AN1564

Configuring and Using the Event Detection Module of MCP795WXX RTCCs

Author: *Alexandru Valeanu*
Microchip Technology Inc.

INTRODUCTION

This application note is designed to take the design engineer through the steps to configure and set up the event detect module of Microchip SPI MCP795WXX Real-Time Clock/Calendar (RTCC) family. Topics covered include:

- Configuring the event detection module
- Using the event detection module
- Overview of the provided drivers/libraries

The information presented in this document is designed to be an example of possible configurations. The code supplied can be modified to change device functionality.

This application note should be read in conjunction with application note, AN1365 – *"Recommended Usage of Microchip Serial RTCC Devices"* (DS00001365) and the device data sheet. The latest documentation can be found on Microchip website, at the following address:

http://www.microchip.com/rtcc

WHAT IS THE EVENT DETECT MODULE?

The event detect module on the MCP795WXX RTCCs enables systems to respond in a fast and automatic manner to external events (through interrupts).

Low-Speed Event Detect

The low-speed event detect (EVLS input) has been designed to interface directly with a mechanical system, for example, a switch (supporting a built-in debounce feature).

This can be used to generate an interrupt to the MCU when a push button event is detected and debounced (for example, detecting a tamper of an enclosure). See Figure 3 for an example.

High-Speed Event Detect

The high-speed event part (EVHS input) can be used to generate an interrupt after detecting a programmable number of digital transitions.

This could be used to interrupt an MCU, when a digital preamble is received from a communication channel. Figure 2 shows an example of the high-speed event detect triggering after 16 transitions.

SCHEMATIC DIAGRAM

The schematic illustrated in Figure 1 shows the minimum components required to operate the RTCC, when using the event detection module.

The schematic also shows the required components for battery backup operation using a lithium coin cell (for other options refer to AN1365). If the V_{BAT} input and battery backup feature is not required, this pin should be tied to GND.

In the provided code, the following signals are defined:

- `#define NCS_SPI_RTCC PORTCbits.RC2 //` $\overline{CS}$ `for the SPI RTCC.`
- `#define IRQ PORTBbits.RB1`

  `// interrupt request from the EVDET module used through polling`

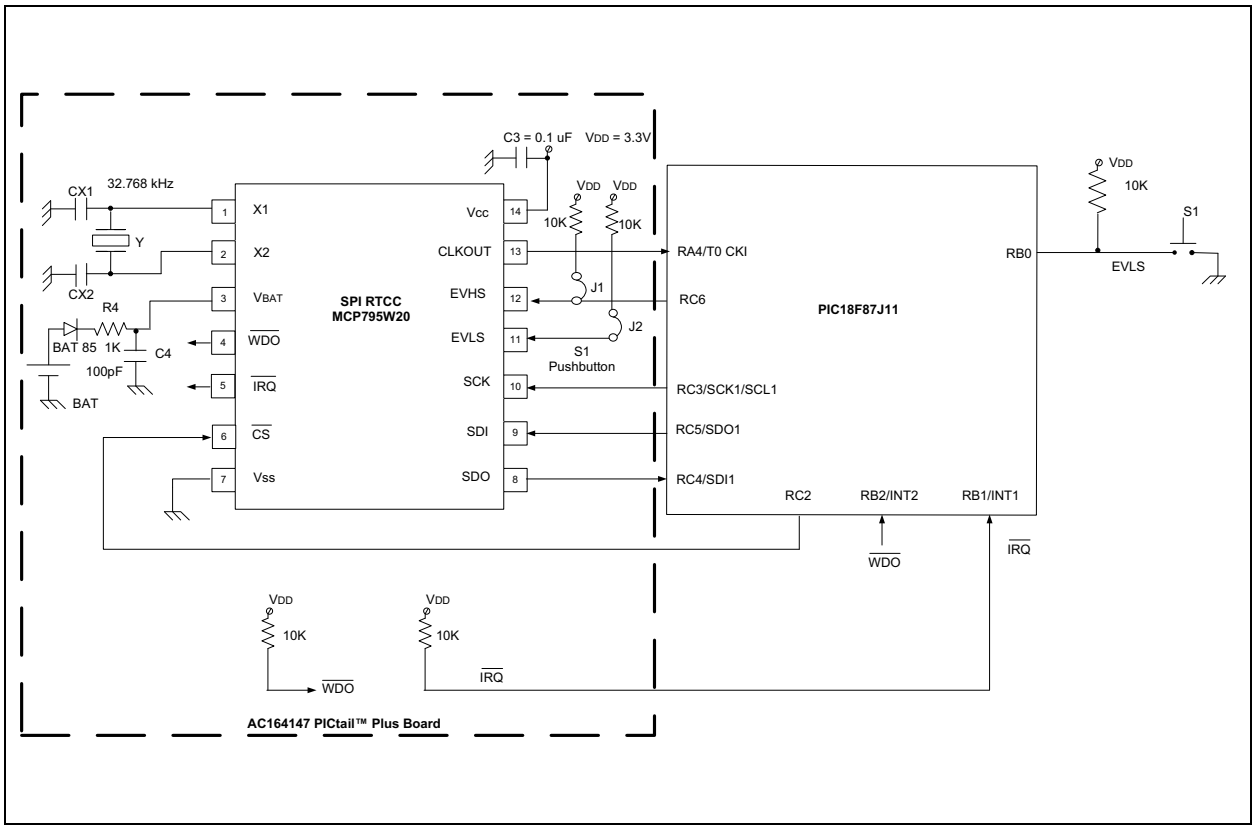
A GPIO (i.e., RC6) on the PIC® microcontroller is used to simulate an external event on the EVHS pin of the RTCC.

