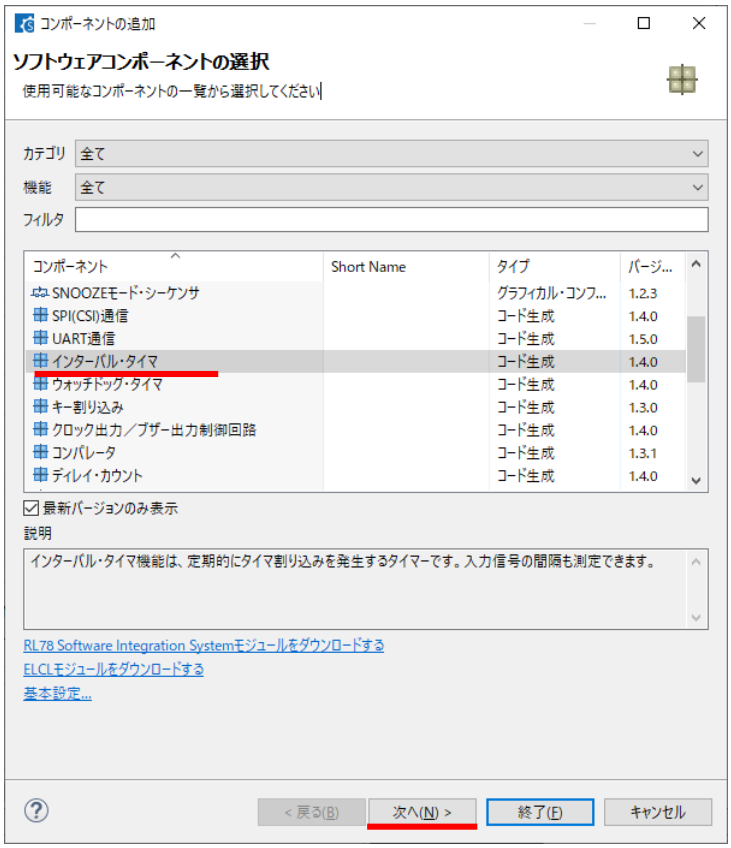

クロックソースは「送信」側と同一にしてください

転送レートは特別な理由が無い限り「送信」側と同じ値としてください

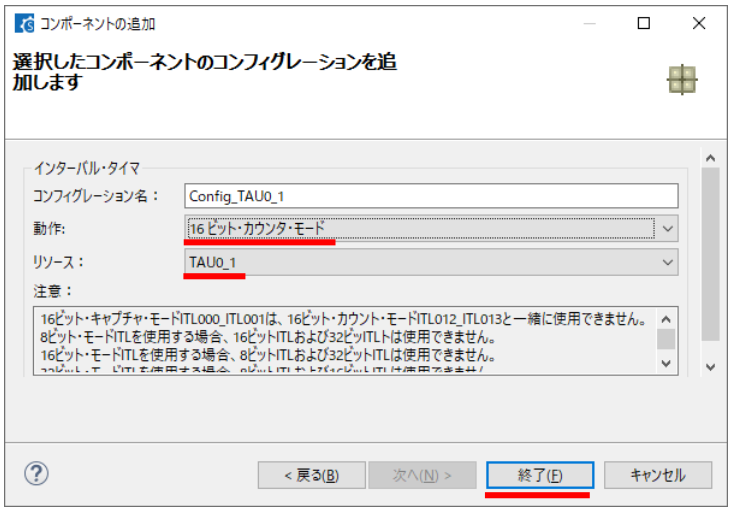
割り込み設定
（デフォルトのまま）

「受信」タブに切り替えて、受信側

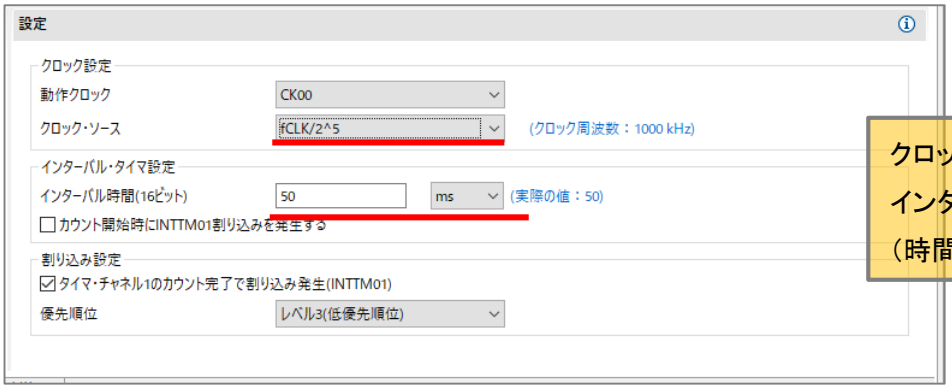
同様に、タイマも追加します。




インターバル・タイマを選択
次へ




16 ビット・カウンタ・モードを選択
TAU0_1 を選択
終了



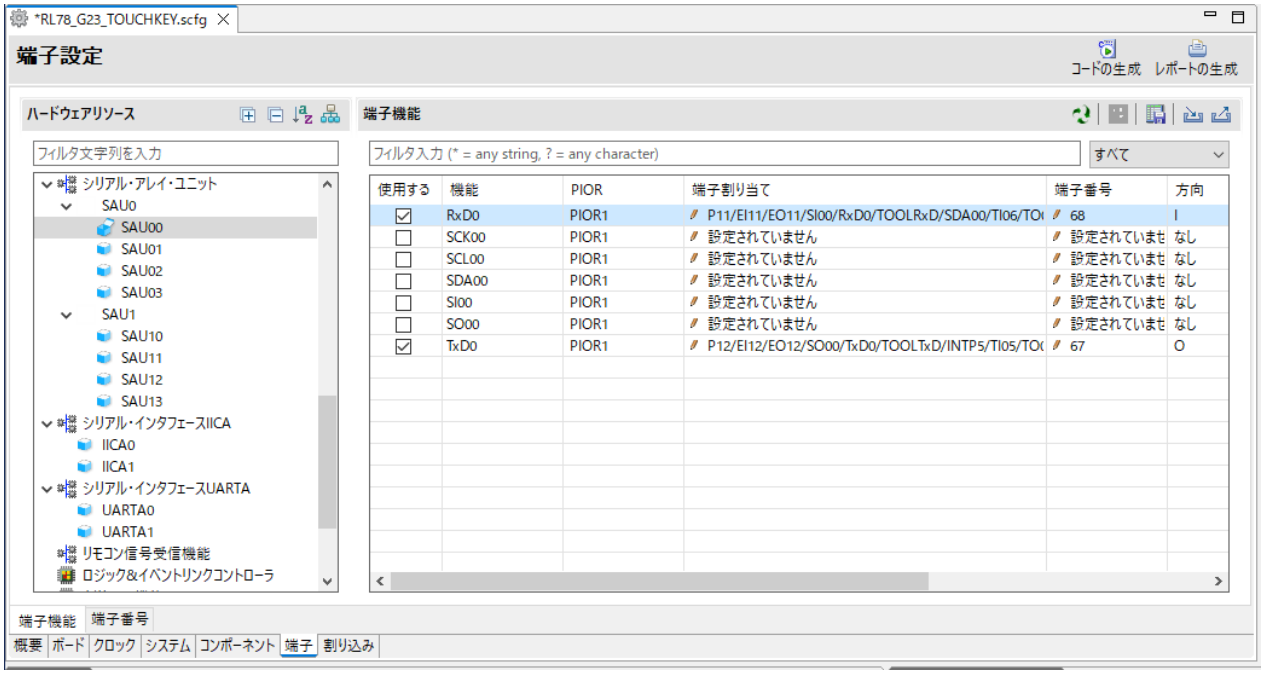

クロックソースを fCLK/2^5
インターバル時間を 50ms に設定
（時間は任意です）

ここで設定した時間が測定周期（50ms であれば、1 秒間に 20 回）となります。

・UART の端子設定

「端子」タブ

SAU00

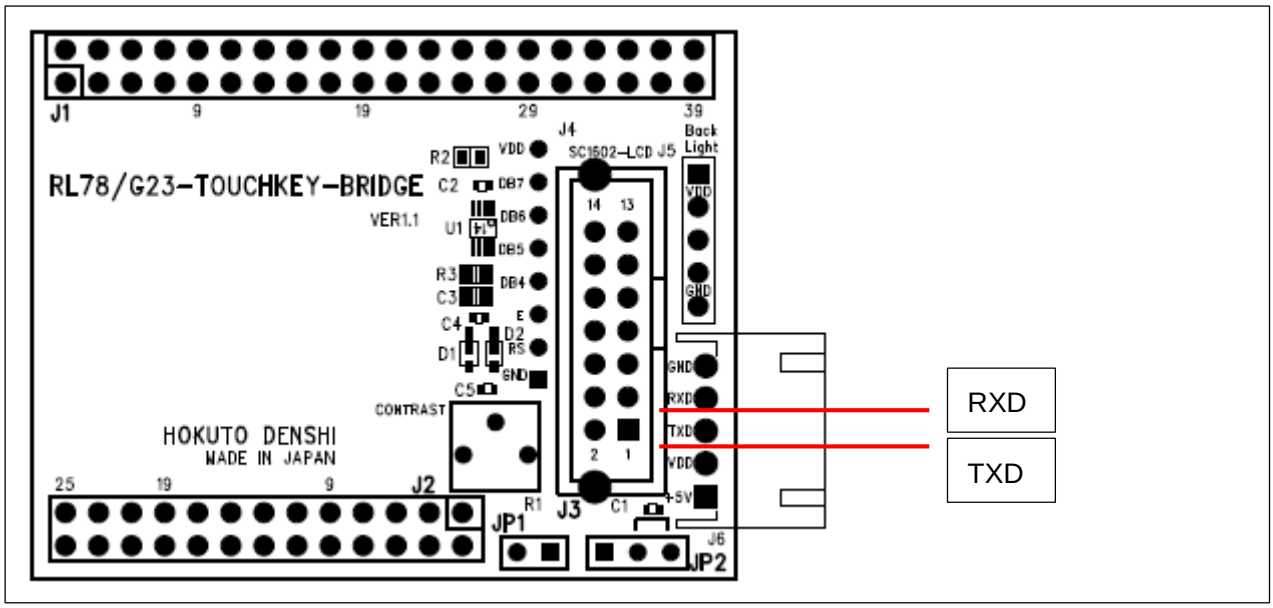



E2, E2Lite を使用してデバッグを行う場合、及びデバッガを使用しない場合は、RXD0(P11), TXD0(P12)を選択するようにしてください。

P11, P12 がマイコンボード上の USB-Serial 変換 IC に接続されており、ボード上の USB-miniB コネクタ経由で PC と通信が可能です。

－COM ポートを使用してデバッグを行う場合－

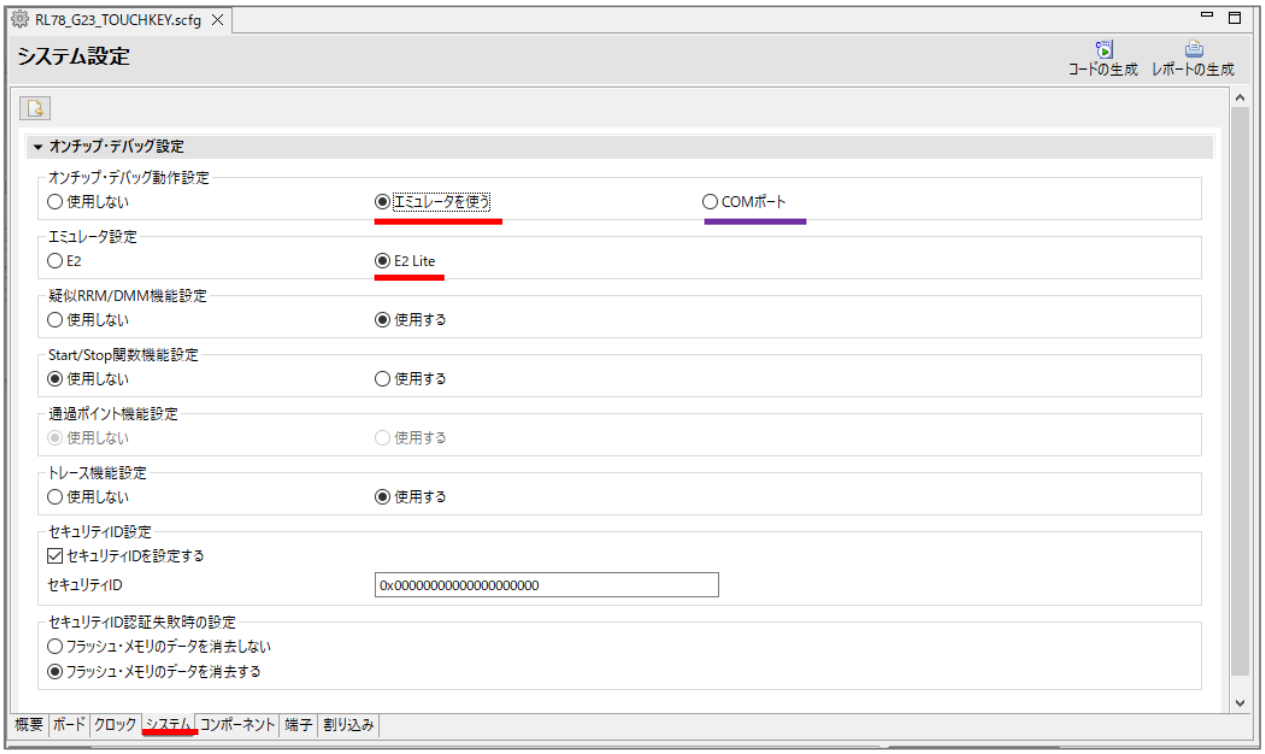
P11(TOOLRxD), P12(TOOLTxD)はデバッガに占有されますので、PC との通信に UARTA1(P41, P42)を使用する様に設定してください。




P41(RxDA1), P42(TxDA1)の信号は、LCD 接続基板上に引き出されていますので、この部分に市販の USB-Serial 変換機器（当社製品では、USB-1S(JST)等）を接続してください。

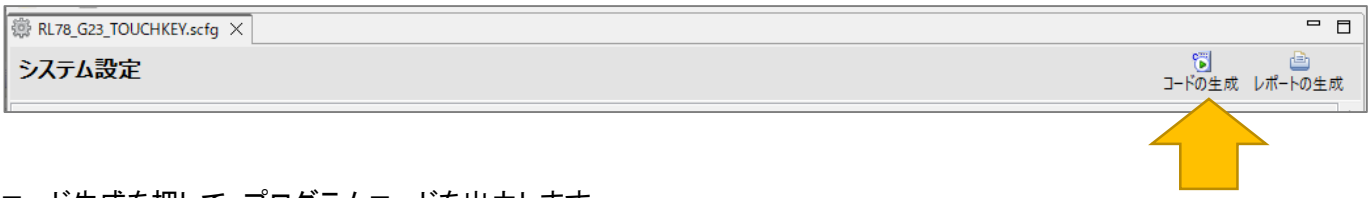
※UART1A ではなく、他の UART-ch を使用する事でも問題ありません

・オンチップ・デバッグ設定

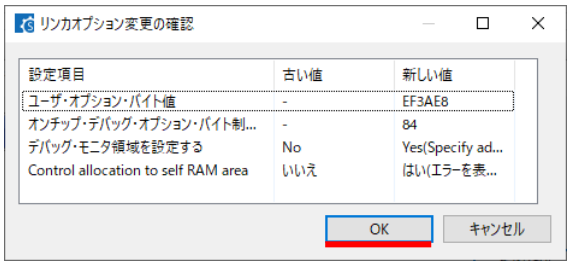



システムタブに、デバッガの設定がありますので、使用するデバッガに応じて選択してください。「COM ポート」を選択した場合、ボード上の USB-miniB 端子経由でデバッグが行えます。（JP6:1-2 ショート, JP5:ショート）

