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.



THIS SPEC IS OBSOLETE

Spec No: 002-05054

Spec Title: AN205054 - UART Multimastermode, MB90F540/5 Series with implementation examples

Replaced by: None

AN205054

UART Multimastermode, MB90F540/5 Series with implementation examples

This application note describes how several microcontroller units communicate together in a master and slave system via Uart. It describes an example implementation for the MB90540 series and gives general hints, which Uart macros can be used for master/slave communication.

Contents

1 Introduction....1
2 Software configuration of the microcontroller 3
3 Hardware configuration: 4
4 Function of the Master-Slave system: 5
5 Conclusion:....7
6 Appendix 8
Document History....11

1 Introduction

This application note describes how several microcontroller units communicate together in a master and slave system via Uart. It describes an example implementation for the MB90540 series and gives general hints, which Uart macros can be used for master/slave communication.

In many applications there is one part which is managing and controlling everything and there are some other parts which are reacting only of requests by the main system. A master and slave system is like that.

The Uart (**U**niversal **A**synchronous **R**eceiver **T**ransmitter) macro is able to transmit from an interface to another one. The Uart macro has two independent buffers to receive and transmit data. Buffer size is one Byte for each buffer. The Uart buffer can be read or write with an interrupt routine or by polling a flag.

Cypress supports in the main two different Uart macros. One Uart can be used only for master purposes, because the resource cannot identify the format of the received data. The other Uart macro can operate in both master and slave mode. A flag identifies the received data, ether it is data or address. The Table 1 shows all devices and their possible use in such a Master-Slave system.

OBSOLETE

Table 1 . Overview of the 16 Bit Family

Device	Master only	Master & Slave	Number of Uart's
470		X (Uart0)	1
495	X (Uart1)		2
520	X (Uart1)		1
540/545	X (Uart1)	X (Uart0)	2
550	X (Uart1)		1
560	X (Uart1)		2
570	X (Uart1)		2
580	X (Uart0-4)		5
595	X (Uart1)	X (Uart0)	2
590		X (Uart0-2)	3
610A	X (Uart1)		3
620A	X (Uart1)		1
630A	X (Uart1)		2
640A	X (Uart1)		2
650A	X(Uart1)		1
660A	X (Uart1)		1
670/675	X (Uart1)	X (Uart0)	2

Even when several microcontrollers are used and they should work together the easiest solution and may be cheapest is a Master and Slave system. It can be considered as a small network with one host and several clients. No protocol to communicate between the different devices is necessary, so it is up to the user to implement a protocol for the application.

The master, which is the active part, frequently asks all or one of his slaves for information or asks if new data is available. If the master has finished his request to one are even all connected slaves, the slaves will be released until the next request appears by the master.

Slaves can be from a huge range of devices. They can be MCU's, peripheral devices and sensors, which can be addressed.

This application shows how the Cypress MCU's can be connected together to master and slave system.

OBSOLETE

2 Software configuration of the microcontroller

The first step to configure the master and the slave Uart is shown in Figure 1. It shows the setting of the different Uart registers that have to be done. The Table 2 shows all available modes for the Uart0 of the MB90540 series. For this application only mode 2 and 3 are of interest.

Table 2. Uart0 Operation Modes

Mode	MC1	MC0	Data Length
0	0	0	7 (6)
1	0	1	8 (7)
2*	1	0	8 + 1
3	1	1	9 (8)

The figures enclosed in parentheses indicate the data length with parity.

*: Mode 2 is used when a number of slave CPUs are connected to a single host.

Mode 2 is used when a number of slaves CPUs are connected to a single host CPU. This setting is used only for the slave CPUs. The parity function cannot be used in this mode. The data length must be set to 9 bits in both modes.

Mode 3 is used for the master and the communication between the selected slave.

```
void InitUart_Master(void)
{
       /* initialize UART0 */
       UMC0 = 0x39;        /* serial mode control register (001 11001) */
       USR0 = 0x0C;        /* status register (00011100) (master)        */
       URD0 = 0x4C;        /* rate and data register  (01001100)      */
       DDR4_D40 = 1;       /* set SOT0 working                        */
}
```

Figure 1. Initialization of the Uart as master or slave

```
void InitUart_Slave(void)
{
       /* initialize UART0 */
       UMC0 = 0x28;  /* serial mode control register (00101000)          */
       USR0 = 0x08;  /* status register (00011000) only in receive mode  */
       URD0 = 0x4C;  /* rate and data register  (01001100)                      */
}
```

The master operates in mode 3 all the time. In this mode the master can decide if the transferred byte will be an address or data. Therefore, the data bit (D8) of the rate and data register (URD0) has to be used.

