TB3209

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
Click to browse repository

Table of Contents

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® 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

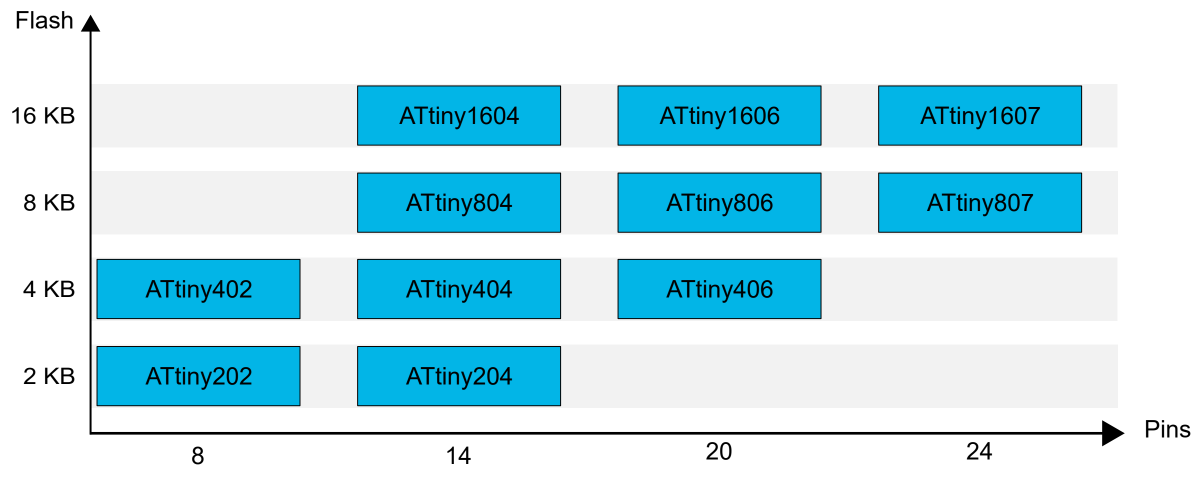


Figure 1-2. tinyAVR® 1-series Overview

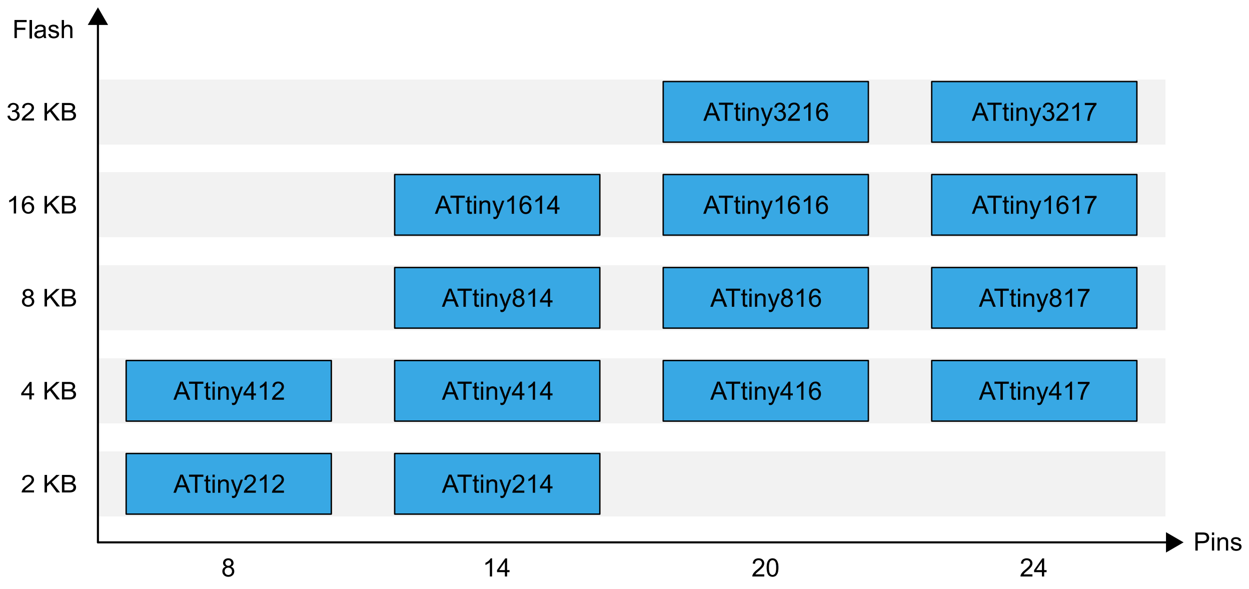


Figure 1-3. megaAVR® 0-series Overview

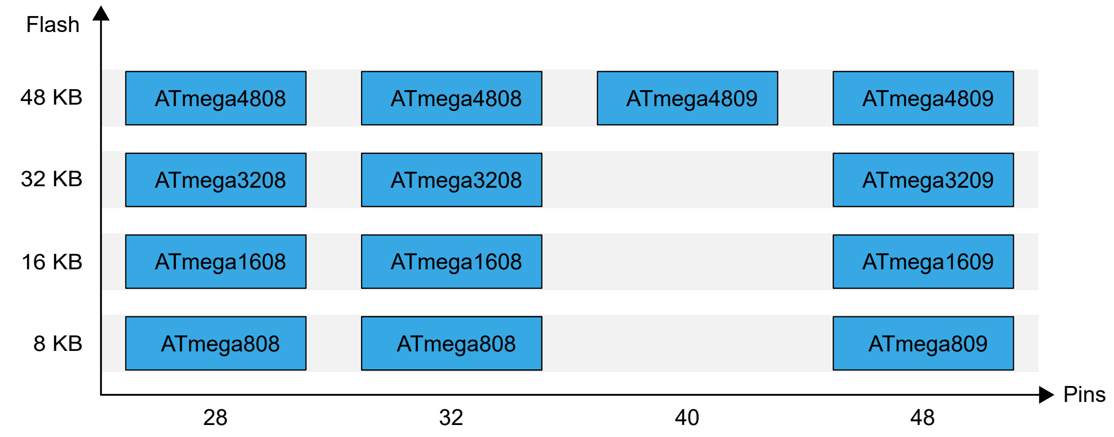
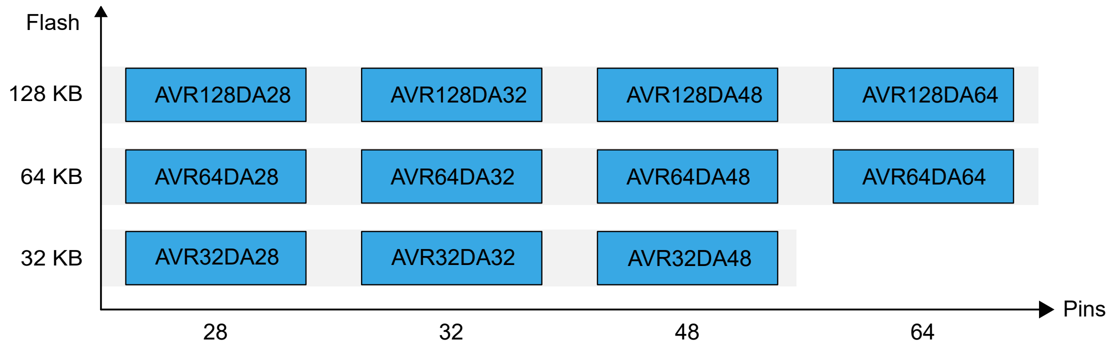


