Preliminary

# SmartLED Datasheet SmartLED V 1.10

Copyright © 2011-2013 Cypress Semiconductor Corporation. All Rights Reserved.

## Resource Usage

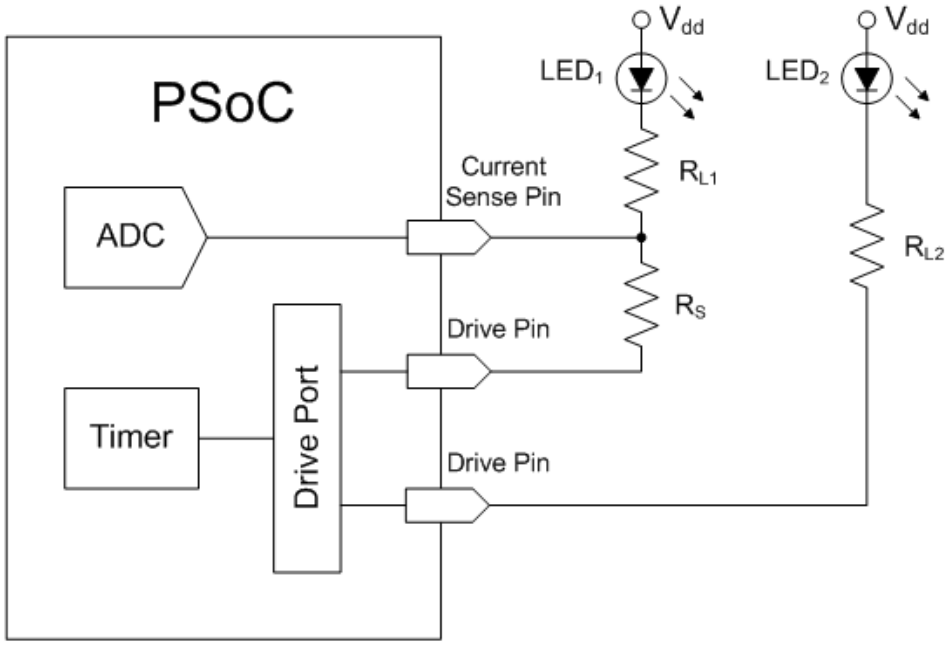
| Resources | PSoC® Blocks | | | | API Memory (Bytes) | | Pins (per External I/O) |
|---|---|---|---|---|---|---|---|
| | CapSense® | Timer | ADC | Comparator | Flash | RAM | |
| CY8C20xx6L | | | | | | | |
| Mixed Mode | | | | | | | |
| PWM Granularity = 5% | - | 1 | 1 | - | 1700 | 34 | - |
| PWM Granularity = 10% | - | 1 | 1 | - | 1700 | 24 | - |
| Each LED in Current control Mode | - | - | - | - | 4 | 1 | 2 |
| Each LED in PWM Mode | - | - | - | - | 4 | 1 | 1 |
| Explicit PWM Duty Cycle Mode | | | | | | | |
| PWM Granularity = 5% | - | 1 | 0 | - | 360 | 29 | - |
| PWM Granularity = 10% | - | 1 | 0 | - | 360 | 19 | - |
| Each LED | - | - | - | - | 1 | 0 | 1 |

## Features and Overview

- Independent PWM control of up to eight LEDs
- PWM duty cycle granularity configurable to either 5 or 10% steps
- Configurable PWM frequency
- 0.4% CPU usage for 60 Hz PMW with 5% duty cycle granularity
- Maximum LED current is 25 mA
- Maximum total current load on the device is 120 mA (60 mA/side) as limited by device specifications
- In quasi closed loop current control mode, each LED requires two GPIO pins, one current limiting, and one current sensing resistor
- In explicit PWM duty cycle mode only one GPIO pin and a current limiting resistor are required
- Wizard for quick and easy LEDs configuration

The SmartLED User Module offers a solution of LEDs brightness control without the need for external active components.

Figure 1. SmartLED User Module Block Diagram




## Functional Description

The SmartLED User Module implements a “soft” PWM drive that provides the LED brightness/current control for up to eight independent LEDs. Configurable PWM duty cycle enables you to implement visual effects such as LED breathing.

The brightness of each LED can be controlled in two modes:

- Quasi closed loop current control
- Explicit PWM duty cycle

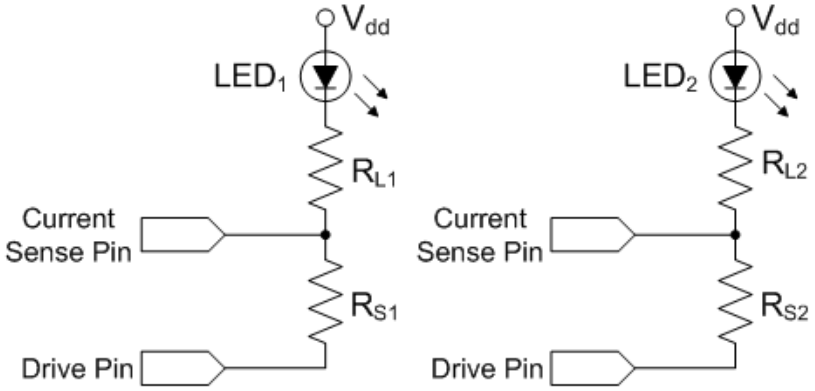
### Quasi Closed Loop Current Control Mode

In this mode, the SmartLED User Module uses Programmable Timer and ADC blocks.

