RL78 Family

Using QE and SIS to Develop Capacitive Touch Applications

Introduction

This document will demonstrate the needed steps to create an application example that integrates capacitive touch sensing using Renesas RL78 Microcontrollers.

Target Device

RL78 family with Capacitive Sensing Unit (CTSU)

Contents

1. Application Example Overview

This document will demonstrate how to implement capacitive touch sensing using Renesas RL78 Microcontrollers. Using these steps, the user will be shown the process of connecting the RL78/G23 MCU board, using the Smart Configurator for project creation, and QE for Capacitive Touch to create, tune and monitor a capacitive touch sensing project.

2. Related Documents

This application example is intended to give a user a short introduction to creating a working RL78 capacitive touch sensing project. A thorough review of all the applicable documentation for the e^2 studio IDE/ Smart Configurator, Software Integration System (SIS) drivers/middleware, Renesas Code Generator, and QE for Capacitive Touch plug-in help (contained within the e^2 studio IDE help index) is strongly suggested to answer any questions or for more details on usage of any of the tools utilized in this application example.

3. High Level Integration Steps

The following high-level steps will give the reader an overview of the steps required integrate touch detection into this project. These same steps should apply to any typical user development application.

- Create the initial project using the e^2 studio project creation wizard
- Use the Smart Configurator to add the required modules to the created e^2 studio project
- Use the QE for Capacitive Touch e^2 studio plug-in to create the capacitive touch interface
- Use the QE for Capacitive Touch e^2 studio plug-in to tune the application project
- Add the needed SIS application code function calls to the user project to enable capacitive touch sensing operations in the application project
- Monitor the application project using QE for Capacitive Touch e^2 studio plug-in to demonstrate capacitive touch sensing detection

4. Required Development Tools and Software Components

The project utilizes the following development environment:

- RL78/G23 Capacitive Touch Evaluation System (RTK0EG0030S01001BJ)
  - Microcontroller used: RL78/G23 (R7F100GSN2DFB)
- Renesas e^2 studio IDE, V2021-07 (21.7.0) or later
  - Renesas Software Integration System (SIS) version: 1.0.0 or later and Renesas Code Generator
  - Renesas QE for Capacitive Touch e^2 studio plug-in, version: 2.0.0 or later
- Compiler: Renesas CCRL v1.10.00
- Emulator: Renesas E2 emulator Lite (RTE0T0002LKCE00000R)

5. Application Example Overview

In the main loop of the application example, the following processing is performed:

- The global flag used to determine if the **rm_touch** middleware is ready to be processed is checked
    - If the flag is set (ready to process)
        - The global flag is reset to 0
        - A call to the **rm_touch** middleware processes data from the previous completed scan, updates needed data, then starts the next touch scan process
        - A call to the **rm_touch** middleware populates a user created global variable with the binary determination of a touch on the sensor board

A code listing of the completed application example is in “16. qe_touch_sample.c Listing” for review.

6. Project Creation

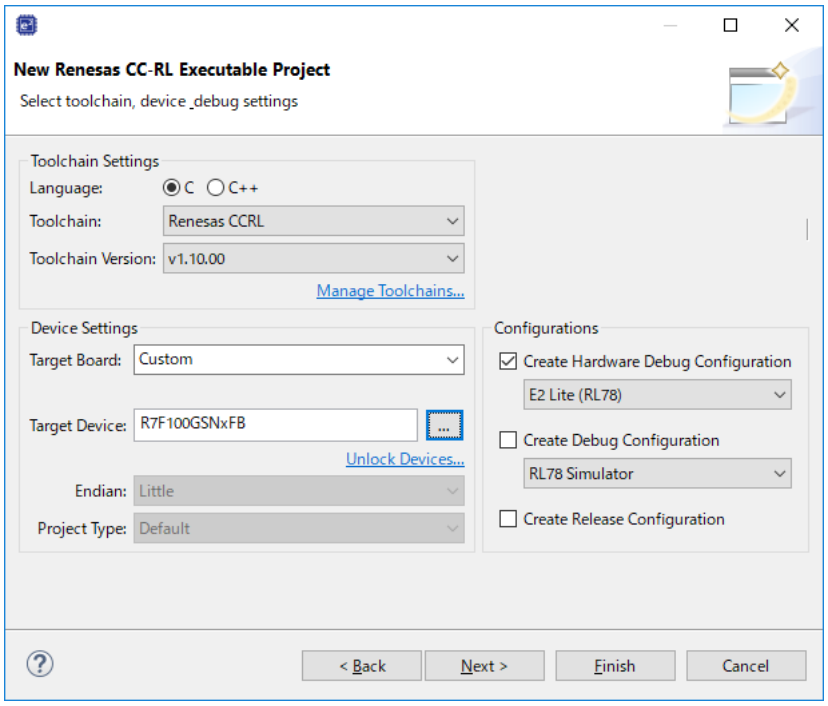
1. On the PC start the IDE using the Windows->Start menu or the icon on the desktop. When the dialog appears, create the Workspace at anywhere

2. Start a new project by clicking File->New->**C/C++ Project**

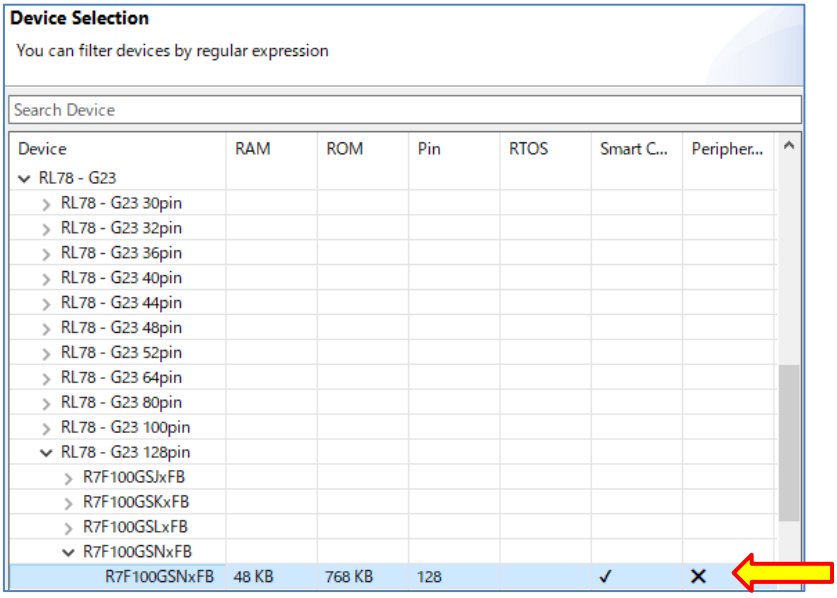
3. At the dialog box that opens, select “**Renesas RL78**” and highlight with a single-click “**Renesas CC-RL C Executable Project**”, then click **Next**.

4. In the next dialog box, enter a Project name-this can be any name desired. The example here uses **Capacitive_Touch_Project_Example**. When complete, click **Next**.

5. In the next dialog box, ensure the following is selected:
   - Language: C
   - Toolchain: Renesas CCRL
   - Toolchain Version: v1.10.00
   - Target Board: Custom
   - Target Device: R7F100GSNxFB
   - Create Hardware Debug Configuration is checked.
     E2 Lite (RL78) is selected in the pull-down.



Note: Use the '…' to select the proper device using the menu that appears



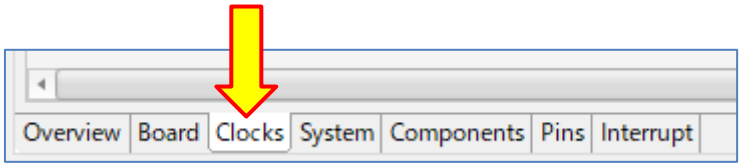
6. Once complete, click **Next**.

7. In the next dialog box that appears, check the box for **Use Smart Configurator**, then click **Finish**.

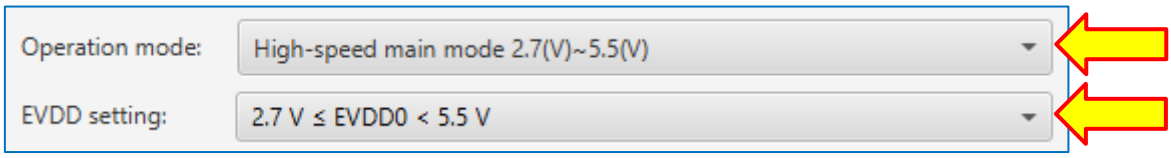
Once complete, a default window will open for the IDE with the Smart Configurator perspective open and ready for project configuration. This completes the project creation.

7. Using Smart Configurator to Add Modules

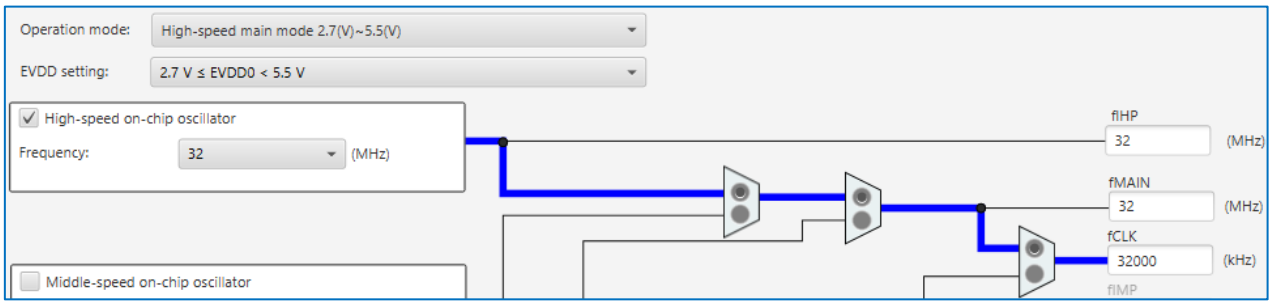
1. Using the tabs in the lower-middle pane of the IDE, select the **Clocks** tab to display the clock tree for the RL78 MCU.



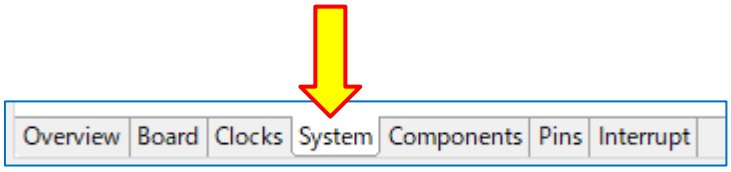
2. For this application example, the MCU V_{DD} will be used at 3.3 V. Select the **Operation mode:** and the **EVDD setting:** as shown in the image below.



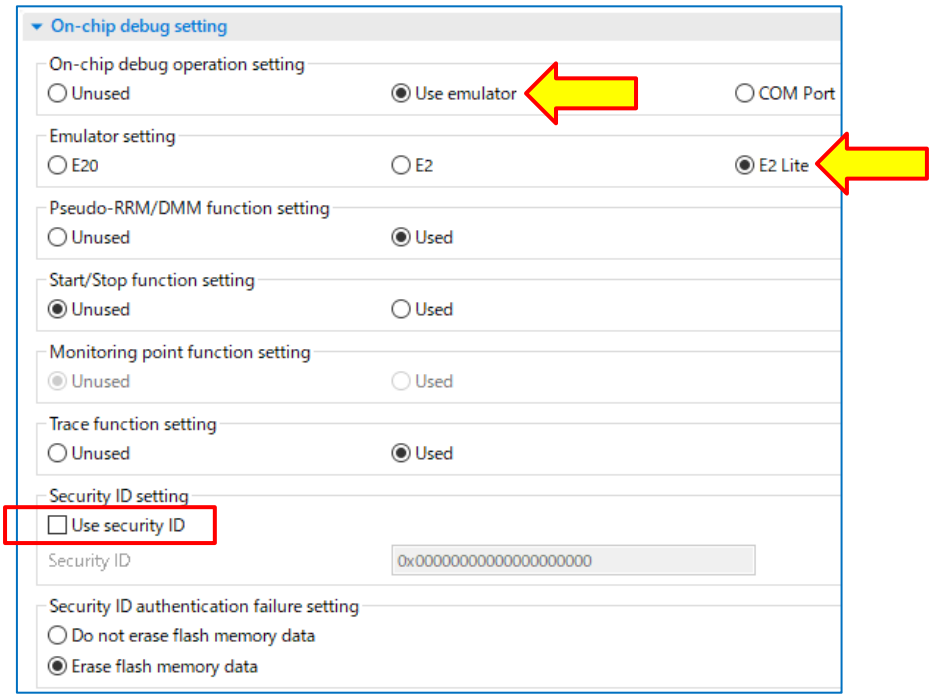
3. The setting of the clock configuration that is used for this application note is shown below.



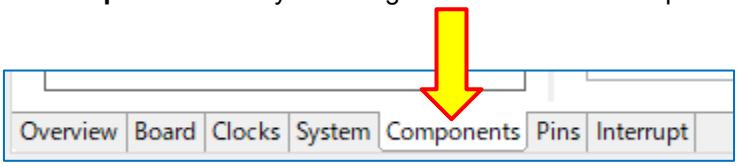
4. Next, move to the **System** tab by selecting it at the bottom of the pane.



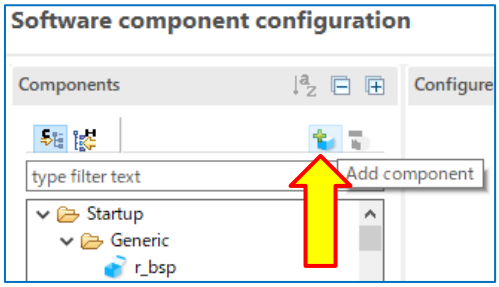
5. Select the **Use emulator** and the **E2 Lite**, deselect the **Use security ID** as shown below.



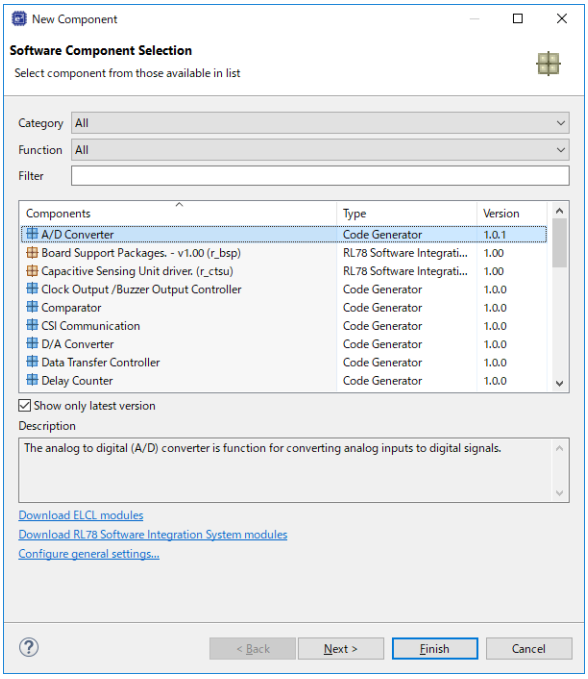
6. Then, move to the **Components** tab by selecting it at the bottom of the pane.



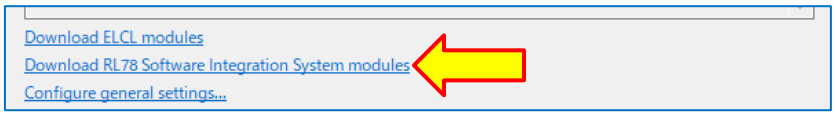
7. Modules now need to be added to the project for application use. Add a module by clicking on the '+' sign in the Components tab.

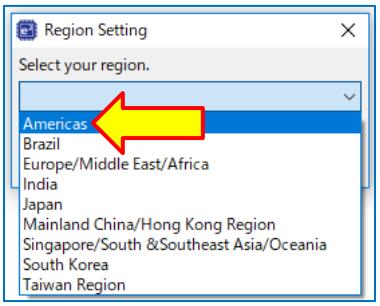


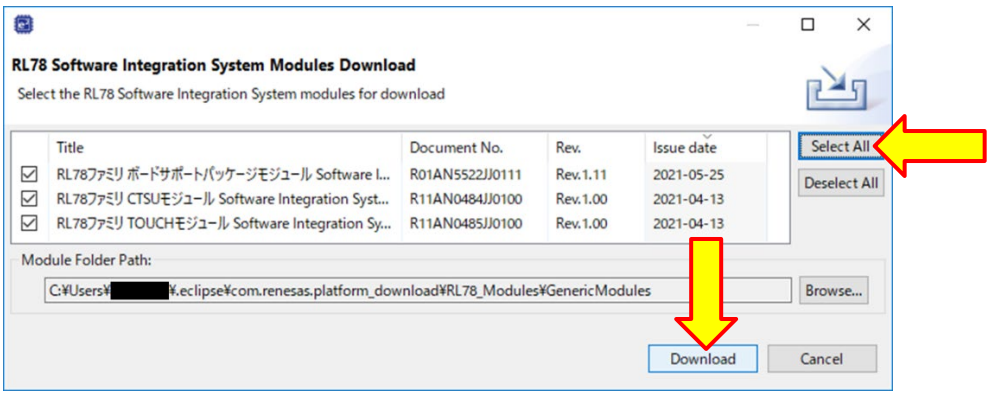
8. A new dialog window will open showing all the available modules that can be added into the project. It will appear similar to the below picture.



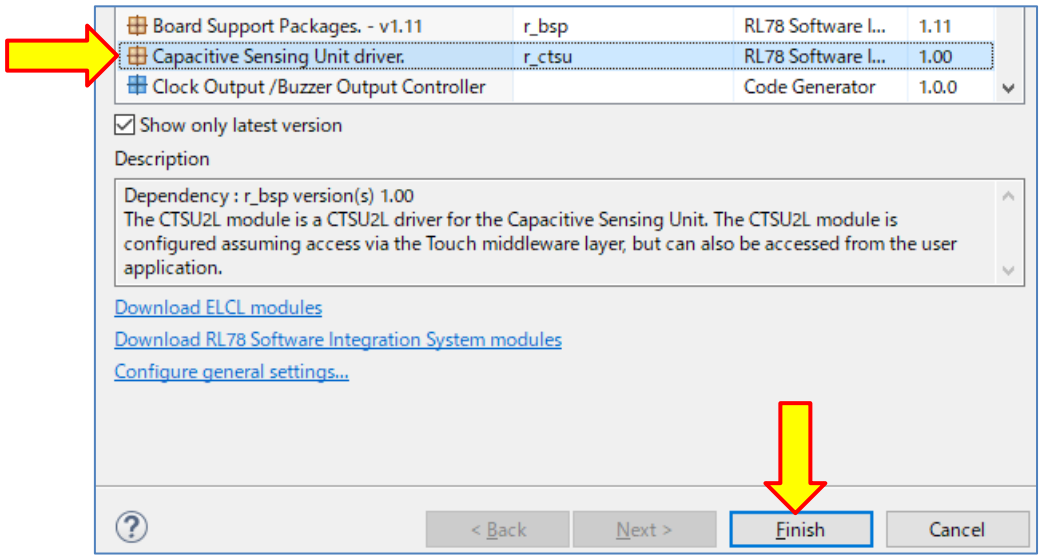
Note: When using SIS modules for the first time, download SIS modules as shown below.



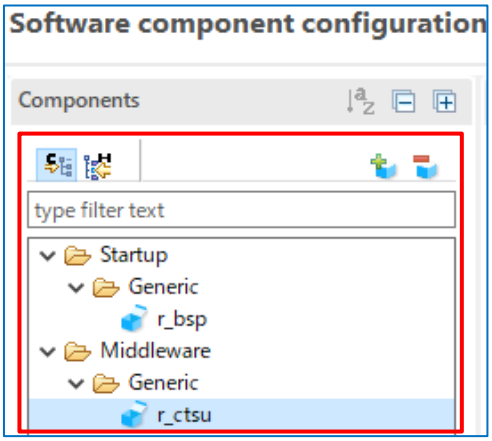




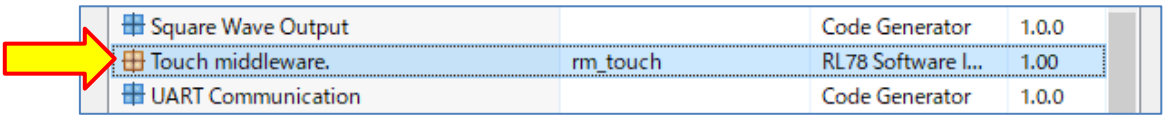
9. Single-click the **r_ctsu** module and click **Finish** in the lower part of that open dialog box.



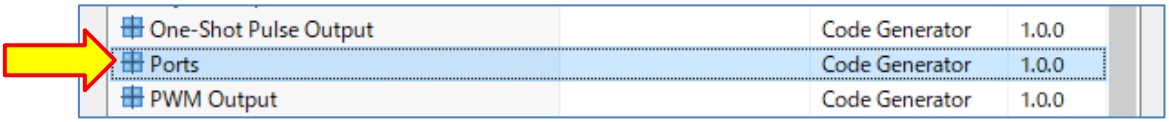
10. The module will appear in the software component configuration as shown in the picture below:



11. Next, add in the **rm_touch** module using the same procedure as was done for the **r_ctsu** module in the previous steps. Remember to click **Finish** to add the module.

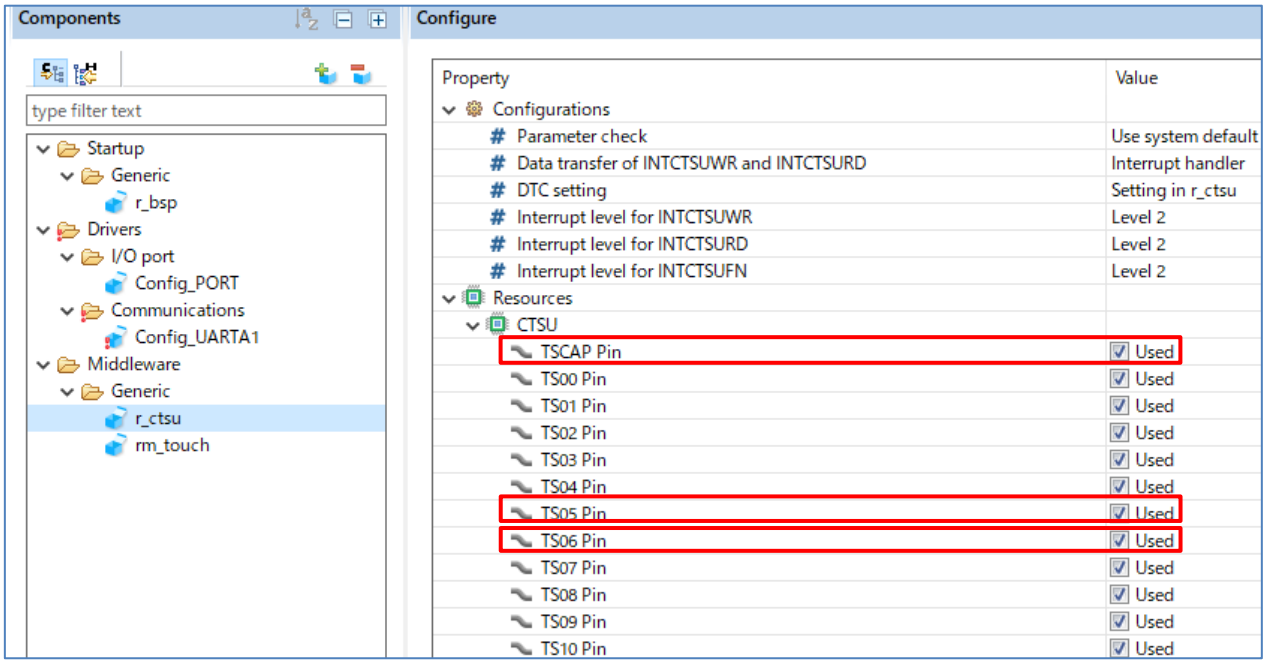


12. Finally, add the **Ports** module to the project. Remember to click **Finish** to add the module.



13. Next, enable the TSxx pins that you use. Select the **r_ctsu** module and in the Configure pane a list of pins associated with the module will appear. This application example will assign the two buttons (**TS05** and **TS06**). Click the following pins in the Configure pane so they can be used in the project:

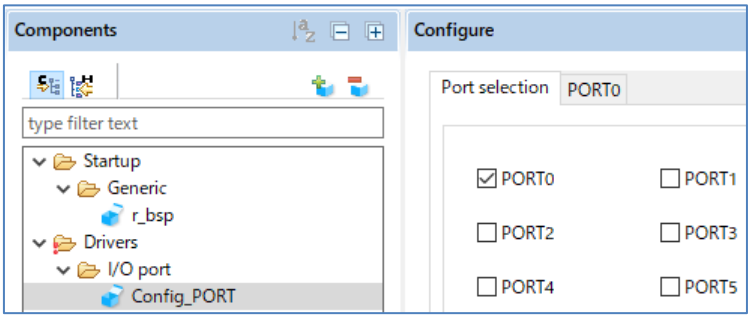
- TSCAP Pin
- TS05 Pin
- TS06 Pin



NOTE: The TSxx pins that are not being used by the touch application are recommended to be setup such that they are driven to low-level output. In the case of CTSU2, If the TSxx pin that is not being used in the touch application is enabled by checking "☐ Used", the TSxx pin is set to low-level output as non-measurement pin. When actually designing your circuit, make sure the design includes sufficient pin processing and meets electrical characteristic requirements.

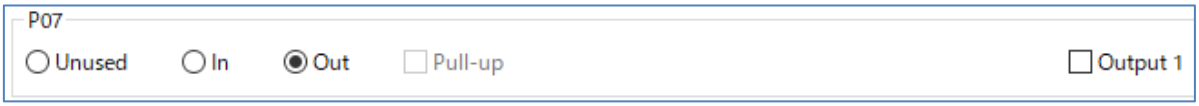
14. Next, the ports that are not being used by the application are recommended to be setup such that they are driven to low-level output at startup. Similar to selecting the used **r_ctsu** module in the previous steps, in the Components tab, select the **Config_PORT** module. Then click the checkbox for **PORT0**. This will cause these tabs to appear in the Configure tab as shown below.

    NOTE: Only one port is setup here as an example of the usage. When actually designing your circuit, make sure the design includes sufficient pin processing and meets electrical characteristic requirements.

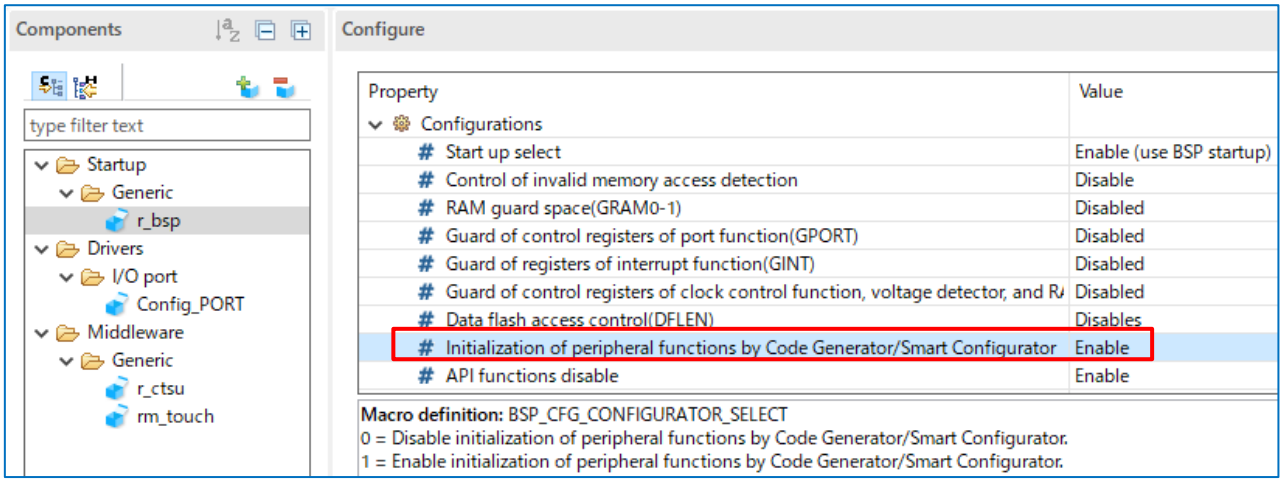


15. Select the **PORT0** tab and select **Out** for P07. The tab should look like the picture below.

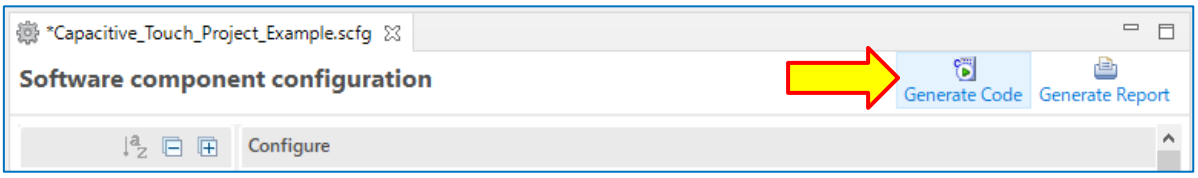
    NOTE: Only one port is setup here as an example of the usage. When actually designing your circuit, make sure the design includes sufficient pin processing and meets electrical characteristic requirements.



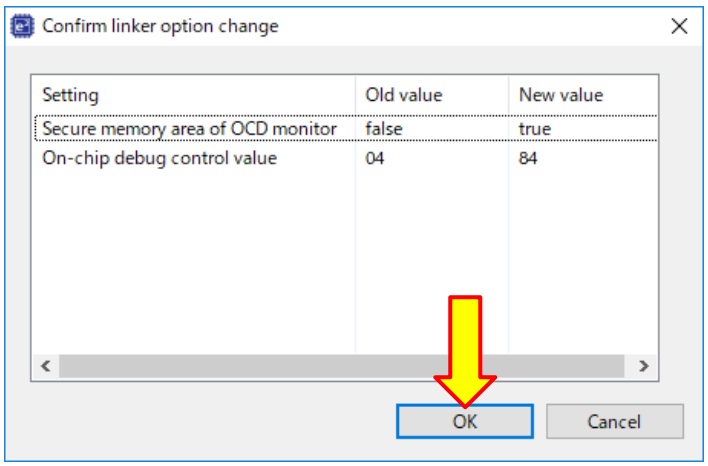
16. Ensure **Initialization of peripheral functions by Code Generator/Smart Configurator** is set to **Enable** as shown below.