Figure 1-4. AVR® DA Family Overview

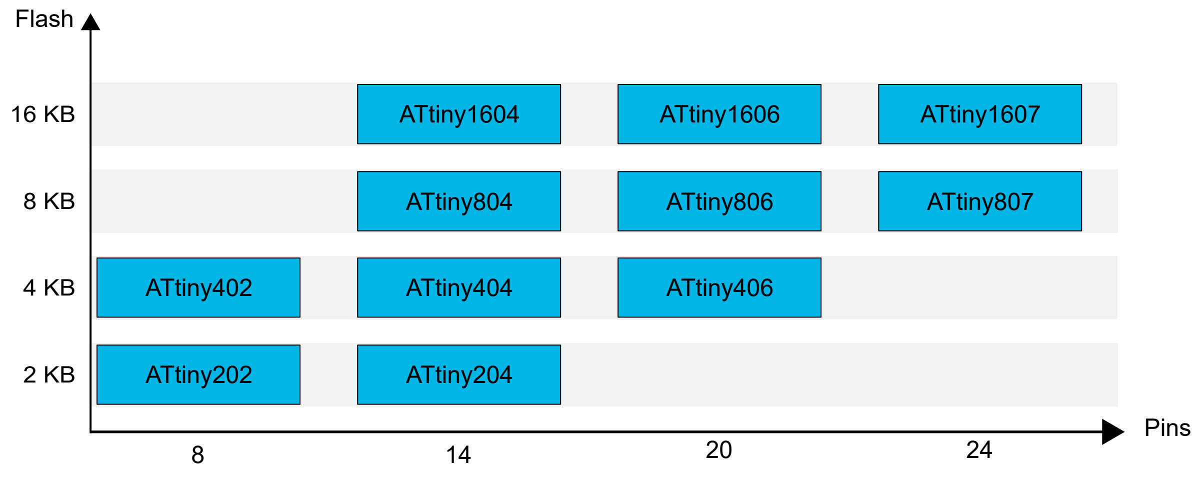


1.1 tinyAVR® 0-series

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

Figure 1-5. tinyAVR® 0-series Overview



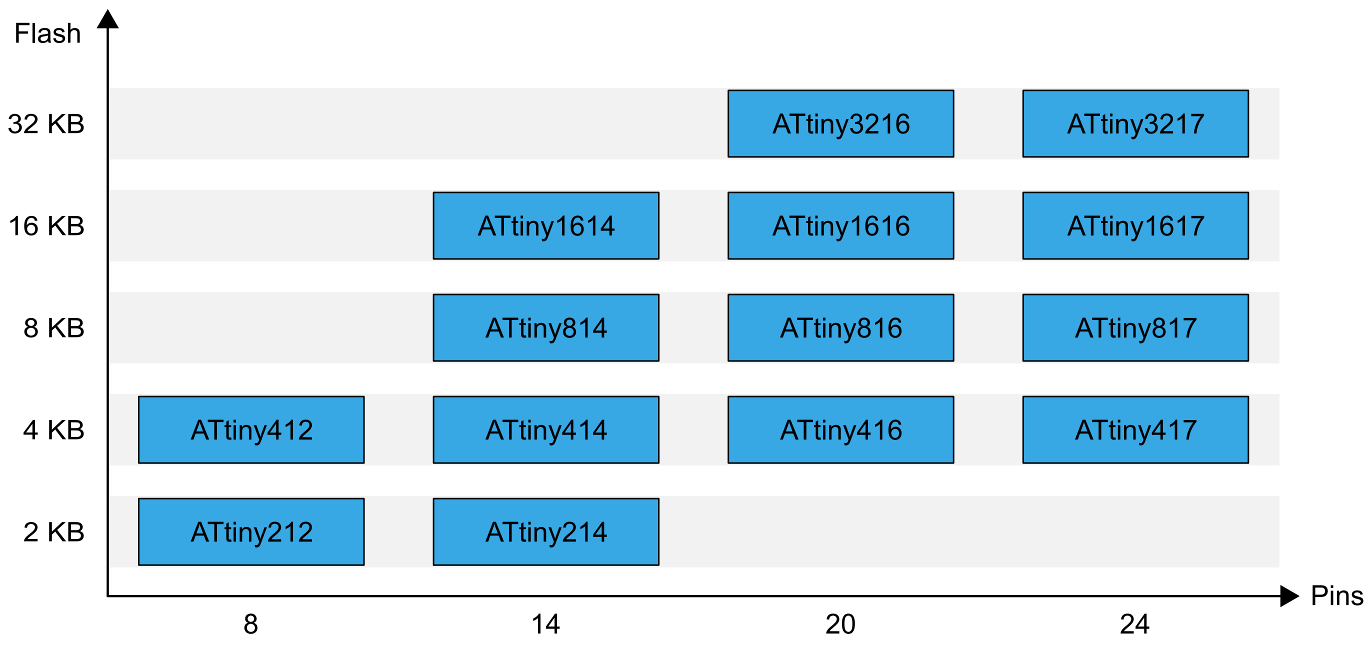
Devices with different Flash memory sizes typically also have different SRAM and EEPROM.

