AN2534

PAC193X Integration Notes for Microsoft® Windows® 10 and Windows 11 Driver Support

Author: *Razvan Ungureanu*
Microchip Technology Inc.

INTRODUCTION

This document describes the basic steps for integrating the PAC193X DC Power Monitor device in a Microsoft® Windows® 10 or Windows 11 host system to enable support of the PAC193X Windows Device Driver.

As the PAC193X device can be used in multiple ways and in different system configurations, there are some specific hardware and BIOS configuration details that need to be addressed before loading the Windows device driver.

The details about the PAC193X Windows device driver loading, feature set and software interfaces are included in the *"PAC193X Microsoft Windows® 10 and Windows 11 Device Driver User's Guide"* that complements the information presented by this document.

TABLE 1: GLOSSARY OF TERMS AND ACRONYMS

Acronym	Term
E3	Energy Estimation Engine
EMI	Energy Metering Interface
OS	Operating System
CPU	Central Processing Unit
SoC	System on Chip
EC	Embedded Controller
SPB	Simple Peripheral Bus
SpbCx	SPB Framework Extension
ACPI	Advanced Configuration and Power Interface
ASL	ACPI Source Language
BIOS	Basic Input/Output System
OEM	Original Equipment Manufacturer
UUID	Universal Unique Identifier
DSM	Device-Specific Method
I^2C	Inter-Integrated Circuit
SMBus	System Management Bus

This document is provided as part of the Microchip PAC193X Windows 10 or Windows 11 device driver release and is subject to change with new releases of the driver.

The document revision is Revision C.

HARDWARE INTEGRATION

The hardware integration must first address all the electrical details specified in the device data sheet. For example, the device V_{DD} I/O must match the I^2C bus voltage. The following hardware notes address only the hardware details that need specific configuration to make them compatible with the Windows 10 or Windows 11 device driver:

- I^2C bus controller Windows support
- PAC193X V_{DD} and SLOW/$\overline{\text{ALERT}}$ pin connections
- Channel shunt resistor values
- Channel polarity

I^2C Bus Controller Windows Support

The PAC193X device I^2C/SMBus interface is, by default, configured in the I^2C mode. Keep in mind that the SMBus protocol is not currently supported by Windows; therefore, the system integrators must connect the PAC devices to the I^2C bus controllers for which a Windows device driver is available and the driver is compatible with the Windows Simple Peripheral Bus Framework Extension (SpbCx). This is a mandatory condition because the PAC193X Windows Device Driver uses the SpbCx standard software interface to communicate with I^2C clients.

The I^2C bus controller may be located in the system CPU (SoC architecture), in the system Embedded Controller (EC), or may be another device connected to one of the system internal buses.

PAC193X V_{DD} and SLOW/$\overline{\text{ALERT}}$ Pin Connections

The usual PAC193X system integration scenario assumes that the system power rails are also monitored during the standby or shutdown system states. Therefore, the PAC device V_{DD} pin should be connected to a system power rail that is not shut down by the system standby or power-down sequence. This way, the PAC device continues to be capable of accumulating the data about the system energy consumption during the standby/shutdown state, if desired. The PAC193X Windows device driver is able to retrieve and report these data to the Windows Energy Estimation Engine (E3) or user applications.

In order to facilitate a longer period in standby/shutdown state, reducing the chances for accumulator register overflow, the device sample rate should be reduced to a minimum. This also reduces energy use by the PAC193X device. In normal system operation, this is done by the Windows device driver that reconfigures the Sample_Rate bits from the PAC193X device CTRL register just before the operating system switches the system in the standby/shutdown state. In case of an operating system crash, eventually the system is forcibly shut down by the user or by some automated safety mechanism. The PAC device driver is not functional in case of a Windows system crash, but the PAC device can still be reconfigured in Slow Sampling mode if the SLOW pin is driven high. The SLOW pin can be driven high by a hardware block that detects the system power-off. For example, the SLOW pin can be driven by a transistor that is gated by the CPU or by the main memory power rails. Alternatively, the SLOW pin may be driven by the system Embedded Controller. The EC is usually an always on device, which controls various hardware low-level housekeeping and also controls the system power rails and the power-on sequence.

When the system is turned back on, the PAC193X Windows Device Driver is able to detect that the SLOW pin was asserted and collects the data accumulated during this time period. This is possible because the driver makes sure that the SLOW/$\overline{\text{ALERT}}$ pin is always configured as SLOW and the SLOW pin transitions trigger the Accumulator registers' limited refresh (these are PAC device default settings).

