Please note that Cypress is an Infineon Technologies Company.
The document following this cover page is marked as “Cypress” document as this is the company that originally developed the product. Please note that Infineon will continue to offer the product to new and existing customers as part of the Infineon product portfolio.

Continuity of document content
The fact that Infineon offers the following product as part of the Infineon product portfolio does not lead to any changes to this document. Future revisions will occur when appropriate, and any changes will be set out on the document history page.

Continuity of ordering part numbers
Infineon continues to support existing part numbers. Please continue to use the ordering part numbers listed in the datasheet for ordering.

CE222827 – PSoC 6 MCU: CapSense Tuner

Objective

This example demonstrates how to implement Tuner GUI interface for CapSense® design using UART and I²C interfaces in PSoC® 6 MCU devices.

Overview

There are two CapSense projects with this code example. Code example 1 demonstrates tuning using the I²C interface; Code example 2 demonstrates tuning using the UART interface. This project features the CapSense Component configured with two CSX-based buttons and a CSD-based linear slider. Tuner GUI is used to monitor CapSense scanning data.

Key differences between I²C- and UART-based tuner interfaces are the following:

- I²C-based tuning allows real-time tuning with read and write (update parameter values) access to CapSense Component parameters; UART-based tuner interface provides only read-only access and it can be used for real-time sensor data monitoring.
- UART transmission and sensor scan are executed sequentially. Therefore, data transfer does not overlap with sensor scan.
- UART sends data after every scan so the graph displays all scanned results. However, with I²C, scan and data transfer execution are asynchronous to each other.
- UART can produce more accurate SNR measurement results because it captures every scan sample.

Requirements

Tool: PSoC Creator™ 4.2; Peripheral Driver Library (PDL) 3.0.1

Programming Language: C (Arm® GCC 5.4-2016-q2-update and Arm MDK 5.22)

Associated Parts: All PSoC 6 MCU parts

Related Hardware: CY8CKIT-062-BLE PSoC 6 BLE Pioneer Kit

Hardware Setup

This example uses the kit's default configuration. Refer to the kit guide to ensure that the kit is configured correctly.

Software Setup

None.

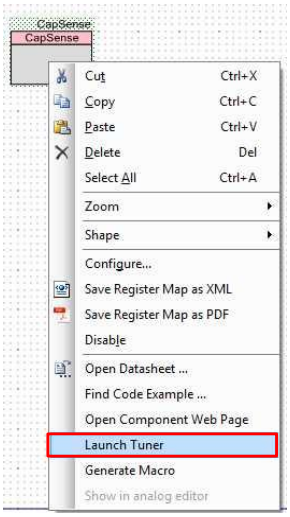
Operation

Plug the CY8CKIT-062-BLE kit board into your computer's USB port and follow the instructions.

Code example 1: CapSense Tuning over I²C Interface

1. **Build project:** Build the project "CapSense_Tuner_EZI2C" and program it into the PSoC 6 MCU device. Choose **Debug** > **Program**. For more information on device programming, see PSoC Creator Help. Flash for both CPUs is programmed in a single program operation.
2. **Launch tuner GUI:** Right-click and select **Launch Tuner** from the CapSense instance context menu, as Figure 1 shows.

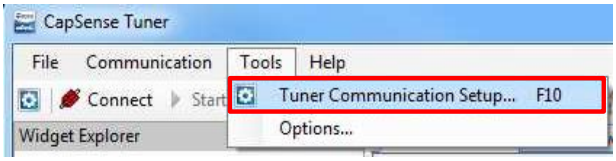
Figure 1. Launch Tuner



3. **Establish I²C communication:** To establish communication between the tuner and a target device, configure the tuner communication parameters to match the I2C Component parameters.

   a. Open the Tuner Communication Setup dialog by selecting **Tools** > **Tuner Communication Setup…** in the menu, or clicking **Tuner Communication Setup**, as Figure 2 shows.

Figure 2. Tuner: Communication Setup

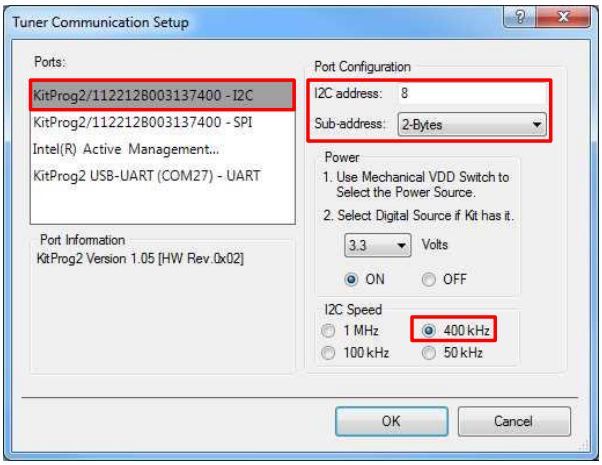


   b. Select the appropriate I2C communication device, which is KitProg2 (or MiniProg3) and set the following parameters:

   - I²C Address: 8 (or the address set in the EzI2C Component configuration wizard)
   - Sub-address: 2 bytes
   - I²C Speed: 400 kHz (or the speed set in the Component configuration wizard)

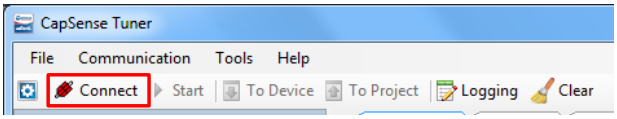
   Note: The I²C address, Sub-address, and I2C speed fields in the Tuner communication setup must be identical to the Primary slave address, Sub-address size, and Data Rate parameters in the EzI2C Component Configuration. Sub-address must be set to 2 bytes in both places.

