Low Power Capacitive Sensing

1. Relevant Devices

C8051F990, C8051F991, C8051F996, and C8051F997

2. Supporting Documentation

- QuickSense™ Studio User's Guide
- AN366: QuickSense Firmware API
- AN367: Understanding Capacitive Sensing Signal to Noise Ratios and Setting Reliable Thresholds
- AN418: Baselining in the QuickSense firmware API
- AN447: Printed Circuit Design Notes for Capacitive Sensing with the CS0 Module

3. Introduction

Capacitive sensing for human interface applications has rapidly grown in popularity due to its ability to reduce manufacturing cost, increase product lifespan by eliminating mechanical components, and enhance product look and feel. In many applications, mechanical push-button switches and potentiometers are being replaced by capacitive switches, sliders, and control wheels to implement functions such as contrast, volume control, and power on.

When capacitive sensing was first introduced into the market, it was primarily for use during the active mode of a system, where the power of the sensing function is small compared to the power draw of the overall system in active mode. To save power, battery powered systems would disable the sensing function in the inactive mode and wake up using a mechanical stimulus.

With the advancement of capacitive sensing technology and its ability to be ultra low power, even the power switch may now be implemented with a capacitive sensor.

This application note discusses how the user interface of ultra low power applications can be designed using capacitive sensing technology.

Copyright © 2010 by Silicon Laboratories

4. System Definition

The key to designing an ultra low power capacitive sensing system is in its system definition, which is the starting point of every embedded system design. The steps necessary to complete the definition are creating a power budget, defining the power modes, and identifying the system tasks that need to be performed in each power mode. Once a rough definition is in place, power targets for each power mode should be established to provide the system designer a starting point for software and hardware development.

4.1. Power Budget

An application's power budget is one of the first parameters that should be specified in the system definition. The first step in determining the power budget is deciding how long the system is required to operate without replacing batteries. For some systems, the battery only needs to last 6 months. However, some systems are sealed or placed in remote areas where battery replacement would not be possible, and the required battery life may be 10–15 years. After required battery life is established, the battery which will be used to power the system should be selected. In many cases, cost or form factor determine the largest battery that will be available to the system. Other factors such as shelf life should be considered for systems that need an extremely long battery life. Finally, the battery capacity information found in the battery data sheet of the selected battery should be used to create the power budget.

The maximum average current (power budget) for a battery powered embedded system can be calculated using the following equation:

$$\text{Maximum Average Current [mA]} = \frac{\text{Battery Capacity [mA - H]}}{\text{Required Battery Life [H]}}$$

The power modes and system tasks should be designed such that the total average current of the system is less than the maximum average current specified by the power budget.

4.2. Power Modes

There are two primary power modes in any low power system: active and inactive. A typical device with a usage profile that involves interaction with a human will spend the vast majority of its life in the inactive mode, only switching to the active mode when interacting with the human. There may be multiple "active" modes depending on the level of interaction with the user or the task being performed, but these will all be grouped under the main "active" mode for the purpose of this discussion. The primary focus of the active mode is to provide a good user interface for short periods of time. The main goal of the inactive mode is to preserve battery life.

The active/inactive power scheme allows the system to provide a good user interface and have a long battery life because for a given battery, the average current of the system is what determines the battery life. Let us analyze a typical user interface that is used for an average of 15 minutes per day. Since there are 1440 minutes in a day, the system would be in the active mode only 1% of the time. This means that if the active mode supply current is 100 µA, its contribution to the average current is only 1 µA, a factor of 100 less than the actual active mode current. It is clear that small variations in the active current do not have a significant impact on battery life. On the other hand, the average current contribution from the inactive mode cannot be divided down (1 µA inactive mode current = 1 µA contribution to average current) since the system stays in this mode for greater than 99% of the time. If we analyze this scenario carefully, we can observe that when designing a human interface, it is most important to focus on the inactive current because reducing it by a small quantity (e.g., < 1 µA) can significantly improve battery life and increasing it by even "tens of micro amps" can cause a significant reduction in battery life.

4.3. System Tasks

The next step in creating the system definition is to identify the required tasks in each power mode. Power targets for each power mode should also be established at this point.

4.3.1. Active Mode

The active mode is enabled when a user is actively interacting with the device. The user interface should be fully functional and respond quickly to user stimulus. All capacitive sense inputs used by the application need to be sampled and gesture detection performed on sliders and control wheels. The switch sampling rate in active mode is typically set in the range of 20–125 Hz to ensure a responsive user interface. The active mode target current for most applications is in the range of 100 to 500 µA.

