Please note that Cypress is an Infineon Technologies Company.
The document following this cover page is marked as “Cypress” document as this is the company that originally developed the product. Please note that Infineon will continue to offer the product to new and existing customers as part of the Infineon product portfolio.

Continuity of document content
The fact that Infineon offers the following product as part of the Infineon product portfolio does not lead to any changes to this document. Future revisions will occur when appropriate, and any changes will be set out on the document history page.

Continuity of ordering part numbers
Infineon continues to support existing part numbers. Please continue to use the ordering part numbers listed in the datasheet for ordering.

CE220263 – PSoC 6 MCU GPIO Pins Example

Objective

This example demonstrates GPIO pin operation in PSoC® 6 MCU.

Overview

This example demonstrates multiple methods of configuring, reading, writing, and generating interrupts with PSoC 6 General Purpose Input/Output (GPIO) pins. Both the PSoC Creator™ schematic Pins Component and PDL GPIO driver methods are shown.

Requirements

Tool: PSoC Creator™ 4.2

Programming Language: C (ARM® GCC 5.4-2016-q2-update)

Associated Parts: PSoC 6 MCU family of devices

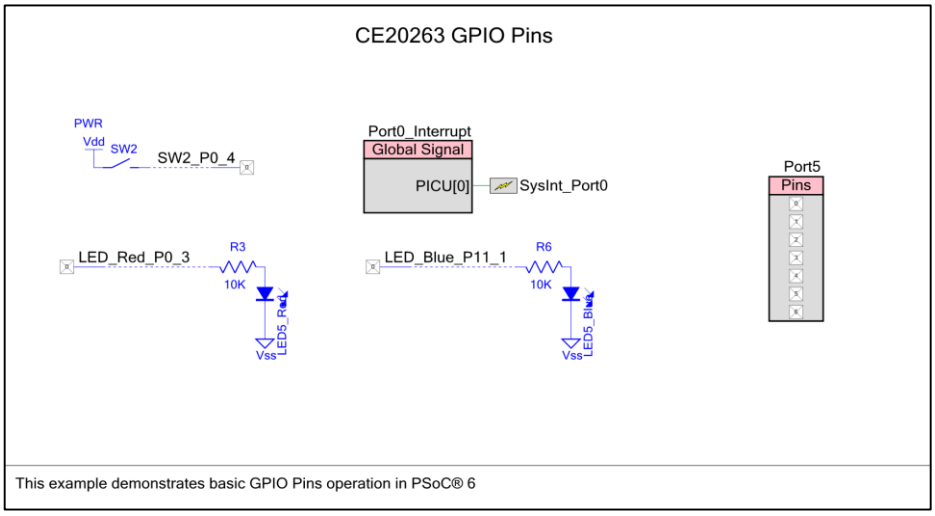
Related Hardware: CY8CKIT-062-BLE PSoC 6 BLE Pioneer Kit and CY8CKIT-062 PSoC 6 Pioneer Kit

Design

GPIO_Pins

The GPIO_Pins design shown in Figure 1 demonstrates GPIO pin configuration, reading, writing, full port access, and interrupts, using methods compatible with the PSoC Creator Pins Component and GPIO PDL driver.

Figure 1. GPIO Pins Component Example Schematic



To demonstrate individual GPIO pin access, the example has a digital input pin connected to the development board user button. The button state is continuously read by the CM4 processor. The value read is then written to a digital output pin connected to a kit LED. When the button is pressed the red LED is on. Several variations of reading and writing pin data are shown; select the method that works best for your design.

The digital input pin is also configured to generate an interrupt on a falling edge, which occurs on a button release. The interrupt routine causes the blue LED on a second digital output pin to turn on for approximately 1 second.

A full port of GPIO pins is configured using a Pins Component array on port 5. The value on port 5 is continuously read with direct register reads, incremented, and written back out to the port. The toggling port pins can be monitored on an oscilloscope.

Pin Configuration

Device configuration tools like PSoC Creator automatically generate GPIO configuration code and execute it as part of the device boot process in *cyfitter_cfg.c*. GPIO configuration methods are typically only used with manual PDL GPIO configuration when not using a configuration tool. They may also be used at run time to dynamically reconfigure GPIO pins independent of how the initial configuration was performed.

