# AN4711
# Application note

## USB audio bridge example with STM32F0 MCUs

## Introduction

This application note describes a method and an example of synchronizing audio playback or audio recording with an upstream or downstream USB audio host, ensuring flawless audio listening or recording using only internal MCU resources.

Focusing on specific properties of USB microcontrollers from the STM32F0 family, the application note describes how the CRS unit can be beneficially employed for USB audio streaming synchronization. In particular, it elaborates a method of HSI48 clock frequency trimming to compensate for timing differences due to independent USB host (computer) and device (STM32F0) clock domains.

The documents listed below should be referred to for complementary information.

- From *www.st.com*:
  - UM1717, 'STM32F0x2xx USB Full Speed Device Library' *[1]*
  - UM1025, 'Getting started with STM-STUDIO' *[2]*
  - RM0091, 'STM32F0x1/STM32F0x2/STM32F0x8 Reference Manual' *[3]*
- From www.usb.org, 'Universal Serial Bus Specification revision 2.0' *[4]* and related documents.

*Table 1* lists the applicable STM32F0 microcontroller part numbers.

**Table 1. Applicable products**

| Type | Part number |
|---|---|
| Microcontroller | STM32F042C4, STM32F042C6, STM32F042T4, STM32F042T6, STM32F042K4, STM32F042K6, STM32F042G4, STM32F042G6, STM32F042F4, STM32F042F6, STM32F072C8, STM32F072CB, STM32F072R8, STM32F072RB, STM32F072V8, STM32F072VB, STM32F048C6, STM32F048G6, STM32F048T6, STM32F078CB, STM32F078RB,STM32F078VB. |

# Contents

# List of figures

# List of tables

# 1 Streaming audio via USB

This chapter describes the USB Audio class and the challenges encountered in USB audio streaming.

## 1.1 USB Audio class

The Audio class is one of the USB device classes defined within the USB specification. It concerns transfer and control of audio data streams between a USB host and a USB device. Three interfaces are available within the Audio class: audio control (AC) interface, zero or more audio streaming (AS) interfaces and zero or more MIDI streaming (MS) interfaces.

The Isochronous transfer type is used to transfer audio data in the USB Audio class. An isochronous endpoint has a specific bandwidth allocated by the USB host that regularly fetches data to and/or from that endpoint. For example, a USB host can send 512 bytes of data to a USB device in each 1 ms time slot (frame in USB FS mode). Although isochronous transfers support error detection through CRC, no retry is supported to repair erroneously transmitted data.

There are two mutually incompatible versions of the USB Audio class:

- USB Audio Class 1.0 defined in 'Universal Serial Bus Device Class Definition for Audio Devices Release 1.0 March 18, 1998'
- USB Audio Class 2.0 defined in 'Universal Serial Bus Device Class Definition for Audio Devices Release 2.0 May 31, 2006'.

The content of this application note applies to the Audio class specification 1.0, natively supported by the current versions of MS Windows (Vista, 7, 8, 10).

## 1.2 Challenges for audio streaming

The basic challenge for audio streaming (real-time serial transfer and playback of digital audio data) is to ensure that the sound be exempt of audible artifacts such as interruptions, dropouts or changes in sound pitch. The difficulty arises from the fact that the transmitting and receiving devices do not usually use common clock domains.

To address this challenge, the audio-source data rate (in this case that of the USB host) must match that of the playback end. As a certain deviation (0.25%) from the nominal data transfer rate is tolerated in the USB specification, the device receiving an audio stream cannot predict its precise data rate. Instead, it must be able to adapt to the audio stream data rate as well as to potential fluctuations within the specification tolerances. This implies ensuring that the amount of audio data retrieved from the USB device endpoint matches the quantity of audio data output for playback, with the playback sample rate remaining as constant as possible.

It is mandatory to synchronize USB device and USB host clock domains. The USB specification determines three clock synchronization models - asynchronous, synchronous and adaptive.

For more details of these models, refer to *section 5.12: Special Considerations for Isochronous Transfers* in the USB Specification 2.0 *[4]*.

# 2 MCU application description

This chapter describes the synchronization method, resources used, and a software algorithm fulfilling the synchronization task for the method described. A way of tuning the algorithm with help of STM Studio is also detailed.

## 2.1 System description

The method described for synchronizing the audio playback sample rate with the USB host data transmission rate makes exclusive use of resources within the STM32F0. No specific additional components are required, while maintaining the aim of flawless audio playback .

A RAM buffer is set between the USB device peripheral of the STM32F0 and a DAC peripheral, providing audio playback through an external amplifier and speakers. It is implemented as a 1.5 Kbyte (8 times 192 bytes) rolling buffer with write and read pointer. Both pointers always move in the same direction and roll back to the buffer start on overflow. All hardware blocks and software involved in the streaming act so as to maintain the two pointers as far apart from each other as possible. Hence their target 'distance' is equal to half the buffer size. This is to prevent audible artifacts occuring on buffer overflow or underflow, that is, when the write pointer meets the read pointer, or the inverse case.

The streaming is set to stereo mode (two audio channels), 16-bit sample depth and 48 kHz sampling frequency. This results, nominally, in 192 Kbytes of audio data to be streamed per second. Due to the tolerance of the USB speed specification, the real figure can be anywhere within +/- 0.05% of nominal. With USB FS mode using a 1 ms frame interval, this translates to 192 bytes of audio data to be sent in each USB frame.

On receipt of frame data by the STM32F0 USB device endpoint peripheral , that is, about once per millisecond, the software copies the 192-byte chunk of audio data from the USB device endpoint to the write pointer location of the RAM buffer, and adjusts the write pointer accordingly.

The audio data in the RAM buffer are transferred by the DMA controller to a digital-to-analog converter (further DAC) at the sampling rate (48 kHz) originating from the TIM6 timer clocked by the HSI48 oscillator.

