SmartRA 学習キット
チュートリアル 8

ルネサス エレクトロニクス社　RA マイコン搭載
HSB シリーズマイコンボード　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.1.0.0.0

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。

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、アクセスランプの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。

製品の故障や、データ消失の原因となります。

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。

1. RA2L1_TOUCHKEY_QE

本チュートリアルでは、RA2L1 の持つタッチキー機能(CTSU2)に関して触れます。

CTSU は、Capacitive Sensing Unit の略で静電容量を使用して、人の指でタッチしたかどうかを識別する機能です。非接触スイッチ等で使用されています。

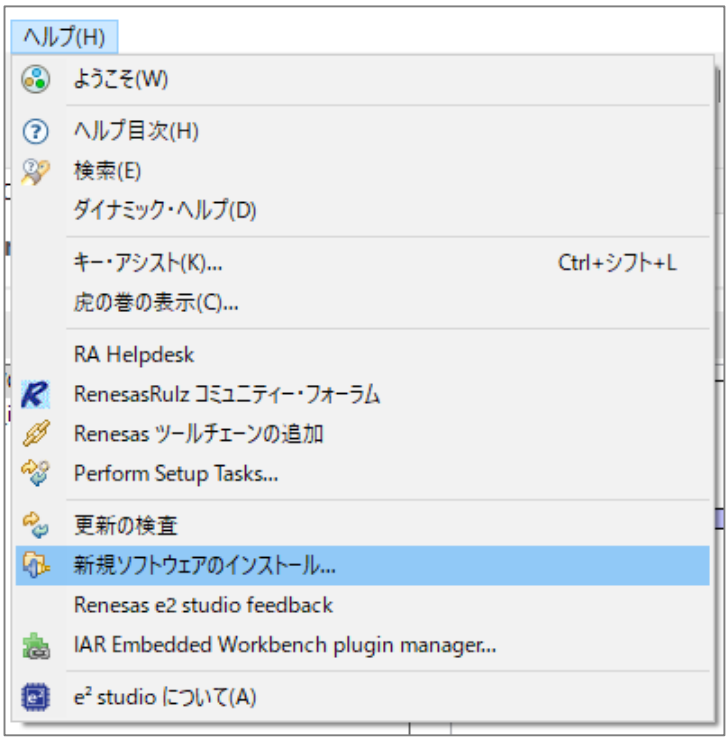
ルネサスの CTSU は、R8C/33T でマイコンの機能として実装され、RX113/RX130/RX231 で 2 世代目に進化しています。RA シリーズのマイコンは基本的には、CTSU を持っているのですが、RA2L1 ではさらに進化した CTSU2 が搭載されています。

本キットのベースボードには、タッチキーパッドを 3 つ持たせていますので、このパッドを用いたチュートリアルとなります。

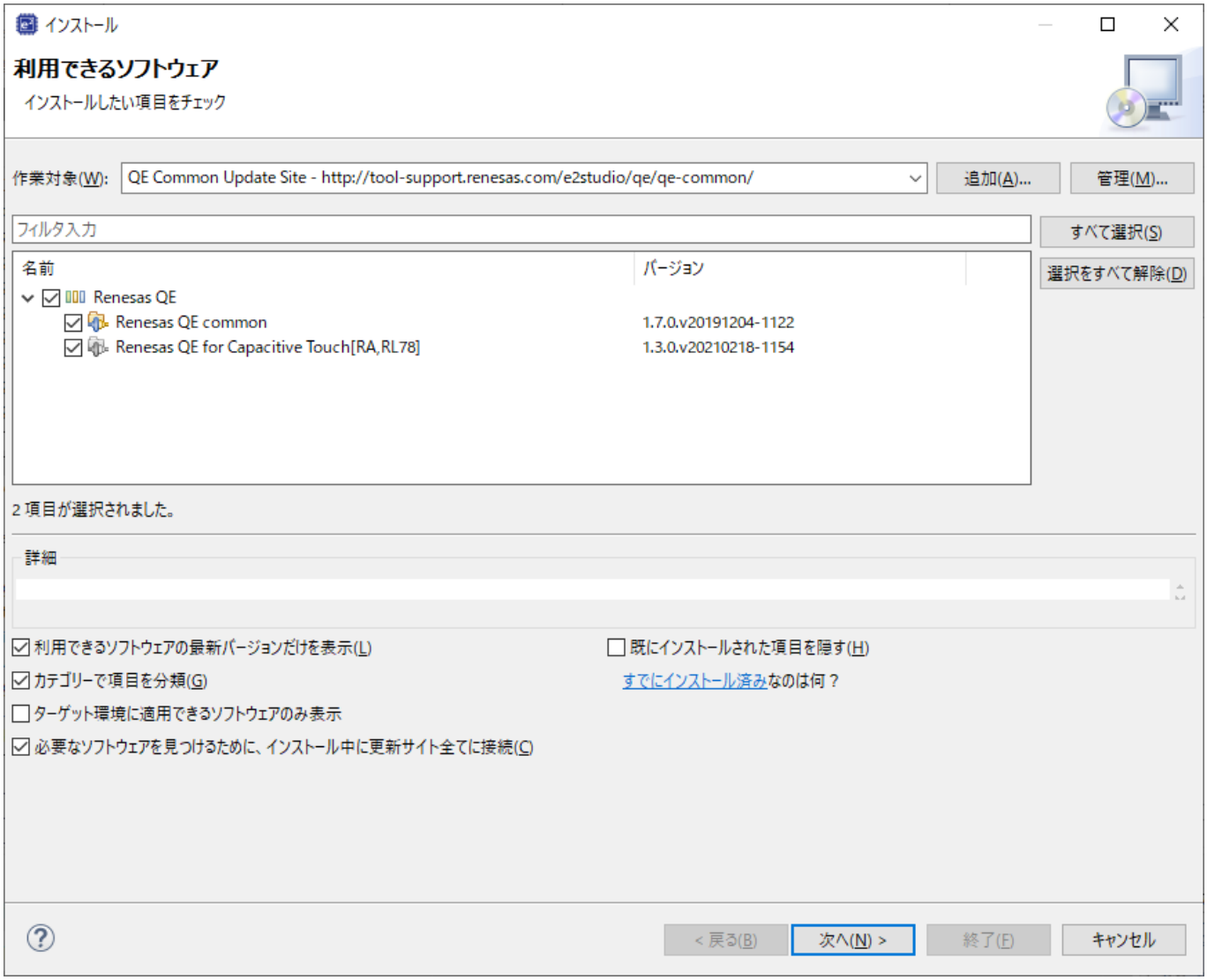
1.1. QE ツールのインストール

タッチキーは、Renesas QE Cap-touch でサポートされる機能となりますので、e2studio に、Renesas QE Cap-touch をインストールしてください。

※[重要] Renesas QE Cap-touch を使って PC からタッチキー動作を見る場合は、エミュレータ（ルネサス E2Lite もしくは E2）が必要です （エミュレータがない場合は、1.5.5 に進み、タッチ動作を確認することができます）



ヘルプ－新規ソフトウェアのインストールから、



QE Common Update Site を選択して、

Renesas QE common

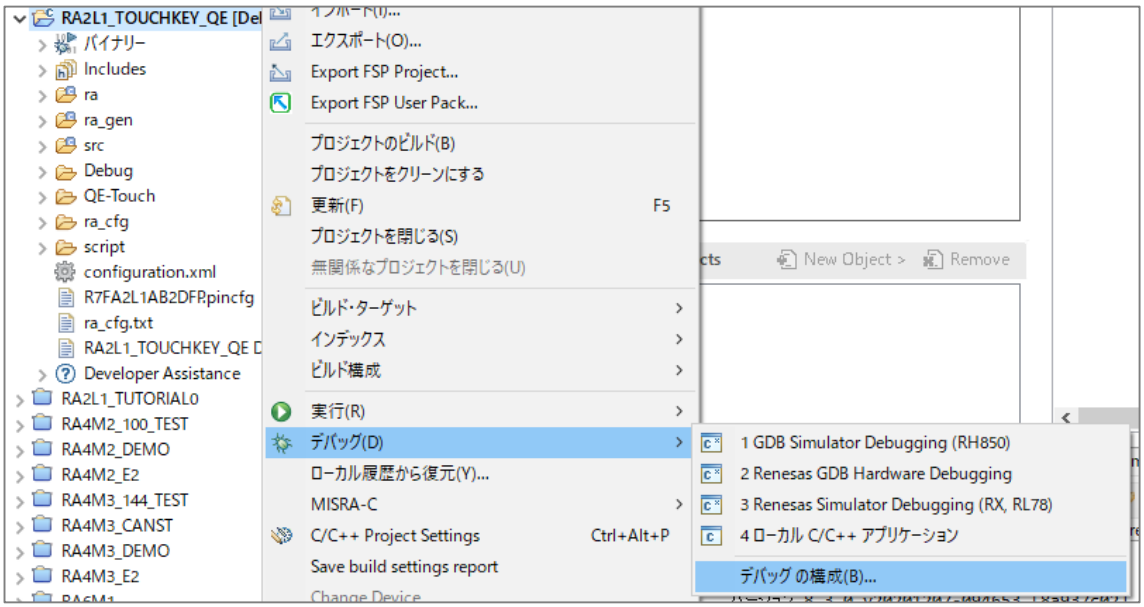
Resnesas QE for Capacitive Touch

をインストールしてください。

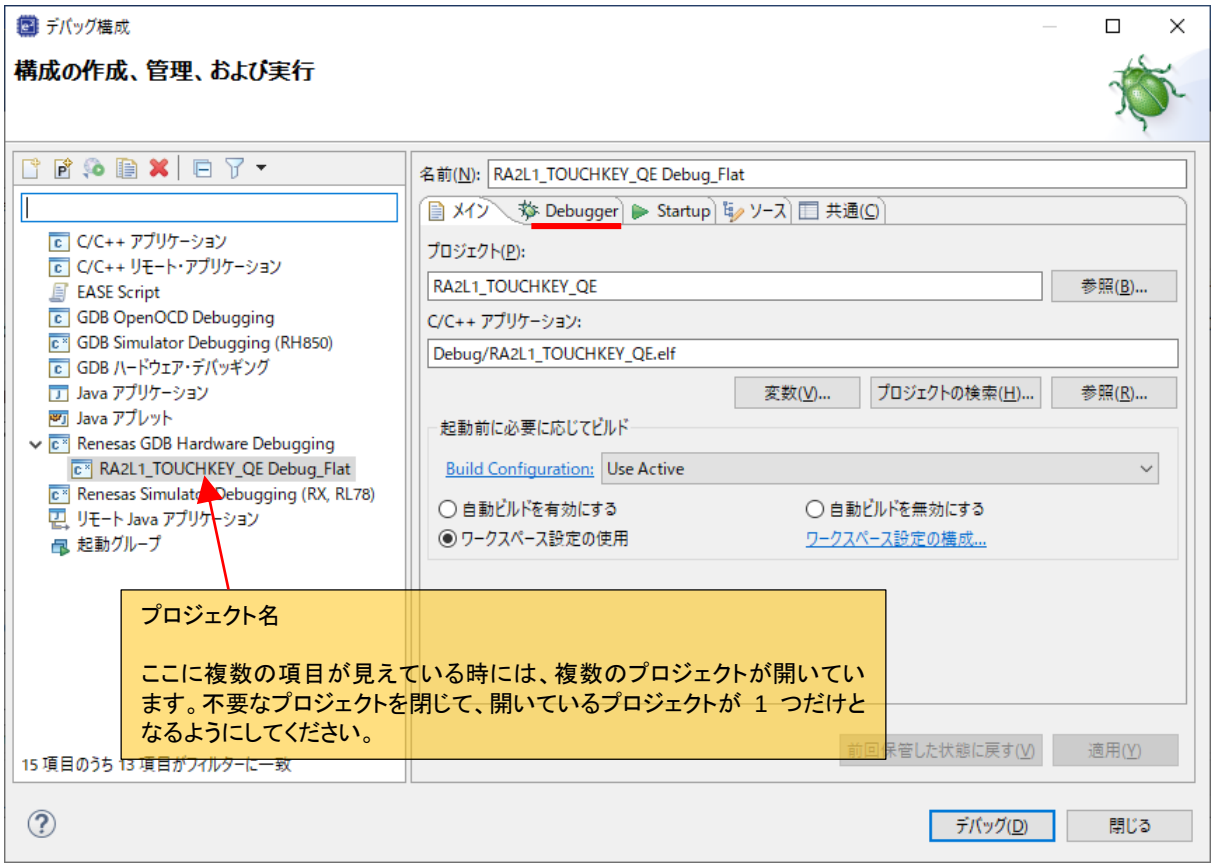
（インターネットに直接接続していない場合は、ルネサスエレクトロニクスのサイトから zip ファイルをダウンロードして、ローカルファイルからインストールしてください）

