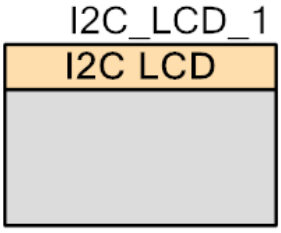
Character LCD with I2C Interface (I2C LCD)

1.20



Features

- Communicate on a 2-wire I^2C bus
- API compatible with the current character LCD component
- One component may drive one or more LCDs on the same I^2C bus
- Can coexist on an existing I^2C bus if the PSoC is the I^2C master
- Support for the NXP PCF2119x command format

General Description

The I^2C LCD component drives an I^2C interfaced 2 line by 16 character LCD. The I^2C LCD component is a wrapper around an I^2C Master component and makes use of an existing I^2C Master component. If a project does not already have an I^2C Master component, one is required in order to operate. When one of the API functions is called, that function calls one or more of the I^2C Master functions in order to communicate with the LCD.

When to Use an I^2C LCD

The I^2C LCD component is used in applications that require a visual or textual display. This component is also used where a character display is needed but seven consecutive GPIOs on a single GPIO port are not possible. In cases where the project already includes an I^2C master, no additional GPIO pins are required.

Input/Output Connections

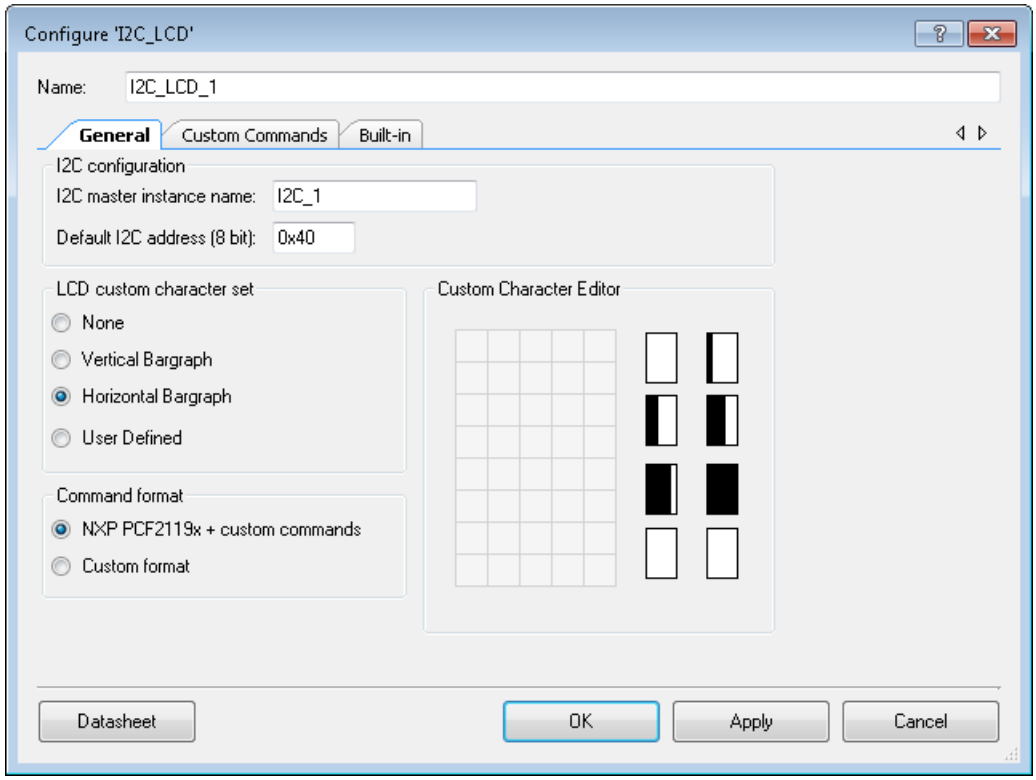
The component does not have any connection because it is a software component.

Component Parameters

Drag an I²C LCD onto your design and double-click it to open the Configure dialog.

General Tab

The **General** tab provides options to configure the general settings of the I²C LCD. These parameters are available.



I²C Master Instance Name

This parameter is set to the instance name of the I²C Master component name. An I²C master must reside in the schematic for this component to operate. It uses this name to access the I²C Master APIs. The LCD must also be connected to the I²C bus controlled by the I²C Master component. Allowed characters are A-Z, 0-9, _. The instance name cannot start or end with an underscore.

Default I²C Address

This parameter sets the default address of the I²C LCD. This address is used when connecting to the I²C Master's API to properly address the LCD. If multiple LCDs are placed on the bus, the LCD's I²C address may change during run time. By default this parameter is set to 0x40.

The address is an 8-bit value that is mapped into a 7-bit address by one bit right shift.

Note For the NXP LCD module, the I²C address should be set to 0x74.

LCD Custom Character Set

This parameter enables the selection of these options:

- **None** (Default) – Do not do anything with custom characters.
- **Vertical Bar Graph** – Generate custom characters and API to manipulate a vertical bar graph.
- **Horizontal Bar Graph** – Generate custom characters and API to manipulate a horizontal bar graph.
- **User Defined** –Create custom characters and API to manipulate them.

