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.

AN204769

F²MC-16FX Family, Direct Memory Access

This application note describes the functionality of the Direct Memory Access (DMA) module and gives some examples.

Contents

1 Introduction....1
1.1 Key Features....1
2 Direct Memory Access....1
2.1 Outline....1
2.2 Registers....4
3 DMA Examples....7
3.1 DMA Example with ADC....7
3.2 DMA Example with UART....10
4 Additional Information....11
5 Document History....12
Worldwide Sales and Design Support....13
Products....13
PSoC® Solutions....13
Cypress Developer Community....13
Technical Support....13

1 Introduction

This application note describes the functionality of the Direct Memory Access (DMA) module and gives some examples.

The DMA is mainly used to transfer data between a source and a destination memory location without any CPU load. The direction can be from resource (peripheral) to memory and vice versa. This transfer can be a single transfer or multiple transfers from single address or an address area. DMA is always triggered by a resource interrupt and does not interrupt the CPU until the transfer has ended.

1.1 Key Features

- 8 bit or 16 bit Transfer
- Transfer from resource (registers in bank `0x00`) to complete address Area (24 bit) and vice versa
- Single Transfer or Multiple Transfers with possibility of update (increment or decrement) of source and destination address
- DMA can be stopped by STOP request (supported by UART-Receive interrupt, in case of receive error)

  It should be noted that the DMA should only be used with such peripheral those interrupts can be cleared by DMA. The same is indicated by “Yes” in the “DMA can clear” column in the interrupt vector table in the datasheet.

2 Direct Memory Access

The Basic Functionality of the DMA Module

2.1 Outline

The DMA module is used to transfer data from one address (space) to another without involving the CPU. The address space includes RAM, ROM or resource registers.

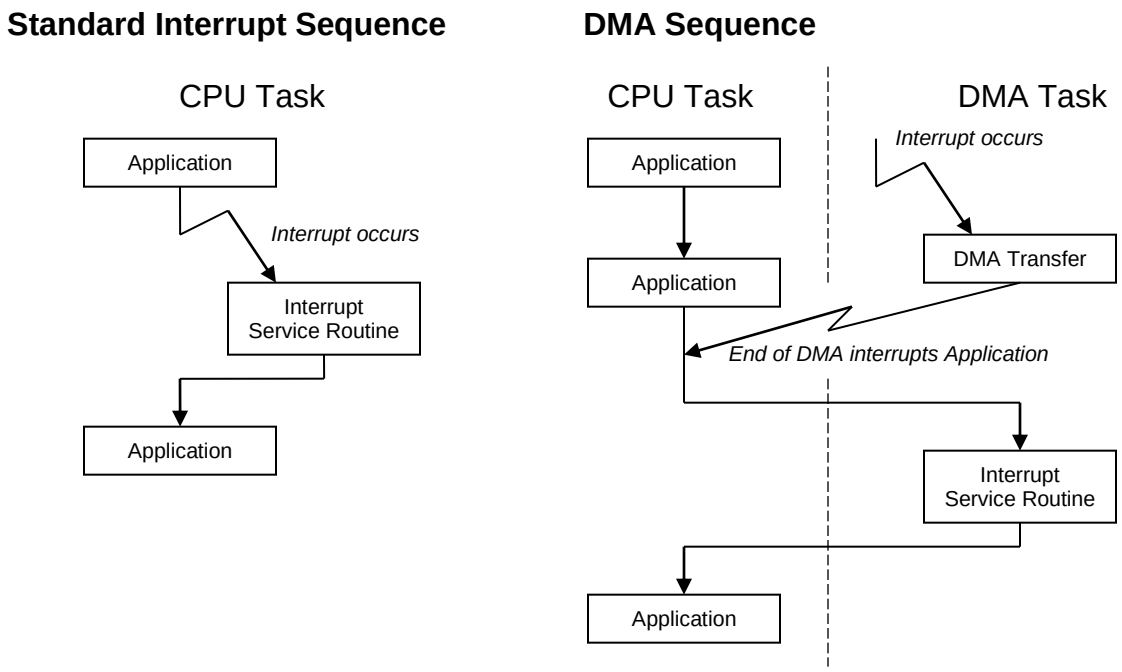
Three conditions must be fulfilled before a DMA request is served:

