TB3245

使用 12 位 ADC 进行转换、累加和触发事件

简介

作者：Ștefan Vlad，Microchip Technology Inc.

本技术简介所述的模数转换器（Analog-to-Digital Converter，ADC）是一款支持差分和单端转换的 12 位逐次逼近寄存器（Successive Approximation Register，SAR），同时也是 AVR® DA MCU（AVR DA）单片机上的一个外设。

本技术简介将介绍该 ADC 模块在 AVR DA 单片机上的工作方式，具体涵盖以下用例：

- **ADC 单次转换：**
  初始化 ADC，启动转换并等待其完成，然后在循环中读取 ADC 结果。
- **ADC 自由运行转换：**
  初始化 ADC，使能自由运行模式，启动转换并等待其完成，然后在无限循环中读取 ADC 结果。
- **ADC 差分转换：**
  初始化 ADC，配置两个引脚以读取差分电压，启动转换并等待其完成，然后在无限循环中读取 ADC 结果。
- **ADC 采样累加器：**
  初始化 ADC，使能累加 64 个采样，启动转换并等待其完成，然后在循环中读取 ADC 结果。
- **ADC 窗口比较器：**
  初始化 ADC，设置转换窗口比较器下限阈值，使能转换窗口模式，使能自由运行模式，启动转换并等待其完成，然后在无限循环中读取 ADC 结果。如果 ADC 结果低于设置的阈值，LED 将翻转为点亮状态。
- **ADC 事件触发：**
  初始化 ADC，初始化实时计数器（Real-Time Counter，RTC），配置事件系统（Event System，EVSYS）以在 RTC 溢出时触发 ADC 转换。每次 ADC 转换后，LED 都会翻转为点亮状态。
- **ADC 温度测量：**
  初始化 ADC，选择内部参考，选择温度传感器作为输入并通过运行 12 位右对齐的单端转换来获取数据。

所有示例的 ADC 结果将通过通用同步异步收发器（Universal Synchronous Asynchronous Receiver-Transmitter，USART）传输，并使用 Data Visualizer 工具绘制。

注： 示例代码是基于 AVR128DA48 Curiosity Nano 开发的。这些代码在 GitHub 上提供，支持 Atmel Studio 和 MPLAB® X 集成开发环境（Integrated Development Environment，IDE）。

目录

1. 相关器件

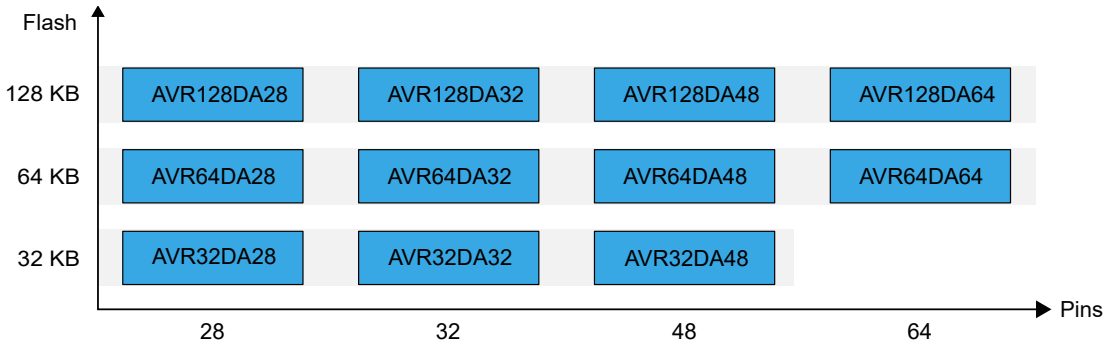
本章列出了本文档的相关器件。

1.1 AVR DA 系列概述

下图所示为 AVR DA 器件，注明了不同的引脚数与存储器大小：

- 垂直迁移无需修改代码，因为这些器件的引脚和功能完全兼容。
- 从右到左迁移会减少引脚数，进而减少可用的功能。

图 1-1. AVR® DA 系列概览



具有不同闪存大小的器件通常也具有不同的 SRAM。

2. 概述

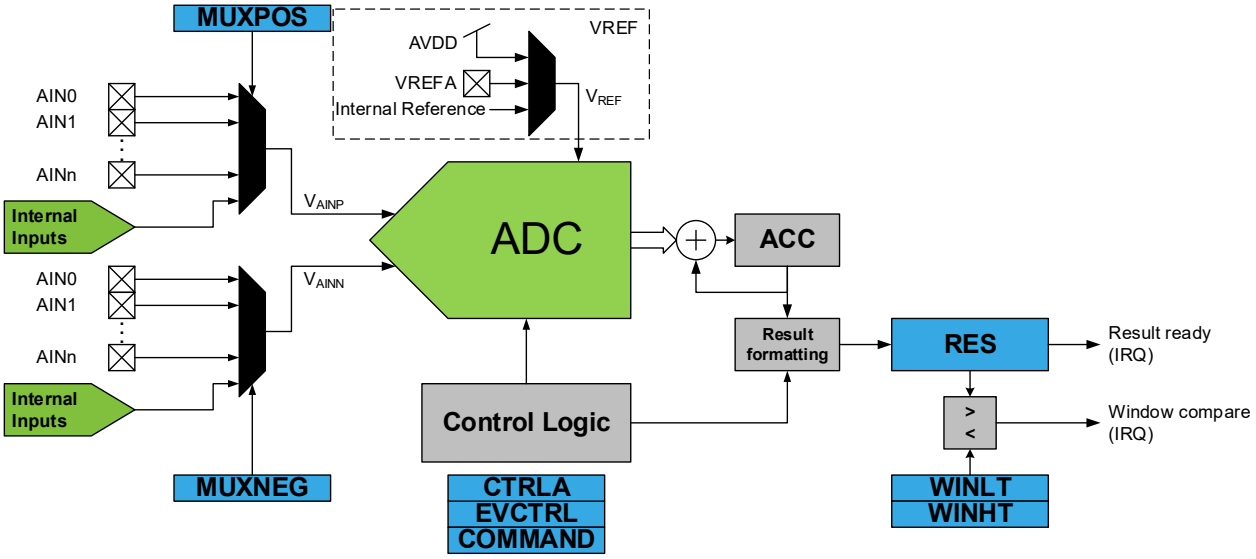
ADC 是采样率最高为 130 ksps、分辨率为 12 位的 SAR。ADC 连接到模拟输入多路开关，以便在多个单端或差分输入之间进行选择。在单端转换中，ADC 将测量选定输入通道与 0V（GND）之间的电压。在差分转换中，ADC 将测量两个选定输入通道之间的电压。选定 ADC 输入通道可以是内部（例如，参考电压）或外部模拟输入引脚。

ADC 支持突发采样，该采样方式会将转换结果（可配置数量）累加到单个 ADC 结果中（采样累加）。ADC 输入信号通过一个采样保持电路馈送，可确保 ADC 的输入电压在采样期间保持在恒定水平。

可选择来自内部参考电压（VREF）外设、AVDD 电源电压或外部 VREF 引脚（VREFA）的参考电压。

数字窗口比较功能可用于监视输入信号，经配置后可仅在采样低于或高于用户定义的阈值或者位于用户定义的窗口内部或外部时触发中断，几乎无需软件干预。

图 2-1. ADC 框图



通过写入 MUXPOS 寄存器（ADCn.MUXPOS）中的 MUXPOS 位域选择模拟输入通道。ADC 的任何输入引脚、GND、内部输入或温度传感器都可以被选作 ADC 的单端输入。通过向控制 A 寄存器（ADCn.CTRLA）中的 ADC ENABLE 位写入 1 来使能 ADC。参考电压和输入通道选择在 ADC 使能之后才会生效。

当 ENABLE 位为 0 时，ADC 不消耗功率。ADC 会生成一个 10 位或 12 位的右对齐结果，可从结果寄存器（ADCn.RES）中读取。

下面列出了转换公式，其中 V_{AINP} 和 V_{AINN} 是 ADC 的正负输入，而 V_{REF} 是选定的 ADC 参考电压。

- 单端 12 位转换：$RES = \frac{V_{AINP}}{V_{REF}} \times 4096 \in [0, 4095]$
- 单端 10 位转换：$RES = \frac{V_{AINP}}{V_{REF}} \times 1024 \in [0, 1023]$
- 差分 12 位转换：$RES = \frac{V_{AINP} - V_{AINN}}{V_{REF}} \times 2048 \in [-2048, 2047]$
- 差分 10 位转换：$RES = \frac{V_{AINP} - V_{AINN}}{V_{REF}} \times 512 \in [-512, 511]$

3. 硬件配置

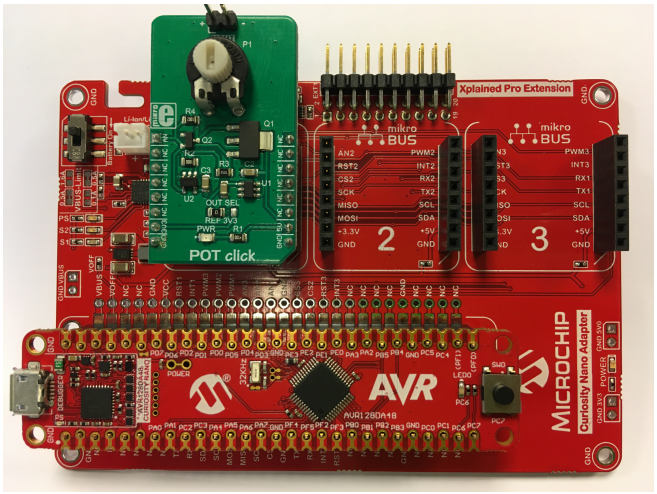
本示例使用的硬件基于：

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

本文档中介绍的用例将使用两种配置：

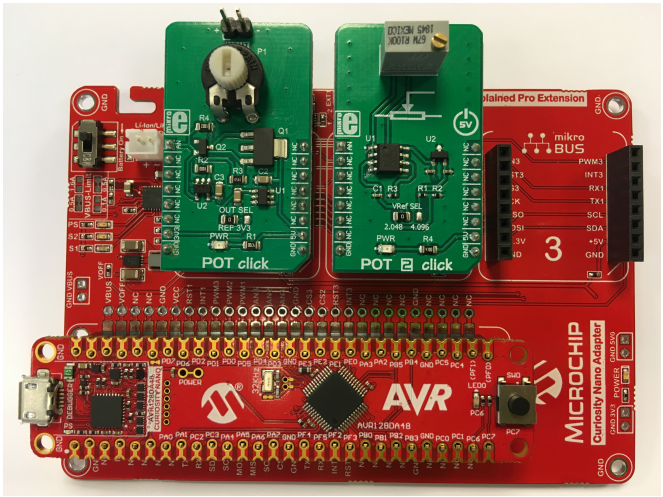
- 第一种配置（配置 A）使用 1 个 POT click 板（插入 mikroBUS 槽 1），并将 AN1 连接到 AVR DA 器件的 PD3（AIN3）引脚。

图 3-1. 使用 1 个 POT Click 板的硬件配置 A



- 第二种配置（配置 B）使用 2 个 POT click 板（分别插入 mikroBUS 槽 1 和槽 2）。槽 1 将 AN1 连接到 AVR DA 器件的 PD3（AIN3），槽 2 将 AN2 连接到 AVR DA 器件的 PD4（AIN4）引脚。

图 3-2. 使用 2 个 POT Click 板的硬件配置 B



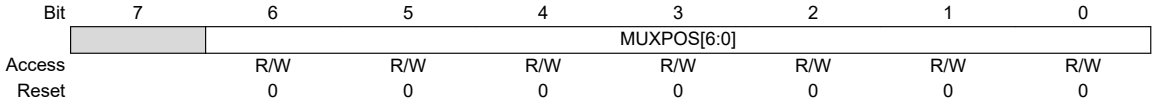
4. ADC 单次转换

4.1 初始化 ADC

本小节介绍了如何初始化 ADC 模块以在单次转换模式下运行。ADC 输入引脚需要禁止数字输入缓冲器和上拉电阻，以获得尽可能高的输入阻抗。

ADC 单次转换使用 3. 硬件配置中所述的配置 A。

图 4-1. ADC0.MUXPOS 选择



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

要选择 ADC 通道 AIN3（PD3），应使用以下代码：

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

可使用 ADC 时钟预分频器对时钟频率进行分频。在本示例中，时钟进行 4 分频。

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

图 4-2. ADC0 参考选择

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

ADC 可以使用 V_{DD}、外部或内部参考作为其正参考。本示例中使用内部参考。

```
VREF.ADC0REF = VREF_REFSEL_2V048_gc; /* 2.048V 内部参考 */
```

通过 ADC0.CTRLA 寄存器中的 RESSEL 位设置 ADC 分辨率。通过将 ADC0.CTRLA 寄存器中的 ENABLE 位置 1 来使能 ADC。

图 4-3. ADC0.CTRLA 分辨率选择

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

相应的代码如下：

```
ADC0.CTRLA = ADC_ENABLE_bm /* ADC 使能：已使能 */
           | ADC_RESSEL_12BIT_gc; /* 12 位模式 */
```

通过将 ADC0.COMMAND 寄存器中的 STCONV 位置 1 来启动 ADC 转换。

图 4-4. ADC0.COMMAND——启动转换

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.

