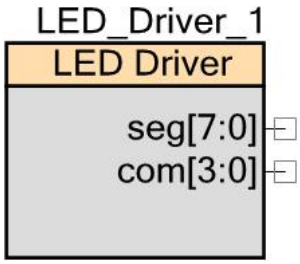
LED Segment and Matrix Driver

1.0



Features

- Up to 8 RGB 7-segment digits, or 24 monochrome 7-segment digits
- Up to 8 14-segment or 16-segment digits
- Up to 192 LEDs in an 8x8 tri-color matrix
- Active high or active low commons
- Active high or active low segments
- Driver is multiplexed requiring no CPU overhead or interrupts
- Functions for numeric and string display using 7-, 14-, and 16-segments
- Independent brightness level for each common signal

General Description

The LED Segment and Matrix Driver component is a multiplexed LED driver that can handle up to 24 segment signals and 8 common signals. It can be used to drive 24 7-segment LEDs, eight 14/16-segment LEDs, eight RGB 7-segment LEDs, or a tri-color matrix of up to 192 LEDs in an 8x8 pattern. APIs are provided to convert alpha-numeric values to their segment codes, and the brightness of each of the commons can be independently controlled. This component is supported for PSoC 3 and PSoC 5LP.

Multiplexing the LEDs is an efficient way to save GPIO pins, however the commons must be multiplexed at a steady rate. To address this latter issue, the component uses PSoC's DMA and UDBs to multiplex the LEDs without CPU overhead. This eliminates cases of non-periodic updating as the multiplexing is handled solely using hardware. The CPU is thus used only when updating the display information and to change the brightness settings.

When displaying the 7/14/16 segment digits, these digits do not have to be grouped as a single numerical display. An 8 digit display could be divided up into one 2-digit and two 3-digit displays for example. When operating in the LED matrix mode, the individual displays do not have to be arranged in a matrix, but instead can be various single or grouped LEDs. The component also supports displaying combined digits with annunciators.

When to Use an LED Segment and Matrix Driver

Although LEDs are slowly being replaced by LCD graphics and vacuum florescent displays, LEDs are still used in products from panel meters to exercise equipment. This component provides a quick and easy way to implement a display in a PSoC based application, with minimal code required by the designer. Since the multiplexing is handled by UDBs and DMA, the multiplexing requires no CPU overhead. Multiplexed displays, if not updated periodically, can be very annoying to the user. Using PSoC 3 and 5LPs UDBs and DMA eliminates non-periodic updating.

Input/Output Connections

This section describes the input and output connections for the LED Segment and Matrix Driver component. Some I/Os may be hidden on the symbol under the conditions listed in the description of that I/O.

Input	May Be Hidden	Description
clock	Y	External clock terminal for controlling the refresh rate of the commons.

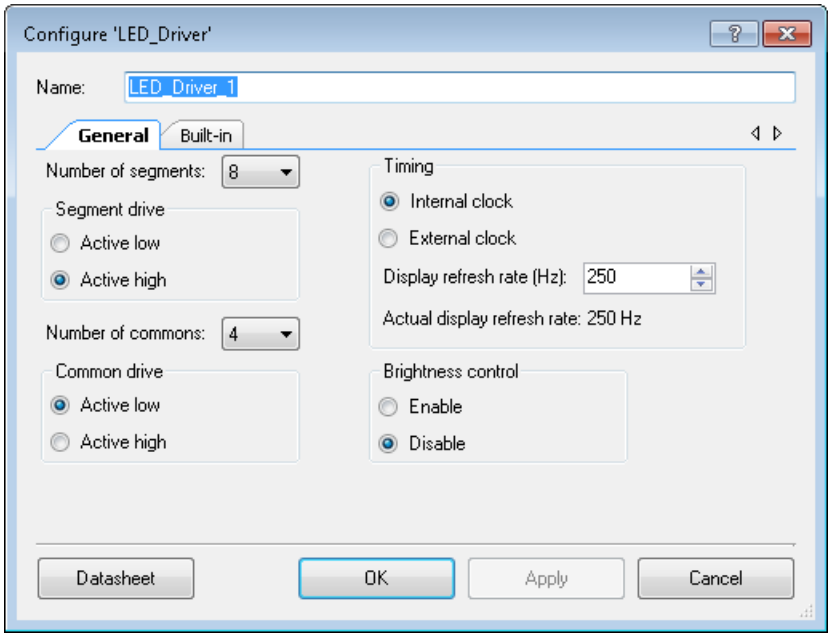
Output	May Be Hidden	Description
seg [23:0]	N	Segment display terminal for driving the segment GPIO pins. The number of segment lines can vary depending on the number of segments setting.
com [7:0]	N	Common display terminal for driving the common GPIO pins. The number of common lines can vary depending on the number of commons setting.

Component Parameters

Drag an LED Segment and Matrix Driver component onto your design and double-click it to open the **Configure** dialog.

General Tab

The **General** tab is used to set the general operational parameters of the component. It contains the following parameters. All of these settings are compile-time selections.



Number of segments

The number of segments is configured using this parameter. This component supports anywhere between 1 and 24 segments.

Segment drive

This parameter is used to specify whether the segments are **Active high** or **Active low**.

Number of commons

The number of commons is configured using this parameter. This component supports anywhere between 1and 8 commons.

Common drive

This parameter is used to specify whether the commons are **Active high** or **Active low**.

Timing – Clock source

The clock source can either be internally provided to automatically set the clock to the closest frequency for the desired refresh rate, or can be external to allow an external clock component to be used.

Timing – Display refresh rate

The refresh rate of each of the commons is configured using this parameter. The valid range for the refresh rate is from 1 Hz to a maximum rate equal to (Bus clock / (100 x number of commons)) for a design without brightness control. With brightness control, the maximum recommended refresh rate is (Bus clock / (100 x number of commons x 256). The **Actual display refresh rate** is also shown and may differ from the requested rate depending on the clock divider settings that are possible in the system. Configuring the refresh rate beyond these constraints may produce incorrect display results.

If an external clock is used, then the refresh rate will be equal to the clock rate divided by the number of commons when **Brightness control** is disabled. With **Brightness control enabled**, the refresh rate is further divided by 256. The same restrictions apply for the maximum refresh rate as in the internally clocked configuration.

Brightness control

The brightness of each of the commons can be changed at run time. This parameter allows the brightness control function to be enabled. Note that enabling the brightness control requires the component clock to be 256x faster than without brightness control.

Clock Selection

The LED Segment and Matrix Driver component contains an embedded internal clock that is configured by the **Display refresh rate** setting defined in the customizer. If the **Clock source** is external, then the internal clock is removed from the design and a clock must be provided on the clock terminal.

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 discuss each function in more detail.

By default, PSoC Creator assigns the instance name “LED_Driver_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 “LED_Driver”.

Function	Description
LED_Driver_Init()	Clears the displays, and initializes the display arrays and registers.
LED_Driver_Enable()	Initializes the DMAs and enables the component.
LED_Driver_Start()	Enables and starts the component.
LED_Driver_Stop()	Clears the display, disables the DMA and stops the component.
LED_Driver_SetDisplayRAM()	Writes a value directly into the display RAM at the specified position.
LED_Driver_SetRC()	Sets the bit in the display RAM in the specified row and column.
LED_Driver_ClearRC()	Clears the bit in the display RAM in the specified row and column.
LED_Driver_ToggleRC()	Toggles the bit in the display RAM in the specified row and column.
LED_Driver_GetRC()	Returns the bit value in the display RAM in the specified row and column.
LED_Driver_ClearDisplay()	Clears the display for the specified common to zero.
LED_Driver_ClearDisplayAll()	Clears the entire display to 0.
LED_Driver_Write7SegNumberDec()	Displays a 7-segment signed integer up to 8 characters long, starting at the specified position and extending to a specified number of digits.
LED_Driver_Write7SegNumberHex()	Displays a 7-segment hexadecimal number up to 8 characters long, starting at the specified position and extending to a specified number of digits.
LED_Driver_WriteString7Seg()	Displays a 7-segment null terminated string starting at the specified position and ending at either the end of the string or the end of the display.
LED_Driver_PutChar7Seg()	Displays a 7-segment ASCII encoded character at the specified position.
LED_Driver_Write7SegDigitDec()	Displays a single 7-segment digit (0…9) on the specified display.
LED_Driver_Write7SegDigitHex()	Displays a single 7-segment digit (0...F) on the specified display.
LED_Driver_Write14SegNumberDec()	Displays a 14-segment signed integer up to 8 characters long, starting at the specified position and extending to a specified number of digits.
LED_Driver_Write14SegNumberHex()	Displays a 14-segment hexadecimal number up to 8 characters long, starting at the specified position and extending to a specified number of digits.
LED_Driver_WriteString14Seg()	Displays a 14-segment null terminated string starting at the specified position and ending at either the end of the string or the end of the display.
LED_Driver_PutChar14Seg()	Displays a 14-segment ASCII encoded character at the specified position.
LED_Driver_Write14SegDigitDec()	Displays a single 14-segment digit (0…9) on the specified display.
LED_Driver_Write14SegDigitHex()	Displays a single 14-segment digit (0...F) on the specified display.

