AN1982

Interrupt System in tinyAVR® 0- and 1-series, and megaAVR® 0-series

Features

- Interrupt Controller Overview
- Interrupt Priority Configuration
- Vector Table Configuration
- Code Examples

Introduction

Microcontrollers (MCUs) contain a wide range of hardware modules (peripherals) designed to perform specialized tasks like communication, timing and waveform generation. Typically, each peripheral has a number of signals indicating its status, an operation is completed, data availability or the peripherals are ready to receive new commands. It is up to the central processing unit (CPU) to check each module for relevant updates. The CPU then has to service the requests from each module. With an increasing number of peripheral modules, the CPU spends increasingly more time to check and service all modules. This CPU load causes longer response time and higher power consumption. Traditionally, MCUs use two main methods to monitor and service peripherals: polling and interrupts.

Polling means manually reading and checking a status bit updated by a monitored peripheral. This gives the designer a high degree of freedom to decide how often and in which order to check the different peripherals. In a simple use case where the application only waits for a specific status, a very tight loop can be made; checking this status bit and immediately servicing it when it occurs. Even though this approach would allow a very fast response time, there are few major drawbacks using the polling method. First, the responsiveness decreases as the number of status bits to check increases. Second, the CPU needs to run in active mode while it is executing the code testing every status bit, which increases power consumption.

As the number of peripherals and status bits involved in an application increases, an interrupt system provides better responsiveness. An interrupt system is a scheme in which the peripherals can send a request to interrupt the CPU execution. Since the peripherals prompt the CPU when service is needed, there is no need for the CPU to actively poll the status bits. However, when a peripheral can request service from the CPU at any time, with no regard to other peripherals' needs, this creates the risk of an interrupt request pile-up. Therefore, an interrupt controller is used to determine the order in which pending interrupts will be serviced by the CPU. Traditionally, interrupt handling for tinyAVR® and megaAVR® devices use a predefined priority, based on the interrupt vector address. The XMEGA® MCU family makes it possible to tailor the priority queue using a Programmable Multilevel Interrupt Controller (PMIC), which allows the user to assign interrupts to three priority levels. Interrupt handling techniques have become more configurable in the tinyAVR 0- and 1-series, and megaAVR 0-series. This application note describes the functionality of the new interrupt controller in detail, which uses a combination of predefined priority and multilevel controller.

Table of Contents

1. Relevant Devices

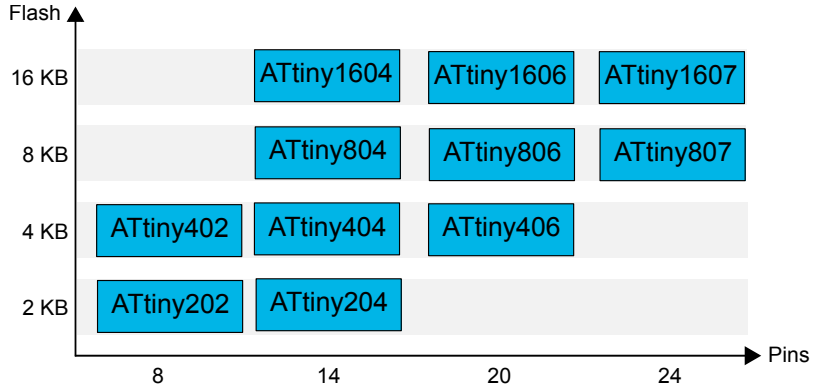
This chapter lists the relevant devices for this document.

1.1 tinyAVR® 0-Series

The figure below shows the tinyAVR® 0-series, laying out pin count variants and memory sizes:

- Vertical migration is possible without code modification, as these devices are fully pin- and feature compatible.
- Horizontal migration to the left reduces the pin count and therefore the available features.

Figure 1-1. tinyAVR® 0-Series Overview



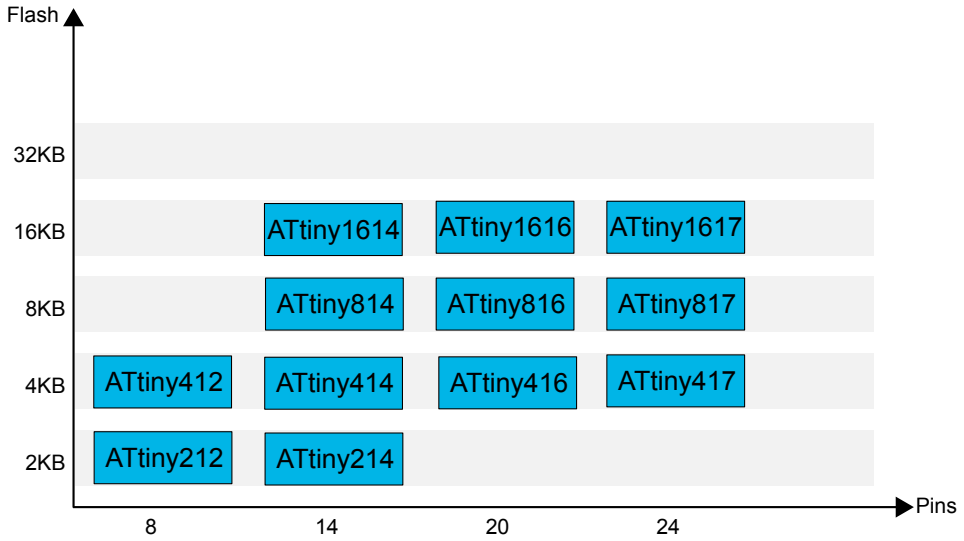
Devices with different Flash memory size typically also have different SRAM and EEPROM.

1.2 tinyAVR® 1-Series

The figure below shows the tinyAVR® 1-series devices, laying out pin count variants and memory sizes:

- Vertical migration upwards is possible without code modification, as these devices are pin compatible and provide the same or more features. Downward migration may require code modification due to fewer available instances of some peripherals.
- Horizontal migration to the left reduces the pin count and therefore the available features.