Channel Sense Resistor (R_{SENSE})

Up to the Version 1.3, the PAC193X Windows device driver supports only sense resistor values that are integer values expressed in milliohm units. Therefore, a value of 1.5 milliohms is not supported. Starting with Version 1.3, the driver supports sense resistor values expressed in microohm units. As such, a sense resistor of 1500 microohms is supported.

The recommended shunt resistor tolerance is 1%.

Channel Polarity

After the Power-on Reset (POR), the PAC device default channel configuration is set for unipolar voltage measurements. This configuration can be changed by the Windows device driver initialization phase, as indicated by the DSM Rev. 1 parameters. In addition, the user applications may change the channel polarity settings by calling a dedicated driver interface. However, because the Windows E3 service supports only positive energy values, it is recommended to connect the PAC device SENSE pins to the power rails, such that by default, the V_{SENSE} voltage is read as a positive value. There are two reasons for this:

- Windows E3 service is not able, by design, to control the channel polarity
- The standard Energy Metering Interface (EMI) through which the driver provides the energy figures to E3 service allows only positive integer values

ACPI BIOS INTEGRATION

Over and above any integration guidance provided in this document, OEMs and system integrators are responsible for creating a valid ACPI BIOS that accurately reflects their hardware platform. This includes the details of how the PAC193X is configured and connected in the system.

The guidance provided in this section is limited to describing the details that are necessary to integrate the PAC193X device that is part of a system with the PAC193X device driver and the Windows 10 or Windows 11 operating systems.

Appendix A: "Device Definition Code" of this document contains an example ACPI device definition, written in ACPI Source Language (ASL). The example illustrates how a PAC193X is to be integrated into the ACPI BIOS. Please refer to the Appendix A example when reading this section.

Appendix B: "Test the PAC193X ACPI Integration" of this document provides an example of easily available tools and the process to follow in order to quickly test the PAC193X integration into the system, without replacing the BIOS firmware in the system Flash.

PAC193X Device Description

There are several per-device and per-channel configuration options that are defined when a PAC193X is built into a system. These options include:

- The I^2C address for each PAC193X device in the system;
- The name of the power rail to which each PAC193X channel is connected;
- The R_{SENSE} value configured for each channel;
- For which of the PAC193X channels the Windows driver must not create the Energy Metering Interface (EMI) (channels defined as private channels).

These options, which are chosen on a per-platform basis, need to be communicated to the PAC193X device driver. This allows the driver to properly inform Windows about system power usage via the standard Windows EMI.

The method for communicating these configuration options to the PAC193X driver is via the PAC193X device definition in the platform's ACPI BIOS. The OEM is responsible for providing a proper ACPI device definition for each PAC193X device configured in the system. Creating a proper definition for the PAC193X device includes providing the device-specific configuration items outlined in this document.

The definition for the PAC193X device may be copied directly from the **Appendix A: "Device Definition Code"** example, multiplied and customized for use on a particular system.

As part of the PAC193X device definition, a DSM (Device-Specific Method) control method that corresponds to the Universal Unique Identifier (UUID), `033771E0-1705-47B4-9535-D1BBE14D9A09`, must be specified as shown in the example. This UUID is reserved to Microchip for the PAC193X and must not be changed. The PAC193X device driver relies on the DSM control method supporting this UUID.

There are six functions implemented by the current DSM Rev. 1. Function #1 is implemented by DSM Rev. 1 the same way as it was implemented by DSM Rev. 0 in order to comply with the ACPI DSM backward compatibility specification.

The PAC193X driver support for DSM Rev. 1 has been implemented starting with Driver Version 1.3.

DSM Function #1 (again, refer to the example in **Appendix A: "Device Definition Code"**) returns a package that describes how each channel on the PAC193X device described by this ACPI device definition is configured. For each channel, the name of the power rail to which the PAC193X channel is connected must be supplied and the R_{SENSE} value configured for that channel.

Note that the supplied power rail names cannot be arbitrary. They must be formed using predefined values that are valid metered hardware names according to the Windows-defined EMI power rail taxonomy. Please refer to Microsoft supplied documentation for a more thorough description of valid power rail names.

The R_{SENSE} value is specified in milliohms and only integer values are allowed.

To indicate that a particular channel is not used on a given PAC193X device (that is, no power calculations should be performed for it), set the channel's R_{SENSE} value to zero.

