Microprocessors Final Project: Deliverable #1 Report

Mustafa Khawaja
Electrical Engineering
McGill University
Montreal, Canada
mustafa.khawaja@mail.mcgill.ca

Alex Hale
Computer Engineering
McGill University
Montreal, Canada
alexander.hale@mail.mcgill.ca

Mathieu Bolduc
ECSE
McGill University
Montreal, Canada
mathieu.bolduc@mail.mcgill.ca

***Abstract*—This document details the implementation taken for the first deliverable and the general overview of the system for the ECSE 444 project. The document will explain the rationale behind the design choices taken and the resolution of any difficulties faced along the way.**

***Keywords*—Sine Waves, QPSI external memory, embedded systems, FastICA**

I. Introduction

This deliverable of the ECSE 444 project has four main tasks: (1) Generate two sine waves from a selection of frequencies and write them to both DAC Channels to play through each ear, (2) Mix the two waves into a single output, and (3) Read/Write this new output from the Quad-SPI (QSPI) external memory on the board, and (4) output the signal to earphones or an oscilloscope to verify that the signals were manipulated correctly. Note that these are the tasks for iteration #1 of this project. For iteration #2, the four main tasks are: (1) Receive a mixed audio input from the two input microphones, (2) Store the input to the Quad-SPI (QSPI) external memory on the board, (3) perform Blind Source Separation using FastICA to separate the input into its two individual components, and (4) play each component to an individual earphone. Note that this report concerns only the first deliverable of the project.

II. Overview

The system has one main thread running in which all logic for sine-wave generation and reading/writing to memory is placed. While the board is running, sine waves are generated at the two specified frequencies (musical notes C_4 and G_4). The waves are mixed, and the mixed signal is output to both DAC channels. Currently, QSPI memory is partially implemented. Blind source separation separation with FastICA will be developed for the next deliverable. For timing, the TIM_17 timer was ultimately chosen because it had already been initialized in the provided base project.

III. Design Approach

The sine waves are generated using the arm_sin_f32 functions as described from the lab handout. The input signals are generated at frequencies corresponding to musical notes C_4 and G_4 by following the equation

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

T_s is set at 16 000 Hz to match the sampling frequency of the board microphones, which will be used in the next deliverable. Accordingly, the TIM_17 timer is set to generate an interrupt every 1/16000th of a second. At each interrupt a flag is raised, allowing the next value of the sine wave to be generated. Next, the signals need to be offset and have their amplitude changed to match the DAC output range. After generating each point of the signal, it is increased by one to shift the range of the sine wave to [0, 2]. Then, the point of the signal is multiplied by 2047 to match the DAC 12-bit output range of [0, 4095]. Note that this is the DAC range because 12 bits of signal can hold 2^{12} = 4096 values. While initially the signals were multiplied by 2048, random bits of blank noise were received due to the values exceeding the DAC output range, so the value was lowered to fit the proper range. The sine waves are first tested on a preliminary basis by transmitting to the UART to read the values of the signal. Then, the integrity of the signal is verified using an oscilloscope and by listening to the signal through a pair of headphones..

Once the individual signals are verified to be properly generated, it is time to perform a mixing of the signals. The signal mixing is performed so that in the next deliverable, the FastICA algorithm can be utilized to separate the signals back into their original values. The signals are mixed using a matrix multiplication,

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

This design takes the value of 0.5 for each coefficient of the “A” matrix. This value is chosen to keep the output values, x_1 and x_2, within the output range of the DAC. When performing the matrix multiplication, the maximum possible value of x_1 or x_1 is

$$(0.5 \times 2047 \times 2) + (0.5 \times 2047 \times 2) = 4094 ,$$

where (2047×2) corresponds to the highest possible output from s_1 and s_2. This result is within the maximum value of 4095 that can be output on the DAC. The minimum value of 0 is also respected, because there are no negative values in the matrix and no negative values in s_1 or s_2. Any values could have been chosen for the "A" matrix such that the row would add up to 1, but for simplicity 0.5 is chosen for all.

The matrix multiplication is performed using the arm_mat_mult_f32 function and written to the "X" matrix. x_1 and x_2 are then output to the DAC channels.

An attempt was made to make use of the QSPI memory to save 2 seconds of the sine waves, then play the audio directly from the QSPI memory to the DAC. First, the memory was erased using thet BSP_QSPI_Erase_Chip function. This function was used because the sector size of the BSP_QSPI_Erase_Sector function was unknown. Once the chip is erased, 32 000 samples (2 seconds of signal due to the 16 kHz sampling frequency) of the data in the "X" matrix was written sequentially to memory using BSP_QSPI_Write. Before each write operation, the memory address being written to was incremented by 4 (the size of one word). While testing this section of the deliverable, the 32 000 samples of the signal were saved to QSPI memory before playing any audio to the DAC, so as not to affect the output frequency due to the delay of reading/writing to the external memory. During the writing to QSPI, a wait was implemented to confirm that one operation had successfully completed before starting the next one: `(if(BSP_QSPI_Write(i) == QSPI_OK))`. Once confirmed, an additional 50ms of wait was allowed using "osDelay(50)" before attempting any reads at the location that had been written.

Unfortunately, when reading the values of the QSPI memory using UART, it was found that only the first value of the "write" function would write an actual value to the memory, the rest being zeros. This value was written and read from the first memory slot of the QSPI. All other memory slots could not be successfully read or written. Ultimately, the QSPI memory has not been implemented fully and needs further development.

IV. Future Plans

Since the DAC audio output is not yet perfectly clear, adjustment to the timing of sending values to the DAC will need to be made. It is likely that the logic occuring inside each loop is taking too long and slowing down the output frequency. As much logic as possible needs to be moved outside the output loop, and alternate timing strategies need to be tested.

Once the timing has been solidified, the QSPI read/write implementation needs to be cleaned up and tested with the finalized timing implementation. The difference between erasing sectors and erasing the chip needs to be investigated. The lengthy time of writing to, and particularly reading from the QSPI is anticipated to be a barrier to a successfully-timed DAC output. Another potential set of drivers to look into is the QSPI_HAL drivers included in the base project. The functionality of the HAL drivers will be compared to the BSP drivers. The QSPI "indirect-write" mode will also be investigated to see if it is possible to read and write simultaneously.

The implementation of the FastICA algorithm needs to be completed, in addition to the capturing of input from microphones instead of using internally-generated sine waves.