Please note that Cypress is an Infineon Technologies Company.
The document following this cover page is marked as “Cypress” document as this is the company that originally developed the product. Please note that Infineon will continue to offer the product to new and existing customers as part of the Infineon product portfolio.

Continuity of document content
The fact that Infineon offers the following product as part of the Infineon product portfolio does not lead to any changes to this document. Future revisions will occur when appropriate, and any changes will be set out on the document history page.

Continuity of ordering part numbers
Infineon continues to support existing part numbers. Please continue to use the ordering part numbers listed in the datasheet for ordering.

THIS SPEC IS OBSOLETE

Spec No: 001-44207

Spec Title: AN44207 - CAPSENSE EXPRESS(TM): APIS FOR REGISTER CONFIGURATION

Replaced by: None

CapSense Express™: APIs for Register Configuration

Associated Project: Yes
Associated Part Family: CY8C201xx
Software Version: PSoC Designer™ 5.0 SP6
Related Application Notes: For a complete list, click here.

To get the latest version of this application note, or the associated project file, please visit http://www.cypress.com/go/AN44207.

AN44207 defines APIs for register configuration. These APIs are used to write to CapSense Express™ registers through the host microcontroller when the CapSense Express slave device is on board.

Contents

1 Introduction

The CapSense Express device provides multifunctional, flexible, cost effective solutions for low IO capacitive sensing applications. It supports IOs configurable as capacitive sensing inputs or as GPIOs for LED drive, interrupt output, wakeup on interrupt input, and other digital IO functionalities. This device supports the I²C™ serial communication interface.

The CapSense Express device is an I²C slave device. After the device is placed onto the system, it is possible to reconfigure the device parameters through the host microcontroller, provided the host device can communicate through an I²C bus.

This application note provides APIs that are called directly from the host microcontroller to configure or modify CapSense Express slave device parameters. These APIs contain code for setting different parameters, which is achieved by writing the required values to appropriate registers. It is not necessary to take the device out of the system or make any external connections to write into the device registers and change its configuration. The values are written directly by the host microcontroller which calls these APIs through its code.

APIs are used to configure the device, read back CapSense values, drive Outputs, and enable sleep.

The APIs described in this application note are divided into two categories according to the operations they perform:

- High level APIs
- Low level APIs

2 High Level APIs

High level APIs contain code to configure different parameters of the CapSense Express device. APIs perform different functions such as enabling or disabling a CapSense pin or a GPIO, setting CapSense button or slider parameters, configuring sleep, reading back CapSense values, changing I^2C device address, and reading and writing to a particular port.

Every high level API contains code to set the required parameter by writing to the appropriate CapSense Express register. These APIs call the host IC hardware specific low level APIs. They then implement the physical I^2C communication between the host microcontroller and the CapSense Express slave device.

The code in high level APIs is not hardware specific. The same APIs can be used in different hardware environments without changing the code.

High level APIs are described in the following sections.

1	**Prototype**	**CE_BOOL CE_fEnableGPIO(CE_BYTE bPort , CE_BYTE bPin , CE_BYTE bEnable);**		
	Description	Enable or disable the specified port pin as GPIO. A pin already enabled as CapSense input cannot be enabled as GPIO using the API. The API returns a value of 0 if the pin cannot be enabled as GPIO; otherwise, it enables or disables the pin and returns 1. The CapSense Express device must be in Setup mode before enabling GPIO.		
	Parameters	Name	Description	Possible Values
		bPort	Port number	0, 1
		bPin	Pin number	0, 1, 2, 3, 4
		bEnable	To enable or disable the selected GPIO	CE_ENABLE , CE_ DISABLE
	Example	CE_fEnableGPIO (0,1,CE_ENABLE) enables port pin P0[1] as GPIO. CE_fEnableGPIO (0,1,CE_DISABLE) disables port pin P0[1].		
2	**Prototype**	**CE_BOOL CE_fEnableCapSense(CE_BYTE bPort , CE_BYTE bPin , CE_BYTE bEnable);**		
	Description	Enable or disable the specified port pin as CapSense input. A pin already enabled as GPIO cannot be enabled as CapSense using the API. The API returns a value of 0 if the pin cannot be enabled as CapSense; otherwise, it enables or disables the pin and returns 1. Make sure that the API is called before calling the API for setting CapSense Finger Threshold and IDAC setting; otherwise, the values of Finger Threshold and IDAC setting are reset. The CapSense express device must be in Setup mode before enabling CapSense.		
	Parameters	Name	Description	Possible Values
		bPort	Port number	0, 1
		bPin	Pin number	0, 1, 2, 3, 4
		bEnable	To enable or disable the selected CapSense input	CE_ENABLE, CE_DISABLE
	Example	CE_fEnableCapSense (0,1,CE_ENABLE) enables port pin P0[1] as CapSense. CE_fEnableCapSense (0,1,CE_DISABLE) disables port pin P0[1].		
3	**Prototype**	**CE_BOOL CE_fEnableSlider(CE_BYTE bEnable);**		
	Description	Enable or disable the slider. The API returns a value of 0 if the slider is not enabled; otherwise, it enables or disables the slider and returns 1.		
	Parameters	Name	Description	Possible Values
		bEnable	To enable or disable the slider	CE_ENABLE , CE_DISABLE
	Example	CE_EnableSlider (CE_ENABLE) enables the slider. CE_EnableSlider (CE_DISABLE) disables the slider.		
	Example	CE_fEnableCapSense (0,1,CE_ENABLE) enables port pin P0[1] as CapSense. CE_fEnableCapSense (0,1,CE_DISABLE) disables port pin P0[1].		