Function	Description
LED_Driver_Write16SegNumberDec()	Displays a 16-segment signed integer up to 8 characters long, starting at the specified position and extending to a specified number of digits.
LED_Driver_Write16SegNumberHex()	Displays a 16-segment hexadecimal number up to 8 characters long, starting at the specified position and extending to a specified number of digits.
LED_Driver_WriteString16Seg()	Displays a 16-segment null terminated string starting at the specified position and ending at either the end of the string or the end of the display.
LED_Driver_PutChar16Seg()	Displays a 16-segment ASCII encoded character at the specified position.
LED_Driver_Write16SegDigitDec()	Displays a single 16-segment digit (0…9) on the specified display.
LED_Driver_Write16SegDigitHex()	Displays a single 16-segment digit (0...F) on the specified display.
LED_Driver_PutDecimalPoint()	Sets or clears the decimal point at the specified position.
LED_Driver_GetDecimalPoint()	Returns zero if the decimal point is not set and one if the decimal point is set.
LED_Driver_EncodeNumber7Seg()	Converts the lower 4 bits of the input into 7-segment data that will display the number in hex on a display.
LED_Driver_EncodeChar7Seg()	Converts the ASCII encoded alphabet character input into the 7-segment data that will display the alphabet character on a display.
LED_Driver_EncodeNumber14Seg()	Converts the lower 4 bits of the input into 14-segment data that will display the number in hex on a display.
LED_Driver_EncodeChar14Seg()	Converts the ASCII encoded alphabet character input into the 14-segment data that will display the alphabet character on a display.
LED_Driver_EncodeNumber16Seg()	Converts the lower 4 bits of the input into 16-segment data that will display the number in hex on a display.
LED_Driver_EncodeChar16Seg()	Converts the ASCII encoded alphabet character input into the 16-segment data that will display the alphabet character on a display.
LED_Driver_SetBrightness()	Sets the desired brightness value (0 = display off; 255 = display at full brightness) for the chosen common.
LED_Driver_GetBrightness()	Returns the brightness value for the specified common.
LED_Driver_Sleep()	Stops the component and saves the user configuration
LED_Driver_Wakeup()	Restores the user configuration and enables the component

Many of the following APIs refer to a “display” and “value” parameter. The “display” refers to the position of the common in the display RAM array, and the “value” refers to the segment value to be written into the position in the display RAM. These are illustrated in Figure 1 for a 7-segment display and an 8x8 LED matrix. The display RAM has eight elements, each of which corresponds

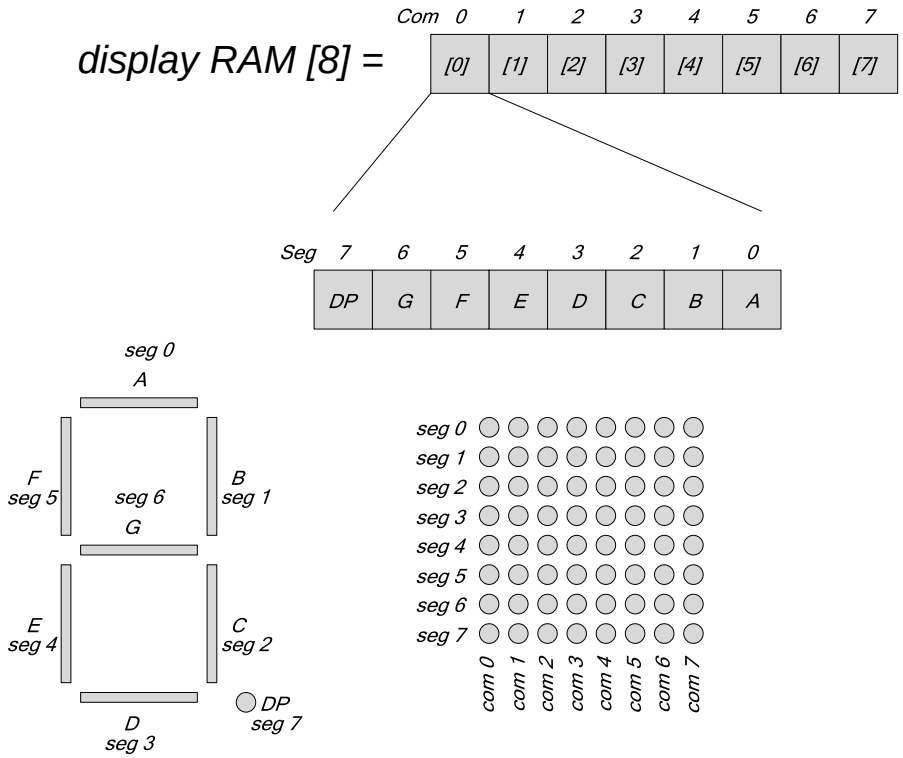
to a common. This is the “display”. Each of these commons holds an 8-bit “value” to be written to the segment lines.

The APIs also refer to “rows” and “columns”. References to “rows” are the same as the segments, and “columns” are the same as the commons.

The following shows the structure of the display RAM and how it is mapped to each common and segment for a 7-segment display and an 8x8 LED array.

When the segment drive is configured for active low drive, the off value for an LED segment is a one in the display RAM. The inversion of the display RAM value is handled automatically by all the APIs that read and write the display RAM. Application code should be written the same for either segment drive configuration and should interpret a one value as ON and a zero value as OFF.

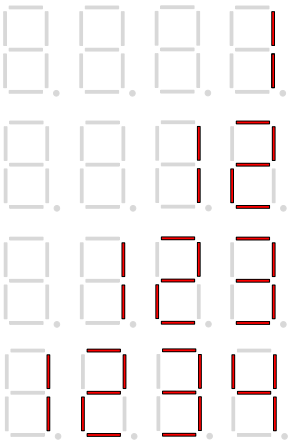
Figure 1. Display RAM



When addressing multi-digit displays the first digit or most significant digit on the left is the lowest column number. For example, if the displayed number is “1234”, the “1” would be driven by common 0, the “2” would be driven by common 1, and so on. The reference to the common 0, common 1 etc. is referred to as the “position” of the display, and can be from 0 up to 23 for a 7-segment display. The number of allowable “positions” for 14 and 16-segment displays is from 0 to 7.

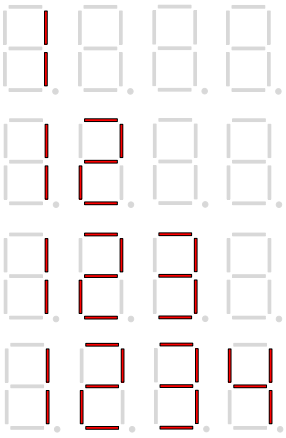
If a display is to be right aligned, the least significant digit is always placed in the right most display position as shown in Figure 2.

Figure 2. Right Justified Display for the values 1, 12, 123, and 1234



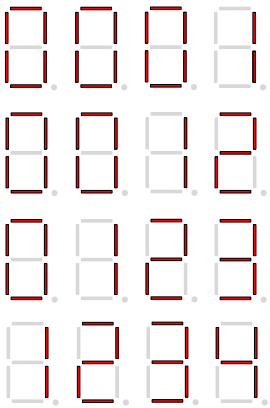
If the display is left aligned, then the most significant digit is always placed in the left most display position as shown in Figure 3.

Figure 3. Left Justified Display for the values 1, 12, 123, and 1234



If zero padding is chosen, then the data is displayed right justified padded with leading zeros for all positions left of the value. This is shown in Figure 4. If the value is negative, then the “-” sign is placed at the left most position.

Figure 4. Zero Padding Display for the values 1, 12, 123, and 1234



void LED_Driver_Init(void)

Description: Clears the display, and initializes the DMAs. Also initializes the brightness array if brightness control is enabled.

Parameters: None

Return Value: None

Side Effects: None

void LED_Driver_Enable(void)

Description: Enables the DMAs and enables the PWM if brightness control is enabled. Once these are complete, the component is enabled.

