CSD with I2CHW Slave on CY8C20xx6

CE54364

Code Example Name: Example_CSD_I^{2}CHW_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: Arvind M

Code Example Objective

CE54364 describes how to scan four CapSense® buttons regularly and send the data to the master using I^{2}C protocol.

Overview

This code example incorporates CapSense Sigma Delta (CSD) and I^{2}CHW modules to send the CapSense data to the I^{2}C master. The CapSense module scans all the buttons continuously and stores the raw count, difference count, and baseline details in a structure defined by MyI^{2}CRegs. This structure is used by the I^{2}CHW module to send the data to the master when 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
CSD	CapSense and comparator, Timer1
I^{2}CHW (normal slave operation)	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.

CSD		
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 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.
Sensors Autoreset	Disabled	When the parameter is set to disabled, the baseline is updated only when raw count and baseline difference is below the noise threshold parameter.
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.
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 a new raw count.
Modulator Capacitor Pin	P0[3]	Indicates to which pin Cmod is connected.
iDAC Value	70	Capacitance measurement range depends on this parameter. Higher value corresponds to wider range.

CSD		
Parameter	**Value**	**Comments**
Resolution	12	Higher the resolution, higher is the sensitivity
Scanning Speed	Normal	Decides the speed of scanning process.
ShieldElectrodeOut	None	Shield electrode is not used in this code example.
PrechargeSource	PRS	This parameter selects PRS as the clock source for precharge switches.
Prescaler	2	This parameter determines the precharge switch output frequency.
PRS Resolution	8 bit	This parameter sets the PRS sequence length. Sequence length for 8 bit is 511.
Autocalibration	Disabled	Autorange of capacitance measurement is disabled.

Note The parameters for CSD 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 CSD Calibration on page 8 to set these CSD parameters.

I^2CHW		
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)
Read_Buffer_Types	RAM ONLY	Only RAM data buffer used
Communication_Service_Type	Interrupt	See Notes below
I^2C Clock	100 kHz 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.

Notes

- When the Read_Buffer_Types is set to RAM ONLY, only RAM buffers may be transmitted over I^2C. If data from Flash buffer is to be read and transmitted, the read buffer type should be set to RAM OR FLASH
- In interrupt based Communication_Service_Type, data is moved in and out of buffer quickly in the background using an ISR.
- The I^2C clock is dependent on the SysClk. The I^2C clock setting in the user module is based on a SysClk of 24 MHz. In devices which support slower Sysclk, the I^2C clock is reduced by the same proportion. For example, if the I^2C clock is set to 400 kHz and SysClk is set to 6 MHz, the actual I^2C clock is only 100 kHz.

