Segment LCD Driver Datasheet SLCD V 2.10

Resources	PSoC® Blocks			API Memory		Pins (per External I/O)
	Digital	Analog CT	Analog SC	Flash[1]	RAM1	
CY8C24x94, CY8C28xxx, CYRF89435, CY8C24x93, CY7C69xxx						
LCD Drive using Analog MUX bus with External 1/2 Bias generator and Timer 16 HW	2	0	0	747[2] 758[3] 753[4]	20[2] 21[3] 22[4]	x
LCD Drive using Analog MUX bus with External 1/2 Bias generator and Timer 16 HW/SW	1	0	0	773[2] 784[3] 779[4]	22[2] 23[3] 24[4]	x
LCD Drive using Analog MUX bus with Internal $V_{DD}/2$ Bias generator and Timer 16 HW	2	1	0	753[2] 764[3] 759[4]	20[2] 21[3] 22[4]	x
LCD Drive using Analog MUX bus with Internal $V_{DD}/2$ Bias generator and Timer 16 HW/SW	1	1	0	779[2] 790[3] 785[4]	22[2] 23[3] 24[4]	x
CY8C20x34, CY8C20xx6, CY8C20xx7/7S, CY8C21x34, CY8C21345, CY8C22x45, CY8C20065						
LCD Drive using Analog MUX bus with External 1/2 Bias generator and Timer 16 HW	2	0	0	747[2] 758[3] 753[4]	20[2] 21[3] 22[4]	x
LCD Drive using Analog MUX bus with External 1/2 Bias generator and Timer 16 HW/SW	1	0	0	773[2] 784[3] 779[4]	22[2] 23[3] 24[4]	x
CY8C20x34, CY8C20xx6, CY8C20336AN, CY8C20436AN, CY8C20636AN, CY8C20xx6AS, CY8C20XX6L, CY8C20xx7/7S, CY8C21x23, CY8C21x34, CY8C21345, CY8C22x45, CY8C23x33, CY8C24x23, CY8C24x94, CY8C27x43, CY8C28xxx, CY8C29x66						
LCD Drive using standalone GPIOs and Timer 16 HW	2	0	0	692[2] 703[3] 698[4]	19[2] 20[3] 21[4]	x
LCD Drive using standalone GPIOs and Timer 16 HW/SW	1	0	0	718[2] 729[3] 724[4]	21[2] 22[3] 23[4]	x

Notes

1. Flash and RAM size depend on LCD configuration, Number of Commons, Ports, Digits, and Displays. In the previous table, Flash and RAM are given for LCD without any Displays where only one segment line in used.

The following formulas evaluate the total Flash and RAM size:

RAM = Base RAM + 1 + 5*(Ports – 1)

Group Size = 0:

Flash = Base ROM + Number of Segments * Digits + GA*(Ports – 1) + 2*(Displays - 1) + DT

Group Size > 0:

Flash = Base ROM + Digits + GA*(Ports – 1) + 2*(Displays - 1) + DT

where:

Base ROM and Base RAM - Flash and RAM sizes in the API Memory column of previous table (for LCD without any Displays where only one segment line in used);

Number of Segments can be 7, 14 or 16;

Digits - Number of all Digits;

GA = 62 for LCD Drive using Analog MUX bus and GA = 55 for LCD Drive using standalone general-purpose I/O (GPIO);

Ports - Number of ports for segments.

Displays - Number of the Displays.

DT is defined in this table:

Display Type	7 segments	14 segments	16 segments
Group Size = 0	270	520	515
Group Size > 0	270	526	528

2. Number of Commons = 2.

3. Number of Commons = 3.

4. Number of Commons = 4.

Features and Overview

- Multiplexed -1/2 bias supported
- 2, 3, and 4 common LCD drive
- 30-150 Hz refresh rate
- LCD Drive technique using analog MUX bus
- Option of 1/2 bias to be generated externally or internally
- Supports Type A waveform only
- Contrast Control

The Segment LCD Driver (SLCD) User Module drives the LCD glass directly without using any external components.

The module demonstrates two techniques of LCD drive that can be selected from the wizard:

1. Using Analog MUX bus (intended for only devices equipped with analog MUX bus)

2. Using standalone GPIOs (applicable to all PSoC devices)

The LCD Drive technique that uses analog MUX bus supports Internal and External Bias. Internal Bias is supported in the CY8C28xxx and CY8C24x94 parts only.

Figure 1. SLCD User Module Block Diagram with Internal Vdd/2 Bias Generator

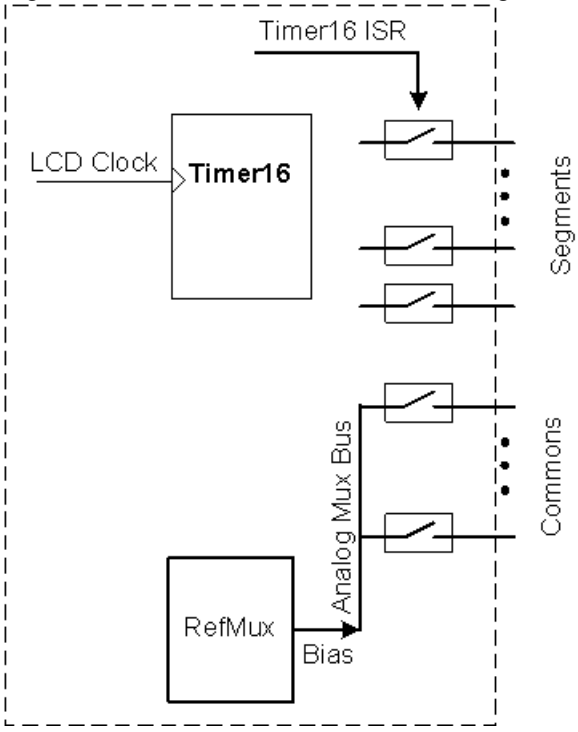
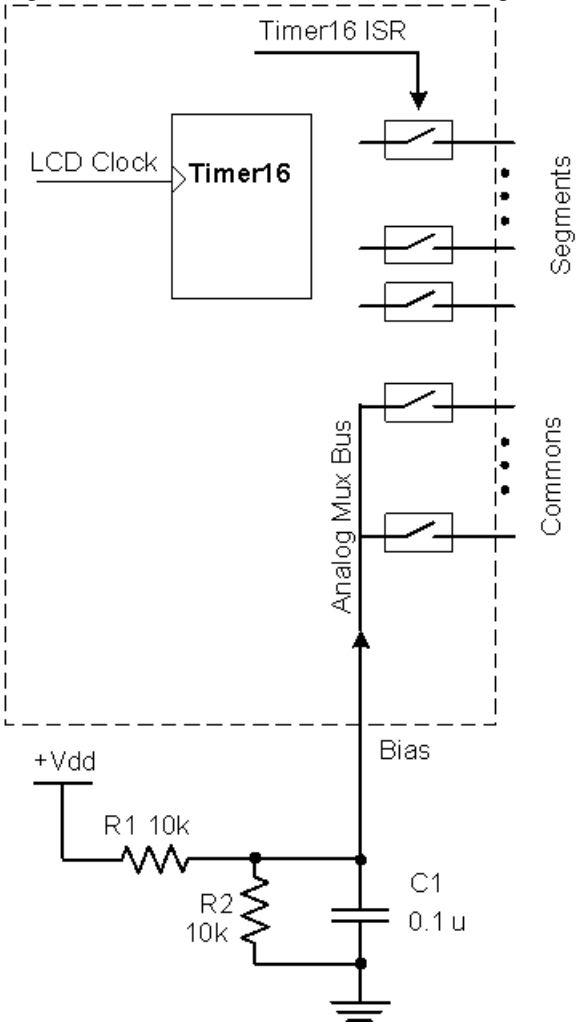


Figure 2. SLCD User Module Block Diagram with External 1/2 Bias Generator



Quick Start

1. Select and place the SLCD User Module.
2. Set the SLCD User Module parameters: LCD Clock, Refresh Rate, and Contrast Level.
3. Right-click on the SLCD User Module and select **SLCD Wizard**.
4. Set the **No. of Commons** to the appropriate number of commons for your display.
5. Enter the **Number of Segment Lines** of the LCD.
6. Click the **+** button below **Add/Remove Display** to add a numeric or alphanumeric LCD display.
7. Click the **+** button below **Increment/Decrement Digits** to add the additional digits to the display.
8. Drag and drop the segments from digits of the Display to the **Segment-Common Mapping Table**. The mapping of all segments should correspond to the LCD datasheet.
9. To assign the segments to PSoC pins, drag and drop the segments from the **Segment-Common Mapping Table** to the **Pin Assignment** area.
10. To assign the commons to PSoC pins, drag and drop the commons from the **Segment-Common Mapping Table** to the **Pin Assignment** area.
11. If you use an external bias generator, drag and drop the **Bias** to the corresponding pin in the Chip view.
12. Press **OK** and then generate the application.
13. Switch to the Application Editor and adapt the sample code as required.
14. Program the PSoC device on the target board.

Functional Description

Segment

A segment is a single bar in a numeric or alphanumeric display.

Segment Control Line

In multiplexed LCD, multiple segments are connected together and a single terminal is taken out which is brought to the pin. This pin is called the Segment Control Line.

Symbols

It covers structures such as decimal points, colons, battery charge indicator symbol, wireless, alarm, and more.

The SLCD User Module can directly drive 2.8V – 5V multiplexed LCD with 1/2 Bias. This user module accompanies a GUI based Wizard to help configure the module according to LCD type.

This user module supports the following types of display structures within the glass:

- 7 segment numerals
- 14 segment alphanumeric
- 16 segment alphanumeric
- Special symbols

There are two techniques of driving LCD that is supported by this user module:

1. LCD drive using analog MUX bus.

In this technique, the analog MUX bus is used to distribute 1/2 Vdd Bias to common lines. This technique is applicable for devices equipped with analog MUX bus.

2. LCD drive using standalone GPIO.

This scheme does not require the device to be equipped with Analog MUX bus. It just uses General Purpose Input and Output (GPIO) to drive the LCD. This scheme also implements contrast control.

The SLCD User Module supports two options for all devices:

1. Timer 16 HW

2. Timer 16 HW/SW

Timer 16 HW consumes two digital blocks. Timer 16 HW/SW consumes one digital block to implement 8 bit HW timer. Remaining 8 bits are implemented by firmware. The Timer ISR controls the timing of refreshing of LCD and the LCD waveforms on the pins.

The SLCD User Module supports functioning when the device is in sleep mode. This is possible only when you select CPU_32K (ILO or ECO) as the clock for LCD Timer. The device wakes up at regular intervals and updates the LCD. If you are setting the device to sleep and if LCD operation is not required, then you can choose any of the available clocks and call the SLCD User Module Stop() API before setting the device to sleep.

Contrast control is achieved by introducing Dead Time between every LCD Refresh Cycle while keeping the same Refresh Rate: that is, as dead time increases, active common signal duration decreases and vice versa. During this Dead Time, common lines and segment lines are held at Vdd. This causes the segment which was ON to be momentarily turned OFF during the dead time, resulting in contrast adjustment. As the dead time is increased, contrast is decreased and vice versa.

The LCD waveforms that describe the Contrast control are shown in Figure 4. When Contrast = 100 %, the LCD waveforms are similar to the waveform shown in Figure 3. The LCD Timer ISR Period is calculated using Equation 1:

Equation 1

$$TimerPeriod = \frac{LCDClock}{2 \times n \times Fr}$$

In Equation 1:

LCD_CLOCK - the LCD timer Clock measured in Hz

n - Number of commons

Fr - LCD Refresh Rate measured in Hz.

Dead time is introduced any time the contrast is less than 100 percent. The dead time duration is inversely proportional to the contrast level.

Dead Time for Analog MUX bus LCD Drive technique is calculated by:

Equation 2

$$Pd = \left(\frac{1}{Fr}\right) - 2 \times n \times (Pc \pm \Delta)$$

Dead Time for Standalone GPIOs LCD Drive technique:

Equation 3

$$Pd = \left(\frac{1}{Fr} \right) - 4 \times n \times (Pc \pm \Delta)$$

In Equation 3, ∆is the increment/decrement period count provided by you;

(Pc ± ∆) - New LCD Timer period for active common signals when contrast < 100 %;

Pd - LCD Timer period for dead phase when contrast < 100%.

