XMC4000

32-bit Microcontroller Series for Industrial Applications

Digital to Analog Converter (DAC)

AP32301

Application Note

About this document

Scope and purpose

This document describes the features of the DAC peripheral and how to configure it for common cases such as data processing, pattern generation and noise generation.

Applicable Products

- XMC4000 Microcontrollers Family

References

Infineon: Example code: http://www.infineon.com/XMC4000 Tab: Documents

Infineon: XMC Lib, http://www.infineon.com/DAVE

Infineon: DAVE™, http://www.infineon.com/DAVE

Infineon: XMC Reference Manual, http://www.infineon.com/XMC4000 Tab: Documents

Infineon: XMC Data Sheet, http://www.infineon.com/XMC4000 Tab: Documents

Table of Contents

1 Digital to Analog Converter (DAC)

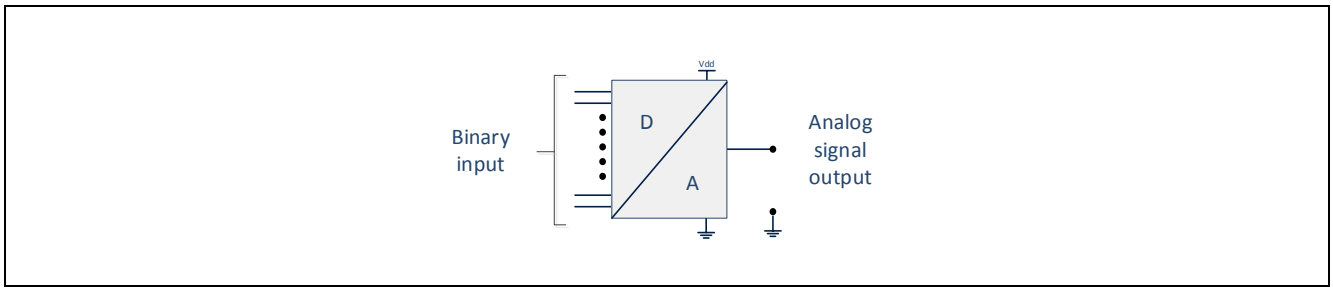

Figure 1 Digital to Analog Converter

A digital to analog converter is a module to convert one or multiple bits digital value into an analog value. It is defined through:

- resolution
- conversation rate
- offset
- accuracy of the analog output
- drive capability

The DAC refill method has an influence on the system performance. To reduce the load on the CPU, the DAC can be refilled by Direct Memory Access (DMA) or the DAC module itself provides waveform generating parts.

In addition in some applications a data refinement, for example a static offset or multiplication, is necessary.

1.1 XMC DAC operation modes

The DAC module provides five operation modes:

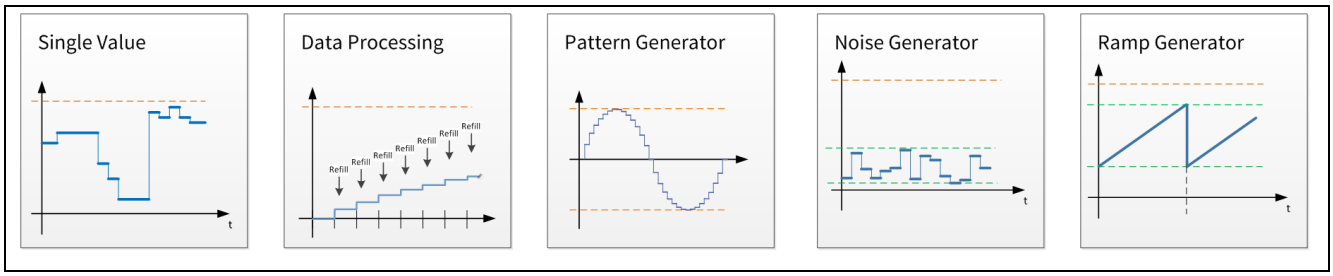

Figure 2 DAC modes overview

The modes differentiate in the refill method to reduce the CPU load. The operation modes are described in the following sections.

1.2 XMC DAC input types

For each operation mode a specific input type is available.