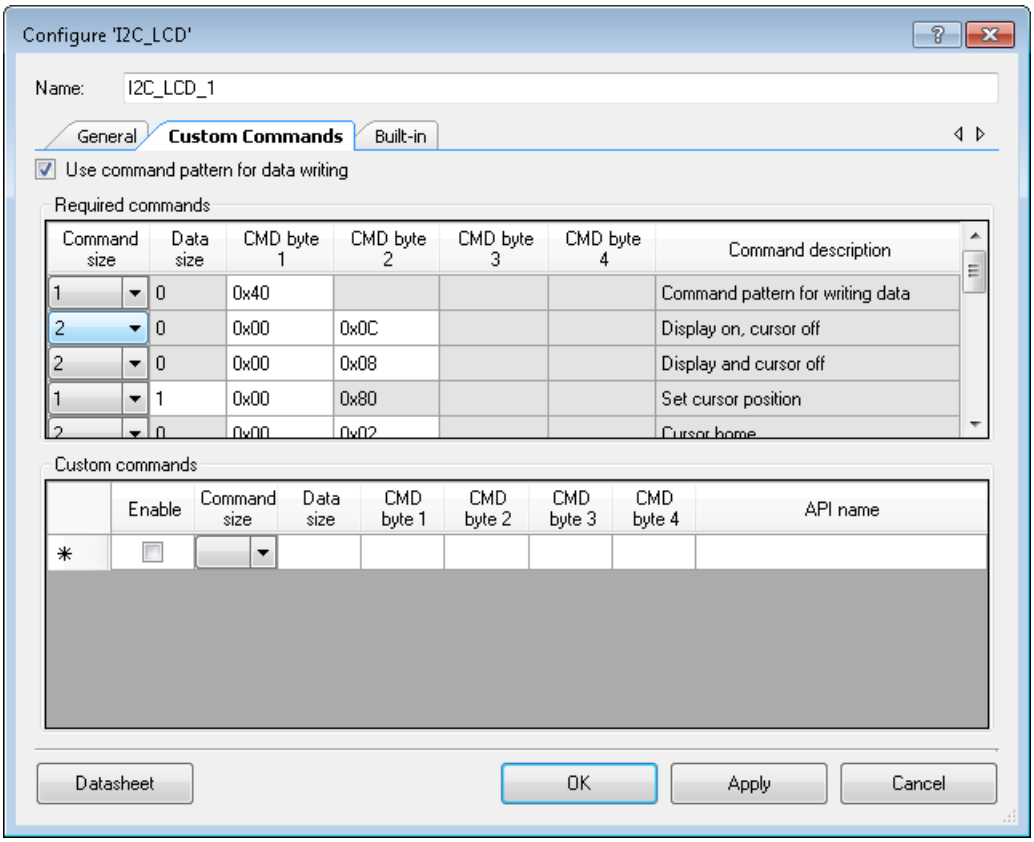
After the component has loaded in the characters, the I2C_LCD_PutChar() function and the custom character constants (from the header file) can be used to display them.

Command format

This parameter has two options. If **NXP PCF2119x + custom commands** is selected this means that component uses embedded support of NXP PCF2119x command protocol and also it allows user to enter any extended (custom) commands that NXP compatible LCD module supports. The custom commands can be entered in the **Custom Commands** tab. If you plan to use the LCD with a command set that is different than the NXP PCF2119x select **Custom format**. In this case, you enter the required commands in the **Custom Commands** tab. You can also enter any other commands supported by the specific LCD module in the **Custom commands** table.

Custom Commands Tab

This tab has a set of options that makes it possible to add custom command support to the component.



Use command pattern for data writing

This option controls whether a command pattern is sent with the data to display text on the LCD using functions such as I2C_LCD_PrintString(). When the box is not checked, the data to be displayed is sent directly. When the box is checked, a command pattern is sent along with the text. The NXP PCF2119x compatible modules require a command pattern, so this option is always checked for that mode of operation. For custom LCD module support, review the datasheet for the module to determine the correct setting for this option.

Required commands

This group box contains a list of mandatory commands that you must define for component operation. This table has a command set that is mandatory for NXP PCF2119x compatible LCD Modules.

#	Command Description
1	Write a byte or multiple bytes of data

#	Command Description
2	Display on, cursor off
3	Display and cursor off
4	Set cursor position
5	Cursor home
6	Reset cursor position
7	Move cursor left one place
8	Move cursor right one place
9	Underline cursor on
10	Display - on, cursor - off, set cursor wink
11	Display - on, cursor - off, set cursor blink
12	Clear screen
13	Load custom character
14	Set display for 2 lines and 16 characters
15	Move cursor/shift display right (cursor position auto increment)
16	Move cursor/shift display right (cursor position auto decrement)

When **NXP PCF2119x + custom commands option** is selected, the **Required commands** table is grayed out as patterns of required commands are already embedded into the component. In the case of a **Custom format** you need to enter proper command patterns for each required command. If some of commands are not supported by the specific LCD module you must enter a dummy command pattern.

