MICROCHIP

TB3210

Getting Started with Digital-to-Analog Converter (DAC)

Introduction

Author: Victor Berzan, Microchip Technology Inc.

The Digital-to-Analog Converter (DAC) converts a digital value written to the Data (DAC.DATA) register to an analog voltage. The output can be connected to a physical pin or used internally. The conversion range is between GND and the selected internal voltage reference (V_{REF}), provided by the Voltage Reference (VREF) peripheral module.

This technical brief describes how the DAC works on tinyAVR® 1-series and AVR® DA microcontrollers (MCUs). It covers the following use cases:

- **Generating Constant Analog Signal:**
  Illustrates how to initialize the DAC, set the voltage reference, set the DAC to output a specific constant voltage
- **Generating Sine Wave Signal:**
  Initializes the DAC, sets the voltage reference, updates the DAC output inside the infinite loop to generate sine wave samples.
- **Reading the DAC Internally with the ADC:**
  Shows how to initialize the DAC and ADC, set the voltage reference, configure the ADC to read the DAC output values. The DAC output voltage is incremented each step, and then it is read using the ADC.
- **Using DAC as Negative Input for AC:**
  Initializes the DAC and AC, configures the AC to use the negative input provided by the DAC, configures the DAC output value. The AC will compare the voltage on its positive input pin with the DAC voltage, and set the output pin to high or low, according to the compare result.

Note: For the first three use cases described in this document, there are two code examples: One bare metal developed on ATtiny817, and one generated with MPLAB® Code Configurator (MCC) developed on AVR128DA48. For the last use case, there is one bare metal code example developed on ATtiny817.

View the ATtiny817 Code Examples on GitHub
Click to browse repository

View the AVR128DA48 Code Examples on GitHub
Click to browse repository

Table of Contents

1. Relevant Devices

This section lists the relevant devices for this document. The following figures show the different family devices, laying out pin count variants and memory sizes:

- Vertical migration upwards is possible without code modification, as these devices are pin-compatible and provide the same or more features. Downward migration on tinyAVR® 1-series devices may require code modification due to fewer available instances of some peripherals.
- Horizontal migration to the left reduces the pin count and, therefore, the available features
- Devices with different Flash memory sizes typically also have different SRAM and EEPROM

Figure 1-1. tinyAVR® 1-series Overview

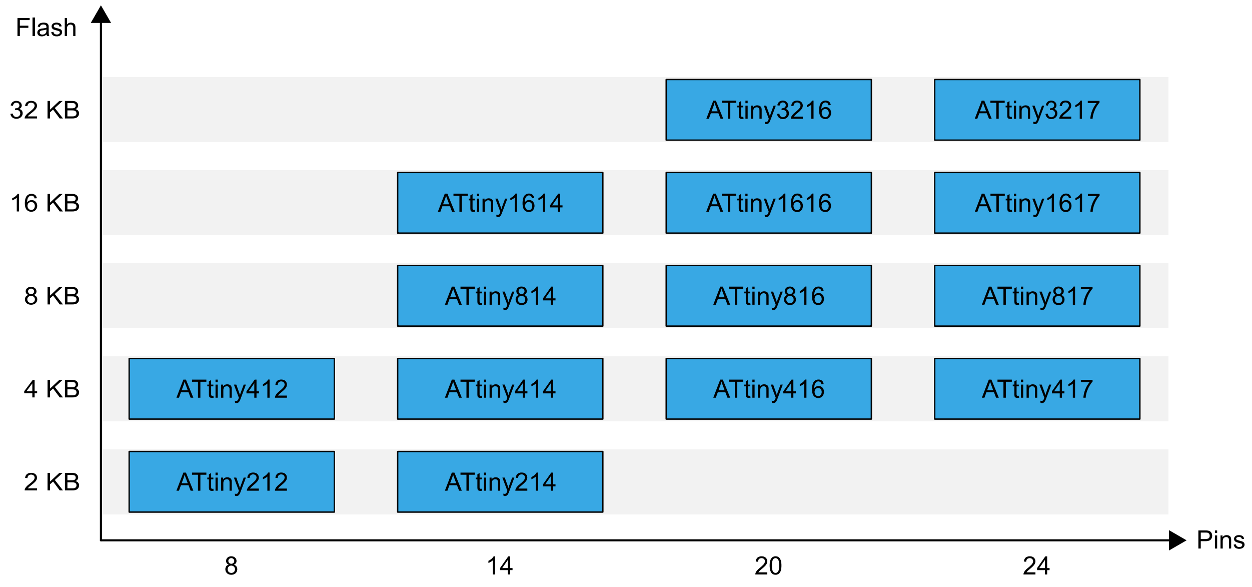
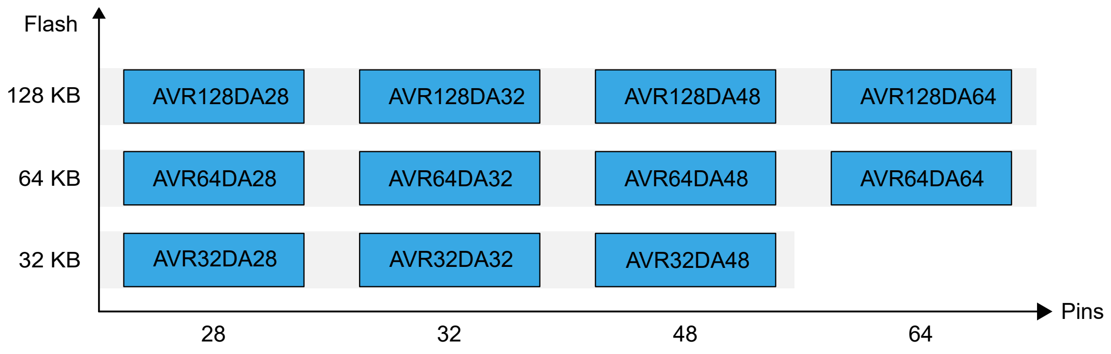


