Application Report
SWRA362B–June 2011–Revised December 2014

RemoTI™ Capacitive Touch Sensing

Low-Power Wireless Products

ABSTRACT

As consumers become accustomed to elegant touch interfaces on a variety of devices, these capabilities are becoming more expected on a greater number of devices. The CC253x devices used in RF4CE applications provide hardware support for Capacitive Touch Sensing (Capacitive Sensing), and enable RF4CE manufacturers to design a wider range of devices with Capacitive Sensing capabilities using a cost-effective approach.

This application report provides information about how to use this hardware capability in the RemoTI software environment as well as other user-selected designs.

Project collateral and source code discussed in this application report can be downloaded from the following URL: http://www.ti.com/lit/zip/swra362.

Contents

List of Figures

List of Tables

1 Introduction to Capacitive Sensing

NOTE: RemoTI is Texas Instrument's implementation of the ZigBee® RF4CE network protocol standard. Additional information about RF4CE is available at www.zigbee.org/rf4ce; information about RemoTI is located at www.ti.com/RemoTI.

Capacitive Sensing, or *Capacitive Touch Sensing*, is based on the concept of measuring the altered capacitance of a sensor or touch pad in the presence of a pressure source, such as a finger. Figure 1 illustrates this principle.

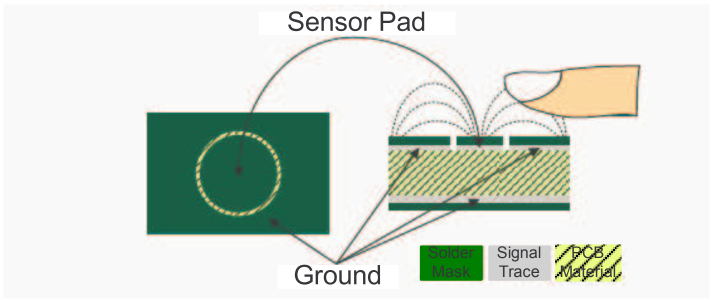

Figure 1. Capacitive Sensing Principle

The capacitance can be found by measuring the rising and falling time when charging and discharging the capacitor. The measurement is done by starting a timer at the instance of charging and discharging, and then capturing the event when the capacitor is charged and discharged. Note that there is more than one way to connect the capacitor to the GPIO ports of the chip.

The CC253x family of RF system-on-chip devices contain hardware features that enable Capacitive Sensing functionality. The following resources are available:

- Timers:
  - Timer 1 (16-bit, five channels)
  - Timer 3 (8-bit, two channels)
  - Timer 4 (8-bit, two channels)
- Individually-configurable GPIOs associated to timer channels

The different buttons are connected to a dedicated GPIO for control of the charging/discharging, while the associated pin in the next phase is configured as a peripheral and operates as an input for the Timer 1 capture mode. There are five channels available on Timer 1 that allow up to five buttons to operate in parallel. More channels are available on Timer 3 and Timer 4, but with less accurate performance. The usage of Timer 3 and 4 is not encouraged and this report will focus on Timer 1 usage.

Timer 1 has five channels, but these channels can be mapped to GPIOs in two different ways. This approach allows up to nine different GPIOs that service one button each; see the extract from SWRU191 (Ref. 3) in Table 1. For even more channels and a more comprehensive system, see also Reference 7.

Table 1. Peripheral I/O Pin Mapping, TIMER 1

Periphery/ Function	P0								P1								P2				
	7	**6**	**5**	**4**	**3**	**2**	**1**	**0**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**	**4**	**3**	**2**	**1**	**0**
TIMER 1		4	3	2	1	0															
Alt. 2	3	4												0	1	2					

In our implementation, two pairs of buttons share a resistor to reduce the overall part count. Figure 2 illustrates the relevant schematic portion; see also Reference 5.

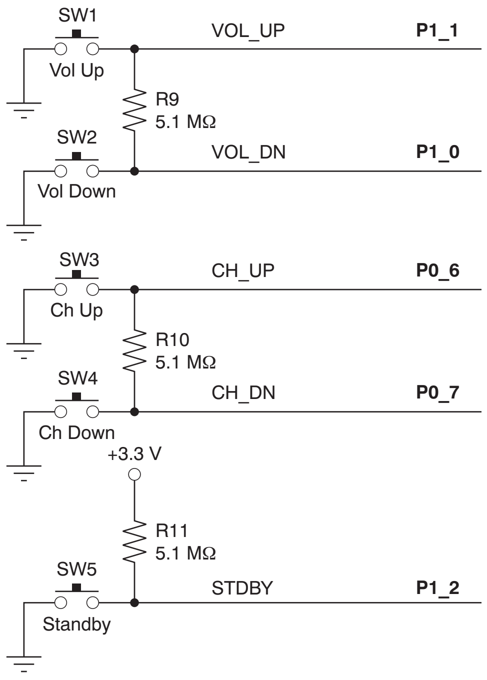

Figure 2. Part of Schematic Showing Capacitive Buttons SW1 to SW5

Figure 3 shows the voltage level at each pin and the complete configuration. Table 2 and Table 3 list the pin states and timer allocation for Figure 3, respectively .