Set cursor position

For the NXP PCF2119x compatible LCD modules the command codes from 0x80 and above are used for setting the address of Display Data RAM (DDRAM).For example, the address of 0x80 maps to row 0 and column 0 of the LCD module and the address 0x83 maps to row 0 and column 3. For detailed information on mapping of DDRAM to LCD location please see the LCD module datasheet.

Command Size

This parameter specifies the size (in bytes) of command sequence without parameters or actual data.

Data size

This parameter defines the size of data that is sent after the command sequence for the specified command.

CMD byte 1 – CMD byte 4

This parameter specifies the values of bytes #1 - #4 of command sequence. IF “Command Size” is less than four (the maximum size) the unused bytes are grayed out in the customizer.**CMD byte 2** defines the unique command code for the command sequence. This command code can be used as a parameter for I2C_LCD_WriteControl() or I2C_LCD_HandleCustom() functions.

Command description

This is short descriptions of what the command does. It helps to match command patterns from the customizer with commands from the LCD Module’s datasheet.

By default, when the **Custom commands** parameter is selected, the required commands are filled with command patterns for the NXP PCF2119x command format.

Custom commands

In this section you define custom command handlers for specific commands. You also have a mechanism to add new commands. The column definitions in this group box are the same as for **Required Commands** but there are two additional columns.

Enable

This option, when checked, enables the use of the respective command pattern in the design. It means that the API function is provided for the command pattern.

API name

This parameter specifies the name of API function that is generated (if enabled) for the command pattern. The API name specified in this column is appended with a component instance name at the beginning of the name. Also depending on the data size the API function will or will not have a parameter. For example, for the API name “**Func1**” with component instance name “I2C_LCD” and **Data size** equal to zero the API function has this prototype:

```
void I2C_LCD_Func1(void);
```

If **Data size** is 1, the prototype is:

```
void I2C_LCD_Func1(uint8 cmdByte);
```

If **Data size** is greater than 1, the prototype is:

```
void I2C_LCD_Func1(uint8 cmdData[]);
```

The size of cmdData[] is specified by **Data size**.

Note that the API name you type cannot be all capital letters. If they are, the component displays a warning about redefining the macro. This is because, for each custom command, the

component generates the #define constant for a command code that has the same name as one specified in **API name** that has all capital letters.

Application Programming Interface

Application Programming Interface (API) routines allow you to configure the component using software. This table lists and describes the interface to each function. The following sections cover each function in more detail.

By default, PSoC Creator assigns the instance name I2C_LCD_1 to the first instance of a component in a given design. You can rename it to any unique value that follows the syntactic rules for identifiers. The instance name becomes the prefix of every global function name, variable, and constant symbol. For readability, the instance name used in the following table is I2C_LCD.

Standard Functions

Functions	Description
I2C_LCD_Start()	Starts the module and loads custom character set to LCD if it was defined.
I2C_LCD_Stop()	Turns off the LCD
I2C_LCD_Init()	Performs initialization required for component's normal work
I2C_LCD_Enable()	Turns on the display
I2C_LCD_DisplayOn()	Turns on the LCD module's display
I2C_LCD_DisplayOff()	Turns off the LCD module's display
I2C_LCD_PrintString()	Prints a null-terminated string to the screen, character by character
I2C_LCD_PutChar()	Sends a single character to the LCD module data register at the current position.
I2C_LCD_Position()	Sets the cursor's position to match the row and column supplied
I2C_LCD_WriteData()	Writes a single byte of data to the LCD module data register
I2C_LCD_WriteControl()	Writes a single-byte instruction to the LCD module control register
I2C_LCD_ClearDisplay()	Clears the data from the LCD module's screen
I2C_LCD_SetAddr()	This function allows the user to change the default I^2C address of the LCD.
I2C_LCD_PrintInt8()	Prints a two-ASCII-character hex representation of the 8-bit value to the Character LCD module.
I2C_LCD_PrintInt16()	Prints a four-ASCII-character hex representation of the 16-bit value to the Character LCD module.

Functions	Description
I2C_LCD_PrintNumber()	Prints the decimal value of a 16-bit value as left-justified ASCII characters
I2C_LCD_HandleOneByteCommand()	This command adds a support of sending custom commands with 1 byte parameter
I2C_LCD_HandleCustomCommand()	Performs sending of the command that has variable parameters.

void I2C_LCD_Start(void)

Description: When this function called first time it initializes the LCD hardware module as follows:

- Turns on the display;
- Enables auto cursor increment;
- Resets the cursor to start position;
- Clears the display;
- It also loads a custom character set to LCD if it was defined in the customizer's GUI. Resets the cursor to start position.

All of the following call to this function will just turn on the LCD module.

Note This function sends commands to the display using the I^2C Master. The I^2C Master must be initialized and global interrupts must be enabled before calling this function.