Figure 1-2. AVR® DA Family Overview



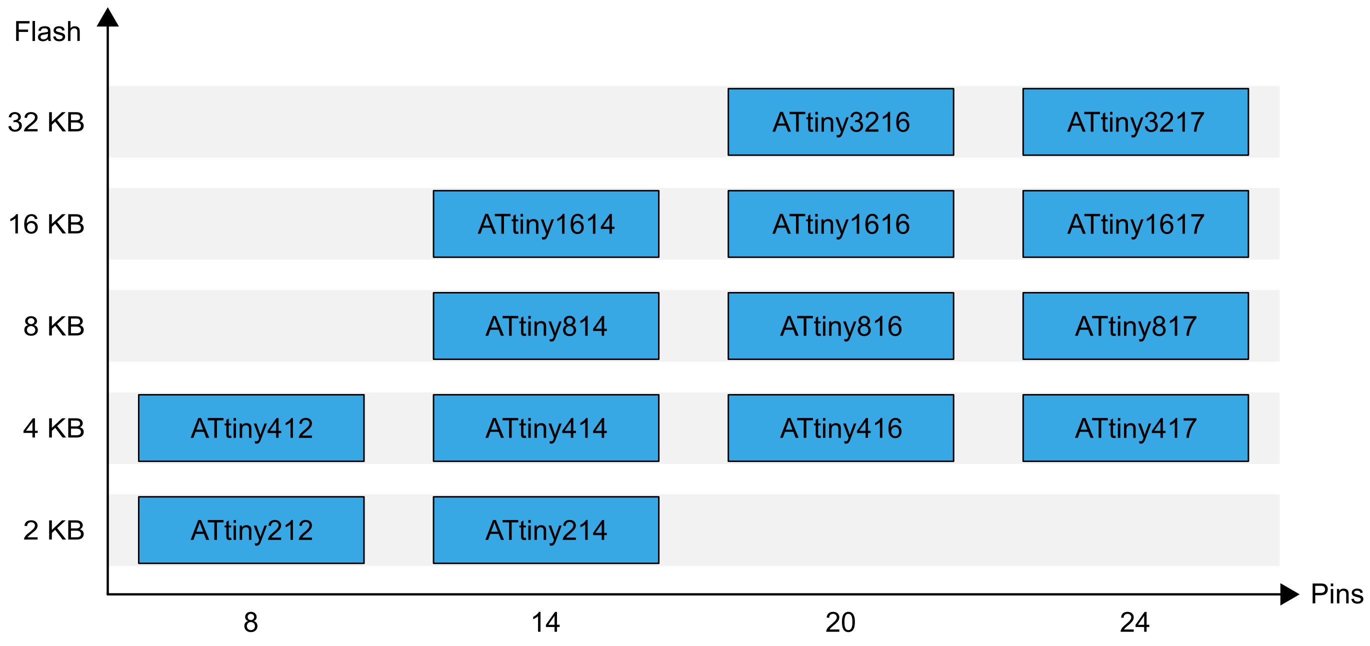
1.1 tinyAVR® 1-series

The following figure shows the tinyAVR 1-series devices, laying out pin count variants and memory sizes:

- Vertical migration upwards is possible without code modification, as these devices are pin-compatible and provide the same or more features. Downward migration may require code modification due to fewer available instances of some peripherals.

- Horizontal migration to the left reduces the pin count and, therefore, the available features

Figure 1-3. tinyAVR® 1-series Overview



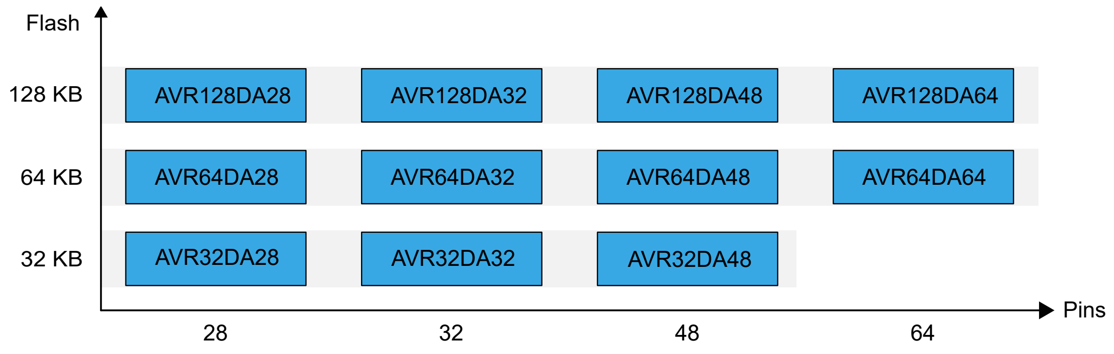
Devices with different Flash memory sizes typically also have different SRAM and EEPROM.

1.2 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 1-4. AVR® DA Family Overview

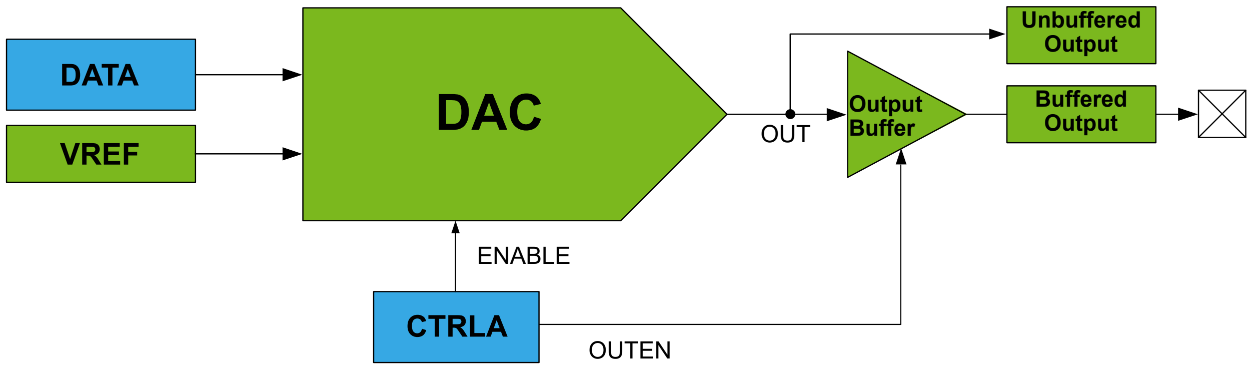


Devices with different Flash memory sizes typically also have different SRAM.

2. Overview

The DAC converts the digital value written to the Data (DACn.DATA) register into an analog voltage that will be available on the DAC output. The conversion range is between GND and the selected voltage reference in the Voltage Reference (VREF) peripheral module. The DAC has one continuous-time output with high drive capabilities. The DAC conversion can be started from the application by writing to the Data (DACn.DATA) register.

Figure 2-1. DAC Block Diagram



3. Generating Constant Analog Signal

The DAC can be used to generate a constant analog signal. The Voltage Reference (VREF) peripheral module is used to provide a voltage reference for the DAC. The DAC output ranges from 0V to $\frac{255 \times V_{REF}}{256}$.

