AN3126
Application note

Audio and waveform generation using the DAC in STM32 microcontrollers

Introduction

This application note provides some examples for generating audio waveforms using the Digital to Analog Converter (DAC) peripheral embedded in the microcontrollers of the STM32Fx and STM32Lx series.

This document applies to products listed in *Table 1*, and should be read in connection with application note AN4566 “Extending the DAC performance of STM32 microcontrollers”.

A digital to analog converter, DAC, is a device that has the opposite function to that of an analog to digital converter, i.e. it converts a digital word to a corresponding analog voltage.

The STM32 DAC module is a 12-bit word converter, with up to three output channels to support audio functions.

The DAC can be used in many audio applications such as security alarms, Bluetooth headsets, talking toys, answering machines, man-machine interfaces, and low-cost music players

STM32 DAC can also be used for many other analog purposes, such as analog waveform generation and control engineering.

The application note is organized in two main sections:

- *Section 1* describes the main features of the STM32 DAC module.
- *Section 2* presents two examples.
  - In the first example, the DAC is used to generate a sine wavefom.
  - In the second example, the DAC is used to generate audio from .WAV files.

Table 1. Applicable products

Type	Product series
Microcontrollers	STM32F0
	STM32F1
	STM32F2
	STM32F3
	STM32F4
	STM32F7
	STM32L0
	STM32L1
	STM32L4

Contents

List of tables

List of figures

1 DAC main features

1.1 Overview

STM32 microcontrollers integrate DAC with different configurations and features:

- 1 to 3 DAC output channels
- Noise waveform generation
- Triangular waveform generation
- DMA under run flag
- Dedicated analog clock

Table 2 summarizes the different STM32 DAC configuration.

Table 2. DAC configurations for STM32 microcontrollers

Series	Product RPN	DAC outputs	White noise generator	Triangular wave generator	DMA capability	DMA underrun error
F0	STM32F030xx STM32F031xx STM32F038xx STM32F042xx STM32F048xx STM32F070xx	0	-	-	-	-
	STM32F051xx STM32F058xx	1	No	No	Yes	No
	STM32F071xx STM32F072xx STM32F078xx STM32F091xx STM32F098xx	2	Yes	Yes	Yes	Yes
F1	STM32F101x4/6/8B STM32F102xx STM32F103x4/6/8B	0	-	-	-	-
	STM32F100xx STM32F101xC/D/E/F/G STM32F103xC/D/E/F/G STM32F105xx STM32F107xx	2	Yes	Yes	Yes	Yes
F2	STM32F2xxxx	2	Yes	Yes	Yes	Yes

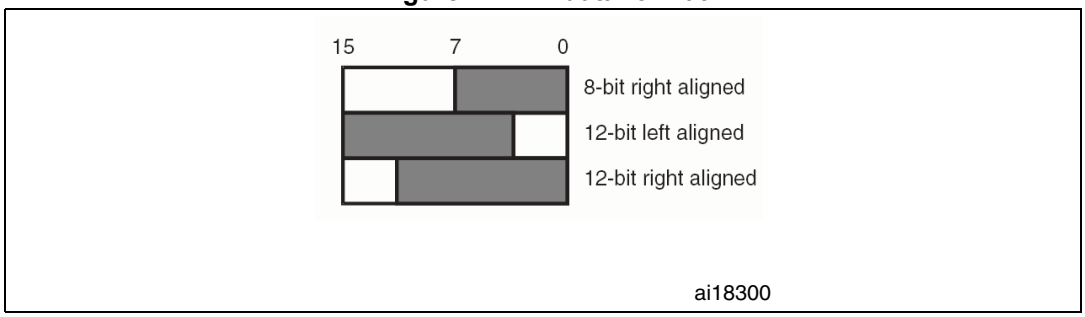
Table 2. DAC configurations for STM32 microcontrollers (continued)

Series	Product RPN	DAC outputs	White noise generator	Triangular wave generator	DMA capability	DMA underrun error
F3	STM32F301xx STM32F302xx STM32F318xx	1	Yes	Yes	Yes	Yes
	STM32F303xB/C/D/E STM32F358xx STM32F398xx	2	Yes	Yes	Yes	Yes
	STM32F3328 STM32F3334 STM32F3373 STM32F3378	3	Yes (only for 2 channels)	Yes (only for 2 channels)	Yes	Yes
F4	STM32F401xx STM32F411xx	0	-	-	-	-
	STM32F405xx STM32F407xx STM32F415xx STM32F417xx STM32F427xx STM32F429xx STM32F437xx STM32F439xx STM32F446xx	2	Yes	Yes	Yes	Yes
F7	STM32F7xxxx	2	Yes	Yes	Yes	Yes
L0	STM32L031xx STM32L041xx STM32L051xx STM32L071xx STM32L081xx	0	-	-	-	-
	STM32L052xx STM32L053xx STM32L062xx STM32L063xx	1	Yes	Yes	Yes	Yes
L1	STM32L1xxxx	2	Yes	Yes	Yes	Yes
L4	STM32L4xxxx	2	Yes	Yes	Yes	Yes

1.2 Data format

The DAC accepts data in 3 integer formats: 8-bit, 12-bit right aligned and 12-bit left aligned. A 12-bit value can range from 0x000 to 0xFFF, with 0x000 being the lowest and 0xFFF being the highest value.

Figure 1. DAC data format



1.3 Dual channel mode

Note: This feature is supported only for products that embed at least 2 DACs.

The DAC has two output channels, each with its own converter. In dual DAC channel mode, conversions could be done independently or simultaneously.

When the DAC channels are triggered by the same source, both channels are grouped together for synchronous update operations and conversions are done simultaneously.

