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.
Objective
This example demonstrates liquid level sensing using capacitive sensors.

Overview
This code example demonstrates how to use PSoC® 4, CapSense® technology, and capacitive sensors to measure the depth or presence of water-based liquids in non-conductive containers. Sensors located on or near the container’s exterior provide real-time reporting of liquid level. Many options exist to use low-cost materials to construct and integrate the sensors while still providing high-precision measurements.

The firmware continuously measures the data from sensors and outputs the calculated liquid level over both a serial UART connection and through a Micrium® µC/Probe™ GUI using the Arm® Serial Wire Debug (SWD) interface. PSoC 4 Pioneer development kits provide both UART and SWD interfaces to the user’s computer through a USB connection.

The two projects LLS_CSD_2RX-042 and LLS_CSD_12RX-042 demonstrate liquid-level sensing on 2-sensor (2RX) and 12-sensor (12RX) patterns using the CY8CKIT-022 CapSense Liquid Level Sensing Shield and CY8CKIT-042 PSoC 4 Pioneer Kit. Both of these projects are contained in this code example’s workspace. These projects support water-based liquids using CapSense self capacitance. An update scheduled for 2016 will support additional liquids, higher-performance CapSense mutual capacitance scanning, and increased noise immunity.

PSoC Resources
Cypress provides a wealth of data at www.cypress.com to help you to select the right PSoC device for your design, and quickly and effectively integrate the device into your design. For a comprehensive list of resources, see KBA86521, How to Design with PSoC 3, PSoC 4, and PSoC 5LP. The following is an abbreviated list for PSoC 4:

- **Overview**: PSoC Portfolio, PSoC Roadmap
- **Product Selectors**: PSoC 1, PSoC 3, PSoC 4, or PSoC 5LP. In addition, PSoC Creator includes a device selection tool.
- **Datasheets** describe and provide electrical specifications for the PSoC 4 device family
- **CapSense Design Guide**: Learn how to design capacitive touch-sensing applications with the PSoC 4 family of devices.
- **Application Notes and Code Examples** cover a broad range of topics, from basic to advanced level. Many of the application notes include code examples.
- **Technical Reference Manuals (TRM)** provide detailed descriptions of the architecture and registers in each PSoC 4 device family.
- **Development Kits**:
  - CY8CKIT-040, CY8CKIT-042, CY8CKIT-042-BLE, CY8CKIT-046 and CY8CKIT-044 PSoC 4 Kits, are easy-to-use and inexpensive development platforms. These kits include connectors for Arduino™ compatible shields and Digilent® Pmod™ daughter cards.
  - CY8CKIT-049 is a very low-cost prototyping platform for sampling PSoC 4 devices.
  - CY8CKIT-001 is a common development platform for all PSoC family devices.
  - The MiniProg3 device provides an interface for flash programming and debug.
PSoc Creator

**PSoc Creator** is a free Windows-based Integrated Design Environment (IDE). It enables concurrent hardware and firmware design of systems based on PSoc 3, PSoc 4, and PSoc 5LP. See Figure 1 – with PSoc Creator, you can:

1. Drag and drop Components to build your hardware system design in the main design workspace
2. Co-design your application firmware with the PSoc hardware
3. Configure Components using configuration tools
4. Explore the library of 100+ Components
5. Review Component datasheets

**Figure 1. PSoc Creator Features**

---

**Requirements**

**Tool:** PSoc Creator 3.3 CP1 or later

**Programming Language:** C (GCC 4.9)

**Associated Parts:** PSoc 4 devices with CapSense

**Related Hardware:** CY8CKIT-022, CY8CKIT-042

**Design**

Figure 2 shows the PSoc Creator schematic for the **LLS_CSD_2RX-042** code example. For additional information on liquid-level sensing theory, algorithms, sensor layout, tuning, calibration, and system design guidance, see Application Note **AN202478** – **PSoc® 4 - Capacitive Liquid Level Sensing**. The projects feature the following Components:

- **CapSense CSD Component** for measuring the signal level on each of the liquid sensors. Liquid level is calculated after each scan is completed.
  - Self-capacitance scan method
  - 2 or 12 generic sensors, depending on the sensor used for custom liquid-level processing of raw values
- **UART Component** for data viewing and command entry.
  - Standard 115200 baud rate, no parity, 8 data bits, 1 stop bits (N81) terminal interface
  - Command input and data output
Design Considerations

The CapSense Component in the projects is tuned to work with the CY8CKIT-022 plus CY8CKIT-042 kit combination and the liquid container provided. The CY8CKIT-022 sensors can be attached to a user-supplied container, or other compatible hardware may be used, although re-tuning of the CapSense Component may be required. The CapSense_CSD Component can also add or remove sensors from the 12RX project to allow operation with custom sensors with a different number of CapSense sensor elements.