1.2. エミュレータ（デバッガ）の接続設定

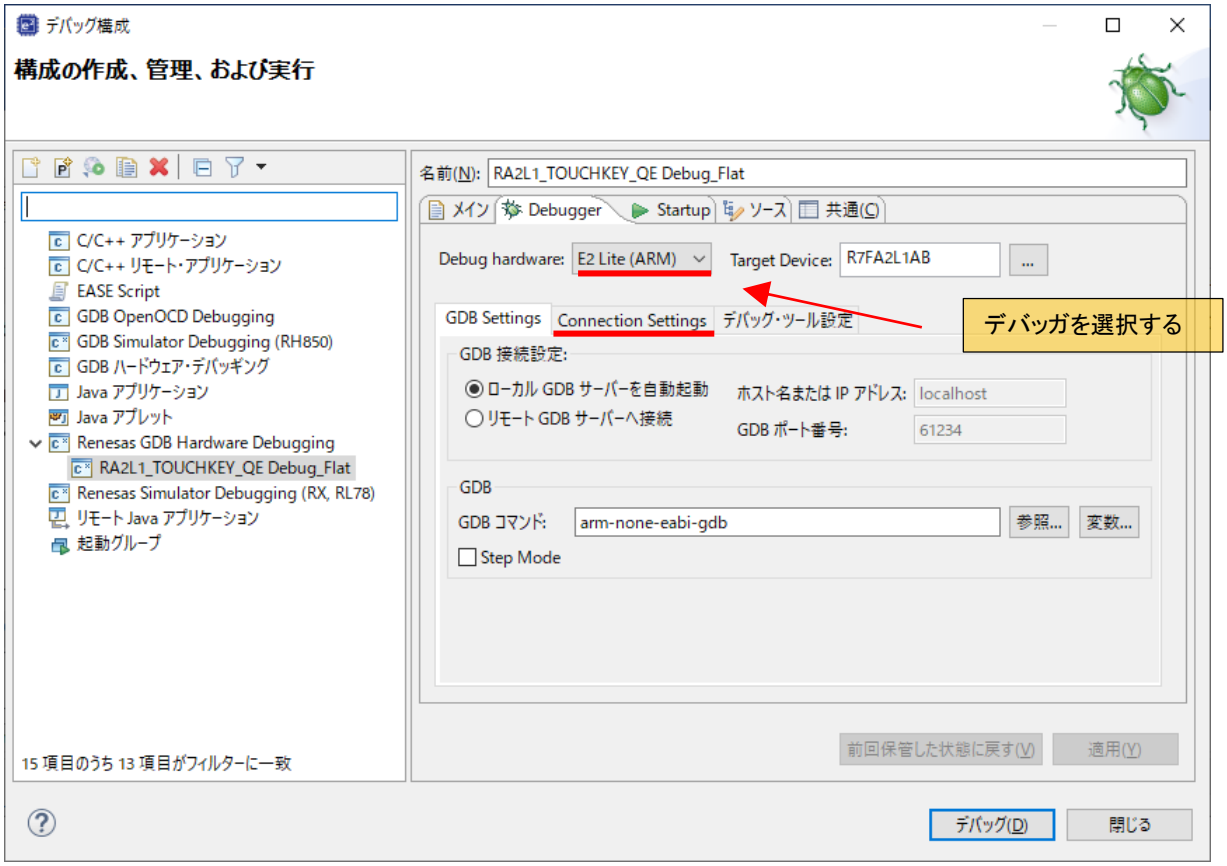
プロジェクトエクスプローラーから、対象プロジェクトを右クリック。



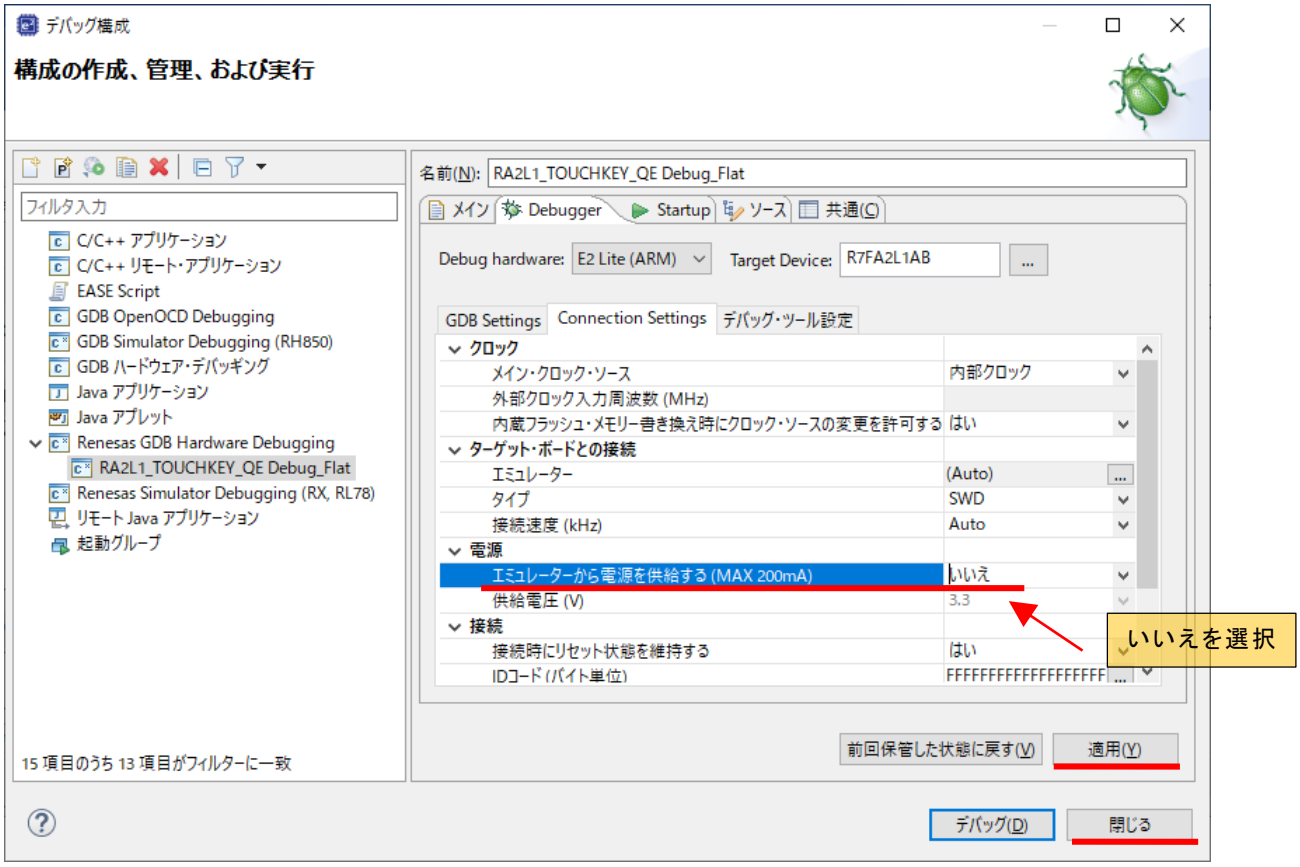
デバッグ – デバッグの構成



Debugger タブを開く。



Connection Settings タブを開く



エミュレータから電源を供給するする:「いいえ」
を選択してください。

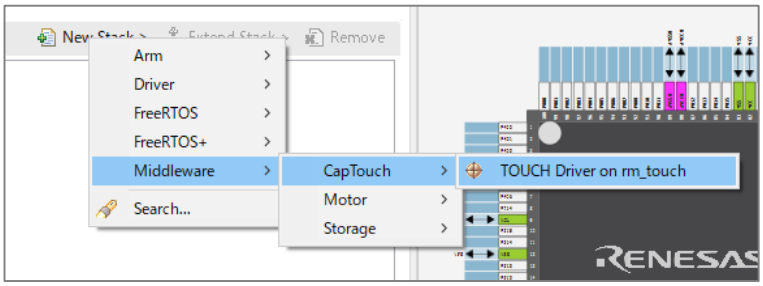
「適用」、「閉じる」で設定画面を閉じてください。

※エミュレータから電源を供給する事も可能ですが、その場合は
・別な箇所(USB 等)から電源を供給しない
様にしてください。

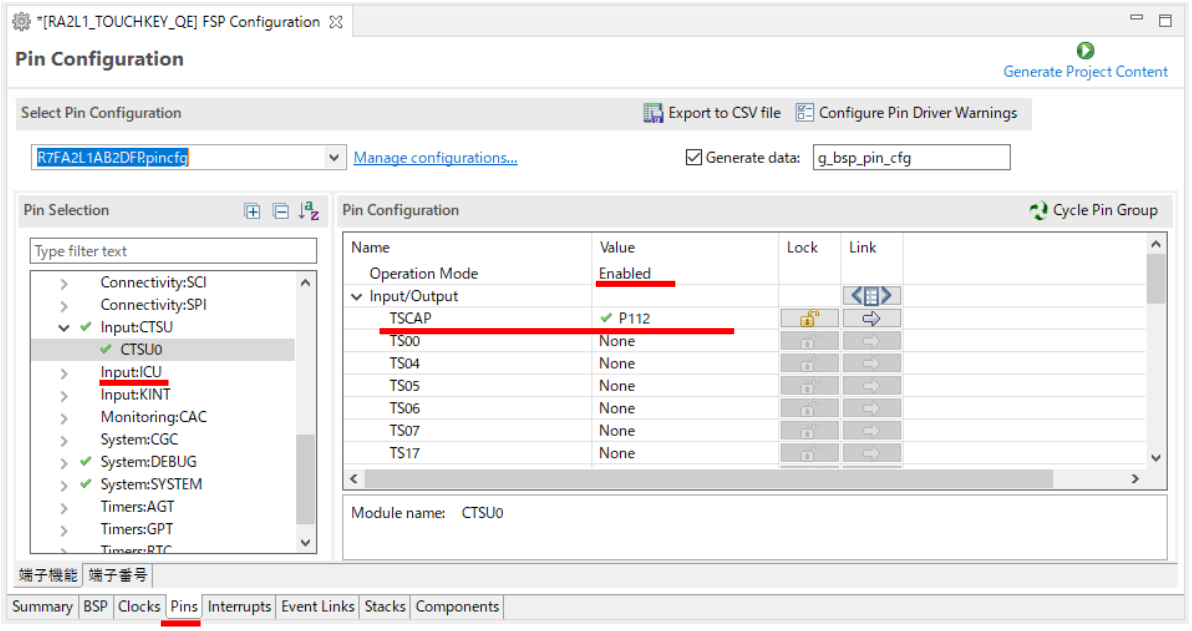
※なお、E2Lite をお使いの場合は、エミュレータから供給する電圧は、3.3V しか選択できません(E2 では、5V と 3.3V が選択可能です)

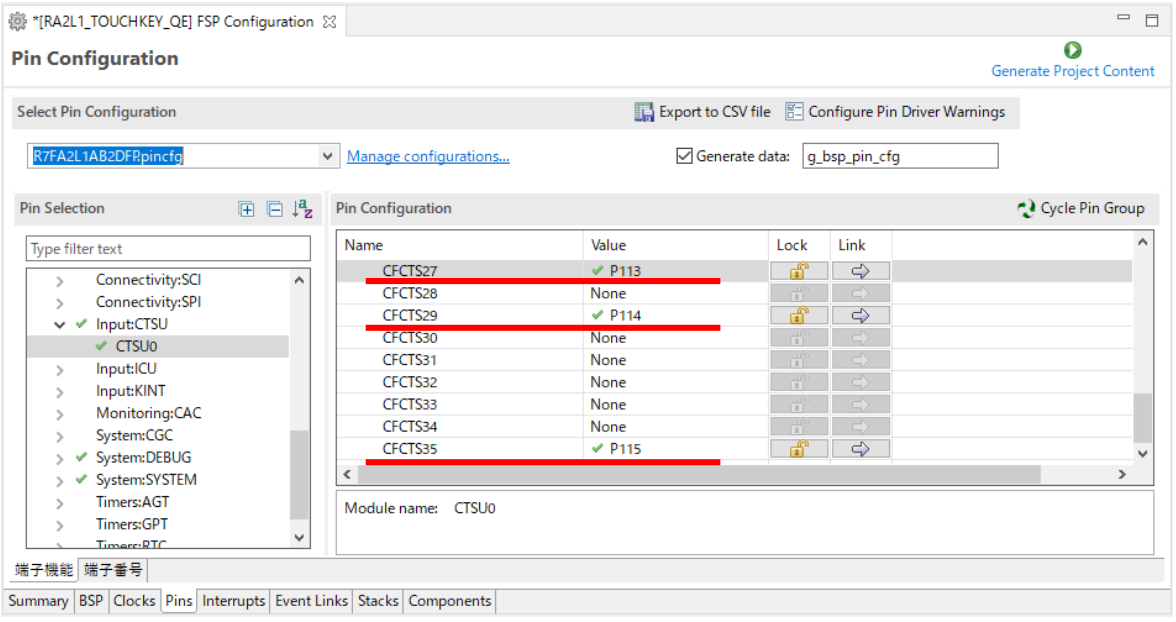
※「エミュレータ」と「デバッガ」、「Debugger」は同義です

1.3. FSP の設定



New Stack – Middleware – CapTouch – TOUCH Driver on rm_touch
を追加。



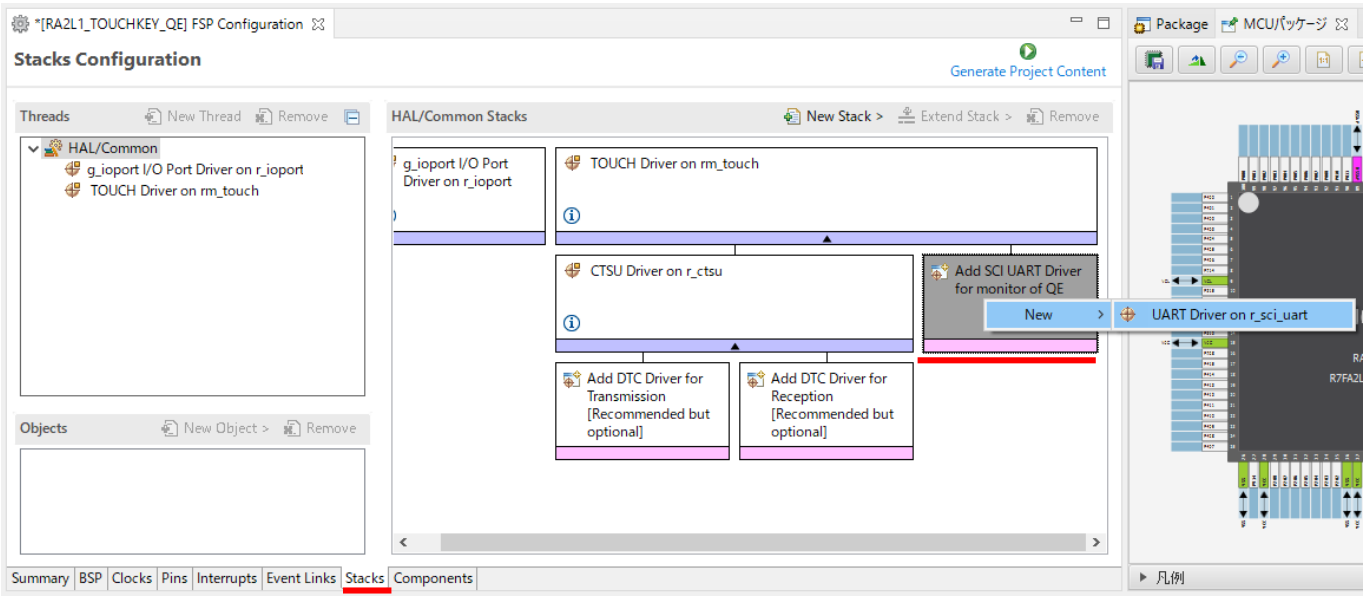


Pins タブ、Peripherals – Input:CTSU – CTSU0

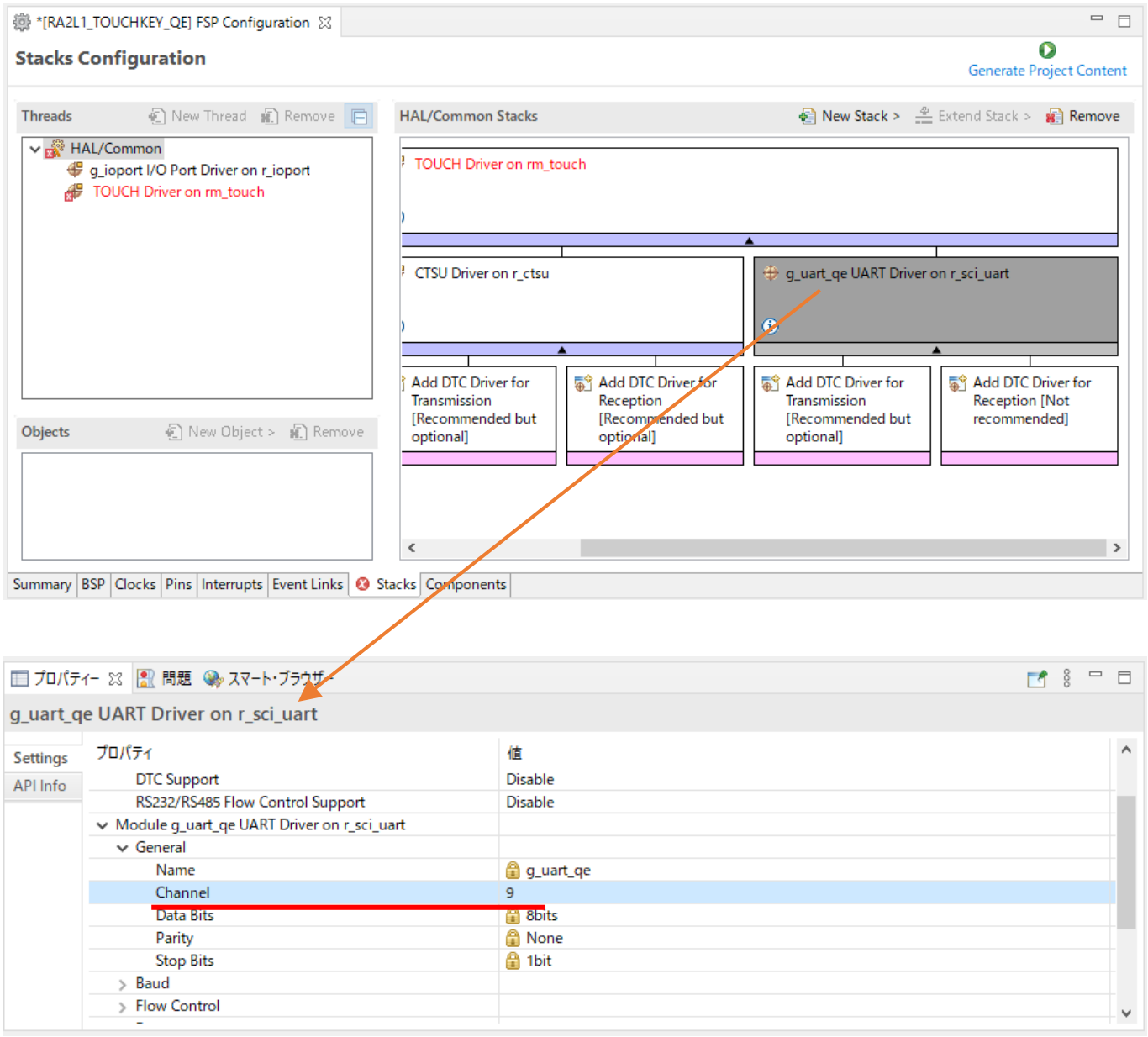
Operation Mode -> Enabled

	割り当てる端子	備考
TSCAP	P112	マイコンボード M-J16 をショートに設定
CFCTS27	P113	ベースボード B-K3
CFCTS29	P114	ベースボード B-K2
CFCTS35	P115	ベースボード B-K1

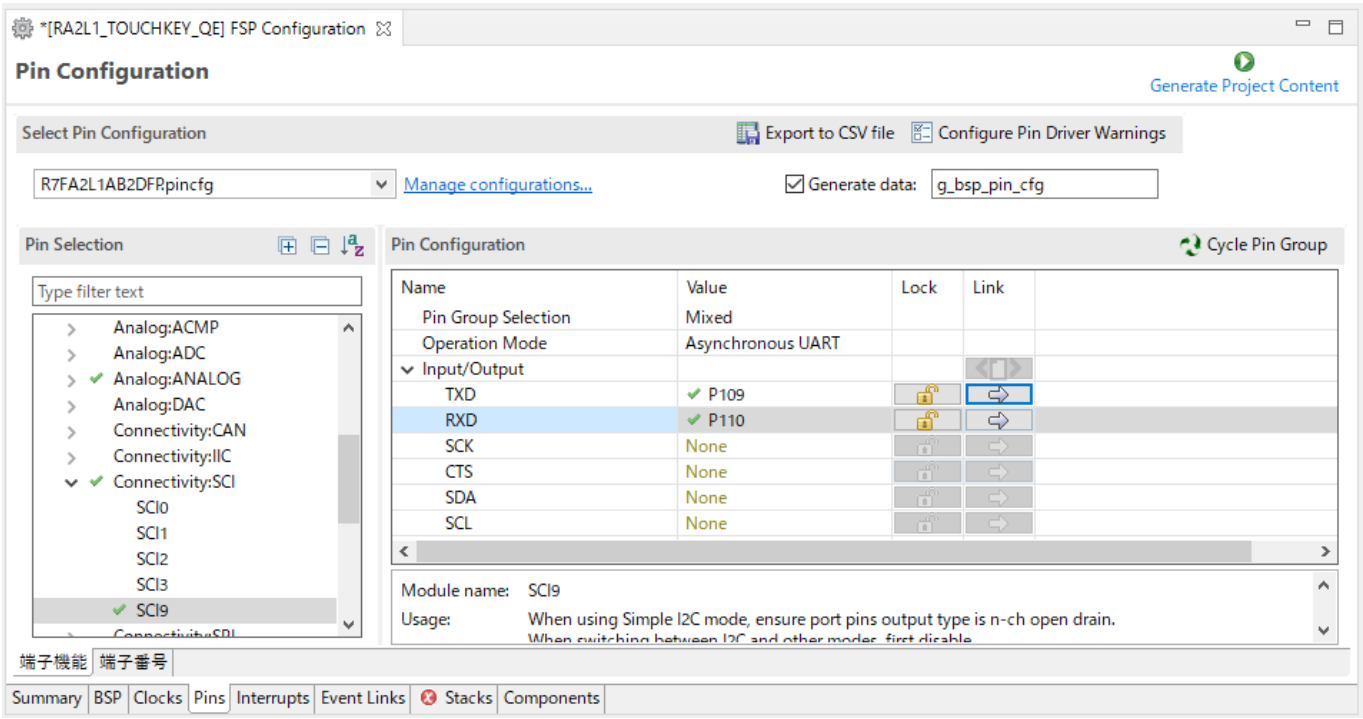
※割り当てる端子は、複数の中から選ぶわけではなく、有効にするか None かの選択です