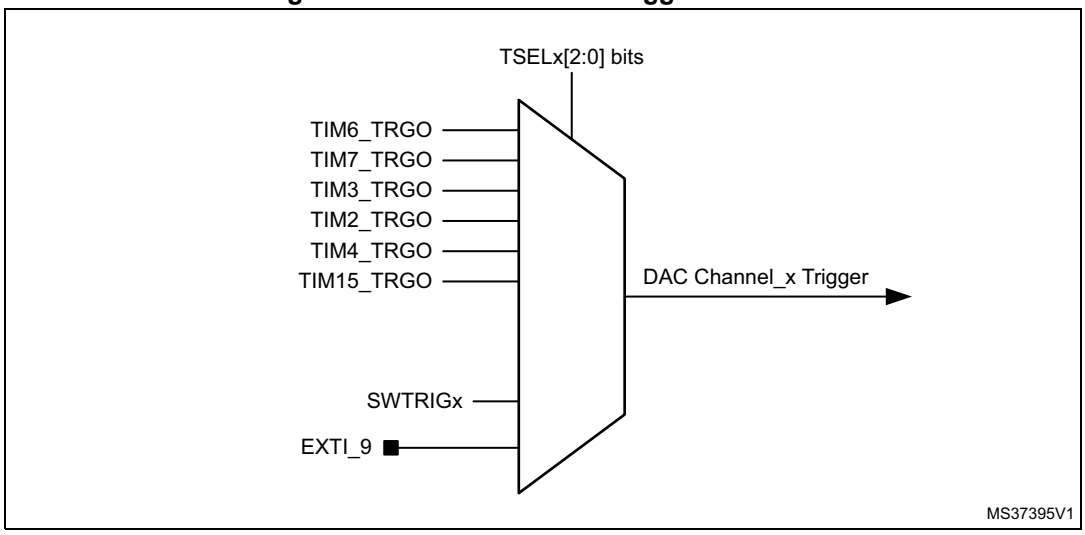
1.4 Dedicated timers

In addition to the software and External triggers, the DAC conversion can be triggered by different timers.

TIM6 and TIM7 are basic timers and are basically designed for DAC triggering.

Each time a DAC interface detects a rising edge on the selected Timer Trigger Output (TIMx_TRGO), the last data stored in the DAC_DHRx register is transferred to the DAC_DORx register (an example for STM32F100x is given in *Figure 2*).

Figure 2. STM32F100x DAC trigger channels

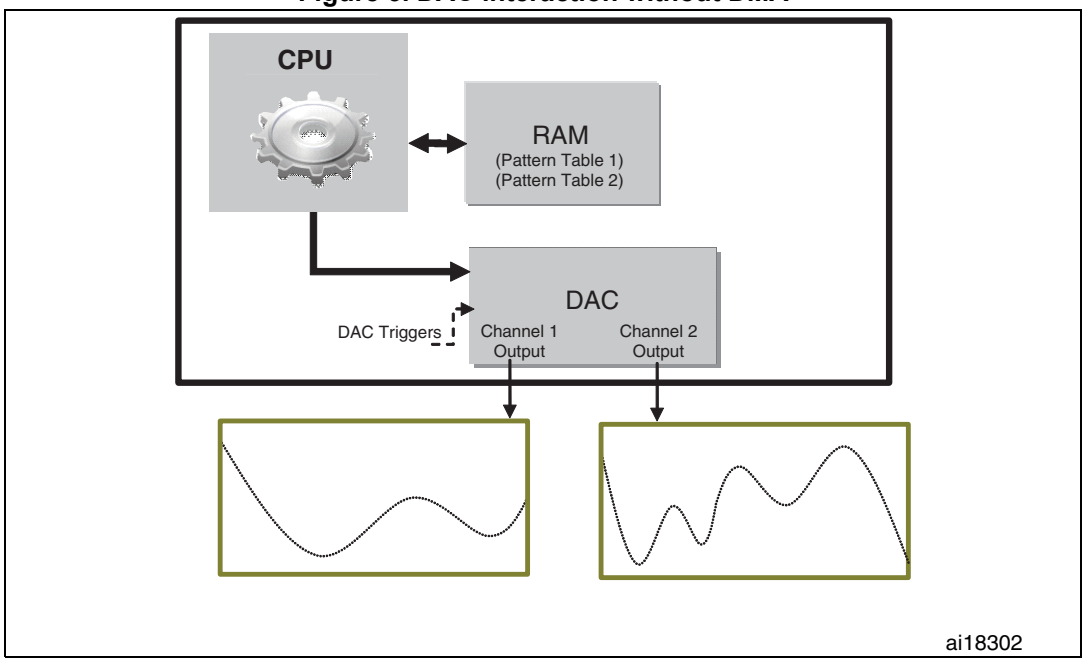


1.5 DMA capabilities

The STM32 microcontrollers have a DMA module with multiple channels. Each DAC channel is connected to an independent DMA channel. In the case of STM32F100x Microcontrollers, the DAC channel 1 is connected to the DMA channel 3 and DAC channel2 is connected to DMA channel 4.

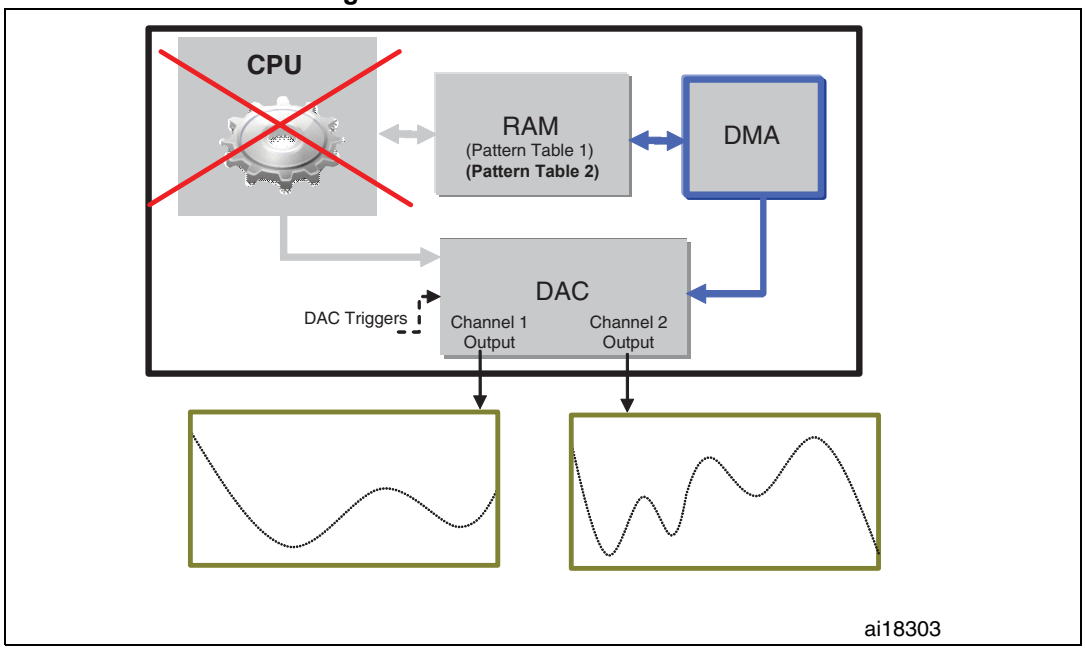
When DMA is not utilized, the CPU is used to provide DAC with the pattern waveform. Generally the waveform is saved in a memory (RAM), and the CPU is in charge of transferring the data from RAM to the DAC.

