868H1: Digital Signal Processing Laboratory
Project Assignment

Candidate Number: 218831

Due date: 26/05/2019

Contents

CONTENTS 2
INTRODUCTION 3
ARM CORTEX-M4 3
CYPRESS CORTEX-M4 STARTER KIT 3
PROJECT DESCRIPTION 4
SYSTEM DESIGN 4
SYSTEM MODEL 5
SIMULATION 6
FILTER DEVELOPMENT 6
PROGRAM DEPLOYMENT 7
RESULT 8
COMMENTS AND CONCLUSION 8
APPENDIX 9
REFERENCES 9

Introduction

This report describes the development, simulation, and deployment of a DSP-based word recognition system. Besides, sections on describing the ARM Cortex-M4 architecture and its benefits for DSP applications and the used hardware are included.

The Keil project for the microcontroller implementation and MATLAB files for design and simulation is available in the Appendix. Although this report focuses on the DSP part of the project, the fully commented description can be found in the Appendix.

ARM Cortex-M4

The ARM Cortex-M4 processor is the middle-end processor in the Cortex-M family, being this an upgrade to the Cortex-M3. The Cortex-M4 is based on the ARM7-M architecture and has every feature of the Cortex-M3 processor while adding a Floating-Point Unit and support for the extended DSP instruction set [1].

These two additions make developing DSP applications more accessible and less computational demanding for the processor, as the Cortex-M3 processor can also do Floating-Point arithmetic and can also perform the DSP instructions but in after many more cycles; while the Cortex-M4 can do them in a single cycle.

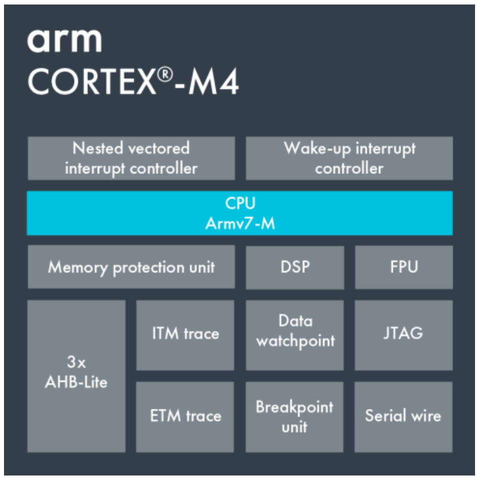

Figure 1. Cortex-M4 High-Level block diagram. [1]

Although the Floating-Point unit is available for every workflow, the DSP instruction set is constrained for Assembly development only [2]. Due to this limit, ARM has developed a set of functions for DSP and general math optimized available in their Cortex Microcontroller Interface Standard (CMSIS) software package. The CMSIS includes functions the general Cortex-M architecture, debugging, Real-Time Operating System, Neural Networks and DSP. The DSP library collection consists of over 60 functions for various data types going from basic math to several digital filters. [3]

Another feature useful for DMA applications is the Direct Memory Access peripheral (DMA). The DMA unit allows transferring data segments from one peripheral to memory without going through the processor. This method decreases the required computational power to fetch store and process the data by leaving the fetching and storing to the DMA and also increase the time between interrupts.

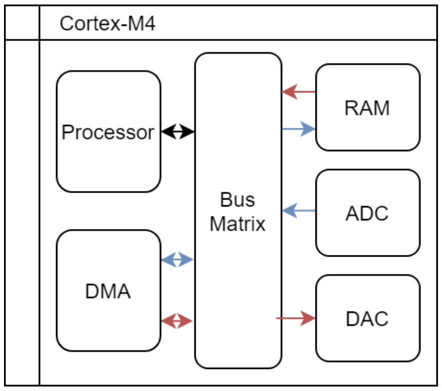

Figure 2. Data flow using DMA. The blue arrow shows the data flow from the ADC to RAM, and the red line shows the dataflow from RAM to DAC. Both paths go through the DMA unit while the processor only accesses the samples directly in RAM.

Cypress Cortex-M4 Starter Kit

As Cortex-M4 is only a processor type based on the ARM architecture, the actual manufacturers of microcontrollers are the third party such as Texas Instruments, ST Semiconductor, NXP and Cypress. Often, these manufacturers make their microcontroller families in the form of development boards, which are meant to aid the development cycle of a product by offering a processor with all the required components for it to work and many external peripherals. Also, manufacturers provide examples, schematics and design notes for them to replicate the working principle of the development board.

