

## **Getting Started with Analog-to-Digital Converter (ADC)**

## Introduction

Author: Victor Berzan, Microchip Technology Inc.

The Analog-to-Digital Converter (ADC) peripheral converts an analog voltage to a numerical value. This peripheral is included in many AVR® microcontrollers (MCUs). A 10-bit single-ended ADC peripheral is available on most of the tinyAVR® and megaAVR® MCUs, while on the AVR DA family there is a 12-bit differential and single-ended ADC peripheral available.

This technical brief describes how the Analog-to-Digital Converter module works on megaAVR® 0-series and AVR DA microcontrollers. It covers the following use cases:

### • ADC Single Conversion:

Initialize the ADC, start conversion, wait until the conversion is done, and read the ADC result.

#### ADC Free-Running:

Initialize the ADC, enable Free-Running mode, start conversion, wait until the conversion is done, and read the ADC result in an infinite loop.

### ADC Sample Accumulator:

Initialize the ADC, enable accumulation of 64 samples, start conversion, wait until the conversion is done, and read the ADC result in a loop.

#### ADC Window Comparator:

Initialize the ADC, set the conversion window comparator low threshold, enable the conversion Window mode, enable the Free-Running mode, start the conversion, wait until the conversion is done and read the ADC result in an infinite loop, and light-up an LED if the ADC result is below the set threshold.

#### ADC Event Triggered:

Initialize the ADC, initialize the Real-Time Counter (RTC), configure the Event System (EVSYS) to trigger an ADC conversion on RTC overflow, toggle an LED after each ADC conversion.

**Note:** For each of the use cases described in this document, there are two code examples: One bare metal developed on ATmega4809, and one generated with MPLAB® Code Configurator (MCC) developed on AVR128DA48.



View the ATmega4809 Code Examples on GitHub Click to browse repository



View the AVR128DA48 Code Examples on GitHub

© 2021 Microchip Technology Inc.

Technical Brief

DS90003209B-page 1

## **Table of Contents**

| Intr | oduction   | 1      |  |  |  |
|------|--|--------|--|--|--|
| 1.   | Relevant Devices  1.1. tinyAVR® 0-series  1.2. tinyAVR® 1-series  1.3. megaAVR® 0-series  1.4. AVR® DA Family Overview | 5<br>6 |  |  |  |
| 2.   | Overview   | 7      |  |  |  |
| 3.   | ADC Single Conversion  | 8      |  |  |  |
| 4.   | ADC Free-Running   | 12     |  |  |  |
| 5.   | ADC Sample Accumulator   | 13     |  |  |  |
| 6.   | ADC Window Comparator  | 15     |  |  |  |
| 7.   | ADC Event Triggered  |        |  |  |  |
| 8.   | References   | 18     |  |  |  |
| 9.   | Revision History   | 19     |  |  |  |
| 10.  | Appendix   | 20     |  |  |  |
| The  | Microchip Website  | 27     |  |  |  |
| Pro  | duct Change Notification Service   | 27     |  |  |  |
| Cus  | stomer Support   | 27     |  |  |  |
| Mic  | rochip Devices Code Protection Feature   | 27     |  |  |  |
| Leg  | gal Notice   | 28     |  |  |  |
| Tra  | demarks  | 28     |  |  |  |
| Qua  | ality Management System  | 29     |  |  |  |
| Wo   | rldwide Sales and Service  | 30     |  |  |  |

## 1. Relevant Devices

This section lists the relevant devices for this document. The following figures show the different family 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 on tinyAVR<sup>®</sup> 1-series devices 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
- · Devices with different Flash memory sizes typically also have different SRAM and EEPROM

Figure 1-1. tinyAVR® 0-series Overview

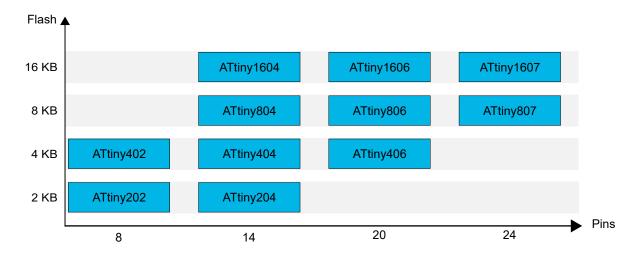


Figure 1-2. tinyAVR® 1-series Overview

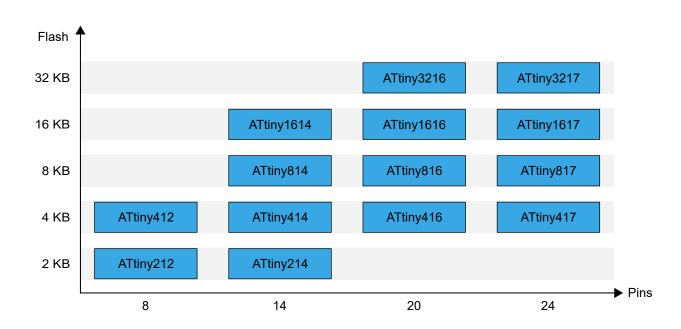


Figure 1-3. megaAVR® 0-series Overview

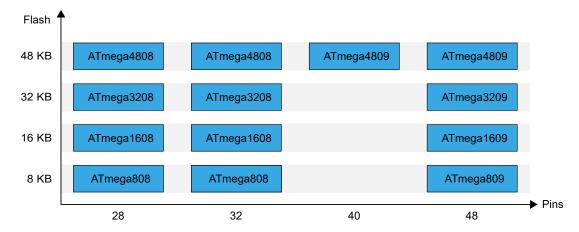


Figure 1-4. AVR® DA Family Overview



## 1.1 tinyAVR® 0-series

The figure below shows the tinyAVR® 0-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
- · Horizontal migration to the left reduces the pin count and, therefore, the available features

Flash A 16 KB ATtiny1604 ATtiny1606 ATtiny1607 8 KB ATtiny804 ATtiny806 ATtiny807 4 KB ATtiny402 ATtiny404 ATtiny406 2 KB ATtiny202 ATtiny204 Pins 20 24 8 14

Figure 1-5. tinyAVR® 0-series Overview

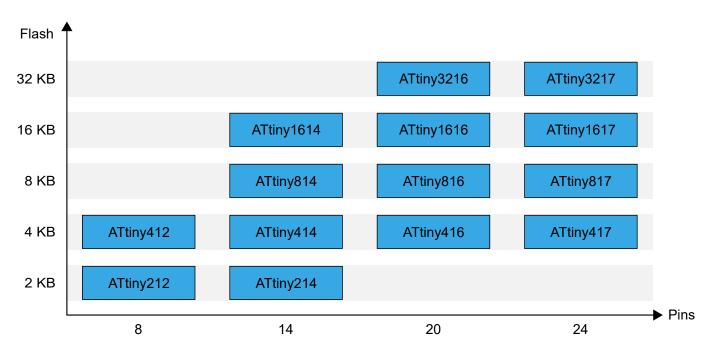
Devices with different Flash memory sizes typically also have different SRAM and EEPROM.