Figure 3. DAC interaction without DMA



When using the DMA, the overall performance of the system is increased by freeing up the core. This is because data is moved from memory to DAC by DMA, without needing any actions by the CPU. This keeps CPU resources free for other operations.

Figure 4. DAC interaction with DMA



1.6 DMA underrun error

When the DMA is used to provide DAC with the pattern waveform, there are some cases where the DMA transfer is faster than the DAC conversion, In this case, DAC detects that a part of the pattern waveform has been ignored and was not converted. It then sets the "DMA underrun Error" flag.

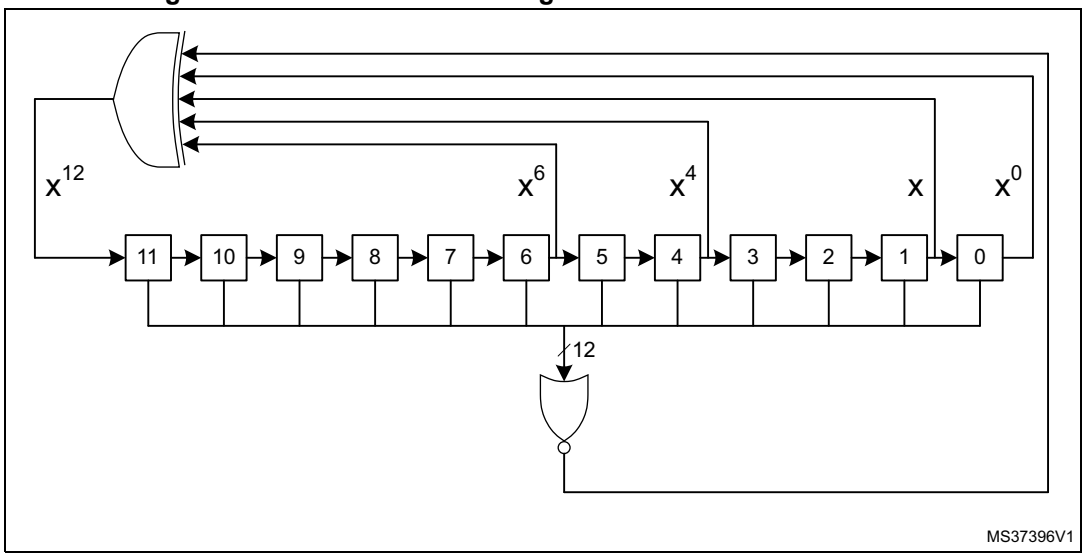
The underrun error can be handled using a shared IRQ channel with the triggering Timer or by a dedicated interrupt when DAC is not triggered by TIM6.

1.7 White noise generator

1.7.1 Definition

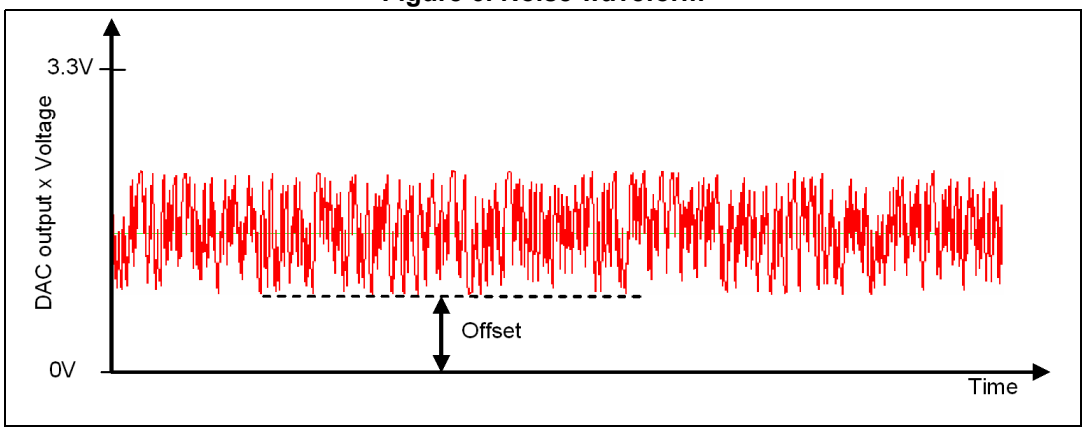
The STM32 microcontrollers DAC provides user with a pseudo random code generator, sketched in *Figure 5*. Depending on what taps are used on the shift register, a sequence of up to 2^{n-1} numbers can be generated before the sequence repeats.

Figure 5. Pseudo random code generator embedded in the DAC



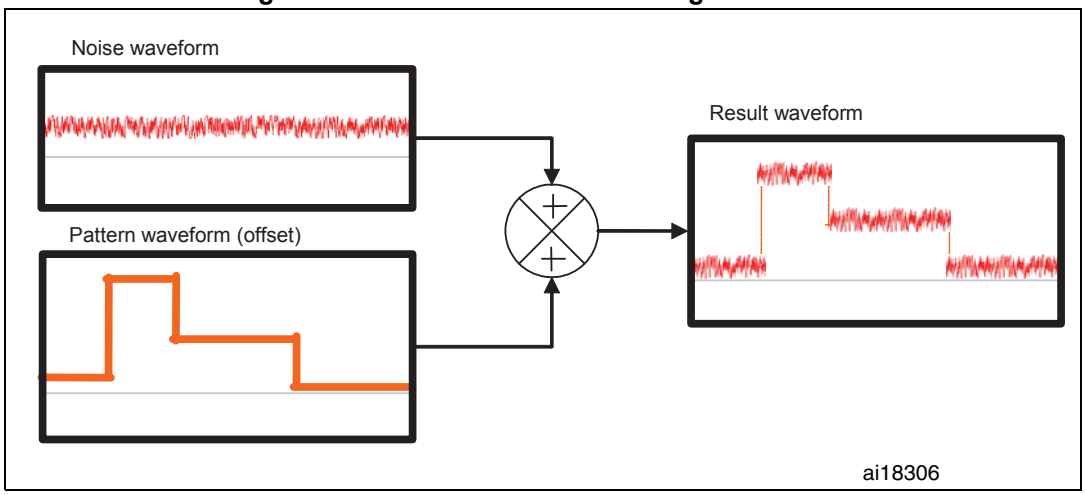
The noise produced by this generator has a flat spectral distribution and can be considered white noise. However, instead of having a Gaussian output characteristics, it is uniformly distributed, see *Figure 6*.

