ECSE 444 Final Project Report

Gauthier, Ian
Computer Engineering
McGill University
260654775
ian.gauthier@mail.mcgill.ca

Misra, Kartik
Computer Engineering
McGill University
260663577
kartik.misra@mail.mcgill.ca

Bieber, Anna
Computer Engineering
McGill University
260678856
anna.bieber@mail.mcgill.ca

Kalia, Paarth
Computer Engineering
McGill University
260662155
paarth.kalia@mail.mcgill.ca

***Abstract* — This report describes the work performed by G01 for the ECSE 444 - Microprocessors Final Project. The goal is to describe in a clear and concise manner the work process to obtain the implementation. The project was based around the creation of a pair of sine waves, saving them to off chip memory, mixing them to create a new pair of waves and then performing a FastICA method to reproduce the original signals before outputting them to an oscilloscope for verification.**

I. Introduction

The goal of the project is to build an audio application that employs Blind Source Separation using the Fast Independent Component Analysis (FastICA) algorithm. To build such an application the solution was divided into multiple tasks:

- Two sine wave generations
- Store and read into flash memory
- Mix the signals
- use the FastICA algorithm to unmix and obtain original signals

The three first tasks were implemented for the initial demo, the last task was implemented for the final demo. The first tasks will be explained briefly, while the last task will be explained in more detail. Generation of samples of a sine signal was accomplished by using the following formula

$$s(t) = sin(2\pi f \frac{t}{f_s})$$

Two seconds of a sine wave were generated, using the CMSIS-DSP library, and sent to the DAC to verify the accuracy of the sine wave frequency by listening to the output. In addition, an oscilloscope was used to verify visually the accuracy of the frequency we were attempting to output -- 440 Hz. Once these tasks were accomplished, they were to be stored into the 8 MB Quad-SPI external memory on the board as 1 MB flash memory and 128 kB of SRAM in the MCU were not enough to store the 4 seconds worth of sine wave output. Once stored into the QSPI external memory the two sine waves were read from memory and mixed using a matrix. These were then sent to the FastICA algorithm in MATLAB through the use of serial communication (UART) to unmix the signals and then sent back to Keil and outputted to the DAC. The target was to recover the original two signals using FastICA after they had been mixed using the mixing matrix.

II. Description of Solution

A. Overview

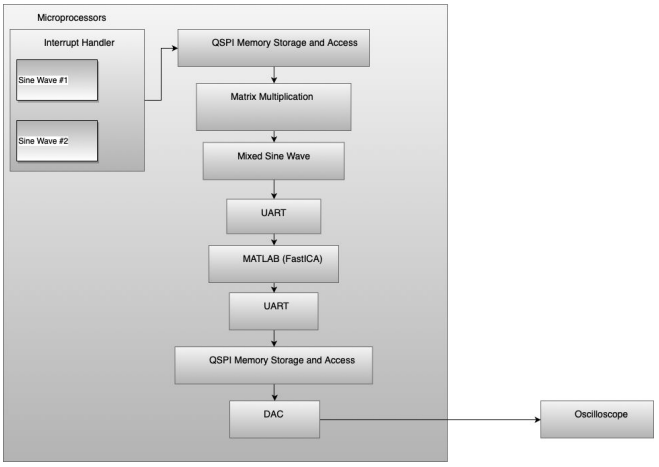

Figure 1: System Block Diagram

At a high level, the implementation of the system worked as follows: the first step is the initialization of the systems that are needed for the successful running of the algorithm - DAC, UART and Systick. The system will then generate the two sine waves of frequencies 261.63 Hz and 392.00 Hz with before storing them into the QSPI memory. Once the waves have been fully generated, the system will retrieve them from the QSPI memory and mix them. This will be followed by sending the signals to MATLAB, through the use of UART, where FastICA can be performed to unmix the signals before returning the unmixed versions of the signals to Keil, through UART once again, where they can be output to the DAC to check that they have been properly reverted to their original values; the oscilloscope is also used to visually inspect the frequencies of the unmixed signals.

B. Microprocessors Components Utilization

The following components were initialized to implement the system as detailed in the previous section. The interrupt handler was used in order to correctly sample the sine waves at the given frequencies. The QSPI memory was utilized to store the original sine waves prior to mixing them and sending them to MATLAB. Both of the DAC channels (D7 and D13 on the STM32L4 Discovery Kit IoT node) were initialized and utilized in order to be able to output data to either headphones connected to the STM32 board or the oscilloscope for the purpose of both testing and observing the final output. The UART was used in order serially communicate with the MATLAB code and vice versa. This MATLAB, in turn, was used in order to perform the FastICA calculations.

C. Implementation

The flow of the system in detail is as follows. First, the system will initialize all of the necessary components for the system using Keil (supra A). From here, the algorithm will erase the QSPI memory on a sector by sector basis for all 256 sectors in the memory. Then, the system will begin the signal creation process. The store function method will be called and for each of the 32000 necessary samples the system will check to ensure that the QSPI is not busy before creating the two sine waves (one at each of the given frequencies) and writing them each to the QSPI memory.