一通り設定が終われば、

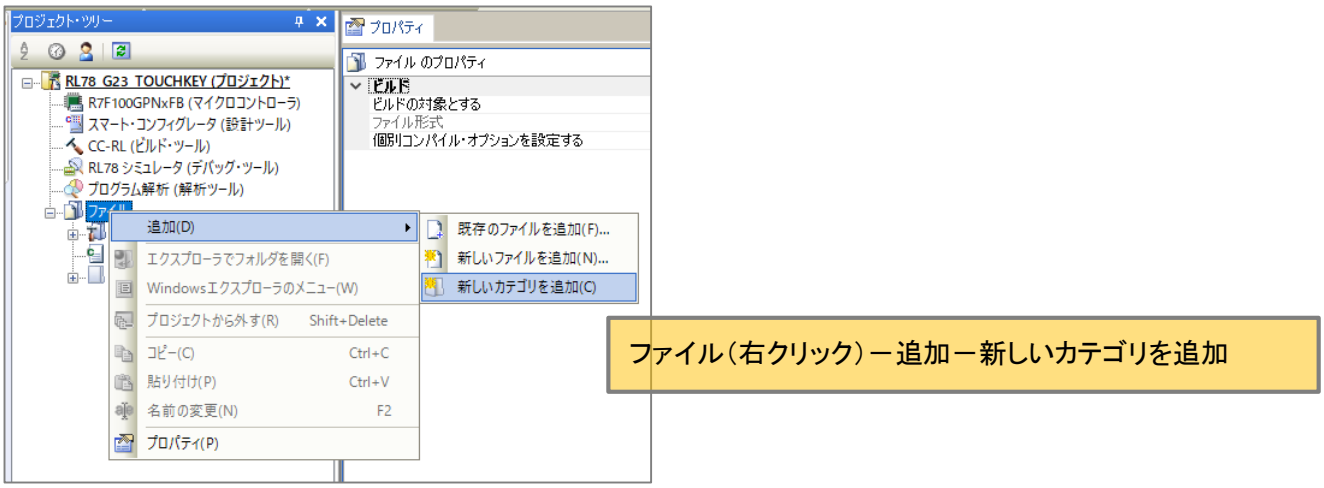



コード生成を押して、プログラムコードを出力します。

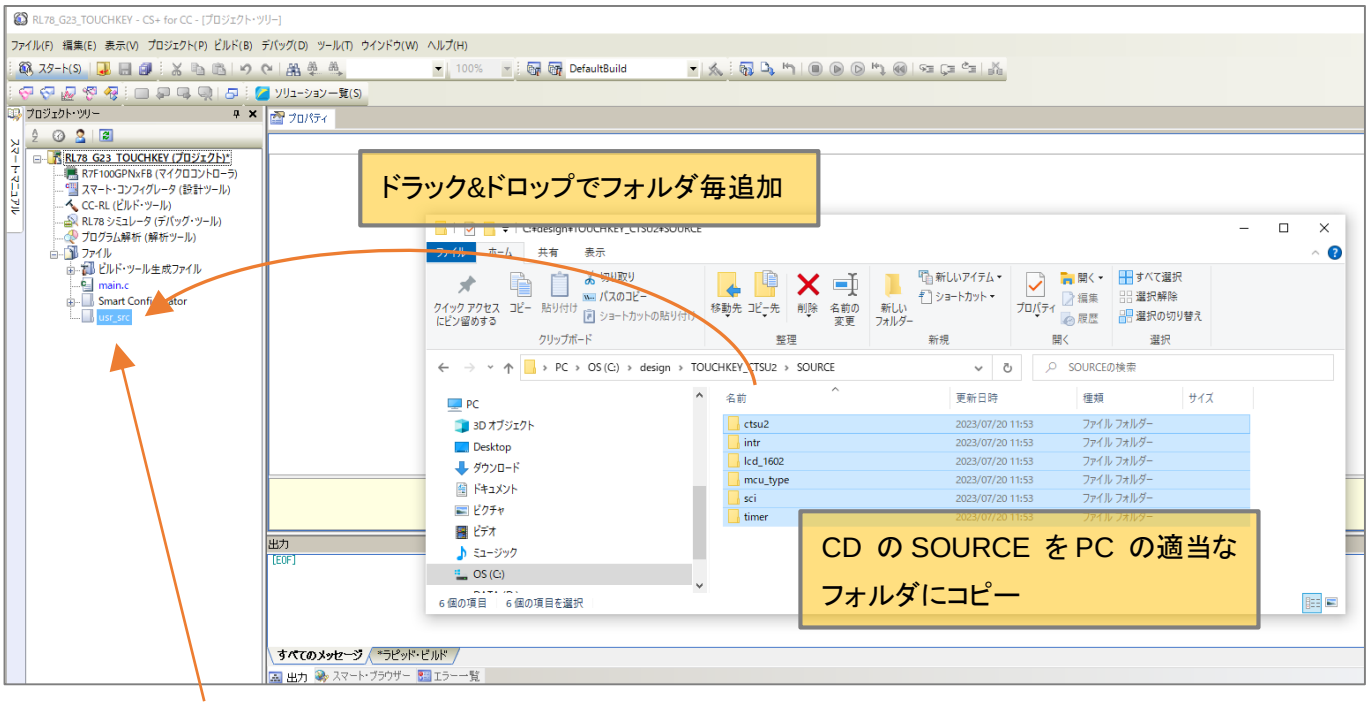



上記画面が出た場合は、OK。

・ソースファイルをプロジェクトに追加

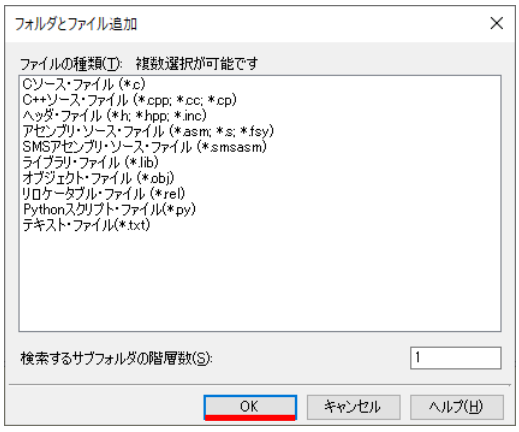



CS+のプロジェクトツリーで、新しいカテゴリを追加する。（ここでは、usr_src という名称で追加する（名称は任意））




新しいカテゴリ
名称を usr_src（任意）に設定

※CD のフォルダを直接 usr_src 以下にドラッグ&ドロップすると、CD を抜いたときソースファイルが見えなくなりますので、ソースファイルは PC のストレージ上にコピーしてください

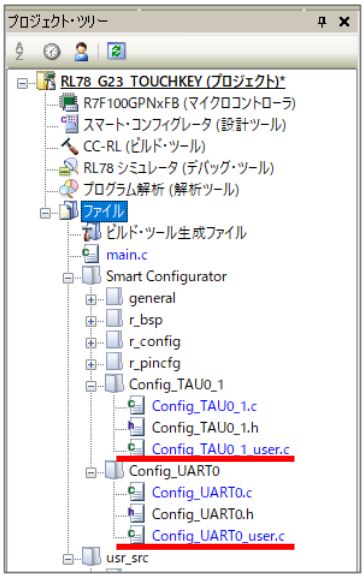



OK を押す


プロジェクト・ツリーにソース一式がぶら下がる形となればOKです。

（上記の手順以外でも、CD の SOURCE フォルダ以下が、プロジェクトに追加できれば、どのような手法でも構いません）

次に、TAU と UART のコード生成されたファイルに、プログラムコードを追加します。



・Config_TAU0_1_user.c の追加（赤字の部分を追加）

（TAU0_1 以外のタイマを使用する場合は、使用するタイマのソースコードに追加してください）

```
/***********************************************************************************************************
***************
Includes
***********************************************************************************************************
**************/
#include "r_cg_macrodriver.h"
#include "r_cg_userdefine.h"
#include "Config_TAU0_1.h"
/* Start user code for include. Do not edit comment generated here */
#include "timer.h"
/* End user code. Do not edit comment generated here */

（中略）
/***********************************************************************************************************
***************
* Function Name: r_Config_TAU0_1_interrupt
* Description  : This function is INTTM01 interrupt service routine.
* Arguments    : None
* Return Value : None
***********************************************************************************************************
**************/
static void __near r_Config_TAU0_1_interrupt(void)
{
    /* Start user code for r_Config_TAU0_1_interrupt. Do not edit comment generated here */
    periodic_process();
    /* End user code. Do not edit comment generated here */
}
```

・Config_UART0_user.c の追加（赤字の部分を追加）

```
/***********************************************************************************************************
***************
Includes
***********************************************************************************************************
**************/
#include "r_cg_macrodriver.h"
#include "r_cg_userdefine.h"
#include "Config_UART0.h"
/* Start user code for include. Do not edit comment generated here */
#include "sci.h"
/* End user code. Do not edit comment generated here */
```

・Config_UART0_user.c の追加（赤字の部分を追加）

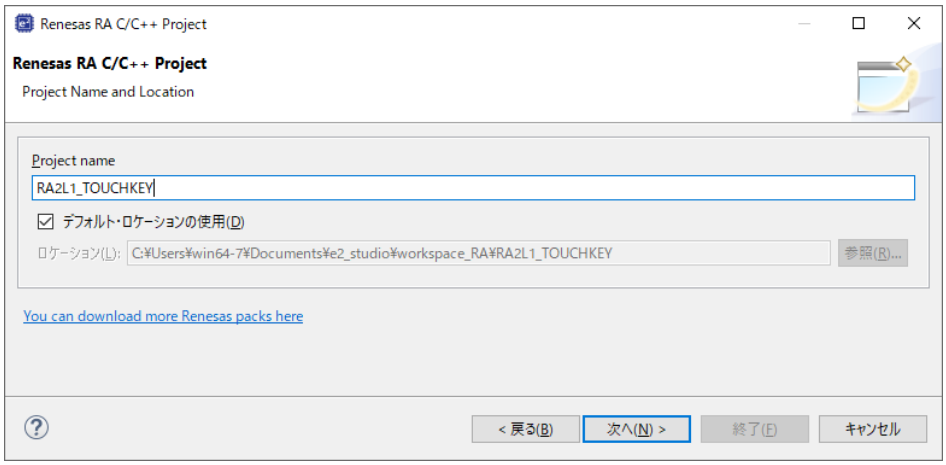
```
/***********************************************************************************************
***************
* Function Name: r_Config_UART0_callback_sendend
* Description  : This function is a callback function when UART0 finishes transmission.
* Arguments    : None
* Return Value : None
***********************************************************************************************
**************/
static void r_Config_UART0_callback_sendend(void)
{
    /* Start user code for r_Config_UART0_callback_sendend. Do not edit comment generated here */
    intr_sci_send_end();
    /* End user code. Do not edit comment generated here */
}

/***********************************************************************************************
***************
* Function Name: r_Config_UART0_callback_receiveend
* Description  : This function is a callback function when UART0 finishes reception.
* Arguments    : None
* Return Value : None
***********************************************************************************************
**************/
static void r_Config_UART0_callback_receiveend(void)
{
    /* Start user code for r_Config_UART0_callback_receiveend. Do not edit comment generated here */
    intr_sci_receive_end();
    /* End user code. Do not edit comment generated here */
}

/***********************************************************************************************
***************
* Function Name: r_Config_UART0_callback_error
* Description  : This function is a callback function when UART0 reception error occurs.
* Arguments    : err_type -
*                    error type info
* Return Value : None
***********************************************************************************************
**************/
static void r_Config_UART0_callback_error(uint8_t err_type)
{
    /* Start user code for r_Config_UART0_callback_error. Do not edit comment generated here */
    intr_sci_receive_error();
    /* End user code. Do not edit comment generated here */
}
```

RL78 に固有の設定は以上です。続きは、8 章を参照してください。

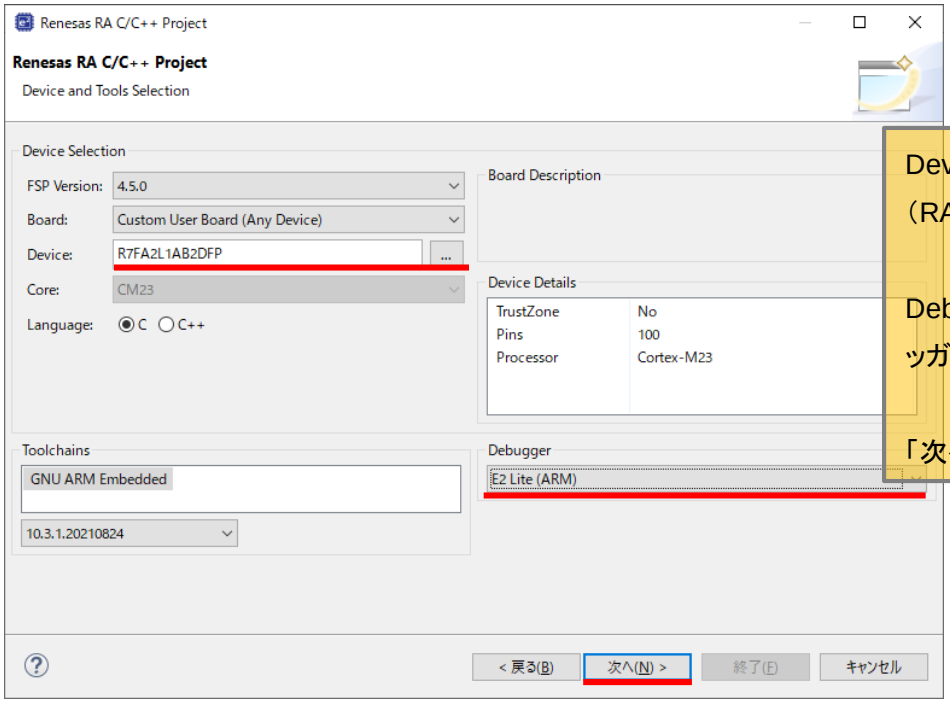
## 7.3. RA

e2studio と FSP を使用しますので、これらのツールを使用する環境が整っていない場合は、予め RA 向けの FSP パッケージをインストールして頂きたく。

