BASELINING IN THE QUICKSENSE™ FIRMWARE API

1. Introduction

Baselining is the QuickSense™ Firmware API's method of adjusting capacitance and proximity/ambient light sensing measurements to compensate for changes in environmental and operating conditions. Without baselining, calibrated active and inactive threshold values can become invalid as conditions around the device change. This document covers the following:

- Problems caused by changing environmental and operating conditions
- Baselining solution and implementation
- Baselining and the human interface serial interface (HISI)
- Best practice recommendations

This document assumes that the user is already familiar with the QuickSense Firmware API and the use of active and inactive thresholds. For more information about the QuickSense Firmware API, refer to AN366. For more information about thresholds, refer to AN367. All functions described in this document are included in the QuickSense Firmware API.

2. Glossary

Below is a set of definitions for some of the key terms found in this document.

- Channel—capacitive sensing pad or light-sensing device measured by the MCU
- Active—when a channel is being used (e.g., pressed)
- Inactive—when a channel is not being used (e.g., unpressed)
- Baseline—a standard at which things are measured or compared
- Range—difference between active and inactive baselines
- Signal—capacitance or light sensing device output measured by the QuickSense Firmware API
- Reference data—saved during calibration into non-volatile memory; compared to runtime data to adjust for changing environmental and operating conditions
- Runtime data—measured by firmware and stored in volatile memory; compared to reference data to adjust for changing environmental and operating conditions
- Offset—difference between Reference and Runtime Inactive Baselines
- Gain—ratio between the channel's reference range and runtime range

Figure 1 illustrates some of the key terms

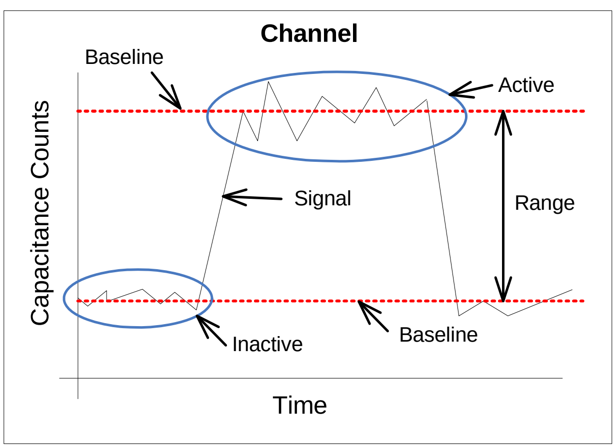

Figure 1. Illustration of Baselining Terms

3. Baselining Overview

During a typical calibration sequence, samples are taken when the channel is inactive and active. This allows the user calibrating the firmware system to know the range of values that can be expected from the channel during operation. The reference baseline magnitude is the difference between the averaged active channel value and averaged inactive channel value measured during calibration. Thresholds are defined as percentages of the reference baseline magnitude. A channel is considered active when the channel's value crosses the inactive-to-active threshold percentage within the channel's expected range and remains active until the channel's value crosses the active-to-inactive threshold percentage. A channel is considered inactive when the channel's value crosses the active-to-inactive threshold percentage within the channel's expected range, and remains inactive until the channel's value crosses the inactive-to-active threshold percentage. Figure 2 illustrates this process. Note that the examples in this section show how environmental and operating conditions can affect capacitive sensing measurements.

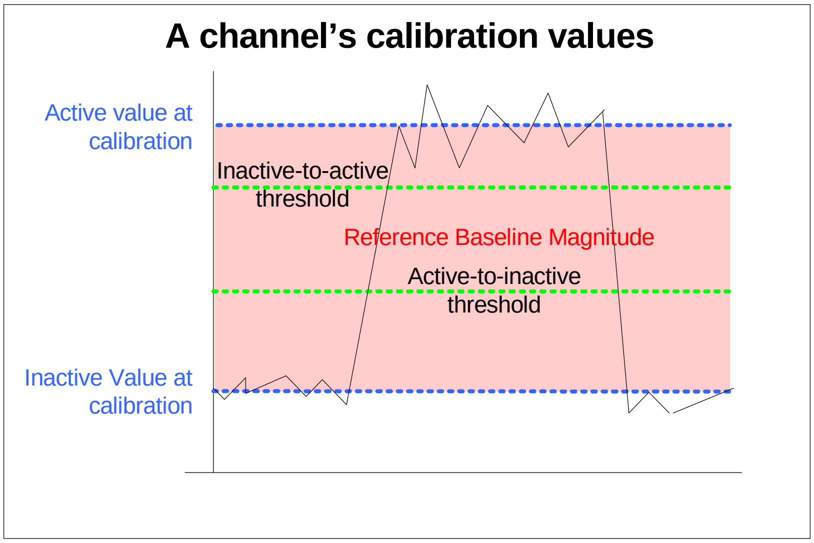

Figure 2. Initial Calibration

Environmental variables such as temperature and humidity could cause changes to a channel's inactive capacitance. Let's assume Figure 2 shows the initial calibration of a system. Figure 3 shows the same system operating in an environment with a lower temperature. This is classified as an offset problem since the Runtime Inactive Baseline is different from the inactive value measured at calibration.

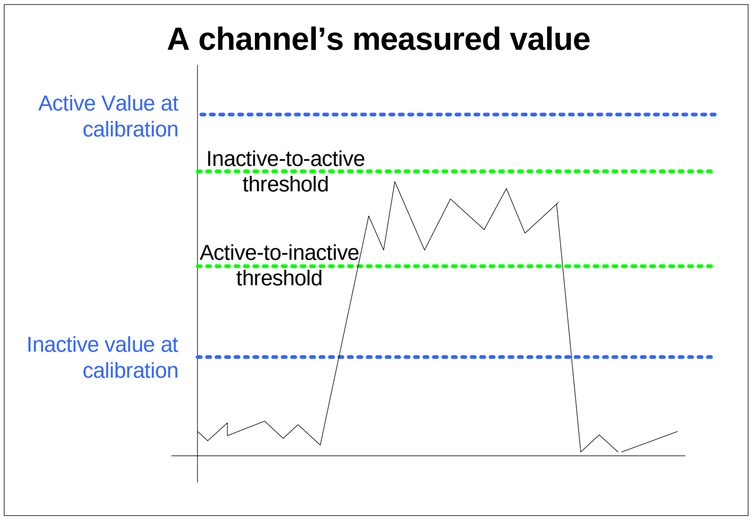

Figure 3. Channel with an Offset Problem

A change in the operating conditions causes another issue. Let's assume that Figure 2 shows a person with a large finger calibrating the system. The reference baseline magnitude defines a gain of one. Figure 4 shows a person with a smaller finger using the system. The smaller finger introduces less conductive material than a larger finger, which causes a lower capacitance to be measured when the channel is active. This results in a gain problem as the gain is roughly one half of the expected value.