Table 2. Pin States

State	Control	Direction	Level	Input Mode
A	GPIO	Output	Low	—
B	GPIO	Output	High	—
C	Peripheral: Timer1	Input	—	3-state

Table 3. Timer Allocation

State	Description
T0	Assigned to Timer 1 Channel 0
T1	Assigned to Timer 1 Channel 1
T2	Assigned to Timer 1 Channel 2
T3	Assigned to Timer 1 Channel 3
T4	Assigned to Timer 1 Channel 4

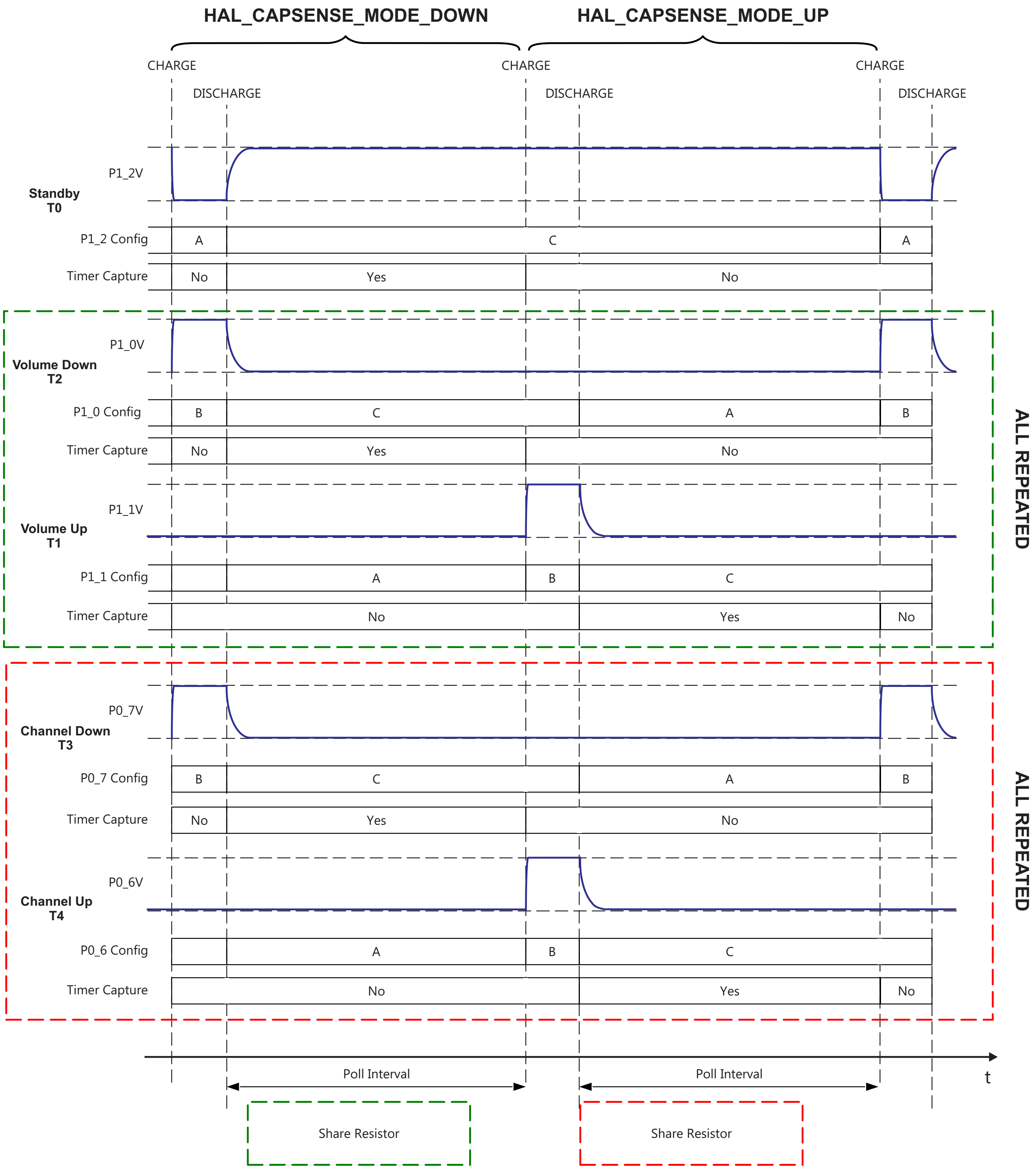

Figure 3. Pin Configuration and Voltage Levels vs Time

1.1 Setting Up the Timer

How the timer is set up defines the precision and range of operation. When a pressure source such as a finger is present, the capacitance decreases differently depending on the distance of the finger from the screen, the geometric properties of the finger, and whether or not the finger actually touches the screen. It is good to have an understanding of the physical properties of capacitors; keep in mind that capacitance is a physical property that depends on the geometry of the conductors and on the conductivity of the medium between them. If the finger touches the surface of a button, the dielectric medium no longer includes as much air, but instead only tissue and likely plastic. Interested readers are encouraged to read Chapter 3 of Reference 1.

As a consequence, it is important to design the buttons such that false detection is avoided. The reference design presented here uses circular conductors to represent the area of a singular button. This design gives the best performance for discrete behavior. In other words, it has the least field leakage, which means that presence of a touch at one button is less likely to affect nearby buttons. It does, however, mean that this design is not useful for proximity detection.

It is quite difficult to measure capacitance without affecting the measurements. To combat this problem, a coarse estimate is used as a baseline for setting up the timer; it is then configured and tracked in real time and adjusted accordingly. For our implementation, the baseline rise/fall time is measured in the area of 25 µs to 75 µs. Thus, each tick on the timer should correspond to this magnitude. However, it is also important to be able to measure much greater rise/fall times in the presence of a pressure source such as a finger. A tick speed of 2 MHz gives two ticks per microsecond. With a 16-bit timer, this speed allows up to 65,536 ticks, or 32.768 ms. This number of ticks is much more than necessary, but reduces the likelihood of a timer overflow. (An overflow could be easily handled, but that would add unnecessary complexity to our design.)

To obtain a tick speed of 2 MHz on the CC253x devices, set the global prescaler for Timer 1, Timer 3, and Timer 4 to 001_b.

`CLKCONCMD.TICKSPD` = 001_b

This value gives a timer tick speed of 16 MHz, given a clock source that is greater than or equal to 16 MHz (see Section 9.1 of Ref. 3). The Timer 1 prescaler is set to 01_b, or a division by 8.

`T1CTL.DIV` = 01_b

This configuration gives a tick speed for Timer 1 of 2 MHz.

Figure 4 compares the effects on rise time with and without a touch. t_1, in blue, is without touch pressure; t_2, in red, is with touch pressure.