Figure 3. Tuner Communication Parameters Selection



c. Click **Connect** to establish connection as Figure 4 shows. If the connection is set up correctly, the “Start” button turns active and, status bar indicates the Bridge status as “Connected” and shows the comunication parameters.

Figure 4. Tuner: Connect Option



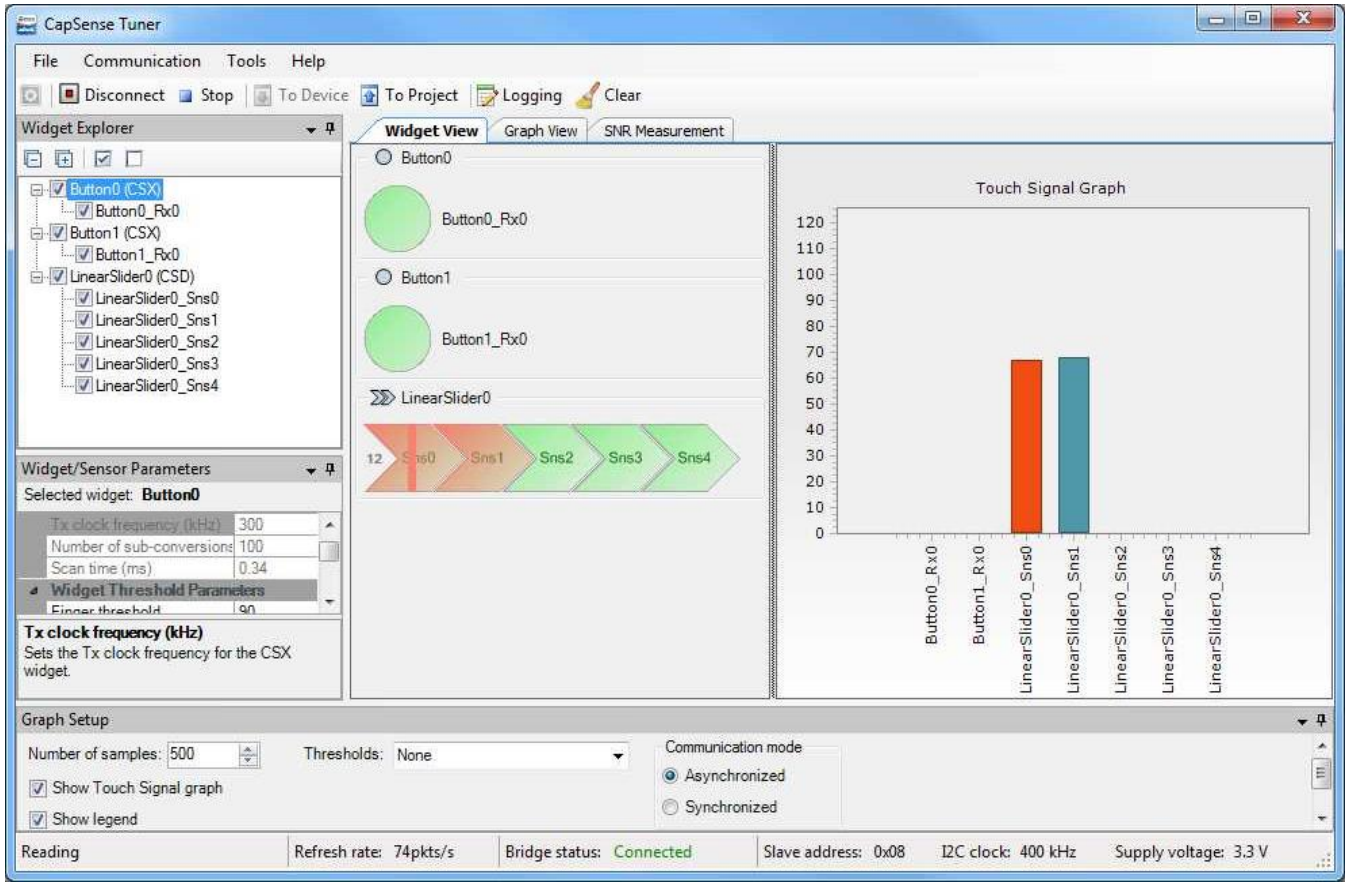
d. Click **Start** to start data streaming from the device. The tuner GUI displays data from the sensor in the widget view and Graph view tabs. See the CapSense Component datasheet for detailed information of tuner GUI. However, the following sections document a few quick examples on Tuner GUI usage.

4. **Monitor data in Tuner GUI:** The application consists of the following tabs:

a. **Widget View:** Displays the widgets, their touch status, and the touch signal bar graph. Widget sensors are highlighted in red color when the device reports their touch status as active. Some additional features are available depending on the widget type.

Touch Signal Graph: The Widget view also displays a Touch Signal Graph when the **Display Touch Signal graph** option is selected in the **Graph Setup Pane**. This graph contains a touch signal level for each sensor selected in the Widget Explorer Pane, as Figure 5 shows.