Global Resources

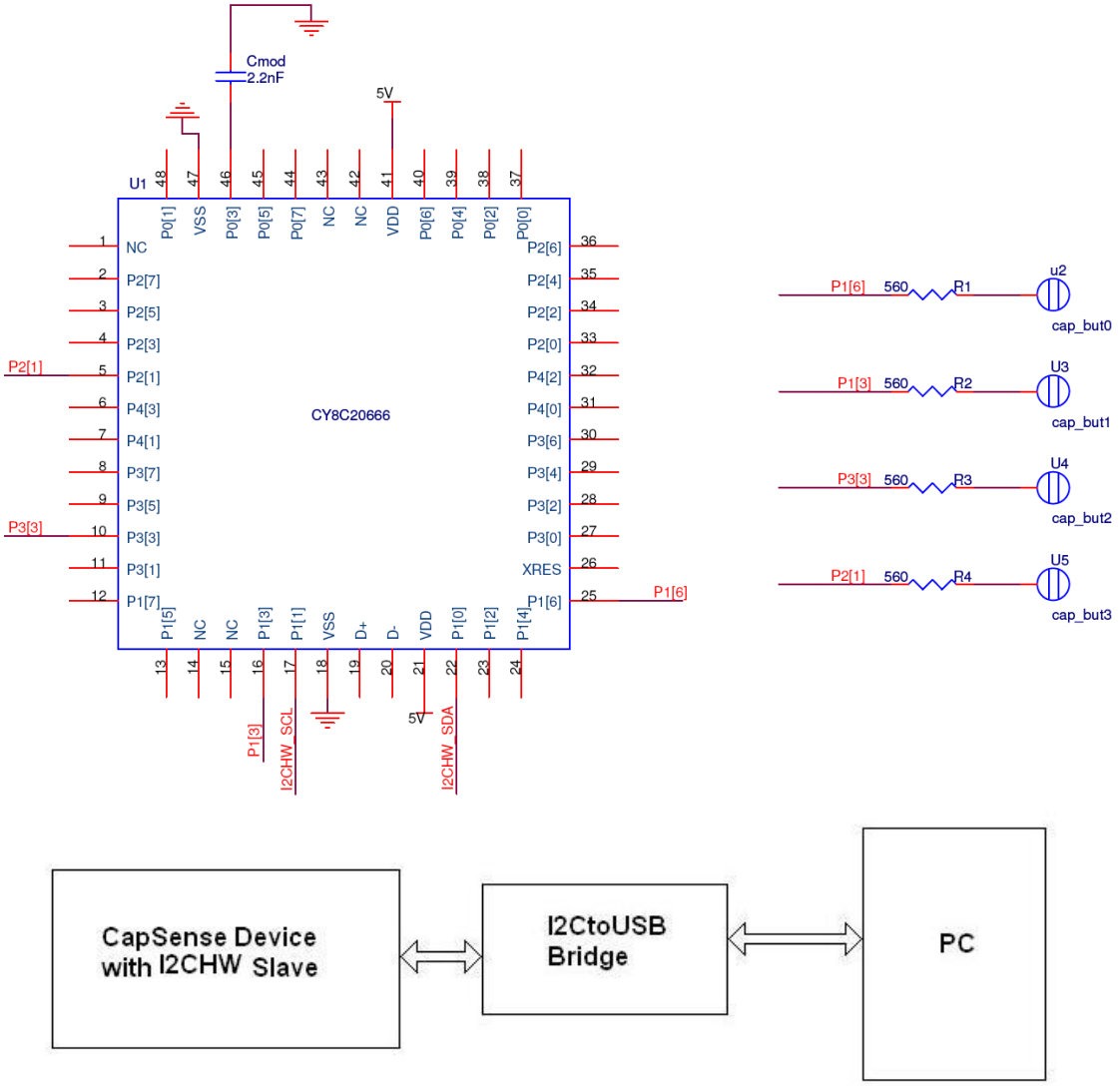
Important Global Resources		
Parameter	**Value**	**Comments**
IMO Setting	24 MHz	Selects 24 MHz SysClk.
CPU_Clock	SysClk/2	Selects 12 MHz as CPU clock.

Note Other parameters are left at their default value.

Hardware Connections

The schematic diagram for the code example follows.

Figure 1. Schematic Diagram



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

The CapSense buttons pin assignment (used in this code example) are as follows:

- Button 0 - P1[6]
- Button 1 - P1[3]
- Button 2 – P3[3]
- Button 3 – P2[1]