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



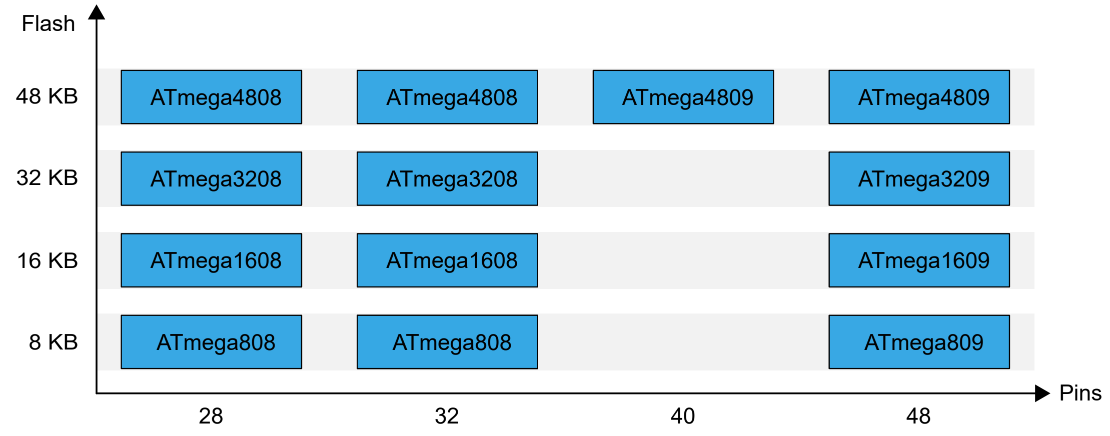
Devices with different Flash memory sizes typically also have different SRAM and EEPROM.

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



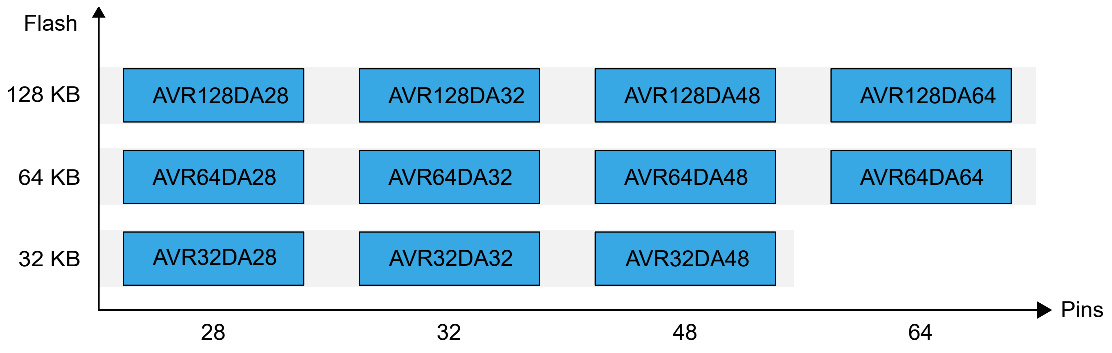
Devices with different Flash memory sizes typically also have different SRAM and EEPROM.

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

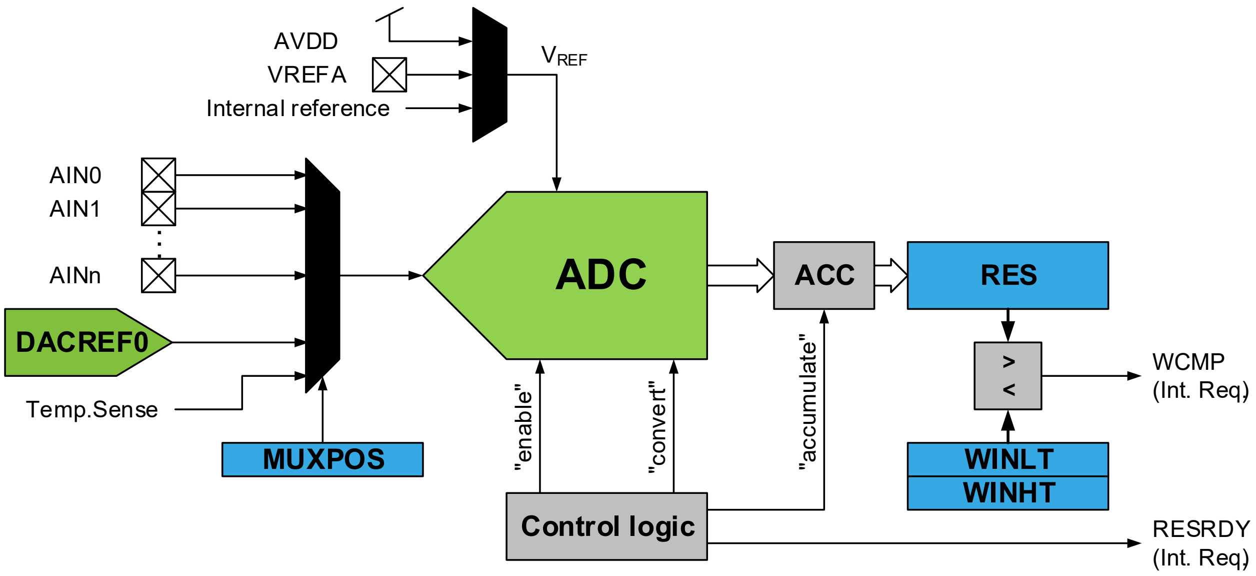
The Analog-to-Digital Converter (ADC) peripheral produces 10-bit results on tinyAVR and megaAVR microcontrollers and 10-bit/12-bit result on AVR DA microcontrollers. The ADC input can either be internal (e.g,. a voltage reference), or external through the analog input pins. The ADC is connected to an 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_{REF} 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

