TB3254

How to Use the 12-Bit Differential ADC with PGA in Burst Accumulation Mode

Features

- 12-Bit Resolution
- Sample Accumulation up to 1024 Samples
- Differential and Single-Ended Conversion
  - Up to 15 analog inputs
    - 15 positive and seven negative inputs
- 4 Internal Inputs
  - GND
  - $V_{DD}/10$
  - Temperature Sensor
  - DACREF from Analog Comparator
- Built-in Internal Reference and External Reference Options
- Programmable Gain Amplifier from 1x to 16x
- Free-Running Mode
- Left or Right Adjusted Result
- Optional: Event-Triggered Conversion
- Configurable Window Comparator

Introduction

Authors: Egil Rotevatn, Rupali Honrao, and Amund Aune, Microchip Technology Inc.

This technical brief explains how to use the Burst mode with the 12-bit Analog-to-Digital Converter (ADC) featured in the tinyAVR® 2 family.

The code examples below are given using the Burst mode:

- Interrupt using Window Comparator
- Event-triggered burst conversion
- ADC oversampling to increase resolution
- Burst Accumulation with Scaling to increase ADC resolution

In Burst mode, the ADC samples a configurable number of samples as fast as possible after a single trigger. The conversion results are accumulated into a single ADC result. A maximum of 1024 samples can be accumulated on a single ADC conversion trigger.

The ADC operation modes can be split into three groups:

- Single mode – Single conversion per trigger, with 8- or 12-bit conversion output
- Series Accumulation mode – One conversion per trigger, with accumulation of n samples
- Burst Accumulation mode – A burst with n samples accumulated as fast as possible after a single trigger

Refer to Section 1. Relevant Documents for details on the other ADC modes.

Table of Contents

1. Relevant Documents

The following documents are relevant to this technical brief:

- Datasheet: tinyAVR 2 Data Sheet (.pdf) on Product Pages:
  - www.microchip.com/wwwproducts/en/ATtiny1624
  - www.microchip.com/wwwproducts/en/ATtiny1626
  - www.microchip.com/wwwproducts/en/ATtiny1627
- How to use the 12-Bit Differential ADC with PGA in Single Mode: www.microchip.com/DS90003256
- How to use the 12-Bit Differential ADC with PGA in Series Accumulation Mode: www.microchip.com/DS90003257

2. Configuration

2.1 Burst Mode Configuration

There are two available Burst modes: Burst Accumulation and Burst Accumulation with Scaling. In both modes, the ADC converts a set number of samples after a trigger, and updates the Sample (ADCn.SAMPLE) register with each conversion result. The samples are automatically accumulated and placed in the Result (ADCn.RESULT) register when the last conversion in the burst is finished. In Burst Accumulation mode, the accumulated result is presented as-is, while in Burst Accumulation with Scaling, the result is scaled to ease output processing as shown in Section 5.2.2 Burst Accumulation with Scaling.

The two modes can be selected by writing the MODE bits in the ADCn.COMMAND register. Below are the code examples showing the configuration of the Burst modes.

```
/* Burst Accumulation mode */
ADC0.COMMAND = ADC_MODE_BURST_gc;
/* Burst Accumulation with Scaling mode */
ADC0.COMMAND = ADC_MODE_BURST_SCALING_gc;
```

2.2 References

- External Reference
- Internal Reference
  - 1.024V
  - 2.048V
  - 2.500V
  - 4.096V
  - V_{DD}

The reference voltage for the ADC (V_{REF}) controls the conversion range of the ADC. External reference and five internal references are available.

```
ADC0.CTRLC = ADC_REFSEL_1024MV_gc;  /* Reference selection 1.024V */
```

Except for V_{DD}, the internal reference voltages are generated from an internal band gap reference. V_{DD} must be at least 0.5V higher than the selected band gap reference voltage.

Changing the reference while a conversion is ongoing will corrupt the output. To safely change input or reference when using Free-Running mode, disable Free-Running mode and wait for the conversion to complete before doing any changes. Enable Free-Running mode before starting the next conversion.

```
ADC0.CTRLF &= ~ADC_FREERUN_bm;                /* Disable Free-Running */
while(!(ADC0.INTFLAGS & ADC_SAMPRDY_bm));     /* Wait until conversion done */
ADC0.CTRLC = ADC_REFSEL_VDD_gc;               /* Configure VDD as reference */
ADC0.CTRLF |= ADC_FREERUN_bm;                 /* Enable Free-Running */
```

2.3 Single-Ended and Differential Modes

In Single-Ended mode, the ADC reads the voltage of a single selectable input source, while in Differential mode, the ADC reads the voltage difference between two input sources.

The Differential mode is configured by writing '1' to the DIFF bit as shown below.

```
/* Differential Mode Configuration */
ADC0.COMMAND |= ADC_DIFF_bm;
```

The Single-Ended mode is configured by writing '0' to DIFF bit as shown below.

```
/* Single-Ended Mode Configuration */
ADC0.COMMAND &= ~ADC_DIFF_bm;
```

2.4 Programmable Gain Amplifier

The Programmable Gain Amplifier (PGA) can be used to amplify the input signal to the ADC. The available range is from 1x to 16x gain. The PGA is enabled by writing a '1' to the PGA Enable (PGAEN) bit and configuring the GAIN bit field in the PGA Control (ADCn.PGACTRL) register.

```
ADC0.PGACTRL |= ADC_GAIN_16X_gc | ADC_PGAEN_bm; /* Enable the PGA with 16x gain */
```

Note: PGA Control is one of few AVR registers with a nonzero reset value. This must be taken into account if only configuring parts of the register.

When PGA is enabled, the configuration of the VIA bit fields in the Positive and Negative Multiplexer (ADCn.MUXPOS and ADCn.MUXNEG) registers is required. The VIA bits are shared, so a value written to the VIA bit field in MUXPOS or MUXNEG is updated in both registers. It is, therefore, not possible to have one input using the PGA and the other not using the PGA.

```
ADC0.MUXPOS |= ADC_VIA_gm;  /* Enable VIA */
```

2.5 Interrupts

The ADC features three separate interrupt vectors. When one of the interrupt conditions occurs, an interrupt flag is set, and the CPU is notified and pointed to the corresponding Interrupt Service Routine (ISR). The following table shows the available interrupt vectors for the ADC.