In general, the LCD Refresh Period consists of active common signals period and Dead Time period:

Equation 4

$$\text{LCD Refresh Period} = M \times n \times (Pc \pm \Delta) + Pd$$

In Equation 4, M=2 for Analog MUX Bus Drive Technique;M=4 for Standalone GPIO Drive Technique.

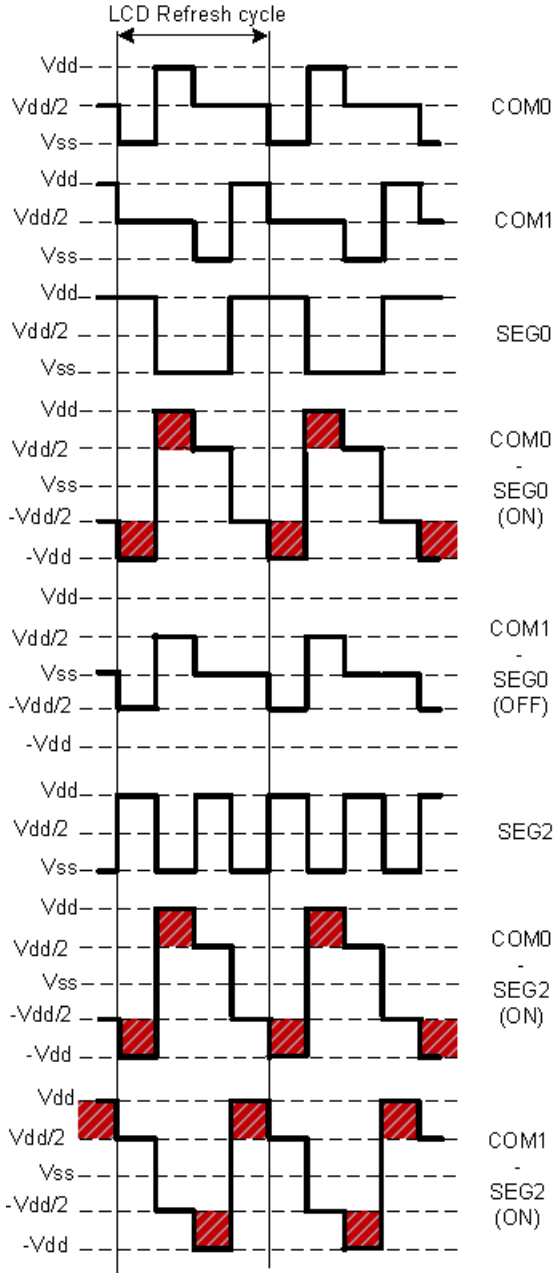
The following application notes are recommended after reading the SLCD User Module datasheet. Application notes are available on the Cypress Semiconductor web site at www.cypress.com:

- *LCD Driving Methods using PSoC* – AN2228
- *Segment LCD Glass Drive Using PSoC with CapSense* – AN56384

Timing

Figure 3 shows the timing for the SLCD User Module when the analog MUX bus is used to form 1/2 bias. Figure 4 illustrates the Contrast control Scheme that is used in the LCD drive using analog MUX bus technique.

Figure 3. LCD Type A Waveforms for 1/2 Bias



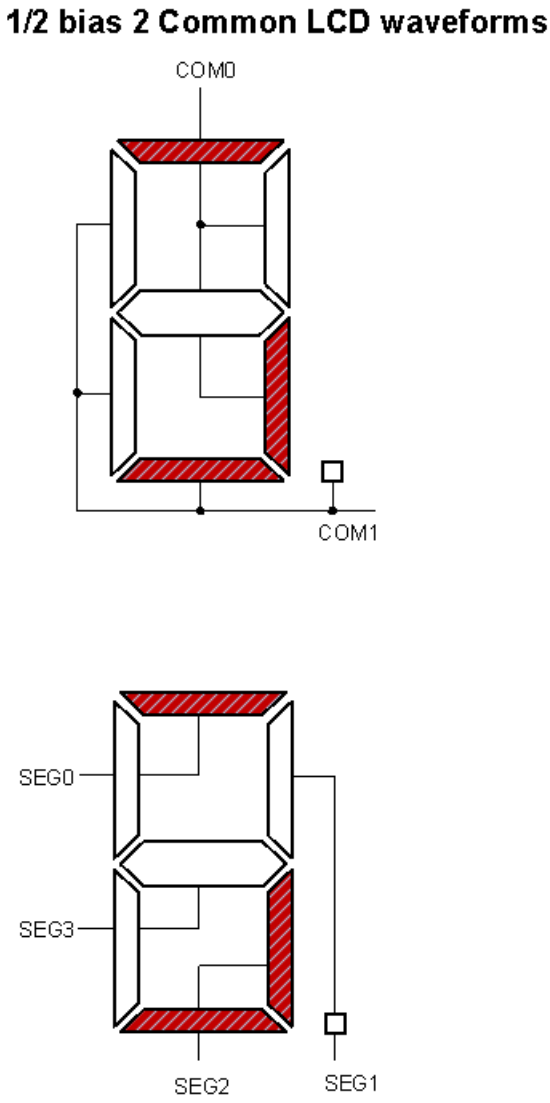


Figure 4. LCD Waveforms for 1/2 Bias and Contrast = 25%

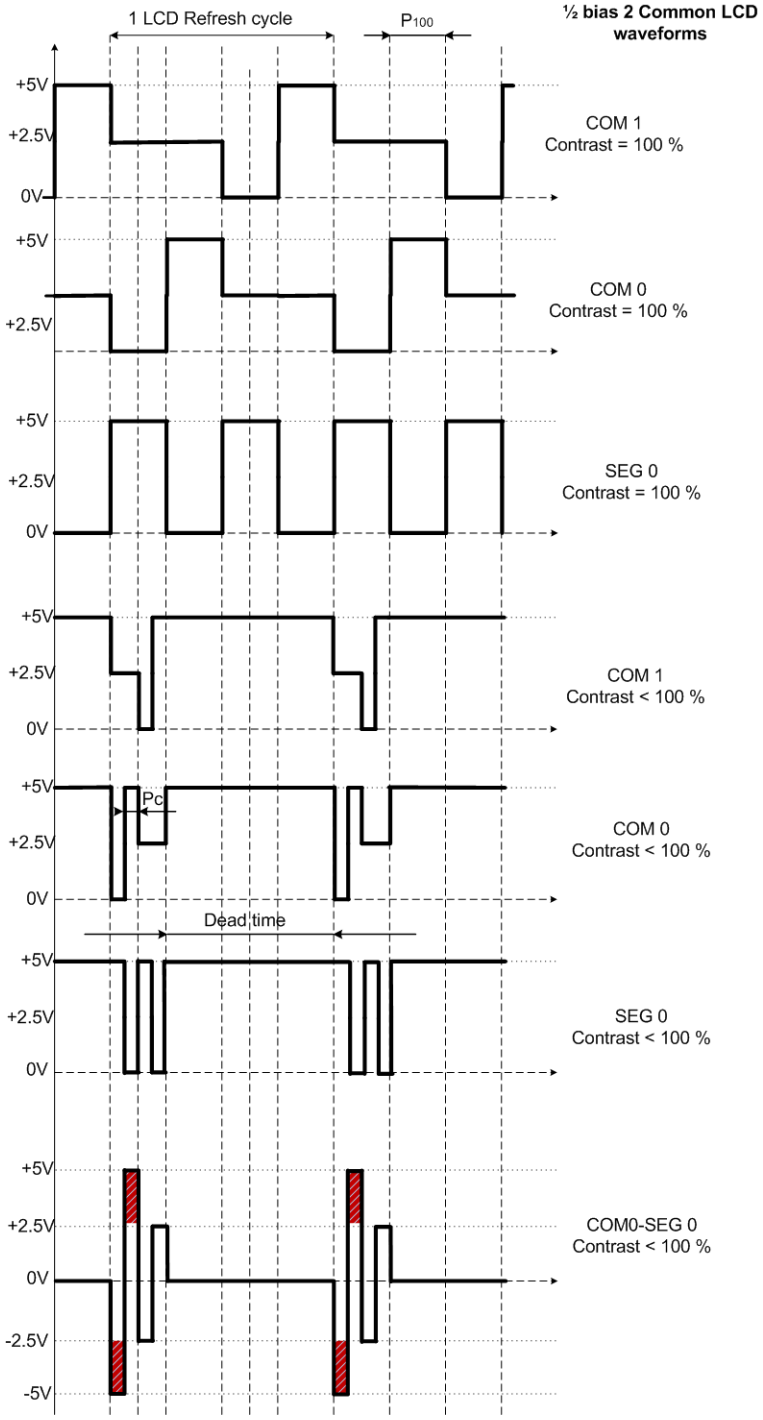
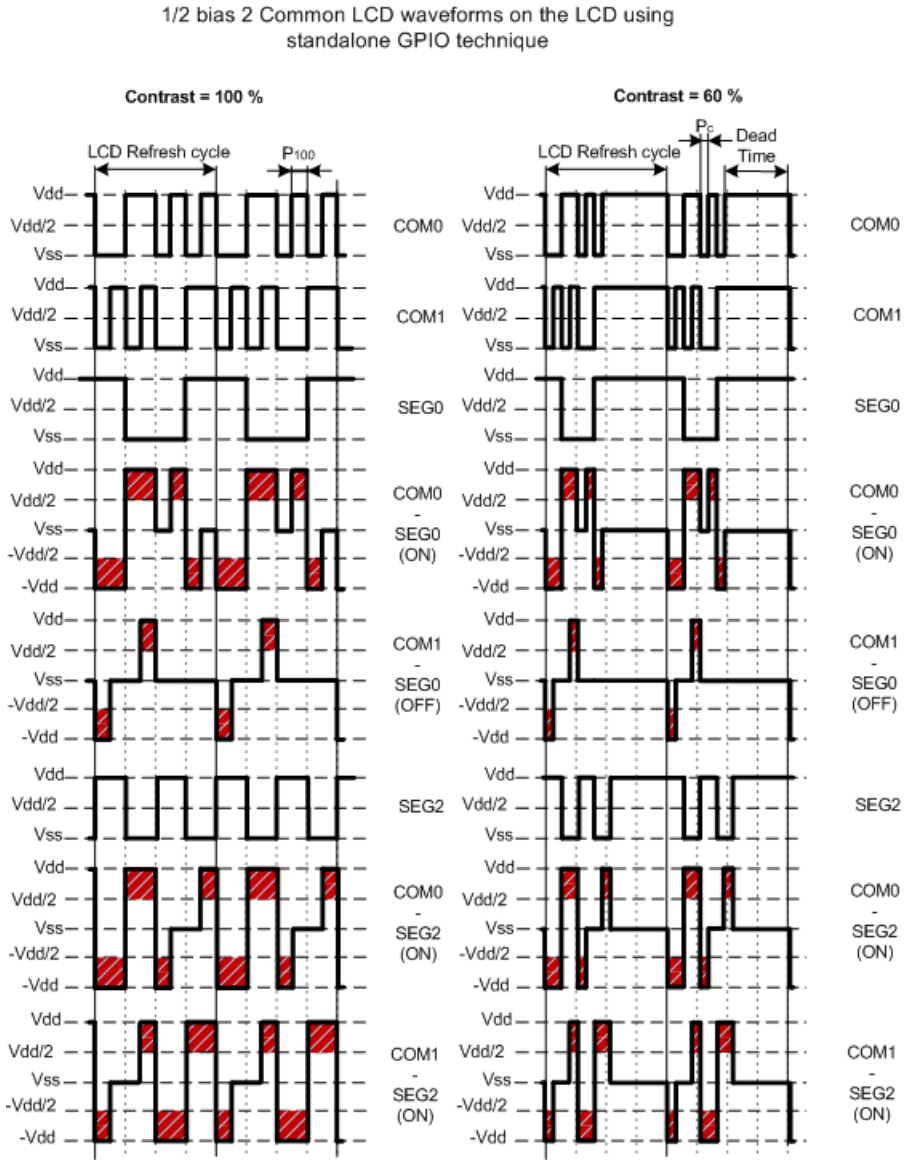


Figure 5. Timing for SLCD User Module when using Standalone GPIO Technique



DC and AC Electrical Characteristics

Table 1. Power Supply Voltage

Parameter	Min	Typical	Max	Unit	Test Conditions and Comments
[1]Value V_{DD}	2.7	5.0	5.25	V	-

[1]Power Supply Voltage is determined by LCD type.

