CE229162 – PSoC 4 CapSense Custom Baseline Update

Objective

This example demonstrates custom methods to decrease baseline update rate in CapSense® Component. This enables implementing features like gesture with very slow baseline update rate.

Requirements

Tool: PSoC® Creator™ 4.3

Programming Language: C (Arm® GCC 5.4.1 and Arm MDK 5.22)

Associated Parts: All PSoC 4 MCU parts

Related Hardware: CY8CKIT-149 PSoC 4100S Plus Prototyping Kit, CY8CKIT-145-40XX PSoC 4000S CapSense Prototyping Kit

Overview

This code example demonstrates how to slow down baseline update rate. Currently, the CapSense Component supports the slowest baseline update rate as 1/255. Some advanced gesture features like double-tap and scroll detection need very high scan rate; the slowest update rate provided by the CapSense Component may not be slow enough.

This project features the CapSense Component configured with three mutual capacitance buttons. However, the custom baseline update technique is equally applicable to any sensor type with self-capacitance sensing, and any widget type such as sliders and touchpads.

Hardware Setup

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

Software Setup

The example uses a terminal software such as the CapSense Tuner (which is included with PSoC Creator Tool).

Background

CapSense Components calculate the baseline by filtering the rawcount using an IIR filter, using the following equation. See the CapSense component datasheet for more details.

$$Output = \frac{N}{K} \times input + \frac{(K - N)}{K} \times previousOutput$$

where:

K is always 256.

N is the IIR filter rawcount coefficient selectable from 1 to 255 in the customizer.

An IIR filter is applied on the rawcount to calculate the baseline; the IIR coefficient determines the baseline update rate. Currently, the CapSense Component supports the slowest baseline update rate as 1/255, when the 'Regular widget baseline co-efficient (N)' is set to '1'.

For some application which needs higher scan rate, the current slowest baseline update rate may not be slow enough. For example, a gesture feature such as 'One-finger Flick[1]' needs to have a high scan rate to identify high-speed displacement. By setting a higher scan rate, baselines update faster as well. This can cause non-detection of finger touchdown when the finger touch frequency is high, as shown in Figure 1.

[1] A flick gesture is a combination of a touchdown followed by a high-speed displacement and a lift off event. Refer CapSense Component Datasheet section 'General Gesture Parameters' for more details on different gesture features.

Figure 1. Custom Use Case Showing Non-Detection of Finger Touch due to Baseline Update[2]



To get maximum scan rate and smaller signal (CY8CKIT-149 PSoC 4100S Plus Prototyping Kit by default has higher signal), to show the behavior shown in Figure 1, the following custom settings are applied (see Figure 7 to Figure 13).

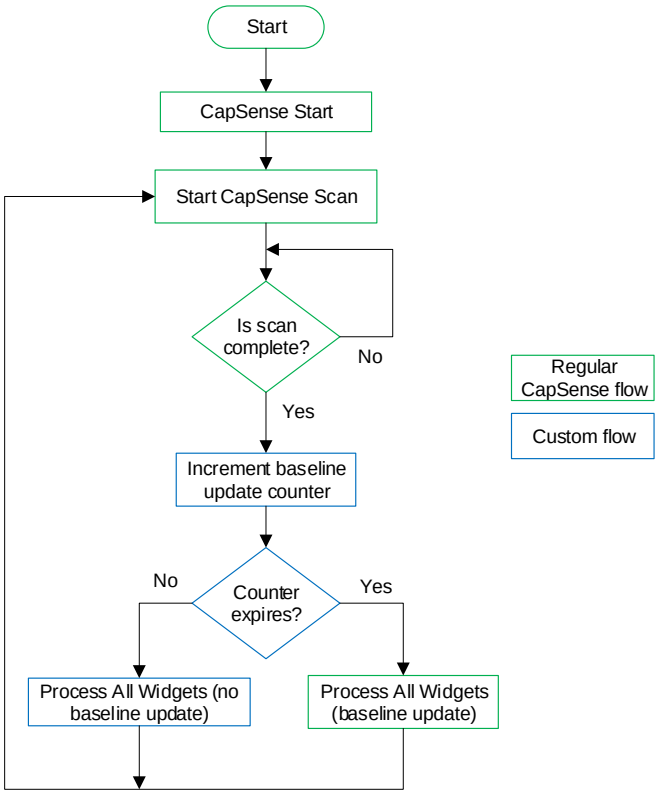
- Number of sub-conversions is set to 20.
- Because Button Signal = 450, Finger Threshold and Noise Threshold are set to 360 and 180 respectively.
- `SAMPLES_BASELINE_UPDATE` can be changed to zero in this code example to show the default behavior without applying the custom baseline update algorithm mentioned in this document. See Figure 6 for the improved response after applying the algorithm mentioned in this document.
- Sensor auto-reset is enabled.

