Freescale Semiconductor
Application Note

Document Number:AN4578
Rev. 0, 8/2012

Integrating Touch-Sensing Software (TSS) on TWR-S08DC-PT60

by: Xianhu Gao
Automotive and Industrial Solutions Group

1 Introduction

The Touch-Sensing Software (TSS) library enables capacitive sensing for all Freescale S08 and ColdFire_ V1, ColdFire+ and ARM®Cortex™-M4 based family microcontroller units (MCUs), providing the common touch sense decoding structures such as keypad, rotary, and slider. It is implemented in software-layered architecture to enable easy integration into the application code and migration to other Freescale MCUs and customer customization.

TWR-S08DC-PT60 demonstrates the capability of MC9S08PT60 targeted for industrial and home appliance applications. It can function as a standalone, low-cost platform for the evaluation of MC9S08PT60 devices. TWR_S08DC_PT60_LABS is the demo code for TWR-S08DC-PT60, in which all the significant features of PT60 are shown. In the code, the touch sense pad acts as a switch keypad to change the system mode and it is controlled by the TSI module independently.

This application note shows how to integrate the TSS into the TWR-S08DC-PT60 labs project, making the user quickly understand the method to integrate the TSS library.

Contents

1.1 TWR-S08DC-PT60

The low-cost TWR-S08DC-PT60 daughter card, a standalone demo board with onboard debugger, is designed to demonstrate the capabilities of the MC9S08P family. It comes preprogrammed with a potentiometer demo, accelerometer demo with orientation/shake/tap/transient modes, flash and EEPROM demo, and a BDM debugger demo. It enables quick and cost-effective product evaluation and application development, and can be used in standalone mode or mounted to the TWR-S08UNIV module to gain access to the full breadth of Freescale's Tower ecosystem.

The demo labs can be found at: http://www.freescale.com/TWR-S08DC-PT60

1.2 Touch-Sensing Software

Freescale's fourth-generation Xtrinsic Touch-Sensing Software Suite (TSS) 2.6, is innovative touch-sensing software that adds value to targeted Freescale Silicon. The free downloading software, in addition to its previous version TSS 2.5, supports a larger MCU portfolio including the HCS08 PT family and two 90 nm families: ARM Cortex-M4 Kinetis and ColdFire+.

The TSSSW can be found at: http://www.freescale.com/TSS

1.3 CodeWarrior

CodeWarrior Development Studio is a complete integrated Development Environment (IDE) that provides a highly visual and automated framework to accelerate the development of the most complex embedded applications.

The latest version of CodeWarrior is CW10.2; it can be found at:

http://www.freescale.com/CodeWarrior

2 Integrating TSS

First of all, download and install TSS and CW using the links given in Touch-Sensing Software and CodeWarrior. The following subsections discuss the steps required to integrate the TSS library into an existing project, initialize and configure it.

2.1 Integrate the TSS into a CW10.2 project

The following steps must be followed to integrate TSS to project TWR_S08PT60_LABS:

1. Choose Start > Programs > Freescale CodeWarrior > CW for MCU v10.2 > CodeWarrior.
2. Select a work space with the selected project in the work space launcher window or create a new project in a work space. Figure 1 is the initial CW window.