Table 2-1. Available Interrupt Vectors and Sources

Name	Vector Description	Interrupt Flag	Conditions
ERROR	Error interrupt	TRIGOVR	A new conversion is triggered while one is ongoing
		SAMPOVR	A new conversion overwrites an unread sample in ADCn.SAMPLE
		RESOVR	A new conversion or accumulation overwrites an unread result in ADCn.RESULT
SAMPRDY	Sample Ready interrupt	SAMPRDY	The sample is available in ADCn.SAMPLE
		WCMP	As defined by WINSRC and WINCM in ADCn.CTRLD
RESRDY	Result Ready interrupt	RESRDY	The result is available in ADCn.RESULT
		WCMP	As defined by WINSRC and WINCM in ADCn.CTRLD

An interrupt source is enabled or disabled by writing to the corresponding bit in the Interrupt Control (ADCn.INTCTRL) register as shown in the code snippet below.

```
ADC0.INTCTRL = ADC_RESRDY_bm; /* Enable Result Ready interrupt */
```

The interrupt flag is cleared by writing a '1' to the bit position in the Interrupt Flags (ADCn.INTFLAGS) register as shown in the code snippet below.

```
ADC0.INTFLAGS = ADC_RESRDY_bm; /* Clear Result Ready interrupt flag */
```

Interrupt flags SAMPRDY and RESRDY can also be cleared by reading respectively the ADCn.SAMPLE and ADCn.RESULT registers.

2.6 Window Comparator

The ADC can raise the Window Comparator Interrupt (WCMP) flag in the Interrupt Flags (ADCn.INTFLAGS) register and request an interrupt (WCMP) when the output of a conversion or accumulation is above and/or below certain thresholds. The available modes are:

- ABOVE – The value is above a threshold
- BELOW – The value is below a threshold
- INSIDE – The value is inside a window (above the lower threshold and below the upper threshold)
- OUTSIDE – The value is outside a window (below the lower threshold or above the upper threshold)

The thresholds are set by writing to the Window Comparator Low and High Threshold (ADCn.WINLT and ADCn.WINHT) registers. The Window mode to use is selected by the Window Comparator Mode (WINCM) bit field in the Control D (ADCn.CTRLD) register.

The Window Mode Source (WINSRC) bit in the Control D (ADCn.CTRLD) register selects if the comparison is done on the 16 LSb of the Result (ADCn.RESULT) register or the Sample (ADCn.SAMPLE) register. If an interrupt request is enabled for the WCMP flag, WINSRC selects which interrupt vector to request, RESRDY or SAMPRDY.

When accumulating multiple samples, if the Window Comparator source is the Result register, the comparison between the result and the threshold(s) will happen after the last conversion is complete. If the source is the Sample register, the comparison will happen after every conversion.

The following code shows how to configure the thresholds of the window comparator, and how to configure the INSIDE mode comparing against the Result register.

```
ADC0.WINHT = 200;        /* Window High Threshold */
ADC0.WINLT = 100;        /* Window Low Threshold */
ADC0.CTRLD |= ADC_WINCM_INSIDE_gc | ADC_WINSRC_RESULT_gc; /* Result as Window Comparator
source*/
```

2.6.1 Code Example

The code example below shows an application example where an ADC reading of below 2000 or above 3000 is considered an invalid signal spike. The window comparator is used to filter these out by restarting the sample accumulation in the SAMPRDY interrupt when the signal is outside of the thresholds. The voltage of the signal is calculated in the RESRDY Interrupt Service Routine (ISR) when all samples are accumulated.

```
#define F_CPU 3333333ul

#include <avr/io.h>
#include <avr/interrupt.h>
#include <math.h>
#include <util/delay.h>

#define TIMEBASE_VALUE      ((uint8_t) ceil(F_CPU*0.000001))
#define ADC_MAX_VALUE       ((1 << 12) - 1) /* In single-ended mode, the max value is 4095 */

/* Defines to easily configure ADC accumulation */
#define ADC_SAMPNUM_CONFIG  ADC_SAMPNUM_ACC256_gc
/* Left shifting (1 << SAMPNUM) results in the number of accumulated samples */
#define ADC_SAMPLES         (1 << ADC_SAMPNUM_CONFIG)

/* Volatile variables to improve debug experience */
static volatile uint32_t adc_reading;
static volatile float voltage;

/*****************************************************************************
ADC initialization
*****************************************************************************/
void adc_init()
{
    ADC0.CTRLA = ADC_ENABLE_bm;
    ADC0.CTRLB = ADC_PRESC_DIV2_gc; /* fCLK_ADC = 3.333333/2 MHz */
    ADC0.CTRLC = ADC_REFSEL_VDD_gc | (TIMEBASE_VALUE << ADC_TIMEBASE_gp);
    ADC0.CTRLE = 17; /* (SAMPDUR + 0.5) * fCLK_ADC = 10.5 µs sample duration */
    ADC0.CTRLF = ADC_SAMPNUM_CONFIG;
```

```
    ADC0.MUXPOS = ADC_MUXPOS_AIN6_gc; /* ADC channel AIN6 -> PA6 */

    ADC0.WINHT = 3000; /* Window High Threshold */
    ADC0.WINLT = 2000; /* Window Low Threshold */
    /* Window Comparator mode: Outside. Use SAMPLE register as Window Comparator source */
    ADC0.CTRLD = ADC_WINCM_OUTSIDE_gc | ADC_WINSRC_SAMPLE_gc;
    /* Enable Window Compare and Result Ready interrupt */
    ADC0.INTCTRL = ADC_WCMP_bm | ADC_RESRDY_bm;

    ADC0.COMMAND = ADC_MODE_BURST_gc; /* Burst Accumulation mode */
}

/*****************************************************************************
Window Compare interrupt:
In this example, when a sample is outside a certain window, this is considered an
invalid signal spike. When such a spike is detected, the accumulated ADC result is
disregarded by restarting the burst conversion.
*****************************************************************************/
ISR(ADC0_SAMPRDY_vect)
{
    ADC0.INTFLAGS = ADC_WCMP_bm;        /* Clear WCMP flag */

    /* Stop the ongoing burst */
    ADC0.COMMAND = ADC_START_STOP_gc;
    /* Start a new burst accumulation */
    ADC0.COMMAND = ADC_MODE_BURST_gc | ADC_START_IMMEDIATE_gc;
}

/*****************************************************************************
Result Ready interrupt:
If no spike was detected, the result is read and the corresponding
voltage is calculated
*****************************************************************************/
ISR(ADC0_RESRDY_vect)
{
    ADC0.INTFLAGS = ADC_RESRDY_bm;      /* Clear RESRDY flag */

    /* Check if the last sample was inside the window */
    if(!(ADC0.INTFLAGS & ADC_WCMP_bm))
    {
        adc_reading = ADC0.RESULT; /* Read ADC result */
        /* Calculate voltage on ADC pin, VDD = 3.3V, 12-bit resolution, 256 samples */
        voltage = (float)(((adc_reading * 3.3) / ADC_MAX_VALUE) / ADC_SAMPLES);
    }
}

int main(void)
{
    adc_init();
    sei(); /* Enable global interrupts */

    while(1)
    {
        /* Start a conversion once every 100 ms */
        ADC0.COMMAND |= ADC_START_IMMEDIATE_gc;
        _delay_ms(100);
    }
}
```