Most GPIO pins only require their basic parameters to be set and can use default values for all other settings. This allows use of a simplified Initialization function. `Cy_GPIO_Pin_FastInit()` only supports parameterized configuration of drive mode, output logic level, and High Speed Input/Output Multiplexer (HSIOM) setting. The HSIOM setting determines a pin's high level software, peripheral, and analog control and connectivity. All other configuration settings are untouched from their reset or previously set state. This function is very useful at run time to dynamically change a pin's configuration. For example, configure a pin to strong drive mode to write data, and then reconfigure the pin as high impedance to read data.

```
Cy_GPIO_Pin_FastInit(P0_3_PORT, P0_3_NUM, CY_GPIO_DM_STRONG, 1, HSIOM_SEL_GPIO);
```

A method to configure all attributes of a single pin is to use the `Cy_GPIO_Pin_Init()` function and a pin configuration structure. While easy to use, it generates larger code than other configuration methods.

```
Cy_GPIO_Pin_Init(P0_4_PORT,P0_4_NUM, &P0_4_Pin_Init);
```

The most code efficient method to configure all attributes for a full port of pins is to use the `Cy_GPIO_Port_Init()` function and a port configuration structure. It packs all the configuration data into direct register writes for the whole port. Its limitation is that it must configure all pins in a port and the user must calculate the combined register values for all pins or copy them from a configuration tool. This is the method used by automated configuration tools as part of the device boot process in *cyfitter_cfg.c*.

```
Cy_GPIO_Port_Init(GPIO_PRT5, &port5_Init);
```

Individual pin configuration settings can also be changed at run time using supplied driver functions. An example of some of these functions are provided below. The function parameters demonstrate use of pin specific #defines provided in *cyfitter_gpio.h* when a configuration tool is used.

```
Cy_GPIO_SetHSIOM(SW2_P0_4_PORT, SW2_P0_4_NUM, SW2_P0_4_INIT_MUXSEL);
Cy_GPIO_SetDrivemode(SW2_P0_4_PORT, SW2_P0_4_NUM, CY_GPIO_DM_PULLUP);
Cy_GPIO_SetVtrip(SW2_P0_4_PORT, SW2_P0_4_NUM, SW2_P0_4_THRESHOLD_LEVEL);
Cy_GPIO_SetSlewRate(SW2_P0_4_PORT, SW2_P0_4_NUM, SW2_P0_4_SLEWRATE);
Cy_GPIO_SetDriveSel(SW2_P0_4_PORT, SW2_P0_4_NUM, CY_GPIO_DRIVE_FULL);
```

Pin Input Read Methods

The following methods all perform the same read from a GPIO pin using the different read methods available. Please choose the most appropriate method for your specific use case. The `Cy_GPIO_Read()` function is thread and multi-core safe. Most GPIO driver functions require a minimum of two arguments to define the port and pin in that port to operate on. The port argument expects the base address of the port's registers. The pin argument expects the pin number within the port.

The preferred read method for use with configuration tools is using #defines provided for configuration tool pin names. In PSoC Creator these are Pin components placed on the schematic and the pin #defines are located in *\\pdl\Pins and Interrupts\cyfitter_gpio.h*

```
pinReadValue = Cy_GPIO_Read(SW2_P0_4_PORT, SW2_P0_4_NUM);
```

The preferred read method for direct PDL use without a configuration tool is with user defined custom #define pin names. User supplied #defines are typically placed in a user created .h file.

```
#define mySwPin_Port P0_4_PORT
#define mySwPin_Num  P0_4_NUM
pinReadValue = Cy_GPIO_Read(mySwPin_Port, mySwPin_Num);
```

Pin reads can also be performed using default device pin name #defines provided for each device and package. The default name #defines are located in *\\Generated_Source\(device family)\pdl\devices\(device family)\(device series)\include\gpio_(series+package).h*

```
pinReadValue = Cy_GPIO_Read(P0_4_PORT, P0_4_NUM);
```

Pins can also be read using default port register name #defines and pin number #defines located in *\\Generated_Source\(device family)\pdl\devices\(device family)\(device series)\include\(part number).h*

```
pinReadValue = Cy_GPIO_Read(GPIO_PRT0, 4);
```