17. At this point, all needed application modules for capacitive touch operations have been added. The final step is to generate the needed application code modules for the project. Do this by clicking the **Generate Code** icon in the upper-right of the Smart Configurator pane as shown in the picture below.



Note: If you change the on-chip debug setting or the option byte setting in the Smart Configurator, the following message will be displayed. After confirming the changes, click OK.



8. [Additional function] Setting the serial communication monitor using UART (1/3)

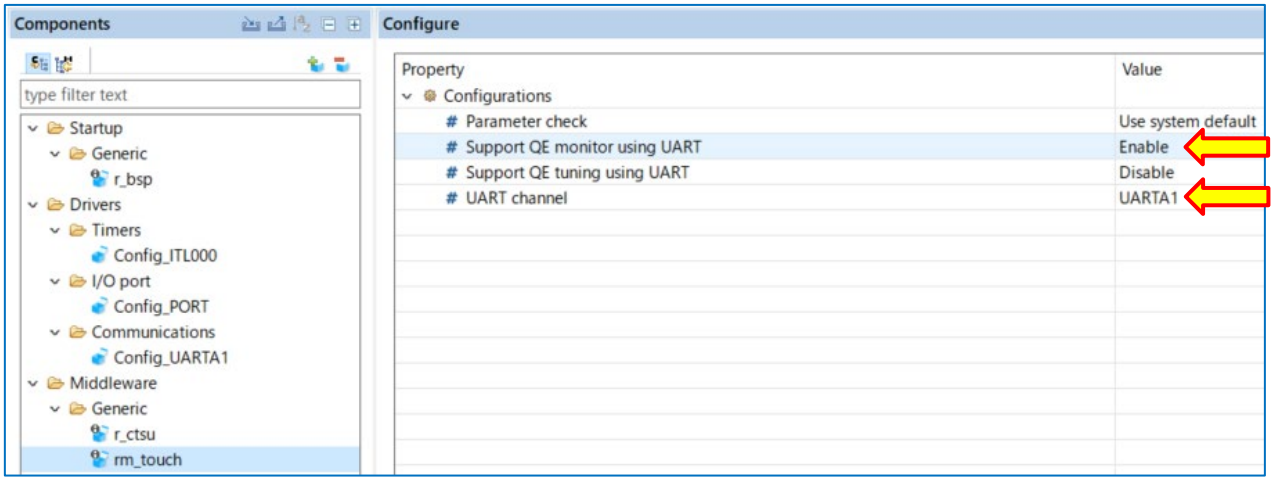
Note: Monitoring touch performance for touch applications can be confirmed by communication via the OCD (On-Chip Debugging) emulator. However, RL78 family case, monitoring performance is limited by the OCD function of the RL78 family.

On the other hand, monitoring touch performance can also be achieved via serial communication. Therefore, if you want to monitor smoothly, please add the monitoring function via serial communication.

Chapters 8, 13 and 15 (including this chapter) shown below describe setting the serial communication monitor using UART.

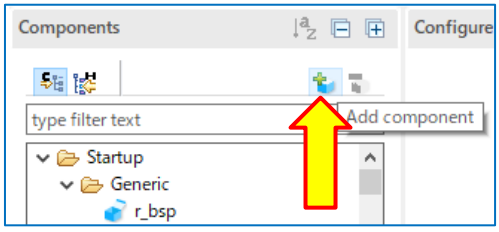
- "8. [Additional function] Setting the serial communication monitor using UART (1/3)"
- "13. [Additional function] Setting the serial communication monitor using UART (2/3)"
- "15. [Additional function] Setting the serial communication monitor using UART (3/3)"

1. In the Components tab, select the **rm_touch** module, set **Support QE monitor using UART** to **Enable** and **UART channel** to **UARTA1** as shown below.

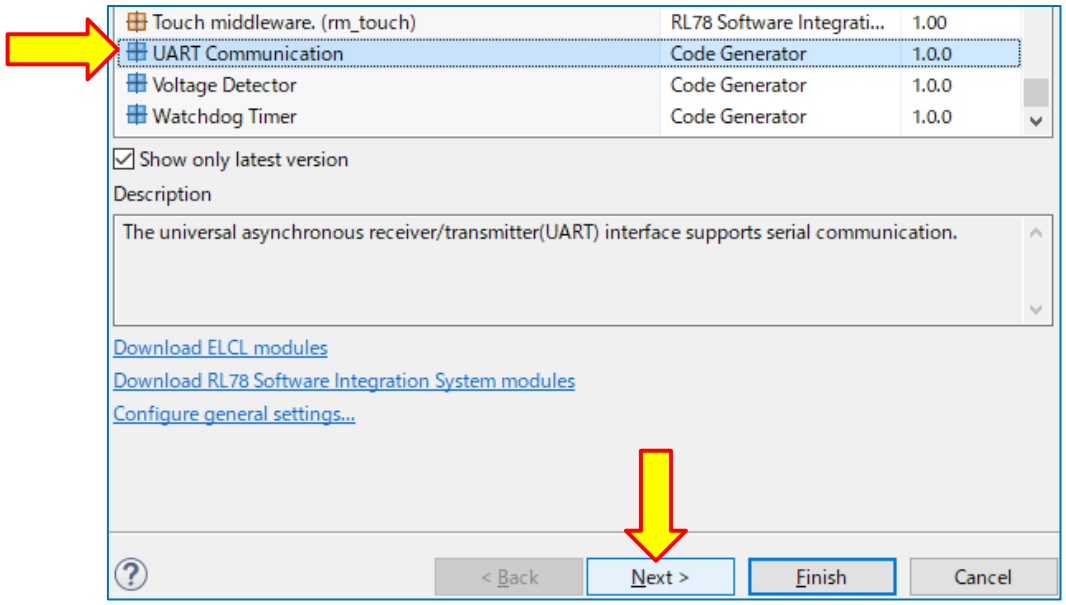


Note: The UART channels and ports used by this tool depend on your target board.

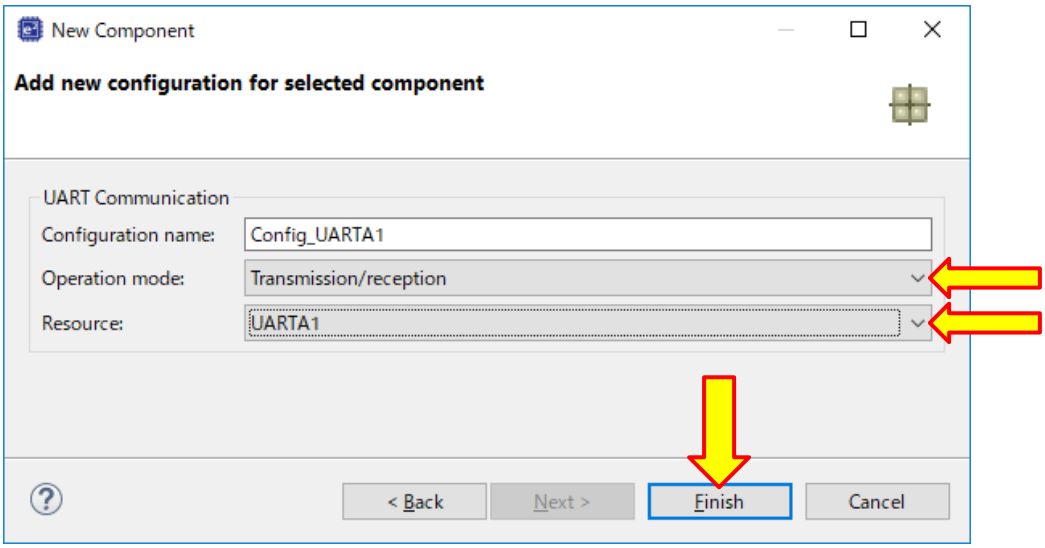
2. Add a module by clicking on the '+' sign in the Components tab.



3. Single-click the **UART Communication** module and click **Next** in the lower part of that open dialog box.

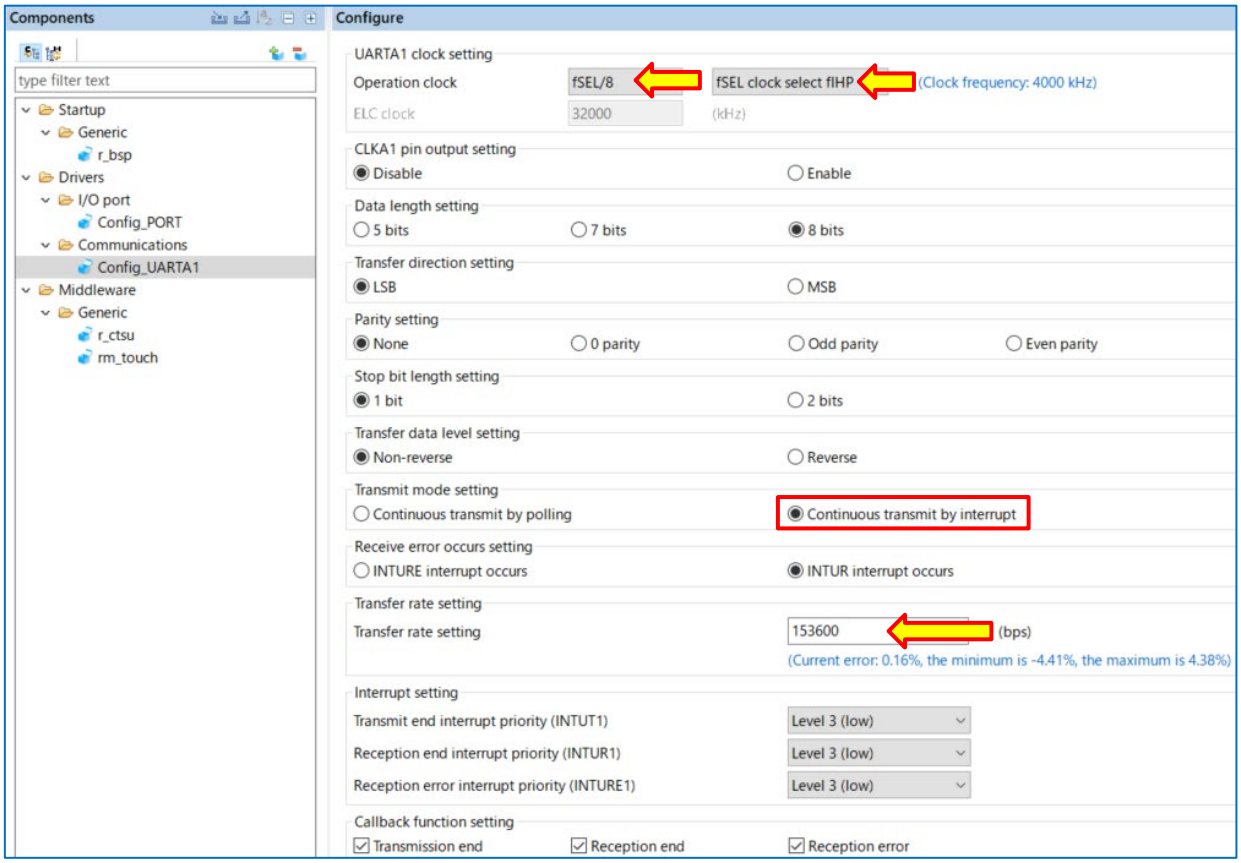


4. Set Operation mode to **Transmission/reception**, Resource to **UARTA1**, and click **Finish** in the lower part of that open dialog box.

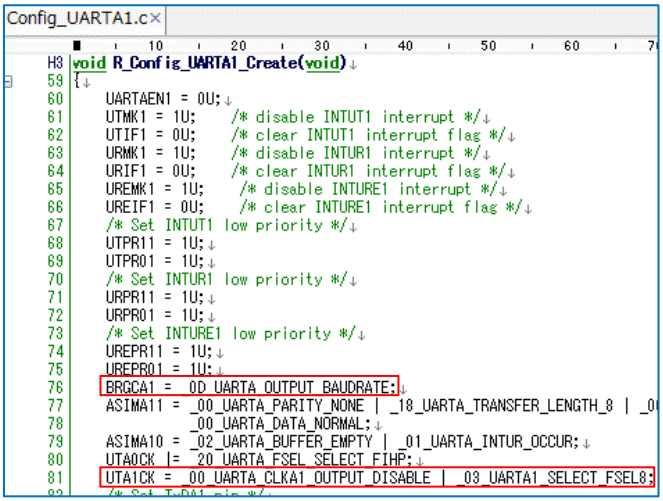


Note: The UART channels and ports used by this tool depend on your target board.

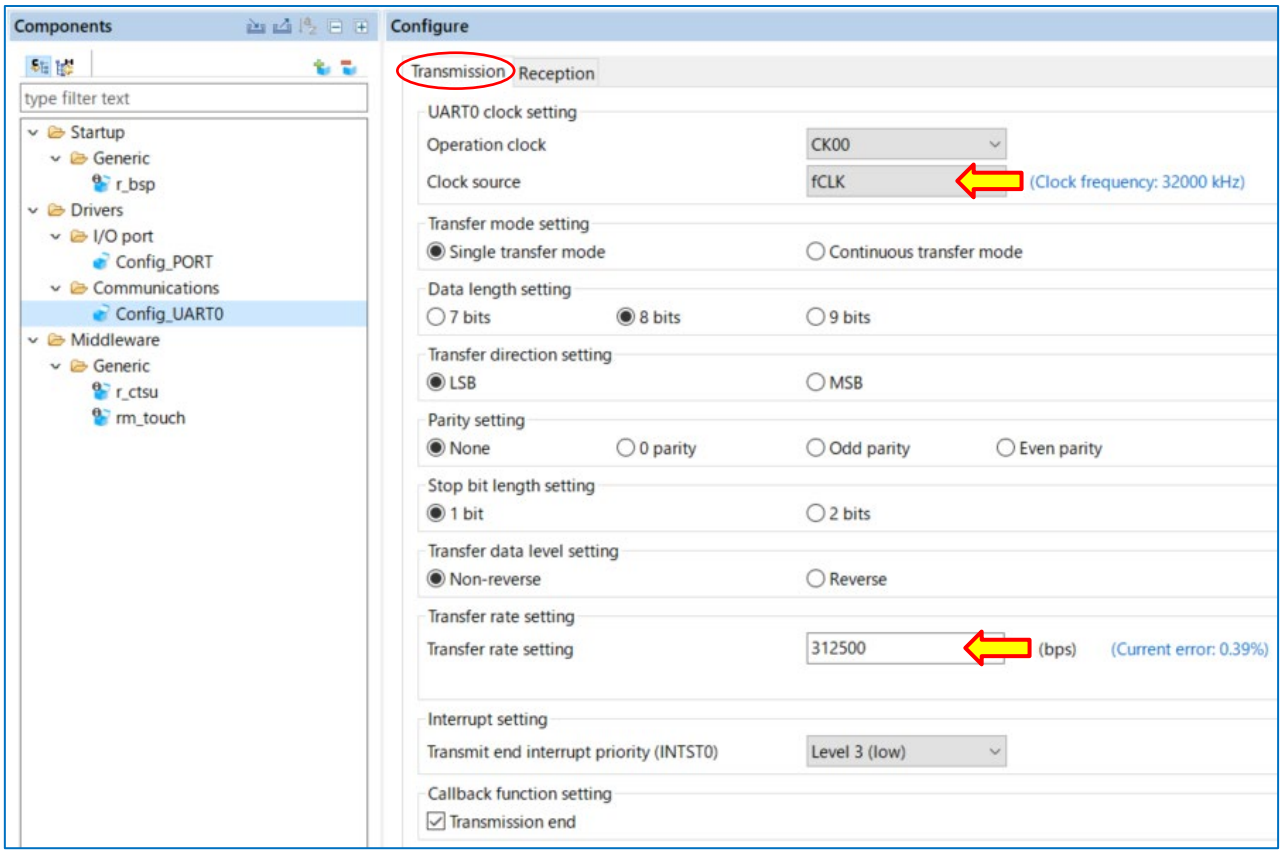
5. In the Components tab, select the **Config_UARTA1** module. Next, set the operation clock to **fSEL/8** and **fSEL clock select fIHP**, select **Continuous transmit by interrupt**, and the transfer rate setting to **153600** (bps).

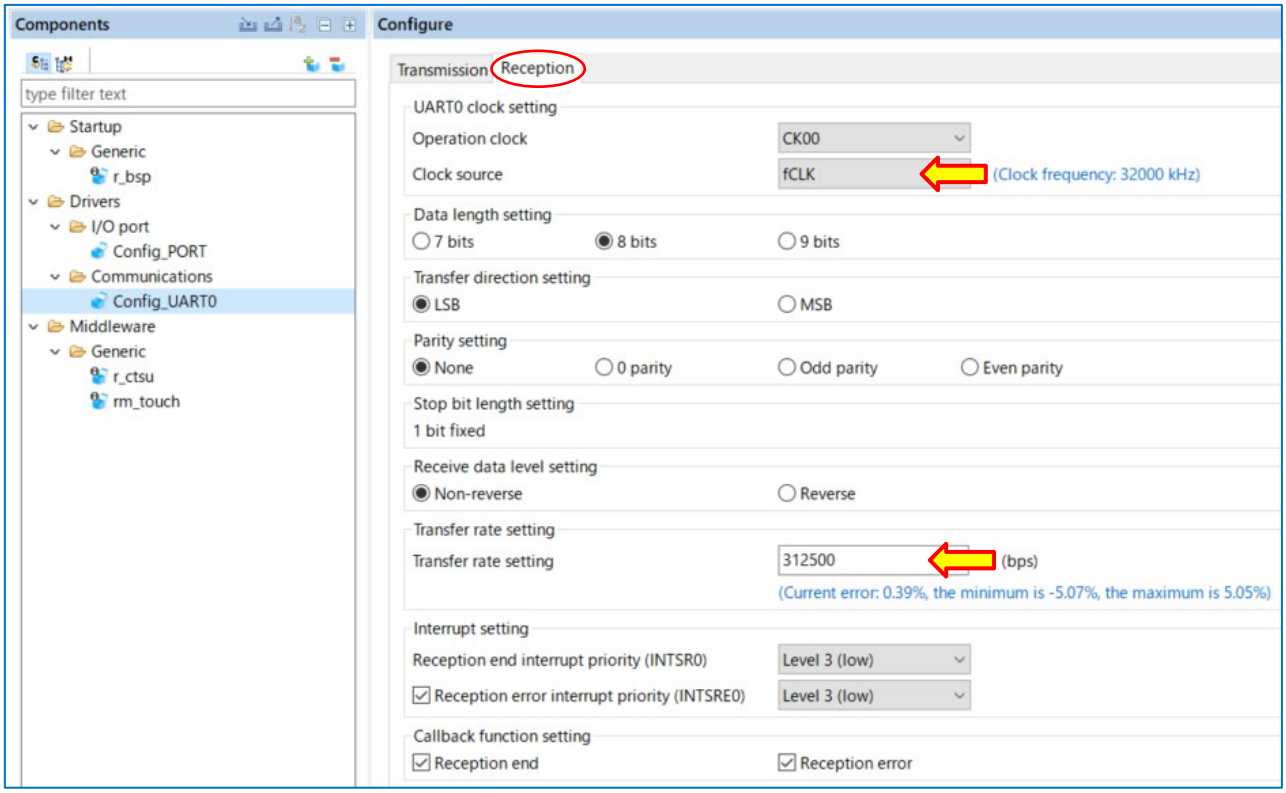


Note1: Depending on the version of the Smart Configurator, an error may occur regarding the transfer rate setting. If this error occurs, change the program in the red frame below to set the transfer rate after generating the code.

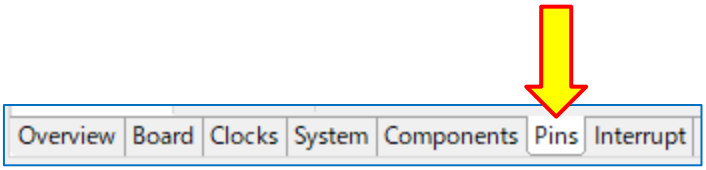


Note2: The UART channels and ports used by this tool depend on your target board. For example, when using UART0, the settings are different as shown in the image below.

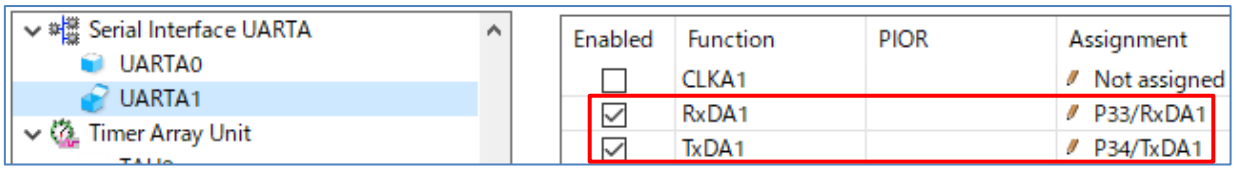




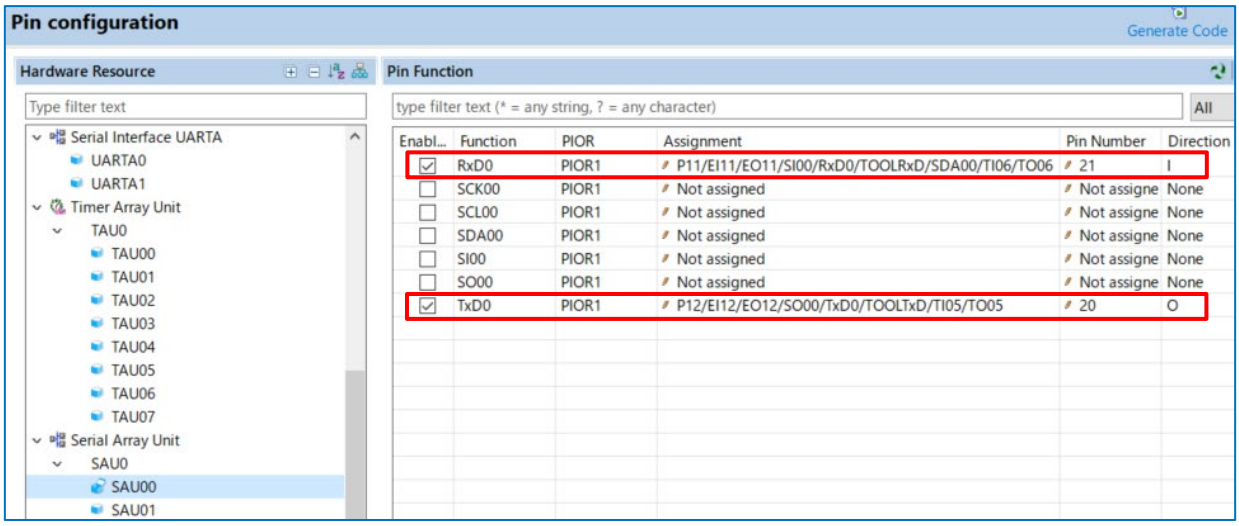
6. Move to the **Pins** tab by selecting it at the bottom of the pane.



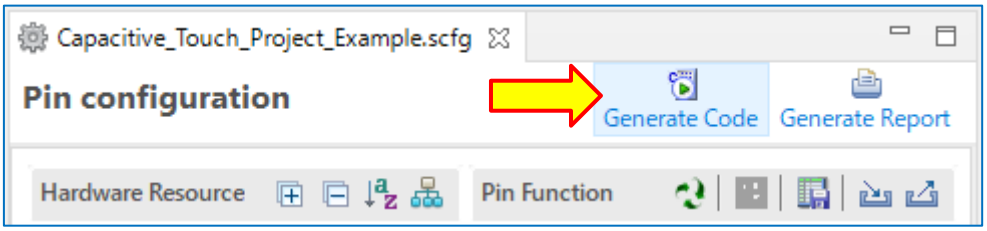
7. Assign RxDA1 function to P33 and TxDA1 function to P34.



Note: The UART channels and ports used by this tool depend on your target board. For example, when using UART0, the settings are different as shown in the image below.

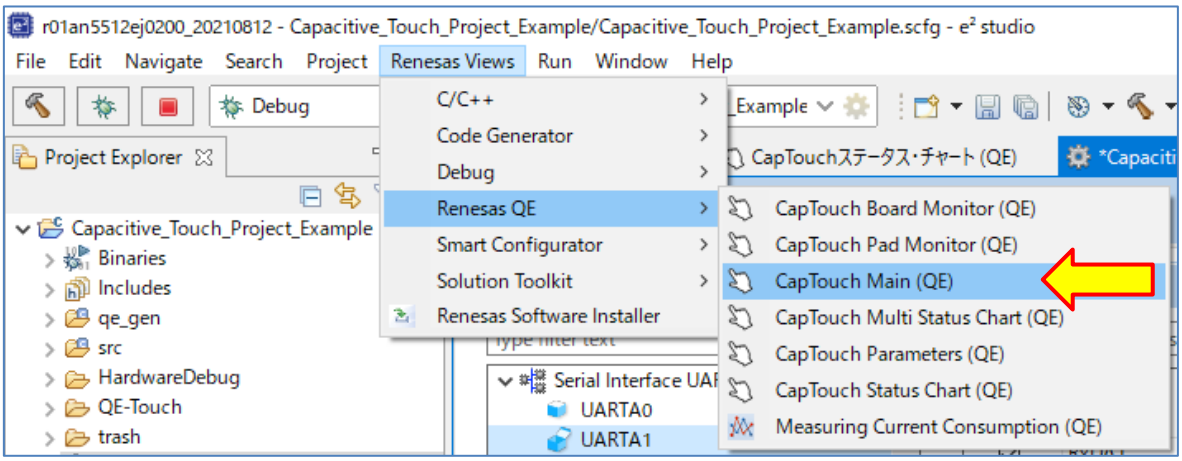


8. Generate the needed application code modules for the project. Do this by clicking the **Generate Code** icon in the upper-right of the Smart Configurator pane as shown in the picture below.

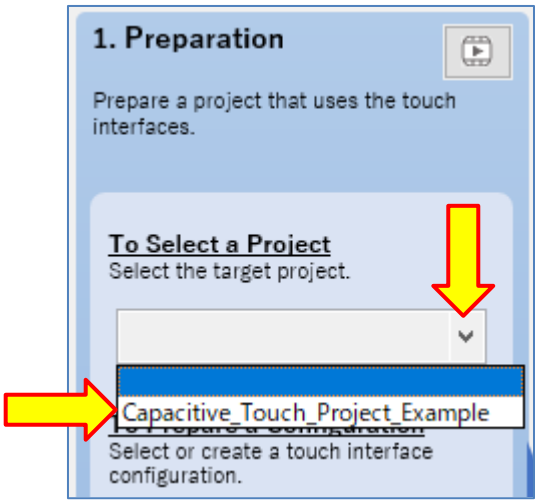


9. Creating the Capacitive Touch Interface

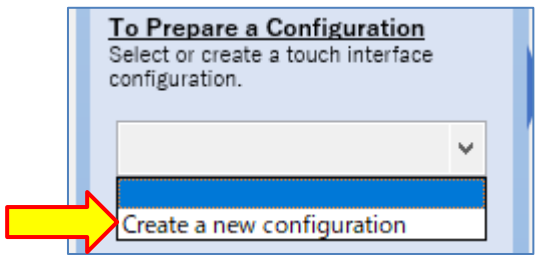
1. From the e² studio IDE, use **Renesas Views -> Renesas QE -> CapTouch Main (QE)** / in QE V3.2.0 or later, **CapTouch Workflow (QE)** to open the main perspecitve for configuring capacitive touch to the project.



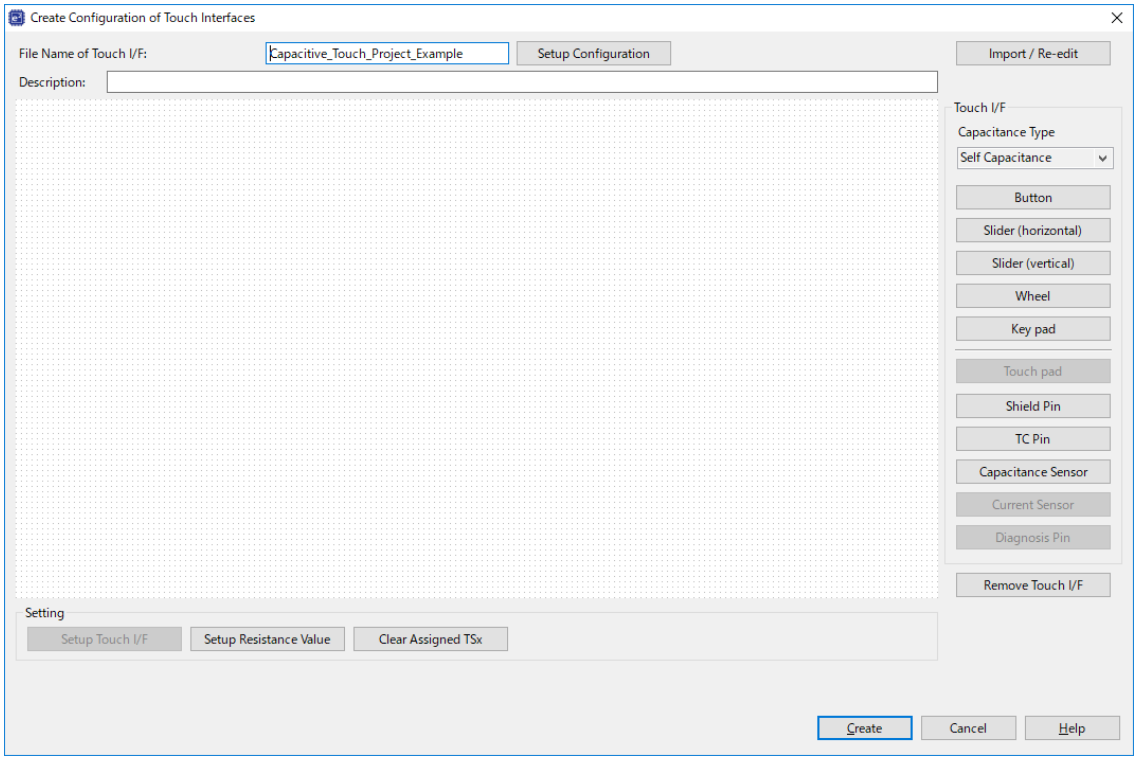
2. In the **CapTouch Main (QE)** / in QE V3.2.0 or later, **CapTouch Workflow (QE)** pane, select the project to configure the touch interface for by using the pull-down tab and selecting the **Capacitive_Touch_Project_Example** project as shown below.



