

BUTTONS AND SLIDER - CSD CY8C21X34

Project Name:	Buttons_Slider_CY3280_21x34
Associated Part Families:	CY8C21x34
Software Version:	PSD5.0 SP5
Programming Language:	C
Related Hardware:	CY3280 – CY8C21x34, CY3280 SLM
Author:	M. Ganesh Raaja

PROJECT OBJECTIVE

Demonstrate the implementation of Capsense buttons and sliders using the CSD user module in the CY8C21x34 family of PSoC.

OVERVIEW

The project implements 5 capsense buttons and a 10 element capsense slider with a resolution of 100 and controls 5 LEDs depending on the status of the buttons and slider. Each LED represents a button. When a button is pressed, the corresponding LED is switched on. Also, all the 5 LEDs form a bar graph to show the slider position.

USER MODULE LIST AND PLACEMENT

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

User Module	Placement
CSD	ACE00, ACE01, ASE11, DBB00, DBB01, DCB02

GLOBAL RESOURCES

Important Global Resources		
Parameter	Value	Comments
CPU_Clk	SysClk/2	Set CPU Clock to 12MHz

Note:

All other Global resources are left at their default values.

USER MODULE PARAMETER SETTINGS

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

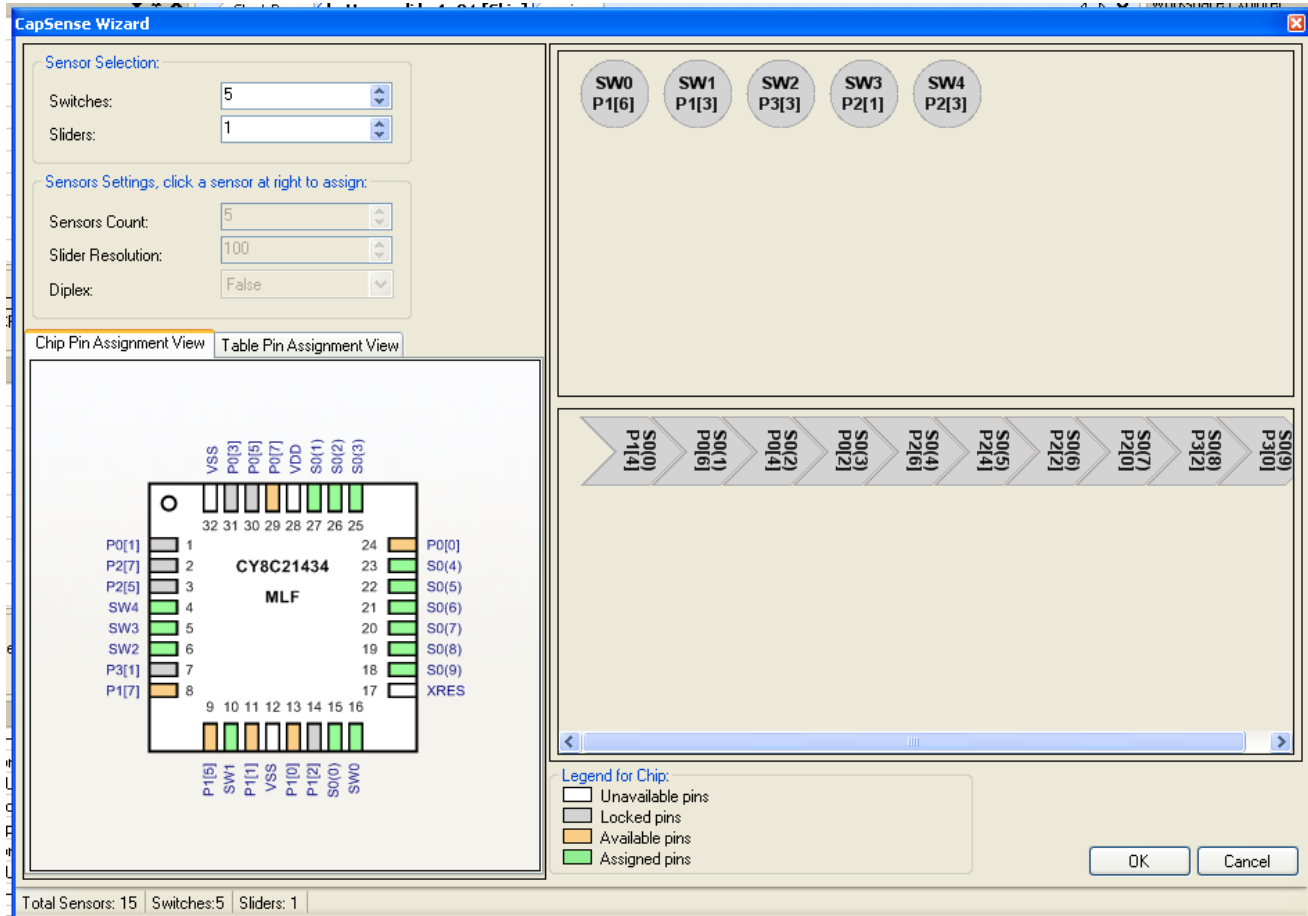
CSD		
Parameter	Value	Comments
Finger Threshold	40	Default Value. Tune for optimum performance. Refer Appendix
Noise Threshold	20	Default Value. Tune for optimum performance. Refer Appendix
Baseline Update Threshold	200	
Sensor Autoreset	Enabled	This prevents the sensor getting permanently enabled
Hysteresis	10	
Debounce	3	Only if the sensor is active for 3 successive scans, it is recognized
NegativeNoiseThreshold	20	Default Value. Tune for optimum performance. Refer Appendix
LowBaselineReset	50	
Scanning Speed	Normal	For buttons and sliders, normal scan speed provides optimum noise