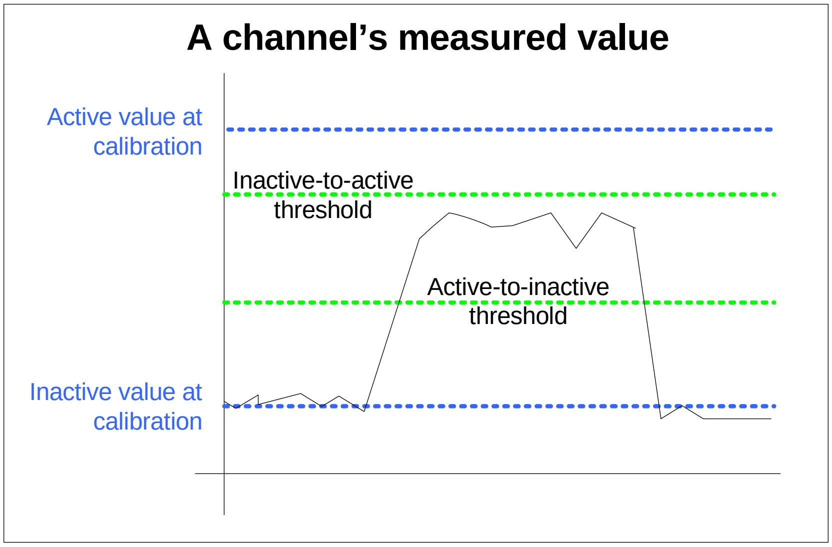

Figure 4. Channel with a Gain Problem

The same situation occurs if a channel was calibrated with an ungloved hand and a gloved hand uses the channel. The increased distance caused by the glove, as well as the glove's change to the dielectric constant, results in a lower measured capacitance.

Figure 5 shows a combination of these two situations. Figure 2 shows a person with a large finger calibrating the device in a warm environment while Figure 5 shows a person with a smaller finger using the device in a cooler environment.

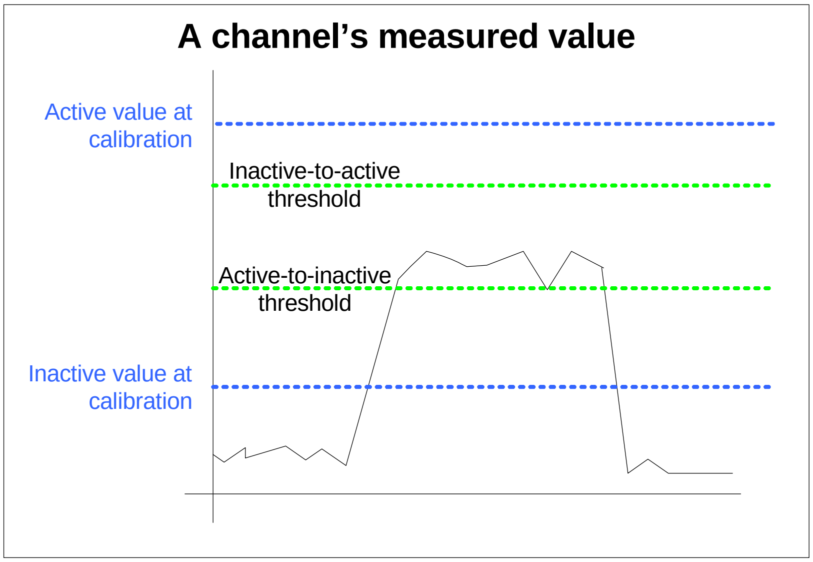

Figure 5. Channel with both a Gain Problem and an Offset Problem

Figure 3, Figure 4, and Figure 5 show cases where the channel's signal never crosses the inactive-to-active threshold whenever the channel is active. This is undesirable since the inactive-to-active threshold signifies when a channel is active to the firmware's application layer.

To compensate for these problems, the QuickSense Firmware API compares runtime baseline values to the calibrated threshold percentages and reference baseline magnitude. The runtime inactive baseline is determined at initialization. The runtime active baseline is compared against the reference baseline magnitude added to the runtime inactive baseline to determine how much the channel's gain has changed. The placement of the runtime baselines are shown in Figure 6.