Parameters: None

Return Value: None

Side Effects: None

void LED_Driver_Start(void)

Description: Configures the hardware (DMA and optional PWM) and enables the LED display by calling LED_Driver_Init() and LED_Driver_Enable(). If LED_Driver_Init() had been called before, then the LEDs will display whatever values that are currently in the display RAM. If it is the first call, then the display RAM will be cleared.

Parameters: None

Return Value: None

Side Effects: None

void LED_Driver_Stop(void)

Description: Clears the display RAM, disables all DMA channels and stops the PWM (if brightness enabled)

Parameters: None

Return Value: None

Side Effects: None

void LED_Driver_SetDisplayRAM(uint8 value, uint8 position)

Description: Writes ‘value’ directly into the display RAM. This function writes a single byte into the display RAM associated with a set of 8 segments designated by the ‘position’ argument.

Parameters: uint8 value: Desired value to write into the display RAM. A “1” enables the segment, a “0” disables the segment.

uint8 position: Value from 0 to 23 designating the position in the Display RAM to write. The position starts at 0 for common 0 and segments 0 to 7 and is numbered sequentially through the commons and then the segments. For example, with a configuration of a 16-segment LED with 4 digits (4 commons), to write to segments 8 to 15 on digit 1, the position used is 5. Refer to Figure 9 and Figure 10 for further details on how positions are numbered.

Return Value: None

Side Effects: None

void LED_Driver_SetRC(uint8 row, uint8 column)

Description: Sets the bit in the display RAM corresponding to the LED in the designated row and column. Note that rows are the segments and columns are the commons.

Parameters: uint8 row: Row value 0 to 23

uint8 column: Column value 0 to 7

Return Value: None

Side Effects: None

void LED_Driver_ClearRC(uint8 row, uint8 column)

Description: Clears the bit in the display RAM corresponding to the LED in the designated row and column.

Parameters: uint8 row: Row value 0 to 23
uint8 column: Column value 0 to 7

Return Value: None

Side Effects: None

void LED_Driver_ToggleRC(uint8 row, uint8 column)

Description: Toggles the bit in the display RAM corresponding to the LED in the designated row and column.

Parameters: uint8 row: Row value 0 to 23
uint8 column: Column value 0 to 7

Return Value: None

Side Effects: None

uint8 LED_Driver_GetRC(uint8 row, uint8 column)

Description: Returns the bit value in the display RAM corresponding to the LED in the designated row and column.

Parameters: uint8 row: Row value 0 to 23
uint8 column: Column value 0 to 7

Return Value: uint8: Returns 1 if the value is high, and 0 if the value is low.

Side Effects: None

void LED_Driver_ClearDisplay(uint8 position)

Description: Clears the display (disables all the LEDs) for a set of 8 segments designated by the 'position' argument.

Parameters: uint8 position: Value from 0 to 23 designating the position in the Display RAM to write. The position starts at 0 for common 0 and segments 0 to 7 and is numbered sequentially through the commons and then the segments. For example, with a configuration of a 16-segment LED with 4 digits (4 commons), to write to segments 8 to 15 on digit 1, the position used is 5. Refer to Figure 9 and Figure 10 for further details on how positions are numbered.

Return Value: None

Side Effects: None

void LED_Driver_ClearDisplayAll(void)

Description: Clears the entire display by writing zeros to all the display RAM locations.

Parameters: None

Return Value: None

Side Effects: None.

void LED_Driver_Write7SegNumberDec(int32 number, uint8 position, uint8 digits, uint8 alignment)

Description: Displays a 7-segment signed integer up to 8 characters long, starting at “position” and extending for “digits” characters. The negative sign will consume one digit if it is required. If the number exceeds the digits specified, the least significant digits will be displayed. For example, if number is -1234, position is 0 and digits is 4, the result will be: -234. Note that the positions of the digits are continuous and it is up to the user to choose the correct position for the application. Also note that any digits that extend beyond the configured number of commons are discarded. For more information, see Figure 9 and Figure 10.

Parameters: int32 number: a signed integer number to display. It is the responsibility of the user to ensure that the display has enough digits to accurately represent the number passed to this function. If not, the least significant digits will be displayed. Also note that a negative number will require 1 more digit than the equivalent positive number to display the negative sign.

uint8 position: Digit position of the Display/Common RAM to start.

uint8 digits: The number of digits to display the value in. The negative sign will consume one digit if it is required. If the digits extend beyond the configured number of commons, then these digits are discarded.

uint8 alignment: How to align the provided number in the allocated digits.

Value	Description
LED_Driver_RIGHT_ALIGN	Least significant digit occupies the rightmost digit (position + digits). Unused digits are turned off.
LED_Driver_LEFT_ALIGN	Most significant digit (or negative sign) occupies the digit specified by position. Unused digits are turned off.
LED_Driver_ZERO_PAD	Unused digits to the left are padded with leading zeros.

Return Value: None

Side Effects: None

void LED_Driver_Write7SegNumberHex(uint32 number, uint8 position, uint8 digits, uint8 alignment)

Description: Displays a 7-segment hexadecimal number up to 8 characters long, starting at “position” and extending for “digits” characters. If the number exceeds the digits specified, the least significant digits will be displayed. For example, if number is 0xDEADBEEF, position is 0 and digits is 4, the result will be: BEEF. Note that the positions of the digits are continuous and it is up to the user to choose the correct position for the application. Also note that any digits that extend beyond the configured number of commons are discarded. For more information, see Figure 9 and Figure 10.

Parameters: uint32 number: The hexadecimal number to display. It is the responsibility of the user to ensure that the display has enough digits to accurately represent the number passed to this function. If not, the least significant digits will be displayed.

uint8 position: Position of the Display/Common to start number.

uint8 digits: The number of digits to display the value in. If the digits extend beyond the configured number of commons, then these digits are discarded.

uint8 alignment: How to align the provided number in the allocated digits.

Value	Description
LED_Driver_RIGHT_ALIGN	Least significant digit occupies the rightmost digit (position + digits). Unused digits are turned off.
LED_Driver_LEFT_ALIGN	Most significant digit occupies the digit specified by position. Unused digits are turned off.
LED_Driver_ZERO_PAD	Unused digits to the left are padded with leading zeros.

Return Value: None

Side Effects: None

void LED_Driver_WriteString7Seg(char8 const character[], uint8 position)

Description: Displays a 7-segment null terminated string starting at “position” and ending at either the end of the string or the end of the configured number of commons. See the Functional Description section for the displayable characters. Non-displayable characters will produce a blank space. Note that the positions of the digits are continuous and it is up to the user to choose the correct position for the application. For more information, see Figure 9 and Figure 10.

Parameters: char8 const character[]: The null terminated string to be displayed.

uint8 position: The position to start the string.

Return Value: None

Side Effects: None

void LED_Driver_PutChar7Seg(char8 character, uint8 position)

Description: Displays a 7-segment ASCII encoded character at “position”. This function can display all alphanumeric characters. The function can also display ‘-‘, ‘.’, ‘_’, ‘ ‘, and ‘=’. All unknown characters are displayed as a space. Note that the positions of the digits are continuous and it is up to the user to choose the correct position for the application. For more information, see Figure 9 and Figure 10.

Parameters: char8 character: ASCII character

uint8 position: The position of the character

Return Value: None

Side Effects: None

void LED_Driver_Write7SegDigDec(uint8 digit, uint8 position)

Description: Displays a single 7-segment digit on the specified display. The number in ‘digit’ (0 – 9) is placed at “position.” Note that the positions of the digits are continuous and it is up to the user to choose the correct position for the application. For more information, see Figure 9 and Figure 10.

Parameters: uint8 digit: A number between 0 and 9 to display.

uint8 position: The position of the digit.

Return Value: None

Side Effects: None

void LED_Driver_Write7SegDigHex(uint8 digit, uint8 position)

Description: Displays a single 7-segment digit on the specified display. The number in ‘digit’ (0 – F) is placed at “position.” Note that the positions of the digits are continuous and it is up to the user to choose the correct position for the application. For more information, see Figure 9 and Figure 10.

Parameters: uint8 digit: A number between 0x0 and 0xF (0 to 15)

uint8 position: The position of the digit.

Return Value: None

Side Effects: None

void LED_Driver_Write14SegNumberDec(int32 number, uint8 position, uint8 digits, uint8 alignment)

Description: Displays a 14-segment signed integer up to 8 characters long, starting at “position” and extending for “digits” characters. The negative sign will consume one digit if it is required. If the number exceeds the digits specified, the least significant digits will be displayed. For example, if number is -1234, position is 0 and digit is 4, the result will be: -234.