The development board used in for the project is the FM4-S6E2CC-ETH ARM Cortex-M4 starter kit from Cypress. This development board is targeted for DSP applications due to the available 3.5 mm jacks and the onboard codec ADC/DAC IC but also supports network-related and USB host applications. Also, the board has a CMSIS DAP debugger to flash and halt the program execution in order to solve any on-going issues correctly. The board is based on the S6E2CCAJ0A microcontroller from Cypress, which has the following features. [4, 5]

- 200 MHz Clock
- 2MB Flash Memory
- 256 KB SRAM
- 2 USB-OTG, CAN, I2C, I2S and SPI
- DMA and Floating-Point units
- 12-bit ADC and DAC

This microcontroller is more than enough to perform advanced DSP algorithms due to the increased clock speed and RAM size. Besides, the onboard WM8731 audio codec IC allows a more straightforward integration for the input and output signals as the little pre-processing is needed, and the output signal has enough power to drive a pair of headphones [6]. The coded supports multiple resolutions levels from 12 to 32 bit for both ADC and DAC, and the sampling rate can go from 8 kHz to 96 kHz [7]. The IC is interfaced with the microcontroller using both the I2C and I2S protocols for configuration and data retrieval, respectively. [8]

The recommended development environment is ARM's Keil Microcontroller Design Kit. This IDE is the most common tool for programming and debugging ARM-based microcontroller since it can download the required startup files and Hardware Abstraction Layer drivers.

Project Description

The intended project is a DSP based system able to distinguish between the words Yes and No; no other words are considered. The system is intended to be deployed in the previously described development board, using the DSP algorithms reviewed previously during the lectures.

Several test audio files were provided in order to test the program using the line input of the development board and the headphones output of a computer. These files are intended to have different pronunciation, tone, and pitch so the intended system must be capable of successfully categorize these impairments.

The development of this system will be split into two categories, the model-based design and the hardware deployment.

System Design

The signal's characteristics must be determined in order to decide what DSP approach to use. It is known that the letter "S" in "Yes" has a high-frequency band component while keeping the low-frequency band from the same syllable, while the word "No" has only a low-frequency band component.

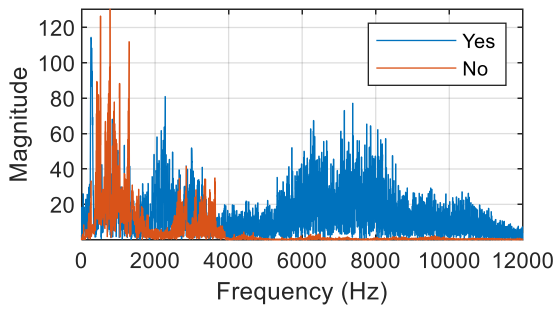

Figure 3. The frequency response of Yes and No files.

The frequency response shown in Figure 3 is one of the several comparisons made for choosing the boundary of the low-frequency component in "No" and the high-frequency component in "Yes". Ultimately, these frequencies were chosen as 2500 and 4500 Hz, respectively, with a stopband frequency difference of 1000 Hz for both boundaries.

In order to determine which was the word, the ratio between the high and low energy spectrums must be computed. Calculating the ratio instead of only the high band power spectrum allows normalizing the input data. Normalizing the data means that a higher volume speech will not trigger a false-positive result if a “No” is spoken at a high volume, and false-negatives if a “Yes” is whispered.

$$r(t) = \frac{RMS\left(HPF\left(x_0(t)\right)\right)}{RMS\left(LPF\left(x_0(t)\right)\right)}$$

Equation 1. Energy ratio.

Where x_0 represents the incoming signal from the DMA peripheral of length greater than 1, the HPF and LPF are the filters with previously mentioned characteristics and RMS is the Root-Mean Square operation. Once the ratio is obtained, it must be compared to an arbitrary threshold to decide whether the incoming signal represents “Yes” or “No”.

$$x_7(t) = \begin{cases} Yes, & r(t) > r_{Threshold} \\ No, & r(t) < r_{Threshold} \end{cases}$$

Equation 2. Yes/No condition.