To ensure an adequate sampling rate, the HSI48 frequency is trimmed in a closed loop, such that it tracks the real data rate of the USB host. As the audio sampling frequency is proportional to the HSI48 frequency, any change of TRIM value (a typical trimming step size is 0.14%) results in output-audio pitch change. While such a change in audio pitch is hard to perceive if it is gradual over an extended period of time, any abrupt change produces an audible artifact. The higher and more frequent the sampling frequency variation, the greater the audio distortion. Thus the variation of the TRIM value must be slow, smooth and minimized in amplitude.

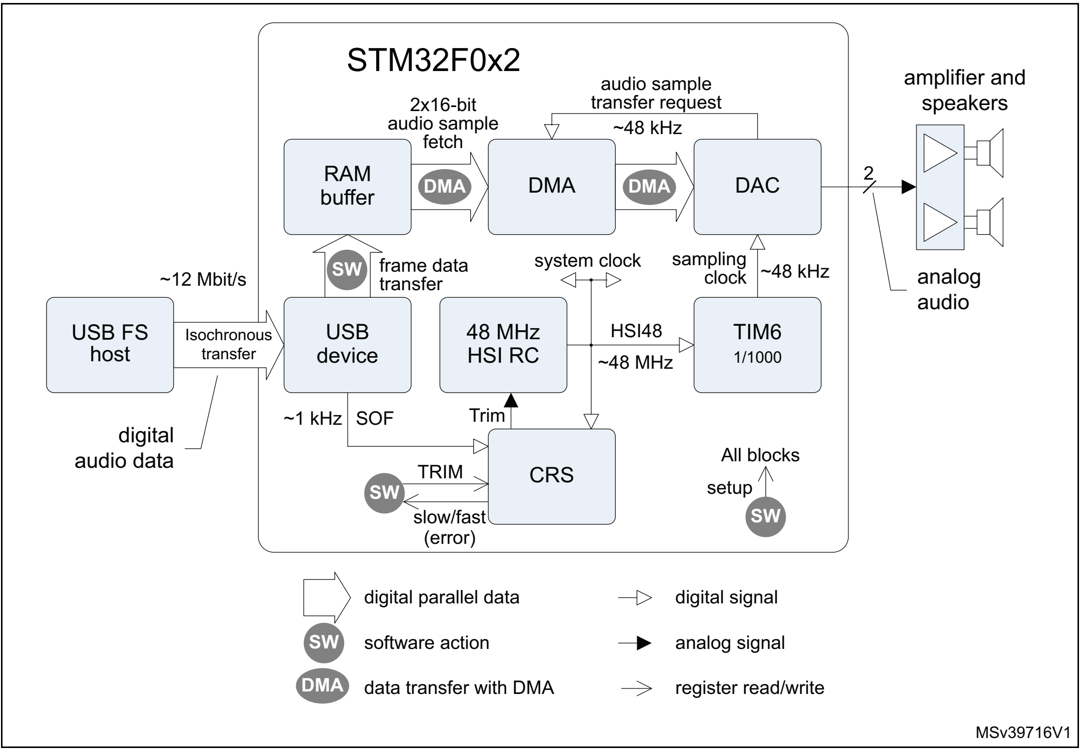
The signal trimming HSI48 frequency is proportional to the value in the TRIM field of the CRS control register within the clock recovery system controller (CRS) described in *Section 2.2.6*. Assessing the USB frame duration while using the HSI48 as the clock source, the CRS indicates the amount of HSI48 deviation from the ideal speed with respect to the actual USB data rate.

The TRIM value is regularly computed and updated by a software algorithm, based on the measurement result from the CRS, based on the ‘distance’ of the RAM buffer pointers and on the evolution of that distance. The algorithm is executed in the CRS interrupt service routine.

## 2.2 Resources

The hardware and software used is shown in *Figure 1*. Arrow directions indicate data and control flows in the configuration used for the streaming method described, rather than the full physical possibilities of the microcontroller.

**Figure 1. Resources used in USB audio streaming and playback**



### 2.2.1 RAM buffer

In the example described in this document, one USB data frame carries 192 bytes of audio data, which corresponds to 48 samples of a 16-bit stereo audio stream. As a trade-off between the standard needs of resource saving and operating robustness, the size of the RAM buffer is set to 8 blocks of 192 bytes, that is, 1536 bytes in total. The buffer is constituted as a set of contiguous memory blocks B1 through Bn, each having individually programmable size and RAM location, as shown in *Figure 2*.

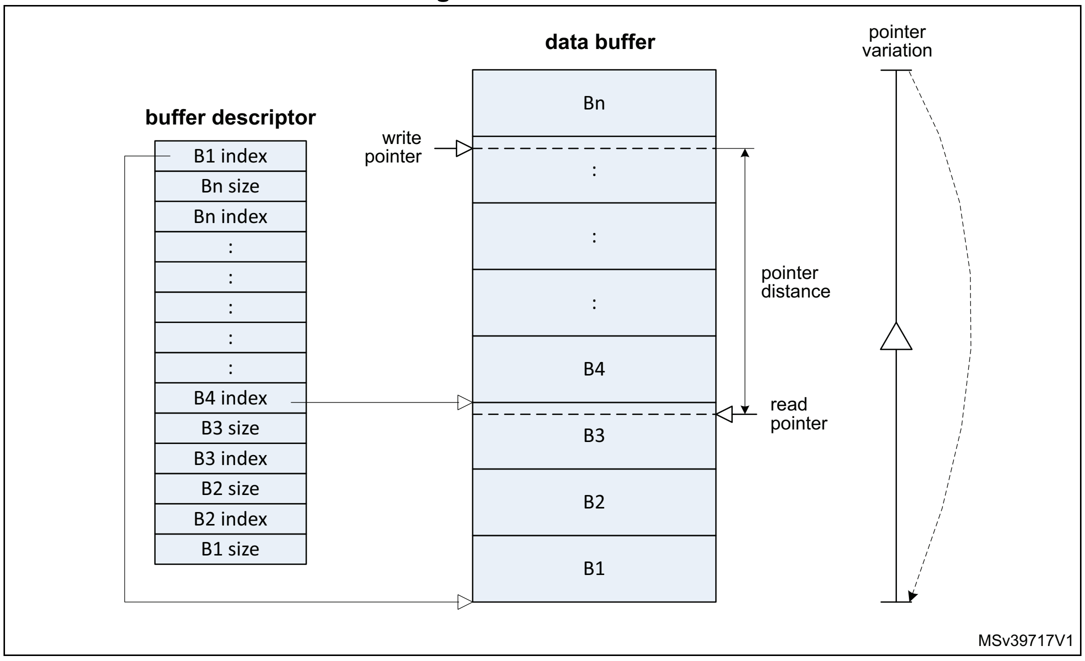
The buffer is rolling, that is, data from USB frame N takes the place of the data in USB frame N-7. Feeding of the buffer with USB frame audio data is handled by software, in the USB interrupt service routine.