Table 2. Timing and DC Characteristics

Parameter	Typical	Limit	Unit	Test Conditions and Comments
Maximum LCD Timer Input Frequency	-	2000	kHz	VDD = 5.0V
Minimum LCD Timer Input Frequency	-	12	kHz	VDD = 5.0V
Minimum LCD Timer Period	-	0.25	msec	VDD = 5.0V
[2]LCD (Segment control line - Common) Offset Error	-	40	mV	LCD Drive using Analog MUX bus, VDD = 5.0V, CPU_Clock = SysClk/2 (12 MHz), Refresh Rate = 150 Hz, Number of Common Lines = 4
	-	25	mV	LCD Drive using Analog MUX bus, VDD = 3.3V, CPU_Clock = SysClk/2 (12 MHz), Refresh Rate = 150 Hz, Number of Common Lines = 4
	-	11	mV	LCD Drive using standalone GPIOs, VDD = 5.0V, CPU_Clock = SysClk/2 (12 MHz), Refresh Rate = 150 Hz, Number of Common Lines = 4
	-	8	mV	LCD Drive using standalone GPIOs, VDD = 3.3V, CPU_Clock = SysClk/2 (12 MHz), Refresh Rate = 150 Hz, Number of Common Lines = 4
	-	30	mV	LCD Drive using standalone GPIOs, VDD = 5.0V, CPU_Clock = SysClk/8 (3 MHz), Refresh Rate = 150 Hz, Number of Common Lines = 4
	-	20	mV	LCD Drive using standalone GPIOs, VDD = 3.3V, CPU_Clock = SysClk/8 (3 MHz), Refresh Rate = 150 Hz, Number of Common Lines = 4

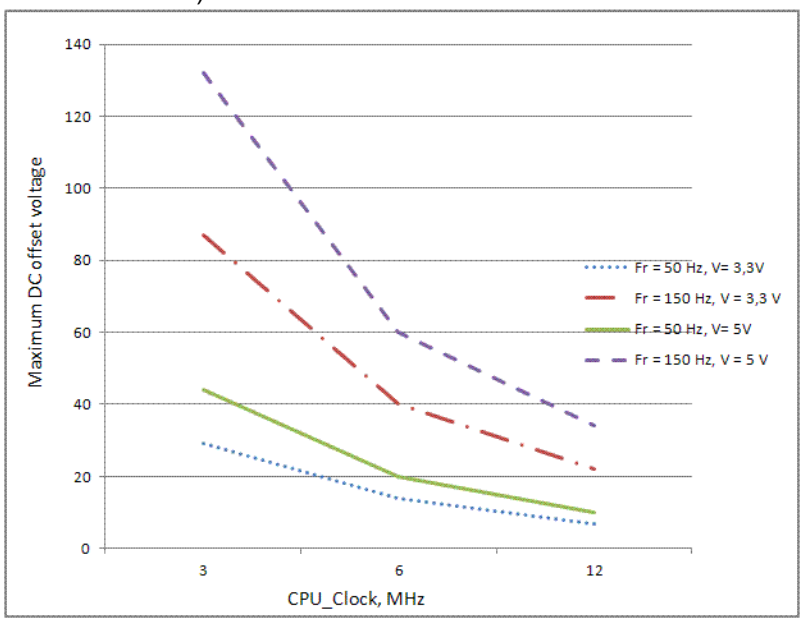
[2]The LCD waveforms are formed by the SLCD User Module interrupt. The SLCD User Module interrupt latency can cause asymmetry in common and segment signals wave shape, causing voltage offset. The large value of DC offset voltage can deteriorate the liquid crystal fluid so it cannot be energized. As a result, DC voltage applied to LCD electrodes decreases the LCD lifetime.

An application that already has ISRs carrying user code or any other module code can delay the LCD update to cause additional DC offset voltage. Also, low CPU_Clock may cause the DC offset voltage to exceed the datasheet value.

The Offset Voltage can be measured as the DC component of a voltage between segment and common pins on full refresh period.

To decrease an interrupt latency influence to DC offset voltage, the CPU_Clock global parameter in the PSoC Designer Device Editor should be increased. Also, DC offset voltage is lower for low LCD refresh rate.

Figure 6. DC Offset Voltage Dependence on CPU_Clock for various Refresh Rates (LCD Drive using Analog MUX bus)



The CPU_Clock global parameter is a critical parameter for the SLCD User Module and its configuration has a large impact on the performance of the user module. Configuring the user module with a CPU_Clock value that is too low may cause a large DC offset, which can damage the LCD over a period of time. It is recommended to use a higher CPU_Clock value when possible. Selecting a CPU_Clock of 12 MHz provides the greatest balance between power and performance in most SLCD User Module configurations.

Placement

Only one SLCD User Module instance is available in a project.

The SLCD User Module allows the following configurations:

- **Segment LCD driver using Analog MUX Bus and External 1/2 Bias generator:**

  Connect an external resistance based potential divider to generate Vdd/2 Bias as shown in Figure 2.

  This Bias must be routed to the PSoC pin, which is internally connected to the analog MUX bus used for the LCD. The pin options are given in the SLCD User Module Wizard (see the Wizard section).

  All supported devices have this option.

- **Segment LCD driver using Analog MUX Bus and Internal Vdd/2 Bias generator:**

  A RefMux User Module is placed to generate a Vdd/2 Bias, which is passed to the analog MUX bus used by the LCD.

  All devices use two digital blocks with this configuration. In addition, an Analog CT block is consumed by CY8C28xxx and CY8C24x94 devices, if you select the Internal Bias generation.

- **Segment LCD driver using standalone GPIOs:**

  The Analog MUX Bus is not used in this configuration.

Table 3. Block Resources

Block	Module	Name	Notes
Digital Block	Timer16/ Timer 8	LCDTimer	This is used by all devices and for both drive techniques. Here you can either select 16 bit HW Timer or 16 bit HW/SW timer. The 16 bit HW timer uses 2 digital resources and the 16 bit HW/SW only uses 1 digital resource. The clock for this timer can be selected in the user module properties.
Analog CT block	RefMux	LCDBias	Only in case of CY8C28xxx and CY8C24x94, if Internal Bias is selected for LCD drive using Analog MUX bus.

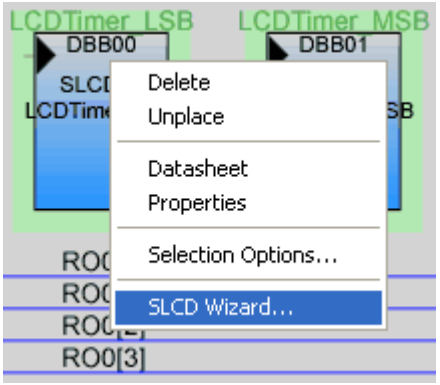
Note There some restrictions that are applicable to an LCD drive using the Analog MUX bus when a CapSense™ module is already placed in the design:

- The CPU_Clock global parameter is a critical parameter for the SLCD User Module and its configuration has a large impact on the performance of the user module. Configuring the user module with a CPU_Clock value that is too low may cause a large DC offset that can damage the LCD over a period of time. It is recommended to use a higher CPU_Clock value when possible. Selecting a CPU_Clock of 12 MHz provides the greatest balance between power and performance in most SLCD User Module configurations.
- The only devices that allow you to place a SLCD User Module in the same design with CapSense buttons are the CY8C21345, CY8C22x45 and CY8C28xxx families of devices. In all other cases, a design rule check will warn you that the analog MUX bus is not free.
- Even on the CY8C21345, CY8C22x45 and CY8C28xxx device families, you cannot place a SLCD User Module if the CapSense module consumes both analog MUX buses. You can use the SLCD User Module with CapSense, only if CapSense uses one analog MUX bus and SLCD uses the other.

Wizard

To access the Wizard, right-click on any block of the SLCD User Module in the Workspace Explorer and select the SLCD Wizard.

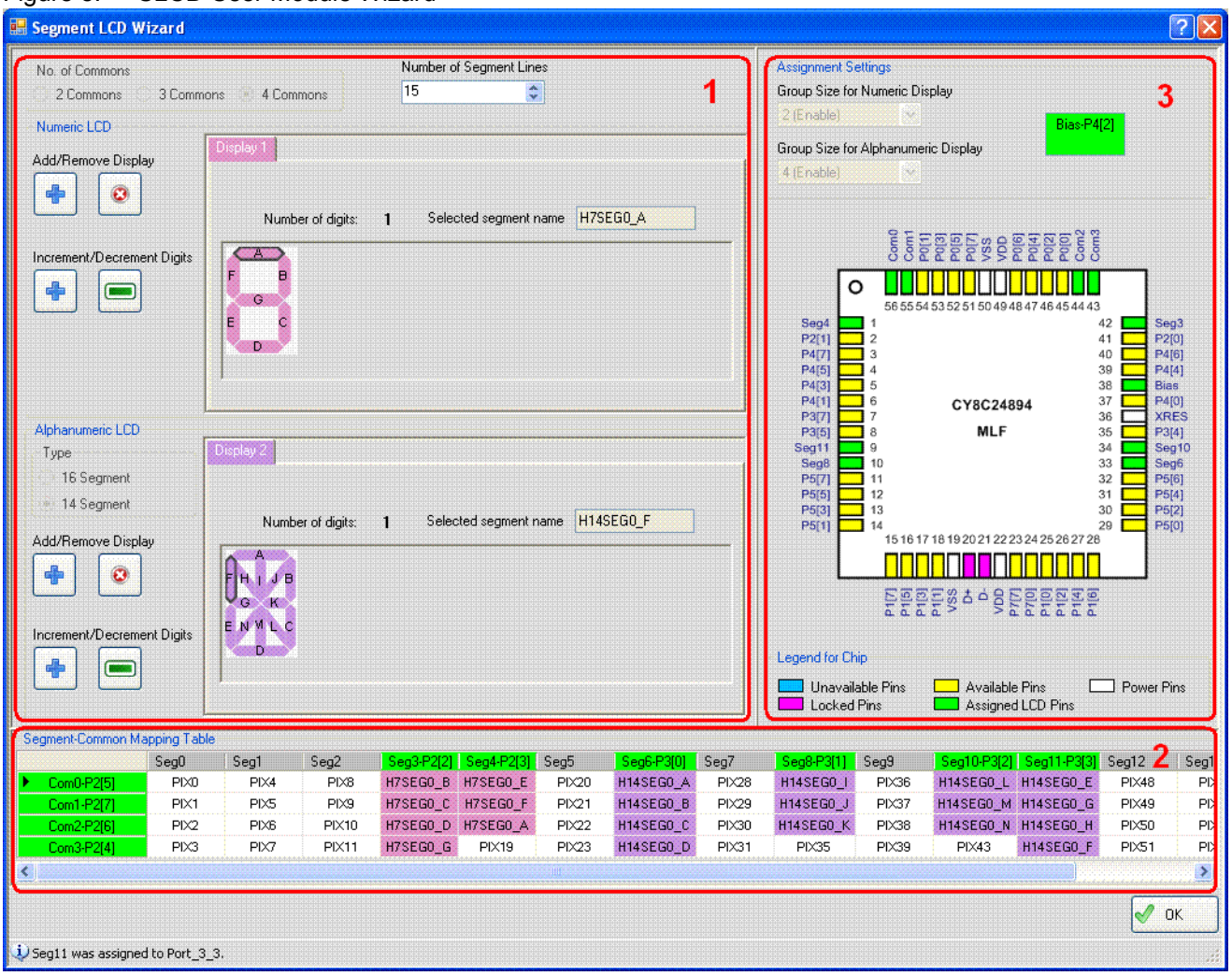
Figure 7. Accessing the Wizard



The general view of the SLCD User Module Wizard is shown in Figure 8.

After the wizard is complete and the project is built, constants for all displays with corresponding IDs are available in the SLCD.inc and SLCD.h files. Constants are generated for special symbols and free segments in the “Segment-Common Mapping Table” that are assigned segment lines in the chip view. Use these constants with the SLCD User Module APIs.

Figure 8. SLCD User Module Wizard



There are three main sections (shown with additional color coding in the figure) of the SLCD Wizard:

1. LCD Specification

Specify the number of common lines and segment lines (in a multiplexed LCD, multiple segments are connected together in a single column which is routed to a pin), add or remove numeric or alphanumeric displays, and add or remove digits to or from a display.

2. Segment-Common Mapping Table

Maps segments (a segment is a single bar in a display) and symbols (a symbol is an item such as a decimal point, colon, battery charge indicator, and more) to commons according to the LCD specification.