Pins reads using port and pin numbers are also supported. This method is useful for algorithmically generated port and pin numbers. `Cy_GPIO_PortToAddr()` is a helper function that converts the port number into the required port register base address required by other GPIO driver functions.

```
portNumber = 0;
pinReadValue = Cy_GPIO_Read(Cy_GPIO_PortToAddr(portNumber), 4);
```

Like any MCU, direct port register access is always available and useful for accessing multiple pins in a port simultaneously or developing application optimized port accesses. The following example shows a port IN register read with mask and shift of the desired pin data.

```
pinReadValue = (GPIO_PRT0->IN >> P0_4_NUM) & CY_GPIO_IN_MASK;
```

Pin Output Write Methods

The following methods all perform the same write to GPIO pins using the different write methods available. Please choose the most appropriate method for your specific use case. The `Cy_GPIO_Write()` function is best used when the desired pin state is not already known and is determined at run time. The Write function uses atomic operations that directly affect only the selected pin without using read-modify-write operations. The Write function is therefore thread and multi-core safe

The preferred method for use with configuration tools like PSoC Creator is pin writes using #defines provided by the configuration tool. The generated #defines are located in *\\pdl\Pins and Interrupts\cyfitter_gpio.h*

```
Cy_GPIO_Write(LED_Red_P0_3_PORT, LED_Red_P0_3_NUM, pinReadValue);
```

The preferred method for direct PDL use without a configuration tool is pin writes with user defined custom #define pin names. The user supplied #defines are typically placed in a user generated .h file.

```
#define myLedPin_Port P0_3_PORT
#define myLedPin_Num  P0_3_NUM
Cy_GPIO_Write(myLedPin_Port, myLedPin_Num, pinReadValue);
```

Pin writes can also be made using the default device pin name #defines located in the *\\Generated_Source\(device family)\pdl\devices\(device family)\(device series)\include\gpio_(series+package).h* file.

```
Cy_GPIO_Write(P0_3_PORT, P0_3_NUM, pinReadValue);
```

Pin writes are possible using default port register name #defines and pin numbers. #defines located in the *\\Generated_Source\(device family)\pdl\devices\(device family)\(device series)\include\(part number).h* file.

```
Cy_GPIO_Write(GPIO_PRT0, 3, pinReadValue);
```

For algorithmically generated port and pin numbers, pin writes using port and pin numbers are very useful. `Cy_GPIO_PortToAddr()` is a helper function that converts the port number into the required port register base address

```
portNumber = 0;
Cy_GPIO_Write(Cy_GPIO_PortToAddr(portNumber), 3, pinReadValue);
```

The most efficient output methods, when the desired pin state is already known at compile time, are to directly Set, Clear, and Invert the pin output state. These register writes are atomic operations that directly affect only the selected pin without using read-modify-write operations. They are therefore thread and multi-core safe. The same argument variations as demonstrated with the `Cy_GPIO_Write()` function can be used.

```
Cy_GPIO_Set(LED_Red_P0_3_PORT, LED_Red_P0_3_NUM);
Cy_GPIO_Clr(LED_Red_P0_3_PORT, LED_Red_P0_3_NUM);
Cy_GPIO_Inv(LED_Red_P0_3_PORT, LED_Red_P0_3_NUM);
```

Port Access

Direct register access is used to interface with multiple pins in one port at the same time. These accesses may not be thread or multi-core safe due to possible read-modify-write operations. All pins in a port under direct register control should only be accessed by a single CPU core unless access protections are provided at the system level..

```
portReadValue = GPIO_PRT5->IN;
portReadValue++;
GPIO_PRT5->OUT = portReadValue;
```

Pin Interrupts

In order for a pin to generate an interrupt it must be configured to trigger on a rising, falling, or both edges and be masked so that the pin signal is sent to the interrupt controller vector for that port.

```
Cy_GPIO_SetInterruptEdge(SW2_P0_4_PORT, SW2_P0_4_NUM, CY_GPIO_INTR_RISING);
Cy_GPIO_SetInterruptMask(SW2_P0_4_PORT, SW2_P0_4_NUM, CY_GPIO_INTR_EN_MASK);
```

The port interrupt vector must then be configured, cleared, and enabled to trigger off of the port interrupt signal and be mapped to the desired Interrupt Service Routine (ISR). See the PDL Cy_SysInt documentation for more information on interrupt configuration and use.

```
Cy_SysInt_Init(&SysInt_Port0_cfg, GPIO_Interrupt);
NVIC_ClearPendingIRQ(SysInt_Port0_cfg.intrSrc);
NVIC_EnableIRQ((IRQn_Type)SysInt_Port0_cfg.intrSrc);
```

