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.

AN205101

F²MC-8L/8FX/16LX16FX, MB89XXX / MB95XXX / MB90xxx / MB96Xxx, Performing SPI

It is said that SPI (Serial Peripheral Interface) is a simple communication between two digital devices. Because of no general specification and much different formats, SPI is not so simple at all. This application note describes the most common SPI formats and how to implement them on Cypress 8-, and16-Bit MCUs.

Contents

1 Introduction....1
2 SPI Formats....2
2.1 General SPI....2
2.2 SPI Clock (SCK)....2
2.3 Data direction....2
2.4 Communication speed....2
2.5 SPI protocols....3
3 Implementing SPI-Master Communication....5
3.1 Implementing SPI Formats with CPHA = 1....5
3.2 Implementing SPI Formats with CPHA = 0....5
4 Implementing SPI-Slave Communication....9
4.1 Preface....9
4.2 SPI-Slave Communication with LIN-U(S)ART....9
5 Overview of SPI-Master capability of several MCU families....10
5.1 SPI capability table....10
6 Appendix A....11
6.1 Software example archives....11
Document History....12

1 Introduction

It is said that SPI (Serial Peripheral Interface) is a simple communication between two digital devices. Because of no general specification and much different formats, SPI is not so simple at all.

This application note describes the most common SPI formats and how to implement them on Cypress 8-, and16-Bit MCUs.

2 SPI Formats

This chapter describes the most common SPI formats

2.1 General SPI

SPI is a single master single slave synchronous serial communication. Each data bit from and to the master has its own clock pulse. Therefore SPI uses at least three different signals:

1. SCK: Serial Clock
2. SI: Serial Input (Data from Slave to Master)
3. SO: Serial Output (Data from Master to Slave)

Some SPI formats use a fourth signal called “Chip Select” (CS) which enables the communication by the master. This signal can be positive or negative active.

2.2 SPI Clock (SCK)

Because there exists no common SPI specification, the timing of the SPI clock signal is device dependent and it seems so, that every producer uses its own timing.

For most SPI protocols four different settings are possible, which are mostly defined by the internal SPI master control settings: CPOL (Clock polarity) and CPHA (Clock phase).

- CPOL defines the active state of the SPI serial clock and thus the mark level.
- CPHA defines the clock phase in respect of the SO-data bit.

  There is no common classification of SPI protocols and SPI devices, which defines the settings of CPOL and CPHA in respect of the protocol itself. So this document defines the settings as follows:

- CPOL = 0: SPI clock has mark level “0”
- CPOL = 1: SPI clock has mark level “1”
- CPHA = 0: SPI clock is delayed by a half bit time in respect of SO-data bit
- CPHA = 1: SPI clock is synchronous to SO-data bits

2.3 Data direction

In the most SPI formats the serial data direction is MSB first.

2.4 Communication speed

The transfer rate of SPI communication is defined by the hardware itself. Mostly the speed is limited by set-up and hold timing of the data signals.

SPI is used in a wide communication speed range. The range reaches from some K Bits/s to some M Bits/s.

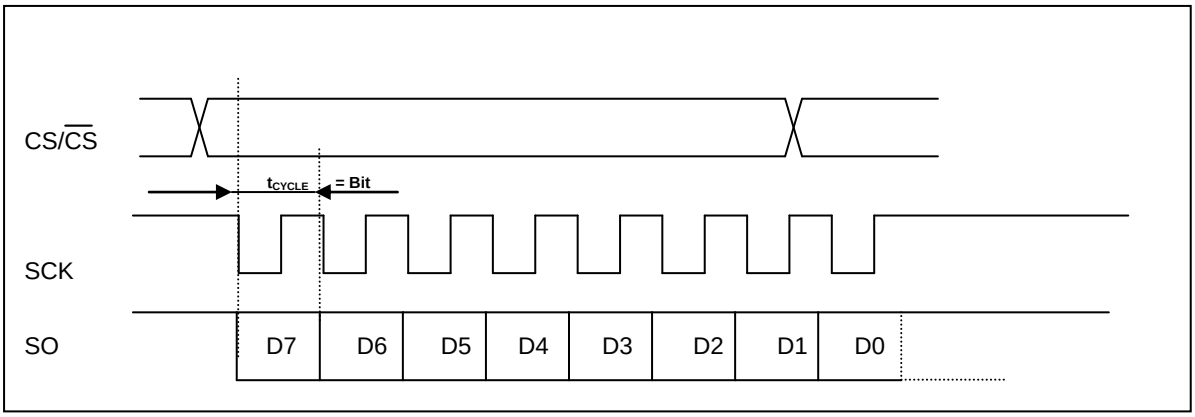
2.5 SPI protocols

This section describes the most common different SPI protocol timings.

