AN92239

Proximity Sensing with CapSense®

Author: Chethan

Associated Project: Yes

Associated Part Family: PSoC® 4 Series

Software Version: PSoC Creator™ 3.2 SP1 or later

Related Application Notes: For a complete list, click here.

To get the latest version of this application note, or the associated project file, please visit http://www.cypress.com/AN92239.

AN92239 shows how to implement capacitive proximity-sensing applications using PSoC® CapSense®. It explains how to design a proximity sensor and tune it to achieve a large proximity-sensing distance and liquid-tolerant proximity sensing. This application note also explains how to implement gesture detection based on proximity sensing and the wake-on-approach method to reduce device power consumption. Example projects demonstrate tuning proximity sensors for a large proximity-sensing distance and liquid tolerance, and implementing gesture detection for PSoC 4 series devices.

Contents

1 Introduction

A proximity sensor detects the presence of a nearby object without any physical contact. There are various types of proximity sensors such as capacitive, inductive, magnetic, Hall effect, optical, and ultrasonic sensors, each of which has its own advantages and disadvantages. Capacitive proximity sensing has gained huge popularity because of its low cost, high reliability, low power, sleek aesthetics, and seamless integration with existing user interfaces.

Cypress's CapSense devices support self-capacitance based proximity sensing and can achieve up to a 30-cm proximity-sensing distance. The process of setting CapSense Sigma Delta (CSD) parameters to achieve optimum performance is called "tuning." See Tuning CapSense CSD Parameters for a Large Proximity-Sensing Distance.

This application note explains how to:

- Design a proximity sensor and tune the CSD parameters to achieve large proximity-sensing distances
- Tune the CSD parameters for a proximity sensor to achieve liquid tolerance
- Implement proximity-based gesture detection
- Implement the wake-on-approach feature to reduce the power consumption of the device

Example Projects demonstrate tuning for the maximum proximity-sensing distance, tuning for liquid tolerance, and implementing proximity-based gesture detection. These example projects are designed to operate on the CY8CKIT-024 CapSense Proximity Shield with the CY8CKIT-040 PSoC 4000 Pioneer Kit or the CY8CKIT-042 PSoC 4 Pioneer Kit.

This application note is for advanced CapSense users. It assumes that you are familiar with Cypress's CapSense technology. If you are new to CapSense technology, refer to the Getting Started with CapSense design guide to learn the basics of CapSense technology. To learn the basics of proximity sensing based on CapSense, refer to the "Proximity Sensing" section in the Getting Started with CapSense design guide. To learn about the implementation of CapSense technology in a specific device family, refer to the device-specific CapSense design guides available at CapSense Design Guides.

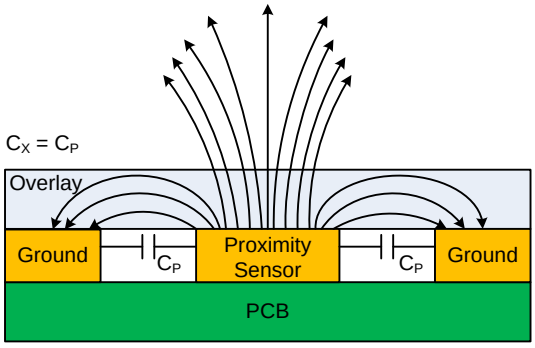
The initial sections of this application note cover the basics of tuning a proximity sensor and implementing gesture detection. If you want to evaluate the example projects, proceed to the Example Projects section.

2 Proximity Sensing with CapSense

The proximity-sensing technique based on CapSense involves measuring the change in the capacitance of a proximity sensor when a target object approaches the sensor. The target object can be a human finger, hand, or any conductive object. Proximity sensors based on CapSense can be constructed using a conductive (usually copper or indium tin oxide) pad or trace laid on a nonconductive material like PCB or glass. The intrinsic capacitance of the PCB trace or other connections to a capacitive sensor results in a sensor parasitic capacitance (C_P).

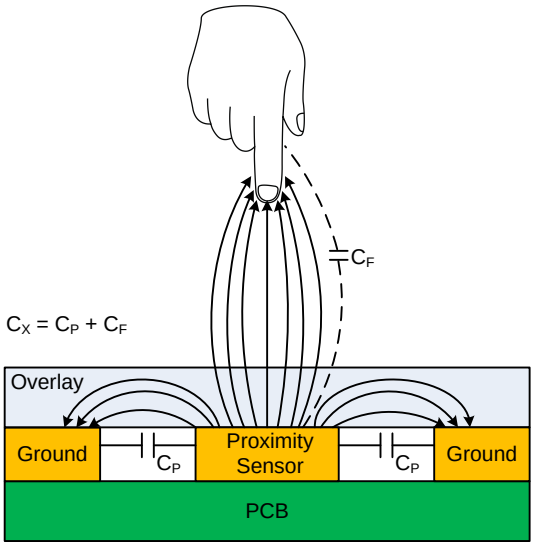
When a proximity sensor based on CapSense is excited by a voltage source, an electric field is created around the sensor. A small number of electric field lines couple to the nearby ground, while most of the electric field lines are projected into the nearby space, as shown in Figure 1.

Figure 1. Proximity Sensing



When a target object approaches the sensor, some of the electric field lines couples to the target object and add a small amount of finger capacitance (C_F) to the existing C_P, as shown in Figure 2. This change in capacitance is measured by the CapSense circuitry to detect the proximity of the target object.

Figure 2. Electric Field Lines Coupling to Finger



C_X = Total capacitance measured by the proximity-sensing system based on CapSense

C_P = Sensor parasitic capacitance

C_F = Capacitance added by a nearby target object

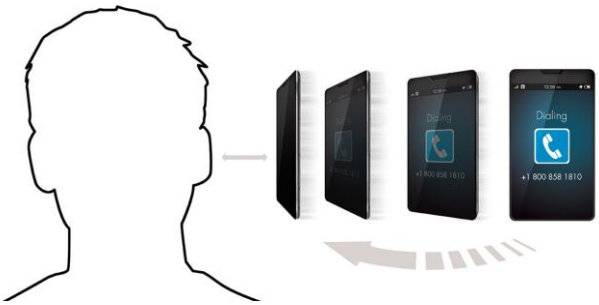
3 Proximity-Sensing Applications Based on CapSense

Proximity sensing based on CapSense is used in a variety of applications such as the following:

- Face detection in mobile phones and tablets
- Specific absorption rate (SAR) regulation in tablets and mobile phones
- Wake-on-approach feature in battery-powered applications
- Gesture detection in a human-machine interface (HMI)
- Replacement for infrared (IR) sensors in applications such as automatic door openers, backlight control in control panels, towel dispensers, faucets, and soap dispensers

Face detection in mobile phones and tablets: Face detection is a feature in mobile phones that disables the phone's touchscreen and dims the brightness of the display when a user answers a phone call, as Figure 3 shows. Face detection prevents false touches when the phone is placed on the ear and optimizes the device's power consumption. Using proximity sensing based on CapSense in this application offers advantages over IR proximity sensing because it does not require cutouts in the overlay material, which reduces the tooling cost and improves the aesthetics of the end product.

Figure 3. Face Detection in Mobile Phones



SAR regulation in tablets and mobile phones: SAR is a measure of the rate at which energy is absorbed by the human body when exposed to an RF electromagnetic field. Regulatory bodies like the Federal Communications Commission (FCC) require devices to limit the absorption of RF energy by reducing the RF transmit power of the device when the device is in close proximity to the human body, as Figure 4 shows. Proximity sensors based on CapSense can be used to detect the proximity of the human body and reduce RF power.

Figure 4. SAR Regulation in Tablets

Wake-on-approach: This feature activates the system from the sleep or standby mode when an object approaches the system, as Figure 5 shows. Wake-on-approach is also used to control the backlight LEDs when the user approaches the system. This feature reduces system wakeup time, improves device responsiveness, reduces device power consumption, and improves aesthetics. It is useful in battery-operated applications such as mice and keyboards.

Figure 5. Wake-on-Approach Implemented in a Mouse

Gesture detection: Gesture detection is the technique of interpreting human body movements and providing gesture-type information to the device. Gesture-based user interfaces provide an intuitive way for the user to interact with the system, improving the user experience. Gesture detection is used in applications such as laptops, tablets, and mobile phones for controlling the user interface.

Proximity sensors based on CapSense can be used in these applications to detect gestures without any physical contact between the user and the device. Figure 6 shows an example of an implementation of gesture detection in a laptop where proximity sensors placed near the trackpad are used for the pan movement of the onscreen map.

Figure 6. Gesture Detection Implementation in a Laptop

IR replacement: Proximity sensors based on CapSense can replace IR proximity sensors in applications such as faucets and soap dispensers, as Figure 7 shows.

Proximity sensing based on CapSense offers the following advantages over IR proximity sensing:

- It is a low-cost solution compared to IR proximity sensing. Proximity sensors based on CapSense can be constructed using a copper trace on the PCB, whereas IR proximity sensing requires extra IR sensors.
- Proximity sensors based on CapSense do not require any cutout in the overlay material to detect proximity sensing, unlike IR proximity sensors. Therefore, proximity sensing based on CapSense reduces the tooling cost and improves the aesthetics of the end product.
- They consume less power than IR proximity sensors.
- They are immune to ambient light, whereas IR-based proximity sensors can have performance issues with varying ambient light.

Figure 7. Proximity Sensing Based on CapSense in a Soap Dispenser

4 Implementing Proximity Sensing with CapSense

Figure 8 shows the steps for designing a proximity-sensing system based on CapSense.

1. **Understand proximity sensing:** See the “Proximity Sensing” section in the Getting Started with CapSense design guide for an explanation of how proximity sensing based on CapSense works and a description of the parameters that affect the proximity-sensing distance.
2. **Evaluate how proximity sensing works:** Use Cypress’s CY8CKIT-024 CapSense Proximity Shield.

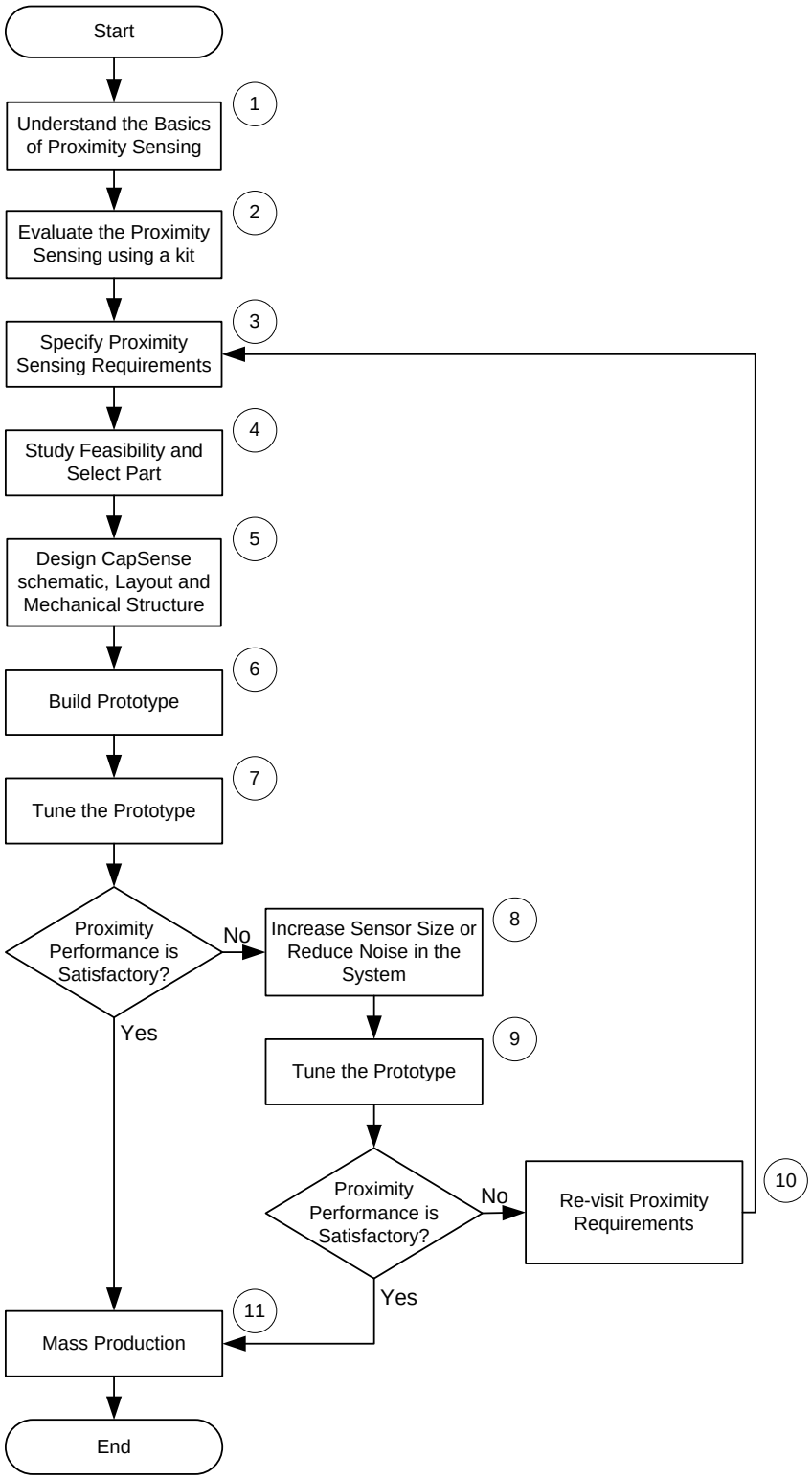
3. **Specify the proximity-sensing requirements:** After evaluating the proximity sensor performance, specify the proximity-sensing requirements such as the required proximity-sensing distance, area available on the PCB for sensor construction, system power consumption requirements, and EMI/ESD performance. These requirements help you to select the right CapSense device and design the sensor layout.
4. **Select the right CapSense device:** After the requirements are finalized, refer to the “CapSense Selector Guide” chapter in the Getting Started with CapSense design guide to select the right CapSense device based on the required proximity-sensing distance.
5. **Design the schematic and layout:** After selecting the CapSense device, design the schematic and layout. See Layout Guidelines, the device datasheet, and device-specific CapSense Design Guides.
6. **Build the prototype:** After the schematic and layout design is completed, build the prototype of the design to check if the design meets the performance requirements.
7. **Tune:** Tune the prototype board to achieve the required performance. See Tuning CapSense CSD Parameters for a Large Proximity-Sensing Distance and device-specific CapSense Design Guides.

   After tuning the sensor, check if the proximity sensor performance meets the requirements. If the requirements are met, proceed to Step 11 otherwise continue with Step 8.
8. **Redesign if necessary:** If the proximity sensor does not provide the required performance after you have set the optimum parameters, increase the sensor size or reduce the noise in the system by shielding the sensor from noise sources and continue with Step 9.
9. **Retune:** After redesigning the proximity sensor, retune the sensor and check if the sensor performance meets the requirements. If the requirements are met, proceed to Step 11; otherwise continue with Step 10.
10. **Revisit the design or requirements:** If the proximity sensor does not meet the required performance even after you have changed the sensor dimensions to the maximum possible value and tuned it with the optimum parameters, you need to revisit the requirements.

    If you are not able to achieve the required proximity-sensing distance, select a device that has a better proximity performance than the current device.

    If you are not able to achieve the required proximity-sensing distance even with the best device, you need to change the proximity sensor requirements, such as the area available for the sensor or the required proximity-sensing distance, and repeat the procedure from Step 3.
11. **Proceed to mass production:** If the proximity sensor meets the required performance, you can proceed to mass production.

Figure 8. Design Flow of a Proximity-Sensing System Based on CapSense



5 Challenges in Implementing Proximity Sensing Based on CapSense

As a designer, you may face several challenges, such as the following, when implementing a proximity-sensing system based on CapSense.

- **Achieving a large proximity-sensing distance:** Proximity-sensing applications based on CapSense require a proximity-sensing distance of 1 to 20 cm. Achieving a large proximity-sensing distance in an end system is a challenge because the proximity-sensing distance depends on various factors such as the sensor layout, sensor tuning, presence of noise sources, and floating or grounded conductive objects. Refer to the “Proximity Sensing” section in the Getting Started with CapSense design guide for a detailed explanation of the factors that affect the proximity-sensing distance.
- **Reducing the noise:** Proximity sensors are more susceptible to noise because of their large sensor area and high-sensitivity setting. High noise makes it difficult to achieve a signal-to-noise ratio (SNR) greater than 5:1, which is required for reliable proximity sensing.
- **Implementing liquid-tolerant proximity sensing:** Proximity sensors are used in applications such as faucets and soap dispensers. These applications require a robust operation even in the presence of water droplets and other liquids. In these applications, the proximity sensor should be carefully tuned to avoid false triggers when liquid droplets fall on the proximity sensor.
- **Implementing gesture detection with proximity sensors:** Reliable gesture detection based on proximity sensing requires a large proximity-sensing distance. Choosing the right type of proximity sensor, sensor size, and sensor placement plays an important role in implementing a reliable gesture detection system.

6 Layout Guidelines

The sensor layout plays a crucial role in achieving the required proximity-sensing distance. Following the layout best practices helps in achieving low sensor C_P, a large proximity-sensing distance, and low power consumption. A proximity sensor with a low C_P value has a high sensitivity and a large proximity-sensing distance.

6.1 Sensor Design

Proximity sensors can be constructed using one of the following methods:

- Button sensor
- Sensor ganging
- PCB trace
- Wire loop

These sensor implementation methods are explained in detail in the “Proximity Sensing” section in the Getting Started with CapSense design guide. Selection of the sensor type depends on the required proximity-sensing distance and the area available for the sensor on the PCB. Table 1 shows when to select a particular sensor implementation method.

Table 1. Selecting the Proximity Sensor Type

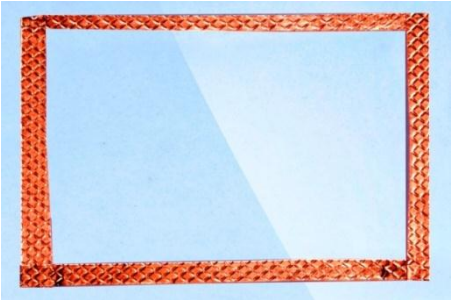
Proximity Sensor Type	When to Use
Button sensor	Use this method when the required proximity-sensing distance or the area available for the sensor is very small.
Sensor ganging	Use this method when there is no sensor pin or area available on the PCB for implementing a proximity sensor. Ganging sensors can achieve a larger proximity-sensing distance compared to using button sensors. See Implementing the Wake-on-Approach Using Sensor Ganging.
PCB trace	Use this method when the required proximity-sensing distance is very large. This method is preferred in most cases.
Wire loop	Use this method when the required proximity-sensing distance is very large. This method has the disadvantage of higher manufacturing cost compared to the implementation with PCB trace.

6.2 Sensor Size

The proximity sensor size depends on various factors such as the required proximity-sensing distance, presence of noise sources, and floating or grounded conductive objects. Noise sources and floating or grounded conductive objects reduce the SNR and the proximity-sensing distance. Therefore, large proximity sensors are needed to achieve the proximity-sensing distance required in your design.

It is very difficult to derive a relationship between the sensor size and the proximity-sensing distance. Depending on the end-system environment, the proximity-sensing distance may vary for a specific sensor size. You can find the sensor size required to achieve a required proximity-sensing distance by making sensor prototypes. You can use a copper foil, as Figure 9 shows, to make a quick sensor prototype to determine the sensor size required to achieve the required proximity-sensing distance.

Figure 9. Proximity Sensor Prototype Using Copper Tape on a Plastic Substrate



As a rule of thumb, start with a sensor loop diameter (in the case of a circular loop) or a diagonal (in the case of a square loop) equal to the required proximity-sensing distance. If you are not able to achieve the required proximity-sensing distance with this design, you can increase the sensor loop diameter or diagonal until the required proximity-sensing distance is achieved.

For a loop sensor, the minimum recommended width of the PCB trace is 1.5 mm. For a wire-loop sensor, you can use either a single-stranded or multistranded wire as the proximity sensor. The thickness of the wire has a very small impact on the proximity-sensing distance.

Table 2 summarizes the proximity sensor layout guidelines. If the area available for the proximity sensor is less than the area required to achieve the required proximity-sensing distance, you can implement firmware filters such as the Advanced Low-Pass Filter (ALP). The ALP filter attenuates the noise in the sensor raw count and increases the SNR. An increase in SNR results in a large proximity-sensing distance.

Table 2. Proximity Sensor Layout Recommendations

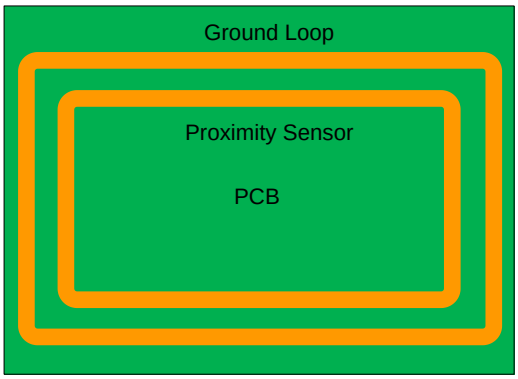
Details	Minimum	Recommendation
Proximity sensor loop diameter or diagonal	The sensor loop diameter or diagonal should at least be equal to the required proximity-sensing distance if the ALP filter is disabled. If the ALP filter is enabled, the sensor loop diameter or diagonal should at least be equal to half of the required proximity-sensing distance.	Start with a sensor loop diameter or diagonal equal to the required proximity-sensing distance and increase the diameter or diagonal until the required proximity-sensing distance is achieved.
Proximity sensor trace width	1.5 mm	1.5 mm

6.3 Sensor Layout

Keep the following points in mind while designing the proximity sensor layout:

- **Maximize the size of the sensor:** The proximity-sensing distance is directly proportional to the sensor area. The larger the proximity sensor area, the larger the proximity-sensing distance. However, a large sensor area results in a high sensor C_P and high noise, and thus reduces the proximity-sensing distance. Using a loop sensor instead of a solid-fill sensor results in a low sensor C_P, low noise and thus a large proximity-sensing distance. Also, loop sensors require less sensor area, providing more space to place components on the PCB.
- **Reduce the noise:** Noise in the sensor raw count reduces the SNR and the proximity-sensing distance. To attenuate the noise in the sensor output and improve the ESD performance, surround the proximity sensor with a ground loop, as Figure 10 shows. However, keep in mind that a ground loop around the sensor reduces the proximity-sensing distance. The minimum recommended width for the ground loop is 1.5 mm, and the minimum recommended clearance between the proximity sensor and the ground loop is 1 mm.