Project tuning and algorithms are optimized for the permittivity, surface tension, and conductivity of water. The default tuning will work with most human-consumable, water-based liquids. Liquids with properties that differ from water may require tuning, algorithmic, or sensor geometry changes outside the scope of this code example.

These projects use a larger number of global variables than normally used to enable interaction with the µC/Probe tool. µC/Probe can only interface with global variables.

Sensor Selection

- 12RX Sensor

The 12 sensor flexible PCB provides the highest accuracy across all operating conditions and is the best choice for most designs. The 12RX sensor is segmented allowing each sensor element to accurately measure the liquid level in its limited range.

Each sensor generates a binary output when it is approximately half covered with liquid. The reported liquid level is the sum of covered sensors multiplied by the height of each sensor. The top and bottom sensors are each half the height of other sensors, thereby increasing the accuracy at the empty and full limits and reducing the sensor height by one. Maximum liquid level error is equal to 100% divided by the number or sensors – 1. The maximum error for the 12RX sensor is 9%. Typical error is 4.5%. The reported liquid level of the 12RX sensor is stair-step compared to the actual liquid level as seen in Figure 3.
The 2-sensor flexible PCB provides reduced accuracy compared to the 12RX sensor but requires only two sensors for reduced sensor cost. The 2RX sensor is composed of two sensors shaped so that the ratio of their values is equal to the percent value of the liquid level.

Any offset on the sensor values generates a liquid-level error that is greatest at low liquid levels. The typical error for the 2RX sensor is 15% and decreases as the container fills. Maximum error is determined by the system’s response to temperature changes that impact the sensors response. The reported liquid level of the 2RX sensor is linear as seen in Figure 4.

Hardware Setup

This code example is designed to run on a CY8CKIT-022 shield board mounted on a CY8CKIT-042 Pioneer kit. For the detailed kit board setup, see the corresponding CY8CKIT-022 kit user guide. Hardware setup consists of the following basic steps:

1. Mount the CY8CKIT-022 shield board on a CY8CKIT-042 Pioneer board.
2. Attach 2RX or 12RX flexible sensor to the container supplied, using the pre-attached adhesive. The sensor chosen must match the firmware programmed into the PSoC device.

3. Connect the flexible sensor to the shield board. Ensure there are no air pockets between the sensor and container as they can cause erratic sensor results if the distance between the sensor and liquid changes.

4. Connect the CY8CKIT-042 Pioneer board to a computer using a USB cable.

Software Setup

This code example firmware supports both serial terminal program and Micrium’s µC/Probe computer interfaces. The µC/Probe interface is preferred because it has a GUI and it has access to most input and output parameters using the ARM SWD programming and debug interface. An alternative UART interface outputs the current liquid level height and allows empty level calibration through a terminal emulator.

Micrium µC/Probe

Micrium µC/Probe is a software development tool that incorporates Micrium’s proprietary Graphical Live Watch to graphically visualize the internals of any embedded system. With µC/Probe, you can evaluate your embedded design effortlessly, with just a few mouse clicks. You can download the µC/Probe Users’ Guide and µC/Probe Target Manual from the following web page: µC/Probe User Guide and µC/Probe Target Manual. To learn more about the µC/Probe, visit: micrium.com/tools/ucprobe/overview/.

Micrium µC/Probe allows you to monitor the values of all the global variables in your project while the project is running. These values can be mapped onto multiple graphical interface elements provided by the µC/Probe software such as angular gauges or numeric indicators. This allows you to use the µC/Probe as an effective debug tool because it does not halt the CPU. Cypress provides pre-designed µC/Probe workspace files for all the projects associated with this code example. These workspace files can be found in the following folders where the code example files were downloaded:

<Download_Directory>\CE202419\uProbe Files\LLS_CSD_2RX.wpx
<Download_Directory>\CE202419\uProbe Files\LLS_CSD_12RX.wpx

Installing and using µC/Probe

1. Download and install the Professional Edition of the µC/Probe software available at micrium.com/download/ucprobe-3-0-trial-version/. The installer includes both the trial and professional versions. Either version can be used with the included workspace files.

