瑞萨电容触摸技术

低功耗应用培训

实验环节

描述

本文档从基本的触摸应用工程创建、使用**QE for Cap touch**监控触摸数据和调试运行参数开始，循序渐进的增加并调试以下功能.

- ✓ **12**个按键矩阵功能(自容式)
- ✓ **RX140 MEC**多电极连接功能
- ✓ 低功耗功能(**RX140 Auto Judgement**功能)
- ✓ 接近传感功能(改变**MEC**灵敏度)
- ✓ 低功耗数据的测试

实验目标 • 基本的 **RX140** 触摸应用工程创建 • 使用 **QE for Cap touch** 监控触摸数据 • 使用 **QE for Cap touch** 调试运行参数 • **RX140 MEC** 多电极连接功能 • 低功耗功能(**RX140 Auto Judgement** 功能) • 接近传感功能的实现方式 • 低功耗数据的测试方法	**实验材料** • **RX140低功耗触摸评估板套件** ○ 评估板 **x 1** ○ **Ez-cube2**仿真器 **x 1** ○ 亚克力板：**1mm**厚度 **x1 , 2mm**厚度 **x1** ○ 塑料柱、塑料螺丝若干 ○ 电池盒 **x 1** ○ **1.5V AAA**电池 **x 2** • **Renesas e² studio & QE for Cap Touch** ○ **e² Studio: v 2023-04** ○ **QE for Cap Touch: v 3.2.0**. • **编译器**: **Renesas CCRX v3.05.00** • **仿真器**：**Ez-CUBE2** 备注:
技能水平 • **C** 语言和嵌入式编程基础 • 瑞萨触摸应用开发环境，包括： ▪ **e2 studio** 集成开发环境 ▪ **Smart Configurator** 驱动代码生成工具 ▪ **QE for Cap Touch** 触摸应用开发工具	**时间** • **90 mins**

实验环节

1 实验前的准备

概述

在本实验环节中，将介绍实验前的准备，包括培训提供的配套资料，软件开发环境的安装、硬件安装以及 **RX140** 低功耗触摸评估板的功能说明。

实验前的准备：

配套资料说明：

Pre-installations 文件夹

|----------**Renesas e2 studio 2023-04** 安装指南**.pdf**

Presentation 文件夹

|---------- 瑞萨电容触摸技术-低功耗应用培训 (基于 **RX140 MEC+AJ** 新功能)**.pdf**

Lab 文件夹

|----------**Checkpoints** 文件夹

|----------**Lab session 1** 文件夹

|----------**Lab session 2** 文件夹

|----------**Lab session 3** 文件夹

|----------**Lab session 4** 文件夹

软件安装

Renesas e2 studio 2023-04。

详见《**Renesas e2 studio 2023-04** 安装指南.PDF》

硬件安装

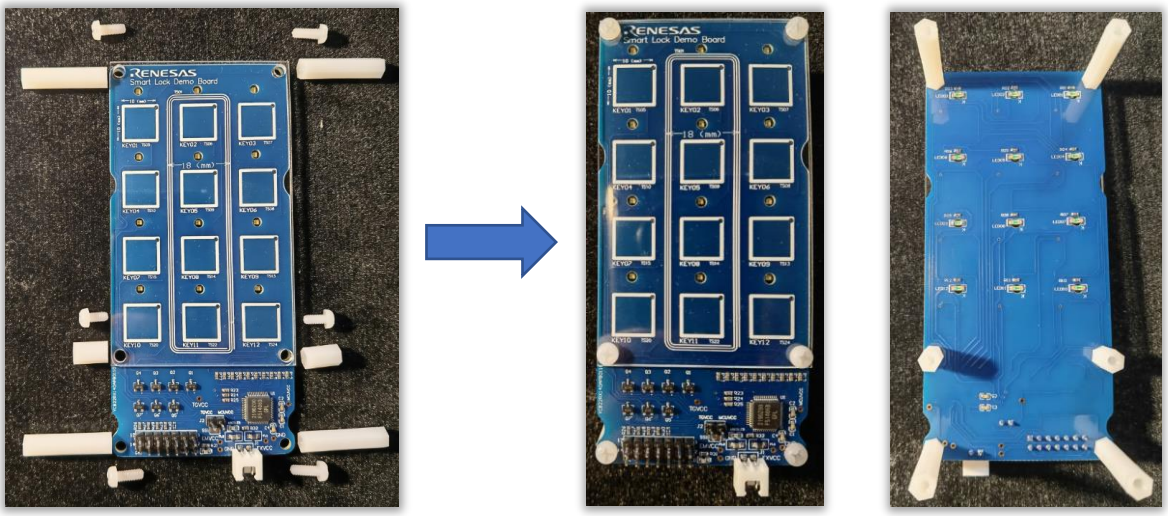
下图为实验环节用的 **RX140** 低功耗触摸评估板套件。

RX140 低功耗触摸评估板的组装:

将 **1mm 厚度**亚克力板，通过塑料柱、塑料螺丝固定在 PCB 板上。

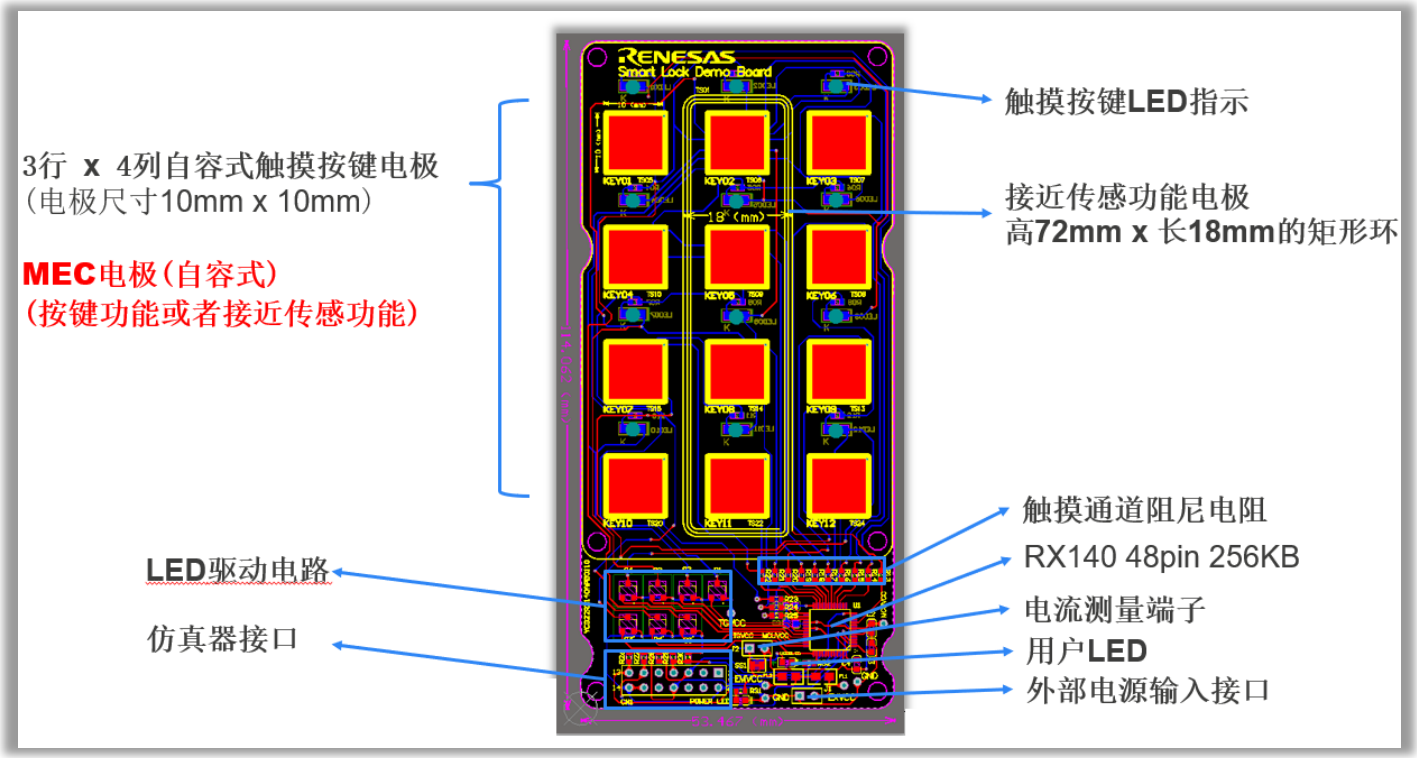
2mm 厚度亚克力板备用，用于体验不同的覆盖物厚度对灵敏度的影响。

电池盒以及电池用于低功耗测试时的电源供电。



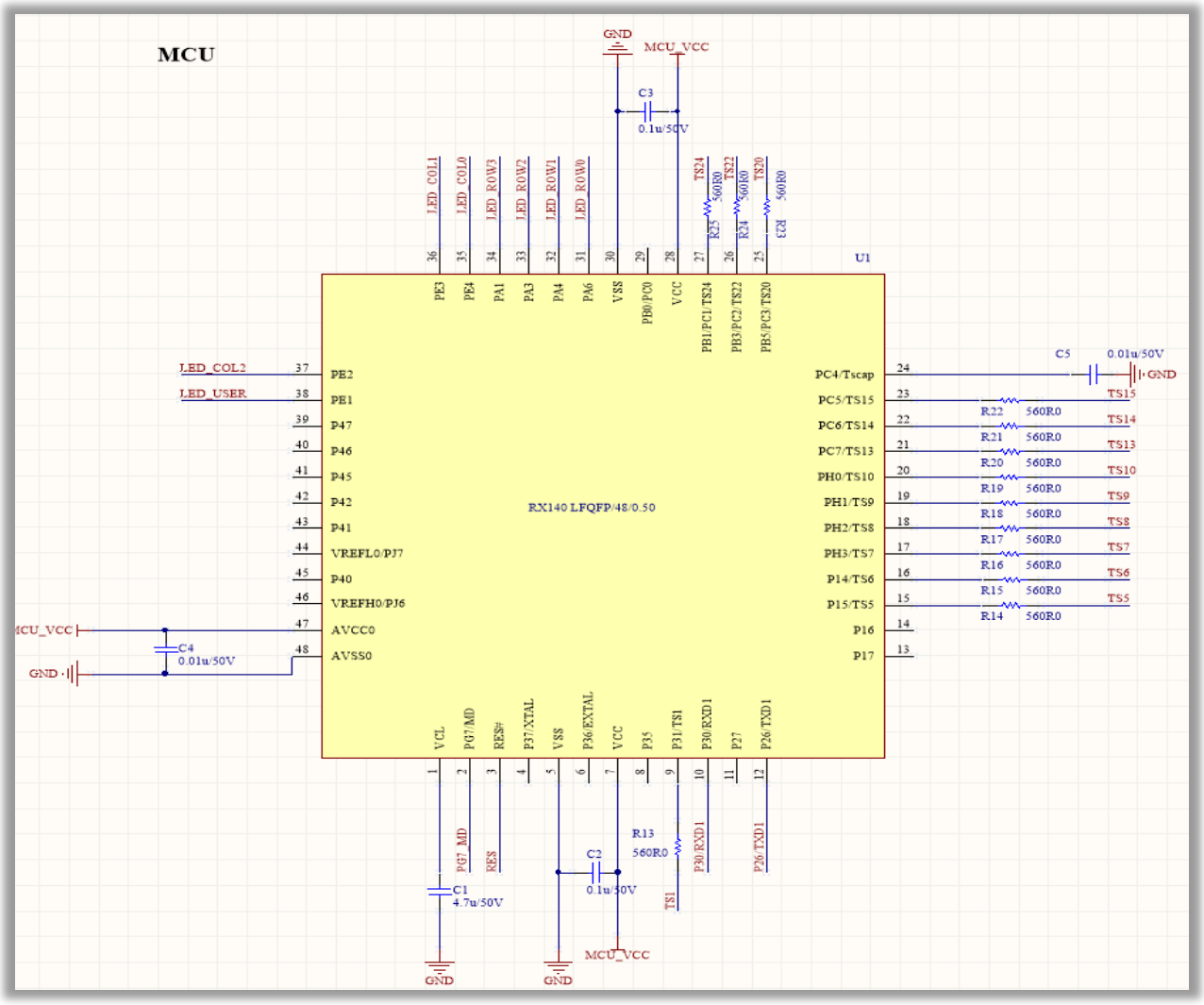
RX140 低功耗触摸评估板的简要说明

功能总览:

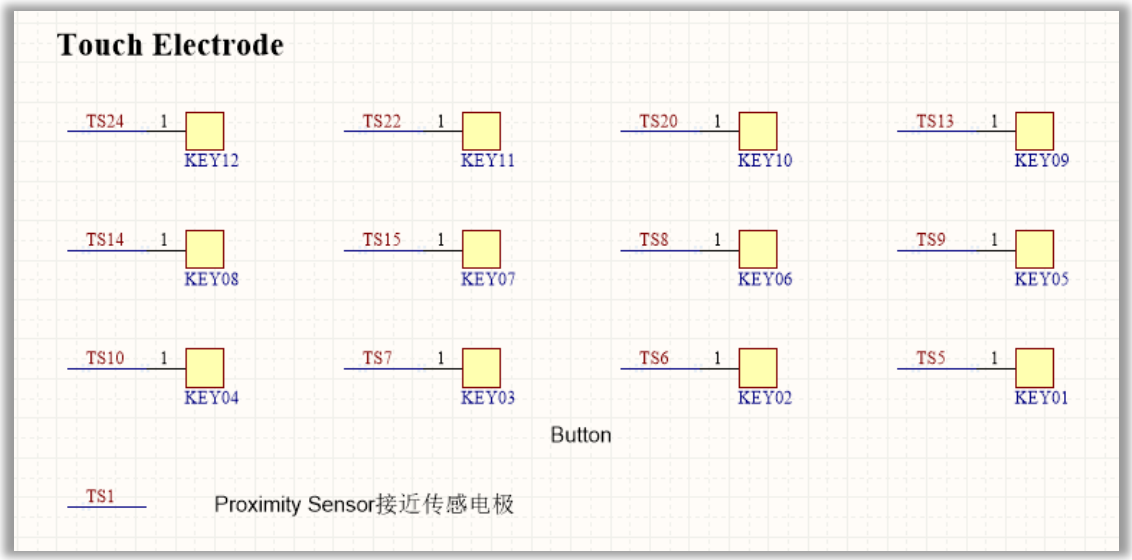


以下为各功能电路的简要说明。

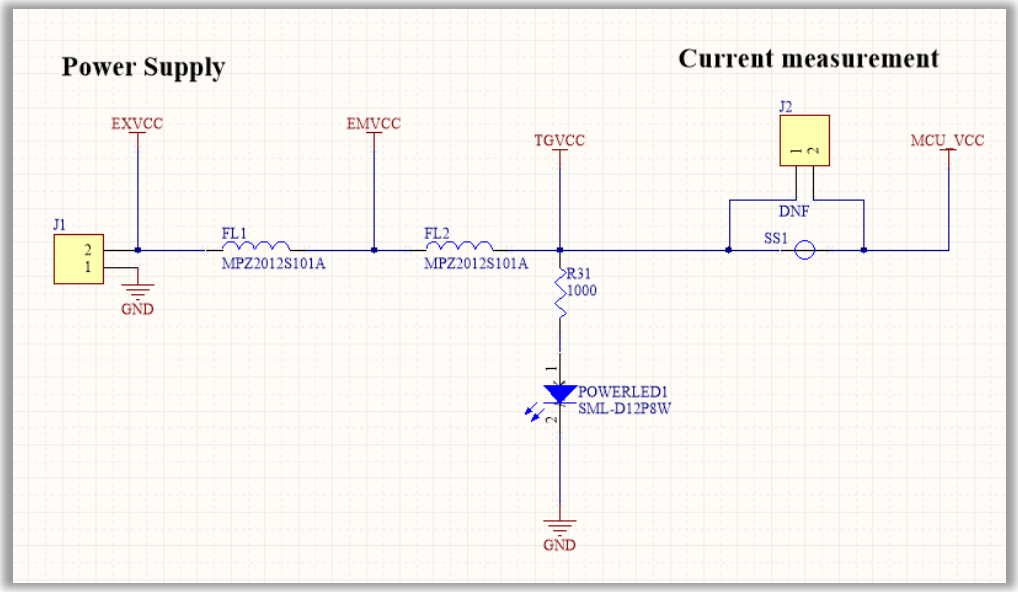
MCU 部分（包含触摸功能必需的 Tscap 滤波电容(C5)、阻尼电阻(从 R14 到 R25)）：



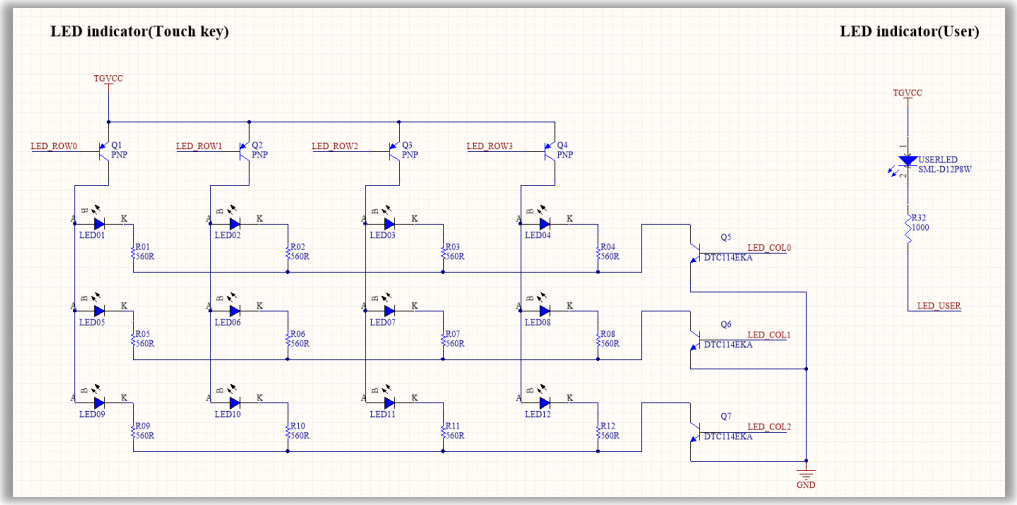
触摸电极电路：



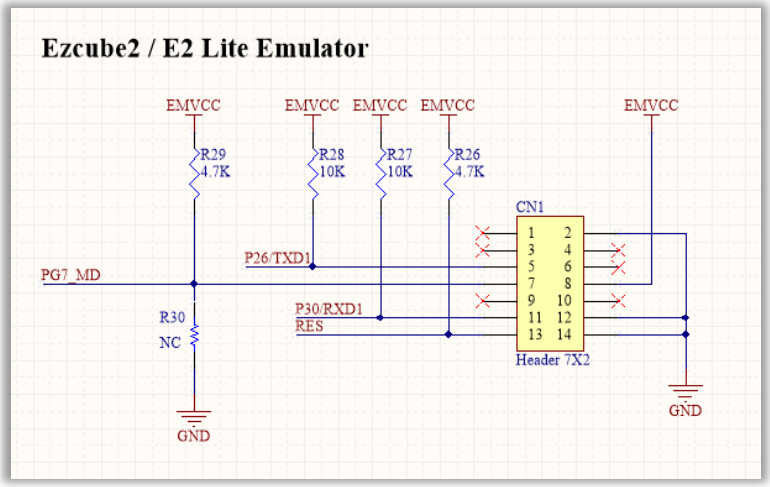
电源电路及低功耗测试端子：



LED 驱动电路：



仿真接口电路：



EZ-CUBE2 仿真器:

使用 **EZ-CUBE2** 前, 将跳线开关设定到“**3.3V** 供电输出”和“调试 **RX**”.
注意线缆在连接 **EZ-CUBE2** 本体时的接口方向，不要插反。

END OF SECTION

2 Lab Session 1： 基于 RX140 创建一个基本的含有 12 个自容按键的触摸应用工程

概述

在本实验环节中, 将基于 **RX140** 创建一个基本的含有 **12** 个自容按键的触摸应用工程，以此了解使用 **QE For Cap Touch** 进行触摸应用开发的软件步骤和流程，主要包括以下方面：

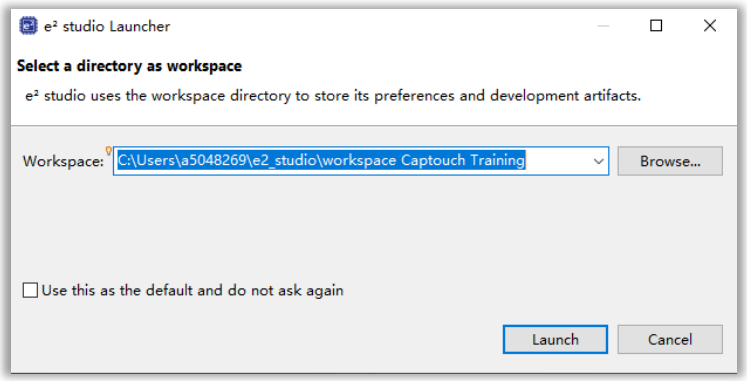
- **2.1** 新建工程
- **2.2** 使用 **Smart configurator** 添加必要的驱动程序
- **2.3** 创建触摸接口(**interface**)或者配置(**Configuration**)
- **2.4** 自动调整过程(**Auto Tuning Process**)
- **2.5** 增加应用程序
- **2.6** 运行程序
- **2.7** 使用指示触摸按键状态的 **LED** 监控触摸行为
- **2.8** 使用全局变量 **button_status** 监控触摸行为
- **2.9** 使用 **QE for Cap Touch** 监控触摸底层数据以及触摸行为
- **2.10** 调试触摸运行参数

如果对 **Lab session 1** 的内容非常熟悉或者有一定困难，可跳过步骤 **2.1** 到步骤 **2.5**，

在 **e2 studio** 中 **import** 导入培训配套资料 **Checkpoints** 文件夹中的工程 **Lab session 1**，

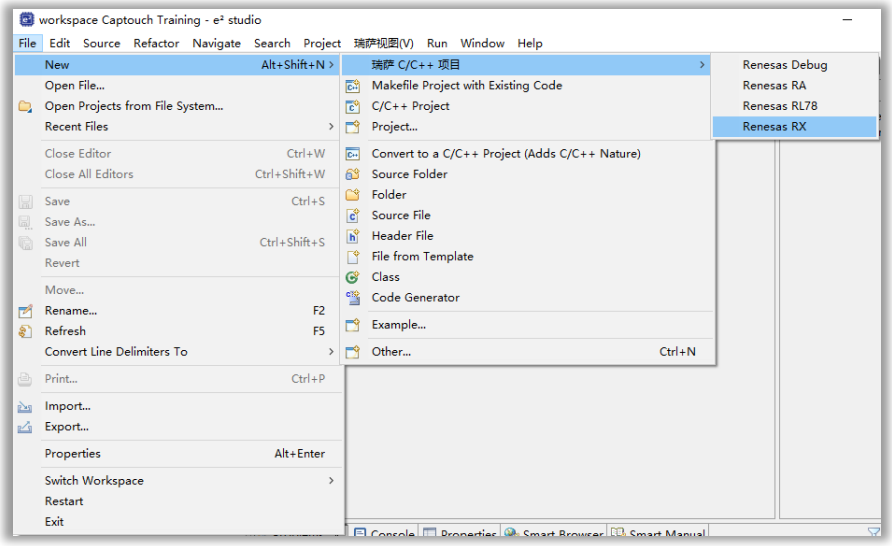
直接进行步骤 **2.6** 到步骤 **2.10** 的实验。

实验步骤

2.1	**新建工程**
2.1.1	**启动"e2 studio 2023 04".** 新建工作空间 **Workspace**： **workspace Captouch Training** 单击**"Launch"** 

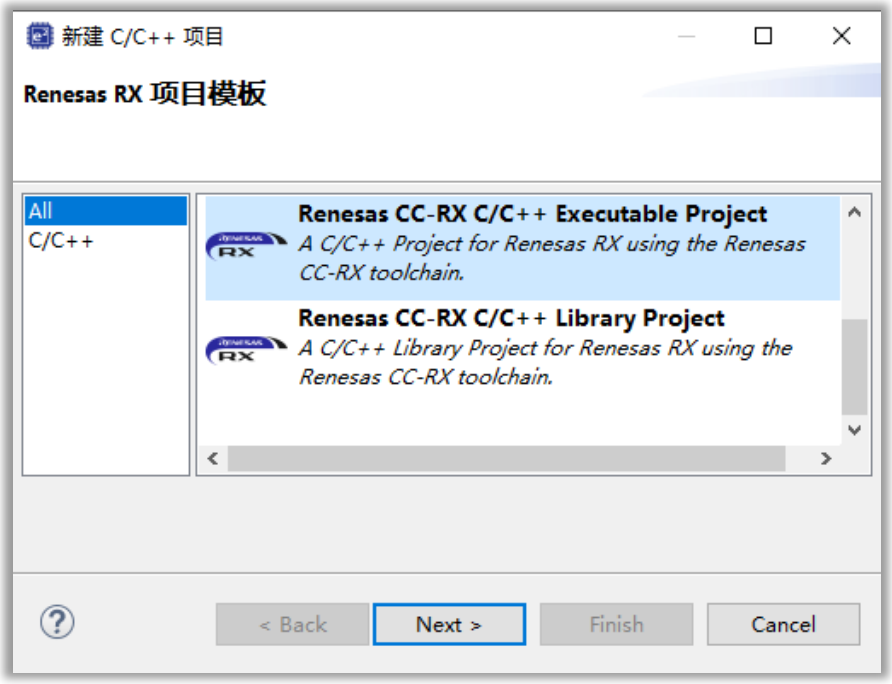
2.1.2 **新建 Renesas RX 工程**

依次选择 **File → New →** 瑞萨 **C/C++项目 → Renesas RX**



2.1.3 **选择"Renesas CC-RX C/C++ Executable Project"**

单击 **Next**

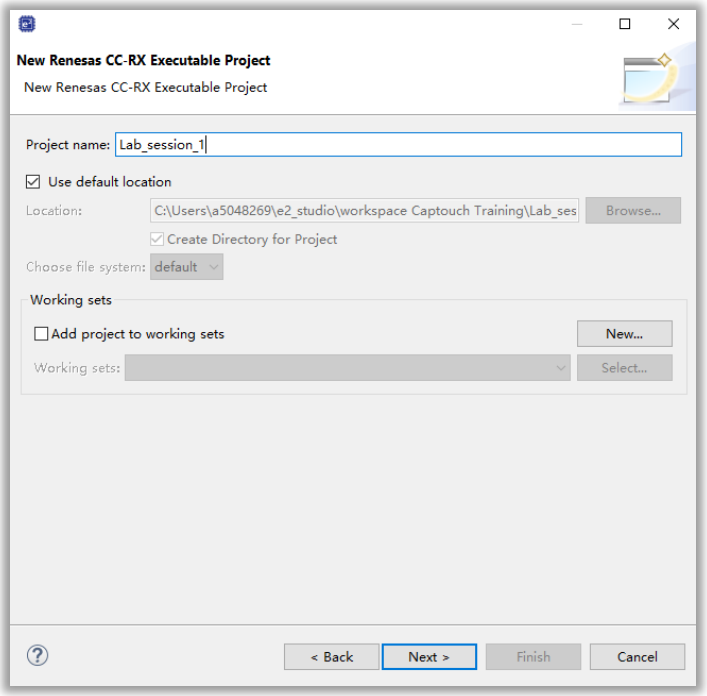


2.1.4

工程名称 Project name:

输入 **Lab_session_1**

单击 **Next**

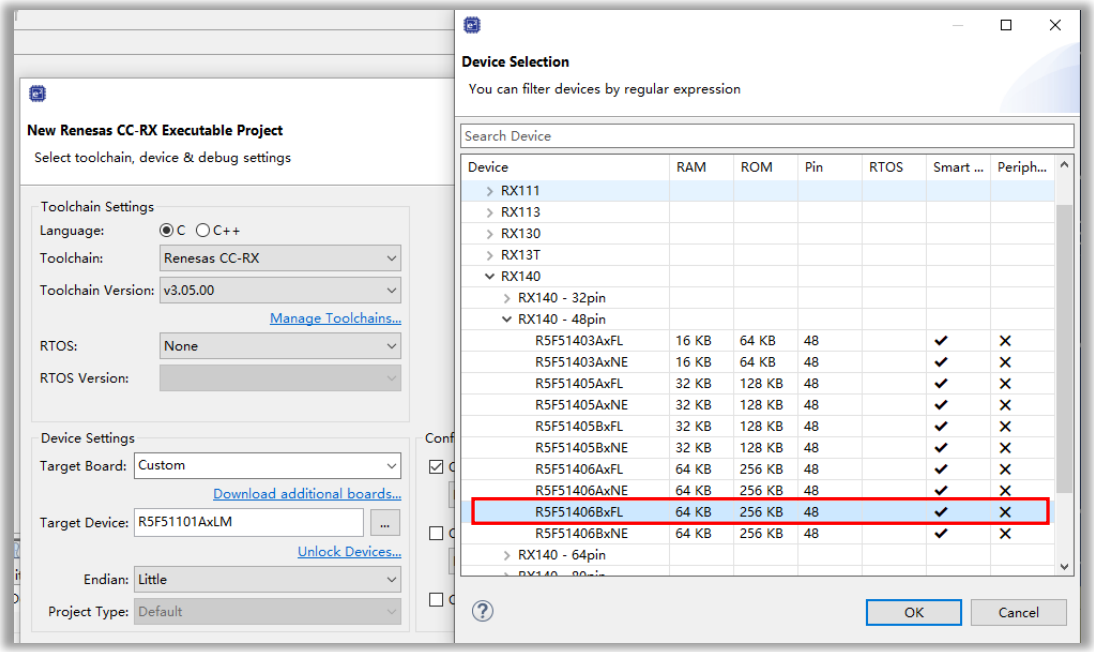


2.1.5

在"**Target Device**"右侧单击 

在弹出的对话框中依次选择 **RX100 → RX140 → RX140-48Pin → R5F51406BxFL**

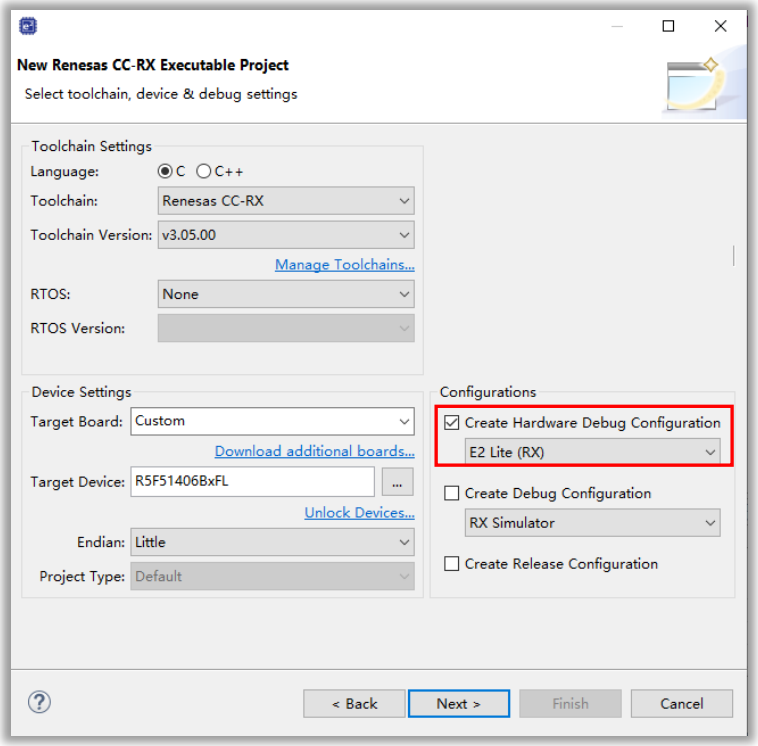
单击 **OK** 关闭对话框



2.1.6

在"**Configurations**" → "**Create Hardware Debug Configuration**"的下拉对话框中选择"**E2 Lite（RX）**"

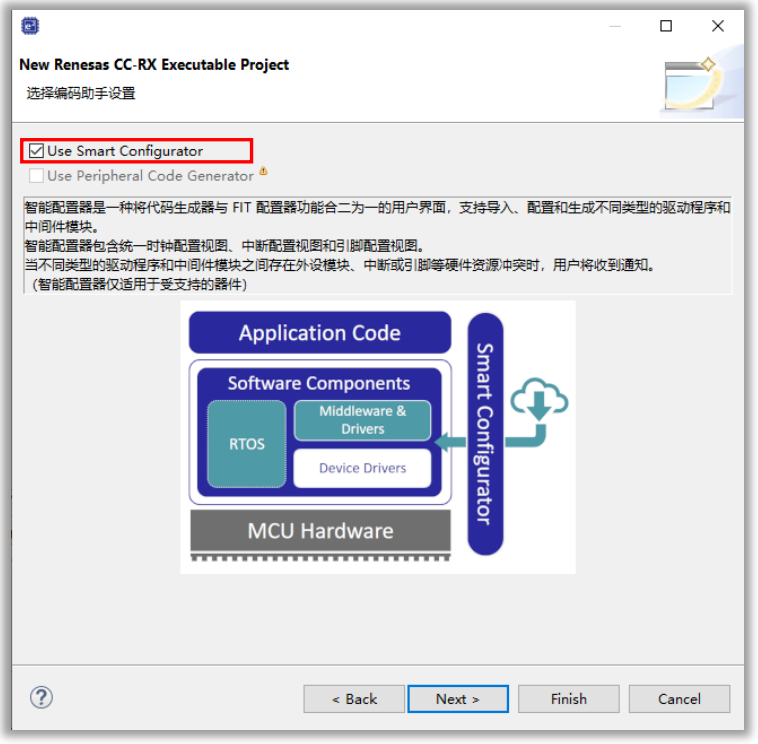
单击 **Next**



2.1.7

默认选择"**Use Smart configurator**"

单击 **Next**

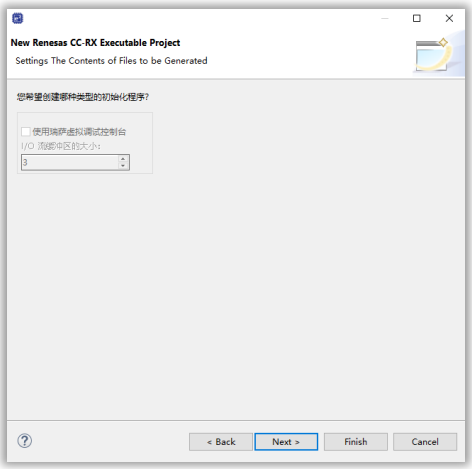


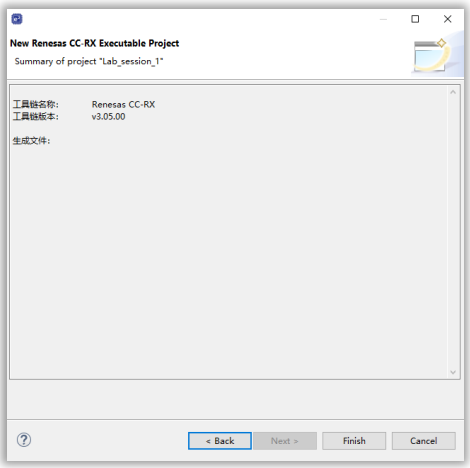
2.1.8

以下两个页面保持默认

第一个页面单击 **Next**

第二个页面 **Finish**



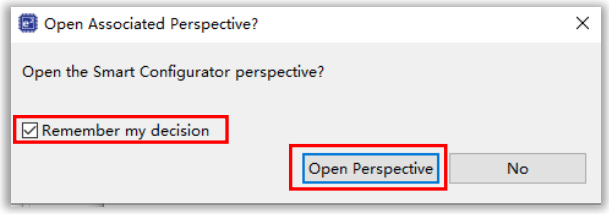


2.1.9

打开"**Smart configutor**"

勾选"**Remember my decision**"

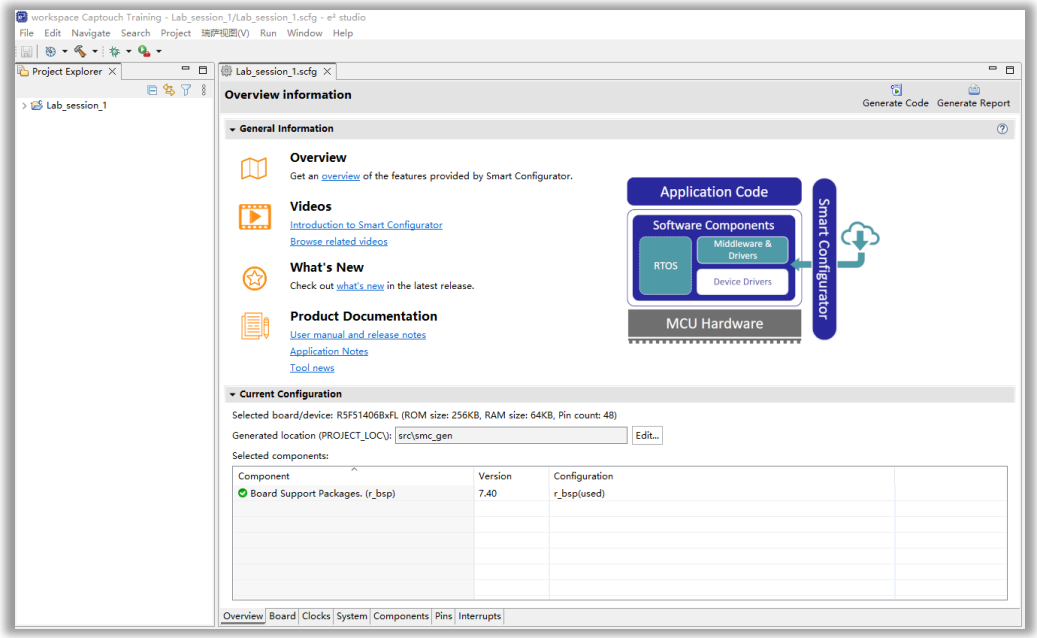
单击"**Open perspective**"



2.1.10

Smart Configurator

新建工程后，默认来到"**Smart Configurator**"的"**Overview**"标签页

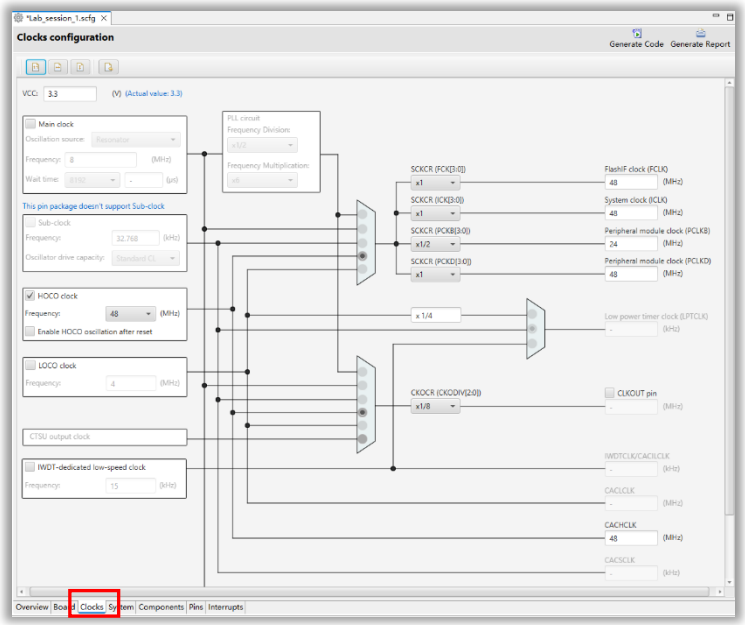


2.2 使用Smart Configurator添加必要的外设驱动程序

2.2.1 Clock设定

切换到"**Smart configurator**"的"**Clocks**"标签页，时钟配置的默认设定如下图

保持默认设定



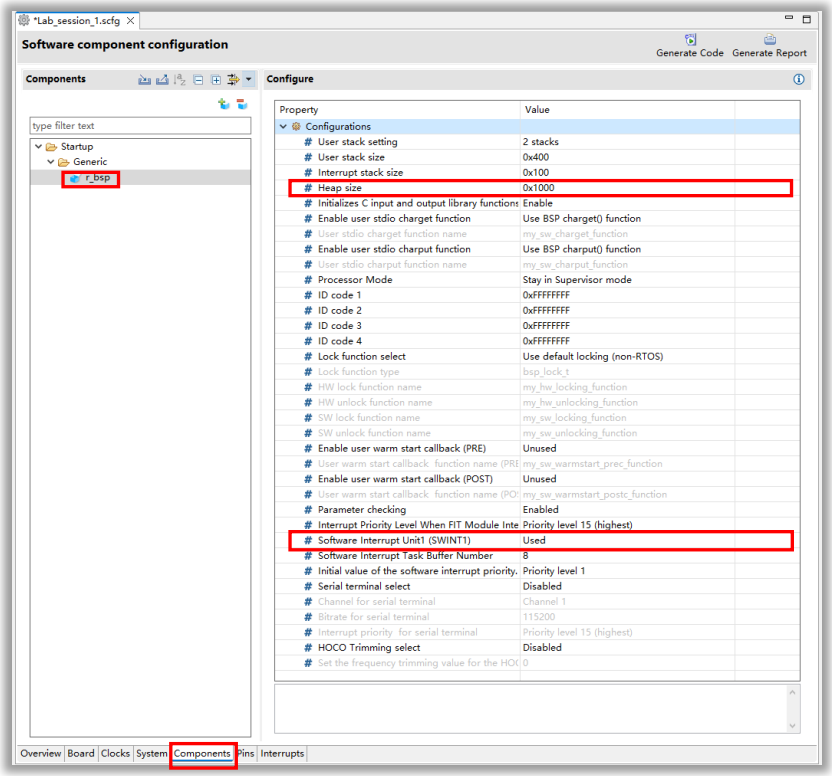
2.2.2 切换到"Smart configurator"的"Components"标签页

r_bsp设定

单击 r_bsp 右侧显示**r_bsp**的详细设定

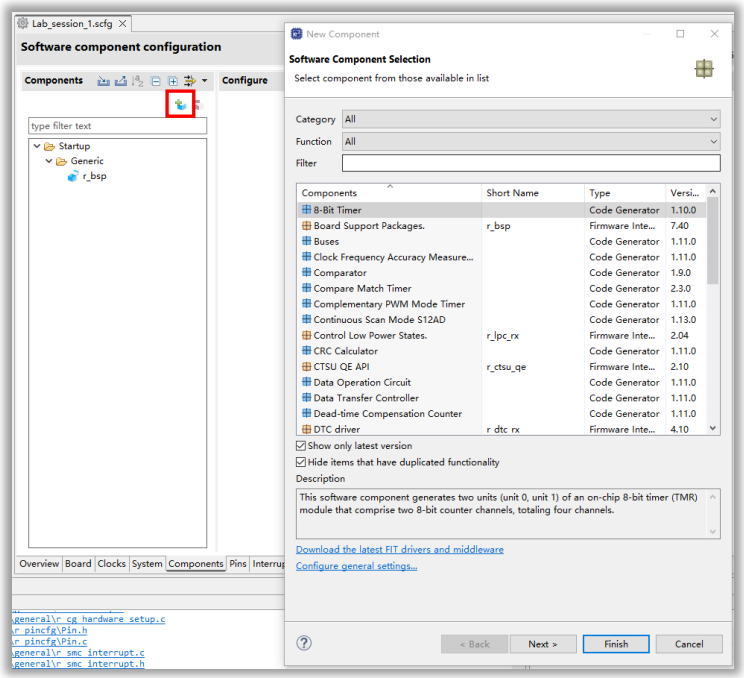
将"**Heap size**"的设定从默认**0x400**改为**0x1000**

