CE229279 – CapSense Custom Scan

Objective

This code example demonstrates CapSense® Component's custom scanning through callback functions that allow altering the sensor parameters during runtime or synchronizing the CapSense scan with non-CapSense operations.

In this code example, the callback function is used to change the inactive sensor state to either shield or ground depending on the sensor being scanned.

Requirements

Tool: PSoC Creator™ 4.2 and PSoC Programmer™ 3.28.6

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

Associated Parts: PSoC 4000S, PSoC 4100S Plus

Related Hardware:

CY8CKIT-145-40XX PSoC® 4000S CapSense Prototyping Kit, CY8CKIT-149 PSoC 4100S Plus Prototyping Kit

Overview

Custom scanning provides fine control over sensor parameters and helps in adjusting the sensitivity for various sensors within a widget. Custom scanning of CapSense sensors might be required for some very specific applications such as:

1. Changing the inactive sensor connection based on the sensor being currently scanned
2. Setting different tuning parameters for various sensors within a widget
3. Optimizing the sensor performance by disabling source of noise when the affected sensors are being scanned

This example demonstrates the custom scanning capability of CapSense using callback functions with the linear slider implementation. In this code example, sensors which are near the sensor being scanned are connected to the shield, while other sensors are connected to ground. This provides liquid tolerance and reduced parasitic capacitance of the sensor as well as the shield. It also decreases emissions because the entire slider segment is not always being switched with high-frequency shield signals.

This document explains the following:

1. Method to check the working of inactive sensors
2. Callback Function of the CapSense Component
3. Extending this code example for flexible CapSense scanning

This code also sends the CapSense data structure over I^2C to the on-board KitProg, which enables reading data from CapSense Tuner GUI. CapSense Tuner can be used to observe and modify the parameters of CapSense sensors and measure the signal-to-noise ratio (SNR) of the sensor.

Prerequisites

This code example is for advanced CapSense users; the user is assumed to be familiar with CapSense technology and basic terms such as raw counts, baseline, and difference counts. If you are new to CapSense, see AN64846 - Getting Started with CapSense, AN85951 – PSoC 4 and PSoC 6 MCU CapSense Design Guide, and the code example CE220891 – CapSense with Breathing LED (In PSoC Creator, select **File** > **Code example** > **Device family** > **PSoC 4100S Plus**).

Hardware Setup

This example uses the CY8CKIT-145-40xx and CY8CKIT-149 kits. Refer to the kit guide to ensure that the kit is configured correctly. Refer the Pin Assignments for CapSense Custom Scan section for details on configuring the pins of the kits.

Software Setup

Ensure that you have all the software tools as mentioned in Requirements section installed.

Theory

Callback Function

The flexibility in shield is obtained by changing the inactive sensor connection in the callback function – `CapSense_StartSampleCallback`. This function is defined in the *cyapicallbacks.c* file. This function is called just before scanning each sensor.

The `CapSense_StartSampleCallback` function takes the widget index and sensor index that is going to be scanned as the parameter. The function then loops through all the sensors in the widget. If the sensor is found to be adjacent to the sensor being scanned (obtained as a parameter), the pin state is set to shield; if not, the pin state is set to ground. This is done using the `CapSense_SetPinState` API function (See CapSense Component datasheet for details on this API function). This takes the sensor ID, widget ID and the state of the pin (`CapSense_SHIELD` or `CapSense_GROUND`, in this case) as the parameter and sets up the status accordingly. If the sensor is adjacent to the sensor being scanned, the status is set to `CapSense_SHIELD`; if not, the status is set to `CapSense_GROUND`. See Figure 1 for the code snippet for custom scan implemented in this code example.