   **Note:** You have to register an account with Micrium to download the software.

2. Open Micrium µC/Probe from **Start > Micrium > uC-Probe > Micrium uC-Probe** shown in **Figure 5**.

   **Note:** Refer to the µC/Probe User Manual for detailed information on µC/Probe. You can access it from **Start > Micrium > uC-Probe > User Manual**.

   ![Figure 5. Micrium uC-Probe in Start Menu](image)

3. Optionally, enter the professional license - The CY8CKIT-022 Liquid Level Sensing kit gives you a 1-month free trial license for the µC/Probe Professional Edition. Refer to the µC/Probe License Card for the µC/Probe license key. You can also purchase a license for the Professional Edition of µC/Probe from Micrium to access all the features in this software. To activate the license, click on **File > Activation**. In the License Manager window, type in the license key and press **Activate**.
4. Connect to the SWD interface device by selecting **File > Settings**. Select the **Communication** tab from the Settings window as shown in Figure 6. Click the **Cypress PSoC Prog** option and select the **KitProg/<serial number>** from the drop-down menu. Select **OK** to close the dialog box.

![Figure 6. KitProg Connection in µC/Probe](image)

5. Select **File > Open** and select the appropriate µC/Probe workspace file for the code example programmed into the PSoC device.

6. A pop-up window shown in Figure 7 may appear asking you to update the location of the .elf file. The .elf file is located in the project workspace directory under the folder:

   `<Download_Directory>\CE202419\LLS_CSD_2RX-042.cydsn\CortexM0\ARM_GCC_493\Debug\LLS_CSD_2RX-042.elf`

   `<Download_Directory>\CE202419\LLS_CSD_12RX-042.cydsn\CortexM0\ARM_GCC_493\Debug\LLS_CSD_12RX-042.elf`
7. Select **Run** from the toolbar shown in Figure 8 to view the status of the global variable you want to monitor.

8. The µC/Probe interactive GUI will be displayed in the workspace screen tab as shown in Figure 9.
Serial Terminal

The documentation for setting up a terminal emulator for this example uses Tera Term but any terminal emulator software may be used that is configurable to the standard UART settings shown in Figure 11. Tera Term is open source and downloadable directly from the author’s website at https://en.osdn.jp/projects/ttssh2/. All serial commands supported by the firmware are lower case and case sensitive.

Supported serial commands are;

- **stop** – Stops outputting data over the serial connection
- **cal** – Stores empty container sensor values to EEPROM for calibration of future readings
- **basic** – Continuously outputs liquid level in millimeters (mm) and percent (%).
- **csv** – Continuously outputs intermediate computation values as well as liquid level in CSV format. The CSV format supports easy terminal emulator logging and data analysis with a spreadsheet or other tools.
- **[Enter]** – Outputs the next set of level values from the sample array
- **Reset** – Resets the sample array pointer to zero

1. Create new connection.

   Launch Tera Term and select **File > New connection.** Select **Serial** as the connection type and choose the **KitProg USB-UART** communication port (COM) as shown in Figure 10. The actual COM port number will vary between computers and USB ports. If multiple communication ports are listed, it can be helpful to disconnect and reconnect the development kit USB cable and look for the COM port that is disappears and reappears.

   **Figure 10. New Connection Creation**

   ![New Connection Creation](image)

2. Setup serial port parameters.

   Open the Tera Term Serial port setup dialog at **Setup > Serial port...** Only the **Baud rate** should require changing to 115200 but it is good to also confirm the other settings as shown in Figure 11.

   **Figure 11. Terminal Emulator Setup Parameters**

   ![Terminal Emulator Setup Parameters](image)
3. Serial output is displayed in the terminal window as shown in Figure 12. On startup, the PSoC device will output information on the project name, version, supported commands, empty sensor calibration values, and continuous output of the current liquid level in millimeters and percent.

Figure 12. Startup Terminal Window Output

Components

Table 1 lists the PSoC Creator Components used in this example, as well as the hardware resources used by each.

