JOPSPEECH

Embedded Java Speech Recognition SDK

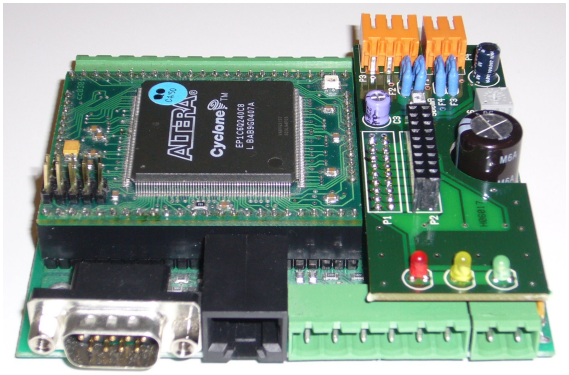
Mikael Lundsgaard
Mikael@jopspeech.com
Department of Informatics
Copenhagen Business School

Jens Kritian Rasmussen
Jens@jopspeech.com
Department of Informatics
Copenhagen Business School

Supervisors:

Rasmus Pedersen
Department of Informatics
Copenhagen Business School, Denmark

Martin Schoeberl
Institute of Computer Engineering
Vienna University of Technology, Austria



December 20, 2006

Abstract

Embedded Java is often mentioned today in respect to embedded systems due to its potential to play a key role in next-generation ubiquitous and pervasive systems, utilizing some of the powers that are inherent to the language such as memory management, thread- and networking- support. Next generation computing environments rely heavily on the platform's ability to communicate, not only with other systems, but also with humans.

This master thesis presents JopSpeech SDK a speech recognition Software Development Kit (SDK) for the Java Optimized Processor (JOP) - an implementation of the Java virtual machine in hardware made by Martin Schoeberl. JopSpeech is the first embedded Java speech recognition SDK developed for JOP. The SDKs is also the first embedded Java speech recognition that consist of a "full circle" approach, defining a set of software components, interfaces and hardware for audio in and out - that allow developers and researchers to take advantage of speech technology in the embedded Java structure of JOP. The proposed JopSpeech SDK consists of a set of components to build speech models: hardware components interfacing with microphones and loudspeaker, software components processing sound and recognizing speech. JopSpeech also proposes an architectural model built on the "Pattern Recognition Approach" proposed by L. Rabiner and B.H. Juang in 1993 as a part of its architecture for developing and testing speech recognition systems.

This master thesis also demonstrates the usages of JopSpeech as an embedded Java speech recognition research architecture with JOP, implementing it on a Altera Cyclone EP1C6Q240 Field Programmable Gate Array (FPGA). Experiments have been conducted on the FPGA using two of the most complex digital signal processing approaches to speech recognition , the "Mel Frequency Cepstral Coefficients" (MFCC) and the "Cepstral Linear Predictive Coding" representation of the speech signals. The research experiments investigate the prediction results, classification speed and sound representation covariance, on both a Speaker-dependent and a semi-Speaker-independent system.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Pervasive computing is the next generation of computing environments with information and communication technology everywhere[1], for everyone, at all times. Being able to speak to your microwave oven, washing machine, refrigerator etc., would have been impossible before, but due to the major advances within the processing power of microchips, this now seems possible. With the processing power being available the only question left seems to be, why our systems are not communicating yet. One of the answers might be the lack of a relatively easy, secure, portable and common development platform. Java presents many of the needed features[2], enabling communication between systems, but it still needs further development and research to be ready for embedded system development.

This master thesis introduces JopSpeech SDK, an embedded Java speech recognition SDK built for JOP [29] defining interfaces, and a set of software components that allow developers and researchers to take advantage of speech technology in the embedded Java structure of JOP.

The JopSpeech SDK is explained in this master thesis however tutorials, examples and the actual Java code can also be found on www.jopspeech.com which is the accompanying web site for the master thesis .

[1] IEEE Pervasive Computing `http://www.computer.org/portal/site/pervasive/`

[2] as cited in [29]

1.1 Objective of Master Thesis

Primary objectives

Our primary objective with this master thesis is to create a software development kit (SDK) for embedded Java speech recognition with a set of components giving other researchers and developers a simple approach to embedded speech recognition system development. The components developed for the speech recognition SDK will all be designed and implemented with low Worst Case Execution Time, analyzed by the WCET tool[3], to address the real time challenges of an embedded system3.

Primary objective listed:

- To define a speech recognition architecture to simplify the speech recognition system development approach.
- To create a set of basic speech recognition components enabling developers and researchers to easier development of speech recognition systems.
- To insure the components have a predictable and low Worst Case Execution Time.
- To introduce a working prototype embedded on a FPGA board, together with JOP.

[3] developed by Rasmus U. Pedersen and Martin Schoeberl[1]

1.2 Problem Description

Speech recognition is commonly used in connection with a lot of computer power and memory utilization[40]. Phone companies have used speech recognition for caller systems (Interactive Voice Response) for a while, and with the introduction of VoiceXML (VXML), this has been spread to even more industries[27]. The medical industries and legal systems have lately started using speech recognition for dictation systems, where users record their "words", and then upload this to a speech server which then makes speech-to-text transcripts of the user's voice. The Danish Radio and Television company has just finished development of a speech recognition based system for subtitled editions of the news [4]. On embedded systems, speech recognition is especially found in mobile phones, PDA's, however the technology is also used in biometric systems for user recognition.

Although major advances have been made in speech recognition and within the field of microprocessors which have made their prices lower, their processing speeds faster and memory capabilities larger, there is still not a lot of speech recognition embedded systems running on Java. This might have something to do with the lack of easy accessible hardware options, but also the lack of support from the programming language. C or Assembler are the most common programming languages for embedded systems[34], but it is our claim that they do not really support the demands which exist for today's systems, options for networking or component nor are the object oriented system development supported by the languages. Why is this? When C first appeared, networking did not exist and object oriented methods were unknown and systems with 32 KBytes of memory were considered large. All this has changed in the past 30 years, and it is within this background that Java originates. Java is a modern, object-oriented language based on open, public standards. Java is much more standardized and probably has a much richer collection of core functions than any other general-purpose computer language[5]. Java programs are potentially much more reliable than C, as C has essentially no runtime error checking. Memory allocation is all manual, etc. Where as Java does memory management automatically, bounds-checks array access and enables networking support and thread management, etc.. However although Java has some advantages it is still a relative newcomer to small embedded systems (some implementations do exists), especially realtime systems.

Java's large components and object oriented approach with all its error checking and listening, sometimes make the Java language unnecessarily slow. This is where we hope we can help in the

[4] `http://videnskabsministeriet.dk/site/forside/nyheder/pressemeddelelser/2006/dansk-talegenkendelse-vinder-international-pris`

[5] `http://www.practicalembeddedjava.com/`

aiding of the embedded Java development by constructing a set of components for speech recognition and proposing a development architecture for easier access to building speech recognition systems. Although some of the speech recognition problems are general for all speech recognition systems we will also deal with the problems which are inherent for embedded systems, such as lack of memory, lack of speed, lack of power, lack of floating point units etc., in our construction of the components.

Speech Recognition systems performance is normally measured in accuracy and speed and can be categorized into one or more of the following types of speech recognizers:

Speaker-independent: Speaker-independent systems are recognizers that can be used by anybody without the need of training. These systems are usually deployed in environments where the system cannot be trained.

Speaker-dependent: Speaker-dependent systems are those systems that first need to be trained for use by a specific speaker. Voice samples of each speaker are taken, analyzed and stored. Speech is then matched with these samples to accurately determine the words spoken.

Continuous Speech Recognition: Continuous speech recognizers allow users to speak naturally and continuously.

Isolated or Discrete Speech Recognition: Isolated speech recognizers require the speaker to pause (about a fifth of a second) between each word that is uttered, so that the recognizer can buffer the word before processing the next one.

Vocabulary Constrained Systems: These systems have a limited vocabulary they can understand. Small vocabulary means that users have to restrict speech to only words that recognizers understand.

Our experiments focus mainly on the ”Vocabulary Constrained Systems”. To test and evaluate the findings and challenges of our speech recognition SDK, we have constructed a prototype of a speech recognition system which is implemented on an FPGA with ”JOP” [29]. By using ”JOP” we hope we can contribute to the testing and development of Martin Schoeberls[29] continuous development of the JOP board. Further using Java and releasing under open source [6] we hope that other people will benefit from the knowledge we find, and be able to contribute to it in the future. As the future goal beyond this thesis is to get the speech recognition SDK completed with all primary speech recognition components developed for the various embedded Java platforms.

[6] See accompanying web site `www.jopspeech.com` for more details

1.2.1 Specific Problem Field

Advances in electronic and computer technology are causing an explosive growth in the use of machines for processing information. There is thus a need for effective ways of transferring information between people and machines, in both directions. One very convenient way in many cases is in the form of speech, because speech is the communication method most widely used between humans. Normal communication between humans and computers has relied mainly on keyboards and screens in transmitting the information in a written text form. For many years in this field Java has been proven to extend the communication between machines with their platform neutrality and large library extending programmers with the ability to ensure extended use of systems everywhere. However Java has had some shortcomings as an embedded language and as a result does not have a very extended library in this specific area.

1.2.2 Problem Definition

To create an embedded Java speech recognition SDK developed for JOP, defining a set of software components and interfaces that allows developers and researchers to take advantage of speech technology and aid in the further development and research of speech recognition systems in the embedded Java structure of JOP.

1.3 Delimitation and Definition of Subjects

1.3.1 Definition of SDK

As the term SDK (software development kit) can at times be quite diffuse, our definition is as follows: A programming package that enables a programmer to develop applications for a specific platform. Typically an SDK includes APIs, System tools and documentation.
JopSpeech SDK therefore consists of the following:

Platform JVM implemented in hardware - JOP described in Chapter 3

API The Java implemented speech recognition classes described in Chapter 8, and its architecture described in Chapter 7

System tools The hardware interface for microphone and loudspeaker with its accompanying A/D and D/A conversion base

Documentation This thesis and `www.jopspeech.com`

1.3.2 Delimitation of Master Thesis

Mathematical principals: We expect the reader of this master thesis to have insight into some mathematical principles beyond the level of a normal Cand.Merc.Dat graduate.

Semantic speech recognition: We will only deal with predefined meanings of words and not the context in which they are spoken.

Speech recognition theory: The reader should be advised that a review of this kind done in 20 pages is, at its best, cursory, but will suffer from lack of information.

1.4 Overview - *How the rest of the paper is structured*

The rest of this master thesis can be subdivided into 4 overall subjects:

Background information Under background information we will go through the basic theory of speech recognition and other relevant material to identifying the requirements of the SDK. Chapters 2, 3, 4 and 5.

Implementation Here we will discuss the architectural design and the actual implementation of the embedded Java speech recognition components. Chapters 7, 8 and 9.

Experiments Here we will show the use of JopSpeech and conduct some experiments to show some of the potential of the SDK. Chapter 10.

Conclusion Final evaluation, future work and our conclusions. Chapters 12, 13 and 14

1.4.1 Methodologies

The master thesis cannot be said to belong to one overall methodology, more to a set of different theories where appropriate, as the master thesis has been very explorative. Here is a short description and references to appropriate primary literature used:

Theory Digital signal processing (DSP) Erik Hüche 1996 [11], speech recognition Xuedong Huang,A. Acero, H. Hon 2001 [40], L. Rabiner and B-H. Juang 1993[25].

Architecture Architectural design, and interfaces L. Rabiner and B-H. Juang 1993[25].

Implementation JOP and Real time systems, Martin Schoerberl 2005[29], Worst Case analysis(WCET) Rasmus U. Pedersen and Martin Schoerberl 2006[1]

Economic Market strategy / price model (Michael Porter 1979 [22]), The Economics of Technology Sharing (L. Josh and J. Tirole 2005 [13])

Chapter 2

Sound and Speech Signal

In this chapter we will give an introduction to speech, the various approaches to speech recognition and some of the challenges these approaches give when dealing with machine decoding.

2.1 Approaches to Speech Recognition

Broadly speaking there are two different overall approaches to decoding speech on a machine [25], where both approaches have their challenges:

- The Acoustic Phonetic approach
- The Pattern Recognition approach

On top of these two approaches there are several mixtures of these that have been researched as a way to improve the overall performance of speech recognizers. Especially neural networks in conjunction with Pattern Recognition techniques are a research field of its own [25].

The Acoustic Phonetic approach is based on the theory of acoustic phonetics Section 2.2 that postulates that there exist finite, distinctive phonetic units in spoken language and that the phonetic units are broadly characterized by a set of properties that are manifested in the speech signal/spectrum over time. These distinctive phonetic units are normally referred to as ”phonemes”, the number of them is dependent on the language spoken, see Subsection 2.1.1 and see Figure 2.3 for a presentation of American English phonemes.

The Pattern Recognition approach is based on the machine learning theory (this is the approach that we will be using in this thesis), where no knowledge of the phonetics units is needed in advanced. As in all other machine learning approaches the method has two steps, a ”training step” of the speech pattern and a ”classification” of the speech pattern. The speech knowledge is brought

into the system via the training method, where different versions of the sound (phoneme/word) are ”taught” by the system depending on the various algorithms used.

2.1.1 Basics of Sounds

Speech sounds as we understand them as humans are words connected in a context, like ”you are my friend” might have two different meanings depending on if you just won in Lotto or not. However if we think about a language we do not understand, the sounds do not makes up a word - but just a set of sounds. The different sounds - or *phonemes* which are the smallest classification of speech sounds, are all produced via a speech-production/speech-perception process in human beings. The production starts as the ”speaker” formulates in his mind what he wants to transmit to the ”receiver” via speech. The speaker then converts the transmission in his mind into a language code (English, Danish, German, etc.), this roughly translates into translating a written text into some phonemes in a specific sequential order.

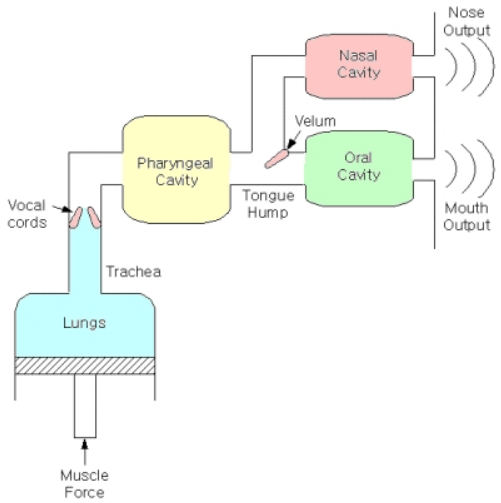


Figure 2.1: Speech Production process [25]

A simplified representation of the complete physiological mechanism for creating speech is shown in Figure 2.1. The lungs and the associated muscles act as the source of air for exciting the vocal mechanism. The muscle force pushes air out of the lungs (shown schematically as a piston pushing up within a cylinder) and through the trachea. When the vocal cords are tensed, the air flow causes them to vibrate, producing so-called voiced speech sounds. When the vocal cords are relaxed, in order to produce a sound, the air flow either must pass through a constriction in the vocal tract and thereby become turbulent, producing so-called unvoiced sounds, or it can build up pressure behind a point of the total closure within the vocal tract, and when the closure is opened,

the pressure is suddenly and abruptly released and the air flow is chopped into quasi-periodic pulses which are then modulated in frequency in passing through the throat, the oral cavity, and possibly nasal cavity. Depending on the positions of the various articulators (i.e., jaw, tongue, velum, lips, mouth) causing a brief transient sound [25]. Once the speech signal/sounds (series of phoneme) has been released and transmitted, the listener then processes the acoustic signal along the basil membrane of the ear, which provides a running spectrum analysis of the incoming signal. In a manner not well understood, the brain then converts this spectral signal received from the basil membrane into a language code and finally an understood signal.

Now we understand the basis of sound production in humans, so lets understand the representation.

2.1.2 Representing Speech in Time and Frequency Domains

Speech is produced as a set of sounds over time Subsection 2.1.1, hence the various sounds characteristics is changing. The characteristics are, though, fairly stationary when examined over a short period of time (5 - 50ms), meaning if we examine a short period of sounds the signal will seem fairly stationary. If we on the other hand examine a sound signal over a longer period of time we are able to see the changes in the sounds characteristics that makes up the various phonemes or words.

If we look at Figure 2.2 the top graph shows the word "pizza", depicted in waveform with 50ms spacing on the graph. We can see that if we look at a period of 50ms the signal is very stationary, but over a longer period we are able to see the changes that make up the full word pizza. We can also see the difference in the various phonemes (see Figure 2.3 for standard phonemes of American English) instead of letters (as we normal use in a alphabet), if we look at the beginning of the word pizza, we see it starts with a silent (SIL) piece, then from $0.15 - 0.20s$ some stationary movement/unvoiced sound which is the consonant sound we start with when we say the *[P]* and then from $0.20 - 0.25s$ we can see the letter *[i]* after the *[p]* in the beginning of the word pizza. In the middle and in the end of the word pizza we can see three letters more, but it is hard to distinguish them from each other - especially the middle and last one, although one is a *[z]* and the last an *[a]*. However if we look at the bottom graph, which shows the same word "pizza" depicted in its spectrum (frequency domain) we can clearly see the different formant frequencies that make up the various phonemes - the phonetic depiction of pizza is depicted in the middle according to the International Phonetic Alphabet (IPA[1]) and the formants by arrows on the spectral graph. A formant frequency is the frequencies that the different phonemes resonate in. In Figure 2.2 the 3

[1] `http://www.arts.gla.ac.uk/IPA/index.html`

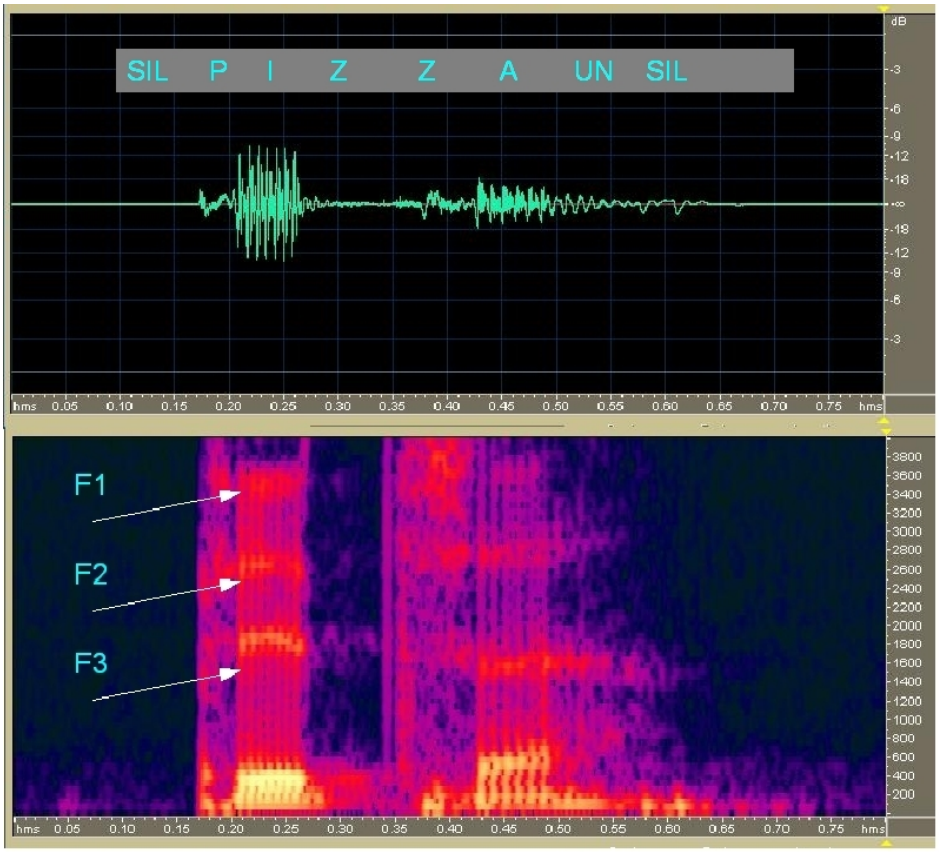


Figure 2.2: ”Pizza” depicted in waveform and spectral view, with phonemes F1, F2 and F3 indicated

formant frequencies corresponding to the vowel *[i]* in Pizza (F1, F2 and F3) are depicted by an arrow in the figure.

2.2 Acoustic Phonetics Approach

Using the knowledge we have gathered from the previous two subsections: 2.1.2 and 2.1.1, we now understand the Acoustic Phonetics approach much better. There are around 40 different phonemes in American English (see Figure 2.3).

Looking at Figure 2.3 we can see the various subdivisions of phonemes into vowels, consonants, diphthongs, etc., it is this separation of phonemes that the Acoustic Phonetics approach builds on. The theory is that each phonetic unit (phonemes) is broadly characterized by a set of properties that can be compared and identified each time. If we look at Figure 2.4 we can see an example with vowels, which are identified by pitch frequency, the front/mid/back placement of the tongue, the tension in the voice and its formants. All of these various characteristics set the

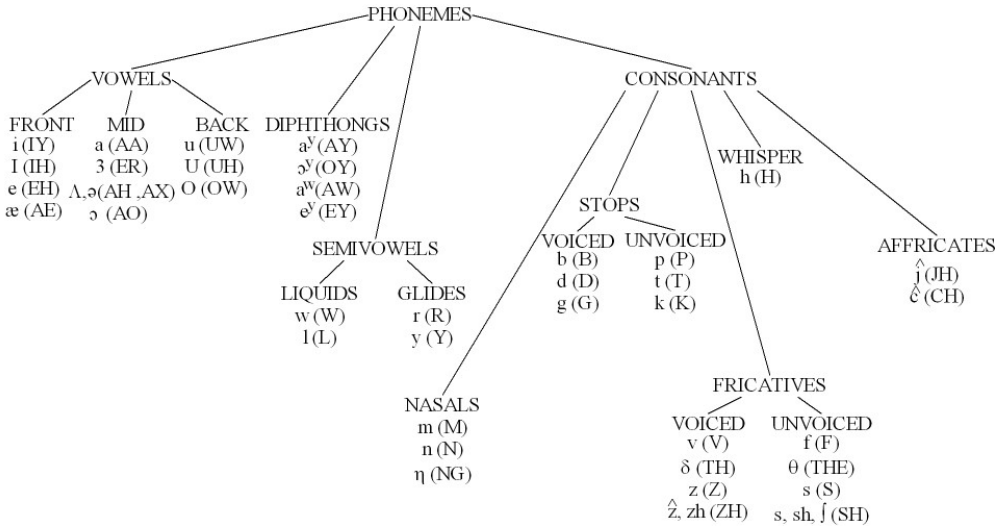


Figure 2.3: Chart of the classification of the standard phonemes of American English[25]

properties/features for the various phonemes that build up words/sentences.

Frequency & Features of Vowels

Vowel	Example	Frequency	Height	Backness	Tenseness
A	ɑlms	30	low	central	tense
æ	ɑx	198	low	front	lax
Q	ɑdd	128	low	back	lax
ai	/des	136	---	---	tense
Au	ɑʊt	44	---	---	tense
e	ɑpe	183	mid	front	tense
E	ɛbb	159	mid	front	lax
R	ɛrg	115	mid	central	tense
i	ɛɑt	210	high	front	tense
I	/f	207	high	front	lax
o	ɑʊts	146	mid	back	tense
O	ɑʊght	110	mid	back	tense
Oi	ɑ/nk	25	---	---	tense
u	ɑʊze	117	high	back	tense
U	pʊsh	38	high	back	lax
V	ʊp	155	mid	central	lax

Figure 2.4: Chart of the different distinct features of vowels[25]

The first step in an Acoustic Phonetics approach is to analyze the speech, and make an appropriate spectral representation of the signal. The second step is then to make feature detection: The idea is to convert the spectral measurements into a set of features that describe the properties of the phonetic signal, as we can see an example of in Figure 2.4 and have already discussed.

Having determined the various properties of the phonetic signal, the third and most important step is to segment the various signals and label them into their respective phoneme. The machine

has to find a segment where the signal is more or less stable. Again if we refer back to Figure 2.2 we can see that the signal is stable during the various phonemes. The stable segment of the sound then has to be compared with how well the features fit the properties of the various phonemes and then classified as one of these. The fourth step is to label the phoneme lattice in respect to a given vocabulary - e.g. find the best matching word in respect to syntax and lexical coherence. Step 4 is usually solved using a probabilistic approach in conjunction with, for instance, a decision tree.

To illustrate some of the difficulties in this approach, try to imagine the following sentence, illustrated in Figure 2.5:

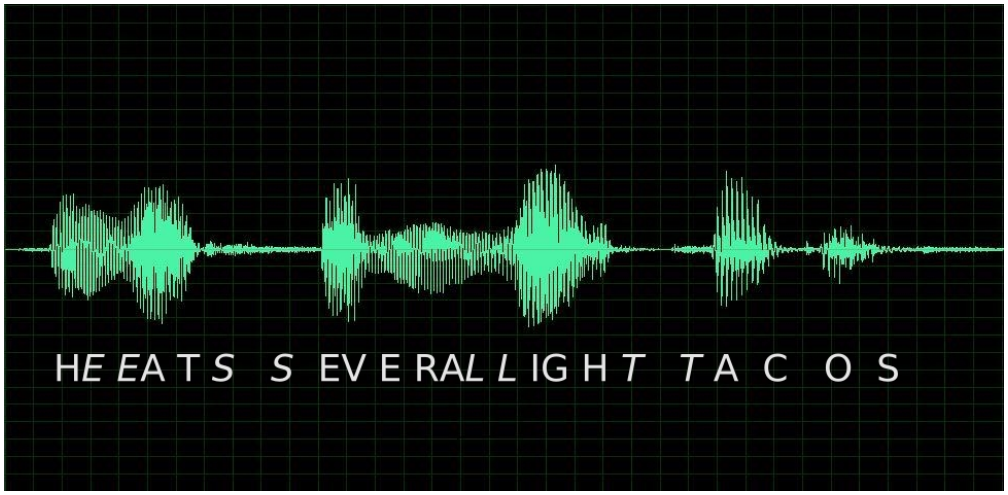


Figure 2.5: Sentence

”he eats several light tacos”

Notice how every word ends on the same phoneme as the next word starts with, this creates substantial problems for the segmentation process. The machine might segment the phonemes into:

”h a t s e v e r a l i g h t a c o s ”

Which could be recognized in the lexical step 4 as being ”hats ever alright as” - and we can probably imagine more ways of segmenting these words. Looking at Figure 2.5 we can especially appreciate the difficulties, the two [s] phonemes and [t] phonemes are both difficult to segment and also to distinguish wether they are noise of voiced segments. The challenge in recognizing the correct words is therefore not an easy feat, nor is it easy just to recognize the phoneme, as no one pronounce the same phoneme exactly the same way every time.

The next big problem is that the method requires extensive knowledge of the acoustic properties of phonetic units. This knowledge is at best incomplete and at worst unavailable. No defined way

of labeling the training data exists [25] which means that somebody might do it through formants, pitch, voiced/unvoiced, energy, nasality etc., and some might do it by means of other features and properties. These problems account in many cases for the lack of success the Acoustic Phonetics approach has had in practical speech recognition systems. However, the Acoustic Phonetics idea still remains interesting, but it is an idea that needs more research and agreed upon standards before it can be used successfully in actual speech recognition problems. To read more about the Acoustic Phonetics approach and linguistics see [21] or [15]

2.3 Pattern Recognition Approach

The Pattern Recognition approach builds on machine learning principals and is therefore not inherently dependent on the pre-knowledge of the acoustic properties of the phonetic units or where this knowledge can be gathered. The factors that differentiate various Pattern Recognition approaches are the types of algorithms and features the chosen model/template is built upon, but the approach is more or less the same. Figure 2.6 represents an extended version of the Speech Pattern Recognition approach as described by L. Rabiner and B.H. Juang 1993[25]. The framework will be used as a guideline through the rest of the master thesis to explain JopSpeech.

Analysis System: The Speech signal is transmitted to the analysis system for pre-processing. This normally means representing the signal in some sort of spectral vector which is then used as a training vector or as the vector to be tested, depending on whether the classification is known or unknown.

Pattern Training: The feature vectors are processed by a chosen algorithm that, depending on the training vectors, identifies a set of pattern representatives that makes up a model or a template.

Templates or Model: The pattern representatives of the previous step is used to build a Classification Model.

Pattern classifier: The feature vectors that are unknown are tested against the saved reference patterns (Models or Templates) and a distance measure is calculated.

Decision logic: Here it is decided which reference pattern matches the unknown pattern best and it is labeled accordingly.

The challenges with the Pattern Recognition approach are dependent on which algorithm is used for modeling - like with all other machine learning approaches, but with sound and embedded

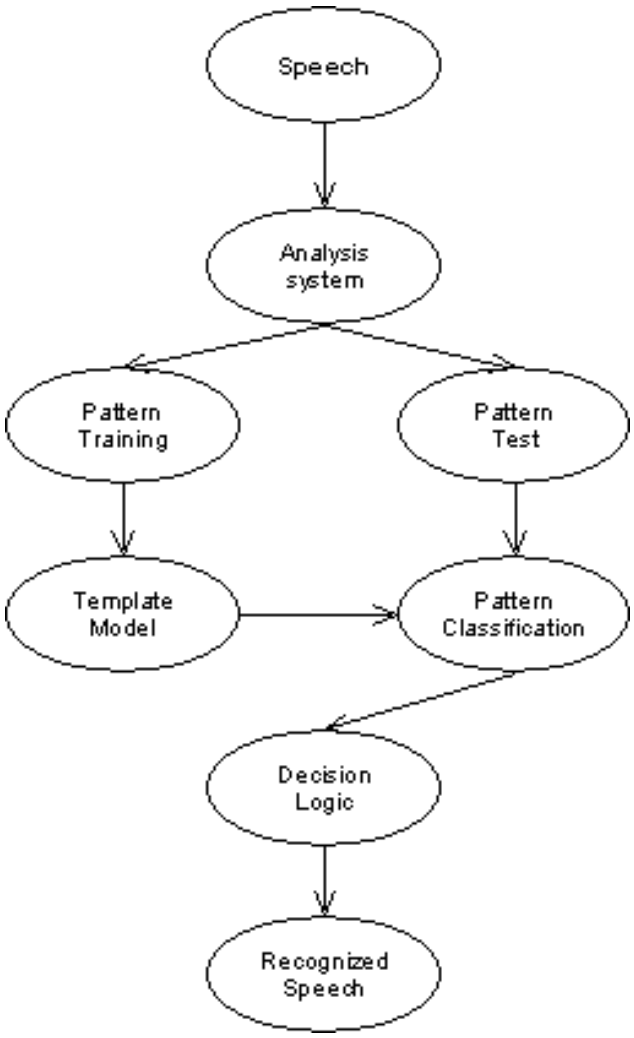


Figure 2.6: Diagram of the Pattern Recognition speech recognition approach[25]

systems there are also a couple of extra challenges. Some of them are summarized here, but we will discuss these more in depth during the rest of the thesis:

Amount of training data: The amount of training data is important for the performance of the system. Generally one might say that the more training data the higher the performance of the system for virtually any task (see section 5.3).

Environment: The reference patterns are all sensitive to the environment they are recorded in. Background noise, poor/good microphone, electrical interference and transmission of the signal are all factors that could influence the reference patterns and thereby the performance of the system.

Pre-knowledge: No speech-specific knowledge of acoustics is required to build the systems,

hence the performance is quite unaffected by the lack of pre-knowledge. Knowledge of general digital sound processing is, however, a necessity, as this will improve the systems performance in choosing the correct acoustic properties of the phonetic units.

Computational load: As more training data is submitted to the system the performance increases, but so does the amount of memory and processing power consumed. The processing consumption in training and classification increases almost linearly with the amount of reference patterns/training data.

Insensitive to size: All pattern-recognition techniques are insensitive to the size of the pattern of the recognized part. The algorithms can, with limited or no change at all, change from recognizing at phoneme level, to word level or sentence level. The algorithms do not distinguish between the size of the sound signal. The algorithm only concentrates on the reference pattern and can therefore be trained on any size pattern.

Tuning and constrains: There are well defined ways to train and tune the various segments of the algorithms in order to enhance the performance of the algorithm and several communities exist where standards are agreed upon[2]. Constrains on semantics and syntax are also fairly easy to implement in the models, using for instance statistical probability algorithms, thereby improving accuracy and maybe reducing the computation.

To read more about the Pattern Recognition Approach and Machine Learning see Frank and Witten [9] or Rabbiner and Juang [25].

[2] `http://www.kdnuggets.com`

Chapter 3

System Platform

This chapter will present an introduction to Embedded Systems, the programming language Java and JOP the Java Optimized Processor.

3.1 Embedded System

The definition of an embedded system is not exact and seems to expand with new inventions both in hardware and software (such as JopSpeech). An embedded system can be an independent or small part of a larger system and it is often hidden. The embedded systems are built for a special purpose and, since the system is dedicated to specific tasks, design engineers can optimize it, reducing the size and cost of the production. Integrating CPU, memory and power until its minimum demands are met is normal when embedded systems is sent into mass production.

In today's digitized world, embedded systems can be found in all most everything: MP3 players, calculators, the microprocessor that controls the anti-brake system of a car, etc. The anti-brake system also introduces another characteristic of an embedded system, that it is often a real-time system. The system needs to function within a given time frame and must be reliable. The worst case execution time for an embedded system is a factor that must be known. Imagine its importance with a anti-brake system; the driver should be able to trust the system to work when it is needed.

Another issue with embedded systems is energy consumption. Many embedded systems are battery driven, i.e. mobile phones, atomic missile guidance system, or PDA's. The CPU power and memory in embedded systems are therefore restricted, as to prevent any unnecessary power consumption. Also the system will normally lay dormant while inactivated.

The list below shows some of the characteristics of a embedded system[34]:

Predictable Response Time: It is known how long the response time is for the system.

Reliability: The system should be reliable and trusted, without any human intervention.

Real-Time: The system runs within a worst case scenario.

Memory: Memory is a limited resource in an embedded system.

Power Consumption: Power consumption should be low because many embedded systems run on batteries.

CPU: The power of the CPU in an embedded system is restricted.

Fulfilling the demands of an embedded system, the development environment needs to be able to handle debugging and testing of the system and provide an easy way to program and install applications on the system. This can be achieved by developing emulators on a PC, or implementing the systems on a Field Programmable Gate Array (FPGA), but still the programming languages need to support some kind of debugging structure. The most common programming languages in embedded systems are currently C and Assembler [34]. These are low-level languages with small libraries and limited support structures.

3.2 Java

Java is currently not the most used language when speaking about embedded systems [29], which is quite ironic since that is what Java actually started as [4]. In 1991 a group of SUN engineers wanted to design a small computer language that could be used for consumer devices like cable tv boxes [4]. In the beginning there where two overall requirements for the language:

1. Since the small devices did not have a lot of memory and CPU power, the language had to be small and compact.

2. Because different manufactures choose different central processing units (CPU's) it was important not to be tied down to an architecture.

The requirements for small, compact and platform-neutral code led to the design of the portable language that generated intermediate code for a hypothetical machine - often referred to as virtual machines, hence the Java Virtual Machine (JVM). Although a small device (a remote control) was developed as a "proof of concept", the language never caught on in the embedded industry. It was not before the emergence of the internet that the language caught on, as the platform-neutral

architecture was just what the big ”www” needed. So in January 1996 the first Java Development Kid 1.0 (JDK) was released [4], and although more or less everything has been changed over the years since the first release, Java’s original requirement of a platform-neutral language is still one of the backbones of the technology.

3.2.1 The ”Buzz” of Java

The advantages and drawbacks of Java compared to other programming languages are discussed by users everywhere, in certain circles it takes on a nearly ”religious” excitement. In this master thesis we will not try to evaluate which language is the best or anything of that nature. It is out of the scope of the master thesis . However, we will give a short presentation of the various concepts in Java:

Object oriented Java is an object oriented language comparable to C++ with all its facets of inheritance and interface definitions.

JIT or Interpreter When compiling Java the Java Compiler (Javac) translates the Java code into bytecode. Then the bytecode is interpreted by the Java Virtual Machine (JVM) on the specific platform. A second way is Just In Time (JIT) compiled. Just In Time compiled is when the bytecode is compiled by the JVM into native code and cached for later reference, this approach normally insures a faster execution[4]. The third way is a a mix of both.

Simple and Robust The Java compiler detects many problems before runtime, such as wrong assignment of variables, the JVM does array bounds check and Java does not use pointers for strings and arrays but instead contains true arrays.

Memory Java keeps you from corrupting memory outside its process space. It allocates memory for you and through the garbage collector frees it again.

Distributed Java had - as we mentioned, its breakthrough as an internet/network language. The platform neutral code enables applications to be distributed easily.

Multi threaded Java has multi-threading support built into the language, supported with monitors with synchronizing calls.

Library Java’s great strength lies in its large and freely available library of methods and classes.

Java has over the years developed from being a network technology for both desktop, server and even mobile technology which has resulted in different versions/editions of Java. The J2SE

is Java's standard edition; currently Standard Edition 1.4.2_12 has a 50 MB large installation file. The J2EE is Java's enterprize edition; currently J2EE SDK 1.4_03 has a 120 MB large installation file. J2ME is Java's micro edition, a stripped down version of Java with only its most necessary components. Currently, there are two Java ME configurations: the Connected Limited Device Configuration (CLDC) and the Connected Device Configuration (CDC). The CLDC is for small wireless devices with intermittent network connections, like mobile phones and personal digital assistants (PDAs). The other base configuration is the Connected Device Configuration (CDC). This configuration is for larger devices (in terms of memory and processing power) with robust network connections. Set-top boxes, Internet appliances, and embedded servers are good examples of CDC devices[1].

3.3 JOP - Java Optimized Processor

JOP is, in short (we quote from Martin Schoeberl's thesis [29]), an optimized Java Virtual Machine implemented as a soft core in a Field Programmable Gate Array (FPGA), designed for resource constrained devices, with time predictable execution of Java programs. On JOP the Java bytecode is translated into microcode instructions or sequences of microcode (see chapter 5 in [29]), that results in a high performance and time-predictable execution.

JOP is still under development which means that some of the features of Java are not fully implemented yet, and other Java specifications are left out of the JOP design as an optimizing of the embedded design. Among these are:

- The garbage collector is not yet implemented. Therefore the memory heap becomes immortal memory.
- JOP has no support for floating point or double, integer arithmetic is therefore needed
- JOP is implemented as a stack machine the stack size is therefore limited, which prohibits recursive programming.
- The stack size limitation prohibit large methods, or too many initializations of declared variables in one method.

JOP is a hard-real-time system, which means that it is possible to analyze the worst-case execution time[29]. The benefits of JOP being a hardware implementation of the JVM are that instructions that is hardware implemented has a very low execution time, the list of execution cycles can be

[1] `http://java.sun.com/javame/technologies/index.jsp`

seen in ”JOP: A Java Optimized Processor for Embedded Real-Time Systems”[29]. As mentioned

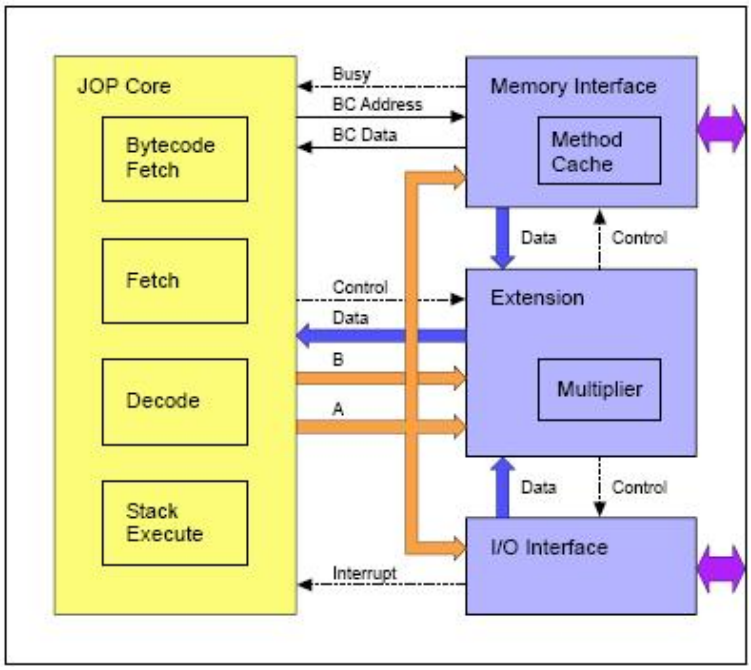


Figure 3.1: Block Diagram of JOP [29]

already the way JOP works is that it maps the Java bytecode to microcode instructions with a single cycle instruction, although the configuration can be adjusted to the chosen FPGA and whatever extra hardware extensions this might be extended with. The configuration of JOP does not change. It contains the processor core, a memory interface, a number of IO devices and an extension module. This can be seen in Figure 3.1. The processor core contains the four microcode pipeline stages: microcode fetch, decode and execute as well as an additional translation stage bytecode fetch[28]. The module called ”Extension”, which can be seen in Figure 3.1, provides the link between the processor core, the memory and the IO modules, handling all the data reading and writing.

JOP can be implemented on various FPGAs with different hardware extensions. This is one of the strengths of JOP and also one of the reasons why it is very suitable as a research platform for an embedded speech recognition solution. The main distribution of JOP comes with a port to different FPGAs[2]:

- Altera Acex 1K30 or 1K50
- Altera Cyclone EP1C6 or EP1C12
- Xilinx Spartan-3

[2] `www.jopdesign.com`

Besides that, several I/O-hardware board extensions can be added to the FPGAs.

3.3.1 Real-Time

JOP's runtime system defines a real-time profile for embedded Java and guidelines for the application structure of systems running on JOP[29]. The application structure on JOP consists of a division of the application into two different phases: *Initialization* and *Mission*.

The Initialization phase Performed at start-up and should consist allocation of all shared objects, thread creation and all non-time critical methods. As the current profile of JOP does not support garbage collection, all memory should be allocated to avoid heap restrictions as this automatically becomes immortal memory.

The Mission phase Performed after the initialization phase, all threads are created and their priority set and remains unchanged.

3.3.2 Hardware

The hardware platform we will use in this master thesis is based on BaseIO, an I/O-hardware board extensions to JOP with an Ethernet connection and the Altera Cyclone EP1C6Q240 FPGA (see Figure 3.2). Using the BaseIO board will give us the Ethernet extension and a COM port

Figure 3.2: The Cyclone EP1C6 FPGA (*left*) and BaseIO board (*right*)

to communicate through. We claim that these features are necessary for a development environment and for utilizing the full power of the Java language. Also debugging and testing would be

extremely hard as the Cyclone FPGA only has a LED light as an indicator. The interesting specifications for the configuration of JOP on the Cyclone FPGA and BaseIO board that we are using are:

- 512KByte FLASH (for FPGA configuration and program code)
- 512KByte fast SRAM (actually 1MByte but half is reserved for the ”not yet working” Garbage collector)
- Board is clocked with 60MHz CPU
- Cirrus Logic CS8900 10Base-T Ethernet Controller, with RJ 45 connector for Ethernet
- COM-Port interface

The specification of the platform meets the characteristics of an embedded system (see Section 3.1). The platform also enables researchers or developers to make WCET predictions, and from this to derive the necessary hardware specifications for any embedded system[3]. The Platform does not support the audio capabilities of a signal recording (microphone) or playback of audio (loudspeaker). This is therefore a requirement that the SDK need to extend to enable a platform to support speech recognition systems.

[3]In Chapter 8 we will show how the WCET times of the speech recognition algorithms are derived

Chapter 4

Related Work

JopSpeech is the first speech recognition SDK developed for JOP, but others also exist for various other platforms. In this chapter we will give a short introduction to some of the most prominent SDK's for embedded and desktop applications. It should be noted that JopSpeech is the first Java developed speech recognition SDK for embedded systems. The SDK's and systems described in this Chapter do not constitute an exhaustive list, but are representative for the market. We are aware that only a small part of the many different commercial and free systems that are available on the market are mentioned here, but since they are all proprietary, we feel the mentioned systems are sufficiently representative.

4.1 Desktops SDKs

4.1.1 Java Speech API (JSAPI)

JSAPI is Sun Microsystems's official Java speech API specification, which can be found at `http://java.sun.com/products/java-media/speech/forDevelopers/jsapi-doc/index.html`. The Java Speech API defines a cross-platform API to support command and control recognizers, dictation systems and speech synthesizers. The Java Speech API (JSAPI) consists of two parts: the *javax.speech.recognition*, which specifics the interfaces for the recognition part, and the *javax.speech.synthesis*, which specifics the interfaces for the synthesis part. JSAPI is not part of the JDK and Sun does not ship an implementation of JSAPI. Instead they have left the actual implementation to other companies. Companies which have made an actual implementation that more or lesser follows the JSAPI can be found on Sun's web site, [1] among these are:

[1] `http://java.sun.com/products/java-media/speech/forDevelopers/jsapifaq.html`

FreeTTS An Open Source speech synthesizer implementation of the JSAPI (javax.speech.synthesis).

ViaVoice JSAPI interface implementation based on IBM's ViaVoice product, which supports continuous dictation, command and control, and speech synthesis.

Sphinx Sphinx is a open source speech recognition system written entirely in Java with interfaces to the recognition part of JSAPI. Sphinx was created via a joint collaboration between the Sphinx group at Carnegie Mellon University, Sun Microsystems Laboratories, Mitsubishi Electric Research Labs (MERL) and Hewlett Packard (HP), with contributions from the University of California at Santa Cruz (UCSC) and Massachusetts Institute of Technology (MIT)[2]. Sphinx is the java project most relevant for us, since the code is released under Open source. An embedded version of Sphinx (Pocketsphinx) also exists but this is written in C instead of Java.

4.1.2 Microsoft Speech SDK

Microsoft Speech SDK provides state of the art continuous speech recognition and text-to-speech engines, a rich set of tools, sample source code and information needed for developing speech-enabled applications for Microsoft Windows[3]. Microsoft has recognized speech recognition as an upcoming business area and will provide extended speech recognition functionality in their new OS Microsoft Vista.

4.1.3 Dragon NaturallySpeaking SDK

Dragon NaturallySpeaking SDK Client Edition 9 allows developers and systems integrators to power Microsoft Windows applications with the interactive dictation features available in the award winning Dragon NaturallySpeaking software. Improved accuracy is 20% higher than the previous release. It can be integrated into medical records applications, CRM applications, legal practice management applications or any other workflow application. It can be developed with any language that supports Active X, including $C++$, $C\sharp$ and Visual Basic. It also comes with comprehensive sample code, documentation and supports for six languages.

[2] `http://cmusphinx.sourceforge.net/html/cmusphinx.php`
[3] `http://www.microsoft.com/speech/default.mspx`

4.2 Embedded SDKs

4.2.1 Embedded ViaVoice

IBM's Embedded ViaVoice Multi-Application SDK is a software development kit (SDK) for embedded speech that allows multiple applications, simultaneous access to Automatic Speech Recognition (ASR) and Text-To-Speech (TTS) resources on an embedded device.

The current Embedded ViaVoice (EVV) SDK is a set of libraries and header files in C++ designed to be linked into a single speech application. The EVV Multi-Application SDK is intended for use in hand-held platform environments such as PDA's (IPAQ is an example currently running on ViaVoice). The SDK will allow independent application developers to create applications and deploy them onto an end-user system. The applications will be able to share and have coordinated use of the speech resources installed on the embedded device.[4] The EVV is currently using approximately 18mb of memory and can run with a sampling frequency of 11kHz and a sample value range of 16bit.

IBM also has a Desktop ViaVoice SDK which is implemented in Java following JSAPI4.1.1, and as a result does not support cross platform communication between the two SDK's yet.[5]

4.2.2 Pocket Sphinx

PocketSphinx is a version of the Open Source Sphinx speech recognition system programmed in C which runs on hand-held PDAs. The Sphinx speech recognition project has been the most influential source for this master thesis as it is an Open Source SDK.

The Sphinx Group at Carnegie Mellon University is committed to releasing the long-time, DARPA-funded Sphinx projects widely, in order to stimulate the creation of speech-using tools and applications, and to advance the state of the art both directly in speech recognition, as well as indirectly in related areas, including dialog systems and speech synthesis.

The Sphinx Group has been supported for many years by funding from the Defense Advanced Research Projects Agency, and the recognition engines to be released are those that the group used for the various DARPA projects and their respective evaluations. The Sphinx system is developed to be very flexible. This means that it is possible to connect different parts of speech recognition in the framework that the Sphinx group has developed[39].

[4] `http://www.alphaworks.ibm.com/tech/evvmasdk`

[5] An implementation of Embedded ViaVoice in Java is, however, frequently requested by customers `http://www.alphaworks.ibm.com/tech/evvmasdk/forum`

4.2.3 Embedded Windows CE *SAPI* from Research Lab

The Embedded Widows CE SAPI is a complete Embedded Speech Recognition SDK for Development of Speech Recognition System (see Figure 4.1). The design is based on speech recognition and can be ported to Windows CE/Pocket PC/Smart Phone/ Symbian OS for Nokia Series 80 and above. The SDK has a memory overhead of 3 MB - for one-sentence dictation. The engine therefore fits the needs of developers looking for porting to the high-end embedded world (such as mobiles, PDA's etc.). [6]The programming language is C and the hardware is equipped with an Ethernet port, 128 Mb memory and a 466 MHz CPU.

Figure 4.1: Embedded Windows CE *SAPI* SDK

The Embedded Windows CE SDK is the only complete set, comparable to JopSpeech, that consists of both a software and hardware part. The cost of the Embedded Windows CE SDK from Research Lab is 799US$. An IDE is required and visual studio is the recommended one.

4.3 Small Devices Featuring Speech Recognition

There are numerous embedded devices that have speech recognition in them, Mobile phones, PDA's, but also decides within larger things like Cars seems to adapt the techniques especially quickly. However on the private speech recognition solution developer market there are actually

[6]`http://www.research-lab.com/wincespeech01read.htm`

very few companies that supplie all the devices with their software. Some of the biggest brands are mentioned here:

VoiceMode VoiceMode is a commercial product developed by VoiceSignal and used in mobile phones to enable the feature to call a person by saying the number or saying a name in the phone book without training of the system[7]

Vocon from Núance Nuance provides a small-footprint DSP-based Automatic Speech Recognition (ASR) solution for developing embedded voice recognition applications that incorporate continuous digit dialing and speaker-independent commands.

Central to the development framework is the VoCon SF engine, a modular speech recognizer. Ideal for driving embedded DSP solutions in automotive and consumer electronics, the VoCon SF engine is available in a full version (<100kb) and a VoCon SF Lite version (<25kb) to meet varying platform requirements. [8]

Voice Command This product is developed by Microsoft and is a commercial product that makes it possible to control different parts of the PDA or Mobile phone by voice and natural speech. There is no training required because the system is speaker independent[9].

SpeechMagic Philips has made this product for use in the medical business. It is used for dictation and transcription of the dictation by speech recognition.[10]

VR stamp Sensory inc. has developed a small speech chip and a development kid that enables a developer to implement a hardware speech recognition system to different other systems. The development language for controlling the VR stamp is Phyton.[11]

4.4 Discussion

The various products described in this Chapter are all products and SDK's for embedded and desktop applications. We can see that all of the speech recognition SDK's for embedded systems are either in C or C++. No implementations of the JSAPI or embedded systems exist in Java. The only solution that exists in Java is for desktop and server environments. This might have something to do with the limitations of Java in execution speed compared to C. However the structure of the

[7] http://www.voicesignal.com/
[8] http://www.nuance.com/vocon/
[9] http://www.microsoft.com/windowsmobile/downloads/voicecommand/default.mspx
[10] http://www.speechrecognition.philips.com
[11] http://www.sensoryinc.com

JSAPI is not really designed for a hard real time system since the JSAPI builds upon a lot of exception handling, which is not proper for real-time embedded systems, where the execution time should be known. Another issue with the implementation of JSAPI is that it is hard to use as described by the developers of Sphinx.[12] A reason why C is the most common language used for speech recognition systems is because of the heavy calculations done in the algorithms used for feature extraction within speech recognition systems. However this is one of the reasons to use JOP as the platform because it executes the Java bytecode quicker then other embedded Java implementations.

Sphinx and The Embedded Windows CE SDK is the two most influential SDK´s to the SDK we are developing in this thesis. Sphinx is written in Java and is open source, and can as such be used for a good indication on what a speech recognition SDK should include. We do not believe that we can make as an extensive library as Sphinx, as the CMU has a historic position in computational speech research and have been developing Sphinx through the last 20+ years. However, using Sphinx as a source of inspiration has been beneficially. The Embedded Windows CE SDK presents in some ways, the same as we have developed in Java, a complete SDK to embedded system with hard and software support. A comparison of the two platforms could therefore be interesting in the future.[13]

[12] `http://cmusphinx.sourceforge.net/html/cmusphinx.php`
[13] This is out of the scope of this thesis.

Chapter 5

Theory

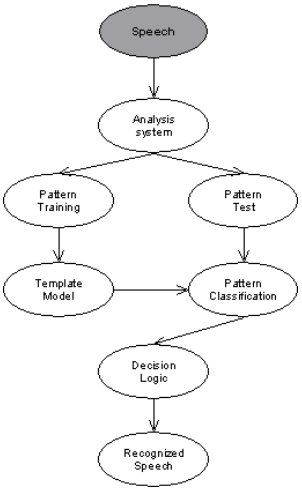
When making an speech recognition SDK several components have to be available, as developer‘s and researcher‘s development goals might differ. The goal of their system might be Speaker-independent or Speaker-dependent, small or large vocabulary, distinct or continuous speech recognition. There might be a lot or few hardware constraints in development. As a result the components available need to enable the developer or researcher to construct the systems as they see fit.

This chapter will go through the basic of Signal Processing and Machine Learning Theory which is will form the requirements to complete a speech recognition SDK, as presented in this thesis. Concepts like Time-Domain, Frequency-Domain, Power Spectrum, Cepstrum, Mel-Frequency and Linear Predictive Coding will be explained.

The first section starts with the most important concepts in digital signal processing theory in respect to speech recognition. Then the machine learning principals of pattern recognition theory / techniques in respect to speech recognition will be explained.

5.1 Frontend

In this section, the various techniques used in digital signal processing (DSP), will be analyzed in the context of speech recognition. The processing done on the signal from the analog signal converted into something digital, to the cleansing and filtering of the signal to insure that only the relevant signal is being analyzed will be the main focus.



5.1.1 Sampling

The first step in working with sound samples is to convert the analog signal that is produced by human speech into a digital form. When sampling we normally talk about two different variations[11], the amplitude/time-continuous *un-quantified signal* and the discrete *quantified signal*. The information carrying signal that is released during the speech production Subsection 2.1.1, is transmitted from the person in a time-continuing *un-quantified* signal $x(t)$ and can as such take any amplitude value at any given time. An amplitude-*discrete* signal is on the other hand characterized by having an amplitude value that takes on a finite set of values[1]. A time-discrete signal is in the same way as a amplitude-discrete signal characterized by only having a finite set of times[2].

The sampling consist of converting the analog signal into series of discrete amplitude values in discrete time defined by the sample resolution. This is in practice done by through an analog-to-digital (A/D) converter. When the sound produced by speech is recorded through a microphone the air-pressure - generated from the sound signal, is passed through a flexible diaphragm (or ribbon in the case of ribbon microphones)[3]. The vibrations of the diaphragm are then converted by various methods into an electrical signal with an infinite set of values. The Analog to Digital Converter (ADC) then converts the voltages into a discrete finite set of values defined by the sample resolution or precision of the conversion[36].

The primary concern in sampling is at what frequency a given continuous signal must be sampled (f_s) and at what resolution, in order to preserve its information. The simple answer to ”at what sample frequency” is that a sampling rate has to be double the value of the highest frequency (f_0) we are examining. That is, we are interested in signal components that are aliased into the frequency band between $-f_s/2$ and $f_s/2$ - referred to as the ”half Nyquist” or ”critical Nyquist”. This

[1]The experimental prototype on the JOP board only have amplitude values between -1250 to 1250

[2]The discrete values in the experiments is defined to be 8000 times pr second = 8000Hz

[3]for more information on microphones see examples at `http://en.wikipedia.org/wiki/Microphone`

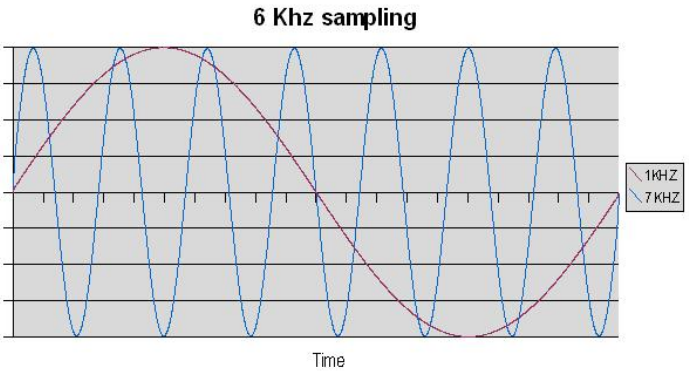


Figure 5.1: 7kHz sin wave depicted at a 6kHz sampling

means if we are examining the frequency values up to $3000Hz$ our highest frequency would have to be $6000Hz$. The reason for $\pm f_s/2$ is that there is a Frequency-Domain ambiguity associated with the discrete-time signal[20].

A Time-Domain sinusoidal signal is defined as:

$$x(t) = sin(2\pi f_0 t) \tag{5.1}$$

Let us imagine that we were given the following sequence of samples and were told that these represented discrete values of a Time-Domain sine wave (see Equation 5.1) taken at periodic intervals.:

$$x(0) = 0)$$
$$x(1) = 0.866$$
$$x(2) = 0.866$$
$$x(3) = 0$$
$$x(4) = -0.866$$
$$x(5) = -0.866$$
$$x(0) = 0$$

Next we were to plot them into a graph, how would we then do this? Having only the sampling values but not knowing the sampling frequency, there would be two different ways we could do this, and both ways would be correct (see Figure 5.1). What we have depicted in Figure 5.1 is a 7000Hz sine wave depicted at a sampling frequency at f_0 6000Hz. As can be seen in Figure 5.1 this gives two potential sin waves. If we convert the amplitude signal from the Time-Domain into the frequency spectra and try and analyze this signal, we cannot say if the power is in the 1kHz

spectrum or the 7kHz spectrum. This is the ambiguity that exists between the Frequency-Domain associated with the discrete-time signal. It is also why any given signal we want to analyze has to be below $\pm f_s/2$. This means for speech recognition that our sampling frequency has to be above approx 6800Hz as the most important frequencies are represented in approx 200-3400Hz[25][4]. And we have to take into account that, since the signal is a discrete sampling of the original continuous signal, frequencies that are above the "half Nyquist" will not be "visible" and will therefore by leaking into the other frequencies[20] [11].

The second concern in sampling is the quantization - or how many bits are used to quantify the input signal into discrete levels without loosing its information value. Since memory usage is a concern for embedded systems using only the minimum is a prerequisite. Imagine a microphone that produces voltage between 0 and 2 V and the sample resolution is 8 bit it would give a integer value between 0 - 255 whenever we measured the voltage of the microphone. A voltage of 2 would be a integer value of 255 and a voltage of 1 would be 128. The formula for this would be:

$$bitValueMax = maxVoltage * ResUnit \Updownarrow$$

$$ResUnit = bitValueMax/maxVoltage \Downarrow$$

$$Value = Res_Unit * Voltage$$

The problem with a finite set of values of only 8 bit is that the small changes in voltage can not be depicted, 1.999 V would still be translated into 255 although there is an amplitude difference.

An ambiguity therefore exists between the quantization and memory constraints on a embedded system: Since quantization of a sample adds noise to the analog signal with a maximum error of $+\_LSB$ (Least Significant Bit the distance between adjacent quantization levels) and a standard deviation of $\frac{1}{\sqrt{12}}LSB$[11]. For example, using an 8 bit quantization on a analog signal adds an rms noise of: $\frac{0.29}{256}$, or about $\frac{1}{900}$ of the full scale value, while a 16 bit conversion adds: $\frac{0.29}{65536} = \frac{1}{227000}$. Choosing a to low resolution - called low quantization[3], will result in a more noise, while choosing higher representation will result in higher memory usages. For example 32 bit might be a better resolution but would take op 4 times the memory of the 8 bit resolution. According to research conducted by Coleman [3] the normal representation used in speech recognition is approx 12 bit, this can though vary with the context and specifics of the usages of the developed system.

[4]This holds true for our prototype, as we are sampling with 8000Hz

5.1.2 Sound Filter

The filters two main functions are to filter out noises / frequency that are disturbing the sound picture of speech for example, or to segment a signal‘s frequencies into several banks. The choice of filter frequencies range to pass through is considered in regards to the context that the system has to work in. For example, in Subsection 5.1.1 we mentioned that speech is normally between 200-3400Hz [25],this could then be the filters pass band range. Filtering a single band out between the overall frequencies could also be necessary, in Chapter 2 we mentioned that systems where dependent on their physical context, so if we, for example, need to develop a speech recognition system for a washing machine we want to filter out the low or high frequency sounds normally produced by this or a dry tumbler standing next to it. Different from that could be noise frequencies represented in signals recorded in a car, or in a kitchen. Filters can also be used to exclude several unwanted signals e.g. sneezes, hisses, breathing, etc.

The primary reason for filtering in speech recognition systems is to get a more precise/enhanced representation of the speech signal - be it represented as a phoneme, word or sentence by filtering out noise or frequency bands where the speech is not represented. Enhancing features of the sound signal that could be an essential part/feature/property of a sound used to train a given machine learning algorithm would inevitably lead to a better prediction.

Although the algorithms and methods might vary, filters can be categorized after their intention into four different types, or a combination or these. Visualized by Figure 5.2, where the ω in the figure is the frequency and the $H(\omega)$ is the function of the amplitude numerically.

Low pass: Lets the desired low frequency pass through the filter.

High pass: Lets the desired high frequency pass through the filter.

Band pass: Lets frequency in a desired range pass through the filter.

Band stop: Lets frequency outside the desired range pass through the filter.

The two primary digital filters used in DSP are called Infinite Impulse Response (IIR) [5] filter and Finite Impulse Response (FIR)[6] filter[25]. The difference between the two implementations are that the IIR filter uses feedback so when you input an impulse the output theoretically rings indefinitely where the FIR filter is ”finite” after N input coefficients and doesn’t affect the next samples [25, 3].

[5] see an example here: `http://en.wikipedia.org/wiki/Infinite_impulse_response`
[6] see an example here: `http://www.dspguru.com/info/faqs/firfaq.htm`

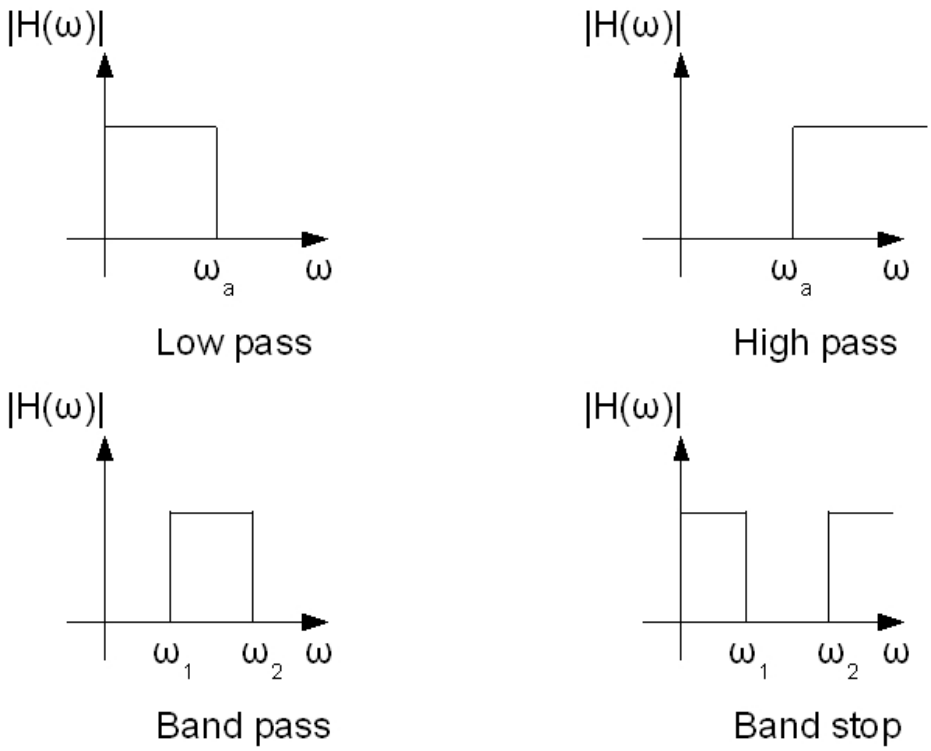


Figure 5.2: The four filter types [11]

A simple FIR filter is a running means 5.2 of the signal S for a given period N.

$$S_i = 1/|N_i| \sum_{j \in N_i}^{I} y_j \tag{5.2}$$

The running mean filter will filter out high frequencies and is therefore referred to as a low-pass filter. The disadvantages of the running mean is that it tends to flatten trends near the endpoints and does not smooth very well in the middle[3]. Any filter is defined by the order of the filter and the order is the number of previous samples/given period N that defines the output of the filter. So a sound sample defined in time series where N is time and X is the sample in 5.3, would give a *first* order filter with Equation 5.4 defined by the present sample and the previous sample.

$$x_1, x_2, x_3, ...x_n \tag{5.3}$$

$$y_n = \frac{x_n + x_{n-1}}{2} \tag{5.4}$$

An *second* order filter uses the two previous samples, with the equation:

$$y_n = \frac{x_n + x_{n-1} + x_{n-2}}{3} \tag{5.5}$$

A IIR filter uses previous output for calculating the new output (see Equation 5.6) said to be feedback or recursive [25, 3]. This is a advantaged compared to the FIR filters which require more memory and calculations to perform the same operation.

$$y_n = x_n + y_{n-1} \tag{5.6}$$

The more general form of the IIR filter is define by coefficient for both the output(y) and the input(x) by *b* and *a*:

$$b_0 y_n + b_1 y_n - 1 = a_0 x_n + a_1 x_{n-1} \tag{5.7}$$

If we compare the two types of implementations FIR and IIR, it is evident that the IIR filter has its advantages in less memory consumption and can be more computational efficient[11]. The advantages of the FIR filters is that they are more simple to implement and they display much more desirable numeric properties so integer/fixpoint arithmetic is plausibly, where IIR filters might loose a lot of information due to difficulties in preserving decimals.

Filtering also works for feature extraction of the signal - called filter banking, a technique used very often in the acoustic phonetic approach Section 2.2. Filter banking works by dividing a signal into frames and then measuring the spectral properties of this signal (see Figure 5.2). Each filter is called a channel and is a band pass filter that only allows sound for the given band pass to slip through. The channels can then be represented by one discrete finite value - instead of all the sample values in the given channel, using codebanks (see Subsection 5.3.4). The value for each band is measured in energy in the channel. This approach also have the advantage that it reduce the amount of memory needed to represent a given signal [11].

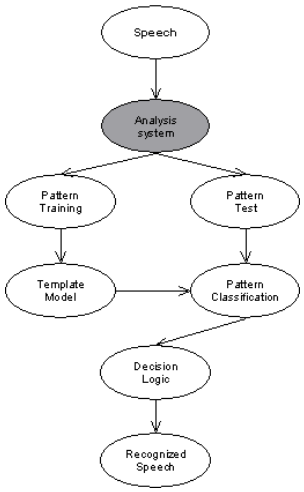
The filter bank can be made of uniformed channels or by non-uniformed channels. A uniformed filter bank is where every filter channel has the same range of frequency. A non-uniformed uses different range, for example, raising logarithmic in scale, or by using the popular Mel Frequency scale, that has shown to be a very powerful method for speech recognition. As the Mel Scale is the same scale that the humane ear is perceiving [7, 33, 35, 6] (see Subsection 5.2.5).

5.2 Speech Analysis

In this Section the basic theory, relevant for making a sufficient description of the signal to secure a good prediction model or minimum estimation error rate when classifying, will be described. In Machine Learning terminology this subject is sometimes referred to as the *cleansing of Data* or *pre-processing of Data* [9].

5.2.1 Fourier Transform

Jean Baptiste Joseph Fourier 1768-1830 was the ”inventor” of one of the most dominant and widespread mathematical mechanisms available for the analysis of physicals systems today[20], known as the ”Fourier Transformation”. The Fourier Transformation is a mathematical procedure to determine the frequency and harmonics in a signal. One of the fundamental parts of signal processing is to transform a time domain signal into the frequency domain. However until now we have not yet discussed how the Time-Domain waveform of a signal is converted into its Frequency-Domain. Since this is a very important and one of the mathematically heavy parts of DSP, we are now going to give a brief discussion of this algorithm.



In signal processing, operations are chosen to improve some aspects of the signal quality by exploiting the differences between the signal and the corrupting influences. When the signal is a sinusoid corrupted by additive random noise, spectral analysis distributes the signal and noises components differently, often making it easier to detect the signal's presence or measure certain characteristics, such as amplitude and frequency. Effectively, the Signal to Noise Ratio (SNR) is improved by distributing the noise uniformly, while concentrating most of the sinusoid's energy around one frequency. Processing gain is a term often used to describe an SNR improvement. Taking a discrete signal and converting it into its frequency domain, the Discrete Fourier Transform (DFT) is the most commonly used algorithm. The DFT origins from the continuous Fourier transform $X(f)$ which defined as:

$$X(f) = \int_{-\infty}^{\infty} x(t)e^{-j2\pi ft}dt \tag{5.8}$$

Where $x(t)$ is some continuous Time-Domain signal, $2\pi ft$ can be recognized from Equation 5.1 and $-j$ is here to indicate its complex number origin where it of course is $j = \sqrt{-1}$

In the field of continuous signal processing Equation 5.8 is used to transform an expression of a continuous Time-Domain function $x(t)$ into a continuous Frequency-Domain function $X(f)$. Subsequent evaluation of the $X(f)$ expression enables us to determine the frequency content of any practical signal of interest and opens up an array of signal analysis and processing possibilities. The DFT is a Fourier Transformation in the discrete time domain, derived from Equation 5.8 and is defined by Equation 5.9.

$$X(m) = \sum_{n=0}^{N-1} x(n) W_N^{mn}, 0 \leq m \leq N-1 \tag{5.9}$$

where

$$W_N = e^{-j2\pi/N} \tag{5.10}$$

The output of the DFT equation is series of complex numbers, the real part Equation 5.11 and the imaginary part Equation 5.12.

$$ReX(m) = \sum_{n=0}^{N-1} x(n) \cdot \cos(2\pi mn/N) \tag{5.11}$$

$$ImX(m) = -\sum_{n=0}^{N-1} x(n) \cdot \sin(2\pi mn/N) \tag{5.12}$$

The DFT can therefore be expressed by the following (rectangular form), where we have spilt Equation 5.9 into its real and imaginary part.

$$X(m) = ReX(m) + ImX(m)$$

$$\Downarrow$$

$$X(m) = \sum_{n=0}^{N-1} x(n)[\cos(2\pi mn/N) - j\sin(2\pi mn/N)] \tag{5.13}$$

Having:
$X(m) =$ the mth DFT output component, i.e., X(0),X(1)
$m =$the index of the DFT output in the frequency domain ($m = 0, 1, 2, 3..N-1$).
$N =$ the number of samples of the input sequence, and the number of frequency points in the DFT output.
$x(n) =$ the sequence of input samples ,$x(0), x(1)...$
$n =$the time domain index of the input sample.
$j = \sqrt{-1}$.

Example with DFT

We can exemplify with the DFT. Imagine that we want to look at the first 4 waveform samples in a sound signal and see how the frequency distribution is in this distribution. Thus the following is given:
$N = 4$
$m = 0, 1, 2, 3$
$n = 0, 1, 2, 3$
Which results in the following formula:

$$X(m) = \sum_{n=0}^{3} x(n)[\cos(2\pi mn/4) - j\sin(2\pi mn/4)] \tag{5.14}$$

Doing the math on all the terms for the first DFT term, with $m = 0$ corresponds to the following:[7]

$$\begin{aligned} X(0) = x(0)\cos(2\pi 0 \cdot 0/4) - jx(0)\sin(2\pi 0 \cdot 0/4) \\ +x(1)\cos(2\pi 1 \cdot 0/4) - jx(1)\sin(2\pi 1 \cdot 0/4) \\ +x(2)\cos(2\pi 2 \cdot 0/4) - jx(2)\sin(2\pi 2 \cdot 0/4) \\ +x(3)\cos(2\pi 3 \cdot 0/4) - jx(3)\sin(2\pi 3 \cdot 0/4) \end{aligned}$$

We have omitted the next three equations when $m = 1, 2, 3$ as they are the same (just change the n value). Each $X(m)$ output term is then a product of both an input sequence of signal values (we just used $x(0), x(1), x(2), x(3)$ instead of exact values in this example), and a complex sinusoid of the form $cos(\phi) - j\sin(\phi)$. The exact frequency of the different sinusoids depends on the rate they originally have been sampled at. If for example the sampling rate were $f_s = 100Hz$, the fundamental frequency of the sinusoids would have been $f_s/N = 100/4$ or $25Hz$. The $X(0)$ term in this example is then a value for the magnitude of any $0hz$ component contained in the input signal, and the $X(1)$ term is a magnitude example for any $25Hz$ component. The $X(2)$ term is the magnitude for any $50Hz$ component and so on.

The DFT output term can also be used to determine the Power (magnitude squared) and the Phase. Let us take an arbitrary value for X(m), then the magnitude ($X_{mag}(m)$) and the phase angel ($X_\phi(m)$) are defined by the following:

$$X(m) = X_{real}(m) + jX_{imag}(m) = X_{mag}(m) \texttt{ at an angel of } X_\phi(m)$$

[7] we use $\cdot$ instead of $*$ to separate the numbers as it dose it better

The magnitude of $X(m)$ is:

$$X_{mag}(m) = |X(m)| = \sqrt{X_{real}(m)^2 + X_{imag}(m)^2}$$

The Phase angel of $X(m)$ is:

$$X_{\phi}(m) = \tan^{-1} \frac{X_{imag}(m)}{X_{real}(m)}$$

Power or "Power spectrum" is:

$$X_{ps}(m) = X_{mag}(m)^2 \tag{5.15}$$

Since the Power Spectrum is quite a large number, the spectral representation is normally done in a $log10$ or $10 * log10$ depiction[3][25][11] defined as:

$$Spectral(X_{ps}) = Log10(X_{ps}) \tag{5.16}$$

Let us calculate the magnitude, phase and power for $40hz = X(2)$. Keeping nearly the same values as before but with 5 samples $x(n)$: $\{x(0) = 0, \quad x(1) = 10, \quad x(2) = -10, \quad x(3) = 1, \quad x(4) = 2\}$.

$$\begin{aligned}
X(2) = &\quad 0\cos(2\pi 0 \cdot 0/5) && -j \cdot 0\sin(2\pi 0 \cdot 0/5) \\
+ &\quad 10\cos(2\pi 1 \cdot 1/5) && -j \cdot 10\sin(2\pi 1 \cdot 1/5) \\
- &\quad 10\cos(2\pi 2 \cdot 2/5) && -j \cdot -10\sin(2\pi 2 \cdot 2/5) \\
+ &\quad 1\cos(2\pi 3 \cdot 3/5) && -j \cdot 1\sin(2\pi 3 \cdot 3/5) \\
+ &\quad 2\cos(2\pi 4 \cdot 4/5) && -j \cdot 2\sin(2\pi 4 \cdot 4/5) \\
X(2) = &\quad 0 \cdot 1 && -j \cdot 0 \\
+ &\quad 10 \cdot -0.8090 && -j(10 \cdot 0.5878) \\
- &\quad 10 \cdot 0.3090 && -j(-10 \cdot -0.9511) \\
+ &\quad 1 \cdot 0.3090 && -j(1 \cdot 0.9511) \\
+ &\quad 2 \cdot -0.8090 && -j(2 \cdot 0.5878) \\
X(2) = &\quad \underline{12.49} && \underline{-j15.16}
\end{aligned}$$

Magnitude at Phase angel

$$X_{mag}(2) = \underline{19.65 \angle \ -50.52^{\circ}}$$

Power spectrum and log10 Power spectrum

$$\begin{aligned} X_{ps}(2) = &\quad \underline{385.93} \\ Spectral(X_{ps}) = &\quad \underline{Log10(385.93)} \\ Spectral(X_{ps}) = &\quad \underline{2.5865086} \end{aligned}$$

The DFT is a very strong mathematic tool, but it contains some bad aspects, because it is very computational hard. The implementation of the DFT Equation 5.9 takes N^2 calculations of multiplication and additions which make it a computationally inefficient and time-consuming task. This makes the DFT unfit for use in an embedded real-time system where it is essential to minimize the multiplication and addition because of the small processor size and also due to increased power consumption. Imagine we have to take the DFT on $N = 8000$, this equals $64.000.000$ calculations. Luckily John W. Tukey and James W. Cooley published an algorithm in 1965 called the Fast Fourier Transform (FFT), which is an efficient algorithm for computing the Discrete Fourier Transform (DFT). While the implementation of the DFT (see Equation 5.9) takes N^2 calculations, the implementation of FFT takes only $Nlog_2N$. This means that for $N = 8000$, the number of calculations is only 103.726. This is a massive improvement over the DFT algorithm, as it only takes 0.16% of the time of a DFT.

Fast Fourier Transform

The Fast Fourier Transform (FFT) is a powerful algorithm that transforms a signal in the time domain into the frequency domain like the DFT. The FFT was developed by John W. Tukey and James W. Cooley and is a algorithm that made the calculation of the Fourier Transformation faster, hence the name Fast Fourier Transformation[3, 25, 8, 40].

The FFT algorithm of Cooley and Tukey has the limitation that the length of the signal has to be in the power of two. This means that N always have to be $\{0, 2, 4, 8, 16, 32, 64, 128, 256, 512....\}$. To insure that the signal is of a finite length and is in a power of 2 the use of a "Window" function (see Subsection 5.2.3)to split the original signal into parts of the power of two.

The FFT algorithm works by splitting the signal into smaller signals and performing a DFT on the sub-signals. This "divide and conquer" approach, obtains its computational gain by dividing the problem into sub-problems. Then the DFT is used on these signals and added to give the finale Fourier Transformation. It is this rewriting of the DFT algorithm into two N/2 DFT's that gives the improved performance of $Nlog_2N$.

The simplest implementation of the FFT algorithm is called Radix-2[8, 40]. The radix-2 FFT

recursively partitions a DFT into two half-length DFTs of the even-indexed and odd-indexed time samples. Then the Radix-2 gain its speed by reusing the results of the smaller computations to compute the multiple DFT frequency outputs[8]. This FFT algorithm is defined in Equation 5.17 where $y(n) = 2n$ equals the even index and $z(n) = 2n - 1$ equals the odd-index:

$$X(m) = \sum_{n=0}^{N-1} x(n) W_N^{mn} = \sum_{n=0}^{\frac{N}{2}-1} y(n) W_{N/2}^{mn} + W_N^m + \sum_{n=0}^{\frac{N}{2}-1} z(n) W_{N/2}^{mn} \tag{5.17}$$

5.2.2 Cepstrum

The cepstrum was first described by Bogert, Tukey et al.[2] in 1963 as a heuristic technique for finding echo arrival times in a composite signal. These authors defined the cepstrum of a function, as the power spectrum of the logarithm of the power spectrum of that function and introduced the following paraphrased terms according to a syllabic interchange rule:

$$\begin{aligned} frequency &= quefrency \\ spectrum &= cepstrum \end{aligned}$$

Bogert, Tukey et al.[2] showed that identifying the frequencies of a echo by calculating the spectrum of the log spectrum wherein the echo‘s frequency will appear as a peak. In speech recognition this technique is used to give information about the rate of change in the different spectrum bands, used to pitch tracking where the pitch will appear as peaks . Let us start by describing the use of the frequency, then it is easier to understands the quefrency‘s usage in speech recognition.

Representing a sound sample in its frequency domain is done by taking the Fourier transform to a short representation/time segment of a sound sample (as just discussed in Subsection 5.2.1. Converting the sound sample back from the frequency domain into the time domain is a simple operation of taking the inverse function to a Fourier Transformation. We can see the representation of a signal frequencies as a ripple in the power spectrum (see Equation 5.17). Where the spectrum is a depiction of the actual power in the signal at a given frequency - or within a frequency band (see Figure 5.3) where the clean peaks are the formant frequencies of the sound sample. However there is a problem with estimating the formants directly from the Spectrum, as can be seen in the Figure 5.3. The spectrum in Figure 5.3 contains two types of peaks. First: all the small peaks that are the *harmonics*[3], second: the larger peaks, which represent the resonances of the vocal tract (see Chapter 2)[25]. The problem in finding the resonances in the vocal tract is impaired by the

[8]Like all FFTs

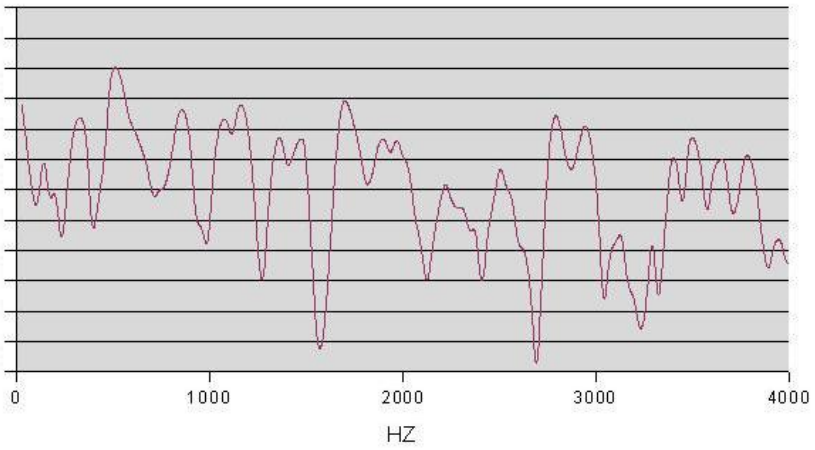


Figure 5.3: Power spectrum

smaller peaks (harmonics), where the harmonic peaks coincide with the resonances. We can see a fairly well defined formant frequency, but if the frequencies are a bit off the signal, they will be cluttered making it difficult to find the formants. The goal is therefore to separate the resonances from the harmonics - using Cepstral analysis.