2.7 Events

The ADC can be connected to the event system. The event system lets peripherals communicate without CPU intervention, enabling the CPU to perform other tasks or stay in a sleep mode. The ADC can be connected either as an event generator, providing signals to another peripheral, or an event user, performing tasks based on the signals from another peripheral.

The following table shows the different available event generators from the ADC.

Table 2-2. ADC Event Generators

Generator Name		Description	Event Type	Generating Clock Domain	Length of Event
Peripheral	Event				
ADCn	RESRDY	Result ready	Pulse	CLK_PER	One CLK_PER period
ADCn	SAMPRDY	Sample ready	Pulse	CLK_PER	One CLK_PER period
ADCn	WCMP	Window compare match	Pulse	CLK_PER	One CLK_PER period

Below is a code snippet showing the configuration of event generator ADC0_RESRDY connected through event channel 1 to the EVOUT event user, which in this case outputs the event to PB2.

- Event Generator: ADC0 RESRDY
- Event USER: EVOUT (PB2)

```
EVSYS.CHANNEL1 = EVSYS_CHANNEL1_ADC0_RES_gc;     /* ADC Result Ready */
EVSYS.USEREVSYSEVOUTB = EVSYS_USER_CHANNEL1_gc; /* Asynchronous Event Channel 1 */
```

The ADC has one event user for detecting and acting upon input events. The table below describes the event user and the associated functionality.

Table 2-3. ADC Event Users and Available Event Actions

User Name		Description	Input Detection	Async/Sync
Peripheral	Event			
ADCn	START	ADC start on event	Edge	Async

The START event action can be triggered if the EVENT_TRIGGER setting is written to the START bit field in the Command (ADCn.COMMAND) register as shown in the code snippet below.

```
ADC0.COMMAND = ADC_START_EVENT_TRIGGER_gc;
```

Below is a code snippet showing the configuration of ADC0_START as an event user, reacting to RTC overflow.

```
EVSYS.CHANNEL0 = EVSYS_CHANNEL0_RTC_OVF_gc;   /* Real Time Counter overflow */
EVSYS.USERADC0START = EVSYS_USER_CHANNEL0_gc; /* Asynchronous Event Channel 0 */
```

2.7.1 Code Example

Below is a code example showing the configuration of the ADC as an event generator and an event user:

