Smart Capacitive Sensing Design with EFM8™ and Simplicity Studio™

Introduction

Adding capacitive sensing to a product can be a daunting challenge that requires the developer to maintain sensor robustness and responsiveness while minimizing current consumption and addressing other system-level priorities of the product. Navigating this embedded development process without the help of tools and support can lead to lengthy trial-and-error design iterations, both in software and hardware, which can result in slipped release schedules and suboptimal solutions.

While fixed function capacitive sensing solutions in the market can ease some of these burdens, those devices are rarely pure 'drop-in' solutions, as many still rely on developer-side configuration and calibration. Additionally, fixed function solutions prevent developers from giving the sensor additional responsibilities in a system, which can lower system current consumption, reduce board size, and decrease BOM cost.

The capacitive sensing solutions developed by Silicon Labs as part of the launch of the EFM8 8-bit MCU line help designers through each step of the development process, enabling customers to generate projects with a built-in capacitive sensing firmware library, add and debug features with an intuitive and powerful Simplicity Studio IDE, and view real-time capacitive sensing performance in-system using the Capacitive Sensing Profiler visualization tool.

Capacitive sensing design basics

Capacitive sensing technologies measure the capacitance of an electrode connected to the sensor's input. The electrode being measured is often a printed circuit board-designed round pattern of solid

copper about 10 mm in diameter that is isolated from the board's ground. A thin overlay, usually made of a type of plastic or glass, is sometimes adhered to the printed circuit board, and the resulting system can function as a touch interface that doesn't require the use of mechanical buttons. When a user touches the area of the overlay covering the electrode, the capacitance of the human body affects the capacitive coupling of the electrode, resulting in a change in capacitance measured of the on-chip capacitive sensor. Post-sample processing on the stream of samples will detect this change in capacitance and qualify the change as a 'touch'.

Because all capacitive sensing technologies share the common goal of sensing a relatively small change in an analog signal while operating in a mixed signal system that has the potential to be electrically noisy, designing a robust and reliable sensing interface can be challenging. In addition to hardware considerations involving board layout and electrode design, the sensor's post-sample processing operations must update state variables while responding dynamically to system-level events.

Touch sensing strategies usually involve maintaining a 'baseline', which is the expected output of the capacitive sensor when the electrode is in an untouched state, as well as setting one or more touch thresholds relative to that baseline. Processing compares these thresholds against sensor output to determine when touch and release events have occurred. In addition to these operations, the firmware must be designed to operate as efficiently as possible to minimize both code size and average current draw.

To help ease the burden of firmware development, many capacitive sensor vendors offer fixed function devices, which provide an integrated circuit that samples capacitance and outputs touch qualification through either a serial interface or port pin logic states. While these products may appear attractive to developers because of their ease of use, the lack of programmable flash and limited customizable feature sets can force system developers to place more responsibilities for touch qualification onto more power-hungry host processors.

For example, a fixed function device (FFD) that becomes susceptible to false positive touch events in high interference environments could force a host processor to ignore touch qualification information from the FFD and instead read the FFD's raw data and develop their own touch qualification algorithms. Similarly, an FFD erroneously sending frequent touch event signals to a host processor will cause that processor to wake from its lower power state to examine touch events, driving up average current draw for a system.

The capacitive sense-enabled EFM8SB1 MCU, when used with the Silicon Labs capacitive sensing firmware library, combines the ease-of-use of a fixed function device with the flexibility of a general purpose mixed-signal MCU.