3. Pin Assignment

Assigns the bias pin, and common and segment lines to PSoC pins and sets group size for displays.

OK Button

This button executes two actions:

- Storing Wizard parameters

- Generation the asm and inc code fragments

LCD Specification Section

This section describes how to set common lines and segment lines, add or remove numeric and alphanumeric displays, and add or remove digits from or to the displays.

No. of Commons

The number of common lines can be 2, 3, or 4. The row count in the Mapping Table is equal to the number of common lines. You cannot change the number of common lines after you have assigned segments in the digit to segment lines in the Mapping Table. To change the number of common lines, unassign all segments from the segment lines.

Number of Segment Lines

The maximum number of segment control lines depends on the number of free I/Os present in the selected PSoC device, the number of common lines, and whether or not the external bias pin is used. The maximum number of segment lines number is:

Equation 5

$$SegmentLines_{MAX} \quad FreePSoCPins - CommonLines - ExternalBiasPin$$

The column count in the Mapping Table is equal to the number of segment lines.

Note If a segment from a digit is assigned to a segment line in the Mapping Table you cannot decrease the number of segment lines.

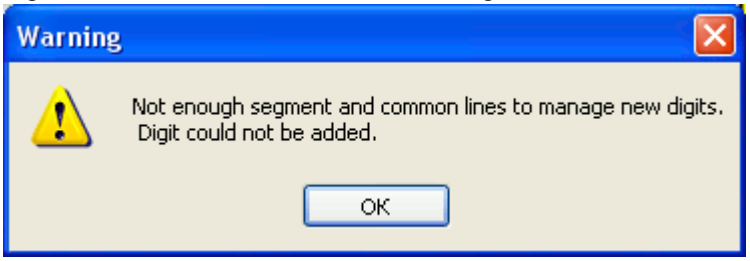
Numeric LCD

These controls enable you to add and remove numeric displays and add or remove digits from the display.

Add/Remove Display: Adds or removes a numeric display to or from the LCD panel. Each display has a unique name: “Display 1”, “Display 2”, and so on.

Increment/Decrement Digits: Adds or removes a 7-segment digit to or from the selected numeric display. If there are insufficient segment and common lines remaining, you will receive the following warning when you attempt to add a digit:

Figure 9. Insufficient Common and Segment Lines



LCD Panel Panels 1-N

This area contains all numeric displays and 7-segment digits.

Number of digits: Shows the number of 7-segment digits in the selected display.

Selected segment name: Shows the name of the selected segment in the display.

Alphanumeric LCD

Adds or removes alphanumeric displays, sets the type display, and adds or removes digits to or from the display.

Type: Chooses whether the digit is a 14- or 16-segment alphanumeric display. This control is disabled after the first alphanumeric display is added to the display, because you cannot mix 14- and 16-segments in the same display. To switch between 14- and 16-segment digits, remove all digits from the display.

Add/Remove Display: Add/remove the alphanumeric display to alphanumeric LCD panel. Each display has the unique name: “Display 1”, “Display 2”, and so on.

Increment/Decrement Digits: Add/remove 14-segment or 16-segment digit (corresponding to Type) to or from selected alphanumeric display. If the number of segments and common lines are not enough for new digit, the warning message shown in Figure 9 appears.

Alphanumeric LCD Panel

Contains the added alphanumeric displays and 14-/16-segment digits in the selected display.

“Number of digits” shows the number of 14-segment or 16-segment digits in selected display.

“Selected segment name” shows the name of selected segment in 14-segment or 16-segment digit.

Segment-Common Mapping Table Section

This section describes the mapping of segments from the digits and symbols to the commons.

The number of rows is equal to the number of common lines. The number of columns is equal to the number of segment lines. When you add or delete common and segment lines, rows and columns are added or subtracted from the table.

All free cells (not occupied by Digit Segments) in the Mapping Table can be used as special symbols, such as comma or period. These segments can be renamed by using a left-click and typing in a new name. The segment line which contains special symbols can be dragged from the Mapping Table to chip by holding a left-click on the appropriate segment line. By doing this, the Wizard generates the special symbol #defines in .h and .inc files corresponding to the chosen pixel name.

Moving A Segment from a Digit to a Mapping Table

Drag the segment from the digit to the Mapping Table. While you are dragging the segment, all free segment cells in Mapping Table are highlighted. Two different methods are available. If you set the group size (for either a numeric or alphanumeric display) to a specific number, use the **common indexed method**. If the group size is set to **None,** use the **universal method**.

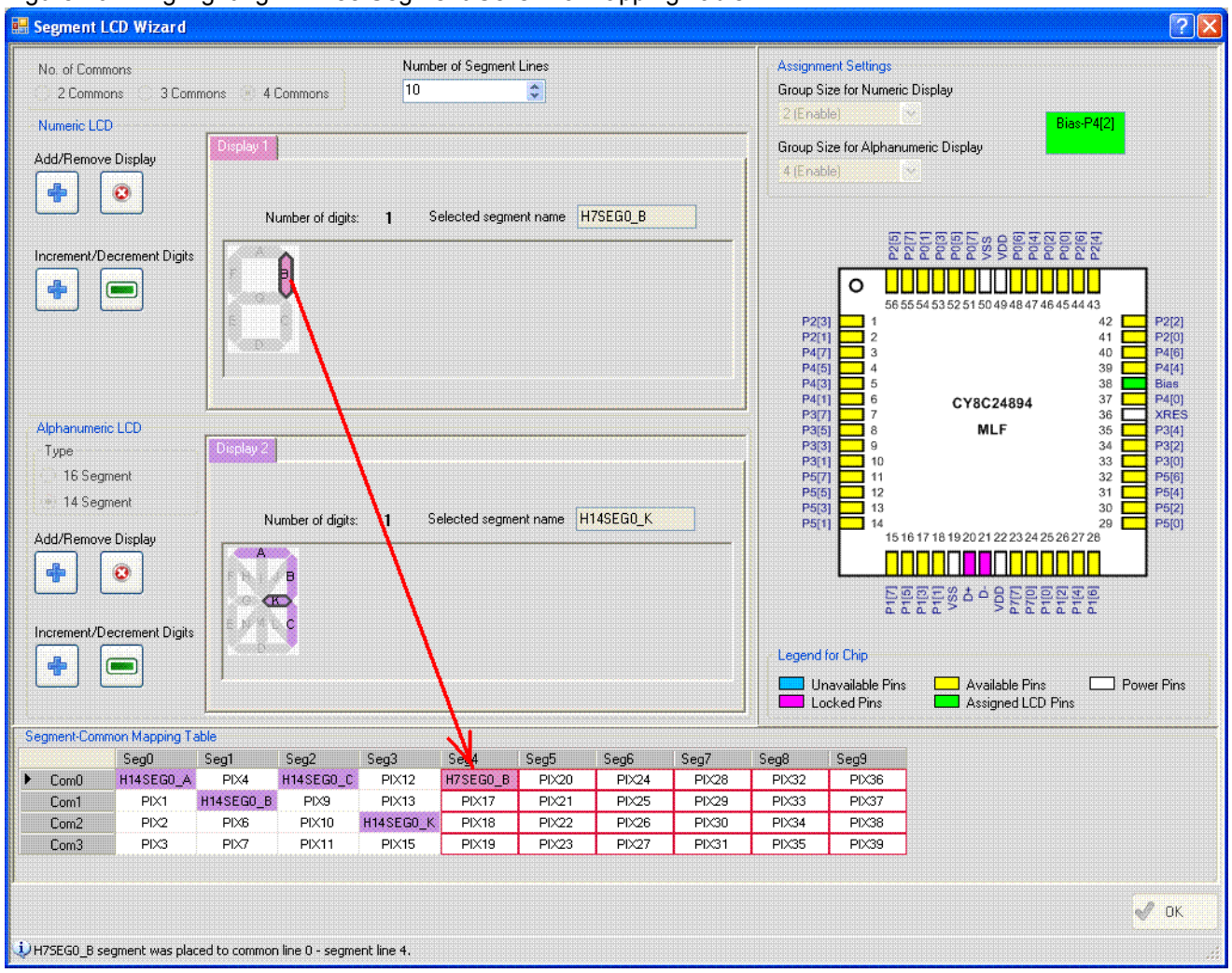
The common indexed method is useful for LCD displays that have the digits with same segment alignments relative common pins. For example, segment “a” should have the same common pin for all digits. This method reduces flash use on LCD displays with many digits. This also helps you assign segments correctly, because the wizard controls which segments are allowed.

The universal method is used for nonstandard displays that have digit segments allocated to different common pins. This method enables you assign a digit segment to any cell in the table.

Common Indexed Method

When the group size is set to a specific number, segments from a specific digit of the display can occupy only a specific segment lines that correspond to the group size. When you drag the first segment from a digit, all cells that are not occupied by other digits are highlighted:

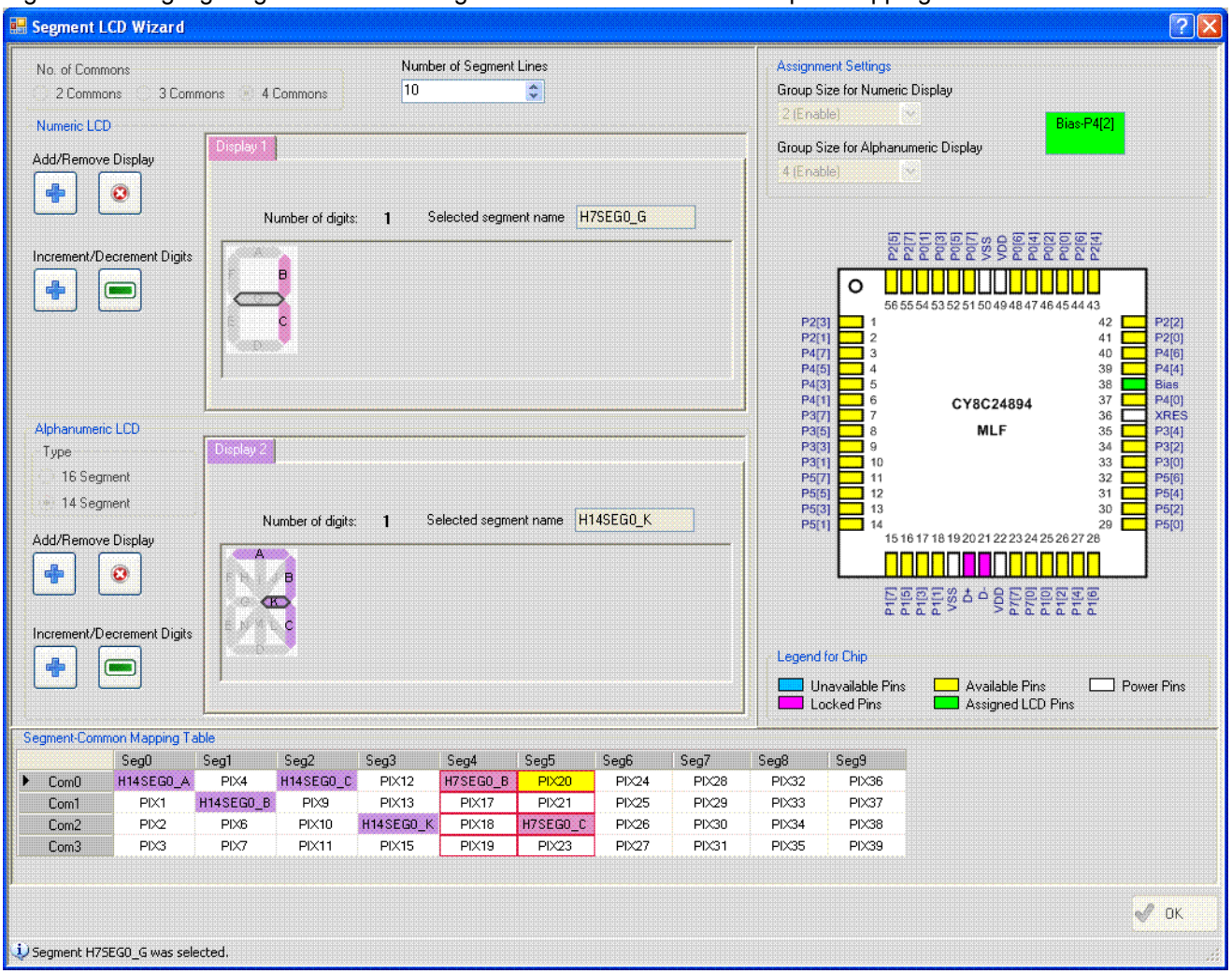
Figure 10. Highlighting All Free Segment Cells in a Mapping Table



In Figure 10, an LCD Display with 4 commons has segment lines Seg0, Seg1, Seg2 and Seg3 reserved for a digit from Display 2. So these segment lines are not available for the segment from Display 1.

