The following document contains information on Cypress products. The document has the series name, product name, and ordering part numbering with the prefix “MB”. However, Cypress will offer these products to new and existing customers with the series name, product name, and ordering part number with the prefix “CY”.

How to Check the Ordering Part Number

1. Go to www.cypress.com/pcn.
2. Enter the keyword (for example, ordering part number) in the **SEARCH PCNS** field and click **Apply**.
3. Click the corresponding title from the search results.
4. Download the Affected Parts List file, which has details of all changes

For More Information

Please contact your local sales office for additional information about Cypress products and solutions.

About Cypress

Cypress is the leader in advanced embedded system solutions for the world's most innovative automotive, industrial, smart home appliances, consumer electronics and medical products. Cypress' microcontrollers, analog ICs, wireless and USB-based connectivity solutions and reliable, high-performance memories help engineers design differentiated products and get them to market first. Cypress is committed to providing customers with the best support and development resources on the planet enabling them to disrupt markets by creating new product categories in record time. To learn more, go to www.cypress.com.

AN205508

F²MC-8FX Family, MB95200 Series Keyboard development using AD

This application note describes how to use the AD Conversion to design the keyboard and also describes the realization theories of AD conversion by examples.

Contents

1 Introduction....1
2 Introduction to AD Function....1
2.1 Function of AD Conversion....1
2.2 Jittering Elimination in Keyboard....1
3 Hardware Design....2
3.1 Hardware Circuit....2
3.2 Relationship between Key Status and Input Voltage....2
4 Resource Usage....3
4.1 8/10-bit A/D Converter Control Register 1 (ADC1)....3
4.2 8/10-bit A/D Converter Control Register 2 (ADC2)....4
4.3 8/10-bit A/D Converter Data Registers Upper/Lower (ADDH, ADDL)....5
4.4 AD Converter Process....5
5 Software Design....6
6 Sample Code....7
7 Additional Information....9
8 Document History....10

1 Introduction

There are three methods to develop a keyboard by MB95200 series MCU: external interrupt, AD conversion and matrix.

This document describes how to use the AD Conversion to design the keyboard.

This document also describes the realization theories of AD conversion by examples.

2 Introduction to AD Function

This chapter introduces the functions of using AD Conversion to realize the keyboard.

2.1 Function of AD Conversion

The function of AD Convertor is to convert an analog voltage (input voltage) value on an analog input pin to a 10-bit digital value. The MCU of MB95200 series has at least 2 analog input pins. When AD convertor is enabled, the analog input signal on the selected channel will be sampled and compared inside the AD convertor and finally converted to a digital value stored into the A/D converter data register (ADDH, ADDL).

2.2 Jittering Elimination in Keyboard

Jittering elimination has been one of important problems in keyboard design. In hardware circuit design, add a capacitor to the analog input pin to weaken the jittering, while in software design, re-confirm the data to avoid jittering occurred on a real keyboard.

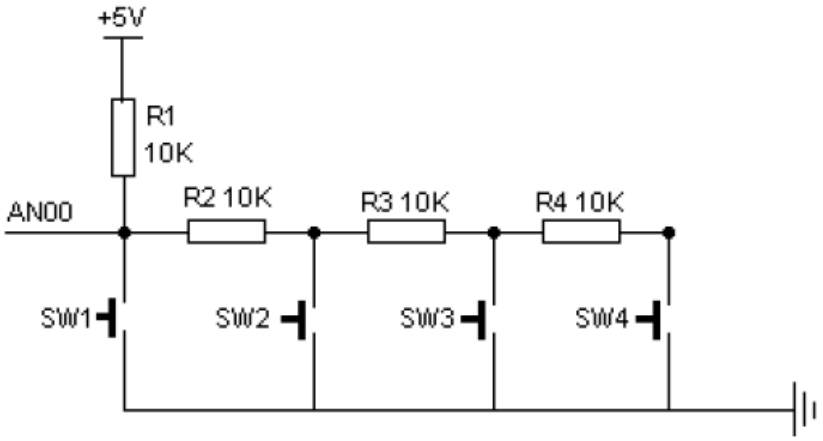
3 Hardware Design

This chapter introduces the hardware design to develop keyboard using AD Conversion.