## 1.2 tinyAVR® 1-series

The following figure 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 1-6. tinyAVR® 1-series Overview



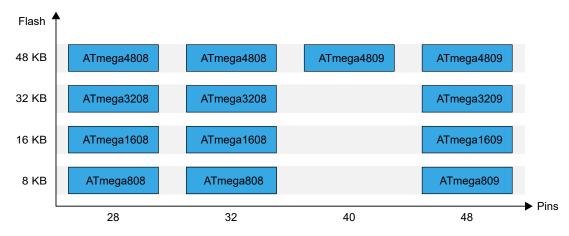
Devices with different Flash memory sizes typically also have different SRAM and EEPROM.

## 1.3 megaAVR® 0-series

The figure below shows the megaAVR® 0-series devices, 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 1-7. megaAVR® 0-series Overview



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

## 1.4 AVR® DA Family Overview

The figure below shows the AVR® DA devices, 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 1-8. AVR® DA Family Overview



Devices with different Flash memory sizes typically also have different SRAM.

## 2. Overview

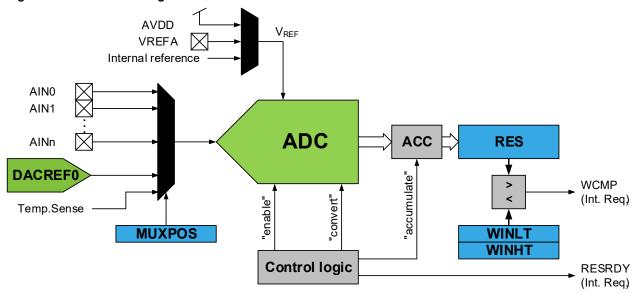
The Analog-to-Digital Converter (ADC) peripheral produces 10-bit results on tinyAVR and megaAVR microcontrollers and 10-bit/12-bit result on AVR DA microcontrollers. The ADC input can either be internal (e.g., a voltage reference), or external through the analog input pins. The ADC is connected to an analog multiplexer, which allows the selection of multiple single-ended voltage inputs. The single-ended voltage inputs refer to 0V (GND).

The ADC supports sampling in bursts where a configurable number of conversion results are accumulated into a single ADC result (Sample Accumulation). The ADC input signal is fed through a sample-and-hold circuit that ensures the input voltage to the ADC is held at a constant level during sampling.

Selectable voltage references from the internal Voltage Reference ( $V_{REF}$ ) peripheral,  $V_{DD}$  supply voltage, or external  $V_{REF}$  pin (VREFA).

A window compare feature is available for monitoring the input signal and can be configured to only trigger an interrupt on user-defined thresholds for under, over, inside, or outside a window, with minimum software intervention required.

Figure 2-1. ADC Block Diagram



The analog input channel is selected by writing to the MUXPOS bits in the MUXPOS (ADCn.MUXPOS) register. Any of the ADC input pins, GND, internal Voltage Reference ( $V_{REF}$ ), or temperature sensor, can be selected as single-ended input to the ADC. The ADC is enabled by writing a '1' to the ADC ENABLE bit in the Control A (ADCn.CTRLA) register. Voltage reference and input channel selections will not go into effect before the ADC is enabled. The ADC does not consume power when the ENABLE bit in ADCn.CTRLA is zero.

The ADC generates a 10-bit result on tinyAVR and megaAVR microcontrollers and a 10-bit/12-bit result on AVR DA microcontrollers. It can be read from the Result (ADCn.RES) register. The result is presented right adjusted.

The result for a 10-bit single-ended conversion is given as:

$$RES = \frac{1023 \times V_{IN}}{V_{REF}}$$

where  $V_{IN}$  is the voltage on the selected input pin and  $V_{RFF}$  is the selected voltage reference.

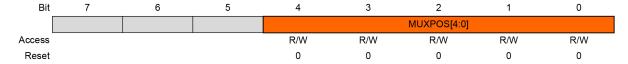
The result for a 12-bit single-ended conversion is given as:

$$RES = \frac{4096 \times V_{IN}}{V_{REF}}$$

## 3. ADC Single Conversion

The simplest mode of using the ADC is to make a single conversion. The ADC input pin needs to have the digital input buffer and the pull-up resistor disabled, to have the highest possible input impedance. Pin PD6/AIN6 is used for ADC input in this example.

Figure 3-1. ADC0.MUXPOS Selection



#### bits 4:0 MUXPOS[4:0]: MUXPOS bits

This bit field selects which single-ended analog input is connected to the ADC. If these bits are changed during a conversion, the change will not take effect until this conversion is complete.

| MUXPOS    | Name       | Input                |
|-----------|------------|----------------------|
| 0x00-0x0F | AIN0-AIN15 | ADC input pin 0 - 15 |
| 0x10-0x1B | -          | Reserved             |
| 0x1C      | DACREF0    | DAC reference in AC0 |
| 0x1D      | -          | Reserved             |
| 0x1E      | TEMPSENSE  | Temperature Sensor   |
| 0x1F      | GND        | GND                  |
| Other     | -          | Reserved             |

ADC0.MUXPOS = ADC\_MUXPOS\_AIN6\_gc;

The ADC Clock Prescaler can be used to divide the clock frequency. In this particular example, the clock is divided by 4. The ADC can use  $V_{DD}$ , external reference, or internal reference for its positive reference. The internal reference is used in this example.