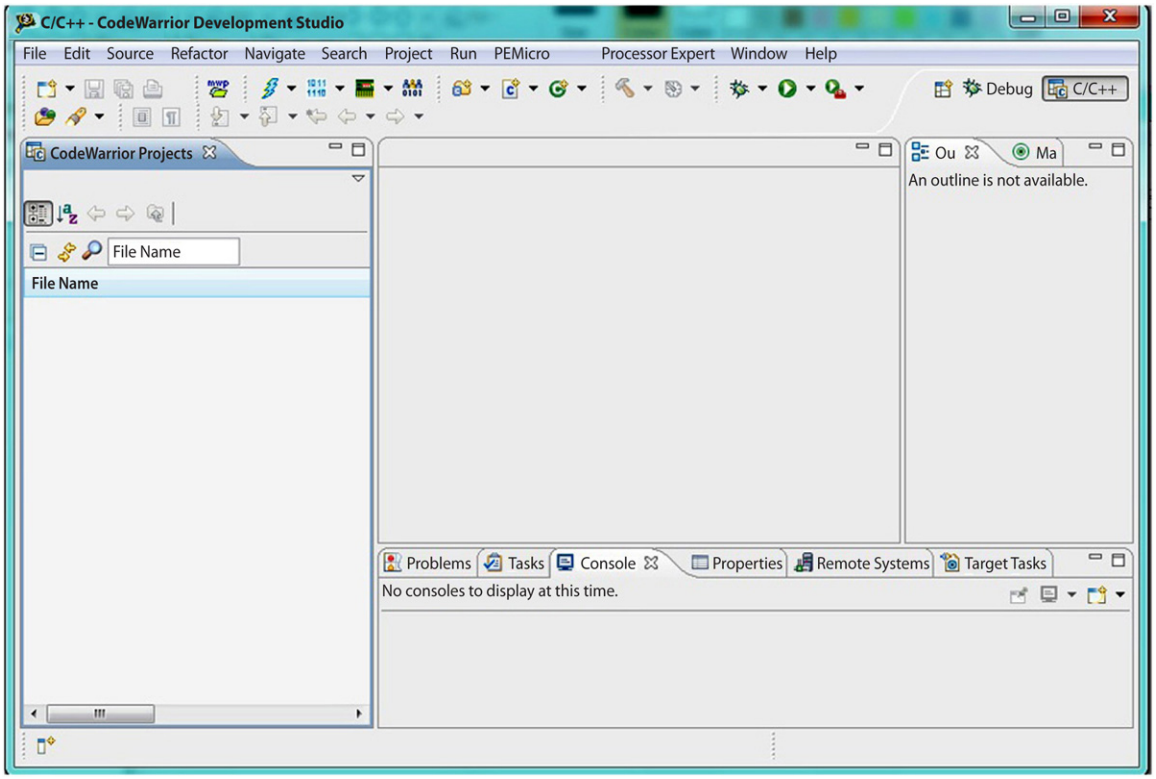

Figure 1. New project

3. Choose File > Import > Existing Projects into Workspace, and then click Next.

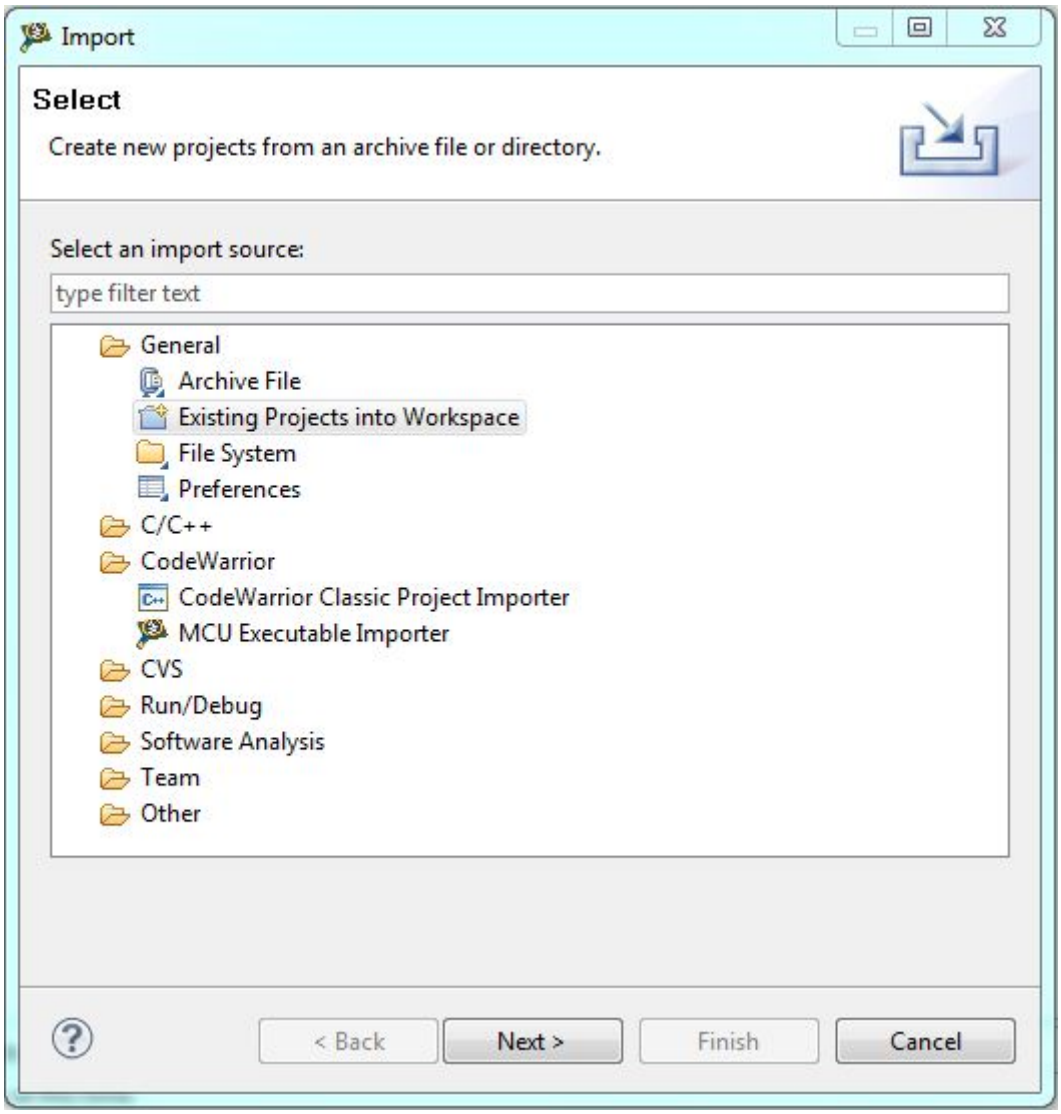

Figure 2. Import project

4. Click Browse to select the project directory and then click Finish.

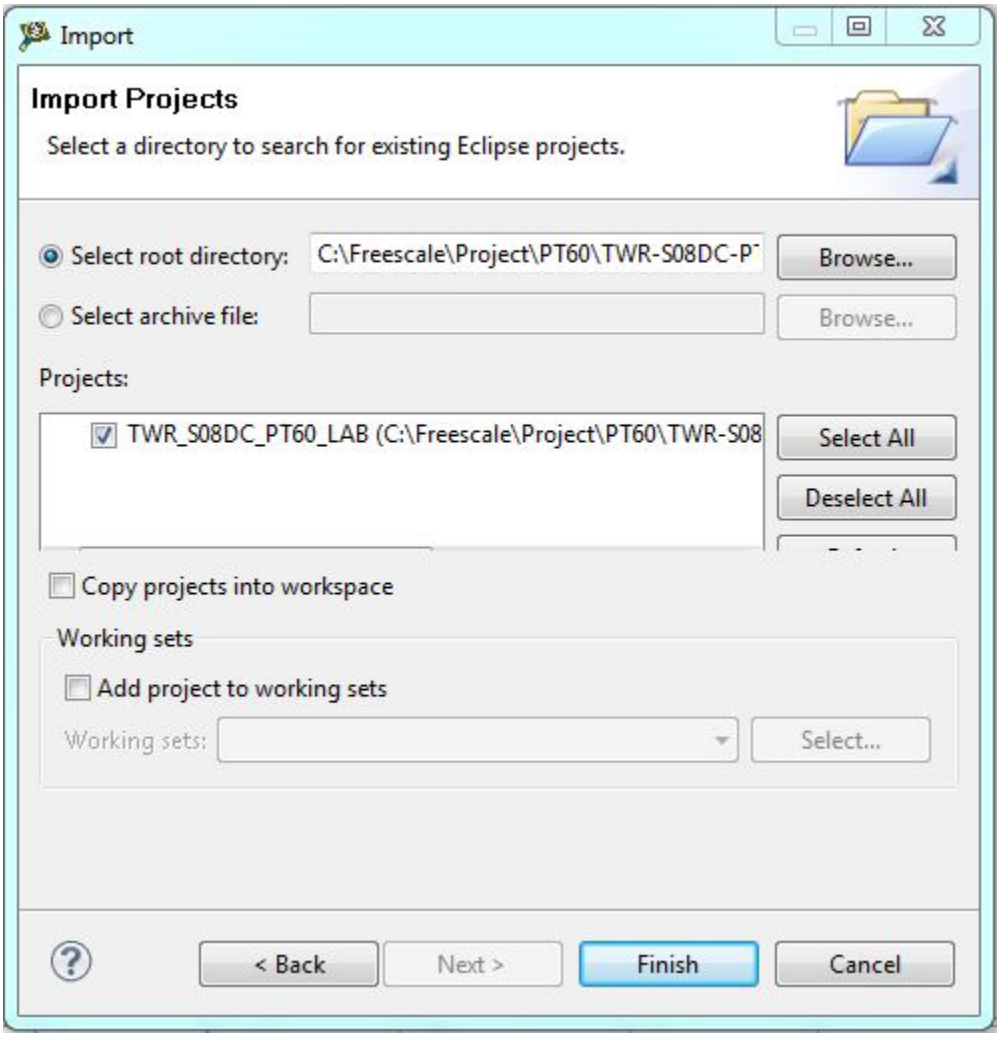

Figure 3. Select project