Once all of the samples have been created and thus, signals are fully formed, the process of mixing the signals can begin. The system initializes the mixing matrix so as to create two new variables to hold the mixed signals in SRAM memory. Once this has been performed, the system should again check that the QSPI is ready to be interacted with and then should read the QSPI to obtain both of the signal values for the current sample. Once these have been obtained, the system should perform the signal mixing calculations to obtain the two new signals. Then, these should be returned to the QSPI memory for later use. This action should be done for each of the 32,000 samples and will result in the system having both of the full mixed signals in the QSPI memory.

Furthermore, with the two mixed signals in QSPI memory, the Fast ICA Algorithm comes in. The goal of this algorithm is to find the mixing matrix through different preprocessing methods and then a fixed point algorithm. The chart flow can be seen in *Figure 2* below. There are multiple components to this algorithm, the main ones are the following: removing the mean, calculating the PCA (Principal Component Analysis), whitening the data and calculating the ICA. There are some additional mathematic calculations that is performed as well.

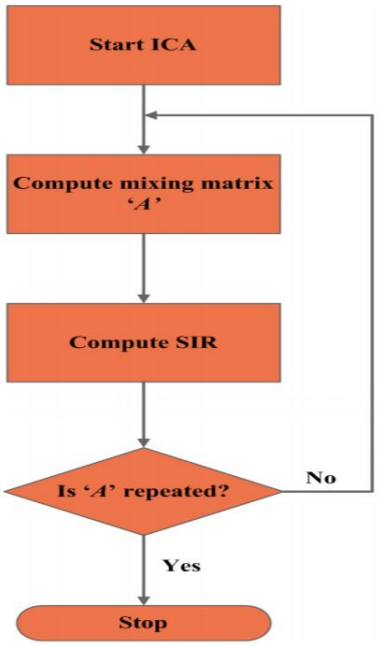

Figure 2: Multi Run ICA Mixing Matrix Computation Flow Chart

The first part is to remove the mean from the vector which consists of the two mixed signals and all of the 32000 samples. This is done by calculating the mean of the vector, then removing said value from the vector and replaced by a one. The vector is then returned as an output. Before performing any additional calculations, the size of the vector is verified to ensure that no space has been lost in the process.

Then, the PCA needs to be calculated. The PCA consists of calculating the covariance matrix to determine the eigenvectors and eigenvalues of the vector. The covariance matrix consists of variances elements appearing along the diagonal and the covariances appear in the off-diagonal elements. With this covariance matrix, the eigenvalues and eigenvectors are calculated and returned as two matrices.

Furthermore, the eigenvectors and eigenvalues are used to calculate the whitening and de-whitening matrices. The whitening matrix consists of the inverse of the root square of the eigenvector matrix multiplied by the complex conjugate transpose of the eigenvalues matrix. The whitening matrix is then used to whitened the data of the mixed signals vector, this is simply done by multiplying the vector by the whitening matrix. The new vector, the whitening and de-whitening matrices are then returned.

Finally, the ICA is calculated with a fixed point algorithm, but before the size of the vector may have changed during the PCA calculations. The fixed point algorithm is fairly complex, thus the details will be omitted. The goal of this algorithm is to find the original frequency of each sample through an initial guess which generates a sequence of improving approximate solutions for the whole vector, in which the n-th approximation is derived from the previous ones.

Once the FastICA algorithm is completed, the vector consists of all the samples of the two unmixed signals, which were then sent through the DAC to confirm the correct frequency of the two signals. The expected output is the two sine waves with frequencies around 261.63 Hz and 392.0 Hz.

III. Results

A. *Results from initial demo*

The results from the initial demo were as follows: the system was able to generate two sine waves, mix them using matrix multiplication, and write to the memory. The sine wave generation was confirmed to be working by outputting the two signals to the DAC where the sine waves were observed and the values of 293 and 395 were recorded for the two output sine waves. These results were also cited in greater detail in the previous report.

B. *FastICA in C*

As the full implementation of the algorithm was not completed in C, there were no apparent results. However, the function to remove the mean was functioning as well whitening the data. These results were concluded to be correct after properly debugging the functions and comparing to the results obtained in the equivalent functions in MATLAB.

C. *FastICA in MATLAB*

The implementation of receiving data over UART from keil was successful and resulted in the following graphs in MATLAB. *Figure 3* represents the sine wave signal of frequency of 392 Hz, *Figure 4* the sine wave of 293 Hz.