Figure 10. Ground Loop Surrounding the Proximity Sensor



- **Reduce sensor parasitic capacitance:** The proximity-sensing distance depends on the ratio of the C_F to the C_P. The proximity-sensing distance increases with an increase in the C_F/C_P ratio. For a given sensor size, the value of C_F depends on the distance between the sensor and the target object. To maximize this ratio, you need to increase C_F and decrease C_P. The C_P of the sensor can be minimized by selecting an optimum sensor area, reducing the sensor trace length, and minimizing the coupling of the sensor electric field lines to the ground.

  To reduce the coupling of sensor electric field lines to the ground, drive the hatch pattern in the top and bottom layer of the PCB (if there is any) with the driven shield signal. A hatch pattern that is connected to the driven shield signal is called a “shield electrode.” The driven shield signal is a replica of the sensor signal.

  For shield electrode layout guidelines, refer to the “Shield Electrode and Guard Sensor” section in the Getting Started with CapSense design guide.

  To minimize the sensor trace length and thereby the sensor C_P, place the CapSense device as close as possible to the sensor.
- **Eliminate the effect of floating or grounded conductive objects:** The proximity-sensing distance is reduced drastically if there is any nearby floating or grounded conductive object. You should either remove the nearby conductive object or use a shield electrode to isolate the proximity sensor from the conductive object.

  Note Surrounding the sensor with a ground loop with a small trace width (1.5 mm) will not reduce the proximity-sensing distance by a drastic amount.

  For more details on the effect of floating or grounded conductive objects on the proximity-sensing distance, refer to the “Proximity Sensing” section in the Getting Started with CapSense design guide. For shield-electrode layout guidelines, refer to the “Shield Electrode and Guard Sensor” section in the Getting Started with CapSense design guide.

In addition to these guidelines, you should also follow the layout best practices mentioned in the “PCB Layout Guidelines” section in the Getting Started with CapSense design guide.

7 Tuning CapSense CSD Parameters for a Large Proximity-Sensing Distance

After you have completed the proximity sensor layout, the next step is to implement the firmware and tune the CapSense CSD parameters for the proximity sensor to achieve an optimum performance. Cypress's PSoC Creator™ integrated design environment (IDE) provides a CapSense CSD Component to simplify CapSense system design. See the CapSense Component datasheet.

The capacitance added by a target object at a distance from the proximity sensor is in tens of femtofarads, unlike touching a button sensor, where the capacitance added is in hundreds of femtofarads. To detect a small change in capacitance, the CapSense circuitry should be tuned for high sensitivity, and the threshold parameters should be set to the optimum values. The process of setting CapSense CSD parameters for an optimum sensor performance is called "tuning." This section shows you how to tune the CapSense CSD parameters in the PSoC Creator IDE for a proximity sensor to achieve optimum performance.

This section assumes that you are familiar with the PSoC 4 CapSense Component. If you are new to CapSense design in PSoC 4, refer to the CapSense Component datasheet.

Figure 11 shows the high-level steps for tuning a proximity sensor implemented with a PSoC 4 device. You can follow the same procedure for tuning proximity sensors in other CapSense devices. Refer to the device-specific design guides at CapSense Design Guides for tuning procedures.

Tuning a proximity sensor in PSoC Creator is similar to tuning a button sensor and can be performed in three high-level steps:

1. Set Optimum CapSense CSD Parameters
2. Ensure SNR is Greater Than or Equal to 5:1
3. Set Optimum Threshold Parameters

7.1 Set Optimum CapSense CSD Parameters

1. Determine the sense clock divider and modulator clock divider values for the proximity sensor.

   The sense clock divider determines the frequency of the precharge switch output, while the modulator clock divider determines the CapSense CSD modulator input frequency. Using Cypress's SmartSense™ algorithm, you can easily determine the sense clock divider and modulator clock divider values for a given sensor layout. Refer to Appendix A: Determining the Sense Clock Divider and Modulator Clock Divider Using SmartSense for details on how to set these parameters using the SmartSense algorithm.

2. Set the general properties of the CapSense Component.

   In the PSoC Creator *TopDesign.cysch* window, double-click on the CapSense Component to configure the CapSense CSD parameters. In the CapSense Component **General** configuration window, set the following parameters to the values listed in Table 3.

3. Add a proximity sensor to your design.

   Click the **Add proximity sensor** button in the CapSense Component **Widgets Config** window. Set the resolution of each proximity sensor to 16 bits to achieve a large proximity-sensing distance. Leave the other parameters unchanged. These parameters will be set in Step 9 .

4. Set the advanced properties of the CapSense Component.

   In the CapSense Component **Advanced** configuration window, set the parameters to the values listed in Table 4. Leave the other parameters unchanged. These parameters will be set in Step 9.

   Refer to the PSoC 4 CapSense Component datasheet or PSoC 4 CapSense Design Guide for more details on these parameters

.

Figure 11. Proximity Sensor Tuning Flow Chart

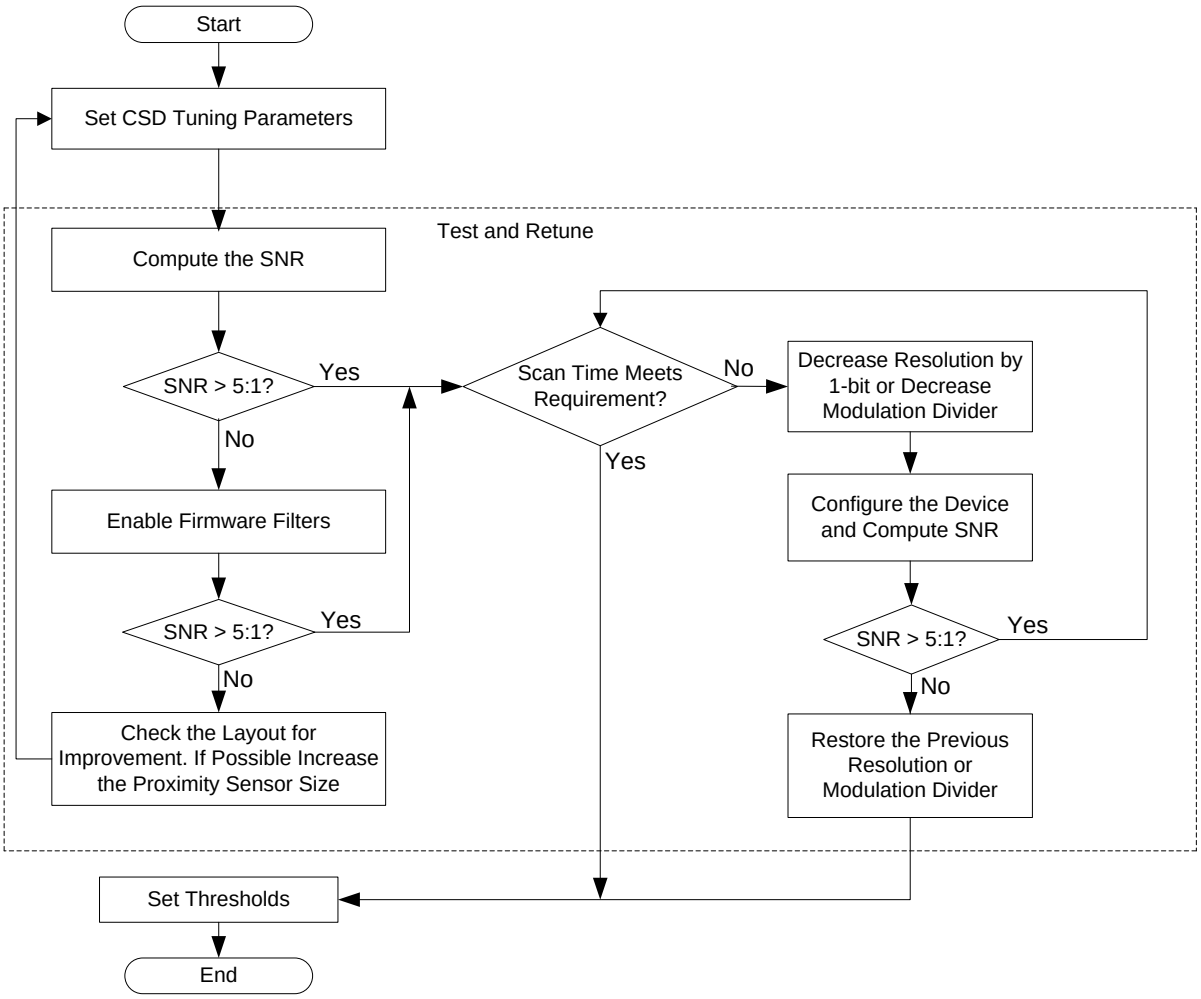


Table 3. CapSense Component General Configuration Window

Parameter	Value	Rationale
Tuning method	Manual with run-time tuning	The manual method with the run-time tuning option allows you to change the tuning parameters during run time.
Raw-data noise filter	None	Filters will be enabled in Step 7, depending on the SNR of the sensor.
Compensation IDAC	Enabled	Enabling the compensation IDAC selects the dual-IDAC mode operation of the CSD. Dual-IDAC mode gives higher signal values compared to single-IDAC mode.
Auto-calibration	Checked	Enabling auto-calibration maintains the tuning by automatically calibrating the IDACs when the sensor C_P value varies from one PCB to another.
Waterproofing and detection	Unchecked	This parameter should be enabled when your design has a guard sensor to eliminate sensor false triggers due to liquid flow over the sensor.
Enable built-in self-test	Unchecked	This parameter is not enabled because you are not using the C_P measurement API to measure the sensor C_P value. You are using SmartSense to determine the sensor parameters.

Table 4. CapSense Component Advanced Configuration Window

Parameter	Value	Rationale
Current source	IDAC sourcing (default)	The IDAC sourcing mode is recommended for most applications, as it is free from power supply noise compared to the IDAC sinking mode.
IDAC range	4x	The 4x range is sufficient for most applications. You can use the 8x range if C_P is very high.
Analog switch drive source	PRS-Auto	The pseudo random sequence (PRS) clock reduces the effects of external EMI on CapSense and reduces sensor scan emissions.
Individual frequency settings	Enabled	This option is enabled to specify the sense clock divider and modulator clock divider for each sensor in the **Scan order** tab.
Sensor auto reset	Disabled (default)	This option is set to “Disabled” to ensure that the baseline is not always updated. Updating the baseline may result in a reduced proximity-sensing distance when the target object approaches the sensor very slowly.
Widget resolution	16-bit	Proximity sensors may have a difference count with a value greater than 255 (8-bit). Therefore, a widget resolution of 16-bit is selected to ensure that the difference count is not truncated when it is greater than 255. Set this value to 8-bit if you are expecting the difference count value to be less than 255.
Shield	Enabled	This option allows you to drive the shield electrode with the driven shield signal.
Shield signal delay	None (default)/OFF	For PSoC 4000 devices, set this parameter to “OFF.” When the driven shield signal is not in phase with the sensor signal, this parameter should be set to either “10 ns” or “50 ns.” For PSoC 4200 devices, set this parameter to “None (default).” When the driven shield signal is not in phase with the sensor signal, this parameter should be set to either “1 cycle” or “2 cycle.”
Shield tank capacitor enable	Enabled	It is recommended to use the shield-tank capacitor to reduce switching transients when the shield electrode is charged and discharged.
Guard sensor	Disabled	The guard sensor should be enabled only when you need to eliminate false triggers due to liquid flow over the sensor.
Inactive sensor connection	Shield	Inactive sensors are connected to shield instead of ground because a nearby inactive sensor creates grounding effects on the active proximity sensor and reduces the proximity-sensing distance.
Shield tank capacitor Precharge Settings	Precharge by IO buffer	The precharge by IO buffer option is recommended for the Csh_tank capacitor because it can quickly charge the capacitor.

7.2 Ensure SNR is Greater Than or Equal to 5:1

After the hardware parameters are set, you need to measure the sensor SNR and ensure that it is greater than or equal to 5:1. An SNR of 5:1 ensures robust operation under all conditions.

SNR is the ratio of the sensor signal and the peak-to-peak noise counts, as shown in Equation 1.

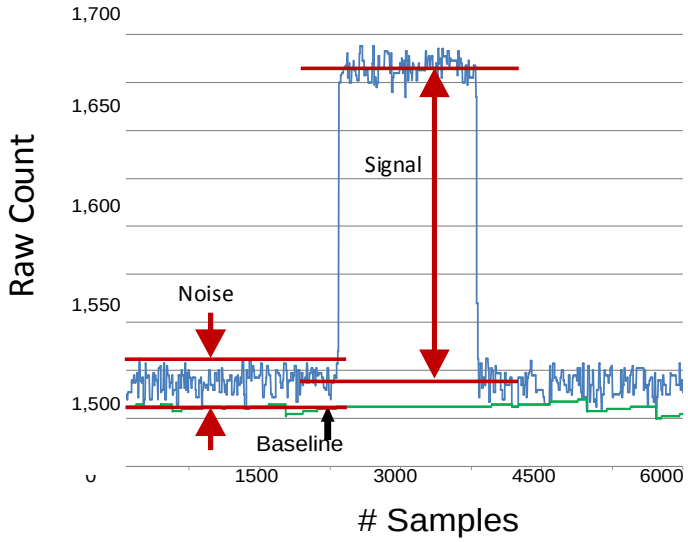
$$\text{SNR} = \frac{Signal}{Peak\text{-}to\text{-}Peak\ Noise} \qquad \text{Equation 1}$$

Where:

Signal = Raw count (with hand) – raw count (without hand)

Peak-to-Peak Noise = Peak-to-peak noise of raw count measured over 3,000 samples, as shown in Figure 12.

Figure 12. Computing SNR



You can compute the SNR using two methods:

- CapSense Tuner: Using the CapSense Tuner is the easiest method for computing the SNR. However, this method supports only I²C communication to read the sensor data and requires you to run a specific set of APIs in the firmware. See the “Tuner GUI Interface” section in the PSoC 4 CapSense Component datasheet.

- Bridge Control Panel (BCP): The BCP is a tool provided by Cypress to read data from a slave device via an I²C/SPI/UART interface. For more details on how to use the BCP for reading data from a slave device, refer to AN2397 – CapSense Data Viewing Tools. Appendix B: Reading Sensor Debug Data in the Bridge Control Panel shows you how to measure the SNR using the BCP tool.

Depending on your requirements, select a suitable method for measuring the SNR and implement the firmware. The Example Projects provided with this application note implement both methods. You can use that section as a reference and implement the method for computing the SNR. Steps 5 to 7 show you how to achieve an SNR greater than 5:1.

5. Program the device with the settings shown in Steps 1 to 4.
6. To compute the SNR, acquire 3,000 raw count samples, as Figure 12 shows, and measure the peak-to-peak noise count. Place your hand at the required proximity-sensing distance and measure the shift in raw counts. The signal will be equal to the raw count (after placing the hand) minus the raw count (before placing the hand.)
7. If the SNR computed in Step 6 is greater than 5:1, proceed to Step 8; otherwise, enable the ALP filter and measure the SNR.

   Simple filters such as median, average, and infinite impulse response (IIR) may not be able to attenuate the high noise in proximity sensors, so you need to use the ALP filter. The ALP filter is designed specifically for attenuating noise in the proximity sensor and providing a fast response time.

   If the SNR is not greater than 5:1 after enabling the ALP filter, check for layout improvements such as shielding the sensor from noise sources to reduce noise or increasing the sensor loop diameter or diagonal to increase the signal.

   Restart from Step 1 if you increase the sensor loop diameter or diagonal.

8. Once the SNR is greater than 5:1, check if the scan time and power consumption meet the requirements.

   If the scan time and power consumption meet the requirements, proceed to Step 9; otherwise, decrease the sensor resolution or modulator clock divider and repeat from Step 5.

7.3 Set Optimum Threshold Parameters

9. If the SNR is greater than 5:1 and the scan time and power consumption requirements are met, set the threshold parameters listed in Table 5.

Table 5. Sensor Threshold Parameters

Threshold Parameter	Tab Present	Recommended Value
Finger threshold	**Widgets Config**	80 percent of signal
Noise threshold		40 percent of signal
Negative noise threshold	**Advanced**	40 percent of signal
Hysteresis	**Widgets Config**	10 percent of signal
Debounce	**Widgets Config**	Set this parameter to '1' if you are using the ALP filter; otherwise, set it to '3'
Low baseline reset	**Advanced**	30

Example Project 1 – Proximity Distance follows this procedure and shows how to achieve a 10-cm proximity-sensing distance with a square-loop proximity sensor whose diagonal length is 10.3 cm.

8 Firmware Filters for Reducing Noise

As explained previously, proximity sensors are susceptible to noise because of their large sensor area and high-sensitivity setting. Filters help to reduce the noise in raw count and improve the SNR. A higher SNR implies a large proximity-sensing distance.

The CapSense Component supports the following types of filters: median, average, and first-order IIR. Refer to the CapSense Component datasheet for more details on these filters. The noise attenuation factor of these filters depends on the order of the filter. The higher the filter order, the higher the noise attenuation will be. However, higher order filters require more memory and result in a slower response.

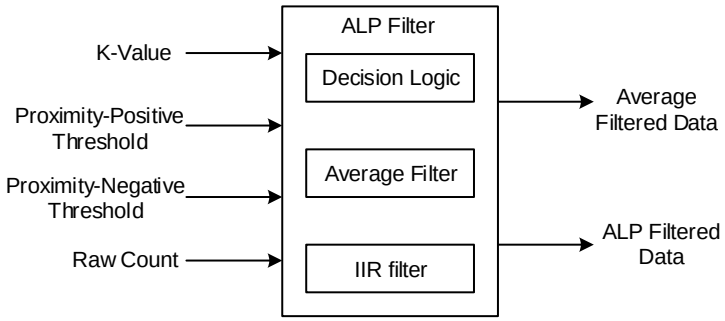
To achieve high noise attenuation and improve the response time, you need to use intelligent adaptive filters such as the ALP filter, as explained in the next section.

8.1 Advanced Low-Pass Filter

The ALP filter is a combination of multiple low-pass filters specifically designed to attenuate the noise in the proximity sensor raw count. Figure 13 shows the block diagram of the ALP filter. The ALP filter switches between multiple low-pass filters depending on the sensor signal and values of the threshold parameters to achieve the maximum noise attenuation and provide a fast response time.

The ALP filter has a slow-response filter and a fast-response filter. The slow-response filter provides the maximum noise attenuation, but its response time is slow. On the other hand, the fast-response filter provides a fast response time but results in less noise attenuation. By switching between these two filters, the ALP filter provides the maximum noise attenuation and a fast response time.

Figure 13. ALP Filter Block Diagram



The inputs to the ALP filter are as follows:

- **Raw count:** The raw count of the sensor is the digital measurement of the capacitance of the sensor.
- **K-value:** The K-value of the ALP filter determines noise attenuation in the proximity sensor raw count. The K-value can be one of the following:
  - IIR_K_16
  - IIR_K_32
  - IIR_K_64

  Noise attenuation decreases in the following order:

  IIR_K_64 > IIR_K_32 > IIR_K_16
- **Proximity-positive threshold:** This parameter determines the turn-on time of the proximity sensor when a hand approaches it. When the sensor signal is greater than this value, the ALP filter switches to the fast-response filter from the slow-response filter.
- **Proximity-negative threshold:** This parameter determines the turn-off time of the proximity sensor when a hand is withdrawn from it. When the sensor signal is less than this value, the ALP filter switches to the fast-response filter from the slow-response filter.

The outputs of the ALP filter are as follows:

- **Average filtered data:** Average filtered data is used to set the proximity-positive threshold and proximity-negative threshold parameters.
- **ALP filtered data:** The ALP filtered data is the final output of the filter, that is, the raw count with very low noise.

8.1.1 Adding an ALP Filter to the Project

Refer to Appendix D: Adding the ALP Filter Library to Any CapSense Project for adding ALP filter to your project.

Note: The ALP filter requires all the proximity sensors to occupy a scan order 0 through (n-1) in the CapSense Component. Here, N is the total number of proximity sensors in the design. For proximity sensors to occupy the scan order from 0, place the proximity sensor widget first followed by other sensor widgets.

8.1.2 ALP Filter Tuning

The ALP filter requires you to specify the K-value, proximity-positive threshold, and proximity-negative threshold for proper operation. Follow these steps to set the ALP filter parameters.

1. Measure the peak-to-peak noise using the raw count without any nearby target object and filters.
2. Set the K-value per the mapping in Table 6.

Table 6. Selecting the K-Value

Peak-to-Peak Noise	Recommended K-Value
Less than 32 counts	IIR_K_16
Greater than 32 counts and less than 64 counts	IIR_K_32
Greater than 64 counts	IIR_K_64

3. Enable the ALP filter and program the device. Measure the peak-to-peak noise in the ALP filter's average filtered data.
4. Set the proximity-positive threshold as equal to 1.5 × peak-to-peak noise of the average filtered data.
5. Set the proximity-negative threshold as equal to 0.5 × peak-to-peak noise of the average filtered data.
6. Set the finger threshold parameter as equal to the proximity-positive threshold. After tuning the ALP filter, the finger threshold parameter should be set to the value listed in Table 5.
7. Program the device with these settings and measure the peak-to-peak noise using the raw count for 3,000 samples.
8. Place your hand at the required proximity-sensing distance and measure the signal, that is, the shift in the raw count when the hand approaches the sensor.
9. Compute the SNR. If the SNR is greater than 5:1, check if the sensor turn-off time is acceptable. If the sensor turn-off time is very slow, increase the proximity-negative threshold value. The maximum limit for the proximity-negative threshold parameter is equal to the proximity-positive threshold value.
10. If the SNR is greater than 5:1 and the sensor response time meets the requirements, proceed to set the threshold parameters listed in Table 5; otherwise, check for layout improvements to minimize the noise or increase the proximity sensor size to increase the signal.

