Performance Evaluation of Digital Signal Processing Architectures Running an Acoustic Echo Cancellation Software

Joel Viklund

Joel Viklund
Spring 2016
Master's Thesis in Computing Science, 30 ECTS
Supervisor: Mikael Rännar
Examiner: Henrik Björklund
Department of Computing Science, Umeå University
Abstract

TrueVoice is a speech enhancement software by Limes Audio AB. It improves the quality of the speech signal by using methods such as acoustic echo cancellation and noise reduction. The real time signal processing of TrueVoice puts some specific requirements on the hardware, which makes some processors more suitable than others to run the code.

This thesis looks for good digital signal processor candidates to port the TrueVoice code to. This is done by evaluating memory and number representation aspects in theory and processing power in practice. The conclusion is that Cirrus WM8281 is a good candidate to port TrueVoice to.
Acknowledgements

I would like to thank my supervisor from Limes Audio, Emil Lundmark, for the guidance and valuable feedback throughout the whole process. I would also like to thank Christian Schüldt for sharing his knowledge about the TrueVoice implementation and Fredric Lindström for letting me do this project for Limes Audio. Thanks to Magnus Berggren, Markus Lindroth and everyone else at Limes Audio as well.

I would also like to thank my supervisor from Umeå University, Mikael Rännar, for his constructive criticism and good tips on the report writing.

Last but not least, thanks to Eric Sjögren, for always being on my (left) side.
## Contents

1 Introduction 3
   1.1 Background ................................................. 3
   1.2 Purpose .................................................. 3
   1.3 Related Work ............................................. 3
   1.4 Outline .................................................. 4

2 Basic Audio Signal Processing 5
   2.1 Digital Audio .............................................. 5
   2.2 Fourier Transform ........................................ 6
   2.3 FIR Filter ................................................ 7
   2.4 LMS Filter ............................................... 8
   2.5 Acoustic Echo Cancellation .............................. 8

3 Digital Signal Processors 11
   3.1 Architecture Layout ...................................... 11
   3.2 Circular Buffer .......................................... 12
   3.3 Instruction Set .......................................... 12
   3.4 Parallelism .............................................. 13
   3.5 Floating- and Fixed point ............................... 13
   3.6 Cirrus .................................................... 14
   3.7 Tensilica HIFI3 DSP ..................................... 16
   3.8 ARM Cortex-A9 ............................................ 16

4 Accomplishment 19
   4.1 Preliminaries ............................................ 19
   4.2 How the Work was Done .................................. 20
5 Results

5.1 Test Program ............................................. 23
   5.1.1 Cirrus ............................................. 23
   5.1.2 ARM ............................................. 24
5.2 Test Results .......................................... 24
5.3 Memory Usage ......................................... 26
5.4 Number Representation ................................. 27

6 Conclusions ............................................. 29

6.1 Discussion ............................................ 29
6.2 Conclusions ........................................... 30
6.3 Future Work ........................................... 31

References ............................................... 33

A Test Results ............................................ 35
   A.1 Relative Execution Times NLMS ......................... 35
   A.2 Relative Execution Times FFT .......................... 36
Chapter 1

Introduction

This chapter gives an introduction to the subject and explains the purpose of the thesis. It also presents some related work and gives an outline of the report.

1.1 Background

TrueVoice is a speech enhancement software by Limes Audio AB. It improves the quality of the speech signal by using methods such as acoustic echo cancellation and noise reduction. To fulfill the needs of their customers, Limes Audio wants to port TrueVoice to new processor architectures. However, the real time signal processing of TrueVoice puts some specific demands on the hardware on which it is implemented. This means that some architectures might be more suitable for TrueVoice than others.

1.2 Purpose

The aim of this master thesis is to evaluate which architectures are the most suitable for TrueVoice by evaluating a couple of different candidates. The candidates might differ in terms of memory, processing power and number representation. The effect of such properties will be evaluated in theory, but also by implementing and running a small test program.

1.3 Related Work

Performance evaluation of digital signal processors are discussed in a publication by Lapsly and Blalock [20]. They conclude that the performance unit million instructions per second (MIPS) is a bad measurement of digital signal processor (DSP) performance since the instruction set between different processors can vary significantly. A better approach is to use algorithm and
application benchmarking which measures the performance of performing a specific task.

1.4 Outline

Chapter 2 contains the theory behind acoustic echo cancellation and basic digital signal processing. Chapter 3 is about the architecture of digital signal processors. Chapter 4 explains how the study was done, Chapter 5 describes the result of the study, which is discussed in Chapter 6.
Chapter 2

Basic Audio Signal Processing

This chapter describes digital signal processing with focus on acoustic echo cancellation. Before going into the echo cancellation itself, it describes the basics of digital audio, Fourier transforms and adaptive filters. These are all important components of the echo cancellation routine that is described at the end of the chapter.

