

BLE Temperature Measurement

1.0

Features

- Health Thermometer Profile in GATT Server role
- Die temperature measurement
- Battery service
- Battery level measurement
- Low Power mode
- LED status indication

General Description

This example demonstrates the Health Thermometer Profile operation of the BLE PSoC Creator component. The device simulates thermometer readings and sends it over the BLE Health Thermometer Service. It also measures a battery level value and sends it over the BLE Battery Service.

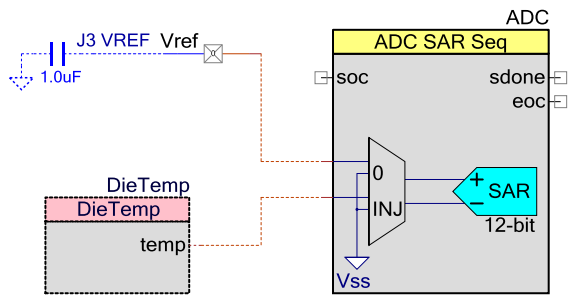
Development Kit Configuration

1. Default CY8CKIT-042 BLE Pioneer Kit configuration.
2. Connect J2 pin P3[0] to J3 pin VREF.

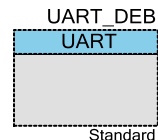
Project Configuration

The example project consists of the following components: ADC_SAR_Seq, DieTemp, BLE, UART, digital output pin, digital input pin, and analog input pin. The ADC_SAR_Seq and DieTemp are used to measure the battery voltage and die temperature. The output pins are used to reflect the line signal output on the LED. The input pin is configured to the resistive pull up mode and is used to wake device from low power hibernate mode. The top design schematic is shown in **Figure 1**.

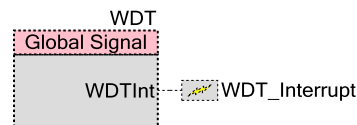
BLE Temperature Measurement Example Project



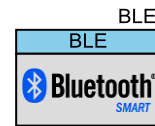
ADC measures battery voltage and die temperature.



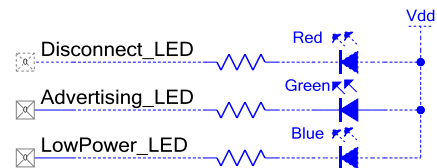
UART is used for transmitting the debug information.



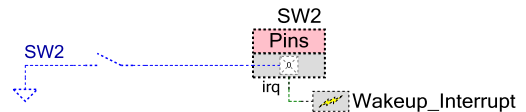
WDT is used as a general timer for LED blinking and Measurement interval.



BLE component configured as Health Thermometer profile with Battery service.



The red LED is used to indicate that the device is disconnected. The green LED is used to indicate that the device is advertising. The blue LED is used to indicate Low battery level (<10%).



The button is used to wake the device up from the hibernate mode.

Figure 1. Top design schematic

The Vref analog input pin is locked to P3[0]. The ADC is configured for a single ended measurement with a sample rate of 3125 SPS.

The BLE component is configured as Health Thermometer Profile in the Peripheral role. Battery Service is used to send a measured battery level.

