UNIVERSITY OF SOUTHAMPTON
Faculty of Physical Sciences and Engineering
Department of Electronics and Computer Science

A project report submitted for the award of
Electronic Engineering with M.S.S.

Supervisor: Dr Steve Gunn
Examiner: Dr Nick Harris

Speech Recognition on Embedded Hardware

by Ricardo da Silva

April, 2013

UNIVERSITY OF SOUTHAMPTON

ABSTRACT

FACULTY OF PHYSICAL SCIENCES AND ENGINEERING
DEPARTMENT OF ELECTRONICS AND COMPUTER SCIENCE

A project report submitted for the award of Electronic Engineering with M.S.S.

by Ricardo da Silva

This report presents a proof of concept system that is aimed at investigating methods of performing speech recognition in embedded systems. It makes use of two new electronic boards that are currently under development at the University of Southampton, and implements one part of a modern speech recognition system using a Spartan 3 FPGA and an ARMv5 Linux applications processor.

Contents

Acknowledgements

Thanks to Steve Gunn, Srinandan Dasmahapatra

Nomenclature and Abbreviations

HMM Hidden Markov Model

LTIB Linux Target Image Builder

Chapter 1

Introduction

1.1 Goals

At the highest level, the primary goal of this project is to implement part of a modern Hidden Markov Model (HMM) based speech recognition system, with the constraint that it must be done on embedded hardware. In pursuing this goal, the aim is to achieve several other goals that will be beneficial to the author and to the University of Southampton.

1.1.1 Speech Recognition

Most modern HMM based speech recognition systems are extremely complex, and can take years to design and optimise for a particular implementation. The goal of this project is not to implement such a system, but rather to explore the possibilities of what may be achieved with a low power applications processor and a relatively small FPGA. In particular, the goal is to use the applications processor (running Linux) to perform pre-processing of speech data, and use the FPGA to perform mathematical manipulations of the observations. It is hoped that the system developed here may be later used as a basis for future research into the subject.

1.1.2 The Micro Arcana

The Micro Arcana is a new hardware platform aimed at undergraduate students, currently under development by Dr Steve Gunn. In terms of hardware, one of the project goals is to further development of the Micro Arcana family of boards, and provide a valuable example of how they may be usefully combined. The aim is to build the entire speech recognition system on two of the Micro Arcana boards, making it a self-contained embedded design. In addition, part of the project is setting up and configuring these two boards, so that they may be easily picked up by undergraduates.

1.1.3 Theoretical understanding

A major goal of the project is to develop a higher level understanding of the algorithms used in speech recognition, and to get experience designing a large-scale embedded application. This complements the interests of the author and the subjects being studied, in particular, Intelligent Algorithms and Digital Systems Design.

1.2 Motivation

Speech recognition is an interesting computational problem, for which there is no fool-proof solution at this time. Recently the industry for embedded devices and small-scale digital systems has expanded greatly, but in general these devices do not have the power or speed to run speech recognition. Field Programmable Gate Arrays (FPGAs) may present a way of increasing the capability of such systems, as they are able to perform calculations much faster than traditional microprocessors. The author's personal interest in embedded systems, combined with the challenges of a complex system such as speech recognition, makes this an appealing area to explore.

As hardware platforms go, most of the Micro Arcana is still very new and untested, as it is still under development. In addition, there are not many examples of how they may be used, and very little documentation. In order to improve their reception by students, it would greatly help to have proven use cases and examples of how these boards may be used individually and together. Using a larger FPGA

(such as an Altera DE board) was considered during the planning stage of this project, but it was decided that it would be more beneficial and interesting to develop and use the Micro Arcana.

1.3 Results and Personal Contribution

The project implements one part of a modern speech recognition system, using two development boards from the Micro Arcana family. It is designed to be a proof of concept exercise, in order to explore the capabilities of the boards, and expand the author's knowledge of the relevant systems. Specifically, the project required substantial research into HMM based speech recognition systems, embedded Linux, and digital design. The resulting system, described in detail in Chapter 4, uses an FPGA to perform the most computationally expensive part of HMM based recognisers – scoring the states of each HMM model for a given input vector. Essentially, the ARM based "L'Imperatrice" is used as the application controller, and is connected to the FPGA based "La Papessa" board which performs the CPU intensive mathematical calculations. The embedded Linux processor reads WAV format speech files, performs the necessary pre-processing and sends observation vectors to the FPGA. Given an observation, the FPGA will process it and send back scores for each state in the speech model, which represent the probability of that state creating the observation. Given these scores, the next step for a speech recogniser would be to perform Viterbi decoding (possibly using a token-passing or similar algorithm) in order to find the most probable sequence of HMMs, and thus eventually find the most probable word or phoneme sequence spoken.

Chapter 2

Background

2.1 Speech Recognition Systems

In general, 'Speech Recognition' refers to the process of translating spoken words or phrases into a form that can be understood by an electronic system, which usually means using mathematical models and methods to process and then decode the sound signal. Translating a speech waveform into this form typically requires three main steps [8]. The raw waveform must be converted into an 'observation vector', which is a representative set of data that is compatible with the chosen speech model. This data is then sent through a decoder, which attempts to recognise which words or sub-word units were spoken. These are then sent through a language model, which imposes rules on what combinations of words of syntax are allowed. This project focusses on implementing pre-processing, and the first stage of the decoder, as these are interesting tasks from an electronic engineering point of view.

There are a variety of different methods and models that have been used to perform speech recognition. An overview of the most popular will be described here, and then the chosen technique (HMMs) is described in Section 2.2.

2.1.1 Tor's Algorithm

The author first became interested in speech recognition when reading about "Tor's Algorithm", which is a very simple small dictionary speech recognition system [3]. This algorithm is capable of very accurate speaker dependent speech recognition

for a dictionary of about ten words. It is based on a fingerprinting model where each word in the dictionary must be trained to form an acoustic 'fingerprint'. This fingerprint is based on the time variations of the speech signal after being filtered appropriately. Then recognition is reduced to finding the Euclidean distance squared between the input vector and each of the stored fingerprints. The 'best' match is the word with the smallest distance from the input. Although this system is very simplistic, it outlines two major components of any speech recognition system – pre-processing and decoding (recognition). More complex systems essentially just use more complex speech models and pre-processing methods.

2.1.2 Dynamic Time Warping

Speech, by nature, is not constrained to be at a certain speed – the duration of words will vary between utterances, and a speech recognition system should be able to handle this. Dynamic Time Warping (DTW) is essentially the process of expanding and contracting the time axis, so that waveforms may be compared, independent of talking speed. Combined with a dynamic programming technique for finding the optimal 'warp' amount, it became a widely used approach to solving the problem of speech duration modelling [6]. One useful property of DTW is that it may offer good performance even with little training, as it only needs one word as a template [8]. Conversly, the performance of DTW based systems cannot be increased much with more training, unlike Hidden Markov Models.

