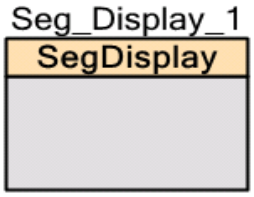
Segment Display (Seg_Display) 1.10



Features

- Available for PSoC 5 devices only (For PSoC 3 and PSoC 5LP devices, use the Segment LCD version 3.10 component.)
- 2 to 768 pixels or symbols
- 1/3, 1/4, and 1/5 bias supported
- 10- to 150-Hz refresh rate
- Integrated bias generation between 2.0 V and 5.2 V with up to 128 digitally controlled bias levels for dynamic contrast control
- Supports both type A (standard) and type B (low power) waveforms
- Pixel state of the display may be inverted for negative image
- 256 bytes of display memory (frame buffer)
- User-defined pixel or symbol map with optional 7-, 14-, or 16-segment character; 5x7 or 5x8 dot matrix; and bar graph calculation routines

General Description

The Segment Display (Seg_Display) component can directly drive 3.3-V and 5.0-V LCD glass at multiplex ratios up to 16x. This component provides an easy method of configuring the PSoC device to drive your custom or standard glass.

Internal bias generation eliminates the need for any external hardware and allows for software based contrast adjustment. Using the Boost Converter, the glass bias may be at a higher voltage than the PSoC supply voltage. This allows increased display flexibility in portable applications.

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

- 7-segment numerals
- 14-segment alphanumeric
- 16-segment alphanumeric

- 5x7 and 5x8 dot matrix alphanumeric (Use the same look-up table on the 5x7 and 5x8. All symbols in the look-up table are the size of 5x7 pixels.)
- 1 to 255 element bar graphs

For more information about using the Segment Display component, refer to application note *AN52927: PSoC® 3: Segment LCD Direct Drive Basics*.

When to Use a Segment Display

Use the Direct Segment Drive LCD component when you need to directly drive 3.3-V or 5.0- V LCD glass at multiplex ratios from 2x to 16x. The Direct Segment Drive LCD component requires that the target PSoC device supports LCD direct drive.

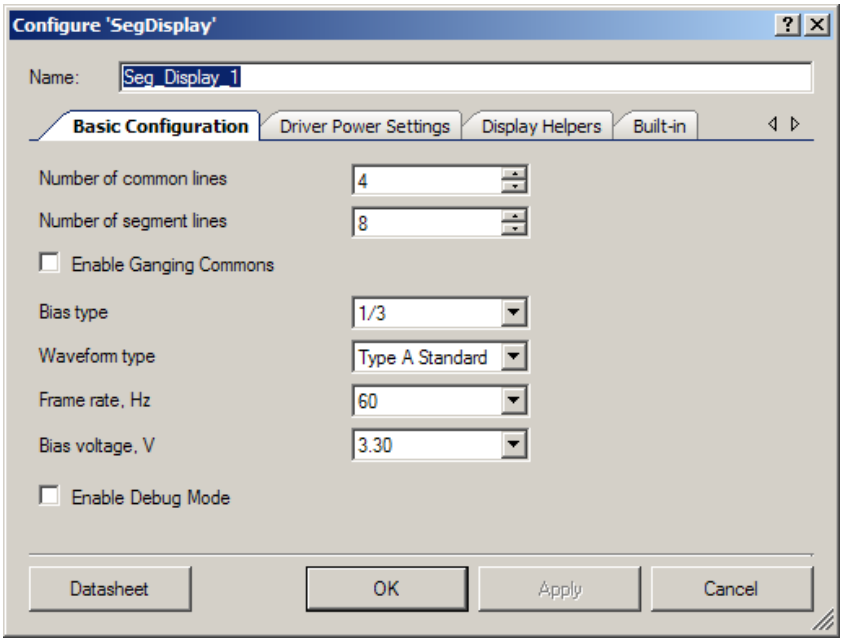
Input/Output Connections

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 Segment Display component onto your design and double click it to open the **Configure** dialog. The **Configure** dialog contains several tabs with different types of parameters to set up the Segment Display component.

Basic Configuration Tab



Number of common lines

Defines the number of common signals required by the display (default is **4**).

Number of segment lines

Defines the number of segment signals required by the display. The range of possible values is from (2 to 62) – **Number of common lines**. The default is **8**.

Enable Ganging Commons

Select this checkbox to gang PSoC pins to drive common signals. Two PSoC pins are allocated for each common signal. This is used to drive larger displays.

Bias type

This value determines the proper bias mode for the set of common and segment lines.

Waveform type

This determines the waveform type: Type A - 0 VDC average over a single frame (default) or Type B - 0 VDC average over two frames.

Frame Rate

This determines the refresh rate of the display. The range of possible values is from 10 Hz to 150 Hz. The default is **60 Hz**.

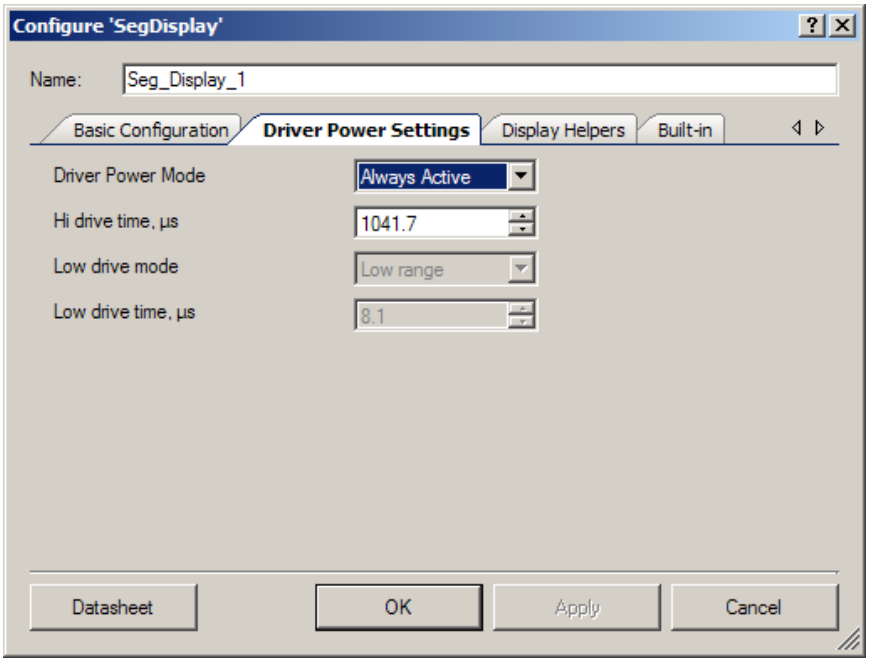
Bias Voltage

This determines the bias voltage level for the LCD DAC. The range of possible values is from 2 V to 5.2 V. The default is **3.3 V**.