Although this work is enough to decide between “Yes” and “No”, a volume threshold is needed to avoid outputting results when no sound is detected. This threshold is going to be obtained considering the RMS value of the incoming signal while silent, staring the speech and at the middle of the word. The silent output will be named “Not Available” (NA).

$$x_8(t) = \begin{cases} x_7(t), & RMS\left(x_0(t)\right) > RMS_{Threshold} \\ NA, & RMS\left(x_0(t)\right) < RMS_{Threshold} \end{cases}$$

Equation 3. Volume threshold condition.

This algorithm is enough to distinguish between words while having a volume threshold properly, although the microcontroller program will need some post-processing to keep and display the result in order for the user to see the output result.

System Model

Simulation tools as MATLAB/Simulink are essential for the design cycle of an embedded product. Since this type of systems often needs cross-compilation tools, testing and scripting can be difficult, as doing this in the actual hardware take time. Using a simulation environment as the previously mentioned can accelerate the development process, as the algorithm can be tested first and then realistic features such as ADC impairments and DMA buffering can be implemented latter to increase the model’s fidelity.

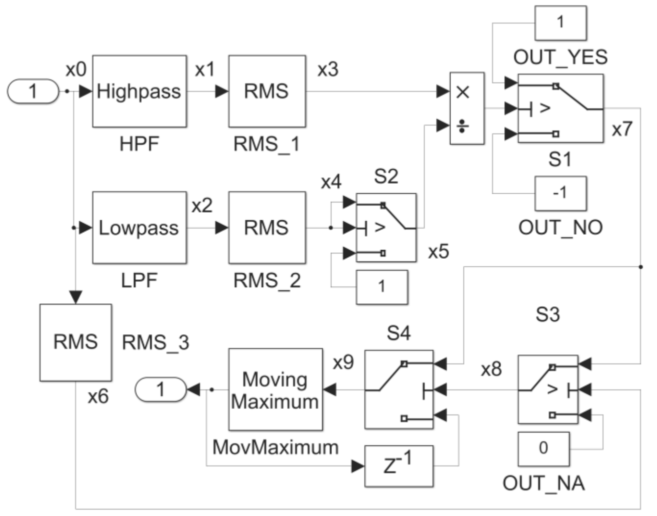

Figure 4. Simulink System Model.

The model shown in Figure 4 includes the algorithm shown in Equation 1, Equation 2 and Equation 3; the signal names are the same as the ones used in the same equations. Besides, an output stage and if statement were added to process the output and avoid division by zero, respectively. As it is not visible, the S1 switch block comparison constant is the Energy ratio threshold, for both S2 and S3 is the input RMS threshold, and S4 evaluates x_8 being different from zero.

A moving maximum block using a buffer size named window length is used to prioritize “Yes” results over “No”; if a single “Yes” is found in the buffer, the overall output will be “Yes”, until the buffer shifts that value out of it. Before this block, an if statement (S_4) is used to keep the output as NA if the signal x_8 is 0 depending on the input volume. Once the x_8

signal is non-zero, the output will not return to NA, and the reset button on board will set the indicators to NA again.

Simulation

The model in Figure 4 is instantiated in a simpler higher-level model, only including an audio input from a media file and output sinks to store and visualize the result. The audio input plays one of the provided test files at a given rate of 24 kHz and while outputting more than one frame per sample to emulate a DMA based buffer.

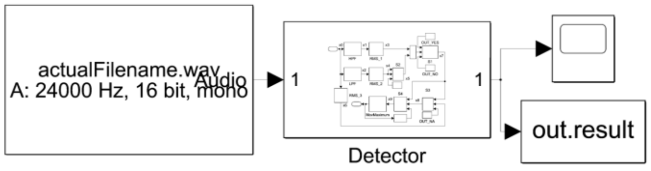

Figure 5. High-Level model.

Since Simulink decodes the audio file from Pulse Code Modulation (PCM) to voltage, the signal processed by the system will be more accurate.