Accordingly, a related `#define directive` was created in the firmware:

```
#define EVHS PORTCbits.RC6
```

The low-speed input (EVLS) is tied to the S1 push button. There is no need to define EVLS, since it is controlled by the S1 push button and not by the GPIO.

FIGURE 1: SCHEMATIC



The best way to get started with the MCP795WXX is to use the SPI RTCC PICtail™ module (AC164147).

FIRMWARE OVERVIEW

The host MCU will communicate with the RTCC using the SPI protocol, based on the MSSP1 serial module of the PIC18 MCU.

The code presented with this application note is designed to compile with the XC8 (V 1.34) compiler and using MPLAB® X IDE (V3.55) for the following hardware:

- Explorer 18 Evaluation Board (DM183032)
- PIC18F87J11 PIM Module (MA180020)
- RTCC SPI PICtail module (AC164147)

The code is presented in C and is portable with minimal effort to other PIC MCU devices. This code is designed to be a starting point for application development and is based on the drivers/libraries presented in this text and the related code.

APPLICATION DESCRIPTION

The code presents the basics of the event detect configuration and usage. After initializing the RTCC and the module, the user can choose between two test functions:

- High-speed test: `evhs_test()`
- Low-speed test: `evls_test()`

The high-speed test will output clock pulses on the EVHS input, using the RC6 GPIO, until the high-speed event detect module generates an interrupt, by asserting IRQ low. The code will then turn on the appropriate LEDs to show the number of clock pulses in binary. The low-speed test will wait the action of the S1 push button, connected to the EVLS input. When this is detected through polling (IRQ=0), the firmware will turn on the LEDs.