2.1 Digital Audio

A sound wave is an analog signal and must therefore be converted to a digital representation before it can be processed by a computer. The analog-to-digital conversion of a signal is called sampling [18] and is illustrated in Figure 2.1.

![Figure 2.1: Digital sampling of a continuous signal [2].](image)

The first step of sampling is to measure a number of sample points. The number of sample points per second is called the sample rate and determines what frequencies that can be digitized [18]. The Nyquist-Shannon sampling
Theorem states that the sample frequency must be twice the maximum signal frequency in order to correctly sample a continuous signal [23]. If the sample rate is lower than twice the maximum frequency, different signal frequencies can give the same sample values which can lead to the incorrect signal being represented; a phenomena known as aliasing [18].

The next step of sampling is to convert each sample point to a digital representation, called quantisation [18]. The detail of the digital representation is determined by the sample bit depth. A low bit depth increases the risk of high round-off errors, which can lead to audible distortion [24]. One way of silencing the distortion is to increase the bit depth, which means more detailed sample representations and therefore a smaller quantisation error. Another way is to use a technique called dithering. Dithering is about randomly rounding up or down to the nearest quantization level with a possibility that is determined by the actual value [24]. This type of introduced noise is not as easy to hear as the error introduced by constantly rounding to the nearest quantization level [24].

Different applications use different sample rates and bit depths. A human can hear frequencies up to 20 kHz [16], which according to the sampling theorem requires a sample rate of 40 kHz to sample. The human voice however, only reaches frequencies to about 4 kHz [16] which can be sampled at a lower rate. Some common bit depths when sampling are 16 and 24 bits [26].

### 2.2 Fourier Transform

The Fourier transform is an important tool in the digital signal processing toolbox because of its many applications [18]. It can take any signal and split it up into a number of frequencies with different phases and amplitudes (as illustrated in Figure 2.2). The frequencies can be analysed or modified and then put together to a combined signal again with a reversed transform.

![Figure 2.2: Fourier transform from time to frequency spectrum](image)

The formula for a discrete Fourier transform from time spectrum to fre-
Frequency spectrum is

\[ X(k) = \sum_{n=0}^{N-1} x(n) \cdot e^{-i2\pi kn/N} \]  

(2.1)

where \( X(k) \) is a complex number with information about the amplitude and phase of frequency \( k \) and \( N \) is the number of sample points [21]. The reversed transform, from frequency spectrum back to time spectrum has the equation

\[ x(n) = \frac{1}{N} \sum_{k=0}^{N-1} X(k) \cdot e^{i2\pi kn/N} \]  

(2.2)

where \( x(n) \) is the sample value of the \( n \):th sample [21].

An implementation of the Fourier transform according to the definition has a complexity of \( O(n^2) \) multiplications. A group of algorithms called Fast Fourier Transforms (FFTs) has been developed to improve the efficiency of the Fourier transform. They can achieve a complexity of \( O(n \log n) \) instead. A commonly used algorithm is the Cooley-Tukey FFT [19]. It takes advantage of that equation 2.1 can be rewritten as

\[ X(k) = E(k) + e^{-i2\pi k/N} O(k) \]
\[ X(k + N/2) = E(k) - e^{-i2\pi k/N} O(k) \]  

(2.3)

for \( 0 \leq k < N/2 \) where \( E(k) \) and \( O(k) \) is the FFT for the even, respectively the odd, sample values. This property of the equation can be used to recursively break down the FFT into smaller transforms. The combination of the results as shown in equation 2.3 is called a butterfly.

### 2.3 FIR Filter

Digital signals can be modified by applying digital filters on them. There are two types of digital filters: finite impulse response (FIR) filter and infinite impulse response (IIR) filter. The FIR filter is more relevant in this project, and will therefore be focused on.

A FIR filter of size \( M \) outputs a signal \( y(n) \) which is a linear combination of the \( M \) latest samples. The function can be described as

\[ y(n) = \sum_{k=0}^{M-1} b_k x(n - k) \]  

(2.4)

where \( y(n) \) is the signal output, \( M \) is the filter size, \( b_k \) is the \( k \):th filter coefficient and \( x(n - k) \) is the sample value of the \( k \):th latest sample [22]. The behaviour of the filter is configured by the filter coefficients.
2.4 LMS Filter

One way of configuring the filter coefficients in a FIR filter is to use a least mean square (LMS) adaptive filter. The LMS filter adjusts the coefficients of a filter to achieve a low mean square error. The basic concept of the LMS filter is illustrated in Figure 2.3.