Enable Debug Mode

If enabled, a Debug Port is added to the component, which allows you to look at the signals that drive the LCD Drivers. It is useful for seeing the high and low drive times, for example.

Driver Power Settings Tab



Driver Power Mode

The **Driver Power Mode** parameter defines the power mode of the component. Two power mode settings are available:

- **Always Active**: LCD DAC is always turned on
- **Low Power**: LCD DAC is turned off between voltage transitions

Refer to Driver Power Modes in the Functional Description section later in this datasheet.

High drive time

This parameter defines the time during which high drive mode is active within one voltage transaction.

The calculation of **High drive time** minimum, maximum, and default values are defined later in this datasheet.

Low drive mode

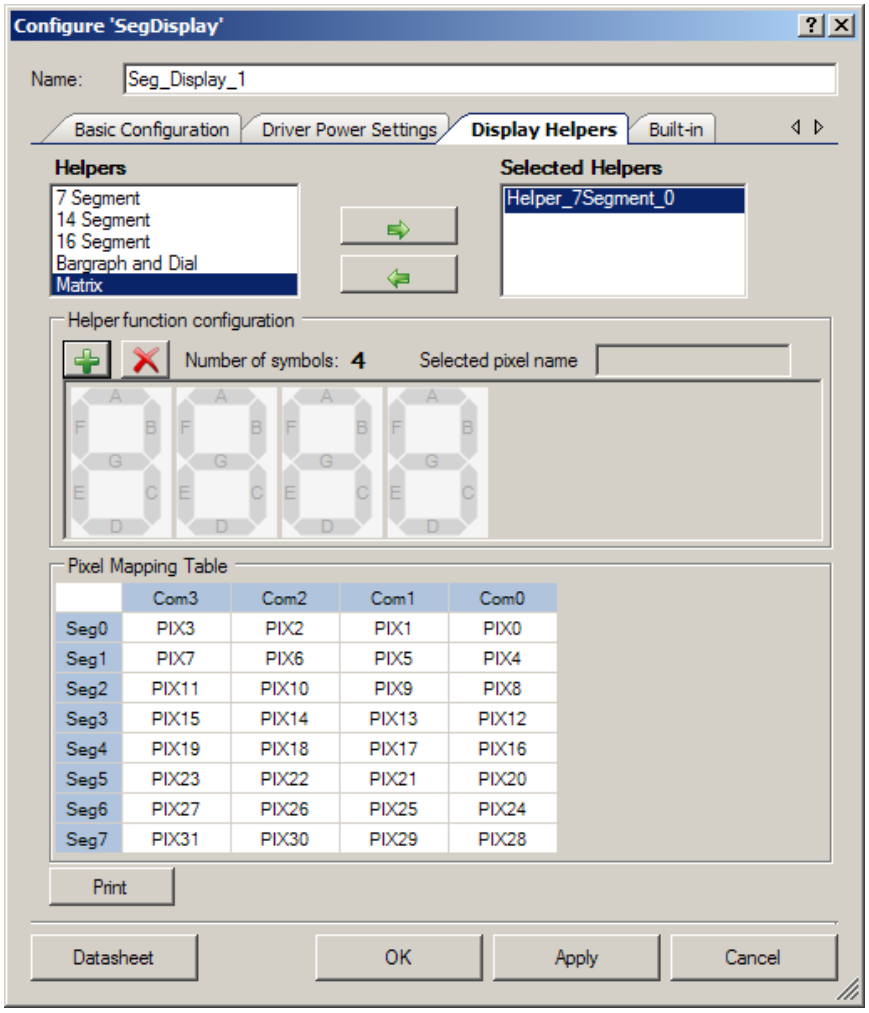
This parameter is available when you set **Driver Power Mode** to **Low Power**. Two **Low drive mode** values are available:

- **Low range** – activates low drive mode
- **High range** – activates second low drive high current mode (Lo2).

Low drive time

The **Low drive time** parameter defines the time during which low drive mode is active within the voltage transaction.

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
- Dot-matrix display (5x7 or 5x8)
- 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

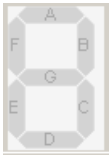
You may add one or more helpers to the **Selected Helpers** list by selecting the desired helper type in the **Helpers** list and clicking the right-arrow button. If there are not enough pins to support the new helper, it will not be added. To delete a helper, select it in the **Selected Helpers** list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

Note Once you have added a display helper to the component, you will not be able to change the number of common or segment lines. It is important to set the number of common and segment lines for the component before defining any display helpers. Any defined display helpers must be removed before you can change the number of common or segment lines.

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 be 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.



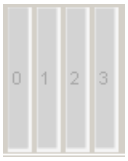
- **14 Segment Helper** – This helper may be up to 20 characters in length. It may display a single ASCII character or a null terminated string. Possible values are standard ASCII printable characters (with codes from 0 to 127).



- **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



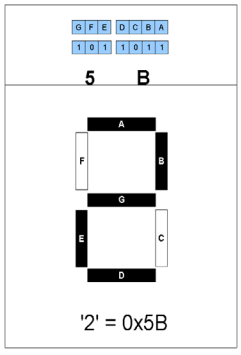
- **Matrix Helper** – This helper supports up to eight character elements. The component supports 5x7 or 5x8 row/column characters. Longer strings of characters can be created by configuring two or more dot matrix helpers to define adjacent dot matrix sections of the display. The helper displays a single ASCII character or a null terminated string.

The dot matrix helper has pinout constraints. It must use 7 or 8 sequential common drivers for the matrix rows and 5 to 40 sequential segment drivers for the matrix columns. The component supports the standard Hitachi HD44780 character set.

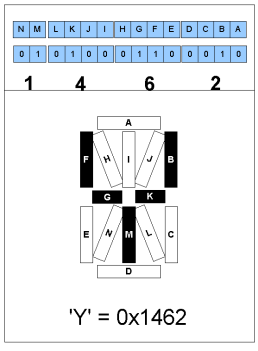
Character Encoding

Each high-level helper API has its own lookup table. 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).