To achieve such a low active mode current, it is necessary to break up the active mode tasks into two categories. The first category of tasks requires the CPU to be active, such as determining the system state, processing of capacitive sensor data, or other housekeeping tasks. The second category of hardware tasks can be accomplished with the CPU in an idle mode, such as sampling the capacitive sensors, taking an ADC measurement, etc. Splitting these tasks into different power modes allows a supply current to be assigned to each power mode. The two active power modes can then be averaged to obtain the final value of “active mode current”.

4.3.2. The Inactive Mode

The primary goal of the inactive mode is to preserve battery life. The inactive mode should implement a low power “wake-on-touch” algorithm to determine when the system needs to be switched to the active mode. A common “wake-on-touch” algorithm periodically samples one switch at a rate of 1–10 Hz to check if a finger has been placed on the capacitive touch pad. The ultra low power RTC is used to schedule the “wake-on-touch” checks. Target power consumption for the inactive mode in most applications is 1 to 3 µA.

5. Hardware Design

The hardware design of an ultra low power capacitive sensing embedded system consists of three easy steps:

- Determine the number of capacitive touch pads required.
- Select an ultra low power capacitive sensing MCU.
- Design a PCB with the MCU and the capacitive touch pads.

5.1. Determining the Capacitive Touch Requirements

The number of capacitive touch inputs required depends on the complexity of the user interface. Each button in the user interface requires one capacitive touch input. Sliders and control wheels are typically implemented with 4 to 8 inputs. Once the number of capacitive sense inputs is determined, it is time to move on to MCU selection.

5.2. Selecting an Ultra Low Power Capacitive Sensing MCU

For ultra low power capacitive sensing applications, it is important to select a capacitive sensing MCU that is power efficient and that has an ultra low power sleep mode that supports periodic wake-up (e.g., real time clock). One example of such an MCU is the C8051F99x family of ultra low power capacitive sensing MCUs.

Key Features of the ‘F99x MCU Family:

- 8 kB Flash, 512 bytes RAM, 25 MIPS CPU with 150 µA/MHz active mode current
- Autonomous capacitive sensing peripheral with less than 40 µs conversion time (adjustable via the CS0MD2 register)
- Ultra low power sleep mode (300 nA) with internal LFO and 2 µs wake-up time
- 10-bit, 300 ksps or 12-bit, 75 ksps ADC with internal voltage reference
- 13/14 capacitive sense inputs in a 3x3/4x4 mm package

In addition to the key features listed above, the capacitive sensing peripheral on the C8051F99x MCU family has some built-in architectural power saving features.

5.2.1. Sensing Multiple Channels in a Single Conversion

Each capacitive sensing conversion requires a finite amount of energy to complete. Reducing the total number of required conversions can reduce the amount of energy needed to perform the necessary sensing. The C8051F99x family features a “multiple channel sense” feature where multiple channels may be bonded together at runtime and sensed using a single conversion. This feature is useful for implementing low power wake-on-touch on any button in a multi-button arrangement. For best performance, the channels bonded together should have capacitive pads that are similar in size and shape. Having capacitive pads with similar size and shape allows bonded channels to provide equal weight to each pad in the combined measurement.

5.2.2. Suspend Mode Wake-Up Source

The capacitive sensing peripheral (CS0) on the C8051F99x MCUs has a built in oscillator that controls conversion timing that is independent of the system clock. This allows the MCU to enter a low power suspend mode while a conversion is taking place. The CS0 peripheral has the ability to wake up the MCU from suspend mode after the capacitive sensing conversion is complete. This allows the supply current while taking conversions to be as low as 120 µA.

5.2.3. Autonomous Hardware Averaging

In noisy environments, multiple capacitive sensing conversions are needed in order to improve resolution. The CS0 peripheral can automatically average 1, 4, 8, 16, 32, or 64 conversions for each convert start without any CPU intervention. The CPU may also enter suspend mode to be awoken after all conversions are accumulated and averaged.

5.3. Designing a PCB with an MCU and capacitive touch pads

User interfaces that use capacitive touch technology require very few external components and can be very simple to design as long as a few basic rules are followed. To demonstrate the simplicity of such designs, we will use the ‘F990 Slider Evaluation Board shown in Figure 1 as an example.