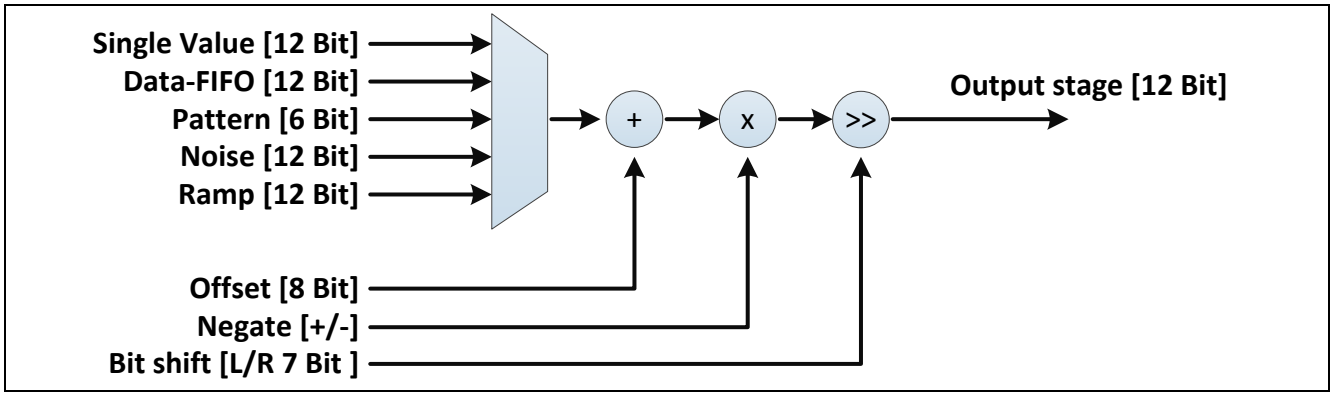

Figure 3 DAC input types and data manipulation

The **Single Value** input provides a fast access to the analog output. This input is used for non-periodic or software controlled output.

The **Data-FIFO** input enables the fill stage and the DAC output stage to be decoupled. This input is used for periodic updates. The refill can be done by software (SW) or hardware (HW) via DMA trigger.

The **Pattern** generator has a 6 bit output. A customized symmetric pattern can be generated, where the first quarter of the pattern is defined by the user.

The **Noise** generator provides a pseudo random value with 12 bit resolution.

The **Ramp** generator provides a ramp with 12 bit resolution, where the value is increased by each clock cycle and the start and stop value can be defined by the user.

The five inputs represent also the five modes of the DAC

- **Single Value** input: **Single Value mode**
- **Data-FIFO** input: **Data Processing mode**
- **Pattern** input: **Pattern Generator mode**
- **Noise** input: **Noise Generator mode**
- **Ramp** input: **Ramp Generator mode**

1.3 XMC DAC data refinement

The **Output stage** is the 12 bit value which is transformed into an analog value. It can be interpreted as unsigned or signed value. In unsigned mode, a 0 is interpreted as lowest analog value. In signed mode, a 0 generates the center voltage. The data can be refined between input and the **Output stage**. The refinement options are: **Offset**, **Negate** and **Bit shift**.

The XMC provides an 8 bit **Offset** which is added to the input. The sign interpretation of the **Offset** follows the **Output stage**.

A **Negate** bit inverts the sign of the input. This can be used with the **Single Value** or **Data Processing mode** as HW inversion, in the **Pattern Generator mode** to start with a negative half wave or in **Ramp Generator mode** to generate a falling ramp. The negate bit is available on specific devices only.

The **Bit shift** allows a multiplication or division of the input. This means the input can either multiplied or divided by 2^1 to 2^7 [2…128]. This can then used for the following use cases:

- scaling in **Single Value** or **Data Processing mode**
- scale the pattern from the **Pattern Generator mode** to the DAC output range
- reduce the scale in the **Noise Generator mode**
- expand the minimum and maximum frequency in the **Ramp Generator mode**

1.4 Basic DAC equations

The transformation from unsigned output stage to analog value can be done with following calculations.