Figure 1. Code Snippet for Custom Scan

```
void CapSense_StartSampleCallback (uint32 currentWidgetIndex, uint32 currentSensorIndex)
{
    /* Variable to loop through every sensor in the widget*/
    uint8 sensorIndex;

    if(currentWidgetIndex == CapSense_LINEARSLIDER_WDGT_ID)
    {
        for(sensorIndex = 0; sensorIndex < CapSense_LINEARSLIDER_NUM_SENSORS; sensorIndex++)
        {
            if(sensorIndex != currentSensorIndex)
            {
                if((sensorIndex == (currentSensorIndex - 1)) || (sensorIndex == (currentSensorIndex + 1)))
                {
                    /* If the sensor is adjacent to the sensor being scanned,
                     * configure it as shield.
                     */
                    CapSense_SetPinState(CapSense_LINEARSLIDER_WDGT_ID, sensorIndex, CapSense_SHIELD);
                }
                else
                {
                    /* If the sensor is not adjacent to the sensor being
                     * scanned, connect it to ground.
                     */
                    CapSense_SetPinState(CapSense_LINEARSLIDER_WDGT_ID, sensorIndex, CapSense_GROUND);
                }
            }
        }
    }
}
```

Custom Scanning in CapSense

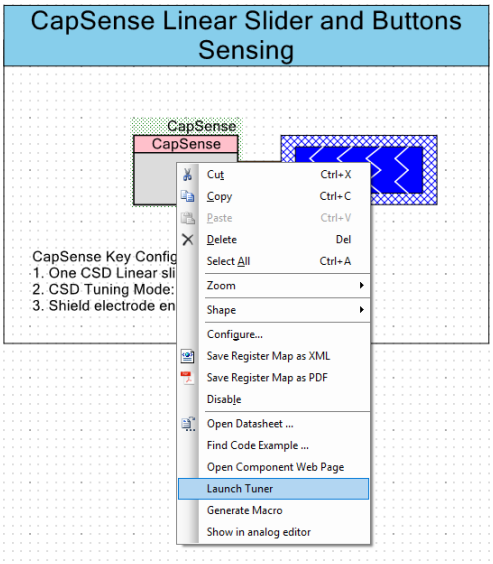
The CapSense Component offers a lot of flexibility in scanning the sensors. This example demonstrates one such use case with the custom shield configuration of inactive sensors; however, this can be extended to many more scenarios. Some examples include

1. Tuning each sensor of a widget separately so that the gain/sensitivity can be configured. The modulator IDAC for each sensor can be set separately, which in turn adjusts the gain of each sensor in a widget. This can be done by writing the modulator IDAC code for different sensors using `CapSense_SetParam` (see CapSense Component datasheet). Once the IDAC is set, `CapSense_SsCSDSetUpIdacs` must be called that configures the IDAC registers. It takes the pointer to `CapSense_RAM_WD_BASE_STRUCT` as the parameter and sets up the IDAC accordingly. Because each sensor in a widget cannot be configured individually, this method can be used to tune each sensor's gain.
2. This can also be used in a system where there might be noises due to external electrical signals affecting CapSense operation. If there is a high-speed switching signal that affects a few sensors in the system, the firmware can turn off such sources of noise before these sensors are scanned. This would help with better CapSense operation while keeping all other operations as functional as possible.

Operation

1. Plug the CY8CKIT-145-40xx or CY8CKIT-149 kit board into your computer's USB port.
2. Build the project by selecting **Build** > **Build CE229279_CapSenseCustomScan** and program it into the PSoC 4 MCU device. Choose **Debug** > **Program**.
3. Slide your finger over the CapSense linear slider and observe that LEDs turn ON up to the position of touch.
4. Observe the actual waveform at the sensors in an oscilloscope by probing the sensors (See Oscilloscope Waveforms). It can be observed that only neighboring sensors to the sensor being scanned are switched with the shield waveform and all other sensors are at ground potential.
5. Launch the Tuner GUI. Right-click on the CapSense Component and choose **Launch Tuner** as shown in Figure 2, or select **Tools** > **Component Tuners** > **CapSense**.

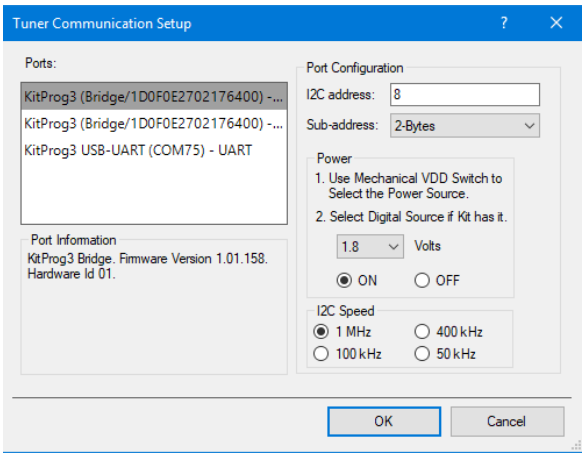
Figure 2. Launch Tuner GUI



A new window called "Sense Tuner" appears on the screen.