2.2 Hidden Markov Models

By far the most prevalent and successful approach to modern speech recognition uses Hidden Markov Models (HMMs) for the statistical modelling and decoding of speech [5]. The flexibility inherent in HMMs is key to their success, as a system can be made more and more accurate by simply improving the HMM models or training the models further. The classic tutorial paper by Rabiner ([10]) is one of the best references for HMMs in speech recognition, and provides a very good overview of modern systems. However, a brief summary of the fundamentals of HMMs is given here. The following sections are based heavily on [10] and [14].

An N-state Markov Chain can be described as a finite state machine of N nodes with an $N \times N$ matrix of probabilities which define the transitions between each

state. According to the notation in [10], the elements of this matrix are defined as $a_{ij} = P(\text{state at time } t = j | \text{state at time } t - 1 = i)$. To make this a 'Hidden' Markov Model, each state is assigned an emission probability for every possible observation, which defines how likely that state will emit that observation. In this case, the actual position in the state machine is unknown – only the state emissions (thus 'Hidden Markov Model'). The probability that a state j will emit observation O is defined as $b_j(O)$, and may be either a discrete value or a continuous distribution depending on the nature of the observations. Thus, an HMM is defined entirely by the matrices a and b, and a set of initial probabilities for each state, π, collectively denoted as $\lambda(\pi, a, b)$.

For speech recognition, the performance is substantially improved by using continuous HMMs, as it removes the need to quantise the speech data which is, by nature, continuous. !!TODOcite!! A common distribution used for continuous probabilities is the multivariate Gaussian Mixture, which is essentially a weighted summation of several different Normal distributions [4]. However, for use in HMMs, the computational complexity is greatly reduced if the covariance matrix is diagonal (ie the components of each gaussian are uncorrelated). This requirement can lead to requiring extra pre-processing of observation data in order to remove correlation between the components.

2.2.1 Levels of Complexity

The simplest HMM based systems use a single HMM for every word in the recognition dictionary. Given a set of observations, each HMM can be scored based on the probability that it would output the observations. The HMM with the highest score is taken as the recognised word. The most apparent limitation of this system is that a very large amount of training would be required if a dictionary of substantial size was to be used. At the very least, one sample of each word would need to be recorded to train the full system, which would be a very time consuming process. However, for simple applications (voice dialling, for example) this is manageable.

The next step up in complexity from single word HMMs is models that consider sub-word utterances (phonemes). This allows a smaller set of HMMs to be used for much larger dictionary recognition, as words are recognised based on sequences of sub-word HMMs. Thus instead of searching through a single HMM to recognise a word, the recognition process becomes a search through a trellis of multiple HMMs

in order to find the best path through them. The most simple HMM system of this form is based on mono-phones, of which there are about 50 in the English language.

Even more complexity (and, potentially, recognition accuracy) can be introduced by using bi- or tri-phone HMMs, which model transitions between two or three mono-phones. Using this form of HMM will increase the acoustic model size greatly however, as there are many possible combinations of mono-phones in the English language. However, it allows context dependent scoring of phonemes, including HMMs that model word endings and starts, or silences. In the Sphinx 3 recognition engine, the internal states of these HMMs are referred to as 'Senones', and the term has been adopted and used extensively in this project [12].

2.3 Speech Pre-Processing

Speech signals are complex waveforms and cannot be processed without some form of feature extraction which reduces the complexity whilst retaining the important features. In modern speech recognition systems the two most common methods of analysing and representing speech are: [7]

- Linear Predictive Coding (LPC)
- Mel-Frequency Cepstrum Coefficients (MFCC)

Both these methods attempt to model the movement and dynamics of the human vocal tract and auditory perception. LPC is more suited to speaker recognition (the process of identifying voices, rather than speech), whilst MFCCs are more useful for speech recognition [1].

The Mel-Frequency Cepstrum is based on a filterbank analysis with a cepstral transformation, which is required due to the high correlation between filterbank amplitudes. The human ear perceives sound on a non-linear frequency scale, and one way of improving recognition performance is by using a similar scale for analysis of speech. A filterbank analysis can be used to perform this discrimination between different frequencies, and the frequency bins are usually spaced using the Mel frequency scale. However, the filterbank amplitudes are highly correlated, which greatly increases the computational complexity of the HMM based recogniser as the covariance matrix will not be diagonal. In order to correct this, a

discrete cosine transform is taken on the log filterbank amplitudes, finally resulting in a set of Mel Frequency Cepstrum Coefficients. The HTK (2.4) defaults to using twelve MFCC filterbank bins. [14] [8]

In order to attain MFCCs, a sampling rate must be chosen such that enough data is gathered while allowing sufficient processing time. In addition, to perform Fourier transforms on the speech, the incoming signal must be windowed appropriately. The HTK has a set of default values for these parameters, which are assumed to be appropriate.

An improvement to both LPC and MFCCs is to compute time derivatives in the feature extraction process, which gives a better idea of how the signal changes over time. In addition, the log energy of each sample may also be computed to also boost recognition ability.

2.4 The HTK and VoxForge

The Hidden Markov Model Toolkit (HTK) is a set of tools and libraries for developing and testing Hidden Markov Models (HMMs), primarily for speech processing and recognition tools [14]. Given a model structure and a set of transcribed speech recordings (a 'speech corpus'), a set of HMMs may be trained using the HTK. This includes performing all pre-processing in a number of formats, and testing recognition capabilities of a model.

Voxforge is an open source speech corpus which is aimed at facilitating speech recognition development. It provides pre-compiled acoustic models – essentially large sets of HMMs – in the format created by HTK, licensed under the GPL (GNU General Public License) [2]. The alternative would be to use another speech corpus (such as TIMIT), and then use the HTK to design and train the acoustic model. This is potentially a very time consuming process, so Voxforge is useful because it essentially cuts this step out. In addition, the Voxforge models may be easily adapted to a specific person's voice using only a few minutes of transcribed speech. The Voxforge model is very complex, with 8000 tri-phone context-dependent HMMs with multivariate Gaussian output probabilities. Thus, implementing a recogniser system based on this model requires a lot more work than if a simpler model was used, such as one based on discrete (output probability) HMMs. However, modern speech recognisers are likely to use a model that is as complex, if not more so.

2.5 Embedded Hardware and Speech Silicon

A wide range of speech recognition software (commercial and open source) exists for desktop PCs or laptops. However, speech recognition for embedded systems is less widespread. Recently there has been increased research into the use of DSPs and FPGAs for speech recognition [8], [11], [9]. Of particular interest is Stephen Melnikoff's PhD Thesis, and the Speech Silicon architecture. The former investigates a variety of HMM based recognisers on an FPGA, using a PC to perform pre and post processing. The latter details a flexible FPGA based system capable of performing recognition on medium-sized vocabularies.

Both Melnikoff and Speech Silicon perform an in-depth analysis of an entire speech recognition system based on programmable logic. As such, both of them require relatively large FPGAs

Chapter 3

Design Theory and Approach

This chapter provides details of the relevant theory behind HMM based speech recognition systems, as well as a description of the development environements used during the project.

