AN4957
Application note

How to synchronize the DFSDMs filters and how to program the pulse skipper on the STM32F413/423 line devices

Introduction

The STM32F413/423 line microcontrollers implement a dedicated mechanism on the top level of the DFSDMs (digital filter for Sigma-Delta modulators) which enables below features:

- Synchronize the two DFSDMs filters
- Pulse skipper

The pulse skipper refers to a scenario when a device dynamically tune the PDM (pulse density modulation) delays of each microphone without having to add any external delay lines. Some examples of audio applications using the pulse skipper feature are the beamforming and the sound source localization.

The purpose of this application note is to describe how to synchronize the DFSDMs filters and how to program the pulse skipper on the STM32F413/423 line devices.

This application note describes in detail:

- How to configure the pulse skipper mechanism to inject audio clock and data to DFSDMs inputs
- How to configure TIM3 and TIM4 timers to generate a requested pulse to delay one or several PDM streams
- How to build a sequence to synchronize the two DFSDMs filters
- How to build a sequence to generate a pulse

This application note assumes that the reader is familiar with the DFSDM and TIMER of the STM32 MCUs as described in the *STM32F413/423 advanced ARM® - based 32-bit MCUs* reference manual (RM0430) available from the ST Microelectronics website *www.st.com*.

Related firmware

- STM32CubeF4 embedded software for STM32F4 Series

Contents

1 Overview of the STM32F413/423 line mechanism implemented on top level of the DFSDMs 5

2 DFSDMs filters synchronization configuration 7

2.1 General description of DFSDMs filters synchronization 7

2.2 Programming sequence for DFSDMs filters synchronization 7

3 Pulse skipper configuration 9

3.1 General description of pulse skipper 9

3.2 Programming sequence for pulse skipper 10

3.3 Pulse skipper within the beamforming use case 12

4 STM32CubeF4 firmware API functions dedicated to this mechanism 15

4.1 MCHDLY (multi-channel delay) configuration 15

4.1.1 HAL DFSDM function: void HAL_DFSDM_BitstreamClock_Start(void) . 15

4.1.2 HAL DFSDM function void HAL_DFSDM_BitstreamClock_Stop(void) . 15

4.1.3 HAL DFSDM function void HAL_DFSDM_DisableDelayClock (uint32_t MCHDLY) 15

4.1.4 HAL DFSDM function void HAL_DFSDM_EnableDelayClock (uint32_t MCHDLY) 15

4.1.5 HAL DFSDM function void HAL_DFSDM_ConfigMultiChannelDelay (DFSDM_MultiChannelConfigTypeDef* mchdlystruct) 16

4.2 Synchronization of DFSDMs filters 17

4.3 Pulse skipper 18

4.3.1 Pulse skipper function void Pulse_Skipper_Init(void) 18

4.3.2 Pulse skipper function void Pulse_Skipper_Bitstream_Start(void) 18

4.3.3 Pulse skipper function void Pulse_Skipper_Bitstream_Stop(void) 18

4.3.4 Pulse skipper function void Pulse_Skipper_Generate_Pulse (PulseSkipper_InitTypeDef* skipperstruct) 18

5 Conclusion 20

6 Relevant terms 21

7 Revision history 22

List of tables

Table 1. Document revision history 22

List of figures

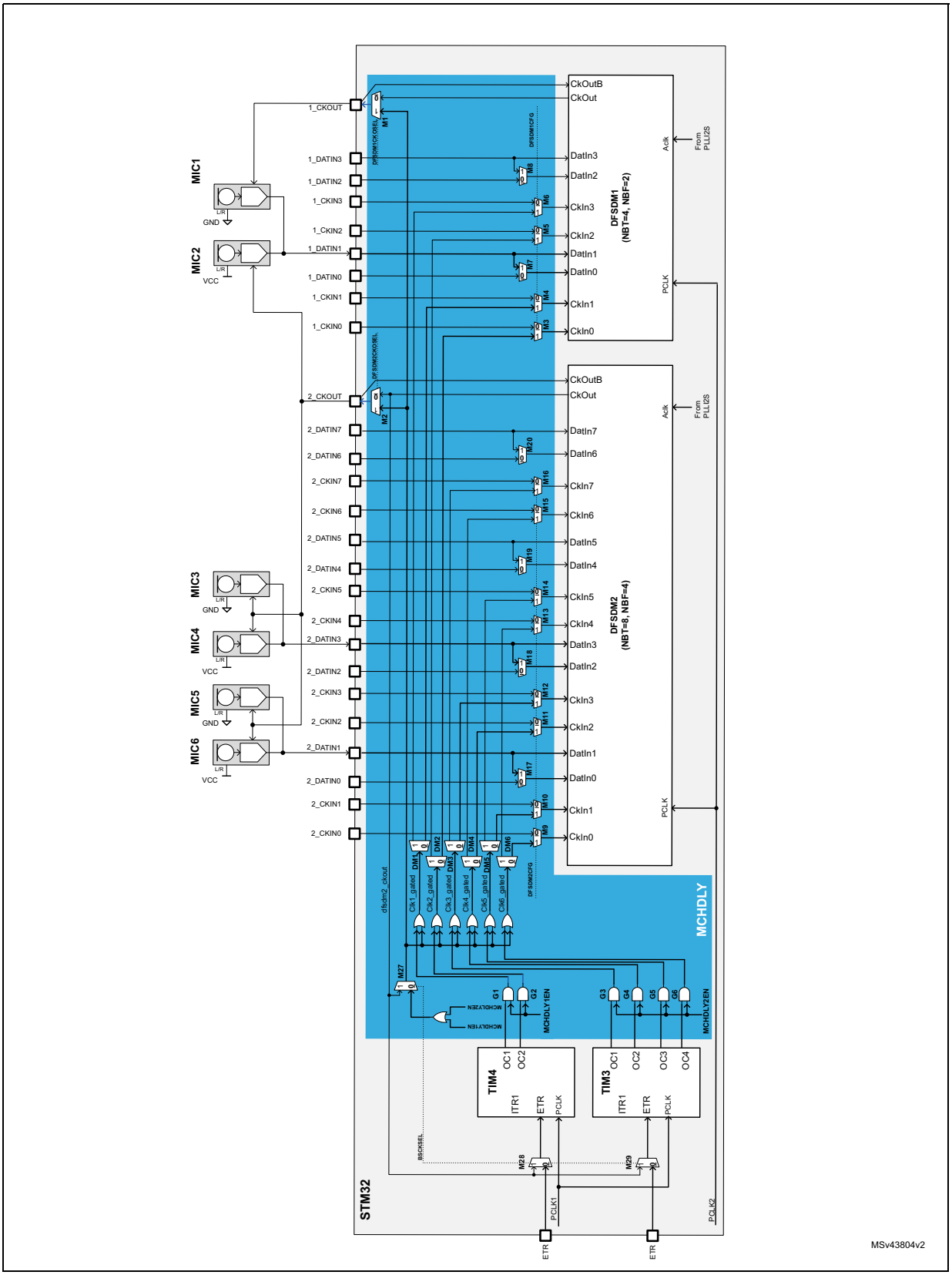
Figure 1. Mechanism implemented on top level of the DFSDMs 6
Figure 2. Clock gating concept 10
Figure 3. Beamforming use case 13
Figure 4. Distance required to have the two microphones set to 1 PCM delay 14

