TB3235

Using 10-Bit DAC for Generating Analog Signals

Introduction

Authors: Cristian Sabiuta, Marius Nicolae, Microchip Technology Inc.

The AVR® DA MCU family of microcontrollers is based on the AVR architecture and brings a DAC peripheral equipped with 10-bit resolution and high-drive capabilities, helping the user to generate precise analog voltages and use them internally or externally on a physical pin.

This technical brief describes how the 10-bit DAC works on the AVR DA microcontroller family, covering the following use cases:

- **Generating Constant Analog Signal Using 10-Bit DAC:**
  Initialize the DAC, set the voltage reference, set the DAC to output a specific constant voltage.
- **Generating Sine Wave Signal Using 10-Bit DAC:**
  Initialize the DAC, set the voltage reference, output in a loop the samples of a sine wave.
- **Reading the DAC Internally with the ADC:**
  Initialize the DAC and ADC, set the voltage reference, set the ADC to read the DAC, increment the DAC output and read it with the ADC for each step.
- **Generating Amplitude Modulated Signal Using 10-Bit DAC:**
  Initialize the DAC with external reference and link the signal that must be modulated to the external reference pin. The AVR core will continuously change the Data (DACn.DATA) register to create a modulated signal.

Note: The code examples were developed on AVR128DA48 Curiosity Nano.

Table of Contents

1. Relevant Devices

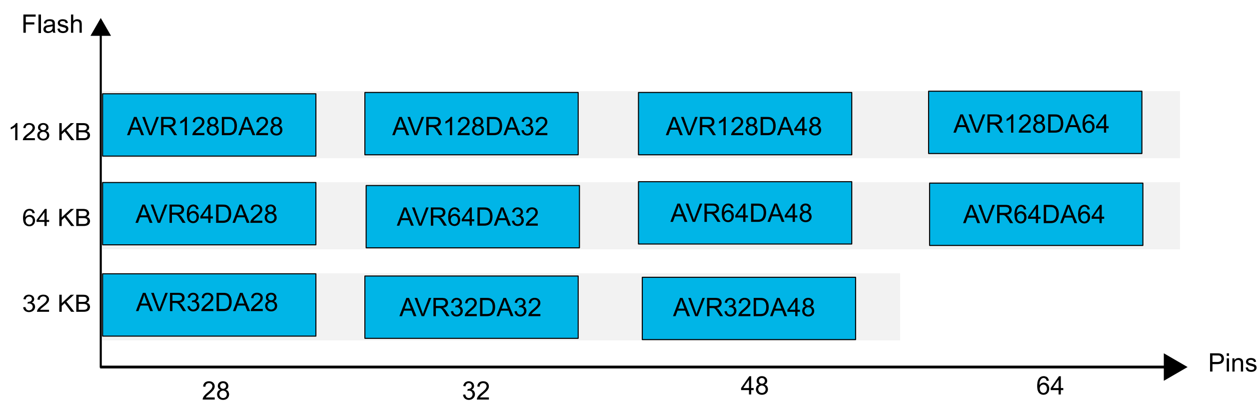
This chapter lists the relevant devices for this document.

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 DAC features a 10-bit resolution and 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 pair.

