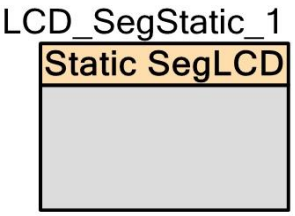
Static Segment LCD (LCD_SegStatic) 2.30

Features



- 1 to 61 pixels or symbols
- 10- to 150-Hz refresh rate
- User-defined pixel or symbol map with optional 7-segment, 14-segment, 16-segment, and bar graph calculation routines
- Direct drive static (one common) LCDs

General Description

The Static Segment LCD (LCD_SegStatic) component can directly drive 3.3-V and 5.0-V LCD glass. This component provides an easy method of configuring the PSoC device for your custom or standard glass.

Each LCD pixel/symbol may be either on or off. The Static Segment LCD component also provides advanced support to simplify the following types of display structures within the glass:

- 7-Segment numeral
- 14-Segment alphanumeric
- 16-Segment alphanumeric
- 1- to 255-element bar graph

When to Use a Static Segment LCD

Use the Static Segment Drive LCD component when you need to directly drive 3.3-V or 5.0-V LCD glass in a static mode (for glass configurations that have only one common line). The Static Segment LCD component does not require the target PSoC device to provide LCD drive hardware resources. The component can be used on any target chip that has sufficient pins to support the required number of common and segment lines.

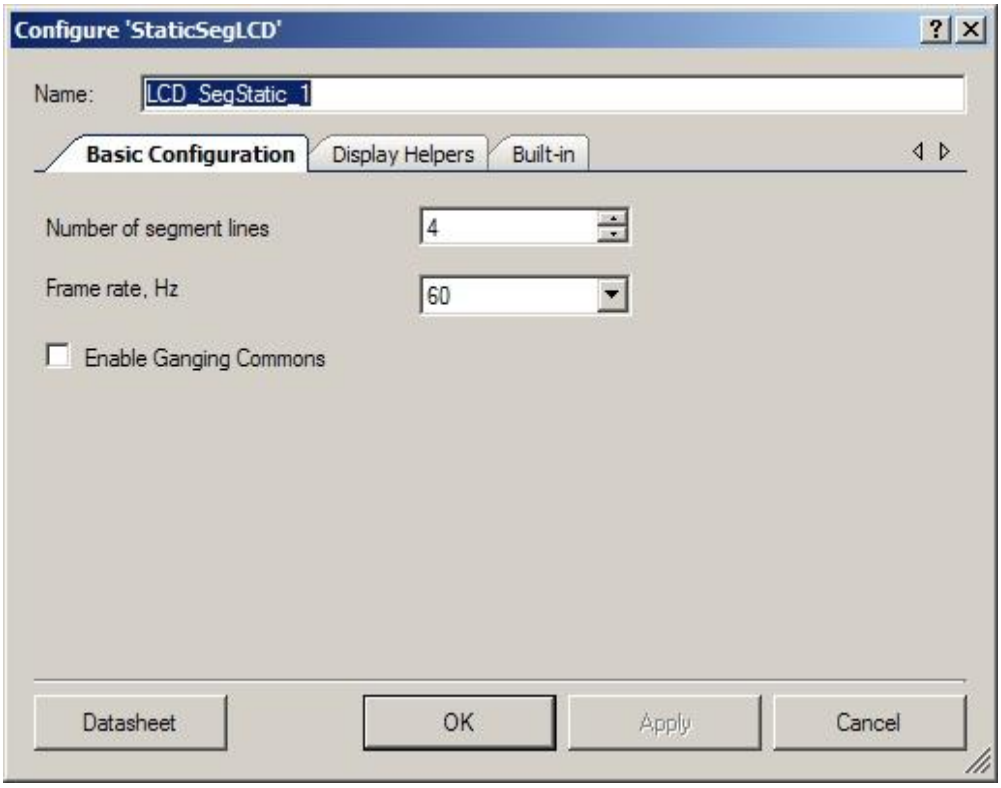
Input/Output Connections

There are no visible connections for the component on the schematic canvas; however, the various signals can be connected to pins using the Design-Wide Resources Pin Editor.

Component Parameters

Drag a Static Segment LCD component onto your design and double-click it to open the **Configure** dialog.

Basic Configuration Tab



Number of segment lines

Defines the number of segment lines in the user-defined display. The default is 4.