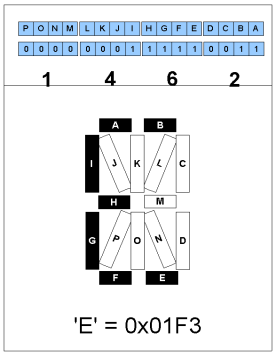
7 Segment Encoding



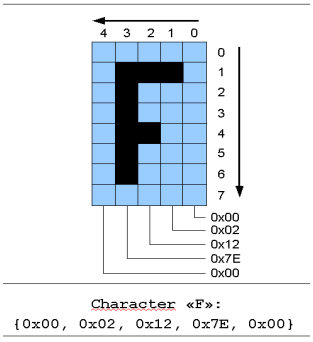
14 Segment Encoding



16 Segment Encoding



Dot Matrix Encoding



Helper function configuration

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

1. Select a helper from the **Selected Helpers** list.
2. 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 enough to support a new symbol, it will not be added.
3. 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 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 right upper corner of **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 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 will be shown on the symbol image even if it shares a common prefix with other pixels.

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 reuse the default pixel name of pixels that are associated with a helper function.

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 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 Seg_Display component uses two internal clocks and does not require an external clock. After the component is placed, the two clocks are automatically dedicated to the LCD component. The first clock generates the refresh rate frequency and the second generates a 100-kHz clock for the low-drive buffers.

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 “Seg_Display_1” to the first instance of a component in a given design. You can rename the instance to any unique value that follows the PSoC Creator syntax 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 “Seg_Display.”

Function	Description
Seg_Display_Start()	Starts the LCD component and enables the required interrupts.
Seg_Display_Stop()	Disables the LCD component and associated interrupts and DMA channels.
Seg_Display_EnableInt()	Enables the LCD interrupts. Not required if Seg_Display_Start() is called
Seg_Display_DisableInt()	Disables the LCD interrupt. Not required if Seg_Display_Stop() is called
Seg_Display_SetBias()	Sets the bias level for the LCD glass to one of 128 values.
Seg_Display_WriteInvertState()	Inverts the display based on an input parameter.
Seg_Display_ReadInvertState()	Returns the current value of the display invert state: normal or inverted
Seg_Display_ClearDisplay()	Clears the display and associated frame buffer RAM.
Seg_Display_WritePixel()	Sets or clears a pixel based on **PixelState**
Seg_Display_ReadPixel()	Reads the state of a pixel in the frame buffer.
Seg_Display_SetAwakeMode()	When in low-power mode, sets LCD Driver buffers output to high impedance.
Seg_Display_Sleep()	Stops the LCD and saves the user configuration.

Function	Description
Seg_Display_Wakeup()	Restores and enables the user configuration.
Seg_Display_Init()	Initializes or restores the LCD according to the Configure dialog settings.
Seg_Display_Enable()	Enables the component.
Seg_Display_SaveConfig()	Saves the LCD configuration.
Seg_Display_RestoreConfig()	Restores the LCD configuration.

Global Variables

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

uint8 Seg_Display_Start(void)

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

Parameters: None

Return Value: uint8 cystatus: Standard API returns values

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

Side Effects: None

void Seg_Display_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 Seg_Display_EnableInt(void)

Description: Enables the LCD interrupts. Not required if Seg_Display_Start() is called. An interrupt occurs after every LCD update (TD completion).

Parameters: None

Return Value: None

Side Effects: None

void Seg_Display_DisableInt(void)

Description: Disables the LCD interrupts. Not required if Seg_Display_Stop() is called.

Parameters: None

Return Value: None

Side Effects: None

uint8 Seg_Display_SetBias(uint8 biasLevel)

Description: This function sets the bias level for the LCD glass to one of 128 values. The actual number of values is limited by the Analog supply voltage, V_{DDA}. The bias voltage cannot exceed V_{DDA}. Changing the bias level affects the LCD contrast.

Parameters: uint8 biasLevel: Bias level for the display

Return Value: uint8 cystatus: Standard API returns values.

Return Value	Description
CYRET_BAD_PARAM	Evaluation of **biasLevel** parameter is failed
CYRET_SUCCESS	Function completed successfully

Side Effects: None

uint8 Seg_Display_WriteInvertState(uint8 invertState)

Description: This function inverts the display based on an input parameter. The inversion occurs in hardware and no change is required to the display RAM in the frame buffer

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

Value	Description
Seg_Display_NORMAL_STATE	Set normal state of display
Seg_Display_INVERTED_STATE	Set inverted state of display

Return Value: uint8 cystatus: Standard API returns values

Return Value	Description
CYRET_BAD_PARAM	Evaluation of **invertState** parameter failed
CYRET_SUCCESS	Function completed successfully

Side Effects: None

uint8 Seg_Display_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

Return Value	Description
Seg_Display_NORMAL_STATE	Normal state of display
Seg_Display_INVERTED_STATE	Inverted state of display

Side Effects: None

void Seg_Display_ClearDisplay(void)

Description: This function clears the display and the associated frame buffer RAM.

Parameters: None

Return Value: None

Side Effects: None

uint8 Seg_Display_WritePixel(uint16 pixelNumber, uint8 pixelState)

Description: This function sets or clears a pixel based on the input parameter **pixelState**. 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. The generated component .h file includes a #define of this format for each pixel.

uint8 pixelState: The **pixelNumber** specified is set to this pixel state.

Value	Description
Seg_Display _PIXEL_STATE_OFF	Set the pixel to off.
Seg_Display_PIXEL_STATE_ON	Set the pixel to on.
Seg_Display_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 Seg_Display_ReadPixel(uint16 pixelNumber)

Description: This function reads the state of a pixel 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. The generated component .h file includes a #define of this format for each pixel.

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

Value	Description
0x00	The pixel is off.
0x01	The pixel is on.
0xFF	The pixel isn't connected

Side Effects: None

void Seg_Display_SetAwakeMode(void)

Description: When in low-power mode, set LCD Driver buffer's output to high impedance.

Parameters: None

Return Value: None

Side Effects: None

void Seg_Display_SetSleepMode(void)

Description: When in low-power mode, set LCD Driver buffer's output to ground.

Parameters: None

Return Value: None

Side Effects: None

void Seg_Display_Sleep(void)

Description: The Seg_Display_Sleep() function checks to see if the component is enabled and saves that state. Then it calls the Seg_Display_Stop() function and calls Seg_Display_SaveConfig() function to save the user configuration.