Example Project 1 – Proximity Distance shows how to use the ALP filter. The ALP filter is provided as a library file. You can call the ALP filter APIs to apply the ALP filter on the sensor raw count. The ALP filter API definitions are provided in Appendix C: Advanced Low-Pass Filter API Definitions.

9 Tuning for Liquid Tolerance

Water droplets or any other liquids may create false triggers when they fall on the proximity sensor. When a liquid falls on the proximity sensor, it adds a capacitance and results in a signal (i.e. shift in raw count) that is equal to the signal when a hand is placed over the proximity sensor at a certain distance from it. To eliminate false triggers due to liquid droplets, Cypress recommends that you tune the CapSense CSD parameters in such a way that when a hand is placed over the proximity sensor (at the required proximity-sensing distance), the signal is at least three times greater than the signal due to liquid droplets. This ensures that the sensor will operate reliably in all conditions throughout the life cycle.

Figure 14 shows the steps to tune a proximity sensor for liquid tolerance.

1. Connect the inactive sensor, hatch pattern, or any trace that is surrounding the proximity sensor to the driven shield instead of connecting them to ground. This will minimize the signal due to the liquid droplets when they fall on the sensor. The driven shield is a signal that replicates the sensor-switching signal. Refer to the "Shield Electrode and Guard Sensor" section in the PSoC 4 CapSense Design Guide for details on how the driven shield works in a PSoC 4 device.
2. Follow the tuning steps explained in the Tuning CapSense CSD Parameters for a Large Proximity-Sensing Distance section to tune the CapSense CSD parameters to achieve an SNR greater than 5:1. This step is to ensure that the SNR of the proximity sensor is greater than 5:1 without liquid droplets.
3. Place a liquid droplet (quantity depends on requirements) over the proximity sensor and measure the signal, that is, the shift in the raw count when a liquid droplet falls on the sensor.

4. Bring your hand towards the sensor and find the distance at which the signal due to the hand is at least three times greater than the signal due to the liquid droplet. This distance is the maximum possible proximity-sensing distance that can be achieved with liquid tolerance for this sensor layout. If the achieved proximity-sensing distance meets your requirement, proceed to Step 5; otherwise, increase the sensor loop diameter or diagonal to achieve the required proximity-sensing distance and restart from Step 1.

   Note There should not be any liquid droplets on the sensor while measuring the signal due to the hand.

5. To reduce the scan time and power consumption, reduce the resolution by one bit and check for the following conditions:

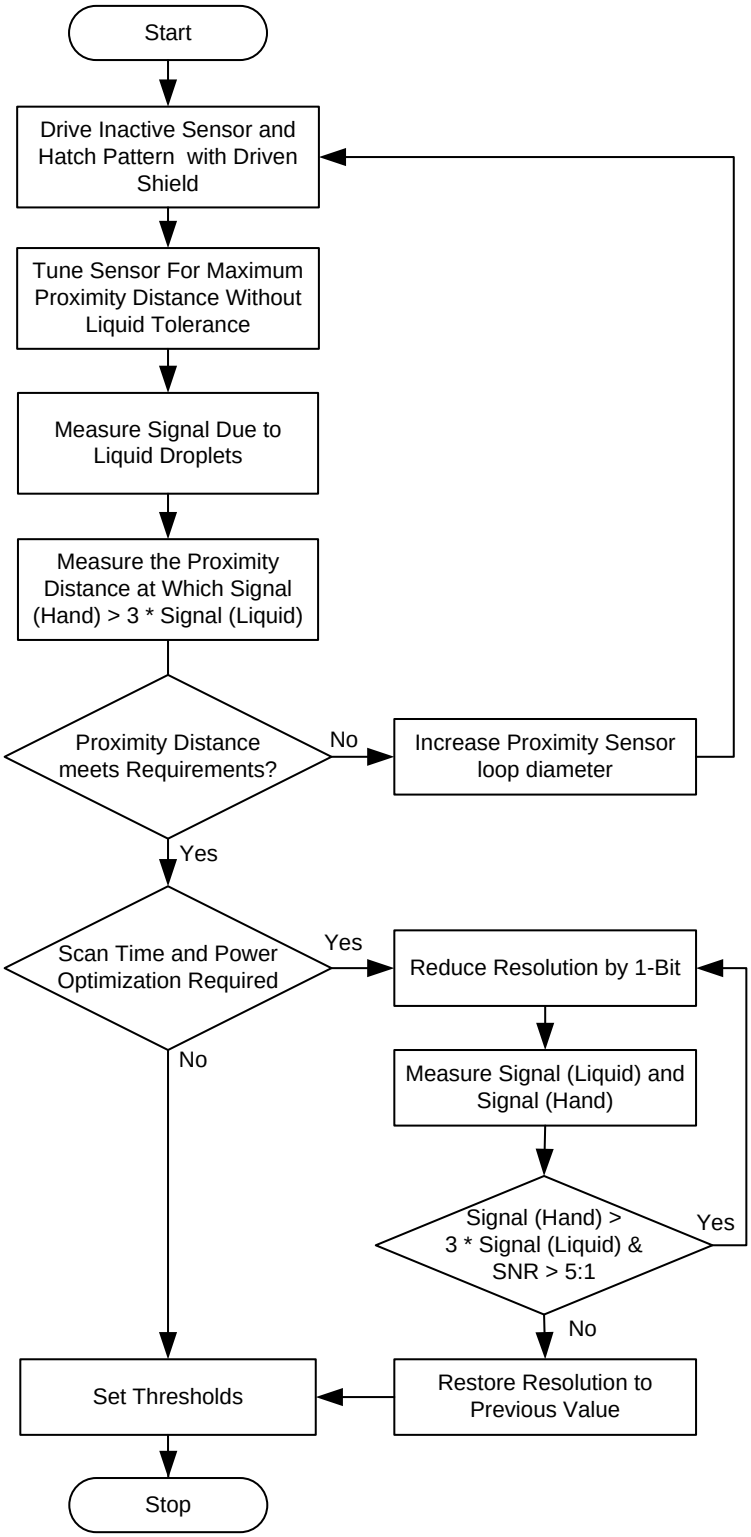
   - SNR is greater than or equal to 5:1.
   - The signal due to the liquid droplet and the signal due to the hand at the new proximity-sensing distance (measured in Step 4) meet the following requirement:

     Signal (hand) > 3 * Signal (liquid droplet)

   If these conditions are met, you can further reduce the resolution until one of the conditions fails. If either of the conditions fails, restore the resolution to the previous value and continue with Step 6.

6. After the signal due to the hand is greater than three times the signal due to the liquid droplet, set the threshold parameters to the values indicated in Table 5. Example Project 2 – Liquid-Tolerant Proximity shows you how to tune a proximity sensor for liquid tolerance.

Figure 14. Flow Chart for Tuning Proximity Sensor for Liquid Tolerance



9.1 Implementing Gesture Detection with Proximity Sensors

Implementing gesture detection with a proximity sensor based on CapSense involves the following three steps:

1. Sensor Design and Placement
2. Sensor Tuning
3. Firmware Design

9.2 Sensor Design and Placement

Proximity sensors detect gestures without any contact with the user. The proximity-sensing distance depends on the sensor type and size. For detecting gestures at a long distance, you can use a PCB trace (Figure 15) to implement a proximity sensor. Refer to the “Proximity Sensing” section in the Getting Started with CapSense design guide for guidelines on the type of sensor to select for a required proximity-sensing distance.

Proximity sensor placement depends on the type of gesture that should be detected. For example, to detect gestures such as horizontal swipes in the X-axis, you can use two proximity sensors, PS1 and PS2, which are placed parallel to the horizontal hand movement in the X-axis, as Figure 15 shows. Similarly, to detect horizontal swipes in the Y-axis, you can place two proximity sensors, PS3 and PS4, parallel to the horizontal hand movement in the Y-axis, as Figure 15 shows.

9.3 Sensor Tuning

After the sensor layout is completed, sensor tuning plays an important role in achieving the required proximity-sensing distance. Follow the steps explained in the Tuning CapSense CSD Parameters for a Large Proximity-Sensing Distance section to tune the sensors to achieve the required proximity-sensing distance.

9.4 Firmware Design

When an object is detected, the firmware algorithm interprets the sensor data and reports the gesture type. There are various firmware techniques to detect gestures based on the sensor data (difference count and sensor status).

A simple method to detect gestures is to compare the difference count or sensor status of the proximity sensors with respect to time. Each type of gesture will have a unique pattern in the difference count or the sensor status with respect to time. By detecting the difference count or sensor status pattern, you can easily determine the type of gesture.

For example, when a left-to-right swipe is performed in the X-axis, the signal from the two proximity sensors, PS1 and PS2, varies as Figure 16 shows. The hand position can be determined by splitting the difference counts of both the sensors into different regions as follows:

- Region A: Sensor PS1 is ON and PS2 is OFF. The hand is close to sensor PS1 and farther from sensor PS2.
- Region B: Both PS1 and PS2 are ON. The hand is between PS1 and PS2.
- Region C: PS2 is ON and PS1 is OFF. The hand is close to PS2 and farther from PS1.

When a left-to-right swipe is performed, PS1 will be triggered first, followed by PS2. You can determine this trigger pattern to interpret the gesture. This method can also be applied to the sensor status of proximity sensors PS3 and PS4 for detecting swipe gestures in the Y-axis.

Example Project 3 – Proximity Gestures shows how to implement gesture detection using proximity sensors based on CapSense.

Figure 15. Proximity Sensor Design and Placement for Gesture Detection

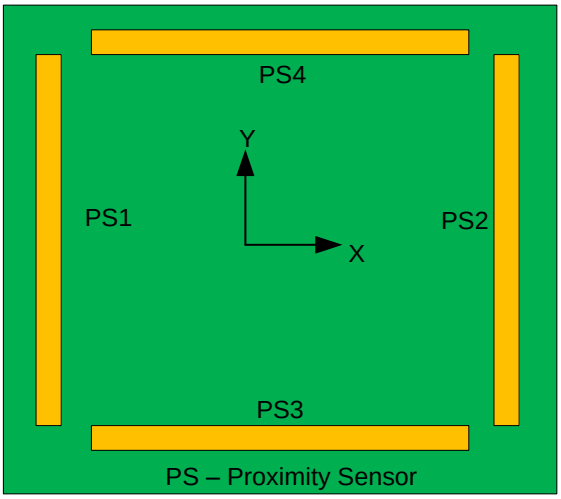
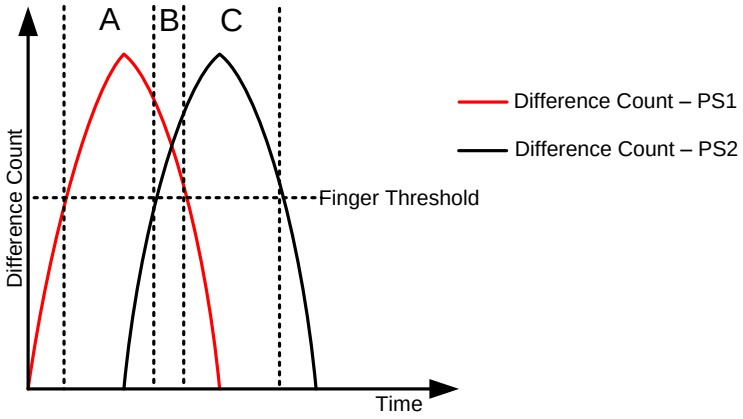


Figure 16. Horizontal Gesture Detection



10 Design Consideration to Achieve 30-cm Proximity-Sensing Distance

CapSense devices support a proximity-sensing distance of up to 30-cm under ideal conditions. The ideal conditions required for achieving a 30-cm proximity sensing distance are:

a) There should not be any noise sources surrounding the sensor.

b) There are no grounded/floating metal objects within the vicinity of the proximity sensor.

The 30-cm proximity-sensing distance was achieved under ideal conditions using the following:

- **Sensor Design**: A wire loop of 17-cm diameter or a copper trace of 1.5-mm width and 17-cm diameter was used as proximity sensor.
- **Sensor Tuning**: The tuning procedure mentioned in section Tuning CapSense CSD Parameters for a Large Proximity-Sensing Distance was followed to tune CapSense CSD parameter.

- **ALP Filter Tuning**: To reduce the noise and provide a fast response time, the ALP filter was applied on sensor raw counts. See Firmware Filters for Reducing Noise for details on using ALP filter.

11 Implementing Low-Power Systems Using the Wake-on-Approach

Low-power embedded design is motivated by the need to run applications for as long as possible while consuming minimum power. In a battery-powered system, this need is magnified. Furthermore, low power implies a lower cost of operation and a smaller battery size to make applications more mobile. When energy comes at a premium as it does with today's green initiatives, ensuring that an embedded system consumes as little energy as possible is even important for wall-powered applications.

In an embedded system, to reduce power consumption, designers use low-power modes such as sleep and deep-sleep. In an embedded system, the device performs its operation and enters a low-power mode. To perform the operations required by the system, the device must wake up from the low-power mode.

Typical applications use one of the following methods:

1. Periodically wake up and perform the required operations and remain in the low-power mode for most of the time. In this case, the system wakes up from low-power mode due to a periodic interrupt source. One such source is a watchdog timer.
2. Remain in the low power mode unless an external event such as a GPIO interrupt is triggered or an I^2C address match needs attention.

These methods are conventional ways to achieve low average power. Method 2 has lower average power consumption compared to method 1 because in method 2, the system is in low-power mode until the user interacts with the system. However, with method 2, the response of the system is very slow. An intelligent system should detect the presence of the user before the user interacts with the system and wake up the system from a low-power mode. This method of waking the system from a low-power mode by detecting the presence of a user is called "wake-on-approach."

The wake-on-approach feature uses proximity sensors to detect the presence of a user to wake the system. This results in very low average power consumption and provides a quick response to user interactions.

This feature is useful in the following cases:

1. To reduce the power consumption of the entire system: In this case, the entire system except the CapSense device can be put in sleep mode. The proximity sensor implemented with the CapSense device detects the presence of a user and wakes the system up.
2. To reduce the average power consumption of the CapSense device: In this case, instead of scanning all the CapSense sensors in the system, you can implement a proximity sensor and scan only the proximity sensor. Alternatively, you can gang all the sensors and scan only the ganged sensor to check for proximity. When proximity is detected, you can scan all the sensors and detect finger touches.

Method 1 is very easy to implement. You need to use the ON/OFF status of a proximity sensor to switch the system between active mode and sleep mode. Method 2 is useful only in a few cases. This section shows you when and how to implement method 2 to reduce the power consumption of a CapSense system.

In a CapSense system, the wake-on-approach feature is useful only when the total time taken to scan all the sensors (excluding the time taken to scan the proximity sensors used for this feature) is greater than the time taken to scan the proximity sensor, which will be used to implement it. The scan time of a sensor depends on the sensor C_P, CSD resolution, and modulator clock frequency.

Consider a CapSense application that has 10 CapSense buttons. Let the resolution of each sensor be 13-bit and the modulator clock divider be '2'. As Figure 17 shows, the time taken to scan each sensor is 1.365 ms, and the total time taken to scan all 10 sensors is 13.652 ms.

Consider a CapSense system that has a proximity sensor surrounding all 10 button sensors. Let the proximity sensor resolution be 16-bit and the modulator clock divider be '1'. As Figure 18 shows, the time taken to scan only the proximity sensor is 5.461 ms.

So, instead of continuously scanning all the button sensors, you can scan only the proximity sensor to detect the presence of a user. Once the proximity sensor detects the presence of a user, you can scan all the button sensors to detect finger touches. By implementing a proximity sensor, you can reduce the scan time from 13.652 ms to 5.461 ms when the user is not near the system and allow the device to sleep longer, lowering power consumption.

Note The wake-on-approach feature is not useful in applications where the time taken for scanning all the sensors is less than the time taken to scan a proximity sensor.

Figure 17. Estimating Button Sensor Scan Time

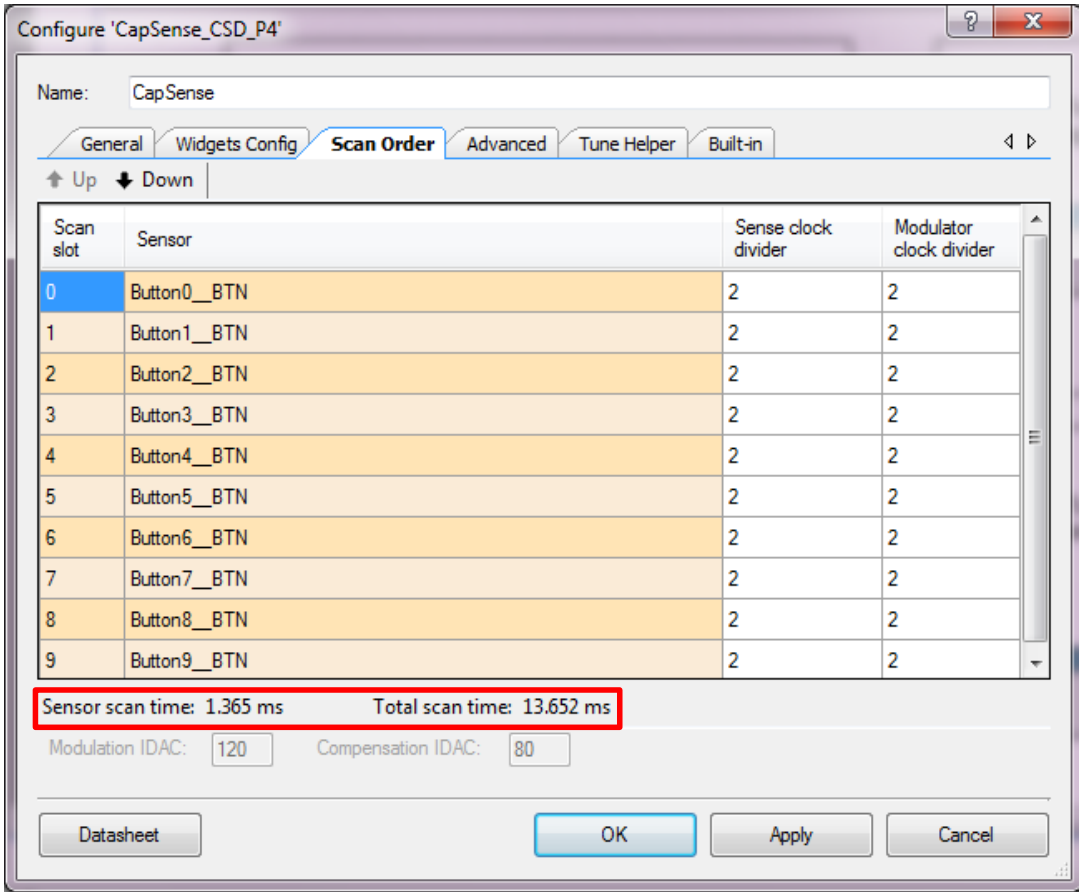
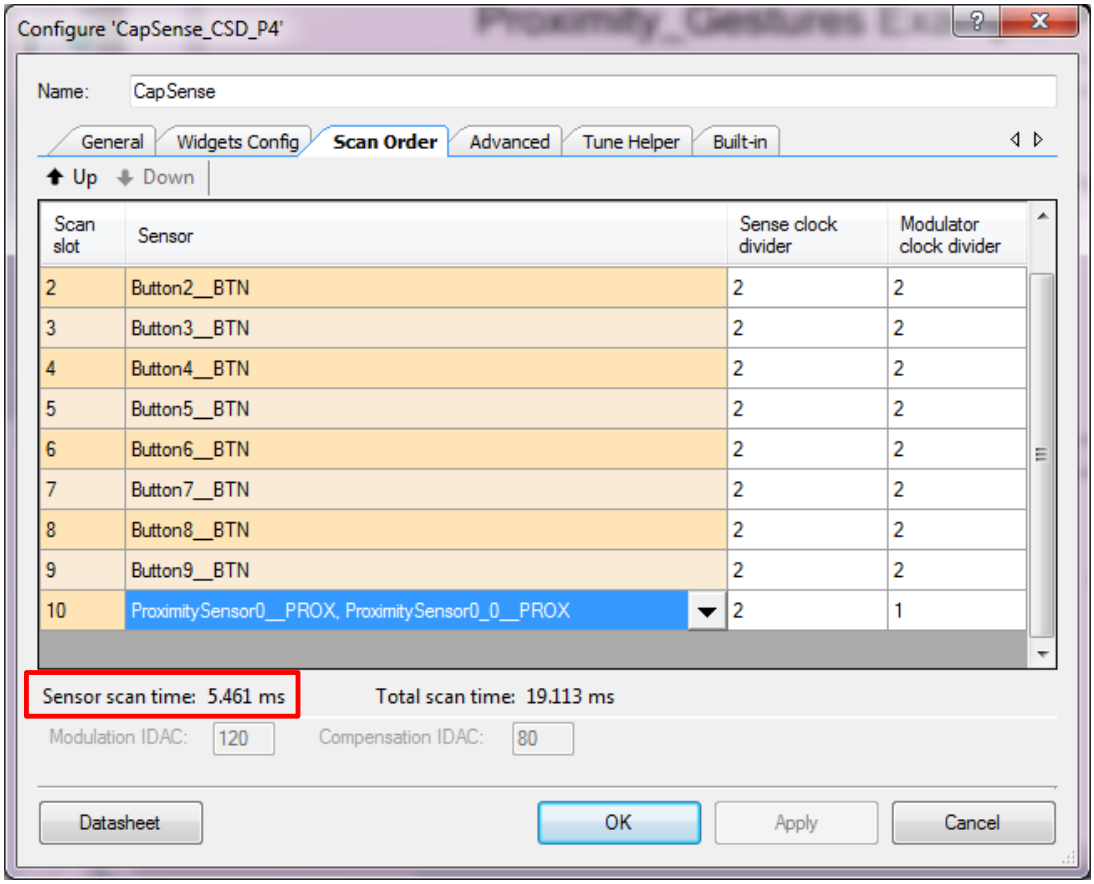


Figure 18. Estimating Proximity Sensor Scan Time



12 Implementing the Wake-on-Approach Using Sensor Ganging

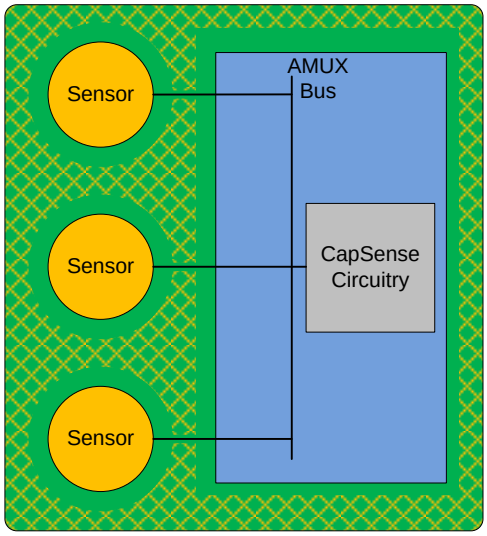
You can implement a proximity sensor by ganging multiple button sensors or proximity sensors. Ganging refers to simultaneously connecting multiple sensors to the CapSense circuitry and scanning them as a single proximity sensor, as Figure 19 shows. Ganging multiple sensors increases the effective sensor area and results in a large proximity-sensing distance.