1 Overview of the STM32F413/423 line mechanism implemented on top level of the DFSDMs

The mechanism implemented on the top level of the DFSDMs enabling the synchronization of the two DFSDMs filters and the pulse skipper use case is represented in *Figure 1*.

As shown in *Figure 1*, the MCHDLY (multi-channel delay) block is part of both the DFSDM filters synchronization and the pulse skipper use case. For the pulse skipper use case, the TIM3 and TIM4 timers are also part of the implemented mechanism.

Figure 1. Mechanism implemented on top level of the DFSDMs



2 DFSDMs filters synchronization configuration

2.1 General description of DFSDMs filters synchronization

The STM32F413/423 line devices have two DFSDM:

- DFSDM1: made of four channels and two filters
- DFSDM2: made of eight channels and four filters.

Each DFSDM can start conversion synchronously with the first DFSDM filter (DFSDM_FLT0), but some cases there is a need to synchronize the DFSDM1 and DFSDM2 filters together.

In order to synchronize the DFSDM1 and DFSDM2 filters together the user must follow the steps below:

1. Use the external clock source and the DFSDM audio source as input clock
2. Disable the DFSDM2 clock output (dfsdm2_ckout)
3. Program each DFSDM
4. Start the filters conversion
5. Enable the DFSDM2 clock output (dfsdm2_ckout)

By performing the steps above, the conversion of the two DFSDMs filters starts synchronously.

2.2 Programming sequence for DFSDMs filters synchronization

The sequence to follow is mainly dependent on the MCHDLY block and is the one below:

1. Disable the DFSDM2 clock output (dfsdm2_ckout) signals (SYSCFG_MCHDLYCR.BSCKSEL set to 0 - **M27**)
2. Enable the external DFSDM output clock
3. Configure the DFSDMs in order to select the external CKINx as input clock
4. Configure the DFSDMs in order to take the channel data from the pins of the same channel
5. Output the DFSDM2 external output clock (dfsdm2_ckout) to DFSDMx CKOUT pads (optional): SYSCFG_MCHDLYCR.DFSDM1CKOSEL (**M1**) and SYSCFG_MCHDLYCR.DFSDM2CKOSEL (**M2**) set to 1
6. Set the DFSDM2 clock output (dfsdm2_ckout) paths to the CKINx depending on the DATINx used by:
   a) Configuring the following DMx switches
   – DFSDM1: SYSCFG_MCHDLYCR. DFSDM1_CK02SEL set to 0 to inject DFSDM2 clock output (dfsdm2_ckout) to CKIN0 (**DM2**)
   – DFSDM1: SYSCFG_MCHDLYCR. DFSDM1_CK02SEL set to 1 to inject DFSDM2 clock output (dfsdm2_ckout) to CKIN2 (**DM2**)
   – DFSDM1: SYSCFG_MCHDLYCR. DFSDM1_CK13SEL set to 0 to inject DFSDM2 clock output (dfsdm2_ckout) to CKIN1 (**DM1**)
   – DFSDM1: SYSCFG_MCHDLYCR. DFSDM1_CK13SEL set to 1 to inject DFSDM2 clock output (dfsdm2_ckout) to CKIN3 (**DM1**)

– DFSDM2: SYSCFG_MCHDLYCR. DFSDM2_CK04SEL set to 0 to inject DFSDM2 clock output (dfsdm2_ckout) to CKIN0 (**DM6**)
– DFSDM2: SYSCFG_MCHDLYCR. DFSDM2_CK04SEL set to 1 to inject DFSDM2 clock output (dfsdm2_ckout) to CKIN4 (**DM6**)
– DFSDM2: SYSCFG_MCHDLYCR. DFSDM2_CK15SEL set to 0 to inject DFSDM2 clock output (dfsdm2_ckout) to CKIN1 (**DM5**)
– DFSDM2: SYSCFG_MCHDLYCR. DFSDM2_CK15SEL set to 1 to inject DFSDM2 clock output (dfsdm2_ckout) to CKIN5 (**DM5**)
– DFSDM2: SYSCFG_MCHDLYCR. DFSDM2_CK26SEL set to 0 to inject DFSDM2 clock output (dfsdm2_ckout) to CKIN2 (**DM4**)
– DFSDM2: SYSCFG_MCHDLYCR. DFSDM2_CK26SEL set to 1 to inject DFSDM2 clock output (dfsdm2_ckout) to CKIN6 (**DM4**)
– DFSDM2: SYSCFG_MCHDLYCR. DFSDM2_CK37SEL set to 0 to inject DFSDM2 clock output (dfsdm2_ckout) to CKIN3 (**DM3**)
– DFSDM2: SYSCFG_MCHDLYCR. DFSDM2_CK37SEL set to 1 to inject DFSDM2 clock output (dfsdm2_ckout) to CKIN7 (**DM3**)

