Character LCD Datasheet LCD V 1.60

Copyright © 2002-2015 Cypress Semiconductor Corporation. All Rights Reserved.

Resources	PSoC® Blocks			API Memory (Bytes)		Pins (per External I/O)
	Digital	Analog CT	Analog SC	Flash	RAM	
CY8C29/27/26/25/24/22/21xxx, CY8C23x33, CY7C603xx/64215, CYWUSB6953, CY8C20x34, CY8CLED02/04/08/16, CY8C21x45, CY8C22x45, CY8C28x45, CY8CPLC20, CY8CLED16P01, CY8C28xxx, CY8C21x12						
Bar Graph Enabled	0	0	0	646	0	7 from One Port
Bar Graph Disabled	0	0	0	434	0	7 from One Port

For one or more fully configured, functional example projects that use this user module go to www.cypress.com/psocexampleprojects.

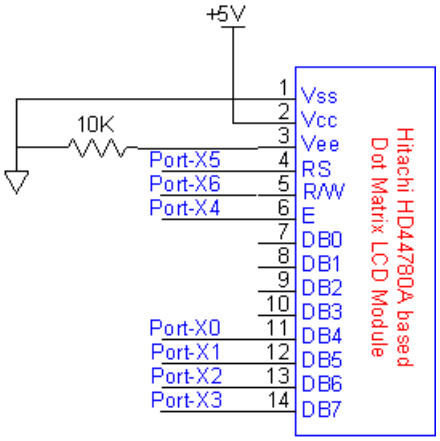
Features and Overview

- Uses the industry standard Hitachi HD44780 LCD display driver chip protocol
- Requires only seven I/O pins
- Routines provided to print RAM or ROM strings
- Routines provided to print numbers
- Routines provided to display horizontal and vertical bar graphs
- Uses a single I/O port

The Character LCD User Module is a set of library routines that writes text strings and formatted numbers to a common two or four-line LCD module. Vertical and horizontal bar graphs are supported, using the character graphics feature of these LCD modules. This module was developed specifically for the industry

standard Hitachi HD44780 two-line by 16 character LCD display driver chip, but works for many other four-line displays. This library uses the 4-bit interface mode to limit the number of I/O pins required.

Figure 1. LCD to PSoC Block Diagram



Note For some displays, it may be good design practice to tie signals DB0-3 on the display to GND with 10K resistors.

Functional Description

The LCD User Module uses a single I/O port to interface to an industry standard Hitachi HD44780A LCD controller. This type of display has a simple interface consisting of 8 data bits, read/write (R/W), register select “RS," and an enable “E" signal. To reduce the number of pins required, the 4-bit interface mode is used. The following LCD to PSoC block diagram and the table describe the 4-bit interface connections.

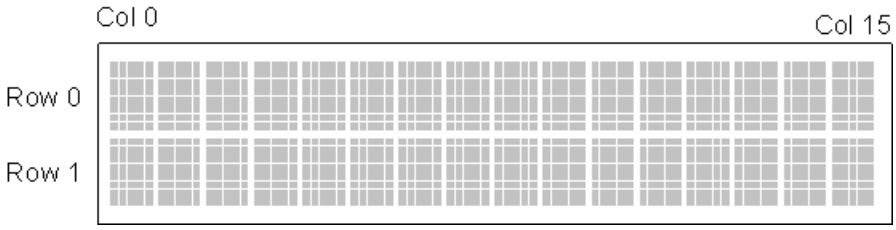
On some displays, DB0, DB1, DB2, and DB3 may need to be pulled down to V_{ss} with a 10K resistor. Pulling these signals low ensures that the 4-bit mode is entered properly.

Table 1. LCD to PSoC Interconnect

PSoC Pin	LCD Pin	Description
Port-X0	DB4	Data Bit 0
Port-X1	DB5	Data Bit 1
Port-X2	DB6	Data Bit 2
Port-X3	DB7	Data Bit 3
Port-X4	E	LCD Enable
Port-X5	RS	Register Select
Port-X6	R/W	Read/ Not Write

A cursor position function places the cursor at any location. For two-line by 16 character displays, the upper left corner is position (0,0) and the lower right corner is position (1,15). Refer the following figure.

Figure 2. Cursor Position



Low level commands are provided to write data to the display Data and Control registers. The LCD manufacturer’s datasheet should be reviewed for specific features and font information.