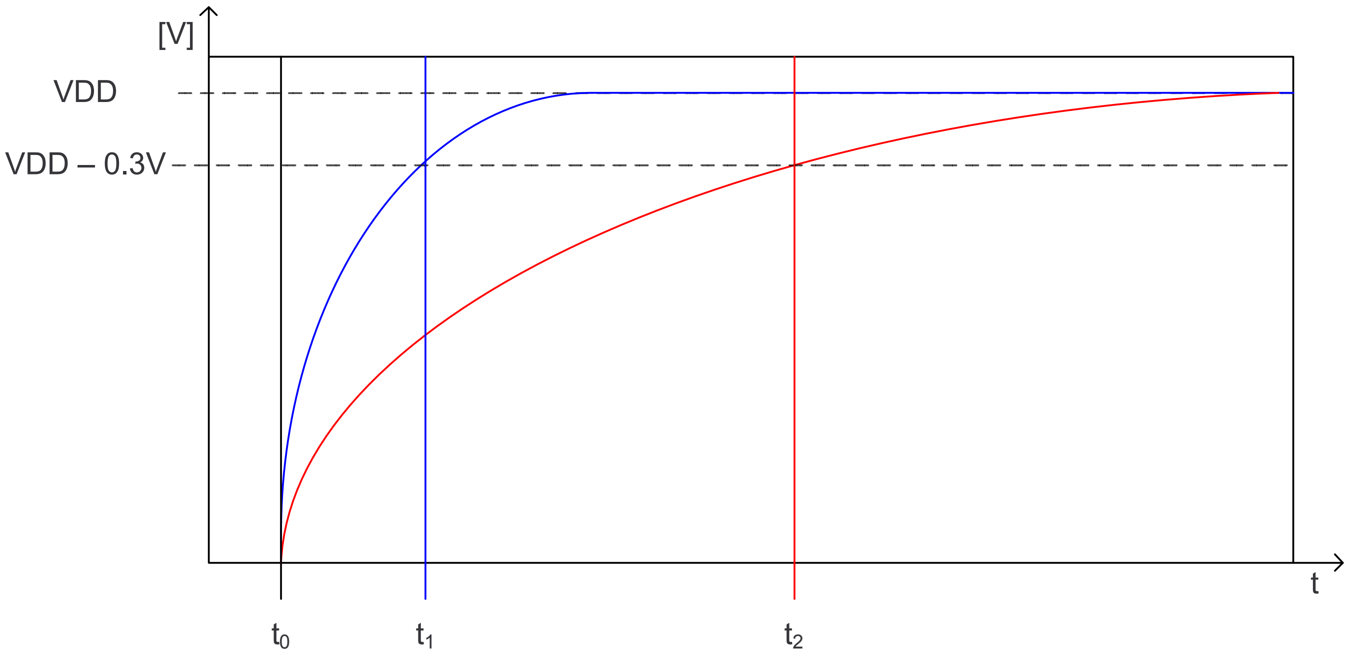

Figure 4. Rise Time With (t_2) and Without (t_1) Touch

One sequence of charging, or discharging, is repeated every `HAL_CAPSENSE_POLLING_VALUE` ms by design; refer to Figure 3 and Table 4. This timing allows for detection of touches that decrease the capacitance such that the rise/fall time can be up to `HAL_CAPSENSE_POLLING_VALUE` ms. Because Timer 1 has five channels, five capacitors can be measured during this time span. Depending on how the capacitors are connected, another set may be measured in the next sequence. How many such sequences are allowed depends on the number of GPIOs available and on the user experience. A typical button press must last for at least the period of two detection sequences plus the number of different sets of capacitors.

1.2 Detecting a Touch

There are two primary considerations when we approach the design: the definition of a defined touch and the definition of a valid sample.

A touch is registered by the device if the measured capacitance exceeds the base capacitance by at least a defined threshold: `HAL_CAPSENSE_THRESHOLD_MIN_DETECT`. In this implementation, we track the mean rise time, µt, and the variance, $\sigma^2{}_t$, by a running mean and variance algorithm. Therefore, we can dynamically (and more precisely) define the threshold.

We have set the threshold, $\Delta t_{threshold}$, to the maximum of `HAL_CAPSENSE_THRESHOLD_MIN_DETECT` and `HAL_CAPSENSE_THRESHOLD_MULTIPLIER_DETECT` times the variance; that is,

$$\Delta t_{threshold} = \max(\texttt{HAL\_CAPSENSE\_THRESHOLD\_MIN\_DETECT}, \texttt{HAL\_CAPSENSE\_THRESHOLD\_MULTIPLIER\_DETECT} \bullet \sigma^2{}_t) \quad (1)$$

The second (and very important) concern is when a sample should count as a valid sample. We have defined that a sample is valid whenever it is within the limits of the max (`HAL_CAPSENSE_THRESHOLD_MIN`, `HAL_CAPSENSE_THRESHOLD_MULTIPLIER` $\bullet \sigma^2{}_t$) from the mean μ_t. This period is denoted as **b** in Figure 5. For **a**, in Figure 5, statistics are updated with $\mu_t - 2\sigma^2{}_t$ as the sample value.