Operation

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

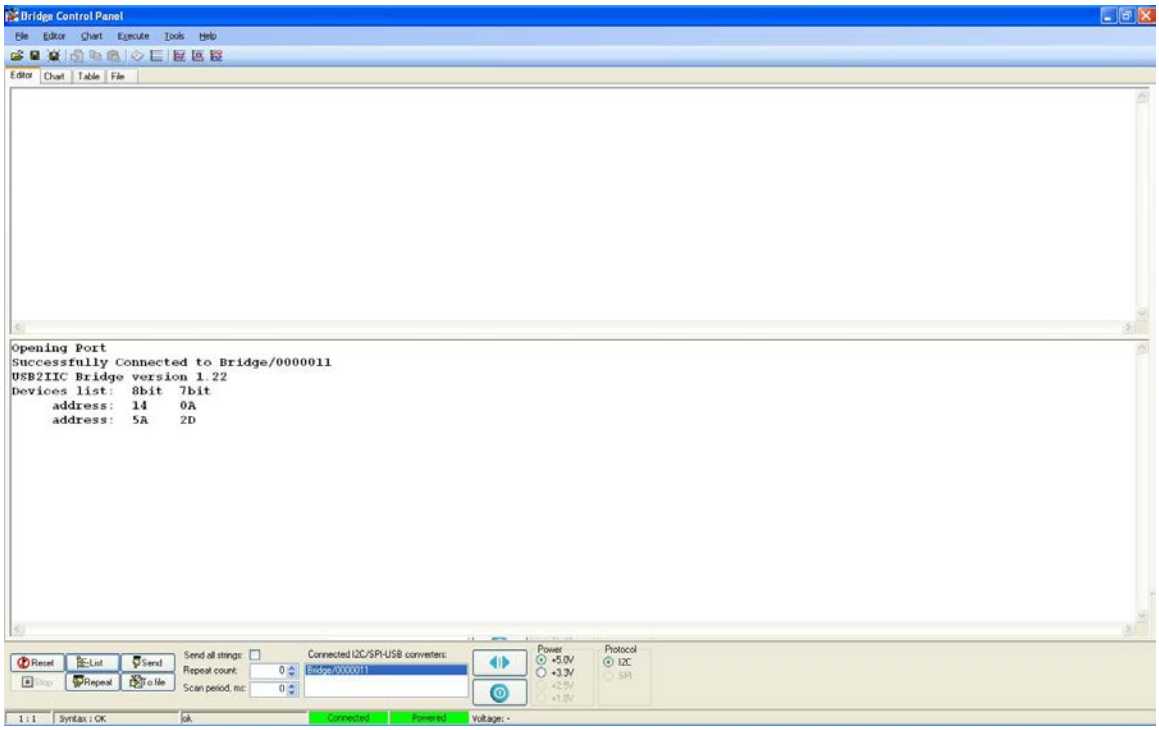
The following operations are performed by the firmware:

- A structure (sI^{2}CRegs) is defined to store the button number, raw count, difference count, baseline, and status of the CapSense button.
- Global interrupt is enabled and CSD user module is started, finger thresholds for buttons are set, and baselines are initialized.
- I^{2}CHW user module is started. Structure "sI^{2}CRegs" is set as the I^{2}C Read buffer and BYTE variable "bButtonNumber" is set as the I^{2}C write buffer.
- To get information of a button, I^{2}C master must write the button number into the I^{2}C write buffer (bButtonNumber). I^{2}C master can get that button information by reading the I^{2}C read buffer (sI^{2}CRegs).
- In an infinite while loop, all the CapSense buttons are scanned, and then all baselines are updated; sI^{2}CRegs is updated with the raw count, difference count, baseline, and status of the requested CapSense button.
- Whenever I^{2}C master writes (or reads) into the buffer, the write flag (Read flag) must be reset and buffer must be set again. This operation is also done in the while loop.

Instructions to Use Bridge Control Panel for Monitoring 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 on the **+5 V** radio button. This powers the target device with 5 V.
4. At the left bottom corner, click the **List** button. This lists all the slave device addresses. In this case, the device address is 10 (0x0A) and it is displayed in the status window, as shown in the following figure.