The voltage reference V_{REF} can be selected from a list of predefined values:

Figure 3-1. CTRLA.DAC0REFSEL Values and Associated V_{REF} Voltages

Bit	7	6	5	4	3	2	1	0
		ADC0REFSEL[2:0]				DAC0REFSEL[2:0]		
Access		R/W	R/W	R/W		R/W	R/W	R/W
Reset		0	0	0		0	0	0

bits 2:0 DAC0REFSEL[2:0]: DAC0 and AC0 Reference Select bits
These bits select the reference voltage for the DAC0 and AC0.

Value	Description
`0x0`	0.55V
`0x1`	1.1V
`0x2`	2.5V
`0x3`	4.3V
`0x4`	1.5V
`other`	Reserved

The 4.34V was selected, to have the widest variation range:

```
VREF_CTRLA |= VREF_DAC0REFSEL_4V34_gc;
VREF_CTRLB |= VREF_DAC0REFEN_bm;
```

A 25 µs delay is recommended after configuring the VREF and enabling the voltage reference output. To implement this delay, the `VREF_STARTUP_MICROS` macro definition is used.

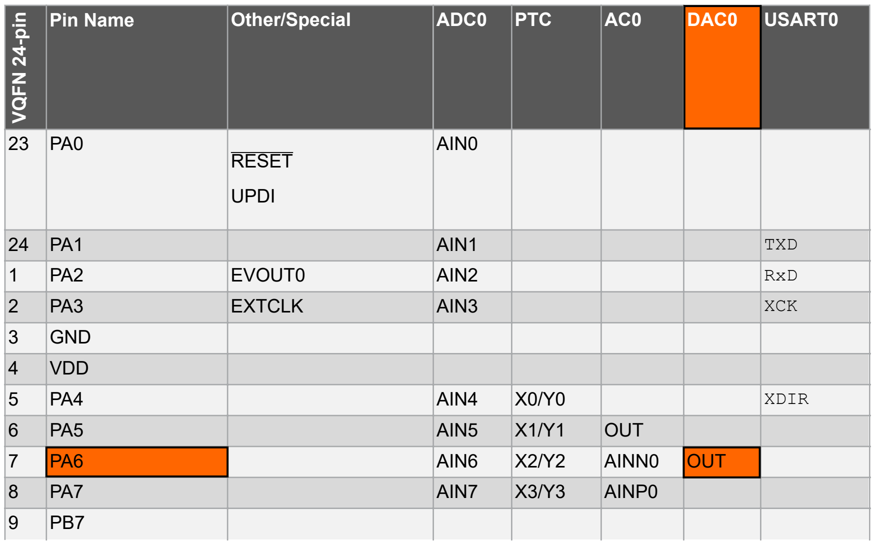
Figure 3-2. Internal Voltage Reference Characteristics

Symbol	Description	Min.	Typ.	Max.	Unit
t_{start}	Start-up time	-	25	-	µs
$V_{DDINT055V}$	Power supply voltage range for INT055V	1.8	-	5.5	V
$V_{DDINT11V}$	Power supply voltage range for INT11V	1.8	-	5.5	
$V_{DDINT15V}$	Power supply voltage range for INT15V	1.9	-	5.5	
$V_{DDINT25V}$	Power supply voltage range for INT25V	2.9	-	5.5	
$V_{DDINT43V}$	Power supply voltage range for INT43V	4.75	-	5.5	

```
_delay_us(VREF_STARTUP_MICROS);
```

The DAC output can be used internally by other peripherals, or it can be linked to an output pin. For the ATtiny817, the DAC output is connected to pin A6 (see Figure 3-3).

Figure 3-3. PORT Function Multiplexing



VQFN 24-pin	Pin Name	Other/Special	ADC0	PTC	AC0	DAC0	USART0
23	PA0	$\overline{\text{RESET}}$ UPDI	AIN0				
24	PA1		AIN1				TXD
1	PA2	EVOUT0	AIN2				RxD
2	PA3	EXTCLK	AIN3				XCK
3	GND						
4	VDD						
5	PA4		AIN4	X0/Y0			XDIR
6	PA5		AIN5	X1/Y1	OUT		
7	PA6		AIN6	X2/Y2	AINN0	OUT	
8	PA7		AIN7	X3/Y3	AINP0		
9	PB7						

The DAC output pin needs to have the digital input buffer and the pull-up resistor disabled to reduce its load.

```
PORTA.PIN6CTRL &= ~PORT_ISC_gm;
PORTA.PIN6CTRL |= PORT_ISC_INPUT_DISABLE_gc;
PORTA.PIN6CTRL &= ~PORT_PULLUPEN_bm;
```

The desired output for the DAC in this particular example is 0.5425V. To achieve it, the following equation is applied:

$$DATA = V_{OUT} \times 256/V_{REF} = 0.5425V \times 256/4.34V = 32 = 0x20$$

Writing to the DAC0.DATA register at initialization is optional; however, it may be useful to make the DAC output a specific voltage from the beginning.

```
DAC0.DATA = 0x20;
```

Enabling DAC, Output Buffer and Run in Standby sleep mode:

```
DAC0.CTRLA = DAC_ENABLE_bm | DAC_OUTEN_bm | DAC_RUNSTDBY_bm;
```

Important: If Run in Standby sleep mode is enabled, the DAC will continue to run when the MCU is in Standby sleep mode.

Starting a Conversion

When the DAC is enabled (ENABLE = `1` in DAC.CTRLA), a conversion starts as soon as the Data (DAC.DATA) register is written.

```
DAC0.DATA = 0x20;
```

After conversion, the output keeps its value of $\frac{\mathrm{DATA} \times \mathrm{V}_{\mathrm{REF}}}{256}$ until the next conversion, as long as the DAC is running. Any change in V_{REF} selection will immediately change the DAC output (if enabled and running).

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

View the ATtiny817 Code Example on GitHub
Click to browse repository

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

View the AVR128DA48 Code Example on GitHub
Click to browse repository

4. Generating Sine Wave Signal

The DAC can be used to obtain a sine wave signal by generating a defined number of discrete samples of the desired analog signal. Each sample is characterized by a voltage level and by a time duration. This mechanism provides an approximation of the desired sine wave signal.

In this application, the number of steps used is 100. To obtain a signal frequency of 50 Hz (translated into a period of 20 ms), the time duration between two signal samples will be (1000000/Desired Frequency)/Number of Steps [µs] = (1000000/50 Hz)/100 (steps) [µs] = 200 µs.