![Figure 2.3: The basic concept of the LMS filter [6].](image)

The LMS algorithm calculates an error which is the difference between a desired signal \( d(n) \) and the actual filter output \( \hat{y}(n) \)

\[
e(n) = d(n) - \hat{y}(n). \tag{2.5}
\]

It then updates the filter coefficients with the formula

\[
\hat{h}(n) = \hat{h}(n-1) + \mu e(n) u^*(n) \tag{2.6}
\]

where \( \hat{h}(n) \) is the new filter coefficients, \( \hat{h}(n-1) \) is the previous filter coefficients, \( \mu \) is the step size, \( e(n) \) is the output error and \( u^*(n) \) is the sample input values. The step size determines the behavior of the filter: a large step size means that the filter converges faster but at the cost of becoming more unstable.

It can be tricky to choose a good step size [25]. One way of solving this is to use the normalized mean square (NLMS) filter instead. The filter works in a similar way as the LMS filter but normalizes the update of the filter coefficients by using the formula

\[
\hat{h}(n) = \hat{h}(n-1) + \mu e(n) \frac{u^*(n)}{\epsilon + u^H(n)u(n)} \tag{2.7}
\]

where \( \epsilon \) is a small number that prevents the denominator to becoming to close to zero.

2.5 Acoustic Echo Cancellation

This section describes acoustic echo cancellation as implemented in TrueVoice. The text is based on internal documentation provided by Limes Audio.
Acoustic echo cancellation is about removing unwanted echos from a signal sample. A typical application is a conference system as illustrated in Figure 2.4.

Figure 2.4: A typical application to use acoustic echo cancellation [12].

The far-end speaker signal $x(k)$ is received and played out loud in the room. At the same time, the microphone record a signal which is the combination of the near-end speaker signal and the echoes of the far-end speaker. Without any form of echo cancellation, the microphone signal will be transmitted back to the far-end system which will result in the far-end speaker hearing echoes of himself. The goal of acoustic echo cancellation is to remove the echoes of $x(k)$ from $y(k)$, so that only the near-end speaker signal is transmitted. The basic concept of this process is illustrated in Figure 2.5. The echo of the received signal $x(k)$ can be described as a linear combination of its previous samples and hence be modeled with a FIR filter. The filter coefficients are not known in advance which is why an adaptive filter, such as the NLMS filter, is necessary.
Without any near end signal \( s(k) \), \( y(k) \) will only consist of the echo and the adaptive filter will adapt its coefficients so that \( \hat{d}(k) \) will be close to \( y(k) \). When the near-end speaker starts talking, the adaptive filter needs to stop update its coefficients, so it does not filter out the speaker signal as well. In TrueVoice, this is solved by using two filters: a background and a foreground filter. The background filter is adaptive and continuously updates its coefficients. The actual output is calculated by the foreground filter which updates its coefficients by copying the background filter coefficients. This is only done when the background filter performs a better filtering than the foreground, which is measured by looking at factors such as error signal and echo return loss enhancement.

TrueVoice performs the acoustic echo cancellation on complex subband signals instead of the fullband signal. This approach makes the filtering more stable and reduces the computational complexity but also means that every subband has its own pair of complex foreground and background filters. The characteristics of the filters differs between the high and low frequency subbands. The high frequency filters are shorter and use NLMS while the low frequency filters are longer and use fast affine projection (FAP) adaption. The FAP algorithm is an alternative to NLMS with faster convergence rate but higher computational complexity [17].
Chapter 3

Digital Signal Processors

Digital signal processing puts some specific requirements on the hardware. A general purpose processor has to be good at many things while a digital signal processor (DSP) is optimized for digital signal processing. DSPs must often work in real time and therefore benchmarks, such as throughput and latency, becomes important. This chapter introduces the reader to digital signal processors and then describes the specific DSP candidates.

3.1 Architecture Layout

Most modern DSPs are based on the Harvard architecture [10]. The Harvard architecture has separate instruction and data memory, in contrast to the von Neumann architecture which has a shared memory (see Figure 3.1). The Harvard architecture can therefore utilize two data buses and retrieve instructions and data simultaneously.

![Figure 3.1: The von Neumann and Harvard architecture [8].](image)

Modifications of the Harvard architecture has been made to improve the
performance of the DSPs. The Super-Harvard Architecture stores the latest instructions in a cache. This can give a performance boost when rereading instructions in a loop [10].

3.2 Circular Buffer

A FIR filter is calculated by summing the product of the $N$ latest samples and the filter coefficients. An efficient way to store the latest samples is by using a circular buffer. A circular buffer has a starting address, a stop address, a stride and a position for the latest sample. Each new sample is placed in front of the latest sample. If the latest sample was at the stop address, the new sample is placed at the starting address. This circular behavior is why the data type is called a circular buffer. By following these rules, the oldest sample will always be overwritten by the newest, and no shifting of the values are required. A circular buffer of size 8 is illustrated in Figure 3.2. The next value will be written into memory address 20046, overriding the oldest value $x[n-7]$.