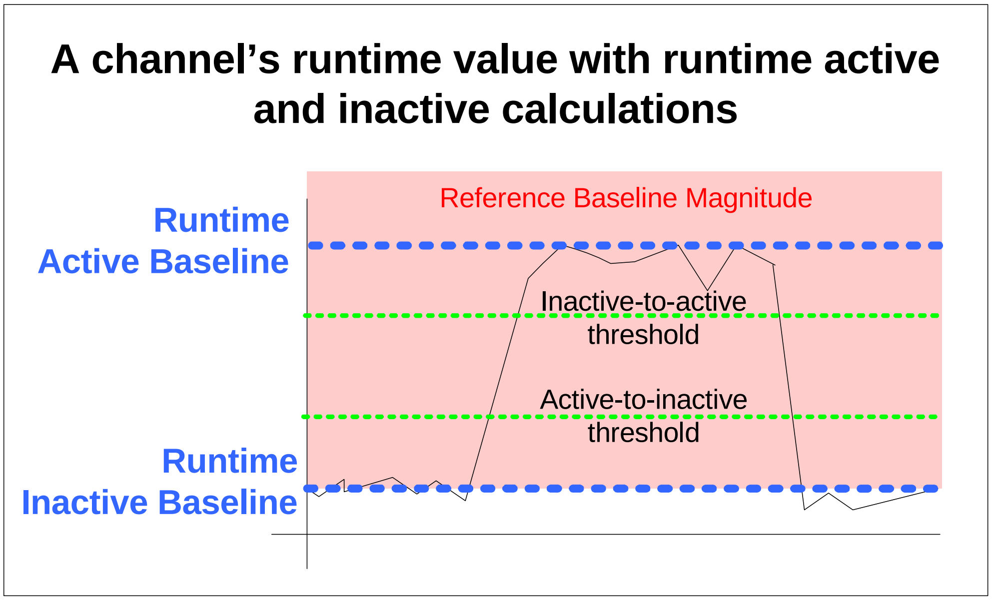

Figure 6. Runtime Baselines

The reference baseline magnitude and threshold percentages are stored in non-volatile memory., the runtime baselines are stored in an xdata array. The channel value is converted to a percentage of the channel's runtime range and then compared to the runtime thresholds. The function that scales the channel is explained in step 3 on page 8.

4. Baseline Operation

During initialization, the QS_RuntimeBaselineInit() function samples and averages the current value for each channel. The values are stored as the initial Runtime Inactive Baseline. The QuickSense Firmware API initially assumes that the gain has not changed from calibration and that the channels are initially inactive. The Runtime Active Baseline is set to be the channel's Reference Baseline Magnitude added to the Runtime Inactive Baseline. Thresholds are a percentage of the channel's runtime range. Figure 7 illustrates the runtime values as well as the threshold percentages. Variants of this figure are used throughout the rest of this section.

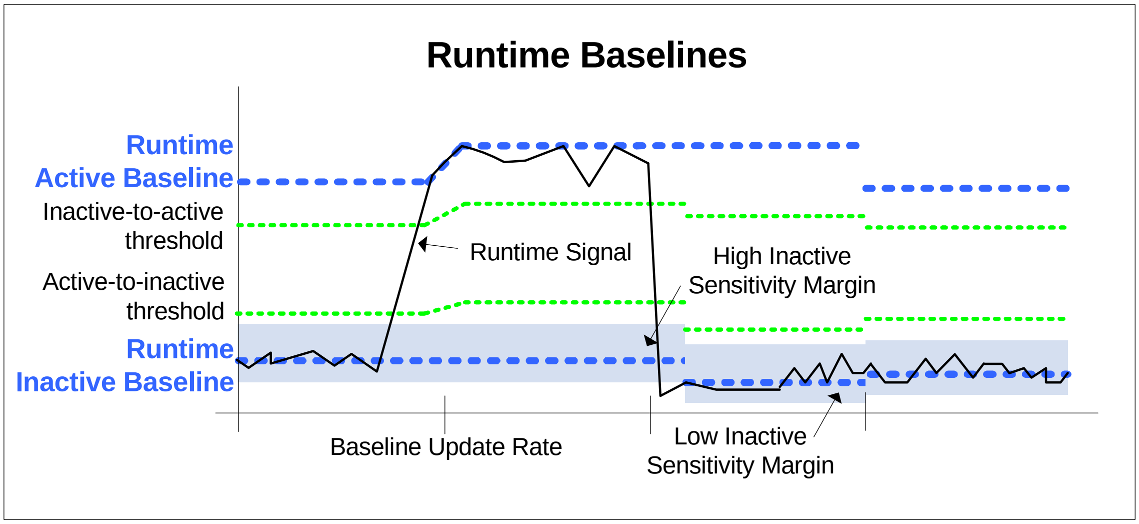

Figure 7. Runtime Baselines and Adjusted Thresholds

The blue borders around the Runtime Inactive Baseline represent sensitivity margins. There are two sensitivity margins: high inactive sensitivity margin (HISM) and low inactive sensitivity margin (LISM). If the channel's value is above the HISM, then the firmware assumes that the channel is active. The firmware uses the LISM to rapidly adjust when the inactive capacitance decreases. Both the HISM and LISM are percentages of the channel's range, defined in QS_Config.h. Additional information about the sensitivity margins can be found in "6. Configuring Baselining and Effects" on page 9.

Every time a channel is measured, firmware calls the function QS_RuntimeBaselineCheck(). This function then does the following:

- Calls QS_RuntimeBaselineInactiveCheck()—adjusts the Runtime Inactive Baseline for lower inactive values
- Calls QS_RuntimeBaselineActiveCheck()—adjusts the Runtime Active Baseline for high active values
- Checks whether channel value is above adjusted runtime active baseline and tracks both active and inactive runtime baselines upward to meet value if necessary

The tick marks on the horizontal axis represent the baseline update rate, which is configurable in units of seconds. At every baseline update interval, two functions are called:

- QS_RuntimeBaselineInactiveRefresh()—if the current value is within the sensitivity margins, the Runtime Inactive Baseline is updated
- QS_RuntimeBaselineActiveRefresh()—the Runtime Active Baseline is decayed exponentially down to a minimum configurable level

Each time the firmware uses thresholds for a calculation, the adjusted thresholds must be calculated. QS_GetBaselineCompensatedThr() uses the reference and runtime baselines to return the adjusted active-to-inactive and inactive-to-active thresholds.

The following sections describe these functions in greater detail.

4.1. QS_RuntimeBaselineInactiveCheck()

This function sets a flag when environmental changes lower the inactive capacitance. If the measured value for a channel is lower than the Runtime Inactive Baseline minus the LISM, then the firmware sets a flag to refresh the runtime baselines after approximately 100 ms. This updates the Runtime Inactive Baseline in case the inactive capacitance has shifted to a lower value. The firmware makes the assumption that a spurious noise event larger than the LISM will not last longer than 100 ms.