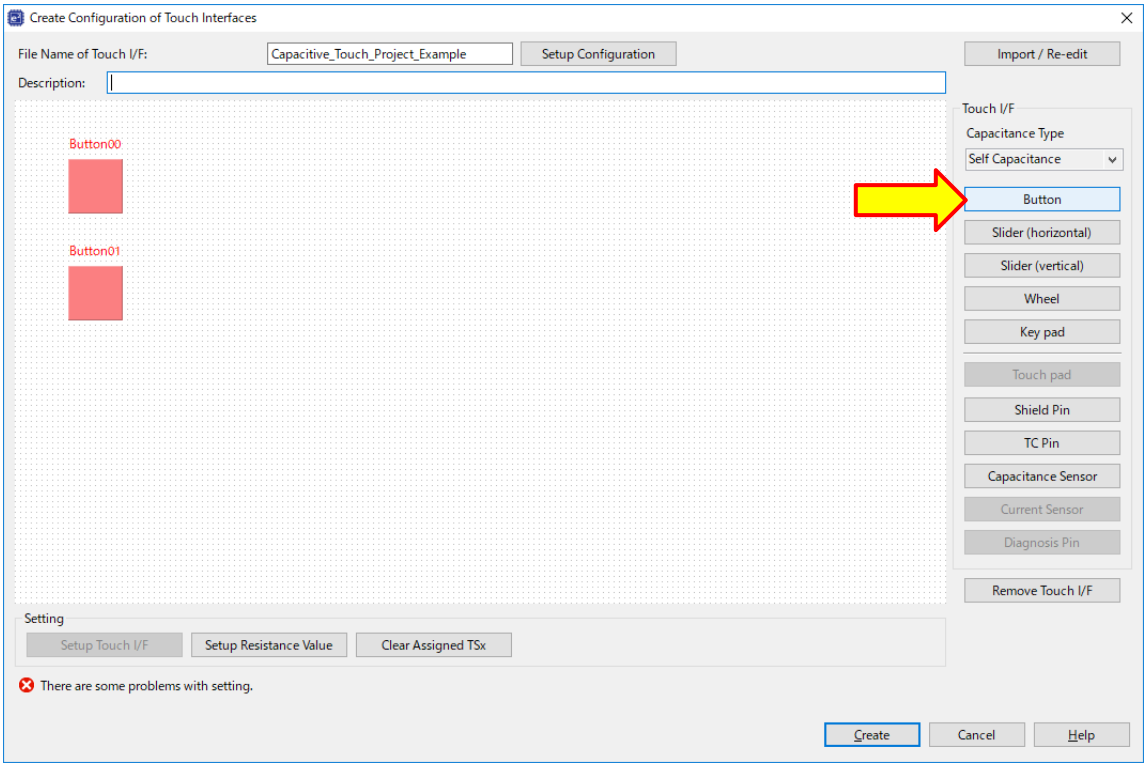
3. Next, create a new touch configuration by using the lower pull-down and selecting **Create a new configuration**.



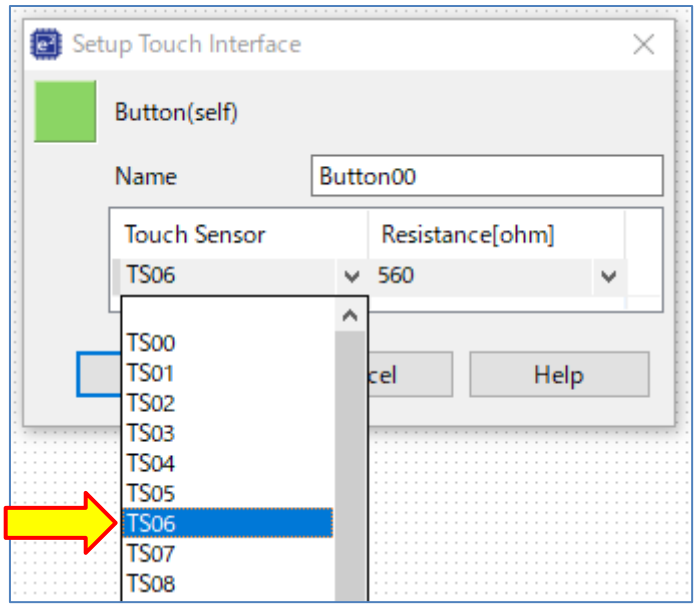
4. A new menu window will open with shows the default blank canvas for creating the touch interface.



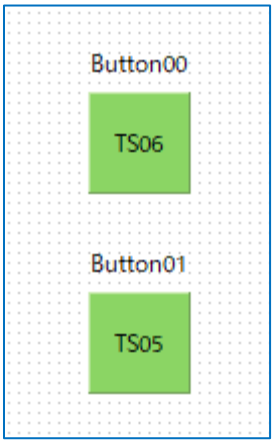
5. Add 2 buttons to the canvas by selecting the **Button** menu item from the right-hand side and adding two (2) buttons to the canvas. Press the **ESC** key to exit once two buttons are added. The canvas will look similar to below.



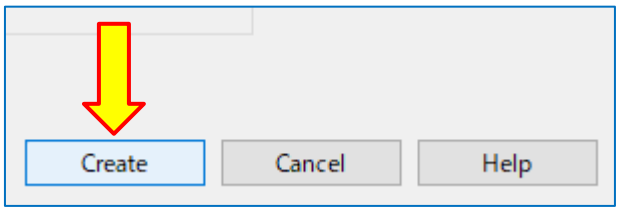
6. To make this connection, double-click on **Button00** and a dialog box will appear. In this case, using the pull-down, select **TS06** as the MCU sensor to assign to this button.



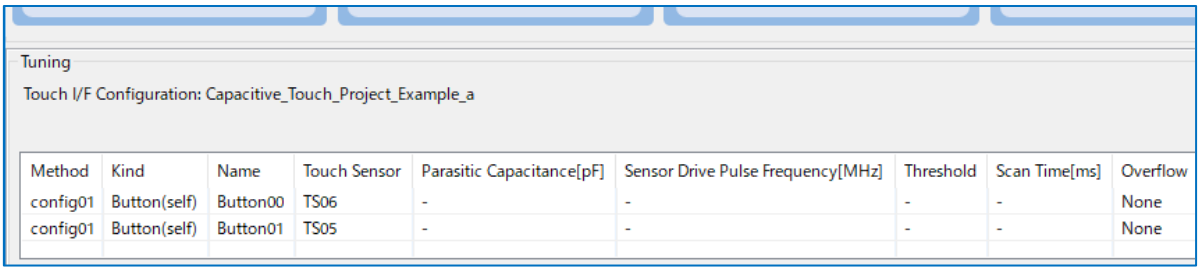
7. Perform the same operation as the previous step for **Button01** and assign it to **TS05**. The canvas should look similar to below. Note also, the indication of a configuration error will go away once all assignment are made properly and correspond to the enabled channels in the Smart Configurator.



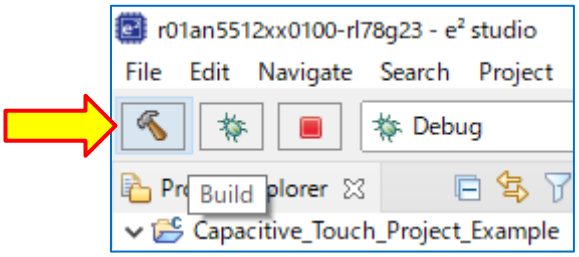
8. Click **Create** in the dialog box. This will setup the touch interface.



9. The **CapTouch Main (QE)** / in QE V3.2.0 or later, **CapTouch Workflow (QE)** window will now display the configuration of the touch interface in the main view pane.

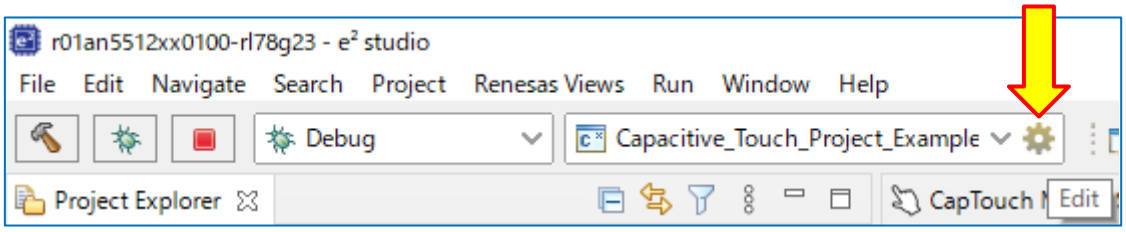


10. Build the project using the hammer icon in the upper left-hand side of the IDE. The project should build without any errors or warnings.

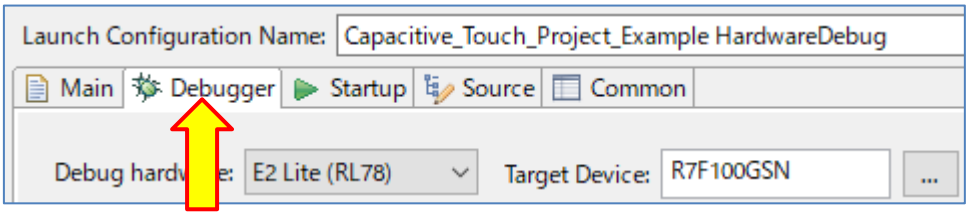


10. Modifying the e² studio Project Debug Session for Capacitive Touch Tuning

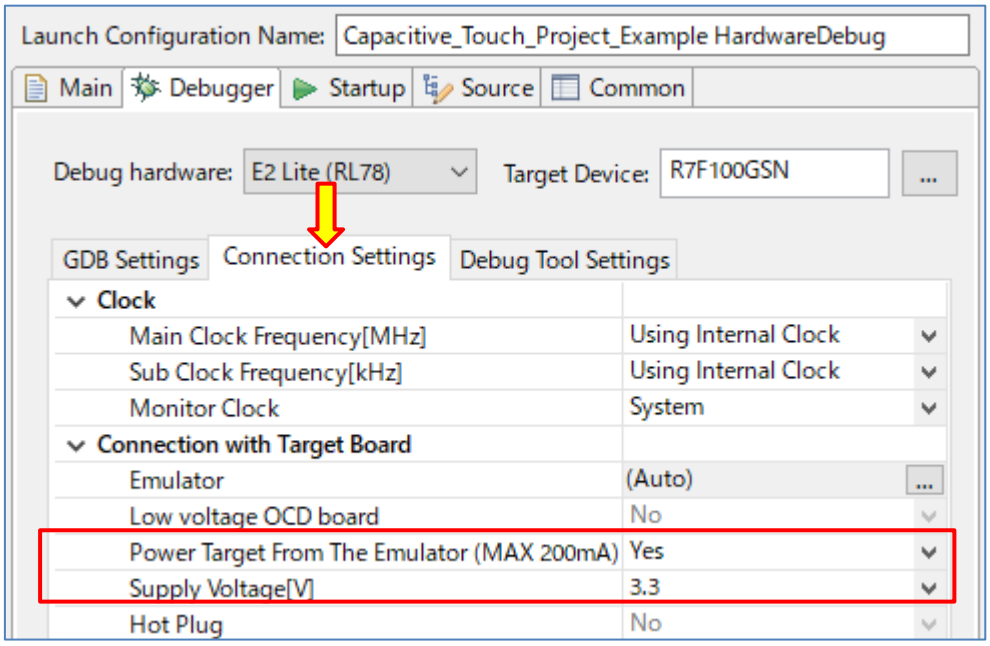
1. The debug session needs to be modified slightly so that a special tuning kernel can be downloaded into the MCU RAM after the debug session starts. Enter the Debug Configuration by clicking the gear icon in the upper left-hand side of the IDE.



2. In the pane that opens, select the **Debugger** tab.

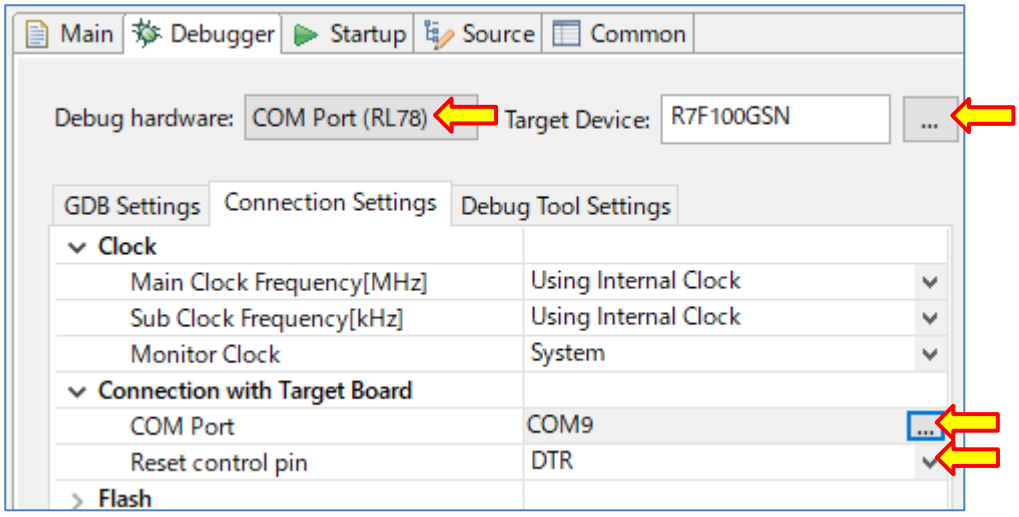


3. Select the **Connection Settings** tab. For this application example, the power for the target board is supplied from the emulator power supply. Ensure **Power Target From The Emulator (MAX 200mA)** and **Supply Voltage [V]** are set as shown in the red frame below.

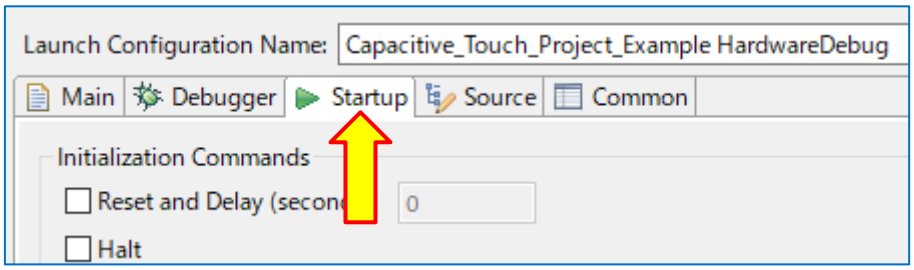


Note 1: For sake of simplicity, in this application example, the power for the target board is supplied from the emulator power supply. Renesas recommends that using be done with the end application power supply to alleviate any voltage deviations that might occur from using the power supplied thru the E2 emulator Lite/PC USB port.

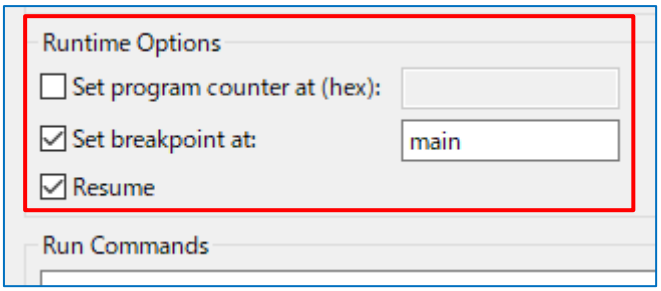
Note 2: The debug method that can be executed depends on the specifications of the target board. Select Debug hardware: according to the debugging method to be executed, and set the debugging configuration. For example, when executing COM port debug, the settings are different as shown in the image below.



4. Select the **Startup** tab



5. Ensure the two check boxes **Set breakpoint at:** and **Resume** are checked and look as follows. You may need to scroll down in the dialog box to see these check boxes.

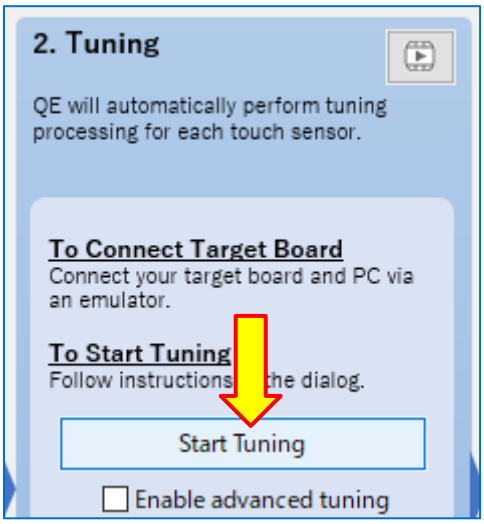


6. Click **OK** to use these modified settings. This completes the project configuration and debug setup for tuning.

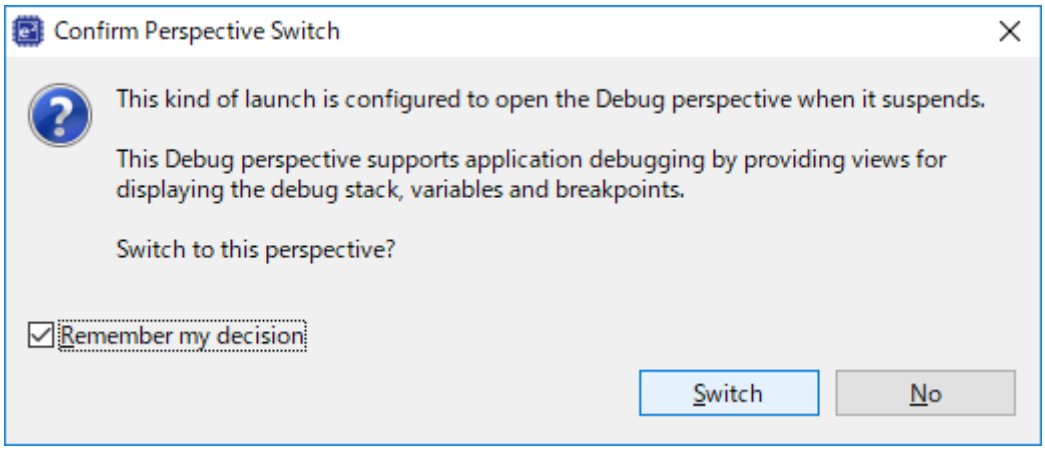
11. Tuning the Capacitive Touch Interface Using QE for Capacitive Touch Plug-in

1. To start the automatic tuning process, click the button **Start Tuning** in the **CapTouch Main (QE)** / in QE V3.2.0 or later, **CapTouch Workflow (QE)** e^2 studio IDE.

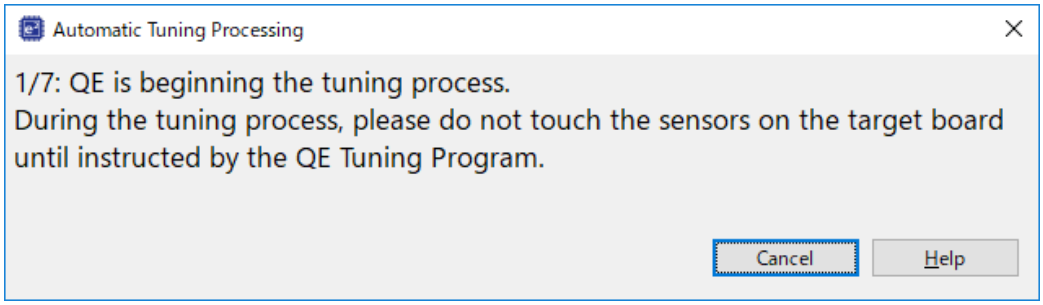
Note: For sake of simplicity, in this application example, the power for the target board is supplied from the emulator power supply. Renesas recommends that tuning be done with the end application power supply to alleviate any voltage deviations that might occur from using the power supplied thru the E2 emulator Lite/PC USB port.



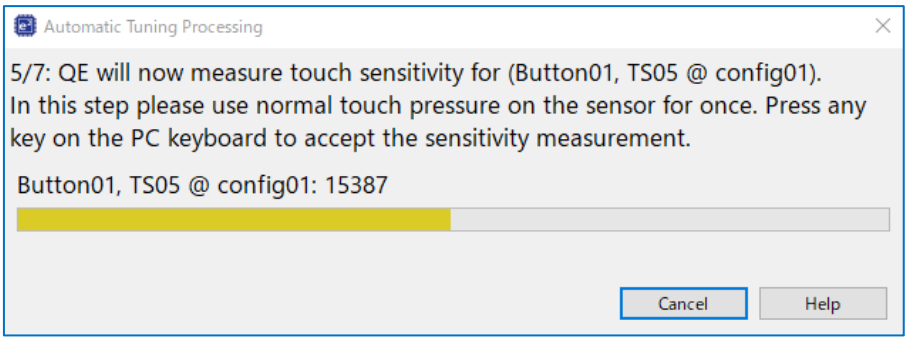
2. At the start of the first debug session, e^2 studio may display a message indicating the Debug perspective will be switched to. Click the **Remember my decision** check box and **Switch** to continue the Debug process and the QE for Capacitive Touch automatic tuning.



3. QE for Capacitive Touch automatic tuning will now begin. Please carefully read the tuning dialog windows as they will guide you thru the tuning process. An example screen is shown below. Typically, no interaction is required during the initial tuning process steps.

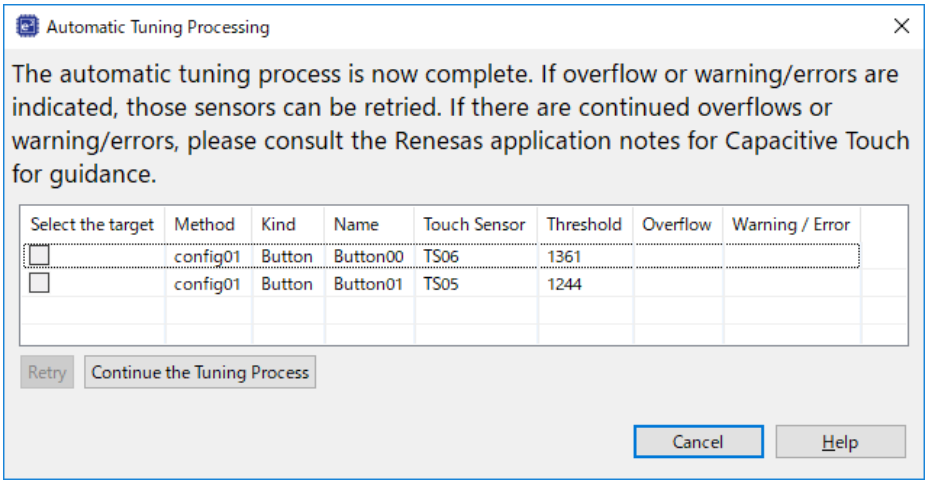


After several automated steps, a dialog box with information similar to what is shown below will appear. This is the **touch sensitivity measurement** step of the tuning process. As the first 'interactive' step of the tuning process, press using **normal touch pressure** on the sensor being indicated in the dialog box (**Button01**/**TS05**). When pressing the bar graph will increase to the right and the touch counts go numerically **UP.** While holding that pressure, **press any key on the PC keyboard** to accept the measurement.

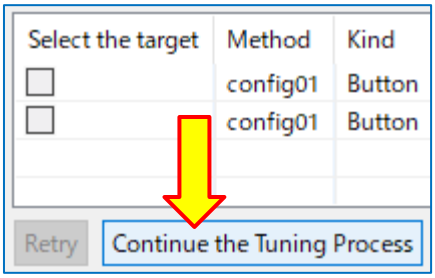


4. Repeat the previous steps for **Button00**/**TS06**.

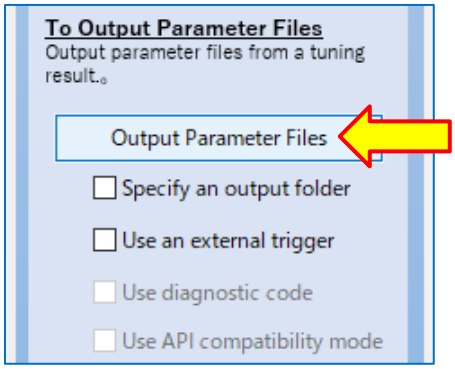
5. Once sensitivity measurement for the buttons is complete, you will see a screen like what is shown below. This is the detection threshold that is used by the middleware to determine if a touch event has occurred.



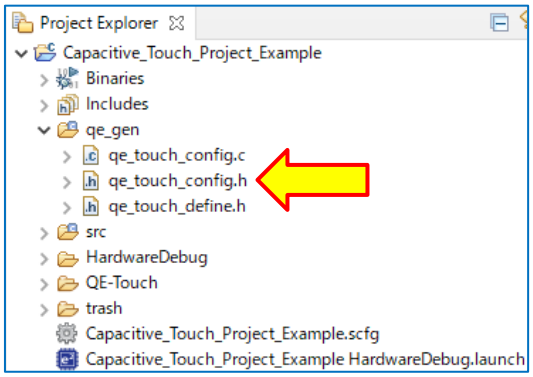
6. Click the **Continue the Tuning Process** button in the dialog box shown. This will exit the tuning process and disconnect from the Debug session on the target. You should return to the default **Cap Touch Main (QE)** / in QE V3.2.0 or later, **CapTouch Workflow (QE)** screen in the e^2 studio IDE.



7. In this example, all that is left is to output the tuning parameter files. Click the button **Output Parameter Files**.



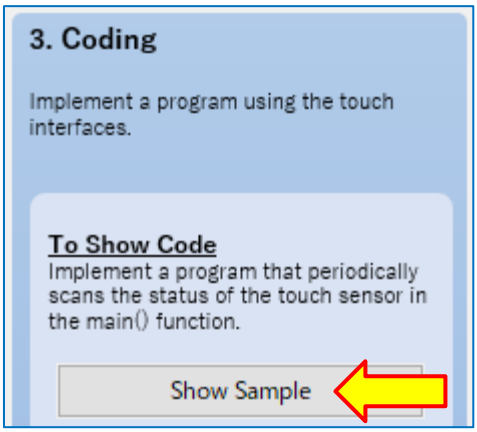
8. In the **Project Explorer** window and you will see that **qe_touch_config.c, qe_touch_config.h** and **qe_touch_define.h** files have been added. These contain the needed tuning information to enable touch detection using the module of SIS.



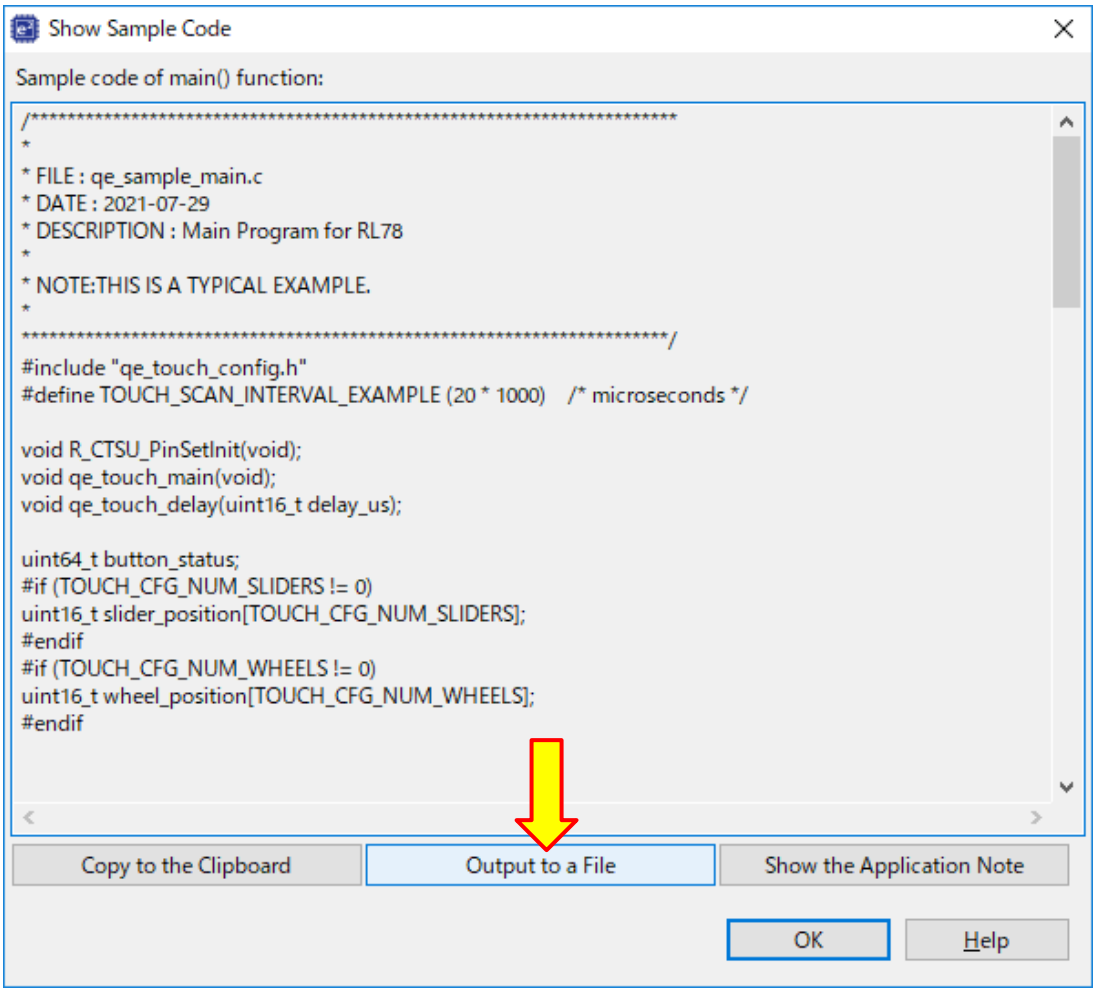
9. Build the project using the hammer icon in the upper left-hand side of the IDE. The Console output showing build results should show no errors.