Call the Seg_Display_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: Doesn't change component pin drive modes.

uint8 Seg_Display_Wakeup(void)

Description: The Seg_Display_Wakeup() function calls the Seg_Display_RestoreConfig() function to restore the user configuration. If the component was enabled before the Seg_Display_Sleep() function was called, the Seg_Display_Wakeup() function will re-enable the component.

Parameters: None

Return Value: uint8 cystatus: Standard API return values.

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

Side Effects: Calling the Seg_Display_Wakeup() function without first calling the Seg_Display_Sleep() or Seg_Display_SaveConfig() function may produce unexpected behavior.

void Seg_Display_Init(void)

Description: Initializes or restores the component parameters according to the Configure dialog settings. Configures and enables all required hardware blocks, clears frame buffer.

Parameters: None

Return Value: None

Side Effects: None

uint8 Seg_Display_Enable(void)

Description: Enables the component. Enables all required clocks and sets initial values for registers and performs component reset.

Parameters: None

Return Value: uint8 cystatus: Standard API return values.

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

Side Effects: None

void Seg_Display_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 Seg_Display_Sleep() function.

Parameters: None

Return Value: None

Side Effects: None

void Seg_Display_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 Seg_Display_Sleep() function.

Parameters: None

Return Value: None

Side Effects: Calling this function without first calling the Seg_Display_Sleep() or Seg_Display_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
Seg_Display_Write7SegDigit_n	Displays a hexadecimal digit on an array of 7-segment display elements.
Seg_Display_Write7SegNumber_n	Displays an integer value on a one- to five-digit array of 7-segment display elements.
Seg_Display_WriteBargraph_n	Displays an integer location on a linear or circular bar graph.
Seg_Display_PutChar14Seg_n	Displays a character on an array of 14-segment alphanumeric character display elements.
Seg_Display_WriteString14Seg_n	Displays a null terminated character string on an array of 14-segment alphanumeric character display elements.
Seg_Display_PutChar16Seg_n	Displays a character on an array of 16-segment alphanumeric character display elements.
Seg_Display_WriteString16Seg_n	Displays a null terminated character string on an array of 16-segment alphanumeric character display elements.
Seg_Display_PutCharDotMatrix_n	Displays a character on an array of dot matrix alphanumeric character display elements.
Seg_Display_WriteStringDotMatrix_n	Displays a null terminated character string on an array of dot matrix alphanumeric character display elements.

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” suffix in the function name.

void Seg_Display_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: The unsigned integer value in the range of 0 to 16 to be displayed as a hexadecimal digit. The ASCII numbers of hexadecimal characters are also valid. In the case of an invalid **digit**, **value** displays 0 in the specified position. Setting this parameter to 16 clears digits in the specified **position**.

uint8 position: The 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 will not be displayed..

Return Value: None

Side Effects: None

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

Description: This function displays a 16-bit integer value on a 1 to 5 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 than **value** requires, the most significant digit or digits will not be displayed

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

Return Value: None

Side Effects: None

void Seg_Display_WriteBargraph_n(uint8 location, uint8 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: uint8 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.

uint8 mode: Sets the bar graph display mode.

Value	Description
0	The 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 to 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 Seg_Display_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 will not be displayed.

Return Value: None

Side Effects: None

void Seg_Display_WriteString14Seg_n(*uint8 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 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 will not be displayed.

Return Value: None

Side Effects: None

void Seg_Display_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 will not be displayed.

Return Value: None

Side Effects: None

(void) Seg_Display_WriteString16Seg_n(*uint8 character, uint8 position)

Description: This function displays a null terminated character string on an array of 16segment 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 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 will not be displayed.

Return Value: None

Side Effects: None

void Seg_Display_PutCharDotMatrix_n(uint8 character, uint8 position)

Description: This function displays an 8-bit character on an array of dot-matrix alphanumeric character display elements. The customizer Display Helpers facility must be used to define the pixel set associated with the dot-matrix display elements. Multiple dot-matrix 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 dot-matrix display element is defined in the component customizer

Parameters: uint8 character: The ASCII value of the character to display.

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 will not be displayed.

Return Value: None

Side Effects: None

void Seg_Display_WriteStringDotMatrix_n(*uint8 character, uint8 position)

Description: This function displays a null terminated character string on an array of dot-matrix alphanumeric character display elements. The customizer Display Helpers facility must be used to define the pixel set associated with the dot-matrix display elements. Multiple dot-matrix 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 dot-matrix display element is defined in the component customizer

Parameters: *uint8 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 will not be displayed.

Return Value: None

Side Effects: None

Pins APIs

These API functions are used to change the drive mode of pins used by the Segment Display component.

Function	Description
Seg_Display_ComPort_SetDriveMode	Sets the drive mode for all pins used by common lines of the Segment Display component.
Seg_Display_SegPort_SetDriveMode	Sets the drive mode for all pins used by segment lines of the Segment Display component.

void Seg_Display_ComPort_SetDriveMode(uint8 mode)

Description: Sets the drive mode for all pins used by common lines of the Segment Display component.

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

Return Value: None

Side Effects: None

Seg_Display_SegPort_SetDriveMode(uint8 mode)

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

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

Return Value: None

Side Effects: None

Macros

Seg_Display_COMM_NUM

Defines the number of common lines in the user-defined display for the current configuration of the component.

Seg_Display_SEG_NUM

Defines the number of segment lines for the user-defined display for the current configuration of the component.

Seg_Display_BIAS_TYPE

Defines the bias type for the user-defined display for the current configuration of the component.

Seg_Display_BIAS_VOLTAGE

Defines default bias voltage level for the user-defined display. This value will be set in LCDDAC control register during the initialization process.

Seg_Display_FRAME_RATE

Defines the refresh rate for the user-defined display for the current configuration of the component.

Seg_Display_WRITE_PIXEL

This is a macro define of the Seg_Display_WritePixel() function with void type.

Seg_Display_READ_PIXEL

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

Seg_Display_FIND_PIXEL

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

Sample Firmware Source Code

PSoC Creator provides many 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.

Functional Description

Default Configuration

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

- Four common lines
- Eight segment lines

- 60-Hz refresh rate
- Always Active power mode
- No display helpers are defined. Default API generation does not include functions for any of the supported display elements.

Custom Configuration

A key feature of the Segment Display component is flexible support for LCDs with different characteristics and layouts.