・e2studio でプロジェクトを作成



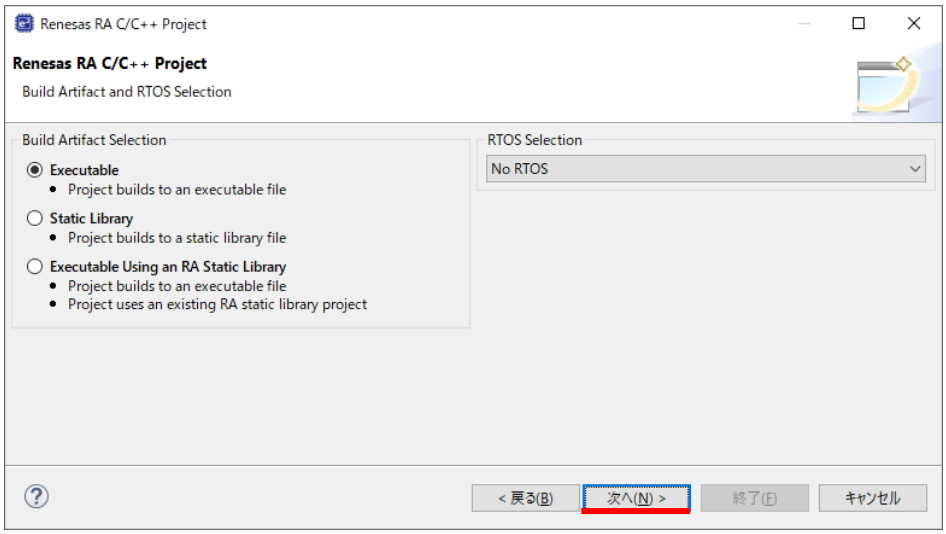

プロジェクト名称は任意です。



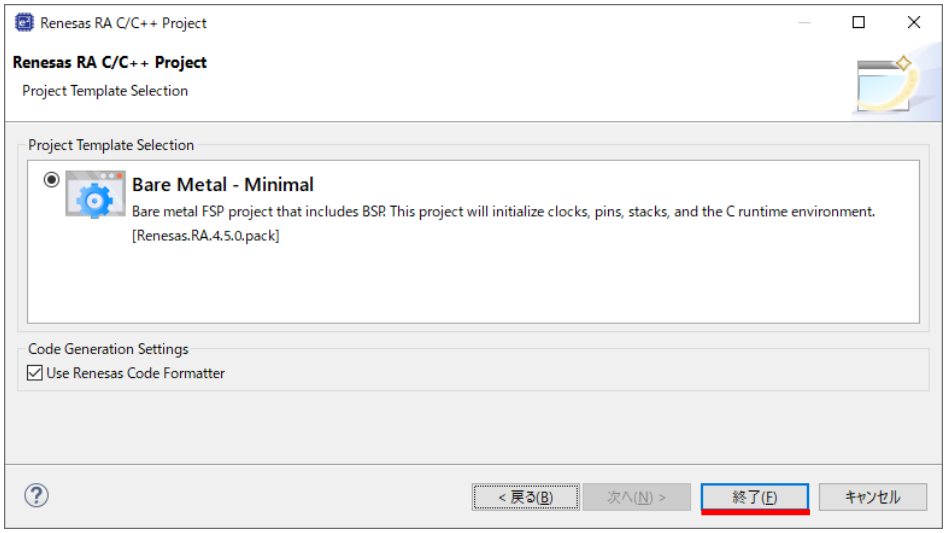

Device R7F2L1AB2DFP を選択
（RA2 - RA2L1 - 100pin 以下）

Debugger 使用する場合お手持ちのデバッガを選択

「次へ」

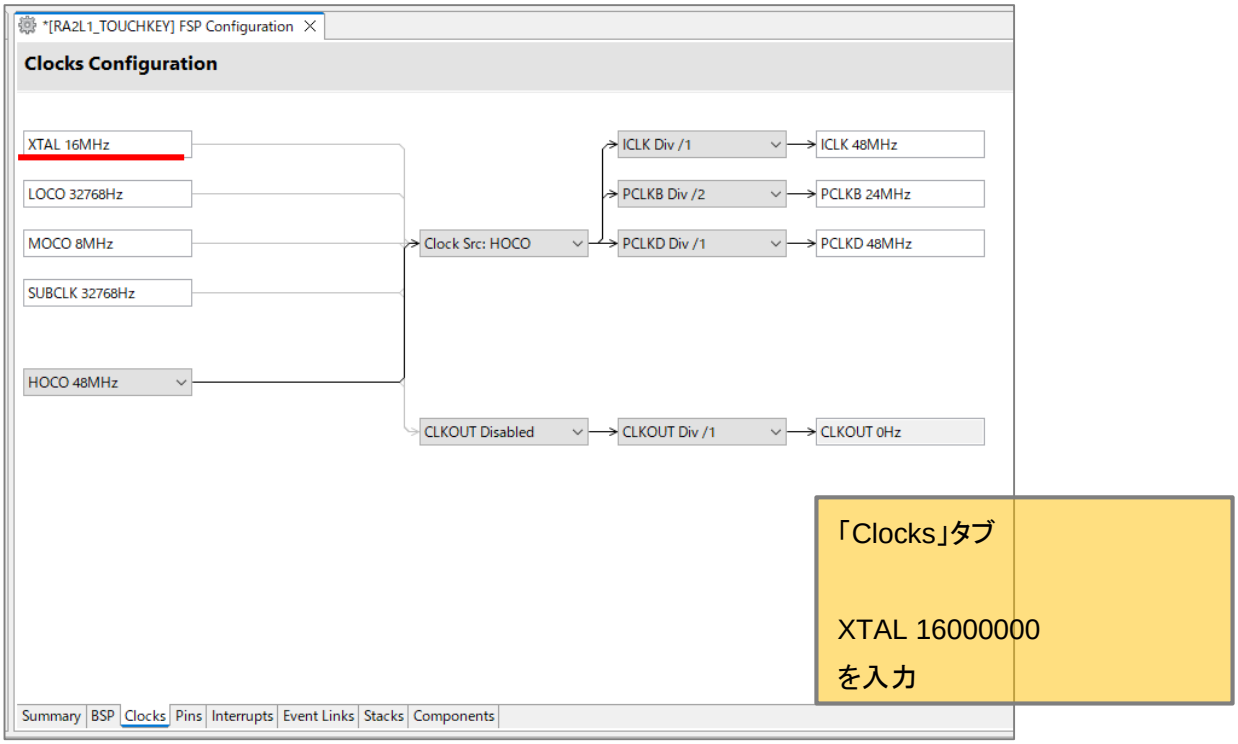



Executable
RTOS: No RTOS
「次へ」



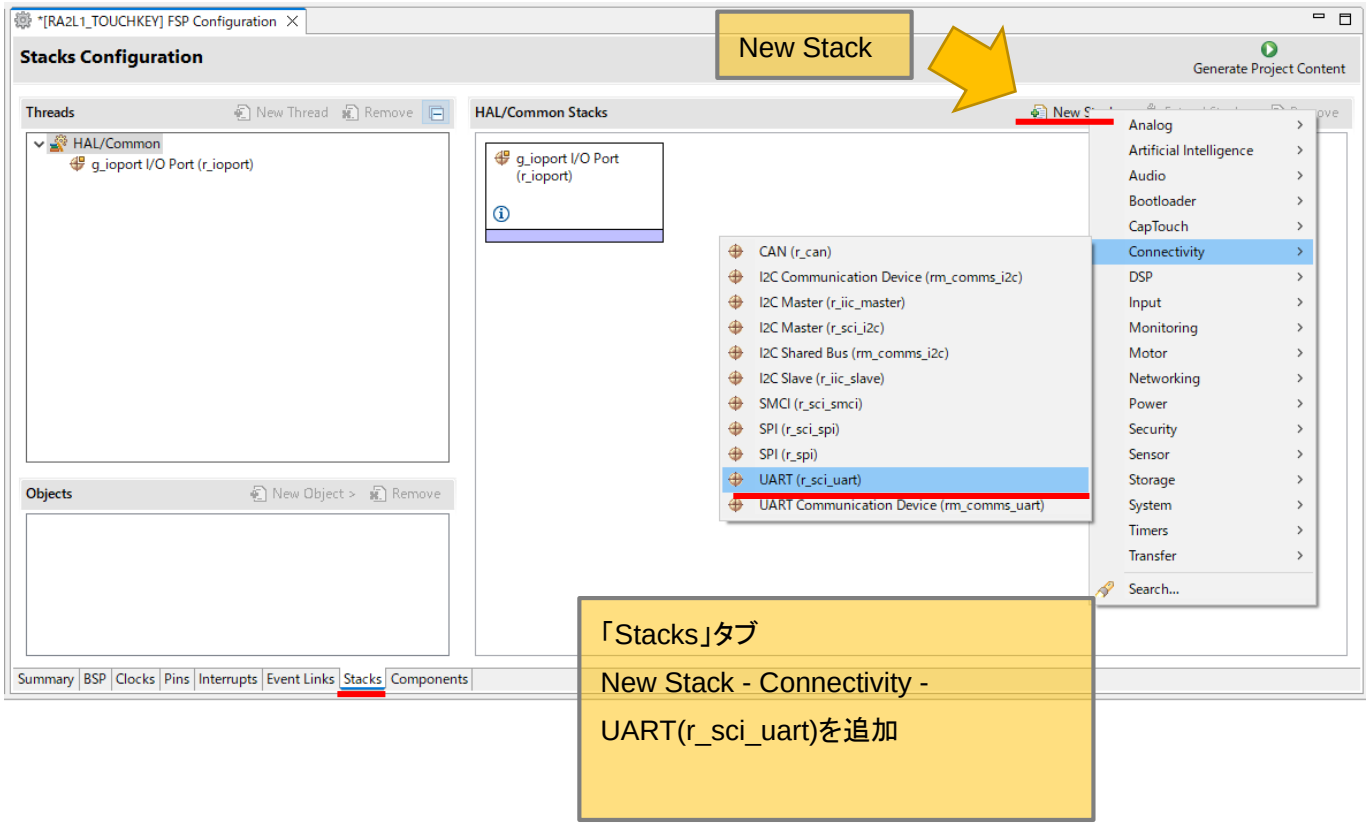

「終了」

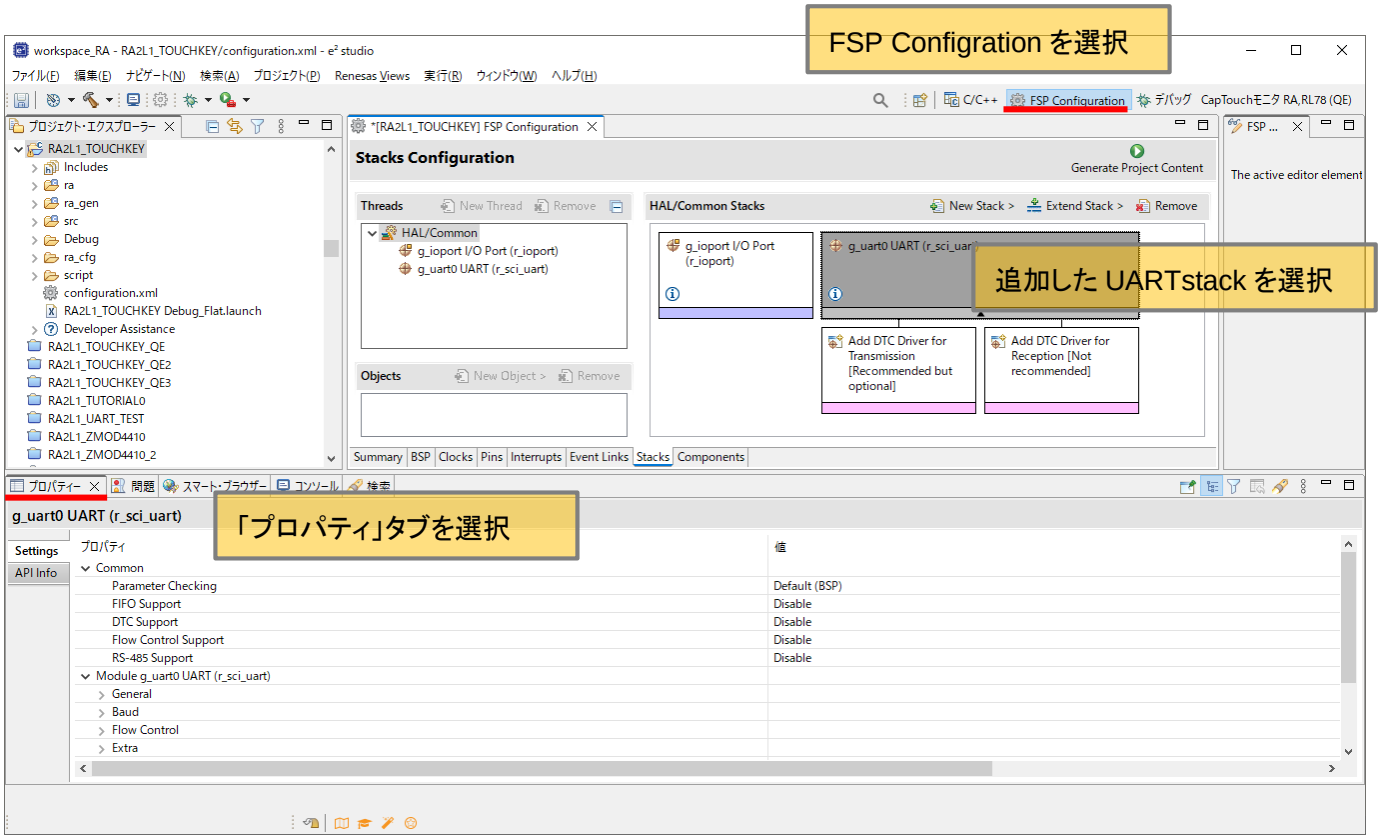
・FSP のクロック設定を行う

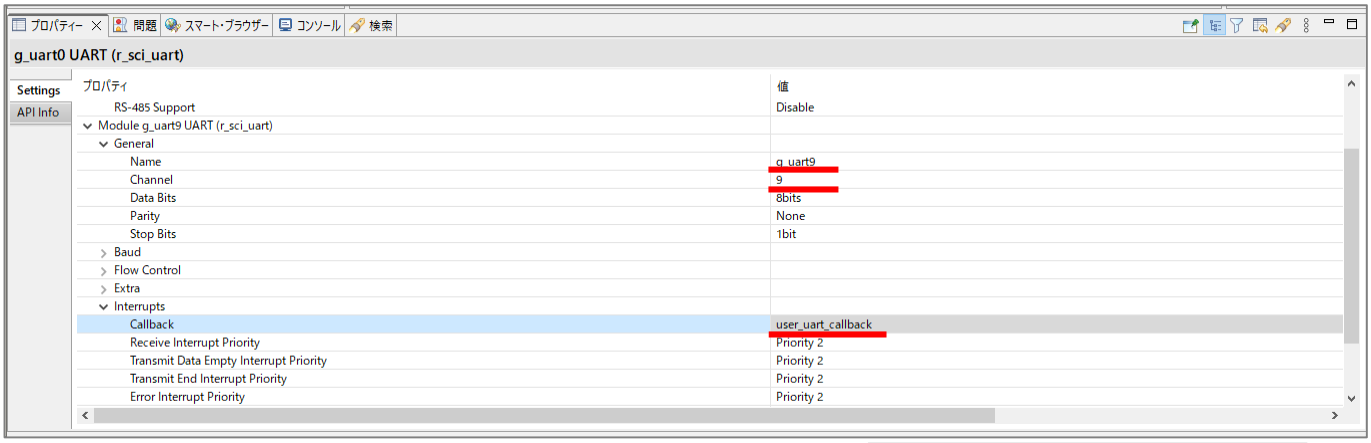



今回のプログラムでは使用致しませんが、ボードに搭載されている XTAL に合わせて 16MHz に設定する。

・FSP で UART(SCI)のコンポーネントを追加する



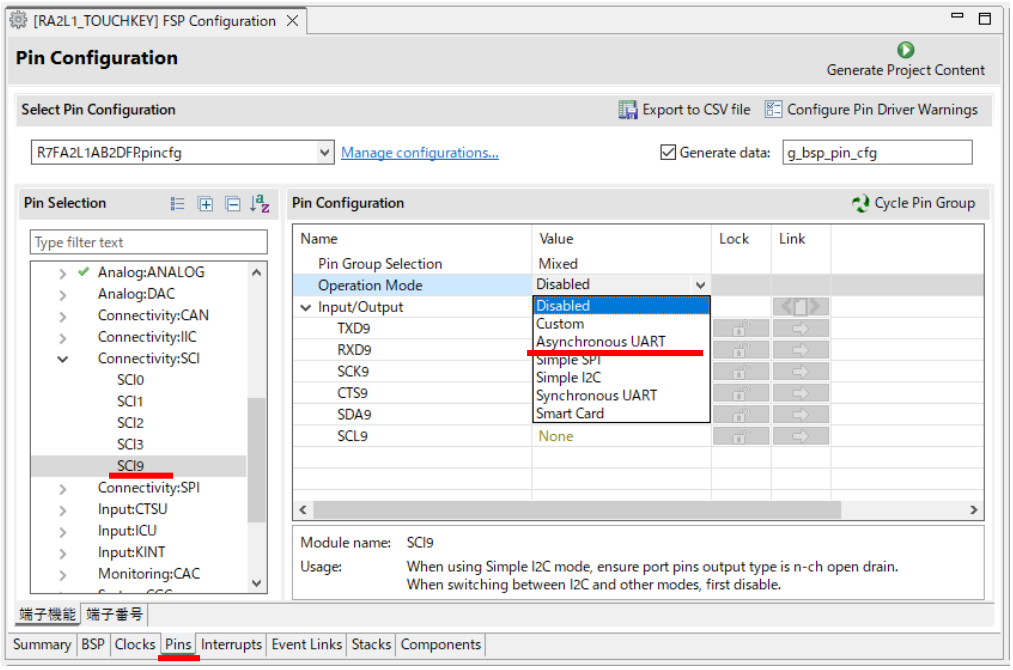


プロパティ値を変更
（3 箇所）

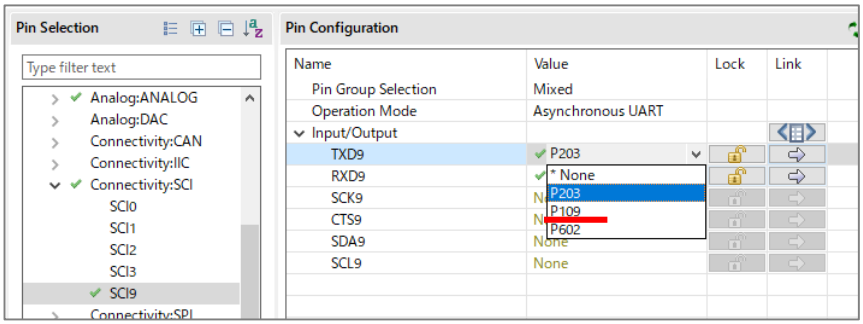
| グループ | プロパティ名 | 変更後の値 |
|---|---|---|
| General | Name | g_uart9 |
| General | Channel | 9 |
| Interrupts | Callback | user_uart_callback |

・UART の端子設定




pins タブの SCI9 の端子設定を開く

Operation Mode : Asyncronous UART を選択

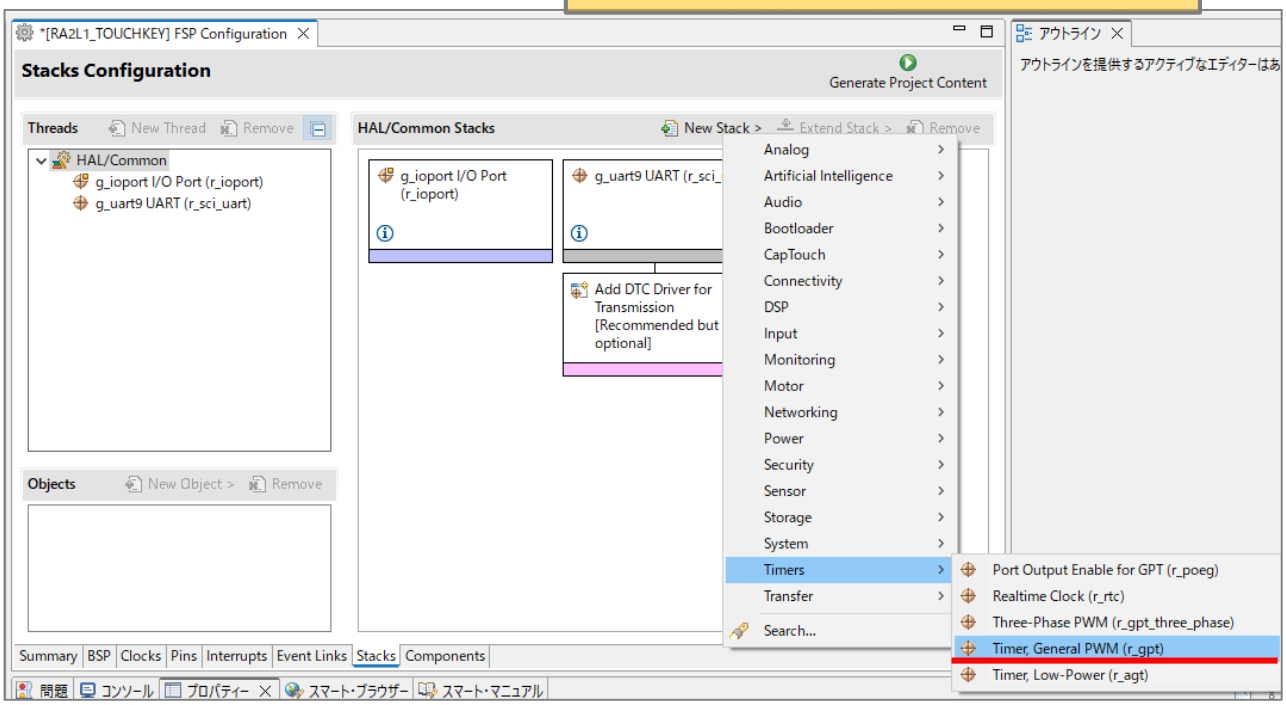


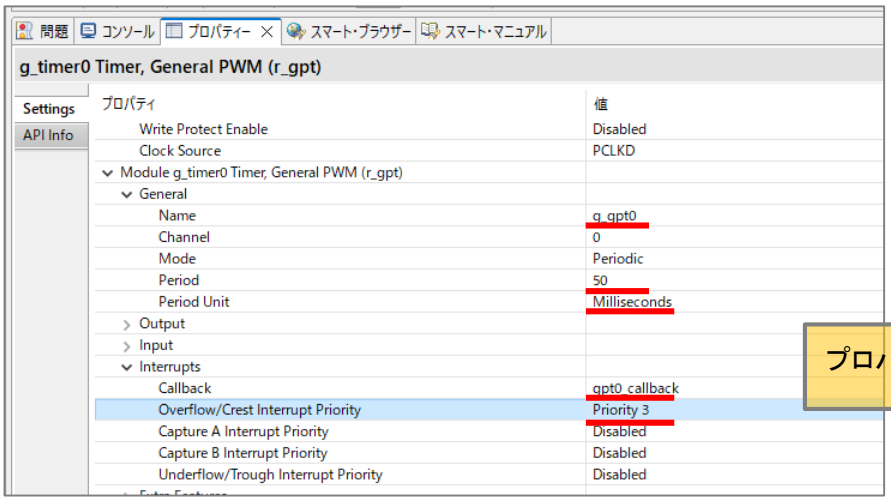

TXD9 P109 を選択
RXD9 P110 を選択

TXD9, RXD9 に割り当てられている端子を変更する。

・FSP で GPT（タイマ）を追加する

New Stack
Timers - Timer, General PWM(r_gpt) を追加



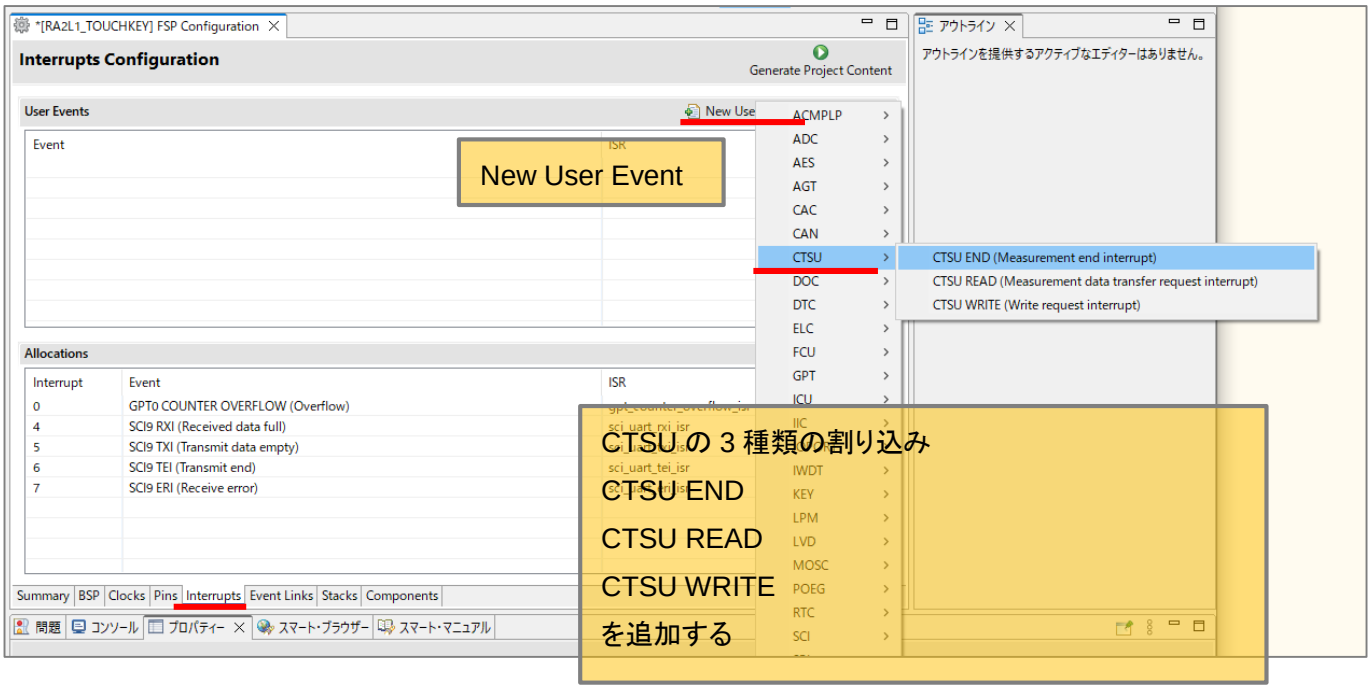




プロパティ値を変更する

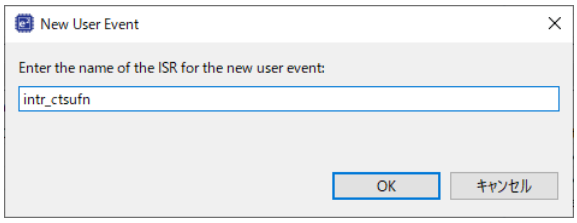
| グループ | プロパティ名 | 変更後の値 | 備考 |
|---|---|---|---|
| General | Name | g_gpt0 | |
| General | Period | 50 | 測定周期（設定値は任意） |
| General | Period Unit | Milliseconds | |
| Interrupts | Callback | gpt0_callback | |
| Interrupts | Overflow/Crest Interrupt Priority | Priority 3 | 割り込み優先度は 3 でなくても構いません |