If the second segment from the same digit is assigned to another segment line, all other segments must be assigned to only the segment lines shown in Figure 11:

Figure 11. Highlighting the Available Segment Cells of the Same Group in Mapping Table



Universal Method

When the Group Size is set to None, any segment from any digit can be assigned to any available segment cell in the Segment-Common Mapping Table. So one segment line can contain segments that belong to 7-segment or 14-/16-segment digits.

The table cells that are not assigned to any digit segments identify LCD symbols. The SLCD Wizard generates the appropriate constants in the SLCD.inc and SLCD.h files that can be used as the parameters for SLCD_EnableSymbol API.

Moving a Segment Line from the Mapping Table to the Chip

After you have mapped the segments to the mapping table, you must map the segment lines to pins on the device. When you click and drag a segment, all available pins on the device are highlighted.

The method that you use depends on whether you used the common indexed method or the universal method to assign segments to the table.

Common Indexed Method

All segment lines that are the members of the same group are highlighted and assigned to chip pins together. The left-most segment line is assigned to the lowest pin number. For example, all segments from digit of Display 2 (see Figure 10) are assigned to Seg0, Seg1, Seg2, and Seg3 in the Mapping Table. The following table indicates the possible variants of assigning these segment lines to chips:

Table 4. Possible Variants of Assigning Four Segment Lines to a Chip

Seg0	Seg1	Seg2	Seg3
Px_0	Px_1	Px_2	Px_3
Px_1	Px_2	Px_3	Px_4
Px_2	Px_3	Px_4	Px_5
Px_3	Px_4	Px_5	Px_6
Px_4	Px_5	Px_6	Px_7

In this table, “x” signifies the chip port.

If pin 4 is not free, only variant 1 is available for assigning these segment lines.

Universal Method

When the group size is set to none, any segment line can be assigned to any available pin on the chip.

After assigning segment lines to the chip, the name of assigned port and pin is shown near the segment line label, for example Seg1–Px[2].

Moving a Common Line from the Mapping Table to the Chip

You can drag common lines to pins on the device. While dragging, all available pins are highlighted. For CY8C28xxx and CY8C22x45 devices, common lines are assigned to even pins (on a single port) if analog MUX bus 1 is selected, and to odd pins on a port if analog MUX bus 0 is selected. For all other devices, common lines can be assigned to any pins on the same port.

Reset an Assigned Segment

To reset the assigned segment in a Mapping Table, right-click the segment in the mapping table and select Reset from the context menu. If a segment line is assigned to the chip and you reset the last segment in this line, the segment line is unassigned from the chip.

Pin Assignment Section

This section assigns the bias pin, common, and segment lines to the pin on the device and sets the group size for numeric and alphanumeric displays. This section differs a little for LCD drive techniques. The Wizard for Segment LCD driver using standalone GPIOs does not have the Analog MUX Bus controls and Bias pin in the Assignment Settings part.

Group Size for Numeric Display

Group Size for Alphanumeric Display

Determines the number of segment lines that form one group. Values available for group size depend on the number of commons and displays the display type:

Table 5. Group Size

Commons Number	Display Type	
	Numeric	Alphanumeric
2	4	8
3	3	6
4	2	4

In addition, you can select the “None” group size value. In this case, any segments from any digits can be assigned to all available segment cells in the Mapping Table, and any separate segment lines can be assigned to any of all free available pins in the chip.

This control is disabled when at least one segment from digit is assigned to the segment line in the Mapping Table and enabled when all segments from all digits are not assigned.

Note If the group size is set to nonzero value, the whole group of segment lines is assigned to contiguous pins on the chip view.

If LCD has the symbols that belong to the segment line that does not have the digit’s segments, then you can move the empty segment line to the chip. In this case, the SLCD Wizard generates the appropriate constants in the SLCD.inc and SLCD.h files that correspond to these symbols.

Bias Pin

The Bias Pin is only available if you choose the External Bias configuration. Depending on the device family you chose, the bias pin must be assigned to an odd or even pin. For the CY8C21345, CY8C22x45 and CY8C28xxx families, you can assign the bias pin to an odd pin if analog MUX bus 0 is selected, and to an even pin if MUX bus 1 is selected. For all other devices, you can assign the bias pin to any available pin.

Click and drag the bias pin to the chip. The pin name updates to show the assignment.

Chip View

Displays the selected PSoC device. Drag segments, common lines, and the bias pin (if used) to pins in the chip view to assign them. The previous sections discuss the rules for assigning pins.

The context menu allows you to reset only one pin, some of assigned pins, or even all assigned pins in chip. This menu appears after you right-click the “Chip” control or pin. It is a helpful feature for clearing “Chip” control.

This context menu tool has four possible entries:

- "Reset Selected Pin" – clears current selected pin (is available after clicking on the pin).
- "Reset All Pins of Numeric Display" – clears all pins that belong to the numeric displays.
- "Reset All Pins of Alphanumeric Display" – clears all pins that belong to the alphanumeric displays.
- "Reset All Pins" – clears all assigned pins.

Following are some of the additional features:

- If there are no assigned pins in "Chip" control, this menu is not opened.
- If there are assigned pins that belong to the numeric displays only, the "Reset All Pins of Alphanumeric Display" menu item is disabled.
- If there are assigned pins that belong to the alphanumeric displays only, the "Reset All Pins of Numeric Display" menu item is disabled.

Example of Using the SLCD Wizard

The following example demonstrates the SLCD User Module Wizard features and capabilities. In this example, the standard VIM-878 LCD and CY3214 PSoC Eval USB Board are used. VIM-878 is a 4-Commons, 14-segment alphanumeric LCD, which has eight alphanumeric symbols. Here two alphanumeric LCD displays, with four digits each, are added and assigned to the CY8C24794-24LFXI chip using the SLCD Wizard. Therefore, one LCD can be used as two separate displays.

1. Create a new project in PSoC Designer based on the CY8C24794-24LFXI chip.
2. Place the SLCD User Module with an internal bias generator.
3. Set the SLCD User Module parameters: LCD Clock, Refresh Rate, and Contrast Level.
4. Right-click on the SLCD User Module and select “SLCD Wizard…” from menu.
5. Set the “No. of Commons” radio button control to “4 Commons” (see section 1 in Figure 8).
6. Enter “37” in the “Number of Segment Lines” numeric up-down control (Section 2 in Figure 8). In this example, 37 Segment Lines are used because the LCD has 36 pins and Segment Lines in the Wizard are numbered from 0. Therefore, in this project the Segment Lines #0 and #17-20 will not be used. In this example, LCD 17-20 pins (see Table 6) are used for Commons. This corresponds to the numbering between the LCD pins and Segment Lines and reduces the probability of errors. Refer to the VIM-878 LCD datasheet for details.
7. Create an alphanumeric display, “Display1”: press the “+” button below the title “Add/Remove Display” of the alphanumeric LCD section (see section 1 in Figure 8). The first display will be added.
8. Add three digits to the display: press the “+” button below the title “Increment/Decrement Digits” of the alphanumeric LCD section (see section 1 in Figure 8). Repeat this operation until you have four alphanumeric digits.
9. Create an alphanumeric display “Display2”: press the “+” button below the title “Add/Remove Display” of the alphanumeric LCD section (see section 1 in Figure 8). The second display will be added.
10. Add three digits to the display: press the “+” button below the title “Increment/Decrement Digits” of the alphanumeric LCD section (see section 1 in Figure 8). Repeat this operation until you have four alphanumeric digits.
11. Select “None” from the drop-down menu of “Group Size of Alphanumeric Display” (section 3 in Figure 8).
12. Drag and drop the segments from digits of Display1 and Display2 (Section 1 in Figure 8) to the Segment-Common Mapping Table (Section 2 in Figure 8). All segments should correspond to the LCD datasheet.

Note 1: Pay attention to real LCD pins correspondence with segment lines on the Segment-Common Mapping Table. The LCD may have unused pins. You can shift the pin numbering or increase the “Number of Segment Lines” parameter.

Table 6. Example of Pin Mapping Table in the LCD Datasheet

Pin	COM0	COM1	COM2	COM3
1	ID	IE	IF	CA1
2	IL	IK	IJ	1I
3	2D	2E	2F	CA2
4	2L	2K	2J	2I
5	3D	3E	3F	CA3
6	3L	3K	3J	3I
7	4D	4E	4F	CA4
8	4L	4K	4J	4I
9	5D	5E	5F	CA5
10	5L	5K	5J	5I
11	6D	6E	6F	CA6
12	6L	6K	6J	6I
13	7D	7E	7F	CA7
14	7L	7K	7J	7I
15	8D	8E	8F	CA8
16	8L	8K	8J	8I
17	COM0	–	–	–
18	–	COM1	–	–
19	–	–	COM2	–
20	–	–	–	COM3
21	DP8	8C	8B	8A
22	8M	8N	8G	8H
23	DP7	7C	7B	7A
24	7M	7N	7G	7H
25	DP6	6C	6B	6A
26	6M	6N	6G	6H
27	DP5	5C	5B	5A
28	5M	5N	5G	5H

Pin	COM0	COM1	COM2	COM3
29	DP4	4C	4B	4A
30	4M	4N	4G	4H
31	DP3	3C	3B	3A
32	3M	3N	3G	3H
33	DP2	2C	2B	2A
34	2M	2N	2G	2H
35	DP1	1C	1B	1A
36	1M	1N	1G	1H

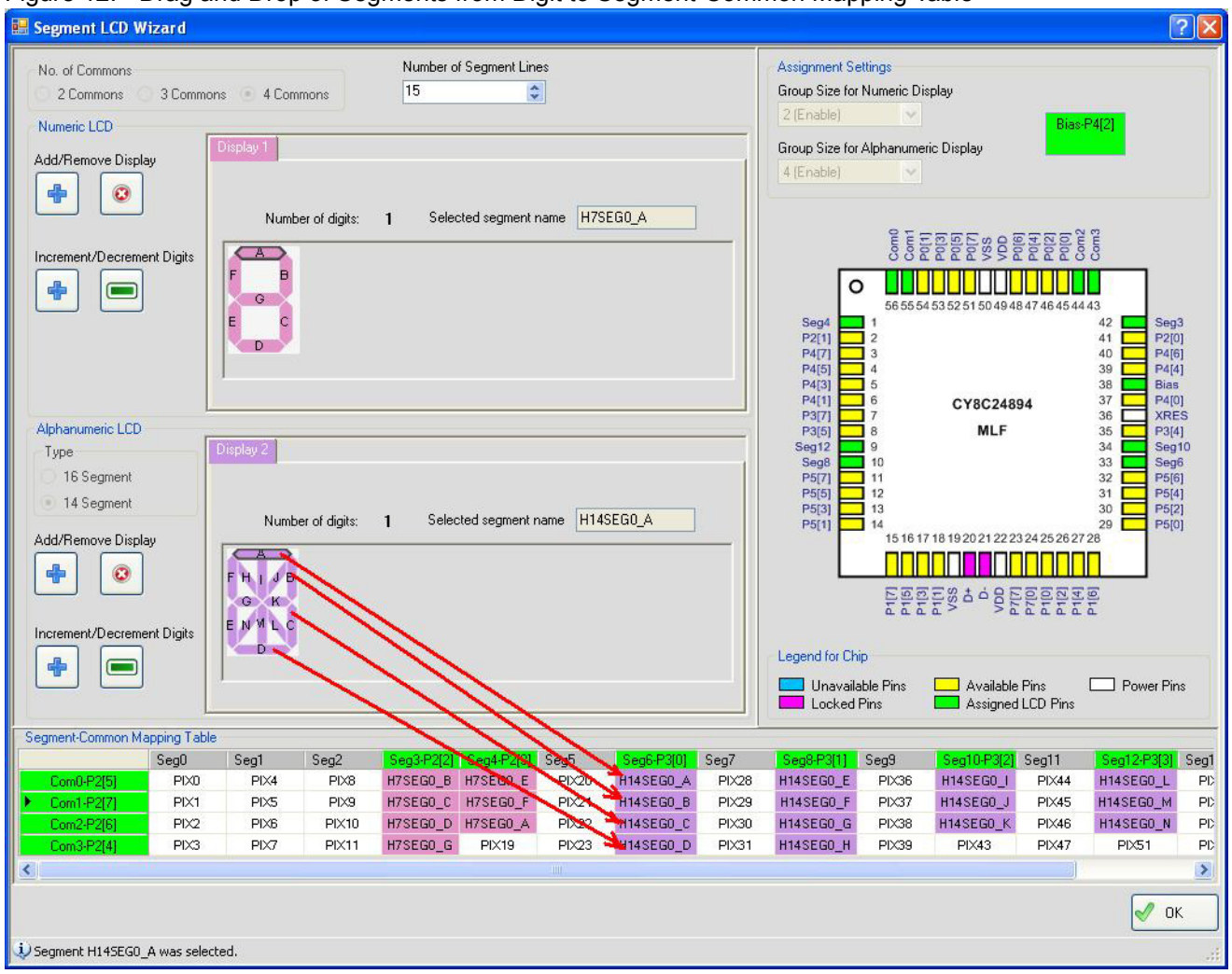
Note 2: The LCD User Module's APIs control the Displays and Digits with IDs that are numbered form 0. In this example, the LCD is divided into two parts. Therefore, in the SLCD Wizard, 0-3 digits of Display1 correspond to 1-4 digits of the LCD datasheet and 0-3 digits of Display2 correspond to 5-8 digits of the LCD datasheet.