Figure 1-2. tinyAVR® 1-Series Overview



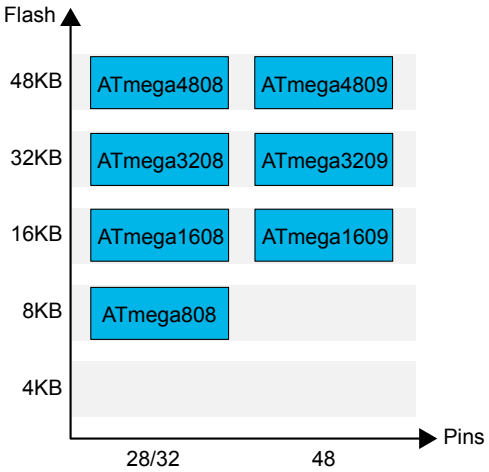
Devices with different Flash memory size typically also have different SRAM and EEPROM.

1.3 megaAVR® 0-Series

Figure 1-3 shows the feature compatible devices in the megaAVR® device family, including pinout variants and memory variants.

- Vertical migration is possible without code modification, as these devices are fully pin and feature compatible.
- Horizontal migration to the left reduces the pin count and therefore the available features.

Figure 1-3. Device Family Overview



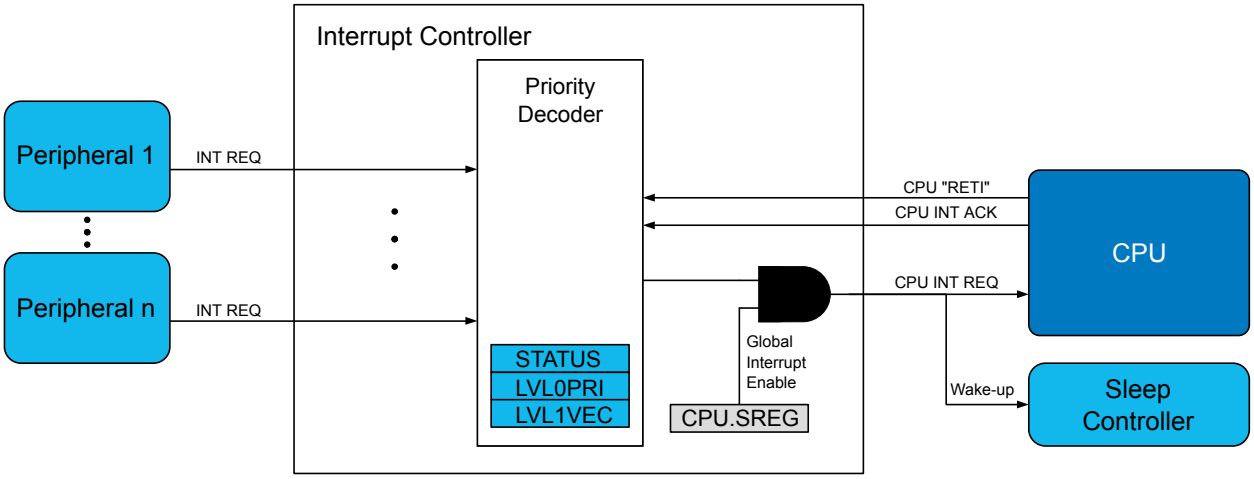
Devices with different Flash memory size typically also have different SRAM and EEPROM.

2. Interrupt Controller Overview

The tinyAVR 0- and 1-series, and megaAVR 0-series Interrupt Controller has support for multiple individual Interrupt Requests (IRQs), with dedicated interrupt vectors. This helps reduce latency when an interrupt is triggered.

When an interrupt is enabled and the interrupt condition occurs, the CPU will receive an IRQ from the Interrupt Controller. Based on the interrupt's priority level and the priority level of any ongoing Interrupt Service Routine (ISR), the IRQ is either acknowledged (and its ISR is executed) or kept pending. See Figure 2-1 for a graphical representation of the Interrupt Controller interaction.

Figure 2-1. Interrupt Controller Flowchart



The Interrupt Controller decides the order of execution for pending ISRs by the combination of the interrupts priority level and vector address. The hardwired vector addresses and different priority modes are configurable from the CPUINT registers, with the following available priority levels:

1. Non-Maskable Interrupts
2. Level 1 Priority Interrupts
3. Level 0 Priority Interrupts

Refer to the device datasheet for the interrupt vector mapping.

Interrupt Flags

The Global Interrupt Enable flag (I-flag) must be set for IRQs to be generated, with Non-Maskable Interrupts as the only exception. Information on these can be found under Non-Maskable Interrupts.

The majority of interrupt flags must be cleared in the corresponding ISR. This is because some interrupt sources share interrupt vectors. Other flags are cleared by hardware when entering the ISR or reading a data register. For detailed information, refer to each peripheral's chapter in the device datasheet.

Interrupt Latency

The CPU's interrupt latency is determined by the number of clock cycles required for the following actions:

1. Complete the ongoing instruction.
   - 1 - 3 clock cycles, dependent on instruction time.
2. Push the program counter on the stack.

– 2 clock cycles.

3. Execute the jump instruction stored in the interrupt vector table.
   – 2 - 3 clock cycles, dependent on Flash size. See Table 2-1.

Table 2-1. Interrupt Jump Instruction

Flash size	Jump instruction	Jump Execution Time
≤ 8kB	RJMP	2 clock cycles
> 8kB	JMP	3 clock cycles

The total interrupt latency will be five to eight clock cycles, depending on the parameters listed above. See the AVR Instruction Set Manual for more information.

If the device is in Sleep mode, the response time increases by five clock cycles, in addition to the CPU start-up time from the used Sleep mode.

3. Static Priority Interrupts

The tinyAVR 0- and 1-series, and megaAVR 0-series by default use an interrupt vector table with static priority. This is identical to other classic tinyAVR® and megaAVR® MCUs and for most standard applications this is the preferred method of operation.