Figure 5. Tuner: Widget View



b. **Graph View:** Displays graphs for selected sensors in the Widget Explorer Pane as Figure 6 shows. The following charts are available:

– **Sensor Data graph** – Displays raw counts and baseline. Use the checkboxes on the right to select the series to be displayed:

▪ Raw counts and baseline

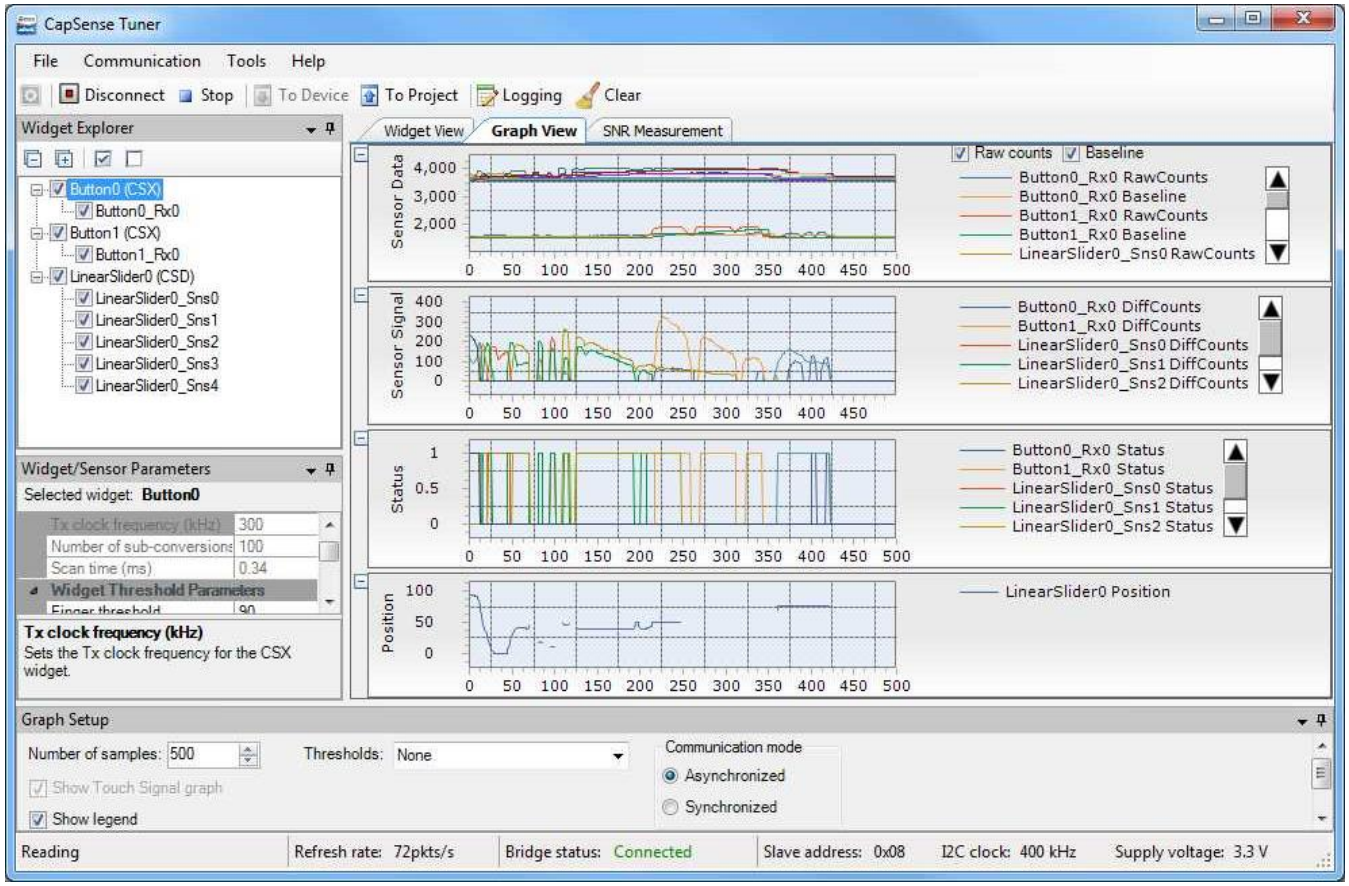
- Raw counts
- Baseline

– **Sensor Signal graph** – Displays a signal difference

– **Status graph** – Displays the sensor status (Touch/No Touch)

– **Position graph** – Displays touch positions for the Linear Slider, Radial Slider, and Touchpad widgets