3.1 Hardware Circuit

The resistors in the following hardware circuit are designed to divide voltage. And the analog input pin will get a different voltage from the circuit. The following circuit is for your reference:

Figure 1. Hardware Design for Keyboard Using AD



3.2 Relationship between Key Status and Input Voltage

Table 1. Relationship between Key Status and Input Voltage

Pressed key	AN00 Voltage	ADDL value	Virtual value
SW1	0 V	0x00	0x00>=ADDL<=0x05
SW2	2.5 V	0x7F	0x79>=ADDL<=0x85
SW3	3.3 V	0xAA	0xA1>=ADDL<=0xB2
SW4	3.75 V	0xBF	0xB5>=ADDL<=0xC8

4 Resource Usage

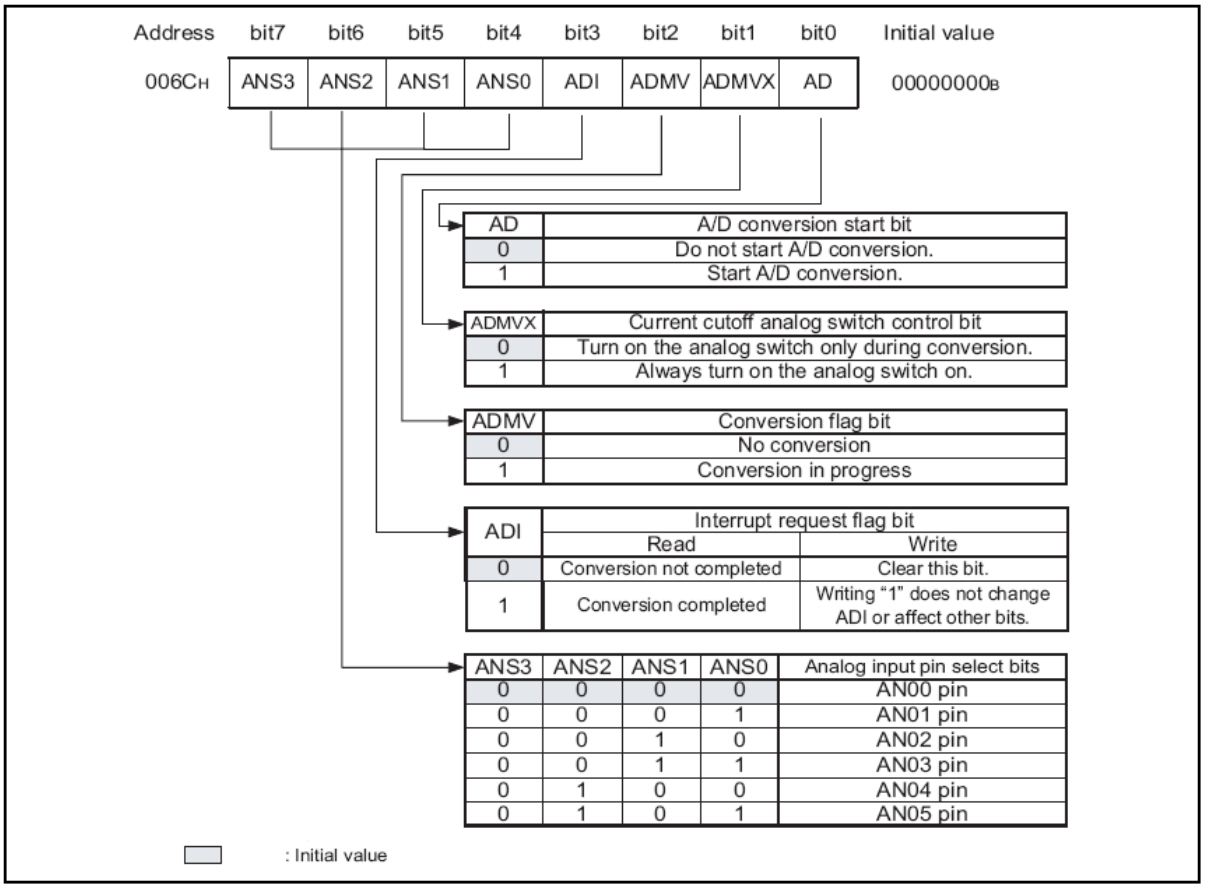
This chapter introduces the resource usage of AD Conversion.

Please refer to Chapter 17 of MB95200 Series Hardware Manual for detailed register setting.

4.1 8/10-bit A/D Converter Control Register 1 (ADC1)

The 8/10-bit A/D converter control register 1 (ADC1) is used to enable and disable individual functions of the 8/10-bit A/D converter, select an analog input pin and check the status of the converter.