Figure 6. Noise waveform



The offset of the noise waveform is programmable. By varying this offset using a preconfigured table of offsets (signal pattern), user can obtain a waveform which correspond to the sum of the signal pattern and the noise waveform.

Figure 7. Noise waveform with changeable offset



1.7.2 Typical applications

The STM32 microcontrollers come with 12-bit enhanced ADC with a sampling rate of up to 1 M samples/s. In most applications, this resolution is sufficient, but in some cases where higher accuracy is required, the concept of oversampling and decimating the input signal can be implemented to save the use of an external ADC solution and to reduce the application power consumption.

More details about these methods are explained in the application note AN2668, in the section titled “Oversampling using white noise”.

White noise generator can be also used in the production of electronic music, usually either directly or as an input for a filter to create other types of noise signal. It is used extensively in audio synthesis, typically to recreate percussive instruments such as cymbals which have high noise content in their frequency domain.

White noise generator can be used for control engineering purposes, it can be used for frequency response testing of amplifiers and electronic filters.

White noise is a common synthetic noise source used for sound masking by a Tinnitus masker.

1.8 Triangular wave generator

1.8.1 Definition

The STM32 DAC provides the user with a triangular waveform generator with a flexible offset, amplitude and frequency.

Theoretically, a triangular waveform is a wave form comprised of an infinite set of odd harmonic sine waves (see *Figure 9*).

The amplitude of the triangular waveform can be fixed using the MAMPx bits in the DAC_CR register.

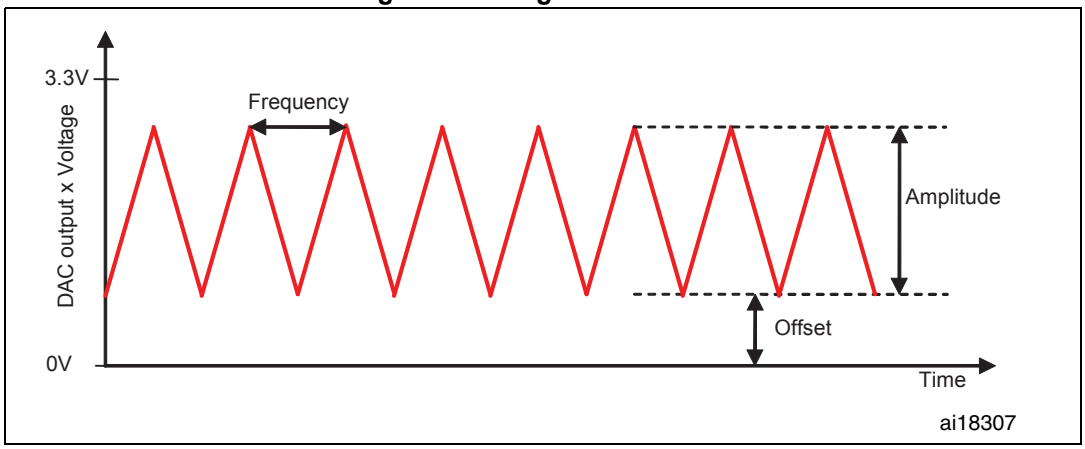
Table 3. Preprogrammable triangular waveform amplitude values

MAMPx[3:0] bits	Digital Amplitude	Analog Amplitude (Volt) (with Vref+ = 3.3V)
0	1	0.0016
1	3	0.0032
2	7	0.0064
3	15	0.0128
4	31	0.0257
5	63	0.0515
6	127	0.1031
7	255	0.2062
8	511	0.4125
9	1023	0.8250
10	2045	1.6483
≥ 11	4095	3.3000

For more details about the triangular waveform, please read the dedicated sections in the reference manuals of the STM32 products.

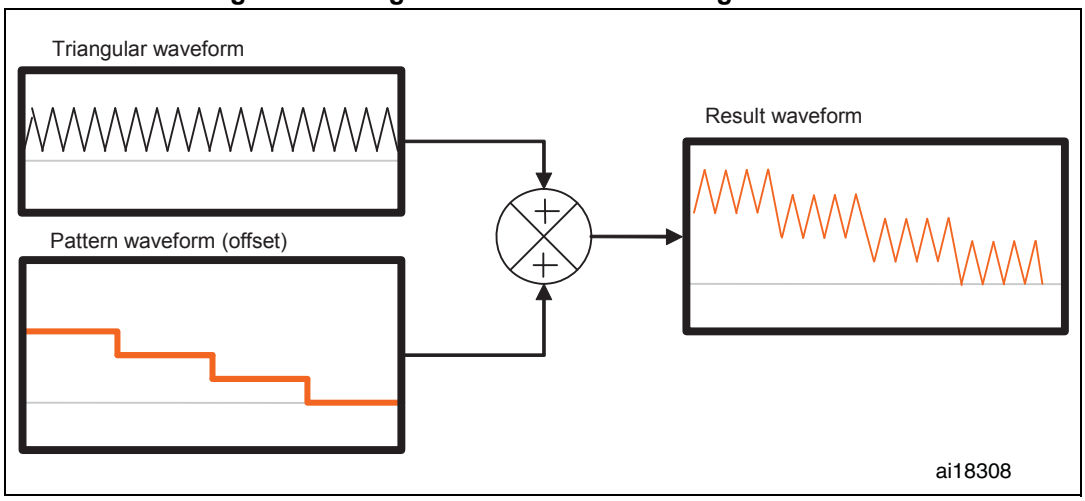
The triangular waveform frequency is related to the frequency of the trigger source.

Figure 8. Triangular waveform



The offset of the triangular waveform is programmable (see *Figure 9*). By varying the offset of the triangular waveform with a preconfigured table of offsets (signal pattern), user can obtain a waveform which corresponds to the sum of the signal pattern and the triangular waveform.

Figure 9. Triangular waveform with changeable offset



1.8.2 Typical applications

Triangular wave generators are often used in sound synthesis as its timbre is less harsh than the square wave because the amplitude of its upper harmonics falls off more rapidly.

Triangular wave generator circuits are also used in many modem circuit applications.

1.9 Buffered output

To drive external loads without using an external operational amplifier, DAC channels have embedded output buffers which can be enabled and disabled depending on the user application.