b) Configuring the following Mx switches

– DFSDM1: SYSCFG_MCHDLYCR. DFSDM1CFG set to 1 to inject DFSDM2 clock output (dfsdm2_ckout) to CKIN0, CKIN1, CKIN2 and CKIN3 (**M3, M4, M5, M6**)
– DFSDM2: SYSCFG_MCHDLYCR. DFSDM2CFG set to 1 to inject DFSDM2 clock output (dfsdm2_ckout) to CKIN0, CKIN1, CKIN2, CKIN3, CKIN4, CKIN5, CKIN6 and CKIN7 (**M9, M10, M11, M12, M13, M14, M15, M16**)

7. Start conversions of all used DFSDM1 and DFSDM2 filters
8. Release the DFSDM2 clock output (dfsdm2_ckout) signals (SYSCFG_MCHDLYCR.BSCKSEL set to 1 - **M27**)

Note that for DFSDM1, the DFSDM2 clock output (dfsdm2_ckout) can be injected on:

- CKIN0 or CKIN2
- CKIN1 or CKIN3

For DFSDM2, the DFSDM2 clock output (dfsdm2_ckout) can be injected on:

- CKIN0 or CKIN4
- CKIN1 or CKIN5
- CKIN2 or CKIN6
- CKIN3 or CKIN7

3 Pulse skipper configuration

3.1 General description of pulse skipper

The purpose of the pulse skipper is to implement a delay-line like behavior for a given input channel(s). The pulse skipper enables that a given number of samples from the input serial data stream (serial stream only) be discarded before they enter into the filter. This data discarding is performed by skipping a given number of sampling input clock pulses (given serial data samples are then not sampled by the filter).

The sampling clock is gated by the pulse skipper function for a given number of clock pulses. When a given number of clock pulses are skipped, the filtering continues for the following input data. Compared to a non-skipped data stream, this operation causes that the final output sample (and the next samples) from the filter are calculated from the later input data.

The final sample looks shifted forward because it is calculated from newer input samples than the "non-skipped" sample. The final “skipped sample” is converted later because the skipped input data samples must be replaced by the subsequent input data samples.

The final data buffers behavior (skipped and non-skipped output data buffers comparison) looks like the non-skipped data stream is somehow delayed as both data buffers are phase shifted.

The implementation of the clock skipping is based on the MCHDLY, TIM3 and TIM4 blocks. The MCHDLY block is controlled via a MCHDLYCR register where all MCHDLY control bits are.

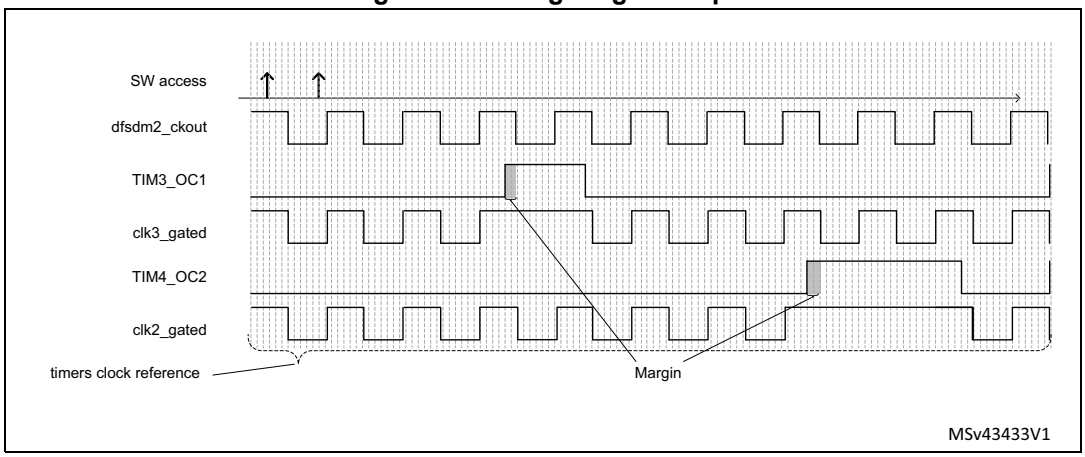
By using the MCHDLY block as shown in *Figure 1*, the DFSDMs can be used for an audio application with up to six digital microphones. The clock to the digital microphones (or Sigma-Delta modulators) is provided by the DFSDM2 clock output (dfsdm2_ckout signal). This clock output signal is then distributed to:

- The OR gates to implement the pulse skipping by clock masking or gating
- The ETR (external trigger) inputs of the TIM4 and the TIM3 timers to define how many pulses must be skipped and they are also used for synchronous purposes
- The DFSDM1_CKOUT and DFSDM2_CKOUT pins outputs through M1 and M2 multiplexers to generate the output clock signal on the DFSDMx pins

The OR gates are used to skip the input serial clock pulses provided to the DFSDMs in order to generate a delay on the corresponding input channel. This clock gating is controlled by two timers (TIM4 and TIM3). The timers are programmed in a one-shot mode to generate the masking pulse with a defined length to gate the required number of clock pulses.

Figure 2 shows a case where a delay of one DFSDM clock period is generated on the input channel 3 or 7 of the DFSDM2 (depending on the selected input channel). It shows also that another delay of two DFSDM clock periods is generated on the input channel 0 or 2 of the DFSDM1 (depending on the selected input channel).