4.2. QS_RuntimeBaselineInactiveRefresh()

This function adjusts the Runtime Inactive Baseline as the environment changes. If the ambient temperature around the device warms up, then the channel's measured value increases. In this situation, the Runtime Inactive Baseline should increase as well. Figure 8 shows how the function adjusts the Runtime Inactive Baseline to the runtime signal.

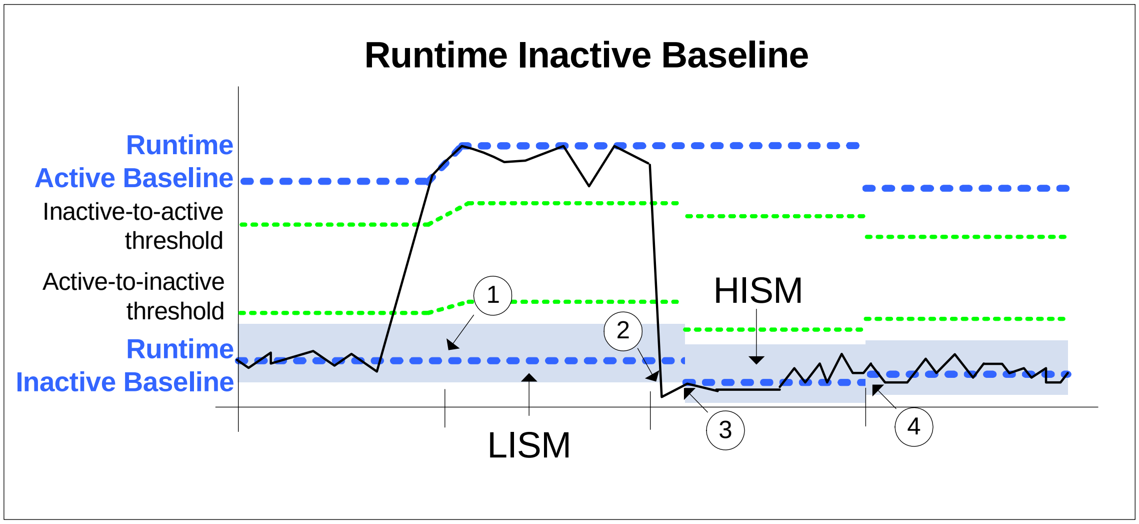

Figure 8. Runtime Inactive Baseline

Four events are illustrated in Figure 8:

1. The first scheduled runtime baseline update occurs. The channel's signal is above the HISM so the Runtime Inactive Baseline is not updated.
2. After the second scheduled runtime baseline update, the channel's signal dips below the LISM. QS_RuntimeBaselineInactiveCheck() sets a flag to update the runtime baselines.
3. After approximately 100 ms, the Runtime Inactive Baseline is updated to the current channel value.
4. The third scheduled baseline update occurs. Since the channel's value is within the HISM and LISM, the Runtime Inactive Baseline is updated to the current channel value.

4.3. QS_RuntimeBaselineActiveCheck()

This function updates the Runtime Active Baseline if the current value exceeds the Runtime Active Baseline. This makes sure that the channel's range is accurate when the channel is active. Figure 9 shows how the Runtime Active Baseline adjusts to the runtime signal.

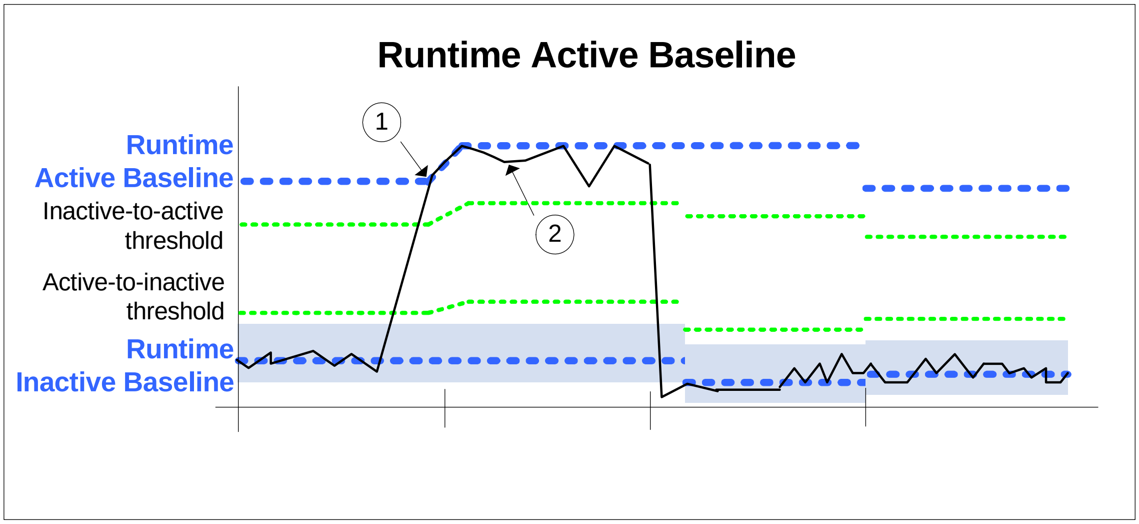

Figure 9. Runtime Active Baseline

Two events are illustrated in Figure 9:

1. The channel's signal is above the current Runtime Active Baseline, so the Runtime Active Baseline follows the channel.
2. The channel's signal dips below the Runtime Active Baseline. The Runtime Active Baseline does not follow the channel.

QS_ActiveBaselineCheck() uses the pre-compiler directive ACTIVE_BASELINE_MAX_PERCENT to restrict the upward movement of the runtime active baseline. The function only allows the active runtime baseline to rise until the difference between the active and inactive baselines equals the percentage of the reference difference define by ACTIVE_BASELINE_MAX_PERCENT. For instance, if ACTIVE_BASELINE_MAX_PERCENT is set to 150, QS_ActiveBaselineCheck() will allow the runtime active baseline to rise with samples until the condition shown in Equation 1 becomes true.

Runtime Active Baseline – Runtime Active Baseline > Reference baseline magnitude (channel) x ACTIVE_BASELINE_MAX_PERCENT

Equation 1.