4	**Prototype**	**void CE_SetDriveMode(CE_BYTE bPort , CE_BYTE bPin , CE_BYTE bDriveMode);**		
	Description	Sets the drive mode for the selected GPIO. Possible values of drive mode are Resistive Pull Up, Strong, High Impedance, and Open Drain Low. Note that the device must be in Setup mode before setting the drive mode.		
	Parameters	Name	Description	Possible Values
		bPort	Port number	0, 1
		bPin	Pin number	0, 1, 2, 3, 4
		bDriveMode	Drive mode for the selected GPIO	CE_PULL_UP , CE_ STRONG , CE_HIGH_IMPEDANCE , CE_OPEN_DRAIN_LOW.
	Example	CE_SetDriveMode (0,1,STRONG) sets the drive mode for P0 [1] as Strong.		
5	**Prototype**	**void CE_SetPortInversion(CE_BYTE bPort , CE_BYTE bPin , CE_BYTE bPortInversion);**		
	Description	Inverts the logic of the input ports and status since last read registers.		
	Parameters	Name	Description	Possible Values
		bPort	Port number	0, 1
		bPin	Pin number	0, 1, 2, 3, 4
		bPortinversion	Inversion logic for the selected pin	CE_INVERT ,CE_ NORMAL
	Example	CE_SetPortInversion (0,1,CE_INVERT) inverts the logic of port pin P0[1]. CE_SetPortInversion (0,1,CE_NORMAL) makes the logic of the pin P0[1] as Normal (not inverted).		
6	**Prototype**	**void CE_SetInterruptMask(CE_BYTE bPort , CE_BYTE bPin , CE_BYTE bInterruptMask);**		
	Description	Sets which GPIO (not CapSense pins) generates hardware interrupts on 0-1 and 1-0 transitions. Its main purpose is to enable wake up from sleep on GPIO changes. Note that the CapSense Express device must be in Setup mode prior to setting the interrupt mask for GPIO.		
	Parameters	Name	Description	Possible Values
		bPort	Port number	0, 1
		bPin	Pin number	0, 1, 2, 3, 4
		bInterruptMask	To set the interrupt mask for the selected pin	CE_ON , CE_OFF
	Example	CE_SetInterruptMask (0, 1, CE_ON) enables the interrupt for pin P0[1]. CE_SetInterruptMask (0, 1, CE_OFF) disables the interrupt for pin P0[1].		
7	**Prototype**	**void CE_SetLatchDirection(CE_BYTE bPort , CE_BYTE bPin , CE_BYTE bLatchDirection);**		
	Description	Defines which value (0 or 1) is held in Port status since last read registers.		
	Parameters	Name	Description	Possible Values
		bPort	Port number	0, 1
		bPin	Pin number	0, 1, 2, 3, 4
		bLatchDirection	Latch direction for the selected GPIO	CE_FALLING (latch '0') , CE_RISING (latch '1')
	Example	CE_SetLatchDirection (0, 2, CE_RISING) latches '1' in the bit corresponding to P0[2] in Port status since last read register for Port 0. CE_SetLatchDirection (0, 2, CE_FALLING) latches '0' in the bit corresponding to P0[2] in Port status since last read register for Port 0.		