The Programmable Timer is used for PWM drive interrupt generation and is not available for other purposes.

The ADC block is used to sample the 100% LED current (at any other time the ADC block is available for other purposes). This information is used to determine the PWM modulation duty cycle needed to achieve the brightness corresponding to the current specified as an argument to the API. This mitigates the effects of component-to-component variations and component deterioration over life cycle. The maximum LED current 25 mA and this parameter is configurable. Quasi closed loop control mode does not provide true closed loop control, so runtime perturbations such as power supply fluctuations are not compensated. In this mode, SmartLED requires two GPIO pins and two external resistors for every LED as shown in Figure 2.

Figure 2. SmartLED User Module in "Quasi closed loop current control" Mode Schematic

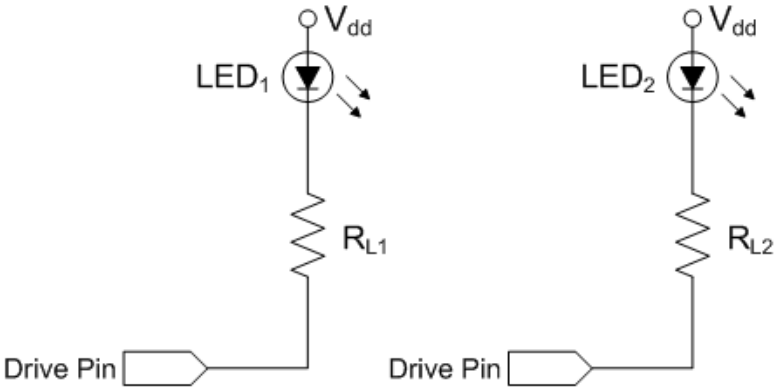



## Explicit PWM Duty Cycle Mode

In this mode the SmartLED User Module occupies only the Programmable Timer block. The Programmable Timer is used for PWM drive interrupt generation and is not available for other purposes.

In the Explicit PWM duty cycle mode, the SmartLED User Module sets the specified PWM duty cycle with no regard for the actual current delivered. In this mode the SmartLED User Module requires one GPIO pin and a current limiting resistor as shown in Figure 3.

Figure 3. SmartLED User Module in "Explicit PWM Duty Cycle" Mode Schematic




The SmartLED User Module has two multi user module (MUM) configurations:

- Mixed mode
- Explicit PWM duty cycle mode

**Note** Depending on the selected MUM configuration, each LED brightness can be controlled using two different LED modes.

In the “Mixed mode” MUM configuration, the LED brightness can be set using both the Current control and PWM duty cycle modes.

In the “Explicit PWM duty cycle mode” MUM configuration the LED brightness is controlled only by the PWM duty cycle with no regard for the actual current delivered.

### Pins and Routing

The SmartLED requires two GPIO pins and two external resistors for every LED in the quasi closed loop current control mode, as shown in Figure 2. One pin is a dedicated current sensing pin and the other is the Drive Pin. For the Explicit duty cycle mode, the current sensing pin and $R_S$ resistor can be omitted, as shown in Figure 3.

For all LED types $R_S$ should be 38 Ohm. $R_L$ depends on the LED type, white, red, blue, and so on. For white LED designs the LED anodes must be powered by a 5.0V rail. $V_{DD}$ of the PSoC device must be powered by the same voltage level supplied to the LED anodes.

## Placement

The SmartLED User Module in the Mixed mode occupies the Programmable Timer and ADC blocks. In the Explicit PWM Duty Cycle mode it occupies only the Programmable Timer. The user module is restricted to only one instance in a project.

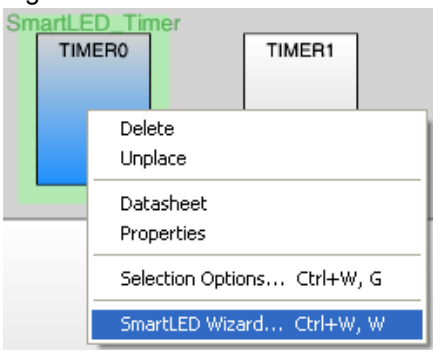
The user module maps onto one PSoC block designated SmartLED_Timer in the PSoC Designer™ Chip Editor. The SmartLED_Timer block is used only for the PWM drive interrupt generation and is not available for other purposes. The ADC is used only to sample the 100% LED current when it is turned on in the quasi closed loop current control mode. Any other time the ADC is available for other purposes.

## Wizard

The SmartLED User Module Wizard is used to configure parameters that apply globally and to each individual LED.

To access the Wizard, right-click the SmartLED User Module in the Workspace Explorer or the Chip Editor and select the SmartLED Wizard (Figure 4).