- **Event user: The ADC conversion triggered by an RTC overflow event**
  - RTC is configured to generate an RTC overflow event at the desired ADC sampling rate. The sampling rate in the example is 100 Hz.
  - ADC conversion is triggered at a rate of 100 Hz, and the result is read when the Result Ready (RESRDY) bit in the Interrupt Flags (ADCn.INTFLAGS) register is set.
- **Event generator: Pin PB2 outputs an event (Pulse) when the ADC result is ready**

```
#define F_CPU 3333333ul

#include <avr/io.h>
#include <math.h>

#define TIMEBASE_VALUE      ((uint8_t) ceil(F_CPU*0.000001))
#define ADC_MAX_VALUE       (((1 << 12) / 2) - 1) /* In differential mode, the max value is
2047 */

/* Defines to easily configure ADC accumulation */
```

```
#define ADC_SAMPNUM_CONFIG  ADC_SAMPNUM_ACC8_gc
/* Left shifting (1 << SAMPNUM) results in the number of accumulated samples */
#define ADC_SAMPLES         (1 << ADC_SAMPNUM_CONFIG)

/* Defines to easily configure RTC event frequency */
#define ADC_SAMPLING_FREQ   100     /* Hz */
#define RTC_CLOCK           32768   /* Hz */
#define RTC_PERIOD          (RTC_CLOCK / ADC_SAMPLING_FREQ)

/* Volatile variables to improve debug experience */
static volatile int32_t adc_reading;
static volatile float voltage;

/*****************************************************************************
EVSYS initialization:
Channel 0:
            Event system generator: RTC Overflow
            Event system user: ADC0
Channel 1:
            Event system generator: ADC0 Result Ready
            Event system user: EVOUTB (PIN PB2)
*****************************************************************************/
void event_system_init(void)
{
    PORTB.DIRSET = PIN2_bm; /* Configure EVOUTB to output */

    EVSYS.CHANNEL0 = EVSYS_CHANNEL0_RTC_OVF_gc;     /* RTC Overflow ->  Channel 0 */
    EVSYS.USERADC0START = EVSYS_USER_CHANNEL0_gc;   /* Channel 0    ->  ADC0 Start */

    EVSYS.CHANNEL1 = EVSYS_CHANNEL1_ADC0_RES_gc;    /* ADC RESRDY   ->  Channel 1 */
    EVSYS.USEREVSYSEVOUTB = EVSYS_USER_CHANNEL1_gc; /* Channel 1    ->  EVOUTB (PB2) */
}

/*****************************************************************************
RTC initialization
*****************************************************************************/
void rtc_init(void)
{
    while(RTC.STATUS > 0);  /* Wait for all registers to be synchronized */
    RTC.CTRLA = RTC_PRESCALER_DIV1_gc | RTC_RTCEN_bm; /* Enable RTC, no prescaler */
    RTC.CLKSEL = RTC_CLKSEL_INT32K_gc; /* Select 32.768 kHz internal RC oscillator */
    RTC.PER = RTC_PERIOD;
    while(RTC.STATUS > 0);  /* Wait for all registers to be synchronized */
}

/*****************************************************************************
ADC initialization
*****************************************************************************/
void adc_init()
{
    ADC0.CTRLA = ADC_ENABLE_bm;
    ADC0.CTRLB = ADC_PRESC_DIV2_gc; /* fCLK_ADC = 3.333333/2 MHz */
    ADC0.CTRLC = ADC_REFSEL_VDD_gc | (TIMEBASE_VALUE << ADC_TIMEBASE_gp);
    ADC0.CTRLE = 17; /* (SAMPDUR + 0.5) * fCLK_ADC = 10.5 µs sample duration */
    ADC0.CTRLF = ADC_SAMPNUM_CONFIG;

    ADC0.MUXPOS = ADC_MUXPOS_AIN6_gc; /* ADC channel AIN6 -> PA6 */
    ADC0.MUXNEG = ADC_MUXNEG_AIN7_gc; /* ADC channel AIN7 -> PA7 */
    /* Start ADC Burst conversion on event trigger */
    ADC0.COMMAND = ADC_DIFF_bm | ADC_MODE_BURST_gc | ADC_START_EVENT_TRIGGER_gc;
}

int main(void)
{
    event_system_init();
    rtc_init();
    adc_init();

    while(1)
    {
        if(ADC0.INTFLAGS & ADC_RESRDY_bm)  /* Check if ADC sample is ready */
        {
            /* Read accumulated ADC result, clears the interrupt flag */
            adc_reading = ADC0.RESULT;
            /* Calculate voltage on ADC pin, VDD = 3.3V, 8 samples in 12-bit resolution */
            voltage = (float)(((adc_reading * 3.3) / ADC_MAX_VALUE) / ADC_SAMPLES);
```

```
            }
        }
}
```

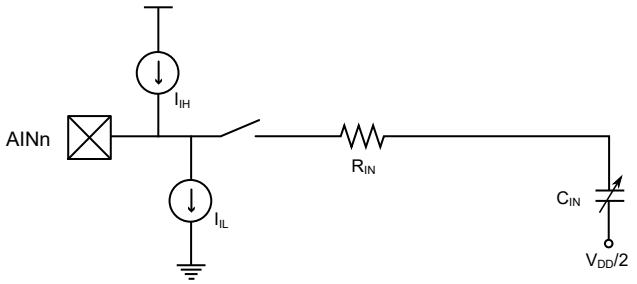
Tip: When using V_{DD} as reference and taking an average of accumulated samples, the ADC can effectively be used as a power supply noise filter.

3. Input Circuitry

3.1 Input Impedance

When a voltage level imposed on a pin is sampled, it is first captured by the Sample-and-Hold capacitor (C_{IN}). This ensures that the voltage does not change while the ADC samples the signal.

Figure 3-1. Model of Internal Analog Input Circuit



The time it takes to charge or discharge C_{IN} to a certain voltage level is limited by the input resistance (R_{IN}). The following equation shows the proportional relation between the time constant τ and the input impedance.

$$\tau = R_{IN} \times C_{IN}$$

Refer to the *Electrical Characteristics* section in the data sheet for details on the input characteristics of the ADC.

The 12-bit resolution of the ADC (and optional gain) requires the impulse response of the input circuit settled to more than 99.9% of the final voltage to be certain the measurement will be correct. The following example calculations without gain and with 16x gain show how settled a signal needs to be for the ADC to sample correctly at 12-bit resolution.

$$V_{\mathrm{MSb}} = V_{\mathrm{REF}} - \frac{V_{\mathrm{REF}}}{4096 \times \mathrm{Gain}}$$

$$V_{\mathrm{MSb}}\,\% = \left(1 - \frac{1}{4096 \times \mathrm{Gain}}\right) \times 100\,\%$$

$$V_{\mathrm{MSb}}\,\%_{\mathrm{without\ gain}} = \left(1 - \frac{1}{4096 \times 1}\right) \times 100\,\% = 99.975\,\%$$

$$V_{\mathrm{MSb}}\,\%_{\mathrm{16x\ gain}} = \left(1 - \frac{1}{4096 \times 16}\right) \times 100\,\% = 99.998\,\%$$

The impulse response for the input circuit is given by the following equation.

$$V(t) = V_{\mathrm{IN}} \times \left(1 - e^{-t/\tau}\right)$$

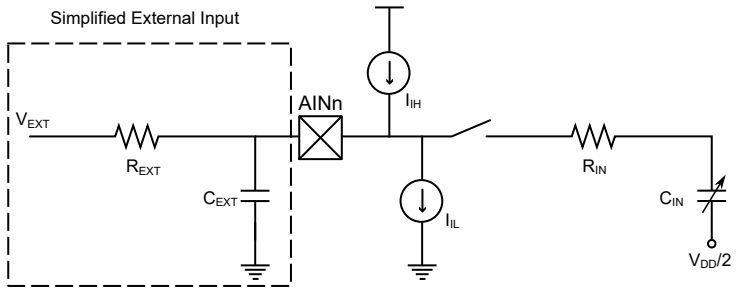
Solving the two examples for V_{MSb} where V_{IN} is 100%, the following settling times are obtained.

$$t_{\mathrm{without\ gain}} = 8.29\tau$$

$$t_{\mathrm{16x\ gain}} = 10.81\tau$$

The impedance of the external signal should also be taken into consideration when calculating the settling time, expanding the circuit into a more complex system as shown in the figure below.

