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.

CE229140 – PSoC 4 CapSense Custom Baseline Reset

Objective

This example demonstrates the custom methods to detect stuck conditions, like those resulting from temperature, humidity, or other environment changes, and to recover from them by resetting the CapSense® baseline.

Requirements

Tool: PSoC Creator™ 4.2

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

The example demonstrates detection of two conditions at which baseline or sensor is stuck. For instance, baseline stuck to a constant value or sensor status stuck at ON. Both these conditions are demonstrated using either time-based detection or sample-based detection.

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

Hardware Setup

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

Software Setup

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

Background

Baseline is a low-pass filtered value of raw count[1] and acts as a reference for calculating the signal. Ideally baseline should always be at 'no touch' raw count value. An abnormal condition such as temperature, humidity, or other environment changes can cause the baseline to be stuck at certain levels.

A change in temperature or humidity can cause variation in actual capacitance value as well as the operation of internal components (IDAC, clock, and so on) and can lead to variation in measured raw count. In normal situations, the baseline should follow the raw count for any environment changes. Sometimes if there are sudden changes to temperature or humidity, and raw count increases suddenly, baseline may not follow immediately and can get stuck at lower values. In some cases, the sudden change may be as big or higher than the touch response, in which case, the sensor will be stuck at ON indefinitely.

You may want to detect the abnormal conditions and reset the baseline to bring the system back to a normal state.

Design and Implementation

This code example demonstrates two approaches to reset the baseline.

[1] Sensor capacitance is converted into a count value by the CapSense algorithm. The unprocessed count value is referred to as raw count. (For more details, see the Definitions section of AN64846 - Getting Started with CapSense).

1. Time-based Baseline Reset

   In this approach, raw count values are monitored to check if the sensor is stuck. A baseline reset is executed if the sensor is stuck for a user defined period. For time-based approach, a software counter array is used to track the duration each sensor is stuck.

   System timer (SysTick) is used to create a callback every 100 milliseconds and increment the software counter of each sensor. When the counter reaches the user defined time-out period, a 'baseline reset' function is called to reset the baseline for that sensor. The counter is reset when the raw count is out of stuck condition or after the baseline reset. In this example SysTick is used as a timer. You can use any method to create a periodic interrupt and increment the software counter.

   The sensor baseline reset time can be calculated as the product of the callback interval (default 100 ms) and user-defined time-out period.

2. Sample-based Baseline Reset

   In this approach, raw count values are monitored to check if the sensor is stuck. A baseline reset is executed if the sensor is stuck for a user defined number of samples (i.e., number of CapSense scans). For sample-based approach, a software counter array is used to track the number of samples for which each sensor is stuck.

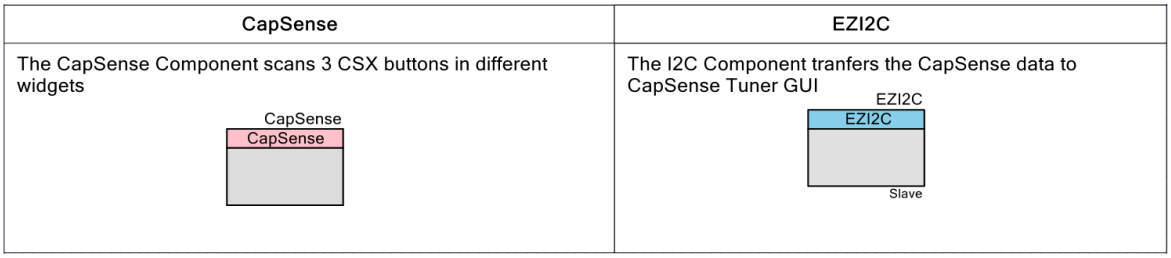
   In this approach, the software counter is incremented when the raw count reaches the user-defined stuck condition. When a counter reaches the user-defined sample limit, a 'baseline reset' function is called to reset the baseline for that sensor. The counter is reset when the raw count is out of stuck condition or after the baseline reset.

