AN79973

PSoC 3 and PSoC 5LP CapSense CSD – IEC 60730 Class B Safety Software Library

Author: Taras Kuzo
Associated Project: Yes
Associated Part Family: CY8C38xx, CY8C55xx
Software Version: PSoC Creator™ 2.0
Related Application Note: AN78175

AN79973 details the self-check tests and their implementation details to match the IEC60730 standards that ensure reliable and safe operation of CapSense® CSD component in PSoC® 3 and PSoC 5LP devices.

Contents

1 Introduction....1
2 Overview of Annex H....2
3 Class B Requirements....3
4 Failure Mode and Effect Analysis (FMEA)....3
5 Operation Principles....8
5.1 CapSense CSD....8
5.2 Test on Sensor Shorts....8
5.3 Test on Sensor Disconnect....10
5.4 Guard Sensor Test....10
5.5 Shield Electrode Test....10
5.6 Modulator Component Test....10
5.7 External Resistor Mode Test....10
5.8 RC Time Measurement....11
5.9 IDAC Sourcing Mode Test....12
5.10 IDAC Sinking Mode Test....14
5.11 VDAC Test....15
5.12 Automatic Tuning Test....15
6 API Functions for PSoC 3 and PSoC 5LP....15
6.1 uint8 SelfTests_CSD_Shorts(void)....16
6.2 uint8 SelfTests_SaveBaselines(void)....16
6.3 uint8 SelfTests_CSD_Disconnect(void)....16
6.4 uint8 SelfTests_CSD_MeasureChannels....17
6.5 uint8 SelfTests_CSD_Modulator(void)....18
6.6 uint8 SelfTests_Shield_Shorts (void)....18
6.7 uint8 SelfTests_AutoTuning(void)....19
7 Summary....19
8 References....19
Worldwide Sales and Design Support....21

1 Introduction

The PSoC 3 and PSoC 5LP IEC60730 Class B Safety Software Library has been developed for self-testing CPU, memory, DAC/ADC, and additional components and peripherals. This application note focuses on only CapSense safety tests unique to PSoC devices. For standard CPU and device safety tests, see AN78175 - PSoC® 3 and PSoC 5 - IEC 60730 Class B Safety Software Library.

The International Electro-technical Commission (IEC) has developed safety standard IEC 60730-1 that discusses mechanical, electrical, electronic, environmental endurance, EMC, and abnormal operation for home appliances.

This application note focuses on Annex H Class B: Requirements for Electronic Controls. This portion of the standard details tests and diagnostic methods to ensure safe operation of embedded control hardware and software for home appliances. A Failure Mode and Effects Analysis (FMEA) was developed to generate the test requirements and is presented for reference.

See the following application notes for more information on PSoC 3 and PSoC 5LP:

- AN54181 - Getting Started with a PSoC® 3
- AN77759 - Getting Started with PSoC® 5LP

2 Overview of Annex H

Annex H of the IEC 60730-1 standard classifies appliances software into the following categories (see Appendix B: “IEC 60730-1 Table H.11.12.7”):

- Class A - Control functions that are not intended to be relied upon for the equipment’s safety such as humidity controls, lighting controls, timers, and switches.
- Class B - Control functions that are intended to prevent unsafe operation of the controlled equipment such as thermal cut-offs and door locks for laundry equipment.
- Class C - Control functions that are intended to prevent special hazards (such as an explosion caused by the controlled equipment). Examples are automatic burner controls and thermal cut-outs for closed, unvented water heater systems.

Large appliance products, such as washing machines, dishwashers, dryers, refrigerators, freezers, and cookers/stoves, tend to fall under the Class B classification. An exception is an appliance that might cause an explosion, such as a gas-fired controlled dryer that falls under class C.

The class B Safety Software Library and the example project described in this application note use the self-test and self-diagnostic methods that are in the Class B category. These methods use various measures to detect software-related faults and errors and respond to them. According to the IEC 60730-1 standard, manufacturers of automatic electronic controls must design software using one of the following structures:

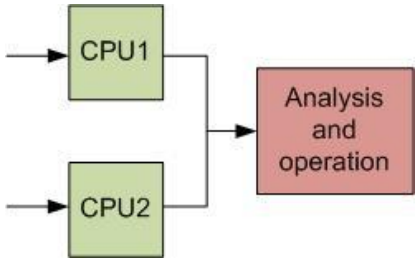
- Single channel with functional test
- Single channel with periodic self-test
- Dual channel without comparison

In the single channel structure with functional test, the software is designed using a single CPU to execute the functions as required. The functional test is executed after application start to ensure that all the critical features are functioning reliably.

In the single channel structure with periodic self-test, the software is designed using a single CPU to execute the functions as required. The periodic tests are embedded within the software, and self-tests occur periodically while the software is in the execution mode. The CPU is expected to regularly check the critical functions of the electronic control without conflicting with the end application’s operation.

In the dual channel structure without comparison, the software is designed using two CPUs to execute the critical functions. Before executing a critical function, both CPUs are required to share the information that they have completed their corresponding tasks. For example, when a laundry door lock is released, one CPU stops the motor spinning the drum and the other CPU checks the drum speed to verify that it has stopped, as shown in Figure 1.

Figure 1. Dual Channel Without Comparison Structure



The dual channel structure implementation is costlier because two CPUs (or two MCUs) are required. In addition, they are more complex because two devices are needed to regularly communicate with each other. The single-channel structure with a functional test is most commonly implemented today. However, appliance manufacturers are moving to the single channel structure with periodic self-test implementation.

3 Class B Requirements

According to IEC60730-1 Class B Annex H Table H.11.12.7 there is a list of the components that must be tested, depending on the software classification. Generally, each component offers optional measures to verify/test the corresponding components and provide flexibility for manufacturers. To provide Class B IEC 60730 compliance for the single channel structures, manufacturers of electronic controls are required to test the components listed in the following table.