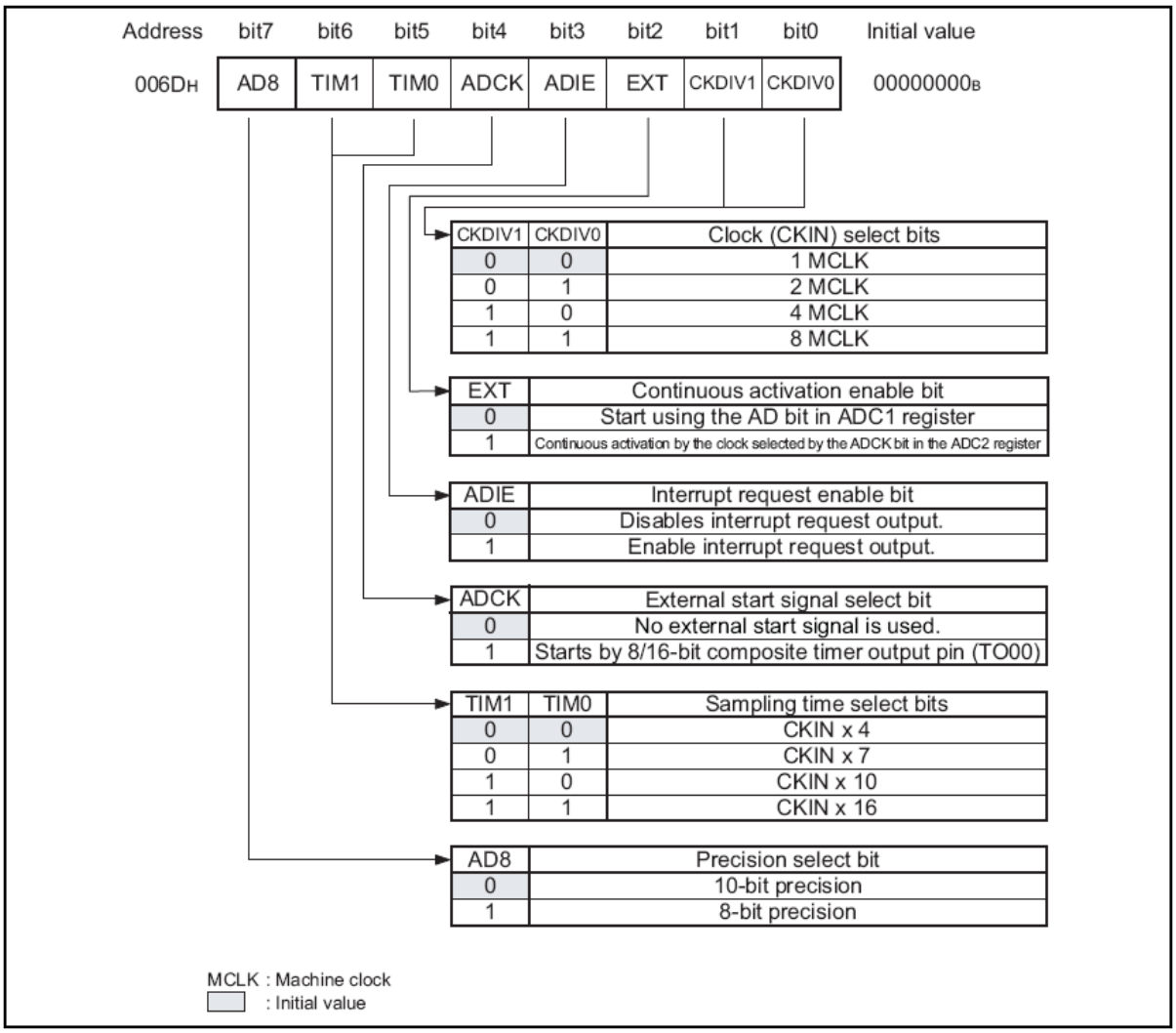
Figure 2. ADC1 Register



4.2 8/10-bit A/D Converter Control Register 2 (ADC2)

The 8/10-bit A/D converter control register 2 (ADC2) is used to control different functions of the 8/10-bit A/D converter, select the input clock, and enable and disable interrupts.

Figure 3. ADC2 Register



4.3 8/10-bit A/D Converter Data Registers Upper/Lower (ADDH, ADDL)

The 8/10-bit A/D converter data registers upper/lower (ADDH, ADDL) store the results of 10- bit A/D conversion during 10-bit A/D conversion. The upper two bits of 10-bit data are stored in the ADDH register and the lower eight bits the ADDL register.

