CSA Software Filters with EzI2Cs Slave on CY8C20xx6

CE63794

Code Example Name: Example_CSA_EzI^{2}Cs_Filters_20xx6
Programming Language: C
Associated Part Families: CY8C20xx6
Software Version: PD5.1 (SP2) Build 2306
Related Hardware: CY3280-20x66 UCC CY3280-SLM Module, CY3240-I2USB Bridge or MiniProg3
Author: Subbarao Lanka

Code Example Objective

This code example describes how to scan one CapSense® button and a five-segment slider regularly and send the data to the master using I^{2}C protocol. This code example also describes how to apply the software filters such as average, IIR, and median filters on raw count and the jitter filter on the slider centroid position.

Overview

This code example incorporates the CapSense successive approximation (CSA) module and EzI2Cs module to send CapSense data to the I^{2}C master. The CapSense module scans all the buttons and the slider segments and applies filters as enabled and continuously stores the raw count, difference count, baseline, sensor status and slider centroid details in a structure defined by MyI2CRegs. This structure is used by the EzI2Cs module to send the data to the master whenever required.

User Module List and Placement

The following table lists user modules used in this code example and the hardware resources occupied by each user module.

User Module	Placement
CSA	CapSense and comparator
EzI^{2}Cs	I^{2}C/SPI block

User Module Parameter Settings

The following tables show the user module parameter settings for each of the user modules used in the code example.

CSA		
Parameter	**Value**	**Comments**
Finger Threshold	100	After the difference count crosses finger threshold plus hysteresis, the button is said to be in On condition.
Noise Threshold	40	If the difference count is less than this, then it is treated as noise and baseline update algorithm takes care of this by putting it into the update bucket.
Baseline Update Threshold	100	As the noise increases, the update bucket is filled and every time it crosses this threshold, baseline is incremented by '1', and the algorithm continues.
Settling Time	80	The settling time parameter controls the duration of the capacitance to voltage conversion phase. The parameter setting controls a software delay that enables the voltage on the Cmod and Cbus capacitance to stabilize and depends on the clock setting.
IDAC Setting	20	This parameter declares the amount of current that the source is going to pump in the second phase of CSA that is during the Cmod trying to reach Vref from Vstart.
ExternalCap	P0[3]	Cmod is connected to the specified pin.
Hysteresis	10	This takes care of false On and Off situations whenever the button is pressed. Set it equal to the noise threshold.
Debounce	3	If the difference count is more than finger threshold for less than 'Debounce' number of samples, it is not taken as a button press.

CSA		
Parameter	**Value**	**Comments**
Negative Noise Threshold	20	If the raw count is below baseline and the difference count is more than this threshold, the baseline does not update.
Low BaseLine Reset	50	If the raw count is below baseline and the difference count is more than negative noise threshold for number of samples given by this parameter, the baseline resets to the new raw count.
Sensors Autoreset	Disabled	When the parameter is set to disabled, the baseline is updated only when the raw count and the baseline difference is below the noise threshold parameter.
High Level API	Enabled	Setting this parameter to Enabled includes the high-level APIs provided by the user module. Disabling this parameter saves RAM and ROM by excluding high-level APIs.
Clock	IMO	Normally, this parameter is left at default IMO setting. Setting a larger divider of IMO increases the effective resistance of sensor, compensating for the high capacitance.

Note

The parameters for CSA given in the table User Module Parameter Settings on page 1 are set to work without overlay on the CapSense buttons. If you have overlay on CapSense buttons in the board, use the flowchart in CSA Calibration on page 10 to set these CSA parameters.

EzI2Cs		
Parameter	**Value**	**Comments**
Slave Address	10	This parameter decides the address that is assigned to the slave. Value assigned can be any value from 0 to 127 (decimal)
Address_Type	Static	If this parameter is set to dynamic, the address can be changed in firmware at any time.
ROM Registers	Disable	If this is set to Enable, you can use RAM area as one slave and ROM area as another slave with the addresses differing by only a most significant bit (MSB) of 7-bit address. Hence, enabling ROM registers limits the address declaration from 0 to 63 (decimal). MSB = 1 → ROM registers addressed. MSB = 0 → RAM area addressed.
I^2C Clock	100k Standard	It decides the maximum clock speed that the slave can operate at.
I^2C Pin	P1[0]-P1[1]	This tells which pins are going to be used as SDA and SCL lines of I^2C.

Note

P1[0]-P1[1] are also used to program the device; if you find it difficult to use these pins as I^2C lines, then update the MiniProg version.

Global Resources

Important Global Resources		
Parameter	**Value**	**Comments**
IMO Setting	24 MHz	Selects 24 MHz SysClk
CPU_Clock	SysClk/2	Selects 12 MHz as CPU clock