Audio sample data are transferred from the RAM buffer to the DAC by the DMA controller. This is programmed by software to move to the next buffer block when the data of the current block have been read out, and to roll back to block 1 start after finishing block 8.

The data received from the Host are 16 bits long, while the DAC can only process 12-bit values. For this reason 12-bit left alignment is used, hence the 4 least-significant bytes are omitted.

Information on pointers can be found in *Section 2.3*.

**Figure 2. RAM buffer**



### 2.2.2 DMA controller

The direct memory access controller hardware block copies stereo audio samples (four data bytes) from the RAM buffer to the DAC on every tick of the 48 kHz sampling clock. The DMA controller generates an interrupt each time it arrives at the end of a RAM buffer block, that is, when it finishes transferring 192 bytes (48 audio samples) of the block to the DAC. This happens approximately once a millisecond.

In the DMA interrupt service routine, the software uses the buffer descriptor data to reprogram the DMA controller. The DMA pointer is set to next buffer block’s start address using the block index data. Block size data is used by the DMA controller to determine the number of bytes to be transfered before generating next interrupt. In this example, all eight blocks have uniform size of 192 bytes, corresponding to 48 stereo 16-bit audio samples(2 x 12 bits due to DAC limits as previously explained).

### 2.2.3 Timer

The TIM6 16-bit timer is used to divide the HSI48 48 MHz clock by 1000 to generate a 48 kHz audio sampling frequency. Any variation of the HSI48 clock directly translates into an equivalent relative variation of the 48 kHz sampling frequency.

### 2.2.4 DAC

Of the four data bytes received from the DMA controller on every tick of the 48 kHz sampling clock, the DAC is programmed to convert two data bytes for the left-channel and two for the right-channel analog audio output. The DAC is 'sampled' with the 48 kHz sampling clock from TIM6.

### 2.2.5 HSI RC 48 MHz oscillator

The high-speed internal 48 MHz RC oscillator (HSI48) is per-part calibrated on the production line to a free-running frequency of 48 MHz ±3% at 25 °C, and with the middle value of the TRIM field programmed in the CRS control register. In an application, its operating frequency depends on the TRIM value and on temperature. In the method and example described in this document, HSI48 is also used as the system clock.

### 2.2.6 CRS

The clock recovery system (CRS) is a hardware unit within the MCU, clocked by the HSI48 oscillator and with fine-frequency control through the TRIM control field. The TRIM value can be set autonomously by the CRS or 'manually' by software. For the method of operation and algorithm described, it is vital that the HSI48 be used as the clock reference for timing the output sampling process, that is, the digital-to-analog conversion events and DMA read accesses from the RAM buffer.

The CRS is specifically designed to synchronize the HSI48 clock to a timing reference such as, for example, an external clock or a USB host. The latter option is selected for the application example described in this document. The regular start-of-frame (SOF) packets received at a rate of one per millisecond (±0.05%) are used as synchronization events.

The CRS contains a 16-bit down/up counter, counting HSI48 ticks. At each synchronization event, it starts counting down from a predefined value and reverts to up-counting upon reaching zero. A subsequent synchronization event occurrence during the down-counting phase indicates that the HSI48 clock is slow, and its momentary value (error) indicates the amount of lag.

If the subsequent synchronization event occurs in the up-counting phase of the counter, the HSI48 clock is considered fast and its momentary value (error) indicates the amount of advance. The counting direction represents the error sign. Meeting the synchronization event when the counter reaches zero (zero error) means that the HSI48 is precisely synchronized to the reference signal.

The HSI48 can speed up or slow down, as a consequence of trimming by the CRS. In this way, the CRS regulates the closed loop so as to minimize the error at each new synchronization event. The refresh value written to the TRIM bits after each synchronization event should be chosen such that the system works within correct USB frequency limits, while maintaining the distance between buffers.

Although an auto-trimming function is available in the CRS, it is not activated in the application example described. The use of auto-trimming would minimize the error of the HSI48 clock but it would not allow control of the pointer distance, which would eventually lead to audible artifacts. See *Figure 5: Automatic trimming*. Instead, the software algorithm described in *Section 2.4* controls the TRIM value. It uses the synchronization error information from the CRS, the distance between RAM buffer write and read pointers, and the evolution of this distance. Synchronization error and warning CRS interrupts are enabled.

The CRS is configured to generate an interrupt at every arrival of the SOF synchronization signal from the USB endpoint. The software algorithm ensuring flawless audio playback is executed in the CRS interrupt service routine.

Further details of the CRS can be found in the reference manual RM0091 *[3]*.