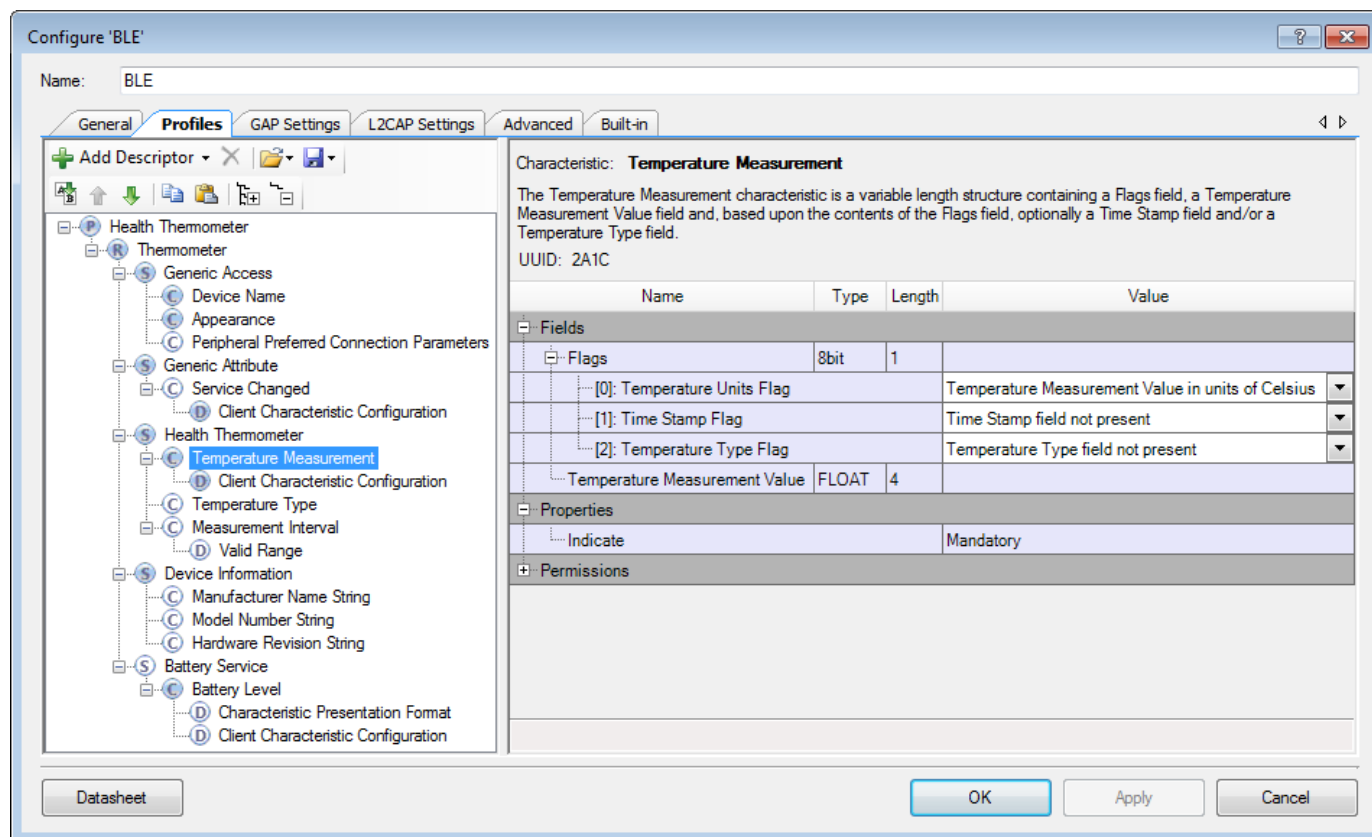


Figure 2. GATT settings

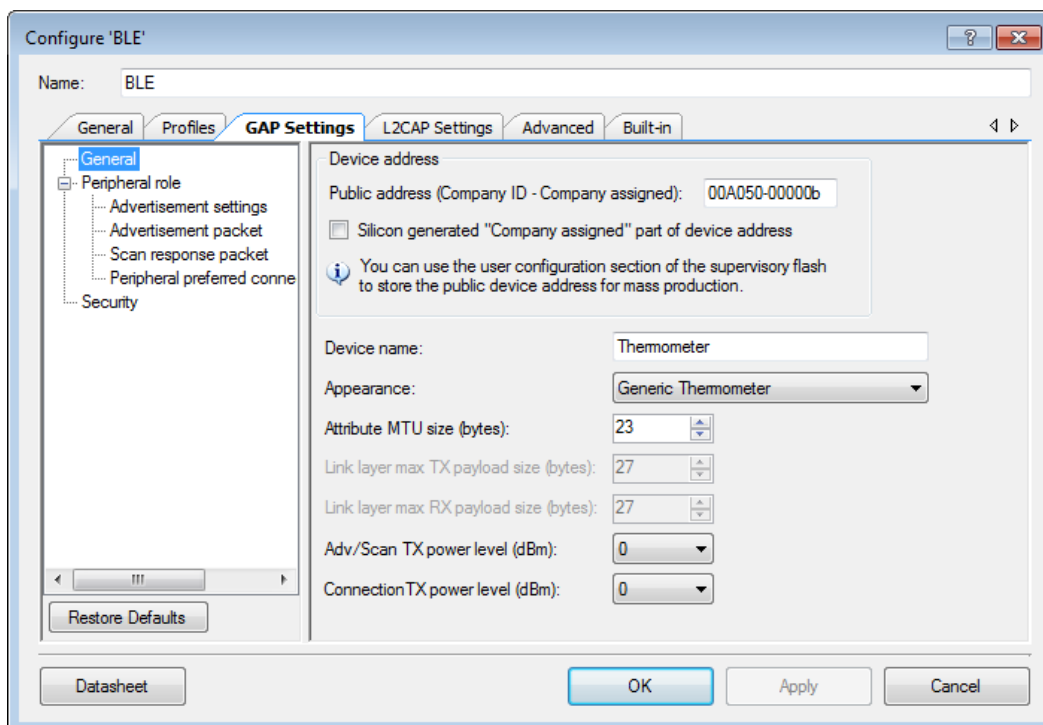


Figure 3. GAP settings