When a user is not interacting with the system, you can gang all the sensors in the system and scan them as a single proximity sensor. When proximity is detected, the device can stop scanning the ganged sensor and scan each sensor individually and detect finger touches. Sensor ganging reduces the total sensor scan time and device power consumption.

The CapSense Component provides an easy method to implement sensor ganging. Consider a CapSense system that has five button sensors. To gang these sensors, you add a proximity widget in the **Widgets Config** tab in the CapSense Component configuration window. In the **Scan Order** tab, click on the proximity sensor. A drop-down list will appear, as Figure 20 shows. This drop-down list allows you to specify which sensors in the system should be ganged and scanned as a single proximity sensor.

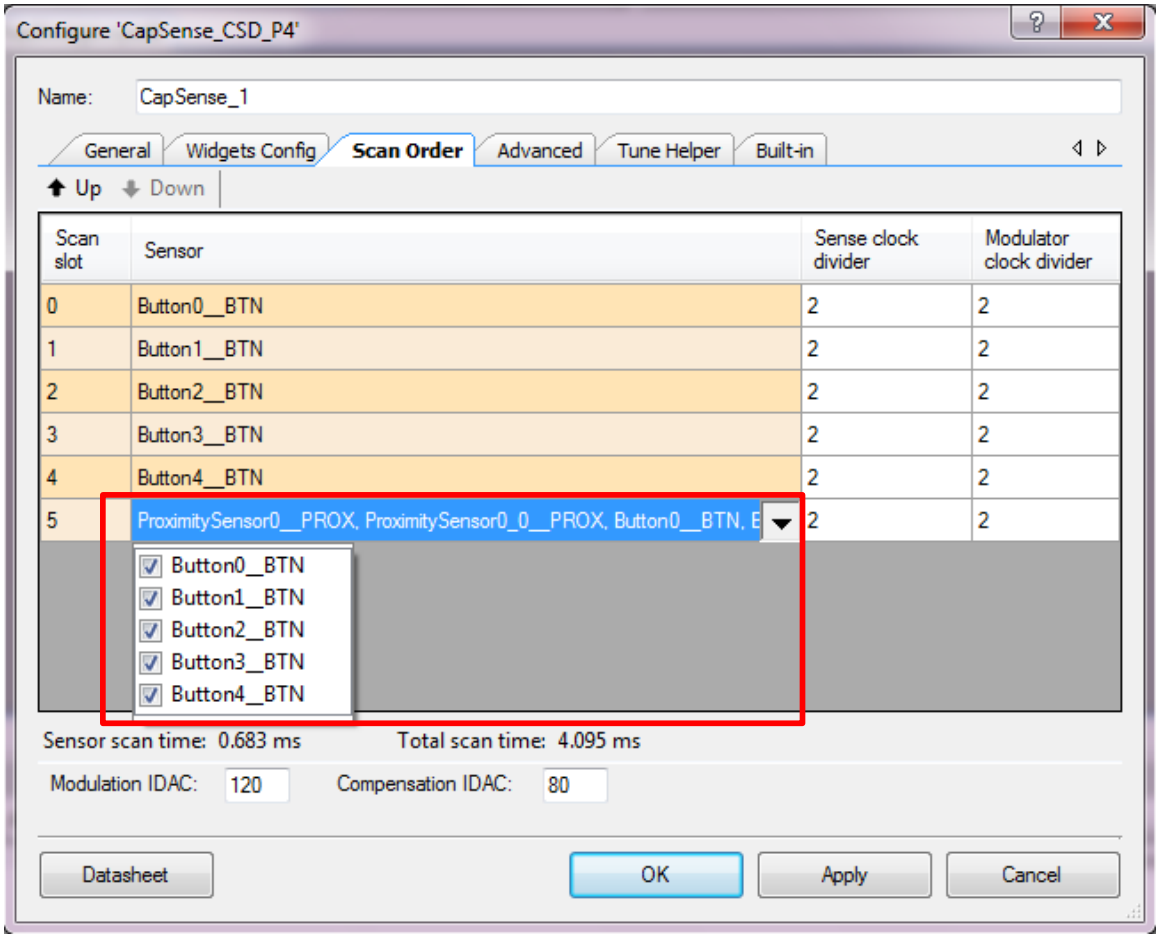
Refer to the application note AN90114 – PSoC 4000 Family Low-Power System Design Techniques for details on how to implement a low-power CapSense system using sensor ganging.

Figure 19. Implementing Wake-on-Approach Using Ganged Sensor



.

Figure 20. Sensor Ganging in CapSense Component



13 Example Projects

This application note contains three example projects that demonstrate how to:

- Tune proximity sensors for maximum proximity-sensing distance
- Tune proximity sensors for liquid tolerance
- Detect gestures using proximity sensors

The following are required for testing the example projects:

- CY8CKIT-024 CapSense Proximity Shield Kit: The CY8CKIT-024 is a shield that has proximity sensors to evaluate the proximity-sensing performance of Cypress's PSoC CapSense devices. The CY8CKIT-024 has Arduino™-compatible headers and can be connected to CY8CKIT-040 PSoC 4000 Pioneer Development Kit and CY8CKIT-042 PSoC 4 Pioneer Kit.
- The CY8CKIT-040 kit (with the PSoC 4000 part) or the CY8CKIT-042 kit (with the PSoC 4200 part): These kits connect to the CY8CKIT-024 shield.
- PSoC Creator software installed on your PC: The PSoC Creator IDE is required to view and edit these example projects. You can download the latest version of PSoC Creator from the PSoC Creator web page.
- PSoC Programmer and the BCP: The PSoC Programmer software is used to program the PSoC devices on the CY8CKIT-040 kit and CY8CKIT-042 kit with the hex files. The BCP software is used to view the CapSense sensor data such as raw count, baseline, and difference count. The BCP software is installed along with PSoC Programmer. You can download PSoC Programmer at www.cypress.com/go/psocprogrammer.

Two sets of example projects are provided to operate on CY8CKIT-024 with the PSoC Pioneer Kits CY8CKIT-040 and CY8CKIT-042. The example projects are available at AN92239 on Cypress's website.

To open an example project in PSoC Creator, navigate to the folder where you have extracted the example project and double-click on the <*Project.cywrk*>.

13.1 Example Project 1 – Proximity Distance

The Proximity_Distance_040 project is designed to operate on CY8CKIT-040 with CY8CKIT-024, while the Proximity_Distance_042 project is designed to operate on CY8CKIT-042 with CY8CKIT-024. These two projects demonstrate how to tune the PROX proximity sensor (square loop with a diagonal length of 10.3 cm) on CY8CKIT-024 to achieve a proximity-sensing distance of 10 cm. The example projects show how to set optimum CapSense CSD parameters for a proximity sensor and apply an Advanced Low-Pass Filter to achieve a proximity-sensing distance of 10 cm.

The proximity of the user is indicated by driving the LEDs (LED1–LED5) with a PWM signal. The LEDs glow at minimum brightness when the hand is at a 10-cm proximity-sensing distance, and the brightness increases as the hand approaches the sensor.

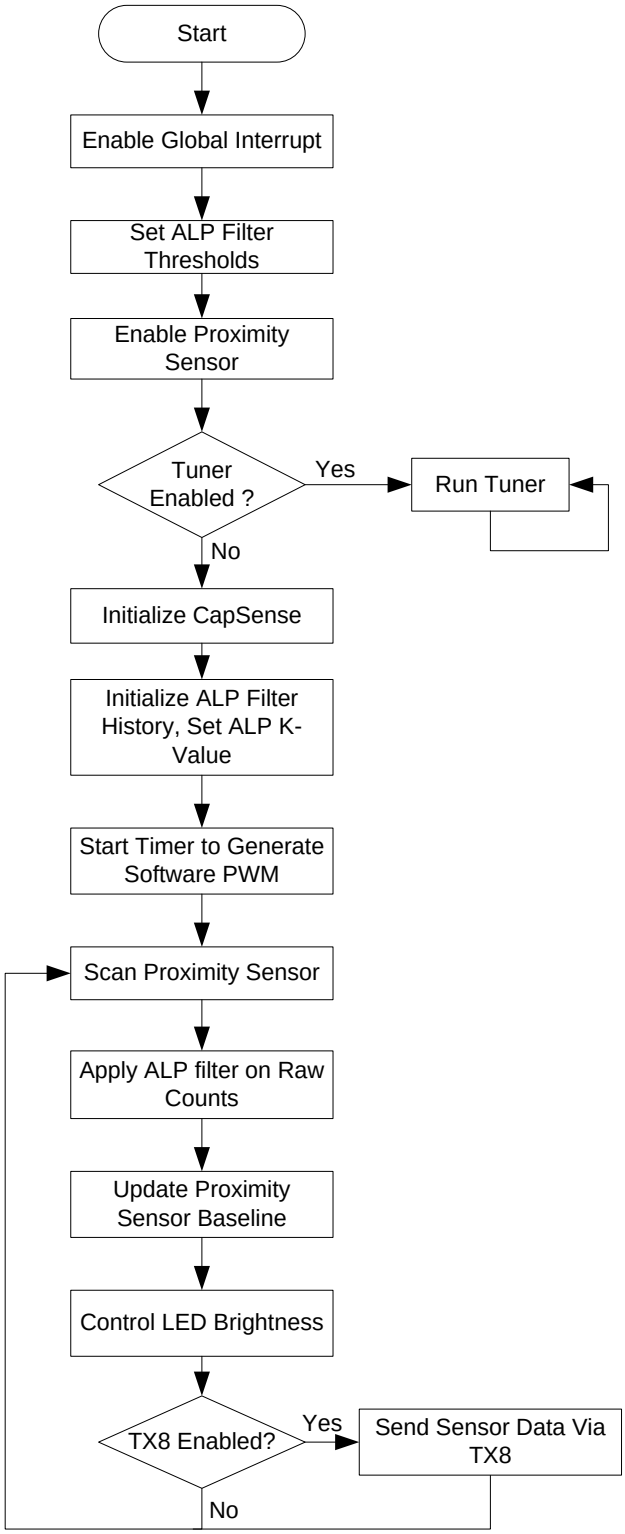
13.1.1 Project Details

The behavior of the project is described in a flow chart in Figure 21. The procedure explained in the Tuning CapSense CSD Parameters for a Large Proximity-Sensing Distance section was followed to tune the CapSense CSD parameters for the PROX sensor.

The Proximity_Distance_04x projects use only the PROX sensor. The PS1, PS2, PS3, and PS4 sensors are not scanned in the firmware and are always connected to ground to reduce noise in the proximity sensor raw count. The GND/SHIELD loop is connected to the shield pin of the PSoC 4 device. As the Eliminate the effect of floating or grounded conductive objects list item explains, driving a nearby conductive object (GND/SHIELD loop) with the driven shield signal eliminates the effect of these conductive objects on the proximity-sensing distance and provides a large proximity-sensing distance.

The following steps show how the CapSense CSD parameters were tuned to achieve a 10-cm proximity-sensing distance with CY8CKIT-040.

Figure 21. Proximity Distance Project Flow Chart



1. In the CapSense CSD Component, the tuning method is selected as SmartSense to determine the sense clock divider and modulator clock divider value for the PROX sensor.
2. Shield is enabled in the **Advanced** tab to drive the, hatch pattern in the bottom layer and the GND/SHIELD loop on CY8CKIT-024 with the driven shield signal. The driven shield signal reduces the effect of floating conductive objects on the PROX sensor proximity-sensing distance and helps in achieving a large proximity-sensing distance.
3. Tuner is enabled to read the sense clock divider and modulator clock divider values. The sense clock divider is set to '2', and the modulator clock divider is set to '1'.
4. In the CapSense Component **General** configuration window, the parameters are set to the values shown Table 3.
5. In the **Widgets Config** section, the resolution for the PROX sensor is set to 16 bits. All other settings retain their default values.
6. TX8 communication (Software Transmit UART) is established with the PSoC 4000 device on CY8CKIT-040 to view the sensor data such as the raw count, baseline, and difference count to measure the SNR. Refer to Appendix B: Reading Sensor Debug Data in the Bridge Control Panel for details on how to view sensor debug data via UART in the BCP tool.
7. Using this configuration, the SNR for the proximity sensor PROX is measured. The noise is measured to be approximately 100 counts. The signal at a 10-cm proximity-sensing distance is measured to be approximately 220. Therefore, the SNR = 220/100 = 2.2. Because the SNR is less than 5:1, the ALP filter is enabled to attenuate the noise and increase the SNR.
8. The ALP filter parameters are tuned as explained in the ALP Filter Tuning section. With the ALP filter, the noise is measured to be approximately 10 counts and the SNR at a 10-cm distance is 220/10 =22:1.

   To achieve an SNR of 5:1 with a noise count of 10, the signal should be 50. Therefore, the threshold parameters are set to the values listed in Table 5, assuming a signal of 50. With a finger-threshold value of 40, a proximity-sensing distance of 12 cm is achieved.

13.1.2 Hardware Configuration

To test the project with CY8CKIT-040, remove jumper J14 and connect the kit to CY8CKIT-024, as Figure 22 shows. When the two kits are connected, the J1, J2, J3, and J4 headers on CY8CKIT-040 connect to the J1, J2, J3, and J4 headers on CY8CKIT-024 respectively.

Similarly, to test the project with CY8CKIT-042, connect the kit to CY8CKIT-024, as Figure 23 shows.

When the two kits are connected, the J1, J2, J3, and J4 headers on CY8CKIT-042 connect to the J1, J2, J3, and J4 headers on CY8CKIT-024 respectively. If you want to read the proximity sensor data via UART, connect a jumper wire between pin P0[1] on the J2 header and P12[6] on the J8 header using a wire, as Figure 24 shows.

On CY8CKIT-024, slide SW1 to select SHIELD to drive the GND/SHIELD loop and the bottom hatch pattern with the driven shield signal. Table 7 lists the pin connections for the example projects.

Figure 22. Hardware Connection to Test the Proximity Distance Project with CY8CKIT-040

Figure 23. Hardware Connection to Test the Proximity Distance Project with CY8CKIT-042

Figure 24. Connection for CY8CKIT-042 to Read Data Via UART

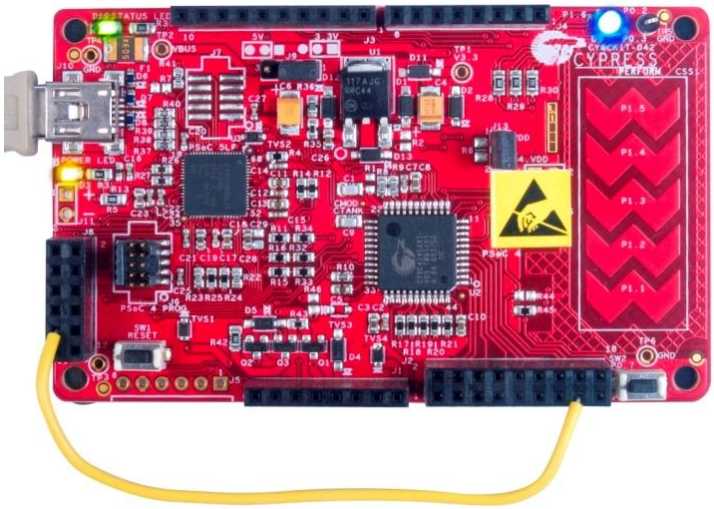


Table 7. Pin Connections for Proximity_Distance_040 Project and Proximity_Distance_042 Project

Pin Name in Project	Pin Name on CY8CKIT-024	Pin Number	
		CY8CKIT-040	CY8CKIT-042
PROX_0_PROX	PROX	P0[3]	P0[0]
PS1_BTN	PS1	P1[7]	P0[6]
PS2_BTN	PS2	P0[5]	P0[4]
PS3_BTN	PS3	P0[1]	P2[1]
PS4_BTN	PS4	P1[0]	P1[0]
Cmod	–	P0[4]	P4[2]
Shield	SHIELD	P0[6]	P0[5]
Cshield_tank	–	P0[2]	P4[3]
LED1	LED1	P1[5]	P3[6]
LED2	LED2	P1[4]	P2[6]
LED3	LED3	P0[7]	P0[7]
LED4	LED4	P2[0]	P2[7]
LED5	LED5	P0[0]	P2[0]
EzI2C:SCL	–	P1[2]	P3[0]
EzI2C:SDA	–	P1[3]	P3[1]
Tx8	–	P3[0]	P0[1]

13.1.3 Testing the Proximity_Distance Project

To test the project on CY8CKIT-040, program the PSoC 4000 device with the *Proximity_Distance_040.hex* file. Similarly, to test the project on CY8CKIT-042, program the PSoC 4200 device with the *Proximity_Distance_042.hex* file.

For details on programming the PSoC 4000 and PSoC 4200 devices, refer to the CY8CKIT-040 User Guide and CY8CKIT-042 User Guide respectively.

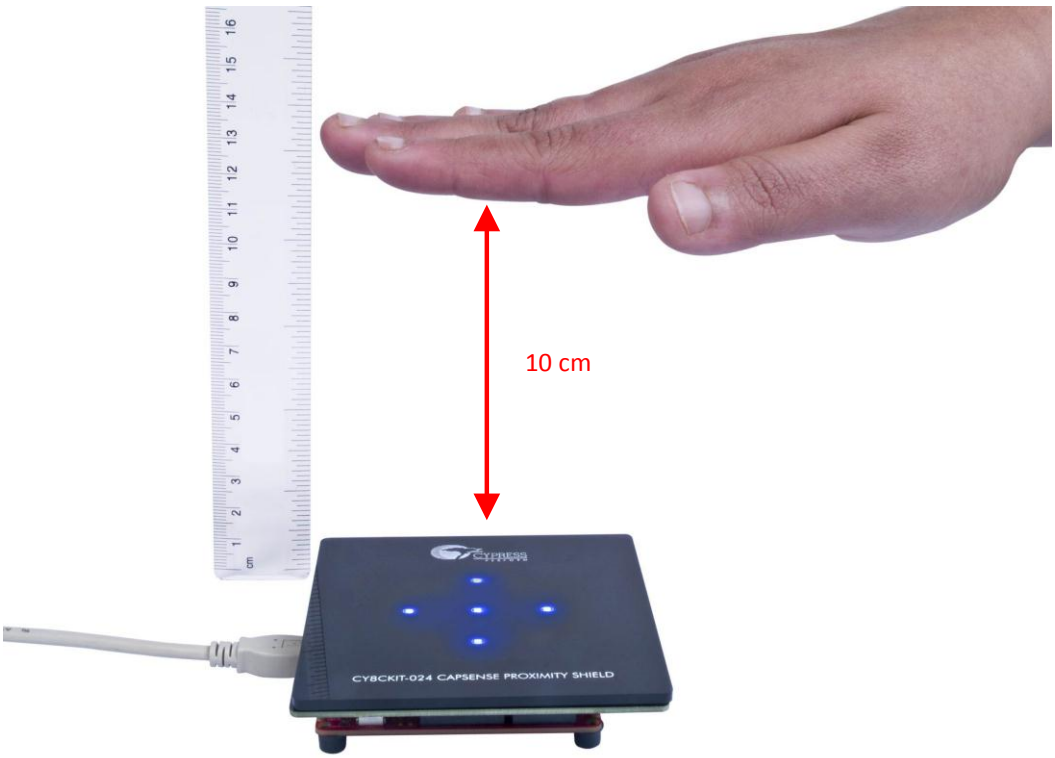
Note the following:

- Connect CY8CKIT-024 to CY8CKIT-040/ CY8CKIT-042 before programming.
- Press the reset switch, SW1, on CY8CKIT-040/CY8CKIT-042 whenever you change the slide switch, SW1, position on CY8CKIT-024 or when you place a ruler near the kit to measure the proximity-sensing distance, as Figure 25 shows.

To measure the proximity-sensing distance, hover your hand over the kit at approximately 10 cm, as Figure 25 shows. The LEDs (LED1–LED5) glow at their minimum brightness, showing that proximity is detected. Bring your hand towards the kit and verify that the LED brightness increases, indicating closer proximity of the hand with respect to the proximity sensor.

Note When UART (software TX8) communication is enabled in the project for reading the CapSense sensor data, the LEDs flicker. This is because the software TX8 disables the global interrupt during data transmission. When the global interrupt is disabled, the timer ISR will not be serviced and the PWM frequency varies, resulting in LED flickering.