Figure 1. C8051F990 Slider Evaluation Board

The bill of materials used for this board is as follows:

- Power Source: CR2032 battery holder and 1 µF bulk decoupling capacitor. In this design, a bulk capacitor is used because a CR2032 battery has high output impedance and its peak output current is limited. Adding a capacitor preserves the battery and increases the peak output current of the power source by instantaneously providing the system with the necessary charge to meet peak current demands. The battery only needs to provide the “average current” to the capacitor in order to maintain the supply voltage.
- MCU circuit: The MCU used is a C8051F990, which comes in a tiny 3x3 mm package and provides 16 I/O pins. The only external components needed are a pull-up resistor for the reset pin (to provide noise immunity) and a small decoupling capacitor close to the VDD pin.
- LEDs: There are 10 LEDs used in this design and 10 current limiting resistors associated with the LEDs. These are application specific and will not be needed for most systems.
- Capacitive Sense Pads: Six capacitive sense pads are used on this evaluation board to provide a method of user input. The pads are arranged in a chevron slider pattern and can be used as “buttons” or combined to form a “slider”. An acrylic overlay is typically placed over the capacitive pads to protect the system from ESD and to provide a uniform surface that may be touched by the end user.

Capacitive sense pads may be connected directly to Capacitive Sensing pins on the MCU. The C8051F990 provides 13 pins with capacitive sensing capability. It is a good idea to route the capacitive pads to the MCU on the bottom layer of a 2-layer board or an inner layer of a multi-layer board because they are sensitive to touch. Guidelines for designing capacitive sensing pads can be found in “AN447: Printed Circuit Design Notes for Capacitive Sensing with the CS0 Module.”

6. Firmware Design

The QuickSense™ development tools make writing firmware that performs capacitive sensing very easy. The QuickSense Configuration Wizard, shown in Figure 2, allows the firmware designer to quickly generate a firmware project that runs on custom hardware. A graphic user interface is used to allow the system designer to input information about the system, such as the pin mapping of capacitive sense pads, group assignments, sampling rate, etc. Finally, the user may use the QuickSense Configuration Wizard to export a fully working project that may be used with the performance analysis tool or customized to suite the needs of the application.

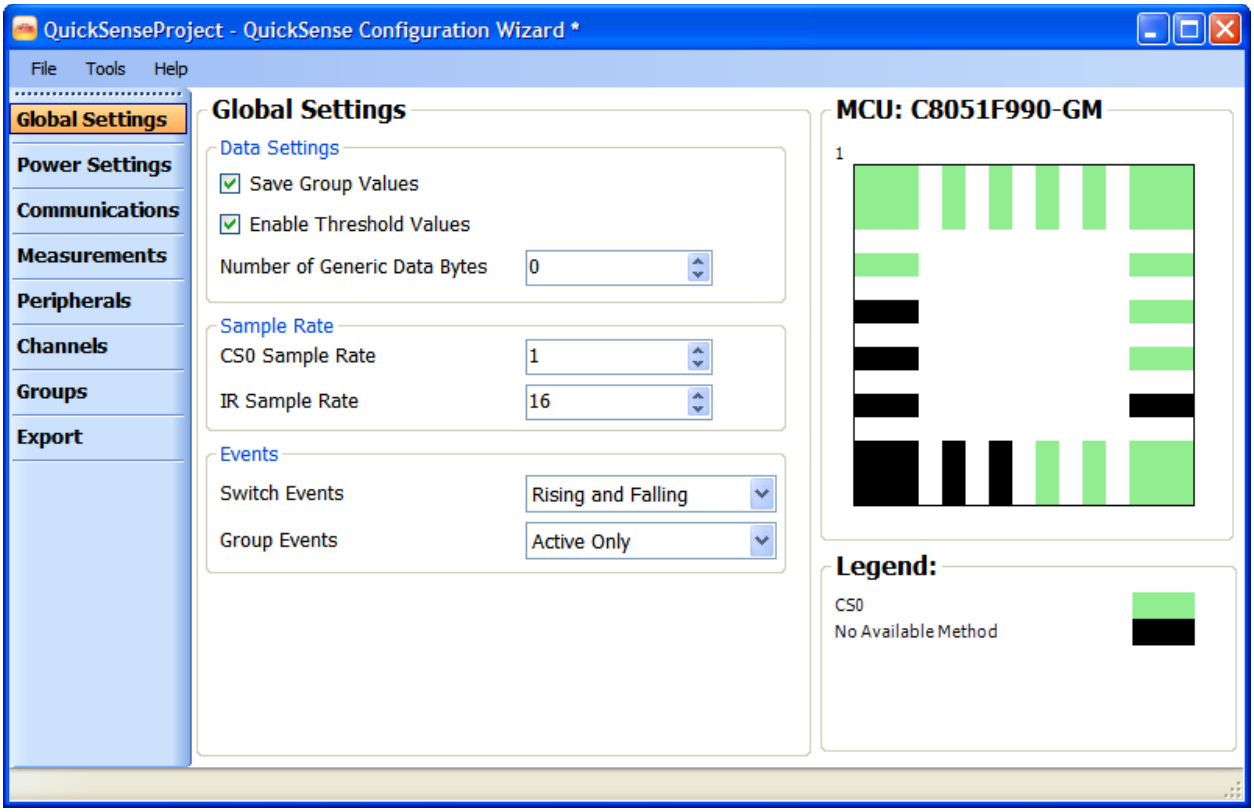

Figure 2. QuickSense™ Configuration Wizard

6.1. Generating a Project for a Low Power System

To generate a project for a low power system, start a new project for an MCU in the C8051F99x family using the File→New Project… menu. Leave all the Global settings at their default values, and click on the Power Settings Tab. Click the Enable Low Power Mode checkbox as shown in Figure 3. Also set the wakeup interval between 10 and 50 ms.