ここで設定した時間が測定周期（50ms であれば、1 秒間に 20 回）となります。

・割り込みの定義の追加




Interrupts タブ、New User Event - CTSU  以下の 3 種類の割り込みを追加する。



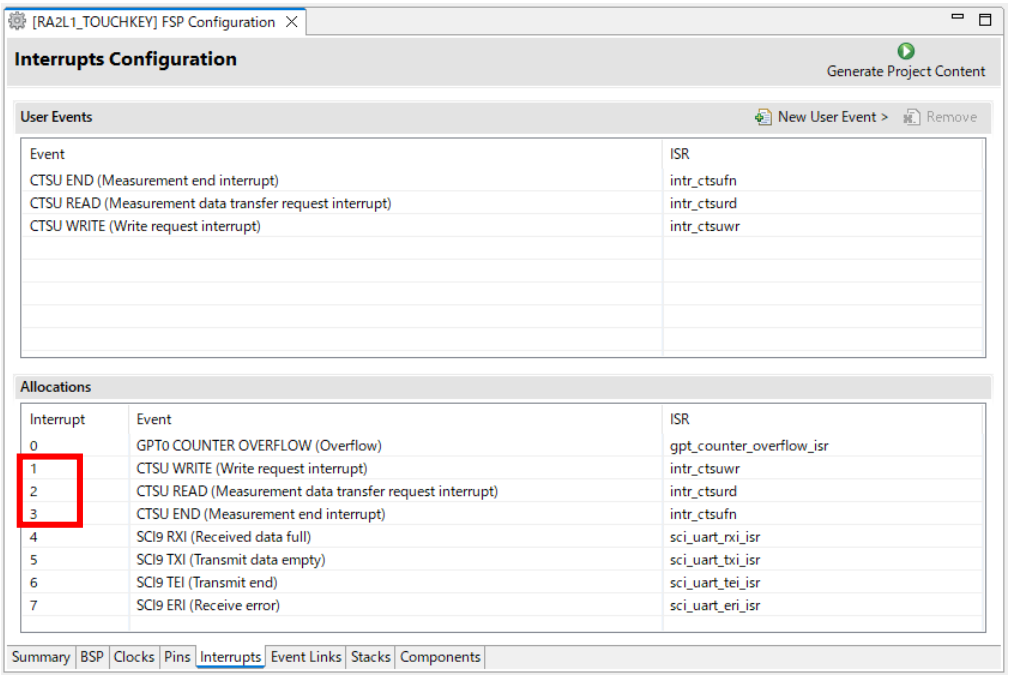

割り込みタイプ → 関数名

CTSU END → intr_ctsufn

CTSU READ → intr_ctsurd

CTSU WRITE → intr_ctsuwr

上記名称で入力する。




| 割り込み番号 | Event | ISR |
|---|---|---|
| 1 | CTSU WRITE | intr_ctsuwr |
| 2 | CTSU READ | intr_ctsurd |
| 3 | CTSU END | intr_ctsufn |

上記番号が 1~3 以外になっている場合は、

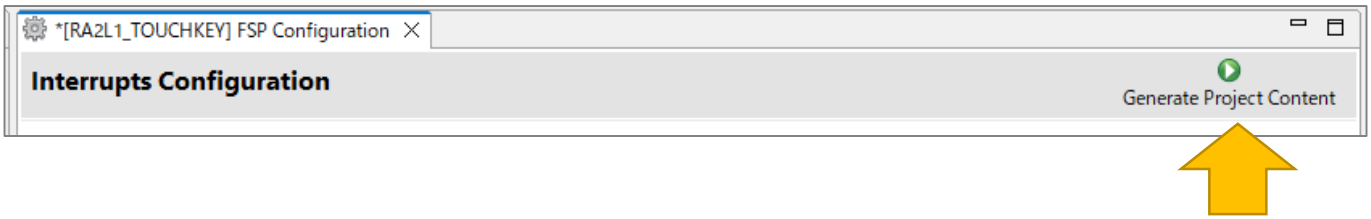
ctsu2/ctsu2_mcu_depend.h

内の番号を、FSP の割り込み設定（上記設定番号）に合わせて書き換えてください。

ctsu2/ctsu2_mcu_depend.h

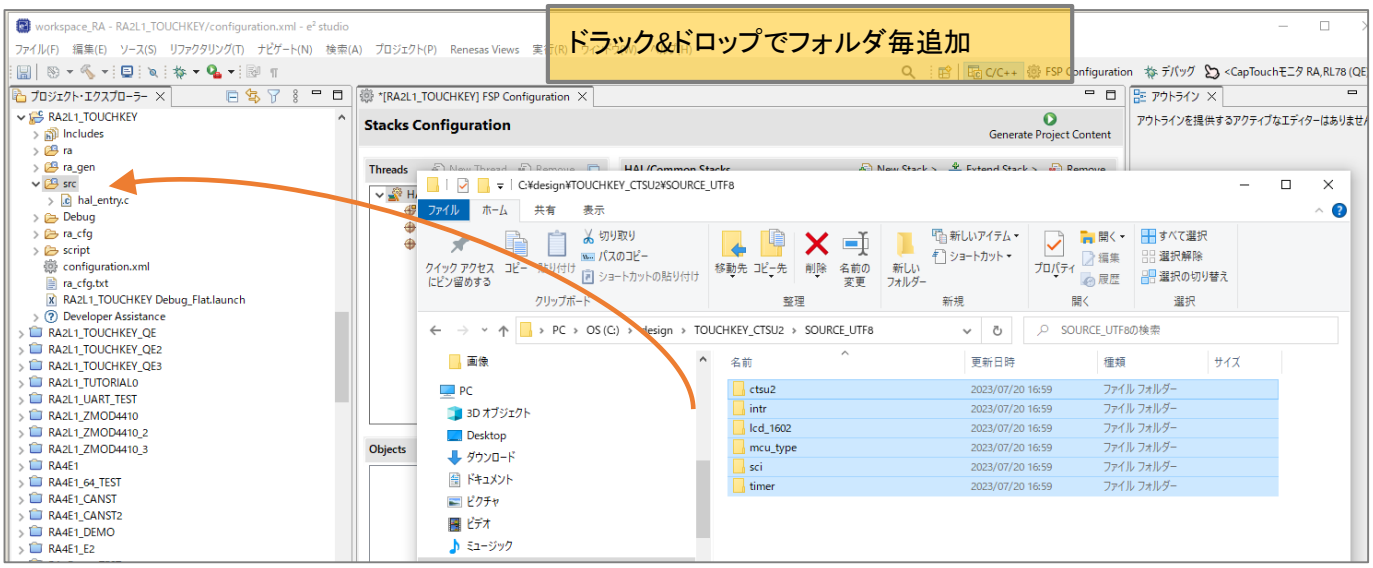
```
//割り込み番号（FSPで設定した番号）
#define INTR_CTSU_CTSUWR (1)
#define INTR_CTSU_CTSURD (2)
#define INTR_CTSU_CTSUFN (3)
```

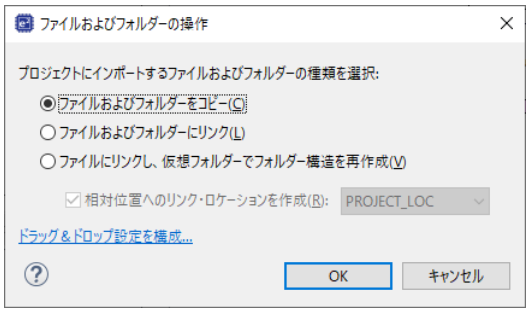
(1)~(3)の数値を、FSP の割り込み番号に合わせてください。




一通り、FSP の設定が終わったタイミングで、Generate Project Content のボタンを押して、FSP の設定をソースコードに反映させてください。

・ソースファイルをプロジェクトに追加




OK を押す

RA に固有の設定は以上です。続きは、8 章を参照してください。

# 8. ソースコードのファイルの書き換え

次にソースコードのファイルを書き換えます。

## (1)マイコン種

mcu_type/mcu_type.h

```
/*-----------------------------------------------------------------------
マイコン種別（いずれか1つを選択）
-----------------------------------------------------------------------*/

#define MCU_TYPE_RX
//#define MCU_TYPE_RL78
//#define MCU_TYPE_RA
```

3 行のうち 1 つのみ、コメントアウトを外してください。

マイコン種により、処理を切り替える部分で使用されます。

## (2)タッチキーパッドタイプ

ctsu2/ctsu2.h

```
//使用するタッチキー基板タイプ(*) ※どちらか一方を選択

#define CTSU_TYPE CTSU_MUTUAL_CAP_S16A
//#define CTSU_TYPE CTSU_SELF_CAP_D55A
```

自己容量(S16A)タイプのキーパッドを使用するか、相互容量(D55A)タイプのキーパッドを使用するかで、どちらかの行のコメントアウトを外してください。

## (3)電源電圧

ctsu2/ctsu2.h

```
//電源電圧 VCC/VDD=4.5V 未満の時に定義を有効にする
//#define CTSU_VOLTAGE_UNDER4_5                    (*1)

//電源電圧 VCC/VDD=2.4V 未満の時に定義を有効にする
//#define CTSU_VOLTAGE_UNDER2_4                    (*2)
```

電源電圧を 4.5V 未満で使用する場合は、(*1)のコメントアウトを外してください。
さらに、電源電圧 2.4V 未満で使用する場合は、(*2)のコメントアウトを外してください。

## (4)メイン関数の追加（RX と RL78 の場合）

RX140_CTSU2_SAMPLE.c （RX の場合、作成したプロジェクト名.c）
main.c （RL78 の場合）

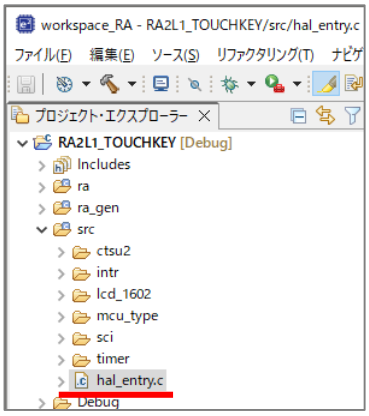
```
#include "ctsu2.h"

void main(void)
{
    ctsu_main();

    while(1);
}
```

黄色でマーキングした行を追加。

RX と RL78 の場合は、CS+で生成されたメイン関数(main())内に、ctsu_main()を追加してください。
（while(1)は無くても構いません、念のために記載しているものです。）

## (4')メイン関数の追加（RA の場合）

・hal_entry.c の書き換え




src
　hal_entry.c をダブルクリック

```
#include "hal_data.h"

#include "ctsu2/ctsu2.h"

FSP_CPP_HEADER
void R_BSP_WarmStart(bsp_warm_start_event_t event);
FSP_CPP_FOOTER

/*******************************************************************************************************************//**
 * main() is generated by the RA Configuration editor and is used to generate threads if an RTOS is used.  This function
 * is called by main() when no RTOS is used.
 **********************************************************************************************************************/
void hal_entry(void)
{
    /* TODO: add your own code here */
    ctsu_main();

    while(1);

#if BSP_TZ_SECURE_BUILD
    /* Enter non-secure code */
    R_BSP_NonSecureEnter();
#endif
}

（後略）
```

黄色でマーキングした行を追加。

設定が必須なのは、上記(1)～(4)です。

# 9. サンプルプログラムの動作説明

具体的なプログラムの動作に関して説明します。基本的な流れは 5 章で説明しているものとなります。

## 9.1. メイン処理

メイン関数から呼び出す事を想定して作成している部分です。サンプルプログラムでは、測定値の表示等を行っています。

・ctsu2/ctsu2_self_cap_main.c　　[自己容量向け]
・ctsu2/ctsu2_mutual_cap_main.c　[相互容量向け]

自己容量キーパッド(S16A)を使う場合は、_self_cap が付くファイル。相互容量キーパッド(D55A)を使う場合は、_mutual_cap が付くファイルを使います。どちらのファイル内でも、

ctsu_main()

という関数が定義されています。

ctsu2/ctsu2.h

内の、

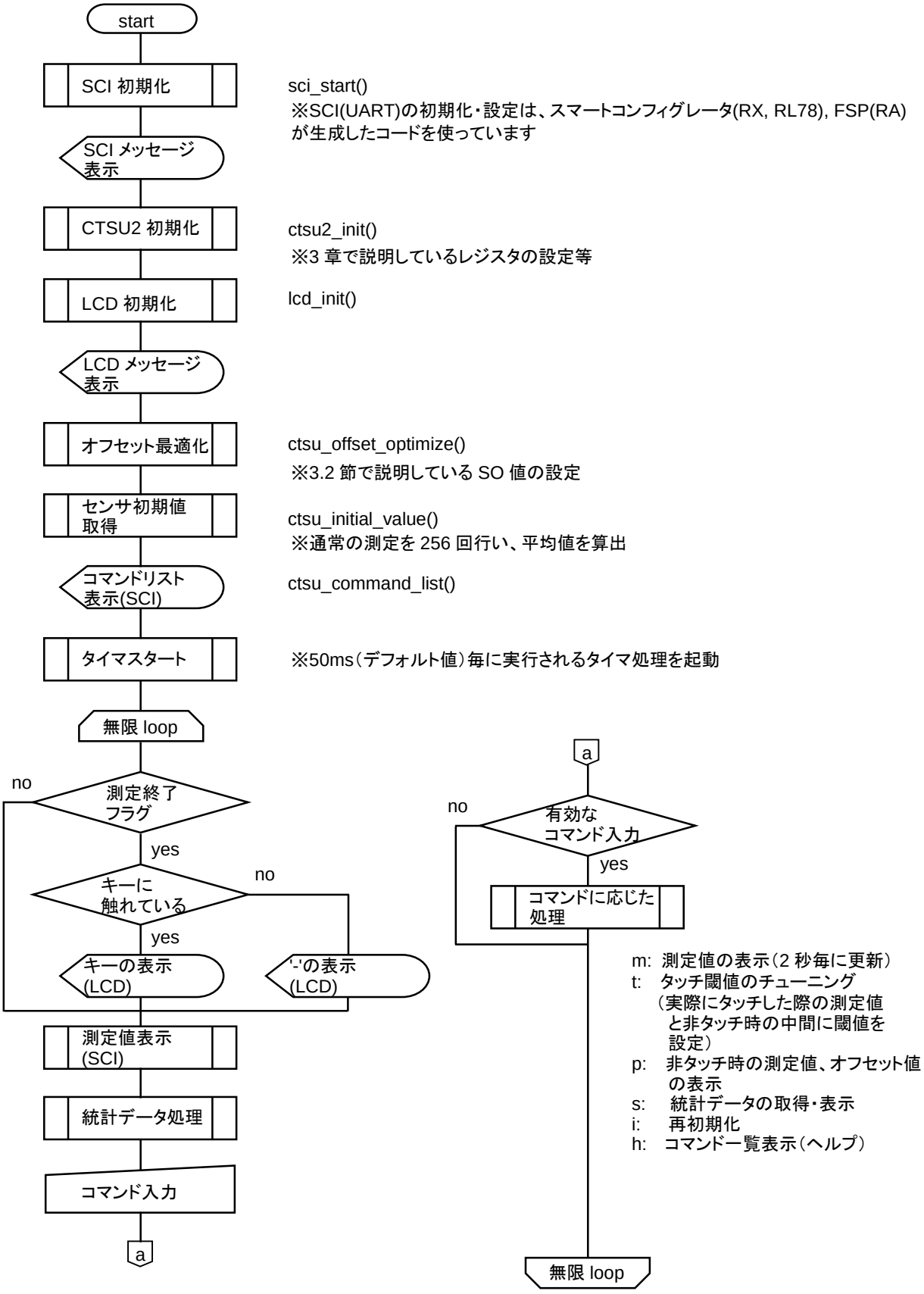
```
//使用するタッチキー基板タイプ(*) ※どちらか一方を選択
//#define CTSU_TYPE CTSU_SELF_CAP_S16A
#define CTSU_TYPE CTSU_MUTUAL_CAP_D55A
```

有効にした方

CTSU_TYPE -> CTSU_SELF_CAP_S16A の場合は、ctsu2_self_cap_main.c内の ctsu_main()
CTSU_TYPE -> CTSU_MUTUAL_CAP_D55A の場合は、ctsu2_mutual_cap_main.c内の ctsu_main()