Figure 2. Clock gating concept



3.2 Programming sequence for pulse skipper

This section describes the sequence to follow to generate a pulse skipper on a given input channel.

Configure the MCDLY block (it also allows to synchronize the DFSDMs filters if the used input channels are on two different DFSDMs)

1. Disable the DFSDM2 clock output (dfsdm2_ckout) signals (SYSCFG_MCHDLYCR.BSCKSEL set to 0 - **M27**)
2. Enable the external DFSDM output clock
3. Configure the DFSDMs in order to select the external CKINx as input clock
4. Configure the DFSDMs in order to take the channel data from the pins of the same channel
5. Output the DFSDM2 external output clock (dfsdm2_ckout) to DFSDMx CKOUT pads: SYSCFG_MCHDLYCR.DFSDM1CKOSEL (**M1**) and SYSCFG_MCHDLYCR.DFSDM2CKOSEL (**M2**) set to 1

   Note: In Figure 1, the MIC1 receives its serial clock via another clock output pin (DFSDM1_CKOUT pin) which can provide either a clock output from DFSDM2 or from DFSDM1 CKOUT signal. This configuration allows to use for low-power use-cases only one microphone (MIC1) while DFSDM2_CKOUT is disabled and DFSDM1_CKOUT enabled (for example for voice detection), and reusing MIC1 for beamforming use cases (when DFSDM2_CKOUT is enabled).
6. Set the DFSDM2 clock output (dfsdm2_ckout) paths to the CKINx depending on the DATINx channels used by
   a) Configuring the following DMx switches
   – DFSDM1: SYSCFG_MCHDLYCR. DFSDM1_CK02SEL set to 0 to inject DFSDM2 clock output (dfsdm2_ckout) to CKIN0 (**DM2**)
   – DFSDM1: SYSCFG_MCHDLYCR. DFSDM1_CK02SEL set to 1 to inject DFSDM2 clock output (dfsdm2_ckout) to CKIN2 (**DM2**)
   – DFSDM1: SYSCFG_MCHDLYCR. DFSDM1_CK13SEL set to 0 to inject DFSDM2 clock output (dfsdm2_ckout) to CKIN1 (**DM1**)

- DFSDM1: SYSCFG_MCHDLYCR. DFSDM1_CK13SEL set to 1 to inject DFSDM2 clock output (dfsdm2_ckout) to CKIN3 (**DM1**)
- DFSDM2: SYSCFG_MCHDLYCR. DFSDM2_CK04SEL set to 0 to inject DFSDM2 clock output (dfsdm2_ckout) to CKIN0 (**DM6**)
- DFSDM2: SYSCFG_MCHDLYCR. DFSDM2_CK04SEL set to 1 to inject DFSDM2 clock output (dfsdm2_ckout) to CKIN4 (**DM6**)
- DFSDM2: SYSCFG_MCHDLYCR. DFSDM2_CK15SEL set to 0 to inject DFSDM2 clock output (dfsdm2_ckout) to CKIN1 (**DM5**)
- DFSDM2: SYSCFG_MCHDLYCR. DFSDM2_CK15SEL set to 1 to inject DFSDM2 clock output (dfsdm2_ckout) to CKIN5 (**DM5**)
- DFSDM2: SYSCFG_MCHDLYCR. DFSDM2_CK26SEL set to 0 to inject DFSDM2 clock output (dfsdm2_ckout) to CKIN2 (**DM4**)
- DFSDM2: SYSCFG_MCHDLYCR. DFSDM2_CK26SEL set to 1 to inject DFSDM2 clock output (dfsdm2_ckout) to CKIN6 (**DM4**)
- DFSDM2: SYSCFG_MCHDLYCR. DFSDM2_CK37SEL set to 0 to inject DFSDM2 clock output (dfsdm2_ckout) to CKIN3 (**DM3**)
- DFSDM2: SYSCFG_MCHDLYCR. DFSDM2_CK37SEL set to 1 to inject DFSDM2 clock output (dfsdm2_ckout) to CKIN7 (**DM3**)

b) Configuring the following Mx switches

- DFSDM1: SYSCFG_MCHDLYCR. DFSDM1CFG set to 1 to inject DFSDM2 clock output (dfsdm2_ckout) to CKIN0, CKIN1, CKIN2 and CKIN3 (**M3, M4, M5, M6**)
- DFSDM2: SYSCFG_MCHDLYCR. DFSDM2CFG set to 1 to inject DFSDM2 clock output (dfsdm2_ckout) to CKIN0, CKIN1, CKIN2, CKIN3, CKIN4, CKIN5, CKIN6 and CKIN7 (**M9, M10, M11, M12, M13, M14, M15, M16**)

7. Start conversions of all used DFSDM1 and DFSDM2 filters
8. Release the DFSDM2 clock output (dfsdm2_ckout) signals (SYSCFG_MCHDLYCR.BSCKSEL set to 1 - **M27**)

Configure TIM3 and TIM4 to generate the pulse skipper on a given input channel

1. The TIM4 outputs (OC[2:1]) connected to the OR gates of the DFSDM1 CKINs and the TIM3 outputs (OC[4:1]) connected to the OR gates of the DFSDM2 CKINs must be low in inactive mode when no gating is required
2. Configure the timers in OPM (one pulse mode). In this mode, the timer allows the counter to be started in response to a rising edge on the ETR input and to generate a pulse with a programmable length. This pulse is used when the bitstream clock is generated by the DFSDM2.

Note: The clock reference used by the timers can be either the APB clock or the APB clock multiplied by 2 or 4. The higher this frequency is, the better it is.