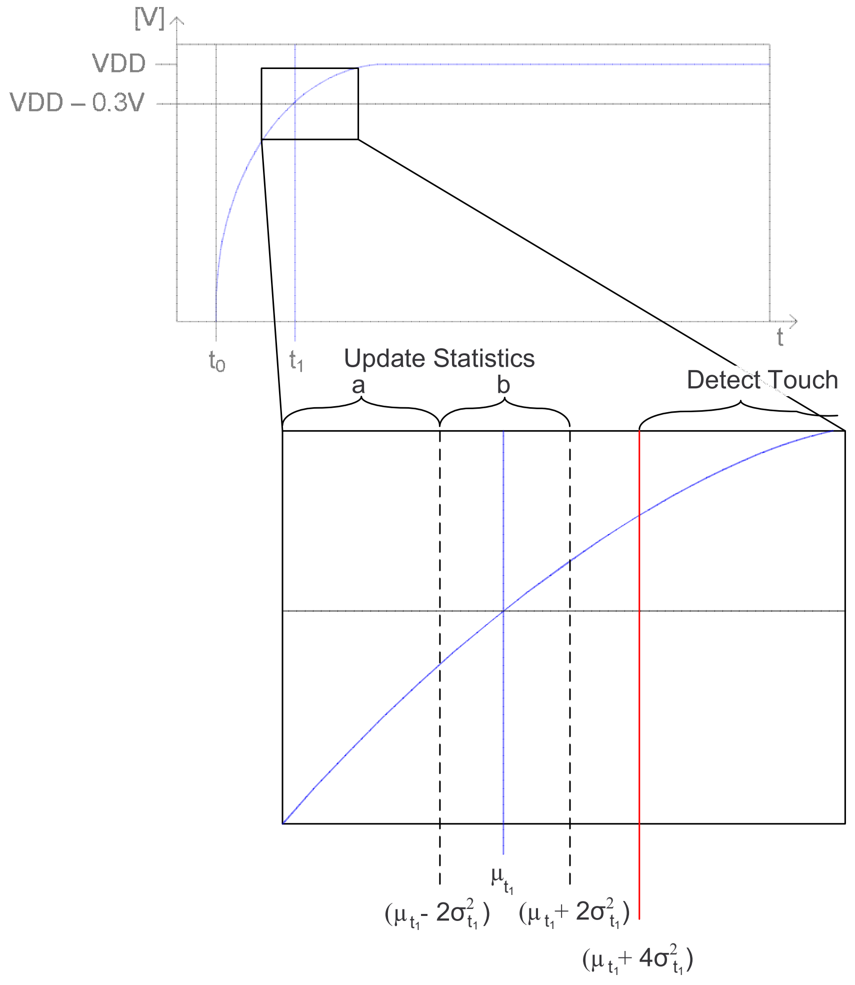
[V]
VDD
VDD – 0.3V
t
t_0
t_1
Update Statistics
a
b
Detect Touch
μ_{t_1}
$(\mu_{t_1} - 2\sigma^2_{t_1})$
$(\mu_{t_1} + 2\sigma^2_{t_1})$
$(\mu_{t_1} + 4\sigma^2_{t_1})$

Figure 5. Detection and Statistics Update Ranges

2 Capacitive Sensing Application

This section describes the software implementation for both the Capacitive Sensing driver and the integration with the RemoTI stack.

2.1 IAR Workspace

The standalone project is preconfigured with a new target: ***CC2531_FrontPanel***. Three configurations exist:

- CC2531F256
- CC2531F256-HEX
- CC2531F256_SB

CC2531F256 is used for debugging, CC2531F256-HEX builds a .hex image that can be downloaded via the SmartRF Flash Programmer, and CC2531F256_SB invokes a post-build tool that prepends the Serial Bootloader to the binary in a .hex image.

To enable the output of statistics over the UART, be sure to build with UART_STATS defined.

2.2 Architecture

The Capacitive Sensing hardware driver is written similar to most RF4CE modules, as shown in Figure 6. The application calls an initialization routine, `HalCapSenseInit()`, as well as subscribes to the function by calling `HalCapSenseConfigure(SA_CapSenseCback)`. In this case, `SA_CapSenseCback` is the function that the application would like the HalCapSense module to call in when detecting a touch.

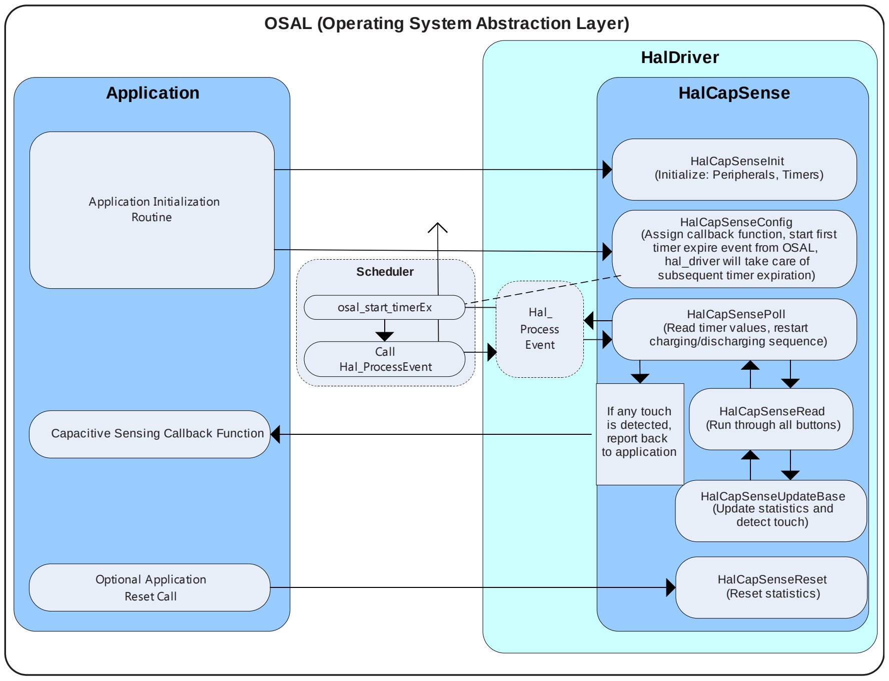
OSAL (Operating System Abstraction Layer)
HalDriver
Application
HalCapSense
HalCapSenseInit (Initialize: Peripherals, Timers)
Application Initialization Routine
HalCapSenseConfig (Assign callback function, start first timer expire event from OSAL, hal_driver will take care of subsequent timer expiration)
Scheduler
osal_start_timerEx
Call Hal_ProcessEvent
Hal_ Process Event
HalCapSensePoll (Read timer values, restart charging/discharging sequence)
If any touch is detected, report back to application
HalCapSenseRead (Run through all buttons)
Capacitive Sensing Callback Function
HalCapSenseUpdateBase (Update statistics and detect touch)
Optional Application Reset Call
HalCapSenseReset (Reset statistics)

Figure 6. Capacitive Sensing Integration Architecture

During runtime, the HalDriver calls `osal_start_timerEx()` on behalf of the HalCapSense module. It also receives the *timer expired* event from OSAL, and subsequently calls the polling routine of HalCapSense, `HalCapSensePoll()`.