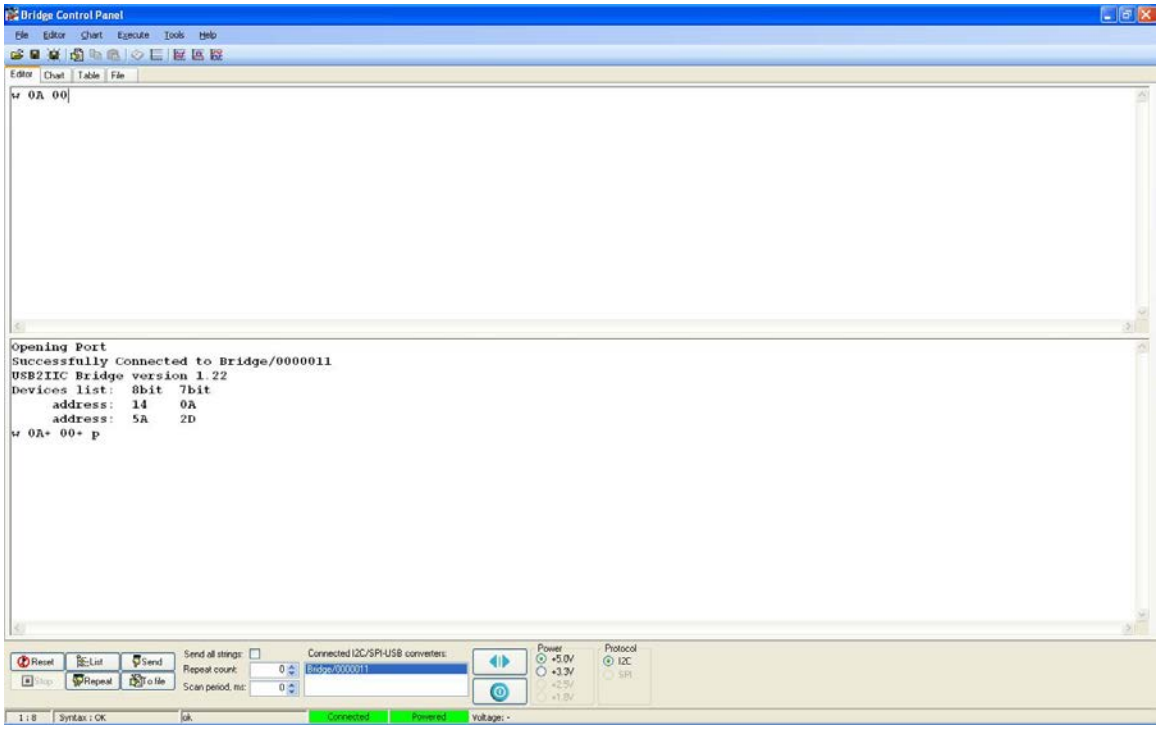
Figure 2. Display of Slave Address in Status Window