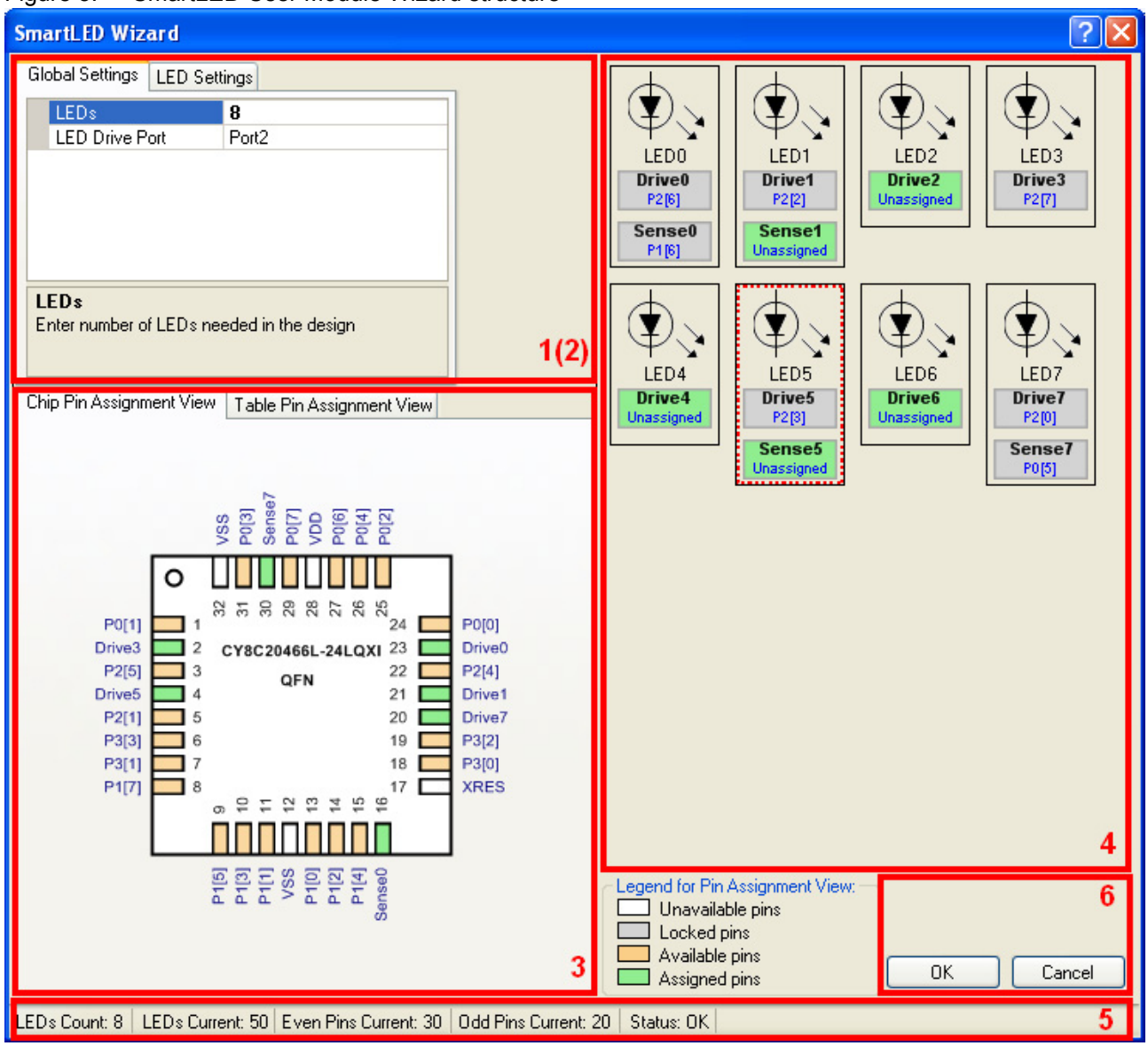
Figure 4. SmartLED User Module Wizard selection




The SmartLED User Module Wizard consists of the following parts (Figure 5):

1. Global Settings tab
2. LED Settings tab
3. Pin Assignment section
4. LEDs Representation section
5. Status Bar
6. Wizard Buttons

Figure 5. SmartLED User Module Wizard structure




## Global Settings Tab

The Global Settings tab of the SmartLED User Module wizard is shown in Figure 6. This tab is used to configure global user module parameters, such as number of LEDs and LED Drive Port.

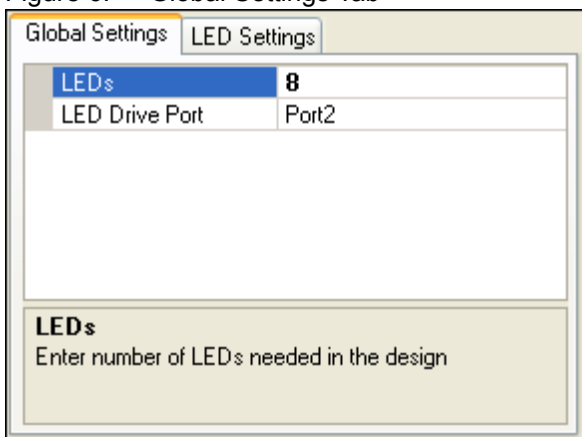
### LEDs

This parameter enables you to set the number of LEDs used in a design.

### LED Drive Port

This parameter enables you to set the Drive Port for LEDs used in a design.

Figure 6. Global Settings Tab




## LED Settings Tab

The LED Settings tab of the SmartLED User Module wizard is shown in Figure 7. This tab is used to configure the Mode and LED Current parameters.

**Note** These parameters are available when you select the “Mixed mode” MUM configuration. In the “Explicit PWM duty cycle mode” MUM configuration, these parameters are not applicable.