After an interrupt occurs the pin interrupt must be cleared prior to exiting the ISR to reset the edge detection logic to allow detection of the next edge.

```
void GPIO_Interrupt()
{
    /* User Code Here */
    Cy_GPIO_ClearInterrupt(SW2_P0_4_PORT, SW2_P0_4_NUM);
}
```

If more than one pin in a port can generate an interrupt the `Cy_GPIO_GetInterruptStatus()` function may be used to determine which pin detected an edge event and generated the interrupt. Optionally direct register reads of the INTR register may be used to determine the interrupt pin.

```
if(Cy_GPIO_GetInterruptStatus(SW2_P0_4_PORT, SW2_P0_4_NUM) == CY_GPIO_INTR_STATUS_MASK)

portIntrStatus = SW2_P0_4_PORT->INTR;
if(CY_GPIO_INTR_STATUS_MASK == ((portIntrStatus >> SW2_P0_4_NUM) &
                                CY_GPIO_INTR_STATUS_MASK))
```

In most designs each port that can generate an interrupt will be assigned to its own interrupt vector. If interrupt vector limitations do not allow discrete vectors to be used, the combined port interrupt (AllPortInt) may be used. The AllPortInt ORs all of the individual port interrupt signals into a single chip wide signal requiring only one vector for any number of pins on the device. The `Cy_GPIO_GetInterruptCause0()` function can be used to determine which port(s) detected interrupt edge events.

```
if((Cy_GPIO_GetInterruptCause0() & 1u) == 1u)
```

Design Considerations

This code example is designed for the PSoC 6 device family and associated CY8CKIT-062 or CY8CKIT-062-BLE kits. The design is easily portable to other PSoC 6 devices and kits, typically by just changing the LED and button pin assignments.

Hardware Setup

This example uses the kit's default configuration. Refer to the kit guide to ensure the kit is configured correctly.

Software Setup

This project requires PSoC Creator to generate the complete device configuration and Pins Component constants. By default this project uses PSoC Creator to configure the device hardware including pins. As an example, the GPIO PDL driver pin and port configuration methods can be optionally enabled for some of the pins by setting the `PDL_PIN_CONFIGURATION` constant to (1u).

Operation

1. Connect the CY8CKIT-062-BLE Kit or CY8CKIT-062 Kit to a USB port on your PC.
2. Build and program the application into the kit. For more information on building a project or programming a device, see PSoC Creator Help.
3. Press SW2 and observe the red LED lights up brightly. You may observe that the red LED glows dimly even when the button is not pressed. This is due to the short on pulse generated by the invert function example in the code.
4. Release SW2 and observe that red LED turns off and blue LED turns on for approximately 1 second.

Components

Table 1 lists the PSoC Creator Components used in this example, as well as the hardware resources used by each.

Table 1. PSoC Creator Components

Component	Instance Name	Hardware Resources
Digital Input Pins	SW2_P0_4	1 Digital input pin
Digital Output Pins	LED_Red_P0_3 LED_Blue_P11_1 Port5	8 Digital input pins
Global System	Port0_Interrupt	Port interrupt 0 (PICU[0])
System Interrupt (SysInt)	SysInt_Port0	None

Parameter Settings

Non-default settings for each Component are outlined in red in the following figures.

Figure 2 shows the SW2_P0_4 Pins Component parameter settings.