Table 1. List of PSoC Creator Components

<table>
<thead>
<tr>
<th>Component</th>
<th>Name</th>
<th>Hardware Resources</th>
<th>Non-default Parameter Settings</th>
</tr>
</thead>
</table>
| CapSense CSD [v2.30] | CapSense_CSD | 1 CapSense block 2 or 12 GPIO pins for sensors 1 GPIO pin for Cmod capacitor | General  
  Tuning method: Manual with run time tuning  
  Widgets  
  Generics: 2 or 12  
  Scan resolution: 14 bits  
  Scan Order  
  Modulation IDAC: Not critical as this value is set by firmware  
  Compensation IDAC: 0  
  Advanced  
  IDAC range: 8x  
  Analog switch: PRS-12b  
  Individual freq settings: Disabled  
  Sense clock divider: 10  
  Modulator clock divider: 10 |
| UART (SCB mode) [v3.0] | UART | 1 SCB block | RX buffer size: 32  
  TX buffer size: 32 |
| Emulated EEPROM [v1.10] | Em_EEPROM | 1 Flash block | None |
| Digital Output Pin [v2.10] | Pin.TxGnd | 1 GPIO pin for 2RX sensor only | Drive mode: Strong  
  Initial drive state: 0 |
Design-Wide Resources

Figure 13 shows the pin selections for the 2RX project and Figure 14 shows the pin selections for the 12RX project.

Figure 13. 2RX project Pins Tab in Design Wide Resources (.cydwr file)

<table>
<thead>
<tr>
<th>Name</th>
<th>Port</th>
<th>Pin</th>
<th>Lock</th>
</tr>
</thead>
<tbody>
<tr>
<td>\CapSense_CSD:Cmod\ (Cmod)</td>
<td>P4[2]</td>
<td>22</td>
<td></td>
</tr>
<tr>
<td>\CapSense_CSD:Sns[0]\ (Generic0_0_GEN)</td>
<td>P0[0]</td>
<td>24</td>
<td></td>
</tr>
<tr>
<td>\CapSense_CSD:Sns[1]\ (Generic1_0_GEN)</td>
<td>P3[7]</td>
<td>18</td>
<td></td>
</tr>
<tr>
<td>UART:rx\</td>
<td>P0[4]</td>
<td>28</td>
<td></td>
</tr>
<tr>
<td>UART:tx\</td>
<td>P0[5]</td>
<td>29</td>
<td></td>
</tr>
<tr>
<td>Pin_TxGnd</td>
<td>P2[5]</td>
<td>7</td>
<td></td>
</tr>
</tbody>
</table>

Figure 14. 12RX project Pins Tab in Design Wide Resources (.cydwr file)

<table>
<thead>
<tr>
<th>Name</th>
<th>Port</th>
<th>Pin</th>
<th>Lock</th>
</tr>
</thead>
<tbody>
<tr>
<td>\CapSense_CSD:Cmod\ (Cmod)</td>
<td>P4[2]</td>
<td>22</td>
<td></td>
</tr>
<tr>
<td>\CapSense_CSD:Sns[0]\ (Generic0_0_GEN)</td>
<td>P2[4]</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>\CapSense_CSD:Sns[1]\ (Generic1_0_GEN)</td>
<td>P2[3]</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>\CapSense_CSD:Sns[2]\ (Generic2_0_GEN)</td>
<td>P2[2]</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>\CapSense_CSD:Sns[3]\ (Generic3_0_GEN)</td>
<td>P1[3]</td>
<td>40</td>
<td></td>
</tr>
<tr>
<td>\CapSense_CSD:Sns[4]\ (Generic4_0 Gen)</td>
<td>P1[5]</td>
<td>42</td>
<td></td>
</tr>
<tr>
<td>\CapSense_CSD:Sns[5]\ (Generic5_0_GEN)</td>
<td>P3[6]</td>
<td>17</td>
<td></td>
</tr>
<tr>
<td>\CapSense_CSD:Sns[6]\ (Generic6_0_GEN)</td>
<td>P2[6]</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>\CapSense_CSD:Sns[7]\ (Generic7_0_GEN)</td>
<td>P2[7]</td>
<td>9</td>
<td></td>
</tr>
<tr>
<td>\CapSense_CSD:Sns[8]\ (Generic8_0_GEN)</td>
<td>P1[0]</td>
<td>27</td>
<td></td>
</tr>
<tr>
<td>\CapSense_CSD:Sns[9]\ (Generic9_0_GEN)</td>
<td>P3[5]</td>
<td>16</td>
<td></td>
</tr>
<tr>
<td>\CapSense_CSD:Sns[10]\ (Generic10_0_GEN)</td>
<td>P0[0]</td>
<td>24</td>
<td></td>
</tr>
<tr>
<td>\CapSense_CSD:Sns[11]\ (Generic11_0_GEN)</td>
<td>P3[7]</td>
<td>18</td>
<td></td>
</tr>
<tr>
<td>UART:rx\</td>
<td>P0[4]</td>
<td>28</td>
<td></td>
</tr>
<tr>
<td>UART:tx\</td>
<td>P0[5]</td>
<td>29</td>
<td></td>
</tr>
</tbody>
</table>