6. Select **Tools** > **Tuner Communication Setup** or click the **Tuner Communication Setup** icon to open the Tuner Communication Setup dialog.
7. Select the appropriate I²C communication device and set the following parameters as shown in Figure 3:

Figure 3. Tuner Communication Setup Parameters



8. Click **Connect** to establish connection.

9. Click **Start** to start data streaming from the device. The Tuner GUI displays the data from the sensor in the widget view and Graph view tabs. See the “CapSense Tuner” section in CapSense Component datasheet for detailed information on Tuner GUI.

Oscilloscope Waveforms

Figure 4 shows the oscilloscope waveform of four sensors in the default scanning implementation. It can be observed that all inactive sensors are always driven to shield. Therefore, there is always a switching signal on all the slider elements for the entire scan duration. This also means that more area is driven to shield and increases the emissions and the parasitic capacitance that needs to be driven.

Figure 4. Sensor Waveforms: Normal Implementation

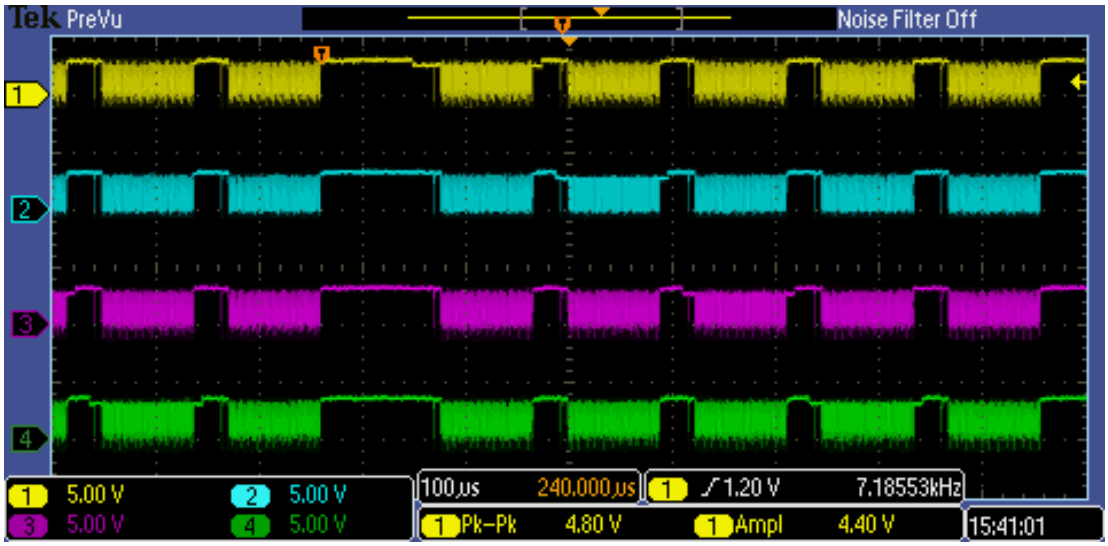
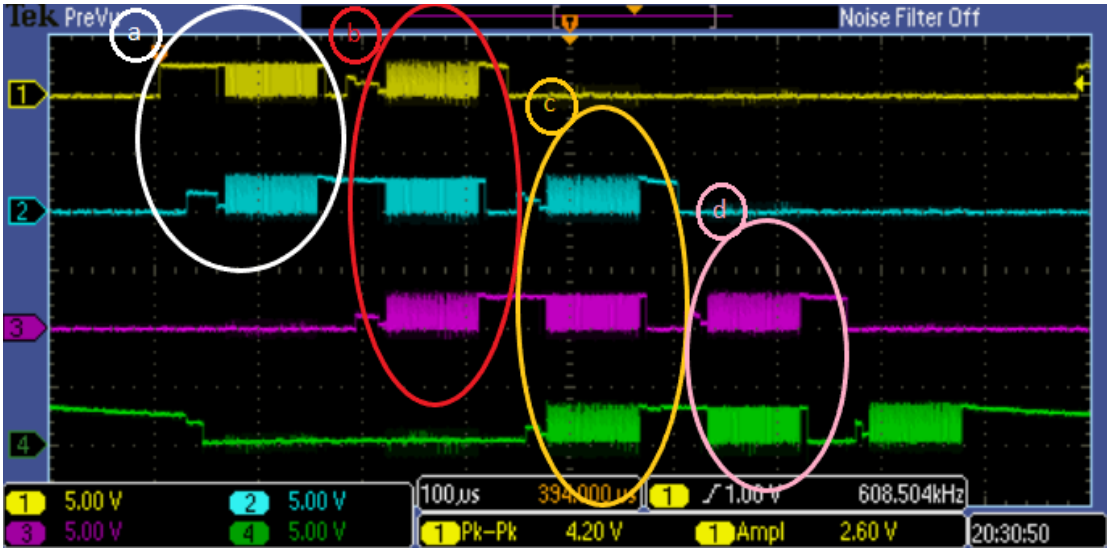