This code example defines two conditions at which the sensors can be stuck. You can enable or disable these conditions:

1. Sensor raw count stuck between noise threshold and finger threshold.
2. Sensor is stuck at 'ON' state.

Figure 1 shows the PSoC Creator schematic for this code example.

Figure 1. TopDesign Schematic of CE229140_CapSense_CustomBaselineReset



Operation

1. Open the '*custom_baseline_reset.h.'* file to configure this project. Define '`RESET_CONFIGURATION`' to time-based reset or sample-based reset.

```
/* Configure to TIME_BASED or SAMPLE_BASED */
#define RESET_CONFIGURATION           TIME_BASED
```

2. Enable or disable the baseline reset for the two stuck conditions.

```
/* Raw count stuck between Noise Threshold and Finger Threshold */
#define SENSOR_STUCK_BETWEEN_NT_FT    ENABLE

/* Sensor status continuously stuck at '1' (ON) */
#define SENSOR_STUCK_AT_ONE           ENABLE
```

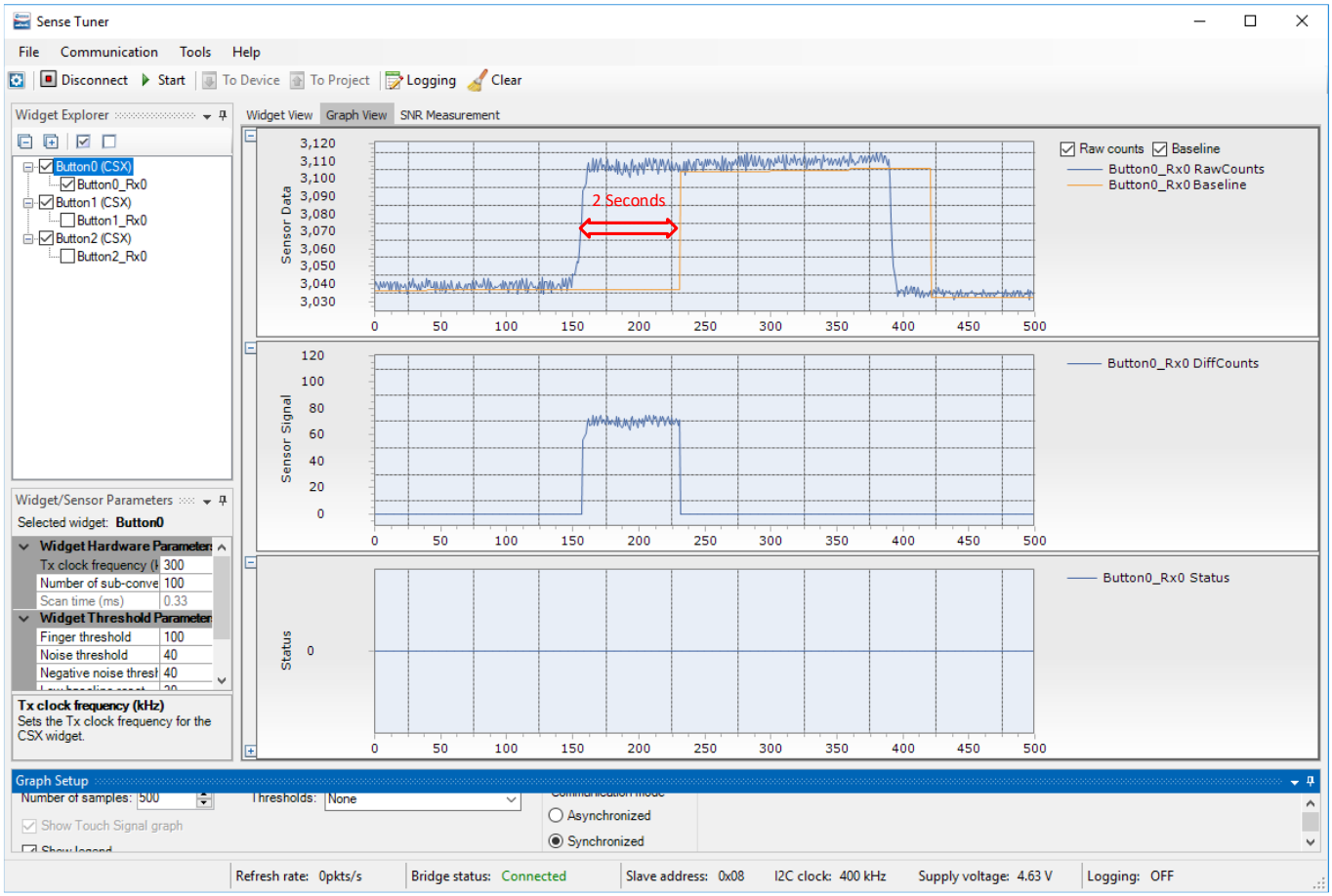
3. Define `SENSOR_STUCK_NT_FT_TIMEOUT` and `SENSOR_STUCK_AT_ONE_TIMEOUT` to set the reset timeout in 100 milliseconds or in number of samples based on the reset configuration.

```
/* Sensor Stuck Timeout in 100 milliseconds or in number of samples */
#define SENSOR_STUCK_NT_FT_TIMEOUT    (20u)
#define SENSOR_STUCK_AT_ONE_TIMEOUT   (20u)
```