This model helped to find the threshold values since the signals can be evaluated in real-time and trial and error approximation is much faster than doing it in the actual hardware. On top of that, a helper function was coded to run the model in Figure 5, using all the provided test files to test several thresholds and window values quickly. Ultimately, $r_{Threshold}$, $RMS_{Threshold}$ and the window length were set as 0.8, 0.006 and 22, respectively. Also, the DMA block size was set to 128 elements. These are the parameters that result in the least number of errors for both "Yes" and "No" files. Figure 6 and Figure 7 show the output signals for both "No" and "Yes" files, respectively, from the model in Figure 4 it can be inferred that 1, -1 and 0 mean "Yes", "No" and NA, respectively.

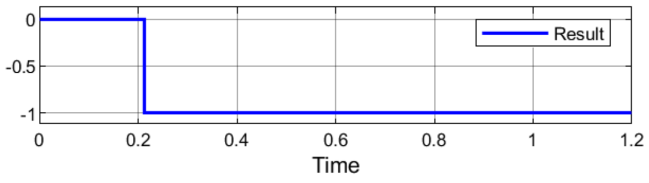

Figure 6. Output signal using a "No" file.

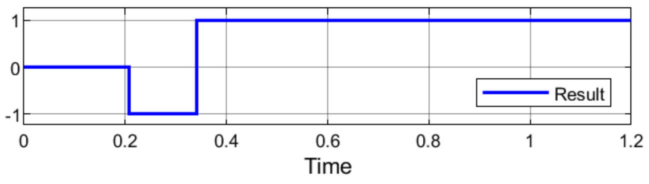

Figure 7. Output signal using a "Yes" file.

With the model working and the parameters defined, the last step is to code this algorithm into the microcontroller. Simulink has the feature of exporting C/C++ code and even using the CMSIS DSP functions, but it is significantly inefficient and lengthy, which are significant drawbacks because this model is straightforward.

```
Result for y1 is correct -> 1
...
Result for y25 is incorrect -> -1
Result for n1 is correct -> -1
...
Result for n24 is incorrect -> 1
Total errors for YES detection -> 2/25
Total errors for NO detection -> 1/24
Total errors -> 3/49
```

Figure 8. Output example for the model tester.

Filter Development

As stated before, the system model requires a Low-Pass filter and a High-Pass filter, with defined passband and stopband frequencies. Filter details such as sampling rate and architecture and design principle are described in the following paragraphs.

The sampling rate of the filter is a fundamental part of digital filters. As stated in the Nyquist Theorem, the sampling rate must at least double the highest frequency of the sampled signal to avoid aliasing. Often, a low pass filter is added in the signal acquisition stage in the analog domain to reduce the maximum sampled frequency; but this feature is handled by the coded IC. For this scenario, the sampling frequency must be supported by the audio codec IC and high enough to get all the features the signal has. The supported frequencies are 8, 32, 44.1, 48, 88.2 and 96 kHz [7]. The audible spectrum goes from 20 Hz to 20 kHz, but the average spectrum used in speech is in the order of hundreds of Hz; also, the high-frequency component is meant to be obtained by the system; hence, a reasonably high sampling frequency is needed, but not high enough to cover the entire audible spectrum. As seen in Figure 3, the most significant difference in the frequency response is found below 10 kHz, so analyzing to spectrum above that is unnecessary. These conditions resulted in choosing the sampling frequency as 32 kHz.

The second consideration is the architecture, referring to the impulse response type of the filter. As known, IIR filters require a lower filter order to achieve the same performance as its FIR counterpart, but some issues as instability and phase delay might occur. Since the drawbacks of using an FIR filter are not demanding for the microcontroller due to the increased clock speed, RAM and ROM; this is the selected architecture.

Both High-Pass and Low-Pass filters were designed as equiripple FIR filters considering the previous decisions using MATLAB and the filter design tool, which resulted in two 89th and 82nd order filters. The filter coefficients were exported and converted into a CMSIS compatible C header file.

The filters were implemented and tested fist on the microcontroller using the CMSIS DSP library, which enabled a quick implementation by only specifying the coefficients array, Number of taps (Filter Order + 1 for FIR filters), a status buffer for previous results and the input block size. [3]

A test application was coded in order to get the real filter response. It consisted of a PRBS signal source inputting samples to both filters and then outputting the filtered signal through the headphones jack. Unfortunately, the lack of lab equipment restricted the data acquisition alternatives; however, the output signal was captured by a microphone connected to a PC using a higher sampling frequency.

