TB3245

Using 12-Bit ADC for Conversions, Accumulation, and Triggering Events

Introduction

Author: Ștefan Vlad, Microchip Technology Inc.

The Analog-to-Digital Converter (ADC) is a 12-bit Successive Approximation Register (SAR) with possibilities for differential and single-ended conversions. This peripheral is available on the AVR® DA MCU (AVR DA) microcontrollers.

This technical brief describes how the ADC module works on the AVR DA microcontrollers, and it covers the following use cases:

- **ADC Single Conversion:**
  Initialize the ADC, start the conversion and wait until it is completed, and read the ADC result in a loop.
- **ADC Free-Running Conversion:**
  Initialize the ADC, enable Free-Running mode, start the conversion and wait until it is completed, and read the ADC result in an infinite loop.
- **ADC Differential Conversion:**
  Initialize the ADC, configure two pins for reading the differential voltage, start the conversion and wait until it is completed, and read the ADC result in an infinite loop.
- **ADC Sample Accumulator:**
  Initialize the ADC, enable accumulation of 64 samples, start the conversion and wait until it is completed, 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 and wait until it is completed, and read the ADC result in an infinite loop. An LED is toggled on 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 the RTC overflow. An LED is toggled on after each ADC conversion.
- **ADC Temperature Measurement:**
  Initialize the ADC, select the internal reference, select the temperature sensor as input and acquire the data by running a 12-bit, right adjusted, single-ended conversion.

The ADC results for all examples will be transmitted through Universal Asynchronous Receiver-Transmitter (USART) and plotted using the Data Visualizer tool.

Note: The code examples were developed on AVR128DA48 Curiosity Nano. They are available on GitHub, supporting both Atmel Studio and MPLAB™ X Integrated Development Environment (IDE).

Table of Contents

1. Relevant Devices

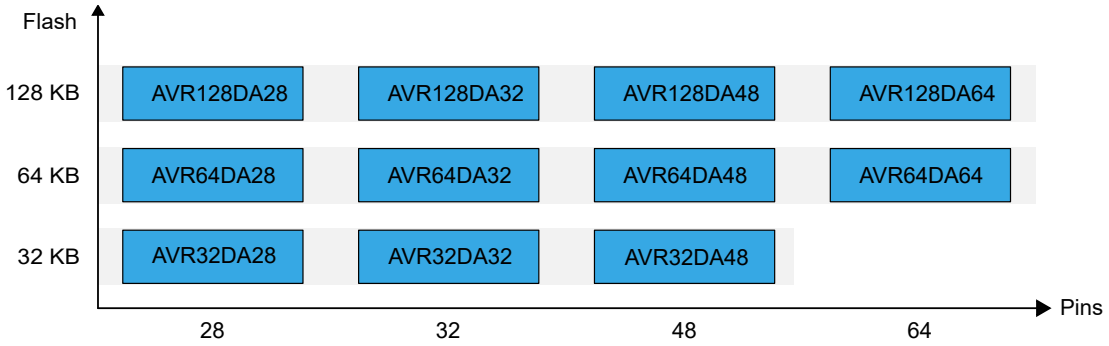
This section lists the relevant devices for this document.

1.1 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 1-1. AVR® DA Family Overview



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

2. Overview

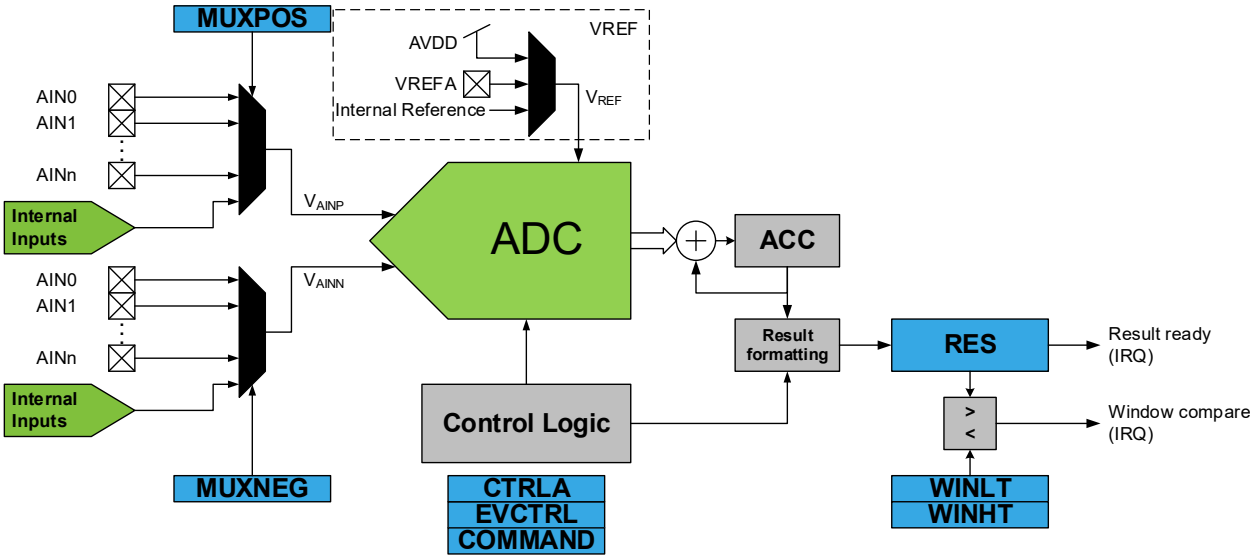
The ADC is a SAR with a sampling rate of up to 130 ksps, at 12-bit resolution. The ADC is connected to an analog input multiplexer for selection between multiple single-ended or differential inputs. In single-ended conversions, the ADC measures the voltage between the selected input channels and 0V (GND). In differential conversions, it measures the voltage between two selected input channels. The selected ADC input channels can either be internal (e.g. a voltage reference) or external analog input pins.

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

Selectable voltage references from the internal Voltage Reference (VREF) peripheral, AVDD supply voltage, or external VREF pin (VREFA).

A digital window compare feature is available for monitoring the input signal and can be configured only to trigger an interrupt, if the sample is below or above a user-defined threshold; or if it is inside or outside a user-defined window, with minimum software intervention required.

Figure 2-1. ADC Block Diagram



The analog input channel is selected by writing to the MUXPOS bit field in the MUXPOS (ADCn.MUXPOS) register. Any of the ADC input pins, GND, internal inputs, or temperature sensor can be selected as a 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. The 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 is '0'. The ADC generates a 10- or 12-bit right-adjusted result which can be read from the Result (ADCn.RES) register.

The conversion equations, where V_{AINP} and V_{AINN} are the positive and negative ADC inputs and V_{REF} is the selected ADC voltage reference are listed below.

- Single-ended 12-bit conversion: $RES = \frac{V_{\mathrm{AINP}}}{V_{\mathrm{REF}}} \times 4096 \in [0, 4095]$
- Single-ended 10-bit conversion: $RES = \frac{V_{\mathrm{AINP}}}{V_{\mathrm{REF}}} \times 1024 \in [0, 1023]$
- Differential 12-bit conversion: $RES = \frac{V_{\mathrm{AINP}} - V_{\mathrm{AINN}}}{V_{\mathrm{REF}}} \times 2048 \in [-2048, 2047]$
- Differential 10-bit conversion: $RES = \frac{V_{\mathrm{AINP}} - V_{\mathrm{AINN}}}{V_{\mathrm{REF}}} \times 512 \in [-512, 511]$

3. Hardware Configuration

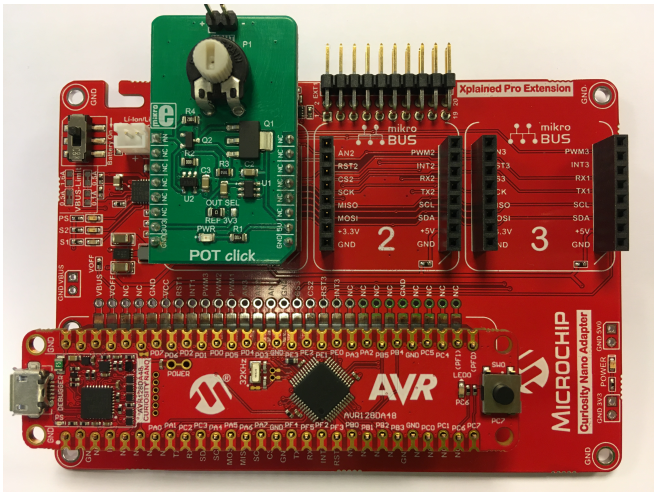
The hardware used for this example is based on the:

- AVR128DA48 Curiosity Nano (DM164151)
- Curiosity Nano Base for Click Boards (AC164162)
- mikroBUS™ POT click board (MIKRO-3402) or
- mikroBUS™ POT 2 click board (MIKROE-3325)

Two setups will be used for the use cases described in this document:

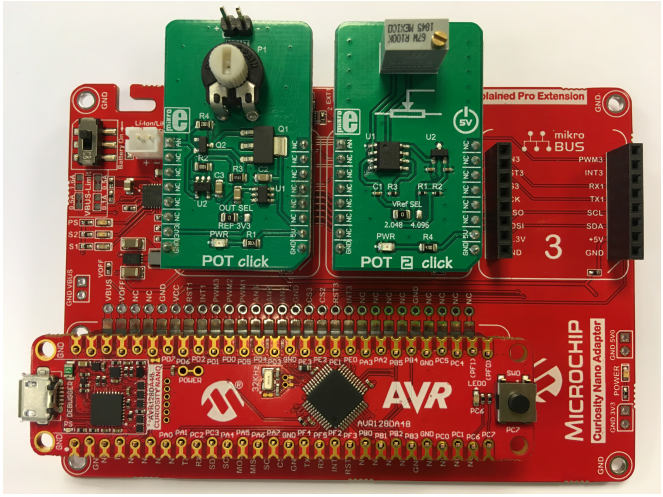
- The first configuration (Config A) uses a POT click, inserted in mikroBUS slot 1, connects AN1 to the PD3 (AIN3) pin of the AVR DA device.

Figure 3-1. Hardware Configuration A with 1 POT Click



- The second configuration (Config B) uses two POT clicks, inserted in mikroBUS slot 1 and slot 2. Slot 1 connects AN1 to the PD3 (AIN3) of the AVR DA device and slot 2 connects AN2 to PD4 (AIN4) pin of the AVR DA device.

Figure 3-2. Hardware Configuration B with 2 POT Clicks



4. ADC Single Conversion

4.1 Initialize the ADC

This subsection showcases how to initialize the ADC module to run in Single Conversion mode. The ADC input pin needs to have the digital input buffer and the pull-up resistor disabled, to have the highest possible input impedance.

The ADC Single Conversion uses the Config A setup explained in 3. Hardware Configuration.

Figure 4-1. ADC0.MUXPOS Selection

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

Bits 6:0 – MUXPOS[6:0] MUX Selection for Positive ADC Input
This bit field selects which analog input is connected to the positive input of the ADC. If this bit field is changed during a conversion, the change will not take effect until the conversion is complete.

Value	Name	Description
0x00-0x0F	AIN0-AIN15	ADC input pin 0-15
0x10-0x15	AIN16-AIN21	ADC input pin 16-21
0x16-0x3F	-	Reserved
0x40	GND	Ground
0x41	-	Reserved
0x42	TEMPSENSE	Temperature sensor
0x43-0x47	-	Reserved
0x48	DAC0	DAC0
0x49	DACREF0	DACREF0
0x4A	DACREF1	DACREF1
0x4B	DACREF2	DACREF2
Other	-	Reserved

To select the ADC channel AIN3, PD3, use the code below:

```
ADC0.MUXPOS = ADC_MUXPOS_AIN3_gc;
```

The ADC Clock Prescaler can be used to divide the clock frequency. In this example, the clock is divided by 4.

```
ADC0.CTRLC |= ADC_PRESC_DIV4_gc;
```

Figure 4-2. ADC0 Reference Selection

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

Bit 7 – ALWAYSON Reference Always On
This bit controls whether the ADC0 reference is always on or not.

Value	Description
0	The reference is automatically enabled when needed
1	The reference is always on

Bits 2:0 – REFSEL[2:0] Reference Select
This bit field controls the reference voltage level for ADC0.
Note:

1. The values given for internal references are only typical. Refer to the *Electrical Characteristics* section for further details.