If we imagine that the harmonics are the fast frequencies and the resonances the slow ones, then taking an FFT to the signal would separate these two signals and thereby enabling us to get a good estimate of the pitch. Separating the underlying resonances from the harmonics, is then done by taking the power spectrum of the FFT signal (As defined by Bogert, Tukey et al.[2]). However since we are already in the frequency domain taking the power spectrum to the signal would lead us into a time domain - not real time but a quefrency Time-Domain [2]. On Figure 5.4 it can be seen that

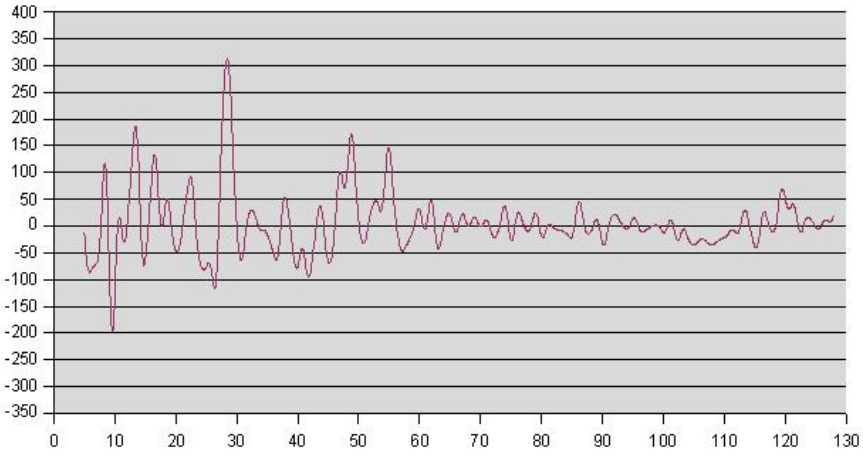


Figure 5.4: Quefrency

there is one sharp spike around 30ms. into the signal (actually at quefrency 29ms), corresponding

to a frequency of $34,5Hz$ which is the fundamental frequency - the frequency of voicing - for this sound sample. Calculated by the duration of one sample (at 8000 Hz) which is $0,125ms$ so the quefrency is the duration (which was 29) divided with $0,125ms$. This equals: $0,125 \cdot quefrency = 29ms => quefrency = 232$ or a quefrency of 232 leading to the fundamental frequency of the sample of $8000/232 = 34,5Hz$

5.2.3 Windowing

The processing gain of spectral analysis depends on the window function, as mentioned in Subsection 5.2.1, so we will focus on the aspects of some of the window-functions in this subsection. The spectral analysis is dependent on perceiving the incoming signal as a series of sinusoids. Therefore the common task of all the different window functions is to cut the infinite impulse-response function, so that it grows and declines varying with time, which makes the sinusoids‘ signals seem complete. If we look at Figure 5.5 we can see how a window adapts a sinusoid to fit within its borders, without its signal being corrupted.

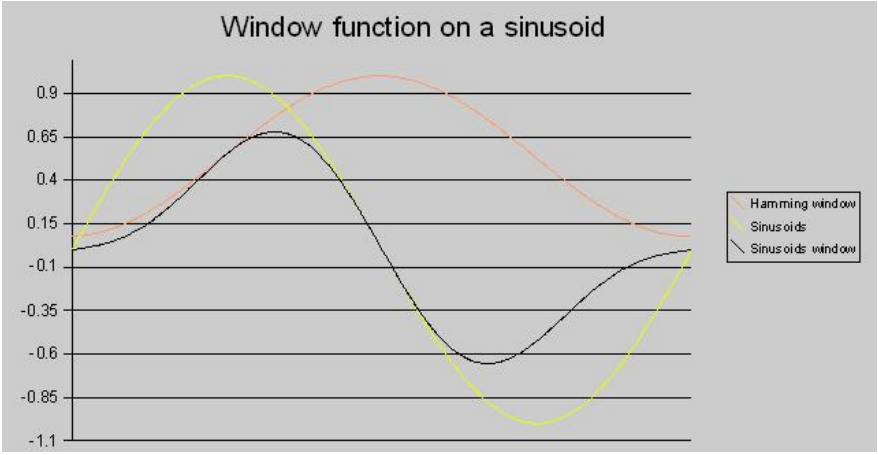


Figure 5.5: Two sinusoid, one fitted within a Hamming window

The sinusoid in Figure 5.5 is also a special kind, where the ends perfectly meet each other. If we instead look at a real world example like the word ”Zero” in Figure 5.6, the signals are not always equally as kind. Let us imagine we would like to find the frequency spectrum, between sample 200 and 600 in Figure 5.7 we can see that the start and end do not meet. This affects the FFT as the frequencies spectrums‘ will be corrupted. The way the FFT is calculated, the frequencies‘ magnitude is calculated on the signal as whole - this means that the energy that lies between and before sample number 200 and after sample 600 will affect the frequency measures. The corruption can however be improved by the introduction of the window function, by seeing

the signal as a finite series of sinusoids with start and end of the signal being conjoint [11] [20] and imagining that the sinusoids have an amplitude of zero on both sides of the ends (no zero crossings is actually the important). As can be seen in Figure 5.7, the window function has now joined the

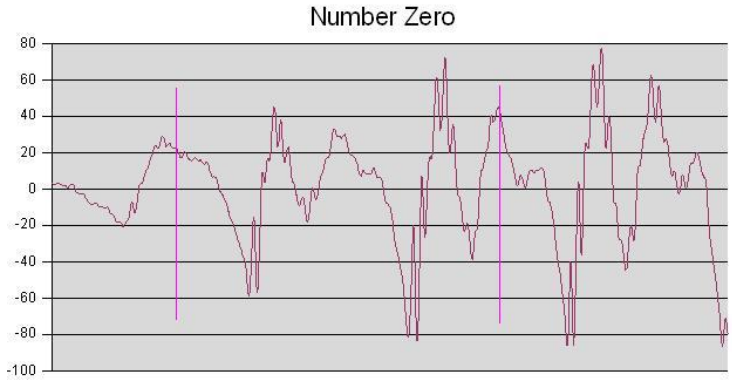


Figure 5.6: the first 1000 hz samples in the signal "Zero"

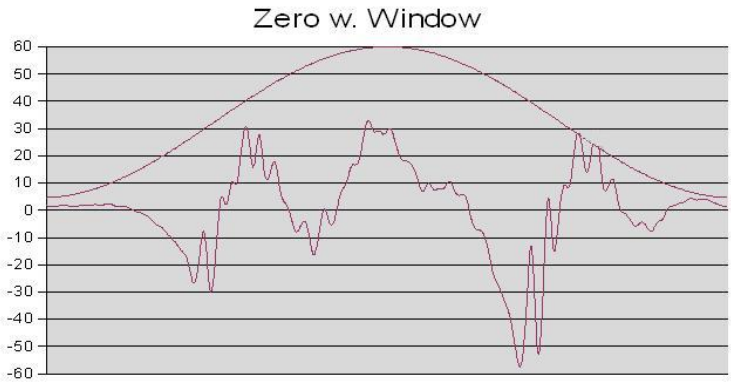


Figure 5.7: 400 samples from the number "Zero" with a hamming window

ends together to make them look like a conjuncted sinusoid, and the amplitude has decreased to zero in the ends. The window function used above in Figure 5.7 is called "hamming window" based upon its creator "Richard Hamming", however several different window functions do exists. Some of them are reprinted here in Figure 5.8 and Figure 5.9, and we will look through a couple of the most important ones which can be seen in the figures. On the left side of the figures is the window function and on the right side the amplitude spectrum it covers[9]:

Hamming Window A Hamming Window is a general function window with a narrow main amplitude lobe and small max value on the side lobes. The algorithm is:

$$w(n) = 0.53836 - 0.46164\cos(\frac{2\Pi n}{N-1})$$

[9]More information and depiction of different window functions can be found in [26]

HANN Window The HANN Window is like the Hamming window a general function window, but with a broader main amplitude lobe and instead a much smaller max value on the side lobes. The algorithm is:

$$w(n) = 0.5(1 - \cos(\frac{2\Pi n}{N-1}))$$

Bartlett Window The Bartlett window is triangular, with both a broad main lobe and large max value on the side lobes. The algorithm is:

$$w(n) = \frac{2}{N-1} \cdot (\frac{N-1}{2} - \mid n - \frac{N-1}{2}) \mid$$

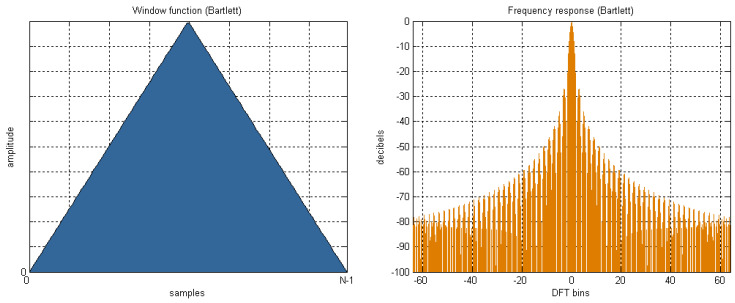


Figure 5.8: Bartlett window

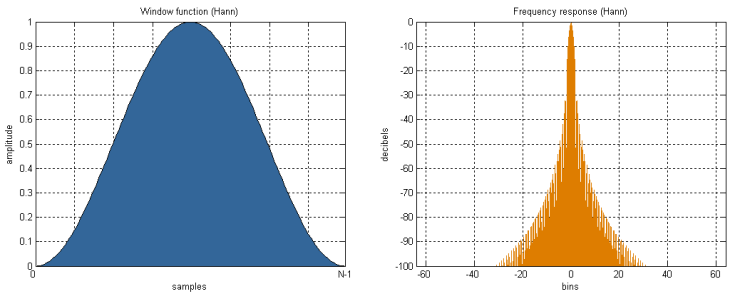


Figure 5.9: Hann window

As you can see in Figure 5.8 and Figure 5.9 the window functions work in different ways, letting more or less amplitude through around the main lobe (the central amplitude on the right side of the figures). Different aspects of window shapes have been researched regarding the effects they have on speech recognition[26][16]. Robert Rozman[26] has proved that the use of a non linear window function has an improved effect on speech with a lot of noise, whereas the use of rectangular windows (special kind of window function where $w(n) = 1$) is better in clean

environments [26]. Hann or Hamming windows have a more general effect on the signal since the amplitude spectrum is more general, where the Hamming window is the most used window function[25][26].

5.2.4 Linear Prediction Coding

Representing the speech signal in a clean frequency or time domain representation is quite a memory intensive task, and, as such, not always an optimal solution for an embedded system. Consider for instance a $16bit$ $8000Hz$ sampling. This sample would need $8000Hz * 16bit = \underline{128000bit/sec}$. With a need for 128 Kbit or 16 Kbyte per second, we would have used all our available memory on JOP after just 32 seconds, since $16kbyte/sec * 32sec = \underline{512kbyte}$. This amount of memory consumption is not desirable, as the amount of memory consumption affects embedded system considerable. This is where Linear Prediction Coding (LPC) comes is being used, as a good way of compressing the signal.

The basic idea behind an LPC model is that a given speech sample at time $n, s(n)$ can be approximated as a linear combination of the past p speech samples, plus a gain factor[25]. The LPC algorithm can therefore be writting like this:

$$s(n) = \sum_{i=1}^{p} a_k * s(n-k) + G(n) \tag{5.18}$$

If we only consider the linear combination of past speech samples as the estimate $\tilde{s}(n)$, defined as:

$$\tilde{s}(n) = \sum_{k=1}^{p} a_k * s(n-k) \tag{5.19}$$

we can now form the prediction error $e(n)$:

$$e(n) = s(n) - \tilde{s}(n) = s(n) - \sum_{k=1}^{p} a_k * s(n-k) \tag{5.20}$$

The objective is the to optimize the a_k coefficients by minimizing the prediction $e(n)$ for a given sound sample - which is normally achieved through minimizing the squared mean average of $e(n)$ of a given sound segment(Frame) $s_n(m) = s(n+m)$:

$$E_n = \sum_{m} e^2(m) \tag{5.21}$$

The compressing that happens through the use of coefficients instead of all recordings, are still a variable of how many coefficients are chosen. Imagine if we have a one second speech signal, which means we use $128000bit$ in 8000 recordings. Now imagine that we divide this signal into frames of $20ms$ (a speech signal varies slowly over time so this would be very representative), as the signal changes over time. This means that each frame now contains:

$$FrameSize = 8000Hz * 20ms \Rightarrow FrameSize = \underline{160}$$

If we instead of 160 recordings, would have only 20 coefficients, we would achieve a compression size of:

$$Compressionsize = 20/160 \Rightarrow Compressionsize = \underline{12.50\%}$$

This means we have achieved a compression of 87.50% and instead of using $128000bit$ we are now only using $128000bit * 12.5\% = 16000bit$ or $2kbyte$. However perhaps we do not even need 20 coefficients to illustrate the signal properly. If we look at the Figure 5.10, 5.11 and 5.12, we can see that, although more coefficients are used to represent the signal, the last coefficients are not representing the big difference in this signal. This mean that the error E_n will not be considerably higher if the last coefficients are omitted.

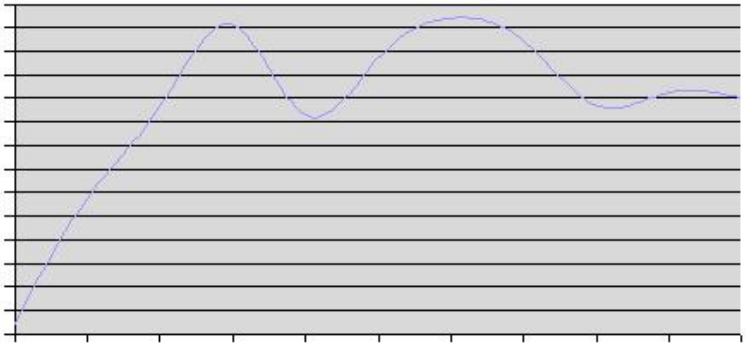


Figure 5.10: LPC with 10 coefficients

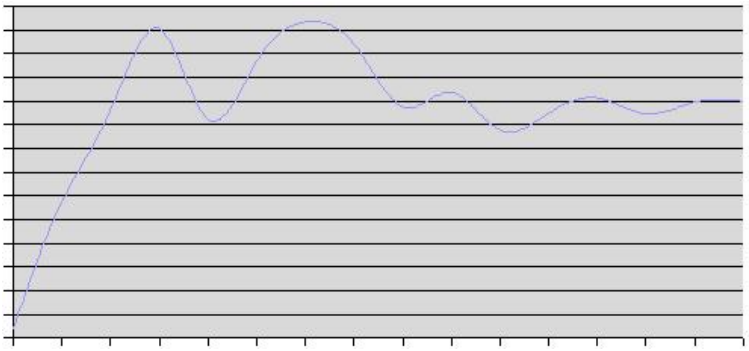


Figure 5.11: LPC with 15 coefficients

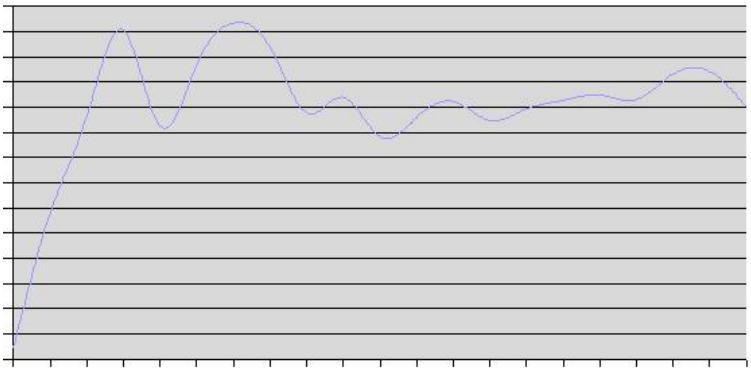


Figure 5.12: LPC with 20 coefficients

Using LPC as a compression of the sound signal has several extra advantages. In the previous Subsection ”Cepstrum”, we concluded that the actual formants were hard to distinguish, due to the cluttering of the signal by the harmonics. What the LPC method actually does is calculating the coefficients from the speech signal by analyzing its formants and its residual (what we have just called error($e(n)$)). Then the LPC system determines the formants from the speech signal, given a certain frame, and makes them the basis of the coefficients. The result of a an LPC are therefore a representation of a frame with a signal cleaned from disturbing harmonics.

5.2.5 Mel Frequency Cepstral Coefficients

The Mel scale, proposed by Stevens, Volkman and Newman in 1937 [37] is a perceptual scale of pitches judged by listeners to be equal in distance from one another. The reference point between this scale and the normal frequency measurement is defined by equating a 1000Hz tone, 40 dB above the listener’s threshold, with a pitch of 1000 mels. Above about 500Hz, larger and larger intervals are judged by listeners to produce equal pitch increments. As a result, four octaves on

the hertz scale above 500Hz are judged to comprise about two octaves on the Mel scale. The name Mel comes from the word melody, to indicate that the scale is based on pitch comparisons[40].

To convert frequency into the Mel scale, the following algorithm is used:

$$Mel(f) = 2595 \log 10(1 + \frac{f}{700})$$

And the inverse (from Mel scale to normal frequency)is calculated:

$$Freq(m) = 700(10^{\frac{m}{2595}} - 1)$$

Using the Mel scale to make a representation of a signal varies. However the most used and simplest, is to use a filter bank solution. With a filter bank solution the power of a signal is divided into a set of banks, depending on then Mel scale distribution will give a power distribution of the signal (see Subsection 5.2.1 for power calculation).

The number of banks are calculated and dependent on; k= sample, M=number of filters, N=Number of samples, H= filter Bank. The power of a signal is calculated using Equation 5.16 into the bank resulting in a distribution algorithm:

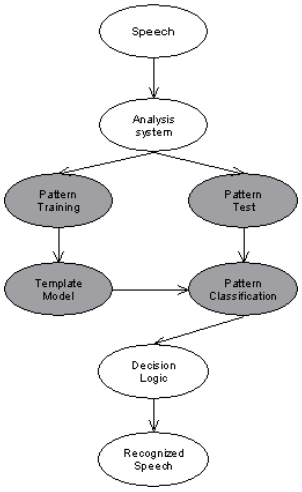
$$MF(m) = \sum_{k=0}^{N-1} H(X_k, m) * Spectral(X_k) \tag{5.22}$$

And then finally get the Cepstrum by taking a Fourier transform like the Discrete Cosines Transform (DCT)[10] to the signal(see Subsection5.2.2)

$$MFCC(l) = \sum_{m=1}^{M} MF(m) * \cos(l * \frac{\Pi}{M}(m - \frac{1}{2})) \tag{5.23}$$

[10]DCT is a Fourier-related transform similar to the discrete Fourier transform (DFT), but using only real numbers.

5.3 Pattern Recognition



Speech recognition techniques rely heavily on different machine learning techniques/Pattern recognition techniques. At the same time this is some of the most challenging problems for machines to handle. Pattern recognition techniques, or the ability to recognize patterns, form the core problems in speech recognition and is also the core of human intelligence[40]. If we could incorporate the human ability to recognize patterns from our daily lives into a machine. The machine would be able to distinguish between different patterns and recognize which is the correct one like a human. This is why ”Pattern Recognition”/”Machine Learning” approaches are sometimes referred to as ”artificial intelligence”.

Minimum error-rate estimation is the ultimate goal of pattern recognition techniques, but there are different approaches to this. Techniques / algorithms built upon class information, called *Supervised Learning*, use Maximum Likelihood Estimation(MLE) and Maximum A Posterior probability(MAP) estimation techniques. Among the most known ”techniques” are the statistical techniques, built on *Bayes Decision Theory*. Techniques / algorithms that do not take class information into account are called *Unsupervised Learning*, and are built upon a more iterative approach. The most known algorithm here is called *k-means* as we will discuss in 5.3.4. However, the *Hidden Markov Model*(A Prediction algorithm predominately used in larger speech recognition systems [40]) also belongs to this category.

The following subsections will go through some of the most important theories / concepts in Pattern Recognition in respect to speech recognition. The most common algorithms used in speech recognition, DWT (Dynamic Time Warping) and K-nearest will be introduced.

5.3.1 Pattern Training

Building the various pattern templates in speech recognition depends first upon ”cleansing” the features in a signal that contains the most information, and then extracting them (see previous Section 5.1 and 5.2). When the features are ”cleansed” of of disturbing interference, the next step is to ”train” a classification model on the patterns. This phase focus more on enhancing the features that contain the most information i.e. if we need to train a prediction model based upon a known signal this signal might have three phonemes, like F1, F2, F3 in Figure 2.2. However one of these phonemes is more important (has more unique information) than the others, we just do not know which. Assigning different probability values based upon the signal‘s phonemes ”information”

would therefore create a stronger prediction model. To train and enhance the phonemes probability distribution effect on the overall classification we can use various algorithms build on Bayes decision theorem.

Bayes Decision theory

Bayes decision theory is based on the assumption that the decision problem can be specified in probabilistic terms and that all of the relevant probability values are known. We might call it a formalization of empirical "common sense" procedures. The *likelihood* that something which does not happen very often under a set of given conditions should all of the sudden start happen often under the same conditions is very low. For example, the likelihood of; snow in the summer, while it is 30 degrees. The *likelihood* is formally written: $p(x|\omega_i)$ for $i = R$ and ω being a discrete variable with, for instance, the following values $\omega = \omega_i(i = (1, 2, 3)$.
Let us exemplify: We have would like to predict if an "answer" from a customer is either: buy, bargain or browse. We already know the probability function $p(\omega)$ given $\omega_i(i = (1, 2, 3)$ or $i = (buy, bargain, browse)$. However let us say that prior information indicates that ω_3 is the most given outcome. We might be inclined to give our customers a bit of a rough time, and probably not make that much money. So although choosing ω_3 might be the most plausible outcome, it is bad for business and therefore it is unreasonable to make a decision based on this information. However if we are given more observable data, such as the pitch of the first phonemes, length of the word, we can even add information that is context related as the general country economy etc., we might then make more informed decisions. So let x be the random value of one of the observable data that can effect our decision, like the rate of change in the spectrum bands in the first 100 ms of the word, then the probability density function is given like $P(x|\omega_i)$ or the probability that x will happen given ω_i. With this knowledge we can compute the conditional probability (also called *posterior* probability as it depends upon the specified value of x) $P(\omega_i|x)$ given Bayes rule[11]:

$$P(\omega_i|x) = \frac{p(x|\omega_i)p(\omega_i)}{p(x)} \tag{5.24}$$

where $p(x) = \sum_{i=3}^{3} p(x|\omega)P(\omega_i)$

Using this knowledge in speech recognition we can help with building the templates as well as predicting/classifing the outcome of what has been said. Knowing a given set of conditions, and under each set of conditions we have a set of words possible to be said. Then we can use the Bayes Theorem to assist us in predicting or just narrowing in on one word which has been said given a

[11] for more information about Bayes Rule see [40] or [9]

certain condition, or given a certain word whose condition we are in.

5.3.2 Comparing Templates

The speech templates are built up upon a combination of digital signal processing and feature extraction / enhancing algorithms, as we have gone through in previous sections. When one or more templates has to be compared with a new sound signal, the new signal has to be processed by the same algorithms as the saved templates, in order to enable a comparisons.

Distance

Measuring the comparison of two templates (signals) is done by calculation the distance between the two signals. However the distance-calculation method depends on the representation of the signal after it has been processed, just as a key factor in distance measuring depends upon the sounds original phonetic relevance[25]. The distance measure should measure the subjective distance more than the real difference. Loosely translated "if it sounds the same it is the same". With this we mean that the perceptually distance between two signals should be related to a distance measure. So if there is a small perceptual difference between two signals then this should be translated into a small real distance. The distance measure should therefore take the pre-processing already done to the sound signal into account before measuring the distance of two signals. This could include affecting the a signals features by weighting, lifting or effecting them via their probability distribution, etc..(see [25] or [41] for a more comprehensive coverage).

Spectral distance

Measuring the difference between two signals/patterns using a spectral distance measure, in terms of either its average or accumulated spectral distortion, is one of the most used techniques, as both the mathematical traceability and psychoacoustic studies of perceived sound studies reveal it to be correct for its subjectivity[25].

The spectral signal is a positive number (see Equation 5.17) which makes the distance $d(S(\omega), S2(\omega))$ into a positive result. The difference between two spectra vectors is given by $V(\omega) = S(\omega) - S2(\omega)$ but the difference is normally calculated by taking log to the signal and then either calculating the mean or the rms (root-mean-square) value, given the following:

$$V(\omega) = \log S(\omega) - \log S2(\omega) \tag{5.25}$$

This then leads to the distance measure at a given vector with p-size:

$$d(S(\omega), S2(\omega))^p = |\log S(\omega) - \log S2(\omega)|^p \tag{5.26}$$

There are two major reasons for this approach. The first reason is that tests have shown that amplitude is subjectively perceived on a logarithmical scale [25]. The second reason is that taking the rms to the log power of the signal gives a good approximation to the *fundamental frequency* or fundamental tone of the signal (see Subsection 5.2.1).

Cepstral distance

When measuring the distances on the Cepstrum, the signal is given as a set of c_n Cepstrum coefficients. The Cepstrum is defined as the Fourier transform of the log of the spectral signal (see Subsection 5.2.1). This means that measuring the Cepstral distance is somewhat easier than the Spectral distance, since the $logS(\omega)$ has already been calculated. The resulting Cepstral coefficient can be both positive and negative numbers; this gives the following distance formula when measuring the Cepstral distance:

$$d(S(\omega), S2(\omega))^2 = \sum_{n=1}^{N} (c_n - c2_n) \tag{5.27}$$

5.3.3 Dynamic Programming

Variance in speech origins (such as the speaker, environment and transmission, things like; dialects, speed, a cold, dry lips [causing pops prior to speaking], environmental things like door slamming, irregular road noise, fans etc.) are all prone to cause disturbance in the signal. The ideal speech recognition condition is therefore always a single speaker dependent system operated in a quiet room under the same conditions. However systems built and operated under these conditions would probably not have the largest usefulness in everyday situations. The necessity for alignment and endpoint detection of pauses/non-voiced sections in a signal is therefore vital for a good recognition.

Endpoint detection

Endpoint detection or (speech) endpoint detection is to separate the acoustic events of a signal into its speech sections and all the acoustic parts which are outside the realm of interest (e.g. background and transmission noise). There are several approaches to endpoint detection but since

they are closely related to environment and hardware it is more a case of practicality. However in all cases the goal is to get a more precise representation of the speech signal. The most used method are a filter option with a threshold value, during a short time representation of a signal like a frame of 5-20 ms where $if(V(s) > threshold) \Rightarrow V(s) = voiced$. Practical speaking this mean that the threshold value is a matter of best case tests, in the operating environment.

$$V(threshold, s) = (threshold < \sum_{i=o}^{frame} s(i))$$

DTW

When two or more speech signals are compared, it is not always enough to detect where the speech is contained in the signals. The possibility to align them over time is even more important. Let us exemplify by comparing two signals (see Figure 5.13 and 5.14). In both cases it is the same word (the number 8) we are looking at, and the samples starts the same place. However the first signal is $425ms$ or 3400 samples long and the other signal is only $325ms$ or 2600 samples long.

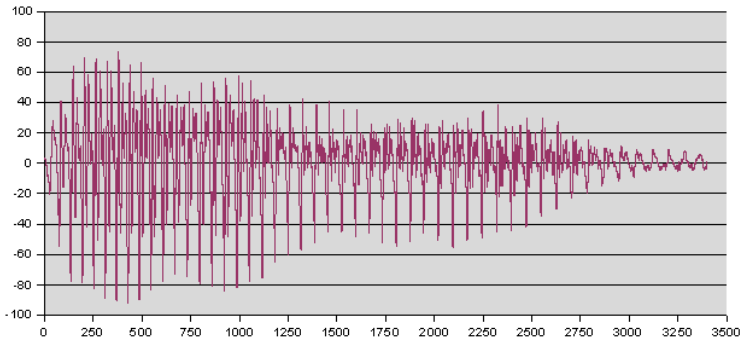


Figure 5.13: Number 8 in 425ms or 3400 samples

As we can see from the figures 5.13 and 5.14, the difference in length between the two signals makes them hard to compare. If we were to measure the distance between the two signals the sum would in both cases, be quite large. This is where the Dynamic Time Warping (DTW) algorithm is used [3]. The goal of the DTW is to ”warp” the signals to fit each other by finding the most ideal path between two signals, the so called ”warp-path”. Let us imagine a somewhat smaller example than 5.13 and 5.14 take two signals given with the following values:

$$\begin{aligned} S1 &= \{0, 1, 2, 3, 4, 5, 6, 7, 2, 1, 0\} \\ S2 &= \{0, 0, 1, 2, 3, 3, 4, 5, 6, 7, 4, 3, 1, 0\} \end{aligned}$$

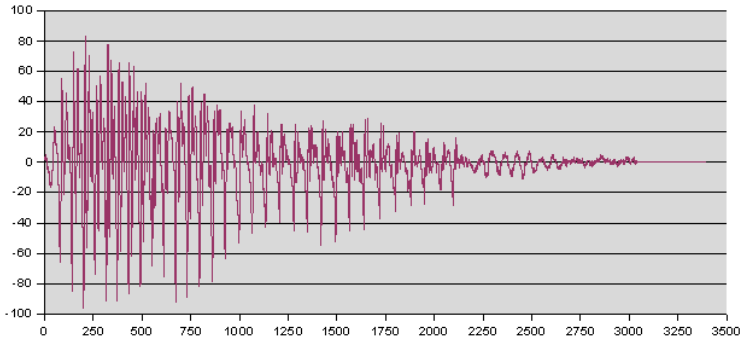


Figure 5.14: Number 8 in 325ms or 2600 samples

The two signals ($S1$ and $S2$) resemblance the signals depicted in 5.13 and 5.14, as signal $S2$ is longer than $S1$. Aligning the two signals using Dynamic Time Warping is done in several steps, to find the optimal warp path.

Step 1: First we measure the distance between the signals. This is done by the methods we looked through in the previous Subsection 5.3.2. Now we are only going to take a simple distance formula into account where we take the absolute value of the distance from one sample to another as the distance $d(S1, S2) = \sum |s1 - s2|$. This has the effect that the distance is the same from 1 to 3 as from 3 to 1. The resulting distance can be seen as a matrix where all the distances from each point are calculated, for $S1$ and $S2$ (the values can be seen in Figure 5.15).

0	0	0	1	2	3	3	4	5	6	7	4	3	1	0
1	1	1	0	1	2	2	3	4	5	6	3	2	0	1
2	2	2	1	0	1	1	2	3	4	5	2	1	1	2
7	7	7	6	5	4	4	3	2	1	0	3	4	6	7
6	6	6	5	4	3	3	2	1	0	1	2	3	5	6
5	5	5	4	3	2	2	1	0	1	2	1	2	4	5
4	4	4	3	2	1	1	0	1	2	3	0	1	3	4
3	3	3	2	1	0	0	1	2	3	4	1	0	2	3
2	2	2	1	0	1	1	2	3	4	5	2	1	1	2
1	1	1	0	1	2	2	3	4	5	6	3	2	0	1
0	0	0	1	2	3	3	4	5	6	7	4	3	1	0
S1 / *S2*	**0**	**0**	**1**	**2**	**3**	**3**	**4**	**5**	**6**	**7**	**4**	**3**	**1**	**0**

Figure 5.15: Distance from S1 to S2

Step 2: Second we need to calculate the cost of the various paths, and find the most optimal path between the two signals. The most optimal path is where the two signals follow a more or less diagonal path through the matrix (5.15) as this would be where they are most alike. This path is identified by following an accumulated path from point$(1, 1)$ and then accumulating the cost of moving from this point in the optimal upper diagonal direction through the matrix, while tracking

which **move** is the ”cheapest”. The *moves* are defined from $(1, 1)$ either by going ”up”, ”to the right” or going ”diagonal up right”. We will take the move which has the lowest cost from starting point$(1, 1)$. From point $(1, 1)$, we then have to evaluate the distance with the best move recursive, following the cheapest route. To indicate which move is the cheapest, we set the value equal to up = 1, diagonal = 2, and right = 3. This results in two new matrixes (see Figures 5.16 and 5.17), given by the formula:

$$accdist(1, y) = accdist(1, y - 1) + d(1, y)move(x, y) = cheapestaccdist(x, y)$$

11	31	31	34	35	37	38	40	43	47	18	9	8	4	3
10	31	32	33	34	35	36	38	41	11	11	5	5	3	4
9	30	32	33	33	34	35	37	6	5	5	2	3	4	6
8	28	28	21	15	10	10	6	3	1	0	3	48	51	55
7	21	21	15	10	6	6	3	1	0	1	44	45	48	52
6	15	15	10	6	3	3	1	0	1	3	42	43	46	49
5	10	10	6	3	1	1	0	1	3	6	41	42	44	47
4	6	6	3	1	0	0	1	3	6	10	41	41	43	43
3	3	3	1	0	1	2	4	7	11	16	40	41	40	41
2	1	1	0	1	3	5	8	12	17	23	38	40	39	40
1	0	0	1	3	6	9	13	18	24	31	35	38	39	39
S1 / *S2*	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**

Figure 5.16: Accumulated Distance from x to y

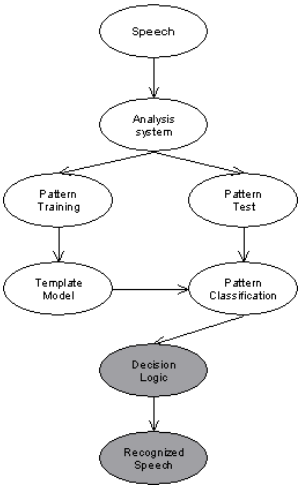
11	1	3	1	2	2	2	2	2	2	2	1	1	1	2
10	1	3	1	1	2	2	2	2	2	2	1	1	2	3
9	1	3	3	3	3	3	3	1	1	1	2	3	3	3
8	1	2	1	1	1	2	1	1	1	2	3	2	2	2
7	1	2	1	1	1	2	1	1	2	3	1	2	2	2
6	1	2	1	1	1	2	1	2	3	3	1	2	2	2
5	1	2	1	1	1	2	2	3	3	3	1	1	2	2
4	1	2	1	1	2	3	3	3	3	3	1	1	3	2
3	1	2	1	2	3	3	3	3	3	3	1	1	1	2
2	1	2	2	3	3	3	3	3	3	3	1	1	1	1
1	0	3	3	3	3	3	3	3	3	3	3	3	3	3
S1 / *S2*	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**

Figure 5.17: Best Moves

Step 3: To conclude the process, we should now identify the best path through the moves. This is done by finding the lowest accumulated distance in the right side of the matrix 5.16 and then moving along the path back to $(1, 1)$ as indicated by the grey color on 5.17.

5.3.4 Speech Classification

Classifying unknown instances of a signal is the final goal of pattern recognition. So far we have presented the key ideas of the speech processing and template modeling, but we still need the ”Decision logic” and Prediction models before we can label the speech segment as ”recognized speech”[12].



In pattern recognition the set of data samples, which is used to calculate the parameters of the prediction model, is often refereed to as the *training set*[18]. In contrast to the *training set*, the set of independent data used to refine/re-evaluate the prediction model(s) is called the *test set*, which is used to prevent overfitting and tuning parameters of the model[18]. Depending on the amount of data available, one might also have an *evaluation set* to assess the expected accuracy of the model when applied to ”real world data”.

There are different approaches to creating a good Classification Model. This could be by combining different classification algorithms, bagging or boosting some of the features in the *training set* when training the Classification Model, or adding a cost to certain predictions (see [9] or [18] for more ways). The combinations of various algorithms are used to make the Classification Model perform better insuring a minimum error-rate estimation. However overfitting and generalization are a problem in speech recognition, so great attention to the *training set* is required. The variation in speaker environment and dialects demands rather large data sets to include all ”situations”[40]. With larger data sets it is therefore hard not to overfit towards certain features, which is why attention to the *training set* is of great importance. Equally important is it to have a *test set* that represents the variation in speaker environment and dialects to verify the Classification Model‘s accuracy.

Building a Classification Model, using the template approach, the two most used algorithms/methods are the ”Nearest Neighbour” classification algorithm [25] and the ”Vector Quantizing” (VQ)[17] clustering algorithm. Since these two methods also utilize many of the previous mentioned algorithms, we will round of this chapter by going through this theory.

[12]It is out of the scope of the master thesis to include all aspects of building ”Prediction models” so please see [9] or [18] for a general approach and [25] or [40] for a more speech recognition orientated approach.

Nearest Neighbour and Clustering

The "Nearest Neighbour" classification algorithm combined with a codebook consisting of several vector quantified templates, built upon some distortion[13] measure (see Subsection 5.3.2, is one of the most used Classification Model[25]. Since the codebook is critical to the ultimate quality of the final Classification Model, using a baysian likelihood distribution algorithm to form the codebook has in some cases shown to be a advantageous approach[40].

The Clustering method can be built upon various algorithms. However the two commonly used algorithms are the *EM-algorithm* and the *K-means clustering algorithm*[40] were both has proved in empirical examples to perform very well.

Nearest Neighbour

Building the Nearest Neighbour Classification Model often means extending the model by averaging over a set of the k-number of nearest neighbours, called the K-nearest model. This prediction method is used as a very good Classification Model, by using a vector quantification distance measure (see Subsection 5.3.2 for distance measures) and then comparing the distance of the vectors to each other.

This K-nearest model can be supplemented by several enhancing algorithms, but a simple approach could be the following:

Step 1: Store the templates, built from the training set as a K-dimensional vector space of frames of LPC encoded cepstral coefficient (see Subsection 5.2.4) $X\{x_1..x_i\}$ each template associated with its class.

Step 2: Build a template of the unknown sound segment from the *test set*, sent to be classified $U\{u_1..u_i\}$.

Step 3: Calculate the frame distance using Equation 5.27 and use these measures as VQ $d(vq_i) = |x_i - u_i|^2$

Step 4: Use End Point detection (see Equation 5.3.3) to clean the distance measures of unnecessary distortion

Step 5: Use Dynamic Time Warping to align all the templates VQ signals to each other $DTW(vq_i, vq_{i-1})$.

Step 6: Use a distance measure (see Equation 5.27) to calculate the distance between the time aligned signals using $DtwMove()$ (see Figure 5.17).

[13]"distortion" is often used in DSP as another word for a distance measure between two signals.

Step 7: Use the distance measures from step 6 to verify which training templates are the closest.

The approach mentioned above is a simple approach; a more complex approach could include an enhancing of the algorithms by iterating through the steps 1 - 6 on the training set to calculate threshold bounders and certainty measures associated with the various classes (words, phonemes or whatever the goal of the prediction is). The threshold and certainty measures are commonly used to exclude unknown test templates from misclassification, i.e. if you have templates covering the letter A and B the test template = C will be labeled unknown (if the threshold values are correct). The simple nearest neighbour template approach (see above) is used especially if you are building a "Speaker-dependent Vocabulary Constrained System" with limited number of templates [40].

K-means Clustering with Vector Quantization

Although there are many clustering algorithms, we will only present the classic clustering algorithm approach called *K-means*[9] [14]. Clustering algorithm works by specifying in advance how many clusters are being used (the K parameter in *K-means*). Then distribute the clusters randomly into different (C_i) cells with each their center/centroid point z_i. The *training set* are then classified into each their respective clusters, depending on the distances from the centroid point. The classic distance measure algorithm is a Euclidean distance metric[9].

In speech recognition, the K-means algorithm is used to design what is normally referred to as a *M-level* codebook, consisting of a partition of d-dimensional space into M-cells C_i[40]. Each M-cell is then associated with a quantized vector (see Figure 5.18 for a 2-dimensional space with 4 VQ metric associated) and the training set is used to optimize the M-level of the VQ, using some overall distortion metric. The overall distortion metric can be defined by[40]:

$$D = \sum_{i=1}^{M} p(x \in C_i) \int_{x \in C_i} d(x, z_i) p(x | x \in C_i) dx \tag{5.28}$$

Where z_i indicates the centroid of the codeword $z_i = cent(C_i)$, and $p(x \in C_i)$ denotes the prior probability of the vector x in cell C_i, remember that we are randomly distributing the first Clusters (codewords), and that we do not know the ideal number of derived VQ codewords. In simple steps the whole K-means algorithm can be explained like this:

Step 1: Derive initial Vector Quantified codewords in a M-vector code-book.

Step 2: Classify each vector x_k into one of the cells by choosing the closest codeword.

[14] A lot of researches are still focusing on finding the optimal clustering algorithm, but there is no conclusive best practice yet[40]

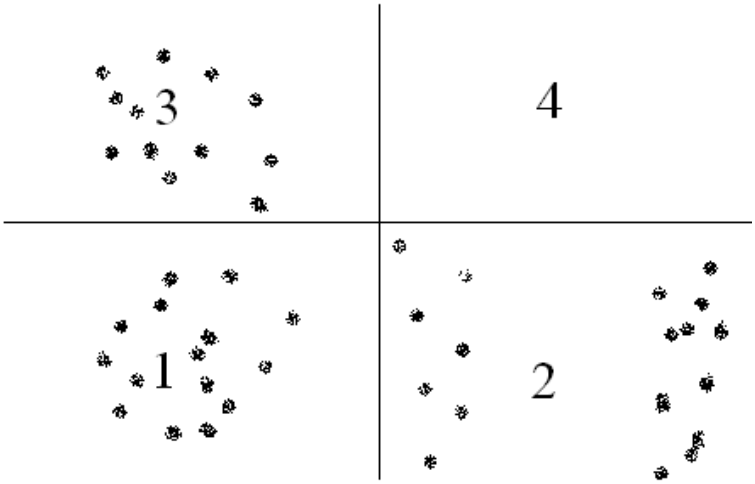


Figure 5.18: 2-dimensional space with 4 VQ metric associated and training points plotted

Step 3: Update the codeword of every cell by computing the centroid of the training vectors.

Step 4: Repeat steps 2 and 3 until the ratio of the new overall distortion/distance D is under a given threshold.

The problem with the Clustering algorithms is that we do not know the ideal number of clusters/codewords and initial distribution of the codewords. If we look at Figure 5.18, we can see that not all codewords have actual training vectors associated with them (codeword 4 has none). However codeword 2 looks like it actually should be split into two separate clusters. This problem is associated with the initial random distribution of the clusters. A cross validation method is therefore incorporated into some of the clustering algorithms to identify an ideal number of codewords[40].

Building a prediction model on the clustering method is essentially the same as the approach in 5.3.4, except that the VQ approach offers an extra compression layer, which means that the comparison is now done in one d-dimensional space and step 3-4 can be omitted. For example, the word ”Me” might consist of two frames with the following values:

$$Me = \{^{1,2,3,4,5,6}_{6,5,4,3,2,1}\}$$

Using the codebook we can now translate the two frames into their representing codewords using their VQ values, so the word ”Me” can now be represented by its associated codewords e.g.:

$$Me = \{4, 12\}$$

This simpler representation of the words or sentences enhances the compression of the original signal, potentially making the comparison between signals much faster. However the extra compression of the original signal also means loss of representation of potential information to distinguish the signal, potentially making the comparison between signals much more prone to errors. Clustering algorithms are commonly used in ”Continuous Speech Recognition” systems, together with a bayesian probability distribution algorithm.

Chapter 6

Summary

In the first part of this master thesis we have looked at the Java language, embedded systems, JOP, signal processing and pattern recognition techniques, and we have discussed the challenges and theory that these present in respect to developing a speech recognition systems. All of the challenges reviewed in the first part of the master thesis define the requirements for our embedded Java speech recognition SDK.

Let us review the requirements, before moving on to the actual design and implementation of the JopSpeech SDK.

Java The SDK should be implemented in Java with a minimum of changes in Java semantics.

Network The SDK's architecture must support a network solution.

Memory The SDK should handle memory as a limited resource in an embedded system and should therefore be implemented as reusable where possible.

Real Time The response time for system developed using the SDK on JOP, should be identifiable within a worst case scenario.

Processing power The power of the CPU is normally restricted in an embedded system to its minimum. The demands should therefore be identifiable by using the SDK on JOP.

Fixpoint As there is no support for fractional arithmetic, the SDK should handle all numeric properties using Integer/Fixpoint arithmetic.

Audio in As there is no hardware support for microphones and loudspeakers for recording and playback on JOP, the SDK should provide a solution for this.

Application structure The software implementation of the SDK should primarily follow the application structure defined on JOP Section 3.3.1. By insuring that memory is allocated, threads are created and all non-time critical methods are initialized in the *Initialization* phase.

Front-end The SDK should provide options for Sampling and Filtering

Analysis The SDK should provide options for: converting between Time and Frequency domains, encoding sound signals and providing filter bank algorithms.

Recognition The SDK should provide endpoint detection algorithms, distance measures, dynamic programming and classification models.

Chapter 7

The Architectural Design

The architectural design of JopSpeech presents a challenge as the design needs to offer the developer a faster approach to speech recognition application development, while at the same time not restrict the researcher in his choice of experiments and the developer in his choice of application.

This Chapter will present the architecture and discuss some of the design issues for developing the architecture. The architecture is designed to be simple, with high cohesion within and low coupling between components making it possible for new developers to have a flexible approach to system development in speech recognition.

7.1 System Architecture

The system architecture is defined with flexibility in mind, as the usages vary from developer to developer. Complexity is reduced by introducing a limited set of components with high cohesion, connected in a horizontal flow, while being able to utilize certain helping tools (in the Utilities component) vertically accessible for all components. The goal has been to create stable and reusable components while not restricting other developers. The components of the architecture is build from the model discussed in Section 2.3 and Figure 2.6 proposed by L. Rabiner and B.H. Juang in 1993 [25]. This approach is chosen because L. Rabiner and B.H. Juang is pioneers on the speech recognition area. The steps in the model Figure 2.6 are in the proposed architecture made into three components:

FrontEnd:

The *FrontEnd* component defines the first and last point of communication with the world outside the system. This could be through a microphone, a loudspeaker, an LED light or an LCD display. No matter the method, the component's responsibility is to ensure correct communication to the outside. *FrontEnd* is structured in an independent component as we are aware of the fact that a lot of final products will implement many of the functions contained in this component into hardware (examples of this could be filters see Section 5.1.2 and sampling Section 5.1.1). However, for researchers where clarity is the goal and for prototypes where processing powers are not issue to be dealt with, the *FrontEnd* component will create a flexible and understandable structure of the system.

Analysis:

The *Analysis* component defines the processing of a given signal from the *FrontEnd*. It is in this component the signal processing is preformed by example Linear Predictive Coding (LPC) see Section 5.2.4 or Filter banking Section 5.2.5. When the signal processing is done the final Template is returned from the component.

Recognizer:

The classification algorithms and the trained models are stored in the *Recognizer* component. This component is responsible for training and predicting results.

Utility:

The component *Utility* defines some utilities for the three other components. This is where utilities to perform fixpoint arithmetic, network communication and normalization will be.

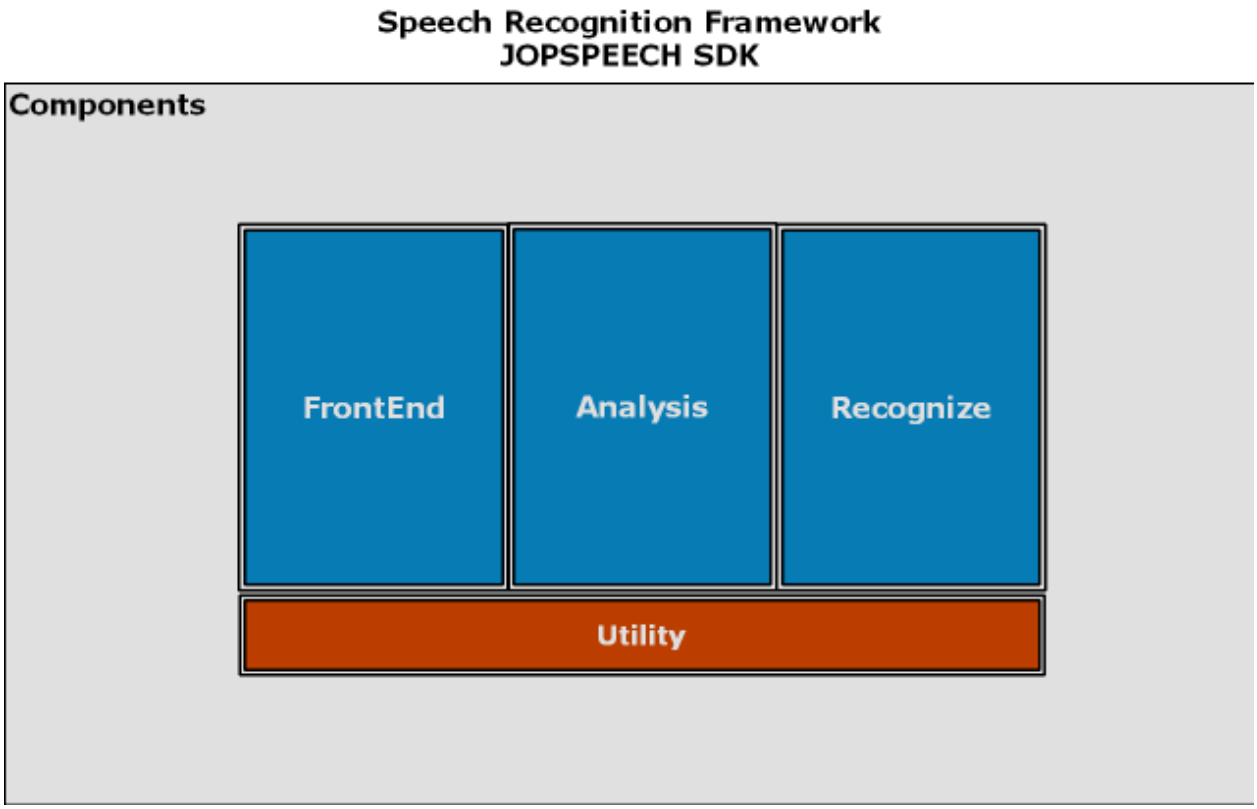


Figure 7.1: Architecture for Speech Recognition on JOP

Each component (except the *Utility* component) has a defined responsibility which new developers can extend upon through defined interfaces for each horizontal component. To ensure flexibility for developers, but also to guide beginners in speech recognition system development, each component handles its own responsibility through a set of abstract classes defining format and flow. We will also aid the developers by implementing a wide range of classes in each component that enable speech recognition (see Chapter 8). However, as some developers might already have a component to handle the various tasks, all they have to do is to extend it with the correct interface and it will work with the rest of the SDK's components. This approach basically consists of decoupling an algorithm from its host and encapsulating the algorithm into a separate class (refereed to as The Strategy Design Pattern[1]).

7.1.1 Interfaces

If any researchers or developers already have some components they want to use instead of the ones supplied by the SDK to change their design, interfaces to each horizontal component are available.

[1] `http://www.javaworld.com/javaworld/jw-04-2002/jw-0426-designpatterns.html`

Audio

The *Audio* interface defines how to get a signal in and out of the system; it could be that a change of microphones or a change of sampling method is necessary. The *Audio* Interface defines the methods required for signal recording and playback.

int record(Signal signalIn) The recording method defines how the system should receive a signal from outside. To ensure that all signals are handled in a certain way, the format input should always be of the type *Signal* (see abstract classes Section 7.1.2). The return of this method is the length of the signal.

void play (Signal signalOut, int length) The playback method defines how the system should play a *Signal* to the outside example a loudspeaker.

Algorithm

The interface *Algorithm* defines how different algorithms should process the signal in the Analysis component they receive and it defines what should be returned.

void process(int type, Signal signal, Template template) The process method is the analysis of a given signal. The signal is transformed into a *Template* format (see Subsection 7.1.2).

Classifier

The interface *Classifier* in the component Recognizer defines how to train a model and test a template. The two methods that need to be implemented by the different *Classifier* are:

void train(Model model) The method is meant to train a given model for the recognizer to use when testing a template.

void test(Template testTemplate, Model model) The test method is the one that classifies a template on a given model and sets the template's `classID`.

7.1.2 Abstract Classes

Abstract classes are excellent candidates inside of the JopSpeech application framework for standardizing communication and structures. For a full overview and the Java implementation see Appendix B.3.

Application Framework

To ensure that the application structure is followed within each component, a class is defined for each component containing a set of methods to handle the communication and work inside the three components (see Section 8.1).

Signal Class

The `Signal` class is the basic structure for a signal that is entering the system in the `FrontEnd`. The `Signal` class pre-defines variables which contain information about a signal and needs to be extended by a class using the singleton pattern. This pattern is used for recommending that only one signal can be processed at a time in the *FrontEnd* component. The signal works in this way as a buffer. This is a design choice, as the system needs to handle memory as a limited resource. There is no pre-defined method in the class. This makes it flexible for the developer to choose if there should be some methods implemented for the signal. The important variables in the `Signal` class are:

`String classID` This variable defines which classification is associated with the signal. Value ”UNKNOWN” indicates that the signal is unknown.

`int[][] frames` This variable contains the signal segmented into frames by, for example, a window function.

`int length` This variable defines the length of the raw signal.

`int numberFrames` This integer defines tha number of frames in the `frames` array.

`int [] raw_sample` This variable is an integer array, set to a certain size in the initialization of the component. The variable is used to contain the raw sample values of a signal.

`int signalID` This variable is a integer identifier for the signal.

Template

The Template class defines a structure that is used to save the analyzed signal's information. For example if the goal of the developer is to save a template with a phoneme, word, or full sentence, then this is extended from the abstract `Template` Class. To identify each template a `classID` is defined. As with the `Signal` there should only be one fixed size instance of this in the `Analysis` component to ensure real-time predictability and memory reusability. The important variables in the `Template` are:

`String classID` The variable defines which classification associated with the signal just as the variable in the `Signal` class.

`int [][] coefficients` The variable consists of a double integer array, set to a certain size in the initialization of the component with frame size as length.

`int numberFrames` The variable consists of an integer, indicating how many integer that are used in the double array.

`int templateID` This variable indicates the identifier for the template by a integer.

Model

The Model class contains information about the model that is used by the application. It also contains information regarding the analyzed templates and information about which threshold values the model can classify unknown templates for. The important variables in the Template class are:

`int NUMTEMPLATES` This variable indicates how many templates that the model contains.

`int readyToTrain` The purpose of this variable is to indicate whether the model is ready to train or not. 0 indicates that the model is not ready for training and 1 indicates that the model is ready for training.

`Template[] templates` The Model consists of an array of known templates.

`int threshold` The threshold for which templates the model can classify

`int trained` This variable indicates if the model is trained or not. 0 means that the model is not trained and 1 indicates that it is.

7.2 Process Distribution

There is no restriction on the process distribution between processors or the three components. Networked Client-Server distributions between PC and JOP can easily be implemented. On Figure 7.2 a distributed solution between PC and JOP is illustrated where the pc does all initial work on the signals that will end up being trained as the Model.

There are some considerations for the developer in implementing a system that distributes its processes between processors regarding deadlocks and data transaction. However, as this is not any

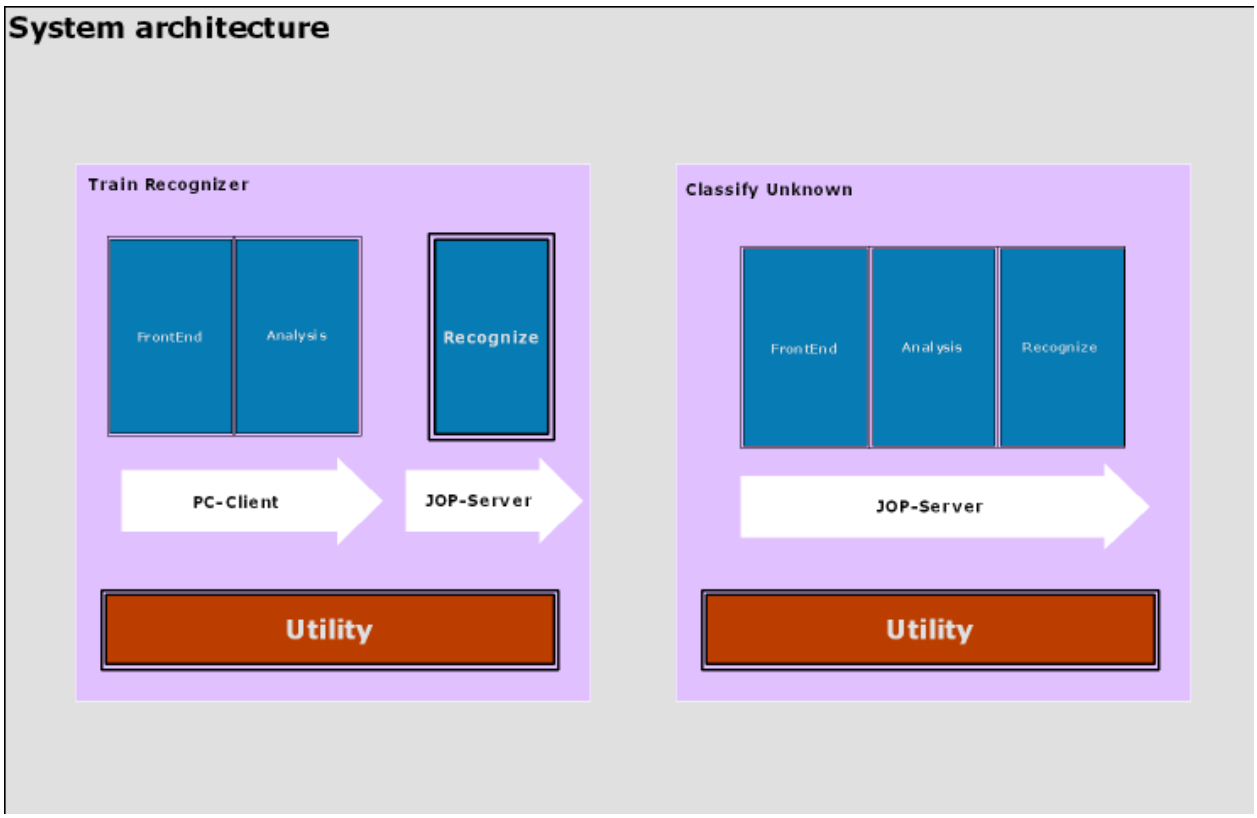


Figure 7.2: System architecture for distribution of components

different from any other distributed solution implementation, and as the power of the CPU is normally restricted in embedded systems (see Chapter 3) as we will leave all check implementations up to the users of the SDK.

7.3 Data Flow

The JopSpeech architecture is defined with a natural flow, horizontally from left to right and back again (see Figures 7.3 and 7.4). This flow needs to be implemented *strictly* through the interfaces defining the communication between the horizontally placed components. The reason for enforcing this strict sequence flow between the horizontally placed components is that it must be possible for new developers to get a more flexible approach to speech recognition system development. We have deemed the simplistic approach to building speech recognition systems a more important requirement than making a less strict data flow between components.

The sequence flow between the components is illustrated by the training sequence flow in Figure 7.3 and the classification sequence flow in Figure 7.4.

The application structure defined on JOP Section 3.3.1 tries to ensure that memory is allocated.

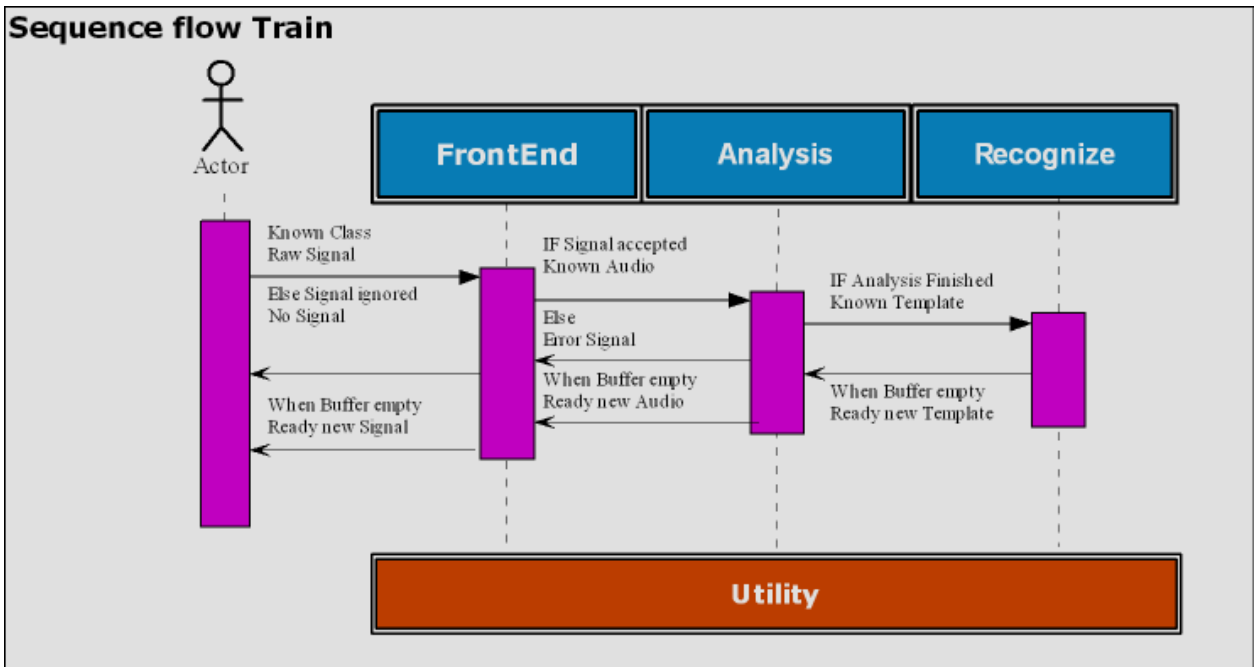


Figure 7.3: The flow for training of speech recognition model

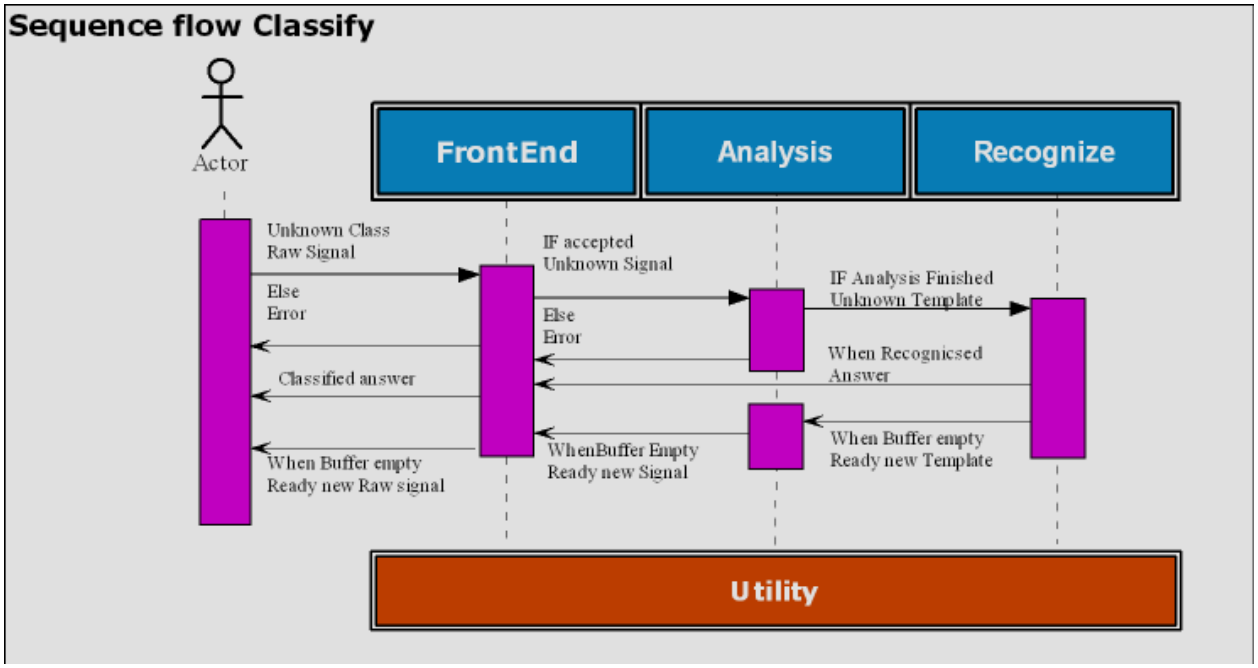


Figure 7.4: The sequence flow for classifying

Threads are created and all non-time critical methods are initialized in the Initialization phase. This is done by implementing an init methods in each class (see Chapter 8). However, it is still up to each developer using the JopSpeech SDK to ensure that no objects are initiated in the runtime of the

application. It is also up to the developer to decide whether the work has to be done periodically or a asynchronously. For this reason two different implementations of threads are implemented on JOP RtThread[2] and SwEvent[3]. The RtThread is for periodic work and the SwEvent is for asynchronous work. (For more information about the use of these we refer to [29]). Chapter 10 will show what an implemented prototype could look like within the framework and by following the design patterns.

7.4 Component Flexibility

The architectural design is supported by flexibility and therefore not made any more restrictive than already discussed in Section 7.3. The possibility to combine different classes within each component and to perform test towards a given context is one of the major achievements with this SDK. It is possible for developers to choose the workload of the different components and even within the different components distribute the workload over several processors.

There are many different implementation options in the JopSpeech SDK (see Figure 7.5 for some of them) which should aide developers and researchers who might have different ”agendas” towards speech recognition in embedded systems. Each implementation pattern and Digital Signal Processing (DSP) algorithm setting in the model has an effect on the prediction quality and speed of the developed systems.

By making the components highly cohesive with low dependence on each other, it is possible to change the components or just implement new methods if other DSP algorithms need researching. This is a needed feature of the SDKs as the various techniques not only change the choice of DSP algorithms', but also the choice of the prediction classifiers and the goal of the implementation. Among the various types of speech recognition systems are:

Speaker independent: Speaker-independent systems are recognizers that can be used by anybody with no training necessary. These systems are usually deployed in environments where the system cannot be trained, and will probably be built in a distributed architecture where the model will be trained locally on a PC and then deployed to the embedded system.

Speaker-dependent: Speaker-dependent systems are those systems that first need to be trained for use with a specific speaker. Voice samples of each speaker are taken, analyzed and stored. The live speech is then matched with these samples to accurately determine the words

[2]Real-Time Thread
[3]Software Event implemented as a thread

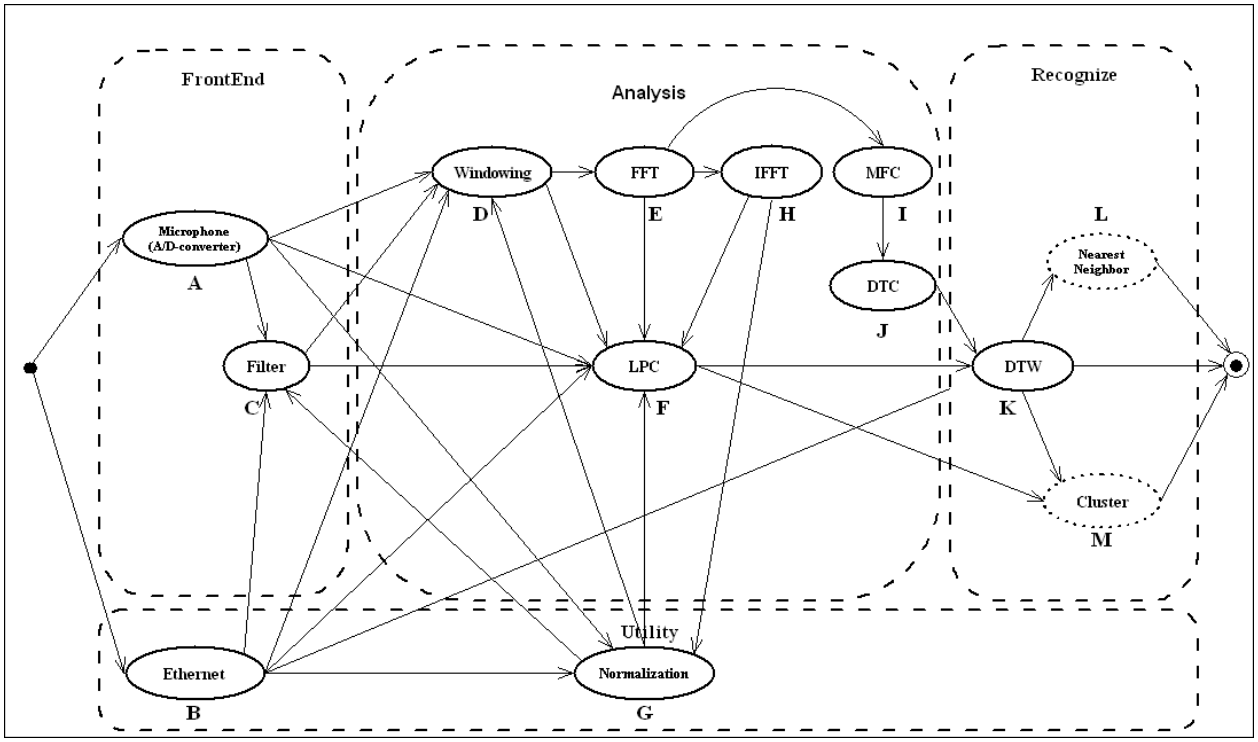


Figure 7.5: Some of the different possible implementations within JopSpeech SDK.

spoken. These systems might also be stand-alone systems, only running on the embedded system (Mobile phones are examples of this design).

Continuous Speech Recognition: Continuous speech recognizers allow users to speak naturally and continuously. This kind of application requires a lot of processing power and will often be seen in a distributed environment, sharing processing loads.

Isolated or Discrete Speech Recognition: Isolated speech recognizers require the speaker to pause (about a fifth of a second) between each word that is uttered, so that the recognizer can buffer the word before processing the next one.

Vocabulary Constrained Systems: These systems have a limited vocabulary that they can understand. A small vocabulary means that users have to restrict their speech to words that the recognizers understand. These systems can be implemented with less signal encoding algorithms and less training, as the vocabulary is restricted.

Chapter 8

Implementation - Software

In this Chapter the software implementation of JopSpeech, and the concerns regarding the class structure's adherence to the application structure suggested in Chapter 6 are discussed.

We will not present all the classes. These can be found in the API in the Appendix B.3 or on `www.jopspeech.com`. However some of the different speech recognition techniques discussed in Chapter 5 and the components in Chapter 7 together with a WCET analysis will be presented.

8.0.1 Note on WCET Analysis

The Worst Case Execution Time (WCET) analysis is done by analyzing the worst-case path in the programs/algorithms and then summarizing the execution cycle of each bytecode in the path[1]. The analysis shows what influence it has to change some of the most essential variables in a speech recognition system. Which algorithms are the most computational hard in the application? This information can give developers the opportunity to identify optimum settings for their speech recognition system and set the worst case time periods exactly for the scheduling of threads performing the actual work in their application.

The WCET tool[2] is a part of the development kit for JOP and is still under development.[3] Therefore some of the execution times are not yet implemented. To get exact measures with the tool, we have measured the execution of two bytecodes that are not yet implemented and then extended them in the tool. The calculations and output from the WCET tool can be seen in Appendix B.2, referring to each method in the subsections under some of the implemented classes.

idiv This is the division of integers. The execution time of this method is calculated by the WCET

[1] A list of execution cycles on JOP can be found in the appendix of the Ph.d. thesis by Martin Schoeberl [29].
[2] Developed by Rasmus Pedersen and Martin Schoeberl [1].
[3] This is true at the time of writing (November 2006), but might have changed by now.

tool to be 2348 plus 75, for invoking a static method. This adds up to 2423.

irem This is the logical integer remainder for the modulus operator. The execution time of this method is calculated by the WCET tool to be 1865 plus 75, for invoking a static method. This adds up to 1940.

8.1 Application Structure

The overall implementation issue for the SDK components is how to insure that the requirements discussed in Chapter 6 for embedded systems will be ideally implemented. For all of the classes implemented we have chosen to follow the application structure proposed by Martin Schoberl [29]. This means that all classes have implemented a initialization and a mission phase. The initialization phase uses a singleton design pattern to restrict the instantiation of a class to only one object[10]. The reasons for the use of this design pattern is the embedded concerns limited memory space and non time-critical initializations. The reason for the pattern is also to help prevent the system from break down in runtime because of extra objects that potential are created in runtime.

The application structure is implemented and constructed so that users of the JopSpeech SDK can extend new classes from each component following the definitions of the abstract class defined in each component. Each definition is implemented more or less equally in each component (see Appendix B.3).

This next part shows the implementation of the *Analysis* component in the different listings of code (8.1, 8.2, 8.3, and 8.4):
All variables, threads and non time-critical methods that each Components needs should be defined under the `Init` method. Notice the singleton design pattern as the `Init` method returns the `Analysis` object.

```
23    /**
24     * The Init methods insures that memory is allocated, threads are created
25     * and all non-time critical methods are initialized
26     *
27     * @return The Analysis object after initilizatiuon
28     */
29    public static Analysis init() {
30        return null;
31    }
```

Listing 8.1: Implementation of the abstract class Analysis method init

The Work that needs to be done by the Component needs to be written under the work method.

```
33    /**
34     * The ectending class defines the work performed in the Component.
35     */
36    protected abstract void work();
```

Listing 8.2: Implementation of the abstract class Analysis method Work

As an example this is the way we have defined the work done by the Analysis component in one of the experiments that we are performing (see Chapter 10). Also notice from this example that each Algorithm is implemented using the interfaces described in Section 7.1.1.

```
50      protected void work() {
51          // Window Algorithm
52          window.process(0, FrontEndWorker.utterance, template);
53
54          // FFT Algorithm
55          fft.process(1, FrontEndWorker.utterance, template);
56
57          // IFFT Algorithm
58          fft.process(-1, FrontEndWorker.utterance, template);
59
60          // LPC Algorithm
61          lpc.process(0, FrontEndWorker.utterance, template);
62
63          // If the template is not unknown add it to the Model
64          if (!FrontEndWorker.utterance.classID.regionMatches(0, "UNKNOWN", 0, 7)) {
65              // Adding Template
66              ModelImpl.addTemplate(FrontEndWorker.utterance, template,
67                      NUMBERCOEFFICIENTS);
68          }
69          // Else start the recognizer and it will have defined
70          // in its work method what to do with the template
71          else {
72              JopSpeech.recognizer.setReady();
73          }
74      }
```

Listing 8.3: Implementation of the AnalysisWorker method Work

The actual "mission" of the component does not start before the mission method is called, which should be after the `Init` method is called and all work is declared.

```
38      /**
39       * The mission method defines how the work of the extending class should be
40       * performed.
41       */
42      public void mission() {
43          if (ready == 1) {
44              ready = 0;
45              work();
46              done = 1;
47          }
48      }
```

Listing 8.4: Implementation of the abstract class Analysis method mission.

8.1.1 The Class Diagram

The class diagram Figure 8.1 shows how each class is implemented within the framework (larger image is in Appendix B.4). Within each component there is (as mentioned in Section 7.4) several classes containing algorithms, classifiers and etc., leading to different approaches to choose from depending on the goal of the system. The next sections will describe some of the implemented classes in the components[4].

[4]For the complete API of the classes, see Appendix B.3.

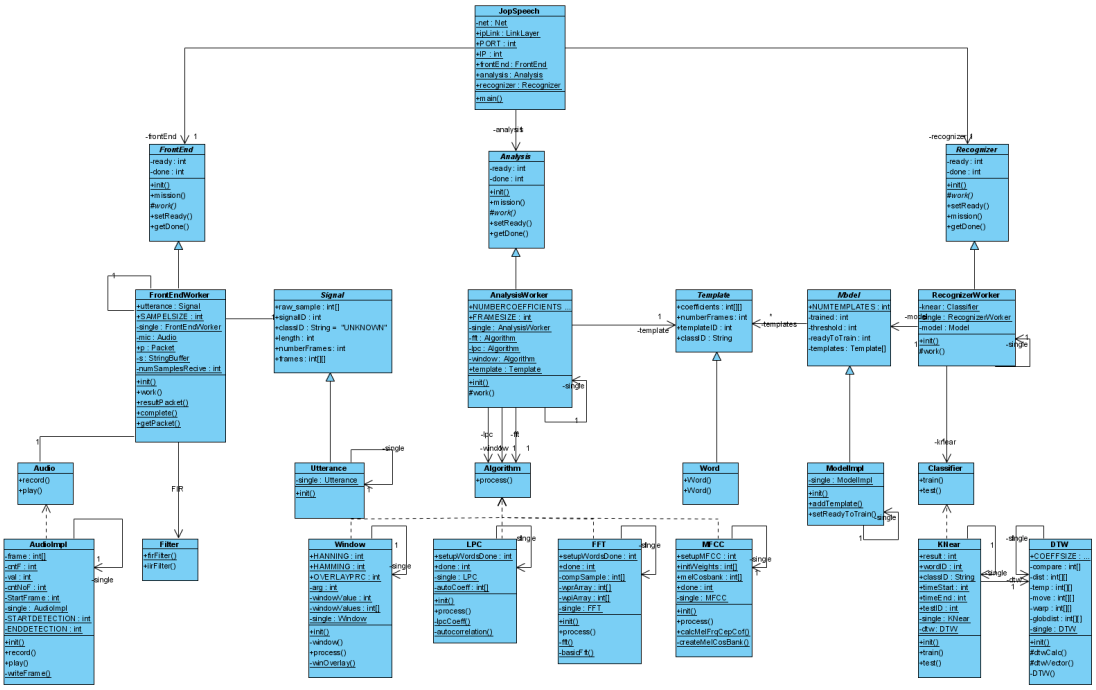


Figure 8.1: The Class Diagram

8.2 Utility Component

All classes in the *Utility* component should be used via static or singleton reference, as they mainly contain methods used by the FrontEnd, Analysis and Recognizer components (see Chapter 7) not already contained in the JOP framework. The fix-point class is an essential part of the *Utility* component of the framework. Some of the issues and methods of `FixPoint` is discussed in the next Section.

8.2.1 FixPoint

The JOP platform does not support complex data types such as float or double, but it does support a full 32 bit Integer structure (see Section 3.3). One of the goals of a speech recognition system is to make as precise a representation of the ”sound units” as possible. To do this the systems need to utilize the 32 bits fully. This is done by using fix-point arithmetic and shifting for calculations on JOP.

Optimizing the precision of the values that we are working with enables us to ensure precision and a high degree of correct classification. Because the split of the 32 bit structure has a great influence on the precision of the classification, a flexible structure of the class `FixPoint` is implemented, with small limitations. A design decision with the class is that the split between

decimals and integral numbers has a limited max of 16:16 bit split. Table 8.2.1 shows an example of a 16:16 bit split, where ”I” is the integer and ”D” is the decimal.

32..	19	18	17	16	15	14	13	..1
I..	I	I	I	D	D	D	D	..D

Table 8.1: Fix-point split 16:16

The reason for making the shift split flexible is that many calculations in the different algorithms discussed in Chapter 5 result in different size values - from very large to very small and decimal demanding values. This means the precision of the values will inevitable change with a different bit shift split and is such of interest to developers for testing towards an ”optimal shift” solution. The goal of the implementation of the class `FixPoint` has therefore been to enable a large variety of bit shifting.

MaxShift Method

Finding the ”optimal shift” value at some of the stages of the analysis of the signals requires attention to the calculations, so they will not result in overflow or underflow. This have to be done manually as Java does not throw any runtime exceptions [5].

Overflow An overflow results when a calculated value is larger than the number of bytes allowed for its type.

Underflow An underflow results when a calculated value is smaller than the number of bytes assigned to its type.

As a result of the various concerns about ”optimal shift” solution combined with keeping the mathematical complexity to a minimum, the fixpoint implementation should enable the user to find an optimal standard ”shift” split. The implementation should ensures maximum precision by identifying the places that cannot perform the calculations without causing overflow and then change the ”shift” value there. One of the methods to help users of the SDK to avoid overflow problems is the maxShift method, supplied with the `FixPoint` class. As can be seen from the code 8.5, our implementation is rather simple, but very effective. At all identified places where there is a risk of overflow the ”maxShift” method can be as shown in Listing 8.5.

[5] `http://www.janeg.ca/scjp/oper/overflow.html`

```
189     /**
190      * Calculates the max shift value for multiplying the number with it self
191      *
192      * @param f1
193      *            Fixpoint value
194      * @param SHIFT
195      *            The number of shifts the f1 value has been shifted
196      * @return The number of times a value needs to be shifted backwards to
197      *         enable shift with a equal value
198      */
199     public static int maxShift(int f1, int SHIFT) {
200         int shift = 0;
201         f1 = abs(f1);
202 
203         while (f1 > 1 << (31 - SHIFT)) {
204             shift++;
205             f1 >>= 1;
206         }
207         return shift;
208     }
```

Listing 8.5: Maximum shift calculation

The ”maxShift” method works by calling it with the current ”shift” value (lets say 16 bit), and with the integers that have to be multiplied. In this case it is the squared value. The number is checked to see if it is possible to square the value without causing overflow - if not, then the number is shifted one back and then this number goes through the check. When a feasible value of the number is identified, then the number shifts are returned, so the new shift value can be used for further calculations.