が使われるようになっています。

－メイン処理　ctsu_main()　フローチャート－



メイン処理では、タッチキー(CTSU2)関連の処理として、
・初期化 ctsu_init()
・オフセット値最適化 ctsu_offset_optimize()
・初期値の取得 ctsu_initial_value()
を行い、タイマを起動しています。

また、キーボードからのコマンド入力を受け付け、各種コマンドの処理を行っています。

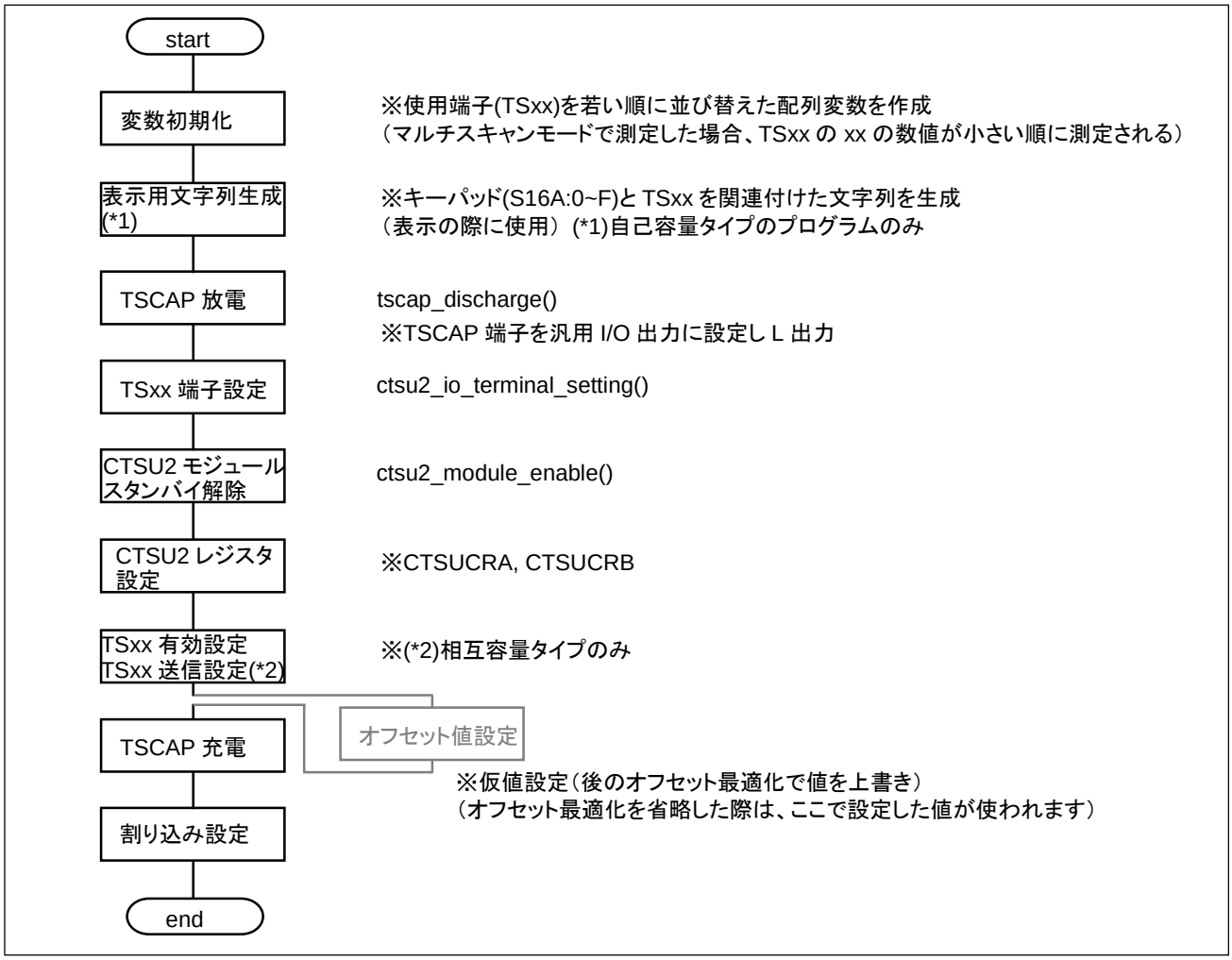
## 9.2. CTSU2 関連処理

タッチキー関連の処理としては、

・ctsu2/ctsu2_self_cap.c [自己容量向け]
・ctsu2/ctsu2_mutual_cap.c [相互容量向け]

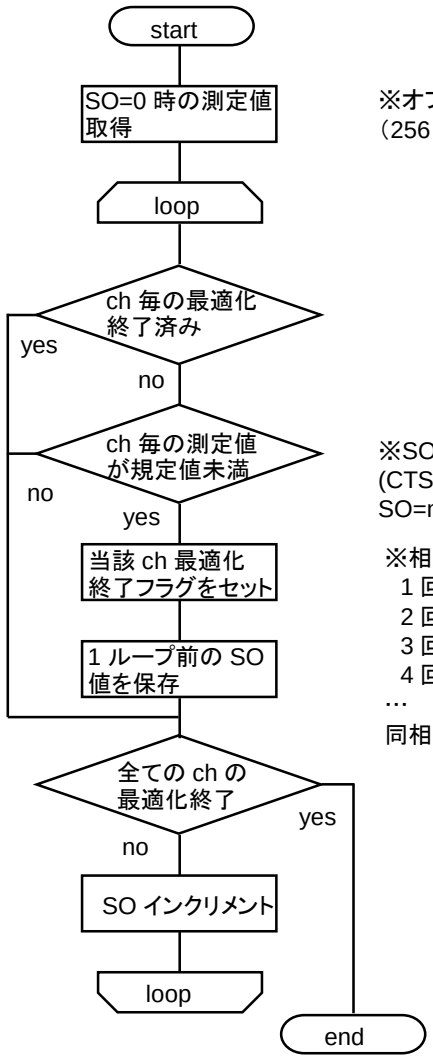
に記載されており、メイン処理同様 ctsu2.h 内で選択した、CTSU_TYPE でどちらのファイル内の記載が使われるかが変わります。

ー初期化 ctsu_init() フローチャートー



## 9.2.1. オフセット調整

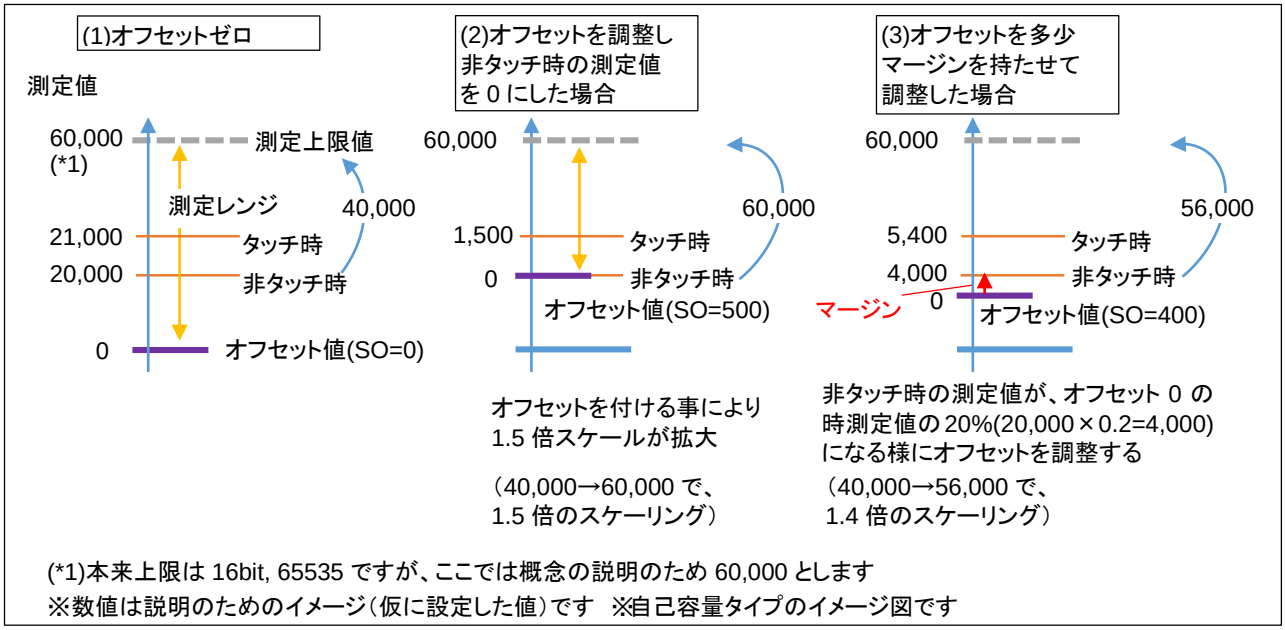
－オフセット最適化　ctsu_offset_optimize()　フローチャート－




※オフセットをかけない場合の端子毎の測定値を取得
（256 回測定して平均値を算出）

※SO=0 の時の測定値に対し SO=n の時の測定値が 20%
(CTSU_OFFSET_MERGIN=0.2)未満で当該 ch の最適化を終了
SO=n-1 の値を変数に保存

※相互容量タイプでは、
1 回目の測定値（同相） ★比較対象
2 回目の測定値（逆相） 未使用
3 回目の測定値（同相） ★比較対象
4 回目の測定値（逆相） 未使用
…
同相測定の結果のみ使用する




オフセットを付ける事により
1.5 倍スケールが拡大

（40,000→60,000 で、
1.5 倍のスケーリング）

非タッチ時の測定値が、オフセット 0 の時測定値の 20%(20,000×0.2=4,000)になる様にオフセットを調整する

（40,000→56,000 で、
1.4 倍のスケーリング）

(*1)本来上限は 16bit, 65535 ですが、ここでは概念の説明のため 60,000 とします
※数値は説明のためのイメージ（仮に設定した値）です　※自己容量タイプのイメージ図です

(1)オフセットゼロ(SO=0)

オフセットを掛けない場合は、測定レンジ 0-60,000 の内、0-20,000 の間は無駄になっています。20,000-60,000 の 40,000 の範囲内で測定している状態です。

(2)非タッチ時の容量分をキャンセルするオフセット量

ここで、非タッチ時の測定値 20,000 の点までオフセットをかけると、オフセット(SO=)0 の 20,000 の点が新たに 0 に設定されますので、(1)に対し 1.5 倍のスケーリングが掛かった状態となります。(1)でタッチと非タッチの差分が 1,000 でしたが、(2)では 1,500 に拡大します。

この状態が一番測定レンジを有利に使っている事になりますが、0 点（紫のライン）が特性変動（電圧変動や温度変動）により上昇した場合、今度はオフセットの付け過ぎといった状態となります。仮に、紫のラインが 1,500 より上に来た場合、

非タッチ時の測定値 ＝ 0

タッチ時の測定値 ＝ 0

となります。（測定不能）

(3)限界から 20%マージンを取ったオフセット量

そこで、目一杯オフセットをかける訳ではなく、多少小さめのオフセットをかける事にします。オフセット 0(SO=0)の時の非タッチ時の測定値 20,000 を基準にして、20%マージンを残すという考え方です。

オフセット値(SO)を大きくしていき、非タッチ時の測定値が、20,000 × 0.2 = 4,000 となった時点で、オフセット調整終了とします。

0.2 という数値は

ctsu2/ctsu2.h

内で

```
//電流オフセット最適化
#define CTSU_OFFSET_MERGIN (0.2f)//オフセット 0 の値の 20%の点にオフセットを設定
```

定義していますので、他の値に変える事も可能です。

オフセットを変えた場合の自己容量タイプの測定値の変化イメージ

| | 非タッチ時の測定値 | タッチ時の測定値 | 差分 |
|---|---|---|---|
| (1)オフセットゼロ | 20,000 | 21,000 | 1,000 |
| (2)非タッチ時の容量分をキャンセルするオフセット量 | 0 | 1,500 | 1,500 |
| (3)限界から 20%マージンを取ったオフセット量 | 4,000 | 5,400 | 1,400 |

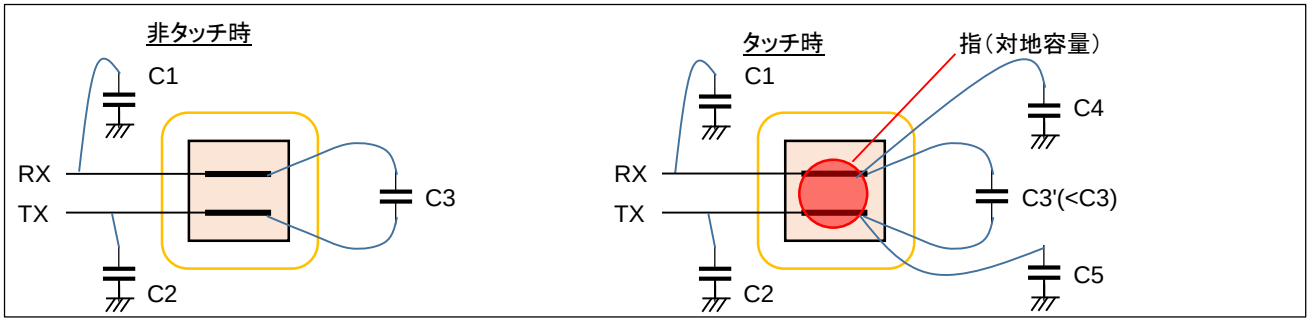
本サンプルプログラムでは(3)を採用します。

本プログラムでは、上記の様なアルゴリズムでオフセット値を決めています。(この手法が必ずしも正解という事ではないと思います。色々な考え方があると思いますが、ある程度シンプルな考え方でオフセット値を決める一案としてサンプルプログラムで採用しました。)

測定 ch(自己容量タイプ:S16A ではキーパッドの数)毎に、非タッチ時の容量(配線やパッド等)は異なりますので、オフセット調整は測定 ch 毎行い、測定 ch 毎に別な値を適用しています。

相互容量では、キーパッド数 25 に対して、測定 ch は 50 となります(同相測定と逆相測定)。

・相互容量の場合




| | 同相測定 | 逆相測定 | **差分<br>(逆相-同相)** |
|---|---|---|---|
| 非タッチ時 | (1):C1 | (2):C1+2×C3 | **2×C3** |
| タッチ時 | (3):C1+C4 | (4):C1+C4+2×C3' | **2×C3'** |

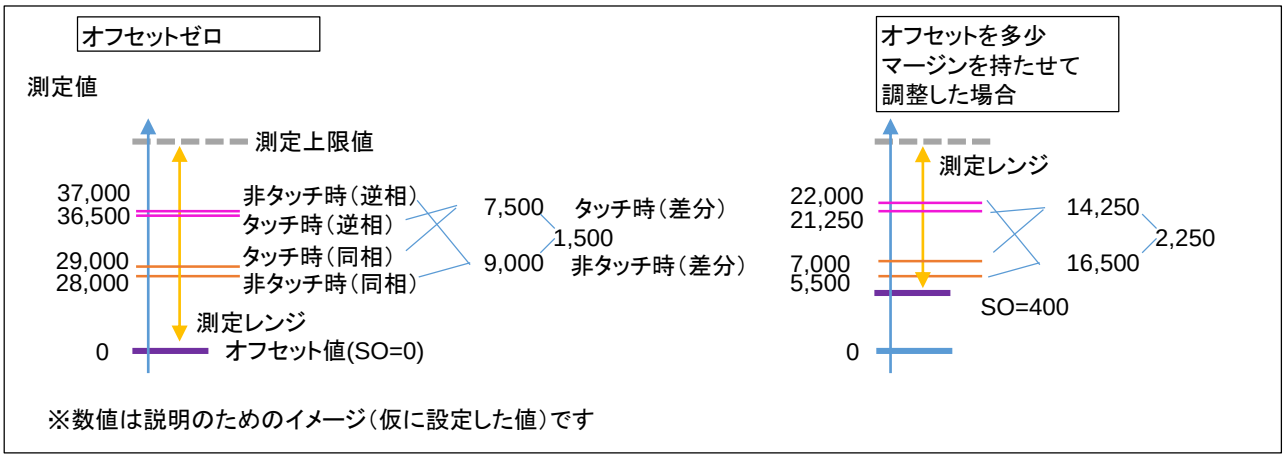
相互容量の場合は、逆相測定の結果から同相測定の結果を差し引く事により、非タッチ時の容量(C1)を測定値から取り除く事が出来ます。

自己容量のオフセット調整は、非タッチ時の容量(C1 相当)の成分を測定値から取り除いてスケーリングする事を目的としていましたが、相互容量でも、同様です。(相互容量の場合、オフセット調整を行わなくても、C1 の値は逆相－同相の演算で取り除く事ができている。オフセット調整で C1 の成分を取り除いた場合は、C3, C3'の差を目立たせる事が可能。)

同相測定時と、逆相測定時でオフセット(＝スケーリングに相当)を別な値とすると、逆相－同相の演算で C1 を取り除く事が出来なくなります。よって、同相測定と逆相測定時のオフセット値は同じ値を用いる事とします。

(1)非タッチ:同相測定
(2)非タッチ:逆相測定
(3)タッチ:同相測定
(4)タッチ:逆相測定

この 4 条件で一番測定値が小さいのは(1)です。よって、(1)の測定値をベースにオフセットを決める事とします。オフセットの決め方は、自己容量タイプと同様、オフセットゼロ(SO=0)の測定値の、20%となる値をオフセット値とします。