Value	Name	Description
0x0	1V024	Internal 1.024V reference(1)
0x1	2V048	Internal 2.048V reference(1)
0x2	4V096	Internal 4.096V reference(1)
0x3	2V500	Internal 2.500V reference(1)
0x4	-	Reserved
0x5	VDD	VDD as reference
0x6	VREFA	External reference from the VREFA pin
0x7	-	Reserved

The ADC can use V_{DD}, external or internal reference for its positive reference. The internal reference is used in this example.

```
VREF.ADC0REF = VREF_REFSEL_2V048_gc; /* Internal 2.048V reference */
```

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 4-3. ADC0.CTRLA Resolution Selection

Bits 3:2 – RESSEL[1:0] Resolution Selection
This bit field selects the ADC resolution. When changing the resolution from 12-bit to 10-bit, the conversion time is reduced from 13.5 CLK_ADC cycles to 11.5 CLK_ADC cycles.

Value	Description
0x00	12-bit resolution
0x01	10-bit resolution
Other	Reserved

This translates into the following code:

```
ADC0.CTRLA = ADC_ENABLE_bm              /* ADC Enable: enabled */
           | ADC_RESSEL_12BIT_gc;       /* 12-bit mode */
```

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

Figure 4-4. ADC0.COMMAND - Start Conversion

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

Bit 1 – SPCONV Stop Conversion
Writing a '1' to this bit will end the current measurement. This bit will take precedence over the Start Conversion (STCONV) bit. Writing a '0' to this bit has no effect.

Bit 0 – STCONV Start Conversion
Writing a '1' to this bit will start a conversion as soon as any ongoing conversions are completed. 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. Writing a '0' to this bit has no effect.

This translates into the following code:

```
ADC0.COMMAND = ADC_STCONV_bm;
```

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

Figure 4-5. ADC0.INTFLAGS - Hardware-Set RESRDY Bit

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

Bit 1 – WCMP Window Comparator Interrupt Flag
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 (ADCn.RES) register. Writing a '0' to this bit has no effect.

Bit 0 – RESRDY Result Ready Interrupt Flag
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 (ADCn.RES) register. Writing a '0' to this bit has no effect.

The user must clear the RESRDY bit by reading the ADC0.RES register before starting another conversion:

```
/* Clear the interrupt flag by reading the result */
    return ADC0.RES;
```

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

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

4.2 Transmit the Results

To visualize the ADC results using the Data Visualizer plug-in, the user must first send the results through USART. The USART BAUD register must be used to configure the baud rate. It is presented in the figure below.

Figure 4-6. USART Baud Register

Name: BAUD
Offset: 0x08
Reset: 0x00
Property: -

The USARTn.BAUDL and USARTn.BAUDH register pair represents the 16-bit value, USARTn.BAUD. The low byte [7:0] (suffix L) is accessible at the original offset. The high byte [15:8] (suffix H) can be accessed at offset + 0x01.

Ongoing transmissions of the transmitter and receiver will be corrupted if the baud rate is changed. Writing to this register will trigger an immediate update of the baud rate prescaler. For more information on how to set the baud rate, see Table 23-1, Equations for Calculating Baud Rate Register Setting.

Bit	15	14	13	12	11	10	9	8
	BAUD[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
	BAUD[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 – BAUD[15:8] USART Baud Rate High Byte
These bits hold the MSB of the 16-bit Baud register.

Bits 7:0 – BAUD[7:0] USART Baud Rate Low Byte
These bits hold the LSB of the 16-bit Baud register.

The following code must be used.

```
USART1.BAUD = (uint16_t)USART1_BAUD_RATE(115200);
```

The USART1_BAUD_RATE macro is used to compute the corresponding value for the BAUD register. It is presented below.

```
#define USART1_BAUD_RATE(BAUD_RATE) (((float)F_CPU*64/(16 *(float)BAUD_RATE))+0.5)
```

The USART transmission can be enabled using the CTRLB register, presented below.

Figure 4-7. Control B

Name: CTRLB
Offset: 0x06
Reset: 0x00
Property: -

Bit	7	6	5	4	3	2	1	0
	RXEN	TXEN		SFDEN	ODME	RXMODE[1:0]		MPCM
Access	R/W	R/W		R/W	R/W	R/W	R/W	R/W
Reset	0	0		0	0	0	0	0

The following code must be used.

```
USART1.CTRLB = USART_TXEN_bm;           /* Enable TX */
```

The character size can be configured using the CTRLC register.

Figure 4-8. Control C

23.5.8 Control C - Asynchronous Mode

Name: CTRLC
Offset: 0x07
Reset: 0x03
Property: -

This register description is valid for all modes except the Master SPI mode. When the USART Communication Mode bits (CMODE) in this register are written to 'MSPI', see CTRLC - Master SPI mode for the correct description.

Bit	7	6	5	4	3	2	1	0
	CMODE[1:0]		PMODE[1:0]		SBMODE	CHSIZE[2:0]		
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	1	1

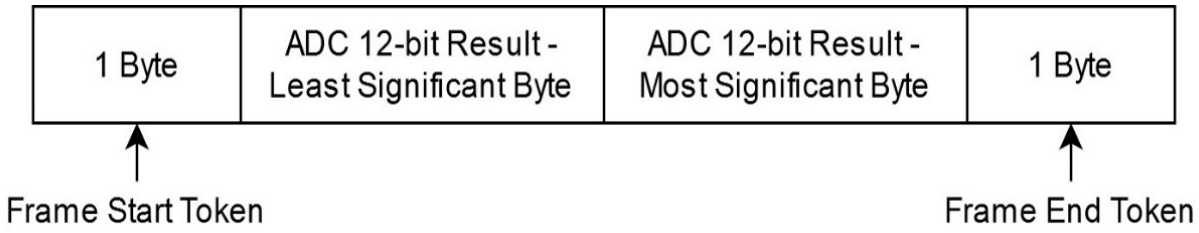
The following code must be used.

```
USART1.CTRLC = USART_CHSIZE_8BIT_gc;    /* Configure character size: 8 bit */
```

To transmit one byte through USART, the user must check if the USART buffer is ready to transmit data and then write the data to the TXDATAL register.

For the data to be correctly interpreted by the Data Visualizer, it must follow a specific format. The ADC result of 12-bit resolution will be provided using a 16-bit format (the sign bit is extended), and the two bytes of the result will be transmitted as described below.

Figure 4-9. ADC Result Transmitting Format



The frame start token (START_TOKEN) and the frame end token (END_TOKEN) macros are defined as described below.

```
#define START_TOKEN 0x03    /* Start Frame Token */
#define END_TOKEN 0xFC      /* End Frame Token */
```

Inside the infinite loop, the ADC result will be read, it will be transmitted through USART using a specific format, and the conversion will be started again.

The following code must be added to the main infinite loop to transmit the ADC result through USART:

```
while (1)
{
    /* Start the ADC conversion */
    ADC0_start();

    /* Read the ADC result */
    adcVal = ADC0_read();

    /* Transmit the ADC result to be plotted using Data Visualizer */
    USART1_Write(START_TOKEN);
    USART1_Write(adcVal &amp; 0x00FF);
    USART1_Write(adcVal >> 8);
    USART1_Write(END_TOKEN);
}
```

4.3 Code Examples

The code examples developed using Atmel Studio are available below.

The code examples developed using MPLAB X IDE are available below.

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

5. ADC Free-Running Conversion

5.1 Initialize the ADC

This subsection presents how to initialize the ADC module to run in the Free-Running Conversion mode. This application uses the Config A setup explained in 3. Hardware Configuration. The FREERUN bit in ADC0.CTRLA must be set to activate this mode, in addition to the normal ADC initialization. When configuring the ADC in Free-Running mode, the next conversion starts immediately after the previous one.

Figure 5-1. ADC0.CTRLA - Set the FREERUN Bit

Bit	7	6	5	4	3	2	1	0
	RUNSTBY		CONVMODE	LEFTADJ	RESSEL[1:0]		FREERUN	ENABLE
Access	R/W		R/W	R/W	R/W	R/W	R/W	R/W
Reset	0		0	0	0	0	0	0

This translates into the following code:

```
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.

5.2 Transmit the Results

The following algorithm must be implemented in the infinite loop to transmit the ADC conversion result:

```
while (1)
{
    /* Start the ADC conversion */
    ADC0_start();

    /* Read the ADC result */
    adcVal = ADC0_read();

    /* Transmit the ADC result to be plotted using Data Visualizer */
    USART1_Write(START_TOKEN);
    USART1_Write(adcVal &amp; 0x00FF);
    USART1_Write(adcVal >> 8);
    USART1_Write(END_TOKEN);
}
```

5.3 Code Examples

The code examples developed using Atmel Studio are available below.

View Code Examples on GitHub
Click to browse repositories

The code examples developed using MPLAB X IDE are available below.

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

6. ADC Differential Conversion

6.1 Initialize the ADC

This subsection presents how to initialize the ADC module to run in Differential Conversion mode. This application uses the Config B setup explained in 3. Hardware Configuration.

The ADC Differential Conversion mode is enabled by setting the CONVMODE bit in the ADC0.CTRLA register:

Figure 6-1. ADC0.CTRLA - Set the CONVMODE Bit

Bit	7	6	5	4	3	2	1	0
	RUNSTBY		CONVMODE	LEFTADJ		RESSEL	FREERUN	ENABLE
Access	R/W		R/W	R/W		R/W	R/W	R/W
Reset	0		0	0		0	0	0

Bit 5 – CONVMODE Conversion Mode
This bit defines if the ADC is working in Single-Ended or Differential mode.

Value	Name	Description
0	SINGLEENDED	The ADC is operating in Single-Ended mode where only positive input is used. The ADC result is presented as an unsigned result.
1	DIFF	The ADC is operating in Differential mode where both positive and negative inputs are used. The ADC result is presented as a signed result.

This translates into the following code:

```
ADC0.CTRLA |= ADC_CONVMODE_bm;
```

For the Differential Conversion, the Pot click is connected to PD3 (AIN3) and the Pot 2 click is connected to PD4 (AIN4) pins of the AVR DA device.

Figure 6-2. MUX Selection for Negative ADC Input

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

Bits 6:0 – MUXNEG[6:0] MUX Selection for Negative ADC Input
This bit field selects which analog input is connected to the negative input of the ADC. If this bit field is changed during a conversion, the change will not take effect until the conversion is complete.

Value	Name	Description
0x00-0x0F	AIN0-AIN15	ADC input pin 0-15
0x10-0x3F	-	Reserved
0x40	GND	Ground
0x42-0x47	-	Reserved
0x48	DAC0	DAC0
Other	-	Reserved

```
ADC0.MUXNEG = ADC_MUXNEG_AIN4_gc;
```

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

6.2 Transmit the Results

The ADC differential conversion results can be read and transmitted, as presented in the code listing below.

```
while (1)
{
    if (ADC0_conversionDone())
    {
        /* Read the ADC result */
        adcVal = ADC0_read();
        /* Transmit the ADC result to be plotted using Data Visualizer */
        USART1_Write(START_TOKEN);
        USART1_Write(adcVal & 0x00FF);
        USART1_Write(adcVal >> 8);
        USART1_Write(END_TOKEN);
    }
}
```

6.3 Code Examples

The code examples developed using Atmel Studio are available below.

View Code Examples on GitHub
Click to browse repositories

The code examples developed using MPLAB X IDE are available below.

View Code Examples on GitHub
Click to browse repositories

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

7. ADC Sample Accumulator

7.1 Initialize the ADC

This subsection presents how to initialize the ADC module to use the Sample Accumulator feature. This application uses the Config A setup explained in 3. Hardware Configuration. In this mode, the ADC can add up to 128 samples in the accumulator register, thus filtering the signal and reducing the noise, which 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 7-1. 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
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 (ADC.RES) register 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	ACC128	128 results accumulated.

This translates into the following code:

```
ADC0.CTRLB = ADC_SAMPNUM_ACC64_gc;
```

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.

The user can read that value and divide it by the number of samples to get the average value. If there are more than 16 accumulated samples, the result will be truncated down in hardware to a 16-bit variable, and it needs to be divided by 16:

```
adcVal = ADC0_read(); /* Clear the interrupt flag by reading the result */
adcVal = adcVal >> ADC_SHIFT_DIV16; /* divide by No of samples or 16, if No. samples &gt; 16
*/
```

7.2 Transmit the Results

The ADC result must be read, divided by the number of samples, and transmitted through USART as presented below:

```
while (1)
{
    /* Read the ADC result */
    adcVal = ADC0_read();
```

```
    /* divide by No of samples or 16, if No. samples > 16 */
    adcVal = adcVal >> ADC_SHIFT_DIV16;
    /* Transmit the ADC result to be plotted using Data Visualizer */
    USART1_Write(START_TOKEN);
    USART1_Write(adcVal & 0x00FF);
    USART1_Write(adcVal >> 8);
    USART1_Write(END_TOKEN);
}
```

7.3 Code Examples

The code examples developed using Atmel Studio are available below.

View Code Examples on GitHub
Click to browse repositories

The code examples developed using MPLAB X IDE are available below.

View Code Examples on GitHub
Click to browse repositories

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

8. ADC Window Comparator

8.1 Initialize the ADC

In the Window Comparator configuration, the device detects 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 inside a specific range or for signaling low battery/overcharge. The Window Comparator can be used in both Free-Running mode and Single Conversion mode.

The ADC Window Comparator uses the Config A setup explained in 3. Hardware Configuration.

In this example, the Window Comparator is used in Free-Running mode. 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 `0x100` threshold is set in the ADC0.WINLT register.

Figure 8-1. ADC-WINLT - Set 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.

In the code snippet below, the WINDOW_CMP_LOW_TH_EXAMPLE macro is defined and set for `0x100` and provided as a low threshold for the Window Comparator:

```
#define WINDOW_CMP_LOW_TH_EXAMPLE    (0x100)
ADC0.WINLT = WINDOW_CMP_LOW_TH_EXAMPLE;
```

The ADC Window Comparator mode is set in the ADC0.CTRLE register:

Figure 8-2. ADC0.CTRLE - Set 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

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

The following line of code sets the threshold for results lower than WINLT:

```
ADC0.CTRLE = ADC_WINCM_BELOW_gc;
```

If the ADC result is below the previously set threshold value, the WCMP bit in the ADC0.INTFLAGS register is set by the hardware, and it needs to be cleared by writing a '1' to it or by reading the conversion result from the RES register.

8.2 Transmit the Results

The ADC result will be read. If the result is under a specific threshold, the LED will be turned on. Otherwise, it will be turned off. Then, the conversion result will be transmitted through USART, as presented below.