Figure 2-1. DAC Block Diagram

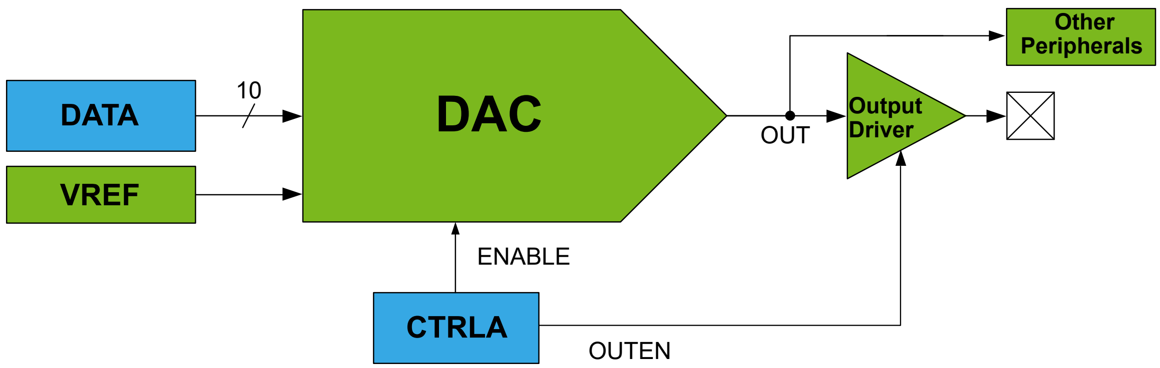


Figure 2-2. Signal Description

Signal	Description	Type
OUT	DAC output	Analog

3. Generating Constant Analog Signal Using 10-Bit DAC

The DAC can be used to generate a constant analog signal. It uses the output of the Voltage Reference (VREF) peripheral as positive reference.

The DAC output ranges from 0V to $\frac{1023 \times V_{REF}}{1024}$

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

- Internal 1.024V reference
- Internal 2.048V reference
- Internal 4.096V reference
- Internal 2.500V reference
- V_{DD} reference
- External reference from the VREFA pin (PD7)

Figure 3-1. VREF.DAC0REF Register

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 DAC0 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 DAC0.
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
0x1	2V048	Internal 2.048V reference
0x2	4V096	Internal 4.096V reference
0x3	2V500	Internal 2.500V reference
0x4	-	Reserved
0x5	VDD	VDD as reference
0x6	VREFA	External reference from the VREFA pin
0x7	-	Reserved

For the purpose of this example, the 2.048V reference voltage was selected:

```
VREF.DAC0REF = VREF_REFSEL_2V048_gc;
```

A 50 μs delay is recommended after enabling the VREF peripheral.

Figure 3-2. Internal Voltage Reference (VREF) Characteristics

Symbol	Description	Min.	Typ.	Max.	Units	Conditions
V_{VREF_1V024}	Internal Voltage Reference 1.024V	-4	—	+4	%	V_{DD}≥2.5V, -40°C to 85°C
V_{VREF_2V048}	Internal Voltage Reference 2.048V	-4	—	+4	%	V_{DD}≥2.5V, -40°C to 85°C
V_{VREF_4V096}	Internal Voltage Reference 4.096V	-4	—	+4	%	V_{DD}≥4.55V, -40°C to 85°C
V_{VREF_2V500}	Internal Voltage Reference 2.5V	-4	—	+4	%	V_{DD}≥4.55V, -40°C to 85°C
T_{VREF_ST}	VREF Start-up Time	—	50	—	µs	

```
_delay_us(50);
```

The DAC output can be used internally by other peripherals, or it can be linked to an output pin. For the AVR128DA48, the DAC output is connected to pin PD6 (see the figure below).

Figure 3-3. PORT Function Multiplexing

VQFN64/ TQFP64	VQFN48/ TQFP48	VQFN32/ TQFP32	SOIC28/ SSOP28	Pin name (1,2)	Special	ADC0	PTC	ACn	DAC0	ZCDn	USARTn	SPIn	TWIn(4)	TCA0	TCA1	TCBn	TCDn	EVSYS	CCL-LUTn
30	24	14	10	PD4		AIN4	X20/Y20	1,AINP2 2,AINP1						WO4					
31	25	15	11	PD5		AIN5	X21/Y21	1,AINN0						WO5					
32	26	16	12	PD6		AIN6	X22/Y22	0,AINP3 1,AINP3 2,AINP3	VOUT										2,OUT
33	27	17	13	PD7	VREFA	AIN7	X23/Y23	0,AINN2 1,AINN2 2,AINN0/AINN2										EVOUTD	