Sin and Cos Method

Since all sound is made out of sinusoids, some of the most important method are the Sinus and Cosines methods. The implementation of ”Sin” and ”Cos” methods is based upon a mixture of tables, lookup function, and first order (Linear) interpolation.[6] This makes the implementation of the methods slow by using many clock cycles (sin uses 5663 cycles and cos uses 5697 cycles)[7]. However following the application structure (see Section 8.1), the computational cost in creating the sin and cos table can be reduced/minimized during the time critical methods in the applications ”mission” by placing the initialization in the init phase of the application. When the actual values have been calculated, it is ”just” a matter of looking in a table for the values representing the values on the Unit circle (see Appendix for details B.3).

[6] http://www.dattalo.com/technical/theory/sinewave.html

[7] See the Worst Case Execution Time in Appendix B.2 for the sin and cos method.

8.3 FrontEnd Component

The *FrontEnd* component is the entering point for any kind of signal that the SDK needs to handle. The implemented classes (see Figure 8.2) consist of a `Filter` class and an audio implementation (*audioImpl*) that controls the microphone and loudspeaker attached to the BaseIO board [8] returning the *utterance* (a specification of the `Signal` class). All work which the component is responsible for should be implemented and specified in a `FrontEndWorker` class that extends the `FrontEnd` class. In the *FrontEnd* component an example of the `FrontEndWorker` is implemented, showing how a model can be received via distribution from a central PC and signals can be received via a microphone attached to the BaseIO board (see Chapter 9 for details on the hardware).

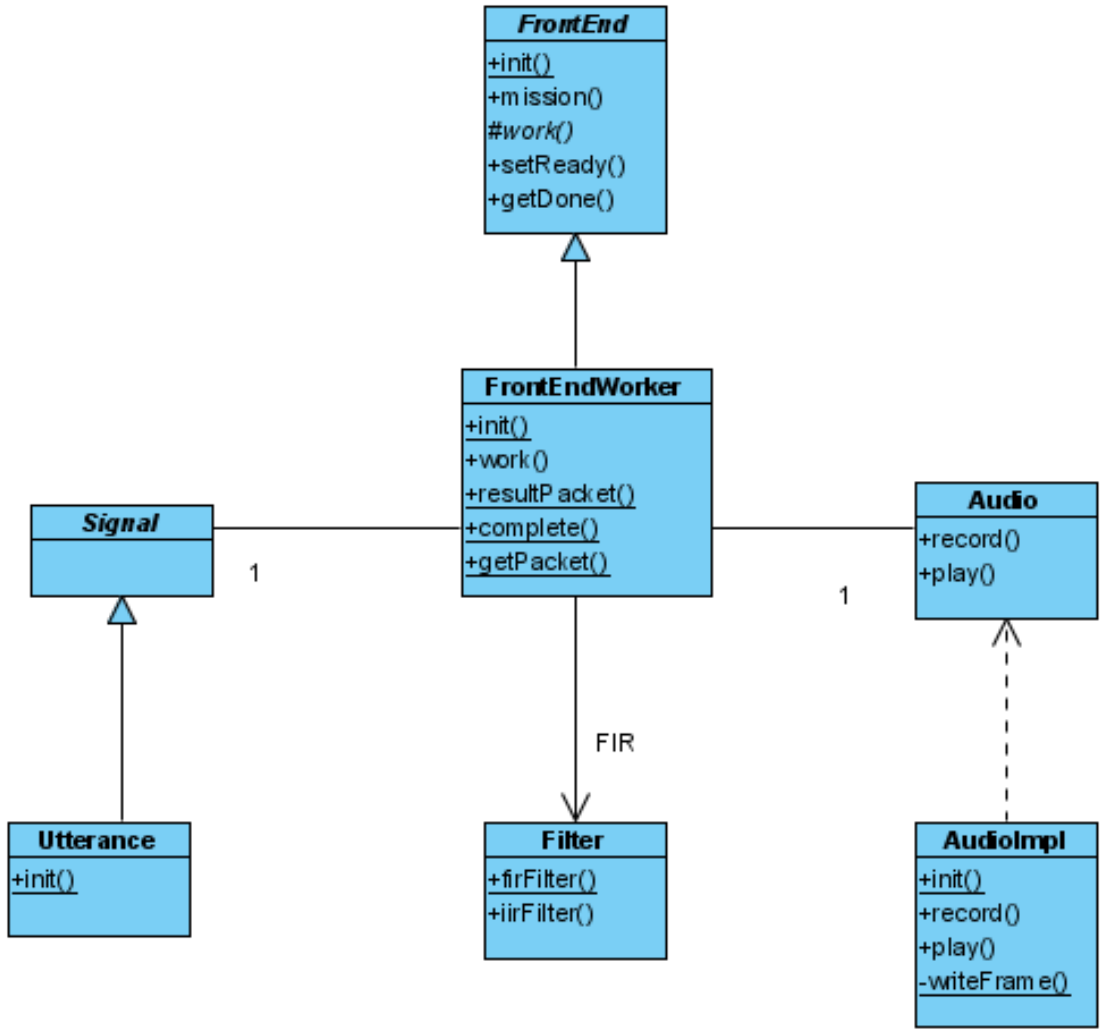


Figure 8.2: FrontEnd Class Diagram

[8] See Chapter 9 on information of the actual hardware implementation.

8.3.1 AudioImpl

One of the main parts of a speech recognition system is to record speech from a given speaker. The class `AudioImpl` is an implementation of the interface `Audio` specifying how to record and play back sound on JOP. The class `AudioImpl` is closely connected to the hardware on JOP and the specially constructed hardware (see Chapter 9). It uses native calls to record and play sound. The two methods for doing this are described in the following. The native calls are implemented as an A/D converter and a D/A converter implemented in VHDL and hardware (see Chapter 9). To record and play sound different variables need to be set in the VHDL code to define the format of the recorded sound. The variables define the sample rate of the sound and the frequency. Those variables set in VHDL are discussed in Section 9.1.1.

Method Record(Signal)

This method makes native calls for receiving a signal from the microphone. The method reads from the A/D converter and then returns a signal when the method has recorded sound and then detected silence again. However let us just go through some of the details of the method.[9]

This is a simple start-point and end-point detection. The threshold value ”STARTDETECTION” for a frame, if its signal's amplitude summed up is not above the start detection value the frame, will be considered empty. The threshold value is dependent on the environment and context in which it is used if no ALC is added to the microphone. The endpoint detection is set to count the number of empty frames after the recording of the signal has started and then stops after a period of silence (see Code in Listing 8.6). The endpoint variable needs adjustment towards the goal of the recording; this is set towards the recording of single words.

```
43      /**
44       * The threshold value for a frame if there signals amplitude summed up is
45       * not above the start detection value the frame will be considered empty.
46       * The threshold value is dependent on the environment and context it is
47       * used, if no ALC is added to the microphone
48       */
49      private static final int STARTDETECTION = 600;
50
51      /**
52       * The end dectection value for a signal if there is more than ENDDETECTION
53       * empty frames being recorded the signal will be considered recorded
54       */
55      private static final int ENDDETECTION = 12;
```

Listing 8.6: The threshold value STARTDETECTION and ENDDETECTION.

One of the issues with recording sound from the hardware is that the offset that defines the zero-point of the signal changes from time to time. This issue is also solved in the method by calculating an offset of the zero-point for the signal each time a new recording is made.

[9] See the Appendix B.3 or `www.jopspeech.com` for the whole class or API.

```
175     /**
176      * *Used to calculate the Offset
177      *
178      * @param lengthOffset
179      *            The number of samples used to calulate the intial center value
180      *            Offset
181      * @return Offset
182      */
183     private int calcOffset(int lengthOffset) {
184         int offSet = 0;
185         if (lengthOffset <= 0)
186             return 1450;
187         for (int i = 0; i < lengthOffset; i++) {
188             val = Native.rdMem(Const.IO_MICRO);
189             if (val > 0) {
190                 lengthOffset--;
191                 continue;
192             }
193             val &= 0xffff;
194             offSet += val;
195         }
196         offSet = offSet / lengthOffset;
197         if (offSet > 1500 || offSet < 1400)
198             return 1450;
199         return offSet;
200     }
```

Listing 8.7: The offset value Method.

8.4 Analysis Component

The *Analysis* component defines the processing of a given signal from the FrontEnd. Each class implemented in the *Analysis* component is either an algorithm used to signal processing or a Template for storing the processed signal (See Figure 8.3). All work which the component is responsible for should be implemented and specified in an `AnalysisWorker` class that extends the Analysis Class. In the *Analysis* component an example of the `AnalysisWorker` is implemented and it show how different signal pre-processing algorithms are used and how their results are added to the prediction Model.

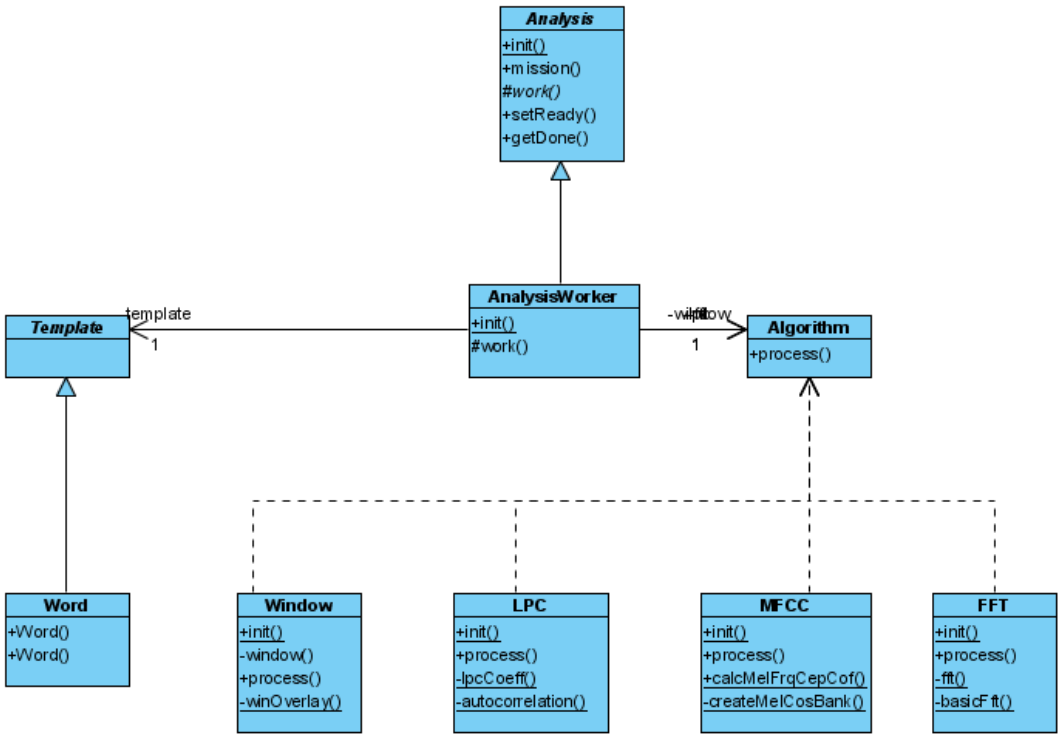


Figure 8.3: Analysis Class Diagram

8.4.1 Window

As described in Section 5.2 in Chapter 5 the algorithm FFT needs a continuous signal to transform the signal from the time domain to the frequency domain. To make the signal continue a window function is used. This class `Window` is a helping class for the analysis part of the architectural design. The class has methods for making the window and window overlay described in Section 5.2.3. The two different window types that are implemented are Hamming and Hanning (see Section 5.2 for theory). One of the issues with these methods of calculating the window for a given

frame is that the cosine functions which is used in this function is relatively slow (see Section 8.2.1). Therefore the implementation is done following the application structure and the cosine calculation is done in the init phase and then saved in the array ”windowValues” (see Listing 8.8).

```
40      /**
41       * The init method of the window functions, must be declared with the window
42       * type before this can be used
43       *
44       * @param type
45       *            Windowtype = 1 for Hanning, 2 for Hamming
46       * @param windowSize
47       *            size of window must be 2 exp.
48       * @param SHIFT
49       * @return Singelton object
50       */
51      public static Algorithm init(int type, int windowSize, int SHIFT) {
52          if (single != null)
53              return single;
54          int arg;
55          int k;
56          windowValues = new int[windowSize];
57          arg = FixPoint.div((FixPoint.PI >> (28 - SHIFT)),
58                  (windowSize - 1) << SHIFT, SHIFT);
59          switch (type) {
60          case HANNING:
61              for (k = 0; k < windowSize; k++) {
62                  windowValues[k] = (1 << (SHIFT - 1));
63                  windowValues[k] -= (FixPoint.mul((1 << (SHIFT - 1)), (FixPoint
64                          .cos((FixPoint.mul(arg, (k << SHIFT), SHIFT)), SHIFT)),
65                          SHIFT));
66              }
67              break;
68          // 579820584 =0.54 shift 30 493921239=0.46 shift 30
69          case HAMMING:
70              for (k = 0; k < windowSize; k++) {
71                  windowValues[k] = (579820584 >> (30 - SHIFT));
72                  windowValues[k] -= (FixPoint.mul((493921239 >> (30 - SHIFT)),
73                          (FixPoint.cos((FixPoint.mul(arg, (k << SHIFT), SHIFT)),
74                                  SHIFT)), SHIFT));
75              }
76              break;
77          }
78          single = new Window();
79          return single;
80      }
```

Listing 8.8: Hamming or Hanning window precalculation

This array is then used by multiplying the values with a given frame, which optimizes the process of multiplying a window on a frame.

Table 8.2 shows how the execution time is when calculating the window function each time for a frame, compared with the technique presented above using an array with pre-calculated window values. This example of using the init method to pre-calculate values shows why the application structure (see Section 8.1) is efficient. The pre-calculation is only called once each time the application is started. The effect is more clear in the two Figures 8.4 and 8.5. Figure 8.4 shows the clock cycles used for each method with a different frame size, and Figure 8.5[10] shows what

[10]The figure is only estimated by multiplying the given time for making one frame with the number of frames, and for the winOverlay method, it is given that the init method has been called.

	Frame Size			
Method	128	256	512	1024
Window values on the frame direct: window()	888131	1773123	3543107	7083075
Pre-calculate the Array: init()	850762	1698378	3393610	6784074
Using pre-calculated Array: winOverlay()	102449	197425	387377	767281

Table 8.2: Worst Case Execution Time for a Hamming Window

happens when more than one frame is used in the system.

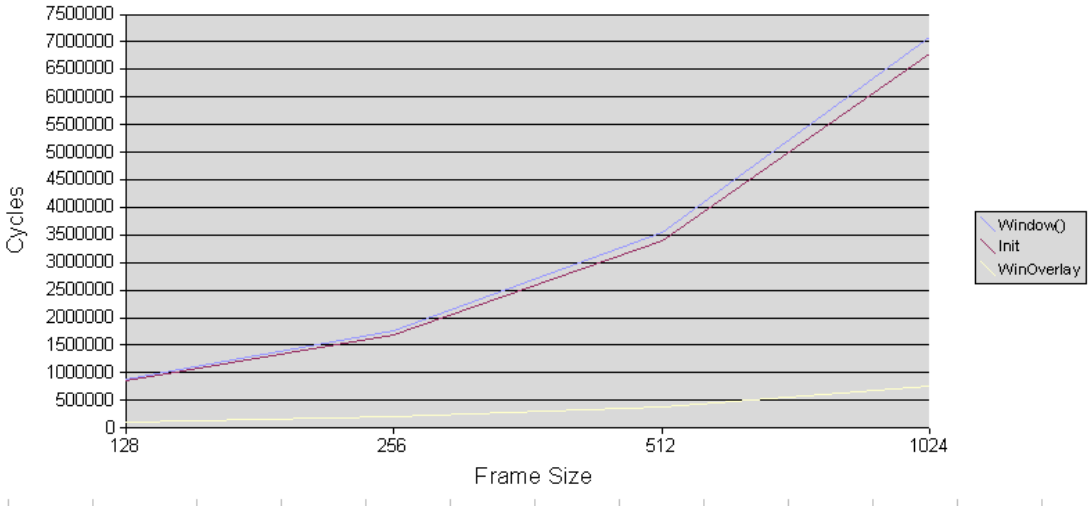


Figure 8.4: Clock Cycles for Window Method Depended on Frame Size

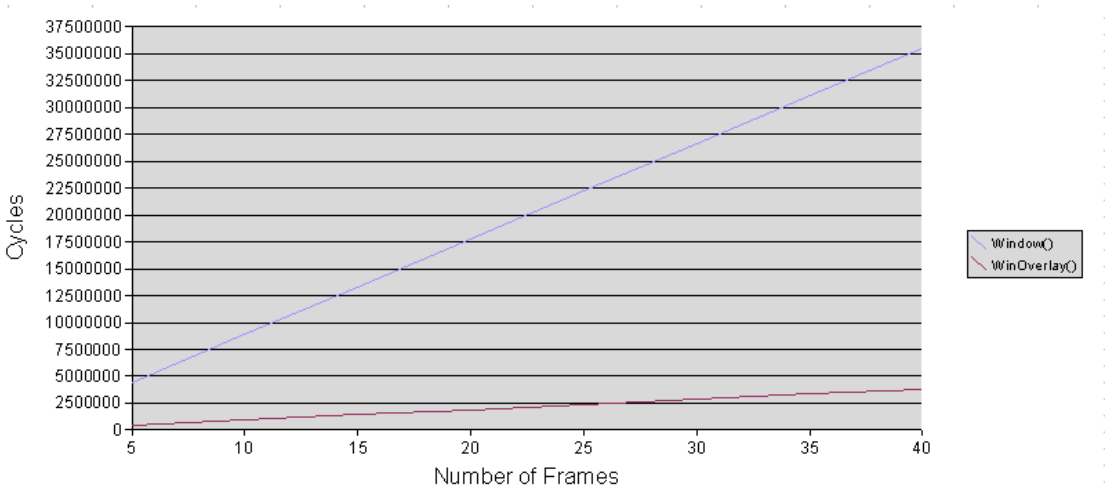


Figure 8.5: Clock Cycles for Window Method Depended on Number of Frames

WCET Analysis

The demonstrated improvement that was made for making the window shows that the execution time is linear with the size of the frame. By looking at the implemented method for the Window method and at how the WCET is depending on the frame size (see Figure 8.6), it is possible to set up a linear formula for the given method because the graph shows that the frame size and the clock cycles used are linear:

$$y = 742x + 92 \tag{8.1}$$

Where x equals the Frame size and y equals the Clock cycles. The 92 cycles that are added are the cycles that do not depend on the loops inside the method.

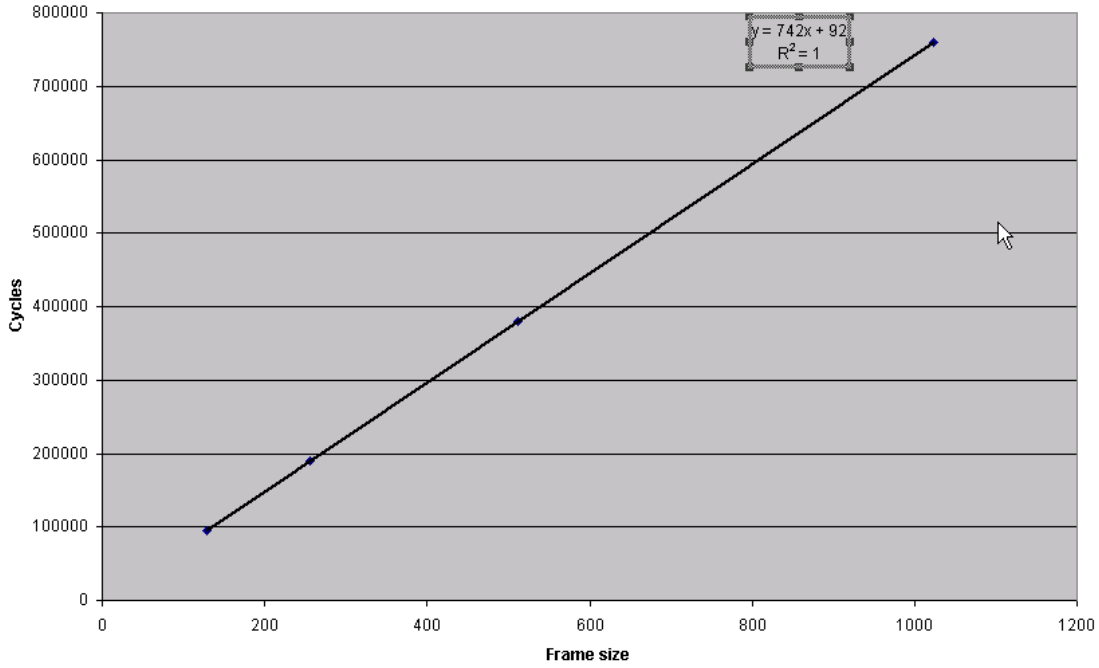


Figure 8.6: Clock Cycles for Window Method

8.4.2 FFT and IFFT

The FFT algorithm or *Fast Fourier Transformation* is used to take the signal from the time domain into the frequency domain (see Section 5.2) and there are many different ways of implementing the algorithm. Our implementation of the ”FFT” class in JopSpeech is a modification of the C function *four1* from ”Numerical Recipes in C”[24]. There are two reasons for modifying the code:

- It has to fit our application structure.

- It is computational very hard, which we seek to optimize.

Optimizing the code has been a trad-off between using extra memory (96 bytes) or using extra time to perform calculations. Given the algorithms 5.11 and 5.12 presented in Section 5.2, instead of calculating the cosine values of the normalized frequency values on the unite circle every time these are needed, the frequency values are pre-calculated and stored in an array, making the FFT faster. This also follows the application structure with an initialization phase. The value for the variables wpr and wpi - which is the wpr for the real part and the wpi for the imaginary part of the complex numbers the FFT returns, it can be seen in the code 8.9.

```
30      /**
31       *
32       * The init method of the FFT Method, must be initialized before the FFT can
33       * be used
34       *
35       * @return FFT Algorithm object
36       *
37       */
38      public static FFT init() {
39          if (single != null)
40              return single;
41
42          compSample = new int[AnalysisWorker.FRAMESIZE * 2];
43          // frequency values for the REAL part on the UNIT circle
44          wprArray = new int[] { -131072, -65536, -19195, -4989, -1259, -316,
45                  -79, -20, -5, -1, 0 };
46          // frequency values for the Imaginary part on the UNIT circle
47          wpiArray = new int[] { 0, 65536, 46341, 25080, 12785, 6424, 3216, 1608,
48                  804, 402, 201 };
49          single = new FFT();
50          return single;
51      }
```

Listing 8.9: FFT initialization of WPR and WPI

Looking at how much the FFT algorithm is improved by storing values, instead of calculating them every time the algorithm is used, can be done by comparing the two techniques by summing up the cycles both methods take in bytecode execution time. If we calculated them instead of saving the wpr and wpi values in an array, every calculation would use *13726 cycles*. The calculations then needs to be multiplied with the times the FFT runs the ”while loop” shown in listing 8.10.

```
206         while (n > mmax) {
207             istep = mmax << 1;
208 
209             wpr = wprArray[cnt] >> (16 - SHIFT);
210 
211             wpi = (wpiArray[cnt] >> (16 - SHIFT)) * isign;
212 
213             wr = 1 << SHIFT;
214             wi = 0;
215             for (m = 1; m < mmax; m += 2) {
216                 for (i = m; i <= n; i += istep) {
217                     ii = i - 1;
218                     jj = ii + mmax;
219 
220                     tempr = FixPoint.mul(wr, dataFixPoint[jj], SHIFT)
221                             - FixPoint.mul(wi, dataFixPoint[jj + 1], SHIFT);
222 
223                     tempi = FixPoint.mul(wr, dataFixPoint[jj + 1], SHIFT)
224                             + FixPoint.mul(wi, dataFixPoint[jj], SHIFT);
225 
226                     dataFixPoint[jj] = dataFixPoint[ii] - tempr;
227 
228                     dataFixPoint[jj + 1] = dataFixPoint[ii + 1] - tempi;
229                     dataFixPoint[ii] += tempr;
230                     dataFixPoint[ii + 1] += tempi;
231                 }
232 
233                 wr = FixPoint.mul((wtemp = wr), wpr, SHIFT)
234                         - FixPoint.mul(wi, wpi, SHIFT) + wr;
235                 wi = FixPoint.mul(wi, wpr, SHIFT)
236                         + FixPoint.mul(wtemp, wpi, SHIFT) + wi;
237 
238             }
239             mmax = istep;
240             cnt++;
241         }
```

Listing 8.10: While Loop in FFT

The number of loops depends on the frame size by which the FFT needs to be calculated. The implementation with the array (see Code 8.10) uses *264 cycles*. The overall improvement is therefore a savings of cycles:

$$CyclesSaved = wpCalc - wpArray \Longrightarrow 13726 - 264 = 13462$$

The improvement of 13462 cycles per while loop is a speed improvement of this small piece of code of almost 52 times per loop.

WCET Analysis

The WCET analysis of the FFT is for transforming the signal from the time domain into the frequency domain and calculating the power spectrum of the signal. This analysis includes both using the FFT algorithm and taking the log to the power spectrum (See the "Example with DFT" in Section 5.2). Another part of the code that is used in the FFT calculation is the "maxshift", discussed in section 8.2.1. This is used for calculating the maximum shift value before transforming the signal with the log method. One of the difficulties with analyzing the FFT method is that some of the loops are dependent on other loops. The flow of these inner loops, therefore, needs to be independently analyzed and then added in order to get the final result, shown in Figure 8.7

The FFT shows the same pattern as the Window method; with a larger frame size, the time (number of cycles) grows linear (see Figure 8.7). The inverse FFT (IFFT) shows the same pattern as the FFT but is a lot faster than the FFT because it is not transforming the signal by using the log function (See Figure 8.7).

Figure 8.7 shows the graph of the execution time for the FFT and the IFFT.

The formulas for the FFT and the IFFT for calculating the execution times on JOP are:

FFT

$$y = 204521x - 408715 \tag{8.2}$$

IFFT

$$y = 12268x - 24261 \tag{8.3}$$

The reason why the formula is with a negative number (See 8.2 and 8.3) is that the method of FFT must be an integer power of 2. The negative number is therefore almost twice the number before x. It is only almost because some variables would be initialized even if the frame size were zero.

8.4.3 LPC

The algorithm for making LPC (Linear Predictive Coefficients) is described in Section 5.2.4, and as described in that section, there are different ways of implementing the algorithm. The chosen way is to implement an autocorrelation and then the LPC algorithm. This means that the implementation of the LPC is executed in two steps.

WCET Analysis

The WCET analysis of the LPC class shows that there is a linear connection between the frame size and the clock cycles if the coefficients are not variable for the two methods autocorrelation

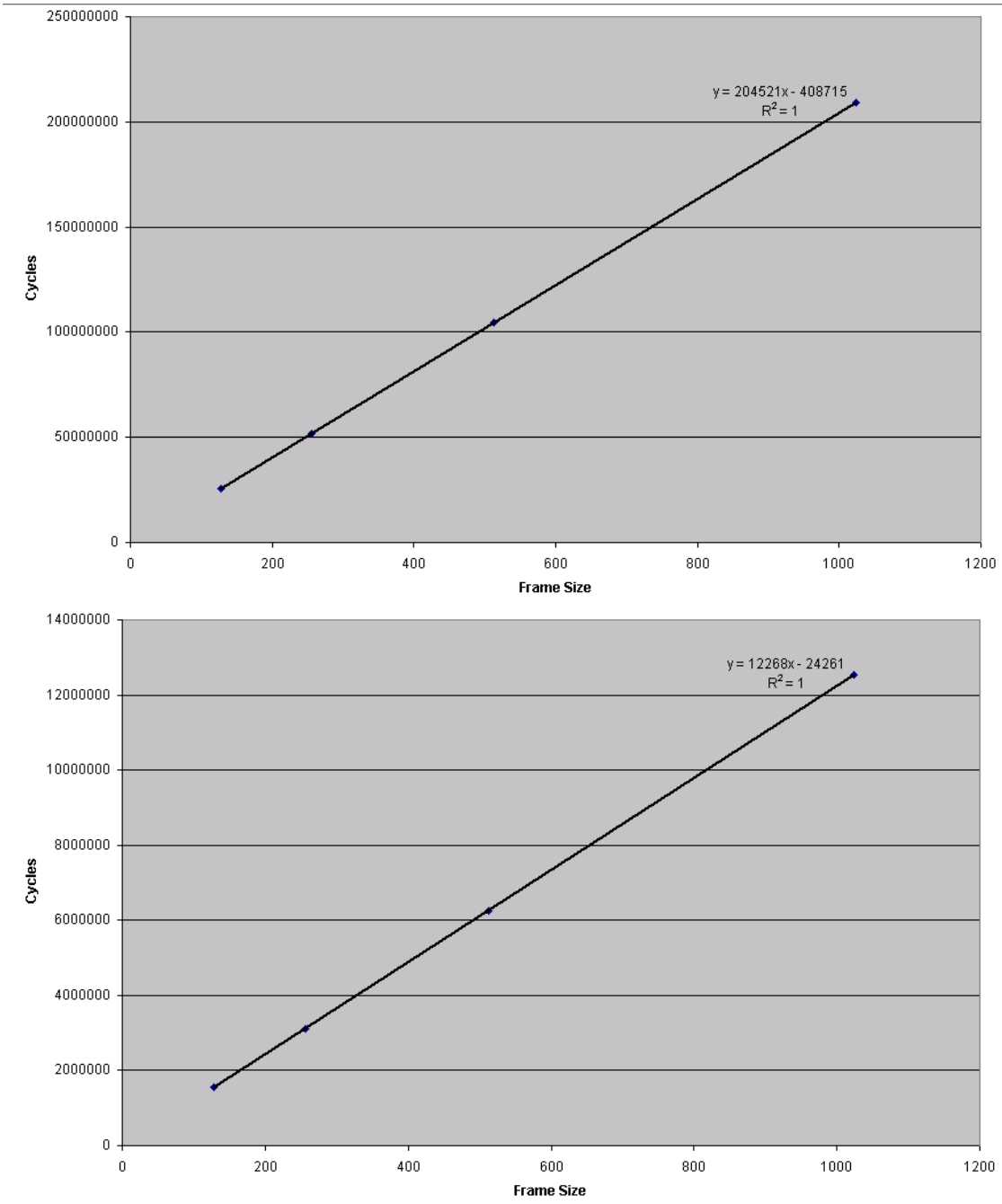


Figure 8.7: Clock Cycles for the FFT and IFFT, depending on frame size.

and LPC. If the frame size is the non-variable class, the formula is a polynomial, as can be seen in the Figure 8.8. The four formulas where the frame size is locked at different stages are:

128

$$y = 1032,5x^2 + 73252x + 577 \tag{8.4}$$

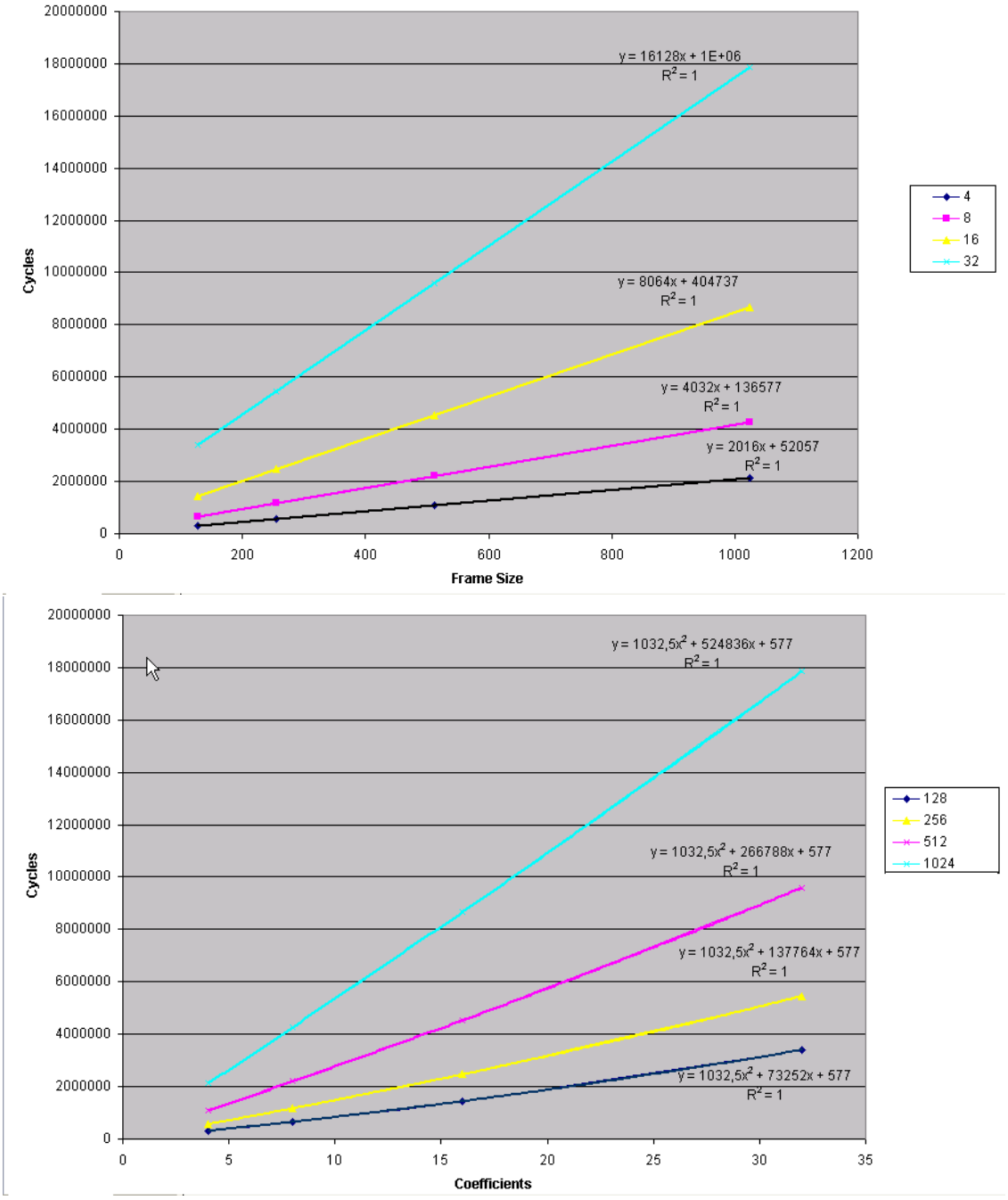


Figure 8.8: Clock Cycles for the LPC, dependent on frame size or the coefficients.

256

$$y = 1032,5x^2 + 137764x + 577 \tag{8.5}$$

512

$$y = 1032,5x^2 + 266788x + 577 \tag{8.6}$$

1024

$$y = 1032,5x^2 + 524836x + 577 \tag{8.7}$$

The four formulas where the coefficient is locked at different stages are:

4

$$y = 2016x + 52057 \tag{8.8}$$

8

$$y = 4032x + 136577 \tag{8.9}$$

16

$$y = 8064x + 404737 \tag{8.10}$$

32

$$y = 16128x + 1E + 06 \tag{8.11}$$

8.5 Recognizer Component

The implemented classes in the *Recognizer* can be seen in the Figure 8.9. The component provides the needed algorithms used for classifying, including distance measures, dynamic programming and classification models. At the time of writing it does not yet have any statistical models implemented.[11] The classes are implemented as either a *Model* or a *Classifier* type, with the needed helping classes providing distance measures and the necessary dynamic programming. All work which the component is responsible for should be implemented and specified in a `RecognizerWorker` class that extends the *Recognizer* class. In the *Recognizer* component, an example of the `RecognizerWorker` is implemented - showing how a Template can be classified and a model can be trained using cross validation on the K-near classification algorithm.

8.5.1 DTW

The `DTW` class is a implementation of the dynamic time warping algorithm explained in Section 5.3.2. The algorithm is implemented as a helping class for Classifiers and can be extended with different distance measures. The current version uses the Cepstral distance measure (see Section 5.3.2).

[11] See `www.jopspeech.com` for updates.

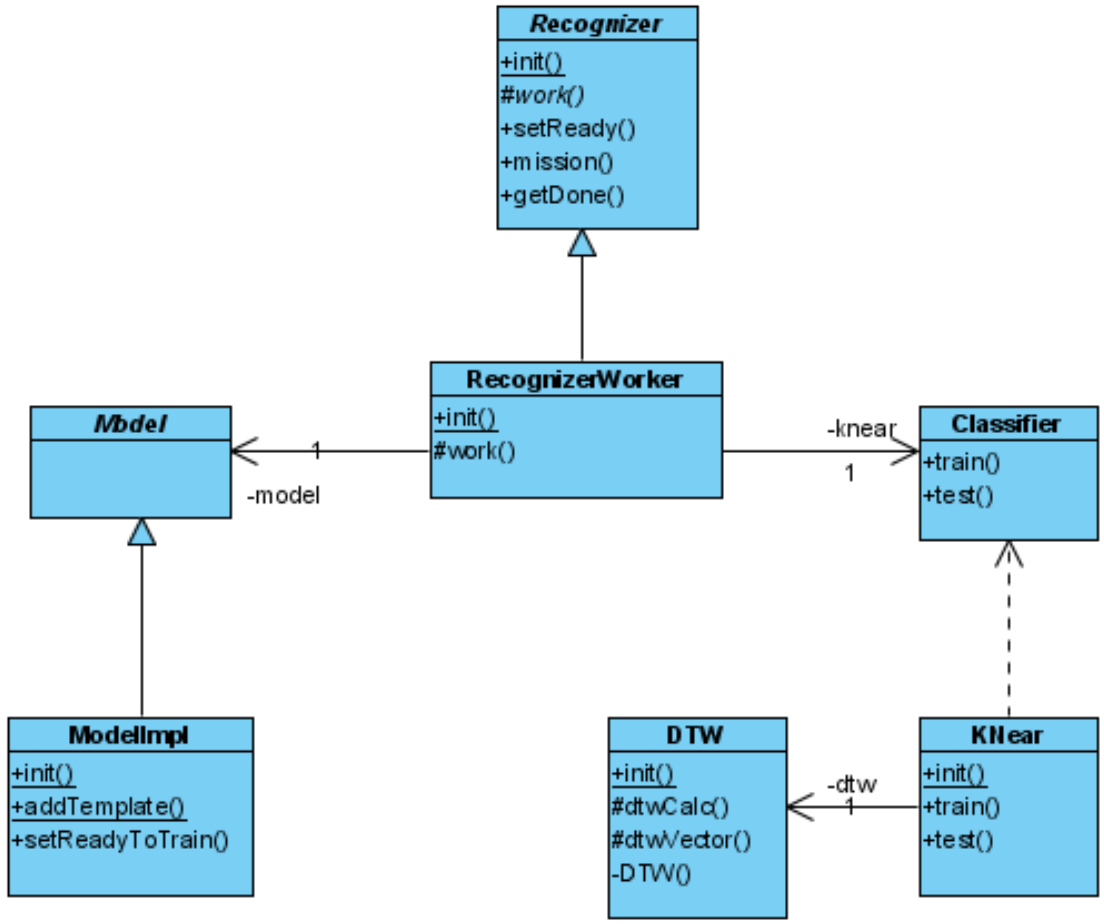


Figure 8.9: Recognizer Class Diagram

WCET Analysis

The WCET analysis of the DTW shows that there is a connection between the number of frames used in the application and the number of coefficients. Figure 8.10 shows the formula fund using Excel. As done with the analysis of the LPC, there is one variable that is held constant and one that is variable for the eight test cases.

The four formulas where the number of frames is constant and the coefficients are x and the cycles are y:

4

$$y = 10208x + 17589 \tag{8.12}$$

8

$$y = 40832x + 69017 \tag{8.13}$$

16

$$y = 163328x + 274689 \tag{8.14}$$

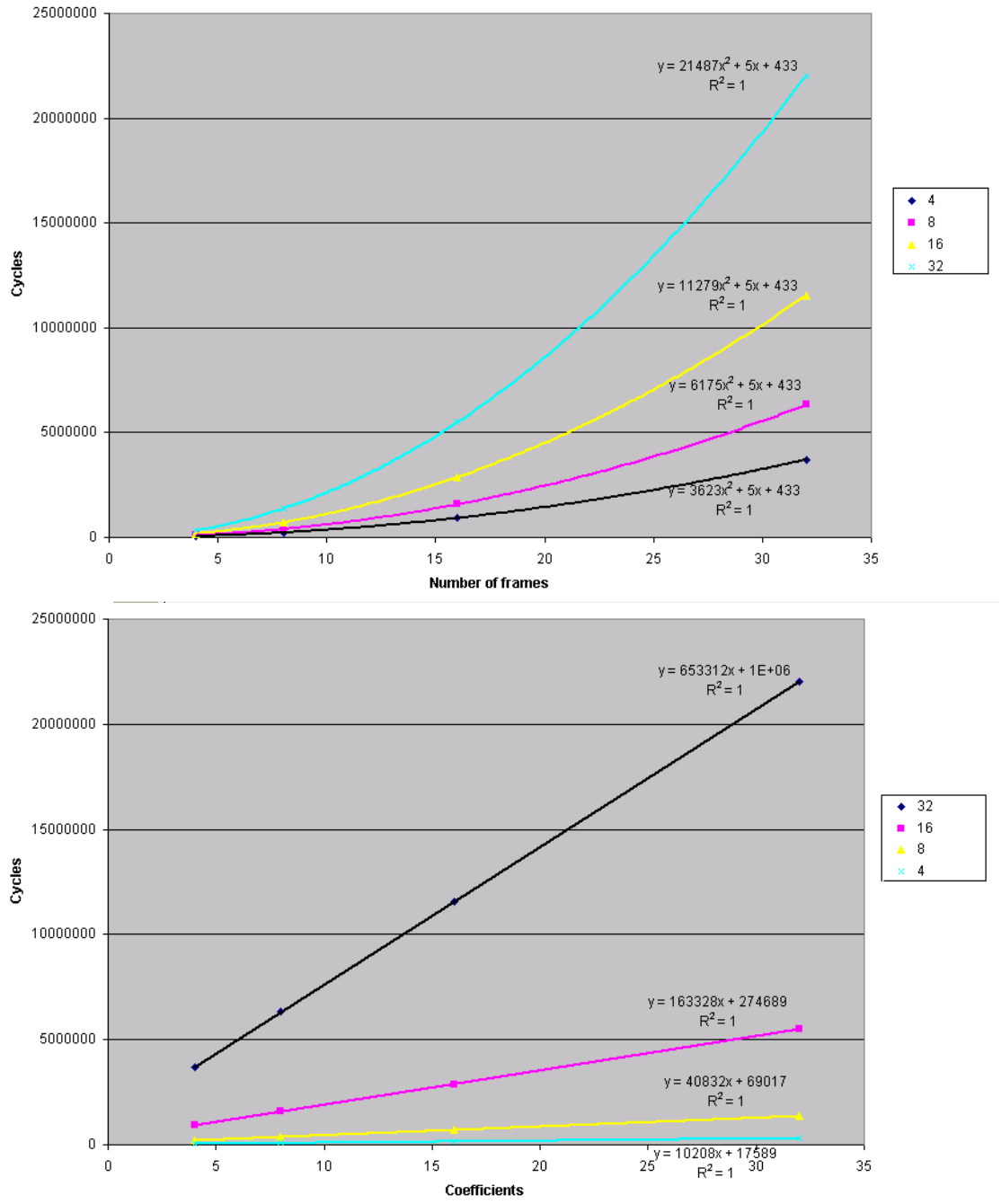


Figure 8.10: Clock Cycles for the LPC, dependent on frame size or the coefficients.

32

$$y = 653312x + 1E + 06 \tag{8.15}$$

The four formulas where the coefficient is a constant and the number of frames is x and the cycles are y:

4

$$y = 3623x2 + 5x + 433 \tag{8.16}$$

8

$$y = 6175x2 + 5x + 433 \tag{8.17}$$

16

$$y = 11279x2 + 5x + 433 \tag{8.18}$$

32

$$y = 21487x2 + 5x + 433 \tag{8.19}$$

Chapter 9

Implementation - Hardware

There are several ways of implementing the different techniques of DSP in embedded systems. Some of the techniques can even be implemented directly in hardware with time gain in the signal processing. In JopSpeech the objective has been that no algorithms should be implemented in hardware, as the possibility for users of the SDK to change the various settings is weighted higher than speed. However, although the objective has not been to implement anything in hardware, we still need to get signals in and out of the system, which demands a microphone and loudspeakers or another out-signaling device. Since The BaseIO board (see Figure 3.2), does not have any hardware interface for receiving signals, we had to include this hardware extension in the JopSpeech SDK as well. The hardware extension is described in this chapter. The hardware extension we have made is an amplifier with an ”Analog-to-Digital” and an amplifier with a ”Digital-to-Analog” converter. Further more, three LED lights are connected to the board. The hardware extension makes it possible to record and play sound through JOP and communicate with the user by the three LED lights. The print circuit board (PCB) has been designed specifically for the BaseIO board and therefore fits onto it seamlessly.

9.1 JopSpeech AD/DA

To ensure that the objective of keeping as much of the JopSpeech SDK in software as possible, some of the code has been implemented into VHDL[1] or "VHSIC Hardware Description Language". The first step, to get sound from a microphone to JOP, requires the implementation of an "Analog-to-Digital" converter (ADC). For the second step, getting sound from JOP to a loudspeaker, a "Digital-to-Analog" converter (DAC) is needed. Both have been implemented using both software and hardware where this could not be avoided. The VHDL code is described in Section 9.1.1 and 9.1.2 and the Java code has already been discussed (see Section 8.3.1).

9.1.1 ADC - Sigma-Delta Converter

The implemented ADC is a sigma-delta converter [19] which uses a small analog system consisting of two resistors and a capacitor (see Figure 9.1 R3,R4 and C5 and a VHDL part which reads current changes from the hardware part on the BaseIO board). The sigma-delta converter works by registering the electric pulses created in hardware. Every time there is a change in the pulses made by the hardware a bit is set in the VHDL code. Then between each sample recording, the bits are summed up (see VHDL code Listing 9.1), resulting in a quantization of the signal representing the discrete value of the sample.

```
147             if cnt=0 then
148                 cnt <= CNT_MAX-1;
149                 audio_in <= std_logic_vector(sum);
150                 sample_rdy <= '1';
151                 sum <= (others => '0');
152                 -- BTW: we miss one sigma-delta sample here...
153             else
154                 cnt <= cnt-1;
155                 if serdata='1' then
156                     sum <= sum+1;
157                 end if;
158             end if;
```

Listing 9.1: The Sigma-Delta converter sum

[1] VHDL code is made by Martin Schoeberl

The Sigma-Delta Frequency (”SDF”) is set at line 52 (see VHDL code 9.2) and then adjusted with the sampling frequency (also defined in VHDL), resulting in the CNT_MAX variable defining the quantization range or values (see Equation 9.1).

```
49 architecture rtl of sc_sigdel is
50
51     -- we use a 20MHz sigma-delta clock
52     constant SDF         : integer := 20000000;
53     constant SDTICK      : integer := (clk_freq+SDF/2)/SDF;
54     signal clksd         : integer range 0 to SDTICK;
55     constant CNT_MAX     : integer := (SDF+fsamp/2)/fsamp;
56     signal cnt           : integer range 0 to CNT_MAX;
57     signal dac_cnt       : integer range 0 to CNT_MAX;
```

Listing 9.2: Setting variables in the Sigma Delta

The sampling frequency (variable fsamp at line 470) of the ADC and the DAC is set in the VHDL code 9.3:

```
468     cmp_audio: entity work.sc_sigdel generic map (
469             addr_bits => SLAVE_ADDR_BITS,
470             fsamp => 8000
471         )
```

Listing 9.3: Setting the Sampling frequency to 8000

The equation for calculating the maximum sample value is:

$$CNTMAX = \frac{(SigmaDeltaFrequency + SampleFrequency/2)}{Frequency} \tag{9.1}$$

$$CNTMAX = \frac{(20000000 + 8000/2)}{8000} = 2500.5 \approx 2500 \tag{9.2}$$

9.1.2 DAC - Pulse-Width Modulation

The implemented "Digital-to-Analog" Converter (DAC) is based on Pulse-Width Modulation (PWM)[12]. When a digital signal has to be transformed into an analog pulse that can be played by a loudspeaker, the PWC method works by translating the signal into a set of high and low power outputs corresponding to the digital signal. The DAC is like the ADC implemented in both hardware and a VHDL. The VHDL code for doing the PWM can be seen in Listing 9.4, where the DAC is set to either 1 or 0. The variables that control the frequency at which the samples are played and the range of sample values are controlled by the variables described in Section 9.1.1.

```
192 process(clk, reset)
193
194 begin
195     if reset='1' then
196         delta <= (others => '0');
197         dac <= '0';
198         dac_cnt <= 0;
199     elsif rising_edge(clk) then
200
201         if dac_cnt=0 then
202             dac_cnt <= CNT_MAX-1;
203             delta <= unsigned(audio_out);
204         else
205             dac_cnt <= dac_cnt-1;
206             dac <= '0';
207             if delta /= 0 then
208                 delta <= delta-1;
209                 dac <= '1';
210             end if;
211         end if;
212
213     end if;
214 end process;
```

Listing 9.4: The PVM in sc_sigdel.vhd made by Martin Schoeberl.

9.1.3 The Circuit Diagram

Because not all microphones have an amplifier and the BaseIO board does not have a sound input and output, two different schematics[2] of the hardware are made. The difference between the two schematics is that the one shown in Figure 9.1 has an amplification of the input signal and the one in Appendix Figure B.3 does not have this amplification because it is meant to be use with a microphone with an amplifier.

The BaseIO has a 3.3V output from connector pin 1 that SV1 on the circuit diagram 9.1 is connected to. This means that the op-amp[3] needs to work with a low voltage. The data sheet [5] of the LMV321 shows that the op-amp can operate with a voltage between 2.7V and 5.5V. The

[2]The diagram and the board design are made in the free edition of the CAD program Eagle `http://www.cadsoft.de/`.

[3]Operational Amplifier

same goes for the TDA7050 (see data sheet [31]) which is the dual op-amp used in the DAC part of circuit diagram 9.1.

Figure 9.1 consists of the amplifier and the ADC at the top. In the middle the amplifier and the DAC are shown and in the bottom the three led lights are diagrammed.

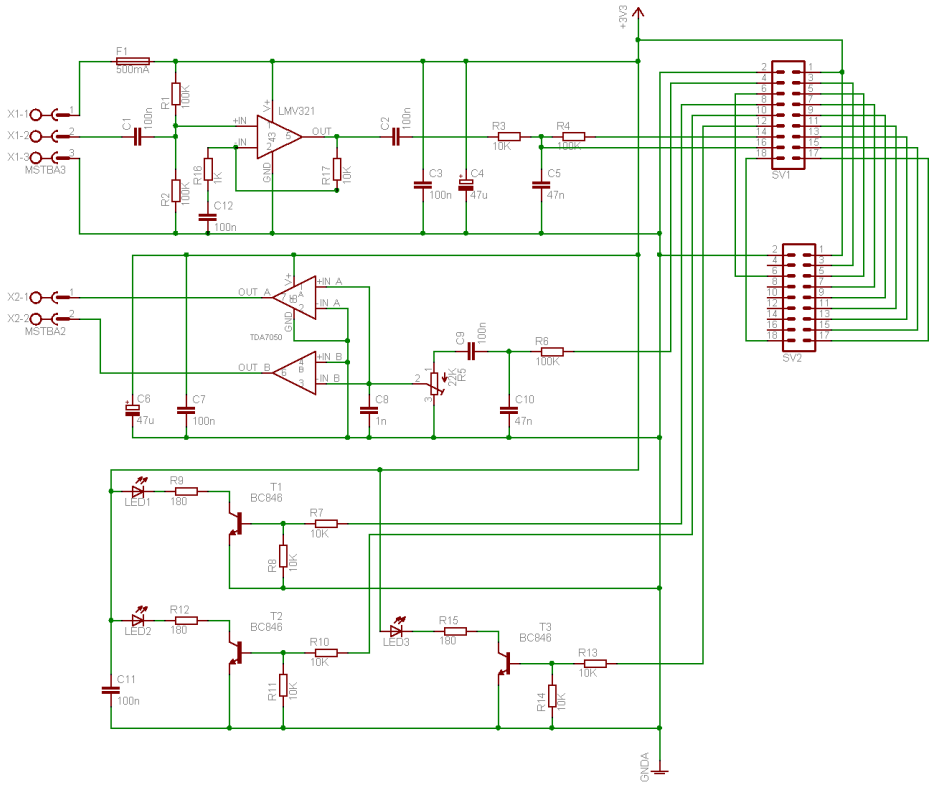


Figure 9.1: Circuit diagram of a ADC, DAC with amplifier, and three LED lights

There are two different types of small op-amps used in the diagram. The first op-amp (U1) is an LMV321[5]. That is a single op-amp used for amplifying the microphone signal if the microphone has no amplifier. When the microphone first registers a signal it emits a signal, and the capacitor (C1) then emits pulses. Then the two resistors R1 and R2 work as a voltage divider, with the dividing factor of two (see Equation 9.3), making the offset for the op-amp.

$$Voltagedividingfactor = \frac{R1}{R1 + R2} \tag{9.3}$$

The op-amp then amplifies the signal by a factor of ten because of the two resistors R16 and R17 connected to ground (see Equation 9.4), -IN and OUT. This implementation of the op-amp is called

a non-inverting amplifier circuit[30]. Equation 9.4 shows the calculation of the amplification ratio.

$$Ampilficationratio = \frac{R16 + R17}{R16} \tag{9.4}$$

The microphone circuit ends with the ADC part R3, R4, and C5. This part is connected to pin 14 and 16 on the SV1, which connects to the BaseIO boards connecter SV1[4]. These two pins are the ones used in the VHDL code (see Listing 9.5) to do the sigma-delta converter.

In the VHDL code, Listing 9.5, it is also possible to see that the DAC uses pin 4 to emit the sound[5].

```
369     -- l(12) <= lo(14);
370     l(13) <= lo(13);
371     -- l(14) <= lo(12);
372     l(15) <= lo(11);
373     b(1) <= lo(10);
374     b(2) <= lo(9);
375     b(3) <= lo(8);
376     b(4) <= lo(7);
377     b(5) <= lo(6);
378     b(6) <= lo(5);
379     b(7) <= lo(4);
380     b(8) <= lo(3);
381     -- b(9) <= lo(2);
382     b(10) <= lo(1);
383     l(16) <= bat;
384
385     -- we use three of the LED pins for the micro sigdel
386     -- and the audio out:
387     mic_sdi <= l(12);
388     l(14) <= mic_sdo;
389     b(9) <= audio_out;
```

Listing 9.5: Part of scio_mikjen.vhd made by Martin Schoeberl

For the DAC there is a variable resistor (R5) makes it possible possibility to control the volume of the signal sent from JOP.

The last part of the schematics shows the connection of three LED lights that are controlled by sending a signal to one of the three pins 6, 8, or 10. The three LED lights are meant to be green, yellow and red. These three LED lights can be used for communicating with the user by turning different combinations of the LED lights on or off.

For the schematics, a board layout[6] has also been made (see Figure 9.2).The half circle is made for the board to fit on the BaseIO.

The drawings 9.1 and 9.2 are only examples of what the real board could look like, with the minimum features. Our final manufactured board has some extra features, including filters to block

[4]see Appendix Section B.1 for the diagram of JOP

[5]The mapping of the pins and the VHDL code can be seen in Appendix Section B.1 where the SV1 is diagrammed.

[6]The layout of the board is reviewed and the design issues are discussed with two engineers from the firm TELETRONIC `www.teletronic.dk`.

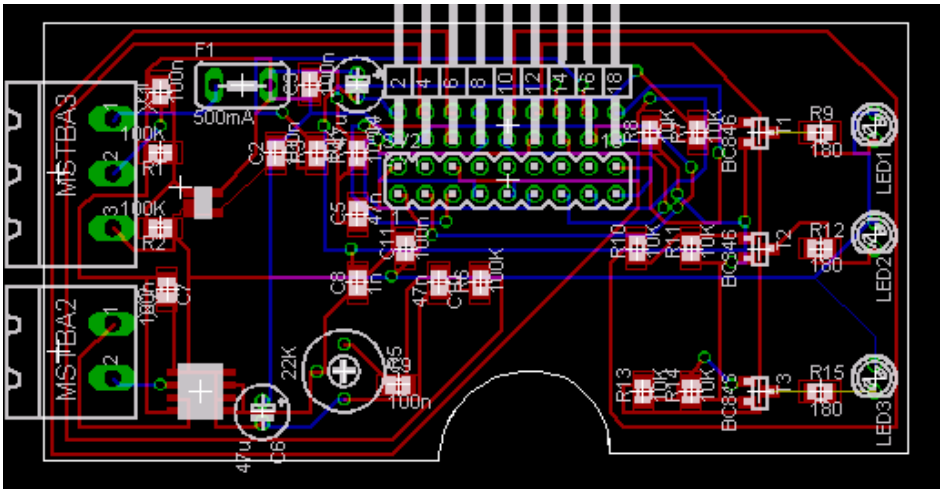


Figure 9.2: Board layout of diagram 9.1

noise from interference signals such as mobil-phones. The final AD/DA extension board can be seen in Figure 9.3.

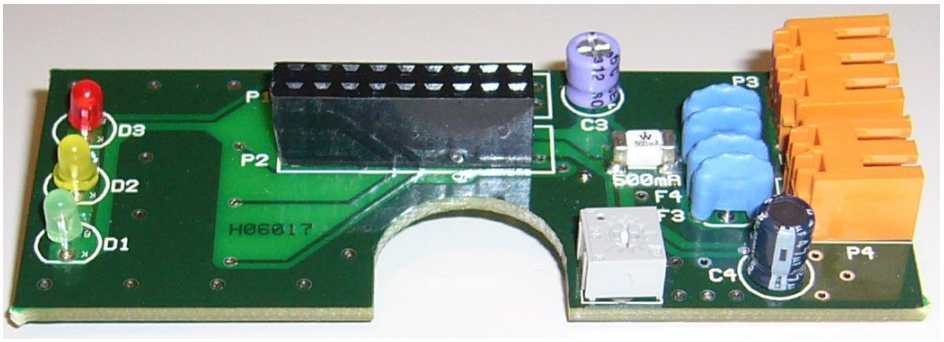


Figure 9.3: The final JopSpeech AD/DA extension ©

For testing the diagram the hardware shown in Figure 9.4 has been used. The board connected to BaseIO makes it possible to change the component without any soldering.

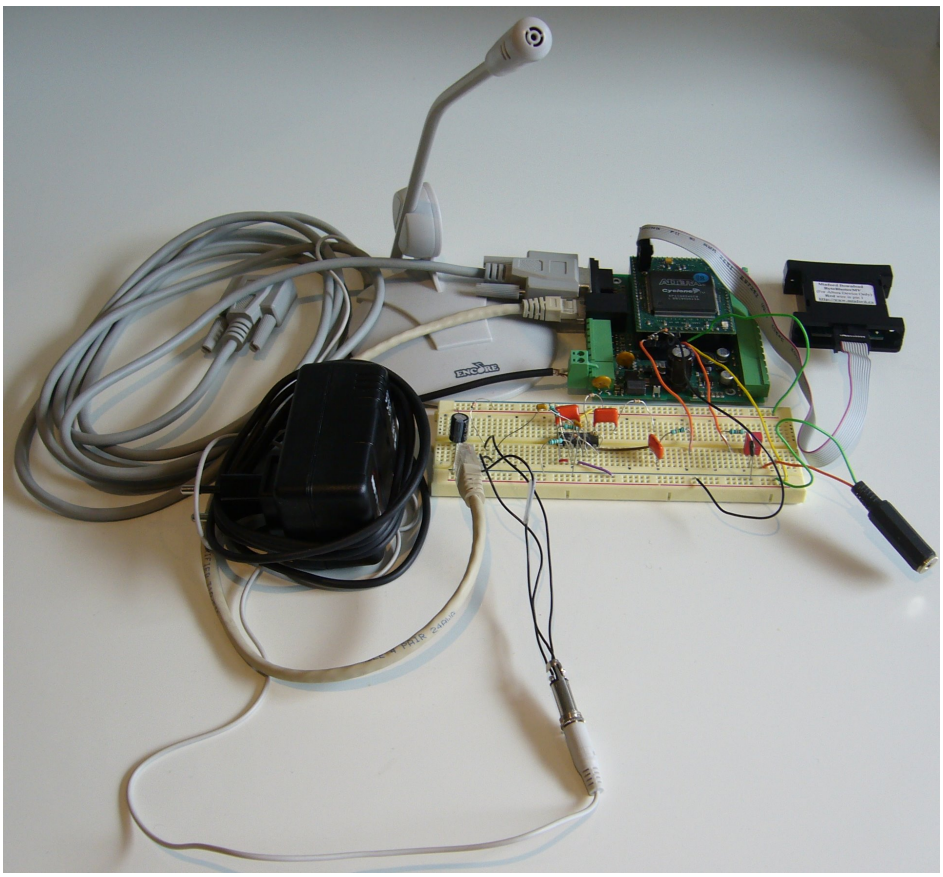


Figure 9.4: The hardware used for testing

Chapter 10

Experiments

In this chapter we will demonstrate the usability of the JopSpeech SDK and show how different experiments can be performed. Two prototypes are developed and we will conduct four different experiments with both of the prototypes. The prototypes have been developed with the purpose of demonstrating both how the JopSpeech SDK architecture can be used to create a networked distributed speech recognition system and to demonstrate the use of the SDK in identifying performance enhancing solutions that can improve speech recognition system.

10.1 Experimental Design: The Prototypes

To ensure comparison between the prediction and classification time between the different experiments, we have designed the `Workers` implementations in the two prototypes more or less similar, so that only the implementation of the `AnalysisWorker` will differ[1]. This means that the FrontEnd and Recognizer components‘ mission phase will be held static, which justify the comparison between the results of the experiments. Although the mission phases of the prototypes are similar, the process distribution in the initialization phases will differ. This has been done in order to experiment with a distributed solution and show the potential in JopSpeech of how it can be implemented. The following is a description of the two prototypes and how they are constructed:

10.1.1 Common Features Of The Prototypes

Both prototypes have some common features and run with the same thread priorities and time settings.

[1]The `Workers` are the extensions to each component (FrontEnd, Analysis, Recognizer), defining the work that needs to be done in each component (See Chapter 8 for more information).

Local Process Distribution

There is one main class `Jopspeech` which is in charge of the initialization phase, initializing the other classes used (see Figure 8.1). When the initialization is done the main threads only responsibility is to check WD (Watch Dog) to ensure that JOP is ”running”. Then for each of the three components there is a thread that controls the work and data flow of the application. These are called `FrontEndWorker`, `AnalysisWorker` and `RecognizeWorker` (see Section 7.3 for the dataflow).

The initialization done by `Jopspeech` consists of initiating five different threads. Two threads are used for the Ethernet (`Linklayer` sends and `Net` receives packets) and the last three threads are used for the `FrontEndWorker`, `AnalysisWorker` and `RecognizeWorker`. The setup and initialization of the five different threads is done by setting the time and priority of the thread. The priority and time for each thread are set by using the WCET analysis (see Chapter 8). This sets the time fairly high to ensure that each thread can finish within its time period, thereby ensuring that no context switch interferes with the results. In Listing 10.1 you can see the setup of the `FrontendWorker` thread.

```
98          new RtThread(5, 5000) {
99              public void run() {
100                 for (;;) {
101                     waitForNextPeriod();
102                     frontEnd.mission();
103                 }
104             }
105         };
```

Listing 10.1: Thread setup of the `FrontEndWorker` class

To ensure that the same signals are being experimented with, each signal is pre-recorded (see Section 10.2), and loaded from a PC using a network solution via the ethernet using the two threads `"Net"` and `"LinkLayer"` (See Appendix B.3).

Load Signal

The Signal class is implemented as the class Utterance (see Appendix B.3) an extension of the interface and is used to contain the raw signal of a speech signal. The Utterance class is implemented using the singleton pattern (as specified by the application structure Section 8.1) ensuring that there is only one utterance in the application. The actual loading of the Utterance is handled by the FrontEndWorker (as specified in Section 7.1). The flow is depicted in Figure 10.1. The signals are loaded into the prototypes by sending the words by UDP. The packages are sent with 80 values at a time. To be sure that all the packages arrive at JOP, an acknowledge package is sent to the computer that tells if JOP is ready for a new command. If this package is not received by the

computer, the same package is resent.

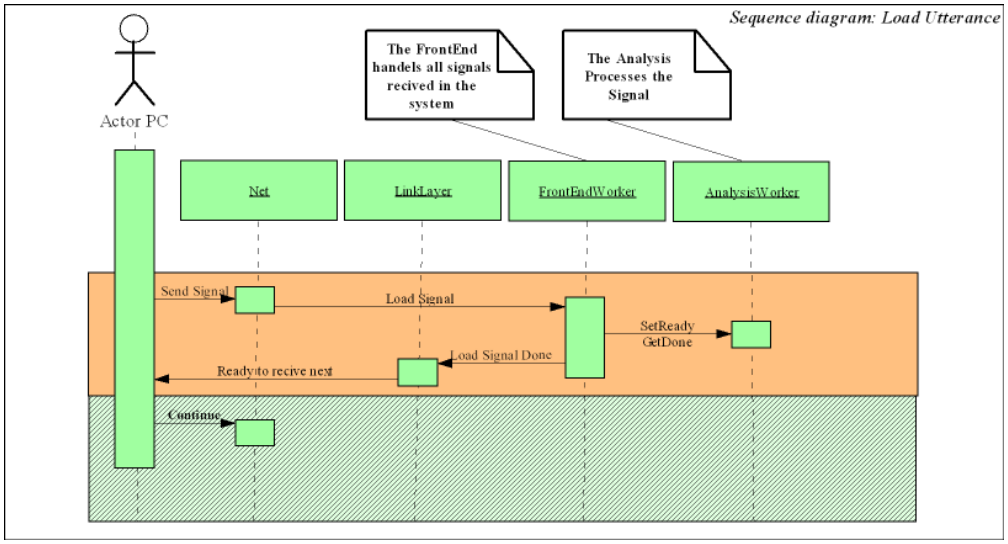


Figure 10.1: Loading of signal from PC

Classify And Train

The Training and classification of the model is the responsibility of the Recognizer component controlled by the defined `work` method in the `RecognizeWorker`. Work defined in `RecognizeWorker` (see Code below 10.2) shows how the training is started when the model is ready to be trained (which is when the model is loaded). Recognition is done by comparing the test *Word*[2] against the trained model using the K-Near classification algorithm.

```
30
31      protected void work() {
32          // test
33          if (model.trained == 1)
34              knear.test(AnalysisWorker.template, model);
35          // train
36          else if (model.readyToTrain == 1) {
37              knear.train(model);
38              model.readyToTrain = 0;
39              knear.test(AnalysisWorker.template, model);
40          }
41      }
```

Listing 10.2: Work defined in `RecognizeWorker` class

The process distribution structure with the threads executing in a straight order is the most optimal solution on a single processing unit, as the recognition part is waiting for the analysis part to finish before it starts the recognition. It could be argued that it would be more efficient if the recognition could start when the first frame of the word to be tested was made by analysis.

[2]The class *Word* is an extension of the interface ”Template”. Other interface implementations of ”Template” could be, for example, a phoneme

10.1.2 Prototype One: The Linear Predictive Based

The difference between the two prototypes is in the `work` method, defined in the Analysis components mission phase and the process distribution between a PC and JOP. The Linear Predictive based prototype is based on the following setup:

Initialization Phase

The Initialization phase is done on JOP, which means that all non-time-critical methods, variables calculations and initializations are done on the FPGA board, and all values used for later calculations and references etc. are therefore done in a single processor. The reason for handling all work in the init phase on JOP in this experiment is to test if the implementation of JopSpeech can handle memory as a limited resource and reuse memory objects, as we have designed it to do.

Mission Phase

The Analysis mission phase in prototype one is defined to process the signal and deliver the result in Linear Predictive coefficients (LPC), build from the cepstrum. Figure 10.2 shows the whole analysis workflow very well, and the code snippet is shown in Section 8.1 in listing 8.2. If we refer back to the implementation paths in the Figure 7.5, the path the mission phase would follow, is (B,D,E,H,F,K).

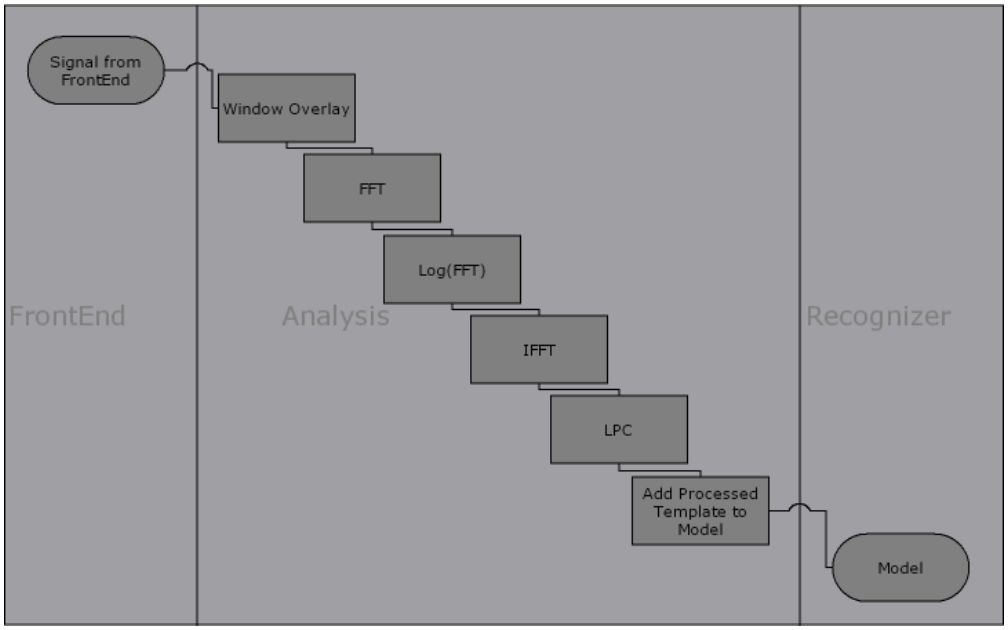


Figure 10.2: Analysis Component sequence flow: Prototype one

10.1.3 Prototype Two: The Mel Frequency Filter Bank Based

The ”Mel Frequency Filter Bank” prototype is based on the following setup:

Initialization Phase

The initialization phase is distributed between the PC and the JOP, utilizing some of the superior processing power the PC's CPU possesses. All Mel filter banks are calculated on the PC, and the banks with a Mel scale distribution (reprinted below in Equation 10.1) are then sent to JOP when needed in the process.

$$MF(m) = \sum_{k=0}^{N-1} H(X_k, m) * Spectral(X_k) \tag{10.1}$$

Mission Phase

The Mission phase is different than the prototype one, as we are working with a mel scale signal filter bank distribution. The code snippet for the `AnalysisWorker` defining the work can be seen in Listing 10.3. If we refer back to the implementation paths in Figure 7.5 the path the mission phase would follow are (B,D,E,I,J,K). The implementation workflow can be seen in Figure 10.3

```
55    protected void work() {
56        // Window Algorithm
57        window.process(0, FrontEndWorker.utterance, template);
58        // FFT Algorithm
59        fft.process(1, FrontEndWorker.utterance, template);
60        // IFFT Algorithm
61        fft.process(-1, FrontEndWorker.utterance, template);
62        // Mel Cepstral Filterbank Algorithm
63        mfcc.process(0, FrontEndWorker.utterance, template);
64        // If the template is not unknown add it to the Model
65        if (!FrontEndWorker.utterance.classID.regionMatches(0, "UNKNOWN", 0, 7)) {
66            // Adding Template
67            ModelImpl.addTemplate(FrontEndWorker.utterance, template,
68                    NUMBERCOEFFICIENTS);
69        }
70        // Else start the recognizer and it will have defined
71        // in its work method what to do with the template
72        else {
73            JopSpeech.recognizer.setReady();
74        }
75    }
```

Listing 10.3: Work defined in `AnalysisWorker` class

10.2 Performance Parameters

Each of the two Prototypes will be subject to 4 different experiments, where the experiments will either be a Speaker Dependent or Semi-independent system. By Semi-independent we mean a

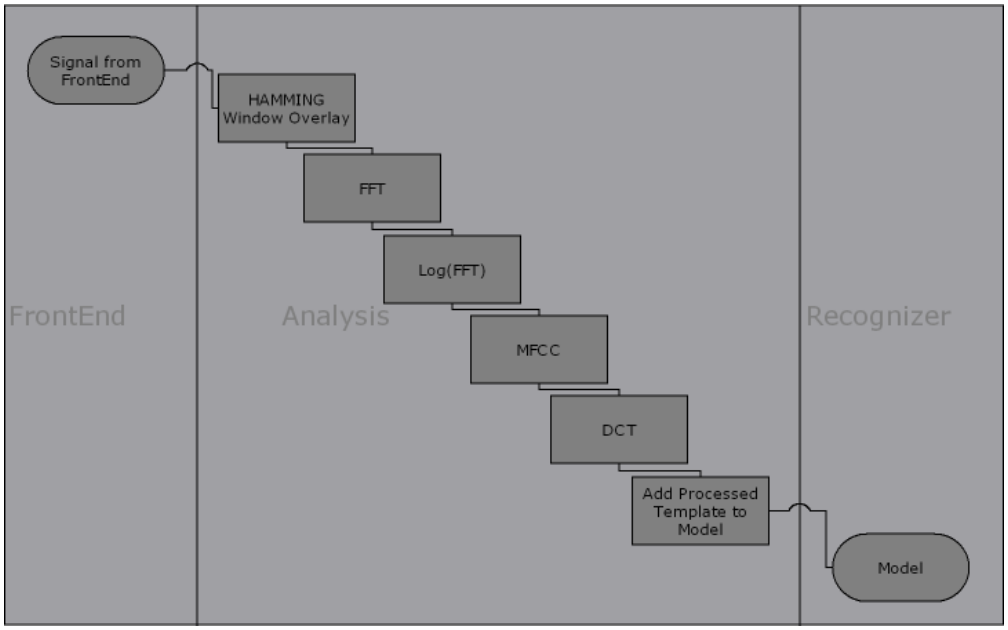


Figure 10.3: Analysis Component sequence flow: Prototype two

system that is trained for two persons, which is the closest to an independent system that we will experiment with. If the experiments had to be properly Speaker independent, the training set needed to represent talkers from different age groups, with different accents, different speaking rates, different speaking levels, and different context conditions would also needed to be taken into account[25]. As this would not have affected our experiments we decided to limit the experiment to two different speakers, recorded under different contexts (noise in the background, etc.).

Template Setup:

1. Speaker Dependent: One speaker and 3 templates for each word and the rest as test words.
2. Speaker Dependent: One speaker and 5 templates for each word and the rest as test words.
3. Speaker Semi-independent: Two speakers and 3 templates for each word and the rest as test words.
4. Speaker Semi-independent: Two speakers and 5 templates for each word and the rest as test words.

Performance Metrics:

1. Prediction score
2. Time to Predict

Zero	One	Two	Three	Four	Five	Six	Seven	Eight	Nine	Program	Start	Stop
10	10	10	10	10	10	10	10	10	10	10	10	10

Table 10.1: The different words and their representation in the data set pr. speaker

The experiments have the following performance parameters. We will be investigating them according to the performance metrics:

Number of speakers What influence the number of speakers has on the performance metrics.

Number of templates What influence the number of templates has on the performance metrics.

Precision What influence the number of coefficients used to represent the sound signal has on the performance metrics.

The list of words chosen as points of reference for the experiments is in Table 10.1. These words were chosen as they could be used in an embedded speech recognition system for a machine that needed to be started and stopped or programmed. The words are all uttered 10 times per speaker.

10.2.1 Memory - Overall Memory Usages

One of the issues with embedded systems is memory usage (see Section 3.1). On JOP the memory available is approximately 512Kb, and because the garbage collector is not implemented the heap becomes immortal (see Chapter 3). The memory usage therefore presents developers with some design issues. However, as there are no dynamic structures implemented in JOP and the design pattern specifies that all initialization should happen in the start of the application, it should be possible to calculate the exact memory usage of the application‘s various experiments, if JopSpeech has been implemented correct.

The general memory usage pattern used for the SDK is that no object or array is created without thinking of how to reuse the memory, the Singleton pattern and the static references are often used. The only memory usage that needs to be accounted for by developers using the SDK is the number of variable sizes set in the worker classes `FrontEndWorker`, `AnalysisWorker`, the overlay percentages in `Window` and the size of the model `ModelImpl`.

FrontEndWorker SAMPLESIZE This is the maximum sample size that the system can use. In the equations below 10.2 it equals $FrWrk$.

AnalysisWorker NUMBERCOEFFICIENTS This is the number of coefficients per frame used for recognition. It defines how many LPC's or MLCC's are made. In the next equations it equals $NumCof$.

FRAMESIZE This is the frame size used for making windows. In the equations below it equals $FrmSz$.

Window OVERLAYPRC This is the percentage of overlay the windows cowers each other with.In the equations below it equals $OvLay$.

ModelImpl NUMTEMPLATES This indicates how many templates of words are used in JOP. In the equations below it equals $NumTmp$.

The most memory-consuming objects in the experiments are the array structures that are used in the different classes just mentioned. For each array, the following equations should define the memory usages. If this is not true, some of our experiments will not be able to run. The elements of the equations consist of the five variables. The equations are:

Class: Utterance
`frames[][]`

$$\frac{FrWrk}{FrmSz} * OvLay * FrmSz \tag{10.2}$$

`raw_sample[]`

$$FrWrk \tag{10.3}$$

Class: Word
coefficients[][]

$$\frac{FrWrk}{FrmSz} * OvLay * NumCof \tag{10.4}$$

Class: ModelImpl `words[]`

$$NumTmp * \frac{FrWrk}{FrmSz} * OvLay * NumCof \tag{10.5}$$

Class: Window
`windowValues[]`

$$FrmSz \tag{10.6}$$

Class: FFT
`compSample[]`

$$FrmSz * 2 \tag{10.7}$$

Class: LPC
`autoCoeff[]`

$$NumCof + 1 \tag{10.8}$$

Class: DTW

`dist[][]`

$$\frac{FrWrk}{FrmSz} * OvLay^{2} \tag{10.9}$$

`move[][]`

$$\frac{FrWrk}{FrmSz} * OvLay^{2} \tag{10.10}$$

`globDist[][]`

$$\frac{FrWrk}{FrmSz} * OvLay^{2} \tag{10.11}$$

`temp[][]`

$$\frac{FrWrk}{FrmSz} * OvLay * 2 * 2 \tag{10.12}$$

`warp[][]`

$$\frac{FrWrk}{FrmSz} * OvLay * 2 * 2 \tag{10.13}$$

`compare[]`

$$NumTmp \tag{10.14}$$

Table 10.2 shows the memory usage of the first experiment[3] and the equation for each array. The table is an example with the following variables:

NUMBERCOEFFICIENTS= 12

FRAMESIZE= 512

OVERLAYPRC= 50

SAMPLESIZE= 8192

NUMTEMPLATES= 130

In this example the memory usages do not excise the total memory usages for JOP (512Kb) and the application can run with the settings.

[3] see Section 7.4

Equation	Pre-calculation	Memory in integer
10.2		16384
10.3		8192
10.4	416	
10.5		54080
10.7		1024
10.8		13
10.9		1024
10.10		1024
10.11		1024
10.12		128
10.13		128
10.14		130
Integer usages		83151
Memory usages in bytes		332604

Table 10.2: Memory usages on JOP

10.3 Results

The result from running the four experiments twice adds up to representing 8 different experiments. Each experiments were manufactured to test the performance metrics with different settings, to see if any optimums or improvements could be identified. Each experiment were tested with three different frame sizes; 256 samples, 512 samples and 1024 samples. The signal representation for each frame size, were tested with an representation of 3 - 30 coefficient. However as the number of coefficients grows so do the memory consumption, and because of the memory restrictions on JOP (see equations above in section 10.2.1) not all test could be conducted. The results of the 6 first experiments can be seen below in the two tables: Table 10.3 and Table 10.4, the rest of the results of the experiments can be found in Appendix A.1.

The ”Time to Predict” shown in the tables is calculated by averaging the time of the prediction of all the words in each experiment.

The ”Prediction Score” is calculated dividing the number of correct predictions by all the number of test words.

Linear Predictive Coefficients Speaker Dependent Experiment Three Templates

The table 10.3 show the results for the first experiments using three templates as *training set* to train the model on a Speaker Dependent speech recognition system using Linear Predictive Coefficients (LPC) to build the templates and represent the signals.