Also if you are using the NXP-compatible LCD modulenote that it requires a 1 ms reset pulse prior to calling I2C_LCD_Start(). Refer to the I2C LCD example project to see the proper way of resetting the LCD module.

Parameters: None

Return Value: None

Side Effects: None

void I2C_LCD_Stop(void)

Description: Turns off the display of the LCD screen but does not stop the I^2C Master component.

Parameters: None

Return Value: None

Side Effects: None

void I2C_LCD_Init(void)

Description: Performs initialization of the LCD module as following:

- Turns on the display;
- Enables auto cursor increment;
- Resets the cursor to start position;
- Clears the display;
- It also loads a custom character set to LCD if it was defined in the customizer's GUI. Resets the cursor to start position.

Parameters: None

Return Value: None

Side Effects: None

void I2C_LCD_Enable(void)

Description: Turns on the display.

Parameters: None

Return Value: None

Side Effects: None

void I2C_LCD_DisplayOn(void)

Description: Turns the display on, without initializing it. It calls function I2C_LCD_WriteControl() with the appropriate argument to activate the display.

Parameters: None

Return Value: None

Side Effects: Disables underlining of the cursor if it was previously enabled.

void I2C_LCD_DisplayOff(void)

Description: Turns the display off, but does not reset the LCD module in any way. It calls function I2C_LCD_WriteControl() with the appropriate argument to deactivate the display.

Parameters: None

Return Value: None

Side Effects: None

void I2C_LCD_PrintString(char8 const string[])

Description: Writes a null-terminated string of characters to the screen beginning at the current cursor location.

Parameters: char8 const string[]: Null-terminated array of ASCII characters to be displayed on the LCD module's screen.

Return Value: None

Side Effects: None

Note Because of the character set that is hardcoded to the NXP PCF2119x LCD module, which is used in the PSoC 4 processor module, some of the characters can't be displayed. Refer to Functional Description section for details.

void I2C_LCD_PutChar(char8 character)

Description: Writes an individual character to the screen at the current cursor location. Used to display custom characters through their named values. (I2C_LCD_CUSTOM_0 through I2C_LCD_CUSTOM_7).

Parameters: char8 character: ASCII character to be displayed on the LCD module's screen.

Return Value: None

Side Effects: None

Note Because of the character set that is hardcoded to the NXP PCF2119x LCD module, which is used in the PSoC 4 processor module, some of the characters can't be displayed. Refer to Functional Description section for details.

void I2C_LCD_Position(uint8 row, uint8 column)

Description: Moves the cursor to the location specified by arguments **row** and **column**.

Parameters: uint8 row: The row number at which to position the cursor. Minimum value is zero.

uint8 column: The column number at which to position the cursor. Minimum value is zero.

Return Value: None

Side Effects: None

void I2C_LCD_WriteData(uint8 dByte)

Description: Writes data to the LCD RAM in the current position. Upon write completion, the position is incremented or decremented depending on the entry mode specified.

Parameters: dByte: A byte value to be written to the LCD module.

Return Value: None

Side Effects: None

void I2C_LCD_WriteControl(uint8 cByte)

Description: Writes a command byte to the LCD module. Different LCD models can have their own commands. Review the specific LCD datasheet for commands valid for that model.

Parameters: cByte: 8-bit value representing the command to be loaded into the command register of the LCD module. Valid command parameters are specified in the table below:

Value	Description
I2C_LCD_CLEAR_DISPLAY	Clear display
I2C_LCD_RESET_CURSOR_POSITION I2C_LCD_CURSOR_HOME	Return cursor and LCD to home position
I2C_LCD_CURSOR_LEFT	Set left cursor move direction
I2C_LCD_CURSOR_RIGHT	Set right cursor move direction
I2C_LCD_DISPLAY_CURSOR_ON	Enable display and cursor
I2C_LCD_DISPLAY_ON_CURSOR_OFF	Enable display, cursor off
I2C_LCD_CURSOR_WINK	Enable display, cursor off, set cursor wink
I2C_LCD_CURSOR_BLINK	Enable display and cursor, set cursor blink
I2C_LCD_CURSOR_SH_LEFT	Move cursor/Shift display left
I2C_LCD_CURSOR_SH_RIGHT	Move cursor/shift display right
I2C_LCD_DISPLAY_2_LINES_5x10	Set display to be 2 lines 10 characters
I2C_LCD_CURSOR_RIGHT I2C_LCD_CURSOR_AUTO_INCR_ON	Move cursor/shift display right (cursor position auto increment)
I2C_LCD_CURSOR_LEFT	Move cursor/shift display right (cursor position auto decrement)

Return Value: None

Side Effects: None

void I2C_LCD_ClearDisplay(void)

Description: Clears the contents of the screen and resets the cursor location to be row and column zero. It calls I2C_LCD_WriteControl() with the appropriate argument to activate the display.