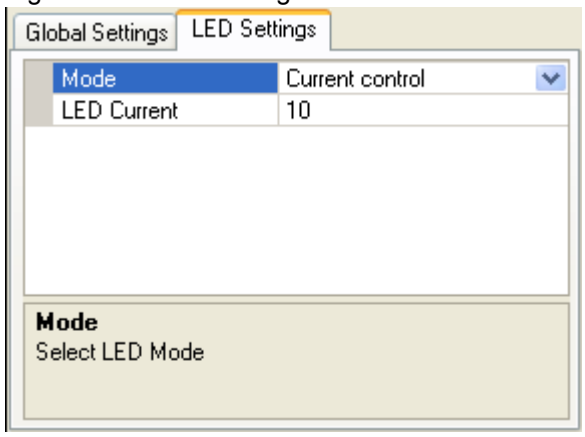
### Mode

This parameter enables you to set the LED Mode for each LED in a design. To set “Mode” for a specific LED, select it (click on LED view in the LEDs representation section).

### LED Current

This parameter enables you to set the LED current for each LED in a design. To set a current for a specific LED, select it (click on LED view in the LEDs representation section). Now you can enter the required LED current value.

Figure 7. LED Settings Tab



## Pin Assignment section

This section is used for LEDs pins assignment using the drag-and-drop mechanism.

The pins available for assignment are highlighted, while dragging the LED pin, and vary depending on the type of the chosen LED pin (Drive Pin/Sense Pin) and LED Drive Port parameter setting. You can place the LED “Drive Pin” only on pins of the selected “LED Drive Port”. The pins appropriate for the LED “Drive Pin” are highlighted. There is no limit for the LED “Sense Pin” assignment.

The user module wizard checks for the pins occupied by other user modules in the project and restricts you from assigning the LED pins on them.

The Legend for Pin Assignment View gives you a reference on the possible pin states (Figure 8).

Figure 8. Pin Assignment Legend



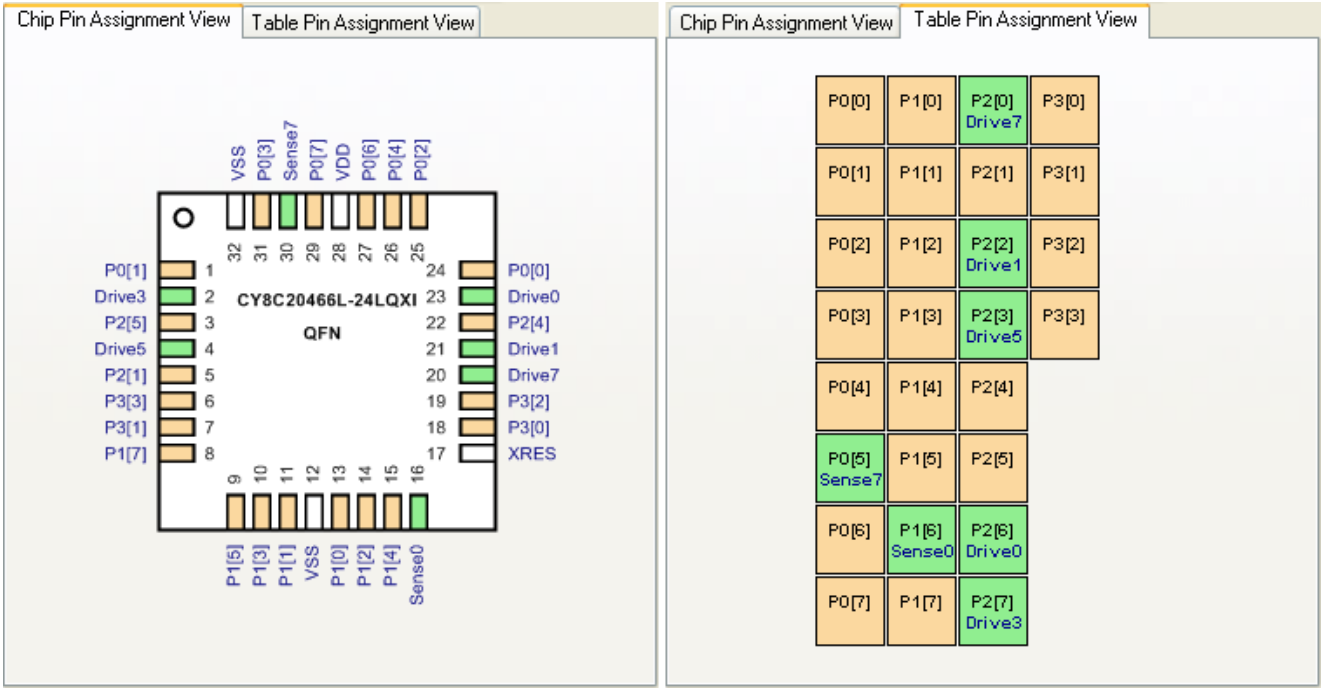

The Pin Assignment section has two tabs (Figure 9):

- Chip Pin Assignment View
- Table Pin Assignment View

The Chip Pin Assignment View reflects the part number corresponding to the one selected by you during the project creation.

The Table Pin Assignment View gives you the port ordered table that represents an assignment state of the chip pins.