2.5.1 CPOL = 1, CPHA = 1

This SPI format is in accordance with the “standard” serial synchronous format. The timing of the signals is shown in the illustration below:

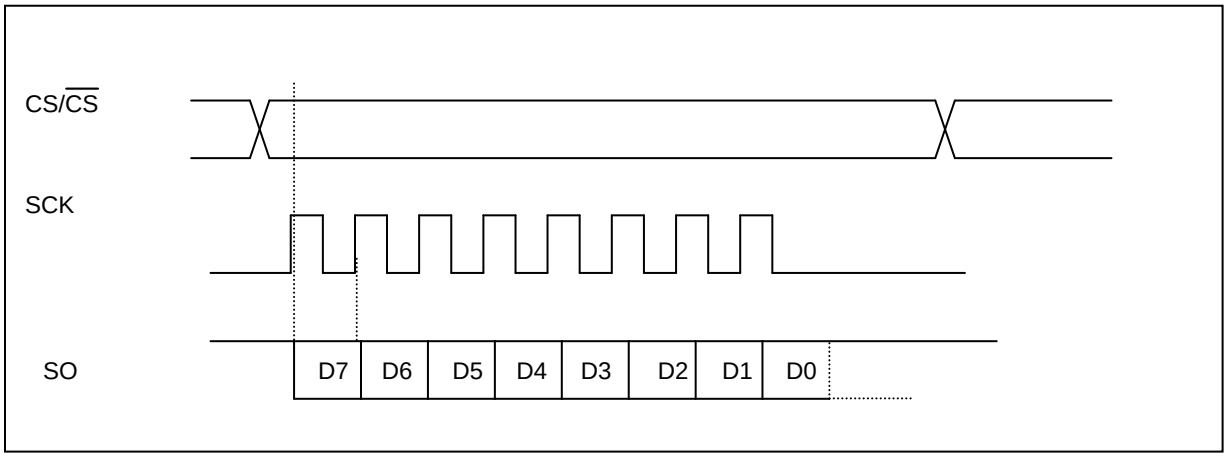
Figure 1. SPI clock in phase with data (mark level = “1”)



2.5.2 CPOL = 0, CPHA = 1

This SPI format is in accordance with the “standard” serial synchronous format with inverted clock signal.

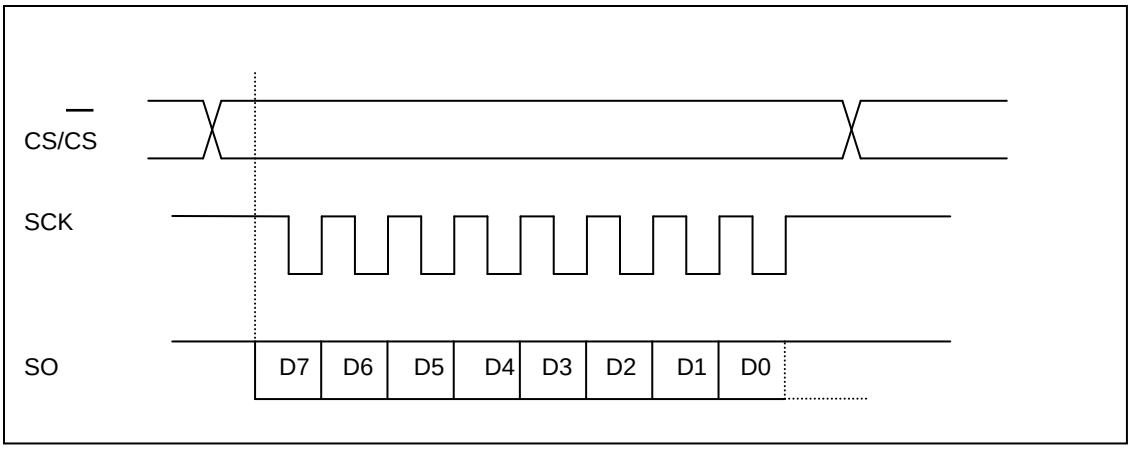
Figure 2. SPI clock in phase with data (mark level = “0”, inverted)



2.5.3 CPOL = 1, CPHA = 0

In this SPI format the clock signal is delayed by a half bit time to the “standard” synchronous format.