From an application point of view, the actual operation of HalCapSense is hidden. Once configured, the application receives a callback only if a touch is detected. Because touch detection is binary *(present/not present)*, the module itself does not notify of a release. There are many parameters that should be configured to optimize performance; refer to Table 4.

If for some reason the application wants to reset the operation of HalCapSense, it may call `HalCapSenseReset()`. This call sets all buttons to an unconfigured state and resets the statistics for each button. This call is equivalent to a power-on-reset from the perspective of the HalCapSense. See Table 4 for a more detailed scenario.

2.3 Optimizing Performance

This section addresses the most important issue: how to configure the operation of detecting a touch, establishing what parameters can be modified, and defining what the effects are. Table 4 provides a list of parameters that can be modified. Note that these parameters are recommended only when a tick speed of 2 MHz is used.

Table 4. List of Parameters to Configure HalCapSense

Macro	Recommended Value (Given 2-MHz Ticks)	Description
`HAL_CAPSENSE_ADV_TRACK_MAX_COUNT`	**200**	Number of samples used before updating the variance.
`HAL_CAPSENSE_THRESHOLD_MIN`	**5**	A minimum threshold in number of ticks.
`HAL_CAPSENSE_THRESHOLD_MIN_DETECT`	**10**	A minimum threshold for detection in number of ticks.
`HAL_CAPSENSE_DEBOUNCE_VALUE_SHORT`	**61**	Short software debounce in ms. Implemented and used by application, not by the HalCapSense module itself.
`HAL_CAPSENSE_DEBOUNCE_VALUE_LONG`	**333**	Long software debounce in ms. Implemented and used by application, not by the HalCapSense module itself.
`HAL_CAPSENSE_THRESHOLD_MULTIPLIER`	**2**	Threshold is equal to this number times $\sigma^2{}_t$, if that product is greater than `HAL_CAPSENSE_THRESHOLD_MIN`.
`HAL_CAPSENSE_THRESHOLD_MULTIPLIER_DETECT`	**4**	Threshold for detection is equal to this number times $\sigma^2{}_t$, if that product is greater than `HAL_CAPSENSE_THRESHOLD_MIN_DETECT`.
`HAL_CAPSENSE_POLLING_VALUE`	**16**	Polling interval in ms.
`HAL_CAPSENSE_NOF_SETTLING_ITERATIONS`	**0**	If there is a need for some iterations to settle, this parameter allows these iterations.
`HAL_CAPSENSE_RESET_PERIOD`	**666**	Optional reset period, used from application to reset the statistics.

The running mean and average algorithms are sample-based algorithms. `HAL_CAPSENSE_ADV_TRACK_MAX_COUNT` sets the number of samples required before the variance is stored and used for detection; see *Detection Range* in Figure 5, Figure 7, and Figure 8. When the variance is updated it is *averaged* in. This calculation is not a true averaging, because the latest contributions are weighted equally as all the earlier ones.

$$\sigma^2_{old,k+1} = \frac{\sigma^2_{old,k} + \sigma^2_{new}}{2} \quad (2)$$

When all buttons are left untouched, the variance is typically 0, given only two ticks-per-µs resolution. However, when a finger is present at one button, it may affect some other button. To help prevent false detection at the other button(s), the variance may typically increase and the decision range exceeds the default minimum. One can disable this feature altogether by setting `HAL_CAPSENSE_THRESHOLD_MULTIPLIER(_DETECT)` to 0. The decision range is then fixed and given by `HAL_CAPSENSE_THRESHOLD_MIN(_DETECT)`.

2.3.1 Remedying Failing Scenarios

There are two scenarios which can make the Capacitive Sensing misbehave. This section describes the two scenarios and suggests how to remedy the situation.

Both scenarios relate to the actual rise/fall time and the tracking of the respective event. The preceding sections explain the tracking of the rise/fall times. There are situations, however, that are difficult to track. If something is left on the capacitor after touching it, the new actual rise/fall time will change abruptly; as a result, the new samples will be outside of the range that is tracked (see Figure 5). Now, all future samples will be detected as *touches*. This scenario is shown in Figure 7.