Figure 9. Pin Assignment section



## LEDs Representation Section

This section graphically represents LEDs, available in the project, on the Wizard form, and enables you to configure each LED independently. Each LED representation consists of the LED icon with the specific LED name, and LED pins controls:

**Drive Pin**

This pin control is displayed as shown in Figure 10 (Drive0, Drive1, Drive2, etc.). The pin supports the drag-and-drop operation on the Pin Assignment View. The Port Pin number for this pin control is displayed when the pin is already assigned.

This pin is available in the both SmartLED MUM configurations.

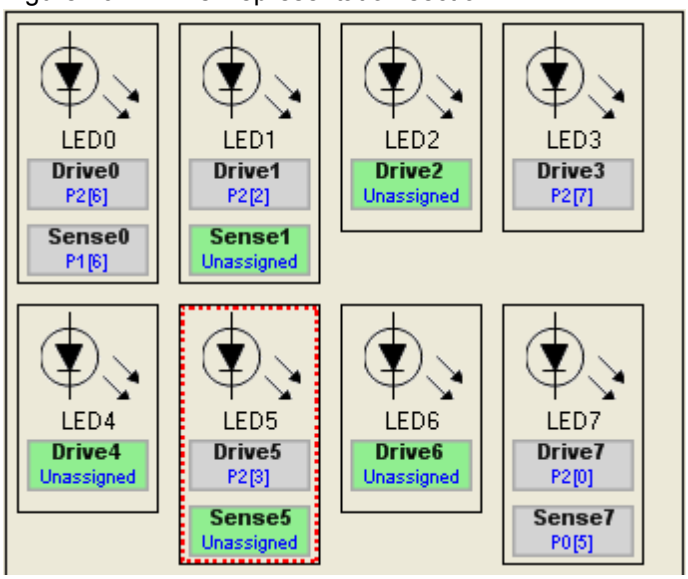
**Sense Pin**

This pin control is displayed as shown in Figure 10 (Sense0, Sense1, Sense5, and Sense7). The pin supports drag-and-drop operation on the Pin Assignment View. The Port Pin number for this pin control is displayed when the pin is already assigned.

This pin is available when you select the “Mixed mode” MUM configuration and the LED “Mode” parameter is set to “Current control”. In other cases the “Sense Pin” control is disabled.

The pin controls for the assigned LED pins are highlighted in grey color. The LED control is set in a red dashed frame when the LED is selected.

Figure 10. LEDs Representation section




## Status Bar

The status bar displays common information about the design:

- LEDs Count – displays the number of LEDs used in the design
- LEDs Current – displays the current consumption for all the LEDs in the project
- Even Pins Current – displays the overall LEDs current consumption for the even pins
- Odd Pins Current – displays the overall LEDs current consumption for the odd pins

- Status – displays warning/error messages. If the status is incorrect then the “OK” button on the main Wizard view is disabled; for example, the total LED current is too large.

Figure 11. Status Bar




### Wizard Buttons

The SmartLED User Module wizard offers four buttons with the predefined functionality.

1. “OK” – this button checks if the wizard parameters are correct and all LEDs pins are assigned. If yes, then the wizard saves parameters and closes. If not, the wizard shows an appropriate warning message, does not save the parameters, and stays opened.
2. “Cancel” – this button closes the wizard without saving any parameters.
3. “Close” – a standard window close button located on the title bar, at the top right corner of the wizard form. If you click on the close button then the wizard closes without saving any parameters.
4. “Help” – this button calls the help page giving a reference information on how to use the SmartLED User Module wizard. It briefly describes the SmartLED User Module wizard features. The Help is available via a standard window help button, labeled with the question mark, and located on the title bar, at the top right corner of the wizard form.

## Parameters and Resources

Most of the properties and parameters of this user module are accessed through the user module wizard. However, the following user module properties are accessed in the Parameters window of PSoC Designer.

### PWM Period

This parameter is used to set up the initial PWM period value. The allowed values range from 1 to 65535. The default value is 19999.

### PWM Granularity

This parameter sets the PWM Granularity value. It defines the number of PWM steps. The valid values are 5% and 10%. The default value is 5%.

A selection of 5% assigns a value of 20 PWM steps and a selection of 10% assigns a value of 10 PWM steps.

### Interrupt Generation Control

The two parameters InterruptAPI and IntDispatchMode are accessed only by setting the Enable Interrupt Generation Control check box in PSoC Designer. This is available under **Project > Settings ... > Chip Editor**.

#### InterruptAPI

The InterruptAPI parameter allows conditional generation of a user module’s interrupt handler and interrupt vector table entry. Select “Enable” to generate the interrupt handler and interrupt vector table entry. Select “Disable” to bypass the generation of the interrupt handler and interrupt vector table entry. If the Receive Command Buffer is to be used, then the InterruptAPI parameter should be set to “Enable”. Properly selecting whether an Interrupt API is to be generated is recommended particularly with projects that have multiple overlays where a single block resource is used by the different over-

lays. By selecting Interrupt API generation only when it is necessary, the need to generate an interrupt dispatch code might be eliminated, thereby reducing the overhead.