Table 1. Components Required to be Tested for Single Channel Structures

Class B IEC 60730 Components Required to be Tested on Electronic Control (see Table H.11.12.7 in Annex H)	Fault/Error
1.1 CPU registers	Stuck at
1.3 CPU program counter	Stuck at
2. Interrupt handling and execution	No interrupt or too frequent interrupt
3. Clock	Wrong frequency
4.1 Invariable memory	All single bit faults
4.2 Variable memory	DC fault
4.3 Addressing (relevant to variable/invariable memory)	Stuck at
5.1 Internal data path Data	Stuck at
5.2 Internal data path Addressing (for expanded memory MCU systems only)	Wrong address
6.1 External communications Data	Hamming distance 3
6.2 External communications Addressing	Hamming distance 3
6.3 Timing	Wrong point in time/sequence
7.1 I/O Periphery	Fault conditions specified in Appendix B: “IEC 60730-1, H.27”)
7.2.1 Analog A/D and D/A converters	Fault conditions specified in Appendix B: “IEC 60730-1, H.27”)
7.2.2 Analog multiplexor	Wrong addressing

The CapSense CSD component uses IDAC, VDAC, registers, memory, clock, multiplexor, pins (I/O Periphery) and other components. Incorrect working of these components, pins shorts, C_{mod} shorts or Rb shorts can cause the CapSense CSD component to work incorrectly resulting in false or missed touches. The diagnostics described in this application note are intended to provide fail-safe functions in capacitive sensing devices where sensor faults may lead to safety concerns. White goods, automotive, and industrial electronic applications are examples of devices that require sensor fault diagnostics for safe operation.

4 Failure Mode and Effect Analysis (FMEA)

One of the main development goals for capacitive sensing applications is to enhance the reliability of devices with capacitive sensors in harsh or sensitive environments. A standard method of determining failure modes, their risks, and containment, is to perform a FMEA. The FMEA developed to generate the CapSense_CSD Class B test list is included for reference.

The FMEA is limited to only cover CapSense_CSD component failures and not general device failures. General device failures are covered in the IEC 60730 specification, and PSoC 3- and PSoC 5LP-specific tests are provided in application note AN78175 - PSoC® 3 and PSoC 5 - IEC 60730 Class B Safety Software Library. The FMEA assumes that AN78175 class B tests are already being conducted to ensure general purpose hardware resources like CPU registers, SRAM, flash, clocking, and interrupts are operating correctly. These general failures covered by the class B library that could impact CapSense operation are not specifically addressed.

The FMEA provided in Table 2 follows a standard FMEA template with weighting criteria listed in Table 3, Table 4, and Table 5.

Table 2. Provides Failure Mode and Effect Analysis (FMEA)

Item/ Function	Failure Mode	Failure Effect	Severity	Failure Cause	Occurrence	Failure Control	Detection	RPN	Recommended Actions
Sensors and Guard sensor	Short to supply	Sensor does not report touch	7	PCB short	3	Check for Short	1	21	uint8 SelfTests_CSD_Shorts(void)
	Short to other sensor	Sensor(s) does not report touch	7	PCB short	3	Check for Short	1	21	uint8 SelfTests_CSD_Shorts(void)
	Open	Sensor does not report touch	7	PCB open	3	Check for open	1	21	uint8 SelfTests_CSD_Disconnect (void)
	I/O configuration	Sensor does not report touch	7	Upset event	3	Check for baseline in range	1	21	uint8 SelfTests_CSD_MeasureChannels
	I/O damage	Sensor does not report touch	7	ESD damage	2	Check for baseline in range	1	14	uint8 SelfTests_CSD_MeasureChannels
Shield	Short to supply	False or missed touches with water drops	2	PCB short	3	Check for Short	1	6	uint8 SelfTests_Shield_Shorts (void)
		Degraded analog operation due to large supply current fluctuation	6	PCB short	3	Check for Short	1	18	uint8 SelfTests_Shield_Shorts (void)
	Short to other sensor	False or missed touches with water drops	2	PCB short	3	Check for Short	1	6	uint8 SelfTests_Shield_Shorts (void)
		Shorted sensor does not report touch	7	PCB short	3	Check for Short	1	21	uint8 SelfTests_Shield_Shorts (void)
	Open	False or missed touches with water drops	2	PCB open	3	Test shield with second I/O	1	6	Test Shield pad with second I/O using separate trace. Confirm I/O shorted to Shield.
	I/O configuration	False or missed touches with water drops	2	Upset event	3	Test shield with second I/O	1	6	Test Shield pad with second I/O using separate trace. Confirm I/O shorted to Shield.

Item/ Function	Failure Mode	Failure Effect	Severity	Failure Cause	Occurrence	Failure Control	Detection	RPN	Recommended Actions
	I/O damage	False or missed touches with water drops	2	ESD damage	2	Test shield with second I/O	1	4	Test Shield pad with second I/O using separate trace. Confirm I/O shorted to Shield.
C_{mod}	Short to supply	Sensors do not report touch on affected channel	7	PCB short	3	Check for Short	1	21	uint8 SelfTests_CSD_Shorts(void)
		Degraded analog operation due to large supply current fluctuation	6	PCB short	3	Check for Short	1	18	uint8 SelfTests_CSD_Shorts(void)
	Open	Sensors do not report touch on affected channel	7	PCB open	3	Check for open	1	21	uint8 SelfTests_CSD_MeasureChannels
	Wrong value	Sensors do not report touch on affected channel	7	PCB assembly error	3	Check for baseline in range	2	42	uint8 SelfTests_CSD_MeasureChannels
		False or missed touches	4	PCB assembly error	3	Check for baseline in range	2	24	uint8 SelfTests_CSD_MeasureChannels
	I/O configuration	Sensors do not report touch on affected channel	7	Upset event	3	Check for baseline in range	1	21	uint8 SelfTests_CSD_MeasureChannels
	I/O damage	Sensors do not report touch on affected channel	7	ESD damage	2	Check for baseline in range	1	14	uint8 SelfTests_CSD_MeasureChannels
Rb	Short to supply	Sensors do not report touch on affected channel	7	PCB short	3	Check for Short	1	21	uint8 SelfTests_CSD_Shorts(void)
		Degraded analog operation due to large supply	6	PCB short	3	Check for Short	1	18	uint8 SelfTests_CSD_Shorts(void)