Figure 25. Measuring Proximity-Sensing Distance



13.2 Example Project 2 – Liquid-Tolerant Proximity

The Liquid_Tolerant_Proximity_040 project is designed to operate on CY8CKIT-040 with CY8CKIT-024, and the Liquid_Tolerant_Proximity_042 *project* is designed to operate on CY8CKIT-042 with CY8CKIT-024. These two projects demonstrate how to tune the proximity sensor, PROX, on CY8CKIT-024 for liquid tolerance. The project is tuned for a proximity-sensing distance of 3 cm with liquid tolerance. The proximity of the user hand is indicated by turning on the LEDs (LED1–LED5).

13.2.1 Project Details

The Liquid_Tolerant_Proximity_04x projects use only the PROX sensor. PS1, PS2, PS3, and PS4 are not scanned in the firmware and are always connected to the driven shield signal along with the GND/SHIELD loop. As explained in the Eliminate the effect of floating or grounded conductive objects list item, driving nearby sensors or PCB traces (GND/SHIELD loop, hatch pattern in the bottom layer) with the driven shield signal helps in reducing the signal (i.e. shift in raw count) due to liquid droplets.

The proximity sensor, PROX, is tuned by following the steps mentioned in the Tuning for Liquid Tolerance section. The following steps show how the sensor was tuned for liquid tolerance with CY8CKIT-040.

1. The resolution of the PROX sensor is set to 16 bits, and a water droplet is poured over the PROX sensor. The signal due to the water droplet is noted.
2. A hand is brought towards the proximity sensor, and it is found that at a 3-cm distance from the sensor, the signal due to the hand is three times the signal due to the water droplet. Therefore, for the present sensor layout, the maximum proximity-sensing distance that can be achieved with water tolerance is 3 cm.
3. To optimize the scan time and power consumption, the resolution is decreased by one bit and the SNR is measured at 3 cm. It is found that with a resolution of 13 bits, a proximity-sensing distance of 3 cm can be achieved with an SNR greater than 5:1.

13.2.2 Hardware Configuration

To test the project, follow the steps explained in the Hardware Configuration section in Page 28. Table 8 lists the pin mapping for Liquid_Tolerant_04x projects.

Table 8. Pin Connections for Liquid_Tolerant_Proximity_040 and Liquid_Tolerant_Proximity_042 Projects

Pin Name in Project	Pin Name on CY8CKIT-024	Pin Number	
		CY8CKIT-040	CY8CKIT-042
PROX_0_PROX	PROX	P0[3]	P0[0]
PS1_BTN	PS1	P1[7]	P0[6]
PS2_BTN	PS2	P0[5]	P0[4]
PS3_BTN	PS3	P0[1]	P2[1]
PS4_BTN	PS4	P1[0]	P1[0]
Cmod	–	P0[4]	P4[2]
Shield	SHIELD	P0[6]	P0[5]
Cshield_tank	–	P0[2]	P4[3]
LED1	LED1	P1[5]	P3[6]
LED2	LED2	P1[4]	P2[6]
LED3	LED3	P0[7]	P0[7]
LED4	LED4	P2[0]	P2[7]
LED5	LED5	P0[0]	P2[0]
EzI2C:SCL	–	P1[2]	P3[0]
EzI2C:SDA	–	P1[3]	P3[1]
Tx8	–	P3[0]	P0[1]

13.2.3 Testing the Liquid_Tolerant_Proximity Project

To test the project on CY8CKIT-040, program the PSoC 4000 device with the *Liquid_Tolerant_Proximity_040.hex* file. Similarly, to test the project on CY8CKIT-042, program the PSoC 4200 device with the *Liquid_Tolerant_Proximity_042.hex* file.

For details on programming the PSoC 4000 and PSoC 4200 devices, refer to the CY8CKIT-040 User Guide and CY8CKIT-042 User Guide respectively.

Note the following:

- Connect CY8CKIT-024 to CY8CKIT-040/CY8CKIT-042 before programming.
- Press the reset switch, SW1, on CY8CKIT-040/CY8CKIT-042 whenever you change the slide switch, SW1, position on CY8CKIT-024 or when you place a ruler near the kit to measure the proximity-sensing distance, as Figure 25 shows.

  To place the water droplets on the proximity sensor (as Figure 26 shows), use the liquid dropper provided with CY8CKIT-024. After placing the water droplet, verify that the LEDs (LED1–LED5) do not turn on when water droplets are present on the sensor. Hover your hand over the kit at a distance of 3 cm, as Figure 27 shows, and verify that the LEDs (LED1–LED5) are turned ON to indicate proximity detection.

  Note: When placing the water droplet, if the hand is within proximity-sensing distance (3 cm), proximity will be detected and the LEDs will be turned ON. The LEDs will be turned off when you remove your hand after placing the water droplet, indicating that the water droplet is not causing false triggers.

Figure 26. Placing Water Droplet over Proximity Sensor

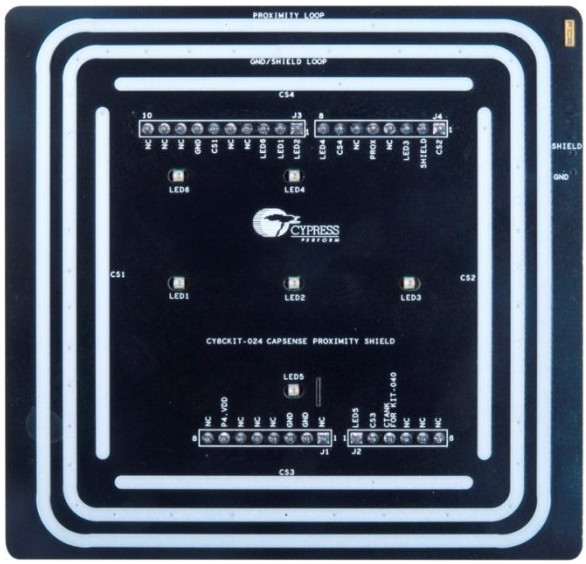
Figure 27. Proximity Detection in Presence of Water Droplet

13.3 Example Project 3 – Proximity Gestures

The project Proximity_Gestures_040 is designed to operate on CY8CKIT-040 with CY8CKIT-024, and the project Proximity_Gestures_042 project is designed to operate on CY8CKIT-042 with CY8CKIT-024. The projects demonstrate how to detect swipe gestures in the X-axis and Y-axis using the proximity sensors on CY8CKIT-024.

The Proximity_Gestures_04x projects use only the PS1, PS2, PS3, and PS4 proximity sensors, as Figure 28 shows. The PROX sensor is not scanned in the firmware and is always connected to the driven shield along with the GND/SHIELD loop. A nearby floating or grounded conductive object reduces the proximity-sensing distance of the active proximity sensors. As explained in the Eliminate the effect of floating or grounded conductive objects list item, driving nearby sensors and PCB traces with the driven shield signal reduces the effect of floating or grounded conductive objects on the proximity-sensing distance of the active sensors. The proximity sensors PS1, PS2, PS3, and PS4 on CY8CKIT-024 are used to detect swipe gestures in the X-axis and Y-axis. PS1 and PS2 are used for detecting swipe gestures in the X-axis, while PS3 and PS4 are used for detecting swipe gestures in the Y-axis.

Figure 28. Sensor Placement in CY8CKIT-024



The projects contain two macros, GESTURE_AXIS and LED_DRIVE_SEQUENCE, in the *gestures.h* file. These macros are used to select the type of gesture to be detected and the LED drive sequence respectively.

- **GESTURE_AXIS:** This macro determines the type of gesture detected by the device. Set this macro to "XAXIS" to detect swipe gestures in the X-axis, and set it to "YAXIS" to detect swipe gestures in the Y-axis. By default, the macro is set to "XAXIS" to detect swipe gestures in the X-axis.
- **LED_DRIVE_SEQUENCE:** This macro determines how the LEDs are driven when a gesture is detected. Set this macro to "LED_DRIVE_DURING_GESTURE" to drive the LEDs based on the current position of the hand, and set it to "LED_DRIVE_AFTER_GESTURE" to drive the LEDs after a gesture is detected. By default, the macro is set to "LED_DRIVE_DURING_GESTURE."

  When the macro is set to "LED_DRIVE_DURING_GESTURE," the LEDs (LED1–LED5) are driven as listed in Table 9. When the macro is set to "LED_DRIVE_AFTER_GESTURE," the LEDs (LED1–LED5) are driven as listed in Table 10.

Table 9. LED Turn-ON Sequence for LED_DRIVE_DURING_GESTURE

Gesture	Gesture Direction	LED Activated
Horizontal Swipe	Left	LED1
	Middle	LED2
	Right	LED3
Vertical Swipe	Bottom	LED5
	Middle	LED2
	Top	LED4

Table 10. LED Turn-ON Sequence for LED_DRIVE_AFTER_GESTURE

Gesture	Gesture Direction	LED Sequence
Horizontal Swipe	Left to right	LED1→ LED2→ LED3
	Right to left	LED3→ LED2→ LED1
Vertical	Top to bottom	LED4→ LED2→ LED5

Swipe	Bottom to top	LED5→ LED2→ LED4

13.3.1 Project Details

The behavior of the project is described in the flow chart in Figure 29. In Proximity_Gestures_04x projects, the status of sensors PS1 and PS2 is compared to determine the swipe gesture in the X-axis, and the status of sensors PS3 and PS4 is compared to determine the swipe gesture in the Y-axis. The sensor status is tracked with respect to the time to determine the type of gesture performed. When the gesture is detected, the LEDs are turned ON in a sequence based on the gesture type and LED drive type, as listed in Table 9 and Table 10.

13.3.2 Hardware Configuration

To test the project, follow the steps explained in the Hardware Configuration section. Table 11 lists the connections for Proximity_Gestures_04x projects.

Figure 29. Proximity Gestures Project Flow Chart

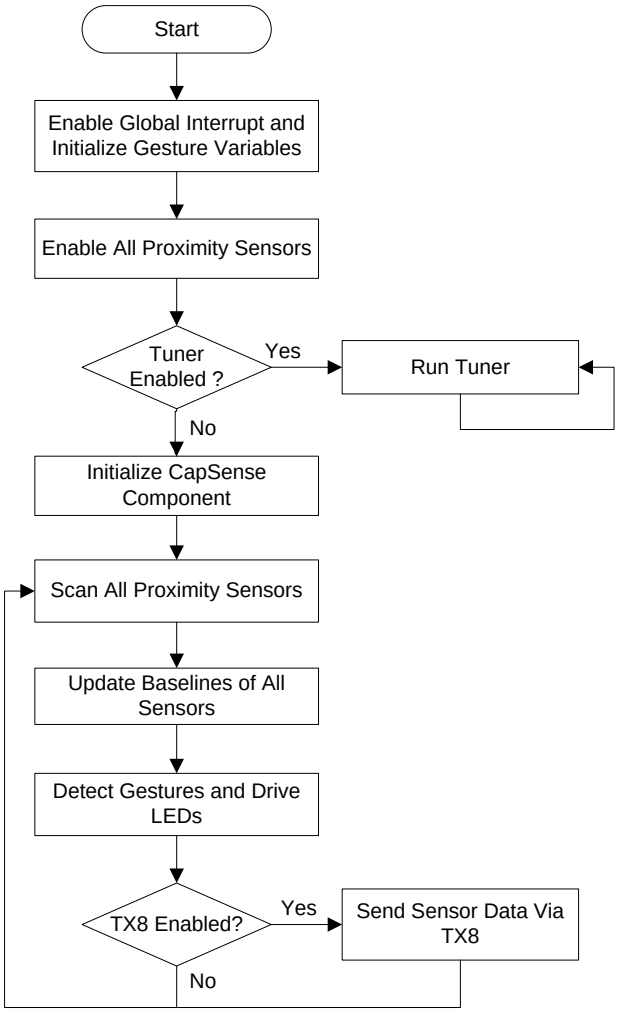


Table 11. Pin Connections for Proximity_Gestures_040 and Proximity_Gestures_042 Projects

Pin Name in Project	Pin Name on CY8CKIT-024	Pin Number	
		CY8CKIT-040	CY8CKIT-042
PS1_0_PROX	PS1	P1[7]	P0[6]
PS2_0_PROX	PS2	P0[5]	P0[4]
PS3_0_PROX	PS3	P0[1]	P2[1]
PS4_0_PROX	PS4	P1[0]	P1[0]
PROX_0_PROX	PROX	P0[3]	P0[0]
Cmod	–	P0[4]	P4[2]
Shield	SHIELD	P0[6]	P0[5]
Cshield_tank	–	P0[2]	P4[3]
LED1	LED1	P1[5]	P3[6]
LED2	LED2	P1[4]	P2[6]
LED3	LED3	P0[7]	P0[7]
LED4	LED4	P2[0]	P2[7]
LED5	LED5	P0[0]	P2[0]
EzI2C:SCL	–	P1[2]	P3[0]
EzI2C:SDA	–	P1[3]	P3[1]
Tx8	–	P3[0]	P0[1]

13.3.3 Testing the Proximity_Gestures Project

To test the project on CY8CKIT-040, program the PSoC 4000 device with the *Proximity_Gestures_040.hex* file. Similarly, to test the project on CY8CKIT-042, program the PSoC 4200 device with the *Proximity_Gestures_042.hex* file.

For details on programming the PSoC 4000 and PSoC 4200 devices, refer to the CY8CKIT-040 User Guide and CY8CKIT-042 User Guide respectively.

Note the following:

- Connect CY8CKIT-024 to CY8CKIT-040/CY8CKIT-042 before programming.
- Press the reset switch, SW1, on CY8CKIT-040/CY8CKIT-042 whenever you change the slide switch, SW1, position on CY8CKIT-024.

To verify the swipe gesture performed in the X-axis, hover your hand over the kit at a distance of 2 cm and move the hand from left to right, as Figure 30 shows, or from right to left. Observe that the LEDs turn on in the sequence listed in Table 9.

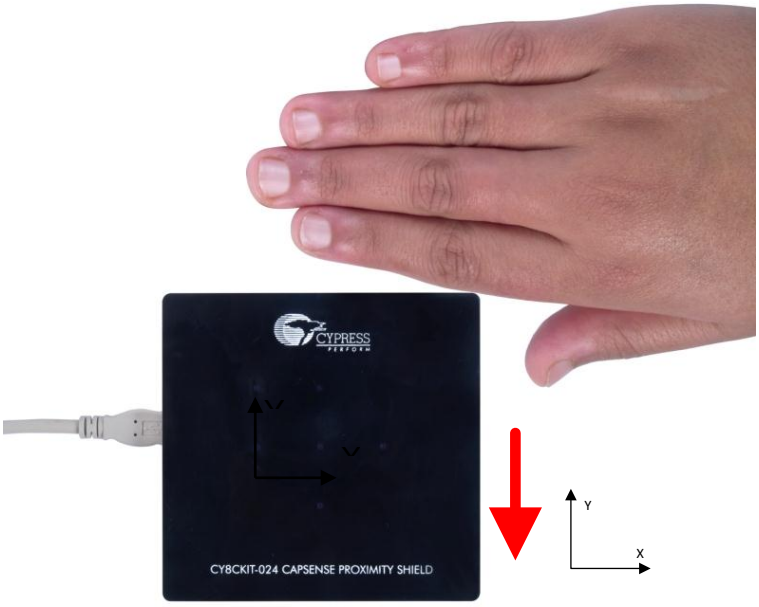
Similarly, to verify the swipe gestures performed in the Y-axis, set the macro GESTURE_AXIS to “YAXIS” and program the device. Hover your hand over the kit at a distance of 2 cm and move it from top to bottom, as Figure 31 shows, or from bottom to top. Observe that the LEDs turn on in the sequence listed in Table 9.

To drive the LEDs after the gesture is completed, set the macro LED_DRIVE_SEQUENCE to “LED_DRIVE_AFTER_GESTURE” and repeat this procedure using Table 10 as a guide.

Note: When the macro LED_DRIVE_SEQUENCE is set to “LED_DRIVE_AFTER_GESTURE,” the LEDs will be turned ON only when the gesture is completed, that is, when the hand has completely moved away from the kit.

Figure 30. Left-to-Right Swipe Gesture

Figure 31. Top-to-Bottom Swipe Gesture



14 Glossary

AMUXBUS

Analog multiplexer bus available inside PSoC that helps to connect I/O pins with multiple internal analog signals.

SmartSense™ Auto-Tuning

A CapSense algorithm that automatically sets sensing parameters for optimal performance after the design phase and continuously compensates for system, manufacturing, and environmental changes.

Baseline

A value resulting from a firmware algorithm that estimates a trend in the Raw Count when there is no human finger present on the sensor. The Baseline is less sensitive to sudden changes in the Raw Count and provides a reference point for computing the Difference Count.

Button or Button Widget

A widget with an associated sensor that can report the active or inactive state (that is, only two states) of the sensor. For example, it can detect the touch or no-touch state of a finger on the sensor.

Difference Count

The difference between Raw Count and Baseline. If the difference is negative, or if it is below Noise Threshold, the Difference Count is always set to zero.

Capacitive Sensor

A conductor and substrate, such as a copper button on a printed circuit board (PCB), which reacts to a touch or an approaching object with a change in capacitance.

CapSense®

Cypress's touch-sensing user interface solution. The industry's No. 1 solution in sales by 4x over No. 2.

CapSense Mechanical Button Replacement (MBR)

Cypress's configurable solution to upgrade mechanical buttons to capacitive buttons, requires minimal engineering effort to configure the sensor parameters and does not require firmware development. These devices include the CY8CMBR3XXX and CY8CMBR2XXX families.

Centroid or Centroid Position

A number indicating the finger position on a slider within the range given by the Slider Resolution. This number is calculated by the CapSense centroid calculation algorithm.

Compensation IDAC

A programmable constant current source, which is used by CSD to compensate for excess sensor C_P. This IDAC is not controlled by the Sigma-Delta Modulator in the CSD block unlike the Modulation IDAC.

CSD

CapSense Sigma Delta (CSD) is a Cypress-patented method of performing self-capacitance (also called self-cap) measurements for capacitive sensing applications.

In CSD mode, the sensing system measures the self-capacitance of an electrode, and a change in the self-capacitance is detected to identify the presence or absence of a finger.

Debounce

A parameter that defines the number of consecutive scan samples for which the touch should be present for it to become valid. This parameter helps to reject spurious touch signals.

A finger touch is reported only if the Difference Count is greater than Finger Threshold + Hysteresis for a consecutive Debounce number of scan samples.

Driven-Shield

A technique used by CSD for enabling liquid tolerance in which the Shield Electrode is driven by a signal that is equal to the sensor switching signal in phase and amplitude.

Electrode

A conductive material such as a pad or a layer on PCB, ITO, or FPCB. The electrode is connected to a port pin on a CapSense device and is used as a CapSense sensor or to drive specific signals associated with CapSense functionality.

Finger Threshold

A parameter used with Hysteresis to determine the state of the sensor. Sensor state is reported ON if the Difference Count is higher than Finger Threshold + Hysteresis, and it is reported OFF if the Difference Count is below Finger Threshold – Hysteresis.

Ganged Sensors

The method of connecting multiple sensors together and scanning them as a single sensor. Used for increasing the sensor area for proximity sensing and to reduce power consumption.

To reduce power when the system is in low-power mode, all the sensors can be ganged together and scanned as a single sensor taking less time instead of scanning all the sensors individually. When the user touches any of the sensors, the system can transition into active mode where it scans all the sensors individually to detect which sensor is activated.

PSoC supports sensor-ganging in firmware, that is, multiple sensors can be connected simultaneously to AMUXBUS for scanning.

Gesture

Gesture is an action, such as swiping and pinch-zoom, performed by the user. CapSense has a gesture detection feature that identifies the different gestures based on predefined touch patterns. In the CapSense component, the Gesture feature is supported only by the Touchpad Widget.

Guard Sensor

Copper trace that surrounds all the sensors on the PCB, similar to a button sensor and is used to detect a liquid stream. When the Guard Sensor is triggered, firmware can disable scanning of all other sensors to prevent false touches.

Hatch Fill or Hatch Ground or Hatched Ground

While designing a PCB for capacitive sensing, a grounded copper plane should be placed surrounding the sensors for good noise immunity. But a solid ground increases the parasitic capacitance of the sensor which is not desired. Therefore, the ground should be filled in a special hatch pattern. A hatch pattern has closely-placed, crisscrossed lines looking like a mesh and the line width and the spacing between two lines determine the fill percentage. In case of liquid tolerance, this hatch fill referred as a shield electrode is driven with a shield signal instead of ground.

Hysteresis

A parameter used to prevent the sensor status output from random toggling due to system noise, used in conjunction with the Finger Threshold to determine the sensor state. See Finger Threshold.

IDAC (Current-Output Digital-to-Analog Converter)

Programmable constant current source available inside PSoC, used for CapSense and ADC operations.

Liquid Tolerance

The ability of a capacitive sensing system to work reliably in the presence of liquid droplets, streaming liquids or mist.

Linear Slider

A widget consisting of more than one sensor arranged in a specific linear fashion to detect the physical position (in single axis) of a finger.

Low Baseline Reset

A parameter that represents the maximum number of scan samples where the Raw Count is abnormally below the Negative Noise Threshold. If the Low Baseline Reset value is exceeded, the Baseline is reset to the current Raw Count.

Manual-Tuning

The manual process of setting (or tuning) the CapSense parameters.

Matrix Buttons

A widget consisting of more than two sensors arranged in a matrix fashion, used to detect the presence or absence of a human finger (a touch) on the intersections of vertically and horizontally arranged sensors.

If M is the number of sensors on the horizontal axis and N is the number of sensors on the vertical axis, the Matrix Buttons Widget can monitor a total of M x N intersections using ONLY M + N port pins.

When using the CSD sensing method (self-capacitance), this Widget can detect a valid touch on only one intersection position at a time.

Modulation Capacitor (CMOD)

An external capacitor required for the operation of a CSD block in Self-Capacitance sensing mode.

Modulator Clock

A clock source that is used to sample the modulator output from a CSD block during a sensor scan. This clock is also fed to the Raw Count counter. The scan time (excluding pre and post processing times) is given by
$(2^N - 1)$/Modulator Clock Frequency, where N is the Scan Resolution.

Modulation IDAC

Modulation IDAC is a programmable constant current source, whose output is controlled (ON/OFF) by the sigma-delta modulator output in a CSD block to maintain the AMUXBUS voltage at V_{REF}. The average current supplied by this IDAC is equal to the average current drawn out by the sensor capacitor.