8	**Prototype**	**void CE_SetFingerThreshold(CE_BYTE bPort , CE_BYTE bPin , CE_BYTE bFingerThreshold);**		
	Description	Sets the value of Finger Threshold for the selected CapSense input. The API must be called after calling the APIs for enabling CapSense, setting settling time, enabling sensor auto reset, enabling external capacitor, and selecting clock; otherwise, the Finger Threshold value is reset.		
	Parameters	Name	Description	Possible Values
		bPort	Port number	0, 1
		bPin	Pin number	0, 1, 2, 3, 4
		bFingerThreshold	Finger Threshold value	3–255
	Example	CE_SetFingerThreshold (1, 2, 100);		
9	**Prototype**	**void CE_SetIDACSetting(CE_BYTE bPort , CE_BYTE bPin , CE_BYTE bIDACSetting);**		
	Description	Sets the value of IDAC setting for the selected CapSense pin. The API must be called after calling the APIs for enabling CapSense, setting settling time, enabling sensor auto reset, enabling external capacitor, and selecting clock; otherwise, the IDAC setting value is reset.		
	Parameters	Name	Description	Possible Values
		bPort	Port number	0, 1
		bPin	Pin number	0, 1, 2, 3, 4
		bIDACSetting	IDAC setting value	1–255
	Example	CE_SetIDACSetting (1, 2,10);		
10	**Prototype**	**void CE_SetThresholds(CE_BYTE bNoiseThreshold , CE_BYTE bBaselineUpdateThreshold , CE_BYTE bNegativeNoiseThreshold , CE_BYTE bLowBaselineReset);**		
	Description	Sets the values of Noise Threshold, Baseline Update Threshold, Negative Noise Threshold, and Low Baseline Reset for the CapSense system.		
	Parameters	Name	Description	Possible Values
		bNoiseThreshold	Noise Threshold value	3 to 255
		bBaselineUpdateThreshold	Baseline Update Threshold value	0 to 255
		bNegativeNoiseThreshold	Negative Noise Threshold value	0 to 255
		bLowBaselineReset	Low Baseline Reset value	0 to 255
	Example	CE_SetThresholds (40,100,20,20);		
11	**Prototype**	**void CE_SetSettlingTime(CE_BYTE bSettlingTime);**		
	Description	Sets the value of the settling time. Make sure that the API is called before calling the API for setting CapSense Finger Threshold and IDAC setting. Otherwise, the values of Finger Threshold and IDAC setting are reset. Also note that the CapSense Express device must be in Setup mode before setting the settling time.		
	Parameters	Name	Description	Possible Values
		bSettlingTime	Settling time value	2–255
	Example	CE_SetSettlingTime (160);		
12	**Prototype**	**void CE_SetHysteresis(CE_BYTE bHysteresis);**		
	Description	Sets the value of Hysteresis.		
	Parameters	Name	Description	Possible Values
		bHysteresis	Hysteresis value	0 to 255
	Example	CE_SetHysteresis (20);		

13	**Prototype**	**void CE_SetDebounce(CE_BYTE bDebounce);**		
	Description	Sets the value of Debounce counter.		
	Parameters	Name	Description	Possible Values
		bDebounce	Debounce counter value	1 to 255
	Example	CE_SetDebounce (3);		
14	**Prototype**	**void CE_EnableSensorAutoReset(CE_BYTE bEnable);**		
	Description	Decides whether to enable the sensor auto reset or not. If the baseline must be updated at all times, then pass CE_ENABLE as the parameter to the API. If the baseline must be updated only when the signal difference is below the noise threshold, then pass CE_DISABLE as the parameter to the API. Before enabling or disabling the sensor auto reset, the Capsense Express device must be in Setup mode.		
	Parameters	Name	Description	Possible Values
		bEnable	To enable or disable sensor auto reset	CE_ENABLE, CE_DISABLE
	Example	CE_EnableSensorAutoReset(CE_ENABLE) enables sensor auto reset. CE_EnableSensorAutoReset(CE_DISABLE) disables sensor auto reset.		
15	**Prototype**	**Void CE_EnableExternalCapacitor(CE_BYTE bEnable);**		
	Description	Selects whether to use the external integrating capacitor or internal capacitor. To configure this option, the device must be in Setup mode.		
	Parameters	Name	Description	Possible Values
		bEnable	To enable or disable the external integrating capacitor	CE_ENABLE, CE_DISABLE
	Example	CE_EnableExternalCapacitor (CE_ENABLE) enables the external integrating capacitor. CE_EnableExternalCapacitor (CE_DISABLE) enables the internal capacitor.		
16	**Prototype**	**void CE_SelectClock(CE_BYTE bClock);**		
	Description	Selects the CapSense module frequency of operation. The frequency can be set to IMO, IMO/2, IMO/4, or IMO/8. Similar to the previous API, this API must only be called after setting the values of Finger Threshold and IDAC setting for each CapSense button; otherwise, the values of Finger Threshold and IDAC setting are reset. To set the clock frequency, the CapSense Express device must be in Setup mode.		
	Parameters	Name	Description	Possible Values
		bClock	CapSense module frequency of operation	CE_IMO ,CE_ IMO_BY_2, CE_ IMO_BY_4, CE_ IMO_BY_8
	Example	CE_SelectClock (CE_ IMO); sets the CapSense module frequency of operation to IMO. CE_SelectClock (CE_IMO_BY_4); sets the CapSense module frequency of operation to IMO/4.		