Placement

The LCD User Module only uses seven I/O pins of one port and does not use any digital or analog blocks. There are no placement restrictions. Multiple LCD modules may be placed in a single project.

Parameters and Resources

LCDPort

Selects which PSoC I/O port is used to interface to the LCD display module.

Note The Shadow Registers should be used to manipulate the Px[7] of a LCD port for other purposes. Refer to the Shadow Registers datasheet for details.

Bargraph

Selects whether the bargraph functions are enabled. If disabled, the bargraph code is not generated, saving ROM space.

Application Programming Interface

The Application Programming Interface (API) routines are provided as part of the user module to allow the designer to deal with the module at a higher level. This section specifies the interface to each function together with related constants provided by the “include" files.

Note

In this, as in all user module APIs, the values of the A and X register may be altered by calling an API function. It is the responsibility of the calling function to preserve the values of A and X before the call if those values are required after the call. This “registers are volatile" policy was selected for efficiency reasons and has been in force since version 1.0 of PSoC Designer. The C compiler automatically takes care of this requirement. Assembly language programmers must ensure their code observes the policy, too. Though some user module API function may leave A and X unchanged, there is no guarantee they may do so in the future.

For Large Memory Model devices, it is also the caller's responsibility to preserve any value in the CUR_PP, IDX_PP, MVR_PP, and MVW_PP registers. Even though some of these registers may not be modified now, there is no guarantee that will remain the case in future releases.

Here are the API programming routines provided for the LCD User Module:

Basic Character LCD Functions

LCD_Start

Description:

Initializes LCD to use the multi-line 4-bit interface. This function should be called before all other LCD functions.

C Prototype:

```
void  LCD_Start(void);
```

Assembly:

```
lcall  LCD_Start
```

Parameters:

None

Return Value:

None

Side Effects:

The A and X registers may be modified by this or future implementations of this function. The same is true for all RAM page pointer registers in the Large Memory Model (CY8C29xxx). When necessary, it is the calling function's responsibility to preserve the values across calls to fastcall16 functions. Currently, only the CUR_PP page pointer register is modified.

LCD_Init

Description:

Initializes LCD to use the multi-line, 4-bit interface. This function is identical to the LCD_Start function and is kept for backward compatibility purposes. For all new projects, the LCD_Start function should be used instead of this function to initialize the LCD User Module.

C Prototype:

```
void  LCD_Init(void);
```

Assembly:

```
lcall  LCD_Init
```

Parameters:

None

Return Value:

None

Side Effects:

The A and X registers may be modified by this or future implementations of this function. The same is true for all RAM page pointer registers in the Large Memory Model (CY8C29xxx). When necessary, it is the calling function's responsibility to preserve the values across calls to fastcall16 functions. Currently, only the CUR_PP page pointer register is modified.

LCD_Position

Description:

Moves cursor to a location specified by the parameters. The upper left character is row 0, column 0. For a two-line by 16 character display, the lower right character is row 1, column 15.

C Prototype:

```
void  LCD_Position(BYTE bRow, BYTE bCol);
```

Assembly:

```
mov   A,01h               ; Load Row
mov   X,02h               ; Load Column
lcall  LCD_Position
```

Parameters:

bRow: The row number at which to position the cursor. Zero specifies the first row.

bCol: The column number at which to position the cursor. Zero specifies the first (left most) column.

Return Value:

None

Side Effects:

The A and X registers may be modified by this or future implementations of this function. The same is true for all RAM page pointer registers in the Large Memory Model (CY8C29xxx). When necessary, it is the calling function's responsibility to preserve the values across calls to fastcall16 functions. Currently, only the CUR_PP page pointer register is modified.

String Printing Functions

LCD_PrString

Description:

Prints a null terminated RAM-based character string to the LCD at the present cursor location.

C Prototype:

```
void  LCD_PrString(CHAR * sRamString);
```

Assembly:

```
mov   A,>sRamString   ; Load MSB part of pointer to RAM-based null
                      ; terminated string.
mov   X,<sRamString   ; Load LSB part of pointer to RAM-based null
                      ; terminated string.
lcall  LCD_PrString    ; Call function to display string at current
                      ; LCD cursor position.
```

Parameters:

sRamString: A pointer to a null-terminated string located in RAM.

Return Value:

None

Side Effects:

The A and X registers may be modified by this or future implementations of this function. The same is true for all RAM page pointer registers in the Large Memory Model (CY8C29xxx). When necessary, it is the calling function's responsibility to preserve the values across calls to fastcall16 functions. Currently, the CUR_PP and IDX_PP page pointer registers are modified.

