Project: Blind Source Separation Using ICA

ECSE 444: Microprocessors

Winter 2019

Introduction

For the final project, you will use all the material you have learned over the course of the labs to build an audio application that employs Blind Source Separation(BSS) using the Fast Independent Components Analysis(FastICA) algorithm. There is an initial deliverable due the week of November 19th in order to ensure you are making sufficient progress to succeed in the project. The project can be broken down into the following steps:

1. Generation, mixing and storing of sine wave signals
2. FastICA implementation and testing via separating the mixed sine waves.

Item (1) will be initially demonstrated and documented in the first deliverable; item (2) along with the entire project will be demonstrated and documented in the final deliverable. Guidelines for the reports will be released separately.

Changelog

- 15-Mar Initial release.

Schedule

Note: the schedule is subject to refinement. Demos are expected to be scheduled at the end of their respective weeks.

1. Mon Mar 25: First deliverable demo week
2. Fri Apr 29: First deliverable report due
3. Mon Apr 8: Final deliverable demo week
4. Fri Apr 12: Final deliverable report due

Grading

- 16% First demo
- 24% First report
- 24% Final demo
- 36% Final report

Sine wave generation

You can generate samples of a sine signal via the formula:

$$s(t) = sin\left(2\pi f \frac{t}{T_s}\right)$$

Here, f is the frequency of the signal and T_s is the sampling frequency, which should be set to 16000 samples/second. Therefore, if you want to generate samples worth 2 seconds of a sine wave, you must generate 32000 samples of $s(t)$, with $t \in [0, 31999]$. You should use the CMSIS-DSP library functions to generate the samples.

To verify that the sine wave is correctly generated, set the frequency to 440 Hz and write the samples to the DAC to hear the sound on the headphones, and compare the sound with the sound of a 440 Hz sine wave via this link. To write samples to the DAC, use timer interrupts with the timeout period equal to the sampling frequency. Note that you will have to scale the signal to have a suitable amplitude for the DAC resolution (8 or 12 bits), and also provide a DC offset to the signal (since the lowest value you can write to the DAC is 0, whereas a sine wave has negative samples). *You may also verify your sine wave is correctly generated by probing the DAC output with the oscilloscope and measuring the frequency, or by printing out the samples and plotting them in MATLAB.*

Once you have verified that your sine wave generation is correct, you must think about how to store the wave. We only have 1 MB of flash memory and 128 kB of SRAM in the MCU, while 1 second of sine wave data requires 64 kB of memory (assuming it is stored as 32 bit floats or integers). To store your generated data, you must make use of the 8 MB Quad-SPI external flash memory on the board. This document will provide you with all the necessary information you need to successfully use the QSPI flash memory.

Finally, create 32000 samples each of two sine waves $s_1(t)$ and $s_2(t)$, each sampled at a rate of $T_s = 16000$ samples/second, and with a frequency of f_1 and f_2 respectively (A pleasant example for frequency values could be C4 and G4 from the table here). Next, create a 2x2 mixing matrix

$$A = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix},$$

in which you may choose the mixing coefficients a_{ij} that are used to mix the signals $s_1(t)$ and $s_2(t)$. Finally, produce the signals $x_1(t)$ and $x_2(t)$, obtained via the matrix multiplication:

$$\begin{bmatrix} x_1(t) \\ x_2(t) \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \begin{bmatrix} s_1(t) \\ s_2(t) \end{bmatrix}.$$

If you play $s_1(t)$ and $s_2(t)$ separately on the left and right headphone, you will notice a clear sine wave tone, while when playing $x_1(t)$ and $x_2(t)$ you will notice that each of them contains two notes that are a linear combination of the original sines. The signals $x_1(t)$ and $x_2(t)$ will be unmixed to via FastICA in the next section to recover the original signals $s_1(t)$ and $s_2(t)$.

Fast Independent components analysis

ICA is arguably the most popular algorithm used for blind source separation. The algorithm basically takes as input the mixed observations $x_1(t)$ and $x_2(t)$, and works to estimate the mixing matrix A (from the previous section) via recursive gradient ascent. This link describes the algorithm and provides examples of an implementation, and should be sufficient to succeed in this task.

Note: *This algorithm can make heavy use of many of the CMSIS-DSP library functions*

Specifications and Tasks to Present

The implementation details are left open to you in order to provide you with a vast array of design choices. Over the course of the project, you will face several hardware and software limitations, due to which you may have to turn to strategies such as DMA, interrupts, CMSIS-DSP, OS threads and synchronization, etc. You must also design the user-interface of your application for playback, recording, and separation. The UI can be implemented via USART, or even the push-button and simple printf's.

The high level deliverables are highlighted below:

Initial deliverable

You must correctly demonstrate that your application is able to:

- Generate sine waves of a desired frequency (via playback on DAC).
- Play and store mixed sine waves.

You should be able to explain and account for all design choices that you have made.

Final project

You must correctly demonstrate that your application is able to perform FastICA BSS on the mixed sine waves generated in the first deliverable, and should be able to explain and account for all design choices that you have made. Note that you will need to provide some justification as to how you have determined that the original source signals have been successfully recovered (eg. Compare original and estimated mixing matrices, measure frequency of unmixed signals via FFT, etc.)