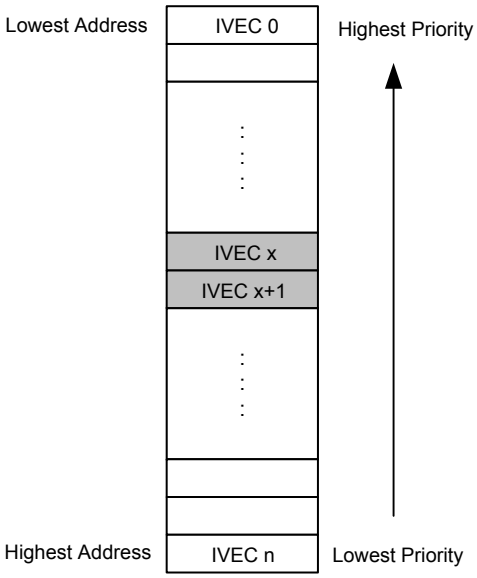
The Static Priority Interrupt scheme is applicable for priority level 0 interrupts only. Information on higher level interrupts is located in the High Priority Interrupt Vector and Non-Maskable Interrupts chapters.

3.1 Operation

When using a static priority interrupt vector table, the lowest interrupt vector address has the highest priority. This means, when two or more interrupts are pending, the order of execution will be set by the interrupt vector address, in ascending order. See Figure 3-1 for a graphical representation.

While the CPU is executing an ISR, any new IRQs will be kept pending until the running routine is complete, and at that time the interrupt with the lowest vector address (highest priority) will be executed first.

Figure 3-1. Interrupt Priority Using Static Priority Scheme



In Static Priority Interrupt Scheme the CPUINT.LVL0PRI register defines the starting point for the highest priority interrupt, so it is possible to statically shift the priority in the vector table by writing the desired interrupt vector address to LVL0PRI. The default value of this register is 0x00, so after reset, the lowest address interrupt vector will have the highest priority.

3.2 Example

Static Priority Interrupt scheme is the default setting for the interrupt controller in tinyAVR 0- and 1-series, and megaAVR 0-series devices, so the CPUINT registers are configured for this at startup. The following code snippet shows an example on how to configure the Static Priority Interrupt scheme.

Static Priority Interrupt scheme configuration

```
// io.h includes the device header file with register and vector defines
#include <avr/io.h>
// interrupt.h contains the ISR related content
#include <avr/interrupt.h>

void static_priority_interrupt_example(void) {
    // If needed, clear the Round-robin Scheduling Enable bit.
    CPUINT.CTRLA &= ~CPUINT_LVL0RR_bm;

    // Example on how to shift the interrupt priority.
    CPUINT.LVL0PRI = PORTA_PORT_vect_num;

    // Enable global interrupts
    sei();
}

ISR(PORTA_PORT_vect){
    // This interrupt will have highest priority.
}
```

4. Round-Robin Priority Scheme

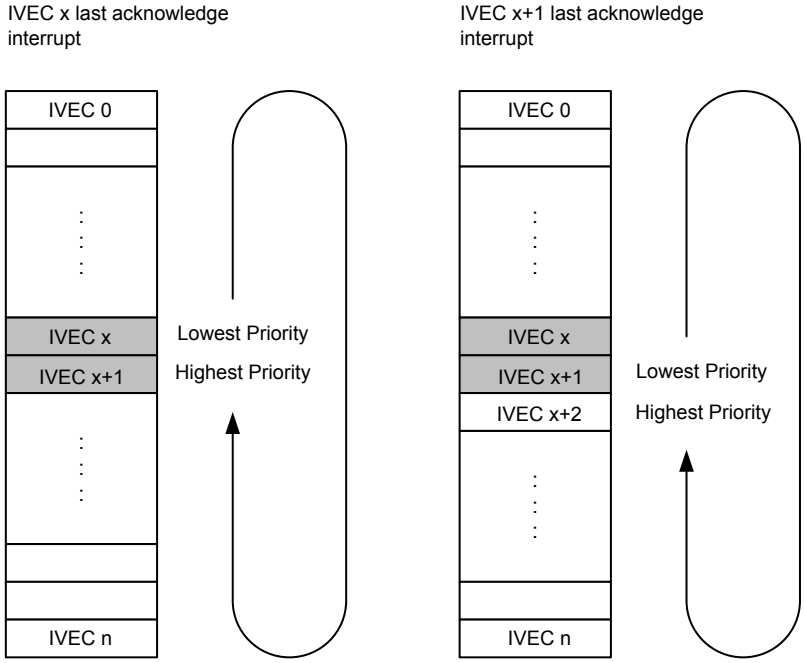
In some cases where Static Priority Interrupt schemes are used, interrupt starvation may occur. This is when applications have higher priority (lower vector address) interrupts that are triggered too often, causing a lack of service for the lower priority interrupts. The Round-Robin Priority Scheme aims to prevent interrupt starvation.

The Round-Robin Priority Scheme is valid for priority level 0 interrupts. Descriptions of higher level interrupts are located under the High Priority Interrupt Vector and Non-Maskable Interrupts chapters.

4.1 Operation

When using the Round-Robin Priority Scheme, the interrupt controller will store the address of the latest acknowledged interrupt; in the event that more than one interrupt request is pending, the stored address will have the lowest priority when the next interrupt is due to be serviced. Consequently, the next higher interrupt vector address will be assigned the highest priority. See Figure 4-1 for a graphical representation of the scheme.

Figure 4-1. Interrupt Priority Using Round-Robin Interrupt Scheme



The Round-Robin Priority Scheme is enabled by writing '1' to the LVL0RR bit in the CPUINT.CTRLA register. The CPUINT.LVL0PRI register will contain the address of the last acknowledged interrupt. This becomes the lowest priority interrupt when the interrupt controller decides which interrupt to execute next.