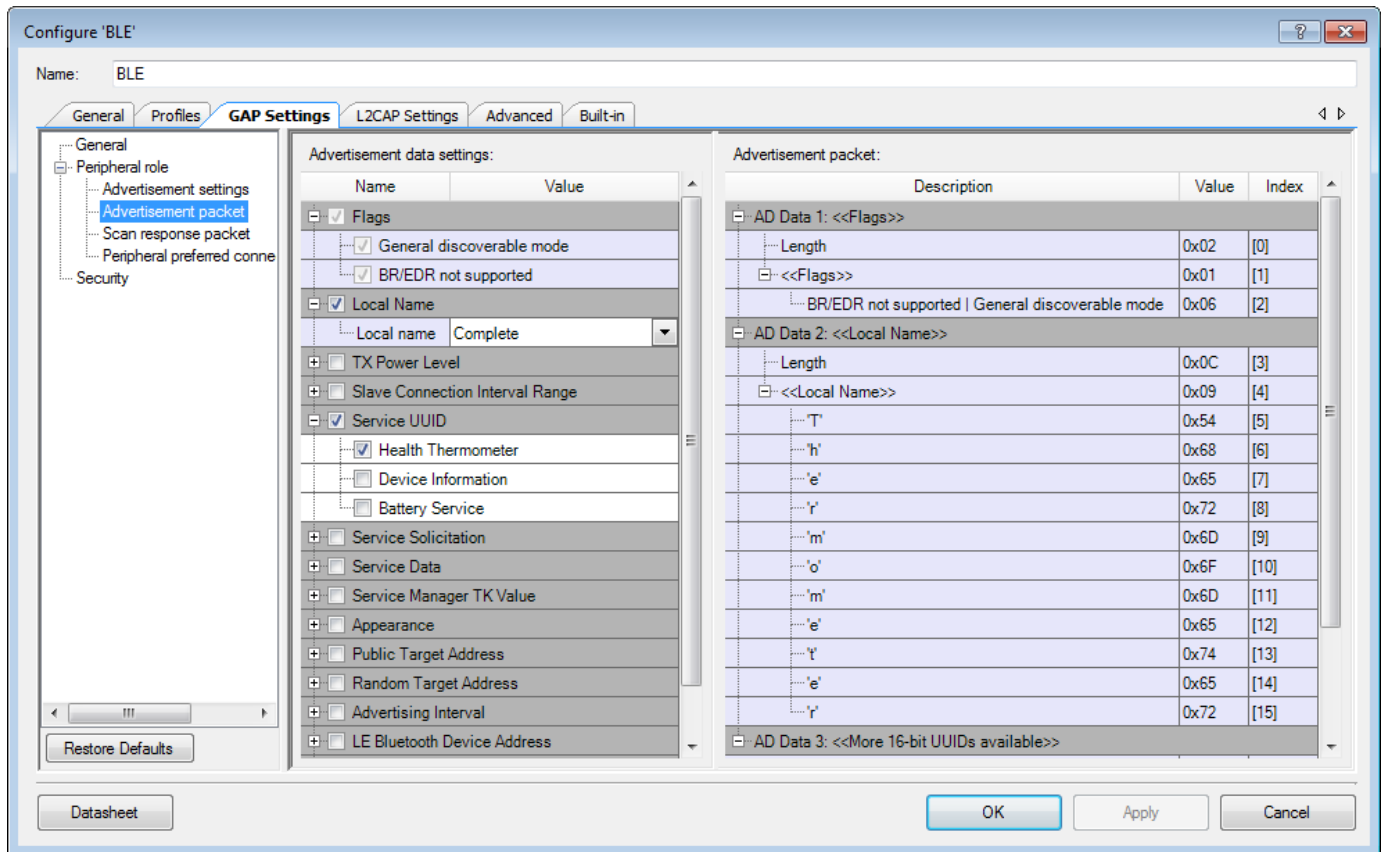


Figure 4. GAP settings -> Advertisement packet

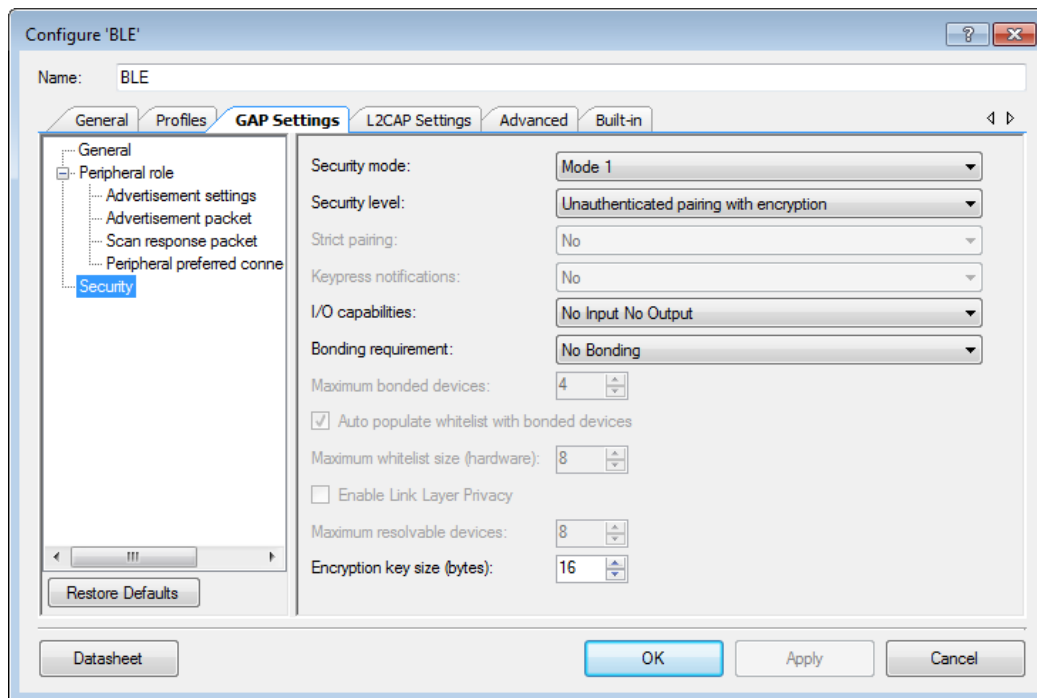


Figure 5. Security settings

Project Description

The project demonstrates the core functionality of BLE component configured as a Health Thermometer GATT Server.