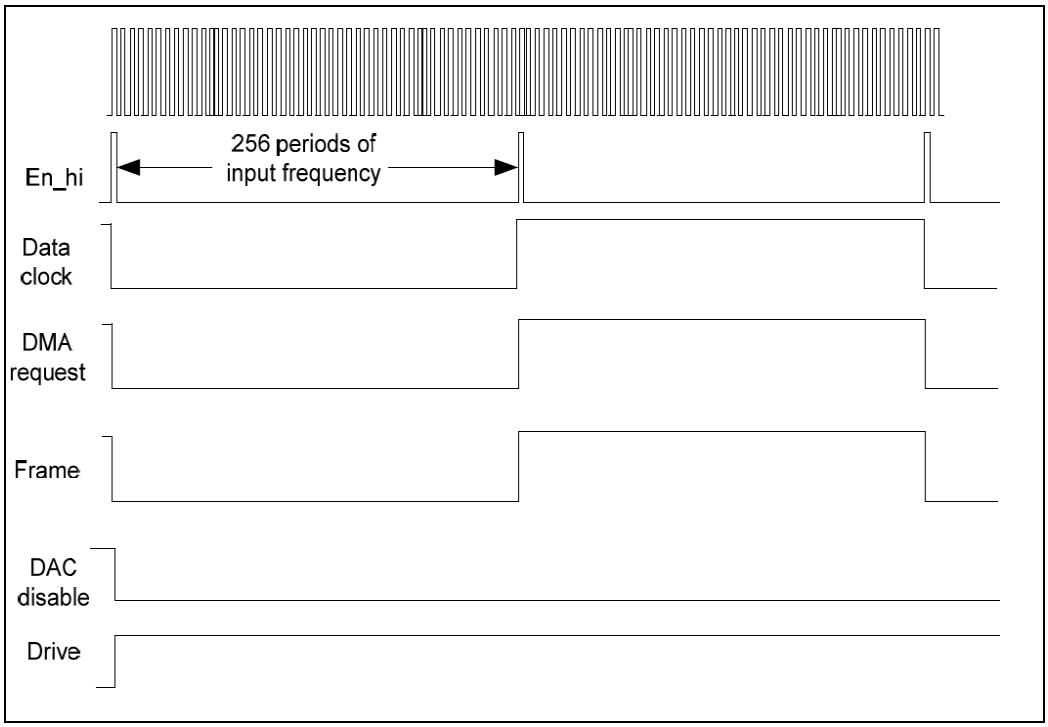
Driver Power Modes

Always Active Mode

In this use model, the LCD is driven throughout the entire frame. This means that the LCD DAC is powered and the internal signal drive is asserted high whenever the component is enabled.

Figure 1 shows waveforms for UDB-generated (internal) signals of the Seg_Display component for Always Active mode (Type A):

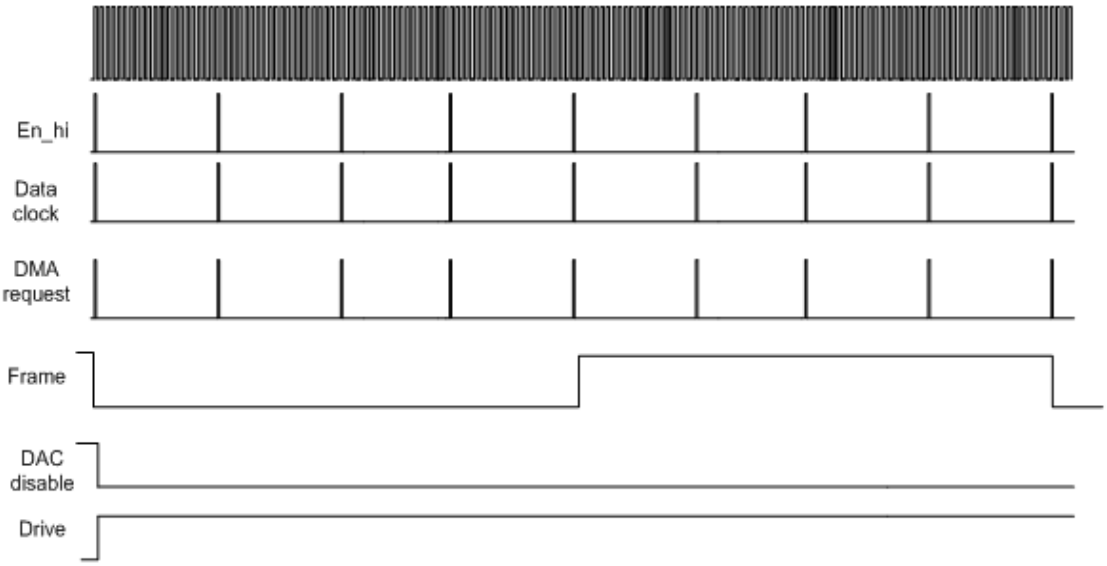
Figure 1. Segment Display Control Signals Type A Waveform (Always Active mode)



Note See Timing Calculations for more information.

Figure 2 shows waveforms for UDB-generated signals of the Seg_Display component for Always Active mode (Type B).

Figure 2. Segment Display Control Signals Type B Waveform (Always Active mode)



Signals shown are for the 1/4 multiplex ratio case.

Low Power Mode

In this use model, the LCD is actively driven only at voltage transitions and the LCD system analog components are powered down between voltage transitions.

Figure 3 shows waveforms for UDB-generated signals of the Seg_Display component for Low Power mode (Type A).