Frame size	256		512		1024	
Coefficient	Avg. time	Prediction %	Avg. time	Prediction %	Avg. time	Prediction %
3	7,15	86,21%	6,08	87,07%	5,21	80,17%
4	7,40	87,93%	6,11	89,66%	5,21	84,48%
5	7,64	87,93%	6,13	91,38%	5,24	84,48%
6	7,89	88,79%	6,17	90,52%	5,26	87,07%
7	8,13	88,79%	6,22	89,66%	5,28	85,34%
8	8,37	88,79%	6,27	90,52%	5,27	83,62%
9	8,57	89,66%	6,28	92,24%	5,31	87,93%
10	8,81	89,66%	6,34	92,24%	5,31	88,79%
11	9,06	89,66%	6,40	91,38%	5,33	88,79%
12	9,29	89,66%	6,46	92,24%	5,34	87,07%
13	9,54	89,66%	6,53	89,66%	5,33	87,07%
14	9,78	88,79%	6,59	90,52%	5,38	88,79%
15	10,03	90,52%	6,67	88,79%	5,39	87,07%
16	10,28	91,38%	6,74	89,66%	5,41	87,93%
17	10,52	90,52%	6,81	91,38%	5,44	86,21%
18	10,77	90,52%	6,88	90,52%	5,46	85,34%
19	11,01	90,52%	6,95	90,52%	5,48	84,48%
20	11,26	90,52%	7,02	90,52%	5,51	85,34%
21			7,09	88,79%	5,51	87,07%
22			7,16	88,79%	5,54	84,48%
23			7,23	89,66%	5,55	85,34%
24			7,30	89,66%	5,58	83,62%
25			7,38	88,79%	5,58	82,76%
26			7,45	87,07%	5,61	87,07%
27			7,52	87,93%	5,65	85,34%
28			7,59	86,21%	5,66	85,34%
29			7,67	87,07%	5,66	85,34%

Table 10.3: Result for the LPC Prototype on a Speaker dependent, 3 templates model

In Figure 10.4 the prediction and time are shown. From the figure one can see an optimal solution exits with a frame size of 512 and a representation of 9 - 12 coefficients which would yield a prediction score of 92,24%. The prediction time is lowest at nine coefficients with an average of 6,28 seconds. From this we can conclude that building an Speaker Dependent speech recognition system, as the LPC Prototype will find its optimum with a framesize of 512 with a

average prediction score of 92,24%.

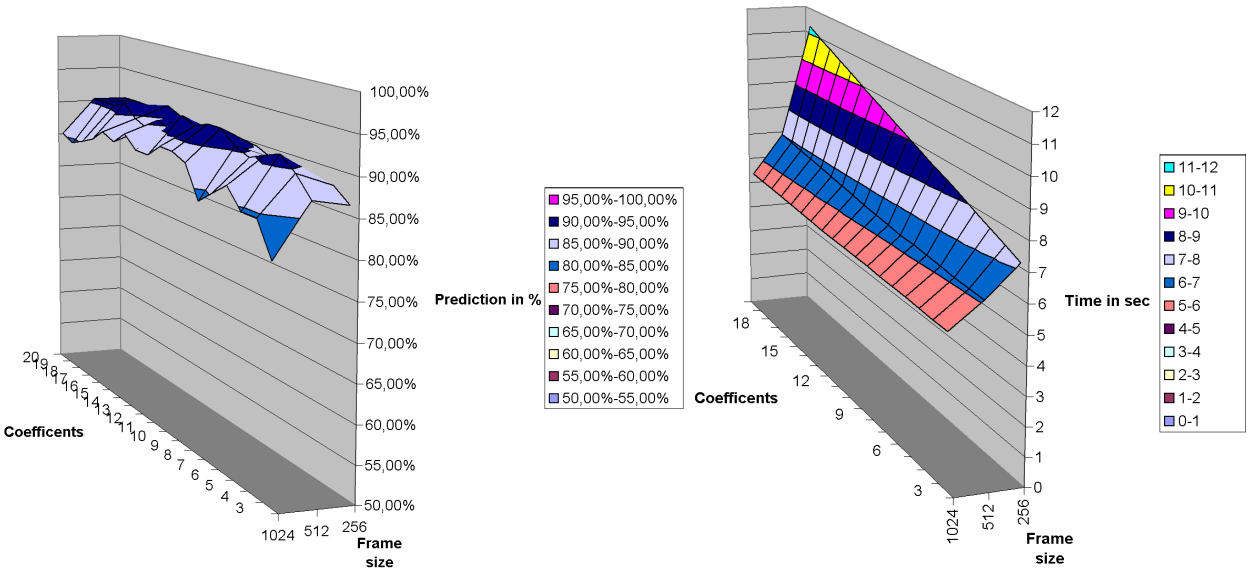


Figure 10.4: Result for the LPC Prototype on a Speaker dependent, 3 templates model

Mel-Frequency Cepstrum Speaker Dependent Experiment Three Templates

The table 10.4 show the results for the experiments using three templates as *training set* to train the model on a Speaker Dependent speech recognition system using Mel-Frequency Cepstrum Coefficients (MFCC) to build the templates and represent the signals.

Frame size	256		512		1024	
Coefficient	Avg. time	Prediction %	Avg. time	Prediction %	Avg. time	Prediction %
3	5,71	56,90%	4,62	59,48%	3,88	58,62%
4	5,93	65,52%	4,64	56,03%	3,9	53,45%
5	6,15	63,79%	4,64	52,59%	3,88	62,07%
6	6,37	67,24%	4,67	56,03%	3,9	65,52%
7	6,59	67,24%	4,69	62,07%	3,89	69,83%
8	6,8	56,90%	4,72	51,72%	3,9	64,66%
9	7,02	65,52%	4,75	64,66%	3,89	68,97%
10	7,24	61,21%	4,78	62,07%	3,91	65,52%
11	7,46	65,52%	4,83	68,10%	3,9	77,59%
12	7,69	68,97%	4,87	65,52%	3,9	68,97%
13	7,91	56,90%	4,92	67,24%	3,92	59,48%
14	8,13	15,52%	4,96	39,66%	3,9	14,66%
15	8,35	13,79%	5,02	22,41%	3,92	16,38%
16	8,57	52,59%	5,07	67,24%	3,92	27,59%
17	8,8	31,90%	5,12	61,21%	3,94	24,14%
18	9,02	6,90%	5,17	12,07%	3,91	8,62%
19	9,25	2,59%	5,22	8,62%	3,94	9,48%
20	9,48	6,03%	5,27	5,17%	3,94	12,07%
21			5,32	5,17%	3,93	11,21%
22			5,38	7,76%	3,95	6,03%
23			5,43	12,07%	3,95	6,03%
24			5,48	10,34%	3,96	10,34%
25			5,53	5,17%	3,97	6,03%
26			5,59	6,03%	3,95	6,03%
27			5,64	6,90%	3,96	8,62%
28			5,7	5,17%	3,97	9,48%
29			5,75	7,76%	3,97	17,24%

Table 10.4: Result for the MFCC prototype on a Speaker dependent, 3 templates model

In Figure 10.5 the prediction and time are shown. From the figure one can see the an optimal solution exits with a frame size of 1024 and a representation of 11 coefficients which would yield a prediction score of 77,59%. The prediction time is 3,9 seconds at eleven coefficients. From the Figure 10.5 we also notice that the prediction score varies a lot as the number of coefficients grows. This means that there might be a potential bug in the system and that we cant conclude that

building an Speaker Dependent speech recognition system, as the MFCC Prototype will find its optimum with a framesize of 1024.

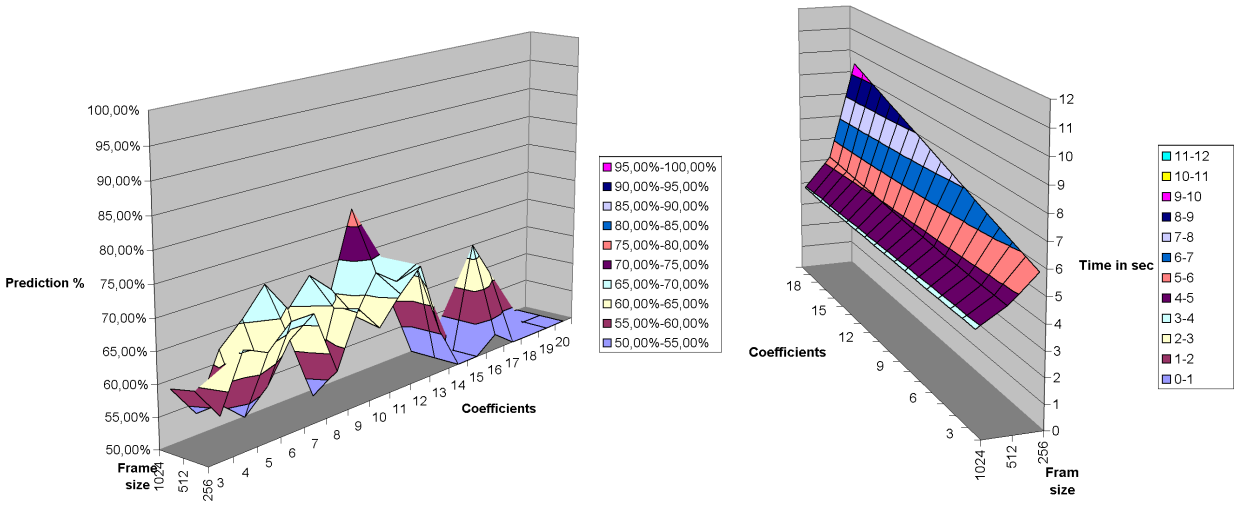


Figure 10.5: Result for the MFCC prototype on a Speaker dependent, 3 templates model

Speaker Semi-Independent Experiments Five Templates

The Figures 10.6 and 10.6 shows the results from the Mel-Frequency Cepstrum and Linear Predictive prototype, in a semi-independent (two users) speech recognition system, using five templates as *training set*. The tables for the results can be found in Appendix A.1.

Again here it can be noticed that the LPC prototype outperforms the MFCC prototype in respect to predictive score, while the MFCC is several seconds faster.

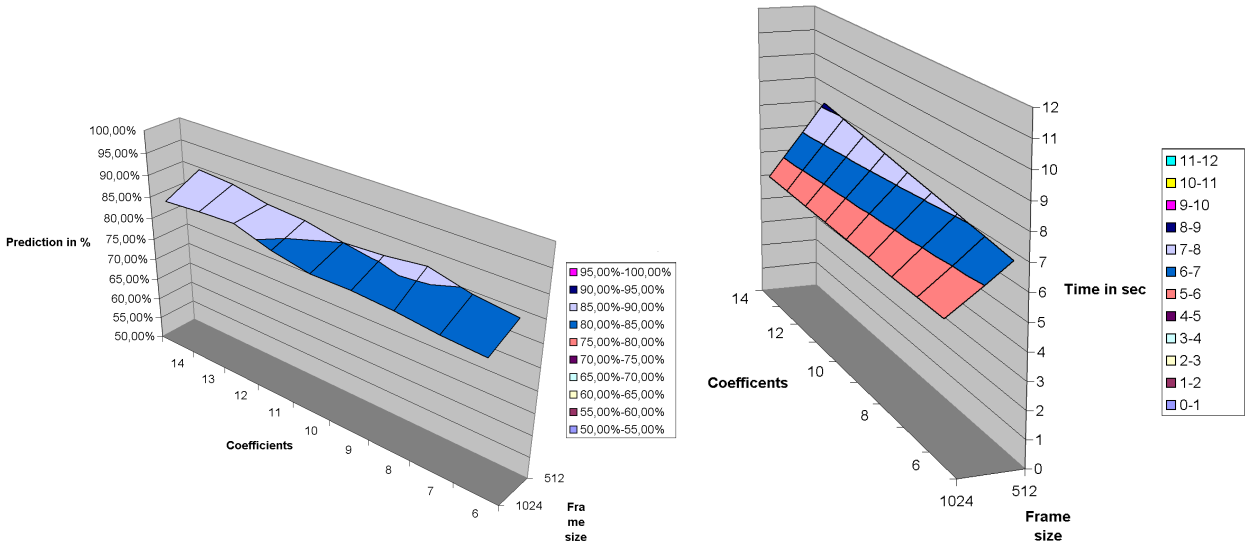


Figure 10.6: Result for the LPC prototype with Semi-independent and 5 templates

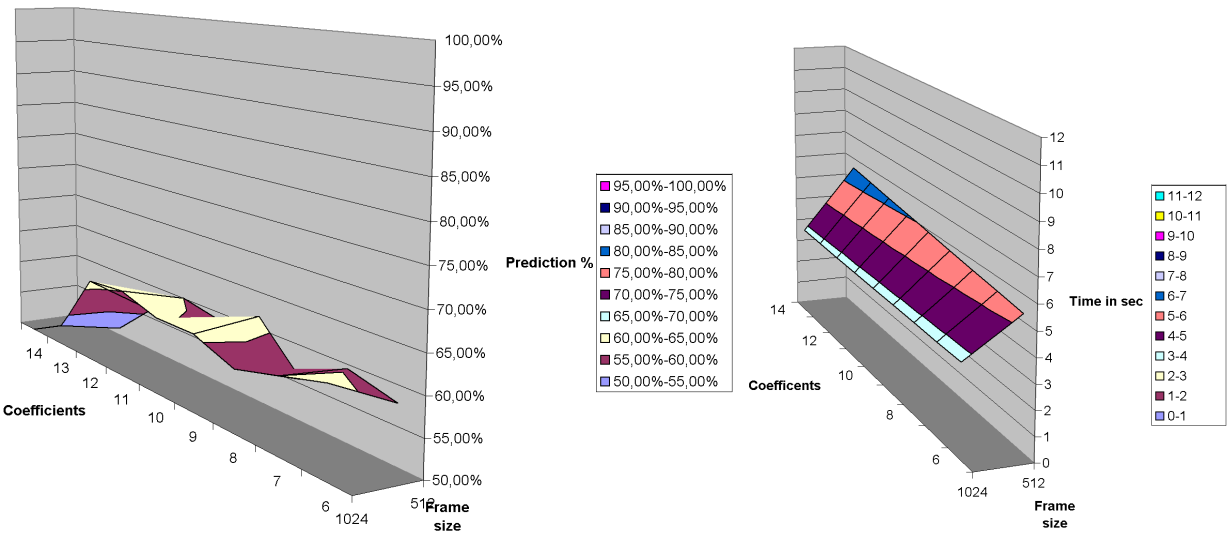


Figure 10.7: Result for the MFCC prototype with Semi-independent and 5 templates

10.4 Evaluation

The results presented in Section 10.3 show the influence of the choice of frame size and number of coefficients.

10.4.1 Average Time To Predict

For each coefficient that is added, the results show that there is a linear connection with the time, like it is expected from the WCET analysis (see Chapter 8). The results presented in the four Figures 10.4, 10.5, 10.6, and 10.4 show that the first prototype with the LPC predictions is slower then the second prototype using MFCC. This is due to to the extra calculations done in the LPC prototype then the MCFF prototype. The results also show that training the Classification Model on more templates also slows down the prediction time.

As it where shown in the WCET analysis in Chapter 8 the execution time for each of the methods used in the different components are linearly. This combine with the knowledge that the two experiments where executed on a 60MHz CPU platform means that if the CPU speed where to be doubled the execution time would be half as long as it is now. For a system that should be used in any given product this means that it is possible to calculate the how fast the CPU needs to be to fulfill a certain demand of response time.

10.4.2 Prediction Score

The prediction score for the two prototypes also differs if we look at the four figures. The LPC prototype generally has a higher prediction score than the MFCC prototype. This again goes for both experiments shown in Section 10.3. For the first described experiments the LPC prototype has a prediction score between 90% and 93% with coefficients between 9 and 12 and a frame size of 512. By comparison, the MFCC prototype has at its highest level between 65% to 77% with coefficients between the same range of coefficients 9 to 12 and a frame size of 1024 at the max prediction score. The last described experiment shows the same picture, although it is a Semi-independent system trained on two users.

10.4.3 Summary

The rest of the results from the experiments are shown in Appendix A.1[4]. The main conclusion for those results is that when using 5 templates as a training set, the prediction score gets higher

[4]The results are shown with the range of values that best showed the conjunctures of the graph.

for both prototypes. We can also see that the LPC prototype outperforms the MFCC in prediction scores, with 95% vs. 75%. The average time to recognize gets higher for both prototypes when more coefficients and templates are used as training sets. When building a Semi-independent speech recognition system the results are similar to the Speaker dependent system, however the prediction scores are a little lower. The average prediction time is higher in the Semi-independent systems than the Speaker dependent as the Classification models are larger.

Analyzing the remaining experiments (can be found in Appendix A), their optimum prediction time and the prediction score are the following:

LPC Prototype Test 1 Frame Size = 512

Number Coefficients = 9

Result 6,3 seconds average prediction time and prediction score 92,2%

Test 2 Frame Size 512

Number Coefficients 14

Result 6,7 seconds average prediction time and prediction score 96,7%

Test 3 Frame Size

Number Coefficients

Test 4 Frame Size 1024

Number Coefficients 23

Result 5,6 seconds average prediction time and prediction score 93,9%

MFCC Prototype Test 1 Frame Size 1024

Number Coefficients 11

Result 3,9 seconds average prediction time and prediction score 77,6%

Test 2 Frame Size 1024

Number Coefficients 10

Result 3,6 seconds average prediction time and prediction score 75,6%

Test 3 Frame Size 1024

Number Coefficients 11

Result 3,9 seconds average prediction time and prediction score 73,7%

Test 4 Frame Size 256

Number Coefficients 5

Result 9,6 seconds average prediction time and prediction score 68,5%

If the all the results from the experiments are compared the conclusion can be that the reason for choosing the MFCC prototype should be that the prediction is faster than the first prototype but the prediction result is not as good as the LPC prototype. The reason for choosing the LPC prototype should be that a high prediction score could be reached, however the average time to predict is slower.

Chapter 11

Economic

The central economic question posed by the development of an SDK is how commercial investments can appropriate value from investments in JopSpeech. Choice of business model, pricing and further development are essential elements in the potential make or break for a commercial launching of JopSpeech.

In this chapter we will be looking at the terms under which a potential further commercial investment in JopSpeech could be conducted.

11.1 Overview

There is a potentially a very large market the creation of a speech recognition SDK for embedded systems, used for classification of signals. The market spans over both mobil-phones, kitchen appliances to biometric systems for human recognition and surveillance. Let us first get an overview of JopSpeech's internal and external factors based upon our current situation[22]:

Strengths	**Weaknesses**
◦ Expertise in Java development and innovative products. ◦ Developed in an Academic environment with its network. ◦ High expertise in speech recognition.	◦ No engineering Skills available among JopSpeech's two developers. ◦ No financial liquidity among the JopSpeech's developers. ◦ There are only two developers on JopSpeech.
Opportunities	**Threats**
◦ The first embedded java speech recognition SDK. ◦ Based on the fastest hardware realization of the JVM available to date[29]. ◦ Demand for an embedded java speech recognition requested by IBM ViaVoice users[1]. ◦ Java is a well supported and recognized programming language with support from the Open Source society.	◦ Only one platform available (JOP). ◦ JOP is not yet Java certified. ◦ IBM might enter the market to respond to the demand.

11.2 Customer Value

JopSpeech consists of both a hardware part, software part (see Chapter 8) and a knowledge approach to speech recognition . The combination offers value for a customer who needs a shorter time to market, or a fast proof of concept in its support activities [14] in the value chain. The technology development phase of any company is based on a set of assumptions about new potentials, and the faster these assumptions or risk factors can be translated into a trustworthy estimate of cost and benefits, the lower the risks. The further into the future a company must look, the greater the uncertainty and higher the risks there are. Shortening time-to-market will reduce these risks to costs and benefits, especially in the area of management (change of management or policy) and

market (change in market or customer conditions) risks.

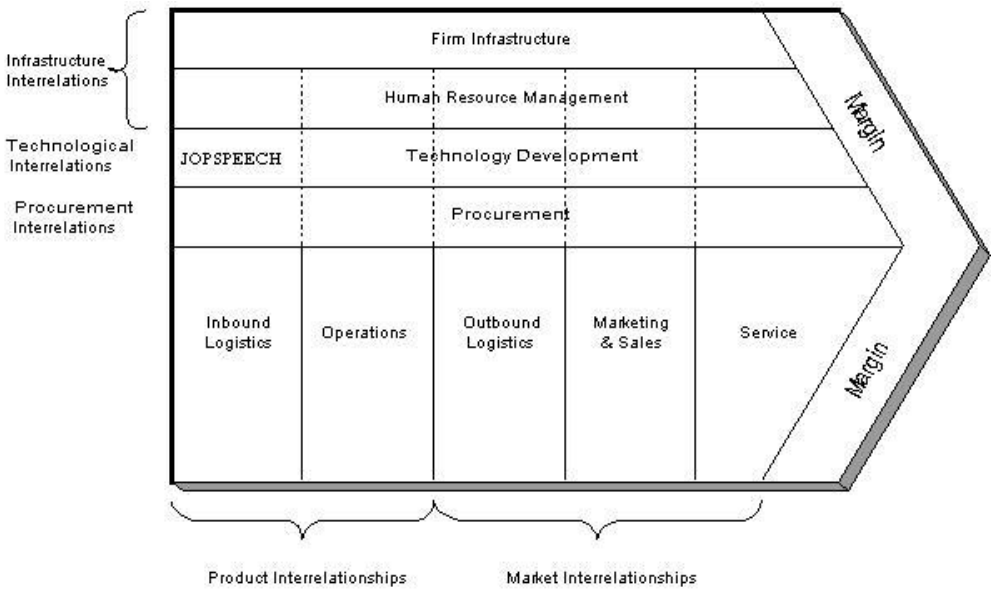


Figure 11.1: JopSpeech placement in the Value chain

The competitive advantage JopSpeech offers customers is valued in risk reduction, considering the benefit estimates with current knowledge and assumptions about future conditions. Consider the customers which have their current business information system running on a Java platform and now wish to make a faster and tighter integration with their production facility. This integration could consist of a shorter pipeline of information to the manager and marketing department in a ubiquitous environment e.g. collecting information from worker by speech. JopSpeech will, in this case, create value for the company in the following areas:

Faster time Lower development time is possible, as JopSpeech already defines an architecture whose development can be constructed.

Competencies Since the company already has their systems running in Java, no new skills are needed for the development phase.

Benefits Faster production planning and lower development costs.

Risk minimizing Faster definition of demands for hardware and lower risk.

Potential value Integration into the company's current production environment enables future development and potential extra services to be integrated into the platform. Extending the company's reporting options throughout their value chain to their customer.

Switching cost Low switching cost involved, as network and language interfaces are known. The only cost involved is in the training of production workers.

11.3 The Production Cost

The current development stage of JopSpeech is on a research or pre-alpha level, thus further development is needed before the SDK can be tested ready for final release. The question is which production models are the most financially beneficially. Let us first look at some of the most viable costs associated with delivering customer value to JopSpeech through the value chain (see Figure 11.1), by looking at the first two primary activities[23]

11.3.1 Inbound Logistics

Let us start by looking at the direct cost associated with buying the hardware (we will take the same as we have constructed our prototype on[2]):

FPGA Altera Cyclone EP1C6 199€.

Network card BASEIO extension 179€.

Extras Byteblaster and power supply 54€.

Speaker A/D and D/A extensions 40€[3].

Licence JOP industrial License 30€.

The total cost associated with hardware and licensing fee is then 502€or 640$. This price might be discounted considerably if the supply chain is optimized, for example partnering with certain electronic manufacturers and JOP manufacturer (*Martin Schoeberl*).

11.3.2 Operations

Some of the biggest initial costs are associated with operations, as the testing and development of the SDK components are vital for a realizing full commercial value for end consumers.

Components The Java classes are costly to produce, but inexpensive to reproduce. The margin cost is close to zero as it only requires a backup of the data and the copy function[32].

[2] `www.jopdesign.com` and Section 10.1
[3] Hardware cost equals 14€. Labor cost equals 26€.

Sunk cost Once the first copy of information is produced, it is sunk and cannot be recovered.

Hardware Multiple copies can be produced roughly at per unit cost.

Production capacity There is no production limit.

It is hard to be exact with the total cost associated with operations, but estimating a cost of 6700€per head/month would give a 250.000€cost of producing the first copy (based on 3 developers and a 12 month development period).

We will not look at the last three activities in the value chain in detail, but still the following should be taken into consideration.

Distribution The cost of distribution is limited to the hardware shipment cost, and the cost of enabling downloads.

Marketing The cost of marketing can be considerable as the product is new.

Documentation The cost of documentation can be considerable as the SDK is not yet finalized.

Sale Service might be considered as an income area of its own as consultancy services could be extended.

Production capacity There is no production limit, as long as the first copy is produced.

11.3.3 Production Model

Summing up the production cost considerations mentioned throughout the previous sections, the following should be noted regarding a product launching of JopSpeech.

Release JopSpeech still needs development before being ready to cover all commercial demands.

Cost The associated cost of producing the first copy is at a minimum, around 250.000€, if all development is done in-house.

Distribution Distribution of the software is essentially cost-free.

Demand Demand for the product is real.

First mover A real first mover advantage is possible since no one else has published an embedded java SDK.

Java Focus can be on the speech recognition SDK, as Java is well supported with free IDE's. For example Eclipse `http://www.eclipse.org/` which has already approved a project from Nokia about extending the eclipse platform to support mobile device Java application development.

Due to the openness of the potential and use of the SDK, it might be an idea to pursue an Open Source production model. An Open Source production model enables the contributors to freely pursue whatever they feel to be most interesting [13], which could only strengthen JopSpeech. Another benefit from the Open Source environment is that it offers expertise in areas that might be directly connected with JopSpeech or just complementary. The concerns that exist in an Open Source model are the intellectual property right and the right to make money out of the contributions. The choice of which licensing form to choose is, as such, important. However studies have shown that although contributors to an Open Source product are not able to make money off the product themselves, this does not limit them from participating in the production [13]. The programmer's benefits from working on an Open Source project is more dependent on the benefits they can get from working on the project, e.g. reputation from doing something, training within a specific area or acknowledgement from peers. Open Source and academia have many parallels; the most obvious relates to motivation. Recognition, career concerns and the desire for peer recognition provide the same powerful inducements in OS as in academia. The recommended licensing and production model would therefore be Open Source with a license form such as Academic Free License (AFL) [4].

11.4 Pricing the Product

In Chapter 4 we mentioned the Embedded Windows CE SDK from Research Lab at 799$ as the bundled product that comes close to JopSpeech . We also know from subsection 11.3.1 that the capacity cost of producing the accompanying hardware is 640$ (not taking any discounting factors into account). Comparing the two sale-prices indicates that a margin of 159$ exists - if we market JopSpeech as a bundle at the same prices as the SDK from Research Lab.

$$799\$ - 640\$ = 159\$$$

We know from theory that a price bundling strategy (either pure or mixed) yields higher revenues than unbundling if conditional reservation prices are asymmetric[38]. Selling the SDK as a bun-

[4] `http://www.opensource.org/licenses/afl-3.0.php`

dled product with more than one component, where the reservation price is asymmetric from the consumers perspective, would yield a higher revenue. We can assume that some people are only interested in the speech recognition library and not interessted in the FPGA board. By bundling the products we whould be able to sell the products as one and thereby get a revenue on all of the products. The bundled product whould consist of an FPGA, a JOP license, a JopSpeech license, a BaseIO board and JopSpeech AD/DA. Because the the company would be small and since the connection between JOP and JopSpeech is so tight the legality in the bundling should not be a problem. Bundling is illegal per se when it involves pure bundling of separate products by a firm with market power and when a substantial amount of commerce is at stake[38], but a small company starting to market JopSpeech should not fall under this category. Open Source is great for the consumers. It lowers the risk of ending in a lock-in strategy, lowers the switching costs and raises the network effects.

11.5 Business Models

JopSpeech is dependent on the two requirements (Java and JOP) such that it is currently limited by their network externalities. Java has already reached critical mass, and the promised development reductions that JopSpeech offers can be realizable. By contrast, JOP is still in its start-up phase and presents as such a challenge for JopSpeech's potential success. A solution to this problem could be a tight partnership / co-operation between JOP and JopSpeech choosing to focus on the network business model, given that the few developers involved and the limited financial liquidity require that the focus be on its core competencies. This could be supplemented by a long-term strategy of buying JOP, thereby supplementing JopSpeech's competencies and market attractiveness. On the other hand although business models with more than one focus area are potential models, focus on the production, distribution and sale of JopSpeech should be the only business areas in the market strategy in the beginning.

To conclude the Chapter and summarize, the recommended overall business model would consist of the following business strategy:

Production The production should be within Open Source and under the Academic Free License

Value propositions The SDK could be bundled in different configurations (see 11.4)

Customer segment The customer segments consists of developers and researchers, both domestic and international - but with focus on being complementary to IBM's ViaVoice product.

Distribution channels The distribution of the software is essentially cost free, done via the internet, and should be done as this way. However the current bundled products should be shipped together with all associated items.

Marketing The marketing of the product should be done through viral commercial and the network within the academia environment and Open Source environment.

Partner network The partner network could be: JOP (Martin Schoeberl), PCB hardware facility, international transport agency (e.g. DHL, UPS), Sphinx project and possibly IBM.

Chapter 12

Evaluation

The solution presented in this master thesis is implemented and tested on the basics of the requirements listed in Chapter 6, an evaluation of the SDK is therefore presented here:

Java *The SDK should be implemented in Java with a minimum of change in Java semantics.*

- The SDK is designed and implemented following the architectural model from the ”Pattern Recognition Approach” proposed by L. Rabiner and B.H. Juang[25] in 1993 as part of its architecture for developing and testing speech recognition systems. We have, in that respect, decided not to follow the Java Speech API (JSAPI) and changed the implementation semantic. We have done this because the JSAPI specifications are not designed for hard real time systems, and therefore it makes sense not to follow the specifications. In order to avoid confusion between semantics, but still seek some conformity with the speech recognition community, we have followed the architectural model by L. Rabiner and B.H. Juang who are the industry gurus. The whole SDK can run on any JVM with the exception of the Audio classes, while Java does not support native calls to hardware the Audio class implementation is JOP specific.

Network *The SDK's architecture must support a network solution.*

- The SDK's components are coupled in a way that makes it possible to use a distributed architecture with the components. This is demonstrated in the two implemented prototypes where part of the *FrontEnd* component runs on a PC and the rest runs on JOP. Furthermore, the MFCC prototype is using a distributed design within the component *Analysis*, where some of the workload is distributed between nodes (PC and JOP), to successfully speed up the process. The network architecture makes it possible to use

more FPGAs with JOP in each of the components, for example two *FrontEnd* component implementations that distribute the work to several *Analysis* components and further to the *Recognizer* component.

Memory *The SDK should handle memory as a limited resource in an embedded system and should therefore be implemented as reusable where possible.*

- All initialization and memory allocation is recommend to be done in the initialization phase using the singleton pattern as the recommended design pattern. No memory allocation should be done during the mission phase, that is why all implemented classes are separated into a mission and initialization structure. All memory allocation is done in the initialization method `Init()`. This design keeps memory usages to a minimum and insures reusability without breaking the object oriented approach of Java. Another benefit of the strict memory usages is the benefit of being able to calculate the exact memory usages helping in the real time predictability and memory requirements.

Real Time *The response time for a system, developed using the SDK on JOP, should be identifiable within a worst case scenario.*

- The WCET analysis has shown that equations can be derived for each class because they are linearly predictable in the real-time system. The experiments also show this linear growth in time as the number of calculations raises, not that this is directly linked to a WCET analysis.

Processing power *The power of the CPU is normally restricted in an embedded system to its minimum. The demands should therefore be identifiable by using the SDK on JOP.*

- The WCET analysis and experiments have shown that demands are identifiable using JopSpeech. The WCET analysis have shown that formulas can be derived from the algorithms, from which the demands to a cpu can be identified.

Fixpoint *As there is no support for fractional arithmetic, the SDK should handle all numeric properties using Integer/Fixpoint arithmetic.*

- The `FixPoint` class has been developed in a way that makes it possible to change the shift value. This makes the SDK more usable for future research using the SDK, because it is not dependent on only one way of splitting an integer value into a fractional number.

Audio in *As there is no hardware support for microphones and loudspeakers for recording and playback on JOP, the SDK should provide a solution for this.*

- We have developed a hardware extension that makes it possible to connect a ordinary PC microphone or a amplified signal for JOP. The hardware extensions are designed to fit the BaseIO board, but all FPGA´s supporting JOP can be used with the hardware extension, it is just a matter of connecting the correct pins.

Application structure *The software implementation of the SDK should primarily follow the application structure defined on JOP Section 3.3.1. By insuring that memory is allocated, threads are created and all non-time critical methods are initialized in the* Initialization *phase.*

- The singleton pattern is used to secure that all initialization is done in the start of the application and there is no initialization in the mission phase. This has the advantages that the application can run ”forever”, without the garbage collector interfering.

Frontend *The SDK should provide options for Sampling and Filtering*

- The *FrontEnd* component is implemented in both hardware and software. This completes the circle of the SDK with both a software part and a hardware part. The *FrontEnd* implemented in software can get data from both the Ethernet and the implemented hardware extension by connecting a microphone.

Analysis *The SDK should provide options for: converting between Time and Frequency domains, encoding sound signals and providing filter bank algorithms.*

- The analysis part of the SDK is developed with several algorithms used for feature extraction of the sound signal, among these are Window functions, Fourier Transform, Linear Predictive Coding, Mel-Frequency Cepstral Coefficients. This enables different experiments by combining the algorithms in different ways. In this thesis we have shown two different implementations,

Recognition *The SDK should provide endpoint detection algorithms, distance measures, dynamic programming and classification models.*

- The SDK provides detection algorithms, distance measures, dynamic programming and classification models. In the prototypes the endpoint detection is used to identify start and end of the signals. The dynamic programming is done via the DTW algorithm that uses the cepstral distance measure to calculate the distance between templates.

Classification models and prediction algorithms are done by a 1-Nearest Neighbour algorithm and Model is built via threshold values calculated through the *training set* distances measures.

Chapter 13

Conclusions

This study has demonstrated and developed an architectural design for implementing speech recognition on embedded systems with the programming language Java. Experiments and analysis have shown that there is a linearity in the implemented software components that enables a researcher or developer to calculate the worst case execution time. Experiments have also shown that the linear time predictability holds true for systems developed on JOP. The implemented SDK consists of both hardware and software parts. The hardware extension enables amplification, recording and playback of audio through an AD/DA converter. The software components enable the frontend Sampling, Signal Analysis and Speech Recognition. The software components can be used to build both Classification models and recognize unknown signals.

Results of the study have shown that simple systems build with JopSpeech SDK can obtain a 96,7% prediction score with a prediction time of 6,7 seconds, for a speaker depended system with a limited vocabulary. The prediction time of 6,7 seconds was obtained using a FPGA chip clocked with a 60MHz CPU. Results of the study have also shown that JopSpeech SDK can be used in a networked solution distributing work among processors. Furthermore experiments have shown that the application structure suggested in JopSpeech insures ideally implementation of speech recognition systems on embedded platforms.

We can therefore conclude that the JopSpeech SDK developed through this study meets all the identified requirements, and is the first embedded Java speech recognition SDK developed for JOP. JopSpeech is also the first embedded Java speech recognition that consists of a full circle approach, defining a set of software components, interfaces and hardware for audio in and out, that allows developers and researchers to take advantage of speech technology in the embedded Java structure of JOP.

Chapter 14

Future Work

In the future it could be interesting to include or investigate some of the following areas.

Statistical Recognition Library The recognition part of the API has two different algorithms implemented. In future work more distance functions could be implemented as well as prediction algorithms as for example, the popular Hidden Markov Model that builds upon Bayesian statistics.

Standardize The Voice Libraries This thesis has taken the pattern recognition approach and the feature extraction was chosen to be on the level of a word. Another approach for future work could be to take the phonetic linguistic approach and implement this in an embedded Java environment.

Graphical User-Interface For making the API even more user-friendly it would be nice to develop a graphic user interface in for example Eclipse. This interface could for example be a workflow application where it would be possible to drag-and-drop different classes into the three different components. Furthermore it could be able to decide where the different components or classes should run. This would give the user that does not know much about speech recognition a possibility to develop an application easily.

Test Within the JopSpeech API, we have conducted a test of two differently implemented prototypes. For future work other word databases could be used for testing in order to compare the implementation with other speech recognition applications.

Bibliography

[1] *WCET Analysis for a Java Processor*, 2006.

[2] D.P. Kemerait R.C. Childers, D.G. Skinner. The cepstrum: A guide to processing. *Proceedings of the IEEE*, 10:1428 – 1443, 1977.

[3] John Coleman. *Introducing Speech and Language Processing*. Cambridge University Press, 2005.

[4] Cay S. HorstMann Gary Cornell. *Core JAVA 2 Volume I-Fundamentals*, volume 1. SUN Microsystems, 2001.

[5] National Semiconductor Corporation. *LMV321/LMV358/LMV324 Single/Dual/Quad General Purpose, Low Voltage, Rail-to-Rail Output Operational Amplifiers*, 2006.

[6] Bruce A. Dautrich, Lawrence R. Rabiner, and Thomas B. Martin. On the effects of varying filter bank parameters on isolated word recognition. *IEEE Transactions On Acoustics, Speech, And Signal Processing*, 4(4):793–807, August 1983.

[7] Steven B. Davis and Paul Mermelsteinl. Comparison of parametric representations for monosyllabic word recognition in continuously spoken sentences. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 28:357 – 366, 1980.

[8] P. Duhamel and M. Vetterli. Fast fourier transforms: a tutorial review and a state of the art. *Signal Process*, 19(4):259–299, 1990.

[9] Ian H. Witten & Eibe Frank. *Data Mining Pratical Machine learning Tools and Techniques*. Morgan Kaufmann, 2005.

[10] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design patterns: elements of reusable object-oriented software*. Addison-Wesley Professional, 1995.

[11] Erik Hüche. *Digital signalbehandling*, volume 5. Teknisk Forlag A/s, 1 edition, 1996.

[12] Hermann J. Helgert. Pulse modulation. McGraw-Hill Encyclopedia of Science and Technology., July 2002. AccessScience@McGraw-Hill.

[13] L. Josh and J. Tirole. The economics of technology sharing: Open source and beyond. *Journal of economics perspectives*, 19:99–120, 2005.

[14] Phillip Kotler. *Marketing Management*. Prentice Hall, 11 edition, 2003.

[15] Peter Ladefoged. *Elements of Acoustic Phonetics*. University Of Chicago Press, 1995.

[16] B. Lilly and K. Paliwal. Effect of speech coders on speech recognition performance. In *Proc. ICSLP '96*, volume 4, pages 2344–2347, Philadelphia, PA, 1996.

[17] Y. Linde, A. Buzo, and R. Gray. An algorithm for vector quantizer design. *IEEE Trans. on Communications*, 1:84–95, Jan. 1980.

[18] Michael J. A. Berry Gordon Linoff. *Mastering Data mining*. WILEY, 2000.

[19] Philip V. Loprest. Sigma-delta converter. McGraw-Hill Encyclopedia of Science and Technology., July 2002. AccessScience@McGraw-Hill.

[20] Richard G. Lyons. *Understanding Digital Signal Processing*. Prentice Hall, 2004.

[21] Sheila E. Blumstein Philip Lieberman. *Speech Physiology, Speech Perception, and Acoustic Phonetics*. Cambridge University Press, 1988.

[22] Michael E. Porter. How competitive forces shape strategy. *Harvard Business Review*, 57:137–145, 1979.

[23] Michael E. Porter. What is strategy? *Harvard Business Review*, 74:61–80, 1996.

[24] William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery. *Numerical Recipes in C: The Art of Scientific Computing*. Cambridge University Press, New York, NY, USA, 1992.

[25] Lawrence R. Rabiner and Biing-Hwang Juang. *Fundamentals Of Speech Recognition*. Printice Hall, 1993.

[26] Robert Rozman and Dusan M. Kodek. Improving speech recognition robustness using nonstandard windows. *Eurocon03*, 2003.

[27] Ruiz Ruiz and Jorge R. Manjarrez Sanchez. Design of a voicexml gateway. *enc*, 00:49, 2003.

[28] Martin Schoeberl. Jop: a real-time java processor. www.jopdesign.com.

[29] Martin Schoeberl. *JOP: A Java Optimized Processor for Embedded Real-Time Systems*. PhD thesis, Vienna University of Technology, 2005.

[30] National Semiconductor. *Op Amp Circuit Collection*, February 1978. Application Note 31.

[31] Philips Semiconductors. *TDA7050T Low voltage mono/stereo power amplifier*. Philips, Juli 1994.

[32] Carl Shapiro and Hal R. Varian. *Information Rules: A Strategic Guide to the Network Economy*. Harvard Business School Press, Harvard, Massachussets, November 1998.

[33] S. Sigurdsson, K. B. Petersen, and T. Lehn-Schiøler. Mel frequency cepstral coefficients: An evaluation of robustness of mp3 encoded music. In *Proceedings of the Seventh International Conference on Music Information Retrieval (ISMIR)*, 2006.

[34] David E. Simon. *An Embedded Software Primer*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1999.

[35] Mark D. Skowronski and John G. Harris. Improving the filter bank of a classic speech feature extraction algorithm. *International Symposium on Circuits and Systems*, 4:281–284, 2003.

[36] Steven W. Smith. *The Scientist and Engineer's Guide to Digital Signal Processing*. California Technical Publishing, 1997.

[37] Volkman Stevens and Newman. Melody scale. *The Journal of the Acoustical Society of America nr 8*, 8:185–190, 1937.

[38] Gerard J. Stremersch, Stefan Tellis. Strategic bundling of products and prices: A new synthesis for marketing. *Journal of Marketing*, 66:55–72, Jan2002.

[39] Willie Walker, Paul Lamere, Philip Kwok, Bhiksha Raj, Rita Singh, Evandro Gouvea, Peter Wolf, and Joe Woelfel. Sphinx-4: A flexible open source framework for speech recognition. Technical report, Sun Microsystems, 2004.

[40] Hsiao-Wuen Hon Xuedong Huang, Alex Acero. *Spoken Language Processing, A Guide to Theory, Algorithm, and System Development*. Prentice Hall, 2001.

[41] Bo XU ZHU Shaohui, Wenju LIU. Comparison between the spectral estimation techniques by different spectral-distortion measures. *ISCSLP Anthology*, 2002.

Appendix A

Appendix - Experiments

A.1 Results

A.1.1 Tables

Frame size	256		512		1024	
Coefficient	Avg. time	Prediction %	Avg. time	Prediction %	Avg. time	Prediction %
4	7,93	90,00%	5,78	86,67%	4,83	86,67%
5	8,32	90,00%	5,85	88,89%	4,84	85,56%
6	8,7	91,11%	5,95	92,22%	4,87	86,67%
7	9,09	92,22%	6,04	93,33%	4,89	84,44%
8	9,46	92,22%	6,13	93,33%	4,92	84,44%
9	9,86	93,33%	6,24	93,33%	4,91	85,56%
10	10,24	94,44%	6,34	93,33%	4,97	87,78%
11	10,64	94,44%	6,44	94,44%	4,97	87,78%
12			6,54	94,44%	5	90,00%
13			6,64	95,56%	5,01	85,56%
14			6,74	96,67%	5,04	88,89%
15			6,85	95,56%	5,05	85,56%
16			6,96	94,44%	5,08	86,67%
17			7,06	94,44%	5,1	87,78%
18			7,17	94,44%	5,12	88,89%
19			7,27	95,56%	5,14	87,78%
20			7,38	95,56%	5,17	87,78%
21			7,48	95,56%	5,21	85,56%
22			7,59	95,56%	5,22	84,44%
23			7,7	95,56%	5,25	86,67%
24			7,79	96,67%	5,27	84,44%
25			7,91	96,67%	5,3	85,56%
26			8,02	94,44%	5,31	83,33%
27			8,12	94,44%	5,35	83,33%
28			8,22	94,44%	5,38	82,22%
29			8,34	96,67%	5,41	85,56%

Table A.1: Result for the first prototype with a single user and 5 templates

Frame size	256		512		1024	
Coefficient	Avg. time	Prediction %	Avg. time	Prediction %	Avg. time	Prediction %
3	8,31	84,48%	6,15	84,05%	5,18	82,76%
4	8,75	87,50%	6,24	85,34%	5,21	86,64%
5	9,19	89,22%	6,34	88,79%	5,23	87,07%
6	9,64	88,79%	6,45	90,09%	5,26	90,95%
7	10,09	89,66%	6,55	90,95%	5,26	89,66%
8	10,53	90,52%	6,67	90,95%	5,29	89,22%
9	10,98	90,52%	6,78	92,24%	5,3	92,24%
10			6,89	91,81%	5,33	91,81%
11			7,01	93,10%	5,34	91,81%
12			7,12	93,53%	5,35	92,24%
13			7,24	92,24%	5,37	92,67%
14			7,35	93,97%	5,4	92,24%
15			7,47	92,24%	5,41	93,10%
16			7,58	92,24%	5,43	92,67%
17			7,69	93,10%	5,46	92,24%
18			7,81	93,10%	5,49	93,10%
19			7,93	93,10%	5,51	92,24%
20			8,04	93,53%	5,53	91,81%
21			8,16	93,10%	5,55	92,67%
22			8,27	92,24%	5,58	92,24%
23			8,39	92,67%	5,61	93,97%
24			8,51	91,81%	5,63	91,81%

Table A.2: Result for the first prototype with two users and 3 templates

Frame size	256		512		1024	
Coefficient	Avg. time	Prediction %	Avg. time	Prediction %	Avg. time	Prediction %
3	9,61	78,33%	6,19	72,78%	4,95	69,44%
4	10,34	80,00%	6,36	76,67%	4,97	78,33%
5	11,07	82,22%	6,54	80,00%	4,99	78,33%
6			6,72	83,33%	5,02	80,00%
7			6,89	84,44%	5,04	80,56%
8			7,07	86,67%	5,06	81,67%
9			7,25	85,56%	5,1	82,22%
10			7,43	85,00%	5,11	82,22%
11			7,61	86,67%	5,15	83,89%
12			7,79	87,22%	5,17	86,67%
13			7,97	88,89%	5,21	86,11%
14			8,14	89,44%	5,24	85,56%

Table A.3: Result for the first prototype with a two users and 5 templates

Frame size	256		512		1024	
Coefficient	Avg. time	Prediction %	Avg. time	Prediction %	Avg. time	Prediction %
3	6,20	58,89%	4,37	60,00%	3,60	53,33%
4	6,56	60,00%	4,41	50,00%	3,59	54,44%
5	6,92	67,78%	4,47	58,89%	3,61	61,11%
6	7,28	65,56%	4,53	55,56%	3,64	63,33%
7	7,64	64,44%	4,61	67,78%	3,62	64,44%
8	8,00	52,22%	4,69	61,11%	3,61	64,44%
9	8,37	67,78%	4,77	65,56%	3,64	72,22%
10	8,73	65,56%	4,85	63,33%	3,64	75,56%
11	9,10	67,78%	4,93	68,89%	3,62	72,22%
12			5,02	66,67%	3,63	74,44%
13			5,10	52,22%	3,64	60,00%
14			5,18	35,56%	3,63	16,67%
15			5,26	23,33%	3,63	10,00%
16			5,35	60,00%	3,64	30,00%
17			5,43	62,22%	3,65	18,89%
18			5,51	5,56%	3,66	5,56%
19			5,59	14,44%	3,65	12,22%
20			5,67	7,78%	3,65	6,67%
21			5,76	10,00%	3,67	10,00%

Table A.4: Result for the second prototype with a single user and 5 templates

A.1.2 Figures

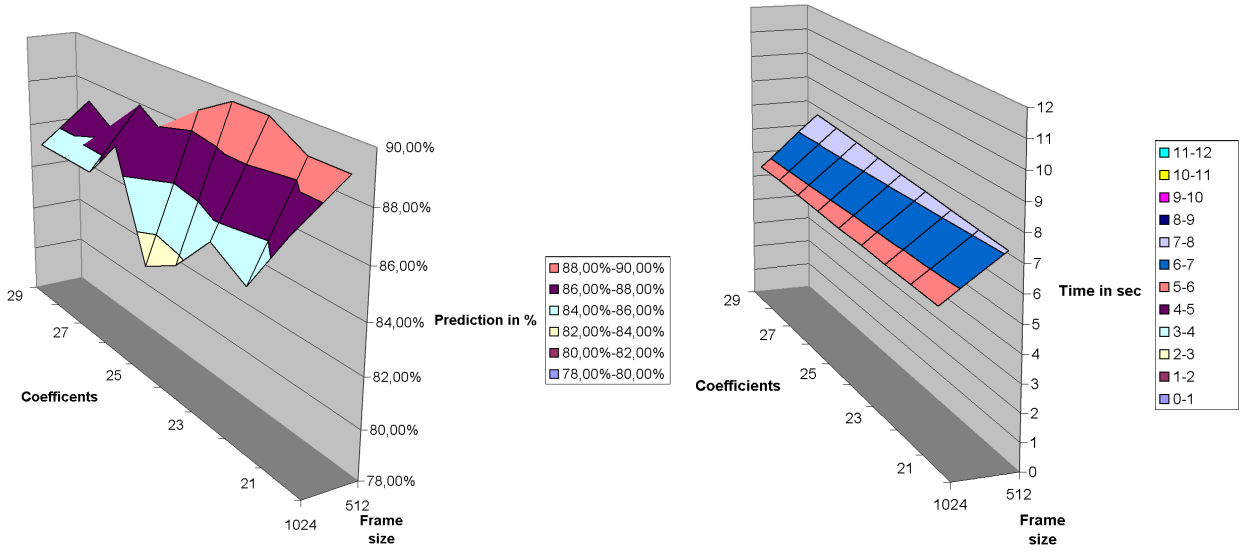


Figure A.1: Result for the first prototype with a single user and 3 templates

Frame size	256		512		1024	
Coefficient	Avg. time	Prediction %	Avg. time	Prediction %	Avg. time	Prediction %
3	6,89	64,66%	4,74	56,90%	3,89	56,47%
4	7,32	63,79%	4,81	59,91%	3,91	57,33%
5	7,74	68,10%	4,89	65,52%	3,89	65,09%
6	8,16	64,66%	4,98	59,48%	3,92	65,52%
7	8,59	70,69%	5,07	67,24%	3,92	67,67%
8	9,01	59,05%	5,16	58,19%	3,92	68,10%
9	9,43	68,53%	5,25	67,24%	3,92	65,52%
10			5,35	64,66%	3,93	67,67%
11			5,45	66,81%	3,94	73,71%
12			5,54	65,09%	3,94	65,95%
13			5,64	63,79%	3,94	62,07%
14			5,73	38,36%	3,94	17,24%
15			5,83	24,57%	3,95	12,07%
16			5,92	64,22%	3,95	21,98%
17			6,02	60,34%	3,97	20,69%
18			6,12	7,76%	3,96	12,07%
19			6,21	7,76%	3,97	9,05%
20			6,31	7,33%	3,99	9,05%
21			6,41	9,05%	4	10,34%
22			6,51	6,47%	4	9,91%
23			6,6	9,91%	4,01	7,33%
24			6,7	8,19%	4,03	8,62%

Table A.5: Result for the second prototype with two users and 3 templates

Frame size	256		512		1024	
Coefficient	Avg. time	Prediction %	Avg. time	Prediction %	Avg. time	Prediction %
3	8,24	65,00%	4,83	53,89%	3,73	54,44%
4	8,95	60,56%	4,98	56,11%	3,74	52,78%
5	9,65	68,33%	5,14	61,67%	3,72	62,22%
			5,29	57,78%	3,73	61,67%
			5,45	59,44%	3,74	60,00%
			5,60	55,56%	3,75	58,33%
			5,75	61,67%	3,75	60,56%
			5,92	58,89%	3,76	60,56%
			6,07	60,00%	3,77	62,22%
			6,23	56,11%	3,78	61,67%
			6,38	51,11%	3,79	52,78%
			6,54	32,22%	3,81	13,89%

Table A.6: Result for the second prototype with a two users and 5 templates

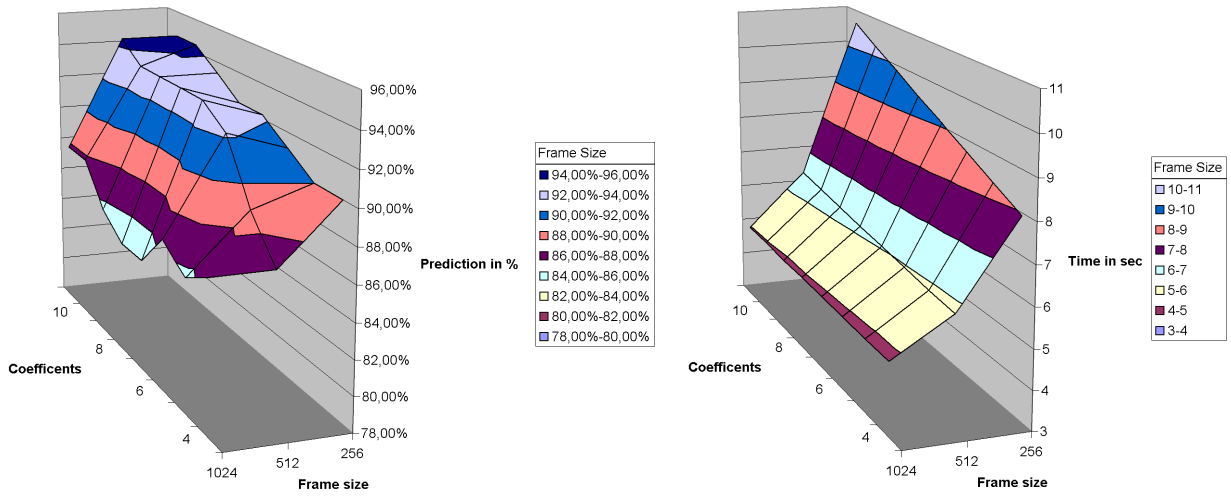


Figure A.2: Result for the first prototype with a single user and 5 templates

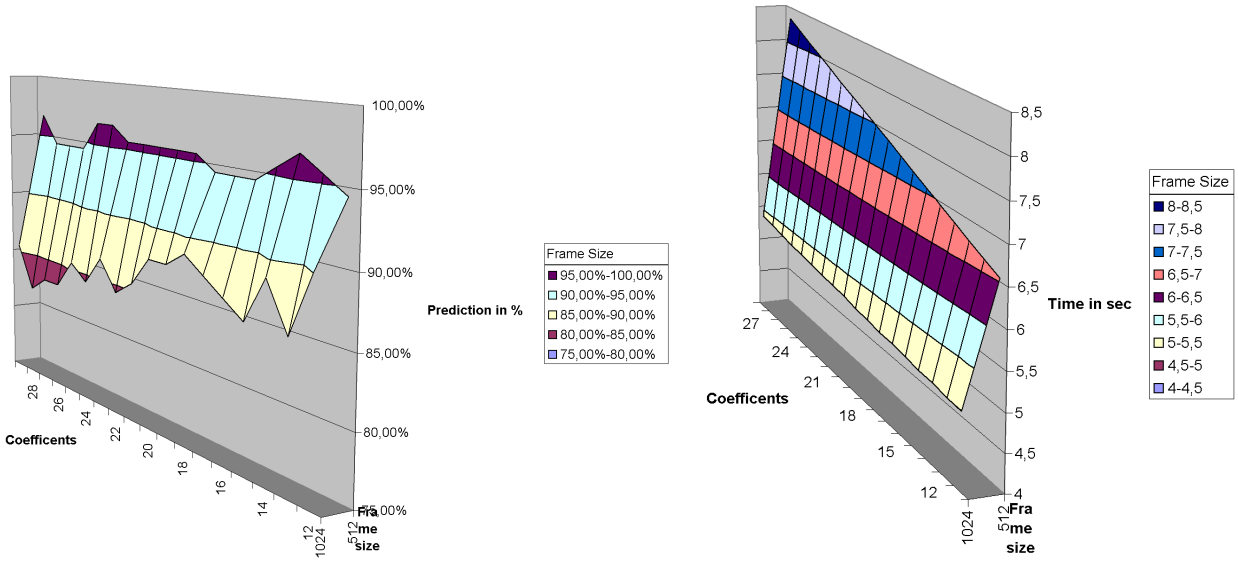


Figure A.3: Result for the first prototype with a single user and 5 templates

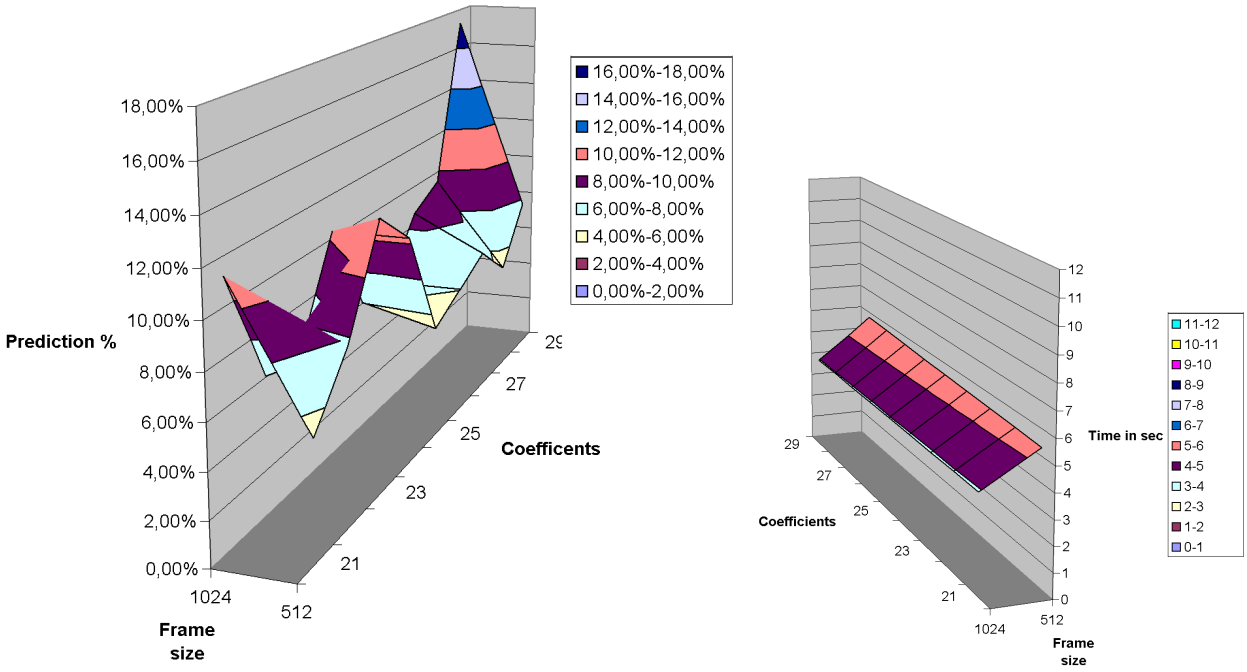


Figure A.4: Result for the second prototype with one user and 3 templates

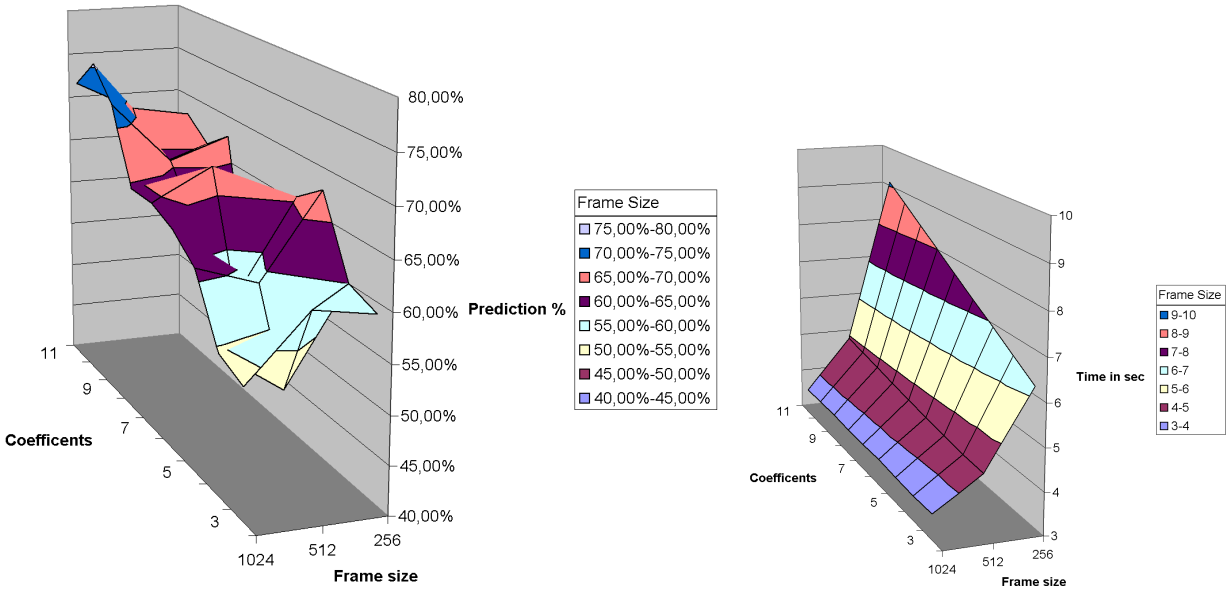


Figure A.5: Result for the second prototype with one user and 5 templates

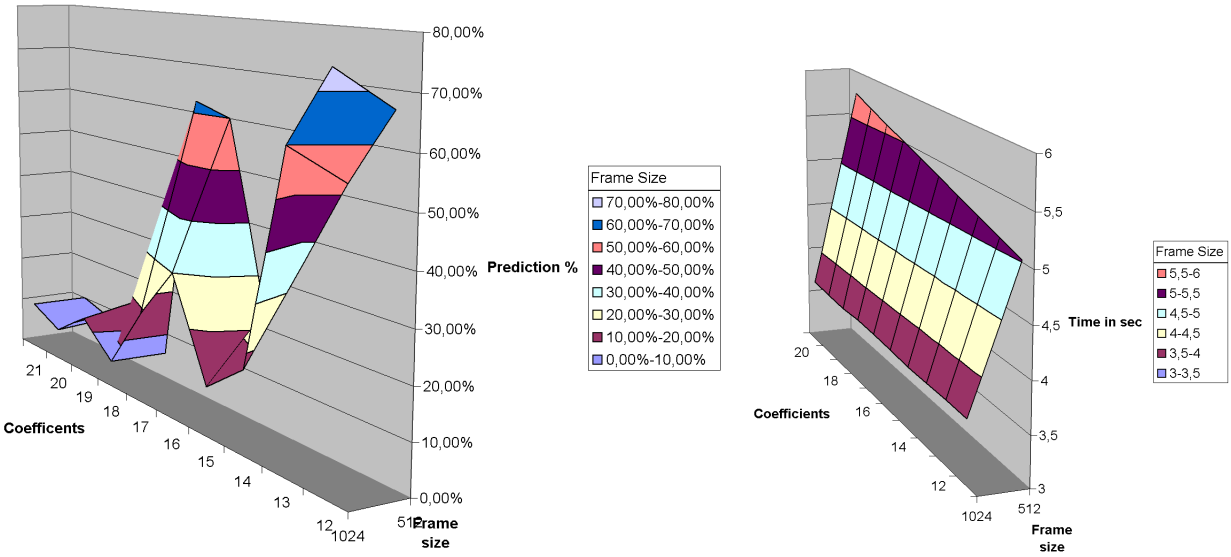


Figure A.6: Result for the second prototype with one user and 5 templates

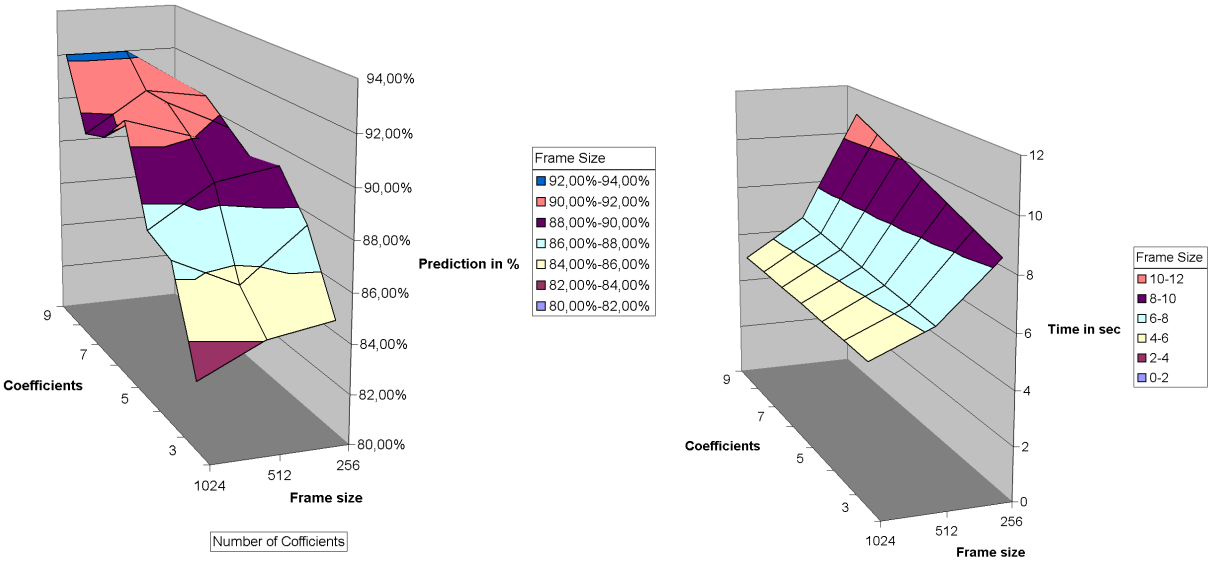


Figure A.7: Result for the first prototype with two user and 3 templates

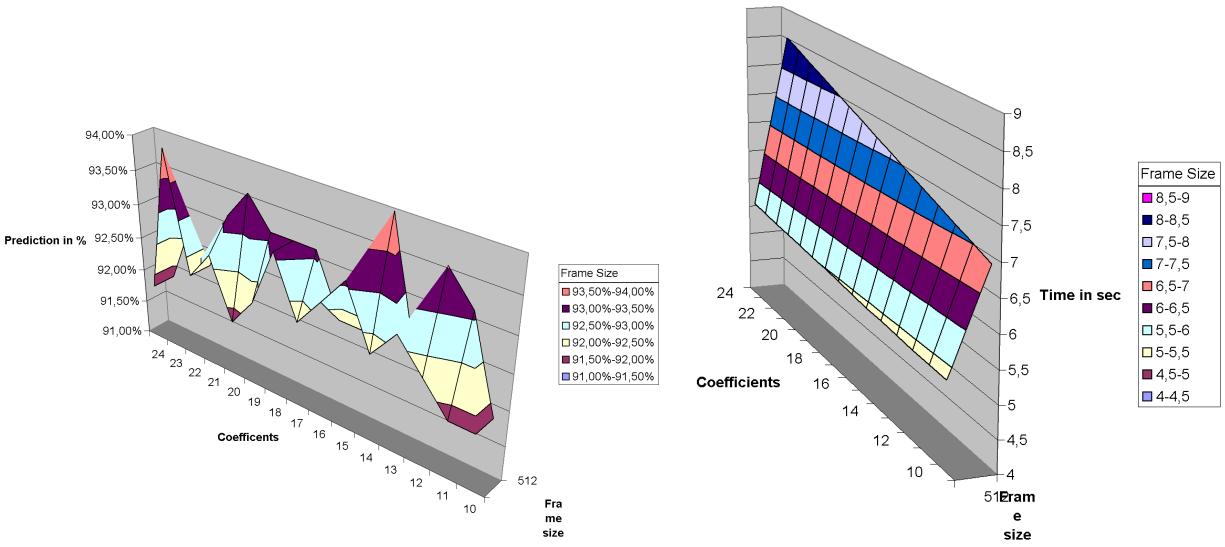


Figure A.8: Result for the first prototype with two user and 3 templates

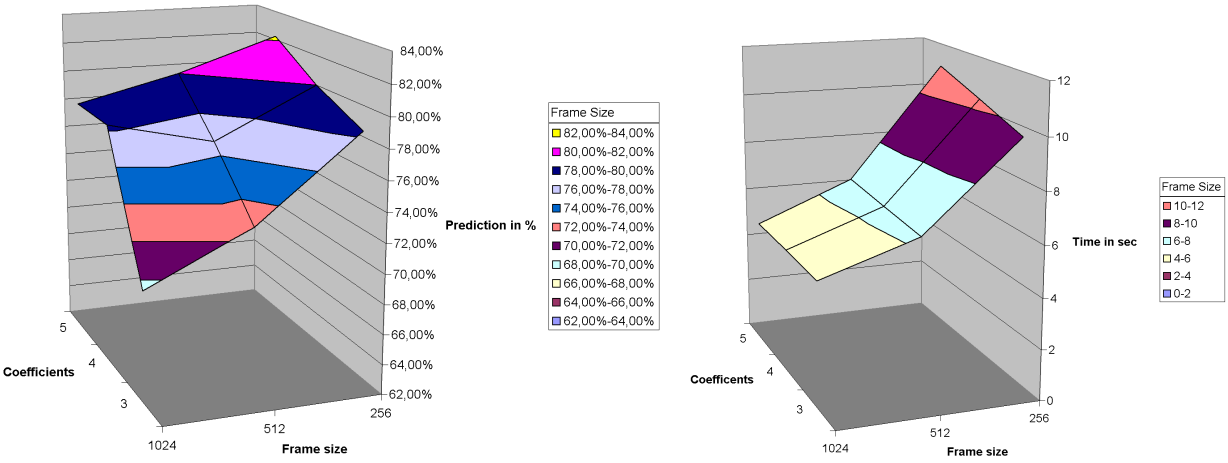


Figure A.9: Result for the first prototype with two user and 5 templates

Figure A.10: Result for the second prototype with one user and 3 templates

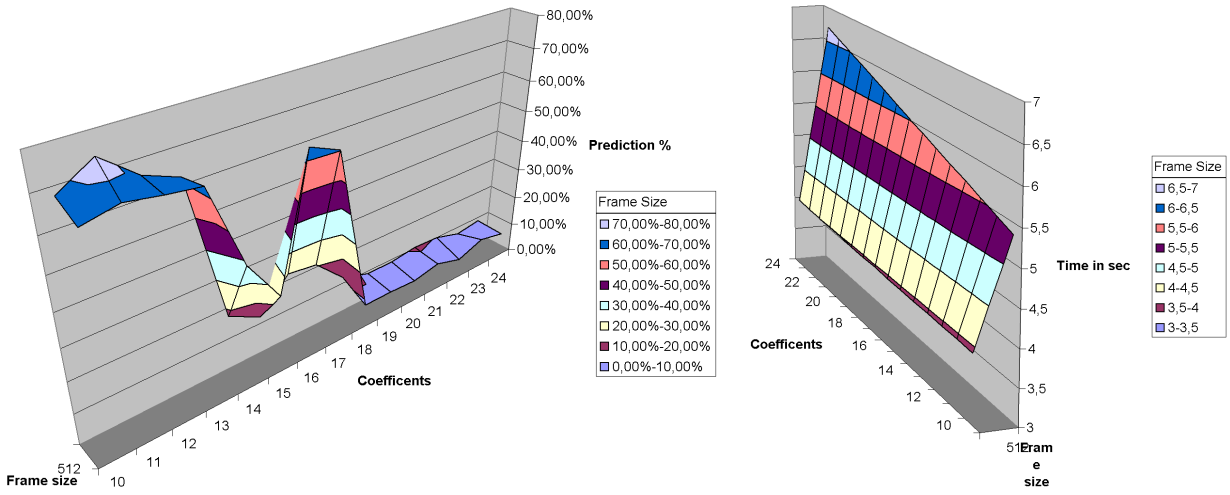


Figure A.11: Result for the second prototype with one user and 5 templates

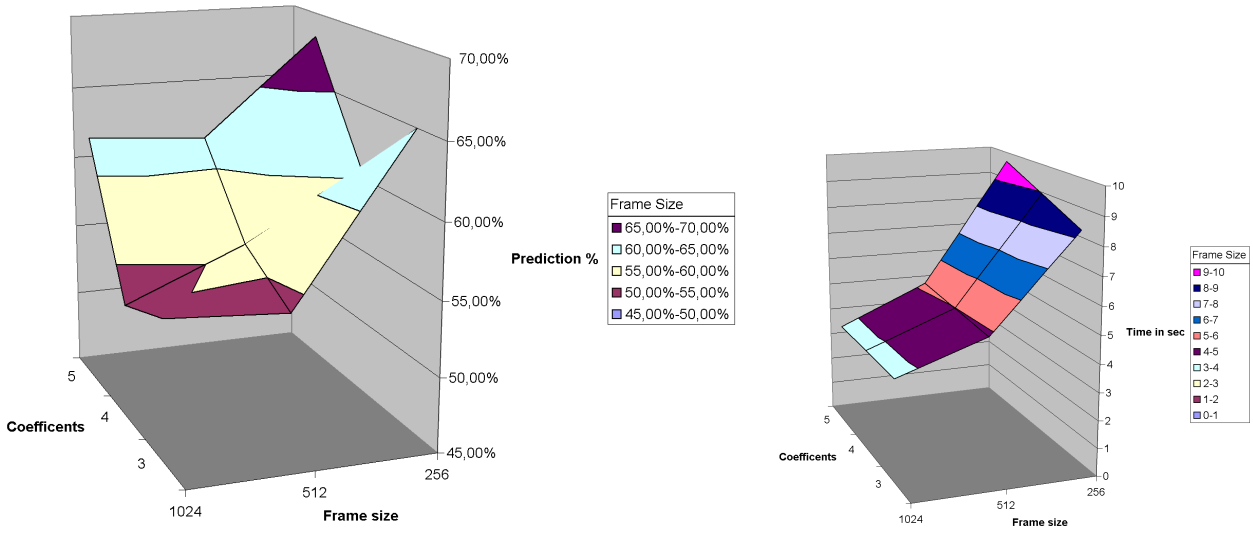
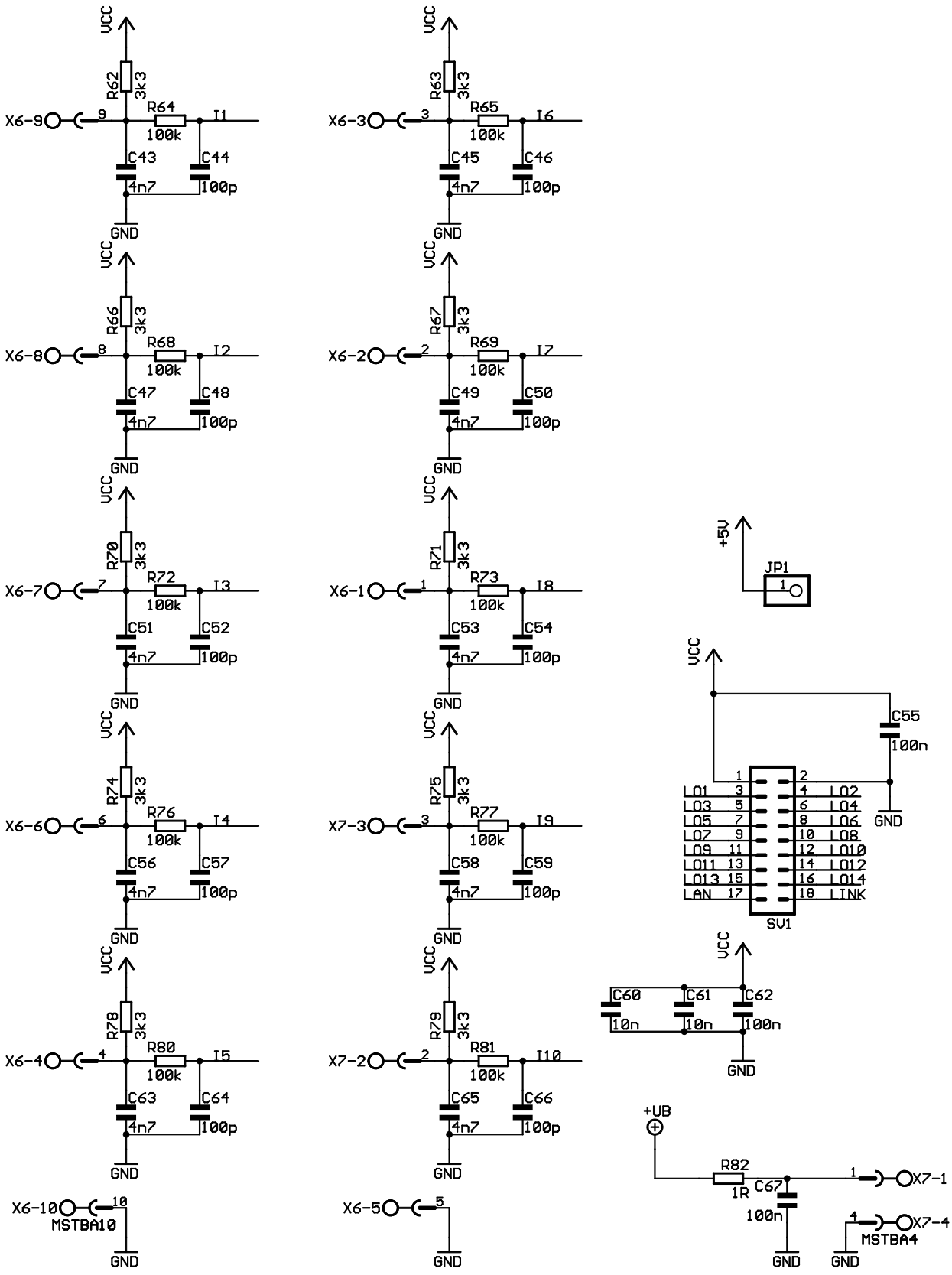


Figure A.12: Result for the second prototype with one user and 5 templates

Appendix B

Appendix - Implementation

B.1 Connections on JOP



Date: 25.01.2004 20:04:42	Ver.: 2004/01
TITLE: baseio	Sheet: 4/4

The rest of the BaseIO schematics can be found at `http://www.jopdesign.com/board.jspbaseio`

B.2 Worst Case Execution Time

The worst case execution time is done by measuring the cycles used in a method. The tool used is WCET from cvs of JOP this paper ”WCET Analysis for a Java Processor”[1] describe the WCET.