- DEC_{target} Target decimal value in the DAC output register
- S_DEC_{target} Target signed decimal value in the DAC output register
- DEC_{res} Decimal resolution of the DAC
- V_{target} Target DAC output Voltage
- V_{max} Maximum DAC output Voltage
- V_{offset} DAC offset Voltage

DAC_{res}, V_{max} and V_{offset} can be found in the Datasheet.

$$DEC_{target} = \frac{V_{target} - V_{offset}}{V_{max} - V_{offset}} * DEC_{res}$$

$$V_{target} = \left(DEC_{target} * \frac{(V_{max} - V_{offset})}{DEC_{res}}\right) + V_{offset}$$

If the Output stage is interpreted as signed, the following calculations can be used:

$$S\_DEC_{target} = \left(\frac{V_{target} - V_{offset}}{V_{max} - V_{offset}} * DEC_{res}\right) - \frac{DEC_{res} + 1}{2}$$

$$V_{target} = \left(\left(S\_DEC_{target} + \frac{DEC_{res} + 1}{2}\right) * \frac{(V_{max} - V_{offset})}{DEC_{res}}\right) + V_{offset}$$

The following example shows the equations with V_{max}=2.5V, V_{offset}=0.3V, DAC_{res}=(2^{12}-1):

$$DEC_{target} = \frac{V_{target} - 0{,}3V}{2{,}2V} * 4095$$

$$V_{target} = \left(DEC_{target} * \frac{2{,}2V}{4095}\right) + 0{,}3V$$

$$S\_DEC_{target} = \left(\frac{V_{target} - 0{,}3V}{2{,}2V} * 4095\right) - 2048$$

$$V_{target} = \left((S\_DEC_{target} + 2048) * \frac{2{,}2V}{4095}\right)$$

2 Single Value mode

The single value mode can be used for simple digital to analog conversion. Writing to the DATA Register leads to a change of the analogue output. The DAC can be modified to accept unsigned or signed data. The following data refinement is available: offset, scaling and negation.

If channel 0 and channel 1 are used and have to be updated simultaneously, the synchronization feature can be used. In this mode the data for both channels can be written into one register. With a trigger of channel 0 both values are updated. A trigger can either be a software trigger or a pseudo write access to the original channel 0 data register.

2.1 XMC Lib configuration

This example shows two DAC configurations in Single Value mode. The DACs are filled by software. If a new value is written into the data register it is emitted with the next DAC clock. The channels are not synchronized. The first channel is configured for unsigned, the second for signed values. Both channels do not use the scaling and negation feature. Both channels are initialized with the corresponding configuration: Channel 0 as unsigned DAC and channel 1 as signed DAC.

```
#define DAC_CH_NR_0       0U
#define DAC_CH_NR_1       1U

XMC_DAC_CH_CONFIG_t ch_config0=
{
   .output_offset   =     0U,
   .data_type       =     XMC_DAC_CH_DATA_TYPE_UNSIGNED,
   .output_scale    =     XMC_DAC_CH_OUTPUT_SCALE_NONE,
   .output_negation =     XMC_DAC_CH_OUTPUT_NEGATION_DISABLED,
};

XMC_DAC_CH_CONFIG_t ch_config1=
{
   .output_offset   =     0U,
   .data_type       =     XMC_DAC_CH_DATA_TYPE_SIGNED,
   .output_scale    =     XMC_DAC_CH_OUTPUT_SCALE_NONE,
   .output_negation =     XMC_DAC_CH_OUTPUT_NEGATION_DISABLED,
};
```

2.2 XMC Lib initialization

The DAC channel is initialized in general and then configured to the specific mode.

```
XMC_DAC_CH_Init(XMC_DAC0, DAC_CH_NR_0, &ch_config0);
XMC_DAC_CH_Init(XMC_DAC0, DAC_CH_NR_1, &ch_config1);
```

After the initialization the DAC mode is changed to “Single Value Mode”.

```
XMC_DAC_CH_StartSingleValueMode(XMC_DAC0, DAC_CH_NR_0);
XMC_DAC_CH_StartSingleValueMode(XMC_DAC0, DAC_CH_NR_1);
```

2.3 Function implementation

And the DAC channels are filled with the value 0. As the channel 0 is configured as unsigned the zero will be interpreted as minimum voltage. While the channel 1 is configured as signed the value zero will be interpreted as center voltage. Channel 0 needs to be filled with unsigned 12 bit values and channel 1 with signed 12 bit values. The “XMC_DAC_CH_Write()” function can be called periodically in order to update and trigger the DAC conversion. The call frequency should not exceed the maximum DAC update frequency.

```
XMC_DAC_CH_Write(XMC_DAC0, DAC_CH_NR_0, 0x0000U);
XMC_DAC_CH_Write(XMC_DAC0, DAC_CH_NR_1, 0x0000U);
```

3 Data Processing mode

The Data Processing mode is similar to the Single Value mode expect that it supports an interrupt based reload of the DAC. The reload does not trigger a conversion and can be done via DMA or SW.