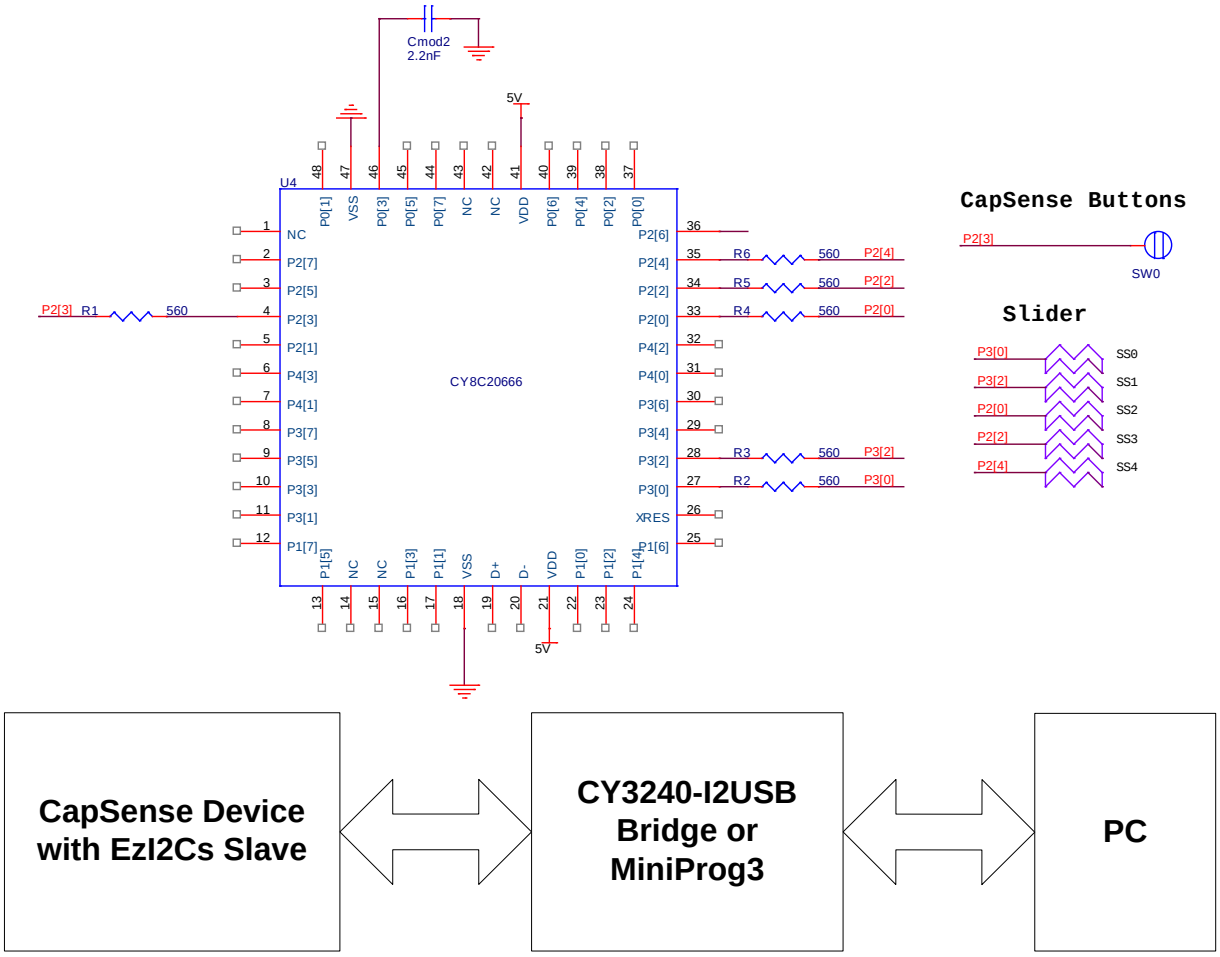
Note

Other parameters may be left at their default value.

Hardware Connections

The schematic diagram for the code example follows.

Figure 1. Schematic Diagram



The CY3280-20x66 Universal CapSense controller board along with the CY3280-SLM Universal CapSense Linear Slider Module is suitable for this code example. Cmod is connected to P0[3]. The CY3240-I2USB Bridge or MiniProg3 can be used as I^2C master to get the CapSense data from slave device. Because P1[0] and P1[1] are used as I^2C lines, the CY3240-I2USB Bridge or MiniProg3 can be connected to the ISSP header. The CY3240-I2USB Bridge or MiniProg3 bridge program can be used to monitor the CapSense data. A 560-ohm resistor is connected in series with each CapSense button to reduce RF interference.

The pin assignment for CapSense buttons used in this code example is as follows:

- Button 0 – P2[3]
- Slider Segment 0 – P3[0]
- Slider Segment 0 – P3[2]
- Slider Segment 0 – P2[0]
- Slider Segment 0 – P2[2]
- Slider Segment 0 – P2[4]

Operation

On reset, all hardware settings from the device configuration are loaded into the device and *main.c* is executed.

The following operations are performed by the firmware.

- A structure (sI2CRegs) is defined to store the button number, raw count, difference count, baseline, status of the CapSense button, and slider centroid position.
- Global interrupt is enabled and the CSA user module is started, finger thresholds for buttons are set, and baselines are initialized.
- EzI2Cs user module is started and 'sI2CRegs' is set as the I^2C buffer.
- IIR filter history is initialized if IIR filter is enabled
- In an infinite while loop, the following functions are performed:
  - All the CapSense buttons are scanned
  - Filters are applied on raw counts if enabled
  - All baselines are updated
  - , Slider centroid position is calculated
  - Jitter filter on slider centroid position is applied if enabled
  - SI2CRegs is updated with the raw count, difference count, baseline, and status of the requested CapSense button.
- I^2C master can request the CapSense data of a particular button by writing the button number into the first byte of the I^2C buffer of slave.