5. The CodeWarrior Projects window is supposed to display the project structure as the example shown in Figure 4.

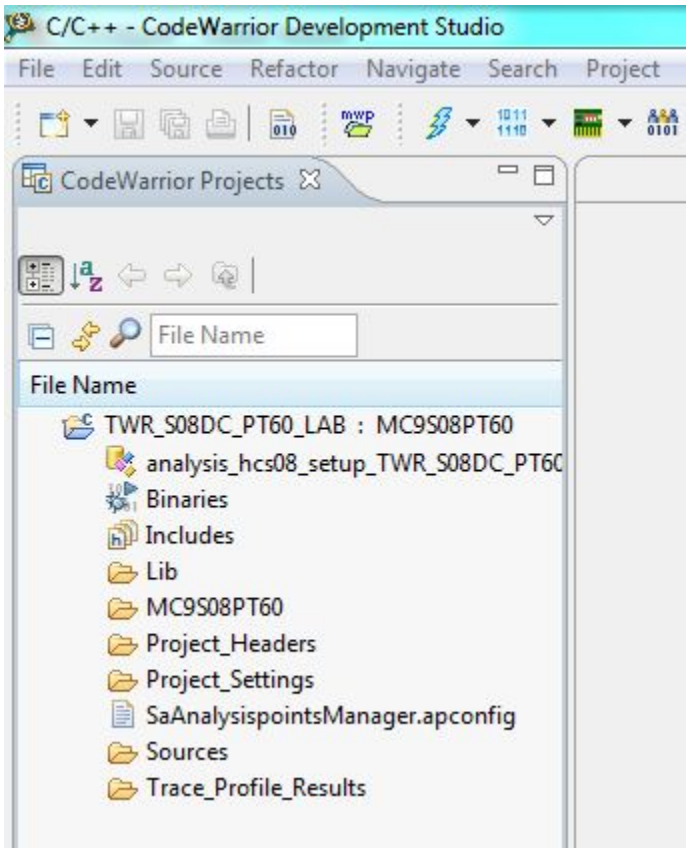

Figure 4. Project structure

6. When adding TSS, create some new folders pointing to TSS files. Choose File > New > Folder, then enter a name for the folder and click Finish. In this application, folders TSS, App_Init, Events, Module_id are added.