When TIM3 and TIM4 are used to gate the dfsdm2_ckout audio clock, the timers reference clock frequency must be at least 12 times the frequency of the DFSDM2 audio clock (assuming that the bitstream clock duty cycle is 50%).

Note: The TIM2 OC1 shall not be used as it has been connected to M27 to backup the DFSDM2 clock output (dfsdm2_ckout) signal.

Enable clock gating for DFSDM1 and DFSDM2

1. Enable the clock gating for DFSDM1: SYSCFG_MCHDLYCR. MCHDLY1EN set to 1 (G1 and G2) to allow TIM4 generated pulse to be injected to the corresponding DFSDM1 CKINx
2. Enable the clock gating for DFSDM2: SYSCFG_MCHDLYCR. MCHDLY2EN set to 1 (G3, G4, G5 and G6) to allow the TIM3 generated pulse to be injected to the corresponding DFSDM2 CKINx

When the pulse skipper has to be issued, the application programs on the TIM3 or the TIM4 OC channel of the respective OR gate the pulse with the required length.

Only one pulse can be generated at a time, so the application must wait the end of the current generated pulse before issuing the next one (if needed).

3.3 Pulse skipper within the beamforming use case

Figure 3 shows the full view of the beamforming use case and where the pulse skipper fits in that use case.

In a case where the programmed PCM (pulse code modulation) sample rate is set to 16 Khz, and in order to avoid CPU processing, the microphones distance must be set to 21 mm, which corresponds to 1 PCM delay between two microphones.

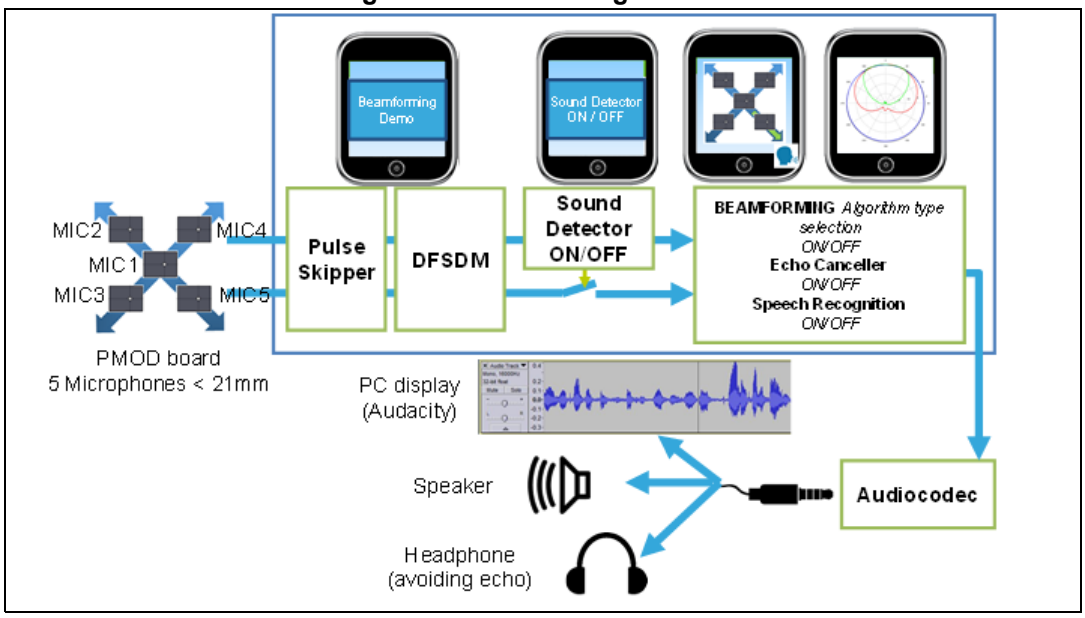
The pulse skipper feature allows to release this distance constraint between two microphones and virtually set the distance of one or several PCM delays between those two microphones.

In the *Figure 3*, a Pmod™ board is composed of five microphones connected to the DFSDMs. The distance between the microphones (M1-M2, M1-M3, M1-M4, M1-M5) is less than 21 mm.

The GUI (graphical user interface) of the beamforming use case allows the user to select the direction (among four possible directions) to apply the beamforming. The previous action defines the two MICs to be used (for instance MIC1 and MIC2).

As the two selected microphones have a distance of less than 21 mm, the pulse skipper block is used to virtually increase the distance between the two selected microphones. This action allows to have a distance of 21 mm, which corresponds to 1 PCM delay between MIC1 and MIC2 (see *Figure 3*) required to run the use case.

Figure 3. Beamforming use case



To calculate the pulse skipper period to use, the following formula must be applied:

NB_BITSTREAM_CLOCK_PERIOD = (Dpcm-Dm12) / (T x c)

where:

- Dpcm = the expected virtual distance between the two microphones corresponding to 1 PCM delay (21mm for a PCM sampling rate set to 16 KHz)
- Dm12 = the actual distance between the two microphones
- T = DFSDM2 audio clock period (for instance 0.5.10E-6 s, assuming that the microphone frequency used = 2.048 MHz)
- c = sound speed 340 m/s

The *Figure 4* shows the distance required to have the two microphones set to a distance of 1 PCM delay.