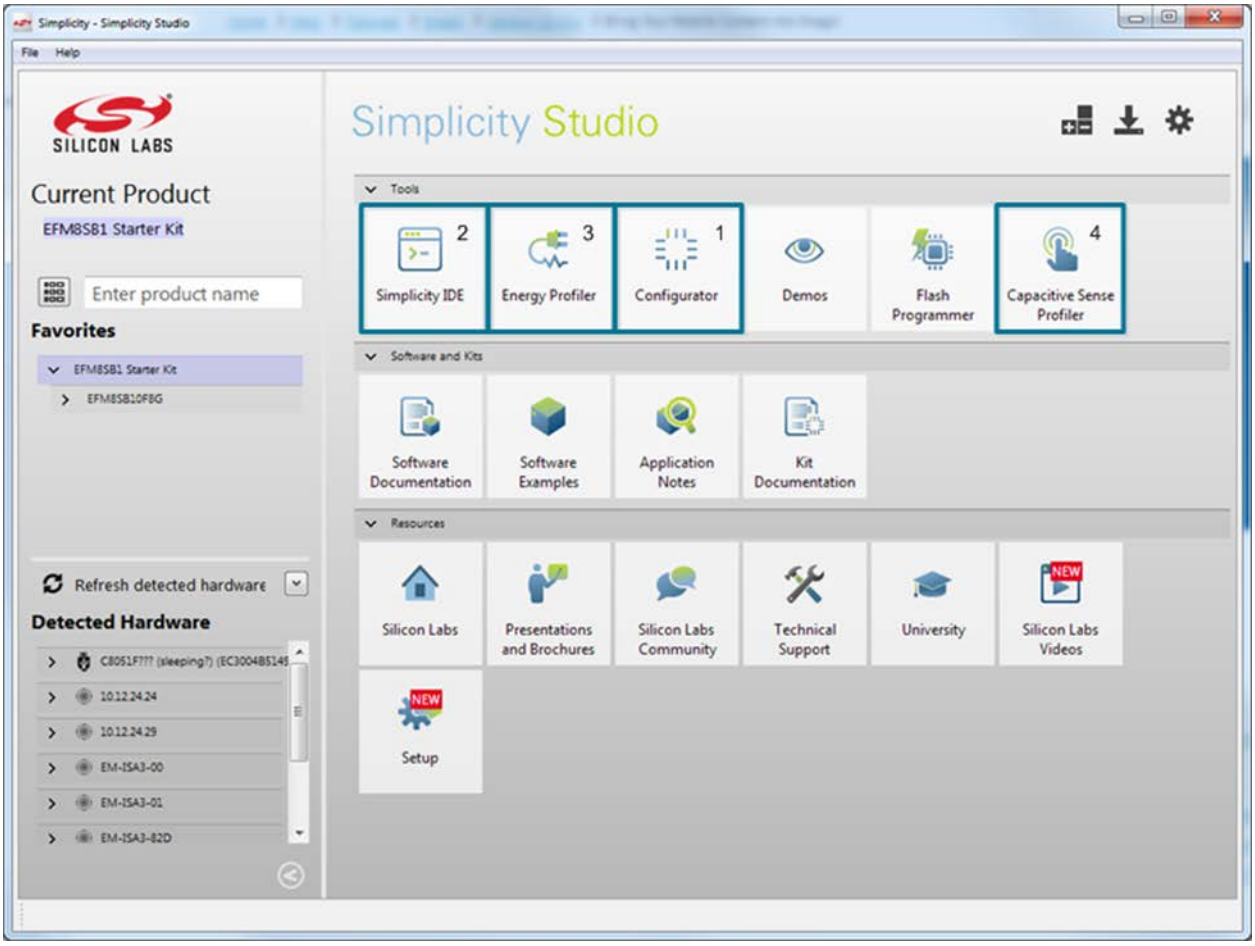
The precompiled library provides touch qualification algorithms, state variable maintenance, and a simple API for retrieving touch event information, while the remaining Flash and numerous on-chip peripherals give developers the opportunity to add more features and responsibilities to the MCU.

The library also leverages the EFM8SB1's power management unit (PMU) and on-chip real-time clock to create a sophisticated, mode-switching MCU that can achieve less than 1 uA average current draw in most use cases.

Simplicity Studio

Silicon Labs' Simplicity Studio provides end-to-end development support for capacitive sensing-enabled embedded system designers, from configuring and importing the capacitive sensing firmware library to in-system debugging and real-time visualization of capacitive sensing output.

The Simplicity Studio launcher window dynamically populates tiles depending on the MCU being used in development, and this window shows the tools offered to support every step of development.