Figure 3. Segment Display Control Signals Type A Waveform (Low Power mode)

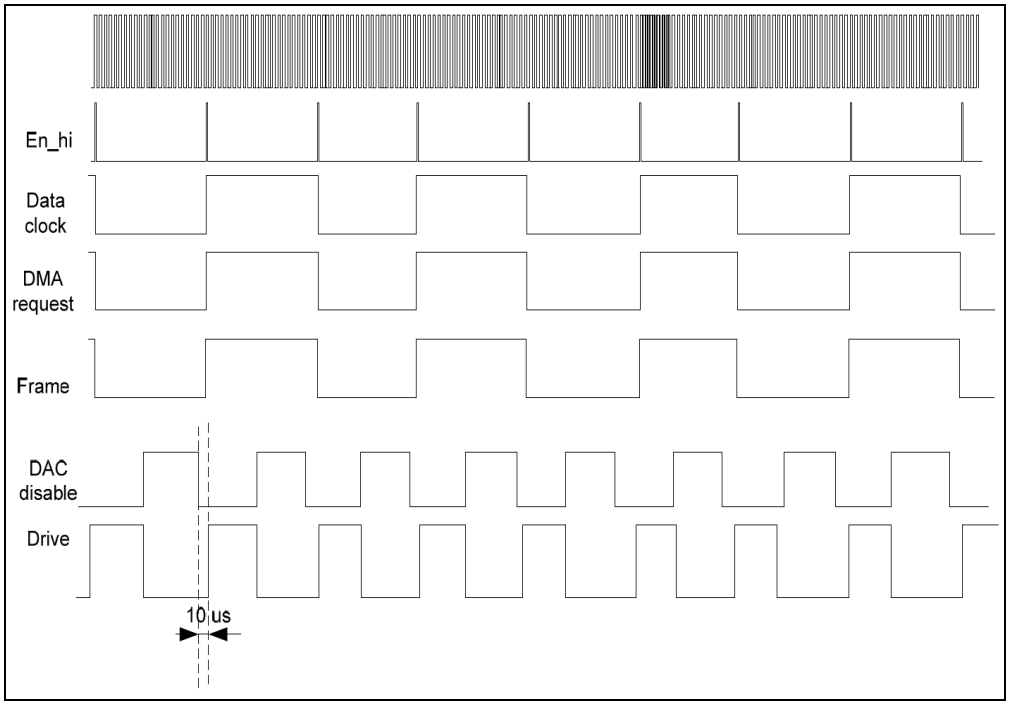
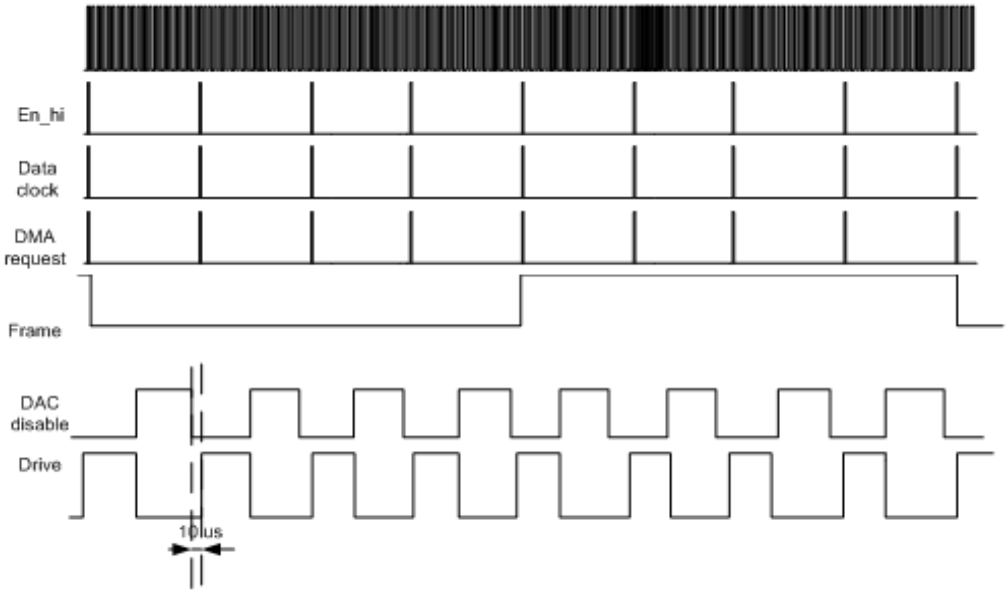


Figure 4 shows waveforms for UDB-generated signals of the Seg_Display component for Low Power mode (Type B).

Figure 4. Segment Display Control Signals Type B Waveform (Low Power mode)



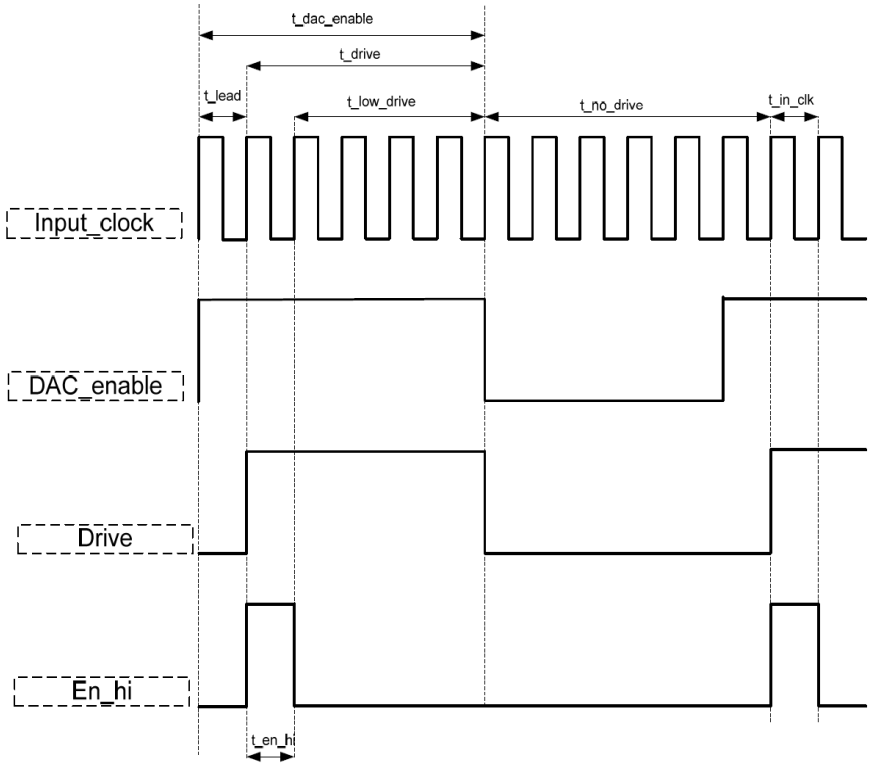
Signals shown are for the j multiplex ratio case.

Timing Calculations

Figure 5 and the following table show the timing information for the UDB-generated signals. In Figure 5, only three internal signals are represented. The other signals can be derived from the previous diagrams. Figure 5 is based on Low Power mode and a Type B waveform.

The DAC_disable signal is generated by inverting the internal DAC_enable signal.