12. Adding rm_touch SIS Function Calls to Application Example

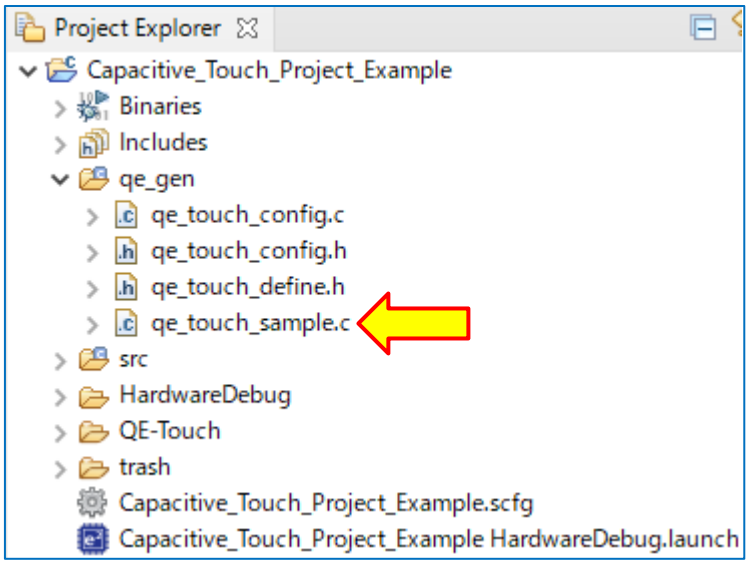
1. To implement application code to scan and report the state of the the touch sensor, click the button **Show Sample** in the **CapTouch Main (QE)** / in QE V3.2.0 or later, **CapTouch Workflow (QE)** e^2 studio IDE.



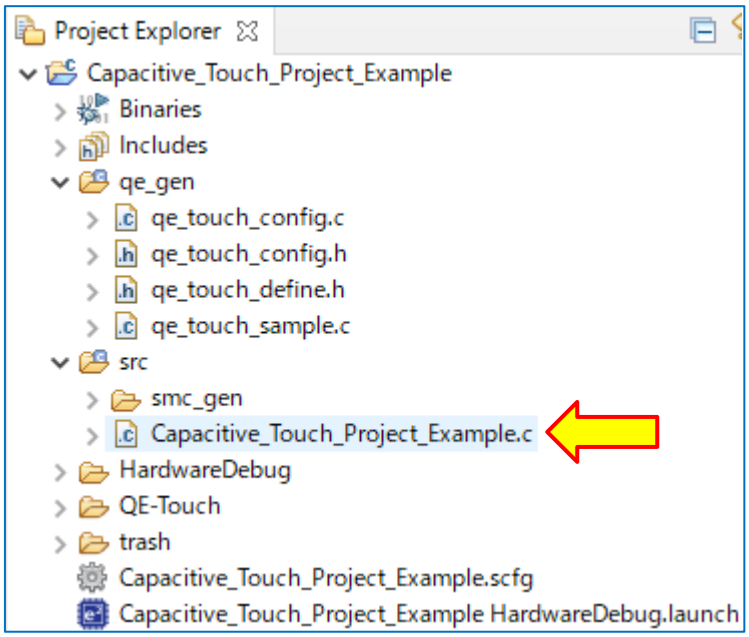
2. A new menu window will open with shows the sample code in text. Click the button **Output to a File.**



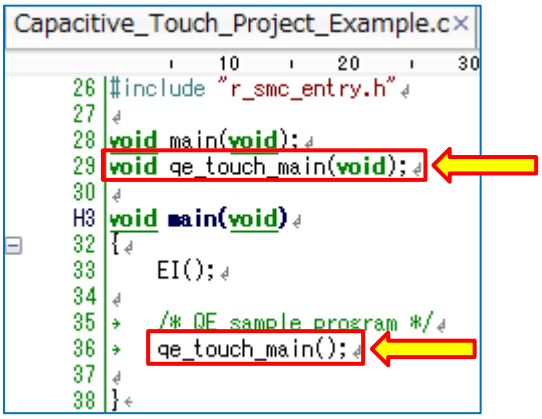
3. Created a new project file that describes the sample code. In the Project Explorer window and you will see that **qe_touch_sample.c** files have been added.



4. Open the **Capacitive_Touch_Project_Example.c** file.



5. In the main(), call the qe_touch_main(). Add the code (“**void qe_touch_main(void);**” and “**qe_touch_main();**”) in the red frame to **Capacitive_Touch_Project_Example.c** file as shown in the image below.



6. This completes all the needed code modifications required for this simple application example. Building the code should result in no errors or warnings for this simplified application example.

13. [Additional function] Setting the serial communication monitor using UART (2/3)

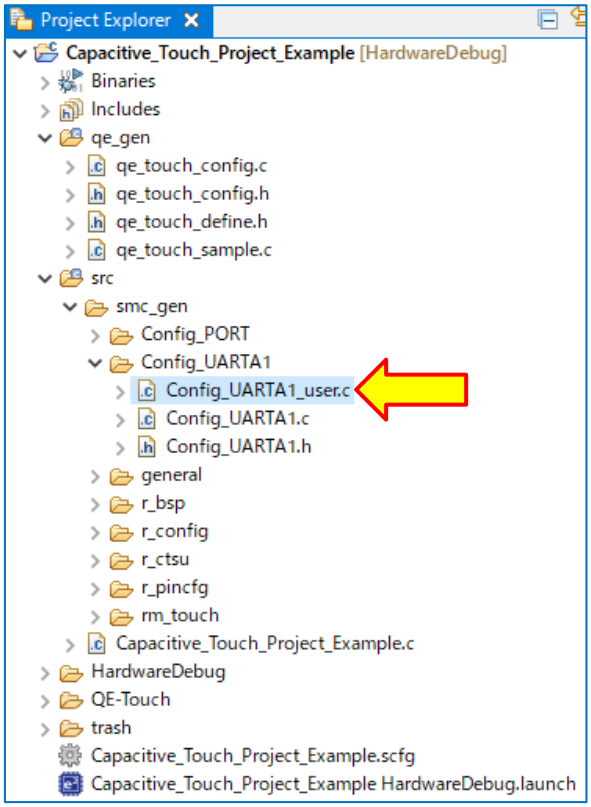
Note: Monitoring touch performance for touch applications can be confirmed by communication via the OCD (On-Chip Debugging) emulator. However, RL78 family case, monitoring performance is limited by the OCD function of the RL78 family.

On the other hand, monitoring touch performance can also be achieved via serial communication. Therefore, if you want to monitor smoothly, please add the monitoring function via serial communication.

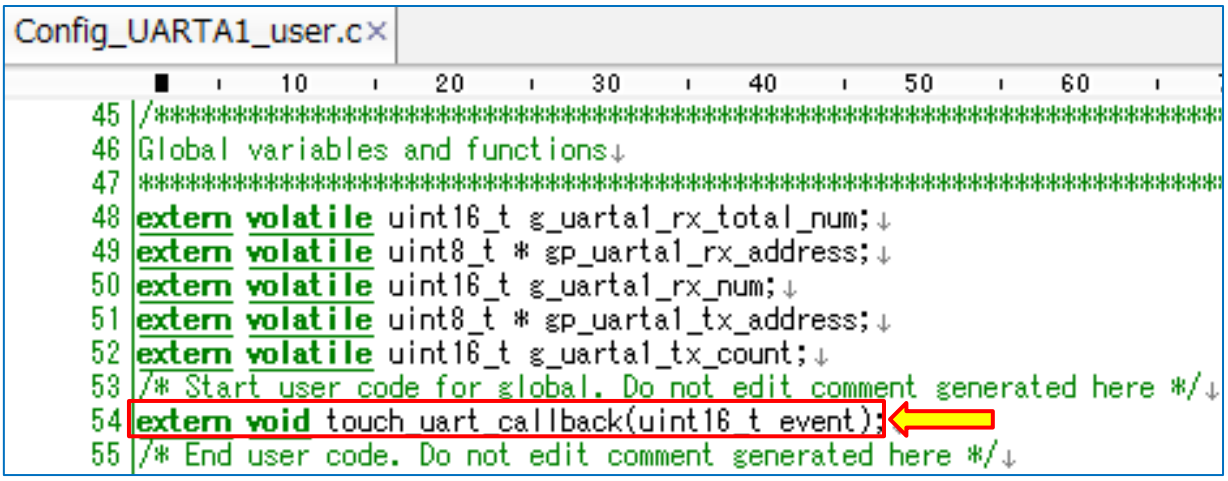
Chapters 8, 13 and 15 (including this chapter) shown below describe setting the serial communication monitor using UART.

- “8. [Additional function] Setting the serial communication monitor using UART (1/3)”
- “13. [Additional function] Setting the serial communication monitor using UART (2/3)”
- “15. [Additional function] Setting the serial communication monitor using UART (3/3)”

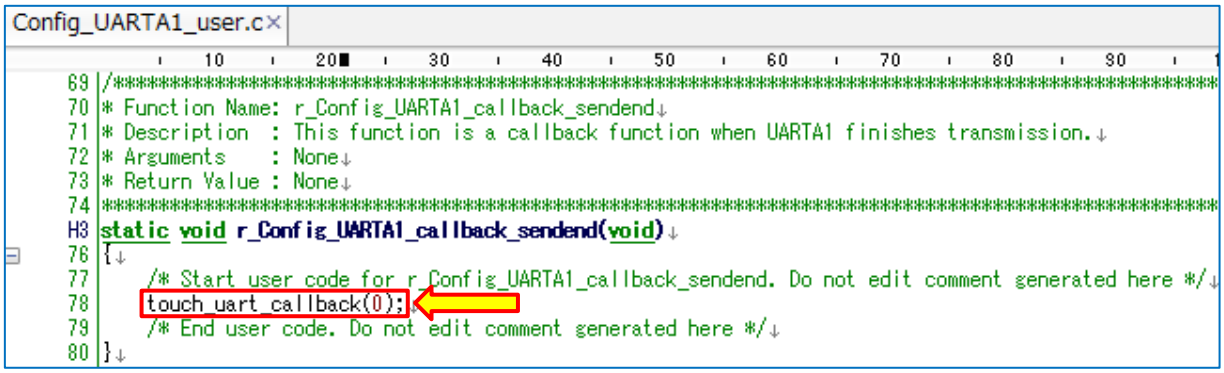
1. Open the **Config_UARTA1_user.c** file.



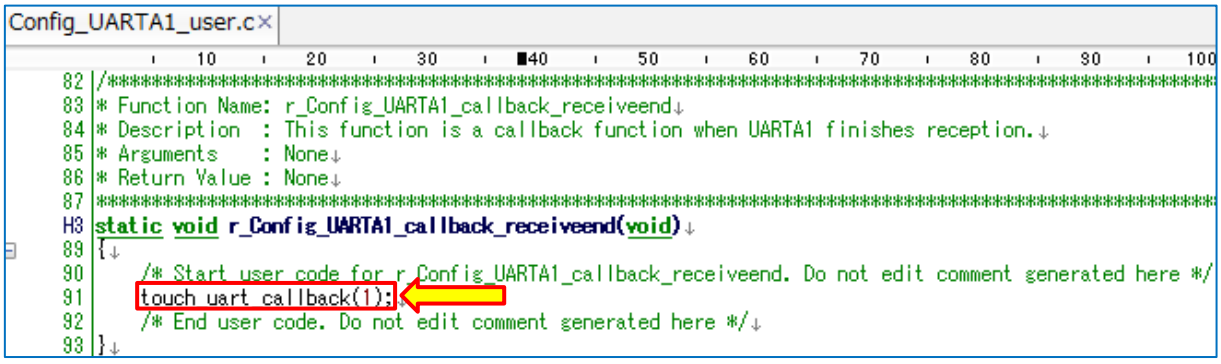
2. Add the code "**extern void touch_uart_callback(uint16_t event);**" in the red frame to **Config_UARTA1_user.c** file as shown in the image below.



3. Add the code "**touch_uart_callback(0);**" in the red frame to **Config_UARTA1_user.c** file as shown in the image below.



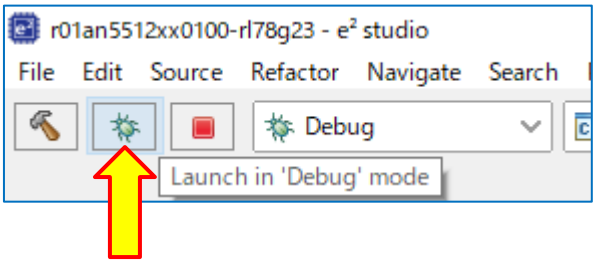
4. Add the code "**touch_uart_callback(1);**" in the red frame to **Config_UARTA1_user.c** file as shown in the image below.



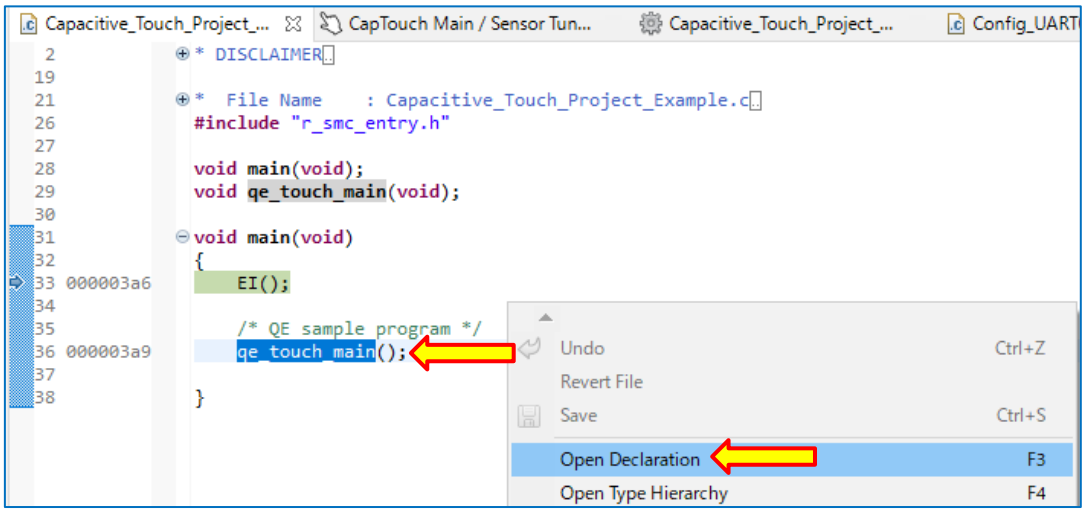
5. Build the project using the hammer icon in the upper left-hand side of the IDE. The Console output showing build results should show no errors.

14. Monitoring Touch Performance using e² studio Expressions Window and QE for Capacitive Touch

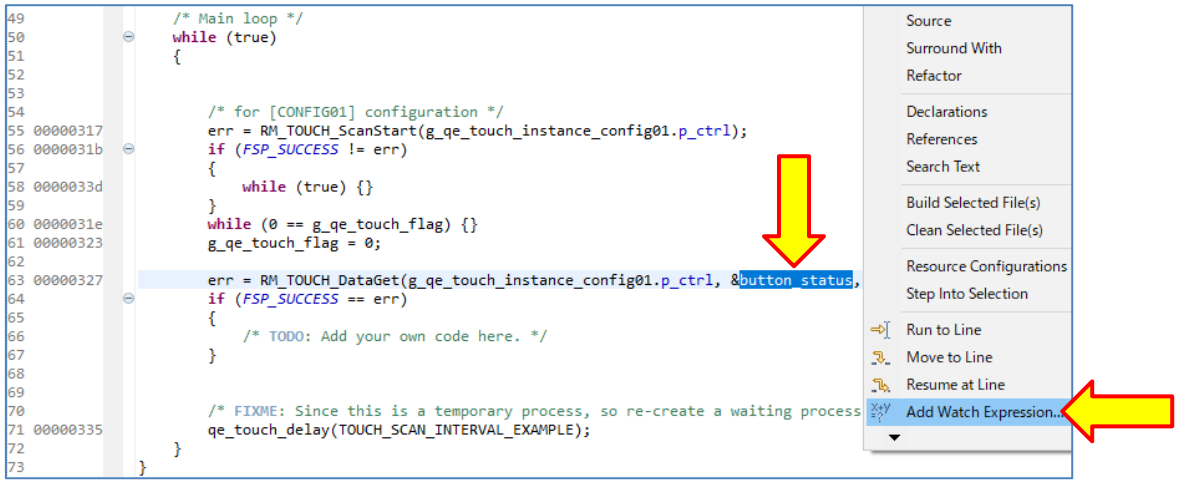
1. Start a Debug session by clicking the Bug icon in the upper left-hand corner of e² studio. A Debug session will commence.



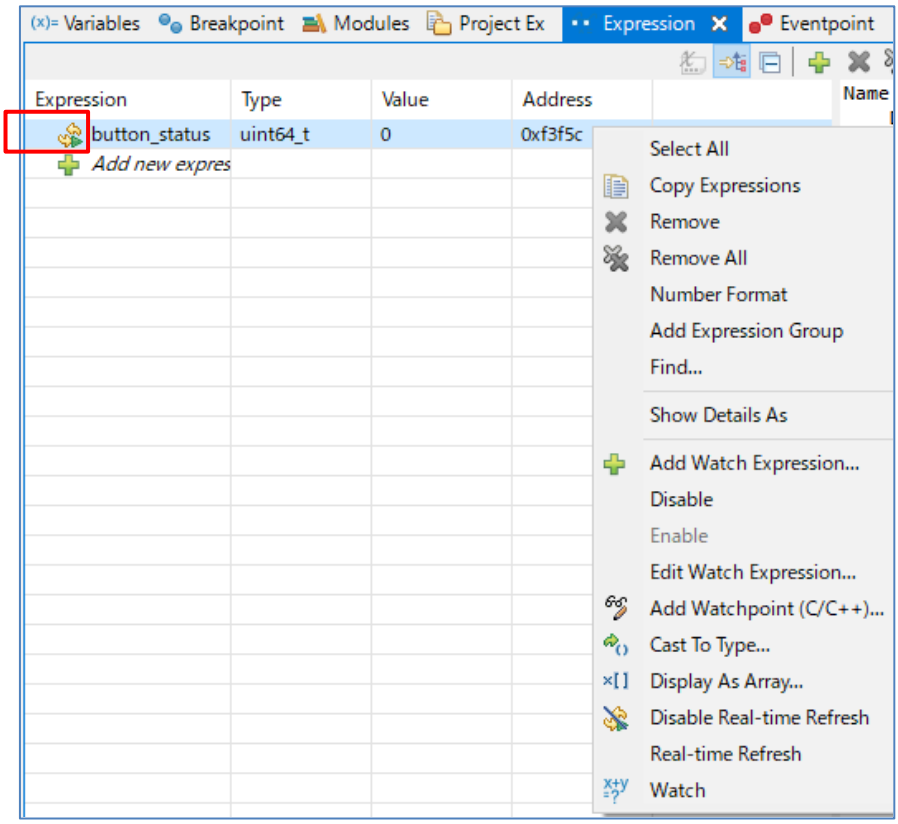
2. The debugger will stop at the **main()** function call.
3. Open the declaration of **qe_touch_main()** function.



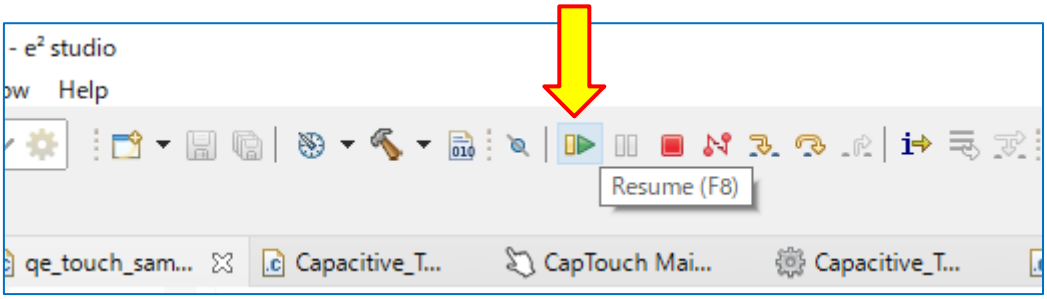
4. Scroll down in the **qe_touch_sample.c** file to the **RM_TOUCH_DataGet()** function in the **while (true)** loop. Add the variable **button_status** to the expressions window.



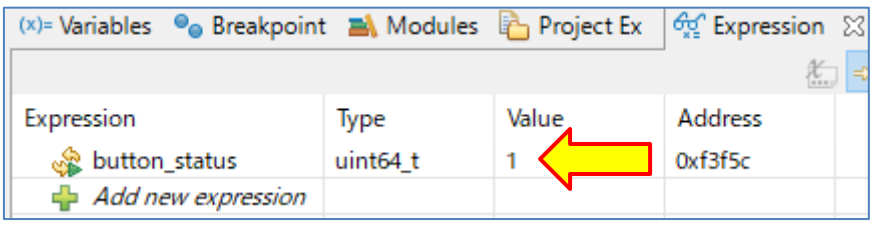
5. Ensure that Real-time Refresh is enabled for the variable in the Expressions window.



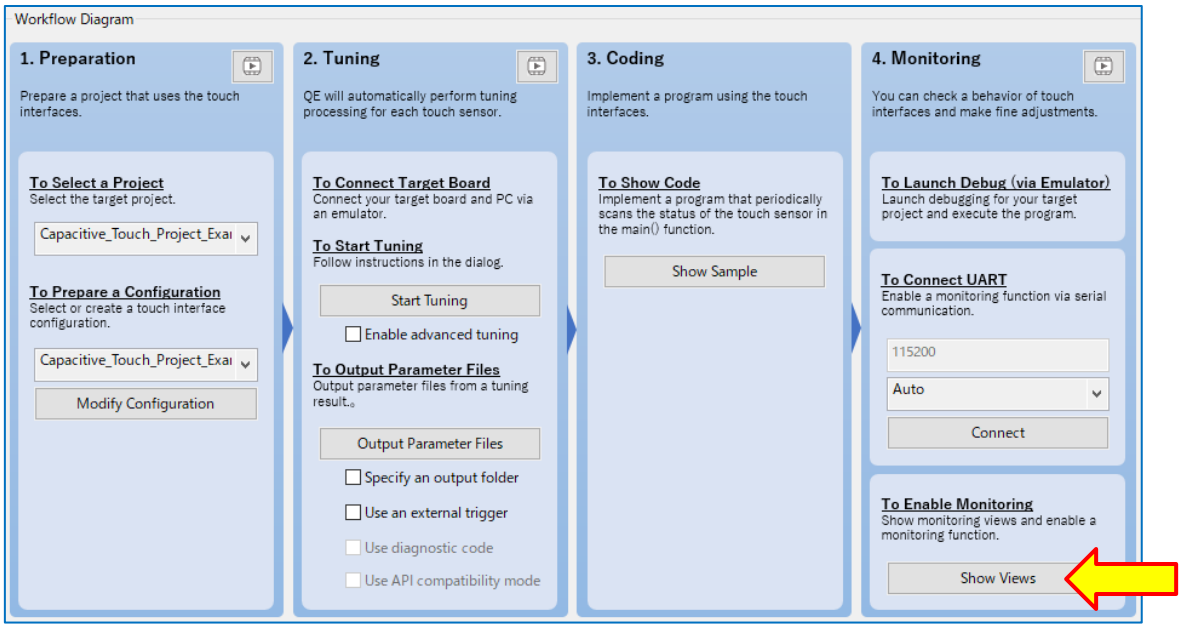
6. Click the '**Resume**' button located approximately in the middle of the e² studio menu bar to continue code execution.



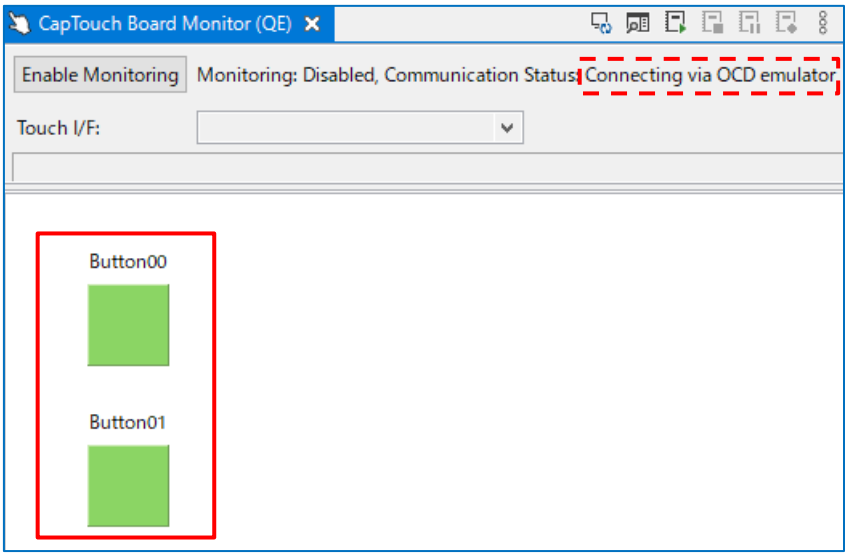
7. Press **TS05** on the board, which was configured as **Button01** in Chapter "**9. Creating the Capacitive Touch Interface**" of this application guide. When pressed, a '1' will appear for **button_status** in the Expression window, indicating a binary indication of touch.



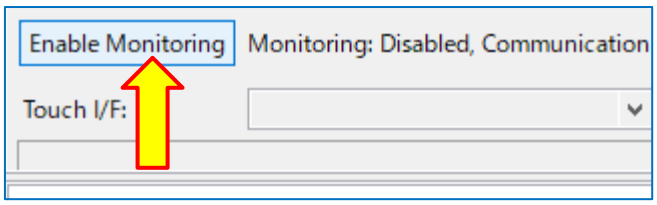
8. Open the Monitoring view from **Show Views** of **CapTouch Main (QE) /** in QE V3.2.0 or later, **CapTouch Workflow (QE)**.



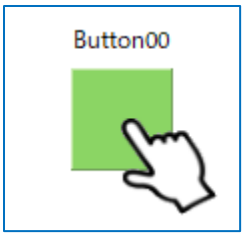
9. It may be necessary to drag the pane up for better viewing, however you should see the **CapTouch Board Monitor (QE)** pane appear like the image below.



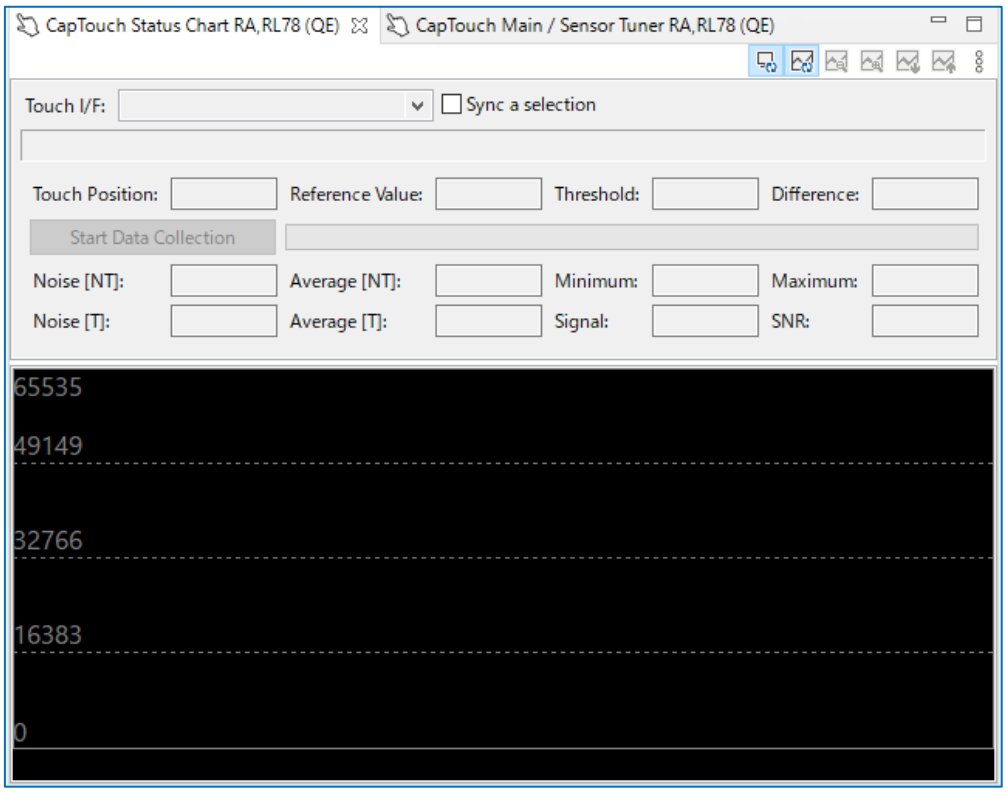
10. Click the **Enable Monitoring** button. The dialog text will change to **Monitoring: Enabled**.



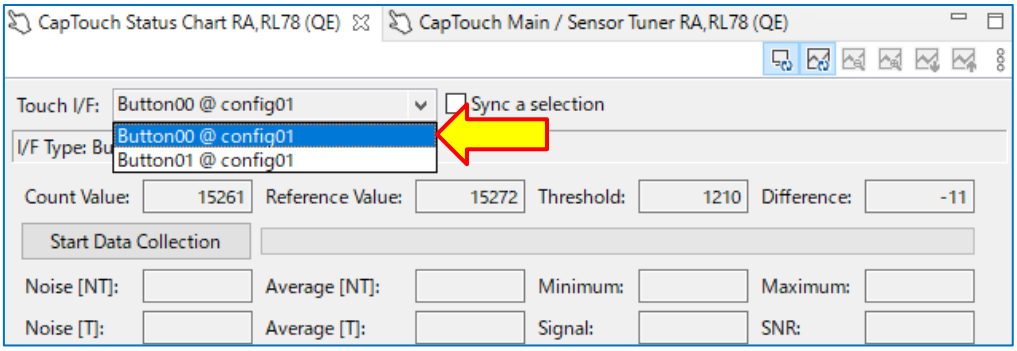
11. Touch the button **TS06** on the target board. The **CapTouch Board Monitor (QE)** will show a touch with a finger image on the button like the below image.