![Figure 3.2: Example of a circular buffer [24].](image)

In DSPs, circular buffers are often implemented at hardware level [24]. This means that no software overhead needs to be added to manage the buffers which is good in terms of efficiency.

3.3 Instruction Set

The instruction set of a digital signal processor is optimized to suit the needs of the signal processing algorithms. One of the most common operations in signal processing is the multiply-accumulate (MAC) operation

$$X = (A \cdot B) + X$$  \hspace{1cm} (3.1)

which is why many DSPs have specialized instructions that performs the operation in a single clock cycle. The MAC operation is for instance the basic operation that builds up a FIR-filter (see Section 2.3).
It is also common for DSPs to support special addressing instructions such as bit reversed addressing and modulo addressing. The bit reversed addressing reverses the bits in the address which is useful in the calculations of the Coley-Tukey FFT. The modulo addressing is used to wrap around when indexing in the circular buffers (as described in Section 3.2).

3.4 Parallelism

One way for digital signal processors to achieve better performance is by utilizing instruction-level parallelism. This can be done with techniques such as superscalar architecture and Very Long Instruction Words (VLIW).

In a superscalar architecture, the hardware looks for data instructions to run in parallel. The choice of which instructions to run in parallel depends on the data dependencies between the instructions.

VLIW is another way of achieving instruction level parallelism. In superscalar architecture, it is the hardware that looks for operations to run in parallel. In VLIW it is the compiler that does that type of work. Operations that can be run in parallel is grouped together in one single large instruction. The hardware can then execute the VLIW instruction operations in parallel without worrying about data dependencies, etc. This means that a similar behavior as in a superscalar architecture is achieved, but the hardware can be simpler and be made more power efficient. Some drawbacks with this approach is that the compiler needs to be more complicated and the backward compatibility is not as good as in an ordinary superscalar architecture.

Another form of parallelism is data level parallelism, which can be achieved by single instruction multiple data (SIMD) instructions. The SIMD instruction operates on multiple data instead of single data. The data needs to be non-correlated for SIMD to work. An example use case could be to add all numbers in a list by some constant. The sequential way of doing this would be to iterate over the list in a for loop and increase each element by the constant. The corresponding SIMD instruction would utilize multiple arithmetic units and execute all additions in parallel, which requires fewer clock cycles.

3.5 Floating- and Fixed point

Digital signal processors (DSPs) can either represent numbers by using fixed or floating point representation [24]. In fixed point representation, the decimal point has a fixed position, meaning that there are always the same number of digits after the decimal point. This results in the representable numbers being uniformly distributed between the minimum and maximum number.

In floating point representation, the decimal point does not have a fixed position. A common floating point representation is the IEEE 754 standard [9], which says that a floating point should be represented with a sign, exponent
and a significand:

\[ (-1)^{\text{sign}} \cdot b^{\text{exponent}} \cdot \text{significand} \] (3.2)

where \( b \) is the base (2 or 10). A 32-bit binary floating point number \((b=2)\) is represented with one sign bit, 8 exponent bits and 23 significand bits. The significand is stored on a fraction format, with an explicit leading one.

Both fixed point and floating point DSPs have their pros and cons. The floating point representations offers a more precise representation and is better for accurate mathematical operations [24]. The fixed point representation often has cheaper hardware cost at the cost of lost accuracy and more complicated development (because of underflow, overflow, etc.) [24].

### 3.6 Cirrus

The Cirrus WM8281 is an audio system with features such as 6 analog or 8 digital microphone inputs and 3 stereo outputs [14]. The processor is a quad-core where each core can run at a maximum of 150 MHz [14]. It supports integer and fixed-point floating numbers that are 24 and 48 bits long. The 24-bit floating point is signed and have 4 integer bits and 20 fraction bits while 48-bit representation has twice as many integer and fraction bits [28]. An overview image of the ADSP2 core is given in Figure 3.3. The description of the different parts are based on the architecture summary document [27] provided by the manufacturer.
Figure 3.3: An overview of the components of the Cirrus core [27].

The core consists of three main components: datapath, address generation units (AGUs) and program control unit (PCU). The datapath block performs calculations and writes the result back to the registers. It supports basic math and logic operations as well as more DSP specific operations such as multiply-accumulate and reversed bit addressing.