将"**Software interrupt Unit1(SWINT1)**"的设定从默认"**Unused**"改为"**Used**"



2.2.3 **在Components标签页，添加驱动程序**

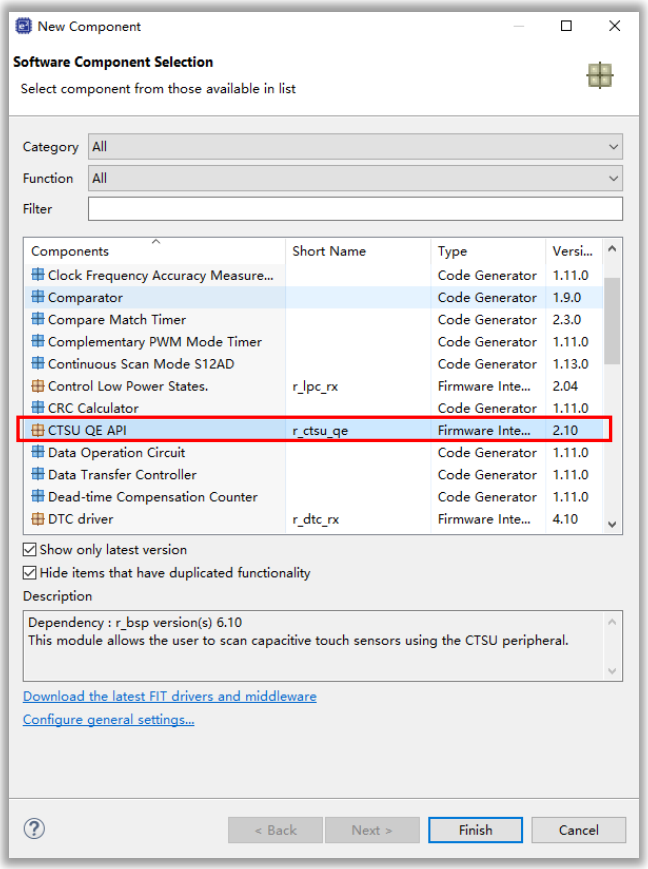
单击  图标，弹出"**Software Components Selection**"对话框，如下图所示

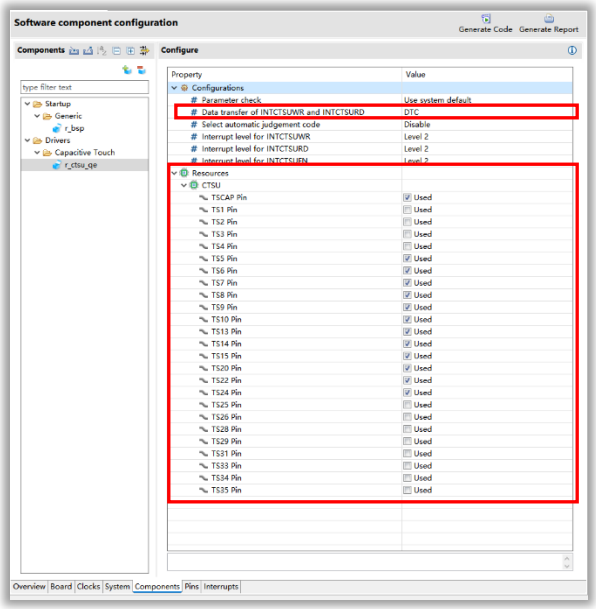
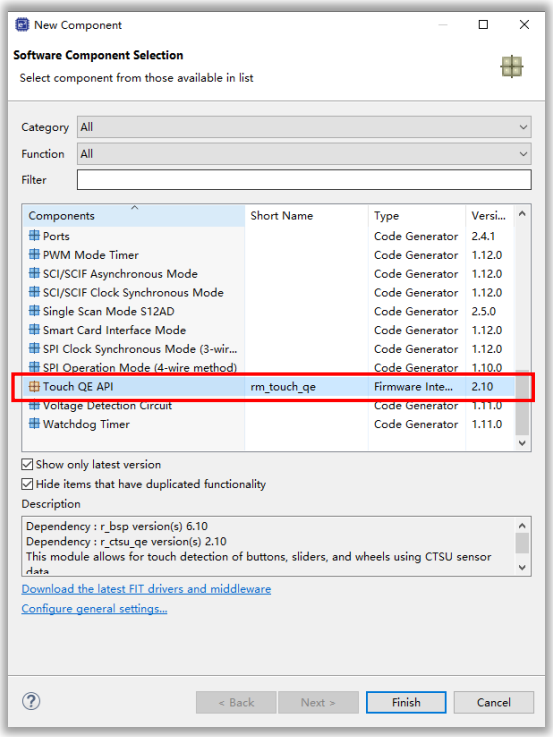


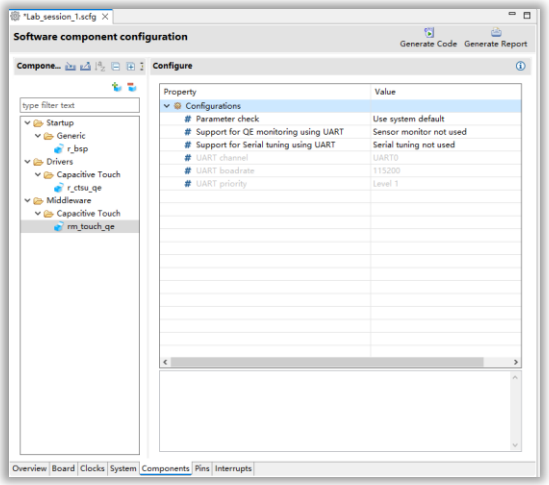
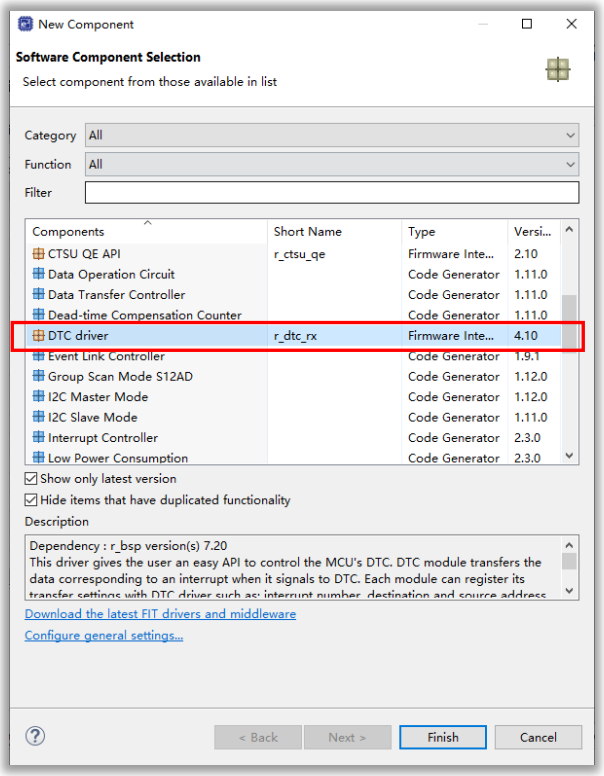
2.2.4 **添加触摸相关的驱动程序**

选择"**CTSU QE API**"

单击 "**Finish**"



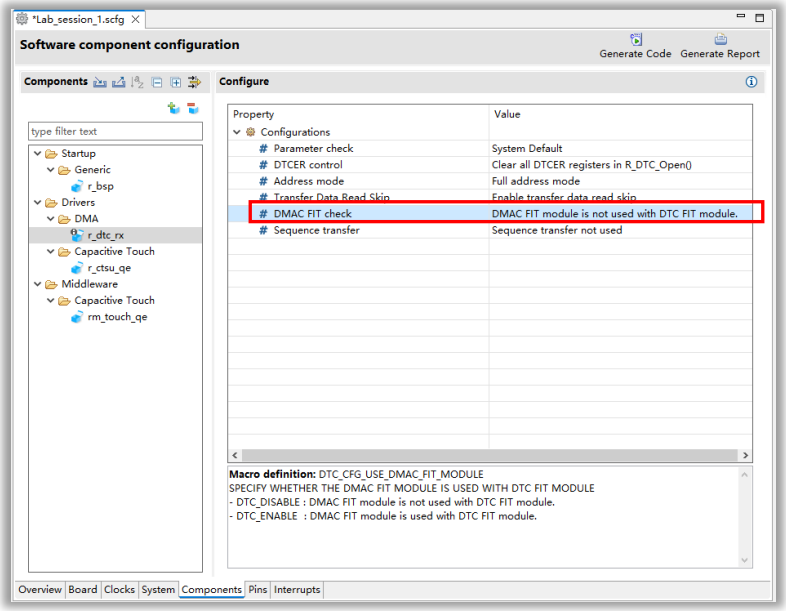
2.2.5	**"r_ctsu_qe"设定** 将"**Data transfer of INTCTSUWR and INTCTSURD**"从默认的"**Interrupt Handler**"改为"**DTC**" 将"**TSCAP**"以及使用的12个触摸通道设定为"**Used**" **TS5、TS6、TS7，TS8，TS9，TS10，TS13，TS14，TS15，TS20，TS22，TS24** 
NOTE	**1. " CTSUWR"和"CTSURD"为CTSU底层中断，用于传输CTSU的运行参数，** **传输的方式可以选择"Interrupt Handler"，也可以选择为"DTC"。** **需要注意的是，CTSU在低功耗模式下工作时，必须选择为"DTC"。** **2. "Tscap"为必选项。**
2.2.6	**添加触摸相关的驱动程序** 选择"**TOUCH QE API**" 单击"**Finish**" 

2.2.7	**"rm_touch_qe"设定** 保持默认不变 
2.2.8	**添加DTC驱动程序** 选择**"DTC Driver"** 单击**"Finish"** 
NOTE	由于**"r_ctsu_qe"**的设定中，使用了**DTC**，因此需要**DTC**驱动程序

2.2.9

"DTC Driver"设定

将"**DMAC FIT check**"的设定从默认"**Used**"改为"**NOT Used**"



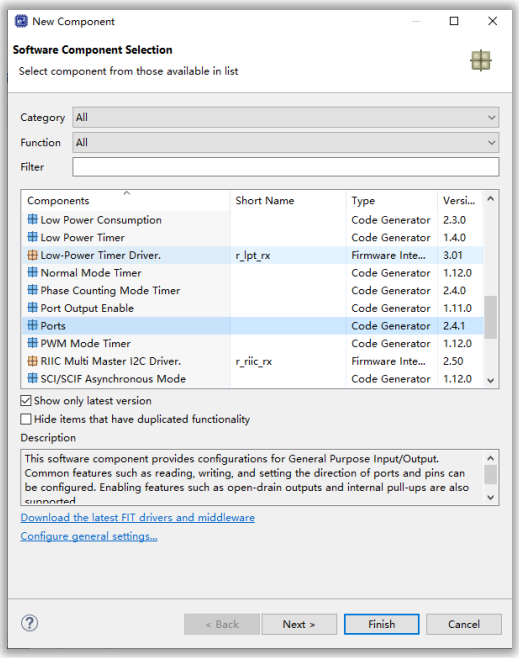
2.2.10

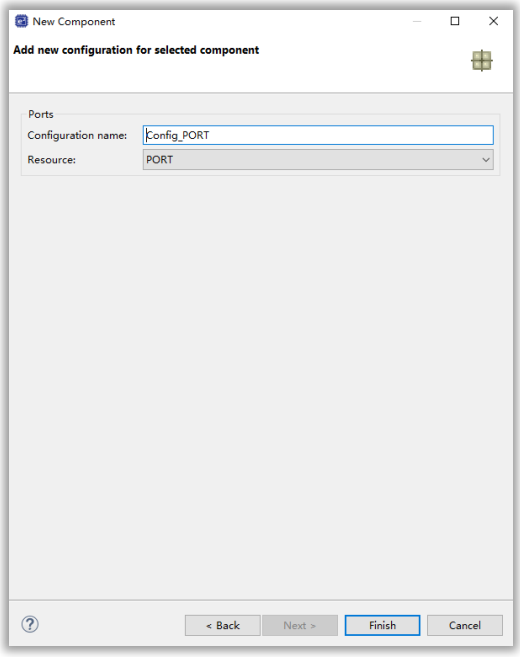
添加"PORT"驱动程序

选择"**Ports**"

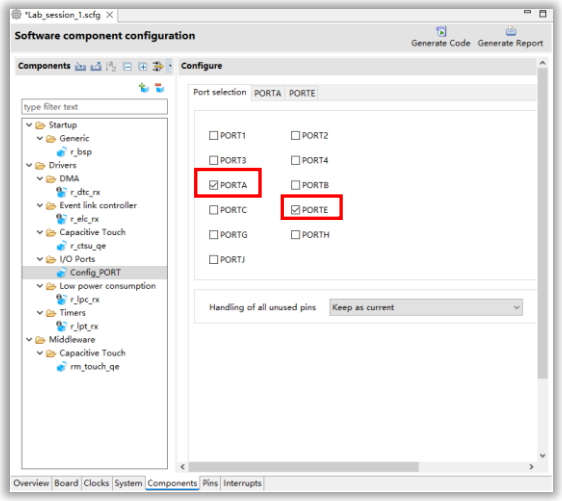
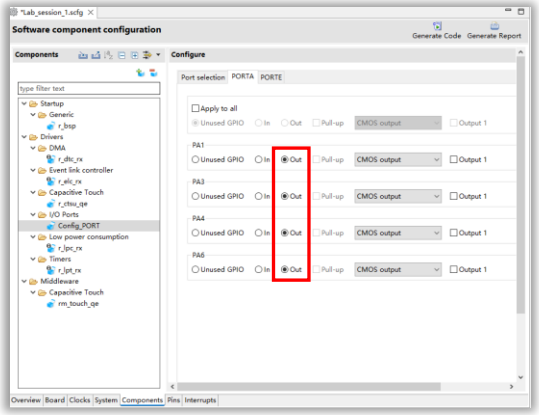
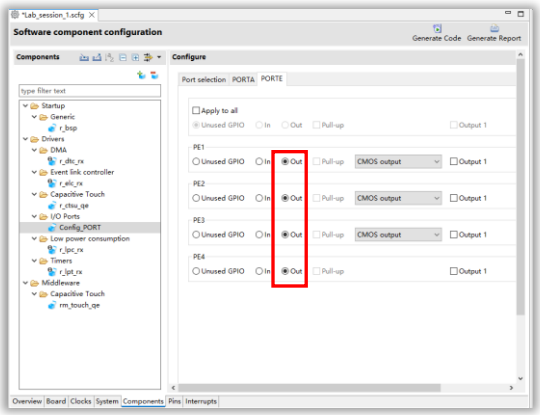
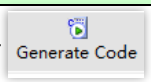
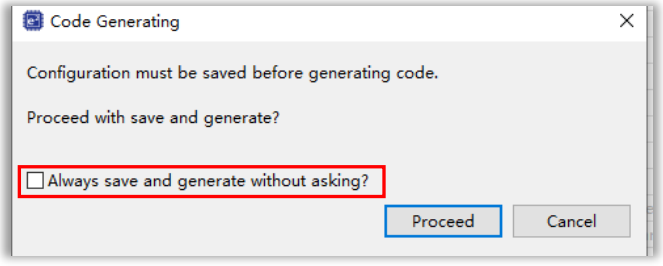
点击"**Next**", 弹出Ports配置对话框，保持默认不变

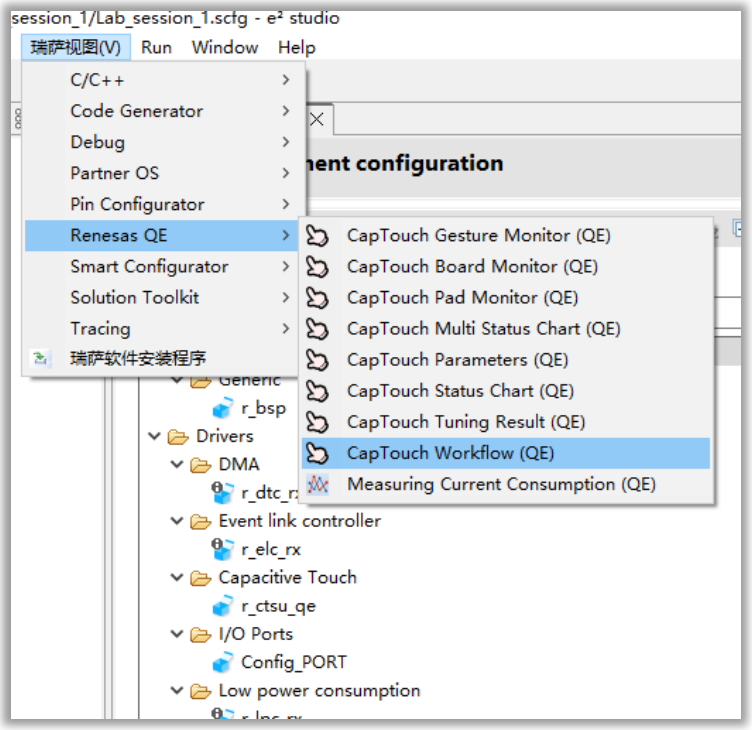
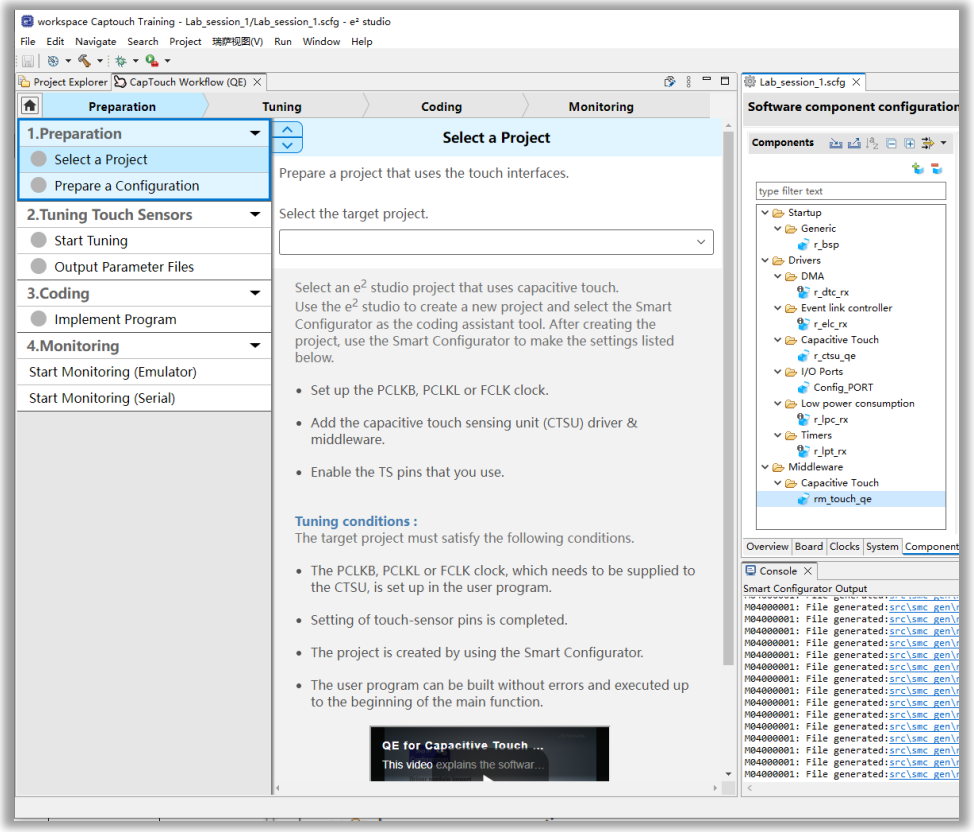
点击"**Finish**"



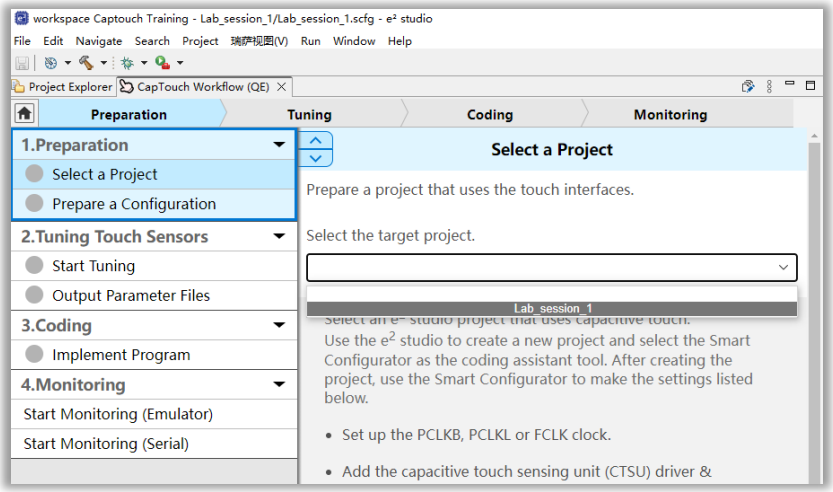


NOTE "**PORT**"用于驱动指示触摸按键状态的**LED**以及用户**LED**

2.2.11	**"PORT"设定** 选择**"PORTA"和"PORTE"** 
2.2.12	**"PORT"设定** 将**"PORTA（PA1，PA3，PA4，PA6）"全部设定为"OUT"** 将**"PORTE（PE1，PE2，PE3，PE4）"全部设定为"OUT"**  
NOTE	**PA1，PA3，PA4，PA6，PE2，PE3，PE4用于驱动指示触摸按键状态的LED** **PE1用于驱动用户LED**
2.2.13	点击  生成驱动程序代码
2.2.14	勾选**"Always save and generate without asking"** 点击**Proceed**继续 

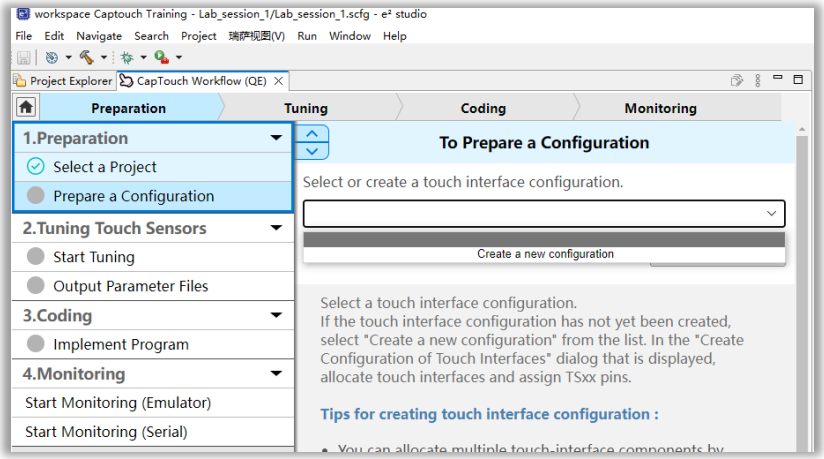
2.3	**创建触摸接口(interface)或者配置(Configuration)**
2.3.1	选择"**Renesas view 瑞萨视图**" → "**Renesas QE**" → "**CapTouch workflow**" 
2.3.2	在"**Cap Touch Workflow**"的"**1.Preparation**"中点击"**Select a project**"， 显示工程选择页面 

2.3.3 在右侧的工程选择页面中，在"**Select the target project**"下拉菜单中选择"**Lab_session_1**"



2.3.4 点击"**Prepare a Configuration**, "

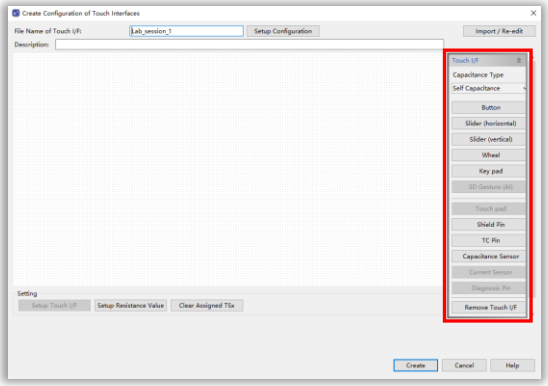
在右侧的触摸接口配置选择页面中选择"**Create a new Configuration**"

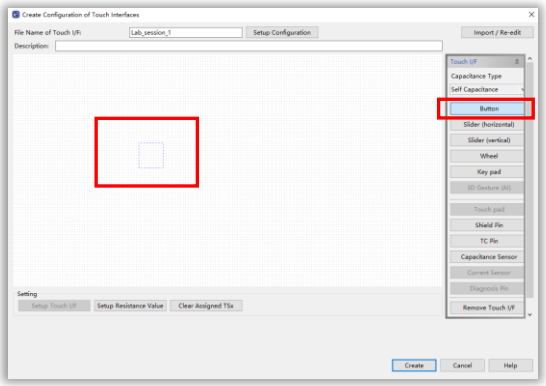


2.3.5 将弹出如下左图所示的画布，

在右侧"**Touch I/F**"列表中，选择"**Button**"并将鼠标移动到画布中间，

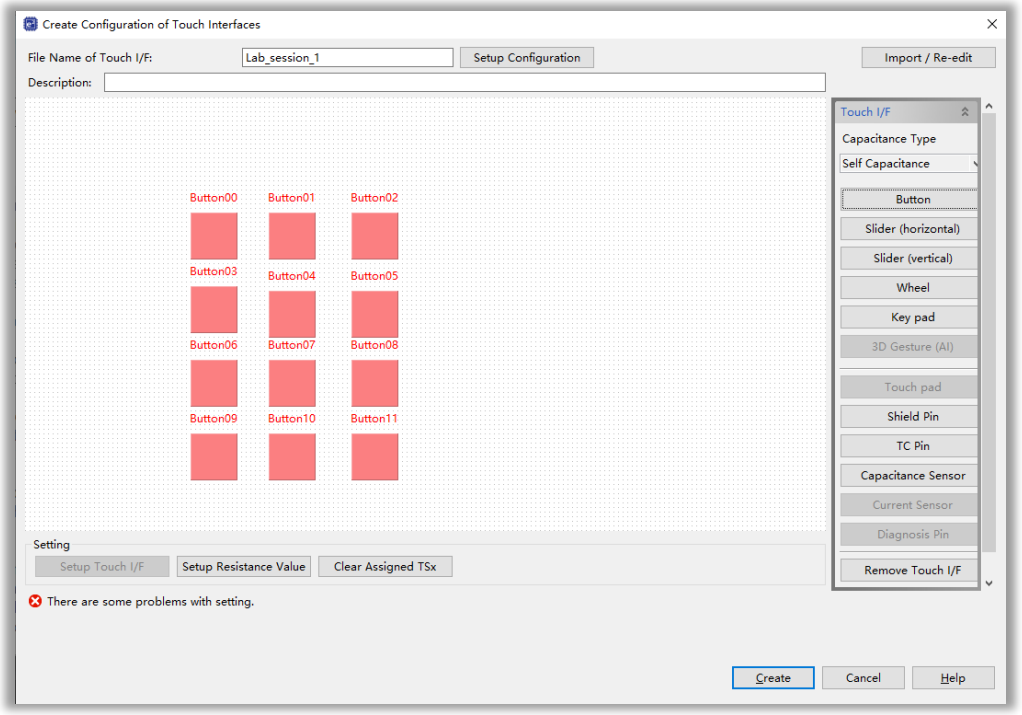
将显示如下右图，鼠标变为正方形虚线框，单击鼠标左键，将创建一个未配置的"**Button**"





2.3.6

依次按下图所示，创建 12 个未配置的"**Button**"，
按下 **PC** 键盘的"**ESC**"键，完成"**Button**"的放置

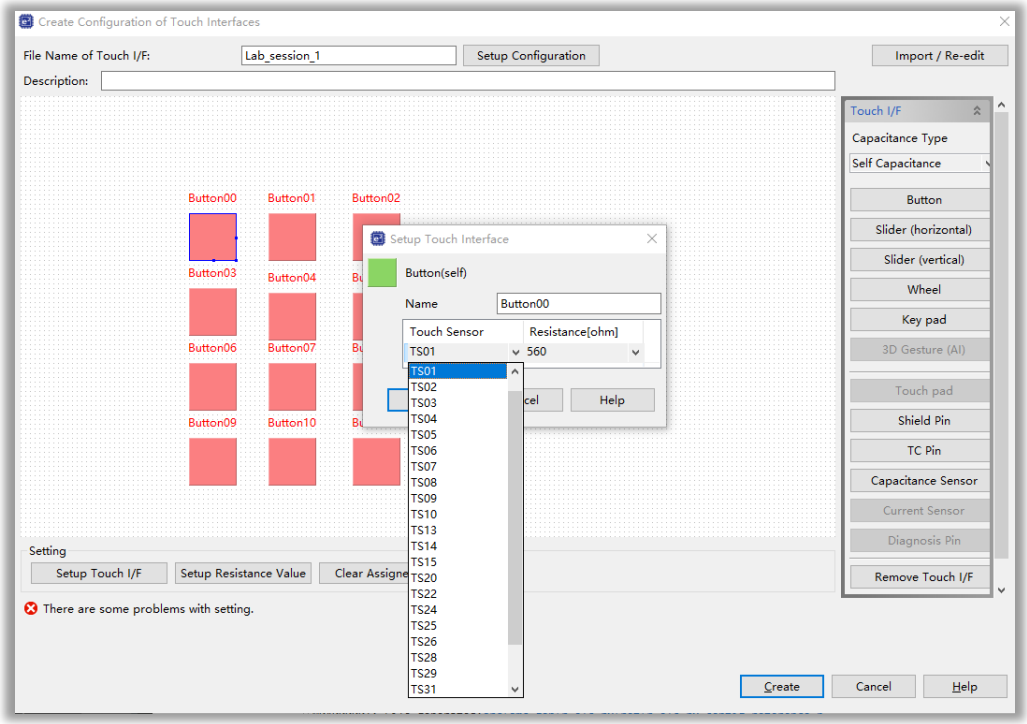


NOTE	如需删除某个"**Button**"，鼠标单击"**Botton**"，选中后，鼠标单击右侧"**Touch I/F**"列表最下方的"**Remove Touch I/F**"

2.3.7

"**Button**"**接口的配置**

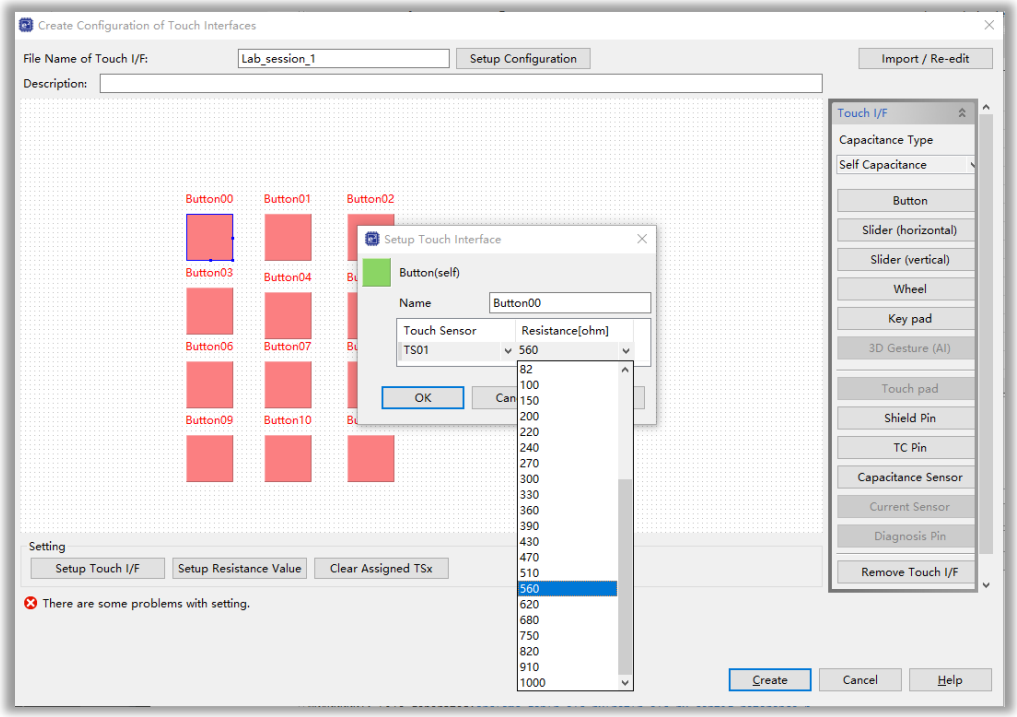
双击某一个"**Button**"，将弹出"**Setup Touch Interface**"对话框
在"**Touch Sensor**"列表中为"**Button**"选择"**TS** 通道"



2.3.8

"Button"接口的配置

在"**Resistance**"列表中为"**Button**"选择阻尼电阻值

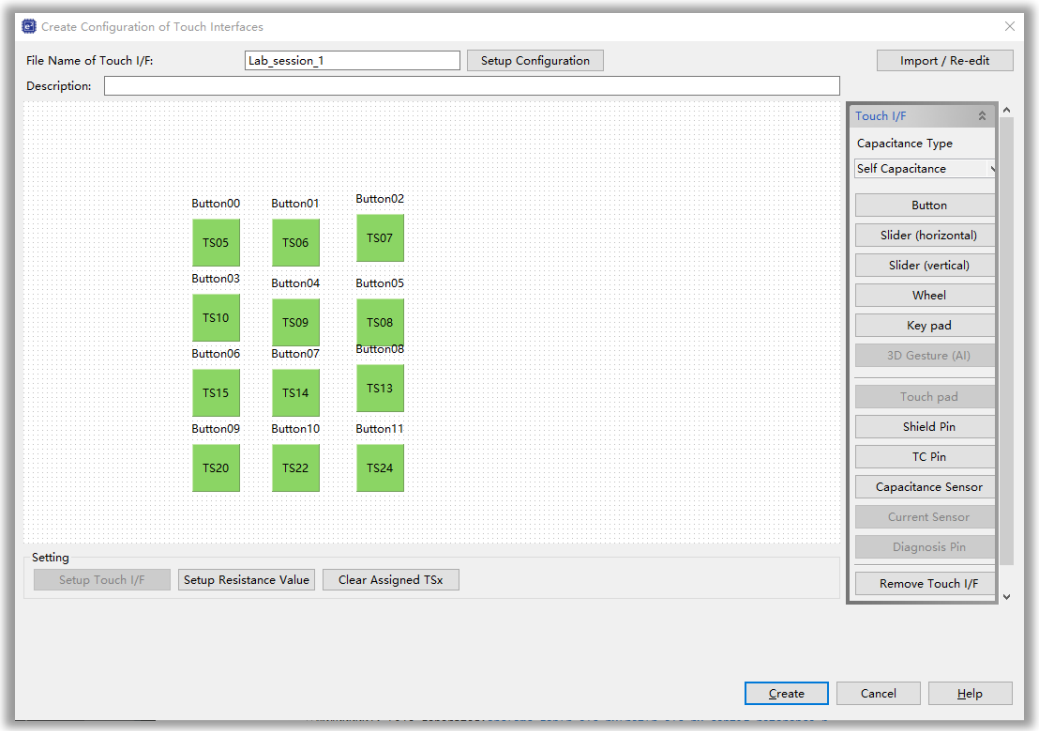


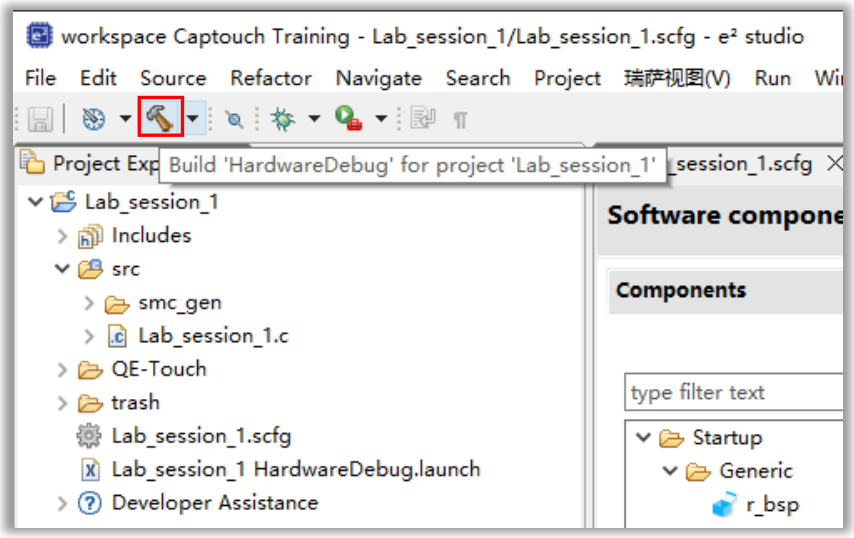
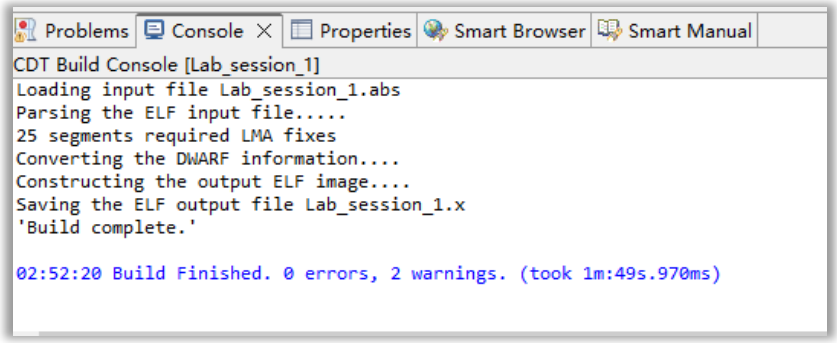
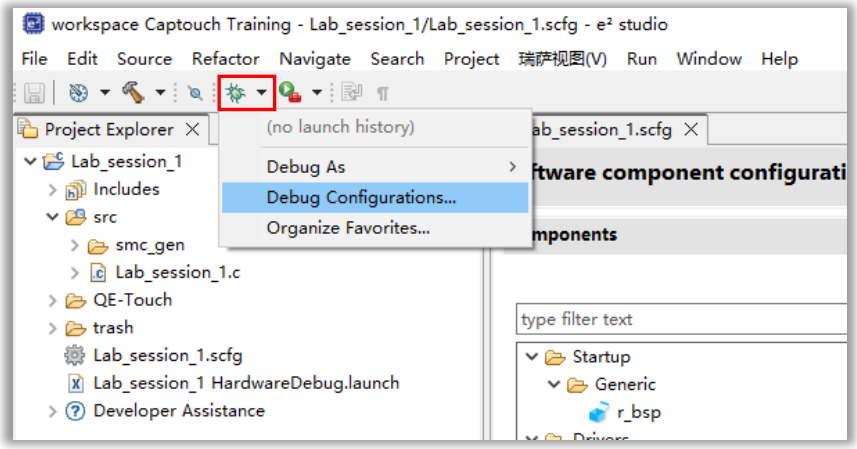
NOTE	阻尼电阻的值，要根据硬件电路中实际使用的阻尼电阻值正确设定。 设定的范围为 **10ohm** 到 **1000ohm**，默认值为 **560ohm**

2.3.9

正确设定完成后，"**Button**"将由红色变为绿色

点击"**Create**"完成设定



2.4	自动调整过程(Auto tuning process)
2.4.1	**编译程序** 点击  图标，编译程序 
2.4.2	**编译完成** 没有错误，如下图显示 
2.4.3	**设定"Debug Configuration"** 点击  图标右侧的向下箭头，选择"**Debug Configuration**" 

2.4.4

在弹出的对话框的左侧列表中，点击"**Renesas GDB Hardware Debugging**"左侧的向下箭头

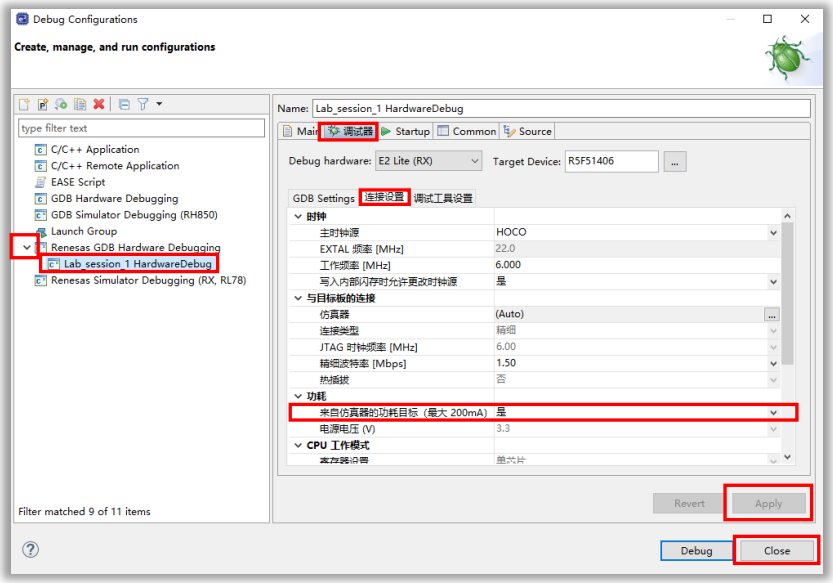
选择"**Lab_sesstion_1_HardwareDebug**"，右侧将显示详细设定

在右侧的详细设定中，选择"调试器"，选择"连接设置"选项卡

确认"来自仿真器的功耗目标(最大 **200mA**) "的设定为"是"

单击"**Apply**"

单击"**Close**"，完成设定

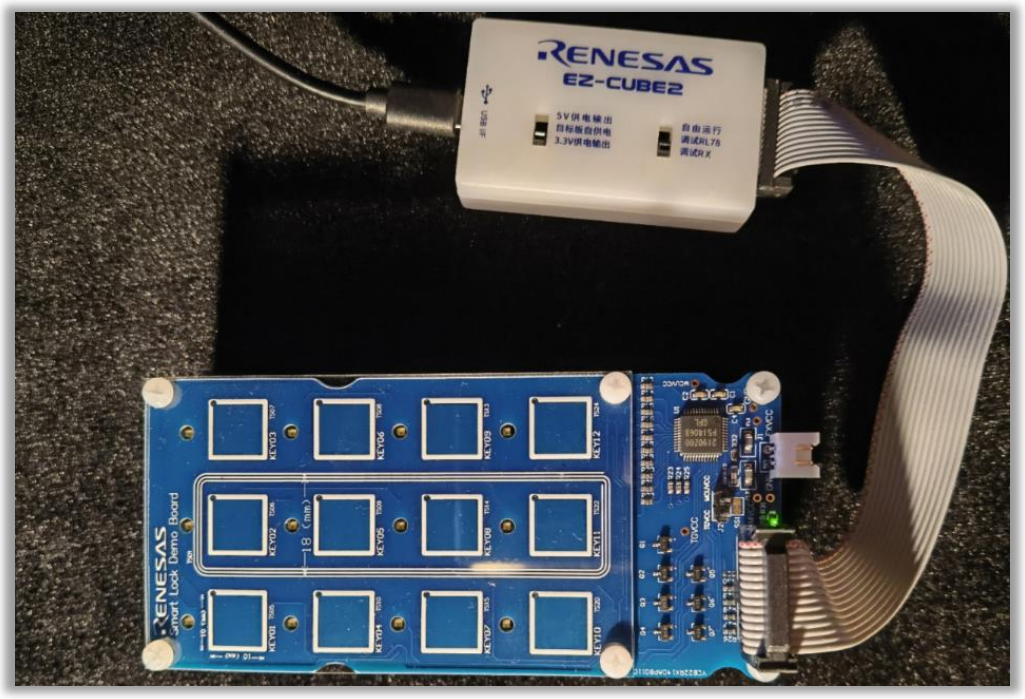


NOTE	此处设定的目的是选择仿真器供电。 如果是目标板供电，此时设定为"否"