B.2.1 FixPoint

mul(int,int,int)

```
************APPLICATION WCET=226*******************
/***WCET calculation source***/
/* WCA WCET objective: util.FP.Mul */
max: tM58_S_E1 tM58_B1_E1 tM58_T_E1;
/* WCA flow constraints: Mul : M58 */
M58_S_E1: 1 = fM58_S_E1_M58_B1_E1; // S flow
M58_B1_E1: fM58_S_E1_M58_B1_E1 + fM58_B1_E1_M58_B1_E1 = fM58_B1_E1_M58_T_E1;
M58_T_E1: fM58_B1_E1_M58_T_E1 = 1; // T flow
/* WCA flow to cycle count */
tM58_B1_E1 = 226 fM58_S_E1_M58_B1_E1 + 226 fM58_B1_E1_M58_B1_E1;
/* Invocation(s) from Mul:[M58] */

Value of objective function: 226

Actual values of the variables:
tM58_S_E1                       0
tM58_B1_E1                    226
tM58_T_E1                       0
fM58_S_E1_M58_B1_E1             1
fM58_B1_E1_M58_B1_E1            0
fM58_B1_E1_M58_T_E1             1

************END APPLICATION WCET******************
********************************************************************************
WCET info for:util.FP.Mul(III)I

Directed graph of basic blocks(row->column):
============================
    M58_S_E0M58_B1_E0M58_T_E0
M58_S_E0 .   1   .
M58_B1_E0 .   .   1
M58_T_E0 .   .   .
============================

Table of basic blocks' and instructions
================================================================================
Block Addr.  Bytecode                Cycles    Cache miss     Misc. info
             [opcode]                        invoke  return
--------------------------------------------------------------------------------
M58_S_E0'S'
  Src. line 112: int res = 0;
M58_B1_E0'B'{}<-[M58_S_E0 M58_B1_E0]0:     iconst_0[3]                  1

      1:     istore_3[62]                1                    ->I:res
  Src. line 113: res = ((f1 >> shift) * (f2 >> shift)) << shift; // AH*BH
      2:     iload_0[26]                 1                    I:f1
      3:     iload_2[28]                 1                    I:shift
      4:     ishr[122]                   1
      5:     iload_1[27]                 1                    I:f2
      6:     iload_2[28]                 1                    I:shift
      7:     ishr[122]                   1
      8:     imul[104]                  35
      9:     iload_2[28]                 1                    I:shift
      10:    ishl[120]                   1
      11:    istore_3[62]                1                    ->I:res
  Src. line 114: res += ((f1 >> shift) * (f2 - ((f2 >> shift) << shift))); // AH*BL
      12:    iload_3[29]                 1                    I:res
      13:    iload_0[26]                 1                    I:f1
      14:    iload_2[28]                 1                    I:shift
```

```
      15:    ishr[122]                    1
      16:    iload_1[27]                  1                    I:f2
      17:    iload_1[27]                  1                    I:f2
      18:    iload_2[28]                  1                    I:shift
      19:    ishr[122]                    1
      20:    iload_2[28]                  1                    I:shift
      21:    ishl[120]                    1
      22:    isub[100]                    1
      23:    imul[104]                   35
      24:    iadd[96]                     1
      25:    istore_3[62]                 1                    ->I:res
  Src. line 115: res += ((f1 - ((f1 >> shift) << shift)) * (f2 >> shift)); // AL*BH
      26:    iload_3[29]                  1                    I:res
      27:    iload_0[26]                  1                    I:f1
      28:    iload_0[26]                  1                    I:f1
      29:    iload_2[28]                  1                    I:shift
      30:    ishr[122]                    1
      31:    iload_2[28]                  1                    I:shift
      32:    ishl[120]                    1
      33:    isub[100]                    1
      34:    iload_1[27]                  1                    I:f2
      35:    iload_2[28]                  1                    I:shift
      36:    ishr[122]                    1
      37:    imul[104]                   35
      38:    iadd[96]                     1
      39:    istore_3[62]                 1                    ->I:res
  Src. line 116: res += (((f1 - ((f1 >> shift) << shift)) >> 1)
      40:    iload_3[29]                  1                    I:res
      41:    iload_0[26]                  1                    I:f1
      42:    iload_0[26]                  1                    I:f1
      43:    iload_2[28]                  1                    I:shift
      44:    ishr[122]                    1
      45:    iload_2[28]                  1                    I:shift
      46:    ishl[120]                    1
      47:    isub[100]                    1
      48:    iconst_1[4]                  1
      49:    ishr[122]                    1
      50:    iload_1[27]                  1                    I:f2
      51:    iload_1[27]                  1                    I:f2
      52:    iload_2[28]                  1                    I:shift
      53:    ishr[122]                    1
      54:    iload_2[28]                  1                    I:shift
      55:    ishl[120]                    1
      56:    isub[100]                    1
      57:    imul[104]                   35
      58:    iconst_1[4]                  1
      59:    ishr[122]                    1
      60:    iload_2[28]                  1                    I:shift
      61:    iconst_2[5]                  1
      62:    isub[100]                    1
      63:    ishr[122]                    1
      64:    iadd[96]                     1
      65:    istore_3[62]                 1                    ->I:res
  Src. line 119: return res;
      66:    iload_3[29]                  1                    I:res
      67:    ireturn[172]                23                    sum(B1):    226
M58_T_E0'T'<-[M58_B1_E0]
=========================================================================
Info: n=17 a=-1 r=1 w=1
```

```
/***WCET calculation source***/
/* WCA WCET objective: util.FP.Mul */
max: tM58_S_E0 tM58_B1_E0 tM58_T_E0;
/* WCA flow constraints: Mul : M58 */
M58_S_E0: 1 = fM58_S_E0_M58_B1_E0; // S flow
M58_B1_E0: fM58_S_E0_M58_B1_E0 + fM58_B1_E0_M58_B1_E0 = fM58_B1_E0_M58_T_E0;
M58_T_E0: fM58_B1_E0_M58_T_E0 = 1; // T flow
/* WCA flow to cycle count */
tM58_B1_E0 = 226 fM58_S_E0_M58_B1_E0 + 226 fM58_B1_E0_M58_B1_E0;
/* Invocation(s) from Mul:[M58] */

Value of objective function: 226

Actual values of the variables:
tM58_S_E0                       0
tM58_B1_E0                    226
tM58_T_E0                       0
fM58_S_E0_M58_B1_E0             1
fM58_B1_E0_M58_B1_E0            0
fM58_B1_E0_M58_T_E0             1

/*util.FP.Mul(III)I*/
digraph G {
size = "10,7.5"
      M58_S_E0->M58_B1_E0 [label="fM58_S_E0_M58_B1_E0=1"];
      M58_B1_E0->M58_T_E0 [label="fM58_B1_E0_M58_T_E0=1"];
      M58_S_E0;
      M58_B1_E0 [label="M58_B1_E0\n226"];
      M58_T_E0;
}

**************************************************
Note: Remember to keep WCETAnalyzer updated
each time a bytecode implementation is changed.
```

sin(int,int)

**********APPLICATION WCET=5663**********
/***WCET calculation source***/
/* WCA WCET objective: util.FixPoint.sin */
max: tM58_S_E1 tM58_B1_E1 tM58_I2_E1 tM58_B3_E1 tM58_B4_E1 tM58_B5_E1 tM58_B6_E1 tM58_B7_E1
tM58_B8_E1 tM58_B9_E1 tM58_B10_E1 tM58_B11_E1 tM58_B12_E1 tM58_B13_E1 tM58_B14_E1 tM58_B15_E1
tM58_B16_E1 tM58_B17_E1 tM58_B18_E1 tM58_B19_E1 tM58_B20_E1 tM58_B21_E1 tM58_B22_E1 tM58_B23_E1
tM58_B24_E1 tM58_B25_E1 tM58_B26_E1 tM58_B27_E1 tM58_B28_E1 tM58_I29_E1 tM58_B30_E1 tM58_T_E1
tM55_S_E1 tM55_B1_E1 tM55_I2_E1 tM55_B3_E1 tM55_T_E1 tM54_S_E1 tM54_B1_E1 tM54_T_E1 tM54_S_E2
tM54_B1_E2 tM54_T_E2;
/* WCA flow constraints: sin : M58 */
M58_S_E1: 1 = fM58_S_E1_M58_B1_E1; // S flow
M58_B1_E1: fM58_S_E1_M58_B1_E1 + fM58_B1_E1_M58_B1_E1 = fM58_B1_E1_M58_I2_E1;
M58_I2_E1: fM58_I2_E1_M58_I2_E1 + fM58_B1_E1_M58_I2_E1 = fM58_I2_E1_M58_B3_E1;
/* Connecting(invoking) to util.FixPoint.Div(III)I id:M55_S_E1*/
M58_I2_E1_S: fcmM58_I2_E1_M55_S_E1 + fchM58_I2_E1_M55_S_E1 = fM58_I2_E1_M58_B3_E1; //cache S paths
M58_I2_E1_T: fM58_I2_E1_M58_B3_E1 = fM55_T_E1_M58_I2_E1; // invo T return path
fchM58_I2_E1_M55_S_E1 = 0; // no cache hits (because not innerloop)
/* Done with util.FixPoint.Div(III)I*/
M58_B3_E1: fM58_I2_E1_M58_B3_E1 + fM58_B3_E1_M58_B3_E1 = fM58_B3_E1_M58_B4_E1 +
fM58_B3_E1_M58_B5_E1;
M58_B4_E1: fM58_B4_E1_M58_B4_E1 + fM58_B3_E1_M58_B4_E1 = fM58_B4_E1_M58_B6_E1;
M58_B5_E1: fM58_B3_E1_M58_B5_E1 = fM58_B5_E1_M58_B6_E1;
M58_B6_E1: fM58_B5_E1_M58_B6_E1 + fM58_B4_E1_M58_B6_E1 = fM58_B6_E1_M58_B7_E1 +
fM58_B6_E1_M58_B8_E1;
M58_B7_E1: fM58_B7_E1_M58_B7_E1 + fM58_B6_E1_M58_B7_E1 = fM58_B7_E1_M58_B8_E1;
M58_B8_E1: fM58_B7_E1_M58_B8_E1 + fM58_B6_E1_M58_B8_E1 = fM58_B8_E1_M58_B9_E1 +
fM58_B8_E1_M58_B11_E1;
M58_B9_E1: fM58_B8_E1_M58_B9_E1 + fM58_B9_E1_M58_B9_E1 = fM58_B9_E1_M58_B10_E1 +
fM58_B9_E1_M58_B11_E1;
M58_B10_E1: fM58_B9_E1_M58_B10_E1 + fM58_B10_E1_M58_B10_E1 = fM58_B10_E1_M58_B18_E1;
M58_B11_E1: fM58_B8_E1_M58_B11_E1 + fM58_B9_E1_M58_B11_E1 = fM58_B11_E1_M58_B12_E1 +
fM58_B11_E1_M58_B14_E1;
M58_B12_E1: fM58_B11_E1_M58_B12_E1 + fM58_B12_E1_M58_B12_E1 = fM58_B12_E1_M58_B13_E1 +
fM58_B12_E1_M58_B14_E1;
M58_B13_E1: fM58_B12_E1_M58_B13_E1 + fM58_B13_E1_M58_B13_E1 = fM58_B13_E1_M58_B18_E1;
M58_B14_E1: fM58_B11_E1_M58_B14_E1 + fM58_B12_E1_M58_B14_E1 = fM58_B14_E1_M58_B15_E1 +
fM58_B14_E1_M58_B17_E1;
M58_B15_E1: fM58_B15_E1_M58_B15_E1 + fM58_B14_E1_M58_B15_E1 = fM58_B15_E1_M58_B16_E1 +
fM58_B15_E1_M58_B17_E1;
M58_B16_E1: fM58_B15_E1_M58_B16_E1 + fM58_B16_E1_M58_B16_E1 = fM58_B16_E1_M58_B18_E1;
M58_B17_E1: fM58_B15_E1_M58_B17_E1 + fM58_B14_E1_M58_B17_E1 = fM58_B17_E1_M58_B18_E1;
M58_B18_E1: fM58_B16_E1_M58_B18_E1 + fM58_B17_E1_M58_B18_E1 + fM58_B13_E1_M58_B18_E1 +
fM58_B10_E1_M58_B18_E1 = fM58_B18_E1_M58_B19_E1 + fM58_B18_E1_M58_B21_E1;
M58_B19_E1: fM58_B19_E1_M58_B19_E1 + fM58_B18_E1_M58_B19_E1 = fM58_B19_E1_M58_B20_E1 +
fM58_B19_E1_M58_B21_E1;
M58_B20_E1: fM58_B19_E1_M58_B20_E1 + fM58_B20_E1_M58_B20_E1 = fM58_B20_E1_M58_B28_E1;
M58_B21_E1: fM58_B19_E1_M58_B21_E1 + fM58_B18_E1_M58_B21_E1 = fM58_B21_E1_M58_B22_E1 +
fM58_B21_E1_M58_B24_E1;
M58_B22_E1: fM58_B21_E1_M58_B22_E1 + fM58_B22_E1_M58_B22_E1 = fM58_B22_E1_M58_B23_E1 +
fM58_B22_E1_M58_B24_E1;
M58_B23_E1: fM58_B22_E1_M58_B23_E1 + fM58_B23_E1_M58_B23_E1 = fM58_B23_E1_M58_B28_E1;
M58_B24_E1: fM58_B21_E1_M58_B24_E1 + fM58_B22_E1_M58_B24_E1 = fM58_B24_E1_M58_B25_E1 +
fM58_B24_E1_M58_B27_E1;
M58_B25_E1: fM58_B25_E1_M58_B25_E1 + fM58_B24_E1_M58_B25_E1 = fM58_B25_E1_M58_B26_E1 +
fM58_B25_E1_M58_B27_E1;
M58_B26_E1: fM58_B25_E1_M58_B26_E1 + fM58_B26_E1_M58_B26_E1 = fM58_B26_E1_M58_B28_E1;
M58_B27_E1: fM58_B25_E1_M58_B27_E1 + fM58_B24_E1_M58_B27_E1 = fM58_B27_E1_M58_B28_E1;
M58_B28_E1: fM58_B26_E1_M58_B28_E1 + fM58_B27_E1_M58_B28_E1 + fM58_B20_E1_M58_B28_E1 +

```
fM58_B23_E1_M58_B28_E1 = fM58_B28_E1_M58_I29_E1;
M58_I29_E1: fM58_I29_E1_M58_I29_E1 + fM58_B28_E1_M58_I29_E1 = fM58_I29_E1_M58_B30_E1;
/* Connecting(invoking) to util.FixPoint.Mul(III)I id:M54_S_E2*/
M58_I29_E1_S: fcmM58_I29_E1_M54_S_E2 + fchM58_I29_E1_M54_S_E2 = fM58_I29_E1_M58_B30_E1; //cache S
paths
M58_I29_E1_T: fM58_I29_E1_M58_B30_E1 = fM54_T_E2_M58_I29_E1; // invo T return path
fchM58_I29_E1_M54_S_E2 = 0; // no cache hits (because not innerloop)
/* Done with util.FixPoint.Mul(III)I*/
M58_B30_E1: fM58_I29_E1_M58_B30_E1 + fM58_B30_E1_M58_B30_E1 = fM58_B30_E1_M58_T_E1;
M58_T_E1: fM58_B30_E1_M58_T_E1 = 1; // T flow
/* WCA flow to cycle count */
tM58_B1_E1 = 69 fM58_S_E1_M58_B1_E1 + 69 fM58_B1_E1_M58_B1_E1;
tM58_I2_E1 = 75 fM58_I2_E1_M58_I2_E1 + 75 fM58_B1_E1_M58_I2_E1;
tM58_B3_E1 = 1955 fM58_I2_E1_M58_B3_E1 + 1955 fM58_B3_E1_M58_B3_E1;
tM58_B4_E1 = 14 fM58_B4_E1_M58_B4_E1 + 14 fM58_B3_E1_M58_B4_E1;
tM58_B5_E1 = 8 fM58_B3_E1_M58_B5_E1;
tM58_B6_E1 = 5 fM58_B5_E1_M58_B6_E1 + 5 fM58_B4_E1_M58_B6_E1;
tM58_B7_E1 = 6 fM58_B7_E1_M58_B7_E1 + 6 fM58_B6_E1_M58_B7_E1;
tM58_B8_E1 = 12 fM58_B7_E1_M58_B8_E1 + 12 fM58_B6_E1_M58_B8_E1;
tM58_B9_E1 = 8 fM58_B8_E1_M58_B9_E1 + 8 fM58_B9_E1_M58_B9_E1;
tM58_B10_E1 = 65 fM58_B9_E1_M58_B10_E1 + 65 fM58_B10_E1_M58_B10_E1;
tM58_B11_E1 = 8 fM58_B8_E1_M58_B11_E1 + 8 fM58_B9_E1_M58_B11_E1;
tM58_B12_E1 = 8 fM58_B11_E1_M58_B12_E1 + 8 fM58_B12_E1_M58_B12_E1;
tM58_B13_E1 = 69 fM58_B12_E1_M58_B13_E1 + 69 fM58_B13_E1_M58_B13_E1;
tM58_B14_E1 = 8 fM58_B11_E1_M58_B14_E1 + 8 fM58_B12_E1_M58_B14_E1;
tM58_B15_E1 = 8 fM58_B15_E1_M58_B15_E1 + 8 fM58_B14_E1_M58_B15_E1;
tM58_B16_E1 = 69 fM58_B15_E1_M58_B16_E1 + 69 fM58_B16_E1_M58_B16_E1;
tM58_B17_E1 = 57 fM58_B15_E1_M58_B17_E1 + 57 fM58_B14_E1_M58_B17_E1;
tM58_B18_E1 = 8 fM58_B16_E1_M58_B18_E1 + 8 fM58_B17_E1_M58_B18_E1 + 8 fM58_B13_E1_M58_B18_E1 + 8
fM58_B10_E1_M58_B18_E1;
tM58_B19_E1 = 9 fM58_B19_E1_M58_B19_E1 + 9 fM58_B18_E1_M58_B19_E1;
tM58_B20_E1 = 66 fM58_B19_E1_M58_B20_E1 + 66 fM58_B20_E1_M58_B20_E1;
tM58_B21_E1 = 9 fM58_B19_E1_M58_B21_E1 + 9 fM58_B18_E1_M58_B21_E1;
tM58_B22_E1 = 9 fM58_B21_E1_M58_B22_E1 + 9 fM58_B22_E1_M58_B22_E1;
tM58_B23_E1 = 70 fM58_B22_E1_M58_B23_E1 + 70 fM58_B23_E1_M58_B23_E1;
tM58_B24_E1 = 9 fM58_B21_E1_M58_B24_E1 + 9 fM58_B22_E1_M58_B24_E1;
tM58_B25_E1 = 9 fM58_B25_E1_M58_B25_E1 + 9 fM58_B24_E1_M58_B25_E1;
tM58_B26_E1 = 70 fM58_B25_E1_M58_B26_E1 + 70 fM58_B26_E1_M58_B26_E1;
tM58_B27_E1 = 58 fM58_B25_E1_M58_B27_E1 + 58 fM58_B24_E1_M58_B27_E1;
tM58_B28_E1 = 7 fM58_B26_E1_M58_B28_E1 + 7 fM58_B27_E1_M58_B28_E1 + 7 fM58_B20_E1_M58_B28_E1 + 7
fM58_B23_E1_M58_B28_E1;
tM58_I29_E1 = 75 fM58_I29_E1_M58_I29_E1 + 75 fM58_B28_E1_M58_I29_E1;
tM58_B30_E1 = 24 fM58_I29_E1_M58_B30_E1 + 24 fM58_B30_E1_M58_B30_E1;
/* Invocation(s) from sin:[M58] */
/* WCA flow constraints: Div : M55 */
M55_S_E1: fchM58_I2_E1_M55_S_E1+ fcmM58_I2_E1_M55_S_E1 = fM55_S_E1_M55_B1_E1; // S flow
// tM55_S_E1 = 0 fcmM58_I2_E1_M55_S_E1; // S cache miss time
M55_B1_E1: fM55_S_E1_M55_B1_E1 + fM55_B1_E1_M55_B1_E1 = fM55_B1_E1_M55_I2_E1;
M55_I2_E1: fM55_I2_E1_M55_I2_E1 + fM55_B1_E1_M55_I2_E1 = fM55_I2_E1_M55_B3_E1;
/* Connecting(invoking) to util.FixPoint.Mul(III)I id:M54_S_E1*/
M55_I2_E1_S: fcmM55_I2_E1_M54_S_E1 + fchM55_I2_E1_M54_S_E1 = fM55_I2_E1_M55_B3_E1; //cache S paths
M55_I2_E1_T: fM55_I2_E1_M55_B3_E1 = fM54_T_E1_M55_I2_E1; // invo T return path
fchM55_I2_E1_M54_S_E1 = 0; // no cache hits (because not innerloop)
/* Done with util.FixPoint.Mul(III)I*/
M55_B3_E1: fM55_I2_E1_M55_B3_E1 + fM55_B3_E1_M55_B3_E1 = fM55_B3_E1_M55_T_E1;
M55_T_E1: fM55_B3_E1_M55_T_E1 = fM55_T_E1_M58_I2_E1;// T interconnect flow
tM55_T_E1 = 156 fM55_T_E1_M58_I2_E1; // T cache miss (not leaf)
/* WCA flow to cycle count */
```

```
tM55_B1_E1 = 2473 fM55_S_E1_M55_B1_E1 + 2473 fM55_B1_E1_M55_B1_E1;
tM55_I2_E1 = 75 fM55_I2_E1_M55_I2_E1 + 75 fM55_B1_E1_M55_I2_E1;
tM55_B3_E1 = 23 fM55_I2_E1_M55_B3_E1 + 23 fM55_B3_E1_M55_B3_E1;
/* Invocation(s) from Div:[M55] */
/* WCA flow constraints: Mul : M54 */
M54_S_E1: fchM55_I2_E1_M54_S_E1+ fcmM55_I2_E1_M54_S_E1 = fM54_S_E1_M54_B1_E1; // S flow
tM54_S_E1 = 5 fcmM55_I2_E1_M54_S_E1; // S cache miss time
M54_B1_E1: fM54_S_E1_M54_B1_E1 + fM54_B1_E1_M54_B1_E1 = fM54_B1_E1_M54_T_E1;
M54_T_E1: fM54_B1_E1_M54_T_E1 = fM54_T_E1_M55_I2_E1;// T interconnect flow
// tM54_T_E1 = 0 fM54_T_E1_M55_I2_E1; // T cache hit (leaf)
/* WCA flow to cycle count */
tM54_B1_E1 = 226 fM54_S_E1_M54_B1_E1 + 226 fM54_B1_E1_M54_B1_E1;
/* Invocation(s) from Mul:[M54] */
/* WCA flow constraints: Mul : M54 */
M54_S_E2: fchM58_I29_E1_M54_S_E2+ fcmM58_I29_E1_M54_S_E2 = fM54_S_E2_M54_B1_E2; // S flow
tM54_S_E2 = 5 fcmM58_I29_E1_M54_S_E2; // S cache miss time
M54_B1_E2: fM54_S_E2_M54_B1_E2 + fM54_B1_E2_M54_B1_E2 = fM54_B1_E2_M54_T_E2;
M54_T_E2: fM54_B1_E2_M54_T_E2 = fM54_T_E2_M58_I29_E1;// T interconnect flow
// tM54_T_E2 = 0 fM54_T_E2_M58_I29_E1; // T cache hit (leaf)
/* WCA flow to cycle count */
tM54_B1_E2 = 226 fM54_S_E2_M54_B1_E2 + 226 fM54_B1_E2_M54_B1_E2;
/* Invocation(s) from Mul:[M54] */

Value of objective function: 5663

Actual values of the variables:
tM58_S_E1                0
tM58_B1_E1              69
tM58_I2_E1              75
tM58_B3_E1            1955
tM58_B4_E1              14
tM58_B5_E1               0
tM58_B6_E1               5
tM58_B7_E1               6
tM58_B8_E1              12
tM58_B9_E1               8
tM58_B10_E1              0
tM58_B11_E1              8
tM58_B12_E1              8
tM58_B13_E1              0
tM58_B14_E1              8
tM58_B15_E1              8
tM58_B16_E1             69
tM58_B17_E1              0
tM58_B18_E1              8
tM58_B19_E1              9
tM58_B20_E1              0
tM58_B21_E1              9
tM58_B22_E1              9
tM58_B23_E1              0
tM58_B24_E1              9
tM58_B25_E1              9
tM58_B26_E1             70
tM58_B27_E1              0
tM58_B28_E1              7
tM58_I29_E1             75
tM58_B30_E1             24
tM58_T_E1                0
```

tM55_S_E1	0
tM55_B1_E1	2473
tM55_I2_E1	75
tM55_B3_E1	23
tM55_T_E1	156
tM54_S_E1	5
tM54_B1_E1	226
tM54_T_E1	0
tM54_S_E2	5
tM54_B1_E2	226
tM54_T_E2	0
fM58_S_E1_M58_B1_E1	1
fM58_B1_E1_M58_B1_E1	0
fM58_B1_E1_M58_I2_E1	1
fM58_I2_E1_M58_I2_E1	0
fM58_I2_E1_M58_B3_E1	1
fcmM58_I2_E1_M55_S_E1	1
fchM58_I2_E1_M55_S_E1	0
fM55_T_E1_M58_I2_E1	1
fM58_B3_E1_M58_B3_E1	0
fM58_B3_E1_M58_B4_E1	1
fM58_B3_E1_M58_B5_E1	0
fM58_B4_E1_M58_B4_E1	0
fM58_B4_E1_M58_B6_E1	1
fM58_B5_E1_M58_B6_E1	0
fM58_B6_E1_M58_B7_E1	1
fM58_B6_E1_M58_B8_E1	0
fM58_B7_E1_M58_B7_E1	0
fM58_B7_E1_M58_B8_E1	1
fM58_B8_E1_M58_B9_E1	1
fM58_B8_E1_M58_B11_E1	0
fM58_B9_E1_M58_B9_E1	0
fM58_B9_E1_M58_B10_E1	0
fM58_B9_E1_M58_B11_E1	1
fM58_B10_E1_M58_B10_E1	0
fM58_B10_E1_M58_B18_E1	0
fM58_B11_E1_M58_B12_E1	1
fM58_B11_E1_M58_B14_E1	0
fM58_B12_E1_M58_B12_E1	0
fM58_B12_E1_M58_B13_E1	0
fM58_B12_E1_M58_B14_E1	1
fM58_B13_E1_M58_B13_E1	0
fM58_B13_E1_M58_B18_E1	0
fM58_B14_E1_M58_B15_E1	1
fM58_B14_E1_M58_B17_E1	0
fM58_B15_E1_M58_B15_E1	0
fM58_B15_E1_M58_B16_E1	1
fM58_B15_E1_M58_B17_E1	0
fM58_B16_E1_M58_B16_E1	0
fM58_B16_E1_M58_B18_E1	1
fM58_B17_E1_M58_B18_E1	0
fM58_B18_E1_M58_B19_E1	1
fM58_B18_E1_M58_B21_E1	0
fM58_B19_E1_M58_B19_E1	0
fM58_B19_E1_M58_B20_E1	0
fM58_B19_E1_M58_B21_E1	1
fM58_B20_E1_M58_B20_E1	0
fM58_B20_E1_M58_B28_E1	0

fM58_B21_E1_M58_B22_E1 1
fM58_B21_E1_M58_B24_E1 0
fM58_B22_E1_M58_B22_E1 0
fM58_B22_E1_M58_B23_E1 0
fM58_B22_E1_M58_B24_E1 1
fM58_B23_E1_M58_B23_E1 0
fM58_B23_E1_M58_B28_E1 0
fM58_B24_E1_M58_B25_E1 1
fM58_B24_E1_M58_B27_E1 0
fM58_B25_E1_M58_B25_E1 0
fM58_B25_E1_M58_B26_E1 1
fM58_B25_E1_M58_B27_E1 0
fM58_B26_E1_M58_B26_E1 0
fM58_B26_E1_M58_B28_E1 1
fM58_B27_E1_M58_B28_E1 0
fM58_B28_E1_M58_I29_E1 1
fM58_I29_E1_M58_I29_E1 0
fM58_I29_E1_M58_B30_E1 1
fcmM58_I29_E1_M54_S_E2 1
fchM58_I29_E1_M54_S_E2 0
fM54_T_E2_M58_I29_E1 1
fM58_B30_E1_M58_B30_E1 0
fM58_B30_E1_M58_T_E1 1
fM55_S_E1_M55_B1_E1 1
fM55_B1_E1_M55_B1_E1 0
fM55_B1_E1_M55_I2_E1 1
fM55_I2_E1_M55_I2_E1 0
fM55_I2_E1_M55_B3_E1 1
fcmM55_I2_E1_M54_S_E1 1
fchM55_I2_E1_M54_S_E1 0
fM54_T_E1_M55_I2_E1 1
fM55_B3_E1_M55_B3_E1 0
fM55_B3_E1_M55_T_E1 1
fM54_S_E1_M54_B1_E1 1
fM54_B1_E1_M54_B1_E1 0
fM54_B1_E1_M54_T_E1 1
fM54_S_E2_M54_B1_E2 1
fM54_B1_E2_M54_B1_E2 0
fM54_B1_E2_M54_T_E2 1

**********END APPLICATION WCET**********
****
WCET info for:util.FixPoint.Mul(III)I

Directed graph of basic blocks(row->column):
==========
M54_S_E0M54_B1_E0M54_T_E0
M54_S_E0 . 1 .
M54_B1_E0 . . 1
M54_T_E0 . . .
==========

Table of basic blocks' and instructions
==========
Block Addr. Bytecode Cycles Cache miss Misc. info
[opcode] invoke return

M54_S_E0'S'

```
  Src. line 112: int res = 0;
M54_B1_E0'B'{}<-[M54_S_E0 M54_B1_E0]0:     iconst_0[3]                 1
     1:    istore_3[62]              1                ->I:res
  Src. line 113: res = ((f1 >> shift) * (f2 >> shift)) << shift; // AH*BH
     2:    iload_0[26]               1                I:f1
     3:    iload_2[28]               1                I:shift
     4:    ishr[122]                 1
     5:    iload_1[27]               1                I:f2
     6:    iload_2[28]               1                I:shift
     7:    ishr[122]                 1
     8:    imul[104]                 35
     9:    iload_2[28]               1                I:shift
     10:   ishl[120]                 1
     11:   istore_3[62]              1                ->I:res
  Src. line 114: res += ((f1 >> shift) * (f2 - ((f2 >> shift) << shift))); // AH*BL
     12:   iload_3[29]               1                I:res
     13:   iload_0[26]               1                I:f1
     14:   iload_2[28]               1                I:shift
     15:   ishr[122]                 1
     16:   iload_1[27]               1                I:f2
     17:   iload_1[27]               1                I:f2
     18:   iload_2[28]               1                I:shift
     19:   ishr[122]                 1
     20:   iload_2[28]               1                I:shift
     21:   ishl[120]                 1
     22:   isub[100]                 1
     23:   imul[104]                 35
     24:   iadd[96]                  1
     25:   istore_3[62]              1                ->I:res
  Src. line 115: res += ((f1 - ((f1 >> shift) << shift)) * (f2 >> shift)); // AL*BH
     26:   iload_3[29]               1                I:res
     27:   iload_0[26]               1                I:f1
     28:   iload_0[26]               1                I:f1
     29:   iload_2[28]               1                I:shift
     30:   ishr[122]                 1
     31:   iload_2[28]               1                I:shift
     32:   ishl[120]                 1
     33:   isub[100]                 1
     34:   iload_1[27]               1                I:f2
     35:   iload_2[28]               1                I:shift
     36:   ishr[122]                 1
     37:   imul[104]                 35
     38:   iadd[96]                  1
     39:   istore_3[62]              1                ->I:res
  Src. line 116: res += (((f1 - ((f1 >> shift) << shift)) >> 1)
     40:   iload_3[29]               1                I:res
     41:   iload_0[26]               1                I:f1
     42:   iload_0[26]               1                I:f1
     43:   iload_2[28]               1                I:shift
     44:   ishr[122]                 1
     45:   iload_2[28]               1                I:shift
     46:   ishl[120]                 1
     47:   isub[100]                 1
     48:   iconst_1[4]               1
     49:   ishr[122]                 1
     50:   iload_1[27]               1                I:f2
     51:   iload_1[27]               1                I:f2
     52:   iload_2[28]               1                I:shift
```

```
    53:   ishr[122]              1
    54:   iload_2[28]            1            I:shift
    55:   ishl[120]              1
    56:   isub[100]              1
    57:   imul[104]              35
    58:   iconst_1[4]            1
    59:   ishr[122]              1
    60:   iload_2[28]            1            I:shift
    61:   iconst_2[5]            1
    62:   isub[100]              1
    63:   ishr[122]              1
    64:   iadd[96]               1
    65:   istore_3[62]           1            ->I:res
 Src. line 119: return res;
    66:   iload_3[29]            1            I:res
    67:   ireturn[172]           23            sum(B1):   226
M54_T_E0'T'<-[M54_B1_E0]
================================================================================
Info: n=17 a=-1 r=1 w=1

/***WCET calculation source***/
/* WCA WCET objective: util.FixPoint.Mul */
max: tM54_S_E0 tM54_B1_E0 tM54_T_E0;
/* WCA flow constraints: Mul : M54 */
M54_S_E0: 1 = fM54_S_E0_M54_B1_E0; // S flow
M54_B1_E0: fM54_S_E0_M54_B1_E0 + fM54_B1_E0_M54_B1_E0 = fM54_B1_E0_M54_T_E0;
M54_T_E0: fM54_B1_E0_M54_T_E0 = 1; // T flow
/* WCA flow to cycle count */
tM54_B1_E0 = 226 fM54_S_E0_M54_B1_E0 + 226 fM54_B1_E0_M54_B1_E0;
/* Invocation(s) from Mul:[M54] */

Value of objective function: 226

Actual values of the variables:
tM54_S_E0                  0
tM54_B1_E0                 226
tM54_T_E0                  0
fM54_S_E0_M54_B1_E0        1
fM54_B1_E0_M54_B1_E0       0
fM54_B1_E0_M54_T_E0        1

/*util.FixPoint.Mul(III)I*/
digraph G {
size = "10,7.5"
        M54_S_E0->M54_B1_E0 [label="fM54_S_E0_M54_B1_E0=1"];
        M54_B1_E0->M54_T_E0 [label="fM54_B1_E0_M54_T_E0=1"];
        M54_S_E0;
        M54_B1_E0 [label="M54_B1_E0\n226"];
        M54_T_E0;
}
********************************************************************************
WCET info for:util.FixPoint.Div(III)I

Directed graph of basic blocks(row->column):
================================================
   M55_S_E0M55_B1_E0M55_I2_E0M55_B3_E0M55_T_E0
M55_S_E0 .   1   .   .   .
M55_B1_E0 .   .   1   .   .
```

```
M55_I2_E0 .  .  .  1  .
M55_B3_E0 .  .  .  .  1
M55_T_E0 .  .  .  .  .
================================================

Table of basic blocks' and instructions
================================================================================
Block Addr.  Bytecode              Cycles    Cache miss     Misc. info
             [opcode]              invoke  return
--------------------------------------------------------------------------------
M55_S_E0'S'
  Src. line 136: f2 = (1 << ((shift * 2) - 2)) / ((f2 + 1) >> 2); // approx. 1/f2
M55_B1_E0'B'{}<-[M55_S_E0 M55_B1_E0]0:     iconst_1[4]               1
     1:     iload_2[28]               1               I:shift
     2:     iconst_2[5]               1
     3:     imul[104]                35
     4:     iconst_2[5]               1
     5:     isub[100]                 1
     6:     ishl[120]                 1
     7:     iload_1[27]               1               I:f2
     8:     iconst_1[4]               1
     9:     iadd[96]                  1
     10:    iconst_2[5]               1
     11:    ishr[122]                 1
     12:    idiv[108]              2423
     13:    istore_1[60]              1               ->I:f2
  Src. line 137: return Mul(f1, f2, shift);
     14:    iload_0[26]               1               I:f1
     15:    iload_1[27]               1               I:f2
     16:    iload_2[28]               1               I:shift sum(B1):   2473
  Src. line 137: return Mul(f1, f2, shift);
M55_I2_E0'I'{}<-[M55_I2_E0 M55_B1_E0]17:    invokestatic[184]          75       5      10   util.FixPoint.Mul(III)I,
invoke(n=17):75/80 return(n=6):23/33sum(B2):     75
  Src. line 137: return Mul(f1, f2, shift);
M55_B3_E0'B'{}<-[M55_I2_E0 M55_B3_E0]20:    ireturn[172]              23                sum(B3):     23
M55_T_E0'T'<-[M55_B3_E0]
================================================================================
Info: n=6 a=-1 r=1 w=1

/***WCET calculation source***/
/* WCA WCET objective: util.FixPoint.Div */
max: tM55_S_E0 tM55_B1_E0 tM55_I2_E0 tM55_B3_E0 tM55_T_E0;
/* WCA flow constraints: Div : M55 */
M55_S_E0: 1 = fM55_S_E0_M55_B1_E0; // S flow
M55_B1_E0: fM55_S_E0_M55_B1_E0 + fM55_B1_E0_M55_B1_E0 = fM55_B1_E0_M55_I2_E0;
M55_I2_E0: fM55_I2_E0_M55_I2_E0 + fM55_B1_E0_M55_I2_E0 = fM55_I2_E0_M55_B3_E0;
M55_B3_E0: fM55_I2_E0_M55_B3_E0 + fM55_B3_E0_M55_B3_E0 = fM55_B3_E0_M55_T_E0;
M55_T_E0: fM55_B3_E0_M55_T_E0 = 1; // T flow
/* WCA flow to cycle count */
tM55_B1_E0 = 2473 fM55_S_E0_M55_B1_E0 + 2473 fM55_B1_E0_M55_B1_E0;
tM55_I2_E0 = 75 fM55_I2_E0_M55_I2_E0 + 75 fM55_B1_E0_M55_I2_E0;
tM55_B3_E0 = 23 fM55_I2_E0_M55_B3_E0 + 23 fM55_B3_E0_M55_B3_E0;
/* Invocation(s) from Div:[M55] */

Value of objective function: 2571

Actual values of the variables:
tM55_S_E0                  0
```

```
tM55_B1_E0                  2473
tM55_I2_E0                  75
tM55_B3_E0                  23
tM55_T_E0                   0
fM55_S_E0_M55_B1_E0            1
fM55_B1_E0_M55_B1_E0           0
fM55_B1_E0_M55_I2_E0           1
fM55_I2_E0_M55_I2_E0           0
fM55_I2_E0_M55_B3_E0           1
fM55_B3_E0_M55_B3_E0           0
fM55_B3_E0_M55_T_E0            1

/*util.FixPoint.Div(III)I*/
digraph G {
size = "10,7.5"
        M55_S_E0->M55_B1_E0 [label="fM55_S_E0_M55_B1_E0=1"];
        M55_B1_E0->M55_I2_E0 [label="fM55_B1_E0_M55_I2_E0=1"];
        M55_I2_E0->M55_B3_E0 [label="fM55_I2_E0_M55_B3_E0=1"];
        M55_B3_E0->M55_T_E0 [label="fM55_B3_E0_M55_T_E0=1"];
        M55_S_E0;
        M55_B1_E0 [label="M55_B1_E0\n2473"];
        M55_I2_E0 [label="M55_I2_E0\n75"];
        M55_B3_E0 [label="M55_B3_E0\n23"];
        M55_T_E0;
}
******************************************************************************
WCET info for:util.FixPoint.sin(II)I

Directed graph of basic blocks(row->column):
====================================================================================================
====================================================================================================
====================================================================================================
==================================================
   M58_S_E0M58_B1_E0M58_I2_E0M58_B3_E0M58_B4_E0M58_B5_E0M58_B6_E0M58_B7_E0M58_B8_E0M58_B
9_E0M58_B10_E0M58_B11_E0M58_B12_E0M58_B13_E0M58_B14_E0M58_B15_E0M58_B16_E0M58_B17_E0M58_
B18_E0M58_B19_E0M58_B20_E0M58_B21_E0M58_B22_E0M58_B23_E0M58_B24_E0M58_B25_E0M58_B26_E0M
58_B27_E0M58_B28_E0M58_I29_E0M58_B30_E0M58_T_E0
M58_S_E0 . 1 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
M58_B1_E0 . . 1 . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
M58_I2_E0 . . . 1 . . . . . . . . . . . . . . . . . . . . . . . . . . . .
M58_B3_E0 . . . . 1 1 . . . . . . . . . . . . . . . . . . . . . . . . . .
M58_B4_E0 . . . . . . 1 . . . . . . . . . . . . . . . . . . . . . . . . .
M58_B5_E0 . . . . . . 1 . . . . . . . . . . . . . . . . . . . . . . . . .
M58_B6_E0 . . . . . . . 1 1 . . . . . . . . . . . . . . . . . . . . . . .
M58_B7_E0 . . . . . . . . 1 . . . . . . . . . . . . . . . . . . . . . . .
M58_B8_E0 . . . . . . . . . 1 . 1 . . . . . . . . . . . . . . . . . . . .
M58_B9_E0 . . . . . . . . . . 1 1 . . . . . . . . . . . . . . . . . . . .
M58_B10_E0 . . . . . . . . . . . . . . . . . . 1 . . . . . . . . . . . . .
M58_B11_E0 . . . . . . . . . . . . 1 . 1 . . . . . . . . . . . . . . . . .
M58_B12_E0 . . . . . . . . . . . . . 1 1 . . . . . . . . . . . . . . . . .
M58_B13_E0 . . . . . . . . . . . . . . . . . . 1 . . . . . . . . . . . . .
M58_B14_E0 . . . . . . . . . . . . . . . 1 . 1 . . . . . . . . . . . . . .
M58_B15_E0 . . . . . . . . . . . . . . . . 1 1 . . . . . . . . . . . . . .
M58_B16_E0 . . . . . . . . . . . . . . . . . . 1 . . . . . . . . . . . . .
M58_B17_E0 . . . . . . . . . . . . . . . . . . 1 . . . . . . . . . . . . .
M58_B18_E0 . . . . . . . . . . . . . . . . . . . 1 . 1 . . . . . . . . . .
M58_B19_E0 . . . . . . . . . . . . . . . . . . . . 1 1 . . . . . . . . . .
M58_B20_E0 . . . . . . . . . . . . . . . . . . . . . . . . . . . . 1 . . .
```

```
M58_B21_E0 . . . . . 1 . 1 . . . . . .
M58_B22_E0 . . . . . 1 1 . . . . . .
M58_B23_E0 . . . . . 1 . . .
M58_B24_E0 . . . . . 1 . 1 . . . .
M58_B25_E0 . . . . . 1 1 . . . .
M58_B26_E0 . . . . . 1 . . .
M58_B27_E0 . . . . . 1 . . .
M58_B28_E0 . . . . . 1 . .
M58_I29_E0 . . . . . 1 .
M58_B30_E0 . . . . . 1
M58_T_E0 . . . . . .
================================================================================================
================================================================================================
================================================================================================
==============================================

Table of basic blocks' and instructions
=======================================================================
Block Addr.  Bytecode              Cycles    Cache miss     Misc. info
          [opcode]                 invoke  return
-----------------------------------------------------------------------
M58_S_E0'S'
  Src. line 178: int Temp1 = 0;
M58_B1_E0'B'{}<-[M58_S_E0 M58_B1_E0]0:     iconst_0[3]              1
     1:     istore_2[61]             1               ->I:Temp1
  Src. line 179: int Temp2 = 0;
     2:     iconst_0[3]              1
     3:     istore_3[62]             1               ->I:Temp2
  Src. line 180: int Rest = 0;
     4:     iconst_0[3]              1
     5:     istore[54]               2               ->I:Rest
  Src. line 181: int b = 0;
     7:     iconst_0[3]              1
     8:     istore[54]               2               ->I:b
  Src. line 183: Rest = Div(a * 180, PI >> (29 - SHIFT), SHIFT);
     10:    iload_0[26]               1               I:a
     11:    sipush[17]                3
     14:    imul[104]                35
     15:    getstatic[178]           14               int util.FixPoint.PI
     18:    bipush[16]                2
     20:    iload_1[27]               1               I:SHIFT
     21:    isub[100]                1
     22:    ishr[122]                1
     23:    iload_1[27]               1               I:SHIFT sum(B1):     69
  Src. line 183: Rest = Div(a * 180, PI >> (29 - SHIFT), SHIFT);
M58_I2_E0'I'{}<-[M58_I2_E0 M58_B1_E0]24:    invokestatic[184]          75       0     156   util.FixPoint.Div(III)I,
invoke(n=6):75/75 return(n=79):23/179sum(B2):     75
  Src. line 183: Rest = Div(a * 180, PI >> (29 - SHIFT), SHIFT);
M58_B3_E0'B'{}<-[M58_I2_E0 M58_B3_E0]27:    istore[54]               2               ->I:Rest
  Src. line 184: a = (Rest >> SHIFT) % 360;
     29:    iload[21]                 2               I:Rest
     31:    iload_1[27]               1               I:SHIFT
     32:    ishr[122]                1
     33:    sipush[17]                3
     36:    irem[112]               1940
     37:    istore_0[59]              1               ->I:a
  Src. line 185: if (a == 0)
     38:    iload_0[26]               1               I:a
```

```
    39:   ifne[154]->55:            4              sum(B3):   1955
  Src. line 186: Rest -= (360 << SHIFT);
M58_B4_E0'B'{}<-[M58_B4_E0 M58_B3_E0]42:   iload[21]              2              I:Rest
    44:   sipush[17]              3
    47:   iload_1[27]             1              I:SHIFT
    48:   ishl[120]               1
    49:   isub[100]               1
    50:   istore[54]              2              ->I:Rest
    52:   goto[167]->63:          4              sum(B4):   14
  Src. line 188: Rest -= (a << SHIFT);
M58_B5_E0'B'{}<-[M58_B3_E0]55:   iload[21]              2              I:Rest
    57:   iload_0[26]             1              I:a
    58:   iload_1[27]             1              I:SHIFT
    59:   ishl[120]               1
    60:   isub[100]               1
    61:   istore[54]              2              ->I:Rest sum(B5):   8
  Src. line 189: if (a < 0)
M58_B6_E0'B'{}<-[M58_B5_E0 M58_B4_E0]63:   iload_0[26]            1              I:a
    64:   ifge[156]->73:          4              sum(B6):   5
  Src. line 190: a += 360;
M58_B7_E0'B'{}<-[M58_B7_E0 M58_B6_E0]67:   iload_0[26]            1              I:a
    68:   sipush[17]              3
    71:   iadd[96]                1
    72:   istore_0[59]            1              ->I:a sum(B7):   6
  Src. line 191: b = a + 1;
M58_B8_E0'B'{}<-[M58_B7_E0 M58_B6_E0]73:   iload_0[26]            1              I:a
    74:   iconst_1[4]             1
    75:   iadd[96]                1
    76:   istore[54]              2              ->I:b
  Src. line 196: if (90 <= a && a <= 180)
    78:   bipush[16]              2
    80:   iload_0[26]             1              I:a
    81:   if_icmpgt[163]->109:    4              sum(B8):   12
  Src. line 196: if (90 <= a && a <= 180)
M58_B9_E0'B'{}<-[M58_B8_E0 M58_B9_E0]84:   iload_0[26]            1              I:a
    85:   sipush[17]              3
    88:   if_icmpgt[163]->109:    4              sum(B9):   8
  Src. line 197: Temp1 = (SIN[180 - a] >> (30 - SHIFT));
M58_B10_E0'B'{}<-[M58_B9_E0 M58_B10_E0]91:   getstatic[178]         14             int[] util.FixPoint.SIN
    94:   sipush[17]              3
    97:   iload_0[26]             1              I:a
    98:   isub[100]               1
    99:   iaload[46]              36             I
    100:  bipush[16]              2
    102:  iload_1[27]             1              I:SHIFT
    103:  isub[100]               1
    104:  ishr[122]               1
    105:  istore_2[61]            1              ->I:Temp1
    106:  goto[167]->186:         4              sum(B10):   65
  Src. line 198: else if (180 <= a && a <= 270)
M58_B11_E0'B'{}<-[M58_B8_E0 M58_B9_E0]109:   sipush[17]            3
    112:  iload_0[26]             1              I:a
    113:  if_icmpgt[163]->142:    4              sum(B11):   8
  Src. line 198: else if (180 <= a && a <= 270)
M58_B12_E0'B'{}<-[M58_B11_E0 M58_B12_E0]116:   iload_0[26]           1              I:a
    117:  sipush[17]              3
    120:  if_icmpgt[163]->142:    4              sum(B12):   8
  Src. line 199: Temp1 = -(SIN[a - 180] >> (30 - SHIFT));
```

```
M58_B13_E0'B'{}<-[M58_B12_E0 M58_B13_E0]123:   getstatic[178]          14              int[] util.FixPoint.SIN
    126:   iload_0[26]            1              I:a
    127:   sipush[17]            3
    130:   isub[100]             1
    131:   iaload[46]            36             I
    132:   bipush[16]            2
    134:   iload_1[27]            1              I:SHIFT
    135:   isub[100]             1
    136:   ishr[122]             1
    137:   ineg[116]             4
    138:   istore_2[61]           1              ->I:Temp1
    139:   goto[167]->186:          4              sum(B13):    69
 Src. line 200: else if (270 <= a && a <= 360)
M58_B14_E0'B'{}<-[M58_B11_E0 M58_B12_E0]142:   sipush[17]            3
    145:   iload_0[26]            1              I:a
    146:   if_icmpgt[163]->175:       4              sum(B14):    8
 Src. line 200: else if (270 <= a && a <= 360)
M58_B15_E0'B'{}<-[M58_B15_E0 M58_B14_E0]149:   iload_0[26]            1              I:a
    150:   sipush[17]            3
    153:   if_icmpgt[163]->175:       4              sum(B15):    8
 Src. line 201: Temp1 = -(SIN[360 - a] >> (30 - SHIFT));
M58_B16_E0'B'{}<-[M58_B15_E0 M58_B16_E0]156:   getstatic[178]          14              int[] util.FixPoint.SIN
    159:   sipush[17]            3
    162:   iload_0[26]            1              I:a
    163:   isub[100]             1
    164:   iaload[46]            36             I
    165:   bipush[16]            2
    167:   iload_1[27]            1              I:SHIFT
    168:   isub[100]             1
    169:   ishr[122]             1
    170:   ineg[116]             4
    171:   istore_2[61]           1              ->I:Temp1
    172:   goto[167]->186:          4              sum(B16):    69
 Src. line 203: Temp1 = (SIN[a] >> (30 - SHIFT));
M58_B17_E0'B'{}<-[M58_B15_E0 M58_B14_E0]175:   getstatic[178]          14              int[] util.FixPoint.SIN
    178:   iload_0[26]            1              I:a
    179:   iaload[46]            36             I
    180:   bipush[16]            2
    182:   iload_1[27]            1              I:SHIFT
    183:   isub[100]             1
    184:   ishr[122]             1
    185:   istore_2[61]           1              ->I:Temp1 sum(B17):    57
 Src. line 205: if (90 <= b && b <= 180)
M58_B18_E0'B'{}<-[M58_B16_E0 M58_B17_E0 M58_B13_E0 M58_B10_E0]186:   bipush[16]            2
    188:   iload[21]             2              I:b
    190:   if_icmpgt[163]->220:       4              sum(B18):    8
 Src. line 205: if (90 <= b && b <= 180)
M58_B19_E0'B'{}<-[M58_B19_E0 M58_B18_E0]193:   iload[21]             2              I:b
    195:   sipush[17]            3
    198:   if_icmpgt[163]->220:       4              sum(B19):    9
 Src. line 206: Temp2 = (SIN[180 - b] >> (30 - SHIFT));
M58_B20_E0'B'{}<-[M58_B19_E0 M58_B20_E0]201:   getstatic[178]          14              int[] util.FixPoint.SIN
    204:   sipush[17]            3
    207:   iload[21]             2              I:b
    209:   isub[100]             1
    210:   iaload[46]            36             I
    211:   bipush[16]            2
    213:   iload_1[27]            1              I:SHIFT
```

```
    214:  isub[100]                1
    215:  ishr[122]                1
    216:  istore_3[62]             1                ->I:Temp2
    217:  goto[167]->304:          4                sum(B20):    66
  Src. line 207: else if (180 <= b && b <= 270)
M58_B21_E0'B'{}<-[M58_B19_E0 M58_B18_E0]220:  sipush[17]              3
    223:  iload[21]                2                I:b
    225:  if_icmpgt[163]->256:     4                sum(B21):    9
  Src. line 207: else if (180 <= b && b <= 270)
M58_B22_E0'B'{}<-[M58_B21_E0 M58_B22_E0]228:  iload[21]               2                I:b
    230:  sipush[17]               3
    233:  if_icmpgt[163]->256:     4                sum(B22):    9
  Src. line 208: Temp2 = -(SIN[b - 180] >> (30 - SHIFT));
M58_B23_E0'B'{}<-[M58_B22_E0 M58_B23_E0]236:  getstatic[178]           14               int[] util.FixPoint.SIN
    239:  iload[21]                2                I:b
    241:  sipush[17]               3
    244:  isub[100]                1
    245:  iaload[46]               36               I
    246:  bipush[16]               2
    248:  iload_1[27]              1                I:SHIFT
    249:  isub[100]                1
    250:  ishr[122]                1
    251:  ineg[116]                4
    252:  istore_3[62]             1                ->I:Temp2
    253:  goto[167]->304:          4                sum(B23):    70
  Src. line 209: else if (270 <= b && b <= 360)
M58_B24_E0'B'{}<-[M58_B21_E0 M58_B22_E0]256:  sipush[17]              3
    259:  iload[21]                2                I:b
    261:  if_icmpgt[163]->292:     4                sum(B24):    9
  Src. line 209: else if (270 <= b && b <= 360)
M58_B25_E0'B'{}<-[M58_B25_E0 M58_B24_E0]264:  iload[21]               2                I:b
    266:  sipush[17]               3
    269:  if_icmpgt[163]->292:     4                sum(B25):    9
  Src. line 210: Temp2 = -(SIN[360 - b] >> (30 - SHIFT));
M58_B26_E0'B'{}<-[M58_B25_E0 M58_B26_E0]272:  getstatic[178]           14               int[] util.FixPoint.SIN
    275:  sipush[17]               3
    278:  iload[21]                2                I:b
    280:  isub[100]                1
    281:  iaload[46]               36               I
    282:  bipush[16]               2
    284:  iload_1[27]              1                I:SHIFT
    285:  isub[100]                1
    286:  ishr[122]                1
    287:  ineg[116]                4
    288:  istore_3[62]             1                ->I:Temp2
    289:  goto[167]->304:          4                sum(B26):    70
  Src. line 212: Temp2 = (SIN[b] >> (30 - SHIFT));
M58_B27_E0'B'{}<-[M58_B25_E0 M58_B24_E0]292:  getstatic[178]           14               int[] util.FixPoint.SIN
    295:  iload[21]                2                I:b
    297:  iaload[46]               36               I
    298:  bipush[16]               2
    300:  iload_1[27]              1                I:SHIFT
    301:  isub[100]                1
    302:  ishr[122]                1
    303:  istore_3[62]             1                ->I:Temp2 sum(B27):    58
  Src. line 215: return (Temp1 + Mul(Temp2 - Temp1, Rest, SHIFT));
M58_B28_E0'B'{}<-[M58_B26_E0 M58_B27_E0 M58_B20_E0 M58_B23_E0]304:  iload_2[28]               1
I:Temp1
```

```
    305:   iload_3[29]              1              I:Temp2
    306:   iload_2[28]              1              I:Temp1
    307:   isub[100]               1
    308:   iload[21]              2              I:Rest
    310:   iload_1[27]              1              I:SHIFT sum(B28):     7
  Src. line 215: return (Temp1 + Mul(Temp2 - Temp1, Rest, SHIFT));
M58_I29_E0'I'{}<-[M58_I29_E0 M58_B28_E0]311:   invokestatic[184]          75      5     156   util.FixPoint.Mul(III)I,
invoke(n=17):75/80 return(n=79):23/179sum(B29):     75
  Src. line 215: return (Temp1 + Mul(Temp2 - Temp1, Rest, SHIFT));
M58_B30_E0'B'{}<-[M58_I29_E0 M58_B30_E0]314:   iadd[96]                1
    315:   ireturn[172]            23              sum(B30):     24
M58_T_E0'T'<-[M58_B30_E0]
================================================================================
Info: n=79 a=-1 r=1 w=1

/***WCET calculation source***/
/* WCA WCET objective: util.FixPoint.sin */
max: tM58_S_E0 tM58_B1_E0 tM58_I2_E0 tM58_B3_E0 tM58_B4_E0 tM58_B5_E0 tM58_B6_E0 tM58_B7_E0
tM58_B8_E0 tM58_B9_E0 tM58_B10_E0 tM58_B11_E0 tM58_B12_E0 tM58_B13_E0 tM58_B14_E0 tM58_B15_E0
tM58_B16_E0 tM58_B17_E0 tM58_B18_E0 tM58_B19_E0 tM58_B20_E0 tM58_B21_E0 tM58_B22_E0 tM58_B23_E0
tM58_B24_E0 tM58_B25_E0 tM58_B26_E0 tM58_B27_E0 tM58_B28_E0 tM58_I29_E0 tM58_B30_E0 tM58_T_E0;
/* WCA flow constraints: sin : M58 */
M58_S_E0: 1 = fM58_S_E0_M58_B1_E0; // S flow
M58_B1_E0: fM58_S_E0_M58_B1_E0 + fM58_B1_E0_M58_B1_E0 = fM58_B1_E0_M58_I2_E0;
M58_I2_E0: fM58_I2_E0_M58_I2_E0 + fM58_B1_E0_M58_I2_E0 = fM58_I2_E0_M58_B3_E0;
M58_B3_E0: fM58_I2_E0_M58_B3_E0 + fM58_B3_E0_M58_B3_E0 = fM58_B3_E0_M58_B4_E0 +
fM58_B3_E0_M58_B5_E0;
M58_B4_E0: fM58_B4_E0_M58_B4_E0 + fM58_B3_E0_M58_B4_E0 = fM58_B4_E0_M58_B6_E0;
M58_B5_E0: fM58_B3_E0_M58_B5_E0 = fM58_B5_E0_M58_B6_E0;
M58_B6_E0: fM58_B5_E0_M58_B6_E0 + fM58_B4_E0_M58_B6_E0 = fM58_B6_E0_M58_B7_E0 +
fM58_B6_E0_M58_B8_E0;
M58_B7_E0: fM58_B7_E0_M58_B7_E0 + fM58_B6_E0_M58_B7_E0 = fM58_B7_E0_M58_B8_E0;
M58_B8_E0: fM58_B7_E0_M58_B8_E0 + fM58_B6_E0_M58_B8_E0 = fM58_B8_E0_M58_B9_E0 +
fM58_B8_E0_M58_B11_E0;
M58_B9_E0: fM58_B8_E0_M58_B9_E0 + fM58_B9_E0_M58_B9_E0 = fM58_B9_E0_M58_B10_E0 +
fM58_B9_E0_M58_B11_E0;
M58_B10_E0: fM58_B9_E0_M58_B10_E0 + fM58_B10_E0_M58_B10_E0 = fM58_B10_E0_M58_B18_E0;
M58_B11_E0: fM58_B8_E0_M58_B11_E0 + fM58_B9_E0_M58_B11_E0 = fM58_B11_E0_M58_B12_E0 +
fM58_B11_E0_M58_B14_E0;
M58_B12_E0: fM58_B11_E0_M58_B12_E0 + fM58_B12_E0_M58_B12_E0 = fM58_B12_E0_M58_B13_E0 +
fM58_B12_E0_M58_B14_E0;
M58_B13_E0: fM58_B12_E0_M58_B13_E0 + fM58_B13_E0_M58_B13_E0 = fM58_B13_E0_M58_B18_E0;
M58_B14_E0: fM58_B11_E0_M58_B14_E0 + fM58_B12_E0_M58_B14_E0 = fM58_B14_E0_M58_B15_E0 +
fM58_B14_E0_M58_B17_E0;
M58_B15_E0: fM58_B15_E0_M58_B15_E0 + fM58_B14_E0_M58_B15_E0 = fM58_B15_E0_M58_B16_E0 +
fM58_B15_E0_M58_B17_E0;
M58_B16_E0: fM58_B15_E0_M58_B16_E0 + fM58_B16_E0_M58_B16_E0 = fM58_B16_E0_M58_B18_E0;
M58_B17_E0: fM58_B15_E0_M58_B17_E0 + fM58_B14_E0_M58_B17_E0 = fM58_B17_E0_M58_B18_E0;
M58_B18_E0: fM58_B16_E0_M58_B18_E0 + fM58_B17_E0_M58_B18_E0 + fM58_B13_E0_M58_B18_E0 +
fM58_B10_E0_M58_B18_E0 = fM58_B18_E0_M58_B19_E0 + fM58_B18_E0_M58_B21_E0;
M58_B19_E0: fM58_B19_E0_M58_B19_E0 + fM58_B18_E0_M58_B19_E0 = fM58_B19_E0_M58_B20_E0 +
fM58_B19_E0_M58_B21_E0;
M58_B20_E0: fM58_B19_E0_M58_B20_E0 + fM58_B20_E0_M58_B20_E0 = fM58_B20_E0_M58_B28_E0;
M58_B21_E0: fM58_B19_E0_M58_B21_E0 + fM58_B18_E0_M58_B21_E0 = fM58_B21_E0_M58_B22_E0 +
fM58_B21_E0_M58_B24_E0;
M58_B22_E0: fM58_B21_E0_M58_B22_E0 + fM58_B22_E0_M58_B22_E0 = fM58_B22_E0_M58_B23_E0 +
fM58_B22_E0_M58_B24_E0;
M58_B23_E0: fM58_B22_E0_M58_B23_E0 + fM58_B23_E0_M58_B23_E0 = fM58_B23_E0_M58_B28_E0;
```

M58_B24_E0: fM58_B21_E0_M58_B24_E0 + fM58_B22_E0_M58_B24_E0 = fM58_B24_E0_M58_B25_E0 + fM58_B24_E0_M58_B27_E0;
M58_B25_E0: fM58_B25_E0_M58_B25_E0 + fM58_B24_E0_M58_B25_E0 = fM58_B25_E0_M58_B26_E0 + fM58_B25_E0_M58_B27_E0;
M58_B26_E0: fM58_B25_E0_M58_B26_E0 + fM58_B26_E0_M58_B26_E0 = fM58_B26_E0_M58_B28_E0;
M58_B27_E0: fM58_B25_E0_M58_B27_E0 + fM58_B24_E0_M58_B27_E0 = fM58_B27_E0_M58_B28_E0;
M58_B28_E0: fM58_B26_E0_M58_B28_E0 + fM58_B27_E0_M58_B28_E0 + fM58_B20_E0_M58_B28_E0 + fM58_B23_E0_M58_B28_E0 = fM58_B28_E0_M58_I29_E0;
M58_I29_E0: fM58_I29_E0_M58_I29_E0 + fM58_B28_E0_M58_I29_E0 = fM58_I29_E0_M58_B30_E0;
M58_B30_E0: fM58_I29_E0_M58_B30_E0 + fM58_B30_E0_M58_B30_E0 = fM58_B30_E0_M58_T_E0;
M58_T_E0: fM58_B30_E0_M58_T_E0 = 1; // T flow
/* WCA flow to cycle count */
tM58_B1_E0 = 69 fM58_S_E0_M58_B1_E0 + 69 fM58_B1_E0_M58_B1_E0;
tM58_I2_E0 = 75 fM58_I2_E0_M58_I2_E0 + 75 fM58_B1_E0_M58_I2_E0;
tM58_B3_E0 = 1955 fM58_I2_E0_M58_B3_E0 + 1955 fM58_B3_E0_M58_B3_E0;
tM58_B4_E0 = 14 fM58_B4_E0_M58_B4_E0 + 14 fM58_B3_E0_M58_B4_E0;
tM58_B5_E0 = 8 fM58_B3_E0_M58_B5_E0;
tM58_B6_E0 = 5 fM58_B5_E0_M58_B6_E0 + 5 fM58_B4_E0_M58_B6_E0;
tM58_B7_E0 = 6 fM58_B7_E0_M58_B7_E0 + 6 fM58_B6_E0_M58_B7_E0;
tM58_B8_E0 = 12 fM58_B7_E0_M58_B8_E0 + 12 fM58_B6_E0_M58_B8_E0;
tM58_B9_E0 = 8 fM58_B8_E0_M58_B9_E0 + 8 fM58_B9_E0_M58_B9_E0;
tM58_B10_E0 = 65 fM58_B9_E0_M58_B10_E0 + 65 fM58_B10_E0_M58_B10_E0;
tM58_B11_E0 = 8 fM58_B8_E0_M58_B11_E0 + 8 fM58_B9_E0_M58_B11_E0;
tM58_B12_E0 = 8 fM58_B11_E0_M58_B12_E0 + 8 fM58_B12_E0_M58_B12_E0;
tM58_B13_E0 = 69 fM58_B12_E0_M58_B13_E0 + 69 fM58_B13_E0_M58_B13_E0;
tM58_B14_E0 = 8 fM58_B11_E0_M58_B14_E0 + 8 fM58_B12_E0_M58_B14_E0;
tM58_B15_E0 = 8 fM58_B15_E0_M58_B15_E0 + 8 fM58_B14_E0_M58_B15_E0;
tM58_B16_E0 = 69 fM58_B15_E0_M58_B16_E0 + 69 fM58_B16_E0_M58_B16_E0;
tM58_B17_E0 = 57 fM58_B15_E0_M58_B17_E0 + 57 fM58_B14_E0_M58_B17_E0;
tM58_B18_E0 = 8 fM58_B16_E0_M58_B18_E0 + 8 fM58_B17_E0_M58_B18_E0 + 8 fM58_B13_E0_M58_B18_E0 + 8 fM58_B10_E0_M58_B18_E0;
tM58_B19_E0 = 9 fM58_B19_E0_M58_B19_E0 + 9 fM58_B18_E0_M58_B19_E0;
tM58_B20_E0 = 66 fM58_B19_E0_M58_B20_E0 + 66 fM58_B20_E0_M58_B20_E0;
tM58_B21_E0 = 9 fM58_B19_E0_M58_B21_E0 + 9 fM58_B18_E0_M58_B21_E0;
tM58_B22_E0 = 9 fM58_B21_E0_M58_B22_E0 + 9 fM58_B22_E0_M58_B22_E0;
tM58_B23_E0 = 70 fM58_B22_E0_M58_B23_E0 + 70 fM58_B23_E0_M58_B23_E0;
tM58_B24_E0 = 9 fM58_B21_E0_M58_B24_E0 + 9 fM58_B22_E0_M58_B24_E0;
tM58_B25_E0 = 9 fM58_B25_E0_M58_B25_E0 + 9 fM58_B24_E0_M58_B25_E0;
tM58_B26_E0 = 70 fM58_B25_E0_M58_B26_E0 + 70 fM58_B26_E0_M58_B26_E0;
tM58_B27_E0 = 58 fM58_B25_E0_M58_B27_E0 + 58 fM58_B24_E0_M58_B27_E0;
tM58_B28_E0 = 7 fM58_B26_E0_M58_B28_E0 + 7 fM58_B27_E0_M58_B28_E0 + 7 fM58_B20_E0_M58_B28_E0 + 7 fM58_B23_E0_M58_B28_E0;
tM58_I29_E0 = 75 fM58_I29_E0_M58_I29_E0 + 75 fM58_B28_E0_M58_I29_E0;
tM58_B30_E0 = 24 fM58_I29_E0_M58_B30_E0 + 24 fM58_B30_E0_M58_B30_E0;
/* Invocation(s) from sin:[M58] */

Value of objective function: 2474

Actual values of the variables:

tM58_S_E0	0
tM58_B1_E0	69
tM58_I2_E0	75
tM58_B3_E0	1955
tM58_B4_E0	14
tM58_B5_E0	0
tM58_B6_E0	5
tM58_B7_E0	6
tM58_B8_E0	12

tM58_B9_E0 8
tM58_B10_E0 0
tM58_B11_E0 8
tM58_B12_E0 8
tM58_B13_E0 0
tM58_B14_E0 8
tM58_B15_E0 8
tM58_B16_E0 69
tM58_B17_E0 0
tM58_B18_E0 8
tM58_B19_E0 9
tM58_B20_E0 0
tM58_B21_E0 9
tM58_B22_E0 9
tM58_B23_E0 0
tM58_B24_E0 9
tM58_B25_E0 9
tM58_B26_E0 70
tM58_B27_E0 0
tM58_B28_E0 7
tM58_I29_E0 75
tM58_B30_E0 24
tM58_T_E0 0
fM58_S_E0_M58_B1_E0 1
fM58_B1_E0_M58_B1_E0 0
fM58_B1_E0_M58_I2_E0 1
fM58_I2_E0_M58_I2_E0 0
fM58_I2_E0_M58_B3_E0 1
fM58_B3_E0_M58_B3_E0 0
fM58_B3_E0_M58_B4_E0 1
fM58_B3_E0_M58_B5_E0 0
fM58_B4_E0_M58_B4_E0 0
fM58_B4_E0_M58_B6_E0 1
fM58_B5_E0_M58_B6_E0 0
fM58_B6_E0_M58_B7_E0 1
fM58_B6_E0_M58_B8_E0 0
fM58_B7_E0_M58_B7_E0 0
fM58_B7_E0_M58_B8_E0 1
fM58_B8_E0_M58_B9_E0 1
fM58_B8_E0_M58_B11_E0 0
fM58_B9_E0_M58_B9_E0 0
fM58_B9_E0_M58_B10_E0 0
fM58_B9_E0_M58_B11_E0 1
fM58_B10_E0_M58_B10_E0 0
fM58_B10_E0_M58_B18_E0 0
fM58_B11_E0_M58_B12_E0 1
fM58_B11_E0_M58_B14_E0 0
fM58_B12_E0_M58_B12_E0 0
fM58_B12_E0_M58_B13_E0 0
fM58_B12_E0_M58_B14_E0 1
fM58_B13_E0_M58_B13_E0 0
fM58_B13_E0_M58_B18_E0 0
fM58_B14_E0_M58_B15_E0 1
fM58_B14_E0_M58_B17_E0 0
fM58_B15_E0_M58_B15_E0 0
fM58_B15_E0_M58_B16_E0 1
fM58_B15_E0_M58_B17_E0 0
fM58_B16_E0_M58_B16_E0 0

```
fM58_B16_E0_M58_B18_E0          1
fM58_B17_E0_M58_B18_E0          0
fM58_B18_E0_M58_B19_E0          1
fM58_B18_E0_M58_B21_E0          0
fM58_B19_E0_M58_B19_E0          0
fM58_B19_E0_M58_B20_E0          0
fM58_B19_E0_M58_B21_E0          1
fM58_B20_E0_M58_B20_E0          0
fM58_B20_E0_M58_B28_E0          0
fM58_B21_E0_M58_B22_E0          1
fM58_B21_E0_M58_B24_E0          0
fM58_B22_E0_M58_B22_E0          0
fM58_B22_E0_M58_B23_E0          0
fM58_B22_E0_M58_B24_E0          1
fM58_B23_E0_M58_B23_E0          0
fM58_B23_E0_M58_B28_E0          0
fM58_B24_E0_M58_B25_E0          1
fM58_B24_E0_M58_B27_E0          0
fM58_B25_E0_M58_B25_E0          0
fM58_B25_E0_M58_B26_E0          1
fM58_B25_E0_M58_B27_E0          0
fM58_B26_E0_M58_B26_E0          0
fM58_B26_E0_M58_B28_E0          1
fM58_B27_E0_M58_B28_E0          0
fM58_B28_E0_M58_I29_E0          1
fM58_I29_E0_M58_I29_E0          0
fM58_I29_E0_M58_B30_E0          1
fM58_B30_E0_M58_B30_E0          0
fM58_B30_E0_M58_T_E0            1

/*util.FixPoint.sin(II)I*/
digraph G {
size = "10,7.5"
        M58_S_E0->M58_B1_E0 [label="fM58_S_E0_M58_B1_E0=1"];
        M58_B1_E0->M58_I2_E0 [label="fM58_B1_E0_M58_I2_E0=1"];
        M58_I2_E0->M58_B3_E0 [label="fM58_I2_E0_M58_B3_E0=1"];
        M58_B3_E0->M58_B4_E0 [label="fM58_B3_E0_M58_B4_E0=1"];
        M58_B3_E0->M58_B5_E0 [style=dashed,label="fM58_B3_E0_M58_B5_E0=0"];
        M58_B4_E0->M58_B6_E0 [label="fM58_B4_E0_M58_B6_E0=1"];
        M58_B5_E0->M58_B6_E0 [style=dashed,label="fM58_B5_E0_M58_B6_E0=0"];
        M58_B6_E0->M58_B7_E0 [label="fM58_B6_E0_M58_B7_E0=1"];
        M58_B6_E0->M58_B8_E0 [style=dashed,label="fM58_B6_E0_M58_B8_E0=0"];
        M58_B7_E0->M58_B8_E0 [label="fM58_B7_E0_M58_B8_E0=1"];
        M58_B8_E0->M58_B9_E0 [label="fM58_B8_E0_M58_B9_E0=1"];
        M58_B8_E0->M58_B11_E0 [style=dashed,label="fM58_B8_E0_M58_B11_E0=0"];
        M58_B9_E0->M58_B10_E0 [style=dashed,label="fM58_B9_E0_M58_B10_E0=0"];
        M58_B9_E0->M58_B11_E0 [label="fM58_B9_E0_M58_B11_E0=1"];
        M58_B10_E0->M58_B18_E0 [style=dashed,label="fM58_B10_E0_M58_B18_E0=0"];
        M58_B11_E0->M58_B12_E0 [label="fM58_B11_E0_M58_B12_E0=1"];
        M58_B11_E0->M58_B14_E0 [style=dashed,label="fM58_B11_E0_M58_B14_E0=0"];
        M58_B12_E0->M58_B13_E0 [style=dashed,label="fM58_B12_E0_M58_B13_E0=0"];
        M58_B12_E0->M58_B14_E0 [label="fM58_B12_E0_M58_B14_E0=1"];
        M58_B13_E0->M58_B18_E0 [style=dashed,label="fM58_B13_E0_M58_B18_E0=0"];
        M58_B14_E0->M58_B15_E0 [label="fM58_B14_E0_M58_B15_E0=1"];
        M58_B14_E0->M58_B17_E0 [style=dashed,label="fM58_B14_E0_M58_B17_E0=0"];
        M58_B15_E0->M58_B16_E0 [label="fM58_B15_E0_M58_B16_E0=1"];
        M58_B15_E0->M58_B17_E0 [style=dashed,label="fM58_B15_E0_M58_B17_E0=0"];
        M58_B16_E0->M58_B18_E0 [label="fM58_B16_E0_M58_B18_E0=1"];
```

```
        M58_B17_E0->M58_B18_E0 [style=dashed,label="fM58_B17_E0_M58_B18_E0=0"];
        M58_B18_E0->M58_B19_E0 [label="fM58_B18_E0_M58_B19_E0=1"];
        M58_B18_E0->M58_B21_E0 [style=dashed,label="fM58_B18_E0_M58_B21_E0=0"];
        M58_B19_E0->M58_B20_E0 [style=dashed,label="fM58_B19_E0_M58_B20_E0=0"];
        M58_B19_E0->M58_B21_E0 [label="fM58_B19_E0_M58_B21_E0=1"];
        M58_B20_E0->M58_B28_E0 [style=dashed,label="fM58_B20_E0_M58_B28_E0=0"];
        M58_B21_E0->M58_B22_E0 [label="fM58_B21_E0_M58_B22_E0=1"];
        M58_B21_E0->M58_B24_E0 [style=dashed,label="fM58_B21_E0_M58_B24_E0=0"];
        M58_B22_E0->M58_B23_E0 [style=dashed,label="fM58_B22_E0_M58_B23_E0=0"];
        M58_B22_E0->M58_B24_E0 [label="fM58_B22_E0_M58_B24_E0=1"];
        M58_B23_E0->M58_B28_E0 [style=dashed,label="fM58_B23_E0_M58_B28_E0=0"];
        M58_B24_E0->M58_B25_E0 [label="fM58_B24_E0_M58_B25_E0=1"];
        M58_B24_E0->M58_B27_E0 [style=dashed,label="fM58_B24_E0_M58_B27_E0=0"];
        M58_B25_E0->M58_B26_E0 [label="fM58_B25_E0_M58_B26_E0=1"];
        M58_B25_E0->M58_B27_E0 [style=dashed,label="fM58_B25_E0_M58_B27_E0=0"];
        M58_B26_E0->M58_B28_E0 [label="fM58_B26_E0_M58_B28_E0=1"];
        M58_B27_E0->M58_B28_E0 [style=dashed,label="fM58_B27_E0_M58_B28_E0=0"];
        M58_B28_E0->M58_I29_E0 [label="fM58_B28_E0_M58_I29_E0=1"];
        M58_I29_E0->M58_B30_E0 [label="fM58_I29_E0_M58_B30_E0=1"];
        M58_B30_E0->M58_T_E0 [label="fM58_B30_E0_M58_T_E0=1"];
        M58_S_E0;
        M58_B1_E0 [label="M58_B1_E0\n69"];
        M58_I2_E0 [label="M58_I2_E0\n75"];
        M58_B3_E0 [label="M58_B3_E0\n1955"];
        M58_B4_E0 [label="M58_B4_E0\n14"];
        M58_B5_E0 [label="M58_B5_E0\n8"];
        M58_B6_E0 [label="M58_B6_E0\n5"];
        M58_B7_E0 [label="M58_B7_E0\n6"];
        M58_B8_E0 [label="M58_B8_E0\n12"];
        M58_B9_E0 [label="M58_B9_E0\n8"];
        M58_B10_E0 [label="M58_B10_E0\n65"];
        M58_B11_E0 [label="M58_B11_E0\n8"];
        M58_B12_E0 [label="M58_B12_E0\n8"];
        M58_B13_E0 [label="M58_B13_E0\n69"];
        M58_B14_E0 [label="M58_B14_E0\n8"];
        M58_B15_E0 [label="M58_B15_E0\n8"];
        M58_B16_E0 [label="M58_B16_E0\n69"];
        M58_B17_E0 [label="M58_B17_E0\n57"];
        M58_B18_E0 [label="M58_B18_E0\n8"];
        M58_B19_E0 [label="M58_B19_E0\n9"];
        M58_B20_E0 [label="M58_B20_E0\n66"];
        M58_B21_E0 [label="M58_B21_E0\n9"];
        M58_B22_E0 [label="M58_B22_E0\n9"];
        M58_B23_E0 [label="M58_B23_E0\n70"];
        M58_B24_E0 [label="M58_B24_E0\n9"];
        M58_B25_E0 [label="M58_B25_E0\n9"];
        M58_B26_E0 [label="M58_B26_E0\n70"];
        M58_B27_E0 [label="M58_B27_E0\n58"];
        M58_B28_E0 [label="M58_B28_E0\n7"];
        M58_I29_E0 [label="M58_I29_E0\n75"];
        M58_B30_E0 [label="M58_B30_E0\n24"];
        M58_T_E0;
}
```

****

Note: Remember to keep WCETAnalyzer updated
each time a bytecode implementation is changed.

cos(int,int)