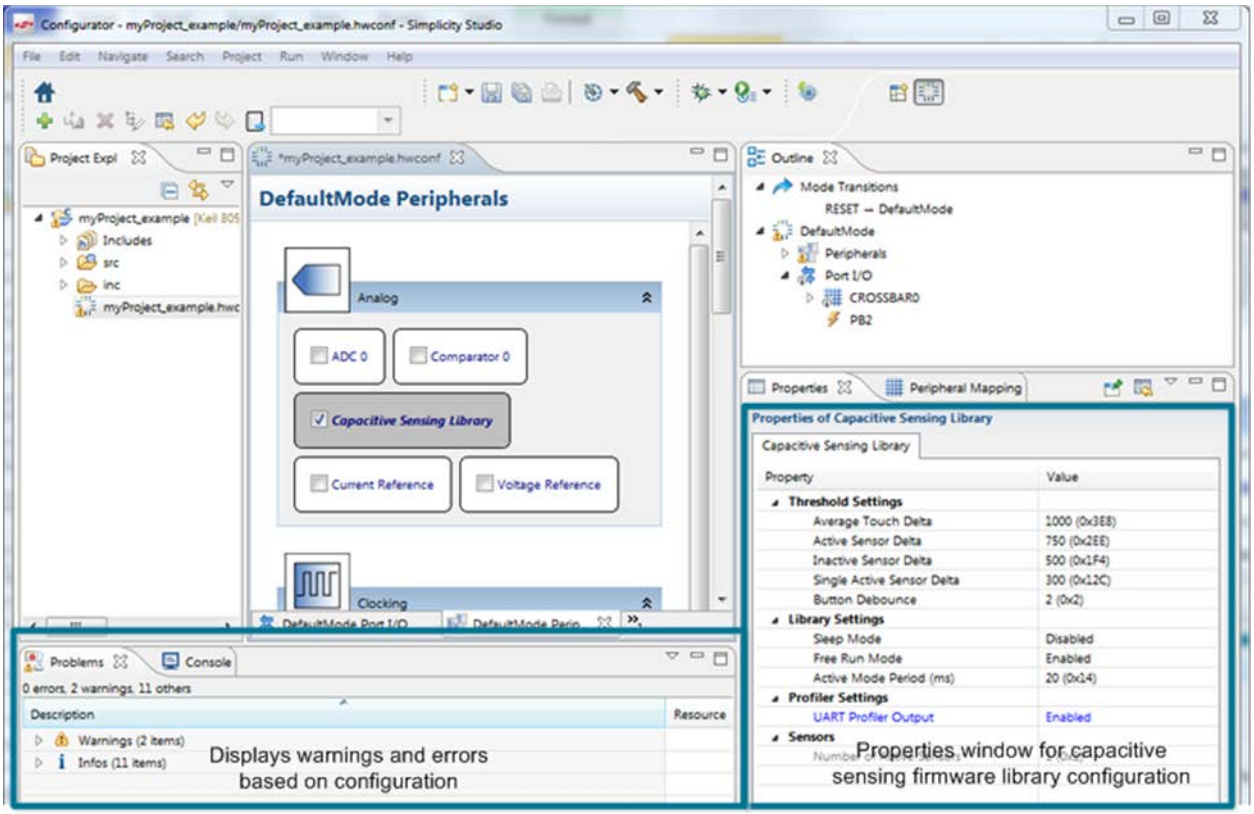
Simplicity Studio launcher window with highlighted tools

The annotations on the launcher figure above show the basic development path that most users will follow:

1. Configurator generates the capacitive sensing library and allows customers to configure hardware peripherals
2. Simplicity IDE provides a code development and debug environment where additional features can be added to a firmware project
3. Energy Profiler provides real-time current draw measurements of a project that has been downloaded to a starter kit evaluation board
4. Capacitive Sense Profiler displays runtime capacitive sensing data raw values and algorithm-derived values such as touch state, baselines, and thresholds

Configuring capacitive sensing projects

A project's development starts with the Configurator tool, which provides users with a graphical interface and properties windows. In this tool, customers can designate port pins as capacitive sensing input and configure library performance settings such as the scan period and low power functionality. If a customer's configuration choices require other configuration settings to be implemented, a Problems Perspective lists the relevant warnings and errors.



Simplicity Studio Configurator

As the user configures the project, appropriate files are generated and organized in the Project Explorer window. Clicking on one of these files will cause the configurator perspective to switch to the Simplicity IDE perspective. Users can go back to configurator and make changes or additions to a configuration anytime by clicking on the configurator file, which is also listed in the Project Explorer window.