OBSOLETE

If the byte should be an address this bit should be set to ‘1’ and ‘0’ if the transferred byte should be data.

The slave is by default in mode 2. If the slave is selected with his address, transfer must be changed to mode 3 by software. Than perform the communication or data exchange between master and slave and changes back to mode 2 after communication has finished. The slave can identify the bit D8 in his receive data register if it is data or address.

All this will be described in more details late or see the flowchart at the appendix.

The Figure 2 shows the transfer format of mode 3.

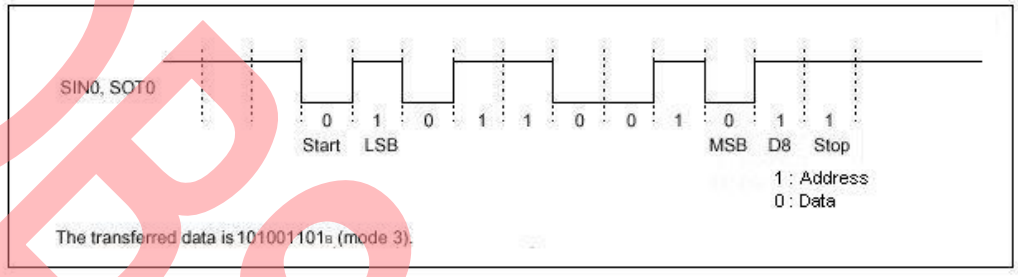


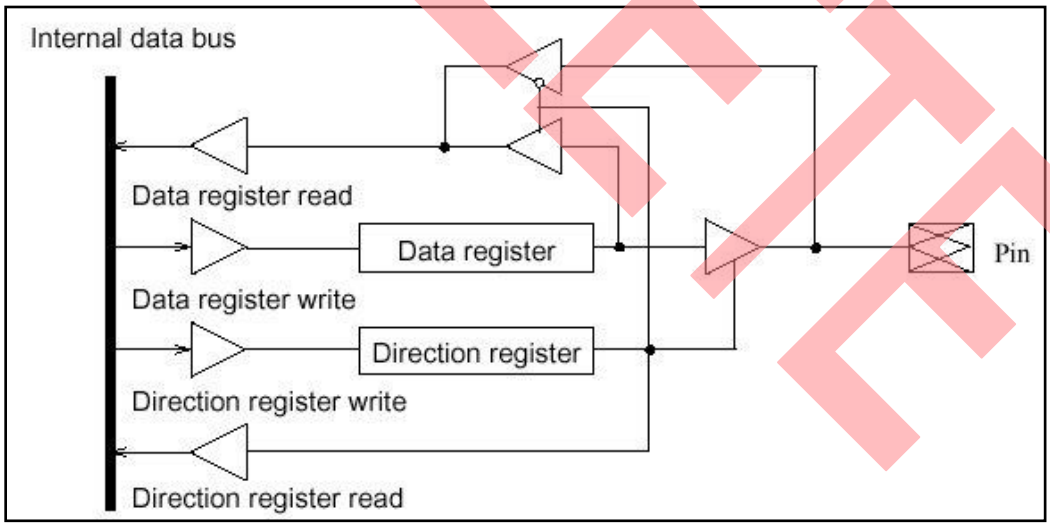
Figure 2. Transfer format in mode 3

In both modes 2 and 3 the data length is 9 bits plus one start and stop bit. The Uart’s only handles NRZ (non-return-to-zero) type data.

3 Hardware configuration:

The Figure 3 shows the port circuit of the ports used by the Uart. If the port is an input data is read from the input pin. As an output, the data is read from the data register.

Figure 3. I/O port block diagram



If the pin SOT and SIN of the MCUs are connected like in figure 4, the output pin of the slave has to set by default as input. Figure 4 shows an abstract example how the devices should be connected together.

OBSOLETE

The problem that occurs in this situation is that the output pins of the Cypress microcontroller are driven logically high (+5V). In cases the bus keeps quite, nothing will happen on the output line. In the case one slave tries to send the message to the master, the data will be corrupted or invalid for the master.

As mentioned before, the output line is driven high, when the port is defined as an output. A selected slave, who tries to write his data to the bus line, starts his asynchronous communication as shown in Figure 2. The communication is started with a start bit, pulling down the bus line low. This situation causes an undefined value, because one slave pulling down the bus line to low, where the remaining slaves pulling up the bus line to high. To avoid these circumstances, all the port pins of the SOT output are initialized as inputs.