**********APPLICATION WCET=5697**********
/***WCET calculation source***/
/* WCA WCET objective: util.FixPoint.cos */
max: tM59_S_E1 tM59_B1_E1 tM59_I2_E1 tM59_B3_E1 tM59_B4_E1 tM59_B5_E1 tM59_B6_E1 tM59_B7_E1
tM59_B8_E1 tM59_B9_E1 tM59_B10_E1 tM59_B11_E1 tM59_B12_E1 tM59_B13_E1 tM59_B14_E1 tM59_B15_E1
tM59_B16_E1 tM59_B17_E1 tM59_B18_E1 tM59_B19_E1 tM59_B20_E1 tM59_B21_E1 tM59_B22_E1 tM59_B23_E1
tM59_B24_E1 tM59_B25_E1 tM59_B26_E1 tM59_B27_E1 tM59_B28_E1 tM59_B29_E1 tM59_B30_E1 tM59_I31_E1
tM59_B32_E1 tM59_T_E1 tM55_S_E1 tM55_B1_E1 tM55_I2_E1 tM55_B3_E1 tM55_T_E1 tM54_S_E1 tM54_B1_E1
tM54_T_E1 tM54_S_E2 tM54_B1_E2 tM54_T_E2;
/* WCA flow constraints: cos : M59 */
M59_S_E1: 1 = fM59_S_E1_M59_B1_E1; // S flow
M59_B1_E1: fM59_S_E1_M59_B1_E1 + fM59_B1_E1_M59_B1_E1 = fM59_B1_E1_M59_I2_E1;
M59_I2_E1: fM59_I2_E1_M59_I2_E1 + fM59_B1_E1_M59_I2_E1 = fM59_I2_E1_M59_B3_E1;
/* Connecting(invoking) to util.FixPoint.Div(III)I id:M55_S_E1*/
M59_I2_E1_S: fcmM59_I2_E1_M55_S_E1 + fchM59_I2_E1_M55_S_E1 = fM59_I2_E1_M59_B3_E1; //cache S paths
M59_I2_E1_T: fM59_I2_E1_M59_B3_E1 = fM55_T_E1_M59_I2_E1; // invo T return path
fchM59_I2_E1_M55_S_E1 = 0; // no cache hits (because not innerloop)
/* Done with util.FixPoint.Div(III)I*/
M59_B3_E1: fM59_I2_E1_M59_B3_E1 + fM59_B3_E1_M59_B3_E1 = fM59_B3_E1_M59_B4_E1 +
fM59_B3_E1_M59_B5_E1;
M59_B4_E1: fM59_B4_E1_M59_B4_E1 + fM59_B3_E1_M59_B4_E1 = fM59_B4_E1_M59_B8_E1;
M59_B5_E1: fM59_B3_E1_M59_B5_E1 = fM59_B5_E1_M59_B6_E1 + fM59_B5_E1_M59_B7_E1;
M59_B6_E1: fM59_B6_E1_M59_B6_E1 + fM59_B5_E1_M59_B6_E1 = fM59_B6_E1_M59_B8_E1;
M59_B7_E1: fM59_B5_E1_M59_B7_E1 = fM59_B7_E1_M59_B8_E1;
M59_B8_E1: fM59_B4_E1_M59_B8_E1 + fM59_B6_E1_M59_B8_E1 + fM59_B7_E1_M59_B8_E1 =
fM59_B8_E1_M59_B9_E1 + fM59_B8_E1_M59_B10_E1;
M59_B9_E1: fM59_B9_E1_M59_B9_E1 + fM59_B8_E1_M59_B9_E1 = fM59_B9_E1_M59_B10_E1;
M59_B10_E1: fM59_B9_E1_M59_B10_E1 + fM59_B8_E1_M59_B10_E1 = fM59_B10_E1_M59_B11_E1 +
fM59_B10_E1_M59_B13_E1;
M59_B11_E1: fM59_B11_E1_M59_B11_E1 + fM59_B10_E1_M59_B11_E1 = fM59_B11_E1_M59_B12_E1 +
fM59_B11_E1_M59_B13_E1;
M59_B12_E1: fM59_B11_E1_M59_B12_E1 + fM59_B12_E1_M59_B12_E1 = fM59_B12_E1_M59_B20_E1;
M59_B13_E1: fM59_B11_E1_M59_B13_E1 + fM59_B10_E1_M59_B13_E1 = fM59_B13_E1_M59_B14_E1 +
fM59_B13_E1_M59_B16_E1;
M59_B14_E1: fM59_B14_E1_M59_B14_E1 + fM59_B13_E1_M59_B14_E1 = fM59_B14_E1_M59_B15_E1 +
fM59_B14_E1_M59_B16_E1;
M59_B15_E1: fM59_B14_E1_M59_B15_E1 + fM59_B15_E1_M59_B15_E1 = fM59_B15_E1_M59_B20_E1;
M59_B16_E1: fM59_B14_E1_M59_B16_E1 + fM59_B13_E1_M59_B16_E1 = fM59_B16_E1_M59_B17_E1 +
fM59_B16_E1_M59_B19_E1;
M59_B17_E1: fM59_B17_E1_M59_B17_E1 + fM59_B16_E1_M59_B17_E1 = fM59_B17_E1_M59_B18_E1 +
fM59_B17_E1_M59_B19_E1;
M59_B18_E1: fM59_B18_E1_M59_B18_E1 + fM59_B17_E1_M59_B18_E1 = fM59_B18_E1_M59_B20_E1;
M59_B19_E1: fM59_B17_E1_M59_B19_E1 + fM59_B16_E1_M59_B19_E1 = fM59_B19_E1_M59_B20_E1;
M59_B20_E1: fM59_B19_E1_M59_B20_E1 + fM59_B18_E1_M59_B20_E1 + fM59_B12_E1_M59_B20_E1 +
fM59_B15_E1_M59_B20_E1 = fM59_B20_E1_M59_B21_E1 + fM59_B20_E1_M59_B23_E1;
M59_B21_E1: fM59_B20_E1_M59_B21_E1 + fM59_B21_E1_M59_B21_E1 = fM59_B21_E1_M59_B22_E1 +
fM59_B21_E1_M59_B23_E1;
M59_B22_E1: fM59_B22_E1_M59_B22_E1 + fM59_B21_E1_M59_B22_E1 = fM59_B22_E1_M59_B30_E1;
M59_B23_E1: fM59_B20_E1_M59_B23_E1 + fM59_B21_E1_M59_B23_E1 = fM59_B23_E1_M59_B24_E1 +
fM59_B23_E1_M59_B26_E1;
M59_B24_E1: fM59_B24_E1_M59_B24_E1 + fM59_B23_E1_M59_B24_E1 = fM59_B24_E1_M59_B25_E1 +
fM59_B24_E1_M59_B26_E1;
M59_B25_E1: fM59_B25_E1_M59_B25_E1 + fM59_B24_E1_M59_B25_E1 = fM59_B25_E1_M59_B30_E1;
M59_B26_E1: fM59_B24_E1_M59_B26_E1 + fM59_B23_E1_M59_B26_E1 = fM59_B26_E1_M59_B27_E1 +
fM59_B26_E1_M59_B29_E1;
M59_B27_E1: fM59_B27_E1_M59_B27_E1 + fM59_B26_E1_M59_B27_E1 = fM59_B27_E1_M59_B28_E1 +
fM59_B27_E1_M59_B29_E1;
M59_B28_E1: fM59_B27_E1_M59_B28_E1 + fM59_B28_E1_M59_B28_E1 = fM59_B28_E1_M59_B30_E1;

```
M59_B29_E1: fM59_B27_E1_M59_B29_E1 + fM59_B26_E1_M59_B29_E1 = fM59_B29_E1_M59_B30_E1;
M59_B30_E1: fM59_B29_E1_M59_B30_E1 + fM59_B25_E1_M59_B30_E1 + fM59_B28_E1_M59_B30_E1 +
fM59_B22_E1_M59_B30_E1 = fM59_B30_E1_M59_I31_E1;
M59_I31_E1: fM59_I31_E1_M59_I31_E1 + fM59_B30_E1_M59_I31_E1 = fM59_I31_E1_M59_B32_E1;
/* Connecting(invoking) to util.FixPoint.Mul(III)I id:M54_S_E2*/
M59_I31_E1_S: fcmM59_I31_E1_M54_S_E2 + fchM59_I31_E1_M54_S_E2 = fM59_I31_E1_M59_B32_E1; //cache S
paths
M59_I31_E1_T: fM59_I31_E1_M59_B32_E1 = fM54_T_E2_M59_I31_E1; // invo T return path
fchM59_I31_E1_M54_S_E2 = 0; // no cache hits (because not innerloop)
/* Done with util.FixPoint.Mul(III)I*/
M59_B32_E1: fM59_I31_E1_M59_B32_E1 + fM59_B32_E1_M59_B32_E1 = fM59_B32_E1_M59_T_E1;
M59_T_E1: fM59_B32_E1_M59_T_E1 = 1; // T flow
/* WCA flow to cycle count */
tM59_B1_E1 = 69 fM59_S_E1_M59_B1_E1 + 69 fM59_B1_E1_M59_B1_E1;
tM59_I2_E1 = 75 fM59_I2_E1_M59_I2_E1 + 75 fM59_B1_E1_M59_I2_E1;
tM59_B3_E1 = 1960 fM59_I2_E1_M59_B3_E1 + 1960 fM59_B3_E1_M59_B3_E1;
tM59_B4_E1 = 14 fM59_B4_E1_M59_B4_E1 + 14 fM59_B3_E1_M59_B4_E1;
tM59_B5_E1 = 11 fM59_B3_E1_M59_B5_E1;
tM59_B6_E1 = 16 fM59_B6_E1_M59_B6_E1 + 16 fM59_B5_E1_M59_B6_E1;
tM59_B7_E1 = 8 fM59_B5_E1_M59_B7_E1;
tM59_B8_E1 = 5 fM59_B4_E1_M59_B8_E1 + 5 fM59_B6_E1_M59_B8_E1 + 5 fM59_B7_E1_M59_B8_E1;
tM59_B9_E1 = 6 fM59_B9_E1_M59_B9_E1 + 6 fM59_B8_E1_M59_B9_E1;
tM59_B10_E1 = 12 fM59_B9_E1_M59_B10_E1 + 12 fM59_B8_E1_M59_B10_E1;
tM59_B11_E1 = 8 fM59_B11_E1_M59_B11_E1 + 8 fM59_B10_E1_M59_B11_E1;
tM59_B12_E1 = 65 fM59_B11_E1_M59_B12_E1 + 65 fM59_B12_E1_M59_B12_E1;
tM59_B13_E1 = 8 fM59_B11_E1_M59_B13_E1 + 8 fM59_B10_E1_M59_B13_E1;
tM59_B14_E1 = 8 fM59_B14_E1_M59_B14_E1 + 8 fM59_B13_E1_M59_B14_E1;
tM59_B15_E1 = 69 fM59_B14_E1_M59_B15_E1 + 69 fM59_B15_E1_M59_B15_E1;
tM59_B16_E1 = 8 fM59_B14_E1_M59_B16_E1 + 8 fM59_B13_E1_M59_B16_E1;
tM59_B17_E1 = 8 fM59_B17_E1_M59_B17_E1 + 8 fM59_B16_E1_M59_B17_E1;
tM59_B18_E1 = 69 fM59_B18_E1_M59_B18_E1 + 69 fM59_B17_E1_M59_B18_E1;
tM59_B19_E1 = 57 fM59_B17_E1_M59_B19_E1 + 57 fM59_B16_E1_M59_B19_E1;
tM59_B20_E1 = 8 fM59_B19_E1_M59_B20_E1 + 8 fM59_B18_E1_M59_B20_E1 + 8 fM59_B12_E1_M59_B20_E1 + 8
fM59_B15_E1_M59_B20_E1;
tM59_B21_E1 = 9 fM59_B20_E1_M59_B21_E1 + 9 fM59_B21_E1_M59_B21_E1;
tM59_B22_E1 = 66 fM59_B22_E1_M59_B22_E1 + 66 fM59_B21_E1_M59_B22_E1;
tM59_B23_E1 = 9 fM59_B20_E1_M59_B23_E1 + 9 fM59_B21_E1_M59_B23_E1;
tM59_B24_E1 = 9 fM59_B24_E1_M59_B24_E1 + 9 fM59_B23_E1_M59_B24_E1;
tM59_B25_E1 = 70 fM59_B25_E1_M59_B25_E1 + 70 fM59_B24_E1_M59_B25_E1;
tM59_B26_E1 = 9 fM59_B24_E1_M59_B26_E1 + 9 fM59_B23_E1_M59_B26_E1;
tM59_B27_E1 = 9 fM59_B27_E1_M59_B27_E1 + 9 fM59_B26_E1_M59_B27_E1;
tM59_B28_E1 = 70 fM59_B27_E1_M59_B28_E1 + 70 fM59_B28_E1_M59_B28_E1;
tM59_B29_E1 = 58 fM59_B27_E1_M59_B29_E1 + 58 fM59_B26_E1_M59_B29_E1;
tM59_B30_E1 = 7 fM59_B29_E1_M59_B30_E1 + 7 fM59_B25_E1_M59_B30_E1 + 7 fM59_B28_E1_M59_B30_E1 + 7
fM59_B22_E1_M59_B30_E1;
tM59_I31_E1 = 75 fM59_I31_E1_M59_I31_E1 + 75 fM59_B30_E1_M59_I31_E1;
tM59_B32_E1 = 24 fM59_I31_E1_M59_B32_E1 + 24 fM59_B32_E1_M59_B32_E1;
/* Invocation(s) from cos:[M59] */
/* WCA flow constraints: Div : M55 */
M55_S_E1: fchM59_I2_E1_M55_S_E1+ fcmM59_I2_E1_M55_S_E1 = fM55_S_E1_M55_B1_E1; // S flow
// tM55_S_E1 = 0 fcmM59_I2_E1_M55_S_E1; // S cache miss time
M55_B1_E1: fM55_S_E1_M55_B1_E1 + fM55_B1_E1_M55_B1_E1 = fM55_B1_E1_M55_I2_E1;
M55_I2_E1: fM55_I2_E1_M55_I2_E1 + fM55_B1_E1_M55_I2_E1 = fM55_I2_E1_M55_B3_E1;
/* Connecting(invoking) to util.FixPoint.Mul(III)I id:M54_S_E1*/
M55_I2_E1_S: fcmM55_I2_E1_M54_S_E1 + fchM55_I2_E1_M54_S_E1 = fM55_I2_E1_M55_B3_E1; //cache S paths
M55_I2_E1_T: fM55_I2_E1_M55_B3_E1 = fM54_T_E1_M55_I2_E1; // invo T return path
fchM55_I2_E1_M54_S_E1 = 0; // no cache hits (because not innerloop)
/* Done with util.FixPoint.Mul(III)I*/
```

```
M55_B3_E1: fM55_I2_E1_M55_B3_E1 + fM55_B3_E1_M55_B3_E1 = fM55_B3_E1_M55_T_E1;
M55_T_E1: fM55_B3_E1_M55_T_E1 = fM55_T_E1_M59_I2_E1;// T interconnect flow
tM55_T_E1 = 172 fM55_T_E1_M59_I2_E1; // T cache miss (not leaf)
/* WCA flow to cycle count */
tM55_B1_E1 = 2473 fM55_S_E1_M55_B1_E1 + 2473 fM55_B1_E1_M55_B1_E1;
tM55_I2_E1 = 75 fM55_I2_E1_M55_I2_E1 + 75 fM55_B1_E1_M55_I2_E1;
tM55_B3_E1 = 23 fM55_I2_E1_M55_B3_E1 + 23 fM55_B3_E1_M55_B3_E1;
/* Invocation(s) from Div:[M55] */
/* WCA flow constraints: Mul : M54 */
M54_S_E1: fchM55_I2_E1_M54_S_E1+ fcmM55_I2_E1_M54_S_E1 = fM54_S_E1_M54_B1_E1; // S flow
tM54_S_E1 = 5 fcmM55_I2_E1_M54_S_E1; // S cache miss time
M54_B1_E1: fM54_S_E1_M54_B1_E1 + fM54_B1_E1_M54_B1_E1 = fM54_B1_E1_M54_T_E1;
M54_T_E1: fM54_B1_E1_M54_T_E1 = fM54_T_E1_M55_I2_E1;// T interconnect flow
// tM54_T_E1 = 0 fM54_T_E1_M55_I2_E1; // T cache hit (leaf)
/* WCA flow to cycle count */
tM54_B1_E1 = 226 fM54_S_E1_M54_B1_E1 + 226 fM54_B1_E1_M54_B1_E1;
/* Invocation(s) from Mul:[M54] */
/* WCA flow constraints: Mul : M54 */
M54_S_E2: fchM59_I31_E1_M54_S_E2+ fcmM59_I31_E1_M54_S_E2 = fM54_S_E2_M54_B1_E2; // S flow
tM54_S_E2 = 5 fcmM59_I31_E1_M54_S_E2; // S cache miss time
M54_B1_E2: fM54_S_E2_M54_B1_E2 + fM54_B1_E2_M54_B1_E2 = fM54_B1_E2_M54_T_E2;
M54_T_E2: fM54_B1_E2_M54_T_E2 = fM54_T_E2_M59_I31_E1;// T interconnect flow
// tM54_T_E2 = 0 fM54_T_E2_M59_I31_E1; // T cache hit (leaf)
/* WCA flow to cycle count */
tM54_B1_E2 = 226 fM54_S_E2_M54_B1_E2 + 226 fM54_B1_E2_M54_B1_E2;
/* Invocation(s) from Mul:[M54] */

Value of objective function: 5697

Actual values of the variables:
tM59_S_E1                0
tM59_B1_E1              69
tM59_I2_E1              75
tM59_B3_E1            1960
tM59_B4_E1               0
tM59_B5_E1              11
tM59_B6_E1              16
tM59_B7_E1               0
tM59_B8_E1               5
tM59_B9_E1               6
tM59_B10_E1             12
tM59_B11_E1              8
tM59_B12_E1              0
tM59_B13_E1              8
tM59_B14_E1              8
tM59_B15_E1              0
tM59_B16_E1              8
tM59_B17_E1              8
tM59_B18_E1             69
tM59_B19_E1              0
tM59_B20_E1              8
tM59_B21_E1              9
tM59_B22_E1              0
tM59_B23_E1              9
tM59_B24_E1              9
tM59_B25_E1              0
tM59_B26_E1              9
tM59_B27_E1              9
```

tM59_B28_E1 70
tM59_B29_E1 0
tM59_B30_E1 7
tM59_I31_E1 75
tM59_B32_E1 24
tM59_T_E1 0
tM55_S_E1 0
tM55_B1_E1 2473
tM55_I2_E1 75
tM55_B3_E1 23
tM55_T_E1 172
tM54_S_E1 5
tM54_B1_E1 226
tM54_T_E1 0
tM54_S_E2 5
tM54_B1_E2 226
tM54_T_E2 0
fM59_S_E1_M59_B1_E1 1
fM59_B1_E1_M59_B1_E1 0
fM59_B1_E1_M59_I2_E1 1
fM59_I2_E1_M59_I2_E1 0
fM59_I2_E1_M59_B3_E1 1
fcmM59_I2_E1_M55_S_E1 1
fchM59_I2_E1_M55_S_E1 0
fM55_T_E1_M59_I2_E1 1
fM59_B3_E1_M59_B3_E1 0
fM59_B3_E1_M59_B4_E1 0
fM59_B3_E1_M59_B5_E1 1
fM59_B4_E1_M59_B4_E1 0
fM59_B4_E1_M59_B8_E1 0
fM59_B5_E1_M59_B6_E1 1
fM59_B5_E1_M59_B7_E1 0
fM59_B6_E1_M59_B6_E1 0
fM59_B6_E1_M59_B8_E1 1
fM59_B7_E1_M59_B8_E1 0
fM59_B8_E1_M59_B9_E1 1
fM59_B8_E1_M59_B10_E1 0
fM59_B9_E1_M59_B9_E1 0
fM59_B9_E1_M59_B10_E1 1
fM59_B10_E1_M59_B11_E1 1
fM59_B10_E1_M59_B13_E1 0
fM59_B11_E1_M59_B11_E1 0
fM59_B11_E1_M59_B12_E1 0
fM59_B11_E1_M59_B13_E1 1
fM59_B12_E1_M59_B12_E1 0
fM59_B12_E1_M59_B20_E1 0
fM59_B13_E1_M59_B14_E1 1
fM59_B13_E1_M59_B16_E1 0
fM59_B14_E1_M59_B14_E1 0
fM59_B14_E1_M59_B15_E1 0
fM59_B14_E1_M59_B16_E1 1
fM59_B15_E1_M59_B15_E1 0
fM59_B15_E1_M59_B20_E1 0
fM59_B16_E1_M59_B17_E1 1
fM59_B16_E1_M59_B19_E1 0
fM59_B17_E1_M59_B17_E1 0
fM59_B17_E1_M59_B18_E1 1
fM59_B17_E1_M59_B19_E1 0

fM59_B18_E1_M59_B18_E1 0
fM59_B18_E1_M59_B20_E1 1
fM59_B19_E1_M59_B20_E1 0
fM59_B20_E1_M59_B21_E1 1
fM59_B20_E1_M59_B23_E1 0
fM59_B21_E1_M59_B21_E1 0
fM59_B21_E1_M59_B22_E1 0
fM59_B21_E1_M59_B23_E1 1
fM59_B22_E1_M59_B22_E1 0
fM59_B22_E1_M59_B30_E1 0
fM59_B23_E1_M59_B24_E1 1
fM59_B23_E1_M59_B26_E1 0
fM59_B24_E1_M59_B24_E1 0
fM59_B24_E1_M59_B25_E1 0
fM59_B24_E1_M59_B26_E1 1
fM59_B25_E1_M59_B25_E1 0
fM59_B25_E1_M59_B30_E1 0
fM59_B26_E1_M59_B27_E1 1
fM59_B26_E1_M59_B29_E1 0
fM59_B27_E1_M59_B27_E1 0
fM59_B27_E1_M59_B28_E1 1
fM59_B27_E1_M59_B29_E1 0
fM59_B28_E1_M59_B28_E1 0
fM59_B28_E1_M59_B30_E1 1
fM59_B29_E1_M59_B30_E1 0
fM59_B30_E1_M59_I31_E1 1
fM59_I31_E1_M59_I31_E1 0
fM59_I31_E1_M59_B32_E1 1
fcmM59_I31_E1_M54_S_E2 1
fchM59_I31_E1_M54_S_E2 0
fM54_T_E2_M59_I31_E1 1
fM59_B32_E1_M59_B32_E1 0
fM59_B32_E1_M59_T_E1 1
fM55_S_E1_M55_B1_E1 1
fM55_B1_E1_M55_B1_E1 0
fM55_B1_E1_M55_I2_E1 1
fM55_I2_E1_M55_I2_E1 0
fM55_I2_E1_M55_B3_E1 1
fcmM55_I2_E1_M54_S_E1 1
fchM55_I2_E1_M54_S_E1 0
fM54_T_E1_M55_I2_E1 1
fM55_B3_E1_M55_B3_E1 0
fM55_B3_E1_M55_T_E1 1
fM54_S_E1_M54_B1_E1 1
fM54_B1_E1_M54_B1_E1 0
fM54_B1_E1_M54_T_E1 1
fM54_S_E2_M54_B1_E2 1
fM54_B1_E2_M54_B1_E2 0
fM54_B1_E2_M54_T_E2 1

**********END APPLICATION WCET**********
****
WCET info for:util.FixPoint.Mul(III)I

Directed graph of basic blocks(row->column):
==========
M54_S_E0M54_B1_E0M54_T_E0
M54_S_E0 . 1 .

```
M54_B1_E0 .  .  1
M54_T_E0 .  .  .
============================

Table of basic blocks' and instructions
================================================================================
Block Addr.  Bytecode              Cycles    Cache miss     Misc. info
             [opcode]              invoke  return
--------------------------------------------------------------------------------
M54_S_E0'S'
  Src. line 112: int res = 0;
M54_B1_E0'B'{}<-[M54_S_E0 M54_B1_E0]0:     iconst_0[3]                1
     1:     istore_3[62]             1               ->I:res
  Src. line 113: res = ((f1 >> shift) * (f2 >> shift)) << shift; // AH*BH
     2:     iload_0[26]              1               I:f1
     3:     iload_2[28]              1               I:shift
     4:     ishr[122]               1
     5:     iload_1[27]              1               I:f2
     6:     iload_2[28]              1               I:shift
     7:     ishr[122]               1
     8:     imul[104]               35
     9:     iload_2[28]              1               I:shift
     10:    ishl[120]               1
     11:    istore_3[62]             1               ->I:res
  Src. line 114: res += ((f1 >> shift) * (f2 - ((f2 >> shift) << shift))); // AH*BL
     12:    iload_3[29]              1               I:res
     13:    iload_0[26]              1               I:f1
     14:    iload_2[28]              1               I:shift
     15:    ishr[122]               1
     16:    iload_1[27]              1               I:f2
     17:    iload_1[27]              1               I:f2
     18:    iload_2[28]              1               I:shift
     19:    ishr[122]               1
     20:    iload_2[28]              1               I:shift
     21:    ishl[120]               1
     22:    isub[100]               1
     23:    imul[104]               35
     24:    iadd[96]                1
     25:    istore_3[62]             1               ->I:res
  Src. line 115: res += ((f1 - ((f1 >> shift) << shift)) * (f2 >> shift)); // AL*BH
     26:    iload_3[29]              1               I:res
     27:    iload_0[26]              1               I:f1
     28:    iload_0[26]              1               I:f1
     29:    iload_2[28]              1               I:shift
     30:    ishr[122]               1
     31:    iload_2[28]              1               I:shift
     32:    ishl[120]               1
     33:    isub[100]               1
     34:    iload_1[27]              1               I:f2
     35:    iload_2[28]              1               I:shift
     36:    ishr[122]               1
     37:    imul[104]               35
     38:    iadd[96]                1
     39:    istore_3[62]             1               ->I:res
  Src. line 116: res += (((f1 - ((f1 >> shift) << shift)) >> 1)
     40:    iload_3[29]              1               I:res
     41:    iload_0[26]              1               I:f1
     42:    iload_0[26]              1               I:f1
```

```
    43:   iload_2[28]          1          I:shift
    44:   ishr[122]            1
    45:   iload_2[28]          1          I:shift
    46:   ishl[120]            1
    47:   isub[100]            1
    48:   iconst_1[4]          1
    49:   ishr[122]            1
    50:   iload_1[27]          1          I:f2
    51:   iload_1[27]          1          I:f2
    52:   iload_2[28]          1          I:shift
    53:   ishr[122]            1
    54:   iload_2[28]          1          I:shift
    55:   ishl[120]            1
    56:   isub[100]            1
    57:   imul[104]            35
    58:   iconst_1[4]          1
    59:   ishr[122]            1
    60:   iload_2[28]          1          I:shift
    61:   iconst_2[5]          1
    62:   isub[100]            1
    63:   ishr[122]            1
    64:   iadd[96]             1
    65:   istore_3[62]         1          ->I:res
  Src. line 119: return res;
    66:   iload_3[29]          1          I:res
    67:   ireturn[172]         23          sum(B1):    226
M54_T_E0'T'<-[M54_B1_E0]
================================================================================
Info: n=17 a=-1 r=1 w=1

/***WCET calculation source***/
/* WCA WCET objective: util.FixPoint.Mul */
max: tM54_S_E0 tM54_B1_E0 tM54_T_E0;
/* WCA flow constraints: Mul : M54 */
M54_S_E0: 1 = fM54_S_E0_M54_B1_E0; // S flow
M54_B1_E0: fM54_S_E0_M54_B1_E0 + fM54_B1_E0_M54_B1_E0 = fM54_B1_E0_M54_T_E0;
M54_T_E0: fM54_B1_E0_M54_T_E0 = 1; // T flow
/* WCA flow to cycle count */
tM54_B1_E0 = 226 fM54_S_E0_M54_B1_E0 + 226 fM54_B1_E0_M54_B1_E0;
/* Invocation(s) from Mul:[M54] */

Value of objective function: 226

Actual values of the variables:
tM54_S_E0                  0
tM54_B1_E0                 226
tM54_T_E0                  0
fM54_S_E0_M54_B1_E0        1
fM54_B1_E0_M54_B1_E0       0
fM54_B1_E0_M54_T_E0        1

/*util.FixPoint.Mul(III)I*/
digraph G {
size = "10,7.5"
        M54_S_E0->M54_B1_E0 [label="fM54_S_E0_M54_B1_E0=1"];
        M54_B1_E0->M54_T_E0 [label="fM54_B1_E0_M54_T_E0=1"];
        M54_S_E0;
        M54_B1_E0 [label="M54_B1_E0\n226"];
```

```
          M54_T_E0;
}
********************************************************************************
WCET info for:util.FixPoint.Div(III)I

Directed graph of basic blocks(row->column):
=============================================
   M55_S_E0M55_B1_E0M55_I2_E0M55_B3_E0M55_T_E0
M55_S_E0 .  1  .  .  .
M55_B1_E0 .  .  1  .  .
M55_I2_E0 .  .  .  1  .
M55_B3_E0 .  .  .  .  1
M55_T_E0 .  .  .  .  .
=============================================

Table of basic blocks' and instructions
================================================================================
Block Addr.  Bytecode              Cycles    Cache miss     Misc. info
          [opcode]                 invoke  return
--------------------------------------------------------------------------------
M55_S_E0'S'
  Src. line 136: f2 = (1 << ((shift * 2) - 2)) / ((f2 + 1) >> 2); // approx. 1/f2
M55_B1_E0'B'{}<-[M55_S_E0 M55_B1_E0]0:     iconst_1[4]               1
     1:     iload_2[28]               1               I:shift
     2:     iconst_2[5]               1
     3:     imul[104]                35
     4:     iconst_2[5]               1
     5:     isub[100]                 1
     6:     ishl[120]                 1
     7:     iload_1[27]               1               I:f2
     8:     iconst_1[4]               1
     9:     iadd[96]                  1
     10:    iconst_2[5]               1
     11:    ishr[122]                 1
     12:    idiv[108]              2423
     13:    istore_1[60]              1               ->I:f2
  Src. line 137: return Mul(f1, f2, shift);
     14:    iload_0[26]               1               I:f1
     15:    iload_1[27]               1               I:f2
     16:    iload_2[28]               1               I:shift sum(B1):   2473
  Src. line 137: return Mul(f1, f2, shift);
M55_I2_E0'I'{}<-[M55_I2_E0 M55_B1_E0]17:    invokestatic[184]          75       5      10   util.FixPoint.Mul(III)I,
invoke(n=17):75/80 return(n=6):23/33sum(B2):     75
  Src. line 137: return Mul(f1, f2, shift);
M55_B3_E0'B'{}<-[M55_I2_E0 M55_B3_E0]20:    ireturn[172]              23                sum(B3):     23
M55_T_E0'T'<-[M55_B3_E0]
================================================================================
Info: n=6 a=-1 r=1 w=1

/***WCET calculation source***/
/* WCA WCET objective: util.FixPoint.Div */
max: tM55_S_E0 tM55_B1_E0 tM55_I2_E0 tM55_B3_E0 tM55_T_E0;
/* WCA flow constraints: Div : M55 */
M55_S_E0: 1 = fM55_S_E0_M55_B1_E0; // S flow
M55_B1_E0: fM55_S_E0_M55_B1_E0 + fM55_B1_E0_M55_B1_E0 = fM55_B1_E0_M55_I2_E0;
M55_I2_E0: fM55_I2_E0_M55_I2_E0 + fM55_B1_E0_M55_I2_E0 = fM55_I2_E0_M55_B3_E0;
M55_B3_E0: fM55_I2_E0_M55_B3_E0 + fM55_B3_E0_M55_B3_E0 = fM55_B3_E0_M55_T_E0;
M55_T_E0: fM55_B3_E0_M55_T_E0 = 1; // T flow
```

```
/* WCA flow to cycle count */
tM55_B1_E0 = 2473 fM55_S_E0_M55_B1_E0 + 2473 fM55_B1_E0_M55_B1_E0;
tM55_I2_E0 = 75 fM55_I2_E0_M55_I2_E0 + 75 fM55_B1_E0_M55_I2_E0;
tM55_B3_E0 = 23 fM55_I2_E0_M55_B3_E0 + 23 fM55_B3_E0_M55_B3_E0;
/* Invocation(s) from Div:[M55] */

Value of objective function: 2571

Actual values of the variables:
tM55_S_E0                  0
tM55_B1_E0                 2473
tM55_I2_E0                 75
tM55_B3_E0                 23
tM55_T_E0                  0
fM55_S_E0_M55_B1_E0        1
fM55_B1_E0_M55_B1_E0       0
fM55_B1_E0_M55_I2_E0       1
fM55_I2_E0_M55_I2_E0       0
fM55_I2_E0_M55_B3_E0       1
fM55_B3_E0_M55_B3_E0       0
fM55_B3_E0_M55_T_E0        1

/*util.FixPoint.Div(III)I*/
digraph G {
size = "10,7.5"
	M55_S_E0->M55_B1_E0 [label="fM55_S_E0_M55_B1_E0=1"];
	M55_B1_E0->M55_I2_E0 [label="fM55_B1_E0_M55_I2_E0=1"];
	M55_I2_E0->M55_B3_E0 [label="fM55_I2_E0_M55_B3_E0=1"];
	M55_B3_E0->M55_T_E0 [label="fM55_B3_E0_M55_T_E0=1"];
	M55_S_E0;
	M55_B1_E0 [label="M55_B1_E0\n2473"];
	M55_I2_E0 [label="M55_I2_E0\n75"];
	M55_B3_E0 [label="M55_B3_E0\n23"];
	M55_T_E0;
}
*********************************************************************************
WCET info for:util.FixPoint.cos(II)I

Directed graph of basic blocks(row->column):
================================================================================
================================================================================
================================================================================
=====================================================================
   M59_S_E0M59_B1_E0M59_I2_E0M59_B3_E0M59_B4_E0M59_B5_E0M59_B6_E0M59_B7_E0M59_B8_E0M59_B
9_E0M59_B10_E0M59_B11_E0M59_B12_E0M59_B13_E0M59_B14_E0M59_B15_E0M59_B16_E0M59_B17_E0M59_
B18_E0M59_B19_E0M59_B20_E0M59_B21_E0M59_B22_E0M59_B23_E0M59_B24_E0M59_B25_E0M59_B26_E0M
59_B27_E0M59_B28_E0M59_B29_E0M59_B30_E0M59_I31_E0M59_B32_E0M59_T_E0
M59_S_E0 . 1 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
M59_B1_E0 . . 1 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
M59_I2_E0 . . . 1 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
M59_B3_E0 . . . . 1 1 . . . . . . . . . . . . . . . . . . . . . . . . . . . .
M59_B4_E0 . . . . . . . . 1 . . . . . . . . . . . . . . . . . . . . . . . . .
M59_B5_E0 . . . . . . 1 1 . . . . . . . . . . . . . . . . . . . . . . . . . .
M59_B6_E0 . . . . . . . . 1 . . . . . . . . . . . . . . . . . . . . . . . . .
M59_B7_E0 . . . . . . . . 1 . . . . . . . . . . . . . . . . . . . . . . . . .
M59_B8_E0 . . . . . . . . . 1 1 . . . . . . . . . . . . . . . . . . . . . . .
M59_B9_E0 . . . . . . . . . . 1 . . . . . . . . . . . . . . . . . . . . . . .
M59_B10_E0 . . . . . . . . . . . 1 . 1 . . . . . . . . . . . . . . . . . . . .
```

```
M59_B11_E0 . . . . . 1 1 . . . . . .
M59_B12_E0 . . . . . 1 . . . . . .
M59_B13_E0 . . . . . 1 . 1 . . . . . .
M59_B14_E0 . . . . . 1 1 . . . . . .
M59_B15_E0 . . . . . 1 . . . . . .
M59_B16_E0 . . . . . 1 . 1 . . . . . .
M59_B17_E0 . . . . . 1 1 . . . . . .
M59_B18_E0 . . . . . 1 . . . . . .
M59_B19_E0 . . . . . 1 . . . . . .
M59_B20_E0 . . . . . 1 . 1 . . . . . .
M59_B21_E0 . . . . . 1 1 . . . . . .
M59_B22_E0 . . . . . 1 . . .
M59_B23_E0 . . . . . 1 . 1 . . . . . .
M59_B24_E0 . . . . . 1 1 . . . . . .
M59_B25_E0 . . . . . 1 . . .
M59_B26_E0 . . . . . 1 . 1 . . . .
M59_B27_E0 . . . . . 1 1 . . . .
M59_B28_E0 . . . . . 1 . . .
M59_B29_E0 . . . . . 1 . . .
M59_B30_E0 . . . . . 1 . .
M59_I31_E0 . . . . . 1 .
M59_B32_E0 . . . . . 1
M59_T_E0 . . . . . .
================================================================================
================================================================================
================================================================================
=====================================================================

Table of basic blocks' and instructions
=========================================================================
Block Addr.  Bytecode              Cycles    Cache miss     Misc. info
             [opcode]              invoke  return
-------------------------------------------------------------------------
M59_S_E0'S'
  Src. line 219: int Temp1 = 0;
M59_B1_E0'B'{}<-[M59_S_E0 M59_B1_E0]0:     iconst_0[3]              1
     1:     istore_2[61]            1              ->I:Temp1
  Src. line 220: int Temp2 = 0;
     2:     iconst_0[3]             1
     3:     istore_3[62]            1              ->I:Temp2
  Src. line 221: int Rest = 0;
     4:     iconst_0[3]             1
     5:     istore[54]              2              ->I:Rest
  Src. line 222: int b = 0;
     7:     iconst_0[3]             1
     8:     istore[54]              2              ->I:b
  Src. line 224: Rest = Div(a * 180, PI >> (29 - SHIFT), SHIFT) + (90 << SHIFT);
    10:     iload_0[26]              1              I:a
    11:     sipush[17]               3
    14:     imul[104]               35
    15:     getstatic[178]           14              int util.FixPoint.PI
    18:     bipush[16]               2
    20:     iload_1[27]              1              I:SHIFT
    21:     isub[100]               1
    22:     ishr[122]               1
    23:     iload_1[27]              1              I:SHIFT sum(B1):     69
  Src. line 224: Rest = Div(a * 180, PI >> (29 - SHIFT), SHIFT) + (90 << SHIFT);
M59_I2_E0'I'{}<-[M59_I2_E0 M59_B1_E0]24:     invokestatic[184]          75       0     172   util.FixPoint.Div(III)I,
```

```
invoke(n=6):75/75 return(n=87):23/195sum(B2):     75
  Src. line 224: Rest = Div(a * 180, PI >> (29 - SHIFT), SHIFT) + (90 << SHIFT);
M59_B3_E0'B'{}<-[M59_I2_E0 M59_B3_E0]27:    bipush[16]                2
     29:    iload_1[27]               1               I:SHIFT
     30:    ishl[120]                 1
     31:    iadd[96]                  1
     32:    istore[54]                2               ->I:Rest
  Src. line 225: a = (Rest >> SHIFT) % 360;
     34:    iload[21]                 2               I:Rest
     36:    iload_1[27]               1               I:SHIFT
     37:    ishr[122]                 1
     38:    sipush[17]                3
     41:    irem[112]                 1940
     42:    istore_0[59]              1               ->I:a
  Src. line 226: if (a == 0)
     43:    iload_0[26]               1               I:a
     44:    ifne[154]->60:            4               sum(B3):   1960
  Src. line 227: Rest -= (360 << SHIFT);
M59_B4_E0'B'{}<-[M59_B4_E0 M59_B3_E0]47:    iload[21]                 2               I:Rest
     49:    sipush[17]                3
     52:    iload_1[27]               1               I:SHIFT
     53:    ishl[120]                 1
     54:    isub[100]                 1
     55:    istore[54]                2               ->I:Rest
     57:    goto[167]->93:            4               sum(B4):     14
  Src. line 228: else if (Rest > (360 << SHIFT))
M59_B5_E0'B'{}<-[M59_B3_E0]60:    iload[21]                 2               I:Rest
     62:    sipush[17]                3
     65:    iload_1[27]               1               I:SHIFT
     66:    ishl[120]                 1
     67:    if_icmple[164]->85:       4               sum(B5):     11
  Src. line 229: Rest -= ((a + 360) << SHIFT);
M59_B6_E0'B'{}<-[M59_B6_E0 M59_B5_E0]70:    iload[21]                 2               I:Rest
     72:    iload_0[26]               1               I:a
     73:    sipush[17]                3
     76:    iadd[96]                  1
     77:    iload_1[27]               1               I:SHIFT
     78:    ishl[120]                 1
     79:    isub[100]                 1
     80:    istore[54]                2               ->I:Rest
     82:    goto[167]->93:            4               sum(B6):     16
  Src. line 231: Rest -= ((a) << SHIFT);
M59_B7_E0'B'{}<-[M59_B5_E0]85:    iload[21]                 2               I:Rest
     87:    iload_0[26]               1               I:a
     88:    iload_1[27]               1               I:SHIFT
     89:    ishl[120]                 1
     90:    isub[100]                 1
     91:    istore[54]                2               ->I:Rest sum(B7):      8
  Src. line 232: if (a < 0)
M59_B8_E0'B'{}<-[M59_B4_E0 M59_B6_E0 M59_B7_E0]93:    iload_0[26]               1               I:a
     94:    ifge[156]->103:           4               sum(B8):      5
  Src. line 233: a += 360;
M59_B9_E0'B'{}<-[M59_B9_E0 M59_B8_E0]97:    iload_0[26]               1               I:a
     98:    sipush[17]                3
     101:   iadd[96]                  1
     102:   istore_0[59]              1               ->I:a sum(B9):      6
  Src. line 234: b = a + 1;
M59_B10_E0'B'{}<-[M59_B9_E0 M59_B8_E0]103:   iload_0[26]               1               I:a
```

```
    104:  iconst_1[4]              1
    105:  iadd[96]                 1
    106:  istore[54]               2               ->I:b
  Src. line 239: if (90 <= a && a <= 180)
    108:  bipush[16]               2
    110:  iload_0[26]              1               I:a
    111:  if_icmpgt[163]->139:     4               sum(B10):    12
  Src. line 239: if (90 <= a && a <= 180)
M59_B11_E0'B'{}<-[M59_B11_E0 M59_B10_E0]114:  iload_0[26]              1               I:a
    115:  sipush[17]               3
    118:  if_icmpgt[163]->139:     4               sum(B11):    8
  Src. line 240: Temp1 = (SIN[180 - a] >> (30 - SHIFT));
M59_B12_E0'B'{}<-[M59_B11_E0 M59_B12_E0]121:  getstatic[178]           14              int[] util.FixPoint.SIN
    124:  sipush[17]               3
    127:  iload_0[26]              1               I:a
    128:  isub[100]                1
    129:  iaload[46]               36              I
    130:  bipush[16]               2
    132:  iload_1[27]              1               I:SHIFT
    133:  isub[100]                1
    134:  ishr[122]                1
    135:  istore_2[61]             1               ->I:Temp1
    136:  goto[167]->216:          4               sum(B12):    65
  Src. line 241: else if (180 <= a && a <= 270)
M59_B13_E0'B'{}<-[M59_B11_E0 M59_B10_E0]139:  sipush[17]              3
    142:  iload_0[26]              1               I:a
    143:  if_icmpgt[163]->172:     4               sum(B13):    8
  Src. line 241: else if (180 <= a && a <= 270)
M59_B14_E0'B'{}<-[M59_B14_E0 M59_B13_E0]146:  iload_0[26]              1               I:a
    147:  sipush[17]               3
    150:  if_icmpgt[163]->172:     4               sum(B14):    8
  Src. line 242: Temp1 = -(SIN[a - 180] >> (30 - SHIFT));
M59_B15_E0'B'{}<-[M59_B14_E0 M59_B15_E0]153:  getstatic[178]           14              int[] util.FixPoint.SIN
    156:  iload_0[26]              1               I:a
    157:  sipush[17]               3
    160:  isub[100]                1
    161:  iaload[46]               36              I
    162:  bipush[16]               2
    164:  iload_1[27]              1               I:SHIFT
    165:  isub[100]                1
    166:  ishr[122]                1
    167:  ineg[116]                4
    168:  istore_2[61]             1               ->I:Temp1
    169:  goto[167]->216:          4               sum(B15):    69
  Src. line 243: else if (270 <= a && a <= 360)
M59_B16_E0'B'{}<-[M59_B14_E0 M59_B13_E0]172:  sipush[17]              3
    175:  iload_0[26]              1               I:a
    176:  if_icmpgt[163]->205:     4               sum(B16):    8
  Src. line 243: else if (270 <= a && a <= 360)
M59_B17_E0'B'{}<-[M59_B17_E0 M59_B16_E0]179:  iload_0[26]              1               I:a
    180:  sipush[17]               3
    183:  if_icmpgt[163]->205:     4               sum(B17):    8
  Src. line 244: Temp1 = -(SIN[360 - a] >> (30 - SHIFT));
M59_B18_E0'B'{}<-[M59_B18_E0 M59_B17_E0]186:  getstatic[178]           14              int[] util.FixPoint.SIN
    189:  sipush[17]               3
    192:  iload_0[26]              1               I:a
    193:  isub[100]                1
    194:  iaload[46]               36              I
```

```
    195:  bipush[16]              2
    197:  iload_1[27]             1              I:SHIFT
    198:  isub[100]               1
    199:  ishr[122]               1
    200:  ineg[116]               4
    201:  istore_2[61]            1              ->I:Temp1
    202:  goto[167]->216:         4              sum(B18):    69
  Src. line 246: Temp1 = (SIN[a] >> (30 - SHIFT));
M59_B19_E0'B'{}<-[M59_B17_E0 M59_B16_E0]205:  getstatic[178]         14              int[] util.FixPoint.SIN
    208:  iload_0[26]             1              I:a
    209:  iaload[46]              36             I
    210:  bipush[16]              2
    212:  iload_1[27]             1              I:SHIFT
    213:  isub[100]               1
    214:  ishr[122]               1
    215:  istore_2[61]            1              ->I:Temp1 sum(B19):    57
  Src. line 248: if (90 <= b && b <= 180)
M59_B20_E0'B'{}<-[M59_B19_E0 M59_B18_E0 M59_B12_E0 M59_B15_E0]216:  bipush[16]              2
    218:  iload[21]               2              I:b
    220:  if_icmpgt[163]->250:    4              sum(B20):    8
  Src. line 248: if (90 <= b && b <= 180)
M59_B21_E0'B'{}<-[M59_B20_E0 M59_B21_E0]223:  iload[21]              2              I:b
    225:  sipush[17]              3
    228:  if_icmpgt[163]->250:    4              sum(B21):    9
  Src. line 249: Temp2 = (SIN[180 - b] >> (30 - SHIFT));
M59_B22_E0'B'{}<-[M59_B22_E0 M59_B21_E0]231:  getstatic[178]         14              int[] util.FixPoint.SIN
    234:  sipush[17]              3
    237:  iload[21]               2              I:b
    239:  isub[100]               1
    240:  iaload[46]              36             I
    241:  bipush[16]              2
    243:  iload_1[27]             1              I:SHIFT
    244:  isub[100]               1
    245:  ishr[122]               1
    246:  istore_3[62]            1              ->I:Temp2
    247:  goto[167]->334:         4              sum(B22):    66
  Src. line 250: else if (180 <= b && b <= 270)
M59_B23_E0'B'{}<-[M59_B20_E0 M59_B21_E0]250:  sipush[17]              3
    253:  iload[21]               2              I:b
    255:  if_icmpgt[163]->286:    4              sum(B23):    9
  Src. line 250: else if (180 <= b && b <= 270)
M59_B24_E0'B'{}<-[M59_B24_E0 M59_B23_E0]258:  iload[21]              2              I:b
    260:  sipush[17]              3
    263:  if_icmpgt[163]->286:    4              sum(B24):    9
  Src. line 251: Temp2 = -(SIN[b - 180] >> (30 - SHIFT));
M59_B25_E0'B'{}<-[M59_B25_E0 M59_B24_E0]266:  getstatic[178]         14              int[] util.FixPoint.SIN
    269:  iload[21]               2              I:b
    271:  sipush[17]              3
    274:  isub[100]               1
    275:  iaload[46]              36             I
    276:  bipush[16]              2
    278:  iload_1[27]             1              I:SHIFT
    279:  isub[100]               1
    280:  ishr[122]               1
    281:  ineg[116]               4
    282:  istore_3[62]            1              ->I:Temp2
    283:  goto[167]->334:         4              sum(B25):    70
  Src. line 252: else if (270 <= b && b <= 360)
```

```
M59_B26_E0'B'{}<-[M59_B24_E0 M59_B23_E0]286:   sipush[17]                3
    289:   iload[21]                2               I:b
    291:   if_icmpgt[163]->322:     4               sum(B26):     9
  Src. line 252: else if (270 <= b && b <= 360)
M59_B27_E0'B'{}<-[M59_B27_E0 M59_B26_E0]294:   iload[21]                2               I:b
    296:   sipush[17]              3
    299:   if_icmpgt[163]->322:     4               sum(B27):     9
  Src. line 253: Temp2 = -(SIN[360 - b] >> (30 - SHIFT));
M59_B28_E0'B'{}<-[M59_B27_E0 M59_B28_E0]302:   getstatic[178]           14              int[] util.FixPoint.SIN
    305:   sipush[17]              3
    308:   iload[21]               2               I:b
    310:   isub[100]               1
    311:   iaload[46]              36              I
    312:   bipush[16]              2
    314:   iload_1[27]              1               I:SHIFT
    315:   isub[100]               1
    316:   ishr[122]               1
    317:   ineg[116]               4
    318:   istore_3[62]            1               ->I:Temp2
    319:   goto[167]->334:         4               sum(B28):     70
  Src. line 255: Temp2 = (SIN[b] >> (30 - SHIFT));
M59_B29_E0'B'{}<-[M59_B27_E0 M59_B26_E0]322:   getstatic[178]           14              int[] util.FixPoint.SIN
    325:   iload[21]               2               I:b
    327:   iaload[46]              36              I
    328:   bipush[16]              2
    330:   iload_1[27]              1               I:SHIFT
    331:   isub[100]               1
    332:   ishr[122]               1
    333:   istore_3[62]            1               ->I:Temp2 sum(B29):     58
  Src. line 258: return (Temp1 + Mul(Temp2 - Temp1, Rest, SHIFT));
M59_B30_E0'B'{}<-[M59_B29_E0 M59_B25_E0 M59_B28_E0 M59_B22_E0]334:   iload_2[28]              1
I:Temp1
    335:   iload_3[29]              1               I:Temp2
    336:   iload_2[28]              1               I:Temp1
    337:   isub[100]               1
    338:   iload[21]               2               I:Rest
    340:   iload_1[27]              1               I:SHIFT sum(B30):     7
  Src. line 258: return (Temp1 + Mul(Temp2 - Temp1, Rest, SHIFT));
M59_I31_E0'I'{}<-[M59_I31_E0 M59_B30_E0]341:   invokestatic[184]         75      5     172   util.FixPoint.Mul(III)I,
invoke(n=17):75/80 return(n=87):23/195sum(B31):     75
  Src. line 258: return (Temp1 + Mul(Temp2 - Temp1, Rest, SHIFT));
M59_B32_E0'B'{}<-[M59_I31_E0 M59_B32_E0]344:   iadd[96]                 1
    345:   ireturn[172]            23              sum(B32):     24
M59_T_E0'T'<-[M59_B32_E0]
================================================================================
Info: n=87 a=-1 r=1 w=1

/***WCET calculation source***/
/* WCA WCET objective: util.FixPoint.cos */
max: tM59_S_E0 tM59_B1_E0 tM59_I2_E0 tM59_B3_E0 tM59_B4_E0 tM59_B5_E0 tM59_B6_E0 tM59_B7_E0
tM59_B8_E0 tM59_B9_E0 tM59_B10_E0 tM59_B11_E0 tM59_B12_E0 tM59_B13_E0 tM59_B14_E0 tM59_B15_E0
tM59_B16_E0 tM59_B17_E0 tM59_B18_E0 tM59_B19_E0 tM59_B20_E0 tM59_B21_E0 tM59_B22_E0 tM59_B23_E0
tM59_B24_E0 tM59_B25_E0 tM59_B26_E0 tM59_B27_E0 tM59_B28_E0 tM59_B29_E0 tM59_B30_E0 tM59_I31_E0
tM59_B32_E0 tM59_T_E0;
/* WCA flow constraints: cos : M59 */
M59_S_E0: 1 = fM59_S_E0_M59_B1_E0; // S flow
M59_B1_E0: fM59_S_E0_M59_B1_E0 + fM59_B1_E0_M59_B1_E0 = fM59_B1_E0_M59_I2_E0;
M59_I2_E0: fM59_I2_E0_M59_I2_E0 + fM59_B1_E0_M59_I2_E0 = fM59_I2_E0_M59_B3_E0;
```

M59_B3_E0: fM59_I2_E0_M59_B3_E0 + fM59_B3_E0_M59_B3_E0 = fM59_B3_E0_M59_B4_E0 +
fM59_B3_E0_M59_B5_E0;
M59_B4_E0: fM59_B4_E0_M59_B4_E0 + fM59_B3_E0_M59_B4_E0 = fM59_B4_E0_M59_B8_E0;
M59_B5_E0: fM59_B3_E0_M59_B5_E0 = fM59_B5_E0_M59_B6_E0 + fM59_B5_E0_M59_B7_E0;
M59_B6_E0: fM59_B6_E0_M59_B6_E0 + fM59_B5_E0_M59_B6_E0 = fM59_B6_E0_M59_B8_E0;
M59_B7_E0: fM59_B5_E0_M59_B7_E0 = fM59_B7_E0_M59_B8_E0;
M59_B8_E0: fM59_B4_E0_M59_B8_E0 + fM59_B6_E0_M59_B8_E0 + fM59_B7_E0_M59_B8_E0 =
fM59_B8_E0_M59_B9_E0 + fM59_B8_E0_M59_B10_E0;
M59_B9_E0: fM59_B9_E0_M59_B9_E0 + fM59_B8_E0_M59_B9_E0 = fM59_B9_E0_M59_B10_E0;
M59_B10_E0: fM59_B9_E0_M59_B10_E0 + fM59_B8_E0_M59_B10_E0 = fM59_B10_E0_M59_B11_E0 +
fM59_B10_E0_M59_B13_E0;
M59_B11_E0: fM59_B11_E0_M59_B11_E0 + fM59_B10_E0_M59_B11_E0 = fM59_B11_E0_M59_B12_E0 +
fM59_B11_E0_M59_B13_E0;
M59_B12_E0: fM59_B11_E0_M59_B12_E0 + fM59_B12_E0_M59_B12_E0 = fM59_B12_E0_M59_B20_E0;
M59_B13_E0: fM59_B11_E0_M59_B13_E0 + fM59_B10_E0_M59_B13_E0 = fM59_B13_E0_M59_B14_E0 +
fM59_B13_E0_M59_B16_E0;
M59_B14_E0: fM59_B14_E0_M59_B14_E0 + fM59_B13_E0_M59_B14_E0 = fM59_B14_E0_M59_B15_E0 +
fM59_B14_E0_M59_B16_E0;
M59_B15_E0: fM59_B14_E0_M59_B15_E0 + fM59_B15_E0_M59_B15_E0 = fM59_B15_E0_M59_B20_E0;
M59_B16_E0: fM59_B14_E0_M59_B16_E0 + fM59_B13_E0_M59_B16_E0 = fM59_B16_E0_M59_B17_E0 +
fM59_B16_E0_M59_B19_E0;
M59_B17_E0: fM59_B17_E0_M59_B17_E0 + fM59_B16_E0_M59_B17_E0 = fM59_B17_E0_M59_B18_E0 +
fM59_B17_E0_M59_B19_E0;
M59_B18_E0: fM59_B18_E0_M59_B18_E0 + fM59_B17_E0_M59_B18_E0 = fM59_B18_E0_M59_B20_E0;
M59_B19_E0: fM59_B17_E0_M59_B19_E0 + fM59_B16_E0_M59_B19_E0 = fM59_B19_E0_M59_B20_E0;
M59_B20_E0: fM59_B19_E0_M59_B20_E0 + fM59_B18_E0_M59_B20_E0 + fM59_B12_E0_M59_B20_E0 +
fM59_B15_E0_M59_B20_E0 = fM59_B20_E0_M59_B21_E0 + fM59_B20_E0_M59_B23_E0;
M59_B21_E0: fM59_B20_E0_M59_B21_E0 + fM59_B21_E0_M59_B21_E0 = fM59_B21_E0_M59_B22_E0 +
fM59_B21_E0_M59_B23_E0;
M59_B22_E0: fM59_B22_E0_M59_B22_E0 + fM59_B21_E0_M59_B22_E0 = fM59_B22_E0_M59_B30_E0;
M59_B23_E0: fM59_B20_E0_M59_B23_E0 + fM59_B21_E0_M59_B23_E0 = fM59_B23_E0_M59_B24_E0 +
fM59_B23_E0_M59_B26_E0;
M59_B24_E0: fM59_B24_E0_M59_B24_E0 + fM59_B23_E0_M59_B24_E0 = fM59_B24_E0_M59_B25_E0 +
fM59_B24_E0_M59_B26_E0;
M59_B25_E0: fM59_B25_E0_M59_B25_E0 + fM59_B24_E0_M59_B25_E0 = fM59_B25_E0_M59_B30_E0;
M59_B26_E0: fM59_B24_E0_M59_B26_E0 + fM59_B23_E0_M59_B26_E0 = fM59_B26_E0_M59_B27_E0 +
fM59_B26_E0_M59_B29_E0;
M59_B27_E0: fM59_B27_E0_M59_B27_E0 + fM59_B26_E0_M59_B27_E0 = fM59_B27_E0_M59_B28_E0 +
fM59_B27_E0_M59_B29_E0;
M59_B28_E0: fM59_B27_E0_M59_B28_E0 + fM59_B28_E0_M59_B28_E0 = fM59_B28_E0_M59_B30_E0;
M59_B29_E0: fM59_B27_E0_M59_B29_E0 + fM59_B26_E0_M59_B29_E0 = fM59_B29_E0_M59_B30_E0;
M59_B30_E0: fM59_B29_E0_M59_B30_E0 + fM59_B25_E0_M59_B30_E0 + fM59_B28_E0_M59_B30_E0 +
fM59_B22_E0_M59_B30_E0 = fM59_B30_E0_M59_I31_E0;
M59_I31_E0: fM59_I31_E0_M59_I31_E0 + fM59_B30_E0_M59_I31_E0 = fM59_I31_E0_M59_B32_E0;
M59_B32_E0: fM59_I31_E0_M59_B32_E0 + fM59_B32_E0_M59_B32_E0 = fM59_B32_E0_M59_T_E0;
M59_T_E0: fM59_B32_E0_M59_T_E0 = 1; // T flow
/* WCA flow to cycle count */
tM59_B1_E0 = 69 fM59_S_E0_M59_B1_E0 + 69 fM59_B1_E0_M59_B1_E0;
tM59_I2_E0 = 75 fM59_I2_E0_M59_I2_E0 + 75 fM59_B1_E0_M59_I2_E0;
tM59_B3_E0 = 1960 fM59_I2_E0_M59_B3_E0 + 1960 fM59_B3_E0_M59_B3_E0;
tM59_B4_E0 = 14 fM59_B4_E0_M59_B4_E0 + 14 fM59_B3_E0_M59_B4_E0;
tM59_B5_E0 = 11 fM59_B3_E0_M59_B5_E0;
tM59_B6_E0 = 16 fM59_B6_E0_M59_B6_E0 + 16 fM59_B5_E0_M59_B6_E0;
tM59_B7_E0 = 8 fM59_B5_E0_M59_B7_E0;
tM59_B8_E0 = 5 fM59_B4_E0_M59_B8_E0 + 5 fM59_B6_E0_M59_B8_E0 + 5 fM59_B7_E0_M59_B8_E0;
tM59_B9_E0 = 6 fM59_B9_E0_M59_B9_E0 + 6 fM59_B8_E0_M59_B9_E0;
tM59_B10_E0 = 12 fM59_B9_E0_M59_B10_E0 + 12 fM59_B8_E0_M59_B10_E0;
tM59_B11_E0 = 8 fM59_B11_E0_M59_B11_E0 + 8 fM59_B10_E0_M59_B11_E0;

tM59_B12_E0 = 65 fM59_B11_E0_M59_B12_E0 + 65 fM59_B12_E0_M59_B12_E0;
tM59_B13_E0 = 8 fM59_B11_E0_M59_B13_E0 + 8 fM59_B10_E0_M59_B13_E0;
tM59_B14_E0 = 8 fM59_B14_E0_M59_B14_E0 + 8 fM59_B13_E0_M59_B14_E0;
tM59_B15_E0 = 69 fM59_B14_E0_M59_B15_E0 + 69 fM59_B15_E0_M59_B15_E0;
tM59_B16_E0 = 8 fM59_B14_E0_M59_B16_E0 + 8 fM59_B13_E0_M59_B16_E0;
tM59_B17_E0 = 8 fM59_B17_E0_M59_B17_E0 + 8 fM59_B16_E0_M59_B17_E0;
tM59_B18_E0 = 69 fM59_B18_E0_M59_B18_E0 + 69 fM59_B17_E0_M59_B18_E0;
tM59_B19_E0 = 57 fM59_B17_E0_M59_B19_E0 + 57 fM59_B16_E0_M59_B19_E0;
tM59_B20_E0 = 8 fM59_B19_E0_M59_B20_E0 + 8 fM59_B18_E0_M59_B20_E0 + 8 fM59_B12_E0_M59_B20_E0 + 8 fM59_B15_E0_M59_B20_E0;
tM59_B21_E0 = 9 fM59_B20_E0_M59_B21_E0 + 9 fM59_B21_E0_M59_B21_E0;
tM59_B22_E0 = 66 fM59_B22_E0_M59_B22_E0 + 66 fM59_B21_E0_M59_B22_E0;
tM59_B23_E0 = 9 fM59_B20_E0_M59_B23_E0 + 9 fM59_B21_E0_M59_B23_E0;
tM59_B24_E0 = 9 fM59_B24_E0_M59_B24_E0 + 9 fM59_B23_E0_M59_B24_E0;
tM59_B25_E0 = 70 fM59_B25_E0_M59_B25_E0 + 70 fM59_B24_E0_M59_B25_E0;
tM59_B26_E0 = 9 fM59_B24_E0_M59_B26_E0 + 9 fM59_B23_E0_M59_B26_E0;
tM59_B27_E0 = 9 fM59_B27_E0_M59_B27_E0 + 9 fM59_B26_E0_M59_B27_E0;
tM59_B28_E0 = 70 fM59_B27_E0_M59_B28_E0 + 70 fM59_B28_E0_M59_B28_E0;
tM59_B29_E0 = 58 fM59_B27_E0_M59_B29_E0 + 58 fM59_B26_E0_M59_B29_E0;
tM59_B30_E0 = 7 fM59_B29_E0_M59_B30_E0 + 7 fM59_B25_E0_M59_B30_E0 + 7 fM59_B28_E0_M59_B30_E0 + 7 fM59_B22_E0_M59_B30_E0;
tM59_I31_E0 = 75 fM59_I31_E0_M59_I31_E0 + 75 fM59_B30_E0_M59_I31_E0;
tM59_B32_E0 = 24 fM59_I31_E0_M59_B32_E0 + 24 fM59_B32_E0_M59_B32_E0;
/* Invocation(s) from cos:[M59] */

Value of objective function: 2492

Actual values of the variables:

tM59_S_E0	0
tM59_B1_E0	69
tM59_I2_E0	75
tM59_B3_E0	1960
tM59_B4_E0	0
tM59_B5_E0	11
tM59_B6_E0	16
tM59_B7_E0	0
tM59_B8_E0	5
tM59_B9_E0	6
tM59_B10_E0	12
tM59_B11_E0	8
tM59_B12_E0	0
tM59_B13_E0	8
tM59_B14_E0	8
tM59_B15_E0	0
tM59_B16_E0	8
tM59_B17_E0	8
tM59_B18_E0	69
tM59_B19_E0	0
tM59_B20_E0	8
tM59_B21_E0	9
tM59_B22_E0	0
tM59_B23_E0	9
tM59_B24_E0	9
tM59_B25_E0	0
tM59_B26_E0	9
tM59_B27_E0	9
tM59_B28_E0	70
tM59_B29_E0	0

tM59_B30_E0 7
tM59_I31_E0 75
tM59_B32_E0 24
tM59_T_E0 0
fM59_S_E0_M59_B1_E0 1
fM59_B1_E0_M59_B1_E0 0
fM59_B1_E0_M59_I2_E0 1
fM59_I2_E0_M59_I2_E0 0
fM59_I2_E0_M59_B3_E0 1
fM59_B3_E0_M59_B3_E0 0
fM59_B3_E0_M59_B4_E0 0
fM59_B3_E0_M59_B5_E0 1
fM59_B4_E0_M59_B4_E0 0
fM59_B4_E0_M59_B8_E0 0
fM59_B5_E0_M59_B6_E0 1
fM59_B5_E0_M59_B7_E0 0
fM59_B6_E0_M59_B6_E0 0
fM59_B6_E0_M59_B8_E0 1
fM59_B7_E0_M59_B8_E0 0
fM59_B8_E0_M59_B9_E0 1
fM59_B8_E0_M59_B10_E0 0
fM59_B9_E0_M59_B9_E0 0
fM59_B9_E0_M59_B10_E0 1
fM59_B10_E0_M59_B11_E0 1
fM59_B10_E0_M59_B13_E0 0
fM59_B11_E0_M59_B11_E0 0
fM59_B11_E0_M59_B12_E0 0
fM59_B11_E0_M59_B13_E0 1
fM59_B12_E0_M59_B12_E0 0
fM59_B12_E0_M59_B20_E0 0
fM59_B13_E0_M59_B14_E0 1
fM59_B13_E0_M59_B16_E0 0
fM59_B14_E0_M59_B14_E0 0
fM59_B14_E0_M59_B15_E0 0
fM59_B14_E0_M59_B16_E0 1
fM59_B15_E0_M59_B15_E0 0
fM59_B15_E0_M59_B20_E0 0
fM59_B16_E0_M59_B17_E0 1
fM59_B16_E0_M59_B19_E0 0
fM59_B17_E0_M59_B17_E0 0
fM59_B17_E0_M59_B18_E0 1
fM59_B17_E0_M59_B19_E0 0
fM59_B18_E0_M59_B18_E0 0
fM59_B18_E0_M59_B20_E0 1
fM59_B19_E0_M59_B20_E0 0
fM59_B20_E0_M59_B21_E0 1
fM59_B20_E0_M59_B23_E0 0
fM59_B21_E0_M59_B21_E0 0
fM59_B21_E0_M59_B22_E0 0
fM59_B21_E0_M59_B23_E0 1
fM59_B22_E0_M59_B22_E0 0
fM59_B22_E0_M59_B30_E0 0
fM59_B23_E0_M59_B24_E0 1
fM59_B23_E0_M59_B26_E0 0
fM59_B24_E0_M59_B24_E0 0
fM59_B24_E0_M59_B25_E0 0
fM59_B24_E0_M59_B26_E0 1
fM59_B25_E0_M59_B25_E0 0

```
fM59_B25_E0_M59_B30_E0          0
fM59_B26_E0_M59_B27_E0          1
fM59_B26_E0_M59_B29_E0          0
fM59_B27_E0_M59_B27_E0          0
fM59_B27_E0_M59_B28_E0          1
fM59_B27_E0_M59_B29_E0          0
fM59_B28_E0_M59_B28_E0          0
fM59_B28_E0_M59_B30_E0          1
fM59_B29_E0_M59_B30_E0          0
fM59_B30_E0_M59_I31_E0          1
fM59_I31_E0_M59_I31_E0          0
fM59_I31_E0_M59_B32_E0          1
fM59_B32_E0_M59_B32_E0          0
fM59_B32_E0_M59_T_E0            1

/*util.FixPoint.cos(II)I*/
digraph G {
size = "10,7.5"
        M59_S_E0->M59_B1_E0 [label="fM59_S_E0_M59_B1_E0=1"];
        M59_B1_E0->M59_I2_E0 [label="fM59_B1_E0_M59_I2_E0=1"];
        M59_I2_E0->M59_B3_E0 [label="fM59_I2_E0_M59_B3_E0=1"];
        M59_B3_E0->M59_B4_E0 [style=dashed,label="fM59_B3_E0_M59_B4_E0=0"];
        M59_B3_E0->M59_B5_E0 [label="fM59_B3_E0_M59_B5_E0=1"];
        M59_B4_E0->M59_B8_E0 [style=dashed,label="fM59_B4_E0_M59_B8_E0=0"];
        M59_B5_E0->M59_B6_E0 [label="fM59_B5_E0_M59_B6_E0=1"];
        M59_B5_E0->M59_B7_E0 [style=dashed,label="fM59_B5_E0_M59_B7_E0=0"];
        M59_B6_E0->M59_B8_E0 [label="fM59_B6_E0_M59_B8_E0=1"];
        M59_B7_E0->M59_B8_E0 [style=dashed,label="fM59_B7_E0_M59_B8_E0=0"];
        M59_B8_E0->M59_B9_E0 [label="fM59_B8_E0_M59_B9_E0=1"];
        M59_B8_E0->M59_B10_E0 [style=dashed,label="fM59_B8_E0_M59_B10_E0=0"];
        M59_B9_E0->M59_B10_E0 [label="fM59_B9_E0_M59_B10_E0=1"];
        M59_B10_E0->M59_B11_E0 [label="fM59_B10_E0_M59_B11_E0=1"];
        M59_B10_E0->M59_B13_E0 [style=dashed,label="fM59_B10_E0_M59_B13_E0=0"];
        M59_B11_E0->M59_B12_E0 [style=dashed,label="fM59_B11_E0_M59_B12_E0=0"];
        M59_B11_E0->M59_B13_E0 [label="fM59_B11_E0_M59_B13_E0=1"];
        M59_B12_E0->M59_B20_E0 [style=dashed,label="fM59_B12_E0_M59_B20_E0=0"];
        M59_B13_E0->M59_B14_E0 [label="fM59_B13_E0_M59_B14_E0=1"];
        M59_B13_E0->M59_B16_E0 [style=dashed,label="fM59_B13_E0_M59_B16_E0=0"];
        M59_B14_E0->M59_B15_E0 [style=dashed,label="fM59_B14_E0_M59_B15_E0=0"];
        M59_B14_E0->M59_B16_E0 [label="fM59_B14_E0_M59_B16_E0=1"];
        M59_B15_E0->M59_B20_E0 [style=dashed,label="fM59_B15_E0_M59_B20_E0=0"];
        M59_B16_E0->M59_B17_E0 [label="fM59_B16_E0_M59_B17_E0=1"];
        M59_B16_E0->M59_B19_E0 [style=dashed,label="fM59_B16_E0_M59_B19_E0=0"];
        M59_B17_E0->M59_B18_E0 [label="fM59_B17_E0_M59_B18_E0=1"];
        M59_B17_E0->M59_B19_E0 [style=dashed,label="fM59_B17_E0_M59_B19_E0=0"];
        M59_B18_E0->M59_B20_E0 [label="fM59_B18_E0_M59_B20_E0=1"];
        M59_B19_E0->M59_B20_E0 [style=dashed,label="fM59_B19_E0_M59_B20_E0=0"];
        M59_B20_E0->M59_B21_E0 [label="fM59_B20_E0_M59_B21_E0=1"];
        M59_B20_E0->M59_B23_E0 [style=dashed,label="fM59_B20_E0_M59_B23_E0=0"];
        M59_B21_E0->M59_B22_E0 [style=dashed,label="fM59_B21_E0_M59_B22_E0=0"];
        M59_B21_E0->M59_B23_E0 [label="fM59_B21_E0_M59_B23_E0=1"];
        M59_B22_E0->M59_B30_E0 [style=dashed,label="fM59_B22_E0_M59_B30_E0=0"];
        M59_B23_E0->M59_B24_E0 [label="fM59_B23_E0_M59_B24_E0=1"];
        M59_B23_E0->M59_B26_E0 [style=dashed,label="fM59_B23_E0_M59_B26_E0=0"];
        M59_B24_E0->M59_B25_E0 [style=dashed,label="fM59_B24_E0_M59_B25_E0=0"];
        M59_B24_E0->M59_B26_E0 [label="fM59_B24_E0_M59_B26_E0=1"];
        M59_B25_E0->M59_B30_E0 [style=dashed,label="fM59_B25_E0_M59_B30_E0=0"];
        M59_B26_E0->M59_B27_E0 [label="fM59_B26_E0_M59_B27_E0=1"];
```

```
        M59_B26_E0->M59_B29_E0 [style=dashed,label="fM59_B26_E0_M59_B29_E0=0"];
        M59_B27_E0->M59_B28_E0 [label="fM59_B27_E0_M59_B28_E0=1"];
        M59_B27_E0->M59_B29_E0 [style=dashed,label="fM59_B27_E0_M59_B29_E0=0"];
        M59_B28_E0->M59_B30_E0 [label="fM59_B28_E0_M59_B30_E0=1"];
        M59_B29_E0->M59_B30_E0 [style=dashed,label="fM59_B29_E0_M59_B30_E0=0"];
        M59_B30_E0->M59_I31_E0 [label="fM59_B30_E0_M59_I31_E0=1"];
        M59_I31_E0->M59_B32_E0 [label="fM59_I31_E0_M59_B32_E0=1"];
        M59_B32_E0->M59_T_E0 [label="fM59_B32_E0_M59_T_E0=1"];
        M59_S_E0;
        M59_B1_E0 [label="M59_B1_E0\n69"];
        M59_I2_E0 [label="M59_I2_E0\n75"];
        M59_B3_E0 [label="M59_B3_E0\n1960"];
        M59_B4_E0 [label="M59_B4_E0\n14"];
        M59_B5_E0 [label="M59_B5_E0\n11"];
        M59_B6_E0 [label="M59_B6_E0\n16"];
        M59_B7_E0 [label="M59_B7_E0\n8"];
        M59_B8_E0 [label="M59_B8_E0\n5"];
        M59_B9_E0 [label="M59_B9_E0\n6"];
        M59_B10_E0 [label="M59_B10_E0\n12"];
        M59_B11_E0 [label="M59_B11_E0\n8"];
        M59_B12_E0 [label="M59_B12_E0\n65"];
        M59_B13_E0 [label="M59_B13_E0\n8"];
        M59_B14_E0 [label="M59_B14_E0\n8"];
        M59_B15_E0 [label="M59_B15_E0\n69"];
        M59_B16_E0 [label="M59_B16_E0\n8"];
        M59_B17_E0 [label="M59_B17_E0\n8"];
        M59_B18_E0 [label="M59_B18_E0\n69"];
        M59_B19_E0 [label="M59_B19_E0\n57"];
        M59_B20_E0 [label="M59_B20_E0\n8"];
        M59_B21_E0 [label="M59_B21_E0\n9"];
        M59_B22_E0 [label="M59_B22_E0\n66"];
        M59_B23_E0 [label="M59_B23_E0\n9"];
        M59_B24_E0 [label="M59_B24_E0\n9"];
        M59_B25_E0 [label="M59_B25_E0\n70"];
        M59_B26_E0 [label="M59_B26_E0\n9"];
        M59_B27_E0 [label="M59_B27_E0\n9"];
        M59_B28_E0 [label="M59_B28_E0\n70"];
        M59_B29_E0 [label="M59_B29_E0\n58"];
        M59_B30_E0 [label="M59_B30_E0\n7"];
        M59_I31_E0 [label="M59_I31_E0\n75"];
        M59_B32_E0 [label="M59_B32_E0\n24"];
        M59_T_E0;
}

**************************************************
Note: Remember to keep WCETAnalyzer updated
each time a bytecode implementation is changed.
```

B.2.2 idiv

**********APPLICATION WCET=2478**********
/***WCET calculation source***/
/* WCA WCET objective: wcet.WCATestIDIV.main */
max: tM66_S_E1 tM66_B1_E1 tM66_I2_E1 tM66_B3_E1 tM66_T_E1 tM67_S_E1 tM67_B1_E1 tM67_B2_E1
tM67_B3_E1 tM67_B4_E1 tM67_B5_E1 tM67_B6_E1 tM67_B7_E1 tM67_B8_E1 tM67_B9_E1 tM67_B10_E1
tM67_B11_E1 tM67_B12_E1 tM67_B13_E1 tM67_B14_E1 tM67_B15_E1 tM67_B16_E1 tM67_B17_E1 tM67_B18_E1
tM67_B19_E1 tM67_T_E1;
/* WCA flow constraints: main : M66 */
M66_S_E1: 1 = fM66_S_E1_M66_B1_E1; // S flow
M66_B1_E1: fM66_S_E1_M66_B1_E1 + fM66_B1_E1_M66_B1_E1 = fM66_B1_E1_M66_I2_E1;
M66_I2_E1: fM66_I2_E1_M66_I2_E1 + fM66_B1_E1_M66_I2_E1 = fM66_I2_E1_M66_B3_E1;
/* Connecting(invoking) to wcet.WCATestIDIV.f_idiv(II)I id:M67_S_E1*/
M66_I2_E1_S: fcmM66_I2_E1_M67_S_E1 + fchM66_I2_E1_M67_S_E1 = fM66_I2_E1_M66_B3_E1; //cache S paths
M66_I2_E1_T: fM66_I2_E1_M66_B3_E1 = fM67_T_E1_M66_I2_E1; // invo T return path
fchM66_I2_E1_M67_S_E1 = 0; // no cache hits (because not innerloop)
/* Done with wcet.WCATestIDIV.f_idiv(II)I*/
M66_B3_E1: fM66_I2_E1_M66_B3_E1 + fM66_B3_E1_M66_B3_E1 = fM66_B3_E1_M66_T_E1;
M66_T_E1: fM66_B3_E1_M66_T_E1 = 1; // T flow
/* WCA flow to cycle count */
tM66_B1_E1 = 4 fM66_S_E1_M66_B1_E1 + 4 fM66_B1_E1_M66_B1_E1;
tM66_I2_E1 = 75 fM66_I2_E1_M66_I2_E1 + 75 fM66_B1_E1_M66_I2_E1;
tM66_B3_E1 = 22 fM66_I2_E1_M66_B3_E1 + 22 fM66_B3_E1_M66_B3_E1;
/* Invocation(s) from main:[M66] */
/* WCA flow constraints: f_idiv : M67 */
M67_S_E1: fchM66_I2_E1_M67_S_E1+ fcmM66_I2_E1_M67_S_E1 = fM67_S_E1_M67_B1_E1; // S flow
tM67_S_E1 = 29 fcmM66_I2_E1_M67_S_E1; // S cache miss time
M67_B1_E1: fM67_S_E1_M67_B1_E1 + fM67_B1_E1_M67_B1_E1 = fM67_B1_E1_M67_B2_E1 +
fM67_B1_E1_M67_B3_E1;
M67_B2_E1: fM67_B2_E1_M67_B2_E1 + fM67_B1_E1_M67_B2_E1 = fM67_B2_E1_M67_T_E1;
M67_B3_E1: fM67_B1_E1_M67_B3_E1 = fM67_B3_E1_M67_B4_E1 + fM67_B3_E1_M67_B5_E1;
M67_B4_E1: fM67_B4_E1_M67_B4_E1 + fM67_B3_E1_M67_B4_E1 = fM67_B4_E1_M67_B5_E1;
M67_B5_E1: fM67_B4_E1_M67_B5_E1 + fM67_B3_E1_M67_B5_E1 = fM67_B5_E1_M67_B6_E1 +
fM67_B5_E1_M67_B10_E1;
M67_B6_E1: fM67_B5_E1_M67_B6_E1 + fM67_B6_E1_M67_B6_E1 = fM67_B6_E1_M67_B7_E1 +
fM67_B6_E1_M67_B8_E1;
M67_B7_E1: fM67_B7_E1_M67_B7_E1 + fM67_B6_E1_M67_B7_E1 = fM67_B7_E1_M67_B9_E1;
M67_B8_E1: fM67_B6_E1_M67_B8_E1 = fM67_B8_E1_M67_B9_E1;
M67_B9_E1: fM67_B7_E1_M67_B9_E1 + fM67_B8_E1_M67_B9_E1 = fM67_B9_E1_M67_B10_E1;
M67_B10_E1: fM67_B9_E1_M67_B10_E1 + fM67_B5_E1_M67_B10_E1 = fM67_B10_E1_M67_B11_E1;
M67_B11_E1: fM67_B10_E1_M67_B11_E1 + fM67_B16_E1_M67_B11_E1 = fM67_B11_E1_M67_B12_E1 +
fM67_B11_E1_M67_B17_E1;
LC_M67_B11_E1: fM67_B11_E1_M67_B12_E1 = 32 fM67_B5_E1_M67_B10_E1;
M67_B12_E1: fM67_B12_E1_M67_B12_E1 + fM67_B11_E1_M67_B12_E1 = fM67_B12_E1_M67_B13_E1 +
fM67_B12_E1_M67_B14_E1;
M67_B13_E1: fM67_B12_E1_M67_B13_E1 + fM67_B13_E1_M67_B13_E1 = fM67_B13_E1_M67_B14_E1;
M67_B14_E1: fM67_B12_E1_M67_B14_E1 + fM67_B13_E1_M67_B14_E1 = fM67_B14_E1_M67_B15_E1 +
fM67_B14_E1_M67_B16_E1;
M67_B15_E1: fM67_B14_E1_M67_B15_E1 + fM67_B15_E1_M67_B15_E1 = fM67_B15_E1_M67_B16_E1;
M67_B16_E1: fM67_B14_E1_M67_B16_E1 + fM67_B15_E1_M67_B16_E1 = fM67_B16_E1_M67_B11_E1;
M67_B17_E1: fM67_B11_E1_M67_B17_E1 = fM67_B17_E1_M67_B18_E1 + fM67_B17_E1_M67_B19_E1;
M67_B18_E1: fM67_B17_E1_M67_B18_E1 + fM67_B18_E1_M67_B18_E1 = fM67_B18_E1_M67_B19_E1;
M67_B19_E1: fM67_B17_E1_M67_B19_E1 + fM67_B18_E1_M67_B19_E1 = fM67_B19_E1_M67_T_E1;
M67_T_E1: fM67_B19_E1_M67_T_E1 + fM67_B2_E1_M67_T_E1 = fM67_T_E1_M66_I2_E1;// T interconnect flow
// tM67_T_E1 = 0 fM67_T_E1_M66_I2_E1; // T cache hit (leaf)
/* WCA flow to cycle count */
tM67_B1_E1 = 5 fM67_S_E1_M67_B1_E1 + 5 fM67_B1_E1_M67_B1_E1;
tM67_B2_E1 = 33 fM67_B2_E1_M67_B2_E1 + 33 fM67_B1_E1_M67_B2_E1;
tM67_B3_E1 = 7 fM67_B1_E1_M67_B3_E1;

```
tM67_B4_E1 = 8 fM67_B4_E1_M67_B4_E1 + 8 fM67_B3_E1_M67_B4_E1;
tM67_B5_E1 = 5 fM67_B4_E1_M67_B5_E1 + 5 fM67_B3_E1_M67_B5_E1;
tM67_B6_E1 = 5 fM67_B5_E1_M67_B6_E1 + 5 fM67_B6_E1_M67_B6_E1;
tM67_B7_E1 = 5 fM67_B7_E1_M67_B7_E1 + 5 fM67_B6_E1_M67_B7_E1;
tM67_B8_E1 = 1 fM67_B6_E1_M67_B8_E1;
tM67_B9_E1 = 7 fM67_B7_E1_M67_B9_E1 + 7 fM67_B8_E1_M67_B9_E1;
tM67_B10_E1 = 8 fM67_B9_E1_M67_B10_E1 + 8 fM67_B5_E1_M67_B10_E1;
tM67_B11_E1 = 8 fM67_B10_E1_M67_B11_E1 + 8 fM67_B16_E1_M67_B11_E1;
tM67_B12_E1 = 24 fM67_B12_E1_M67_B12_E1 + 24 fM67_B11_E1_M67_B12_E1;
tM67_B13_E1 = 6 fM67_B12_E1_M67_B13_E1 + 6 fM67_B13_E1_M67_B13_E1;
tM67_B14_E1 = 11 fM67_B12_E1_M67_B14_E1 + 11 fM67_B13_E1_M67_B14_E1;
tM67_B15_E1 = 10 fM67_B14_E1_M67_B15_E1 + 10 fM67_B15_E1_M67_B15_E1;
tM67_B16_E1 = 12 fM67_B14_E1_M67_B16_E1 + 12 fM67_B15_E1_M67_B16_E1;
tM67_B17_E1 = 5 fM67_B11_E1_M67_B17_E1;
tM67_B18_E1 = 6 fM67_B17_E1_M67_B18_E1 + 6 fM67_B18_E1_M67_B18_E1;
tM67_B19_E1 = 24 fM67_B17_E1_M67_B19_E1 + 24 fM67_B18_E1_M67_B19_E1;
/* Invocation(s) from f_idiv:[M67] */
```

Value of objective function: 2478

Actual values of the variables:

tM66_S_E1	0
tM66_B1_E1	4
tM66_I2_E1	75
tM66_B3_E1	22
tM66_T_E1	0
tM67_S_E1	29
tM67_B1_E1	5
tM67_B2_E1	0
tM67_B3_E1	7
tM67_B4_E1	8
tM67_B5_E1	5
tM67_B6_E1	0
tM67_B7_E1	0
tM67_B8_E1	0
tM67_B9_E1	0
tM67_B10_E1	8
tM67_B11_E1	264
tM67_B12_E1	768
tM67_B13_E1	192
tM67_B14_E1	352
tM67_B15_E1	320
tM67_B16_E1	384
tM67_B17_E1	5
tM67_B18_E1	6
tM67_B19_E1	24
tM67_T_E1	0
fM66_S_E1_M66_B1_E1	1
fM66_B1_E1_M66_B1_E1	0
fM66_B1_E1_M66_I2_E1	1
fM66_I2_E1_M66_I2_E1	0
fM66_I2_E1_M66_B3_E1	1
fcmM66_I2_E1_M67_S_E1	1
fchM66_I2_E1_M67_S_E1	0
fM67_T_E1_M66_I2_E1	1
fM66_B3_E1_M66_B3_E1	0
fM66_B3_E1_M66_T_E1	1
fM67_S_E1_M67_B1_E1	1

fM67_B1_E1_M67_B1_E1 0
fM67_B1_E1_M67_B2_E1 0
fM67_B1_E1_M67_B3_E1 1
fM67_B2_E1_M67_B2_E1 0
fM67_B2_E1_M67_T_E1 0
fM67_B3_E1_M67_B4_E1 1
fM67_B3_E1_M67_B5_E1 0
fM67_B4_E1_M67_B4_E1 0
fM67_B4_E1_M67_B5_E1 1
fM67_B5_E1_M67_B6_E1 0
fM67_B5_E1_M67_B10_E1 1
fM67_B6_E1_M67_B6_E1 0
fM67_B6_E1_M67_B7_E1 0
fM67_B6_E1_M67_B8_E1 0
fM67_B7_E1_M67_B7_E1 0
fM67_B7_E1_M67_B9_E1 0
fM67_B8_E1_M67_B9_E1 0
fM67_B9_E1_M67_B10_E1 0
fM67_B10_E1_M67_B11_E1 1
fM67_B16_E1_M67_B11_E1 32
fM67_B11_E1_M67_B12_E1 32
fM67_B11_E1_M67_B17_E1 1
fM67_B12_E1_M67_B12_E1 0
fM67_B12_E1_M67_B13_E1 32
fM67_B12_E1_M67_B14_E1 0
fM67_B13_E1_M67_B13_E1 0
fM67_B13_E1_M67_B14_E1 32
fM67_B14_E1_M67_B15_E1 32
fM67_B14_E1_M67_B16_E1 0
fM67_B15_E1_M67_B15_E1 0
fM67_B15_E1_M67_B16_E1 32
fM67_B17_E1_M67_B18_E1 1
fM67_B17_E1_M67_B19_E1 0
fM67_B18_E1_M67_B18_E1 0
fM67_B18_E1_M67_B19_E1 1
fM67_B19_E1_M67_T_E1 1