Bit	7	6	5	4	3	2	1	0
				MUXPOS[4:0]				
Access				R/W	R/W	R/W	R/W	R/W
Reset				0	0	0	0	0

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	REFSEL[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_{REFA}
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

```
ADC0.CTRLC |= ADC_PRESC_DIV4_gc;
ADC0.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	7	6	5	4	3	2	1	0
	RUNSTBY					RESSEL	FREERUN	ENABLE
Access	R/W					R/W	R/W	R/W
Reset	0					0	0	0

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

Value	Description
0	ADC is disabled
1	ADC is enabled

```
ADC0.CTRLA |= ADC_RESSEL_10BIT_gc;
ADC0.CTRLA |= ADC_ENABLE_bm;
```

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

Figure 3-4. ADC0.COMMAND - Start Conversion

Bit	7	6	5	4	3	2	1	0
								STCONV
Access								R/W
Reset								0

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.

```
ADC0.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	7	6	5	4	3	2	1	0
							WCOMP	RESRDY
Access							R/W	R/W
Reset							0	0

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 (!(ADC0.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	7	6	5	4	3	2	1	0
	RUNSTBY					RESSEL	FREERUN	ENABLE
Access	R/W					R/W	R/W	R/W
Reset	0					0	0	0

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.

```
ADC0.CTRLA |= ADC_FREERUN_bm;
```

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

```
ADC0.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.

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

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

Bit	7	6	5	4	3	2	1	0
						SAMPNUM[2:0]		
Access						R/W	R/W	R/W
Reset						0	0	0

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 (!(ADC0.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.

```
ADC0.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
	WINLT[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
	WINLT[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

Bit	7	6	5	4	3	2	1	0
						WINCM[2:0]		
Access						R/W	R/W	R/W
Reset						0	0	0

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	7	6	5	4	3	2	1	0
							WCOMP	RESRDY
Access							R/W	R/W
Reset							0	0

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	7	6	5	4	3	2	1	0
								STARTEI
Access								R/W
Reset								0

bit 0 STARTEI: Start Event Input bit
This bit enables using the event input as trigger for starting a conversion.

```
ADC0.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.
3. 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.

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

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
B	03/2021	Updated the GitHub repository links, the *References* section, and the use cases sections. Added the *AVR® DA Family Overview* and *Revision History* sections. Added MCC versions for each use case, running on AVR128DA48. Other minor corrections.
A	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;

    ADC0.CTRLC = ADC_PRESC_DIV4_gc          /* CLK_PER divided by 4 */
               | ADC_REFSEL_INTREF_gc;      /* Internal reference */

    ADC0.CTRLA = ADC_ENABLE_bm              /* ADC Enable: enabled */
               | 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)
{
    ADC0_init();

    adcVal = ADC0_read();

    while (1)
    {
        ;
    }
}
```