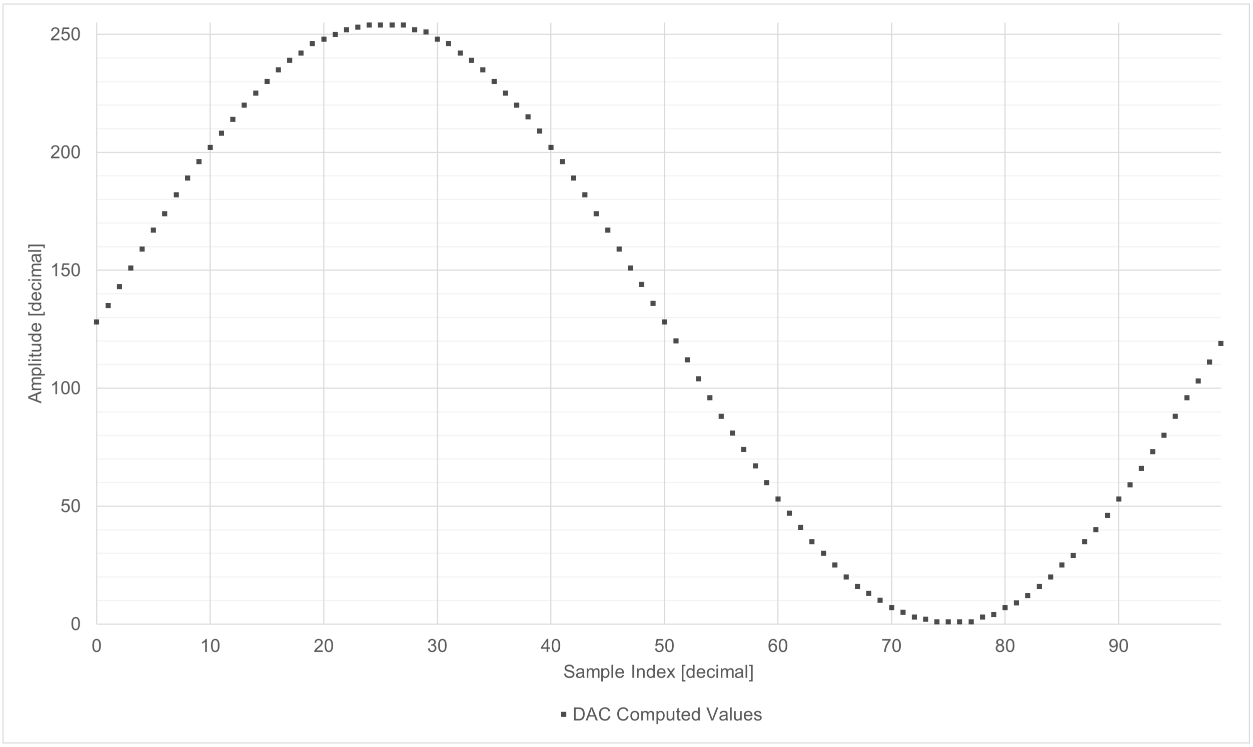
To generate the “steps” that approximate the sine wave, the integer values to be written to the DAC0.DATA register will be computed using the formula below:

$$sineWave[i] = DC\,Offset + A \cdot sin\left(\frac{\mathrm{i} \cdot 2\pi}{Number\ of\ Steps}\right)$$

,where *i* is the sample index and takes values from 0 to Number of Steps, *A* is the signal amplitude (the maximum possible value is the DAC reference voltage), and the *Number of Steps* represents the number of samples used to construct the sine wave signal.

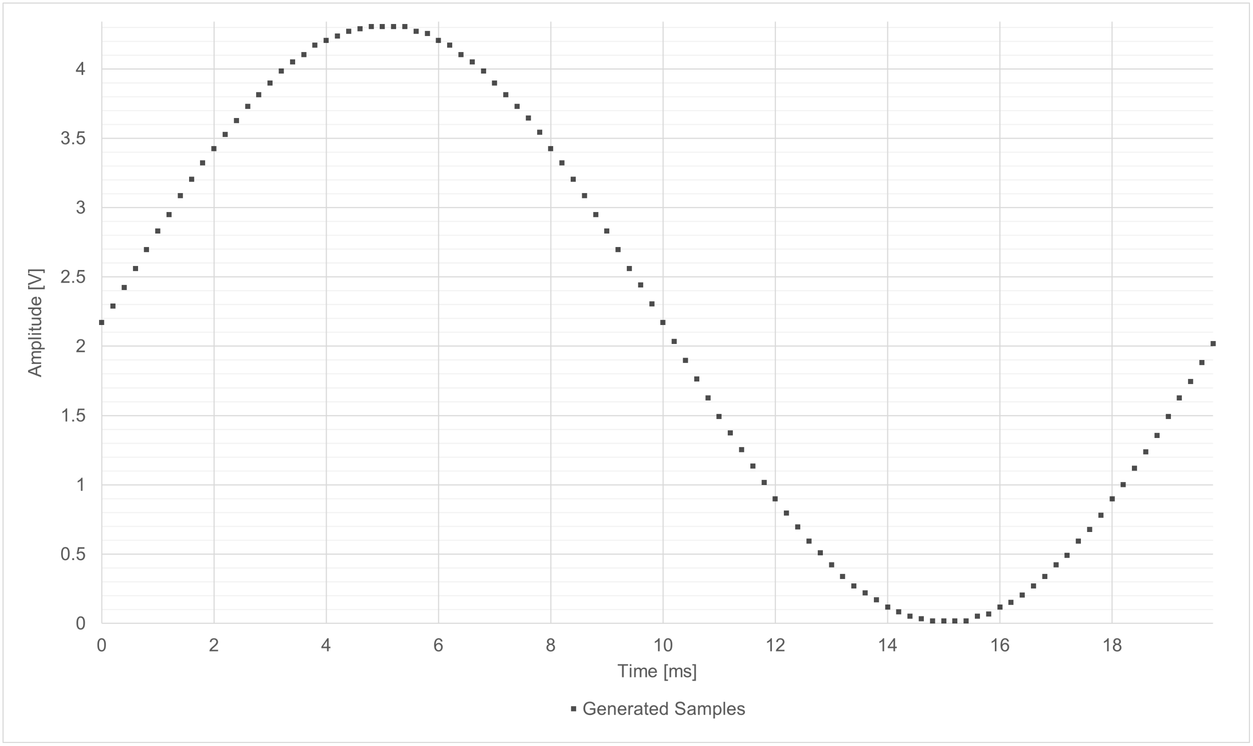
To generate a sine wave signal between 0V and 4.34V, the DAC must convert integers ranged between 0 and 255. Therefore, because the DAC will only generate positive voltage values, the DC Offset will be 128 (this is also the mean value of the sine wave signal). To obtain the maximum of 4.34V, the amplitude must be 127. The data to be written to the DAC0.DATA register is presented on the graph below.