Item/ Function	Failure Mode	Failure Effect	Severity	Failure Cause	Occurrence	Failure Control	Detection	RPN	Recommended Actions
		current fluctuation							
	Open	Sensors do not report touch on affected channel	7	PCB open	3	Check for open	1	21	uint8 SelfTests_CSD_MeasureChannels
	Wrong value	Sensors do not report touch on affected channel	7	PCB assembly error	3	Check for baseline in range	2	42	uint8 SelfTests_CSD_MeasureChannels
		False or missed touches	4	PCB assembly error	3	Check for baseline in range	2	24	uint8 SelfTests_CSD_MeasureChannels
	I/O configuration	Sensors do not report touch on affected channel	7	Upset event	3	Check for baseline in range	1	21	uint8 SelfTests_CSD_MeasureChannels
	I/O damage	Sensors do not report touch on affected channel	7	ESD damage	2	Check for baseline in range	1	14	uint8 SelfTests_CSD_MeasureChannels
CapSense device configuration	CapSense configuration register upset	False or missed touches	4	Upset event	2	Check for baseline in range	1	8	uint8 SelfTests_CSD_MeasureChannels
	CapSense engine hardware damage	False or missed touches	4	ESD damage	1	Check for baseline in range	1	4	uint8 SelfTests_CSD_MeasureChannels
CapSense SRAM	SRAM/register upset	False or missed touches	4	Upset event	3	Debounce touches	3	36	Debounce performed in CapSense_CSD component
		False or missed touches	4	Upset event	3	Check for baseline in range	3	36	uint8 SelfTests_CSD_MeasureChannels
Tuning parameters	Auto tune parameters out of range	Sensor does not report touch	7	Internal configuration(1), I/O(2), or PCB(3) error	3	Check tuning parameters	2	42	uint8 SelfTests_AutoTuning(void)
		False or missed touches	4	Internal configuration(1), I/O(2), or PCB(3) error	3	Check tuning parameters	2	24	uint8 SelfTests_AutoTuning(void)
VDAC	Wrong or no output	Sensors do not report touch on	7	Upset event changes configuration	1	Check modulator operation	2	14	uint8 SelfTests_CSD_Modulator (void)

Item/ Function	Failure Mode	Failure Effect	Severity	Failure Cause	Occurrence	Failure Control	Detection	RPN	Recommended Actions
		affected channel							
		False or missed touches	4	Upset event changes configuration	1	Check modulator operation	2	8	uint8 SelfTests_CSD_Modulator (void)
		False or missed touches	4	Upset event changes configuration	1	Check for baseline in range	1	4	uint8 SelfTests_CSD_MeasureChannels
IDAC	Wrong or no output	Sensors do not report touch on affected channel	7	Upset event changes configuration	1	Check modulator operation	2	14	uint8 SelfTests_CSD_Modulator (void)
		False or missed touches	4	Upset event changes configuration	1	Check modulator operation	2	8	uint8 SelfTests_CSD_Modulator (void)
		False or missed touches	4	Upset event changes configuration	1	Check for baseline in range	1	4	uint8 SelfTests_CSD_MeasureChannels
Comparator	Wrong output	Sensors do not report touch on affected channel	7	Upset event changes configuration	1	Check modulator operation	2	14	uint8 SelfTests_CSD_Modulator (void)
		Sensors do not report touch on affected channel	7	Upset event changes configuration	1	Check for baseline in range	1	7	uint8 SelfTests_CSD_MeasureChannels
Timer	Wrong output	Sensors do not report touch on affected channel	7	Upset event changes configuration	1	Check for baseline in range	2	14	uint8 SelfTests_CSD_MeasureChannels
Interrupt	ISR code not executed	Sensors do not report touch on affected channel	7	Upset event changes configuration	1	Check for baseline in range	1	7	uint8 SelfTests_CSD_MeasureChannels

Table 3. Occurrence Scoring

1	No known occurrences on similar products or processes
2/3	Low (relatively few failures)
4/5/6	Moderate (occasional failures)
7/8	High (repeated failures)
9/10	Very high (failure is almost inevitable)

Table 4. Severity Scoring

1	No effect
2	Very minor (only noticed by discriminating customers)
3	Minor (affects very little of the system, noticed by average customer)
4/5/6	Moderate (most customers are annoyed)
7/8	High (causes a loss of primary function; customers are dissatisfied)
9/10	Very high and hazardous (product becomes inoperative; customers angered; the failure may result unsafe operation and possible injury)

Table 5. Detection Scoring

1	Certain - fault will be caught on test
2	Almost Certain
3	High
4/5/6	Moderate
7/8	Low
9/10	Fault will be passed to customer undetected

Failure mode severity occurrence and detection weighting is based on hundreds of CapSense designs; the resulting highest risk items correlate well with actual failures. The most common failure source is with the physical PCB, touch surface or connection to sensors. Physical failures are most often opens or shorts followed by a small number of PCB processing, assembly, or component value errors. Fortunately, these types of failures are most easily detected at manufacturing test and rarely occur during end customer use. As seen in the FMEA, failure detection is very good with ESD events and component value errors are the highest risk. Both ESD and component errors can be designed and tested prior to device operation, further reducing risk.

5 Operation Principles

The class B CapSense CSD Safety Software Library described in this application note can be used with PSoC 3 and PSoC 5LP devices. The library includes APIs intended to maximize the application reliability through fault detection.

All self-tests can be applied by adding an appropriate API function and **.c* and **.h* files from the Class B CapSense CSD Safety Software Library.

Self-tests can be executed after device startup, before the main loop. This provides an opportunity to check whether the chip is suitable for operation.

Also, self-tests must be executed periodically while the device is working. This provides an opportunity to make sure the chip was not damaged during operation.

The following sections describe self-test and the implementation details for each test according to IEC 60730-1 Class B Annex H. In addition, each section lists the APIs required to execute a corresponding test for the supported architectures.