The reference to the ADC in PSoC 4 can be either internal 1.024V or VDD. A simple method to measure VDD in PSoC 4 would be to use a resistive divider and scale the VDD to 1.024V and set the ADC reference to internal 1.024V. This project measures the battery voltage which is equal to the VDD voltage without an external resistive divider.

When VREF is selected as reference to the ADC, and if the reference bypass is enabled, this will bring out VREF, through an internal series resistance, to the external dedicated pin VREF (J16). An external 1uF bypass capacitor is connected to this pin on CY8CKIT-042 BLE Pioneer Kit.

Following is the procedure to measure VDD that exploits this feature.

- Make sure that J2 pin P3[0] is connected to J3 pin VREF.
- Set the reference of ADC to VREF and enable a bypass. Keep the bypass enabled for a short time (25 ms in this test) and the external capacitor is charged to VREF.
- Change the reference of the ADC to VDD and measure the voltage on P3[0].
- Calculate the VDD voltage using the formula:

$$VDD = (1.024 * \text{Full Scale Counts}) / \text{ADC Counts P3[0]}$$

where Full Scale Counts = 2047. With the ADC configured for a single ended measurement, the effective resolution is 11 bits and the full scale count is 2047.

One callback function (AppCallBack()) is required to receive generic events from BLE Stack. CyBle_GappStartAdvertisement() API is called after CYBLE_EVT_STACK_ON event to start advertising with the packet shown in **Figure 4**. HtsCallBack() callback function receives events from the Health Thermometer Service.

The other callback function (BasCallBack()) is required for receiving events from the BAS Services.

On advertisement timeout, the system remains in the sleep mode. Press the mechanical button on CY8CKIT-042 BLE (SW2) to wake up the system and start re-advertising.

The project measures the die temperature and sets it as an initial value at the system startup.

The current temperature is increased and overlapped between 15 and 40 degree. The temperature unit flag is toggled on each temperature update. A fixed temperature type flag value of "Body (general)" has been used.

The initial measurement interval value is set to 10 seconds. If a central device writes a new value to the measurement interval, then a peripheral device updates the timer period and sends a notification on every measurement interval.

To indicate that the device is advertising, the green LED is blinking. The red LED is turned on after disconnection to indicate that no Client is connected to the device. When a Client is

connected successfully, both red and green LEDs are turned off. When the measured battery voltage drops below 10% limit, the blue LED will be on.

Expected Results

After pairing with CySmart mobile app ([Android](#) / [iOS](#)), the BLE device will measure and send the die temperature. You can notice that temperature unit is changing every 10 seconds between °C (Celsius) and °F (Fahrenheit).