As seen in Figure 9 and Figure 10, some features of the frequency responses as the cut-off frequency and the filter order slope. However, the passband region has a significant gap between the real and measured responses in some frequencies, and the real stopband attenuation is not as low as the real. Again, this is because of acquiring the signals using a microphone and not a dedicated line input to the PC.

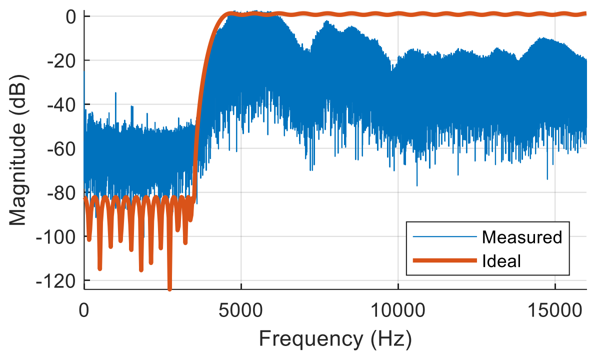

Figure 9. High-Pass filter response. Ideal vs Real.

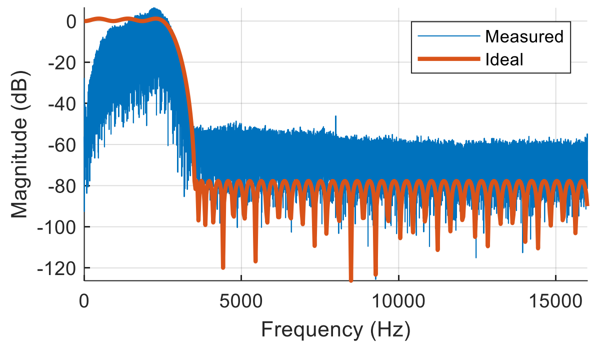

Figure 10. Low-Pass filter response. Ideal vs Real.

Program Deployment

The program was coded into a provided Keil project which has the required drivers to use the codec and DMA and an example source file that only requires to implement the DSP algorithm. The CMSIS DSP library was used to get the filters working from the Simulink Model, and the definitions of the signals x_m in the same model were represented as a single line of code, each. The code representing the model can be seen in Script 1.

The threshold parameters obtained in the Simulation section were used as macros to fulfil the same purpose. As the window length and energy ratio threshold units are the same in the model as in the implementation, these can remain as they are. However, the RMS input threshold must be translated from volts to PCM again to work with the raw samples the ADC gets. The relation is done using the number of bits and the reference voltage of the ADC.

$$RMS_{PCM} = \frac{2^n}{V_{ref}} RMS_v$$

Equation 4. RMS volts to PCM equation.

The ADC is configured (In the provided drivers) to have a 16-bit resolution, and according to the development board schematic, the reference voltage is 3.3 V. This combination results in an RMS threshold of ~120. The following table shows the relation between the macros and the model parameters, while other macros such as the system output signals are defined but not included in Table 1.

Parameter	**Macro Name**	**Model Value**	**Macro Value**
Input RMS threshold	*InputThreshold*	0.006	120
Energy Ratio Threshold	*YNThreshold*	0.8	0.8
Window Length	*MaxWindow*	22	22

Table 1. Parameter macro definitions.

The implementation of the model algorithm into the microcontroller is shown in Script 1; other instructions such as variable declarations and filter initializations are not shown but were required.

The resulting signal is displayed to the user in the form of light using the onboard RGB LED. The colour code is red for "No" and green for "Yes", while NA is represented by LED being off. As mentioned before, the systems do not go back to NA once the algorithm is triggered, so the reset button on board is meant to go back to that state, but this is not necessary as several successive words can be analyzed without going back to NA.

```
// Each line translates to a block
arm_fir_f32(&HPS, x0, x1, DMA_BUFFER_SIZE); // HPF
arm_fir_f32(&LPS, x0, x2, DMA_BUFFER_SIZE); // LPF
arm_rms_f32(x1, DMA_BUFFER_SIZE, & x3);     // RMS_1
arm_rms_f32(x2, DMA_BUFFER_SIZE, & x4);     // RMS_2
arm_rms_f32(x0, DMA_BUFFER_SIZE, & x6);     // RMS_3
x5 = (x4>InputThreshold)? x4 : 1;           // S2
x7 = (x3/x5>YNThreshold)? OUT_YES: OUT_NO;  // S1
x8 = (x6>InputThreshold)? x7 : OUT_NA;      // S3
x9 = (x8) ? x7 : y[1];                      // S4
y[0] = movingMax(x9);                       // MovMax
y[1] = y[0];                                // Delay
```

Script 1. Code snippet representing the Simulink model.