When the DAC output is not buffered, and there is a load in the user application circuit, the voltage output will be lower than the desired voltage. Enabling the buffer, the voltage output and the voltage desired are similar.

Figure 10. Non buffered channel voltage (with and without load)

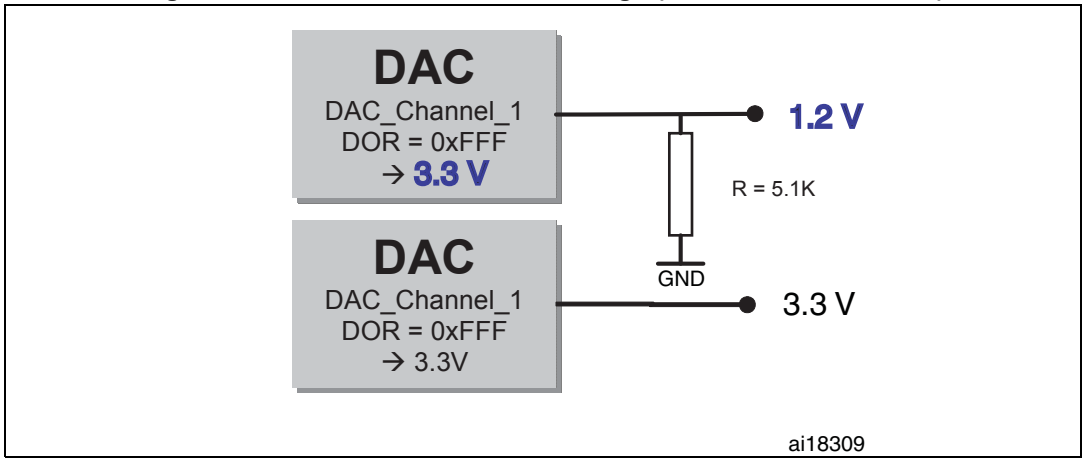
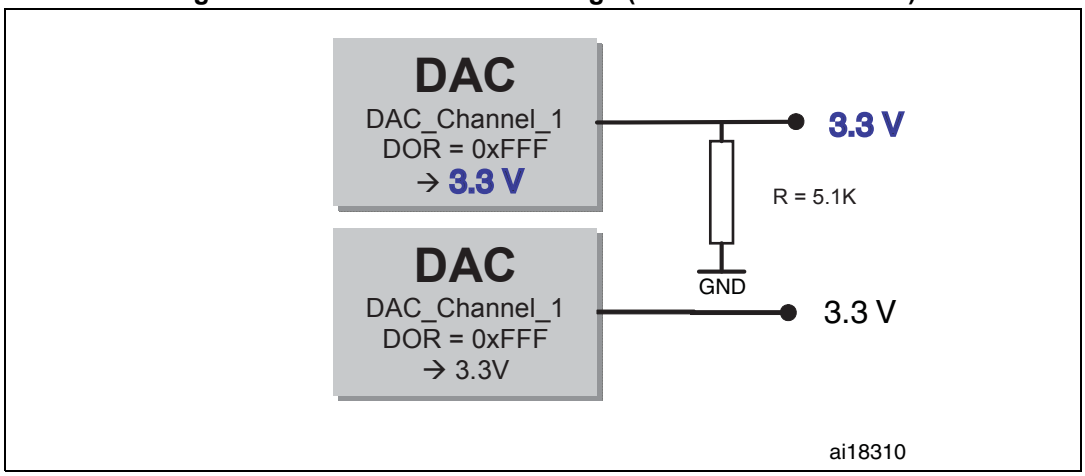


Figure 11. Buffered channel voltage (with and without load)



2 Application examples

2.1 Using the DAC to generate a sine waveform

2.1.1 Description

This example describes step by step how to generate a sine waveform.

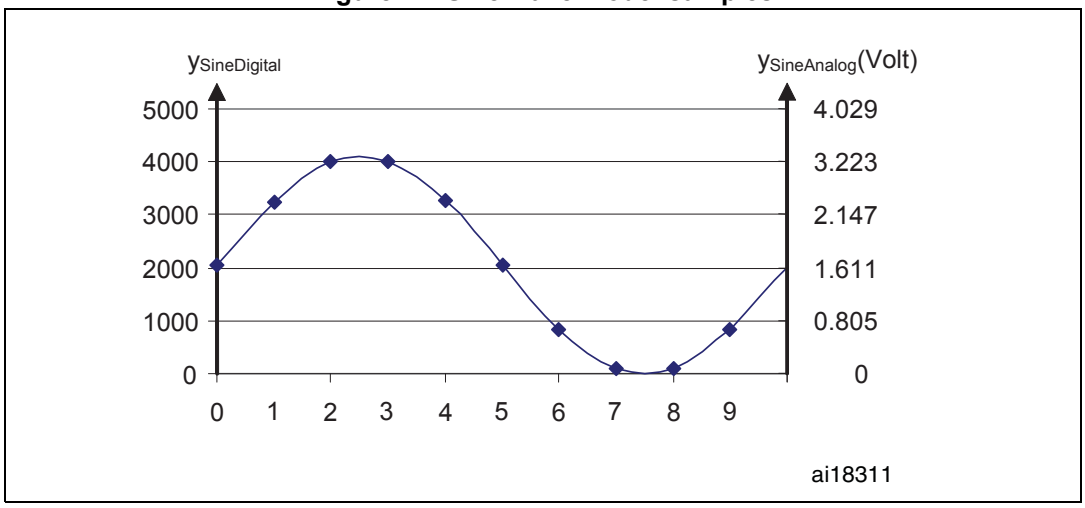
A sine waveform is also called a sine tone with a single frequency, it is known as a pure tone or sinus tone. The sine tones are traditionally used are stimuli in determining the various responses of the auditory system.

2.1.2 Digital Sine waveform pattern preparation

To prepare the digital pattern of the waveform, we have to do some mathematics.

Our objective is to have 10 digital pattern data (samples) of a sine wave form which varies from 0 to 2*PI.

Figure 12. Sine wave model samples



The sampling step is (2*PI)/ n_s (number of samples).

The result value of sin(x) is between -1 and 1, we have to recalibrate it to have a positive sinewave with samples varying between 0 and 0xFFF (which correspond, the range from 0 V to 3.3 V).

$$y_{SineDigital}(x) = \left(\sin\left(x \cdot \frac{2\pi}{n_s}\right) + 1\right)\left(\frac{(0xFFF + 1)}{2}\right)$$

Digital inputs are converted to output voltages on a linear conversion between 0 and V_{REF+}.