Using Filters in the Code Example

- First disable all the filters and tune the CapSense system using the flowchart in CSA Calibration on page 10. To disable all the filters, set the macros to the following values:
  - AVERAGE_FILTER_ENABLED – 0x00
  - IIR_FILTER_ENABLED – 0x00
  - MEDIAN_FILTER_ENABLED – 0x00
  - JITTER_FILTER_RAWCOUNT_ENABLED – 0x00
  - JITTER_FILTER_SLIDER_ENABLED – 0x00
  - FILTER_ORDER_AVERAGE – 0x00
  - FILTER_ORDER_IIR – 0x00
  - FILTER_ORDER_MEDIAN – 0x00
- Monitor the raw count using the CY3240-I2USB Bridge or MiniProg3 (refer to How to Use USB-I2C Tool to Monitor CapSense Data on page 6 on how to monitor raw counts) and measure the noise on raw counts (Noise on raw count = Max raw count – Min raw count when the finger is not present on button). Apply a filter if the noise is more on raw counts.
- To enable filters on raw counts set the following macros
  - Enable average filter → AVERAGE_FILTER_ENABLED – 0x01
  - Enable IIR filter → IIR_FILTER_ENABLED – 0x01
  - Enable median filter → MEDIAN_FILTER_ENABLED – 0x01
  - Enable jitter Filter → JITTER_FILTER_RAWCOUNT_ENABLED – 0x01

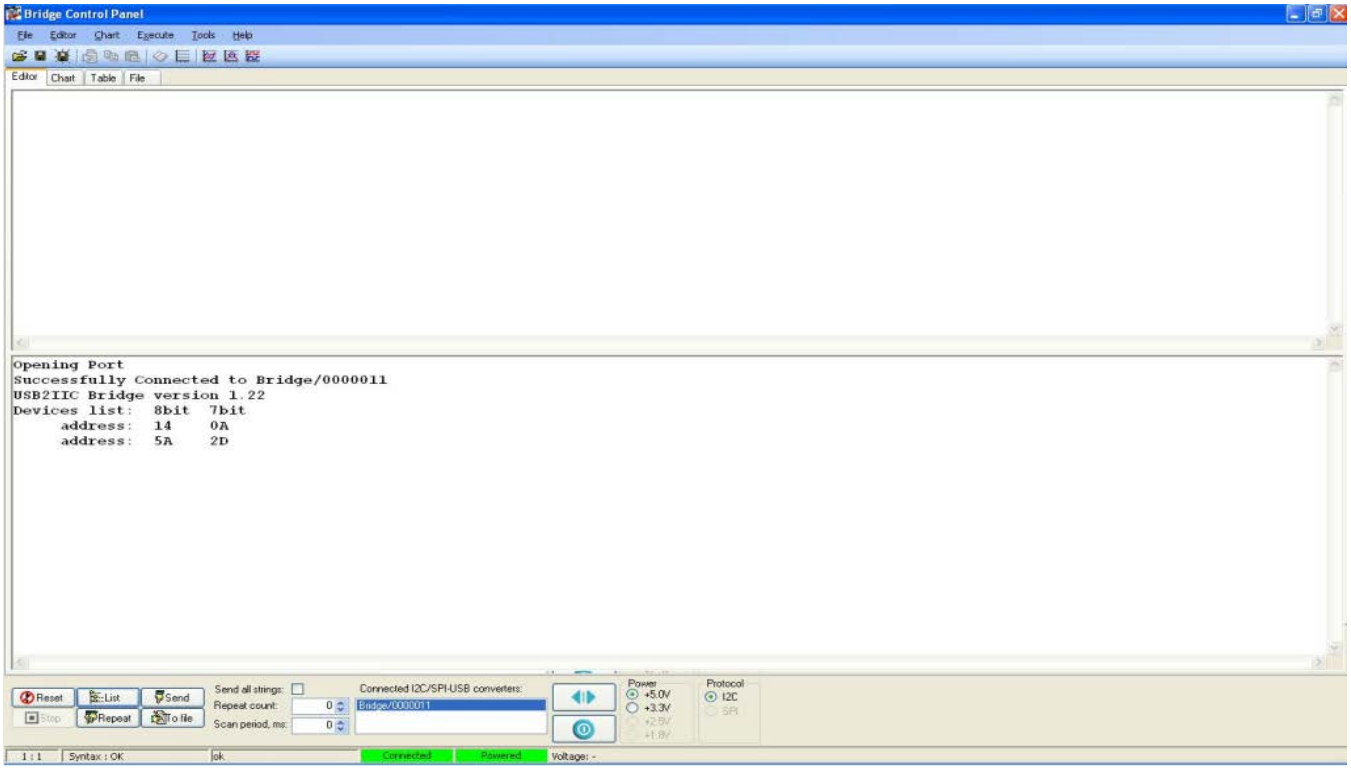
Note If the average, IIR, and median filters are enabled, the filter is also applied to the raw counts of slider segments.