#### Figure 3-2. ADC0.CTRLC Voltage Reference Selection

| Bit    | 7 | 6       | 5    | 4       | 3 | 2   | 1          | 0   |
|--------|---|---------|------|---------|---|-----|------------|-----|
|        |   | SAMPCAP | REFS | EL[1:0] |   |     | PRESC[2:0] |     |
| Access | R | R/W     | R/W  | R/W     | R | R/W | R/W        | R/W |
| Reset  | 0 | 0       | 0    | 0       | 0 | 0   | 0          | 0   |

## bits 5:4 REFSEL[1:0]: Reference Selection bits

These bits select the voltage reference for the ADC.

| Value | Name     | Description                          |
|-------|----------|--------------------------------------|
| 0x0   | INTERNAL | Internal reference                   |
| 0x1   | VDD      | $V_{DD}$                             |
| 0x2   | VREFA    | External reference V <sub>REFA</sub> |
| Other | -        | Reserved.                            |

## bits 2:0 PRESC[2:0]: Prescaler bits

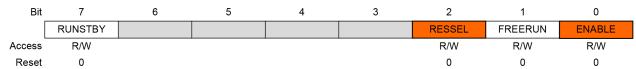
These bits define the division factor from the Peripheral Clock (CLK\_PER) to the ADC Clock (CLK\_ADC).

| Value | Name   | Description            |
|-------|--------|------------------------|
| 0x0   | DIV2   | CLK_PER divided by 2   |
| 0x1   | DIV4   | CLK_PER divided by 4   |
| 0x2   | DIV8   | CLK_PER divided by 8   |
| 0x3   | DIV16  | CLK_PER divided by 16  |
| 0x4   | DIV32  | CLK_PER divided by 32  |
| 0x5   | DIV64  | CLK_PER divided by 64  |
| 0x6   | DIV128 | CLK_PER divided by 128 |
| 0x7   | DIV256 | CLK_PER divided by 256 |

ADCO.CTRLC |= ADC\_PRESC\_DIV4\_gc; ADCO.CTRLC |= ADC\_REFSEL\_INTREF\_gc;

The ADC resolution is set by the RESSEL bit in the ADC0.CTRLA register. The ADC is enabled by setting the ENABLE bit in the ADC0.CTRLA register.

#### Figure 3-3. ADC0.CTRLA Resolution Selection



# **bit 2 RESSEL:** Resolution Selection bit This bit selects the ADC resolution.

| Value | Description   |
|-------|---|
| 0     | Full 10-bit resolution. The 10-bit ADC results are accumulated or stored in the ADC Result register (ADC.RES).  |
| 1     | 8-bit resolution. The conversion results are truncated to eight bits (MSbs) before they are accumulated or stored in the ADC Result register (ADC.RES). The two Least Significant bits are discarded. |

#### bit 0 ENABLE: ADC Enable bit



The ADC conversion is started by setting the STCONV bit in the ADC0.COMMAND register.

## Figure 3-4. ADC0.COMMAND - Start Conversion



#### bit 0 STCONV: Start Conversion bit

Writing a '1' to this bit will start a single measurement. If in Free Running mode this will start the first conversion. STCONV will read as '1' as long as a conversion is in progress. When the conversion is complete, this bit is automatically cleared.

```
ADCO.COMMAND = ADC_STCONV_bm;
```

When the conversion is done, the RESRDY bit in the ADC0.INTFLAGS gets set by the hardware. The user must wait for that bit to get set before reading the ADC result.

#### Figure 3-5. ADC0.INTFLAGS - Hardware-Set RESRDY Bit



#### bit 0 RESRDY: Result Ready Interrupt Flag bit

The Result Ready Interrupt flag is set when a measurement is complete and a new result is ready. The flag is cleared by either writing a '1' to the bit location or by reading the Result register (ADCn.RES). Writing a '0' to this bit has no effect.

```
while (!(ADCO.INTFLAGS & ADC_RESRDY_bm))
{
   ;
}
```

The user must clear the RESRDY bit by writing '1' to it before starting another conversion.

```
ADC0.INTFLAGS = ADC_RESRDY_bm;
```

The conversion result can be read from the ADC0.RES register.

```
adcVal = ADC0.RES;
```



**Tip:** The full code example is also available in the Appendix section.



## View the ATmega4809 Code Example on GitHub

Click to browse repository

An MCC generated code example for AVR128DA48, with the same functionality as the one described in this section, can be found here:



View the AVR128DA48 Code Example on GitHub

Click to browse repository

## 4. ADC Free-Running

When configuring the ADC in Free-Running mode, the next conversion starts immediately after the previous one completes. To activate this mode, the FREERUN bit in the ADC0.CTRLA must be set in addition to the normal ADC initialization.

Figure 4-1. ADC0.CTRLA - Set the FREERUN Bit



#### bit 1 FREERUN: Free-Running bit

Writing a '1' to this bit will enable the Free Running mode for the data acquisition. The first conversion is started by writing the STCONV bit in ADC.COMMAND high. In the Free Running mode, a new conversion cycle is started immediately after or as soon as the previous conversion cycle has completed. This is signaled by the RESRDY flag in ADCn.INTFLAGS.

```
ADCO.CTRLA |= ADC_FREERUN_bm;
```

The ADC conversion is started by setting the STCONV bit in the ADC0.COMMAND register.

```
ADCO.COMMAND = ADC_STCONV_bm;
```

Then the ADC results can be read in a while loop.

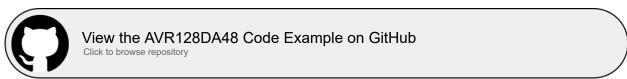
```
while(1)
{
    if (ADC0_conersionDone())
    {
        adcVal = ADC0_read();
    }
}
```



**Tip:** The full code example is also available in the Appendix section.



An MCC generated code example for AVR128DA48, with the same functionality as the one described in this section, can be found here:



## 5. ADC Sample Accumulator

In Sample Accumulator mode the ADC can add up to 64 samples in an accumulator register, thus filtering the signal and reducing the noise. It is useful when reading analog sensor data where a smooth signal is required. By using a hardware accumulator instead of adding those readings in software, it reduces the CPU load. To activate this mode, the Sample Accumulation Number in the ADC0.CTRLB register must be set in addition to the normal ADC initialization.

Figure 5-1. ADC0.CTRLB - Set the SAMPNUM Bit



#### bits 2:0 SAMPNUM[2:0]: Sample Accumulation Number Select bits

These bits select how many consecutive ADC sampling results are accumulated automatically. When this bit is written to a value greater than 0x0, the according number of consecutive ADC sampling results are accumulated into the ADC Result register (ADC.RES) in one complete conversion.

| Value | Name  | Description             |
|-------|-------|-------------------------|
| 0x0   | NONE  | No accumulation.        |
| 0x1   | ACC2  | 2 results accumulated.  |
| 0x2   | ACC4  | 4 results accumulated.  |
| 0x3   | ACC8  | 8 results accumulated.  |
| 0x4   | ACC16 | 16 results accumulated. |
| 0x5   | ACC32 | 32 results accumulated. |
| 0x6   | ACC64 | 64 results accumulated. |
| 0x7   | -     | Reserved.               |

```
ADC0.CTRLB = ADC SAMPNUM ACC64 gc;
```

The ADC conversion is started by setting the STCONV bit in the ADC0.COMMAND register.

```
ADC0.COMMAND = ADC STCONV bm;
```

The samples will be added up in the ADC0.RES register. The ADC\_RESRDY flag is set after the number of samples specified in ADC0.CTRLB is acquired.

```
while (!(ADCO.INTFLAGS & ADC_RESRDY_bm))
{
    ;
}
```