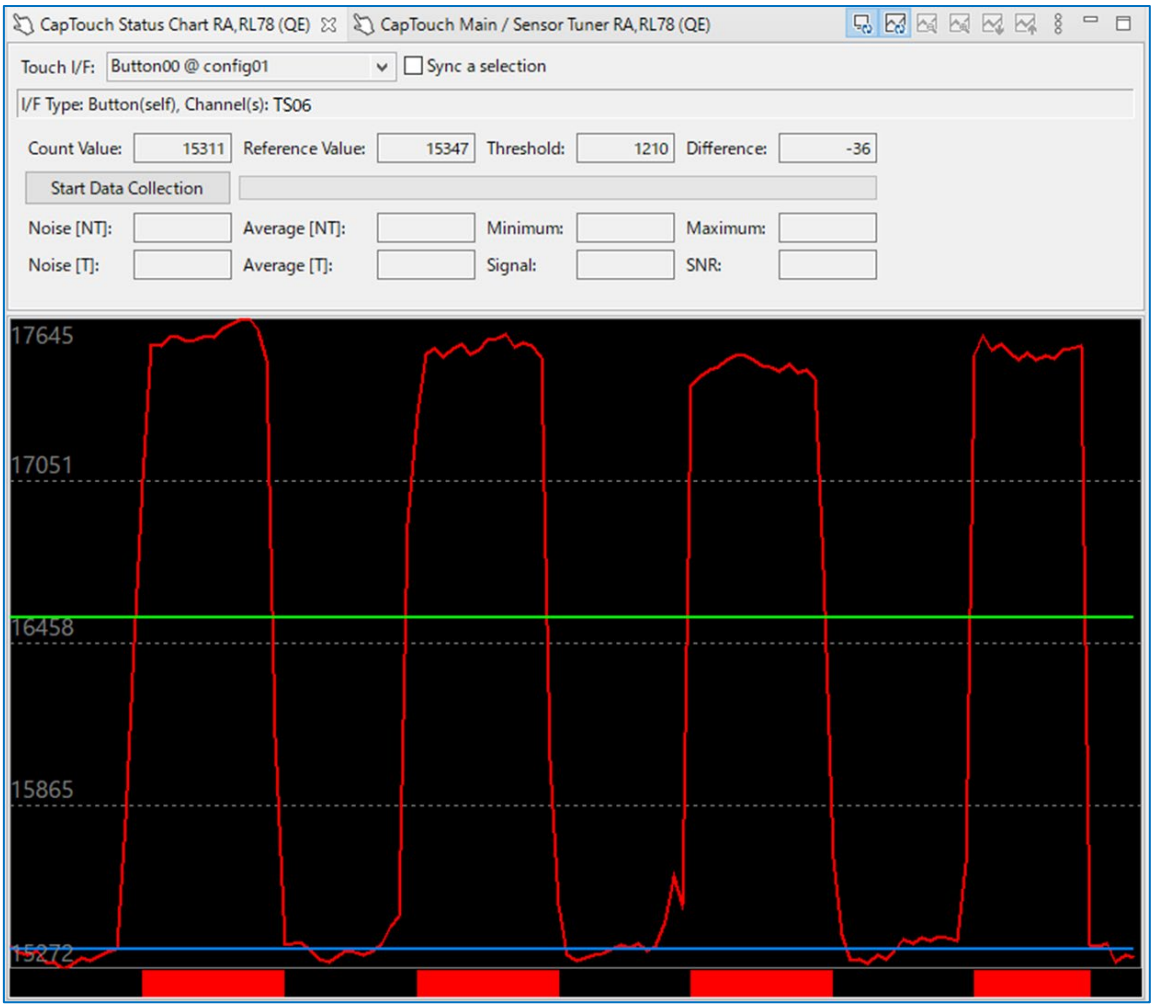
12. To see a graphical representation of the 'touch counts' from the board, use the **CapTouch Status Chart (QE)**.



13. Using the pulldown, select **Button00 @ config01**.

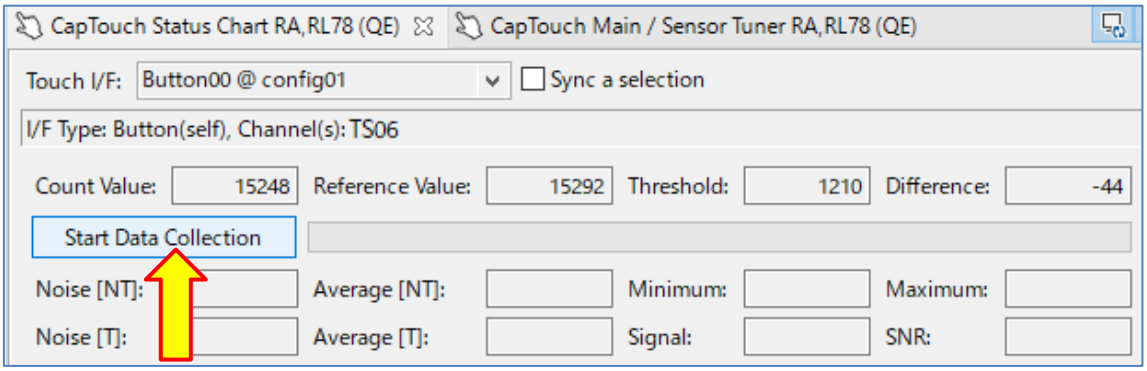


14. The graph will begin to display running values. Touch **TS06** on the board and you should see the ‘touch counts’ show as a step change on the running graph. The **GREEN** line is the touch ‘**Threshold’**, which the middleware uses to determine whether a button is actuated/touched. The **RED BELT** at the bottom of the graph is a visual indication to the user that the ‘touch counts’ have crossed above the threshold and a touch is detected**.**

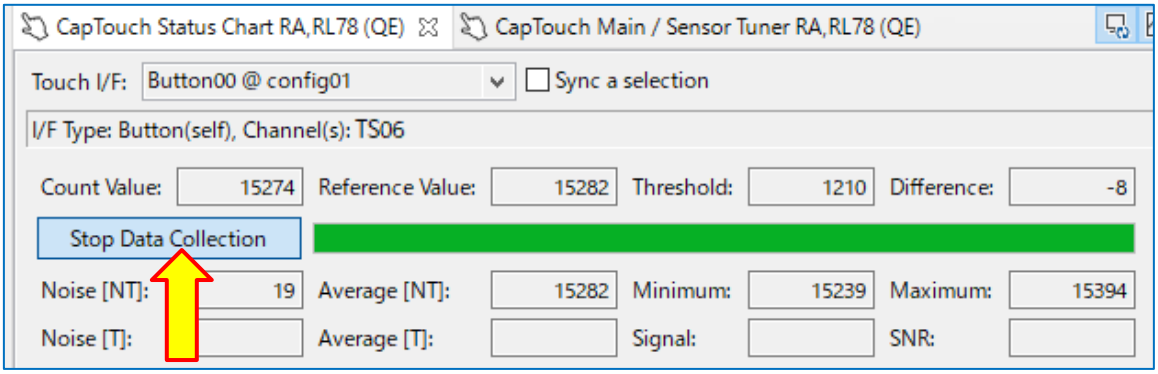


Note: Sections 15 to 18 should only be set when displaying and measuring standard deviation.

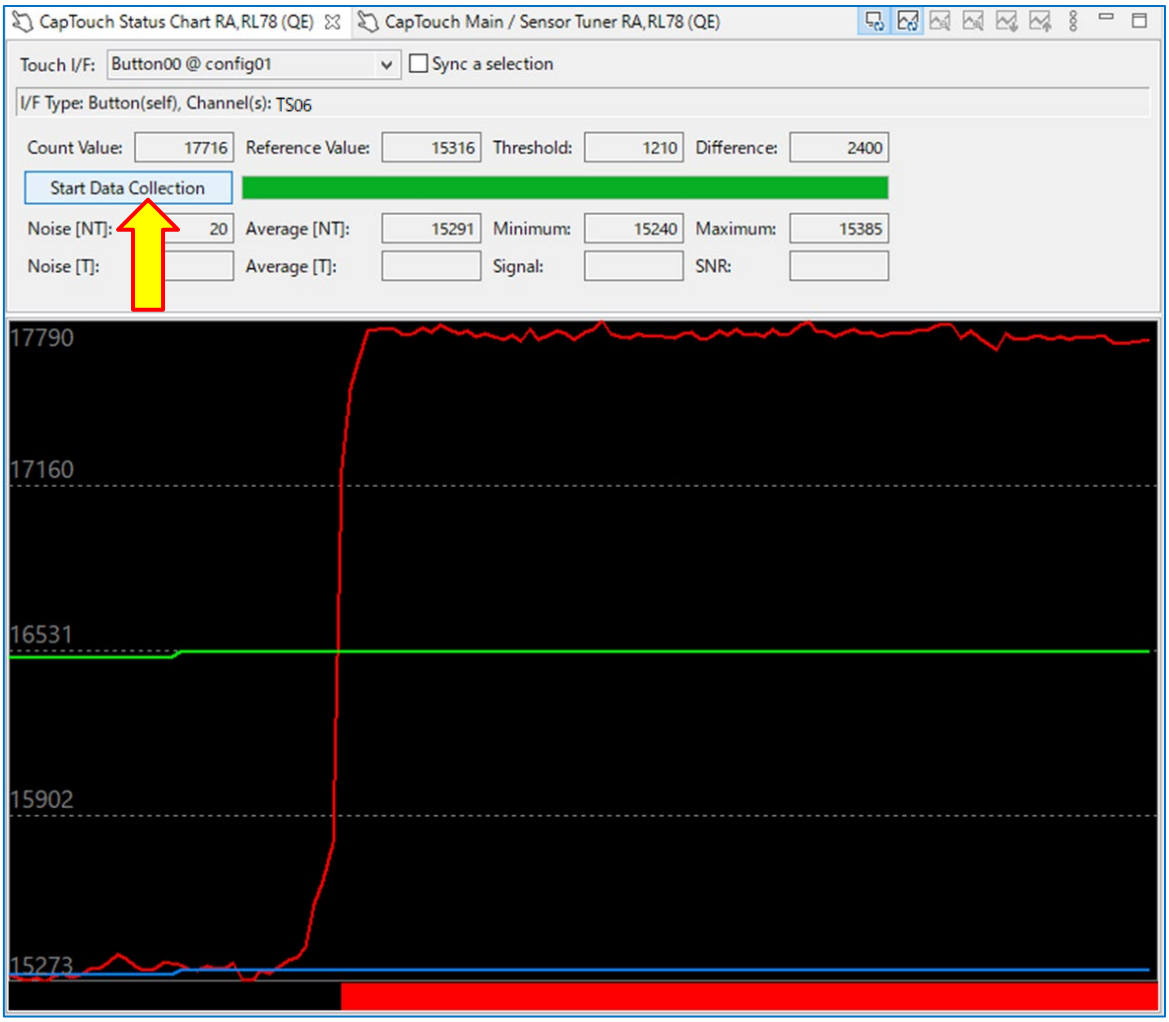
15. Next, measure standard deviation. Click the **Start Data Collection** button. Don't touch the electrode as this will collect data of **touch-off** state. The green bar is the data collection rate. When the green bar goes all the way to the right, the data collection of **touch-off** state is complete.



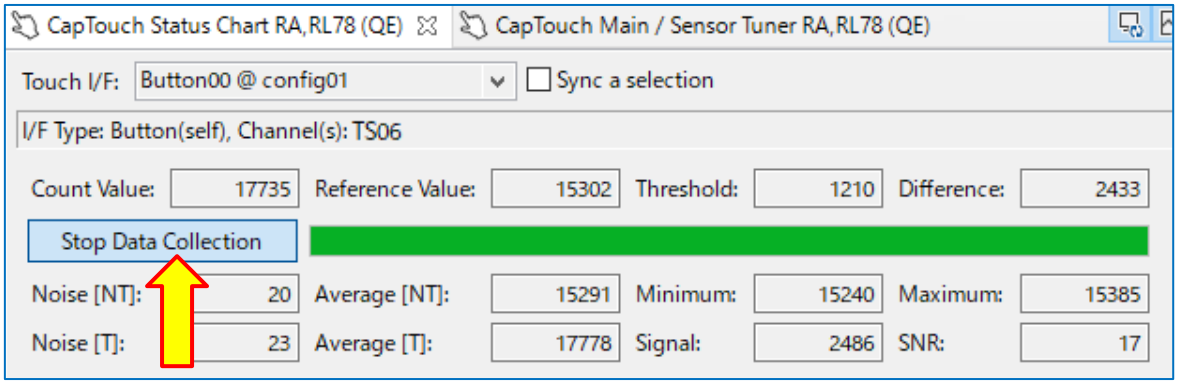
16. Click the **Stop Data Collection** button, when the green bar goes all the way to the right.



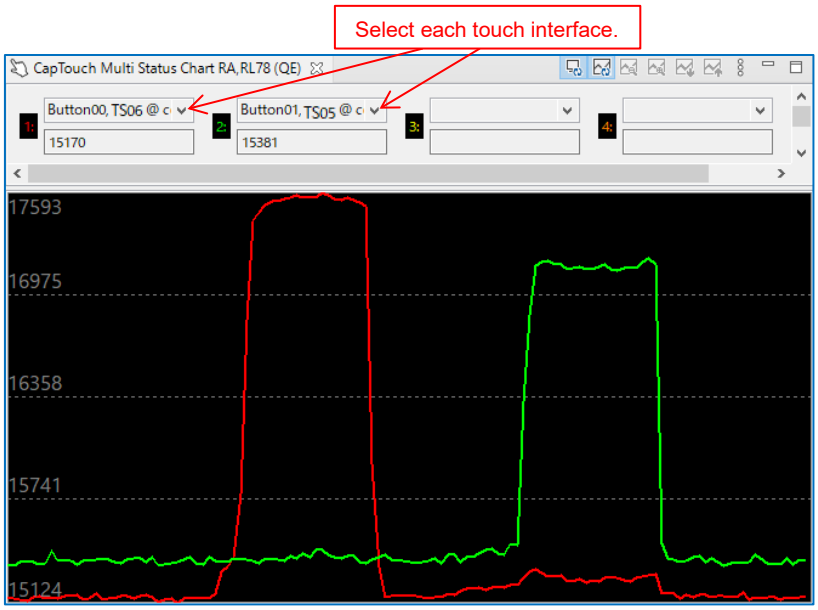
17. Next, Touch the electrode as this will collect data of **touch-on** state. Click the **Start Data Collection** button while touching the electrode.



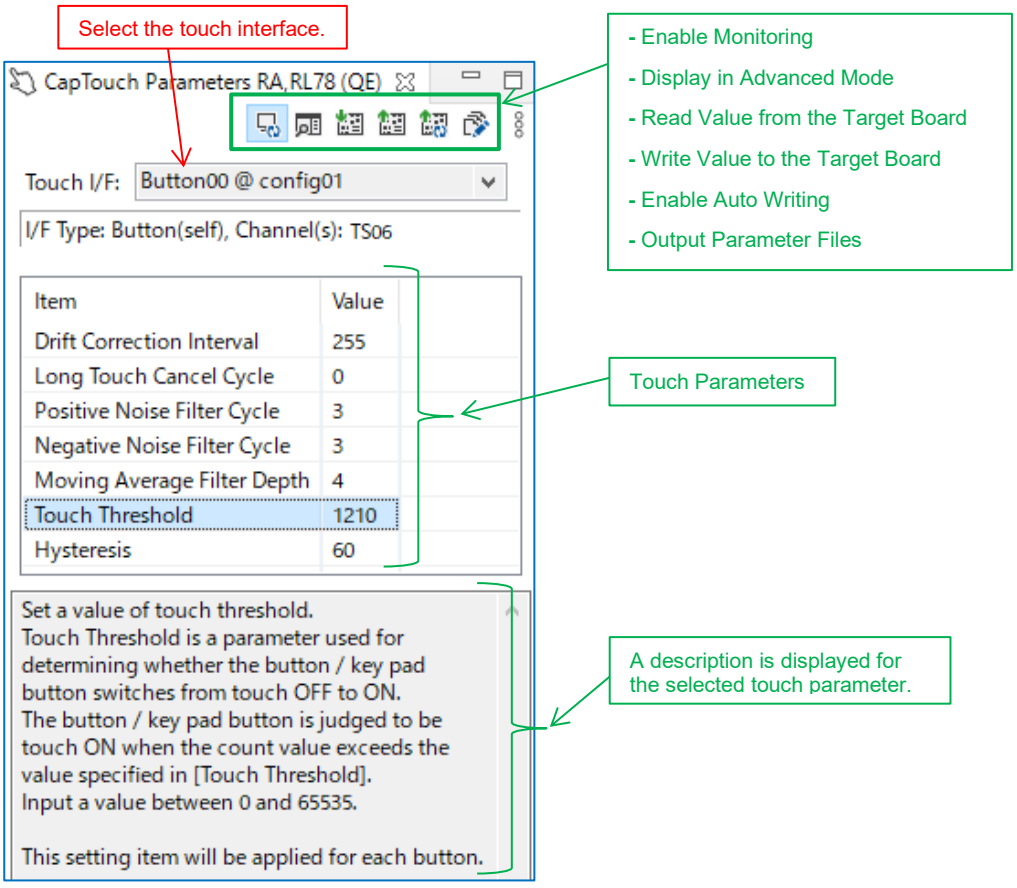
18. Click the **Stop Data Collection** button, when the green bar goes all the way to the right. The SNR is displayed when data collection is complete.



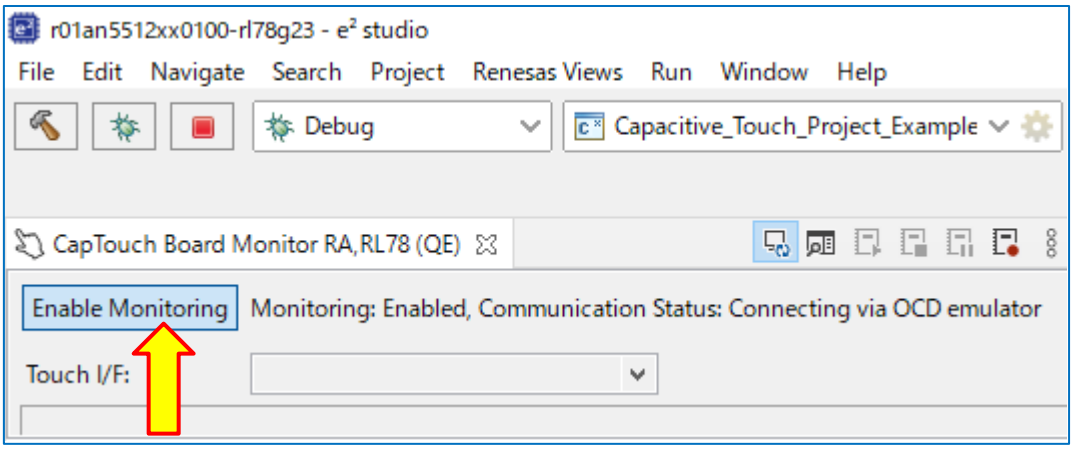
19. To see a graphical representation of the 'touch counts' for multiple touch sensors, use the **CapTouch Multi Status Chart (QE)**.



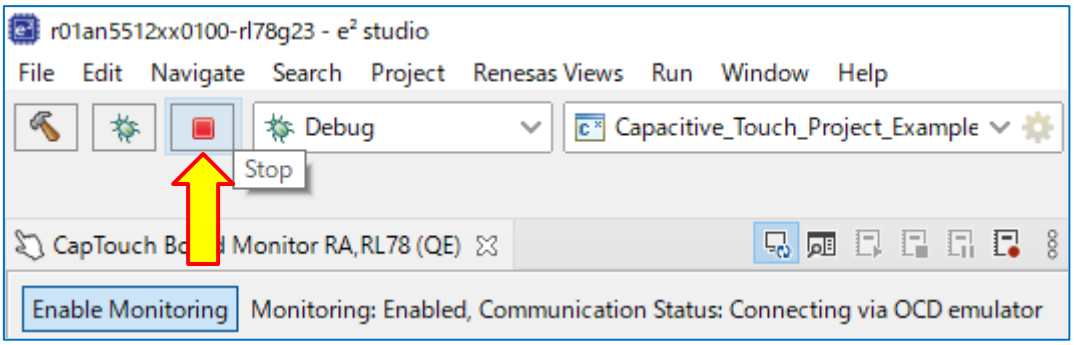
20. If you will check and adjust the touch parameters, use the **CapTouch Parameters (QE).**



21. If you will finish monitoring, click the **Enable Monitoring** button. The dialog text will change to **Monitoring: Disabled**.



22. If you will finish the debug session, click the **stop** icon to end the debug session.



15. [Additional function] Setting the serial communication monitor using UART (3/3)

Note: Monitoring touch performance for touch applications can be confirmed by communication via the OCD (On-Chip Debugging) emulator. However, RL78 family case, monitoring performance is limited by the OCD function of the RL78 family.

On the other hand, monitoring touch performance can also be achieved via serial communication. Therefore, if you want to monitor smoothly, please add the monitoring function via serial communication.

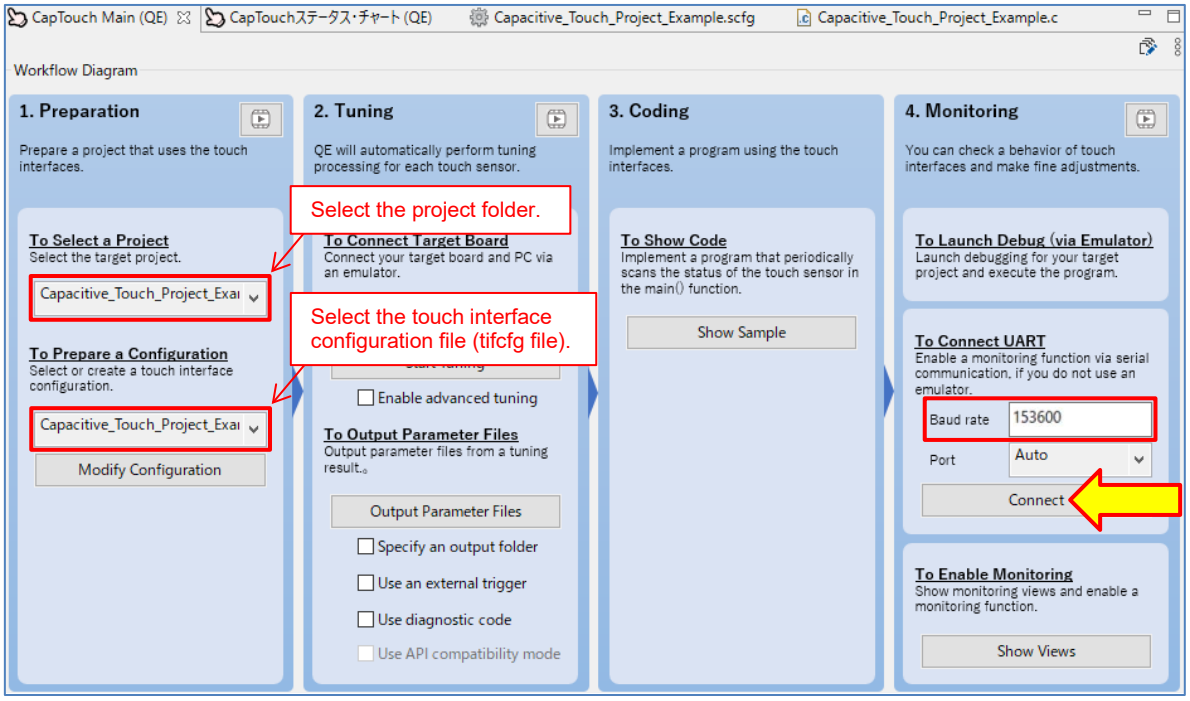
Chapters 8, 13 and 15 (including this chapter) shown below describe setting the serial communication monitor using UART.

- “8. [Additional function] Setting the serial communication monitor using UART (1/3)”
- “13. [Additional function] Setting the serial communication monitor using UART (2/3)”
- “15. [Additional function] Setting the serial communication monitor using UART (3/3)”

1. Connect the target board to the PC via serial connection (UART / USB).

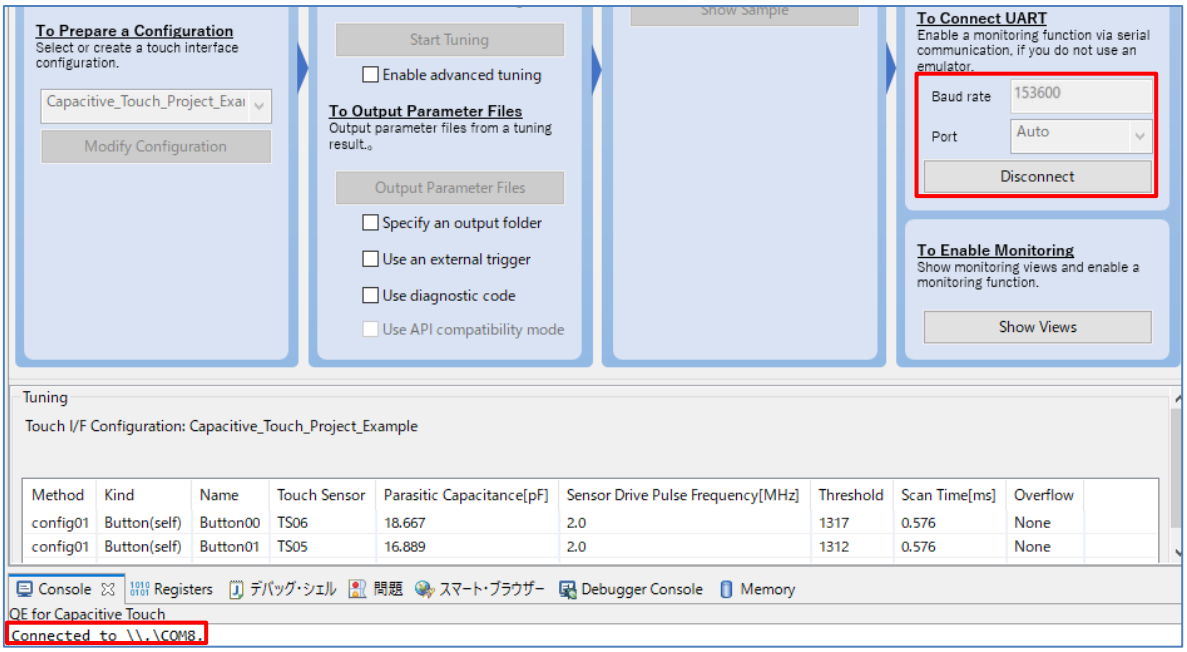
2. Run the touch application program on the target board.

3. Open the **CapTouch Main (QE)** / in QE V3.2.0 or later, **CapTouch Workflow (QE)** pane. Ensure the project folder (**Capacitive_Touch_Project_Example**) and the tifcfg file (**Capacitive_Touch_Project_Example.tifcfg**) are set as shown in the red frame below, then set the **Baud rate** to **153600** (bps) and click the **Connect** button.

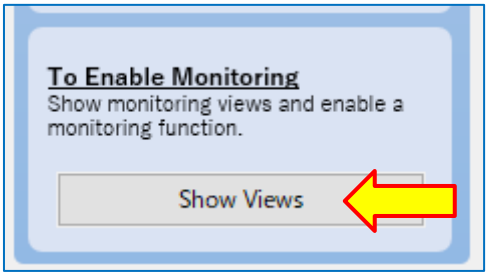


Note: For the baud rate (bps) setting value, use the baud rate (bps) set in step 5 of chapter "8. [Additional function] Setting the serial communication monitor using UART (1/3)".

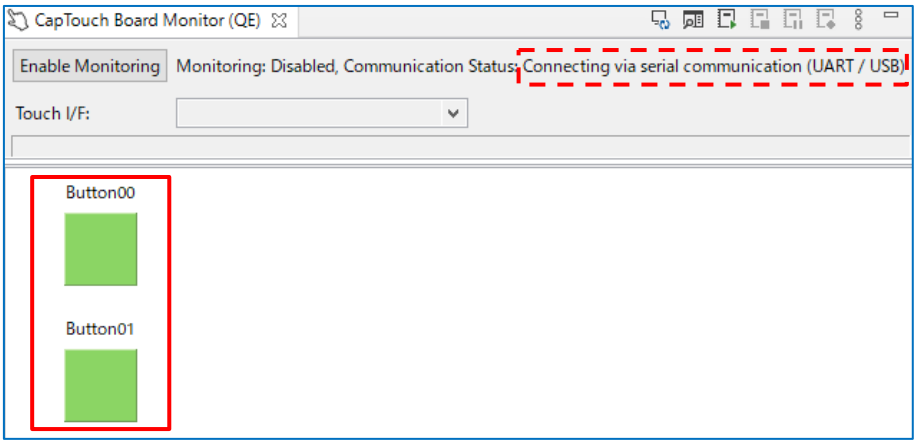
4. When serial connection is executed, the following display will be changed. Confirm a displaying message.



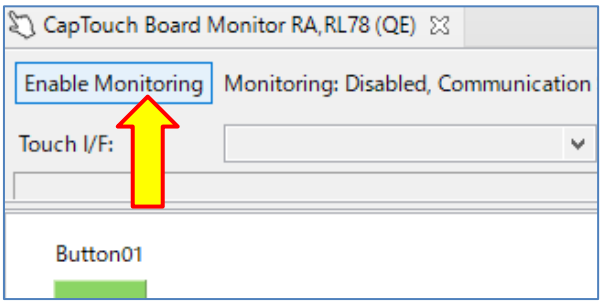
5. Open the Monitoring view from **Show Views** of **CapTouch Main (QE)** / in QE V3.2.0 or later, **CapTouch Workflow (QE)**.