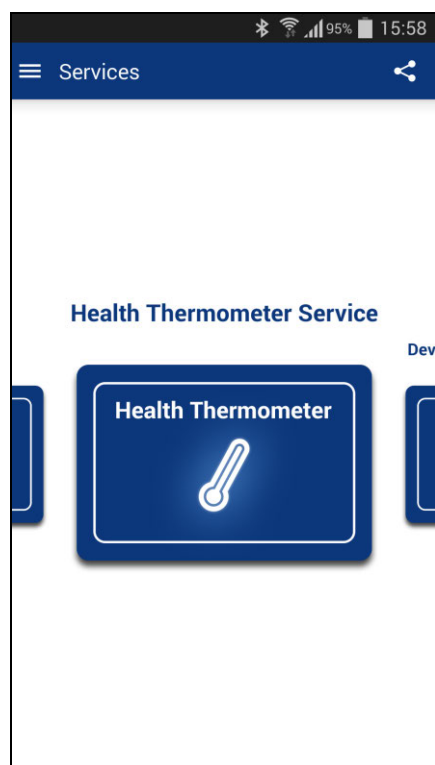


Figure 6. CySmart app on Android

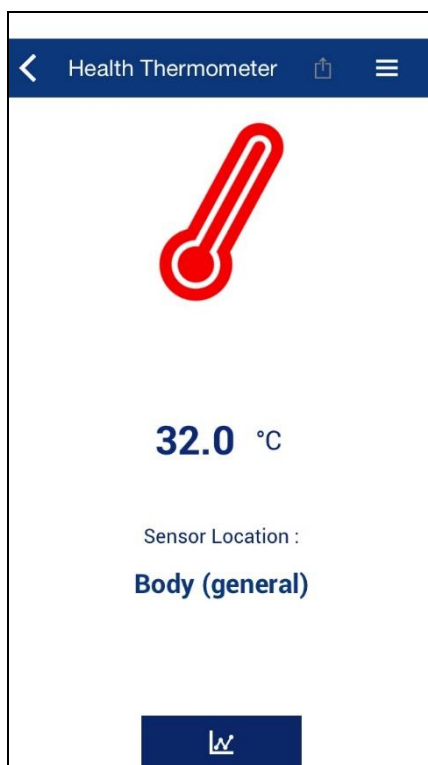


Figure 7. CySmart app on iOS. Measurement unit is Celsius degrees

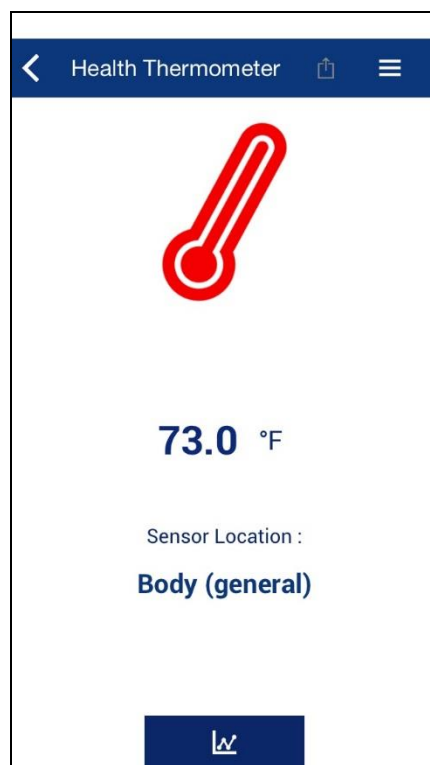


Figure 8. CySmart app on iOS. Measurement unit is Fahrenheit degrees

Also, the Thermometer project can be used together with [CySmart app for Windows](#). It is required to match the security settings between Thermometer and CySmart Client and perform pairing (bonding) before any writing (enabling notifications etc.) into Server's GATT database. For further instructions on how to use CySmart application, see [CySmart User Guide](#).

The simple example how to use CySmart Windows application as Health Thermometer Service client is the next:

- Connect the CySmart BLE dongle to a USB port on the PC.
- Launch CySmart app and select connected dongle in the dialog window.
- Reset the development kit to start advertising by pressing SW1 button.
- Click **Start Scan** button to discover available devices.
- Select **Thermometer** in the list of available devices and connect to it.
- Click **Pair**, then **Discover All Attributes**, and **Enable All Notifications** in CySmart app.
- Observe the Temperature Measurement characteristic indications with measured (first time) and simulated (to show changes) data:

The screenshot shows the CySmart Windows application interface. At the top, there are tabs for 'Select Dongle', 'Configure Master Settings', 'Manage PSMs', and 'Disconnect'. Below these, the 'Master' tab is selected, showing the device name 'Thermometer [00:A0:50:00:00:0B]'. The 'Attributes' section is active, displaying a table of service and characteristic attributes. The table has columns for Handle, UUID, UUID Description, Value, and Properties. The 'Primary Service Declaration: Health Thermometer' is expanded, showing several characteristics. The 'Characteristic Declaration: Temperature Measurement' is highlighted, with its value '00:1A:00:00:00' and properties '0x20' circled in red. Below this, the 'Characteristic Declaration: Temperature Type' and 'Characteristic Declaration: Measurement Interval' are also visible. At the bottom, the 'Log' section shows a list of events. Three events are highlighted with red boxes: 'Attribute Handle: 0x000E Temperature indication Value: [00:18:00:00:00] <- 24 Celsius degrees', 'Attribute Handle: 0x000E 77 Fahrenheit degrees', and 'Attribute Handle: 0x000E 26 Celsius degrees'.