Figure 3-2. Model of Analog Input Circuit with External Signal



The characteristics of the external impedance determine how complex the settling time calculation will be. However, this is not covered by this technical brief.

3.1.1 PGA

The PGA is connected between the analog input pin and the ADC, with an input impedance depending on the selected gain setting. Refer to the *Electrical Characteristics* section for details on the input characteristics of the PGA. The equations above are the same for calculating the appropriate sample duration when using the PGA impedance values.

When the PGA is used, it is continuously sampling and will only be in the Hold state when the ADC is sampling the PGA. If the time between conversions is longer than the needed sampling time, this can be utilized to get a shorter total conversion time by setting the SAMPDUR to the minimum supported value.

3.2 Sample Duration

A suitable ADC sample duration can either be calculated based on the impulse response of the circuit, as shown in Section 3.1 Input Impedance, or found by tuning the sample duration in firmware until a stable output from the ADC conversion is achieved.

The sample duration for this ADC can be a maximum of 256 ADC clock (CLK_ADC) cycles, and is configured using the Sample Duration (SAMPDUR) bit field in the Control E (ADCn.CTRLE) register. The sample duration is SAMPDUR + 0.5 (CLK_ADC) cycles when the PGA is disabled, and SAMPDUR + 1 (CLK_ADC) when the PGA is enabled. If the input impedance is very high, increasing the ADC prescaler can also be used to further increase the sample duration.

Minimum sample duration is configured as shown in the following code snippet. The calculations are based on the CPU clock running at 16 MHz, with PGA disabled.

```
ADC0.CTRLB = ADC_PRESC_DIV2_gc; /* ADC clock: 8 MHz */
ADC0.CTRLE = 0; /* Sample Duration: (0 + 0.5) / 8 MHz = 0.06 µS */
```

Maximum sample duration is configured as shown in the following code snippet. The calculations are based on the CPU clock running at 16 MHz, with PGA disabled.

```
ADC0.CTRLB = ADC_PRESC_DIV40_gc; /* ADC clock: 400 kHz */
ADC0.CTRLE = 255; /* Sample Duration: (255 + 0.5) / 400 kHz = 639 µS */
```

4. Power and Timing

4.1 Clock

The ADC clock (CLK_ADC) is scaled down from the peripheral clock (CLK_PER). This can be configured by the Prescaler (PRESC) bit field in the CTRLB (ADCn.CTRLB) register.

```
ADC0.CTRLB = ADC_PRESC_DIV20_gc; /* CLK_ADC = CLK_PER/20 */
```

Some of the internal timings in the ADC and the PGA are independent of CLK_ADC. To ensure correct internal timing regardless of the ADC clock frequency, a 1 µs timebase (given in CLK_PER cycles) must be written to the TIMEBASE bit field in the Control C (ADCn.CTRLC) register. The timebase must be rounded up to the closest integer. The following code snippet shows how this can be done using the `ceil` function.

```
#include <math.h>
#define F_CPU 3333333ul
#define TIMEBASE_VALUE ((uint8_t) ceil(F_CPU*0.000001))

ADC0.CTRLC = (TIMEBASE_VALUE << ADC_TIMEBASE_gp);
```

4.2 PGA Bias and Output Sample Duration

The PGA Bias Select (PGABIASSEL) bit field in the ADC PGA Control (ADCn.PGACTRL) register can be configured to reduce the power consumption depending on the ADC clock frequency. The ADC PGA Sample Duration (ADCPGASAMPDUR) bit field can be configured to reduce the number of CLK_ADC cycles the ADC is sampling the output of the PGA. This is also dependent of the ADC clock frequency.

See the register description for these bit fields in the data sheet for recommended combinations of f_{CLK_ADC} and PGABIASSEL and ADCPGASAMPDUR.

An example configuration is shown below.

```
ADC0.PGACTRL = ADC_GAIN_16X_gc | /* 16x gain */
               ADC_PGABIASSEL_100PCT_gc | /* 100% bias current */
               ADC_ADCPGASAMPDUR_32CLK_gc | /* 32 cycles sampling of the PGA */
               ADC_PGAEN_bm; /* Enable the PGA */
```

Note: PGA Control is one of few AVR registers with a nonzero reset value. This must be taken into account if only configuring parts of the register.

4.3 Conversion Time

The total conversion time for a single result is calculated by:

$$\text{Total Conversion Time} = \text{Initialization} + \frac{(\text{SAMPDUR} + 13.5) \times \text{SAMPNUM} + 2}{f_{\text{CLK\_ADC}}}$$

For example, given initialization = 60 µs, SAMPDUR = 2, SAMPNUM = 32 and f_{CLk_ADC} = 1 MHz, the total conversion time is given by:

$$\text{Total Conversion Time} = 60\ \mu\text{s} + \frac{(2 + 13.5) \times 32 + 2}{f_{\text{CLK\_ADC}}} = 558\ \mu\text{s}$$

With the Low Latency (LOWLAT) bit written to '1' in the Control A (ADCn.CTRLE) register, the initialization time is only needed once upon enabling the ADC. After that, the example above will give a total conversion time of 498 µs.

The sampling period of the ADC is configured through the Sample Duration (SAMPDUR) bit field in the Control E (ADCn.CTRLE) register as (SAMPDUR + ½) CLK_ADC cycles.

```
ADC0.CTRLE = 2; /* Sample duration configured to 2 */
```

If PGA is used, the input sample duration is (SAMPDUR + 1) CLK_ADC cycles, while the ADC PGA Sample Duration (ADCPGASAMPDUR) bit field in the PGA Control (ADCn.PGACTRL) register controls how long the ADC samples the PGA.

```
ADC0.PGACTRL = ADC_ADCPGASAMPDUR_15CLK_gc; /* 15 CLK_ADC cycles */
```

4.4 Free-Running Mode

In Free-Running mode, a new conversion is started as soon as the previous accumulation has completed.