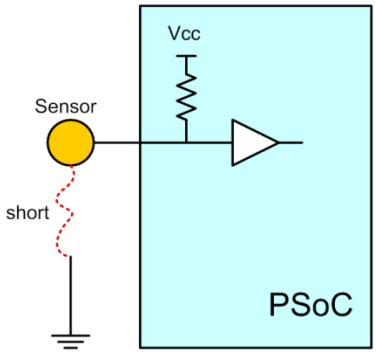
5.1 CapSense CSD

PSoC 3 and PSoC 5 devices have a capacitive sensing feature called CapSense. This feature allows users to take advantage of the capacitive properties of their fingers to toggle buttons, sliders, and wheels on a clean and modern interface. Touchpads and touchscreens are common examples of capacitive sensing interfaces. The underlying principle of these technologies is the measurement of capacitance between a plate (the sensor) and its environment.

5.2 Test on Sensor Shorts

In normal operating conditions, the sensor-to-ground, sensor-to-sensor, and sensor-to-VCC resistances are higher. To detect any shorts, actual resistance values are compared to the PSoC internal pull-up resistors. Figure 2 shows a simplified schematic for the sensor-to-ground short detection test.

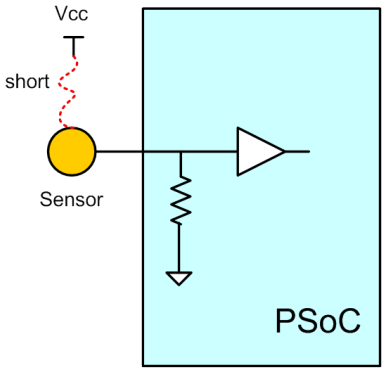
Figure 2. Sensor-to-Ground Short Detection Schematic



The sensor pin is configured in the resistive pull-up drive mode. In normal conditions, the CPU reads a logical one because of the pull-up resistor. If the sensor is connected to ground through a small resistance, the input level is recognized as a logical zero.

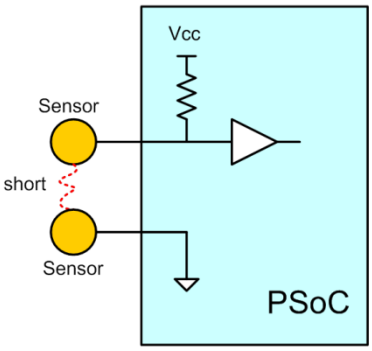
Sensor-to-V_{CC} shorting is detected by a similar method. Figure 3 shows the corresponding schematic.

Figure 3. Sensor-to-VCC Short Detection Schematic



In this case, the sensor pin is configured in the resistive pull down drive mode. The input level is zero in normal conditions. Figure 4 shows the schematic for the sensor-to-sensor short check.

Figure 4. Sensor-to-Sensor Short Detection Schematic



All sensor pins are connected to ground internally by writing zero values to the data registers in the strong drive mode. The measured sensor pin is configured to the resistive pull-up mode. The input level is the logical one for good sensors. A logical zero signifies a detected short.

5.3 Test on Sensor Disconnect

Another task is to detect sensor disconnects. Unfortunately, this task cannot be accomplished with I/O hardware methods, as in the case of short detection. Sensor disconnects require a different method.

The detection method is based on observing the sensor baseline data. If a sensor is disconnected from the sensing IC, the area of copper attached to the sensing IC is smaller than expected. This leads to a significantly lower raw count and baseline values due to the decreased capacitance. To detect sensor disconnects, the baseline values under normal conditions are stored in the internal EEPROM. The actual baseline values are compared to the stored values. If the actual value is less than the stored value and the difference exceeds the error threshold, a system fault is detected. The threshold value must not be too small to avoid false fault triggering based on environmental condition changes like temperature. In addition, this value must not be too large to ensure reliable disconnect detection. This implementation provides two predefined threshold values: 25% and 12.5% of the stored baseline value. These values are sufficient for most applications. Choose the actual value in your system by running tests on real boards in real systems.

5.4 Guard Sensor Test

The guard sensor is a specific sensor that should surround all the sensors and is commonly used in water-proof applications to detect large quantity of water on the surface and then disable the other sensors.

In the CSD component, the guard sensor is scanned as a simple sensor and is automatically added to the sensors table when enabled in the component wizard. The same Sensor Shorts and Sensor Disconnect tests are applied to the guard sensor.

5.5 Shield Electrode Test

The shield electrode provides the shield waveform to a fill that surrounds all sensors. The shield waveform reduces the effect of small amounts of water so that sensors may still be used. The amplitude of the shield signal is equal to V_{DDIO}. The shield-to-ground, shield-to-V_{CC}, shield-to-sensor, and shield-to-shield shorts tests can be applied to the shield electrode.

5.6 Modulator Component Test

The CapSense CSD Sigma-Delta modulator requires a precision current source for detecting a touch on the sensors. There are three CSD modulator operating modes in the CSD component:

- IDAC Sourcing (default) – The IDAC sources current into the modulation capacitor CMOD. The analog switches are configured to alternate between the modulation capacitor CMOD and GND providing a sink for the current. IDAC Sourcing is recommended for most designs because it provides the greatest signal-to-noise ratio of the three methods, but it may require an additional VDAC resource to set the Vref level that the other modes do not require.
- IDAC Sinking – The IDAC sinks current from the modulation capacitor CMOD. The analog switches are configured to alternate between the VDD and the modulation capacitor CMOD providing a source for the current. This works well in most designs, although the SNR is generally not as high as the IDAC Sourcing mode.
- External Resistor – This functions the same as the IDAC sinking configuration except the IDAC is replaced with a bleed resistor to ground, Rb. The bleed resistor is connected between the modulation capacitor, CMOD and a GPIO. The GPIO is configured to Open-Drain Drives Low drive mode allowing CMOD to be discharged through Rb. This mode requires the fewest analog resources and should only be used when needed because of resource constraints. Since this mode does not require an IDAC or VDAC, it can result in the lowest power configuration of the component. This is useful if power is a critical system consideration.