Handle	UUID	UUID Description	Value	Properties
Primary Service Declaration: Health Thermometer				
0x000C	0x2800	Primary Service Declaration	09:18 (Health Thermometer)	
Characteristic Declaration: Temperature Measurement				
0x000D	0x2803	Characteristic Declaration	20:0E:00:1C:2A	
0x000E	0x2A1C	Temperature Measurement	00:1A:00:00:00	0x20
0x000F	0x2902	Client Characteristic Configuration	02:00 <- indication is enabled	
Characteristic Declaration: Temperature Type				
0x0010	0x2803	Characteristic Declaration	02:11:00:1D:2A	
0x0011	0x2A1D	Temperature Type		0x02
Characteristic Declaration: Measurement Interval				
0x0012	0x2803	Characteristic Declaration	0A:13:00:21:2A	
0x0013	0x2A21	Measurement Interval		0x0A
0x0014	0x2906	Valid Range		
Primary Service Declaration: Device Information				

Log

- [16:20:32:533] : Attribute Handle: 0x000E Temperature indication Value: [00:18:00:00:00] <- 24 Celsius degrees
- [16:20:42:533] : Characteristic Value Indication event received
- [16:20:42:533] : Attribute Handle: 0x000E 77 Fahrenheit degrees
- [16:20:52:535] : Characteristic Value Indication event received
- [16:20:52:535] : Attribute Handle: 0x000E 26 Celsius degrees

The details about the Health Thermometer Service characteristic data structures are in the [HTS Specification](#).

© Cypress Semiconductor Corporation, 2009-2015. The information contained herein is subject to change without notice. Cypress Semiconductor Corporation assumes no responsibility for the use of any circuitry other than circuitry embodied in a Cypress product. Nor does it convey or imply any license under patent or other rights. Cypress products are not warranted nor intended to be used for medical, life support, life saving, critical control or safety applications, unless pursuant to an express written agreement with Cypress. Furthermore, Cypress does not authorize its products for use as critical components in life-support systems where a malfunction or failure may reasonably be expected to result in significant injury to the user. The inclusion of Cypress products in life-support systems application implies that the manufacturer assumes all risk of such use and in doing so indemnifies Cypress against all charges.

PSoC® is a registered trademark, and PSoC Creator™ and Programmable System-on-Chip™ are trademarks of Cypress Semiconductor Corp. All other trademarks or registered trademarks referenced herein are property of the respective corporations.

Any Source Code (software and/or firmware) is owned by Cypress Semiconductor Corporation (Cypress) and is protected by and subject to worldwide patent protection (United States and foreign), United States copyright laws and international treaty provisions. Cypress hereby grants to licensee a personal, non-exclusive, non-transferable license to copy, use, modify, create derivative works of, and compile the Cypress Source Code and derivative works for the sole purpose of creating custom software and or firmware in support of licensee product to be used only in conjunction with a Cypress integrated circuit as specified in the applicable agreement. Any reproduction, modification, translation, compilation, or representation of this Source Code except as specified above is prohibited without the express written permission of Cypress.

Disclaimer: CYPRESS MAKES NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS MATERIAL, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. Cypress reserves the right to make changes without further notice to the materials described herein. Cypress does not assume any liability arising out of the application or use of any product or circuit described herein. Cypress does not authorize its products for use as critical components in life-support systems where a malfunction or failure may reasonably be expected to result in significant injury to the user. The inclusion of Cypress' product in a life-support systems application implies that the manufacturer assumes all risk of such use and in doing so indemnifies Cypress against all charges.

Use may be limited by and subject to the applicable Cypress software license agreement.

