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.

www.infineon.com

Using Timer Interrupt In Cypress EZ-USB® FX2LP™ Based Applications

Associated Project: Yes
Associated Part Family: CY7C68013/14/15/16
Software Version: NA
Related Application Note: AN65209
To get the latest version of this application note, or the associated project file, please visit http://www.cypress.com/go/AN1193.

More code examples? We heard you.
To access a variety of FX2LP code examples, visit our USB High-Speed Code Examples webpage.
Are you looking for USB 3.0 peripheral controllers?
To access USB 3.0 product family, visit our USB 3.0 Product Family webpage

AN1193 is aimed at helping firmware developers to use timer interrupts in their applications. It is assumed that the reader has a general understanding of how interrupts work within the 8051 core. The associated code example increments the count from 0 to 9 on the seven-segment display of the FX2LP Development Board (CY3684) at every timer interrupt. The inter-digit delay is controlled using BULK OUT endpoint from the Control Center application.

1 Introduction

Cypress' EZ-USB® FX2LP™ is a low-power, highly-integrated USB 2.0 peripheral controller designed to handle the full USB 2.0 bandwidth. The FX2LP webpage has more details on FX2LP and additional learning resources such as datasheets and application notes. Application note AN65209 provides an introduction to FX2LP.

EZ-USB FX2LP includes three timers/counters (Timer 0, Timer 1, and Timer 2). Each timer/counter can operate as either a timer with a clock rate based on the internal 24-MHz clock, or as an event counter clocked by the T0 pin (Timer 0), T1 pin (Timer 1), or T2 pin (Timer 2). Each timer/counter consists of a 16-bit register.

- Timer 0 - TL0 and TH0
- Timer 1 - TL1 and TH1
- Timer 2 - TL2 and TH2

Timer 0 and Timer 1 are mostly used to generate timer tick interrupts, and thus we will be concerned with the details regarding these two timers. Timer 2 is mostly used as a baud rate generator.

Two special function registers (SFRs) control the Timer 0 and Timer 1 modes, **TCON** and **TMOD**, as Figure 1 and Figure 2 shows.

The **TCON SFR** turns on the counting or timing, and TH1/TH0 and TL1/TL0 registers determine the initial values for each timer. The timer will start counting from the initial values loaded in the TH1/TH0 and TL1/TL0 registers.

Figure 1. TCON Register

(MSB)						(LSB)	
TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0

TF0, TF1—Overflow flag is set by hardware when the timer overflows. Cleared by hardware when the CPU vectors to ISR.

TR0, TR1—Turns timer on when set, turns timer off when cleared.

IE0, IE1—Interrupt edge flag is set by the hardware when external interrupt falling edge or low level is detected. This flag is cleared when the interrupt is processed.

IT0, IT1—Interrupt type is set by software. Set indicates a falling-edge triggered interrupt and cleared indicates a low-level triggered interrupt.

The **TMOD SFR** configures the timers to be in timer mode or counter mode. In *timer* mode, the timer counts the internal clock. In *counter* mode, the timer counts transitions on a designated input pin of the 8051, in this case the T0 or T1 pins. Bits 7-4 are associated with Timer 1 and bits 3-0 are associated with Timer 0.

Figure 2. TMOD Register

(MSB)						(LSB)	
Gate	C/T	M1	M0	Gate	C/T	M1	M0

Timer 1 Timer 0

GATE = 0—timer runs when TR0 (TR1) is set

GATE = 1—timer runs only when INT0 (INT1) is high along with TR0 (TR1)

C/T = 0—input from system clock

C/T = 1—input from T0 (T1) pin

M1 M0

Mode 00—13-bit counter, lower five bits of TL0 (TL1) and all eight bits of TH0 (TH1)

Mode 10—8-bit auto-reload. The value in TH0 (TH1) will be stored in TL0 (TL1) when the latter overflows.

Modes 01—16-bit counter (which is what we want!)

Mode 11

For Timer 0—Two 8-bit counters

For Timer 1—Timer 1 inactive

The default timer clock scheme for the 8051 timers is 12 CLK24 (system clock of 8051) cycles per increment, the same as in the standard 8051. However, in the EZ-USB 8051 one instruction cycle is four CLK24 cycles. Another SFR called CKCON allows the EZ-USB 8051 to use the default rate (12 clocks per timer increment), keeping compatibility with existing application code. However, applications that require fast timing can set the timers to increment every four CLK24 cycles by setting bits in the CKCON register.

Figure 3 shows the details associated with the **CKCON** register. For further details, see Appendix C of the EZ-USB TRM.