```
while (1)
{
    while (!ADC0_resultReady());

    if (ADC0_resultBelowTreshold())
    {
        LED0_on();
    }
    else
    {
        LED0_off();
    }

    adcVal = ADC0_read();

    /* Transmit the ADC result to be plotted using Data Visualizer */
    USART1_Write(START_TOKEN);
    USART1_Write(adcVal & 0x00FF);
    USART1_Write(adcVal >> 8);
    USART1_Write(END_TOKEN);
}
```

8.3 Code Examples

The code examples developed using Atmel Studio are available below.

The code examples developed using MPLAB X IDE are available below.

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

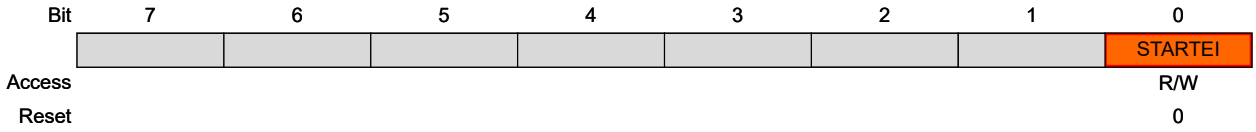
9. ADC Event Triggered

9.1 Initialize the ADC

An ADC conversion can also be triggered by an event, which is enabled by writing a '1' to the Start Event Input (STARTEI) bit in the Event Control (ADCn.EVCTRL) register (ADC0 instance will be used in this example).

The ADC Event Triggered use case is based on the Config A setup explained in 3. Hardware Configuration.

Figure 9-1. ADC0.EVTRL - Enable the STARTEI Bit



Bit 0 – STARTEI Start Event Input
This bit enables using the event input as trigger for starting a conversion. When a '1' is written to this bit, a rising event edge will trigger an ADC conversion.

The following line of code enables the ADC to be triggered by a rising edge event:

```
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, and STCONV will be cleared when the conversion is complete.

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

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 ADC0.EVCTRL register of the ADC must be set to allow the ADC conversion to be triggered by the 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 */
```

9.2 Transmit the Results

The ADC result will be transmitted through USART. To avoid corrupting the adcVal value because of the interrupt, the result will be stored using another variable, as presented below.

```
while (1)
{
    if (adcResultReady == 1)
    {
        /* Store the result to avoid altering adcVal because of the interrupt. This operation
must be atomic */
        cli();
        result = adcVal;
        sei();

        /* Update the flag value */
        adcResultReady = 0;
        /* Toggle the LED */
        LED0_toggle();

        /* Transmit the ADC result to be plotted using Data Visualizer */
        USART1_Write(START_TOKEN);
        USART1_Write(result & 0x00FF);
        USART1_Write(result >> 8);
```

```
            USART1_Write(END_TOKEN);
        }

    }
```

9.3 Code Examples

The code examples developed using Atmel Studio are available below.

The code examples developed using MPLAB X IDE are available below.

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

10. ADC Temperature Measurement

10.1 Initialize the ADC

The ADC Temperature Measurement uses the AVR DA on-chip temperature sensor, and no extra hardware is needed for this use case, except the AVR128DA48 Curiosity Nano Board. In addition to that, the input on the ADC0.MUXPOS register needs to be set on the temperature sensor:

Figure 10-1. ADC0.MUXPOS Selection

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

Bits 6:0 – MUXPOS[6:0] MUX Selection for Positive ADC Input
This bit field selects which analog input is connected to the positive input of the ADC. If this bit field is changed during a conversion, the change will not take effect until the conversion is complete.

Value	Name	Description
0x00-0x0F	AIN0-AIN15	ADC input pin 0-15
0x10-0x15	AIN16-AIN21	ADC input pin 16-21
0x16-0x3F	-	Reserved
0x40	GND	Ground
0x41	-	Reserved
0x42	TEMPSENSE	Temperature sensor
0x43-0x47	-	Reserved
0x48	DAC0	DAC0
0x49	DACREF0	DACREF0
0x4A	DACREF1	DACREF1
0x4B	DACREF2	DACREF2
Other	-	Reserved

To achieve more accurate readings, the result of the temperature sensor measurement must be processed in the application software using compensation values from the device production or user calibration.

The temperature is calculated by the following equation, and it is expressed in Kelvin:

$$T = \frac{(\text{Offset} - \text{ADC Result}) \times \text{Slope}}{4096}$$

The Temperature Sensor calibration value contains correction factors for temperature measurements from the on-chip temperature sensor. The SIGROW.TEMPSENSE0 is a correction factor for the gain/slope (unsigned), and SIGROW.TEMPSENSE1 is a correction factor for the offset (signed).

The following snippet code is recommended when using the compensation values from the signature row:

```
uint16_t sigrow_offset = SIGROW.TEMPSENSE1;
uint16_t sigrow_slope = SIGROW.TEMPSENSE0;

/* Clear the interrupt flag by reading ADC0.RES */
temp = sigrow_offset - ADC0.RES;
temp *= sigrow_slope; /* Result will overflow 16-bit variable */
temp += 0x0800; /* Add 4096/2 to get correct rounding on division below */
temp >>= 12; /* Round off to nearest degree in Kelvin, by dividing with 2^12 (4096) */
return temp - 273; /* Convert from Kelvin to Celsius (0 Kelvin - 273.15 = -273.1Â°C) */
```

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

10.2 Transmit the Results

The temperature will be transmitted through USART. After reading the ADC value, the result will be converted to obtain the degrees Celsius corresponding value.

```
while (1)
{
    /* Read the conversion result */
    adcVal = ADC0_read();
    /* Convert the ADC result in degrees C */
    temp_C = temperatureConvert(adcVal);