When the master has selected a slave, the slave changes the SOT form input to Output State and starts communication with the master. After communication, the slave changes the SOT State back to input and releases the bus line.

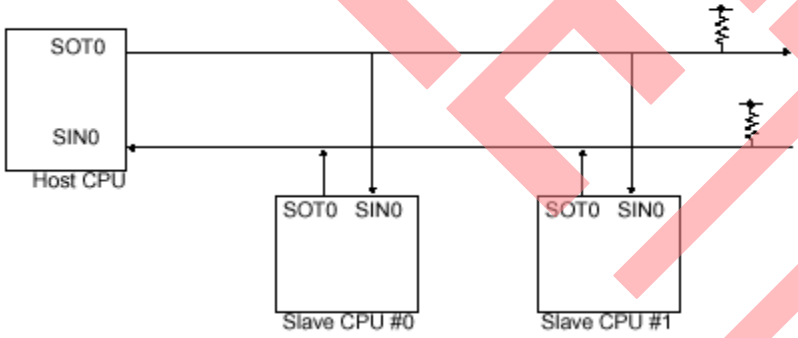
4 Function of the Master-Slave system:

When the system is powered-on, one MCU is initialized as the master. The rest of the MCUs will be initialized as slaves. The devices are connected with the SOT and SIN pins of the MCUs. Figure 4 shows the way the devices should be connected in general.

All slave pins, which are connected together to the bus, are initialized as inputs. This is only necessary when the slave is in the inactive state. The system distinguishes two kinds of states: the active state, this means the slave is communicating with the master and the passive state, this means the slave waits that the master wake him up and change the state to active.

A master MCU will be all the time in active state, as long as the controller is powered. The slaves change between the passive and the active state, which doesn't mean that the slave MCU is powered up and down. When the master selects a slave, the slave change from the passive state, which is the Default State of the slaves, into the active state. The selected slave MCU communicates with the host MCU using a protocol determined by the user. Unselected slave MCUs wait in standby until they are selected. When the communication is finished, the slave change back to passive state and remains so until it is called again by the master.

Figure 4. Connection of the MCUs



OBSOLETE

The master MCU starts a session by sending out an address on the bus to select the slave to whom the master wants to communicate with. All the slaves receive this message and compare the data with the given address. This address is only a software address and can be defined individually by the user. When the slave has identify its address, it first switches its output pin (SOT) from input to output direction to enable the transmit path, and than the slave becomes active. The next step is to enable the transmitter of the Uart.

The changing of the output direction to input direction is necessary to avoid data corruption on the bus. As shown in Figure 4 all slave outputs are connected together. In the case one slave is active and sending data on the bus, the active pin of the selected MCU shortcuts all other output pins. This can cause damage to the MCUs. Using additional external devices like open collector or open drain drivers with pull up resistors is another possibility.

The selected MCU slave is sending now an acknowledge informing the master that it is ready to receive or send data. If the slave does not respond in a specified period of time the master stops the communication with the slave and starts a new session with the same or another MCU.

In the case the handshaking was successful, the slave and the master start a communication without getting interrupted by any other slave MCU. As mentioned before, the slaves are set in mode 2 and the master in mode 3. If the master sends an address, the entire slaves connected to the 2 wire bus, receive this address. In case of an address match, the corresponding slave switches it mode from 2 to 3. During the communication between the master and the slave, - which operate in mode 2 - the entire connected slaves will not be interfered. The reason is, the slaves will read the incoming bytes as empty frames, as long as the bit D8 is zero, so no interrupts are generated for those slaves in mode 2 if just data frames are transferred. A MCU in mode 2 need this bit set to logic high to accept the incoming bytes as valid receive.