The Data Processing mode allows large and precise tables for waveform generation, where data are stored in the flash or RAM.

3.1 XMC Lib configuration

This example shows a DAC configuration in Data Processing mode. When a trigger occurs the DAC is filled by software in the interrupt service routine (ISR). The DAC module generates a trigger with 10 kHz by itself.

```
#define DAC_CH_NR_0          0U
#define Trigger_ISR   DAC0_0_IRQHandler

XMC_DAC_CH_CONFIG_t ch_config0=
{
   .output_offset     =      0U,
   .data_type         =      XMC_DAC_CH_DATA_TYPE_UNSIGNED,
   .output_scale      =      XMC_DAC_CH_OUTPUT_SCALE_NONE,
   .output_negation   =      XMC_DAC_CH_OUTPUT_NEGATION_DISABLED,
};
```

3.2 XMC Lib initialization

The DAC channel is initialized in general and then configured to the specific mode.

```
XMC_DAC_CH_Init(XMC_DAC0, DAC_CH_NR_0, &ch_config0);
```

After the general initialization the DAC needs to be configured to Data Process mode (“DataMode”). The DAC trigger is generated by the DAC module. Therefore the frequency is set to 10 kHz. If an external trigger is used, the frequency selection is not required and can be set to zero. After the DAC is initialized, the DAC event and service request are enabled.

```
XMC_DAC_CH_StartDataMode(XMC_DAC0,DAC_CH_NR_0,
                         XMC_DAC_CH_TRIGGER_INTERNAL,
                         10000U);

XMC_DAC_CH_EnableEvent(XMC_DAC0,DAC_CH_NR_0);
NVIC_EnableIRQ(DAC0_0_IRQn);
```

3.3 Function implementation

In the DAC0 ISR the new value from the data table is written into the DATA register.

```
void DAC0_0_IRQHandler(void)
{
        XMC_DAC_CH_Write(XMC_DAC0,DAC_CH_NR_0,table_u[table_i]);
        table_i++;
        if(table_i>=LENGTH_OF_TABLE)
        {table_i=0;}
};
```

4 Pattern Generator mode

The Pattern Generator mode generates a pattern without software load. One quarter of the waveform needs to be defined. The rest of the waveform is generated by horizontal and vertical mirroring of the first quarter.