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

```
PORTD.PIN6CTRL &= ~PORT_ISC_gm;

PORTD.PIN6CTRL |= PORT_ISC_INPUT_DISABLE_gc;

PORTD.PIN6CTRL &= ~PORT_PULLUPEN_bm;
```

The DACn.DATA register is used to generate a specific analog output voltage. The value of this output voltage can be determined using the following equation:

$$V_{OUT} = \frac{(DACn.DATA \times V_{REF})}{1024}$$

Writing to the DACn.DATA register at initialization is optional; however, it is useful to make the DAC output a specific voltage from the beginning. The DAC features a 10-bit resolution, therefore, the DACn.DATAL and DACn.DATAH register pair represents the 10-bit value DACn.DATA (see Figure 3-4). The two LSbs [1:0] are accessible at the original offset and the eight MSbs [9:2] can be accessed at offset +1.

The output will be updated after DACn.DATAH is written.

Figure 3-4. DACn.DATA Register

Bit	15	14	13	12	11	10	9	8
	DATA[9:2]							
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
	DATA[1:0]							
Access	R/W	R/W						
Reset	0	0						

Bits 15:6 – DATA[9:0]
These bits contain the digital data, which will be converted to an analog voltage.

The desired output for the DAC in this example is 1.2V. To achieve this, the following equation is applied:

$$DACn.DATA = \frac{(V_{OUT} \times 1024)}{V_{REF}} = \frac{(1.2V \times 1024)}{2.048V} = 600 = 0x258$$

In order to enable the DAC, Output Buffer, and Run in Standby mode, use the following code:

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

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

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

When the DAC is disabled (ENABLE = `0` in DACn.CTRLA), writing to the Data registers does not trigger a conversion. Instead, the conversion starts on writing a '`1`' to the ENABLE bit in the DACn.CTRLA register.

```
DAC0.DATAL = (value & (0x03)) << 6;

DAC0.DATAH = value >> 2;
```

After a conversion, the output keeps its value of $\frac{DACn.DATA \times V_{REF}}{1024}$ until the next conversion, as long as the DAC is running. Any change in the V_{REF} selection will immediately change the DAC output (if enabled and running).

View Code Example on GitHub
Click to browse repository

Tip: The full code example is also available in 8. Appendix.

4. Generating Sine Wave Signal Using 10-Bit DAC

The DAC can be used to generate a sine wave signal.

To generate this signal, the VREF and the DAC are initialized first, then the output value can be changed by writing a new value to the DACn.DATA register.

Before the sine wave is generated, the samples corresponding to a period are calculated and stored in a buffer.

```
for(i = 0; i < SINE_PERIOD_STEPS; i++)
{
    sineWave[i] = SINE_DC_OFFSET + SINE_AMPLITUDE * sin(2 * M_PI * i / SINE_PERIOD_STEPS);
}
```

The sinusoidal waveform is created using a fixed number of steps (N_SAMPLES). To create a sine wave signal with a specific frequency (SIGNAL_FREQ), all steps are executed in one period resulting in the following sample rate:

$$SAMPLE\_RATE = \frac{1}{STEP\_DELAY\_TIME} = SIGNAL\_FREQ \times N\_SAMPLES$$

```
while (1)
{
    DAC0_setVal(sineWave[sineIndex++]);
    if(sineIndex == SINE_PERIOD_STEPS)
        sineIndex = 0;
    _delay_us(STEP_DELAY_TIME);
}
```