- Set the order of the filter. Use the macros mentioned to set the order of the filter. The valid filter orders for different filters is as follows:
  - Average filter → FILTER_ORDER_AVERAGE → Valid Order 2-8
  - IIR filter → FILTER_ORDER_IIR → Valid Order 1,2
  - Median filter → FILTER_ORDER_MEDIAN → Valid Order 3,5
- You can enable different filters and use the filter that reduces noise in the code example. Some general recommendations are given below:
  - Medium noise in the system - Average filter order 2,4 and IIR filter of order-1
  - High noise in the system - Average filter order 8 and IIR filter order-2
  - Spikes in the raw counts- Median filter of order 3, 5
- When slider is touched and finger is not moved on slider, the slider centroid position should be constant. But due to noise in the system, the centroid position may vary continuously by ±1 count. To avoid noise on slider, centroid position enables the Jitter filter. To enable jitter filter, set the following macros:
  - JITTER_FILTER_ENABLED – 0x01
- When the slider is touched and the finger does not move on the slider, the slider centroid position should be constant. But due to noise in the system, the centroid position may vary continuously by one or two counts. To avoid noise on the slider, centroid position enables the jitter filter. To enable the jitter filter, set the following macros:
  - JITTER_FILTER_ENABLED – 0x01

How to Use Bridge Control Panel to Monitor CapSense Data

1. Open the Bridge Control Panel Software and connect the CY3240-I2USB Bridge or MiniProg3 to USB port.
2. Click on **Tools** and select protocol configuration. Select IIC Speed as 100 kHz.
3. At the right bottom corner, click the '**+5 V**' radio button. This powers the target device with 5 V.
4. At the left bottom corner, click **List**. This lists all the slave device addresses. In this case, device address is 10 (0x0A) and it is displayed in the status window, as shown in the following figure.

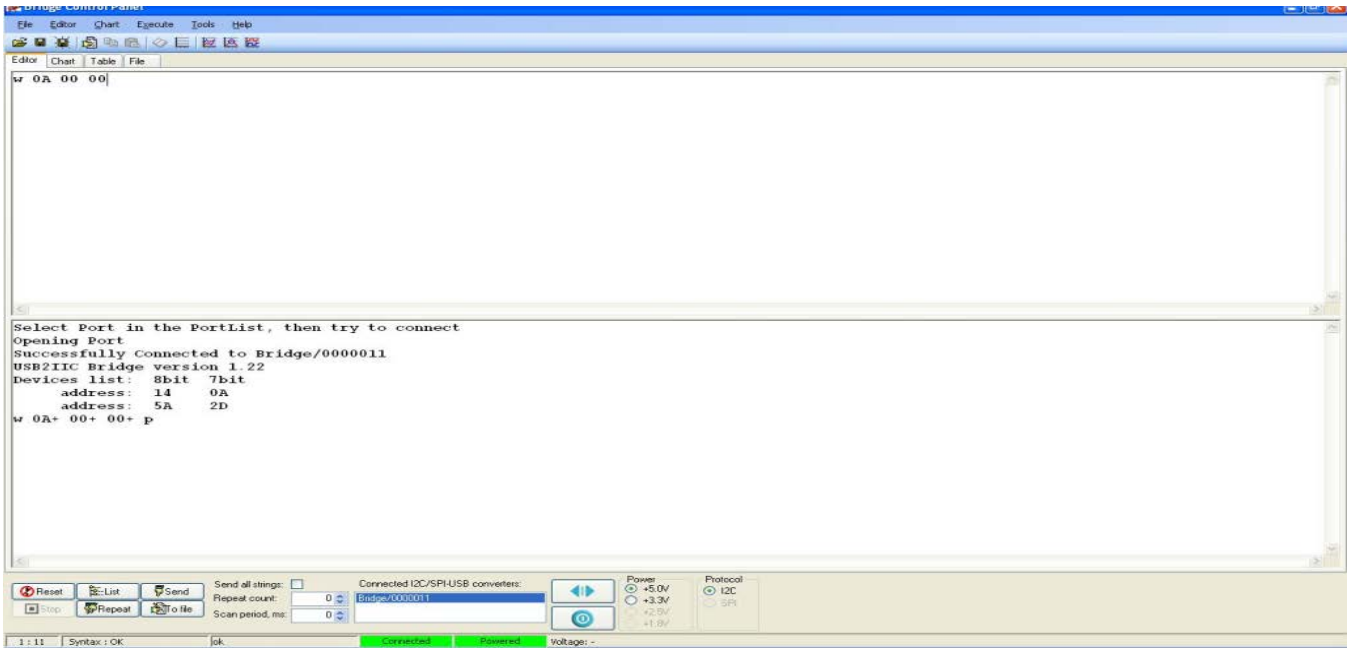
Figure 2. Display of Slave Address in Status Window