FIGURE 2: EVHS INPUT

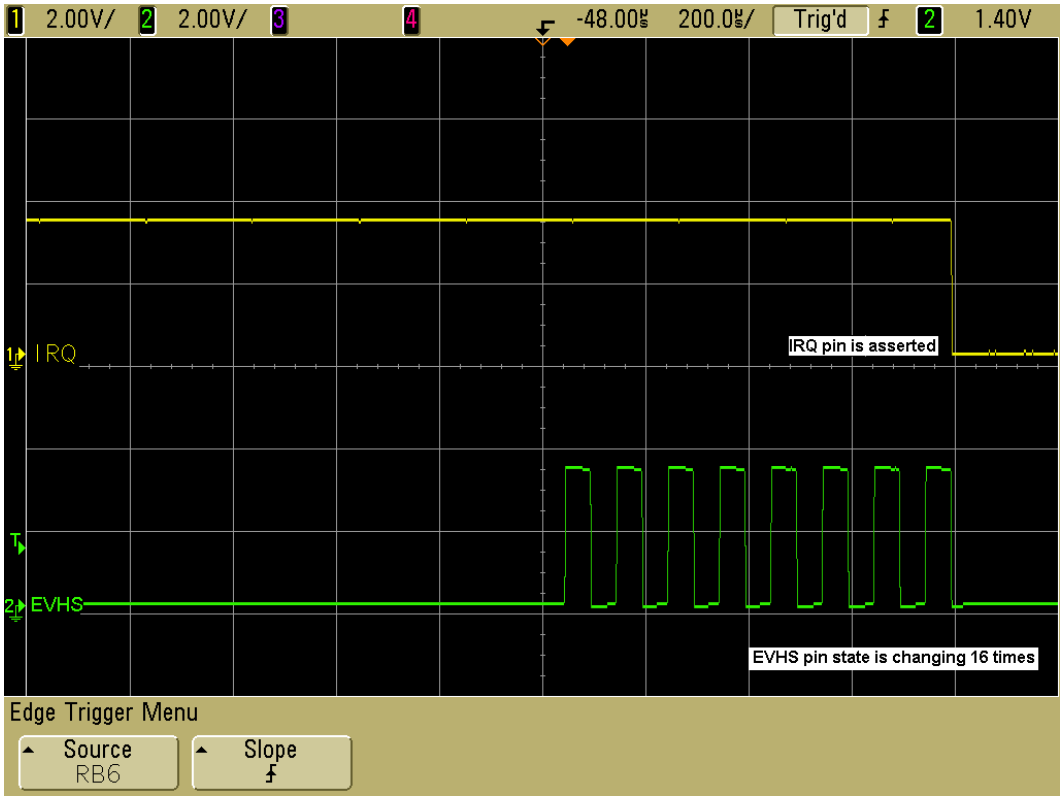
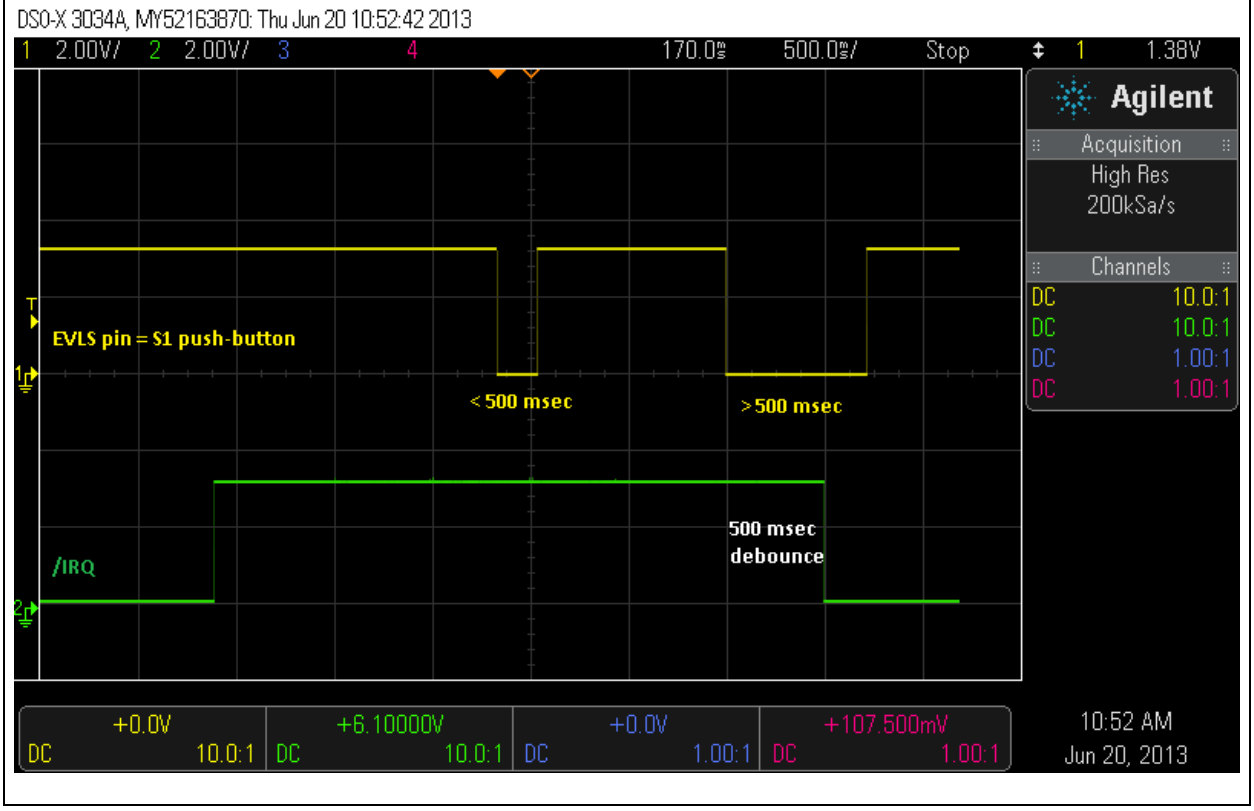