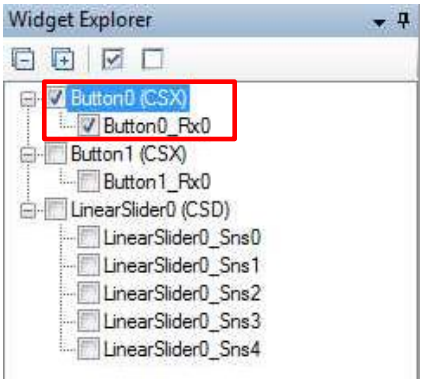
Figure 6. Tuner: Graph View



5. **Manual Parameter Tuning:**

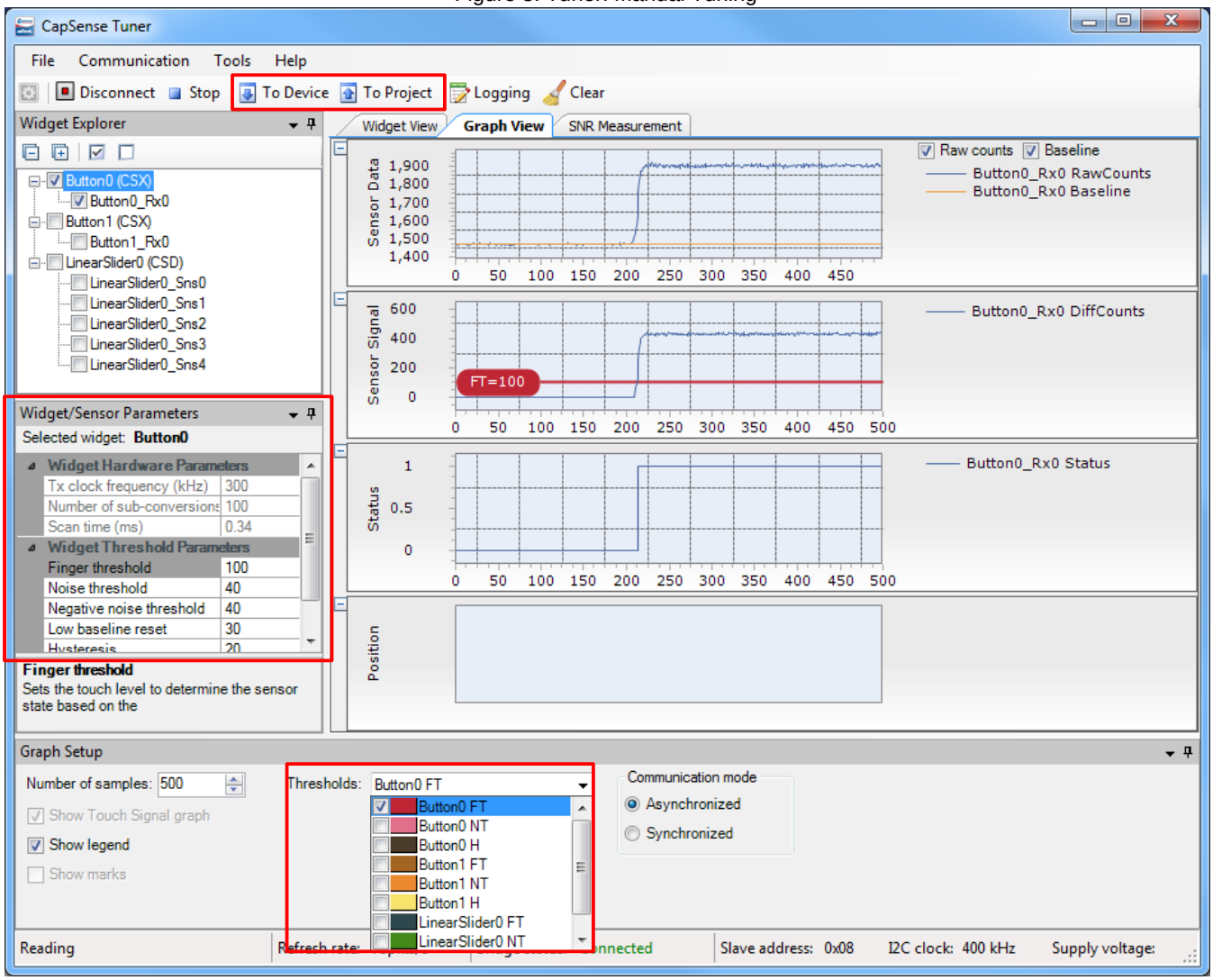
   a. Select one of the widget or sensors to display the parameters associated with sensor/widget.

Figure 7. Tuner: Sensor Selection



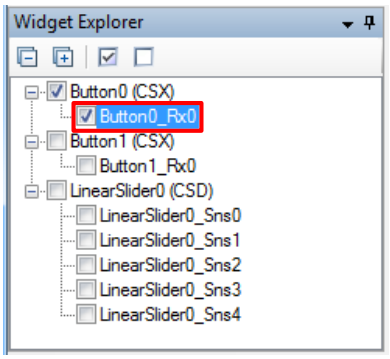
b. Change the parameter value and press **Enter**. The new value is highlighted in bold. To apply the new value to the device, click **To Device**. The new value is applied to device and respective change is displayed in the device behavior. The **To Project** button applies the attest parameter to the CapSense Component Customizer in the project. The changes are applied after the Tuner is closed and the Customizer is opened. Changes to widget / sensor parameters made in the Tuner GUI are not automatically updated to the PSoC Creator project, unless specially saved.

Figure 8. Tuner: Manual Tuning



6. **SNR Measurement** – Provides the SNR measurement functionality.
   a. Select the sensor for SNR measurement.