The user can read that value and divide it by the number of samples, to get an average value.

```
adcVal = ADC0.RES;
adcVal = adcVal >> ADC_SHIFT_DIV64;
```

The user must clear the RESRDY bit by writing '1' to it before starting another conversion.

```
ADCO.INTFLAGS = ADC_RESRDY_bm;
```



**Tip:** The full code example is also available in the Appendix section.



# View the ATmega4809 Code Example on GitHub Click to browse repository

An MCC generated code example for AVR128DA48, with the same functionality as the one described in this section, can be found here:



View the AVR128DA48 Code Example on GitHub Click to browse repository

## 6. ADC Window Comparator

In Window Comparator mode, the device can detect if the ADC result is below or above a specific threshold value. This is useful when monitoring a signal that is required to be maintained in a specific range, or for signaling low battery/overcharge, etc. The window comparator can be used in both Free-Running mode and Single Conversion mode. In this example, the window comparator is used in Free-Running mode, because a monitored signal requires continuous sampling, and the Free-Running mode reduces the CPU load by not requiring a manual start for each conversion.

For this example, a low threshold is set in the ADC0.WINLT register.

Figure 6-1. ADC-WINLT - Set the Low Threshold

| Bit    | 15  | 14  | 13  | 12   | 11     | 10  | 9   | 8   |
|--------|-----|-----|-----|------|--------|-----|-----|-----|
|        |     |     |     | WINL | [15:8] |     |     |     |
| Access | R/W | R/W | R/W | R/W  | R/W    | R/W | R/W | R/W |
| Reset  | 0   | 0   | 0   | 0    | 0      | 0   | 0   | 0   |
|        |     |     |     |      |        |     |     |     |
| Bit    | 7   | 6   | 5   | 4    | 3      | 2   | 1   | 0   |
|        |     |     |     | WINL | T[7:0] |     |     |     |
| Access | R/W | R/W | R/W | R/W  | R/W    | R/W | R/W | R/W |
| Reset  | 0   | 0   | 0   | 0    | 0      | 0   | 0   | 0   |

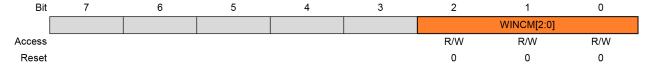
**bits 15:8 WINLT[15:8]:** Window Comparator Low Threshold High Byte These bits hold the MSB of the 16-bit register.

**bits 7:0 WINLT[7:0]:** Window Comparator Low Threshold Low Byte These bits hold the LSB of the 16-bit register.

ADC0.WINLT = WINDOW\_CMP\_LOW\_TH\_EXAMPLE;

The conversion Window mode is set in the ADC0.CTRLE register.

Figure 6-2. ADC0.CTRLE - Set the Window Comparator Mode



bits 2:0 WINCM[2:0]: Window Comparator Mode bits

This field enables and defines when the interrupt flag is set in Window Comparator mode. RESULT is the 16-bit accumulator result. WINLT and WINHT are 16-bit lower threshold value and 16-bit higher threshold value, respectively.

| Value | Name    | Description                      |
|-------|---------|----------------------------------|
| 0x0   | NONE    | No Window Comparison (default)   |
| 0x1   | BELOW   | RESULT < WINLT                   |
| 0x2   | ABOVE   | RESULT > WINHT                   |
| 0x3   | INSIDE  | WINLT < RESULT < WINHT           |
| 0x4   | OUTSIDE | RESULT < WINLT or RESULT >WINHT) |
| Other | -       | Reserved                         |

ADC0.CTRLE = ADC\_WINCM\_BELOW\_gc;

If the ADC result is below the set threshold value, the WCOMP bit in the ADC0.INTFLAGS register is set by the hardware.

## Figure 6-3. ADC0.INTFLAGS - Hardware-Set WCOMP Bit



bit 1 WCOMP: Window Comparator Interrupt Flag bit

This Window Comparator flag is set when the measurement is complete and if the result matches the selected Window Comparator mode defined by WINCM (ADCn.CTRLE). The comparison is done at the end of the conversion. The flag is cleared by either writing a '1' to the bit position or by reading the Result register (ADCn.RES). Writing a '0' to this bit has no effect.

The user must clear that bit by writing '1' to it.

ADC0.INTFLAGS = ADC\_WCMP\_bm;



**Tip:** The full code example is also available in the Appendix section.



## View the ATmega4809 Code Example on GitHub

Click to browse repository

An MCC generated code example for AVR128DA48, with the same functionality as the one described in this section, can be found here:



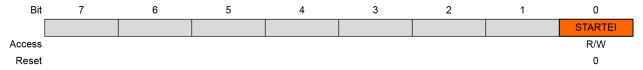
View the AVR128DA48 Code Example on GitHub

Click to browse repository

## 7. ADC Event Triggered

An ADC conversion can be triggered by an event. This is enabled by writing a '1' to the Start Event Input (STARTEI) bit in the Event Control (ADCn.EVCTRL) register.

Figure 7-1. ADC0.EVTRL - Enable the STARTEI Bit



#### bit 0 STARTEI: Start Event Input bit

This bit enables using the event input as trigger for starting a conversion.

```
ADCO.EVCTRL |= ADC_STARTEI_bm;
```

Any incoming event routed to the ADC through the Event System (EVSYS) will trigger an ADC conversion. The event trigger input is edge sensitive. When an event occurs, STCONV in ADCn.COMMAND is set. STCONV will be cleared when the conversion is complete.

For example, to start the ADC conversion on RTC overflow, the following settings must be made:

- 1. The RTC overflow event must be linked to channel 0 of the Event System.
- 2. The Event User ADC0 must be configured to take its input from channel 0.
- The STARTEI bit in the EVCTRL register of the ADC must be set to enable the ADC conversion to be triggered by events.

```
EVSYS.CHANNEL0 = EVSYS_GENERATOR_RTC_OVF_gc; /* Real Time Counter overflow */
EVSYS.USERADC0 = EVSYS_CHANNEL_CHANNEL0_gc; /* Connect user to event channel 0 */
ADC0.EVCTRL |= ADC_STARTEI_bm; /* Enable event triggered conversion */
```



**Tip:** The full code example is also available in the Appendix section.



An MCC generated code example for AVR128DA48, with the same functionality as the one described in this section, can be found here:



## 8. References

- 1. AN2573 ADC Basics with tinyAVR® 0- and 1-series, and megaAVR® 0-series
- 2. AVR128DA48 product page: www.microchip.com/wwwproducts/en/AVR128DA48
- 3. AVR128DA48 Curiosity Nano Evaluation Kit product page: https://www.microchip.com/Developmenttools/ ProductDetails/DM164151
- 4. AVR128DA28/32/48/64 Data Sheet
- 5. Getting Started with the AVR® DA Family
- 6. ATmega4809 product page: www.microchip.com/wwwproducts/en/ATMEGA4809
- 7. megaAVR® 0-series Family Data Sheet
- 8. ATmega809/1609/3209/4809 48-Pin Data Sheet megaAVR® 0-series
- 9. ATmega4809 Xplained Pro product page: https://www.microchip.com/developmenttools/ProductDetails/atmega4809-xpro

# 9. Revision History

| Document Revision | Date    | Comments   |
|-------------------|---------|--|
| В                 | 03/2021 | Updated the GitHub repository links, the <i>References</i> section, and the use cases sections. Added the <i>AVR® DA Family Overview</i> and <i>Revision History</i> sections. Added MCC versions for each use case, running on AVR128DA48. Other minor corrections. |
| Α                 | 05/2019 | Initial document release.  |

## 10. Appendix

#### **Example 10-1. ADC Single Conversion Code Example**

```
/* RTC Period */
#define RTC_PERIOD
                              (511)
#include <avr/io.h>
#include <avr/interrupt.h>
uint16 t adcVal;
void ADC0_init(void);
uint16 t ADC0 read(void);
void ADC0 init(void)
    /* Disable digital input buffer */
    PORTD.PIN6CTRL &= ~PORT_ISC_gm;
    PORTD.PIN6CTRL |= PORT ISC INPUT DISABLE gc;
    /* Disable pull-up resistor */
PORTD.PIN6CTRL &= ~PORT_PULLUPEN_bm;
                                              /* CLK_PER divided by 4 */
/* Internal reference */
    ADC0.CTRLC = ADC_PRESC_DIV4_gc
                | ADC_REFSEL_INTREF_gc;
                                              /* ADC Enable: enabled */
    ADC0.CTRLA = ADC_ENABLE_bm
                | ADC RESSEL 10BIT gc;
                                              /* 10-bit mode */
    /* Select ADC channel */
    ADC0.MUXPOS = ADC_MUXPOS_AIN6_gc;
uint16_t ADC0_read(void)
    /* Start ADC conversion */
    ADC0.COMMAND = ADC STCONV bm;
    /* Wait until ADC conversion done */
    while ( !(ADC0.INTFLAGS & ADC_RESRDY_bm) )
    /* Clear the interrupt flag by writing 1: */
    ADC0.INTFLAGS = ADC_RESRDY_bm;
    return ADC0.RES;
int main(void)
    ADCO_init();
    adcVal = ADC0_read();
    while (1)
```

#### Example 10-2. ADC Free-Running Code Example

```
#include <avr/io.h>
#include <stdbool.h>
uint16 t adcVal;
void ADC0 init(void);
uint16_t ADC0_read(void);
void ADC0 start(void);
bool ADC0_conersionDone(void);
void ADC0 init(void)
    /* Disable digital input buffer */
    PORTD . PIN6CTRL &= ~PORT ISC gm;
    PORTD.PIN6CTRL |= PORT_ISC_INPUT_DISABLE_gc;
    /* Disable pull-up resistor */
PORTD.PIN6CTRL &= ~PORT_PULLUPEN_bm;
    ADCO CTRLC = ADC PRESC DIV4_gc | ADC_REFSEL_INTREF_gc;
                                               /* CLK_PER divided by 4 */
/* Internal reference */
    ADC0.CTRLA = ADC_ENABLE_bm
                                                /* ADC Enable: enabled */
                                                /* 10-bit mode */
                 | ADC RESSEL 10BIT gc;
    /* Select ADC channel */
    ADC0.MUXPOS = ADC MUXPOS AIN6 gc;
    /* Enable FreeRun mode */
    ADCO.CTRLA |= ADC_FREERUN_bm;
uint16_t ADC0_read(void)
    /* Clear the interrupt flag by writing 1: */
ADC0.INTFLAGS = ADC_RESRDY_bm;
    return ADC0.RES;
void ADC0 start(void)
    /* Start conversion */
    ADC0 COMMAND = ADC STCONV bm;
bool ADC0 conersionDone(void)
    return (ADC0.INTFLAGS & ADC_RESRDY_bm);
int main (void)
    ADC0 init();
    ADCO_start();
    while(1)
        if (ADC0_conersionDone())
             adcVal = ADC0 read();
             /* In FreeRun mode, the next conversion starts automatically */
```

#### Example 10-3. ADC Sample Accumulator Code Example

```
#define ADC SHIFT DIV64
                                (6)
#include <avr/io.h>
uint16 t adcVal;
void ADC0_init(void);
uint16_t ADC0_read(void);
void ADC0 init(void)
    /* Disable digital input buffer */
PORTD.PIN6CTRL &= ~PORT_ISC_gm;
    PORTD PIN6CTRL |= PORT ISC INPUT DISABLE gc;
    /* Disable pull-up resistor */
PORTD.PIN6CTRL &= ~PORT_PULLUPEN_bm;
                                                  /* CLK_PER divided by 4 */
/* Internal reference */
    ADC0.CTRLC = ADC PRESC DIV4 gc
                  | ADC_REFSEL_INTREF_gc;
                                                  /* ADC Enable: enabled */
/* 10-bit mode */
    ADC0.CTRLA = ADC_ENABLE_bm
                  | ADC_RESSEL_10BIT_gc;
    /* Select ADC channel */
ADC0.MUXPOS = ADC_MUXPOS_AIN6_gc;
     /* Set the accumulator mode to accumulate 64 samples */
    ADC0.CTRLB = ADC_SAMPNUM_ACC64_gc;
uint16_t ADC0_read(void)
     /* Start ADC conversion */
    ADC0.COMMAND = ADC_STCONV_bm;
     /* Wait until ADC conversion done */
    while ( ! (ADCO.INTFLAGS & ADC RESRDY bm) )
     /* Clear the interrupt flag by writing 1: */
    ADC0 INTFLAGS = ADC_RESRDY_bm;
    return ADC0.RES;
int main (void)
    ADCO_init();
    while (1)
         adcVal = ADC0_read();
         /* divide by 64 */
         adcVal = adcVal >> ADC_SHIFT_DIV64;
```