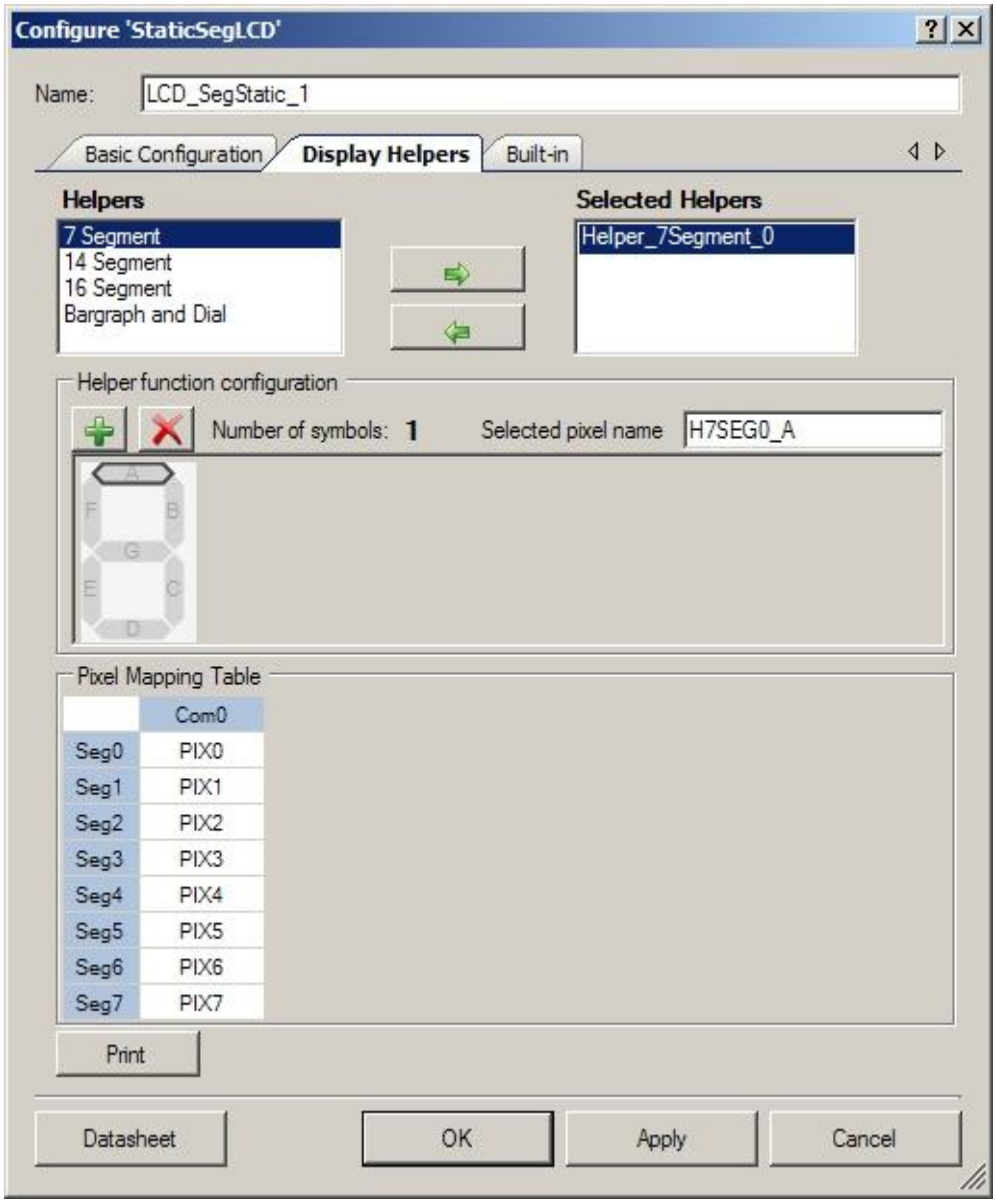
Frame rate, Hz

Determines the refresh rate of the display. Possible values are 10 Hz to 150 Hz. The default is 60 Hz.

Enable Ganging Commons

Select this check box to gang PSoC pins to drive common signal. If enabled then two PSoC pin will be allocated common signal. This is used to drive larger displays.

Display Helpers Tab



Display Helpers allow you to configure a group of display segments to be used together as one of several predefined display element types:

- 7-, 14-, or 16-segment displays
- Linear or circular bar graph display

The character-based display helpers can be used to combine multiple display symbols to create multicharacter display elements.

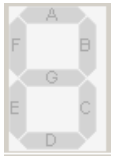
Helpers / Selected Helpers

You may add one or more helpers to the **Selected Helpers** list by selecting the desired helper type in the **Helpers** list and clicking the right-arrow button. If there are not enough pins to support the new helper, it will not be added. To delete a helper, select it in the **Selected Helpers** list and click the left-arrow button.

The order in which the selected helpers appear in the list is significant. By default, the first Helper of a given type added to the **Selected Helper** list is named with a 0 suffix, the next one of the same type will have a suffix of 1, and so on. If a selected helper is removed from the list, the remaining helpers are not renamed. When a helper is added, the name will use the lowest available suffix.

APIs are provided for each helper. Refer to the Application Programming Interface section for more information.

- **7 Segment Helper** – This helper may be one to five digits in length and can display either hexadecimal digits 0 to F or decimal 16-bit unsigned integer (uint16) values. A decimal point is not supported by the helper functions.



- **14 Segment Helper** – This helper may be up to 20 characters in length. It may display a single ASCII character or a null terminated string. Possible values are standard ASCII printable characters (with codes from 0 to 127).



- **16 Segment Helper** – This helper may be up to 20 characters in length. It may display a single ASCII character or a complete null terminated string. Possible values are standard ASCII characters and table of extended codes (with codes from 0 to 255). A table of extended codes is not supplied.



- **Bargraph and Dial Helper** – These helpers are used for bar graphs and dial indicators with 1 to 255 segments. The bar graph may be a single selected pixel or the selected pixel and all the pixels to the left or right of the specified pixel

Helper function configuration

This section of the dialog allows you to configure a helper; this includes adding or removing symbols to or from a helper as well as naming the pixels.

Select a helper from the **Selected Helpers** list.

Click the **[+]** or **[x]** button to add or remove a symbol for the selected helper. The maximum number of symbols you may add depends on the helper type and the total number of pixels supported by the component. If the number of available pins is not sufficient to support a new symbol, it will not be added.

To rename a pixel that is a part of a helper function, select the pixel on the symbol image in the **Helper function configuration** display. The current name displays in the selected pixel name field and it can be modified as desired.

Pixel Naming

The default pixel names have the form “PIX#”, where “#” is the number of the pixel in incremental order starting from the upper right corner of the **Pixel Mapping Table**.

The default naming for pixels associated with a helper symbol have a different format. The default name consists of a prefix portion, common to all of the pixels in a symbol, and a unique segment identifier. The default prefix indicates the helper type and the symbol instance. For example, the default name of a pixel in one of the symbols in a 7 Segment display helper might be “H7SEG4_A” where:

H7	Indicates the pixel is part of a 7 Segment helper
SEG4	Indicates the pixel is part of the symbol designated as the fourth 7-segment symbol in the project
A	Identifies the unique segment within the 7-segment symbol

For default pixel names, only the unique portion of the pixel name is shown on the symbol image. If you modify the pixel name, then the entire name is shown on the symbol image even if it has a prefix in common.

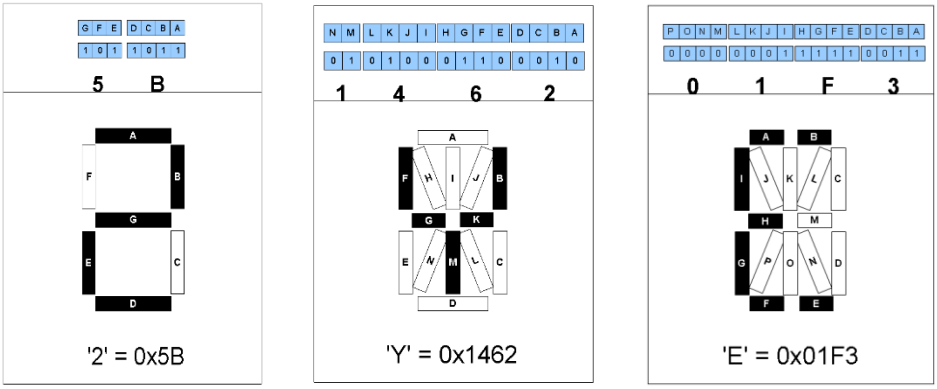
Note All pixel names must be unique.