LCD_PrCString

Description:

Prints a null terminated ROM-based character string to the LCD at the present cursor location.

C Prototype:

```
void  LCD_PrCString(const char * sRomString);
```

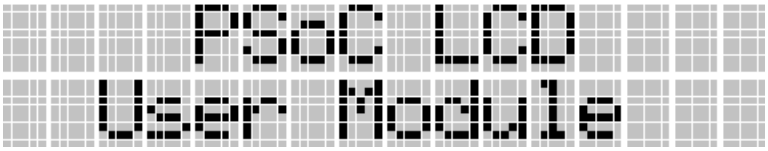
Assembly:

```
mov   A,>sRomString   ; Load MSB part of pointer to ROM-based null
                      ; terminated string.
mov   X,<sRomString   ; Load LSB part of pointer to ROM-based null
                      ; terminated string.
lcall  LCD_PrCString   ; Call function to display string at current
                      ; LCD cursor position.
```

Example String Printing Code:

```
char str[ ] = "User Module";     // Define "RAM" based string
LCD_Start();                     // Initialize LCD hardware
LCD_Position(0,4);              // Position cursor @ row 0, col 4
LCD_PrCString("PsoC LCD");       // Print a constant "ROM" string
LCD_Position(1,2);               // Position cursor @ row 1, col 2
LCD_PrString(str);               // Print "RAM" based string.
```

Figure 3. Example Text Display



Parameters:

sRomString: A pointer to a null-terminated string located in ROM.

Return Value:

None

Side Effects:

The A and X registers may be modified by this or future implementations of this function. The same is true for all RAM page pointer registers in the Large Memory Model (CY8C29xxx). When necessary, it is the calling function's responsibility to preserve the values across calls to fastcall16 functions. Currently, only the CUR_PP page pointer register is modified.

Number Printing Functions

LCD_PrHexByte

Description:

Prints a byte as a two-character hex string at the present LCD cursor position.

C Prototype:

```
void  LCD_PrHexByte(BYTE bValue);
```

Assembly:

```
mov   A, [bValue]          ; Load byte to be printed
lcall  LCD_PrHexByte        ; Call function
```

Parameters:

bValue: An 8-bit value to display as a two-character hex string.

Return Value:

None

Side Effects:

The A and X registers may be modified by this or future implementations of this function. The same is true for all RAM page pointer registers in the Large Memory Model (CY8C29xxx). When necessary, it is the calling function's responsibility to preserve the values across calls to fastcall16 functions. Currently, only the CUR_PP page pointer register is modified.

LCD_PrHexInt

Description:

Prints an integer as a four-character hex string at the present LCD cursor position.

C Prototype:

```
void  LCD_PrHexInt(INT iValue);
```

Assembly:

```
mov   A, [iValue+1]    ; Load LSB byte to be printed
mov   X, [iValue]      ; Load MSB byte to be printed
lcall  LCD_PrHexInt     ; Call function
```

Parameters:

iValue: A 16-bit value to display as a four-character hex string.

Return Value:

None

Side Effects:

The A and X registers may be modified by this or future implementations of this function. The same is true for all RAM page pointer registers in the Large Memory Model (CY8C29xxx). When necessary, it is the calling function's responsibility to preserve the values across calls to fastcall16 functions. Currently, only the CUR_PP page pointer register is modified.

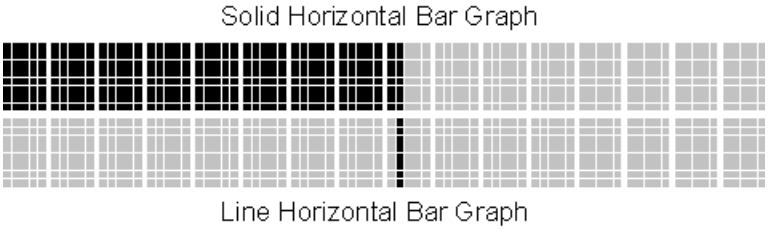
Horizontal Bar Graph Functions

Each display character consists of five horizontal pixels by eight vertical pixels. Horizontal bar graphs display a set of vertical lines that are each composed of one horizontal pixel by eight vertical pixels, within a single character – a pixel column. Each character can display one to five vertical pixel columns, where five pixel columns display the entire character.