If Equation 1 becomes true, the runtime active baseline is set to the highest possible value it can be set to without violating the bounds set by ACTIVE_BASELINE_MAX_PERCENT. In this case, another check on the baselines performed by QS_RuntimeBaselineCheck adjusts baselines further.

4.4. QS_RuntimeBaselineCheck

After QS_RuntimeBaselineCheck calls QS_RuntimeInactiveBaselineCheck() and QS_RuntimeActiveBaselineCheck(), it performs one final comparison on the new channel value and the newly adjusted runtime active and inactive baselines. This process is described in "6.4. ACTIVE_BASELINE_MAX_PERCENT" on page 12.

4.5. QS_RuntimeBaselineActiveRefresh()

This function is used to accommodate variable levels of gain by reducing the runtime active range, and therefore the Runtime Active Baseline. This allows the firmware to recognize that the channel is active if a person with a smaller finger uses a channel that was calibrated by a person with a larger finger.

The runtime range is decayed exponentially every baseline update if the channel is not active. The most the runtime range can decay is defined as a percentage of the reference range, and is set by ACTIVE_BASELINE_DECAY_PERCENT. This pre-compiler definition is located in QS_Config.h. Figure 10 shows the Runtime Active Baseline decaying.

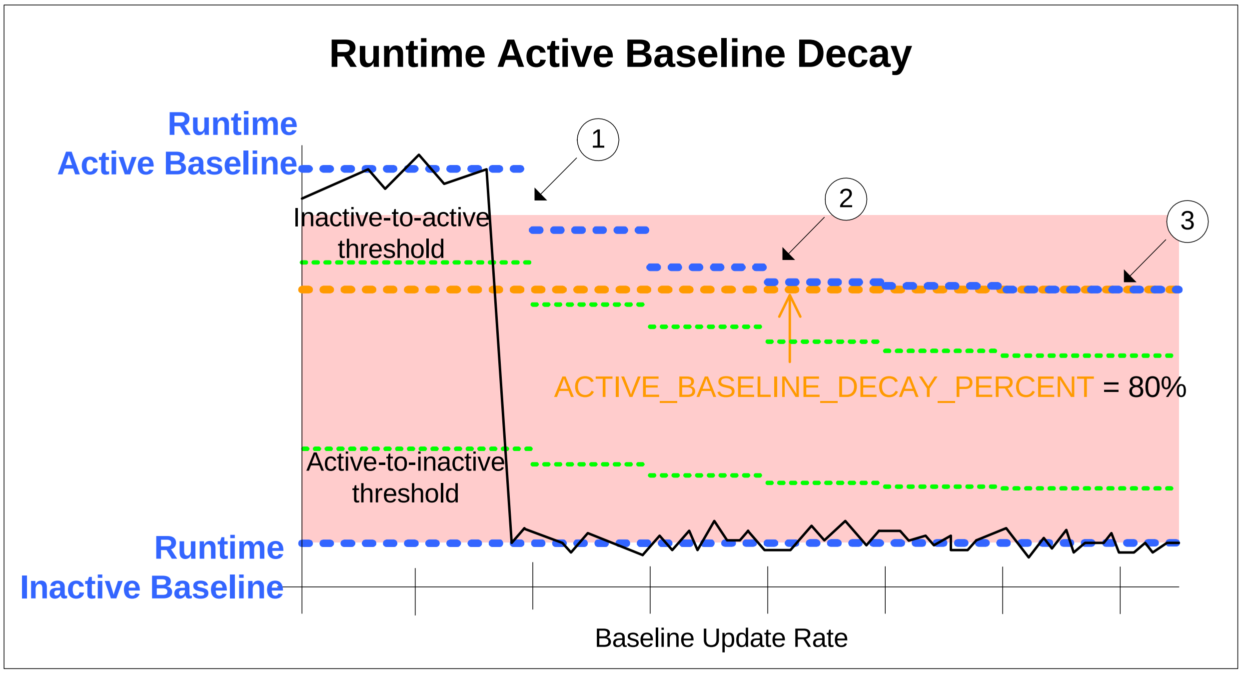

Figure 10. Runtime Active Baseline Decay

Figure 10 illustrates three events:

1. At the second runtime baseline update, the channel is inactive so the Runtime Active Baseline starts to decay.
2. At the next runtime baseline updates, the Runtime Active Baseline is decayed exponentially towards a minimum configurable value.
3. By the seventh runtime baseline update, the Runtime Active Baseline does not change since it has decayed to the minimum value.

5. Baselining and HISI

The MCU can provide baselining data through the human interface serial interface (HISI). HIS streams runtime baseline data after a host transmits a Start Transfer command. This command also initiates transfers of channel, group positions, and thresholds states.

The following are host commands that interact with baseline data stored in non-volatile memory.

GET_NVCCA	- Reads threshold percentages, reference baseline magnitude and baseline update rate
SET_NVCCA	- Writes threshold percentages, reference baseline magnitude, and baseline update rate

For more information about these commands, refer to Appendix B of AN366.

6. Configuring Baselining and Effects

There are five main configurable variables in baselining.

HISM	- Defined in QS_Config.h
LISM	- Defined in QS_Config.h
ACTIVE_BASELINE_DECAY_PERCENT	- Defined in QS_Config.h
ACTIVE_BASELINE_MAX_PERCENT	- Defined in QS_Config.h
QS_BaselineUpdateRate	- Configured through HISI

Each of these variables affects the system differently. Balancing each tradeoff helps ensure optimal system robustness to application specific conditions and changing environmental variables.

6.1. HISM

The High Inactive Sensitivity Margin is used by QS_RuntimeInactiveBaselineRefresh() to decide if a channel's Runtime Inactive Baseline should be updated. If the current channel value is above this margin, then it is assumed that the channel is active. HISM is expressed as a percentage of the channel's range.