[2] Similar behavior is applicable for long press with the auto reset feature enabled.

Design and Implementation

A software counter is used to control the execution of the baseline update algorithm.

Figure 2. Custom Baseline Update Flowchart



The `CapSense_ProcessWidgetExt(uint32 widgetId, uint32 processMode)` function performs data processes for the specified widget specified by the `processMode` parameter. If this parameter is set to `CapSense_PROCESS_ALL`, this function executes all the tasks, with each bit representing each process.

If you clear the baseline update process `CapSense_PROCESS_BASELINE` bit in `processMode`, this function performs all the tasks except the baseline update (Code 1). See CapSense Component datasheet.

Code 1. Example for Skipping Baseline Update Process

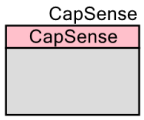
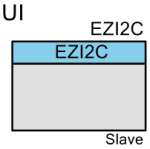
```
uint32 processMode = CapSense_PROCESS_ALL;

processMode &= (~(uint32 )CapSense_PROCESS_BASELINE);

CapSense_ProcessWidgetExt(widgetId, processMode);
```

Figure 3 shows the PSoC Creator schematic for this code example.

Figure 3. TopDesign Schematic of CE229162_CapSense_CustomBaselineUpdate

CapSense	EZI2C
The CapSense Component scans 3 CSX buttons in different widgets 	The I2C Component tranfers the CapSense data to CapSense Tuner GUI 

Operation

Open the *main.c* file to configure this project. Define `SAMPLES_BASELINE_UPDATE` to set the number of scan samples before updating the baseline.

```
/* User configuration: number of scan samples before updating baseline */
#define SAMPLES_BASELINE_UPDATE        (10u)
```

1. **Build the project:** Build the project “CE229162_CapSense_CustomBaselineUpdate” and program it into the PSoC 4100S Plus device. Choose **Debug** > **Program**.
2. **Launch Tuner GUI and monitor data:** See the CapSense Component datasheet section **CapSense tuner** to set up the Tuner GUI.

   This example requires to use the **Graph view** tab in Tuner GUI. As shown in Figure 4 and Figure 5, the baseline updates to rawcount value per the baseline IIR filter setting (rate = 1/255), every `SAMPLES_BASELINE_UPDATE` number of samples.

   Figure 4 shows the baseline update waveform for `SAMPLES_BASELINE_UPDATE` set to 0 (baseline update is executed every sample).

Figure 4. Custom Baseline Update (`SAMPLES_BASELINE_UPDATE` = 0)