It is configured by writing the Free-Running (FREERUN) bit to '1' in the Control F (ADCn.CTRLF) register as shown in the code snippet below.

```
ADC0.CTRLF = ADC_FREERUN_bm; /* ADC Free-Running mode enabled */
```

A new conversion can be started immediately after a result is available in the Result (ADCn.RESULT) register. This is signaled by RESRDY in the Interrupt Flags (ADCn.INTFLAGS) register.

5. Output Processing

5.1 Result Range

The output from an ADC conversion is given by the following equations:

$$\text{Single-Ended conversion} = \frac{V_{\text{INP}} \times \text{Gain}}{V_{\text{REF}}} \times 4096 \in [0, 4095]$$

$$\text{Differential conversion} = \frac{(V_{\text{INP}} - V_{\text{INN}}) \times \text{Gain}}{V_{\text{REF}}} \times 2048 \in [-2048, 2047]$$

V_{INP} and V_{INN} are the positive and negative inputs to the ADC, and V_{REF} is the selected voltage reference. The gain is between 1x and 16x as configured in the PGA, and 1x if the PGA is not in use.

The ADC has two output registers, the Sample (ADCn.SAMPLE) and Result (ADCn.RESULT) registers. The 16-bit Sample register will always be updated with the latest ADC conversion output (one sample). In Burst Accumulation mode, after the SAMPNUM number of conversions are finished, the Result register will be updated with the accumulated result.

With a Single-Ended conversion, the average voltage applied to the analog pin is calculated by:

$$V_{\text{INP}} = \frac{\text{ADCn.RESULT} \times V_{\text{REF}}}{4096 \times \text{Gain} \times \text{SAMPNUM}}$$

5.2 Accumulation

The ADC supports sampling in bursts where a configurable number of conversion results are automatically accumulated into a single ADC result. A maximum of 1024 samples can be accumulated. Sample accumulation is configured by writing the Sample Accumulation Number Select (SAMPNUM) bit field in the Control F (ADCn.CTRLF) register.

The samples are accumulated in an internal sample accumulator which is sufficiently wide to avoid overflow for all supported accumulation configurations. The accumulated result is transferred to the 32-bit Result (ADCn.RESULT) register when the selected number of samples are finished, and the Result Ready (RESRDY) bit in the Interrupt Flags (ADCn.INTFLAGS) register is set.

The code snippet below shows the configuration to accumulate 1024 samples.

```
ADC0.CTRLF = ADC_SAMPNUM_ACC1024_gc;
```

5.2.1 Oversampling

The Burst Accumulation mode can be used to achieve higher ADC resolution. For example, by using 1024 12-bit samples, a 17-bit result could be achieved.

To increase the resolution, for each additional bit of ADC resolution (*n*), the signal must be oversampled 4^n samples. To achieve 17-bit ADC, an additional 5-bit resolution is needed. Hence, the signal has to be sampled $4^5 = 1024$ times. To scale the result down to the desired resolution, it must be right-shifted by *n* which is equivalent to dividing the result by 2^n.

5.2.1.1 Code Example

The following code example shows how to increase the ADC resolution from 12-bit to 17-bit by using oversampling, and how to calculate the voltage applied to ADC channel AIN6.

```
#define F_CPU 3333333ul

#include <avr/io.h>
#include <math.h>
#include <util/delay.h>
```

```
#define TIMEBASE_VALUE          ((uint8_t) ceil(F_CPU*0.000001))

/* Defines to configure ADC accumulation */
#define OVERSAMPLING_BITS       5 /* 5 bits extra */
#define OVERSAMPLING_MAX_VALUE  ((uint32_t) ((1 << 12) - 1) << OVERSAMPLING_BITS) /* 12 + 5
bits = 17 bits */
#define ADC_SAMPNUM_CONFIG      (OVERSAMPLING_BITS << 1) /* The SAMPNUM bit field setting
match this formula */
#define ADC_SAMPLES             (1 << ADC_SAMPNUM_CONFIG) /* 5 bits = 1024 samples */

/* Volatile variables to improve debug experience */
static volatile uint32_t adc_reading;
static volatile float voltage;

/*****************************************************************************
ADC initialization
*****************************************************************************/
void adc_init()
{
    ADC0.CTRLA = ADC_ENABLE_bm;
    ADC0.CTRLB = ADC_PRESC_DIV2_gc; /* fCLK_ADC = 3.333333/2 MHz */
    ADC0.CTRLC = ADC_REFSEL_VDD_gc | (TIMEBASE_VALUE << ADC_TIMEBASE_gp);
    ADC0.CTRLE = 17; /* (SAMPDUR + 0.5) * fCLK_ADC = 10.5 µs sample duration */
    ADC0.CTRLF = ADC_SAMPNUM_CONFIG;

    ADC0.MUXPOS = ADC_MUXPOS_AIN6_gc; /* ADC channel AIN6 -> PA6 */
    ADC0.COMMAND = ADC_MODE_BURST_gc; /* Burst Accumulation mode */
}

int main(void)
{
    adc_init();

    while(1)
    {
        ADC0.COMMAND |= ADC_START_IMMEDIATE_gc;
        while(!(ADC0.INTFLAGS & ADC_RESRDY_bm)); /* Wait until conversion is done */

        /* Oversampling compensation as explained in the tech brief */
        adc_reading = ADC0.RESULT >> OVERSAMPLING_BITS; /* Scale accumulated result by right
shifting the number of extra bits */
        voltage = (float)((adc_reading * 3.3) / OVERSAMPLING_MAX_VALUE); /* Calculate voltage
using 17-bit resolution, VDD = 3.3V */
    }
}
```

5.2.2 Burst Accumulation with Scaling

Like in the Burst Accumulation mode, a SAMPNUM number of samples in this mode are automatically accumulated into a single result. The difference is that in Burst Accumulation with Scaling, the result is automatically scaled to a 16-bit value. Scaling is always applied after accumulating the last sample and is carried out by right-shifting the accumulated result by SAMPNUM-4 bits.

This mode can simply be used to accumulate and automatically get an averaged result. It may also be used in combination with the oversampling technique as described in Section 5.2.1 Oversampling to achieve a 16-bit resolution by accumulating at least 4^4 = 256 samples and automatically scaling down to 16 bits.