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

```
W 0A 00
```

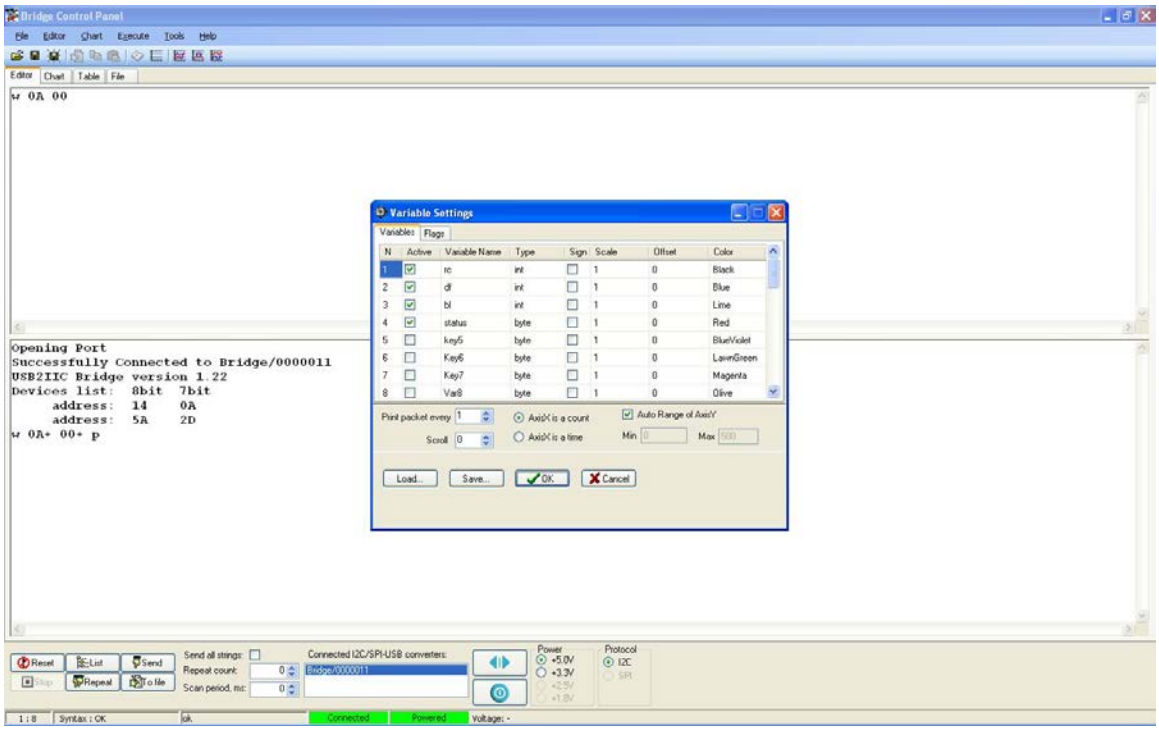
Figure 3. Write Command Execution



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 four check boxes.
8. In the **Variable Name** column enter the first four names as rc (RawCounts), df (difference counts), bl (baseline), and status (button ON/OFF status).

9. In the **Type** column, select int for rc, df, and bl. Select byte for status. Click **OK**.

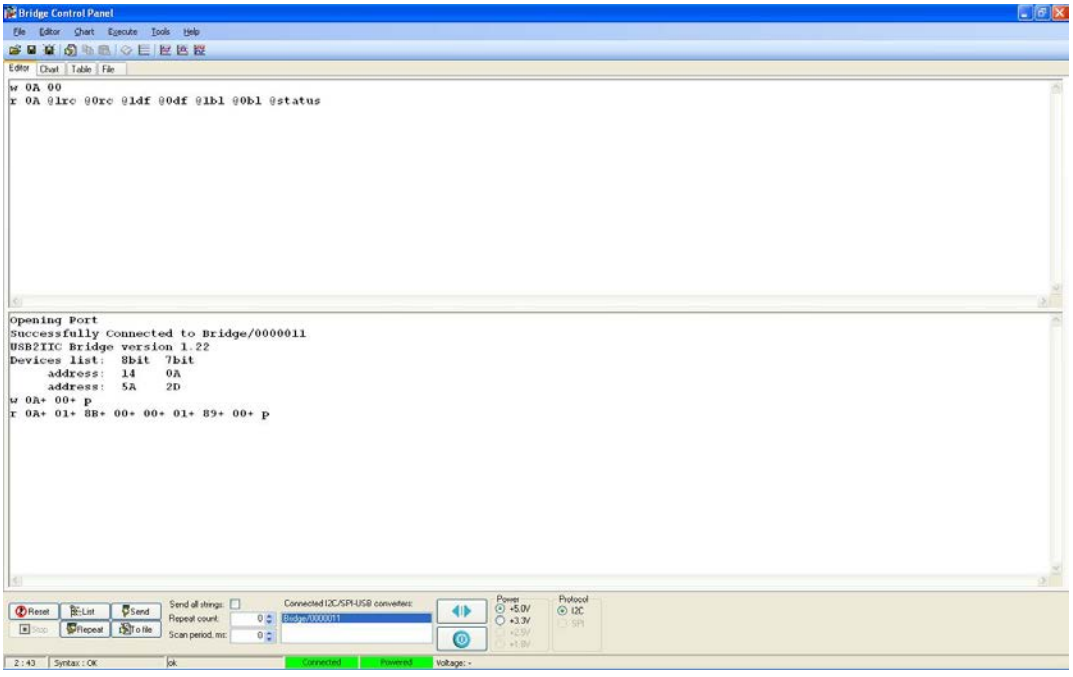
Figure 4. Variable Settings Window



10. Press <Ctrl> + <Enter> buttons to move the cursor to next line.
11. Write the following command in the command window.