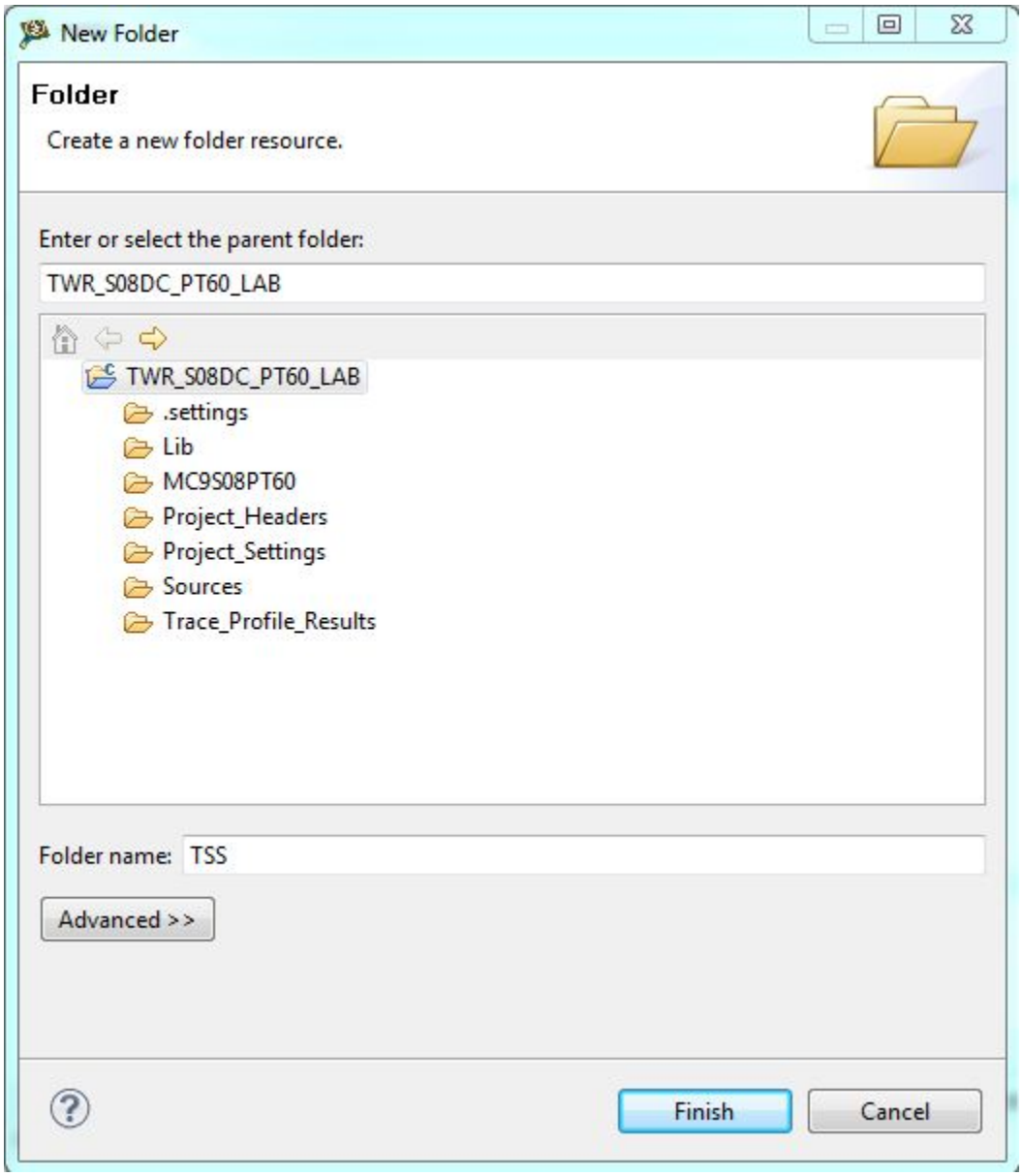

Figure 5. Add new folder

7. The application project that is using the TSS library makes only a reference to the TSS files. The user can copy the library files into the application project directory or can leave them in the installation folder. Leaving the library files in the installation folder enables the user to upgrade the project to newer TSS library versions more easily in the future. In this application note, the associated TSS files are copied into the existing project directory, so that the user is free to move or modify the project.

   Enter the TSS installation folder, open Freescale TSS 2.6\lib\shared, and copy the files listed in the following table, into the existing project directory TWR_S08DC_PT60_LAB_TSS\Sources\TSS:

Table 1. TSS file functions

Source files	Purpose
TSS_Sensor.c	Contains functions to perform the sensing to the electrodes and set the status for each electrode

Table continues on the next page...

Table 1. TSS file functions (continued)

Source files	Purpose
TSS_Sensor.h	Contains the function prototypes, constants, variables and macros for the sensing of electrodes
TSS_SensorTSIL.c	Suitable for low-end devices with TSI module like PT60, no low-power wake-up mode
TSS_SensorTSIL.h	Contains the function prototypes, macros and constants to the TSI module of PT60
TSS_Timer.h	Contains the function prototypes, constants, variables and macros for control and configuration of the HW timer
TSS_API.h	Defines the structs, constants, Types and registers of the TSS library
TSS_DataTypes.h	Defines the structs and constants of the TSS library used by the User Application Level and also internally in the library
TSS_GPIO.h	Defines Macros and constants to control the GPIOs
TSS_StatusCodes.h	Defines the Return Status Codes used by the TSS Library
TSS_SystemSetupData.c	Creates the variables required that depend on the electrode structure of a particular application. Do not edit this file.
TSS_SystemSetupVal.h	Checks if the application configuration defined in the TSS_SystemSetup.h file is consistent. Do not edit this file.