**IntDispatchMode**

The IntDispatchMode parameter is used to specify how an interrupt request is handled for interrupts shared by multiple user modules existing in the same block but in different overlays. Selecting “ActiveStatus” causes firmware to test which overlay is active before servicing the shared interrupt request. This test occurs every time the shared interrupt is requested. This adds latency and also produces a nondeterministic procedure of servicing shared interrupt requests, but does not require any RAM. Selecting “OffsetPreCalc” causes firmware to calculate the source of a shared interrupt request only when an overlay is initially loaded. This calculation decreases interrupt latency and produces a deterministic procedure for servicing shared interrupt requests, but at the expense of a byte of RAM.

## Application Programming Interface

The following two paragraphs and note should be altered as appropriate for the user module and included as an introduction to the API section.

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

Each time a user module is placed, it is assigned an instance name. By default, PSoC Designer assigns UMName_1 to the first instance of this user module in a given project. This name can be changed to any unique value that follows the syntactic rules for identifiers. The assigned instance name becomes the prefix of every global function name, variable and constant symbol. In the following descriptions the instance name has been shortened to UMName for simplicity.

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

### Global Arrays and Variables

API functions use different global variables. The following is a list of the user module global arrays and variables:

**SmartLED_baPWM[]** - This is a byte array, each element is sequentially substituted into the Drive Port data register by SmartLED_Timer_ISR. Each bit in each element corresponds to the LED with the same index. The API determines the state of each bit such that the sequential substitution of these elements into the Drive Port data register results in the specified duty cycle for each LED. The duty cycle can be specified as a quasi closed loop controlled current value or explicitly depending on which API function is used to turn on the LED. The PWM Granularity user module parameter determines the number of elements in this array and in so doing sets the PWM granularity.

**SmartLED_baLEDCurrent[]** - This is a byte array, each element contains the nominal PWM modulated current that result when the corresponding LED is turned on in the quasi closed loop current control mode by a call to SmartLED_On().

**SmartLED_bPWMIndex** - This byte variable serves as the index for the SmartLED_baPWM array in the Programmable Timer ISR.

**SmartLED_bDrivePortPinMask** - This byte variable is used as a bit mask for the pins in the LED Drive Port to which LEDs are attached. This mask is used to ensure that only those pins assigned as LED Drive Pins are “touched” when the Drive Port data register is substituted with SmartLED_baPWM elements by SmartLED_Timer_ISR.

**SmartLED_bOnMask** - This byte variable is used to keep track of which LEDs in a design are turned on.

**SmartLED_wADCSlope** - This word variable is used to store the ADC sample result of Vref (1.0V). The SmartLED API uses this information to compensate for part-to-part variability when the LED current is sampled in quasi closed loop current control mode.

**SmartLED_bCurrentCoefficient** - This byte variable is used to store a coefficient that converts the raw ADC result into current in mA in SmartLED_On. This variable is initialized by 27 and is based on the 38 Ohm $R_S$. This variable allows for compensation of application induced errors in the measured LED current. This is done by writing a different value to this variable in user code after the call to SmartLED_Start, but before any calls to SmartLED_On.

## SmartLED_Start

**Description:**

This API function initializes the SmartLED User Module. Sets all bits corresponding to attached LEDs in all SmartLED_baPWM elements such that LEDs initialize to a 0% duty cycle (off) state. Populates SmartLED_bDrivePortPinMask from SmartLED_Drive_Pin_Table. Puts all LED Drive Pins to strong drive mode and starts Programmable Timer.

When the user module operates in the Mixed mode the API also performs start of the ADC (with resolution of 10 bits, 8 MHz data clock, and AnalogMux bus input) and samples Vref (1.0V) to obtain SmartLED_wADCSlope. Populates SmartLED_baLEDCurrent from SmartLED_Current_Table.

**C Prototype:**

```
void   SmartLED_Start(void)
```

**Assembly:**

```
lcall  SmartLED_Start
```

**Parameters:**

None

**Return Value:**

None

**Side Effects:**

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

## SmartLED_Stop

**Description:**

This API function shuts down the SmartLED. Restores the Drive Pins to the default High-Z analog drive mode and clears the corresponding GPIO data registers bits. Disables the Programmable Timer interrupt and stops the timer. This API function does not stop the ADC.

**C Prototype:**

```
void  SmartLED_Stop(void)
```

**Assembly:**

```
lcall  SmartLED_Stop
```

**Parameters:**

None

**Return Value:**

None

**Side Effects:**

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

## SmartLED_bGetDrivePin

**Description:**

This API function returns the Drive Pin of the LED corresponding to the index passed as an argument to the function.

**C Prototype:**

```
BYTE   SmartLED_bGetDrivePin(BYTE bLEDIndex)
```

**Assembly:**

```
mov    A, bLEDIndex
lcall  SmartLED_bGetDrivePin
```

**Parameters:**

bLEDIndex – index of LED which the Drive Pin is identified for.

**Return Value:**

None

**Side Effects:**

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

## SmartLED_On (available only in the Mixed Mode)

**Description:**

The API function that turns on the LED specified by the argument bLEDIndex in quasi closed loop current control mode. Sets the bit corresponding to bLEDIndex in SmartLED_bOnMask. Disables the Programmable Timer interrupt. Turns on the LED at 100% duty cycle just long enough to measure the LED current with the ADC. The current measurement takes ~500 µs so the blink is not perceivable to the human eye. Uses the measured LED current, the target current (stored in SmartLED_baLEDCurrent), and SmartLED_bCurrentCoefficient to calculate the duty cycle. Iterates through the elements of SmartLED_baPWM and sets or clears the bit corresponding to bLEDIndex in each element as needed to drive the LED at the calculated duty cycle. Enables the Programmable Timer Interrupt.