Parameters: None

Return Value: None

Side Effects: Cursor position reset to row – 0, column - 0.

void I2C_LCD_SetAddr (uint8 address)

Description: This function allows you to change the default I^2C address of the LCD. This function is not used for designs with a single LCD. Systems that have two or more LCDs on a single I^2C bus use this function to select the appropriate LCD.

Parameters: uint8 address: I^2C address of a LCD device.

Return Value: None

Side Effects: None

void I2C_LCD_PrintInt8(uint8 value)

Description: Prints a two ASCII character representation of the 8-bit value to the character I^2C LCD module.

Parameters: uint8 value: The 8-bit value to be printed in hexadecimal ASCII characters.

Return Value: None

Side Effects: None

void I2C_LCD_PrintInt16(uint16 value)

Description: Prints a four ASCII character representation of the 16-bit value to the Character I2C LCD module.

Parameters: uint16 value: The 16-bit value to be printed in hexadecimal ASCII characters.

Return Value: None

Side Effects: None

void I2C_LCD_PrintNumber(uint16 value)

Description: Prints the decimal value of a 16-bit value as left-justified ASCII characters.

Parameters: uint16 value: The 16-bit value to be printed in ASCII characters as a decimal number.

Return Value: None

Side Effects: None

void I2C_LCD_HandleOneByteCommand(uint8 cmdId, uint8 cmdByte)

Description: This command adds a support of sending custom commands with 1 byte parameter.

Parameters: uint8 cmdId: Command code.
uint8 cmdByte: one byte of data to be send to I2C LCD.

Return Value: None

Side Effects: None

void I2C_LCD_HandleCustomCommand(uint8 cmdId, uint8 dataLength, uint8 const cmdData[])

Description: Performs sending of the command that has variable parameters.

Parameters: uint8 cmdId: Command code.
uint8 dataLength: length of data to be sent for this command.
uint8 const cmdData[]: the data to be send to I^2C LCD.

Return Value: None

Side Effects: None

Optional Custom Font Functions

These optional functions are included, when needed, if you select a custom font. The I2C_LCD_LoadCustomFonts() function comes with every custom font set, whether it is user-defined or PSoC Creator generated. The I2C_LCD_LoadCustomFonts() function can be used to load the user-defined or the bar graph characters into the LCD hardware. If you load custom fonts created by the tool, you need to import a pointer to the custom font to your project before using this function (see the description of the I2C_LCD_LoadCustomFonts()). By default, the I2C_LCD_Init() routine loads the user-selected custom font. The draw bar graph commands are generated when a bar graph is selected and enable the easy, dynamic adjustment of bar graphs.

Optional Custom Font Functions	Description
I2C_LCD_LoadCustomFonts()	Loads custom characters into the LCD module
I2C_LCD_DrawHorizontalBG()	Draws a horizontal bar graph. Only available when a bar graph character set has been selected.
I2C_LCD_DrawVerticalBG()	Draws a vertical bar graph. Only available when a bar graph character set has been selected.

void I2C_LCD_LoadCustomFonts(uint8 const customData[])

Description: Loads eight custom characters (bar graph or user-defined fonts) into the LCD module to use the custom fonts during runtime. Only available if a custom character set was selected in the customizer.

Parameters: uint8 const customData[]: Pointer to the head of an array of bytes. Array should be 64 bytes long as 5x8 characters require 8 bytes per character.

Return Value: None

Side Effects: Overwrites any previous custom characters that may have been stored in the LCD module.

void I2C_LCD_DrawHorizontalBG(uint8 row, uint8 column, uint8 maxCharacters, uint8 value)

Description: Draws a horizontal bar graph. Only available if a horizontal or vertical bar graph was selected.

Parameters: uint8 row: The row of the first character in the bar graph.

uint8 column: The column of the first character in the bar graph.

uint8 maxCharacters: Number of whole characters the bar graph consumes. Represents height or width depending upon the bar graph selection. Each character is 5 pixels wide and 8 pixels high.

uint8 value: Number of shaded pixels to draw. May not exceed total pixel length (height) of the bar graph.

Return Value: None

Side Effects: None

void I2C_LCD_DrawVerticalBG(uint8 row, uint8 column, uint8 maxCharacters, uint8 value)

Description: Draws a vertical bar graph. Only available if a horizontal or vertical bar graph was selected.

Parameters: uint8 row: The row of the first character in the bar graph.

uint8 column: The column of the first character in the bar graph.

uint8 maxCharacters: Number of whole characters the bar graph consumes. Represents height or width depending upon the bar graph selection. Each character is 5 pixels wide and 8 pixels high.

uint8 value: Number of shaded pixels to draw. May not exceed total pixel length (height) of the bar graph.

Return Value: None

Side Effects: None

Global Variables

Function	Description
I2C_LCD_initVar	The I2C_LCD_initVar variable is used to indicate initial configuration of this component. This variable is prepended with the component name. The variable is initialized to zero and set to 1 the first time I2C_LCD_Start() is called. This allows for component initialization without reinitialization in all subsequent calls to the I2C_LCD_Start() routine. It is necessary to reinitialize the component when the device is going through sleep cycles. Therefore, the variable is set to zero when going into sleep I2C_LCD_Sleep() and set during the reinitialization done in I2C_LCD_Wakeup().
I2C_LCD_customFonts[]	Stores custom character set that was defined in the customizer's GUI.