The address generation units controls the access to three different memories: X, Y and Z. The width of the memories are 24 bits, which means that a 48-bit data type takes two clock cycles to retrieve. The X memory is the standard location for variables, while the Y memory is an extra read/write memory in which global variables can be annotated to be located to. The Z memory is a read only memory and are used to retrieve constants. The sizes of the memories varies between the different cores in the WM8281 as shown in Table 3.1. Core 2 and 3 are designed to run more advanced programs and therefore have bigger memories. The address generation units supports efficient circular buffer traversing.

The program control unit retrieves the instructions from memory. The program memory is 40 kB for cores 1 and 4 and 100 kB for cores 2 and 3. The PCU retrieves two 20-bit instructions or one 40-bit instruction at a time. One of its features is the zero overhead for-loop which means that the for-loop overhead is handled at hardware level and does not consume any extra clock
Table 3.1: The different sizes of X, M and Y memory in the four cores of Cirrus WM8281 [14].

<table>
<thead>
<tr>
<th></th>
<th>Core 1</th>
<th>Core 2</th>
<th>Core 3</th>
<th>Core 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>P (kByte)</td>
<td>40</td>
<td>100</td>
<td>100</td>
<td>40</td>
</tr>
<tr>
<td>X (kByte)</td>
<td>48</td>
<td>72</td>
<td>108</td>
<td>48</td>
</tr>
<tr>
<td>Y (kByte)</td>
<td>12</td>
<td>72</td>
<td>12</td>
<td>12</td>
</tr>
<tr>
<td>Z (kByte)</td>
<td>12</td>
<td>12</td>
<td>12</td>
<td>12</td>
</tr>
<tr>
<td>Total (kByte)</td>
<td></td>
<td></td>
<td>729</td>
<td></td>
</tr>
</tbody>
</table>

cycles. The PCU supports two levels of zero overhead-loops before it needs to handle the for loops with code [27].

### 3.7 Tensilica HIFI3 DSP

The Tensilica HiFi DSP family contains multiple members [7], but this section will focus on the DSP used by the audio processor DA14195A from Dialog, namely the HiFi 3 DSP. The HiFi 3 DSP can be run up to 290 MHz in the DA149195A configuration [15]. It is based on the Harvard architecture and uses 16- and 24-bit instructions [15]. The DSP also supports 64-bit VLIW instructions that can execute three instructions in parallel [3]. The processor uses fixed-point number representation and supports 16-, 24-, 32- and 64-bit representations [15]. The processor features multiple MAC-units and supports two 32x32 or four 24x24/16x16 multiply-accumulates per clock cycle [3].

The DA14195A has a total of 256 kB DSP RAM and two load/storage units that handles the data memory access [15].

### 3.8 ARM Cortex-A9

The ARM Cortex-A9 is a general purpose processor that runs TrueVoice. The processor is designed to suit applications that requires low power consumption, such as mobile phones [4]. The following information was retrieved from the Cortex-A9 Technical Reference Manual [11].

The Cortex-A9 is a 32-bit processor that implements the ARMv7-A architecture. It supports instructions that are 32 and 16 bits long but can also be extended with 128-bit SIMD instructions to achieve data level parallelism. The processor architecture is dual-issue superscalar which means that two instructions can be run in parallel. The architecture can be modified to support up to four cores. The processor cores features a floating point unit that supports floating point operations on 32- and 64-bit floating numbers. The floating point numbers are represented according to the IEEE 754 standard [9].

The architecture is based on the modified Harvard architecture, with separate L1 caches and buses for data and instructions. The sizes of the two L1
caches can be configured to be between 16 and 64 kB.
Chapter 4

Accomplishment

This section describes the different steps in the project: how the planning was made, how it was executed and what was the outcome of the different parts.

4.1 Preliminaries

Some questions that needed to be answered before starting the practical work was:

– What properties of the candidates should be evaluated?
– What should be tested in practice?
– Which processors should be prioritized?

The first question was answered quite early in the process, after discussing with one of the TrueVoice engineers. The decision was made to focus on three different requirements: processing power, memory and number representation. The processing power is about finding out if the processor is fast enough to run the TrueVoice algorithms in real time. The memory requirements is about finding out if all the buffers that are used in TrueVoice fits in the memory of the DSP. The number representation aspect is about fixed-point number representation. A fixed-point number representation using few bits can give unwanted behavior (in the FFT for instance) if no precautions are taken.

The processing power can be tested in different ways. The best thing would be to port the whole TrueVoice to the candidates, but unfortunately that would take too much time. Instead the decision was made to take out some crucial parts of TrueVoice and test them separately. The decision was made to test two of the main components in TrueVoice: the NLMS filtering used in the acoustic echo cancellation and the FFT that splits the fullband signal into subbands.