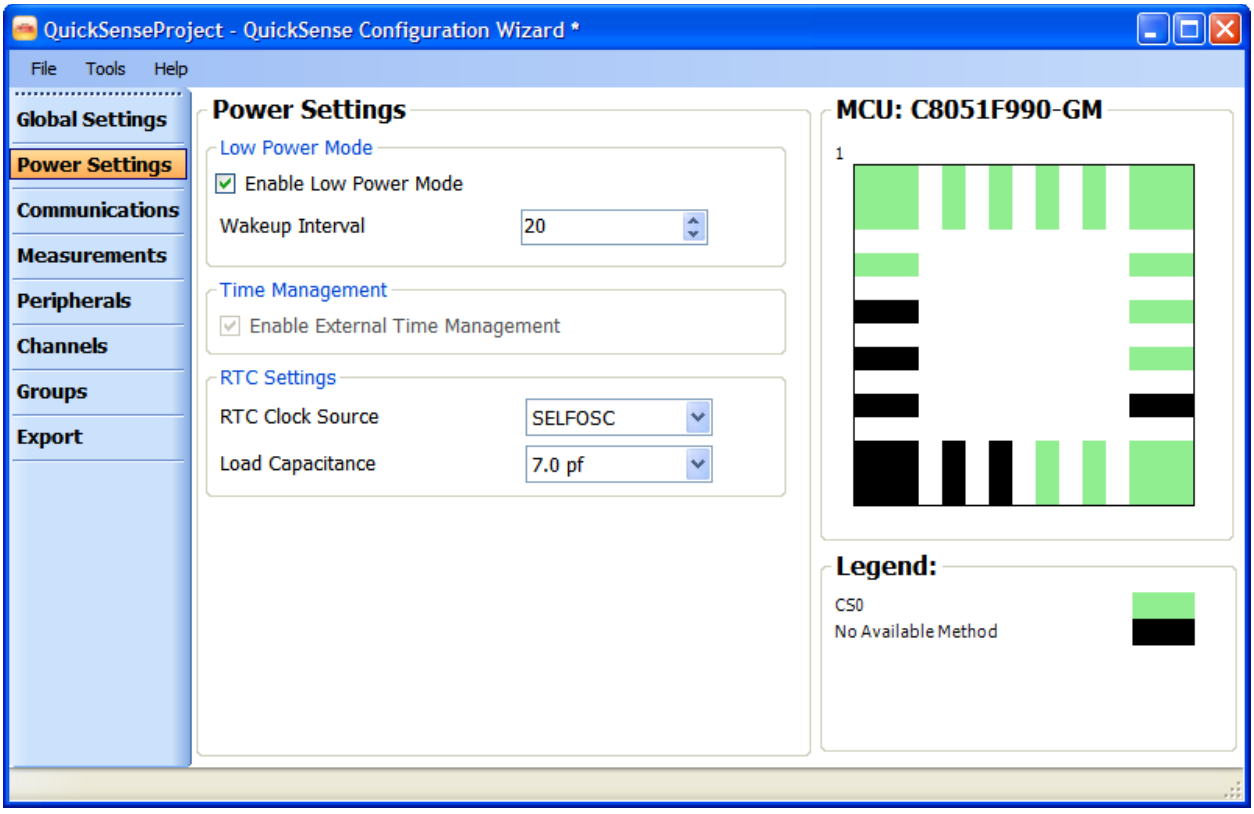

Figure 3. Power Settings

The Communications, Measurements, and Peripherals tabs can be left at their default values. The channels which will be assigned to capacitive sense pins should be specified in the Channels tab as shown in Figure 4. Simply click on a channel to select it as a capacitive sense input. Channels bound together to form a group, such as a Slider, are specified in the Groups tab. See the Help menu of the QuickSense Configuration Wizard for details on how to use the wizard.