- IRQ must match with a DMA channel configuration
- The channel which matches must be enabled
- The `DTE` (Data Transfer End) bit of the matching channel must be zero

2.1.1 Single Transfer

The following diagram shows a standard interrupt sequence and a single transfer DMA sequence:

Figure 1. Single Transfer

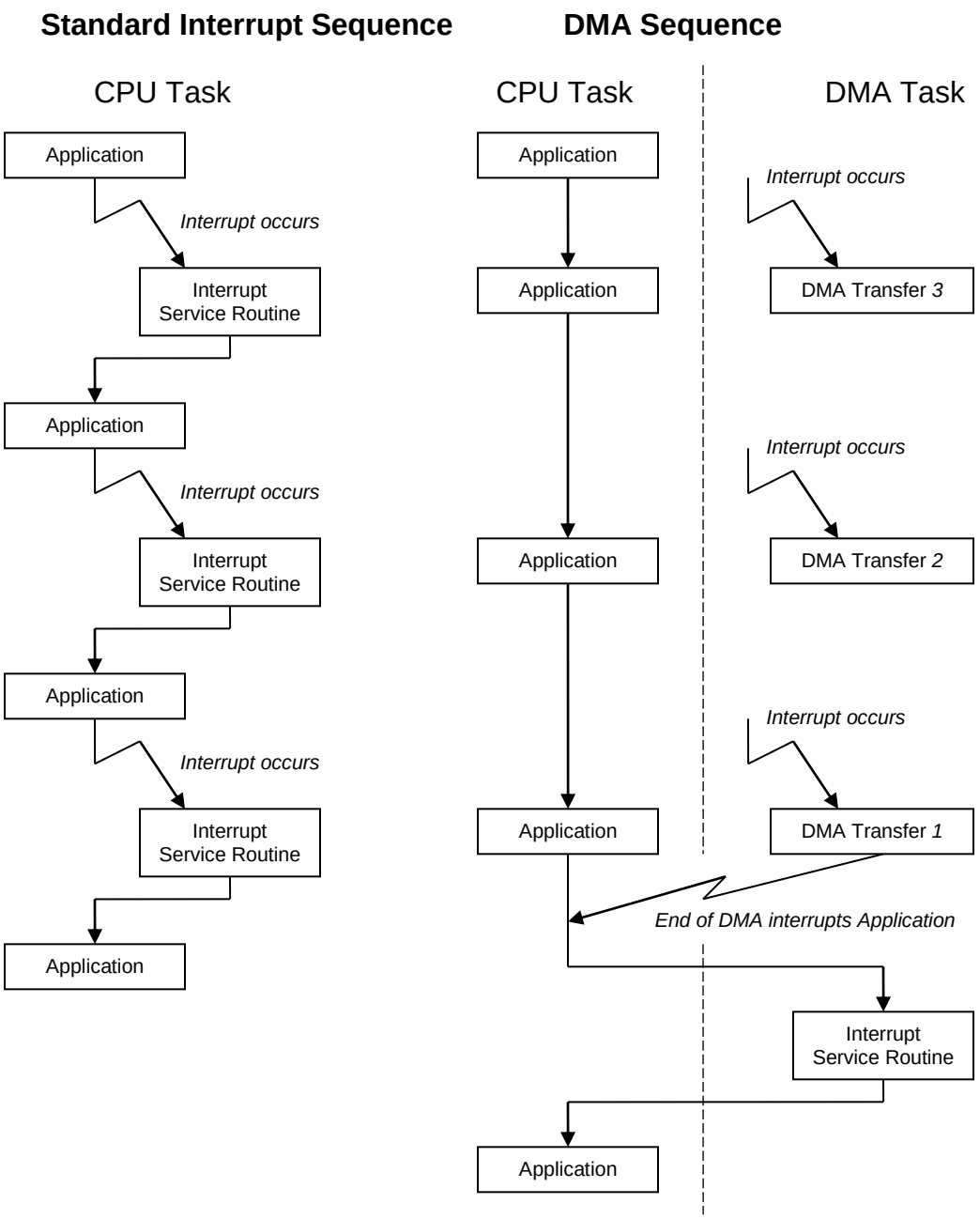


For a single transfer the DMA does not offer a real advantage against a standard interrupt.

2.1.2 Multiple Transfers

The following diagram shows a standard interrupt sequence and a multiple transfer DMA sequence:

Figure 2. Multiple Transfers



For a multiple transfer the DMA advantage is obvious. During Transfer *3* and *2* the application is not interrupted as in the standard interrupt sequence. Only after the Transfer *1*, the interrupt service routine of the resource which is generating the interrupts is executed.

However, it should be also noted that the DMA shares the same bus as of CPU, hence during the time DMA is engaged in the data transfer the CPU is either waiting for the DMA to finish the transfer or it is executing the code from the pre-fetch queue.

2.2 Registers

2.2.1 DMA Interrupt Request Select Register (DISELn)

This register specifies the IRQ number of the corresponding peripheral that is used to trigger DMA transfer on the specific channel.

Table 1. DISELn

Bit No.	Name	Explanation	Initial Value	Value	Operation
7 … 4	IS7 … IS4	Interrupt Number for Channel 0 –15	0	–	Assigns Interrupt Number to DMA Channel
3, 2	IS3, IS2		1	–	
1, 0	IS1, IS0		0	–	

It is recommended to maintain all unused DISEL registers to 0x0C, if DMA is used.

2.2.2 DMA Status Register (DSRL/H)

This 16 bit register contains the DMA transfer end interrupt request flags for each of the DMA channel. That means the bit 0 is corresponding to the request flag for DMA channel 0 and bit 15 is corresponding to the request flag for DMA channel 15.