FIGURE 3: EVLS INPUT



CONFIGURATION OF THE EVENT DETECTION MODULE

The operation of the module is tied around the related Event Detection register (EVDTCON) (address 0Bh).

TABLE 1: EVENT DETECTION REGISTER

R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
EVHIF	EVLIF	EVHEN	EVLEN	EVWDTEN	EVLPS	EVHCS1	EVHCS0
bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0

Legend:		
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'

bit 7 **EVHIF:** High-Speed Event Detect Interrupt Flag bit
1 = High-speed event detection occurred (must be cleared in software)
0 = High-speed event detection did not occur

bit 6 **EVLIF:** Low-Speed Event Detect Interrupt Flag bit
1 = Low-speed event detection occurred (must be cleared in software)
0 = Low-speed event detection did not occur

bit 5 **EVHEN:** High-Speed Event Detect Module Enable bit
If EVWDTEN = 0:
1 = High-Speed Event Detect enabled
0 = High-Speed Event Detect disabled
If EVWDTEN = 1:
Unused.

bit 4 **EVLEN:** Low-Speed Event Detect Module Enable bit
1 = Low-Speed Event Detect enabled
0 = Low-Speed Event Detect disabled

bit 3 **EVWDTEN:** EVHS Input WDT Clear Enable bit
1 = Enable Watchdog Timer clear on EVHS input transition. Disables high-speed event detect module.
0 = Disable EVHS input clearing Watchdog Timer.

bit 2 **EVLPS:** Low-Speed Event Detect Debounce Period Select bit
1 = 16,384 oscillator cycles (500 ms nominal)
0 = 1,024 oscillator cycles (31.25 ms nominal)

bit 1-0 **EVHCS<1:0>:** High-Speed Event Detect Transition Count Select bits
Selects how many transitions must occur on the EVHS input before an interrupt is triggered
00 = 1 transition
01 = 4 transitions
10 = 16 transitions
11 = 32 transitions

To program the register, several masks were used, as in Example 1:

EXAMPLE 1:

```
#define EVHIF          0x80     // EVDT High Speed Interrupt Flag
#define EVLIF          0x40     // EVDT Low Speed Interrupt Flag
#define EVDT_BOTH_OFF  0x00     // Low Speed and High Speed Modules are OFF
#define EVDT_LS_ON     0x10     // Low Speed    ON
#define EVDT_HS_ON     0x20     // High Speed   ON
#define EVDT_BOTH_ON   0x30     // Both modules ON
#define EVWDTEN        0x08     // High Speed Event Detection will clear the WDT
#define EVLPS          0x04     // Low Speed Event period: 500 ms (0 value -> 31.25 ms)
#define EVDT_EV_NR_1   0x00     // 1st Event on EVHS will assert low /IRQ
#define EVDT_EV_NR_4   0x01     // 4th Event on EVHS will assert low /IRQ
#define EVDT_EV_NR_16  0x02     // 16th Event on EVHS will assert low /IRQ
#define EVDT_EV_NR_32  0x03     // 32nd Event on EVHS will assert low /IRQ
#define EVDTCON        0x0B     // address of the event detect register
#define IRQ   PORTBbits.RB1     // interrupt from the event detect module (polling)
#define EVHS PORTCbits.RC6      // GPIO creating pulses on the EVHS input
```

The high-speed test will initialize the related register as in Example 2:

EXAMPLE 2:

```
spi_rtcc_wr(EVDTCON, EVDT_HS_ON + EVDT_EV_NR_32);
// enable the high-speed module, set event number at 32
```

The low-speed test will initialize the related register as in Example 3:

EXAMPLE 3:

```
spi_rtcc_wr(EVDTCON, EVDT_LS_ON + EVLPS);
// enable the low-speed module, debounce value = 500 msec
```

Any other combination is allowed using the above `#define directives`.

CLEARING THE EVENT DETECTION INTERRUPT FLAG

The interrupt flags must be cleared by the user to obtain additional interrupts. To write to the flags, the entire Event Detection register must be written. The most robust method is to read the current value from the Event Detection register, clear the necessary bits, and write back the result. The following code shows an example of how to perform this:

EXAMPLE 4:

```
unsigned char aux ;
aux = spi_rtcc_rd(EVDTCON)&(~EVHIF) ;
spi_rtcc_wr(EVDTCON,aux);
```

Alternatively, disabling the high-speed and low-speed modules by clearing the Event Detection register, as well as rewriting the initial configuration value to the register, will also clear the interrupt flags.

CONCLUSION

Following the steps in this application note, along with the included MPLAB® X IDE project, will help set up the basic configuration of the event detection module available in the Microchip MCP795WXX SPI RTCC devices. By using available off-the-shelf development tools, any hardware issues will be mitigated, allowing the engineer to concentrate on the firmware development.

This allows the engineer to easily work with low-speed components, such as mechanical switches, as well as high-speed interfaces that require the detection of a number of digital transitions.

The code is presented in C (XC8-V1.34 compiler on MPLAB® X IDE-V3.55) and can easily be ported to other PIC® device platforms.

APPENDIX A: TEST FUNCTIONS FOR THE EVENT DETECT MODULE

High-speed test

```
evhs_test(){
spi_rtcc_wr(EVDTCON, EVDT_HS_ON + EVDT_EV_NR_32);
// enable the high speed module, set event number at 32
while(1){
EVHS = ~EVHS ;         // toggle the high speed input
dly1_5ms()   ; cnt++ ; // delay = 1.5 msec, increment counter
if(!IRQ)break;       } // if the interrupt will assert low, break
                       // the infinite while(1) loop
LATD = cnt   ;       } // display the EVHS counter, exit
```

Low-speed test

```
evls_test() {
spi_rtcc_wr(EVDTCON, EVDT_LS_ON + EVLPS);
// enable the low speed module, debounce value = 500 msec.
While(IRQ)  ;           // wait IRQ to assert low,when S1 push button is pressed
LATD = 0xff ;        }  // print message when IRQ=0 was detected
```

APPENDIX B: DRIVERS LIBRARY

Delay Drivers (`delay_drivers.h`)

Not accessed in the present project. Used by the LCD drivers or as general purpose timing functions.