#### **Example 10-4. ADC Window Comparator Code Example**

```
#define WINDOW CMP LOW TH EXAMPLE
                                        (0x100)
#include <avr/io.h>
#include <stdbool.h>
uint16 t adcVal;
void ADC0_init(void);
uint16_t ADC0_read(void);
void ADC0 start(void);
bool ADC0 conersionDone(void);
bool ADCO_resultBelowTreshold(void);
void ADCO_clearWindowCmpIntFlag(void);
void LED0 init(void);
void LED0_on(void);
void LED0_off(void);
void ADC0 init(void)
    /* Disable digital input buffer */
    PORTD PIN6CTRL &= ~PORT_ISC_gm;
    PORTD.PIN6CTRL |= PORT_ISC_INPUT_DISABLE_gc;
    /* Disable pull-up resistor */
    PORTD PIN6CTRL &= ~PORT PULLUPEN bm;
                                              /* CLK_PER divided by 4 */
/* Internal reference */
    ADC0.CTRLC = ADC_PRESC_DIV4_gc
                 | ADC_REFSEL_INTREF gc;
    ADC0.CTRLA = ADC_ENABLE_bm
                                              /* ADC Enable: enabled */
/* 10-bit mode */
                | ADC_RESSEL_10BIT_gc;
    /* Select ADC channel */
    ADCO MUXPOS = ADC MUXPOS AIN6 gc;
    /* Set conversion window comparator low threshold */
    ADCO.WINLT = WINDOW CMP LOW TH EXAMPLE;
    /* Set conversion window mode */
    ADC0.CTRLE = ADC_WINCM_BELOW_gc;
    /* Enable FreeRun mode */
    ADC0.CTRLA |= ADC_FREERUN bm;
uint16 t ADC0 read(void)
    /* Clear the interrupt flag by writing 1: */
    ADCO . INTFLAGS = ADC_RESRDY_bm;
    return ADC0.RES;
void ADC0 start(void)
    /* Start conversion */
    ADC0.COMMAND = ADC STCONV bm;
bool ADCO conersionDone(void)
    return (ADC0.INTFLAGS & ADC RESRDY bm);
bool ADC0 resultBelowTreshold(void)
    return (ADCO.INTFLAGS & ADC WCMP bm);
void ADC0 clearWindowCmpIntFlag(void)
    /* Clear the interrupt flag by writing 1: */
    ADC0.INTFLAGS = ADC_WCMP_bm;
```

DS90003209B-page 24

```
void LED0 init(void)
     /* Make High (OFF) */
    PORTB.OUT |= PIN5_bm;
     /* Make output */
    PORTB.DIR |= PIN5_bm;
void LED0_on(void)
    /* Make Low (ON) */
PORTB.OUT &= ~PIN5_bm;
void LED0 off(void)
    /* Make High (OFF) */
PORTB.OUT |= PIN5_bm;
int main(void)
    ADCO_init();
LEDO_init();
    ADCO_start();
    while(1)
         if (ADC0_conersionDone())
              if(ADC0_resultBelowTreshold())
                  LED0_on();
ADC0_clearWindowCmpIntFlag();
              else
                   LED0_off();
              adcVal = ADC0_read();
```

#### **Example 10-5. ADC Event Triggered Code Example**

```
/* RTC Period */
#define RTC_PERIOD
                                (511)
#include <avr/io.h>
#include <avr/interrupt.h>
volatile uint16_t adcVal;
void ADC0_init(void);
void LED0_init(void);
void LED0_toggle(void);
void RTC_init(void)
void EVSYS_init(void);
void ADC0 init(void)
    /* Disable digital input buffer */
    PORTD.PIN6CTRL &= ~PORT_ISC_gm;
    PORTD.PIN6CTRL |= PORT_ISC_INPUT_DISABLE_gc;
    /* Disable pull-up resistor */
    PORTD.PIN6CTRL &= ~PORT_PULLUPEN_bm;
    ADCO.CTRLC = ADC PRESC DIV4 gc
                                             /* CLK PER divided by 4 */
                | ADC_REFSEL_INTREF_gc;
                                              /* Internal reference */
                                              /* ADC Enable: enabled */
/* 10-bit mode */
    ADCO.CTRLA = ADC ENABLE bm
                 | ADC RESSEL 10BIT gc;
    /* Select ADC channel *
    ADC0 MUXPOS = ADC_MUXPOS_AIN6_gc;
    /* Enable interrupts */
    ADCO INTCTRL |= ADC RESRDY bm;
    /* Enable event triggered conversion */
    ADC0.EVCTRL |= ADC STARTEI bm;
void LED0_init(void)
    /* Make High (OFF) */
    PORTB.OUT |= PIN5 bm;
    /* Make output */
    PORTB.DIR |= PIN5_bm;
void LED0 toggle(void)
    PORTB.IN |= PIN5 bm;
ISR(ADC0 RESRDY vect)
    /* Clear flag by writing '1': */
    ADC0.INTFLAGS = ADC RESRDY bm;
    adcVal = ADC0.RES;
    LED0_toggle();
void RTC init(void)
    uint8 t temp;
    /* Initialize 32.768kHz Oscillator: */
    /* Disable oscillator: */
    temp = CLKCTRL.XOSC32KCTRLA;
    temp &= ~CLKCTRL_ENABLE_bm;
    /* Enable writing to protected register */
    CPU_CCP = CCP_IOREG_gc;
    CLKCTRL.XOSC32KCTRLA = temp;
    while (CLKCTRL MCLKSTATUS & CLKCTRL XOSC32KS bm)
```

```
; /* Wait until XOSC32KS becomes 0 */
    /* SEL = 0 (Use External Crystal): */
    temp = CLKCTRL.XOSC32KCTRLA;
    temp &= ~CLKCTRL SEL bm;
    /* Enable writing to protected register */
CPU_CCP = CCP_IOREG_gc;
    CLKCTRL.XOSC32KCTRLA = temp;
    /* Enable oscillator: */
    temp = CLKCTRL.XOSC32KCTRLA;
    temp |= CLKCTRL_ENABLE_bm;
    /* Enable writing to protected register */
    CPU CCP = CCP IOREG gc;
    CLKCTRL XOSC32KCTRLA = temp;
    /* Initialize RTC: */
    while (RTC.STATUS > 0)
        ; /* Wait for all register to be synchronized */
                                              /* 32 */
    RTC.CTRLA = RTC_PRESCALER_DIV32_gc
                                                /* Enable: enabled */
                 RTC RTCEN bm
                 RTC_RUNSTDBY_bm;
                                               /* Run In Standby: enabled */
    /* Set period */
    RTC.PER = RTC PERIOD;
    /* 32.768kHz External Crystal Oscillator (XOSC32K) */
RTC.CLKSEL = RTC_CLKSEL_TOSC32K_gc;
    /* Run in debug: enabled */
    RTC.DBGCTRL |= RTC DBGRUN bm;
void EVSYS_init(void)
    /* Real Time Counter overflow */
    EVSYS.CHANNEL0 = EVSYS_GENERATOR_RTC_OVF_gc;
/* Connect user to event channel 0 */
    EVSYS.USERADC0 = EVSYS_CHANNEL_CHANNEL0_gc;
int main(void)
    ADCO init();
    LEDO_init();
RTC init();
    EVSYS init();
    /* Enable Global Interrupts */
    sei();
    while (1)
```

## The Microchip Website

Microchip provides online support via our website at <a href="www.microchip.com/">www.microchip.com/</a>. This website is used to make files and information easily available to customers. Some of the content available includes:

- 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 (FAQs), technical support requests, online discussion groups, Microchip design partner 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

## **Product Change Notification Service**

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

To register, go to www.microchip.com/pcn and follow the registration instructions.