Starting on the left side of the display, the first pixel column is numbered 1 and the last pixel column is numbered N * 5, where N is the number of characters. A 16-character display has 80 possible pixel columns numbered 1 to 80.

Solid bar graphs display 1 to N pixel columns, within a set of specified continuous characters. Line bar graphs display only the specified pixel column. Here is an example of both types of horizontal bar graphs:

Figure 4. Bar Graph Types



LCD_InitBG

Description:

Initializes the LCD to display the specified type of horizontal bar graph. This function should be called before calling LCD_DrawBG(). The type of bar graph must be specified. This function does not draw a bar graph, but loads the custom character RAM with the data required to display the specified type of bar graph. This routine must be called to change between horizontal bar-graph types.

LCD_SOLID and LCD_LINE are defined as input constants.

C Prototype:

```
void  LCD_InitBG(BYTE bBGType);
```

Assembly:

```
mov   A, LCD_SOLID_BG
lcall  LCD_InitBG
```

Parameters:

BYTE bBGType: Type of bar graph specified as one of the following:

```
LCD_SOLID_BG
LCD_LINE_BG
```

Return Value:

None

Side Effects:

The A and X registers may be modified by this or future implementations of this function. The same is true for all RAM page pointer registers in the Large Memory Model (CY8C29xxx). When necessary, it is the calling function's responsibility to preserve the values across calls to fastcall16 functions. Currently, only the CUR_PP page pointer register is modified.

LCD_DrawBG

Description:

Draws the horizontal bar graph starting at character location (bRow,bCol) with a character length of “bLen" to column position of “bPixelColEnd".

C Prototype:

```
void LCD_DrawBG(BYTE bRow, BYTE bCol, BYTE bLen, BYTE bPixelColEnd);
```

Assembly:

Note When using the large memory model, calls to LCD_DrawBG should be made using an underscore in front of the function name, call _LCD_DrawBG.

Small Memory Model Assembly:

```
mov   A,19h           ; Set bPixelColEnd = 25
push  A
mov   A,06h           ; Set bLen = 6 pixel columns
push  A
mov   A,03h           ; Set bCol = 3
push  A
mov   X,SP            ; Setup data pointer (X)
dec   X
mov   A,01h           ; Set bRow = 1 -> the second line
lcall LCD_DrawBG
add   SP,-3           ; Restore the stack
```

Large Memory Model Assembly:

```
mov   A,19h           ; Set bPixelRowEnd = 25
push  A
mov   A,06h           ; Set bLen = 6 pixel columns
push  A
mov   A,03h           ; Set bCol = 3
push  A
mov   A,01h           ; Set bRow = 1
push  A
lcall _LCD_DrawBG
add   SP,-4          ; Restore the stack
```

Parameters:

bRow: Defines the starting character row – range of 0 to number of rows minus1.

bCol: Defines the starting character column – range of 0 to number of character columns minus 1.

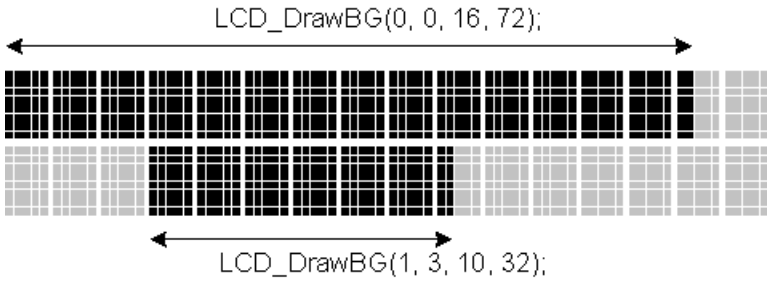
bLen: Defines the length of the bargraph in whole characters.

bPixelColEnd: Defines at which pixel column to draw the following.

Note Solid bar graphs draw all the pixel columns from the first pixel column of the character defined by bRow and bCol to the pixel column specified by bPixelColEnd. Line bar graphs draw the specific pixel column specified in the define character.

For line bargraphs bLen=1 and bPixelColEnd is in the range of 1 to 5.

Figure 5. Examples of Horizontal Bar Graphs



Return Value:

None

Side Effects:

The A and X registers may be modified by this or future implementations of this function. The same is true for all RAM page pointer registers in the Large Memory Model (CY8C29xxx). When necessary, it is the calling function's responsibility to preserve the values across calls to fastcall16 functions. Currently, only the CUR_PP page pointer register is modified.