6. **CapTouch Board Monitor (QE)** pane will appear as shown below**.**

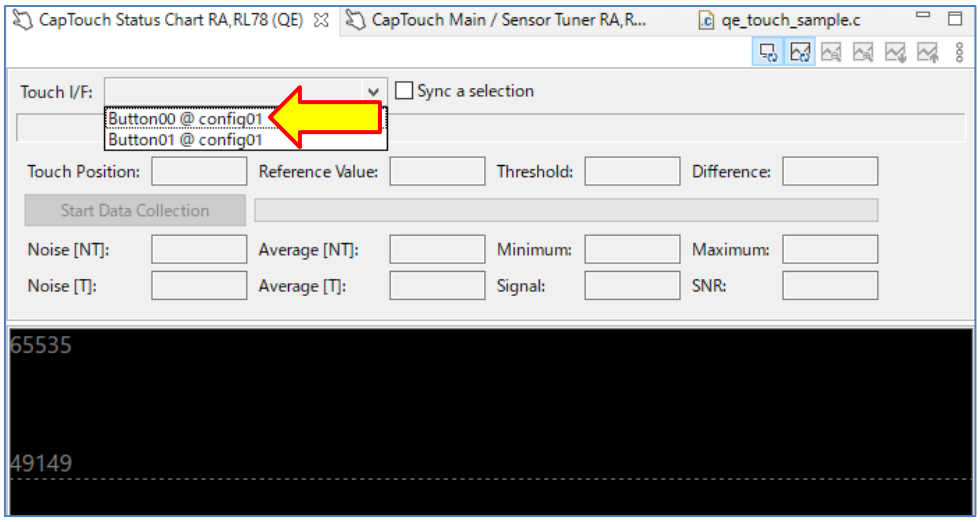


7. Click the **Enable Monitoring** button. The dialog text will change to **Monitoring: Enabled**.

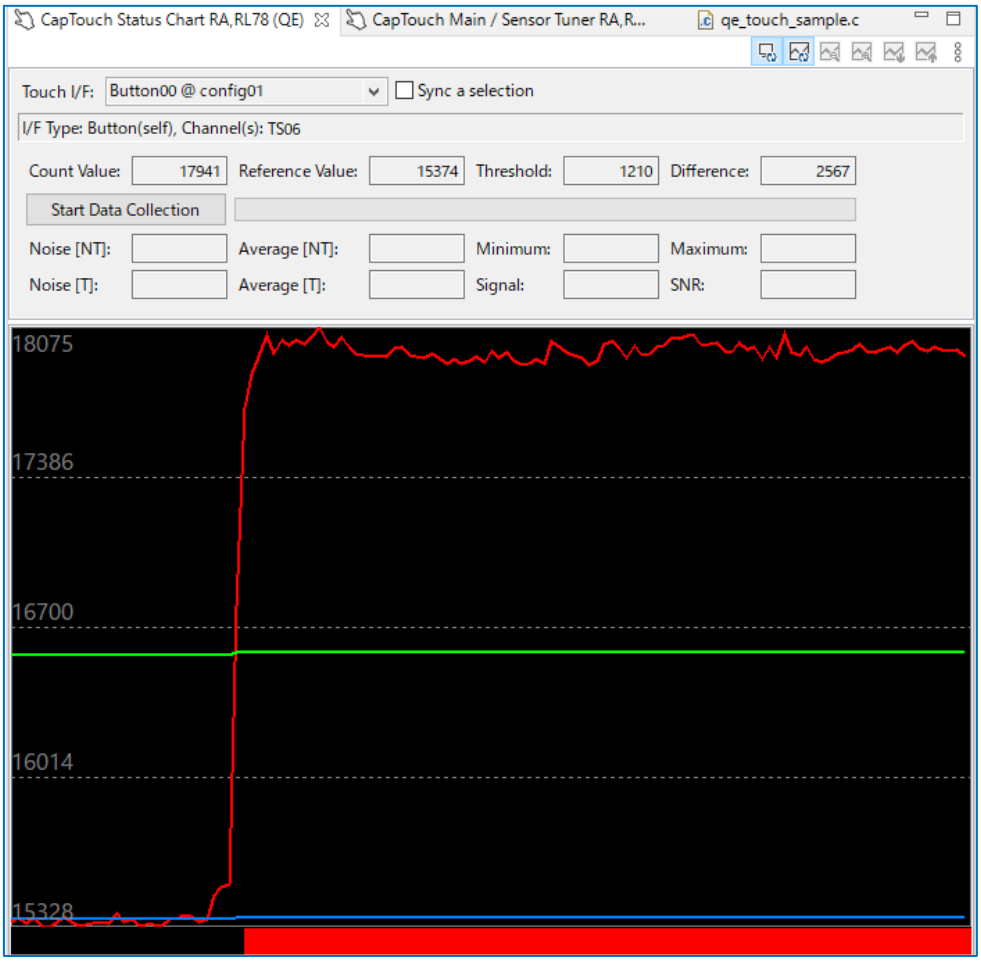


8. Touch the button **TS06** on the target board. The **CapTouch Board Monitor (QE)** will show a touch with a finger image on the button like the below image.

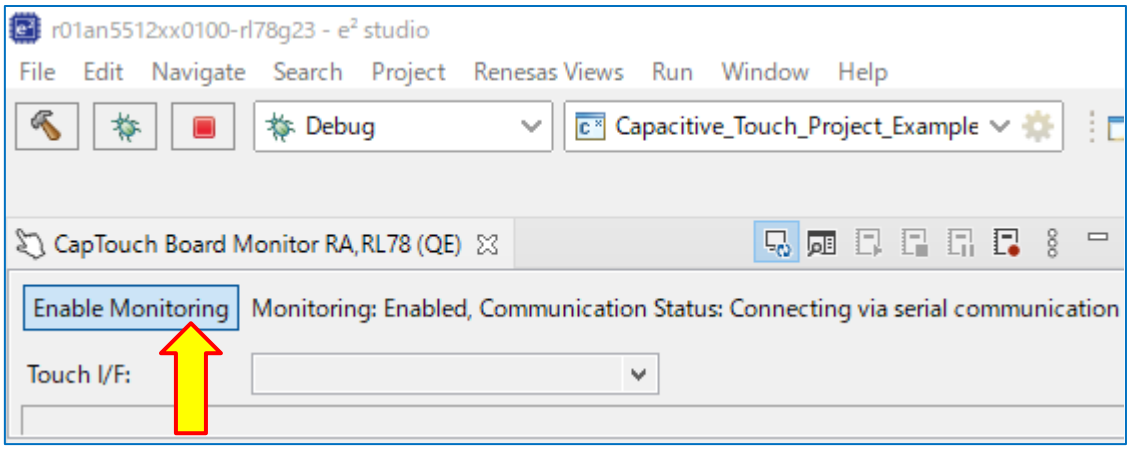
9. To see a graphical representation of the ‘touch counts’ from the board, use the **CapTouch Status Chart (QE)**. Using the pulldown, select **Button00 @ config01**.



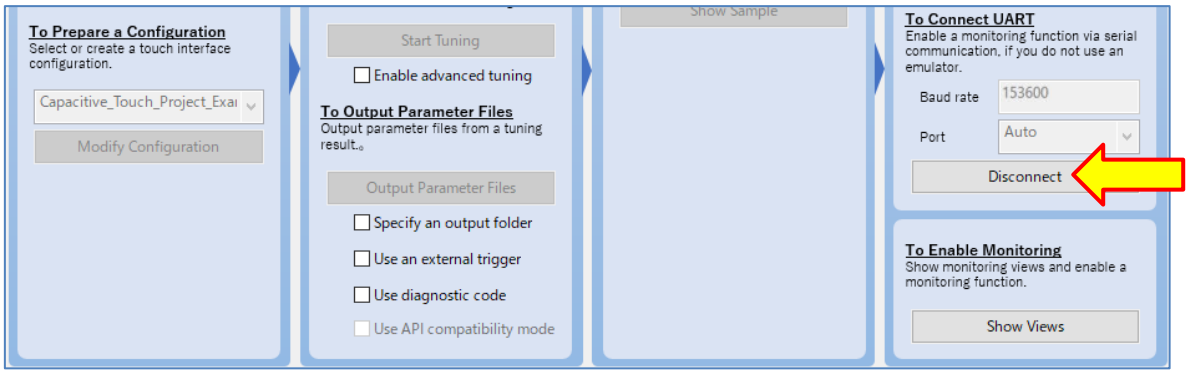
10. The graph will begin to display running values. Touch **TS06** on the board and you should see the ‘touch counts’ show as a step change on the running graph.



11. If you will finish monitoring, click the **Enable Monitoring** button. The dialog text will change to **Monitoring: Disabled**.



12. If you will finish serial connection, click the **Disconnect** button to disconnect from the serial port.



16. qe_touch_sample.c Listing (Example of Using a Software Timer)

The sample program output from QE for Capacitive Touch is shown below.

```
/***********************************************************************
*
* FILE : qe_sample_main.c
* DATE : 2021-07-29
* DESCRIPTION : Main Program for RL78
*
* NOTE:THIS IS A TYPICAL EXAMPLE.
*
***********************************************************************/
#include "qe_touch_config.h"
#define TOUCH_SCAN_INTERVAL_EXAMPLE (20 * 1000)    /* microseconds */

void R_CTSU_PinSetInit(void);
void qe_touch_main(void);
void qe_touch_delay(uint16_t delay_us);

uint64_t button_status;
#if (TOUCH_CFG_NUM_SLIDERS != 0)
uint16_t slider_position[TOUCH_CFG_NUM_SLIDERS];
#endif
#if (TOUCH_CFG_NUM_WHEELS != 0)
uint16_t wheel_position[TOUCH_CFG_NUM_WHEELS];
#endif




void qe_touch_main(void)
{
    fsp_err_t err;

    BSP_ENABLE_INTERRUPT();

    /* Initialize pins (function created by Smart Configurator) */
    R_CTSU_PinSetInit();




    /* Open Touch middleware */
    err = RM_TOUCH_Open(g_qe_touch_instance_config01.p_ctrl, g_qe_touch_instance_config01.p_cfg);
    if (FSP_SUCCESS != err)
    {
        while (true) {}
    }
```

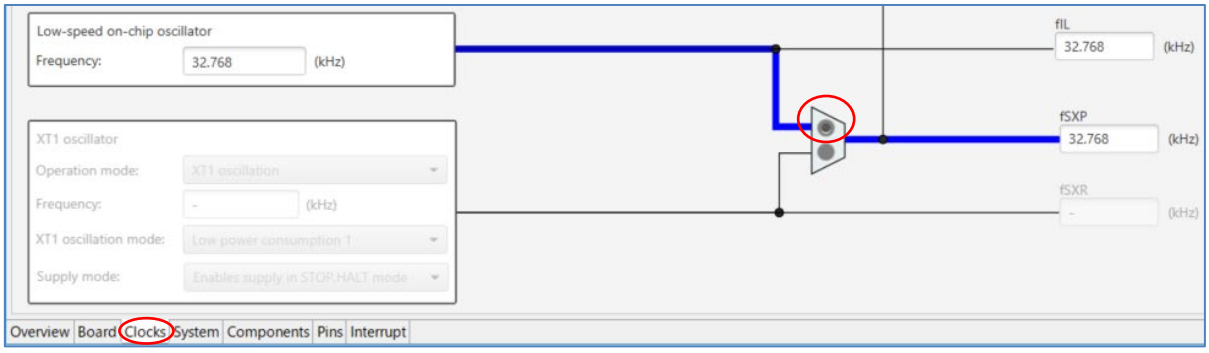
```
    /* Main loop */
    while (true)
    {


        /* 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. */
        qe_touch_delay(TOUCH_SCAN_INTERVAL_EXAMPLE);
    }
}

void qe_touch_delay(uint16_t delay_us)
{
    uint32_t i;
    uint32_t loops_required;
    uint16_t clock_mhz;

    clock_mhz = (uint16_t)(R_BSP_GetFclkFreqHz() / 1000000);
    if (0 == clock_mhz)
    {
        clock_mhz = 1;
    }

    loops_required = ((uint32_t)delay_us * (uint32_t)clock_mhz);
    loops_required /= 20;
    for (i = 0; i < loops_required; i++)
    {
        BSP_NOP();
    }
}
```

17. [Appendix] Touch Measurement by Hardware Timer

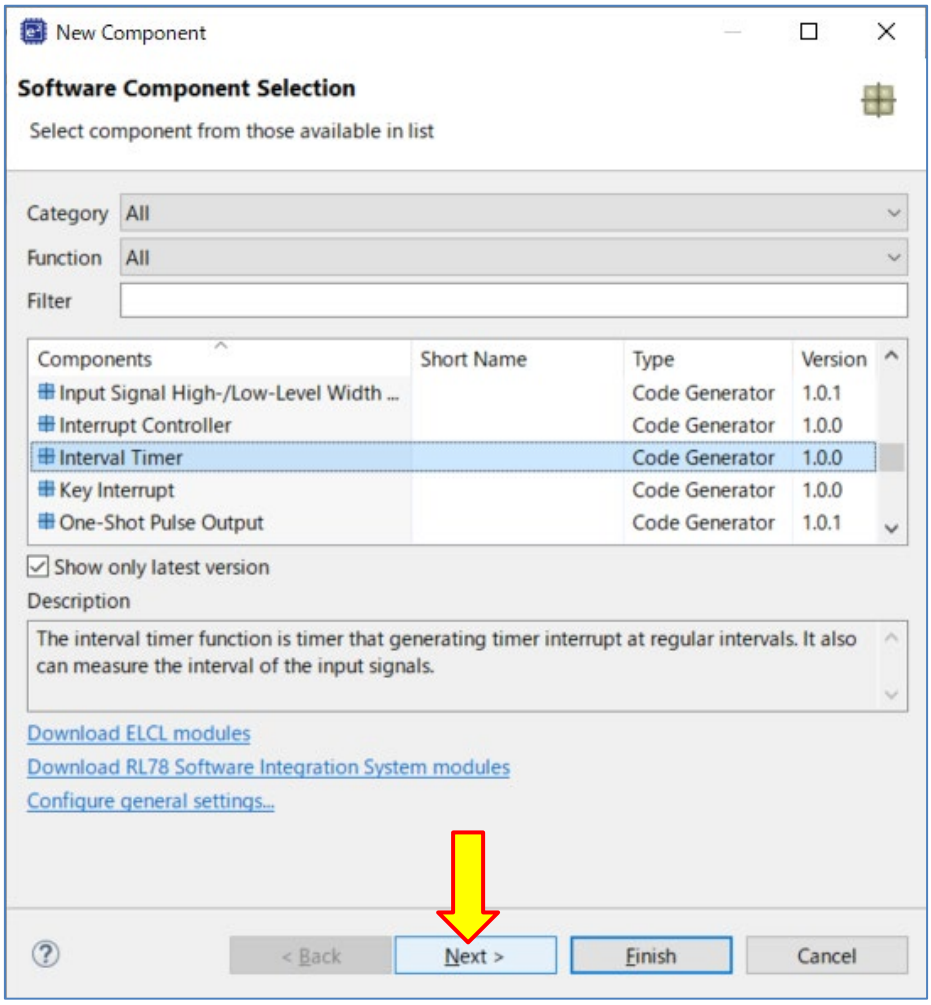
This section describes an example of implementing a touch measurement cycle using a hardware timer.

17.1 Procedure for Setting the Hardware Timer

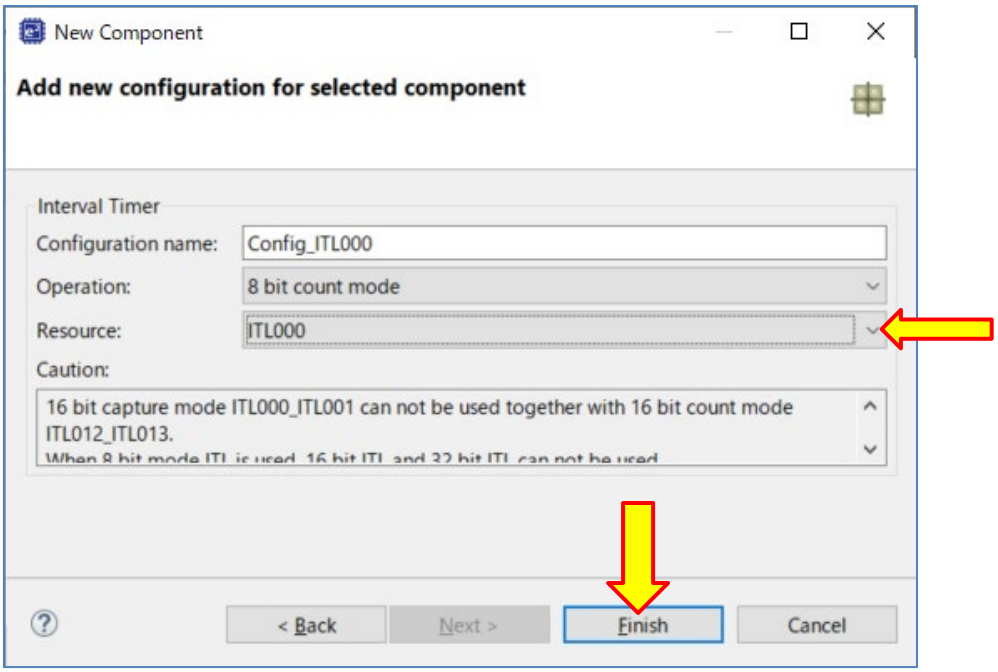
1. Set the clock to be used for the touch measurement cycle as shown below.



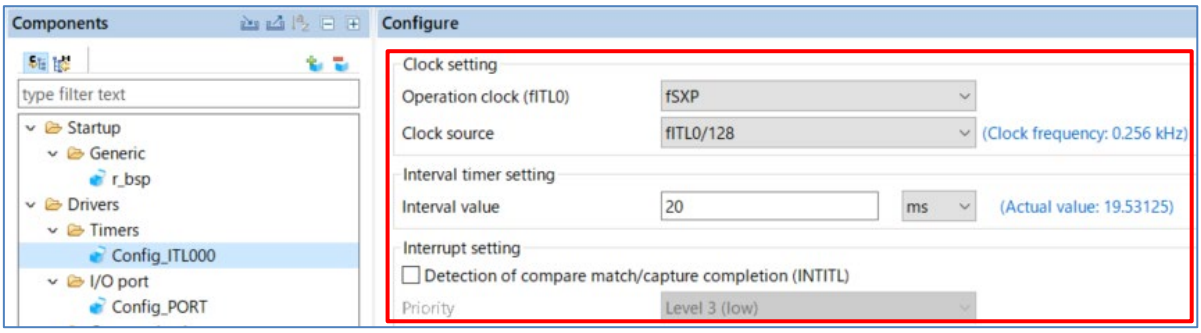
2. In the Software Component Selection dialog, single-click the **Interval Timer** module and click **Next** in the lower part of that open dialog box.



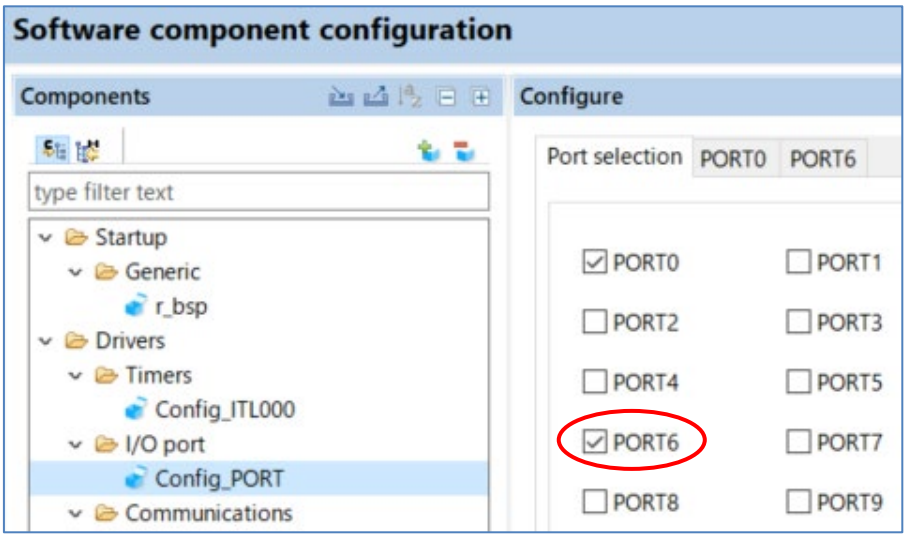
3. Set the interval timer configuration as shown below, and click **Finish** in the lower part of that open dialog box.

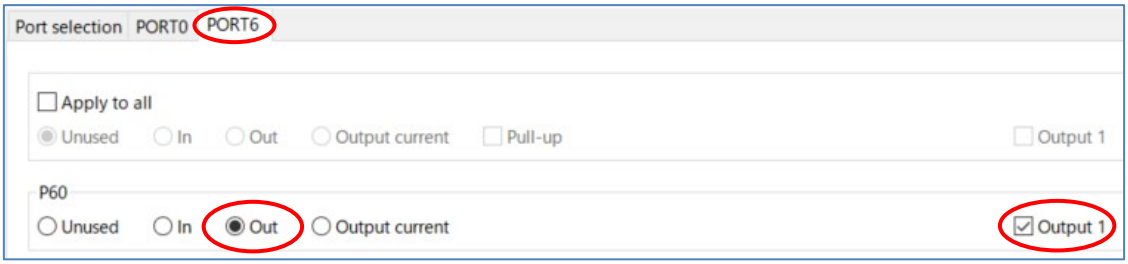


4. Set the interval timer as shown below.

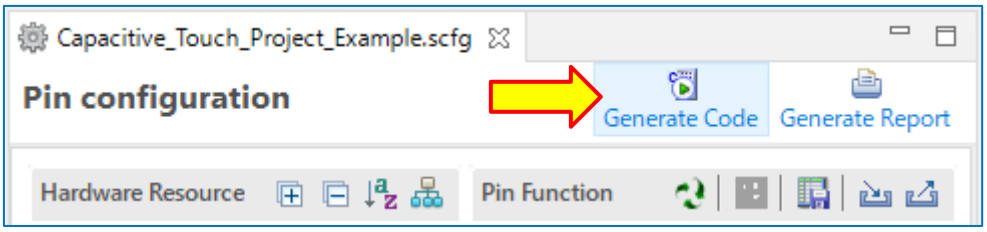


5. Set the port to be used for turning on/off the LED for operation check as shown below.





6. Generate the needed application code modules for the project. Do this by clicking the **Generate Code** icon in the upper-right of the Smart Configurator pane as shown in the picture below.



7. For an example of program implementation, please refer to "17.2 Touch Measurement Program (Example of Using a Hardware Timer)".

17.2 Touch Measurement Program (Example of Using a Hardware Timer)

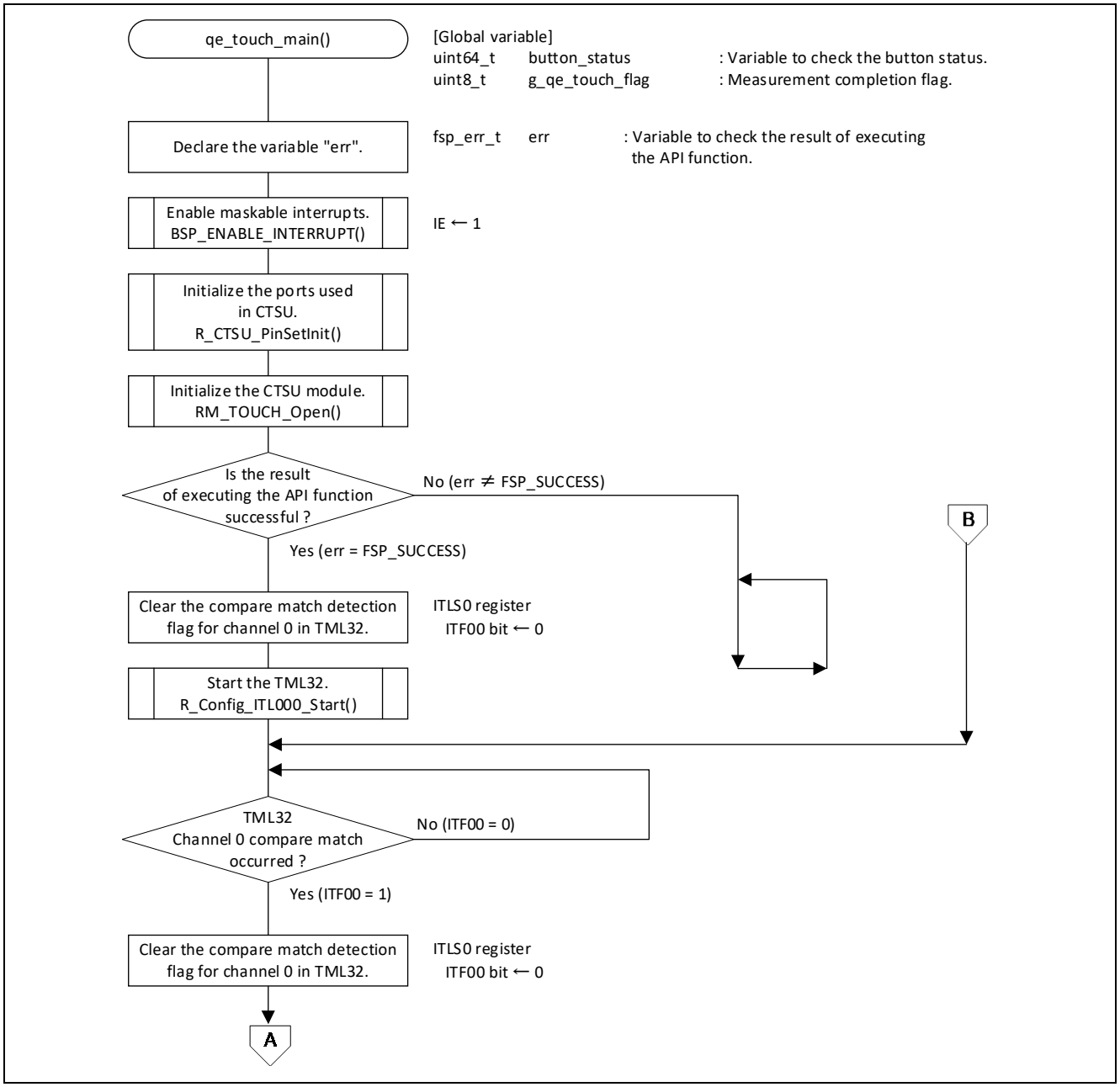
An implementation example of a touch measurement program using a hardware timer is shown below.

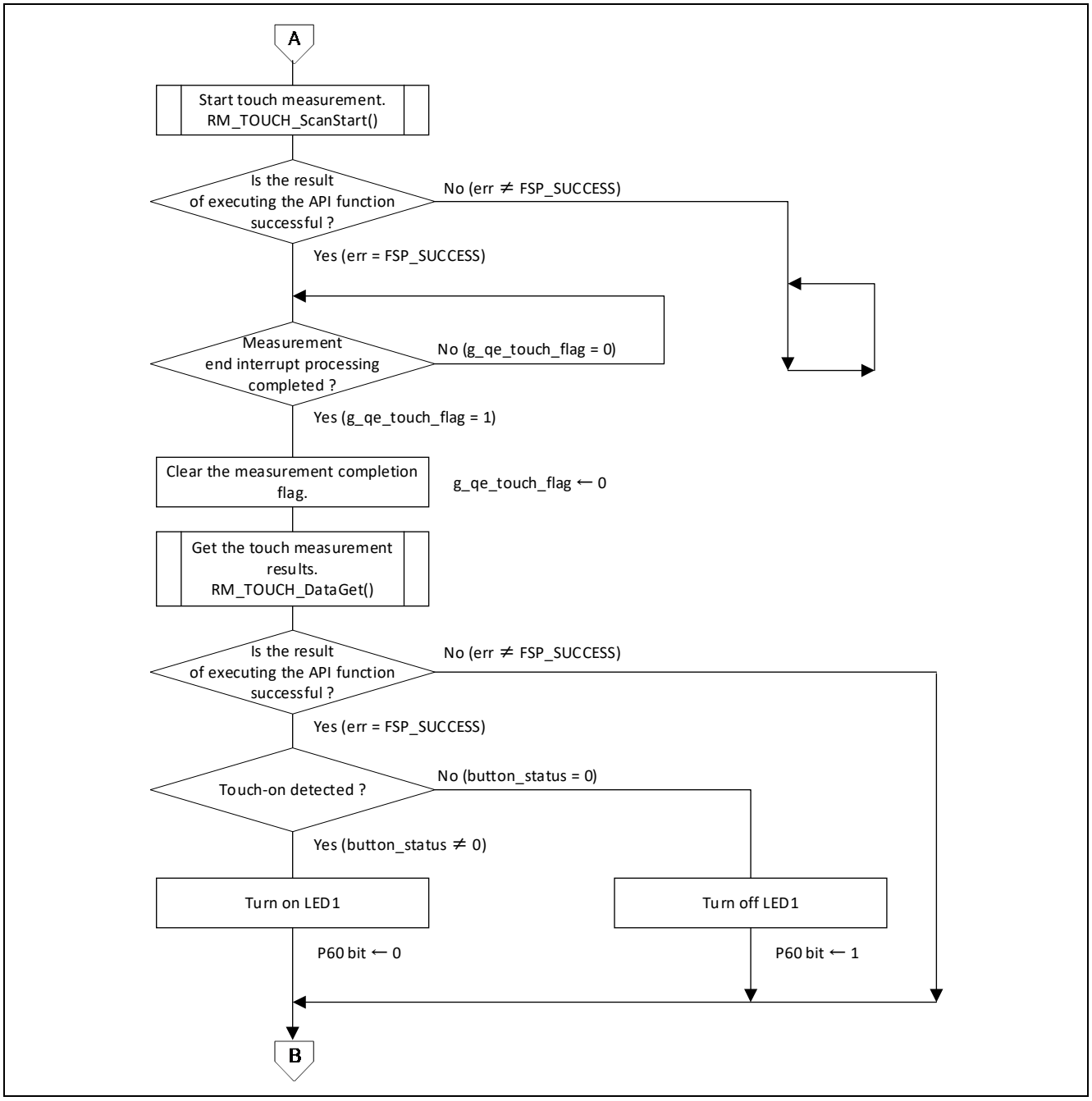
```
/***********************************************************************
*
* FILE : qe_sample_main.c
* DATE : 2022-05-09
* DESCRIPTION : CTSU2L Program for RL78
*
* NOTE:THIS IS A TYPICAL EXAMPLE.
*
***********************************************************************/
#include "qe_touch_config.h"
#include "Config_ITL000.h"

void R_CTSU_PinSetInit(void);
void qe_touch_main(void);

uint64_t button_status;
#if (TOUCH_CFG_NUM_SLIDERS != 0)
uint16_t slider_position[TOUCH_CFG_NUM_SLIDERS];
#endif
#if (TOUCH_CFG_NUM_WHEELS != 0)
uint16_t wheel_position[TOUCH_CFG_NUM_WHEELS];
#endif

void qe_touch_main(void)
{
    fsp_err_t err;

    BSP_ENABLE_INTERRUPT();

    /* Initialize pins (function created by Smart Configurator) */
    R_CTSU_PinSetInit();

    /* Open Touch middleware */
    err = RM_TOUCH_Open(g_qe_touch_instance_config01.p_ctrl, g_qe_touch_instance_config01.p_cfg);
    if (FSP_SUCCESS != err)
    {
        while (true) {}
    }

    ITLS0 &= ~_01_ITL_CHANNEL0_COUNT_MATCH_DETECTE;

    R_Config_ITL000_Start();

    /* Main loop */
    while (true)
    {
        while (_00_ITL_CHANNEL0_COUNT_MATCH_NOT_DETECTE == (ITLS0 & _01_ITL_CHANNEL0_COUNT_MATCH_DETECTE)) {}
        ITLS0 &= ~_01_ITL_CHANNEL0_COUNT_MATCH_DETECTE;

        /* 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 (0 != button_status)
            {
                P6_bit.no0 = 0;
            }
            else
            {
                P6_bit.no0 = 1;
            }
        }
    }
}
```

17.3 Flowcharts (Example of Using a Hardware Timer)

The flowchart of a touch measurement program using a hardware timer is shown below.