Figure 9. Tuner: Select sensor for SNR Measurement

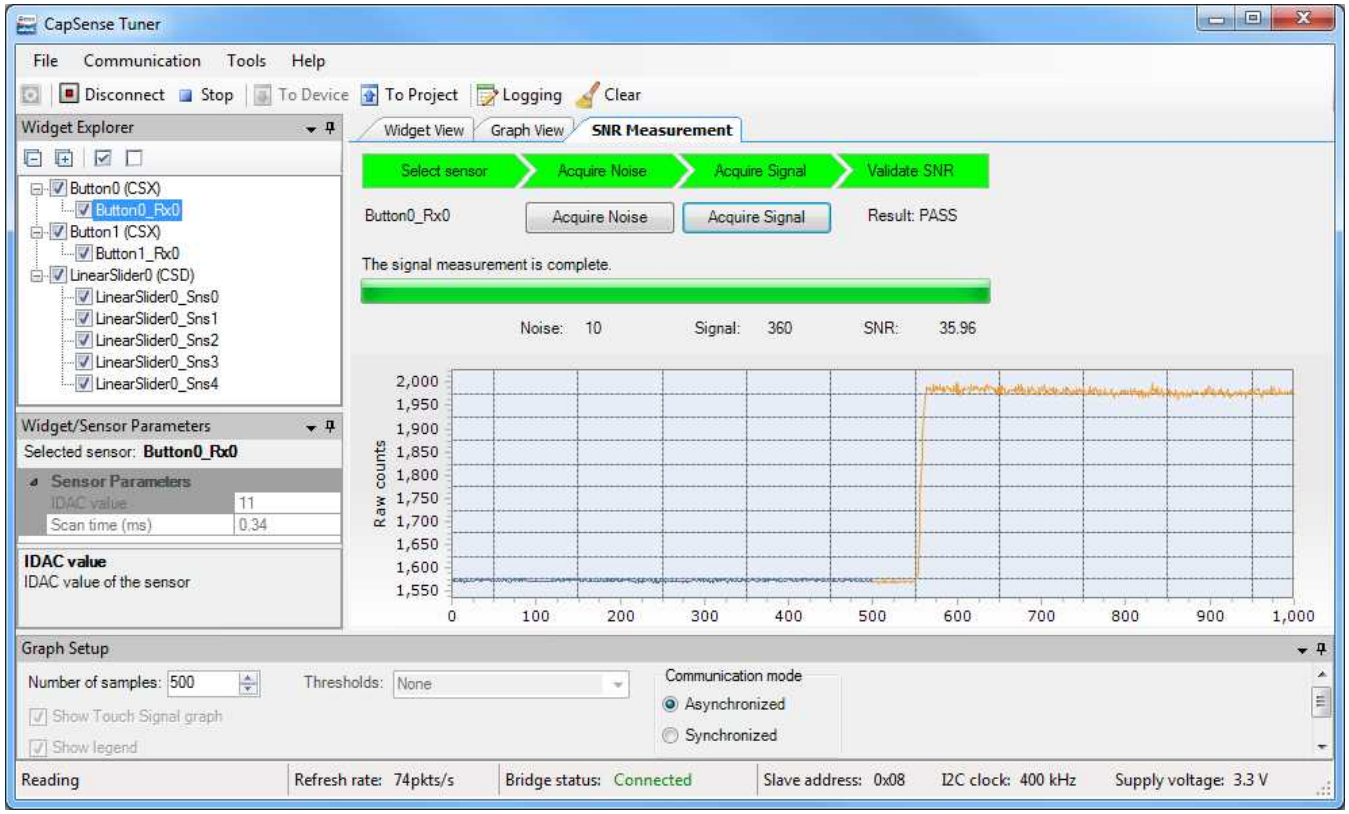


b. Click on **Acquire Noise** and wait for noise measurement to complete (do not touch any sensor during noise measurement).

c. Touch the selected sensor and click **Acquire Signal**. (Do not release the finger until signal measurement is completed).

d. After noise and signal are measured, tuner GUI displays the SNR, as shown in Figure 10.

Figure 10. Tuner: SNR Measurement



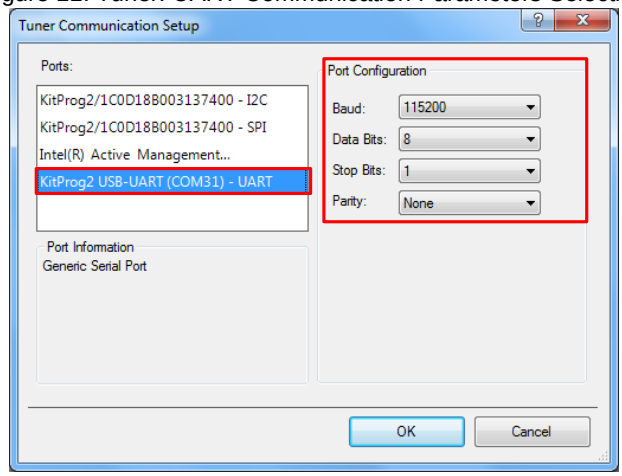
For more details on CapSense tuning, see CapSense Component datasheet and PSoC 4 and PSoC 6 MCU CapSense Design Guide.

Code example 2: CapSense Tuning over UART Interface

1. **Build project:** Build the project “CapSense_Tuner_UART” and program it into the PSoC 6 MCU device. Choose **Debug** > **Program**. For more information on device programming, see PSoC Creator Help. Flash for both CPUs is programmed in a single program operation.
2. **Launch tuner GUI:** Right-click and select **Launch Tuner** from the CapSense instance context menu, as Figure 1 shows.
3. **Establish UART communication:** To establish communication between the tuner and a target device, configure the tuner communication parameters to match the UART Component parameters.
   a. Open the Tuner Communication Setup dialog by selecting **Tools** > **Tuner Communication Setup…** in the menu or clicking **Tuner Communication Setup** as Figure 2 shows.
   b. Select the appropriate UART communication device which is KitProg2 (or MiniProg3) and set the following parameters:
      - Baud: 115200
      - Data Bits: 8
      - Stop Bits: 1
      - Parity: None

      Note: The parameters in the Tuner Communication Setup must be identical to the parameters in the UART SCB Component Configure dialog.