Vertical Bar Graph Functions

Each display character consists of five horizontal pixels by eight vertical pixels. Vertical bar graphs display a set of horizontal lines that are each composed of one vertical pixel by five horizontal pixels, within a single character – a pixel row. Each character can display one to eight horizontal pixel rows, where eight pixel rows display the entire character.

Starting on the bottom of a character, the first pixel row is numbered 1 and the last pixel row is numbered 8. Combining two rows can generate a vertical bar graph of 16 pixel rows.

LCD_InitVBG

Description:

Initializes the LCD to display vertical bar graphs. This should be called before calling LCD_DrawVGB(). This function initializes the custom character RAM with the data required to draw vertical bar graphs.

C Prototype:

```
void  LCD_InitVBG(void);
```

Assembly:

```
lcall  LCD_InitVBG
```

Parameters:

None

Return Value:

None

Side Effects:

The A and X registers may be modified by this or future implementations of this function. The same is true for all RAM page pointer registers in the Large Memory Model (CY8C29xxx). When necessary,

it is the calling function's responsibility to preserve the values across calls to fastcall16 functions. Currently, only the CUR_PP page pointer register is modified.

LCD_DrawVBG

Description:

Draws a vertical bar graph starting from the first pixel row at character location (bRow, bCol), with a character height of bHeight, up to the specified vertical pixel row bPixelRowEnd.

C Prototype:

```
void  LCD_DrawVBG(BYTE bRow, BYTE bCol, BYTE bHeight, BYTE bPixelRowEnd);
```

Assembly:

Note When using the large memory model, calls to LCD_DrawVBG should be made using an underscore in front of the function name, call _LCD_DrawVBG.

Small Memory Model Assembly:

```
mov   A,0Ah           ; Set bPixelRowEnd = 10
push  A
mov   A,02h           ; Set bHeight = 2 columns
push  A
mov   A,03h           ; Set bCol = 3
push  A
mov   X,SP            ; Setup data pointer (X)
dec   X
mov   A,01h           ; Set bRow = 1 -> the second line
lcall LCD_DrawVBG
add   SP,-3           ; Restore the stack
```

Large Memory Model Assembly:

```
mov   A,0Ah           ; Set bPixelRowEnd = 10
push  A
mov   A,02h           ; Set bHeight = 2 columns
push  A
mov   A,03h           ; Set bCol = 3
push  A
mov   A,01h           ; Set bRow = 1
push  A
lcall _LCD_DrawVBG
add   SP, -4          ; Restore the stack
```

Parameters:

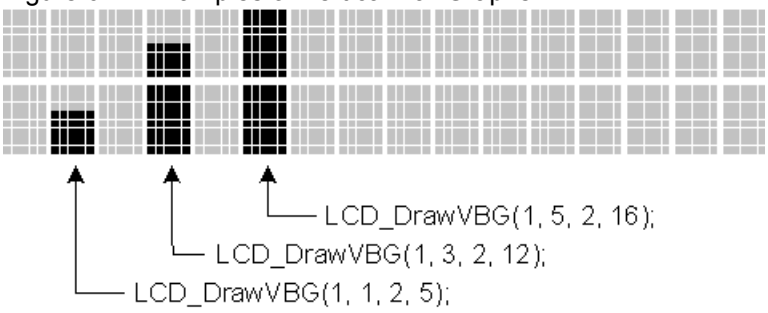
bRow: Defines the starting character row – range of 0 to number of rows minus 1.

bCol: Defines the starting character column – range 0 to number of character columns minus 1.

bHeight: Defines the height of the vertical bargraph in whole characters.

bPixelRowEnd: Defines at which vertical pixel row to draw to.

Figure 6. Examples of Vertical Bar Graphs



Return Value:

None

Side Effects:

The A and X registers may be modified by this or future implementations of this function. The same is true for all RAM page pointer registers in the Large Memory Model (CY8C29xxx). When necessary, it is the calling function's responsibility to preserve the values across calls to fastcall16 functions. Currently, only the CUR_PP page pointer register is modified.

Optional Functions

LCD_Control

Description:

Writes a byte to the LCD Control register. Review the specific LCD datasheet for specific LCD valid commands.

C Prototype:

```
void  LCD_Control(BYTE bCmd);
```

Assembly:

```
mov   A,03h          ; Load data to be written to Control register.
lcall  LCD_Control    ; Call function
```

Parameters:

bCmd: Byte value to send to the Control register.