Figure 3. SPI clock delayed by half bit time (mark level = “1”)

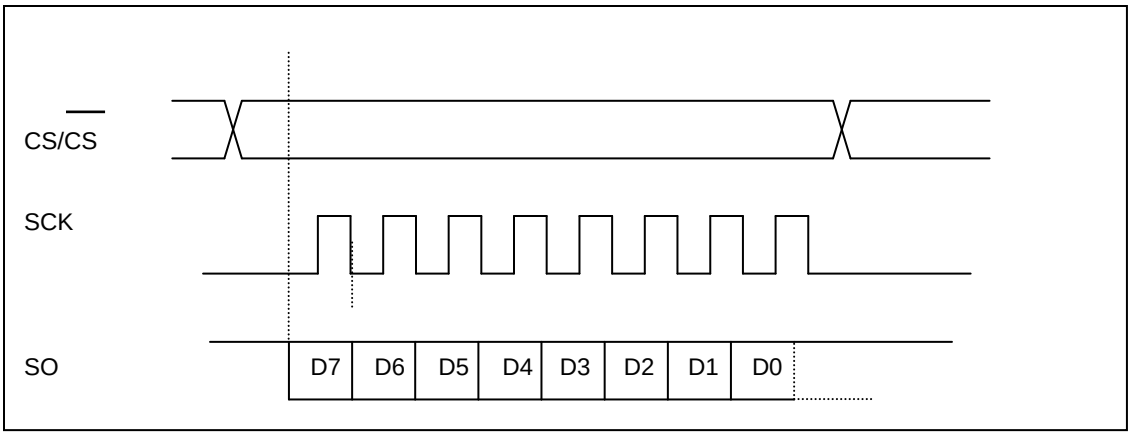


Note, that in this format the first data bit has to be set on SO *before* the first clock pulse occurs.

2.5.4 CPOL = 0, CPHA = 0

In this SPI format the clock signal is delayed by a half bit time to the “standard” synchronous protocol and the clock signal is inverted.

Figure 4. SPI clock delayed by half bit time (mark level = “0”, inverted)



Note, that in this format the first data bit has to be set on SO *before* the first clock pulse occurs.

3 Implementing SPI-Master Communication

This chapter describes how to implement the different SPI formats

3.1 Implementing SPI Formats with CPHA = 1

3.1.1 SPI with CPHA = 1 and CPOL = 1

This SPI format is available for all Cypress SIO interfaces.

Cypress UARTs with synchronous mode can also handle this format.

3.1.2 SPI with CPHA = 1 and CPOL = 0

This SPI format is available for all Cypress SIO interfaces with serial clock inversion (NEG-Bit).

Cypress UARTs with synchronous mode and serial clock inversion can also handle this format (NEG- or SCES-Bit).

3.2 Implementing SPI Formats with CPHA = 0

This format is available for the LIN-UART since MB90340 series. To implement this, programmer has to use synchronous mode and SCDE = 1 (Serial Clock Delay Enable) and SCES = x (Serial Clock Edge Select, means CPOL = x).

For all other SIOs or UARTs some tricks have to do.

3.2.1 SPI with SIO and CPHA = 0, CPOL = 1

The SIO can only handle SPI formats with CPHA = 1 in its normal operation mode. When external shift clock mode is selected, the SIO can be clocked by the toggling the port in respect of its SCK pin.

An undocumented feature is that the SIO can also be clocked by toggling the NEG bit of the SES register (Serial Edge Select). Thus it is possible to set the first data bit to SO before the clock is generated via the port pin.

The following C code shows an example of this method. Note, that the maximum communication speed depends on the execution time of the instructions (MCU speed) and the function `SPI_Byte` must not be broken by an Interrupt:

Example C code of CPHA = 0 and CPOL = 1 with SIO

```
void InitSIO (void)
{
   PDR4 = 0x80;        // SCK2 port 47 pin MB90540 series
   DDR4 = 0x80;        // SCK2 set to "1"

   SMCS_SOE = 1;       // Serial Output enable
   SMCS_SCOE = 0;      // Internal serial clock disable
   SES2_NEG = 1;       // Invert clock
   SMCS_STOP = 0;      // Reset STOP
   SMCS_SMD = 0x05;    // External shift clock mode
   SMCS_BDS = 1;       // Set MSB first
}
```

```
unsigned char SPI_Byte(data)
{
   SDR = data; // Write data to shifter
   SMCS_STRT = 1;      // Set communication start

   SES2_NEG = 0;       // Generate "internal clock" pulse
   SES2_NEG = 1;       // to set first data bit on SOT2

   PDR4_P47 = 0;       // generate 8 clock pulses on SCK2
   PDR4_P47 = 1;
   PDR4_P47 = 0;
   PDR4_P47 = 1;
   PDR4_P47 = 0;
   PDR4_P47 = 1;
   PDR4_P47 = 0;
   PDR4_P47 = 1;
   PDR4_P47 = 0;
   PDR4_P47 = 1;
   PDR4_P47 = 0;
   PDR4_P47 = 1;
   PDR4_P47 = 0;
   PDR4_P47 = 1;
   PDR4_P47 = 0;
   PDR4_P47 = 1;

   SES2_NEG = 0;              // "internal clock" pulse to fulfill
   SES2_NEG = 1;              //   SIO communication

   data = SDR;         // read shifter
   return data;
}
```

Note: A CS signal can easily performed using another port pin.

3.2.2 SPI with SIO and CPHA = 0, CPOL = 0

The method for CPOL = 0 is much like the method above (3.2.1).

The following C code shows an example of this method. Note, that the maximum communication speed also depends on the execution time of the instructions (MCU speed) and the function `SPI_Byte` must not be broken by an Interrupt:

Example C code of CPHA = 0 and CPOL = 0 with SIO

```
void InitSIO (void)
{
   PDR4 = 0x00;        // SCK2 port 47 pin MB90540 series
   DDR4 = 0x80;        // SCK2 set to "0"

   SMCS_SOE = 1;       // Serial Output enable
   SMCS_SCOE = 0;      // Internal serial clock disable
   SES2_NEG = 0;       // Normal clock
   SMCS_STOP = 0;      // Reset STOP
   SMCS_SMD = 0x05;    // External shift clock mode
   SMCS_BDS = 1;       // Set MSB first
}
```

Note: A CS signal can easily performed using another port pin.

3.2.3 SPI with UART synchronous and CPHA = 0, CPOL = 1

The method for SPI (CPHA = 0 and CPOL = 1) with a UART in synchronous mode is similar to the method used with the SIO.In this example the clock of the UART is also generated internal with toggling the NEG-Bit and generating an “external” clock via the Port state of the corresponding SCK pin.

Note, that this example only works for UARTs, which support a synchronous serial mode and a pre-scalar, not a reload counter. Those UARTs are identifiable by the CDC register (Clock Division Control).

Example C code of CPHA = 0 and CPOL = 1 with UART synchronous

```
void InitUART(void)
{
   SMR1_MD1 = 1;       // Set synchronous mode
   SMR1_MD0 = 0;
   SMR1_CS2 = 1;       // Set external clock source
   SMR1_CS1 = 1;
   SMR1_CS0 = 1;
   SMR1_SCKE = 0;      // External clock
```

Note: A CS signal can easily performed using another port pin.

3.2.4 SPI with UART synchronous and CPHA = 0, CPOL = 1

The method for SPI (CPHA = 0 and CPOL = 1) with a UART in synchronous mode is much like the example above (3.2.3)

Example C code of CPHA = 0 and CPOL = 1 with UART synchronous

```
void InitUART(void)
{
   SMR1_MD1 = 1;       // Set synchronous mode
   SMR1_MD0 = 0;
   SMR1_CS2 = 1;       // Set external clock source
   SMR1_CS1 = 1;
   SMR1_CS0 = 1;
   SMR1_SCKE = 0;      // External clock
   SMR1_SOE = 1;       // Serial Output Enable
   SES1_NEG = 0;       // Normal Clock
   SCR1_TXE = 1;       // Transmission Enable

   PDR4 = 0x00;        // SCK1 Port 44 pin on MB90540 series
   DDR4 = 0x10;        // SCK1 set to "0"
}

unsigned char SPI_Byte(data)
{
   SODR1 = data;              // Write data to transmission register

   DI();               // Disable all interrupts

   SES1_NEG = 1;              // Generate "internal clock" pulse
   SES1_NEG = 0;              //   to set first data bit on SOT1

   PDR4_P44 = 1;              // generate 8 clock pulses on SCK1
   PDR4_P44 = 0;
   PDR4_P44 = 1;
   PDR4_P44 = 0;
   PDR4_P44 = 1;
   PDR4_P44 = 0;
   PDR4_P44 = 1;
   PDR4_P44 = 0;
   PDR4_P44 = 1;
   PDR4_P44 = 0;
   PDR4_P44 = 1;
   PDR4_P44 = 0;
   PDR4_P44 = 1;
   PDR4_P44 = 0;
   PDR4_P44 = 1;
   PDR4_P44 = 0;

   SES1_NEG = 1;              // "internal clock" pulse to fulfill
   SES1_NEG = 0;              //   serial task

   EI();               // Enable all interrupts (optional)

   data = SIDR;               // read from reception register
   return data;
}
```