Figure 5. Control Signal Timing Diagram Example



Parameter	Time	Description	Calculation
Input clock frequency	-	Input clock frequency value for Type A waveforms (set automatically by customizer)	f_in = f_frame_rate × N_common_lines × 256 × 2
		Input clock frequency value for Type A waveforms (set automatically by customizer)	f_in = f_frame_rate × N_common_lines × 256
Input clock period	t_in_clk	Specifies period of input clock	t_in_clk = 1/(f_in)
High drive time	t_en_hi	Specifies period of time HiDrive will be active.	t_en_hi = t_in_clk * N, where N is a constant step value from range 1 .. 253 (for Always Active mode) or 1 .. 247 (for Low Power mode)

Parameter	Time	Description	Calculation
	t_lead	Specifies LCD DAC setup time.	~10 µs
	-	HiDrive inactive time	t_low_drive (Always Active) t_low_drive + t_no_drive (Low Power)
	t_drive	Specifies drive time	t_drive = t_en_hi + t_low_drive
	t_drive	Specifies drive time	t_drive = t_en_hi + t_low_duty_cycle
	t_low_drive	Specifies low drive time	t_low_drive = t_drive - t_en_hi
	t_no_drive	Specifies the time when drive signal will be asserted low	t_no_drive = t_in_clk * 256 - t_drive
	t_dac_enable	Specifies the time during which LCD DAC is turned on (only for Low Power mode)	t_dac_enable = t_drive + t_lead

User-Specific Configuration

Signal timing can be adjusted by changing the timing parameters in the component customizer.

High Drive Time

By default, depending on the operating mode, High drive time equals:

High drive time = 128 × t_in_clk for Always Active mode, or

High drive time = 64 × t_in_clk for Low Power mode.

This value is automatically calculated by the customizer. You can increase this time up to a maximum value of:

HiDriveTimemax = HiDriveTimemin × 253, for Always Active mode

HiDriveTimemax = HiDriveTimemin × 247, for Low Power mode

The HiDriveTime value increases in increments corresponding to one cycle of the input clock. This results in the active time for the en_hi signal being extended.

Block Diagram and Configuration

Figure 6. Segment Display Component Schematic

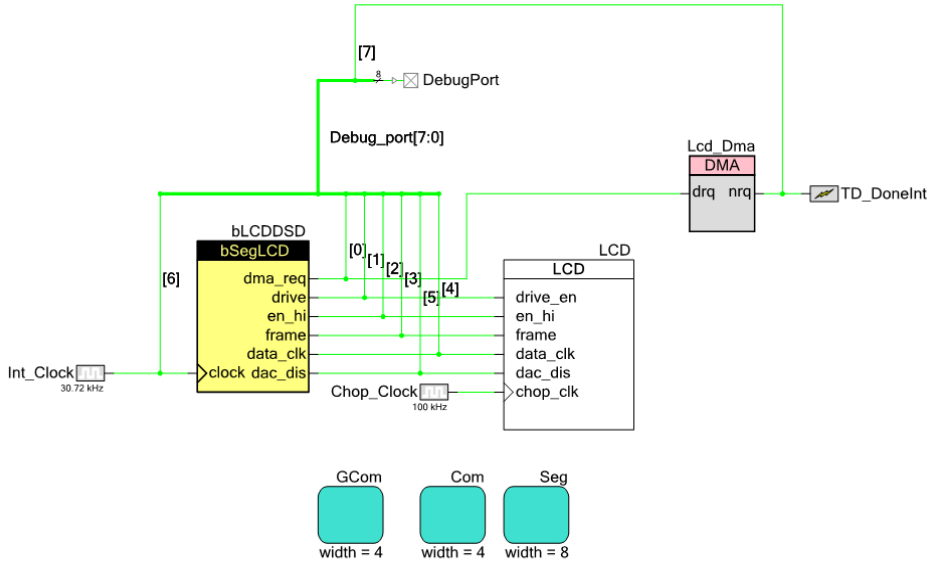
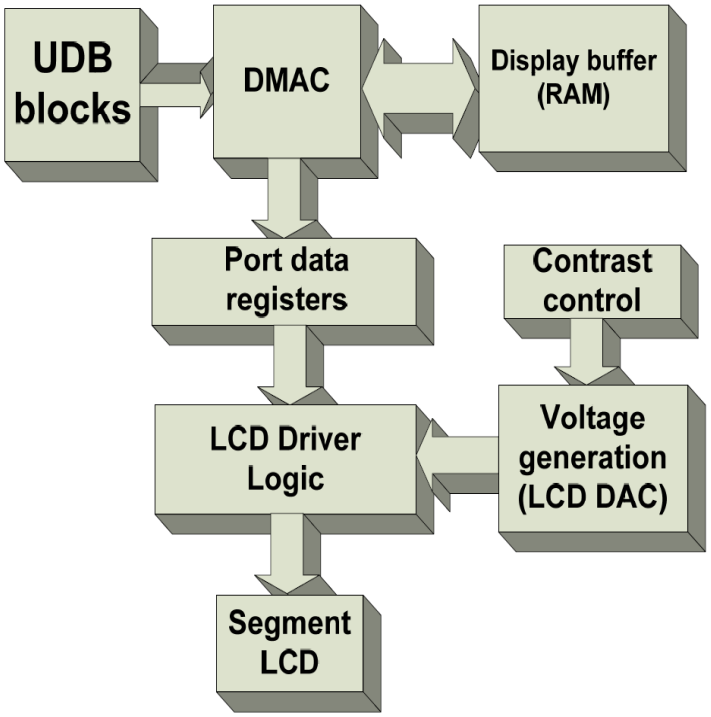


Figure 6 shows the internal schematic for the Segment Display component. It consists of a basic Segment Display component, LCD Control block (LCD) component, DMA component, two LCD ports, one digital port, an ISR component, and two clocks.

- The basic Segment Display component is responsible for generating the proper timing signals for the LCD Port and DMA components.
- The DMA component is used to transfer data from the frame buffer to the LCD data registers through the aliased memory area.
- The LCD component handles the required DSI routing. This block also provides the required register names as defined in *cyfitter.h*.
- The LCD Ports (GCom, 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. The LCD Port for the common signals is limited to 16 pins wide and the LCD Port for segment signals is limited to 48 pins wide.
- The DebugPort component is used only for debug purposes. This component is removed by default.

Top Level Architecture

Figure 7. Segment Display Top Level



Registers

Seg_Display_CONTRAST_CONTROL

Holds bias voltage level which is used by LCD DAC to generate proper bias voltage. An API is provided to change bias voltage level.