When a helper function symbol element is assigned to a pixel in the **Pixel Mapping Table** (described below), the pixel assumes the name of the helper symbol element. The helper symbol element name supersedes the default pixel name, but does not replace it. You cannot re use the default pixel name of pixels that are associated with a helper function.

Character Encoding

All high-level helper APIs have their own lookup tables. The tables include a set of encoded pixel states, which construct a specific character reflection. The following examples show how the specific character can be encoded (segment names may be different than shown in the customizer).

7 Segment Encoding **14 Segment Encoding** **16 Segment Encoding**



Pixel Mapping Table

The **Pixel Mapping Table** is a representation of the frame buffer. For the API functions to work properly, each pixel from the **Helper function configuration** must be assigned to a pixel location in the **Pixel Mapping Table**. Refer to the datasheet for your LCD glass for the information you will need to make the correct assignments.

To assign pixels, select the desired pixel in the **Helper function configuration** panel and drag it to the correct location in the **Pixel Mapping Table**.

You can rename a pixel in the **Pixel Mapping Table** by double clicking on the pixel in the table display and entering the desired name. You can use this method to name a pixel that is not associated with one of the available helper types.

The **Print** button prints the pixel mapping table.

Clock Selection

The Static Segment LCD component uses one internal clock and does not require an external clock. Once the component is placed, the clock is automatically dedicated to the LCD component. The clock generates frame rate frequency.

Application Programming Interface

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

By default, PSoC Creator assigns the instance name “LCD_SegStatic_1” to the first instance of a component in a given design. You can rename the instance to any unique value that follows the syntactic rules for identifiers. The instance name becomes the prefix of every global function name, variable, and constant symbol. For readability, the instance name used in the following table is “LCD_SegStatic.”

Functions

Function	Description
LCD_SegStatic_Start()	Starts the LCD component and DMA channels. Initializes the frame buffer. Does not clear the frame buffer RAM if it was previously defined.
LCD_SegStatic_Stop()	Disables the LCD component and associated interrupts and DMA channels. Does not clear the frame buffer.
LCD_SegStatic_EnableInt()	Enables the LCD interrupts.
LCD_SegStatic_DisableInt()	Disables the LCD interrupt.
LCD_SegStatic_ClearDisplay()	Clears the display RAM of the frame buffer.
LCD_SegStatic_WritePixel()	Sets or clears a pixel based on PixelState. The pixel is addressed by a packed number.
LCD_SegStatic_ReadPixel()	Reads the state of a pixel in the frame buffer. The pixel is addressed by a packed number.
LCD_SegStatic_WriteInvertState()	Inverts the display based on an input parameter.
LCD_SegStatic_ReadInvertState()	Returns the current value of the display invert state: normal or inverted.
LCD_SegStatic_Sleep()	Stops the LCD and saves the user configuration.
LCD_SegStatic_Wakeup()	Restores and enables the user configuration.
LCD_SegStatic_Init()	Configures every-frame interrupt and initializes the frame buffer.
LCD_SegStatic_Enable()	Enables clock generation for the component.
LCD_SegStatic_SaveConfig()	Saves the LCD configuration.
LCD_SegStatic_RestoreConfig()	Restores the LCD configuration.

Note: Function names that contain a suffix “n” indicate that multiple display helpers of the same symbol type were created in the component customizer. Specific display helper elements are controlled by the API functions with the respective “n” index value in the function name.

uint8 LCD_SegStatic_Start(void)

Description: Starts the LCD component, DMA channels, frame buffer, and hardware. Does not clear the frame buffer RAM.

Parameters: None

Return Value: uint8 cystatus: Standard API return values.

Return Value	Description
CYRET_LOCKED	Some of TDs or channel already in use.
CYRET_SUCCESS	Function completed successfully

Side Effects: None

void LCD_SegStatic_Stop(void)

Description: Disables the LCD component and associated interrupts and DMA channels. Automatically blanks the display to avoid damage from DC offsets. Does not clear the frame buffer.

Parameters: None

Return Value: None

Side Effects: None

void LCD_SegStatic_EnableInt(void)

Description: Enables the LCD interrupts. An interrupt occurs after every LCD update (TD completion).

Parameters: None

Return Value: None

Side Effects: None

void LCD_SegStatic_DisableInt(void)

Description: Disables the LCD interrupts.

Parameters: None

Return Value: None

Side Effects: None

void LCD_SegStatic_ClearDisplay(void)

Description: This function clears the display RAM of the page buffer.

Parameters: None

Return Value: None

Side Effects: None

uint8 LCD_SegStatic_WritePixel(uint16 pixelNumber, uint8 pixelState)

Description: This function sets or clears a pixel in the frame buffer based on the PixelState parameter. The pixel is addressed with a packed number.

Parameters: uint16 pixelNumber: The packed number that points to the pixel's location in the frame buffer. The lowest three bits in the LSB low nibble are the bit position in the byte, the LSB upper nibble (four bits) is the byte address in the multiplex row and the MSB low nibble (four bits) is the multiplex row number.

uint8 pixelState: The PixelNumber specified is set to this pixel state. Symbolic names for the pixel state are provided, and their associated values are shown here:

Value	Description
LCD_SegStatic_PIXEL_STATE_OFF	Set the pixel to off.
LCD_SegStatic_PIXEL_STATE_ON	Set the pixel to on.
LCD_SegStatic_PIXEL_STATE_INVERT	Invert the pixel's current state.

Return Value: uint8 status: Pass or fail based on a range check of the byte address and multiplex row number. No check is performed on bit position.

Return Value	Description
CYRET_BAD_PARAM	Packed byte address or row value was invalid
CYRET_SUCCESS	Function completed successfully

Side Effects: None

uint8 LCD_SegStatic_ReadPixel(uint16 pixelNumber)

Description: This function reads a pixel's state in the frame buffer. The pixel is addressed by a packed number.