The decision was made to only test the processing power in practice. The memory requirements mostly depends on the size of different buffers, which can
be calculated in theory. Optionally, the memory usage of an existing solution could have been tested in practice. But focus was chosen to be put on the processing power since the theoretical value of the execution speeds are harder to predict.

Which processors to focus on was a decision that was made on a higher level of the company. The decision was mainly based on what architectures that were best from a strategic perspective. The two top candidates that were chosen were Cirrus and Tensilica. It was also decided to add ARM Cortex-A9, which already runs TrueVoice, to the list of candidates. By comparing the other candidates with Cortex-A9, it will be easier to make conclusions about how TrueVoice actually would work.

With these things in mind, the following requirements were put on the test program. The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [13]. The test program

R1 MUST implement a signal processing routine with similar characteristics to TrueVoice.

R2 MUST measure wall-clock execution time of processing a fixed size signal sample.

R3 MUST be implemented on Cirrus architecture.

R4 MUST be implemented on ARM Cortex-A9 processor.

R5 SHOULD utilize hardware specific optimization techniques.

R6 SHOULD be implemented on Tensilica architecture.

R7 MAY be implemented on Qualcomm architecture.

R8 MAY be implemented on CSR architecture.

R9 MAY be implemented on Dilogic architecture.

4.2 How the Work was Done

Due to time limitations, the test program was only implemented on the Cirrus WM8281 and ARM Cortex-A9 (integrated on an i.M6Q board) processor. The Cirrus was configured to run at maximum clock rate (147 MHz) while the ARM Cortex-A9 was run at default speed (792 MHz).

The execution time of the ARM was measured by retrieving the wall-clock time before and after the function call and then calculating the difference between the two time points. The Cirrus WM8281 does not have any clock or timer functionality. Instead, the solution was to count the number of function
calls that was made during some fixed time period and then divide the fixed time with the number of function calls to get time per function call. With other words, if $N$ number of executions are made in $X$ seconds, then each execution takes $X/N$ seconds to execute. The Cirrus test program was also evaluated in theory by analyzing the generated assembly code and calculating the number of clock cycles to run the program. The theoretical evaluation of the different architectures was mostly done by reading data sheets about the specific processor.

The focus of the project was with other words to fulfill requirements R1 to R5 while requirements R6 to R9 were omitted. How well the requirements were fulfilled is discussed in Section 6.1.
Chapter 5

Results

This section describes the implemented test program as well as the results of running the tests on different architectures. The three different evaluation criteria are memory usage, number representation and processing power.

5.1 Test Program

The test program tests two of the main parts of the TrueVoice algorithm: a Normalized Least Mean Square (NLMS) filter and a Fast Fourier Transform (FFT). TrueVoice performs the NLMS filtering on the subbands extracted by the FFT, but these parts were separated in the test program.

The test program was implemented on two different architectures: Cirrus WM8281 and ARM Cortex-A9. Because of the fundamental differences between the two architectures, two separate programs had to be implemented.

5.1.1 Cirrus

The Cirrus test program was implemented by using the provided signal processing library. The most complex part of the NLMS function is run \( N \cdot (M - 1) \) times, where \( N \) is the number of input samples and \( M \) is the filter length. The provided signal library implemented this part using four cycles:

1. \( r2 = *(i2 - m0 \% 10) \)
2. \( a1 = rnd(r0 * r2), r2 = *y:i1 \)
3. \( r2 = fix(r2 + a1), r1 = *i2 \)
4. \( a0 += r1 * r2, *y:(i1 -= m0) = r2 \)

Line 1 is traversing the delay line of the filter, retrieving delayed inputs and storing them in a register \( r2 \). Line 2 computes the coefficient change while retrieving the old coefficient. The coefficient change is calculated by multiplying...
the delayed input stored in r2 with the pre-calculated constant \( c = \frac{\mu e(n)}{c + u^H(n)u(n)} \) stored in register r0. Line 3 is calculating the new coefficient value (and storing it in register r2) while retrieving the delayed input again (storing it in register r1). Line 4 is calculating the new output of the FIR filter by adding the product of the delayed input (r1) and the new coefficient (r2) to register a0. The second part of line 4 is writing the new coefficient value to memory.

The FFT function was implemented using the same signal library, but without doing the same research on the assembly code.

### 5.1.2 ARM

The ARM test program uses a custom NLMS implemention that is based on the NLMS filter used in TrueVoice. The program was compiled using `arm-poky-linux-gnueabi` (GCC) version 5.2.0.