    /* Transmit the ADC result to be plotted using Data Visualizer */
    USART1_Write(START_TOKEN);
    USART1_Write(temp_C & 0x00FF);
    USART1_Write(temp_C >> 8);
    USART1_Write(END_TOKEN);
}
```

10.3 Code Examples

The code examples developed using Atmel Studio are available below.

View Code Examples on GitHub
Click to browse repositories

The code examples developed using MPLAB X IDE are available below.

View Code Examples on GitHub
Click to browse repositories

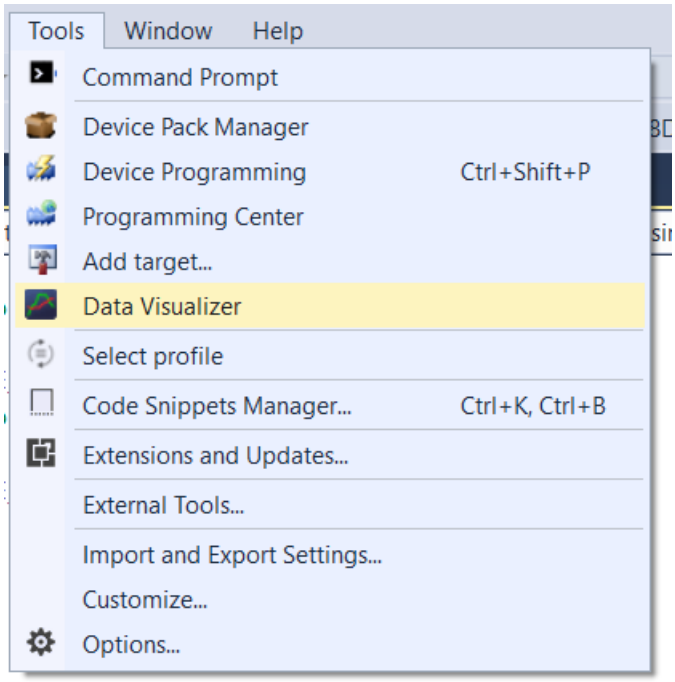
Tip: The full code example is available in the Appendix section.

11. Data Visualizer

To visualize and interpret the ADC results, the user must follow these steps:

1. Open the Data Visualizer tool by selecting *Tools >Data Visualizer*.

Figure 11-1. Open the Data Visualizer

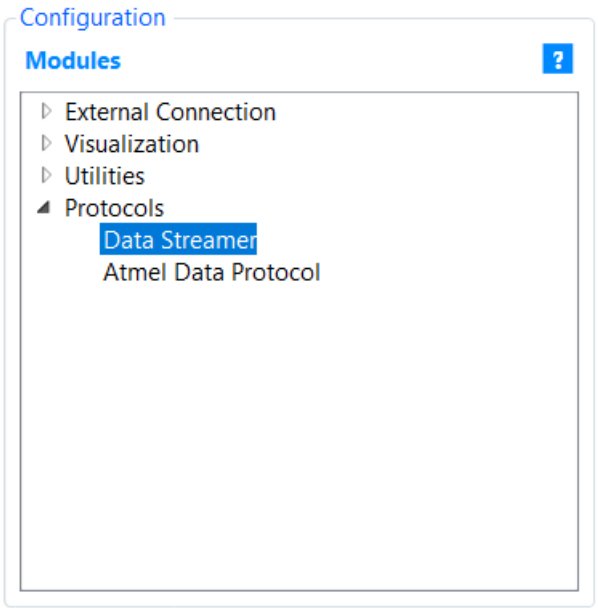


2. Connect to the Curiosity Nano COMn port, as described in the figure below.

Figure 11-2. Connect to the Desired Communication Port

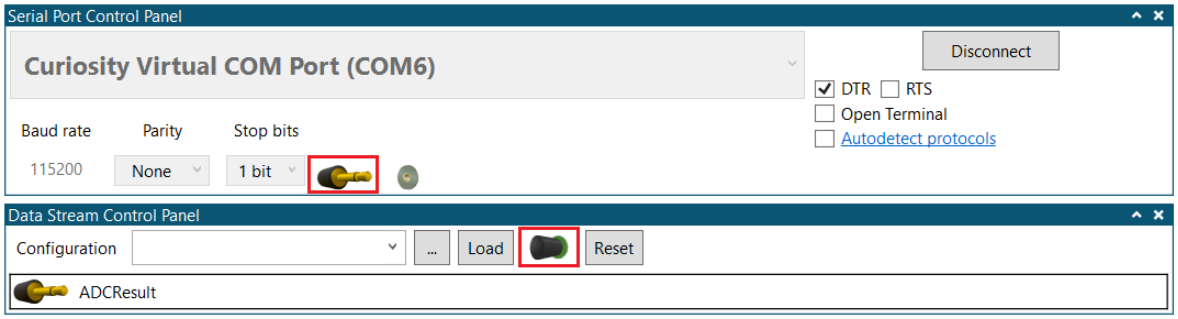
3. From *Configuration >Protocols*, open **Data Streamer** by double clicking.

Figure 11-3. Open a New Data Streamer Protocol



4. Connect the COMn port source to the Data Stream Control Panel input by dragging and dropping.

Figure 11-4. Link the Communication Port to the Data Streamer



5. Select the Configuration file and click **Load**.

Figure 11-5. Browse for the Configuration File

Figure 11-6. Select the Configuration File

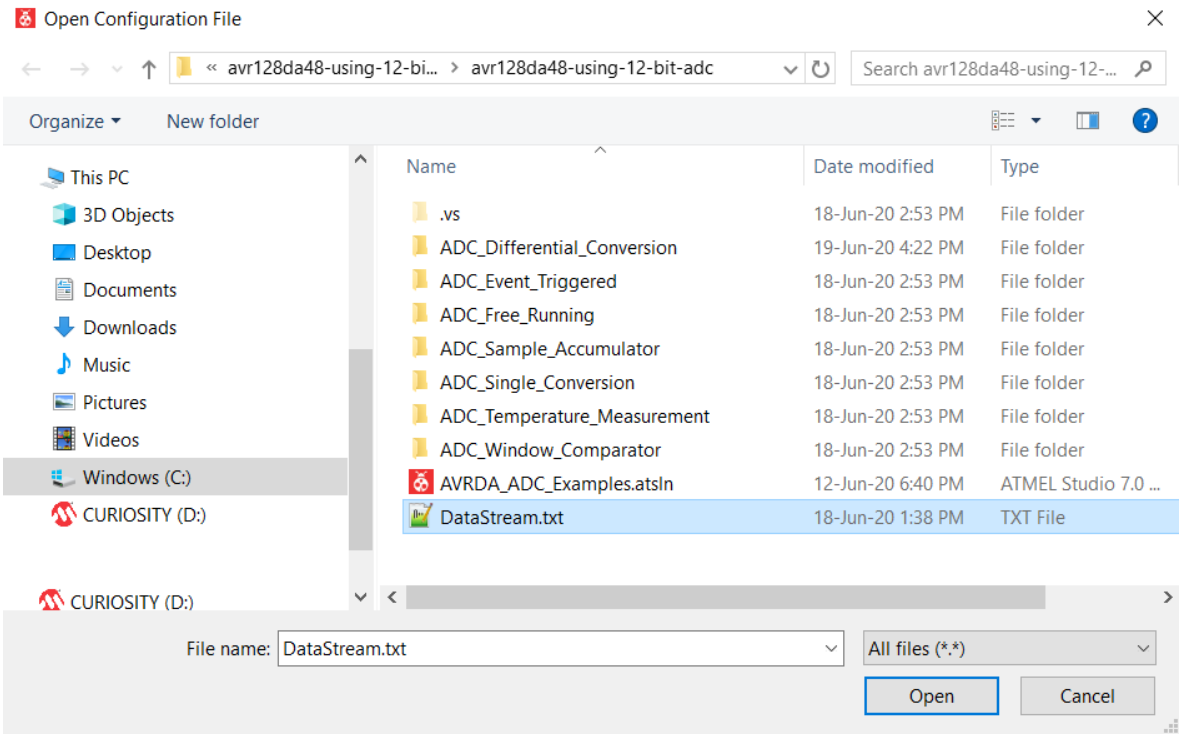
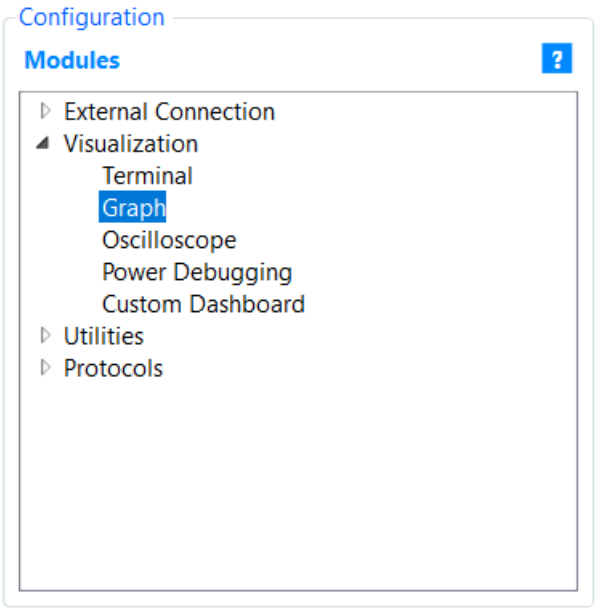


Figure 11-7. Load the Configuration File

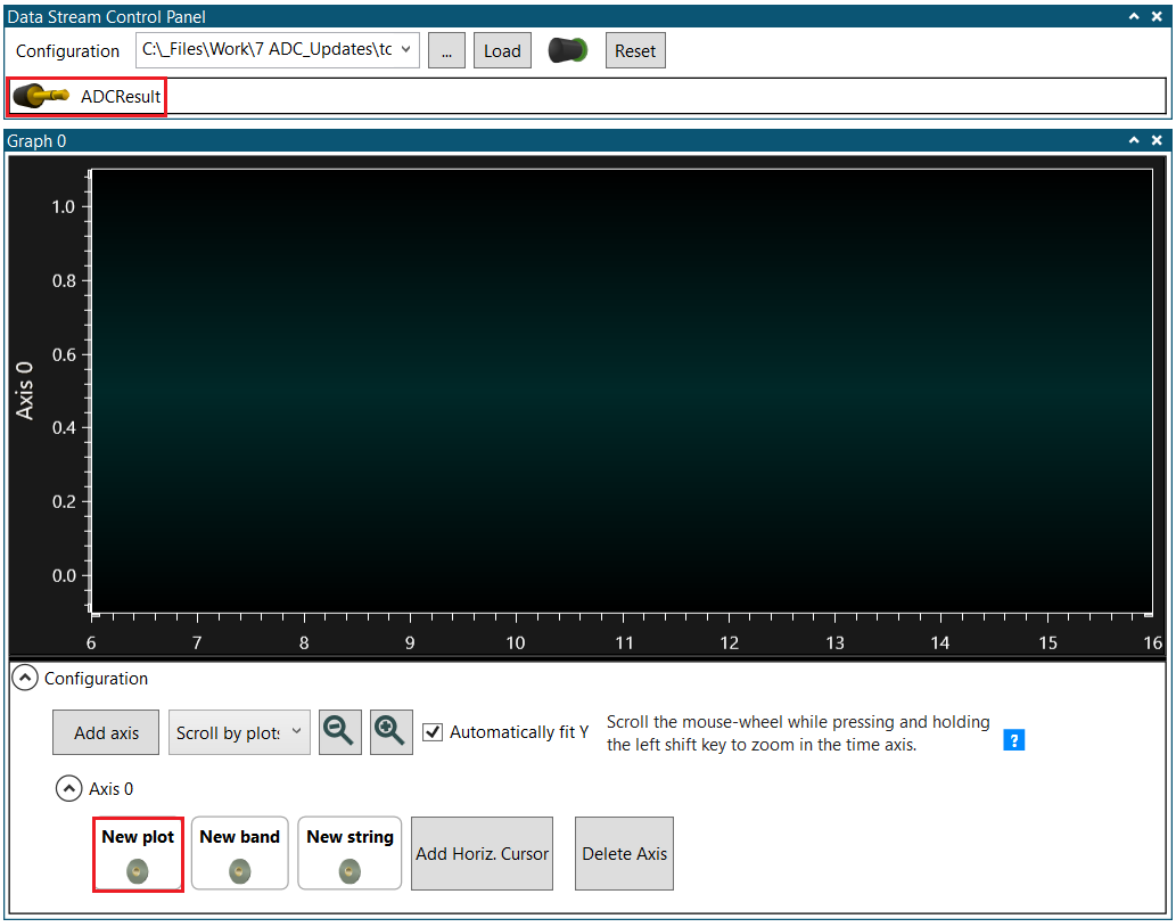
6. From *Configuration > Visualization*, open a new **Graph** by double clicking.

Figure 11-8. Open a New Graph



7. Connect the ADC Result source to the Graph: *Configuration >Axis 0 > New plot* input by dragging and dropping.

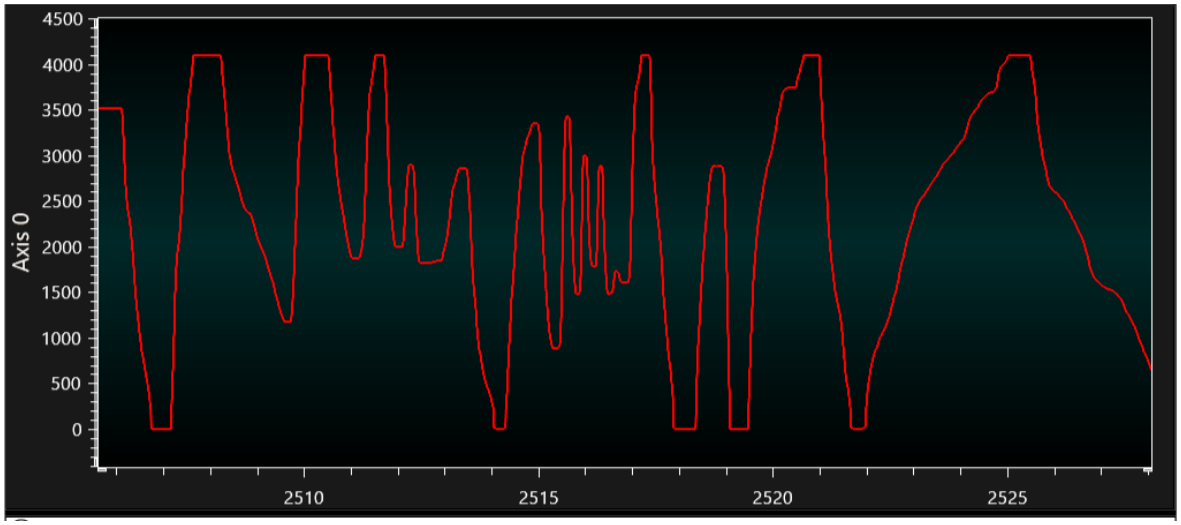
Figure 11-9. Link the ADC Result to the Plot on Graph



The results for all the use cases are presented in the figures below. The results were obtained by randomly turning the potentiometers rotors.

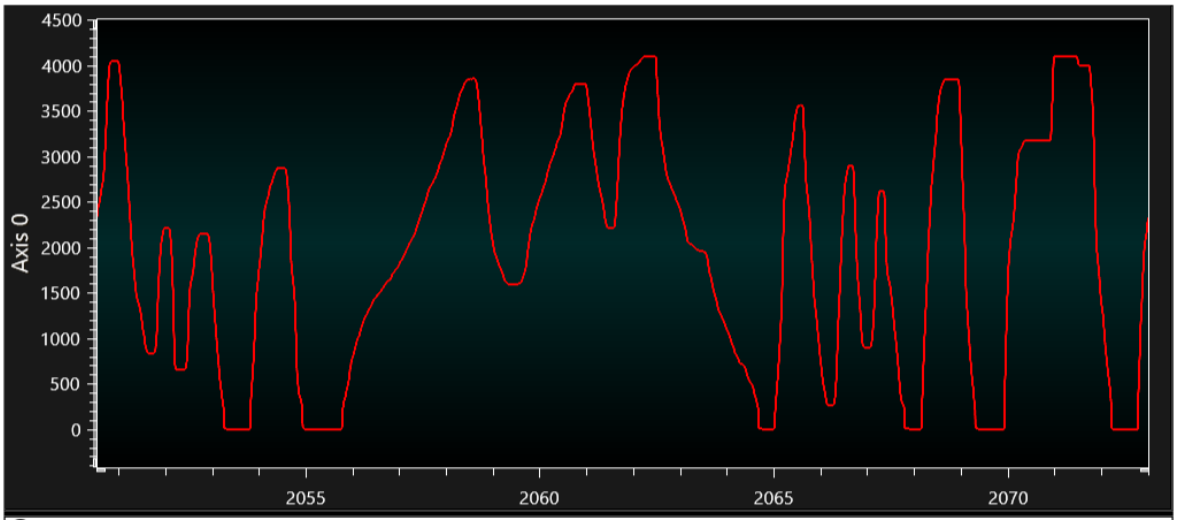
The ADC Single Conversion use case results are plotted according to the figure below. The results are in the range [0; 4095], as expected for a 12-bit resolution value.

Figure 11-10. ADC Single Conversion



The ADC Free-Running use case results are presented below. They look similar to the Single Conversion results.

Figure 11-11. ADC Free-Running Conversion



The ADC Differential Conversion results are presented below. The results are in the range [-2048; +2047], as expected for a 12-bit signed value.

Figure 11-12. ADC Differential Conversion

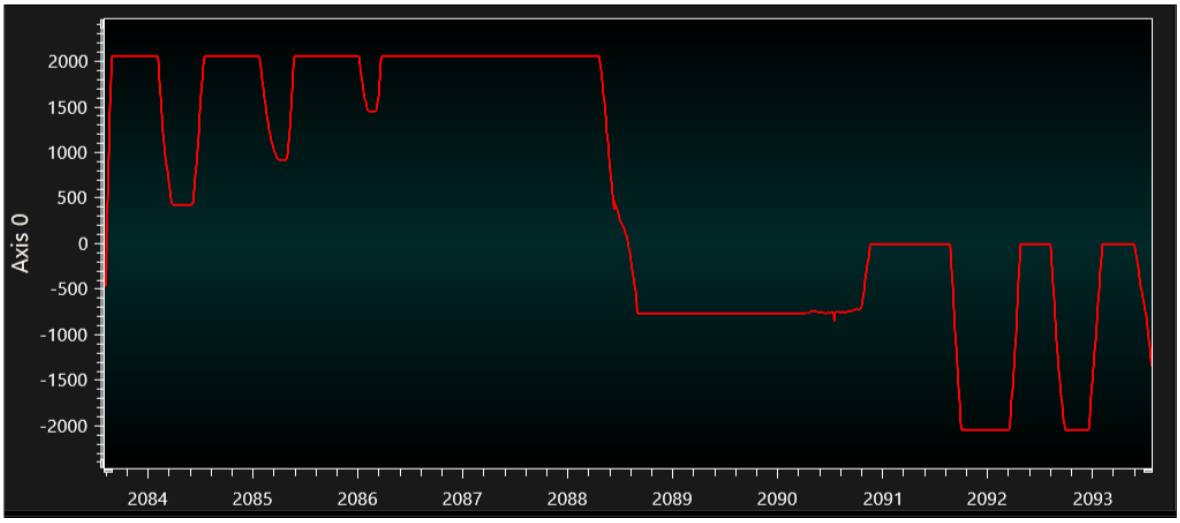
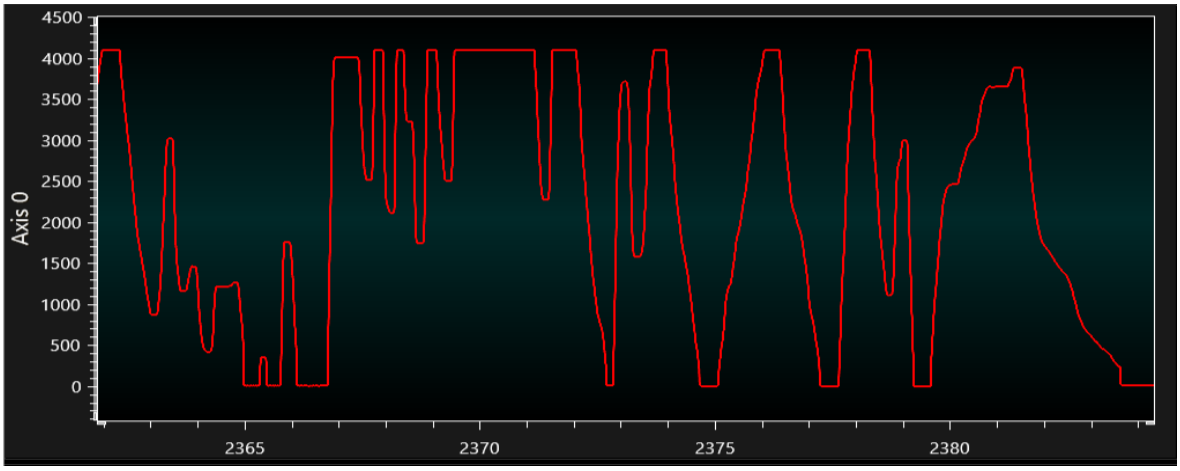


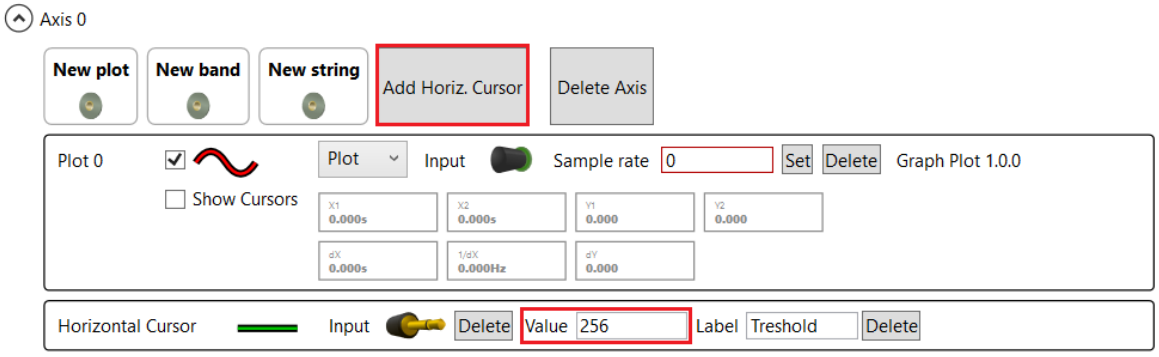
Figure 11-13. ADC Sample Accumulator



For the ADC Window Comparator use case, a horizontal cursor can be added to measure the desired threshold. The step is described in the figure below.

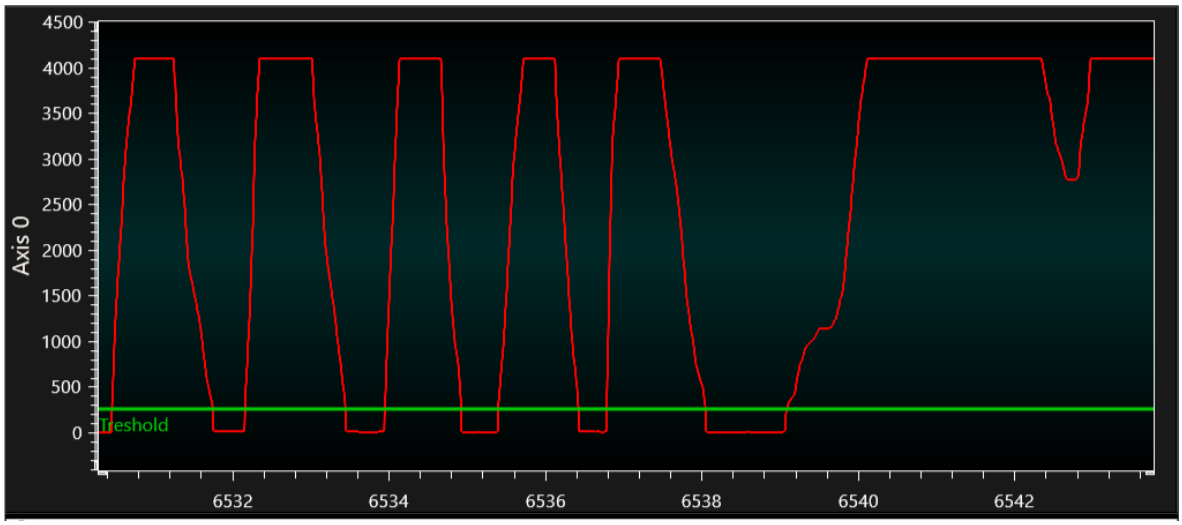
In this use case, rotating the potentiometer will modify the ADC value until LED0 is turned on or off. The threshold (the value of the ADC conversion result corresponding to the LED0 changing state) is highlighted by the horizontal cursor.

Figure 11-14. Add Horizontal Cursor



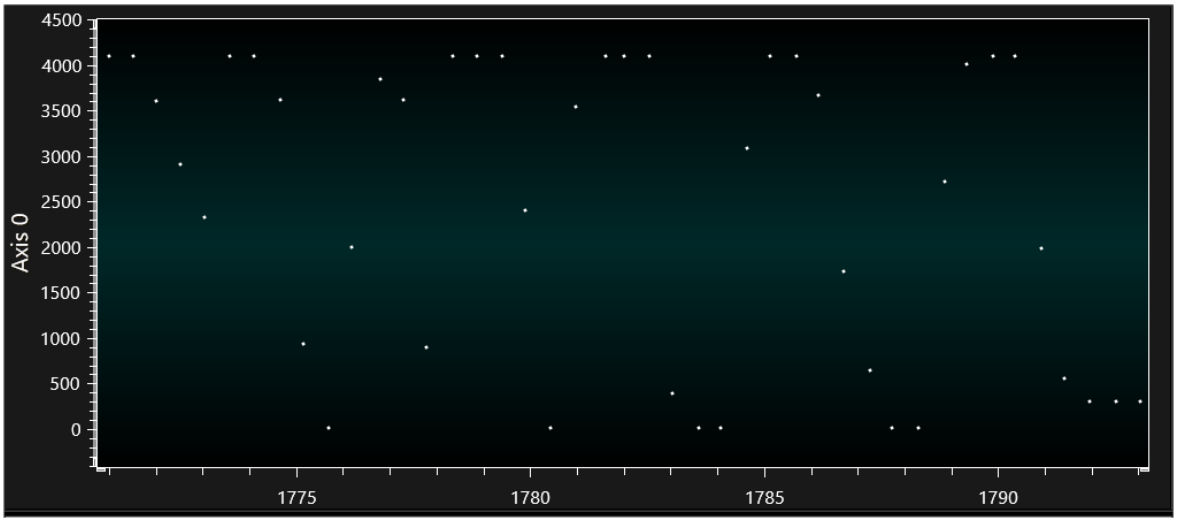
The plot is visible in the figure below.

Figure 11-15. ADC Window Comparator



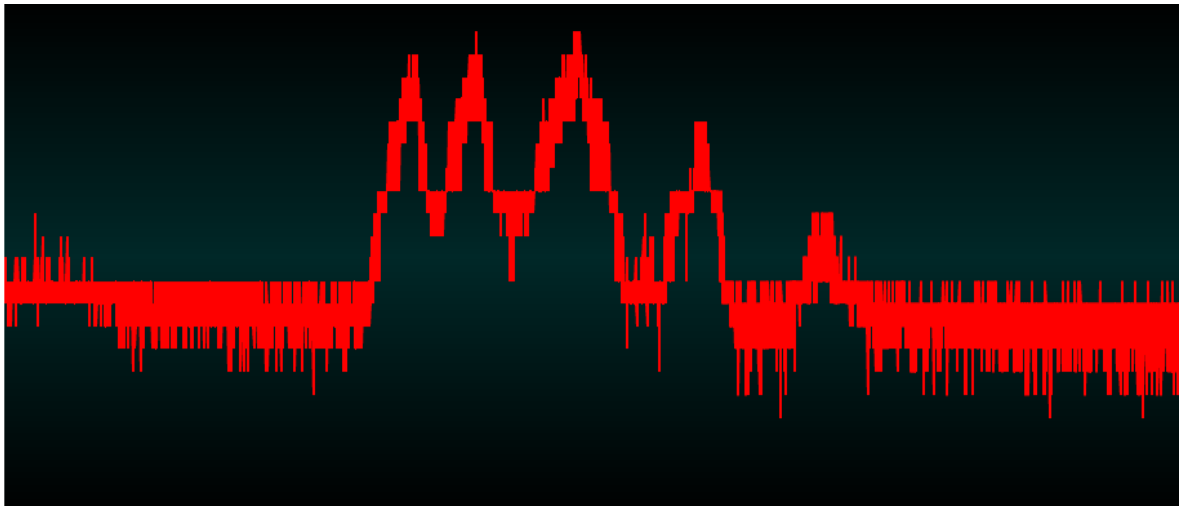
The ADC Event Triggered use case results are presented in the figure below.

Figure 11-16. ADC Event Triggered



The ADC Temperature Measurement use case results are presented in the figure below. To obtain these results, small temperature variations were introduced in the setup environment.

Figure 11-17. ADC Temperature Measurement



12. References

1. AVR128DA28/32/48/64 Preliminary Data Sheet.
2. AVR128DA48 Curiosity Nano User's Guide.
3. Data Visualizer Software User's Guide

13. Appendix

Example 13-1. ADC Single Conversion Code Example