By changing the value in CPUINT.LVL0PRI it is possible for the CPU to change which interrupts have the lowest and highest priority next. The default value of this register is 0x00, so after reset the lowest interrupt vector will have the highest priority.

4.2 Example

The following code snippet shows an example on how to configure the Round-robin Scheduling scheme.

Round-robin Interrupt Scheduling configuration

```
// io.h includes the device header file with register and vector defines
#include <avr/io.h>
// interrupt.h contains the ISR related content
#include <avr/interrupt.h>

void round_robin_interrupt_scheduling_example(void) {
    // Set the Round-robin Scheduling Enable bit.
    CPUINT.CTRLA |= CPUINT_LVL0RR_bm;

    // Example on how to shift the initial interrupt priority.
    CPUINT.LVL0PRI = PORTA_PORT_vect_num;

    // Enable global interrupts
    sei();
}

ISR(PORTA_PORT_vect){
    // This interrupt will initially have lowest priority, however
    // that will change since the LVL0PRI will be updated by the interrupt
    // controller each time an interrupt is executed.
}
```

Atmel | START example

A use case example for Round-robin Interrupt Scheduling using Atmel | START is also available. For more information, see the Get Source Code from Atmel | START chapter.

5. High Priority Interrupt Vector

When a higher interrupt priority is required over what is already defined, one interrupt vector can be assigned priority level 1. That interrupt will have priority over all level 0 interrupts and has the ability to interrupt an ongoing level 0 interrupt handler.

All interrupt sources are by default assigned level 0 priority, except for Non-Maskable Interrupts (NMI). These are the only interrupts that have higher priority than the level 1 interrupt. For more info see the Non-Maskable Interrupts chapter.

5.1 Operation

To use the high priority vector feature, simply write the desired interrupt vector address to the CPUINT.LVL1VEC register and this interrupt will be assigned level 1 priority. This feature is disabled when the register value is 0x00. Refer to the device datasheet for the interrupt vector mapping.

If interrupted by the level 1 interrupt, the level 0 handler will resume execution when the level 1 handler is done. It is important to make sure a level 1 interrupt will not interfere with the execution of any time critical level 0 interrupts. The LVL0EX bit in the CPUINT.STATUS register can be read to check if the level 1 interrupt has interrupted an executing level 0 ISR.

5.2 Example

The following code snippet shows an example on how to configure the High Priority Interrupt vector.

High Priority Interrupt configuration

```
// io.h includes the device header file with register and vector defines
#include <avr/io.h>
// interrupt.h contains the ISR related content
#include <avr/interrupt.h>

void high_priority_interrupt_example(void) {
    // Set the level 1 interrupt vector
    CPUINT.LVL1VEC = PORTA_PORT_vect_num;

    // Enable global interrupts
    sei();
}

ISR(PORTA_PORT_vect){
    // This interrupt will have level 1 priority
    // after the configuration in CPUINT.LVL1VEC.

    // Check if a level 0 ISR has been interrupted
    if (CPUINT.STATUS & CPUINT_LVL0EX_bm) {

    }
}
```

6. Non-Maskable Interrupts

Non-Maskable Interrupts (NMI) are usually system critical interrupts and will always take priority over all other interrupts. Also, they will be acknowledged even when global interrupts are disabled.

6.1 Operation

There are specific vectors that have a hardwired NMI priority level, which cannot be changed. They must be enabled the same way as other interrupts. Refer to the Interrupt Vector Mapping of the device datasheet for available NMI sources.

By reading the LVL0EX and LVL1EX bits in the CPUINT.STATUS register, the application can check whether any level 0 or level 1 ISR has been interrupted by an NMI.

6.2 Example

The following code snippet demonstrates how a Non-Maskable Interrupt can be implemented.

Non-Maskable Interrupt configuration, CRCSCAN

```
// io.h includes the device header file with register and vector defines
#include <avr/io.h>
// interrupt.h contains the ISR related content
#include <avr/interrupt.h>

void nmi_example(void) {
    // Configure the CRCSCAN peripheral with NMI
    CRCSCAN.CTRLB = CRCSCAN_MODE_PRIORITY_gc | CRCSCAN_SRC_FLASH_gc;
    CRCSCAN.CTRLA = CRCSCAN_NMIEN_bm | CRCSCAN_ENABLE_bm;
}

ISR(CRCSCAN_NMI_vect) {
    // If the CRC scan fails, the NMI interrupt will be handled here
    // The NMI request remains active until a system Reset, and can not be
disabled

    // Check if level 0 or 1 ISRs were interrupted
    if (CPUINT.STATUS & (CPUINT_LVL0EX_bm | CPUINT_LVL1EX_bm) {

    }
}
```

For more information on how to prepare a code project for using the CRCSCAN peripheral, refer to Application Note AN2521 - CRCSCAN on Devices in the tinyAVR® 1-Series

7. Compact Vector Table

If an application is using only a few interrupts, only those vector locations are needed; therefore, most of the vector table will be unused space. The Compact Vector Table (CVT) is an approach to a more efficient interrupt management scheme, since this will truncate the interrupt vector table to a few vectors.

7.1 Operation

The tinyAVR 0- and 1-series, and megaAVR 0-series MCUs support a three-vector CVT mode, where the interrupt vector table is reorganized per Table 7-1. Each vector will be used to service all interrupts of corresponding priority level.