17 (a)	**Prototype**	**void CE_SetSliderResolution1(CE_ WORD wSliderResolution);**		
	Description	Sets the value of slider resolution to any value from 10 to 1000.		
	Parameters	Name	Description	Possible Values
		WSliderResolution	Slider resolution value	10 to 1000
	Example	CE_SetSliderResolution1(100);		
17 (b)	**Prototype**	**void CE_SetSliderResolution(CE_BYTE bSliderResolution);**		
	Description	Sets the value of the slider resolution to any value among 50, 100, 150, 200, and 250. This API is specially written for some commonly used values of resolution. Calling this API, the slider resolution can be set to any of the values among 50, 100, 150, 200, and 250. The advantage of using this API is that it contains a table look up for these values and so the time to execute the API reduces drastically. For these values, use this API. If you need to set the slider resolution to any other value, then you can use the other API described earlier. In the .c file provided with this application note, the slider API described in 17(a) is commented; uncomment the API before using it.		
	Parameters	Name	Description	Possible Values
		bSliderResolution	Slider resolution value	50,100,150,200,250.
	Example	CE_SetSliderResolution (100);		
18	**Prototype**	**void CE_ReadButtonValues (CE_BYTE bPort , CE_BYTE bPin , CE_BYTE * abButtonReadBackValues);**		
	Description	Reads the values of Baseline, Sensor Difference, and Result (Raw Count) for the specified CapSense button. A pointer to a 6 byte array is passed as parameter to the API which receives the values of Baseline, Sensor Difference, and Result in order.		
	Parameters	Name	Description	Possible Values
		bPort	Port number	0,1
		bPin	Pin number	0,1,2,3,4
		abButtonReadBackValues	Pointer to the 6 byte array	-
	Example	CE_ReadButtonValues (1, 4, abButtonReadBackValues) reads the values for the button connected at P1[4] and passes these values to abButtonReadBackValues (pointer to the 6 byte array).		

19	**Prototype**	**void CE_ReadButton(CE_BYTE bPort , CE_BYTE bPin , CE_BYTE bValue, CE_BYTE *abButtonReadBackValue);**		
	Description	Reads the value of Baseline, Sensor Difference, or Result for the selected CapSense button. A pointer to a 2 byte array is passed as parameter to the API which receives the read value.		
	Parameters	Name	Description	Possible Values
		bPort	Port number	0,1
		bPin	Pin number	0,1,2,3,4
		bValue	Value to read Baseline, Sensor Difference, or Result	CE_BASELINE,CE_RESULT, CE_SENSOR_DIFFERENCE, CE_SENSOR_ON_STATUS
		abButtonReadBackValue	Pointer to the 2 byte array	-
	Example	CE_ReadButton (1, 4, CE_BASELINE, abButtonReadBackValue); reads the value of BASELINE for the button connected at P1 [4] and passes this value to abButtonReadBackValue (pointer to the 2 byte array). CE_ReadButton(1,4,CE_RESULT, abButtonReadBackValue); reads the value of RESULT (Raw Count) for button connected at P1 [4] and passes this value to abButtonReadBackValue (pointer to the 2 byte array).		
20	**Prototype**	**void CE_ReadSliderValues(CE_BYTE *abSliderReadBackValues);**		
	Description	Reads the values of Centroid Position and Centroid Peak. A pointer to a 4 byte array is passed as parameter to the API which receives the values of Centroid Position and Centroid Peak, in order.		
	Parameters	Name	Description	Possible Values
		abSliderReadBackValues	Pointer to the 4 byte array	-
	Example	CE_ReadSliderValues (abSliderReadBackValues); receives the values of Centroid Position and Centroid Peak in order and passes it to abSliderReadBackValues.		
21	**Prototype**	**void CE_ReadDeviceIDStatus(CE_BYTE *abDeviceIDStatus);**		
	Description	Reads device ID and status and passes the values to a pointer to a 2 byte array.		
	Parameters	Name	Description	Possible Values
		abDeviceIDStatus	Pointer to the 2 byte array	-
	Example	CE_ReadDeviceIDStatus(abDeviceIDStatus)		
22	**Prototype**	**CE_BYTE CE_bReadInputPort(CE_BYTE bPort);**		
	Description	Reads input port and returns the value read.		
	Parameters	Name	Description	Possible Values
		bPort	Port number	0,1
	Example	CE_bReadInputPort (0); reads Port 0 and returns the value read. CE_bReadInputPort (1); reads Port 1 and returns the value read.		
23	**Prototype**	**CE_BYTE CE_bReadPortStatus(CE_BYTE bPort);**		
	Description	Reads the value in port status since last read registers and returns the value read.		
	Parameters	Name	Description	Possible Values
		bPort	Port number	0,1
	Example	CE_ bReadPortStatus (0) returns the value of port status since last register for Port 0. CE_ bReadPortStatus (1) returns the value of port status since last register for Port 1.		