Note 3: The redundant segments of the Segment-Common Mapping Table are used for symbols such as "comma" and "dot". You can rename them and the wizard generates the corresponding constants in the.inc and .h files.

Further the "drag and drop" operation for the digit is shown more detailed (or digit #3 of Display2 in the Wizard).

To assign the digit segments to the Segment-Common Mapping Table, refer to the Pins Mapping Table of the LCD datasheet. The digit occupies 15, 16, 21, and 22 pins on the LCD. Therefore, drag and drop each segment from the digit to the segment lines of the Segment-Common Mapping Table so that it corresponds to the LCD datasheet. For example, segment "8D" drops to the segment line "Seg15" of Com0 row (its name in the Segment-Common Mapping Table is "H14SEG7_D"), "8K" – "Seg16" of Com1 row, and so on (see Figure 12).

Figure 12. Drag and Drop of Segments from Digit to Segment-Common Mapping Table



13. After all segments are assigned to Segment-Common Mapping Table they can be assigned to the pins of the PSoC chip in the Pin Assignment section (Section 3 in Figure 1). You should drag and drop the Commons first column and segment lines from Segment-Common Mapping Table to Pin Assignment section (Figure 5).

Figure 13. Drag and drop of the Commons and Segment lines to the Chip

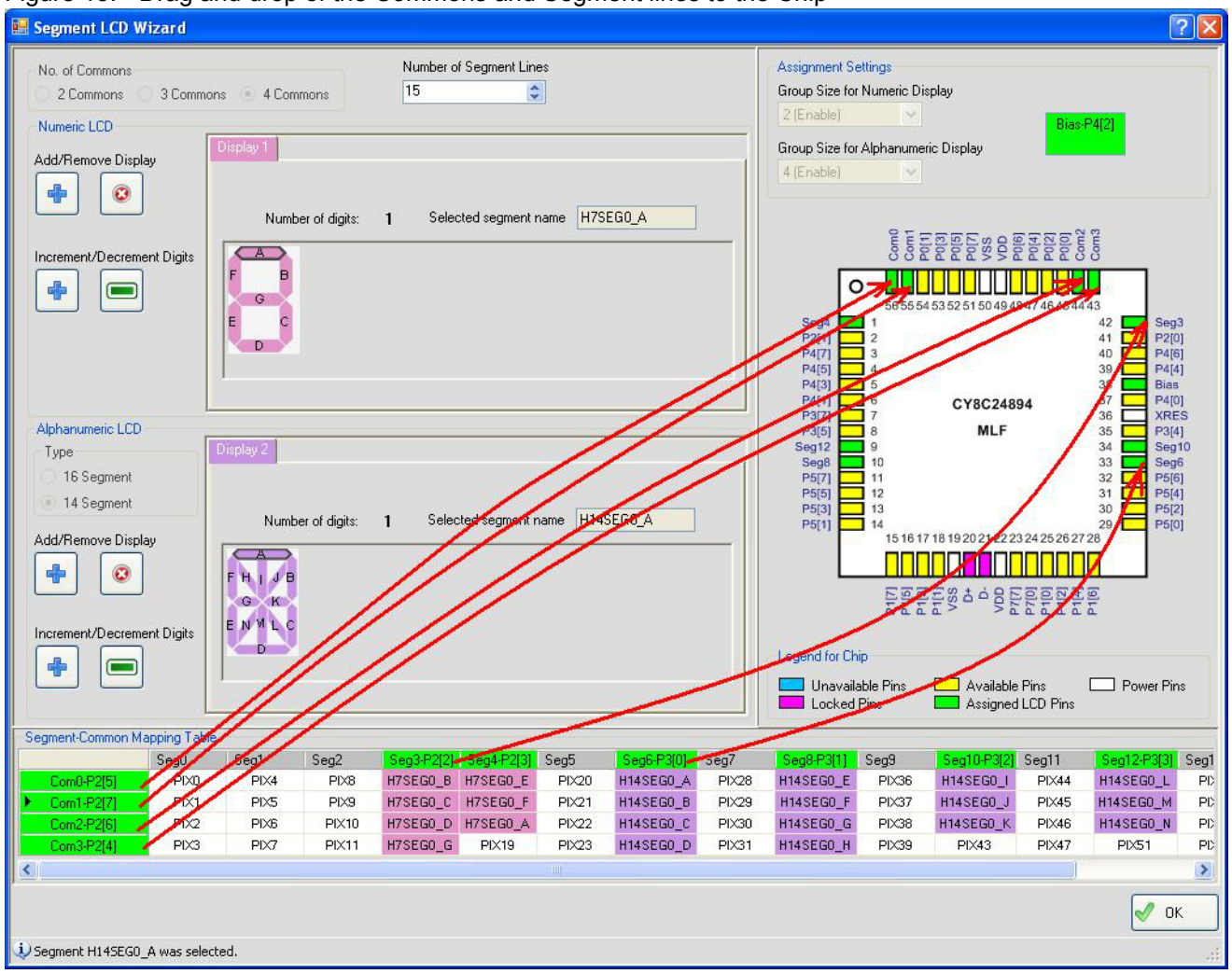


Table 6 shows the ports and pins of the example that are used for Commons and Segment lines.

Table 7. Example of Ports and Pins Assignment on the Chip

	PSoC Ports and Pins
Commons	P0(0), P0(2), P0(4), P0(6)
Digit 1	P1(0)–P1(3)
Digit 2	P1(4)–P1(7)
Digit 3	P2(0)–P2(3)
Digit 4	P2(4)–P2(7)

	PSoC Ports and Pins
Digit 5	P3(0)–P3(3)
Digit 6	P3(4)–P3(7)
Digit 7	P4(0)–P4(3)
Digit 8	P4(4)–P4(7)

14. Click **OK** to generate the project.
15. Add the following code to main.c:

```
//--------------------------------------------------------------------------
// C main line
//--------------------------------------------------------------------------

#include <m8c.h>        // part specific constants and macros
#include "PSoCAPI.h"    // PSoC API definitions for all User Modules

char theStr[] = "PSOC";   // Define RAM string
void main(void)
{
    // Insert your main routine code here.
        M8C_EnableGInt;
        SLCD_1_Start(SLCD_1_NORMAL_STATE);
        SLCD_1_ChangeContrast(SLCD_1_SET_TO_MAX, 0);
        SLCD_1_PrintString(0, 0, theStr, 4);
        SLCD_1_PrintHexInt(1,0x1234);

        while(1)
        {

        }
}
```

16. Build the project.
17. Program the part.
18. Connect the LCD to the PSoC chip according to Table 6.
19. Apply 3.3 V to the board.

The LCD displays “PSOC1234”.

Parameters and Resources

LCD Clock Source

Sets the clock source for LCDTimer.

Table 8. Settings

Range	Default	Comment
VC1, VC2, VC3, CPU_32K, Row_x_Output_0, Row_x_Output_1, Row_x_Output_2, Row_x_Output_3, Row_x_Input_0, Row_x_Input_1, Row_x_Input_2, Row_x_Input_3, Row_x_Broadcast, Disable	VC1 (VC2*)	This sets the clock source for the SLCD User Module

The minimum LCD Timer Clock equals $20nF_r$, where:

n – the number of Commons

F_r – LCD Refresh Rate, measured in Hz

The lower LCD Timer Clock values do not guarantee the whole Contrast Level Range. Also, LCD Refresh Rate error is introduced near the minimum LCD Timer Clock.

The maximum LCD Timer Clock is limited by LCD Timer resolution.

Note If a CapSense User Module is used in the design, you must take into account that the CapSense User Module can change VC1, VC2, and VC3 values based on the scanning speed and resolution. You must be aware of these changes and put the appropriate value in the LCD Clock Value parameter. If sleep mode operation is needed, select CPU_32K. If CPU_32K is selected, LCDTimer uses the ILO if the 32K_Select parameter (global settings) is internal or the ECO if it is external. To use CapSense with VC1, VC2, VC3 or one of the rows as the clock, enter a clock frequency value in the LCD Clock Value parameter.

*SLCD User Module with Timer 16 HW/SW does not have VC1 value.

LCD Clock Value

Gives the clock frequency value for the LCDTimer. Enter a clock value in kHz. When set to the default value of 0, the user module calculates the selected clock source value based on the dividers set in global settings. When the LCD clock source differs from VC1, VC2, VC3, or CPU_32K, then the LCD Clock Value parameter should be set manually.

If a CapSense User Module is used in the design, you must remember that the CapSense User Module APIs can change VC1, VC2, and VC3 values. Therefore, clock sources such as CPU_32K or rows, are recommended if the CapSense User Module is used.

Table 9. Settings

Type	Range	Default	Comment
Integer	100 to 2000 (in kHz)	0	Sets the clock value for LCDTimer

Refresh Rate

Sets the refresh rate of LCD. The possible options are 30-150 Hz when VC1, VC2, VC3 or rows is the clock source, or 30-60 Hz when CPU_32K is selected.

Note The timer period is calculated based on LCDTimer Input Clock, refresh rate, and contrast level.

Table 10. Settings

Type	Range	Default	Comment
Decimal Integer	Min = 30, Max = 150	50	Sets the clock value for LCDTimer, when VC1, VC2, VC3 or rows is selected as Clock Source
Decimal Integer	Min = 30, Max = 60	50	Sets the clock value for LCDTimer, when CPU_32K is selected as Clock Source

Contrast Level

Sets the Contrast Level of LCD.

Table 11. Settings

Type	Range	Default	Comment
Text	Min, Max, and Med	Med	Sets the contrast level of LCD

Note For an analog MUX bus drive, contrast is adjusted by setting the dead time between LCD frames. Contrast Level Max value is calculated by the following equation:

Equation 6

$$Max = \left(\frac{\frac{1}{Refresh\ Rate} - 500\mu s}{2 \times n} \right) \times Timer\ Input\ Frequency$$

In Equation 6:

Max - maximum value of Contrast Level;

Timer Input Frequency - LCD Timer input clock frequency value;

n - number of commons.

Contrast Level Min value is calculated by the following equation:

Equation 7

$$Min = Timer\ Input\ Frequency \times 500\mu s$$

In Equation 7:

Min - minimum value of Contrast Level

Timer Input Frequency - LCD Timer input clock frequency value

Med - medium value of Contrast Level

CPU Loading

The LCD is refreshed in the LCDTimer interrupt routine. The LCDTimer ISR determines the LCD Refresh Rate and Contrast. In addition, it updates the data on the ports for segments and commons. The

LCDTimer ISR period depends on LCD Clock Value, Number of Commons, LCD Refresh Rate, and Contrast Level.

If the LCDTimer Period becomes too low, the Timer Overflow Rate increases. This results in increased interrupt overhead on the processor.

To eliminate the processor overflow the minimum value of LCDTimer ISR Period is limited to 0.5 msec (or 0.25 msec when standalone GPIO technique is used). If you set the SLCD User Module parameters that cause the LCDTimer to the limit, then the LCD Refresh Rate and Contrast Level differ from their values in the user module properties.

The Design Rule Check warning appears in case of LCDTimer period limiting.