View Code Example on GitHub
Click to browse repository

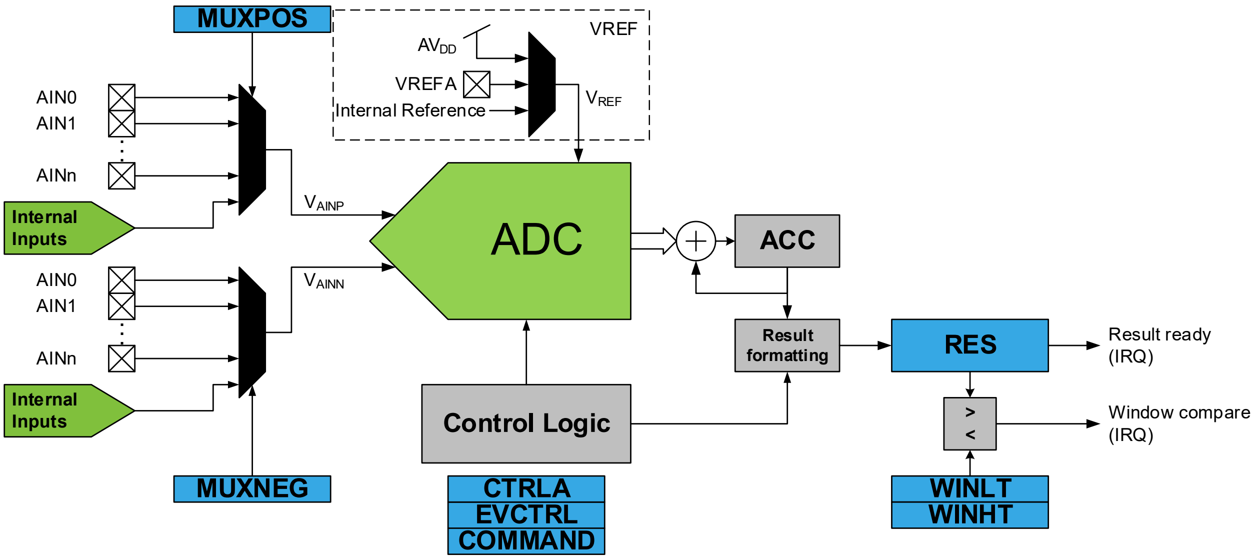
Tip: The full code example is also available in 8. Appendix.

5. Reading the DAC Internally with the ADC

The analog output of the DAC can be internally connected to other peripherals when the DAC is enabled (ENABLE = `1` in DACn.CTRLA). When the DAC analog output is only being used internally, it is not necessary to enable the pin output driver (OUTEN = `0` in DACn.CTRLA is acceptable).

Referring to the ADC block diagram below, the output of the 10-bit DAC can be used as the internal input to the ADC.

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



The DAC voltage reference is initialized as mentioned in 3. Generating Constant Analog Signal Using 10-Bit DAC and the ADC voltage reference is initialized in the same way from the VREF peripheral.

Figure 5-2. VREF.DAC0REF Register

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 DAC0 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 DAC0.
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
`0x1`	2V048	Internal 2.048V reference
`0x2`	4V096	Internal 4.096V reference
`0x3`	2V500	Internal 2.500V reference
`0x4`	-	Reserved
`0x5`	VDD	VDD as reference
`0x6`	VREFA	External reference from the VREFA pin
`0x7`	-	Reserved

Figure 5-3. VREF.ADC0REF Register

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:

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
0x1	2V048	Internal 2.048V reference
0x2	4V096	Internal 4.096V reference
0x3	2V500	Internal 2.500V reference
0x4	-	Reserved
0x5	VDD	VDD as reference
0x6	VREFA	External reference from the VREFA pin
0x7	-	Reserved

The complete VREF initialization is shown below:

```
VREF.DAC0REF = VREF_REFSEL_2V048_gc /* Select the 2.048V Internal Voltage Reference for DAC */
             | VREF_ALWAYSON_bm;  /* Set the Voltage Reference in Always On mode */
VREF.ADC0REF = VREF_REFSEL_2V048_gc /* Select the 2.048V Internal Voltage Reference for ADC */
             | VREF_ALWAYSON_bm;  /* Set the Voltage Reference in Always On mode */
/* Wait VREF start-up time */
_delay_us(VREF_STARTUP_TIME);
```

Then, the ADC must be initialized:

```
ADC0.CTRLC = ADC_PRESC_DIV2_gc;
ADC0.CTRLA = ADC_ENABLE_bm | ADC_RESSEL_12BIT_gc;
```

To read the DAC with the ADC, the MUXPOS register of the ADC must be set to `0x48`, corresponding to DAC0.