Macros

To simplify the use of a system that uses multiple LCDs on a single bus, a macro for each API is added that includes the I^2C address of the LCD. For example, the address macro for the command “I2C_LCD_Position(row, col)” is “I2C_LCD_AddrPosition(addr, row, col)”. The parameter “addr” is the I2C address of the LCD. Calling the macro function is the same as calling the I2C_LCD_SetAddr() function before calling the native function. For example, calling the macro function:

```
I2C_LCD_AddrPosition(addr, row, col);
```

This function is exactly like calling these two functions, one after the other:

```
I2C_LCD_SetAddr(addr);
I2C_LCD_Position(row, col);
```

As expected, the I^2C LCD address remains until another macro is called or the I2C_LCD_SetAddr() function itself is called. Here is a summary table of the macros that are supported.

Functions	Address Macro
I2C_LCD_Start()	I2C_LCD_AddrStart()
I2C_LCD_Stop()	I2C_LCD_AddrStop()
I2C_LCD_DisplayOn()	I2C_LCD_AddrDisplayOn()
I2C_LCD_DisplayOff()	I2C_LCD_AddrDisplayOff()
I2C_LCD_PrintString()	I2C_LCD_AddrPrintString()
I2C_LCD_PutChar()	I2C_LCD_AddrPutChar()
I2C_LCD_Position()	I2C_LCD_AddrPosition()
I2C_LCD_WriteData()	I2C_LCD_AddrWriteData()

Functions	Address Macro
I2C_LCD_WriteControl()	I2C_LCD_AddrWriteControl()
I2C_LCD_ClearDisplay()	I2C_LCD_AddrClearDisplay()
I2C_LCD_Init()	N/A
I2C_LCD_Enable()	I2C_LCD_AddrEnable()
I2C_LCD_LoadCustomFonts()	I2C_LCD_AddrLoadCustomFonts()
I2C_LCD_DrawHorizontalBG()	I2C_LCD_AddrDrawHorizontalBG()
I2C_LCD_DrawVerticalBG()	I2C_LCD_AddrDrawVerticalBG()
I2C_LCD_SetAddr()	N/A
I2C_LCD_PrintInt8()	I2C_LCD_AddrPrintInt8()
I2C_LCD_PrintInt16()	I2C_LCD_AddrPrintInt16()
I2C_LCD_PrintNumber()	I2C_LCD_AddrPrintNumber()
I2C_LCD_HandleCustomCommand()	I2C_LCD_AddrHandleCustomCommand()

To simplify access to custom commands defined in the component's GUI, these macros are generated. The names shown below correspond to "Func1" entered in the API name of **Custom command** table.

- **`I2C_LCD_Func1()`**–Function-like macro, which is generated for the case when custom a command has the **data size** parameter set to zero.
- **`I2C_LCD_Func1(cmdByte)`** – Function-like macro which is generated for the case when a custom command has **Data size** parameter equal to one. cmdByte is a one byte parameter for the command.
- **`I2C_LCD_Func1(cmdData)`** – Function-like macro which is generated for the case when a custom command has a **Data size** parameter greater than 1. cmdData is an array with the size specified by the **Data size** parameter.
- **`I2C_LCD_FUNC1`** – Defines the command code (the value that is entered in **CMD byte 2**), that can be used as a parameter of I2C_LCD_WriteControl(), I2C_LCD_HandleOneByteCommand() or I2C_LCD_HandleCustomCommand().

Sample Firmware Source Code

PSoC Creator provides numerous example projects that include schematics and example code in the Find Example Project dialog. For component-specific examples, open the dialog from the Component Catalog or an instance of the component in a schematic. For general examples, open the dialog from the Start Page or **File** menu. As needed, use the **Filter Options** in the dialog to narrow the list of projects available to select.

See the "Find Example Project" topic in the PSoC Creator Help for more information.

MISRA Compliance

This section describes the MISRA-C:2004 compliance and deviations for the component. There are two types of deviations defined: project deviations – deviations that are applicable for all PSoC Creator components and specific deviations – deviations that are applicable only for this component. This section provides information on component specific deviations. The project deviations are described in the MISRA Compliance section of the *System Reference Guide* along with information on the MISRA compliance verification environment.

The I^2C LCD component has the following specific deviations:

Rule	Class	Rule Description	Description of Deviation(s)
19.7	Req.	A function shall be used in preference to a function-like macro.	A function-like macro is used in the component to handle differences in the API function names of the I^2C Master implemented on the UDB and I^2C Master implemented on the SCB. Also function-like macros are used to simplify usage of custom commands.

API Memory Usage

The component memory usage varies significantly, depending on the compiler, device, number of APIs used, and component configuration. This table shows the memory use for all APIs available in the given component configuration.