Figure 11. Tuner: UART Communication Parameters Selection



   c. Click **Connect** to establish connection as Figure 4 shows. If the connection is set up correctly, the **Start** button turns active and, the status bar indicates the Bridge status as “Connected”.
   d. Click **Start** to start data streaming from the device. The tuner GUI displays the data from sensor in the widget view and Graph view tabs. See the CapSense Component datasheet for a detailed information on the tuner GUI. The following sections document a few quick examples on Tuner GUI usage.
4. **Monitor data in Tuner GUI:** To monitor CapSense data using the Tuner application, follow Step 4 Monitor Data in Tuner GUI of code example 1.

Design and Implementation

This example features CapSense with two buttons (mutual capacitance) and liner slider(self-capacitance). Tuner GUI is used to visualize CapSense scanning data. I2C or UART is used to establish communication with the Tuner application.

Figure 12 and Figure 13 show the PSoC Creator schematic for this code example.

Figure 12. TopDesign Schematic of CapSense_Tuner_EZI2C

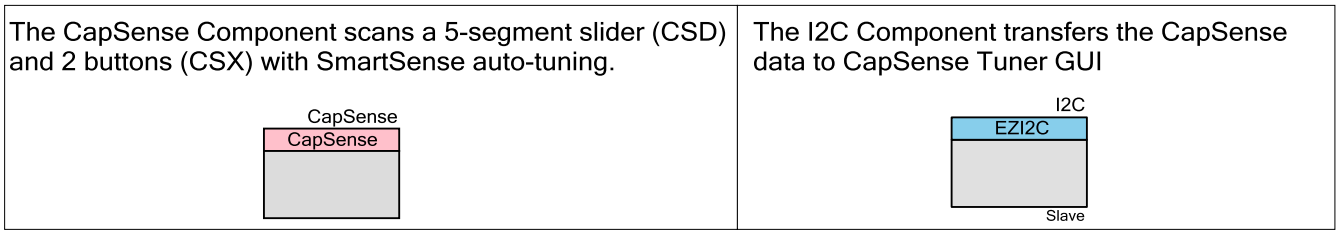
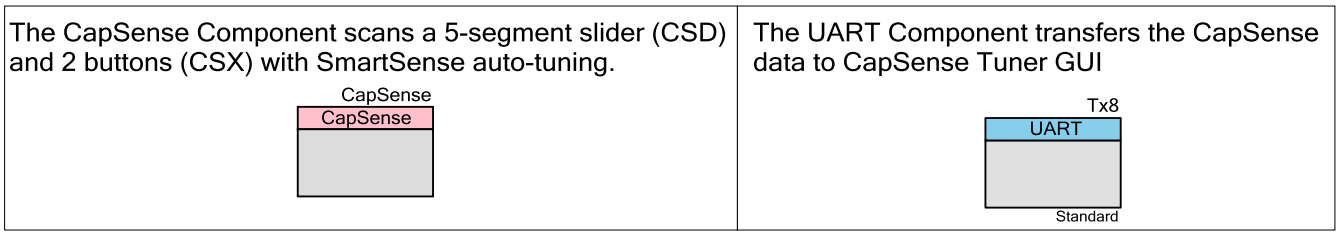


Figure 13. TopDesign Schematic of CapSense_Tuner_UART



Components and Settings

Table 1 lists the PSoC Creator Components used in this example, how they are used in the design, and the non-default settings required so they function as intended.

Table 1. PSoC Creator Components

Component	Instance Name	Purpose	Non-default Settings
CapSense	CapSense	The CapSense Component is configured to scan a 5-segment slider (CSD) and 2 buttons (CSX) with SmartSense auto-tuning	Refer Parameter Settings section
UART	Tx8	To establish communication with the Tuner application	TX/RX Mode: Tx only
EZI2C	I2C	To establish communication with the Tuner application	Data Rate (kbps): 400

For information on the hardware resources used by a Component, see the Component datasheet.

Parameter Settings

Figure 14 and Table 2 show the modified settings for the CapSense Component.

Figure 14. CapSense: Basic Tab

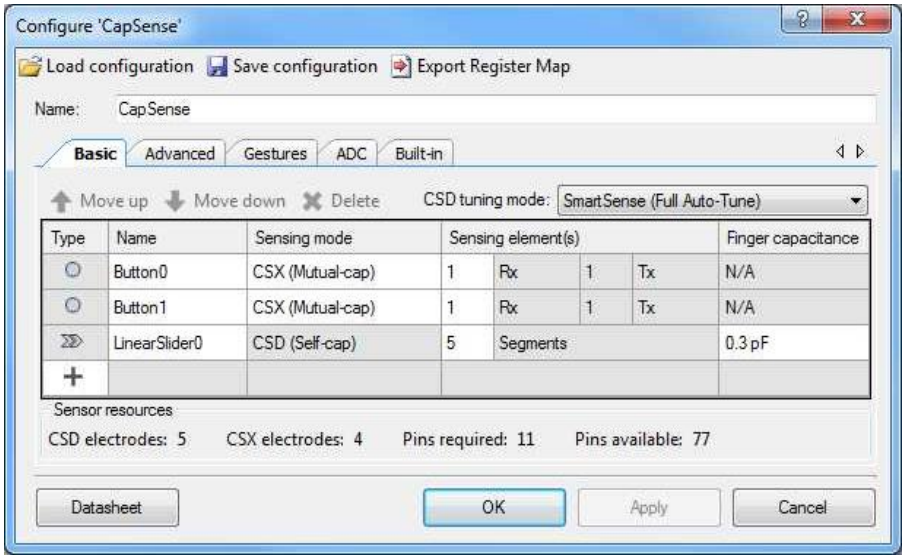


Table 2. CapSense: Advance Tab Non-Default Settings

Tab	Parameter	Value
CSD Settings	Modulator clock frequency (kHz)	6000
	Sense clock source	Auto
CSX Settings	Modulator clock frequency (kHz)	12000
	Sense clock source	Auto
Widget Setting	Button0_Rx0	Dedicated pin
	Button0_Tx	Dedicated pin
	Button1_Rx0	Dedicated pin
	Button0_Tx	Button0_Tx
	LinearSlide0 (for all pins)	Dedicated pin

Figure 15 and Figure 16 shows the configuration for the **Interrupts** tab in the **Design Wide Resources** window. The interrupt is enabled on the CM4, and given a priority of 7. These are the default settings for an interrupt. The ISR is compiled as part of the CM4 code.