Table 2. DSRL/H

Bit No.	Name	Explanation	Initial Value	Value	Operation
15 … 0	DTE15 … DTE0	Data Transfer End Interrupt (Channel 0 – 15)	0	0	Read: No Interrupt for Channel n Write: Clear Bit
				1	DMA transfer is completed or stopped for Channel n

Caution:

For cyclic interrupts such as ADC interrupts, it is not enough to clear the corresponding DTE bit for ending the DMA transfer (Condition described in 2.1 is fulfilled again)!

To end the DMA transfer please also disable the interrupts of the resource itself.

2.2.3 MA Enable Register (DERL/H)

This 16 bit register contains the DMA enable bits for each of the DMA channel. That means the bit 0 is corresponding DMA channel 0 and bit 15 is corresponding to DMA channel 15.

Table 3. DERL/H

Bit No.	Name	Explanation	nitial Valu	Value	Operation
15 … 0	EN15 … EN0	DMA Enable (Channel 0 – 15)	0	0	No DMA. Resource Interrupts are enabled
				1	Resource Interrupts are passed to the DMA controller until last transfer

2.2.4 DMAStop Status Register (DSSRL/H)

This 16 bit register contains the DMA Stop status flags for each of the DMA channel. That means the bit 0 is corresponding DMA channel 0 and bit 15 is corresponding to DMA channel 15.

Table 4. DSSRL/H

Bit No.	Name	Explanation	nitial Valu	Value	Operation
15 … 0	STP15 … STP0	Data Stop Status (Channel 0 – 15)	0	0	Read: No Stop request occurred for channel n Write: Clear Bit
				1	DMA transfer is stopped due to Stop request issued by resource peripheral for Channel n

The DMA Stop request feature is only provided by the UART Receive interrupt. If the SE bit of DMACS register of the corresponding DMA channel is set and if there is an error while the UART reception, then the ongoing DMA transfer would be stopped and the STPx bit for the corresponding DMA channel will be set.