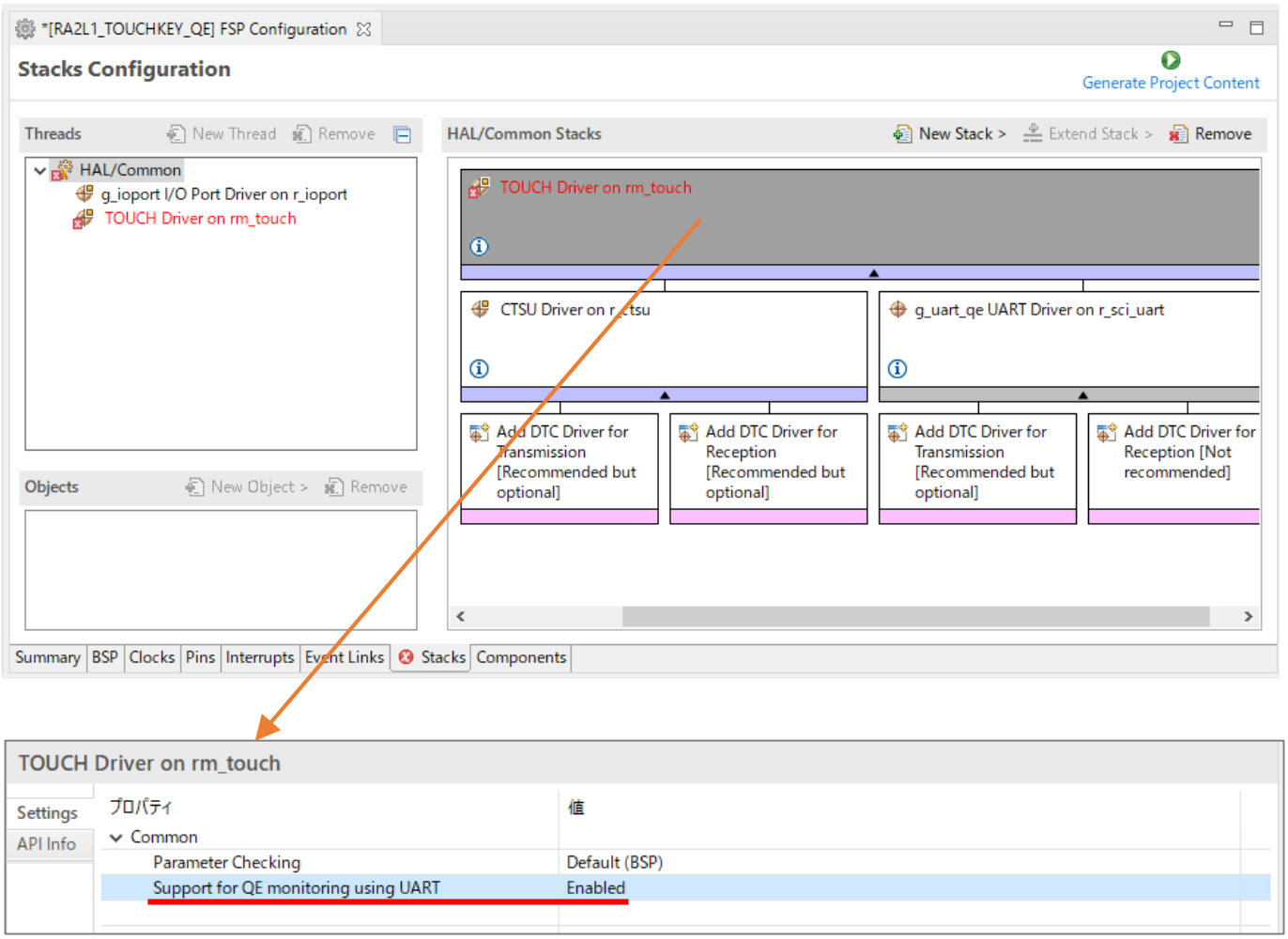


Stacks タブ、Add SCI UART Driver for monitor of QE に New – UART Driver on r_sci_uart を追加

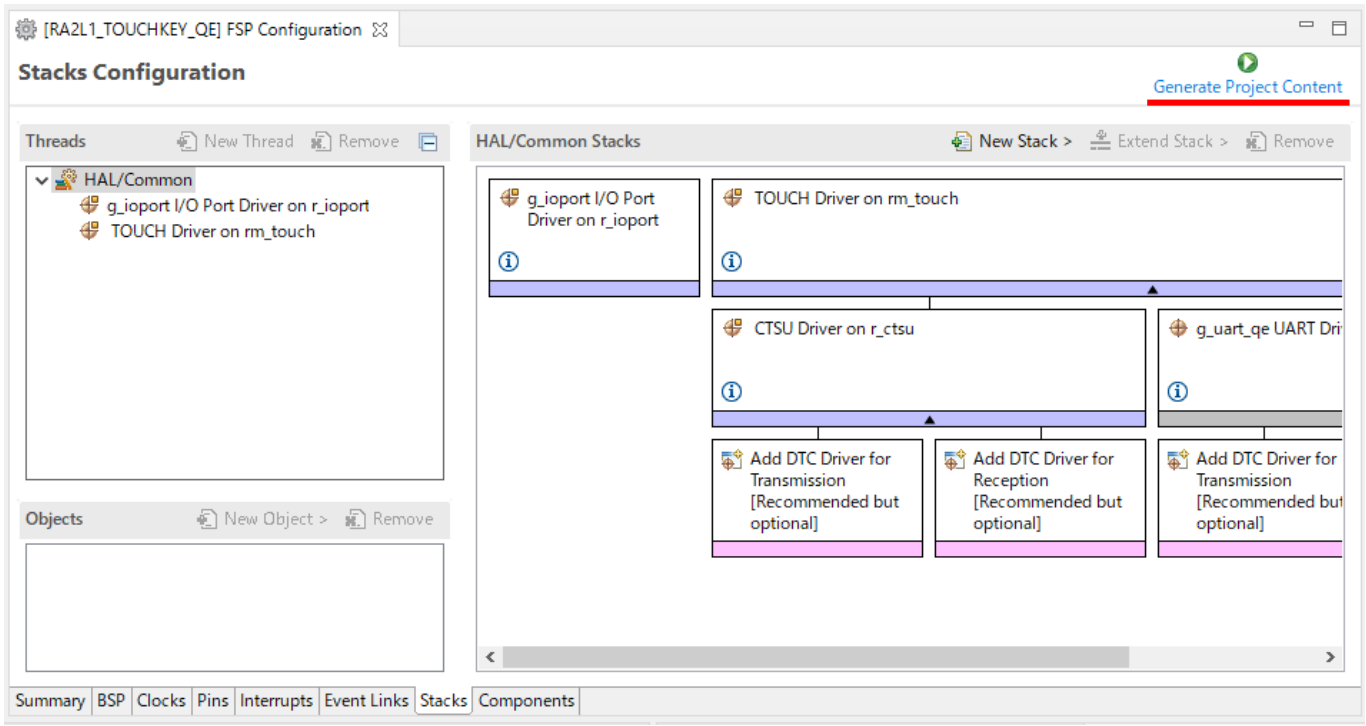


追加した、UART の Channel を 9 に変更。チュートリアル 5 を参考に、端子設定(TX:P109, RX:P110)を割り当て。



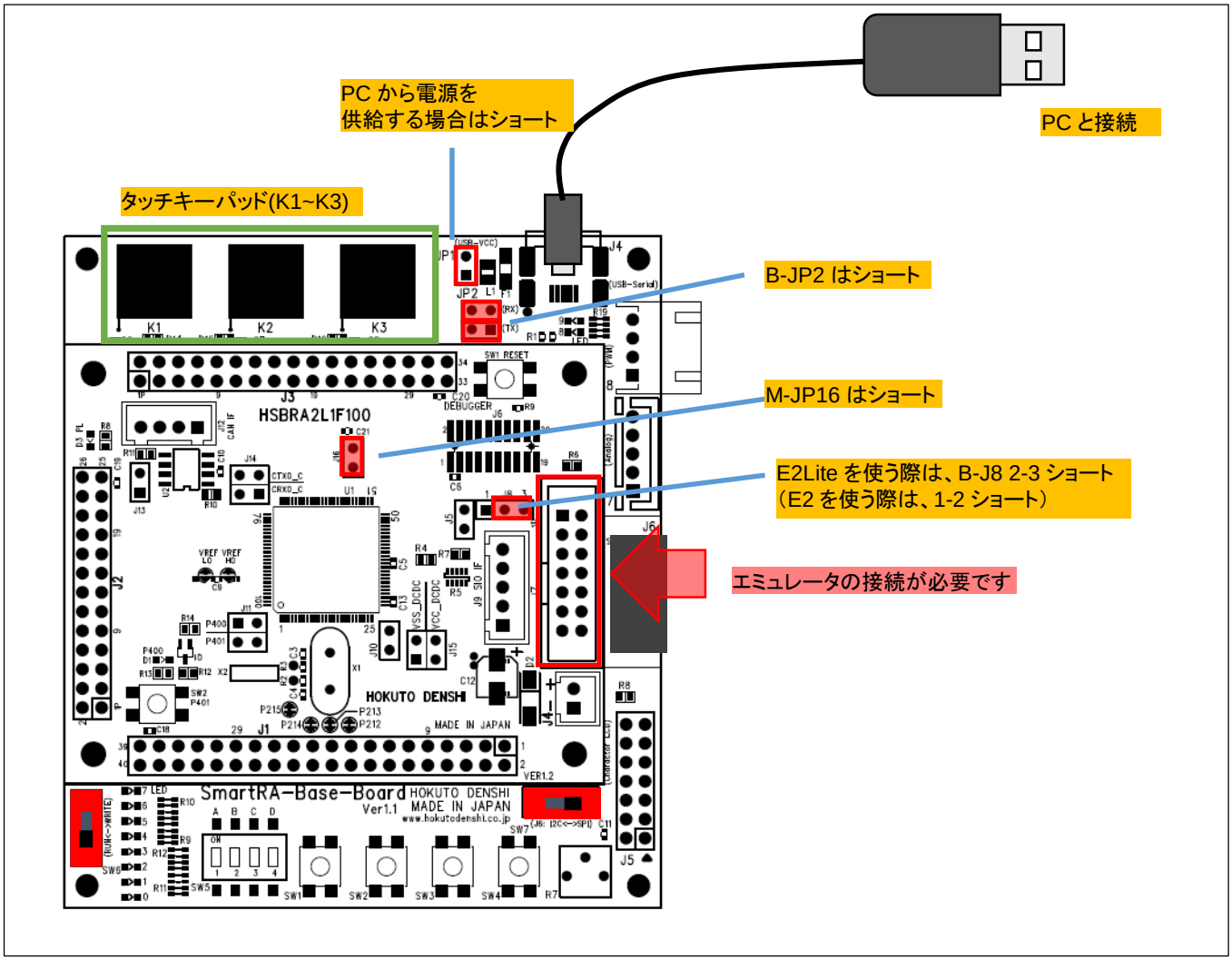


Support for QE monitoring using UART を Enabled に変更。(この変更で、Stacks のエラー(×印)が消えるはずです。)