相应的代码如下：

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

转换完成后，硬件会将 ADC0.INTFLAGS 中的 RESRDY 位置 1。用户必须等待此位置 1，然后才能读取 ADC 结果。

图 4-5. ADC0.INTFLAGS——硬件置 1 的 RESRDY 位

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.

在启动另一次转换之前，用户必须通过读取 ADC0.RES 寄存器来将 RESRDY 位清零：

```
/* 通过读取结果来清零中断标志 */
    return ADC0.RES;
```

可以从 ADC0.RES 寄存器读取转换结果：

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

4.2 发送结果

要使用 Data Visualizer 插件将 ADC 结果可视化，用户必须先通过 USART 发送结果。必须使用 USART 波特率寄存器配置波特率。具体如下图所示。

图 4-6. USART 波特率寄存器

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.

必须使用以下代码。

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

USART1_BAUD_RATE 宏用于计算波特率寄存器的相应值。具体如下所示。

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

可以使用 CTRLB 寄存器使能 USART 传输，如下所示。

图 4-7. 控制 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

必须使用以下代码。

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

可以使用 CTRLC 寄存器配置字符大小。

图 4-8. 控制 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

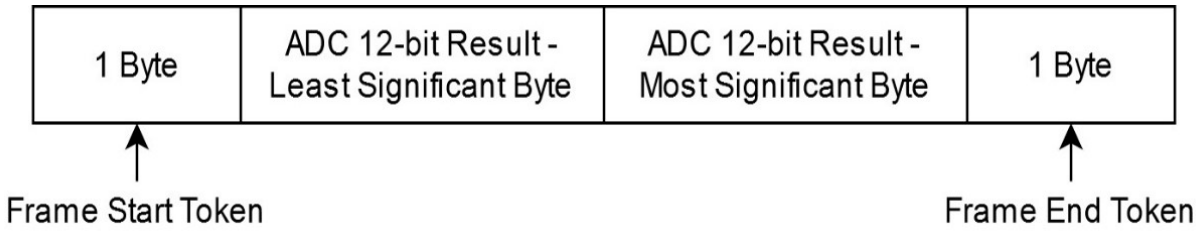
必须使用以下代码。

```
USART1.CTRLC = USART_CHSIZE_8BIT_gc;     /* 配置字符大小：8 位 */
```

要通过 USART 发送一个字节，用户必须检查 USART 缓冲区是否已做好发送准备，然后将数据写入 TXDATAL 寄存器。

为确保 Data Visualizer 正确解析数据，数据必须遵循特定格式。12 位分辨率的 ADC 结果将以 16 位格式（扩展符号位）提供，结果中的两个字节将按如下方式发送。

图 4-9. ADC 结果发送格式



帧开始令牌（START_TOKEN）和帧结束令牌（END_TOKEN）宏按以下方式定义。

```
#define START_TOKEN 0x03     /* 开始帧令牌 */
#define END_TOKEN 0xFC       /* 结束帧令牌 */
```

在无限循环中，将读取 ADC 结果，使用特定的格式通过 USART 发送该结果，然后再次启动转换。

要通过 USART 发送 ADC 转换结果，必须在主无限循环中添加以下代码：

```
while (1)
{
    /* 启动 ADC 转换 */
    ADC0_start();

    /* 读取 ADC 结果 */
    adcVal = ADC0_read();

    /* 发送要使用 Data Visualizer 绘制的 ADC 结果 */
    USART1_Write(START_TOKEN);
    USART1_Write(adcVal & 0x00FF);
    USART1_Write(adcVal >> 8);
    USART1_Write(END_TOKEN);
}
```

4.3 代码示例

可通过以下链接查看使用 Atmel Studio 开发的代码示例。

可通过以下链接查看使用 MPLAB X IDE 开发的代码示例。

提示： 有关完整代码的示例，请参见附录部分。

5. ADC 自由运行转换

5.1 初始化 ADC

本小节介绍了如何初始化 ADC 模块以在自由运行转换模式下运行。本应用使用 3. 硬件配置中所述的配置 A。要激活此模式，除了正常的 ADC 初始化之外，还必须将 ADC0.CTRLA 中的 FREERUN 位置 1。将 ADC 配置为自由运行模式时，每次完成转换后都会立即启动下一次转换。

图 5-1. ADC0.CTRLA——将 FREERUN 位置 1

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

相应的代码如下：

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

通过将 ADC0.COMMAND 寄存器中的 STCONV 位置 1 来启动 ADC 转换：

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

随后，可以在 `while` 循环中读取 ADC 结果。

5.2 发送结果

要发送 ADC 转换结果，必须在无限循环中实现以下算法：

```
while (1)
{
    /* 启动 ADC 转换 */
    ADC0_start();

    /* 读取 ADC 结果 */
    adcVal = ADC0_read();

    /* 发送要使用 Data Visualizer 绘制的 ADC 结果 */
    USART1_Write(START_TOKEN);
    USART1_Write(adcVal & 0x00FF);
    USART1_Write(adcVal >> 8);
    USART1_Write(END_TOKEN);
}
```

5.3 代码示例

可通过以下链接查看使用 Atmel Studio 开发的代码示例。

View Code Examples on GitHub
Click to browse repositories

可通过以下链接查看使用 MPLAB X IDE 开发的代码示例。

提示： 有关完整代码的示例，请参见附录部分。

6. ADC 差分转换

6.1 初始化 ADC

本小节介绍了如何初始化 ADC 模块以在差分转换模式下运行。本应用使用 3. 硬件配置中所述的配置 B。

通过将 ADC0.CTRLA 寄存器中的 CONVMODE 位置 1 来使能 ADC 差分转换模式：

图 6-1. ADC0.CTRLA——将 CONVMODE 位置 1

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.

相应的代码如下：

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

在差分转换中，我们将一个 Pot click 板连接到 AVR DA 器件的 PD3（AIN3）引脚，将另一个 Pot 2 click 板连接到 AVR DA 器件的 PD4（AIN4）引脚。

图 6-2. ADC 负输入的多路开关选择

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;
```

随后，可以在 `while` 循环中读取 ADC 结果。

6.2 发送结果

可以读取和发送 ADC 差分转换结果，方法如以下代码所示。

```
while (1)
{
    if (ADC0_conversionDone())
    {
        /* 读取 ADC 结果 */
        adcVal = ADC0_read();
        /* 发送要使用 Data Visualizer 绘制的 ADC 结果 */
        USART1_Write(START_TOKEN);
        USART1_Write(adcVal & 0x00FF);
        USART1_Write(adcVal >> 8);
        USART1_Write(END_TOKEN);
    }
}
```

6.3 代码示例

可通过以下链接查看使用 Atmel Studio 开发的代码示例。

可通过以下链接查看使用 MPLAB X IDE 开发的代码示例。

提示： 有关完整代码的示例，请参见附录部分。

7. ADC 采样累加器

7.1 初始化 ADC

本小节介绍了如何初始化 ADC 模块以使用采样累加器功能。本应用使用 3. 硬件配置中所述的配置 A。在这种模式下，ADC 可以在累加器寄存器中添加最多 128 个采样，从而对信号进行滤波并降低噪声，这在需要平滑信号的模拟传感器数据读取操作中非常有用。通过使用硬件累加器（而不是在软件中添加这些读数），可以减轻 CPU 负荷。

要激活此模式，除了正常的 ADC 初始化之外，还必须在 ADC0.CTRLB 寄存器中设置采样累加数量。

图 7-1. ADC0.CTRLB——将 SAMPNUM 位置 1

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.

相应的代码如下：

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

采样将加到 ADC0.RES 寄存器中。当获得的采样达到 ADC0.CTRLB 中指定的数量后，ADC_RESRDY 标志将置 1。

用户可以读取该值并将其除以采样数以获得平均值。如果累加的采样超过 16 个，则结果将在硬件中被截断为 16 位变量，需要除以 16：

```
adcVal = ADC0_read(); /* 通过读取结果来清零中断标志 */
adcVal = adcVal >> ADC_SHIFT_DIV16; /* 如果采样数 > 16，则除以采样数（16） */
```

7.2 发送结果

必须读取 ADC 结果，将其除以采样数，然后通过 USART 发送，如下所示：

```
while (1)
{
    /* 读取 ADC 结果 */
    adcVal = ADC0_read();
    /* 如果采样数 > 16，则除以采样数（16） */
    adcVal = adcVal >> ADC_SHIFT_DIV16;
    /* 发送要使用 Data Visualizer 绘制的 ADC 结果 */
    USART1_Write(START_TOKEN);
    USART1_Write(adcVal & 0x00FF);
    USART1_Write(adcVal >> 8);
```

```
    USART1_Write(END_TOKEN);
}
```

7.3 代码示例

可通过以下链接查看使用 Atmel Studio 开发的代码示例。

可通过以下链接查看使用 MPLAB X IDE 开发的代码示例。

提示： 有关完整代码的示例，请参见附录部分。

8. ADC 窗口比较器

8.1 初始化 ADC

在窗口比较器配置中，器件会检测 ADC 结果是否低于或高于特定阈值。该配置非常实用，它可以监视信号是否按照要求保持在特定范围内，或者在电池电量不足/过充时发出信号。窗口比较器可以在自由运行模式和单次转换模式下使用。

ADC 窗口比较器使用 3. 硬件配置中所述的配置 A。

在本示例中，窗口比较器以自由运行模式使用。监视的信号需要连续采样，自由运行模式不需要每次转换都手动启动，从而可减轻 CPU 负荷。对于本例，ADC0.WINLT 寄存器中设置了阈值 0x100。

图 8-1. ADC-WINLT——设置下限阈值

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.

以下代码片段定义了 WINDOW_CMP_LOW_TH_EXAMPLE 宏，并将其设为 0x100 作为窗口比较器的下限阈值：

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

ADC 窗口比较器模式在 ADC0.CTRLE 寄存器中设置：

图 8-2. ADC0.CTRLE——设置窗口比较器模式

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

下面这行代码用于设置低于 WINLT 的结果的阈值：

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

如果 ADC 结果低于先前设置的阈值，则硬件会将 ADC0.INTFLAGS 寄存器中的 WCMP 位置 1，需要通过向其中写 1 或从 RES 寄存器中读取转换结果来清零。

8.2 发送结果

将读取 ADC 结果。如果结果低于特定阈值，LED 将点亮。否则，LED 将熄灭。然后转换结果将通过 USART 发送，如下所示。

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

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

    adcVal = ADC0_read();

    /* 发送要使用 Data Visualizer 绘制的 ADC 结果 */
    USART1_Write(START_TOKEN);
    USART1_Write(adcVal & 0x00FF);
    USART1_Write(adcVal >> 8);
    USART1_Write(END_TOKEN);
}
```

8.3 代码示例

可通过以下链接查看使用 Atmel Studio 开发的代码示例。

View Code Examples on GitHub
Click to browse repositories

可通过以下链接查看使用 MPLAB X IDE 开发的代码示例。

View Code Examples on GitHub
Click to browse repositories

提示： 有关完整代码的示例，请参见附录部分。

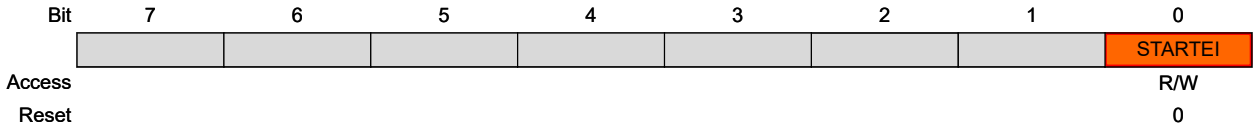
9. ADC 事件触发

9.1 初始化 ADC

ADC 转换也可以由事件触发，具体通过将 1 写入事件控制（ADCn.EVCTRL）寄存器中的启动事件输入位（STARTEI）来使能（本示例中将使用 ADC0 实例）。

ADC 事件触发用例基于 3. 硬件配置中所述的配置 A。

图 9-1. ADC0.EVTRL——使能 STARTEI 位



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.