2.2.5 DMA Descriptor

Each of the DMA channels has an 8 byte descriptor. The following table shows the structure of a single descriptor:

Table 5. DMA Descriptor

Address	Descriptor Register
0x00107 + 8 * ch	Upper 8 Bits of Data Counter (DCTH)
0x00106 + 8 * ch	Lower 8 Bits of Data Counter (DCTL)
0x00105 + 8 * ch	Upper 8 Bits of I/O Register Address Pointer (IOAH)
0x00104 + 8 * ch	Lower 8 Bits of I/O Register Address Pointer (IOAL)
0x00103 + 8 * ch	DMA Control Register (DMACS)
0x00102 + 8 * ch	Upper 8 Bits of Buffer Address Pointer (BAPH)
0x00101 + 8 * ch	Middle 8 Bits of Buffer Address Pointer (BAPM)
0x00100 + 8 * ch	Lower 8 Bits of Buffer Address Pointer (BAPL)

In the above table the value of ch can be between 0 to Number of DMA channels-1.

Note that all the DMA descriptor registers have undefined initial values and are read and writeable.

Data Count Register (DCTL/H)

This 16 bit register holds the number of data transfers in terms of bytes per DMA. The decrement value of this register depends on the transfer type and transfer cycles are dependent on the alignment of the I/O and buffer address.

The following table describes various scenarios for the DMA transfer:

Table 6. Transfer Type And Descriptors

Configuration/Status				Transfer Type		Post Transfer Update of			Transfer Cycles
BW	DCT	IOA	BAP	I/O	Buffer	DCT	IOA (DMACS_IF = 1)	IOA (DMACS_BF = 1)	
0	> 0	-	-	Byte	Byte	-1	+1	+/-1	2
1	> 1	Even	Even	Word	Word	-2	+2	+/-2	2
1	> 1	Even	Odd	Word	2x Byte	-2	+2	+/-2	3
1	> 1	Odd	Even	2x Byte	Word	-2	+2	+/-2	3
1	> 1	Odd	Odd	Byte	Byte	-1	+1	+/-1	2
1	= 1	-	-	Byte	Byte	-1	+1	+/-1	2
0	= 0	-	-	Byte	Byte	-1	+1	+/-1	2 (x 65536)

As shown in the 2nd to last row of the above table, even if the BW bit of DMACS register is configured for the word transfer, since the DCT is written with a value of 1, a single byte transfer would be performed.

The last row shows, that for a full 64 KBytes data block to be transferred the DCT should contain the count value "0" for byte transfer.

Please note that a count value "0" means the maximum of possible DMA transfers (65535 for byte and 32768 for word) – not "no transfer"!

I/O Register Address Pointer (IOAL/H)

This 16 bit register holds the address of the I/O register address. The upper 8 bits (A16-A23) are "0". This allows an address space from 0x000000 to 0x00FFFF. The post transfer update of this register is described in the above Table 6

DMA Control Register (DMACS)

This register controls the overall DMA transfer functionality.

Table 7. DMACS

Bit No.	Name	Explanation	nitial Valu	Value	Operation
15, 14	-	*Reserved*	X	-	*Reserved. Always write "0"*
13	BPD	Buffer Pointer Decrement Bit	X	0	BAP increments after each transfer, if BF=0
				1	BAP decrements after each transfer, if BF=0. If BPD=BW=1, please set even values to IOA, BAP, and DCT
12	IF	IOA update/fixed Selection Bit	X	0	IOA increments after each transfer
				1	IOA stays fixed after transfer
11	BW	Byte/Word Selection Bit	X	0	Enable Byte Transfer
				1	Enable Word Transfer
10	BF	BAP update/fixed Selection Bit	X	0	BAP is updated after transfer
				1	BAP stays fixed after transfer
9	DIR	Data Transfer Direction Bit	X	0	Transfer from IOA to BAP address
				1	Transfer from BAP to IOA address
8	SE	DMA Stop Request Enable Bit	1	0	No reaction on Stop Request
				1	DMA is stopped on Stop Request from resource (UART-Rx)

Buffer Address Pointer (BAPL/M/H)

This 24 bit register holds the address of the buffer. The whole memory area can be accessed.