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

Figure 5-4. MUXPOS DAC Output 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 these bits are 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
`0x16-0x1F`	-	Reserved
`0x20-0x3F`	-	Reserved
`0x40`	GND	Ground
`0x41`	-	Reserved
`0x42`	TEMPSENSE	Temperature sensor
`0x48`	DAC0	DAC0
`Other`	-	Reserved

The ADC conversion is started by writing the corresponding bit to the ADCn.COMMAND register:

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

When the conversion is done, the RESRDY bit in the ADCn.INTFLAGS register will be set by hardware.

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

The flag is cleared by either writing a '`1`' to the RESRDY bit location or by reading the Result (ADCn.RES) register. Writing a '`0`' to this bit has no effect.

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

The ADC data can be read from the Result (ADCn.RES) register.

The DAC output can be set to different values, and read with the ADC in a loop:

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

View Code Example on GitHub
Click to browse repository

Tip: The full code example is also available in 8. Appendix.

6. Generating Amplitude Modulated Signal Using 10-bit DAC

The DAC can be used to generate amplitude modulated signal. In this case, the DAC uses the voltage reference from the VREFA pin as modulation waveform (information signal), while it is configured to generate sine wave for the carrier signal.

```
VREF.DAC0REF = VREF_REFSEL_VREFA_gc;
```

$$V_{OUT} = \frac{DACnDATA \times VREFA}{1024}$$

Therefore, the amplitude of the carrier signal will vary according to the signal that needs to be modulated.

Important: The frequency of the carrier signal must be greater than that of the information signal in order for the resulting signal to be relevant.

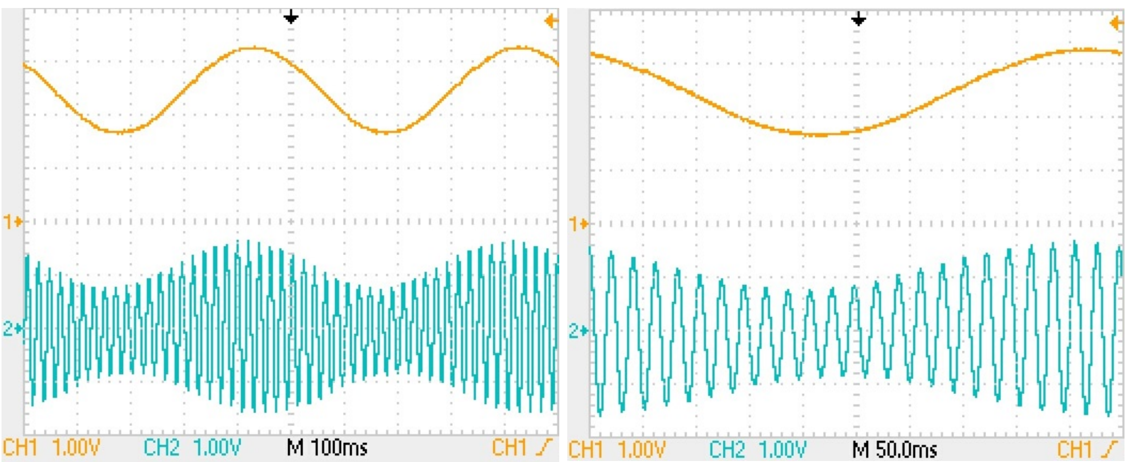
Based on the carrier wave frequency (SIGNAL_FREQ), the core will use a fixed number of samples (N_SAMPLES) to build the signal as described in "*Generating Sine Wave Signal Using 10-Bit DAC*". The resulting sample rate will be:

$$SAMPLE\_RATE = N\_SAMPLES * SIGNAL\_FREQ$$

The resulting modulated signal will be available on the DAC output external pin (PD6).

In this example, a 50 Hz sine wave is used as the carrier signal and the modulated signal is a 2 Hz sine wave from a signal generator.