Mutual-Capacitance

Capacitance associated with an electrode (say TX) with respect to another electrode (say RX) is known as mutual capacitance.

Negative Noise Threshold

A threshold used to differentiate usual noise from the spurious signals appearing in negative direction. This parameter is used in conjunction with the Low Baseline Reset parameter.

Baseline is updated to track the change in the Raw Count as long as the Raw Count stays within Negative Noise Threshold, that is, the difference between Baseline and Raw count (Baseline – Raw count) is less than Negative Noise Threshold.

Scenarios that may trigger such spurious signals in a negative direction include: a finger on the sensor on power-up, removal of a metal object placed near the sensor, removing a liquid-tolerant CapSense-enabled product from the water; and other sudden environmental changes.

Noise (CapSense Noise)

The variation in the Raw Count when a sensor is in the OFF state (no touch), measured as peak-to-peak counts.

Noise Threshold

A parameter used to differentiate signal from noise for a sensor. If Raw Count – Baseline is greater than Noise Threshold, it indicates a likely valid signal. If the difference is less than Noise Threshold, Raw Count contains nothing but noise.

Overlay

A non-conductive material, such as plastic and glass, which covers the capacitive sensors and acts as a touch-surface. The PCB with the sensors is directly placed under the overlay or is connected through springs. The casing for a product often becomes the overlay.

Parasitic Capacitance (C_P)

Parasitic capacitance is the intrinsic capacitance of the sensor electrode contributed by PCB trace, sensor pad, vias, and air gap. It is unwanted because it reduces the sensitivity of CSD.

Proximity Sensor

A sensor that can detect the presence of nearby objects without any physical contact.

Radial Slider

A widget consisting of more than one sensor arranged in a specific circular fashion to detect the physical position of a finger.

Raw Count

The unprocessed digital count output of the CapSense hardware block that represents the physical capacitance of the sensor.

Refresh Interval

The time between two consecutive scans of a sensor.

Scan Resolution

Resolution (in bits) of the Raw Count produced by the CSD block.

Scan Time

Time taken for completing the scan of a sensor.

Self-Capacitance

The capacitance associated with an electrode with respect to circuit ground.

Sensitivity

The change in Raw Count corresponding to the change in sensor capacitance, expressed in counts/pF. Sensitivity of a sensor is dependent on the board layout, overlay properties, sensing method, and tuning parameters.

Sense Clock

A clock source used to implement a switched-capacitor front-end for the CSD sensing method.

Sensor

See Capacitive Sensor.

Sensor Auto Reset

A setting to prevent a sensor from reporting false touch status indefinitely due to system failure, or when a metal object is continuously present near the sensor.

When Sensor Auto Reset is enabled, the Baseline is always updated even if the Difference Count is greater than the Noise Threshold. This prevents the sensor from reporting the ON status for an indefinite period of time. When Sensor Auto Reset is disabled, the Baseline is updated only when the Difference Count is less than the Noise Threshold.

Sensor Ganging

See Ganged Sensors.

Shield Electrode

Copper fill around sensors to prevent false touches due to the presence of water or other liquids. Shield Electrode is driven by the shield signal output from the CSD block. See Driven-Shield.

Shield Tank Capacitor (C_{SH})

An optional external capacitor (C_{SH} Tank Capacitor) used to enhance the drive capability of the CSD shield, when there is a large shield layer with high parasitic capacitance.

Signal (CapSense Signal)

Difference Count is also called Signal. See Difference Count.

Signal-to-Noise Ratio (SNR)

The ratio of the sensor signal, when touched, to the noise signal of an untouched sensor.

Slider Resolution

A parameter indicating the total number of finger positions to be resolved on a slider.

Touchpad

A Widget consisting of multiple sensors arranged in a specific horizontal and vertical fashion to detect the X and Y position of a touch.

Trackpad

See Touchpad.

Tuning

The process of finding the optimum values for various hardware and software or threshold parameters required for CapSense operation.

V_{REF}

Programmable reference voltage block available inside PSoC used for CapSense and ADC operation.

Widget

A user-interface element in the CapSense component that consists of one sensor or a group of similar sensors. Button, proximity sensor, linear slider, radial slider, matrix buttons, and touchpad are the supported widgets.

15 Summary

This application note explained how to design a proximity sensor and tune the CapSense CSD parameters for a large proximity-sensing distance and liquid tolerance. It also provided details on how to implement gesture detection based on proximity sensing and the wake-on-approach feature.

16 Related Application Notes

AN79953 – Getting Started with PSoC 4

AN86233 – PSoC 4 Low-Power Modes and Power Reduction Techniques

Getting Started with CapSense

PSoC 4 CapSense Design Guide

About the Author

Name: Chethan

Title: Applications Engineer

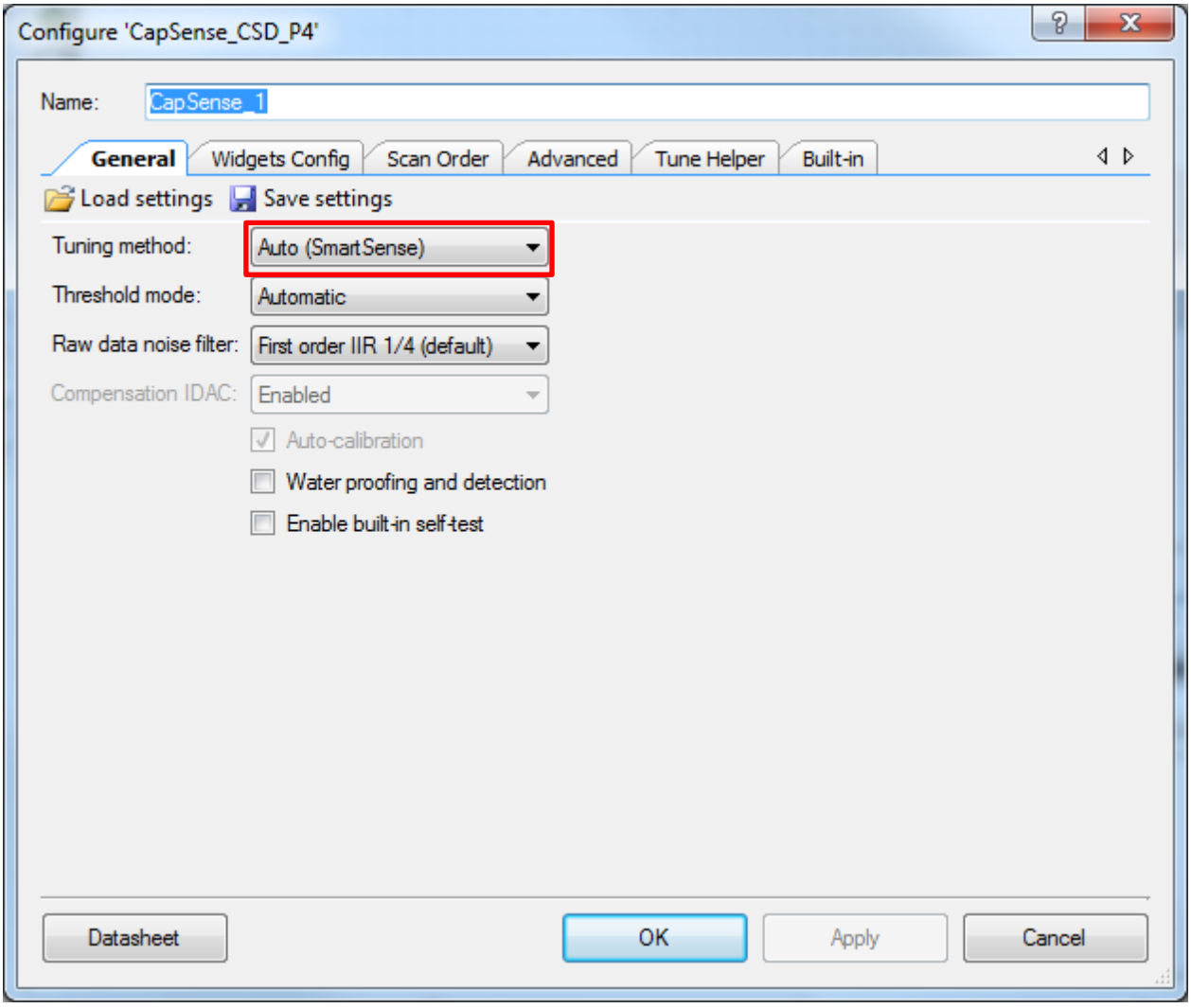
Background: Chethan earned his BE in Electronics and Communication Engineering from BMS College of Engineering. He is currently working on CapSense based applications and assisting customers with their designs.

A Appendix A: Determining the Sense Clock Divider and Modulator Clock Divider Using SmartSense

For proper CapSense operation, you need to set the sense clock divider and modulator clock divider in the CapSense CSD Component configuration window. The values for these parameters depend on the C_P of the sensor. To determine the sense clock divider and modulator clock divider values for a sensor using the SmartSense algorithm, follow these steps.

1. In the PSoC Creator *TopDesign.cysch* window, place the CapSense CSD Component (if not placed already). Double-click on the CapSense Component to configure the CapSense CSD parameters.
2. In the CapSense Component **General** tab, set the parameters to the values shown in Figure 32.

Figure 32. General Settings Tab in CapSense Component Configuration Window



3. In the CapSense Component **Widgets Config** tab, select the “Proximity sensors” widget and click **Add proximity sensor**. Add more proximity sensors as required in your design.
4. In the CapSense Component **Advanced** configuration windows, do the following:
   a. If the shield electrode is used in your design, select the “Enabled” option for the **Shield** parameter. If not, then select “Disabled.”
   b. Set the **Csh_tank precharge** parameter to “Precharge by IO buffer.”

Note: This only needs to be done if using a shield electrode.

c. Leave all other parameters at their default values, as shown in Figure 33.

5. In the CapSense Component **Tune Helper** configuration tab, do the following:
   a. Select the **Enable tune helper** check box.
   b. For the **Instance name for the SCB component** parameter, enter “EzI2C.”
   c. Click **OK** to save the settings and close the window.
6. In the *TopDesign.cysch* panel, place an EzI2C Slave Component and do the following:
   a. Double-click the EzI2C Component and set the parameters to the values shown in Figure 34.
   b. Click **OK** to save the settings and close the window.

Figure 33. Advanced Settings Tab in CapSense Component Configuration Window

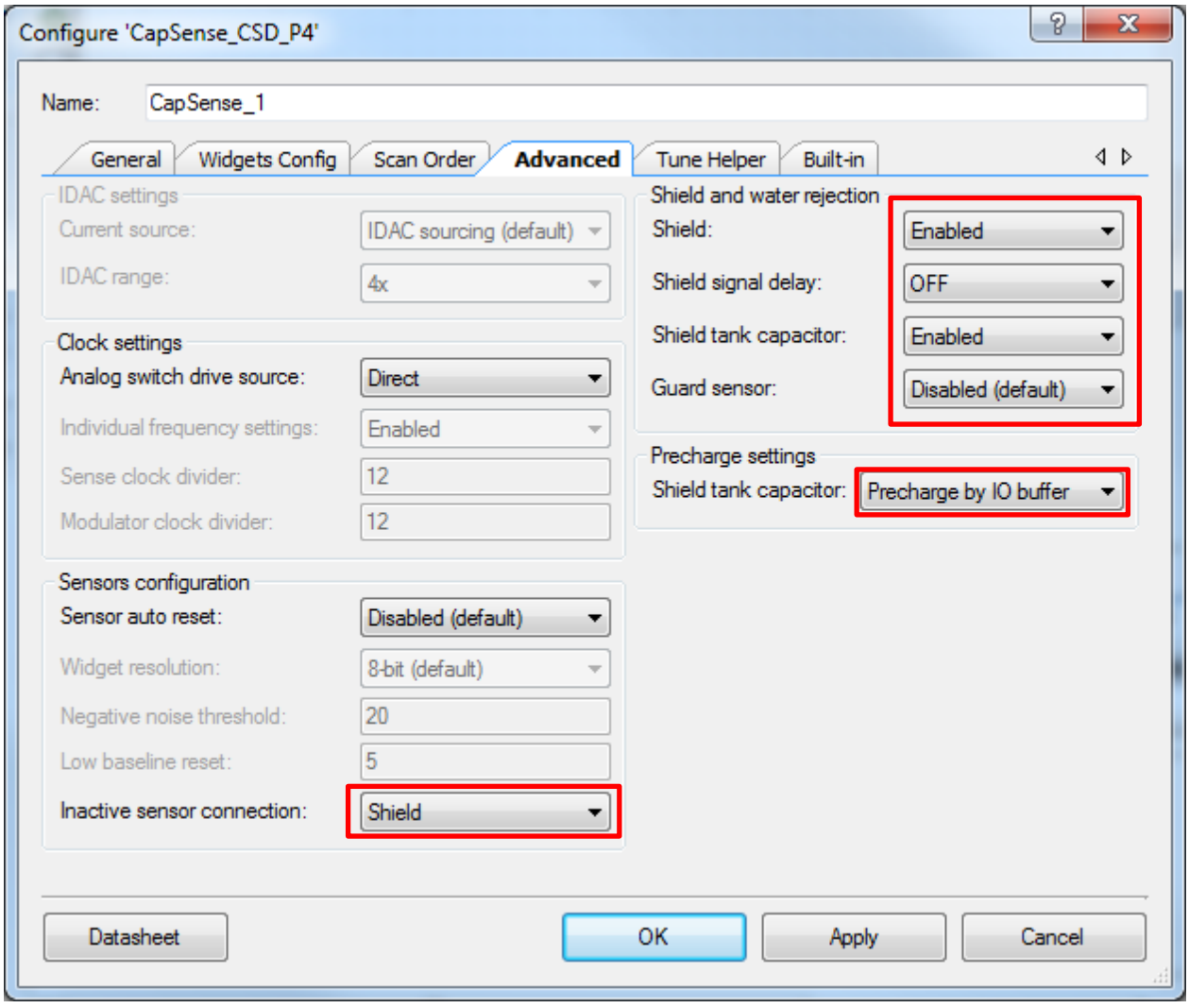
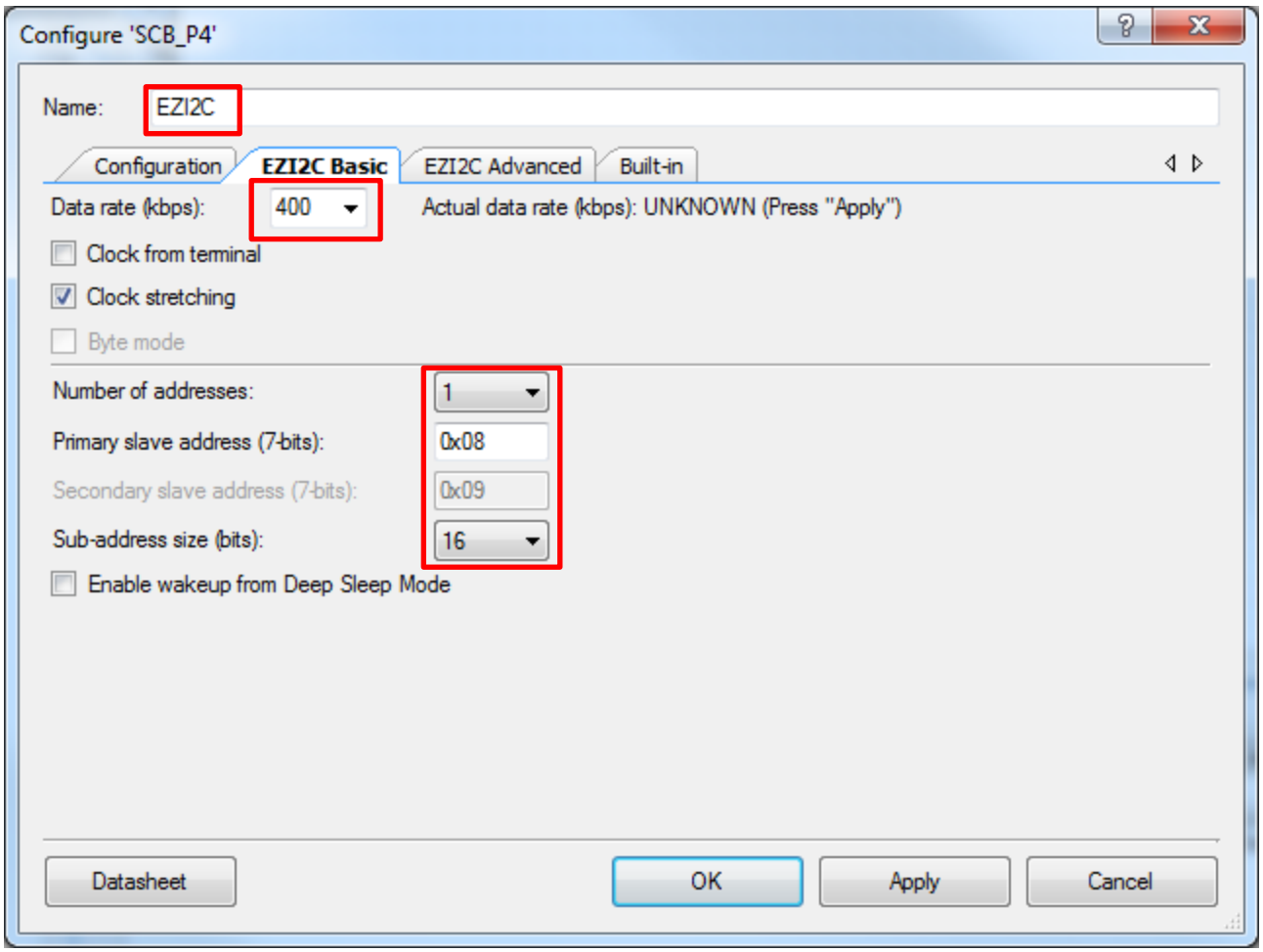


Figure 34. EzI2C Configuration Window



7. In the *main.c* file, do the following:

   a. Enable global interrupt.

   b. Enable proximity sensors.

   c. Start the CapSense Tuner.

   d. Run the tuner in an infinite loop as follows.

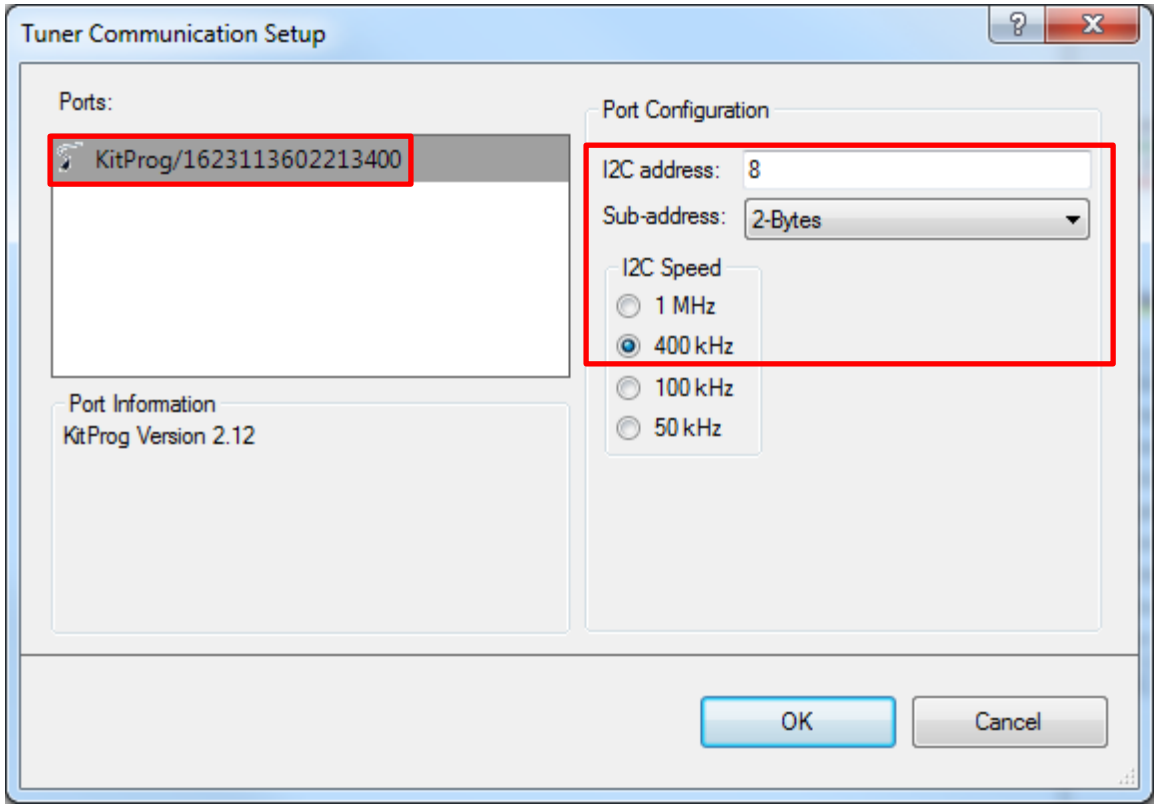
   In the CapSense Component, proximity sensors are not enabled by default. You should call the CapSense_EnableWidget (uint32 widget) API for each proximity sensor with the proximity sensor name as the parameter to enable it as follows.

```
int main()
{
    CyGlobalIntEnable;
    CapSense_1_EnableWidget (CapSense_1_PROXIMITYSENSOR0__PROX);
    CapSense_1_EnableWidget (CapSense_1_PROXIMITYSENSOR1__PROX);
    CapSense_1_EnableWidget (CapSense_1_PROXIMITYSENSOR2__PROX);
```

```
    CapSense_1_EnableWidget (CapSense_1_PROXIMITYSENSOR3__PROX);
    CapSense_1_TunerStart();
    while (1)
    {
        CapSense_1_TunerComm();
    }
}
```