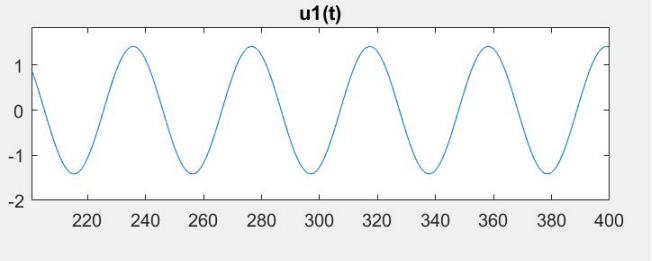

Figure 3: Signal 1

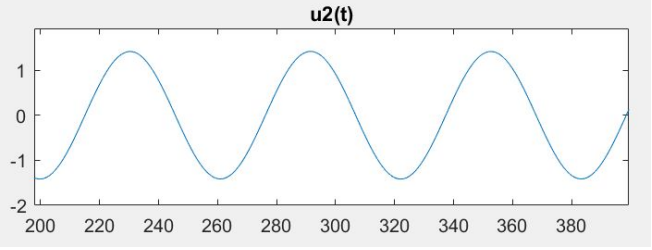

Figure 4: Signal 2

These samples were then sent to Keil through UART, however, they were unable to be processed correctly so as to be able to see results on the oscilloscope. Therefore, no results were observed.

IV. Encountered Problems

Throughout the project, a few problems were encountered. The first issue, was reading from the memory at the correct frequency. For the first demo, threads were being used; however, performing the reading and writing in this way affected how the memory was being accessed. For this reason, the team decided that the use of threads for this application was not a prudent decision and that portion of the solution was removed. This solved the issues found with using the QSPI system and the team was able to move onto the next task.

The second issue came when the team was attempting to implement the FastICA algorithm in C. This involved a great deal of research time as the only resource initially provided to students to understand the theory behind the algorithm was a set of slides on the general idea behind the algorithm. Beyond this, FastICA required a great deal of mathematical knowledge in order to be properly coded. Specifically, the team would have needed to be well versed in linear algebra which is not a field where the team was highly knowledgeable as a whole.
However, when the MATLAB version of the algorithm became available, the implementation became clearer. With the help of the slides, the team was able to port over some of the methods necessary for a full implementation into C code. The problem arose, however, that C was not as well suited for the implementation of this particular program as MATLAB is, which is able to perform vector and matrix assignments and calculations that C is not able to do succinctly. As a result, though some of the implementation of a C based FastICA was possible, it was not feasible for the group to migrate the full system from MATLAB. Once the group ran into the more mathematically intensive portions of the FastICA algorithm, they were unable to continue their implementation in a timely manner. With this in mind and the demo date approaching, this solution was abandoned and another was discussed and implemented.

The group made the decision of attempting to transmit the signals over UART to and from MATLAB. This proved to be more difficult than the team anticipated and they did not allow for enough time before deciding to use the MATLAB algorithm rather than creating one in C. Due to this error in time management, the overall result of the project was an unfinished implementation in which the output was only able to be viewed in MATLAB rather than through the Keil IDE.

Finally, our last issue was one that we couldn't properly diagnose in time for the final demo. As mentioned earlier, we successfully transmitted the signal from Keil to Matlab in order to perform the required computations. However, we were unable to retrieve the separated signals afterwards. We implemented a fastICA_Matlab_Recieve() function that uses scanf() to "capture" the incoming signal through the COM serial port, then we called BSP_QSPI_Write() to save the data to flash memory, after

which we could, in theory, pass the values onto the DAC for listening. When calling fastICA_Matlab_Recieve(), we noticed mixing the signals stopped working. With the help of our TAs we found out that the fprint() method used to transmit the data operated at a much higher speed the the scanf() used to receive, therefore we sent an arbitrary message signaling that the data was ready to be received and implemented an if statement that checked that message before scanning and storing the data. This fixed our mixing problem however we were still unable to write to memory and transmit to DAC successfully, and had run out of time to diagnose the issue. Our best theory as to why this specific part is failing is that we are storing and reading the data in one go. Had we tried to read/write block by block, we may have been successful in achieving the desired result.

V. Moving Forward

The main issue that arose when attempting to implement the system as it was detailed in the initial project description was the scope and magnitude of implementing FastICA in C. This led to a significant amount of difficulty in that the team had to choose between attempting to complete Fast ICA in C or using the provided MATLAB specification of Fast ICA to make a functional product that did not completely adhere to the description. The group ended up doing a bit of both of these approaches which resulted in an unfinished final product. It would likely have been smart for the group to ensure that they were able to complete the version of the program using the MATLAB FastICA before beginning any work on the C implementation as this would have left them with a working product for the code even if it was not perfectly within the specifications. This helped the team understand that if this was to be pitched to a customer, a working product, albeit it with changes to the specifications the customer provided, is more beneficial than showing an unfinished product in terms of the original specifications and the altered specifications.

VI. Conclusion

The goal of this project was to build an audio application using the Fast Independent Component Analysis. The group attempted, without success, to complete this application.

However, we have gained insight and understanding of the STM32 board and Keil software. This includes the skill of debugging a Keil program and understanding the usage of the different drivers of the board as well as the different components of memory. The group considers the lessons that were learned through the failure to be a be a bigger success than completing the application would have been

.