Figure 6-1. Resulting Signal Visualized on Oscilloscope



View Code Example on GitHub
Click to browse repository

Tip: The full code example is also available in 8. Appendix.

7. References

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

8. Appendix

Example 8-1. Generating Constant Analog Signal Code

```
/* 4 MHz (needed for delay function) */
#define F_CPU                    (4000000UL)

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

/* DAC Value */
#define DAC_EXAMPLE_VALUE        (0x258)
/* VREF start-up time */
#define VREF_STARTUP_TIME        (50)
/* Mask needed to get the 2 LSb for DAC Data Register */
#define LSB_MASK                 (0x03)

static void VREF_init(void);
static void DAC0_init(void);
static void DAC0_setVal(uint16_t value);

static void VREF_init(void)
{
    VREF.DAC0REF = VREF_REFSEL_2V048_gc /* Select the 2.048V Internal Voltage Reference for DAC */
                 | VREF_ALWAYSON_bm;    /* Set the Voltage Reference in Always On mode */
    /* Wait VREF start-up time */
    _delay_us(VREF_STARTUP_TIME);
}

static void DAC0_init(void)
{
    /* Disable digital input buffer */
    PORTD.PIN6CTRL &= ~PORT_ISC_gm;
    PORTD.PIN6CTRL |= PORT_ISC_INPUT_DISABLE_gc;
    /* Disable pull-up resistor */
    PORTD.PIN6CTRL &= ~PORT_PULLUPEN_bm;
    DAC0.CTRLA = DAC_ENABLE_bm          /* Enable DAC */
               | DAC_OUTEN_bm           /* Enable output buffer */
               | DAC_RUNSTDBY_bm;       /* Enable Run in Standby mode */
}

static void DAC0_setVal(uint16_t value)
{
    /* Store the two LSbs in DAC0.DATAL */
    DAC0.DATAL = (value & LSB_MASK) << 6;
    /* Store the eight MSbs in DAC0.DATAH */
    DAC0.DATAH = value >> 2;
}

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

    DAC0_setVal(DAC_EXAMPLE_VALUE);

    while (1)
    {
        ;
    }
}
```

Example 8-2. Generating Sine Wave Signal Code

```
/* 4 MHz (needed for delay function) */
#define F_CPU                    (4000000UL)

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

/* VREF start-up time */
#define VREF_STARTUP_TIME        (50)
/* Mask needed to get the 2 LSb for DAC Data Register */
#define LSB_MASK                 (0x03)
/* Number of samples for a sine wave period */
#define SINE_PERIOD_STEPS        (100)
/* Sine wave amplitude */
#define SINE_AMPLITUDE           (511)
/* Sine wave DC offset */
#define SINE_DC_OFFSET           (512)
/* Frequency of the sine wave */
#define SINE_FREQ                (100)
/* Step delay for the loop */
#define STEP_DELAY_TIME          ((1000000 / SINE_FREQ) / SINE_PERIOD_STEPS)

static void sineWaveInit(void);
static void VREF_init(void);
static void DAC0_init(void);
static void DAC0_setVal(uint16_t value);

/* Buffer to store the sine wave samples */
uint16_t sineWave[SINE_PERIOD_STEPS];

static void sineWaveInit(void)
{
    uint8_t i;
    for(i = 0; i < SINE_PERIOD_STEPS; i++)
    {
        sineWave[i] = SINE_DC_OFFSET + SINE_AMPLITUDE * sin(2 * M_PI * i / SINE_PERIOD_STEPS);
    }
}

static void VREF_init(void)
{
    VREF.DAC0REF = VREF_REFSEL_2V048_gc /* Select the 2.048V Internal Voltage Reference for DAC */
                 | VREF_ALWAYSON_bm;    /* Set the Voltage Reference in Always On mode */
    /* Wait VREF start-up time */
    _delay_us(VREF_STARTUP_TIME);
}

static void DAC0_init(void)
{
    /* Disable digital input buffer */
    PORTD.PIN6CTRL &= ~PORT_ISC_gm;
    PORTD.PIN6CTRL |= PORT_ISC_INPUT_DISABLE_gc;
    /* Disable pull-up resistor */
    PORTD.PIN6CTRL &= ~PORT_PULLUPEN_bm;
    DAC0.CTRLA = DAC_ENABLE_bm          /* Enable DAC */
               | DAC_OUTEN_bm           /* Enable output buffer */
               | DAC_RUNSTDBY_bm;       /* Enable Run in Standby mode */
}

static void DAC0_setVal(uint16_t value)
{
    /* Store the two LSbs in DAC0.DATAL */
    DAC0.DATAL = (value & LSB_MASK) << 6;
    /* Store the eight MSbs in DAC0.DATAH */
    DAC0.DATAH = value >> 2;
}

int main(void)
{
    uint8_t sineIndex = 0;

    VREF_init();
    DAC0_init();

    sineWaveInit();

    while (1)
    {
        DAC0_setVal(sineWave[sineIndex++]);
        if(sineIndex == SINE_PERIOD_STEPS)
            sineIndex = 0;
        _delay_us(STEP_DELAY_TIME);
    }
}
```