Parameters: int32 number: a signed integer number to display. It is the responsibility of the user to ensure that the display has enough digits to accurately represent the number passed to this function. If not, the least significant digits will be displayed. Also note that a negative number will require 1 more digit than the equivalent positive number to display the negative sign.

uint8 position: Digit position of the Display/Common RAM to start.

uint8 digits: The number of digits to display the value in. The negative sign will consume one digit if it is required.

uint8 alignment: How to align the provided number in the allocated digits.

Value	Description
LED_Driver_RIGHT_ALIGN	Least significant digit occupies the rightmost digit (position + digits). Unused digits are turned off.
LED_Driver_LEFT_ALIGN	Most significant digit (or negative sign) occupies the digit specified by position. Unused digits are turned off.
LED_Driver_ZERO_PAD	Unused digits to the left are padded with leading zeros.

Return Value: None

Side Effects: None

void LED_Driver_Write14SegNumberHex(uint32 number, uint8 position, uint8 digits, uint8 alignment)

Description: Displays a 14-segment hexadecimal number up to 8 characters long, starting at “position” and extending for “digits” characters. If the number exceeds the digits specified, the least significant digits will be displayed. For example, if number is 0xDEADBEEF, position is 0 and digits is 4, the result will be: BEEF.

Parameters: uint32 number: The hexadecimal number to display. It is the responsibility of the user to ensure that the display has enough digits to accurately represent the number passed to this function. If not, the least significant digits will be displayed.

uint8 position: Position of the Display/Common to start number.

uint8 digits: The number of digits to display the value in.

uint8 alignment: How to align the provided number in the allocated digits.

Value	Description
LED_Driver_RIGHT_ALIGN	Least significant digit occupies the rightmost digit (position + digits). Unused digits are turned off.
LED_Driver_LEFT_ALIGN	Most significant digit occupies the digit specified by position. Unused digits are turned off.
LED_Driver_ZERO_PAD	Unused digits to the left are padded with leading zeros.

Return Value: None

Side Effects: None

void LED_Driver_WriteString14Seg(char8 const character[], uint8 position)

Description: Displays a 14-segment null terminated string starting at “position” and ending at either the end of the string or the end of the display. See the Function Description section for the displayable characters. Non-displayable characters will produce a blank space.

Parameters: char8 const character[]: The null terminated string to be displayed.

uint8 position: The position to start the string.

Return Value: None

Side Effects: None

void LED_Driver_PutChar14Seg(char8 character, uint8 position)

Description: Displays a 14-segment ASCII encoded character at “position”. This function can display all alphanumeric characters. The function can also display ‘-‘, ‘.’, ‘_’, ‘ ‘, and ‘=’. All unknown characters are displayed as a space.

Parameters: uint8 character: ASCII character

uint8 position: The position of the character

Return Value: None

Side Effects: None

void LED_Driver_Write14SegDigDec(uint8 digit, uint8 position)

Description: Displays a single 14-segment digit on the specified display. The number in ‘digit’ (0 – 9) is placed at “position.”

Parameters: uint8 digit: A number between 0 and 9 to display.

uint8 position: The position of the digit.

Return Value: None

Side Effects: None

void LED_Driver_Write14SegDigHex(uint8 digit, uint8 position)

Description: Displays a single 14-segment digit on the specified display. The number in ‘digit’ (0 – F) is placed at “position.”

Parameters: uint8 digit: A number between 0x0 and 0xF (0 to 15)

uint8 position: The position of the digit.

Return Value: None

Side Effects: None

void LED_Driver_Write16SegNumberDec(int32 number, uint8 position, uint8 digits, uint8 alignment)

Description: Displays a 16-segment signed integer up to 8 characters long, starting at “position” and extending for “digits” characters. The negative sign will consume one digit if it is required. If the number exceeds the digits specified, the least significant digits will be displayed. For example, if number is -1234, position is 0 and digits is 4, the result will be: -234.

Parameters: int32 number: a signed integer number to display. It is the responsibility of the user to ensure that the display has enough digits to accurately represent the number passed to this function. If not, the least significant digits will be displayed. Also note that a negative number will require 1 more digit than the equivalent positive number to display the negative sign.

uint8 position: Digit position of the Display/Common RAM to start.

uint8 digits: The number of digits to display the value in.

uint8 alignment: How to align the provided number in the allocated digits.

Value	Description
LED_Driver_RIGHT_ALIGN	Least significant digit occupies the rightmost digit (position + digits). Unused digits are turned off.
LED_Driver_LEFT_ALIGN	Most significant digit (or negative sign) occupies the digit specified by position. Unused digits are turned off.
LED_Driver_ZERO_PAD	Unused digits to the left are padded with leading zeros.

Return Value: None

Side Effects: None

void LED_Driver_Write16SegNumberHex(uint32 number, uint8 position, uint8 digits, uint8 alignment)

Description: Displays a 16-segment hexadecimal number up to 8 characters long, starting at “position” and extending for “digits” characters. If the number exceeds the digits specified, the least significant digits will be displayed. For example, if number is 0xDEADBEEF, position is 0 and digits is 4, the result will be: BEEF.

Parameters: uint32 number: The hexadecimal number to display. It is the responsibility of the user to ensure that the display has enough digits to accurately represent the number passed to this function. If not, the least significant digits will be displayed.

uint8 position: Position of the Display/Common to start number.

uint8 digits: The number of digits to display the value in.

uint8 alignment: How to align the provided number in the allocated digits.

Value	Description
LED_Driver_RIGHT_ALIGN	Least significant digit occupies the rightmost digit (position + digits). Unused digits are turned off.
LED_Driver_LEFT_ALIGN	Most significant digit occupies the digit specified by position. Unused digits are turned off.
LED_Driver_ZERO_PAD	Unused digits to the left are padded with leading zeros.

Return Value: None

Side Effects: None

void LED_Driver_WriteString16Seg(char8 const character[], uint8 position)

Description: Displays a 16-segment null terminated string starting at “position” and ending at either the end of the string or the end of the display. See the Function Description section for the displayable characters. Non-displayable characters will produce a blank space.

Parameters: char8 const character[]: The null terminated string to be displayed.

uint8 position: The position to start the string.

Return Value: None

Side Effects: None

void LED_Driver_PutChar16Seg(char8 character, uint8 position)

Description: Displays a 16-segment ASCII encoded character at “position”. This function can display all alphanumeric characters. The function can also display ‘-‘, ‘.’, ‘_’, ‘ ‘, and ‘=’. All unknown characters are displayed as a space.

Parameters: char8 character: ASCII character

uint8 position: The position of the character

Return Value: None

Side Effects: None

void LED_Driver_Write16SegDigDec(uint8 digit, uint8 position)

Description: Displays a single 16-segment digit on the specified display. The number in ‘digit’ (0 – 9) is placed at “position.”

Parameters: uint8 digit: A number between 0 and 9 to display.

uint8 position: The position of the digit.

Return Value: None

Side Effects: None

void LED_Driver_Write16SegDigHex(uint8 digit, uint8 position)

Description: Displays a single 16-segment digit on the specified display. The number in ‘digit’ (0 – F) is placed at “position.”

Parameters: uint8 digit: A number between 0x0 and 0xF (0 to 15)

uint8 position: The position of the digit.

Return Value: None

Side Effects: None

void LED_Driver_PutDecimalPoint(uint8 dp, uint8 position)

Description: Sets or clears the decimal point at the specified position.

Parameters: uint8 dp: If the value is > 0 the decimal point will be set, if zero, the decimal point will be cleared.

uint8 position: The position at which to adjust the decimal point.

Return Value: None

Side Effects: None

uint8 LED_Driver_GetDecimalPoint(uint8 position)

Description: Returns zero if the decimal point is not set and one if the decimal point is set.

Parameters: uint8 position: The position in which to query the decimal point value.

Return Value: Returns the current state of the decimal point (segment #7 on a 7-seg display). '1' indicates the decimal point is on. '0' indicates the decimal point is off.

Side Effects: None

uint8 LED_Driver_EncodeNumber7Seg(uint8 number)

Description: Converts the lower 4 bits of the input into 7-segment data that will display the number in hex on a display. The returned data can be written directly into the display RAM to display the desired number. It is not necessary to use this function since higher level API are provided to both decode the value and write it to the display RAM.

Parameters: uint8 number A number between 0x0 and 0xF to be converted into segment data.