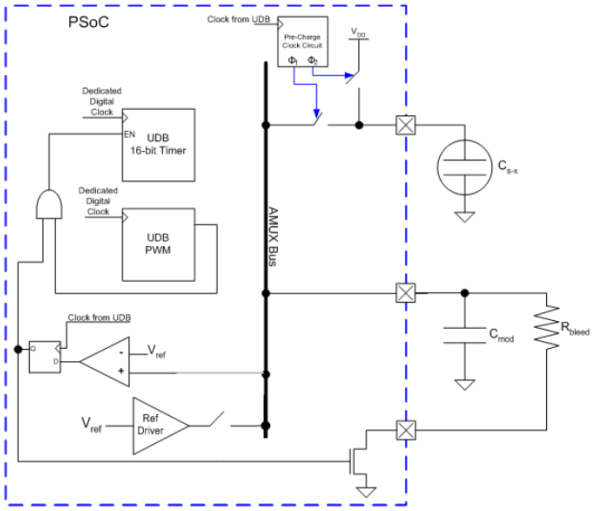
IDAC Sinking and IDAC Sourcing require use of the hardware IDAC on the PSoC device. The External Resistor uses a user-supplied resistor on the PCB rather than an IDAC and is useful in IDAC constrained applications.

Self-tests for all three modulator modes are described in the following sections.

5.7 External Resistor Mode Test

In this instance, an external Rb is used. C_{mod} and Rb must be tested.
Using an external bleed resistor, Rb functions the same as the IDAC Sinking configuration except the IDAC is replaced by a resistor-to-ground, Rb. The bleed resistor is physically connected between CMOD and a GPIO. The GPIO is configured in the "Open-Drain Drives Low" drive mode. This mode allows the CMOD to be discharged through Rb and is shown in Figure 5.

Figure 5. External Resistor Mode



Rb and C_{mod} external components can also cause errors in the capacitance measurement. For example, these components could be shorted, opened, or the wrong value. However, the techniques described for sensor diagnostics are not applicable in this case, because the C_{mod} pin appears to be disconnected when tested at DC.

The simplest method to test both C_{mod} and R_b simultaneously is to estimate the RC time constant during C_{mod} discharge through R_b. This measurement requires minimal hardware reconfiguration and can be achieved easily due to the ability to reconfigure the PSoC.

5.8 RC Time Measurement

To estimate the RC time measurement, the RC constant needs to measure the time of C_{mod} discharging from V_{cc} to Vref through R_b.

If C_{mod} charges to V_{cc} and then discharges through R_b, the voltage on the capacitor changes according to Equation 1.

$$V_C\,(t) = V_{CC}\,e^{-t/\tau} \qquad \text{Equation 1}$$

In this equation, the time constant (t) = R_bC_{mod}.

The capacitor discharges until its voltage drops below the comparator reference voltage (Vref). This reference voltage depends on the CSD Component settings. In the example case, the reference voltage is equal to 1.024 V. The time needed to discharge the capacitor from V_{cc} to 1.024 V is determined according to Equation 2 or Equation 3.

$$t = ln(V_{CC}/V_{ref}) * T \qquad \text{Equation 2}$$

$$t = 1.16 * T \qquad \text{Equation 3}$$

Figure 6. Oscilloscope Connections to Measure Discharge Time on C_{mod}

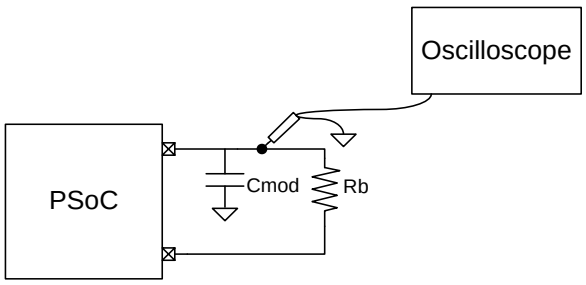


Figure 7. Voltage on C_{mod} during Discharge



Figure 6 shows oscilloscope connections to C_{mod} for measuring discharge time on C_{mod}.

The scope image in Figure 7 shows the proposed RC test method. In the first stage, C_{mod} charged to V_{CC} and, after a pause, discharges to V_{ref}. The measured discharge time is about 58 µs. This agrees with the previous equations for C_{mod} = 22 nF and Rb = 2.2 k, as shown in Equation 4.

$$t = 1.16 * 2.2\,k * 22\,nf = 57\,\mu s$$ Equation 4

In the software, it is more convenient to measure the discharge time in CPU cycles, as shown in Equation 5.

$$\mathrm{Nclk} = 1.16 * Rb * Cmod * CPU_Clock$$ Equation 5

For example, if C_{mod} = 22 nF, R_b = 2.2 k, and CPU_Clock = 24 MHz, the measured discharge time is N_{CLK} = 1347 CPU cycles.

The software needs to monitor the discharge if the voltage on C_{mod} drops to comparator V_{ref}. For example:

```
uint8 Result = 0u;
…
while((Result < 0xFFu) && (CSD_CompCH0_GetCompare()))
{
    Result ++;
}
```

One pass through this ‘while' loop needs 49 CPU cycles for PSoC 3 with CPU_Clock = 24 MHz.

To calculate the discharge time in CPU cycles, use Equation 6.

$$\mathrm{Nclk} = \mathrm{Result} * 49\mathrm{cycles}$$ Equation 6

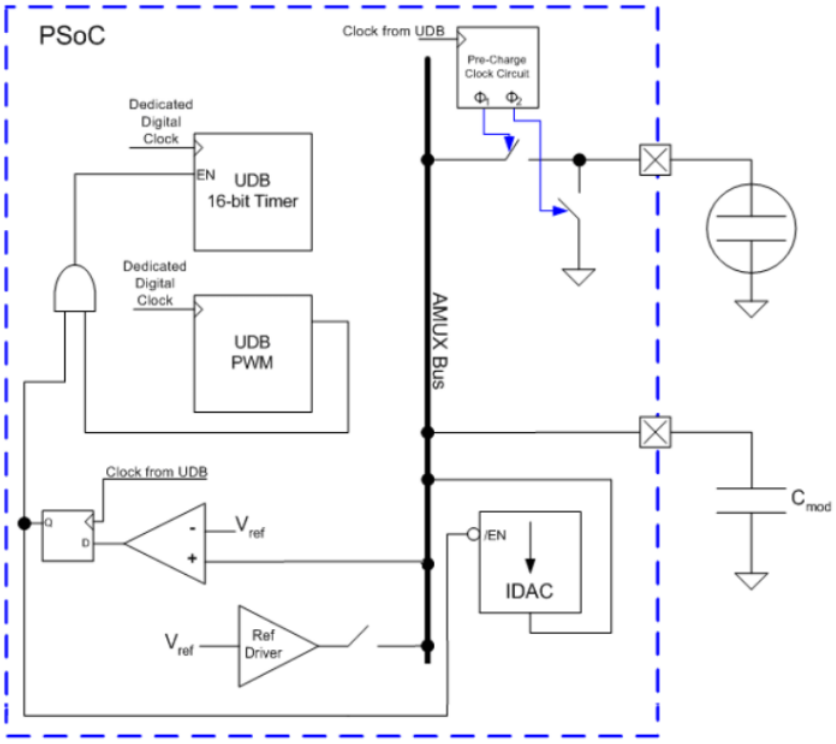
In most cases, when an absolute time constant value is not required, it is enough to compare the resultant value to zero and FFh. If the result is zero, it means that C_{mod} is open or R_b is shorted. If the result is FFh, it means that R_b is open. Other values correspond to normal operation.