Example 8-3. Reading the DAC Internally with the ADC Code

```
/* 4 MHz (needed for delay function) */
#define F_CPU                     (4000000UL)

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

/* VREF start-up time */
#define VREF_STARTUP_TIME       (50)
/* Mask needed to get the 2 LSb for DAC Data Register */
#define LSB_MASK                (0x03)

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

static void VREF_init(void)
{
    VREF.DAC0REF = VREF_REFSEL_2V048_gc /* Select the 2.048V Internal Voltage Reference for DAC */
                 | VREF_ALWAYSON_bm;    /* Set the Voltage Reference in Always On mode */
    VREF.ADC0REF = VREF_REFSEL_2V048_gc /* Select the 2.048V Internal Voltage Reference for ADC */
                 | VREF_ALWAYSON_bm;    /* Set the Voltage Reference in Always On mode */
    /* Wait VREF start-up time */
    _delay_us(VREF_STARTUP_TIME);
}

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

static void DAC0_setVal(uint16_t value)
{
    /* Store the two LSbs in DAC0.DATAL */
    DAC0.DATAL = (value & LSB_MASK) << 6;
    /* Store the eight MSbs in DAC0.DATAH */
    DAC0.DATAH = value >> 2;
}

static void ADC0_init(void)
{
    /* CLK_PER divided by 2 */
    ADC0.CTRLC = ADC_PRESC_DIV2_gc;
    ADC0.CTRLA = ADC_ENABLE_bm          /* Enable ADC */
               | ADC_RESSEL_12BIT_gc;   /* Use 12-bit resolution */
    /* Select ADC channel */
    ADC0.MUXPOS = ADC_MUXPOS_DAC0_gc
}

static uint16_t ADC0_read(void)
{
    /* Start conversion */
    ADC0.COMMAND = ADC_STCONV_bm;
    /* Wait until ADC conversion is done */
    while(!(ADC0.INTFLAGS & ADC_RESRDY_bm))
    {
        ;
    }
    /* The interrupt flag is cleared when the conversion result is accessed */
    return ADC0.RES;
}


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

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

    while (1)
    {
        adcVal = ADC0_read();

        /* do something with adcVal */

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

Example 8-4. Generating Amplitude Modulation Signal Code

```
/* 4 MHz (needed for delay function) */
#define F_CPU                   (4000000UL)

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

/* VREF start-up time */
#define VREF_STARTUP_TIME       (50)
/* Mask needed to get the 2 LSb for DAC Data Register */
#define LSB_MASK                (0x03)
/* Number of samples for a sine wave period */
#define SINE_PERIOD_STEPS       (100)
/* Sine wave amplitude */
#define SINE_AMPLITUDE          (511)
/* Sine wave DC offset */
#define SINE_DC_OFFSET          (512)
/* Frequency of the sine wave */
#define SINE_FREQ               (50)
/* Step delay for the loop */
#define STEP_DELAY_TIME         ((1000000 / SINE_FREQ) / SINE_PERIOD_STEPS)

static void PORT_init (void);
static void sineWaveInit(void);
static void VREF_init(void);
static void DAC0_init(void);
static void DAC0_setVal(uint16_t val);