Return Value:

None

Side Effects:

The A and X registers may be modified by this or future implementations of this function. The same is true for all RAM page pointer registers in the Large Memory Model (CY8C29xxx). When necessary, it is the calling function's responsibility to preserve the values across calls to fastcall16 functions. Currently, only the CUR_PP page pointer register is modified.

LCD_WriteData

Description:

Writes a byte or character to the LCD Data register.

C Prototype:

```
void  LCD_WriteData(BYTE bData);
```

Assembly:

```
mov   A,03h            ; Load data to be written to Data register
lcall  LCD_WriteData    ; Call function
```

Parameters:

bData: Byte value to send to the Data register.

Return Value:

None

Side Effects:

The A and X registers may be modified by this or future implementations of this function. The same is true for all RAM page pointer registers in the Large Memory Model (CY8C29xxx). When necessary, it is the calling function's responsibility to preserve the values across calls to fastcall16 functions. Currently, only the CUR_PP page pointer register is modified.

LCD_Delay50uTimes

Description:

Delays for “bTimes" multiples of 50 µs. This delay loop is CPU clock independent.

C Prototype:

```
void  LCD_Delay50uTimes(BYTE bTimes);
```

Assembly:

```
mov   A,03h             ; Load delay time (example 3 would be 150uSec).
lcall  LCD_Delay50uTimes ; Call function
```

Parameters:

bTimes: Number of times to delay 50 µSec.

Return Value:

None

Side Effects:

The A and X registers may be modified by this or future implementations of this function. The same is true for all RAM page pointer registers in the Large Memory Model (CY8C29xxx). When necessary, it is the calling function's responsibility to preserve the values across calls to fastcall16 functions.

LCD_Delay50u

Description:

Delays for 50 µs. This function is clock independent.

C Prototype:

```
void  LCD_Delay50u(void);
```

Assembly:

```
lcall  LCD_Delay50u      ; Call function
```

Parameters:

None

Return Value:

None

Side Effects:

The A and X registers may be modified by this or future implementations of this function. The same is true for all RAM page pointer registers in the Large Memory Model (CY8C29xxx). When necessary, it is the calling function's responsibility to preserve the values across calls to fastcall16 functions.

Sample Firmware Source Code

Here is a simple assembly and C example for printing a string on the LCD:

```
;;-------------------------------------------------------------------
;; Sample asm LCD Code
;;
;;  Print the string "PSoC LCD" on the top row starting at the 6th
;;  location on an LCD.
;;-------------------------------------------------------------------

include "m8c.inc"       ; part specific constants and macros
include "PSoCAPI.inc"   ; PSoC API definitions for all User Modules
export _main

area text (ROM, REL)

_main:
   call   LCD_Start       ; Initialize LCD
   mov    A,00h           ; Set cursor position at row = 0
   mov    X,05h           ; col = 5
   call   LCD_Position
   mov    A,>THE_STR      ; Load pointer to ROM string
   mov    X,<THE_STR
   call   LCD_PrCString   ; Print constant "ROM" string

loop:
   jmp    loop

.LITERAL
THE_STR:
DS "PSoC LCD"
DB  00h                   ; String should always be null terminated
.ENDLITERAL
```

Here is a sample project written in C:

```
//-------------------------------------------------------------------
// Sample C code for LCD
```

```
//
//  Print the string "PSoC LCD" on the top row starting at the 6th
//  location on an LCD.
//
//-------------------------------------------------------------------
#include <m8c.h>        // part specific constants and macros
#include "PSoCAPI.h"    // PSoC API definitions for all User Modules
void main(void)
{
   char theStr[] = "PSoC LCD";   // Define RAM string
   LCD_Start();                  // Initialize LCD
   LCD_Position(0,5);            // Place LCD cursor at row 0, col 5.
   LCD_PrString(theStr);         // Print "PSoC LCD" on the LCD
}
```

Version History

Version	Originator	Description
1.5	DHA	Added Version History
1.60	DHA	Corrected mechanism for creating list of available ports.
1.60.b	DHA	The following updates were done to this datasheet: 1. Updated sample code for LCD_DrawBG and LCD_DrawVBG. 2. Added more information about Shadow Registers for LCD port manipulation.

Note PSoC Designer 5.1 introduces a Version History in all user module datasheets. This section documents high level descriptions of the differences between the current and previous user module versions.

Copyright © 2002-2015 Cypress Semiconductor Corporation. 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 Designer™ and Programmable System-on-Chip™ are trademarks and PSoC® is a registered trademark 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.