Basic Operation

1. Load the workspace into PSoC Creator by opening
   <Download Directory>\CE202479\CE202479.cywrk.

2. Select the project you wish to program into the PSoC device and set it as the active project by right-clicking on the project branch and selecting ‘Set As Active Project’ as shown in Figure 15. The active project name will be displayed in bold.
3. Build the example project by navigating to **Build > Build <Project Name>**.

4. The default programming interface for the kit is a USB-based, onboard programming interface. To program the device, plug the USB cable into the USB connector of the CY8CKIT-042 Pioneer kit. The kit will enumerate as a composite device. For more details about programming, refer to the Programming and Debugging section of your Pioneer kit’s user guide.

5. Program the example to the device by choosing **Debug > Program**.

6. Power the device, if not already powered.

7. Calibrate sensors for an empty container on first use to compensate for system differences and manufacturing tolerances that affect parasitic capacitance. Calibration stores the empty sensor value for each sensor in flash memory and automatically reloads them on future device startups. *(Please see errata section to correct calibration reload error)* The empty calibration values are subtracted from all future sensor values to provide a consistent reference value for use in calculations. Any changes to CapSense tuning or the sensor connection will require updated calibration values. Execute the calibrate command with just enough liquid in the container to reach the 0mm (0%) mark on the sensor.

   **Note:** Reprogramming the PSoC device will overwrite previously stored calibration values.

   a. **Terminal** – Type ‘cal’ into the terminal. Calibration values for each sensor will be displayed and output will return to the previously selected mode as shown in **Figure 16**.

   **Figure 16. Terminal Calibration**

   ![Terminal Calibration](image)

   b. **µC/Probe** – Press the **Calibrate** button as shown in **Figure 17**.

   ![Calibrate Button](image)
8. Fill and empty the container as desired with water, and view the calculated liquid level.
   a. **Terminal** – The default output displays the current liquid level in percentage values and millimeters to one decimal place approximately once every second. See Figure 12 for an example. Entering the command **basic** at any time returns to the default output.
   b. **µC/Probe** – The liquid level is displayed numerically as **Level Percent (%)** and **Level Millimeter (mm)**, as well as a cylinder bar graph. See Figure 17 for an example.

Water that is significantly colder than the ambient air temperature may cause condensation to form on the sensor surface depending on the humidity level. Condensation may cause the liquid level calculation to return a liquid level with increased error. Condensation during low-temperature evaluation can be reduced by insulating the exposed surface of the sensor.

Hot water above 170°F / 80°C can cause PET plastic to deform. Extremely hot water should be avoided as the liquid container provided in the CY8CKIT-022 kit is made of PET plastic.

Variations in the hardware setup can change the sensor capacitance and increase error in the reported level. Common variations include touching the sensor during operation and moving the liquid container or Pioneer board after calibration.

### Advanced Operation

A set of commands is provided to assist accuracy measurement and analysis of the liquid level system. An array of heights matching the millimeter ruler printed on the CY8CKIT-022 sensors is provided in firmware. The predefined levels are -5, 0, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 110, 120, 130, 140, 150, 153, and 160 mm. While filling the liquid container, the **Output Value** command can be executed when the predefined level is reached. On execution of the command, the predefined level value, sensor difference values, and the currently calculated level are output. The predefined level is automatically incremented in preparation for the next **Output Value** command. This process simplifies taking a complete set of accuracy measurements for analysis because it documents both the expected and actual level together. A second command, **Reset Array**, allows resetting the predefined level to its initial value to start a new test cycle. A full test cycle is shown in Figure 18.

---

**Figure 17. uC/Probe Calibration**

---

**Figure 18. Accuracy Measurement Example**

---
Terminal – Pressing the [Enter] key with no other characters executes the Output Value command. Repeated command execution results in an ordered table of liquid levels suitable for data logging. Entering the command reset executes the Reset Array command.