The analog output voltages on each DAC channel pin are determined by the following equation:

$$DAC_{Output} = V_{REF}\frac{DOR}{DAC\_MaxDigitalValue}$$

Note: For right-aligned 12-bit resolution: DAC_MaxDigitalValue = 0xFFF

For right-aligned 8-bit resolution: DAC_MaxDigitalValue = 0xFF

So the analog sine waveform $y_{SineAnalog}$ can be determined by the following equation

$$y_{SineAnalog}(x) = 3.3Volt\frac{y_{SineDigital}(x)}{0xFFF + 1}$$

Table 4. Digital and analog sample values of the sine wave

Sample (x)	Digital Sample Value $y_{SineDigital}(x)$	Analog Sample Value (Volt) $y_{SineAnalog}(x)$
0	2048	1.650
1	3251	2.620
2	3995	3.219
3	3996	3.220
4	3253	2.622
5	2051	1.653
6	847	0.682
7	101	0.081
8	98	0.079
9	839	0.676

The table is saved in the memory and transferred by the DMA, the transfer is triggered by the same timer that triggers the DAC

2.1.3 Fixing the sine wave frequency

To fix the frequency of the sinewave signal, you have to set the frequency of the Timer Trigger output.

The frequency of the produced sine wave is

$$f_{Sinewave} = \frac{f_{TimerTRGO}}{n_s}$$

So, if TIMx_TRGO is 1 MHz, the frequency of the DAC sine wave is 10 kHz.

Note: To have a high quality sine wave curve, it is recommended to use a high number of samples n_s (the difference can be appreciated by comparing *Figure 13* with *Figure 14*).

Figure 13. Sine wave generated with n_s = 10

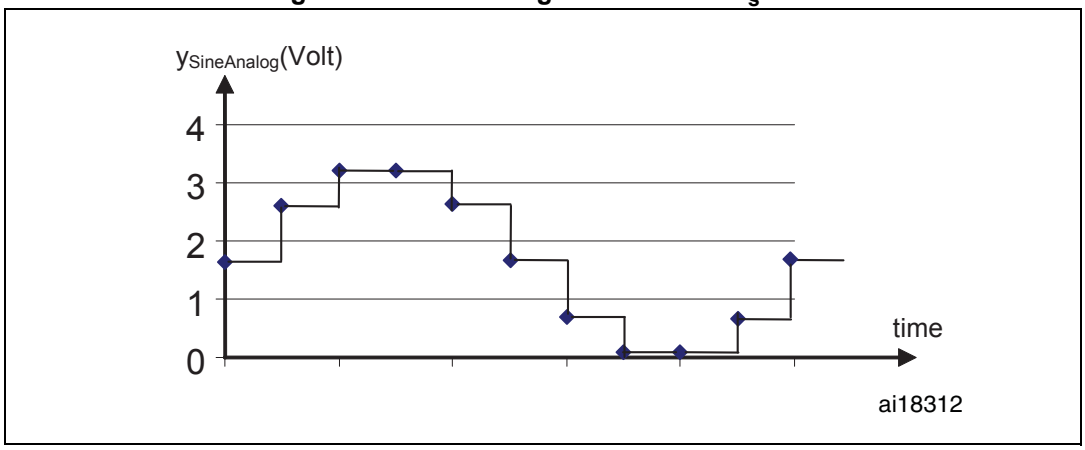
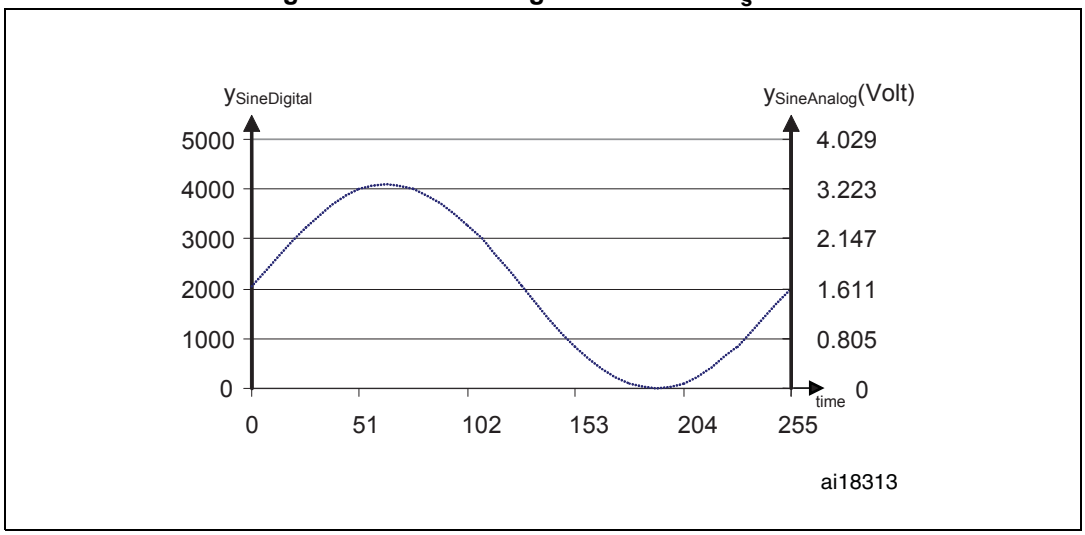