2.4.5

硬件连接

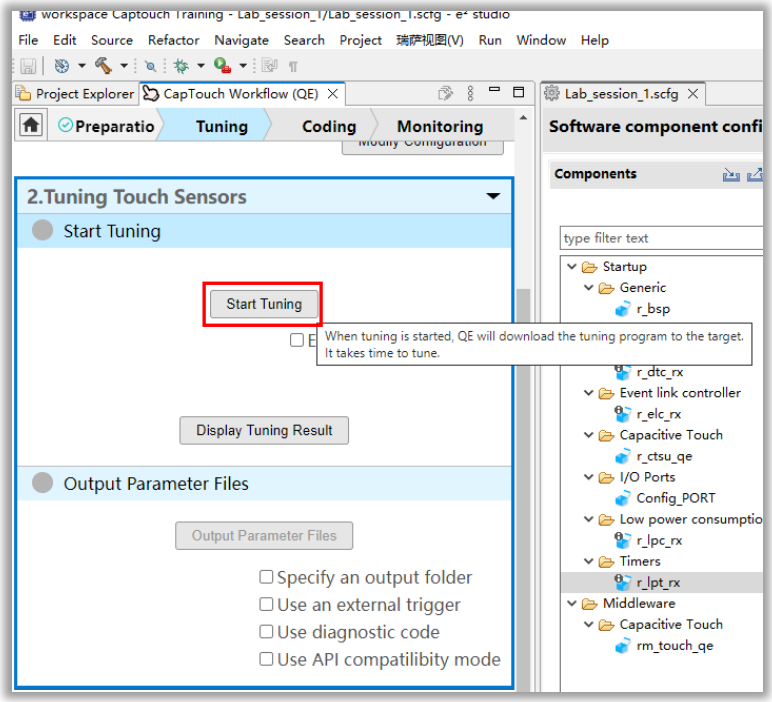
使用"**Ez-cube2**"按下图方式连接评价板和电脑



2.4.6

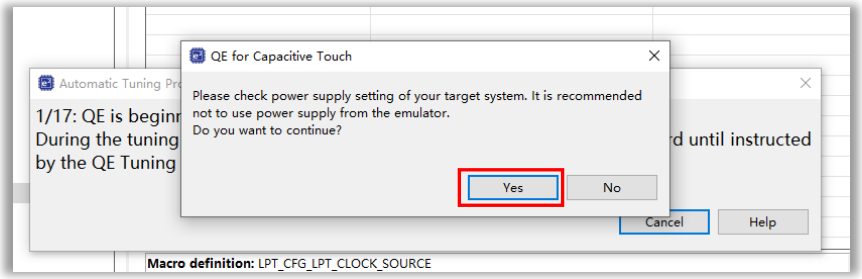
开始自动调整过程(**Auto Tuning Process**)

在"**Cap Touch Workflow**"的"**2.Tuning Touch Sensors**"中，单击"**Start Tuning**"



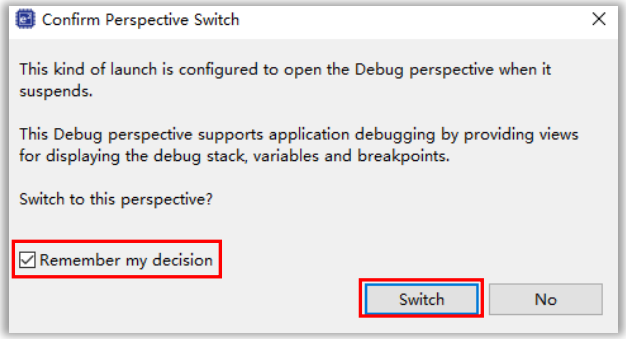
2.4.7

在弹出的对话框中，选择"**Yes**"



在弹出的对话框中，选择"**remember my decision**"

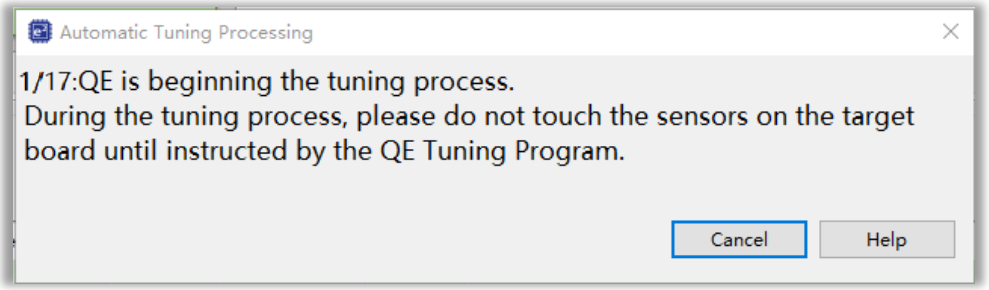
单击"**Switch**"



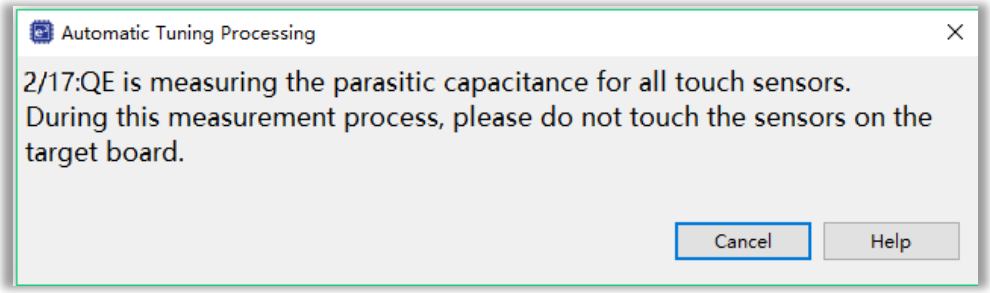
2.4.8

自动调整过程(**Auto Tuning Process**)开始，依次显示如下四步，这时不需要用户操作。

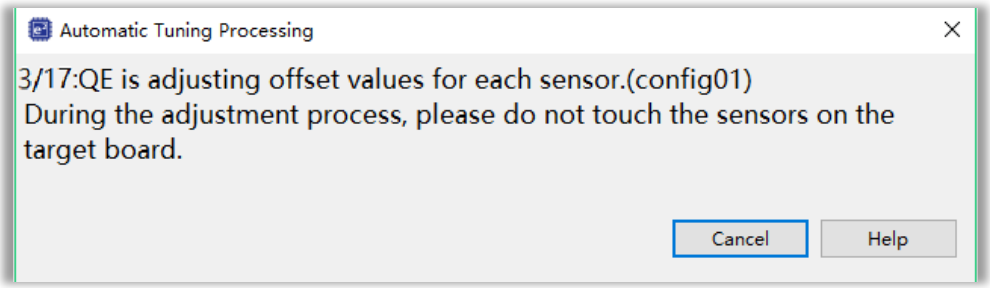
第一步：开始自动调整过程，引导用户按提示操作，按照要求"触摸按键"或者"不要触摸按键"。



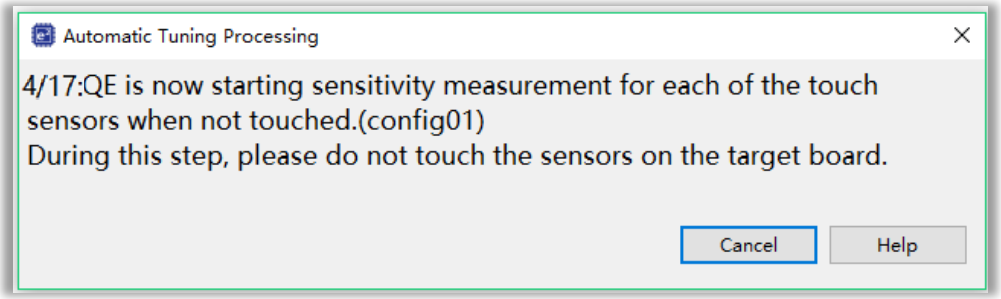
第二步：QE 正在测量所有触摸按键的寄生电容。



第三步：QE 正在调整触摸按键的偏置电流值。



第四步：QE 开始进行灵敏度测量。



NOTE	对话框左上角，显示了当前的步骤，以及总计步骤，例如 1/17: 显示当前步骤为 1，总计步骤为 17

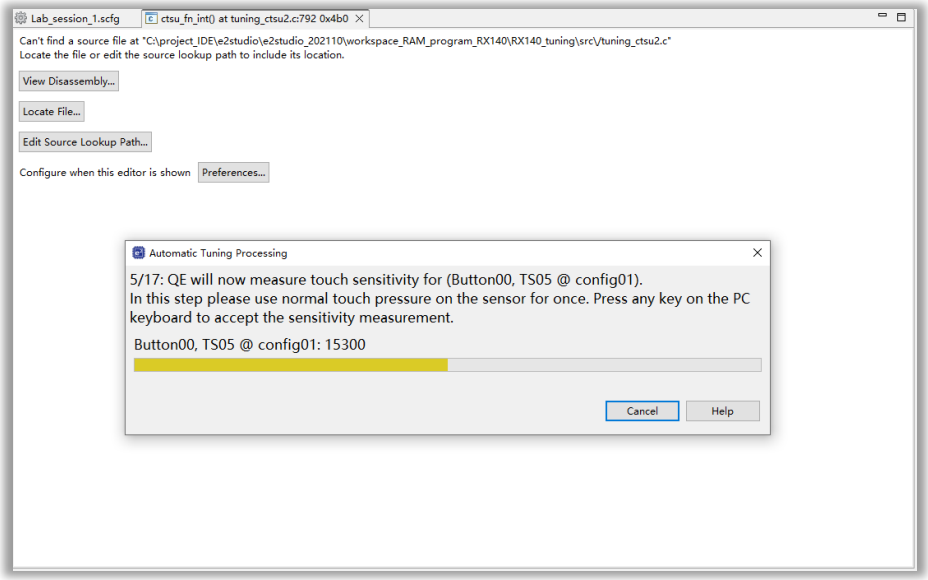
2.4.9

第五步：灵敏度调整

自动调整过程(**Auto Tuning Process**)完成前四步准备工作后，开始第五步。

如下图所示，从触摸按键"**Button00/TS05**"开始依次进行灵敏度测量

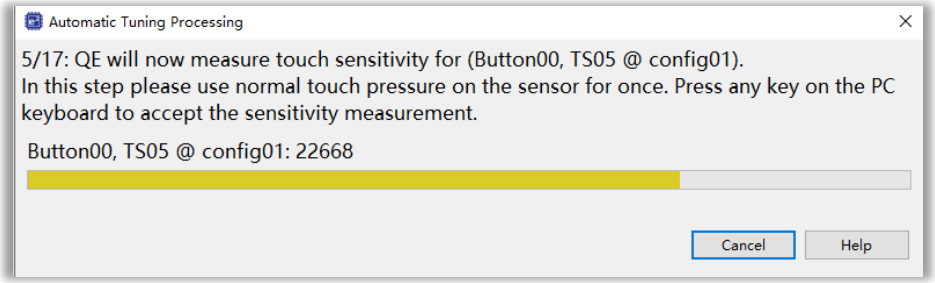
在没有按下触摸按键时，自容式按键的灵敏度测量的基准值为 **15360**。



按照提示，使用手指以正常压力按住"**Button00/TS05**"的触摸按键，

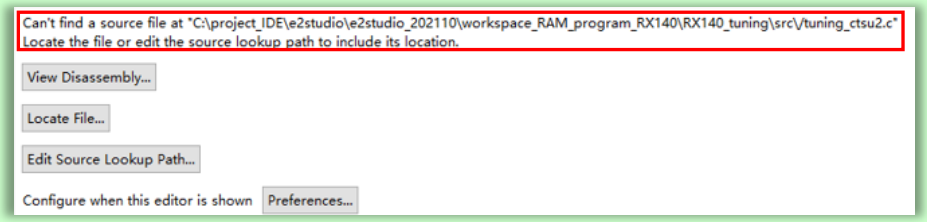
此时黄色进度条将根据手指按压触摸按键的力度而变化，

保持期望的按压力度，同时按下 PC 键盘的任意键，接受该触摸按键的灵敏度测量。



NOTE

在自动调整过程(**Auto Tuning Process**)对话框的背后，提示无法找到源文件，并不是错误，可忽略。



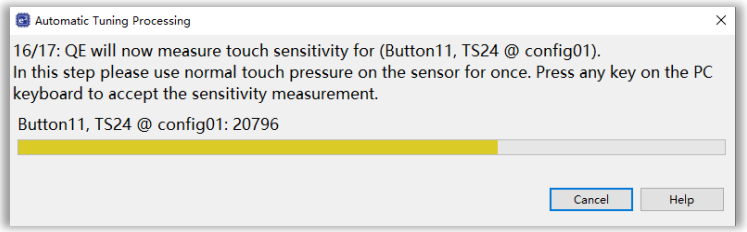
2.4.10

第六步~第十七步：

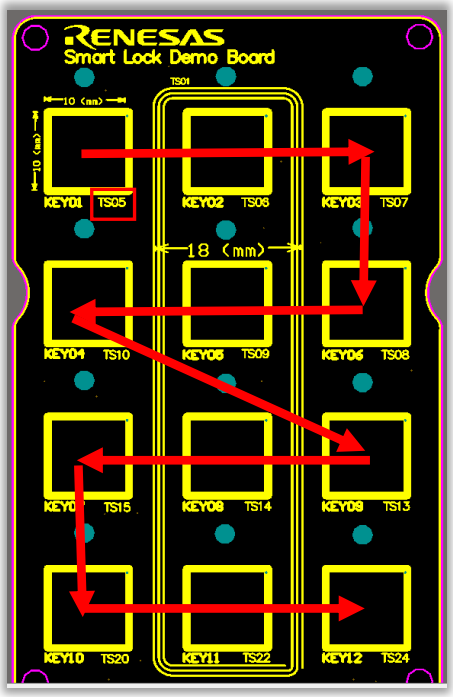
按照上述方式，依次对 **12** 个按键进行灵敏度测量

12 个按键的灵敏度测量顺序从 TS 通道号的最小值开始，然后从小到大

依次为**TS05、TS06、TS07，TS08，TS09，TS10，TS13，TS14，TS15，TS20，TS22，TS24**



评价板每个触摸按键的右下方标注了"**TS** 通道"的号码，"**TS** 通道"灵敏度测量顺序，如下图红色箭头所示。

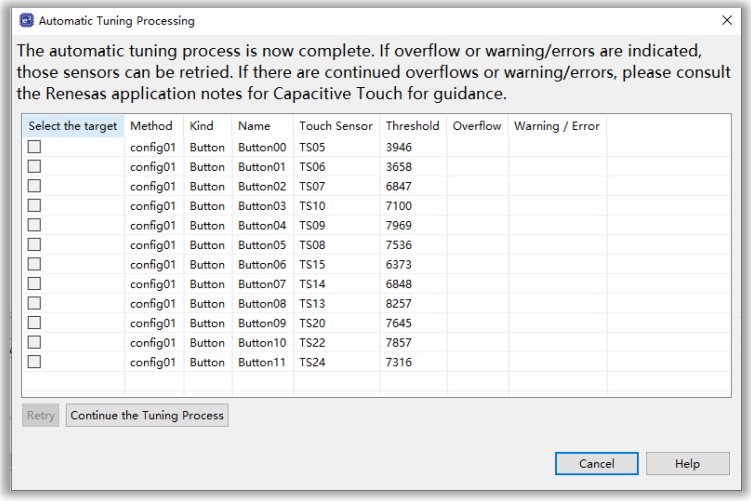


2.4.11

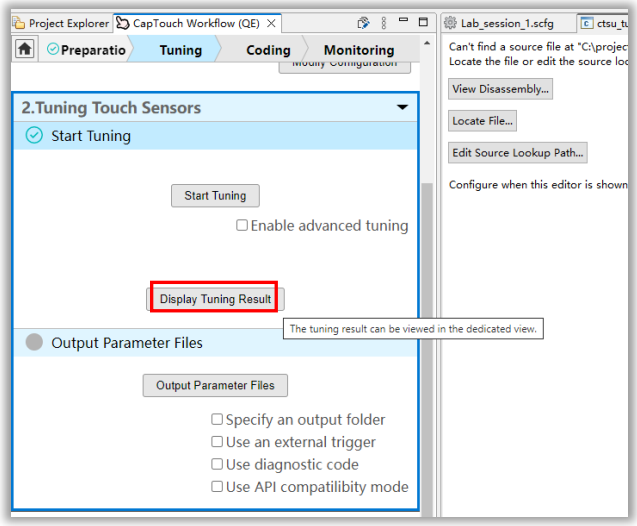
完成自动调整过程(**Auto Tuning Process**)后，自动弹出结果，显示了各个触摸按键的阈值 **Threshold**。

点击"**Continue the Tuning Process**"，自动调整过程的结果对话框关闭。

自动调整过程(**Auto Tuning Process**) 全部完成。



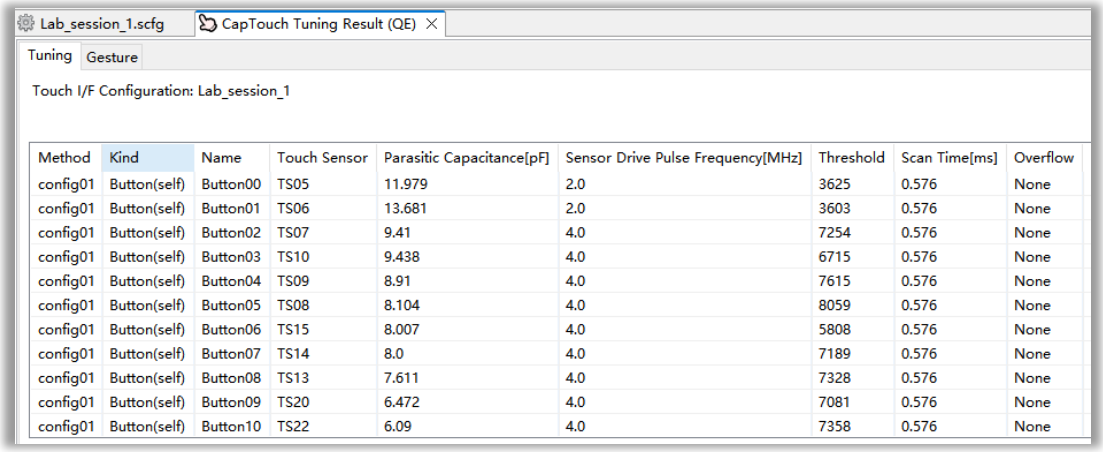
2.4.12 在"**Cap Touch Workflow**"的"**2.Tuning Touch Sensors**"中，点击"**Display Tuning Result**"



自动调整过程(**Auto Tuning Process**)的结果，如下图所示:

包括 **Method，Kind，Name，Touch Sensor，Parasitic Capacitance，Sensor Driver Pulse Frequency，Threshold，Scan Time**，以及 **Overflow** 等重要信息。

(受环境影响，重新进行自动调整过程时，寄生电容值会有细微差异，传感器驱动脉冲频率也有可能因寄生电容值的变化发生变化；阈值 **Threshold** 也会因按压力度的变化发生变化，阈值也可以在配置文件中直接修改)


Lab_session_1.scfg | CapTouch Tuning Result (QE)

Tuning | Gesture

Touch I/F Configuration: Lab_session_1

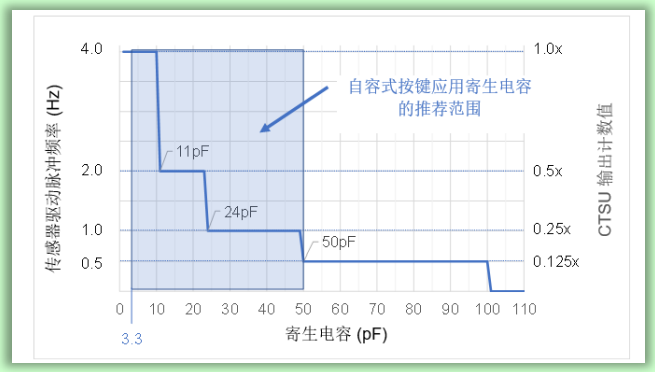
Method	Kind	Name	Touch Sensor	Parasitic Capacitance[pF]	Sensor Drive Pulse Frequency[MHz]	Threshold	Scan Time[ms]	Overflow
config01	Button(self)	Button00	TS05	11.979	2.0	3625	0.576	None
config01	Button(self)	Button01	TS06	13.681	2.0	3603	0.576	None
config01	Button(self)	Button02	TS07	9.41	4.0	7254	0.576	None
config01	Button(self)	Button03	TS10	9.438	4.0	6715	0.576	None
config01	Button(self)	Button04	TS09	8.91	4.0	7615	0.576	None
config01	Button(self)	Button05	TS08	8.104	4.0	8059	0.576	None
config01	Button(self)	Button06	TS15	8.007	4.0	5808	0.576	None
config01	Button(self)	Button07	TS14	8.0	4.0	7189	0.576	None
config01	Button(self)	Button08	TS13	7.611	4.0	7328	0.576	None
config01	Button(self)	Button09	TS20	6.472	4.0	7081	0.576	None
config01	Button(self)	Button10	TS22	6.09	4.0	7358	0.576	None

NOTE

这里要重点关注触摸按键的寄生电容，寄生电容从根本上决定了触摸按键的灵敏度。

寄生电容值在完成自动调整过程(**Auto Tuning Process**)后，不可修改。

不同的寄生电容值，**QE** 会自动设定不同的传感器驱动脉冲频率，寄生电容值与传感器驱动脉冲频率的关系，如下图所示，不同频率的切换点为 **11pF，24pF，50pF。**

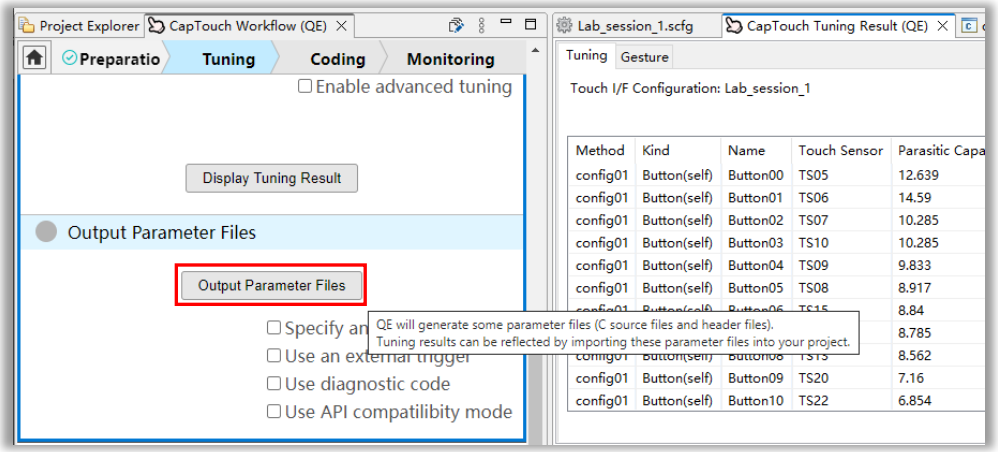


从自动调整过程(**Auto Tuning Process**)的结果中，可以看到 **TS05** 和 **TS06** 由于寄生电容超过了 **11pF**，因此只能采用 **2MHz** 的传感器驱动脉冲频率，通过自动调整过程(**Auto Tuning Process**)产生的阈值为 **3625** 和 **3603**；而 **TS07** 和 **TS10** 的寄生电容值小于 **11pF**，可以采用 **4MHz** 的传感器驱动脉冲频率，通过自动调整过程(**Auto Tuning Process**)产生的阈值为 **7254** 和 **6715**，是 **TS05** 和 **TS06** 的两倍。

Method	Kind	Name	Touch Sensor	Parasitic Capacitance[pF]	Sensor Drive Pulse Frequency[MHz]	Threshold	Scan Time[ms]	Overflow
config01	Button(self)	Button00	TS05	11.979	2.0	3625	0.576	None
config01	Button(self)	Button01	TS06	13.681	2.0	3603	0.576	None
config01	Button(self)	Button02	TS07	9.41	4.0	7254	0.576	None
config01	Button(self)	Button03	TS10	9.438	4.0	6715	0.576	None

2.4.13 **输出参数文件**

在"**Cap Touch Workflow**"的"**2.Tuning Touch Sensors**"中，点击"**Output Parameter FIles**"

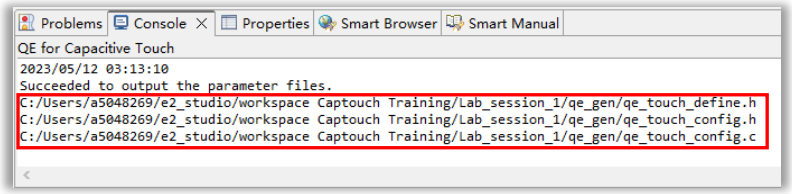


生成以下三个参数文件

Qe_touch_define.h

Qe_touch_config.h

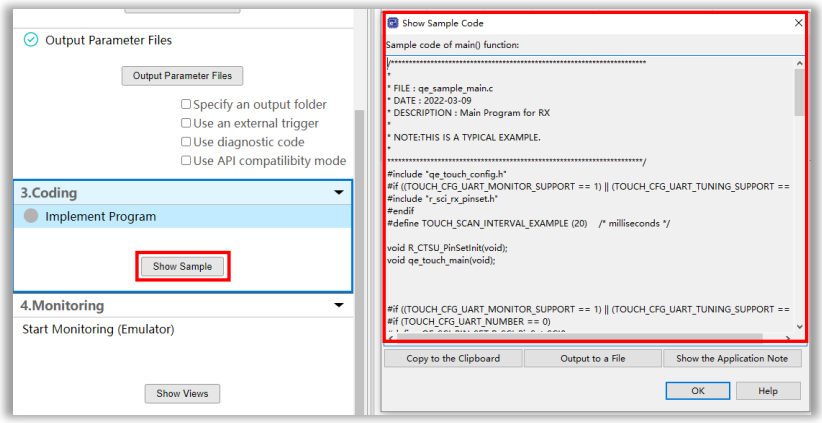
QE_touch_config.c



2.4.14 **生成样例程序**

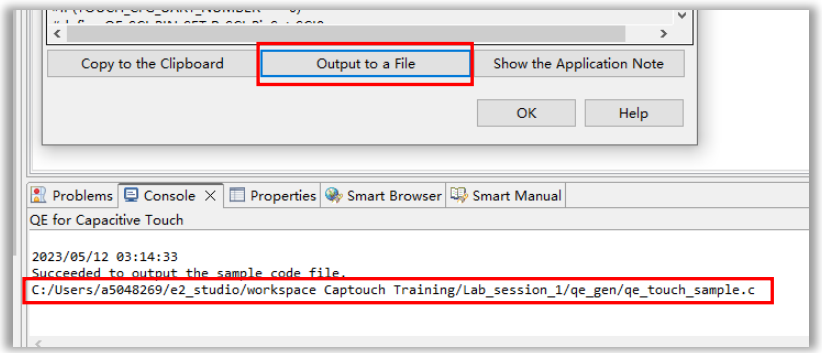
在"**Cap Touch Workflow**"的"**3.Coding**"中，点击"**Show Sample**"

弹出样例程序的预览对话框



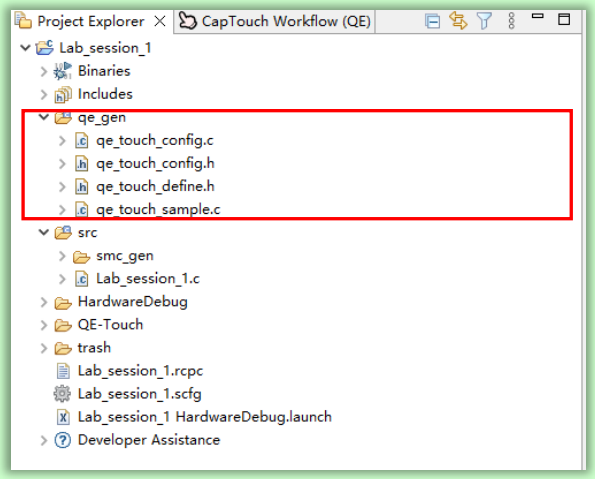
2.4.15 点击对话框中的"**Output a File**"

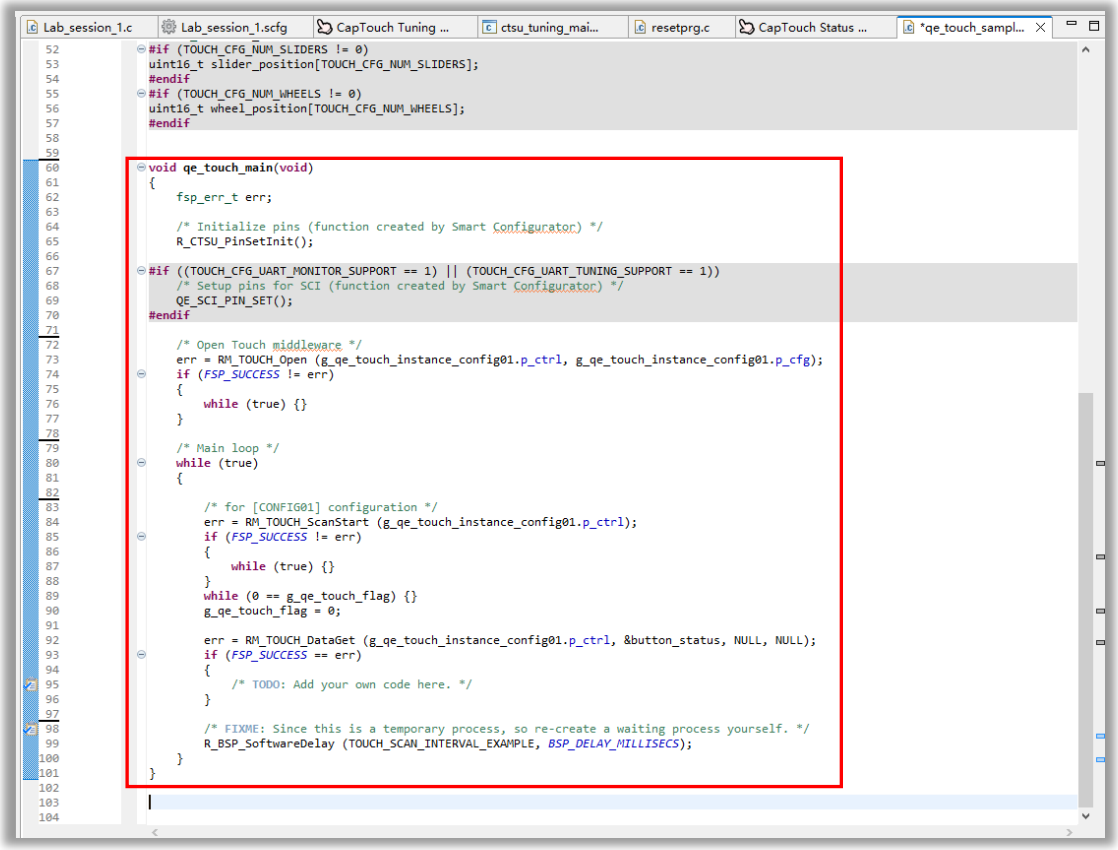
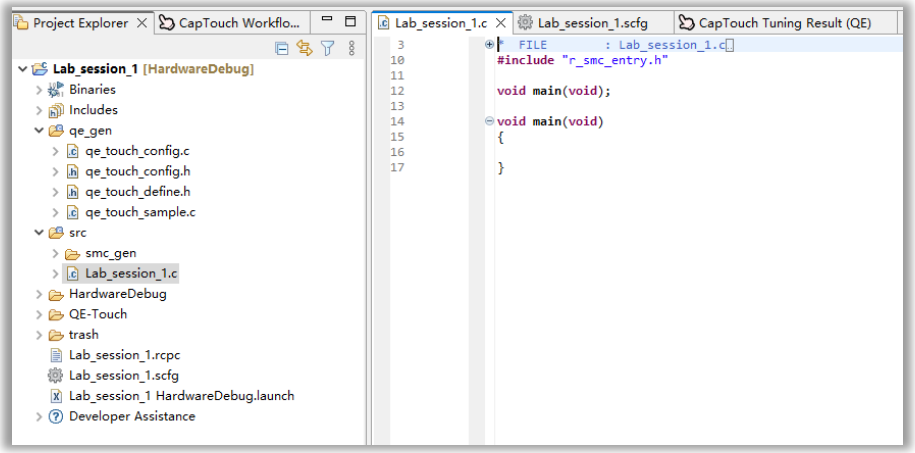
生成含有触摸样例应用程序的文件"**qe_touch_sample.c**"



NOTE

在"**Cap Touch Workflow**"的"**3.Coding**"步骤生成的全部文件，如下所示

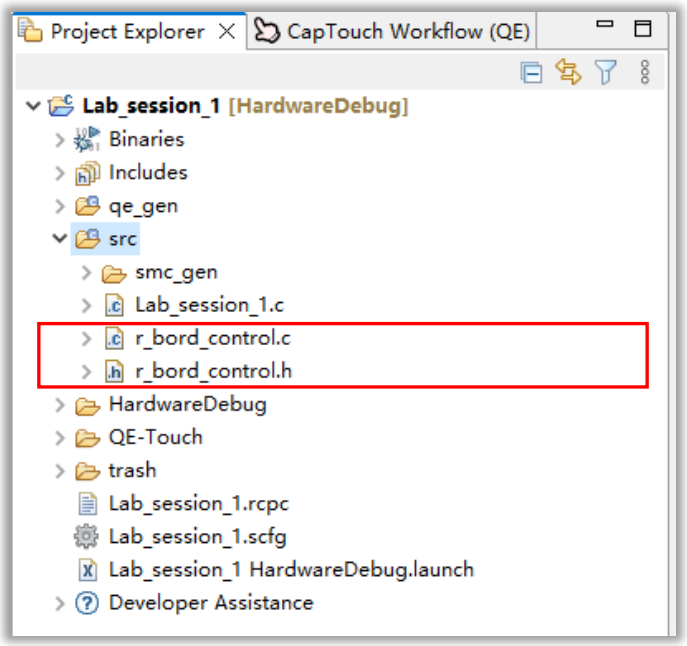


2.4.16	在"**qe_touch_sample.c**"中，包含触摸应用样例程序"**qe_touch_main**（）", 如下图所示: 
NOTE	触摸样例应用程序，主要通过以下三个 API 完成操作。 **RM_TOUCH_Open()** **RM_TOUCH_ScanStart()** **RM_TOUCH_DataGet()**
2.5	**增加应用程序。**
2.5.1	在 **main** 主函数中增加触摸应用函数的调用 在"**Project Explorer**"中，选择工程 **Lab_Session_1** → 文件夹 **src** → 文件 **Lab_session_1.c** 

2.5.2

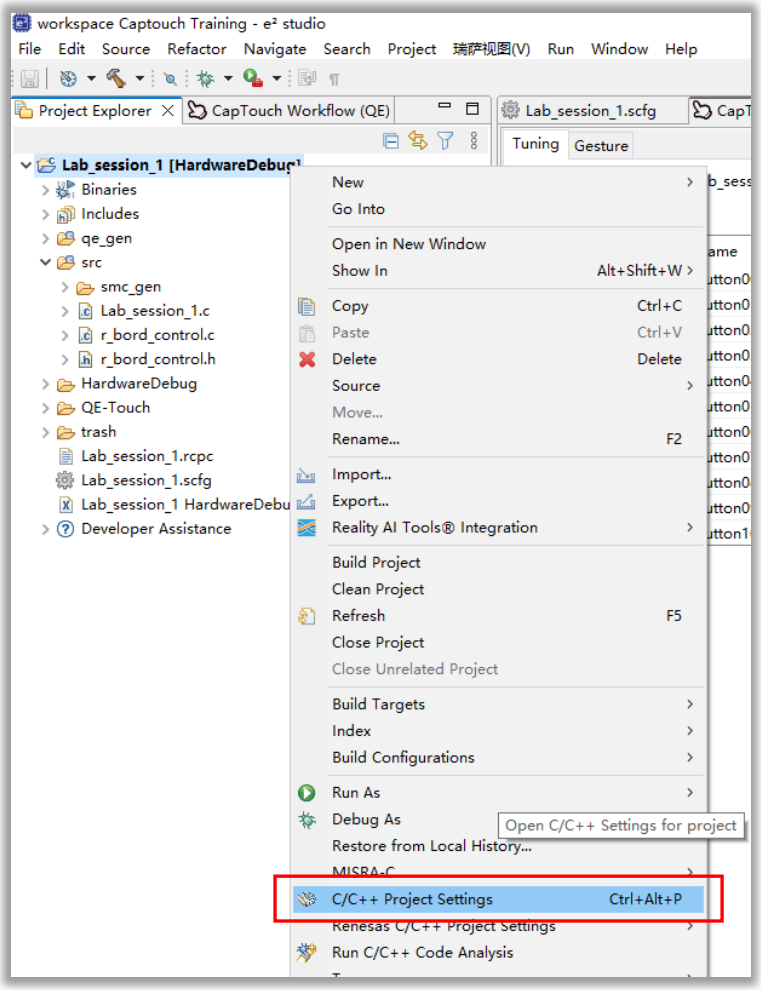
增加指示触摸按键状态的 **LED** 的驱动程序

将"**r_bord_control.c**"和"**r_bord_control.h**"两个文件拷贝到工程 **Lab_Session_1** → **src** 文件夹下



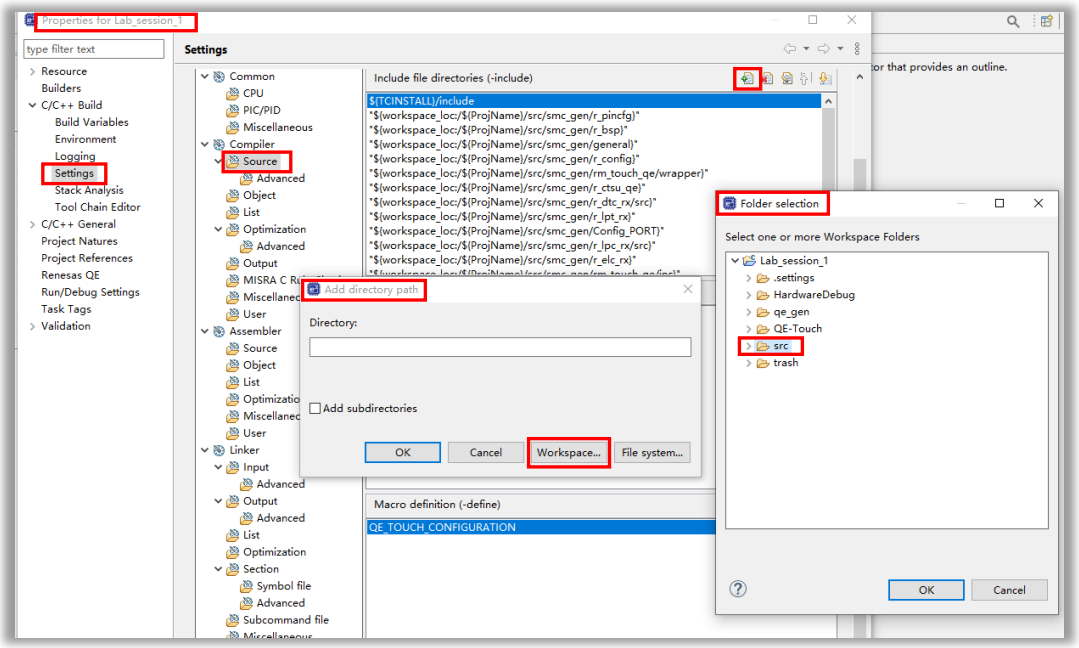
2.5.3

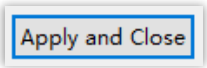
右键单击工程名"**Lab_session_1**", 在弹出的菜单中，选择"**C/C++ Setting for project**"

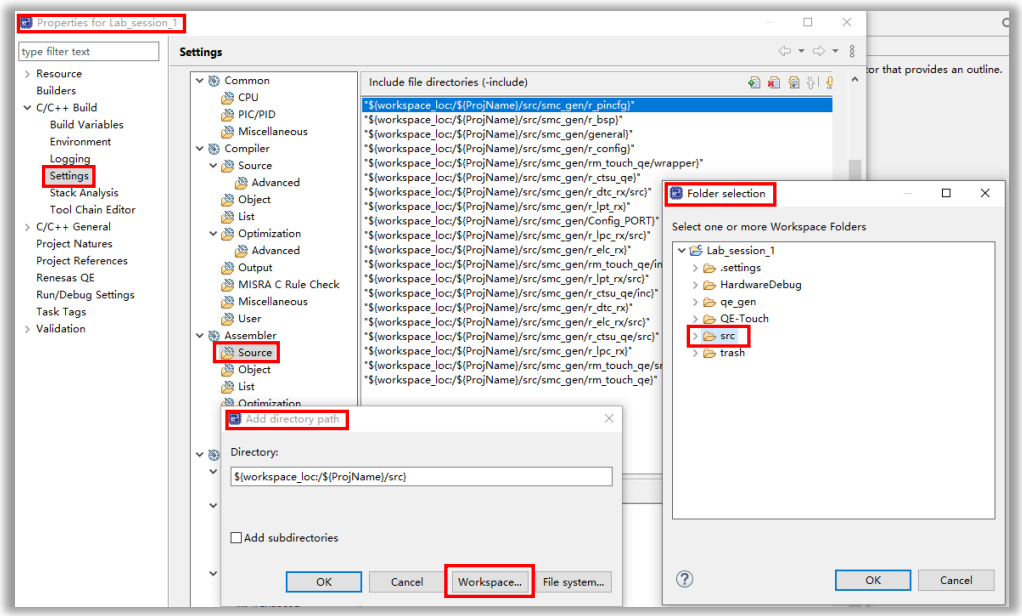


2.5.4

在弹出的 "**Properties for session_1**"对话框中，选择"**Settings**"，
在"**Settings**"的目录中选择"Compiler"中的"**Source**", 在右侧的具体设定中点击右上角的  图标，
在弹出的"**Add directory path**"对话框中，点击"**Workspace**"
在弹出的"**Folder selection**"中，选择"**src**"目录，单击 **OK**
单击"**Add directory path**"对话框中的 **OK**

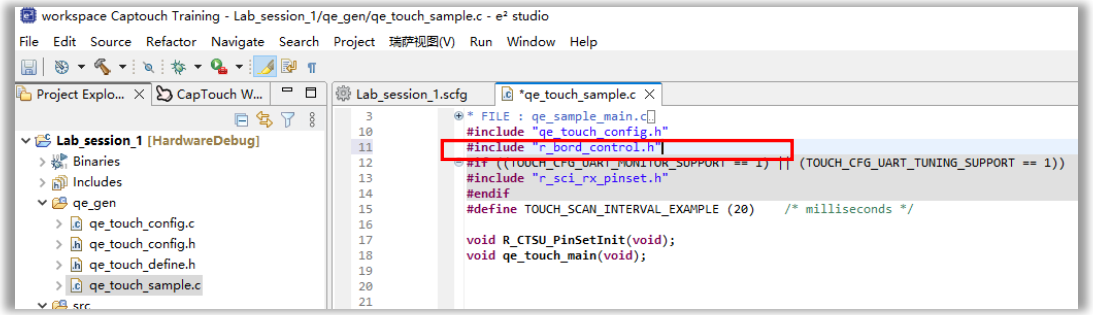


在"**Settings**"的目录中选择"**Assembler**"中的"**Source**", 在右侧的具体设定中点击右上角的  图标，
在弹出的"**Add directory path**"对话框中，点击"**Workspace**"
在弹出的"**Folder selection**"中，选择"**src**"目录，单击 **OK**
单击"**Add directory path**"对话框中的 **OK**
最后，单击"**Properties for session_1**"对话框右下角的  完成设定



