ECSE 444: Microprocessors
Lab 5: MEMS Microphone and DFSDM

Abstract

In this lab you will learn how to use the on-board MEMS microphone and play the resulting audio through the on-chip DAC. To do so, you'll use the on-chip digital filter for sigma-delta modulators (DFSDM) to convert the microphone's pulse density modulation (PDM) output into a conventional digital signal interpretable by the DAC.

Deliverables for demonstration

- Matching input and output sampling configuration in MX
- C implementation of processing and rescaling of DFSDM output for DAC
- C implementation of appropriate interrupts for push-button recording and playback
- Audible confirmation of push-button recording and playback

Grading

- 20% Input and output sampling
- 30% Data preparation for DAC
- 30% Appropriate interrupts for buttons and DFSDM
- 20% Push-button recording and playback

Changelog

- 17-Aug-2020 Initial revision.
- 03-Oct-2022 Updated for B-L4S5I-IOT01A.

Overview

In this lab, we'll learn about the on-board MEMS microphone and continue to explore DMA and interrupts, building on techniques introduced in Lab 4 and earlier. The objective is to record a short audio clip, and then play it back continuously on our speaker. We'll configure a new peripheral, a digital filter for sigma-delta modulator (DFSDM), which converts the pulse density modulated output (i.e., digital, with duty cycle in proportion to input frequency) of the microphone into a 24-bit signed integer. We'll then take this value and transform it for output to our DAC. The application will be driven by button presses: press the button to record, after which the recorded clip automatically plays on loop until the button is pressed again.

Resources

B-L475E-IOT01A User Manual
B-L4S5I-IOT01A User Manual
HAL Driver User Manual
Interfacing PDM digital microphones Application Note
MP34DT01-M Datasheet

© Brett H. Meyer

STM32L475VG Datasheet

STM32L4S5VI Datasheet for B-L4S5I-IOT01A
STM32L47xxx Reference Manual

Configuration

Initialization

Start a new project in STM32CubeMX, and initialize it as in earlier labs. For this lab, we need:

- The blue push-button,
- One DAC output channel, and associated timer,
- A DFSDM channel and filter pair.

DFSDM

We'll configure the DFSDM first, and the settings for the other peripherals will follow. The DFSDM must be configured to read the output of the microphone and filter the binary signal into digital values. We'll read a single microphone's output, rather than both. For information on the basic mechanics of this process, see Interfacing PDM digital microphones Application Note and MP34DT01-M Datasheet.

The DFSDM input channels are wired to different pins in order to interface with different peripherals. Therefore, as we have in the past, we must start in the development kit user manual and microcontroller datasheet then figure out which pins are wired to the on-board microphones. (Check the schematic!) The microphone interface consists of two signals: one for data, and another for clock. Identify the channel that connects to the correct data signal, and enable it (*PDM/SPI input from chX and internal clock*). Check the CKOUT to enable the output clock signal that will be used to synchronize with the microphone.

Note: by default, MX does not assign the clock to the correct pin even when you've identified the appropriate channel; it is sufficient to select the appropriate pin, and choose the clock signal, as this disables the assignment of the signal to any other pin.

Once you've ensured that the DFSDM channel that you've enabled is connected to the correct pins, it's time to configure the DFSDM itself. We need to set up: (a) a filter to convert the PDM input into a digital output, (b) the clock to the microphone, and (c) DMA to transfer from the DFSDM to memory.

Choose a filter; it doesn't matter which one, as any channel may be connected to any filter. Enable the appropriate channel as a "regular channel." Ensure that *Continuous Mode*, and *Software trigger* are selected. Enable *Fast Mode* and *Dma Mode*.

The other filter parameters must be set with the Output Clock settings in mind, because they work together to determine the sampling rate. We're using the system clock to drive the microphone. DFSDM *Output Clock* configuration gives an option to configure a clock divider. Choose an appropriate value, given (a) the operating frequency of your system (*Clock Configuration*), and the requirements of the microphone (MP34DT01-M Datasheet). Under filter configuration, *Fors* specifies the oversampling rate of the filter. (Leave Iosr as 1.) The clock rate to the microphone, divided by the oversampling rate, determines the true sampling rate. Not all oversampling rates are supported by all Sinc filter types; a higher order generally achieves better resolution. (In testing I used Sinc 3, but did not experiment with other options.)

A trade-off emerges: a higher sampling rate (e.g., 44.1kHz) will allow us to produce a clearer output. However, this requires more storage for the same recording interval (e.g., 1 second). A lower sampling rate (e.g., 8kHz) will produce less clear output, but requiring less storage, and enabling us to record for longer and still fit the data in SRAM.

Choose a target sampling frequency (which you will need to reproduce at the DAC for correct playback), and select system clock, output clock divider, and oversampling frequency, accordingly.

Finally, we need to configure the DMA. Note that given our current setup (a single channel and microphone), the DFSDM will produce a 24-bit output, padded in the least significant 8 bits with channel information. (Resulting in a complete 32-bit word.) Also note that "normal" or "circular" mode are reasonable selections, depending on how you ultimately structure your program. (See Lab 4 for a very brief description of each mode.)

DAC and Timer

Configure a single DAC output channel and timer, as in Lab 4. Configure for DMA transfer to the DAC. Timer parameters should be chosen such that the output sampling rate matches the input sampling rate above.

Push button

Configure the blue push button as in Lab 4 so that when pressed it generates an interrupt.

Implementation

Now implement a program that records microphone input and plays it back on the speaker. Ultimately, your program must begin recording when the button is pressed, and begin playback, without user intervention once the recording buffer is full and has been processed. You should be able to record for a couple of seconds; you will get a linking error if the arrays you allocate are too large to be stored in SRAM.

The new functions you will need to control the DFSDM can be found in Section 19.2 of HAL Driver User Manual.

Notes

- Be mindful of data types. The DFSDM will produce 32-bits per sample; the most significant 24 is *signed* data, the lower 8 is information about the channel (and can be discarded). The data must be rescaled for the (*unsigned*) output range of the DAC.
- *Normal* DMA will perform a transfer once and stop; starting it again will repeat this process. *Circular* DMA, once started, will run until it is explicitly stopped by function call.
- In normal mode, DFSDM DMA transfer will generate a *CpltCallback* interrupt when it is done with the array. Like the `HAL_GPIO_EXTI_Callback(...)` function that we overwrote in Lab 4, there are corresponding functions for other peripherals. Note that we are using regular conversion, not injected conversion.
- In circular mode, DFSDM DMA transfer will additionally generate a *HalfCpltCallback* interrupt when it is done with half of the array. This makes it possible to process the first half of an array while the second half is being filled. This is necessary in circular mode, because otherwise, e.g., the DFSDM will continue to write, overwriting the array before you've processed it.
- Carefully test your setup to ensure that there's a match between input sampling and output production. The microphone and speaker are both suitable for reproducing speech when configured properly. (Don't forget to start your timer! I did.)
- Note that you need to be quite close to the microphone for it to register at full dynamic range. You can find it on the board layout by looking in the user manual of the development kit.