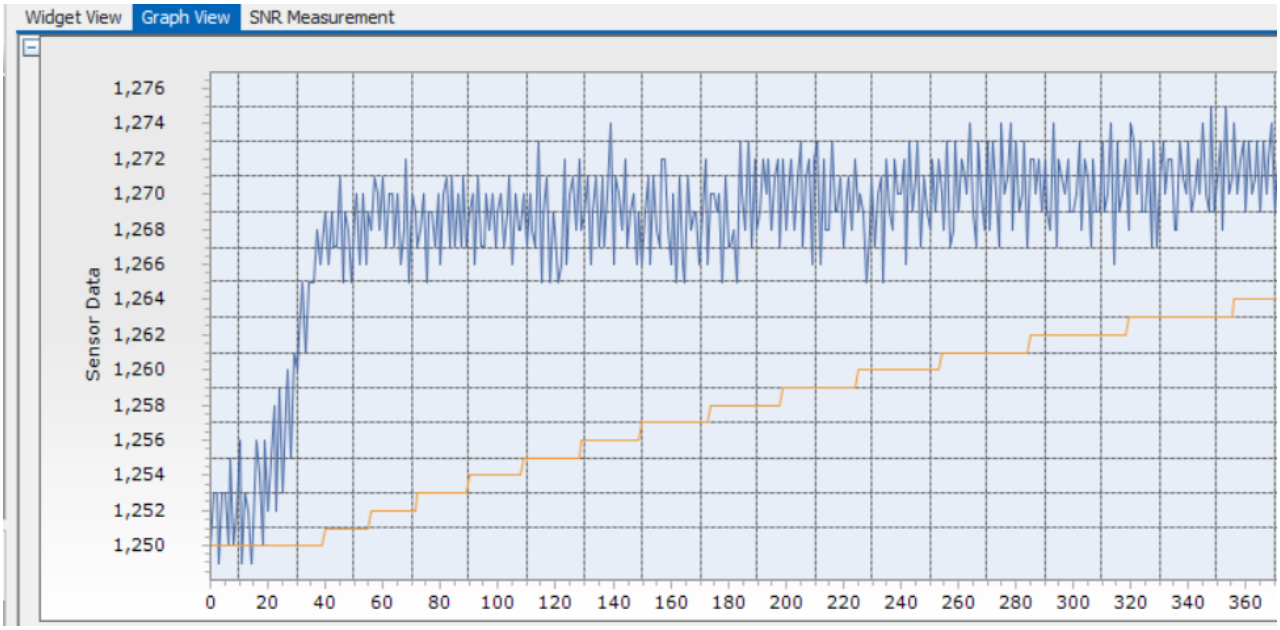
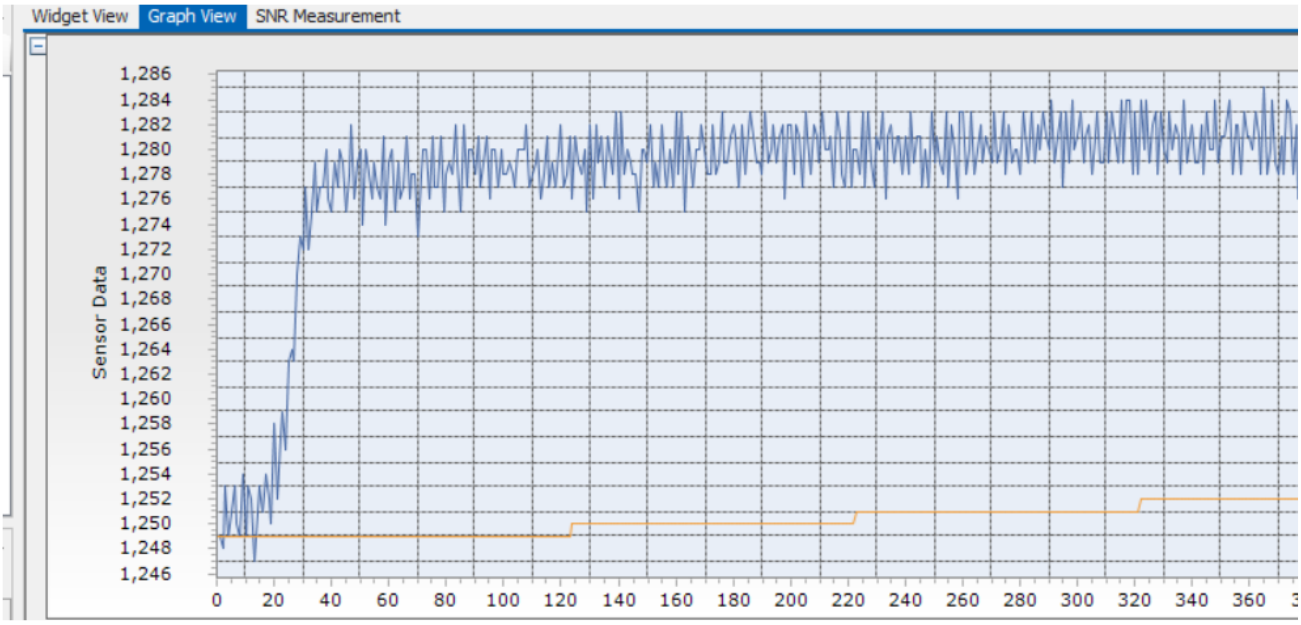


Figure 5 shows the baseline update waveform for `SAMPLES_BASELINE_UPDATE` set to 10 (10 samples are skipped before executing baseline update).

Figure 5. Custom Baseline Update (`SAMPLES_BASELINE_UPDATE` = 10)



Note: Figure 4 and Figure 5 show the zoomed-in version of behavior shown in Figure 1.

Figure 6 shows the use case mentioned in the Background section after applying the above algorithm, where SAMPLES_BASELINE_UPDATE is set to 10. The following result shows that the baseline is not reaching a higher level due to frequent touches.