Figure 3. CKCON Register

(MSB)						(LSB)	
Rsrvd	Rsrvd	T2M	T1M	T0M	MD2	MD1	MD0

Rsrvd—Reserved

T2M—Timer 2 clock select. When T2M = 0, Timer 2 uses CLK24/12 (for compatibility with 80C32); when T2M = 1, Timer 2 uses CLK24/4. This bit has no effect when Timer 2 is configured for baud rate generation.

T1M—Timer 1 clock select. When T1M = 0, Timer 1 uses CLK24/12 (for compatibility with 80C32); when T1M = 1, Timer 1 uses CLK24/4.

T0M—Timer 0 clock select. When T0M = 0, Timer 0 uses CLK24/12 (for compatibility with 80C32); when T0M = 1, Timer 0 uses CLK24/4.

MD2, MD1, MD0—Controls the number of cycles to be used for external MOVX instructions, or stretch. These bits are not used to configure timers.

In our application, we are interested in using Timer 0 as a 16-bit counter**.** Therefore, when the timer count reaches past 65535, a timer overflow interrupt will occur. When this happens, the 8051 looks for the Timer 0 ISR at address 000Bh (the interrupt vector address for Timer 0 interrupt).

2 Time Calculation

As an example, let us consider triggering a timer overflow interrupt every 10 ms or 100 Hz. The enhanced 8051 in the EZ-USB chip runs at 48 MHz; if we configure it with a 12-clock bus cycle, to cause a timer overflow every 10 ms we need to set TH0/TL0 to**:**

65536 - (48,000,000 / 12 * FREQ) (where FREQ = 100Hz)

=10000h-9C40h

= 63C0h

= TIMER0_COUNT

2.1 Timer Setup

The following code segment initializes the timer and allows for a timer overflow interrupt to occur every 10 ms. Put this code in the *timer.c* file that is a part of the associated project and call it from wherever you initialize your target hardware. In this case, TD_Init () function.

```
// This function enables Timer 0. Timer 0 generates a synchronous interrupt
// once every 100Hz or 10 ms.
void timer0_init (void)
{
    EA = 0; // disables all interrupts
    timer0_tick = 0;
    TR0 = 0; // stops Timer 0
    CKCON = 0x03; // Timer 0 using CLKOUT/12
    TMOD &= ~0x0F; // clear Timer 0 mode bits
    TMOD |= 0x01; // setup Timer 0 as a 16-bit timer
    TL0 = (TIMER0_COUNT & 0x00FF); // loads the timer counts
    TH0 = (TIMER0_COUNT >> 8);
    PT0 = 0; // sets the Timer 0 interrupt to low priority
    ET0 = 1; // enables Timer 0 interrupt
    TR0 = 1; // starts Timer 0
    EA = 1; // enables all interrupts
```

After the timer is initialized, the target hardware has a 10-ms timer. The above code-snippet initializes Timer 0, configures it as a 16-bit timer, enables the timer interrupt, and starts the timer. The source clock for Timer 0 is configured to be ticking every 12 clock cycles. For more details on the SFRs, refer to the EZ-USB TRM.

2.2 Timer 0 ISR

When Timer 0 overflows the following ISR is invoked, and the timer tick is incremented.

```
// Timer Interrupt
// This function is an interrupt service routine for Timer 0. It should never
// be called by a C or assembly function. It will be executed automatically
// when Timer 0 overflows.
// "interrupt 1" tells the compiler to look for this ISR at address 000Bh
// "using 1" tells the compiler to use register bank 1

void timer0 (void) interrupt 1 using 1
{
// Stop Timer 0, adjust the Timer 0 counter so that we get another
    // in 10ms, and restart the timer.
    TR0 = 0; // stop timer
    TL0 = TL0 + (TIMER0_COUNT & 0x00FF);
    TH0 = TH0 + (TIMER0_COUNT >> 8);
    TR0 = 1; // start Timer 0
  // Increment the timer tick. This interrupt should occur approximately every 10 ms
    // So the resolution of the timer will be 100Hz not including interrupt latency.
    timer0_tick++;
}
```

“Interrupt 1” indicates to the compiler that it is an ISR for Timer 0 interrupt and its function body should be placed at the address 000Bh. In the ISR, Timer 0 is stopped and the initial values of the timer counts are reloaded. The timer is restarted and the code returns from the ISR.

The next two functions are very useful because timer0_count is used to get the current tick, and timer0_delay can be used to wait for an input or flash an LED every second. If you invoke this routine using timer0_delay(100), the delay will be approximately 1 second.