5. To monitor the CapSense data of button 0, write the following command in the command window and press enter.

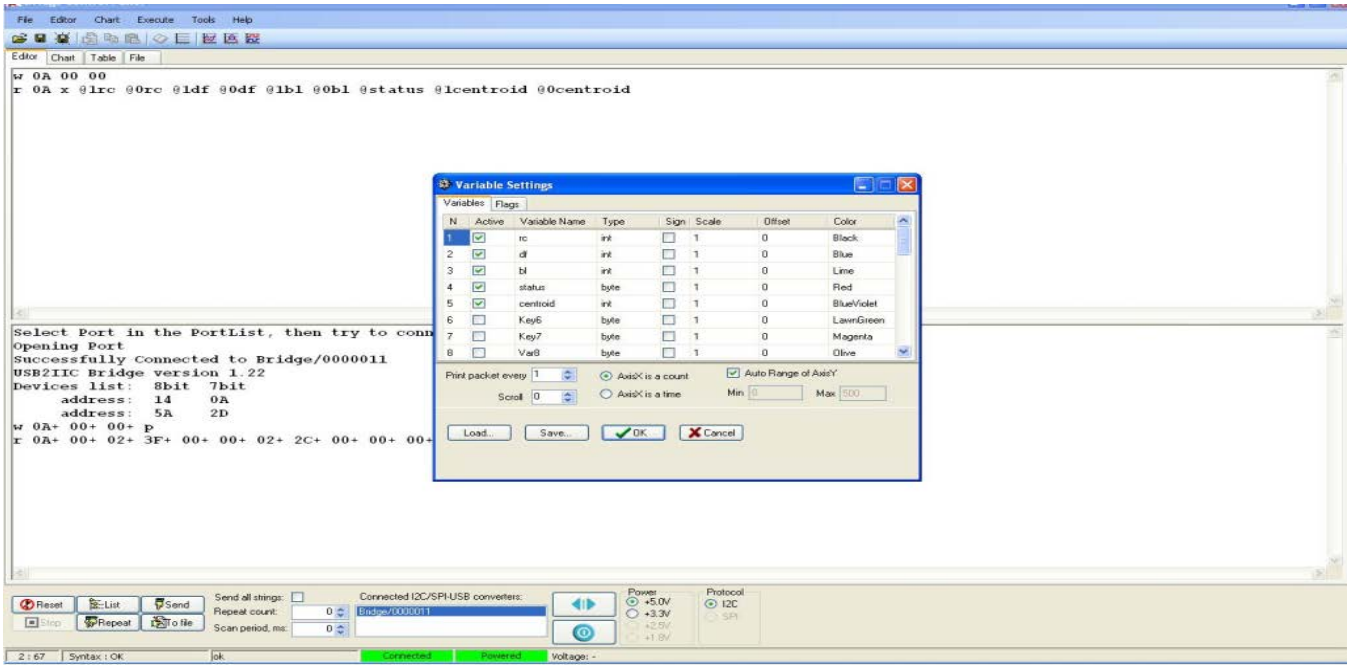
```
W 0A 00 00
```

Figure 3. Write Command Execution



6. To view CapSense data as a graph, click **Chart** and select **Variable Settings**.
7. The **Variables Settings** window appears. In the **Variables** tab, in the **Active** column check the first five check boxes.
8. In the Variable Name column enter the first five names as rc (RawCounts), df (difference counts), bl (baseline), status (button ON/OFF status), and centroid (slider centroid position)In the **Type** column, select int for rc, df, bl and centroid . Select byte for status. Click **OK**.

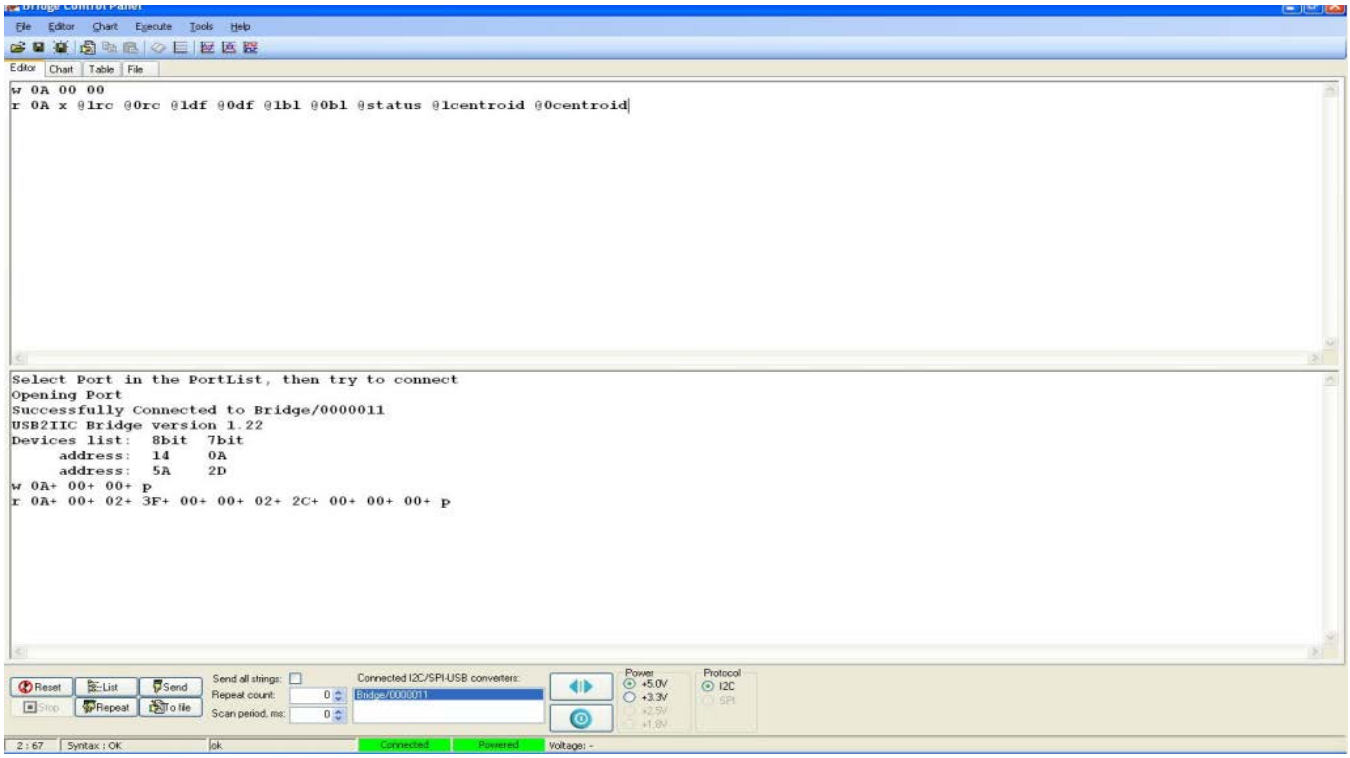
Figure 4. Variable Settings Window



9. Press Ctrl + Enter buttons to move the cursor to next line.
10. Write the following command in the command window:

    r 0A x @1rc @0rc @1df @0df @1bl @0bl @status @1centroid @0centroid

Figure 5. Read Command Execution



11. Click the **Chart** tab and then click **Repeat** button. Now, CapSense data can be seen.
12. To monitor only raw counts, uncheck uncheck df, bl, status, and centroid. This is shown in the following figure.