下面这行代码可实现由上升沿事件触发 ADC：

```
ADC0.EVCTRL |= ADC_STARTEI_bm;
```

通过事件系统（EVSYS）发送到 ADC 的任何传入事件都将触发 ADC 转换。事件触发输入对边沿敏感。事件发生时，ADCn.COMMAND 中的 STCONV 置 1；转换完成后，STCONV 将清零。

例如，要在 RTC 溢出时启动 ADC 转换，必须进行以下设置：

1. 必须将 RTC 溢出事件连接到事件系统的通道 0。
2. 必须将事件用户 ADC0 配置为从通道 0 获取其输入。
3. 必须将 ADC 的 ADC0.EVCTRL 寄存器中的 STARTEI 位置 1，以允许由事件触发 ADC 转换。

```
EVSYS.CHANNEL0 = EVSYS_GENERATOR_RTC_OVF_gc; /* 实时计数器溢出 */
EVSYS.USERADC0 = EVSYS_CHANNEL_CHANNEL0_gc; /* 将用户连接到事件通道 0 */
ADC0.EVCTRL |= ADC_STARTEI_bm; /* 使能事件触发的转换 */
```

9.2 发送结果

ADC 结果将通过 USART 发送。为了避免由于中断而损坏 adcVal 值，将使用另一个变量存储结果，如下所示。

```
while (1)
{
    if (adcResultReady == 1)
    {
        /* 存储结果以避免由于中断而改变 adcVal。此操作必须为原子操作 */
        cli();
        result = adcVal;
        sei();

        /* 更新标志值 */
        adcResultReady = 0;
        /* 翻转 LED 状态 */
        LED0_toggle();

        /* 发送要使用 Data Visualizer 绘制的 ADC 结果 */
        USART1_Write(START_TOKEN);
        USART1_Write(result & 0x00FF);
        USART1_Write(result >> 8);
        USART1_Write(END_TOKEN);
    }
```

```
}
```

9.3 代码示例

可通过以下链接查看使用 Atmel Studio 开发的代码示例。

可通过以下链接查看使用 MPLAB X IDE 开发的代码示例。

提示： 有关完整代码的示例，请参见附录部分。

10. ADC 温度测量

10.1 初始化 ADC

ADC 温度测量使用 AVR DA 片上温度传感器，除 AVR128DA48 Curiosity Nano 板外，本用例不需要任何额外的硬件。此外，还需要通过 ADC0.MUXPOS 寄存器将温度传感器设置为 ADC 输入：

图 10-1. ADC0.MUXPOS 选择

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

为获得更准确的读数，必须使用器件生产或用户校准过程中的补偿值通过应用软件来处理温度传感器测量的结果。

温度计算公式如下（单位为开氏度）：

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

温度传感器校准值包含校正因子，用于片上温度传感器的温度测量值。SIGROW.TEMPSENSE0 为增益/斜率（无符号）的校正因子，SIGROW.TEMPSENSE1 为失调（有符号）的校正因子。

使用签名行的补偿值时，建议使用以下代码片段：

```
uint16_t sigrow_offset = SIGROW.TEMPSENSE1;
uint16_t sigrow_slope = SIGROW.TEMPSENSE0;
/* 通过读取 ADC0.RES 来清零中断标志 */
temp = sigrow_offset - ADC0.RES;
temp *= sigrow_slope; /* 变量超过 16 位时，结果将溢出 */
temp += 0x0800; /* 加上 4096/2 以对下面的除法进行正确的舍入 */
temp >>= 12; /* 除以 2^12（4096），舍入为最接近的开氏度 */
return temp - 273; /* 从开氏度转换为摄氏度（0K - 273.15 = -273.1°C）*/
```

随后，可以在 `while` 循环中读取 ADC 结果。

10.2 发送结果

温度将通过 USART 发送。读取 ADC 值后，结果将转换为对应的摄氏温度值。

```
while (1)
{
    /* 读取转换结果 */
    adcVal = ADC0_read();
    /* 将 ADC 结果转换为摄氏度 */
    temp_C = temperatureConvert(adcVal);

    /* 发送要使用 Data Visualizer 绘制的 ADC 结果 */
    USART1_Write(START_TOKEN);
    USART1_Write(temp_C & 0x00FF);
    USART1_Write(temp_C >> 8);
    USART1_Write(END_TOKEN);
}
```

10.3 代码示例

可通过以下链接查看使用 Atmel Studio 开发的代码示例。

View Code Examples on GitHub
Click to browse repositories

可通过以下链接查看使用 MPLAB X IDE 开发的代码示例。

View Code Examples on GitHub
Click to browse repositories

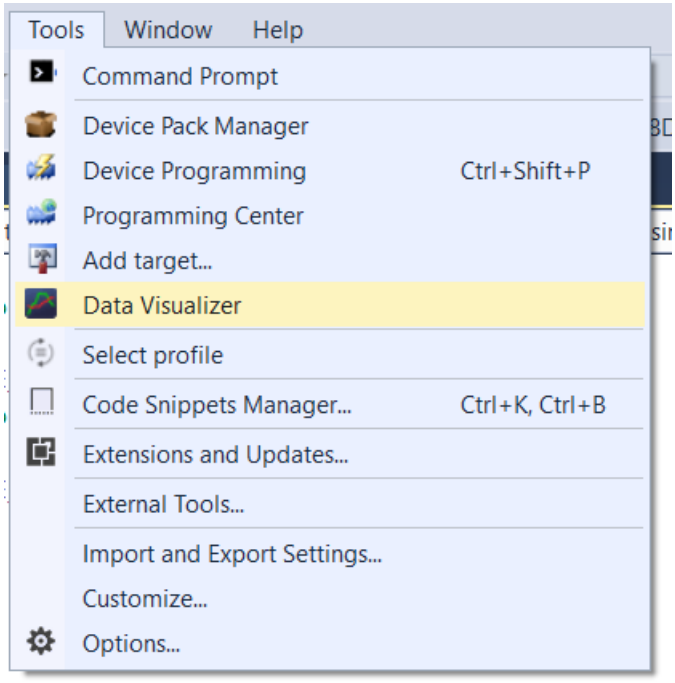
提示： 有关完整代码的示例，请参见附录部分。

11. Data Visualizer

要解析 ADC 结果并进行可视化，用户必须执行以下步骤：

1. 通过选择 *Tools > Data Visualizer*（工具 > 数据可视化工具），打开 Data Visualizer。

图 11-1. 打开 Data Visualizer

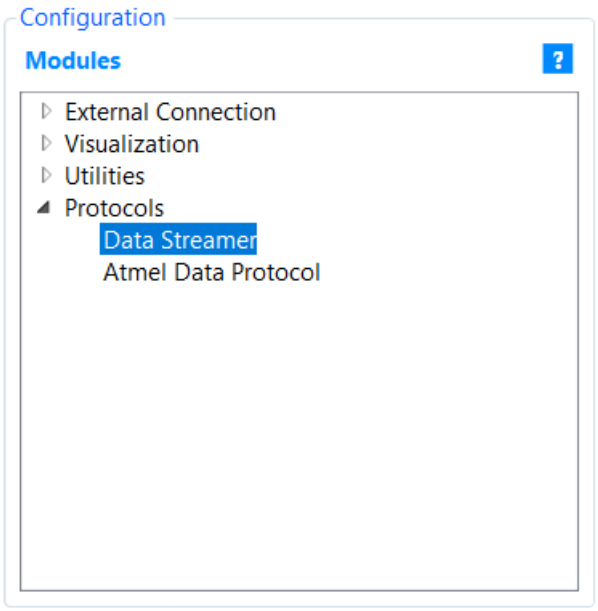


2. 连接到 Curiosity Nano COMn 端口，如下图所示。

图 11-2. 连接到所需的通信端口

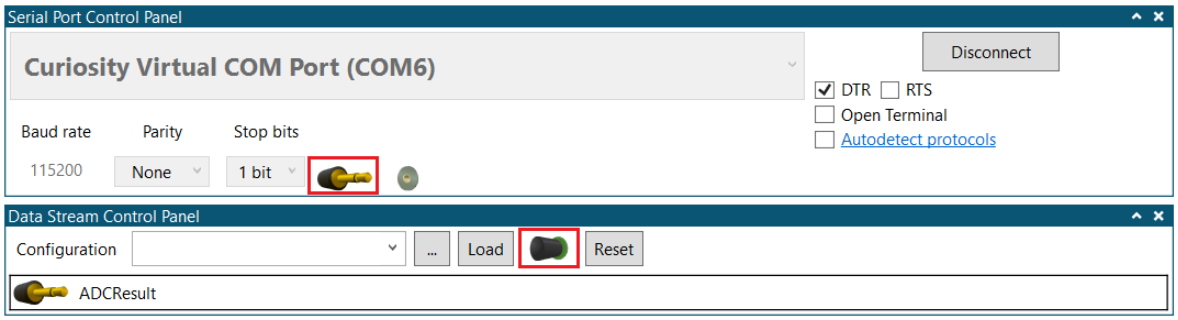
3. 在 *Configuration > Protocols*（配置 > 协议）部分，双击打开 **Data Streamer**。

图 11-3. 打开新的 Data Streamer 协议



4. 通过拖放操作将 COMn 端口源连接到数据流控制面板输入。

图 11-4. 将通信端口链接到 Data Streamer



5. 选择配置文件，然后单击 **Load**（加载）。

图 11-5. 浏览并找到配置文件

图 11-6. 选择配置文件

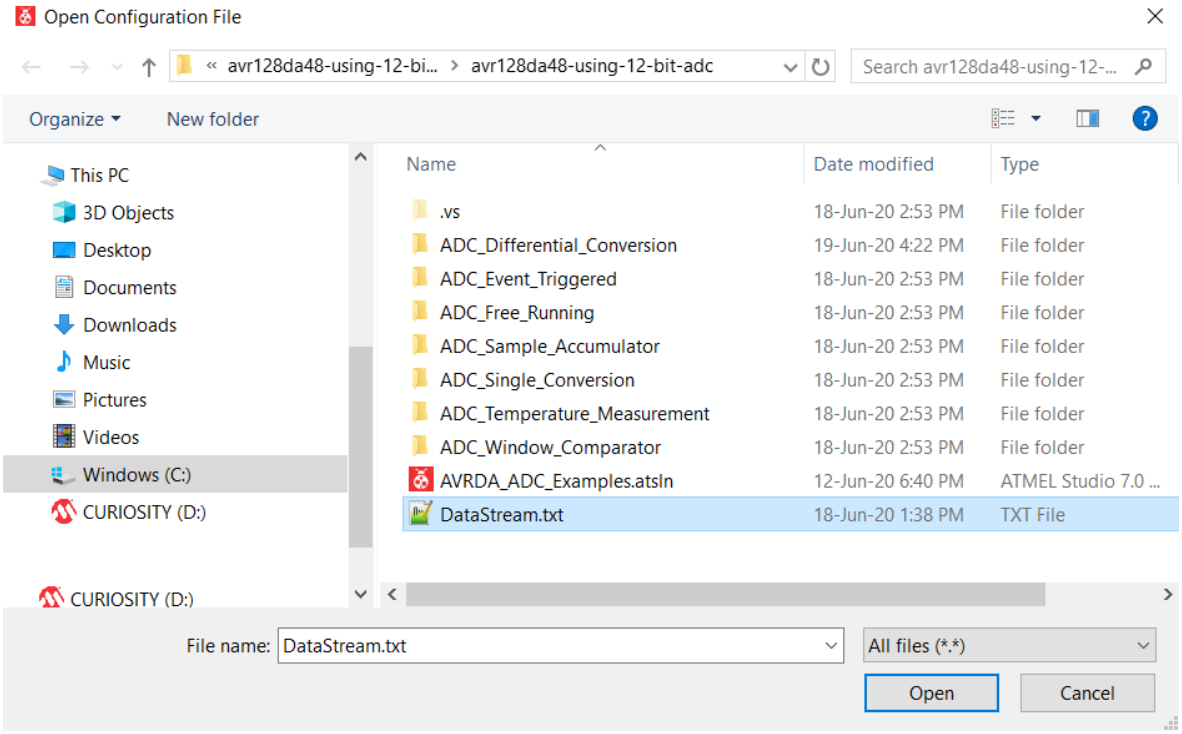
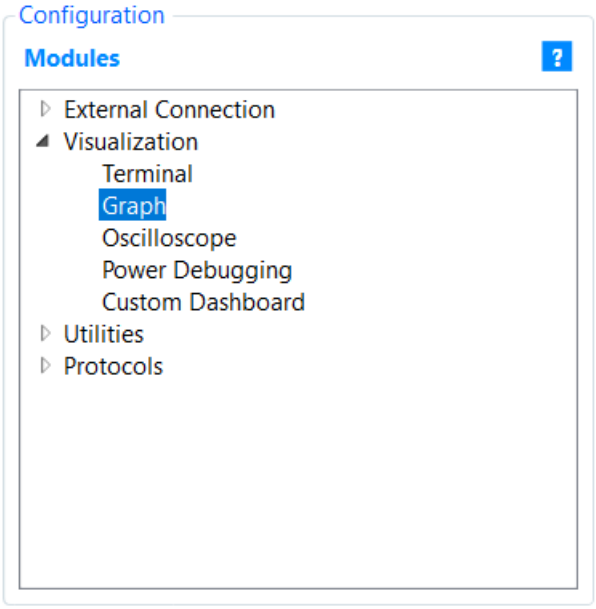