It is possible to use one or more PAC193X channels to monitor power rails without exposing those measurements to the Windows Energy Metering Interface. Such channels are referred to as 'Private' channels. The PAC193X device driver will calculate the power usage for Private channels too, but will not create the standard EMI interface, and so the results will not be accessible to Windows. Still, the data calculated for a Private channel can be queried by user applications using the Microchip-defined PAC193X device-specific IOCTLs. Before the PAC193X Driver Version 1.3, the only way to configure a channel as 'Private' (it has no EMI) was to set the DSM Function #1 to return an empty string (that is, "") for the channel name. Starting with Driver Version 1.3, all the channels may have names because the DSM Function #3 can be used to indicate which channels are EMI or Private. Note that the R_{SENSE} value for Private channels must be valid and must not be zero.

OEMs and systems integrators must take great care in properly configuring the values for power rails and R_{SENSE}, because both Windows and the PAC193X device driver rely on these values being correct. There is no way for the PAC193X driver to know if a given power rail name is valid, or more importantly, to know if the indicated PAC193X channel is actually connected to the specified power rail.

DSM Function #2 returns a package containing the sense resistor values expressed in microohm units. To improve the data precision, a driver version that supports this function may use the returned microohm values instead of the milliohm values returned by Function #1.

DSM Function #3 returns a bit mask indicating which channels may have EMI interfaces or may be Private, even if they have valid names. Note that the PAC193X driver may not create EMI interfaces for the channels with no name.

DSM Function #4 returns the channel polarity bit mask (V_{BUS} and V_{SENSE} polarity) in the same format as the device NEG_PWR register.

DSM Function #5 returns the sample frequency values that the driver may use to configure the device when the system enters the ACTIVE mode, respectively when the system exits the ACTIVE mode.

DSM Function #6 returns the value of the maximum time interval that the driver may allow between consecutive device REFRESH commands, in order to avoid accumulator saturation.

TABLE 2: PAC193X ACPI INTEGRATION PARAMETERS SUMMARY

Parameter	Type	Value	Comments
_UID	Integer	≥ 1	Each PAC device in the system must have a unique _UID.
I^2C 7-Bit Client Address	Integer	The value selected by the ADDRSEL pin	The Address Select Resistor table from the device data sheet provides the supported address values and their associated resistor values. There are 16 possibilities.
I^2C Connection Speed	Integer	Typ: 400000 Max: 1000000	The selected speed must match the I^2C bus controller supported speed.
I^2C Controller Name	string	The ACPI name of the I^2C bus controller	The I^2C bus controller driver must have the Windows® SpbCx compliant driver.
Channel Name	string	"" or non-empty string	• "": Empty string denotes a Private channel. • Non-empty string must comply with Microsoft® power rail naming taxonomy.
R_{SENSE} Value	Integer	0 or non-zero	• Expressed in milliohms when returned by Function #1. • Expressed in microohms when returned by Function #2 (DSM Version 1 feature). • 0: The channel is not connected. No data are reported for this channel. • Non-zero: The value of the shunt resistor, must fit into a 32-bit unsigned integer.
EMI Enable/Disable Mask	Integer	0x0 ≤ bit mask ≤ 0xF	• Per channel bit allocation: CH1:CH2:CH3:CH4 • Bit values: - CHn = 1: EMI enabled for Channel ‘n’ - CHn = 0: EMI disabled for Channel ‘n’ (Private channel) • If CHn = 1 but the channel name string is NULL (""), the EMI interface cannot be created by the driver, so the channel remains “Private”.
Channel Polarity	Integer	0x00 ≤ bit mask ≤ 0xFF	• This parameter is the initialization value for the device NEG_PWR register (1Dh). • Per channel bit allocation: CH1_BIDI : CH2_BIDI : CH3_BIDI : CH4_BIDI : CH1_BIDV : CH2_BIDV : CH3_BIDV : CH4_BIDV • Bit values: - CHn_BIDI = 0: V_{SENSE} for Channel ‘n’ is unipolar - CHn_BIDI = 1: V_{SENSE} for Channel ‘n’ is bipolar - CHn_BIDV = 0: V_{BUS} for Channel ‘n’ is unipolar - CHn_BIDV = 1: V_{BUS} for Channel ‘n’ is bipolar
“ACTIVE” Sample Frequency	Integer	Valid Values = {1024, 256, 64, 8}	Sample frequency to be used when the system is “ACTIVE”. If the parameter value is not correct, the driver must keep the device POR default – 1024 Hz.