The program has two separate loops for the calculation of the output of the FIR-filter and the update of the filter coefficients:

```c
/* Output of fir filter */
for(i=0;i<FILTER_LENGTH;i++) {
    temp += coefs[i]*dly[idx + i];
}
...

/* Update coefficients */
for(i=0;i<FILTER_LENGTH;i++) {
    coefs[i] += BETA*dly[idx + i];
}
```

where `coefs` is an array with the filter coefficients, `idx` is the index of the oldest sample, `dly` is an array with the delayed input samples and `BETA` is the filter step size. The instructions inside the loops are executed \( N \times M \) times, where \( N \) is the number of inputs and \( M \) is the filter length.

The FFT test uses the same FFT library as the TrueVoice ARM implementation, namely the FFT open source library Kiss FFT [5]. It was compiled using similar flags to the NLMS.

### 5.2 Test Results

The execution times of performing an NLMS filter with different filter lengths on the two architectures are illustrated in Figure 5.1. The Cirrus performs the filtering in about twice as long time as the ARM (2.4 times slower with filter length 32 and 1.5 times slower with filter length 4,200). The relation
between the execution times are illustrated in greater detail in Figure A.1 in the appendix.

Figure 5.1: Execution times for Cirrus and ARM when filtering 16,000 samples with NLMS filters of various lengths.

The execution times of the FFT tests are illustrated in Figure 5.2. With 128 points, the execution takes about 3.3 times longer on the Cirrus. In terms of absolute numbers, the execution times differs with about 2 ms. The relation between the execution times are illustrated in greater detail in Figure A.2 in the appendix.
5.3 Memory Usage

The biggest part of the memory usage of TrueVoice is the buffers of the acoustic echo cancellation (AEC) algorithm. For each subband, the microphones have a background and a foreground filter and a shared delay buffer. This gives \((1 + 2 \cdot M)\) number of buffers per subband, where \(M\) is the number of microphones. If there are \(B\) number of subbands and each buffer is \(L\) words long, then the total size of the buffers are:

\[
2 \cdot (1 + 2 \cdot M) \cdot B \cdot L \tag{5.1}
\]

To calculate the theoretical memory usage of the Cirrus AEC buffers, the number of subbands and the filter length must be known for both the low subbands for the FAP algorithm and the high subbands for the NLMS. A good candidate to retrieve these numbers from are the Sharc DSP implementation. The Sharc implementation uses the values shown in Table 5.1.

With only one microphone, this means that the total size of the FAP buffers is \(2 \cdot (1 + 2 \cdot 1) \cdot 22 \cdot 72 = 9504\) words, and the size of the NLMS buffers is \(2 \cdot (1 + 2 \cdot 1) \cdot 43 \cdot 32 = 8256\) words. This gives a total AEC buffer size of \(9504 + 8256 = 17760\) words. The word size of Cirrus WM8281 is 24 bits, which means that 17760 words corresponds to 53 820 bytes. The total memory usage for 1 to 4 microphones are shown in Table 5.2.
<table>
<thead>
<tr>
<th>Number of FAP subbands</th>
<th>22</th>
</tr>
</thead>
<tbody>
<tr>
<td>FAP filter length</td>
<td>72</td>
</tr>
<tr>
<td>Number of NLMS subbands</td>
<td>43</td>
</tr>
<tr>
<td>NLMS filter length</td>
<td>32</td>
</tr>
</tbody>
</table>

Table 5.1: Number of FAP and NLMS subbands and their respective lengths for the Sharc DSP implementation.

<table>
<thead>
<tr>
<th>Number of mics</th>
<th>Memory usage (B)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>53 820</td>
</tr>
<tr>
<td>2</td>
<td>88 880</td>
</tr>
<tr>
<td>3</td>
<td>124 320</td>
</tr>
<tr>
<td>4</td>
<td>159 840</td>
</tr>
</tbody>
</table>

Table 5.2: Cirrus memory usage for AEC buffers with same number of subbands and filter lengths as the Sharc DSP implementation.

### 5.4 Number Representation

Cirrus WM8281 works with 24-bit fixed-point number representation. The main issue with using fixed-point numbers in the TrueVoice algorithm is the risk of overflow when performing the FFT. To avoid saturation in the FFT algorithm, the numbers that are added in each step of the Butterfly diagram are divided by two, which results in that one bit of precision is lost. TrueVoice uses 128 subbands in its FFT. This means that the Butterfly algorithm uses $\log_2(128) = 7$ number of steps and that 7 bits of precision are lost. However, the Cirrus WM8281 uses a 24-bit representation of its fix-point numbers and the sound sampling is only 16 bits. This means that the 8 extra bits can be used to avoid saturation.
Chapter 6

Conclusions

This chapter contains a discussion about the result of the project.