Figure 2. SW2_P0_4 Pins Component Parameter Settings

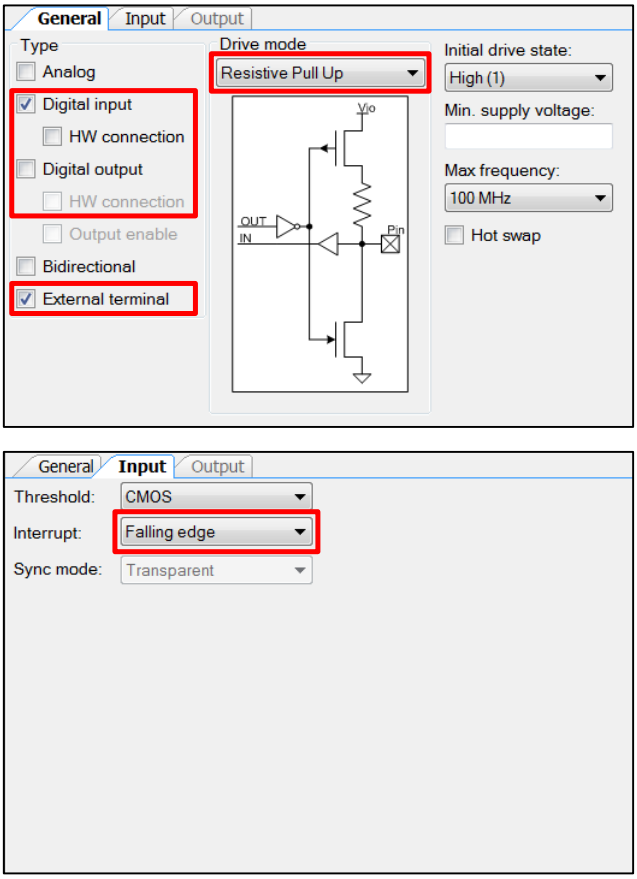


Figure 3 shows the LED_Red_P0_3 and LED_Blue_P11_1 Pins Component parameter settings.

Figure 3. LED_Red_P0_3 and LED_Blue_P11_1 Pins Component Parameter Settings

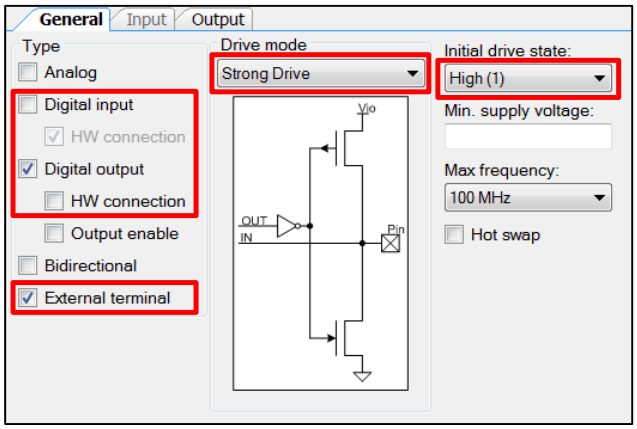


Figure 4 shows the Port5 Pins Component parameter settings.

Figure 4. Port5 Pins Component Parameter Settings

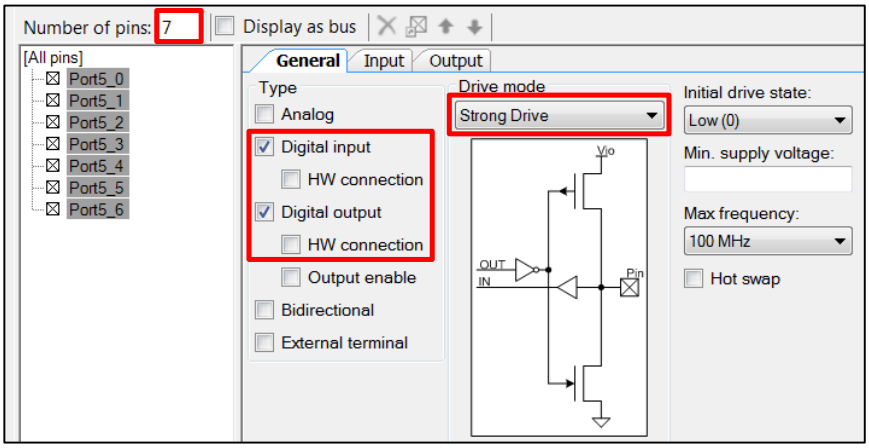
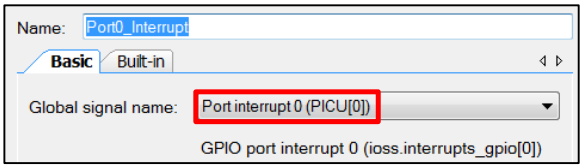


Figure 5 shows the Port0 Pins Component parameter settings.

Figure 5. Port0 Pins Component Parameter Settings



Design-Wide Resources

Figure 6 lists the PSoC Creator Pin connection settings required on CY8CKIT-062-BLE and CY8CKIT-062 Kits.