Figure 4-1. DAC Input Digital Values



The ideal voltage values that will be available on the DAC output are presented on the graph below.

Figure 4-2. DAC Output Analog Values



To avoid spending time inside the infinite loop by computing the signal samples in real-time, the samples are computed at the beginning of the application and stored in a Look-up Table. The function implementation for this computation is presented below.

```
for(uint16_t i = 0; i < SINE_WAVE_STEPS; i++)
{
    sineWave[i] = SINE_DC_OFFSET + SINE_AMPLITUDE * sin(i * M_2PI / SINE_WAVE_STEPS);
}
```

The data is then written to the DAC0.DATA register inside the infinite loop using a 200 μs delay, as described above.

```
while (1)
{
    DAC0.DATA = sineWave[i++];
    i = i % SINE_WAVE_STEPS;
    _delay_us(STEP_DELAY_MICROS);
}
```

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

View the ATtiny817 Code Example on GitHub
Click to browse repository

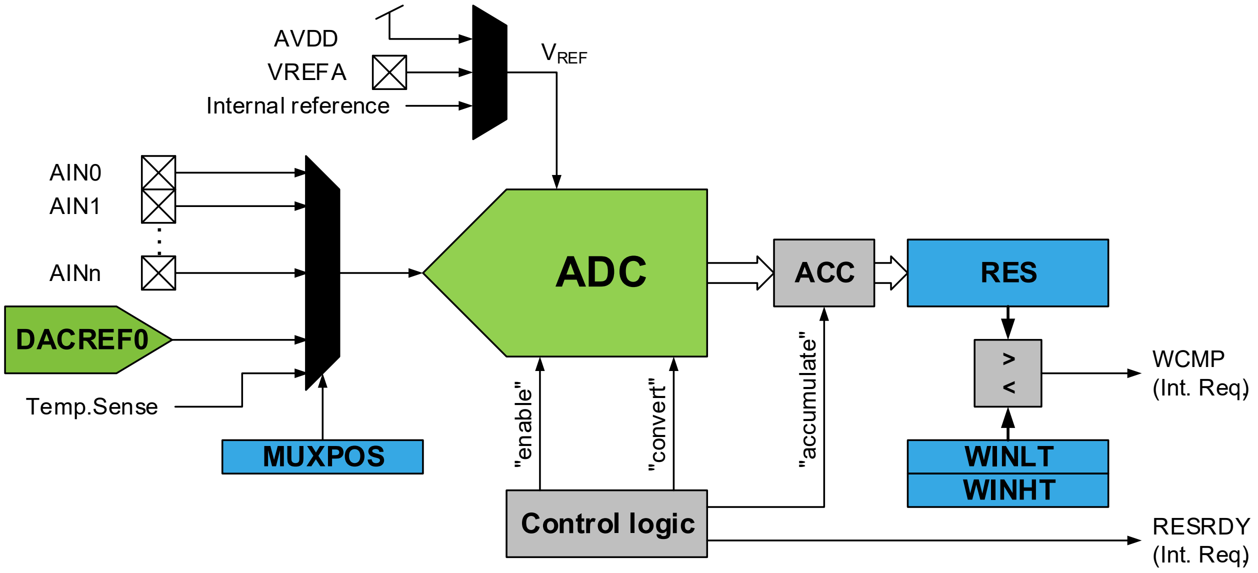
An MCC generated code example for AVR128DA48, with the same functionality as the one described in this section, can be found here:

View the AVR128DA48 Code Example on GitHub
Click to browse repository

5. Reading the DAC Internally with the ADC

As seen in Figure 5-1, the ADC input can be connected to the DAC output. Therefore, the DAC conversion result (a digital value) can be transmitted directly to the ADC. Then, the ADC will convert the analog result provided by the DAC again, obtaining a digital result.

Figure 5-1. Analog-to-Digital Converter Block Diagram



The VREF is configured to provide the voltage reference for the ADC0 and DAC0, as presented below.

Figure 5-2. VREF.CTRLA Register

Bit	7	6	5	4	3	2	1	0
		ADC0REFSEL[2:0]				DAC0REFSEL[2:0]		
Access		R/W	R/W	R/W		R/W	R/W	R/W
Reset		0	0	0		0	0	0

bits 6:4 ADC0REFSEL[2:0]: ADC0 Reference Select bits
These bits select the reference voltage for the ADC0.

Value	Description
0x0	0.55V
0x1	1.1V
0x2	2.5V
0x3	4.3V
0x4	1.5V
other	Reserved

bits 2:0 DAC0REFSEL[2:0]: DAC0 and AC0 Reference Select bits
These bits select the reference voltage for the DAC0 and AC0.

Value	Description
0x0	0.55V
0x1	1.1V
0x2	2.5V
0x3	4.3V
0x4	1.5V
other	Reserved

The VREF peripheral module is initialized as illustrated in the code snippet below.

```
VREF_CTRLA |= VREF_DAC0REFSEL_4V34_gc;
VREF_CTRLB |= VREF_DAC0REFEN_bm;
VREF_CTRLA |= VREF_ADC0REFSEL_4V34_gc;
VREF_CTRLB |= VREF_ADC0REFEN_bm;
_delay_us(VREF_STARTUP_MICROS);
```

Then, the ADC is initialized as presented below.

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

ADC0.CTRLA = ADC_ENABLE_bm
           | ADC_RESSEL_10BIT_gc;
```

To read the DAC with the ADC, the MUXPOS register of the ADC must be set to `0x1C`.

Figure 5-3. MUXPOS DAC Output Selection

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

bits 4:0 MUXPOS[4:0]: MUXPOS bits
This bit field selects which single-ended analog input is connected to the ADC. If these bits are changed during a conversion, the change will not take effect until this conversion is complete.

Value	Name	Description
`0x0B`	AIN11	ADC input pin 11
`0x1C`	DAC0	DAC0
`0x1D`	INTREF	Internal reference (from VREF peripheral)
`0x1E`	TEMPSENSE	Temperature sensor
`0x1F`	GND	0V (GND)
`Other`	-	Reserved

```
ADC0.MUXPOS = ADC_MUXPOS_DAC0_gc;
```

The ADC conversion is started by writing the corresponding bit to the ADC0.COMMAND.

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

When the conversion is done, the Result Ready (RESRDY) flag in ADC0.INTFLAGS will be set by the hardware.