Figure 6. Custom Use Case Showing Detection of Finger Touch After Applying Baseline Update Algorithm



Components and Settings

Table 2 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 2. PSoC Creator Components

Component	Instance Name	Purpose	Non-default Settings
CapSense	CapSense	The CapSense Component is configured to scan 3 – CSX buttons	For Basic Setting, see Figure 7. CapSense Basic Settings
			For Advance Settings, see Figure 8. CapSense Advanced Settings
			For Widget Settings, see Figure 9 to Figure 13
EZI2C	EZI2C	To establish communication with the Tuner application	Data Rate (kbps): 400 Sub-address size (bits): 16

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

Figure 7. CapSense Basic Settings

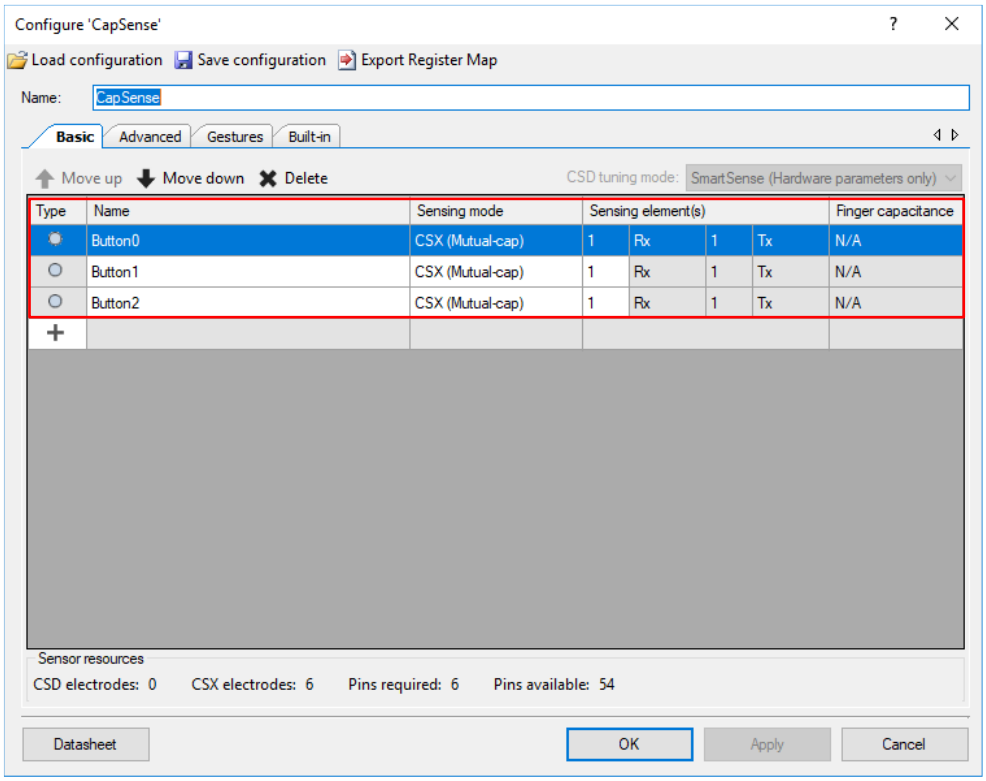