3.1 The HMM based model

Due to the flexibility of HMMs, and the complexity of speech, there have been several different approaches to building speech models (the sheer size of the HTK book indicates how much flexibility exists). However, at this stage the author is more interested in the implementation of the algorithms, rather than devising the best way of modelling speech. Therefore, it was decided to use the pre-designed models from Voxforge for this project, and build the hardware to work with these models. Thus, various parameters were fixed from the start, including:

- Sampling rate of audio: 8kHz.
- Window size: 25ms (duration of observation frames).
- Frame period: 10ms (time between observation frames).
- Pre-processing output: 12 MFCCs, 12 MFCC derivatives, 1 Energy measure.
- Output probabilities of HMM states: Single Gaussian distribution, 25-element mean and variance vectors.

- Number of monophones: 51 (Includes a monophone for silence. This is also the number of transition matrices).
- Number of senones: 7000.
- Number of HMMs: 8300[1] (each with 3 outputting states).

The only modification made to the Voxforge models was that they were adapted for the author's voice, primarily to gain confidence with using the HTK and HMMs. Please see Appendix !!TODO!! for the scripts and HTK configuration files used to generate these models.

The term 'outputting states' refers to states that produce an observation – most of the HMMs have 5 states in total, but the first and last are non-emitting. The transition probability from the fourth to last state is the probability of exiting that particular HMM, which is useful for decoding purposes. The senones are context dependent, that is, there are many different senones for each monophone, each with different predecessor and successor monophones. The number of transition matrices is equal to the number of monophones because...

3.1.1 The HMM tasks

For an HMM model, denoted as λ, there are usually three important problems:

- Design and train the model to accurately represent real data
- Finding the probability that an HMM produced a given observation sequence, $P(\lambda|O)$.
- Finding the 'best' path through a trellis of HMMs to produce a given observation sequence !!TODOequation!!

For this project, the first problem is solved by using Voxforge (2.4). The second problem is potentially very computationally expensive, as the speech model may be complex or large. In particular, this step requires scoring the senones of each HMM for every new observation frame, which is particularly time consuming if the HMMs have continuous output distributions. !!TODO: Speech silicon has refs to confirm this is most expensive!! This is the step that the project focusses on.

[1] There are more HMMs than senones because some senones are used in more than one HMM

In all literature encountered, the Viterbi algorithm is the preferred method for solving problem 3. It is an iterative approach to solving the optimisation problem, and has the added bonus that not much data needs to be stored during the calculation [11]. This problem is beyond the scope of the current project, but a full explanation of the Viterbi decoding process is available from [10],[8],[13].

3.1.2 Senone scoring

As described in previous sections, the FPGA is intended to be used for scoring every senone in the model, for every observation vector. In this system the new vectors arrive once every 10ms, and there are about 7000 senones that must be evaluated. The mathematical operations required to do this are now outlined.

Each senone j has an N-element vector of means, μ_j, and a $N \times N$ matrix of covariances, σ_j. However, since MFCCs are uncorrelated, the covariance matrix is diagonal, and σ_j is taken as an N-element vector. If the observation vector at time t is denoted as $\mathbf{O}_t = O_{t1}, O_{t2}, ..., O_{tN}$, then the score of senone j is given in Equation 3.1. However, the hardware complexity may be greatly reduced by taking the logarithms of both sides, removing the requirement to evaluate N exponentials for each senone. In addition, several parts of the equation may be precomputed, thus reducing the necessary sequence of operations to subtract, square, multiply, and accumulate. This derivation is shown in Equations 3.2–3.3, with the final result being the one most suited to hardware implementation.

$$b_j(O_t) = N(O_t; \mu_j, \sigma_j) = product... \tag{3.1}$$

$$lnofboth \tag{3.2}$$

$$final \tag{3.3}$$

3.2 Hardware environment

As the Micro Arcana is still under active development, part of the project involved setting up and testing the two boards that the project used.

3.2.1 L'Imperatrice

The ARM-based L'Imperatrice board is still under active development, and several important features on it are very untested. It is based on a Freescale iMX23 ARMv5 applications processor. To be used for the project, the following items were required (in order of importance):

- Native or cross compiler set-up
- Application UART functionality
- GPIO functionality

An Linux Target Image Builder (LTIB) environment, which is primarily used for setting up board support packages (BSP), was installed on an Ubuntu virtual machine. It has been used to build and test various kernel configurations, and also includes full cross-compiler support for the board. It essentially provides a platform on which software for L'Imperatrice may be developed and deployed. In addition to LTIB, the ArchLinux build system (ABS) was investigated as a potential alternative to LTIB. The primary advantage of the ABS is that only a small number of files need to be distributed, which, when run, will download and compile all dependencies of the build. An ABS configuration exists for the Olinuxino, a linux board also based on the Freescale iMX23, which may be tweaked to suit the L'Imperatrice. However, due to lack of time and the relative ease of the LTIB set-up, this was not done.

3.2.2 La Papessa

The Xilinx FPGA-based La Papessa board is also actively being developed, and some of its features have not been tested. In order to facilitate the development of code on the La Papessa board, several combinations of software environments were explored. The FPGA is a Xilinx Spartan XC3S50AN, which is compatible with the Xilinx ISE Webpack design software package. However, one drawback to the ISE Webpack is its lack of support for synthesis in SystemVerilog. Besides being syntatically more powerful, SystemVerilog is the HDL that is currently taught to all new undergraduates at the University of Southampton. Having some documentation of a proven way to use SystemVerilog with this board would improve its reception and usage. In addition, SystemVerilog has advantages over Verilog for verification and simulation, which will be used to improve the design.

Synplify Pro/Premier is an alternative HDL synthesis tool, which is compatible with the Xilinx software toolchain and also supports SystemVerilog. For this reason, and because the author is more familiar with the Synplify design flow, it was decided that Synplify Premier would be used for synthesis during the project. The other design tasks (port mapping, programming file generation) are accomplished with ISE Webpack (See Appendix !!! for detailed description of this process).

Chapter 4

Design Detail

4.1 System Overview

The hardware related goals outlined in Chapters 1 and 2 can be summarised as:

1. Design a system in programmable logic that can efficiently evaluate Equation 3.3.
2. Design a C program to pre-process speech data according to the required form described in Section 3.1.

The overall system layout, shown in Figure 4.1, is comprised of two primary blocks – L'Imperatrice and La Papessa (referred to from now on as "The processor" and "The FPGA" respectively). The entire system is powered from a single supply connected to the processor board's battery connector, and the FPGA is powered through ribbon cable connecting the two boards. This was done in order to minimise the amount of external circuitry needed, and to show that the two devices are able to work together fairly easily.

From the list above, the first task is implemented on the FPGA, and the second on the processor. These two blocks will now be examined in greater detail.