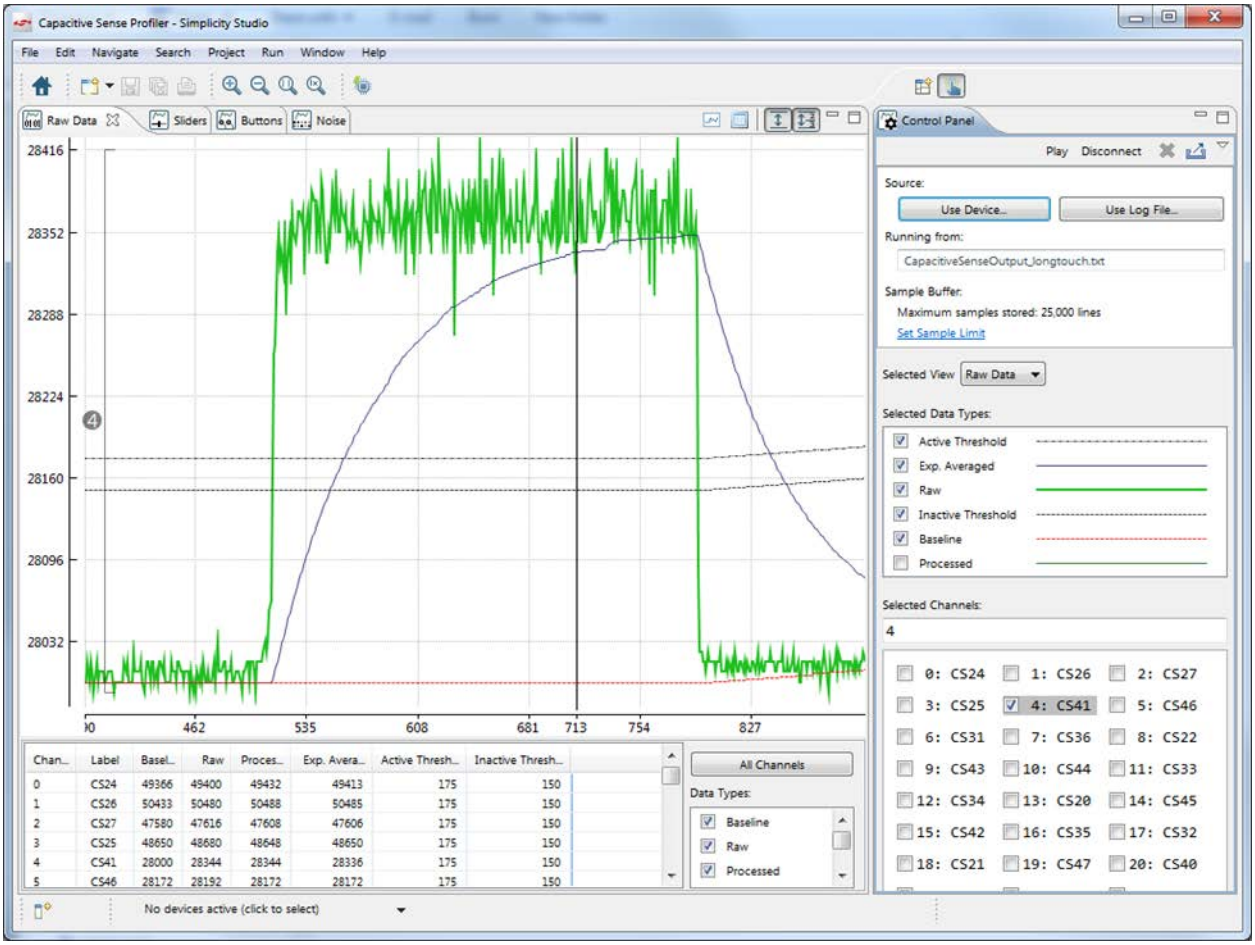
Code development and download in the IDE

The Simplicity IDE has all of the features that customers expect in an Eclipse-based environment, including code completion and formatting tools. The IDE also provides deep integration with the debug circuit on the evaluation boards as well as the Silicon Labs USB debug adapters, enabling full read/write access to MCU registers and variables.

When customers include the capacitive sensing firmware library in their projects, the configurator provides a build-ready project that will scan enabled sensors, qualify touches, and even output information through the serial interface. This feature enables customers to generate a project without any additional development in the IDE, meaning that if a user wants to test out capacitive sensing on a board, the IDE can simply be used to build and download the image, without any coding required.

Characterizing performance with Energy Profiler and Capacitive Sense Profiler

Once code has been downloaded to the EFM8SB1, both the Energy Profiler and the Capacitive Sense Profiler can be used to see how the configuration choices made in Configurator and any additional features created in the IDE affect current consumption and capacitive sensing performance. For capacitive sensing applications, the Capacitive Sense Profiler is particularly useful because it allows users to examine performance in-system, complete with product overlays and other late stage development system components in place.