2.3 timer0_count

```
// This function returns the current Timer 0 tick count.
unsigned timer0_count (void)
{
    unsigned t;
    EA = 0;
    t = timer0_tick;
    EA = 1;
    return (t);
}
```

2.4 timer0_delay

This function takes count as an input parameter, which is the number of 10 ms delays to be introduced, that is, timer0_delay(count) will introduce a delay of (count x 10) ms delay. It will check the timer0_tick value being returned by the timer0_count() and wait until the timer0 interrupt occurs ‘count’ number of times. Timer0 interrupt will occur after every 10 ms, because we have configured the timer0 to be a 10 ms timer.

```
// This function waits for 'count' timer ticks to pass.
void timer0_delay (unsigned count)
{
    unsigned start_count;
    start_count = timer0_count(); // get the starting count
    // wait for timer0_tick to reach count
    while ((timer0_count() - start_count) <= count){}
}
```

3 Endpoint Control Code

The code that controls how fast the 0-9 count steps is in TD_Poll(). Basically, you can write bytes to EP2 (ranging from 0x01 to 0xFF), and each value will be passed into the timer0_delay function.

For example, if a BULK OUT transfer of 01h is performed on EP2, the seven segment display would flash every second.

Note The number of bytes written to EP2 will determine the delay between displaying each digit of 0-9 count. Performing an IN transfer afterwards will give the timer tick counts on EP6. Code is placed into TD_Init() function to flash the seven-segment display every 0.4 seconds after the hex file is downloaded.

```
void TD_Poll(void) // Called repeatedly while the device is idle
{
    BYTE count, i, j, temp;
    BYTE factor = 0x64;
    WORD new;

    if(!(EP2468STAT & bmEP2EMPTY))
    {
        // check EP2 EMPTY(busy) bit in EP2468STAT (SFR), core set's this bit
        //when FIFO is empty

        count = (EP2BCH << 8) + EP2BCL;
        // loop EP2OUT buffer data to EP6IN
        for( i = 0x0000; i < count; i++ )
            {
                temp = EP2FIFOBUF[i];
                // stores control byte in temp variable
                new = temp*factor;
                // multiplies value of 0x01-0xFF by 64 to give 1s-5s range
                if(!(EP2468STAT & bmEP6FULL))
                {
                // check EP6 FULL(busy) bit in EP2468STAT (SFR), core set's this
                //bit when FIFO is full

                    EP6FIFOBUF[i] = timer0_count();
                // if you'd like to see the value of timer0_tick
                    }
                for(j=0;j<10;j++) // loop counts from 0-9
            {
            // Writes to LED with Digit
                EZUSB_WriteI2C(LED_ADDR, 0x01, &(Digit[j]));
                // Waits for write
                 EZUSB_WaitForEEPROMWrite(LED_ADDR);
                 // Delays digit update by byte written to EP2
                 timer0_delay(new);
            }
        }
        EP6BCL = i;
        EP2BCL = 0x80;

    }
}
```

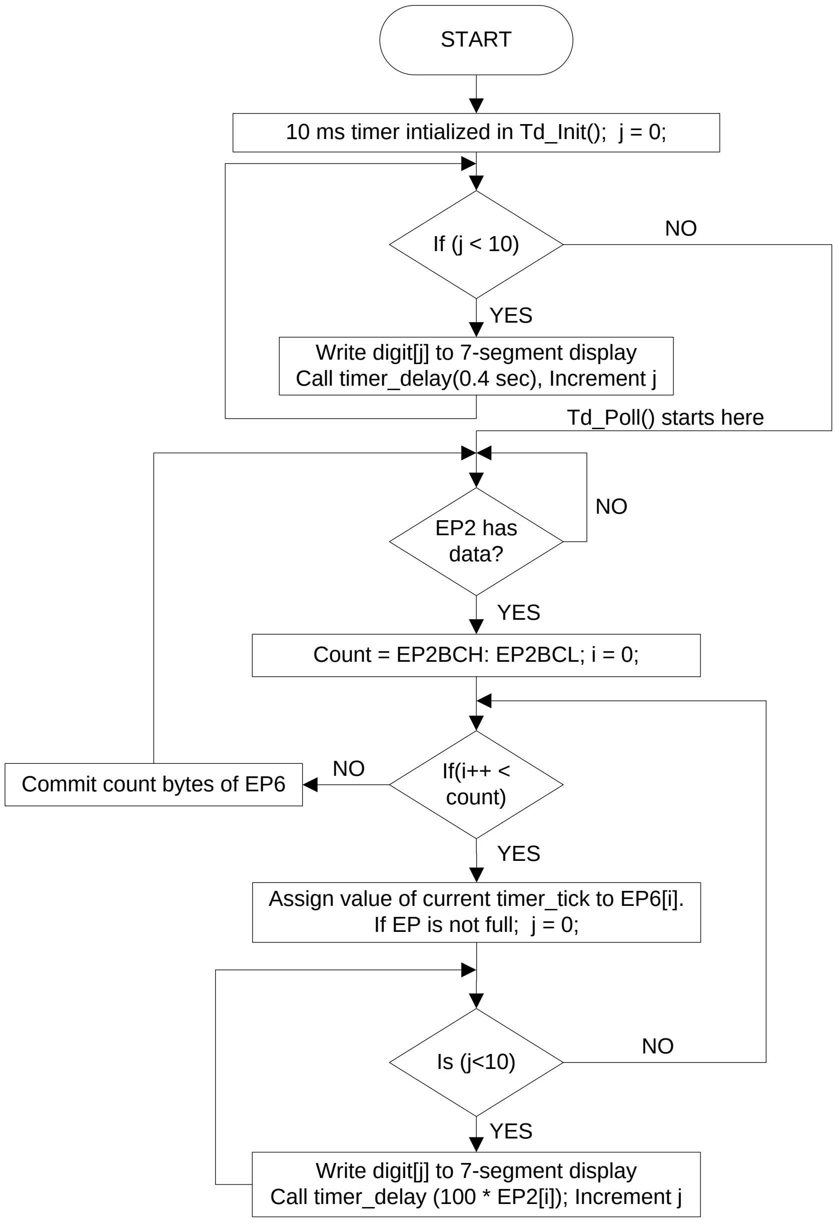
Table 1 shows the hex values that correspond to delays ranging from 1 second to 5 seconds. Note that delays can range from 1s to 255s depending on the byte you transfer (01H – FFH).