5.9 IDAC Sourcing Mode Test

In this case, the IDAC is enabled and no external Rb is used; therefore, only the IDAC and C_{mod} require testing.

The sensor switch stage is configured to alternate between GND and the AMUX bus that connects to the modulation capacitor. In this configuration, the IDAC is configured to source current to the sensor as showed in Figure 8.

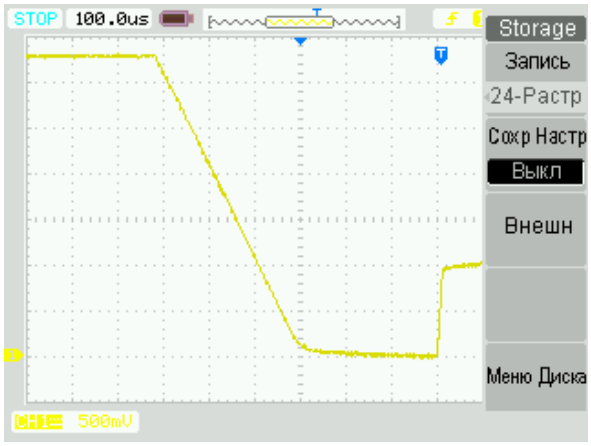
Figure 8. IDAC Sourcing Mode



To test C_{mod} in this mode, the same algorithm as the External Resistor mode can be used. The simplest method to test C_{mod} discharge is to discharge through the IDAC. The CSD component and IDAC need reconfiguration in this case, but this can be achieved easily due to the ability to reconfigure the PSoC.

To implement this test, all the sensors and V_{ref} need to be disconnected from the analog bus and the CapSense Buffer disabled. Reconfigure the IDAC to the sinking mode to provide a discharge sink for C_{mod}. Only Cmode, the comparator and IDAC can be connected to the AMUX bus. Charge C_{mod} to V_{CC}, set the C_{mod} pin to Analog Hi-Z and measure the discharge time.

Figure 9. Voltage on C_{mod} during Discharge in IDAC Sourcing Mode

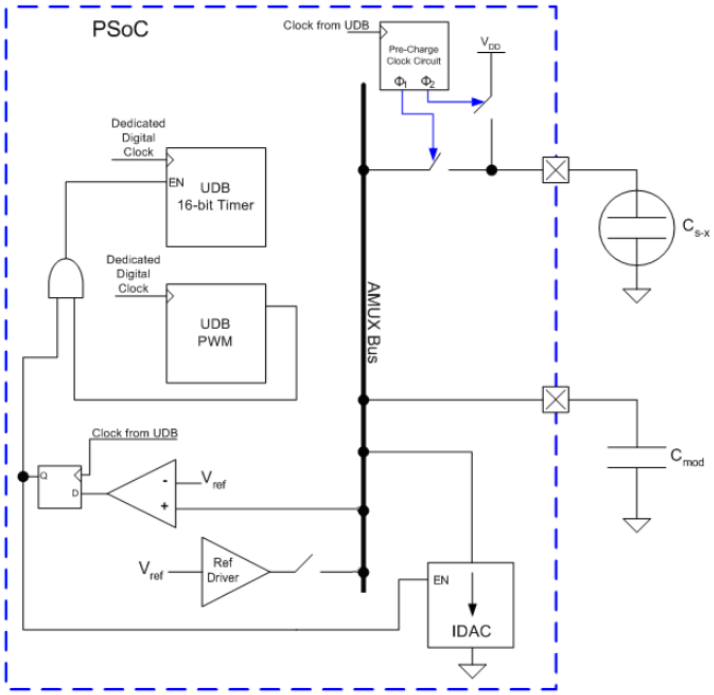


5.10 IDAC Sinking Mode Test

In this case the IDAC is enabled and no external Rb is used; therefore, only the IDAC and C_{mod} require testing.

The sensor switch stage is configured to alternate between the GND and the AMUX bus that connects to the modulation capacitor. In this configuration, the IDAC is configured to source current to the sensor as shown in Figure 10.

Figure 10. IDAC Sinking Mode



To test C_{mod} in this mode, the same algorithm as in previous modes is used. The simplest method to implement the C_{mod} test is to discharge through the IDAC. The CSD component and IDAC need reconfiguration in this case, but this can be achieved easily due to the ability to reconfigure the PSoC.

To implement this, test all the sensors. V_{ref} should be disconnected from the analog bus and the CapSense Buffer disabled. Reconfigure the IDAC to the sinking mode to provide a discharge sink for C_{mod}. Only Cmode, the comparator and IDAC should be connected to the AMUX bus. Charge C_{mod} to V_{cc}, set the C_{mod} pin to Analog High Z and measure the discharge time.

Figure 11. Voltage on C_{mod} during Discharge in IDAC Sinking Mode



5.11 VDAC Test

A VDAC can be used instead of V_{ref} for the IDAC source mode, which allows adjustment of the reference voltage using a Voltage DAC to maximize the available range. This requires a VDAC device resource.

In this case, the VDAC allows additional tests and checks of the measuring channels. If the VDAC value changes, the voltage reference on the comparator changes, and the raw data should also change.

The smaller the VDAC value is, the smaller raw data values must be and vice versa. The VDAC value is 8-bits, and it can change from 0 to 255.