5.2.2.1 Code Example

The following code example shows how to perform differential measurements using the Burst Accumulation with Scaling mode, the PGA with 16x gain and oversampling to achieve a 16-bit resolution.

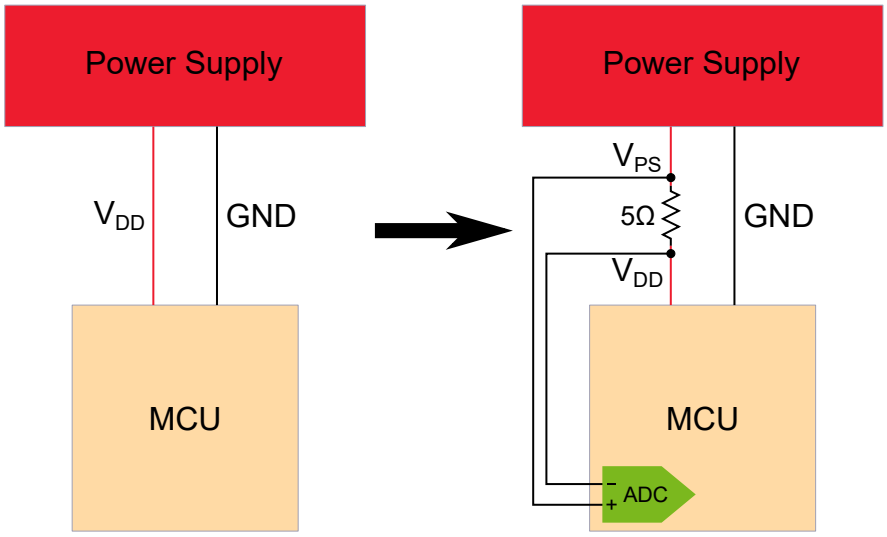
```
#define F_CPU 3333333ul

#include <avr/io.h>
#include <math.h>
#include <util/delay.h>

#define TIMEBASE_VALUE              ((uint8_t) ceil(F_CPU*0.000001))
#define ADC_DIFF_MAX_VALUE          (((1 << 12) / 2) - 1)                   /* In
differential mode, the max value is 2047 */
#define ADC_DIFF_MAX_VALUE_16BIT    ((uint32_t) ADC_DIFF_MAX_VALUE << 4)    /* In
differential mode, the max value for a 16-bit result is 32752 */
```

```
/* Defines to easily configure ADC accumulation */
#define ADC_SAMPNUM_CONFIG  ADC_SAMPNUM_ACC256_gc
/* Left shifting (1 << SAMPNUM) results in the number of accumulated samples */
#define ADC_SAMPLES         (1 << ADC_SAMPNUM_CONFIG)

/* Volatile variables to improve debug experience */
static volatile int32_t adc_reading;
static volatile float voltage;
static volatile float current;

/*****************************************************************************
ADC initialization
*****************************************************************************/
void adc_init()
{
    ADC0.CTRLA = ADC_ENABLE_bm;
    ADC0.CTRLB = ADC_PRESC_DIV2_gc; /* fCLK_ADC = 3.333333/2 MHz */
    ADC0.CTRLC = ADC_REFSEL_1024MV_gc | (TIMEBASE_VALUE << ADC_TIMEBASE_gp);
    ADC0.CTRLE = 17; /* (SAMPDUR + 0.5) * fCLK_ADC = 10.5 µs sample duration */
    ADC0.CTRLF = ADC_LEFTADJ_bm | ADC_SAMPNUM_CONFIG; /* Enable left adjust if accumulating <
16 samples */

    ADC0.MUXPOS = ADC_VIA_PGA_gc | ADC_MUXPOS_AIN6_gc; /* ADC channel AIN6 -> PA6 */
    ADC0.MUXNEG = ADC_VIA_PGA_gc | ADC_MUXNEG_AIN7_gc; /* ADC channel AIN7 -> PA7 */
    ADC0.COMMAND = ADC_DIFF_bm | ADC_MODE_BURST_SCALING_gc; /* Burst Accumulation with
Scaling */

    /* Enable PGA with 16x gain.
    Set full bias current for fast sampling. Configure ADCPGASAMPDUR according to data sheet.
*/
    ADC0.PGACTRL = ADC_GAIN_16X_gc | ADC_PGABIASSEL_1X_gc | ADC_ADCPGASAMPDUR_6CLK_gc |
ADC_PGAEN_bm;
}

int main(void)
{
    adc_init();

    while(1)
    {
        ADC0.COMMAND |= ADC_START_IMMEDIATE_gc;
        while(!(ADC0.INTFLAGS & ADC_RESRDY_bm)); /* Wait until conversion is done */

        adc_reading = ADC0.RESULT; /* Read 16 bit scaled or left adjusted result */

        /* Calculate the differential voltage, VREF = 1.024V, 16-bit resolution, 16x gain. */
        voltage = (float)((adc_reading * 1.024) / ADC_DIFF_MAX_VALUE_16BIT) / 16;
        //current = voltage / 5;   /* Uncomment this line if measuring across a 5 ohm
resistor in series with the power supply */

        _delay_ms(500);
    }
}
```

A use case example of the code above is current measurement. Like shown in the image below, minor hardware modification must be done to be able to measure the device's current consumption. Selecting a resistor size for current measurement depends on multiple factors. For example, if the resistance is high, it may introduce a big voltage drop which may lead to shortened battery life in a battery powered application. On the other hand, if the resistance is low, it may lead to lower resolution. A 5Ω resistor in this example leads to a measuring step size of 400 nA and a range of 400 nA to 12.8 mA.

1. Connect the resistor between V_{PS} and V_{DD}.
2. Connect PA6 to V_{PS} and PA7 to V_{DD}.

Figure 5-1. Current Measurement Connection



In software, the following must be done to read the current consumption measurements:

1. Uncomment the line below the voltage calculation in the `main()` function in the code.
2. Add a breakpoint in the code and click *Start Debugging* to start debugging.
3. When the program has been halted at the breakpoint, right click on the `current` variable and select **Add Watch**. The `current` variable will now be visible in the watch window.
4. Click **Continue** to update the value(1).