**********END APPLICATION WCET**********

```
********************************************************************************
WCET info for:wcet.WCATestIDIV.f_idiv(II)I

Directed graph of basic blocks(row->column):
================================================================================================================================================
M67_S_E0M67_B1_E0M67_B2_E0M67_B3_E0M67_B4_E0M67_B5_E0M67_B6_E0M67_B7_E0M67_B8_E0M67_B9
_E0M67_B10_E0M67_B11_E0M67_B12_E0M67_B13_E0M67_B14_E0M67_B15_E0M67_B16_E0M67_B17_E0M67_
B18_E0M67_B19_E0M67_T_E0
M67_S_E0      . 1 . . . . . .
M67_B1_E0     . . 1 1 . . . . . .
M67_B2_E0     . . . . . 1
M67_B3_E0     . . . . 1 1 . . . . . .
M67_B4_E0     . . . . . 1 . . . . . .
M67_B5_E0     . . . . . 1 . . . 1 . . . . . .
M67_B6_E0     . . . . . 1 1 . . . . . .
M67_B7_E0     . . . . . 1 . . . . . .
M67_B8_E0     . . . . . 1 . . . . . .
M67_B9_E0     . . . . . 1 . . . . . .
M67_B10_E0    . . . . . 1 . . . . . .
M67_B11_E0    . . . . . 1 . . . . 1 . . .
M67_B12_E0    . . . . . 1 1 . . . . . .
M67_B13_E0    . . . . . 1 . . . . . .
M67_B14_E0    . . . . . 1 1 . . . .
M67_B15_E0    . . . . . 1 . . . .
M67_B16_E0    . . . . . 1 . . . . . .
M67_B17_E0    . . . . . 1 1 .
M67_B18_E0    . . . . . 1 .
M67_B19_E0    . . . . . 1
M67_T_E0      . . . . . .
================================================================================================================================================
========================

Table of basic blocks' and instructions
================================================================================
Block Addr.  Bytecode              Cycles    Cache miss     Misc. info
             [opcode]              invoke  return
--------------------------------------------------------------------------------
M67_S_E0'S'
  Src. line 22: if (b == 0) {
M67_B1_E0'B'{}<-[M67_S_E0 M67_B1_E0]0:     iload_1[27]               1               I:b
     1:     ifne[154]->12:             4               sum(B1):      5
  Src. line 24: Native.wrMem(0x02, Const.IO_EXCPT);
M67_B2_E0'B'{}<-[M67_B2_E0 M67_B1_E0]4:     iconst_2[5]               1
     5:     bipush[16]               2
     7:     invokestatic[184]          6               com.jopdesign.sys.Native.wrMem(II)V
  Src. line 25: return 0;
    10:     iconst_0[3]              1
    11:     ireturn[172]             23              sum(B2):     33
  Src. line 27: boolean neg = false;
M67_B3_E0'B'{}<-[M67_B1_E0]12:    iconst_0[3]               1
    13:     istore_2[61]             1               ->Z:neg
  Src. line 28: if (a < 0) {
    14:     iload_0[26]              1               I:a
    15:     ifge[156]->23:             4               sum(B3):      7
  Src. line 29: neg = true;
```

```
M67_B4_E0'B'{}<-[M67_B4_E0 M67_B3_E0]18:    iconst_1[4]              1
    19:    istore_2[61]             1              ->Z:neg
 Src. line 30: a = -a;
    20:    iload_0[26]              1              I:a
    21:    ineg[116]              4
    22:    istore_0[59]             1              ->I:a sum(B4):      8
 Src. line 32: if (b < 0) {
M67_B5_E0'B'{}<-[M67_B4_E0 M67_B3_E0]23:    iload_1[27]              1              I:b
    24:    ifge[156]->40:           4              sum(B5):      5
 Src. line 33: neg = !neg;
M67_B6_E0'B'{}<-[M67_B5_E0 M67_B6_E0]27:    iload_2[28]              1              Z:neg
    28:    ifne[154]->35:           4              sum(B6):      5
 Src. line 33: neg = !neg;
M67_B7_E0'B'{}<-[M67_B7_E0 M67_B6_E0]31:    iconst_1[4]              1
    32:    goto[167]->36:           4              sum(B7):      5
 Src. line 33: neg = !neg;
M67_B8_E0'B'{}<-[M67_B6_E0]35:    iconst_0[3]              1              sum(B8):      1
 Src. line 33: neg = !neg;
M67_B9_E0'B'{}<-[M67_B7_E0 M67_B8_E0]36:    istore_2[61]             1              ->Z:neg
 Src. line 34: b = -b;
    37:    iload_1[27]              1              I:b
    38:    ineg[116]              4
    39:    istore_1[60]             1              ->I:b sum(B9):      7
 Src. line 37: int c = 0;
M67_B10_E0'B'{ld}<-[M67_B9_E0 M67_B5_E0]40:    iconst_0[3]              1
    41:    istore_3[62]             1              ->I:c
 Src. line 38: int r = 0;
    42:    iconst_0[3]              1
    43:    istore[54]             2              ->I:r
Annotated Src. line :39: for (int i = 0; i < 32; ++i) // @WCA loop=32
    45:    iconst_0[3]              1
    46:    istore[54]             2              ->I:i sum(B10):      8
Annotated Src. line :39: for (int i = 0; i < 32; ++i) // @WCA loop=32
M67_B11_E0'B'{lc(ld=10)}<-[M67_B10_E0 M67_B16_E0]48:    iload[21]              2              I:i
    50:    bipush[16]             2
    52:    if_icmpge[162]->104:       4              sum(B11):      8
 Src. line 41: c <<= 1;
M67_B12_E0'B'{}<-[M67_B12_E0 M67_B11_E0]55:    iload_3[29]              1              I:c
    56:    iconst_1[4]              1
    57:    ishl[120]              1
    58:    istore_3[62]             1              ->I:c
 Src. line 42: r <<= 1;
    59:    iload[21]              2              I:r
    61:    iconst_1[4]              1
    62:    ishl[120]              1
    63:    istore[54]             2              ->I:r
 Src. line 43: if ((a & 0x80000000) != 0) {
    65:    iload_0[26]              1              I:a
    66:    ldc[18]              8
    68:    iand[126]              1
    69:    ifeq[153]->78:           4              sum(B12):     24
 Src. line 44: r |= 1;
M67_B13_E0'B'{}<-[M67_B12_E0 M67_B13_E0]72:    iload[21]              2              I:r
    74:    iconst_1[4]              1
    75:    ior[128]               1
    76:    istore[54]             2              ->I:r sum(B13):      6
 Src. line 46: a <<= 1;
M67_B14_E0'B'{}<-[M67_B12_E0 M67_B13_E0]78:    iload_0[26]              1              I:a
```

```
    79:   iconst_1[4]              1
    80:   ishl[120]               1
    81:   istore_0[59]             1              ->I:a
  Src. line 47: if (r >= b) {
    82:   iload[21]               2              I:r
    84:   iload_1[27]              1              I:b
    85:   if_icmplt[161]->98:      4              sum(B14):    11
  Src. line 48: r -= b;
M67_B15_E0'B'{}<-[M67_B14_E0 M67_B15_E0]88:   iload[21]              2              I:r
    90:   iload_1[27]              1              I:b
    91:   isub[100]               1
    92:   istore[54]              2              ->I:r
  Src. line 49: c |= 1;
    94:   iload_3[29]              1              I:c
    95:   iconst_1[4]              1
    96:   ior[128]                1
    97:   istore_3[62]             1              ->I:c sum(B15):    10
Annotated Src. line :39: for (int i = 0; i < 32; ++i) // @WCA loop=32
M67_B16_E0'B'{}<-[M67_B14_E0 M67_B15_E0]98:   iinc[132]              8              I:i
   101:   goto[167]->48:          4              sum(B16):    12
  Src. line 53: if (neg) {
M67_B17_E0'B'{}<-[M67_B11_E0]104:   iload_2[28]              1              Z:neg
   105:   ifeq[153]->111:         4              sum(B17):    5
  Src. line 54: c = -c;
M67_B18_E0'B'{}<-[M67_B17_E0 M67_B18_E0]108:   iload_3[29]              1              I:c
   109:   ineg[116]              4
   110:   istore_3[62]            1              ->I:c sum(B18):    6
  Src. line 56: return c;
M67_B19_E0'B'{}<-[M67_B17_E0 M67_B18_E0]111:   iload_3[29]              1              I:c
   112:   ireturn[172]           23             sum(B19):    24
M67_T_E0'T'<-[M67_B19_E0 M67_B2_E0]
================================================================================
Info: n=29 a=-1 r=1 w=1

/***WCET calculation source***/
/* WCA WCET objective: wcet.WCATestIDIV.f_idiv */
max: tM67_S_E0 tM67_B1_E0 tM67_B2_E0 tM67_B3_E0 tM67_B4_E0 tM67_B5_E0 tM67_B6_E0 tM67_B7_E0
tM67_B8_E0 tM67_B9_E0 tM67_B10_E0 tM67_B11_E0 tM67_B12_E0 tM67_B13_E0 tM67_B14_E0 tM67_B15_E0
tM67_B16_E0 tM67_B17_E0 tM67_B18_E0 tM67_B19_E0 tM67_T_E0;
/* WCA flow constraints: f_idiv : M67 */
M67_S_E0: 1 = fM67_S_E0_M67_B1_E0; // S flow
M67_B1_E0: fM67_S_E0_M67_B1_E0 + fM67_B1_E0_M67_B1_E0 = fM67_B1_E0_M67_B2_E0 +
fM67_B1_E0_M67_B3_E0;
M67_B2_E0: fM67_B2_E0_M67_B2_E0 + fM67_B1_E0_M67_B2_E0 = fM67_B2_E0_M67_T_E0;
M67_B3_E0: fM67_B1_E0_M67_B3_E0 = fM67_B3_E0_M67_B4_E0 + fM67_B3_E0_M67_B5_E0;
M67_B4_E0: fM67_B4_E0_M67_B4_E0 + fM67_B3_E0_M67_B4_E0 = fM67_B4_E0_M67_B5_E0;
M67_B5_E0: fM67_B4_E0_M67_B5_E0 + fM67_B3_E0_M67_B5_E0 = fM67_B5_E0_M67_B6_E0 +
fM67_B5_E0_M67_B10_E0;
M67_B6_E0: fM67_B5_E0_M67_B6_E0 + fM67_B6_E0_M67_B6_E0 = fM67_B6_E0_M67_B7_E0 +
fM67_B6_E0_M67_B8_E0;
M67_B7_E0: fM67_B7_E0_M67_B7_E0 + fM67_B6_E0_M67_B7_E0 = fM67_B7_E0_M67_B9_E0;
M67_B8_E0: fM67_B6_E0_M67_B8_E0 = fM67_B8_E0_M67_B9_E0;
M67_B9_E0: fM67_B7_E0_M67_B9_E0 + fM67_B8_E0_M67_B9_E0 = fM67_B9_E0_M67_B10_E0;
M67_B10_E0: fM67_B9_E0_M67_B10_E0 + fM67_B5_E0_M67_B10_E0 = fM67_B10_E0_M67_B11_E0;
M67_B11_E0: fM67_B10_E0_M67_B11_E0 + fM67_B16_E0_M67_B11_E0 = fM67_B11_E0_M67_B12_E0 +
fM67_B11_E0_M67_B17_E0;
LC_M67_B11_E0: fM67_B11_E0_M67_B12_E0 = 32 fM67_B5_E0_M67_B10_E0;
M67_B12_E0: fM67_B12_E0_M67_B12_E0 + fM67_B11_E0_M67_B12_E0 = fM67_B12_E0_M67_B13_E0 +
```

fM67_B12_E0_M67_B14_E0;
M67_B13_E0: fM67_B12_E0_M67_B13_E0 + fM67_B13_E0_M67_B13_E0 = fM67_B13_E0_M67_B14_E0;
M67_B14_E0: fM67_B12_E0_M67_B14_E0 + fM67_B13_E0_M67_B14_E0 = fM67_B14_E0_M67_B15_E0 +
fM67_B14_E0_M67_B16_E0;
M67_B15_E0: fM67_B14_E0_M67_B15_E0 + fM67_B15_E0_M67_B15_E0 = fM67_B15_E0_M67_B16_E0;
M67_B16_E0: fM67_B14_E0_M67_B16_E0 + fM67_B15_E0_M67_B16_E0 = fM67_B16_E0_M67_B11_E0;
M67_B17_E0: fM67_B11_E0_M67_B17_E0 = fM67_B17_E0_M67_B18_E0 + fM67_B17_E0_M67_B19_E0;
M67_B18_E0: fM67_B17_E0_M67_B18_E0 + fM67_B18_E0_M67_B18_E0 = fM67_B18_E0_M67_B19_E0;
M67_B19_E0: fM67_B17_E0_M67_B19_E0 + fM67_B18_E0_M67_B19_E0 = fM67_B19_E0_M67_T_E0;
M67_T_E0: fM67_B19_E0_M67_T_E0 + fM67_B2_E0_M67_T_E0 = 1; // T flow
/* WCA flow to cycle count */
tM67_B1_E0 = 5 fM67_S_E0_M67_B1_E0 + 5 fM67_B1_E0_M67_B1_E0;
tM67_B2_E0 = 33 fM67_B2_E0_M67_B2_E0 + 33 fM67_B1_E0_M67_B2_E0;
tM67_B3_E0 = 7 fM67_B1_E0_M67_B3_E0;
tM67_B4_E0 = 8 fM67_B4_E0_M67_B4_E0 + 8 fM67_B3_E0_M67_B4_E0;
tM67_B5_E0 = 5 fM67_B4_E0_M67_B5_E0 + 5 fM67_B3_E0_M67_B5_E0;
tM67_B6_E0 = 5 fM67_B5_E0_M67_B6_E0 + 5 fM67_B6_E0_M67_B6_E0;
tM67_B7_E0 = 5 fM67_B7_E0_M67_B7_E0 + 5 fM67_B6_E0_M67_B7_E0;
tM67_B8_E0 = 1 fM67_B6_E0_M67_B8_E0;
tM67_B9_E0 = 7 fM67_B7_E0_M67_B9_E0 + 7 fM67_B8_E0_M67_B9_E0;
tM67_B10_E0 = 8 fM67_B9_E0_M67_B10_E0 + 8 fM67_B5_E0_M67_B10_E0;
tM67_B11_E0 = 8 fM67_B10_E0_M67_B11_E0 + 8 fM67_B16_E0_M67_B11_E0;
tM67_B12_E0 = 24 fM67_B12_E0_M67_B12_E0 + 24 fM67_B11_E0_M67_B12_E0;
tM67_B13_E0 = 6 fM67_B12_E0_M67_B13_E0 + 6 fM67_B13_E0_M67_B13_E0;
tM67_B14_E0 = 11 fM67_B12_E0_M67_B14_E0 + 11 fM67_B13_E0_M67_B14_E0;
tM67_B15_E0 = 10 fM67_B14_E0_M67_B15_E0 + 10 fM67_B15_E0_M67_B15_E0;
tM67_B16_E0 = 12 fM67_B14_E0_M67_B16_E0 + 12 fM67_B15_E0_M67_B16_E0;
tM67_B17_E0 = 5 fM67_B11_E0_M67_B17_E0;
tM67_B18_E0 = 6 fM67_B17_E0_M67_B18_E0 + 6 fM67_B18_E0_M67_B18_E0;
tM67_B19_E0 = 24 fM67_B17_E0_M67_B19_E0 + 24 fM67_B18_E0_M67_B19_E0;
/* Invocation(s) from f_idiv:[M67] */

Value of objective function: 2348

Actual values of the variables:

tM67_S_E0	0
tM67_B1_E0	5
tM67_B2_E0	0
tM67_B3_E0	7
tM67_B4_E0	8
tM67_B5_E0	5
tM67_B6_E0	0
tM67_B7_E0	0
tM67_B8_E0	0
tM67_B9_E0	0
tM67_B10_E0	8
tM67_B11_E0	264
tM67_B12_E0	768
tM67_B13_E0	192
tM67_B14_E0	352
tM67_B15_E0	320
tM67_B16_E0	384
tM67_B17_E0	5
tM67_B18_E0	6
tM67_B19_E0	24
tM67_T_E0	0
fM67_S_E0_M67_B1_E0	1
fM67_B1_E0_M67_B1_E0	0

```
fM67_B1_E0_M67_B2_E0          0
fM67_B1_E0_M67_B3_E0          1
fM67_B2_E0_M67_B2_E0          0
fM67_B2_E0_M67_T_E0           0
fM67_B3_E0_M67_B4_E0          1
fM67_B3_E0_M67_B5_E0          0
fM67_B4_E0_M67_B4_E0          0
fM67_B4_E0_M67_B5_E0          1
fM67_B5_E0_M67_B6_E0          0
fM67_B5_E0_M67_B10_E0         1
fM67_B6_E0_M67_B6_E0          0
fM67_B6_E0_M67_B7_E0          0
fM67_B6_E0_M67_B8_E0          0
fM67_B7_E0_M67_B7_E0          0
fM67_B7_E0_M67_B9_E0          0
fM67_B8_E0_M67_B9_E0          0
fM67_B9_E0_M67_B10_E0         0
fM67_B10_E0_M67_B11_E0        1
fM67_B16_E0_M67_B11_E0       32
fM67_B11_E0_M67_B12_E0       32
fM67_B11_E0_M67_B17_E0        1
fM67_B12_E0_M67_B12_E0        0
fM67_B12_E0_M67_B13_E0       32
fM67_B12_E0_M67_B14_E0        0
fM67_B13_E0_M67_B13_E0        0
fM67_B13_E0_M67_B14_E0       32
fM67_B14_E0_M67_B15_E0       32
fM67_B14_E0_M67_B16_E0        0
fM67_B15_E0_M67_B15_E0        0
fM67_B15_E0_M67_B16_E0       32
fM67_B17_E0_M67_B18_E0        1
fM67_B17_E0_M67_B19_E0        0
fM67_B18_E0_M67_B18_E0        0
fM67_B18_E0_M67_B19_E0        1
fM67_B19_E0_M67_T_E0          1

/*wcet.WCATestIDIV.f_idiv(II)I*/
digraph G {
size = "10,7.5"
        M67_S_E0->M67_B1_E0 [label="fM67_S_E0_M67_B1_E0=1"];
        M67_B1_E0->M67_B2_E0 [style=dashed,label="fM67_B1_E0_M67_B2_E0=0"];
        M67_B1_E0->M67_B3_E0 [label="fM67_B1_E0_M67_B3_E0=1"];
        M67_B2_E0->M67_T_E0 [style=dashed,label="fM67_B2_E0_M67_T_E0=0"];
        M67_B3_E0->M67_B4_E0 [label="fM67_B3_E0_M67_B4_E0=1"];
        M67_B3_E0->M67_B5_E0 [style=dashed,label="fM67_B3_E0_M67_B5_E0=0"];
        M67_B4_E0->M67_B5_E0 [label="fM67_B4_E0_M67_B5_E0=1"];
        M67_B5_E0->M67_B6_E0 [style=dashed,label="fM67_B5_E0_M67_B6_E0=0"];
        M67_B5_E0->M67_B10_E0 [label="fM67_B5_E0_M67_B10_E0=1"];
        M67_B6_E0->M67_B7_E0 [style=dashed,label="fM67_B6_E0_M67_B7_E0=0"];
        M67_B6_E0->M67_B8_E0 [style=dashed,label="fM67_B6_E0_M67_B8_E0=0"];
        M67_B7_E0->M67_B9_E0 [style=dashed,label="fM67_B7_E0_M67_B9_E0=0"];
        M67_B8_E0->M67_B9_E0 [style=dashed,label="fM67_B8_E0_M67_B9_E0=0"];
        M67_B9_E0->M67_B10_E0 [style=dashed,label="fM67_B9_E0_M67_B10_E0=0"];
        M67_B10_E0->M67_B11_E0 [label="fM67_B10_E0_M67_B11_E0=1"];
        M67_B11_E0->M67_B12_E0 [label="fM67_B11_E0_M67_B12_E0=32"];
        M67_B11_E0->M67_B17_E0 [label="fM67_B11_E0_M67_B17_E0=1"];
        M67_B12_E0->M67_B13_E0 [label="fM67_B12_E0_M67_B13_E0=32"];
        M67_B12_E0->M67_B14_E0 [style=dashed,label="fM67_B12_E0_M67_B14_E0=0"];
```

```
        M67_B13_E0->M67_B14_E0 [label="fM67_B13_E0_M67_B14_E0=32"];
        M67_B14_E0->M67_B15_E0 [label="fM67_B14_E0_M67_B15_E0=32"];
        M67_B14_E0->M67_B16_E0 [style=dashed,label="fM67_B14_E0_M67_B16_E0=0"];
        M67_B15_E0->M67_B16_E0 [label="fM67_B15_E0_M67_B16_E0=32"];
        M67_B16_E0->M67_B11_E0 [label="fM67_B16_E0_M67_B11_E0=32"];
        M67_B17_E0->M67_B18_E0 [label="fM67_B17_E0_M67_B18_E0=1"];
        M67_B17_E0->M67_B19_E0 [style=dashed,label="fM67_B17_E0_M67_B19_E0=0"];
        M67_B18_E0->M67_B19_E0 [label="fM67_B18_E0_M67_B19_E0=1"];
        M67_B19_E0->M67_T_E0 [label="fM67_B19_E0_M67_T_E0=1"];
        M67_S_E0;
        M67_B1_E0 [label="M67_B1_E0\n5"];
        M67_B2_E0 [label="M67_B2_E0\n102"];
        M67_B3_E0 [label="M67_B3_E0\n7"];
        M67_B4_E0 [label="M67_B4_E0\n8"];
        M67_B5_E0 [label="M67_B5_E0\n5"];
        M67_B6_E0 [label="M67_B6_E0\n5"];
        M67_B7_E0 [label="M67_B7_E0\n5"];
        M67_B8_E0 [label="M67_B8_E0\n1"];
        M67_B9_E0 [label="M67_B9_E0\n7"];
        M67_B10_E0 [label="M67_B10_E0\n8"];
        M67_B11_E0 [label="M67_B11_E0\n8"];
        M67_B12_E0 [label="M67_B12_E0\n24"];
        M67_B13_E0 [label="M67_B13_E0\n6"];
        M67_B14_E0 [label="M67_B14_E0\n11"];
        M67_B15_E0 [label="M67_B15_E0\n10"];
        M67_B16_E0 [label="M67_B16_E0\n12"];
        M67_B17_E0 [label="M67_B17_E0\n5"];
        M67_B18_E0 [label="M67_B18_E0\n6"];
        M67_B19_E0 [label="M67_B19_E0\n24"];
        M67_T_E0;
}
********************************************************************************
WCET info for:wcet.WCATestIDIV.main([Ljava/lang/String;)V

Directed graph of basic blocks(row->column):
=============================================
   M66_S_E0M66_B1_E0M66_I2_E0M66_B3_E0M66_T_E0
M66_S_E0 .  1  .  .  .
M66_B1_E0 .  .  1  .  .
M66_I2_E0 .  .  .  1  .
M66_B3_E0 .  .  .  .  1
M66_T_E0 .  .  .  .  .
=============================================

Table of basic blocks' and instructions
================================================================================
Block Addr.  Bytecode              Cycles    Cache miss     Misc. info
          [opcode]                 invoke  return
--------------------------------------------------------------------------------
M66_S_E0'S'
  Src. line 15: f_idiv(-2, -2);
M66_B1_E0'B'{}<-[M66_S_E0 M66_B1_E0]0:     bipush[16]              2
     2:     bipush[16]              2                sum(B1):      4
  Src. line 15: f_idiv(-2, -2);
M66_I2_E0'I'{}<-[M66_I2_E0 M66_B1_E0]4:     invokestatic[184]          75      29      4   wcet.WCATestIDIV.f_idiv(II)I,
invoke(n=29):75/104 return(n=3):23/27sum(B2):     75
  Src. line 15: f_idiv(-2, -2);
M66_B3_E0'B'{}<-[M66_I2_E0 M66_B3_E0]7:     pop[87]                 1
```

```
  Src. line 18: }
     8:     return[177]              21                sum(B3):     22
M66_T_E0'T'<-[M66_B3_E0]
================================================================================
Info: n=3 a=-1 r=1 w=1

/***WCET calculation source***/
/* WCA WCET objective: wcet.WCATestIDIV.main */
max: tM66_S_E0 tM66_B1_E0 tM66_I2_E0 tM66_B3_E0 tM66_T_E0;
/* WCA flow constraints: main : M66 */
M66_S_E0: 1 = fM66_S_E0_M66_B1_E0; // S flow
M66_B1_E0: fM66_S_E0_M66_B1_E0 + fM66_B1_E0_M66_B1_E0 = fM66_B1_E0_M66_I2_E0;
M66_I2_E0: fM66_I2_E0_M66_I2_E0 + fM66_B1_E0_M66_I2_E0 = fM66_I2_E0_M66_B3_E0;
M66_B3_E0: fM66_I2_E0_M66_B3_E0 + fM66_B3_E0_M66_B3_E0 = fM66_B3_E0_M66_T_E0;
M66_T_E0: fM66_B3_E0_M66_T_E0 = 1; // T flow
/* WCA flow to cycle count */
tM66_B1_E0 = 4 fM66_S_E0_M66_B1_E0 + 4 fM66_B1_E0_M66_B1_E0;
tM66_I2_E0 = 75 fM66_I2_E0_M66_I2_E0 + 75 fM66_B1_E0_M66_I2_E0;
tM66_B3_E0 = 22 fM66_I2_E0_M66_B3_E0 + 22 fM66_B3_E0_M66_B3_E0;
/* Invocation(s) from main:[M66] */

Value of objective function: 101

Actual values of the variables:
tM66_S_E0                   0
tM66_B1_E0                  4
tM66_I2_E0                 75
tM66_B3_E0                 22
tM66_T_E0                   0
fM66_S_E0_M66_B1_E0         1
fM66_B1_E0_M66_B1_E0        0
fM66_B1_E0_M66_I2_E0        1
fM66_I2_E0_M66_I2_E0        0
fM66_I2_E0_M66_B3_E0        1
fM66_B3_E0_M66_B3_E0        0
fM66_B3_E0_M66_T_E0         1

/*wcet.WCATestIDIV.main([Ljava/lang/String;)V*/
digraph G {
size = "10,7.5"
        M66_S_E0->M66_B1_E0 [label="fM66_S_E0_M66_B1_E0=1"];
        M66_B1_E0->M66_I2_E0 [label="fM66_B1_E0_M66_I2_E0=1"];
        M66_I2_E0->M66_B3_E0 [label="fM66_I2_E0_M66_B3_E0=1"];
        M66_B3_E0->M66_T_E0 [label="fM66_B3_E0_M66_T_E0=1"];
        M66_S_E0;
        M66_B1_E0 [label="M66_B1_E0\n4"];
        M66_I2_E0 [label="M66_I2_E0\n75"];
        M66_B3_E0 [label="M66_B3_E0\n22"];
        M66_T_E0;
}

**************************************************
Note: Remember to keep WCETAnalyzer updated
each time a bytecode implementation is changed.
```

B.2.3 irem

**********APPLICATION WCET=1981**********
/***WCET calculation source***/
/* WCA WCET objective: wcet.WCATestIREM.main */
max: tM66_S_E1 tM66_B1_E1 tM66_I2_E1 tM66_B3_E1 tM66_T_E1 tM67_S_E1 tM67_B1_E1 tM67_B2_E1
tM67_B3_E1 tM67_B4_E1 tM67_B5_E1 tM67_B6_E1 tM67_B7_E1 tM67_B8_E1 tM67_B9_E1 tM67_B10_E1
tM67_B11_E1 tM67_B12_E1 tM67_B13_E1 tM67_B14_E1 tM67_B15_E1 tM67_B16_E1 tM67_T_E1;
/* WCA flow constraints: main : M66 */
M66_S_E1: 1 = fM66_S_E1_M66_B1_E1; // S flow
M66_B1_E1: fM66_S_E1_M66_B1_E1 + fM66_B1_E1_M66_B1_E1 = fM66_B1_E1_M66_I2_E1;
M66_I2_E1: fM66_I2_E1_M66_I2_E1 + fM66_B1_E1_M66_I2_E1 = fM66_I2_E1_M66_B3_E1;
/* Connecting(invoking) to wcet.WCATestIREM.f_irem(II)I id:M67_S_E1*/
M66_I2_E1_S: fcmM66_I2_E1_M67_S_E1 + fchM66_I2_E1_M67_S_E1 = fM66_I2_E1_M66_B3_E1; //cache S paths
M66_I2_E1_T: fM66_I2_E1_M66_B3_E1 = fM67_T_E1_M66_I2_E1; // invo T return path
fchM66_I2_E1_M67_S_E1 = 0; // no cache hits (because not innerloop)
/* Done with wcet.WCATestIREM.f_irem(II)I*/
M66_B3_E1: fM66_I2_E1_M66_B3_E1 + fM66_B3_E1_M66_B3_E1 = fM66_B3_E1_M66_T_E1;
M66_T_E1: fM66_B3_E1_M66_T_E1 = 1; // T flow
/* WCA flow to cycle count */
tM66_B1_E1 = 4 fM66_S_E1_M66_B1_E1 + 4 fM66_B1_E1_M66_B1_E1;
tM66_I2_E1 = 75 fM66_I2_E1_M66_I2_E1 + 75 fM66_B1_E1_M66_I2_E1;
tM66_B3_E1 = 22 fM66_I2_E1_M66_B3_E1 + 22 fM66_B3_E1_M66_B3_E1;
/* Invocation(s) from main:[M66] */
/* WCA flow constraints: f_irem : M67 */
M67_S_E1: fchM66_I2_E1_M67_S_E1+ fcmM66_I2_E1_M67_S_E1 = fM67_S_E1_M67_B1_E1; // S flow
tM67_S_E1 = 15 fcmM66_I2_E1_M67_S_E1; // S cache miss time
M67_B1_E1: fM67_S_E1_M67_B1_E1 + fM67_B1_E1_M67_B1_E1 = fM67_B1_E1_M67_B2_E1 +
fM67_B1_E1_M67_B3_E1;
M67_B2_E1: fM67_B2_E1_M67_B2_E1 + fM67_B1_E1_M67_B2_E1 = fM67_B2_E1_M67_T_E1;
M67_B3_E1: fM67_B1_E1_M67_B3_E1 = fM67_B3_E1_M67_B4_E1 + fM67_B3_E1_M67_B5_E1;
M67_B4_E1: fM67_B4_E1_M67_B4_E1 + fM67_B3_E1_M67_B4_E1 = fM67_B4_E1_M67_B5_E1;
M67_B5_E1: fM67_B4_E1_M67_B5_E1 + fM67_B3_E1_M67_B5_E1 = fM67_B5_E1_M67_B6_E1 +
fM67_B5_E1_M67_B7_E1;
M67_B6_E1: fM67_B5_E1_M67_B6_E1 + fM67_B6_E1_M67_B6_E1 = fM67_B6_E1_M67_B7_E1;
M67_B7_E1: fM67_B5_E1_M67_B7_E1 + fM67_B6_E1_M67_B7_E1 = fM67_B7_E1_M67_B8_E1;
M67_B8_E1: fM67_B7_E1_M67_B8_E1 + fM67_B13_E1_M67_B8_E1 = fM67_B8_E1_M67_B9_E1 +
fM67_B8_E1_M67_B14_E1;
LC_M67_B8_E1: fM67_B8_E1_M67_B9_E1 = 32 fM67_B5_E1_M67_B7_E1;
M67_B9_E1: fM67_B8_E1_M67_B9_E1 + fM67_B9_E1_M67_B9_E1 = fM67_B9_E1_M67_B10_E1 +
fM67_B9_E1_M67_B11_E1;
M67_B10_E1: fM67_B10_E1_M67_B10_E1 + fM67_B9_E1_M67_B10_E1 = fM67_B10_E1_M67_B11_E1;
M67_B11_E1: fM67_B10_E1_M67_B11_E1 + fM67_B9_E1_M67_B11_E1 = fM67_B11_E1_M67_B12_E1 +
fM67_B11_E1_M67_B13_E1;
M67_B12_E1: fM67_B11_E1_M67_B12_E1 + fM67_B12_E1_M67_B12_E1 = fM67_B12_E1_M67_B13_E1;
M67_B13_E1: fM67_B11_E1_M67_B13_E1 + fM67_B12_E1_M67_B13_E1 = fM67_B13_E1_M67_B8_E1;
M67_B14_E1: fM67_B8_E1_M67_B14_E1 = fM67_B14_E1_M67_B15_E1 + fM67_B14_E1_M67_B16_E1;
M67_B15_E1: fM67_B15_E1_M67_B15_E1 + fM67_B14_E1_M67_B15_E1 = fM67_B15_E1_M67_B16_E1;
M67_B16_E1: fM67_B15_E1_M67_B16_E1 + fM67_B14_E1_M67_B16_E1 = fM67_B16_E1_M67_T_E1;
M67_T_E1: fM67_B16_E1_M67_T_E1 + fM67_B2_E1_M67_T_E1 = fM67_T_E1_M66_I2_E1;// T interconnect flow
// tM67_T_E1 = 0 fM67_T_E1_M66_I2_E1; // T cache hit (leaf)
/* WCA flow to cycle count */
tM67_B1_E1 = 5 fM67_S_E1_M67_B1_E1 + 5 fM67_B1_E1_M67_B1_E1;
tM67_B2_E1 = 33 fM67_B2_E1_M67_B2_E1 + 33 fM67_B1_E1_M67_B2_E1;
tM67_B3_E1 = 7 fM67_B1_E1_M67_B3_E1;
tM67_B4_E1 = 8 fM67_B4_E1_M67_B4_E1 + 8 fM67_B3_E1_M67_B4_E1;
tM67_B5_E1 = 5 fM67_B4_E1_M67_B5_E1 + 5 fM67_B3_E1_M67_B5_E1;
tM67_B6_E1 = 6 fM67_B5_E1_M67_B6_E1 + 6 fM67_B6_E1_M67_B6_E1;
tM67_B7_E1 = 5 fM67_B5_E1_M67_B7_E1 + 5 fM67_B6_E1_M67_B7_E1;
tM67_B8_E1 = 8 fM67_B7_E1_M67_B8_E1 + 8 fM67_B13_E1_M67_B8_E1;

```
tM67_B9_E1 = 18 fM67_B8_E1_M67_B9_E1 + 18 fM67_B9_E1_M67_B9_E1;
tM67_B10_E1 = 4 fM67_B10_E1_M67_B10_E1 + 4 fM67_B9_E1_M67_B10_E1;
tM67_B11_E1 = 10 fM67_B10_E1_M67_B11_E1 + 10 fM67_B9_E1_M67_B11_E1;
tM67_B12_E1 = 4 fM67_B11_E1_M67_B12_E1 + 4 fM67_B12_E1_M67_B12_E1;
tM67_B13_E1 = 12 fM67_B11_E1_M67_B13_E1 + 12 fM67_B12_E1_M67_B13_E1;
tM67_B14_E1 = 5 fM67_B8_E1_M67_B14_E1;
tM67_B15_E1 = 6 fM67_B15_E1_M67_B15_E1 + 6 fM67_B14_E1_M67_B15_E1;
tM67_B16_E1 = 24 fM67_B15_E1_M67_B16_E1 + 24 fM67_B14_E1_M67_B16_E1;
/* Invocation(s) from f_irem:[M67] */
```

Value of objective function: 1981

Actual values of the variables:

tM66_S_E1	0
tM66_B1_E1	4
tM66_I2_E1	75
tM66_B3_E1	22
tM66_T_E1	0
tM67_S_E1	15
tM67_B1_E1	5
tM67_B2_E1	0
tM67_B3_E1	7
tM67_B4_E1	8
tM67_B5_E1	5
tM67_B6_E1	0
tM67_B7_E1	5
tM67_B8_E1	264
tM67_B9_E1	576
tM67_B10_E1	128
tM67_B11_E1	320
tM67_B12_E1	128
tM67_B13_E1	384
tM67_B14_E1	5
tM67_B15_E1	6
tM67_B16_E1	24
tM67_T_E1	0
fM66_S_E1_M66_B1_E1	1
fM66_B1_E1_M66_B1_E1	0
fM66_B1_E1_M66_I2_E1	1
fM66_I2_E1_M66_I2_E1	0
fM66_I2_E1_M66_B3_E1	1
fcmM66_I2_E1_M67_S_E1	1
fchM66_I2_E1_M67_S_E1	0
fM67_T_E1_M66_I2_E1	1
fM66_B3_E1_M66_B3_E1	0
fM66_B3_E1_M66_T_E1	1
fM67_S_E1_M67_B1_E1	1
fM67_B1_E1_M67_B1_E1	0
fM67_B1_E1_M67_B2_E1	0
fM67_B1_E1_M67_B3_E1	1
fM67_B2_E1_M67_B2_E1	0
fM67_B2_E1_M67_T_E1	0
fM67_B3_E1_M67_B4_E1	1
fM67_B3_E1_M67_B5_E1	0
fM67_B4_E1_M67_B4_E1	0
fM67_B4_E1_M67_B5_E1	1
fM67_B5_E1_M67_B6_E1	0
fM67_B5_E1_M67_B7_E1	1

```
fM67_B6_E1_M67_B6_E1          0
fM67_B6_E1_M67_B7_E1          0
fM67_B7_E1_M67_B8_E1          1
fM67_B13_E1_M67_B8_E1         32
fM67_B8_E1_M67_B9_E1         32
fM67_B8_E1_M67_B14_E1         1
fM67_B9_E1_M67_B9_E1          0
fM67_B9_E1_M67_B10_E1         32
fM67_B9_E1_M67_B11_E1         0
fM67_B10_E1_M67_B10_E1        0
fM67_B10_E1_M67_B11_E1        32
fM67_B11_E1_M67_B12_E1        32
fM67_B11_E1_M67_B13_E1        0
fM67_B12_E1_M67_B12_E1        0
fM67_B12_E1_M67_B13_E1        32
fM67_B14_E1_M67_B15_E1        1
fM67_B14_E1_M67_B16_E1        0
fM67_B15_E1_M67_B15_E1        0
fM67_B15_E1_M67_B16_E1        1
fM67_B16_E1_M67_T_E1          1

*************END APPLICATION WCET*******************
*****************************************************************************
WCET info for:wcet.WCATestIREM.f_irem(II)I

Directed graph of basic blocks(row->column):
=============================================================================================
=============================================================================
   M67_S_E0M67_B1_E0M67_B2_E0M67_B3_E0M67_B4_E0M67_B5_E0M67_B6_E0M67_B7_E0M67_B8_E0M67_
B9_E0M67_B10_E0M67_B11_E0M67_B12_E0M67_B13_E0M67_B14_E0M67_B15_E0M67_B16_E0M67_T_E0
M67_S_E0     . 1 . . . . . . . . . . . . . . . . .
M67_B1_E0    . . 1 1 . . . . . . . . . . . . . . .
M67_B2_E0    . . . . . . . . . . . . . . . . . . 1
M67_B3_E0    . . . . 1 1 . . . . . . . . . . . . .
M67_B4_E0    . . . . . 1 . . . . . . . . . . . . .
M67_B5_E0    . . . . . . 1 1 . . . . . . . . . . .
M67_B6_E0    . . . . . . . 1 . . . . . . . . . . .
M67_B7_E0    . . . . . . . . 1 . . . . . . . . . .
M67_B8_E0    . . . . . . . . . 1 . . . . 1 . . .
M67_B9_E0    . . . . . . . . . . 1 1 . . . . . . .
M67_B10_E0 . . . . . . . . . . . 1 . . . . . .
M67_B11_E0 . . . . . . . . . . . . 1 1 . . . .
M67_B12_E0 . . . . . . . . . . . . . 1 . . . .
M67_B13_E0 . . . . . . . . 1 . . . . . . . . .
M67_B14_E0 . . . . . . . . . . . . . . . . . 1 1 .
M67_B15_E0 . . . . . . . . . . . . . . . . . 1 .
M67_B16_E0 . . . . . . . . . . . . . . . . . . 1
M67_T_E0     . . . . . . . . . . . . . . . . . .
=============================================================================================
=============================================================================

Table of basic blocks' and instructions
=============================================================================
Block Addr.  Bytecode              Cycles    Cache miss     Misc. info
             [opcode]              invoke  return
-----------------------------------------------------------------------------
M67_S_E0'S'
  Src. line 20: if (b==0) {
```

```
M67_B1_E0'B'{}<-[M67_S_E0 M67_B1_E0]0:     iload_1[27]                1                I:b
    1:     ifne[154]->12:             4                sum(B1):      5
  Src. line 22: Native.wrMem(0x02, Const.IO_EXCPT);
M67_B2_E0'B'{}<-[M67_B2_E0 M67_B1_E0]4:     iconst_2[5]                1
    5:     bipush[16]                 2
    7:     invokestatic[184]          6                com.jopdesign.sys.Native.wrMem(II)V
  Src. line 23: return 0;
    10:    iconst_0[3]                1
    11:    ireturn[172]               23               sum(B2):      33
  Src. line 25: boolean neg = false;
M67_B3_E0'B'{}<-[M67_B1_E0]12:    iconst_0[3]                1
    13:    istore_2[61]               1                ->Z:neg
  Src. line 26: if (a<0) {
    14:    iload_0[26]                1                I:a
    15:    ifge[156]->23:             4                sum(B3):      7
  Src. line 27: neg = true;
M67_B4_E0'B'{}<-[M67_B4_E0 M67_B3_E0]18:    iconst_1[4]                1
    19:    istore_2[61]               1                ->Z:neg
  Src. line 28: a = -a;
    20:    iload_0[26]                1                I:a
    21:    ineg[116]                  4
    22:    istore_0[59]               1                ->I:a sum(B4):      8
  Src. line 30: if (b<0) {
M67_B5_E0'B'{}<-[M67_B4_E0 M67_B3_E0]23:    iload_1[27]                1                I:b
    24:    ifge[156]->30:             4                sum(B5):      5
  Src. line 31: b = -b;
M67_B6_E0'B'{}<-[M67_B5_E0 M67_B6_E0]27:    iload_1[27]                1                I:b
    28:    ineg[116]                  4
    29:    istore_1[60]               1                ->I:b sum(B6):      6
  Src. line 34: int r = 0;
M67_B7_E0'B'{ld}<-[M67_B5_E0 M67_B6_E0]30:    iconst_0[3]                1
    31:    istore_3[62]               1                ->I:r
Annotated Src. line :35: for (int i=0; i<32; ++i) //@WCA loop=32
    32:    iconst_0[3]                1
    33:    istore[54]                 2                ->I:i sum(B7):      5
Annotated Src. line :35: for (int i=0; i<32; ++i) //@WCA loop=32
M67_B8_E0'B'{lc(ld=7)}<-[M67_B7_E0 M67_B13_E0]35:    iload[21]                2                I:i
    37:    bipush[16]                 2
    39:    if_icmpge[162]->76:        4                sum(B8):      8
  Src. line 37: r <<= 1;
M67_B9_E0'B'{}<-[M67_B8_E0 M67_B9_E0]42:    iload_3[29]                1                I:r
    43:    iconst_1[4]                1
    44:    ishl[120]                  1
    45:    istore_3[62]               1                ->I:r
  Src. line 38: if ((a & 0x80000000)!=0) {
    46:    iload_0[26]                1                I:a
    47:    ldc[18]                    8
    49:    iand[126]                  1
    50:    ifeq[153]->57:             4                sum(B9):      18
  Src. line 39: r |= 1;
M67_B10_E0'B'{}<-[M67_B10_E0 M67_B9_E0]53:    iload_3[29]                1                I:r
    54:    iconst_1[4]                1
    55:    ior[128]                   1
    56:    istore_3[62]               1                ->I:r sum(B10):     4
  Src. line 41: a <<= 1;
M67_B11_E0'B'{}<-[M67_B10_E0 M67_B9_E0]57:    iload_0[26]                1                I:a
    58:    iconst_1[4]                1
    59:    ishl[120]                  1
```

```
    60:   istore_0[59]              1               ->I:a
  Src. line 42: if (r>=b) {
    61:   iload_3[29]              1               I:r
    62:   iload_1[27]              1               I:b
    63:   if_icmplt[161]->70:        4               sum(B11):    10
  Src. line 43: r -= b;
M67_B12_E0'B'{}<-[M67_B11_E0 M67_B12_E0]66:   iload_3[29]              1               I:r
    67:   iload_1[27]              1               I:b
    68:   isub[100]                1
    69:   istore_3[62]             1               ->I:r sum(B12):     4
Annotated Src. line :35: for (int i=0; i<32; ++i) //@WCA loop=32
M67_B13_E0'B'{}<-[M67_B11_E0 M67_B12_E0]70:   iinc[132]               8               I:i
    73:   goto[167]->35:           4               sum(B13):    12
  Src. line 47: if (neg) {
M67_B14_E0'B'{}<-[M67_B8_E0]76:   iload_2[28]              1               Z:neg
    77:   ifeq[153]->83:           4               sum(B14):     5
  Src. line 48: r = -r;
M67_B15_E0'B'{}<-[M67_B15_E0 M67_B14_E0]80:   iload_3[29]              1               I:r
    81:   ineg[116]               4
    82:   istore_3[62]             1               ->I:r sum(B15):     6
  Src. line 50: return r;
M67_B16_E0'B'{}<-[M67_B15_E0 M67_B14_E0]83:   iload_3[29]              1               I:r
    84:   ireturn[172]            23              sum(B16):    24
M67_T_E0'T'<-[M67_B16_E0 M67_B2_E0]
================================================================================
Info: n=22 a=-1 r=1 w=1

/***WCET calculation source***/
/* WCA WCET objective: wcet.WCATestIREM.f_irem */
max: tM67_S_E0 tM67_B1_E0 tM67_B2_E0 tM67_B3_E0 tM67_B4_E0 tM67_B5_E0 tM67_B6_E0 tM67_B7_E0
tM67_B8_E0 tM67_B9_E0 tM67_B10_E0 tM67_B11_E0 tM67_B12_E0 tM67_B13_E0 tM67_B14_E0 tM67_B15_E0
tM67_B16_E0 tM67_T_E0;
/* WCA flow constraints: f_irem : M67 */
M67_S_E0: 1 = fM67_S_E0_M67_B1_E0; // S flow
M67_B1_E0: fM67_S_E0_M67_B1_E0 + fM67_B1_E0_M67_B1_E0 = fM67_B1_E0_M67_B2_E0 +
fM67_B1_E0_M67_B3_E0;
M67_B2_E0: fM67_B2_E0_M67_B2_E0 + fM67_B1_E0_M67_B2_E0 = fM67_B2_E0_M67_T_E0;
M67_B3_E0: fM67_B1_E0_M67_B3_E0 = fM67_B3_E0_M67_B4_E0 + fM67_B3_E0_M67_B5_E0;
M67_B4_E0: fM67_B4_E0_M67_B4_E0 + fM67_B3_E0_M67_B4_E0 = fM67_B4_E0_M67_B5_E0;
M67_B5_E0: fM67_B4_E0_M67_B5_E0 + fM67_B3_E0_M67_B5_E0 = fM67_B5_E0_M67_B6_E0 +
fM67_B5_E0_M67_B7_E0;
M67_B6_E0: fM67_B5_E0_M67_B6_E0 + fM67_B6_E0_M67_B6_E0 = fM67_B6_E0_M67_B7_E0;
M67_B7_E0: fM67_B5_E0_M67_B7_E0 + fM67_B6_E0_M67_B7_E0 = fM67_B7_E0_M67_B8_E0;
M67_B8_E0: fM67_B7_E0_M67_B8_E0 + fM67_B13_E0_M67_B8_E0 = fM67_B8_E0_M67_B9_E0 +
fM67_B8_E0_M67_B14_E0;
LC_M67_B8_E0: fM67_B8_E0_M67_B9_E0 = 32 fM67_B5_E0_M67_B7_E0;
M67_B9_E0: fM67_B8_E0_M67_B9_E0 + fM67_B9_E0_M67_B9_E0 = fM67_B9_E0_M67_B10_E0 +
fM67_B9_E0_M67_B11_E0;
M67_B10_E0: fM67_B10_E0_M67_B10_E0 + fM67_B9_E0_M67_B10_E0 = fM67_B10_E0_M67_B11_E0;
M67_B11_E0: fM67_B10_E0_M67_B11_E0 + fM67_B9_E0_M67_B11_E0 = fM67_B11_E0_M67_B12_E0 +
fM67_B11_E0_M67_B13_E0;
M67_B12_E0: fM67_B11_E0_M67_B12_E0 + fM67_B12_E0_M67_B12_E0 = fM67_B12_E0_M67_B13_E0;
M67_B13_E0: fM67_B11_E0_M67_B13_E0 + fM67_B12_E0_M67_B13_E0 = fM67_B13_E0_M67_B8_E0;
M67_B14_E0: fM67_B8_E0_M67_B14_E0 = fM67_B14_E0_M67_B15_E0 + fM67_B14_E0_M67_B16_E0;
M67_B15_E0: fM67_B15_E0_M67_B15_E0 + fM67_B14_E0_M67_B15_E0 = fM67_B15_E0_M67_B16_E0;
M67_B16_E0: fM67_B15_E0_M67_B16_E0 + fM67_B14_E0_M67_B16_E0 = fM67_B16_E0_M67_T_E0;
M67_T_E0: fM67_B16_E0_M67_T_E0 + fM67_B2_E0_M67_T_E0 = 1; // T flow
/* WCA flow to cycle count */
```

```
tM67_B1_E0 = 5 fM67_S_E0_M67_B1_E0 + 5 fM67_B1_E0_M67_B1_E0;
tM67_B2_E0 = 33 fM67_B2_E0_M67_B2_E0 + 33 fM67_B1_E0_M67_B2_E0;
tM67_B3_E0 = 7 fM67_B1_E0_M67_B3_E0;
tM67_B4_E0 = 8 fM67_B4_E0_M67_B4_E0 + 8 fM67_B3_E0_M67_B4_E0;
tM67_B5_E0 = 5 fM67_B4_E0_M67_B5_E0 + 5 fM67_B3_E0_M67_B5_E0;
tM67_B6_E0 = 6 fM67_B5_E0_M67_B6_E0 + 6 fM67_B6_E0_M67_B6_E0;
tM67_B7_E0 = 5 fM67_B5_E0_M67_B7_E0 + 5 fM67_B6_E0_M67_B7_E0;
tM67_B8_E0 = 8 fM67_B7_E0_M67_B8_E0 + 8 fM67_B13_E0_M67_B8_E0;
tM67_B9_E0 = 18 fM67_B8_E0_M67_B9_E0 + 18 fM67_B9_E0_M67_B9_E0;
tM67_B10_E0 = 4 fM67_B10_E0_M67_B10_E0 + 4 fM67_B9_E0_M67_B10_E0;
tM67_B11_E0 = 10 fM67_B10_E0_M67_B11_E0 + 10 fM67_B9_E0_M67_B11_E0;
tM67_B12_E0 = 4 fM67_B11_E0_M67_B12_E0 + 4 fM67_B12_E0_M67_B12_E0;
tM67_B13_E0 = 12 fM67_B11_E0_M67_B13_E0 + 12 fM67_B12_E0_M67_B13_E0;
tM67_B14_E0 = 5 fM67_B8_E0_M67_B14_E0;
tM67_B15_E0 = 6 fM67_B15_E0_M67_B15_E0 + 6 fM67_B14_E0_M67_B15_E0;
tM67_B16_E0 = 24 fM67_B15_E0_M67_B16_E0 + 24 fM67_B14_E0_M67_B16_E0;
/* Invocation(s) from f_irem:[M67] */
```

Value of objective function: 1865

Actual values of the variables:

tM67_S_E0	0
tM67_B1_E0	5
tM67_B2_E0	0
tM67_B3_E0	7
tM67_B4_E0	8
tM67_B5_E0	5
tM67_B6_E0	0
tM67_B7_E0	5
tM67_B8_E0	264
tM67_B9_E0	576
tM67_B10_E0	128
tM67_B11_E0	320
tM67_B12_E0	128
tM67_B13_E0	384
tM67_B14_E0	5
tM67_B15_E0	6
tM67_B16_E0	24
tM67_T_E0	0
fM67_S_E0_M67_B1_E0	1
fM67_B1_E0_M67_B1_E0	0
fM67_B1_E0_M67_B2_E0	0
fM67_B1_E0_M67_B3_E0	1
fM67_B2_E0_M67_B2_E0	0
fM67_B2_E0_M67_T_E0	0
fM67_B3_E0_M67_B4_E0	1
fM67_B3_E0_M67_B5_E0	0
fM67_B4_E0_M67_B4_E0	0
fM67_B4_E0_M67_B5_E0	1
fM67_B5_E0_M67_B6_E0	0
fM67_B5_E0_M67_B7_E0	1
fM67_B6_E0_M67_B6_E0	0
fM67_B6_E0_M67_B7_E0	0
fM67_B7_E0_M67_B8_E0	1
fM67_B13_E0_M67_B8_E0	32
fM67_B8_E0_M67_B9_E0	32
fM67_B8_E0_M67_B14_E0	1
fM67_B9_E0_M67_B9_E0	0

```
fM67_B9_E0_M67_B10_E0          32
fM67_B9_E0_M67_B11_E0           0
fM67_B10_E0_M67_B10_E0          0
fM67_B10_E0_M67_B11_E0         32
fM67_B11_E0_M67_B12_E0         32
fM67_B11_E0_M67_B13_E0          0
fM67_B12_E0_M67_B12_E0          0
fM67_B12_E0_M67_B13_E0         32
fM67_B14_E0_M67_B15_E0          1
fM67_B14_E0_M67_B16_E0          0
fM67_B15_E0_M67_B15_E0          0
fM67_B15_E0_M67_B16_E0          1
fM67_B16_E0_M67_T_E0           1

/*wcet.WCATestIREM.f_irem(II)I*/
digraph G {
size = "10,7.5"
        M67_S_E0->M67_B1_E0 [label="fM67_S_E0_M67_B1_E0=1"];
        M67_B1_E0->M67_B2_E0 [style=dashed,label="fM67_B1_E0_M67_B2_E0=0"];
        M67_B1_E0->M67_B3_E0 [label="fM67_B1_E0_M67_B3_E0=1"];
        M67_B2_E0->M67_T_E0 [style=dashed,label="fM67_B2_E0_M67_T_E0=0"];
        M67_B3_E0->M67_B4_E0 [label="fM67_B3_E0_M67_B4_E0=1"];
        M67_B3_E0->M67_B5_E0 [style=dashed,label="fM67_B3_E0_M67_B5_E0=0"];
        M67_B4_E0->M67_B5_E0 [label="fM67_B4_E0_M67_B5_E0=1"];
        M67_B5_E0->M67_B6_E0 [style=dashed,label="fM67_B5_E0_M67_B6_E0=0"];
        M67_B5_E0->M67_B7_E0 [label="fM67_B5_E0_M67_B7_E0=1"];
        M67_B6_E0->M67_B7_E0 [style=dashed,label="fM67_B6_E0_M67_B7_E0=0"];
        M67_B7_E0->M67_B8_E0 [label="fM67_B7_E0_M67_B8_E0=1"];
        M67_B8_E0->M67_B9_E0 [label="fM67_B8_E0_M67_B9_E0=32"];
        M67_B8_E0->M67_B14_E0 [label="fM67_B8_E0_M67_B14_E0=1"];
        M67_B9_E0->M67_B10_E0 [label="fM67_B9_E0_M67_B10_E0=32"];
        M67_B9_E0->M67_B11_E0 [style=dashed,label="fM67_B9_E0_M67_B11_E0=0"];
        M67_B10_E0->M67_B11_E0 [label="fM67_B10_E0_M67_B11_E0=32"];
        M67_B11_E0->M67_B12_E0 [label="fM67_B11_E0_M67_B12_E0=32"];
        M67_B11_E0->M67_B13_E0 [style=dashed,label="fM67_B11_E0_M67_B13_E0=0"];
        M67_B12_E0->M67_B13_E0 [label="fM67_B12_E0_M67_B13_E0=32"];
        M67_B13_E0->M67_B8_E0 [label="fM67_B13_E0_M67_B8_E0=32"];
        M67_B14_E0->M67_B15_E0 [label="fM67_B14_E0_M67_B15_E0=1"];
        M67_B14_E0->M67_B16_E0 [style=dashed,label="fM67_B14_E0_M67_B16_E0=0"];
        M67_B15_E0->M67_B16_E0 [label="fM67_B15_E0_M67_B16_E0=1"];
        M67_B16_E0->M67_T_E0 [label="fM67_B16_E0_M67_T_E0=1"];
        M67_S_E0;
        M67_B1_E0 [label="M67_B1_E0\n5"];
        M67_B2_E0 [label="M67_B2_E0\n102"];
        M67_B3_E0 [label="M67_B3_E0\n7"];
        M67_B4_E0 [label="M67_B4_E0\n8"];
        M67_B5_E0 [label="M67_B5_E0\n5"];
        M67_B6_E0 [label="M67_B6_E0\n6"];
        M67_B7_E0 [label="M67_B7_E0\n5"];
        M67_B8_E0 [label="M67_B8_E0\n8"];
        M67_B9_E0 [label="M67_B9_E0\n18"];
        M67_B10_E0 [label="M67_B10_E0\n4"];
        M67_B11_E0 [label="M67_B11_E0\n10"];
        M67_B12_E0 [label="M67_B12_E0\n4"];
        M67_B13_E0 [label="M67_B13_E0\n12"];
        M67_B14_E0 [label="M67_B14_E0\n5"];
        M67_B15_E0 [label="M67_B15_E0\n6"];
        M67_B16_E0 [label="M67_B16_E0\n24"];
```

```
          M67_T_E0;
}
********************************************************************************
WCET info for:wcet.WCATestIREM.main([Ljava/lang/String;)V

Directed graph of basic blocks(row->column):
=============================================
   M66_S_E0M66_B1_E0M66_I2_E0M66_B3_E0M66_T_E0
M66_S_E0 .  1  .  .  .
M66_B1_E0 .  .  1  .  .
M66_I2_E0 .  .  .  1  .
M66_B3_E0 .  .  .  .  1
M66_T_E0 .  .  .  .  .
=============================================

Table of basic blocks' and instructions
================================================================================
Block Addr.  Bytecode              Cycles    Cache miss     Misc. info
          [opcode]                 invoke  return
--------------------------------------------------------------------------------
M66_S_E0'S'
  Src. line 15: f_irem(-2, -2);
M66_B1_E0'B'{}<-[M66_S_E0 M66_B1_E0]0:     bipush[16]                2
     2:     bipush[16]               2                 sum(B1):     4
  Src. line 15: f_irem(-2, -2);
M66_I2_E0'I'{}<-[M66_I2_E0 M66_B1_E0]4:     invokestatic[184]          75      15      4
wcet.WCATestIREM.f_irem(II)I, invoke(n=22):75/90 return(n=3):23/27sum(B2):     75
  Src. line 15: f_irem(-2, -2);
M66_B3_E0'B'{}<-[M66_I2_E0 M66_B3_E0]7:     pop[87]                   1
  Src. line 16: }
     8:     return[177]              21                sum(B3):     22
M66_T_E0'T'<-[M66_B3_E0]
================================================================================
Info: n=3 a=-1 r=1 w=1

/***WCET calculation source***/
/* WCA WCET objective: wcet.WCATestIREM.main */
max: tM66_S_E0 tM66_B1_E0 tM66_I2_E0 tM66_B3_E0 tM66_T_E0;
/* WCA flow constraints: main : M66 */
M66_S_E0: 1 = fM66_S_E0_M66_B1_E0; // S flow
M66_B1_E0: fM66_S_E0_M66_B1_E0 + fM66_B1_E0_M66_B1_E0 = fM66_B1_E0_M66_I2_E0;
M66_I2_E0: fM66_I2_E0_M66_I2_E0 + fM66_B1_E0_M66_I2_E0 = fM66_I2_E0_M66_B3_E0;
M66_B3_E0: fM66_I2_E0_M66_B3_E0 + fM66_B3_E0_M66_B3_E0 = fM66_B3_E0_M66_T_E0;
M66_T_E0: fM66_B3_E0_M66_T_E0 = 1; // T flow
/* WCA flow to cycle count */
tM66_B1_E0 = 4 fM66_S_E0_M66_B1_E0 + 4 fM66_B1_E0_M66_B1_E0;
tM66_I2_E0 = 75 fM66_I2_E0_M66_I2_E0 + 75 fM66_B1_E0_M66_I2_E0;
tM66_B3_E0 = 22 fM66_I2_E0_M66_B3_E0 + 22 fM66_B3_E0_M66_B3_E0;
/* Invocation(s) from main:[M66] */

Value of objective function: 101

Actual values of the variables:
tM66_S_E0                  0
tM66_B1_E0                 4
tM66_I2_E0                75
tM66_B3_E0                22
tM66_T_E0                  0
```

```
fM66_S_E0_M66_B1_E0            1
fM66_B1_E0_M66_B1_E0           0
fM66_B1_E0_M66_I2_E0           1
fM66_I2_E0_M66_I2_E0           0
fM66_I2_E0_M66_B3_E0           1
fM66_B3_E0_M66_B3_E0           0
fM66_B3_E0_M66_T_E0            1

/*wcet.WCATestIREM.main([Ljava/lang/String;)V*/
digraph G {
size = "10,7.5"
        M66_S_E0->M66_B1_E0 [label="fM66_S_E0_M66_B1_E0=1"];
        M66_B1_E0->M66_I2_E0 [label="fM66_B1_E0_M66_I2_E0=1"];
        M66_I2_E0->M66_B3_E0 [label="fM66_I2_E0_M66_B3_E0=1"];
        M66_B3_E0->M66_T_E0 [label="fM66_B3_E0_M66_T_E0=1"];
        M66_S_E0;
        M66_B1_E0 [label="M66_B1_E0\n4"];
        M66_I2_E0 [label="M66_I2_E0\n75"];
        M66_B3_E0 [label="M66_B3_E0\n22"];
        M66_T_E0;
}

**************************************************
Note: Remember to keep WCETAnalyzer updated
each time a bytecode implementation is changed.
```

B.2.4 Window

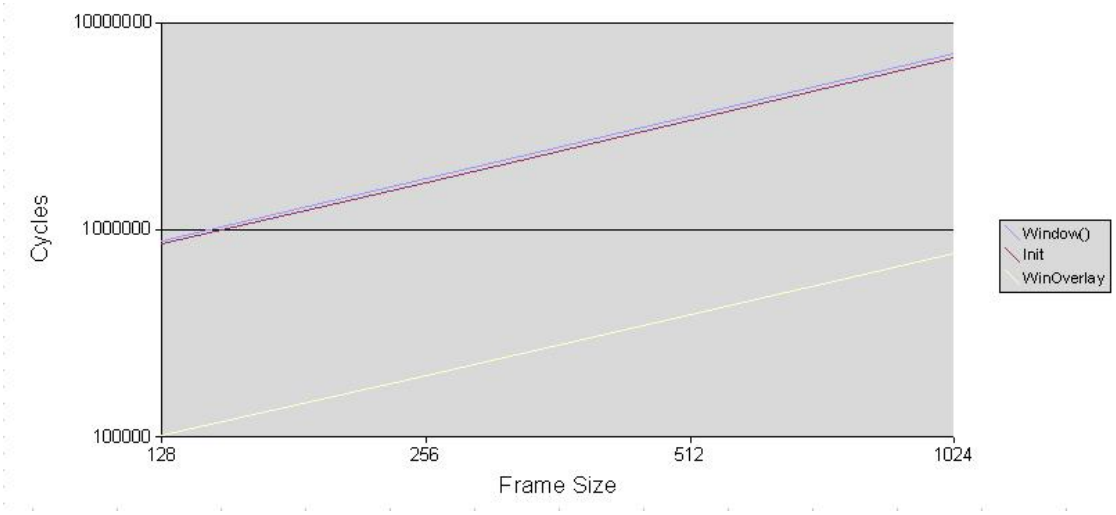


Figure B.1: Clock Cycles for Window Method Depended on Frame Size on Log scale

B.3 The classes in Java

This section contains all the different classes implemented in Java.

B.3.1 JopSpeech

```
1  package jopspeech;
2
3  import jopspeech.frontend.FrontEnd;
4  import jopspeech.frontend.FrontEndWorker;
5  import jopspeech.analysis.Analysis;
6  import jopspeech.analysis.AnalysisWorker;
7  import jopspeech.recognizer.Recognizer;
8  import jopspeech.recognizer.RecognizerWorker;
9  import jopspeech.utility.FixPoint;
10 import jopspeech.utility.Protocol;
11
12 import joprt.RtThread;
13
14 import util.Dbg;
15 import util.Timer;
16
17 import ejip.CS8900;
18 import ejip.LinkLayer;
19 import ejip.Net;
20 import ejip.Packet;
21 import ejip.Udp;
22 import ejip.UdpHandler;
23
24 /**
25  *
26  * Embedded Java Speech Recognition on JOP <br />
```

```
27  * Website: <a href="http://www.jopspeech.com">www.jopspeech.com</a>
28  *
29  * @author Mikael Lundsgaard and Jens Kristian Rasmussen
30  * @version 0.1.0
31  *
32  * This is the client that runs on JOP. The class is starting the Threads that
33  * is needed for doing speech recognition. All variables is set from this class
34  * notice that the IP address needs to match the PC's IP address
35  */
36 public class JopSpeech {
37     static Net net;
38 
39     public static LinkLayer ipLink;
40 
41     public static int IP = (130 << 24) + (226 << 16) + (39 << 8) + 54;
42 
43     public static int PORT = 2346;
44 
45     public static FrontEnd frontEnd;
46 
47     public static Analysis analysis;
48 
49     public static Recognizer recognizer;
50 
51     // Defines the work done in this experiment and handles the
52     // data flow
53 
54     /**
55      * Start network and enter forever loop.
56      */
57     public static void main(String[] args) {
58 
59         Dbg.initSerWait();
60 
61         //
62         // start initialization
63         //
64 
65         net = Net.init();
66 
67         ipLink = CS8900.init(Net.eth, Net.ip);
68 
69         FixPoint.init();
70 
71         frontEnd = FrontEndWorker.init();
72         analysis = AnalysisWorker.init();
73         recognizer = RecognizerWorker.init();
74 
75         //
76         // start device driver threads
77         //
78 
79         new RtThread(4, 500) {
80             public void run() {
81                 for (;;) {
82                     waitForNextPeriod();
83                     // System.out.print("Receive ");
84                     net.loop();
85                 }
86             }
87         };
88         new RtThread(4, 500) {
89             public void run() {
90                 for (;;) {
91                     waitForNextPeriod();
92                     // System.out.println("Sent ");
93                     ipLink.loop();
94                 }
```

```
95              }
96          };
97
98          new RtThread(5, 5000) {
99              public void run() {
100                 for (;;) {
101                     waitForNextPeriod();
102                     frontEnd.mission();
103                 }
104             }
105         };
106
107         new RtThread(6, 8000) {
108             public void run() {
109                 for (;;) {
110                     waitForNextPeriod();
111
112                     analysis.mission();
113
114                 }
115             }
116         };
117
118         new RtThread(7, 8000) {
119             public void run() {
120                 for (;;) {
121                     waitForNextPeriod();
122                     recognizer.mission();
123                 }
124             }
125         };
126
127         UdpHandler adder;
128         adder = new UdpHandler() {
129             public void request(Packet p) {
130                 if (p.buf[Udp.DATA] == Protocol.INIT
131                         || p.buf[Udp.DATA] == Protocol.LOAD
132                         || p.buf[Udp.DATA] == Protocol.ANALYSE
133                         || p.buf[Udp.DATA] == Protocol.TEST
134                         || p.buf[Udp.DATA] == Protocol.TRAIN
135                         || p.buf[Udp.DATA] == Protocol.PING
136                         || p.buf[Udp.DATA] == Protocol.RECORD
137                         || p.buf[Udp.DATA] == Protocol.MAININIT) {
138                     while (frontEnd.getDone() == 0)
139                         RtThread.sleepMs(200);
140                     FrontEndWorker.p = p;
141                     frontEnd.setReady();
142
143                 }
144             }
145
146         };
147
148         Udp.addHandler(2345, adder);
149
150         // Start all the threads defined above and let them do their work
151         RtThread.startMission();
152
153         // Check if still running
154         for (;;) {
155             RtThread.sleepMs(5000);
156             Timer.wd();
157         }
158     }
159 }
```

Listing B.1: JopSpeech.java

B.3.2 FrontEnd

```
1  package jopspeech.frontend;
2
3  /**
4   * Embedded Java Speech Recognition on JOP <br />
5   * Website: <a href="http://www.jopspeech.com">www.jopspeech.com</a>
6   *
7   * @author Mikael Lundsgaard and Jens Kristian Rasmussen
8   * @version 0.1.0
9   *
10  * <p>
11  * The FrontEnd component contains the following classes <img src="../../resources/FrontEnd.png"/>
12  */
13 public abstract class FrontEnd {
14
15     private int ready = 0;
16
17     private int done = 1;
18
19     public static FrontEnd init() {
20         return null;
21     }
22
23     public void mission() {
24         if (ready == 1) {
25             ready = 0;
26             work();
27             done = 1;
28         }
29     }
30
31     protected abstract void work();
32
33     public synchronized int setReady() {
34         if (ready == 1) {
35             return ready;
36         } else {
37             ready = 1;
38             done = 0;
39             return ready;
40         }
41     }
42
43     public synchronized int getDone() {
44         return done;
45     }
46
47 }
```

Listing B.2: FrontEnd.java

B.3.3 FrontEndWorker

```
1  package jopspeech.frontend;
2
3  import ejip.Packet;
4  import ejip.Udp;
5  import joprt.RtThread;
6  import jopspeech.JopSpeech;
7  import jopspeech.analysis.AnalysisWorker;
8  import jopspeech.analysis.MFCC;
9  import jopspeech.analysis.Window;
10 import jopspeech.recognizer.KNear;
11 import jopspeech.utility.FixPoint;
12 import jopspeech.utility.Protocol;
```

```
13
14 /**
15  * Embedded Java Speech Recognition on JOP <br />
16  * Website: <a href="http://www.jopspeech.com">www.jopspeech.com</a>
17  *
18  * @author Mikael Lundsgaard and Jens Kristian Rasmussen
19  * @version 0.1.0
20  *
21  */
22 public class FrontEndWorker extends FrontEnd {
23
24     public static Signal utterance;
25
26     public final static int SAMPELSIZE = 8192;
27
28     private static FrontEndWorker single;
29
30     private static Audio mic;
31
32     public static Packet p;
33
34     static StringBuffer s;
35
36     private static int numSamplesRecive = 0;
37
38     public static FrontEnd init() {
39         if (single != null)
40             return single;
41         single = new FrontEndWorker();
42         s = new StringBuffer(20);
43         mic = AudioImpl.init();
44         utterance = Utterance.init(SAMPELSIZE, AnalysisWorker.FRAMESIZE);
45         return single;
46     }
47
48     public void work() {
49         switch (p.buf[Udp.DATA]) {
50         case Protocol.INIT:
51             if (p.buf[Udp.DATA + 1] == Protocol.TESTWORD) {
52                 utterance.classID = "UNKNOWN";
53                 utterance.length = p.buf[Udp.DATA + 2];
54                 utterance.signalID = p.buf[Udp.DATA + 1];
55                 AnalysisWorker.template.numberFrames = ((100 / Window.OVERLAYPRC)
56                         * utterance.length / AnalysisWorker.FRAMESIZE) - 1;
57
58             } else {
59                 System.out.print("Word: ");
60                 System.out.println(p.buf[Udp.DATA + 1]);
61                 // The length of the word
62                 utterance.signalID = p.buf[Udp.DATA + 1];
63
64                 s.setLength(0);
65                 // System.out.println(p.len);
66                 for (int i = (Udp.DATA + 3) * 4; i < p.len; i++) {
67                     s
68                             .append((char) ((p.buf[i >> 2] >> (24 - ((i & 3) << 3))) & 0xff));
69                 }
70                 utterance.classID = s.toString();
71
72                 System.out.println(utterance.classID);
73                 utterance.length = p.buf[Udp.DATA + 2];
74             }
75             complete(p);
76             break;
77         case Protocol.LOAD:
78
79             int i;
80             int j;
```

```
81
82              numSamplesRecive = p.buf[Udp.DATA + 2];
83              // lastSampleRecive = p.buf[Udp.DATA + 3];
84
85              j = 4;
86              if (p.buf[Udp.DATA + 1] == Protocol.MFCC) {
87                  for (i = 0; i < numSamplesRecive; i++) {
88                      MFCC.initWeights[p.buf[Udp.DATA + j]] = p.buf[Udp.DATA + j
89                              + 1];
90                      j += 2;
91                  }
92              } else if (p.buf[Udp.DATA + 1] == Protocol.MFCC + 1) {
93                  for (i = 0; i < numSamplesRecive; i++) {
94                      MFCC.melCosbank[p.buf[Udp.DATA + j]] = p.buf[Udp.DATA + j
95                              + 1];
96                      j += 2;
97                  }
98
99              } else {
100                 for (i = 0; i < numSamplesRecive; i++) {
101                     utterance.raw_sample[p.buf[Udp.DATA + j]] = p.buf[Udp.DATA
102                             + j + 1] << FixPoint.SHIFT;
103                     j = j + 2;
104                 }
105             }
106             complete(p);
107             break;
108         // Analyse the signal recived and add to model
109         case Protocol.ANALYSE:
110             if (p.buf[Udp.DATA + 1] != Protocol.TESTWORD) {
111                 JopSpeech.analysis.setReady();
112                 while (JopSpeech.analysis.getDone() == 0)
113                     RtThread.sleepMs(500);
114                 complete(p);
115             } else {
116                 JopSpeech.analysis.setReady();
117                 while (JopSpeech.analysis.getDone() == 0)
118                     RtThread.sleepMs(500);
119                 while (JopSpeech.recognizer.getDone() == 0)
120                     RtThread.sleepMs(500);
121                 resultPacket(p, KNear.timeStart, KNear.timeEnd, KNear.wordID,
122                         KNear.result);
123             }
124             break;
125
126         case Protocol.RECORD:
127             // use the mic as input for recognition
128             System.out.println("Record word");
129             utterance.classID = "UNKNOWN";
130             utterance.signalID = Protocol.TESTWORD;
131
132             mic.record(utterance);
133             RtThread.sleepMs(50);
134
135             System.out.println("Word recorded");
136             JopSpeech.analysis.setReady();
137             while (JopSpeech.analysis.getDone() == 0)
138                 RtThread.sleepMs(500);
139             System.out.println("Recognizing...");
140
141             while (JopSpeech.recognizer.getDone() == 0)
142                 RtThread.sleepMs(500);
143
144             System.out.print("You said:");
145             System.out.println(KNear.classID);
146             System.out.print("Time to recognize: ");
147             System.out.println(KNear.timeEnd - KNear.timeStart);
148             System.out.print("Result: ");
```

```
149             System.out.println(KNear.result);
150
151             resultPacket(p, KNear.timeStart, KNear.timeEnd, KNear.wordID,
152                     KNear.result);
153             break;
154         case Protocol.PING:
155
156             complete(p);
157             break;
158         }
159     }
160
161     /**
162      * Send a result to the PC
163      *
164      * @param resultPacket
165      *            the packet to be sent
166      * @param timeStart
167      *            The start time of the Test
168      * @param timeEnd
169      *            The end time of the Test
170      * @param wordID
171      *            The result WordID
172      * @param result
173      *            the result from the DTW
174      */
175     public static void resultPacket(Packet resultPacket, int timeStart,
176             int timeEnd, int wordID, int result) {
177         // int i;
178         resultPacket = getPacket(resultPacket);
179         resultPacket.buf[Udp.DATA] = Protocol.RESULT;
180         resultPacket.buf[Udp.DATA + 1] = timeStart;
181         resultPacket.buf[Udp.DATA + 2] = timeEnd;
182         resultPacket.buf[Udp.DATA + 3] = wordID;
183         resultPacket.buf[Udp.DATA + 4] = result;
184         // for (i = 0; i < Model.NUMTEMPLATES; i++)
185         // resultPacket.buf[Udp.DATA + i + 5] = DTW.compare[i];
186
187         resultPacket.len = (Udp.DATA + 4) << 2;
188         // System.out.println(" About to send packet");
189         Udp.build(resultPacket, JopSpeech.IP, JopSpeech.PORT);
190         // System.out.println("Result packet send");
191
192     }
193
194     /**
195      * Sent a complete packet to the PC to notifice that the request where done
196      *
197      * @param completePacket
198      *            the packet to be sent
199      */
200     public static void complete(Packet completePacket) {
201         completePacket = getPacket(completePacket);
202         completePacket.buf[Udp.DATA] = Protocol.COMPLETE;
203         completePacket.len = (Udp.DATA + 1) << 2;
204         // System.out.println(" About to send packet");
205         Udp.build(completePacket, JopSpeech.IP, JopSpeech.PORT);
206         // System.out.println("Packet sent");
207     }
208
209     /**
210      * get a packet from the packet pool
211      *
212      * @param p
213      *            a packet to get a packet
214      * @return a packet
215      */
216     public static Packet getPacket(Packet p) {
```

```
217         p.setStatus(Packet.FREE);
218         p = null;
219         for (; p == null;) {
220             p = Packet.getPacket(Packet.FREE, Packet.ALLOC);
221             // System.out.println("got packet");
222             RtThread.sleepMs(10);
223         }
224         p.interf = JopSpeech.ipLink;
225         return p;
226     }
227 }
```

Listing B.3: FrontEndWorker.java

B.3.4 Audio

```
1  package jopspeech.frontend;
2
3  /**
4   * Embedded Java Speech Recognition on JOP <br />
5   * Website: <a href="http://www.jopspeech.com">www.jopspeech.com</a>
6   *
7   * @author Mikael Lundsgaard and Jens Kristian Rasmussen
8   * @version 0.1.0
9   *
10  * The Audio interface defines how to get a signal in and out of the system,
11  * this could be a change of microphones, or sampling method that is necessary.
12  * The Audio Interface defines the methods required to Record and Play back the
13  * signal.
14  *
15  */
16 public interface Audio {
17     /**
18      * The record method defines how the system should receive a signal from
19      * outside. To ensure that all signals handled in a certain way the format
20      * returned should always be of the Signal format
21      *
22      * @param signalIn
23      * @return length of recorded signal
24      */
25     public int record(Signal signalIn);
26
27     /**
28      * The playback method defines how the system should playback a Signal to
29      * the outside.
30      *
31      * @param signalOut
32      *            Signal to be played
33      * @param length
34      *            Lenght of Signal
35      */
36     public void play(Signal signalOut, int length);
37 }
```