NOTE

The other files are not needed by the MC9S08PT60 project.

Enter the TSS installation folder, open Freescale TSS 2.6\lib\lib_cw, copy the TSS_S08 library file into the PT60 project's folder: TWR_S08DC_PT60_LAB_TSS\Lib, the key detector module is implemented in the object code integrated inside the library, TSS_S08.lib.

Besides these significant files, some other files must also be copied. Enter the TSS installation folder, open Freescale TSS 2.6\examples\TWRS08PTXX_DEMO\src, copy the files listed in the following table into TWR_S08DC_PT60_LAB_TSS\Sources.

Table 2. Application files

Source files	Purpose
app_init.c	Contains the init functions
app_init.h	Contains function types of RTC, keypad, FreeMASTER, and ports initialization
events.c	This is user's event module, wherein the event handler code can be inserted.
events.h	Contains the user function types of Call Back, Fault, and Initial events.

Table continues on the next page...

Table 2. Application files (continued)

Source files	Purpose
module_id.h	Detects modules and devices on TWR-S08PT60 board
TSS_SystemSetup.h	Customizes electrode structure requirements for specific applications.

Copying and reconfiguring these files will give the user a quick guide from a wider sight of project configuration and saves a lot of time.

8. In the CodeWarrior projects window, right-click the newly created folder to select this item. From the context menu, select Add Files. The Open dialog box appears. Locate the TSS files in the TWR_S08DC_PT60_LAB_TSS\Sources \TSS directory of the project folder, select all the files and click Open. The File and Folder Import dialog box will open.

   Select how files and folders should be imported in the project and click OK.

   In the same way, add files app_init.c and app_init.h to the App_Init folder, add files events.c and events.h to the Events folder, add file module_id.h to the Module_id folder, add file TSS_SystemSetup.h to the Sources folder, and add TSS_S08.lib file to the Lib folder.

9. Set the file path. Choose File > Properties > C/C++ Build > Settings > HCS08 Compiler > Input, then add the new added files path as the format shown in Figure 6.

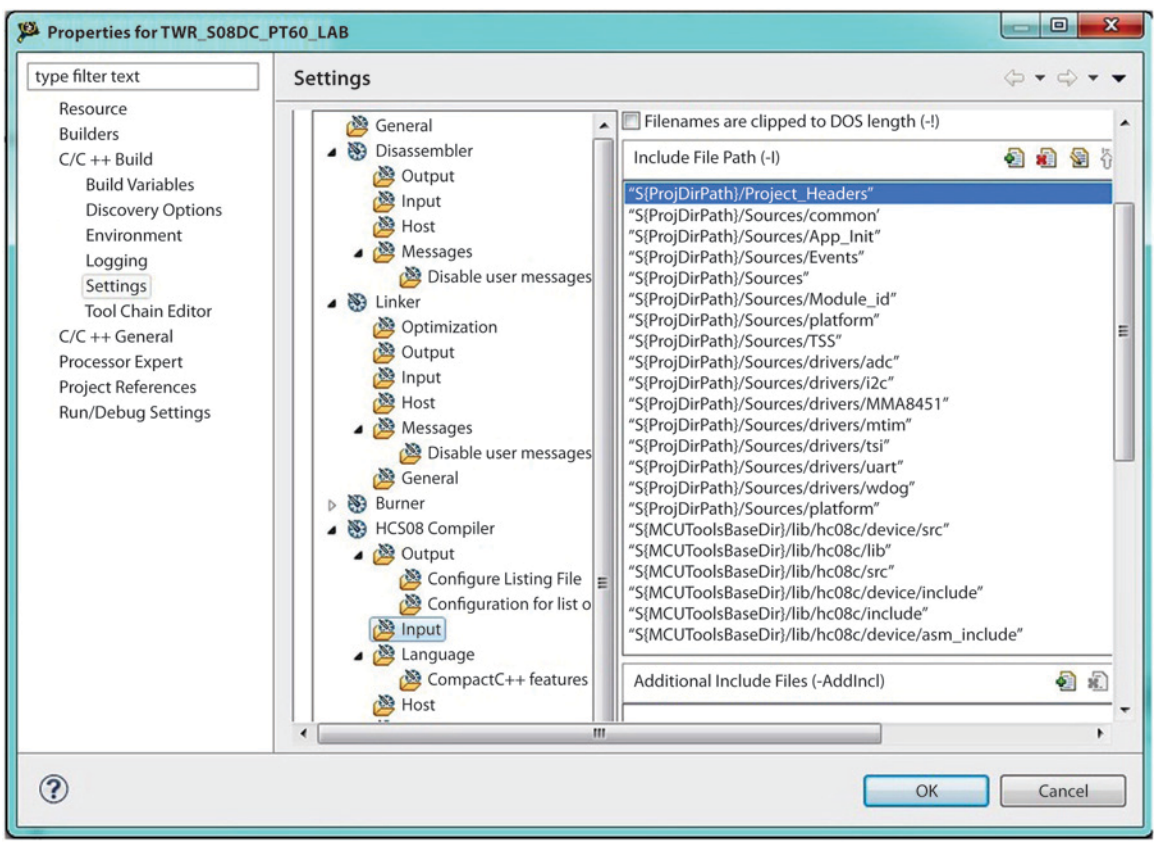

Figure 6. Add head file path

10. Set the link library. Choose File > Properties > C/C++ Build > Settings > Linker > Input, and then add TSS_S08.lib as link library. See Figure 7.