Note:

1. If controlling the measurements with a multimeter, the results might deviate. Since the multimeter also measures at times where the ADC is inactive, it will normally show a lower current consumption.

5.3 Left Adjust

The Left Adjust (LEFTADJ) bit in the Control F (ADCn.CTRLF) register enables left-shift of the output data in the modes where this is supported. If enabled, this will left-shift the output from both the Result and the Sample registers. It is configured as shown in the following code snippet.

```
ADC0.CTRLF = ADC_LEFTADJ_bm; /* Enable Left Adjust bit */
```

Left adjust is available in ADC mode 5, Burst Accumulation with Scaling.

The following tables show how the left adjust feature affects the Result register output format in Single-Ended and Differential modes.

Table 5-1. RESULT Register – Single-Ended Mode – ADC Mode 5

LEFTADJ	RES[31:24]	RES[23:16]	RES[15:12]	RES[11:8]	RES[7:0]
0	0x00			Scaled accumulation[15:0]	
1	0x00			Scaled accumulation[15:0](1)	

Table 5-2. RESULT Register – Differential Mode – ADC Mode 5

LEFTADJ	RES[31:24]	RES[23:16]	RES[15:12]	RES[11:8]	RES[7:0]
0	Sign Extension			Signed scaled accumulation[15:0]	
1	Sign Extension			Signed scaled accumulation[15:0](1)	

If SAMPNUM < 4, the result is left-shifted 4 - SAMPNUM bits such that bit 15 is the MSb. If SAMPNUM > 4, the scaling feature of the ADC mode right shifts the result by SAMPNUM - 4 bits.

Example 1: SAMPNUM = 3 (Accumulation with 8 samples)

When the Left Adjust bit is '0', ADCn.SAMPLE = 0xFFF for all samples, and 8 samples are accumulated, the accumulated value in ADCn.RESULT is 8 x 4095 = 0x7FF8.

After enabling the Left Adjust bit, the result will be left-shifted by 4 - SAMPNUM = 1 bit (SAMPNUM is 3 to accumulate 8 samples). So by left-shifting 0x7FF8 by 1 bit, ADCn.RESULT = 0xFFF0.

Example 2: SAMPNUM = 5 (Accumulation with 32 samples)

When the Left Adjust bit is '0' and if the ADCn.SAMPLE = 0xFFF for all samples, and 32 samples are accumulated, the accumulated value is 32 x 4095 = 0x1FFE0.

In Scaling mode, the result will be right-shifted by SAMPNUM - 4 = 1 bit (SAMPNUM is 5 to accumulate 32 samples). So by right-shifting 0x1FFE0 by 1 bit, ADCn.RESULT = 0xFFF0.

The following table shows how the left adjust feature affects the Sample register output format in Single-Ended and Differential modes.

Table 5-3. SAMPLE Register – Single-Ended/Differential Mode – ADC Mode 5

LEFTADJ	DIFF	SAMPLE[15:12]	SAMPLE[11:8]	SAMPLE[7:0]
0	0	0x00	Conversion[11:0]	
	1	Sign extension	Signed conversion[11:0]	
1	0	Conversion[11:0] << 4		
	1	Signed conversion[11:0] << 4		

For example, if the Left Adjust feature is disabled and the ADCn.SAMPLE is 0x0FFF, the corresponding ADCn.SAMPLE value when Left Adjust is enabled is 0xFFF0.

5.4 Signed and Unsigned Output

The data format for a sample in Single-Ended mode is unsigned one's complement, where 0x0000 represents zero and 0x0FFF represents the largest number. If the analog input is higher than the reference level of the ADC, the 12-bit ADC output will be equal to the maximum value of 0x0FFF. Likewise, if the input is below 0V, the ADC output will be 0x0000.

For Differential mode, the data format is two's complement with sign extension.

Sample Register Output

The data type of the sample variable should be uint16_t when using Single-Ended mode.

The data type of the sample variable should be int16_t when using Differential mode.

For example, when using Single-Ended mode in 12-bit mode, the voltage of a single sample may be interpreted as shown in the code snippet below.

```
uint16_t sample_variable = ADCn.SAMPLE;
float sample_voltage = (sample_variable * VREF) / 4095;
```

When using Differential mode in 12-bit mode, the voltage of a single sample may be interpreted as shown in the code snippet below.

```
int16_t sample_variable = ADCn.SAMPLE;
float sample_voltage = (sample_variable * VREF) / 2047;
```

Result Register Output

The data type of the result variable should be `uint32_t` when using Single-Ended mode.

The data type of the result variable should be `int32_t` when using Differential mode.

For example, when using Single-Ended mode in 12-bit mode, the voltage of SAMPNUM accumulated samples may be interpreted as shown in the code snippet below.

```
uint32_t result_variable = ADCn.SAMPLE;
float result_voltage = ((result_variable * VREF) / SAMPNUM) / 4095;
```

When using Differential mode in 12-bit mode, the voltage of SAMPNUM accumulated samples may be interpreted as shown in the code snippet below.

```
int32_t result_variable = ADCn.SAMPLE;
float result_voltage = ((result_variable * VREF) / SAMPNUM) / 2047;
```

6. Get Code Examples from GitHub

The code examples are available through GitHub, which is a web-based server that provides the application codes through a Graphical User Interface (GUI). The code examples can be opened in both Atmel Studio and MPLAB X. To open the Atmel Studio project in MPLAB X, select from the menu in MPLAB X, *File > Import > Atmel Studio Project* and navigate to `.cproj` file.

The GitHub webpage: GitHub.

Code Examples

Finding example code for devices in the tinyAVR 2 family can be done by searching for the device name, e.g. ATtiny1627, in the GitHub example browser.

Download the code as a `.zip` file from the example page on GitHub by clicking the **Clone** or **download** button.

7. Revision History

Revision	Date	Description
A	07/2020	Initial document release

The Microchip Website

Microchip provides online support via our website at www.microchip.com/. 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 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

Information contained in this publication 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

Trademarks

ISBN: 978-1-5224-6462-4

Quality Management System

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

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: 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: 45-4485-5910 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: 47-72884388 **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