图 11-7. 加载配置文件

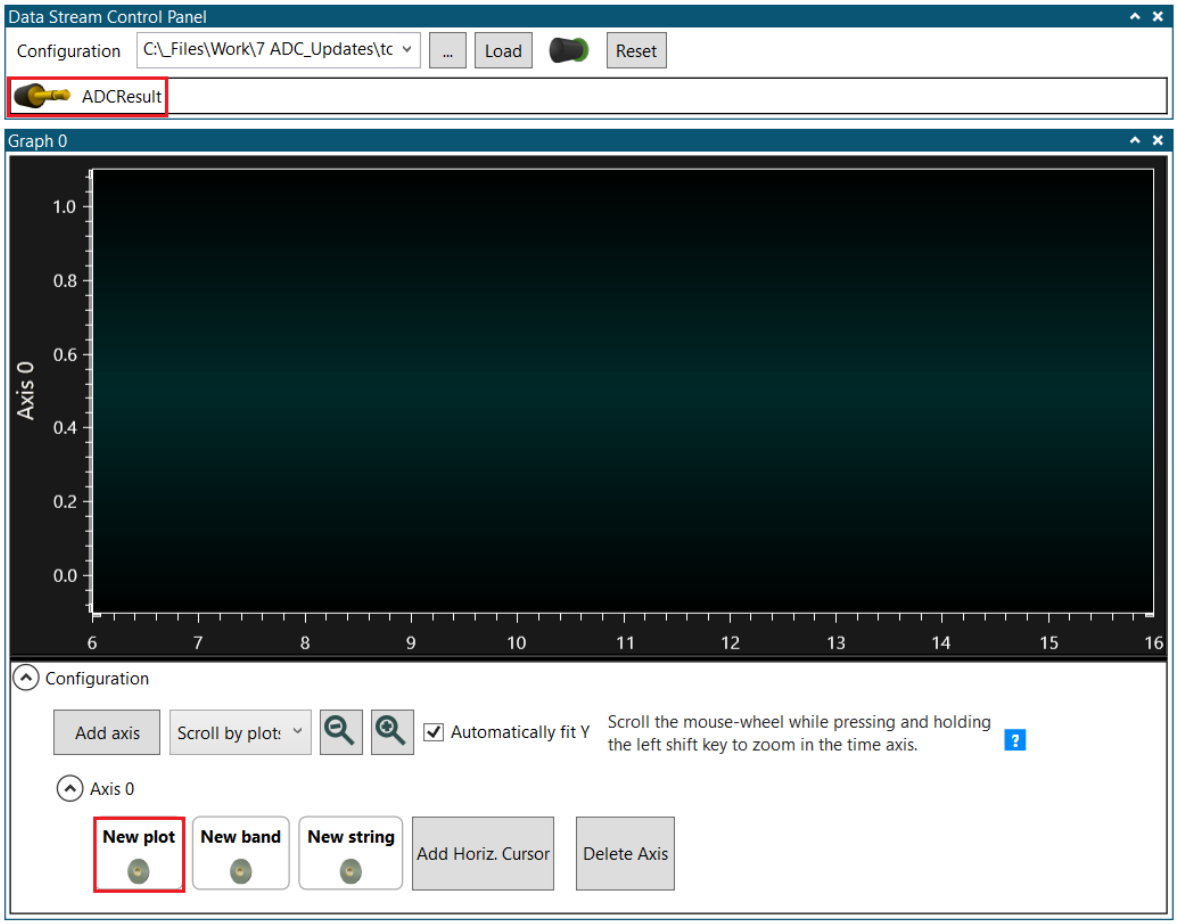
6. 在 *Configuration > Visualization*（配置 > 可视化）部分，双击打开新的 **Graph**（图表）。

图 11-8. 打开新图



7. 通过拖放操作将 ADC 结果源连接到图表中：*Configuration > Axis 0 > New plot*（配置 > 轴 0 > 新建曲线图）输入。

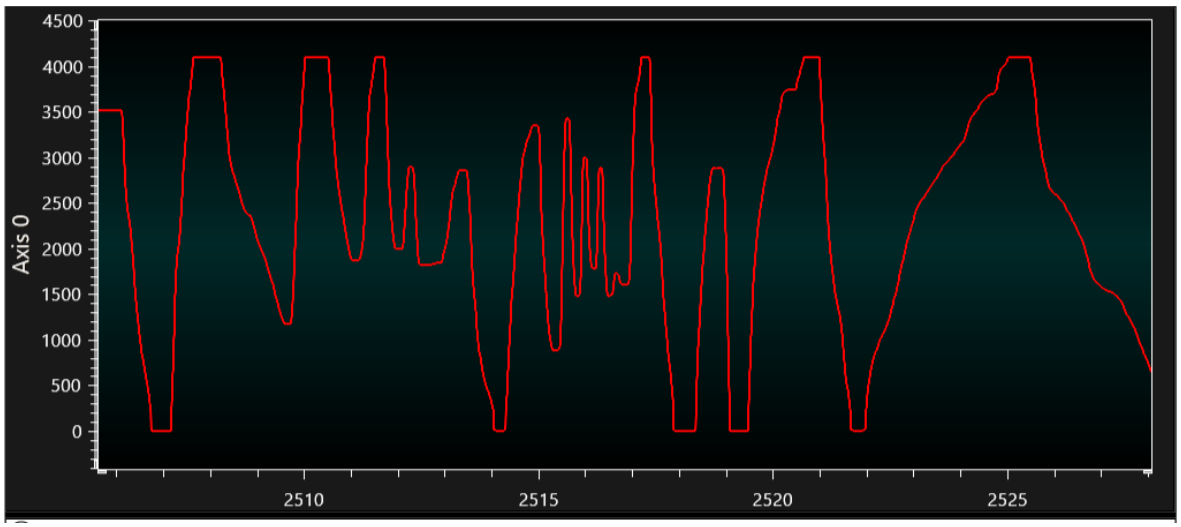
图 11-9. 将 ADC 结果连接到图表中的曲线图



所有用例的结果如下图所示。结果是通过随机转动电位器转子获得的。

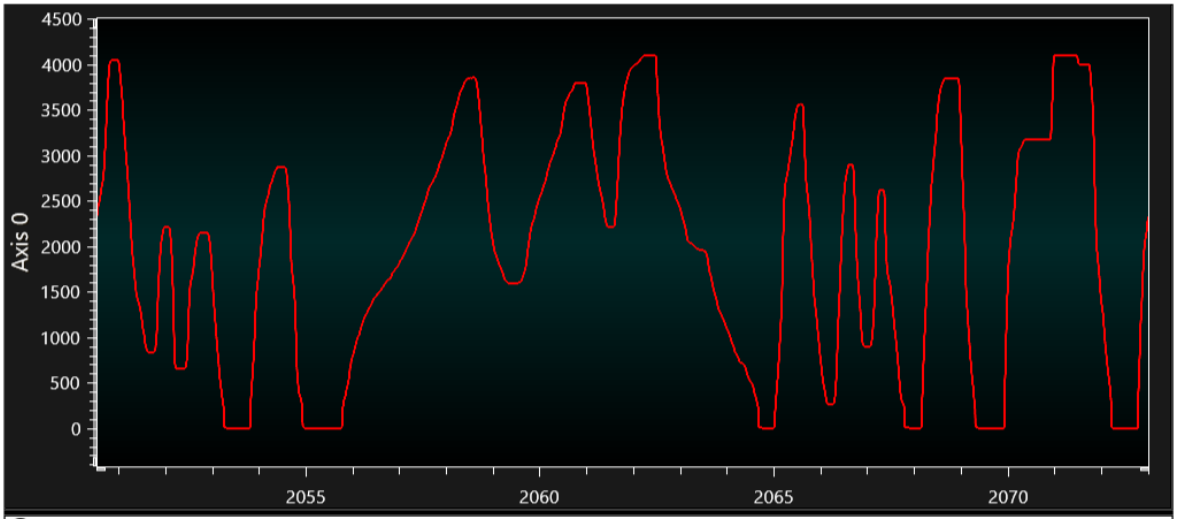
ADC 单次转换用例结果的曲线图如下图所示。结果范围为[0; 4095]，与 12 位分辨率值的预期结果相同。

图 11-10. ADC 单次转换



ADC 自由运行用例结果如下所示。这些结果与单次转换用例结果类似。

图 11-11. ADC 自由运行转换



ADC 差分转换结果如下所示。结果范围为[-2048; +2047]，与 12 位有符号值的预期结果相同。

图 11-12. ADC 差分转换

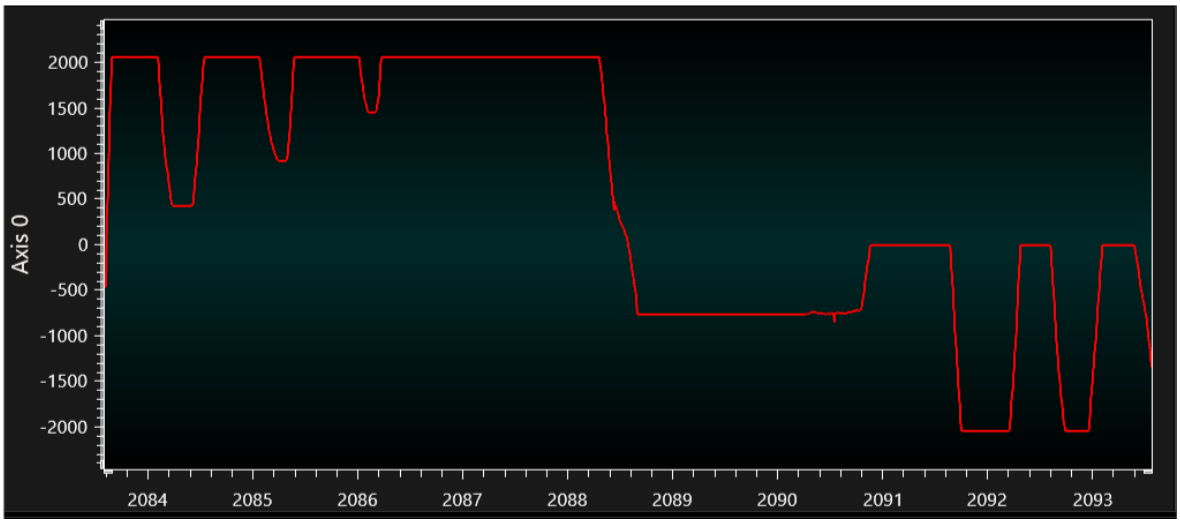
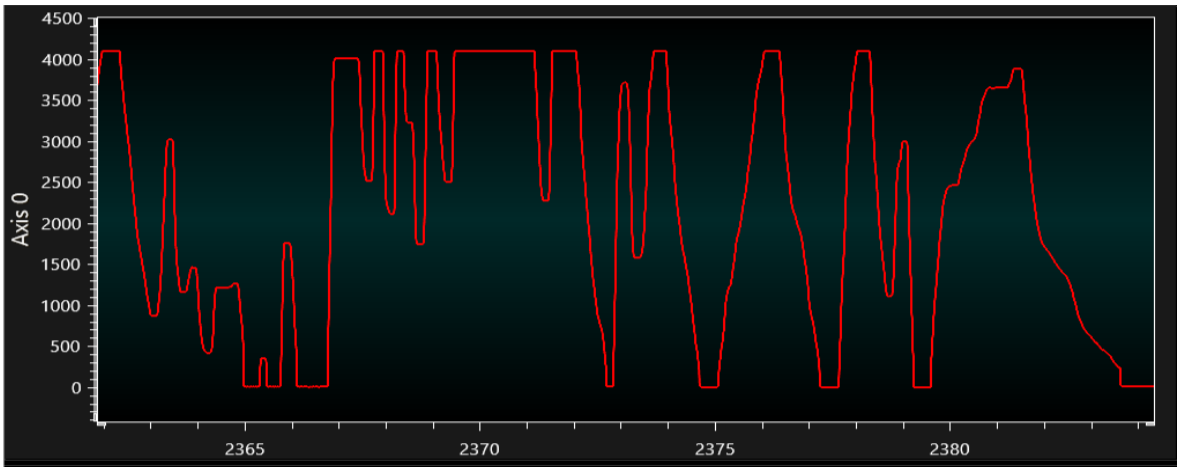