24	**Prototype**	**void CE_EnableOutputLogicOperation (CE_BYTE bPort , CE_BYTE bPin , CE_BYTE bEnable);**		
	Description	The function of this API is to decide whether to perform the operation as specified by the operation select register on the specified GPIO. To enable the logic operation as specified by the operation select register of the specified pin, pass CE_ENABLE as parameter to the API; to disable, pass CE_DISABLE.		
	Parameters	Name	Description	Possible Values
		bPort	Port number	0,1
		bPin	Pin number	0,1,2,3,4
		bEnable	Enable or disable	CE_ENABLE,CE_DISABLE
	Example	CE_EnableOutputLogicOperation (0,0, CE_ENABLE) ; sets the logic at P0[0] as defined by the operation select register for the selected GPIO.		
25	**Prototype**	**void CE_SetOutputLogic(CE_BYTE bPort , CE_BYTE bPin , CE_BYTE bLogic);**		
	Description	Writes Logic 0 or Logic 1 to the specified GPIO.		
	Parameters	Name	Description	Possible Values
		bPort	Port number	0,1
		bPin	Pin number	0,1,2,3,4
		bLogic	Logic 0 or Logic 1	CE_LOGIC_0 , CE_LOGIC_1
	Example	CE_SetOutputLogic(0,0,CE_LOGIC_1); writes LOGIC 1 to P0 [0]. CE_SetOutputLogic(0,0,CE_LOGIC_0); writes LOGIC 0 to P0 [0].		
26	**Prototype**	**void CE_EnterCommandCode(CE_BYTE bCommandCode);**		
	Description	Writes the specified command code to command register.		
	Parameters	Name	Description	Possible Values
		bCommandCode	Command code	'GET_FIRMWARE_REVISION' , 'STORE_CURRENT_CONFIGURATION_TO_NVM', 'RESTORE_FACTORY_CONFIGURATION', 'WRITE_POR_DEFAULTS', 'READ_POR_DEFAULTS', 'READ_DEVICE_CONFIGURATION', 'RECONFIGURE_DEVICE', 'NORMAL_OPERATION_MODE', 'SETUP_OPERATION_MODE',
	Example	CE_EnterCommandCode (NORMAL_OPERATION_MODE); device enters into Normal operation mode. CE_EnterCommandCode (SETUP_OPERATION_MODE); device enters into Setup operation mode. For details on the functions performed by other values for parameter bCommandCode, refer to the CY8C201xx Register Reference Guide.		