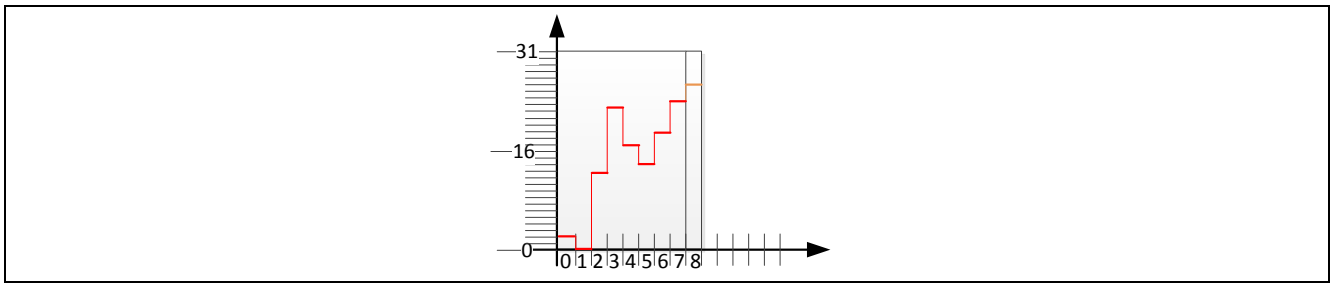

Figure 4 DAC pattern configuration

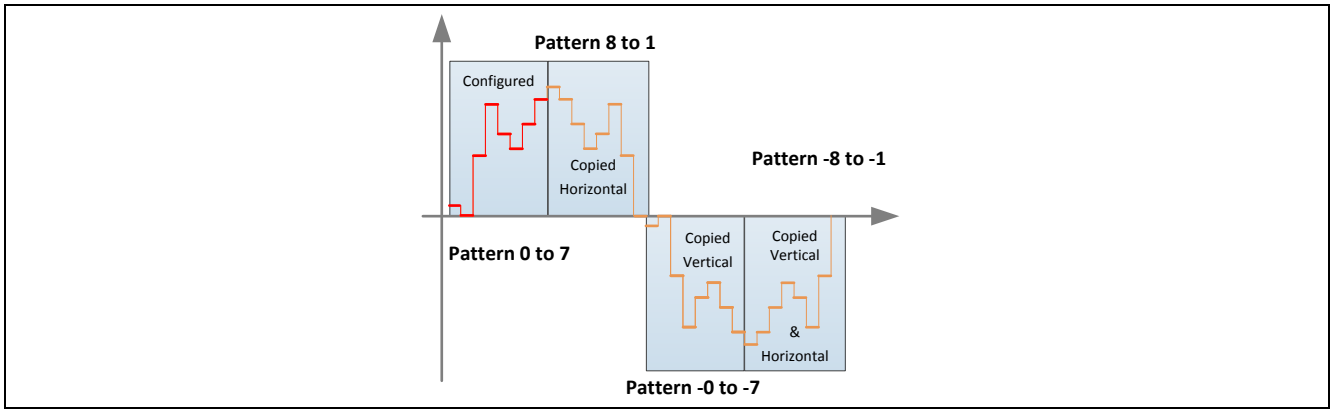

Figure 5 DAC pattern wave

The wave is fixed to a length resolution of 32 steps and an amplitude resolution of 64 steps. The 6 bit amplitude resolution can be scaled to reach the 12 bit DAC output.

With negation enabled the wave starts inverted:

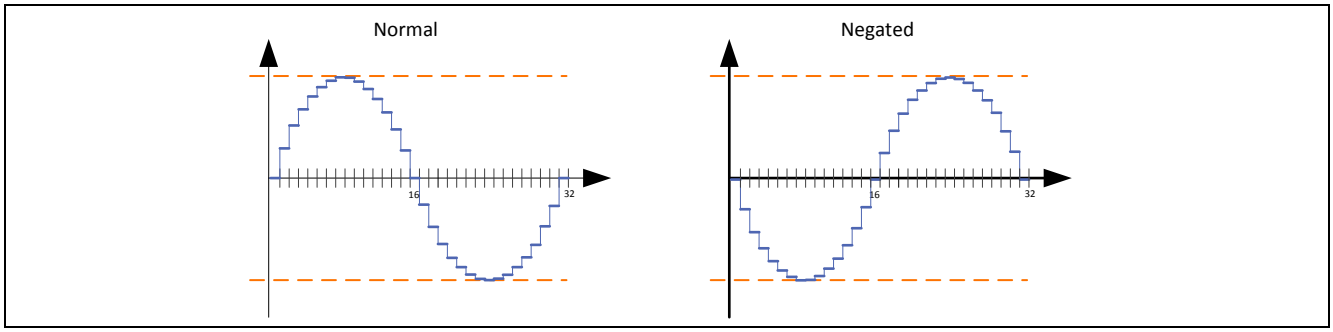

Figure 6 DAC Pattern negated

An offset can also be added. The offset is in the Pattern Generator scale (6 bit). If a scaling is added, the pattern is scaled after adding the offset.

The Pattern Generator also provides a sign output. This can be used to trigger the ADC or routed through the ERU to trigger other peripherals such as a timer. The signal can also be routed through the ERU to a GPIO.

4.1 XMC Lib configuration

This example shows how the DAC can be configured in Pattern Generator mode. Channel 0 is configured to generate a 500Hz sine wave. The output of the Pattern Generator is a 6 bit signed data. Therefore the DAC is configured in signed mode. To reach the 12 bit DAC scale the data need to be multiplied by 64 (shifted by 6 bit). Neither the offset nor the negation is used.