Parameters: uint16: pixelNumber: The packed number that points to the pixel's location in the frame buffer. The lowest three bits in the LSB low nibble are the bit position in the byte, the LSB upper nibble (four bits) is the byte address in the multiplex row and the MSB low nibble (four bits) is the multiplex row number.

Return Value: uint8 pixelState: Returns the current status of the PixelNumber specified.

Value	Description
0	The pixel is off.
1	The pixel is on.

Side Effects: None

uint8 LCD_Seg_WriteInvertState(uint8 invertState)

Description: This function inverts the display based on an input parameter.

Parameters: uint8 invertState: Sets the invert state of the display.

Value	Description
LCD_SegStatic_NORMAL_STATE	Normal noninverted display.
LCD_SegStatic_INVERTED_STATE	Inverted display.

Return Value: uint8 cystatus: Standard API return values.

Value	Description
CYRET_SUCCESS	Function completed successfully.
CYRET_BAD_PARAM	Evaluation of invertState parameter is failed.

Side Effects: None

uint8 LCD_Seg_ReadInvertState(void)

Description: This function returns the current value of the display invert state: normal or inverted.

Parameters: None

Return Value: (uint8) invertState: The invert state of the display.

Value	Description
LCD_SegStatic_NORMAL_STATE	Normal noninverted display.
LCD_SegStatic_INVERTED_STATE	Inverted display.

Side Effects: None

void LCD_SegStatic_Init(void)

Description: Initializes or restores the component according to the customizer Configure dialog settings. It is not necessary to call LCD_SegStatic_Init() because the LCD_SegStatic_Start() routine calls this function and is the preferred method to begin component operation. Configures every frame interrupt and initializes the frame buffer.

Parameters: None

Return Value: None

Side Effects: None

void LCD_SegStatic_Enable(void)

Description: Enables clock generation for the component.

Parameters: None

Return Value: None

Side Effects: None

void LCD_SegStatic_Sleep(void)

Description: This is the preferred routine to prepare the component for sleep. The LCD_SegStatic_Sleep() routine saves the current component state. Then it calls the LCD_SegStatic_Stop() function and calls LCD_SegStatic_SaveConfig() to save the hardware configuration.

Call the LCD_SegStatic_Sleep() function before calling the CyPmSleep() or the CyPmHibernate() function. Refer to the PSoC Creator *System Reference Guide* for more information about power management functions.

Parameters: None

Return Value: None

Side Effects: Don’t change component pin drive modes.

void LCD_SegStatic_Wakeup(void)

Description: This is the preferred routine to restore the component to the state when LCD_SegStatic_Sleep() was called. The LCD_SegStatic_Wakeup() function calls the LCD_SegStatic_RestoreConfig() function to restore the configuration. If the component was enabled before the LCD_SegStatic_Sleep() function was called, the LCD_SegStatic_Wakeup() function will also re-enable the component.

Parameters: None

Return Value: None

Side Effects: Calling the LCD_SegStatic_Wakeup() function without first calling the LCD_SegStatic_Sleep() or LCD_SegStatic_SaveConfig() function may produce unexpected behavior.

void LCD_SegStatic_SaveConfig(void)

Description: This function saves the component configuration and nonretention registers. It also saves the current component parameter values, as defined in the Configure dialog or as modified by appropriate APIs. This function is called by the LCD_SegStatic_Sleep() function.

Parameters: None

Return Value: None

Side Effects: None

void LCD_SegStatic_RestoreConfig(void)

Description: This function restores the component configuration and nonretention registers. It also restores the component parameter values to what they were before calling the LCD_SegStatic_Sleep() function.

Parameters: None

Return Value: None

Side Effects: Calling the LCD_SegStatic_RestoreConfig() function without first calling the LCD_SegStatic_Sleep() or LCD_SegStatic_SaveConfig() function may produce unexpected behavior.

Optional Helper APIs

The following APIs are present only when the respective helper has been selected in the Configure dialog.

Function	Description
LCD_SegStatic_Write7SegDigit_n	Displays a hexadecimal digit on an array of 7-segment display elements.
LCD_SegStatic_Write7SegNumber_n	Displays an integer value on a one- to five-digit array of 7-segment display elements.
LCD_SegStatic_WriteBargraph_n	Displays an integer location on a linear or circular bar graph
LCD_SegStatic_PutChar14Seg_n	Displays a character on an array of 14-segment alphanumeric character display elements.
LCD_SegStatic_WriteString14Seg_n	Displays a null terminated character string on an array of 14-segment alphanumeric character display elements.
LCD_SegStatic_PutChar16Seg_n	Displays a character on an array of 16-segment alphanumeric character display elements.
LCD_SegStatic_WriteString16Seg_n	Displays a null terminated character string on an array of 16-segment alphanumeric character display elements.

void LCD_SegStatic_Write7SegDigit_n(uint8 digit, uint8 position)

Description: This function displays a hexadecimal digit on an array of 7-segment display elements. Digits can be hexadecimal values in the range of 0 to 9 and A to F. The customizer Display Helpers facility must be used to define the pixel set associated with the 7-segment display elements Multiple 7-segment display elements can be defined in the frame buffer and are addressed through the suffix (n) in the function name. This function is only included if a 7-segment display element is defined in the component customizer.

Parameters: uint8 digit: Unsigned integer value in the range of 0 to 15 to be displayed as a hexadecimal digit.

uint8 position: Position of the digit as counted right to left starting at 0 on the right. If the position is outside the defined display area, the character is not displayed.

Return Value: None

Side Effects: None

void LCD_SegStatic Write7SegNumber_n(uint16 value, uint8 position, uint8 mode)

Description: This function displays a 16-bit integer value on a one- to five-digit array of 7-segment display elements. The customizer Display Helpers facility must be used to define the pixel set associated with the 7-segment display elements. Multiple 7-segment display element groups can be defined in the frame buffer and are addressed through the suffix (n) in the function name. Sign conversion, sign display, decimal points, and other custom features must be handled by application-specific user code. This function is only included if a 7-segment display element is defined in the component customizer.

Parameters: uint16 value: The unsigned integer value to be displayed.

uint8 position: The position of the least significant digit as counted right to left starting at 0 on the right. If the defined display area contains fewer digits then the Value requires, the most significant digit or digits will not be displayed

uint8 mode: Sets the display mode. Can be zero or one.