一通り設定が終われば、Generate Project Content で、プロジェクトファイル一式を出力。

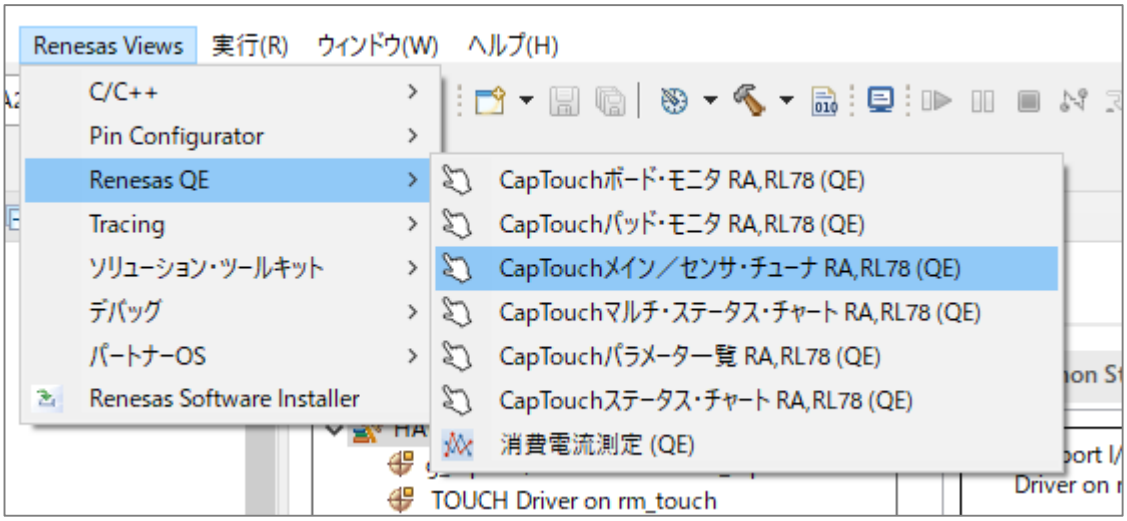
1.4. タッチキー動作を確認するためのボード設定



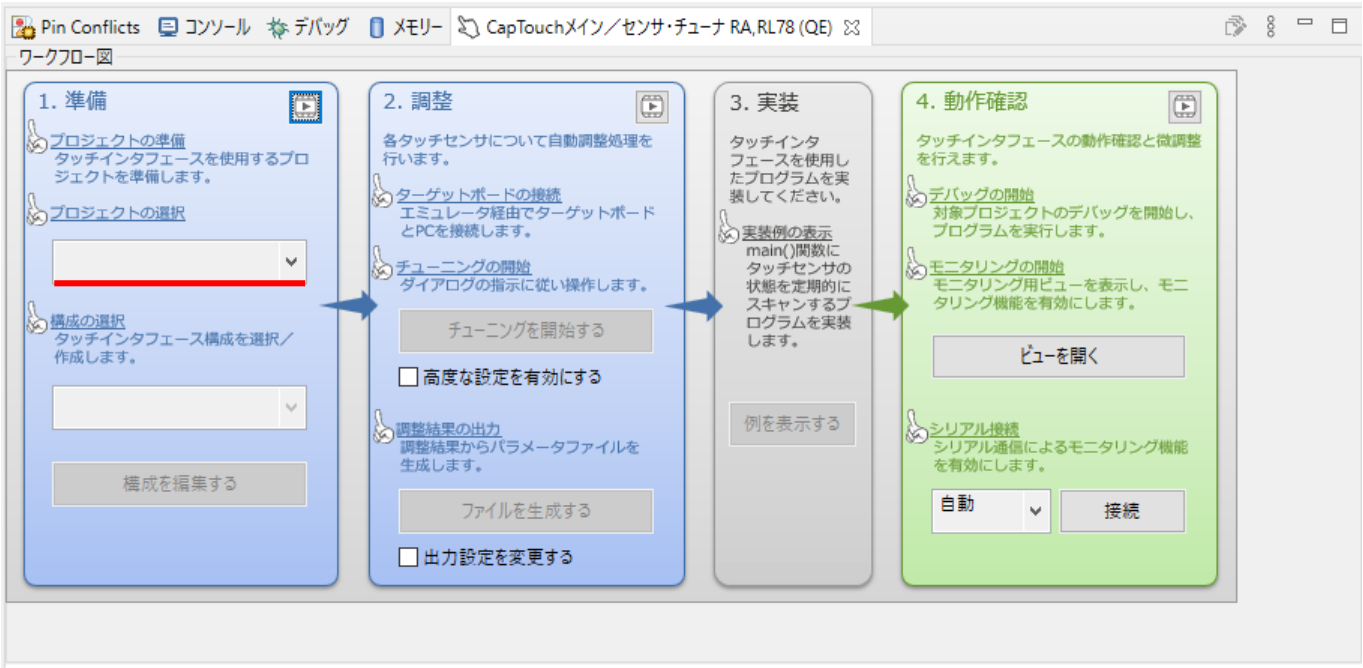
マイコンボードと PC を接続してください。ジャンパは上記設定とする。

1.5. QE for CapTouch の使用

1.5.1. プロジェクトの作成

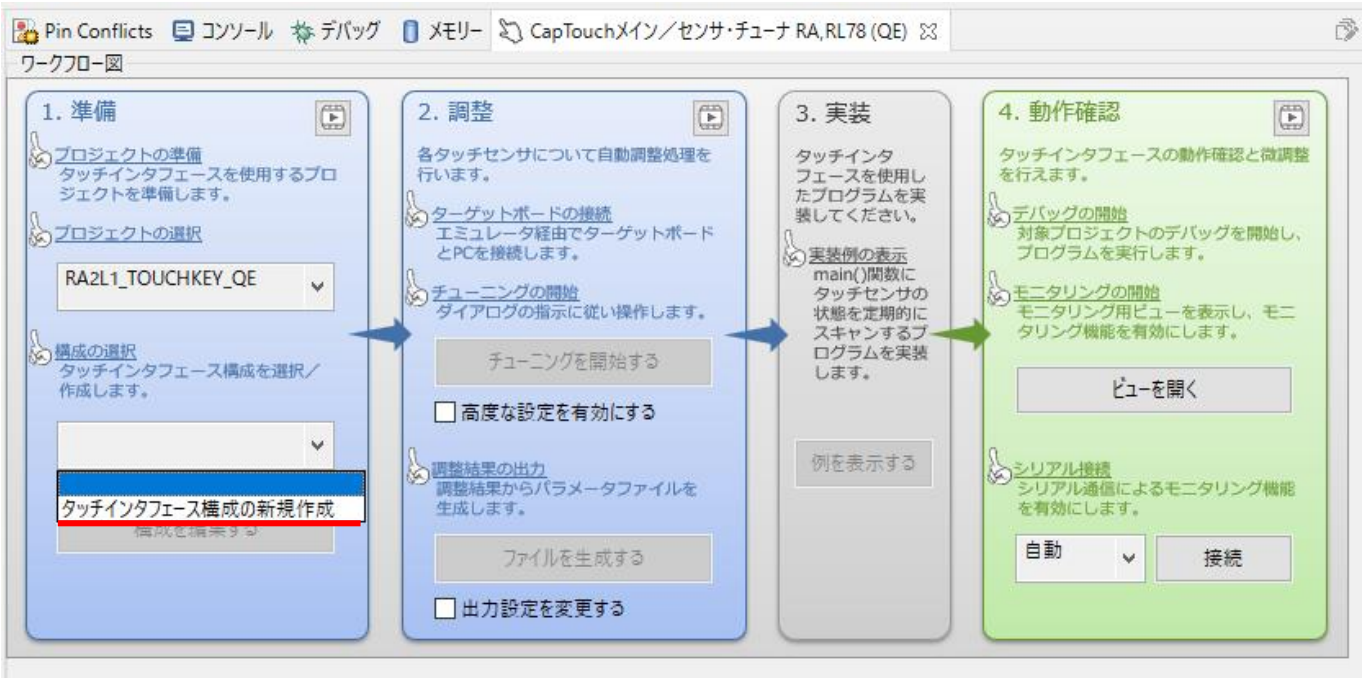


Renesas Views - Renesas QE – CapTouch メイン／センサ・チューナ
を起動。

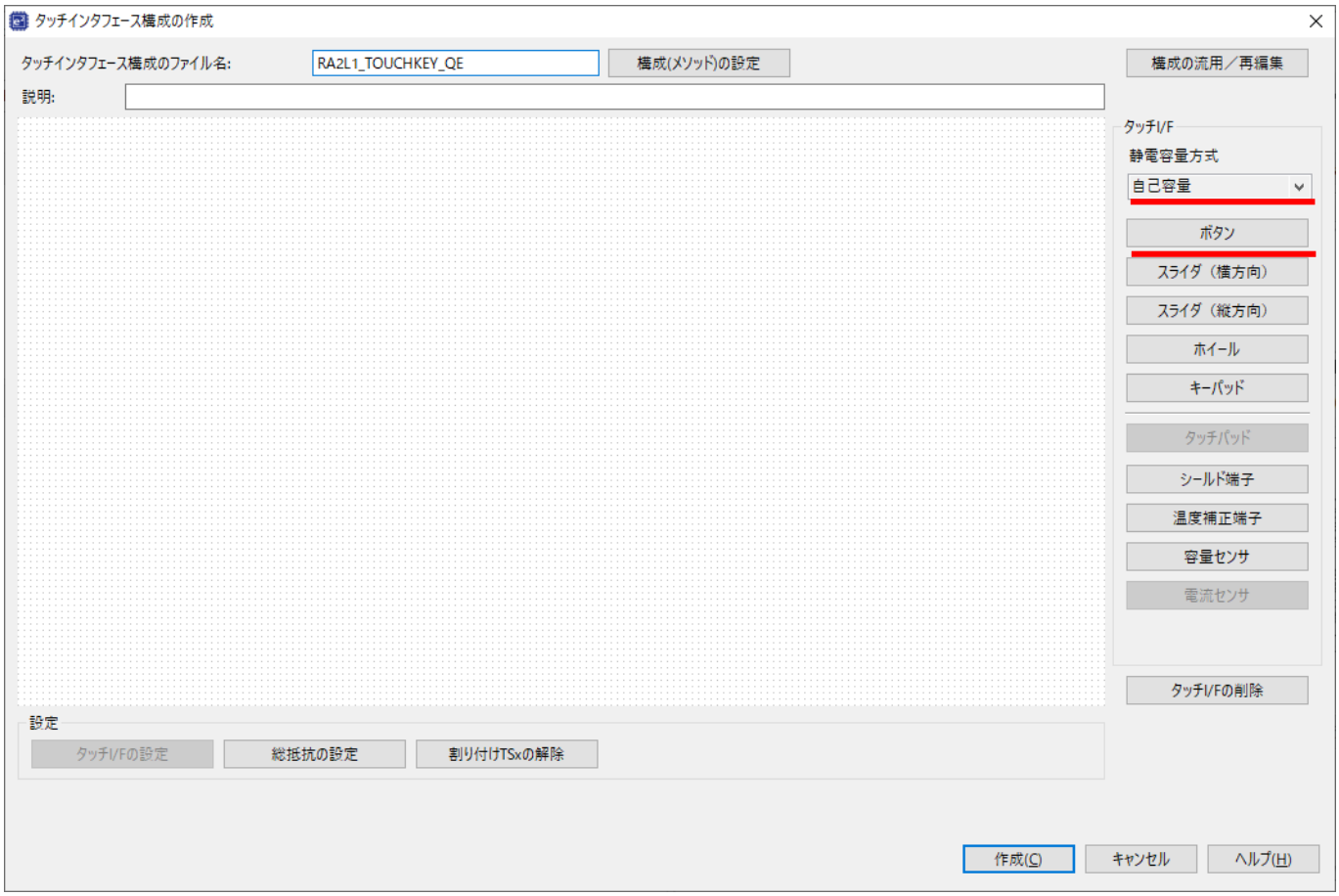


プロジェクトの選択

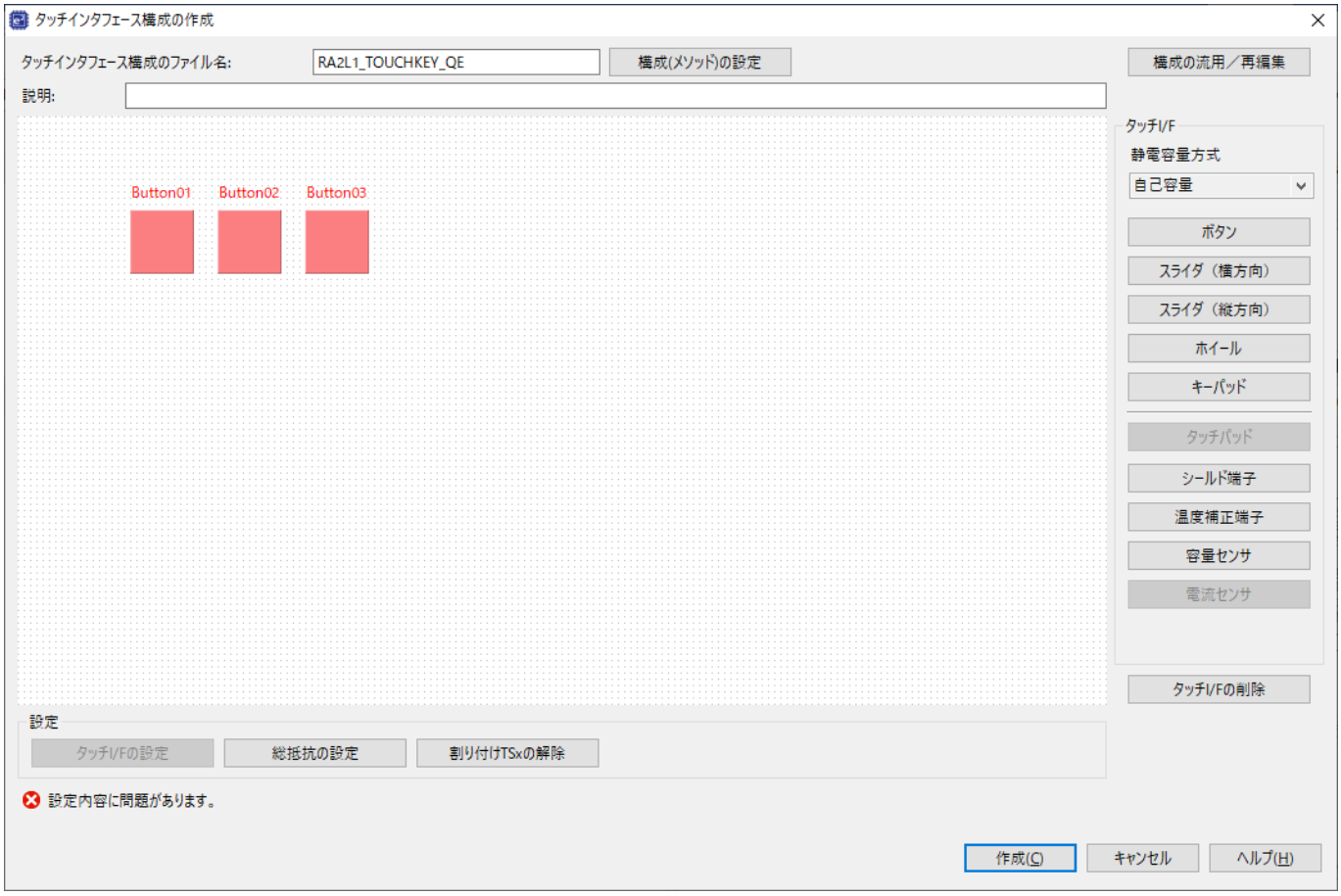
作成したプロジェクト（ここでは、RA2L1_TOUCHKEY_QE プロジェクト）を選択。