Table 7-1. Compact Vector Mapping

Vector Number	Peripheral Source	Definition
0	RESET	RESET
1	NMI	Non-Maskable Interrupt vector
2	LVL1	Level 1 Interrupt vector
3	LVL0	Level 0 Interrupt vector

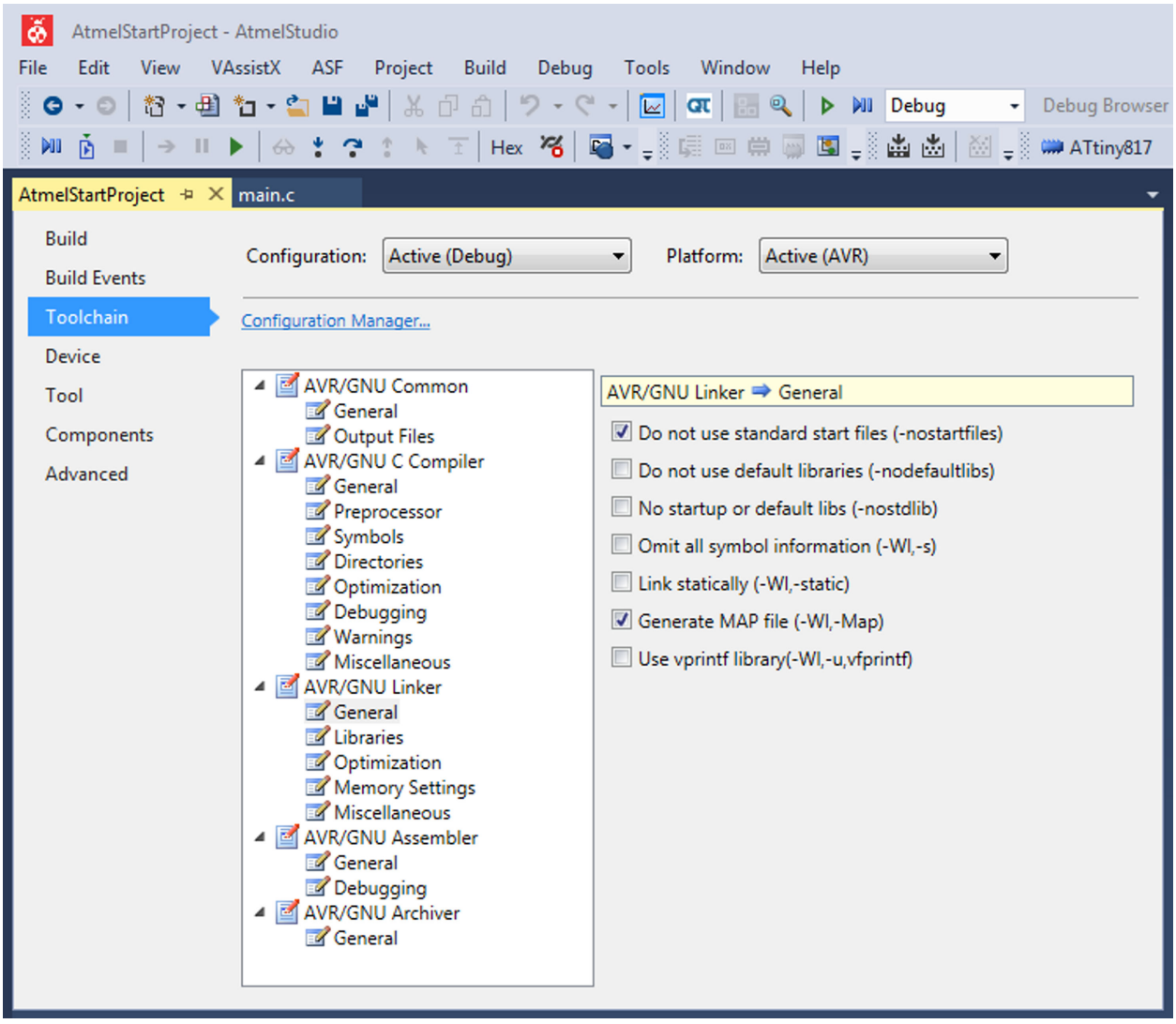
This can be very useful when an application only needs a few interrupts. Reducing the vector table size means it will occupy less Flash, freeing up space for application code. Be aware, however, when using this approach, additional code may be needed in the ISR to find out which interrupt is actually requesting service as this will add execution time.

The Compact Vector Table is enabled by writing '1' to the CVT bit in CPUINT.CTRLA. This system critical register bit has Configuration Change Protection (CCP) to prevent accidental modification. Refer to the CPU chapter in the relevant device data sheet for details on CCP.

7.2 Example

To take advantage of the smaller vector table it is necessary to replace the standard start files used by AVR-GCC. These are disabled by setting the linker flag *-nostartfiles* when compiling the project. In Atmel Studio 7.0 this can be found in Project Properties (*Alt+F7*) - Toolchain - AVR/GNU Linker - General, as seen in Figure 7-1.

Figure 7-1. Configuring "Do not use standard start files", Atmel Studio 7.0



The following code snippet shows an example for the code needed to replace the standard start files. In a future Toolchain release, this step will not be needed to configure the Compact Vector Table.

Example start-up code for Compact Vector Table

```
// io.h includes the device header file with necessary defines
#include <avr/io.h>

// Letting the compiler know there will be something called main
extern int main(void);

// Being nice to the compiler by predeclaring the ISRs
void __vector_cvt_nmi(void) __attribute__((weak, alias("dummy_handler")));
void __vector_cvt_lvl1(void) __attribute__((weak, alias("dummy_handler")));
void __vector_cvt_lvl0(void) __attribute__((weak, alias("dummy_handler")));

// Setting up the vector section
// The rjmp instruction can handle 8k code space, so this is used for
// vector tables on devices with 8k flash or smaller. Other devices
// use the jmp instruction.
__attribute__((section(".vectors"), naked)) void vectors(void)
{
#if (PROGMEM_SIZE <= 0x2000)
```

```
    asm("rjmp __ctors_end");
    asm("rjmp __vector_cvt_nmi");
    asm("rjmp __vector_cvt_lvl1");
    asm("rjmp __vector_cvt_lvl0");
#else
    asm("jmp __ctors_end");
    asm("jmp __vector_cvt_nmi");
    asm("jmp __vector_cvt_lvl1");
    asm("jmp __vector_cvt_lvl0");
#endif
}

// Initialize the AVR core
__attribute__((section(".init2"), naked)) void init2(void)
{
    // GCC expects r1 to be zero
    asm("clr r1");
    // Make sure the status register has a known state
    SREG = 0;
    // Point the stack pointer to the end of stack
    SPL  = INTERNAL_SRAM_END & 0xff; // (low byte)
    SPH  = (INTERNAL_SRAM_END >> 8); // (high byte)
}

// Handling main
__attribute__((section(".init9"), naked)) void init9(void)
{
    main();

    // Jump to avr libc defined exit handler
    // (Disables interrupts and runs an empty loop forever)
    asm("jmp _exit");
}

// Dummy handler alias for unused ISRs
void dummy_handler(void)
{
    while (1)
        ;
}
```

The following code snippet shows an example on how to configure the Compact Vector Table, using the interrupt vector names defined in the example start-up code from above.