Value	Description
LCD_SegStatic_MODE_0	No leading 0s are displayed.
LCD_SegStatic_MODE_1	Leading 0s are displayed

Return Value: None

Side Effects: None

void LCD_SegStatic_WriteBargraph_n(uint16 location, int8 Mode)

Description: This function displays an 8-bit integer location on a 1- to 255-segment bar graph (numbered left to right).The bar graph may be any user-defined size between 1 and 255 segments. A bar graph may also be created in a circle to display rotary position. The customizer Display Helpers facility must be used to define the pixel set associated with the bar graph display elements Multiple bar graph displays can be created in the frame buffer and are addressed through the suffix (n) in the function name. This function is only included if a bar graph display element is defined in the component customizer

Parameters: uint16 location: The unsigned integer location to be displayed. Valid values are from zero to the number of segments in the bar graph. A zero value turns all bar graph elements off. Values greater than the number of segments in the bar graph result in all elements on.

int8 mode: Sets the bar graph display mode.

Value	Description
0	Specified location segment is turned on
1	The location segment and all segments to the left are turned on
–1	The location segment and all segments to the right are turned on.
2-10	Display the location segment and 2 to 10 segments to the right. This mode can be used to create wide indicators.

Return Value: None

Side Effects: None

void LCD_SegStatic_PutChar14Seg_n(uint8 character, uint8 position)

Description: This function displays an 8-bit character on an array of 14-segment alphanumeric character display elements. The customizer Display Helpers facility must be used to define the pixel set associated with the 14-segment display element. Multiple 14-segment alphanumeric display element groups can be defined in the frame buffer and are addressed through the suffix (n) in the function name. This function is only included if a 14-segment element is defined in the component customizer.

Parameters: uint8 character: The ASCII value of the character to display (printable characters with ASCII values 0 to 127)

uint8 position: The position of the character as counted left to right starting at 0 on the left. If the position is outside the defined display area, the character is not displayed.

Return Value: None

Side Effects: None

void LCD_SegStatic_WriteString14Seg_n(uint8 const character[], uint8 position)

Description: This function displays a null terminated character string on an array of 14-segment alphanumeric character display elements. The customizer Display Helpers facility must be used to define the pixel set associated with the 14-segment display elements Multiple 14-segment alphanumeric display element groups can be defined in the frame buffer and are addressed through the suffix (n) in the function name. This function is only included if a 14-segment display element is defined in the component customizer

Parameters: uint8 const character[]: The pointer to the null terminated character string.

uint8 position: The position of the first character as counted left to right starting at 0 on the left. If the length of the string exceeds the size of the defined display area, the extra characters are not displayed.

Return Value: None

Side Effects: Doesn't clear display prior data output. All the locations that weren't affected will remain their previous pixel states.

void LCD_SegStatic_PutChar16Seg_n(uint8 character, uint8 position)

Description: This function displays an 8-bit character on an array of 16-segment alphanumeric character display elements. The customizer Display Helpers facility must be used to define the pixel set associated with the 16-segment display elements. Multiple 16-segment alphanumeric display element groups can be defined in the frame buffer and are addressed through the suffix (n) in the function name. This function is only included if a 16-segment display element is defined in the component customizer

Parameters: uint8 character: The ASCII value of the character to display (printable ASCII and table extended characters with values 0 to 255)

uint8 position: The position of the character as counted left to right starting at 0 on the left. If the position is outside the defined display area, the character is not displayed.

Return Value: None

Side Effects: None

(void) LCD_SegStatic_WriteString16Seg_n(uint8 const character[], uint8 position)

Description: This function displays a null terminated character string on an array of 16-segment alphanumeric character display elements. The customizer Display Helpers facility must be used to define the pixel set associated with the 16-segment display elements. Multiple 16-segment alphanumeric display element groups can be defined in the frame buffer and are addressed through the suffix (n) in the function name. This function is only included if a 16-segment display element is defined in the component customizer.

Parameters: uint8 const character[]: The pointer to the null terminated character string.

uint8 position: The position of the first character as counted left to right starting at 0 on the left. If the length of the string exceeds the size of the defined display area, the extra characters are not displayed.

Return Value: None

Side Effects: Doesn't clear display prior data output. All the locations that weren't affected will remain their previous pixel states.

Pins APIs

These API functions are used to change the drive mode of pins used by the Static Segment LCD component. These APIs are most often used to place Static Segment LCD component pins into the HI-Z mode to minimize leakage while the device is in a low-power mode.

Function	Description
LCD_SegStatic_ComPort_SetDriveMode	Sets the drive mode for the pin used by a common line of the Static Segment LCD component.
LCD_SegStatic_SegPort_SetDriveMode	Sets the drive mode for all pins used by segment lines of the Static Segment LCD component.

void LCD_SegStatic_ComPort_SetDriveMode(uint8 mode)

Description: Sets the drive mode for the pin used by a common line of the Static Segment LCD component.

Parameters: uint8 mode: The desired drive mode. See the Pins component datasheet for information on drive modes.

Return Value: None

Side Effects: None

LCD_SegStatic_SegPort_SetDriveMode(uint8 mode)

Description: Sets the drive mode for all pins used by segment lines of the Static Segment LCD component.

Parameters: uint8 mode: The desired drive mode. See the Pins component datasheet for information on drive modes.

Return Value: None

Side Effects: None

Global Variables

Variable	Description
LCD_SegStatic_initVar	Indicates whether the LCD_SegStatic has been initialized. The variable is initialized to 0 and set to 1 the first time LCD_SegStatic_Start() is called. This allows the component to restart without reinitialization after the first call to the LCD_SegStatic_Start() routine. If reinitialization of the component is required, then the LCD_SegStatic_Init() function can be called before the LCD_SegStatic_Start() or LCD_SegStatic_Enable() function.

Defines

LCD_SegStatic_SEG_NUM

This macro defines the number of segment lines for the user-defined display current configuration of the component.

LCD_SegStatic_FRAME_RATE

This macro defines the refresh rate for the user-defined display current configuration of the component.

LCD_SegStatic_WRITE_PIXEL

This is a macro define of the LCD_SegStatic_WritePixel() function that returns void.

LCD_SegStatic_READ_PIXEL

This is a macro define of the LCD_SegStatic_ReadPixel() function.