構成の選択 – タッチインタフェース構成の新規作成

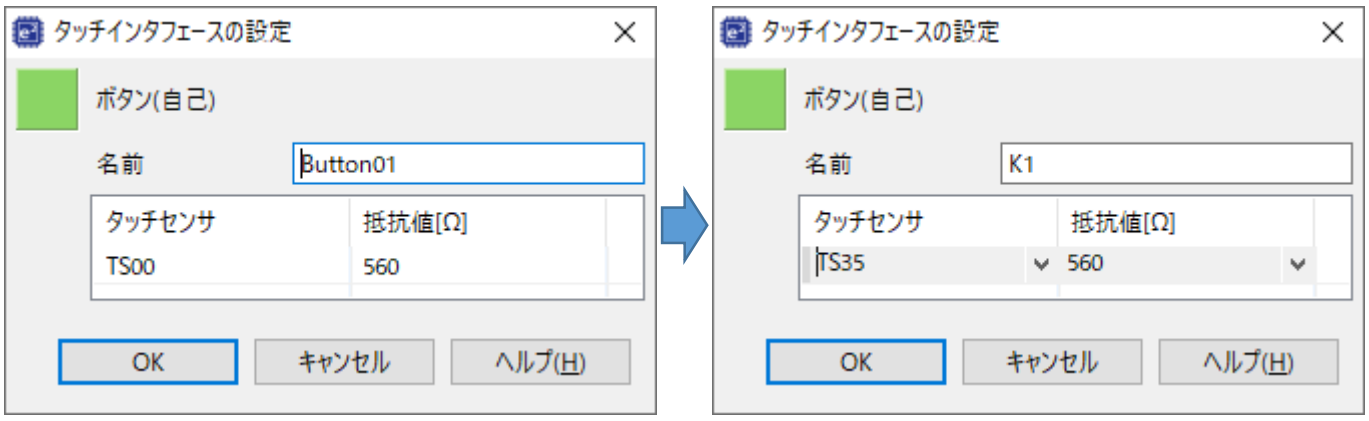


静電容量方式 – 自己容量を選択。ボタンを押す。

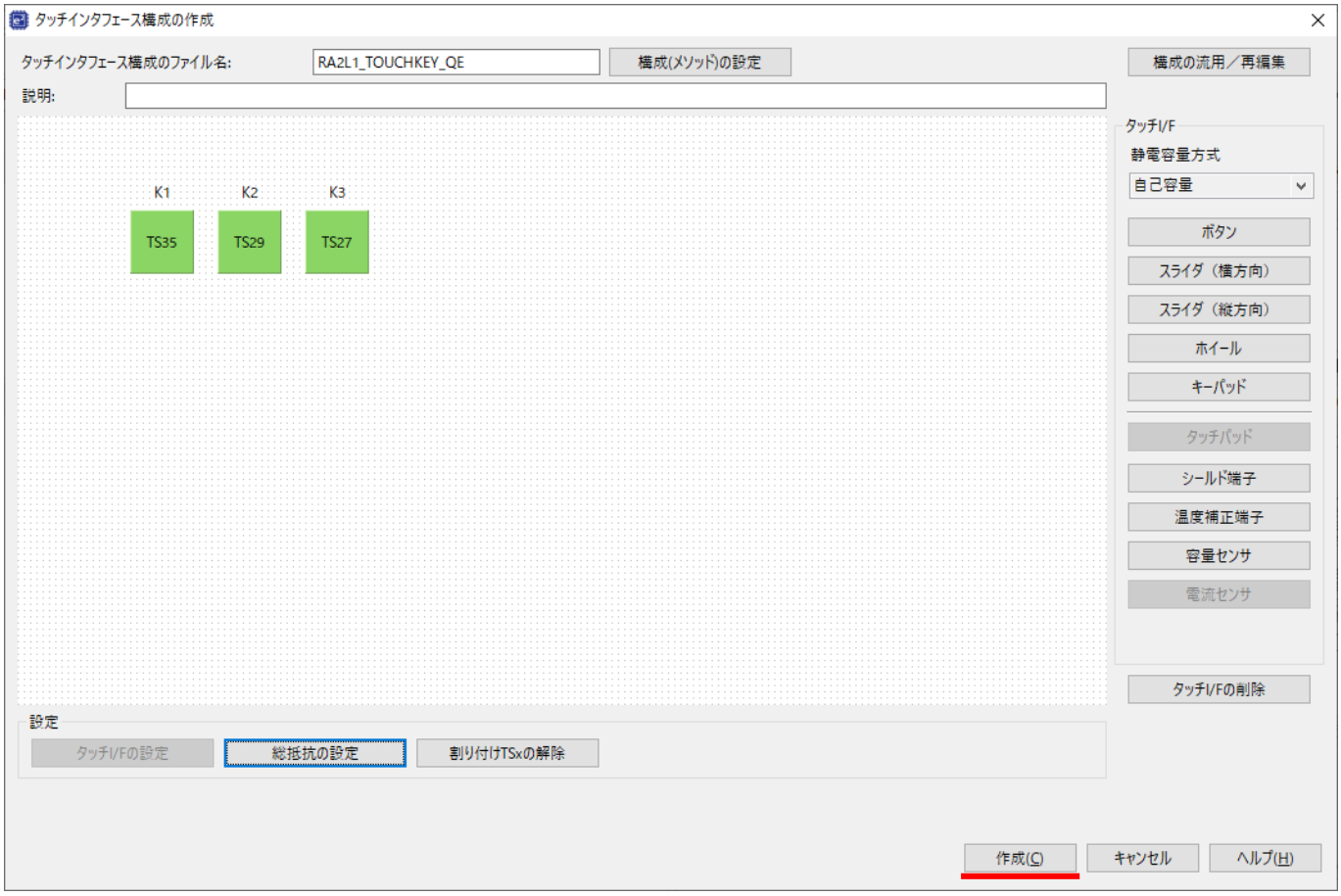


基板に合わせて、3 つのボタンを配置。

Button01 をダブルクリック。

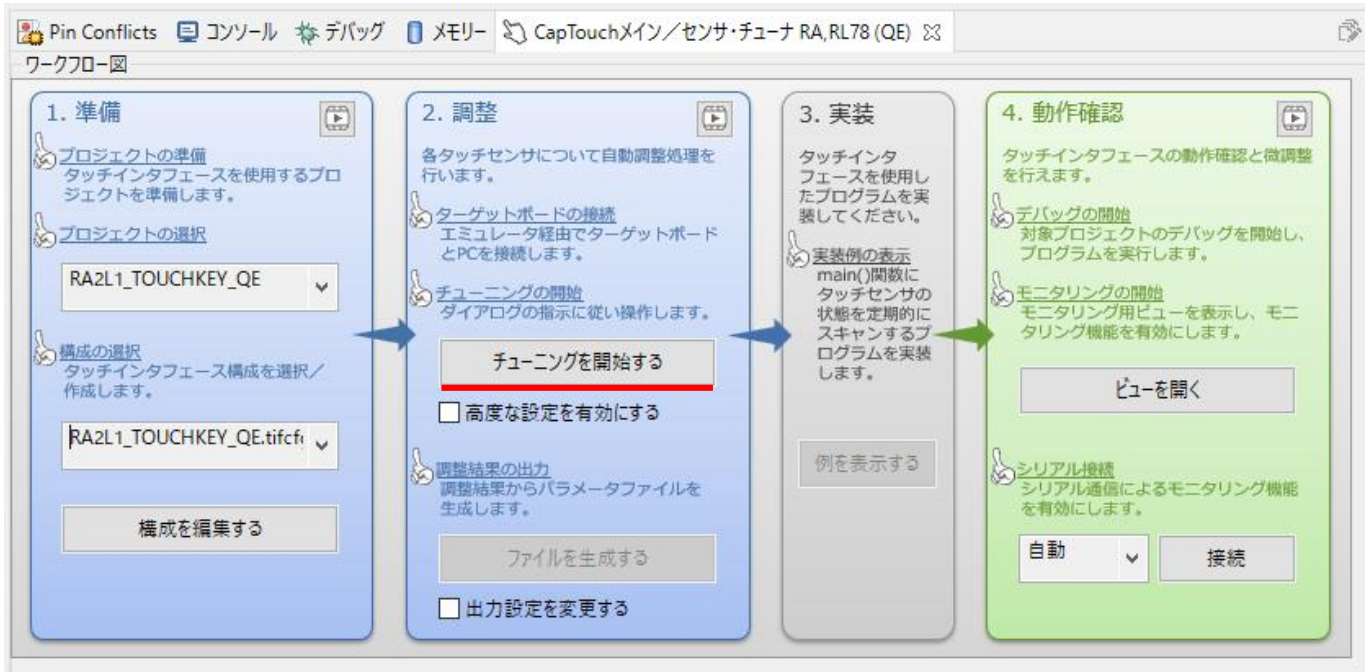


名前を K1、タッチセンサを TS35 に変更。560 Ωはそのままで良いです。

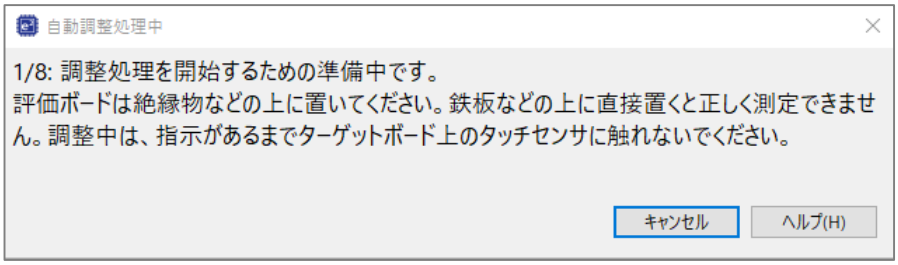


同様に、K2-TS29, K3-TS27 を設定してください。作成を押す。

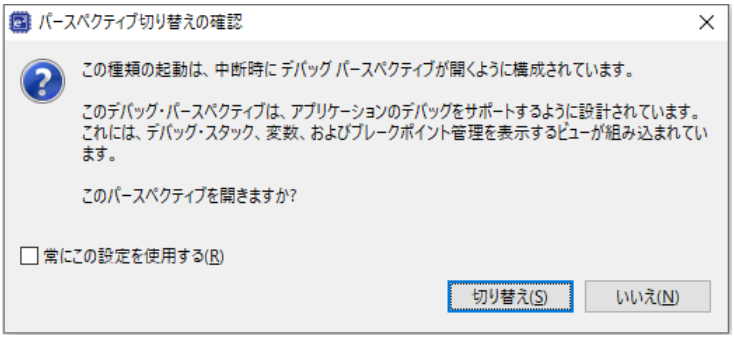
1.5.2. チューニング



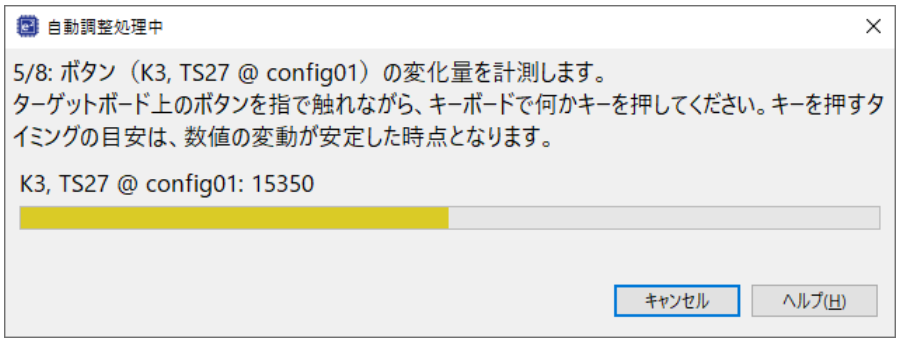
チューニングを開始する。



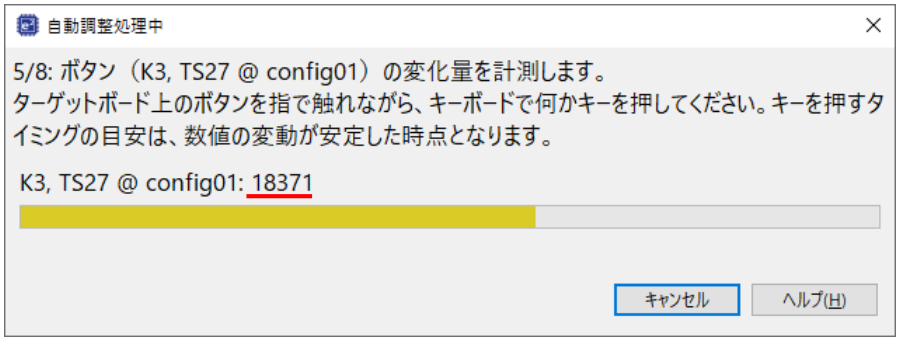
上記の様な画面が表示され、バックグラウンドで処理が進みます。



上記画面が表示された場合は、「切り替え」を押してください。



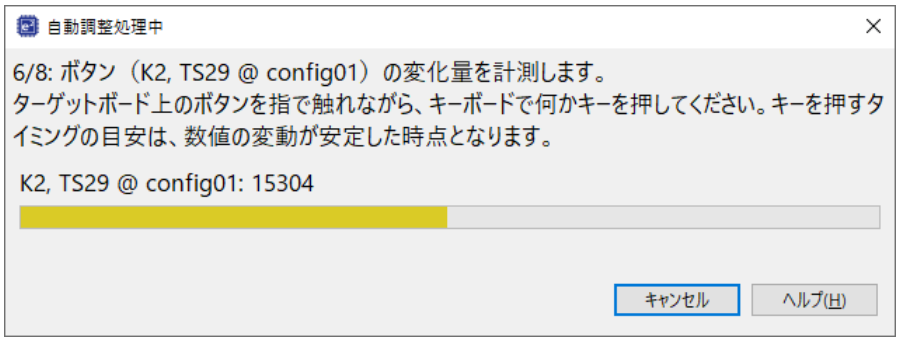
K3 のタッチパッドを指でタッチする様にしてください。（※最初にタッチするのは K3 です）



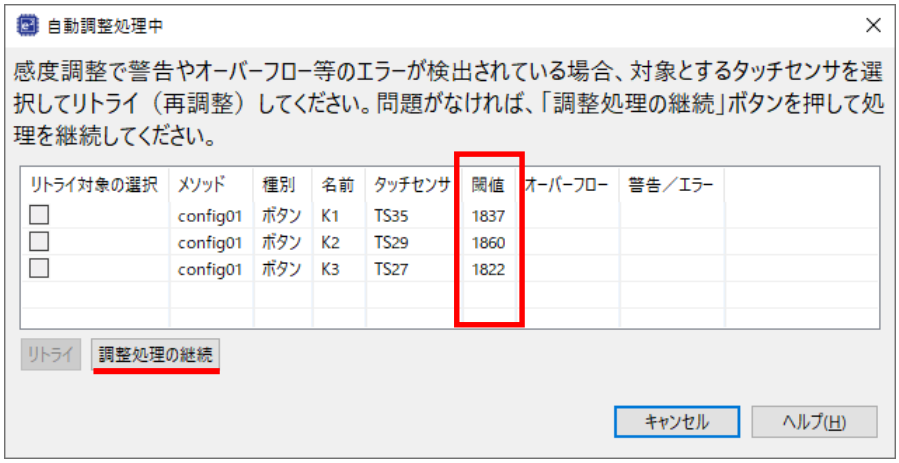
表示されている数値が大きくなれば、タッチを検出しています。

e2studio の画面(e2studio がフォアグラウンドになっている状態で)でいずれかキーを叩いてください。

（次のキーに進みます）

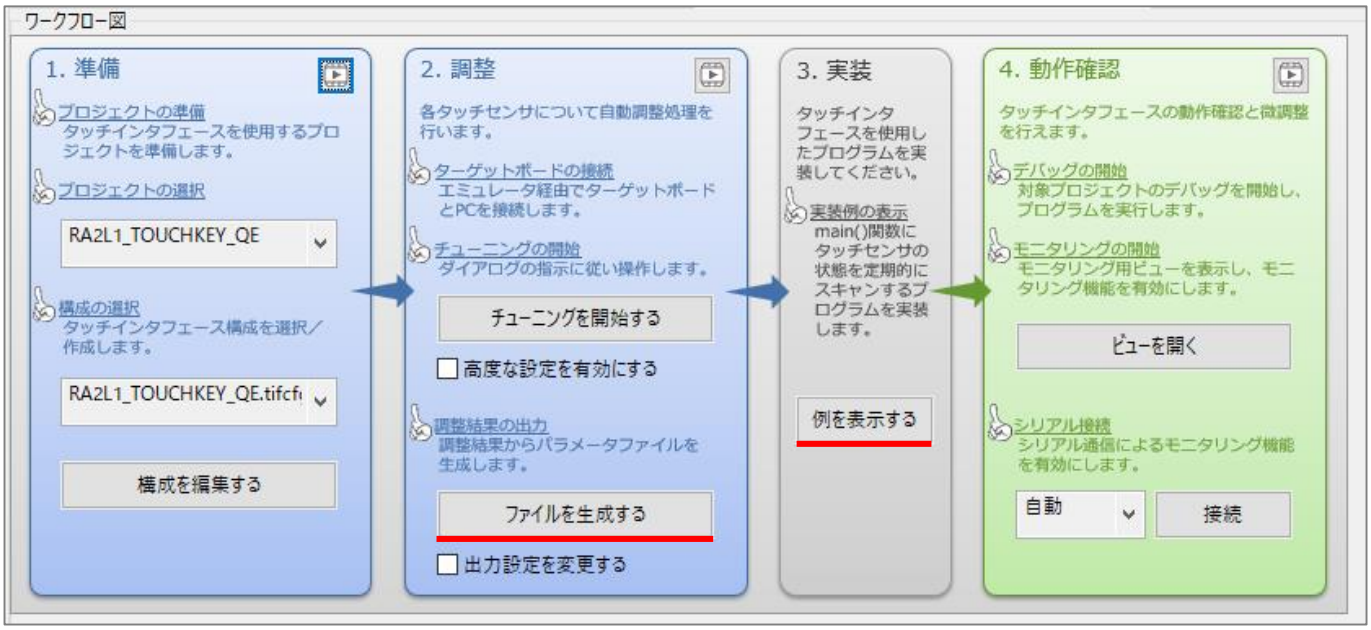


K2, K1 も同様に処理します。



閾値の欄がタッチしたことによる数値の増加に伴って決定された値です。3 値が大体同じ値になっている事が期待値です。極端に外れた値があれば、「リトライ対象の選択」のチェックボックスにチェックを入れてリトライしてください。問題なさそうであれば、「調整処理の継続」で先に進んでください。

1.5.3. プログラムコードへの反映



「ファイルを生成する」で、チューニング結果をソースファイルに反映させることができます。

「例を表示する」で、サンプルコートが表示されますので、hal_entry.c に反映させてください。

```
    /* Main loop */
    while (true)
    {

        /* for [CONFIG01] configuration */
        err = RM_TOUCH_ScanStart(g_qe_touch_instance_config01.p_ctrl);
        if (FSP_SUCCESS != err)
        {
            while (true) {}
        }
        while (0 == g_qe_touch_flag) {}
        g_qe_touch_flag = 0;

        err = RM_TOUCH_DataGet(g_qe_touch_instance_config01.p_ctrl, &button_status, NULL, NULL);
        if (FSP_SUCCESS == err)
        {
            /* TODO: Add your own code here. */
        }



        /* FIXME: Since this is a temporary process, so re-create a waiting process yourself. */
        R_BSP_SoftwareDelay(TOUCH_SCAN_INTERVAL_EXAMPLE, BSP_DELAY_UNITS_MILLISECONDS);
    }
```

上記が「例を表示する」で表示される、メインループ（無限ループ）の部分のコードです。「Add your own code here.」のところにユーザコードを追加します。

button_status が、キーをタッチしている情報が格納される変数です。

タッチキー	button_status	
K3	....xx**1**	キーに触れている際、b0 が 1 となる
K2	...x**1**x	キーに触れている際、b1 が 1 となる
K1	...**1**xx	キーに触れている際、b2 が 1 となる

button_status は、64bit の変数で、K3 に触れている場合は、b0 に 1 が立ちます。

・K3 のみに触れている　button_status = 0x1
・K2 のみに触れている　button_status = 0x2
・K1 のみに触れている　button_status = 0x4
・K3, K2 に触れている　button_status = 0x3
・K1~K3 に触れている　button_status = 0x7

となります。

```
    /* Main loop */
    while (true)
    {

        /* for [CONFIG01] configuration */
        err = RM_TOUCH_ScanStart(g_qe_touch_instance_config01.p_ctrl);
        if (FSP_SUCCESS != err)
        {
            while (true) {}
        }
        while (0 == g_qe_touch_flag) {}
        g_qe_touch_flag = 0;

        err = RM_TOUCH_DataGet(g_qe_touch_instance_config01.p_ctrl, &button_status, NULL, NULL);
        if (FSP_SUCCESS == err)
        {
            /* TODO: Add your own code here. */
            if(button_status & 0x1) led_on(3);  //K3
            else led_off(3);

            if(button_status & 0x2) led_on(2);  //K2
            else led_off(2);

            if(button_status & 0x4) led_on(1);  //K1
            else led_off(1);
        }
```

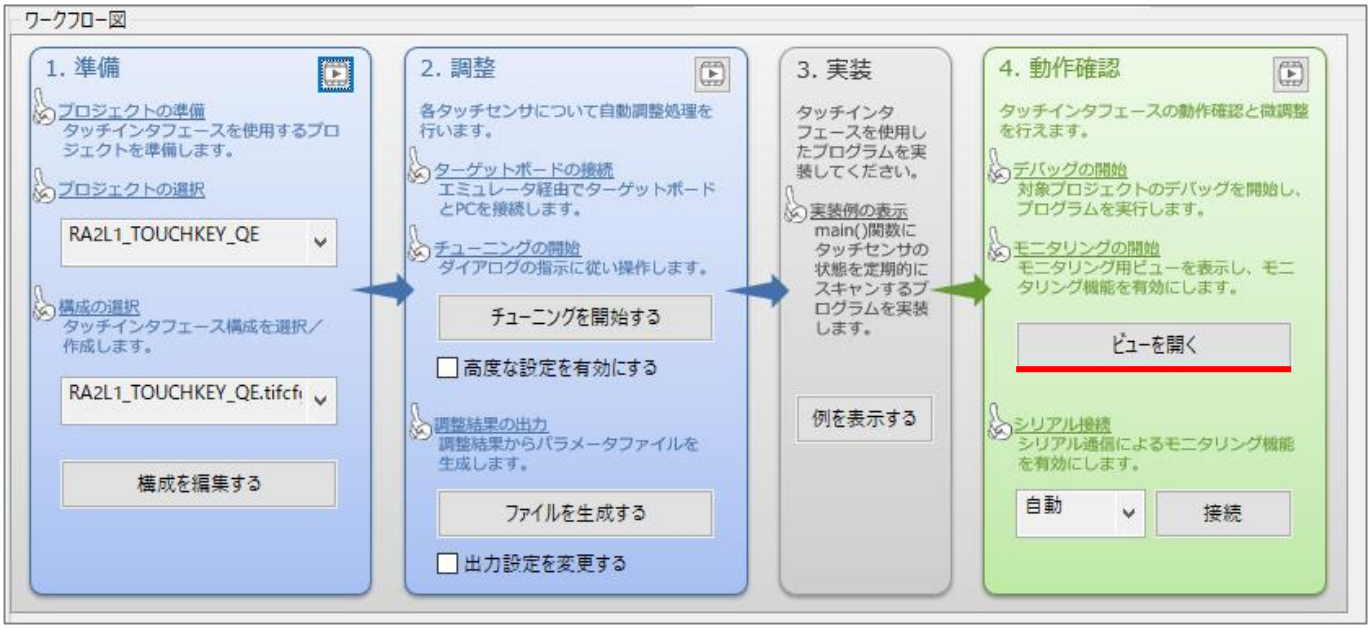
「Add your own code here.」のところに、LED を ON, OFF させるコードを追加すると上記の様になります。

※FSP の端子設定で、P000~P007 を出力に設定する必要があります

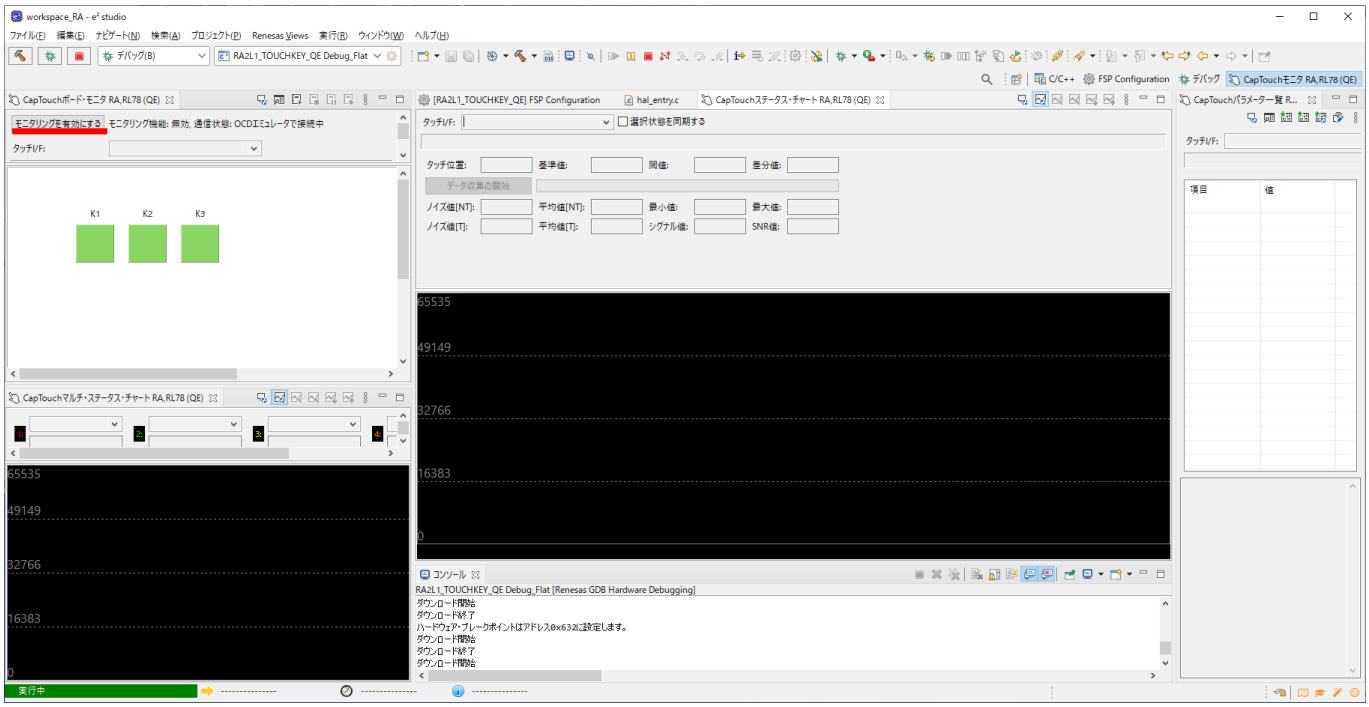
上記のコードを追加すると、K1 を押したとき LED0 が点灯します。K2（→LED1）, K3（→LED2）も同様です。
（キーにタッチしたときの動作とユーザプログラムを結びつけることができます）