Compact Vector Table configuration code

```
// io.h includes the device header file with register and vector defines,
// it also includes xmega.h, containing the CCP macro _PROTECTED_WRITE().
#include <avr/io.h>
// interrupt.h contains the ISR related content
#include <avr/interrupt.h>

void compact_vector_table_example(void) {
    // Set the Compact Vector Table
    // read-modify-write to preserve other configured bits
    _PROTECTED_WRITE(CPUINT.CTRLA, (CPUINT.CTRLA | CPUINT_CVT_bm));

    //Enable global interrupts
    sei();
}

ISR(__vector_cvt_lvl0) {
    // All level 0 interrupt will be handled here
    if (PORTA.INTFLAGS) {
        // Example on how to handle PORTA interrupts in CVT mode
    }
}

ISR(__vector_cvt_lvl1) {
    // If enabled, level 1 interrupt will be handled here
```

```
}

ISR(__vector_cvt_nmi) {
    // If enabled, NMI interrupts will be handled here
}
```

Atmel | START example

A use case example for Compact Vector Table using Atmel | START is also available. For more information, see the Get Source Code from Atmel | START chapter.

8. Relocating Vector Table

By default, the vector table is located at the application section in the tinyAVR 0- and 1-series, and megaAVR 0-series devices. However, it is possible to relocate the vector table to the start of the boot section; a bootloader can relocate the vector table location in its code, and then change the location back before entering the main application.

It is also possible to combine the relocation of the vector table with use of the Compact Vector Table (CVT), for more information on the CVT see the Compact Vector Table chapter.

8.1 Operation

Moving the interrupt vector table to the start of the boot section in Flash is enabled by writing '1' to the IVSEL bit in CPUINT.CTRLA. This system critical register bit has Configuration Change Protection (CCP) to prevent accidental modification. Refer to the CPU chapter in the relevant device data sheet for details on CCP.

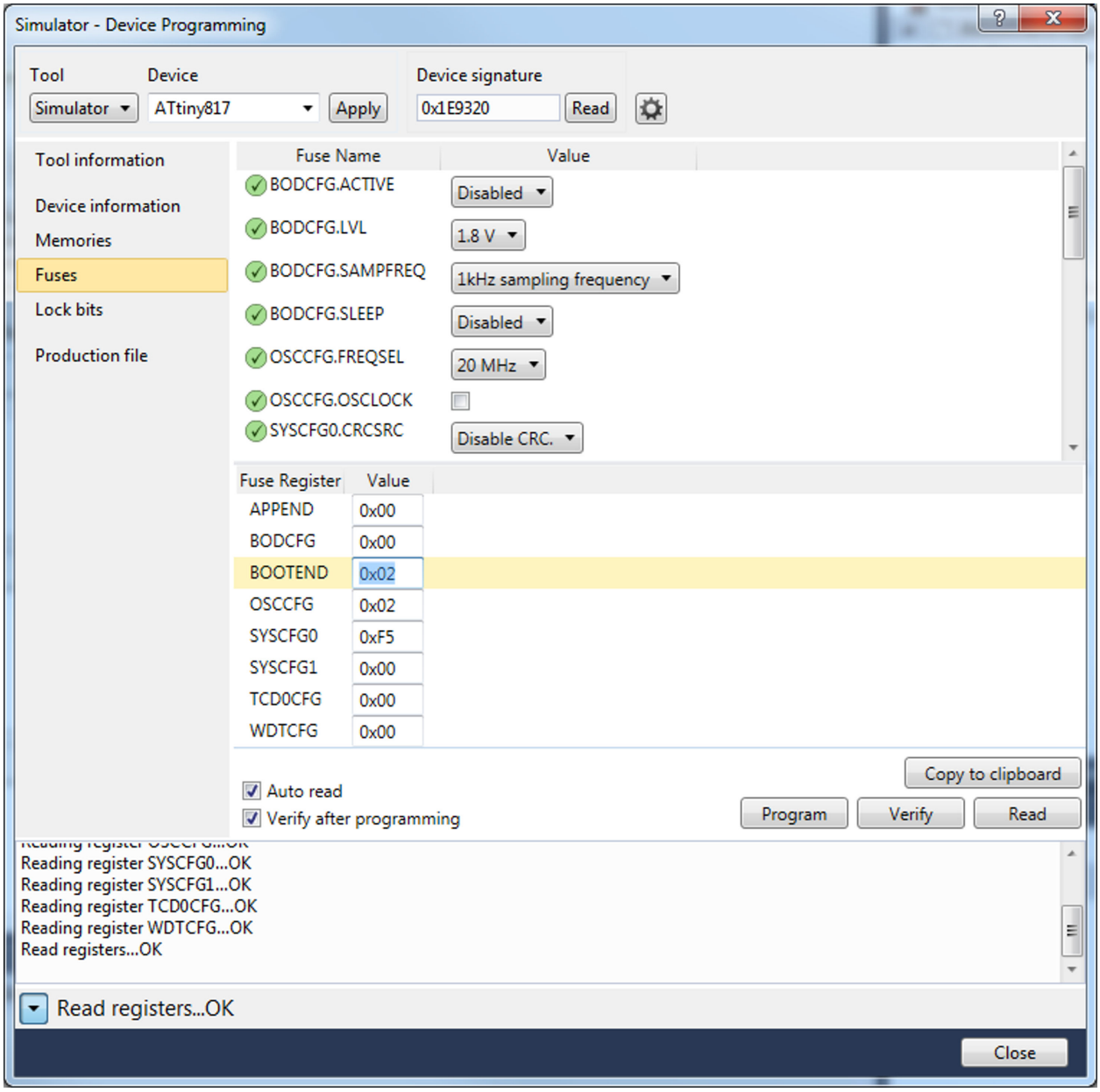
Since the application section is placed after the boot section, the actual address of the interrupt vector table is determined by the BOOTEND fuse. This fuse sets the size of the boot section in blocks of 256 bytes. When IVSEL is set, the interrupt vector table will be at the start of Flash. Be aware that if both the BOOTEND fuse and APPEND fuse (size of application section) are set to 0x00 the entire Flash is configured to be the boot, and the interrupt vector table will be at the start of Flash.