This test requires only one sensor scan in each channel.

For best performance, the tuning option for this test must meet the following requirements:

- If the VDAC value is set to “255” the raw data must not be in saturation after the scan
- If the VDAC value is set to “0”, the raw data must not be “0” after the scan

By meeting these two requirements, the measuring channels can be tested over the whole VDAC range.

But if these requirements cannot be met, the tested VDAC range can be set from 0 to 127, from 127 to 255 or any other range.

This self-test checks the complete measurement channels and also detects errors if C_{mod} or Rb are shorted or open.

5.12 Automatic Tuning Test

This option provides automatic tuning of the CapSense CSD component. Automatic tuning is performed each time after device start up. If operation occurs incorrectly with PSoC, C_{mod}, sensor/sensors or Rb during the device idle state, the tuning parameters will be incorrect after the device starts up and performs automatic tuning. The simplest method to detect such errors is to confirm that the tuning parameters are within the predefined range after automatic tuning. There are two arrays in the CSD component with tuning parameters:

- CSD_idacSettings[ ] – contains IDAC values for all sensors. The IDAC value is 8-bit and can be from 0 to 255.
- CSD_AnalogSwitchDivider – contains prescaler values for all scanning slots. The prescaler value is 8-bit and can be from 0 to 255.

When the sensor is shorted-to-ground, the IDAC value is 0xFF; the prescaler is the maximum value possible and increases. The maximum possible prescaler depends on the sensor, but it is usually 6, 7 or 8 when the sensor is shorted and 2, 3 or 4 for a good sensor.

When sensor is shorted to $V_{CC,}$ the IDAC value is 0x00 and the prescaler is the minimum value possible and decreases.

In addition, these parameters must be approximately the same for the same panels. For example, there are 1000 units of the same good panels. Auto tuning uses the same tuning algorithm for all panels and the parameters will be approximately the same with some tolerance from the average values. Tolerance for IDAC is 2-3 points; for prescaler it is 1 point. If one of sensors is disconnected or shorted to another sensor the IDAC or prescaler change more than the tolerance.

If C_{mod} or Rb is shorted-to-ground or V_{CC} the IDAC and prescaler will be the same as for shorted sensors (0xFF or 0x00).

Using these practices, the IDAC and prescaler can be checked in the predefined range after auto tuning.

6 API Functions for PSoC 3 and PSoC 5LP

The FMEA functions are implemented as a standalone library. The library consists of a C and header file. The FMEA library contains direct calls to the CSD User Module API functions. The instance of the CSD Component in the PSoC Creator must be “CSD” so that these functions calls work correctly.

Note If C_{mod} and Rb are used, they must be located on pins according to the wizard settings in PSoC Creator, otherwise an error is reported.

Note The CapSense CSD component must be named “CSD”.

6.1 uint8 SelfTests_CSD_Shorts(void)

Located in: *SelfTest_Sensors.c*

SelfTest_ Sensors.h

Description: The function checks all the sensors and detects sensors shorts-to-ground, V_{CC} or other sensors. This function also checks the guard sensor for shorts if it is used in the design.

This function should be called after all scanning is completed.

Parameters: None

ReturnValue: Returns an error code as shown in Table 6.

Table 6. Error Code Descriptions for SelfTests_CSD_Shorts()

Error Code	Value	Description
OK_STATUS	0x00	No errors detected
SHORT_TO_VCC	0x01	Short to V_{CC} is detected
SHORT_TO_GND	0x02	Short to GND is detected
SHORT_TO_OTHER	0x04	Sensor-to-sensor short is detected

The number of the sensor that caused the error is stored in the global variable CSD_bSensorNum.

Side Effects: None

6.2 uint8 SelfTests_SaveBaselines(void)

Located in: *SelfTest_Sensors.c*

SelfTest_ Sensors.h

Description: Stores the current (reference) sensors and guard sensor (if used) baseline values to the selected EEPROM area. It writes CSD_TOTAL_SENSOR_COUNT data words that contain the baseline values and one additional word '0x55AA' as a semaphore. This semaphore allows for checking of the EEPROM to see if it contains the stored values. The Flash block number used to save data is defined with the constant EEPROM_ADR in *SelfTest_ Sensors.h*.

Call this function only once after initialization of the CSD component and sensors baseline before `SelfTests_CSD_Disconnect()` test function. This function store sensors baseline which are needed for the disconnect detection algorithm's correct operation (`SelfTests_CSD_Disconnect()` function).

Parameters: None

ReturnValue: Returns an error code as shown in Table 7.

Table 7. Error Code Descriptions for SelfTests_SaveBaselines ()

Error Code	Value	Description
OK_STATUS	0x00	Baseline values greater than zero: Sensor is OK.
ERROR_BASELINE	0x01	Baseline value is Zero: Indicates an error.
ERROR_NOT_WRITE	0x02	Error during writing to EEPROM

Side Effects: This function needs (CSD_TOTAL_SENSOR_COUNT*2+2) bytes in EEPROM.

6.3 uint8 SelfTests_CSD_Disconnect(void)

Located in: *SelfTest_Sensors.c*

SelfTest_ Sensors.h

Description: The function detects sensor disconnects. This function also detects guard sensor disconnects if it is used in the design.

The function compares all the sensor baselines to the stored reference values. If the baseline is below tolerance, an error code is returned. Tolerance is defined by the constant, BASELINES_TOLERANCE. The allowed values are:

TOLERANCE_12_5_PERCENT (tolerance is 12.5%)

TOLERANCE_25_PERCENT (tolerance is 25%)

This function should be called after scanning is completed. `SelfTests_SaveBaselines()` function needs to be called once before this function for correct operation.

Parameters: None

ReturnValue: Returns an error code as shown in Table 8.

Table 8. Error Code Descriptions for SelfTests_CSD_Disconnect()

Error Code	Value	Description
OK_STATUS	0x00	No errors detected
NO_FLASH_DATA	0x08	Reference baselines values are not stored in flash memory
BASELINE_DOWN	0x10	Baseline is below allowed value

The number of the sensor with the fault is stored in the global variable CSD_bSensorNum.