Figure 6. PSoC Creator Pin Connection Settings on CY8CKIT-062-BLE and CY8CKIT-062 Kits

Name	Port	Pin	Lock
LED_Blue_P11_1	P11[1]	C9	☑
LED_Red_P0_3	P0[3]	F3	☑
Port5[0]	P5[0]	N7	☑
Port5[1]	P5[1]	L8	☑
Port5[2]	P5[2]	M8	☑
Port5[3]	P5[3]	N8	☑
Port5[4]	P5[4]	L9	☑
Port5[5]	P5[5]	M9	☑
Port5[6]	P5[6]	N9	☑
SW2_P0_4	P0[4]	F2	☑

Related Documents

Application Notes	
AN210781	Getting Started with PSoC 6 MCU with Bluetooth Low Energy (BLE) Connectivity
PSoC Creator Component Datasheets	
Pins Component	Supports Analog, Digital I/O, and Bidirectional signal types
PDL Users Guide	Documents PDL driver library including GPIO functions
General Purpose Input / Output (GPIO) PDL Driver	Supports all GPIO pin features
System Interrupt (SysInt)	Interrupt vectoring and control
Global Signal	Provides access to GPIO port interrupt signals
Device Documentation	
PSoC 6 MCU: PSoC 63 with BLE Datasheet	
PSoC 6 MCU: PSoC 63 with BLE Architecture Technical Reference Manual	
PSoC 6 MCU: PSoC 63 with BLE Register Technical Reference Manual	
PSoC 6 MCU: PSoC 62 Datasheet	
Development Kit (DVK) Documentation	
PSoC 6 BLE Pioneer Kit PSoC 6 Pioneer Kit	

Document History

Document Title: CE220263 - PSoC 6 MCU GPIO Pins Example

Document Number: 002-20263

Revision	ECN	Orig. of Change	Submission Date	Description of Change
**	5791434	GJV	8/29/2017	New code example

Worldwide Sales and Design Support

Cypress maintains a worldwide network of offices, solution centers, manufacturer's representatives, and distributors. To find the office closest to you, visit us at Cypress Locations.

Products

ARM® Cortex® Microcontrollers	cypress.com/arm
Automotive	cypress.com/automotive
Clocks & Buffers	cypress.com/clocks
Interface	cypress.com/interface
Internet of Things	cypress.com/iot
Memory	cypress.com/memory
Microcontrollers	cypress.com/mcu
PSoC	cypress.com/psoc
Power Management ICs	cypress.com/pmic
Touch Sensing	cypress.com/touch
USB Controllers	cypress.com/usb
Wireless Connectivity	cypress.com/wireless

PSoC® Solutions

PSoC 1 | PSoC 3 | PSoC 4 | PSoC 5LP | PSoC 6

Cypress Developer Community

Forums | WICED IOT Forums | Projects | Videos | Blogs | Training | Components

Technical Support

cypress.com/support

All other trademarks or registered trademarks referenced herein are the property of their respective owners.

Cypress Semiconductor
198 Champion Court
San Jose, CA 95134-1709

© Cypress Semiconductor Corporation, 2017. This document is the property of Cypress Semiconductor Corporation and its subsidiaries, including Spansion LLC ("Cypress"). This document, including any software or firmware included or referenced in this document ("Software"), is owned by Cypress under the intellectual property laws and treaties of the United States and other countries worldwide. Cypress reserves all rights under such laws and treaties and does not, except as specifically stated in this paragraph, grant any license under its patents, copyrights, trademarks, or other intellectual property rights. If the Software is not accompanied by a license agreement and you do not otherwise have a written agreement with Cypress governing the use of the Software, then Cypress hereby grants you a personal, non-exclusive, nontransferable license (without the right to sublicense) (1) under its copyright rights in the Software (a) for Software provided in source code form, to modify and reproduce the Software solely for use with Cypress hardware products, only internally within your organization, and (b) to distribute the Software in binary code form externally to end users (either directly or indirectly through resellers and distributors), solely for use on Cypress hardware product units, and (2) under those claims of Cypress's patents that are infringed by the Software (as provided by Cypress, unmodified) to make, use, distribute, and import the Software solely for use with Cypress hardware products. Any other use, reproduction, modification, translation, or compilation of the Software is prohibited.

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

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