Figure 4. AD Data Registers

ADDH	bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0	Initial value
Address $006E_H$	-	-	-	-	-	-	SAR9	SAR8	00000000_B
ADDL	bit15	bit14	bit13	bit12	bit11	bit10	bit9	bit8	Initial value
Address $006F_H$	SAR7	SAR6	SAR5	SAR4	SAR3	SAR2	SAR1	SAR0	00000000_B

The upper two bits of 10-bit A/D data correspond to bit1 and bit0 in the ADDH register and the lower eight bits bit15 to bit8 in the ADDL register.

If the AD8 bit in ADC2 register is set to "1", 8-bit precision is selected. Reading the ADDL register can obtain 8-bit data.

4.4 AD Converter Process

After the conversion time according to the register settings elapses, the results of conversion are finalized and stored in the ADDH and ADDL registers. After A/D conversion is completed and before the next A/D conversion is started, it's required to read A/D data registers (conversion results) and clear ADI flag bit (bit 3) in the ADC1 register.

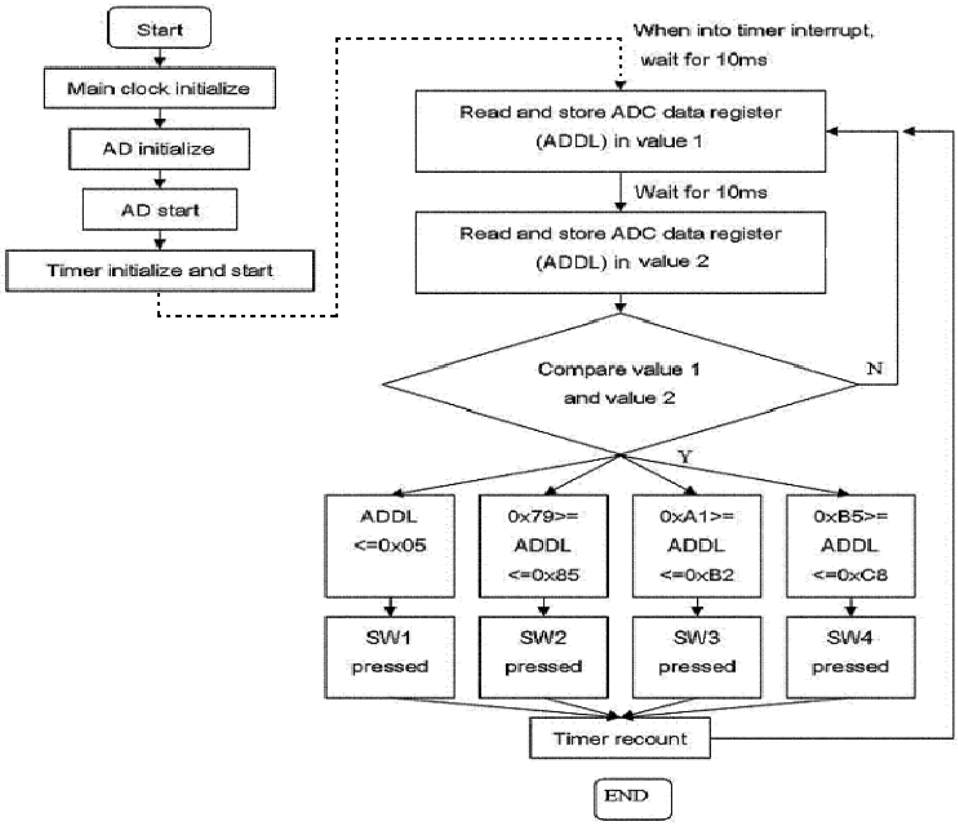
5 Software Design

This chapter introduces the software design to develop keyboard using AD conversion.

Once the AD convertor is enabled, it will capture the status information of keyboard. If the value (Distem0) belongs to the key-value, restored it, if not, the AD Convertor continues to sample. After this, stop the AD Convertor and make the system wait for about 20ms, then restart the AD Convertor and get a sample value (Distem1) again. Finally, judge this value to see if it is equal to Distem0. If it is different from Distem0, it is regarded that jittering happens; otherwise, judge which key has been pressed.

Below is the flow chart of this software design.

Figure 5. Software Design Flow Chart for Keyboard Development



6 Sample Code

This chapter illustrates keyboard development using AD Conversion.

The following example shows how to use the AD Conversion input pin AN00 to design a 4- key keyboard.