Represent a good starting point for further development on PIC18-based projects (they use the timers of the MCU) (see AN1355, AN1364, AN1413 for the usage of these drivers).

LCD Drivers (`lcd_drivers.h`)

Not accessed in the present project. Represent a good starting point on Explorer18-based projects (see AN1355, AN1364, AN1413 for the usage of these drivers).

SPI Drivers (`spi_drivers.h`)

Represent the low-level communication between the MSSP1 module of the PIC18 and the SPI RTCC.

The related functions will be detailed in the next paragraph, as called functions.

SPI RTCC Drivers (`spi_rtcc_drivers.h`)

Represent the medium-level communication between the MSSP1 module of the PIC18 and the SPI RTCC.

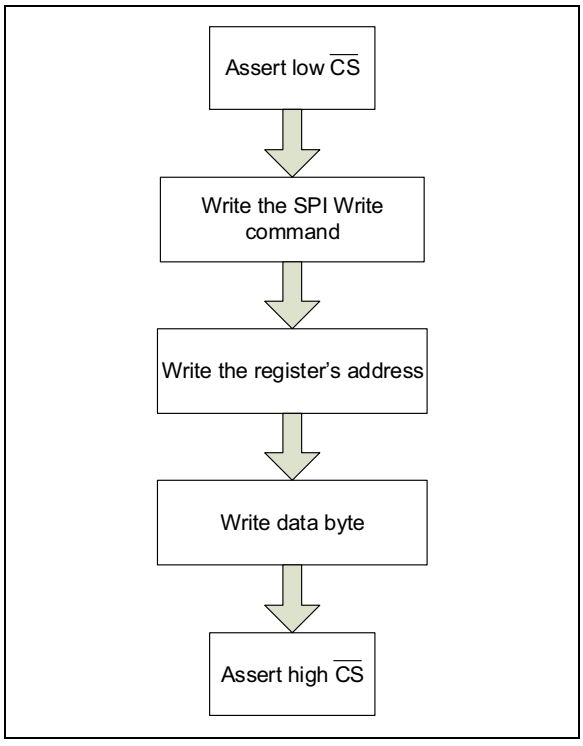
The related functions call the SPI drivers, as described below. Moreover, the library defines all necessary constants, as register addresses and masks.

EXAMPLE 5: WRITING A BYTE TO THE SPI RTCC

```
void spi_rtcc_wr (unsigned char rtcc_reg, unsigned char time_var){
                                          // SPI write to the SPI RTCC
spi_rtcc_start()                        ;   // start SPI comm with the SPI RTCC
spi_wrbyte(SPI_RTCC_WRITE)              ;   // send the SPI WRITE command
spi_wrbyte(rtcc_reg)                    ;   // send the register's address
spi_wrbyte(time_var)                    ;   // send the SPI data byte
spi_rtcc_stop()                         ; } // stop SPI comm
```

The firmware for writes to the RTCC is shown in Example 6.

EXAMPLE 6:

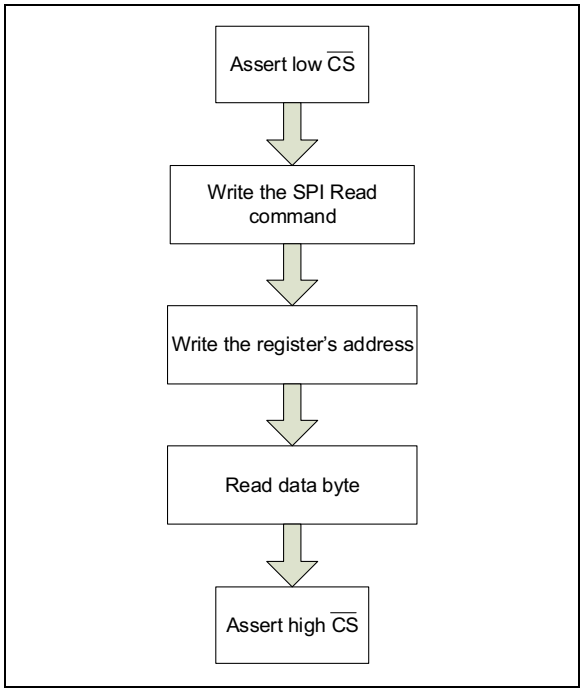


EXAMPLE 7: READING A BYTE FROM THE SPI RTCC

```
unsigned char spi_rtcc_rd(unsigned char rtcc_reg){
                                          // SPI read from the SPI RTCC
spi_rtcc_start()                        ;  // start the SPI comm with the SPI RTCC
spi_wrbyte(SPI_RTCC_READ)               ;  // send the SPI READ command
spi_wrbyte(rtcc_reg)                    ;  // send the register's address
rtcc_buf = spi_rdbyte()                 ;  // read the result and store it
spi_rtcc_stop()                         ;  // stop the SPI comm with the SPI RTCC
return rtcc_buf                         ;} // return the read result
```

The firmware for reads from the RTCC is shown in Example 8.

EXAMPLE 8:



APPENDIX C: REVISION HISTORY

Revision A (09/2013)

Initial release of this document.

Revision B (11/2017)

Updated register and bit names.

NOTES:

Note the following details of the code protection feature on Microchip devices:

- Microchip products meet the specification contained in their particular Microchip Data Sheet.
- Microchip believes that its family of products is one of the most secure families of its kind on the market today, when used in the intended manner and under normal conditions.
- There are dishonest and possibly illegal methods used to breach the code protection feature. All of these methods, to our knowledge, require using the Microchip products in a manner outside the operating specifications contained in Microchip's Data Sheets. Most likely, the person doing so is engaged in theft of intellectual property.
- Microchip is willing to work with the customer who is concerned about the integrity of their code.
- Neither Microchip nor any other semiconductor manufacturer can guarantee the security of their code. Code protection does not mean that we are guaranteeing the product as "unbreakable."

Code protection is constantly evolving. We at Microchip are committed to continuously improving the code protection features of our products. Attempts to break Microchip's code protection feature may be a violation of the Digital Millennium Copyright Act. If such acts allow unauthorized access to your software or other copyrighted work, you may have a right to sue for relief under that Act.

Microchip received ISO/TS-16949:2009 certification for its worldwide headquarters, design and wafer fabrication facilities in Chandler and Tempe, Arizona; Gresham, Oregon and design centers in California and India. The Company's quality system processes and procedures are for its PIC® MCUs and dsPIC® DSCs, KEELOQ® code hopping devices, Serial EEPROMs, microperipherals, nonvolatile memory and analog products. In addition, Microchip's quality system for the design and manufacture of development systems is ISO 9001:2000 certified.

QUALITY MANAGEMENT SYSTEM
CERTIFIED BY DNV
═ ISO/TS 16949 ═

Trademarks

ISBN:

Worldwide Sales and Service

AMERICAS

Corporate Office
2355 West Chandler Blvd.
Chandler, AZ 85224-6199
Tel: 480-792-7200
Fax: 480-792-7277
Technical Support:
http://www.microchip.com/support
Web Address:
www.microchip.com

Atlanta
Duluth, GA
Tel: 678-957-9614
Fax: 678-957-1455

Austin, TX
Tel: 512-257-3370

Boston
Westborough, MA
Tel: 774-760-0087
Fax: 774-760-0088

Chicago
Itasca, IL
Tel: 630-285-0071
Fax: 630-285-0075

Dallas
Addison, TX
Tel: 972-818-7423
Fax: 972-818-2924

Detroit
Novi, MI
Tel: 248-848-4000

Houston, TX
Tel: 281-894-5983

Indianapolis
Noblesville, IN
Tel: 317-773-8323
Fax: 317-773-5453
Tel: 317-536-2380

Los Angeles
Mission Viejo, CA
Tel: 949-462-9523
Fax: 949-462-9608
Tel: 951-273-7800