Bits	7	6	5	4	3	2	1	0
Value	reserved	contrast level						

- contrast level: Bias voltage level described above.

Seg_Display_LCDDAC_CONTROL

Bits	7	6	5	4	3	2	1	0
Value	reserved					DAC disable	bias select	

- bias select: Selects bias.
- DAC disable: Disables LCD DAC if contrast control is not needed.

Seg_Display_DRIVER_CONTROL

Bits	7	6	5	4	3	2	1	0
Value	reserved					invert	lo2	sleep mode

- sleep mode: Sets the sleep mode for Segment Display.
- lo2: Enables or disables the high-current mode of loDrive mode of the LCD Driver block
- invert: If set, inverts all data in on the Segment Display.

Seg_Display_CONTROL

Bits	7	6	5	4	3	2	1	0
Value	reserved						control reset	clock enable

- clock enable: Enables generation of all internal signals described in above sections.
- control reset: Performs initial reset of the digital portion of the component.

Seg_Display_LCDDAC_SWITCH_REG[0..4]

Bits	7	6	5	4	3	2	1	0
Value	reserved						switch control[0..4]	

- switch control[0..4]: This set of bit-fields selects voltage sources for LCD Driver.

Resources

The Segment Display component is a dedicated piece of hardware (LCD Fixed Block) in PSoC 5 that is used in conjunction with UDBs to allow direct driving of external LCD glass. The component utilizes the following resources:

Configuration[1]	Resource Type					
	Datapath Cells	Macrocells	Status Cells	Control Cells	DMA Channels	Interrupts
Always Active Mode	1	5	–	1	1	1
Low Power Mode	2	9	–	1	1	1

1. The configurations are using Type A waveforms, Type B waveforms additionally utilize 1 Control Cell and 28 bytes of flash.

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, Dot Matrix helper: Low-level API functions set + Dot Matrix helper high-level API

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

Configuration[2]	PSoC 3 (Keil_PK51)		PSoC 5 (GCC)		PSoC 5LP (GCC)	
	Flash Bytes	SRAM Bytes	Flash Bytes	SRAM Bytes	Flash Bytes	SRAM Bytes
Basic	N/A	N/A	1162	81	N/A	N/A
Basic, 7-segment helper	N/A	N/A	322	81	N/A	N/A
Basic, 14-segment helper	N/A	N/A	1528	81	N/A	N/A
Basic, 16-segment helper	N/A	N/A	1532	81	N/A	N/A
Basic, Dot Matrix helper	N/A	N/A	2620	97	N/A	N/A
Basic, Bar Graph helper	N/A	N/A	1266	81	N/A	N/A

2. The configurations are using Type A waveforms, Type B waveforms additionally utilize 1 Control Cell and 28 bytes of flash.

DC and AC Electrical Characteristics

Specifications are valid for –40 °C ≤ T_A ≤ 85 °C and T_J ≤ 100 °C, except where noted.
Specifications are valid for 2.7 V to 5.5 V, except where noted.

DC Characteristics

Parameter	Description	Conditions	Min	Typical	Max	Units
I_{cc}	LCD system operating current	Bus clock = 3 MHz, VDDIO = VDDA = 3 V, 4 commons, 16 segments, ¼ duty cycle, 50 Hz rame rate, no glass connected	–	63	–	µA
I_{CC_SEG}	Current per segment driver		–	148	–	µA
V_{BIAS}	LCD bias range (V_{BIAS} refers to the main output voltage(V0) of LCD DAC)	3 V ≤ V_{BIAS} ≤ V_{DDIO} for the drive pin	2.09	–	5.2	V
	LCD bias step size	3 V ≤ V_{BIAS} ≤ V_{DDIO} for the drive pin	–	25.8	–	mV
	LCD capacitance per segment/common driver	Drivers may be combined	–	500	5000	pF
	Long term segment offset	V_{BIAS} ≤ V_{DDA} - 0.5 V	–	–	20	mV
I_{OUT}	Output drive current per segment driver	V_{DDIO} = 5.5 V	90	–	165	µA

AC Characteristics

Parameter	Description	Min	Typ	Max	Units
f_{LCD}	LCD frame rate	10	50	150	Hz

Component Changes

Version	Description of Changes	Reason for Changes / Impact
1.10	Minor datasheet corrections.	
	Updated component resource usage and API memory usage data.	
1.0.a	Datasheet corrections	
1.0	First release.	

Note The Segment Display component is available for PSoC 5 devices only. For PSoC 3 Production and PSoC 5LP devices, use the Segment LCD version 3.10 or later component.

© Cypress Semiconductor Corporation, 2012. The information contained herein is subject to change without notice. Cypress Semiconductor Corporation assumes no responsibility for the use of any circuitry other than circuitry embodied in a Cypress product. Nor does it convey or imply any license under patent or other rights. Cypress products are not warranted nor intended to be used for medical, life support, life saving, critical control or safety applications, unless pursuant to an express written agreement with Cypress. Furthermore, Cypress does not authorize its products for use as critical components in life-support systems where a malfunction or failure may reasonably be expected to result in significant injury to the user. The inclusion of Cypress products in life-support systems application implies that the manufacturer assumes all risk of such use and in doing so indemnifies Cypress against all charges.

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

Any Source Code (software and/or firmware) is owned by Cypress Semiconductor Corporation (Cypress) and is protected by and subject to worldwide patent protection (United States and foreign), United States copyright laws and international treaty provisions. Cypress hereby grants to licensee a personal, non-exclusive, non-transferable license to copy, use, modify, create derivative works of, and compile the Cypress Source Code and derivative works for the sole purpose of creating custom software and or firmware in support of licensee product to be used only in conjunction with a Cypress integrated circuit as specified in the applicable agreement. Any reproduction, modification, translation, compilation, or representation of this Source Code except as specified above is prohibited without the express written permission of Cypress.

Disclaimer: CYPRESS MAKES NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS MATERIAL, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. Cypress reserves the right to make changes without further notice to the materials described herein. Cypress does not assume any liability arising out of the application or use of any product or circuit described herein. Cypress does not authorize its products for use as critical components in life-support systems where a malfunction or failure may reasonably be expected to result in significant injury to the user. The inclusion of Cypress' product in a life-support systems application implies that the manufacturer assumes all risk of such use and in doing so indemnifies Cypress against all charges.

Use may be limited by and subject to the applicable Cypress software license agreement.