		performance. For proximity sensing, slow scan speed is preferred
Resolution	12	For buttons and sliders, 12 bit resolution is sufficient. For proximity sensing it is preferable to use the highest resolution
Modulator Capacitor	P0[3]	This parameter is set according to the CY3280 hardware. Change this to suit your board
Feedback Resistor	P3[1]	This parameter is set according to the CY3280 hardware. Change this to suit your board
Reference	ASE11	
RefValue	4	Default Value. Tune for optimum performance. Refer Appendix
Shield Electrode	None	

Note:

Most of the CSD parameters are left at their default values. This works fine for the CY3280 or CY3213 development boards. But when the project has to be run on the actual hardware with the overlay, all the parameters have to be tuned to achieve best sensitivity and SNR. Refer Appendix for the procedure to tune a CSD capsense application

Below is the screen shot of the CSD wizard to set the Button and Slider elements

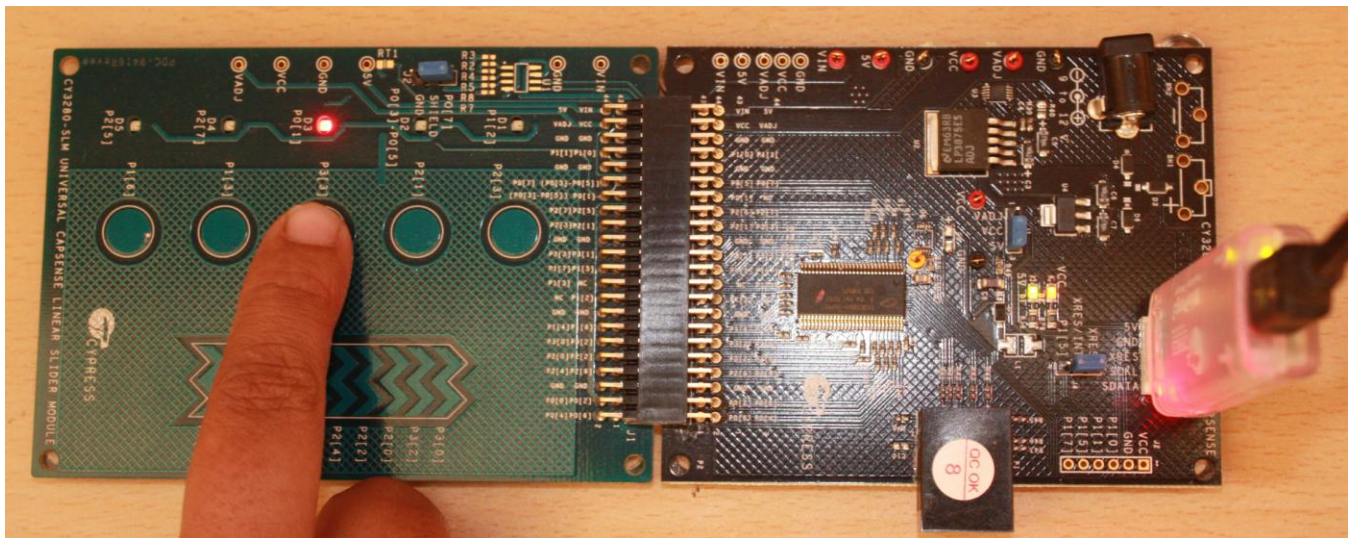


GPIO SETTINGS FOR THE LEDs

PORT	NAME	MODE, DRIVE MODE, INTERRUPT
P2[5]	LED1	StdCPU, Strong, Disable Interrupt
P2[7]	LED2	StdCPU, Strong, Disable Interrupt
P0[1]	LED3	StdCPU, Strong, Disable Interrupt
P0[5]	LED4	StdCPU, Strong, Disable Interrupt
P1[2]	LED5	StdCPU, Strong, Disable Interrupt