Figure 6. RawCounts Chart without Button Press

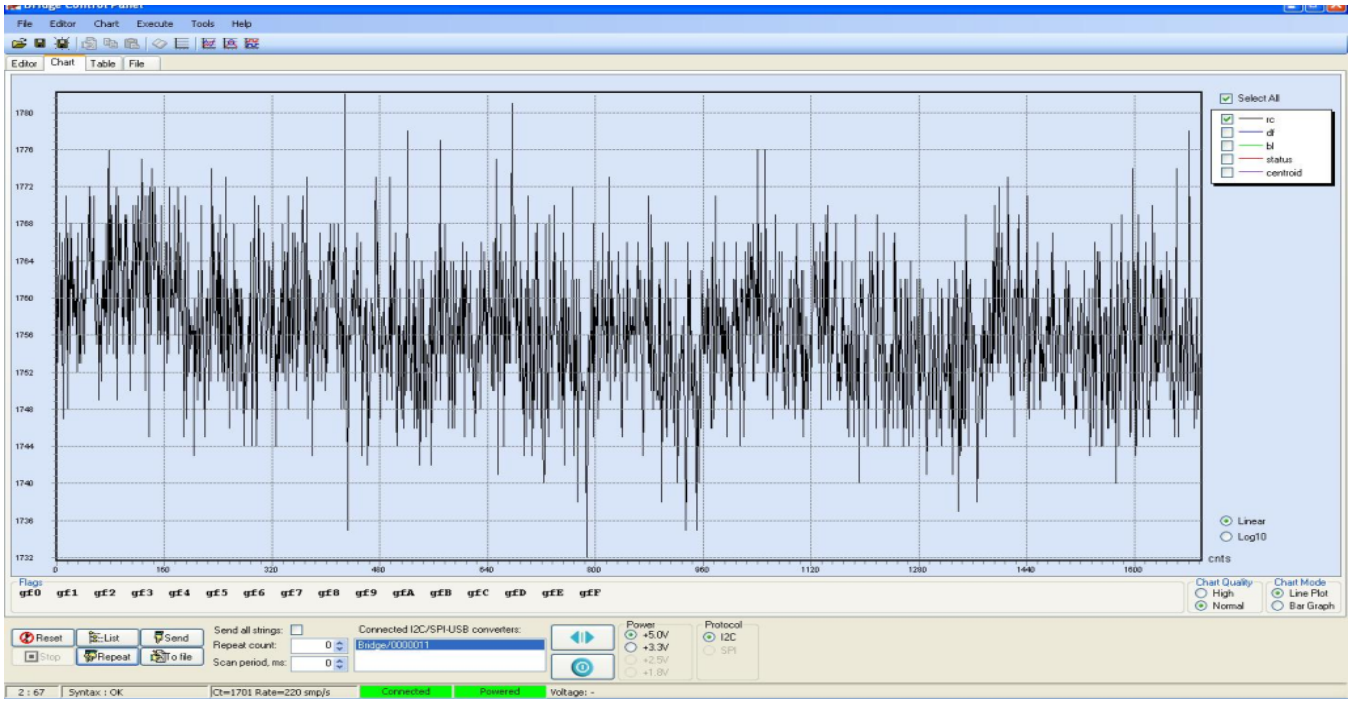
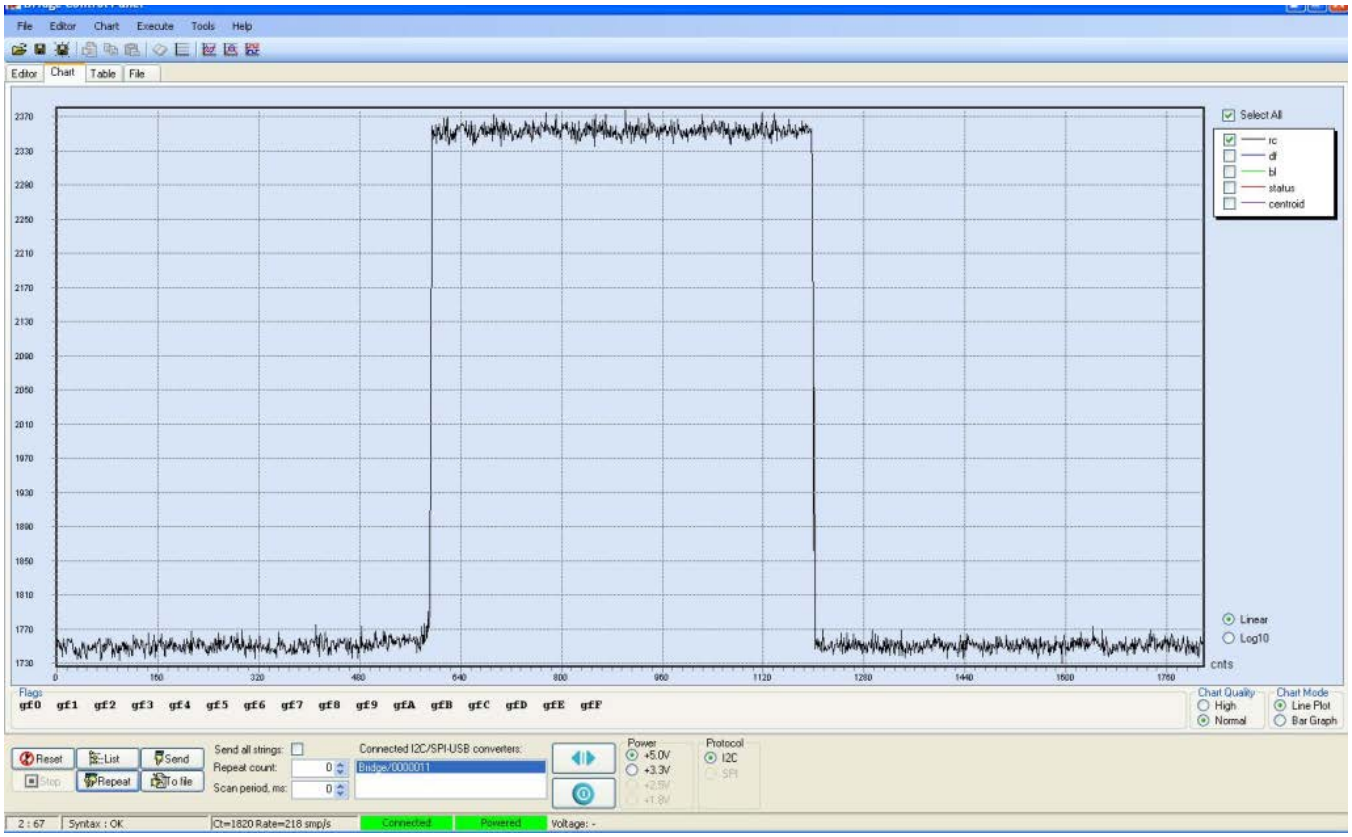