2.5.5

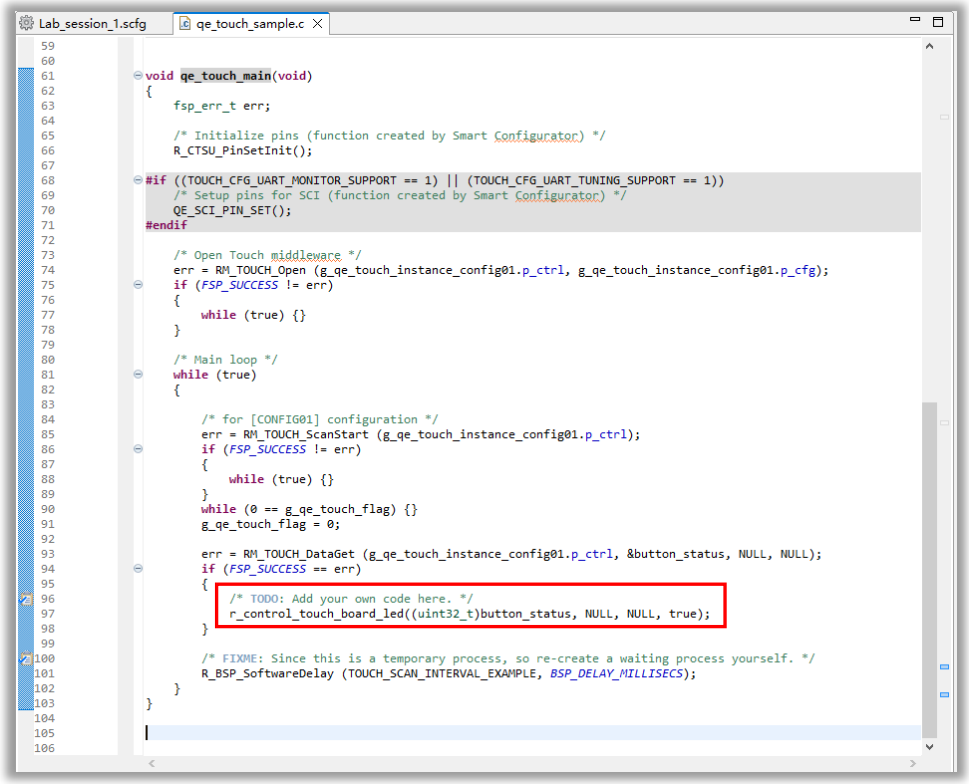
在"文件 **qe_touch_sample.c**"中，添加**#include "r_bord_control.h"**



2.5.6

在"**void qe_touch_main(void)** "中添加 **LED** 控制函数的调用，如下所示：

```
/* TODO: Add your own code here. */

r_control_touch_board_led((uint32_t)button_status, NULL, NULL, true);
```

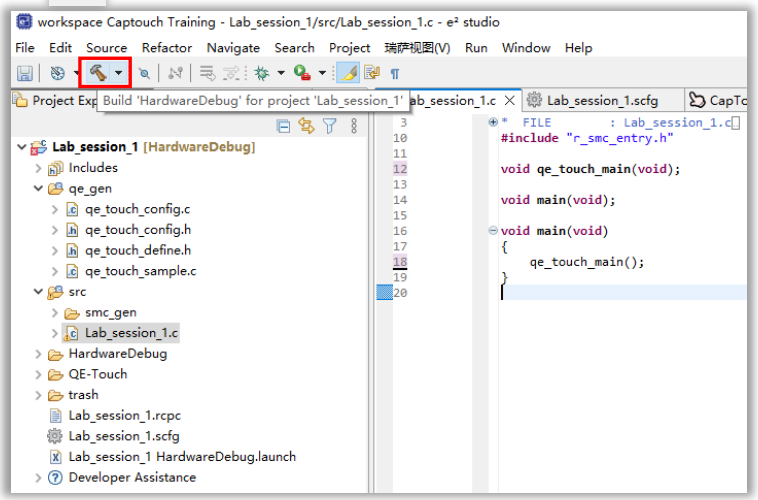


2.5.7

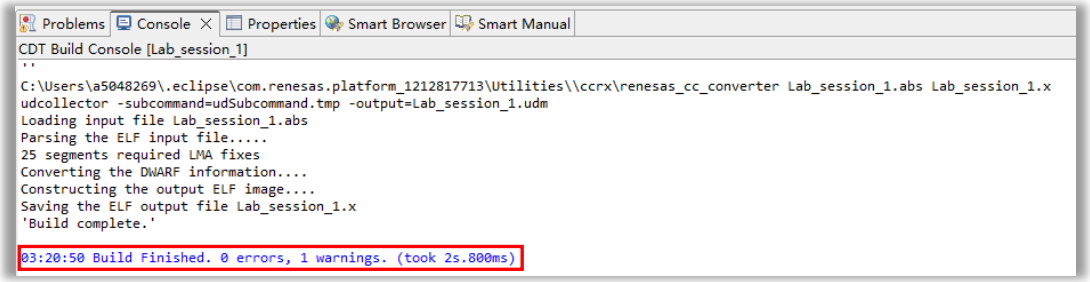
按以下方式修改文件"**Lab_session_1.c**"，黄色背景的代码为修改或者增加的部分。

```
#include "r_smc_entry.h"

void qe_touch_main(void);

void main(void);

void main(void)
{
    qe_touch_main();
}
```

2.5.8 点击  按钮编译工程

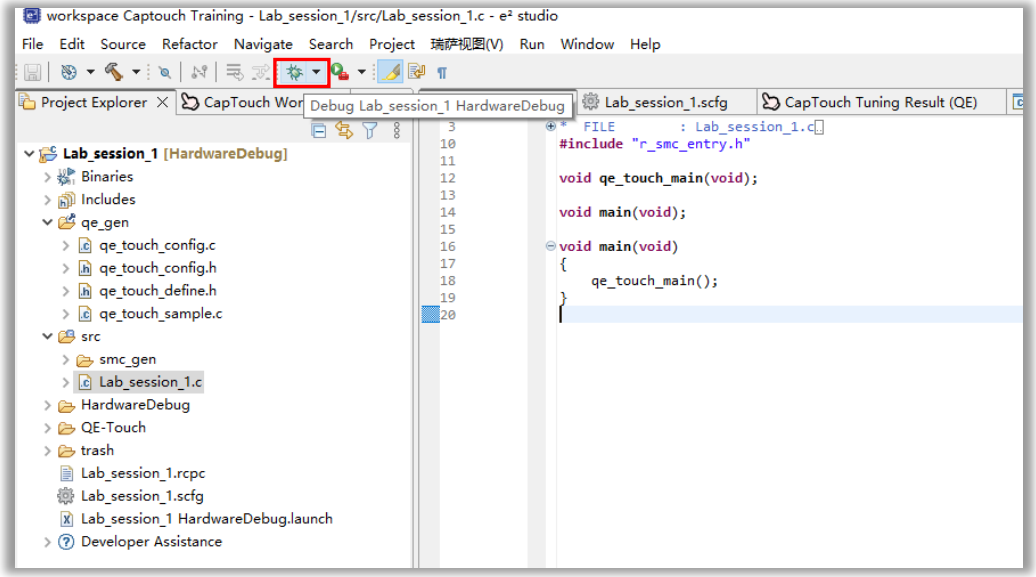


2.5.9 如果没有错误，将显示如下结果：



2.6 **运行程序.**

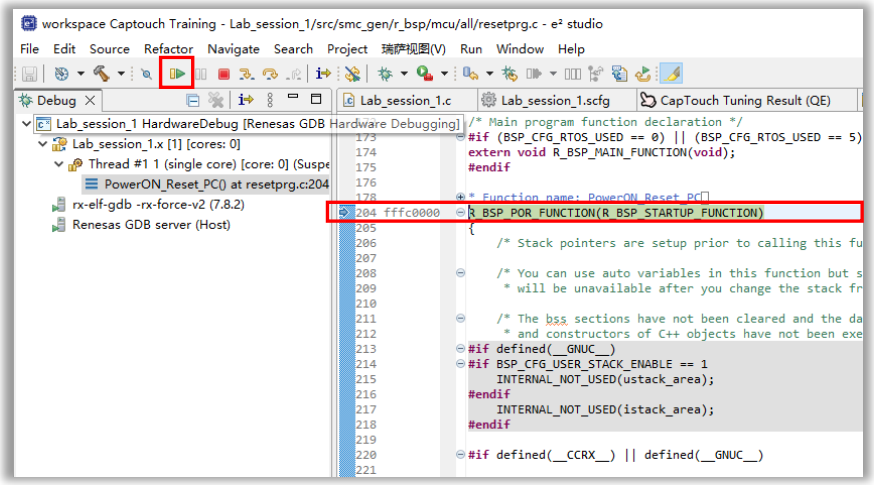
2.6.1 点击  按钮进入仿真状态，如下图所示：



2.6.2

程序停止在"文件 **resetprg.c** "的"204 行"

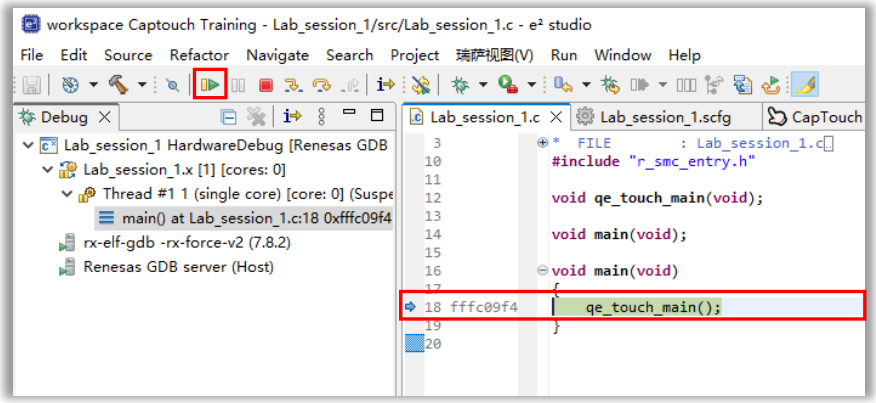
点击  按钮，继续



2.6.3

程序停止在"文件 **Lab_session_1.c**"的"**18** 行"，如下图所示：

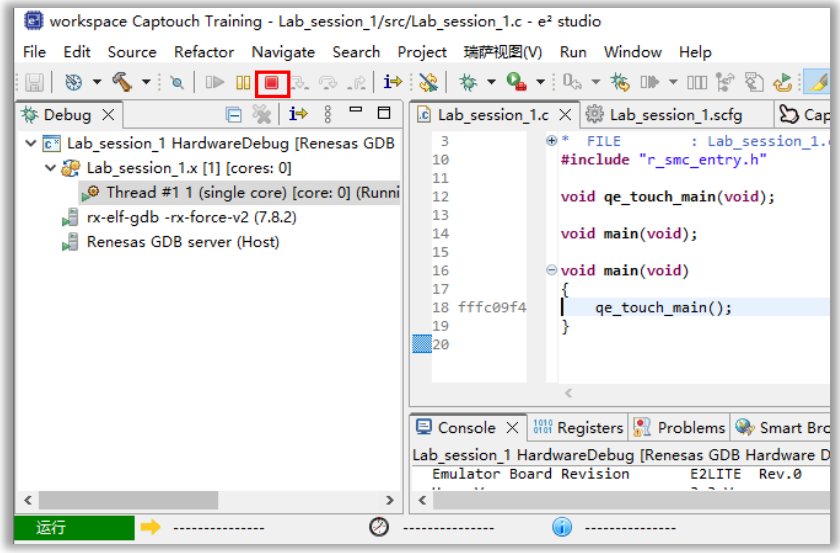
点击  按钮，继续

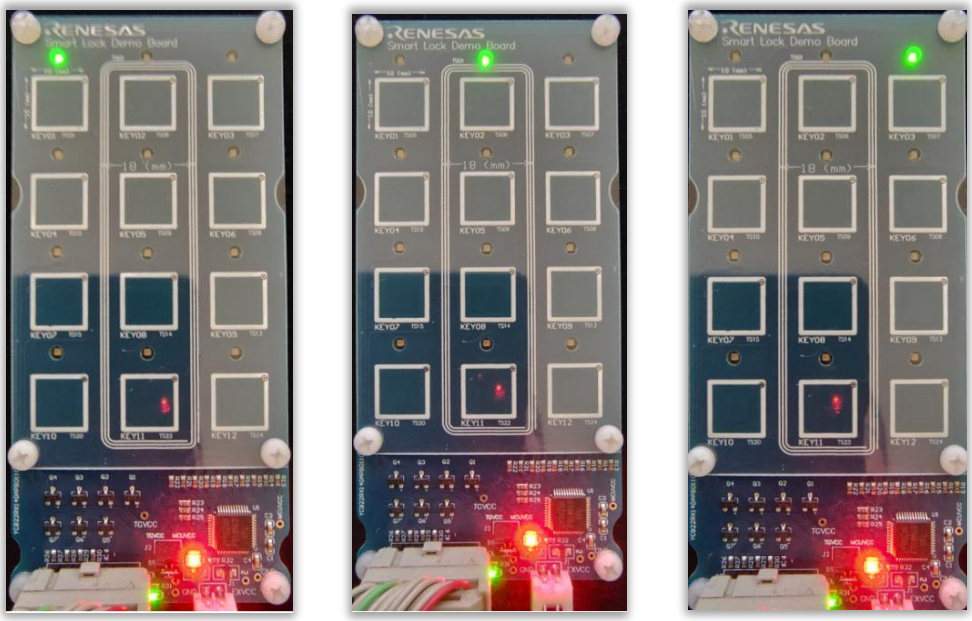
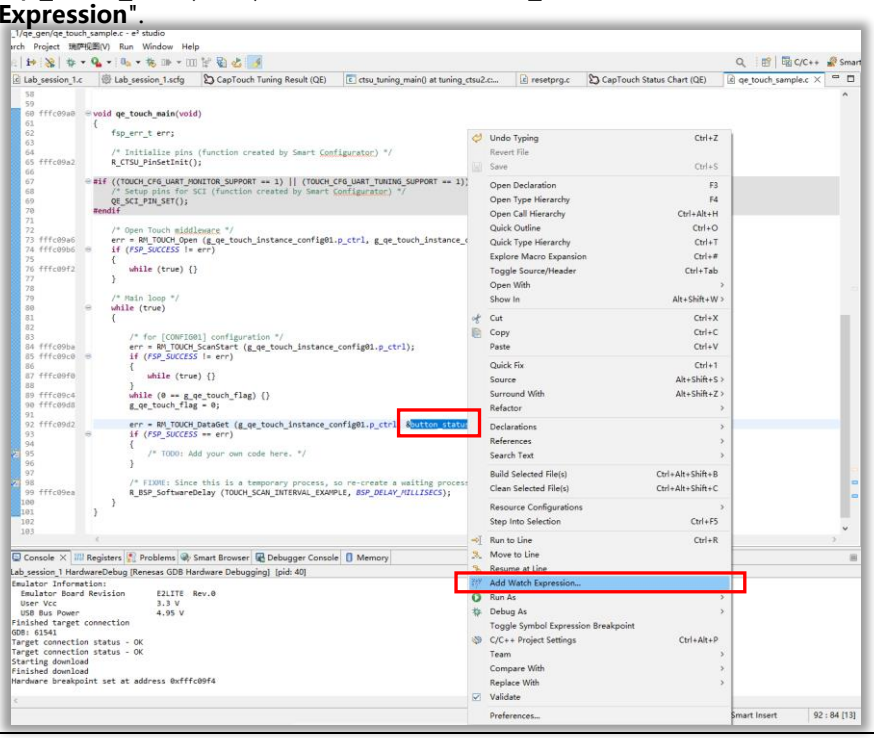


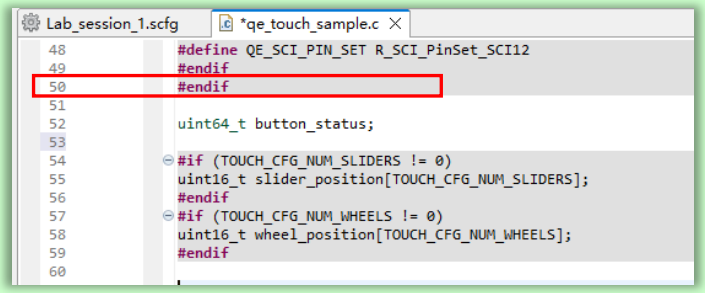
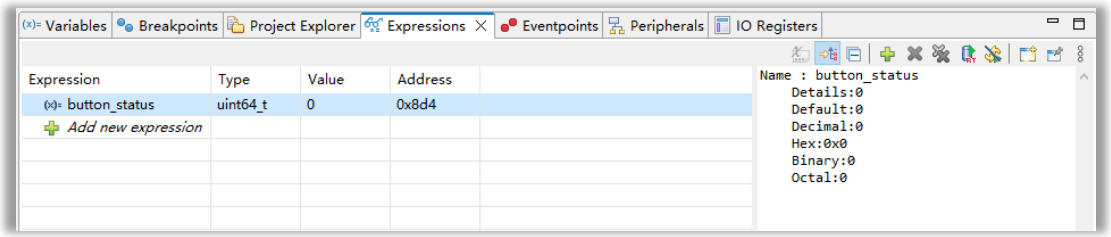
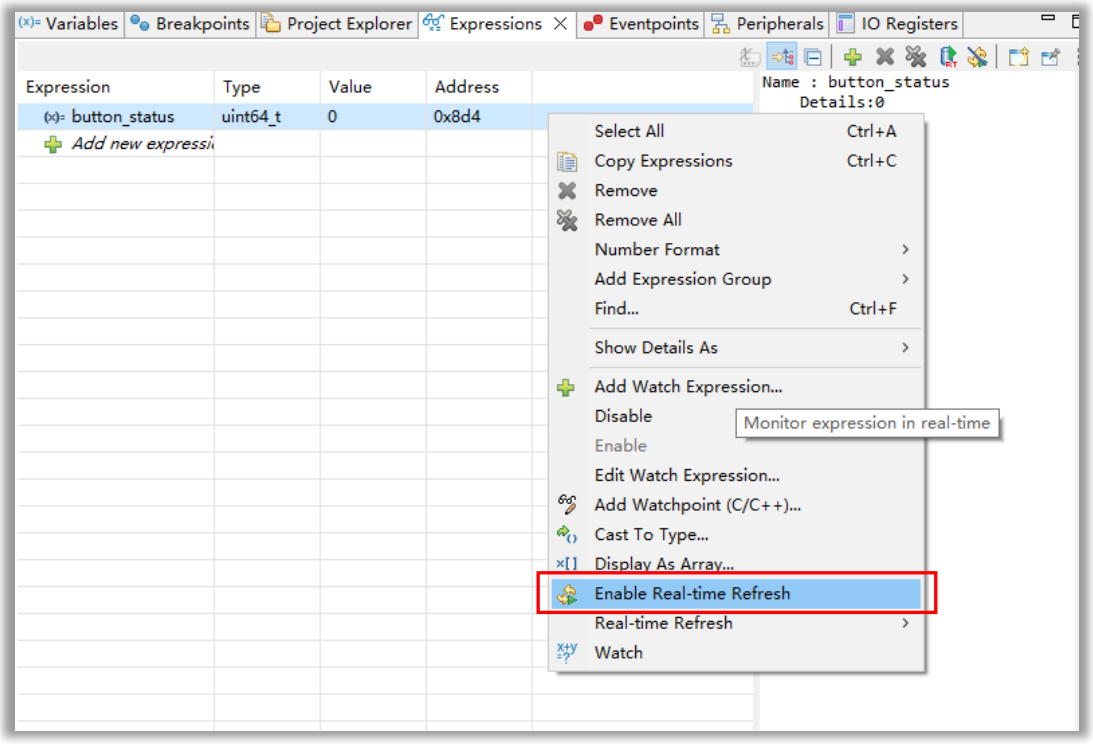
2.6.4

此时，程序进入全速运行状态。

点击  ，可以停止程序运行。

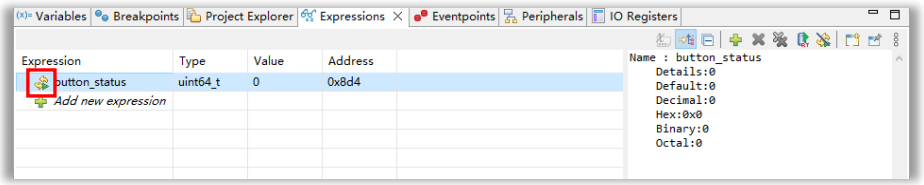


2.7	**使用指示触摸按键状态的 LED 监控触摸行为**
2.7.1	按照"**2.6 运行程序**"小节介绍的方法，在仿真状态下全速运行程序。 当某一个触摸按键被按下时，触摸按键上方对应的绿色 LED 被点亮。 
2.8	**使用全局变量 button_status 监控触摸行为**
2.8.1	点击  按钮进入仿真状态。 点击两次  按钮，程序停止在"文件 **Lab_session_1.c**"的"**18** 行" 此时，打开"文件 **qe_touch_sample.c**" 在"`void qe_touch_main(void)` "中, 在全局变量"`button_status`"上单击右键，在菜单中选择"**Add Watch Expression**". 

NOTE	"**button_status**"为 **64** 位全局变量 在"**qe_touch_sample.c**"中定义, 如下所示: 
2.8.2	"**Add Watch Expression**"添加完成后，显示在"**Expression**"窗口 
2.8.3	在"**Expression**"窗口，在全局变量"**button_status**"上单击右键， 在弹出的菜单中，选择"**Enable Real-time Refresh**" 

2.8.4

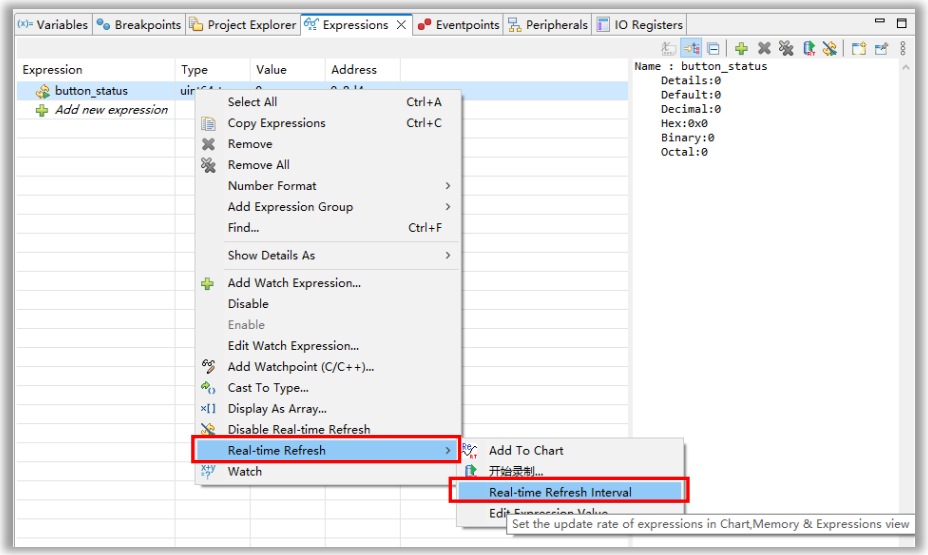
选择"**Enable Real-time Refresh**"后，全局变量"**button_status**"前面图标变为  ，如下图所示：



2.8.5

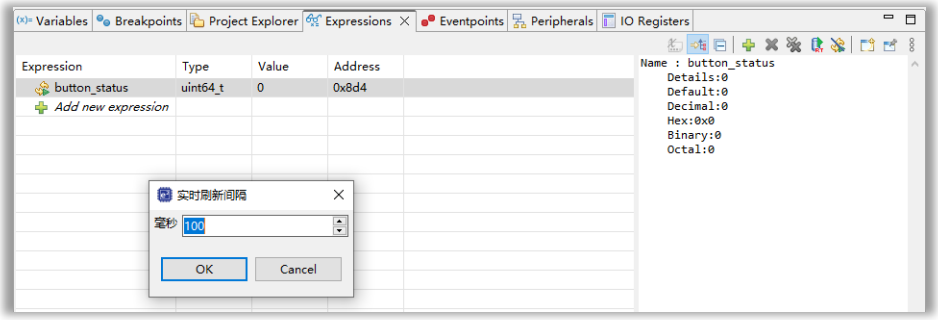
在"**Expression**"窗口，在全局变量"**button_status**"上单击右键，

在弹出的菜单中选择"**Real-time Refresh**"，在弹出的下一级菜单中选择" **Real-time Refresh Interval**"



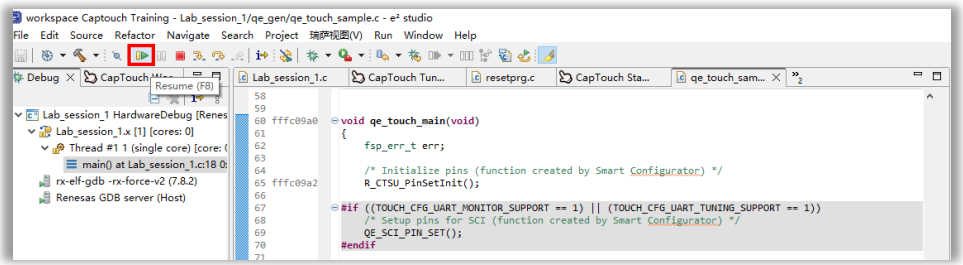
2.8.6

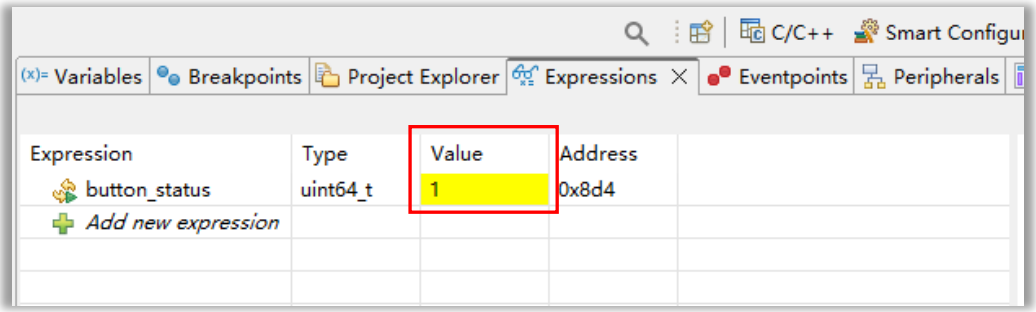
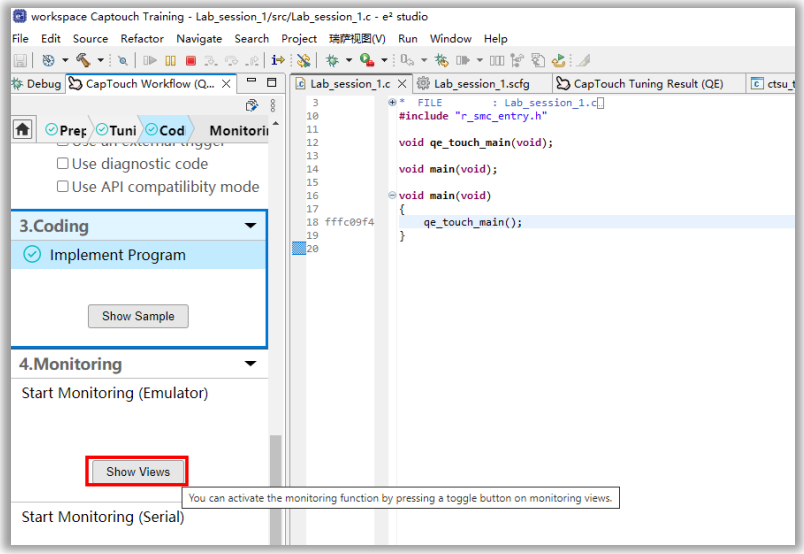
在弹出的对话框中，将"**Real-time Refresh Interval**"设定为"**100ms**"

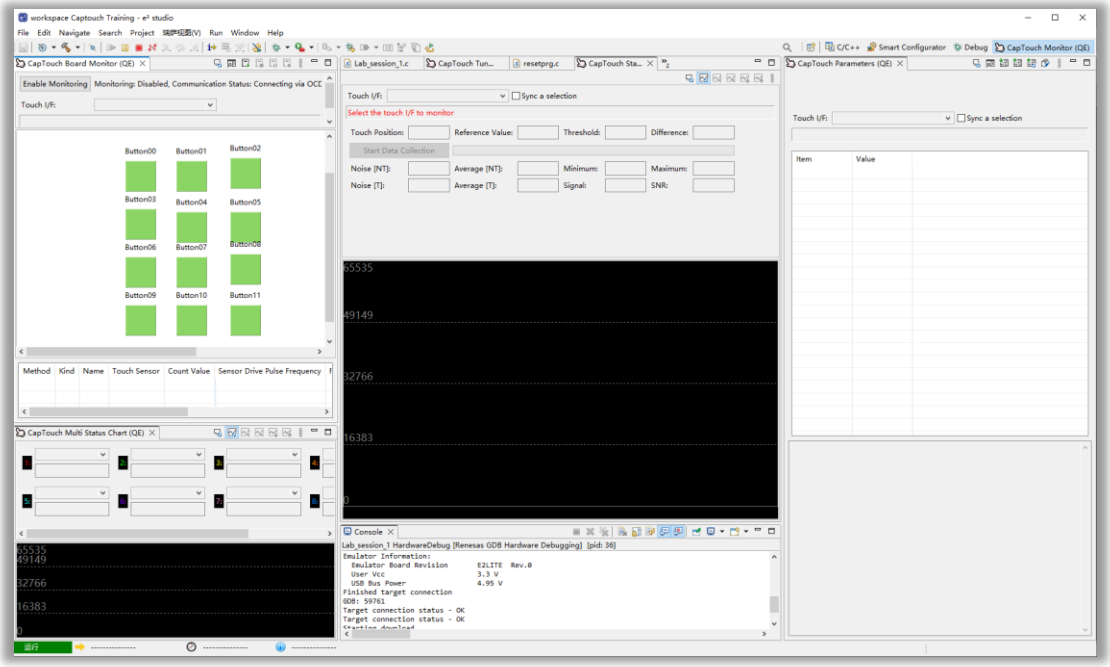


2.8.7

点击  按钮，全速运行程序。



2.8.8	按下某一个按键，在"**Expression**"窗口，可实时看到全局变量"**button_status**"的数值。  **触摸 TS_05(Key01) = 1** **触摸 TS_06(Key02) = 2** **触摸 TS_07(Key03) = 4** **触摸 TS_10(Key04) = 32** **触摸 TS_09(Key05) = 16** **触摸 TS_08(Key06) = 8** **触摸 TS_15(Key07) = 256** **触摸 TS_14(Key08) = 128** **触摸 TS_13(Key09) = 64** **触摸 TS_20(Key10) = 512** **触摸 TS_22(Key11) = 1024** **触摸 TS_24(Key12) = 2048** 同时触摸多个按键.例如: **触摸 TS_05(Key01) + TS_06(Key02) + TS_07(Key03) =1+2+4 = 7** 以此类推
2.9	**使用 QE for Cap Touch 监控触摸底层数据以及触摸行为**
2.9.1	按照"**2.6 运行程序**"小节介绍的方法，在仿真状态下全速运行程序。 在"**Cap Touch Workflow**"的"**4.monitoring**"中，点击"**Start Monitoring(Emulator)**"下方的"**Show Views**" 

2.9.2	"**e2 studio**"切换到"**QE for CapTouch**", 显示如下监控窗口。 界面左侧显示：**CapTouch Board Monitor (QE) View** **CapTouch Multi Status Chart (QE) View** 界面中间显示：**CapTouch Status Chart (QE) View** 界面右侧显示：**CapTouch Parameters (QE) View** 

2.9.3

在"**QE for CapTouch**"的各个监控窗口中设定需要监控的"**Touch I/F**", 例如:

CapTouch Board Monitor (QE) View ： 设定 **Button00 @config01**

CapTouch Multi Status Chart (QE) View ： 设定 **Button00，TS05 @config01**

(最多设定 **8** 个 **TS** 通道) 设定 **Button01，TS06 @config01**

设定 **Button02，TS07 @config01**

设定 **Button03，TS10 @config01**

设定 **Button04，TS09 @config01**

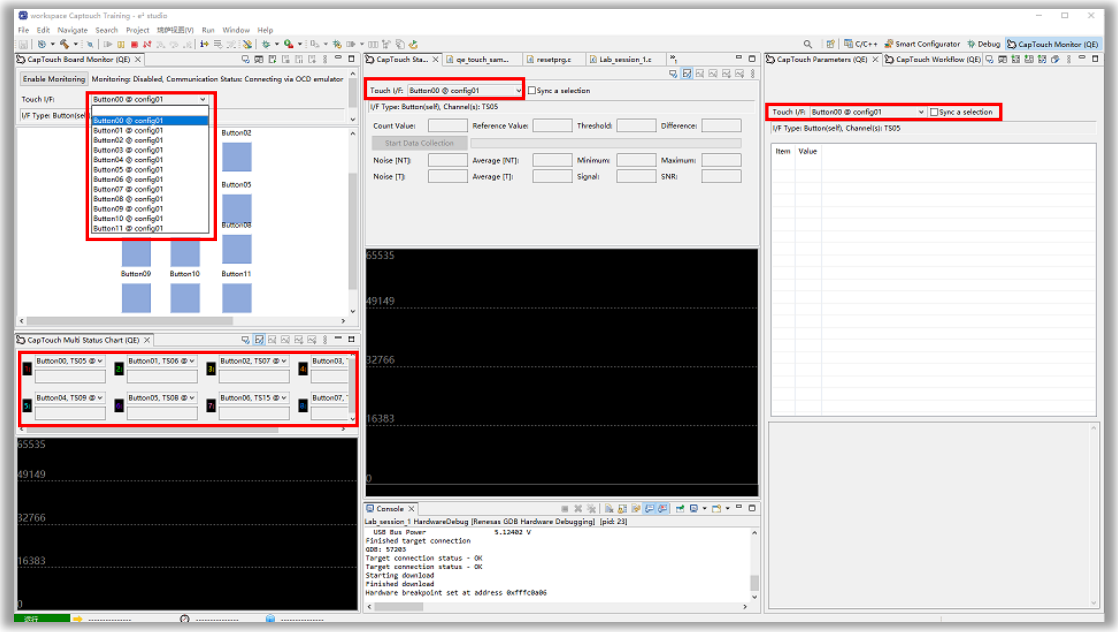
设定 **Button05，TS08 @config01**

设定 **Button06，TS15 @config01**

设定 **Button07，TS14 @config01**

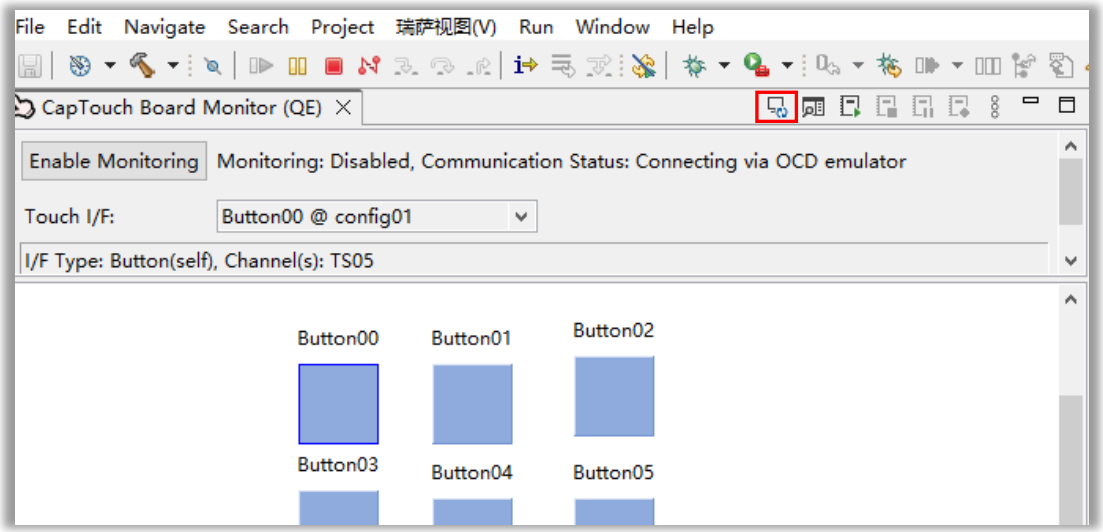
CapTouch Status Chart (QE) View ： 设定 **Button00 @config01**

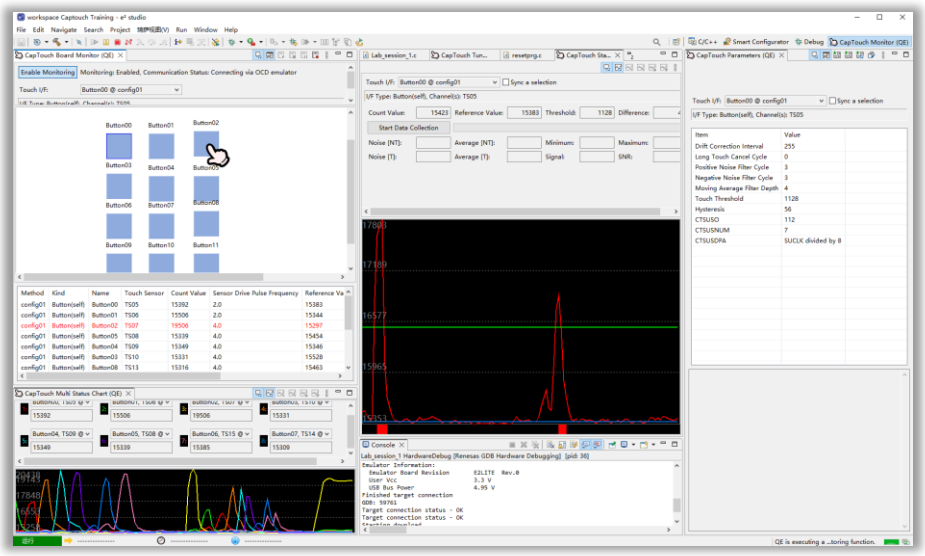
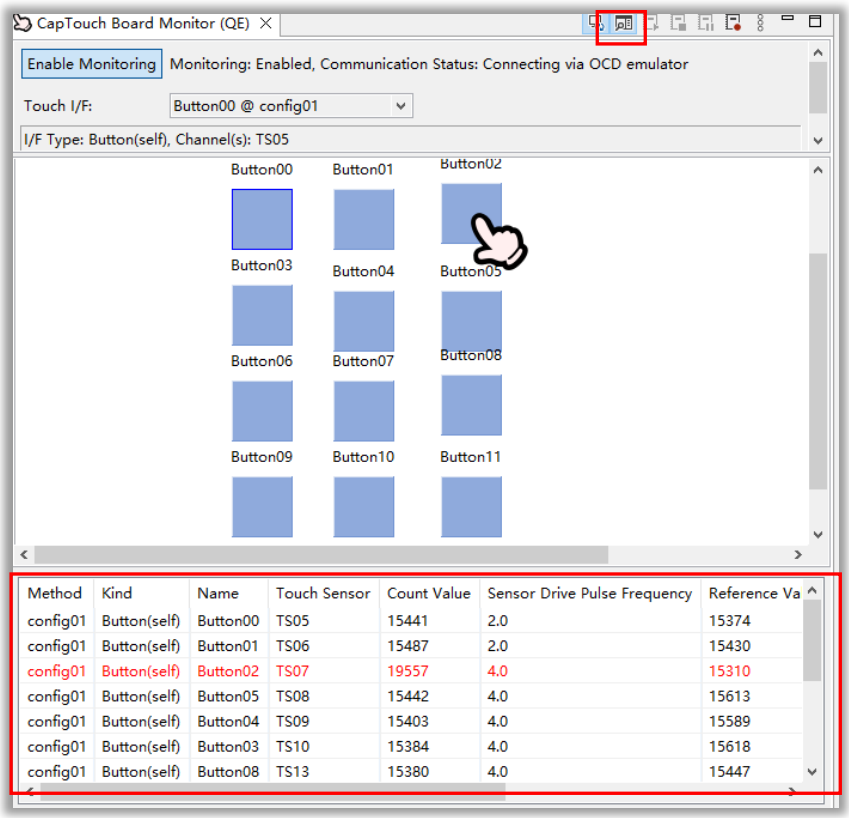
CapTouch Parameters (QE) View ：设定 **Button00 @config01**



2.9.4

在"**CapTouch Board Monitor (QE) View**" 中，点击 "**Enabling Monitoring**" 按钮  。



NOTE	"**Enabling Monitoring**"按钮  ，在各个监控窗口中都有，在使能监控功能时，点击个监控窗口中的"**Enabling Monitoring**"按钮可以，
2.9.5	点击 "**Enabling Monitoring**" 按钮  ，使能监控功能后，各个监控窗口将以数据或者图形的形式，显示触摸底层数据或者触摸行为。 
2.9.6	在"**CapTouch Board Monitor (QE) View**" 中，触摸按键的 **On/Off** 状态，通过"手形图标"  显示。 如下图，**Button02** 即为按下状态。 可通过点击  按钮，打开高级模式，将在页面下方动态显示各个按键的具体信息，包括 **Method**，**Kind**，**Name**，**Touch Sensor**，**Parasitic Capacitance**，**Sensor Driver Pulse Frequency**，**Threshold**，**Scan Time**，以及 **Overflow** 等等。 

CapTouch Board Monitor (QE)

Enable Monitoring | Monitoring: Enabled, Communication Status: Connecting via OCD emulator

Touch I/F: Button00 @ config01

I/F Type: Button(self), Channel(s): TS05

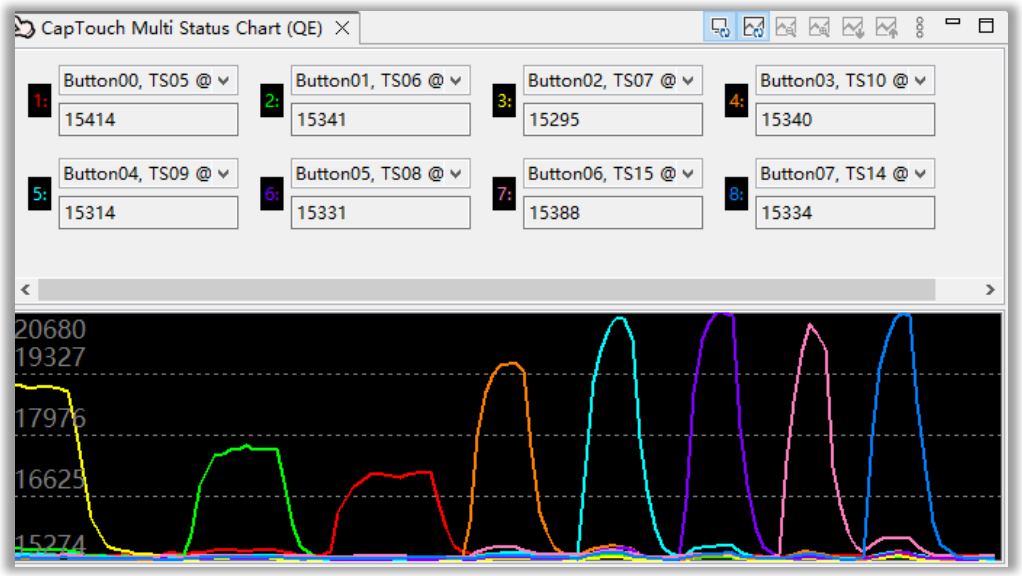
Button00 Button01 Button02 Button03 Button04 Button05 Button06 Button07 Button08 Button09 Button10 Button11