The channel 0 is initialized with the corresponding configuration:

```
#define DAC_CH_NR_0       0U

XMC_DAC_CH_CONFIG_t ch_config0=
{
   .output_offset   =     0U,
   .data_type       = XMC_DAC_CH_DATA_TYPE_SIGNED,
   .output_scale    = XMC_DAC_CH_OUTPUT_SCALE_MUL_64,
   .output_negation= XMC_DAC_CH_OUTPUT_NEGATION_DISABLED,
};
```

4.2 XMC Lib initialization

The DAC channel is initialized in general and then configured to the specific mode.

```
XMC_DAC_CH_Init(XMC_DAC0, DAC_CH_NR_0, &ch_config0);
```

After the general initialization the DAC needs to be configured to Pattern Generator mode (“PatternMode”). One of the three predefined patterns, the sine pattern, is used. The sign output is disabled, the DAC clock source is set to intern and the pattern frequency is set to 500Hz.

```
const uint8_t pattern[] = XMC_DAC_PATTERN_SINE;

XMC_DAC_CH_StartPatternMode(XMC_DAC0, DAC_CH_NR_0,
                    pattern,
                    XMC_DAC_CH_PATTERN_SIGN_OUTPUT_DISABLED,
                    XMC_DAC_CH_TRIGGER_INTERNAL,
                    500);
```

An additional function implementation is not necessary.

5 Noise Generator mode

The noise generator mode provides a 20 bit linear feedback shift register to produce a pseudo random number. The 12 bit random numbers are sampled with the preselected trigger. The output can be additionally scrambled by changing the scaling, the offset, the negation and the signed/unsigned interpretation at runtime.

By using negative scaling (division by 2^x) the amplitude can be limited. If the Output stage is interpreted as signed value, an offset of half the DAC output is generated.

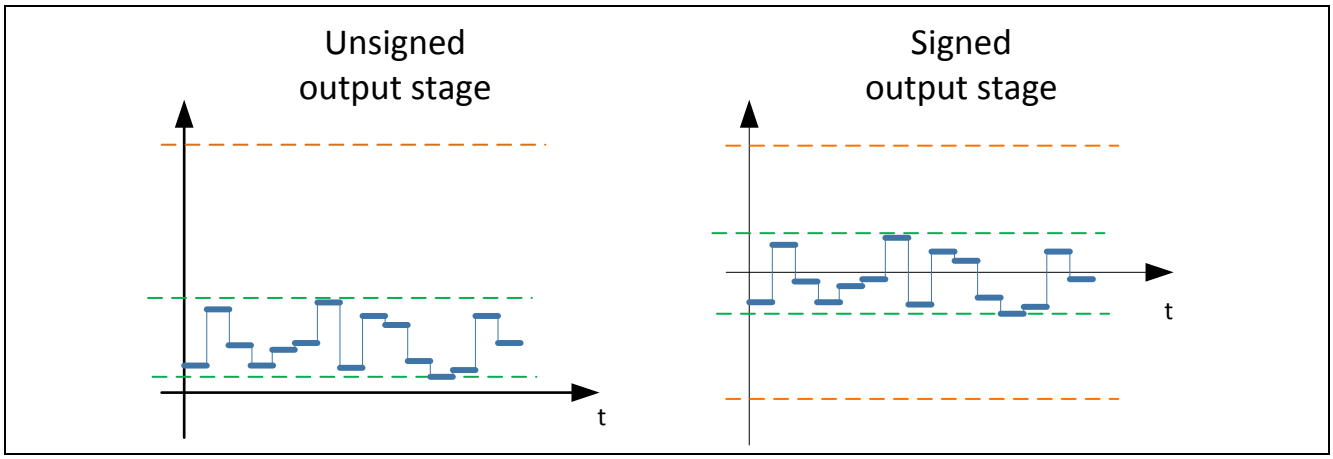

Figure 7 DAC noise sign and unsigned output stage

5.1 XMC Lib configuration

In this example the DAC is configured in Noise Generator mode. The output of the noise generator is 12 bit unsigned data. Therefore the offset is set to zero, the data type is set to unsigned and the scaling and the negation are disabled.

The channel 0 is initialized with the corresponding configuration:

```
#define DAC_CH_NR_0       0U

XMC_DAC_CH_CONFIG_t ch_config0 =
{
   .output_offset   =     0U,
   .data_type       = XMC_DAC_CH_DATA_TYPE_UNSIGNED,
   .output_scale    = XMC_DAC_CH_OUTPUT_SCALE_NONE,
   .output_negation= XMC_DAC_CH_OUTPUT_NEGATION_DISABLED,
};
```

5.2 XMC Lib initialization

The DAC channel is initialized in general and then configured to the specific mode.

```
XMC_DAC_CH_Init(XMC_DAC0, DAC_CH_NR_0, &ch_config0);
```

After the general initialization the DAC needs to be configured to Noise Generator mode (“NoiseMode”). The DAC clock source is set to internal and the noise frequency is set to 50 kHz.

```
XMC_DAC_CH_StartNoiseMode(XMC_DAC0, DAC_CH_NR_0,
             XMC_DAC_CH_TRIGGER_INTERNAL,
             50000U);
```

An additional function implementation is not necessary.