Figure 15. Interrupt Assignments for Code Example 1

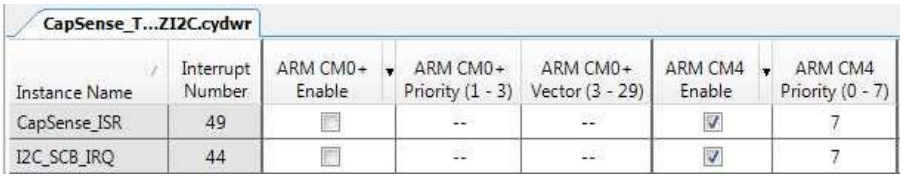
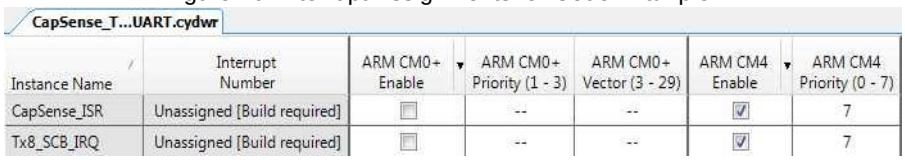


Figure 16. Interrupt Assignments for Code Example 2



CapSense_T...UART.cydwr

Instance Name	Interrupt Number	ARM CM0+ Enable	ARM CM0+ Priority (1 - 3)	ARM CM0+ Vector (3 - 29)	ARM CM4 Enable	ARM CM4 Priority (0 - 7)
CapSense_ISR	Unassigned [Build required]	☐	--	--	☑	7
Tx8_SCB_IRQ	Unassigned [Build required]	☐	--	--	☑	7

Figure 17 and Figure 18 show the pin assignments for the project done through the **Pins** tab in the **Design Wide Resources** window. These assignments are compatible with CY8CKIT-062-BLE.

Figure 17. Pin Assignments for Code Example 1

	Name	Port	Pin	Lock
	\CapSense:CintA\	P7[1]	H10	☑
	\CapSense:CintB\	P7[2]	H8	☑
	\CapSense:Cmod\ (Cmod)	P7[7]	G7	☑
	\CapSense:Rx[0]\ (Button0_Rx0)	P8[1]	F9	☑
	\CapSense:Rx[1]\ (Button1_Rx0)	P8[2]	F8	☑
	\CapSense:Sns[0]\ (LinearSlider0_Sns0)	P8[3]	F7	☑
	\CapSense:Sns[1]\ (LinearSlider0_Sns1)	P8[4]	G6	☑
	\CapSense:Sns[2]\ (LinearSlider0_Sns2)	P8[5]	E9	☑
	\CapSense:Sns[3]\ (LinearSlider0_Sns3)	P8[6]	E8	☑
	\CapSense:Sns[4]\ (LinearSlider0_Sns4)	P8[7]	E7	☑
	\CapSense:Tx\ (Button0_Tx)	P1[0]	G3	☑
	\I2C:scl\	P6[0]	K8	☑
	\I2C:sda\	P6[1]	J8	☑

Figure 18. Pin Assignments for Code Example 2

	Name	Port	Pin	Lock
	\CapSense:CintA\	P7[1]	H10	☑
	\CapSense:CintB\	P7[2]	H8	☑
	\CapSense:Cmod\ (Cmod)	P7[7]	G7	☑
	\CapSense:Rx[0]\ (Button0_Rx0)	P8[1]	F9	☑
	\CapSense:Rx[1]\ (Button1_Rx0)	P8[2]	F8	☑
	\CapSense:Sns[0]\ (LinearSlider0_Sns0)	P8[3]	F7	☑
	\CapSense:Sns[1]\ (LinearSlider0_Sns1)	P8[4]	G6	☑
	\CapSense:Sns[2]\ (LinearSlider0_Sns2)	P8[5]	E9	☑
	\CapSense:Sns[3]\ (LinearSlider0_Sns3)	P8[6]	E8	☑
	\CapSense:Sns[4]\ (LinearSlider0_Sns4)	P8[7]	E7	☑
	\CapSense:Tx\ (Button0_Tx)	P1[0]	G3	☑
	\Tx8:tx\	P5[1]	K6	☑

Reusing This Example

This example is designed for the CY8CKIT-062-BLE Pioneer Kit. To port the design to a different PSoC 6 MCU device and/or kit, change the target device using **Device Selector** and update the pin assignments in the **Design Wide Resources Pins** settings as needed. For single-core PSoC 6 MCU devices, port the code from *main_cm4.c* to *main.c*.