TABLE 2: PAC193X ACPI INTEGRATION PARAMETERS SUMMARY (CONTINUED)

Parameter	Type	Value	Comments
“IDLE” Sample Frequency	Integer	Valid Values = {1024, 256, 64, 8}	Sample frequency to be used when the system is in a Low-Power mode (SLEEP, CONNECTED-STANDBY, HIBERNATE, OFF). If the parameter value is not correct, the driver must use 8 Hz (same sps as when device SLOW pin is activated) in order to minimize the risk of accumulator saturation during long periods of system low power. (**Note**: if possible, the activation of the SLOW pin by hardware design ensures the 8 Hz sample rate during Low-Power mode.)
REFRESH Watchdog Timer Interval	Integer	60 ≤ time ≤ 900	• The value is expressed in seconds. • The driver must ensure that the time elapsed in “ACTIVE” mode between two consecutive REFRESH commands is not larger than this parameter. If there is no pending driver service request which needs a device REFRESH command to be serviced, the driver must initiate, by itself, a REFRESH in order to reset the device accumulators. • The driver must ignore an invalid parameter value and use the hard-coded default (240 seconds in Driver Versions up to 1.3).

REVISION HISTORY

Revision C (January 2022)

Added the PAC193X Windows Device Driver support for Windows 11. The information in this document applies to the PAC193X driver releases 1.4.1, 1.4.2 and later.

Revision B (June 2019)

The information in this document applies to the PAC193X driver releases up to Release 1.4.1. The new release introduces the ACPI DSM Rev. 1.0 for MCHP1930 devices.

Revision A (June 2017)

The information in this document applies to the PAC193X Driver Releases: 1.0, 1.1 and 1.2.

APPENDIX A: DEVICE DEFINITION CODE

EXAMPLE A-1: SAMPLE ACPI PAC193X DEVICE DEFINITION

```
Device(PA01)
{
    Name(_HID, "MCHP1930")
    Name(_UID, 1)

    // Lowest power D-State supported by the device is D3
    Name (_S0W, 3)

    // Device Status: present, enabled, and functioning properly
    Method(_STA, 0x0, NotSerialized)
    {
        Return(0xf)
    }

    // Current Resources Settings
    Method(_CRS, 0x0, NotSerialized)
    {
        Name(RBUF, ResourceTemplate()
        {
            I2CSerialBus(0x10,                    // 7-bit Client Address
                   ControllerInitiated,           // Client or host?
                   400000,                        // Connection Speed in hz
                   AddressingMode7Bit,            // 7-bit or 10-bit addressing?
                   "\\_SB.PCI0.I2C0",             // I2C Controller to which PAC is connected
                   0,                             // Resource Index
                   ResourceConsumer)              // Consumer or Producer?
        })
        Return(RBUF)
    }

    // _DSM - Device Specific Method
    //
    // This method returns configuration information that tells the driver
    // which devices each line in the PAC193x is wired to monitor.
    // Names of the monitored devices must come from the Microsoft-defined
    // power rail taxonomy.
    //
    // The UUID for the Microchip PAC193x's DSM is {033771E0-1705-47B4-9535-D1BBE14D9A09}.
    // This is unique to the device, and must match what the Windows PAC193x driver expects.
    //
    // Returns:
    //         Either: A Buffer (for supported Functions, or an error)
    //                 A Package containing PAC1934 resources (rail/resistor values/configs)
    //
    // Input Arguments (per _DSM standard):
    //     Arg0: UUID - Function Identifier
    //     Arg1: Integer - Revision
    //     Arg2: Integer - Function Index
    //     Arg3: Package - Parameters (not used in our implementation)
    //
    Function(_DSM, {BuffObj, PkgObj}, {BuffObj, IntObj, IntObj, PkgObj})
    {

        //  Is our UUID being invoked?
        if(LNotEqual(Arg0, ToUUID("033771E0-1705-47B4-9535-D1BBE14D9A09")))
        {
            return(Buffer() {0x0})                // incorrect UUID, return NULL for error
        }

        // Function number check
        switch(ToInteger(Arg2))
        {
            // Function 0 returns a bit-mask of supported functions
            case(0)
            {
                switch(ToInteger(Arg1))           // revision check
                {
                    // Revision 0: function 1 is supported
                    case(0) {return (Buffer() {0x3})}

                    // Revision 1: functions 1->6 are supported
                    case(1) {return (Buffer() {0x7f})}
                }
                break;
            }
```