**C Prototype:**

```
void  SmartLED_On(BYTE bLEDIndex)
```

**Assembly:**

```
mov    A, bLEDIndex
lcall  SmartLED_On
```

**Parameters:**

bLEDIndex – index of LED which needs to be turned on.

**Return Value:**

None

**Side Effects:**

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

## SmartLED_Off

**Description:**

The API function that turns off the LED specified by the argument bLEDIndex. Clears the bit corresponding to bLEDIndex in SmartLED_bOnMask. Disables the Programmable Timer interrupt. Iterates through the elements of SmartLED_baPWM and sets the bit corresponding to bLEDIndex in each element. This causes the LED duty cycle to be 0, effectively turning it off. Enables the Programmable Timer Interrupt.

**C Prototype:**

```
void  SmartLED_Off(BYTE bLEDIndex)
```

**Assembly:**

```
mov    A, bLEDIndex
lcall  SmartLED_Off
```

**Parameters:**

bLEDIndex – index of LED that needs to be turned off.

**Return Value:**

None

**Side Effects:**

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

## SmartLED_SetDutyCycle

**Description:**

The API function that sets the duty cycle of LED specified by the argument bLEDIndex to that specified by the argument bDutyCycle. Sets or clears the bit corresponding to bLEDIndex in SmartLED_bOnMask depending on bDutyCycle. If bDutyCycle is 0 the bit cleared, otherwise the bit is set. Disables the Programmable Timer interrupt. Iterates through the elements of SmartLED_baPWM and sets or clears the bit corresponding to bLEDIndex in each element as needed to drive the LED at the duty cycle specified by bDutyCycle. The actual duty cycle is always bDutyCycle rounded down to the nearest increment as determined by the number of PWM steps. Enables the Programmable Timer Interrupt.

**C Prototype:**

```
void  SmartLED_SetDutyCycle(BYTE bLEDIndex, BYTE bDutyCycle)
```

**Assembly:**

```
mov    A, bLEDIndex
mov    X, bDutyCycle
lcall  SmartLED_SetDutyCycle
```

**Parameters:**

bLEDIndex – index of LED that needs to be affected.

bDutyCycle – PWM DutyCycle value. The valid range is from 0 to 100.

**Return Value:**

None

**Side Effects:**

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

## SmartLED_wGetLEDCurrent (available only in the Mixed Mode)

**Description:**

This API function measures and returns the 100% duty cycle current of the LED corresponding to the index passed as an argument to the function, using the ADC with resolution of 10 bits and 8 MHz data clock.

**C Prototype:**

```
WORD   SmartLED_wGetLEDCurrent(BYTE bLEDIndex)
```

**Assembly:**

```
mov    A, bLEDIndex
lcall  SmartLED_wGetLEDCurrent
```

**Parameters:**

bLEDIndex – index of LED where current needs to be measured.

**Return Value:**

Return value is the two bytes current value. LSB is in the A, MSB in the X.

**Side Effects:**

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

## Timer and ADC API Functions

The SmartLED User Module extends all API functions of the Timer16 and ADCINC User Modules, except for Timer16_SetMode().

The prefixes for the embedded API functions of the Timer16 and ADCINC User Modules are the following:

- Timer16 API: SmartLED_Timer
- ADCINC API: SmartLED_ADC

**Note** The prototype of the ADCINC_Start() is changed to the following:

### *SmartLED_ADC_Start*

**Description:**

This API function sets the ADC input, clock, resolution, and starts the ADC.

**C Prototype:**

```
void  SmartLED_ADC_Start(BYTE bMux, BYTE bClock, BYTE bRes)
```

**Assembly:**

```
mov    A, bRes
push   A
mov    A, bClock
push   A
mov    A, bMux
push   A
lcall  SmartLED_ADC_Start
add    SP, 253   ; restores Stack Pointer
```

**Parameters:**

bMux – Byte that specifies the ADC input connection (internal temp sensor or analog mux bus). The symbolic names given in C and assembly, and their associated values, are listed in the following table:

| Symbolic Name | Value |
|---|---|
| SmartLED_ADC_INPUT_ANALOG_BUS | 0 |
| SmartLED_ADC_INPUT_TEMP_SENSOR | 1 |

bClock – This parameter determines the sample rate. The symbolic names given in C and assembly, and their associated values, are listed in the following table:

| Symbolic Name | Value |
|---|---|
| SmartLED_ADC_CLOCK_12_0MHz | 0x02 |
| SmartLED_ADC_CLOCK_8_0MHz | 0x03 |
| SmartLED_ADC_CLOCK_7_2MHz | 0x04 |
| SmartLED_ADC_CLOCK_6_0MHz | 0x05 |
| SmartLED_ADC_CLOCK_5_1MHz | 0x06 |
| SmartLED_ADC_CLOCK_4_5MHz | 0x07 |
| SmartLED_ADC_CLOCK_4_0MHz | 0x08 |
| SmartLED_ADC_CLOCK_3_6MHz | 0x09 |
| SmartLED_ADC_CLOCK_3_3MHz | 0x0A |

| Symbolic Name | Value |
|---|---|
| SmartLED_ADC_CLOCK_3_0MHz | 0x0B |
| SmartLED_ADC_CLOCK_2_8MHz | 0x0C |
| SmartLED_ADC_CLOCK_2_6MHz | 0x0D |
| SmartLED_ADC_CLOCK_2_4MHz | 0x0E |

bRes – this parameter determines the data format of the return result. Valid resolution options are from 8 to10 bits. The symbolic names given in C and assembly, and their associated values, are listed in the following table:

| Symbolic Name | Value |
|---|---|
| SmartLED_ADC_RES_8BIT | 0x00 |
| SmartLED_ADC_RES_9BIT | 0x01 |
| SmartLED_ADC_RES_10BIT | 0x02 |