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

Figure 5. Read Command Execution



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

Figure 6. Raw Counts Chart without Button Press

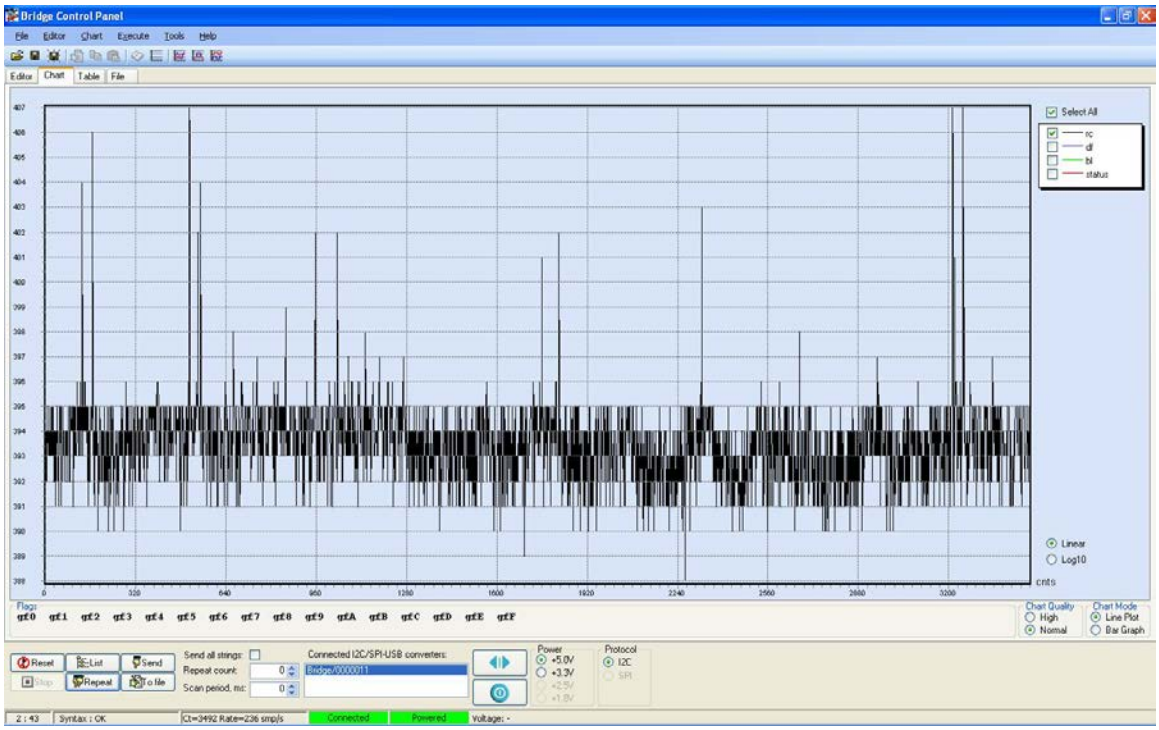
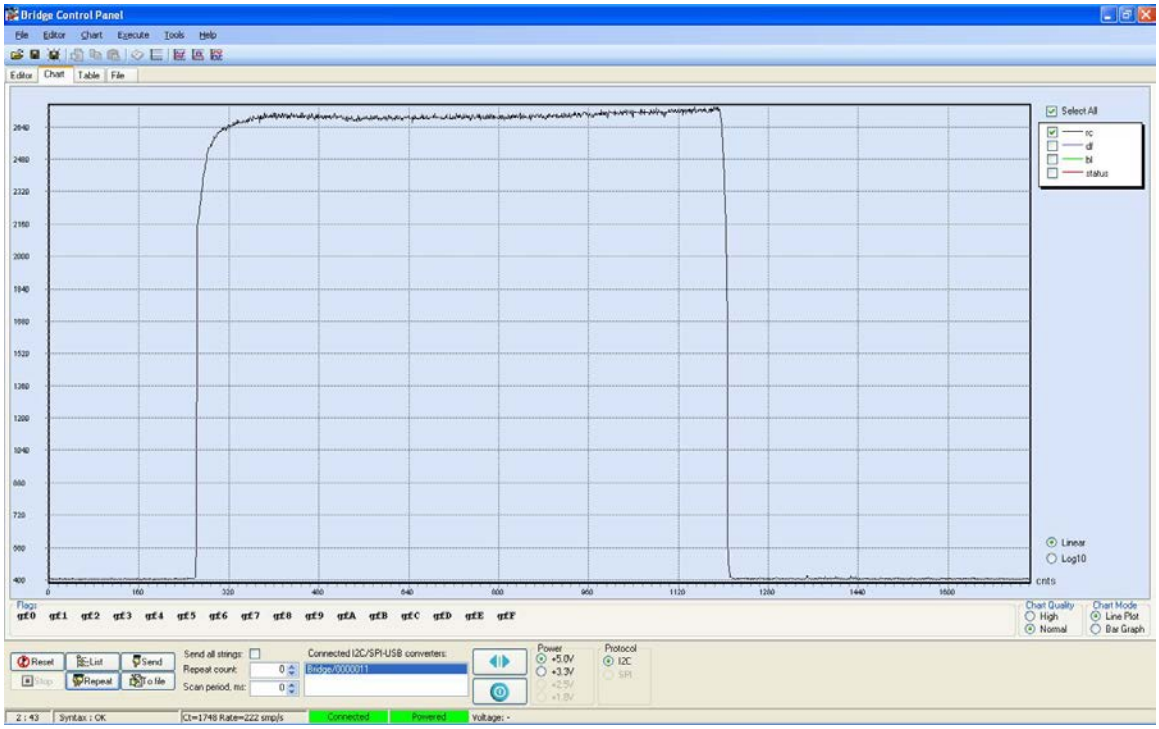