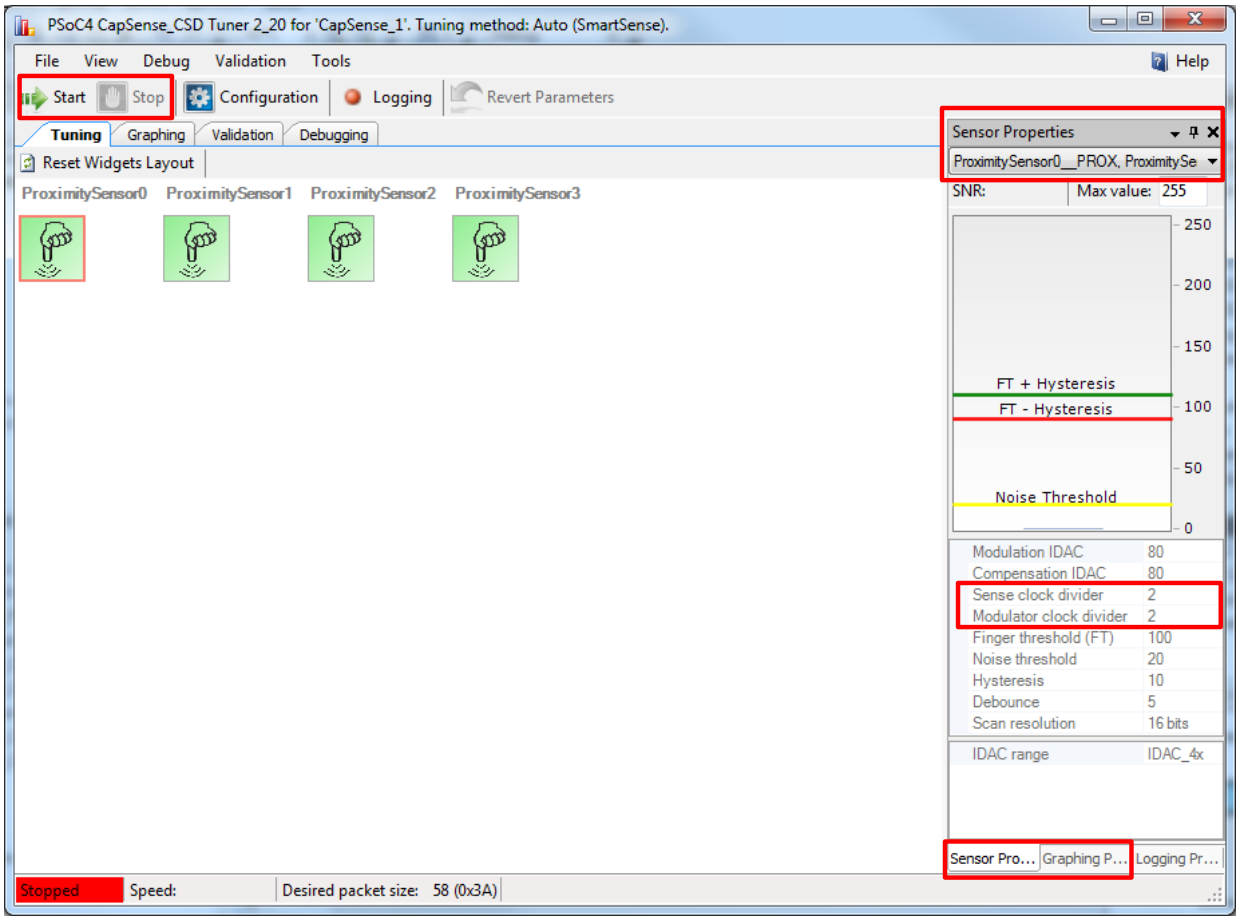
8. In the *.cydwr* file, select the sensor and EzI2C pins and build the project (press **[Shift] [F6]**). Ensure that there are zero errors in the project.
9. After the project is built without errors, press **[Ctrl] [F5]** to program the device.
10. After the device is programmed, in the *TopDesign.cysch* window, right-click on the "CapSense CSD" Component and select "Launch Tuner."
11. In the CapSense Tuner, click on "Configuration" and set the parameters to the values shown in Figure 35 and click **OK**.

Figure 35. CapSense Tuner Configuration



12. In the CapSense Tuner, click the **Start** button to read the CapSense CSD parameters set by the SmartSense algorithm. The sensor parameters are shown in the bottom right of the **Sensor Properties** window. If you have more than one sensor, select the sensor for which you want to view the sensor parameters at the top right part of the **Sensor Properties** window, as Figure 36 shows.
13. Choose **File** > **Apply Changes and Close**.

Figure 36. Reading Sensor Parameters Using Tuner



B Appendix B: Reading Sensor Debug Data in the Bridge Control Panel

To tune the CapSense CSD parameters for a sensor, you need to view the sensor data such as the raw count, baseline, and difference count. For viewing the sensor data, you can use the CapSense Tuner available in the CapSense Component or use the Bridge Control Panel (BCP) tool.

Using the BCP tool, you can view other parameters in addition to the raw count, baseline, and difference count. The CapSense Tuner requires I^2C communication for viewing the sensor data, whereas the BCP tool supports both I^2C and UART communication for viewing the sensor data.

The example projects provided with this application note support viewing the sensor data in the CapSense Tuner (using I^2C) and the BCP (using UART). This appendix explains how to view the sensor data in the BCP tool through UART communication. For details on how to use the CapSense Tuner, refer to the "Manual Tuning" section in the PSoC 4 CapSense Design Guide.

Table 12 lists the general structure of a UART TX packet sent to the BCP tool. The TX packet consists of a header (2 bytes), data (of variable length), and a tail (3 bytes).

Table 12. UART TX Data Packet Structure

Header		Data	Tail		
0x0D	0x0A	Variable-length data	0x00	0xFF	0xFF

Note: To learn the exact data for any example project, refer to the *.IIC* file provided with it.

Follow these steps to set up the BCP tool for viewing the data:

1. Connect CY8CKIT-040 or CY8CKIT-042 to the PC using the USB cable (USB A to mini-B). If you are using CY8CKIT-042, connect pin P0[1] on J2 and pin P12[6] on J8 using a wire, as Figure 24 shows. Ensure that the TX8_ENABLE macro is set to '1' in the *main.c* file of the project and build the project.
2. Press [**Ctrl] [F5]** to program the CY8CKIT-040 kit or CY8CKIT-042 kit.
3. Open the BCP tool (choose **Start** > **All Programs** > **Cypress** > **Bridge Control Panel <version>** > **Bridge Control Panel <version>**).
4. In the BCP tool, select the CY8CKIT-040 kit or CY8CKIT-042 kit UART-bridge COM port in the "Connected I2C /SPI/RX8 Ports:" window and set RX8 as its protocol, as Figure 37 shows. The CY8CKIT-040 kit or CY8CKIT-042 kit UART-bridge COM port will be listed in the Device Manager, as Figure 38 shows.

Figure 37. Bridge Control Panel – COM Port and Protocol Selection

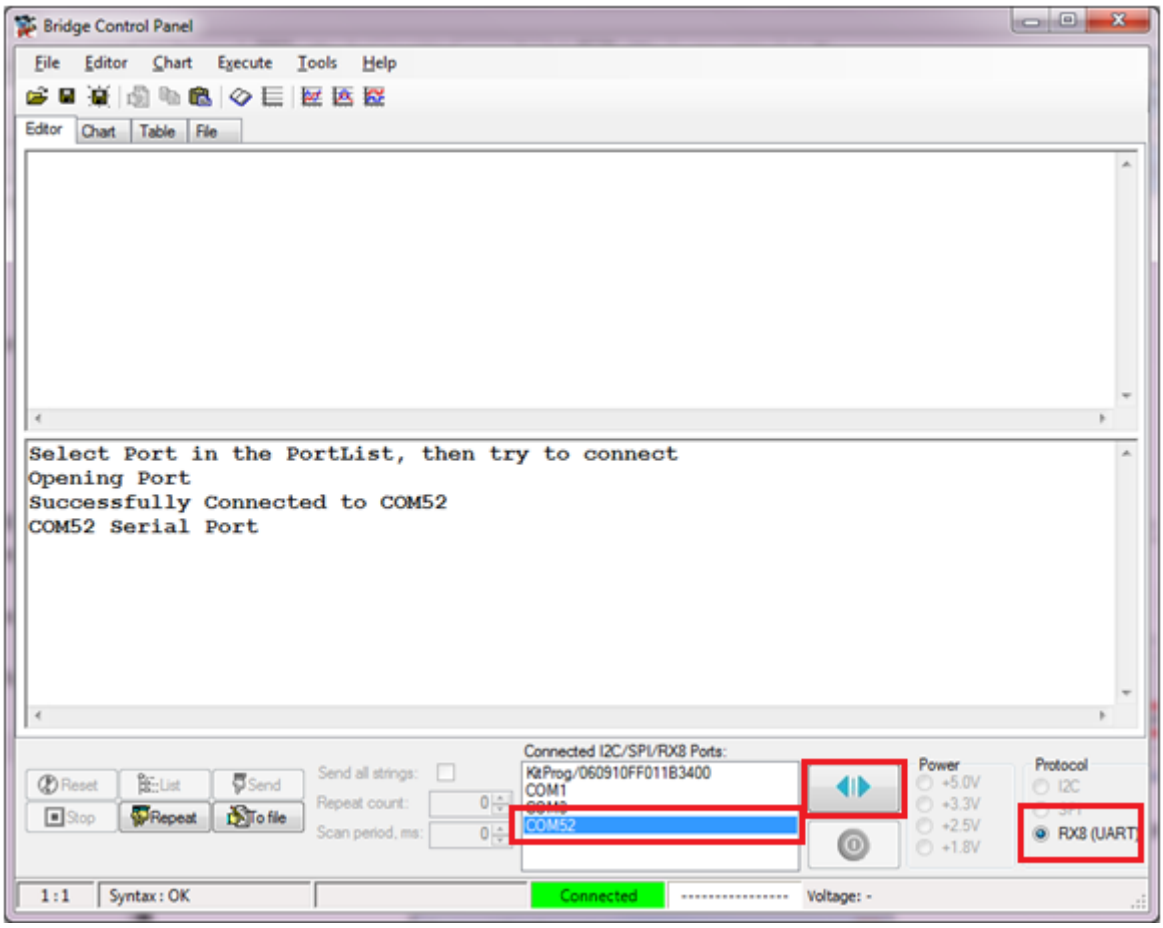
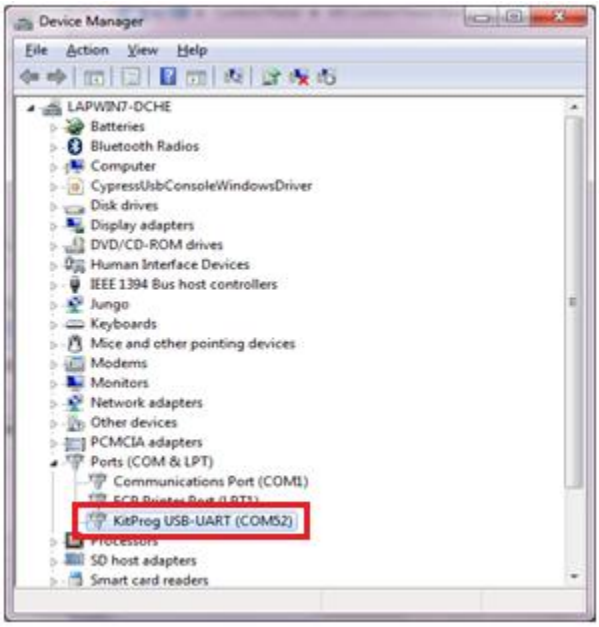
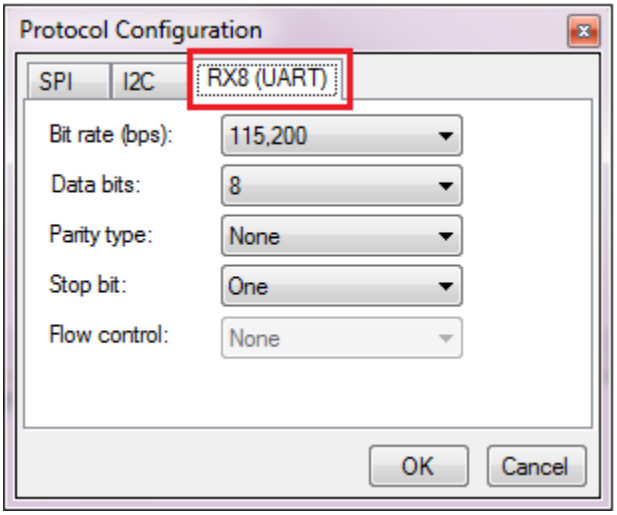


Figure 38. KitProg COM Port in Device Manager



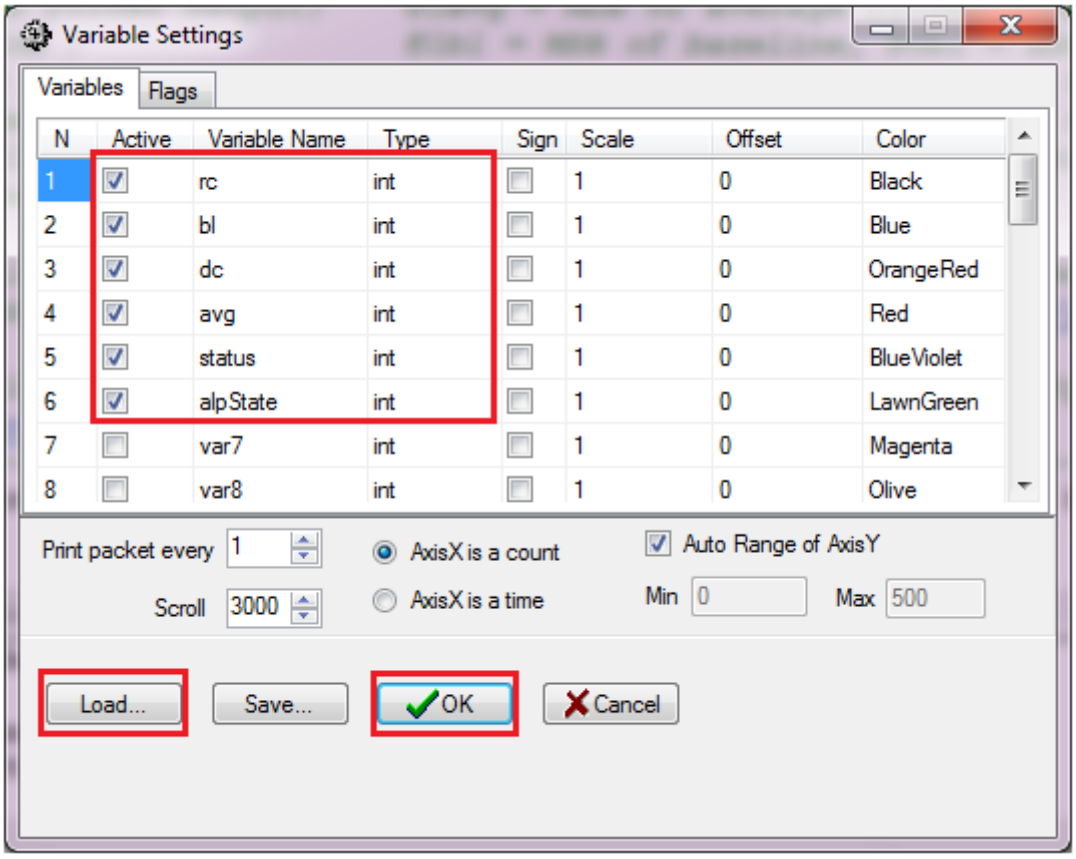
5. Choose **Tools** > **Protocol Configuration** or press **[F7]** and configure the RX8 protocol parameters, as Figure 39 shows.

Figure 39. RX8 Protocol Configuration



6. Choose **Chart** > **Variable Settings** > **Load** and navigate to the example project directory. Select the *<Project_Name>.ini* file and click **Open**. Click **OK** to apply the settings and close the window, as shown in Figure 40.

Figure 40. Bridge Control Panel – Variable Settings



7. Choose **File** > **Open File** and select the <*Project_Name*>.*iic* file provided with the project and click **Open**.
8. Place the cursor on the command line and then click **Repeat,** as Figure 41, shows to start receiving the packets.
9. To view the sensor data in a graphical format, click on the **Chart** tab. Uncheck the **Select All** option and check the **rc** and **bl** options to view the raw count and baseline, as shown in Figure 42. Similarly, you can view the difference count (**dc**) and the average filtered data.

Figure 41. Reading CapSense Debug Data in Bridge Control Panel

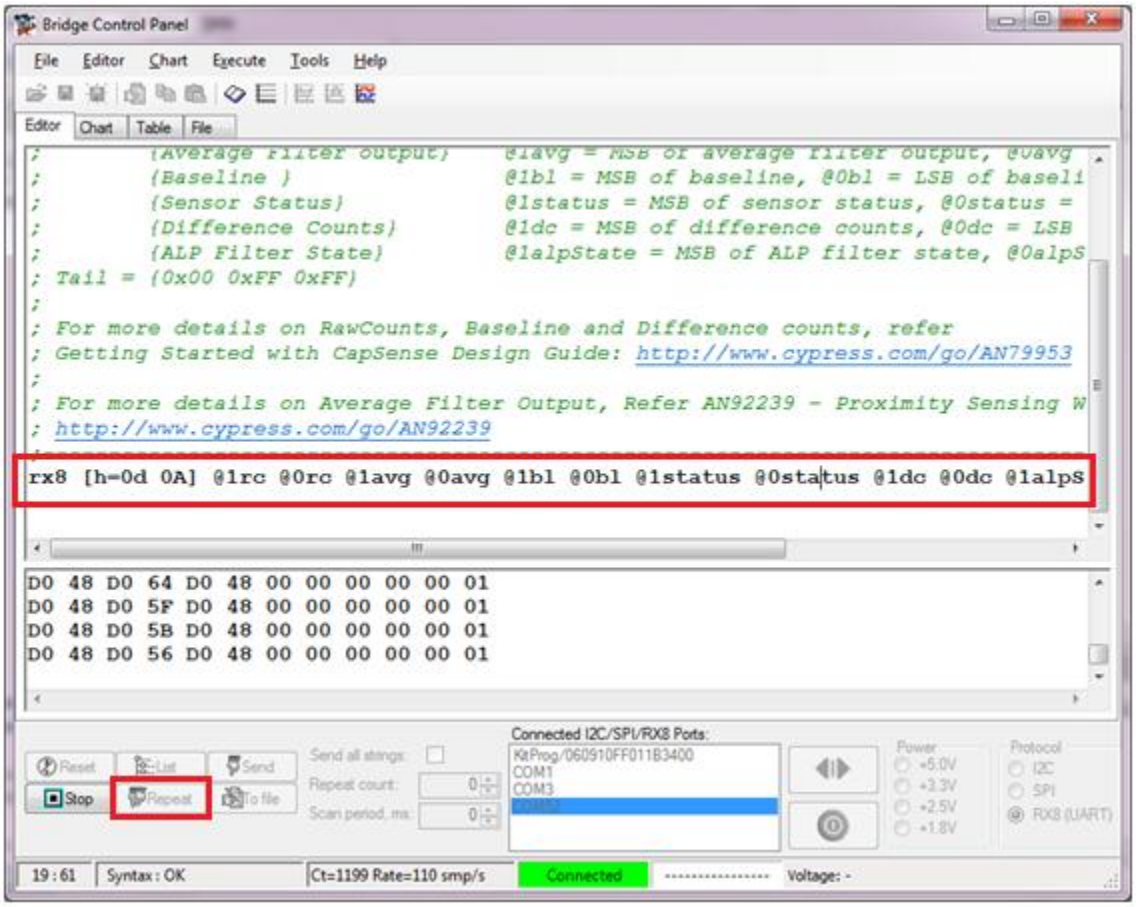
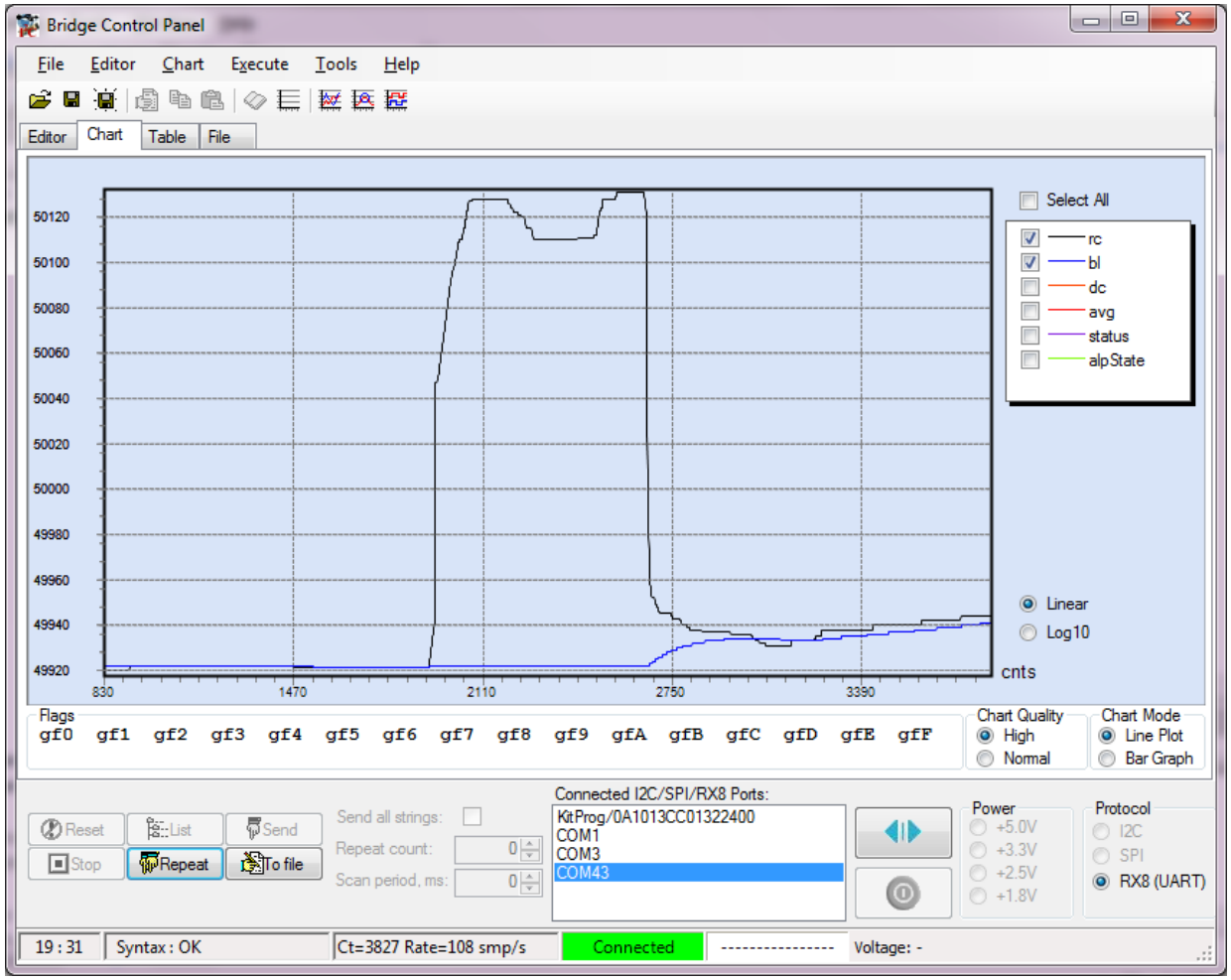


Figure 42. Viewing CapSense Debug Data in Graphical Format in Bridge Control Panel



C Appendix C: Advanced Low-Pass Filter API Definitions

Table 13 lists the ALP filter APIs. Example Project 1 – Proximity Distance shows how to use these APIs.