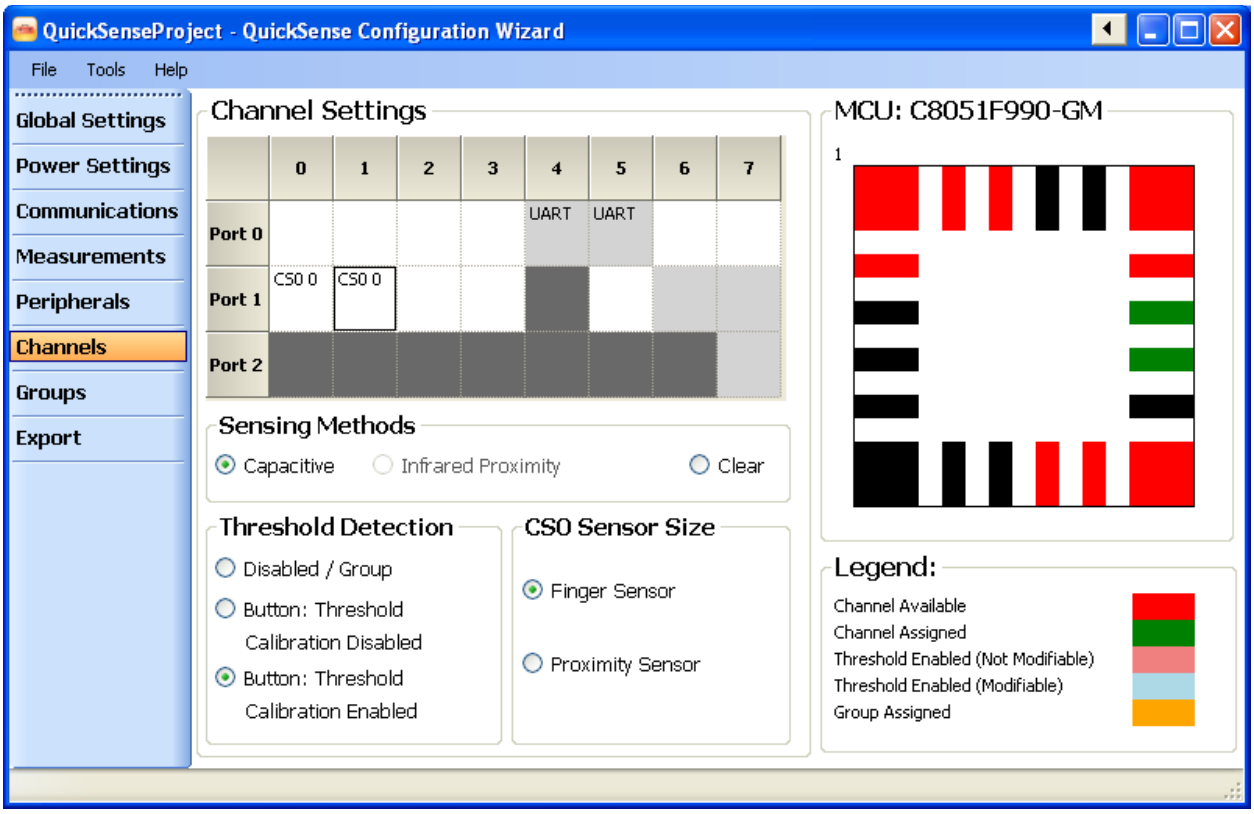

Figure 4. Selecting Channels for Capacitive Sensing

After system parameters are specified in the wizard, it is time to export a project. The Export tab, shown in Figure 5 can be used to select the desired tool chain and create a project that can be compiled and downloaded to your hardware. After generating the project, the wizard will prompt you to open the new project in Silicon Laboratories IDE. This project should compile and build without any warnings or errors.