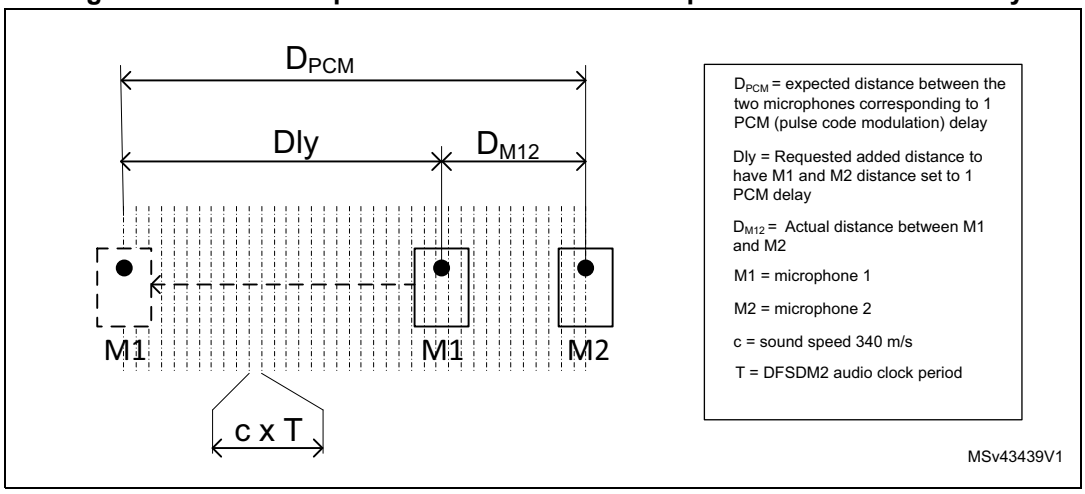
For instance:

- PCM sampling rate set to 16 KHz: Dpcm = 21mm (1 PCM delay)
- Dm12 = 5 mm (actual distance between M1 and M2)
- PDM (pulse density modulation) sampling rate set to 2.048Mhz ==> T = 0.510E-6s

NB_BITSTREAM_CLOCK_PERIOD = (21-5).10E-5 / (340x0,510E-6) = 94

This number corresponds to the number of dfsdm clock periods to gate (TIM OC pulse duration) as seen in *Figure 2*.

Figure 4. Distance required to have the two microphones set to 1 PCM delay



4 STM32CubeF4 firmware API functions dedicated to this mechanism

An example explaining how to generate a pulse skipper is provided as part of the STM32CubeF4 firmware v1.15.0 at: STM32Cube_FW_F4_V1.15.0\Projects\STM32F413H-Discovery\Examples\DFSDM.

4.1 MCHDLY (multi-channel delay) configuration

This section presents the functions that are part of the DFSDM driver stm32f4xx_hal_dfsdm.c, and that are used to configure and control the MCHDLY block.

4.1.1 HAL DFSDM function: void HAL_DFSDM_BitstreamClock_Start(void)

- **Function**: void HAL_DFSDM_BitstreamClock_Start(void)
- **Description**: distribute the DFSDM2 clock output (dfsdm2_ckout) to the OR gates and the DFSDM1/2 audio output (DFSDMx CKOUT) if they are configured as part of the PulseSkipper_InitTypeDef structure (see the structure definition in *Section 4.1.5*)
- **MCHDLY impact**: SYSCFG_MCHDLYCR.BSCKSEL set to 1 - **M27.**

4.1.2 HAL DFSDM function void HAL_DFSDM_BitstreamClock_Stop(void)

- **Function**: void HAL_DFSDM_BitstreamClock_Stop(void)
- **Description**: block distribution of the DFSDM2 clock output (dfsdm2_ckout) to the OR gates and the DFSDM1/2 audio output (DFSDMx CKOUT) if they are configured as part of the PulseSkipper_InitTypeDef structure (see the structure definition in *Section 4.1.5*)
- **MCHDLY impact**: SYSCFG_MCHDLYCR.BSCKSEL set to 0 - **M27**.

4.1.3 HAL DFSDM function void HAL_DFSDM_DisableDelayClock (uint32_t MCHDLY)

- **Function**: void HAL_DFSDM_DisableDelayClock(uint32_t MCHDLY)
- **Description**: block the distribution of the DFSDM2 clock output (dfsdm2_ckout) to the DFSDM1/2 CkInx
- **MCHDLY impact:** SYSCFG_MCHDLYCR. MCHDLY1EN set to 0 (**G1** and **G2**) for DFSDM1 and/or SYSCFG_MCHDLYCR. MCHDLY2EN set to 0 (**G3, G4, G5** and **G6**) for DFSDM2

4.1.4 HAL DFSDM function void HAL_DFSDM_EnableDelayClock (uint32_t MCHDLY)

- **Function**: void HAL_DFSDM_EnableDelayClock(uint32_t MCHDLY)
- **Description**: distribute the DFSDM2 clock output (dfsdm2_ckout) to the DFSDM1/2 CkInx
- **MCHDLY impact:** SYSCFG_MCHDLYCR.MCHDLY1EN set to 1 (**G1** and **G2**) for DFSDM1 and/or SYSCFG_MCHDLYCR. MCHDLY2EN set to 1 (**G3, G4, G5** and **G6**) for DFSDM2