Related Documents

Application Notes	
AN210781 – Getting Started with PSoC 6 MCU with BLE Connectivity	Describes PSoC 6 MCU with BLE Connectivity devices and how to build your first PSoC Creator project
AN215656 – PSoC 6 MCU Dual-Core CPU system Design	Describes the dual-core CPU architecture in PSoC 6 MCU, and shows how to build a simple dual-core design
AN219434 – Importing PSoC Creator Code into an IDE for a PSoC 6 MCU Project	Describes how to import the code generated by PSoC Creator into your preferred IDE
PSoC Creator Component Datasheets	
Pins	Supports connection of hardware resources to physical pins
CapSense	Provides guidelines to use the CapSense component.
UART	Provides asynchronous communications
EZI2C	Supports simplified I2C slave implementation
Device Documentation	
PSoC 6 MCU: PSoC 63 with BLE Datasheet	PSoC 6 MCU: PSoC 63 with BLE Architecture Technical Reference Manual
Development Kit (DVK) Documentation	
CY8CKIT-062-BLE PSoC 6 BLE Pioneer Kit	

Document History

Document Title: CE222827 – PSoC 6 MCU: CapSense Tuner

Document Number: 002-22827

Revision	ECN	Orig. of Change	Submission Date	Description of Change
**	6048847	AJYA	01/30/2018	New code example
*A	6088700	AJYA	03/06/2018	Updated to PSoC Creator 4.2

Worldwide Sales and Design Support

Cypress maintains a worldwide network of offices, solution centers, manufacturer's representatives, and distributors. To find the office closest to you, visit us at Cypress Locations.

Products

Arm® Cortex® Microcontrollers	cypress.com/arm
Automotive	cypress.com/automotive
Clocks & Buffers	cypress.com/clocks
Interface	cypress.com/interface
Internet of Things	cypress.com/iot
Memory	cypress.com/memory
Microcontrollers	cypress.com/mcu
PSoC	cypress.com/psoc
Power Management ICs	cypress.com/pmic
Touch Sensing	cypress.com/touch
USB Controllers	cypress.com/usb
Wireless Connectivity	cypress.com/wireless

PSoC® Solutions

PSoC 1 | PSoC 3 | PSoC 4 | PSoC 5LP | PSoC 6 MCU

Cypress Developer Community

Community Forums | Projects | Videos | Blogs | Training | Components

Technical Support

cypress.com/support

All other trademarks or registered trademarks referenced herein are the property of their respective owners.

Cypress Semiconductor
198 Champion Court
San Jose, CA 95134-1709

© Cypress Semiconductor Corporation, 2018. This document is the property of Cypress Semiconductor Corporation and its subsidiaries, including Spansion LLC ("Cypress"). This document, including any software or firmware included or referenced in this document ("Software"), is owned by Cypress under the intellectual property laws and treaties of the United States and other countries worldwide. Cypress reserves all rights under such laws and treaties and does not, except as specifically stated in this paragraph, grant any license under its patents, copyrights, trademarks, or other intellectual property rights. If the Software is not accompanied by a license agreement and you do not otherwise have a written agreement with Cypress governing the use of the Software, then Cypress hereby grants you a personal, non-exclusive, nontransferable license (without the right to sublicense) (1) under its copyright rights in the Software (a) for Software provided in source code form, to modify and reproduce the Software solely for use with Cypress hardware products, only internally within your organization, and (b) to distribute the Software in binary code form externally to end users (either directly or indirectly through resellers and distributors), solely for use on Cypress hardware product units, and (2) under those claims of Cypress's patents that are infringed by the Software (as provided by Cypress, unmodified) to make, use, distribute, and import the Software solely for use with Cypress hardware products. Any other use, reproduction, modification, translation, or compilation of the Software is prohibited.

TO THE EXTENT PERMITTED BY APPLICABLE LAW, CYPRESS MAKES NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS DOCUMENT OR ANY SOFTWARE OR ACCOMPANYING HARDWARE, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. No computing device can be absolutely secure. Therefore, despite security measures implemented in Cypress hardware or software products, Cypress does not assume any liability arising out of any security breach, such as unauthorized access to or use of a Cypress product. In addition, the products described in these materials may contain design defects or errors known as errata which may cause the product to deviate from published specifications. To the extent permitted by applicable law, Cypress reserves the right to make changes to this document without further notice. Cypress does not assume any liability arising out of the application or use of any product or circuit described in this document. Any information provided in this document, including any sample design information or programming code, is provided only for reference purposes. It is the responsibility of the user of this document to properly design, program, and test the functionality and safety of any application made of this information and any resulting product. Cypress products are not designed, intended, or authorized for use as critical components in systems designed or intended for the operation of weapons, weapons systems, nuclear installations, life-support devices or systems, other medical devices or systems (including resuscitation equipment and surgical implants), pollution control or hazardous substances management, or other uses where the failure of the device or system could cause personal injury, death, or property damage ("Unintended Uses"). A critical component is any component of a device or system whose failure to perform can be reasonably expected to cause the failure of the device or system, or to affect its safety or effectiveness. Cypress is not liable, in whole or in part, and you shall and hereby do release Cypress from any claim, damage, or other liability arising from or related to all Unintended Uses of Cypress products. You shall indemnify and hold Cypress harmless from and against all claims, costs, damages, and other liabilities, including claims for personal injury or death, arising from or related to any Unintended Uses of Cypress products.

Cypress, the Cypress logo, Spansion, the Spansion logo, and combinations thereof, WICED, PSoC, CapSense, EZ-USB, F-RAM, and Traveo are trademarks or registered trademarks of Cypress in the United States and other countries. For a more complete list of Cypress trademarks, visit cypress.com. Other names and brands may be claimed as property of their respective owners.