4.2 Number Format

Before beginning implementation of any part of the project, a number representation which would be appropriate for the FPGA had to be decided upon. Firstly,

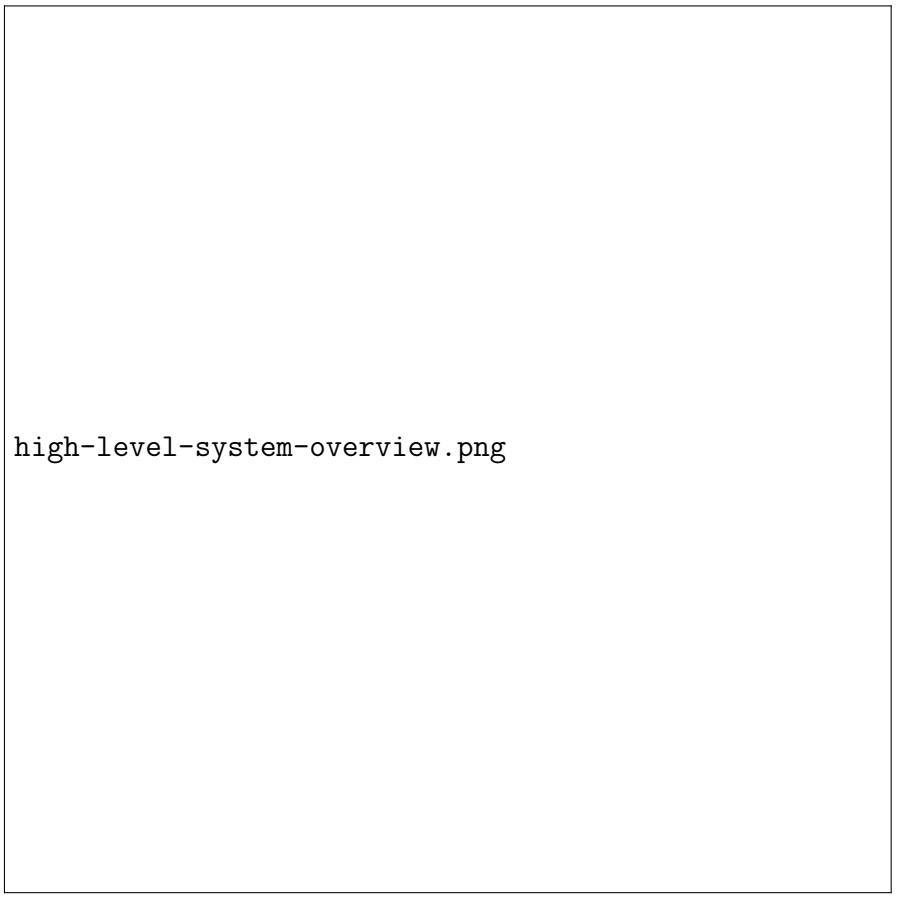

FIGURE 4.1: High level system layout overview

it was recognised that using floating-point arithmetic on the FPGA would not be worth the effort, and therefore some form of fixed-point system was needed. Further, the number magnitudes vary greatly between stages and parameters in the system. In order to solve this problem, different scaling factors were used to being most of the parameters to a similar magnitude.

The inputs and outputs of the Gaussian Distance Pipe, the module that performs the core calculation, are all signed 16-bit fixed point numbers, with varying scaling factors. In particular, the k parameter was generally larger than x, $mean$, and $omega$, and thus was scaled down. The scaling factors were decided primarily by analysing the HMM models, to determine the largest and smallest numbers used.

These decisions were influenced by [8], and the HTK, as they both use scaled 16-bit numbers to represent the parameters and scores.

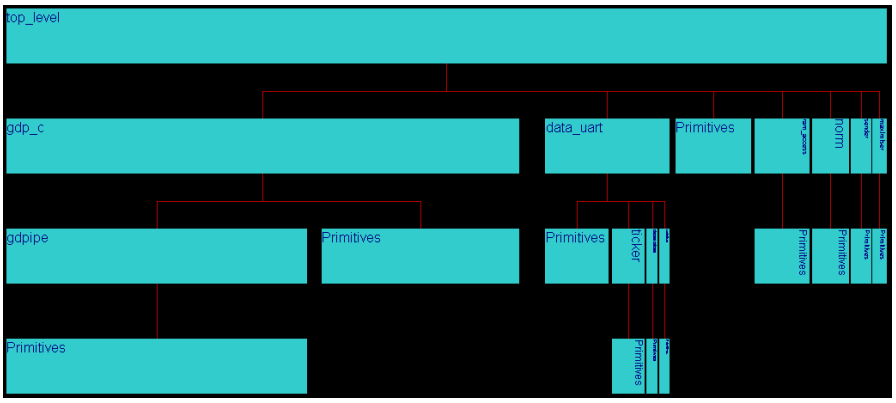

FIGURE 4.2: Hierarchical component diagram of Top Level Module (Section 4.3.1)

4.3 La Papessa (The FPGA)

The system on La Papessa performs these main tasks:

- Receives observation vectors from the processor.
- Calculates Equation 3.3 for every senone in the model, with the given observation.
- Normalises the senone scores.
- Sends the scores back to the processor.

4.3.1 Top Level Module

A simplified hierarchical diagram of the top level module is given in Figure 4.2, showing the main components of the system. This module included the main controller logic, which essentially waited for a new observation vector, then cycled through the necessary operation states. Figure 4.3 shows an ASM of this logic, and also outlines the main areas of the design that need explanation.

The top level module is also responsible for handling access to the on-board SRAM chip, which several modules need to write or read from. It essentially multiplexes the required signals, and leaves them floating (high impedance) when they are not needed. The 'Debug signals' shown in Figure 4.2 are a number of internal signals that are routed to output ports in order to facilitate hardware debugging.