Figure 6 shows the oscilloscope waveform of four sensors of the linear slider widget with the custom scanning implemented. It can be observed that whenever a sensor is scanned, only the adjacent sensors are driven to shield. All other sensors are connected to ground, thereby reducing the total area of emission. See Callback Function for details on how to implement the custom shield connection through firmware.

Figure 5. Sensor Waveforms – Custom Scan



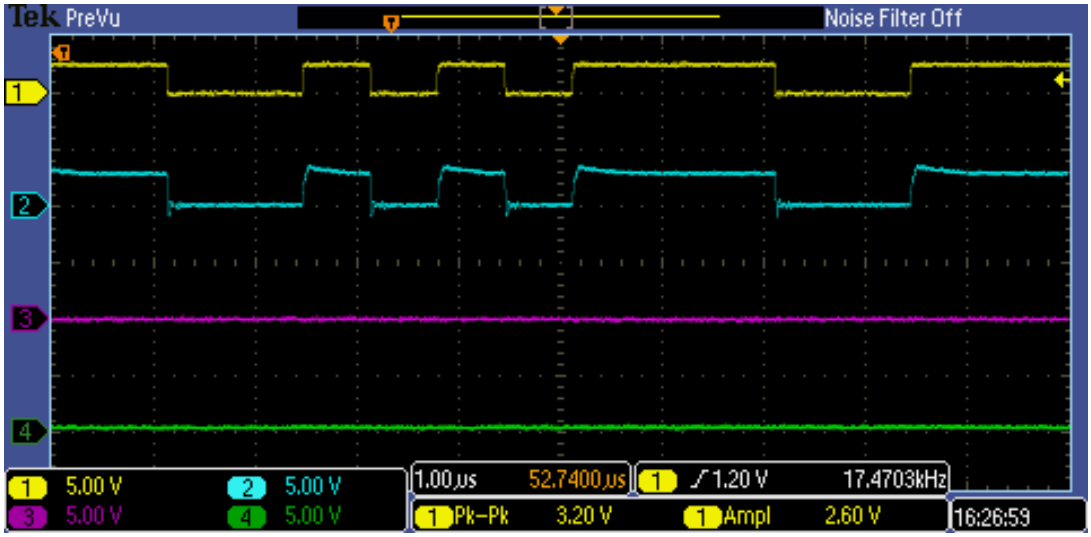
a. Sns0 is being scanned and Sns1 is being driven to shield. All other sensors are connected to ground.

b. Sns1 is being scanned and Sns0 and Sns2 (adjacent sensors) are driven to shield. All others are connected to ground

c. Sns2 is being scanned and Sns1 and Sns3 are driven to shield. All other sensors are connected to ground.

d. Sns3 is being scanned and Sns2 and Sns4 (not displayed in Figure 4) are driven to shield. All other sensors are connected to ground.

Figure 6. Sns0 (yellow) Being Scanned, Sns1 (blue) Driven to Shield, Sns2 and Sns3 Connected to Ground



This effectively reduces the area where the switching signals are driven and therefore, the amount of radiation by CapSense scanning is less, while being tolerant to liquid drops.

Figure 7 and Figure 8 show the difference between normal scanning and custom scanning with respect to slider segments. The segment shown in red is the actual sensor being scanned, while segments in yellow show sensors driven to shield. Segments in green are sensors that are connected to ground.