4.1.5 HAL DFSDM function void HAL_DFSDM_ConfigMultiChannelDelay (DFSDM_MultiChannelConfigTypeDef* mchdlystruct)

- **Function**: void HAL_DFSDM_ConfigMultiChannelDelay(DFSDM_MultiChannelConfigTypeDef* mchdlystruct)
- **Description**: used to set the path of the DFSDM2 clock output (dfsdm2_ckout) to the DFSDM1/2 CkInx and data inputs channels by configuring following MCHDLY muxes or demuxes: **M1, M2, M3, M4, M5, M6, M7, M8, DM1, DM2, DM3, DM4, DM5, DM6, M9, M10, M11, M12, M13, M14, M15, M16, M17, M18, M19, M20** based on the contains of the DFSDM_MultiChannelConfigTypeDef structure
- **MCHDLY impact:**
  - **DFSDM1**: SYSCFG_MCHDLYCR.DFSDM1CKOSEL (**M1**), SYSCFG_MCHDLYCR.DFSDM1CFG (**M3, M4, M5, M6**), SYSCFG_MCHDLYCR.DFSDM1CK13SEL (**DM1**), SYSCFG_MCHDLYCR.DFSDM1CK02SEL (**DM2**), SYSCFG_MCHDLYCR.DFSDM1D0SEL (**M7**), SYSCFG_MCHDLYCR.DFSDM1D2SEL (**M8**)
  - **DFSDM2**: SYSCFG_MCHDLYCR.DFSDM2CKOSEL (**M2**), SYSCFG_MCHDLYCR.DFSDM2CFG (**M9, M10, M11, M12, M13, M14, M15, M16**), SYSCFG_MCHDLYCR.DFSDM2CK37SEL (**DM3**), SYSCFG_MCHDLYCR.DFSDM2CK26SEL (**DM4**), SYSCFG_MCHDLYCR.DFSDM2CK15SEL (**DM5**), SYSCFG_MCHDLYCR.DFSDM2CK04SEL (**DM6**), SYSCFG_MCHDLYCR.DFSDM2D0SEL (**M17**), SYSCFG_MCHDLYCR.DFSDM2D2SEL (**M18**), SYSCFG_MCHDLYCR.DFSDM2D4SEL (**M19**), SYSCFG_MCHDLYCR.DFSDM2D6SEL (**M20**)
- **Structure definition:**

```
typedef struct
{
  uint32_t  DFSDM1ClockIn;           /*!< Source selection for DFSDM1_Ckin.
This parameter can be a value of @ref DFSDM1_CLOCKIN_SELECTION*/
  uint32_t  DFSDM2ClockIn;           /*!< Source selection for DFSDM2_Ckin.
This parameter can be a value of @ref DFSDM2_CLOCKIN_SELECTION*/
  uint32_t  DFSDM1ClockOut;            /*!< Source selection for
DFSDM1_Ckout.
This parameter can be a value of @ref DFSDM1_CLOCKOUT_SELECTION*/
  uint32_t  DFSDM2ClockOut;            /*!< Source selection for
DFSDM2_Ckout.
This parameter can be a value of @ref DFSDM2_CLOCKOUT_SELECTION*/
  uint32_t  DFSDM1BitClkDistribution; /*!< Distribution of the DFSDM1
bitstream clock gated by TIM4 OC1 or TIM4 OC2.
This parameter can be a value of @ref DFSDM1_BIT_STREAM_DISTRIBUTION
@note The DFSDM2 audio gated by TIM4 OC2 can be injected on CKIN0 or CKIN2
@note The DFSDM2 audio gated by TIM4 OC1 can be injected on CKIN1 or CKIN3
*/
  uint32_t  DFSDM2BitClkDistribution; /*!< Distribution of the DFSDM2
bitstream clock gated by TIM3 OC1 or TIM3 OC2 or TIM3 OC3 or TIM3 OC4.
```

```
This parameter can be a value of @ref DFSDM2_BIT_STREAM_DISTRIBUTION
@note The DFSDM2 audio gated by TIM3 OC4 can be injected on CKIN0 or CKIN4
@note The DFSDM2 audio gated by TIM3 OC3 can be injected on CKIN1 or CKIN5
@note The DFSDM2 audio gated by TIM3 OC2 can be injected on CKIN2 or CKIN6
@note The DFSDM2 audio gated by TIM3 OC1 can be injected on CKIN3 or CKIN7
*/
  uint32_t  DFSDM1DataDistribution;   /*!< Source selection for DatIn0 and
DatIn2 of DFSDM1.
This parameter can be a value of @ref DFSDM1_DATA_DISTRIBUTION */
  uint32_t  DFSDM2DataDistribution;  /*!< Source selection for DatIn0,
DatIn2, DatIn4 and DatIn6 of DFSDM2.
This parameter can be a value of @ref DFSDM2_DATA_DISTRIBUTION */
}DFSDM_MultiChannelConfigTypeDef;
```

- **DFSDM1ClockIn**: used to select the DFSDM1 CkInx source, either DFSDM2 clock output (dfsdm2_ckout) or external audio clock from CkInx PAD (**M3, M4, M5, M6**)
- **DFSDM2ClockIn**: used to select the DFSDM2 CkInx source, either DFSDM2 clock output (dfsdm2_ckout) or external audio clock from CkInx PAD (**M9, M10, M11, M12, M13, M14, M15, M16**)
- **DFSDM1ClockOut**: used to select the DFSDM1 clock output, either DFSM2 audio clock or DFSDM1 CkOut (**M1**)
- **DFSDM2ClockOut**: used to select the DFSDM2 clock output, either DFSM2 audio clock or DFSDM2 CkOut (**M2**)
- **DFSDM1BitClkDistribution**: used to distribute the DFSDM2 clock audio to DFSDM1 CkIn0 or CkIn2 (**DM1**) and/or to DSFDM1 CkIn1 or CkIn3 (**DM2**)
- **DFSDM2BitClkDistribution**: used to distribute the DFSDM2 clock audio to DFSDM2 CkIn3 or CkIn7 (**DM3**) and/or to DSFDM2 CkIn2 or CkIn6 (**DM4**) and/or DFSDM2 CkIn1 or CkIn5 (**DM5**) and/or to DSFDM2 CkIn0 or CkIn4 (**DM6**)
- **DFSDM1DataDistribution**: used to inject the DFSDM1 DataIn0 or DataIn1 PAD to DFSDM1 DatIn0 input (**M7**) and/or DFSDM1 DataIn2 or DataIn3 PAD to DFSDM1 DatIn2 input (**M8**)
- **DFSDM2DataDistribution**: used to inject the DFSDM2 DataIn0 or DataIn1 PAD to DFSDM2 DatIn0 input (**M17**) and/or DFSDM2 DataIn2 or DataIn3 PAD to DFSDM2 DatIn2 input (**M18**) and/or DFSDM2 DataIn4 or DataIn5 PAD to DFSDM2 DatIn4 input (**M19**) and/or DFSDM2 DataIn6 or DataIn7 PAD to DFSDM2 DatIn6 input (**M20**)

4.2 Synchronization of DFSDMs filters

There is no specific function for the synchronization of the DFSDMs filters. The HAL DFSDM functions described in *Section 4.1: MCHDLY (multi-channel delay) configuration* shall be used and sequenced as described in *Section 2.2: Programming sequence for DFSDMs filters synchronization*.