A
Start touch measurement.
RM_TOUCH_ScanStart()
Is the result of executing the API function successful ?
No (err ≠ FSP_SUCCESS)
Yes (err = FSP_SUCCESS)
Measurement end interrupt processing completed ?
No (g_qe_touch_flag = 0)
Yes (g_qe_touch_flag = 1)
Clear the measurement completion flag.
g_qe_touch_flag ← 0
Get the touch measurement results.
RM_TOUCH_DataGet()
Is the result of executing the API function successful ?
No (err ≠ FSP_SUCCESS)
Yes (err = FSP_SUCCESS)
Touch-on detected ?
No (button_status = 0)
Yes (button_status ≠ 0)
Turn on LED1
Turn off LED1
P60 bit ← 0
P60 bit ← 1
B

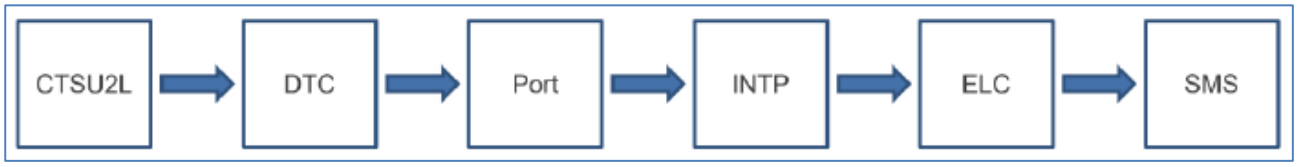
18. [Applied Usage] Setting The Automatic Judgement (using SMS) and MEC functions

Note: The "Automatic Judgement (using SMS) and MEC functions" are supported by products equipped with CTSU2La.

This Chapters shown below describe setting The Automatic Judgement (using SMS) and MEC (Multi Electrode Connection) functions of the RL78/G22.

1. In the case of RL78/G22, module flow used for SMS measurement

The following figure shows the flow of modules used for SMS measurement with RL78/G22. Port output using DTC from CTSU2La. An interrupt signal is generated using the signal output from the port. An interrupt signal triggers the ELC to start SMS processing.

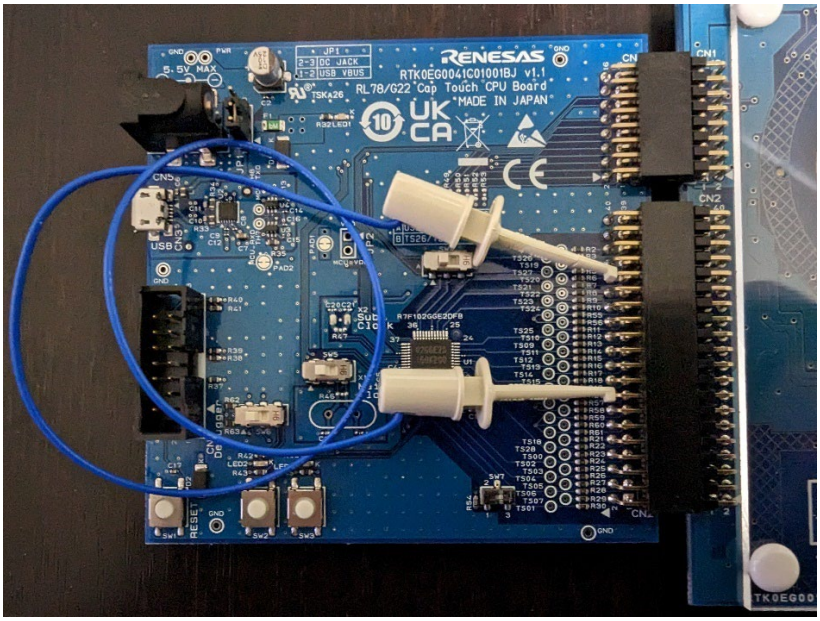


2. Target board

The following explanation is based on the use of RL78/G22 Capacitive Touch Evaluation System (Model No.: RTK0EG0042S01001BJ).

3. Connection of external trigger

To perform automatic judgment measurement using SMS with RL78/G22, please make the following settings.
Connect the CN2 32th pin (P22/TS20) and 16th pin (P16/INTP5/TS17) on the MCU board side as follows.

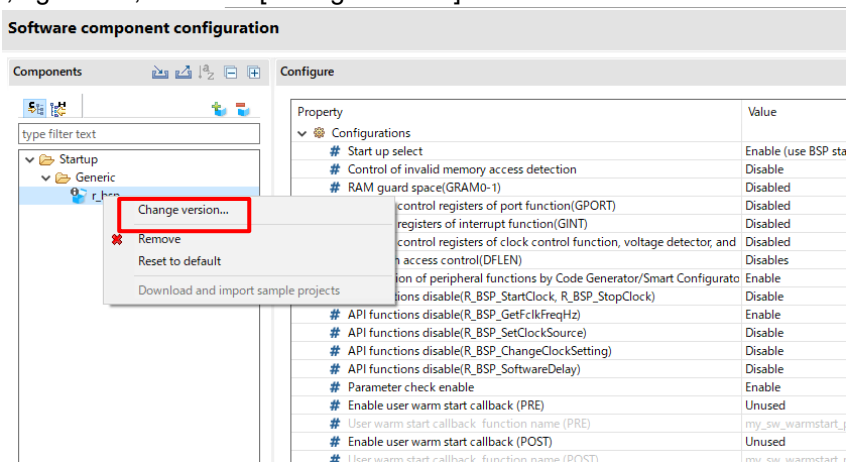


4. Create a project and add modules following the same procedure as described in the “6. Project Creation” and “7. Using Smart Configurator to Add Modules” chapters.

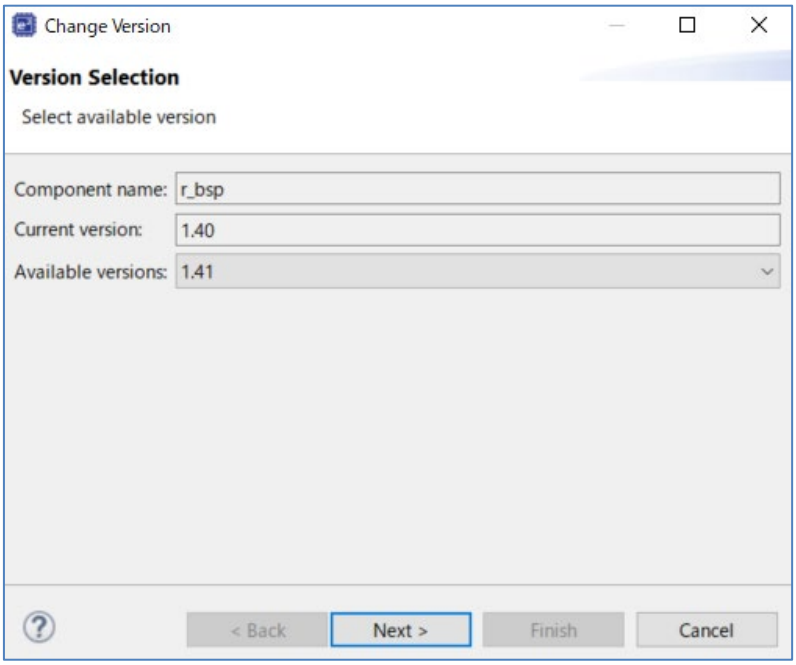
5. Add Component

5.1. r_bsp module

Select the [Components] tab and check the version of r_bsp.
Select r_bsp, right-click, and click [Change version].

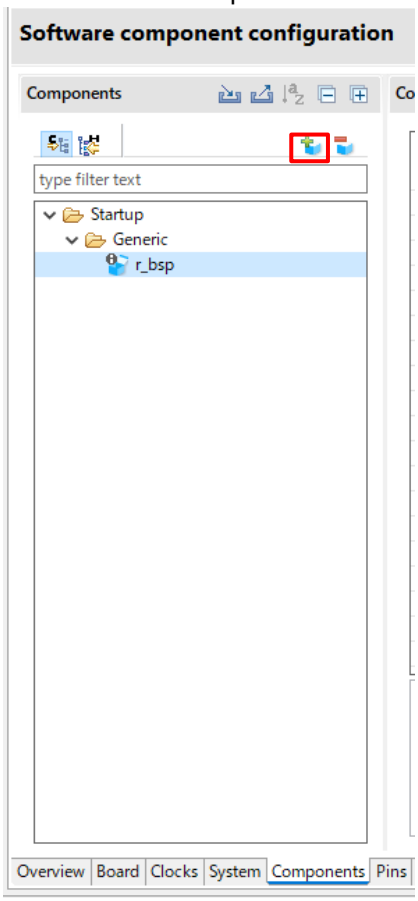


If [Current version] is different from 1.40, select 1.40 or later in [Available versions:] and click [Next] to change the version.

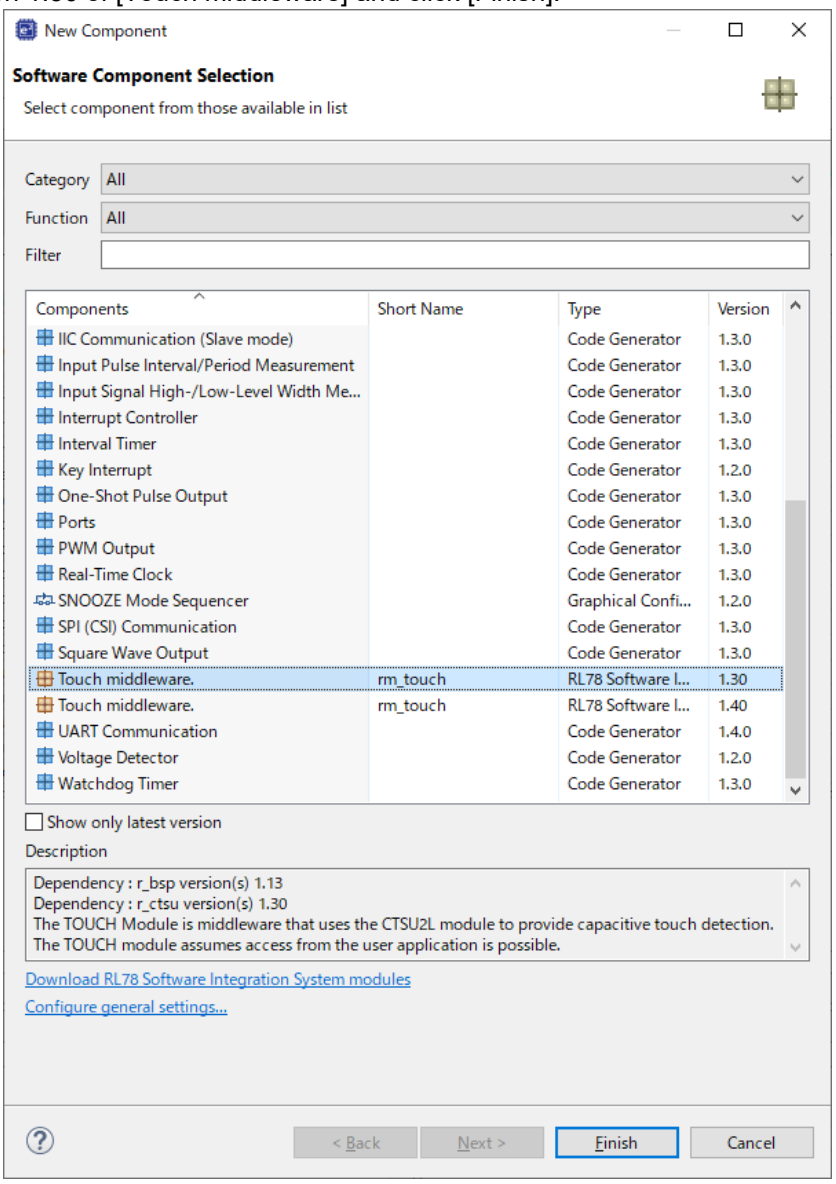


5.2. rm_touch module, r_ctsu module

Click the Add Component button and select the component to add from the list.



Select version 1.30 of [Touch middleware] and click [Finish].



When adding rm_touch, the dependent module r_ctsu is also added at the same time.

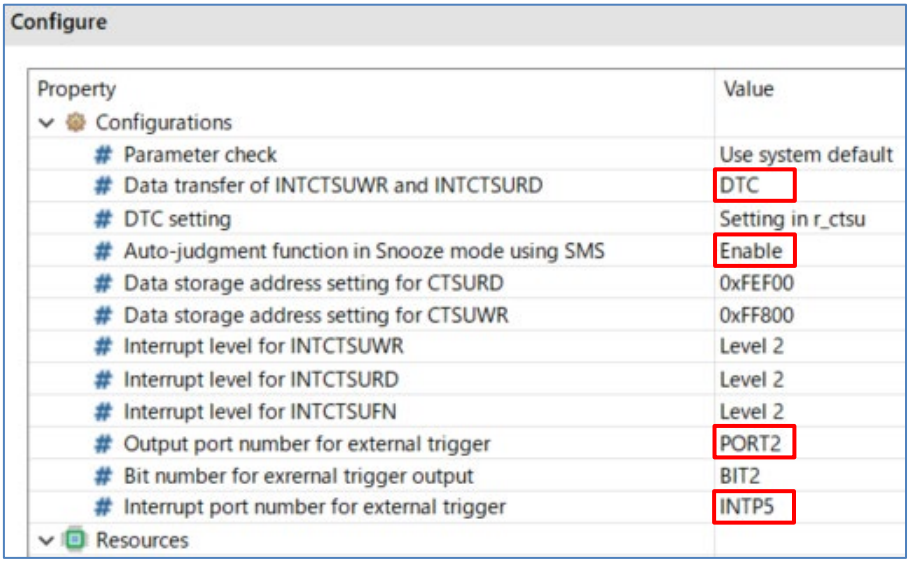
Select r_ctsu, open [Configure] and change the following settings.

[Data transfer of INTCTSUWR and INTCTSURD] select [DTC].

[Auto-judgment function in Snooze mode using SMS] select [Enable].

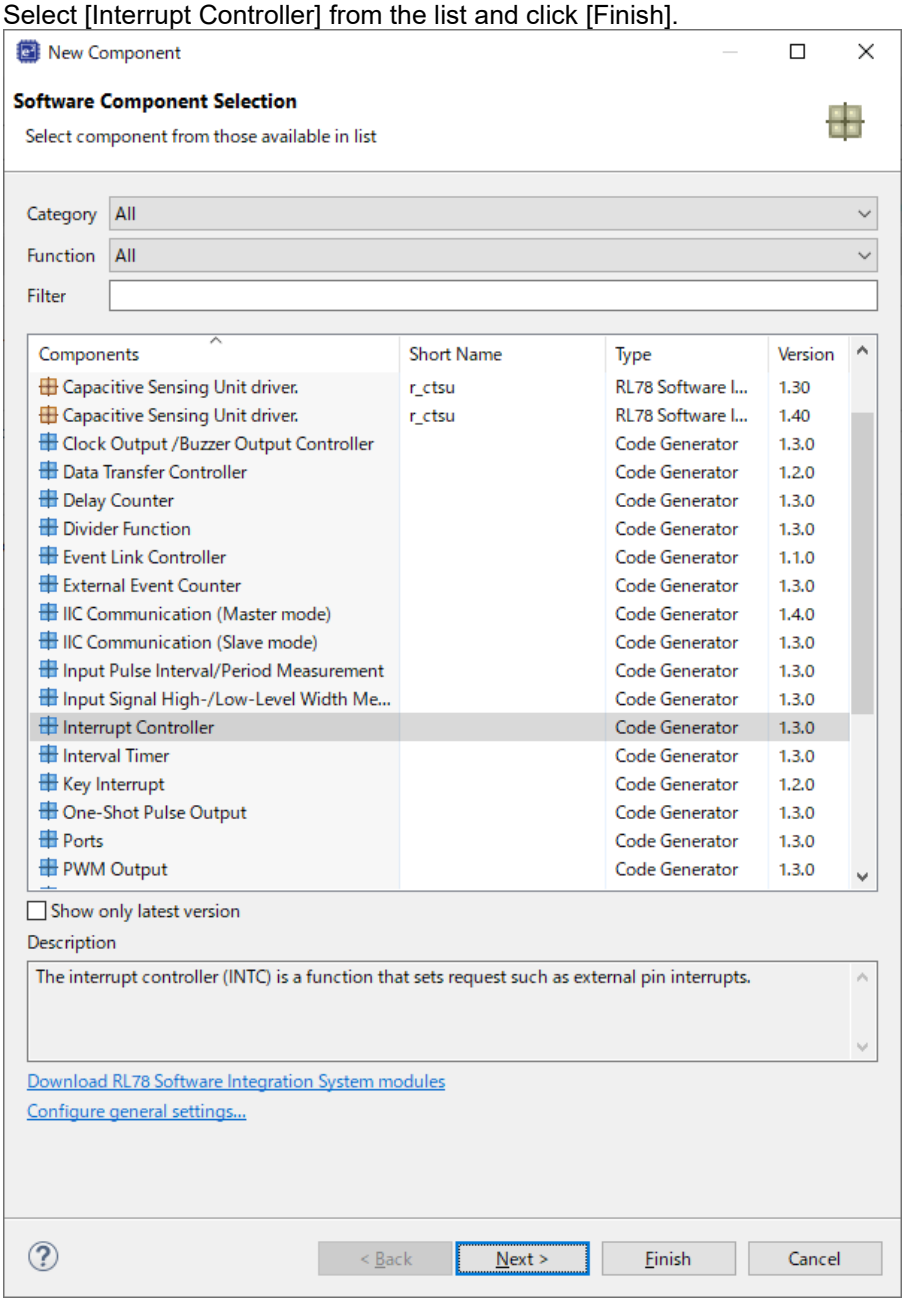
Set the output port in [Output port number for external trigger].

Set the interrupt signal in [Interrupt port number for external trigger].

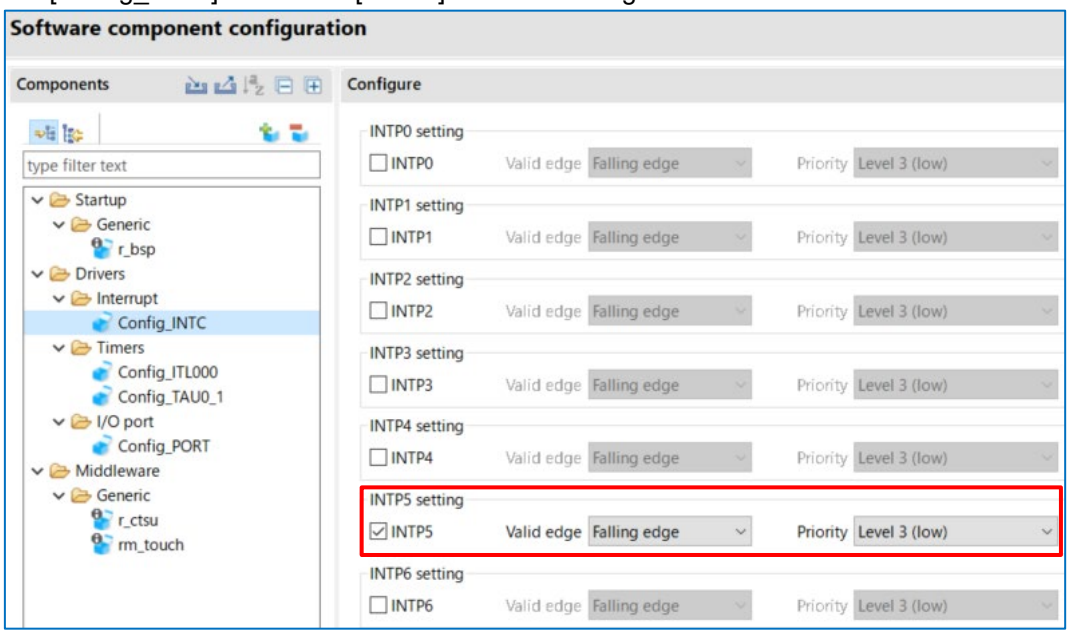


5.3. Adding an Interrupt Controller Component

Select [Interrupt Controller] from the list and click [Finish].

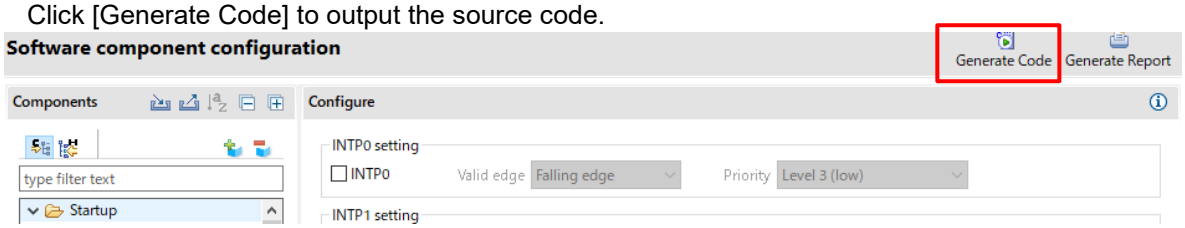


Select [Config_INTC] and check [INTP5] from the settings.



5.4. Source code generation and building

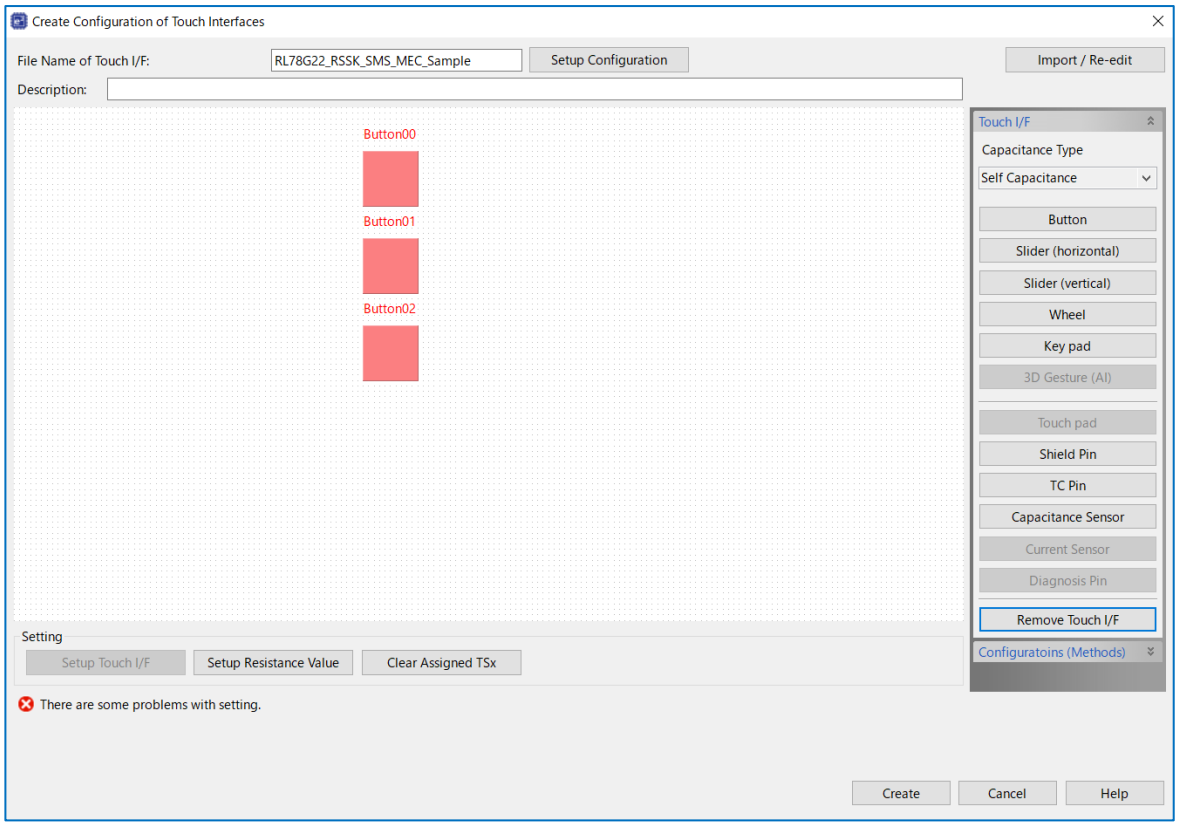
Click [Generate Code] to output the source code.



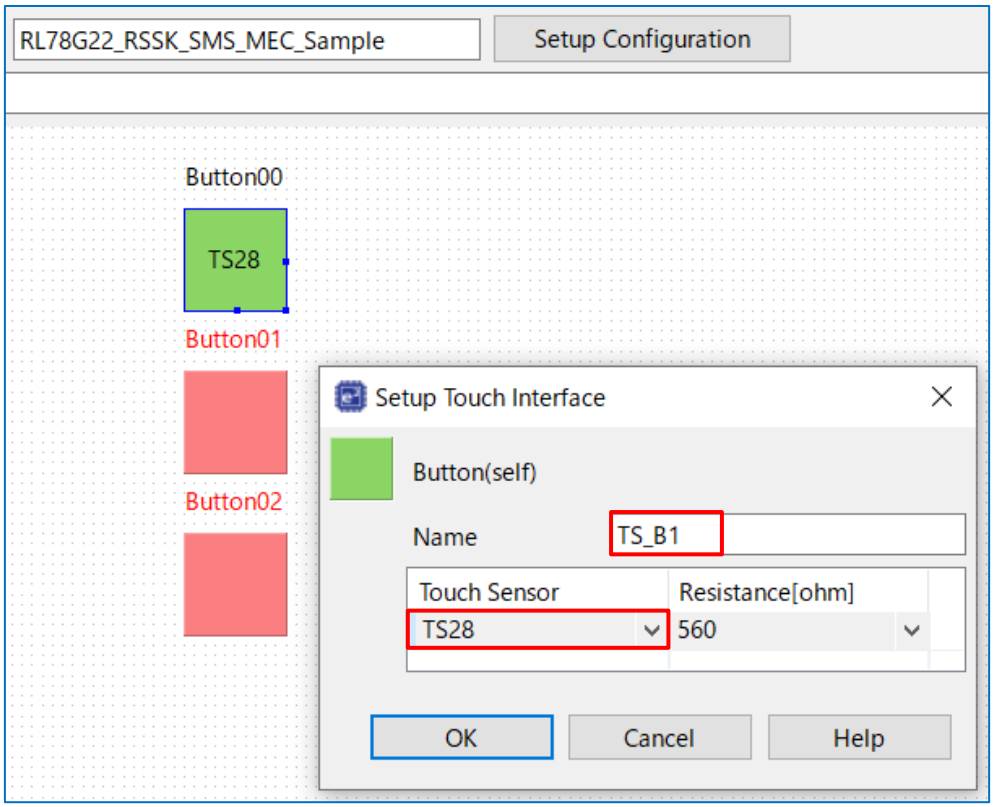
6. Create Touch interface

Set up the touch interface following the same procedure as described in chapter "9. Creating the Capacitive Touch Interface".