Listing B.4: Audio.java

B.3.5 AudioImpl

```
1  package jopspeech.frontend;
2
3  import jopspeech.utility.FixPoint;
4
5  import com.jopdesign.sys.Const;
6  import com.jopdesign.sys.Native;
7
```

```
8  /**
9   * Embedded Java Speech Recognition on JOP <br />
10  * Website: <a href="http://www.jopspeech.com">www.jopspeech.com</a>
11  *
12  *
13  * @author Martin Schoerberl, Mikael Lundsgaard and Jens Kristian Rasmussen
14  * @version 0.1.0
15  *
16  *
17  * This AudioImpl Class is a implementation of the audio interface specifying
18  * how the recording and play back of sound can be done on JOP. The class
19  * "Audio" is closely connected to the hardware of JOP. It uses native calls to
20  * record and play sound and is implmented with the Hardware constructed by
21  * Mikael Lundsgaard and Jens Kristian Rasmussen specified on www.jopspeech.com
22  * and cannot be promised to work with any other
23  *
24  *
25  */
26 public class AudioImpl implements Audio {
27     /**
28      * The size and frame of the implementation
29      */
30     private static int[] frame = new int[80];
31
32     private static int cntF, val, cntNoF;
33
34     private static AudioImpl single;
35
36     /**
37      * The StartFrame means that the method is able to detect when it is
38      * receiving sound from a speaker. STARTFRAME frames needs to be recorded in
39      * a row for the method to accept the siganl as anything other than noise
40      */
41     private static final int STARTFRAME = 4;
42
43     /**
44      * The threshold value for a frame if there signals amplitude summed up is
45      * not above the start detection value the frame will be considered empty.
46      * The threshold value is dependent on the environment and context it is
47      * used, if no ALC is added to the microphone
48      */
49     private static final int STARTDETECTION = 600;
50
51     /**
52      * The end dectection value for a signal if there is more than ENDDETECTION
53      * empty frames being recorded the signal will be considered recorded
54      */
55     private static final int ENDDETECTION = 12;
56
57     /**
58      * Init method of the Class
59      *
60      * @return Audio object
61      */
62     public static Audio init() {
63
64         if (single != null)
65             return single;
66         single = new AudioImpl();
67         return single;
68     }
69
70     /**
71      * This is the method specifing how a recording is done It works by
72      * recording maximum the lenght of the Signal that is submitted to the
73      * method by adding one frame at a time to the Signal. The frame is always
74      * checked for sound every time it is recorded, and is only added to the
75      * signal if minimum 4 full frames have already been added else it will be
```

```
76      * considered as noise and omitted
77      *
78      * @param signalIn
79      *            The Signal being recorded in a specified fixpoint Shift.
80      * @see FixPoint
81      */
82     public int record(Signal signalIn) {
83
84         // Counters used in the method
85         cntF = 0;
86         cntNoF = 0;
87         int cnt = 80;
88         int sum = 0;
89         int lengthOffset = 200;
90
91         // Inital centerValue for the sample
92         int offSet = calcOffset(lengthOffset);
93
94         val = 0;
95         for (;;) {
96             // read signal
97             val = Native.rdMem(Const.IO_MICRO);
98             if (val > 0) {
99                 continue;
100            }
101            val &= 0xffff;
102
103            val -= offSet;
104
105            frame[80 - cnt] = val;
106            if (val < 0)
107                val = -val;
108
109            sum += val;
110            --cnt;
111
112            if (cnt == 0) {
113                cnt = 80;
114                // When the frame is full check if there is sound in it, else if
115                // not then evaluate if it has been noise than restart or its
116                // okay or stop recording
117                if (sum > STARTDETECTION) {
118                    writeFrame(signalIn.raw_sample, frame, cntF);
119                } else {
120                    if (cntNoF > ENDDETECTION) {
121                        signalIn.length = cntF * 80;
122                        for (int i = 0; i < signalIn.length; i++)
123                            signalIn.raw_sample[i] <<= FixPoint.SHIFT;
124                        // Return the lenght of the signal, Recording finished
125                        System.out.print("Recorded signal");
126                        return cntF * 80;
127                    }
128                    // if recording of the signal has started and minimum
129                    // 4 frames have been written
130                    else if (cntF > STARTFRAME) {
131                        writeFrame(signalIn.raw_sample, frame, cntF);
132                        cntNoF++;
133                    }
134                    // Assume it has
135                    // 4 frames have been written
136                    else {
137                        cntF = 0;
138                    }
139                }
140                sum = 0;
141            }
142
143            // Test if we missed the sample output
```

```
144             val = Native.rdMem(Const.IO_MICRO);
145             if (val < 0) {
146                 System.out.print('*');
147                 continue;
148             }
149         }
150     }
151
152     /**
153      * Playback the signal
154      *
155      * @param signalOut
156      *            The signal to played ack to a loudspeaker connected
157      * @param length
158      *            The lenght of the Signal
159      */
160     public void play(Signal signalOut, int length) {
161         int val;
162         int offSet = calcOffset(200);
163         for (int j = 0; j < length; j++) {
164             //
165             val = Native.rdMem(Const.IO_MICRO);
166             for (; val > 0;) {
167                 val = Native.rdMem(Const.IO_MICRO);
168             }
169
170             val = signalOut.raw_sample[j] + offSet;
171             Native.wrMem(val, Const.IO_MICRO);
172         }
173     }
174
175     /**
176      * *Used to calculate the Offset
177      *
178      * @param lengthOffset
179      *            The number of samples used to calulate the intial center value
180      *            Offset
181      * @return Offset
182      */
183     private int calcOffset(int lengthOffset) {
184         int offSet = 0;
185         if (lengthOffset <= 0)
186             return 1450;
187         for (int i = 0; i < lengthOffset; i++) {
188             val = Native.rdMem(Const.IO_MICRO);
189             if (val > 0) {
190                 lengthOffset--;
191                 continue;
192             }
193             val &= 0xffff;
194             offSet += val;
195         }
196         offSet = offSet / lengthOffset;
197         if (offSet > 1500 || offSet < 1400)
198             return 1450;
199         return offSet;
200     }
201
202     /**
203      * Private method to determine if the sample is out of bounce because of
204      * noise or electrical interference
205      *
206      * @param sample
207      *            The sample being recorded
208      * @param frame
209      *            The frame with values to be recorded
210      * @param cntFin
211      *            The number of frames already written to the sample
```

```
212      */
213     private void writeFrame(int[] sample, int[] frame, int cntFin) {
214         int j = cntFin * 80;
215 
216         // If there is room in the sample write the frame.
217         if (j < sample.length - 80) {
218             for (int i = 0; i < 80; i++) {
219                 sample[i + j] = frame[i];
220             }
221         } else {
222             System.out.println("Sample out of bounce");
223             cntF = 0;
224             cntNoF = 0;
225         }
226         cntF++;
227     }
228 
229 }
```

Listing B.5: AudioImpl.java

B.3.6 Filter

```
1  package jopspeech.frontend;
2  
3  /**
4   * Embedded Java Speech Recognition on JOP <br />
5   * Website: <a href="http://www.jopspeech.com">www.jopspeech.com</a>
6   *
7   * @author Mikael Lundsgaard and Jens Kristian Rasmussen
8   * @version 0.1.0
9   */
10 public class Filter {
11 
12     public static int[] firFilter(int sample[], int length) {
13         return sample;
14     }
15 
16     public static int[] iirFilter(int sample[], int length) {
17         int i, y[];
18         // System.out.println("før float");
19         int yf[];
20         // System.out.println("efter float");
21         // length = sample.length;
22         y = new int[length];
23         yf = new int[length];
24         /* Coefficients for a High-pass Butterworth filter */
25         /* (<3 kHz at 8000 samples/s) */
26         // int a[] = {0,((int) (2.4744 * 65536.0 + 0.5)) ,((int) (2.811 *
27         // 65536.0 + 0.5)) , ((int) (1.7038 * 65536.0 + 0.5)), ((int) (0.5444 *
28         // 65536.0 + 0.5)), ((int) (0.0723 * 65536.0 + 0.5))};
29         // int b[] = {((int) (0.0033 * 65536.0 + 0.5)), ((int) (-0.0164 *
30         // 65536.0 + 0.5)), ((int) (0.0328 * 65536.0 + 0.5)), ((int) (-0.0328 *
31         // 65536.0 + 0.5)), ((int) (0.0164 * 65536.0 + 0.5)), ((int) (-0.0033 *
32         // 65536.0 + 0.5))};
33         /* Coefficients for a Low-pass Butterworth filter */
34         /* (>3 kHz at 8000 samples/s) */
35         int a[] = { 0, ((int) (2.4744 * 65536.0 + 0.5)),
36                 ((int) (2.811 * 65536.0 + 0.5)),
37                 ((int) (1.7038 * 65536.0 + 0.5)),
38                 ((int) (0.5444 * 65536.0 + 0.5)),
39                 ((int) (0.0723 * 65536.0 + 0.5)) };
40         int b[] = { ((int) (0.2689 * 65536.0 + 0.5)),
41                 ((int) (1.3447 * 65536.0 + 0.5)),
42                 ((int) (2.6894 * 65536.0 + 0.5)),
43                 ((int) (2.6894 * 65536.0 + 0.5)),
```

```
                ((int) (1.3447 * 65536.0 + 0.5)),
                ((int) (0.2689 * 65536.0 + 0.5)) };
        // float a[] = new float[6];
        // System.out.println("efter float");
        // //a[0] = 0;
        // //System.out.println("efter float1");
        // a[1] = 65536;//0;//(float) 2.4744;0.
        // a[2] = 65536;//0;//(float) 2.811;
        // a[3] = 65536;//0;//(float) 1.7038;
        // a[4] = 65536;//0;//(float) 0.5444;
        // a[5] = 65536;//0;//(float) 0.0723;
        // // //System.out.println("efter float");
        // //
        // // float b[] = new float[6];
        // b[0] = 65536;//(float) 0.2689;
        // b[1] = 65536;//(float) 1.3447;
        // b[2] = 65536;//(float) 2.6894;
        // b[3] = 65536;//(float) 2.6894;
        // b[4] = 65536;//(float) 1.3447;
        // b[5] = 65536;//(float) 0.2689;

        // System.out.println("efter float2");

        // int dg = ((int) (0.01 * 65536.0 + 0.5));
        // System.out.println("efter float2" + dg );
        // float dg1 = 1.5f;
        // //System.out.println("efter float2" + dg1 );
        // float dg3 = 1.5f;
        // float dg2 = dg3*dg1;
        // yf[0] = 0;//(float) 0.2689;
        // yf[1] = 0;//(float) 1.3447;
        // yf[2] = 0;//(float) 2.6894;
        // yf[3] = 0;//(float) 2.6894;
        // yf[4] = 0;//(float) 1.3447;
        // yf[5] = 0;//(float) 0.2689;
        // System.out.println("Før For ");
        for (i = 0; i < length; i++)
            sample[i] = (int) (sample[i] * (65535 / 1250));
        for (i = 6; i < length; i++) {
            yf[i] = (Mul(b[0], intToFP(sample[i]))
                    + Mul(b[1], intToFP(sample[i - 1]))
                    + Mul(b[2], intToFP(sample[i - 2]))
                    + Mul(b[3], intToFP(sample[i - 3]))
                    + Mul(b[4], intToFP(sample[i - 4]))
                    + Mul(b[5], intToFP(sample[i - 5])) - Mul(a[1], yf[i - 1])
                    - Mul(a[2], yf[i - 2]) - Mul(a[3], yf[i - 3])
                    - Mul(a[4], yf[i - 4]) - Mul(a[5], yf[i - 5]));
            // System.out.print(" før cast to int");
            y[i] = toInt(yf[i]);
            // System.out.print("I For ");
        }
        for (i = 0; i < length; i++)
            y[i] = (int) (y[i] / (65535 / 1250));

        return y;
    }
}
```

Listing B.6: Filter.java

B.3.7 Signal

```
package jopspeech.frontend;

/**
 * Embedded Java Speech Recognition on JOP <br />
```

```
 * Website: <a href="http://www.jopspeech.com">www.jopspeech.com</a>
 *
 * @author Mikael Lundsgaard and Jens Kristian Rasmussen
 * @version 0.1.0
 *
 * The Signal class is the basic for a signal that is entering the system in the
 * Front-end. The Signal class pre-defines variables which contain information
 * about signal and needs to be extended in a Static class. This means that
 * there can be only one signal at a time being processed in the frontend
 * component, the signal works in this way as a buffer.
 *
 */
public abstract class Signal {
    public int[] raw_sample;

    public int signalID;

    public String classID = "UNKNOWN";

    public int length;

    public int numberFrames;

    public int[][] frames;

}
```

Listing B.7: Signal.java

B.3.8 Utterance

```
package jopspeech.frontend;

import jopspeech.analysis.Window;

/**
 * Embedded Java Speech Recognition on JOP <br />
 * Website: <a href="http://www.jopspeech.com">www.jopspeech.com</a>
 *
 * @author Mikael Lundsgaard and Jens Kristian Rasmussen
 * @version 0.1.0
 *
 * The utterance of a word with the raw sample
 */
public class Utterance extends Signal {

    private static Utterance single;

    public static Utterance init(int raw_sampleLength, int frameSize) {
        if (single != null)
            return single;

        single = new Utterance();
        single.raw_sample = new int[raw_sampleLength];
        single.frames = new int[((100 / Window.OVERLAYPRC) * raw_sampleLength / frameSize) - 1][frameSize];
        return single;
    }
}
```

Listing B.8: Utterance.java

B.3.9 Analysis

```
package jopspeech.analysis;

```

```
3  /**
4   * Embedded Java Speech Recognition on JOP <br />
5   * Website: <a href="http://www.jopspeech.com">www.jopspeech.com</a>
6   *
7   * @author Mikael Lundsgaard and Jens Kristian Rasmussen
8   * @version 0.1.0
9   *
10  * <p>
11  * The abstract class Analysis supplies some methods for how to define a worker
12  * class extending for this in the Analysis package
13  * <p>
14  * The Analysis component contains the following classes <img src="../../resources/Analysis.png"/>
15  *
16  */
17 public abstract class Analysis {
18
19     private int ready;
20
21     private int done = 0;
22
23     /**
24      * The Init methods insures that memory is allocated, threads are created
25      * and all non-time critical methods are initialized
26      *
27      * @return The Analysis object after initilizatiuon
28      */
29     public static Analysis init() {
30         return null;
31     }
32
33     /**
34      * The ectending class defines the work performed in the Component.
35      */
36     protected abstract void work();
37
38     /**
39      * The mission method defines how the work of the extending class should be
40      * performed.
41      */
42     public void mission() {
43         if (ready == 1) {
44             ready = 0;
45             work();
46             done = 1;
47         }
48     }
49
50     /**
51      * The setReady method sets the Component ready to perform some work
52      *
53      * @return 1 for ready
54      */
55     public synchronized int setReady() {
56         if (ready == 1) {
57             return ready;
58         } else {
59             ready = 1;
60             done = 0;
61             return ready;
62         }
63     }
64
65     /**
66      * Handels communication with other processes or threads that needs to
67      * communicate with the extending component
68      *
69      * @return 1 if it is ready to perform some work
70      */
```

```
71      public synchronized int getDone() {
72          return done;
73      }
74  }
```

Listing B.9: Analysis.java

B.3.10 AnalysisWorker

```
1   package jopspeech.analysis;
2
3   import jopspeech.JopSpeech;
4   import jopspeech.frontend.FrontEndWorker;
5   import jopspeech.recognizer.ModelImpl;
6   import jopspeech.utility.FixPoint;
7   import jopspeech.utility.Protocol;
8
9   /**
10   * Embedded Java Speech Recognition on JOP <br />
11   * Website: <a href="http://www.jopspeech.com">www.jopspeech.com</a>
12   *
13   * @author Mikael Lundsgaard and Jens Kristian Rasmussen
14   * @version 0.1.0
15   *
16   * <p>
17   * This class is an extentions of the Analysis Class showing what
18   * work can be done in the Analysis component
19   */
20  public class AnalysisWorker extends Analysis {
21
22      public static final int NUMBERCOEFFICIENTS = 14;
23
24      public static final int FRAMESIZE = 128;
25
26      private static AnalysisWorker single;
27
28      private static Algorithm fft;
29
30      private static Algorithm lpc;
31
32      private static Algorithm window;
33
34      public static Template template;
35
36      public static Analysis init() {
37
38          if (single != null)
39              return single;
40          single = new AnalysisWorker();
41          window = Window.init(Window.HAMMING, FRAMESIZE, FixPoint.SHIFT);
42          fft = FFT.init();
43          lpc = LPC.init();
44          template = new Word(FrontEndWorker.SAMPELSIZE, FRAMESIZE,
45                  Window.OVERLAYPRC, NUMBERCOEFFICIENTS, Protocol.TESTWORD,
46                  "UNKNOWN");
47          return single;
48      }
49
50      protected void work() {
51          // Window Algorithm
52          window.process(0, FrontEndWorker.utterance, template);
53
54          // FFT Algorithm
55          fft.process(1, FrontEndWorker.utterance, template);
56
57          // IFFT Algorithm
```

```
58          fft.process(-1, FrontEndWorker.utterance, template);
59
60          // LPC Algorithm
61          lpc.process(0, FrontEndWorker.utterance, template);
62
63          // If the template is not unknown add it to the Model
64          if (!FrontEndWorker.utterance.classID.regionMatches(0, "UNKNOWN", 0, 7)) {
65              // Adding Template
66              ModelImpl.addTemplate(FrontEndWorker.utterance, template,
67                      NUMBERCOEFFICIENTS);
68          }
69          // Else start the recognizer and it will have defined
70          // in its work method what to do with the template
71          else {
72              JopSpeech.recognizer.setReady();
73          }
74      }
75  }
```

Listing B.10: AnalysisWorker.java

B.3.11 Algorithm

```
1  package jopspeech.analysis;
2
3  import jopspeech.frontend.Signal;
4
5  /**
6   * Embedded Java Speech Recognition on JOP <br />
7   * Website: <a href="http://www.jopspeech.com">www.jopspeech.com</a>
8   *
9   * @author Mikael Lundsgaard and Jens Kristian Rasmussen
10  * @version 0.1.0
11  *
12  * The interface Algorithms interface defines how different algorithms should
13  * process the signal in the Analysis component they receive and what should be
14  * returned.
15  */
16 public interface Algorithm {
17      /**
18       * The process method is the analysis of a given signal. The signal is
19       * transformed into a Template format
20       *
21       * @param type
22       * @param signal
23       *            The signal to be analysid
24       * @param template
25       *            The result after the process of the signal, Notice that not
26       *            all algorithms uses this
27       */
28      public void process(int type, Signal signal, Template template);
29  }
```

Listing B.11: Algorithm.java

B.3.12 FFT

```
1  package jopspeech.analysis;
2
3  import jopspeech.frontend.Signal;
4  import jopspeech.utility.FixPoint;
5
6  /**
7   *
8   * Embedded Java Speech Recognition on JOP <br />
```

```
 9  * Website: <a href="http://www.jopspeech.com">www.jopspeech.com</a>
10  *
11  * @author Mikael Lundsgaard and Jens Kristian Rasmussen
12  * @version 0.1.0
13  *
14  * This FFT Class is a implementation of the algorithm interface specifying how
15  * the processing of a signal can be done on JOP. The class FFT is used to
16  * convert a sound signal from its time domain into its frequency domain, or
17  * back again by tacking the inverse operation.
18  */
19 public class FFT implements Algorithm {
20     public static int setupWordsDone = -1;
21
22     public static int done = 0;
23
24     static int[] compSample;
25
26     private static int wprArray[], wpiArray[];
27
28     private static FFT single;
29
30     /**
31      *
32      * The init method of the FFT Method, must be initialized before the FFT can
33      * be used
34      *
35      * @return FFT Algorithm object
36      *
37      */
38     public static FFT init() {
39         if (single != null)
40             return single;
41
42         compSample = new int[AnalysisWorker.FRAMESIZE * 2];
43         // frequency values for the REAL part on the UNIT circle
44         wprArray = new int[] { -131072, -65536, -19195, -4989, -1259, -316,
45                 -79, -20, -5, -1, 0 };
46         // frequency values for the Imaginary part on the UNIT circle
47         wpiArray = new int[] { 0, 65536, 46341, 25080, 12785, 6424, 3216, 1608,
48                 804, 402, 201 };
49         single = new FFT();
50         return single;
51     }
52
53     /**
54      * @param type
55      *             1 if the FFT needs to be calculated -1 if the inverse FFT
56      *             (IFFT)
57      * @param signal
58      *             The Signal to convert into its frequency domain, or time
59      *             domain by IFFT. if FFT the power spectrum will be calculated
60      * @param template
61      *             Not used - ignored
62      */
63     public void process(int type, Signal signal, Template template) {
64         if (type == 1)
65             fft(1, compSample, signal);
66         else if (type == -1)
67             fft(-1, compSample, signal);
68     }
69
70     /**
71      *
72      * @param type
73      *             1 if the FFT needs to be calculated -1 if the inverse FFT
74      *             (IFFT)
75      * @param compSample
76      *             the complex Sample to return from FFT
```

```
77      * @param signal
78      *            The Signal to convert into its frequency domain, or time
79      *            domain by IFFT
80      */
81     private static void fft(int type, int[] compSample, Signal signal) {
82         int i = 0, j = 0;
83         int limitReal = 0;
84         int limitImg = 0;
85 
86         for (i = 0; i < signal.numberFrames; i++) {
87             // opretter et complex array af double størrelse af orginal
88             // array
89 
90             for (j = 0; j < AnalysisWorker.FRAMESIZE; j++) {
91                 // REAL DEL
92                 compSample[j * 2] = signal.frames[i][j];
93                 // IMG DEL
94                 compSample[j * 2 + 1] = 0;
95             }
96             // tager en FFT på fixpoint.
97             basicFft(compSample, AnalysisWorker.FRAMESIZE, type,
98                     FixPoint.SHIFT, wprArray, wpiArray);
99 
100            // udregner power spectrum
101            if (type == 1) {
102                for (j = 0; j < signal.frames[i].length; j++) {
103 
104                    limitReal = FixPoint.maxShift(compSample[2 * j],
105                            FixPoint.SHIFT);
106                    limitImg = FixPoint.maxShift(compSample[2 * j + 1],
107                            FixPoint.SHIFT);
108                    compSample[2 * j] >>= limitReal;
109                    compSample[2 * j + 1] >>= limitImg;
110                    // Her udregnes img og real deal opløfted i anden
111                    compSample[2 * j] = FixPoint.mul(compSample[2 * j],
112                            compSample[2 * j], (FixPoint.SHIFT - limitReal));
113                    compSample[2 * j + 1] = FixPoint.mul(compSample[2 * j + 1],
114                            compSample[2 * j + 1], (FixPoint.SHIFT - limitImg));
115 
116                    // tjekker hvilken der er shifted mest
117                    if (limitImg > limitReal) {
118                        signal.frames[i][j] = (compSample[2 * j] >> (limitImg - limitReal))
119                                + compSample[2 * j + 1];
120 
121                        signal.frames[i][j] = FixPoint.logBase(
122                                signal.frames[i][j],
123                                (10 << (FixPoint.SHIFT - limitImg)),
124                                (FixPoint.SHIFT - limitImg));
125 
126                        signal.frames[i][j] <<= limitImg;
127                    } else {
128                        signal.frames[i][j] = compSample[2 * j]
129                                + (compSample[2 * j + 1] >> (limitReal - limitImg));
130                        signal.frames[i][j] = FixPoint.logBase(
131                                signal.frames[i][j],
132                                (10 << (FixPoint.SHIFT - limitReal)),
133                                (FixPoint.SHIFT - limitReal));
134 
135                        signal.frames[i][j] <<= limitReal;
136                    }
137                    signal.frames[i][j] = FixPoint.mul(signal.frames[i][j],
138                            10 << FixPoint.SHIFT, FixPoint.SHIFT);
139 
140                }
141 
142            } else {
143                for (j = 0; j < signal.frames[i].length; j++)
144                    signal.frames[i][j] = FixPoint.div(compSample[j * 2],
```

```
145                             AnalysisWorker.FRAMESIZE << FixPoint.SHIFT,
146                             FixPoint.SHIFT);
147             }
148         }
149     }
150
151     /**
152      * Base method for performing a Fast Fourier Transform Based on the
153      * Numerical Recipes procedure four1 If isign is set to +1 this method
154      * replaces fftData[0 to 2*nn-1] by its discrete Fourier Transform If isign
155      * is set to -1 this method replaces fftData[0 to 2*nn-1] by nn times its
156      * inverse discrete Fourier Transform nn MUST be an integer power of 2. This
157      * is not checked for in this method, fastFourierTransform(...), for speed.
158      * If not checked for by the calling method, e.g. powerSpectrum(...) does,
159      * the method checkPowerOfTwo() may be used to check this. The real and
160      * imaginary parts of the data are stored adjacently i.e. fftData[0] holds
161      * the real part, fftData[1] holds the corresponding imaginary part of a
162      * data point data array and data array length over 2 (nn) transferred as
163      * arguments result NOT returned to this.transformedDataFft Based on the
164      * Numerical Recipes procedure four1
165      *
166      *
167      */
168     private static void basicFft(int[] dataFixPoint, int nn, int isign,
169             int SHIFT, int[] wprArray, int[] wpiArray) {
170         int dtemp = 0;
171         int wtemp = 0;
172         int tempr = 0;
173         int tempi = 0;
174         int wr = 0;
175         int wpr = 0;
176         int wpi = 0;
177         int wi = 0;
178         int istep = 0;
179         int m = 0;
180         int mmax = 0;
181         int cnt = 0;
182         int ii = 0;
183         int i = 0;
184         int n = nn << 1;
185         int j = 1;
186         int jj = 0;
187         for (i = 1; i < n; i += 2) {
188             jj = j - 1;
189             if (j > i) {
190                 ii = i - 1;
191                 dtemp = dataFixPoint[jj];
192                 dataFixPoint[jj] = dataFixPoint[ii];
193                 dataFixPoint[ii] = dtemp;
194                 dtemp = dataFixPoint[jj + 1];
195                 dataFixPoint[jj + 1] = dataFixPoint[ii + 1];
196                 dataFixPoint[ii + 1] = dtemp;
197             }
198             m = n >> 1;
199             while (m >= 2 && j > m) {
200                 j -= m;
201                 m >>= 1;
202             }
203             j += m;
204         }
205         mmax = 2;
206         while (n > mmax) {
207             istep = mmax << 1;
208
209             wpr = wprArray[cnt] >> (16 - SHIFT);
210
211             wpi = (wpiArray[cnt] >> (16 - SHIFT)) * isign;
212
```

```
213             wr = 1 << SHIFT;
214             wi = 0;
215             for (m = 1; m < mmax; m += 2) {
216                 for (i = m; i <= n; i += istep) {
217                     ii = i - 1;
218                     jj = ii + mmax;
219
220                     tempr = FixPoint.mul(wr, dataFixPoint[jj], SHIFT)
221                             - FixPoint.mul(wi, dataFixPoint[jj + 1], SHIFT);
222
223                     tempi = FixPoint.mul(wr, dataFixPoint[jj + 1], SHIFT)
224                             + FixPoint.mul(wi, dataFixPoint[jj], SHIFT);
225
226                     dataFixPoint[jj] = dataFixPoint[ii] - tempr;
227
228                     dataFixPoint[jj + 1] = dataFixPoint[ii + 1] - tempi;
229                     dataFixPoint[ii] += tempr;
230                     dataFixPoint[ii + 1] += tempi;
231                 }
232
233                 wr = FixPoint.mul((wtemp = wr), wpr, SHIFT)
234                         - FixPoint.mul(wi, wpi, SHIFT) + wr;
235                 wi = FixPoint.mul(wi, wpr, SHIFT)
236                         + FixPoint.mul(wtemp, wpi, SHIFT) + wi;
237
238             }
239             mmax = istep;
240             cnt++;
241         }
242     }
243 }
```

Listing B.12: FFT.java

B.3.13 LPC

```
1  package jopspeech.analysis;
2
3  import jopspeech.frontend.Signal;
4  import jopspeech.utility.FixPoint;
5
6  /**
7   * Embedded Java Speech Recognition on JOP <br />
8   * Website: <a href="http://www.jopspeech.com">www.jopspeech.com</a>
9   *
10  * @author Mikael Lundsgaard and Jens Kristian Rasmussen
11  * @version 0.1.0
12  *
13  * <p>
14  * This class is for doing the LPC. The methods in this class is created with
15  * inspiration from Fundamentals Of Speech Recognition by Lawrence Rabiner and
16  * Biing-Hwang Juang
17  * <p>
18  * LPC of a signal with 20 coffiecients <img src="../../resources/LPC20cof.png" title="Hamming"/>
19  *
20  *
21  */
22 public class LPC implements Algorithm {
23
24     public static int setupWordsDone = -1;
25
26     public static int done = 0;
27
28     private static LPC single;
29
30     static int autoCoeff[];
```

```
31
32      /**
33       * This is the init methode it needs to be called before the Class can work
34       *
35       * @return LPC algorithm object
36       */
37      public static LPC init() {
38          if (single != null)
39              return single;
40
41          autoCoeff = new int[AnalysisWorker.NUMBERCOEFFICIENTS + 1];
42
43          single = new LPC();
44          return single;
45      }
46
47      public void process(int type, Signal signal, Template template) {
48          int l;
49
50          for (l = 0; l < signal.numberFrames; l++) {
51
52              autocorrelation(autoCoeff.length, AnalysisWorker.FRAMESIZE / 2,
53                      signal.frames[l], autoCoeff, FixPoint.SHIFT);
54              lpcCoeff(template.coefficients[l], autoCoeff,
55                      AnalysisWorker.NUMBERCOEFFICIENTS, FixPoint.SHIFT);
56          }
57      }
58
59      /**
60       * Do the actual LPC
61       *
62       * @param lpcCoeff
63       * @param autoCoeff
64       * @param numCoeff
65       * @param SHIFT
66       * @return Error from the LPC cofficients, The b value of a function y= ax +
67       *         b
68       */
69      private static int lpcCoeff(int[] lpcCoeff, int[] autoCoeff, int numCoeff,
70              int SHIFT) {
71          int i, j, tmp;
72          int r = autoCoeff[0];
73          int error = autoCoeff[0];
74
75          if (autoCoeff[0] == 0) {
76              return 0;
77          }
78          for (i = 0; i < numCoeff; i++) {
79              /* Sum up this iteration's reflection coefficient. */
80              r = -autoCoeff[i + 1];
81              for (j = 0; j < i; j++)
82                  r -= FixPoint.mul(lpcCoeff[j], autoCoeff[i - j], SHIFT);
83
84              r = FixPoint.div(r, error, SHIFT);
85              /* Update LPC coefficients and total error. */
86              lpcCoeff[i] = r;
87              for (j = 0; j < i / 2; j++) {
88                  tmp = lpcCoeff[j];
89                  lpcCoeff[j] += FixPoint.mul(r, lpcCoeff[i - 1 - j], SHIFT);
90                  lpcCoeff[i - 1 - j] += FixPoint.mul(r, tmp, SHIFT);
91              }
92              if ((i % 2) != 0)
93                  lpcCoeff[j] += FixPoint.mul(lpcCoeff[j], r, SHIFT);
94              error = FixPoint.mul(((1 << SHIFT) - FixPoint.mul(r, r, SHIFT)),
95                      error, SHIFT);
96          }
97          return error;
98      }
```

```
99
100     /**
101      * Do the auto correlation
102      *
103      * @param coffSize
104      * @param frameSize
105      * @param frame
106      * @param R
107      * @param SHIFT
108      */
109     private static void autocorrelation(int coffSize, int frameSize,
110             int frame[], int[] R, int SHIFT) {
111         int p = coffSize;
112         int k = 0;
113         int sum = 0;
114         while (p--> 0) {
115             for (k = p, sum = 0; k < frameSize; k++) {
116                 sum += FixPoint.mul(frame[k], frame[k - p], SHIFT);
117             }
118             R[p] = sum;
119         }
120     }
121
122 }
```

Listing B.13: LPC.java

B.3.14 MFCC

```
1  package jopspeech.analysis;
2
3  import jopspeech.frontend.Signal;
4  import jopspeech.utility.FixPoint;
5
6  /**
7   * Embedded Java Speech Recognition on JOP <br />
8   * Website: <a href="http://www.jopspeech.com">www.jopspeech.com</a>
9   *
10  * @author Mikael Lundsgaard and Jens Kristian Rasmussen
11  * @version 0.1.0
12  */
13 public class MFCC implements Algorithm {
14
15     public static int setupMFCC = -1;
16
17     public static int[] initWeights;
18
19     public static int[] melCosbank;
20
21     public static int done = 0;
22
23     private static MFCC single;
24
25     /*************************************************************************
26      * This is the init methode it needs to be called befor the thread can work
27      * Use only if MelFilter array is accesible already for calcFPMelCepCof
28      *
29      * @param numFilters
30      *            the number of filters
31      * @param filterLength
32      */
33     public static MFCC init(int numFilters, int filterLength) {
34
35         if (single != null)
36             return single;
37         melCosbank = new int[numFilters * numFilters];
```

```
38          initWeights = new int[filterLength];
39          single = new MFCC();
40          return single;
41          // createMelCosBank(shift);
42      }
43
44      /**
45       * The process method implemented from interface algorithm
46       */
47      public void process(int type, Signal signal, Template template) {
48
49          for (int i = 0; i < signal.numberFrames; i++) {
50              calcMelFrqCepCof(signal.frames[i], template.coefficients[i],
51                      AnalysisWorker.NUMBERCOEFFICIENTS, FixPoint.SHIFT);
52
53              DCT(template.coefficients[i], AnalysisWorker.NUMBERCOEFFICIENTS,
54                      FixPoint.SHIFT);
55          }
56      }
57
58      /**
59       * Calculates the Mel cepstral cofficients
60       *
61       * input data is the Frame to be calculated and the vector to put the
62       * calculated mel cofficients into
63       *
64       * @param frame
65       * @param melCepCofficients
66       */
67      public static void calcMelFrqCepCof(int frame[], int melCepCofficients[],
68              int numFilters, int shift) {
69
70          int res = 0;
71          int p = 2;
72          int i;
73          int j;
74          // The firste frequencr index where it all starts
75          int count;
76
77          // The length of the first set of wights
78          int size = 0;
79
80          for (j = 0; j < numFilters; j++) {
81              count = initWeights[size];
82              for (i = 0; i < initWeights[size + 1]; i++, p++) {
83                  res += FixPoint.mul(frame[i + count], initWeights[p], shift);
84              }
85
86              p += 2;
87              size += initWeights[size + 1] + 2;
88              melCepCofficients[j] = res;
89              res = 0;
90          }
91
92      }
93
94      /*************************************************************************
95       * The Discrete Cosines transform algorithm only works after
96       * createMelCosBank is initialized
97       *
98       * @param melCepCofficients
99       * @param numFilters
100      * @param shift
101      */
102     public static void DCT(int melCepCofficients[], int numFilters, int shift) {
103         int j;
104         // do the DCT on the Mel filter bank
105         for (int l = 0; l < numFilters; l++)
```

```
106             for (j = 0; j < numFilters; j++) {
107                 melCepCofficients[l] += FixPoint.mul(melCepCofficients[j],
108                         melCosbank[numFilters * l + j], shift);
109             }
110     }
111
112     /**
113      * Computes the Mel cosinus bank
114      *
115      * @param numFilters
116      * @param shift
117      */
118     private static void createMelCosBank(int numFilters, int shift) {
119         int temp;
120         temp = FixPoint.div(FixPoint.PI >> (29 - shift), numFilters << shift,
121                 shift);
122         for (int l = 1; l <= numFilters; l++) {
123             for (int m = 1; m <= numFilters; m++) {
124                 melCosbank[(m - 1) + (l - 1) * numFilters] = FixPoint.cos(
125                         FixPoint.mul(l << shift, FixPoint.mul(temp,
126                                 ((m << shift) - (1 << (shift - 1))), shift),
127                                 shift), shift);
128             }
129         }
130
131     }
132
133 }
```

Listing B.14: MFCC.java

B.3.15 Window

```
1  package jopspeech.analysis;
2
3  import jopspeech.frontend.Signal;
4  import jopspeech.utility.*;
5
6  /**
7   * Embedded Java Speech Recognition on JOP <br />
8   * Website: <a href="http://www.jopspeech.com">www.jopspeech.com</a>
9   *
10  * @author Mikael Lundsgaard and Jens Kristian Rasmussen
11  * @version 0.1.0
12  *
13  * <p>
14  * Window class for converting a time serie of sound into window frames
15  * <p>
16  * Hamming Window and Hann Window( Reprinted with respect to author see ( <a
17  * href="http://en.wikipedia.org/wiki/Window_Function">http://en.wikipedia.org/wiki/Window_Function</a> )
18  * <br />
19  * <br />
20  * <img src="../../resources/Windowhamming.png"
21  * title="Hamming"/>
22  *
23  * <img src="../../resources/Windowhann.png" title="Hann"/>
24  */
25 public class Window implements Algorithm {
26     public final static int HANNING = 1;
27
28     public final static int HAMMING = 2;
29
30     public final static int OVERLAYPRC = 50;
31
32     static int arg = 0;
33
```

```
34      static int windowValue = 0;
35
36      static int[] windowValues;
37
38      private static Window single;
39
40      /**
41       * The init method of the window functions, must be declared with the window
42       * type before this can be used
43       *
44       * @param type
45       *            Windowtype = 1 for Hanning, 2 for Hamming
46       * @param windowSize
47       *            size of window must be 2 exp.
48       * @param SHIFT
49       * @return Singelton object
50       */
51      public static Algorithm init(int type, int windowSize, int SHIFT) {
52          if (single != null)
53              return single;
54          int arg;
55          int k;
56          windowValues = new int[windowSize];
57          arg = FixPoint.div((FixPoint.PI >> (28 - SHIFT)),
58                  (windowSize - 1) << SHIFT, SHIFT);
59          switch (type) {
60          case HANNING:
61              for (k = 0; k < windowSize; k++) {
62                  windowValues[k] = (1 << (SHIFT - 1));
63                  windowValues[k] -= (FixPoint.mul((1 << (SHIFT - 1)), (FixPoint
64                          .cos((FixPoint.mul(arg, (k << SHIFT), SHIFT)), SHIFT)),
65                          SHIFT));
66              }
67              break;
68          // 579820584 =0.54 shift 30 493921239=0.46 shift 30
69          case HAMMING:
70              for (k = 0; k < windowSize; k++) {
71                  windowValues[k] = (579820584 >> (30 - SHIFT));
72                  windowValues[k] -= (FixPoint.mul((493921239 >> (30 - SHIFT)),
73                          (FixPoint.cos((FixPoint.mul(arg, (k << SHIFT), SHIFT)),
74                                  SHIFT)), SHIFT));
75              }
76              break;
77          }
78          single = new Window();
79          return single;
80      }
81
82      /**
83       * @param type
84       *            Window type
85       * @param signal
86       *            Signal to be windowed
87       * @param template
88       *            Not used
89       */
90      public void process(int type, Signal signal, Template template) {
91          signal.numberFrames = winOverlay(signal.raw_sample, signal.length,
92                  signal.frames, AnalysisWorker.FRAMESIZE, Window.HAMMING,
93                  FixPoint.SHIFT);
94      }
95
96      /**
97       * Calculates the window function on one window/frame in Fix point given by
98       * the SHIFT value
99       *
100      * @param type
101      * @param windowSize
```

```
102      * @param frame
103      * @param SHIFT
104      */
105     private void window(int type, int windowSize, int[] frame, int SHIFT) {
106         int k;
107         arg = FixPoint.div((FixPoint.PI >> (28 - SHIFT)),
108                 (windowSize - 1) << SHIFT, SHIFT);
109 
110         switch (type) {
111         case HANNING:
112             for (k = 0; k < windowSize; k++) {
113                 windowValue = (1 << (SHIFT - 1));
114                 windowValue -= (FixPoint.mul((1 << (SHIFT - 1)), (FixPoint.cos(
115                         (FixPoint.mul(arg, (k << SHIFT), SHIFT)), SHIFT)),
116                         SHIFT));
117                 frame[k] = FixPoint.mul(frame[k], windowValue, SHIFT);
118             }
119             break;
120         // 579820584 =0.54 shift 30 493921239=0.46 shift 30
121         case HAMMING:
122             for (k = 0; k < windowSize; k++) {
123                 windowValue = (579820584 >> (30 - SHIFT));
124                 windowValue -= (FixPoint.mul((493921239 >> (30 - SHIFT)),
125                         (FixPoint.cos((FixPoint.mul(arg, (k << SHIFT), SHIFT)),
126                                 SHIFT)), SHIFT));
127                 frame[k] = FixPoint.mul(frame[k], windowValue, SHIFT);
128             }
129             break;
130         }
131     }
132 
133     /**
134      * sample framsize and overLay ust be in modolus 2 compliant
135      *
136      * @param sample
137      * @param sampleLength
138      * @param frames
139      * @param frameSize
140      * @param winType
141      * @param SHIFT
142      * @return The resulting number of frames except the last one since it as a
143      *         minimum will consist of 50 % zeros
144      */
145     private static int winOverlay(int sample[], int sampleLength,
146             int[][] frames, int frameSize, int winType, int SHIFT) {
147         // number of frames in a 50 % overlay is the double
148         int numberFrames = (100 / OVERLAYPRC) * sampleLength / frameSize;
149         int overlay = frameSize * OVERLAYPRC / 100;
150         int i, j;
151         for (i = 0; i < numberFrames - 1; i++) {
152             for (j = 0; j < frameSize && sampleLength > (i * overlay + j); j++) {
153                 frames[i][j] = sample[i * overlay + j];
154                 // make the window
155                 frames[i][j] = FixPoint.mul(frames[i][j], windowValues[j],
156                         SHIFT);
157             }
158         }
159         return numberFrames - 1;
160     }
161 
162 }
```

Listing B.15: Window.java

B.3.16 Template

```java
package jopspeech.analysis;

/**
 * Embedded Java Speech Recognition on JOP <br />
 * Website: <a href="http://www.jopspeech.com">www.jopspeech.com</a>
 *
 * @author Mikael Lundsgaard and Jens Kristian Rasmussen
 * @version 0.1.0
 * <p>
 *
 * The Template class defines a structure that is used to save the analyzed
 * signals information. Eg. if the goal of the developer is to save a template
 * with a phoneme, or word, or full sentence then this is extended from the
 * abstract Template Class. To identify each template a classID is defined. As
 * with the Signal there should only be one fixed size instance of this in the
 * Analysis component to insure real-time predictability and memory reusability.
 */
public abstract class Template {

    public int[][] coefficients;

    public int numberFrames;

    public int templateID;

    public String classID;
}
```

Listing B.16: Template.java

B.3.17 Word

```java
package jopspeech.analysis;

/**
 * Embedded Java Speech Recognition on JOP <br />
 * Website: <a href="http://www.jopspeech.com">www.jopspeech.com</a>
 *
 * @author Mikael Lundsgaard and Jens Kristian Rasmussen
 * @version 0.1.0
 *
 * <P>
 * This class is a extendtion of the Template of the raw utterance of the word
 */
public class Word extends Template {

    public Word(int sampleLength, int frameSize, int overLayPrc, int numCoeff,
            int wordID, String classID) {

        this.templateID = wordID;
        this.classID = classID;
        numberFrames = ((100 / overLayPrc) * sampleLength / frameSize) - 1;
        coefficients = new int[numberFrames][numCoeff + 1];

    }

    public Word(int numberFrames, int numCoeff, String classID, int wordID) {

        this.templateID = wordID;
        this.classID = classID;
        this.numberFrames = numberFrames;
        coefficients = new int[numberFrames][numCoeff + 1];
        System.out.println("New word");
        System.out.println(templateID);
        System.out.println(classID);
        System.out.println(numberFrames);
```

```
35     }
36 }
```

Listing B.17: Word.java

B.3.18 Recognizer

```
1  package jopspeech.recognizer;
2
3  /**
4   *
5   * Embedded Java Speech Recognition on JOP <br />
6   * Website: <a href="http://www.jopspeech.com">www.jopspeech.com</a>
7   *
8   * @author Mikael Lundsgaard and Jens Kristian Rasmussen
9   * @version 0.1.0
10  *
11  * <p>
12  * The Recognizer component contains the following classes <p>
13  * <img src="../../Recognizer.png"/>
14  *
15  */
16 public abstract class Recognizer {
17     private int ready;
18
19     private int done;
20
21     public static Recognizer init() {
22         return null;
23     }
24
25     protected abstract void work();
26
27     public synchronized int setReady() {
28         if (ready == 1) {
29             return ready;
30         } else {
31             ready = 1;
32             done = 0;
33             return ready;
34         }
35     }
36
37     public void mission() {
38         if (ready == 1) {
39             ready = 0;
40             work();
41             done = 1;
42         }
43     }
44
45     public synchronized int getDone() {
46         return done;
47     }
48 }
```

Listing B.18: Recognizer.java

B.3.19 RecognizerWorker

```
1  package jopspeech.recognizer;
2
3  import jopspeech.analysis.AnalysisWorker;
4
```

```
5  /**
6   * Embedded Java Speech Recognition on JOP <br />
7   * Website: <a href="http://www.jopspeech.com">www.jopspeech.com</a>
8   *
9   * @author Mikael Lundsgaard and Jens Kristian Rasmussen
10  * @version 0.1.0
11  *
12  */
13 public class RecognizerWorker extends Recognizer {
14
15     private static Classifier knear;
16
17     private static RecognizerWorker single;
18
19     private static Model model;
20
21     public static Recognizer init() {
22
23         if (single != null)
24             return single;
25         model = ModelImpl.init();
26         knear = KNear.init();
27         single = new RecognizerWorker();
28         return single;
29     }
30
31     protected void work() {
32         // test
33         if (model.trained == 1)
34             knear.test(AnalysisWorker.template, model);
35         // train
36         else if (model.readyToTrain == 1) {
37             knear.train(model);
38             model.readyToTrain = 0;
39             knear.test(AnalysisWorker.template, model);
40         }
41     }
42
43 }
```

Listing B.19: RecognizerWorker.java

B.3.20 Classifier

```
1  package jopspeech.recognizer;
2
3  import jopspeech.analysis.Template;
4
5  /**
6   * Embedded Java Speech Recognition on JOP <br />
7   * Website: <a href="http://www.jopspeech.com">www.jopspeech.com</a>
8   *
9   * @author Mikael Lundsgaard and Jens Kristian Rasmussen
10  * @version 0.1.0
11  *
12  * This interface Classifier defined in the component Recognizer defines how tho
13  * train a model and test a template.
14  */
15 public interface Classifier {
16     /**
17      * The method is meant to train a given model for the recognizer to use when
18      * testing a template.
19      *
20      * @param model
21      *             The model to be trained
22      */
```

```
23      public void train(Model model);
24
25      /**
26       * The test method is the one that classifies a template on a given model
27       * and sets the templates classID.
28       *
29       * @param testTemplate
30       *            The Template to be tested
31       * @param model
32       *            The model to use for testing
33       */
34      public void test(Template testTemplate, Model model);
35
36  }
```

Listing B.20: Classifier.java

B.3.21 KNear

```
1   package jopspeech.recognizer;
2
3   import jopspeech.analysis.AnalysisWorker;
4   import jopspeech.analysis.Template;
5   import jopspeech.utility.FixPoint;
6
7   import com.jopdesign.sys.Const;
8   import com.jopdesign.sys.Native;
9
10  /**
11   * Embedded Java Speech Recognition on JOP <br />
12   * Website: <a href="http://www.jopspeech.com">www.jopspeech.com</a>
13   *
14   * @author Mikael Lundsgaard and Jens Kristian Rasmussen
15   * @version 0.1.0
16   *
17   * <p>
18   * A implementaion of the K-nearest classifying algorithm. Can classify a given
19   * template from a trained model
20   *
21   */
22  public class KNear implements Classifier {
23
24      public static int result, wordID;
25
26      public static String classID = "";
27
28      public static int timeStart = 0, timeEnd = 0;
29
30      public static int testID = 0;
31
32      private static KNear single;
33
34      private static DTW dtw;
35
36      public static KNear init() {
37          if (single != null)
38              return single;
39          dtw = DTW.init();
40          single = new KNear();
41          return single;
42      }
43
44      /**
45       * Method for training a model using a K-near approach
46       *
47       * @param model
```

```
48      *            The Model that needs to be trained, will set treshold value
49      */
50     public void train(Model model) {
51         int numberTemplates = Model.NUMTEMPLATES;
52         int i, j, k = 0;
53         int certainty = 0;
54         for (i = 0; i < numberTemplates; i++) {
55             for (j = 0; j < numberTemplates; j++) {
56                 if (model.templates[i].templateID != model.templates[j].templateID
57                         && model.templates[i].classID.regionMatches(0,
58                                 model.templates[j].classID, 0,
59                                 model.templates[i].classID.length())) {
60 
61                     k++;
62                 }
63             }
64         }
65 
66         for (i = 0; i < numberTemplates; i++) {
67             for (j = 0; j < numberTemplates; j++) {
68                 if (model.templates[i].templateID != model.templates[j].templateID
69                         && model.templates[i].classID.regionMatches(0,
70                                 model.templates[j].classID, 0,
71                                 model.templates[i].classID.length())) {
72                     certainty = dtw.dtwVector(model.templates[i].coefficients,
73                             model.templates[i].numberFrames,
74                             model.templates[j].coefficients,
75                             model.templates[j].numberFrames,
76                             AnalysisWorker.NUMBERCOEFFICIENTS + 1,
77                             FixPoint.SHIFT, dtw.dist, dtw.temp, dtw.move,
78                             dtw.warp, dtw.globdist);
79                     model.threshold += FixPoint.div(certainty,
80                             k << FixPoint.SHIFT - 2, FixPoint.SHIFT - 2);
81                 }
82             }
83         }
84         System.out.print("Threshold: ");
85         System.out.println(model.threshold);
86         System.out.println("Train end!");
87         model.trained = 1;
88     }
89 
90     /**
91      * The Classification algorithm using a 1-near approach
92      *
93      * @param testTemplate
94      *            The Template to be classfied
95      * @param model
96      *            The trained Model
97      *
98      */
99     public void test(Template testTemplate, Model model) {
100        // System.out.println("KNear test");
101        timeStart = Native.rd(Const.IO_US_CNT);
102        dtw.dtwCalc(testTemplate, model, Model.NUMTEMPLATES,
103                testTemplate.coefficients[0].length - 1, FixPoint.SHIFT);
104        result = (655636 << 10);
105        wordID = 0;
106 
107        testID++;
108        int numWords = Model.NUMTEMPLATES;
109        System.out.print("Now we are testing word num: ");
110        System.out.println(testID);
111        for (int i = 0; i < numWords; i++) {
112 
113            if (dtw.compare[i] <= result) {
114                result = dtw.compare[i];
115                wordID = i + 1;
```

```
                classID = model.templates[i].classID;
            }

        }
        if (result > model.threshold) {
            System.out.println("UNKNOWN_WORD!");
            classID = "UNKNOWN";
            wordID = -1;

        }
        timeEnd = Native.rd(Const.IO_US_CNT);
    }

}
```

Listing B.21: KNear.java

B.3.22 DTW

```
package jopspeech.recognizer;

import jopspeech.analysis.Template;
import jopspeech.utility.FixPoint;

/**
 * Embedded Java Speech Recognition on JOP <br />
 * Website: <a href="http://www.jopspeech.com">www.jopspeech.com</a>
 *
 * @author Mikael Lundsgaard and Jens Kristian Rasmussen
 * @version 0.1.0
 *
 * The Dynamic Time Warping algorithm DTW inspiration for this class where found
 * in Numerical Recipes in C The Art of Scientific Computing, Second Edition by
 * William H. Press, Saul A. Teukolsky, William T. Vetterling, Brian P. Flannery.
 * <p>
 * The Class finds the optimal path between two vectors S1 and S2.
 * <br />
 * <img src="../../resources/dtwMove.jpg"/>
 */
public class DTW {
    public static final int COEFFSIZE = 100;

    int[] compare;

    int[][] dist;

    int[][] temp;

    int[][] move;

    int[][] warp;

    int[][] globdist;

    private static DTW single;

    /**
     * Private constructer
     *
     */
    private DTW() {
        compare = new int[Model.NUMTEMPLATES];
        dist = new int[COEFFSIZE][COEFFSIZE];
        temp = new int[COEFFSIZE * 2][2];
        move = new int[COEFFSIZE][COEFFSIZE];
        warp = new int[COEFFSIZE * 2][2];
```

```
48          globdist = new int[COEFFSIZE][COEFFSIZE];
49      }
50
51      /**
52       * This is the init methode it needs to be called befor the Class can work
53       *
54       */
55      public static DTW init() {
56          if (single != null)
57              return single;
58          single = new DTW();
59          return single;
60      }
61
62      /**
63       *
64       * @param testTemplate
65       *            The template to be tested
66       * @param model
67       *            Contains Templates
68       * @param numwords
69       *            Number of Templates in the model
70       * @param numbercoff
71       *            Number of Coefficients in each Template
72       * @param shift
73       *            The Fixpoint shift.
74       */
75      protected void dtwCalc(Template testTemplate, Model model, int numwords,
76              int numbercoff, int shift) {
77          for (int i = 0; i < numwords; i++) {
78              compare[i] = dtwVector(testTemplate.coefficients,
79                      testTemplate.numberFrames, model.templates[i].coefficients,
80                      model.templates[i].numberFrames, numbercoff + 1, shift,
81                      dist, temp, move, warp, globdist);
82          }
83
84      }
85
86      /**
87       *
88       * @param test
89       *            Vector 1 to be tested
90       * @param testLength
91       *            The lenght of Vector 1
92       * @param saveCompared
93       *            Vector 2 to be tested
94       * @param saveComparedLength
95       *            The lenght of Vector 2
96       * @param params
97       *            coffiecients
98       * @param SHIFT
99       * @param Dist
100      * @param temp
101      * @param move
102      * @param warp
103      * @param globdist
104      * @return Cepstrum distance between Vector 1 and Vector 2
105      */
106     protected int dtwVector(int[][] test, int testLength, int[][] saveCompared,
107             int saveComparedLength, int params, int SHIFT, int[][] Dist,
108             int[][] temp, int[][] move, int[][] warp, int[][] globdist) {
109         int VERY_BIG = 655636 << 8;
110         int i, j, k, I, n, X, Y, total, top, mid, bot, cheapest;
111
112         {
113             for (i = 0; i < testLength; i++) {
114                 for (j = 0; j < saveComparedLength; j++) {
115                     total = 0;
```

```
                for (k = 0; k < params; k++) {
                    total = total
                            + FixPoint.mul(
                                    ((test[i][k]) - (saveCompared[j][k])),
                                    ((test[i][k]) - (saveCompared[j][k])),
                                    SHIFT);
                }

                if (total < 0) {
                    // System.out.println("overflow");
                    total = 1 << 28;
                }
                if (total > 1 << 29) {
                    // System.out.println("næsten overflow");
                }
                Dist[i][j] = total;
            }
        }

        /* % for first frame, only possible match is at (0,0) */

        globdist[0][0] = Dist[0][0];
        for (j = 1; j < testLength; j++)
            globdist[j][0] = VERY_BIG;

        globdist[0][1] = VERY_BIG;
        globdist[1][1] = globdist[0][0] + Dist[1][1];

        move[1][1] = 2;

        for (j = 2; j < testLength; j++)
            globdist[j][1] = VERY_BIG;

        for (i = 2; i < saveComparedLength; i++) {
            globdist[0][i] = VERY_BIG;
            globdist[1][i] = globdist[0][i - 1] + Dist[1][i];

            for (j = 2; j < testLength; j++) {
                top = globdist[j - 1][i - 2] + Dist[j][i - 1] + Dist[j][i];
                mid = globdist[j - 1][i - 1] + Dist[j][i];
                bot = globdist[j - 2][i - 1] + Dist[j - 1][i] + Dist[j][i];
                if (top < 0) {
                    // System.out.println("top");
                    top = VERY_BIG;
                }
                if (mid < 0) {
                    // System.out.println("mid");
                    mid = VERY_BIG;
                }
                if (bot < 0) {
                    // System.out.println("bot");
                    bot = VERY_BIG;
                }
                if ((top < mid) && (top < bot)) {
                    cheapest = top;
                    I = 1;
                } else if (mid < bot) {
                    cheapest = mid;
                    I = 2;
                } else {
                    cheapest = bot;
                    I = 3;
                }

                /* if all costs are equal, pick middle path */
                if ((top == mid) && (mid == bot))
                    I = 2;
```

```
                    globdist[j][i] = cheapest;
                    move[j][i] = I;
                }
            }
        }
        // System.out.println("End Compute distance matrix");

        // lenght of the 2 vectors
        X = saveComparedLength - 1;
        Y = testLength - 1;
        n = 0;

        warp[n][0] = X;
        warp[n][1] = Y;

        while (X > 0 && Y > 0) {
            n = n + 1;

            if (move[Y][X] == 1) {
                warp[n][0] = X - 1;
                warp[n][1] = Y;
                n = n + 1;
                X = X - 2;
                Y = Y - 1;
            } else if (move[Y][X] == 2) {
                X = X - 1;
                Y = Y - 1;
            } else if (move[Y][X] == 3) {
                warp[n][0] = X;
                warp[n][1] = Y - 1;
                n = n + 1;
                X = X - 1;
                Y = Y - 2;
            } else {
                if (X < Y)
                    Y--;
                else
                    X--;
                // System.out.println("Error: move not defined for X ");

            }
            warp[n][0] = X;
            warp[n][1] = Y;

        }

        /* flip warp */
        for (i = 0; i <= n; i++) {
            temp[i][0] = warp[n - i][0];
            temp[i][1] = warp[n - i][1];

        }

        for (i = 0; i <= n; i++) {
            warp[i][0] = temp[i][0];
            warp[i][1] = temp[i][1];

        }

        I = 0;
        for (i = 0; i <= n; i++) {
            I += (globdist[warp[i][1]][warp[i][0]] >> 2);
        }
        if (I < 0)
            I = 1 << 30;
        return I;
    }
```

```
252 }
```

Listing B.22: DTW.java

B.3.23 Model

```
1  package jopspeech.recognizer;
2
3  import jopspeech.analysis.Template;
4
5  /**
6   * Embedded Java Speech Recognition on JOP <br />
7   * Website: <a href="http://www.jopspeech.com">www.jopspeech.com</a>
8   *
9   * @author Mikael Lundsgaard and Jens Kristian Rasmussen
10  * @version 0.1.0
11  *
12  * The Model class contains information about the model that is used by the
13  * application. The Model contain information regarding the analyzed templates
14  * and information regarding for which threshold values the model can Classify
15  * unknown templates
16  *
17  */
18 public abstract class Model {
19     public static final int NUMTEMPLATES = 20;
20
21     int trained = 0;
22
23     int threshold = 200000;
24
25     int readyToTrain = 0;
26
27     Template[] templates;
28 }
```

Listing B.23: Model.java

B.3.24 ModelImpl

```
1  package jopspeech.recognizer;
2
3  import jopspeech.analysis.Template;
4  import jopspeech.analysis.Word;
5  import jopspeech.frontend.Signal;
6
7  /**
8   * Embedded Java Speech Recognition on JOP <br />
9   * Website: <a href="http://www.jopspeech.com">www.jopspeech.com</a>
10  *
11  * @author Mikael Lundsgaard and Jens Kristian Rasmussen
12  * @version 0.1.0
13  *
14  */
15 public class ModelImpl extends Model {
16
17     private static ModelImpl single;
18
19     public static ModelImpl init() {
20         if (single != null)
21             return single;
22         single = new ModelImpl();
23         single.templates = new Word[NUMTEMPLATES];
24         return single;
25     }
26
```

```
27      public static void addTemplate(Signal knownSignal, Template template,
28              int numCoeff) {
29          single.templates[knownSignal.signalID - 1] = new Word(
30                  knownSignal.numberFrames, numCoeff, knownSignal.classID,
31                  knownSignal.signalID);
32          int i, j;
33          for (i = 0; i < single.templates[knownSignal.signalID - 1].numberFrames; i++) {
34              for (j = 0; j < numCoeff; j++) {
35                  single.templates[knownSignal.signalID - 1].coefficients[i][j] = template.coefficients[i][j];
36              }
37          }
38          if (knownSignal.signalID == NUMTEMPLATES)
39              single.readyToTrain = 1;
40      }
41
42      public void setReadyToTrain() {
43          readyToTrain = 1;
44      }
45
46  }
```

Listing B.24: ModelImpl.java

B.3.25 FixPoint

```
1   package jopspeech.utility;
2
3   /**
4    * Embedded Java Speech Recognition on JOP <br />
5    * Website: <a href="http://www.jopspeech.com">www.jopspeech.com</a>
6    *
7    * @author Mikael Lundsgaard and Jens Kristian Rasmussen
8    * @version 0.1.0
9    *
10   *
11   * This Fix point class is made for the JOPSPEECH SDK and can handle shift
12   * values with full 32 bit integer structure but some can only handle up to 16
13   * bit shifted values. This library is own work, but we would like to thank
14   * Henry Minsky for his FP.java library without this work we would not have know
15   * where to start
16   */
17
18  public class FixPoint {
19
20      /**
21       * SHIFT is the variable that indicates the split between decimal and whole
22       * numbers. A 14 Shift value means that 14 bits are used for the decimal
23       * numbers
24       */
25      public static int SHIFT = 14;
26
27      /**
28       * Table with sine values from 0 to 359 degree. Cosine can use the same
29       * table by moving the index by 90 the tabel is in fixed point 30
30       */
31      private static int[] SIN;
32
33      /**
34       * PI in 29 fixed point
35       */
36      public static int PI = 1686629713;
37
38      /**
39       * PI/2 in 29 fixed point
40       */
41      public static int PI_OVER_2 = 843314857;
```

```
42
43      private static int HALF = 32768;
44
45      private static int log2Arr[];
46
47      private static int lnScale[];
48
49      private static int log0Arr[];
50
51      /**
52       * Must be Initializaed before the full functionality can be used of this
53       * fix point library
54       */
55      public static void init() {
56          initLOG();
57          initSIN();
58      }
59
60      /**
61       * Init Log tables
62       *
63       */
64      private static void initLOG() {
65          // LOG arrays are intialized
66          log2Arr = new int[] { 26573, 14624, 7719, 3973, 2017, 1016, 510, 256,
67                  128, 64, 32, 16, 8, 4, 2, 1, 0, 0, 0 };
68
69          log0Arr = new int[] { -45427, -90852, -136278, -181704 - 227130,
70                  -272557, -317983, -363409, -408835, -454261, -499687, -545113,
71                  -590539, -635965, -681391, -726818, -772244, -817670, -863096,
72                  -908522 };
73          lnScale = new int[] { 0, 45426, 90852, 136278, 181704, 227130, 272557,
74                  317983, 363409, 408835, 454261, 499687, 545113, 590539, 635965,
75                  681391, 726817, 817669, 863095, 908521, 953947, 999374,
76                  1044800, 1090226, 1135652, 1181078, 1226504, 1271930, 1317356 };
77      }
78
79      /**
80       * Init Sin table
81       */
82      private static void initSIN() {
83          SIN = new int[] { 0, 18739379, 37473049, 56195305, 74900443, 93582766,
84                  112236583, 130856211, 149435979, 167970228, 186453311,
85                  204879599, 223243478, 241539355, 259761657, 277904834,
86                  295963357, 313931728, 331804471, 349576144, 367241333,
87                  384794656, 402230767, 419544355, 436730145, 453782903,
88                  470697435, 487468587, 504091252, 520560366, 536870912,
89                  553017922, 568996477, 584801711, 600428808, 615873009,
90                  631129609, 646193961, 661061475, 675727625, 690187940,
91                  704438018, 718473518, 732290163, 745883746, 759250125,
92                  772385229, 785285058, 797945680, 810363241, 822533958,
93                  834454122, 846120104, 857528349, 868675383, 879557810,
94                  890172315, 900515665, 910584710, 920376381, 929887697,
95                  939115760, 948057759, 956710970, 965072759, 973140576,
96                  980911966, 988384560, 995556083, 1002424350, 1008987269,
97                  1015242840, 1021189159, 1026824413, 1032146887, 1037154959,
98                  1041847103, 1046221891, 1050277989, 1054014162, 1057429273,
99                  1060522280, 1063292242, 1065738315, 1067859754, 1069655912,
100                 1071126243, 1072270298, 1073087729, 1073578288, 1073741824 };
101     }
102
103     /**
104      * Multiply two fixed-point numbers with shift 16
105      *
106      * @param f1
107      *            Fixpoint value
108      * @param f2
109      *            Fixpoint value
```

```
110      * @return f1 * f2 with 16 shift value
111      */
112     public static int mul(int f1, int f2) {
113         int res;
114 
115         res = ((f1 >> 16) * (f2 >> 16)) << 16; // AH*BH
116         res += ((f1 >> 16) * (f2 & 0x0000FFFF)); // AH*BL
117         res += ((f1 & 0x0000FFFF) * (f2 >> 16)); // AL*BH
118         res += ((f1 & 0x0000FFFF) * (f2 & 0x0000FFFF)) >> 16; // AL*BL
119 
120         return res;
121     }
122 
123     /**
124      * Multiply two fixed-point numbers with defined shift both fixpoint values
125      * has too have been shifted equally
126      *
127      * @param f1
128      *            Fixpoint value
129      * @param f2
130      *            Fixpoint value
131      * @param SHIFT
132      *            The number of shifts the f1 and f2 value has been shifted
133      * @return f1 * f2 in a fixpoint SHIFT value
134      */
135     public static int mul(int f1, int f2, int SHIFT) {
136         int res = 0;
137         res = ((f1 >> SHIFT) * (f2 >> SHIFT)) << SHIFT; // AH*BH
138         res += ((f1 >> SHIFT) * (f2 - ((f2 >> SHIFT) << SHIFT))); // AH*BL
139         res += ((f1 - ((f1 >> SHIFT) << SHIFT)) * (f2 >> SHIFT)); // AL*BH
140         res += (((f1 - ((f1 >> SHIFT) << SHIFT)) >> 1)
141                 * ((f2 - ((f2 >> SHIFT) << SHIFT))) >> 1) >> (SHIFT - 2); // AL*BL
142 
143         return res;
144     }
145 
146     /**
147      * Divied two fixed-point numbers with shift 16
148      *
149      * @param f1
150      *            Fixpoint value
151      * @param f2
152      *            Fixpoint value
153      * @return The value of f1/f2
154      */
155     public static int div(int f1, int f2) {
156         f2 = (1 << 30) / ((f2 + 1) >> 2); // approx. 1/f2
157         return mul(f1, f2);
158     }
159 
160     /**
161      * Divides two fixed-point numbers with defined shift both numbers has to
162      * have the same shift the shift can't be lower then 3 for this function to
163      * work
164      *
165      * @param f1
166      *            Fixpoint value
167      * @param f2
168      *            Fixpoint value
169      * @param SHIFT
170      *            The number of shifts the f1 and f2 value has been shifted
171      * @return The value of f1/f2
172      */
173     public static int div(int f1, int f2, int SHIFT) {
174         f2 = (1 << ((SHIFT * 2) - 2)) / ((f2 + 1) >> 2); // approx. 1/f2
175         return mul(f1, f2, SHIFT);
176     }
177 
```

```
178     /**
179      * Returns the absolut value of a
180      *
181      * @param f1
182      *            Fixpoint value
183      * @return The Absoulute value of f1
184      */
185     public static int abs(int f1) {
186         return (f1 < 0) ? -f1 : f1;
187     }
188 
189     /**
190      * Calculates the max shift value for multiplying the number with it self
191      *
192      * @param f1
193      *            Fixpoint value
194      * @param SHIFT
195      *            The number of shifts the f1 value has been shifted
196      * @return The number of times a value needs to be shifted backwards to
197      *         enable shift with a equal value
198      */
199     public static int maxShift(int f1, int SHIFT) {
200         int shift = 0;
201         f1 = abs(f1);
202 
203         while (f1 > 1 << (31 - SHIFT)) {
204             shift++;
205             f1 >>= 1;
206         }
207         return shift;
208     }
209 
210     /**
211      * Returns the trigonometric Sinuous of an angle.
212      *
213      * @param f1Angle
214      *            Fixpoint angle, in radians
215      * @param SHIFT
216      *            The number of shifts the f1Angle value has been shifted
217      * @return The sinuous of the argument
218      */
219     public static int sin(int f1Angle, int SHIFT) {
220         int Temp1 = 0;
221         int Temp2 = 0;
222         int Rest = 0;
223         int b = 0;
224 
225         Rest = div(f1Angle * 180, PI >> (29 - SHIFT), SHIFT);
226         f1Angle = (Rest >> SHIFT) % 360;
227         if (f1Angle == 0)
228             Rest -= (360 << SHIFT);
229         else
230             Rest -= (f1Angle << SHIFT);
231         if (f1Angle < 0)
232             f1Angle += 360;
233         b = f1Angle + 1;
234 
235         // y = sine[ 180 - x]; /* 90 <= x <= 180 */
236         if (90 <= f1Angle && f1Angle <= 180)
237             Temp1 = (SIN[180 - f1Angle] >> (30 - SHIFT));
238         // y = -sine[x - 180]; /* 180 <= x <= 270 */
239         else if (180 <= f1Angle && f1Angle <= 270)
240             Temp1 = -(SIN[f1Angle - 180] >> (30 - SHIFT));
241         // y = -sine[360 - x]; /* 270 <= x <= 360 */
242         else if (270 <= f1Angle && f1Angle <= 360)
243             Temp1 = -(SIN[360 - f1Angle] >> (30 - SHIFT));
244         else
245             Temp1 = (SIN[f1Angle] >> (30 - SHIFT));
```

```
246
247         if (90 <= b && b <= 180)
248             Temp2 = (SIN[180 - b] >> (30 - SHIFT));
249         else if (180 <= b && b <= 270)
250             Temp2 = -(SIN[b - 180] >> (30 - SHIFT));
251         else if (270 <= b && b <= 360)
252             Temp2 = -(SIN[360 - b] >> (30 - SHIFT));
253         else
254             Temp2 = (SIN[b] >> (30 - SHIFT));
255
256         // y = sine[x] + (sine[x+1] - sine[x]) * delta_x (a-(a>>SHIFT)<<SHIFT))
257         return (Temp1 + mul(Temp2 - Temp1, Rest, SHIFT));
258     }
259
260     /**
261      * Returns the trigonometric Cosines of an angle.
262      *
263      * @param f1Angle
264      *            Fixpoint angle, in radians
265      * @param SHIFT
266      *            The number of shifts the f1Angle value has been shifted
267      * @return The Cosine of the argument
268      */
269     public static int cos(int f1Angle, int SHIFT) {
270         int Temp1 = 0;
271         int Temp2 = 0;
272         int Rest = 0;
273         int b = 0;
274
275         Rest = div(f1Angle * 180, PI >> (29 - SHIFT), SHIFT) + (90 << SHIFT);
276         f1Angle = (Rest >> SHIFT) % 360;
277         if (f1Angle == 0)
278             Rest -= (360 << SHIFT);
279         else if (Rest > (360 << SHIFT))
280             Rest -= ((f1Angle + 360) << SHIFT);
281         else
282             Rest -= ((f1Angle) << SHIFT);
283         if (f1Angle < 0)
284             f1Angle += 360;
285         b = f1Angle + 1;
286
287         // y = sine[ 180 - x]; /* 90 <= x <= 180 */
288         if (90 <= f1Angle && f1Angle <= 180)
289             Temp1 = (SIN[180 - f1Angle] >> (30 - SHIFT));
290         // y = -sine[x - 180]; /* 180 <= x <= 270 */
291         else if (180 <= f1Angle && f1Angle <= 270)
292             Temp1 = -(SIN[f1Angle - 180] >> (30 - SHIFT));
293         // y = -sine[360 - x]; /* 270 <= x <= 360 */
294         else if (270 <= f1Angle && f1Angle <= 360)
295             Temp1 = -(SIN[360 - f1Angle] >> (30 - SHIFT));
296         else
297             Temp1 = (SIN[f1Angle] >> (30 - SHIFT));
298
299         if (90 <= b && b <= 180)
300             Temp2 = (SIN[180 - b] >> (30 - SHIFT));
301         else if (180 <= b && b <= 270)
302             Temp2 = -(SIN[b - 180] >> (30 - SHIFT));
303         else if (270 <= b && b <= 360)
304             Temp2 = -(SIN[360 - b] >> (30 - SHIFT));
305         else
306             Temp2 = (SIN[b] >> (30 - SHIFT));
307
308         // y = sine[x] + (sine[x+1] - sine[x]) * delta_x (a-(a>>SHIFT)<<SHIFT))
309         return (Temp1 + mul(Temp2 - Temp1, Rest, SHIFT));
310     }
311
312     /**
313      * * (2) Knuth, Donald E., "The Art of Computer Programming Vol 1",
```

```
314      * Addison-Wesley Publishing Company, ISBN 0-201-03822-6 ( this comes from
315      * Knuth (2), section 1.2.3, exercise 25).
316      *
317      * http://www.dattalo.com/technical/theory/logs.html
318      *
319      * @param f1
320      *            Fixpoint value
321      * @param SHIFT
322      *            The number of shifts the f1 value has been shifted
323      * @return The natural log to f1
324      */
325     public static int ln(int f1, int SHIFT) {
326         int shift = 0;
327         // prescale so x is be between 1 and 2
328         while (f1 > (1 << (SHIFT + 1))) {
329             shift++;
330             f1 >>= 1;
331         }
332 
333         int g = 0;
334         int d;
335 
336         if (f1 != 0 || f1 != (1 << SHIFT)) {
337 
338             if (f1 < (1 << SHIFT)) {
339                 d = 1;
340                 for (int i = 1; i < SHIFT; i++) {
341                     if (f1 > d) {
342                         g = (log0Arr[SHIFT - i] >> (16 - SHIFT));
343                     }
344                     d = d << 1;
345                 }
346             } else {
347                 d = HALF >> (16 - SHIFT);
348                 for (int i = 1; i < SHIFT; i++) {
349                     if (f1 > ((1 << SHIFT) + d)) {
350                         f1 = FixPoint.div(f1, ((1 << SHIFT) + d), SHIFT);
351                         g += (log2Arr[i - 1] >> (16 - SHIFT)); // log2arr[i-1]
352                         // =
353                         // log2(1+d);
354                     }
355                     d >>= 1;
356                 }
357             }
358         }
359         return g + (lnScale[shift] >> (16 - SHIFT));
360     }
361 
362     /**
363      * Used for getting a other log base the the natural
364      *
365      * @param f1
366      *            Fixpoint value
367      * @param base
368      *            Base Value for the Log base
369      * @param SHIFT
370      *            The number of shifts the f1 value has been shifted
371      * @return The Log value of the f1 value with log base
372      */
373     public static int logBase(int f1, int base, int SHIFT) {
374 
375         if (SHIFT == 0)
376             return (div(ln(f1 << 1, 1), ln(base << 1, 1), 1)) >> 1;
377         else if (16 - SHIFT > 0)
378             return (div(ln(f1, SHIFT), ln(base, SHIFT), SHIFT));
379         else
380             return div(ln(f1 >> (SHIFT - 16), 16), ln(base >> (SHIFT - 16), 16)) << (SHIFT - 16);
381     }
```

```
382 }
```

Listing B.25: FixPoint.java

B.3.26 JSutil

```
1  package jopspeech.utility;
2
3  /**
4   * Embedded Java Speech Recognition on JOP <br />
5   * Website: <a href="http://www.jopspeech.com">www.jopspeech.com</a>
6   *
7   * @author Mikael Lundsgaard and Jens Kristian Rasmussen
8   * @version 0.1.0
9   *
10  */
11 public class JSutil {
12
13     /**
14      * @param args
15      */
16     /**
17      * Normelize the values in the vector
18      *
19      * @param vector
20      *             a range of values that need to be normelized
21      * @param normValue
22      *             the normelize value
23      * @param length
24      *             the length of the vector
25      */
26     public static void norMalizeFix(int[] vector, int normValue, int length) {
27         int i, max;
28         // System.out.println("norm start");
29         max = 0;
30         for (i = 0; i < length; i++) {
31             vector[i] = vector[i] << FixPoint.SHIFT;
32             if (max < FixPoint.abs(vector[i]))
33                 max = FixPoint.abs(vector[i]);
34         }
35
36         max = FixPoint.div(normValue, max, FixPoint.SHIFT);
37         // System.out.print(normValue);
38         // System.out.print(" ");
39
40         for (i = 0; i < length; i++)
41             vector[i] = FixPoint.mul(max, vector[i], FixPoint.SHIFT);
42     }
43
44 }
```

Listing B.26: JSutil.java

B.3.27 Protocol

```
1  package jopspeech.utility;
2
3  /**
4   * Embedded Java Speech Recognition on JOP <br />
5   * Website: <a href="http://www.jopspeech.com">www.jopspeech.com</a>
6   *
7   * @author Mikael Lundsgaard and Jens Kristian Rasmussen
8   * @version 0.1.0
9   *
10  * This class is for comunication between the PC and JOP the different "Code" is
```

```
11  * used to say what JOP or the PC needs to do
12  *
13  */
14 public class Protocol {
15     public static final int MAININIT = 0;
16
17     public static final int INIT = 1;
18
19     public static final int LOAD = 2;
20
21     public static final int ANALYSE = 3;
22
23     public static final int START = 4;
24
25     public static final int COMPLETE = 5;
26
27     public static final int TEST = 6;
28
29     public static final int CLASS = 7;
30
31     public static final int AUDIO = 8;
32
33     public static final int PLAY = 9;
34
35     public static final int RECORD = 10;
36
37     public static final int STOP = 11;
38
39     public static final int TESTWORD = 1110;
40
41     public static final int GETWORD = 12;
42
43     public static final int RESULT = 13;
44
45     public static final int ERROR = 10000;
46
47     public static final int PING = 100;
48
49     public static final int MFCC = 1120;
50
51     public static final int TRAIN = 1130;
52 }
```

Listing B.27: Protocol.java

B.4 The Class Diagram

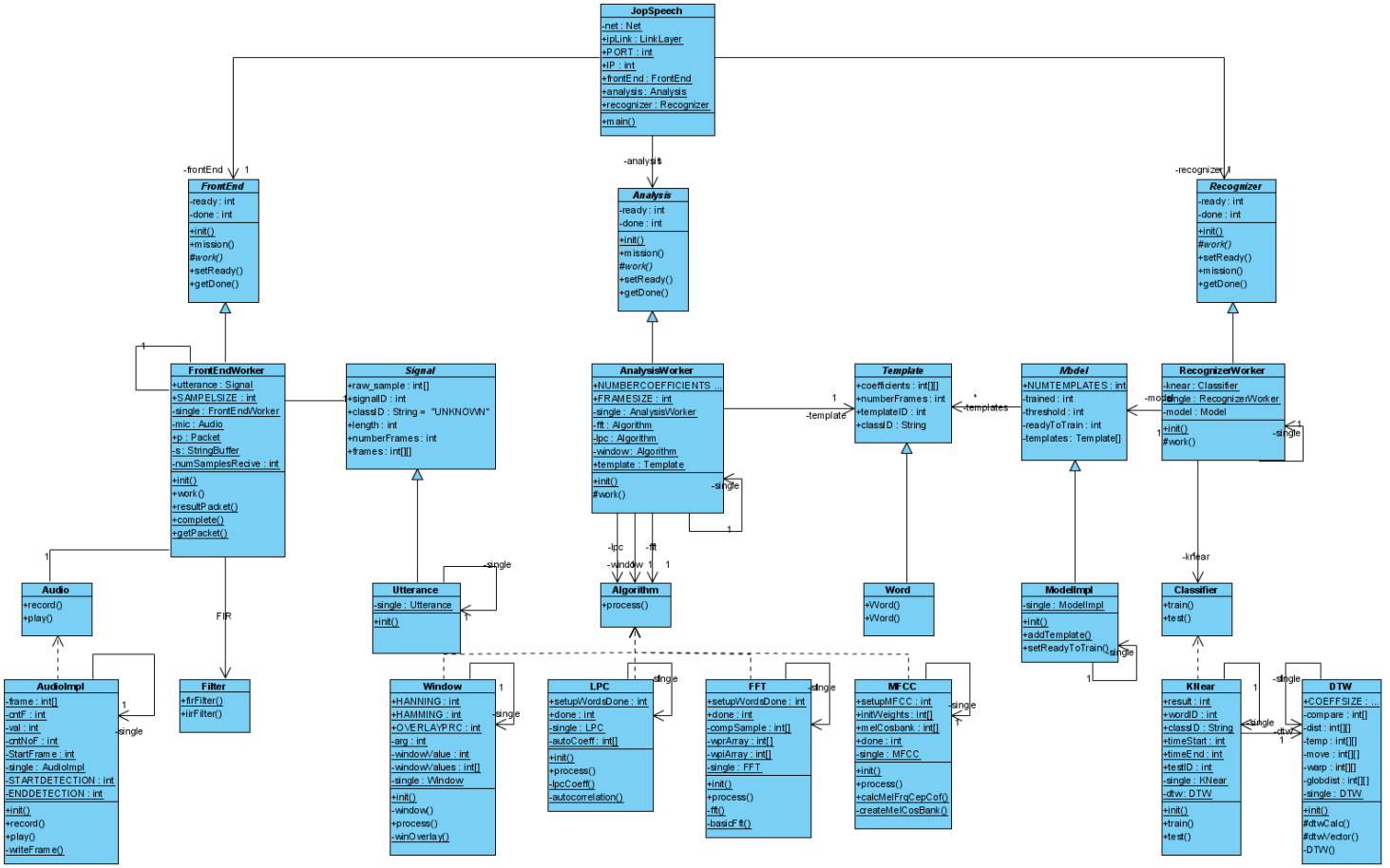


Figure B.2: The Class Diagram

B.5 Hardware

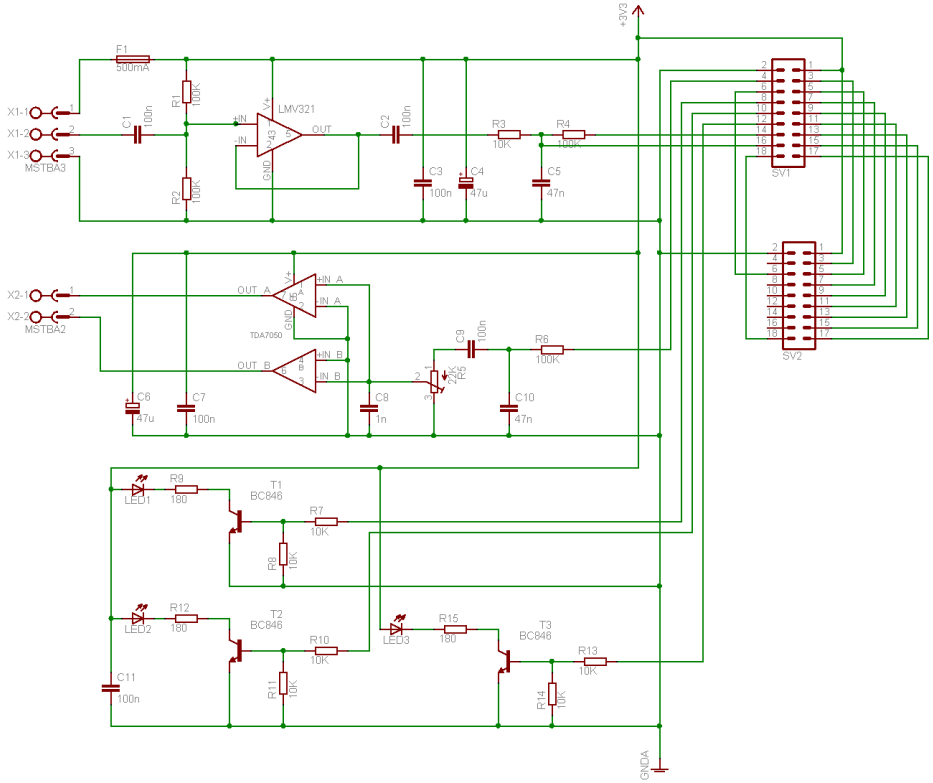


Figure B.3: Circuit diagram of a ADC, DAC with amplifier, and three led lights