```
/*
    \file   main.c
    \brief  ADC Single Conversion
    (c) 2019 Microchip Technology Inc. and its subsidiaries.
    Subject to your compliance with these terms, you may use Microchip
software and any
    derivatives exclusively with Microchip products. It is your responsibility
to comply with third-party
    license terms applicable to your use of third-party software (including
open source software) that
    may accompany Microchip software.
    THIS SOFTWARE IS SUPPLIED BY MICROCHIP "AS IS". NO WARRANTIES, WHETHER
    EXPRESS, IMPLIED OR STATUTORY, APPLY TO THIS SOFTWARE, INCLUDING ANY
    IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY, AND FITNESS
    FOR A PARTICULAR PURPOSE.
    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 SOFTWARE, 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 THIS SOFTWARE WILL NOT EXCEED THE AMOUNT
    OF FEES, IF ANY, THAT YOU HAVE PAID DIRECTLY TO MICROCHIP FOR THIS
    SOFTWARE.
*/
#define F_CPU           4000000UL   /* Main clock frequency */
#define START_TOKEN     0x03        /* Start Frame Token */
#define END_TOKEN       0xFC        /* End Frame Token */
/* Compute the baud rate */
#define USART1_BAUD_RATE(BAUD_RATE) (((float)F_CPU * 64 / (16 *
(float)BAUD_RATE)) + 0.5)
#include <avr/io.h>
#include <stdbool.h>
#include <avr/cpufunc.h>
void CLKCTRL_init(void);
void PORT_init(void);
void VREF0_init(void);
void ADC0_init(void);
void USART1_init(void);
uint16_t ADC0_read(void);
void ADC0_start(void);
void USART1_Write(const uint8_t data);
/* This function initializes the CLKCTRL module */
void CLKCTRL_init(void)
{
    /* FREQSEL 4M */
    ccp_write_io((void*)&(CLKCTRL.OSCHFCTRLA), (CLKCTRL.OSCHFCTRLA |
CLKCTRL_FREQSEL_4M_gc));
}
/* This function initializes the PORT module */
void PORT_init(void)
{
    /* Configure PC0 as output for USART1 TX */
    PORTC.DIRSET = PIN0_bm;