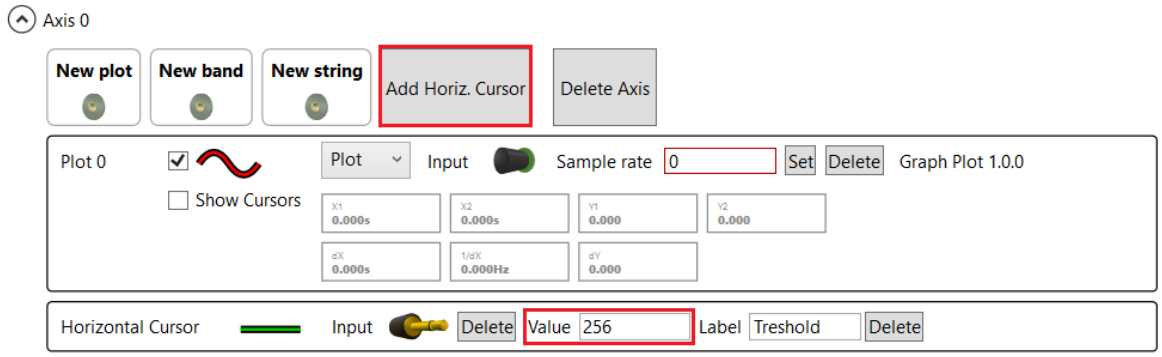
图 11-13. ADC 采样累加器



对于 ADC 窗口比较器用例，可以添加一个水平光标来测量所需的阈值。具体步骤如下图所示。

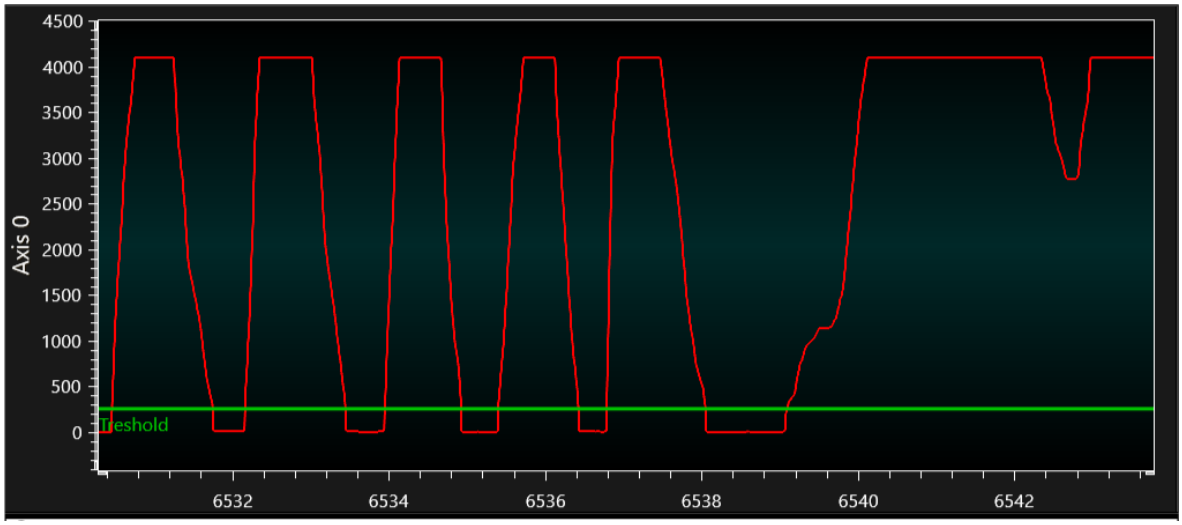
在该用例中，旋转电位器将修改 ADC 值，直到 LED0 点亮或熄灭。阈值（与 LED0 状态变化相对应的 ADC 转换结果的值）由水平光标突出显示。

图 11-14. 添加水平光标



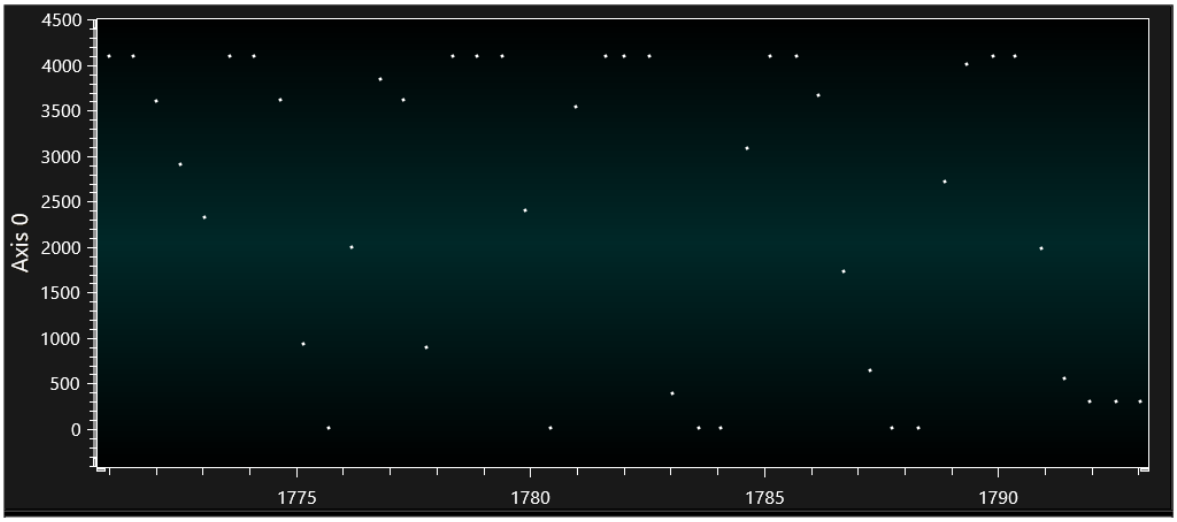
曲线图如下图所示。

图 11-15. ADC 窗口比较器



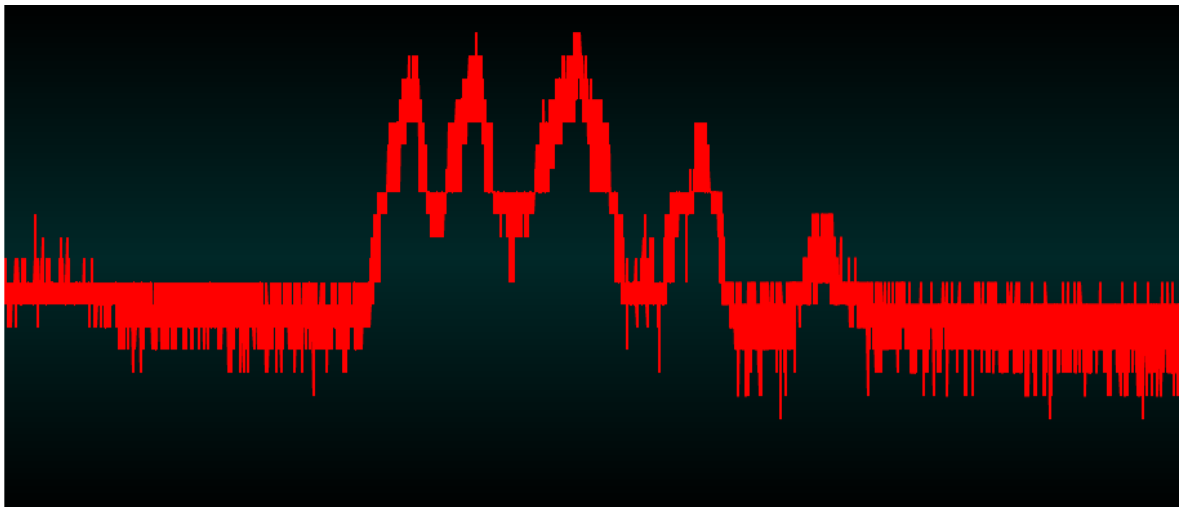
ADC 事件触发用例结果如下图所示。

图 11-16. ADC 事件触发



ADC 温度测量用例结果如下图所示。为获得这些结果，我们在设置环境中引入了微小的温度变化。

图 11-17. ADC 温度测量



12. 参考资料

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

13. 附录

例 13-1. ADC 单次转换代码示例

```
/*
    \文件   main.c
    \简述   ADC 单次转换
    (c) 2019 Microchip Technology Inc 及其子公司。
    在遵守这些条款的前提下，您可以将 Microchip 软件和任何衍生产品专门用于 Microchip 产品。您有责任遵守适用于您使用可能附带 Microchip 软件的第三方软件（包括开源软件）时的第三方许可条款。
    本软件由 Microchip“按原样”提供。本软件不作任何明示、暗示或法定的保证，包括对非侵权、适销性和适销性的任何默示保证。
    在任何情况下，Microchip 均不对与软件相关的任何类型的间接、特殊、惩罚性、附带或后果性损失、损害、成本或费用负责，即使是 Microchip 已被告知这种可能性或损害可以预见。在法律允许的最大范围内，Microchip 对与本软件有关的所有索赔的全部责任不会超过您为本软件直接支付给 Microchip 的费用（如果有）。
*/
#define F_CPU           4000000UL   /* 主时钟频率 */
#define START_TOKEN     0x03        /* 开始帧令牌 */
#define END_TOKEN       0xFC        /* 结束帧令牌 */
/* 计算波特率 */
#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);
/* 此函数初始化 CLKCTRL 模块 */
void CLKCTRL_init(void)
{
    /* FREQSEL 4M */
    ccp_write_io((void*)&(CLKCTRL.OSCHFCTRLA), (CLKCTRL.OSCHFCTRLA |
CLKCTRL_FREQSEL_4M_gc));
}
/* 此函数初始化 PORT 模块 */
void PORT_init(void)
{
    /* 将 PC0 配置为 USART1 TX 的输出 */
    PORTC.DIRSET = PIN0_bm;

    /* 禁止 PD3 上的中断和数字输入缓冲器 */
    PORTD.PIN3CTRL &= ~PORT_ISC_gm;
    PORTD.PIN3CTRL |= PORT_ISC_INPUT_DISABLE_gc;

    /* 禁止所有上拉电阻 */
    PORTD.PIN3CTRL &= ~PORT_PULLUPEN_bm;
}
/* 此函数初始化 VREF 模块 */
void VREF0_init(void)
{
    VREF.ADC0REF = VREF_REFSEL_2V048_gc;  /* 2.048V 内部参考 */
}
/* 此函数初始化 ADC 模块 */
void ADC0_init(void)
{
    ADC0.CTRLC = ADC_PRESC_DIV4_gc;         /* CLK_PER 进行 4 分频 */
    ADC0.CTRLA = ADC_ENABLE_bm              /* ADC 使能：已使能 */
                | ADC_RESSEL_12BIT_gc;      /* 12 位模式 */
    ADC0.MUXPOS = ADC_MUXPOS_AIN3_gc;       /* 选择 ADC 通道 AIN3 <-> PD3 */
}
/* 此函数初始化 USART 模块 */
```

```
void USART1_init(void)
{
    /* 配置波特率：115200 */
    USART1.BAUD = (uint16_t)USART1_BAUD_RATE(115200);
    USART1.CTRLB = USART_TXEN_bm;           /* 使能 TX */
    USART1.CTRLC = USART_CHSIZE_8BIT_gc;    /* 配置字符大小：8 位 */
}
/* 此函数返回 ADC 转换结果 */
uint16_t ADC0_read(void)
{
    /* 等待 ADC 结果就绪 */
    while (!(ADC0.INTFLAGS & ADC_RESRDY_bm));
    /* 通过读取结果来清零中断标志 */
    return ADC0.RES;
}
/* 此函数启动 ADC 转换 */
void ADC0_start(void)
{
    /* 启动 ADC 转换 */
    ADC0.COMMAND = ADC_STCONV_bm;
}
/* 此函数通过 USART 传输一个字节 */
void USART1_Write(const uint8_t data)
{
    /* 检查 USART 缓冲区是否已做好发送准备 */
    while (!(USART1.STATUS & USART_DREIF_bm));
    /* 使用 TXDATAL 寄存器发送数据 */
    USART1.TXDATAL = data;
}
int main(void)
{
    uint16_t adcVal;

    /* 初始化所有外设 */
    CLKCTRL_init();
    PORT_init();
    VREF0_init();
    ADC0_init();
    USART1_init();

    while (1)
    {
        /* 启动 ADC 转换 */
        ADC0_start();

        /* 读取 ADC 结果 */
        adcVal = ADC0_read();

        /* 发送要使用 Data Visualizer 绘制的 ADC 结果 */
        USART1_Write(START_TOKEN);
        USART1_Write(adcVal & 0x00FF);
        USART1_Write(adcVal >> 8);
        USART1_Write(END_TOKEN);
    }
}
```