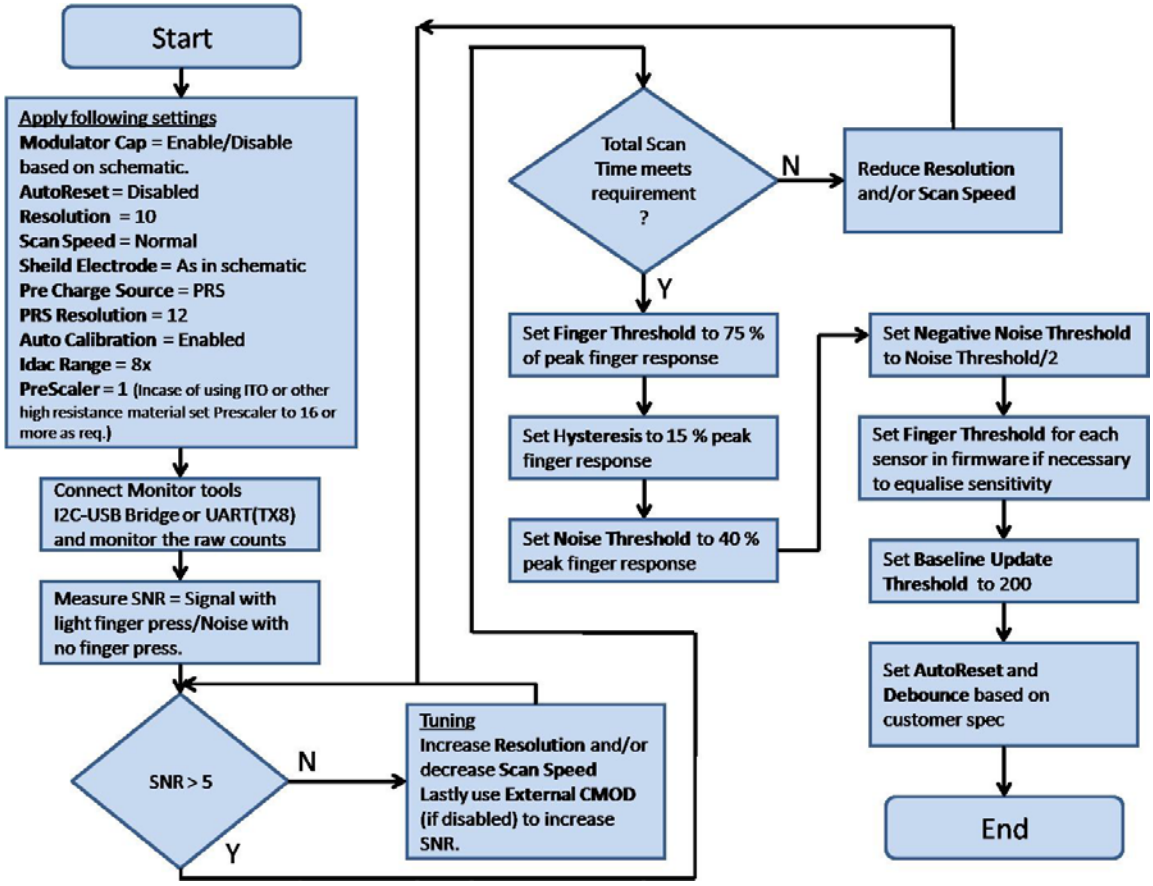
Figure 7. Raw Counts Chart (Button Pressed)