Example 10-2. ADC Free-Running Code Example

```c
#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;

    ADC0.CTRLC = ADC_PRESC_DIV4_gc          /* CLK_PER divided by 4 */
               | ADC_REFSEL_INTREF_gc;      /* Internal reference */

    ADC0.CTRLA = ADC_ENABLE_bm              /* ADC Enable: enabled */
               | ADC_RESSEL_10BIT_gc;       /* 10-bit mode */

    /* Select ADC channel */
    ADC0.MUXPOS = ADC_MUXPOS_AIN6_gc;

    /* Enable FreeRun mode */
    ADC0.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();
    ADC0_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

```c
#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;

    ADC0.CTRLC = ADC_PRESC_DIV4_gc          /* CLK_PER divided by 4 */
               | ADC_REFSEL_INTREF_gc;      /* Internal reference */

    ADC0.CTRLA = ADC_ENABLE_bm              /* ADC Enable: enabled */
               | ADC_RESSEL_10BIT_gc;       /* 10-bit mode */

    /* 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 ( !(ADC0.INTFLAGS & ADC_RESRDY_bm) )
    {
        ;
    }

    /* Clear the interrupt flag by writing 1: */
    ADC0.INTFLAGS = ADC_RESRDY_bm;

    return ADC0.RES;
}

int main(void)
{
    ADC0_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 ADC0_resultBelowTreshold(void);
void ADC0_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;

    ADC0.CTRLC = ADC_PRESC_DIV4_gc           /* CLK_PER divided by 4 */
               | ADC_REFSEL_INTREF_gc;      /* Internal reference */

    ADC0.CTRLA = ADC_ENABLE_bm               /* ADC Enable: enabled */
               | ADC_RESSEL_10BIT_gc;       /* 10-bit mode */

    /* Select ADC channel */
    ADC0.MUXPOS = ADC_MUXPOS_AIN6_gc;

    /* Set conversion window comparator low threshold */
    ADC0.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: */
    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);
}

bool ADC0_resultBelowTreshold(void)
{
    return (ADC0.INTFLAGS & ADC_WCMP_bm);
}

void ADC0_clearWindowCmpIntFlag(void)
{
    /* Clear the interrupt flag by writing 1: */
    ADC0.INTFLAGS = ADC_WCMP_bm;
}

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)
{
    ADC0_init();
    LED0_init();

    ADC0_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;

    ADC0.CTRLC = ADC_PRESC_DIV4_gc          /* CLK_PER divided by 4 */
               | ADC_REFSEL_INTREF_gc;      /* Internal reference */

    ADC0.CTRLA = ADC_ENABLE_bm              /* ADC Enable: enabled */
               | ADC_RESSEL_10BIT_gc;       /* 10-bit mode */

    /* Select ADC channel */
    ADC0.MUXPOS = ADC_MUXPOS_AIN6_gc;

    /* Enable interrupts */
    ADC0.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 */
    }

    RTC.CTRLA = RTC_PRESCALER_DIV32_gc      /* 32 */
              | RTC_RTCEN_bm                /* Enable: enabled */
              | 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)
{
    ADC0_init();
    LED0_init();
    RTC_init();
    EVSYS_init();

    /* Enable Global Interrupts */
    sei();

    while (1)
    {
        ;
    }
}
```

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

Legal Notice

Trademarks

ISBN: 978-1-5224-7819-5

Quality Management System

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

Worldwide Sales and Service

AMERICAS

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

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

Austin, TX
Tel: 512-257-3370

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

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

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

Detroit
Novi, MI
Tel: 248-848-4000

Houston, TX
Tel: 281-894-5983

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

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

Raleigh, NC
Tel: 919-844-7510

New York, NY
Tel: 631-435-6000

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

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

ASIA/PACIFIC

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

China - Xiamen
Tel: 86-592-2388138

China - Zhuhai
Tel: 86-756-3210040

ASIA/PACIFIC

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

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

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

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

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

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

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

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

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

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

Singapore
Tel: 65-6334-8870

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

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

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

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

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

EUROPE

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

Denmark - Copenhagen
Tel: 45-4485-5910
Fax: 45-4485-2829

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

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

Germany - Garching
Tel: 49-8931-9700

Germany - Haan
Tel: 49-2129-3766400

Germany - Heilbronn
Tel: 49-7131-72400

Germany - Karlsruhe
Tel: 49-721-625370

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

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

Israel - Ra’anana
Tel: 972-9-744-7705

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

Italy - Padova
Tel: 39-049-7625286

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

Norway - Trondheim
Tel: 47-72884388

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