    /* Disable interrupt and digital input buffer on PD3 */
    PORTD.PIN3CTRL &= ~PORT_ISC_gm;
    PORTD.PIN3CTRL |= PORT_ISC_INPUT_DISABLE_gc;

    /* Disable pull-up resistor */
    PORTD.PIN3CTRL &= ~PORT_PULLUPEN_bm;
}
/* This function initializes the VREF module */
void VREF0_init(void)
{
    VREF.ADC0REF = VREF_REFSEL_2V048_gc;  /* Internal 2.048V reference */
}
/* This function initializes the ADC module */
void ADC0_init(void)
```

```
{
    ADC0.CTRLC = ADC_PRESC_DIV4_gc;         /* CLK_PER divided by 4 */
    ADC0.CTRLA = ADC_ENABLE_bm              /* ADC Enable: enabled */
               | ADC_RESSEL_12BIT_gc;       /* 12-bit mode */
    ADC0.MUXPOS = ADC_MUXPOS_AIN3_gc;       /* Select ADC channel AIN3 <-> PD3
*/
}
/* This function initializes the USART module */
void USART1_init(void)
{
    /* Configure the baud rate: 115200 */
    USART1.BAUD = (uint16_t)USART1_BAUD_RATE(115200);
    USART1.CTRLB = USART_TXEN_bm;           /* Enable TX */
    USART1.CTRLC = USART_CHSIZE_8BIT_gc;    /* Configure character size: 8 bit
*/
}
/* This function returns the ADC conversion result */
uint16_t ADC0_read(void)
{
    /* Wait for ADC result to be ready */
    while (!(ADC0.INTFLAGS & ADC_RESRDY_bm));
    /* Clear the interrupt flag by reading the result */
    return ADC0.RES;
}
/* This function starts the ADC conversions*/
void ADC0_start(void)
{
    /* Start ADC conversion */
    ADC0.COMMAND = ADC_STCONV_bm;
}
/* This function transmits one byte through USART */
void USART1_Write(const uint8_t data)
{
    /* Check if USART buffer is ready to transmit data */
    while (!(USART1.STATUS & USART_DREIF_bm));
    /* Transmit data using TXDATAL register */
    USART1.TXDATAL = data;
}
int main(void)
{
    uint16_t adcVal;

    /* Initialize all peripherals */
    CLKCTRL_init();
    PORT_init();
    VREF0_init();
    ADC0_init();
    USART1_init();

    while (1)
    {
        /* Start the ADC conversion */
        ADC0_start();

        /* Read the ADC result */
        adcVal = ADC0_read();

        /* Transmit the ADC result to be plotted using Data Visualizer */
        USART1_Write(START_TOKEN);
        USART1_Write(adcVal & 0x00FF);
        USART1_Write(adcVal >> 8);
        USART1_Write(END_TOKEN);
    }
}
```

Example 13-2. ADC Free-Running Conversion Code Example

```
/*
    \file   main.c
    \brief  ADC Free-Running
    (c) 2019 Microchip Technology Inc. and its subsidiaries.
    Subject to your compliance with these terms, you may use Microchip
software and any
```

```
    derivatives exclusively with Microchip products. It is your responsibility
to comply with third-party
    license terms applicable to your use of third-party software (including
open source software) that
    may accompany Microchip software.
    THIS SOFTWARE IS SUPPLIED BY MICROCHIP "AS IS". NO WARRANTIES, WHETHER
    EXPRESS, IMPLIED OR STATUTORY, APPLY TO THIS SOFTWARE, INCLUDING ANY
    IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY, AND FITNESS
    FOR A PARTICULAR PURPOSE.
    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 SOFTWARE, 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 THIS SOFTWARE WILL NOT EXCEED THE AMOUNT
    OF FEES, IF ANY, THAT YOU HAVE PAID DIRECTLY TO MICROCHIP FOR THIS
    SOFTWARE.
*/
#define F_CPU           4000000UL   /* Main clock frequency */
#define START_TOKEN     0x03        /* Start Frame Token */
#define END_TOKEN       0xFC        /* End Frame Token */
/* Compute the baud rate */
#define USART1_BAUD_RATE(BAUD_RATE) (((float)F_CPU * 64 / (16 *
(float)BAUD_RATE)) + 0.5)
#include <avr/io.h>
#include <stdbool.h>
#include <avr/cpufunc.h>
void CLKCTRL_init(void);
void PORT_init(void);
void VREF0_init(void);
void ADC0_init(void);
void USART1_init(void);
uint16_t ADC0_read(void);
void ADC0_start(void);
void USART1_Write(const uint8_t data);
/* This function initializes the CLKCTRL module */
void CLKCTRL_init(void)
{
    /* FREQSEL 4M */
    ccp_write_io((void*)&(CLKCTRL.OSCHFCTRLA), (CLKCTRL.OSCHFCTRLA |
CLKCTRL_FREQSEL_4M_gc));
}
/* This function initializes the PORT module */
void PORT_init(void)
{
    /* Configure PC0 as output for USART1 TX */
    PORTC.DIRSET = PIN0_bm;

    /* Disable interrupt and digital input buffer on PD3 */
    PORTD.PIN3CTRL &= ~PORT_ISC_gm;
    PORTD.PIN3CTRL |= PORT_ISC_INPUT_DISABLE_gc;
    /* Disable pull-up resistor */
    PORTD.PIN3CTRL &= ~PORT_PULLUPEN_bm;
}
/* This function initializes the VREF module */
void VREF0_init(void)
{
    VREF.ADC0REF = VREF_REFSEL_2V048_gc;  /* Internal 2.048V reference */
}
/* This function initializes the ADC module */
void ADC0_init(void)
{
    ADC0.CTRLC = ADC_PRESC_DIV4_gc;        /* CLK_PER divided by 4 */
    ADC0.CTRLA = ADC_ENABLE_bm             /* ADC Enable: enabled */
               | ADC_RESSEL_12BIT_gc       /* 12-bit mode */
               | ADC_FREERUN_bm;           /* Enable Free-Run mode */
    ADC0.MUXPOS = ADC_MUXPOS_AIN3_gc;      /* Select ADC channel AIN3 <-> PD3
*/
}
/* This function initializes the USART module */
void USART1_init(void)
{
    /* Configure the baud rate: 115200 */
    USART1.BAUD = (uint16_t)USART1_BAUD_RATE(115200);
    USART1.CTRLB = USART_TXEN_bm;           /* Enable TX */
    USART1.CTRLC = USART_CHSIZE_8BIT_gc;    /* Configure character size: 8 bit
```

```
*/
}
/* This function returns the ADC conversion result */
uint16_t ADC0_read(void)
{
    /* Wait for ADC result to be ready */
    while (!(ADC0.INTFLAGS & ADC_RESRDY_bm));
    /* Clear the interrupt flag by reading the result */
    return ADC0.RES;
}
/* This function starts the ADC conversions*/
void ADC0_start(void)
{
    /* Start ADC conversion */
    ADC0.COMMAND = ADC_STCONV_bm;
}
/* This function transmits one byte through USART */
void USART1_Write(const uint8_t data)
{
    /* Check if USART buffer is ready to transmit data */
    while (!(USART1.STATUS & USART_DREIF_bm));
    /* Transmit data using TXDATAL register */
    USART1.TXDATAL = data;
}
int main(void)
{
    uint16_t adcVal;

    /* Initialize all peripherals */
    CLKCTRL_init();
    PORT_init();
    VREF0_init();
    ADC0_init();
    USART1_init();

    /* Start the ADC conversions */
    ADC0_start();

    while (1)
    {
        /* Read the ADC result */
        adcVal = ADC0_read();

        /* Transmit the ADC result to be plotted using Data Visualizer */
        USART1_Write(START_TOKEN);
        USART1_Write(adcVal & 0x00FF);
        USART1_Write(adcVal >> 8);
        USART1_Write(END_TOKEN);
    }
}
```

Example 13-3. ADC Differential Conversion Code Example

```
/*
    \file   main.c
    \brief  ADC Differential Conversion
    (c) 2019 Microchip Technology Inc. and its subsidiaries.
    Subject to your compliance with these terms, you may use Microchip
software and any
    derivatives exclusively with Microchip products. It is your responsibility
to comply with third-party
    license terms applicable to your use of third-party software (including
open source software) that
    may accompany Microchip software.
    THIS SOFTWARE IS SUPPLIED BY MICROCHIP "AS IS". NO WARRANTIES, WHETHER
    EXPRESS, IMPLIED OR STATUTORY, APPLY TO THIS SOFTWARE, INCLUDING ANY
    IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY, AND FITNESS
    FOR A PARTICULAR PURPOSE.
    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 SOFTWARE, 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 THIS SOFTWARE WILL NOT EXCEED THE AMOUNT
    OF FEES, IF ANY, THAT YOU HAVE PAID DIRECTLY TO MICROCHIP FOR THIS
    SOFTWARE.
*/
#define F_CPU           4000000UL   /* Main clock frequency */
#define START_TOKEN     0x03        /* Start Frame Token */
#define END_TOKEN       0xFC        /* End Frame Token */
/* Compute the baud rate */
#define USART1_BAUD_RATE(BAUD_RATE) (((float)F_CPU * 64 / (16 *
(float)BAUD_RATE)) + 0.5)
#include <avr/io.h>
#include <stdbool.h>
#include <avr/cpufunc.h>
int16_t adcVal;
void CLKCTRL_init(void);
void PORT_init(void);
void VREF0_init(void);
void ADC0_init(void);
void USART1_init(void);
void ADC0_start(void);
bool ADC0_conversionDone(void);
int16_t ADC0_read(void);
void USART1_Write(const uint8_t data);
void SYSTEM_init(void);
void CLKCTRL_init(void)
{
    /* FREQSEL 4M */
    ccp_write_io((void*)&(CLKCTRL.OSCHFCTRLA), (CLKCTRL.OSCHFCTRLA |
CLKCTRL_FREQSEL_4M_gc));
}
void PORT_init(void)
{
    /* Configure PC0 as output for USART1 TX */
    PORTC.DIRSET = PIN0_bm;

    /* Disable interrupt and digital input buffer on PD3 */
    PORTD.PIN3CTRL &= ~PORT_ISC_gm;
    PORTD.PIN3CTRL |= PORT_ISC_INPUT_DISABLE_gc;
    /* Disable interrupt and digital input buffer on PD4 */
    PORTD.PIN4CTRL &= ~PORT_ISC_gm;
    PORTD.PIN4CTRL |= PORT_ISC_INPUT_DISABLE_gc;

    /* Disable pull-up resistor */
    PORTD.PIN3CTRL &= ~PORT_PULLUPEN_bm;
    PORTD.PIN4CTRL &= ~PORT_PULLUPEN_bm;
}
void VREF0_init(void)
{
    VREF.ADC0REF = VREF_REFSEL_2V048_gc;  /* Internal 2.048V reference */
}
void ADC0_init(void)
{
    ADC0.CTRLC = ADC_PRESC_DIV4_gc;       /* CLK_PER divided by 4 */
    ADC0.CTRLA = ADC_ENABLE_bm            /* ADC Enable: enabled */
               | ADC_RESSEL_12BIT_gc      /* 12-bit mode */
               | ADC_CONVMODE_bm          /* Differential Conversion */
               | ADC_FREERUN_bm;          /* Enable Free-Run mode */
    ADC0.MUXPOS = ADC_MUXPOS_AIN3_gc;     /* Select ADC channel AIN3 <-> PD3 */
    ADC0.MUXNEG = ADC_MUXNEG_AIN4_gc;     /* Select ADC channel AIN4 <-> PD4 */
}
void USART1_init(void)
{
    /* Configure the baud rate: 115200 */
    USART1.BAUD = (uint16_t)USART1_BAUD_RATE(115200);
    USART1.CTRLB = USART_TXEN_bm;           /* Enable TX */
    USART1.CTRLC = USART_CHSIZE_8BIT_gc;    /* Configure character size: 8 bit
*/
}
int16_t ADC0_read(void)
{
    /* Clear the interrupt flag by reading the result */
    return ADC0.RES;
}
void ADC0_start(void)
{
    /* Start conversion */
```