As discussed in chapter 5, if using AD Conversion to design a keyboard system, initialize the AD Conversion first, then AD conversion starts sampling, and gets an sample value, read this value and judge that if it belongs to the key-value. if does not, system continues to sample; if does, stop the AD Conversion and let the system waits for about 20 ms, and then restart AD Conversion, get an another sample value. If the two sample values are the same, it shows that one of the keys has been pressed; if not, it shows only a jittering happens.

```
#include "../MB95200_IO/mb95200.h". . .
void clock_Select(void)    // Main clock initialize
{
       SYCC = 0x00;
       WATR = 0x00;
       STBC = 0x01;
       SYCC2= 0xF4;
}
void syinit(void)
{
       DDR0_P00 = 0;        //P00 input enable
       ADC1 = 0x01;         //AN00 as input
       ADC2 = 0x80;         //8-bit resolution,1×MCLK
       ADC1_AD = 1;         //start A/D converter
       DDR0 = 0xF0;         //P4~P7 as output pin
       PUL0 = 0x00;
       AIDRL = 0xFE;
}
void timer_init(void)
{
       TMCR0 = 0x10;        // 16-bit
       T00CR0 = 0x81;       // interval timer with continuous mode
       T00CR1 = 0xA0;       // disable output, start timer
       T01DR = 0x27;        // 10000us
       T00DR = 0x10;
       while(STBC_MRDY);
}
__interrupt void Timer_Interrupt (void)
{
       T00CR1_IF = 0;
       T00CR1_IE = 0;
       timer_counter++;
       if(timer_counter==1) // Read and store ADDL in value 1
       {
       AD_Sample();
       }
```

```
        if(timer_counter==2)   // Read and store ADDL in value 2 and
                                       compare value and value 2
        {
                key_judge();
        }
        if(timer_counter>=3)   /Timer_counter restart
        {
        timer_counter=0;
        }
}
void AD_Sample(void)
{
        if(ADC1_ADI==1)
        {
        while(!ADC1_ADI);
        Distem0 = ADD_ADDL; // Read AD Sample value 1
        }
        ADC1_AD=1;             // Start AD again
}
void key_judge(void)
{
        if(ADC1_ADI==1)        // Read and store ADDL in value 2
        {
                while(!ADC1_ADI);
                Distem1 = ADD_ADDL;
        }
        if(Distem1==Distem0)   // Judge if value 1 and value 2 equality
        {
                if(Distem1 == Range_value0) // Key process
                {
                        DDR0_P04 = 1;
                        PDR0_P04 = 0;
                }
                if((Distem1 >= Range_value1)&&(Distem1 <= Range_value2))
                {
                        DDR0_P05 = 1;
                        PDR0_P05 = 0;
                }
                if((Distem1 >= Range_value3)&&(Distem1 <= Range_value4))
                {
                        DDR0_P06 = 1;
                        PDR0_P06 = 0;
                }
                if((Distem1 >= Range_value5)&&(Distem1 <= Range_value6))
                {
                        DDR0_P07 = 1;
                        PDR0_P07 = 0;
                }
        }
}
void main(void)
{
        InitIrqLevels();       //initialise Interrupt level and IRQ vector table
        __EI();                // global interrupt enable
        clock_Select();
        while(1)
        {
        syinit();
        timer_init();
        }
}
```

Note: This sample project is based on our starter-kit MB2146-410A. The target MCU is MB95F304K. AN0 is used as an analog input pin, and LED1 to LED4 an indicator. If keyboard hardware circuit as shown in chapter 3.1 is not provided, stop a jumper and connect VR3 to MCU so that VR3 can be used as an analog input. Then observe how LED changes in adjusting VR3.

7 Additional Information

For more Information on MB95200 Products, visit the following website:

http://www.cypress.com/8fx-mb95200

8 Document History

Document Title: AN205508 - F²MC-8FX Family, MB95200 Series Keyboard development using AD

Document Number: 002-05508

Revision	ECN	Orig. of Change	Submission Date	Description of Change
**	-	HUAL	02/24/2009	Initial release
*A	5264394	HUAL	05/13/2016	Migrated Spansion Application Note MCU-AN-500037-E-10 to Cypress format

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
Lighting & Power Control	cypress.com/powerpsoc
Memory	cypress.com/memory
PSoC	cypress.com/psoc
Touch Sensing	cypress.com/touch
USB Controllers	cypress.com/usb
Wireless/RF	cypress.com/wireless

PSoC® Solutions

cypress.com/psoc

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

Cypress Developer Community

Community | Forums |Blogs | Video |Training

Technical Support

cypress.com/support

PSoC is a registered trademark and PSoC Creator is a trademark of Cypress Semiconductor Corporation. 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

Phone : 408-943-2600
Fax : 408-943-4730
Website : www.cypress.com

© Cypress Semiconductor Corporation, 2009-2016. 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.