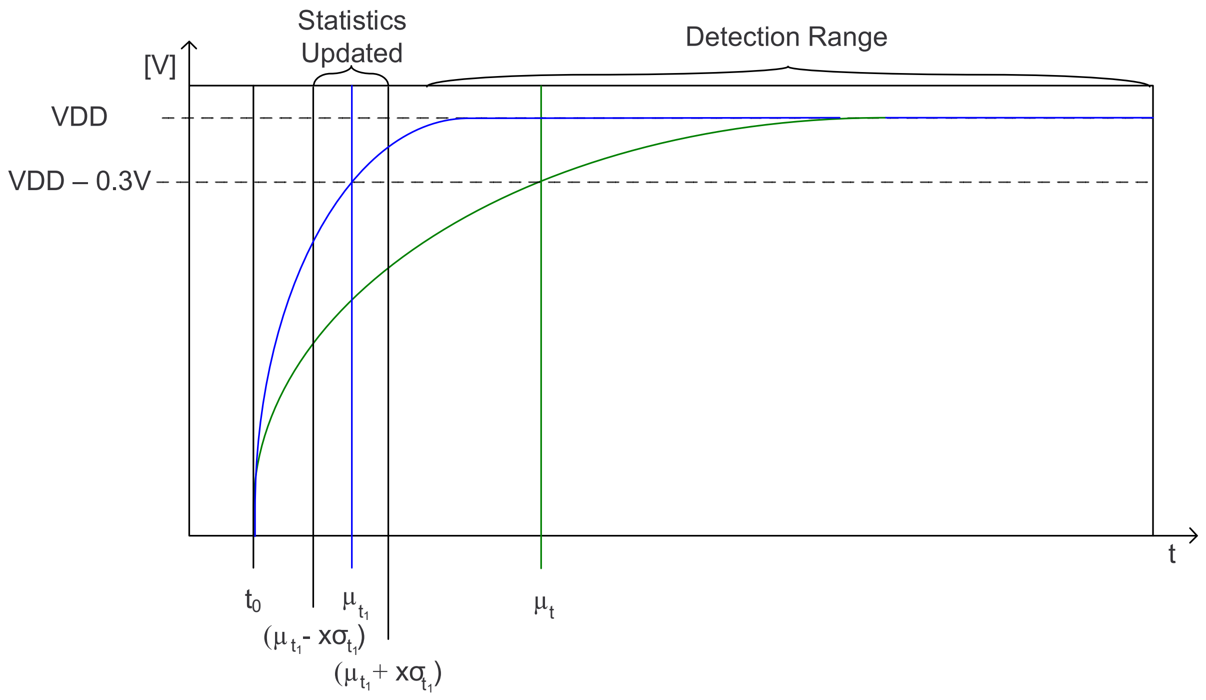

Figure 7. Detection Fails: Statistics Are Not Updated While the True Base Increases

Another scenario can occur if the ranges where statistics are updated and where a touch is detected are not well defined. As a result, touches on nearby buttons may cause the tracking to increase such that the range where statistics are updated cover a typical touch; see Figure 8. Now, a typical touch will not be detected.

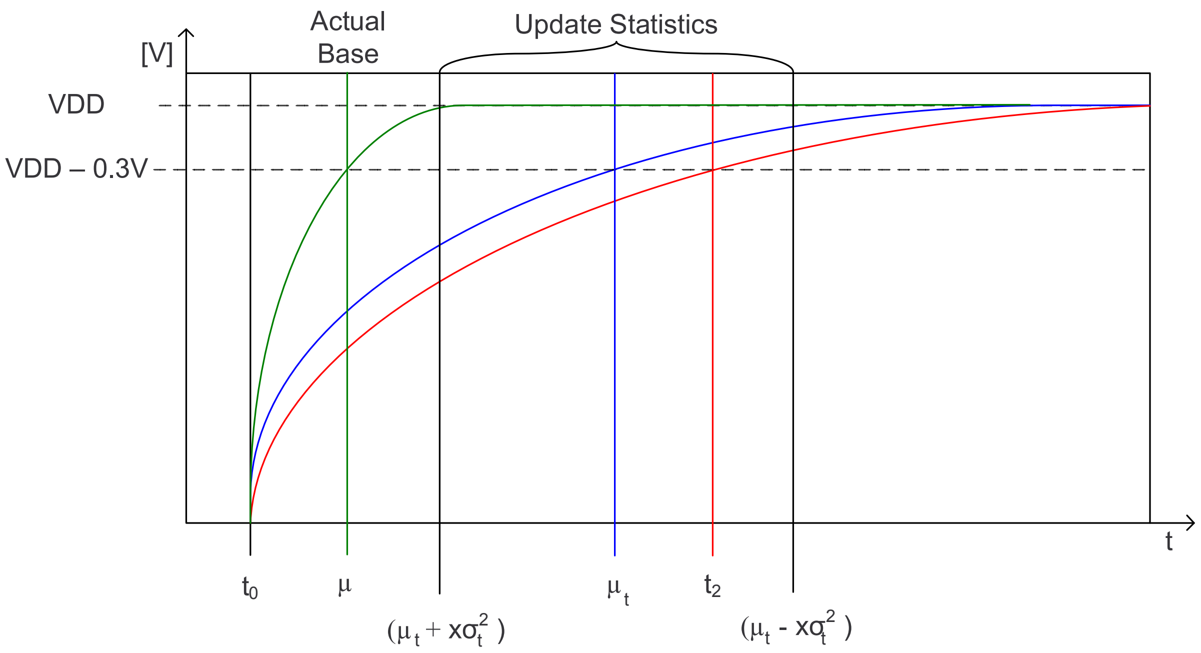

Figure 8. Detection Fails: Statistics Are Tracked Off the True Base

The `HAL_CAPSENSE_RESET_PERIOD` can be used as a self-repair mechanism. It is used in the sample application after the last touch has debounced. Thus, when the application believes there is *no touch*, it resets the HalCapSense module. This call is useful to recover quickly from the second scenario explained; see Figure 7. The device will also recover from this scenario if the button is left untouched for a period of time, because of the downwards-tracking shown as **a** in Figure 5.

It also allows the user to recover from the first scenario explained; where touches are always detected. The application would have to define a constraint to the length in time of a continuous touch, and issue a reset if this constraint has been exceeded. Note that this use case has not been implemented in the sample application. It is left up to the designer to select how long a series of detected button presses can be.

2.4 Application Functionality

This application is based on the RNP project for the CC2531, and includes the RTIS, RemoTI Surrogate layer. See Section 4.4 of Reference 2 for a discussion of RNP operation via a virtual serial port over a USB interface.

Apart from the RNP functionality, the LEDs are toggled when a touch is registered. The buttons are organized to reflect a typical usage as a front panel, as shown in Figure 9. To further reflect the typical use of the buttons, they have different software debounce times, as Table 5 summarizes. Figure 10 shows the reverse side of the board.