µC/Probe – Pressing the Output Value and Reset Array buttons shown in Figure 19 executed their respective commands.

Figure 19. µC/Probe Value Array Controls

- Executing the terminal stop command causes the continuous output to halt. This is useful to ensure only desired data is logged. Execution of any other command will resume the output.
- Executing the terminal csv command causes internal sensor values as well as the liquid level to be output in the comma-separated value format for easy import into spreadsheet programs. The first line output after the csv command is entered is the header with a label describing what each value is.
- µC/Probe provides Liquid Level Calibration Controls shown in Figure 20 to directly adjust calibration variables used in liquid-level calculation. µC/Probe provides CapSense Tuning Controls shown in Figure 20 to directly adjust CapSense hardware tuning. For more details on their use, see the Liquid Level Sensing Application Note AN202478.

Figure 20. µC/Probe CapSense Tuning Controls

Errata Information

A bug in the firmware has been discovered that causes the calibration values to return ‘0’ after a reset event. Please make the following edits in project LLS_CSD_12RX-042. No changes are required for project LLS_CSD_2RX-042.

1. In main.c, line 79, replace the code with:
   ```c
   sensorEmptyOffset[i] = ((volatile int16 *)eepromEmptyOffset)[i];
   ```

2. In main.c, line 56, replace the code with:
   ```c
   const int16 CYCODE eepromEmptyOffset[NUMSENSORS] = {0u};
   ```

3. In interface.c, line 92, replace the code with:
   ```c
   status = Em_EEPROM_Write((uint8 *)sensorEmptyOffset, (uint8 *)eepromEmptyOffset, sizeof *sensorEmptyOffset * NUMSENSORS);
   ```
### Related Documents

Table 2 lists all relevant application notes, code examples, knowledge base articles, device datasheets, and Component datasheets.

#### Table 2. Related Documents

<table>
<thead>
<tr>
<th>Application Notes</th>
<th>AN202478 – PSoC® 4 - Capacitive Liquid Level Sensing</th>
<th>AN85951 – PSoC 4® CapSense Design Guide</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Provides design, sensor layout, and tuning guidance for liquid level sensing with PSoC 4 devices</td>
<td>Shows how to design capacitive touch sensing applications with the PSoC 4 and PRoC BLE families of devices and use the CapSense component.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Code Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>CE95285</td>
</tr>
<tr>
<td>CE95286</td>
</tr>
<tr>
<td>CE95288</td>
</tr>
<tr>
<td>CE95366</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>PSoC Creator Component Datasheets</th>
</tr>
</thead>
<tbody>
<tr>
<td>CapSense CSD</td>
</tr>
<tr>
<td>UART (SCB mode)</td>
</tr>
<tr>
<td>Digital Output Pin</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Device Documentation</th>
</tr>
</thead>
<tbody>
<tr>
<td>PSoC 4 Datasheets</td>
</tr>
<tr>
<td>PSoC 4 Technical Reference Manuals</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Development Kits</th>
</tr>
</thead>
<tbody>
<tr>
<td>PSoC 4 Kits</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Software</th>
</tr>
</thead>
<tbody>
<tr>
<td>PSoC Creator Training</td>
</tr>
<tr>
<td>PSoC 3/4/5 Code Examples</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Video Library</th>
</tr>
</thead>
</table>
## Document History

Document Title: CE202479 - PSoC® 4 Capacitive Liquid Level Sensing  
Document Number: 002-02479

<table>
<thead>
<tr>
<th>Revision</th>
<th>ECN</th>
<th>Orig. of Change</th>
<th>Submission Date</th>
<th>Description of Change</th>
</tr>
</thead>
<tbody>
<tr>
<td>**</td>
<td>5032590</td>
<td>GJV</td>
<td>12/02/2015</td>
<td>New code example</td>
</tr>
<tr>
<td>*A</td>
<td>5730193</td>
<td>AESATP12</td>
<td>05/16/2017</td>
<td>Updated logo and copyright.</td>
</tr>
<tr>
<td>*B</td>
<td>6484923</td>
<td>GJV</td>
<td>02/15/2019</td>
<td>Added errata to correct firmware error causing calibration data to return '0' after reset.</td>
</tr>
</tbody>
</table>
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/mcuc |
| 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 Corporation, 2015-2019. 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, Spanion, the Spanion 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.

www.cypress.com