A smaller HISM value causes the system to be less responsive to rapid changes. For instance, if the temperature in the room becomes warmer, the inactive capacitance rises. The maximum positive capacitance rate of change allowed by a system is defined by the magnitude of the HISM divided by the Baseline Update Rate. If the rate of change exceeds this value then the channel becomes active when it should be updating to the new offset value, shown in Figure 11.

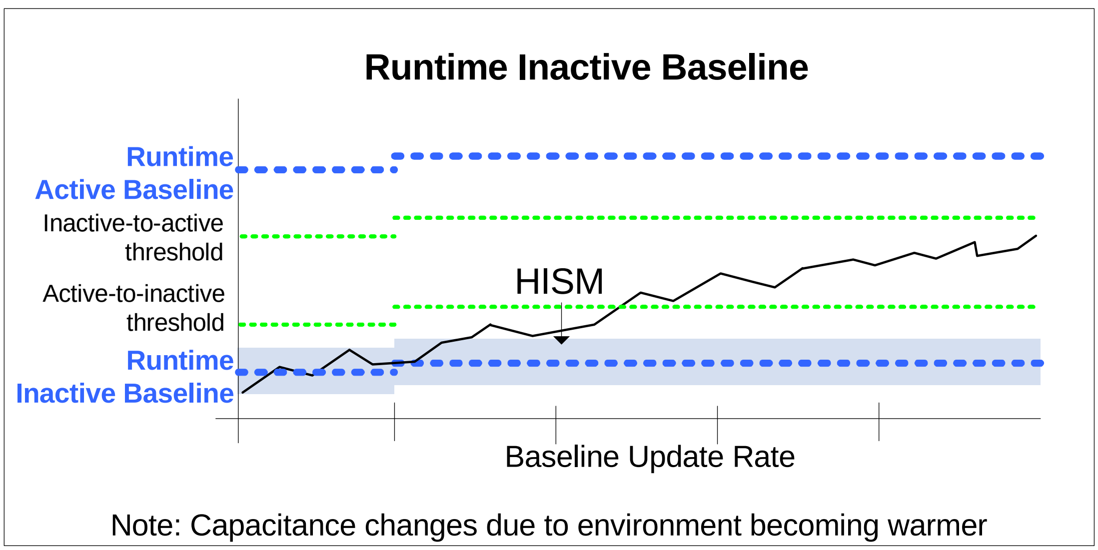

Figure 11. Runtime Inactive Baseline Unable to Compensate for Temperature Changes

Another problem occurs when a sudden drop in capacitance is sampled during a QS_RuntimeInactiveBaselineRefresh(). If the HISM value is too low, then the Runtime Inactive Baseline cannot recover to the appropriate runtime inactive value. This is illustrated in Figure 12.

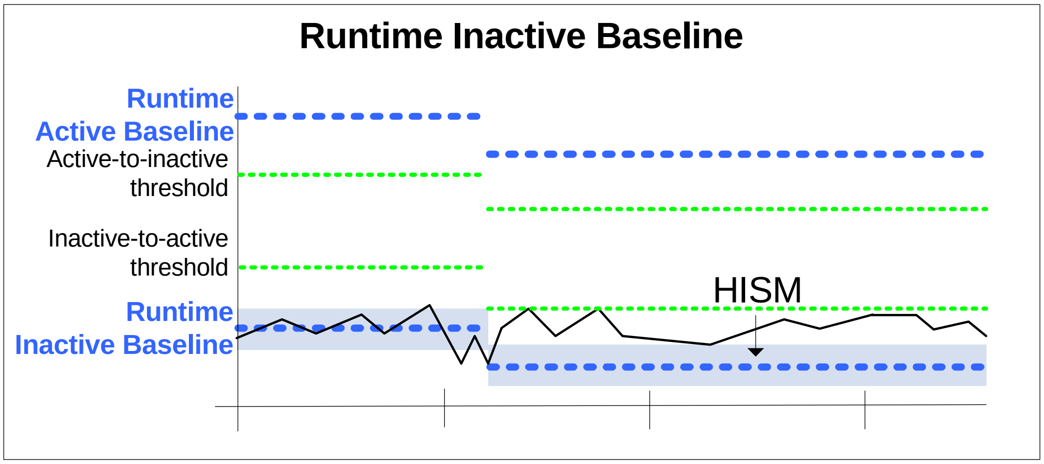

Figure 12. Runtime Inactive Baseline Unable to Compensate in Noisy Environment

A larger HISM can fix this problem, shown in Figure 14. It can also adapt to the temperature situation explained above.

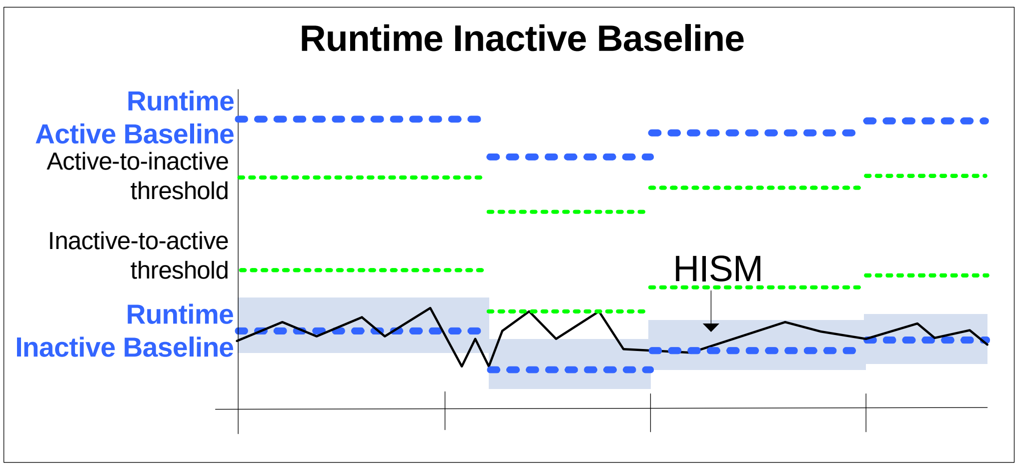

Figure 13. Runtime Inactive Baseline Compensating Successfully in Noisy Environment

The problem with a large HISM occurs whenever there is a situation that intentionally causes a slow capacitance increase. For instance, if a finger slowly approaches the channel's capacitive sensor. If the finger approaches the sensor and the channel's current value stays within the HISM, the Runtime Inactive Baseline moves towards the initial Runtime Active Baseline over time. The end result is that the finger could touch the sensor and the firmware does not recognize that the channel is active. This scenario is shown in Figure 14.