The Reset vector address is 0x0000 (Start of Flash) in both IVSEL states. This corresponds with the lowest address in the boot interrupt vector table, which means the Reset vector will be the first instruction to execute after a reset has occurred.

8.2 Example

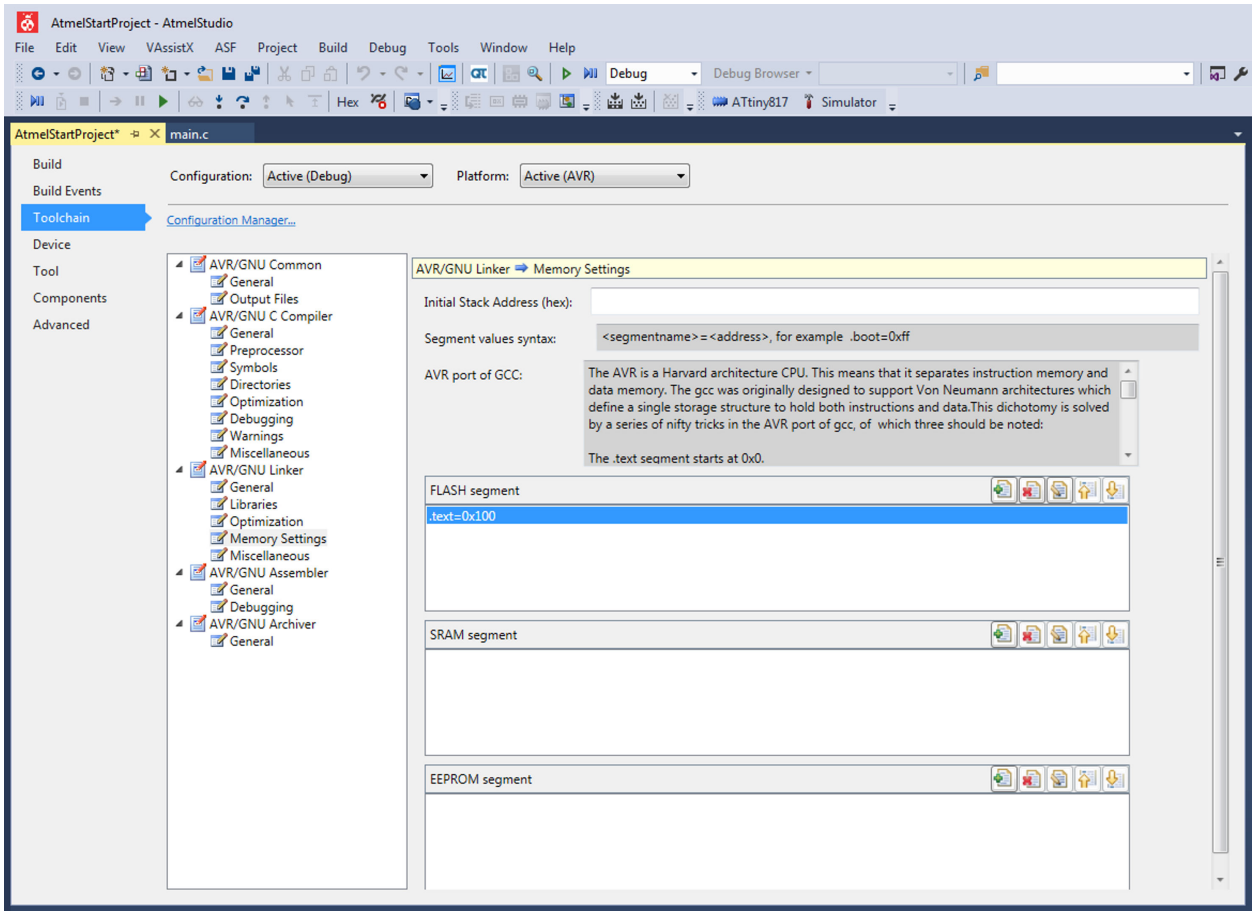
This example will use a bootloader section with 512 bytes size. This means the BOOTEND fuse needs to be configured to 512/256 = 0x02. In Atmel Studio 7.0 this fuse can be written to the device using Device Programming (*Ctrl+Shift+P*) - Fuses, as seen in Figure 8-1.

Figure 8-1. Configure BOOTEND fuse, Atmel Studio 7.0



The AVR® GCC project must also be configured to shift the start of the application section by 512 bytes. This is done by changing the start of the *text* segment in the linker, and this value must be set in hexadecimal word size, for example: *-Wl,-section-start=bootloader=0x100*. In Atmel Studio 7.0 this can be found in Project Properties (*Alt+F7*) - Toolchain - AVR/GNU Linker - Memory Settings and adding *.text=0x100* to the FLASH segment, as seen in Figure 8-2.

Figure 8-2. Configure application section start, AVR GCC, Atmel Studio 7.0



The following code example shows how to relocate the vector table.