27	**Prototype**	**void CE_ConfigureSleepParameters(CE_BYTE bPort , CE_BYTE bPin , CE_BYTE bReference , CE_BYTE bSleepInterval, CE_BYTE bStayAwakeCounter);**		
	Description	The API performs the following functions: 1. Selects sleep control pin 2. Selects reference 3. Sets sleep interval 4. Sets stay awake counter The CapSense Express device must be in Setup mode to execute this API properly.		
	Parameters	Name	Description	Possible Values
		bPort	Port number	0,1
		bPin	Pin number	0,1,2,3,4
		bReference	Voltage bandgap powered or not	CE_ENABLE , CE_DISABLE
		bSleepInterval	Sleep interval value	CE_SLEEP_INTERVAL_1_95 (1.95 ms), CE_SLEEP_INTERVAL_15_6 (15.6 ms), CE_SLEEP_INTERVAL_125 (125 ms), CE_SLEEP_INTERVAL_1000 (1s).
		bStayAwakeCounter	Stay awake counter value	0–255
	Example	CE_ConfigureSleepParameters (1, 2,CE_ENABLE,CE_SLEEP_INTERVAL_1_95, 100); Sets P1 [2] as sleep control pin, enables voltage bandgap reference, sets sleep interval as 1.95 ms, and sets the stay awake counter to 100.		
28	**Prototype**	**void CE_GoToSleep(CE_BYTE bMode);**		
	Description	Sets the device in sleep (normal sleep mode or deep sleep mode).		
	Parameters	Name	Description	Possible Values
		bMode	To select the sleep mode	CE_NORMAL_SLEEP_MODE, CE_DEEP_SLEEP_MODE
	Example	CE_GoToSleep (CE_NORMAL_SLEEP_MODE); sets the device in normal sleep mode.		
29	**Prototype**	**void CE_DisableSleep(void);**		
	Description	After calling this API the device does not enter into sleep again unless the sleep mode is again set.		
	Parameters	None		
	Example	CE_DisableSleep ( );		
30	**Prototype**	**CE_BOOL CE_SetDeviceI2CAddress(CE_BYTE bNewDeviceAddress , CE_BYTE bI2CPinDriveMode);**		
	Description	The function of this API is to set the device I2C address. The address is set in the range 0 to 127. The I2C pin drive mode is set either as ‘Resistive Pull up’ or ‘Open Drain Low’. If the address passed is greater than 127, then the API returns 0 (fail); otherwise, it sets the new address and returns 1 (pass).		
	Parameters	Name	Description	Possible Values
		bNewDeviceAddress	I2C device address	0–127
		bI2CPinDriveMode	I2C pin drive mode	RESISTIVE_PULL_UP, OPEN_DRAIN_LOW
	Example	CE_SetDeviceI2CAddress (5, OPEN_DRAIN_LOW); sets the device’s new address as 5, sets the I2C pin drive mode as Open Drain Low, and returns 1.		

31	**Prototype**	**CE_BOOL CE_PWMEnableOutputLogic (CE_BYTE bPort, CE_BYTE bPin, CE_BOOL PWM_Enable)**		
	Description	The function of this API is to enable PWM output and select the GPIO to route the PWM output. It returns 1 if the selected GPIO is enabled for PWM output; otherwise, it returns 0. Before selecting any port pin as PWM output it should be enabled as GPIO.		
	Parameters	Name	Description	Possible Values
		bPort	Port number	0, 1
		bPin	Pin number	0, 1, 2, 3, 4
		PWM_Enable	PWM output enable	0, 1 (CE_DISABLE, CE_ENABLE)
	Example	CE_BOOL CE_PWMEnableOutputLogic (0, 1, 1) In the above API, the port 0 pin 1 is enabled for PWM output. This returns 1 if the GPIO is selected as PWM output; otherwise, the return value is 0.		
32	**Prototype**	**void CE_SetPWMModeDutyDelay (CE_BYTE Mode, CE_BYTE Duty, CE_BYTE EnableDelay, CE_BYTE Delay, CE_BOOL Link)**		
	Description	The function of this API is to set PWM parameters such as mode selection and duty cycle selection for dimming, It also enables the delay and delay value for the dimming functionality and the link parameter.		
	Parameters	Name	Description	Possible Values
		Mode	Mode of operation	0, 1, 2, 3
		Duty	Sets the duty cycle for dimming	0–15. Here 0 = 2.5% and 15 = 99.5%
		EnableDelay	Enables the delay value for the PWM. This delays the dimming functionality.	0, 1 (CE_DISABLE, CE_ENABLE)
		Delay	Specifies the delay for dimming functionality. If Enable Delay is not set, then this has no effect.	0–254. The value of 255 is undefined. 0 indicates delay of 0ms 254 indicates delay of 5s
		Link	Link between 0/1 logic state of the pin and the PWM output.	0, 1 (CE_DISABLE, CE_ENABLE)
	Example	CE_SetPWMModeDutyDelay (2, 0, 1, 254, 0) sets the mode 2 operation of PWM, where the GPIO goes off, after specified delay time of 5s, after the button is pressed. The link parameter is set to zero.		
33	**Prototype**	**void CE_EnableFilter(CE_BOOL RstBL, CE_BOOL EnableDTS, CE_BOOL EnableAvg, CE_BYTE NoofSamples)**		
	Description	This function is used to set the required filtering such as Averaging and Drop The Sample filtering. This is used to improve the noise performance.		
	Parameters	Name	Description	Possible Values
		RstBL	This bit set at '1' forces an immediate baseline initialization for all sensors	0, 1
		EnableDTS	This bit set at '1' enables the I2C "Drop The Sample" filter	0, 1
		EnableAvg	This bit enables or disables the Averaging filter	0, 1
		NoofSamples	These two bits are used to select the number of CapSense samples to average	0, 1, 2, 3 Here 0 is for 2 sample average 1 is for 4 sample average 2 is for 8 sample average 3 is for 16 sample average
	Example	CE_EnableFilter (1, 1, 1, 0) forces immediate baseline initialization for all sensors, enables Drop the Sample filter, enables Averaging filter, and uses two samples for averaging.		