Method	Kind	Name	Touch Sensor	Count Value	Sensor Drive Pulse Frequency	Reference Va
config01	Button(self)	Button00	TS05	15441	2.0	15374
config01	Button(self)	Button01	TS06	15487	2.0	15430
config01	Button(self)	Button02	TS07	19557	4.0	15310
config01	Button(self)	Button05	TS08	15442	4.0	15613
config01	Button(self)	Button04	TS09	15403	4.0	15589
config01	Button(self)	Button03	TS10	15384	4.0	15618
config01	Button(self)	Button08	TS13	15380	4.0	15447

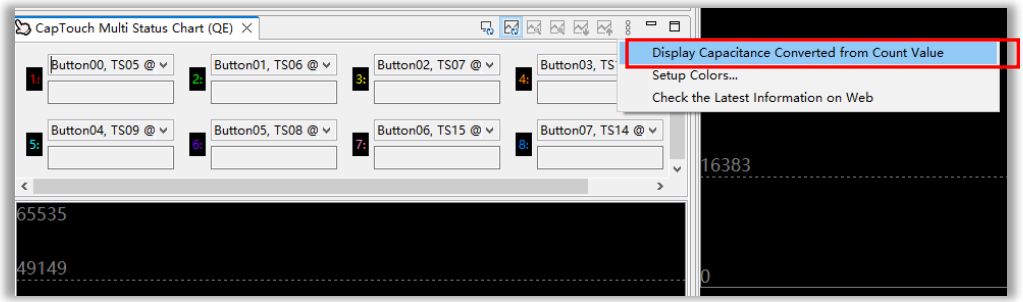
2.9.7

在"**CapTouch Multi Status Chart (QE) View**"中，可在同一监控窗口，实时以数值的形式和波形的形式，显示多个 **TS** 通道的测量值数据。

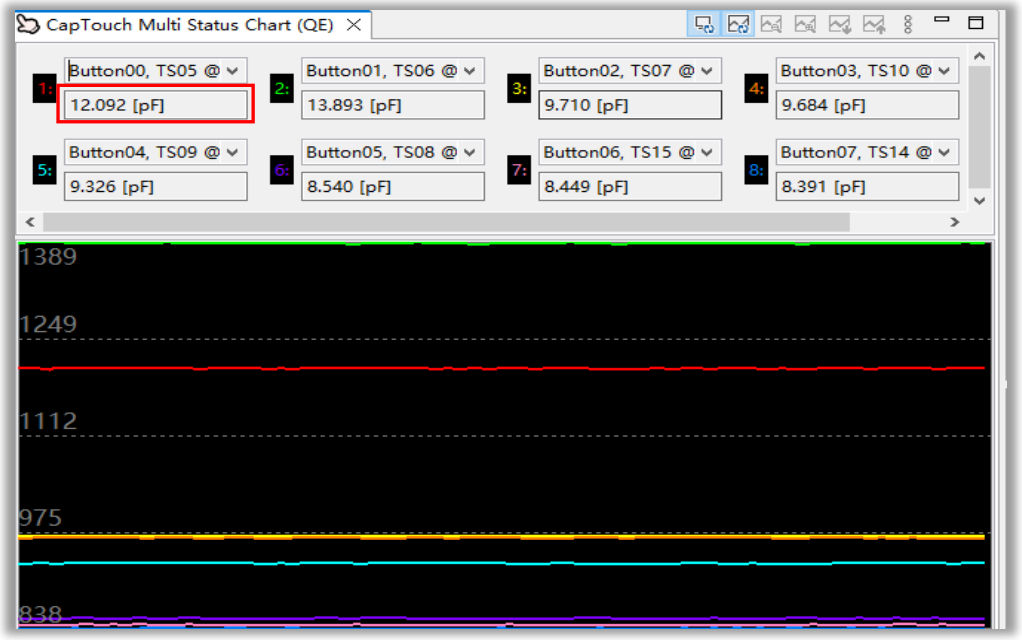
最多可以设定八个 **TS** 通道。

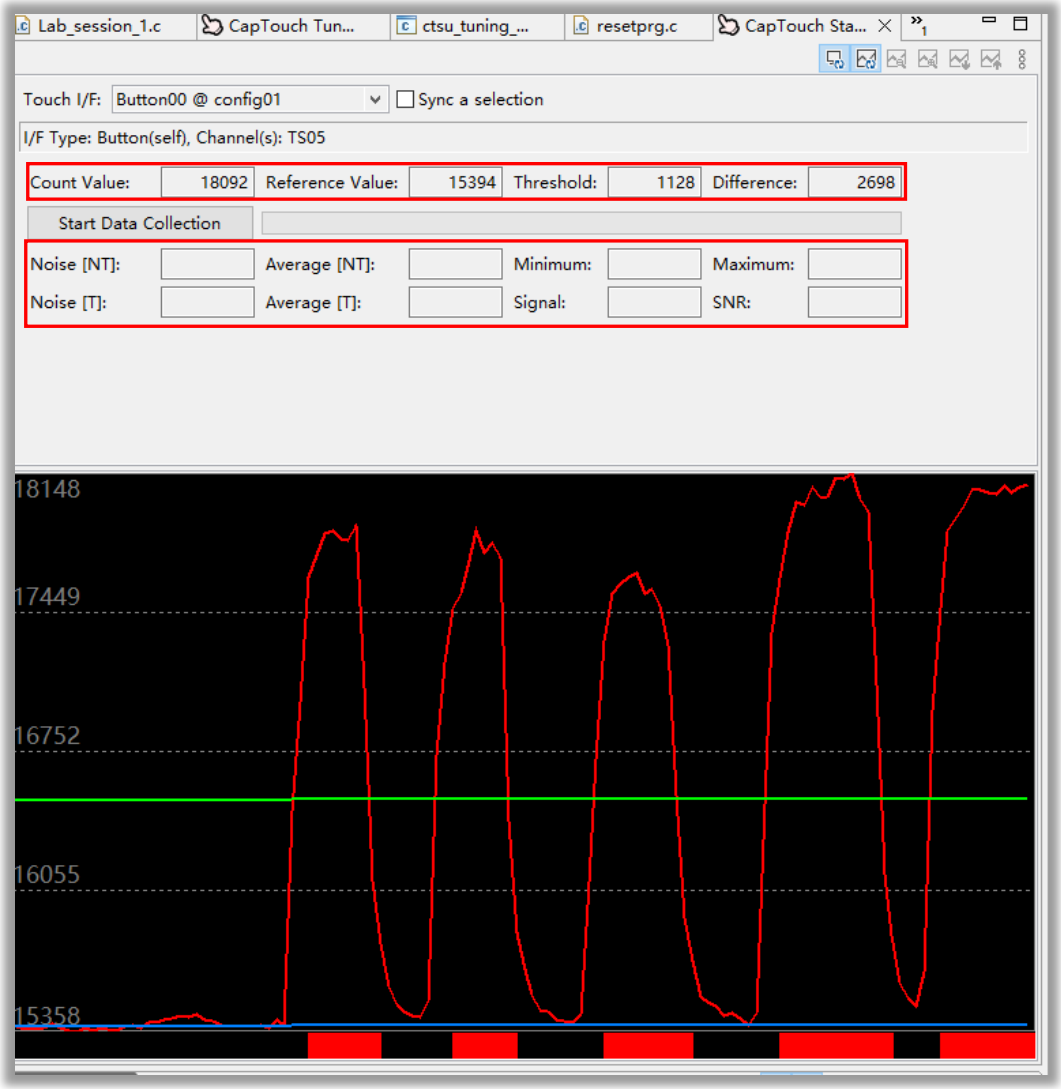


可以点击 按钮，选择"**Display Capacitance Converted from Count Value**"，将各个 TS 通道下方的测量值数据由计数值改为电容值。

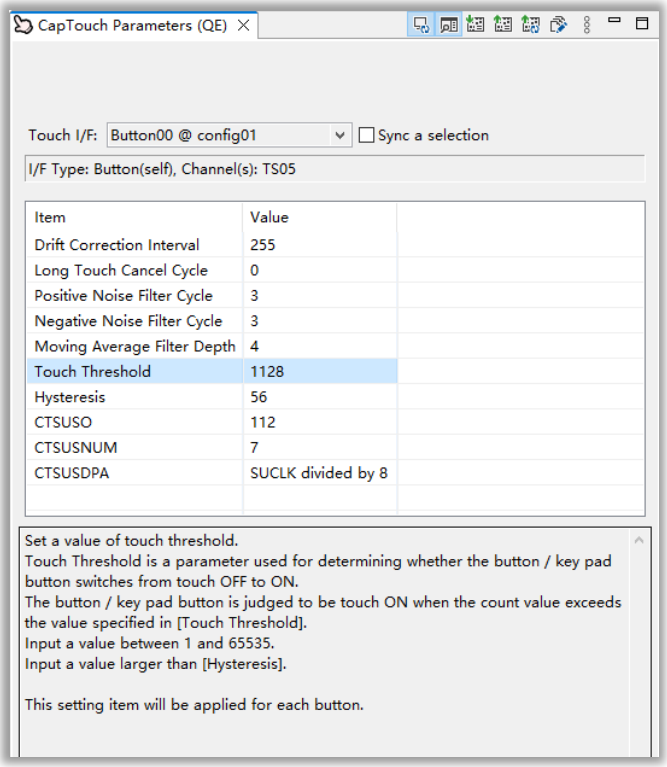


各个 **TS** 通道下方的测量值数据由计数值改为电容值，如下所示：



2.9.8	在"**CapTouch Status Chart (QE) View**"中, 可以实时显示单个 **TS** 通道的详细数据，例如： 实时测量值 **Count Value**，波形图中的**红色曲线**。 基准参考值 **Reference Value**, 也叫 **Baseline**，波形图中的**蓝色曲线**。为实时测量值 **Count Value** 的长期平均值。 阈值 **Threshold**，波形图中的**绿色曲线**。改变阈值，可调整触摸按键的灵敏度。 差分值 **Difference**, 为实时测量值 **Count Value** 与基准参考值 **Reference Value** 的差值。 另外，用户可以在"**CapTouch Status Chart (QE) View**"中，通过"**Start Data Collection**"按钮，测量 **SNR** 数据。 
NOTE	通过"**Start Data Collection**"按钮，测量 **SNR** 数据的方法，可参考如下各个 **MCU** 系列的文档: **RA FSP** : Using QE and FSP to Develop Capacitive Touch Applications **RX FIT** : Using QE and FIT to Develop Capacitive Touch Applications Rev.2.00 **RL78 SIS** : Using QE and SIS to Develop Capacitive Touch Applications Rev.2.10

2.9.9	在"**CapTouch Parameters (QE) View**"中, 可以显示和修改，触摸按键的运行参数。 相同分组(**configuration**)内的各个按键，共享相同的运行参数。 单击选中各个参数后，下方窗口会显示该参数的意义。 修改各个参数的数值后按下回车键，然后可以通过单击右上方的按钮，写入目标板，实时生效。 右上方各个功能按钮的功能，详见下面右侧图片。 点击 按钮后，可显示/隐藏高级运行参数 **CTSUSO、CTSUSNUM、CTSUSDPA**，通常不需要修改。

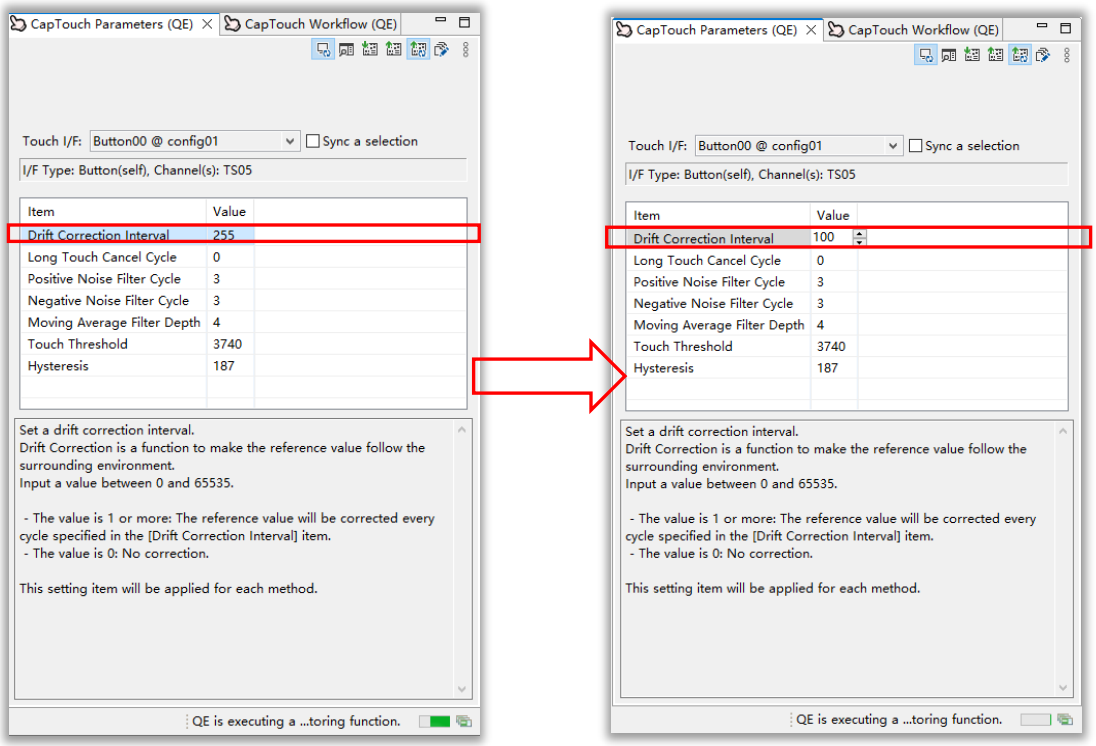


Icon	Tooltip
	Enable Monitoring
	Display in Advanced Mode
	Read Value from the Target Board
	Write Value to the Target Board
	Enable Auto Writing
	Output Parameter File
None (▽)	Check the Latest Information on Web

2.10	**调试触摸运行参数**
2.10.1	在"**CapTouch Parameters (QE) View**"中，可以对触摸运行参数进行调整，包括： **Drift Correction Interval** 漂移校正间隔 **Long Touch Cancel Cycle** 长按键取消周期 **Positive Noise Filter Cycle** 按键 On 判断的噪声滤波周期 **Negative Noise Filter Cycle** 按键 Off 判断的噪声滤波周期 **Moving Average Filter Depth** 移动平均滤波深度 **Touch Threshold** 触摸阈值 **Hysteresis** 迟滞
NOTE	在使用以上相关参数进行灵敏度调整时，只能进行微调。寄生电容值从根本上决定了灵敏度的高低。
2.10.2	在"**CapTouch Parameters (QE) View**"中，点击 按钮，使能自动写入参数功能。 该功能打开后，修改参数，按下回车键，新参数即可立即生效。

2.10.3

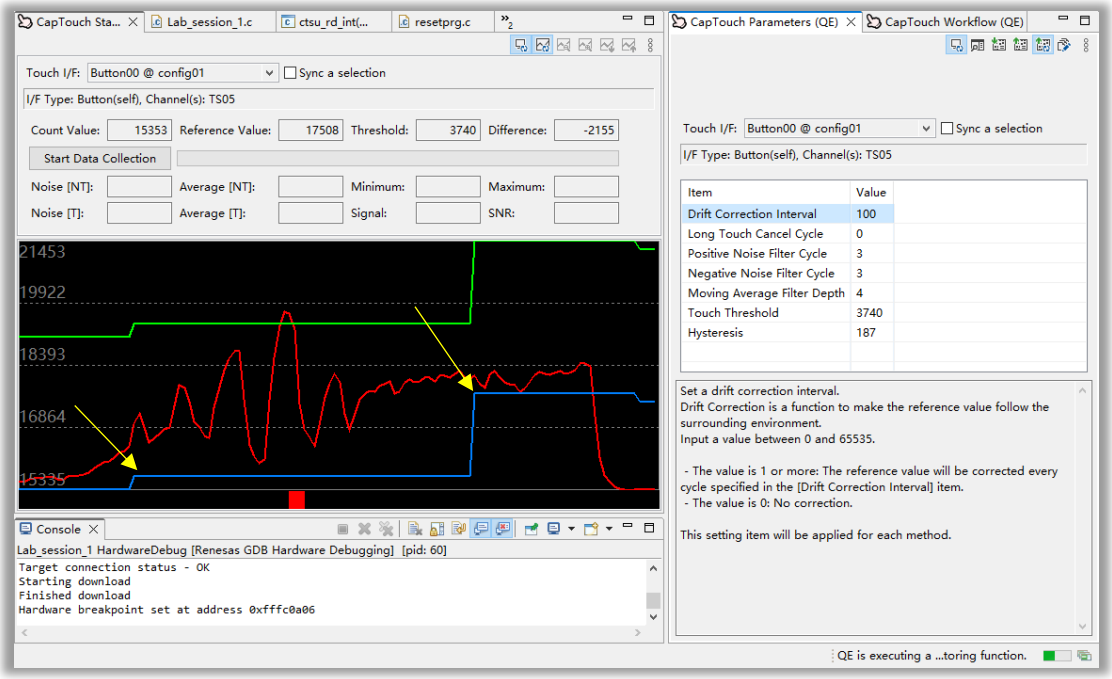
在"**CapTouch Parameters (QE) View**"中，将"**Drift Correction Interval**"从 **255** 修改为 **100**，回车。



"漂移校正间隔 **Drift Correction Interval**"用于应对环境、器件老化等电容环境变化非常缓慢的情况。

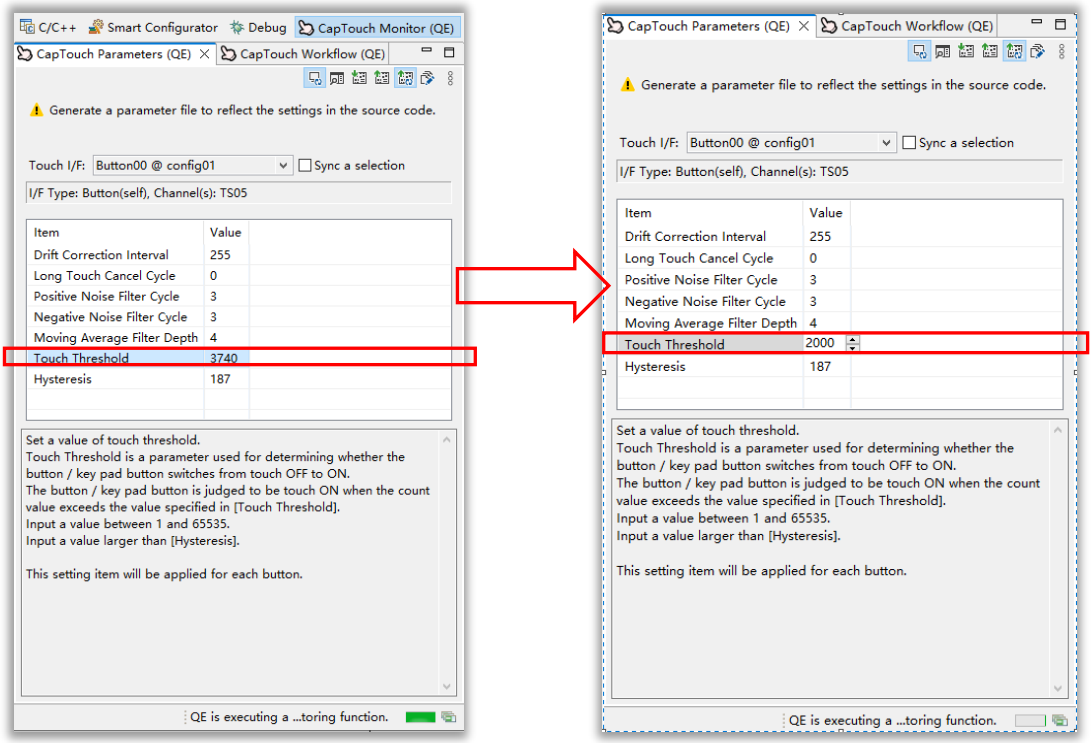
根据应用的实际需要，调整设定值。

在本实验例中，将"**Drift Correction Interval**"的设定值从默认 **255**，改为 **100** 后，代表基准参考值"**Reference Value**"的**蓝色曲线**，将以设定值为 **100** 的时间间隔进行更新，此时加快了对环境变化的响应速度。



2.10.4

在"**CapTouch Parameters (QE) View**"中，将"**Touch Threshold**"从 **3740** 修改为 **2000**，回车。

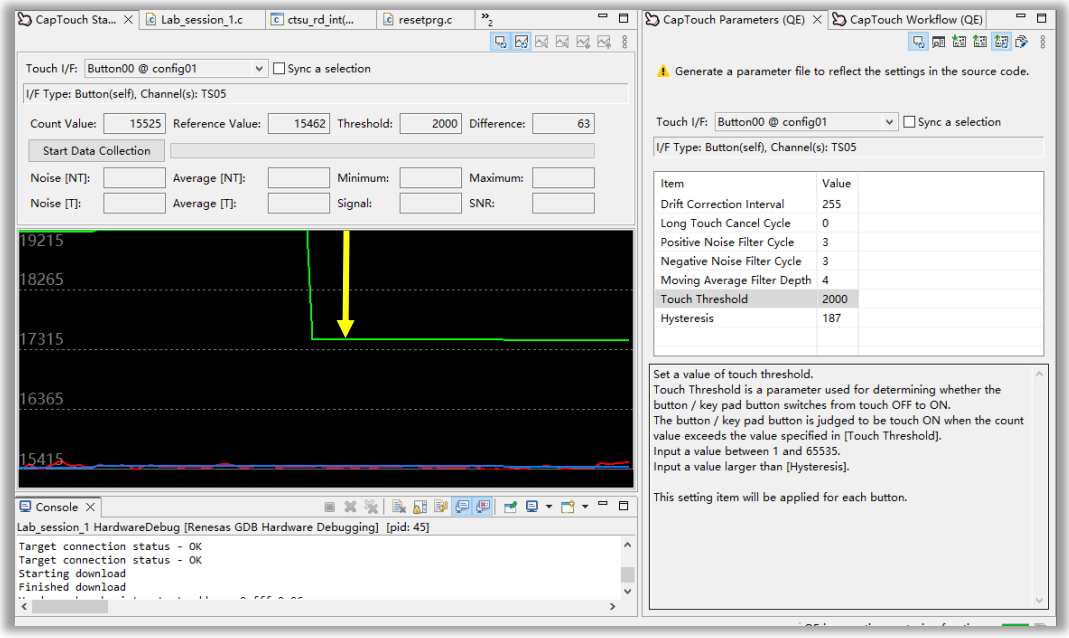


代表"阈值 **Threshold**"的**绿色曲线**立即发生变化，从 **3740** 修改为 **2000**。

原来按下按键后，测量值需要超过 **3740**，才能判定为按键按下，

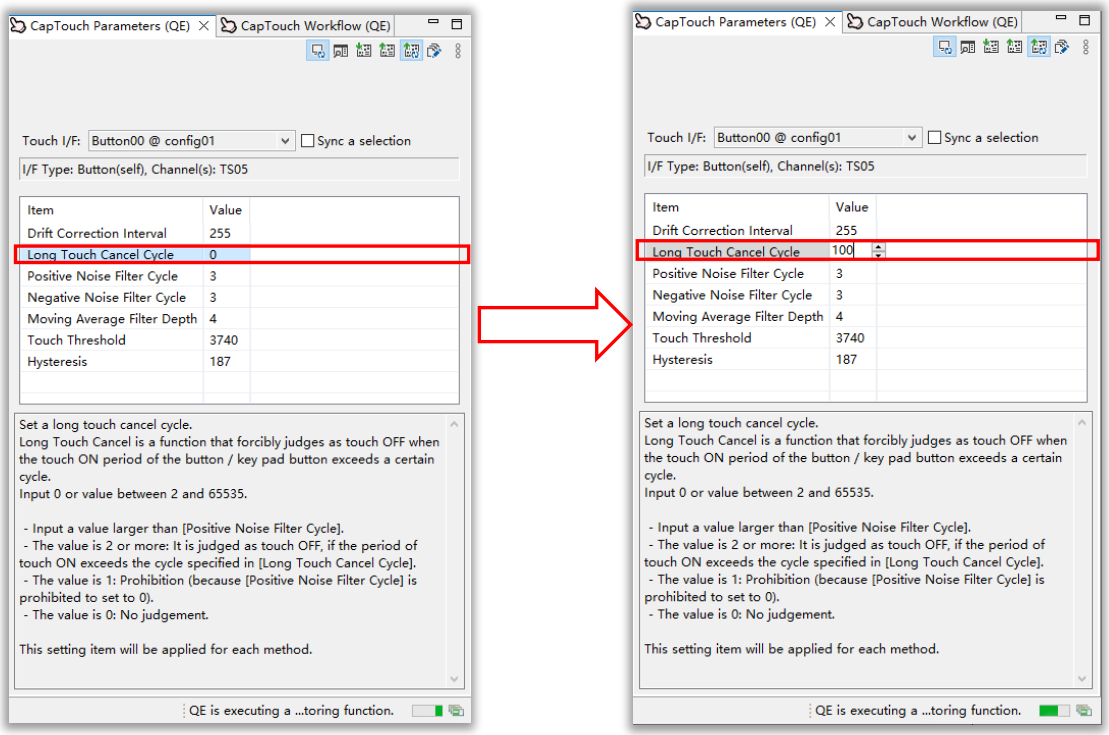
现在只需要超过 **2000**，即可判定为按键按下。

那么在按压力度不变的情况下，通过改变"阈值 **Threshold**"，由 **3740** 缩小 **2000**，从而提高了按键灵敏度。

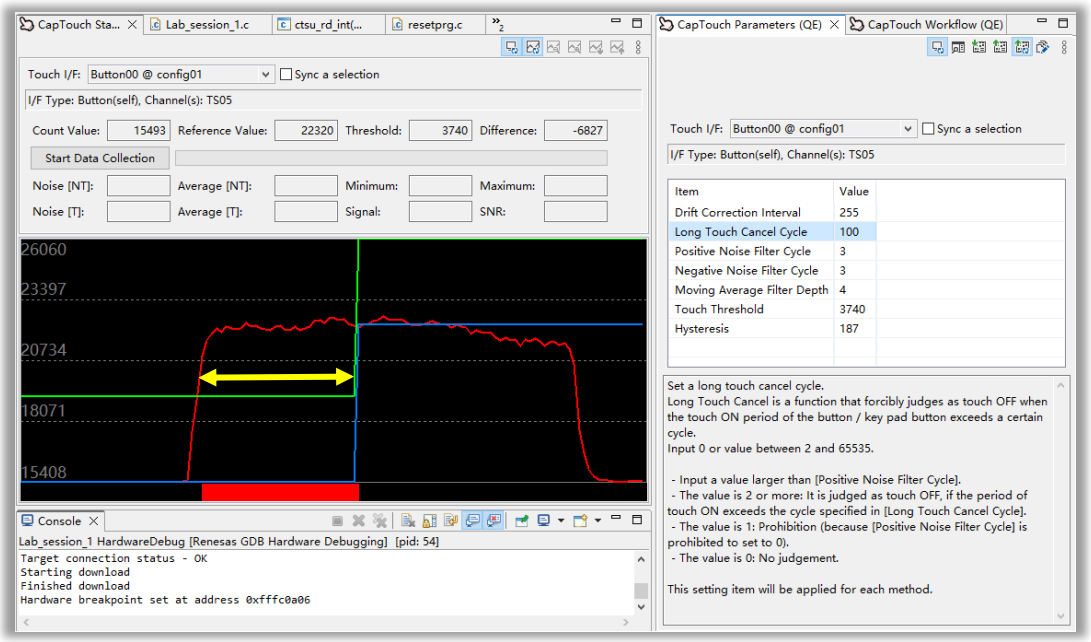


2.10.5

在"**CapTouch Parameters (QE) View**"中，将"**Long Touch Cancel Cycle**"从 **0** 修改为 **100**，回车。

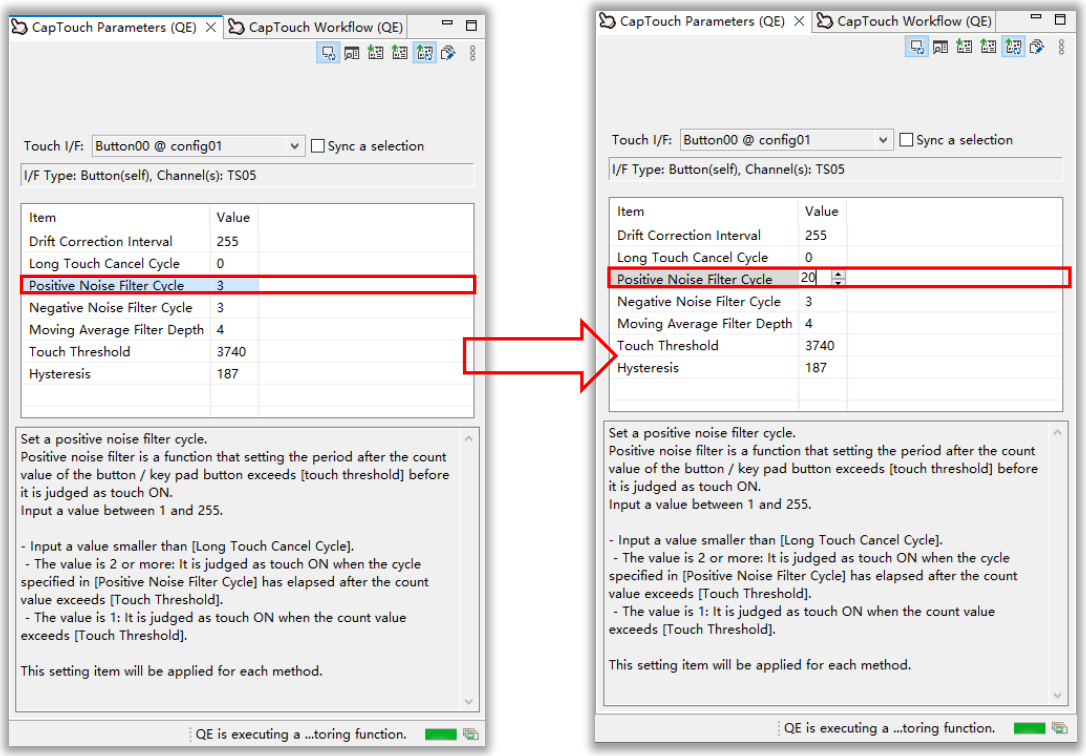


在按键一直按下，时间超过设定值为 **100** 的时间后，按键状态强制由 **On** 改为 **Off**，如下图所示：

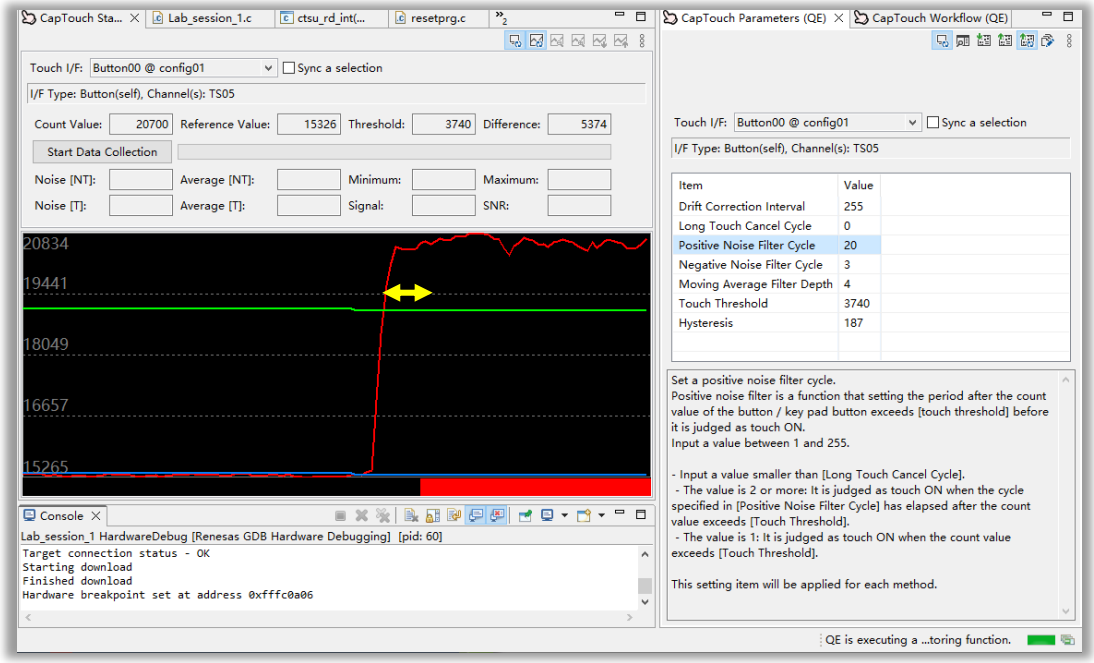


2.10.6

在"**CapTouch Parameters (QE) View**"中，将"**Positive Noise Filter Cycle**"从 **3** 修改为 **20**，回车。

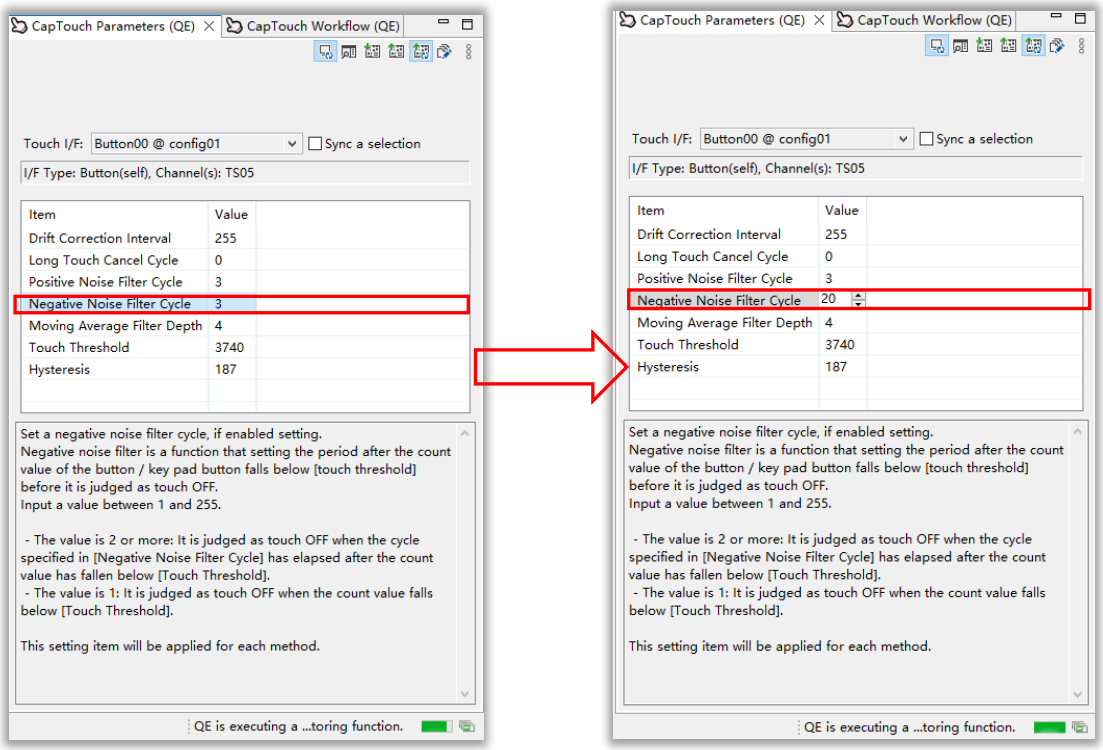


在按键按下，测量值超过门槛 **20+1** 个周期后，才进行按键状态 **On** 的判断，如下图所示：

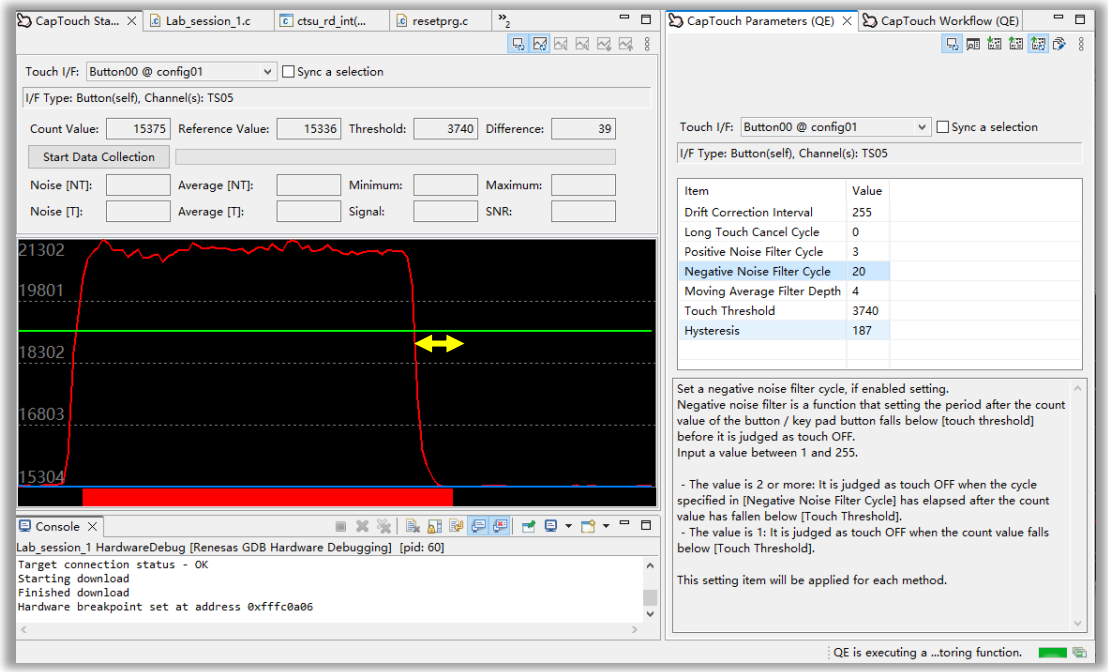


2.10.7

在"**CapTouch Parameters (QE) View**"中，将"**Negative Noise Filter Cycle**"从 **3** 修改为 **20**，回车。

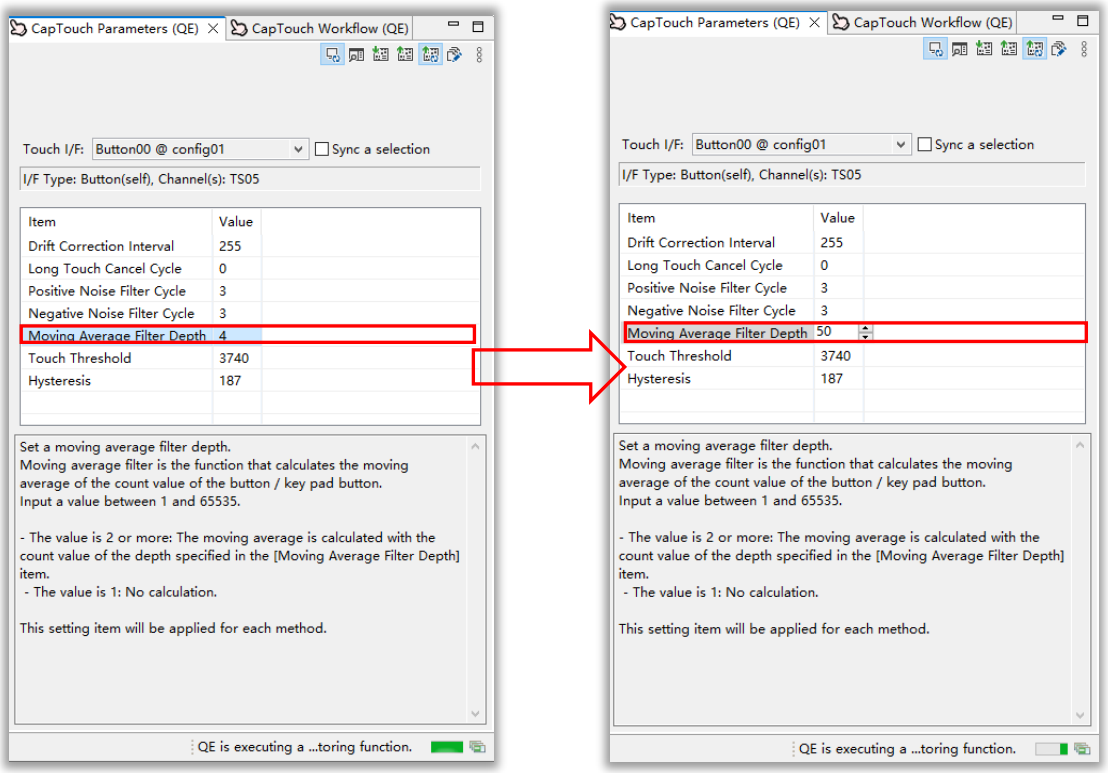


在按键抬起，测量值跌落门槛 **20+1** 个周期后，才进行按键状态 **Off** 的判断，如下图所示：



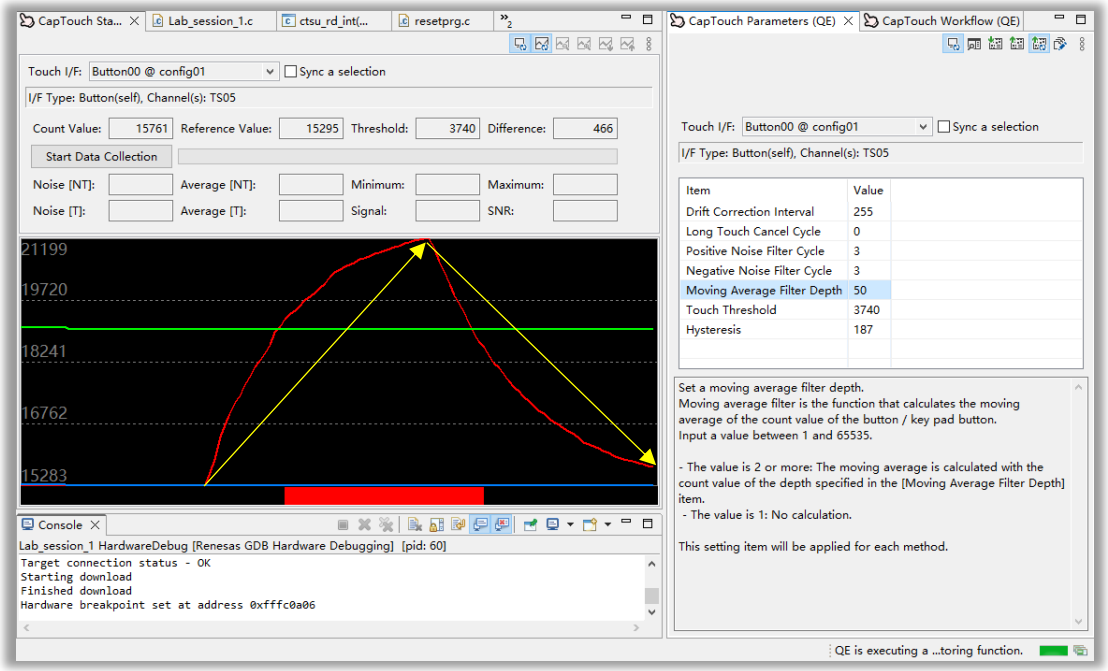
2.10.8

在"**CapTouch Parameters (QE) View**"中，将"**Moving Average Filter Depth**"从 **4** 修改为 **50** 回车。