The measurements were done with an associated compiler configured in release mode with optimization set for size. For a specific design, the map file generated by the compiler can be analyzed to determine the memory usage.

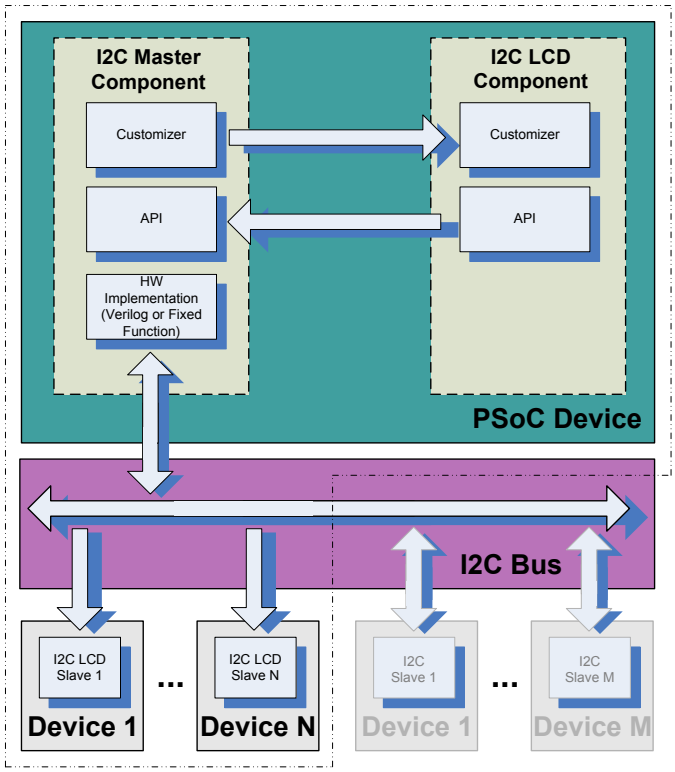
Configuration	PSoC 3 (Keil_PK51)		PSoC 4 (GCC)		PSoC 5LP (GCC)	
	Flash Bytes	SRAM Bytes	Flash Bytes	SRAM Bytes	Flash Bytes	SRAM Bytes
None	1059	259	748	268	772	265
Horizontal or Vertical bargraph	1654	259	1136	268	1148	265
User Defined	1200	259	870	268	894	265

Functional Description

Block Diagram and Configuration

The I²C LCD component is a wrapper around an I²C Master component. The component provides the library for generating transactions defined by the NXP PCF2119x chip. It also has support for the generation of custom I²C LCD modules, which are similar to the NXP PCF2119x command format. With the I²C LCD component it is possible to communicate with several LCDs, of the same type using address macros. Also, you can use several components to add support for LCDs with different command formats. All of the I²C components can share the same I²C Master component.

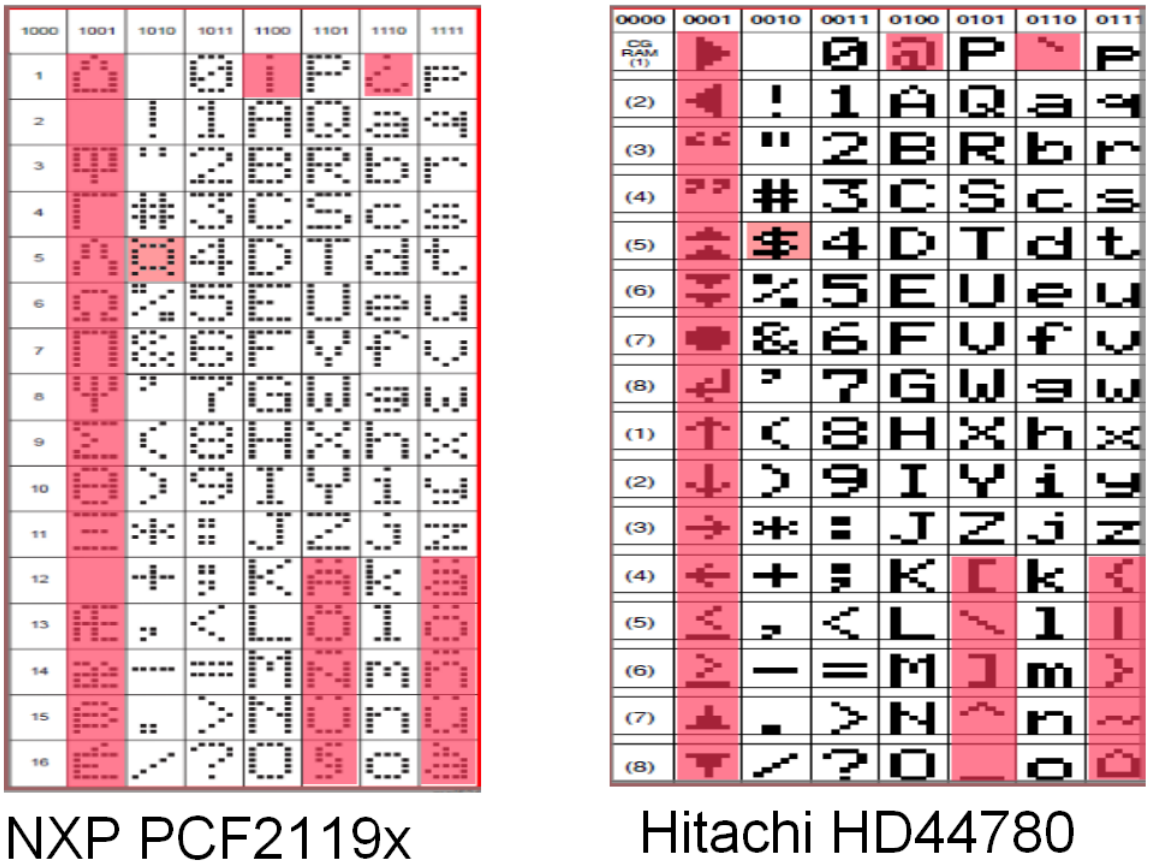
Figure 1. Interaction of I²C LCD and I²C Master Components