Table 5. Buttons and Respective Software Debounce Times

Button	Macro	Debounce Time (ms)	LED
Standby	HAL_CAPSENSE_DEBOUNCE_VALUE_LONG	333	4
Volume Up	HAL_CAPSENSE_DEBOUNCE_VALUE_LONG	333	1
Volume Down	HAL_CAPSENSE_DEBOUNCE_VALUE_LONG	333	2
Channel Up	HAL_CAPSENSE_DEBOUNCE_VALUE_LONG	333	1
Channel Down	HAL_CAPSENSE_DEBOUNCE_VALUE_LONG	333	2

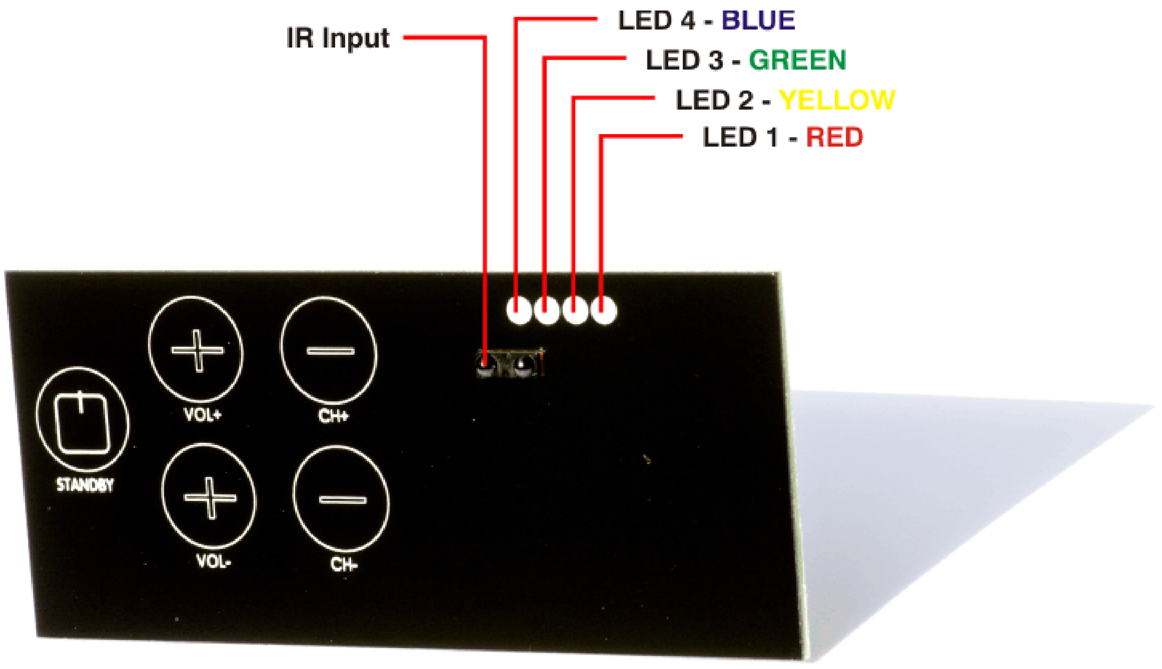

Figure 9. Front Panel Reference Design: Front

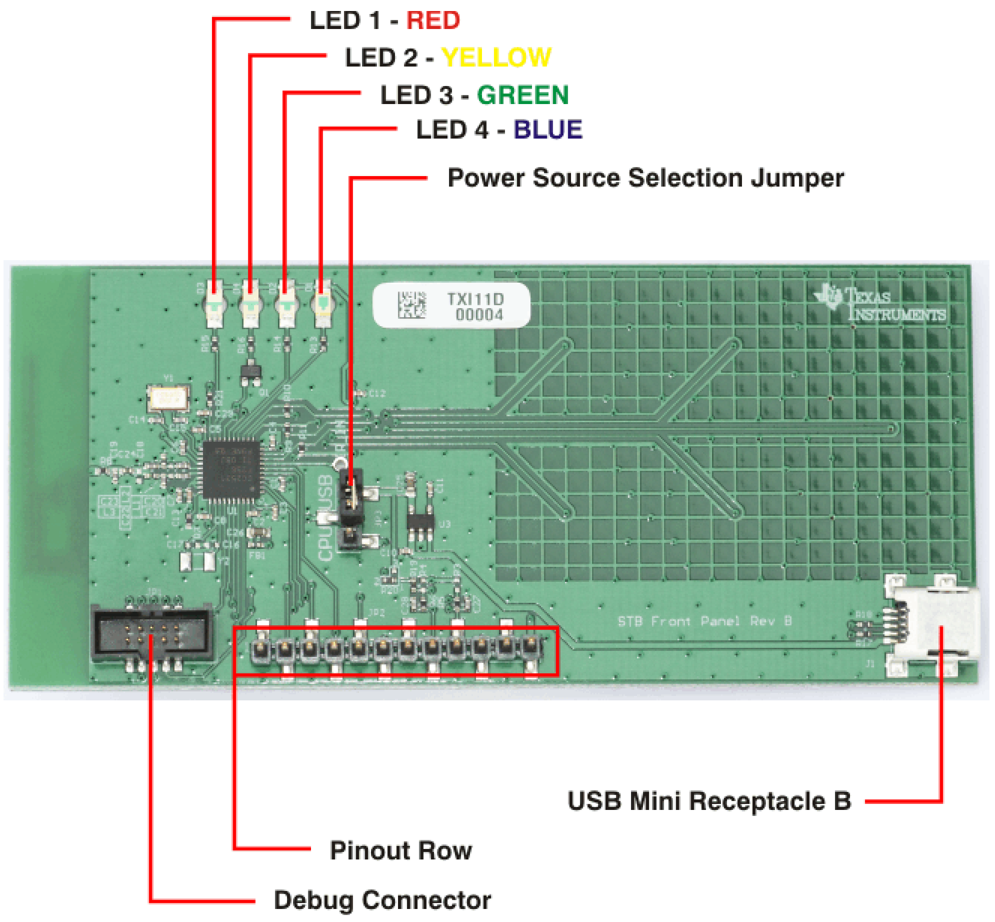

Figure 10. Front Panel Reference Design: Back

2.5 Execution Sequence

Step 1. Build the project.

Step 2. Connect the mini-USB cable to the front panel to apply power.