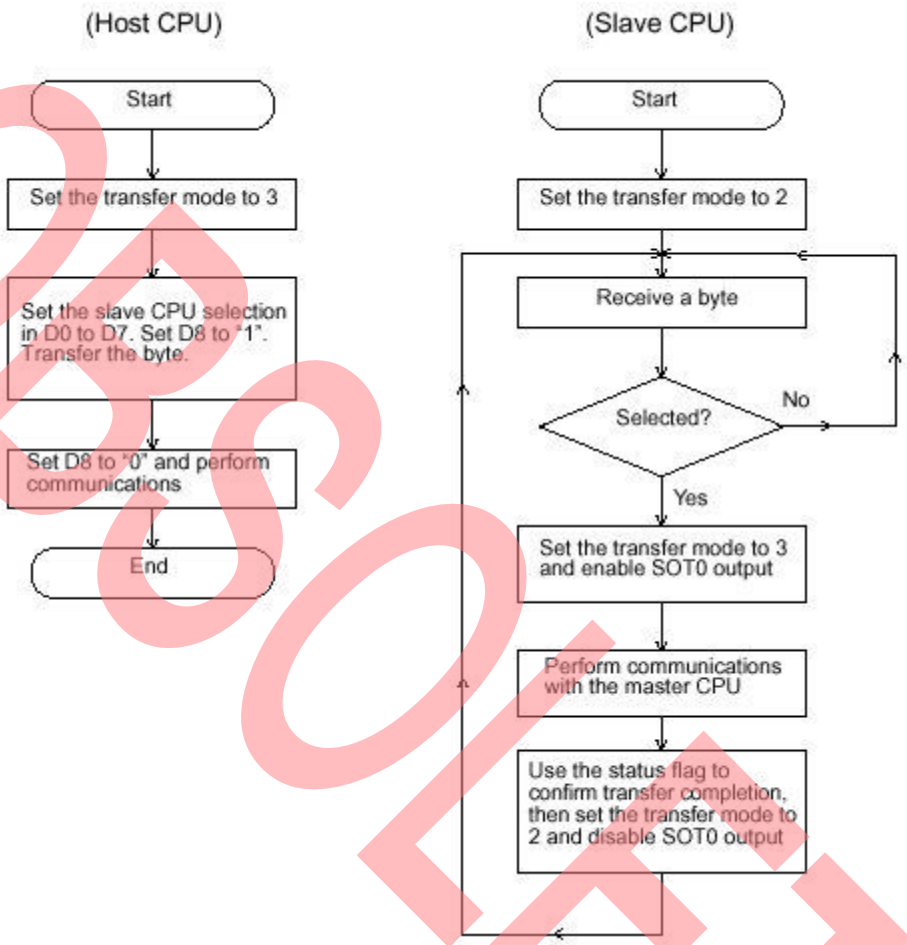
After the communication is finished, the slave switches his SOT output back into input. Than disables the transmitter to avoid irregular working of the Uart and will be ready for another session.

The slaves must start in the mode 2. In this mode the bit D8 must be set to ‘1’. If the bit is not set, the data on the bus will be ignored, the slave is unable to detect any data/address. After the slave has received a valid address, it switches to the same mode like the master (mode 3). Now communication can be done without interference of the other slaves. Therefore, the slave transfers data with bit D8 always 0. If the communication between the master and the slave is complete, the master finishes the communication. On the other hand the slaves switch back to mode two and change also the SOT pin from output to input.

OBSOLETE

Figure 5 shows the flowchart for a master and slave CPU.

Figure 5. Flowchart for master and slave



OBSOLETE

5 Conclusion:

This has been an overview how the Master-Slave system could be implemented with the Uart's macros of the Cypress microcontroller. It can be seen that within a few steps it is possible to implement a master and slave system.

The things, which are fixed, are only the hardware settings for the Uart's. All other mentioned facts can be defined by software as the user wishes.

A Appendix

Program code:

```
/*-----------------------------------------------------------------------
  MAIN.C
  - description
  - See README.TXT for project description and disclaimer.

-----------------------------------------------------------------------*/

/****************************** Includes ***********************************/
#include "mb90540.h"

/****************************** Defines ************************************/
//#define MASTER
//#define SLAVE1
#define SLAVE2

/****************************** Gloabals ***********************************/
unsigned char rbuf = 0;
unsigned char temp = 0;
unsigned char address = 0;
unsigned char tbuf = 0;
unsigned char slaves[] = {'1','2'};
unsigned char *p_string;
unsigned int cnt = 0;
char index,length;
char counter = 0;

/************************** Prototyps **********************************/
void InitUart_Master(void);
void InitUart_Slave(void);

/*========================== PROCEDURES ======================================*/
void InitUart_Master(void)
{
    /* initialize UART0 */
    UMC0 = 0x39;            /* serial mode control register (00111001) */
    USR0 = 0x0C;            /* status register (00011100) (master)     */
    URD0 = 0x4C;            /* rate and data register  (01001100)      */
    DDR4_D40 = 1;           /* set SOT0 working                                */
}

void InitUart_Slave(void)
{
    /* initialize UART0 */
    UMC0 = 0x28;     /* serial mode control register (00101000)                */
    USR0 = 0x08;     /* status register (00011000) only in receive mode  */
    URD0 = 0x4C;     /* rate and data register  (01001100)                     */
}
/*========================== END OF PROCEDURES ===============================*/



/************************** M A I N ************************************/
void main(void)
{
    InitIrqLevels();
    __set_il(7);               /* allow all levels */
```