LCD_SegStatic_FIND_PIXEL

This macro calculates pixel location in the frame buffer. It uses information from the customizer pixel table and information about physical pins dedicated for the LCD. This macro is the basis of the pixel mapping mechanism. Every pixel name from the pixel table is defined with the calculated pixel location in the frame buffer. APIs use pixel names to access the respective pixel.

Macro Callbacks

Macro callbacks allow users to execute code from the API files that are automatically generated by PSoC Creator. Refer to the PSoC Creator Help and *Component Author Guide* for the more details.

In order to add code to the macro callback present in the component's generated source files, perform the following:

- Define a macro to signal the presence of a callback (in *cyapicallbacks.h*). This will "uncomment" the function call from the component's source code.
- Write the function declaration (in *cyapicallbacks.h*). This will make this function visible by all the project files.
- Write the function implementation (in any user file).

Callback Function [1]	Associated Macro	Description
LCD_SegStatic_ISR_Interrupt Callback	LCD_SegStatic_ISR_INTERRUPT_CALLBACK	Used at the beginning of the LCD_SegStatic_ISR() interrupt handler to perform additional application-specific actions.

Sample Firmware Source Code

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

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

1 The callback function name is formed by component function name optionally appended by short explanation and "Callback" suffix.

MISRA Compliance

This section describes the MISRA-C:2004 compliance and deviations for the component. There are two types of deviations defined:

- project deviations – deviations that are applicable for all PSoC Creator components
- specific deviations – deviations that are applicable only for this component

This section provides information on component-specific deviations. Project deviations are described in the MISRA Compliance section of the *System Reference Guide* along with information on the MISRA compliance verification environment.

The Static Segment LCD component has not been verified for MISRA-C:2004 coding guidelines compliance.

Functional Description

The Static Segment LCD component provides a powerful and flexible mechanism for driving different types of LCD glass. The configuration dialog provides access to the parameters that can be used to customize the component. A standard set of API routines provide control of the display and of specific pixels. Additional display APIs are generated based on the type and number of display helpers defined.

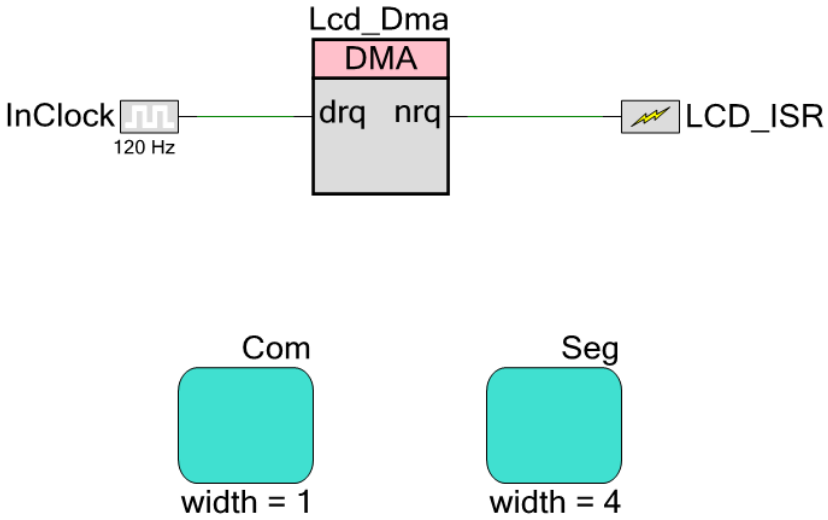
Default Configuration

The default configuration of the LCD_SegStatic component provides a generic Direct LCD Segment drive controller. The default LCD_SegStatic configuration is:

- Four segment lines
- 30-Hz refresh rate
- No display helpers are defined. Default API generation will not include functions for any of the supported display elements.

Block Diagram and Configuration

The following diagram shows the schematic representation of the Static Segment LCD component. The component does not require special-purpose LCD drive hardware. The component makes use of the DMA and standard digital port I/Os.



This diagram shows the internal schematic for the Static Segment LCD component. It consists of a DMA component, ISR component, clock component, and two LCD ports.

- The DMA component is used to transfer data from the frame buffer to the LCD data registers through the aliased memory area.
- The LCD Ports (Com and Seg) are used to map the logical signals to physical pins. There are two instances of the LCD Port; one for common lines and one for segment lines.
- ISR component available for user interrupt.

Top-Level Architecture

The architecture of the component is very simple. It only uses a few blocks. The base of the component is a frame buffer that contains LCD data. The frame buffer can be modified using the LCD_SegStatic_WritePixel() function or a high-level function that refers to this function. The DMA then transfers the LCD data to port registers using aliased memory. The DMA transactions are triggered by an internal clock set with a precalculated value to provide a proper refresh rate for the LCD.

Resources

The Static Segment LCD component uses one DMA component and one ISR. Also component utilizes an I/O pin for common and one pin for each segment line.

API Memory Usage

The component memory usage varies significantly, depending on the compiler, device, number of APIs used and component configuration. The following table provides the memory usage for all APIs available in the given component configuration.
The measurements have been done with the associated compiler configured in Release mode with optimization set for Size. For a specific design the map file generated by the compiler can be analyzed to determine the memory usage.

Basic: Low-level API functions set without any high-level helper API

Basic, 7-segment helper: Low-level API functions set + 7-segment helper API

Basic, 14-segment helper: Low-level API functions set + 14-segment helper API

Basic, 16-segment helper: Low-level API functions set + 16-segment helper API

Basic, Bar Graph helper: Low-level API functions set + bar graph helper high-level API

Configuration	PSoC 3 (Keil_PK51)		PSoC 5LP (GCC)	
	Flash Bytes	SRAM Bytes	Flash Bytes	SRAM Bytes
Basic	1652	40	750	45
Basic, 7-segment helper	1963	40	884	45
Basic, 14-segment helper	2131	40	1118	45
Basic, 16-segment helper	2135	40	1122	45
Basic, Bar Graph helper	2195	40	986	45

DC and AC Electrical Characteristics

N/A

Component Errata

This section lists known problems with the component.

Cypress ID	Component Version	Problem	Workaround
191257	v2.30	This component was modified without a version number change in PSoC Creator 3.0 SP1. For further information, see Knowledge Base Article KBA94159 (www.cypress.com/go/kba94159).	No workaround is necessary. There is no impact to designs.