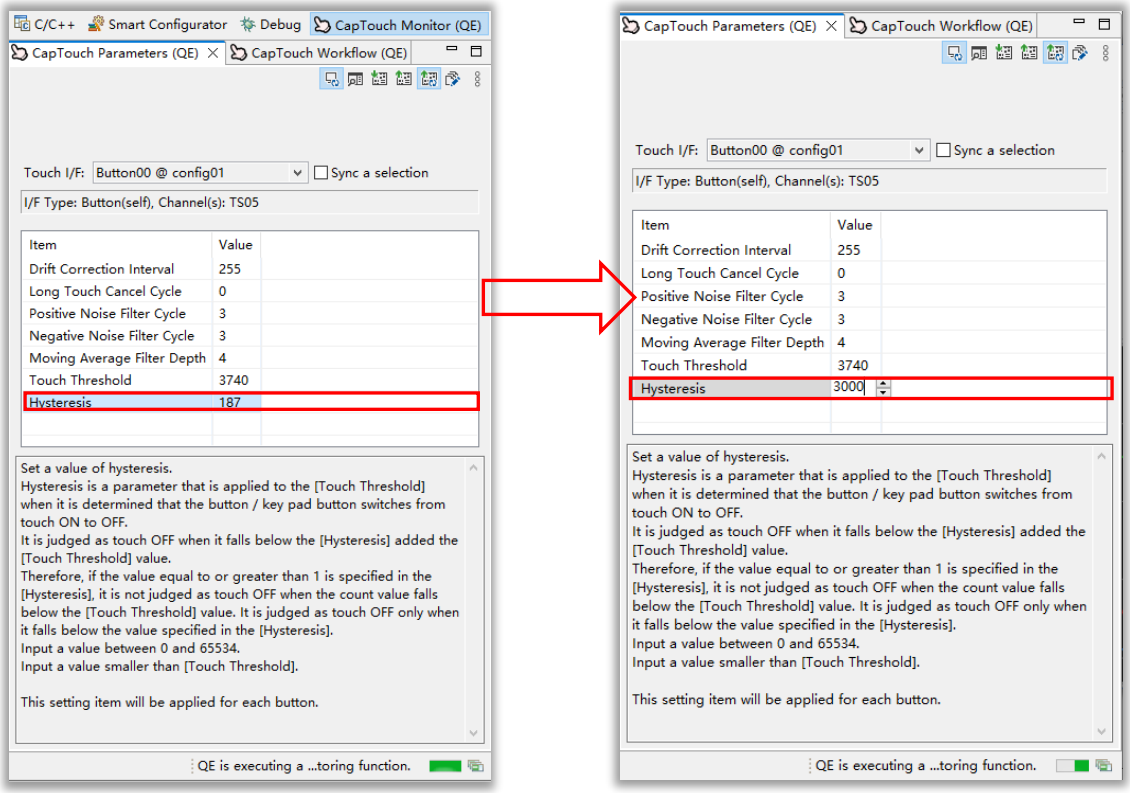
在按键按下和抬起时，当"**Moving Average Filter Depth**"的设定值较大时，移动平均滤波效果越强，测量值曲线的斜率越缓慢。

在按键按下和抬起时，当"**Moving Average Filter Depth**"的设定值较小时，移动平均滤波效果越弱，测量值曲线的斜率越陡峭。

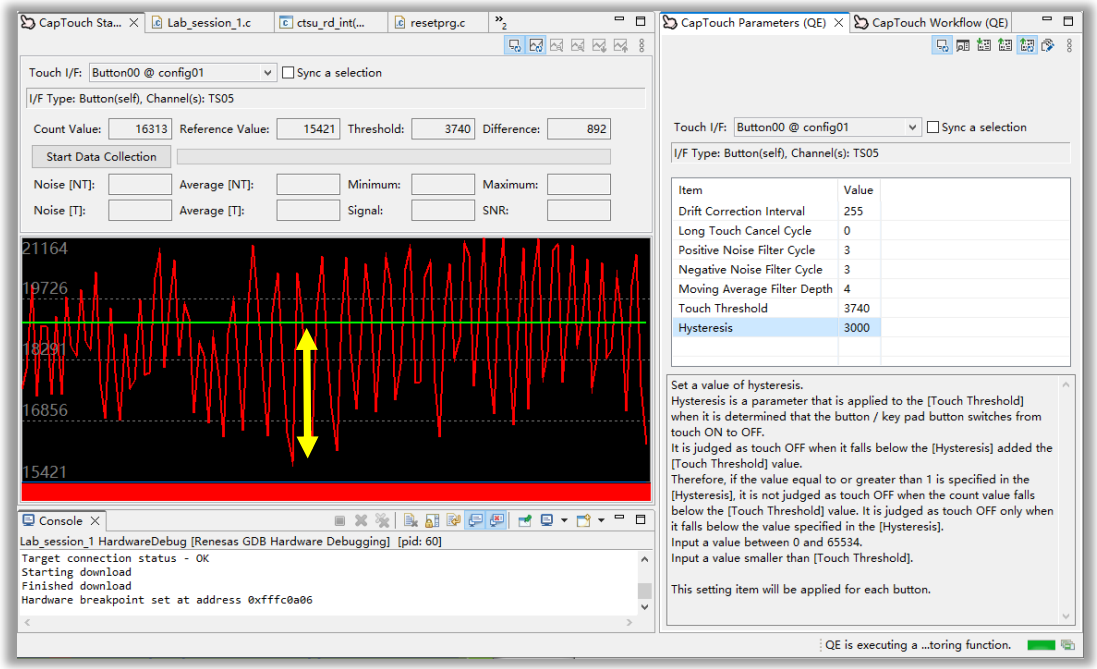


2.10.9

在"**CapTouch Parameters (QE) View**"中，将"**Hysteresis**"从 **187** 修改为 **3000**，回车。



当测量值在代表"阈值 **Threshold**"的**绿色曲线**下方， 在设定值为 **3000** 的区间内波动时， 即便超过或者跌落"阈值 **Threshold**"，按键状态也不改变。



END OF SECTION

3 Lab Session 2： 在 Lab 1 的基础上增加 MEC 功能

概述

在本实验环节中，将在 **Lab session 1** 的基础上，增加 **MEC** 多电极连接功能，**12** 个按键电极将在内部连接在一起，作为一个 **MEC** 电极工作，此时不识别 **12** 个按键电极中的哪个按键电极被按下。

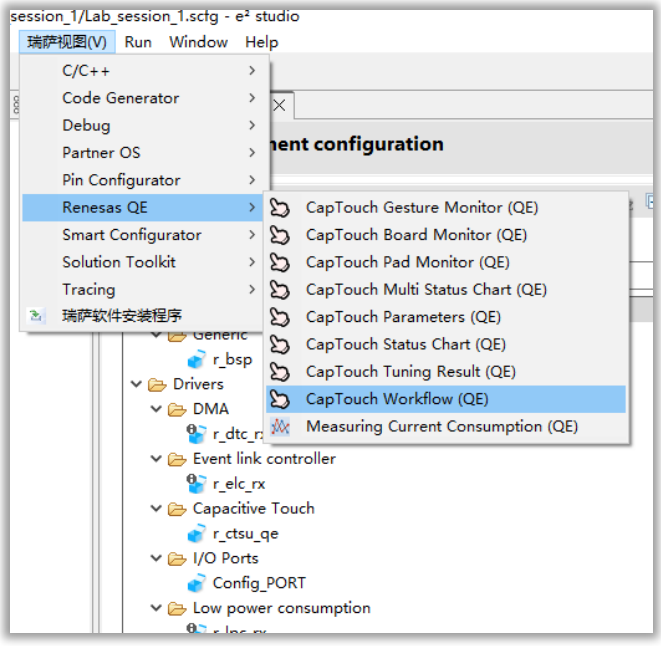
- **3.1** 修改触摸接口(**interface**)或者配置(**Configuration**)
- **3.2** 自动调整过程 (**Auto Tuning Process**)
- **3.3** 使用 **QE for Cap Touch** 监控 **MEC** 电极的触摸底层数据以及触摸行为
- **3.4** 调试 **MEC** 电极的运行参数

如果对 **Lab session 2** 的内容非常熟悉或者有一定困难，可跳过步骤 **3.1** 到步骤 **3.2**，

在 **e2 studio** 中 **import** 导入培训配套资料 **Checkpoints** 文件夹中的工程 **Lab session 2**，

直接进行步骤 **3.3** 到步骤 **3.4** 的实验。

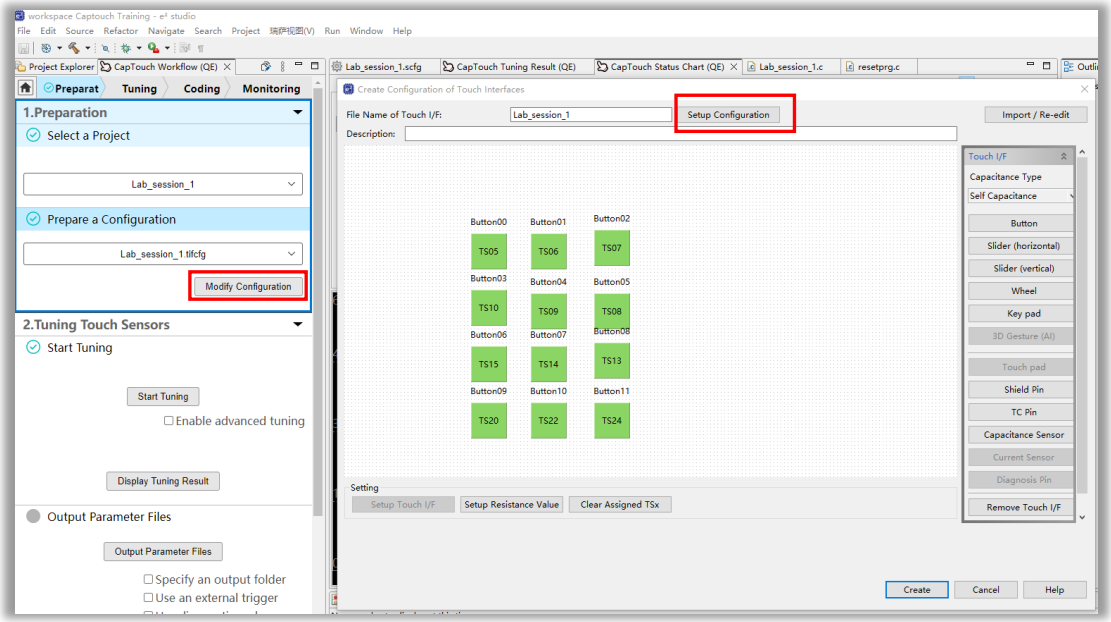
实验步骤

3.1	**修改触摸接口(interface)或者配置(Configuration)**
3.1.1	在"**Lab session 1**" 的 **e2 studio** 工程中， 选择"**Renesas View** 瑞萨视图" → **Renesas QE** → **CapTouch workflow** 

3.1.2

在"**CapTouch workflow**"中，在"**1.preparation**"页面中点击"**Modify Configuration**"，

弹出"**Create Configuration of Touch Interfaces**"页面，如下图所示，点击"**Setup Configuration**"

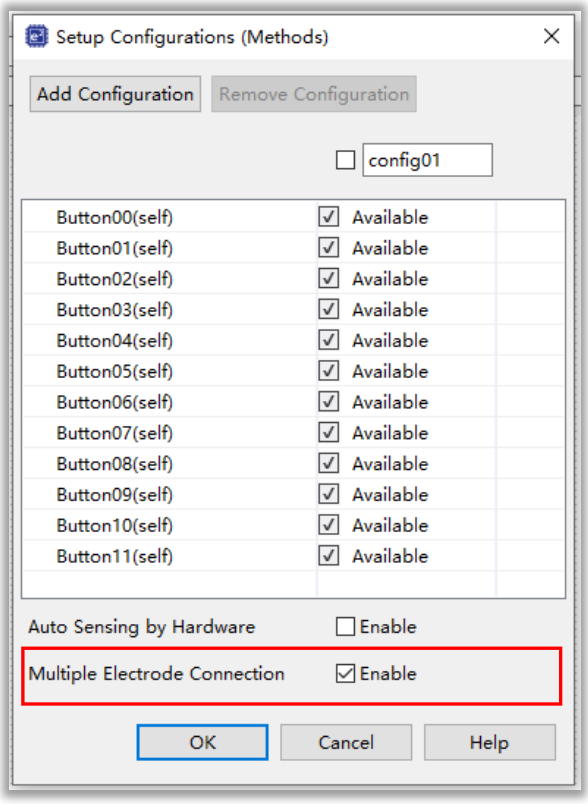


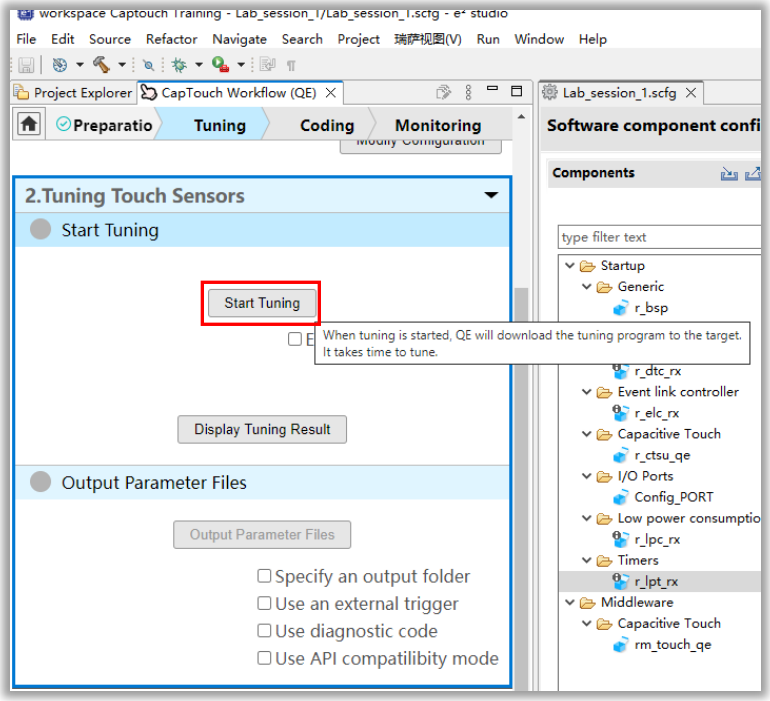
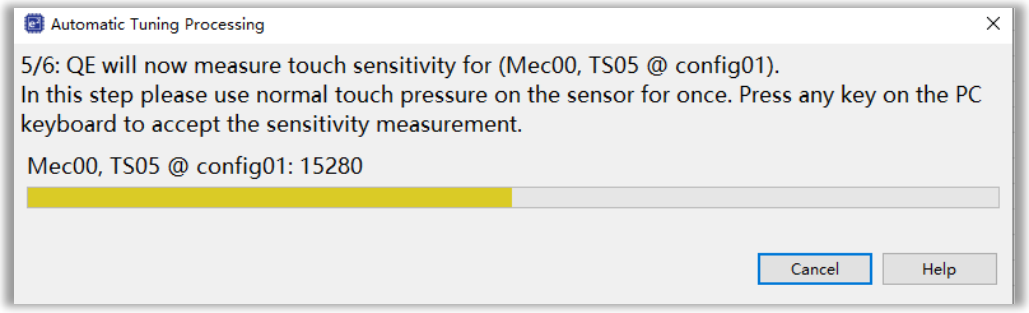
可以通过"**Add Configuration**",以及勾选 **Config01** 下方的 **Available**，为 **Button** 分组 (**Configuration**)，下图中，**Button00** 到 **Button11** 的 **12** 个 **Button** 都在 **Config01** 组中。

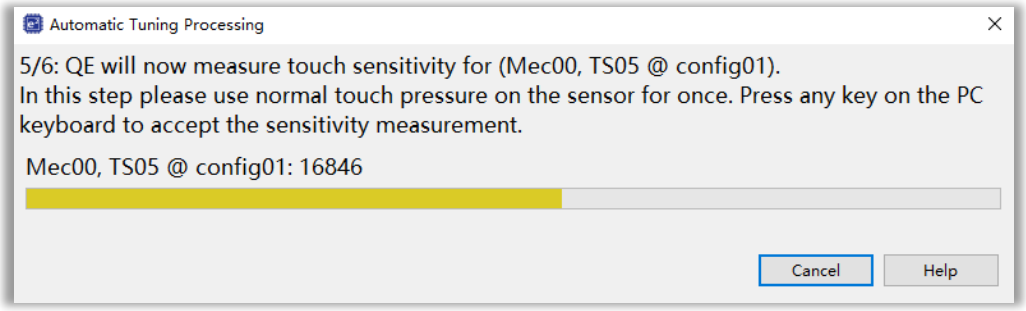
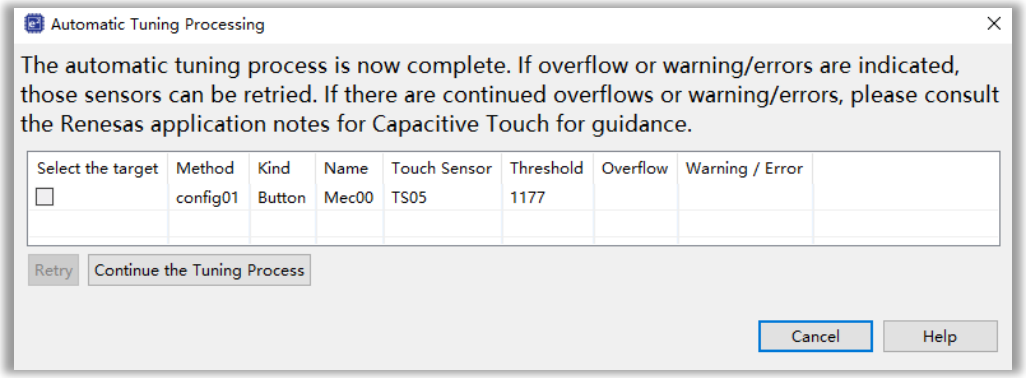
勾选 **Config01** 下方的"**Multiple Electrode Connection**"右侧的 **Enable**，将 **config01** 配置为 **MEC** 电极。

单击 **OK**，关闭"**Setup Configuration**"对话框，回到"**Create Configuration of Touch Interfaces**"页面.

单击 **Create**，在对话框中选择 **Yes** 覆盖之前的设定，完成触摸接口(**interface**)或者配置(**Configuration**)的设定。

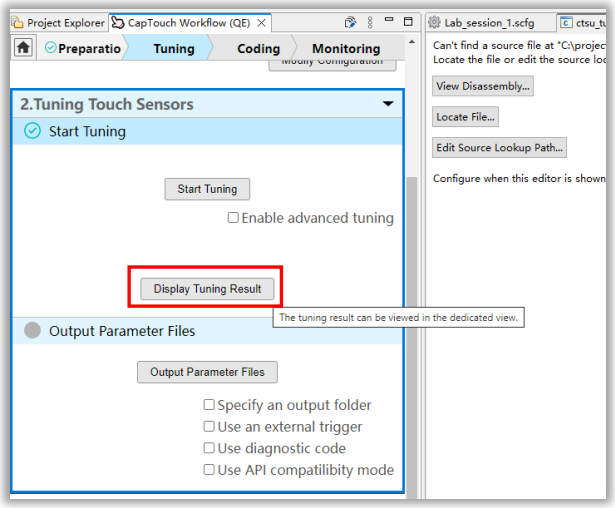


3.2	自动调整过程(**Auto Tuning Process**)
3.2.1	开始自动调整过程(**Auto Tuning Process**) 在"**Cap Touch Workflow**"的"**2.Tuning Touch Sensors**"中，单击"**Start Tuning**" 
3.2.2	自动调整过程(**Auto Tuning Process**)开始，依次显示如下四步，这时不需要用户操作。 第一步：开始自动调整过程，引导用户按提示操作，按照要求"触摸按键"或者"不要触摸按键"。 第二步：**QE** 正在测量所有触摸按键的寄生电容。 第三步：**QE** 正在调整触摸按键的偏置电流值 第四步：**QE** 开始进行灵敏度测量
NOTE	以上自动调整过程(**Auto Tuning Process**)开始时的四个步骤的图片可参考
	第五步：灵敏度测量 自动调整过程(**Auto Tuning Process**)完成前四步准备工作后，开始第五步。 如下图所示，仅有一个 **MEC** 电极，"**Mec00，TS05**"需要进行灵敏度测量。 在没有按下触摸按键时，自容式按键的灵敏度测量的基准值为 **15360** 左右。 

3.2.3	按照提示，使用手指以正常压力按住 **12** 个按键中的任意一个按键， 此时黄色进度条将根据手指按压触摸按键的力度而变化， 保持期望的按压力度，同时按下 **PC** 键盘的任意键，接受该触摸按键的灵敏度测量。 
NOTE	通过 **MEC** 多电极连接功能，**12** 个按键电极已经在内部连接在一起，作为一个 **MEC** 电极工作，此时无论哪个按键被按下，都可以进行 **MEC** 电极的灵敏度测量。
3.2.4	完成自动调整过程(**Auto Tuning Process**)后，自动弹出结果，显示了 **MEC** 电极的阈值 **Threshold**。 点击"**Continue the Tuning Process**"，自动调整过程的结果对话框关闭。 自动调整过程(**Auto Tuning Process**) 完成。 

3.2.5

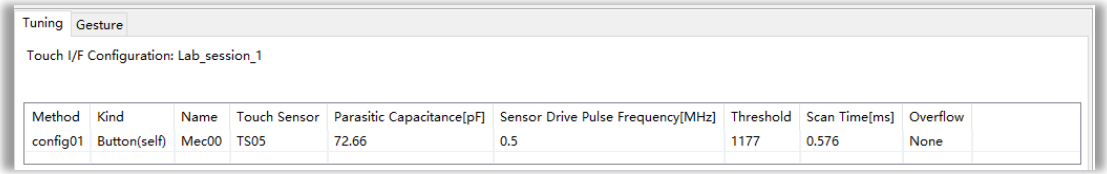
在"**Cap Touch Workflow**"的"**2.Tuning Touch Sensors**"中，点击"**Display Tuning Result**"



自动调整过程(**Auto Tuning Process**)的结果，如下图所示:

包括 **Method**，**Kind**，**Name**，**Touch Sensor**，**Parasitic Capacitance**，**Sensor Driver Pulse Frequency**，**Threshold**，**Scan Time**，以及 **Overflow** 等重要信息。

(受环境影响，重新进行自动调整过程时，寄生电容值会有细微差异，传感器驱动脉冲频率也有可能因寄生电容值的变化发生变化；阈值 **Threshold** 也会因按压力度的变化发生变化，阈值也可以在配置文件中直接修改)



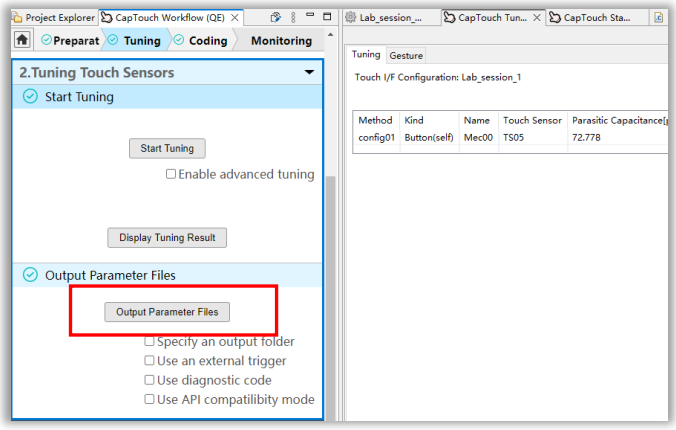
NOTE

这里要特别注意 **MEC00** 的寄生电容值 **72.66pF**，由于超过了 **50pF**，因此只能使用 **0.5MHz** 的传感器驱动脉冲频率，因此阈值只有 **1177**，灵敏度大幅度降低。

3.2.6

输出参数文件

在"**Cap Touch Workflow**"的"**2.Tuning Touch Sensors**"中，点击"**Output Parameter FIles**"

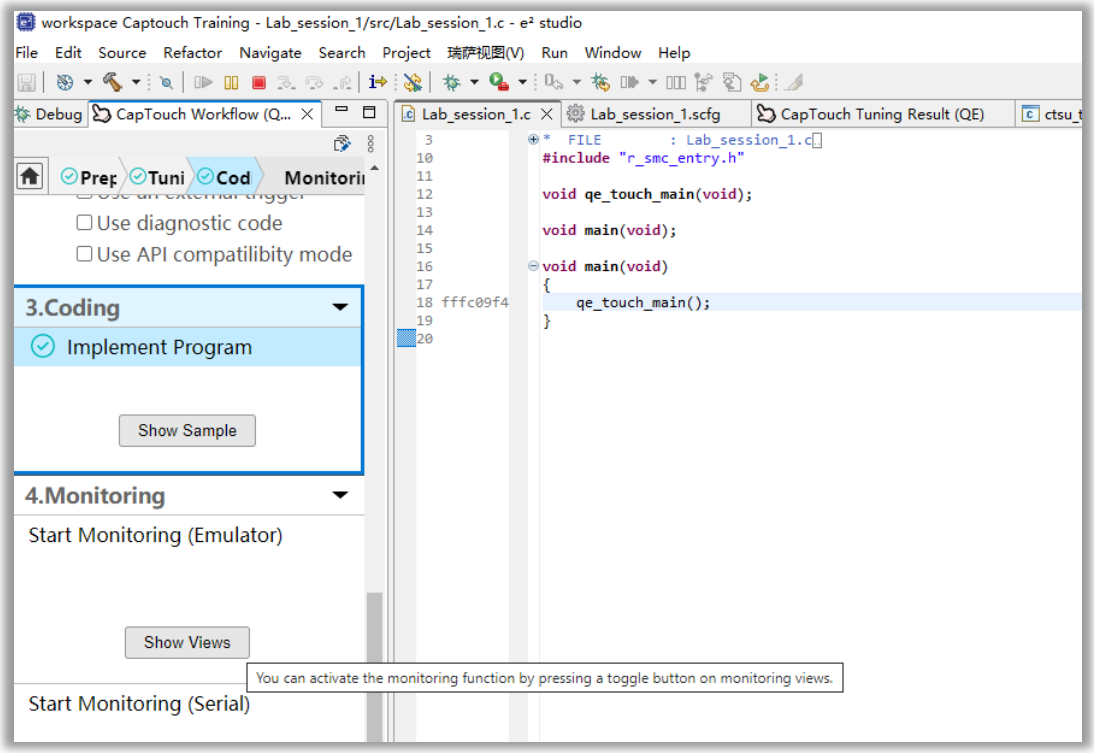
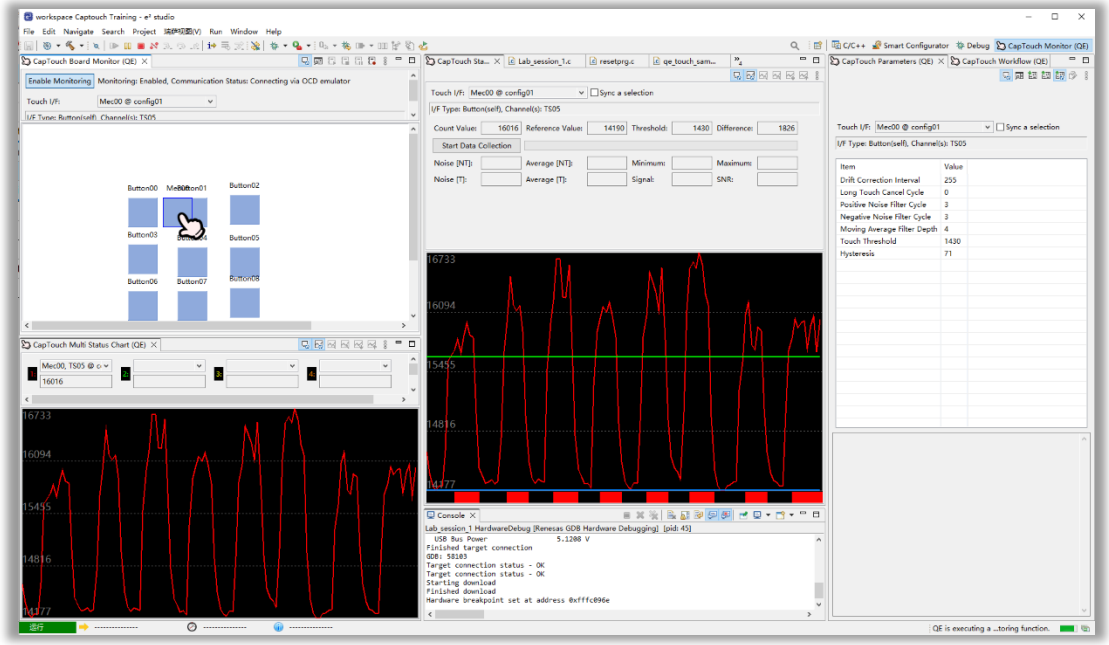


以下三个参数文件将被覆盖

Qe_touch_define.h

Qe_touch_config.h

QE_touch_config.c

3.2.7	点击  图标，编译程序
3.3	**使用 QE for Cap Touch 监控 MEC 电极的触摸底层数据以及触摸行为**
3.3.1	按照"**2.6 运行程序**"小节介绍的方法，在仿真状态下全速运行程序。 在"**Cap Touch Workflow"的"4.monitoring**"中，点击"**Start Monitoring(Emulator)**"下方的"**Show Views**" 
3.3.2	具体操作可按照"**2.9 使用 QE for Cap Touch 监控触摸底层数据以及触摸行为**"小节介绍的方法进行。 

3.4	**调试 MEC 电极的运行参数**
3.4.1	**MEC** 电极的运行参数，以及调试 **MEC** 电极的运行参数方法与 **2.10** 小节介绍的完全相同 包括： **Drift Correction Interval** 漂移校正间隔 **Long Touch Cancel Cycle** 长按键取消周期 **Positive Noise Filter Cycle** 按键 On 判断的噪声滤波周期 **Negative Noise Filter Cycle** 按键 Off 判断的噪声滤波周期 **Moving Average Filter Depth** 移动平均滤波深度 **Touch Threshold** 触摸阈值 **Hysteresis** 迟滞

END OF SECTION

4 Lab Session 3： 在 Lab 2 的基础上通过改变 MEC 电极的灵敏度增加接近传感功能

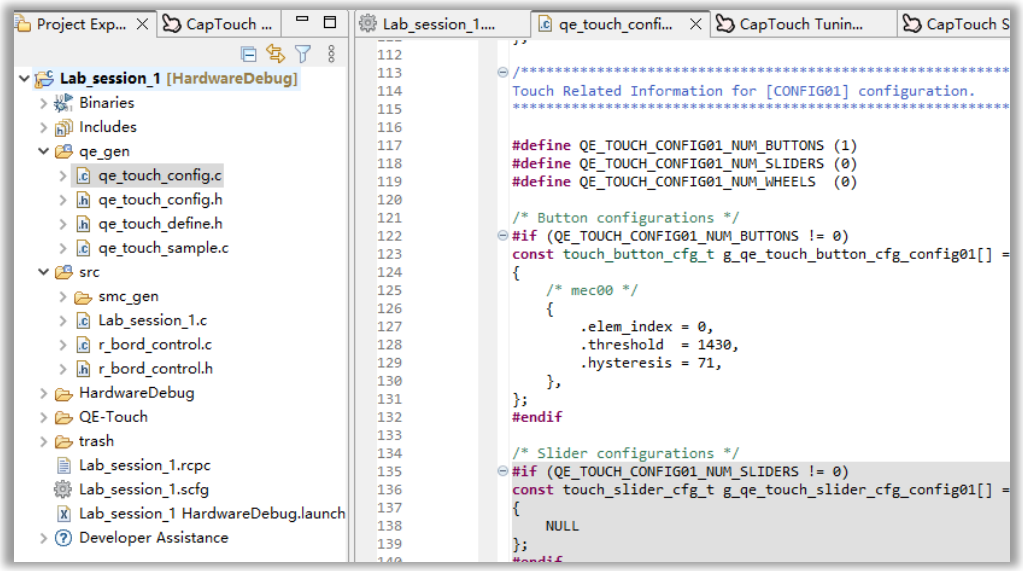
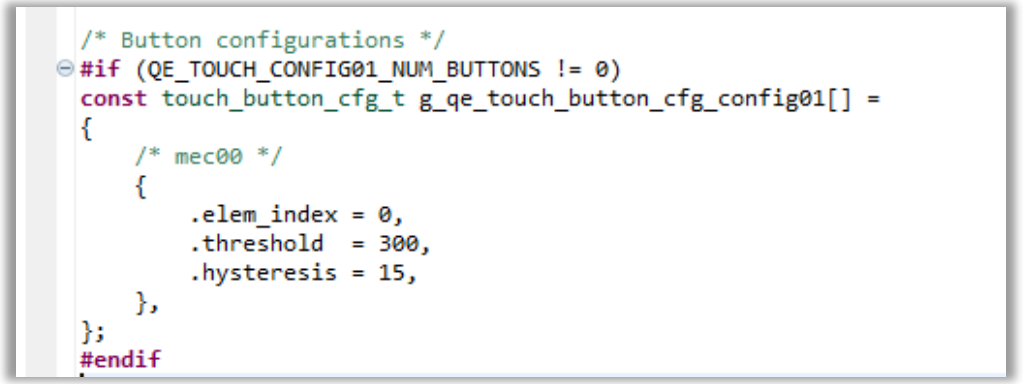
概述

在本实验环节中，将在 **Lab session 2** 的基础上，通过调整 MEC 电极的运行参数，提高灵敏度，增加接近传感功能。

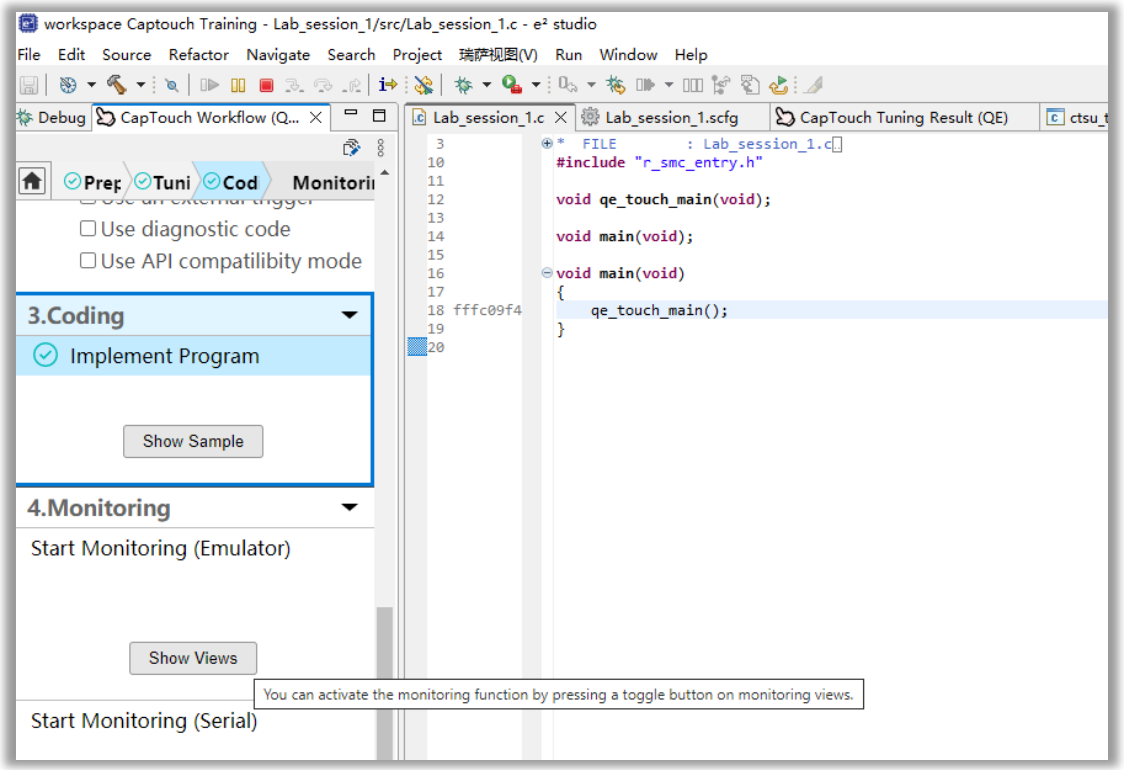
- **4.1** 修改 **MEC** 电极的阈值
- **4.2** 使用 **QE for Cap Touch** 监控 **MEC** 电极的触摸底层数据以及触摸行为
- **4.3** 调试 **MEC** 电极的运行参数

如果对 **Lab session 3** 的内容非常熟悉或者有一定困难，可跳过步骤 **4.1** 到步骤 **4.2**，
在 **e2 studio** 中 **import** 导入培训配套资料 **Checkpoints** 文件夹中的工程 **Lab session 3**，
直接进行步骤 **4.2** 到 **4.3** 的实验。

实验步骤

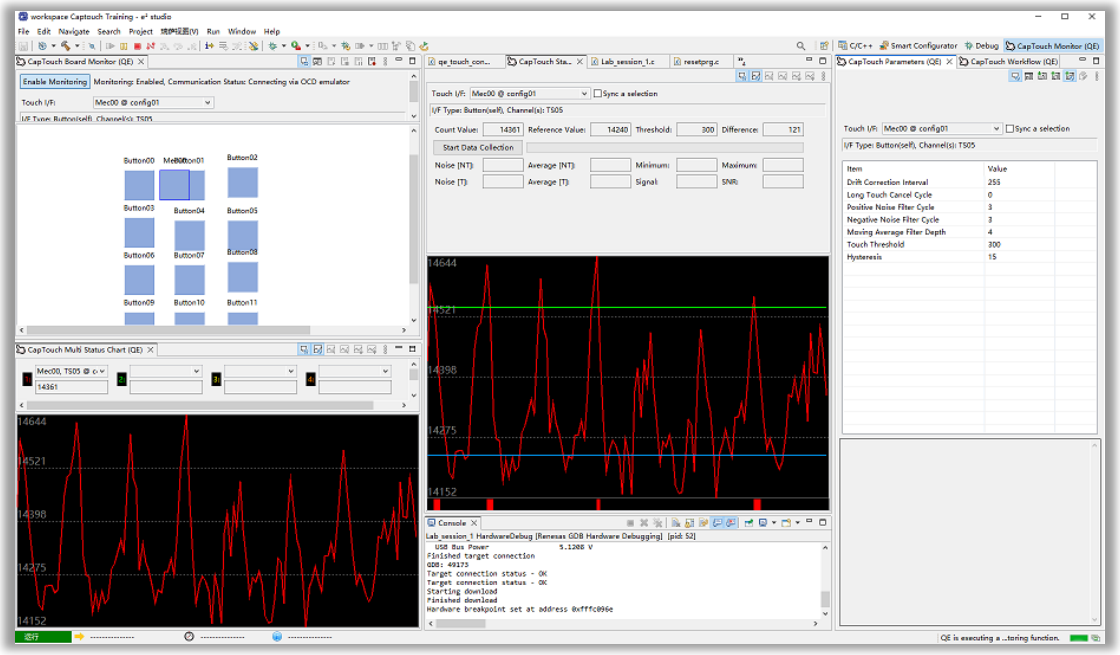
4.1	修改 **MEC** 电极的阈值
4.1.1	在"**Lab session 2**"的基础上，在"**Project Explorer**"的"**Lab_session_1**"工程中，双击打开"**qe_touch_config.c**"文件。  在 **128** 行，找到 **MEC00** 电极的阈值设定，先将阈值"**.threshold**"改为 **300**，" **.hysteresis**"改为 **15** 

```
/* Button configurations */
#if (QE_TOUCH_CONFIG01_NUM_BUTTONS != 0)
const touch_button_cfg_t g_qe_touch_button_cfg_config01[] =
{
    /* mec00 */
    {
        .elem_index = 0,
        .threshold  = 300,
        .hysteresis = 15,
    },
};
#endif
```

4.1.2	点击  图标，编译程序
4.2	**使用 QE for Cap Touch 监控 MEC 电极的触摸底层数据以及触摸行为**
4.2.1	按照"**2.6 运行程序**"小节介绍的方法，在仿真状态下全速运行程序。 在"**Cap Touch Workflow**"的"**4.monitoring**"中，点击"**Start Monitoring(Emulator)**"下方的"**Show Views**" 

4.2.2

具体操作可按照"**2.9 使用 QE for Cap Touch 监控触摸底层数据以及触摸行为**"小节介绍的方法进行。

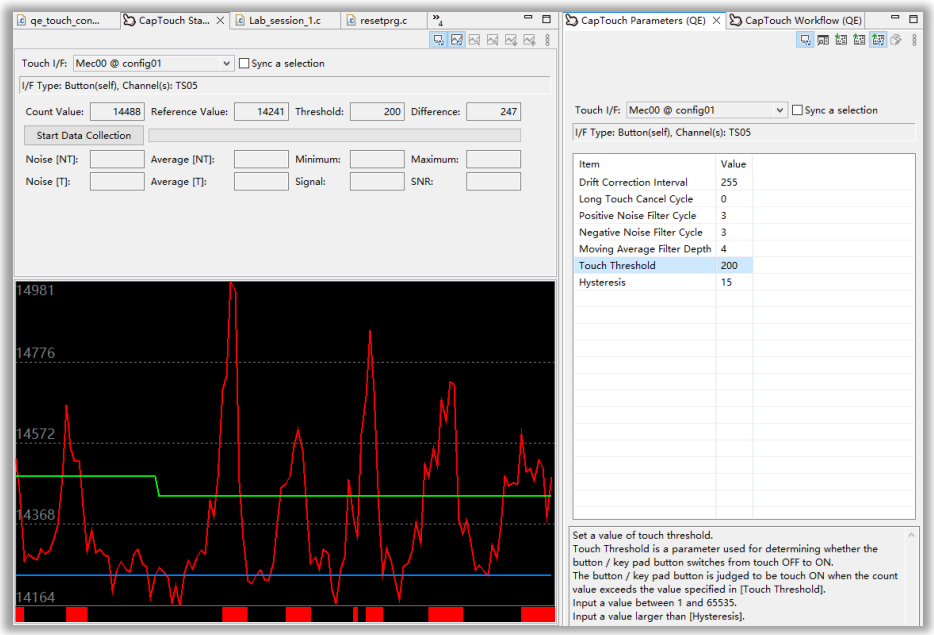


4.3 调试 MEC 电极的运行参数

4.3.1

使用手指或者手掌靠近 **MEC** 电极，在期望的接近传感距离停住，观测实时测量值的波形，设定合理的 **MEC** 电极的阈值 **Threshold**，反复调试，知道满意为止。

例如，将 **MEC** 电极的阈值 **Threshold** 设定为 **200**，可获得比较理想的使用手掌方式接近的 **1.5cm** 左右的接近传感距离.