Table 1. Hex Values and Corresponding Delays

Hex Value	Seconds
01	1
02	2
03	3
04	4
05	5

4 Flowchart

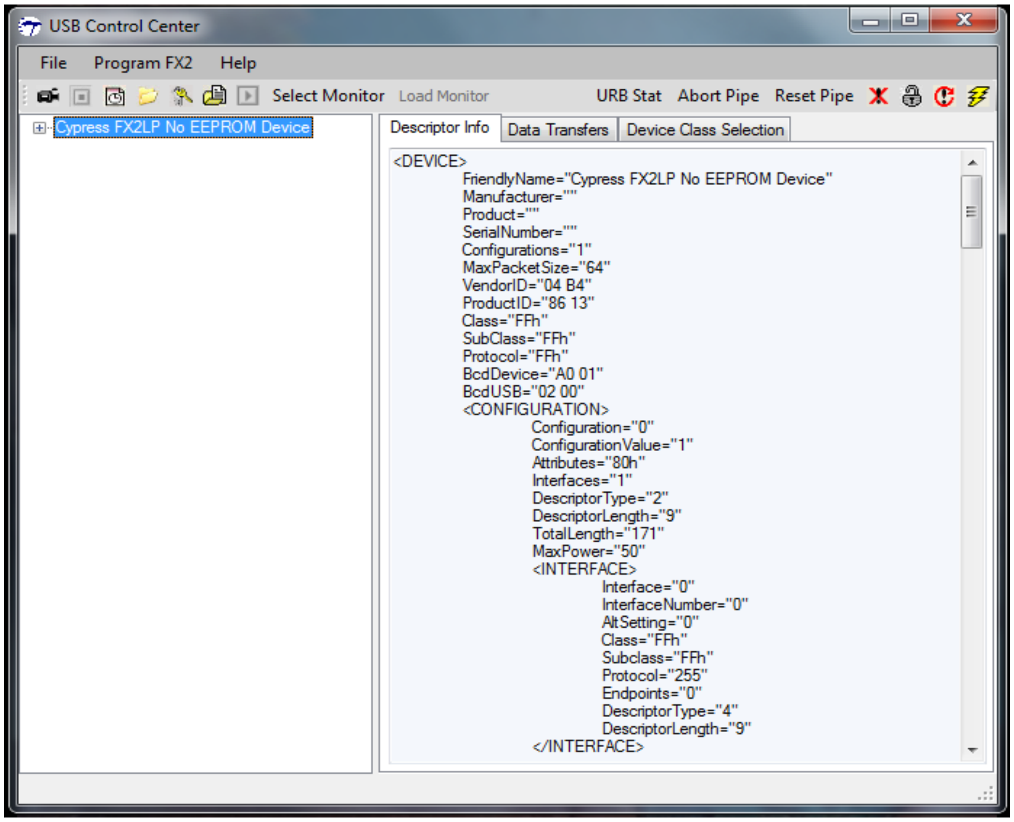
The following flowchart gives the overall flow of the firmware, summing up what was explained in the previous section.