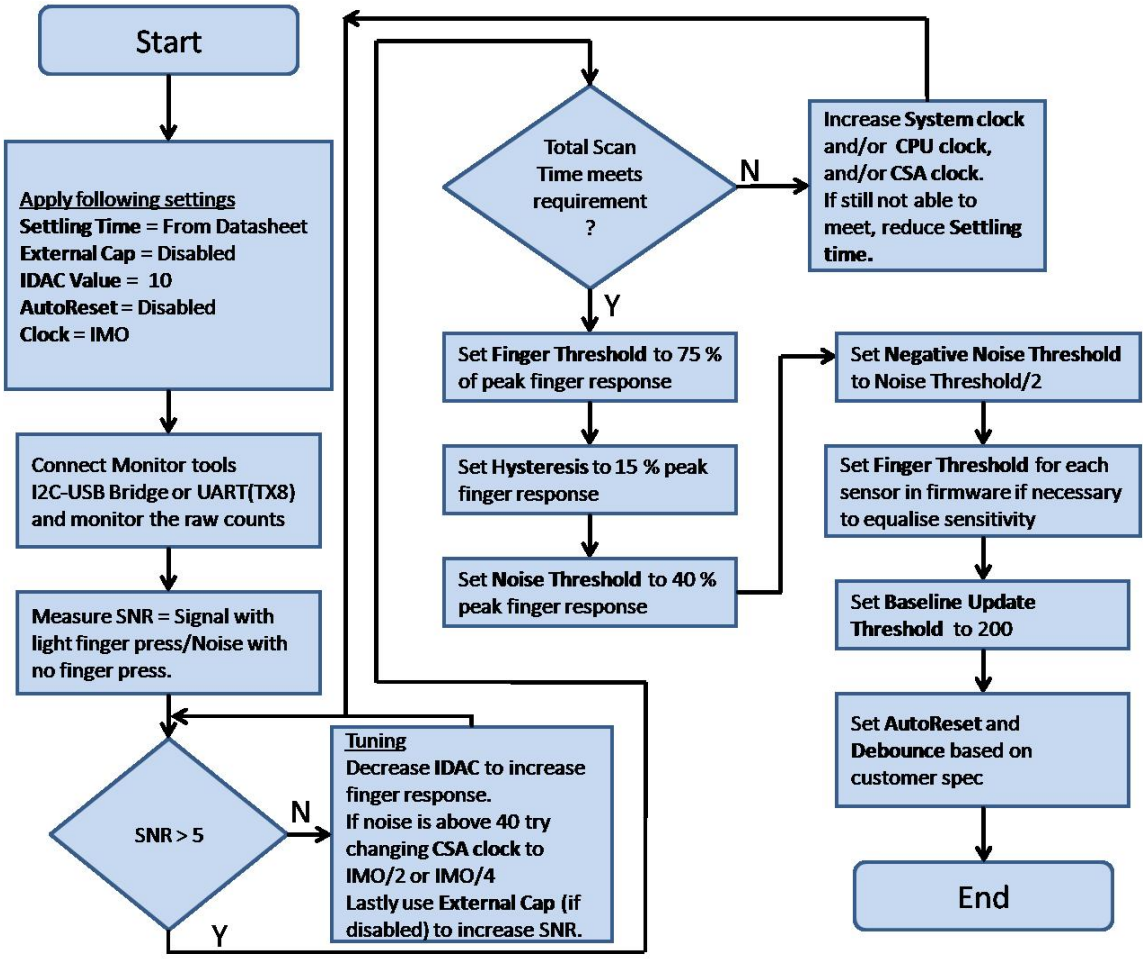
Figure 7. Raw Counts Chart (Button Pressed)



CSA Calibration

For optimum performance, the CSA parameters are tuned with the actual CapSense hardware and overlay. The following flowchart shows the steps to be performed for calibrating CSA.