例 13-2. ADC 自由运行转换代码示例

```
/*
    \文件   main.c
    \简述   ADC 自由运行
    (c) 2019 Microchip Technology Inc.及其子公司。
    在遵守这些条款的前提下，您可以将 Microchip 软件和任何衍生产品专门用于 Microchip 产品。您有责任遵守适用于您使用可能附带 Microchip 软件的第三方软件（包括开源软件）时的第三方许可条款。
    本软件由 Microchip“按原样”提供。本软件不作任何明示、暗示或法定的保证，包括对非侵权、适销性和适销性的任何默示保证。
    在任何情况下，Microchip 均不对与软件相关的任何类型的间接、特殊、惩罚性、附带或后果性损失、损害、成本或费用负责，即使是 Microchip 已被告知这种可能性或损害可以预见。在法律允许的最大范围内，Microchip 对与本软件有关的所有索赔的全部责任不会超过您为本软件直接支付给 Microchip 的费用（如果有）。
```

```
*/
#define F_CPU           4000000UL   /* 主时钟频率 */
#define START_TOKEN     0x03        /* 开始帧令牌 */
#define END_TOKEN       0xFC        /* 结束帧令牌 */
/* 计算波特率 */
#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);
/* 此函数初始化 CLKCTRL 模块 */
void CLKCTRL_init(void)
{
    /* FREQSEL 4M */
    ccp_write_io((void*)&(CLKCTRL.OSCHFCTRLA), (CLKCTRL.OSCHFCTRLA |
CLKCTRL_FREQSEL_4M_gc));
}
/* 此函数初始化 PORT 模块 */
void PORT_init(void)
{
    /* 将 PC0 配置为 USART1 TX 的输出 */
    PORTC.DIRSET = PIN0_bm;

    /* 禁止 PD3 上的中断和数字输入缓冲器 */
    PORTD.PIN3CTRL &= ~PORT_ISC_gm;
    PORTD.PIN3CTRL |= PORT_ISC_INPUT_DISABLE_gc;
    /* 禁止所有上拉电阻 */
    PORTD.PIN3CTRL &= ~PORT_PULLUPEN_bm;
}
/* 此函数初始化 VREF 模块 */
void VREF0_init(void)
{
    VREF.ADC0REF = VREF_REFSEL_2V048_gc;  /* 2.048V 内部参考 */
}
/* 此函数初始化 ADC 模块 */
void ADC0_init(void)
{
    ADC0.CTRLC = ADC_PRESC_DIV4_gc;        /* CLK_PER 进行 4 分频 */
    ADC0.CTRLA = ADC_ENABLE_bm             /* ADC 使能：已使能 */
               | ADC_RESSEL_12BIT_gc;      /* 12 位模式 */
               | ADC_FREERUN_bm;           /* 使能自由运行模式 */
    ADC0.MUXPOS = ADC_MUXPOS_AIN3_gc;      /* 选择 ADC 通道 AIN3 <-> PD3 */
}
/* 此函数初始化 USART 模块 */
void USART1_init(void)
{
    /* 配置波特率：115200 */
    USART1.BAUD = (uint16_t)USART1_BAUD_RATE(115200);
    USART1.CTRLB = USART_TXEN_bm;           /* 使能 TX */
    USART1.CTRLC = USART_CHSIZE_8BIT_gc;    /* 配置字符大小：8 位 */
}
/* 此函数返回 ADC 转换结果 */
uint16_t ADC0_read(void)
{
    /* 等待 ADC 结果就绪 */
    while (!(ADC0.INTFLAGS & ADC_RESRDY_bm));
    /* 通过读取结果来清零中断标志 */
    return ADC0.RES;
}
/* 此函数启动 ADC 转换 */
void ADC0_start(void)
{
    /* 启动 ADC 转换 */
    ADC0.COMMAND = ADC_STCONV_bm;
```

```
}
/* 此函数通过 USART 传输一个字节 */
void USART1_Write(const uint8_t data)
{
    /* 检查 USART 缓冲区是否已做好发送准备 */
    while (!(USART1.STATUS & USART_DREIF_bm));
    /* 使用 TXDATAL 寄存器发送数据 */
    USART1.TXDATAL = data;
}
int main(void)
{
    uint16_t adcVal;

    /* 初始化所有外设 */
    CLKCTRL_init();
    PORT_init();
    VREF0_init();
    ADC0_init();
    USART1_init();

    /* 启动 ADC 转换 */
    ADC0_start();

    while (1)
    {
        /* 读取 ADC 结果 */
        adcVal = ADC0_read();

        /* 发送要使用 Data Visualizer 绘制的 ADC 结果 */
        USART1_Write(START_TOKEN);
        USART1_Write(adcVal & 0x00FF);
        USART1_Write(adcVal >> 8);
        USART1_Write(END_TOKEN);
    }
}
```

例 13-3. ADC 差分转换代码示例

```
/*
    \文件   main.c
    \简述   ADC 差分转换
    (c) 2019 Microchip Technology Inc.及其子公司。
    在遵守这些条款的前提下，您可以将 Microchip 软件和任何衍生产品专门用于 Microchip 产品。您有责任遵守适用于您使用可能附带 Microchip 软件的第三方软件（包括开源软件）时的第三方许可条款。
    本软件由 Microchip “按原样”提供。本软件不作任何明示、暗示或法定的保证，包括对非侵权、适销性和适销性的任何默示保证。
    在任何情况下，Microchip 均不对与软件相关的任何类型的间接、特殊、惩罚性、附带或后果性损失、损害、成本或费用负责，即使是 Microchip 已被告知这种可能性或损害可以预见。在法律允许的最大范围内，Microchip 对与本软件有关的所有索赔的全部责任不会超过您为本软件直接支付给 Microchip 的费用（如果有）。
*/
#define F_CPU           4000000UL   /* 主时钟频率 */
#define START_TOKEN     0x03        /* 开始帧令牌 */
#define END_TOKEN       0xFC        /* 结束帧令牌 */
/* 计算波特率 */
#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)
{
    /* 将 PC0 配置为 USART1 TX 的输出 */
    PORTC.DIRSET = PIN0_bm;

    /* 禁止 PD3 上的中断和数字输入缓冲器 */
    PORTD.PIN3CTRL &= ~PORT_ISC_gm;
    PORTD.PIN3CTRL |= PORT_ISC_INPUT_DISABLE_gc;
    /* 禁止 PD4 上的中断和数字输入缓冲器 */
    PORTD.PIN4CTRL &= ~PORT_ISC_gm;
    PORTD.PIN4CTRL |= PORT_ISC_INPUT_DISABLE_gc;

    /* 禁止所有上拉电阻 */
    PORTD.PIN3CTRL &= ~PORT_PULLUPEN_bm;
    PORTD.PIN4CTRL &= ~PORT_PULLUPEN_bm;
}
void VREF0_init(void)
{
    VREF.ADC0REF = VREF_REFSEL_2V048_gc;  /* 2.048V 内部参考 */
}
void ADC0_init(void)
{
    ADC0.CTRLC = ADC_PRESC_DIV4_gc;       /* CLK_PER 进行 4 分频 */
    ADC0.CTRLA = ADC_ENABLE_bm            /* ADC 使能：已使能 */
               | ADC_RESSEL_12BIT_gc      /* 12 位模式 */
               | ADC_CONVMODE_bm          /* 差分转换 */
               | ADC_FREERUN_bm;          /* 使能自由运行模式 */
    ADC0.MUXPOS = ADC_MUXPOS_AIN3_gc;     /* 选择 ADC 通道 AIN3 <-> PD3 */
    ADC0.MUXNEG = ADC_MUXNEG_AIN4_gc;     /* 选择 ADC 通道 AIN4 <-> PD4 */
}
void USART1_init(void)
{
    /* 配置波特率：115200 */
    USART1.BAUD = (uint16_t)USART1_BAUD_RATE(115200);
    USART1.CTRLB = USART_TXEN_bm;           /* 使能 TX */
    USART1.CTRLC = USART_CHSIZE_8BIT_gc;    /* 配置字符大小：8 位 */
}
int16_t ADC0_read(void)
{
    /* 通过读取结果来清零中断标志 */
    return ADC0.RES;
}
void ADC0_start(void)
{
    /* 启动转换 */
    ADC0.COMMAND = ADC_STCONV_bm;
}
bool ADC0_conversionDone(void)
{
    /* 检查转换是否完成 */
    return (ADC0.INTFLAGS & ADC_RESRDY_bm);
}
void USART1_Write(const uint8_t data)
{
    /* 检查 USART 缓冲区是否已做好发送准备 */
    while (!(USART1.STATUS & USART_DREIF_bm));
    /* 使用 TXDATAL 寄存器发送数据 */
    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())
        {
            /* 读取 ADC 结果 */
            adcVal = ADC0_read();
            /* 发送要使用 Data Visualizer 绘制的 ADC 结果 */
            USART1_Write(START_TOKEN);
            USART1_Write(adcVal & 0x00FF);
            USART1_Write(adcVal >> 8);
            USART1_Write(END_TOKEN);
        }
    }
}
```

例 13-4. ADC 采样累加器代码示例

```
/*
    \文件   main.c
    \简述   ADC 采样累加器
    (c) 2019 Microchip Technology Inc.及其子公司。
    在遵守这些条款的前提下，您可以将 Microchip 软件和任何衍生产品专门用于 Microchip 产品。您有责任遵守适用于您使用可能附带 Microchip 软件的第三方软件（包括开源软件）时的第三方许可条款。
    本软件由 Microchip“按原样”提供。本软件不作任何明示、暗示或法定的保证，包括对非侵权、适销性和适销性的任何默示保证。
    在任何情况下，Microchip 均不对与软件相关的任何类型的间接、特殊、惩罚性、附带或后果性损失、损害、成本或费用负责，即使是 Microchip 已被告知这种可能性或损害可以预见。在法律允许的最大范围内，Microchip 对与本软件有关的所有索赔的全部责任不会超过您为本软件直接支付给 Microchip 的费用（如果有）。
*/
#define ADC_SHIFT_DIV16     (4)          /* 4 代表 2^4 = 16 */
#define F_CPU               4000000UL    /* 主时钟频率 */
#define START_TOKEN         0x03         /* 开始帧令牌 */
#define END_TOKEN           0xFC         /* 结束帧令牌 */
/* 计算波特率 */
#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);
/* 此函数初始化 CLKCTRL 模块 */
void CLKCTRL_init(void)
{
    /* FREQSEL 4M */
    ccp_write_io((void*)&(CLKCTRL.OSCHFCTRLA), (CLKCTRL.OSCHFCTRLA |
CLKCTRL_FREQSEL_4M_gc));
}
/* 此函数初始化 PORT 模块 */
void PORT_init(void)
{
    /* 将 PC0 配置为 USART1 TX 的输出 */
    PORTC.DIRSET = PIN0_bm;

    /* 禁止 PD3 上的中断和数字输入缓冲器 */
    PORTD.PIN3CTRL &= ~PORT_ISC_gm;
```

```
    PORTD.PIN3CTRL |= PORT_ISC_INPUT_DISABLE_gc;

    /* 禁止所有上拉电阻 */
    PORTD.PIN3CTRL &= ~PORT_PULLUPEN_bm;
}
/* 此函数初始化 VREF 模块 */
void VREF0_init(void)
{
    VREF.ADC0REF = VREF_REFSEL_2V048_gc;  /* 2.048V 内部参考 */
}
/* 此函数初始化 ADC 模块 */
void ADC0_init(void)
{
    ADC0.CTRLC = ADC_PRESC_DIV4_gc;        /* CLK_PER 进行 4 分频 */
    ADC0.CTRLA = ADC_ENABLE_bm             /* ADC 使能：已使能 */
               | ADC_RESSEL_12BIT_gc;      /* 12 位模式 */
               | ADC_FREERUN_bm;           /* 使能自由运行模式 */
    ADC0.MUXPOS = ADC_MUXPOS_AIN3_gc;      /* 选择 ADC 通道 AIN3 <-> PD3 */
    ADC0.CTRLB = ADC_SAMPNUM_ACC64_gc;     /* 设置为累加 64 个采样 */
}
/* 此函数初始化 USART 模块 */
void USART1_init(void)
{
    /* 配置波特率：115200 */
    USART1.BAUD = (uint16_t)USART1_BAUD_RATE(115200);
    USART1.CTRLB = USART_TXEN_bm;           /* 使能 TX */
    USART1.CTRLC = USART_CHSIZE_8BIT_gc;    /* 配置字符大小：8 位 */
}
/* 此函数返回 ADC 转换结果 */
uint16_t ADC0_read(void)
{
    /* 等待 ADC 结果就绪 */
    while (!(ADC0.INTFLAGS & ADC_RESRDY_bm));
    /* 通过读取结果来清零中断标志 */
    return ADC0.RES;
}
/* 此函数启动 ADC 转换 */
void ADC0_start(void)
{
    /* 启动 ADC 转换 */
    ADC0.COMMAND = ADC_STCONV_bm;
}
/* 此函数通过 USART 传输一个字节 */
void USART1_Write(const uint8_t data)
{
    /* 检查 USART 缓冲区是否已做好发送准备 */
    while (!(USART1.STATUS & USART_DREIF_bm));
    /* 使用 TXDATAL 寄存器发送数据 */
    USART1.TXDATAL = data;
}
int main(void)
{
    uint16_t adcVal;

    /* 初始化所有外设 */
    CLKCTRL_init();
    PORT_init();
    VREF0_init();
    ADC0_init();
    USART1_init();

    /* 启动 ADC 转换 */
    ADC0_start();

    while (1)
    {
        /* 读取 ADC 结果 */
        adcVal = ADC0_read();
        /* 如果采样数 > 16，则除以采样数（16） */
        adcVal = adcVal >> ADC_SHIFT_DIV16;
        /* 发送要使用 Data Visualizer 绘制的 ADC 结果 */
        USART1_Write(START_TOKEN);
```

```
        USART1_Write(adcVal & 0x00FF);
        USART1_Write(adcVal >> 8);
        USART1_Write(END_TOKEN);
    }
}
```