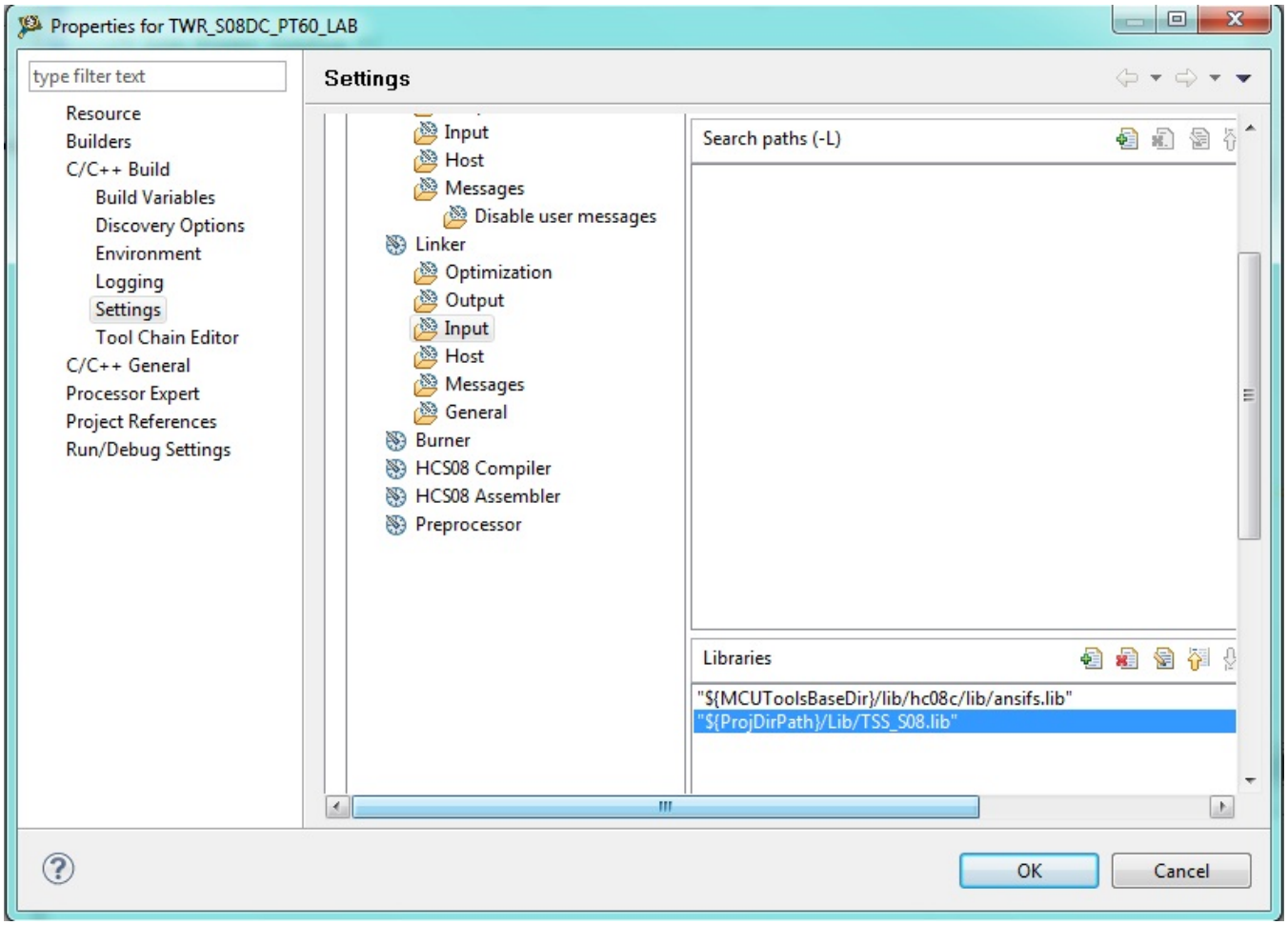

Figure 7. Add link library

2.2 Configuration

After the TSS files have been added into the existing project, if the customers want to call the TSS API function, the configurations must be done first. The following subsections outline steps to let the TSS run.