HARDWARE

The project can be tested using the CY3280-21x34 + CY3280 SLM combination of Development boards from Cypress Semiconductors. The CY3280 SLM board has 5 buttons, a 10 element slider and 5 LEDs. The CY3280 SLM board is connected to the CY3280-21x34 board using the 44 pin header. The code is downloaded on the CY3280-21x34 board using the ISSP header. Power up the board either using a MiniProg through the ISSP header or through the 12V DC adaptor connector. Now pressing the buttons will light the corresponding LED. Moving the finger on the slider will simulate a bar graph on the 5 LEDs.



OPERATION

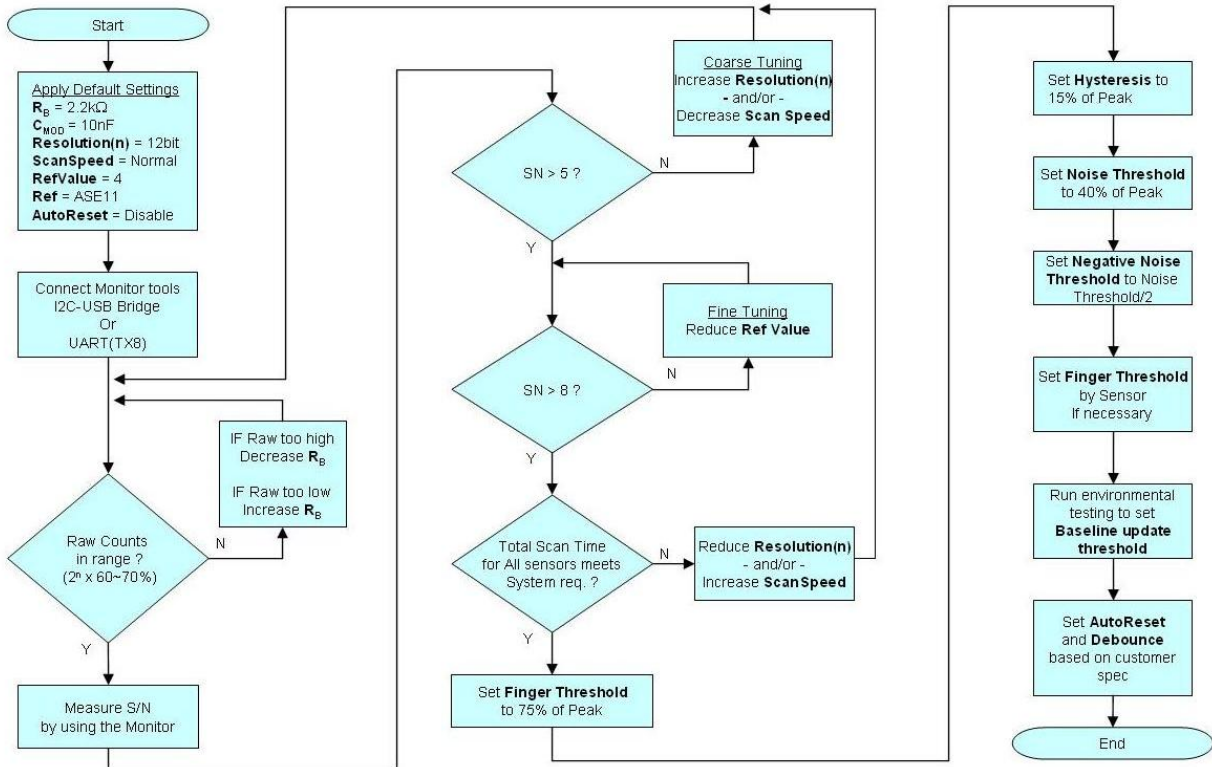
On power up, the code in boot.asm is executed. The device configuration is loaded inside boot.asm and all the hardware resources are configured as they are defined in the device editor. After the hardware resources are configured, main.c is executed. The following operations are performed inside main.c.