START
10 ms timer intialized in Td_Init(); j = 0;
If (j < 10)
NO
YES
Write digit[j] to 7-segment display
Call timer_delay(0.4 sec), Increment j
Td_Poll() starts here
EP2 has data?
NO
YES
Count = EP2BCH: EP2BCL; i = 0;
Commit count bytes of EP6
NO
If(i++ < count)
YES
Assign value of current timer_tick to EP6[i].
If EP is not full; j = 0;
Is (j<10)
NO
YES
Write digit[j] to 7-segment display
Call timer_delay (100 * EP2[i]); Increment j

5 Running the Project

1. Download and install Cypress SuiteUSB 3.4. This installs the Control Center utility, shown in Figure 4. The ‘Descriptor Info’ tab in the utility provides information about the selected device.
2. Refer to the CY3684 DVK Quick Start Guide and ensure that the jumpers are properly configured. Now, connect the board to the PC with the EEPROM enable switch in the **No EEPROM** position. It will enumerate with the default internal descriptor. Use the *CyUSB.inf* file in the Drivers folder to bind with the device.
3. Download the *timer0.hex* file, which can be found in the attachment along with this application note at the location “\Timer in FX2LP\firmware\timer0.hex” using the Control Center.
4. Perform bulk transfers to EP2 to see its functionality. To do this, select the endpoint and type in the ‘Data to send’ under the ‘Data Transfers’ tab. For example, transfer ’01 02’ to EP2 and observe that 0 to 9 is displayed, first with 1s delays between digits and then with 2s delays.

Figure 4. Control Center



```
<DEVICE>
        FriendlyName="Cypress FX2LP No EEPROM Device"
        Manufacturer=""
        Product=""
        SerialNumber=""
        Configurations="1"
        MaxPacketSize="64"
        VendorID="04 B4"
        ProductID="86 13"
        Class="FFh"
        SubClass="FFh"
        Protocol="FFh"
        BcdDevice="A0 01"
        BcdUSB="02 00"
        <CONFIGURATION>
                Configuration="0"
                ConfigurationValue="1"
                Attributes="80h"
                Interfaces="1"
                DescriptorType="2"
                DescriptorLength="9"
                TotalLength="171"
                MaxPower="50"
                <INTERFACE>
                        Interface="0"
                        InterfaceNumber="0"
                        AltSetting="0"
                        Class="FFh"
                        Subclass="FFh"
                        Protocol="255"
                        Endpoints="0"
                        DescriptorType="4"
                        DescriptorLength="9"
                </INTERFACE>
```

Thus, depending on the values you send to the EP2 OUT endpoint, the step rate (inter-digit delay) of the counts displayed in the seven-segment display is modified. Table 1 shows the hex value that can be sent to EP2 and the corresponding delay; the hex values can range from 0x01 to 0xFF. In this example, you can also see the value of timer_tick that is assigned to EP6 and the committed in TD_POLL() function. This example only demonstrates how the timers in FX2LP can be used. Users can customize this accordingly.

6 Summary

AN1193 guides the firmware developers on using the timer interrupts in Cypress EZ-USB FX2LP-based applications by explaining the step by step details using an example code.

Document History

Document Title: AN1193 - Using Timer Interrupt in Cypress EZ-USB® FX2LP™ Based Applications

Document Number: 001-16828

Revision	ECN	Orig. of Change	Submission Date	Description of Change
**	1345183	YIS	08/03/2007	Recataloged application note.
*A	3176743	SSJO	02/18/2011	Updated Title. Updated for related projects. Updated in new template.
*B	3356912	GAYA	08/29/2011	Updated Source and Endpoint codes. Removed Appendix section. Updated ‘Running the Project’ section.
*C	3446128	GAYA	11/23/2011	Updated Source and Endpoint codes. Added flowchart. Updated ‘Running the Project’ section. Updated in new template.
*D	4496260	MDDD	09/08/2014	Updated in new template. Completing Sunset Review.
*E	5179367	MDDD	05/13/2016	Added reference to code examples webpage Updated the firmware flowchart Added related application note Updated the template
*F	5703573	AESATMP9	04/20/2017	Updated logo and copyright.
*G	5956283	HENA	11/03/2017	Added reference to FX2LP webpage and CY3684 DVK webpage Modified the ‘Introduction’ section Corrected the flowchart Added Figure 4 and related text in the ‘Running the Project’ section

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

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

©Cypress Semiconductor Corporation, 2007-2017. 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.