Figure 7. Normal Scanning of Slider



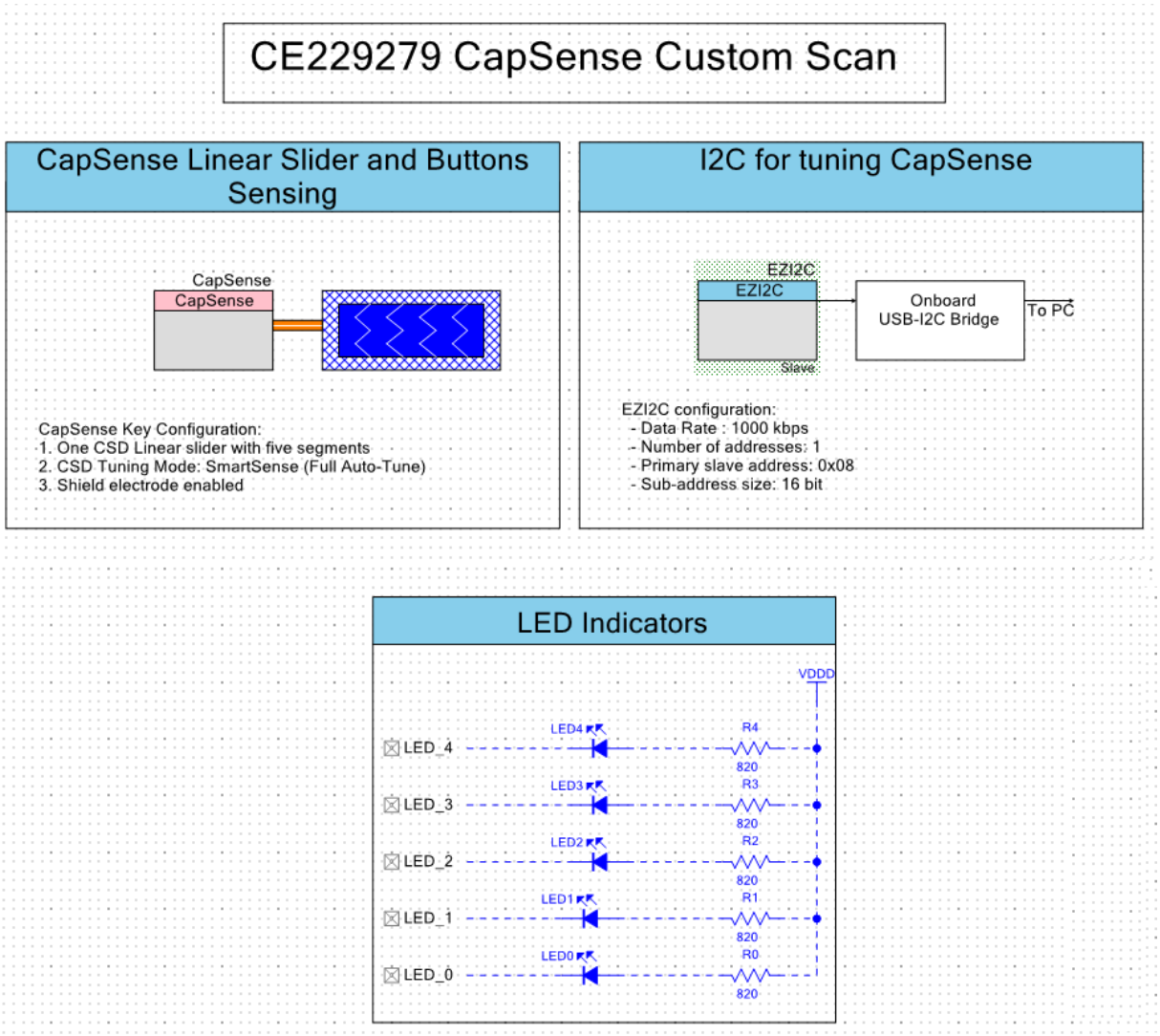
Figure 8. Custom Scanning of Slider



Design and Implementation

Figure 9 shows the PSoC Creator schematic for this code example. This code example uses CapSense, SCB (configured as EZI2C Slave), and GPIO pins.