Table 13. Advanced Low-Pass Filter Definitions

1	**CapSenseFilters_SetAdvancedLowPassK**	
	Description	This API sets the filter coefficient (K-value) for the ALP filter.
	Prototype	`void CapSenseFilters_SetAdvancedLowPassK(CAPSENSEFILTERS_IIR_K_ENUM k)`

	Parameters	Following is the Enum type definition used by this API:
		`typedef enum`
		`{`
		`CAPSENSEFILTERS_IIR_K_2 = 0x01,`
		`CAPSENSEFILTERS_IIR_K_4 = 0x02,`
		`CAPSENSEFILTERS_IIR_K_16 = 0x04,`
		`CAPSENSEFILTERS_IIR_K_32 = 0x05,`
		`CAPSENSEFILTERS_IIR_K_64 = 0x06`
		`} CAPSENSEFILTERS_IIR_K_ENUM;`
		
		The K-value of the ALP filter determines the attenuation of noise in the proximity sensor raw counts.
		
		Noise attenuation decreases in the order CAPSENSEFILTERS_IIR_K_64 > CAPSENSEFILTERS_IIR_K_32 > CAPSENSEFILTERS_IIR_K_16. The CAPSENSEFILTERS_IIR_K_4 and CAPSENSEFILTERS_IIR_K_2 are used for compatibility purposes and should not be used. Refer to the ALP Filter Tuning section to set this parameter.

	Return value	None
2	**CapSenseFilters_InitializeAdvancedLowPass**	
	Description	This API initializes the internal history of the specified sensor to the specified raw count value.
	Prototype	`void CapSenseFilters_InitializeAdvancedLowPass(uint32 sensorId, uint16 initializationRawValue)`
	Parameters	`sensorId`: This parameter indicates the sensor number for which the ALP filter history is to be initialized to a given `initializationRawValue`. `initializationRawValue`: This is the value to which the filter history gets initialized. This value should be equal to the sensor raw count.
	Return value	None

3	**CapSenseFilters_RunAdvancedLowPass**	
	Description	This API applies the ALP filter to the raw count of the proximity sensor. Filtered data is stored back in the raw count array.
	Prototype	`void CapSenseFilters_RunAdvancedLowPass(uint32 sensorId)`
	Parameters	`sensorId`: This parameter indicates the sensor number on which the filter is to be applied.
	Return value	None
4	**CapSenseFilters_ResetAdvancedLowPass**	
	Description	This API resets the ALP filter for a given `sensorId`. Resetting the ALP filter ensures that the filter history is automatically reinitialized the next time the "CapSenseFilters_RunAdvancedLowPass()" API is called.
	Prototype	`void CapSenseFilters_ResetAdvancedLowPass(uint32 sensorId)`
	Parameters	`sensorId`: This parameter indicates the sensor number for which the filter is to be reset.
	Return value	None
5	**CapSenseFilters_GetAverageData**	
	Description	This API returns the output of the average filter for the specified sensor (`sensorId`).
	Prototype	`uint16 CapSenseFilters_GetAverageData(uint32 sensorId)`
	Parameters	`sensorId`: This parameter indicates the sensor number for which the `averageVal` is to be read.
	Return value	`averageVal[sensorId]`

6	**CapSenseFilters_GetALPFilterState**	
	Description	This API returns the current state of the ALP filter.
	Prototype	`CAPSENSEFILTERS_ALP_STATE_ENUM CapSenseFilters_GetALPFilterState (uint32 sensorId)`
	Parameters	`sensorId`: This parameter indicates the sensor number for which the ALP filter state is to be read.

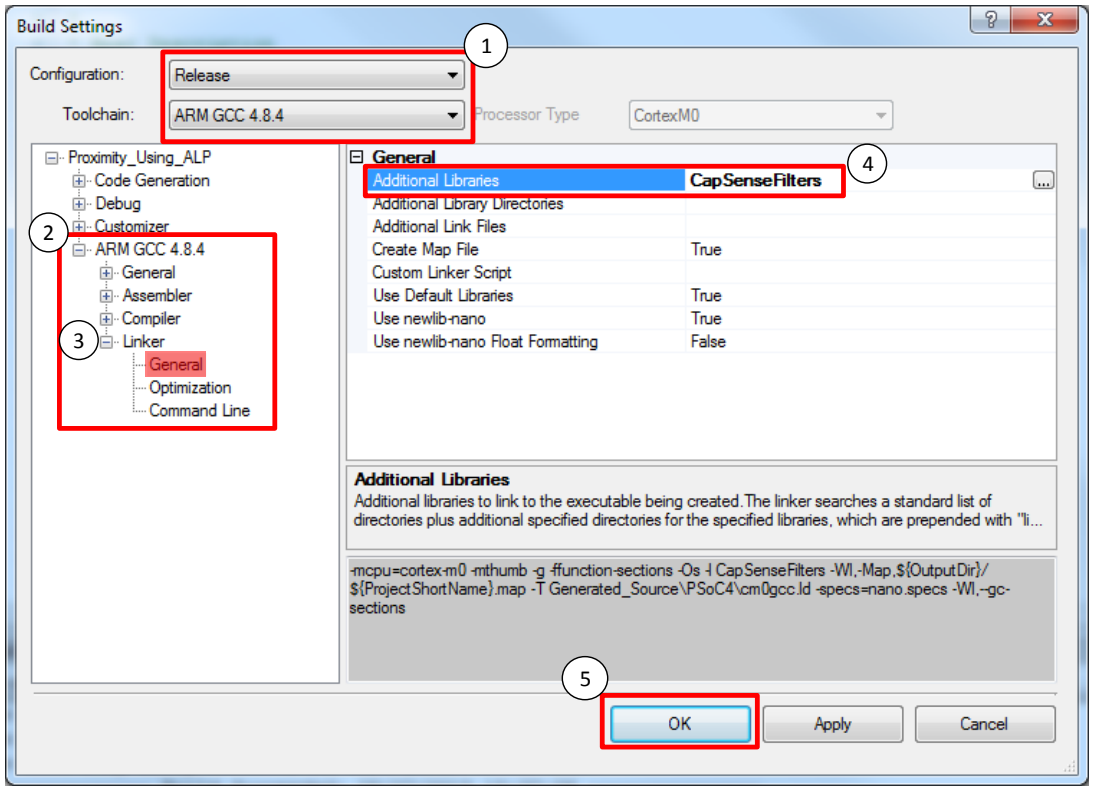
	Return value	`stateAdvancedFilter [sensorId]`
		
		Following is the Enum type definition used by this API:
		`typedef enum`
		`{`
		`CAPSENSEFILTERS_ALP_STATE_UNINITIALIZED = 0x00,`
		`CAPSENSEFILTERS_ALP_STATE_IDLE = 0x01,`
		`CAPSENSEFILTERS_ALP_STATE_POS_EDGE_TRACK = 0x02,`
		`CAPSENSEFILTERS_ALP_STATE_SIGNAL_DETECTED = 0x03,`
		`CAPSENSEFILTERS_ALP_STATE_NEG_EDGE_TRACK = 0x04,`
		`CAPSENSEFILTERS_ALP_STATE_NEG_SPIKE = 0x05`
		`} CAPSENSEFILTERS_ALP_STATE_ENUM;`
		
		UNINITIALIZED state indicates that the filter history variables are not initialized.
		IDLE state indicates the normal operation of a sensor when there is no target object near the sensor. A slow response filter is applied in this state.
		POS_EDGE_TRACK state indicates that there is a target object approaching the sensor. A fast response filter is applied in this state.
		SIGNAL_DETECTED state indicates that the target object is detected. A slow response filter is applied in this state.
		NEG_EDGE_TRACK state indicates that the target object is receding from the sensor. A fast response filter is applied in this state.
		NEG_SPIKE state indicates an unexpected negative noise on the sensor raw count. This state is not expected to occur during normal sensor operation.

D Appendix D: Adding the ALP Filter Library to Any CapSense Project

If you want to add an ALP filter to your project, follow the below steps. This section assumes that you already have a PSoC Creator project with proximity sensor implemented.

1. Download the ALP_Filter_Library.zip file from http://www.cypress.com/AN92239.
2. Extract the files from **ALP_Filter_Library.zip** and open the folder ALP_Filter_Library.
3. The library file is provided for three compilers. Depending on the compiler used in your project, select the *libCapSenseFilters.a* file from the specific compiler folder and copy it to the location where *main.c* is located.
   For example, if you are using **ARM GCC 4.8.4** complier, open **ALP_Filter_Library** > **ARM_GCC_473** and select the *libCapSenseFilters.a* file and copy it to the project location where *main.c* is located.
4. Copy header file *CapSenseFilter.h* and *CyBitOperations.h* to project location where *main.c* is located.
5. Open your project in **PSoC Creator**.
6. Click on **Project** menu and select **Build Settings**.
7. In the Build Settings window, specify the **Configuration** option as **Release** or **Debug** based on your requirement and follow the steps mentioned below**.**
8. Select the required complier from the **Toolchain** option. For example, if you want to use **ARM GCC 4.8.4** then select **ARM GCC 4.8.4.** The next steps assume that the compiler is selected as **ARM GCC 4.8.4**. If you are using a different compiler, you can follow the same steps.
9. Expand the **ARM GCC 4.8.4** list in the left-most column as shown in #2 in Figure 43, and click on **Linker** as shown in #3 in Figure 43. In the **General** tab of **Linker** menu, click on **Additional Libraries,** as shown in #4 in Figure 43 and type *CapSenseFilters* then click **OK** as shown in #5 in Figure 43.
   Note: If you want to use both **Release** and **Debug** configuration, you need to repeat step 9 for both the configurations.

Figure 43. Adding ALP Filter Library Files to PSoC Creator



10. Add the library header files to your project by following the below steps.

    Navigate to **Workspace Explorer** and right-click on **Header Files** and select **Add > Existing Item**.

    Navigate to <**your project location**>-> <**project name.cydsn**> >. In the explorer, press **Ctrl** and select *CapSenseFilters.h* and *CyBitOperations.h* files and click **Open,** as Figure 44 and Figure 45 shows.

Figure 44. Adding Header Files to PSoC Creator Project

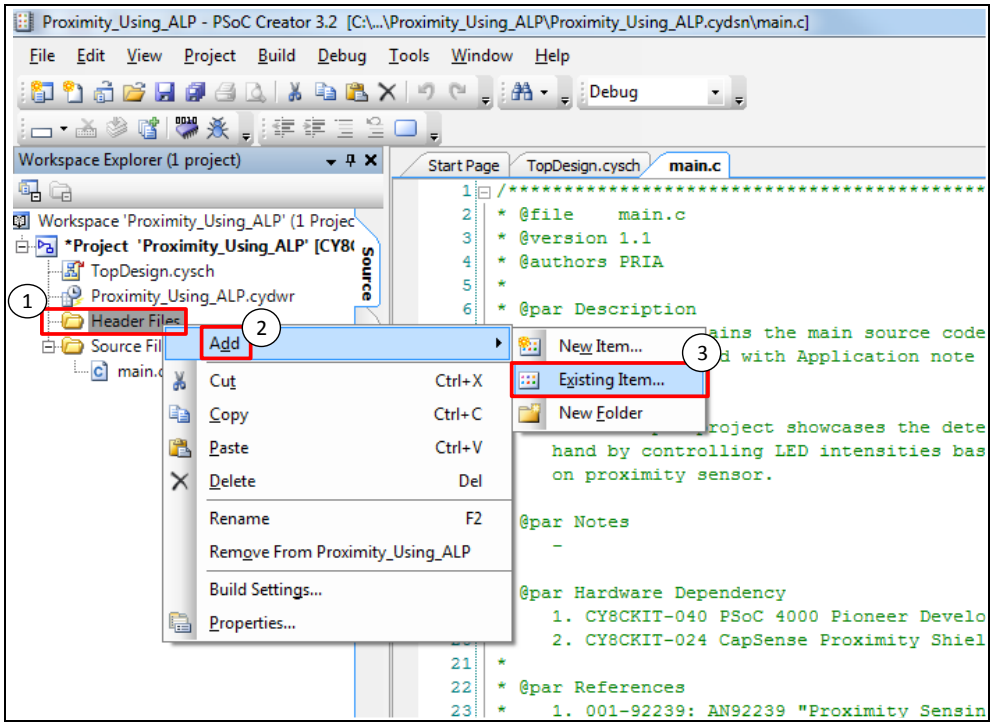
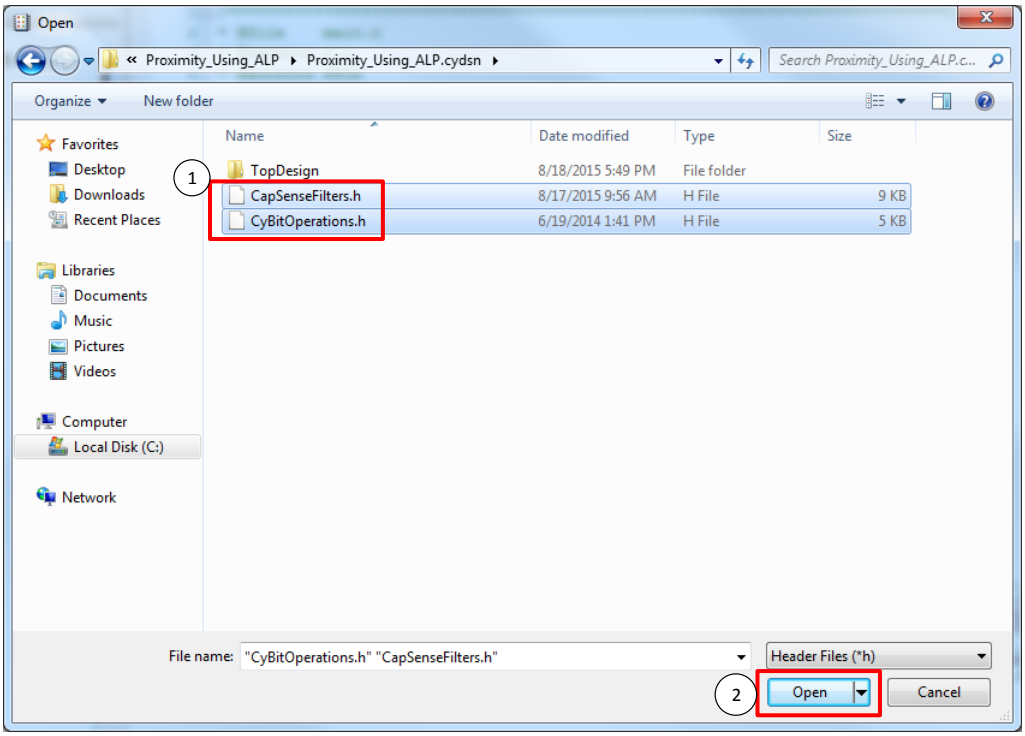


Figure 45. Adding ALP Filter Definition Files to the PSoC Creator Project



11. In the *main.c* file, include *CapSenseFilters.h* by using following line of code:

```
# include "CapSenseFilters.h"
```

12. Define the ALP thresholds by using the following lines of code:

```
/* Positive and Negative threshold used by ALP filter.
     * Refer AN92239 for details on how to tune positive and negative thresholds.
     */
    #define PROX_POS_THRESHOLD          (100u)
    #define PROX_NEG_THRESHOLD          (100u)
```

Note: The procedure to find out ALP thresholds is mentioned in section ALP Filter Tuning.

See the *main.c* of the example project Proximity_Distance_040 for details on where to place code for defining ALP filter thresholds.

13. Declare ALP filter variables by using the following lines of code:

```
/* ALP Filter variables */
/* Thresholds for adjusting ALP filter response */
uint8 proxPositiveTh[CapSense_TOTAL_PROX_SENSOR_COUNT];
uint8 proxNegativeTh[CapSense_TOTAL_PROX_SENSOR_COUNT];

/*Average Filter history */
uint16 averageHistory[CapSense_TOTAL_PROX_SENSOR_COUNT][AVERAGE_HISTORY_SIZE];

/* Indicates the oldest data amongst Average Filter histories */
uint8 avgOldestDataIndex[CapSense_TOTAL_PROX_SENSOR_COUNT] = {0};

/* State variable for ALP filter state machine */

CAPSENSEFILTERS_ALP_STATE_ENUM stateAdvancedFilter[CapSense_TOTAL_PROX_SENSOR_COUNT];

/* Average Filter output */
uint16 averageVal[CapSense_TOTAL_PROX_SENSOR_COUNT];

/* Following variables contain input histories for ALP filter */
```

```
uint16 filterHistory0[CapSense_TOTAL_PROX_SENSOR_COUNT];
uint16 filterHistory1[CapSense_TOTAL_PROX_SENSOR_COUNT];
uint16 filterHistory2[CapSense_TOTAL_PROX_SENSOR_COUNT];
```

Note: See the *main.c* of the example project Proximity_Distance_040 for details on where to place the code to declare ALP filter thresholds.

14. In the main() function in main.c, initialize the proximity positive, negative threshold, initialize advanced low pass filter with current raw of sensor and setting the k value for ALP filter by using following lines of codes

```
/* Initialize the proximity positive and negative threshold values */
proxPositiveTh[CapSense_SENSOR_PROX_0__PROX] = PROX_POS_THRESHOLD;
proxNegativeTh[CapSense_SENSOR_PROX_0__PROX] = PROX_NEG_THRESHOLD;

/* Initialize Advanced Low Pass (ALP) filter variables with current raw count value */
CapSenseFilters_InitializeAdvancedLowPass(
                                      CapSense_SENSOR_PROX_0__PROX,
                                        CapSense_ReadSensorRaw(CapSense_SENSOR_PROX_0__PROX)
                                            );
/* Set the k value for ALP filter */
 CapSenseFilters_SetAdvancedLowPassK(CAPSENSEFILTERS_IIR_K_64);
```

Note: The parameter < CapSense_SENSOR_PROX_0__PROX> is the sensor number for the proximity sensor. You need to replace this parameter with the proximity sensor # from your project

See the *main.c* of the example project Proximity_Distance_040 for details on where to place the code to declare ALP filter thresholds

15. The value of the argument to the function CapSenseFilters_SetAdvancedLowPassK() depends on the peak-to-peak noise in the raw count. Use peak-to-peak noise value to set the K-value per mapping in Table 6.
16. In the main.c function, enable the ALP filter by using following code just after while(CapSense_IsBusy()) code

```
CapSenseFilters_RunAdvancedLowPass(CapSense_SENSOR_PROX_0__PROX);
```

Note: See the *main.c* of the example project Proximity_Distance_040 for details on where to place the ALP code in your project.

17. Build the project, ensure that the project compiled without any errors.
18. Observe the raw count data and confirm that the ALP filter has reduced the noise in raw counts.
19. To tune the ALP filter, follow the procedure given in section ALP Filter Tuning.

Document History

Document Title: AN92239 - Proximity Sensing with CapSense®

Document Number: 001-92239

Revision	ECN	Orig. of Change	Submission Date	Description of Change
**	4421289	DCHE	07/22/2014	New Application Note
*A	4897476	AKSM	09/04/2015	Added section 10 Design Consideration to Achieve 30-cm Proximity-Sensing Distance Added Appendix D: Adding the ALP Filter Library to Any CapSense Project Updated Figure 3, Figure 4, Figure 5, Figure 6, Figure 7, Figure 12, Figure 22, Figure 23, Figure 24, Figure 25, Figure 26, Figure 27, Figure 28, Figure 30, Figure 31, Figure 32, Figure 33, Figure 34, Figure 35, Figure 36 Changed template of AN
*B	5094088	VAIR	01/20/2016	Added Glossary.

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

Automotive	cypress.com/go/automotive
Clocks & Buffers	cypress.com/go/clocks
Interface	cypress.com/go/interface
Lighting & Power Control	cypress.com/go/powerpsoc
Memory	cypress.com/go/memory
PSoC	cypress.com/go/psoc
Touch Sensing	cypress.com/go/touch
USB Controllers	cypress.com/go/usb
Wireless/RF	cypress.com/go/wireless

PSoC® Solutions

psoc.cypress.com/solutions

PSoC 1 | PSoC 3 | PSoC 4 | PSoC 5LP

Cypress Developer Community

Community | Forums | Blogs | Video | Training

Technical Support

cypress.com/go/support

Cypress Semiconductor
198 Champion Court
San Jose, CA 95134-1709

Phone : 408-943-2600
Fax : 408-943-4730
Website : www.cypress.com