Final Project: Initial Report
Group 3

André Vallières
260742187

Umar Tahir
260681853

Ayden Aamir Malik
260627064

Ismail Faruk
26066352

Karl Godin
260726027

***Abstract*— This document will discuss the core logic and implementation details of the ECSE 444 final project with focus on the design decisions taken and an overview of the project deliverables that have been completed. Work to be done/plans for the future can be found in the section further improvements/results and are not touched upon for the rest of the document.**

***Keywords*— DAC(Digital to Analog Converter), USART(Universal Synchronous/Asynchronous Receiver) Sine Waves, FastICA, BSS(Blind Source Separation), CMSIS(Cortex Microcontroller Software Interface Standard)**

I. Introduction

The final project consists of three components: 1) to generate and play 2 sine waves of a user provided frequency over the two DAC channels to which headphones or a test instrument (oscilloscope) are interfaced. 2) Mix these generated waves with a mixer matrix to obtain a new superimposed note. 3) To use the FastICA algorithm to perform BSS of the mixed signals, and re-obtain the original unmixed signals; this part is yet to be implemented and will be further discussed in the final report.

II. Overview

A simple User Interface (UI) was implemented to allow the user to trigger either of the 4 options. 1) Generate signals. 2) Mix Signals. 3) Play Unmixed Signals. 4) Play mixed signals. Within these options, further user-defined parameters (frequencies, matrix coefficients) are prompted from the user via UART to ensure waves are generated and mixed appropriately. Incase a deadlock/freeze is encountered, the user may press the reset button to restart the User Interface at it's initial state. Furthermore, any user provided parameters and generated/mixed waves are reset including all other on-board peripherals being utilized (QSPI Flash, DAC,USART,RAM etc.)

III. Functional Description

On a Hard Reset of the System, the previous unmixed and mixed signals stored on the QSPI flash are cleared at their predefined starting addresses. Furthermore, the showMenuOptions() function is triggered which prompts the user to select either of the 4 options. If an invalid option is provided, the showMenuOptions() function is triggered again waiting on the user for a valid option. If option 1 is requested, the UI prompts the user for two frequencies. A confirmation message is received and the store_sine() function is invoked twice which generates the respective samples using the CMSIS arm_sine_f32() function . As the onboard SRAM cannot store anything beyond the 2 s worth of samples(takes up all 128 KB of SRAM capacity), a temporary buffer of an arbitrary size is initially written to. After it is filled, the contents of the buffer are written to the QSPI and the buffer is cleared. The write address is determined by the n number of times the buffer is filled and a base address.

If option 2 is requested, the UI prompts the user for the mixer matrix coefficients. After these are provided, these are normalized to ensure that the final mixed wave is not clipped (after multiplication, values exceed 255). The previously generated wave samples are read back from the QSPI flash memory and are used to compute the samples for the mixed wave by the formula given below:

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

The unmixed sine waves s_1 and s_2 are multiplied by the 2x2 coefficient matrix according to the

laws of matrix multiplication. Each coefficient is normalized to be between 0-1. Each sample s_1 and s_2 is read and then multiplied by the normalized coefficients. Similar to how the unmixed wave sample/s generation utilized a temporary buffer, an additional one is prefilled before storing to the QSPI flash. Similarly, the write address for the mixed wave sample/s are determined by a base mixed wave address and the n number of times the buffer is filled.

If options 3 /4 is selected, a message is printed notifying the user of signals being played on both DAC channels. The preconfigured timer (TIM2) is started in interrupt mode and feeds the samples to the DAC at the sampling frequency (interrupts generated after period timeout). In our TIM2 interrupt handler, whenever an interrupt is generated, an index (samples fetched) is used to fetch the appropriate samples from flash memory. These are written to the respective DAC channel. When the samples fetched counter equalizes with the predetermined number of samples, the counter is reset. This way, the 2s worth of mixed/generated waves plays indefinitely until the user presses any key to opt out of playing mode; both DAC channels are set to 0.. Since the DAC does not accept negative values , the samples are offset appropriately to ensure all values are positive.

IV. Parameters

Throughout the lab, multiple parameters had to be set in order to achieve the prescribed behavior in the project definition. Although, not all behavior was outlined in the lab such as the resolution of the signal. A challenge was encountered early on in determining base read/write address locations for each of the 2 unmixed/mixed ways. These had to be chosen so as to prevent any possibility of address overlap at any state of the program. The system offers two options for resolution, 8-bit or 12-bit encoding. Initially, higher precision seemed better. However, the implementation of 12-bit would require much more processing and memory to accomplish. This is mostly due to the word size of flash: 8-bits (byte addressable). Therefore, all interactions with flash and the DAC were done so with 8 bit long integers such as when converting the data and when sending it to the DAC.

Furthermore, a timer was used to send the generated wave over to the DAC. This was chosen to allow UI input while simultaneously sending data. According to the project specifications, the frequency of the generated signal should be 16 kHz. With a clock frequency for the processor of 80 MHz, the desired number of ticks between each timer interrupt was 5000 as shown in [eq. 1].

$$\frac{80\,000\,000\,s}{16\,000\,s} = 5000\ ticks$$

Eq. 1 Calculation for period of timer

Transmission of data was done in two ways, over UART and over pins. More precisely, the DAC signal was sent over the GPIO pins A4 and A5 for stereo output. UART was configured to transmit over the USB connection using a similar way to previous labs. It transmitted on GPIO pins B6 and B7. These are associated to USB output. Furthermore, the baud rate was set to 115200 Bd for optimal transmission speeds.

V. Results

For the given task, our team came through and completed the task within the specifications but we realized some improvements could be made. For example increasing the data size to the DAC from 8 to 12 bits. Reason being that, with 8-bit resolution, aliasing of the sine wave appear to be somewhat obvious so we can step-up that resolution to 12-bits for more accurate recreation of a continuous sine wave. However, as mentioned previously, this would require significantly more processing time and memory space due to the byte alignment of data in flash. Furthermore, the CMSIS library matrices function by also be employed for signal mixing since the CMSIS library functions are much more optimized in most cases. Another improvement could be to increase the sampling rate to create

Also, there can be a slight clipping that happens every 2 seconds. This is caused by the sample being replayed over a 2 second period. If the requested frequency is a multiple of 2, then the signal would mix smoothly at each replay point. However, when it is not, the sample jumps abruptly to zero and causes clipping. A simple fix could be to round to the closest multiple of 2. A linear interpolation could also be done to blend the signal in between each replay.