## 2.3 Pointers and pointer distance

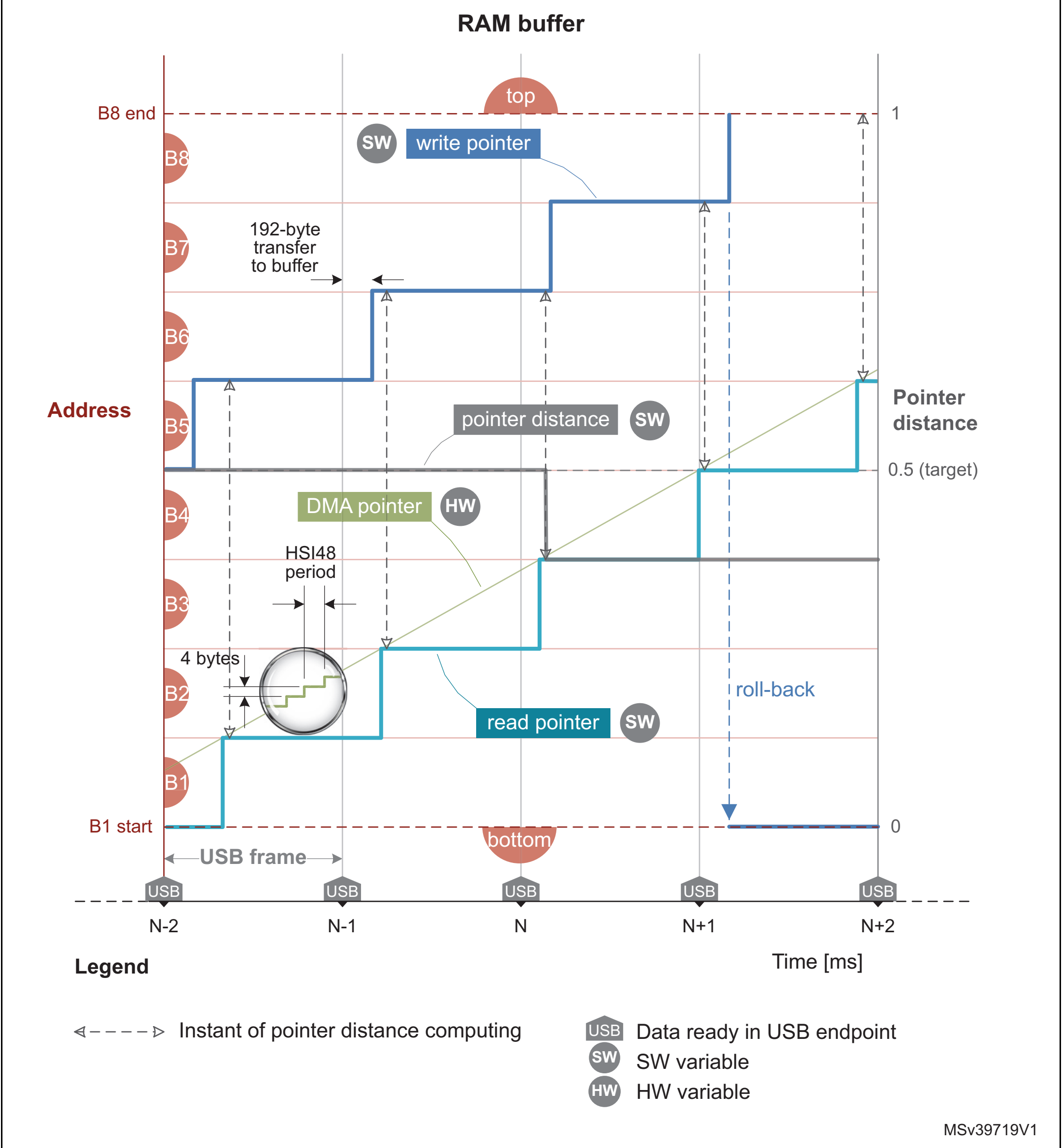
The write pointer value is set in the USB interrupt service routine after the transfer of each 192-byte audio data chunk from the USB endpoint to the RAM buffer, and remains unchanged until the following USB interrupt, which occurs about a millisecond later.

The read pointer variable value is set at the beginning of the DMA interrupt service routine and is equal to the following RAM buffer block start address index. It remains unchanged until the following DMA interrupt, which occurs about a millisecond later.

As a consequence, the resulting pointer distance variable value computed by the CRS interrupt service routine varies in turn by multiples of 192.

*Figure 3* shows four USB frame periods. To illustrate the pointer-distance computation mechanism, the synchronization shown is intentionally non-ideal in order to magnify one pointer distance change. Time periods representing the variation of read pointer, DMA write pointer and write pointer are shown to aid understanding of the event chronology.

**Figure 3. Variation of pointers and of pointer distance**



## 2.4 Synchronization algorithm

This section covers the algorithm that fulfills the synchronization task according to the method described in *Section 2.1*. The flow chart is first described in detail, followed by a description of algorithm tuning using STM Studio software.

### 2.4.1 Algorithm description

The simplified flowchart in *Figure 4* illustrates the algorithm, which is executed in the CRS interrupt service routine.

*Table 2* explains the terms used in the flowchart.