3 DMA Examples

Examples for Direct Memory Access

3.1 DMA Example with ADC

The following example code shows how to set up the DMA with the ADC. The ADC converts channel 0 to 7 and the 8 bit results are transferred via DMA0 to the global array `unsigned char adc_array[8]`. After this the ADC interrupt is executed and the transfer is initialized again.

Main.c

```
__far unsigned char adc_array[8]; // Declared as __far for 24 bit addressing.
                                  // If declared as __near, BAPHn must be set
                                  // to DATA bank (usually 0x00).
void init_adc(void)
{
    ADCS = 0xA020;    // single, 8 bit
    ADSR = 0x9007;    // 24 cycle sampling, 88 cycle conversion, channel 0-7
    ADER0 = 0xFF;     // enable ADC pins
}

void init_dma(void)
{
    DISEL0 = 76;       // ADC interrupt number for MB9634x Series
    DCTH0 = 0x00;      // 8 Bytes
    DCTL0 = 0x08;
    IOAH0 = (unsigned char) &ADCR >> 8; // I/O Bank 00
    IOAL0 = (unsigned char) &ADCR & 0xFF;
    DMACS0 = 0x10; // no IOA update, BAP increment, byte transfer, IOA -> BAP

    BAPH0 = (__far unsigned long) &adc_array >> 16;
    BAPM0 = (__far unsigned long) &adc_array >> 8;
    BAPL0 = (__far unsigned long) &adc_array & 0xFF;

    DSR = 0x0000;// Clear transfer end interrupt, if any
    DER = 0x0001;     // DMA 0 enable
}

void main(void)
{
    InitIrqLevels();
    __set_il(7);      // allow all levels
    init_dma();
    init_adc();

    __EI();           // globally enable interrupts

    ADCS_STRT = 1;    // start ADC

    while(1);         // Do nothing: DMA, ADC interrupt do the rest
}

__interrupt void irq_adc(void)
{
    ADCS = 0xA020;  // Clear ADC interrupt
    DSR = 0x0000;   // Clear DMA end request

    init_dma();

    ADCS_STRT = 1;  // restart ADC
}
```

Vectors.c

Please note, that the corresponding interrupt vector and level has to be defined in the *vectors.c* module of our standard template project.

```
void InitIrqLevels(void)
{
. . .

ICR = ((76 & 0xFF) << 8) | 6;  // Priority Level 6 for ADC of MB9634x
                               // Series
. . .
}

__interrupt void irq_adc (void);         // Prototype

   . . .

#pragma intvect irq_adc          76      // ADC of MB9634x Series

   . . .
```

Note that this example has no direct result output, but the converted channels can be observed during runtime with an emulation system and by watching the global array

`unsigned char adc_array[8].`

3.2 DMA Example with UART

The following example code shows how to configure the DMA with the UART0. If the data is received on UART0 receive buffer `RDR0` then the DMA channel 0 transfers the same data to the UART0 transmit buffer `TDR0`. The transfer takes places unless 50 receptions or if there is any error in the reception. After that the receive interrupt clears DMA transfer end flag and also disables the reception interrupt.