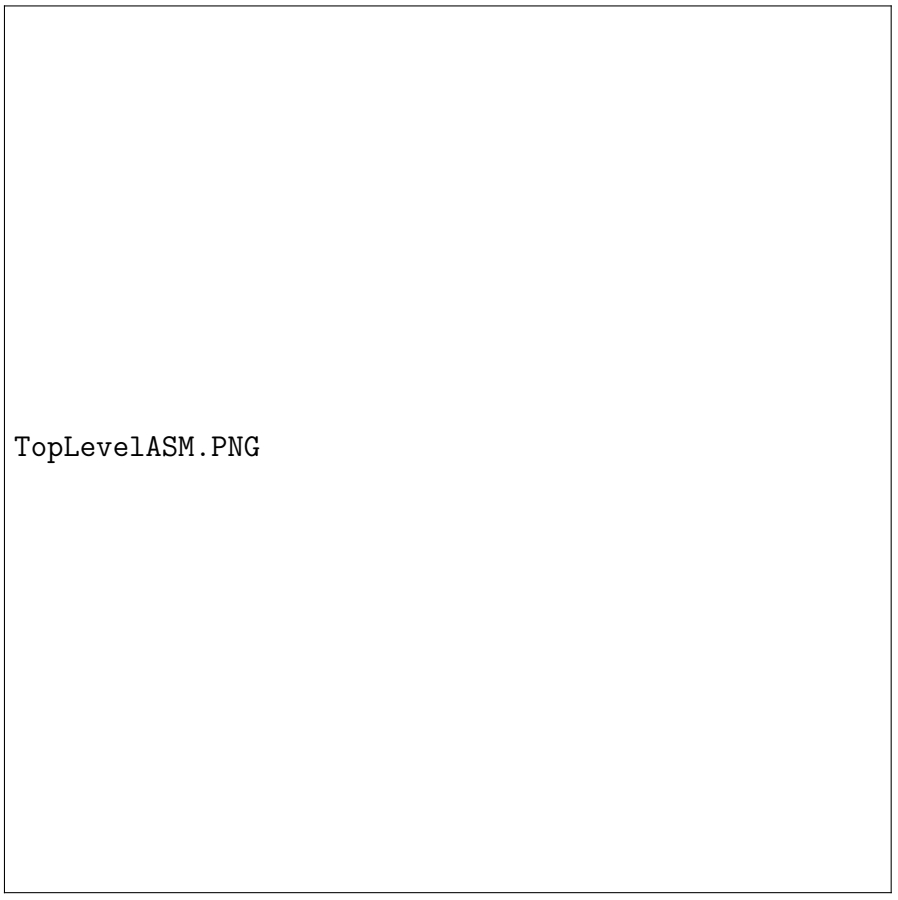

FIGURE 4.3: ASM diagram of Top Level Module (Section 4.3.1)

4.3.2 Gaussian Distance Pipeline

The Gaussian Distance Pipeline (GDP) is the core component of the system, and computes Equation 3.3. It is a relatively simple 4-stage pipeline, with one stage for every step in the equation (subtract, square, scale, accumulate). Although the gains from using a pipeline in this case are relatively small, it would be very useful if more complex models were used. The Speech Silicon [11] project had a substantially more complex GDP, as their senones have several Gaussian distributions that must be mixed to produce the final output distribution. A block diagram of the GDP module is shown in Figure 4.4.

'`n_senones`' and '`n_components`' are both parameter inputs to the module, which determine the number of senones and the number of components per mean and variance in the model. This allowed the design to be scaled down as size constraints became restrictive.

The pipeline itself is a static object with not much control logic, and thus requires

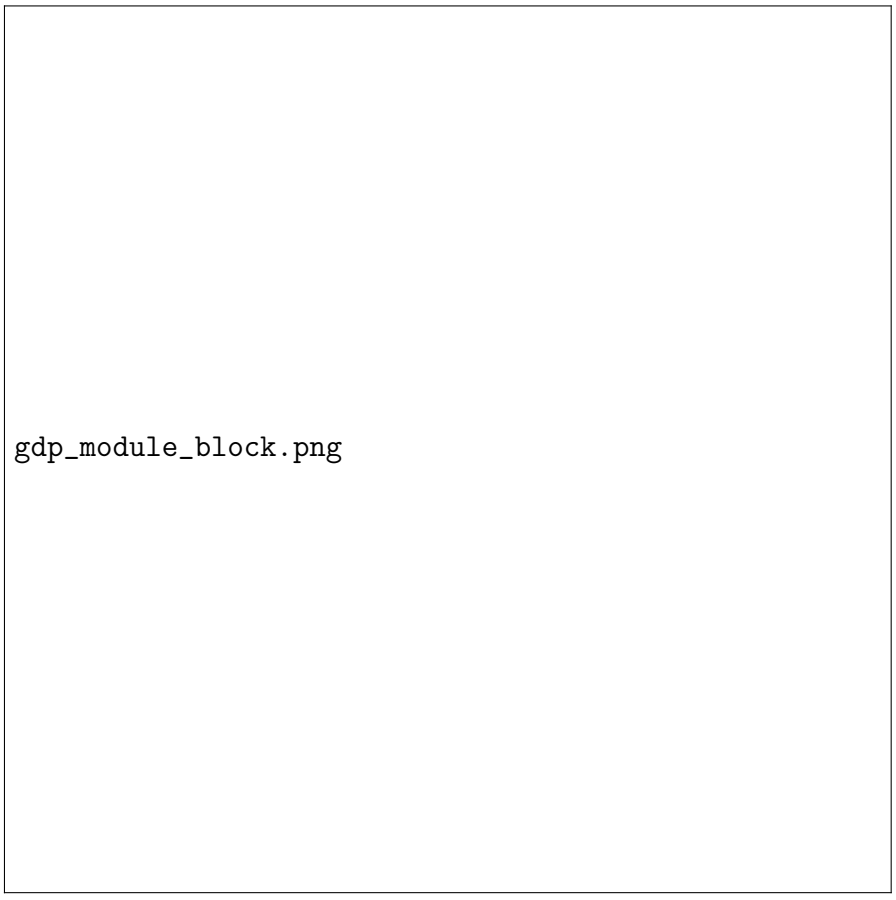

FIGURE 4.4: Gaussian Distance Pipeline block diagram

a controller which sequentially uses it to score each senone in the model. This controller is a simple state machine of only two states (`IDLE` and `LOAD_GDP`), which begins feeding the pipe when a '`new_vector_available`' input flag is asserted. This module is shown in Figure 4.5. A '`last_senone`' output flag is asserted when the GDP has finished processing the last senone. When the controller is in the `LOAD_GDP` state, it essentially loops through the senones in the model, extracting their parameters and sending them to the GDP.

In order to facilitate the extraction and manipulation of senone parameters, a SystemVerilog structure was created, shown in Listing 4.1 below. A SystemVerilog ROM module, connected to the controller, was populated with the senone parameters, stored in this structure. Thus, when '`new_vector_available`' is asserted, the controller simply counts from 0 up to `n_senones` (the number of senones in the model), and pulls the required parameters out of the ROM.

```
typedef struct packed {
  num k;
  num [n_components-1:0] omegas;
  num [n_components-1:0] means;
```