Figure 9. PSoC Creator Schematic



The CapSense Component is configured with one linear slider widget with five segments. This code example uses CapSense Sigma-Delta (CSD) sensing method for sliders. When the CapSense linear slider is touched, the LEDs turns ON to indicate the position of touch.

This code example is designed for the PSoC 4000S and PSoC 4100S Plus device and associated kit CY8CKIT-145-40xx and CY8CKIT-149. Table 1 shows the pin assignments of the PSoC Creator Components.

Table 1. Pin Assignments for CapSense Custom Scan

Pin name	CY8CKIT-145-40XX (CY8C4045AZI-S413)	CY8CKIT-149 (CY8C4147AZI-S475)
\Capsense:Cmod\ (Cmod)	P4[1]	P4[1]
\CapSense:Shield\ (Shield)	P2[7]	P2[5]
\CapSense:Sns[0]\ (SLD_Sns0)	P0[0]	P2[7]
\CapSense:Sns[1]\ (SLD_Sns1)	P0[1]	P6[0]
\CapSense:Sns[2]\ (SLD_Sns2)	P0[2]	P6[1]
\CapSense:Sns[3]\ (SLD_Sns3)	P0[3]	P6[2]
\CapSense:Sns[4]\ (SLD_Sns4)	P0[6]	P6[4]
\EZI2C:scl\	P1[0]	P3[0]
\EZI2C:sda\	P1[1]	P3[1]
LED_0	P2[0]	P1[0]
LED_1	P2[1]	P1[2]
LED_2	P2[2]	P1[4]
LED_3	P2[3]	P1[6]
LED_4	P2[4]	P2[0]

Components and Settings

Table 2 lists the PSoC Creator Components used in this example, their instance name in the project, the component version and the hardware resources that are consumed.

Table 2. PSoC Creator Components

Component	Instance Name	Version	Hardware resource
CapSense	CapSense	V7.0	One CSD
EZI2C Slave (SCB Mode)	EZI2C	V4.0	One Serial Communication Block (SCB) and two GPIOS
Digital Output Pins	LED_0	V2.20	GPIO
	LED_1	V2.20	GPIO
	LED_2	V2.20	GPIO
	LED_3	V2.20	GPIO
	LED_4	V2.20	GPIO

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

Reusing This Example

The custom scan implementation is supported in CapSense Component version 4.0 and above. This is supported in all CapSense-enabled devices in PSoC 4 family including:

- **Third generation CapSense:** PSoC 4000, PSoC 4100, PSoC 4200, PSoC 4100M, PSoC 4200M, PSoC 4200L, PSoC 4100 BLE and PSoC 4200 BLE.
- **Fourth generation CapSense:** PSoC 4000S, PSoC 4100S PSoC 4100 S Plus.

The tuning parameters must be changed in accordance with the kit used and sensor properties. See the PSoC 4 and PSoC 6 MCU CapSense Design Guide for more information on CapSense sensor tuning.

Related Documents

Application Notes		
AN64846	Getting Started with CapSense	This guide is an ideal starting point for those new to capacitive touch sensing (CapSense®) as well as for learning key design considerations and layout best practices to ensure design success.
AN85951	PSoC 4 and PSoC 6 MCU CapSense Design Guide	Describes how to design capacitive touch sensing applications with PSoC 4 and PSoC 6 MCU
AN79953	Getting Started with PSoC 4	Introduces the PSoC 4 device and explains how to build a PSoC Creator project
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.
I2C		Supports I2C Slave, Master, Multi-Master, and Multi-Master-Slave configurations
Device Documentation		
PSoC 4100S Plus Datasheet		PSoC 4100S Plus Architecture Technical Reference Manuals PSoC 4100S Plus Register Technical Reference Manuals
PSoC 4000S Datasheet		PSoC 4000S Architecture Technical Reference Manuals PSoC 4000S Register Technical Reference Manuals
Development Kit (DVK) Documentation		
CY8CKIT-149 PSoC 4100S Plus Prototyping Kit		
CY8CKIT-145-40XX PSoC® 4000S CapSense Prototyping Kit		

Document History

Document Title: CE229279 – CapSense Custom Scan

Document Number: 002-29279

Revision	ECN	Date	Description of Change
**	6825272	03/17/2020	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.