Result

The compiled binary utilizes 5.4 kB for the program, 1.3 kB for storing read-only data (Coefficients and macros) and 7.3 kB for allocating variables in the RAM. Compared to the capacity of an Arduino Uno, the RAM is insufficient, as it only has 2 kB; hence, it would not be able to run this solution.

The program was tested using the line input on the development board to avoid clipping, as the microphone input has a 20 dB gain amplifier. When first ran, the code had an error ratio of 7/49 files, not as low as the model implied. Nevertheless, after changing the Energy ratio threshold from 0.8 to 0.7, the error ratio lowered to 4/49 files.

Some files have more definitive attributes than others, for example, a "yes" file with a long and exaggerated "S" sound or another "yes" file with the "S" being omitted. The second example triggers several false negatives, while the first has a behaviour close to the one shown in Figure 7. Another impairment is a high-frequency component caused by the PC just after playing the audio file, and this triggered many false positives. The program did work in real-time by saying yes and no by bypassing the PC microphone input to the board's line input.

Comments and Conclusion

As stated before, this solution fulfils the task, but many other alternatives could be used. The use of an

RTOS may simplify the DMA handler and efficiently handle the idle time between processes. Increasing the DMA block size would decrease the number of interrupts and have more idle time to do some other tasks. The given template project used a 16-bit ADC, but by tweaking the source files a 32-bit ADC can be configured, although not every sample rate might be supported. Also, a higher-order IIR biquad filter may have a sharper response, and the passband frequency could be tuned to be more discriminant.

Some other approaches as adaptive filters can be used, but several more testing and design parameters must be considered.

Appendix

The files used for the model design in MATLAB/Simulink, filter response audio files and Keil project are available in the following repository.

Every code in the repository is fully commented and describes the functionality of the program rather than focusing on the DSP part, as this report did.

https://github.com/JoseAmador95/DSP_YN_Detect

References

[1] ARM, "Cortex-M4," ARM, 2020. [Online]. Available: https://developer.arm.com/ip-products/processors/cortex-m/cortex-m4. [Accessed 16 May 2020].

[2] ARM, "ARM Cortex-M4 Processor Technical Reference Manual Revision r0p1," ARM, 2020. [Online]. Available: https://developer.arm.com/docs/100166/0001. [Accessed 16 May 2020].

[3] ARM, “CMSIS 5 Github Repository,” Github, May 2020. [Online]. Available: https://github.com/ARM-software/CMSIS_5. [Accessed 16 May 2020].

[4] Cypress Semiconductor, “FM4 S6E2C-Series High Performance Arm® Cortex®-M4 Microcontroller (MCU) Family,” 26 March 2018. [Online]. Available: https://www.cypress.com/file/235126/download. [Accessed 16 May 2020].

[5] Cypress Semiconductor, “FM4-176L-S6E2CC-ETH - Arm® Cortex®-M4 MCU Starter Kit with Ethernet and USB Host,” 13 May 2020. [Online]. Available: https://www.cypress.com/documentation/development-kitsboards/sk-fm4-176l-s6e2cc-fm4-family-quick-start-guide. [Accessed 16 May 2020].

[6] Cypress Semiconductor, “FM4 Starter Kit Guide,” 26 March 2015. [Online]. Available: https://www.cypress.com/file/290916/download. [Accessed 16 May 2020].

[7] Wolfson Microelectronics, “WM8731 Datasheet,” October 2012. [Online]. Available: https://statics.cirrus.com/pubs/proDatasheet/WM8731_v4.9.pdf. [Accessed 16 May 2020].

[8] Cypress Semiconductor, "FM4-176L-S6E2CC-ETH Schematic," 21 August 2014. [Online]. Available: https://www.cypress.com/file/290921/download. [Accessed 16 May 2020].