Component Changes

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

Version	Description of Changes	Reason for Changes / Impact
2.30.d	Minor datasheet edits.	
2.30.c	Updated Description for LCD_SegStatic_EnableInt() and LCD_SegStatic_DisableInt() APIs.	Removed interdependence with LCD_SegStatic_Start() and LCD_SegStatic_Stop() APIs.
2.30.b	Datasheet update.	Added Macro Callbacks section.
	Enabled component to be nested into other components.	Support for hierarchical component design.
2.30.a	Edited datasheet to add Component Errata section.	Document that the component was changed, but there is no impact to designs.
2.30	The component was verified for MISRA compliance.	Updated MISRA Compliance section.
	LCD_SegStatic_dispN[][] declaration moved inside functions where it is used.	MISRA-related change. An instance of LCD_SegStatic_dispN[][] (N=0, ...,7) is declared for each helper in the component and that instance is used in only one function of specific helper.
	Constant LCD_SegStatic digitNum_n (n=0, ...,7) was replaced with #define LCD_SegStatic_DIGIT_NUM_n.	MISRA-related change. As the value of LCD_SegStatic_digitNum_n didn't change in the code it is preferable for it to declare it as #define.
	Following variables were made to be “static”: • LCD_SegStatic_7SegDigits[]; • LCD_SegStatic_14SegChars[]; • LCD_SegStatic_16SegChars[].	MISRA-related change. These variables are intended to be used only by the helper API functions and therefore should be declared as “static”.
	Declaration of parameter “character” in LCD_Seg_WriteString14Seg_n() and LCD_Seg_WriteString16Seg_n() API was changed from “uint8 * character” to “uint8 const character[]”.	MISRA-related change. MISRA doesn’t allow using array indexing of variables when they are declared as pointers.
	Fixed parameter type for pixelNumber parameter from uint8 to uint16 in LCD_SegStatic_WritePixel() and LCD_SegStatic_ReadPixel().	The implementation of those functions didn’t to match the datasheet.
	Fixed incorrect description of Mode parameter of LCD_SegStatic_WriteBargraph_n().	The datasheet specified the type of Mode parameter as "uint8" instead of "int8".
	The default value of Frame Rate, on page 2, was changed from 30 to 60 Hz to match implementation.	

	Fixed description of return value of LCD_SegStatic_Start() function.	The description specified CYRET_BAD_PARAM instead of CYRET_LOCKED.
	PSoC 5 support was removed.	
2.20	Added MISRA Compliance section.	The component was not verified for MISRA compliance.
	Removed a note that states it is impossible to modify number of common lines after helper function was added.	Component now allows modifying number of commons when helper is added.
	Updated API Memory Usage table.	
	Fixed issue in description of LCD_SegStatic_Write7SegNumber_n (). Parameter “Value” is of type uint16 but datasheet stated that it is uint8.	
	Fixed issue in description of LCD_SegStatic_WriteBargraph_n(). Parameter “Location” is of type uint16 but datasheet stated that it is uint8.	
	Fixed issue with LCD_SegStatic_WriteStringDotMatrix_n() and Seg_Display_WriteString16Seg_n() API.	Previous implementation expected a null pointer to terminate the string instead of a single zero byte. That could cause an incorrect string to be displayed.
	Fixed issue with LCD_SegStatic_WriteBargraph_n() API.	Previously this API didn't clear its previous output on display. This could cause unexpected results to be displayed on the LCD.
	Fixed issue with LCD_SegStatic_Write7SegNumber_n() API.	Previously this API didn't clear its previous output on display. This could cause unexpected results to be displayed on the LCD.
2.10	Changed default instance name from LCD_SegStat to LCD_SegStatic.	
	Added characterization data to datasheet	
	Added two new APIs: LCD_SegStatic_ReadInvertState(); LCD_SegStatic_WriteInvertState();	LCD_SegStatic_WriteInvertState() allows to invert the entire LCD glass. This feature can be used for testing purposes. LCD_SegStatic_ReadInvertState() used for reading thr sate of a display if it in normal or inverted state.
	Fixed LCD_SegStatic_Write7SegNumber() API.	There was a bug in the API wich overrrided the last digit of the number displated on the LCD with 0. The issue only occured when "leading zeroes" mode of output was used.
	Added all component APIs with the CYREENTRANT keyword when they are included in the .cyre file.	Add the capability for customers to specify any individual generated functions as reentrant.

	Enable ganging commons checkbox added to the component customizer.	This a new feature that is helpful for driving large displays.
2.0	Low-power APIs support was removed for PSoC 5 silicon.	The following APIs were added to conditional compilation: LCD_SegStat_SaveConfig(); LCD_SegStat_RestoreConfig(); LCD_SegStat_Sleep(); LCD_SegStat_Wakeup(). These APIs are not included to the project if it runs on a PSoC 5 device.
	Fixed LCD_SegStat_Stop() API issue.	The LCD_SegStat_Stop() function wasn't releasing the DMA configuration. That caused the component to stop functioning after several LCD_SegStat_Start() -> LCD_SegStat_Stop() sequences. This was happening because the component ran out of DMA resources that were allocated in the LCD_SegStat_Start() function and not released in the LCD_SegStat_Stop() API. Proper DMA release procedure was implemented and used in the LCD_SegStat_Stop() API.
	Changed WRITE_PIXEL() macros definition from #define LCD_SegStat_WRITE_PIXEL(pixelNumber, pixelState) LCD_SegStat_WritePixel(pixelNumber, pixelState) to #define LCD_SegStat_WRITE_PIXEL(pixelNumber, pixelState) (void) LCD_SegStat_WritePixel(pixelNumber, pixelState)	There is no need to analyze the return value of the LCD_SegStat_WritePixel() API inside component realization, so the macros were updated to handle that.
	Deleted following obsolete names from header file: LCD_SegStat_Buffer LCD_SegStat_Channel LCD_SegStat_TermOut LCD_SegStat_TD LCD_SegStat_GCommons LCD_SegStat_Commons	These names were marked as obsolete in the previous component version.
	Function LCD_SegStat_WriteBargraph() was optimized.	Code refactoring allowed the component to save some space in flash memory for this API.
	Functions LCD_SegStat_WriteString14Seg() and LCD_SegStat_WriteString16Seg() were optimized.	Array indexing was changed to pointer indexing, which allowed the component to save some RAM space and get better execution time from these functions.