Main.c

```
void init_uart0(void)
{
    PIER08_IE2 = 1;  //Enable UART0 RX
    DDR08_D2 = 0;    //Enable UART0 RX

    BGR0 = 1666;     // 9600 baud at 16MHz CLKP1
    SCR0 = 0x17;     // 8 bit, clear reception errors, Tx & Rx enabled

    ESIR0 = 0x01;    //Disable USART automatic interrupt clear,
                     // Clear TDRE and RDRF flags
    SSR0_RIE = 1;    // Enable receive interrupt
    SMR0 = 0x0D;     // Mode 0, Reset Counter, Reset UART, SOT0 enabled
}

void init_dma(void)
{
    unsigned long temp;
    DISEL0 = 79;     // UART0 receive interrupt number for MB9634x Series

    DCT0 = 0x0032;    // 50 Bytes
    IOA0 = (unsigned int)&RDR0;   //Source
    temp = (unsigned long)&TDR0;  //Destination
    BAPL0 = (unsigned char) temp;
    temp >>= 8;
    BAPM0 = (unsigned char) temp;
    temp >>= 8;
    BAPH0 = (unsigned char) temp;
    DMACS0 = 0x15;   // no IOA & BAP update, byte transfer, IOA -> BAP, DMA
           // Stop request by UART if error in Receive
    DSR = 0x0000;// Clear transfer end interrupt, if any
    DER = 0x0001;    // DMA 0 enable
}

void main(void)
{
    InitIrqLevels();
    __set_il(7);     // allow all levels
    init_dma();
    init_uart0();

    __EI();          // globally enable interrupts

    while(1);        // Do nothing: DMA, UART0-Rx interrupt do the rest
}

__interrupt voidUART0_RXISR(void)
{
    DSR = 0x0000;    // Clear DMA end request
    ESIR0_RDRF = 0;  // Clear RDRF flags
    SCR0_CRE = 1;    //Clear all error flags
    SSR0_RIE = 0;    // Disable reception interrupt
}
```

Vectors.c

Please note, that the corresponding interrupt vector and level has to be defined in the *vectors.c* module of our standard template project.

```
void InitIrqLevels(void)
{
       . . .

       ICR = ((79 & 0xFF) << 8) | 6; // Priority Level 6 for UART0-Rx of MB9634x
          // Series
   . . .
}

__interrupt voidUART0_RXISR(void);      // Prototype

   . . .

#pragma intvect UART0_RXISR79     // UART0-Rx of MB9634x Series

   . . .
```

Note:

For usage of consecutive DMA transmission via UART please read the application note:

AN205498 - F²MC 16FX Family, USART

4 Additional Information

Information about Cypress Microcontrollers can be found on the following Internet page:

http://www.cypress.com/cypress-microcontrollers

The software examples related to this application note is:

96340_dma_uart0

96340_ppg_rlt_adc_dma

96340_adc_dma

5 Document History

Document Title: AN204769 - F²MC-16FX Family, Direct Memory Access

Document Number: 002-04769

Revision	ECN	Orig. of Change	Submission Date	Description of Change
**	-	MKEA	07/24/2006	Initial Release
			03/23/2007	Reviewed the document and updated with review findings
			08/14/2007	Updated with review findings from PHu
			02/22/2010	UART TX DMA reference to other application note added
			02/25/2010	DCT remarks added, conditions for DMA request added
			09/05/2013	ADC IRQ number corrected
*A	5056457	MKEA	12/22/2015	Migrated Spansion Application Note from MCU-AN-300220-E-V15 to Cypress format
*B	5836253	AESATMP9	07/28/2017	Updated logo and copyright.
*C	6036637	NOFL	01/18/2018	Updated logo. Updated links. Updated Sales page and Copyright year.

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 MCU

Cypress Developer Community

Community | 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, 2006-2018. 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. No computing device can be absolutely secure. Therefore, despite security measures implemented in Cypress hardware or software products, Cypress does not assume any liability arising out of any security breach, such as unauthorized access to or use of a Cypress product. In addition, the products described in these materials may contain design defects or errors known as errata which may cause the product to deviate from published specifications. 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.