Return Value: The value to be written into the display RAM for displaying the specific number.

Side Effects: None

uint8 LED_Driver_EncodeChar7Seg(char8 input)

Description: Converts the ASCII encoded alphabet character input into the 7-segment data that will display the alphabet character on a display. The returned data can be written directly into the display RAM to display the desired number. It is not necessary to use this function since higher level API are provided to both decode the value and write it to the display RAM.

Parameters: char8 input: An ASCII alphabet character to be converted into segment data.

Return Value: The value to be written into the display RAM for displaying the specified character.

Side Effects: None

uint16 LED_Driver_EncodeNumber14Seg(uint8 number)

Description: Converts the lower 4 bits of the input into 14-segment data that will display the number in hex on a display. The returned data can be written directly into the display RAM to display the desired number. It is not necessary to use this function since higher level API are provided to both decode the value and write it to the display RAM.

Parameters: uint8 number: A number between 0x0 and 0xF to be converted into segment data.

Return Value: The value to be written into the display RAM for displaying the specific number.

Side Effects: None

uint16 LED_Driver_EncodeChar14Seg(char8 input)

Description: Converts the ASCII encoded alphabet character input into the 14-segment data that will display the alphabet character on a display. The returned data can be written directly into the display RAM to display the desired number. It is not necessary to use this function since higher level API are provided to both decode the value and write it to the display RAM.

Parameters: char8 input: An ASCII alphabet character to be converted into segment data.

Return Value: The value to be written into the display RAM for displaying the specified character.

Side Effects: None

uint16 LED_Driver_EncodeNumber16Seg(uint8 number)

Description: Converts the lower 4 bits of the input into 16-segment data that will display the number in hex on a display. The returned data can be written directly into the display RAM to display the desired number. It is not necessary to use this function since higher level API are provided to both decode the value and write it to the display RAM.

Parameters: uint8 number: A number between 0x0 and 0xF to be converted into segment data.

Return Value: The value to be written into the display RAM for displaying the specific number.

Side Effects: None

uint16 LED_Driver_EncodeChar16Seg(char8 input)

Description: Converts the ASCII encoded alphabet character input into the 16-segment data that will display the alphabet character on a display. The returned data can be written directly into the display RAM to display the desired number. It is not necessary to use this function since higher level API are provided to both decode the value and write it to the display RAM.

Parameters: char8 input: An ASCII alphabet character to be converted into segment data.

Return Value: The value to be written into the display RAM for displaying the specified character.

Side Effects: None

void LED_Driver_SetBrightness(uint8 bright, uint8 position)

Description: Sets the desired brightness value (0 = display off; 255 = display at full brightness) for the chosen display by applying a PWM duty cycle to the common when the display is active.

Parameters: uint8 bright: The brightness value by duty cycle, between 0 and 255.

uint8 position: The position in which to set the brightness.

Return Value: None

Side Effects: None

uint8 LED_Driver_GetBrightness(uint8 position)

Description: Returns the brightness value for the specific display location.

Parameters: uint8 position: Position in which to return the brightness value.

Return Value: None

Side Effects: None

void LED_Driver_Sleep(void)

Description: Prepares the component for sleep. If the component is currently enabled it will be disabled and reenabled by LED_Driver_Wakeup().

Parameters: None

Return Value: None

Side Effects: None

void LED_Driver_Wakeup(void)

Description: Returns the component to its state before the call to LED_Driver_Sleep().

Parameters: None

Return Value: None

Side Effects: None

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

The LED Segment and Matrix Driver component is used to drive 7-segment, 14-segment, or 16-segment displays or an LED matrix display using PSoC 3 or PSoC 5LP. The segment and common pins of the component are connected to the GPIOs via digital pins, and the external LED displays can thus be controlled using PSoC. The following information contains the pin assignment of the displays, and the various modes of configuration that are available for this component.

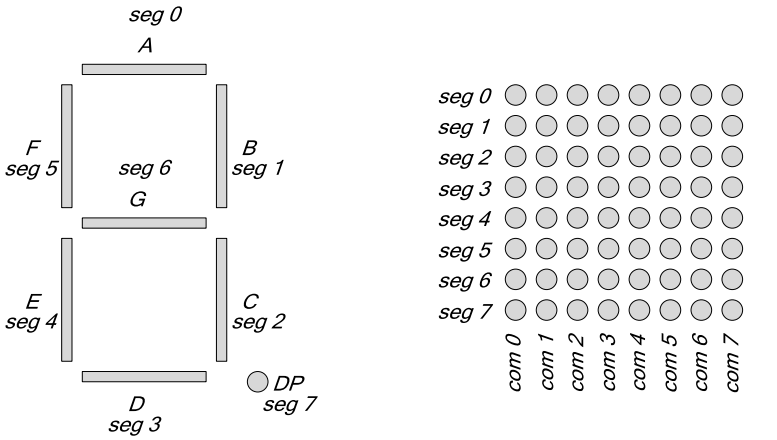
Pin Assignment

The component assumes the following pin assignment for displaying the data on the LEDs. The supported symbols for these displays are also shown here.

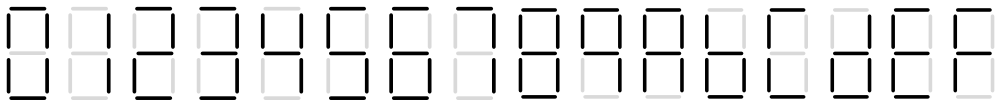
7-Segment and Matrix Array

For 7-segment LEDs, the segments are arranged as shown on the left of Figure 5. The right figure shows the pin assignment for a sample 8x8 LED array.

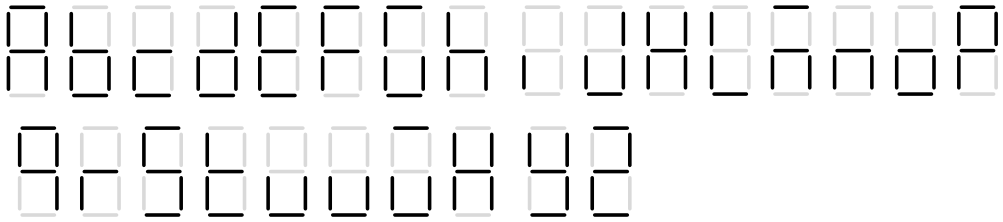
Figure 5. 7-Segment and Matrix Array



When decoding numerical digits the following patterns are used for 7-segment displays:

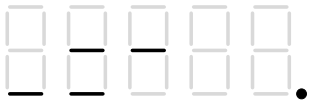


When decoding letters between “A” and “Z” the following patterns are used for both upper and lower cases:



Special characters are shown as follows:

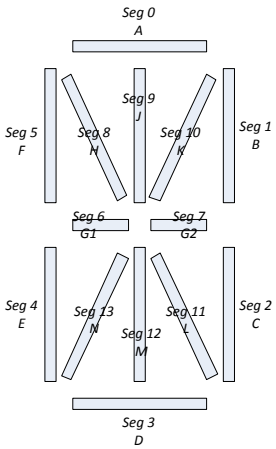
- Underscore “ _ ”
- Equal sign “ = ”
- Negative sign “ - ”
- Space “ ”
- Decimal point “ . ”



14-Segment Display

Figure 6 shows the assumed pin assignment of a 14-segment display.