I²C LCD and Character LCD

The I²C LCD component was designed to have the same user interface as the existing PSoC Creator Character LCD component. Although the I²C LCD has the same character set as Character LCD, there is a difference in output. Because of character set that is hardcoded to NXP PCF2119x LCD module, which is used in the PSoC4 processor module, some of characters, for example brackets (‘[’,’]’), cannot be displayed on the LCD. The lists of characters for both LCDs are shown in the following figure, and those characters that differ are marked with red color.

Figure 2. Character Sets for NXP PCF2119x and Hitachi HD44780 LCD Modules



Resources

The I²C LCD component is a software component and uses no hardware resources.

Registers

N/A

DC and AC Electrical Characteristics

The following values indicate expected performance and are based on initial characterization data.

Parameter	Description	Conditions	Min	Typ	Max	Units
NumLCD	Number of LCD Modules on the LCD that can be placed on the I^2C Bus	Determined by the capacitance on the I^2C bus and the pull up resistors.	1	-	>10	
t_{fnc_ex} [1]						
	I2C_LCD_Start()	Data rate = 100kbps	-	14.8	-	ms
	I2C_LCD_Stop()		-	1.94	-	ms
	I2C_LCD_DisplayOn()		-	0.44	-	ms
	I2C_LCD_DisplayOff()		-	1.94	-	ms
	I2C_LCD_PrintString()		-	1.56 [2]	-	ms
	I2C_LCD_PutChar()		-	0.43	-	ms
	I2C_LCD_Position()		-	0.43	-	ms
	I2C_LCD_WriteData()		-	0.43	-	ms
	I2C_LCD_WriteControl()		-	0.49	-	ms
	I2C_LCD_ClearDisplay()		-	1.98	-	ms
	I2C_LCD_Init()		-	14.4	-	ms
	I2C_LCD_Enable()		-	0.44	-	ms
	I2C_LCD_SetAddr()		-	2	-	us
	I2C_LCD_HandleCustomCommand()		-	0.62 [3]	-	ms
	I2C_LCD_HandleOneByteCommand()		-	0.62	-	ms
	I2C_LCD_LoadCustomFonts()		-	9.0	-	ms
	I2C_LCD_DrawHorizontalBG()		-	1.94 [4]	-	ms
	I2C_LCD_DrawVerticalBG()		-	3.46 [5]	-	ms

1. The values were received on PSoC3 device with BUS_CLK set to 24 MHz
2. The value is for 10-characters string
3. The value is for command pattern that consists with 2 bytes and data size = 1 byte
4. The value is for printing 10 bars
5. The value is for printing 4 bars

Parameter	Description	Conditions	Min	Typ	Max	Units
	I2C_LCD_PrintInt8()		-	0.55	-	ms
	I2C_LCD_PrintInt16()		-	1.1	-	ms
	I2C_LCD_PrintNumber()		-	1.03	-	ms

Component Changes

This section lists the major changes in the component from the previous version.

Version	Description of Changes	Reason for Changes/Impact
1.20	Added support for Bluetooth Low Energy devices.	
	Fixed address parameter validator to allow 8-bit address.	To remove issue that limited LCD address range.
1.10.a	Add I2C_LCD_SetAddr () function description to Application Programming Interface section.	The function description was missing.
1.10	Fixed a warning about static function - I2C_LCD_SendCmd(), which was declared but not defined, was fixed.	The warning was appearing when the command format option was set to "Custom format".
1.0a	A note of resetting the NXP-compatible LCD module was added to description of I2C_LCD_Start()	Address an issue with the NXP-compatible LCD module.
1.0	Version 1.0 is the first release of the I^2C LCD component	

© Cypress Semiconductor Corporation, 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.

PSoC® is a registered trademark, and PSoC Creator™ and Programmable System-on-Chip™ are trademarks of Cypress Semiconductor Corp. All other trademarks or registered trademarks referenced herein are property of the respective corporations.

Any 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.