1.5.4. モニタリング

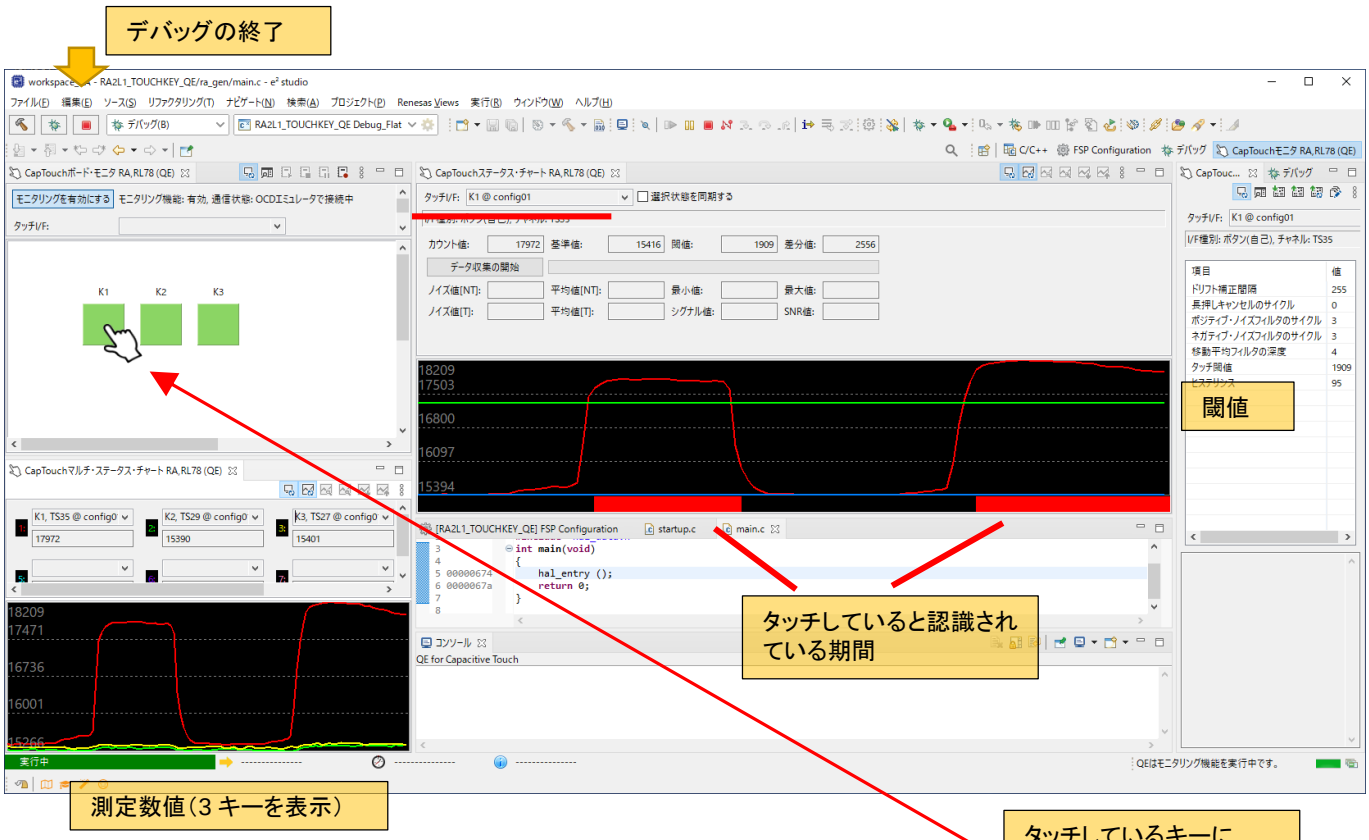
CapTouch には、タッチキーに触れている強さ等の情報をモニタする機能があります。



「ビューを開く」



モニタリングを有効にする。

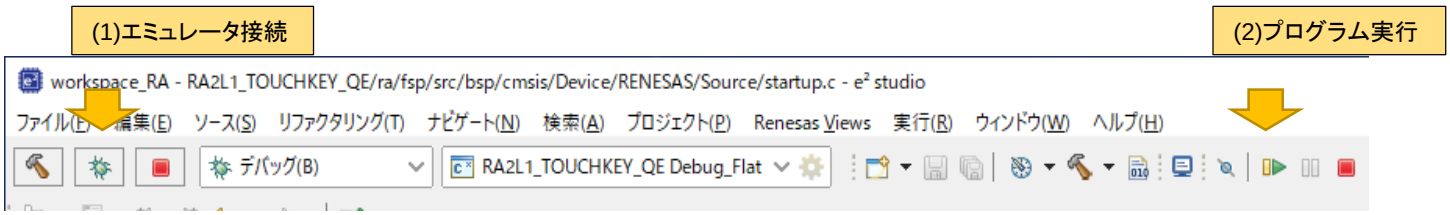


赤線の部分（タッチ IF の選択）で、表示させたいキーを選択。

タッチキーはタッチしていると、観測数値が上昇します。閾値（緑のライン）を超えるとタッチしているとみなされます。

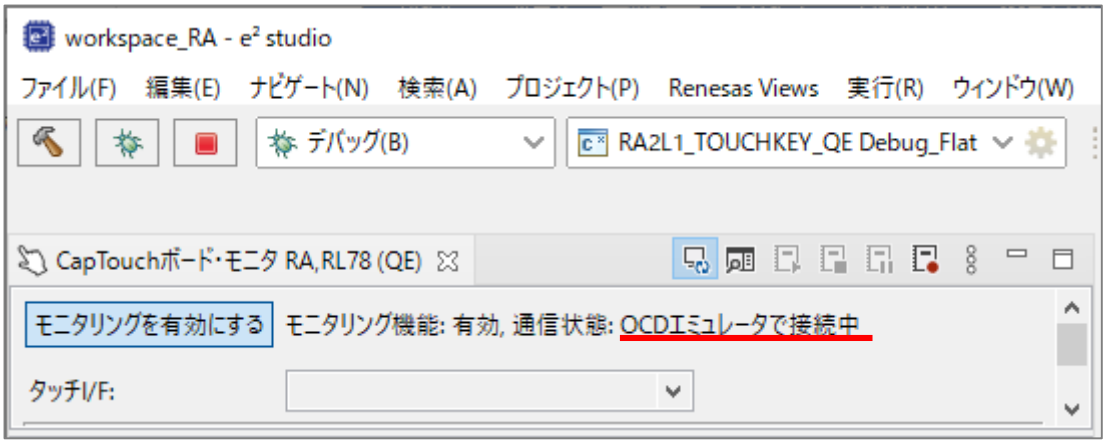
モニタを終了させる時は、
・「モニタリングを有効にする」のボタンを押し、モニタを停止する
・赤の四角のアイコンでデバッグを終了する
としてください。

※本機能は、エミュレータを使ったモニタなので、エミュレータ経由でプログラムを実行する必要があります。エミュレータ経由でプログラムが実行されていない場合は、以下の手順でプログラム実行後に、モニタリングを実行してください。

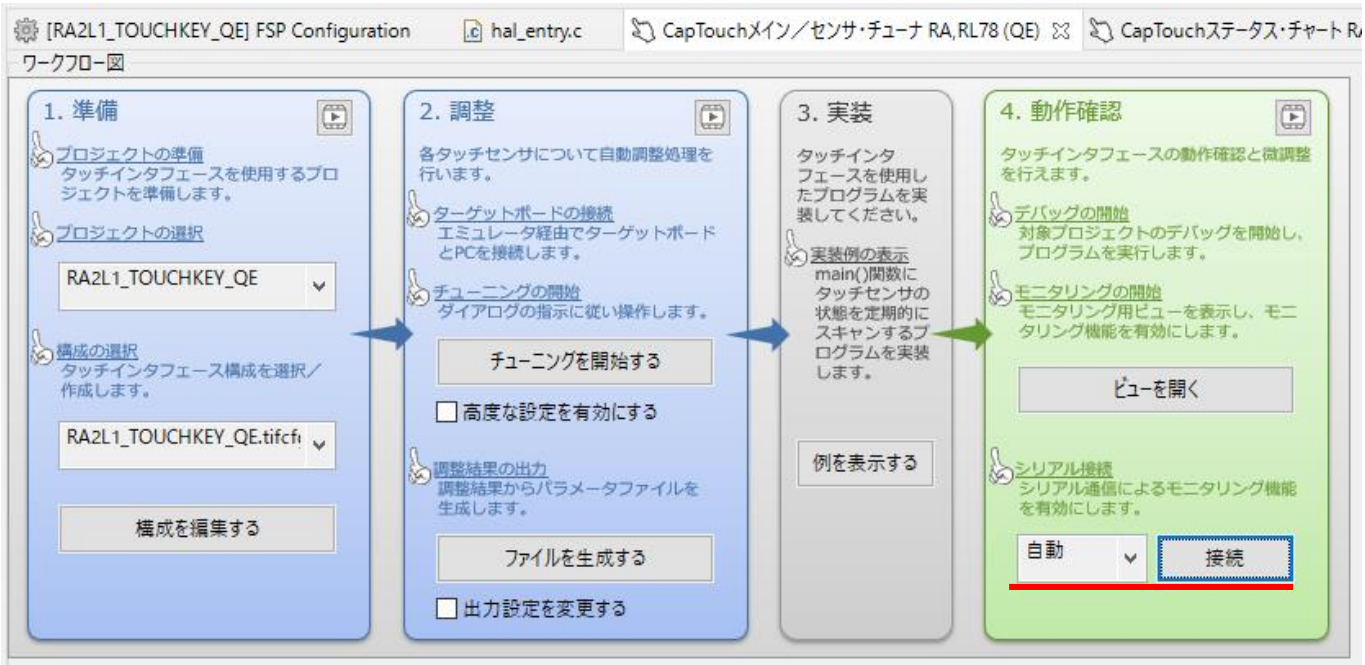


(1)エミュレータ接続（プログラムをエミュレータ経由でマイコンボードにダウンロード）
(2)実行

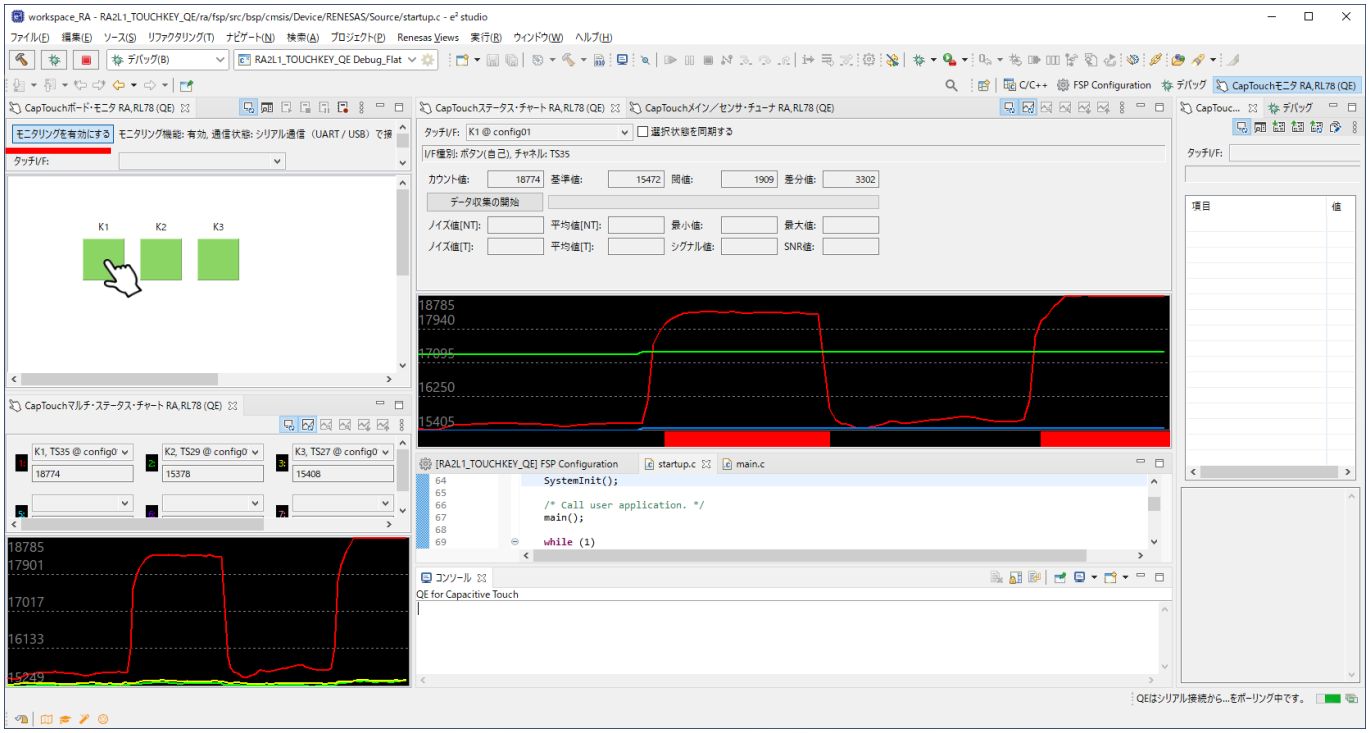
1.5.5. COM ポートでのモニタリング



Cap Touch のモニタは、エミュレータ経由で実行されますが、COM ポートを使ってモニタする方法もあります。
COM ポート経由では、ボード単体で動作します。（エミュレータ不要です）
※上記画面はエミュレータ接続の場合です



CaTouch メイン／センサ・チューナの画面で、シリアル接続「接続」を押す



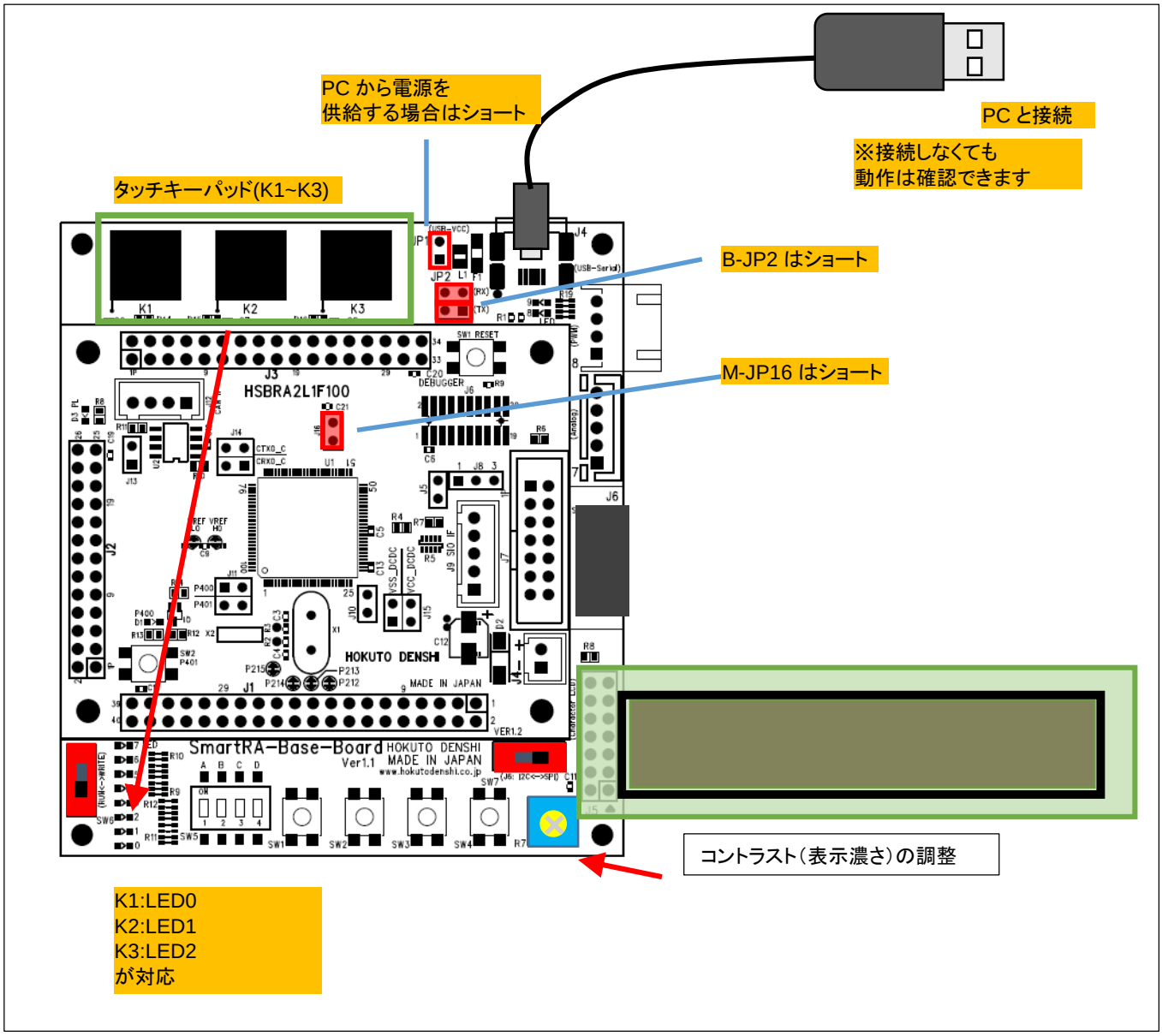
エミュレータ接続と同じように、キーの状態をモニタ可能です。初期設定時（最初のチューニング）はエミュレータが必要ですが、動作をモニタする際は、ボード単体（マイコンボード＋ベースボードの組み合わせ）でも実行できます。

（サンプルプロジェクト RA2L1_TOUCHKEY_QE の動作を見るのであれば、エミュレータがなくても、シリアル通信経由で行えます。新規にプロジェクトを作成して、チューニングを行う場合は、E2Lite エミュレータ等が必要になります）

2. RA2L1_CTSU

前の章では、ミドルウェア(CapTouch)を使って、タッチキーの動作環境を構築しました。本チュートリアルは、タッチキー(CTSU)のプログラムを（自動コード生成を使わずに）書き下してみたというサンプルとなります。

2.1. プログラムの動作

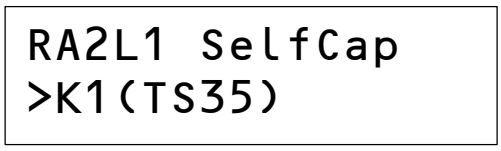


```
Copyright (C) 2021 HokutoDenshi. All Rights Reserved.

SmartRA KIT CTSU(TouchKey) TUTORIAL

Please do NOT touch key pad!

offset optimize & no-touch data scanning...
SO = 0
SO = 20
SO = 40
 （中略）
SO = 600
SO = 620
  K1(TS35)  SO setting value = 622
  K2(TS29)  SO setting value = 628
  K3(TS27)  SO setting value = 634
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
>
```