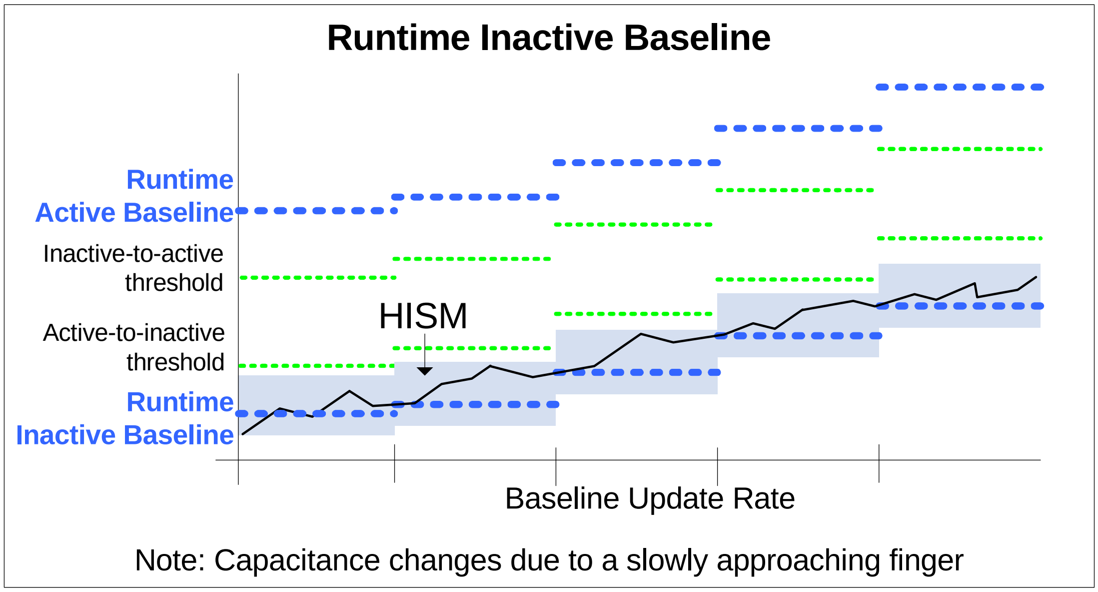

Figure 14. Updating Inactive Runtime Baseline when Channel is Active

A slower Baseline Update Rate can compensate for the issue shown in Figure 14 by requiring that the signal stay within the HISM for longer periods of time.

6.2. LISM

The Low Inactive Sensitivity Margin is used by QS_RuntimeInactiveBaselineCheck() to decide if the firmware should immediately update that channel's inactive baseline. This allows the firmware to react quickly if the inactive baseline is lowered. For example, if a hand is on the channel's sensor during initialization causing a higher initial Runtime Inactive Baseline. LISM is expressed as a percentage of the channel's range.

A smaller LISM can be a problem if there is a large amount of noise in the system. A noise spike that causes a capacitance drop can create the problem shown in Figure 12.

A larger LISM fixes this problem but is slower to react to a capacitance decrease such as temperature in the room cooling down.

6.3. ACTIVE_BASELINE_DECAY_PERCENT

The Active Baseline Decay Percent is used to address the gain problem illustrated in Figure 3 on page 2. When a channel becomes inactive, the active baseline for that channel is exponentially decayed down to the Active Baseline Decay Percent. It is the lowest value the Runtime Active Baseline can be relative to the channel's range. This allows the firmware to detect a smaller gain value when a channel is active compared to the gain value established during calibration.

A low Active Baseline Decay Percent causes the firmware to be more sensitive to sensor activity. Since the adjusted thresholds depend on the runtime active and inactive baselines, the active-to-inactive threshold could be pushed very close to the Runtime Inactive Baseline. If the firmware is running on a very noisy system, the signal can often cross the active-to-inactive threshold or cross the inactive-to-active threshold, signifying that the channel has been pressed. This scenario is shown in Figure 15.