例 13-5. ADC 窗口比较器代码示例

```
/*
    \文件   main.c

    \简述  ADC 窗口比较器

    (c) 2019 Microchip Technology Inc.及其子公司。

    在遵守这些条款的前提下，您可以将 Microchip 软件和任何衍生产品专门用于 Microchip 产品。您有责任遵守适用于您使用可能附带 Microchip 软件的第三方软件（包括开源软件）时的第三方许可条款。

    本软件由 Microchip“按原样”提供。本软件不作任何明示、暗示或法定的保证，包括对非侵权、适销性和适销性的任何默示保证。

    在任何情况下，Microchip 均不对与软件相关的任何类型的间接、特殊、惩罚性、附带或后果性损失、损害、成本或费用负责，即使是 Microchip 已被告知这种可能性或损害可以预见。在法律允许的最大范围内，Microchip 对与本软件有关的所有索赔的全部责任不会超过您为本软件直接支付给 Microchip 的费用（如果有）。
*/

#define WINDOW_CMP_LOW_TH_EXAMPLE       (0x100)
#define F_CPU                           4000000UL   /* 主时钟频率 */
#define START_TOKEN                     0x03        /* 开始帧令牌 */
#define END_TOKEN                       0xFC        /* 结束帧令牌 */
/* 计算波特率 */
#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);

/* 此函数初始化 CLKCTR 模块 */
void CLKCTRL_init(void)
{
    /* FREQSEL 4M */
    ccp_write_io((void*)&(CLKCTRL.OSCHFCTRLA), (CLKCTRL.OSCHFCTRLA |
CLKCTRL_FREQSEL_4M_gc));
}

/* 此函数初始化 PORT 模块 */
void PORT_init(void)
{
    /* 将 PC0 配置为 USART1 TX 的输出 */
    PORTC.DIRSET = PIN0_bm;

    /* 禁止 PD3 上的中断和数字输入缓冲器 */
    PORTD.PIN3CTRL &= ~PORT_ISC_gm;
    PORTD.PIN3CTRL |= PORT_ISC_INPUT_DISABLE_gc;
```

```
    /* 禁止所有上拉电阻 */
    PORTD.PIN3CTRL &= ~PORT_PULLUPEN_bm;
}

/* 此函数初始化 VREF 模块 */
void VREF0_init(void)
{
    VREF.ADC0REF = VREF_REFSEL_2V048_gc;    /* 2.048V 内部参考 */
}

/* 此函数初始化 ADC 模块 */
void ADC0_init(void)
{
    ADC0.CTRLC = ADC_PRESC_DIV4_gc;         /* CLK_PER 进行 4 分频 */
    ADC0.CTRLA = ADC_ENABLE_bm              /* ADC 使能：已使能 */
                | ADC_RESSEL_12BIT_gc       /* 12 位模式 */
                | ADC_FREERUN_bm;           /* 使能自由运行模式 */
    ADC0.MUXPOS = ADC_MUXPOS_AIN3_gc;       /* 选择 ADC 通道 AIN3 <-> PD3 */
    ADC0.WINLT = WINDOW_CMP_LOW_TH_EXAMPLE; /* 设置转换窗口比较器下限阈值 */
    ADC0.CTRLE = ADC_WINCM_BELOW_gc;        /* 设置转换窗口模式 */
}

/* 此函数初始化 USART 模块 */
void USART1_init(void)
{
    /* 配置波特率：115200 */
    USART1.BAUD = (uint16_t)USART1_BAUD_RATE(115200);
    USART1.CTRLB = USART_TXEN_bm;           /* 使能 TX */
    USART1.CTRLC = USART_CHSIZE_8BIT_gc;    /* 配置字符大小：8 位 */
}

/* 此函数返回 ADC 转换结果 */
uint16_t ADC0_read(void)
{
    /* 通过读取结果来清零中断标志 */
    return ADC0.RES;
}

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

/* 此函数启动 ADC 转换 */
void ADC0_start(void)
{
    /* 启动转换 */
    ADC0.COMMAND = ADC_STCONV_bm;
}

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

/* 此函数通过 USART 传输一个字节 */
void USART1_Write(const uint8_t data)
{
    /* 检查 USART 缓冲区是否已做好发送准备 */
    while (!(USART1.STATUS & USART_DREIF_bm));
    /* 使用 TXDATAL 寄存器发送数据 */
    USART1.TXDATAL = data;
}

/* 此函数初始化 LED 引脚 */
void LED0_init(void)
{
    /* 将引脚配置为输出 */
    PORTC.DIRSET = PIN6_bm;
    /* 将输出配置为高电平（LED == 熄灭） */
    PORTC.OUTSET = PIN6_bm;
}
```

```
void LED0_on(void)
{
    /* 将输出配置为低电平（LED == 点亮） */
    PORTC.OUTCLR = PIN6_bm;
}

void LED0_off(void)
{
    /* 将输出配置为高电平（LED == 熄灭） */
    PORTC.OUTSET = PIN6_bm;
}

int main(void)
{
    uint16_t adcVal;

    /* 初始化所有外设 */
    CLKCTRL_init();
    PORT_init();
    VREF0_init();
    ADC0_init();
    USART1_init();
    LED0_init();

    /* 启动 ADC 转换 */
    ADC0_start();

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

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

        adcVal = ADC0_read();

        /* 发送要使用 Data Visualizer 绘制的 ADC 结果 */
        USART1_Write(START_TOKEN);
        USART1_Write(adcVal & 0x00FF);
        USART1_Write(adcVal >> 8);
        USART1_Write(END_TOKEN);
    }
}
```

例 13-6. ADC 事件触发代码示例

```
/*
    \文件   main.c
    \简介   RTC 溢出触发的 ADC 事件
    (c) 2019 Microchip Technology Inc.及其子公司。
    在遵守这些条款的前提下，您可以将 Microchip 软件和任何衍生产品专门用于 Microchip 产品。您有责任遵守适用于您使用可能附带 Microchip 软件的第三方软件（包括开源软件）时的第三方许可条款。
    本软件由 Microchip“按原样”提供。本软件不作任何明示、暗示或法定的保证，包括对非侵权、适销性和适销性的任何默示保证。
    在任何情况下，Microchip 均不对与软件相关的任何类型的间接、特殊、惩罚性、附带或后果性损失、损害、成本或费用负责，即使是 Microchip 已被告知这种可能性或损害可以预见。在法律允许的最大范围内，Microchip 对与本软件有关的所有索赔的全部责任不会超过您为本软件直接支付给 Microchip 的费用（如果有）。
*/
/* RTC 周期 */
#define RTC_PERIOD      (511)       /* RTC 周期 */
#define F_CPU           4000000UL   /* 主时钟频率 */
#define START_TOKEN     0x03        /* 开始帧令牌 */
```

```
#define END_TOKEN       0xFC        /* 结束帧令牌 */
/* 计算波特率 */
#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);
/* 此函数初始化 CLKCTRL 模块 */
void CLKCTRL_init(void)
{
    /* FREQSEL 4M */
    ccp_write_io((void*)&(CLKCTRL.OSCHFCTRLA), (CLKCTRL.OSCHFCTRLA |
CLKCTRL_FREQSEL_4M_gc));
}
/* 此函数初始化 PORT 模块 */
void PORT_init(void)
{
    /* 将 PC0 配置为 USART1 TX 的输出 */
    PORTC.DIRSET = PIN0_bm;

    /* 禁止 PD3 上的中断和数字输入缓冲器 */
    PORTD.PIN3CTRL &= ~PORT_ISC_gm;
    PORTD.PIN3CTRL |= PORT_ISC_INPUT_DISABLE_gc;

    /* 禁止上拉电阻 */
    PORTD.PIN3CTRL &= ~PORT_PULLUPEN_bm;
}
/* 此函数初始化 VREF 模块 */
void VREF0_init(void)
{
    VREF.ADC0REF = VREF_REFSEL_2V048_gc;  /* 2.048V 内部参考 */
}
/* 此函数初始化 VREF 模块 */
void ADC0_init(void)
{
    ADC0.CTRLC = ADC_PRESC_DIV4_gc;         /* CLK_PER 进行 4 分频 */
    ADC0.CTRLA = ADC_ENABLE_bm              /* ADC 使能：已使能 */
               | ADC_RESSEL_12BIT_gc;       /* 12 位模式 */
    ADC0.MUXPOS = ADC_MUXPOS_AIN3_gc;       /* 选择 ADC 通道 AIN3 <-> PD3 */
    ADC0.INTCTRL |= ADC_RESRDY_bm;          /* 允许中断 */
    ADC0.EVCTRL |= ADC_STARTEI_bm;          /* 使能事件触发的转换 */
}
/* 此函数初始化 LED 引脚 */
void LED0_init(void)
{
    /* 将引脚配置为输出 */
    PORTC.DIRSET = PIN6_bm;
    /* 将输出配置为高电平（LED == 熄灭） */
    PORTC.OUTSET = PIN6_bm;
}
/* 此函数初始化 USART 模块 */
void USART1_init()
{
    /* 配置波特率：115200 */
    USART1.BAUD = (uint16_t)USART1_BAUD_RATE(115200);
    USART1.CTRLB = USART_TXEN_bm;            /* 使能 TX */
    USART1.CTRLC = USART_CHSIZE_8BIT_gc;     /* 配置字符大小：8 位 */
}
/* 此函数翻转 LED 引脚 */
void LED0_toggle(void)
```

```
{
    PORTC.OUTTGL = PIN6_bm;
}
/* 此函数通过 USART 传输一个字节 */
void USART1_Write(const uint8_t data)
{
    /* 检查 USART 缓冲区是否已做好发送准备 */
    while (!(USART1.STATUS & USART_DREIF_bm));
    /* 使用 TXDATAL 寄存器发送数据 */
    USART1.TXDATAL = data;
}
ISR(ADC0_RESRDY_vect)
{
    /* 通过读取结果来清零中断标志 */
    adcVal = ADC0.RES;
 /* 更新标志 */
 adcResultReady = 1;
}
/* 此函数初始化 RTC 模块  */
void RTC_init(void)
{
    /* 初始化 RTC: */
    while (RTC.STATUS > 0)
    {
        ; /* 等待所有寄存器均同步 */
    }
    RTC.CTRLA = RTC_PRESCALER_DIV32_gc  /* 32 */
              | RTC_RTCEN_bm            /* 使能：已使能 */
              | RTC_RUNSTDBY_bm;        /* 在待机模式下运行：已使能 */
    RTC.PER = RTC_PERIOD;               /* 设置周期 */
    RTC.CLKSEL = RTC_CLKSEL_OSC32K_gc;  /* 32.768kHz 内部晶振（OSC32K） */
}
/* 此函数初始化 EVSYS 模块  */
void EVSYS_init(void)
{
    /* 实时计数器溢出 */
    EVSYS.CHANNEL0 = EVSYS_CHANNEL0_RTC_OVF_gc;
    /* 将用户连接到事件通道 0 */
    EVSYS.USERADC0START = EVSYS_USER_CHANNEL0_gc;
}
int main(void)
{
    uint16_t result;

    /* 初始化所有外设 */
    CLKCTRL_init();
    PORT_init();
    VREF0_init();
    ADC0_init();
    LED0_init();
    USART1_init();
    RTC_init();
    EVSYS_init();

    /* 允许全局中断 */
    sei();

    while (1)
    {
        if (adcResultReady == 1)
  {
            /* 存储结果以避免由于中断而改变 adcVal。此操作必须为原子操作 */
            cli();
            result = adcVal;
            sei();

   /* 更新标志值 */
   adcResultReady = 0;
   /* 翻转 LED 状态 */
   LED0_toggle();

   /* 发送要使用 Data Visualizer 绘制的 ADC 结果 */
   USART1_Write(START_TOKEN);
```

```
    USART1_Write(result & 0x00FF);
    USART1_Write(result >> 8);
    USART1_Write(END_TOKEN);
   }
     }
}
```