Figure 14. Sine wave generated with n_s = 255

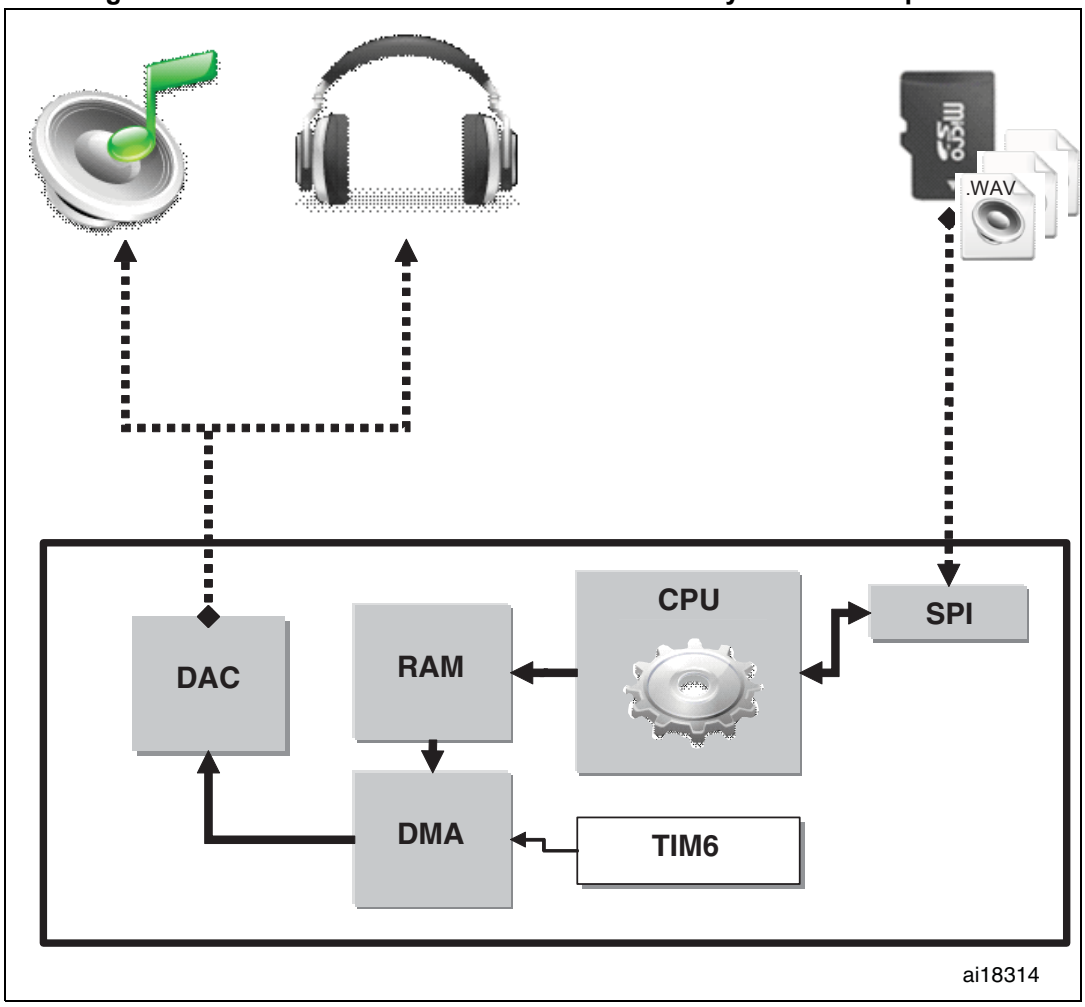


2.2 Using the DAC to implement an audio wave player

2.2.1 Description

The purpose of this application demo is to provide an audio player solution for the STM32 microcontroller for playing .WAV files. The approach is optimized to use a minimum number of external components, and offers the flexibility for end-users to use their own .WAV files. The audio files are provided to the STM32 from a MicroSD memory card.

Figure 15. Flow of data from MicroSD Flash memory to external speakers



The audio wave player demonstration described in this section is a part of the STM32100B-EVAL demonstration firmware. You can download this firmware and the associated user manual (UM0891) from the STMicroelectronics website *www.st.com*.

2.2.2 Audio wave file specifications

This application assumes that the .WAV file to be played has the following format:

- Audio Format: PCM (an uncompressed wave data format in which each value represents the amplitude of the signal at the time of sampling)
- Sample rate: may be 8000, 11025, 22050 or 44100 Hz
- Bits Per Sample: 8-bit (Audio sample data values are in the range [0-255])
- Number of Channels: 1 (Mono)

2.2.3 .WAV file format

The .WAV file format is a subset of the Resource Interchange File Format (RIFF) specification used for the storage of multimedia files. A RIFF file starts with a file header followed by a sequence of data chunks. A .WAV file is often just a RIFF file with a single "WAVE" chunk consisting of two sub-chunks:

1. a **fmt** chunk specifying the data format
2. a **data** chunk containing the actual sample data.

The WAVE file format starts with the RIFF header: it indicates the file length.

Next, the fmt chunk describes the sample format, it contains information about: Format of the wave audio : (PCM/...), Number of channels (mono/stereo), sample rate (number of samples per seconds : e.g., 22050), and the sample Data size (e.g. 8bit/16bit). Finally, the data chunk contains the sample data.

2.3 Audio wave player implementation

The Audio wave player application is based on the SPI, DMA, TIM6, and DAC peripherals.

At start up, the application first uses the SPI to interface with the MicroSD card and parses its content, using the DOSFS file system, looking for available .wav files in the USER folder. Once a valid .wav file is found, it is read back though the SPI, and the data are transferred using the CPU to a buffer array located in the RAM. The DMA is used to transfer data from RAM to DAC peripheral. TIM6 is used to trigger the DAC which will convert the Audio digital data to an analog waveform.

Before the audio data can be played, the header of the WAV file is parsed so that the sampling rate of the data and its length can be determined.

The task of reproducing audio is achieved by using sampled data (data contained in the .WAV file) to update the value of the DAC output, this data is coded in 8 bits (with values from 0 to 255),

The DAC Channel 1 is triggered by TIM6 at regular interval specified by the sample rate of the .WAV file header.

The .WAV files are read from the MicroSD Memory using a DosFS file system.