2.2.1 Head files

To use the API functions and data structure in the TSS, some associated head files should be added as following, in main.h the TSS trigger mode is defined:

```
#include "TSS_API.h"
#include "module_id.h"
#include "app_init.h"
#include "events.h"
#include "main.h"
```

2.2.2 TSS_SystemSetup.h file

The TSS_SystemSetup.h file is used to configure the TSS system. For the sake of convenience, just copy one from the example project and configure it as per the requirement.

```
#define TSS_USE_DCTRACKER          1 /* Enable DC Tracker filtering mechanism  */
#define TSS_USE_IIR_FILTER         1 /* Enables IIR filter */
#define TSS_USE_TRIGGER_FUNCTION   1 /* Enable triggering feature */
#define TSS_USE_TRIGGER_MODE       1 //ALWAYS /*Choose AlWAYS trigger mode */
#define TSS_ONFAULT_CALLBACK    TSS_fOnFault /* The name of a function that matches the
OnFault  callback prototype*/
#define TSS_ONINIT_CALLBACK     TSS_fOnInit  /* The name of a function that matches the
OnInit  callback prototype*/
#define TSS_N_ELECTRODES       1  /*Set the number of electrodes to be used*/
#define TSS_E0_TYPE    TSI_CH12 /* Determines the measurement method for an electrode*/
#define TSS_N_CONTROLS         1  /*Sets the number of controls to be used*/
#define TSS_C0_TYPE            TSS_CT_KEYPAD /* Determines the type of the control*/
#define TSS_C0_ELECTRODES      1  /*Determines the amount of electrodes that compose the
control*/
#define TSS_C0_STRUCTURE       cKey0 /*Indicates the name of the configuration and status
structure of the control*/
#define TSS_C0_CALLBACK        KEY1_Processing /*The name of a valid function that matches
the callback prototype*/
#define TSS_TSI_RESOLUTION           11 /* Defines resolution of TSI in bits for auto
calibration*/
#define TSS_TSI_EXTCHRG_LOW_LIMIT    0  /* Defines low limit of EXTCHRG for TSI auto
calibration*/
#define TSS_TSI_EXTCHRG_HIGH_LIMIT   7  /* Defines high limit of EXTCHRG for TSI auto
calibration*/
#define TSS_TSI_PS_LOW_LIMIT         0  /* Defines low limit of PS for TSI auto calibration*/
#define TSS_TSI_PS_HIGH_LIMIT        7  /* Defines high limit of PS for TSI auto calibration*/
```

2.2.3 Call Back function

The Call Back function is called by the decoder module if an event occurs and the callback function is enabled. It is defined as a macro TSS_C0_CALLBACK in TSS_SystemSetup.h file. In this application, the event function name is KEY1_Processing. This Call Back function is defined by the user and acts just as a hardware interrupt. It can be placed in event.h file, but in this application, it is placed in the main.c file. KEY1_Processing() function can not only detect the touch event but can also distinguish whether it is a touch action or a release action.

```
void KEY1_Processing( void )
{
    /* Write your code here ... */
    UINT8 u8Event; /* 8 bits local variable used to store the event information *
    while (!TSS_KEYPAD_BUFFER_EMPTY(cKey0))  /* While unread events are in the buffer */
    {
        /* Read the buffer and store the event in the u8Event variable */
        TSS_KEYPAD_BUFFER_READ(u8Event,cKey0);
        if (u8Event & 0x80)                      /* If was a release event */
            u8Event = (UINT8) (u8Event & 0x0F);  /* Remove the release flag */
        else
        {                                        /* If was a touch event */
            if (u8Event == 0x00)
            {
               LED_ELECTROD_PTG0_Toggle();  /* change the  led D14 state on board */
                if(LedElectrodOnFlag == 0)
                    LedElectrodOnFlag = 1;   /* set the global flag for the lab */
                else
                     LedElectrodOnFlag = 0;
             }
         }
     }
}
```

2.2.4 TSS and hardware initial

In this application's main() function, the following initial code must be added into the existing S08DC-PT60 project:

```
/* Init HW, replace the function InitPorts() in TSS Lib */
LED_ELECTROD_PTG0_Init();  //  PORT_PTGOE_PTGOE0 = 1, initial the LED D14 on board
/* Default TSS init */
TSS_Init_Keypad0();
```

The TSS will be initialized by calling the function TSS_Init_Keypad0() in main function and this function is located at app_init.c file. Use the following code to set the function.

```
void TSS_Init_Keypad0(void)
{
    UINT8 lcv;
    #if ((TSS_USE_TRIGGER_MODE == SW) || (TSS_USE_TRIGGER_MODE == AUTO))
        TSS_RTCStop();
    #endif
/* Delay For Signal Stabilization */
DelayMS(300);
/* Initializes the TSS */
(void)TSS_Init();
/* Set Number of Samples */
(void)TSS_SetSystemConfig(System_NSamples_Register, 0x08);
/* Sets the Sensitivity value for each electrode */
(void)TSS_SetSystemConfig(System_Sensitivity_Register + lcv, 0x40);
 /* Enablers Settings */
(void)TSS_SetSystemConfig(System_ElectrodeEnablers_Register + 0u, 0x01);
/* Low Power Config */
//(void) TSS_SetSystemConfig(System_LowPowerScanPeriod_Register, 0x08);
//(void) TSS_SetSystemConfig(System_LowPowerElectrode_Register, 0u);
//(void) TSS_SetSystemConfig(System_LowPowerElectrodeSensitivity_Register, 0x40);
/* Auto Trigger Config */
#if (TSS_USE_TRIGGER_MODE == AUTO)
    (void) TSS_SetSystemConfig(System_SystemTrigger_Register, TSS_TRIGGER_MODE_AUTO);
#elif (TSS_USE_TRIGGER_MODE == ALWAYS)
     (void) TSS_SetSystemConfig(System_SystemTrigger_Register, TSS_TRIGGER_MODE_ALWAYS);
#elif (TSS_USE_TRIGGER_MODE == SW)
     (void) TSS_SetSystemConfig(System_SystemTrigger_Register, TSS_TRIGGER_MODE_SW);
#endif

/* Configure the TSS Keyapd Control to report the touch and release events */

(void)TSS_SetKeypadConfig(cKey0.ControlId,Keypad_Events_Register,
(TSS_KEYPAD_TOUCH_EVENT_EN_MASK | TSS_KEYPAD_RELEASE_EVENT_EN_MASK));
/* Enables Callback function. Enables the control */
(void)TSS_SetKeypadConfig(cKey0.ControlId,Keypad_ControlConfig_Register,
(TSS_KEYPAD_CALLBACK_EN_MASK TSS_KEYPAD_CONTROL_EN_MASK));
#if TSS_USE_DCTRACKER
         /* Enables the TSS. Enables the DC Tracking feature. Default DC Tracking value is
0x64 */
         (void)TSS_SetSystemConfig(System_SystemConfig_Register,(TSS_SYSTEM_EN_MASK |
         TSS_DC_TRACKER_EN_MASK));
    #else
         /* Enables the TSS */
         (void)TSS_SetSystemConfig(System_SystemConfig_Register,(TSS_SYSTEM_EN_MASK));
    #endif
    #if ((TSS_USE_TRIGGER_MODE == SW) || (TSS_USE_TRIGGER_MODE == AUTO))
         TSS_RTCInit();
    #endif
}
```