**Table 2. Terms used in the synchronization algorithm flowchart**

| Term | Description |
|---|---|
| HSI48 clock too fast | HSI48 clock is outside a defined USB range and faster than target |
| HSI48 clock too slow | HSI48 clock is outside a defined USB range and slower than target |
| HSI48 fast | HSI48 clock is within a defined range, yet faster than target |
| HSI48 slow | HSI48 clock is within a defined range, yet slower than target |
| TRIM set up | Increase TRIM value to make it correspond to target HSI48 frequency |
| TRIM up | Increment TRIM value (if not at maximum) to speed HSI48 clock up |
| TRIM set down | Decrease TRIM value to make it correspond to target HSI48 frequency |
| TRIM down | Decrement TRIM value (if not at minimum) to slow HSI48 clock down |
| Dead time | Period during which changes to TRIM are forbidden, to avoid frequent changes |

The algorithm first evaluates the HSI48 clock speed by computing its error with respect to the target speed, expressed in number of trimming steps. If the error exceeds defined limits, the TRIM value is adjusted and the service routine quits. In this way, the HSI48 clock speed converges to a value within a defined USB range after a number of interrupt service cycles.

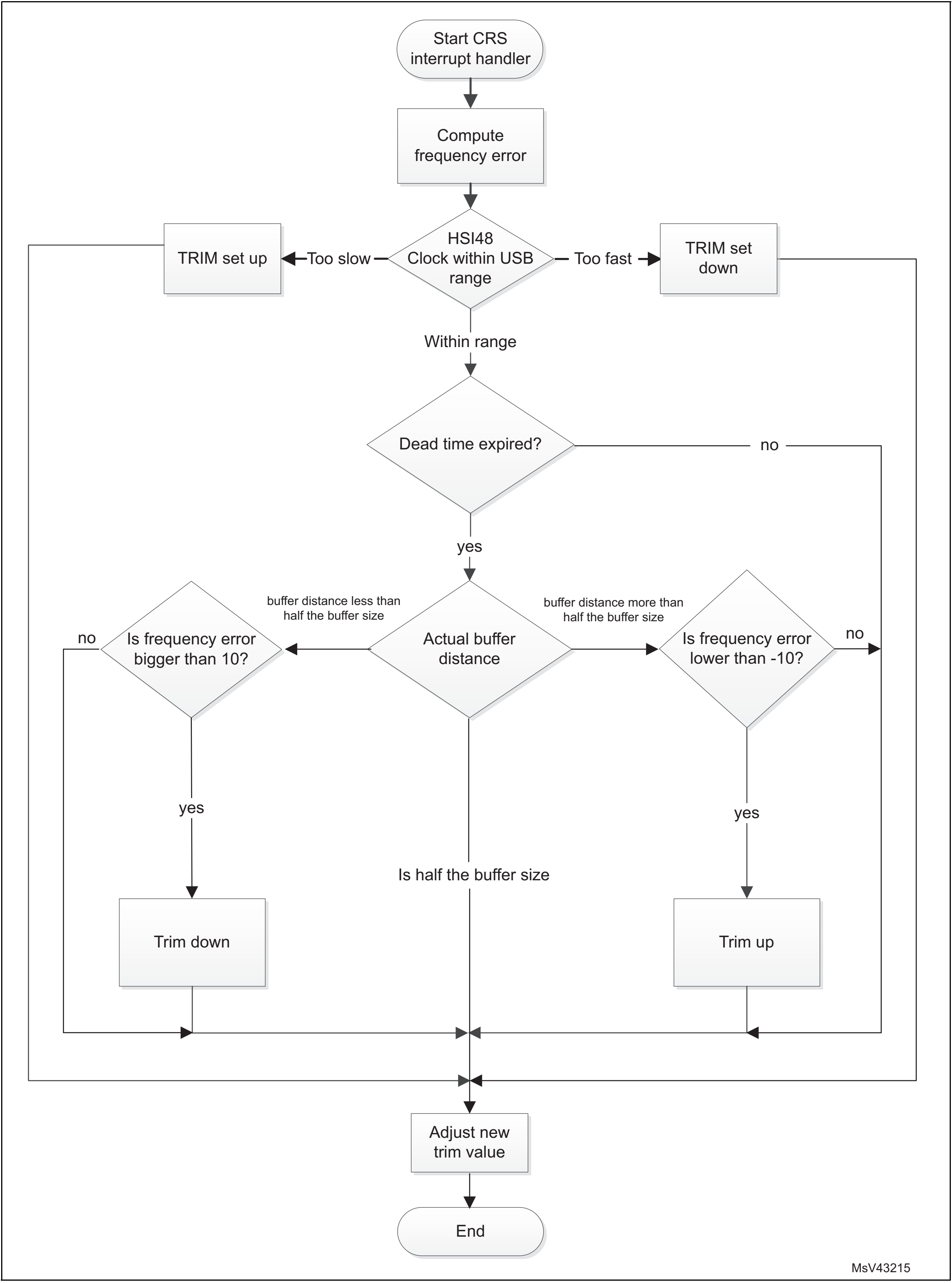
If the HSI48 clock speed error is within a defined range and the call counter marks the end of the dead time period, the algorithm evaluates the distance between the buffer pointers and the current frequency error. Based on this analysis, it adjusts or keeps the TRIM value. If the algorithm changes the TRIM value, it reloads the call counter with a value corresponding to the dead time.

The frequency error provided by the CRS is used for setting the trimming value. To avoid the possibly of frequent changes of the TRIM value when the frequency error is close to zero, a non-responsive zone of +/- 10 is defined.

As the algorithm acts to converge to a pointer distance of half the buffer size, any temperature variation of the HSI48 free-running frequency is also compensated, provided that the TRIM field value is within the allowed range.

Once the algorithm converges, the TRIM value does not remain steady over time, even when the die temperature has stabilized. This is because the accuracy of the HSI48 frequency synchronization to the USB reference is not absolute. It is limited by the TRIM value minimum step, which by definition cannot be infinitely small. Hence the algorithm tends to toggle the TRIM value between two neighboring values.