相互容量タイプでも、自己容量と同様にオフセット調整で測定レンジを大きく取る事ができ、差分を大きく測定する事が可能です。

### 9.2.2. タッチ判定

・自己容量

測定値が、10%以上増加した際、タッチしていると判断します。

例えば、非タッチ時の測定値 4,000 の場合、測定値が 4,400 以上となった際タッチしていると見なします。

・測定値の増加率で判断する
・測定値の増加幅で判断する

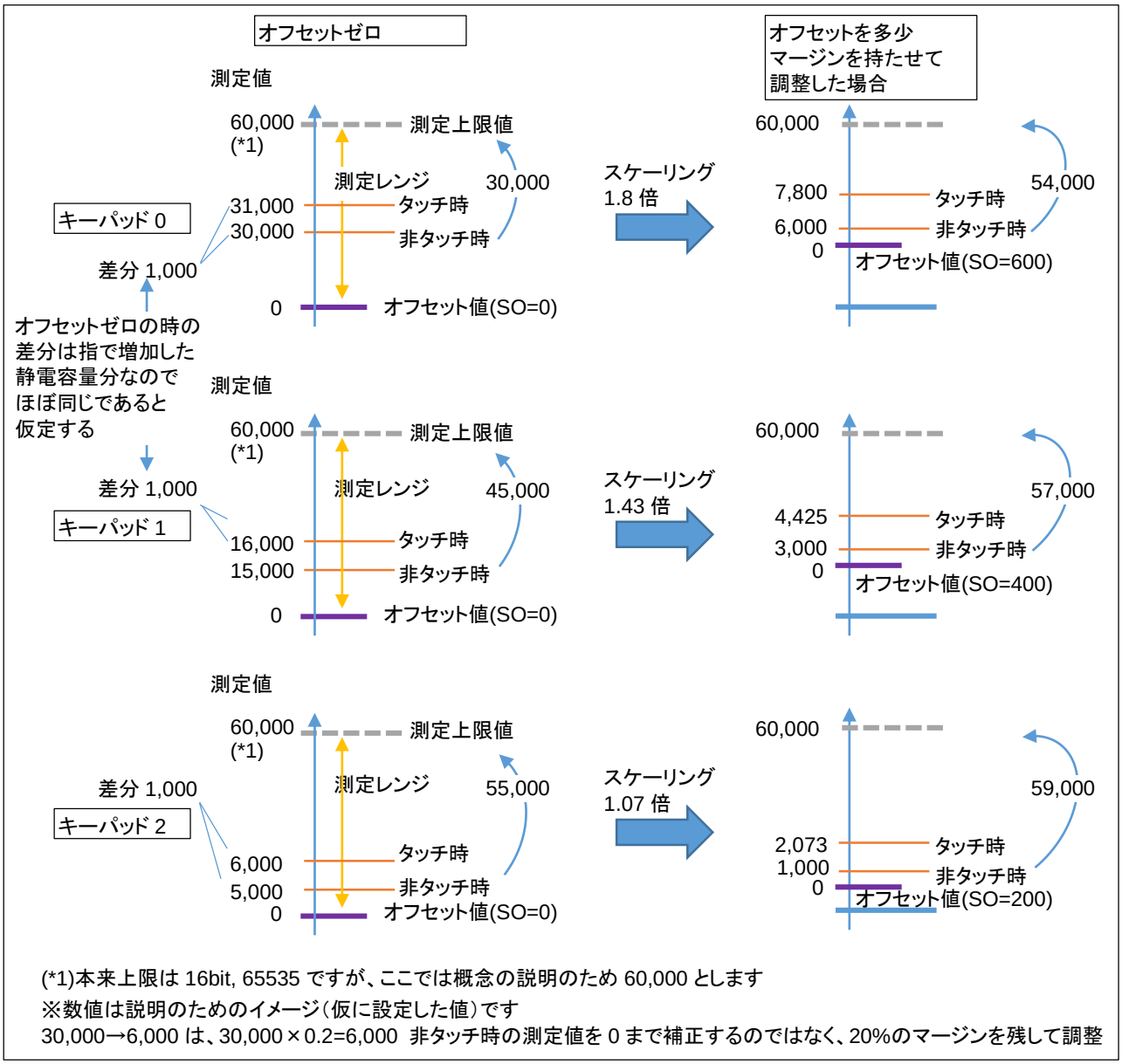
考え方としては、難しいところではあるのではないかと思います。本サンプルプログラムでは前者としています。

仮に、下記のように、非タッチ時の測定値の触れ幅が大きい場合

| | 非タッチ時の<br>測定値(V1) | タッチ時の<br>測定値(V2) | 測定値<br>増加幅(V2-V1) | 測定値<br>増加率(1-V2/V1) | 備考 |
|---|---|---|---|---|---|
| ケース 1 | 4,000 | 5,400 | 1,400 | 35% | オフセット調整を行った場合のイメージ |
| ケース 2 | 20,000 | 21,000 | 1,000 | 5% | オフセットゼロのイメージ |

測定値の増加率で考えると、大きくばらつきが出ます。こういう（一般的にオフセット調整が掛かっていない）ケースでは、増加率で考えるのは妥当ではないと思います。
（ケース 1 では、非タッチ時の測定値+10%の点を閾値に設定しても問題ないが、ケース 2 では非タッチ時の測定値+10%の点を閾値に設定すると、タッチ判定ができない。）

次に、オフセットゼロ時の（非タッチ時の）測定値が、30,000 になるキーパッドと 15,000 になるキーパッド、及び、5,000 になるキーパッドがあったと仮定します。
（キーパッド毎に配線長=容量は異なるので、多少はばらつきが出ます。ここでは極端なケースで仮定します。）




(*1)本来上限は 16bit, 65535 ですが、ここでは概念の説明のため 60,000 とします
※数値は説明のためのイメージ（仮に設定した値）です
30,000→6,000 は、30,000×0.2=6,000　非タッチ時の測定値を 0 まで補正するのではなく、20%のマージンを残して調整

・オフセット調整後の測定値イメージ

| | 非タッチ時の<br>測定値(V1) | タッチ時の<br>測定値(V2) | 測定値<br>増加幅(V2-V1) | 測定値<br>増加率(1-V2/V1) | 備考 |
|---|---|---|---|---|---|
| キーパッド 0 | 6,000 | 7,800 | **1,800** | 30% | スケーリング大 |
| キーパッド 1 | 3,000 | 4,425 | **1,425** | 47.5% | スケーリング中 |
| キーパッド 2 | 1,000 | 2,073 | **1,073** | 107% | スケーリング小 |

スケーリングで考えると、キーパッド 0 が一番スケーリング効果が大きく、測定値の変動幅では一番マージンが大きくなります。

仮に、閾値を「増加幅」ベースで決めたとすると、この場合、スケーリングの小さな条件では、動作マージンが小さくなります(*1)。増加幅で考えた場合、スケーリングと動作マージンは正の相関を持ちます。

・閾値をキーパッド 1 の変動幅(4425-3000=1425)の中間(V1+713)に設定した場合[変動幅ベースでの定義]

| | スケーリング | 非タッチ時の測定値(V1) | タッチ時の測定値(V2) | 閾値 V1+713 | **動作マージン（幅）** | 動作マージン（率） |
|---|---|---|---|---|---|---|
| キーパッド 0 | 1.800 | 6,000 | 7,800 | 6,713 | -713<br>**+1087(*1)** | -10.6%<br>+16.2% |
| キーパッド 1 | 1.425 | 3,000 | 4,425 | 3,713 | -713<br>**+712(*1)** | -19.2%<br>+19.2% |
| キーパッド 2 | 1.073 | 1,000 | 2,073 | 1,713 | -713<br>**+360(*1)** | -41.6%<br>+21.0% |

閾値を、（代表的な値として）キーパッド 1 を基準に、1425/2=713 増加した値とする。

動作マージン（率）は、閾値を基準に V1 と V2 がそれぞれ何%離れた値であるかを示しています。

キーパッド 1 を基準に、キーパッド 1 の非タッチとタッチ時の値のセンターに閾値を設定したので、キーパッド 1 はプラス側とマイナス側のマージンは同一。

それに対し、キーパッド 0 はマイナス側のマージンが減少し、キーパッド 2 はプラス側のマージンが減少します。

※動作マージン値に関して、プラス側のマージンが小さければ、「タッチしていないのにタッチ判定される」という事が起こります。マイナス側のマージンが小さければ「タッチしてもタッチ判定されない」という事が起こります。

次に、閾値を「増加率」ベースで決めたとすると、このケースではスケーリングの小さな方が増加率が大きく(*2)なります。測定値の絶対値の小さな方が増加率で考えると大きくなります。

・閾値をキーパッド 1 の変動率の中間(V1+V1×23.8%)に設定した場合[変動率ベースでの定義]

| | スケーリング | 非タッチ時の測定値(V1) | タッチ時の測定値(V2) | 閾値 V1×1.238 | 動作マージン（幅） | **動作マージン（率）** |
|---|---|---|---|---|---|---|
| キーパッド 0 | 1.800 | 6,000 | 7,800 | 7,428 | -1,428<br>+372 | **-19.2%**<br>+5.0%(*2) |
| キーパッド 1 | 1.425 | 3,000 | 4,425 | 3.714 | -714<br>+711 | **-19.2%**<br>+19.1%(*2) |
| キーパッド 2 | 1.073 | 1,000 | 2,073 | 1,238 | -238<br>+835 | **-19.2%**<br>+67.4%(*2) |

閾値を、キーパッド 1 を基準に、47.5%/2=23.8%増加した値とする。

キーパッド 1 を基準にしたので、キーパッド 1 のマージンは変動幅ベースで決めた場合と同一。

ここで、変動率ベースでの定義とした場合の、動作マージン（率）のマイナス側の値は全て同じとなっています。これは偶然ではなく、

動作マージン（率）マイナス側 $= \frac{\cancel{6000}\times 0.238}{\cancel{6000}\times 1.238} = \frac{0.238}{1.238} = 0.192$

V1 の値に拘わらず、23.8%という値を選んだ事で決まっています。

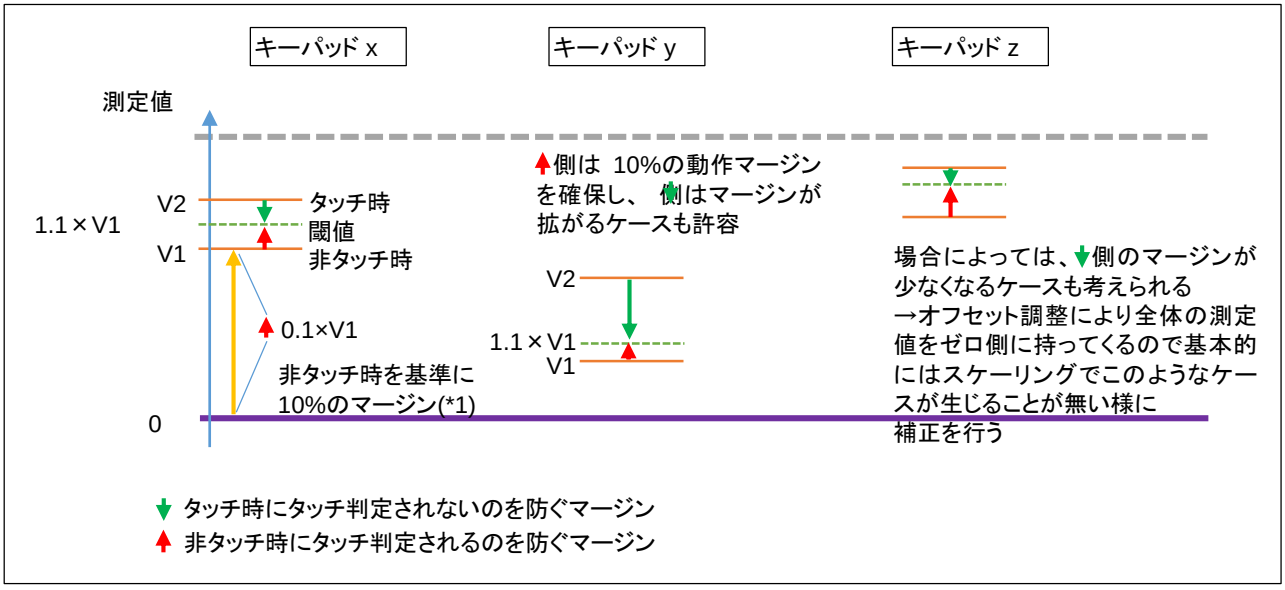
変動幅ベースで閾値を決めた場合は、マイナス側のマージンは固定されて、プラス側のマージンは変動します。（※マージンの挙動は、変動幅ベースと変動率ベースで異なります。）

閾値を+23.8%とすると、キーパッド 0 のプラス側のマージンは 5%となるので（マージン小）、閾値をもう少し小さな値（仮に+10%）とします。

・閾値を(V1+V1×10%)に設定した場合[変動率ベースでの定義]

| | スケーリング | 非タッチ時の測定値(V1) | タッチ時の測定値(V2) | 閾値 V1×1.1 | 動作マージン（幅） | **動作マージン（率）** |
|---|---|---|---|---|---|---|
| キーパッド 0 | 1.800 | 6,000 | 7,800 | 6,600 | -600<br>+1,200 | **-9.1%(*1)**<br>+18.2% |
| キーパッド 1 | 1.425 | 3,000 | 4,425 | 3,300 | -300<br>+1,125 | **-9.1%(*1)**<br>+34.1% |
| キーパッド 2 | 1.073 | 1,000 | 2,073 | 1,100 | -100<br>+1,073 | **-9.1%(*1)**<br>+88.5% |

動作マージン率のマイナス側は、タッチしていないにも拘わらずタッチしていると判定されるのを避けるためのマージンですが、タッチ判定の閾値を、「増加率」ベースで決めると、マイナス側（下図の赤↑）のマージン値は固定できるという事となります。仮に、閾値を非タッチの時の値の 10%増しの点とすると、以下の様になります。




(*1)マイナス側の動作マージンは、非タッチ時の測定値を基準とすると 10%、閾値を基準にすると、9.1%のマージン値となります

考え方として、

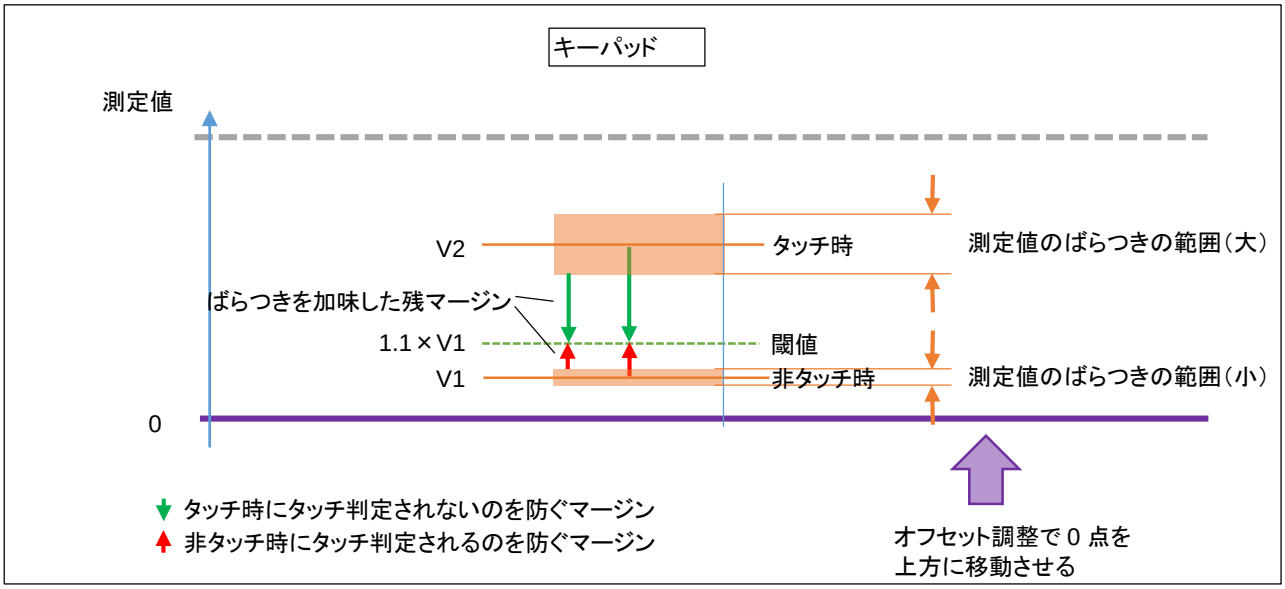
・非タッチ時の測定値は大きくばらつかない（非タッチ時にタッチ判定されるのを防ぐマージン（マイナス側、図赤↑）は固定値で良い）

→非タッチ時のばらつきは、温度変化や電圧変動などマイコン内部回路での特性変動が主要因でそれ程大きくばらつく事がないと仮定する（ばらつきの範囲は事前に想定しておく事が可能）

・タッチ時にタッチ判定されないことを防ぐマージン（プラス側、図緑↓）が拡がる（2 つのマージンのバランスが崩れる）事は問題としない

→タッチ時のばらつきは、キーのタッチの仕方や指の大きさで変わるものなので、マイナス側のマージンを最低限の値に固定し、残りのマージンをプラス側に割り振っておいた方が有利であると考える

という前提であれば、タッチ判定の閾値を「増加率」ベースで決める方が良いと考えられます。



仮に、マイナス側は容量が 5%変動する。プラス側は 40%変動するというモデルで考えてみます。

オフセット調整でスケーリングが掛かるので、オフセット調整前の測定値でまずは考えます。

・オフセット調整前の測定値

| | 非タッチ時の<br>測定値(V1) | タッチ時の<br>測定値(V2) |
|---|---|---|
| キーパッド 0 | 28,500 ~ (30,000) ~ **31,500** | ← **+600** ~ (+1,000) ~ +1,400 |
| キーパッド 1 | 14,250 ~(15,000) ~ **15,750** | ← **+600** ~ (+1,000) ~ +1,400 |
| キーパッド 2 | 4,750 ~(5,000) ~ **5,250** | ← **+600** ~ (+1,000) ~ +1,400 |

※太字部分が閾値に近づく値, ()typ 値

typ 値でオフセット調整を掛けた場合の測定値は、スケーリング値を加味すると以下の値となります。

・オフセット調整後の測定値

| | スケーリング | 非タッチ時の<br>測定値(V1) | タッチ時の<br>測定値(V2) |
|---|---|---|---|
| キーパッド 0 | 1.800 | 5,460 ~ (6,000) ~ **6,540** | ← **+1,080** ~ (+1,800) ~ +2,520 |
| キーパッド 1 | 1.425 | 2,786 ~(3,000) ~ **3,214** | ← **+855** ~ (+1,425) ~ +1,995 |
| キーパッド 2 | 1.073 | 946 ~(1,000) ~ **1,054** | ← **+644** ~ (+1,073) ~ +1,502 |

閾値に近づく方の値で動作マージンを考えてみます。

・閾値をキーパッド 1 の変動幅(4425-3000=1425)の中間(V1+713)に設定した場合[変動幅ベースでの定義]

| | スケーリング | 非タッチ時の<br>測定値(V1) | タッチ時の(*1)<br>測定値(V2) | **閾値(*2)<br>V1+713** | 動作マージン<br>（幅） | 動作マージン<br>（率） |
|---|---|---|---|---|---|---|
| キーパッド 0 | 1.800 | 6,540 | 7,080 | **6,713** | -173<br>+367 | -2.6%<br>+5.5% |
| キーパッド 1 | 1.425 | 3,214 | 3,855 | **3,713** | -499<br>+142 | -13.4%<br>+3.8% |
| キーパッド 2 | 1.073 | 1,054 | 1,644 | **1,713** | -659<br>**+Δ69(*3)** | -38.5%<br>**+ Δ4.0%(*3)** |