END OF SECTION

5 Lab Session 4： 在 Lab 3 的基础上增加低功耗(Auto Judgement)功能

概述

在本实验环节中，将在 **Lab session 3** 的基础上，增加低功耗功能(**Auto Judgement**)功能.

修改触摸接口(**interface**)或者配置(**Configuration**)，将按键分组(**Configuration**)，将用于接近传感功能的 **MEC** 电极设定为 **Config1**，将 **12** 个按键设定为 **Config2**。

上电复位后，系统进入低功耗工作模式，此时接近传感电极工作，以 **100ms** 的控制周期进行测量。

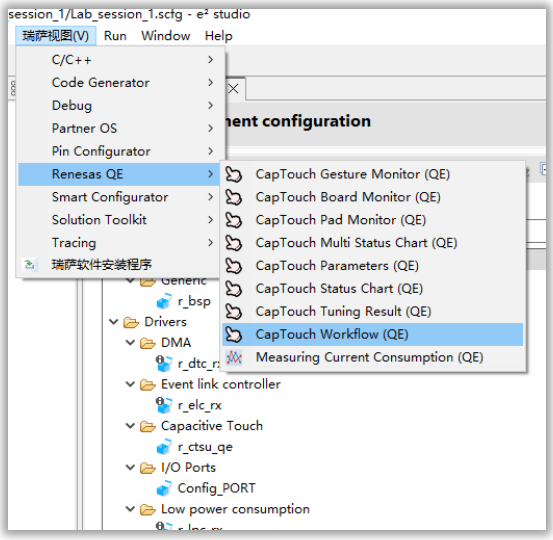
当用于接近传感功能的 **MEC** 电极，通过 **AJ** 自动判断功能，判断为没有触发时，系统始终保持在低功耗模式下工作。

当用于接近传感功能的 **MEC** 电极，通过 **AJ** 自动判断功能，判断为触发时，退出低功耗模式，系统在 **Normal** 模式下对 **Config1** 的 **MEC** 电极进行 **baseline** 调整, 然后对 **Config2** 的 **12** 个按键进行测量和判断。

- **5.1** 修改触摸接口(**interface**) 或者配置 (**Configuration**)
- **5.2** 使用 **Smart configurator** 添加必要的驱动程序
- **5.3** 自动调整过程 (**Auto Tuning Process**)
- **5.4** 增加低功耗 (**Auto Judgement**) 功能应用程序
- **5.5** 使用 **QE for Cap Touch** 监控触摸底层数据以及触摸行为
- **5.6** 调试低功耗 (**Auto Judgement**) 功能运行参数

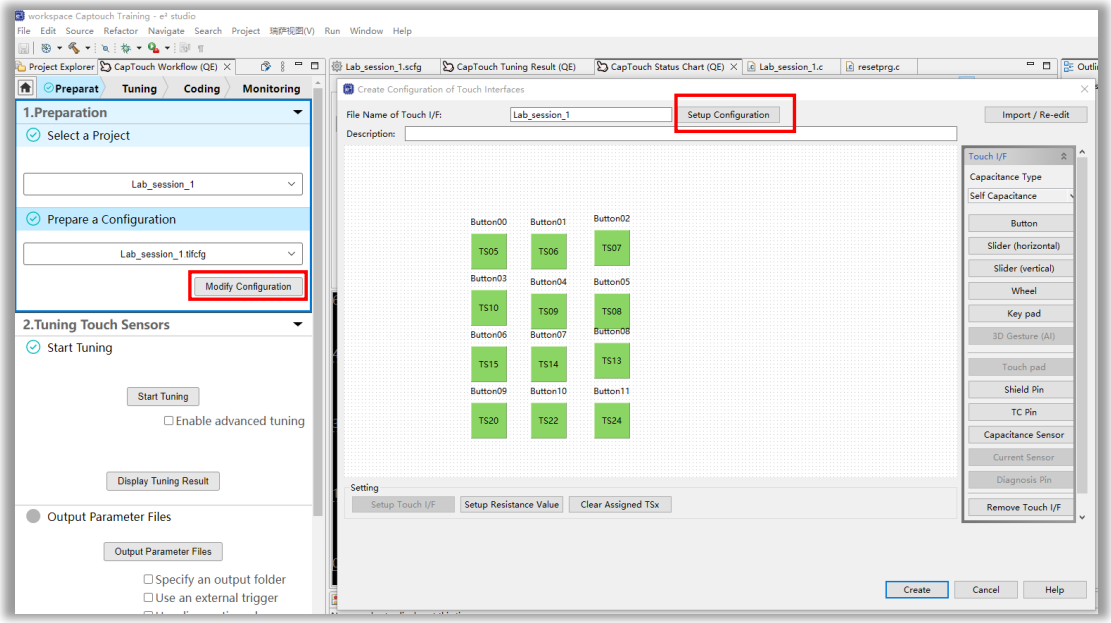
如果对 **Lab session 4** 的内容非常熟悉或者有一定困难，可跳过步骤 **5.1** 到步骤 **5.4**，
在 **e2 studio** 中 **import** 导入培训配套资料 **Checkpoints** 文件夹中的工程 **Lab session 4**，
直接进行步骤 **5.5** 到 **5.6** 的实验

实验步骤

5.1	**修改触摸接口(interface)或者配置(Configuration)**
5.1.1	在"**Lab session 3**" 的 **e2 studio** 工程中， 选择"**Renesas View** 瑞萨视图" → **Renesas QE** → **CapTouch workflow** 

5.1.2

在"**CapTouch workflow**"中，在"**1.preparation**"页面中点击"**Modify Configuration**"，
弹出"**Create Configuration of Touch Interfaces**"页面，如下图所示，点击"**Setup Configuration**"



可以通过"**Add Configuration**", 新建 **Config02**,

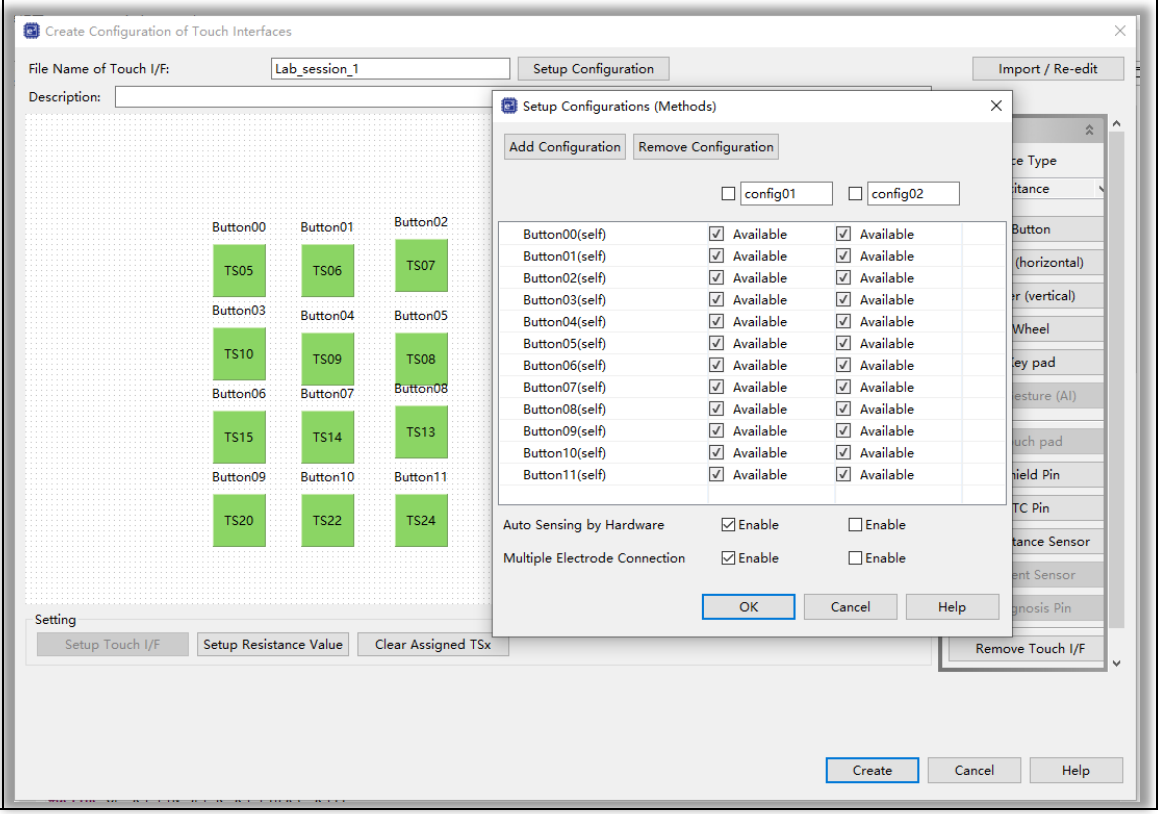
勾选 **Config01** 和 **Config02** 下方的 **Available**，为 **Button** 分组(**Configuration**)，下图中，**Config01** 和 **Config02** 都包含 **Button00** 到 **Button11** 的 **12** 个 **Button**。

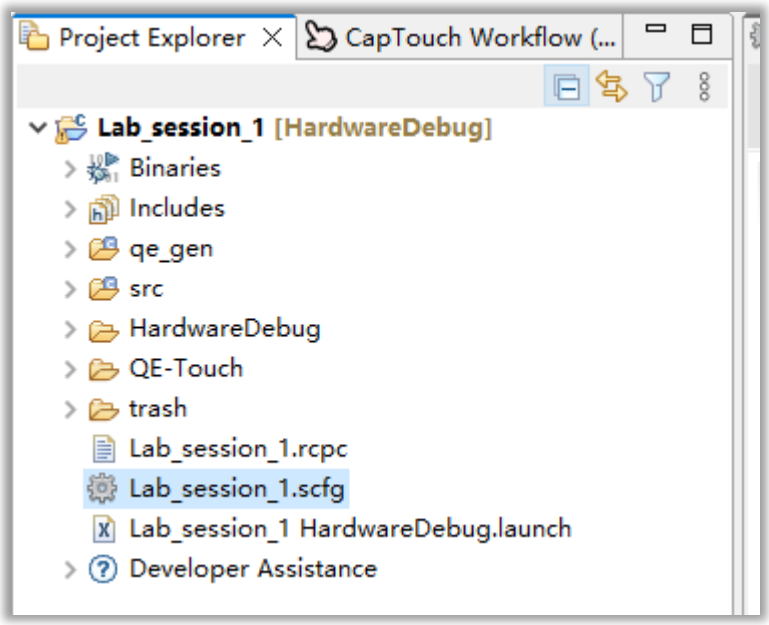
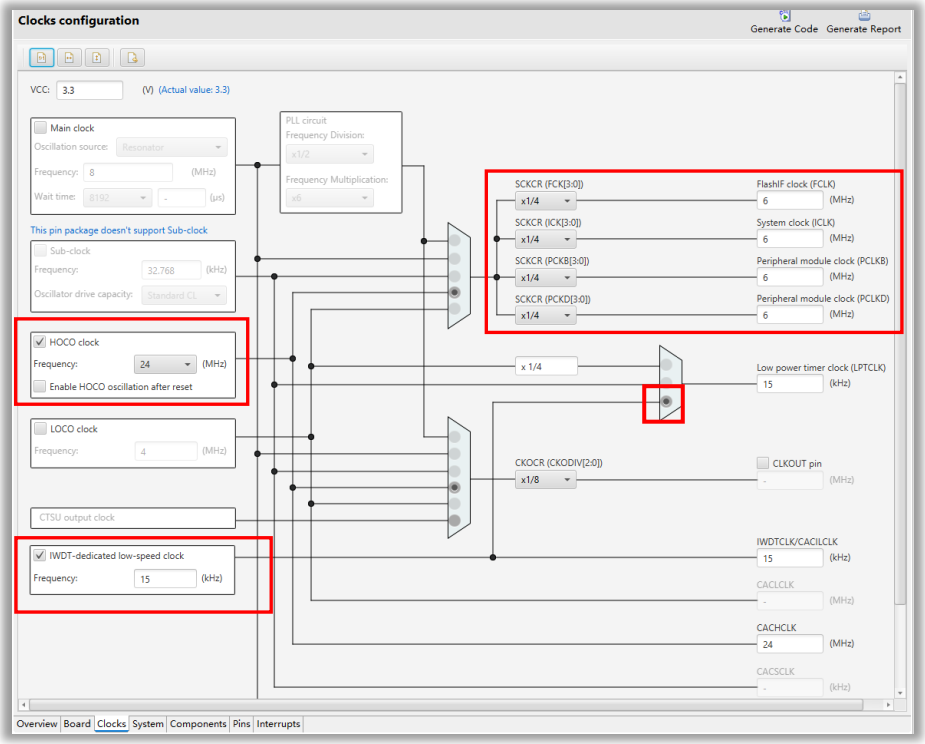
勾选 **Config01** 下方的"**Multiple Electrode Connection**"右侧的 **Enable**，将 **config01** 配置为 **MEC** 电极。

勾选 **Config01** 下方的"**Auto Sensing by hardware**"右侧的 **Enable**，使能自动判断功能。

单击 **OK**，关闭"**Setup Configuration**"对话框，回到"**Create Configuration of Touch Interfaces**"页面

单击 **Create**，在对话框中选择 **Yes** 覆盖之前的设定，完成触摸接口(**interface**)或者配置(**Configuration**)的设定。

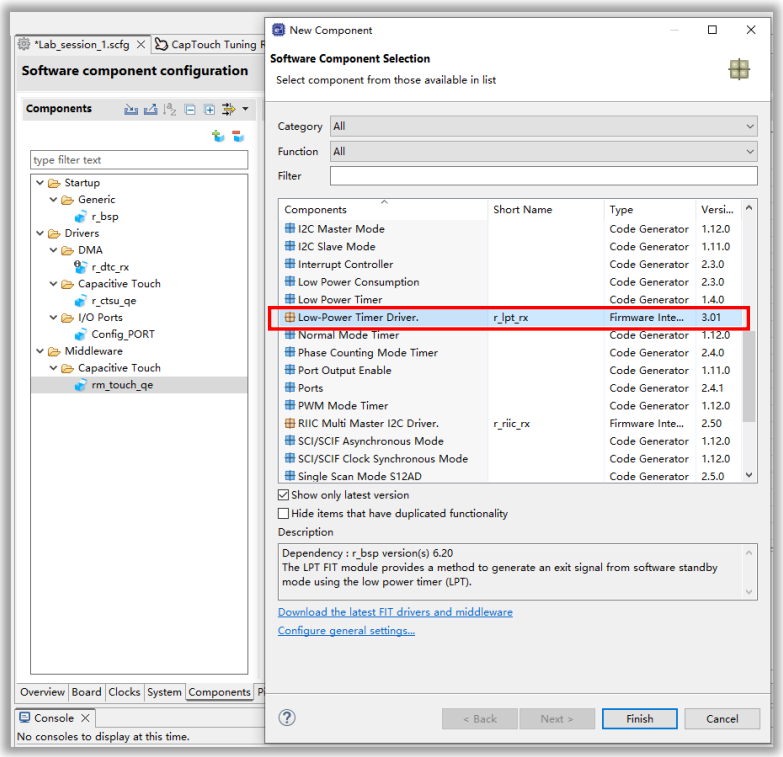


5.2	**使用 Smart Configurator 添加必要的驱动程序**
5.2.1	在"**Project Explorer**"的"**Lab_session_1**"工程中，双击"**Lab_session_1.scfg**" 打开"**Smart Configurator**" 
5.2.2	**设定Low power timer clock(LPTCLK)的时钟源** 选择"**Smart Configurator**"的"**Clocks**"选项卡 将**HOCO**改为**24MHz**，**SCKCR**都改为**1/4**, 将**FCLK**、**ICLK**、**PCLKB**、**PCLKD**都改为**6MHz** 勾选"**IWDT-dedicated low-speed clock**" 单击下图红色框内的圆点，将"**Low power timer clock(LPTCLK)** "的时钟源设定为"**IWDT-dedicated low-speed clock**" 

5.2.3 添加 Low-power Timer Driver 驱动程序

点击"**Smart Configurator**"的"**Components**"选项卡，点击  按钮，

在弹出的"**Software Components Selection**"中，取消选择"**Hide items that have duplicated functionality**"，然后找到"**Low-power Timer Driver**"，选中后点击**Finish**，完成添加。

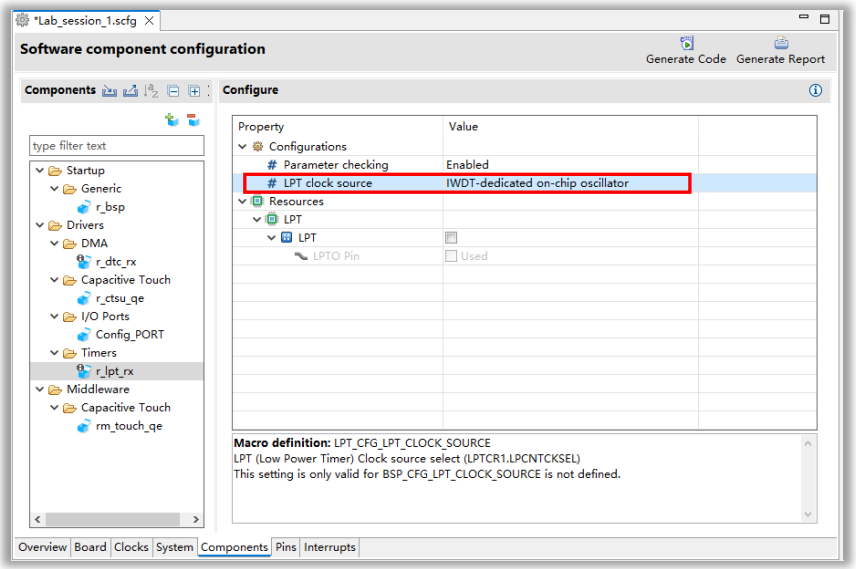


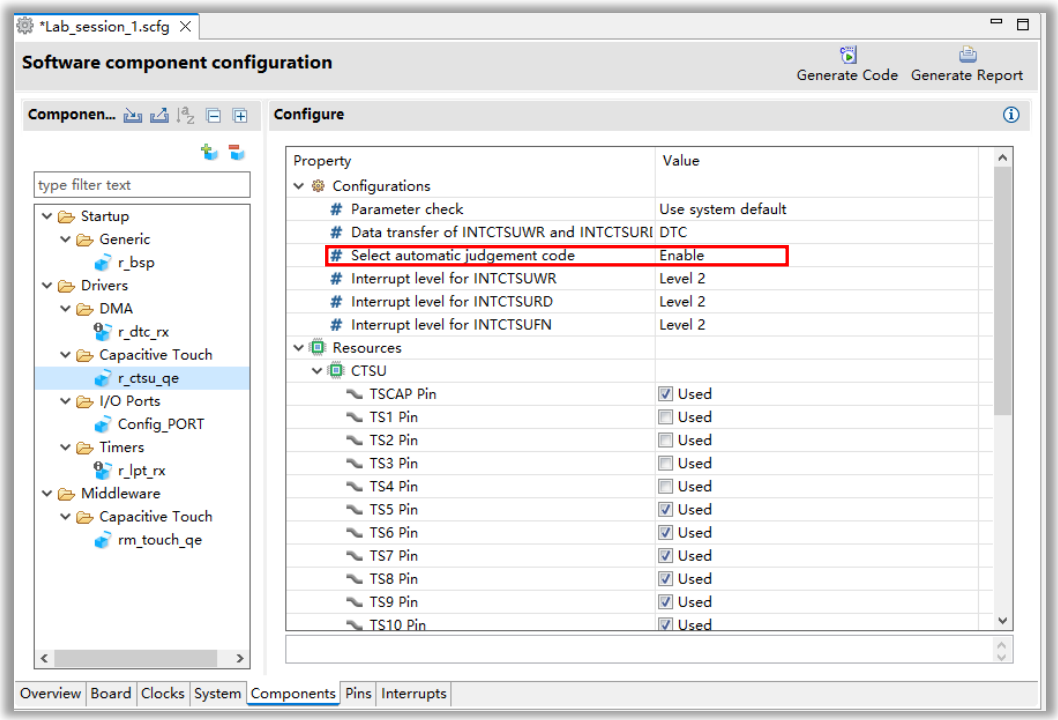
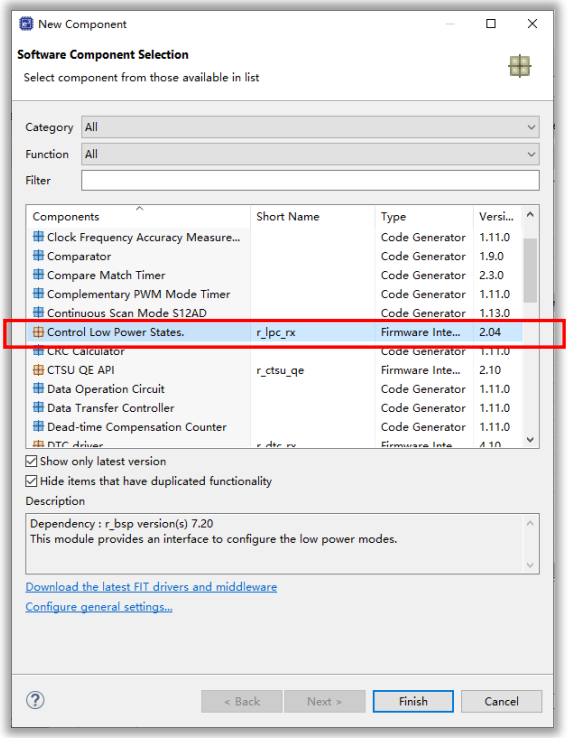
NOTE

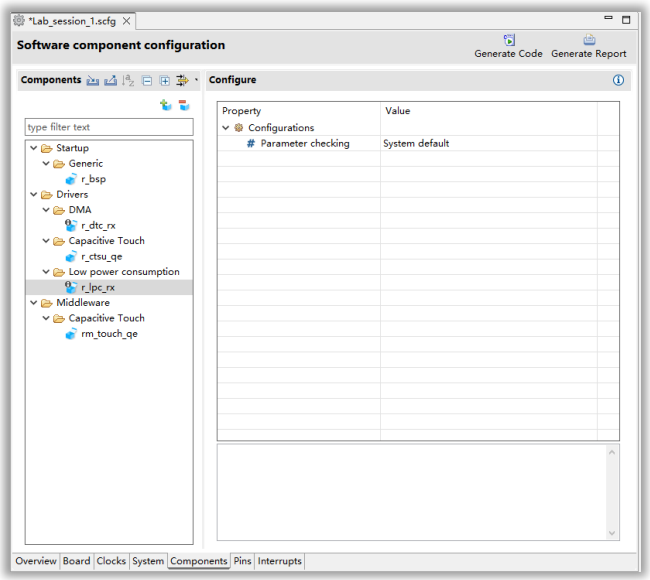
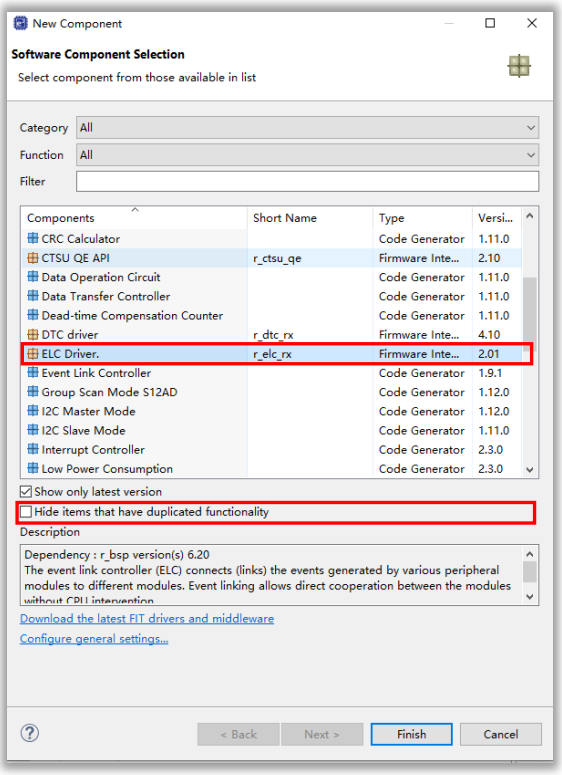
注意需要取消选择"Hide items that have duplicated functionality"
才能显示"Low-Power Timer Driver"

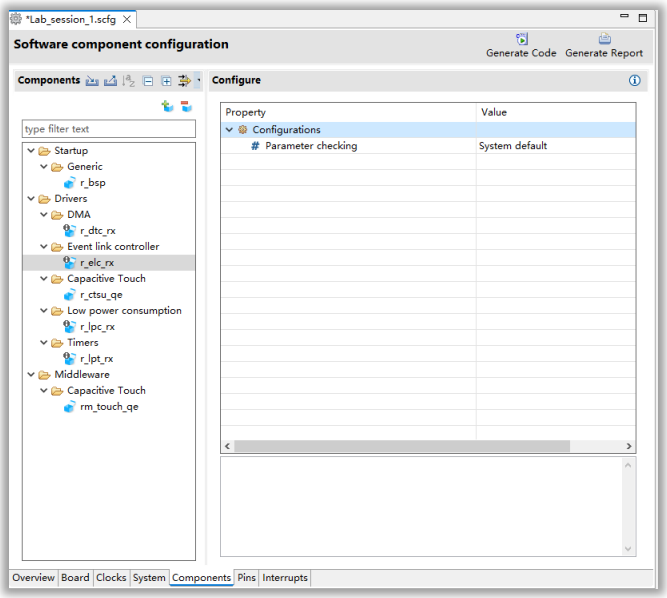
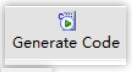
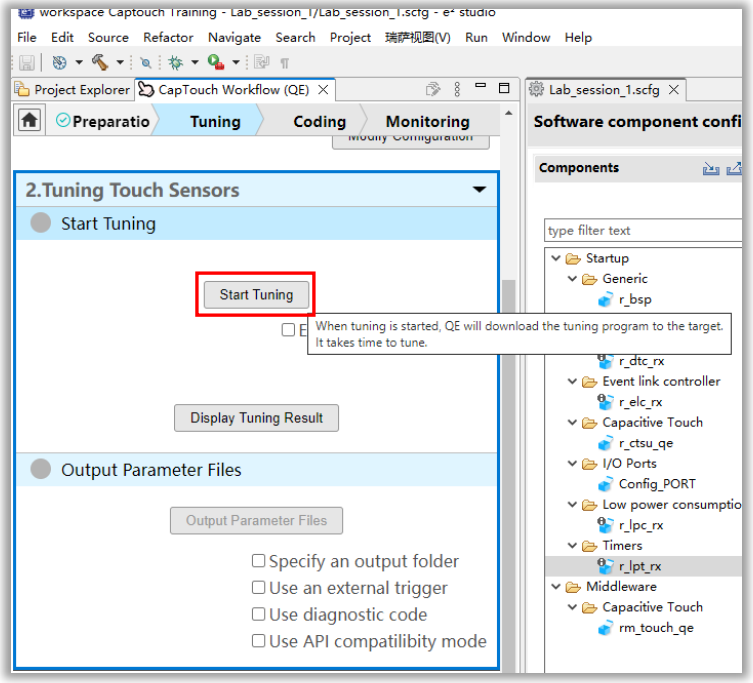
5.2.4 Low-Power Timer Driver设定

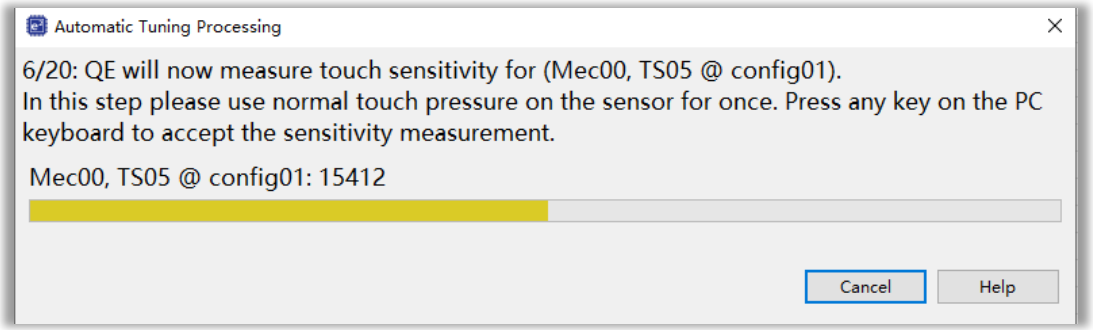
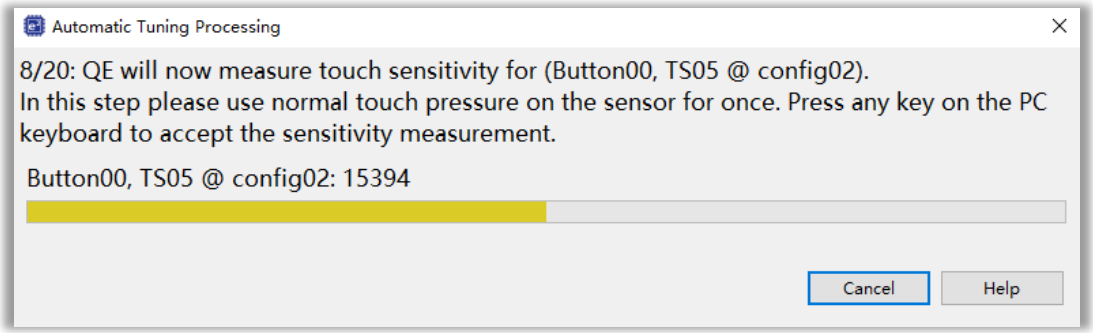
将"**LPT clock source**"的设定改为"**IWDT-dedicated on-chip oscillator**"



NOTE	"Low-Power Timer"将作为CTSU的external trigger使用
5.2.5	**r_ctsu_qe设定** 确保"**Select automatic judgement code**"的设定为**Enable** 
NOTE	automatic judgement与auto sensing、auto judgement是同一个功能
5.2.6	**添加Control Low Power States驱动程序** 选择"**Control Low Power States**" 单击**Finish** 

5.2.7	**"Control Low Power States"设定** 保持默认不变 
5.2.8	**添加ELC驱动程序** 选择**"ELC Driver"** 
NOTE	注意需要取消选择**"Hide items that have duplicated functionality"** 才能显示**"ELC Driver"**

5.2.9	**ELC Driver设定** 保持默认不变 
NOTE	**CTSU在低功耗模式下工作时，需要使用ELC**
5.2.10	点击  生成驱动程序代码 点击  图标，编译程序
5.3	**自动调整过程 (Auto Tuning Process)**
5.3.1	开始自动调整过程(**Auto Tuning Process**) 在"**Cap Touch Workflow**"的"**2.Tuning Touch Sensors**"中，单击"**Start Tuning**”． 

5.3.2	自动调整过程(**Auto Tuning Process**)开始，依次显示如下四步，这时不需要用户操作。 第一步：开始自动调整过程，引导用户按提示操作，按照要求"触摸按键"或者"不要触摸按键"。 第二步：**QE** 正在测量所有触摸按键的寄生电容。 第三步：**QE** 正在调整触摸按键的偏置电流值 第四步：**QE** 开始进行灵敏度测量
NOTE	以上自动调整过程(Auto Tuning Process)开始时的四个步骤的图片可参考
5.3.3	第五步：灵敏度测量 自动调整过程(**Auto Tuning Process**)完成前四步准备工作后，开始第五步。 如下图所示，为 **MEC** 电极，"**Mec00，TS05**"进行灵敏度测量。 使用手指或者手掌靠近 **MEC** 电极，在期望的接近传感距离停住，例如距离 **MEC** 电极 **1.5cm** 处，查看进度条的变化，按下 **PC** 键盘的任意键，接受灵敏度测量。  如下图所示，为 **Button00** 到 **Button11** 的 **12** 个按键，进行灵敏度测量。 按照提示，使用手指以正常压力按住 **Button00/TS05** 的触摸按键， 此时黄色进度条将根据手指按压触摸按键的力度而变化， 保持期望的按压力度，同时按下 **PC** 键盘的任意键，接受该触摸按键的灵敏度测量。 
NOTE	注意此时 MEC 电极作为接近传感电极工作

5.3.4

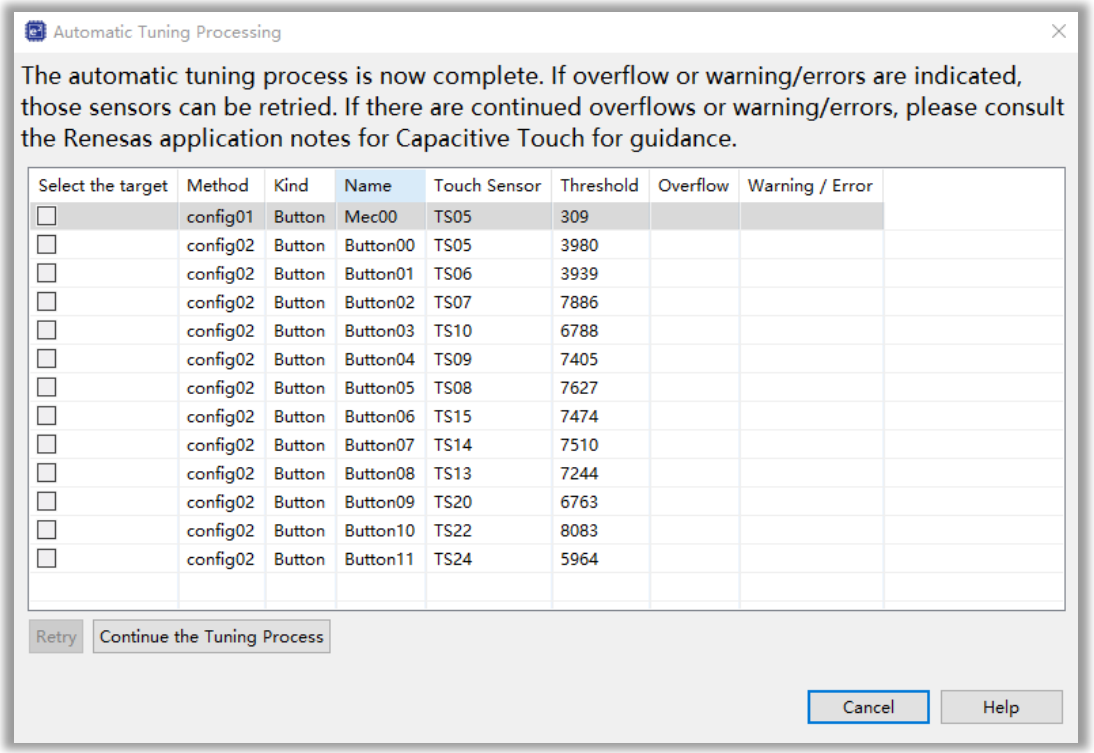
完成自动调整过程(**Auto Tuning Process**)后，自动弹出结果，

显示了 **MEC** 电极的阈值 **Threshold**。

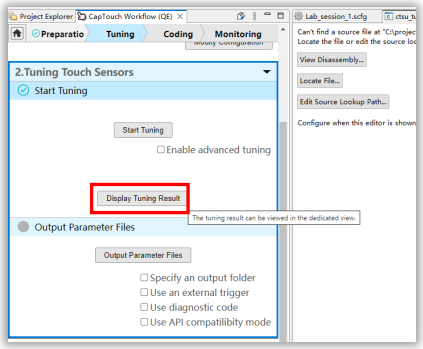
显示 **12** 个按键的阈值 **Threshold**

点击"**Continue the Tuning Process**", 自动调整过程的结果对话框关闭。

自动调整过程(**Auto Tuning Process**) 完成。



5.3.5 在"**Cap Touch Workflow**"的"**2.Tuning Touch Sensors**"中，点击"**Display Tuning Result**"



自动调整过程(**Auto Tuning Process**)的结果，如下图所示:

包括 **Method**，**Kind**，**Name**，**Touch Sensor**，**Parasitic Capacitance**，**Sensor Driver Pulse Frequency**，**Threshold**，**Scan Time**，以及 **Overflow** 等重要信息。

(受环境影响，重新进行自动调整过程时，寄生电容值会有细微差异，传感器驱动脉冲频率也有可能因寄生电容值的变化发生变化；阈值 **Threshold** 也会因按压力度的变化发生变化，阈值也可以在配置文件中直接修改)

Tuning | Gesture

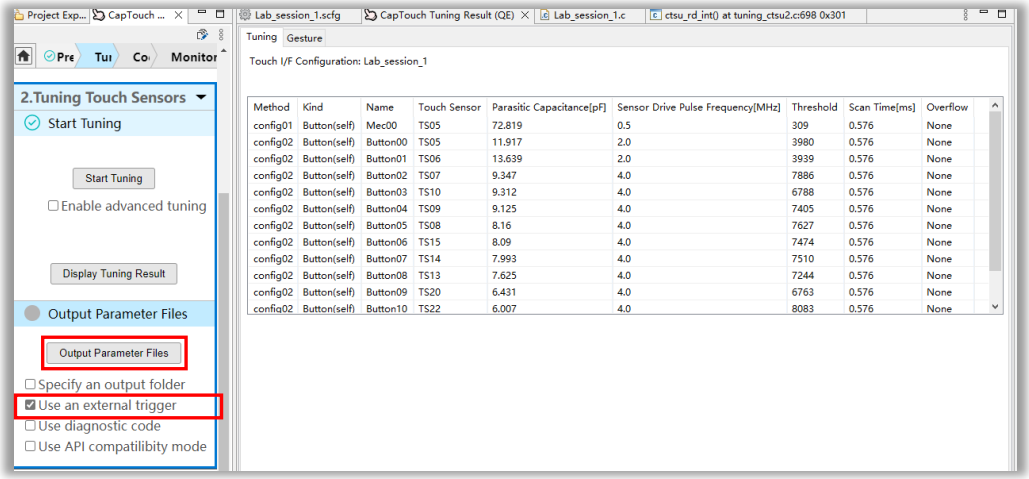
Touch I/F Configuration: Lab_session_1

Method	Kind	Name	Touch Sensor	Parasitic Capacitance[pF]	Sensor Drive Pulse Frequency[MHz]	Threshold	Scan Time[ms]	Overflow
config01	Button(self)	Mec00	TS05	72.819	0.5	309	0.576	None
config02	Button(self)	Button00	TS05	11.917	2.0	3980	0.576	None
config02	Button(self)	Button01	TS06	13.639	2.0	3939	0.576	None
config02	Button(self)	Button02	TS07	9.347	4.0	7886	0.576	None
config02	Button(self)	Button03	TS10	9.312	4.0	6788	0.576	None
config02	Button(self)	Button04	TS09	9.125	4.0	7405	0.576	None
config02	Button(self)	Button05	TS08	8.16	4.0	7627	0.576	None
config02	Button(self)	Button06	TS15	8.09	4.0	7474	0.576	None
config02	Button(self)	Button07	TS14	7.993	4.0	7510	0.576	None
config02	Button(self)	Button08	TS13	7.625	4.0	7244	0.576	None
config02	Button(self)	Button09	TS20	6.431	4.0	6763	0.576	None
config02	Button(self)	Button10	TS22	6.007	4.0	8083	0.576	None

5.3.6 **输出参数文件**

在"**Cap Touch Workflow**"的"**2.Tuning Touch Sensors**"中，勾选"**Use an external trigger**"

点击"**Output Parameter FIles**"



以下三个参数文件将被覆盖

Qe_touch_define.h

Qe_touch_config.h

QE_touch_config.c

NOTE	**"Low-Power Timer"**将作为 **CTSU** 的 **external trigger** 使用
5.4	**增加低功耗(Auto Judgement)功能应用程序**
5.4.1	将培训配套资料 **Checkpoints** 文件夹中的工程**"Lab session 4"**中的 **qe_gen** 文件的 **"qe_touch_sample.c"**拷贝并覆盖**"Project Explorer"**的 **Lab_session_1** 工程中 **qe_gen** 文件的 **"qe_touch_sample.c"**.