EXAMPLE A-1: SAMPLE ACPI PAC193X DEVICE DEFINITION (CONTINUED)

```
            // Function 1 returns the channel "Rail Name" and
            // the Resistor Value expressed in milli-Ohms.
            case(1)
            {
                // All Revisions supported (0 and 1 presently defined)
                Name(BUF1, Package()
                {
                    // Rail Name,    Resistor Value
                    "GPU"       ,       24900,        // Channel 1
                    "CPU_CORES" ,       49900,        // Channel 2
                    "SENSORS"   ,       75000,        // Channel 3
                    "WIFI"      ,      100000         // Channel 4
                })
                return(BUF1)
            }

            // Function 2 returns the Resistor values expressed in micro-Ohms.
            case(2)
            {
                If(LLess(Arg1,1)){ break; }           // Revision 0 not supported.

                // Return enhanced precision resistor values.
                Name(BUF2, Package()
                {
                    // Value
                     24900000,             // Channel 1
                     49900000,             // Channel 2
                     75000000,             // Channel 3
                    100000000              // Channel 4
                })
                return(BUF2)
            }

            // Function 3 returns the EMI enabled/disabled flags
            case(3)
            {
                If(LLess(Arg1,1)){ break; }           // Revision 0 not supported.

                // Return ON/OFF flags.
                Name(BUF3, Package()
                {
                    0xF         // driver EMI ENABLE mask, CH1:CH2:CH3:CH4 - 1b=ON, 0b=OFF
                                // EMI mask bit set but rail name is NULL  -> EMI is not created (private channel)
                                // channel name valid but mask bit not set -> EMI is not created (private channel)
                })
                return(BUF3)
            }

            // Function 4 returns the channel bipolar settings
            case(4)
            {
                If(LLess(Arg1,1)){ break; }           // Revision 0 not supported.

                // Return the target NEG_PWR flags.
                Name(BUF4, Package()
                {
                    0x00                              // CH1:CH2:CH3:CH4-BIDI : CH1:CH2:CH3:CH4-BIDV
                })
                return(BUF4)
            }
```

EXAMPLE A-1: SAMPLE ACPI PAC193X DEVICE DEFINITION (CONTINUED)

```
            // Function 5 returns the SPS for ACTIVE and IDLE
            case(5)
            {
                If(LLess(Arg1,1)){ break; }   // Revision 0 not supported.

                Name(BUF5, Package()
                {
                    1024,                     // ACTIVE sps - accepted values = {1024, 256, 64, 8}
                    8                         // IDLE sps   - accepted values = {1024, 256, 64, 8}
                })
                return(BUF5)
            }

            // Function 6 returns the watchdog interval
            case(6)
            {
                If(LLess(Arg1,1)){ break; }   // Revision 0 not supported.

                Name(BUF6, Package()
                {
                    900                       // seconds (min=60, max=900)
                })
                return(BUF6)
            }

        } // switch(Arg2)

        // Return an error (a buffer with a value of zero)
        // if we didn't return anything else above
        return(Buffer() {0x0})

    } // _DSM

} // Device(PA01)
```

APPENDIX B: TEST THE PAC193X ACPI INTEGRATION

CAUTION: The tools and the process presented in this appendix are not the only method to integrate the PAC193X ASL, compile and deploy it on a target system. This method does not replace the original system BIOS firmware from the Flash and works only if Windows 10 or Windows 11 OS on the target system is configured in Test mode. The method works only if the target system already has Windows 10 or Windows 11 installed on it and the user has the possibility to install and run on it the presented tools. ASL code changes and deployment must be done with great care because of the potential risk to cause the Windows OS to crash and the loss of data.

The Tools

- iASL and AcpiDump tools from ACPICA: https://acpica.org/downloads/binary-tools
- ASL tool from Microsoft Windows Driver Kit (WDK): https://docs.microsoft.com/en-us/windows-hardware/drivers/bringup/microsoft-asl-compiler (use the 64-bit or the 32-bit version, depending on the target system OS type)
- BCDEdit utility, installed by default in Windows 10 or Windows 11
- Notepad utility (installed by default in Windows 10 or Windows 11) or any other text editor that runs on the target system
- Shutdown utility (installed by default in Windows 10 or Windows 11)