Relocating Vector Table configuration

```
// io.h includes the device header file with register and vector defines,
// it also includes xmega.h, containing the CCP macro _PROTECTED_WRITE().
#include <avr/io.h>
// interrupt.h contains the ISR related content
#include <avr/interrupt.h>

void relocating_vector_table_example(void) {
    // Set the Interrupt Vector Select bit,
    // read-modify-write to preserve other configured bits
    _PROTECTED_WRITE(CPUINT.CTRLA, (CPUINT.CTRLA | CPUINT_IVSEL_bm));

    //Enable global interrupts
    sei();
}

ISR(PORTA_PORT_vect){
    // After setting the IVSEL bit, the interrupt vector table
    // will now be shifted from BOOTEND*256 bytes to start of Flash (0x0000).

    // When using an interrupt in the application code, IVSEL must be cleared
and
    // the code compiled with the .text section in Flash set to BOOTEND*128
words.
}
```

9. Summary

The choice between polling and interrupts depends heavily on the application requirements, however, it is possible to follow a few guidelines.

First, if your product is battery powered or otherwise power restricted, avoid polling. Polling requires the CPU to stay in Active mode, potentially consuming a lot more power than an interrupt driven system. However, if increased power consumption is not an issue, and only one or two status bits need checking, polling is very easy to implement in software.

Second, it can be difficult to manage and maintain a complex application in an effective and predictable way. This is made easier with the available interrupt priority schemes and the possibility to elevate one interrupt to a higher priority level.

Third, if an application is restricted in code size, this can be reduced by using the compact vector table. In some cases, it could be advantageous to use a compact vector table only for the bootloader and then switch back to the full-size table in the application section. This could potentially reduce boot section size, leaving more space for the application section.

As shown in this application note, the new tinyAVR 0- and 1-series, and megaAVR 0-series MCUs offer powerful flow control features with respect to interrupt handling. This allows designers to customize power consumption and responsiveness to fit their design requirements.

10. Related Documents

The following documents are related to the devices and topics covered by this application note.

- AVR Instruction Set Manual:
  - http://www.microchip.com/mymicrochip/filehandler.aspx?ddocname=en589391
- AN2521 - CRCSCAN on Devices in the tinyAVR® 1-Series:
  - http://www.microchip.com//wwwAppNotes/AppNotes.aspx?appnote=en599876

11. Get Source Code from Atmel | START

The example code is available through Atmel | START, which is a web-based tool that enables configuration of application code through a Graphical User Interface (GUI). The code can be downloaded for both Atmel Studio 7.0 and IAR Embedded Workbench® via the direct example code-link(s) below or the *BROWSE EXAMPLES* button on the Atmel | START front page.

Atmel | START web page: http://microchip.com/start

Example Code

- CPUINT Round-Robin Interrupt Scheme:
  - http://start.atmel.com/#example/Atmel:cpuint_examples_tinyavr_megaavr_01:1.0.0::Application:CPUINT_Round-Robin_Scheme:
- CPUINT Compact Vector Table:
  - http://start.atmel.com/#example/Atmel:cpuint_examples_tinyavr_megaavr_01:1.0.0::Application:CPUINT_Compact_Vector_Table:

Press *User guide* in Atmel | START for details and information about example projects. The *User guide* button can be found in the example browser, and by clicking the project name in the dashboard view within the Atmel | START project configurator.

Atmel Studio

Download the code as an .atzip file for Atmel Studio from the example browser in Atmel | START, by clicking *DOWNLOAD SELECTED EXAMPLE*. To download the file from within Atmel | START, click *EXPORT PROJECT* followed by *DOWNLOAD PACK*.

Double-click the downloaded .atzip file and the project will be imported to Atmel Studio 7.0.

IAR Embedded Workbench

For information on how to import the project in IAR Embedded Workbench, open the Atmel | START user guide, select *Using Atmel Start Output in External Tools*, and *IAR Embedded Workbench*. A link to the Atmel | START user guide can be found by clicking *About* from the Atmel | START front page or *Help And Support* within the project configurator, both located in the upper right corner of the page.

12. Revision History

Doc. Rev.	Date	Comments
B	02/2018	Updated description of Interrupt Vector Select (IVSEL)
A	12/2017	Initial document release

The Microchip Web Site

Microchip provides online support via our web site at http://www.microchip.com/. This web site is used as a means to make files and information easily available to customers. Accessible by using your favorite Internet browser, the web site contains the following information:

- **Product Support** – Data sheets and errata, application notes and sample programs, design resources, user's guides and hardware support documents, latest software releases and archived software
- **General Technical Support** – Frequently Asked Questions (FAQ), technical support requests, online discussion groups, Microchip consultant program member listing
- **Business of Microchip** – Product selector and ordering guides, latest Microchip press releases, listing of seminars and events, listings of Microchip sales offices, distributors and factory representatives

Customer Change Notification Service

Microchip's customer notification service helps keep customers current on Microchip products. Subscribers will receive e-mail notification whenever there are changes, updates, revisions or errata related to a specified product family or development tool of interest.

To register, access the Microchip web site at http://www.microchip.com/. Under "Support", click on "Customer Change Notification" and follow the registration instructions.

Customer Support

Users of Microchip products can receive assistance through several channels:

- Distributor or Representative
- Local Sales Office
- Field Application Engineer (FAE)
- Technical Support

Customers should contact their distributor, representative or Field Application Engineer (FAE) for support. Local sales offices are also available to help customers. A listing of sales offices and locations is included in the back of this document.

Technical support is available through the web site at: http://www.microchip.com/support

Microchip Devices Code Protection Feature

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.

Legal Notice

Trademarks

ISBN: 978-1-5224-2666-0

Quality Management System Certified by DNV

ISO/TS 16949

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.

Worldwide Sales and Service

AMERICAS	ASIA/PACIFIC	ASIA/PACIFIC	EUROPE
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	**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	**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	**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