```
while (!(ADC0.INTFLAGS & ADC_RESRDY_bm))
{
    ;
}
```

The flag must be cleared by software, as presented below.

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

The ADC conversion result is available in the ADC0.RES register.

The DAC output can be set to different values and read using the ADC inside the infinite loop.

```
while (1)
{
    adcVal = ADC0_read();
    dacVal++;
    DAC0_setVal(dacVal);
}
```

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

View the ATtiny817 Code Example on GitHub
Click to browse repository

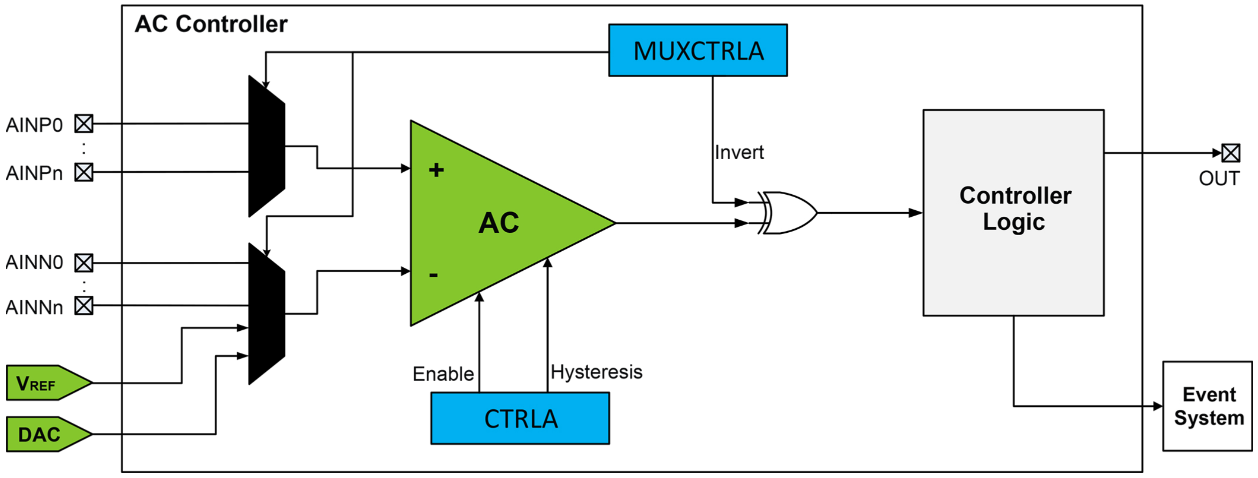
An MCC generated code example for AVR128DA48, with the same functionality as the one described in this section, can be found here:

View the AVR128DA48 Code Example on GitHub
Click to browse repository

6. Using DAC as Negative Input for AC

As seen in Figure 6-1, the AC can use the DAC output as the negative input. Therefore, the user can generate the AC negative voltage reference by simply writing the corresponding digital value to the DATA register of the DAC peripheral module. Then, the DAC will generate the conversion result, a constant voltage, which will be directly connected to the AC input.

Figure 6-1. Analog Comparator Block Diagram



In this code example, the voltage reference for the DAC module is configured at 1.5V.

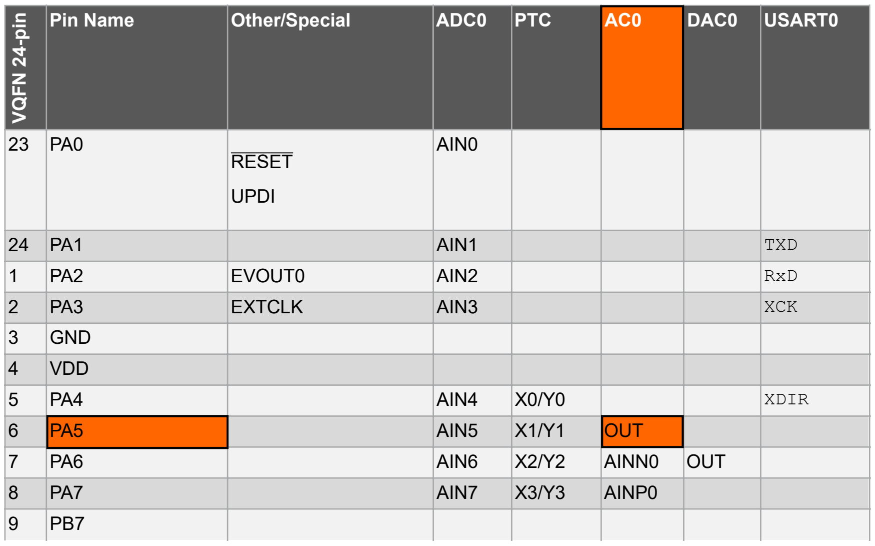
```
VREF.CTRLA |= VREF_DAC0REFSEL_1V5_gc;
VREF.CTRLB |= VREF_DAC0REFEN_bm;
_delay_us(VREF_STARTUP_MICROS);
```

The AC is configured to receive its negative input from DAC and positive input from pin PA7.

VQFN 24-pin	Pin Name	Other/Special	ADC0	PTC	AC0	DAC0	USART0
23	PA0	RESET UPDI	AIN0				
24	PA1		AIN1				TXD
1	PA2	EVOUT0	AIN2				RxD
2	PA3	EXTCLK	AIN3				XCK
3	GND						
4	VDD						
5	PA4		AIN4	X0/Y0			XDIR
6	PA5		AIN5	X1/Y1	OUT		
7	PA6		AIN6	X2/Y2	AINN0	OUT	
8	PA7		AIN7	X3/Y3	AINP0		
9	PB7						

```
AC0.MUXCTRLA = AC_MUXNEG_DAC_gc
             | AC_MUXPOS_PIN0_gc;
```

Pin PA5 is configured as the AC output.



VQFN 24-pin	Pin Name	Other/Special	ADC0	PTC	AC0	DAC0	USART0
23	PA0	$\overline{\text{RESET}}$ UPDI	AIN0				
24	PA1		AIN1				TXD
1	PA2	EVOUT0	AIN2				RxD
2	PA3	EXTCLK	AIN3				XCK
3	GND						
4	VDD						
5	PA4		AIN4	X0/Y0			XDIR
6	PA5		AIN5	X1/Y1	OUT		
7	PA6		AIN6	X2/Y2	AINN0	OUT	
8	PA7		AIN7	X3/Y3	AINP0		
9	PB7						

```
AC0.CTRLA = AC_ENABLE_bm
            | AC_OUTEN_bm
            | AC_RUNSTDBY_bm
            | AC_LPMODE_bm;
```

After initializing the VREF, DAC, and AC, the DAC output is set to 1.4V.