例 13-7. ADC 温度测量代码示例

```
/*
    \文件   main.c
    \简述   ADC 温度测量
    (c) 2019 Microchip Technology Inc.及其子公司。
    在遵守这些条款的前提下，您可以将 Microchip 软件和任何衍生产品专门用于 Microchip 产品。您有责任遵守适用于您使用可能附带 Microchip 软件的第三方软件（包括开源软件）时的第三方许可条款。
    本软件由 Microchip“按原样”提供。本软件不作任何明示、暗示或法定的保证，包括对非侵权、适销性和适销性的任何默示保证。
    在任何情况下，Microchip 均不对与软件相关的任何类型的间接、特殊、惩罚性、附带或后果性损失、损害、成本或费用负责，即使是 Microchip 已被告知这种可能性或损害可以预见。在法律允许的最大范围内，Microchip 对与本软件有关的所有索赔的全部责任不会超过您为本软件直接支付给 Microchip 的费用（如果有）。
*/
#define F_CPU           4000000UL   /* 主时钟频率 */
#define START_TOKEN     0x03        /* 开始帧令牌 */
#define END_TOKEN       0xFC        /* 结束帧令牌 */
/* 计算波特率 */
#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);
/* 此函数初始化 CLKCTRL 模块 */
void CLKCTRL_init(void)
{
    /* FREQSEL 4M */
    ccp_write_io((void*)&(CLKCTRL.OSCHFCTRLA), (CLKCTRL.OSCHFCTRLA |
CLKCTRL_FREQSEL_4M_gc));
}
/* 此函数初始化 PORT 模块 */
void PORT_init(void)
{
    /* 将 PC0 配置为 USART1 TX 的输出 */
    PORTC.DIRSET = PIN0_bm;
}
/* 此函数初始化 VREF 模块 */
void VREF0_init(void)
{
    VREF.ADC0REF = VREF_REFSEL_2V048_gc;  /* 2.048V 内部参考 */
}
/* 此函数初始化 ADC 模块 */
void ADC0_init(void)
{
    ADC0.CTRLC = ADC_PRESC_DIV4_gc;         /* CLK_PER 进行 4 分频 */
    ADC0.CTRLA = ADC_ENABLE_bm              /* ADC 使能：已使能 */
                | ADC_RESSEL_12BIT_gc       /* 12 位模式 */
                | ADC_FREERUN_bm;           /* 自由运行模式 */
    ADC0.MUXPOS = ADC_MUXPOS_TEMPSENSE_gc;  /* 选择 ADC 通道，温度传感器*/
}
/* 此函数初始化 USART 模块 */
void USART1_init(void)
```

```
{
    /* 配置波特率：115200 */
    USART1.BAUD = (uint16_t)USART1_BAUD_RATE(115200);
    USART1.CTRLB = USART_TXEN_bm;           /* 使能 TX */
    USART1.CTRLC = USART_CHSIZE_8BIT_gc;    /* 配置字符大小：8 位 */
}
/* 此函数返回 ADC 转换结果 */
uint16_t ADC0_read(void)
{
    /* 等待 ADC 结果就绪 */
    while (!(ADC0.INTFLAGS & ADC_RESRDY_bm));
    /* 通过读取结果来清零中断标志 */
    return ADC0.RES;
}
/* 此函数返回以摄氏度为单位的温度值 */
int16_t temperatureConvert(uint16_t data)
{
    uint16_t sigrow_offset = SIGROW.TEMPSENSE1;
    uint16_t sigrow_slope = SIGROW.TEMPSENSE0;
    int32_t temp;
    /* 通过读取结果（ADC0.RES）来清零中断标志 */
    temp = sigrow_offset - data;
    /* 变量超过 16 位时，结果将溢出 */
    temp *= sigrow_slope;
    /* 加上 4096/2 以对下面的除法进行正确的舍入 */
    temp += 0x0800;
    /* 除以 2^12（4096），舍入为最接近的开氏度 */
    temp >>= 12;
    /* 从开氏度转换为摄氏度（0K - 273.15 = -273.1°C） */
    return temp - 273;
}
/* 此函数启动 ADC 转换 */
void ADC0_start(void)
{
    /* 启动转换 */
    ADC0.COMMAND = ADC_STCONV_bm;
}
/* 此函数通过 USART 传输一个字节 */
void USART1_Write(const uint8_t data)
{
    /* 检查 USART 缓冲区是否已做好发送准备 */
    while (!(USART1.STATUS & USART_DREIF_bm));
    /* 使用 TXDATAL 寄存器发送数据 */
    USART1.TXDATAL = data;
}
int main(void)
{
    int16_t temp_C;
    uint16_t adcVal;

    /* 初始化所有外设 */
    CLKCTRL_init();
    PORT_init();
    VREF0_init();
    ADC0_init();
    USART1_init();

    /* 启动 ADC 转换 */
    ADC0_start();

    while (1)
    {
        /* 读取转换结果 */
        adcVal = ADC0_read();
        /* 将 ADC 结果转换为摄氏度 */
        temp_C = temperatureConvert(adcVal);

        /*发送要使用 Data Visualizer 绘制的 ADC 结果 */
        USART1_Write(START_TOKEN);
        USART1_Write(temp_C & 0x00FF);
        USART1_Write(temp_C >> 8);
        USART1_Write(END_TOKEN);
```

```
    }
}
```

14. 版本历史

文档版本	日期	备注
D	2020 年 10 月	更新了 GitHub 徽标。 添加了 Data Visualizer 功能描述和步骤。 更新了发送 ADC 转换结果的代码示例。
C	2020 年 5 月	根据最新商标，将 AVR® MCU DA (AVR-DA)更新为 AVR® DA MCU，AVR-DA 更新为 AVR DA。
B	2020 年 3 月	更新了资源库链接。 根据最新商标，将 AVR-DA 更新为 AVR MCU DA (AVR-DA)。
A	2020 年 2 月	文档初始版本。

Microchip 网站

Microchip 网站（www.microchip.com/）为客户提供在线支持。客户可通过该网站方便地获取文件和信息。我们的网站提供以下内容：

- **产品支持**——数据手册和勘误表、应用笔记和示例程序、设计资源、用户指南以及硬件支持文档、最新的软件版本以及归档软件
- **一般技术支持**——常见问题解答（FAQ）、技术支持请求、在线讨论组以及 Microchip 设计伙伴计划成员名单
- **Microchip 业务**——产品选型和订购指南、最新 Microchip 新闻稿、研讨会和活动安排表、Microchip 销售办事处、代理商以及工厂代表列表

产品变更通知服务

Microchip 的产品变更通知服务有助于客户了解 Microchip 产品的最新信息。注册客户可在他们感兴趣的某个产品系列或开发工具发生变更、更新、发布新版本或勘误表时，收到电子邮件通知。

欲注册，请访问 www.microchip.com/pcn，然后按照注册说明进行操作。

客户支持

Microchip 产品的用户可通过以下渠道获得帮助：

- 代理商或代表
- 当地销售办事处
- 应用工程师（ESE）
- 技术支持

客户应联系其代理商、代表或 ESE 寻求支持。当地销售办事处也可为客户提供帮助。本文档后附有销售办事处的联系方式。

也可通过 www.microchip.com/support 获得网上技术支持。

Microchip 器件代码保护功能

请注意以下有关 Microchip 器件代码保护功能的要点：

- Microchip 的产品均达到 Microchip 数据手册中所述的技术规范。
- Microchip 确信：在正常使用的情况下，Microchip 系列产品非常安全。
- 目前，仍存在着用恶意、甚至是非法的方法来试图破坏代码保护功能的行为。我们确信，所有这些行为都不是以 Microchip 数据手册中规定的操作规范来使用 Microchip 产品的。这种试图破坏代码保护功能的行为极可能侵犯 Microchip 的知识产权。
- Microchip 愿与那些注重代码完整性的客户合作。
- Microchip 或任何其他半导体厂商均无法保证其代码的安全性。代码保护并不意味着我们保证产品是“牢不可破”的。代码保护功能处于持续发展中。Microchip 承诺将不断改进产品的代码保护功能。任何试图破坏 Microchip 代码保护功能的行为均可视为违反了《数字器件千年版权法案（Digital Millennium Copyright Act）》。如果这种行为导致他人在未经授权的情况下，能访问您的软件或其他受版权保护的成果，您有权依据该法案提起诉讼，从而制止这种行为。

法律声明

提供本文档的中文版本仅为了便于理解。请勿忽视文档中包含的英文部分，因为其中提供了有关 Microchip 产品性能和使用情况的有用信息。Microchip Technology Inc.及其分公司和相关公司、各级主管与员工及事务代理机构对译文中可能存在的任何差错不承担任何责任。建议参考 Microchip Technology Inc.的英文原版文档。

商标

ISBN： 978-1-5224-8306-9

质量管理体系

有关 Microchip 的质量管理体系的信息，请访问 www.microchip.com/quality。

全球销售及服务网点

美洲	亚太地区	亚太地区	欧洲
公司总部 2355 West Chandler Blvd. Chandler, AZ 85224-6199 电话：480-792-7200 传真：480-792-7277 技术支持： www.microchip.com/support 网址： www.microchip.com **亚特兰大** 德卢斯，佐治亚州 电话：678-957-9614 传真：678-957-1455 **奥斯汀，德克萨斯州** 电话：512-257-3370 **波士顿** 韦斯特伯鲁，马萨诸塞州 电话：774-760-0087 传真：774-760-0088 **芝加哥** 艾塔斯卡，伊利诺伊州 电话：630-285-0071 传真：630-285-0075 **达拉斯** 阿迪森，德克萨斯州 电话：972-818-7423 传真：972-818-2924 **底特律** 诺维，密歇根州 电话：248-848-4000 **休斯顿，德克萨斯州** 电话：281-894-5983 **印第安纳波利斯** 诺布尔斯维尔，印第安纳州 电话：317-773-8323 传真：317-773-5453 电话：317-536-2380 **洛杉矶** 米慎维荷，加利福尼亚州 电话：949-462-9523 传真：949-462-9608 电话：951-273-7800 **罗利，北卡罗来纳州** 电话：919-844-7510 **纽约，纽约州** 电话：631-435-6000 **圣何塞，加利福尼亚州** 电话：408-735-9110 电话：408-436-4270 **加拿大 - 多伦多** 电话：905-695-1980 传真：905-695-2078	**澳大利亚 - 悉尼** 电话：61-2-9868-6733 **中国 - 北京** 电话：86-10-8569-7000 **中国 - 成都** 电话：86-28-8665-5511 **中国 - 重庆** 电话：86-23-8980-9588 **中国 - 东莞** 电话：86-769-8702-9880 **中国 - 广州** 电话：86-20-8755-8029 **中国 - 杭州** 电话：86-571-8792-8115 **中国 - 香港特别行政区** 电话：852-2943-5100 **中国 - 南京** 电话：86-25-8473-2460 **中国 - 青岛** 电话：86-532-8502-7355 **中国 - 上海** 电话：86-21-3326-8000 **中国 - 沈阳** 电话：86-24-2334-2829 **中国 - 深圳** 电话：86-755-8864-2200 **中国 - 苏州** 电话：86-186-6233-1526 **中国 - 武汉** 电话：86-27-5980-5300 **中国 - 西安** 电话：86-29-8833-7252 **中国 - 厦门** 电话：86-592-2388138 **中国 - 珠海** 电话：86-756-3210040	**印度 - 班加罗尔** 电话：91-80-3090-4444 **印度 - 新德里** 电话：91-11-4160-8631 **印度 - 浦那** 电话：91-20-4121-0141 **日本 - 大阪** 电话：81-6-6152-7160 **日本 - 东京** 电话：81-3-6880-3770 **韩国 - 大邱** 电话：82-53-744-4301 **韩国 - 首尔** 电话：82-2-554-7200 **马来西亚 - 吉隆坡** 电话：60-3-7651-7906 **马来西亚 - 槟榔屿** 电话：60-4-227-8870 **菲律宾 - 马尼拉** 电话：63-2-634-9065 **新加坡** 电话：65-6334-8870 **台湾地区 - 新竹** 电话：886-3-577-8366 **台湾地区 - 高雄** 电话：886-7-213-7830 **台湾地区 - 台北** 电话：886-2-2508-8600 **泰国 - 曼谷** 电话：66-2-694-1351 **越南 - 胡志明市** 电话：84-28-5448-2100	**奥地利 - 韦尔斯** 电话：43-7242-2244-39 传真：43-7242-2244-393 **丹麦 - 哥本哈根** 电话：45-4485-5910 传真：45-4485-2829 **芬兰 - 埃斯波** 电话：358-9-4520-820 **法国 - 巴黎** 电话：33-1-69-53-63-20 传真：33-1-69-30-90-79 **德国 - 加兴** 电话：49-8931-9700 **德国 - 哈恩** 电话：49-2129-3766400 **德国 - 海尔布隆** 电话：49-7131-72400 **德国 - 卡尔斯鲁厄** 电话：49-721-625370 **德国 - 慕尼黑** 电话：49-89-627-144-0 传真：49-89-627-144-44 **德国 - 罗森海姆** 电话：49-8031-354-560 **以色列 - 若那那市** 电话：972-9-744-7705 **意大利 - 米兰** 电话：39-0331-742611 传真：39-0331-466781 **意大利 - 帕多瓦** 电话：39-049-7625286 **荷兰 - 德卢内市** 电话：31-416-690399 传真：31-416-690340 **挪威 - 特隆赫姆** 电话：47-72884388 **波兰 - 华沙** 电话：48-22-3325737 **罗马尼亚 - 布加勒斯特** 电话：40-21-407-87-50 **西班牙 - 马德里** 电话：34-91-708-08-90 传真：34-91-708-08-91 **瑞典 - 哥德堡** 电话：46-31-704-60-40 **瑞典 - 斯德哥尔摩** 电话：46-8-5090-4654 **英国 - 沃金厄姆** 电话：44-118-921-5800 传真：44-118-921-5820