In addition to the above APIs, there is one more API that is called to write the complete I2C configuration in one step. Calling this API, the I2C file generated by the CapSense Express Software Tool (refer AN42137) can be written as it is to the device. This is a text file containing a string with hexadecimal values for the specific configuration selected in PSoC Designer™. This string must be arranged in a character array of the required size. The pointer to this array must be passed to the API. The API receives each character one by one and performs the required operation. This is useful to change the entire configuration of the device in one step.

34	Prototype	void CE_WriteI2Cfile(const char * acI2CFile);		
	Description	The API is used to write the whole I2C file generated by PSoC Designer™ in a single step. You can store all the configuration values together in a character array of required size. The pointer to that array is passed as a parameter to the API. The API receives each character from the array one by one and performs the required action.		
	Parameters	Name	Description	Possible Values
		acI2CFile	Pointer to the character array where the set of values are stored	-
	Example	Define the array in main file, using the following method: const char acI2Cfile [ ] = "w 00 79 3C A5 69 w 00 7C 80 w 00 79 96 5A C3 w 00 A0 08……" Then call the API: CE_WriteI2Cfile (acI2Cfile); writes the whole set of commands.		

3 Low Level APIs

Low level APIs are used in the host IC to enable the physical I^2C read and write communication between the host microcontroller and the CapSense Express device.

They contain hardware specific code to read and write to or from the I^2C slave device. The code depends on the system hardware and the CPU. An example of low level APIs for PSoC is provided with this application note.

In the low level APIs provided with this application note, the PSoC I2CHW user module is used to perform read and write operations and the code is written accordingly.

Low level APIs are described in the following sections.

There are two APIs in this category. One API is to read from the CapSense Express device and the other is to write to the device. As mentioned earlier, these APIs are hardware specific. The following APIs are for PSoC. These APIs must be changed according to the system specific hardware.

1	**Prototype**	**CE_Read(CE_BYTE bDeviceAddress , CE_BYTE bAddress , CE_BYTE bBytesToBeRead , CE_BYTE *RdPrtAddress);**		
	Description	Reads the specified number of bytes from the CapSense Express device.		
	Parameters	Name	Description	Possible Values
		bDeviceAddress	I2C device address	0–127
		bAddress	Address of the location from where to start reading	-
		bBytesToBeRead	No. of bytes to read	-
		RdPrtAddress	Pointer to the array where to store the data read	-
	Example	CE_Read(0, 10, 5,bReadBuffer); reads 5 bytes from the device having address 0, starting from RAM address 10h, and stores the values to the array 'bReadBuffer'		
2	**Prototype**	**CE_Write(CE_BYTE bDeviceAddress , CE_BYTE *bArray , CE_BYTE bBytesToBeWritten);**		
	Description	Writes the specified number of bytes to the CapSense Express device.		
	Parameters	Name	Description	Possible Values
		bDeviceAddress	I2C device address	0–127
		bArray	Pointer to the array of bytes where data is stored	-
		bBytesToBeWritten	No. of bytes to write	-
	Example	CE_Write (0, bWriteBuffer, 5); writes 5 bytes taken from the array 'bWriteBuffer' to the I2C device with address 0.		

4 Command Execution

Commands are executed in CapSense Express device by writing the command code to register 0xA0. The device cannot be accessed for the time needed to execute the command. The time for which the device cannot be accessed, after execution of each command, is given in the table below:

Command	Description	Execution Mode	Expected ACK Time in Normal Mode	Device Inaccessible Time after ACK
00h	Get Firmware Revision	Setup or Normal	<100 µs	0s
01h	Store Current Configuration to NVM	Setup or Normal	<100 µs	~120 ms
02h	Restore Factory Configuration	Setup or Normal	<100 µs	~120 ms
03h	Write NVM POR Defaults	Setup or Normal	<100 µs	~120 ms
04h	Read NVM POR Defaults	Setup or Normal	<100 µs	~5 ms
05h	Read Active Device Configuration (RAM)	Setup or Normal	<100 µs	~5 ms
06h	Reconfigure Device (POR)	Setup	<100 µs	~5 ms (Setup mode)
07h	Set Normal Operation Mode	Setup or Normal	<100 µs	0s
08h	Set Setup Operation Mode	Setup or Normal	<100 µs	1.2 * (loop time[1] + 1 ms)
09h	Start CapSense Scan	Setup or Normal	<100 µs	~10 ms
0Ah	Stop CapSense Scan	Setup or Normal	<100 µs	~5 ms
0Bh	Get CapSense Scan Status	Setup or Normal	<100 µs	0s

[1] Loop time can be measured by probing any sensor using an oscilloscope and measuring the time between two consecutive scans.