The Process

1. Copy the tools and the sample ASL code to the target system:
   - Create a working folder (ex: ACPI_MCHP1930) and copy the PAC193X ASL sample code in a simple text file.
   - Copy the ASL and the ACPICA tools in the working folder or, alternatively, create for them a dedicated Tools folder and put it in the system path.
2. Dump the target system ACPI code:
   - Open an elevated command prompt (administrator command prompt) and change the working directory to the working folder just created.
   - Use the AcpiDump tool to obtain the system ACPI tables in binary format:

     ```
     C:\ACPI_MCHP1930>acpidump -b
     ```

     Several files will be created on the working folder, one for each ACPI table type, having the name of the contained ACPI table and the `.dat` extension.
3. Disassemble the binary ACPI code:
   - Use the iASL tool to disassemble the binary ACPI code into ASL code:

     ```
     C:\ACPI_MCHP1930>iasl -d *.dat
     ```

     Several files will be created on the working folder, one for each ACPI table type, having the name of the contained ACPI table and the `.dsl` extension.
4. Add the PAC193X ASL code into ACPI SSDT:
   - Open the `ssdt.dsl` file in the text editor, add the PAC193X ASL sample code and modify it to match the target system architecture.

   Tip: Look for the I^2C controller ACPI definition in DSDT or SSDT and note the ACPI name. This name is needed to define the PAC193X device I^2C connection; see the I^2C Serial Bus section from **Appendix A: "Device Definition Code"**.
5. Compile the modified SSDT:
   - Use iASL tool to generate the binary SSDT code:

     ```
     C:\ACPI_MCHP1930>iasl ssdt.dsl
     ```

   The file, `ssdt.aml`, is generated if there is no compilation error.
6. Load the new ACPI SSDT in the Windows registry.

   Windows `acpi.sys` driver provides the ACPI code developers the option to override the original ACPI code from the system Flash with a different version saved in the Windows registry.
   - Use the ASL tool to save the ACPI table into the Windows registry:

     ```
     C:\ACPI_MCHP1930>asl /loadtable -v
     ssdt.aml
     ```

   Note: This command works only from the elevated command prompt.

   Tip: In case the Windows registry already contains an ACPI code override, it must be "unloaded" before loading the new version. Provide the option, "-d", to the ASL tool in order to achieve this:

   ```
   C:\ACPI_MCHP1930>asl /loadtable -v
   -d ssdt.aml
   ```
7. Switch Windows in **Test mode**.

   The ACPI table override feature presented above is enabled only in Windows **Test mode**.
   - Use the BCDEdit utility to enable the Test mode:

     ```
     C:\ACPI_MCHP1930>bcdedit /set
     testsigning ON
     ```

   Tip: In order to disable the Test mode, replace ON with OFF in the above example.

8. Reboot the system
   - Use the Shutdown utility in order to make sure that a full power cycle is executed, not affected by the Windows 10 or Windows 11 Fast Start facility:

     ```
     C:\ACPI_MCHP1930>shutdown -r -t 0
     ```
9. Check the **Device Manager**.

   At this stage, because there is no driver installed yet, the PAC193X devices should be enumerated by the Windows **Device Manager** utility under **Unknown Devices** category.
10. Install the PAC193X Windows device driver.

Read and follow the steps presented by the Driver installation section from *"PAC193X Microsoft Windows® 10 and Windows 11 Device Driver User's Guide"*.

CAUTION: If a BIOS upgrade or a BIOS configuration change is needed, the following is recommended:

- First, disable the ACPI code override, either by unloading the code from the Windows registry (see Step #6) or by turning off the Test mode (see the Step #7).
- Repeat the steps listed above, starting with Step #2.

NOTES:

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

- Microchip products meet the specifications contained in their particular Microchip Data Sheet.
- Microchip believes that its family of products is secure when used in the intended manner, within operating specifications, and under normal conditions.
- Microchip values and aggressively protects its intellectual property rights. Attempts to breach the code protection features of Microchip product is strictly prohibited and may violate the Digital Millennium Copyright Act.
- Neither Microchip nor any other semiconductor manufacturer can guarantee the security of its code. Code protection does not mean that we are guaranteeing the product is “unbreakable”. Code protection is constantly evolving. Microchip is committed to continuously improving the code protection features of our products.

For information regarding Microchip’s Quality Management Systems, please visit www.microchip.com/quality.

Trademarks

ISBN: 978-1-5224-9457-7

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: 45-4485-5910
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: 49-7131-72400

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: 47-7288-4388

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