4.3 Pulse skipper

The MCHDLY shall be programmed as described in *Section 4.1: MCHDLY (multi-channel delay) configuration* to configure the DFSDM2 clock output (dfsdm2_ckout) path.

This section describes four additional functions which are part of the proposed STM32F413/423 Discovery kit firmware (under preparation and to be published early 2017).

4.3.1 Pulse skipper function void Pulse_Skipper_Init(void)

- **Function**: void Pulse_Skipper_Init(void)
- **Description**: configure the TIM3 and TIM4 in OPM (one pulse mode). The length of the generated pulse is a #define NB_BITSTREAM_CLOCK_PERIOD part of the pulse_skipper.c file and passed as a parameter for the TIM3 and TIM4 timers initialization function. Then the DFSMD1 and the DFSDM2 clock gating are enable (**G1** to **G7**), in other words, the DFSDM2 clock output (dfsdm2_ckout) signal is injected on all used CkINs channels.

4.3.2 Pulse skipper function void Pulse_Skipper_Bitstream_Start(void)

- **Function**: void Pulse_Skipper_Bitstream_Start(void)
- **Description**: release to all OR gates the DFSDM2 clock output (dfsdm2_ckout) signal (**M27**).

4.3.3 Pulse skipper function void Pulse_Skipper_Bitstream_Stop(void)

- **Function**: void Pulse_Skipper_Bitstream_Stop(void)
- **Description**: block to all OR gates the DFSDM2 clock output (dfsdm2_ckout) signal (**M27**).

4.3.4 Pulse skipper function void Pulse_Skipper_Generate_Pulse (PulseSkipper_InitTypeDef* skipperstruct)

- **Function**: void Pulse_Skipper_Generate_Pulse(PulseSkipper_InitTypeDef* skipperstruct)
- **Description**: generate from the selected TIM3 or TIM4 Output Compare channel the pulse with the initialized pulse length (#define NB_BITSTREAM_CLOCK_PERIOD). Before issuing a new pulse, the previous generated one must be finished
- **Structure definition:**

```
Structuretypedef struct
{
  uint32_t  DFSDM1PulseSkipperCh;    /*!< Channels selection to generate
pulse skipper of DFSDM1.
This parameter can be a value of @ref DFSDM1_PULSESKIPPER_CH */
  uint32_t  DFSDM2PulseSkipperCh;    /*!< Channels selection to generate
pulse skipper of DFSDM2.
This parameter can be a value of @ref DFSDM2_PULSESKIPPER_CH */
```

```
}PulseSkipper_InitTypeDef;
```

- **DFSDM1PulseSkipperCh**: used to generate a pulse on the DFSDM1 Channel 0, Channel1, Channel 2 or Channel 3 selected as part of the MCHDLY configuration thanks to DFSDM1BitClkDistribution
- **DFSDM2PulseSkipperCh**: used to generate a pulse on the DFSDM2 Channel 0, Channel 1, Channel 2, Channel 3, Channel 4, Channel 5, Channel 6 or Channel 7 selected as part of the MCHDLY configuration thanks to DFSDM2BitClkDistribution.

Note: For DFSDM1: pulse can be generated on Channel 0 or Channel 2, Channel 1 or Channel 3.

For DFSDM2: pulse can be generated on Channel 0 or Channel 4, Channel 1 or Channel 5, Channel 2 or Channel 6, Channel 3 or Channel 5.

Note: Only one pulse must be generated at a time, the application must wait the end of the pulse before generating another pulse. If the pulse length has to be updated, the TIM3 or TIM4 timers must be re-initialized with the new pulse length (update #define NB_BITSTREAM_CLOCK_PERIOD).

5 Conclusion

This application note explains how to configure the mechanism implemented on top of the DFSDMs in order to:

- Synchronize the two DFSDMs filters
- Apply the pulse skipper use case by dynamically tune the PDM delays of each microphone without the need to add external delay lines (some audio applications using the pulse skipper feature are the beamforming and the sound source localization).

The configuration of the involved blocks (MCHDLY, TIM3 and TIM4) with the sequence to follow are described in *Section 2: DFSDMs filters synchronization configuration* and *Section 3: Pulse skipper configuration*.

The STM32CubeF4 firmware API functions dedicated to this mechanism are described in *Section 4: STM32CubeF4 firmware API functions dedicated to this mechanism*. A code example to generate a pulse skipper is available as part of the STM32CubeF4 FW v1.15.0 at STM32Cube_FW_F4_V1.15.0\Projects\STM32F413H-Discovery\Examples\DFSDM.

6 Relevant terms

- PDM: pulse density modulation. The digital microphones (or other kind of sensors) are providing data encoded into PDM. The rate of these data is controlled by the DFSDM clock output. A usual rate for PDM samples is between 1 and 3 MHz for digital microphones.
- PCM: pulse code modulation. The PDM encoded data received from the digital microphones (or other sensors) are filtered and decimated by the DFSDM. The resulting samples are PCM coded, and provided at a rate dependent of the decimation ratio and the DFSDM clock output frequency. A usual sampling rate for PCM samples is 16 KHz.
- DFSDM1 clock output: external DFDSM1 clock output from DFSDM audio clock or system clock.
- DFSDM2 clock output: external DFDSM2 clock output from DFSDM audio clock or system clock.

7 Revision history

Table 1. Document revision history

Date	Revision	Changes
19-Dec-2016	1	Initial release.