Side Effects: None

6.4 uint8 SelfTests_CSD_MeasureChannels

Located in: *SelfTest_CSD.c*
SelfTest_CSD.h

Description: The function detects an error in the measurement channels; it also detects an error if C_{mod} or Rb (if used) is shorted.

The function can be used in the IDAC source mode when V_{ref} from VDAC is enabled in the Wizard. The function sets different voltage references for the comparator and observes the raw data change. The raw data should change with different voltage references. The smaller the VDAC value is, the smaller raw data values must be and vice versa. The VDAC value is 8-bit, and can be from 0 to 255.

To choose the VDAC values, the user must fill the Vef array located in the *SelfTest_CSD.c* file.

uint8 vref[ ] = {8u, 16u, 32u, 64u, 128u }; (example values)

The function scans only one sensor for each channel.

This function is called after scanning is completed.

Parameters: None

ReturnValue: Returns an error code as shown in Table 9.

Table 9. Error Code Descriptions for SelfTests_CSD_MeasureChannels()

Error Code	Value	Description
OK_STATUS	0x00	No errors detected
ERROR_CHANNEL_0	0x01	Error detected in measuring channel0
ERROR_CHANNEL_1	0x02	Error detected in measuring channel1

Side Effects: The interrupts are not disabled during measurement because this function is not intended for precise measurement, only for failure detection. If precise measurement is required, disable the interrupts before calling the `SelfTests_CSD_MeasureChannels()`.

6.5 uint8 SelfTests_CSD_Modulator(void)

Located in: *SelfTest_CSD.c*
SelfTest_CSD.h

Description: The function detects an error in C_{mod} when using Rb or IDAC.

Disconnects all the sensors and Vref from the analog bus and disables the CapSense Buffer. Charges C_{mod} to V_{cc} and measures the discharge time. The result is an integer value. To calculate the discharge time in the CPU cycles, use Equation 7.

$$Nclk = Result * 49cycles \quad \text{Equation 7}$$

For more information, see the RC Time Measurement section.

DISCH_TIME_MIN and DISCH_TIME_MAX define minimum and maximum allowed C_{mod} discharge time for the test.

In most cases when the absolute time constant value is not required, it is enough to compare the result value to zero and FFh. If the result is zero, C_{mod} is open or R_b or the IDAC is shorted. If the result is FFh, R_b or the IDAC is open. Other values correspond to normal operation.

This function is called after scanning is completed.

Parameters: None

ReturnValue: Returns an error code as shown in Table 10.

Table 10. Error Code Descriptions for SelfTests_CSD_Modulator()

Error Code	Value	Description
OK_STATUS	0x00	No errors detected
CH0_CMOD_DISCH_TO_FAST	0x01	C_{mod} discharge to fast in channel0. May be shorted to GND
CH1_CMOD_DISCH_TO_FAST	0x02	C_{mod} discharge to fast in channel1. May be shorted to GND
CH0_CMOD_NOT_DISCH	0x03	C_{mod} does not discharge in channel0. May be shorted to Vcc
CH1_CMOD_NOT_DISCH	0x04	C_{mod} does not discharge in channel1. May be shorted to Vcc

Side Effects: The interrupts are not disabled during measurement because this function is not intended for precision measurement, only for failure detection. If precision measurement is required, disable the interrupts before calling the `SelfTests_CSD_Modulator ()`.

6.6 uint8 SelfTests_Shield_Shorts (void)

Located in: *SelfTest_Shield.c*

SelfTest_Shield.h

Description: The function checks all the shields and detects shields shorts to Ground, V_{CC} or sensors.

This function is called after scanning is completed.

Parameters: None

ReturnValue: Returns an error code as shown in Table 11.

Table 11. Error Code Descriptions for SelfTests_Shield_Shorts()

Error Code	Value	Description
OK_STATUS	0x00	No errors detected
SHORT_TO_VCC	0x01	A short to Vcc is detected
SHORT_TO_GND	0x02	A short to GND is detected
SHORT_TO_SENSOR	0x04u	A short to sensor is detected
SHORT_TO_SHIELD	0x08u	A short to shield is detected

The number of the shield that caused the error is stored in the global variable CSD_bShieldNum.

Side Effects: None

6.7 uint8 SelfTests_AutoTuning(void)

Located in: *SelfTest_CSD.c*
SelfTest_CSD.h

Description: Check if the IDAC and prescaler are in the predefined range after auto tuning. MIN_IDAC, MAX_IDAC, MIN_PRESCALER, and MAX_PRESCALER define range for test and located in *SelfTest_CSD.h* file. If the parameters are not within the predefined range, detect an error.

Parameters: None

ReturnValue: Returns an error code as shown in Table 12.

Table 12. Error Code Descriptions for SelfTests_AutoTuning ()

Error Code	Value	Description
OK_STATUS	0x00	No errors detected
ERROR_IDAC	0x01	IDAC is out of the predefined range
ERROR_PRESCALER	0x02	Prescaler is out of the predefined range

Side Effects: Call this function once after finishing auto tuning to check that the tuning parameters are correct for the CSD component normal operation.

7 Summary

This application note describes how to implement various diagnostic measures for CapSense CSD proposed by the IEC 60730 standard. Introduction of IEC 60730-1 into the design of white goods and other appliances will add a new level of safety for consumers. By taking advantage of PSoC 3 and PSoC 5LP, design teams can comply with regulations while maintaining or reducing electronic systems cost. PSoC and Class B CapSense CSD Safety Software Library allows for a strong system-level development platform, superior performance, fast time to market, and energy efficiency.

8 References

IEC 60730 Standard, “Automatic Electrical Controls for Household and Similar Use”, IEC 60730-1 Edition 3.2, 2007

Document History

Document Title: AN79973 - PSoC 3 and PSoC 5LP CapSense CSD – IEC 60730 Class B Safety Software Library

Document Number: 001-79973

Revision	ECN	Orig. of Change	Submission Date	Description of Change
**	3631447	TARK_UKR	06/12/2012	New application note.
*A	4837865	GJV	08/03/2015	Updated template
*B	5832257	AESATMP8	07/25/2017	Updated logo and Copyright.
*C	6230598	GJV	07/19/2018	Sunset Review Updated template

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 | 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, 2012-2018. 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.