Figure 8. CapSense Advanced Settings

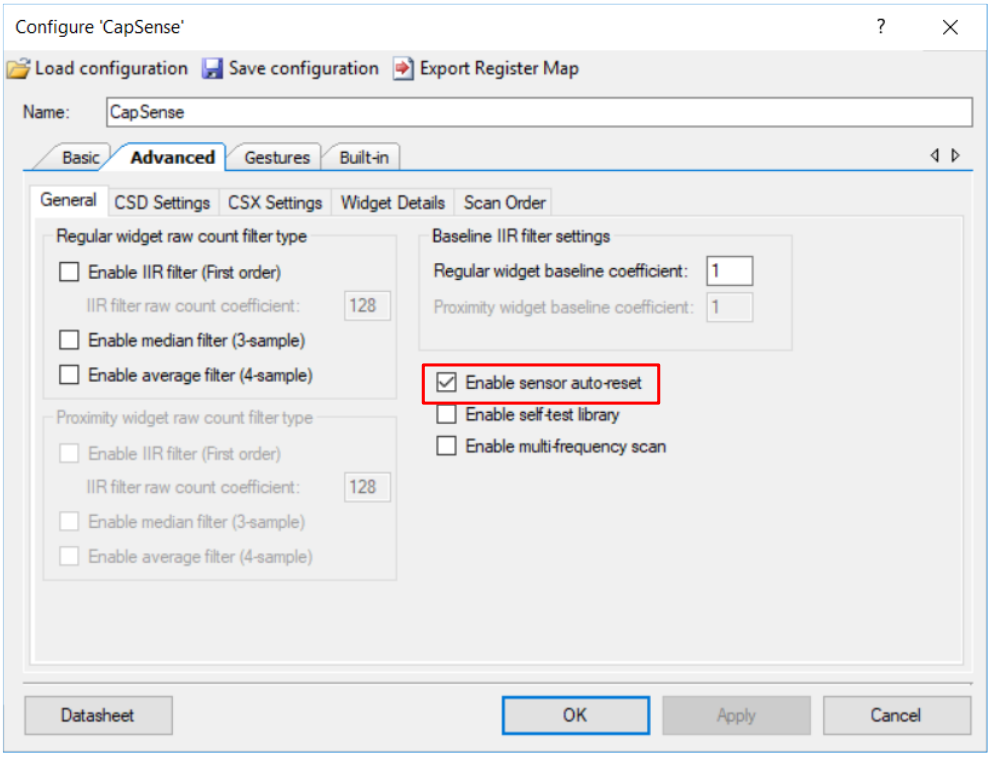


Figure 9. CapSense Button0 Widget Settings

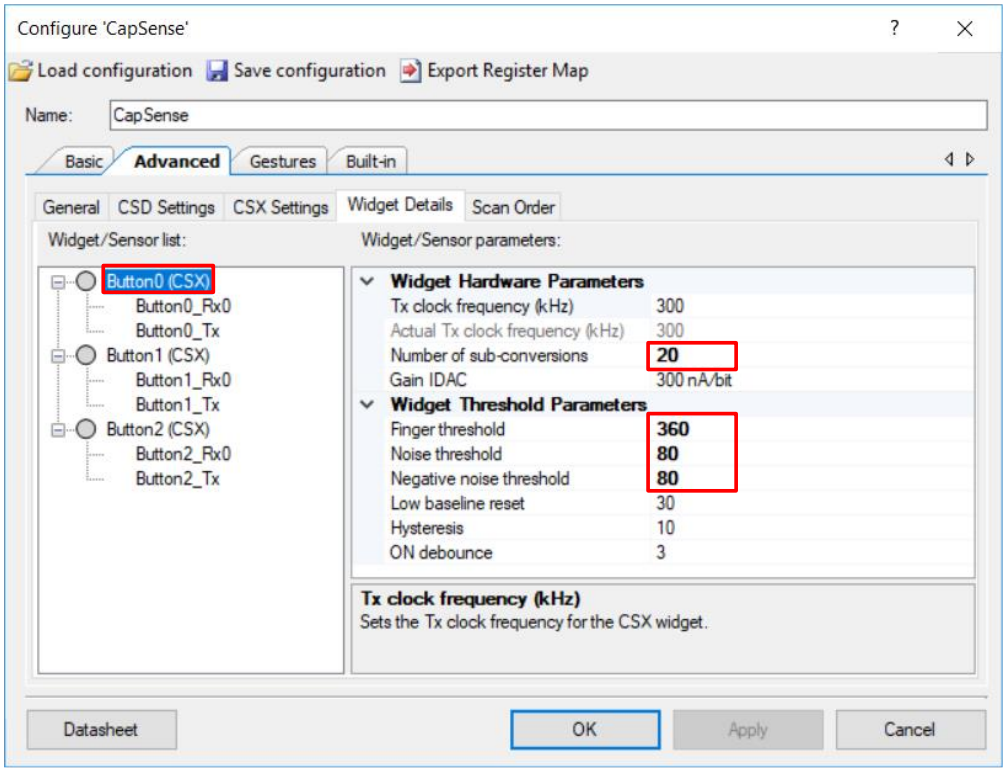


Figure 10. CapSense Button1 Widget Settings

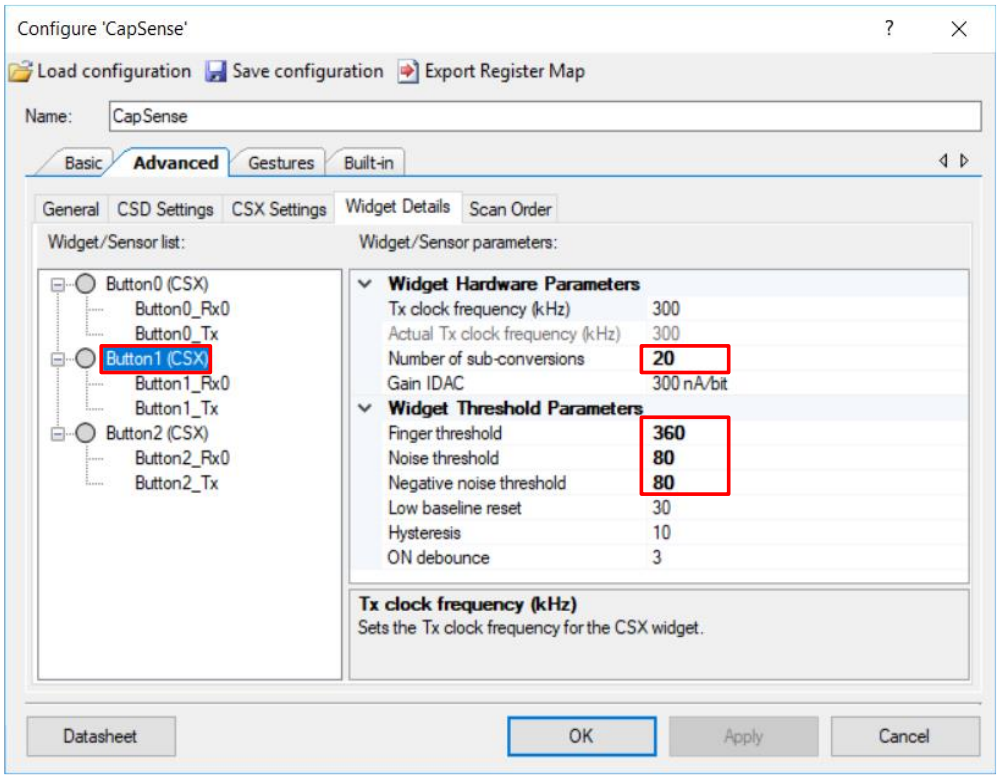


Figure 11. CapSense Button1 Tx Widget Settings

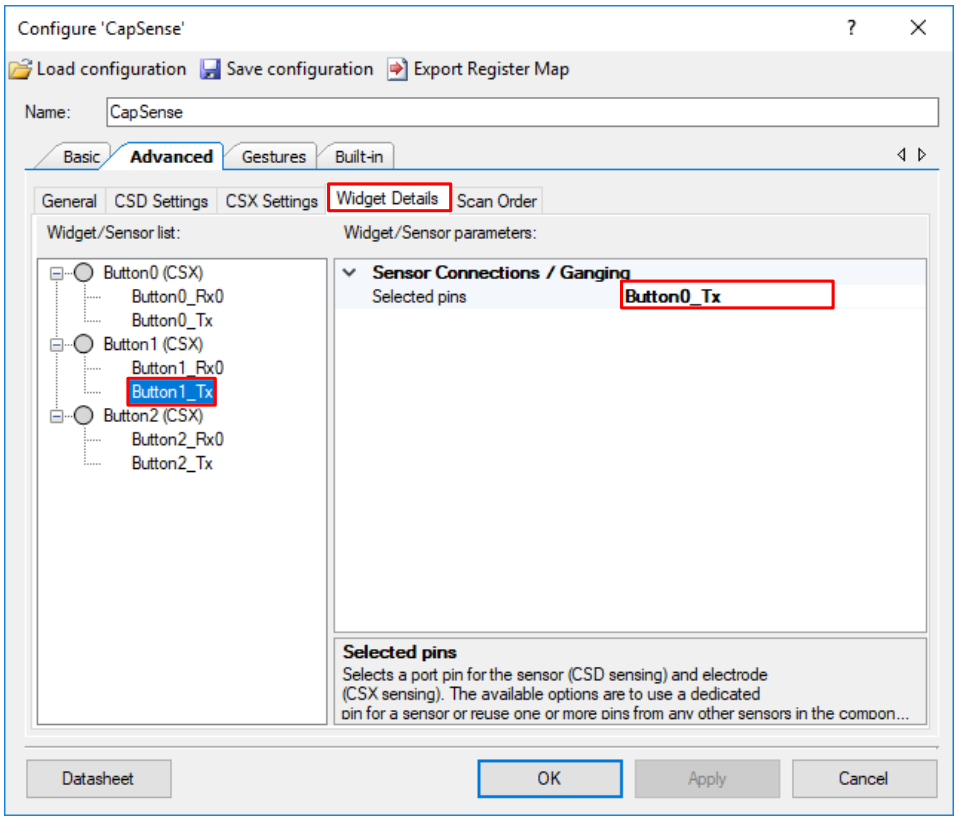


Figure 12. CapSense Button2 Widget Settings

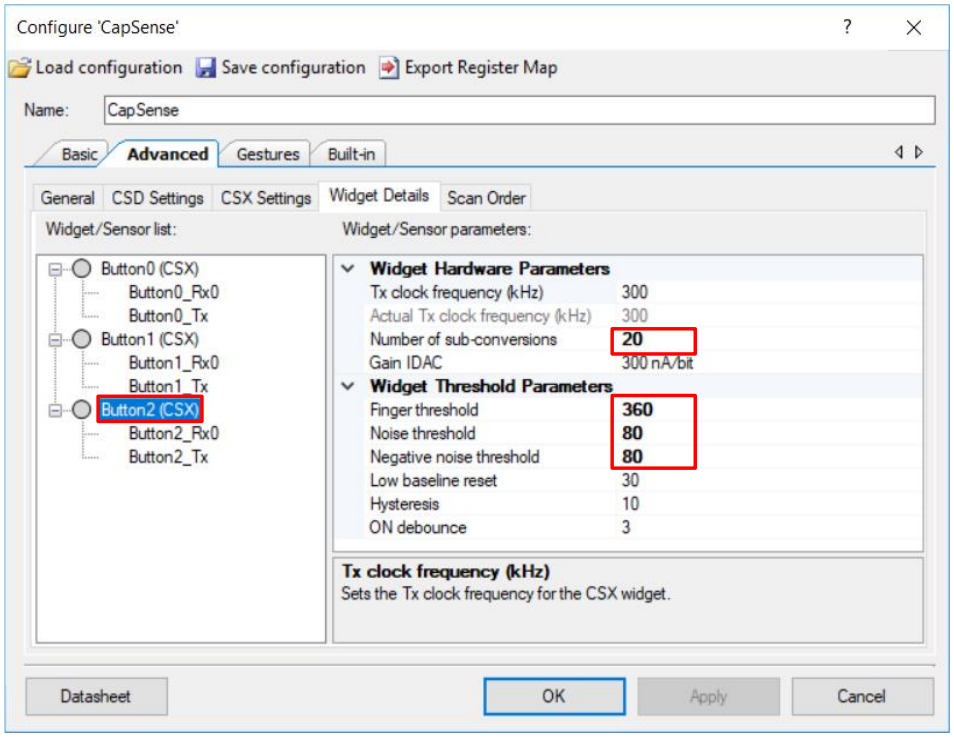