5.4.2	**应用程序代码说明**
	使用 init_peripheral_function 初始化需要使用的外设
	使用 R_CTSU_Open()初始化 config01(MEC 电极)
	使用 RM_TOUCH_Open()初始化 config02(12 个自容式按键)
	`/* Initialize peripheral functions */`
	`init_peripheral_function();`
	`/* Open Touch middleware */`
	`err = R_CTSU_Open (g_qe_ctsu_instance_config01.p_ctrl, g_qe_ctsu_instance_config01.p_cfg);`
	`ctsu_ctrl = (ctsu_instance_ctrl_t *)g_qe_ctsu_instance_config01.p_ctrl;`
	`err = RM_TOUCH_Open (g_qe_touch_instance_config02.p_ctrl, g_qe_touch_instance_config02.p_cfg);`

5.4.3	**应用程序代码说明** 以下代码完成 **config01**(**MEC** 电极)和 **config02**(**12** 个自容式按键电极)的初始化偏置电流调整。

```
/* Initial Offset Tuning */
{
    (void)R_LPT_SetCMT(LPT_CH1, (uint32_t)WAKEUP_LPT_PERIOD_NORMAL);
    /* Method1 offset tuning */
    do
    {
        err = R_CTSU_ScanStart (g_qe_ctsu_instance_config01.p_ctrl);
        if (FSP_SUCCESS != err)
        {
            while (true) {}
        }
        (void)R_LPT_Control(LPT_CMD_START);
        while (0 == g_qe_touch_flag) {}
        g_qe_touch_flag = 0;
        err = R_CTSU_OffsetTuning (g_qe_ctsu_instance_config01.p_ctrl);
    } while(err != FSP_SUCCESS);
    /* Method2 offset tuning */
    do
    {
        err = RM_TOUCH_ScanStart (g_qe_touch_instance_config02.p_ctrl);
       if (FSP_SUCCESS != err)
       {
           while (true) {}
       }
       (void)R_LPT_Control(LPT_CMD_START);
       while (0 == g_qe_touch_flag) {}
       g_qe_touch_flag = 0;
       err = RM_TOUCH_DataGet(g_qe_touch_instance_config02.p_ctrl,
&button_status02, NULL, NULL);
    } while(err != FSP_SUCCESS);
}
```

5.4.4

应用程序代码说明

以下代码完成 **config01**(**MEC** 电极)**在 Normal 模式下**的 **baseline** 调整。

```
/* base line setting @method1 */
for (uint32_t i = 0U; i < WAKEUP_TIME_BASELINE; i++)
{
     err = R_CTSU_ScanStart (g_qe_ctsu_instance_config01.p_ctrl);
     if (FSP_SUCCESS != err)
     {
         while (true) {}
     }
     (void)R_LPT_Control(LPT_CMD_START);
     R_BSP_SoftwareDelay(WAKEUP_WAIT_MEASUREEND, BSP_DELAY_MILLISECS);
     (void)R_LPT_Control(LPT_CMD_STOP);
     (void)R_LPT_Control(LPT_CMD_COUNT_RESET);
     ctsu_ctrl->state = CTSU_STATE_SCANNED;
     err = R_CTSU_AutoJudgementDataGet (g_qe_ctsu_instance_config01.p_ctrl,
&button_status01);
     if (FSP_SUCCESS == err)
     {
        RM_TOUCH_MonitorAddressGet (g_qe_touch_instance_config02.p_ctrl,
                                    &monitor_buf_address,
                                    &monitor_id_address,
                                    &monitor_size_address);
       qe_monitor_autojudge (g_qe_ctsu_instance_config01.p_ctrl);
     }
}
```

5.4.5

应用程序代码说明

以下代码完成进入低功耗模式的操作，在低功耗模式里完成 **config01**(**MEC** 电极)的测量和自动判断，当 **config01**(**MEC** 电极)有按键 **On** 判断是时，退出低功耗，并通过 **`R_CTSU_AutoJudgementDataGet()`** 取得结果。

```
/* Standby mode */
{
    /* for [CONFIG01] configuration */
    (void)R_LPT_SetCMT(LPT_CH1, (uint32_t)WAKEUP_LPT_PERIOD_STANDBY);
    err = R_CTSU_ScanStart (g_qe_ctsu_instance_config01.p_ctrl);
    /* Enter software standby mode */
    lpc_err = R_LPC_LowPowerModeActivate(&activate_standby_callback);
    if (LPC_SUCCESS != lpc_err)
    {
        while (true) {}
    }
    while (0 == g_qe_touch_flag) {}
    g_qe_touch_flag = 0;
    err = R_CTSU_AutoJudgementDataGet (g_qe_ctsu_instance_config01.p_ctrl,
&button_status01);
    if (FSP_SUCCESS == err)
    {
        RM_TOUCH_MonitorAddressGet (g_qe_touch_instance_config02.p_ctrl,
                                    &monitor_buf_address,
                                    &monitor_id_address,
                                    &monitor_size_address);
        qe_monitor_autojudge (g_qe_ctsu_instance_config01.p_ctrl);
    }
}
```

5.4.6

应用程序代码说明

以下代码为进入**Software Standby Mode**时的**Callback**程序，用于启动**Low-power Timer**定时器

```
/* activate_standby_callback */
static void activate_standby_callback(void *p_data)
{
    lpt_err_t  lpt_err;
    /* Start LPT count */
    lpt_err = R_LPT_Control(LPT_CMD_START);
    if (LPT_SUCCESS != lpt_err) while(1);
}
```

5.4.7

应用程序代码说明

以下代码为退出**Snooze Mode**时的**Callback**程序，用于停止和复位**Low-power Timer**定时器，以及**disable snooze release interrupt**.

```
/*snooze_callback*/
static void snooze_callback(void *p_data)
{
    lpt_err_t lpt_err;
    lpc_err_t lpc_err;
    /* Stop LPT count */
    lpt_err = R_LPT_Control(LPT_CMD_STOP);
    if (LPT_SUCCESS != lpt_err) while(1);
    /* Reset LPT count */
    lpt_err = R_LPT_Control(LPT_CMD_COUNT_RESET);
    if (LPT_SUCCESS != lpt_err) while(1);
    /* Disable snooze release interrupt */
    lpc_err = R_LPC_SnoozeModeConfigure(&gs_snooze_mode);
    if (LPC_SUCCESS != lpc_err) while(1);
}
```

5.4.8

点击  图标，编译程序。

5.4.9

以上应用程序代码的详细解释，

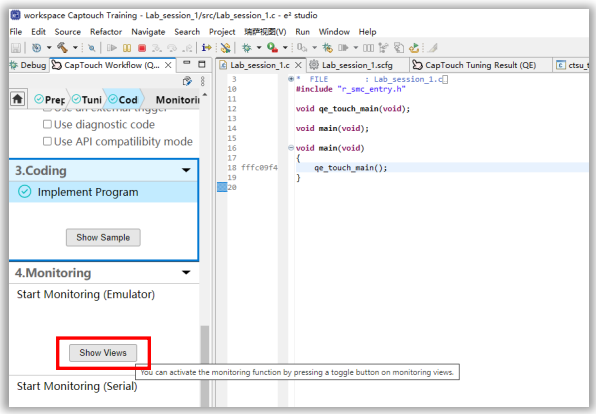
可参考瑞萨官网的应用笔记 **RX140 Group Smart Wakeup Solution Rev.1.00** 以及配套的**样例程序**

5.5 使用 QE for Cap Touch 监控触摸底层数据以及触摸行为

5.5.1

按照"**2.6 运行程序**"小节介绍的方法，在仿真状态下全速运行程序。

在"**Cap Touch Workflow**"的"**4.monitoring**"中，点击"**Start Monitoring(Emulator)**"下方的"**Show Views**"

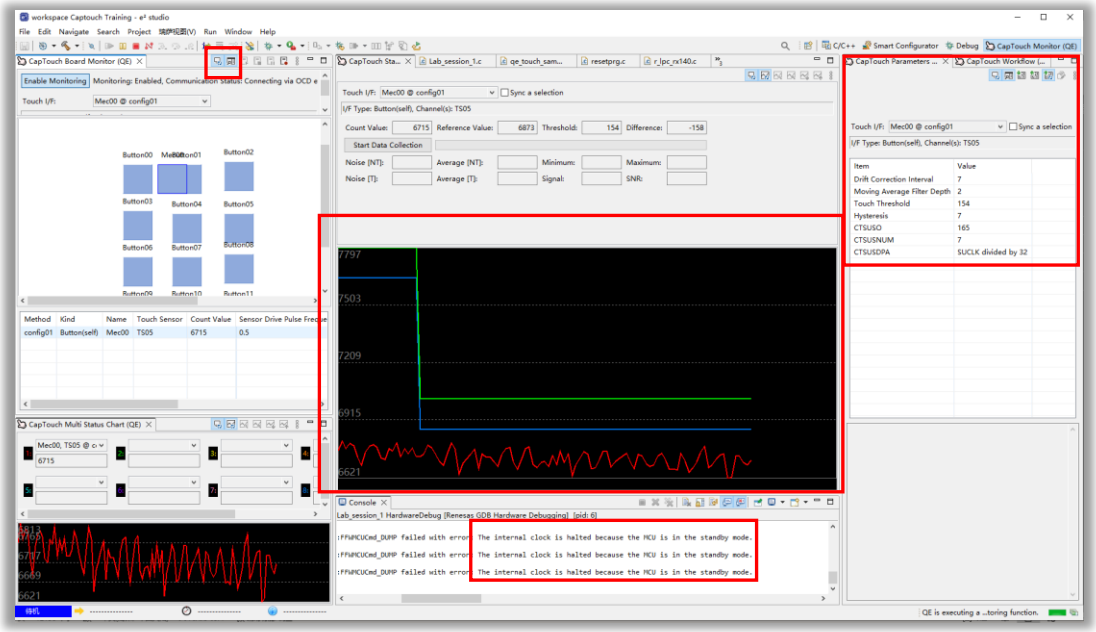


5.5.2

具体操作**大体上**可以按照"**2.9 使用 QE for Cap Touch 监控触摸底层数据以及触摸行为**"小节介绍的方法进行。

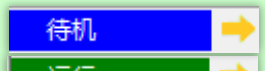
不同点在于：

- 在低功耗状态下，即时监控窗口"**Enbale Monitoring**"，也无法使用 **QE for Cap Touch** 监控触摸底层数据以及触摸行为。各个监控窗口处于停止状态，数据和波形曲线停止刷新。
- 在"**CapTouch Parameters (QE) View**"中的 **MEC** 电极(**Config01**)的参数项比之前少。
- 在"**e2 studio**"下方"**Console** 控制台"窗口显示了 **MCU** 当前处于低功耗状态。
- 在"**e2 studio**"状态栏的左下角，显示了当前工程的运行状态处于"待机"状态。

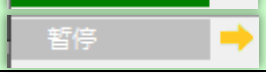


NOTE

在 **e2 studio** 左下角的状态栏，显示了当前工程的运行状态，包括以下几种。

在低功耗时显示： 

在正常运行时显示： 

在程序暂时显示： 

5.6 调试低功耗(Auto Judgement)功能运行参数

5.6.1

在这一部分，我们主要关注以下三部分的参数调整：

1. 低功耗工作模式下 MEC 电极的运行参数

- **MEC** 电极的阈值 **Threshold**（为 **Normal** 模式下工作时的 **50%**）

2. 低功耗工作模式下的 Auto Judgement 自动判断功能的相关参数

在**"Lab_session_1" → src → smc_gen → r_ctsu_qe → doc → en** 文件夹下的

应用笔记**"r01an4469ej0210-rx.pdf"**中，对**"Auto Judgement"**自动判断功能的相关参数进行了详细的说明，如下图所示：

其中主要的控制参数如下：

Tlot，**Non-touch judgment continuous count**，与 **Button 的 Negative Noise Filter Cycle** 意义相同

Thot，**Touch judgment continuous count**，与 **Button** 的 **Positive Noise Filter Cycle** 意义相同

Ajmmat，**Moving average**，与 **Button** 的 **Moving Average Filter Depth** 意义相同

Ajbmat，**Baseline average count**，与 **Button** 的 **Drift Correction Interval** 意义相同

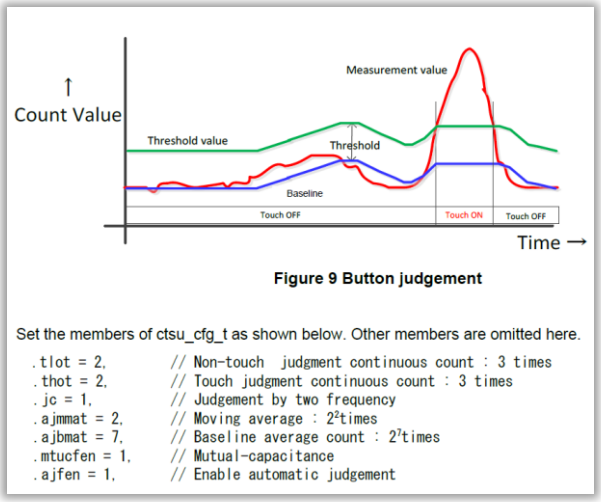
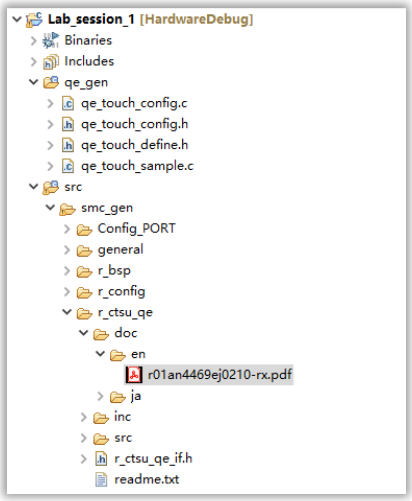


Figure 9 Button judgement

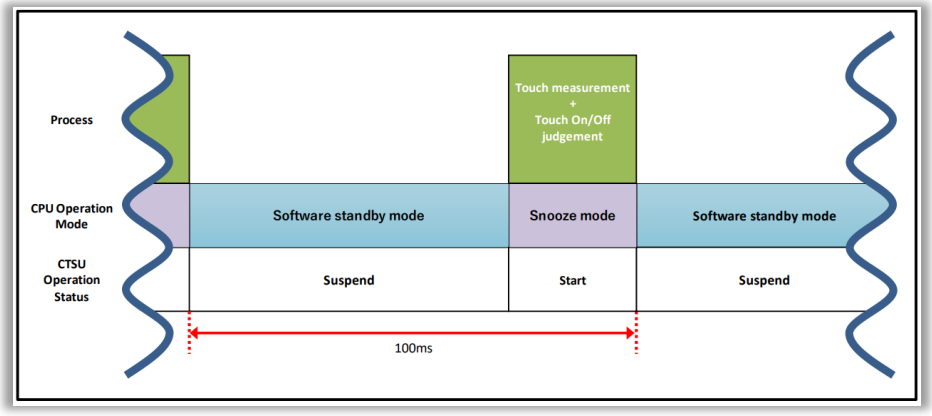
Set the members of ctsu_cfg_t as shown below. Other members are omitted here.

```
.tlot = 2,          // Non-touch  judgment continuous count : 3 times
.thot = 2,          // Touch judgment continuous count : 3 times
.jc = 1,            // Judgement by two frequency
.ajmmat = 2,        // Moving average : 2^2times
.ajbmat = 7,        // Baseline average count : 2^7times
.mtucfen = 1,       // Mutual-capacitance
.ajfen = 1,         // Enable automatic judgement
```



3.其他低功耗工作相关的参数

- 低功耗控制周期, 见下图
- 低功耗模式下的 **MEC** 电极的 **Sensor** 驱动脉冲频率
- 无按键按下时的等待时间

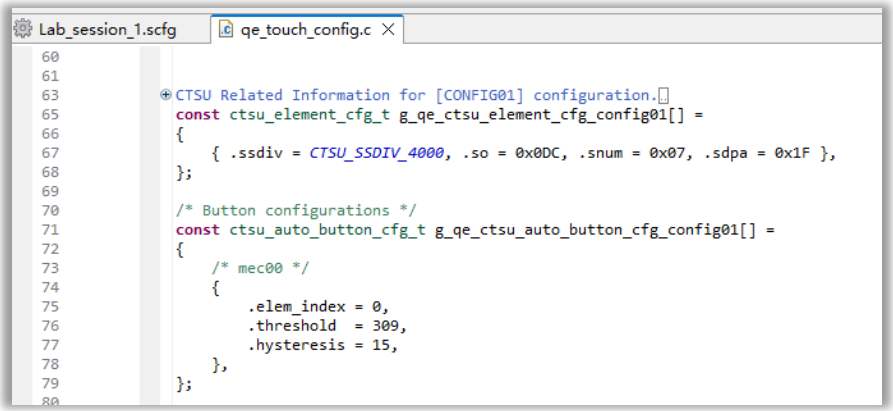


5.6.2

1. 低功耗工作模式下 MEC 电极的运行参数

- **MEC** 电极的阈值 **Threshold**（为 **Normal** 模式下工作时的 **50%**）

举例来说，在"**qe_touch_config.c**"中可以看到 **MEC** 电极的阈值 **Threshold** 为 **309.**



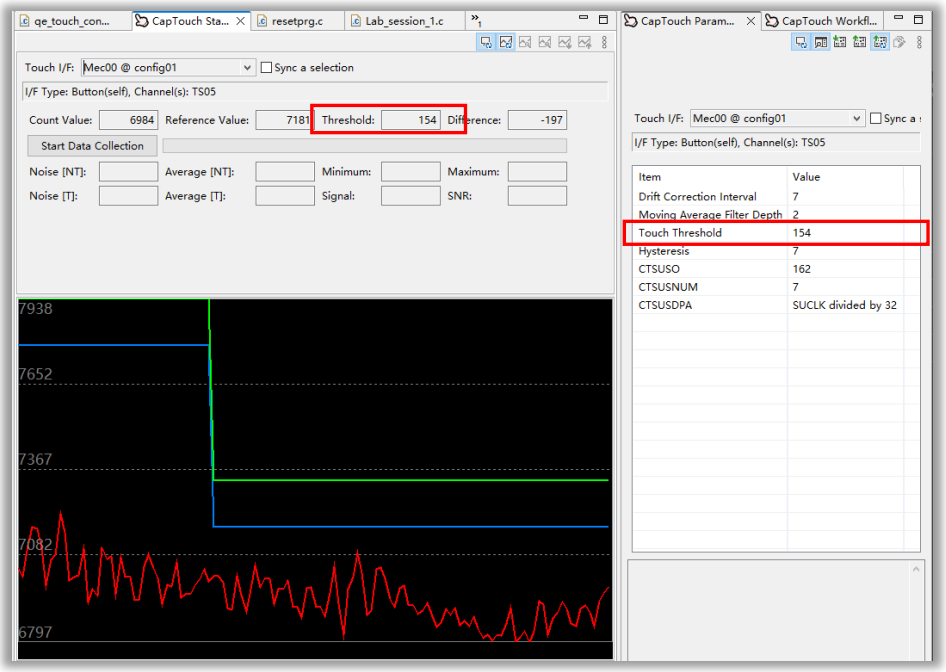
但是在 **QE** 监控窗口看到的阈值 **Threshold** 为 **309** 的 **50%**，为 **154.**

为 **Normal** 模式下工作时的 **50%**的原因如下：

CTUS2 为三频率测量，三频率测量是丢弃一个异常值，最终两个频率的值求和，因此自动调整过程 (**Auto tuning process**) 输出一个频率测量 **x2** 的结果，并反映在 **Log** 日志中。使用低功耗 (**Auto Judgement**) 功能时的监测只显示一个频率的结果，因此结果是日志中显示的阈值的一半。这是因为即使在测量三个频率时，自动判断版本也不会组合这些值，而是对每个频率单独执行触摸判断【这个机制以后可能会修改】。

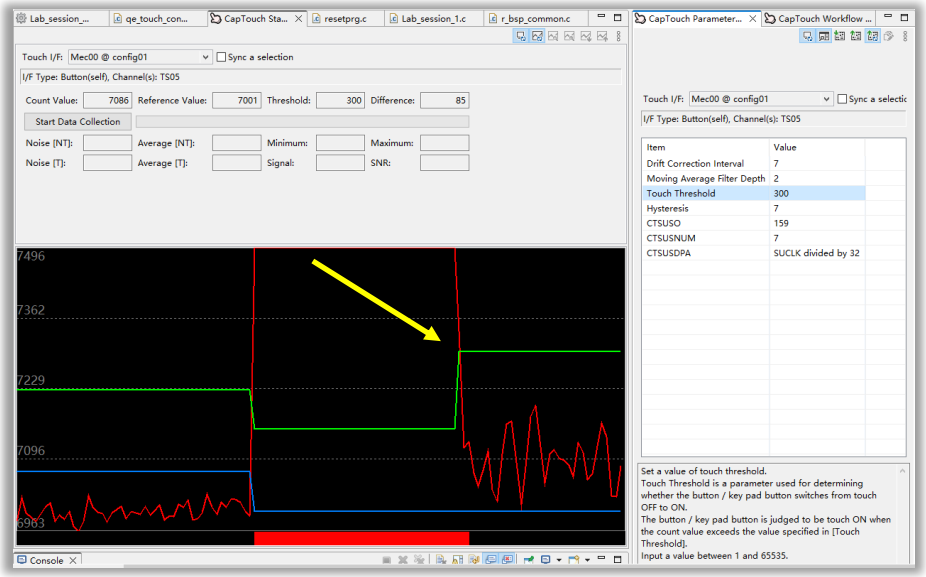
低功耗模式下 **MEC** 电极阈值 **Threshold**，**建议**使用"**CapTouch Parameters (QE) View**"在程序运行状态下通过"**Touch Threshold**"修改并调试，然后在"**qe_touch_config.c**"中直接修改。

调试时，先通过手指接近 **MEC** 电极，使系统自动判断有按键动作，退出低功耗模式返回 **Normal** 模式，然后"**CapTouch Parameters (QE) View**"中修改"**Touch Threshold**"，手指再次接近 **MEC** 电极观测灵敏度变化，由于 **MEC** 电极在 **Normal** 模式下运行的时间非常短，因此需要反复调试以达到满意的效果。



5.6.3

将低功耗工作模式下 **MEC** 电极的阈值 **Threshold**，调整为 **300** 的示例操作，如下图所示：



NOTE

低功耗工作模式下的 **MEC** 电极除了在低功耗模式下运行外，在 **Normal** 模式下的也会短暂运行，因为需要在 **Normal** 模式下进行 **baseline** 调整，因此在 **baseline** 调整期间可以通过 **QE** 的监控窗口调试和修改阈值 **Threshold**。在 **Normal** 模式下的短暂运行的时间，也可以调整。

5.6.4

2. 低功耗工作模式下的 Auto Judgement 自动判断功能的相关参数

以下三个参数，由于无法在低功耗模式下仿真调试，因此只能直接在"**qe_touch_config.c**"中直接修改。

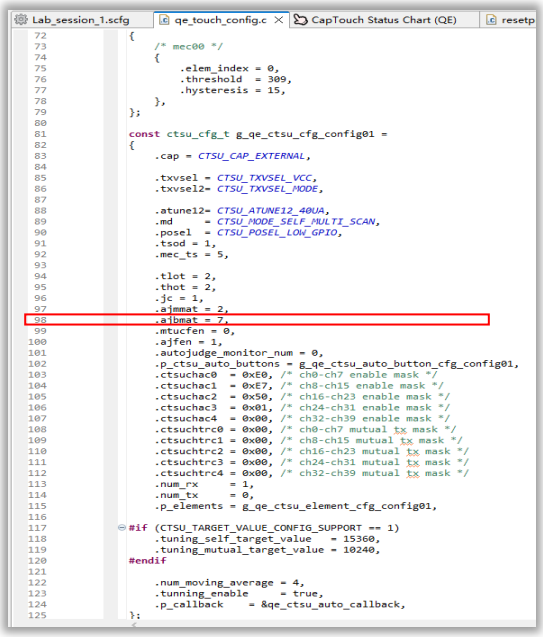
Tlot，**Non-touch judgment continuous count**，与 **Button** 的"**Negative Noise Filter Cycle**"意义相同

Thot，**Touch judgment continuous count**，与 **Button** 的"**Positive Noise Filter Cycle**"意义相同

Ajmmat，**Moving average**，与 **Button** 的"**Moving Average Filter Depth**"意义相同

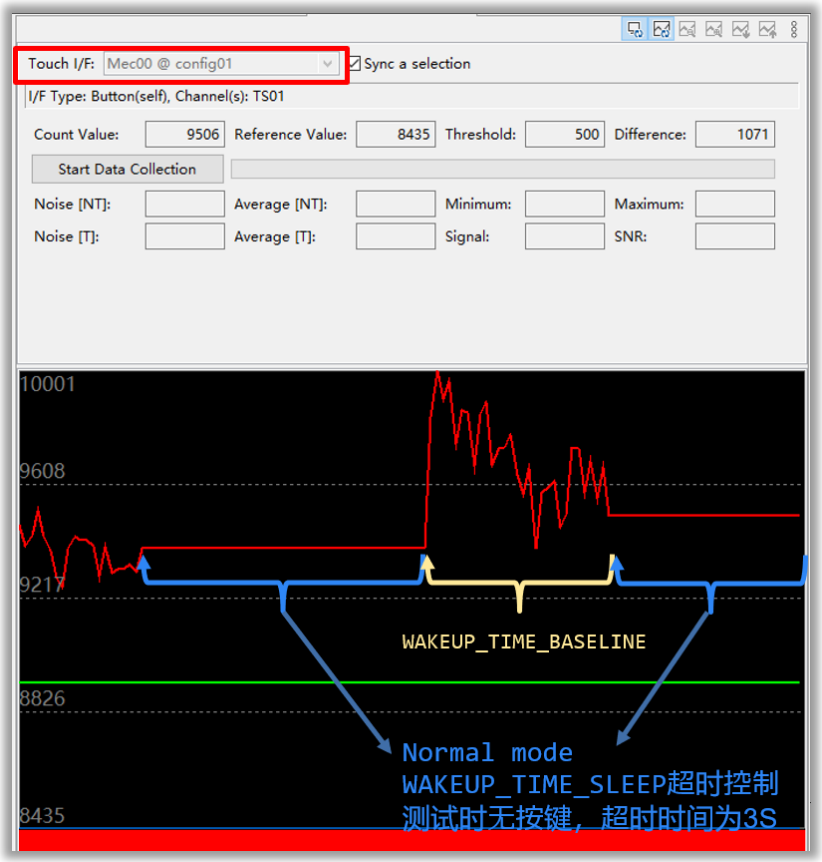
下面的参数，由于工作在 **Normal** 模式下，因此可以 **QE** 监控窗口进行调试和修改。

Ajbmat，**Baseline average count**，与 **Button** 的"**Drift Correction Interval**"意义相同，也可以在"**qe_touch_config.c**"中直接修改，如下图：



```
{
    /* mec00 */
    {
        .elem_index = 0,
        .threshold  = 309,
        .hysteresis = 15,
    },
};

const ctsu_cfg_t g_qe_ctsu_cfg_config01 =
{
    .cap = CTSU_CAP_EXTERNAL,

    .txvsel = CTSU_TXVSEL_VCC,
    .txvsel2= CTSU_TXVSEL_MODE,

    .atune12= CTSU_ATUNE12_40UA,
    .md     = CTSU_MODE_SELF_MULTI_SCAN,
    .posel  = CTSU_POSEL_LOW_GPIO,
    .tsod = 1,
    .mec_ts = 5,

    .tlot = 2,
    .thot = 2,
    .jc = 1,
    .ajmmat = 2,
    .ajbmat = 7,
    .mtucfen = 0,
    .ajfen = 1,
    .autojudge_monitor_num = 0,
    .p_ctsu_auto_buttons = g_qe_ctsu_auto_button_cfg_config01,
    .ctsuchac0  = 0xE0, /* ch0-ch7 enable mask */
    .ctsuchac1  = 0xE7, /* ch8-ch15 enable mask */
    .ctsuchac2  = 0x50, /* ch16-ch23 enable mask */
    .ctsuchac3  = 0x01, /* ch24-ch31 enable mask */
    .ctsuchac4  = 0x00, /* ch32-ch39 enable mask */
    .ctsuchtrc0 = 0x00, /* ch0-ch7 mutual tx mask */
    .ctsuchtrc1 = 0x00, /* ch8-ch15 mutual tx mask */
    .ctsuchtrc2 = 0x00, /* ch16-ch23 mutual tx mask */
    .ctsuchtrc3 = 0x00, /* ch24-ch31 mutual tx mask */
    .ctsuchtrc4 = 0x00, /* ch32-ch39 mutual tx mask */
    .num_rx     = 1,
    .num_tx     = 0,
    .p_elements = g_qe_ctsu_element_cfg_config01,

#if (CTSU_TARGET_VALUE_CONFIG_SUPPORT == 1)
    .tuning_self_target_value   = 15360,
    .tuning_mutual_target_value = 10240,
#endif

    .num_moving_average = 4,
    .tunning_enable     = true,
    .p_callback     = &qe_ctsu_auto_callback,
};
```

5.6.5	**特别注意，在"qe_touch_config.c"中直接修改 Ajbmat 后，还需要在"qe_touch_sample.c"中，修改如下宏定义：**
	`/* Baseline number = 256[Times] at AJBMAT = 7 */`
	`#define WAKEUP_TIME_AJBMAT (256)`
	`#define WAKEUP_TIME_BASELINE (WAKEUP_TIME_AJBMAT * 2)`
	WAKEUP_TIME_AJBMAT 的设定值为 Ajbmat 的设定值的 2+1 次方，
	例如 Ajbmat 的设定值为 7 时 WAKEUP_TIME_AJBMAT 的设定值为 2^{7+1}=256

NOTE	**Baseline 调整的应用程序代码详见 5.4.4 小节。** **如果 baseline 调整的时间设定过长，会影响按键的响应时间，影响用户体验**
5.6.6	下图为将 **Ajbmat** 设定为 **3** 时的调整示例，如下图： 在"**qe_touch_config.c**"中，将 **Ajbmat** 设定为 **3**， 在"**qe_touch_sample.c**"中，将 **WAKEUP_TIME_AJBMAT** 设定为 **16** 可以看到，**baseline** 调整的时间非常短。 此时，**MEC** 电极的测量值有可能来不及跌落阈值之下，此时即便回到低功耗模式之中，由于按键判定依然为 **On** 状态，所以会立即退出低功耗模式回到 **Normal** 模式。因此，要合理设定 **Ajbmat** 的值。 

5.6.7

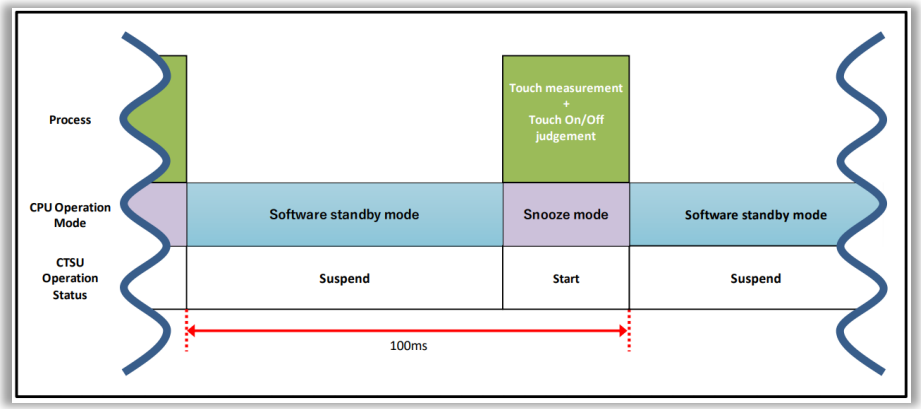
3.其他低功耗工作相关的参数

- 低功耗控制周期
- 低功耗模式下的 **MEC** 电极的 **Sensor** 驱动脉冲频率
- 无按键按下时的等待时间

低功耗控制周期

低功耗控制周期的设定越长，功耗越低，响应时间也越长，即灵敏度很低。

要根据产品应用，合理的设定低功耗控制周期。



在"**qe_touch_sample.c**"中，通过修改以下两个宏定义，修改控制周期，当前设定值为 **100000**，即 **100ms**。

修改低功耗控制周期后，可通过电流表查看电流波形以及功耗数据。

```
/* LPT cycle = 100000[microseconds] (100 microseconds) */

#define WAKEUP_LPT_PERIOD (100000)

/* LPT compare = 100000[microseconds] (100 microseconds) */

#define WAKEUP_LPT_PERIOD_STANDBY (100000)
```

低功耗模式下的 MEC 电极的 Sensor 驱动脉冲频率

在低功耗模式下，仅 **MEC** 电极在工作，修改 **MEC** 电极的 **Sensor** 驱动脉冲频率，会影响功耗数据。

低功耗模式下的 **MEC** 电极的 **Sensor** 驱动脉冲频率设定越大，功耗越大。

Sensor 驱动脉冲频率可设定的最小值为 **0.5MHz**。

在本例中，MEC 电极的 Sensor 驱动脉冲频率从 **0.5MHz** 改为 **1MHz** 后，功耗数据会有 **5uA** 左右的提高。

无按键按下时的等待时间

在"**qe_touch_sample.c**"中，通过修改以下宏定义，修改在 **Normal** 模式下，无按键按键返回低功耗模式的等待时间。

```
#define WAKEUP_TIME_SLEEP (3000U) /* 3sec*/
```

END OF SECTION

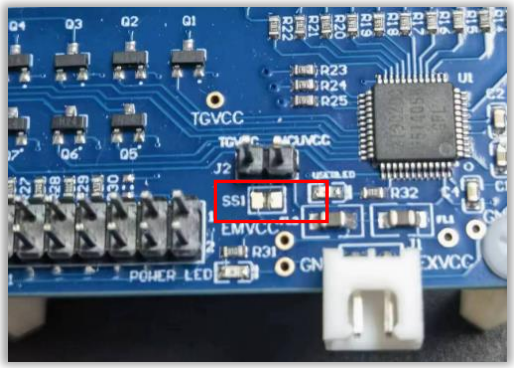
6 Lab Session 5： 在 Lab 4 的基础上使用 DMM7510 测试低功耗数据

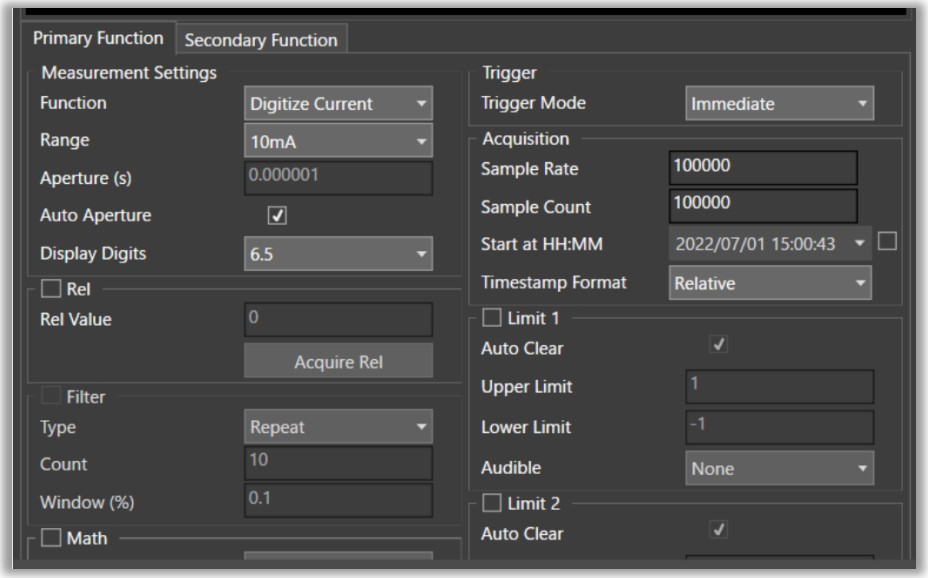
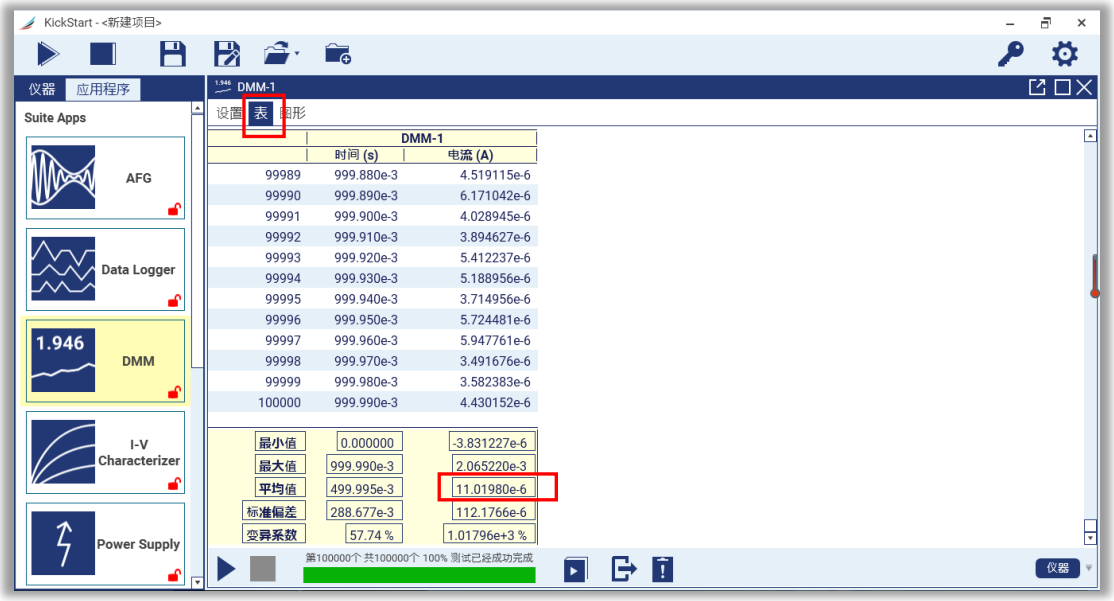
概述

在本实验环节中，将介绍使用 **Keithley DMM7510** 数字多功能表，在 **Lab session 4** 的基础上测试低功耗数据.

- **DMM7510** 需要设定在数字化电流的测量模式，量程 **10mA**，采样率 **100000** 点/秒，采样数 **100000**.
- 启动测量，查看结果
  - 以图形的形式，查看系统在低功耗模式下(**Software Standby mode** 和 **Snooze mode**)的电流波形
  - 查看平均电流/功耗数据
- **6.1** 硬件准备
- **6.2 Keithley DMM7510** 设定
- **6.3** 启动测量
- **6.4** 修改低功耗控制周期

实验步骤

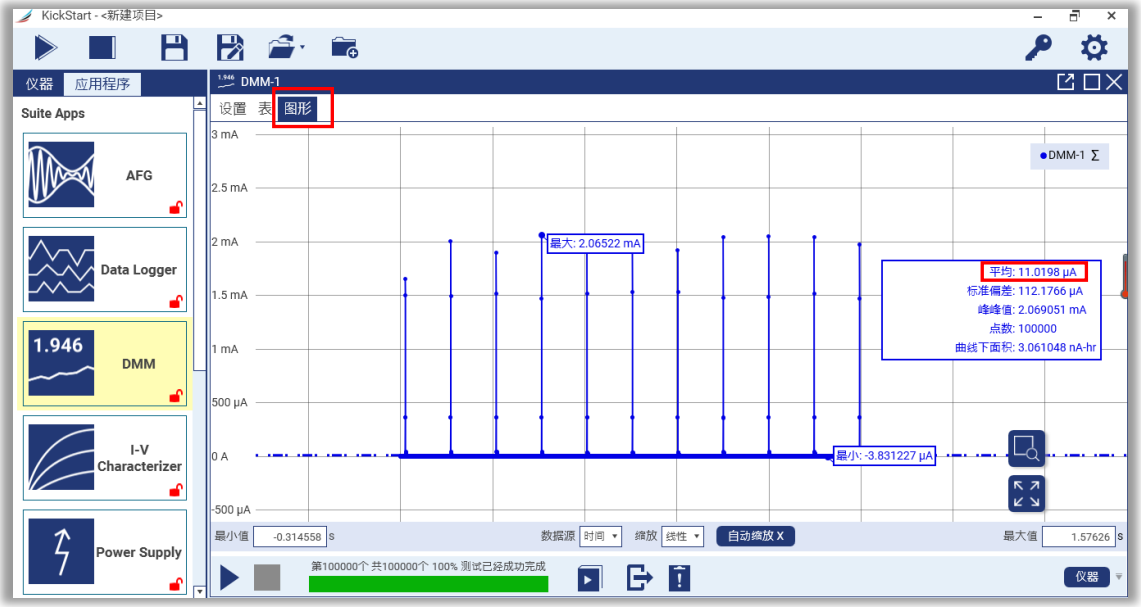
6.1	**硬件准备**
6.1.1	将评价板的 **SS1** 跳线断开，如下图所示 
6.1.2	按照下图将 **Keithley DMM7510** 的电流测量表笔连接到 **J2**，使用装有两节电池的电池盒给评价板供电。 

6.2	**Keithley DMM7510 设定**
6.2.1	将 **Keithley DMM7510** 按下图设定: **Function：Digitize Current** **Range：10mA** **Sample Rate: 100000** **Sample Count: 100000** 
6.3	**启动测量**
6.3.1	**启动测量后，得到测量结果** **以数据的形式，查看功耗数据** **100000** 个测试数据的平均功耗为：**11uA** 

6.3.2

以图形的形式，查看功耗数据

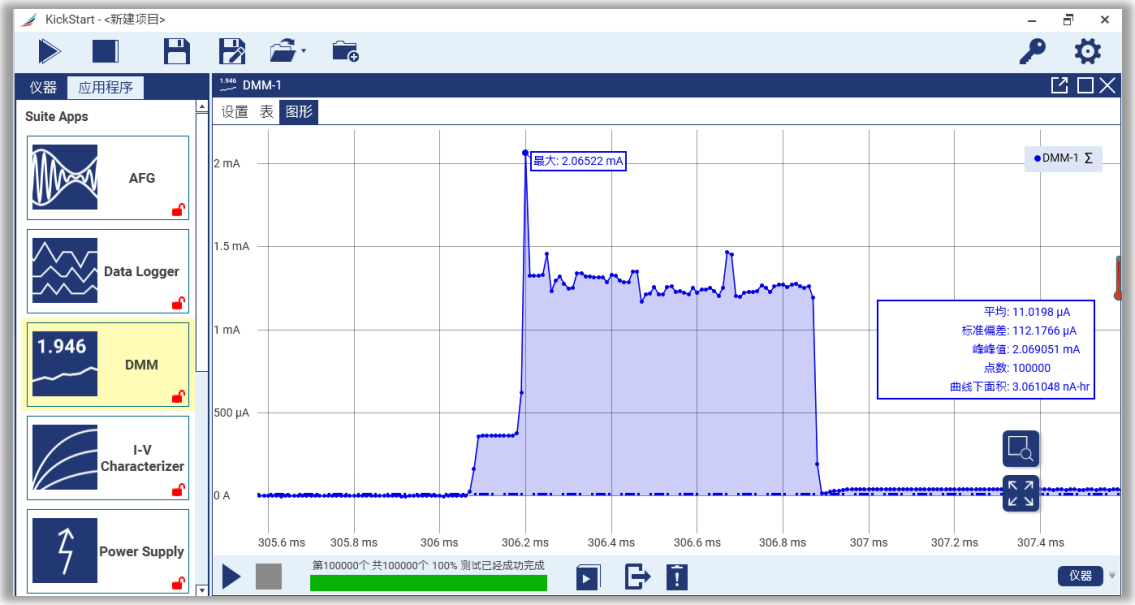
100000 个测试数据的波形



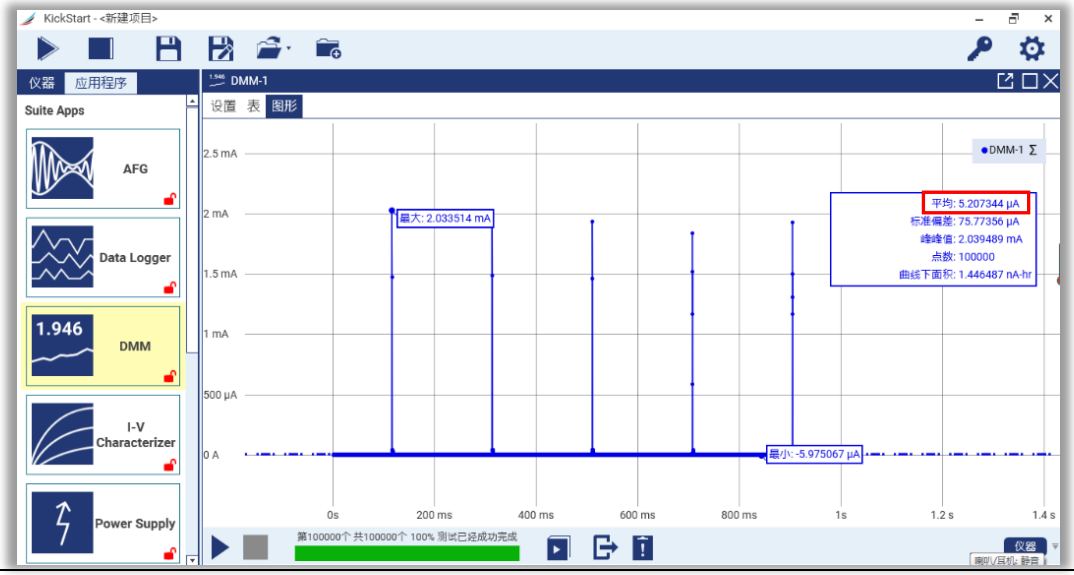
6.3.3

以图形的形式，查看功耗数据

放大电流波形，可以看到，低功耗控制周期为 **100ms**

6.3.4	**以图形的形式，查看功耗数据** 放大电流波形，可以看到，在低功耗模式下，**MEC** 电极工作时的电流波形 
6.4	**修改低功耗控制周期**

6.4.1	在"**qe_touch_sample.c**"中，通过修改以下两个宏定义，修改控制周期，该为 **200000**，即 **200ms**。
	修改低功耗控制周期后，可通过电流表查看电流波形以及功耗数据。
	`/* LPT cycle = 100000[microseconds] (200 microseconds) */`
	`#define WAKEUP_LPT_PERIOD (200000)`
	`/* LPT compare = 100000[microseconds] (200 microseconds) */`
	`#define WAKEUP_LPT_PERIOD_STANDBY (200000)`

6.4.2	**启动测量** **100000** 个测试数据的平均功耗为：**5.2uA** 

END OF SECTION