## **Customer Support**

Users of Microchip products can receive assistance through several channels:

- · Distributor or Representative
- · Local Sales Office
- Embedded Solutions Engineer (ESE)
- Technical Support

Customers should contact their distributor, representative or ESE for support. Local sales offices are also available to help customers. A listing of sales offices and locations is included in this document.

Technical support is available through the website at: 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 specifications contained in their particular Microchip Data Sheet.
- Microchip believes that its family of products is secure when used in the intended manner and under normal conditions.
- There are dishonest and possibly illegal methods being used in attempts to breach the code protection features
  of the Microchip devices. We believe that these methods require using the Microchip products in a manner
  outside the operating specifications contained in Microchip's Data Sheets. Attempts to breach these code
  protection features, most likely, cannot be accomplished without violating Microchip's intellectual property rights.
- Microchip is willing to work with any customer who is concerned about the integrity of its code.
- 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. 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.

© 2021 Microchip Technology Inc.

Technical Brief

DS90003209B-page 27

## **Legal Notice**

Information contained in this publication is provided for the sole purpose of designing with and using Microchip products. Information regarding device applications and the like is provided only for your convenience and may be superseded by updates. It is your responsibility to ensure that your application meets with your specifications.

THIS INFORMATION IS PROVIDED BY MICROCHIP "AS IS". MICROCHIP MAKES NO REPRESENTATIONS OR WARRANTIES OF ANY KIND WHETHER EXPRESS OR IMPLIED, WRITTEN OR ORAL, STATUTORY OR OTHERWISE, RELATED TO THE INFORMATION INCLUDING BUT NOT LIMITED TO ANY IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY, AND FITNESS FOR A PARTICULAR PURPOSE OR WARRANTIES RELATED TO ITS CONDITION, QUALITY, OR PERFORMANCE.

IN NO EVENT WILL MICROCHIP BE LIABLE FOR ANY INDIRECT, SPECIAL, PUNITIVE, INCIDENTAL OR CONSEQUENTIAL LOSS, DAMAGE, COST OR EXPENSE OF ANY KIND WHATSOEVER RELATED TO THE INFORMATION OR ITS USE, HOWEVER CAUSED, EVEN IF MICROCHIP HAS BEEN ADVISED OF THE POSSIBILITY OR THE DAMAGES ARE FORESEEABLE. TO THE FULLEST EXTENT ALLOWED BY LAW, MICROCHIP'S TOTAL LIABILITY ON ALL CLAIMS IN ANY WAY RELATED TO THE INFORMATION OR ITS USE WILL NOT EXCEED THE AMOUNT OF FEES, IF ANY, THAT YOU HAVE PAID DIRECTLY TO MICROCHIP FOR THE INFORMATION. Use of Microchip devices in life support and/or safety applications is entirely at the buyer's risk, and the buyer agrees to defend, indemnify and hold harmless Microchip from any and all damages, claims, suits, or expenses resulting from such use. No licenses are conveyed, implicitly or otherwise, under any Microchip intellectual property rights unless otherwise stated.

#### **Trademarks**

The Microchip name and logo, the Microchip logo, Adaptec, AnyRate, AVR, AVR logo, AVR Freaks, BesTime, BitCloud, chipKIT, chipKIT logo, CryptoMemory, CryptoRF, dsPIC, FlashFlex, flexPWR, HELDO, IGLOO, JukeBlox, KeeLoq, Kleer, LANCheck, LinkMD, maXStylus, maXTouch, MediaLB, megaAVR, Microsemi, Microsemi logo, MOST, MOST logo, MPLAB, OptoLyzer, PackeTime, PIC, picoPower, PICSTART, PIC32 logo, PolarFire, Prochip Designer, QTouch, SAM-BA, SenGenuity, SpyNIC, SST, SST Logo, SuperFlash, Symmetricom, SyncServer, Tachyon, TimeSource, tinyAVR, UNI/O, Vectron, and XMEGA are registered trademarks of Microchip Technology Incorporated in the U.S.A. and other countries.

AgileSwitch, APT, ClockWorks, The Embedded Control Solutions Company, EtherSynch, FlashTec, Hyper Speed Control, HyperLight Load, IntelliMOS, Libero, motorBench, mTouch, Powermite 3, Precision Edge, ProASIC, ProASIC Plus, ProASIC Plus logo, Quiet-Wire, SmartFusion, SyncWorld, Temux, TimeCesium, TimeHub, TimePictra, TimeProvider, WinPath, and ZL are registered trademarks of Microchip Technology Incorporated in the U.S.A.

Adjacent Key Suppression, AKS, Analog-for-the-Digital Age, Any Capacitor, Anyln, AnyOut, Augmented Switching, BlueSky, BodyCom, CodeGuard, CryptoAuthentication, CryptoAutomotive, CryptoCompanion, CryptoController, dsPICDEM, dsPICDEM.net, Dynamic Average Matching, DAM, ECAN, Espresso T1S, EtherGREEN, IdealBridge, In-Circuit Serial Programming, ICSP, INICnet, Intelligent Paralleling, Inter-Chip Connectivity, JitterBlocker, maxCrypto, maxView, memBrain, Mindi, MiWi, MPASM, MPF, MPLAB Certified logo, MPLIB, MPLINK, MultiTRAK, NetDetach, Omniscient Code Generation, PICDEM, PICDEM.net, PICkit, PICtail, PowerSmart, PureSilicon, QMatrix, REAL ICE, Ripple Blocker, RTAX, RTG4, SAM-ICE, Serial Quad I/O, simpleMAP, SimpliPHY, SmartBuffer, SMART-I.S., storClad, SQI, SuperSwitcher, SuperSwitcher II, Switchtec, SynchroPHY, Total Endurance, TSHARC, USBCheck, VariSense, VectorBlox, VeriPHY, ViewSpan, WiperLock, XpressConnect, and ZENA are trademarks of Microchip Technology Incorporated in the U.S.A. and other countries.

SQTP is a service mark of Microchip Technology Incorporated in the U.S.A.