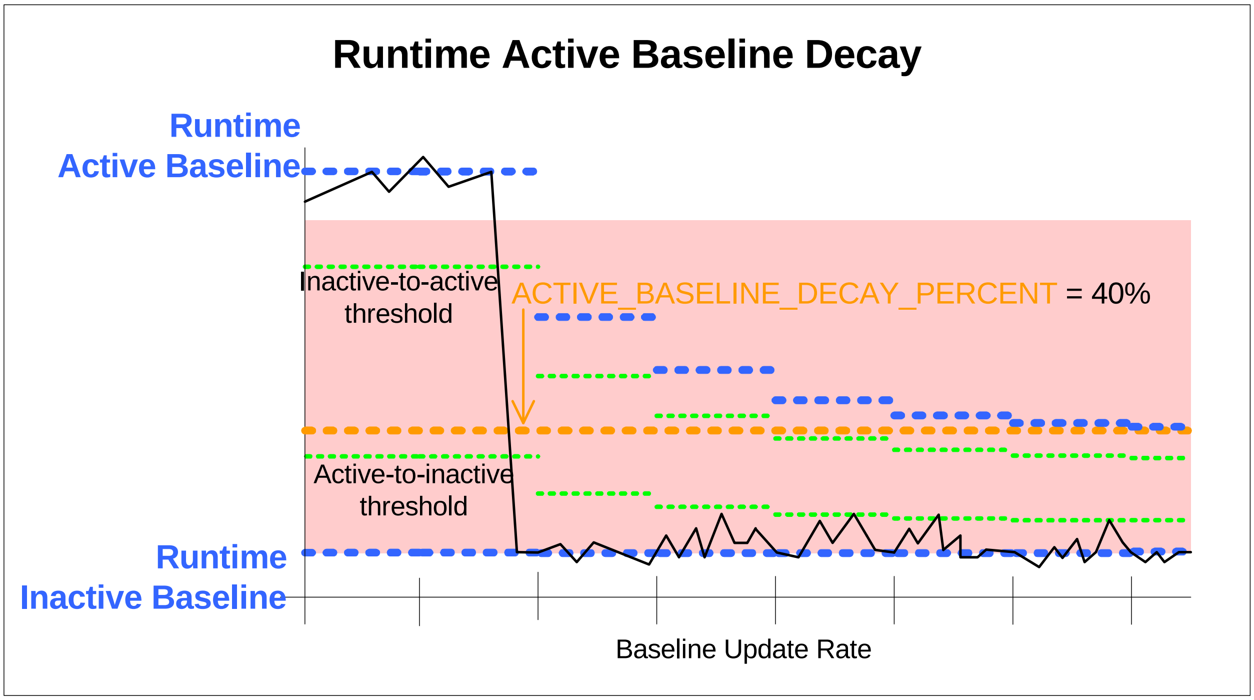

Figure 15. Low Active Baseline Decay Percent

A higher Active Baseline Decay Percent ensures that the thresholds are less affected by noise, but setting the decay percent too high makes the system insensitive to gains smaller than the gain established at calibration.

6.4. ACTIVE_BASELINE_MAX_PERCENT

The function checks to see if the channel value is above the runtime active baseline. If that is the case, then the channel value must have attempted to push the runtime active baseline above the bounds defined by ACTIVE_BASELINE_MAX_PERCENT.

In this case, the function will exponentially average the runtime active baseline upward toward the channel value. The function will also force the runtime inactive baseline to track upward, so that ACTIVE_BASELINE_MAX_PERCENT won't be violated. This process is shown in Figure 16. This behavior is the analog to the way that QS_RuntimeInactiveBaselineRefresh() will aggressively track downward when a value falls outside the bounds set by LISM.

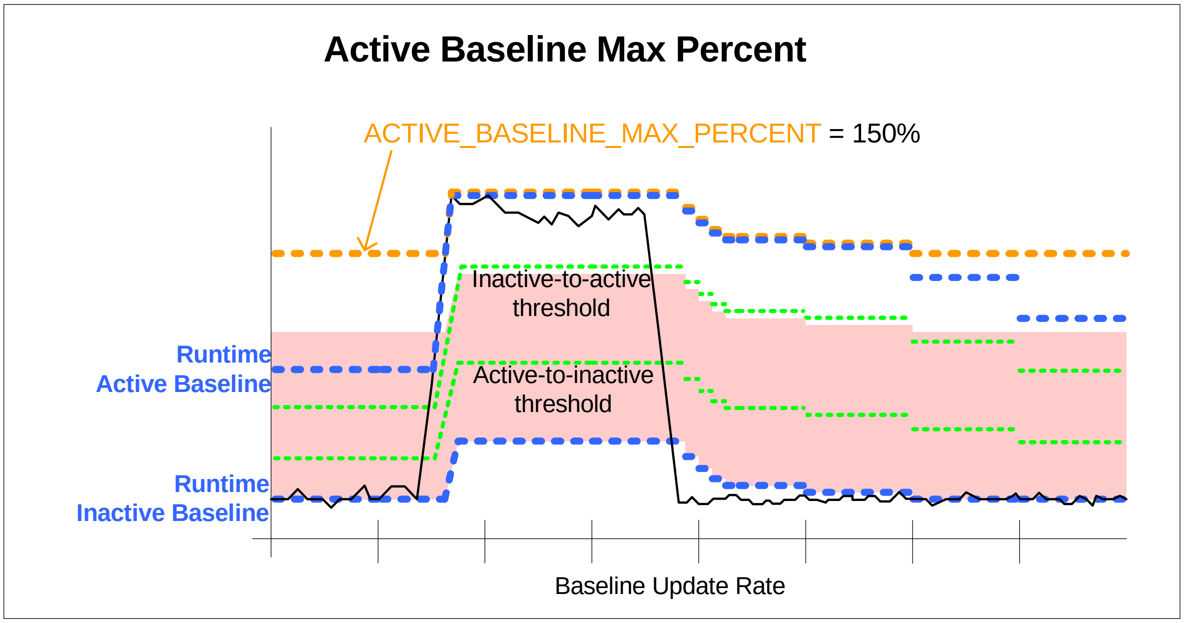

Figure 16. Active Baseline Max Percent

6.5. QS_BaselineUpdateRate

The Baseline Update Rate is a configurable variable stored in non-volatile memory. It defines the amount of time, in seconds, when the runtime baselines are updated. The baseline algorithm uses this variable as the time base for most of its operations.

A shorter Baseline Update Rate is more responsive to general capacitance changes. If the channel's inactive value is increasing due to temperature, it is easier to adjust the inactive baseline by sampling more often. However, a problem occurs if a slowly approaching finger causes the channel’s value to rise but stay within the HISM threshold during the shorter baseline update rate, as shown in Figure 14. It is important for a system’s configuration to be tuned so that the HISM allows for optimal sensitivity while the Baseline Update Rate’s period is long enough to avoid inadvertently updating the Runtime Inactive Baseline while a finger or other object affects the channel’s measured capacitance.

A longer Baseline Update Rate is less responsive to noise. The tradeoff is that the firmware takes a longer time to update the runtime baselines to the correct runtime value.

Document Change List

Revision 0.1 to Revision 0.2

- Updated figures to reflect QuickSense Firmware API 2.x functionality
- Added description of RUNTIME_ACTIVE_MAX_PERCENT
- Added references to proximity/ambient light sensing devices

NOTES:

Contact Information

Silicon Laboratories Inc.
400 West Cesar Chavez
Austin, TX 78701
Tel: 1+(512) 416-8500
Fax: 1+(512) 416-9669
Toll Free: 1+(877) 444-3032

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.