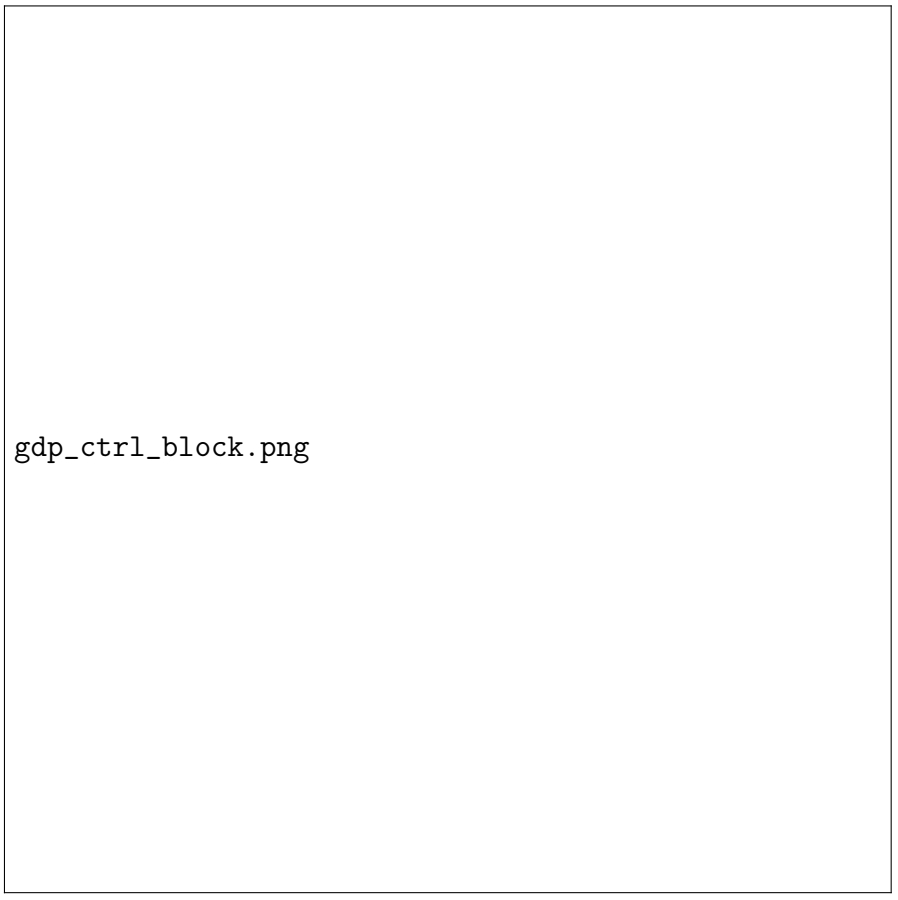

FIGURE 4.5: Gaussian Distance pip Controller block diagram

```
} senone_data;
```

LISTING 4.1: Senone parameter data structure

After asserting `new_vector_available`, and providing the revelant vector on the '`x`' input, the top level module will see senone scores being sequentially produced, along with an index or ID which is unique to each senone. The Top Level module is then responsible for storing the score in SRAM, and a 'Maximiser' module registers the highest score seen.

4.3.3 SRAM

In order for the normaliser to access the senone scores, they must be stored somewhere as they are processed through the GDP. One option would be to store them on the FPGA itself, where they would be accessible by all the modules. However, this is impractical due to the potential size of an HMM model, and thus the large

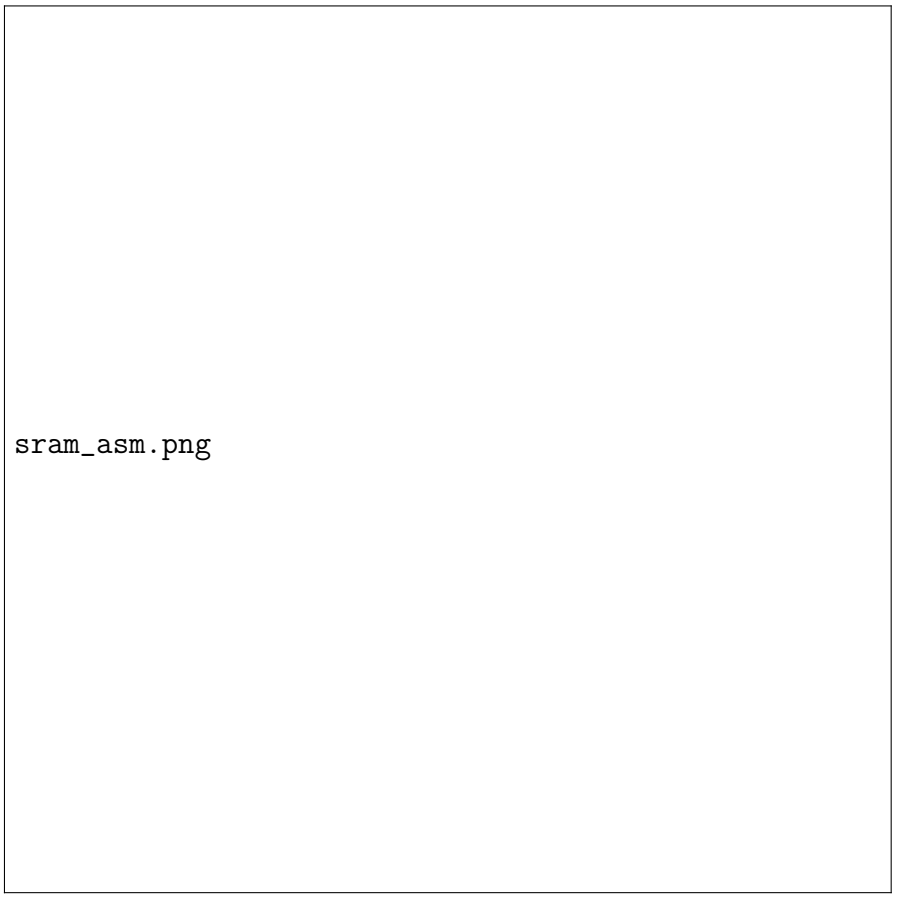

FIGURE 4.6: ASM of the SRAM access module (4.3.3)

amount of RAM that would be required. A better alternative would be to use external SRAM with low latency that is made accessible to whichever modules need it. In the future, this could also allow the full speech recognition decoder to be implemented on the FPGA, as it needs access to the normalised scores.

Revision C of La Papessa board includes an onboard SRAM chip, which has an 8 bit data bus and 21 bit address bus, and was completely untested before this work. As with the UART module, an SRAM module was created primarily to interface the 8 bit data bus with the 16 bit number system used. It's only operations are to write and read 16 bit values to the SRAM, and signal when it is ready or idle. This is accomplished with the state machine shown in Figure 4.6.

4.3.4 Communications

The primary method of communication between the FPGA and processor is a standard UART bus, running at 115200 baud. The communications module on

the FPGA is essentially comprised of standard UART receive/transmit modules and a module that is wrapped around them to provide a higher level of data abstraction. Because numbers are all 16 bits long in this system, and UART words are normally 8 bits, one of the wrapper functions is to receive and transmit 16 bit numbers. In addition, it is known beforehand that the FPGA should receive a certain number of bytes per observation vector. This allows the UART module to wait until a 'packet' with that number of bytes has arrived, before signalling to the main controller that a new vector has arrived. In this implementation, a buffer is simply filled up as new bytes arrive, and then is passed to the main controller when full. However, it may happen that the UART module erroneously receives a byte, causing the buffer to be one byte fuller than it should be, and therefore causing the last byte of a new observation to be dropped. In order to work around this, a '`new_vector_incoming`' flag is added, which will empty the buffer when asserted by the processor. Another (possibly better way) of avoiding errors would be to automatically empty the buffer if new data is not received for a timeout period.

The Baudticker module generates clock signals for the transmit and receive modules. The transmit module requires a signal at approximately 115200Hz, but the receive module requires a signal at 8 times that frequency. The receiver oversamples the input rx signal at this higher frequency, which enables it to detect and ignore signal glitches.