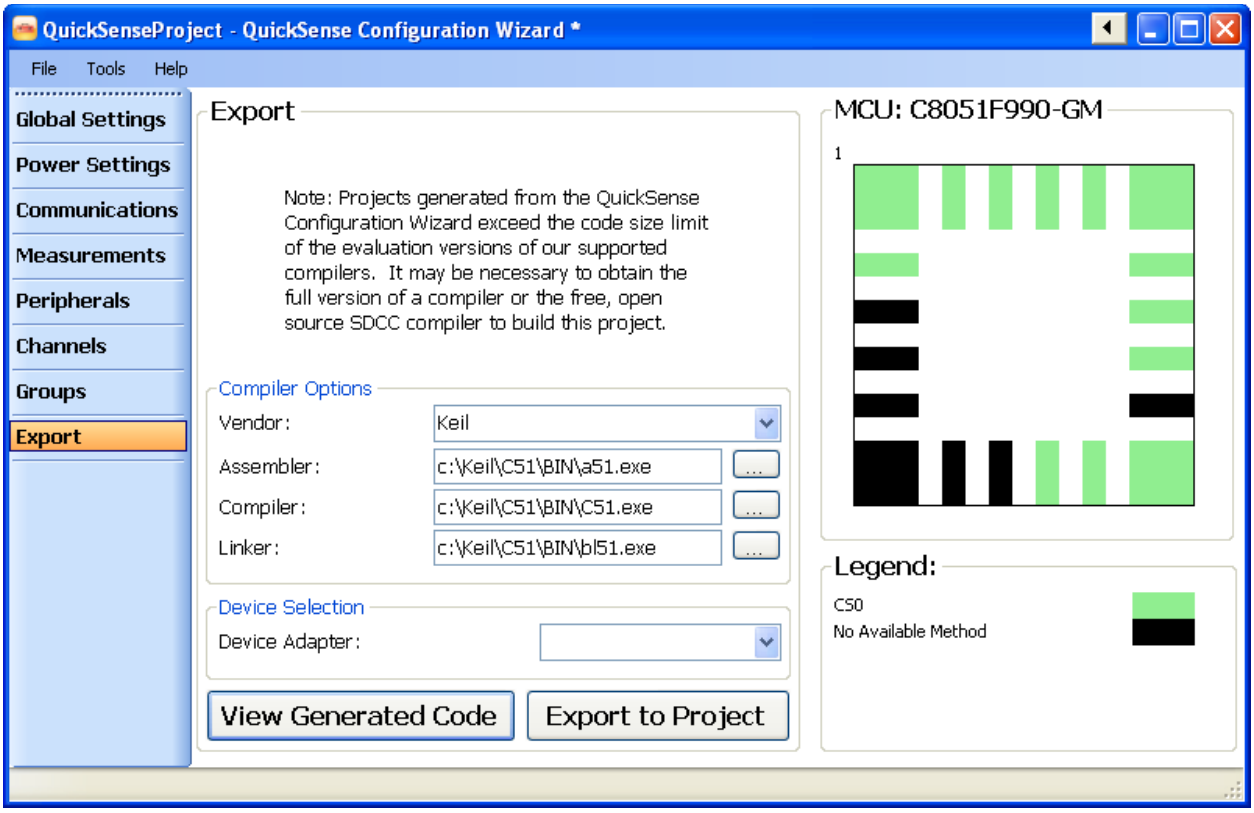

Figure 5. Export Tab in QuickSense Configuration Wizard

After exporting a new project, a few modifications should be made to the software to allow use with the performance analysis tool. These changes are described below.

In QS_Config.h:

- Set the SERIAL_INTERFACE to QSCI_LITE.
- Set the FLASH_MODE to READ_WRITE.

In QA_Config.c:

- In the QS_ChannelInfo[] array, make sure channels are set to THRESHOLD_MODIFIABLE.

After making these changes, the code should be re-built. Check the code size to ensure it does not exceed 8192 bytes. After debugging, these settings may be reversed in order to reduce program memory size. It may be a good idea to keep two versions/copies of the project or use conditional compilation to include or exclude code used for debugging. The final image released to production will typically have the serial interface disabled, as it is only needed during debugging and the final firmware can make better use of the program memory space.

6.2. Using the Performance Analysis Tool to Set Capacitive Pad Thresholds

The project generated by the QuickSense Configuration Wizard includes the QuickSense Firmware Library which provides functionality for sampling capacitive touch pads, applying baselining principles to ensure proper detection thresholds, providing callback functions for the detection of “button down” and “button up” events, and detecting the location of a press on multi-pad inputs such as sliders and control wheels. It also provides all required system timing and an event driven framework that keeps the system in sleep mode waking up only to perform required system tasks and quickly return to sleep mode. Functions to manage the SmaRTClock are included in the QuickSense Firmware Library.

The QuickSense Firmware Library needs a one-time calibration to set channel thresholds so that it can properly detect a finger press. The performance analysis tool, shown in Figure 6 allows the system designer to view the raw channel data in a graphical format and set channel thresholds so that the QuickSense firmware library is able to properly detect the channel state.