6.1 Discussion

The Cirrus NLMS filtering took about twice as long as the ARM NLMS filtering. However, the ARM clock rate is about five times higher than the Cirrus clock rate which indicates that the Cirrus could utilize its clock cycles more efficiently. The main reason for the Cirrus not being five times slower is probably due to the fact that its instruction set is optimized for signal processing, while the ARM instruction set is not. The ARM's general purpose memory architecture could also have slowed down its processing speed, because of slow memory access due to cache misses. It is also possible that the ARM processing was slowed down by the operating system consuming clock cycles.

The test program implemented two of the main parts of the acoustic echo cancellation algorithm in TrueVoice and hence fulfilled requirement R1. Requirement R2 said that the test program must measure the wall-clock execution time of processing a fixed signal sample. This was achieved in two different ways: the ARM processor could measure the execution time with built-in library functions while the execution time of the Cirrus processor was measured as described in Section 4.2. By measuring the time in a longer period the results are probably accurate enough to make the statement that R2 was achieved as well. Requirements R3, R4 and R6 stated that the test program should be implemented on Cirrus, Tensilica and ARM processors. In reality, the test program was run only on the Cirrus and ARM because of time limitations. Another approach would have been to focus on the Cirrus and Tensilica, but the decision to include the ARM instead of the Tensilica was mostly based on two things. First, the ARM is a multi-purpose processor that can run common C-code which makes it faster to get things running. Second, TrueVoice already runs on ARM so by comparing the Cirrus with ARM instead of with Tensilica
makes it easier to say something about how well TrueVoice can be run on the processor, instead of just make a conclusion on which one of the Cirrus and Tensilica that has best performance. But the best thing would be to implement the program on all three processors.

Requirement R5 stated that the test program should utilize hardware specific optimization techniques. On the Cirrus platform this was achieved by using the built-in library functions for the NLMS and FIR functions. The probability that the manufacturer of the processor can utilize the hardware better than a rookie is quite high. The generated microcode showed different hardware optimizations such as zero-overhead for loops and circular buffer addressing. The ARM processor test program uses similar functions that are already used in TrueVoice. In this case, performance that was more similar to the current implementation felt more relevant than to create the most efficient one.

Requirements R7, R8, R9 were about implementing the test program on three more architectures. Because of limited time resources, the decision to skip those processors were made quite early. More processors in the evaluation had probably resulted in a more interesting comparison however.

The execution time of the ARM was measured with built-in functions while the execution time of the Cirrus was made by counting the number of function calls made during a fixed time period. The Cirrus tests were performed in practice by manually stopping the program after five minutes. A better approach would have been to automatize this process. However, the test results themselves proved to be very close to the theoretical values. Another improvement of the method would have been to complement the ARM wall-clock execution time measurements by counting the number of clock ticks used by the program. That approach would have excluded the clock cycles used for operating system level operations and would make a comparison between the implementations easier. A greater understanding of the ARM implementation could also have been achieved by running the code in a simulator and compare it with the Cirrus simulation results.

6.2 Conclusions

The aim of this project was to find the most suitable candidate to port TrueVoice to. In reality, most of the focus was put on evaluating if it is possible to run TrueVoice on the Cirrus WM8281. The results from running the test program showed that the NLMS filtering took about twice as long to run on the Cirrus compared with the ARM. The ARM can process four microphones which means that the processing power of the Cirrus should be able to handle about two microphones. The calculated memory requirements of the AEC buffers showed that one microphone would require 54 kB of memory while two mics would require 89 kB. However, other parts of TrueVoice consumes memory as well, which means that core 1 and 4 probably do not have enough memory to
run the processing. Core 2 seems to have enough memory to handle one microphone while core 3 should be able to handle one to two microphones. With other words should the two cores be able to process two to three microphones in total. The fixed-point issues in the FFT should not be of any problem since the architecture uses a 24-bit number representation. With these things in mind, the conclusion is that Cirrus WM8281 is a good candidate to port TrueVoice to.

6.3 Future Work

The future work of this subject could either go wider or deeper. A wider approach would be to include more of the candidates listed in the requirements. This would result in a better comparison. The deeper approach would be to start porting different parts of the TrueVoice code while continuously performing measurements of performance and memory usage. The final step in this case would be to port the whole program.

It would also be interesting to evaluate more aspects such as how easy the development tools of the different architectures are to use.
References


Appendix A

Test Results

A.1 Relative Execution Times NLMS

Figure A.1: The execution time for Cirrus divided by the execution time for ARM when filtering 16,000 samples with NLMS filters of various lengths. The normalized line illustrates the theoretical ratio if both processors had the same clock rate.
A.2 Relative Execution Times FFT

Figure A.2: The execution time for Cirrus divided by the execution time for ARM when performing FFT transforms of various sizes on 16,000 samples. The normalized line illustrates the theoretical ratio if both processors had the same clock rate.