**Figure 4. Software algorithm flow chart**


Start CRS interrupt handler
Compute frequency error
HSI48 Clock within USB range
Too slow
TRIM set up
Too fast
TRIM set down
Within range
Dead time expired?
no
yes
Actual buffer distance
buffer distance less than half the buffer size
buffer distance more than half the buffer size
Is frequency error bigger than 10?
no
yes
Trim down
Is frequency error lower than -10?
no
yes
Trim up
Is half the buffer size
Adjust new trim value
End
MsV43215

### 2.4.2 Algorithm tuning in STM Studio

STM Studio PC software is a run-time variable monitoring and visualization tool for STM32 microcontrollers. It is available from *www.st.com*.

STM Studio helps the designer to optimize the algorithm operation, by tuning the dead time period through the down-counter restart value.

For USB audio streaming to the STM32072B-EVAL board, one of the computer USB ports is cabled to a CN4 USB receptacle on the board. To use STM Studio in parallel, another computer USB port is cabled to the ST-Link/V2-1 USB receptacle on the STM32072B-EVAL board.

The ST-Link/V2-1 facility on the STM32072B-EVAL board mediates the communication between STM Studio and the target STM32F0 microcontroller. In this way, STM Studio monitors selected variables in the STM32F0. In the following figures, the variables monitored and plotted are:

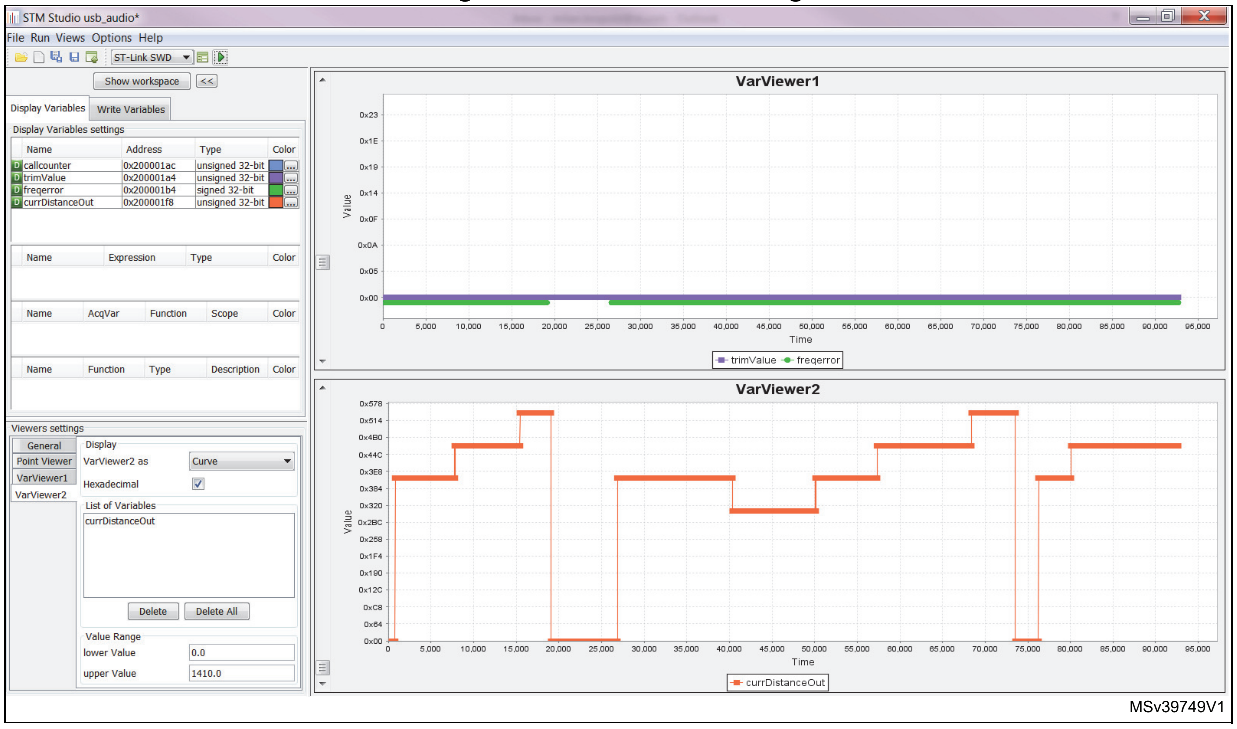
- RAM buffer pointer distance, called ‘currDistanceOut’, plotted in orange
- TRIM field value, called ‘trimValue’, plotted in violet
- error in trimming steps, called ‘freqerror’, plotted in green.

The time scale is in seconds and is common to the upper (VarViewer1) and the lower (VarViewer2) plot.

*Note: Because of the sampling freqeuncy of STStudio, some value changes of short duration might not be caught.*

*Figure 5* shows the impact of the automatic trimming mode. By controlling the TRIM field, the CRS keeps the HSI48 frequency at a value within a maximum of half a trimming step of the target frequency. As the CRS trimming step is approximately 67 kHz, which is the allowed range of the HSI48 frequency, the TRIM field ideally varies by +/- 1, which corresponds to the least-significant bit. The ‘freqerror’ variable at zero shows that the system has reached convergence. The remaining deviation from the target frequency is between zero and approximately 67 kHz. This deviation causes the pointer distance to diverge from  the target distance and eventually leads to pointer overlap, which results in audio artifacts.