OBSOLETE

```
    __EI();                         /* globaly enable interrupts */

    /* Master mode */
    #ifdef MASTER
       InitUart_Master();
       URD0_D8 = 0;
      while(1)
       {
   send(slaves[address]);                          /* send address */
   index = wait(slaves[address]);                        /* wait for acknowlegde */
       echo_ch();                                        /* echo acknowlegde */
       send('d');                                        /* send data */
       index = wait(slaves[address++]);           /* wait for acknowlegde */
       echo_ch();
       if(address > 1)
          address = 0;                        /* start communication with slave1 */
       tbuf = 'e';                            /* end of communication */
       while(tbuf != 0);
       }
    #endif


    /* Slave 1 */
    #ifdef SLAVE1
       InitUart_Slave();
       index = 0;
       while(1)
       {
       while(rbuf != '1');          /* waiting for the address */
       receive_adr();               /* detecting the address */
       UMC0 = 0x39;                 /* change mode to 3 */
   DDR4_D40 = 1;                    /* set SOT0 working */
   USR0 = 0x0C;                     /* enable transmit interrupt */
   while(rbuf != 'd');              /* waiting for data */
       receive_data();              /* start communication */
   while(rbuf != 'e');              /* wait for communication end */
       UMC0 = 0x28;                 /* change back to mode 2 */
   USR0 = 0x08;                     /* disable transmit interrupt*/
   DDR4_D40 = 0;                    /* change to input */
       }
    #endif

    /* Slave 2 */
    #ifdef SLAVE2
       InitUart_Slave();
       while(1)
       {
       while(rbuf != '2');          /* waiting for the address */
       receive_adr();               /* detecting the address */
   UMC0 = 0x39;                     /* change mode to 3 */
   DDR4_D40 = 1;                    /* set SOT0 working */
   USR0 = 0x0C;                     /* enable transmit interrupt */
   while(rbuf != 'd');              /* waiting for data */
       receive_data();              /* start communication */
   while(rbuf != 'e');              /* wait for communication end */
       UMC0 = 0x28;                 /* change back to mode 2 */
   USR0 = 0x08;                     /* disable transmit interrupt*/
   DDR4_D40 = 0;                    /* change to input */
       }
    #endif
}
```

OBSOLETE

```
/********************* end of main ***************************************/

/********************** interrupts ***************************************/
void __interrupt Uart0_Transmit(void)
{
    if(index != 0)                  /* transmit string */
    {
        if(counter < length)
        {
            temp = p_string[counter++];
            UODR0 = temp;
        }
        if(counter >= length)
        {
            counter = 0;
            index = 0;
        }
    }
    else
    {
        UODR0 = tbuf;               /* write data to the output buffer */
        tbuf = 0;                       /* clear tbuf */
    }
}
void __interrupt Uart0_Receive(void)
{
    if(USR0_ORFE)
    {
        temp = UIDR0;               /* clear input buffer */
        UMC0_RFC = 0;               /* clear error flags */
    }
    else
    {
        rbuf = UIDR0;               /* read data from the input buffer */
    }
    cnt++;
}
```

OBSOLETE

6 Document History

Document Title: AN205054 - UART Multimastermode, MB90F540/5 series with implementation examples

Document Number: 002-05054

Revision	ECN	Orig. of Change	Submission Date	Description of Change
**	-	NOFL	04/03/2000	Initial release
			05/20/2000	Updated version
			09/13/2000	Some slide changes
*A	5134187	NOFL	02/25/2016	Migrated Spansion Application Note MCU-AN-390062-E-V11to Cypress format This product is old, So this application note should be obsoleted.

OBSOLETE

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

OBSOLETE

PSoCisa registered trademark and PSoC Creator is a trademark of Cypress Semiconductor Corp. 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, 2000-2016. 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 under its copyright rights in the Software, a personal, non-exclusive, nontransferable license (without the right to sublicense) (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. Cypress also grants you a personal, non-exclusive, nontransferable, license (without the right to sublicense) 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 to the minimum extent that is necessary for you to exercise your rights under the copyright license granted in the previous sentence. Any other use, reproduction, modification, translation, or compilation of the Software is prohibited.

CYPRESS MAKES NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS DOCUMENT OR ANY SOFTWARE, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. 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 Company shall and hereby does release Cypress from any claim, damage, or other liability arising from or related to all Unintended Uses of Cypress products. Company 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.