起動すると、シリアル端末には上記の様な表示が出ます。

また、LCD 画面には、

の表示が出ます。ここで、タッチキーパッドの K1 に触れてみてください。



上記の様な表示となるはずです。

K1 を押している時には、LED0 が点灯します。同様に、LED1(K2), LED2(K3)が点灯します。

タッチ（触れた）キーが LCD に表示されます（複数のキーにタッチした場合は、一番強く（しっかりと）触れているキーが表示されます。

シリアル端末側は、キーボードからコマンドを受け付ける様になっていて、"m"コマンドを打つと

・m コマンド

```
sens value monitor(press any key to exit)>
key :     K3(TS27)  K2(TS29   K1(TS35)
initial :   2626     2600      2598

sens value :
            2600     2601      2596
            2672     2633      3478*
            2652     2620      4060*
            2645     2600      2601
            2662     4012*     2669
            2601     2607      2607
            3998*    2655      2603
            2670     2618      2603
            2656     3759*     4118*
            2669     3807*     4111*
            2719     2656      2673

             K3       K2        K1
```

2 秒毎に 1 行表示

K1 に触れた場合

K2 に触れた場合

K3 に触れた場合

K1 と K2 に触れた場合

上記の様に、リアルタイムにキー毎の測定値（どのぐらいタッチしているか）が表示されます。

・t コマンド

タッチの閾値をチューニングするコマンドです。

```
threshold value optimize...

Please touch key pad -> K1
& PC keyboard enter
```

上記表示がでますので、K1 のタッチキーにタッチして、PC のキーボードを（任意）叩いてください。

```
touch data scaning...

 key scan end.

  key pad(TS ch) = K1(TS35)

  touch(ave) = 4054 no-touch(ave) = 2589
```

タッチ時と、非タッチ時の数値が表示され、次のキー(K2)のチューニングに移ります。

```
Please touch key pad -> K2
& PC keyboard enter

touch data scaning...

 key scan end.

  key pad(TS ch) = K2(TS29)

  touch(ave) = 4025 no-touch(ave) = 2626


Please touch key pad -> K3
& PC keyboard enter

touch data scaning...

 key scan end.

  key pad(TS ch) = K3(TS27)

  touch(ave) = 4040 no-touch(ave) = 2630


 tuning end.
```

・p コマンド

```
print initial reference value>

TSxx    :   SO :  Sens : threshold
K3(TS27):  635 :  2630 : 1.100
K2(TS29):  629 :  2626 : 1.100
K1(TS35):  624 :  2589 : 1.100
```

SO（後述）値と、タッチしていない場合の測定値(Sens)、タッチキー判定の閾値（割合）が表示されます。上記は、チューニング（t コマンド）前の値です。

```
print initial reference value>

TSxx    :   SO :  Sens : threshold
K3(TS27):  635 :  2630 : 1.268
K2(TS29):  629 :  2626 : 1.266
K1(TS35):  624 :  2589 : 1.283
```

t コマンドでチューニングを行うと、threshold の値が更新されます。

・ベースボードのタッチキーに直接タッチした場合の測定値、チューニング後の閾値例

	非タッチ時の 測定値	タッチ時の 測定値	閾値 （非タッチ時の 測定値に対する 倍率のセンター）
K1	2589	4054	1.283
K2	2626	4025	1.266
K3	2630	4040	1.268

通常タッチキーは、導体（本キットでは基板パターンの銅箔）の上に、アクリル板等を設けます。本キットでは、アクリル板が無い代わりに基板（1.6mm 厚）を誘電体として使用しています。

・ベースボードのタッチキーにアクリル板（3mm 厚）を載せてタッチした場合の測定値、チューニング後の閾値例

	非タッチ時の 測定値	タッチ時の 測定値	閾値 （非タッチ時の 測定値に対する 倍率）
K1	2605	2875	1.052
K2	2592	2836	1.047
K3	2618	2849	1.044

アクリル板を挟んだ場合は、タッチ時の測定値の上がり方がマイルドになります。（このケースでは、導体の上に、基板＋アクリル板の 4.6mm 厚の誘電体が挟まっています。）

・s コマンド

```
sens statistics(measure 15 sec)>

sens value statistics data

key(TSxx) :   Ave (   Max -   Min , 3sigma )
K3(TS27)   :  2640 (  2711 -  2576 , 77 )
K2(TS29)   :  2652 (  2721 -  2622 , 36 )
K1(TS35)   :  3866 (  3901 -  3832 , 35 )
```

15 秒間の測定を行い、15 秒間の最大－最小、標準偏差 x3(3σ)値を表示します。上記は、15 秒間ずっと K1 のみタッチしていた場合の表示例です。

・i コマンド

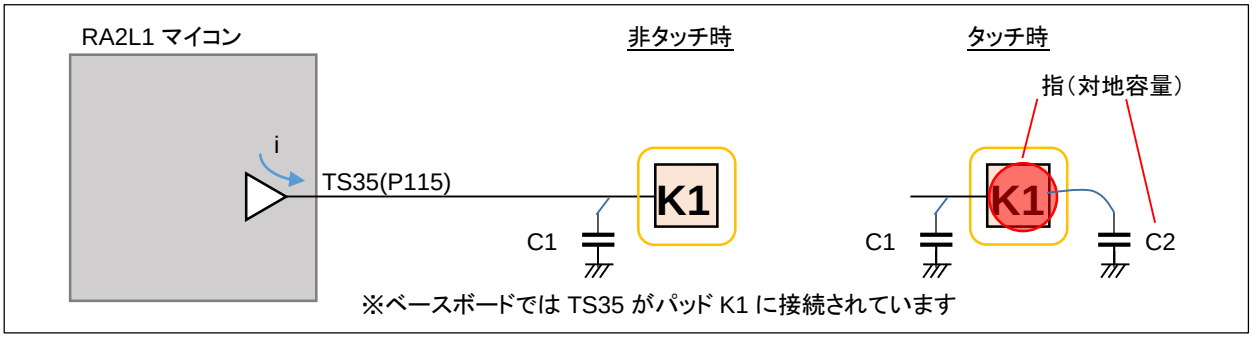
```
re-initialize(please do not touch key)>
ref current, current offset optimize.
SO = 0
SO = 20
..（中略）
initial sens value measure...
initialize finished.
```

再初期化を行います（リセットボタンを押した場合と同じです）。

2.2. タッチ判定の原理

本キットでは、「自己容量」タイプのタッチキーとなっています。

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）。

・非タッチ時の TS35 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 TS35 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 1.1（10%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。

2.3. タッチキー処理の流れ

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)CTSU 初期化

・TSCAP 端子設定
・TSxx 端子設定
・CTSU レジスタ設定

(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）（t コマンド）

・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「電流オフセット値」の設定により変化するため、上記(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。

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。

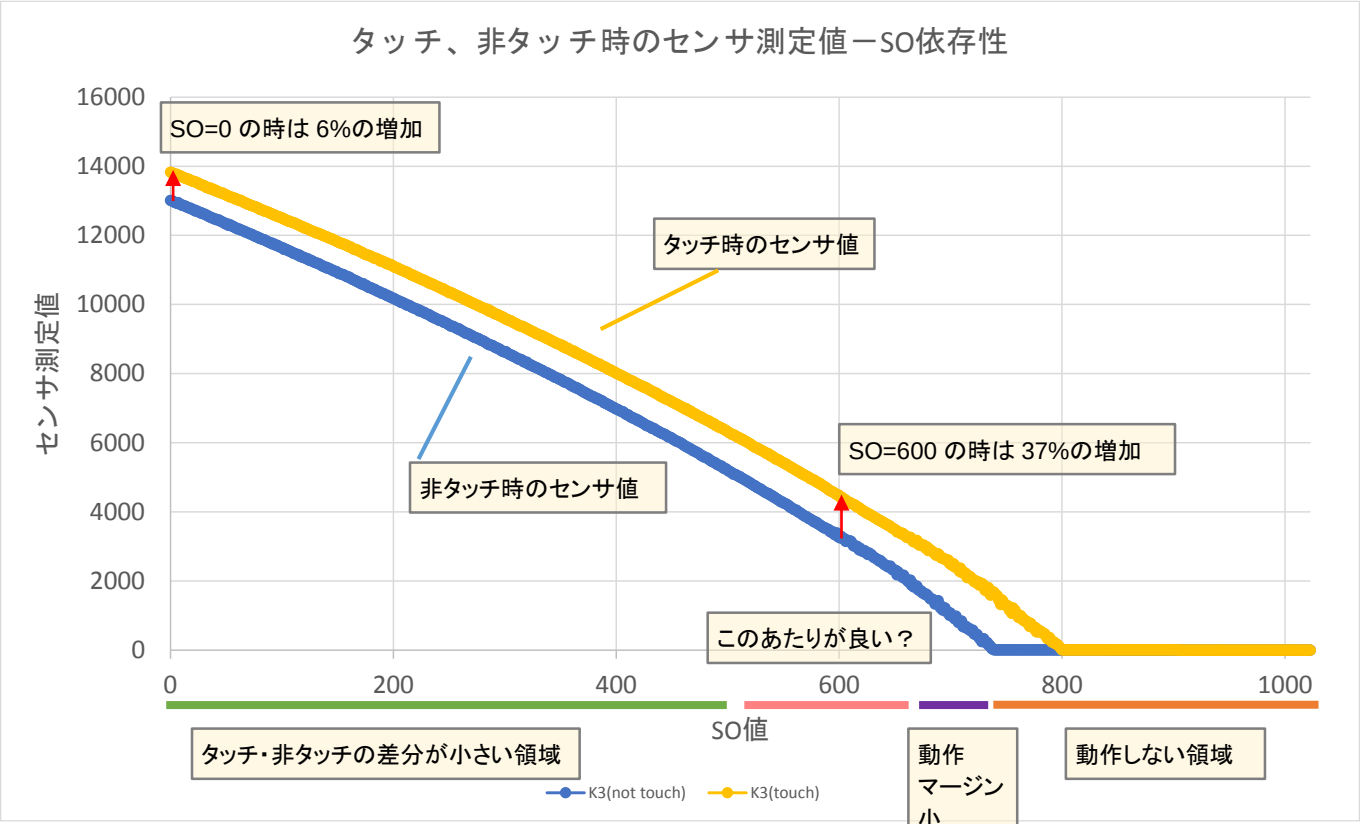


図 2-1 センサ値 SO 依存性

電流オフセット値(SO)(0~1024 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。

このグラフからは、SO 値を大きく(740 以上)した場合は、センサ測定値は 0 に張り付く事が判ります。また、SO 値が小さいときは、タッチと非タッチの測定値の差分が小さい事も見て取れます。

できるだけ右側(SO が大きな値)としたいが、動作限界近傍にすると経時変化等で動作しなくなってしまう事も考えられます。そのため、上図では 600 あたりを選ぶのが良いかと考えます。

※どの動作点を選ぶかは考え方次第であると思います。SO 値は大きく取った方がタッチと・非タッチの差分が大きく取れるが、大きくし過ぎるとそもそも測定値が取れなくなるという事を念頭に、値を決めればよいと思います。

本サンプルプログラムでは、SO=0 の非タッチ時のセンサ測定値の 20%(ctsu.h 内で定数定義)を下回らない点とします。

非タッチ時, SO=0 のセンサ測定値が、13,000 程度なので、その 20%である 2,600 のあたり、SO=640 を電流オフセット値とします。

※オフセット値はマイコンチップの個体差があるので、お手元のボードでは 640 ぐらいにはならない事もあり得ます。

本サンプルプログラムでは、

src¥ctsu¥ctsu.h

```
//電流オフセット最適化
#define CTSU_OFFSET_MERGIN (0.20f)    //SO=0 のセンス値の 20%の点で動作させる
```

上記ファイル内でこの割合値を定義しており、初期値は 0.2(20%)です。

タッチの感度を上げたい場合は、この値を小さくする（但し誤作動の確率は上がる）。感度を下げたいときは、大きくすると良いと思います。設定可能なのは、0~1 の範囲内です。

電流オフセット調整レジスタ値(SO)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。

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測定値を取得し、非タッチ／タッチ時の中間のセンサ測定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。

閾値の初期値は、非タッチ時のセンサ測定値に対して、センサ測定値が 10%増加した点としています。

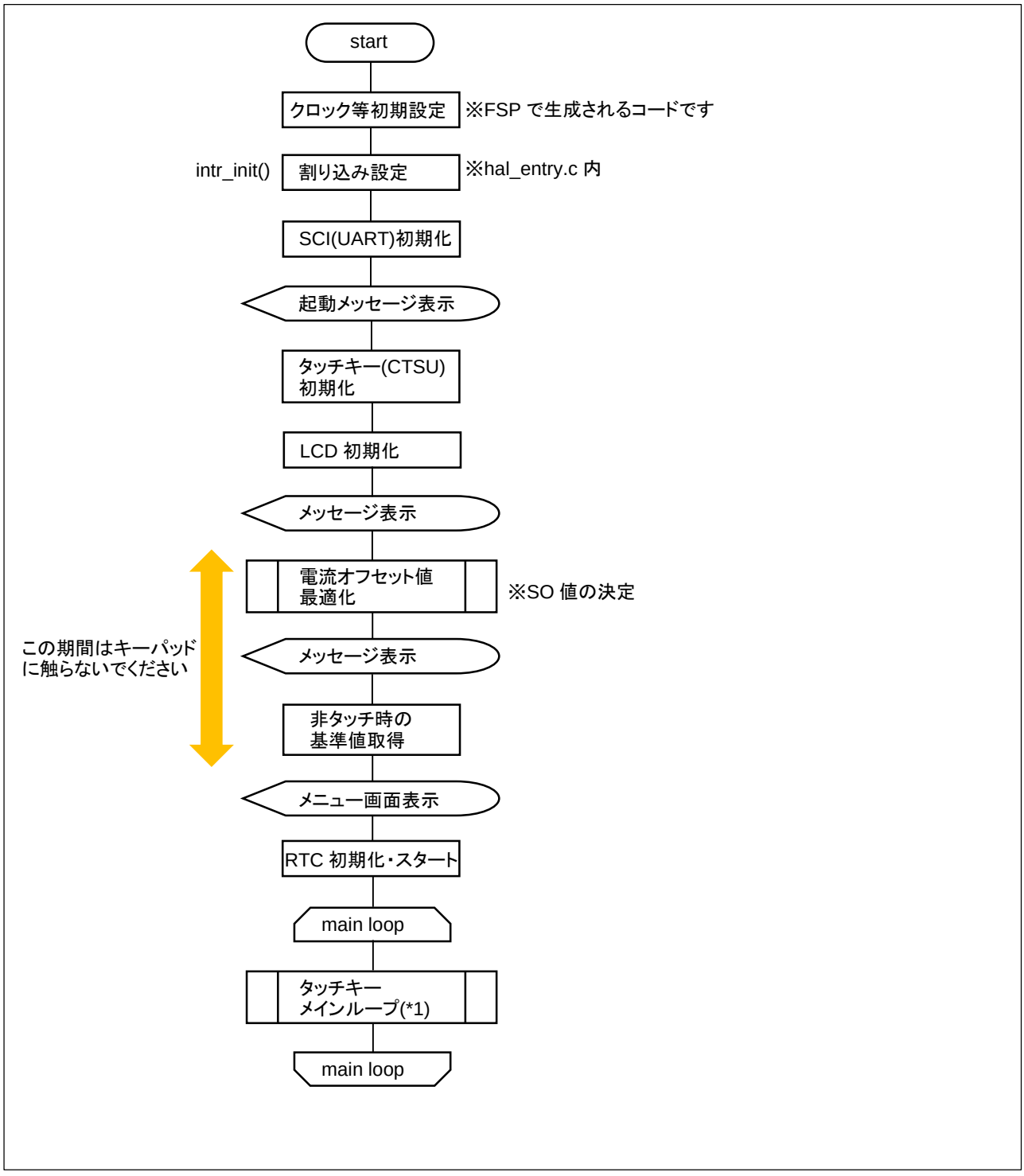
src¥ctsu¥ctsu.h

```
//判定閾値（初期値）
#define CTSU_DEFAULT_THRESHOLD (1.1f)
```

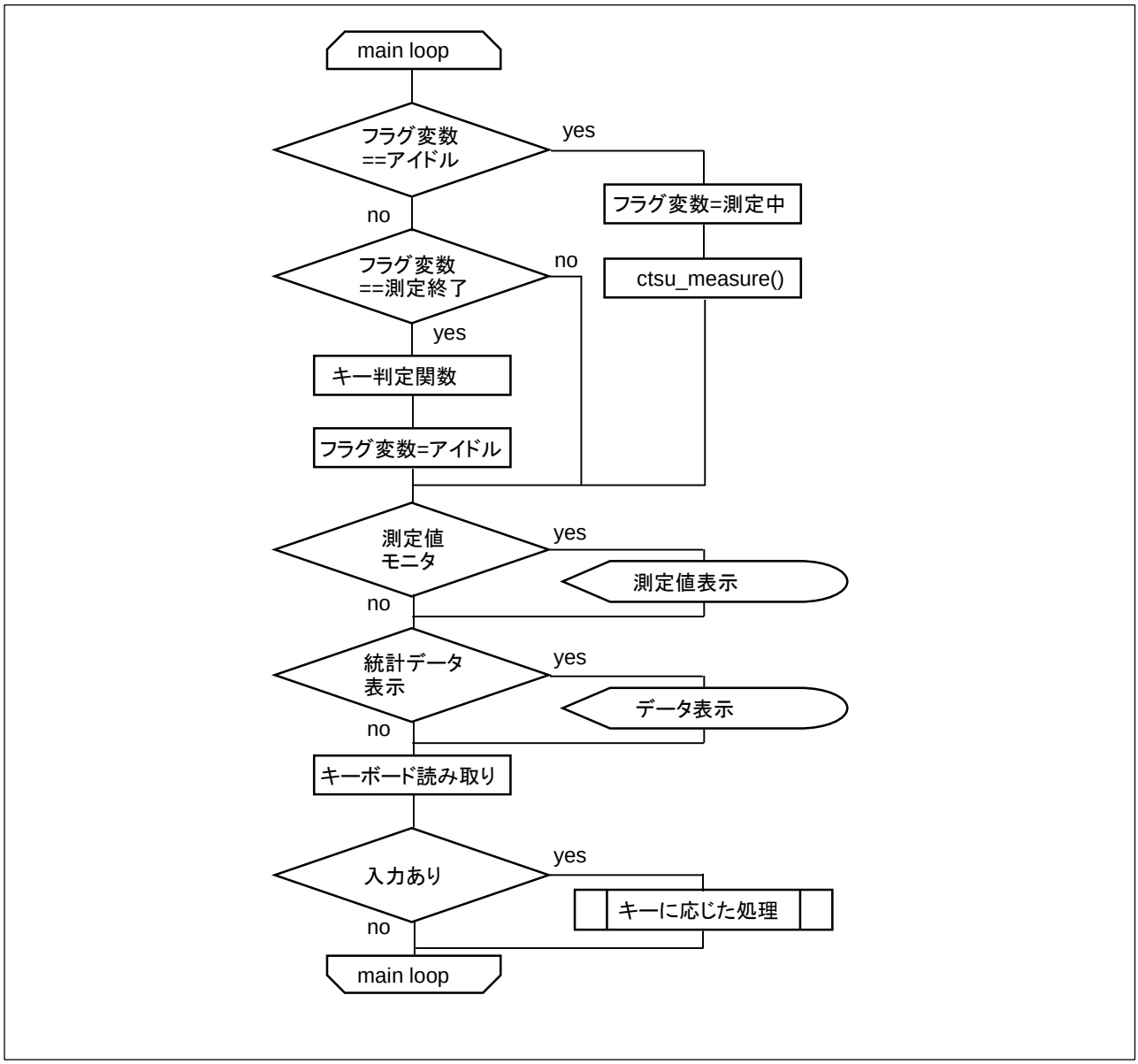
t コマンドで、閾値の最適化が行えますので、t コマンドで取得した閾値を初期値にしてしまっても良いかと考えます。