・閾値を(V1+V1×10%)に設定した場合[変動率ベースでの定義]

| | スケーリング | 非タッチ時の<br>測定値(V1) | タッチ時の<br>測定値(V2) | **閾値(*2)<br>V1×1.1** | 動作マージン<br>（幅） | 動作マージン<br>（率） |
|---|---|---|---|---|---|---|
| キーパッド 0 | 1.800 | 6,540 | 7,080 | **6,600** | -60<br>+480 | -0.9%(*4)<br>+7.3% |
| キーパッド 1 | 1.425 | 3,214 | 3,855 | **3,300** | -86<br>+555 | -2.6%<br>+16.8% |
| キーパッド 2 | 1.073 | 1,054 | 1,644 | **1,100** | -46<br>+544 | -4.2%<br>+49.5% |

(*1)typ 値+タッチ時の差分の min 側

(*2)typ ベースで前出で設定した値

上記モデルでは、変動幅ベースの場合キーパッド 2 がタッチした時に応答しない(*3)という計算結果となります。

また、スケーリングが効いた場合（オフセット値が大きい場合）の非タッチ時の動作マージンが微小(*4)です。

計算では、9.1%のマージンを取っているのですが、5%変動×スケーリング 1.8 では変動率は 9%となり、ほとんどマージンが残っていません。（一律 10%とするのではなく、オフセット値により、閾値の掛け率を調整するという手法も考えられるかと思います。）

なお、閾値の決め方に関して考察を行ってきましたが、「非タッチ時の測定値」と「タッチ時の測定値」の両方判っている場合は比較的簡単に閾値を決められると考えます。（一番単純なのは、タッチ時と非タッチ時の中心に閾値を設定する事です。）一般的なタッチキーのアプリケーションでは、起動後初期化時に、タッチ時の測定値が判らないとうケースが多いということも想定しています。
（※起動後に、利用者にキーにタッチすることを要求。タッチ時の測定値を取得した後で、タッチキーのシステムが利用可能になるというシステムはあまり考えにくいです。）

・閾値の決定時

| | スケーリング (*1) | 非タッチ時の測定値(V1) | タッチ時の測定値(V2) | 閾値[変動率] V1×1.1 | 閾値[変動幅] |
|---|---|---|---|---|---|
| キーパッド 0 | 1.800 | 6,000 | 不明 | 6,600（値決めが容易） | ?（値を決めづらい） |
| キーパッド 1 | 1.425 | 3,000 | 不明 | 3,300（値決めが容易） | ?（値を決めづらい） |
| キーパッド 2 | 1.073 | 1,000 | 不明 | 1,100（値決めが容易） | ?（値を決めづらい） |

(*1)スケーリング値は、オフセット調整を行えば値が判ります

非タッチ時の測定値は、オフセット調整の後測定を行えば判ります。上記の値だけしか判っていない場合、変動率ベースの閾値は簡単に決められますが、変動幅ベースの閾値を決めるのは比較的難しいと考えます。

閾値をどう決めるかは、色々な考え方や前提条件があると思いますので、必ずしも方式的にこの手法が良いという訳ではありませんが、一つの考え方として、本サンプルプログラムでは「増加率」ベースの閾値方式を採用します。

本サンプルプログラムでは、デフォルトの閾値を

ctsu2/ctsu2.h

```
#define CTSU_DEFAULT_THRESHOLD (1.1f)//判定閾値：初期値に対し 10%増加を閾値とする
```

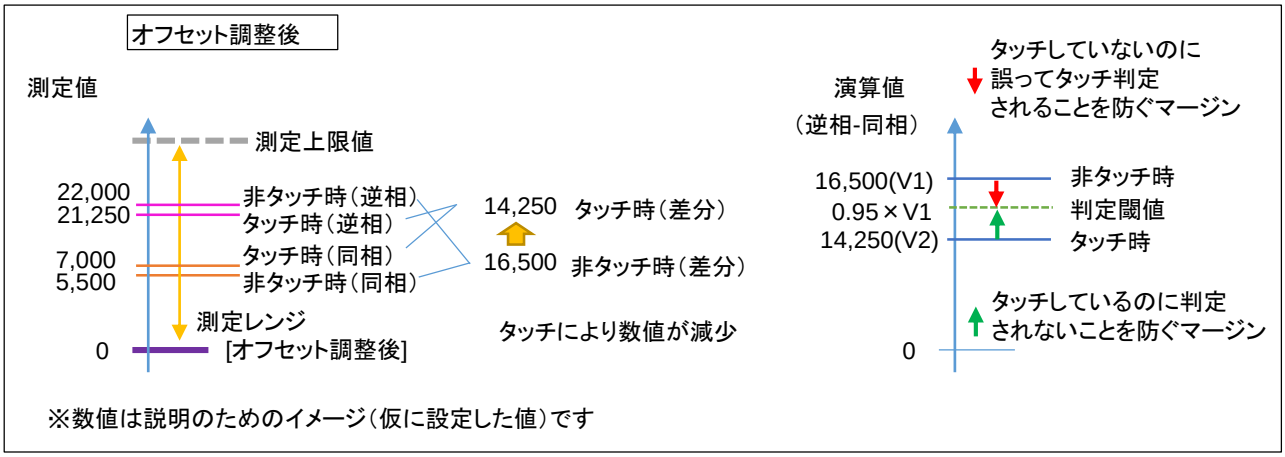
増加率 10%で定義しています。

マイナス側（非タッチ時にタッチされていると誤判定される）側のマージンは、非タッチの測定値(V1)基準だと 10%です。（閾値を基準にすると、V1 に対するマージンは、0.1/1.1=9.09%です。）

・相互容量

相互容量においても、自己容量同様に「変動幅」「変動幅」のどちらで判断するかという話となります（もしくはそれ以外ということも当然考えられます）。

本サンプルプログラムでは演算値が、5%以上減少した際、タッチしていると判断します。（「変動率」での判定）




考え方としては自己容量と同じです。

・逆相－同相の演算値で非タッチ時の値の 95%の値を閾値とする
・タッチしていないのに誤ってタッチ判定されるマージン（赤↓）側は変動率の値が固定
・タッチしているのにタッチ判定されない事を防止するマージン（緑↑）は、V1, V2 の値により変動

自己容量では、値が増加方向に変化しましたが、相互容量では値が減少方向に変化します。上下の関係が逆になるだけで、基本的な考え方は同じです。

ctsu2/ctsu2.h

```
#define CTSU_DEFAULT_THRESHOLD (0.95f)//判定閾値：初期値に対し 5%減少を閾値とする
```

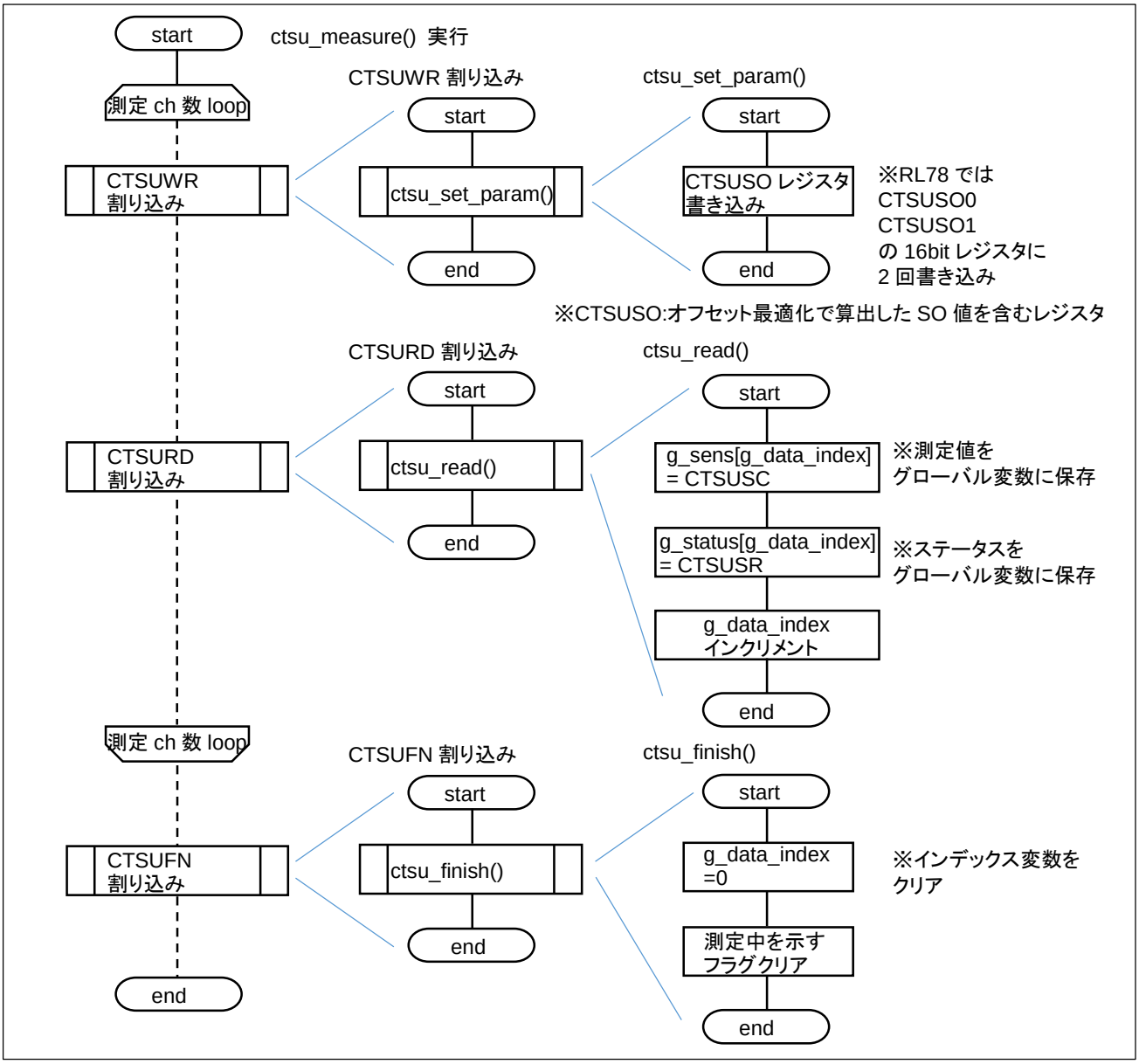
で、定義していますので、他の値に変える事は可能です。

### 9.2.3. 計測開始＋割り込み処理

計測を開始する ctsu_measure() 関数を実行すると、割り込み処理により、計測が実行されます。

CTSU2 の割り込み処理は、ctsu2/ctsu2_common.c 内で定義されています。

－ctsu_measure() 実行時 CTSU2 割り込み フローチャート－




CTSUWR, CTSURD, CTSUFN は割り込み処理なので、割り込み条件が整った時点で独立して呼び出されるものですが、測定開始指示後

・CTSUWR 割り込み

→1 つ目の測定 ch の向けに算出したオフセット値を CTSUSO レジスタに書き込む事で実際の測定開始

・CTSURD 割り込み

→1 つ目の測定 ch の測定が完了したので、測定値を読み出す

・CTSUWR 割り込み
→2 つ目の測定 ch の向けに算出したオフセット値を CTSUSO レジスタに書き込む事で実際の測定開始
・CTSURD 割り込み
→2 つ目の測定 ch の測定が完了したので、測定値を読み出す
…測定 ch 数繰り返す
・CTSUFN 割り込み
→指定した全ての計測 ch の測定が終了した事を示す

という流れで割り込み関数が呼び出されます。

測定 ch 数は、自己容量ではキーパッド数（RX, RL78 では 16 キー、RA では有効なのは 0~9 の 10 キー）の 16ch/10ch。

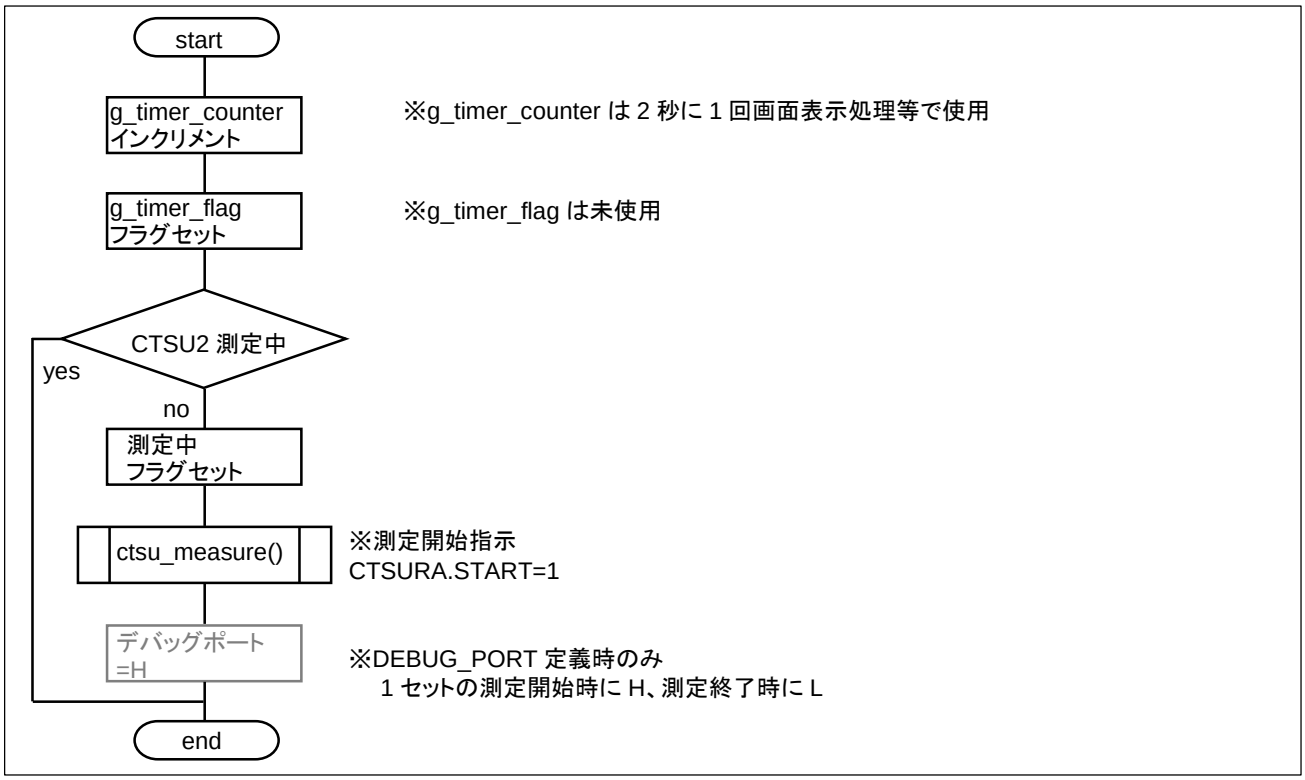
相互容量では、0~24 の 25 キーの同相、逆相測定で、50ch です。相互容量の場合、
・1 つ目の ch の同相
・1 つ目の ch の逆相
・2 つ目の ch の同相
・2 つ目の ch の逆相
…
の順に測定されます。

## 9.3. タイマ処理

タイマ周期毎に、測定開始指示(ctsu_measure() 関数）が実行されます。サンプルプログラムでは、50ms(20Hz)周期に設定してありますが、アプリケーションに応じて変更しても問題ありません。

timer/timer.c 内

ータイマ処理 periodic_process() フローチャートー



# 10.サンプルプログラムで使用してる定数・変数・関数

## 10.1.定数

ctsu2/ctsu2.h　内で定義されている定数。

```
#define CTSU_TYPE CTSU_SELF_CAP_S16A   //自己容量キーパッド(S16A)使用時
#define CTSU_TYPE CTSU_SELF_CAP_D55A   //相互容量キーパッド(D55A)使用時
```

自己容量タイプか相互容量タイプのどちらのキーパッド、プログラムを有効化するかを決める定数です。どちらか一方の行を有効化して、もう一方はコメントアウトしてください。

```
//#define CTSU_VOLTAGE_UNDER4_5  //電源電圧 VCC/VDD=4.5V未満の時に定義を有効にする
```

電源電圧を4.5V未満で使用する場合は、先頭のコメントアウトを外してください。(コメントアウトを外すと、CTSU2のPUMPON=1に設定され、昇圧回路が有効化されます。)

```
//#define CTSU_VOLTAGE_UNDER2_4  //電源電圧 VCC/VDD=2.4V未満の時に定義を有効にする
```

電源電圧を2.4V未満で使用する場合は、先頭のコメントアウトを外してください。(CTSU_VOLTAGE_UNDER4_5も有効化してください。)(コメントアウトを外すと、CTSU2のATUNE0=1に設定され計測電圧が1.2Vに設定されます。)

・自己容量の場合

```
#define CTSU_VALID_TERMINALS (16)
```

自己容量で使用する端子数です。使用するキーパッド数と同じとなります。RX, RL78では16(キー0~F)。RAでは、10(キー0~9)が有効(ハード的につながっています)です。RA(RA2L1)のキットでは10を指定してください。

```
#define CTSU_CH_NUM CTSU_VALID_TERMINALS
```

CTSU_CH_NUM　は測定ch数を定義します。自己容量の場合、キーパッド数=測定ch数です。

```
#define CTSU_DEFAULT_THRESHOLD (1.1f)
```

自己容量のタッチ判定閾値です。非タッチ時の測定値に対する割合で定義します。1.1を指定した場合は、非タッチ時の測定値が1000の時、判定閾値は非タッチ時の測定値×CTSU_DEFAULT_THRESHOLD(1.1)=1100に設定されます。(測定値、1100以上の場合、タッチしていると判定されます。)

・相互容量の場合

```
#define CTSU_VALID_TERMINALS (10)
```

相互容量で使用する端子数です。RX5本、TX5本の10端子となります。

```
#define CTSU_CH_NUM (50)
```

測定ch数です。5x5の25キーに対して、同相・逆相測定の50chです

```
#define CTSU_KEY_NUM (25)
```

相互容量タイプキーパッド(D55A)のキーパッド数です、5x5の0~24の25個のキーパッドで構成されています。

```
#define CTSU_TX_TERMINALS (5)
#define CTSU_RX_TERMINALS (5)
```

送信端子(TX)と受信端子(RX)の数です。どちらも5端子(5x5)です。

```
#define CTSU_COLUMN_NUM (5)
#define CTSU_ROW_NUM (5)
```

キーマトリックスの行数(COLUMN)と列数(ROW)です、どちらも5です。

```
#define CTSU_DEFAULT_THRESHOLD (0.95f)
```