Conflicts in Port Control

The SLCD User Module uses shadow registers named Port_x_Data_SHADE to update the data at segment and common pins, where x is the port number. If you want to write to the port, which is shared by the segments and commons, then it is important to use the same shadow registers created by the SLCD User Module. The following example shows how to make use of shadow registers to update one of the ports:

```
Port_2_Data_SHADE |= 0x10;PRT2DR = Port_2_Data_SHADE;
```

This example sets bit 4 of Port 2 to logic 1.

Application Programming Interface

The Application Programming Interface (API) functions are provided as part of the user module to allow you 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.

Each time a user module is placed, it is assigned an instance name. By default, PSoC Designer assigns the SLCD_1 to the first instance of this user module in a given project. It can be changed to any unique value that follows the syntactic rules for identifiers. The assigned instance name becomes the prefix of every global function name, variable and constant symbol. In the following descriptions the instance name has been shortened to SLCD for simplicity.

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 also ensure their code observes this policy. Though some user module API functions may leave A and X unchanged, there is no guarantee they may do so in the future.

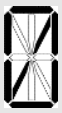
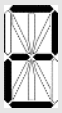
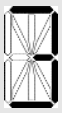
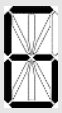
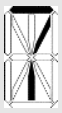
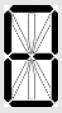
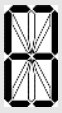
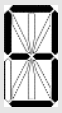
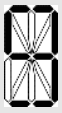
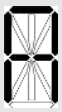
For Large Memory Model devices, it is also the caller's responsibility to preserve any value in the CUR_PP, IDX_PP, MVR_PP, and MVW_PP registers. Even though some of these registers may not be modified now, there is no guarantee that will remain the case in future releases.

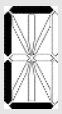
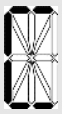
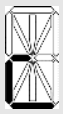
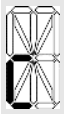
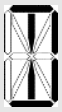
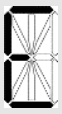
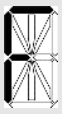
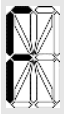
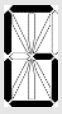
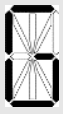
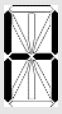
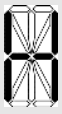
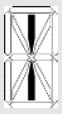
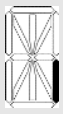
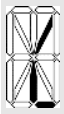
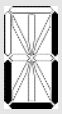
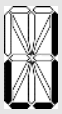
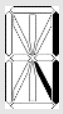
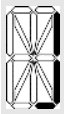
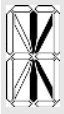
Entry Points are supplied to initialize, start, and stop the SLCD User Module. In all cases, the instance name of the module replaces the SLCD prefix shown in the following entry points. Failure to use the correct instance name is a common cause of syntax errors.

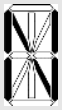
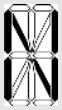
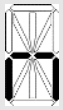
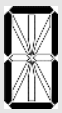
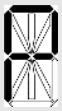
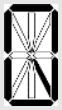
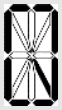
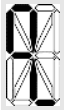
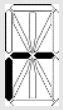
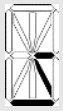
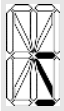
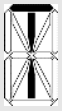
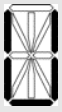
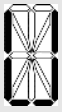
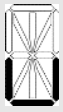
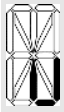
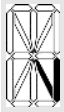
It is strongly recommended to enable global interrupts before calling the SLCD_Start() API to avoid damage to the LCD. In addition, always call the SLCD_Stop() API before disabling global interrupts. If

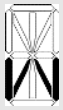
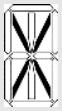
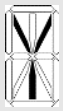
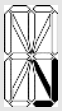
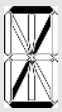
sleep mode is used and LCD operation is not required, then the LCD SLCD_Stop() API is required before setting the device to sleep.

Table 12. SLCD User Module characters table

Character	14-segment Display	16-segment Display	Character	14-segment Display	16-segment Display
0					
1					
2					
3					
4					
5					
6					
7					
8					
9					
A			a		

Character	14-segment Display	16-segment Display	Character	14-segment Display	16-segment Display
B			b		
C			c		
D			d		
E			e		
F			f		
G			g		
H			h		
I			i		
J			j		
K			k		
L			l		
M			m		

Character	14-segment Display	16-segment Display	Character	14-segment Display	16-segment Display
N			n		
O			o		
P			p		
Q			q		
R			r		
S			s		
T			t		
U			u		
V			v		

Character	14-segment Display	16-segment Display	Character	14-segment Display	16-segment Display
W			w		
X			x		
Y			y		
Z			z		

SLCD_Start

Description:

Enables the user module. You can set the refresh rate and contrast level in the device editor.

This function performs the following tasks:

- Configures all the pins
- Calculates Timer period for active COM time and dead state depending on the refresh rate and contrast level set
- Starts LCDBias RefMux module (if available)
- Enables LCDTimer interrupt

C Prototype:

```
void  SLCD_Start(BYTE InvertState)
```

Assembly:

```
mov   A, InvertState
lcall  SLCD_Start
```

Parameters:

InvertState - Specifies Normal State (normal display) or InvertedState (inverting display).

Symbolic Name	Value	Description
NORMAL_STATE	0	Normal display
INVERTED_STATE	1	Inverting display

Return Value:

None

Side Effects:

See Note ** at the beginning of the API section.

It is strongly recommended to enable global interrupts before calling the SLCD_Start() API to avoid damage to the LCD. In addition, always call the SLCD_Stop() API before disabling global interrupts. If sleep mode is used and LCD operation is not required, then the LCD SLCD_Stop() API is required before setting the device to sleep.

SLCD_Stop

Description:

Stops the functioning of the user module.

This stops the timer module and puts all the LCD pins to High Z Analog. If there is RefMux module in the design, then it shuts off the power to that module.

C Prototype:

```
void  SLCD_Stop(void)
```

Assembly:

```
lcall  SLCD_Stop
```

Parameters:

None

Return Value:

None

Side Effects:

See Note ** at the beginning of the API section.

It is strongly recommended to enable global interrupts before calling the SLCD_Start() API to avoid damage to the LCD. In addition, always call the SLCD_Stop() API before disabling global interrupts. If sleep mode is used and LCD operation is not required, then the LCD SLCD_Stop() API is required before setting the device to sleep.

SLCD_SetContrast

Description:

Sets the contrast level of LCD. It takes value in percent (0-100).

C Prototype:

```
void  SLCD_SetContrast(BYTE PercentContrast)
```

Assembly:

```
mov   A, PercentContrast
lcall  SLCD_SetContrast
```

Parameters:

PercentContrast: This can be any value between 0 to 100. If 0 is passed to this function, minimum contrast is set and passing 100 will give maximum contrast on LCD.

Return Value:

None

Side Effects:

See Note ** at the beginning of the API section.

SLCD_ChangeContrast

Description:

Changes the contrast level of LCD.

C Prototype:

```
BYTE  SLCD_ChangeContrast(BYTE bOption, BYTE bDelta)
```

Assembly:

```
mov   A, bOption
mov   X, bDelta
lcall SLCD_ChangeContrast
```

Parameters:

bOption: This can be any of the following:

Option	Value	Purpose
INCREASE_CONTRAST	0	Increases the contrast level
DECREASE_CONTRAST	1	Decreases the contrast level
SET_TO_MAX	2	Sets the contrast to max value
SET_TO_MIN	3	Sets the contrast to min value
SET_TO_MED	4	Sets the contrast to medium value

bDelta: This value determines by how much counts the period value of the timer (during active COM time) needs to be increased or decreased. This parameter is valid only when INCREASE_CONTRAST or DECREASE_CONTRAST is passed in the Option field of this API. For other options, the bDelta field value is ignored. bDelta can be anywhere between 1 to 20 in the API; the resultant timer period is limited so that it does not go beyond the range.

Return Value:

BYTE value: This API returns byte value indicating whether the API has reached the maximum or minimum value.

Return Value	Description
0x1	Timer period has reached maximum value*
0x2	Timer period has reached minimum value*
0x0	If timer period is within the limits

*Note that the API returns non-zero value only when INCREASE_CONTRAST or DECREASE_CONTRAST option is passed and Timer Period reaches the limit with the delta value provided.

Side Effects:

See Note ** at the beginning of the API section.

SLCD_GetInterruptStatus

Description:

Gets the timer interrupt status and returns to the user application.

C Prototype:

```
BYTE  SLCD_GetInterruptStatus(void)
```

Assembly:

```
lcall  SLCD_GetInterruptStatus
```

Parameters:

None

Return Value:

Return type is BYTE. One flag is maintained by the user module, which is set to 1 in the LCDTimer interrupt routine. API returns this flag value when called. This flag is automatically cleared at the end of this API.

Side Effects:

See Note ** at the beginning of the API section.

If sleep mode is used and if LCD operation is not required, then it is strongly recommended to call LCD SLCD_Stop() API before setting the device to sleep.

SLCD_ClearAll

Description:

Clears all the segments of the LCD.

C Prototype:

```
void  SLCD_ClearAll(void)
```

Assembly:

```
lcall  SLCD_ClearAll
```

Parameters:

None

Return Value:

None

Side Effects:

See Note ** at the beginning of the API section.

SLCD_ClearDisplay

Description:

Clears complete alphanumeric or numeric display section of the LCD.

If the LCD contains multiple sections of numeric or alphanumeric displays, each of these sections have a unique DisplayID. Only the section whose DisplayID is passed to this function is cleared.

C Prototype:

```
void  SLCD_ClearDisplay(BYTE DisplayID)
```

Assembly:

```
mov   A, DisplayID
lcall  SLCD_ClearDisplay
```

Parameters:

DisplayID - Specifies which display section (alphanumeric or numeric) of the LCD needs to be cleared.

The SLCD.inc and SLCD.h files have unique constant declarations for each DisplayID.

Return Value:

None

Side Effects:

See Note ** at the beginning of the API section.

SLCD_PrintDigit

Description:

Prints the number in the specified digit position of the numeric display or alphanumeric display in hex format.

C Prototype:

```
void  SLCD_PrintDigit(BYTE DisplayID, BYTE DigitPosition, BYTE Number)
```

Assembly:

```
mov   A, Number
push  A
mov   A, DigitPosition
push  A
mov   A, DisplayID
push  A
lcall  SLCD_PrintDigit
```

Parameters:

DisplayID - Specifies a particular section of the Alphanumeric or Numeric Display.

DigitPosition - Specifies the Digit position of Numeric/Alphanumeric display where the number needs to be displayed. It can take values from 0 to (n-1), where n is the number of digits in the Display.

Number - Data to be displayed. This can take values from 0 to 15.

The SLCD.inc and SLCD.h files have unique constant declarations for each DisplayID.

Return Value:

None

Side Effects:

See Note ** at the beginning of the API section.

SLCD_ClearDigit

Description:

Clears specified digit of alphanumeric or numeric display section of LCD.

If the LCD contains multiple sections of numeric or alphanumeric displays, each of these sections have a unique DisplayID. Only the digit whose DisplayID and DigitPosition are passed to this function will be cleared.

C Prototype:

```
void  SLCD_ClearDigit(BYTE DisplayID, BYTE DigitPosition)
```

Assembly:

```
mov   A, DisplayID
mov   X, DigitPosition
lcall  SLCD_ClearDigit
```

Parameters:

DisplayID - Specifies a particular section of Alphanumeric or Numeric Display.

DigitPosition - Specifies the Digit position of Numeric/Alphanumeric display where Number needs to be cleared.

The SLCD.inc and SLCD.h files have unique constant declarations for each DisplayID.

Return Value:

None

Side Effects:

See Note ** at the beginning of the API section.

SLCD_PrintHexInt

Description:

Prints the specified Number on numeric display or alphanumeric display in Hex Format.

C Prototype:

```
void  SLCD_PrintHexInt(BYTE DisplayID, int Number)
```

Assembly:

```
mov   A, >Number
push  A
mov   A, <Number
push  A
mov   A, DisplayID
push  A
lcall  SLCD_PrintHexInt
```

Parameters:

DisplayID - Specifies a particular section of numeric or alphanumeric display.

Number - Data to be displayed. It can take values from 0-0xFFFF, given that the specified display section has at least 4 digits. If the number exceeds the capability of the display section, then some digits (most significant) are discarded. If the display section contains more than 4 digits then leaving the least significant 4 digits, other digits can be cleared by SLCD_ClearDigit API.