```
    ADC0.COMMAND = ADC_STCONV_bm;
}
bool ADC0_conversionDone(void)
{
    /* Check if the conversion is done  */
    return (ADC0.INTFLAGS & ADC_RESRDY_bm);
}
void USART1_Write(const uint8_t data)
{
    /* Check if USART buffer is ready to transmit data */
    while (!(USART1.STATUS & USART_DREIF_bm));
    /* Transmit data using TXDATAL register */
    USART1.TXDATAL = data;
}
void SYSTEM_init(void)
{
    CLKCTRL_init();
    PORT_init();
    VREF0_init();
    ADC0_init();
    USART1_init();
}
int main(void)
{
    SYSTEM_init();
    ADC0_start();

    while (1)
    {
        if (ADC0_conversionDone())
        {
            /* Read the ADC result */
            adcVal = ADC0_read();
            /* Transmit the ADC result to be plotted using Data Visualizer */
            USART1_Write(START_TOKEN);
            USART1_Write(adcVal & 0x00FF);
            USART1_Write(adcVal >> 8);
            USART1_Write(END_TOKEN);
        }
    }
}
```

Example 13-4. ADC Sample Accumulator Code Example

```
/*
    \file   main.c
    \brief  ADC Sample Accumulator
    (c) 2019 Microchip Technology Inc. and its subsidiaries.
    Subject to your compliance with these terms, you may use Microchip
software and any
    derivatives exclusively with Microchip products. It is your responsibility
to comply with third-party
    license terms applicable to your use of third-party software (including
open source software) that
    may accompany Microchip software.
    THIS SOFTWARE IS SUPPLIED BY MICROCHIP "AS IS". NO WARRANTIES, WHETHER
    EXPRESS, IMPLIED OR STATUTORY, APPLY TO THIS SOFTWARE, INCLUDING ANY
    IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY, AND FITNESS
    FOR A PARTICULAR PURPOSE.
    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 SOFTWARE, 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 THIS SOFTWARE WILL NOT EXCEED THE AMOUNT
    OF FEES, IF ANY, THAT YOU HAVE PAID DIRECTLY TO MICROCHIP FOR THIS
    SOFTWARE.
*/
#define ADC_SHIFT_DIV16     (4)         /* where 4 is 2^4 = 16 */
#define F_CPU               4000000UL   /* Main clock frequency */
#define START_TOKEN         0x03        /* Start Frame Token */
#define END_TOKEN           0xFC        /* End Frame Token */
```

```
/* Compute the baud rate */
#define USART1_BAUD_RATE(BAUD_RATE) (((float)F_CPU * 64 / (16 *
(float)BAUD_RATE)) + 0.5)
#include <avr/io.h>
#include <stdbool.h>
#include <avr/cpufunc.h>
void CLKCTRL_init(void);
void PORT_init(void);
void VREF0_init(void);
void ADC0_init(void);
void USART1_init(void);
uint16_t ADC0_read(void);
void ADC0_start(void);
void USART1_Write(const uint8_t data);
/* This function initializes the CLKCTRL module */
void CLKCTRL_init(void)
{
    /* FREQSEL 4M */
    ccp_write_io((void*)&(CLKCTRL.OSCHFCTRLA), (CLKCTRL.OSCHFCTRLA |
CLKCTRL_FREQSEL_4M_gc));
}
/* This function initializes the PORT module */
void PORT_init(void)
{
    /* Configure PC0 as output for USART1 TX */
    PORTC.DIRSET = PIN0_bm;

    /* Disable interrupt and digital input buffer on PD3 */
    PORTD.PIN3CTRL &= ~PORT_ISC_gm;
    PORTD.PIN3CTRL |= PORT_ISC_INPUT_DISABLE_gc;

    /* Disable pull-up resistor */
    PORTD.PIN3CTRL &= ~PORT_PULLUPEN_bm;
}
/* This function initializes the VREF module */
void VREF0_init(void)
{
    VREF.ADC0REF = VREF_REFSEL_2V048_gc;  /* Internal 2.048V reference */
}
/* This function initializes the ADC module */
void ADC0_init(void)
{
    ADC0.CTRLC = ADC_PRESC_DIV4_gc;        /* CLK_PER divided by 4 */
    ADC0.CTRLA = ADC_ENABLE_bm             /* ADC Enable: enabled */
               | ADC_RESSEL_12BIT_gc       /* 12-bit mode */
               | ADC_FREERUN_bm;           /* Enable Free-Run mode */
    ADC0.MUXPOS = ADC_MUXPOS_AIN3_gc;      /* Select ADC channel AIN3 <-> PD3
*/
    ADC0.CTRLB = ADC_SAMPNUM_ACC64_gc;     /* Set to accumulate 64 samples */
}
/* This function initializes the USART module */
void USART1_init(void)
{
    /* Configure the baud rate: 115200 */
    USART1.BAUD = (uint16_t)USART1_BAUD_RATE(115200);
    USART1.CTRLB = USART_TXEN_bm;           /* Enable TX */
    USART1.CTRLC = USART_CHSIZE_8BIT_gc;    /* Configure character size: 8 bit
*/
}
/* This function returns the ADC conversion result */
uint16_t ADC0_read(void)
{
    /* Wait for ADC result to be ready */
    while (!(ADC0.INTFLAGS & ADC_RESRDY_bm));
    /* Clear the interrupt flag by reading the result */
    return ADC0.RES;
}
/* This function starts the ADC conversions*/
void ADC0_start(void)
{
    /* Start ADC conversion */
    ADC0.COMMAND = ADC_STCONV_bm;
}
/* This function transmits one byte through USART */
void USART1_Write(const uint8_t data)
{
```

```
    /* Check if USART buffer is ready to transmit data */
    while (!(USART1.STATUS & USART_DREIF_bm));
    /* Transmit data using TXDATAL register */
    USART1.TXDATAL = data;
}
int main(void)
{
    uint16_t adcVal;

    /* Initialize all peripherals */
    CLKCTRL_init();
    PORT_init();
    VREF0_init();
    ADC0_init();
    USART1_init();

    /* Start the ADC conversions */
    ADC0_start();

    while (1)
    {
        /* Read the ADC result */
        adcVal = ADC0_read();
        /* divide by No of samples or 16, if No. samples > 16 */
        adcVal = adcVal >> ADC_SHIFT_DIV16;
        /* Transmit the ADC result to be plotted using Data Visualizer */
        USART1_Write(START_TOKEN);
        USART1_Write(adcVal & 0x00FF);
        USART1_Write(adcVal >> 8);
        USART1_Write(END_TOKEN);
    }
}
```

Example 13-5. ADC Window Comparator Code Example

```
/*
    \file   main.c

    \brief  ADC Window Comparator

    (c) 2019 Microchip Technology Inc. and its subsidiaries.

    Subject to your compliance with these terms, you may use Microchip
software and any
    derivatives exclusively with Microchip products. It is your responsibility
to comply with third-party
    license terms applicable to your use of third-party software (including
open source software) that
    may accompany Microchip software.

    THIS SOFTWARE IS SUPPLIED BY MICROCHIP "AS IS". NO WARRANTIES, WHETHER
    EXPRESS, IMPLIED OR STATUTORY, APPLY TO THIS SOFTWARE, INCLUDING ANY
    IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY, AND FITNESS
    FOR A PARTICULAR PURPOSE.

    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 SOFTWARE, 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 THIS SOFTWARE WILL NOT EXCEED THE AMOUNT
    OF FEES, IF ANY, THAT YOU HAVE PAID DIRECTLY TO MICROCHIP FOR THIS
    SOFTWARE.
*/

#define WINDOW_CMP_LOW_TH_EXAMPLE       (0x100)
#define F_CPU                           4000000UL   /* Main clock frequency */
#define START_TOKEN                     0x03        /* Start Frame Token */
#define END_TOKEN                       0xFC        /* End Frame Token */
/* Compute the baud rate */
#define USART1_BAUD_RATE(BAUD_RATE) (((float)F_CPU * 64 / (16 *
(float)BAUD_RATE)) + 0.5)
```

```
#include <avr/io.h>
#include <avr/cpufunc.h>

void CLKCTRL_init(void);
void PORT_init(void);
void VREF0_init(void);
void ADC0_init(void);
void USART1_init(void);
uint16_t ADC0_read(void);
uint8_t ADC0_resultReady(void);
void ADC0_start(void);
uint8_t ADC0_resultBelowTreshold(void);
void USART1_Write(const uint8_t data);
void LED0_init(void);
void LED0_on(void);
void LED0_off(void);

/* This function initializes the CLKCTRL module */
void CLKCTRL_init(void)
{
    /* FREQSEL 4M */
    ccp_write_io((void*)&(CLKCTRL.OSCHFCTRLA), (CLKCTRL.OSCHFCTRLA |
CLKCTRL_FREQSEL_4M_gc));
}

/* This function initializes the PORT module */
void PORT_init(void)
{
    /* Configure PC0 as output for USART1 TX */
    PORTC.DIRSET = PIN0_bm;

    /* Disable interrupt and digital input buffer on PD3 */
    PORTD.PIN3CTRL &= ~PORT_ISC_gm;
    PORTD.PIN3CTRL |= PORT_ISC_INPUT_DISABLE_gc;

    /* Disable pull-up resistor */
    PORTD.PIN3CTRL &= ~PORT_PULLUPEN_bm;
}

/* This function initializes the VREF module */
void VREF0_init(void)
{
    VREF.ADC0REF = VREF_REFSEL_2V048_gc;    /* Internal 2.048V reference */
}

/* This function initializes the ADC module */
void ADC0_init(void)
{
    ADC0.CTRLC = ADC_PRESC_DIV4_gc;         /* CLK_PER divided by 4 */
    ADC0.CTRLA = ADC_ENABLE_bm              /* ADC Enable: enabled */
                | ADC_RESSEL_12BIT_gc       /* 12-bit mode */
                | ADC_FREERUN_bm;           /* Free-run mode */
    ADC0.MUXPOS = ADC_MUXPOS_AIN3_gc;       /* Select ADC channel AIN3 <-> PD3
*/
    ADC0.WINLT = WINDOW_CMP_LOW_TH_EXAMPLE; /* Set conversion window
comparator low threshold */
    ADC0.CTRLE = ADC_WINCM_BELOW_gc;        /* Set conversion window mode */
}

/* This function initializes the USART module */
void USART1_init(void)
{
    /* Configure the baud rate: 115200 */
    USART1.BAUD = (uint16_t)USART1_BAUD_RATE(115200);
    USART1.CTRLB = USART_TXEN_bm;           /* Enable TX */
    USART1.CTRLC = USART_CHSIZE_8BIT_gc;    /* Configure character size: 8 bit
*/
}

/* This function returns the ADC conversion result */
uint16_t ADC0_read(void)
{
    /* Clear the interrupt flag by reading the result */
    return ADC0.RES;
}
```

```
uint8_t ADC0_resultReady(void)
{
    return (ADC0.INTFLAGS & ADC_RESRDY_bm);
}

/* This function starts the ADC conversions*/
void ADC0_start(void)
{
    /* Start conversion */
    ADC0.COMMAND = ADC_STCONV_bm;
}

uint8_t ADC0_resultBelowTreshold(void)
{
    return (ADC0.INTFLAGS & ADC_WCMP_bm);
}

/* This function transmits one byte through USART */
void USART1_Write(const uint8_t data)
{
    /* Check if USART buffer is ready to transmit data */
    while (!(USART1.STATUS & USART_DREIF_bm));
    /* Transmit data using TXDATAL register */
    USART1.TXDATAL = data;
}

/* This function initializes the LED pin */
void LED0_init(void)
{
    /* Configure the pin as output */
    PORTC.DIRSET = PIN6_bm;
    /* Configure the output high (LED == OFF) */
    PORTC.OUTSET = PIN6_bm;
}

void LED0_on(void)
{
    /* Configure the output low (LED == ON) */
    PORTC.OUTCLR = PIN6_bm;
}

void LED0_off(void)
{
    /* Configure the output high (LED == OFF) */
    PORTC.OUTSET = PIN6_bm;
}

int main(void)
{
    uint16_t adcVal;

    /* Initialize all peripherals */
    CLKCTRL_init();
    PORT_init();
    VREF0_init();
    ADC0_init();
    USART1_init();
    LED0_init();

    /* Start the ADC conversion */
    ADC0_start();

    while (1)
    {
        while (!ADC0_resultReady());

        if (ADC0_resultBelowTreshold())
        {
            LED0_on();
        }
        else
        {
            LED0_off();
        }

        adcVal = ADC0_read();
```

```
        /* Transmit the ADC result to be plotted using Data Visualizer */
        USART1_Write(START_TOKEN);
        USART1_Write(adcVal & 0x00FF);
        USART1_Write(adcVal >> 8);
        USART1_Write(END_TOKEN);
    }
}
```

Example 13-6. ADC Event Triggered Code Example