プログラム内では無限ループで、センサ測定値(CTSUSC)を取得して、非タッチ時のセンサ測定値(CTSUSC)と比較を行い、タッチの判定を行います。

2.4. フローチャート

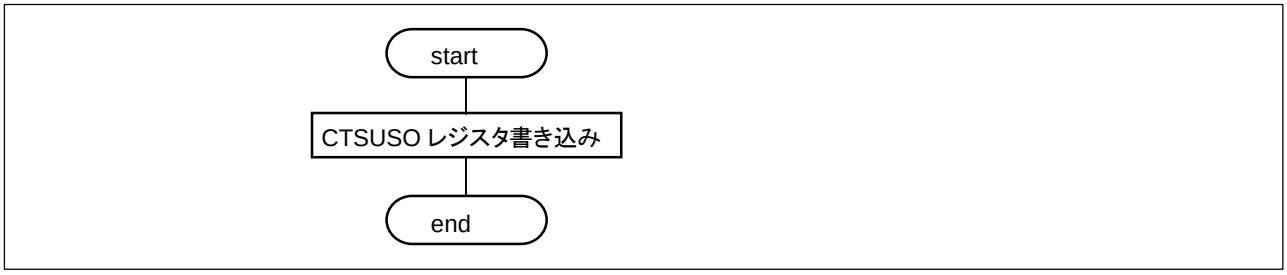

start
クロック等初期設定
※FSP で生成されるコードです
intr_init()
割り込み設定
※hal_entry.c 内
SCI(UART)初期化
起動メッセージ表示
タッチキー(CTSU)
初期化
LCD 初期化
メッセージ表示
電流オフセット値
最適化
※SO 値の決定
この期間はキーパッド
に触らないでください
メッセージ表示
非タッチ時の
基準値取得
メニュー画面表示
RTC 初期化・スタート
main loop
タッチキー
メインループ(*1)
main loop

ータッチキーメインループ(*1)　フローチャートー



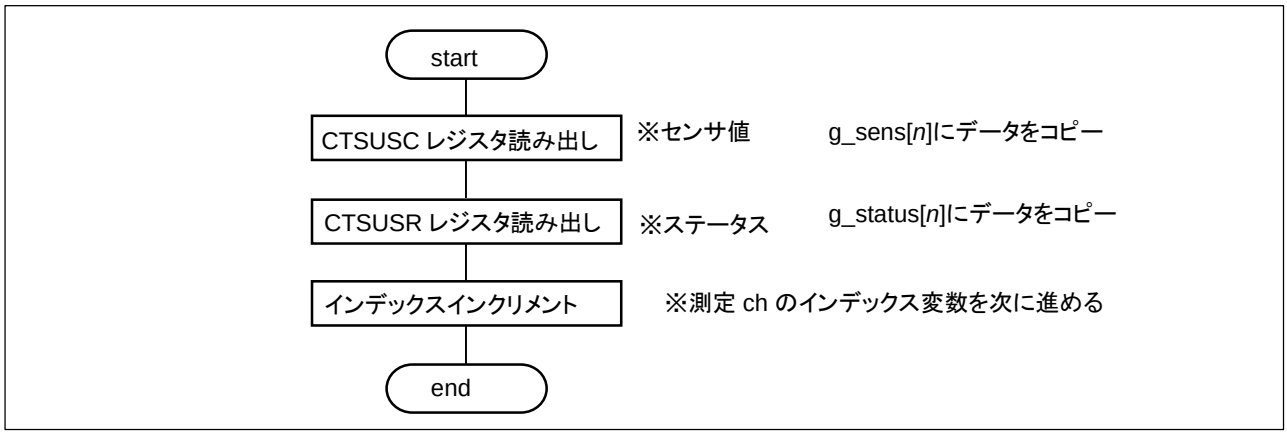
－CTSU 割り込み処理　フローチャート－

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



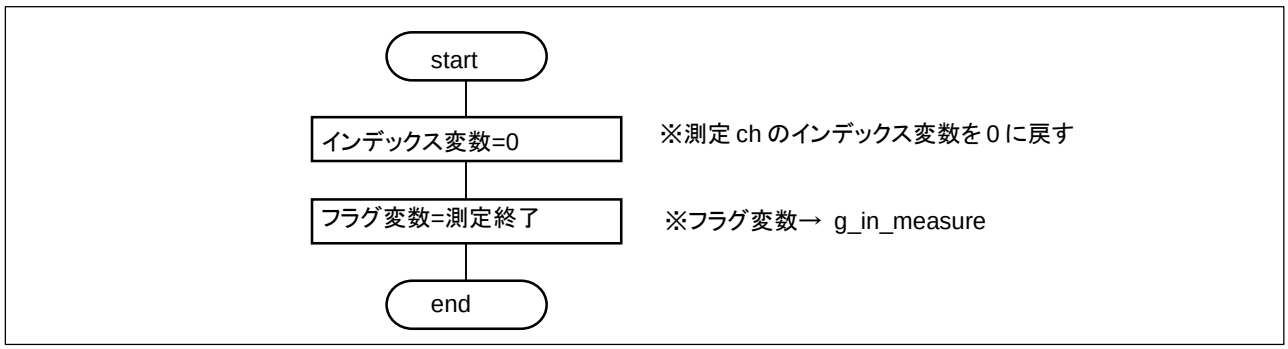
CTSU WRITE 割り込みでは、電流設定値等のレジスタにデータをセットする処理を行っています。CTSU の処理をスタートさせると、この割り込みが入りますので、そのタイミングで上記レジスタに値を書き込みます。
（CTSUSO レジスタに値が書き込まれると、実際のセンサ値の測定に移ります）

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



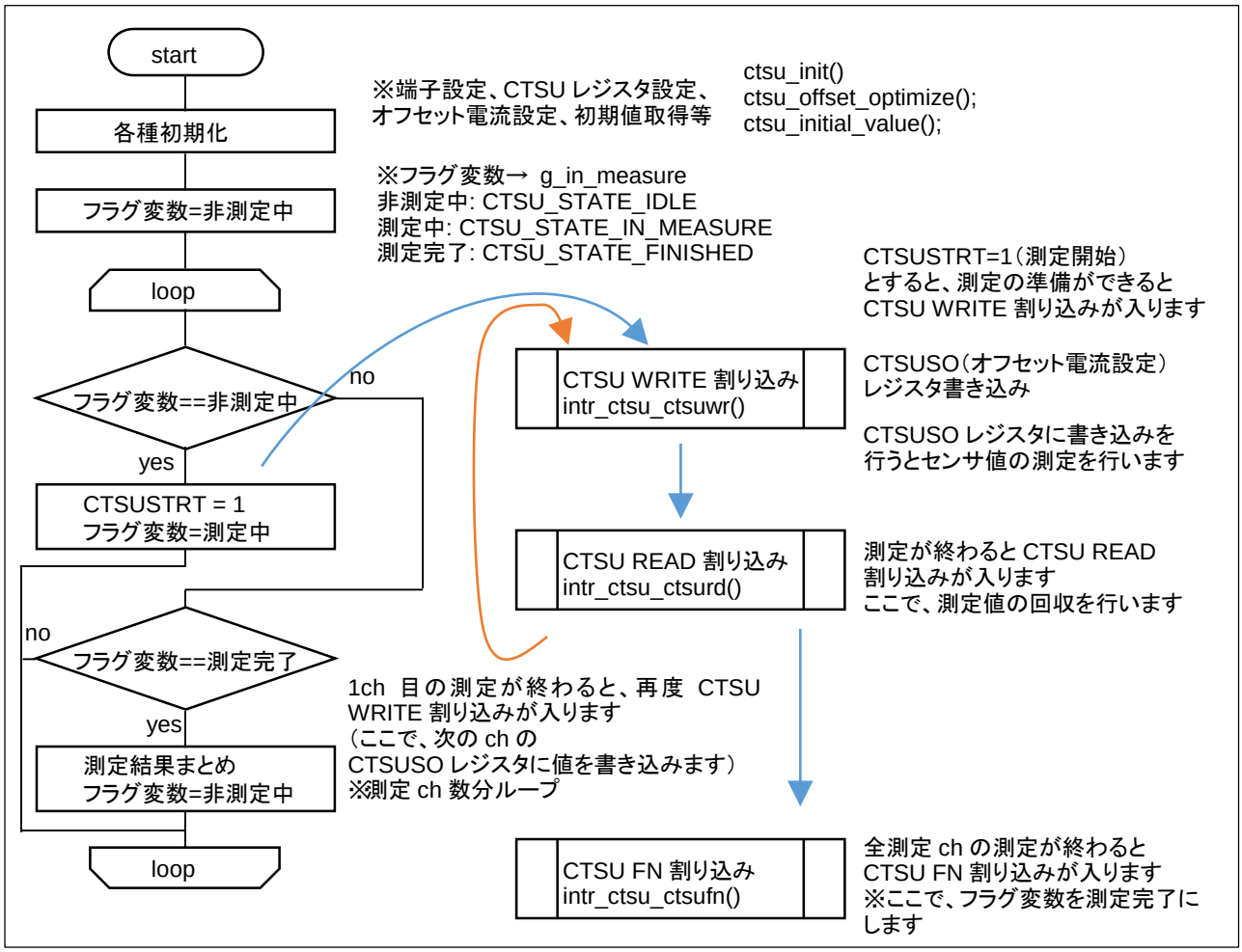
CTSU RD 割り込みは、1ch（1 つのキー）の測定が終わった段階で呼び出される割り込みです。測定値を、グローバル変数にコピーする処理を行っています。
※CTSU WRITE, CTSU READ 割り込みでは、（処理が判り易い様に）直接レジスタへの書き込み、読み出しを行っていますが、ここで使用しているレジスタアクセスは DTC が使用可能です。（実際のアプリケーションプログラムでは、DTC の使用もご検討ください。）

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



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。

タッチキー機能(CTSU)使用時のセンサ値測定のフローは、以下の様になります。



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、CTSUSO(オフセット電流設定レジスタ)に、値を書き込みます。

※CTSUSO 設定値(SO 値)は、測定 ch 毎に最適化されている事が望ましいです

CTSUSO レジスタに値を書き込むと、(端子容量の)センサ測定値が測定され、1ch(1 つのキー)の測定が終わると、CTSU READ 割り込みが入ります。ここでは、センサ値の測定値(CUSUSC)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）

CTSUSC 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 CTSUSO 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

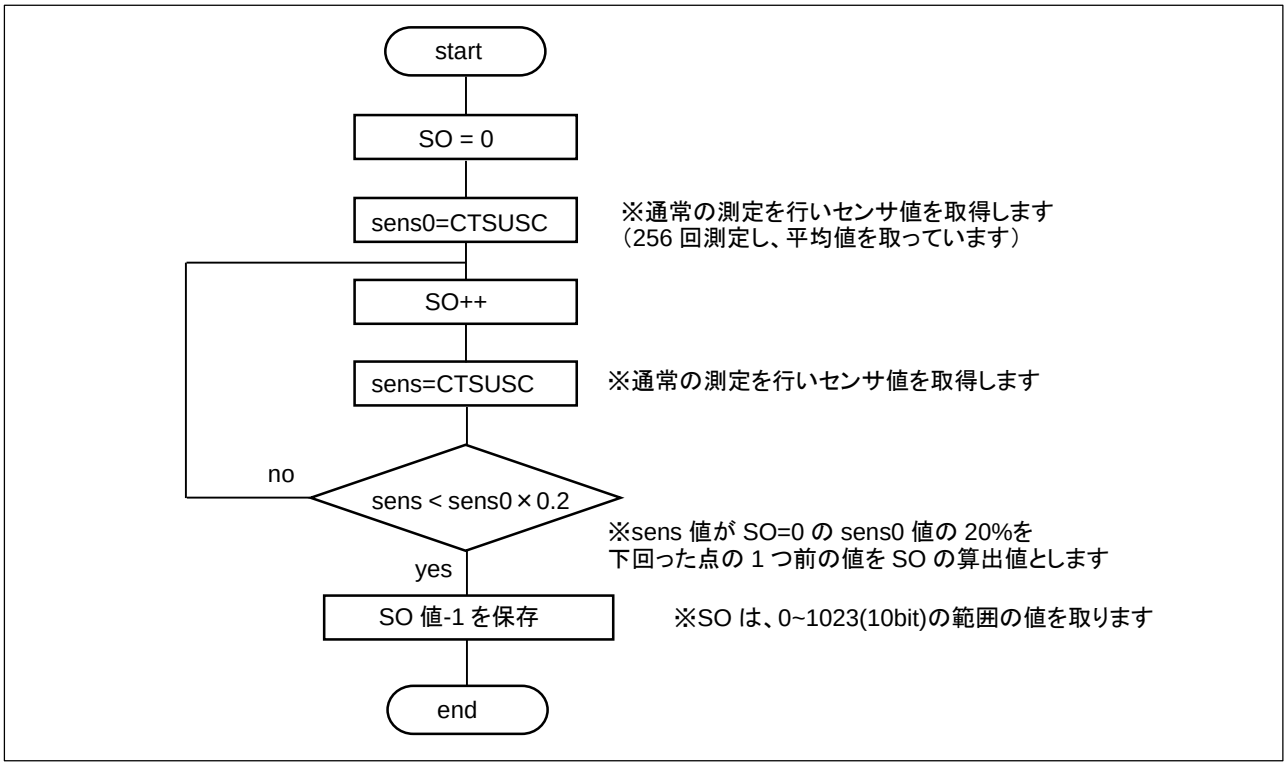
最後の測定 ch まで一連の処理が終わると、CTSU FN 割り込みが入ります。

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

電流オフセット値(SO)は、非タッチ時の測定値をベースに最適化しています。

・電流オフセット値の最適化フロー ctsu_offset_optimize()



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 SO 値を算出しています。

2.5. CTSU 関連の関数

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 CTSUSO 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), ステータスの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_offset_optimize

概要:電流オフセット値(SO)の最適化関数
宣言:
　　void ctsu_offset_optimize(void);
説明:
　・SO 値の最適値の算出
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[n]が、押しているキー0x01, 押されていないキー0x00 となります。キー数(3)
　　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。

・割り込み関数

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 SO 設定)の呼び出し、セット)
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

2.6. CTSU 関連のグローバル変数

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（3）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（3）の配列変数で、メインループ内で随時更新されます。

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 long 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 long g_ctsu_ctsuso[ ]

測定 ch 毎の、CTSUSO(SO)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。

float g_touch_threshold[ ]

タッチ、非タッチの閾値を示す変数です。初期値 1.1 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

2.7. CTSU 関連の定数値

```
#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 (35)
```

キー等を、TS??の名称でアクセスするための定義値。

#define TS_MAX (35)

TS 端子最大数。

#define CTSU_VALID_TERMINALS (3)

TS??で使用している端子数。

#define CTSU_CH_NUM (3)

測定 ch 数。自己容量タイプの場合は、=キーパッド数となります。

#define CTSU_OFFSET_MERGIN (0.20f)

SO=0 のセンサ測定値の 20%を下回らない点を SO（電流オフセット）の設定値とします。

#define CTSU_CTSUSO_SO (140)

電流オフセット（SO 値）の最適化を省略した際に使用される SO 値です。（通常は未使用です）

#define CTSU_DEFAULT_THRESHOLD (1.1f)

判定閾値の初期値です。測定値が 10%以上増加した際、タッチしているとみなされます。"t"コマンドで最適化（値の更新）が行われます。

#define CTSU_MONITOR_PERIOD (2)

"m"コマンドでの表示更新頻度。2 秒に 1 回。

//#define CTSU_OFFSET_OPTIMIZE_OMIT

本定数定義時は、起動後のオフセット電流(SO)は、固定値とします。

#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 DEBUG_PORT

定義時は測定終了毎に、ポート(P400)を反転させます。

2.8. まとめ

本チュートリアルは、RA2L1 が持つタッチキー(CTSU2)の機能を扱いました。前半は、ミドルウェアを使った場合。後半は、スクラッチ（ミドルウェアを使わずにプログラムを書き下した場合）です。

タッチキーは、実際にタッチしなくても、手を近づけるだけでもタッチ判定が可能です。（ボタンのパターンの作り方や感度調整にも拠ります。）

物理的なスイッチを持たず、触れたり手をかざすことにより反応する機器は、身近にも結構ありますが、その中身は本チュートリアルで紹介しているものかと思います（容量を測定して、容量値の変化から判定を行っている）。

RA2L1 の持つ CTSU2 の機能を使うと、タッチの判定が比較的簡単に精度良く行えますので、タッチで動作するアプリケーションを作成する場合は、専用の IC 等を使うことなく、RA2L1 のマイコンのみで作る事を検討してみてください。

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	2021.7.6	—	初版発行

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 RA マイコン搭載
HSB シリーズマイコンボード 評価キット

SmartRA 学習キット
チュートリアル 8

株式会社 北斗電子

 2021 年 7 月 6 日改訂 REV.1.0.0.0 (210706)