In the original app_init.c file copied from the TWRS08PTXX_DEMOTSS project in Freescale TSS 2.6\examples \TWRS08PTXX_DEMOTSS, TSS_Init_Keypad1(), TSS_Init_Rotary(), FreeMASTER_x and MODULE_ID_x functions are also initialized, which will offer a good GUI interface and TSI pad operation and identification, but they are not used in this application, so, the user can just remove them. The head file module_id.h must not be removed because it contains a lot of data structures.

2.2.5 TSS_Task()

This function must be called periodically by the user application to provide CPU time to the TSS library. All electrodes are processed during a single execution of this function, but the measured data are evaluated after at least two executions. The process status is reported by the return value.

In this application, this function is called periodically in a timer interrupt:

```
interrupt VectorNumber_Vmtim1 void  Mtim1_ISR(void)
{
static UINT8 counter;
    if(MTIM1_SC_TOF)  // clear the flag
    MTIM1_SC_TOF = 0;
    if(counter++ == 10)   // update the baseline and filter
    {
        counter = 0;
        if (TSS_Task() == TSS_STATUS_OK){};   // execute the fun periodically
    }
}
```

2.2.6 Interrupt management

The TSS leaves interrupts enabled while taking electrode measurements.

The electrode measurement routine may get interrupted by a user application interrupt that causes the sampled value to be invalid. All user interrupt handlers must register themselves with the TSS library by calling the TSS_SET_SAMPLE_INTERRUPTED() macro.

```
interrupt  VectorNumber_Vkbi1 void  MMA8451_Int_ISR(void)
{
    KBI1_SC_KBACK = 1;
    TSS_SET_SAMPLE_INTERRUPTED();     // tell the TSS in int_ISR now
}
```

3 Conclusion

Until now all the necessary steps and changes have been finished including IAR setting and code porting.

The old version of TWR-S08DC-PT60 demo code controls the TSI module directly, but the TSS offers an advanced solution which is more powerful and configurable. With the help of TSS, the customers can easily control more electrode TSI pads.

This application note shows the costumers how to integrate the TSS into an existing project in a stepwise manner. Although it is based on the S08 Core device, it is a good example for customers to learn the TSS integration in other platforms including ColdFire V1, ColdFire+, and ARM Cortex-M4 projects.

4 References

The following documents are available at http://www.freescale.com for further reference.

- TSSUG : TSSUG, Touch Sensing Software Users Guide
- TSSAPIRM : TSSAPIRM, Touch Sensing Software API Reference Manual
- AN4330: Writing Touch Sensing Software Using TSI Module

How to Reach Us:

Home Page:
www.freescale.com

Web Support:
http://www.freescale.com/support

USA/Europe or Locations Not Listed:
Freescale Semiconductor
Technical Information Center, EL516
2100 East Elliot Road
Tempe, Arizona 85284
+1-800-521-6274 or +1-480-768-2130
www.freescale.com/support

Europe, Middle East, and Africa:
Freescale Halbleiter Deutschland GmbH
Technical Information Center
Schatzbogen 7
81829 Muenchen, Germany
+44 1296 380 456 (English)
+46 8 52200080 (English)
+49 89 92103 559 (German)
+33 1 69 35 48 48 (French)
www.freescale.com/support

Japan:
Freescale Semiconductor Japan Ltd.
Headquarters
ARCO Tower 15F
1-8-1, Shimo-Meguro, Meguro-ku,
Tokyo 153-0064
Japan
0120 191014 or +81 3 5437 9125
support.japan@freescale.com

Asia/Pacific:
Freescale Semiconductor China Ltd.
Exchange Building 23F
No. 118 Jianguo Road
Chaoyang District
Beijing 100022
China
+86 10 5879 8000
support.asia@freescale.com

Document Number: AN4578
Rev. 0, 8/2012