Capacitive Sense Profiler

The profiler supports advanced features such as standard deviation and signal-to-noise ratio calculations to characterize board performance. The tool also allows the captured data to be exported into a text file for further examination in other programs such as Microsoft® Excel®.

Development workflow flexibility

One of the challenges in designing capacitive sensing systems is managing a project through an iterative development process where a user builds a project, tests performance, and then returns to development to further optimize performance. The tools in Simplicity Studio are carefully designed to support these kinds of workflow cycles.

For example, a developer might create a capacitive sensing project designed for thinner 1/16 inch overlays, build the project, and test performance in Capacitive Sense Profiler. If the product's requirements change, modifying the thickness of the overlay to 1/8 inch, the developer only needs

to click on the configurator file for that project and adjust the touch threshold settings. Once that change has been made, the developer just needs to rebuild and download the project, and performance with the thicker overlay can be seen in real-time through Profiler.

Adding other firmware features to a project as it develops provides another example of iterative development. Let's say that one developer has created a project with capacitive sensing, and has checked that all performance and current consumption requirements for the project are met using Simplicity's diagnostic tools. The developer then starts work on a serial interface that controls other system components based on capacitive sensing input. This new component could impose new requirements on current draw and capacitive sensing responsiveness.

With Simplicity Studio, this new component can be added to the project's code base without breaking the link between the source code and the configurator tool. Once the project is built, the developer might find that the functionality of the new component requires a modification to the capacitive sensing component. For example, the system might need to stay in an active mode and scan for button touches for a longer period of time than was first configured because the serial interface to the rest of the system needs to stay active and optimally responsive to input. The developer can go to configurator, adjust the amount of time the system stays awake before entering a low power sleep state between touch sessions, and then rebuild code.

Key sequence example

Simplicity Studio's ability to generate code and minimize the amount of low-level capacitive sensing code that the developer needs to write leads to many use cases where application-specific sensing features can be added to the sensing MCU. These use cases distinguish the Silicon Labs solution from fixed function solutions where all application-specific functionality must be added to more power-hungry processors in the system. A key sequence detector feature provides a clear example of the benefits of using a general purpose MCU in capacitive sensing.

In many products with control panels, the end user must enter a key sequence or password to unlock product functionality. In a system that uses a fixed function device, the main processor would have to wake up upon the first keystroke, wait in an active state while the user enters the key sequence, and then process the sequence to determine if it matches the correct password. This might not seem like a significant amount of time for a high power MCU to be active, but let's examine the total power budget. Let's say that this battery-powered control panel sees an average of 100 fifteen-second user interaction sessions per day. The sessions have two phases: a validation phase where the key sequence is entered and checked against a password, and a response phase where the control panel issues commands elsewhere in the system. In a design with a fixed function device, the main processor must be awake for both the validation and response phase, which in this example is 15 seconds per session. In the case where an MCU such as the EFM8SB1 MCU is used for the validation phase, the main processor only needs to be awake for the response phase, or about 7.5 seconds. Let's also say that the main processor uses 10 mA when active, which would be

common among many 32-bit processors, and the EFM8SB1 would use 20 uA when active, which is achievable due to the sophisticated power management state machine in the capacitive sensing firmware library. In this example, we're simplifying the current draw a bit and not taking into account the current used between user interaction sessions.

In the case where an FFD requires that the main processor be active during validation, the average current draw of the system would be about 17 uA. In the case where the lower power EFM8SB1 can be used for part of the interaction session, the average current draw would be about 9 uA. If this battery-powered system were running on a coin cell battery such as a CR2032, which typically has a capacity of around 225 mAh, the design using the EFM8SB1 would run without a battery change for almost 3 years. The system with an FFD would require a battery change in about 1.5 years, which is a significant reduction in run time that could mean the difference between a well-received product and a frustrating product destined for one-star Amazon reviews.

Smart design leads to better products

No two product designs will have the same requirements or end customer use cases. For this reason, using a 'one size fits all' fixed function device is unlikely to yield a design that manages responsibilities in coordination with other system ICs in a way that results in optimal functionality with minimal current draw. Using the EFM8SB1 MCU running the capacitive sensing firmware library with its tight integration with the tools offered by Simplicity Studio enables developers to find smart ways of incorporating capacitive sensing into a design while still balancing the other goals and priorities within the system.

Learn more about Silicon Labs' EFM8 MCUs visit www.silabs.com/efm8