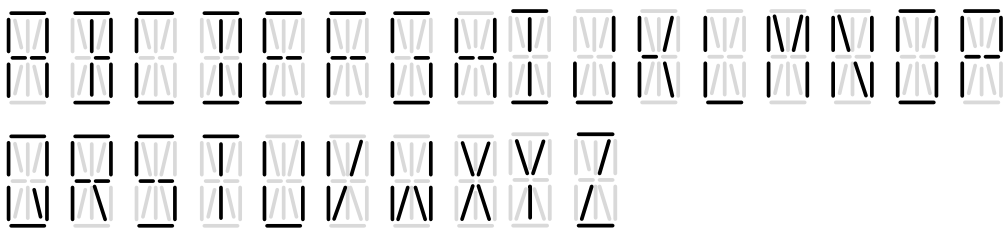
Figure 6. 14-Segment Display



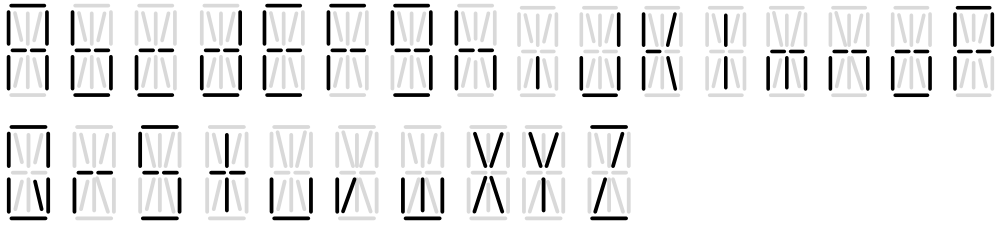
When decoding numerical digits the following patterns are used for 14-segment displays:



When decoding letters between “A” and “Z” the following patterns are used for the upper cases:



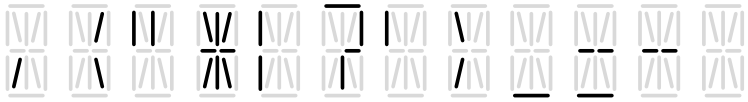
When decoding letters between “a” and “z”, the following patterns are used for the lower cases:



Special characters are shown as follows:

- Comma “ , ”
- Open bracket “ (”
- Double quote “ “ ”
- Star “ * ”
- Exclamation mark “ ! ”
- Question mark “ ? ”
- Single quote “ ‘ ”
- Close bracket “) ”
- Underscore “ _ ”
- Equal sign “ = ”

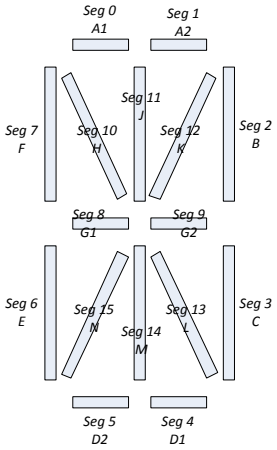
- Negative sign “ - ”
- Space “ ”



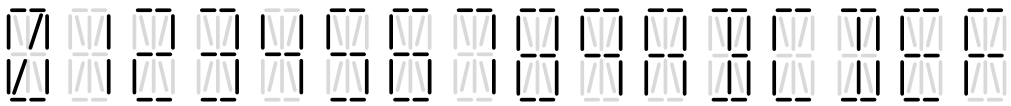
16-Segment Display

Figure 7 shows the assumed pin assignment of a 16-segment display.

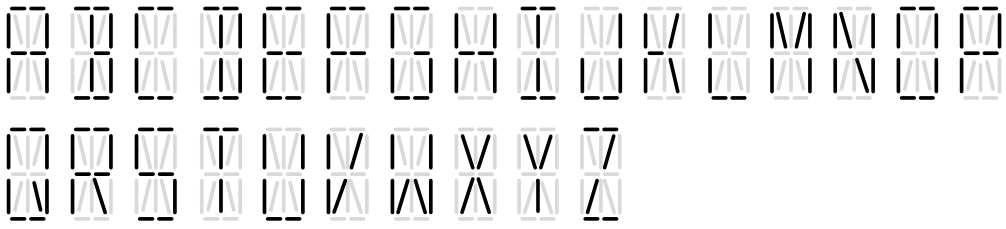
Figure 7. 16-Segment Display



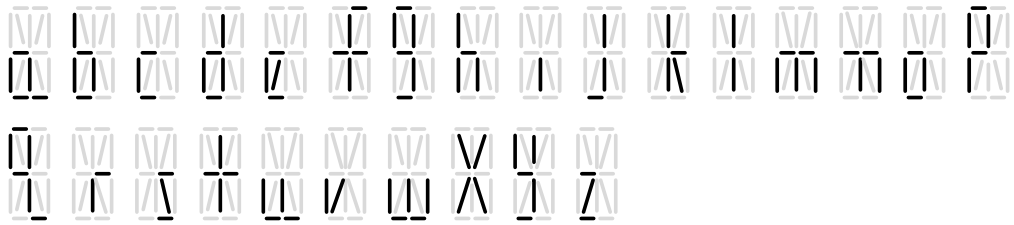
When decoding numerical digits the following patterns are used for 16-segment displays:



When decoding letters between “A” and “Z” the following patterns are used for the upper cases:

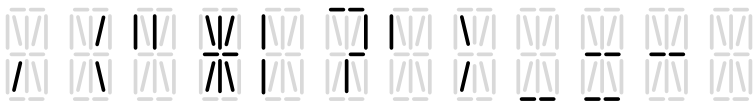


When decoding letters between “a” and “z”, the following patterns are used for the lower cases:



Special characters are shown as follows:

- Comma “ , ”
- Open bracket “ (”
- Double quote “ “ ”
- Star “ * ”
- Exclamation mark “ ! ”
- Question mark “ ? ”
- Single quote “ ‘ ”
- Close bracket “) ”
- Underscore “ _ ”
- Equal sign “ = ”
- Negative sign “ - ”
- Space “ ”



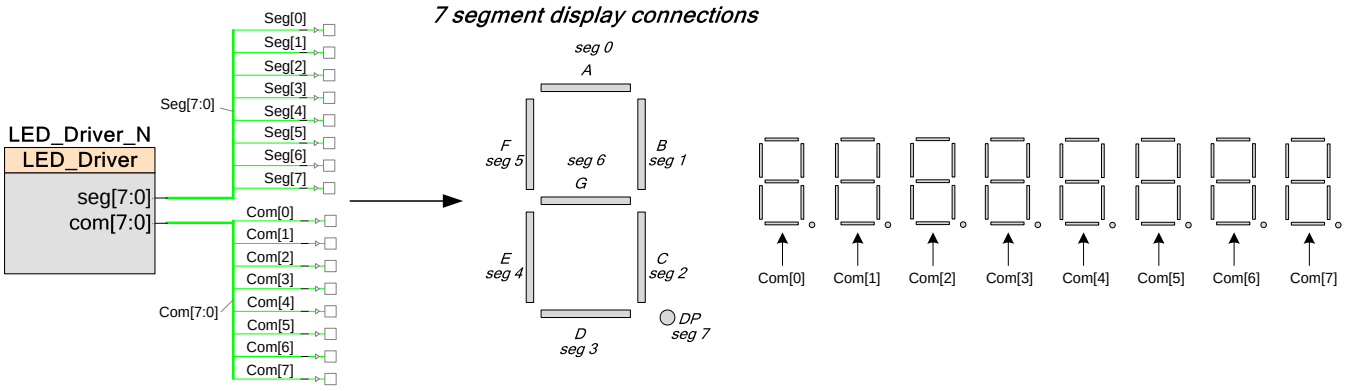
Supported Configurations

The LED Segment and Matrix Driver component supports 7-segment, 14-segment, 16-segment and matrix LEDs. This section describes configurations for each of these modes.

7-segment display

This component is capable of driving up to eight 7-segment displays with eight commons and eight segments. This standard configuration is shown in Figure 8.

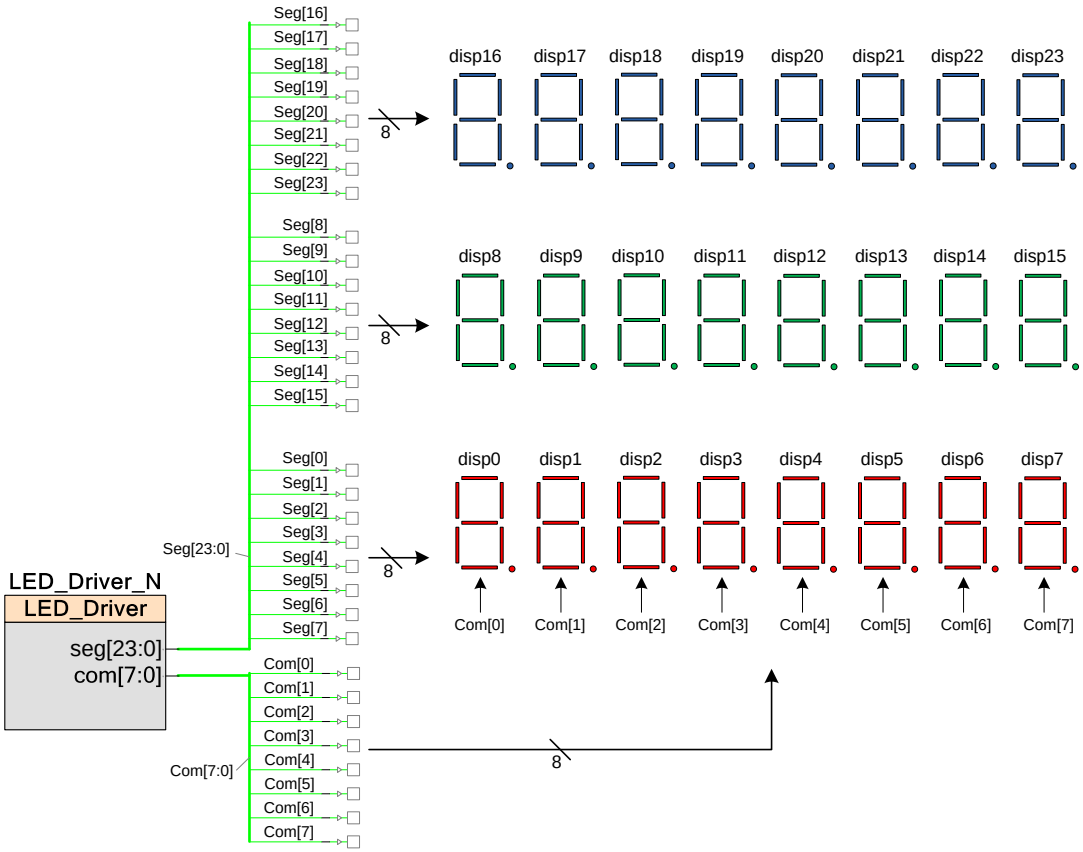
Figure 8. Standard 7-Segment Display Configuration