6 Ramp Generator mode

The Ramp Generator mode generates a voltage ramp with a 12 bit resolution. The amplitude is defined by the start and stop value. The slope is defined by the clock input.

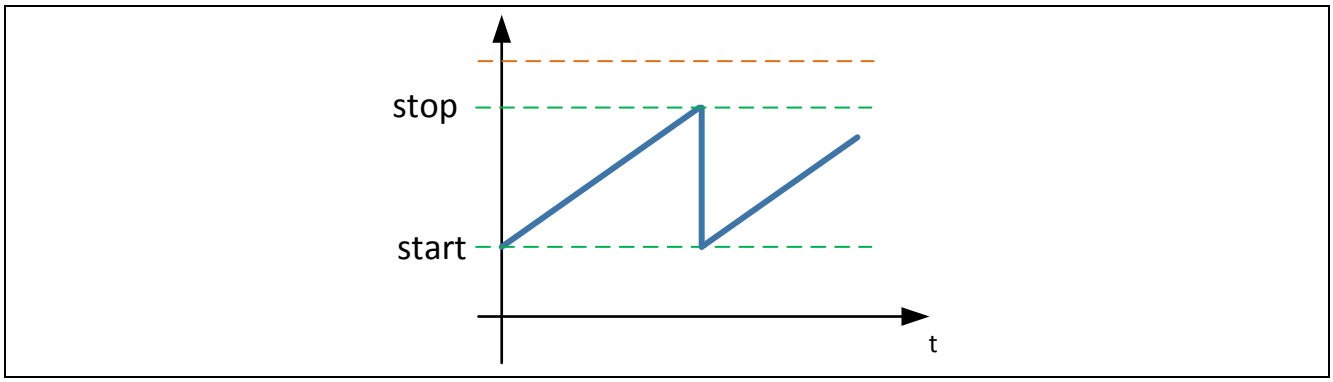

Figure 8 DAC ramp mode

The negation feature allows also a negative ramp. The negation bit builds the 2's complement of the start and stop value in 12 bit format. Therefore the initial start and stop values also need to be 2's complement format and swapped. For example original start value is 100U and stop value is 3500U. The 2's complement of 100 in 12 bit format is 3996U and of 3500U is 569U. Now the values need to be swapped. The new start value is 596U and stop value is 3996U.

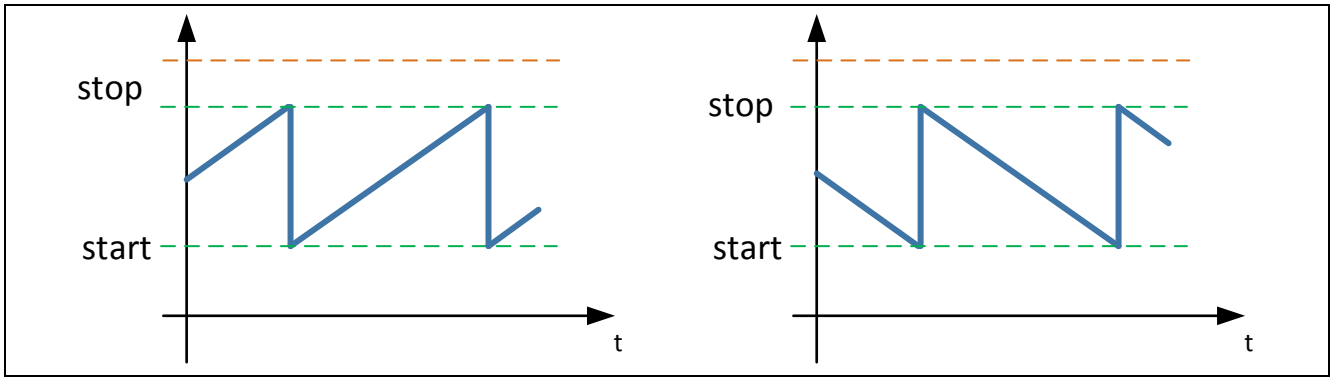

Figure 9 DAC ramp positive and negative slope

With the scaling feature minimum and maximum ramp frequency can be enlarged:

The slope of the ramp is defined by the DAC clock. Each clock cycle increases the digital output by one. When small amplitudes are used, this can limit the minimum ramp frequency. This can be compensated by doubling the start and stop value and adding a negative scale 2^{x-1}.