相互容量のタッチ判定閾値です。非タッチ時の測定値に対する割合で定義します。0.95を指定した場合は、非タッチ時の測定値が1000の時、判定閾値は非タッチ時の測定値×CTSU_DEFAULT_THRESHOLD(0.95)=950に設定されます。(測定値、950以下の場合、タッチしていると判定されます。)

・自己容量、相互容量で共通で使用する定数

```
#define CTSU_OFFSET_MERGIN (0.2f)
```

オフセット調整の時のマージン設定値です。オフセットゼロ(SO=0)の時の測定値の0.2(20%)の点にオフセット(測定値=0)を設定します。値を小さくするとマージンが減少しますが、測定のスケーリングが拡大します(測定レンジを有効に使えます)。値を大きくすると、動作マージンを大きく取ります(特性変動しても、測定値が測定レンジから外れる事がなくなります)。

```
#define CTSU_CTSUSO_SO (140UL)
//#define CTSU_OFFSET_OPTIMIZE_OMIT
```

オフセット最適化を制御する定数です。CTSU_OFFSET_OPTIMIZE_OMITを定義した場合、オフセット最適化は省略され、オフセット値として一律CTSU_CTSUSO_SOの値が使われます。CTSU_OFFSET_OPTIMIZE_OMIT未定義の場合、CTSU_CTSUSO_SOは使われません。デフォルトはオフセット最適化ONです。

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

（mコマンド）での測定値の表示更新間隔設定値。デフォルトでは、2秒間隔で表示更新。

```
#define CTSU_SENS_INDEX_MAX  (100)   //5秒間（データサンプリング周期20Hz×5秒=100個）
```

（sコマンド）での統計データ取得個数。（データはstatic変数に格納されます、メモリの空き容量を超えない範囲で設定してください。）

```
#define CTSU_KEYPAD_ORDER
```

定義時、自己容量タイプで、画面表示がキーパッド順(0~9,A~F)となります。コメントアウトすると、測定ch順（TSxxの若い順）になります。

```
#define DEBUG_PRINT
```

定義時、SCU(UART)デバッグ表示を行う様になります。コメントアウトすると、オフセット最適化時の進捗状況等が表示されなくなります。

```
#define DEBUG_PORT
```

定義時、汎用ポートで1回の測定に掛かる時間を観測できる様になります。1セット（自己容量タイプ16または10、相互容量タイプ50回）の測定前にデバッグポートがHとなり、測定終了でLになります。

－デバッグ用端子－

| マイコンボード | ポート名 | 基板スルーホール |
|---|---|---|
| HSBRX140F80 | PE3 | J2-1 |
| HSBRL78G23-100 | P147 | J2-2 |
| HSBRA2L1F100 | P202 | J1-1 |

## 10.2.変数（グローバル変数）

ctsu2/ctsu_self_cap.c
ctsu2/ctsu_mutual_cap.c
内で定義されているグローバル変数。

```
volatile unsigned short g_in_measure
```

現在の測定状態を示す変数です

測定を行っていない時 ： CTSU_STATE_IDLE (0)
測定中 ： CTSU_STATE_IN_MEASURE (1)
測定終了 ： CTSU_STATE_FINISHED (2)

が格納されます。（測定終了後、データの回収等を行った後、CTSU_STATE_IDLE に移行させる。）

※割り込み関数内で更新される変数なので、volatile 指定

```
unsigned short g_initialize_finished
```

初期化処理（オフセット最適化、非タッチ時の初期値の取得）状態を示す変数です。

初期化完了前 ： CTSU_INITIAL_UNFINISHED (0)
初期化完了 ： CTSU_INITIAL_FINISHED (1)

となります。

```
unsigned char g_key
```

いま、タッチしているキーパッドを示す変数です。自己容量の場合は、0~15（キーパッド0~Fに対応）。相互容量の場合は0~24となります。複数のキーパッドをタッチしているときは、一番大きい測定値となっているキーパッド。タッチしていない時は、0xFFとなります。

```
unsigned char g_key_state[CTSU_KEY_NUM]
```

いま、タッチしているキーパッドを示す変数です。キーパッド数の配列となっており、タッチしていないキーパッドは0。タッチしているキーパッドは1が格納されます。複数のキーパッドにタッチしているときのタッチ、非タッチの判定に使用できます。[配列数: 自己容量16(RA:10)、相互容量25]

`volatile unsigned short g_sens[CTSU_CH_NUM]`

ch毎の測定値を保存する変数です。測定ch数の配列となります。[配列数:  自己容量16(RA:10)、相互容量50]

`long g_initial_sens[CTSU_CH_NUM]`

初期（起動時、非タッチ時）のch毎の測定値を保存している変数です。256回測定した平均値が格納されます。（256回の平均を取る際、測定値の加算を累積し最後に256で除算します。計算中は、測定値の最大値65535を超えた値となっていますが、初期化(ctsu_init)が終わった後は、0~65535の値を取ります。）[配列数:  自己容量16(RA:10)、相互容量50]

`long g_initial_diff[CTSU_KEY_NUM]`

初期（起動時、非タッチ時）のキーパッド毎、逆相－同相の差分値です。（基本的には、逆相の方が大きな測定値になりますので、結果は負数にはならないのですが、unsigned short同士の引き算の結果を格納する変数なので、long型としています。）[※相互容量のみ] [配列数:  相互容量25]

`long g_diff[CTSU_KEY_NUM]`

逆相－同相の差分値を保存する配列変数（配列数25）です。[※相互容量のみ] [配列数:  相互容量25]

`volatile unsigned long g_status[CTSU_CH_NUM]`

測定時の、ステータス(CTSUS)を保存しておく変数です。プログラム上は未使用です。[配列数:  自己容量16(RA:10)、相互容量50]

`unsigned long g_ctsu_ctsuso[CTSU_CH_NUM]`

ch毎の最適化されたオフセット値(SO)を含む、CTSUSOレジスタの値です。1ch測定毎に、この変数の値をCTSUSOレジスタに書き込みながら測定します。[配列数:  自己容量16(RA:10)、相互容量50]

※RL78では、CTSUSOレジスタはCTSUSO0, CTSUSO1の16bitレジスタ2本で構成されています。RX, RAでは32bitレジスタです。本変数は、32bitのunsignd long型としており、RX, RAの場合はそのままCTSUSOレジスタに書き込み。RL78では、上位・下位に分割し、CTSUSO0, CTSUSO1レジスタに書き込みを行っています。

```
volatile unsigned short g_data_index
```

配列変数の現在のインデックスを示す変数です。1chの測定終了時にインクリメントし、1セット（全ch）測定終了時に0にしています。

```
unsigned short g_touch_threshold[CTSU_KEY_NUM]
```

キーパッド毎の、タッチの閾値が格納される変数です。

自己容量では、全てのキーパッドに対し初期値983（測定レンジ65536×1.5%）。相互容量では、1310（測定レンジ65536×2%）が設定されます。相互容量では、閾値は負数となりますが、本変数は絶対値での表現となります。キーパッド毎の閾値の最適化（tコマンド）を行うと、キーパッド毎に別な値で更新されます。（pコマンドで現在の閾値を表示可能です。）[配列数:  自己容量16(RA:10)、相互容量25]

```
unsigned char g_ctsu_s16a_ch_vs_ts[CTSU_KEY_NUM]
unsigned short g_ctsu_s16a_keypad_vs_ch[CTSU_KEY_NUM]
unsigned short g_ctsu_s16a_ch_vs_keypad[CTSU_KEY_NUM]
```

自己容量タイプキーパッド(S16A)で
ch 番号－TS 番号
キーパッド番号－ch 番号
ch 番号－キーパッド番号

の対応を示す変数です。

ch番号は、0から測定数(RX, RL78: 0~15, RA, 0~9)で、キーパッド番号は0~15(0~F)、TS番号はキーパッドにつながっている、TS端子の番号で、使用マイコンボードによって変わります。[自己容量のみ] [配列数:  自己容量16(RA:10)]

```
char g_ctsu_s16a_keypad_str[CTSU_KEY_NUM][8]
```

自己容量タイプキーパッドで、画面表示用に

0(TS00)

の様な、"キーパッド名(TSxx)"の文字列を格納する変数です。[自己容量のみ] [配列数:  自己容量16(RA:10)]

```
unsigned char g_ctsu_d55a_key_table[CTSU_KEY_NUM];
```

キーパッド番号(0~24)と測定chの対応テーブル。[相互容量のみ] [配列数: 25]

```
unsigned char g_ctsu_d55a_row_index[CTSU_ROW_NUM];
unsigned char g_ctsu_d55a_column_index[CTSU_COLUMN_NUM];
```

キーパッドの行、列とTXxxの対応テーブル。[相互容量のみ] [配列数: それぞれ5]

ctsu2/ctsu_pin_def.c
内で定義されているグローバル変数。

```
const unsigned char g_ctsu_s16a_keypad_table_char[];
```

自己容量キーパッドの0~Fのキャラクタ(文字)を定義している変数。

例:
g_ctsu_s16a_keypad_table_char[3] → "3";(文字列に変換)

```
const unsigned char g_ctsu_s16a_keypad_vs_ts[];
```

自己容量キーパッドのキーパッド番号(0~15)とTSxxの関係表。

例:
g_ctsu_s16a_keypad_vs_ts[] = {TS10, TS03, TS08, ....

キーパッド0にTS10が接続
キーパッド1にTS03が接続
キーパッド2に、TS08が接続

キーパッドとTSxxの接続関係を定義します。

```
const unsigned char g_ctsu_d55a_tx_terminal[];
```

相互容量キーパッドのTX側端子とTSxxの関係表。

例:

一番下の行 .....一番上の行

TX-0　TX-1　TX-2　TX-3　TX4

g_ctsu_d55a_tx_terminal[] = {TS35, TS34, TS33, TS32, TS31};

相互容量キーパッド(D55A)のTXは横方向のライン（基板表面に配線）で、一番下の行につながっている端子がTS35、一番上の行につながっている端子がTS31のとき、上記の様に定義する。

```
const unsigned char g_ctsu_d55a_rx_terminal[];
```

相互容量キーパッドのRX側端子とTSxxの関係表。

例:

一番左の列 .....一番右の列

RX-0　RX-1　RX-2　RX-3　RX4

g_ctsu_d55a_rx_terminal[] = {TS30, TS25, TS24, TS23, TS22};

相互容量キーパッド(D55A)のRXは縦方向のライン（基板裏面に配線）で、一番左の列につながっている端子がTS30、一番右の列につながっている端子がTS22のとき、上記の様に定義する。

－測定 ch と TS、キーパッドの関係－

マイコンには、タッチキーの測定対象端子 TSxx（xx は 0~35 等）が予め決められています。TSxx とタッチキーパッドは、可能な限り配線がクロスしない様に、配線が短くなる様に接続するのが望ましいです。そのため、キーパッド番号と接続される TSxx の関係はある程度ランダムになる事が考えられます。

・自己容量

TSxx とどのパッドが接続されているかは、ctsu2/ctsu_pin_def.c 内の

g_ctsu_s16a_keypad_vs_ts[] = {TS10, TS03, TS08, TS12, TS13, TS20, TS07, TS18, TS05, TS01}; //一例

で定義しています。（g_ctsu_s16a_keypad_vs_ts はユーザが定義する）

仮に、0~9 のキーを上記の様に定義した場合、自己容量マルチスキャンモードで測定すると、1 回の測定指示(ctsu_measure()）で、全 ch が測定されます。測定順は、TSxx の xx の若い番号順です。

| 測定 ch | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|---|
| TSxx | TS01 | TS03 | TS05 | TS07 | TS08 | TS10 | TS12 | TS13 | TS18 | TS20 |
| キーパッド番号 | 9 | 1 | 8 | 6 | 2 | 0 | 3 | 4 | 7 | 5 |

測定は、測定 ch 順、上表の左から順に実行されます。例えば、TS07, キーパッド 6 は、4 回目の測定となります。

プログラム上の基準は、あくまで使用している（TSxx 端子に TS の機能を割り当ててる）端子の番号の若い順です。但し、チューニング時等、キーパッド順に処理したいというケースがありますので、どのキーパッドと測定 ch、TSxx が紐づいているかのテーブルを作って適宜処理を行っています。

テーブルに相当するのが、以下の配列変数です。

g_ctsu_s16a_ch_vs_ts[] = {TS01, TS03, TS05,....} 測定ch ー TS番号の関係
g_ctsu_s16a_keypad_vs_ch[] = {5, 1, 4, ...} キーパッド ー 測定chの関係
g_ctsu_s16a_ch_vs_keypad[] = {9, 1, 8, ...} 測定 ch ー キーパッドの関係

ctsu_init()内でこれらのテーブルを作成しています。

・相互容量

TSxx とどのパッドが接続されているかは、ctsu2/ctsu_pin_def.c 内の

g_ctsu_d55a_tx_terminal[] = {TS35, TS34, TS33, TS32, **TS31**};
g_ctsu_d55a_rx_terminal[] = {TS30, TS25, TS24, TS23, **TS22**};

で定義しています。（g_ctsu_d55a_tx_terminal, g_ctsu_d55a_rx_terminal はユーザが定義する）

キーパッドの TX（D55A では 5 行）, RX（D55A 基板では 5 列）のマトリックスを構成する信号にどの TS がつながっているかを記載します。

キーパッドが 25 個ある場合、同相、逆相の測定で、測定 ch 数は 50 となります。

| 測定 ch | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|---|
| RX:TSxx | TS22 | TS22 | TS22 | TS22 | TS22 | TS22 | TS22 | TS22 | TS22 | TS22 |
| TX:TSxx | TS31 | TS31 | TS32 | TS32 | TS33 | TS33 | TS34 | TS34 | TS35 | TS35 |
| | 同相 | 逆相 | 同相 | 逆相 | 同相 | 逆相 | 同相 | 逆相 | 同相 | 逆相 |
| キーパッド番号 | 24 | 24 | 23 | 23 | 22 | 22 | 21 | 21 | 20 | 20 |

| 測定 ch | 10 | 11 | 12 | 13 |
|---|---|---|---|---|
| RX:TSxx | TS23 | TS23 | TS23 | TS23 |
| TX:TSxx | TS31 | TS31 | TS32 | TS32 |
| | 同相 | 逆相 | 同相 | 逆相 |
| キーパッド番号 | 19 | 19 | 18 | 18 |

---

(1)RX の TSxx の一番若い番号（太字）－TX の TSxx の若い番号（太字） → 一番の大きな番号（RX 固定で TX をスキャンしていく、測定 ch0~9）
(2)RX の次の番号－TX の若い番号→一番大きな番号（測定 ch10~19）
…
(5)RX の最後の番号－TX の若い番号→一番大きな番号（測定 ch40~49）

の様に、TS 番号の小さい順に測定していきます。

測定 ch とキーパッド番号の対応表は、

g_ctsu_d55a_key_table[] = {24, 23, 22, .....}

キーパッド番号を測定ch順に並べたもの、として作成して使用しています。

g_ctsu_d55a_row_index, g_ctsu_d55a_column_index は、RX, TX の若い順に並べた場合の対応表です。

g_ctsu_d55a_row_index[] = {**4, 3, 2, 1 ,0**}

RX-0　RX-1　RX-2　RX-3　RX-4
g_ctsu_d55a_rx_terminal[] = { TS30, TS25, TS24, TS23, TS22};
TSの小さな順に並び変え　　TS22(RX-**4**)　TS23(RX-**3**)　TS24(RX-**2**)　TS25(RX-**1**)　TS30(RX-**0**)

RX の 4-3-2-1-0 の順で並んでいるので、g_ctsu_d55a_row_index は、4,3,2,1,0 の配列となります。

多少複雑ですが、
・測定は TSxx の若い番号から順に行う
・RX 固定で TX を（TSxx を増やしながら）スキャン
・RX を増やして同様
の順に測定されるので、測定 ch とキーパッドの対応は重要となります。（今見ている測定値が、どのキーパッドの測定値かが判るようにする必要があります。）

## 10.3.関数

ctsu2/ctsu_self_cap.c
ctsu2/ctsu_mutual_cap.c
内で定義されている関数。（自己容量と相互容量で同じ関数名の関数は役割が同一です）

### ctsu_init

概要：タッチキーユニットの初期化関数
宣言：
　　void ctsu_init(void)
説明：
　・CTSU2 の初期化・設定
　・TSxx 端子設定
　・CTSU2 割り込み設定
を行います

引数：
　　なし
戻り値：
　　なし

### ctsu_offset_optimize

概要：オフセット最適化関数
宣言：
　　void ctsu_offset_optimize(void)
説明：
　・オフセット最適化
を行います
引数：
　　なし
戻り値：
　　なし
補足：
　　g_ctsu_ctsuso[]に、最適化結果が入ります
　　本関数を実行しない場合は、ctsu2.h 内で指定している初期値(SO=140)が適用されます
　　（本関数の実行を省略してもタッチキーの測定自体は可能です）
　　本関数実行中はタッチキーパッドには触れないでください

## ctsu_initial_value

概要:初期値(非タッチ時の測定値)取得関数
宣言:
　　void ctsu_initial_value(void)
説明:
　・非タッチ時の測定値の取得
を行います
引数:
　　なし
戻り値:
　　なし
補足:
　　g_initlal_sens[]に、測定 ch 毎の非タッチ時の測定値
　　測定値は、256 回計測した際の平均値です
　　本関数実行中はタッチキーパッドには触れないでください

## ctsu_result

概要:タッチ判定関数
宣言:
　　unsigned char ctsu_result(unsigned char *key_state)
説明:
　・タッチ判定
を行います
引数:
　　unsigned char *key_state:　キーパッド数の配列　非タッチ時:0,　タッチ時 1 が格納される
戻り値:
　　unsigned char:　一番非タッチ時から変動が大きなキーパッド番号
　　　　タッチしているキーがない場合は、0xff

# 11.付録

## 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 | 2022.1.26 | — | 初版発行 |
| REV.1.0.1.0 | 2023.7.28 | | RX140 タッチキー評価キットの販売開始に合わせて内容を見直し |
| REV.1.0.2.0 | 2023.9.26 | | RL78/G23 タッチキー評価キット向けに追記 |

## 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:https://www.hokutodenshi.co.jp

ルネサス エレクトロニクス RX/RL78/RA グループマイコン搭載
HSB シリーズマイコンボード向けキット

# RX140 タッチキー評価キット
# RL78/G23 タッチキー評価キット
# RA2L1 タッチキー評価キット
## [ソフトウェア編] マニュアル

株式会社 北斗電子
 2023 年 9 月 26 日改訂 REV.1.0.2.0 (230926)