The Adaptec logo, Frequency on Demand, Silicon Storage Technology, and Symmcom are registered trademarks of Microchip Technology Inc. in other countries.

GestIC is a registered trademark of Microchip Technology Germany II GmbH & Co. KG, a subsidiary of Microchip Technology Inc., in other countries.

All other trademarks mentioned herein are property of their respective companies.

© 2021, Microchip Technology Incorporated, Printed in the U.S.A., All Rights Reserved.

ISBN: 978-1-5224-7819-5

© 2021 Microchip Technology Inc.

Technical Brief

DS90003209B-page 28

# **Quality Management System**

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

© 2021 Microchip Technology Inc. Technical Brief DS90003209B-page 29



# **Worldwide Sales and Service**

| AMERICAS                  | ASIA/PACIFIC          | ASIA/PACIFIC            | EUROPE                |
|---------------------------|-----------------------|-------------------------|-----------------------|
| Corporate Office          | Australia - Sydney    | India - Bangalore       | Austria - Wels        |
| 2355 West Chandler Blvd.  | Tel: 61-2-9868-6733   | Tel: 91-80-3090-4444    | Tel: 43-7242-2244-39  |
| Chandler, AZ 85224-6199   | China - Beijing       | India - New Delhi       | Fax: 43-7242-2244-393 |
| Tel: 480-792-7200         | Tel: 86-10-8569-7000  | Tel: 91-11-4160-8631    | Denmark - Copenhagen  |
| Fax: 480-792-7277         | China - Chengdu       | India - Pune            | Tel: 45-4485-5910     |
| Technical Support:        | Tel: 86-28-8665-5511  | Tel: 91-20-4121-0141    | Fax: 45-4485-2829     |
| www.microchip.com/support | China - Chongqing     | Japan - Osaka           | Finland - Espoo       |
| Web Address:              | Tel: 86-23-8980-9588  | Tel: 81-6-6152-7160     | Tel: 358-9-4520-820   |
| www.microchip.com         | China - Dongguan      | Japan - Tokyo           | France - Paris        |
| Atlanta                   | Tel: 86-769-8702-9880 | Tel: 81-3-6880- 3770    | Tel: 33-1-69-53-63-20 |
| Duluth, GA                | China - Guangzhou     | Korea - Daegu           | Fax: 33-1-69-30-90-79 |
| Tel: 678-957-9614         | Tel: 86-20-8755-8029  | Tel: 82-53-744-4301     | Germany - Garching    |
| Fax: 678-957-1455         | China - Hangzhou      | Korea - Seoul           | Tel: 49-8931-9700     |
| Austin, TX                | Tel: 86-571-8792-8115 | Tel: 82-2-554-7200      | Germany - Haan        |
| Tel: 512-257-3370         | China - Hong Kong SAR | Malaysia - Kuala Lumpur | Tel: 49-2129-3766400  |
| Boston                    | Tel: 852-2943-5100    | Tel: 60-3-7651-7906     | Germany - Heilbronn   |
| Westborough, MA           | China - Nanjing       | Malaysia - Penang       | Tel: 49-7131-72400    |
| Tel: 774-760-0087         | Tel: 86-25-8473-2460  | Tel: 60-4-227-8870      | Germany - Karlsruhe   |
| Fax: 774-760-0088         | China - Qingdao       | Philippines - Manila    | Tel: 49-721-625370    |
| Chicago                   | Tel: 86-532-8502-7355 | Tel: 63-2-634-9065      | Germany - Munich      |
| Itasca, IL                | China - Shanghai      | Singapore               | Tel: 49-89-627-144-0  |
| Tel: 630-285-0071         | Tel: 86-21-3326-8000  | Tel: 65-6334-8870       | Fax: 49-89-627-144-44 |
| Fax: 630-285-0075         | China - Shenyang      | Taiwan - Hsin Chu       | Germany - Rosenheim   |
| Dallas                    | Tel: 86-24-2334-2829  | Tel: 886-3-577-8366     | Tel: 49-8031-354-560  |
| Addison, TX               | China - Shenzhen      | Taiwan - Kaohsiung      | Israel - Ra'anana     |
| Tel: 972-818-7423         | Tel: 86-755-8864-2200 | Tel: 886-7-213-7830     | Tel: 972-9-744-7705   |
| Fax: 972-818-2924         | China - Suzhou        | Taiwan - Taipei         | Italy - Milan         |
| Detroit                   | Tel: 86-186-6233-1526 | Tel: 886-2-2508-8600    | Tel: 39-0331-742611   |
| Novi, MI                  | China - Wuhan         | Thailand - Bangkok      | Fax: 39-0331-466781   |
| Tel: 248-848-4000         | Tel: 86-27-5980-5300  | Tel: 66-2-694-1351      | Italy - Padova        |
| Houston, TX               | China - Xian          | Vietnam - Ho Chi Minh   | Tel: 39-049-7625286   |
| Tel: 281-894-5983         | Tel: 86-29-8833-7252  | Tel: 84-28-5448-2100    | Netherlands - Drunen  |
| Indianapolis              | China - Xiamen        |                         | Tel: 31-416-690399    |
| Noblesville, IN           | Tel: 86-592-2388138   |                         | Fax: 31-416-690340    |
| Tel: 317-773-8323         | China - Zhuhai        |                         | Norway - Trondheim    |
| Fax: 317-773-5453         | Tel: 86-756-3210040   |                         | Tel: 47-72884388      |
| Tel: 317-536-2380         |                       |                         | Poland - Warsaw       |
| Los Angeles               |                       |                         | Tel: 48-22-3325737    |
| Mission Viejo, CA         |                       |                         | Romania - Bucharest   |
| Tel: 949-462-9523         |                       |                         | Tel: 40-21-407-87-50  |
| Fax: 949-462-9608         |                       |                         | Spain - Madrid        |
| Tel: 951-273-7800         |                       |                         | Tel: 34-91-708-08-90  |
| Raleigh, NC               |                       |                         | Fax: 34-91-708-08-91  |
| Tel: 919-844-7510         |                       |                         | Sweden - Gothenberg   |
| New York, NY              |                       |                         | Tel: 46-31-704-60-40  |
| Tel: 631-435-6000         |                       |                         | Sweden - Stockholm    |
| San Jose, CA              |                       |                         | Tel: 46-8-5090-4654   |
| Tel: 408-735-9110         |                       |                         | UK - Wokingham        |
| Tel: 408-436-4270         |                       |                         | Tel: 44-118-921-5800  |
| Canada - Toronto          |                       |                         | Fax: 44-118-921-5820  |
| Tel: 905-695-1980         |                       |                         |                       |
| Fax: 905-695-2078         |                       |                         |                       |