The SLCD.inc and SLCD.h files have unique constant declarations for each DisplayID.

Return Value:

None

Side Effects:

See Note ** at the beginning of the API section.

SLCD_PrintHexByte

Description:

Prints the specified Number on numeric or alphanumeric display in Hex Format.

C Prototype:

```
void  SLCD_PrintHexByte(BYTE DisplayID, BYTE Number)
```

Assembly:

```
mov   A, DisplayID
mov   X, Number
lcall  SLCD_PrintHexByte
```

Parameters:

DisplayID - Specifies particular section of numeric or alphanumeric display.

Number - Data to be displayed. It can take values from 0-0xFF, given that the specified display section has at least 2 digits. If the number exceeds the capability of the display section, then some digits (starting from most significant) are discarded. If the numeric or alphanumeric contains more than 4 digits, then leaving the least significant two digits, the other digits can be cleared by SLCD_ClearDigit API.

The SLCD.inc and SLCD.h files have unique constant declarations for each DisplayID.

Return Value:

None

Side Effects:

See Note ** at the beginning of the API section.

SLCD_PrintDecNumber

Description:

Prints the specified Number on numeric or alphanumeric display in decimal Format.

C Prototype:

```
void  SLCD_PrintDecNumber(BYTE DisplayID, int Number)
```

Assembly:

```
mov   A, >Number
push  A
mov   A, <Number
push  A
mov   A, DisplayID
push  A
lcall  SLCD_PrintDecNumber
```

Parameters:

DisplayID - Specifies particular section of numeric or alphanumeric display.

Number - Data to be displayed. It can take values from 0-65535, given that the specified display section has at least 5 digits. If the number exceeds the capability of the display section, then some digits (starting from most significant) are discarded. If the display section contains more than 5 digits, then the Data is displayed on the least significant 5 digits. Other digits can be cleared by SLCD_ClearDigit API.

The SLCD.inc and SLCD.h files have unique constant declarations for each DisplayID.

Return Value:

None

Side Effects

See Note ** at the beginning of the API section.

SLCD_PrintString

Description:

Prints the string specified by pointer ‘sRAMString’ on alphanumeric display. The length of the string is specified by NumberOfChars. This API is valid for Alphanumeric LCD only.

C Prototype:

```
void  SLCD_PrintString(BYTE DisplayID, BYTE StartDigitPosition, Char * sRAMString, BYTE NumberOfChars)
```

Assembly:

```
mov   A, NumberOfChars
push  A
mov   A, >sRAMString
push  A
mov   A, <sRAMString
push  A
mov   A, StartDigitPosition
push  A
mov   A, DisplayID
push  A
lcall  SLCD_PrintString
```

Parameters:

DisplayID - Specifies a particular section of the alphanumeric display.

StartDigitPosition - Specifies from which digit position of the alphanumeric display the string needs to be printed. It can take values from 0 to (n-1), where n is the number of digits in the Display.

*sRAMString: Pointer to the string in RAM.

NumberOfChars - Number of characters in a string.

The SLCD.inc and SLCD.h files have unique constant declarations for each DisplayID.

Return Value:

None

Side Effects

See Note ** at the beginning of the API section.

SLCD_PrintCString

Description:

Prints the string specified by pointer ‘sROMString’ on alphanumeric display. The length of the string is specified by NumberOfChars. This API is valid for Alphanumeric LCD only.

C Prototype:

```
void  SLCD_PrintCString(BYTE DisplayID, BYTE StartDigitPosition, Const Char
*sROMString, BYTE NumberOfChars)
```

Assembly:

```
mov   A, NumberOfChars
push  A
mov   A, >sROMString
push  A
mov   A, <sROMString
push  A
mov   A, StartDigitPosition
push  A
mov   A, DisplayID
push  A
lcall  SLCD_PrintCString
```

Parameters:

DisplayID - Specifies a particular section of alphanumeric display.

StartDigitPosition - Specifies from which digit position of the alphanumeric display the string needs to be printed. It can take values from 0 to (n-1), where n is the number of digits in the Display.

*sROMString: Pointer to the string in ROM.

NumberOfChars - Number of characters in a string.

The SLCD.inc and SLCD.h files have unique constant declarations for each DisplayID.

Return Value:

None

Side Effects

See Note ** at the beginning of the API section.

SLCD_PrintCharacter

Description:

Prints the character specified by the ASCII value stored in ‘Data’ on alphanumeric display. This API is valid for Alphanumeric LCD only.

C Prototype:

```
void  SLCD_PrintCharacter(BYTE DisplayID, BYTE DigitPos, char Data)
```

Assembly:

```
mov   A, Data
push  A
mov   A, DigitPos
push  A
mov   A, DisplayID
```

```
push  A
lcall  SLCD_PrintCharacter
```

Parameters:

DisplayID - Specifies a particular section of the alphanumeric display.

DigitPos - Specifies the position in the display where the character needs to be printed. It can take values from 0 to (n-1), where n is the number of digits in the Display.

Data - Holds the ASCII value of character that needs to be printed. It can be a number, upper case, or smaller case alphabets.

The SLCD.inc and SLCD.h files have unique constant declarations for each DisplayID.

Return Value:

None

Side Effects

See Note ** at the beginning of the API section.

SLCD_SetInvertState

Description:

Inverts all the segments of the LCD.

C Prototype:

```
void  SLCD_SetInvertState(BYTE InvertState)
```

Assembly:

```
mov   A, InvertState
lcall  SLCD_SetInvertState
```

Parameters:

InvertState - Specifies Normal State (normal display) or Inverted State (inverting display).

Symbolic Name	Value	Description
NORMAL_STATE	0	Normal display
INVERTED_STATE	1	Inverting display

Return Value:

None

Side Effects:

See Note ** at the beginning of the API section.

SLCD_EnableSymbol

Description:

Turns ON or OFF an individual segment.

C Prototype:

```
void  SLCD_EnableSymbol(BYTE SegmentID, BYTE ON_OFF)
```

Assembly:

```
mov   A, SegmentID
mov   X, ON_OFF
lcall  SLCD_EnableSymbol
```

Parameters:

SegmentID - Identifies the segment that needs to be controlled.

ON_OFF - if 1 is passed, it turns ON the segment, otherwise it turns it OFF.

The SLCD.inc and SLCD.h files have unique constant declarations for each SegmentID.

Return Value:

None

Side Effects:

See Note ** at the beginning of the API section.

Sample Firmware Source Code

This assembly code starts the user module and prints the number “1234” on the LCD display:

```
include "m8c.inc" ; part specific constants and macros
include "memory.inc" ; Constants & macros for SMM/LMM and Compiler
include "PSoCAPI.inc" ; PSoC API definitions for all user modules
export _main

_main:
    ; Enables global interrupt
    M8C_EnableGInt
    ; Strongly recommended to enable global interrupts
    ; before SLCD_Start() API to avoid the LCD damage!!!
    ; Start UM
    mov A, SLCD_NORMAL_STATE
    lcall _SLCD_Start
    ; set the contrast level to maximum
    mov A, SLCD_SET_TO_MAX
    lcall _SLCD_ChangeContrast
loop:
    mov A, 0x12
    push A
    mov A, 0x34
    push A
    mov A, SLCD_DISPLAY_ID_1
    push A
    ; prints the integer on display, specified by DisplayID
    lcall _SLCD_PrintHexInt
    jmp loop
```

The same code in C is:

```
//-------------------------------------------------------------------------
// C main line
//-------------------------------------------------------------------------
#include <m8c.h> // part specific constants and macros
#include "PSoCAPI.h" // PSoC API definitions for all user modules

void main(void)
{
    BYTE DisplayID;
    int Number;
    DisplayID = 0; /*init Display ID */
    Number = 0x1234; /*init Number */
    M8C_EnableGInt; /*enable global interrupt*/
    // Strongly recommended to enable global interrupts
    // before SLCD_Start() API to avoid the LCD damage!!!
    SLCD_Start(SLCD_NORMAL_STATE); /*starts the module*/
    /*sets the contrast level to maximum*/
    SLCD_ChangeContrast(SLCD_SET_TO_MAX, 0);
    while(1)
    {
        /*user code*/
        SLCD_PrintHexInt(DisplayID, Number);
        /* prints the integer on display, specified by DisplayID */
```

```
    }
}
```

Configuration Registers

The SLCD User Module uses two Timer digital PSoC blocks, one Analog CT block, and Analog Mux Bus resource. Each block is personalized and parameterized through a set of registers. The following tables give the "personality" values as constants and the parameters as named bitfields with brief descriptions. Symbolic names for these registers are defined in the user module instance's C and assembly language interface files (the ".h" and ".inc" files).

Only the parameterized symbols are explained in this section.

LCDTimer Block Registers

Table 13. Block LCDTimer, Register: Function, Bank 1

Block/Bit	7	6	5	4	3	2	1	0
MSB	0	0	1	0	0	0	0	0
LSB	0	0	0	0	0	0	0	0

This register defines the settings of the Digital Basic/Communications Type 'B' Block to be the LCDTimer digital block of the SLCD User Module.

Table 14. Block LCDTimer, Register: Input, Bank 1

Block/Bit	7	6	5	4	3	2	1	0
MSB	0	0	1	1	LCD Clock			
LSB	0	0	0	1	LCD Clock			

This register is used to select the clock input for the LCDTimer digital block of the SLCD User Module.

Table 15. Block LCDTimer, Register: Output, Bank 1

Block/Bit	7	6	5	4	3	2	1	0
MSB	0	1	0	0	0	0	0	0
LSB	0	1	0	0	0	0	0	0

This register is used to control the connection of the LCDTimer digital block outputs to the available row interconnect and control clock resynchronization.

Table 16. Block LCDTimer, Register: Count (DR0), Bank 0

Block/Bit	7	6	5	4	3	2	1	0
MSB	0	0	0	0	0	0	0	0
LSB	0	0	0	0	0	0	0	0

This register is the Data Register 0 for LCDTimer digital block of the SLCD User Module.

Table 17. Block LCDTimer, Register: Period (DR1), Bank 0

Block/Bit	7	6	5	4	3	2	1	0
MSB	Timer Period LSB							
LSB	Timer Period LSB							

This register is the Data Register 1 for LCDTimer digital block of the SLCD User Module.

Table 18. Block LCDTimer, Register: Compare (DR2), Bank 0

Block/Bit	7	6	5	4	3	2	1	0
MSB	0	0	0	0	0	0	0	0
LSB	0	0	0	0	0	0	1	0

This register is the Data Register 2 for LCDTimer digital block of the SLCD User Module.

Table 19. Block LCDTimer, Register: Control (CR0), Bank 0

Block/Bit	7	6	5	4	3	2	1	0
MSB	0	1	0	0	0	0	0	0
LSB	0	0	0	0	0	0	0	Start/Stop

This register is the Control Register 0 for the LCDTimer digital block of the SLCD User Module.

Table 20. Block LCDBias, Register: CR0 Bank 0

Bit	7	6	5	4	3	2	1	0
Value	0	0	0	0	0	0	0	1

Table 21. Block LCDBias, Register: CR1 Bank 0

Bit	7	6	5	4	3	2	1	0
Value	0	0	0	0	1	0	0	1

Table 22. Block LCDBias, Register: CR2 Bank 0

Bit	7	6	5	4	3	2	1	0
Value	0	0	0	1	0	1	0	0

Table 23. ANALOG_MUX_BUS, Register: MUX_CRx Bank 1

Bit	7	6	5	4	3	2	1	0
Value	Analog MUX Pin							

This register is the Analog Mux Bus Register of the SLCD User Module. This register is used to control the connection between the analog mux bus and the corresponding pin.

Version History

Version	Originator	Description
1.0	DHA	Initial version
1.10	DHA	Changed default value of MuxBusNumber wizard parameter for external bias configuration. Removed .LITERAL/.ENDLITERAL brackets from asm files.
1.10.b	DHA	Added wizard help button and file.
1.20	DHA	Added code to fix counter period issues for the CY8C20xx6 device.
2.00	DHA	1. Updated the SLCD User Module to support the new SetContrast API. 2. Added the SLCD_SetContrast API to this user module datasheet. 3. Updated this user module datasheet to describe the Analog MUX bus for CY8C20x34 and CY8C20xx6 devices.
2.10	DHA	1. Modified SetContrast API to add correction to 1 for negative difference between current and previous contrast. 2. Added CYRF89435 device support.
2.10.b	MYKZ	Added CY8C24x93 and CY7C69000 support.

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.