4. Build the project CE229140_CapSense_CustomBaselineReset and program it into the PSoC 4100S Plus device. Choose **Debug** > **Program**. For more information on device programming, see PSoC Creator Help.
5. Launch the tuner GUI and monitor data in the tuner GUI. To set up the tuner GUI, see the CapSense Tuner section of the CapSense Component Datasheet.

   Go to the **Graph View** table of the Tuner GUI. As shown in Figure 2, when the raw count is stuck between Noise threshold (40) and Finger threshold (100), a baseline reset is applied after 2 seconds as per the user configuration.

Figure 2. Tuner: Graph View



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 3 – CSX buttons	For Basic Setting, see Figure 3. CapSense Basic Settings For Widget Settings, see Figure 4 and Figure 5.
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 Component datasheet.

Figure 3 highlights the non-default settings for the CapSense Component.

Figure 3. CapSense Basic Settings

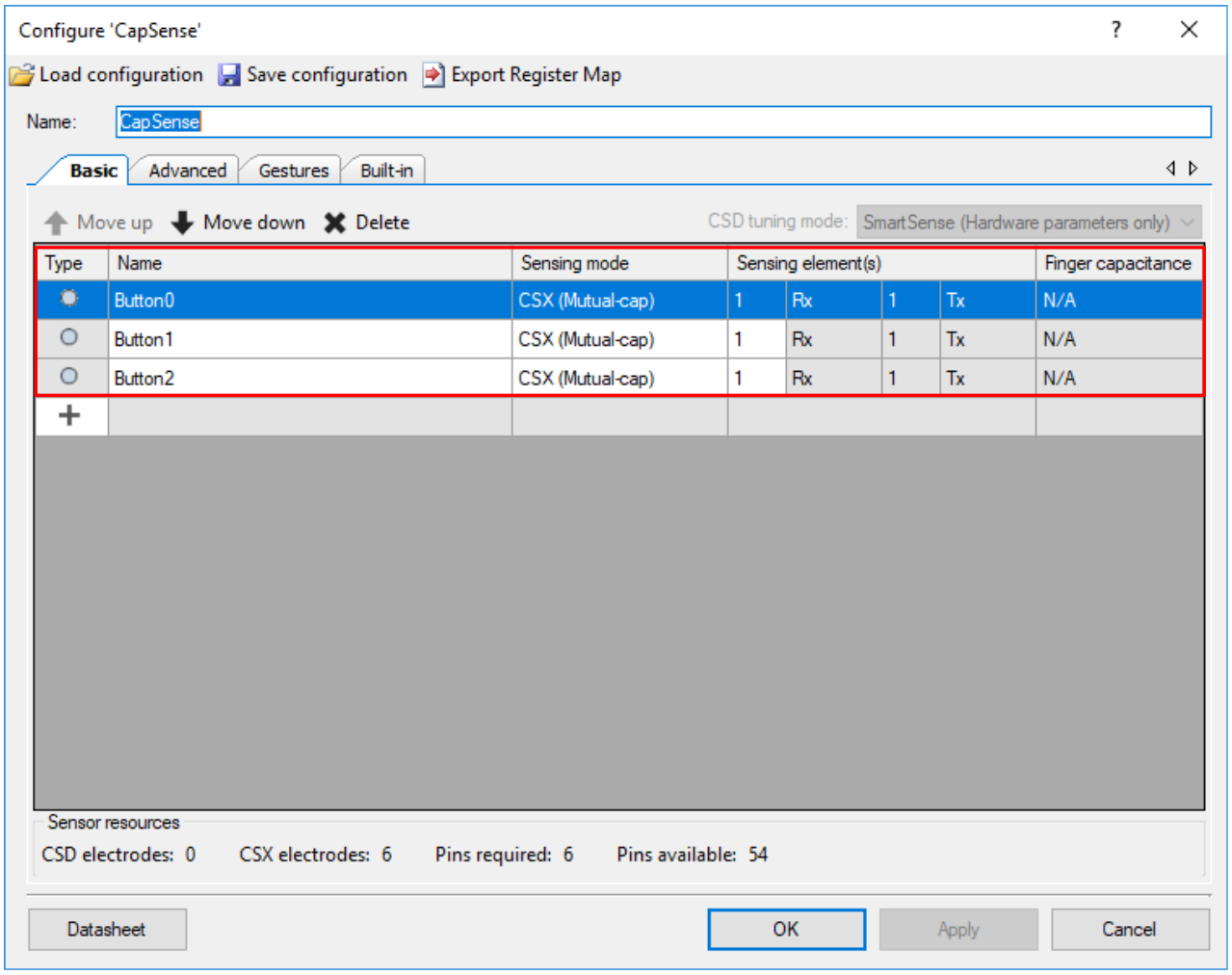


Figure 4. CapSense Button1 Widget Settings

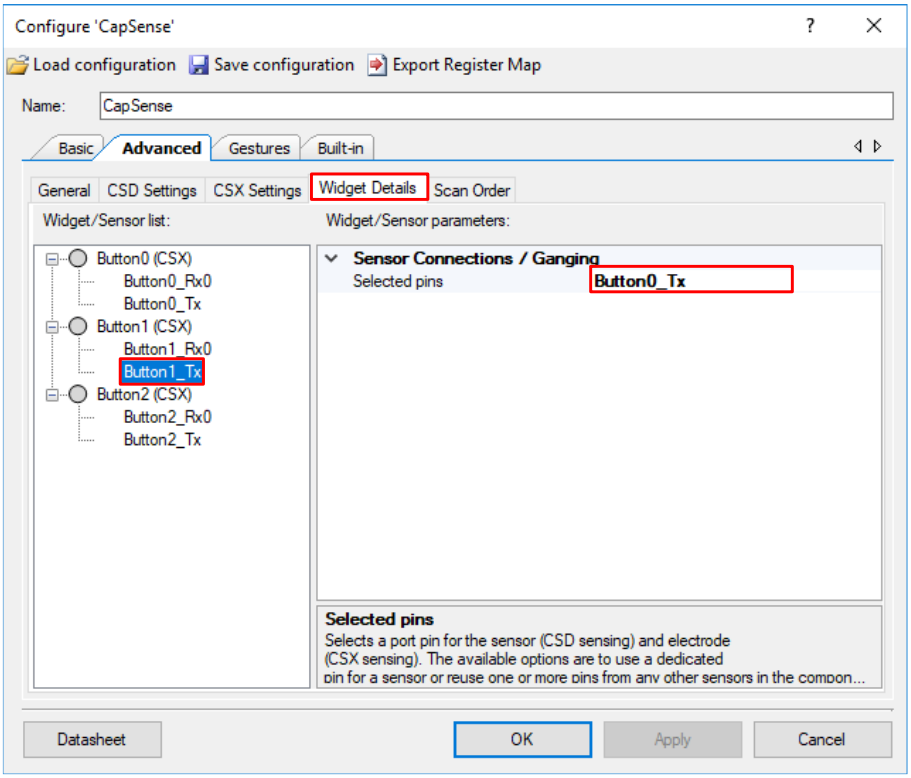


Figure 5. CapSense Button2 Widget Settings

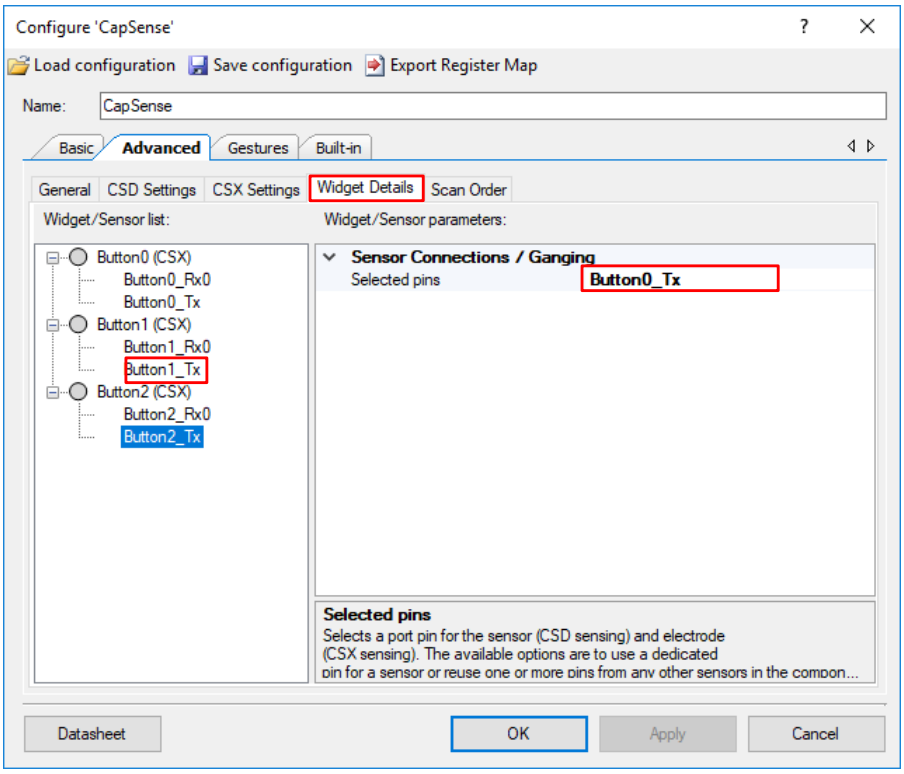


Table 2 shows the pin assignments for the project done in the **Pins** tab of the **Design Wide Resources** window for this code example. These assignments are compatible with the CY8CKIT-149 kit.

Table 2. 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[4]
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
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
SysTick	Supports a down counter with the capability to generate an interrupt
Device Documentation	
PSoC 4100S Plus Datasheet	PSoC 4 Technical Reference Manuals
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: CE229140 – PSoC 4 CapSense Custom Baseline Reset

Document Number: 002-29140

Revision	ECN	Submission Date	Description of Change
**	6756887	01/20/2020	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.