```
DAC0_setVal(DAC_DATA_1V4);
```

The AC will compare the voltage from PA7 to 1.4V and set its output according to the comparison result.

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

View the ATtiny817 Code Example on GitHub
Click to browse repository

This application can be implemented on AVR128DA48 by using the internal reference voltage generator (DACREF) feature of the AC. There is no need to configure the DAC to provide a negative input for the AC on this architecture.

7. References

1. ATtiny417/817 - AVR® Microcontroller with Core Independent Peripherals and picoPower® Technology (DS40001901)
2. ATtiny817 Xplained Mini board
3. ATtiny817 product page: www.microchip.com/wwwproducts/en/ATTINY817
4. AVR128DA48 product page: www.microchip.com/wwwproducts/en/AVR128DA48
5. AVR128DA48 Curiosity Nano Evaluation Kit product page: https://www.microchip.com/Developmenttools/ProductDetails/DM164151
6. AVR128DA28/32/48/64 Data Sheet
7. Getting Started with the AVR® DA Family

8. Appendix

Example 8-1. Generating Constant Analog Signal Code Example

```
/* 3.33 MHz (needed for delay) */
#define F_CPU                           (3333333UL)
/* DAC value */
#define DAC_EXAMPLE_VALUE               (0x20)
/* VREF Startup time */
#define VREF_STARTUP_MICROS             (25)

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

void VREF_init(void);
void DAC0_init(void);
void DAC0_setVal(uint8_t val);

void VREF_init(void)
{
    /* Voltage reference at 4.34V */
    VREF.CTRLA |= VREF_DAC0REFSEL_4V34_gc;
    /* DAC0/AC0 reference enable: enabled */
    VREF.CTRLB |= VREF_DAC0REFEN_bm;
    /* Wait VREF start-up time */
    _delay_us(VREF_STARTUP_MICROS);
}

void DAC0_init(void)
{
    /* Disable digital input buffer */
    PORTA.PIN6CTRL &= ~PORT_ISC_gm;
    PORTA.PIN6CTRL |= PORT_ISC_INPUT_DISABLE_gc;
    /* Disable pull-up resistor */
    PORTA.PIN6CTRL &= ~PORT_PULLUPEN_bm;

    /* Enable DAC, Output Buffer, Run in Standby */
    DAC0.CTRLA = DAC_ENABLE_bm | DAC_OUTEN_bm | DAC_RUNSTDBY_bm;
}

void DAC0_setVal(uint8_t val)
{
    DAC0.DATA = val;
}

int main(void)
{
    VREF_init();
    DAC0_init();

    DAC0_setVal(DAC_EXAMPLE_VALUE);

    while (1)
    {
        ;
    }
}
```

Example 8-2. Generating Sine Wave Signal Code Example

```
    /* 3.33 MHz (needed for delay) */
    #define F_CPU                               (3333333UL)
    /* VREF Startup time */
    #define VREF_STARTUP_MICROS                 (25)
    /* Number of steps for a sine wave period */
    #define SINE_WAVE_STEPS                     (100)
    /* Sine wave amplitude */
    #define SINE_AMPLITUDE                      (127)
    /* Sine wave DC offset */
    #define SINE_DC_OFFSET                      (128)
    /* 2*PI */
    #define M_2PI                               (2 * M_PI)
    /* Frequency of the sine wave */
    #define OUTPUT_FREQ                         (50)
    /* Step delay for the synthesis loop */
    #define STEP_DELAY_MICROS                   ((1000000 / OUTPUT_FREQ) / 
SINE_WAVE_STEPS)

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

    /* Buffer to store the sine wave samples */
    uint8_t sineWave[SINE_WAVE_STEPS];

    void sineWaveInit(void);
    void VREF_init(void);
    void DAC0_init(void);
    void DAC0_setVal(uint8_t val);

    void sineWaveInit(void)
    {
        for(uint16_t i = 0; i < SINE_WAVE_STEPS; i++)
        {
            sineWave[i] = SINE_DC_OFFSET + SINE_AMPLITUDE * sin(i * M_2PI / 
SINE_WAVE_STEPS);
        }
    }

    void VREF_init(void)
    {
        /* Voltage reference at 4.34V */
        VREF.CTRLA |= VREF_DAC0REFSEL_4V34_gc;
        /* DAC0/AC0 reference enable: enabled */
        VREF.CTRLB |= VREF_DAC0REFEN_bm;
        /* Wait VREF start-up time */
        _delay_us(VREF_STARTUP_MICROS);
    }

    void DAC0_init(void)
    {
        /* Disable digital input buffer */
        PORTA.PIN6CTRL &= ~PORT_ISC_gm;
        PORTA.PIN6CTRL |= PORT_ISC_INPUT_DISABLE_gc;
        /* Disable pull-up resistor */
        PORTA.PIN6CTRL &= ~PORT_PULLUPEN_bm;

        /* default value */
        DAC0.DATA = SINE_DC_OFFSET;
        /* Enable DAC, Output Buffer, Run in Standby */
        DAC0.CTRLA = DAC_ENABLE_bm | DAC_OUTEN_bm | DAC_RUNSTDBY_bm;
    }

    void DAC0_setVal(uint8_t val)
    {
        DAC0.DATA = val;
    }

    int main(void)
    {
        uint16_t i = 0;

        VREF_init();
        DAC0_init();

        sineWaveInit();

        while (1)
        {
            DAC0_setVal(sineWave[i++]);
            i = i % SINE_WAVE_STEPS;
            _delay_us(STEP_DELAY_MICROS);
        }
    }
```

Example 8-3. Reading DAC Internally With ADC Code Example

```
    /* 3.33 MHz (needed for delay) */
    #define F_CPU                         (3333333UL)
    /* VREF Startup time */
    #define VREF_STARTUP_MICROS           (25)