Raleigh, NC
Tel: 919-844-7510

New York, NY
Tel: 631-435-6000

San Jose, CA
Tel: 408-735-9110
Tel: 408-436-4270

Canada - Toronto
Tel: 905-695-1980
Fax: 905-695-2078

ASIA/PACIFIC

Australia - Sydney
Tel: 61-2-9868-6733

China - Beijing
Tel: 86-10-8569-7000

China - Chengdu
Tel: 86-28-8665-5511

China - Chongqing
Tel: 86-23-8980-9588

China - Dongguan
Tel: 86-769-8702-9880

China - Guangzhou
Tel: 86-20-8755-8029

China - Hangzhou
Tel: 86-571-8792-8115

China - Hong Kong SAR
Tel: 852-2943-5100

China - Nanjing
Tel: 86-25-8473-2460

China - Qingdao
Tel: 86-532-8502-7355

China - Shanghai
Tel: 86-21-3326-8000

China - Shenyang
Tel: 86-24-2334-2829

China - Shenzhen
Tel: 86-755-8864-2200

China - Suzhou
Tel: 86-186-6233-1526

China - Wuhan
Tel: 86-27-5980-5300

China - Xian
Tel: 86-29-8833-7252

China - Xiamen
Tel: 86-592-2388138

China - Zhuhai
Tel: 86-756-3210040

ASIA/PACIFIC

India - Bangalore
Tel: 91-80-3090-4444

India - New Delhi
Tel: 91-11-4160-8631

India - Pune
Tel: 91-20-4121-0141

Japan - Osaka
Tel: 81-6-6152-7160

Japan - Tokyo
Tel: 81-3-6880- 3770

Korea - Daegu
Tel: 82-53-744-4301

Korea - Seoul
Tel: 82-2-554-7200

Malaysia - Kuala Lumpur
Tel: 60-3-7651-7906

Malaysia - Penang
Tel: 60-4-227-8870

Philippines - Manila
Tel: 63-2-634-9065

Singapore
Tel: 65-6334-8870

Taiwan - Hsin Chu
Tel: 886-3-577-8366

Taiwan - Kaohsiung
Tel: 886-7-213-7830

Taiwan - Taipei
Tel: 886-2-2508-8600

Thailand - Bangkok
Tel: 66-2-694-1351

Vietnam - Ho Chi Minh
Tel: 84-28-5448-2100

EUROPE

Austria - Wels
Tel: 43-7242-2244-39
Fax: 43-7242-2244-393

Denmark - Copenhagen
Tel: 45-4450-2828
Fax: 45-4485-2829

Finland - Espoo
Tel: 358-9-4520-820

France - Paris
Tel: 33-1-69-53-63-20
Fax: 33-1-69-30-90-79

Germany - Garching
Tel: 49-8931-9700

Germany - Haan
Tel: 49-2129-3766400

Germany - Heilbronn
Tel: 49-7131-67-3636

Germany - Karlsruhe
Tel: 49-721-625370

Germany - Munich
Tel: 49-89-627-144-0
Fax: 49-89-627-144-44

Germany - Rosenheim
Tel: 49-8031-354-560

Israel - Ra'anana
Tel: 972-9-744-7705

Italy - Milan
Tel: 39-0331-742611
Fax: 39-0331-466781

Italy - Padova
Tel: 39-049-7625286

Netherlands - Drunen
Tel: 31-416-690399
Fax: 31-416-690340

Norway - Trondheim
Tel: 47-7289-7561

Poland - Warsaw
Tel: 48-22-3325737

Romania - Bucharest
Tel: 40-21-407-87-50

Spain - Madrid
Tel: 34-91-708-08-90
Fax: 34-91-708-08-91

Sweden - Gothenberg
Tel: 46-31-704-60-40

Sweden - Stockholm
Tel: 46-8-5090-4654

UK - Wokingham
Tel: 44-118-921-5800
Fax: 44-118-921-5820