Figure 13. CapSense Button2 Tx Widget Settings

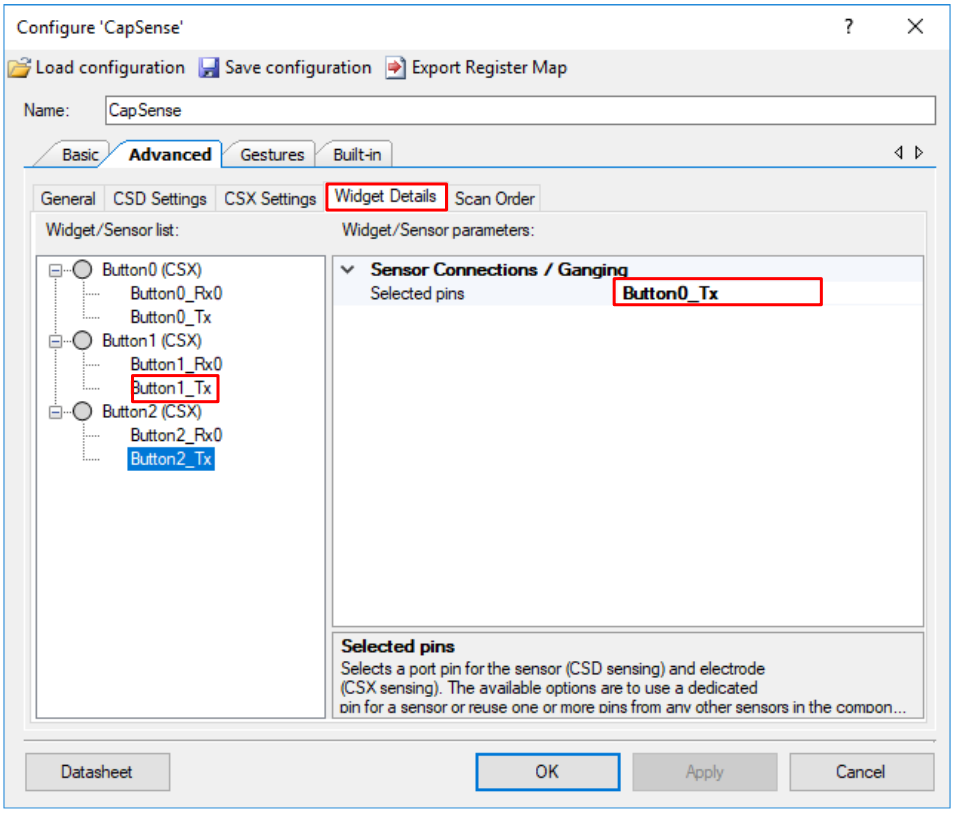


Table 3 shows the pin assignments for the project done through the **Pins** tab in the **Design Wide Resources** window for this code examples. These assignments are compatible with CY8CKIT-149 PSoC 4100S Plus Prototyping Kit and CY8CKIT-145-40XX PSoC 4000S CapSense Prototyping Kit.