Accessing the device immediately after the command is sent does not affect the functionality of the system. This is because CapSense Express simply does not respond to I^2C requests.

The following files are provided with this application note:

- ce_201xx_high_api.c

  This file contains the definition for all high level APIs that are used to configure the various device parameters.

- ce_201xx_low_api.c

  This file contains low level APIs. The code is written for PSoC master. The code in these APIs must be changed for other microcontrollers, as explained in the previous sections.

- ce_201xx.h

  This file contains the prototype for all APIs, the system types used, and the constants used in the APIs

5 Important Notes

- The device address is stored in a CE_BYTE variable (bDeviceAddress) in the .c file provided with this application note. The address is initialized to 0x00. The low level APIs perform the read and write operations on the address specified by this variable. If the address of CapSense Express device in the system is not 0x00 or if you change the address by calling the API for changing the device address, then reinitialize this variable to a new value in .c file on the next power on or reset.
- For any parameter, pass only the values which are mentioned in the application note. Passing any value outside the range causes unexpected behavior.
- Always make sure that before writing Logic 0 or Logic 1 to any GPIO, the output logic operation as specified by the operation select register is disabled. To do this, call the following API:

  CE_EnableOutputLogicOperation (CE_DISABLE) first and then call the API, CE_SetOutputLogic(CE_LOGIC_1) or CE_SetOutputLogic (CE_LOGIC_0).

- The two APIs CE_SetFingerThreshold and CE_SetIDACSetting must be called only after calling the following APIs:

  CE_fEnableCapSense,

  CE_SetsettlingTime,

  CE_EnableSensorAutoReset,

  CE_EnableExternalCapacitor,

  CE_SelectClock,

  Otherwise, the values of Finger Threshold and IDAC setting are reset.

- To retain the configuration changes made, call API: CE_EnterCommandCode(STORE_CURRENT_CONFIGURATION_TO_NVM) after all other APIs are called. This retains the changes made on the next power on or reset.
- The low level APIs for PSoC call the User module (I2CHardWare) inside it. When placing user module into the project, rename the module as I2CHW. Otherwise, to retain the module name according to the project, the user must update the user module name in the low level APIs:

  CE_Read (CE_BYTE bDeviceAddress, CE_BYTE bAddress, CE_BYTE bBytesToBeRead, CE_BYTE *RdPrtAddress)

  CE_Write (CE_BYTE bDeviceAddress, CEs_BYTE *bArray , CE_BYTE bBytesToBeWritten).

6 Summary

AN44207 defines APIs for register configuration. These APIs are used to write to CapSense Express™ registers through the host microcontroller when the CapSense Express slave device is on board.

7 Related Application Notes

AN42137 – CapSense™ Express Software Tool

AN44208 – CapSense® Express™ - I2C Communication Timing Analysis

AN44209 – CapSense® Express™ Power and Sleep Considerations

Document History

Document Title: AN44207 - CapSense Express™: APIs for Register Configuration

Document Number: 001-44207

Revision	ECN	Orig. of Change	Submission Date	Description of Change
**	2345367	MOHD	04/11/08	New application note.
*A	2559285	MOHD	08/29/08	Updated text in all instances across the document. Updated High Level APIs: Added two APIs. Added two APIs in attached associated project.
*B	3249521	SLAN	05/05/11	Updated Software Version as “PSoC Designer 5.0 SP6” in page 1. Updated attached associated project to PSoC Designer 5.0.
*C	4387056	PRIA	05/22/2014	Updated Command Execution: Updated details in “Device Inaccessible Time after ACK” column corresponding to “08h” Command in the table. Updated Related Application Notes: Removed references of AN44203, and AN47716 as these application notes are obsolete. Updated to new template. Completing Sunset Review.
*D	5031306	PRIA	12/01/2015	Updated to new template.
*E	5092581	PRIA	01/19/2016	Obsoleting the Application Note

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
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

Cypress Semiconductor
198 Champion Court
San Jose, CA 95134-1709

Phone : 408-943-2600
Fax : 408-943-4730
Website : www.cypress.com