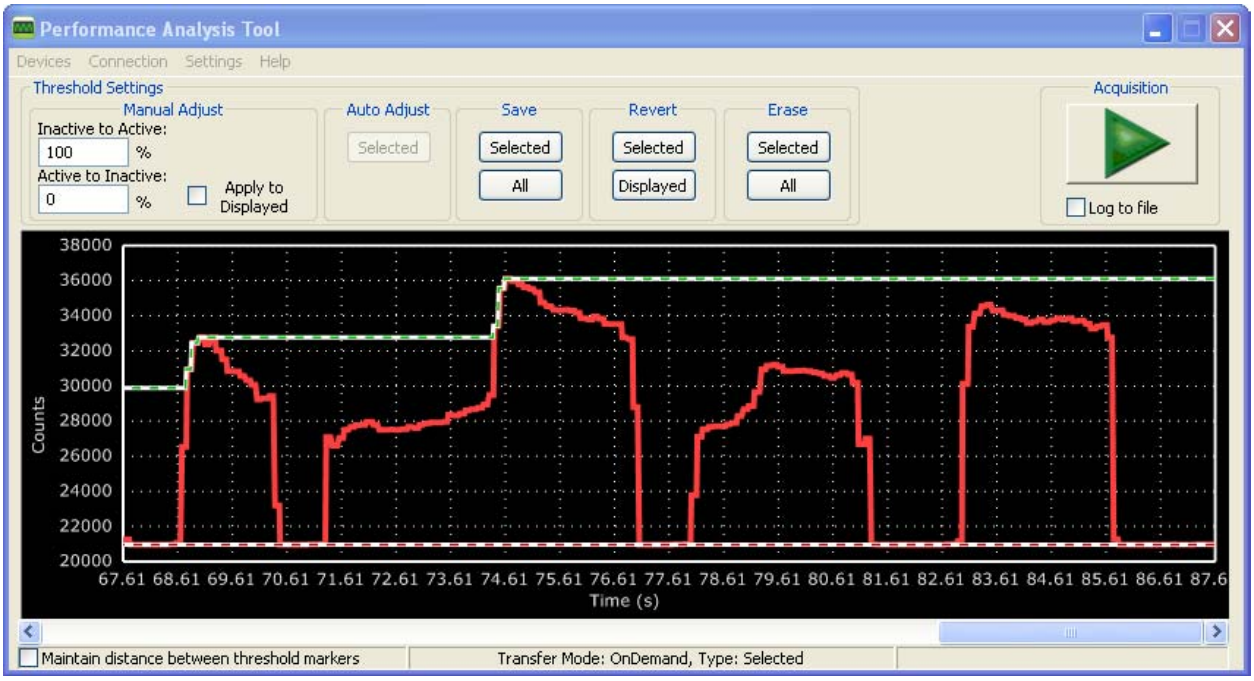

Figure 6. Performance Analysis Tool

The performance analysis tool operates over the device serial port. In most systems, the pins used for UART (typically P0.4 and P0.5) may be “borrowed” to allow communication with the performance analysis tool An easy way to connect the microcontroller to the PC for using the performance analysis tool is to use a CP2102 USB to UART bridge. This device takes TTL level UART signals and creates a virtual COM port on the PC, as shown in Figure 7. The communication interface between the CP2102 and the PC is USB. Once debugging with the performance analysis tool has been complete, these pins may be re-used in the system.

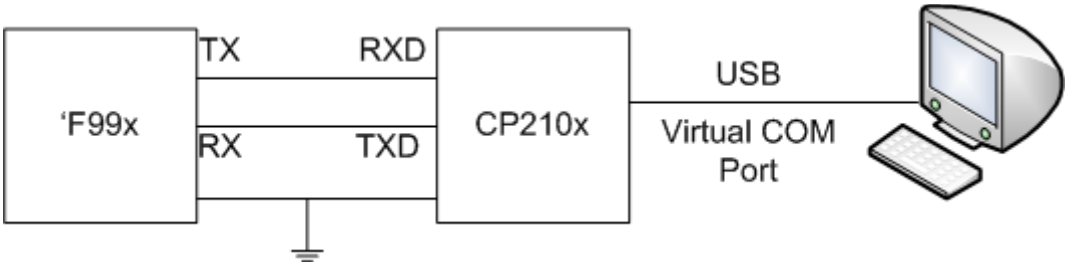

Figure 7. Creating a Virtual COM port using a CP210x USB to UART Bridge

Once the MCU's serial port is connected to the PC, the performance analysis tool is ready to be launched. After establishing a connection with the target, enable the channel for which thresholds will be set by clicking on the button (not the checkbox) on the right hand side of the screen as shown in Figure 8. Channels for which the checkbox is checked are "visible" on the screen, however, the channel for which the adjacent button is pressed is "selected" and its thresholds may be set. The channel being operated on must be "selected".

Figure 8. Selecting a Channel in the Performance Analysis Tool

There are three parameters which may be set for each channel: Upper Threshold, Lower Threshold, and the Reference Baseline Magnitude. These constants are stored in flash memory at the beginning of the second sector. On devices where flash memory is organized in 512 byte pages, the starting address of the channel calibration data is 0x200 (512d). The layout of the calibration data on the second flash sector is shown in Figure 9. The size of the calibration space in bytes is (4 x number of channels) + 1. The very last byte of the calibration space is an 8-bit number indicating the baseline update rate in seconds.