CSD Calibration

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

Figure 8. CSA Calibration Flowchart



1. Start with the default settings of the CSD user module.
2. Using the 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, increasing the resolution of the CSD, using external CMOD, and reducing the scan speed of the CSD. For PCB guidelines see the section 3.7 of Getting Started with CapSense. For details about SNR and how to measure SNR, see the section 4.1.1 of CY8C20xx6A/H CapSenseÂ® Design Guide.
4. Check if total scan time for all the sensors meets requirements. If it does not, reduce resolution and/or increase scan speed. Because these parameters also affect SNR, go back to Step 3. With a couple of passes, arrive at the optimum resolution and scan speed 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.
6. Set the noise threshold to 40 percent of the peak value.
7. Set the hysteresis to 15 percent of the peak value.
8. Set the negative noise threshold to half the noise threshold.
9. Set finger thresholds for individual sensors if necessary. This is done by writing to the `CSD_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 a user moves their 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. See the CSD user module datasheet for details of these parameters.
12. For any other parameters refer to the user module datasheet.

Document History

Document Title: CSD with I2CHW Slave on CY8C20xx6 - CE54364

Document Number: 001-54364

Revision	ECN	Orig. of Change	Submission Date	Description of Change
**	2727566	ARVM	07/01/2009	New Spec
*A	2973490	ARVM	07/08/2010	Updated the software version to "PD5.0 SP6 Build 1127." Added "Related Hardware" and "Author" fields on page 1, below the title.
*B	3328605	UDYG/ARVM	07/26/2011	Updated the software version to “PD5.1 SP2 Build 2306.” Renamed Example projects to Code Examples.
*C	4491055	SSHH	09/02/2014	No technical updates. Completing Sunset Review.
*D	4740306	SSHH	04/24/2015	Updated Related Hardware as “CY3280-20x66 UCC, CY3280-SLM Module, CY3240-I2USB Bridge or MiniProg3”. Replaced “I^2CtoUSB Bridge” with “CY3240-I2USB bridge or MiniProg3” in all instances across the document. Updated to new template.

Worldwide Sales and Design Support

Cypress maintains a worldwide network of offices, solution centers, manufacturer's representatives, and distributors. To find the office closest to you, visit us at Cypress Locations.

Products

Automotive	cypress.com/go/automotive
Clocks & Buffers	cypress.com/go/clocks
Interface	cypress.com/go/interface
Lighting & Power Control	cypress.com/go/powerpsoc cypress.com/go/plc
Memory	cypress.com/go/memory
PSoC	cypress.com/go/psoc
Touch Sensing	cypress.com/go/touch
USB Controllers	cypress.com/go/usb
Wireless/RF	cypress.com/go/wireless

PSoC® Solutions

psoc.cypress.com/solutions

PSoC 1 |PSoC 3 | PSoC 4 |PSoC 5LP

Cypress Developer Community

Community | Forums |Blogs | Video |Training

Technical Support

cypress.com/go/support

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, 2009-2015. 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.