4.3.5 Normalising and Sending Scores

The Speech Silicon architecture included a module which normalised the senones before they were used for decoding. A very similar module is implemented here to perform the same normalisation. The highest score is found while senones are being evaluated, and then this score is subtracted from all the final scores. This causes the senone with the highest score to have a score of 0, which corresponds to a probablity of 1 (the scores are log probabilities, see Section 3.1.2).

The Sender module is fairly simple in its operation – it loops through all the senone scores stored in external SRAM and sends them over UART to the processor.

4.4 L'Imperatrice (The processor)

The processor performs these main tasks:

- Read speech data from a WAV file.
- Pre-process – pre-emphasise and window data, calculate FFT, calculate MFCCs, 'Lifter' MFCCs.
- Convert MFCCs to the correct binary format.
- Send observation vector.
- Receive and display scores.

4.4.1 Pre-processing

Instead of reading data in from a microphone, the system uses pre-recorded speech stored in WAV formatted audio files. This method was chosen because it allows the pre-processing to be very easily tested, without worrying that the input data was changing. In addition, it allowed the results to be compared with pre-processed data from other libraries, such as the HTK, in order to verify correct operation.

The audio files were prepared specially, in a format that is simple to read and use. The audio manipulation software 'Audacity'[1] was used to record and store speech as uncompressed (Microsoft) WAV files with unsigned 8-bit PCM encoding. As Audacity does not support saving files with 8kHz sampling rates, the 'sox'[2] utility was used to downsample them from 48kHz to 8kHz.

In pre-processing the data, an attempt was made to match the processes that the HTK used as closely as possible, thus making it easier to verify that the process works correctly. Samples from the audio files are read in sequential blocks of between 80 and 200 samples, depending on the window size required. They blocks are windowed with a Hamming window to remove discontinuities at the edges which could cause excess spectral noise. The DFT is taken of this data, using the FFTW library, which had to be specially packaged and compiled for use in LTIB (See Appendix !!TODO!!). Finally, the magnitude is taken, so that the spectral data is fully real and may be used to calculate Mel Frequency Cepstral

[1] Audacity is available at http://audacity.sourceforge.net/, last checked April 2013.
[2] SOX (Sound eXchange) is available at http://sox.sourceforge.net/, last checked April 2013.

Coefficients (MFCCs). An external library (LibMFCC) was used for this purpose. The last operation, shown in Equation 4.1, is to perform 'Liftering'[3] on each of the coefficients, where L is a parameter. This results in the cepstral coefficients having similar magnitudes [14], which is particularly convenient when they must be represented with a 16-bit fixed-point number.

$$c'_n = \left(1 + \frac{L}{2} sin \frac{\pi n}{L}\right) c_n \tag{4.1}$$

4.4.2 GPIO and Application UART

In order to communicate with the FPGA, the processor required access to a serial port. The ideal solution is the built-in application uart port on the iMX23 chip, which only needs configuring. In addition, access to GPIO from inside a C program was required to assert the 'new vector incoming' signal.

Both GPIO and Application UART must be enabled by selecting the relevant entry in the Kernel configuration menu at compile time (See Appendix !!TODO!!). Using the Application UART from a C program requires opening the serial port file (`/dev/ttySP1`, usually) and using the `termios.h` library to configure it correctly. The options used to configure the serial port are shown in Listing 4.2.

```
/* Set important serial port parameters: */
stty.c_cc[VMIN] = 0;                          // No blocking read
stty.c_cc[VTIME] = timeout;                   // Max time to wait for data
stty.c_cflag = (stty.c_cflag & ~CSIZE) | CS8; // 8 bit chars
stty.c_iflag &= ~IGNBRK;                      // Ignore break commands
stty.c_lflag = 0;
stty.c_oflag = 0;
stty.c_iflag &= ~(IXON | IXOFF | IXANY);      // No flow ctrl
stty.c_cflag |= (CLOCAL | CREAD);             // enable receiver
stty.c_cflag &= ~(PARENB | PARODD);           // No parity
stty.c_cflag &= ~CSTOPB;                      // Send 1 stop bit
stty.c_cflag &= ~CRTSCTS;
```

Listing 4.2: Serial Port Configuration

The GPIO is accessible in more than one way – either through sysfs or via direct register access. The kernel is configured to include the sysfs gpio interface, which creates an entry under `/sys/class/gpio` that allows GPIO pins to be written and read as standard files. Alternatively, the Olinuxino board project (which uses the same processor as L'Imperatrice) includes C code to directly write and read the

[3] The name 'Liftering' comes fromthe HTK book [14]

GPIO registers as mapped memory[4]. The project uses the second method, as it was straightforward to adapt the existing code for the purposes of the project.

4.5 Support Software

A set of utilities were written in Common Lisp in order to facilitate and accelerate various parts of the project development. In particular, they greatly helped with the following tasks

- Parsing HMM definitions created by HTK, and extracting senone parameters
- SystemVerilog testbench generation
- Senone data file generation
- Verifying hardware functionality
- Number format conversion tasks

The HTK stores HMM definitions (along with transition matrices and senone definitions) in a human-readable plain-text ascii file. However, due to the very large model size, it was impractical to attempt to copy out parameters by hand. For this reason, a parser was created that read an HMM definition file and stored the extracted data in a useful structure. From here, it was possible to automatically generate C header files and SystemVerilog modules containing the parameters, in a suitable format. In addition, it became possible to easily and quickly generate testbenches to test the GDP with many different senones.

Another set of utilities was created to aid converting between floating point numbers and the custom fixed-point representations used.

[4] Accessible on the Olinuxino Github repository at https://github.com/OLIMEX/OLINUXINO/

Chapter 5

System Testing and Results

The testing methods and results are documented in this chapter, along with an analysis of the final design. The two primary blocks, described in the last chapter, were tested independently and then together, and will be presented here in the same manner. A wide range of tools and techniques have been used to perform the testing, and to identify and fix errors encountered.

5.1 FPGA Design and Test methodology

There were several phases of development, and thus testing, of this part of the design. Initially, the full system was written and simulated in ModelSim, which supports SystemVerilog assertion based verification. Each of the components of the design was built and tested separately, and then incrementally joined together and simulated as a whole. Finally, the modules were tested in hardware – first individually, and then as a full system.

A set of testbenches were developed for use within ModelSim, which tested each of the modules used by the system. Most of the modules were fairly simple to simulate and verify, as the expected behaviour is generally constant and determinate. For example, verifying the UART clock generator was a case of instantiating it, asserting the enable signal, and verifying the frequency of the output signals. However, some modules and their testbenches deserve extra attention, as special methods were used to test them. In several areas, the benefits of using SystemVerilog become clear, as assertions greatly simplified the validation and testing process.

FIGURE 5.1: GDP testbench waveform

5.2 Gaussian Distance Calculation Block

The Gaussian Distance Pipe was initially tested by creating a testbench which provided the sequence of inputs that the GDP required, and then asserted that the correct score was provided. Figure 5.1 shows the correct operation of this module, with a set of test data being used as inputs to the module. The expected results were determined using the software GDP and the binary conversion software described in Section 4.5, thus confirming the validity of the results produced.