The component can also display up to x24 7-segment displays by using all 24 segments and 8 commons. This usage case can occur when the user needs to display 24 digits or 8 digit RGB

(Red, Green, Blue) using 7-segment displays. Figure 9 shows the component pin assignment to drive an 8-digit display with RGB 7-segment displays.

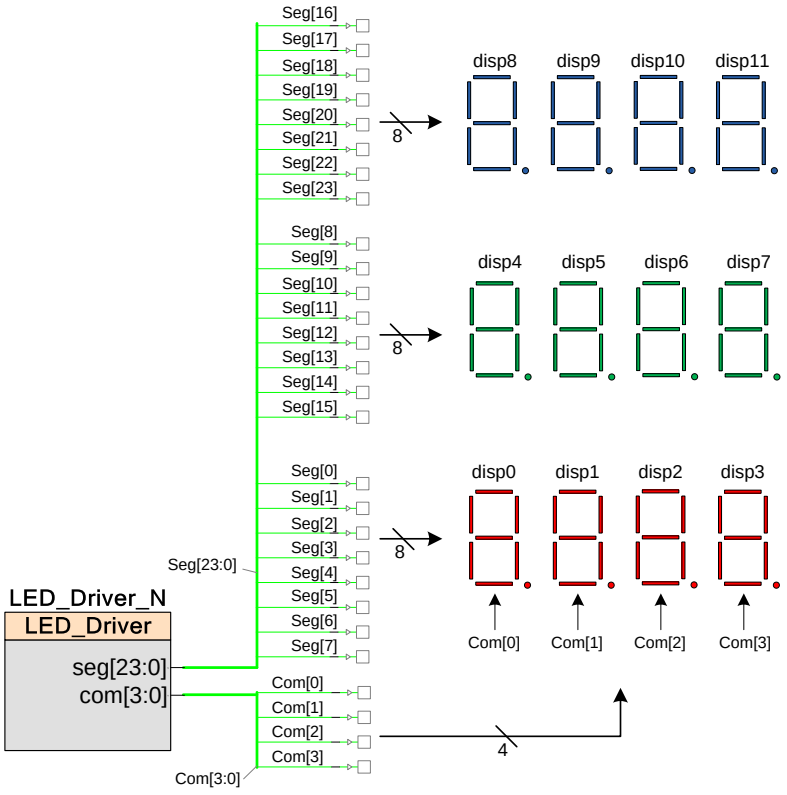
Figure 9. Large 24 Digit Display or 8 Digit RGB Display



If the number of defined commons is less than 8 and the number of segments is larger than 8, then the references to the display positions is shown as continuous. For example, Figure 10

shows a 7-segment display array with 4 commons and 24 segments that drive 12 displays. The numbering of the display is continuous – progressing as disp[0, 1, 2, 3, 4, 5 … 11].

Figure 10. Display Array with 4 Commons and 12 Digits



14-Segment and 16-Segment Displays

The LED Segment and Matrix Driver component is capable of driving up to eight 14-segment displays and eight 16-segment displays. These are illustrated in Figure 11 and Figure 12, respectively.

Figure 11. Standard 14-Segment Display Configuration

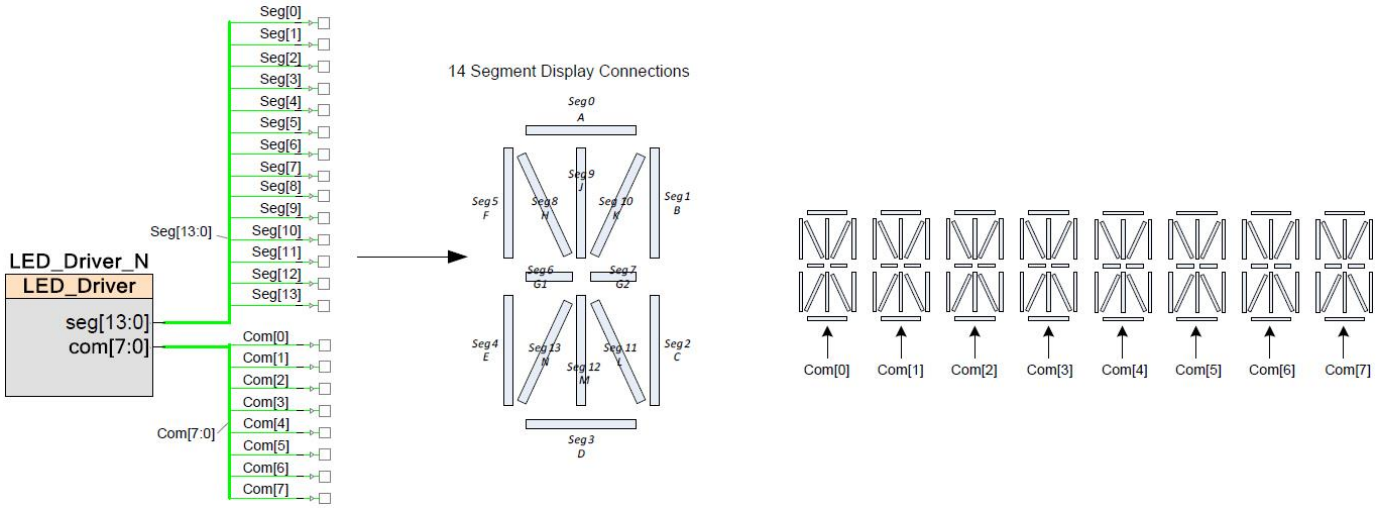
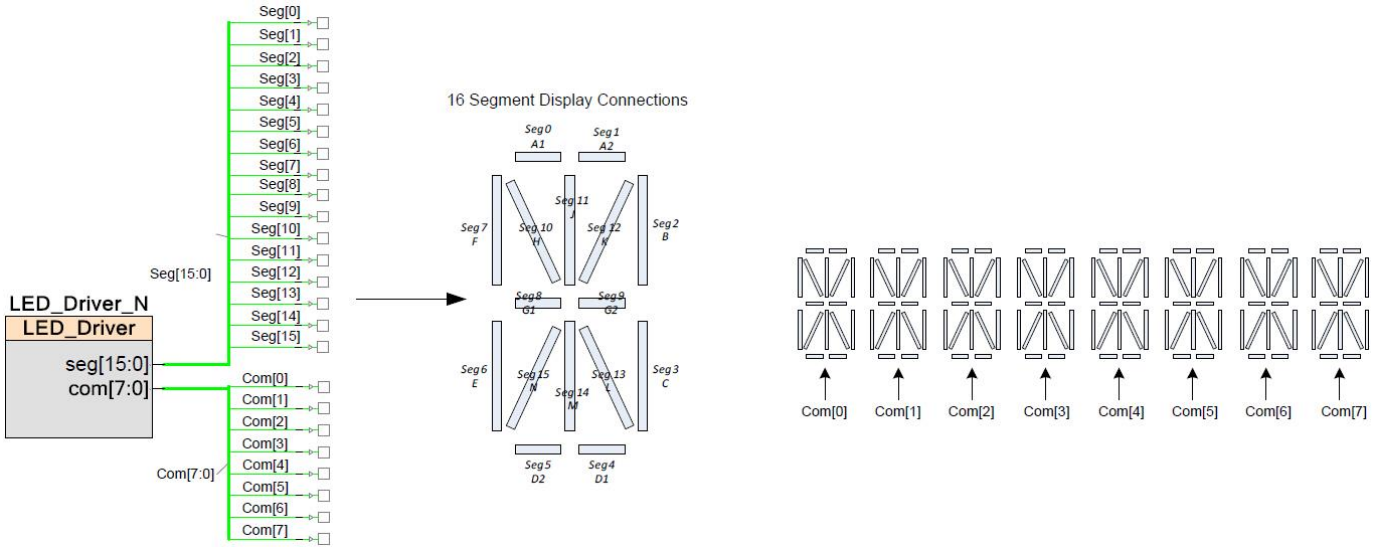