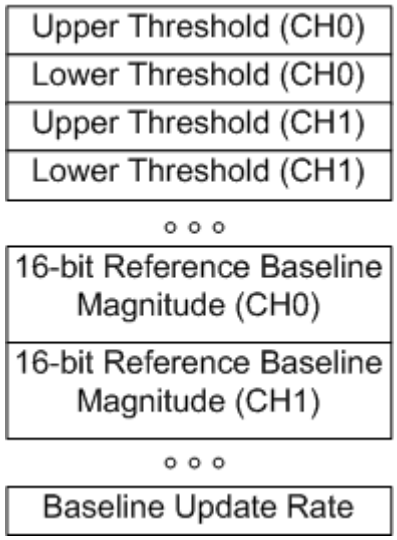

Figure 9. Channel Calibration Space Layout

The channel calibration data is described in AN367. Let us briefly summarize the user configurable parameters:

- Reference Baseline Magnitude: The number of CS0 codes between a touched channel and an untouched channel.
- Upper Threshold: A percentage (0 to 100%) of the reference baseline magnitude which results in the detection of a rising edge event.
- Lower Threshold: A percentage (0 to 100%) of the reference baseline magnitude which results in the detection of a falling edge event.
- Baseline Update Rate: The number of seconds between baseline updates.

The following methods can be used to store calibration data in the device:

- Using performance analysis tool to store data for each channel.
- Creating an array in firmware starting at the second Flash sector.
- Manually filling in the thresholds using the memory window.

In most applications, a combination of the methods above will be used during the development cycle. In the following paragraphs, we will describe how the performance analysis tool can be used to arrive at the desired values. After the values are determined, the method which requires the least amount of maintenance is creating an array in firmware to hold the calibration data.

When using the performance analysis tool on a channel that has not been calibrated, the first parameter that should be set is the reference baseline magnitude. Touch and release the capacitive sense pad using normal pressure until the waveform on the screen looks like Figure 6. If you apply too much pressure to a pad and the baseline magnitude increases too much, simply erase the threshold settings for the selected channel and start over. Once you have the proper reference baseline displayed on the screen, you can save data for the selected channel using the Save button. For additional control of the baseline magnitude, you can select the Settings→Baseline Magnitude menu item while the tool is acquiring data.

After the baseline magnitude is set, it is time to set the upper and lower threshold. Place the cursor over the dashed lines until the cursor turns into a “hand”. The hand tool will allow you to grab the upper and lower thresholds and move them to the desired location. You can also manually adjust the threshold by typing a percentage into the text boxed located at the top of the screen. Data acquisition may be running or stopped for this process. Once you have set the desired thresholds, click the Save Selected button to save threshold data to flash memory.

After all the channels have been calibrated, it is a good idea to go ahead and set the baseline update rate using the Settings→Baseline Update Period menu command.

6.3. Adding Application Code

The QuickSense firmware library provides the framework for sampling capacitive sense switches and comparing values against pre-determined thresholds. It also contains support for groups, such as sliders, control wheels, etc. The QuickSense firmware library interacts with application code through the use of events. Rising edge (finger detected), falling edge (finger removed), button active, and group active events are the primary events that are captured by the firmware library. The system designer can modify the QS_Events.c file to specify the actions that should be taken when a button or slider is touched. Refer to AN366 for complete details on the QuickSense firmware library.

Contact Information

Silicon Laboratories Inc.

Silicon Laboratories Inc.
400 West Cesar Chavez
Austin, TX 78701
Please visit the Silicon Labs Technical Support web page:
https://www.silabs.com/support/pages/contacttechnicalsupport.aspx
and register to submit a technical support request.

The information in this document is believed to be accurate in all respects at the time of publication but is subject to change without notice. Silicon Laboratories assumes no responsibility for errors and omissions, and disclaims responsibility for any consequences resulting from the use of information included herein. Additionally, Silicon Laboratories assumes no responsibility for the functioning of undescribed features or parameters. Silicon Laboratories reserves the right to make changes without further notice. Silicon Laboratories makes no warranty, representation or guarantee regarding the suitability of its products for any particular purpose, nor does Silicon Laboratories assume any liability arising out of the application or use of any product or circuit, and specifically disclaims any and all liability, including without limitation consequential or incidental damages. Silicon Laboratories products are not designed, intended, or authorized for use in applications intended to support or sustain life, or for any other application in which the failure of the Silicon Laboratories product could create a situation where personal injury or death may occur. Should Buyer purchase or use Silicon Laboratories products for any such unintended or unauthorized application, Buyer shall indemnify and hold Silicon Laboratories harmless against all claims and damages.

Silicon Laboratories and Silicon Labs are trademarks of Silicon Laboratories Inc.
Other products or brandnames mentioned herein are trademarks or registered trademarks of their respective holders