Figure 8. CSA Calibration Flowchart



1. Start with the default settings of the CSA user module.
2. Using CY3240-I2USB Bridge or MiniProg3 or UART and the actual hardware and overlay capture the raw counts, baseline, and difference counts for the sensors.
3. **Coarse Tuning**. Check if signal-to-noise ratio (SNR) is greater than 5. If SNR is less than 5, increase SNR by following recommended PCB guidelines, decreasing the IDAC value, changing CSA clock, and using an external Capacitor. For PCB guidelines refer to the application note CapSense Best Practices - AN2394. For details about SNR and how to measure SNR, refer to the application note Capacitance Sensing - Signal-to-Noise Ratio Requirement for CapSense Applications - AN2403.
4. Check if total scan time for all the sensors meets requirement. If it does not, increase system clock, CPU clock, and/or CSA clock. Because these parameters also affect SNR, go back to Step 3. With a couple of passes, arrive at the optimum IDAC and CSA clock parameters that produce the best SNR and the desired scan time.
5. Capture the difference counts when the button is activated. Set the finger threshold parameter to 75 percent of the peak finger response.
6. Set the hysteresis parameter to 15 percent of the peak finger response.
7. Set the noise threshold to 40 percent of the peak finger response.
8. Set the negative noise threshold to half the noise threshold.

9. Set the finger thresholds for individual sensors if necessary. This is done by writing to the `CSA_baBtnFThreshold` array in firmware.
10. Set the baseline update threshold according to requirements. The frequency with which the baseline is updated must be determined on a project-to-project basis. The baseline should be a slow-moving reference, which helps to reduce the affects of noise and temperature on the capacitive sensor.
    - **Fast update baseline rates:** This can create problems if you move your finger slowly to the button. This is called ‘Baselining out the finger.’
    - **Slow update baseline rates**: This can leave the buttons vulnerable to temperature fluctuations and potentially lead to ‘button lock.’
11. Set AutoReset and Debounce parameters as required. Refer to the CSA user module datasheet for details of these parameters.
12. For any other parameters refer to the user module datasheet.

Document History

Document Title: CSA Software Filters with EzI2Cs Slave on CY8C20xx6 - CE63794

Document Number: 001-63794

Revision	ECN	Orig. of Change	Submission Date	Description of Change
**	3015652	SLAN	11/15/2011	New example projects
*A	3315563	ARVM	07/15/2011	Modified title. Updated as per code example template. Added Document History section. Figure 2 to Figure 7 updated.
*B	4196214	SLAN	11/19/2013	No technical updates. Completing Sunset Review.
*C	4602248	SLAN	12/19/2014	Replaced “I2CtoUSB bridge”, “USB-I2C bridge”, “USBtoI2C Bridge”, and “I2C-USB bridge” with “CY3240-I2USB Bridge or MiniProg3” in all instances across the document.

PSoC and CapSense are registered trademarks of Cypress Semiconductor Corp. PSoC Designer is a trademark of Cypress Semiconductor Corp. All other trademarks or registered trademarks referenced herein are the property of their respective owners.

Cypress Semiconductor
198 Champion Court
San Jose, CA 95134-1709
Phone: 408-943-2600
Fax: 408-943-4730
http://www.cypress.com/

© Cypress Semiconductor Corporation, 2011-2014. The information contained herein is subject to change without notice. Cypress Semiconductor Corporation assumes no responsibility for the use of any circuitry other than circuitry embodied in a Cypress product. Nor does it convey or imply any license under patent or other rights. Cypress products are not warranted nor intended to be used for medical, life support, life saving, critical control or safety applications, unless pursuant to an express written agreement with Cypress. Furthermore, Cypress does not authorize its products for use as critical components in life-support systems where a malfunction or failure may reasonably be expected to result in significant injury to the user. The inclusion of Cypress products in life-support systems application implies that the manufacturer assumes all risk of such use and in doing so indemnifies Cypress against all charges.

This Source Code (software and/or firmware) is owned by Cypress Semiconductor Corporation (Cypress) and is protected by and subject to worldwide patent protection (United States and foreign), United States copyright laws and international treaty provisions. Cypress hereby grants to licensee a personal, non-exclusive, non-transferable license to copy, use, modify, create derivative works of, and compile the Cypress Source Code and derivative works for the sole purpose of creating custom software and or firmware in support of licensee product to be used only in conjunction with a Cypress integrated circuit as specified in the applicable agreement. Any reproduction, modification, translation, compilation, or representation of this Source Code except as specified above is prohibited without the express written permission of Cypress.

Disclaimer: CYPRESS MAKES NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS MATERIAL, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. Cypress reserves the right to make changes without further notice to the materials described herein. Cypress does not assume any liability arising out of the application or use of any product or circuit described herein. Cypress does not authorize its products for use as critical components in life-support systems where a malfunction or failure may reasonably be expected to result in significant injury to the user. The inclusion of Cypress’ product in a life-support systems application implies that the manufacturer assumes all risk of such use and in doing so indemnifies Cypress against all charges.

Use may be limited by and subject to the applicable Cypress software license agreement.