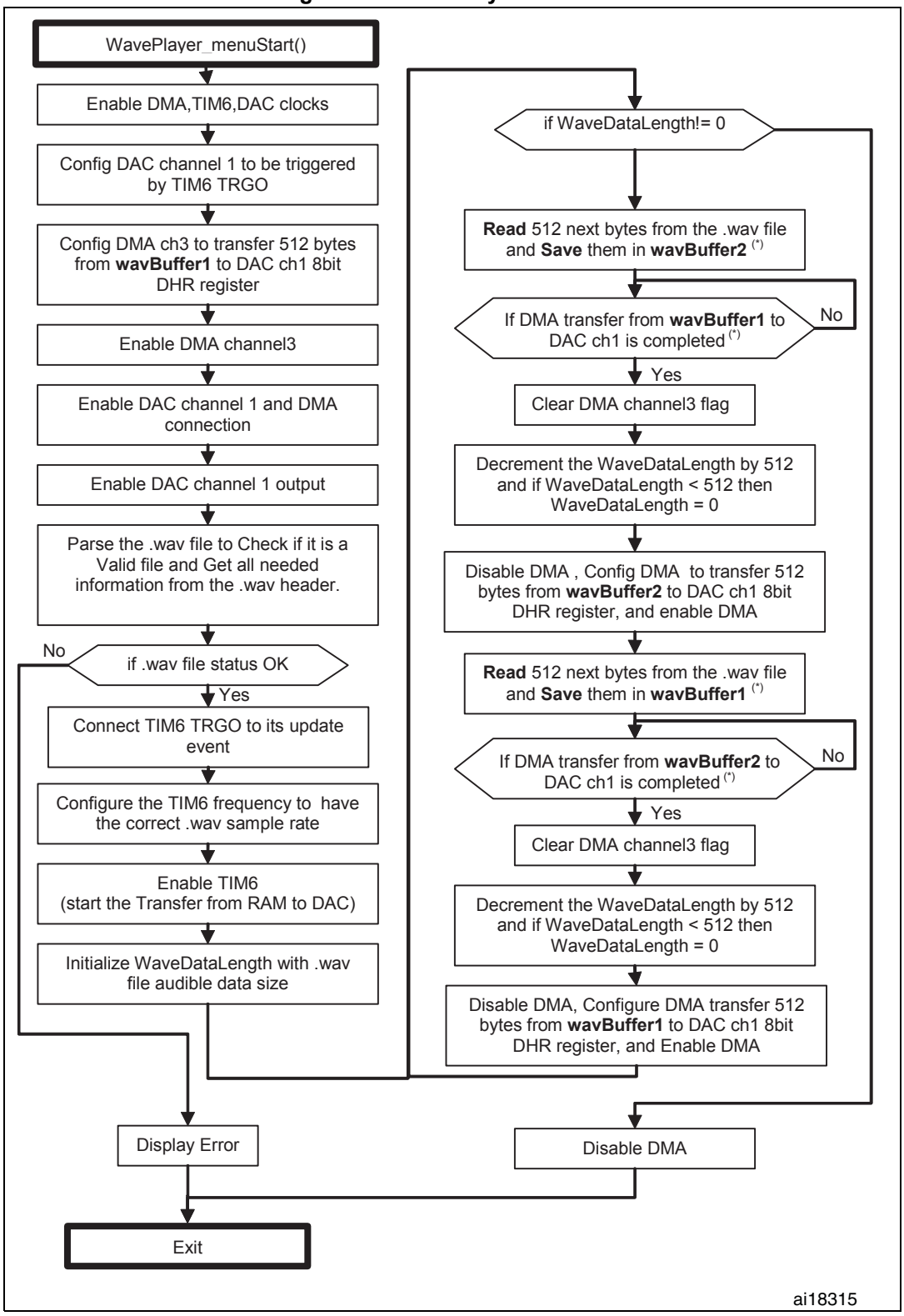
In the Demo code, code files handling the waveplayer demo are:

waveplayer.c

waveplayer.h

The wave player demo is called using `WavePlayerMenu_Start()` function which has the flowchart shown in *Figure 16*.

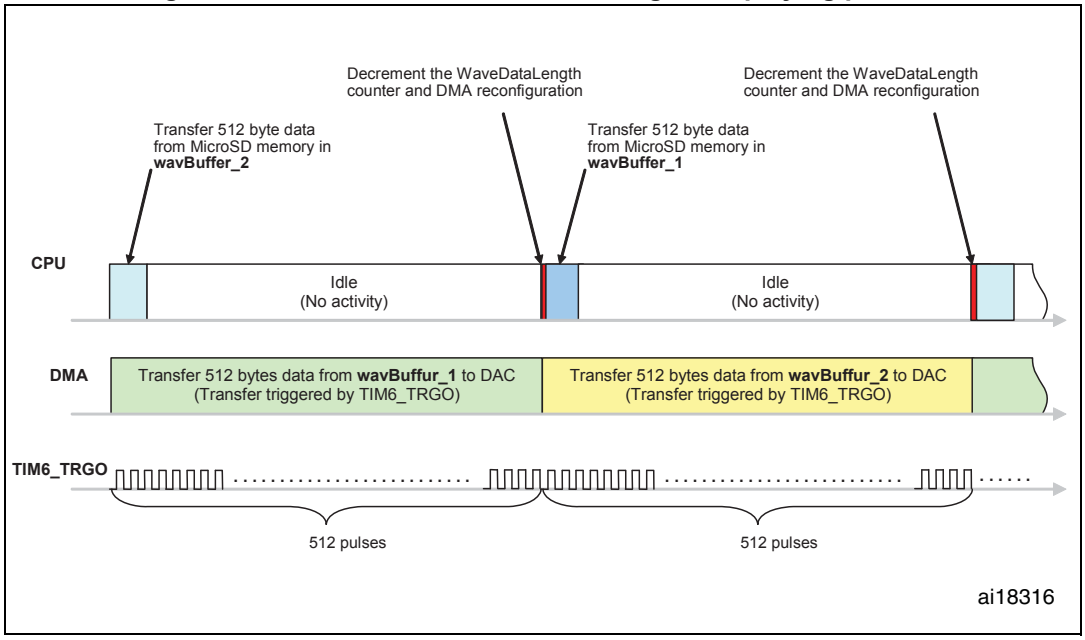
Figure 16. Wave Player flowchart



(*) when DMA is transferring data from one RAM buffer, CPU is transferring data from the MicroSD Flash memory to the other RAM buffer.

In this application, coprocessing is mandatory to permit a simultaneous Wave read (from the external memory source) and write (in the DAC register).

Figure 17. CPU and DMA activities during wave playing process



3 Conclusion

This application note and in particular the examples given in*Section 2* have been provided to help you get familiar with the DAC's main features.

The first example (in *Section 2.1*) shows how to generate an analog waveform, using the example of a sine waveform. The second example (in *Section 2.2*) offers a straightforward and flexible solution for using the STM32, to play .WAV files, stored in an SPI MicroSD Flash memory.

You can use these examples as starting points for developing your own solution using STM32 microcontrollers.

4 Revision history

Table 5. Document revision history

Date	Revision	Changes
28-May-2010	1	Initial release.
16-Apr-2015	2	Updated *Introduction*, *Section 1.3: Dual channel mode* and *Section 3: Conclusion*. Updated formulas in *Section 2.1.2: Digital Sine waveform pattern preparation*. Updated *Figure 2: STM32F100x DAC trigger channels* and *Figure 5: Pseudo random code generator embedded in the DAC*. Added *Table 1: Applicable products* and *Table 2: DAC configurations for STM32 microcontrollers*. Added *Section 1.1: Overview*, and Note in *Section 1.3: Dual channel mode*.