	Fixed multiple variable declarations in one line all over the source code.	This was violating internal code standards specification.
	LCD_SegStat_Write7SegNumber() API implementation was optimized.	There was some ineffective code in the LCD_SegStat_Write7SegNumber() API that was reflected in longer execution time of this API.
	Fixed LCD_SegStat_Wakeup() API.	The LCD_SegStat_Wakeup() API always returned CYRET_LOCKED even if the function executed correctly. Implemented procedure for proper return status handling.
1.50.a	Added manual call workaround to LCD_SegStat_Stop() description in datasheet	Missing call to LCD_SegStat_DmaDispose() in the LCD_SegStat_Stop() routine.
	Added Pin APIs section to datasheet.	
	Minor datasheet edits and updates.	
1.50	Added sleep and wakeup APIs.	To support low power modes.
	Added backup structure which stores the state of the component.	Backup structure stores component state (enabled/disabled) prior to entering sleep mode. When LCD_SegStat_Wakeup() API called to wake the component from sleep this structure puts the component back into the state it was before entering the Sleep mode.
	Added tooltip in the Helpers tab.	This was done because the default size of the Configure dialog does not show the Pixel Mapping Table.
	Added numeric up-down controls and edit fields for Number of Segment Lines set enabled even if user added helper functions, earlier they were disabled.	Previously, if you wanted to change the number of segment lines after adding helpers, you had to delete all helpers and lose all configuration information. After adding a helper to a segment LCD component, you can go back and change the number of segment lines. If the number of segments increases, everything is clear. If the number of segments decreases, you will see a warning that you may lose configuration information. If you have assigned pixels on segment lines that you want to remove, they will be unassigned.
	Added error provider for selected pixel name text box. If you enter a wrong value in the text box, an error icon appears and you cannot leave the text box until the value is correct. Error icon has a tool tip with a problem description.	Error provider is a better way to handle incorrect values. It is used instead of showing message boxes.
	Following global variables were redefined with low camel case names: #define `$LCD_SegStat`_Buffer #define `$LCD_SegStat`_Channel #define `$LCD_SegStat`_TermOut #define `$LCD_SegStat`_TD #define `$$LCD_SegStat`_GCommons	Coding standards require variables to follow low camel case naming style. These variables became obsolete and will be removed in future.

	#define `$$LCD_SegStat`_Commons	
1.30.b	Added information to the component that advertizes its compatibility with silicon revisions.	The tool reports an error/warning if the component is used on incompatible silicon. If this happens, update to a revision that supports your target device.
	Made change log cumulative for v1.30 datasheet	
1.30.a	Moved local parameters to formal parameter list.	To address a defect that existed in PSoC Creator v1.0 Beta 4.1 and earlier, the component was updated so that it could continue to be used in newer versions of the tool. This component used local parameters, which are not exposed to the user, to do background calculations on user input. These parameters have been changed to formal parameters which are visible, but un-editable. There are no functional changes to the component but the affected parameters are now visible in the “expression view” of the customizer dialog.
1.30	Updated generated source code.	Added StaticSegLCD_INT.c file to component's library. Component's source code was changed to use CyLib.h function for internal interrupt configuration. Added a fix for DMA to work properly in 32-bit address space. Added an equation to component source file which selects proper termination signal from DMA.

© Cypress Semiconductor Corporation, 2013-2016. This document is the property of Cypress Semiconductor Corporation and its subsidiaries, including Spansion LLC (“Cypress”). This document, including any software or firmware included or referenced in this document (“Software”), is owned by Cypress under the intellectual property laws and treaties of the United States and other countries worldwide. Cypress reserves all rights under such laws and treaties and does not, except as specifically stated in this paragraph, grant any license under its patents, copyrights, trademarks, or other intellectual property rights. If the Software is not accompanied by a license agreement and you do not otherwise have a written agreement with Cypress governing the use of the Software, then Cypress hereby grants you a personal, non-exclusive, nontransferable license (without the right to sublicense) (1) under its copyright rights in the Software (a) for Software provided in source code form, to modify and reproduce the Software solely for use with Cypress hardware products, only internally within your organization, and (b) to distribute the Software in binary code form externally to end users (either directly or indirectly through resellers and distributors), solely for use on Cypress hardware product units, and (2) under those claims of Cypress’s patents that are infringed by the Software (as provided by Cypress, unmodified) to make, use, distribute, and import the Software solely for use with Cypress hardware products. Any other use, reproduction, modification, translation, or compilation of the Software is prohibited.

TO THE EXTENT PERMITTED BY APPLICABLE LAW, CYPRESS MAKES NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS DOCUMENT OR ANY SOFTWARE OR ACCOMPANYING HARDWARE, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. To the extent permitted by applicable law, Cypress reserves the right to make changes to this document without further notice. Cypress does not assume any liability arising out of the application or use of any product or circuit described in this document. Any information provided in this document, including any sample design information or programming code, is provided only for reference purposes. It is the responsibility of the user of this document to properly design, program, and test the functionality and safety of any application made of this information and any resulting product. Cypress products are not designed, intended, or authorized for use as critical components in systems designed or intended for the operation of weapons, weapons systems, nuclear installations, life-support devices or systems, other medical devices or systems (including resuscitation equipment and surgical implants), pollution control or hazardous substances management, or other uses where the failure of the device or system could cause personal injury, death, or property damage (“Unintended Uses”). A critical component is any component of a device or system whose failure to perform can be reasonably expected to cause the failure of the device or system, or to affect its safety or effectiveness. Cypress is not liable, in whole or in part, and you shall and hereby do release Cypress from any claim, damage, or other liability arising from or related to all Unintended Uses of Cypress products. You shall indemnify and hold Cypress harmless from and against all claims, costs, damages, and other liabilities, including claims for personal injury or death, arising from or related to any Unintended Uses of Cypress products.

Cypress, the Cypress logo, Spansion, the Spansion logo, and combinations thereof, WICED, PSoC, CapSense, EZ-USB, F-RAM, and Traveo are trademarks or registered trademarks of Cypress in the United States and other countries. For a more complete list of Cypress trademarks, visit cypress.com. Other names and brands may be claimed as property of their respective owners.