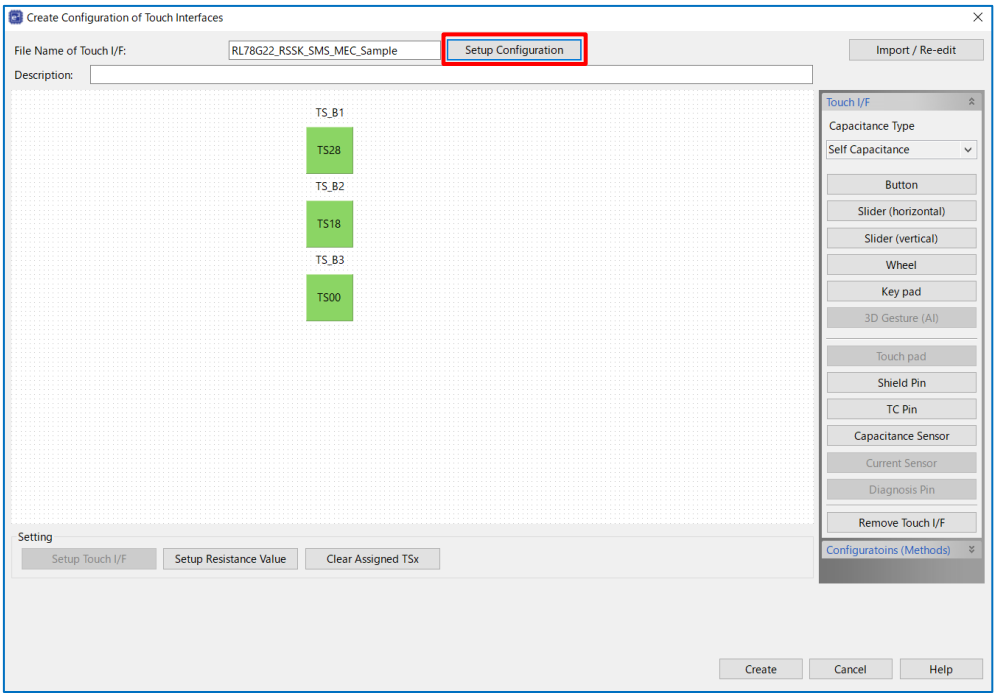
Since some ports are used as external triggers for SMS activation, the ports and corresponding touch sensor wheels and buttons are not available. Also, since we will be using three electrodes as MECs this time, we will place the buttons as shown below.
Select [Button] from the list on the right, place it on the screen, and press the [Esc] key to exit placement mode.



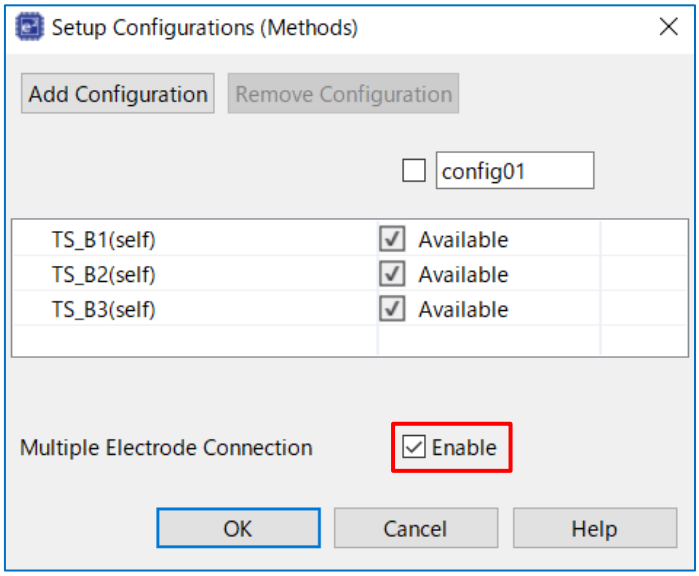
Assign each button name and touch sensor.
Double-click the placed button to open the setting window, change touch sensor and click [OK].



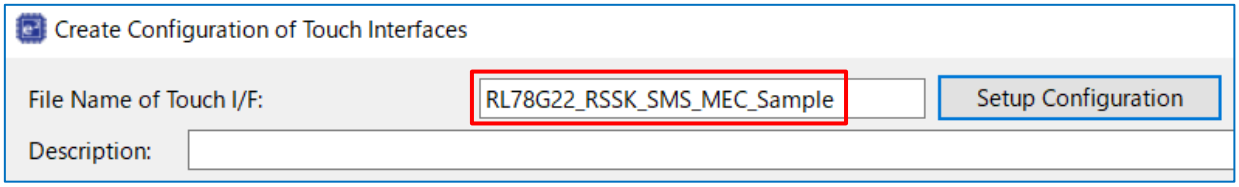
Set a name and touch sensor for each button as follows.
After completing the settings, click [Setup Configuration].



Enable [Multiple Electrode Connection] , and click [OK].



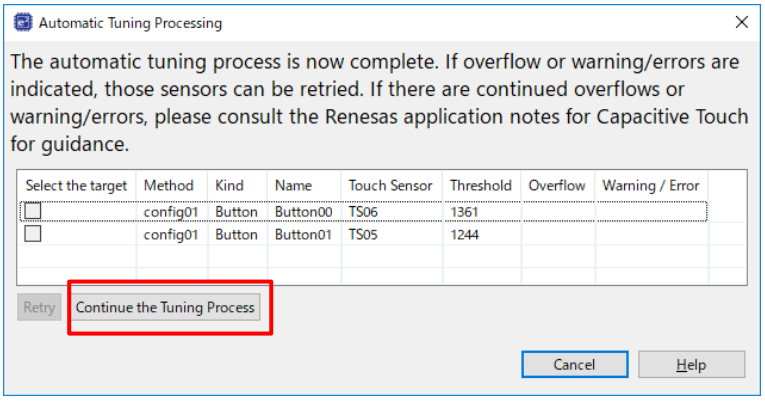
Finally, set the [File Name of Touch I/F] (red frame) and click the [Create] button.



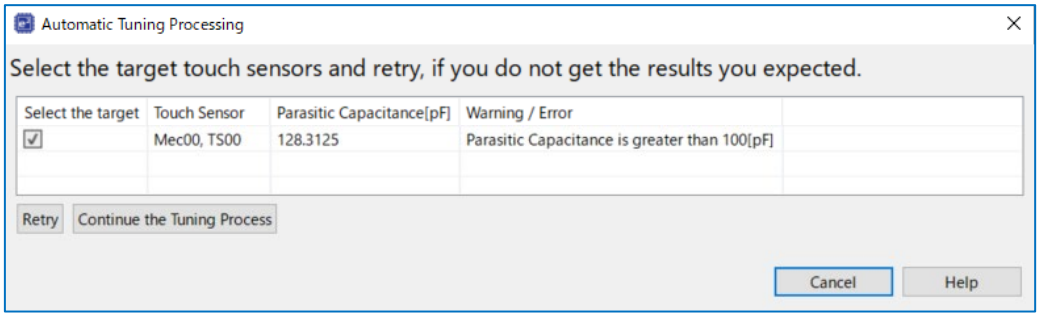
7. QE tuning

Capacitive touch sensor tuning can be performed by following the same procedure as described in chapter "10. Modifying the e^2 studio Project Debug Session for Capacitive Touch Tuning" and thereafter.

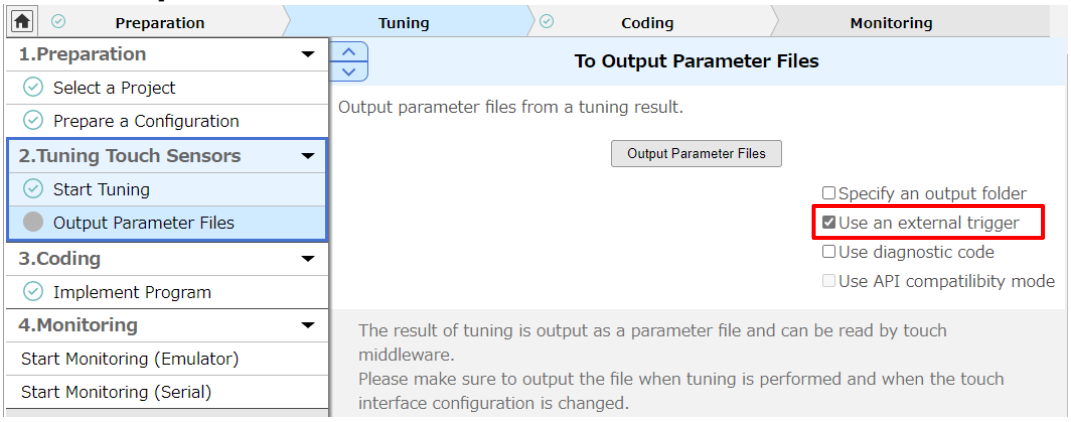
When the adjustment is completed, the following window will appear. Click [Continue the Tuning Process] to finish.



During the process, you may receive a warning about the measured parasitic capacity as shown below, but please click "Continue the Tuning Process" to proceed.



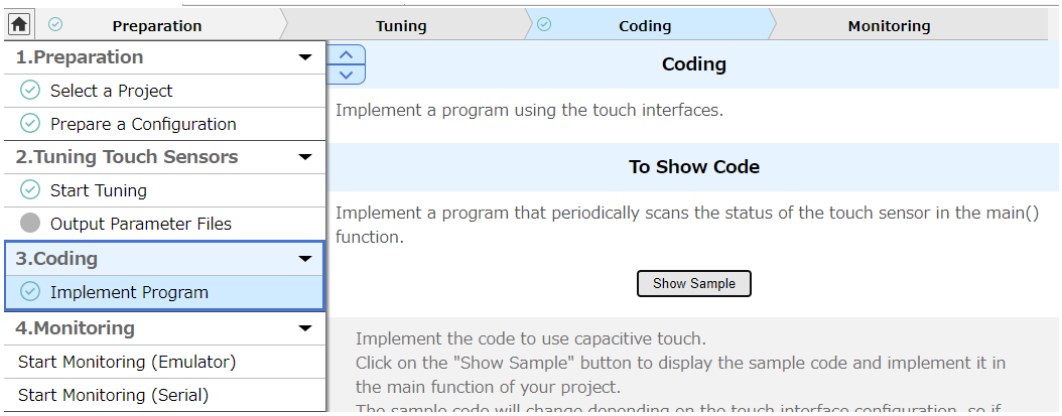
Select [Output Parameter Files] in the workflow, check [Use the external trigger] , and click the [Output Parameter Files] button.



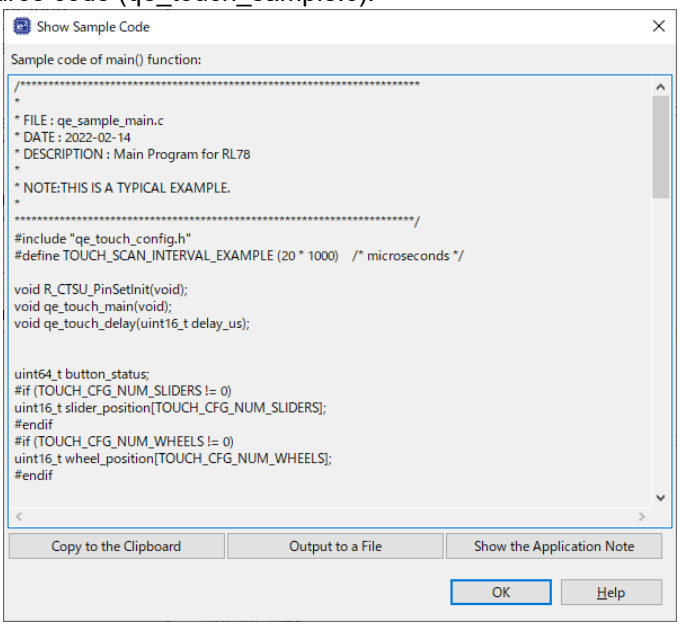
This creates “qe_gen” folder in the project and outputs the source code with defined settings.

8. Sample code

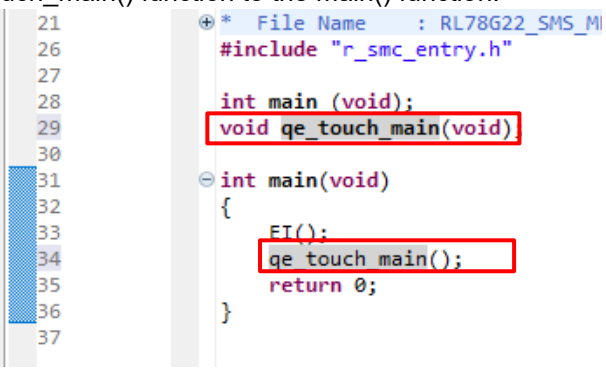
Select [Implement Program] from the workflow and click the [Show Sample] button.



A code example of the qe_touch_main() function is displayed as shown below, and it is possible to output it under “qe_gen” folder, but the code output here is not compatible with external triggers or RL78 STOP mode. Therefore, output “qe_gen/qe_touch_sample.c” using [Output to a File] button, and then overwrite the source code (qe_touch_sample.c).



Add a call to the qe_touch_main() function to the main() function.



Details of overwriting sample code

```
void qe_touch_main(void)
{
    fsp_err_t err;
    uint16_t b_status = 0;

    BSP_ENABLE_INTERRUPT();

    /* Initialize pins (function created by Smart Configurator) */
    R_CTSU_PinSetInit();

    /*    LED    */
    /* P62 GPIO Low */
    PM6_bit.no2 = 0;
    LED2 = LED_OFF;
    /* P63 GPIO Low */
    PM6_bit.no3 = 0;
    LED3 = LED_OFF;

    /* P140 setting */
    PMCT14_bit.no0 = 0;
    PM14_bit.no0 = 0;
    P14_bit.no0 = 0;

    /* Open Touch middleware */
    err = RM_TOUCH_Open(g_qe_touch_instance_config01.p_ctrl, g_qe_touch_instance_config01.p_cfg);
    if (FSP_SUCCESS != err)
    {
        while (true) {}
    }

    MK2H |= 0x40;    /* Disable interrupt servicing of "write request interrupt for setting registers for each channel" */
    MK3L |= 0x01;    /* Disable interrupt servicing of "measurement data transfer request interrupt" */

    /* Select the event source for the CTSU in the ELCL */
    ELSELR10 = 0x06;    /* ELCITL0(32bit interval timer0) -> CTSU */
```

Initialize CTSU PINS

Initialize CPU BOARD LED

Initialize PORT using SMS

Reduce standby power consumption

ELC setting for CTSU measurement triggered by interval timer

```
    /* initial offset tuning */
    /* for [CONFIG01] configuration */
    err = RM_TOUCH_ScanStart(g_qe_touch_instance_config01.p_ctrl);
    if (FSP_SUCCESS != err)
    {
        while (true) {}
    }

    g_qe_touch_flag = 0;

    ITLS0 &= (uint8_t)~_01_ITL_CHANNEL0_COUNT_MATCH_DETECTE;

    R_Config_ITL000_Start();

    /* Main loop */
    while (true)
    {
        __stop();

        ITLS0 &= (uint8_t)~_01_ITL_CHANNEL0_COUNT_MATCH_DETECTE;

        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)
        {
            R_Config_ITL000_Stop();
            break;
        }
    }

    button_status = 0;
    g_qe_touch_flag = 0;

    LED2 = LED_ON;
    LED3 = LED_ON;
```

Initial Offset tuning Process

Flag clear

CPU BOARD LED turn on

```
/* normal snooze mode */
/* for [CONFIG01] configuration */
err = RM_TOUCH_ScanStart(g_qe_touch_instance_config01.p_ctrl);
if (FSP_SUCCESS != err)
{
    while (true) {}
}

g_qe_touch_flag = 0;

ITLS0 &= (uint8_t)~_01_ITL_CHANNEL0_COUNT_MATCH_DETECTE;

R_Config_ITL000_Start();

while (true)
{
    __stop();

    ITLS0 &= (uint8_t)~_01_ITL_CHANNEL0_COUNT_MATCH_DETECTE;

    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)
    {
        if(1 == button_status)
        {
            LED2 = LED_ON;
            LED3 = LED_OFF;
            b_status = 1;
        }

        if (b_status == 1)
        {
            if (!button_status)
            {
                R_Config_ITL000_Stop();
                LED2 = LED_ON;
                LED3 = LED_ON;
                b_status = 0;
                break;
            }
        }
    }
}
```

SNOOZE mode scan process

```
/* SMS setting */
err = RM_TOUCH_SmsSet(g_qe_touch_instance_config01.p_ctrl);
err = RM_TOUCH_ScanStart(g_qe_touch_instance_config01.p_ctrl);
if (FSP_SUCCESS != err)
{
    while (true) {}
}

R_Config_INTC_INTP1_Start();
R_Config_ITL000_Start();

while (true)
{
    while (true)
    {
        __stop();

        /* When the threshold is exceeded, callback function is called. */
        if (g_qe_touch_flag)
        {
            break;
        }
    }
    err = RM_TOUCH_DataGet(g_qe_touch_instance_config01.p_ctrl, &button_status, NULL, NULL);
    if (FSP_SUCCESS == err)
    {
        if (!button_status)
        {
            LED2 = LED_ON;
            LED3 = LED_OFF;

        }
```

SNOOZE + SMS mode Scan process

LED extinguishing process after waking up from SNOOZE + SMS mode

```
            RM_TOUCH_ScanStart(g_qe_touch_instance_config01.p_ctrl);
            LED2 = LED_ON;
            LED3 = LED_ON;

            while (true)
            {
                __stop();

                ITLS0 &= (uint8_t)~_01_ITL_CHANNEL0_COUNT_MATCH_DETECTE;

                while (0 == g_qe_touch_flag) {}    /* Wait until the interrupt servicing of "measurement end interrupt" is completed */
                g_qe_touch_flag = 0;

                err = RM_TOUCH_DataGet(g_qe_touch_instance_config01.p_ctrl, &button_status, NULL, NULL);
                if (FSP_SUCCESS == err)
                {
                    if (button_status)
                    {
                        LED2 = LED_OFF;
                        LED3 = LED_ON;
                        b_status = 1;
                    }
```

Snooze mode scan

LED turn off process in ActiSnooze mode scan

```
                    if (b_status == 1)
                    {
                        if (!button_status)
                        {
                            R_Config_ITL000_Stop();
                            R_Config_INTC_INTP1_Stop();
                            LED2 = LED_ON;
                            LED3 = LED_ON;
                            b_status = 0;

                            /* Start SMS measurement */
                            RM_TOUCH_SmsSet(g_qe_touch_instance_config01.p_ctrl);
                            err = RM_TOUCH_ScanStart(g_qe_touch_instance_config01.p_ctrl);
                            if (FSP_SUCCESS != err)
                            {
                                while (true) {}
                            }

                            R_Config_INTC_INTP1_Start();
                            R_Config_ITL000_Start();
                            break;
                        }
                    }
                }
            }
        }
    }
}
```

Return processing to SNOOZE + SMS mode

19. Documents for Reference

RL78/Gxx User's Manual: Hardware

RL78 Family User's Manual: Software (R01US0015)

(The latest versions of the documents are available on the Renesas Electronics Website.)

Technical Updates/Technical Brochures

(The latest versions of the documents are available on the Renesas Electronics Website.)

Application Note RL78 Family Capacitive Touch Sensing Unit (CTSU2L) Operation Explanation (R01AN5744)

Application Note RL78 Family CTSU Module Software Integration System (R11AN0484)

Application Note RL78 Family TOUCH Module Software Integration System (R11AN0485)

Application Note Capacitive Sensor Microcontrollers
CTSU Capacitive Touch Electrode Design Guide (R30AN0389)

Application Note RL78 Family
RL78/G23 Capacitive Touch Low Power Guide (SNOOZE function) (R01AN5886)

Application Note RL78/G23 Group
RL78/G23 Capacitive Touch Low Power Guide (SMS function) (R01AN6670)

Application Note RL78/G22 Capacitive Touch Low Power Guide (SMS / MEC function) (R01AN6847)

(The latest versions of the documents are available on the Renesas Electronics Website.)

Website and Support

Renesas Electronics Website
http://www.renesas.com/

Capacitive Sensing Unit related page
https://www.renesas.com/solutions/touch-key
https://www.renesas.com/qe-capacitive-touch

Inquiries
http://www.renesas.com/contact/

Revision History

Rev.	Date	Description	
		Page	Summary
1.00	Apr.30.21	-	First edition issued
2.00	Aug.31.21	-	Updated the development environment
2.10	May.20.22	-	Updated chapter “8. [Additional function] Setting the serial communication monitor using UART (1/3)”. Updated chapter “15. [Additional function] Setting the serial communication monitor using UART (3/3)”. Added chapter “17. [Appendix] Touch Measurement by Hardware Timer”.
3.00	May.22.23	-	Added chapter “18. [Applied Usage] Setting The Automatic Judgement (using SMS) and MEC functions”.

General Precautions in the Handling of Microprocessing Unit and Microcontroller Unit Products

The following usage notes are applicable to all Microprocessing unit and Microcontroller unit products from Renesas. For detailed usage notes on the products covered by this document, refer to the relevant sections of the document as well as any technical updates that have been issued for the products.

1. Precaution against Electrostatic Discharge (ESD)
   A strong electrical field, when exposed to a CMOS device, can cause destruction of the gate oxide and ultimately degrade the device operation. Steps must be taken to stop the generation of static electricity as much as possible, and quickly dissipate it when it occurs. Environmental control must be adequate. When it is dry, a humidifier should be used. This is recommended to avoid using insulators that can easily build up static electricity. Semiconductor devices must be stored and transported in an anti-static container, static shielding bag or conductive material. All test and measurement tools including work benches and floors must be grounded. The operator must also be grounded using a wrist strap. Semiconductor devices must not be touched with bare hands. Similar precautions must be taken for printed circuit boards with mounted semiconductor devices.
2. Processing at power-on
   The state of the product is undefined at the time when power is supplied. The states of internal circuits in the LSI are indeterminate and the states of register settings and pins are undefined at the time when power is supplied. In a finished product where the reset signal is applied to the external reset pin, the states of pins are not guaranteed from the time when power is supplied until the reset process is completed. In a similar way, the states of pins in a product that is reset by an on-chip power-on reset function are not guaranteed from the time when power is supplied until the power reaches the level at which resetting is specified.
3. Input of signal during power-off state
   Do not input signals or an I/O pull-up power supply while the device is powered off. The current injection that results from input of such a signal or I/O pull-up power supply may cause malfunction and the abnormal current that passes in the device at this time may cause degradation of internal elements. Follow the guideline for input signal during power-off state as described in your product documentation.
4. Handling of unused pins
   Handle unused pins in accordance with the directions given under handling of unused pins in the manual. The input pins of CMOS products are generally in the high-impedance state. In operation with an unused pin in the open-circuit state, extra electromagnetic noise is induced in the vicinity of the LSI, an associated shoot-through current flows internally, and malfunctions occur due to the false recognition of the pin state as an input signal become possible.
5. Clock signals
   After applying a reset, only release the reset line after the operating clock signal becomes stable. When switching the clock signal during program execution, wait until the target clock signal is stabilized. When the clock signal is generated with an external resonator or from an external oscillator during a reset, ensure that the reset line is only released after full stabilization of the clock signal. Additionally, when switching to a clock signal produced with an external resonator or by an external oscillator while program execution is in progress, wait until the target clock signal is stable.
6. Voltage application waveform at input pin
   Waveform distortion due to input noise or a reflected wave may cause malfunction. If the input of the CMOS device stays in the area between V_{IL} (Max.) and V_{IH} (Min.) due to noise, for example, the device may malfunction. Take care to prevent chattering noise from entering the device when the input level is fixed, and also in the transition period when the input level passes through the area between V_{IL} (Max.) and V_{IH} (Min.).
7. Prohibition of access to reserved addresses
   Access to reserved addresses is prohibited. The reserved addresses are provided for possible future expansion of functions. Do not access these addresses as the correct operation of the LSI is not guaranteed.
8. Differences between products
   Before changing from one product to another, for example to a product with a different part number, confirm that the change will not lead to problems. The characteristics of a microprocessing unit or microcontroller unit products in the same group but having a different part number might differ in terms of internal memory capacity, layout pattern, and other factors, which can affect the ranges of electrical characteristics, such as characteristic values, operating margins, immunity to noise, and amount of radiated noise. When changing to a product with a different part number, implement a system-evaluation test for the given product.

Notice

1. Descriptions of circuits, software and other related information in this document are provided only to illustrate the operation of semiconductor products and application examples. You are fully responsible for the incorporation or any other use of the circuits, software, and information in the design of your product or system. Renesas Electronics disclaims any and all liability for any losses and damages incurred by you or third parties arising from the use of these circuits, software, or information.
2. Renesas Electronics hereby expressly disclaims any warranties against and liability for infringement or any other claims involving patents, copyrights, or other intellectual property rights of third parties, by or arising from the use of Renesas Electronics products or technical information described in this document, including but not limited to, the product data, drawings, charts, programs, algorithms, and application examples.
3. No license, express, implied or otherwise, is granted hereby under any patents, copyrights or other intellectual property rights of Renesas Electronics or others.
4. You shall be responsible for determining what licenses are required from any third parties, and obtaining such licenses for the lawful import, export, manufacture, sales, utilization, distribution or other disposal of any products incorporating Renesas Electronics products, if required.
5. You shall not alter, modify, copy, or reverse engineer any Renesas Electronics product, whether in whole or in part. Renesas Electronics disclaims any and all liability for any losses or damages incurred by you or third parties arising from such alteration, modification, copying or reverse engineering.
6. Renesas Electronics products are classified according to the following two quality grades: “Standard” and “High Quality”. The intended applications for each Renesas Electronics product depends on the product’s quality grade, as indicated below.
   "Standard": Computers; office equipment; communications equipment; test and measurement equipment; audio and visual equipment; home electronic appliances; machine tools; personal electronic equipment; industrial robots; etc.
   "High Quality": Transportation equipment (automobiles, trains, ships, etc.); traffic control (traffic lights); large-scale communication equipment; key financial terminal systems; safety control equipment; etc.

   Unless expressly designated as a high reliability product or a product for harsh environments in a Renesas Electronics data sheet or other Renesas Electronics document, Renesas Electronics products are not intended or authorized for use in products or systems that may pose a direct threat to human life or bodily injury (artificial life support devices or systems; surgical implantations; etc.), or may cause serious property damage (space system; undersea repeaters; nuclear power control systems; aircraft control systems; key plant systems; military equipment; etc.). Renesas Electronics disclaims any and all liability for any damages or losses incurred by you or any third parties arising from the use of any Renesas Electronics product that is inconsistent with any Renesas Electronics data sheet, user’s manual or other Renesas Electronics document.
7. No semiconductor product is absolutely secure. Notwithstanding any security measures or features that may be implemented in Renesas Electronics hardware or software products, Renesas Electronics shall have absolutely no liability arising out of any vulnerability or security breach, including but not limited to any unauthorized access to or use of a Renesas Electronics product or a system that uses a Renesas Electronics product. RENESAS ELECTRONICS DOES NOT WARRANT OR GUARANTEE THAT RENESAS ELECTRONICS PRODUCTS, OR ANY SYSTEMS CREATED USING RENESAS ELECTRONICS PRODUCTS WILL BE INVULNERABLE OR FREE FROM CORRUPTION, ATTACK, VIRUSES, INTERFERENCE, HACKING, DATA LOSS OR THEFT, OR OTHER SECURITY INTRUSION (“Vulnerability Issues”). RENESAS ELECTRONICS DISCLAIMS ANY AND ALL RESPONSIBILITY OR LIABILITY ARISING FROM OR RELATED TO ANY VULNERABILITY ISSUES. FURTHERMORE, TO THE EXTENT PERMITTED BY APPLICABLE LAW, RENESAS ELECTRONICS DISCLAIMS ANY AND ALL WARRANTIES, EXPRESS OR IMPLIED, WITH RESPECT TO THIS DOCUMENT AND ANY RELATED OR ACCOMPANYING SOFTWARE OR HARDWARE, INCLUDING BUT NOT LIMITED TO THE IMPLIED WARRANTIES OF MERCHANTABILITY, OR FITNESS FOR A PARTICULAR PURPOSE.
8. When using Renesas Electronics products, refer to the latest product information (data sheets, user’s manuals, application notes, “General Notes for Handling and Using Semiconductor Devices” in the reliability handbook, etc.), and ensure that usage conditions are within the ranges specified by Renesas Electronics with respect to maximum ratings, operating power supply voltage range, heat dissipation characteristics, installation, etc. Renesas Electronics disclaims any and all liability for any malfunctions, failure or accident arising out of the use of Renesas Electronics products outside of such specified ranges.
9. Although Renesas Electronics endeavors to improve the quality and reliability of Renesas Electronics products, semiconductor products have specific characteristics, such as the occurrence of failure at a certain rate and malfunctions under certain use conditions. Unless designated as a high reliability product or a product for harsh environments in a Renesas Electronics data sheet or other Renesas Electronics document, Renesas Electronics products are not subject to radiation resistance design. You are responsible for implementing safety measures to guard against the possibility of bodily injury, injury or damage caused by fire, and/or danger to the public in the event of a failure or malfunction of Renesas Electronics products, such as safety design for hardware and software, including but not limited to redundancy, fire control and malfunction prevention, appropriate treatment for aging degradation or any other appropriate measures. Because the evaluation of microcomputer software alone is very difficult and impractical, you are responsible for evaluating the safety of the final products or systems manufactured by you.
10. Please contact a Renesas Electronics sales office for details as to environmental matters such as the environmental compatibility of each Renesas Electronics product. You are responsible for carefully and sufficiently investigating applicable laws and regulations that regulate the inclusion or use of controlled substances, including without limitation, the EU RoHS Directive, and using Renesas Electronics products in compliance with all these applicable laws and regulations. Renesas Electronics disclaims any and all liability for damages or losses occurring as a result of your noncompliance with applicable laws and regulations.
11. Renesas Electronics products and technologies shall not be used for or incorporated into any products or systems whose manufacture, use, or sale is prohibited under any applicable domestic or foreign laws or regulations. You shall comply with any applicable export control laws and regulations promulgated and administered by the governments of any countries asserting jurisdiction over the parties or transactions.
12. It is the responsibility of the buyer or distributor of Renesas Electronics products, or any other party who distributes, disposes of, or otherwise sells or transfers the product to a third party, to notify such third party in advance of the contents and conditions set forth in this document.
13. This document shall not be reprinted, reproduced or duplicated in any form, in whole or in part, without prior written consent of Renesas Electronics.
14. Please contact a Renesas Electronics sales office if you have any questions regarding the information contained in this document or Renesas Electronics products.

(Note1) “Renesas Electronics” as used in this document means Renesas Electronics Corporation and also includes its directly or indirectly controlled subsidiaries.

(Note2) “Renesas Electronics product(s)” means any product developed or manufactured by or for Renesas Electronics.

(Rev.5.0-1 October 2020)

Corporate Headquarters

TOYOSU FORESIA, 3-2-24 Toyosu,
Koto-ku, Tokyo 135-0061, Japan
www.renesas.com

Trademarks

Renesas and the Renesas logo are trademarks of Renesas Electronics Corporation. All trademarks and registered trademarks are the property of their respective owners.

Contact information

For further information on a product, technology, the most up-to-date version of a document, or your nearest sales office, please visit:
www.renesas.com/contact/.