Table 3. Pin Assignments

Pin Name	CY8CKIT-149	CY8CKIT-145
CapSense:CintA	P4[2]	P4[2]
CapSense:CintB	P4[3]	P4[3]
CapSense:Rx[0]	P4[6]	P1[4]
CapSense:Rx[1]	P4[5]	P1[5]
CapSense:Rx[2]	P4[4]	P1[6]
CapSense:Tx	P0[2]	P1[3]
EZI2C:scl	P3[0]	P1[0]
EZI2C:sda	P3[1]	P1[1]

Reusing This Example

This example is designed for CY8CKIT-149 PSoC 4100S Plus Prototyping Kit and CY8CKIT-145-40XX PSoC 4000S CapSense Prototyping Kit. To port the design to a different PSoC 4 MCU device and/or kit, change the target device using the Device Selector and pin assignments in the Design Wide Resources Pins settings as needed.

Related Documents

For a comprehensive list of PSoC 3, PSoC 4, and PSoC 5LP resources, see KBA86521 in the Cypress community.

Application Notes	
AN85951 - PSoC 4 and PSoC 6 MCU CapSense Design Guide	Describes how to design capacitive touch sensing applications with PSoC 4 and PSoC 6
AN79953 - Getting Started with PSoC 4	Introduces the PSoC 4 device and explains how to build a PSoC Creator project
AN64846 - Getting Started with CapSense	This guide is an ideal starting point for those who are new to capacitive touch sensing (CapSense).
PSoC Creator Component Datasheets	
CapSense	Supports various interfaces such as Button, Matrix Buttons, Slider, Touchpad, and Proximity Sensor
EZI2C Slave	Supports one or two address decoding with independent memory buffers
Pins	Supports connection of hardware resources to physical pins
Clock	Supports local clock generation
Device Documentation	
PSoC 4100S Plus Datasheet	PSoC 4 Technical Reference Manuals
PSoC 4000S Datasheet	
Development Kit Documentation	
CY8CKIT-149 PSoC 4100S Plus Prototyping Kit	
CY8CKIT-145-40XX PSoC® 4000S CapSense Prototyping Kit	
Tool Documentation	
PSoC Creator	Look in the downloads tab for Quick Start and User Guides

Document History

Document Title: CE229162 – PSoC 4 CapSense Custom Baseline Update

Document Number: 002-29162

Revision	ECN	Date	Description of Change
**	6834455	03/23/2020	New code example

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 | Code Examples | 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, 2020. This document is the property of Cypress Semiconductor Corporation and its subsidiaries ("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 shall have no liability arising out of any security breach, such as unauthorized access to or use of a Cypress product. CYPRESS DOES NOT REPRESENT, WARRANT, OR GUARANTEE THAT CYPRESS PRODUCTS, OR SYSTEMS CREATED USING CYPRESS PRODUCTS, WILL BE FREE FROM CORRUPTION, ATTACK, VIRUSES, INTERFERENCE, HACKING, DATA LOSS OR THEFT, OR OTHER SECURITY INTRUSION (collectively, "Security Breach"). Cypress disclaims any liability relating to any Security Breach, and you shall and hereby do release Cypress from any claim, damage, or other liability arising from any Security Breach. 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. "High-Risk Device" means any device or system whose failure could cause personal injury, death, or property damage. Examples of High-Risk Devices are weapons, nuclear installations, surgical implants, and other medical devices. "Critical Component" means any component of a High-Risk Device whose failure to perform can be reasonably expected to cause, directly or indirectly, the failure of the High-Risk Device, 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 any use of a Cypress product as a Critical Component in a High-Risk Device. You shall indemnify and hold Cypress, its directors, officers, employees, agents, affiliates, distributors, and assigns harmless from and against all claims, costs, damages, and expenses, arising out of any claim, including claims for product liability, personal injury or death, or property damage arising from any use of a Cypress product as a Critical Component in a High-Risk Device. Cypress products are not intended or authorized for use as a Critical Component in any High-Risk Device except to the limited extent that (i) Cypress's published data sheet for the product explicitly states Cypress has qualified the product for use in a specific High-Risk Device, or (ii) Cypress has given you advance written authorization to use the product as a Critical Component in the specific High-Risk Device and you have signed a separate indemnification agreement.

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.