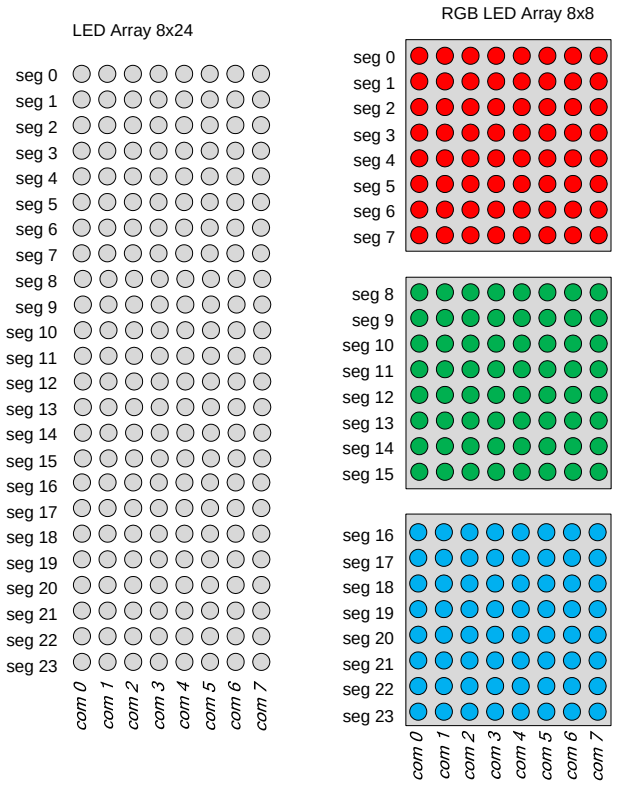
Figure 12. Standard 16-Segment Display Configuration



LED matrix displays

Aside from driving segment displays, this component can also be used to drive matrix LEDs of up to 8x24 LEDs. This can either be an 8x24 array of LEDs as shown on the left of Figure 13 or it can be an 8x8 RGB LED array as shown on the right of Figure 13.

Figure 13. LED Matrix Array Configurations

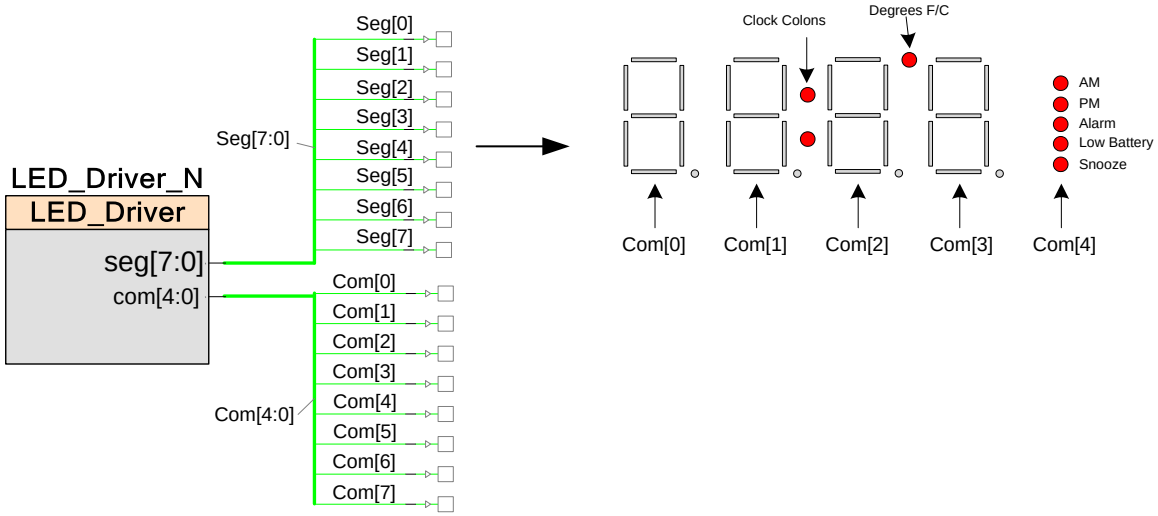


Annunciator displays

The component also supports a combination of encoded 7/14/16 segment displays with random LEDs that may be needed for annunciators. An example of this is shown in Figure 14, where x4 7-segment displays with 8 annunciators are driven with the LED Segment and Matrix Driver

component. The AM, PM, Alarm, etc. annunciators are driven by Com[4], and the digits are displayed using Com[3:0].

Figure 14. Combination of 7-Segment and random LEDs used for annunciators



Segment Registers

There are 3 registers that are used to hold the segment values. Each contain a number of elements equal to the number of commons, and can be directly accessed to modify the display. The table below shows the names of the registers, where “LED_Driver” is the instance name of the component, and “*n*” is the number of enabled commons. If the number of segments is less than 8, then LED_Driver_SegMBuffer and LED_Driver_SegHBuffer are not available. If the number of segments is less than 16, then LED_Driver_SegHBuffer is not available.

Register name	Description
LED_Driver_SegLBuffer[*n*]	Used to hold segment values for segments 0 to 7.
LED_Driver_SegMBuffer[*n*]	Used to hold segment values for segments 8 to 15.
LED_Driver_SegHBuffer[*n*]	Used to hold segment values for segments 16 to 23.

Resources

The table below shows the resource requirements for the LED Segment and Matrix Driver component.

Configuration		Resource Type					
		Datapath Cells	Macro-cells	Status Cells	Control Cells	DMA Channels	Interrupts
Brightness control disabled	1-8 segments	0	1	-	2	2	-
	9-16 segments	0	1	-	3	2	-
	17-24 segments	0	1	-	4	2	-
Brightness control enabled	1-8 segments	1	7	-	3	3	-
	9-16 segments	1	7	-	4	3	-
	17-24 segments	1	7	-	5	3	-

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 associated compiler configured in Release mode with optimization set for Size. For a specific design the map file generated by the compiler can be analyzed to determine the memory usage.

Configuration		PSoC 3 (Keil_PK51)		PSoC 5LP (GCC)	
		Flash Bytes	SRAM Bytes	Flash Bytes	SRAM Bytes
Brightness control disabled	1-8 segments	5924	64	4560	73
	9-16 segments	7375	88	5164	101
	17-24 segments	8178	112	5572	125
Brightness control enabled	1-8 segments	6284	75	4844	82
	9-16 segments	7655	99	5440	110
	17-24 segments	8458	123	5848	134

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 LED Segment and Matrix Driver component does not have any specific deviations.

This component has the following embedded component - DMA. Refer to the corresponding component datasheet for information on their MISRA compliance and specific deviations.

DC and AC Electrical Characteristics

Specifications are valid for –40 °C ≤ TA ≤ 85 °C and TJ ≤ 100 °C, except where noted. Specifications are valid for 1.71 V to 5.5 V, except where noted.

DC Characteristics

Parameter	Description	Min	Typ[1]	Max	Units[2]
I_{DD}	Component current consumption (Brightness control disabled)				
	8 commons, 8 segments	–	30	–	µA/kHz
	8 commons, 16 segments	–	40	–	µA/kHz
	8 commons, 24 segments	–	50	–	µA/kHz
	Component current consumption (Brightness control enabled)				
	8 commons, 8 segments	–	120	–	µA/kHz
	8 commons, 16 segments	–	145	–	µA/kHz
	8 commons, 24 segments	–	150	–	µA/kHz

[1] Device IO and clock distribution current are not included. The values are at 25 °C.

[2] Current consumption is specified with respect to the refresh rate. More information on the relationship between component clock and the refresh rate can be found in the Timing – Display refresh rate section.

Component Errata

This section lists known problems with the component.

Cypress ID	Component Version	Problem	Workaround
191257	v1.0	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
1.10.b	Minor datasheet edits.	
1.0.a	Edited datasheet to add Component Errata section.	Document that the component was changed, but there is no impact to designs.
1.0	First release	

© 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, 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.