Note: A CS signal can easily performed using another port pin.

4 Implementing SPI-Slave Communication

How to implement spi slave communication

4.1 Preface

SPI is a 1 master to n slave communication. All described SPI modes are valid for both master and slave. The main difference is, that a slave does not generate a serial clock signal, but is clocked by the master.

4.2 SPI-Slave Communication with LIN-U(S)ART

The following program flow should be used to set the LIN-U(S)ART (to be found e. g. in the MB90340 series) in the synchronous serial SPI mode, so that its state is well defined for communication.

Register	CPOL = 1 CPHA = 1	CPOL = 0 CPHA = 1	CPOL = 1 CPHA = 0	CPOL = 0 CPHA = 0	Remark
SSR	0x00	0x00	0x00	0x00	Disallow interrupts, if previously enabled
SCR	0x04	0x04	0x04	0x04	Cut of all possible previous communication, clear possible reception errors
SMR	0xB1	0xB1	0xB1	0xB1	Mode2, One-to-one external clock, enable SOT pin
ECCR	0x20	0x20	0x30	0x30	Slave mode, adjust Clock phase
ESCR	0x00	0x01	0x00	0x01	Adjust Clock polarity
SMR	0xB9	0xB9	0xB9	0xB9	Reset LIN-U(S)ART
SSR*	0x03	0x03	0x03	0x03	Allow interrupts (optional)
SCR	0x03	0x03	0x03	0x03	Allow communication

* optional - only needed, if interrupts are used

Note, that the TXE and RXE control bits of the Serial Communication Register (SCR) should be enabled as the **last step** in the initialization process. Otherwise a correct behaviour of the LIN-U(S)ART cannot be guaranteed.

5 Overview of SPI-Master capability of several MCU families

The SPI-Master Capability Of UARTS In Several MCU Families

5.1 SPI capability table

Note, that this table gives an overview of the UART SPI-Master mode capability *without* “port clocking”.

MCU family	CPOL = 1	CPOL = 0	CPHA = 1	CPHA = 0
MB89210	X	X	X	X[1]
MB90330/335	X[2]	X[2]	X[2]	X[2]
MB90340/350/360	X	X	X	X
MB90390	X	X[3]	X	X[3]
MB90385	X		X	
MB90495	X		X	
MB90540	X		X	
MB90945	X	X[3]	X	X[3]
MB95xxx	X	X	X	X
MB96xxx	X	X	X	X

X = setting possible in serial synchronous mode of UART

0 delay only by 1 MCU clock, not by half serial communication bit time

1 only two combinations possible: CPOL = CPHA = 1 or CPOL = CPHA = 0

2 only UART3

6 Appendix A

Further Information

6.1 Software example archives

- 90340_uart_sync_spi_nm93cs46-v10.zip
- 90340_uart_sync_sio_max1286-v10.zip

Document History

Document Title: AN205101 - F²MC-8L/8FX/16LX16FX, MB89XXX / MB95XXX / MB90xxx / MB96Xxx, Performing SPI

Document Number:002-05101

Revision	ECN	Orig. of Change	Submission Date	Description of Change
**	-	MKEA	02/26/2003	MWi; First version
			10/04/2005	MWi; 8FX series, UART cross table, and appendix added
			04/05/2006	MWi; SPI-Slave chapter added
			06/11/2010	MWi; CPOL, CPHA logic corrected
*A	5062290	HKLO	04/06/2016	Converted Spansion Application Note “MCU-AN-3000002-E-V13” to Cypress format
*B	5873121	AESATP12	09/05/2017	Updated logo and copyright.

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, 2003-2017. 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.