**Figure 5. Automatic trimming**


STM Studio usb_audio*

Display Variables settings

| Name | Address | Type | Color |
|---|---|---|---|
| callcounter | 0x200001ac | unsigned 32-bit | 🟦 |
| trimValue | 0x200001a4 | unsigned 32-bit | 🟪 |
| freqerror | 0x200001b4 | signed 32-bit | 🟩 |
| currDistanceOut | 0x200001f8 | unsigned 32-bit | 🟥 |

Viewers settings: VarViewer2

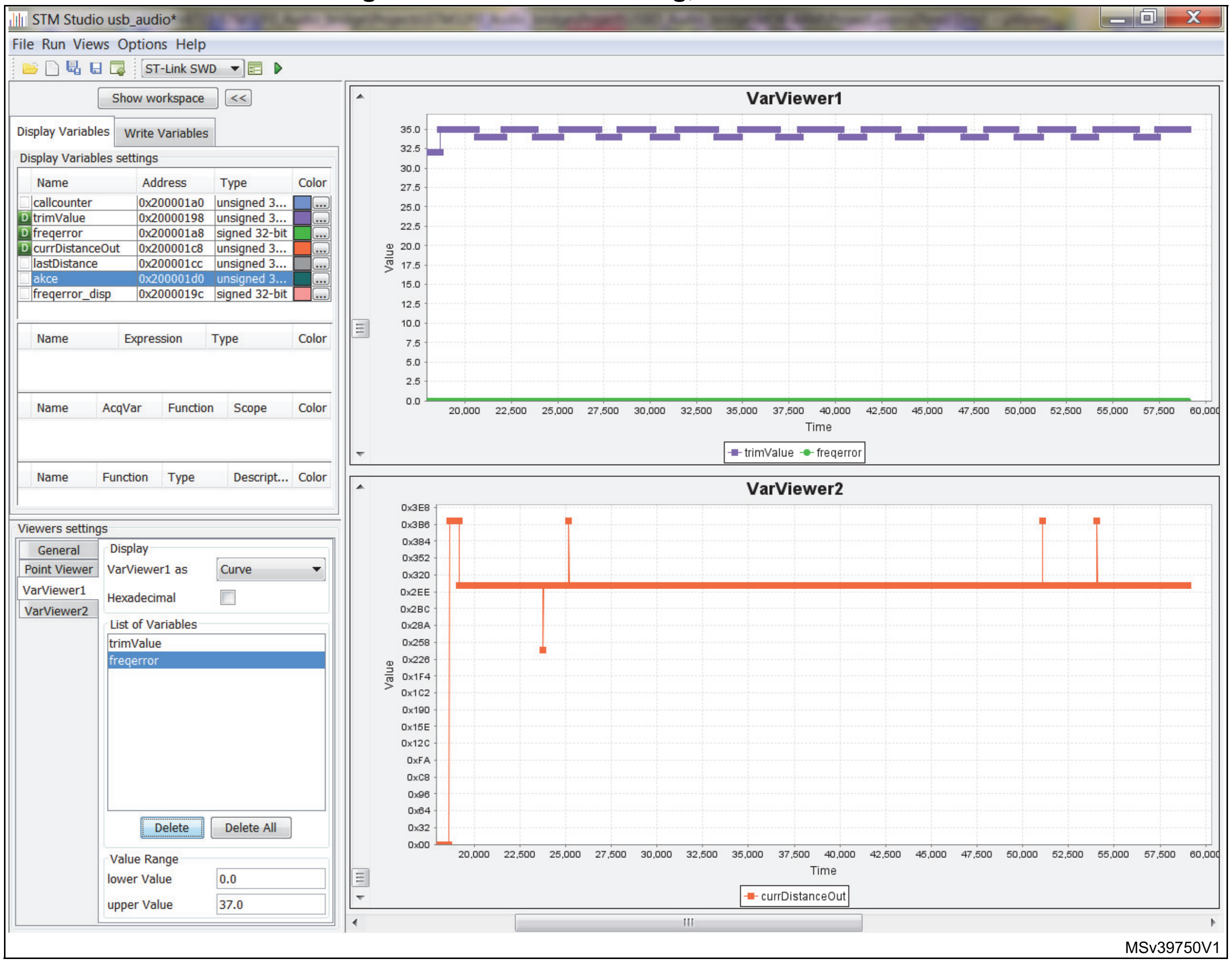
Display
VarViewer2 as: Curve
Hexadecimal: ✓

List of Variables
currDistanceOut

Value Range
lower Value: 0.0
upper Value: 1410.0

*Figure 6* shows the behavior of the algorithm with a dead time of 5 ms, and both stabilized and varying temperature. Depending on the choice of dead time, the algorithm can minimize the risk of audio artifacts at the expense of audio distortion, minimize audio distortion at the expense of increased risk of audio artifacts, or set a trade-off with regard to either aspect.

**Figure 6. Software trimming, 5 ms dead time**



Display Variables settings

| | Name | Address | Type | Color |
|---|---|---|---|---|
| | callcounter | 0x200001a0 | unsigned 3... | 🟦 |
| D | trimValue | 0x20000198 | unsigned 3... | 🟪 |
| D | freqerror | 0x200001a8 | signed 32-bit | 🟩 |
| D | currDistanceOut | 0x200001c8 | unsigned 3... | 🟧 |
| | lastDistance | 0x200001cc | unsigned 3... | ⬜ |
| | akce | 0x200001d0 | unsigned 3... | 🟩 |
| | freqerror_disp | 0x2000019c | signed 32-bit | 🟥 |