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

    void VREF_init(void);
    void DAC0_init(void);
    void ADC0_init(void);
    uint16_t ADC0_read(void);
    void DAC0_setVal(uint8_t val);

    void VREF_init(void)
    {
        /* Voltage reference at 4.34V */
        VREF_CTRLA |= VREF_DAC0REFSEL_4V34_gc;
        /* DAC0/AC0 reference enable: enabled */
        VREF_CTRLB |= VREF_DAC0REFEN_bm;
        /* Voltage reference at 4.34V */
        VREF_CTRLA |= VREF_ADC0REFSEL_4V34_gc;
        /* ADC0 reference enable: enabled */
        VREF_CTRLB |= VREF_ADC0REFEN_bm;
        /* Wait VREF start-up time */
        _delay_us(VREF_STARTUP_MICROS);
    }

    void DAC0_init(void)
    {
        /* Enable DAC */
        DAC0.CTRLA = DAC_ENABLE_bm;
    }

    void ADC0_init(void)
    {
        ADC0.CTRLC = ADC_PRESC_DIV4_gc      /* CLK_PER divided by 4 */
                   | ADC_REFSEL_INTREF_gc   /* VDD reference */
                   | ADC_SAMPCAP_bm;        /* Sample Capacitance Selection: 
enabled */

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

        /* Select ADC channel */
        ADC0.MUXPOS = ADC_MUXPOS_DAC0_gc;
    }

    uint16_t ADC0_read(void)
    {
        /* Start ADC conversion */
        ADC0.COMMAND = ADC_STCONV_bm;

        /* Wait until ADC conversion done */
        while ( !(ADC0.INTFLAGS & ADC_RESRDY_bm) )
        {
            ;
        }
        /* Clear the interrupt flag by writing 1: */
        ADC0.INTFLAGS = ADC_RESRDY_bm;

        return ADC0.RES;
    }

    void DAC0_setVal(uint8_t val)
    {
        DAC0.DATA = val;
    }

    int main(void)
    {
        uint8_t dacVal = 0;
        volatile uint16_t adcVal = 0;

        VREF_init();
        DAC0_init();
        ADC0_init();

        /* Wait VREF start-up time */
        _delay_us(VREF_STARTUP_MICROS);

        DAC0_setVal(dacVal);

        while (1)
        {
            adcVal = ADC0_read();

            /* do something with the adcVal */

            dacVal++;
            DAC0_setVal(dacVal);
        }
    }
```

Example 8-4. Using DAC as Negative Input for AC Code Example

```
    /* 3.33 MHz (needed for delay) */
    #define F_CPU                          (3333333UL)
    /* 1.4V output @ VREF = 1.5V */
    #define DAC_DATA_1V4                   (239)
    /* VREF Startup time */
    #define VREF_STARTUP_MICROS            (25)

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

    void VREF_init(void);
    void DAC0_setVal(uint8_t val);
    void DAC0_init(void);
    void AC0_init(void);

    void VREF_init(void)
    {
        /* Voltage reference at 1.5V */
        VREF.CTRLA |= VREF_DAC0REFSEL_1V5_gc;
        /* DAC0/AC0 reference enable: enabled */
        VREF.CTRLB |= VREF_DAC0REFEN_bm;
        /* Wait VREF start-up time */
        _delay_us(VREF_STARTUP_MICROS);
    }

    void DAC0_setVal(uint8_t val)
    {
        DAC0.DATA = val;
    }

    void DAC0_init(void)
    {
        /* Disable digital input buffer */
        PORTA.PIN6CTRL &= ~PORT_ISC_gm;
        PORTA.PIN6CTRL |= PORT_ISC_INPUT_DISABLE_gc;
        /* Disable pull-up resistor */
        PORTA.PIN6CTRL &= ~PORT_PULLUPEN_bm;

        /* Enable DAC, Output Buffer, Run in Standby */
        DAC0.CTRLA = DAC_ENABLE_bm | DAC_OUTEN_bm | DAC_RUNSTDBY_bm;
    }

    void AC0_init(void)
    {
        /* Negative input from DAC0 */
        /* Positive input from pin PA7 */
        AC0.MUXCTRLA = AC_MUXNEG_DAC_gc
                     | AC_MUXPOS_PIN0_gc;

        /* Enable, Output on PA5, Low Power mode */
        AC0.CTRLA = AC_ENABLE_bm
                     | AC_OUTEN_bm
                     | AC_RUNSTDBY_bm
                     | AC_LPMODE_bm;
    }

    int main(void)
    {
        /* Voltage divider -> VDD/2 input on PA7 */
        /* AC output on LED on PA5 */
        /* LED turns OFF when battery is below 2.8V (PA7 below 1.4V) */

        VREF_init();
        DAC0_init();
        AC0_init();

        /* 1.4V output @ VREF = 1.5V */
        DAC0_setVal(DAC_DATA_1V4);

        while (1)
        {
            ;
        }
    }
```

9. Revision History

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

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

ISBN: 978-1-5224-7853-9

Quality Management System

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

Worldwide Sales and Service

AMERICAS

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

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

Austin, TX
Tel: 512-257-3370

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

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

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

Detroit
Novi, MI
Tel: 248-848-4000

Houston, TX
Tel: 281-894-5983

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

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

Raleigh, NC
Tel: 919-844-7510

New York, NY
Tel: 631-435-6000

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

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

ASIA/PACIFIC

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

China - Xiamen
Tel: 86-592-2388138

China - Zhuhai
Tel: 86-756-3210040

ASIA/PACIFIC

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

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

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

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

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

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

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

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

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

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

Singapore
Tel: 65-6334-8870

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

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

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

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

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

EUROPE

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

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

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

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

Germany - Garching
Tel: 49-8931-9700

Germany - Haan
Tel: 49-2129-3766400

Germany - Heilbronn
Tel: 49-7131-72400

Germany - Karlsruhe
Tel: 49-721-625370

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

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

Israel - Ra’anana
Tel: 972-9-744-7705

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

Italy - Padova
Tel: 39-049-7625286

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

Norway - Trondheim
Tel: 47-72884388

Poland - Warsaw
Tel: 48-22-3325737

Romania - Bucharest
Tel: 40-21-407-87-50

Spain - Madrid
Tel: 34-91-708-08-90
Fax: 34-91-708-08-91

Sweden - Gothenberg
Tel: 46-31-704-60-40

Sweden - Stockholm
Tel: 46-8-5090-4654

UK - Wokingham
Tel: 44-118-921-5800
Fax: 44-118-921-5820