Step 3. Connect the CC Debugger and download the image.

Step 4. Execution of the CapSenseApp begins immediately.

2.6 Additional Help for Improving User Experience

If the project is built with UART_STATS defined, a simple UART output is set up in this manner: flow control disabled, 8-bit transfer enabled, one stop bit is set (high), and the start bit is low.

Simply strap a cable from Pin 3 on the front panel reference design (see Ref. 5; see also Figure 10) to P1_7 on a SmartRF05 Evaluation Board, for example (see Ref. 6), and then connect this board to a serial port on a computer.

What is written to the UART depends on what is detected; denote charge time *t*, then refer to Table 6 for a list of possible outputs over the UART.

Table 6. Data Written to UART and When Written

Statement		1 byte	4 bytes	4 bytes	4 bytes
$\Delta t > \Delta t_{threshold\_detect}$ [1]	Touch detected	x [2]	t_x	μ_{tx}	σ^2_{tx}
$\Delta t < -\Delta t_{threshold}$ [3]	Tracking down	x [2]	$\Delta t = t_x - \mu_t$	μ_{tx}	σ^2_{tx}
Every HAL_CAPSENSE_ADV_TRACK_MAX_COUNT samples	Send data for all buttons.	0	t_0	μ_{t0}	σ^2_{t0}
		1	t_1	μ_{t1}	σ^2_{t1}
		2	t_2	μ_{t2}	σ^2_{t2}
		3	t_3	μ_{t3}	σ^2_{t3}
		4	t_4	μ_{t4}	σ^2_{t4}

(1) $\Delta t_{threshold}$ = max (5 µs, 2 × σ^2_t µs)
(2) **x** represent the detected button index
(3) $\Delta t_{threshold\_detect}$ = max (2.5 µs, 4 × σ^2_t µs)

4 bytes are transmitted per 16-bit value; 1 byte represents four bits as an ASCII character .

This information could help users choose the correct values in Table 4 for a specific implementation.

```
for(I=3; I > -1; I--) {
    temp = btnBaseCapMeanOld[btnId] >> (4*I);
    temp &= 0x0F;
    if(temp < 10) uartPutc((temp + '0'));
    else uartPutc(((temp - 10) + 'A'));
}
```

3 Integrating Capacitive Sensing in Custom Solution

To add Capacitive Sensing capabilities to your custom solution, simply add the `HAL_CAPSENSE` module to your project. Make sure there is no conflict with the Timer 1 usage; otherwise, all timer-related macros must be redefined. The polling routine `HalCapSensePoll()` calls the macros `CAP_DISCHARGE(mode)` and `CAP_CHARGE(mode)` with the mode alternating between `HAL_CAPSENSE_MODE_UP` and `HAL_CAPSENSE_MODE_DOWN`. To configure the charging/discharging mechanism to your layout, edit the subsequent calls from the macros `CAP_DISCHARGE(mode)` and `CAP_CHARGE(mode)`. Refer to Figure 3 for a better understanding of this mechanism.

4 Limitations and Tips

Because the rise/fall time is a continuously increasing function of presence, it could happen that the presence of a pressure source is not detected as a touch but taken as part of the statistics. The statistics would then be off, and no touch would be detected, until statistics are recalibrated. This event is a slightly better case than that shown in Figure 8. However, the statistics may recalibrate slowly, and you may want to invoke the reset functionality.

It makes good sense to thoroughly analyze the end product. Make sure the physical design is as optimal as possible for your desired behavior.

`btnBaseCapTrackCounter` is updated every iteration. Therefore, even though some button may be touched (and therefore, the respective statistics not updated), the counter is global so it continues to keep counting. This condition could be remedied by giving each button its own counter; however, that technique could increase RAM usage even more.

If your physical implementation is very stable and not affected by changes in temperature, etc., you may find that you do not need to track the base rise/fall time. In this case, you could save all the variables used to track the statistics.

There exist dedicated microcontrollers that can handle more buttons. See MSP430 LaunchPad for a very good example of a more comprehensive capactive touch system (Ref. 7).

5 References

Unless otherwise noted, these document are available through the Texas Instruments website at www.ti.com.

1. Cheng, D. K. (1989). *Field and Wave Electromagnetics* (second edition). New York: Addison Wesley.
2. RemoTI Sample Applications. Texas Instruments user guide. SWRU201B
3. CC253x/CC2540 System-on-Chip Solution. Texas Instruments user guide. SWRU191B
4. CC2533, RF/IF, and ZigBee Product Information. Texas Instruments website.
5. STB_Front_Panel Schematic, Rev B.
6. SmartRF05EB version 1.8.1 schematics. Available at: SmartRF05EB_1_8_1_Schematics
7. Capacitive Touch BoosterPack (430BOOST-SENSE1). Product information page. MSP430 LaunchPad

Revision History

NOTE: Page numbers for previous revisions may differ from page numbers in the current version.

Products		**Applications**	
Audio	www.ti.com/audio	Automotive and Transportation	www.ti.com/automotive
Amplifiers	amplifier.ti.com	Communications and Telecom	www.ti.com/communications
Data Converters	dataconverter.ti.com	Computers and Peripherals	www.ti.com/computers
DLP® Products	www.dlp.com	Consumer Electronics	www.ti.com/consumer-apps
DSP	dsp.ti.com	Energy and Lighting	www.ti.com/energy
Clocks and Timers	www.ti.com/clocks	Industrial	www.ti.com/industrial
Interface	interface.ti.com	Medical	www.ti.com/medical
Logic	logic.ti.com	Security	www.ti.com/security
Power Mgmt	power.ti.com	Space, Avionics and Defense	www.ti.com/space-avionics-defense
Microcontrollers	microcontroller.ti.com	Video and Imaging	www.ti.com/video
RFID	www.ti-rfid.com		
OMAP Applications Processors	www.ti.com/omap	**TI E2E Community**	e2e.ti.com
Wireless Connectivity	www.ti.com/wirelessconnectivity		