Viewers settings

General / Point Viewer / VarViewer1 / VarViewer2

Display: VarViewer1 as: Curve

Hexadecimal: ☐

List of Variables: trimValue, freqerror

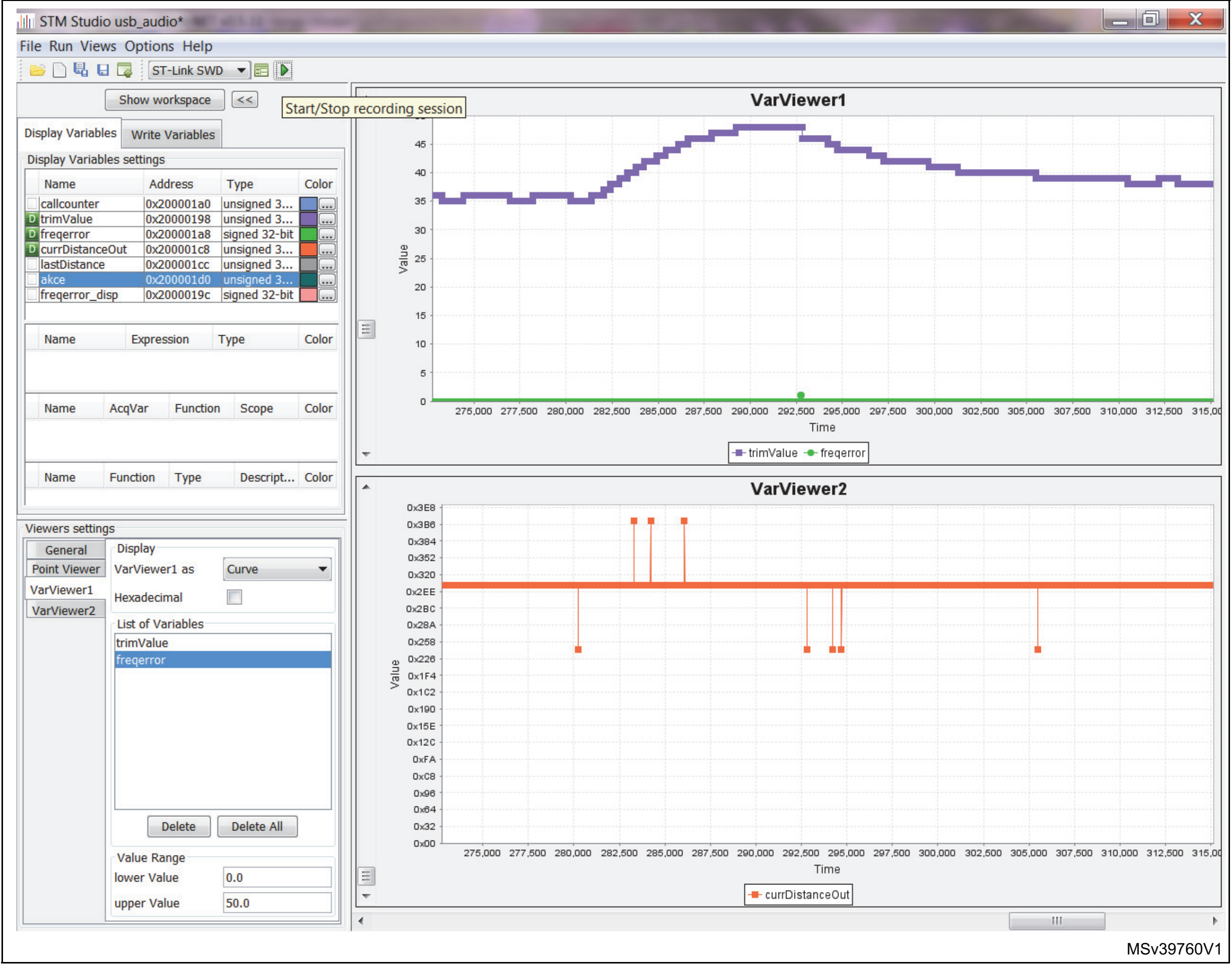
Value Range: lower Value 0.0; upper Value 37.0

With 5 ms of dead time the system is very responsive. The convergence time at the start of the audio playback is very short. The span of the pointer distance is limited, and its off-center linger time is very short. The risk of overlap of read and write pointers is minimized.

*Figure 7* shows the behavior of the algorithm with temperature variation. Here, the MCU is heated for a few seconds with hot air, starting at 28 seconds on the plot.

With the dead time set to 5 ms, the responsiveness to temperature changes is very fast, which leads to very short off-center linger time. Consequently, the risk of overlap of the buffer pointers due to temperature variation is extremely low.

**Figure 7. Software trimming, 5 ms dead time, temperature variation**



Display Variables settings

| | Name | Address | Type | Color |
|---|---|---|---|---|
| | callcounter | 0x200001a0 | unsigned 3... | 🟦 |
| D | trimValue | 0x20000198 | unsigned 3... | 🟪 |
| D | freqerror | 0x200001a8 | signed 32-bit | 🟩 |
| D | currDistanceOut | 0x200001c8 | unsigned 3... | 🟧 |
| | lastDistance | 0x200001cc | unsigned 3... | ⬜ |
| | akce | 0x200001d0 | unsigned 3... | 🟦 |
| | freqerror_disp | 0x2000019c | signed 32-bit | 🟥 |

Viewers settings: VarViewer1

Display
VarViewer1 as: Curve
Hexadecimal: ☐

List of Variables
trimValue
freqerror

Value Range
lower Value: 0.0
upper Value: 50.0

# 3 Conclusion

The method and algorithm described allows flawless USB audio playback or recording with STM32F0 microcontrollers using the USB interface and CRS unit, without any need of auxiliary resources (such as a crystal or special PLL).

Although objective measurements of audio distortion are not available, the sound does not exhibit perceivable distortion with any of the tested dead-time values.

# 4 Reference documents

Table 3. Reference documents

| Reference | Title | Version | Author |
|---|---|---|---|
| [1] | STM32F0x2xx USB Full Speed Device Library, UM1717 | Latest version | STMicroelectronics |
| [2] | Getting started with STM-STUDIO, UM1025 | Latest version | STMicroelectronics |
| [3] | STM32F0x1/STM32F0x2/STM32F0x8 Reference Manual, RM0091 | Latest version | STMicroelectronics |
| [4] | Universal Serial Bus Specification | Revision 2.0, April 27, 2000 | USB Implementer’s Forum, Inc. |

# 5 Revision history

Table 4. Document revision history

| Date | Revision | Changes |
|---|---|---|
| 19-Sep-2016 | 1 | First release. |