```
/*
    \file   main.c
    \brief  ADC Event Triggered by RTC Overflow
    (c) 2019 Microchip Technology Inc. and its subsidiaries.
    Subject to your compliance with these terms, you may use Microchip
software and any
    derivatives exclusively with Microchip products. It is your responsibility
to comply with third-party
    license terms applicable to your use of third-party software (including
open source software) that
    may accompany Microchip software.
    THIS SOFTWARE IS SUPPLIED BY MICROCHIP "AS IS". NO WARRANTIES, WHETHER
    EXPRESS, IMPLIED OR STATUTORY, APPLY TO THIS SOFTWARE, INCLUDING ANY
    IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY, AND FITNESS
    FOR A PARTICULAR PURPOSE.
    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 SOFTWARE, 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 THIS SOFTWARE WILL NOT EXCEED THE AMOUNT
    OF FEES, IF ANY, THAT YOU HAVE PAID DIRECTLY TO MICROCHIP FOR THIS
    SOFTWARE.
*/
/* RTC Period */
#define RTC_PERIOD      (511)       /* RTC period */
#define F_CPU           4000000UL   /* Main clock frequency */
#define START_TOKEN     0x03        /* Start Frame Token */
#define END_TOKEN       0xFC        /* End Frame Token */
/* Compute the baud rate */
#define USART1_BAUD_RATE(BAUD_RATE) (((float)F_CPU * 64 / (16 *
(float)BAUD_RATE)) + 0.5)
#include <avr/io.h>
#include <avr/interrupt.h>
#include <avr/cpufunc.h>
#include <stdbool.h>
volatile uint16_t adcVal;
volatile bool adcResultReady = 0;
void CLKCTRL_init(void);
void PORT_init(void);
void VREF0_init(void);
void ADC0_init(void);
void LED0_init(void);
void USART1_init();
void LED0_toggle(void);
void USART1_Write(const uint8_t data);
void RTC_init(void);
void EVSYS_init(void);
/* This function initializes the CLKCTRL module */
void CLKCTRL_init(void)
{
    /* FREQSEL 4M */
    ccp_write_io((void*)&(CLKCTRL.OSCHFCTRLA), (CLKCTRL.OSCHFCTRLA |
CLKCTRL_FREQSEL_4M_gc));
}
/* This function initializes the PORT module */
void PORT_init(void)
{
    /* Configure PC0 as output for USART1 TX */
    PORTC.DIRSET = PIN0_bm;
```

```
    /* Disable interrupt and digital input buffer on PD3 */
    PORTD.PIN3CTRL &= ~PORT_ISC_gm;
    PORTD.PIN3CTRL |= PORT_ISC_INPUT_DISABLE_gc;

    /* Disable pull-up resistor */
    PORTD.PIN3CTRL &= ~PORT_PULLUPEN_bm;
}
/* This function initializes the VREF module */
void VREF0_init(void)
{
    VREF.ADC0REF = VREF_REFSEL_2V048_gc;  /* Internal 2.048V reference */
}
/* This function initializes the ADC module */
void ADC0_init(void)
{
    ADC0.CTRLC = ADC_PRESC_DIV4_gc;        /* CLK_PER divided by 4 */
    ADC0.CTRLA = ADC_ENABLE_bm             /* ADC Enable: enabled */
               | ADC_RESSEL_12BIT_gc;      /* 12-bit mode */
    ADC0.MUXPOS = ADC_MUXPOS_AIN3_gc;      /* Select ADC channel AIN3 <-> PD3
*/
    ADC0.INTCTRL |= ADC_RESRDY_bm;         /* Enable interrupts */
    ADC0.EVCTRL |= ADC_STARTEI_bm;         /* Enable event triggered
conversion */
}
/* This function initializes the LED pin */
void LED0_init(void)
{
    /* Configure the pin as output */
    PORTC.DIRSET = PIN6_bm;
    /* Configure the output high (LED == OFF) */
    PORTC.OUTSET = PIN6_bm;
}
/* This function initializes the USART module */
void USART1_init()
{
    /* Configure the baud rate: 115200 */
    USART1.BAUD = (uint16_t)USART1_BAUD_RATE(115200);
    USART1.CTRLB = USART_TXEN_bm;           /* Enable TX */
    USART1.CTRLC = USART_CHSIZE_8BIT_gc;    /* Configure character size: 8 bit
*/
}
/* This function toggles the LED pin */
void LED0_toggle(void)
{
    PORTC.OUTTGL = PIN6_bm;
}
/* This function transmits one byte through USART */
void USART1_Write(const uint8_t data)
{
    /* Check if USART buffer is ready to transmit data */
    while (!(USART1.STATUS & USART_DREIF_bm));
    /* Transmit data using TXDATAL register */
    USART1.TXDATAL = data;
}
ISR(ADC0_RESRDY_vect)
{
    /* Clear the interrupt flag by reading the result */
    adcVal = ADC0.RES;
 /* Update the flag */
 adcResultReady = 1;
}
/* This function initializes the RTC module  */
void RTC_init(void)
{
    /* 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 */
    RTC.PER = RTC_PERIOD;               /* Set period */
    RTC.CLKSEL = RTC_CLKSEL_OSC32K_gc;  /* 32.768kHz Internal Crystal (OSC32K)
*/
}
```

```
/* This function initializes the EVSYS module  */
void EVSYS_init(void)
{
    /* Real Time Counter overflow */
    EVSYS.CHANNEL0 = EVSYS_CHANNEL0_RTC_OVF_gc;
    /* Connect user to event channel 0 */
    EVSYS.USERADC0START = EVSYS_USER_CHANNEL0_gc;
}
int main(void)
{
    uint16_t result;

    /* Initialize all peripherals */
    CLKCTRL_init();
    PORT_init();
    VREF0_init();
    ADC0_init();
    LED0_init();
    USART1_init();
    RTC_init();
    EVSYS_init();

    /* Enable Global Interrupts */
    sei();

    while (1)
    {
        if (adcResultReady == 1)
  {
            /* Store the result to avoid altering adcVal because of the
interrupt. This operation must be atomic */
            cli();
            result = adcVal;
            sei();

   /* Update the flag value */
   adcResultReady = 0;
   /* Toggle the LED */
   LED0_toggle();

   /* Transmit the ADC result to be plotted using Data Visualizer */
   USART1_Write(START_TOKEN);
   USART1_Write(result & 0x00FF);
   USART1_Write(result >> 8);
   USART1_Write(END_TOKEN);
  }
    }
}
```

Example 13-7. ADC Temperature Measurement Code Example

```
/*
    \file   main.c
    \brief  ADC Temperature Measurement
    (c) 2019 Microchip Technology Inc. and its subsidiaries.
    Subject to your compliance with these terms, you may use Microchip
software and any
    derivatives exclusively with Microchip products. It is your responsibility
to comply with third-party
    license terms applicable to your use of third-party software (including
open source software) that
    may accompany Microchip software.
    THIS SOFTWARE IS SUPPLIED BY MICROCHIP "AS IS". NO WARRANTIES, WHETHER
    EXPRESS, IMPLIED OR STATUTORY, APPLY TO THIS SOFTWARE, INCLUDING ANY
    IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY, AND FITNESS
    FOR A PARTICULAR PURPOSE.
    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 SOFTWARE, 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 THIS SOFTWARE WILL NOT EXCEED THE AMOUNT
```

```
    OF FEES, IF ANY, THAT YOU HAVE PAID DIRECTLY TO MICROCHIP FOR THIS
    SOFTWARE.
*/
#define F_CPU           4000000UL   /* Main clock frequency */
#define START_TOKEN     0x03        /* Start Frame Token */
#define END_TOKEN       0xFC        /* End Frame Token */
/* Compute the baud rate */
#define USART1_BAUD_RATE(BAUD_RATE) (((float)F_CPU * 64 / (16 *
(float)BAUD_RATE)) + 0.5)
#include <avr/io.h>
#include <stdbool.h>
#include <avr/cpufunc.h>
void CLKCTRL_init(void);
void VREF0_init(void);
void ADC0_init(void);
void USART1_init(void);
uint16_t ADC0_read(void);
int16_t temperatureConvert(uint16_t data);
void ADC0_start(void);
void USART1_Write(const uint8_t data);
/* This function initializes the CLKCTRL module */
void CLKCTRL_init(void)
{
    /* FREQSEL 4M */
    ccp_write_io((void*)&(CLKCTRL.OSCHFCTRLA), (CLKCTRL.OSCHFCTRLA |
CLKCTRL_FREQSEL_4M_gc));
}
/* This function initializes the PORT module */
void PORT_init(void)
{
    /* Configure PC0 as output for USART1 TX */
    PORTC.DIRSET = PIN0_bm;
}
/* This function initializes the VREF module */
void VREF0_init(void)
{
    VREF.ADC0REF = VREF_REFSEL_2V048_gc;  /* Internal 2.048V reference */
}
/* This function initializes the ADC module */
void ADC0_init(void)
{
    ADC0.CTRLC = ADC_PRESC_DIV4_gc;         /* CLK_PER divided by 4 */
    ADC0.CTRLA = ADC_ENABLE_bm              /* ADC Enable: enabled */
                | ADC_RESSEL_12BIT_gc       /* 12-bit mode */
                | ADC_FREERUN_bm;           /* Free-run mode */
    ADC0.MUXPOS = ADC_MUXPOS_TEMPSENSE_gc;  /* Select ADC channel, Temp. */
}
/* This function initializes the USART module */
void USART1_init(void)
{
    /* Configure the baud rate: 115200 */
    USART1.BAUD = (uint16_t)USART1_BAUD_RATE(115200);
    USART1.CTRLB = USART_TXEN_bm;           /* Enable TX */
    USART1.CTRLC = USART_CHSIZE_8BIT_gc;    /* Configure character size: 8 bit
*/
}
/* This function returns the ADC conversion result */
uint16_t ADC0_read(void)
{
    /* Wait for ADC result to be ready */
    while (!(ADC0.INTFLAGS & ADC_RESRDY_bm));
    /* Clear the interrupt flag by reading the result */
    return ADC0.RES;
}
/* This function returns the temperature value in degrees C */
int16_t temperatureConvert(uint16_t data)
{
    uint16_t sigrow_offset = SIGROW.TEMPSENSE1;
    uint16_t sigrow_slope = SIGROW.TEMPSENSE0;
    int32_t temp;
    /* Clear the interrupt flag by reading the result (ADC0.RES) */
    temp = sigrow_offset - data;
    /* Result will overflow 16-bit variable */
    temp *= sigrow_slope;
    /* Add 4096/2 to get correct rounding on division below */
    temp += 0x0800;
```

```
    /* Round off to nearest degree in Kelvin, by dividing with 2^12 (4096) */
    temp >>= 12;
    /* Convert from Kelvin to Celsius (0 Kelvin - 273.15 = -273.1°C) */
    return temp - 273;
}
/* This function starts the ADC conversions*/
void ADC0_start(void)
{
    /* Start conversion */
    ADC0.COMMAND = ADC_STCONV_bm;
}
/* This function transmits one byte through USART */
void USART1_Write(const uint8_t data)
{
    /* Check if USART buffer is ready to transmit data */
    while (!(USART1.STATUS & USART_DREIF_bm));
    /* Transmit data using TXDATAL register */
    USART1.TXDATAL = data;
}
int main(void)
{
    int16_t temp_C;
    uint16_t adcVal;

    /* Initialize all peripherals */
    CLKCTRL_init();
    PORT_init();
    VREF0_init();
    ADC0_init();
    USART1_init();

    /* Start the ADC conversion */
    ADC0_start();

    while (1)
    {
        /* Read the conversion result */
        adcVal = ADC0_read();
        /* Convert the ADC result in degrees C */
        temp_C = temperatureConvert(adcVal);

        /* Transmit the ADC result to be plotted using Data Visualizer */
        USART1_Write(START_TOKEN);
        USART1_Write(temp_C & 0x00FF);
        USART1_Write(temp_C >> 8);
        USART1_Write(END_TOKEN);
    }
}
```

14. Revision History

Doc. Rev.	Date	Comments
D	10/2020	Updated the GitHub logo. Added Data Visualizer functionality description and steps. Updated code examples to transmit the ADC conversion results.
C	05/2020	Updated AVR® MCU DA (AVR-DA) to AVR® DA MCU and AVR-DA to AVR DA, per latest trademarking.
B	03/2020	Updated repository links. Updated AVR-DA to AVR® MCU DA (AVR-DA), per latest trademarking.
A	02/2020	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 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.

Legal Notice

Trademarks

ISBN: 978-1-5224-6951-3

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