/* Buffer to store the sine wave samples */
uint16_t sineWave[SINE_PERIOD_STEPS];

static void PORT_init (void)
{
    /* Set the VREFA pin (PD7) as input */
    PORTD.DIRCLR |= PIN7_bm;
}

static void sineWaveInit(void)
{
    uint8_t i;
    for(i = 0; i < SINE_PERIOD_STEPS; i++)
    {
        sineWave[i] = SINE_DC_OFFSET + SINE_AMPLITUDE * sin(2 * M_PI * i / SINE_PERIOD_STEPS);
    }
}

static void VREF_init(void)
{
    VREF.DAC0REF = VREF_REFSEL_VREFA_gc /* Select the External Reference from VREFA pin */
                 | VREF_ALWAYSON_bm;    /* Set the Voltage Reference in Always On mode */
    /* Wait VREF start-up time */
    _delay_us(VREF_STARTUP_TIME);
}

static void DAC0_init(void)
{
    /* Disable digital input buffer */
    PORTD.PIN6CTRL &= ~PORT_ISC_gm;
    PORTD.PIN6CTRL |= PORT_ISC_INPUT_DISABLE_gc;
    /* Disable pull-up resistor */
    PORTD.PIN6CTRL &= ~PORT_PULLUPEN_bm;
    DAC0.CTRLA = DAC_ENABLE_bm          /* Enable DAC */
               | DAC_OUTEN_bm           /* Enable output buffer */
               | DAC_RUNSTDBY_bm;       /* Enable Run in Standby mode */
}

static void DAC0_setVal(uint16_t value)
{
    /* Store the two LSbs in DAC0.DATAL */
    DAC0.DATAL = (value & LSB_MASK) << 6;
    /* Store the eight MSbs in DAC0.DATAH */
    DAC0.DATAH = value >> 2;
}

int main(void)
{
    uint8_t sineIndex = 0;

    PORT_init();
    VREF_init();
    DAC0_init();

    sineWaveInit();

    while (1)
    {
        DAC0_setVal(sineWave[sineIndex++]);
        if(sineIndex == SINE_PERIOD_STEPS)
            sineIndex = 0;
        _delay_us(STEP_DELAY_TIME);
```

```
        }
}
```

9. Revision History

Doc. Rev.	Date	Comments
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 http://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 http://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: http://www.microchip.com/support

Microchip Devices Code Protection Feature

Note the following details of the code protection feature on Microchip devices:

- Microchip products meet the specification contained in their particular Microchip Data Sheet.
- Microchip believes that its family of products is one of the most secure families of its kind on the market today, when used in the intended manner and under normal conditions.
- There are dishonest and possibly illegal methods used to breach the code protection feature. All of these methods, to our knowledge, require using the Microchip products in a manner outside the operating specifications contained in Microchip's Data Sheets. Most likely, the person doing so is engaged in theft of intellectual property.
- Microchip is willing to work with the customer who is concerned about the integrity of their code.
- Neither Microchip nor any other semiconductor manufacturer can guarantee the security of their code. Code protection does not mean that we are guaranteeing the product as "unbreakable."

Code protection is constantly evolving. We at Microchip are committed to continuously improving the code protection features of our products. Attempts to break Microchip's code protection feature may be a violation of the Digital Millennium Copyright Act. If such acts allow unauthorized access to your software or other copyrighted work, you may have a right to sue for relief under that Act.

Legal Notice

Trademarks

ISBN: 978-1-5224-6064-0

Quality Management System

For information regarding Microchip's Quality Management Systems, please visit http://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: http://www.microchip.com/support Web Address: http://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