This can also be done in the opposite way. Full scaled amplitude limits the maximum ramp frequency. This can be changed by halving the start and stop values and multiplying two. This reduces the ramp resolution.

6.1 XMC Lib configuration

This example configures a DAC channel in Ramp Generator mode. The Ramp Generator output is a 12 bit unsigned value. Therefore the DAC channel is configured as unsigned and unscaled. The offset and the negation are disabled.

```
#define DAC_CH_NR_0       0U

XMC_DAC_CH_CONFIG_t ch_config0=
{
   .output_offset   =     0U,
   .data_type       = XMC_DAC_CH_DATA_TYPE_UNSIGNED,
   .output_scale    = XMC_DAC_CH_OUTPUT_SCALE_NONE,
   .output_negation= XMC_DAC_CH_OUTPUT_NEGATION_DISABLED,
};
```

6.2 XMC Lib initialization

The DAC channel is initialized in general and then configured to the specific mode.

```
XMC_DAC_CH_Init(XMC_DAC0, DAC_CH_NR_0, &ch_config0);
```

After the general initialization the DAC needs to be configured to Ramp Generator mode (“RampMode”). The START_VALUE is defined as 80 the STOP_VALUE as 3500. The DAC clock source is set to internal and the ramp frequency is set to 500Hz.

```
#define START_VALUE       80
#define STOP_VALUE        3500

XMC_DAC_CH_StartRampMode(XMC_DAC0, DAC_CH_NR_0,
                         START_VALUE,
                         STOP_VALUE,
                         XMC_DAC_CH_TRIGGER_INTERNAL,
                         500U);
```

An additional function implementation is not necessary.

7 Revision History

Current Version is V1.0, 2015-07

Page or Reference	Description of change
V1.0, 2015-07	
	Initial Version

www.infineon.com

Edition 2015-07

Published by

Infineon Technologies AG

81726 Munich, Germany

Do you have a question about any aspect of this document?

Email: erratum@infineon.com

Document reference

AP32301

Legal Disclaimer

THE INFORMATION GIVEN IN THIS APPLICATION NOTE (INCLUDING BUT NOT LIMITED TO CONTENTS OF REFERENCED WEBSITES) IS GIVEN AS A HINT FOR THE IMPLEMENTATION OF THE INFINEON TECHNOLOGIES COMPONENT ONLY AND SHALL NOT BE REGARDED AS ANY DESCRIPTION OR WARRANTY OF A CERTAIN FUNCTIONALITY, CONDITION OR QUALITY OF THE INFINEON TECHNOLOGIES COMPONENT. THE RECIPIENT OF THIS APPLICATION NOTE MUST VERIFY ANY FUNCTION DESCRIBED HEREIN IN THE REAL APPLICATION. INFINEON TECHNOLOGIES HEREBY DISCLAIMS ANY AND ALL WARRANTIES AND LIABILITIES OF ANY KIND (INCLUDING WITHOUT LIMITATION WARRANTIES OF NON-INFRINGEMENT OF INTELLECTUAL PROPERTY RIGHTS OF ANY THIRD PARTY) WITH RESPECT TO ANY AND ALL INFORMATION GIVEN IN THIS APPLICATION NOTE.

Information

For further information on technology, delivery terms and conditions and prices, please contact the nearest Infineon Technologies Office (**www.infineon.com**).

Warnings

Due to technical requirements, components may contain dangerous substances. For information on the types in question, please contact the nearest Infineon Technologies Office. Infineon Technologies components may be used in life-support devices or systems only with the express written approval of Infineon Technologies, if a failure of such components can reasonably be expected to cause the failure of that life-support device or system or to affect the safety or effectiveness of that device or system. Life support devices or systems are intended to be implanted in the human body or to support and/or maintain and sustain and/or protect human life. If they fail, it is reasonable to assume that the health of the user or other persons may be endangered.