**Return Value:**

None

**Side Effects:**

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

# Sample Firmware Source Code

## SmartLED Mixed Mode Example

The following C sample code demonstrates the SmartLED User Module functionality in the "Mixed mode" MUM configuration. The first two LEDs operate in the current control mode and the second two LEDs operate in the explicit PWM duty cycle mode:

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

void main(void)
{
    M8C_EnableGInt;     // Uncomment this line to enable Global Interrupts
    SmartLED_Start();   // Start SmartLED

    SmartLED_On(0);     // Turn on the first LED
    SmartLED_On(1);     // Turn on the second LED

    SmartLED_SetDutyCycle(2, 20);   // Turn on the third LED and set DutyCycle to 20
    SmartLED_SetDutyCycle(3, 60);   // Turn on the fourth LED and set DutyCycle to 60
}
```

The equivalent code written in Assembly is:

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

export _main

_main:

    M8C_EnableGInt ; Uncomment this line to enable Global Interrupts
    lcall SmartLED_Start ; Start SmartLED

    mov  A, 0
    lcall SmartLED_On ; Turn on the first LED

    mov  A, 1
    lcall SmartLED_On ; Turn on the second LED

    mov  A, 2
    mov  X, 20
    lcall SmartLED_SetDutyCycle ; Turn on the third LED and set DutyCycle to 20

    mov  A, 3
    mov  X, 60
    lcall SmartLED_SetDutyCycle ; Turn on the fourth LED and set DutyCycle to 60

.terminate:
    jmp .terminate
```

### SmartLED Explicit PWM Duty Cycle Mode Example

The following C sample code demonstrates the SmartLED User Module functionality in the "Explicit PWM duty cycle mode" MUM configuration:

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

void main(void)
{
    M8C_EnableGInt;     // Uncomment this line to enable Global Interrupts
    SmartLED_Start();   // Start SmartLED

    SmartLED_SetDutyCycle(0, 10);   // Turn on the first LED and set DutyCycle to 10
    SmartLED_SetDutyCycle(1, 20);   // Turn on the second LED and set DutyCycle to 20
    SmartLED_SetDutyCycle(2, 40);   // Turn on the third LED and set DutyCycle to 40
    SmartLED_SetDutyCycle(3, 60);   // Turn on the fourth LED and set DutyCycle to 60
}
```

The equivalent code written in Assembly is:

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

export _main

_main:

    M8C_EnableGInt ; Uncomment this line to enable Global Interrupts
    lcall SmartLED_Start ; Start SmartLED

    mov  A, 0
    mov  X, 10
    lcall SmartLED_SetDutyCycle ; Turn on the first LED and set DutyCycle to 10

    mov  A, 1
    mov  X, 20
    lcall SmartLED_SetDutyCycle ; Turn on the second LED and set DutyCycle to 20

    mov  A, 2
    mov  X, 40
    lcall SmartLED_SetDutyCycle ; Turn on the third LED and set DutyCycle to 40

    mov  A, 3
    mov  X, 60
    lcall SmartLED_SetDutyCycle ; Turn on the fourth LED and set DutyCycle to 60

.terminate:
    jmp .terminate
```

## Configuration Registers

The SmartLED User Module in the Mixed mode occupies the Programmable Timer and ADC blocks. In the Explicit PWM Duty Cycle mode it occupies only the Programmable Timer.

The ADC block does not provide user configurable registers.

To obtain a list of user configurable registers for Programmable Timer block, see the *Timer16 User Module Datasheet* and *PSoC® CY8C20xx6A/AS Family Technical Reference Manual*.

**Note** It is recommended that you set the configuration registers using only the user module API.

# Version History

| Version | Originator | Description |
|---|---|---|
| 1.00 | DHA | Initial version. |
| 1.10 | DHA | Corrected SmartLED_Stop API function. Fixed issue with drive modes for LEDs. |
| 1.10.b | MYKZ | Moved User Module to Legacy state. This is no longer recommended for new designs. |

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



Copyright © 2011-2013 Cypress Semiconductor Corporation. The information contained herein is subject to change without notice. Cypress Semiconductor Corporation assumes no responsibility for the use of any circuitry other than circuitry embodied in a Cypress product. Nor does it convey or imply any license under patent or other rights. Cypress products are not warranted nor intended to be used for medical, life support, life saving, critical control or safety applications, unless pursuant to an express written agreement with Cypress. Furthermore, Cypress does not authorize its products for use as critical components in life-support systems where a malfunction or failure may reasonably be expected to result in significant injury to the user. The inclusion of Cypress products in life-support systems application implies that the manufacturer assumes all risk of such use and in doing so indemnifies Cypress against all charges.

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

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

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

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