- Global interrupts are enabled.
- As there are only 5 LEDs on the CY3280-SLM board and these 5 LEDs should be used to indicate the status of both the buttons and Slider, a global variable called LEDShadow is used. Bit0 to Bit4 of this variable represent the status of LED1 to LED5. In the program, the function ProcessEvents sets or clears these bits depending on the status of the Buttons and Sliders. And the function “ControlOutputs” controls the LEDs based on the status of the bits.
- CSD is started, baselines are initialized and the finger thresholds are set to default values
- Inside an infinite loop, the following operations are performed

- CSD_ScanAllSensors function is called. This will scan the 5 buttons and 10 elements of the slider.
- CSD_UpdateAllBaselines function is called to update the baseline of all the sensors.
- The function ProcessEvents is called. Following operations are performed inside the ProcessEvents function.
 - All bits in the LEDShadow variable are cleared.
 - The function CSD_blsAnySensorActive is called. This function returns TRUE if any of the buttons or slider is active
 - If CSD_blsAnySensorActive returns TRUE, the function CSD_blsSensorActive is called for each button. The blsSensorActive returns TRUE if the particular button is pressed. On TRUE, the corresponding bit in the LEDShadow register is set.
 - CSD_wGetCentroidPos function is called to read the Slider status. If the Slider is active, then the wGetCentroidPos function returns a value from 0 to 100 and this value is stored in the SliderPosition variable. If the Slider is not active, then the wGetCentroidPos function returns -1 (65535) and in this case, the variable SliderPosition is made 0.
 - Based on the value of SliderPosition, the Bit0 to Bit4 in the LEDShadow register are set to simulate a bargraph. For values 1 to 19, Bit0 is set. For values 20 to 39, Bit0 and Bit1 are set. For values 40 to 59, Bit0 to Bit2 are set. For values 60 to 79, Bit0 to Bit3 are set. For values 80 and above Bit0 to Bit4 are set.
- The function ControlOutputs is called. This function switches On or Off each LED based on the status of Bit0 to Bit4 of the LEDShadow variable.

APPENDIX CSD CALIBRATION

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



1. First start with the default settings of the CSD user module.
2. Using I2C-USB bridge or UART and the actual hardware and overlay capture the raw counts, baseline and difference counts for the sensors.
3. Check if the Raw counts of each sensor is within 60% to 70% of the full scale. Full scale counts is $2^n - 1$ where n is the resolution. If Raw counts it too high, reduce Rfb. If Raw counts is too low, increase Rfb. Raw counts can also be controlled by adjusting the Ref Value parameter. But at this stage, do not use RefValue parameter as this parameter is to be adjusted in Step#5.
4. Coarse Tuning. Check if Signal to Noise Ratio is greater than 5. If SNR is less than 5, increase SNR by following recommended PCB guidelines, increasing the resolution of the CSD and reducing the scan speed of the CSD. For PCB guidelines refer Cypress application note [CapSense™ Best Practices - AN2394](#). For details about Signal to Noise ratio and how to measure SNR, refer Cypress application note [Capacitance Sensing - Signal-to-Noise Ratio Requirement for CapSense Applications - AN2403](#)
5. Fine Tuning. Check if SNR is greater than 8. If less than 8, reduce the Ref Value parameter to increase SNR.
6. Check if total scan time for all the sensors meets requirement. If not, reduce resolution and / or increase scan speed. As 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.
7. Capture the difference counts when the button is activated. Set the finger threshold parameter to 75% of the peak.
8. Set the Noise threshold to 40% of the peak value.
9. Set the Negative noise threshold to half the Noise threshold.

10. Set finger thresholds for individual sensors if necessary. This can be done by writing to the `CSD_baBtnFThreshold` array in firmware.
11. Set the Baseline update threshold as per requirements. The frequency that the baseline is updated must be determined on a project-to-project basis. The point of a baseline is to 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 potentially be problematic 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."
12. Set AutoReset and Debounce parameters as required. Refer CSD user module data sheet for details of these parameters.
13. For all other parameters refer the user module data sheet.

Copyright 2009, PlanetPSoc.com

Permission is hereby granted, free of charge, to any person or organization obtaining a copy of the software and accompanying documentation covered by this license (the "Software") to use, reproduce, display, distribute, execute, and transmit the Software, and to prepare derivative works of the Software, and to permit third-parties to whom the Software is furnished to do so, all subject to the following:

The copyright notices in the Software and this entire statement, including the above license grant, this restriction and the following disclaimer, must be included in all copies of the Software, in whole or in part, and all derivative works of the Software, unless such copies or derivative works are solely in the form of machine-executable object code generated by a source language processor.

THE SOFTWARE AND THIS DOCUMENTATION ARE PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, TITLE AND NON-INFRINGEMENT. IN NO EVENT SHALL THE COPYRIGHT HOLDERS OR ANYONE DISTRIBUTING THE SOFTWARE BE LIABLE FOR ANY DAMAGES OR OTHER LIABILITY, WHETHER IN CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.