The Gaussian Distance pipe controller testbench gives a new observation to the controller, and then watches as senone scores are produced by the module. SystemVerilog assertions are used to check whether correct scores are arriving at the time they should. However, in order to test a large number of different inputs (which would result in different score outputs), a set of software utilities were written to automatically generate the testbench code. This software made use of the HMM definition parser and a software version of the GDP that had already

FIGURE 5.2: GDP Controller testbench waveform and error messages

been created. This allowed a very large number of senones to be tested in a matter of minutes, and determine how well the GDP pipe was working. Figure 5.2 shows the GDP Controller being tested, with an example of the type of error message used to examine its behaviour. In this test,

Being able to easily test many senones was important because the use of fixed-point arithmetic inevitably causes numerical errors that vary with the operations and numbers being used. The GDP may produce the correct result for one set of inputs, but another set of inputs may produce an erroneous result that was caused by the fixed-point number not having high enough precision. In fact, in most cases, the least significant bits of the result were usually wrong. Because of this, it was important to determine the distribution of the error magnitudes, in order to decide whether the number format needed changing. Automatic testbench generation made this far easier, as testbenches could be created which automatically displayed which senone scores where wrong, and by how much.

5.2.1 Synthesis and Hardware Testing

Unfortunately, due to the small size of the FPGA used, the initial synthesised design did not fit on it. The full Voxforge model had over 7000 senones, and each observation had 25 parameters of 2 bytes each – over 700kB of storage would be required for the full model. As this far exceeds the resources available, the model size was reduced for this proof-of-concept project. The number of statistical parameters was reduced to 6, and only 3 senones were scored at once, bringing the total resource usage to about 90%.

A variety of tools were used to repeatedly test the functionality of this hardware. A Bus Pirate[1] was used to communicate with the FPGA, which allows hexadecimal values to be easily sent and received over UART. In order to debug the SystemVerilog, a large number of signals were routed to pins on the FPGA, which were then monitored with a Saleae Logic analyser.

5.3 UART communications

Simulating and testing the UART module was done by creating two instances of the module, and cross-connecting their RX and TX pins. This allowed both transmit and receive to be tested, and confirmed correct operation. An example of this testbench running is shown in Figure 5.3.

After simulation, the UART module was finally verified by programming the FPGA with the UART module, and a simple controller which echoed back whatever it received. An FTDI USB to serial cable connected the FPGA to a computer, so that the setup could be confirmed to work with a real UART connection. In addition, a Saleae Logic analyser was used to examine and ensure the absence of glitches in the UART signals generated by the FPGA.

5.4 SRAM access

A custom SystemVerilog module was designed in order to test writing and reading values to the onboard SRAM chip, as it was a completely untested part of the

[1] Multi-purpose debugging tool that supports many different protocols. See http://dangerousprototypes.com/docs/Bus_Pirate

FIGURE 5.3: UART testbench waveform

board. The module performed fairly basic operations, such as looping through a set of values, writing them to SRAM, and then reading them out again. Debug information was displayed on the output pins, allowing the process to be monitored. This confirmed that the chip worked as expected, and would be usable for the project.

5.5 Pre-Processing

There were several different stages to the pre-processing, and these were tested individually. First and foremost, the FFT code was tested by using, as input data, a set of data containing known sinusoidals. The library used, FFTW, is well established and proven to work, and so the primary purpose here was to test the speed of the FFT. It ran almost instantaneously (execution took 0.00000s, according to the Kernel time.h library), giving the correct results.

The other major operation that required extensive testing was the MFCC calcu-

lation. The other operations, pre-emphasis, windowing and liftering, are fairly simple mathematical operations that are easy to verify. One of the desired properties of the system is that it would produce MFCCs that matched those produced by the HTK. Unfortunately, this was not quite acheived. In most tests, the energy coefficients matched, but the others were relatively different. This implies either that there is a limitation or problem with the MFCC library used, or that the HTK performs extra (or different) pre-processing.

To be used in a real system, an important requirement is that all the pre-processing is much faster than the frame period. With the current design, the only component breaking this requirement is the MFCC computation, due to the library's slowness and inefficiency. All the other steps take a tiny fraction of the frame period, allowing a large amount of time for processing, and in later designs, decoding.

5.6 Full system

Once all the separate modules were tested and verified, the final task was to identify and eliminate the inevitable errors that occur once a design is integrated.

Chapter 6

Project Analysis

6.1 Analysis of Solution

6.1.1 The FPGA

The FPGA used was very small, and the full required design could not fit on it. In particular, the size of the model had to be reduced, so that each Senone had fewer than 25 components, and not all 7000 senones were processed. This is mainly due to the small amount of onboard RAM on the FPGA, so the design would greatly benefit from having more RAM available.

6.1.2 The processor

6.2 Deviations From Initial Goals

6.3 Time Management

Chapter 7

Conclusions

This report presented a proof of concept system for performing speech recognition related operations on embedded hardware. The system was built on two boards from the Micro Arcana family of development boards, and performed speech pre-processing and Gaussian distance calculations. A background of the relevant speech recognition theory was given, along with examples of various other techniques and approaches. Descriptions and analyses of the completed design were presented, along with information on how it was tested. In addition to the hardware based system, a multi-purpose software toolkit was built that greatly helped during the design and testing stages of the project.

The principle goal was to design and implement part of an HMM based speech recognition system in embedded hardware, in order to evaluate and learn about such a system. It was not intended to be used as a real recogniser, but rather to broaden the author's knowledge and experience, and provide a platform on which embedded speech recognition may be researched further. The conclusions are given below.

7.1 Usefulness of Results

Due to size limitations of the FPGA, it was not possible to use a complete speech model with the implemented system. However, the intention was never to build something that was usable with such a model, but rather to attempt to judege the usefulness of an FPGA in this situation. With the results achieved it was possible to estimate the relationship between the system speed and model size, and thus

show that the FPGA was capable of processing the data faster than a traditional processor.

Furthermore, this project is a valuable example of how two of the Micro Arcana family may be connected and used in a practical setting. Some of the problems encountered are certainly not unique to this design, and therefore the solutions presented may be helpful in other projects.

As a development platform, there is a large potential for further investigation into embedded speech recognition using the two systems built here. Indeed, there are several different aspects of the design that may be expanded upon or improved, as described in the further work section. This is not a failing of the current project – speech recognition is such a large and complex area that only the most basic system could possibly have been implemented in the time frame given.

7.2 Future Work

There are many possibilities for developing this project futher – either improving the system already built, or implementing other speech recognition tasks such as decoding. With the system already built, there are a few areas that may be interesting to invstigate and develop futher.

Mel Frequency Cepstral Coefficients were computed using an existing library (LibMFCC) for the purposes of this project. However, LibMFCC is extremely slow, and should definitely be optimised. In addition, it would be very beneficial if the pre-processing generated the same MFCCs as those generated by the HTK. This would require an in-depth analysis of the HTK system in order to determine the exact sequence of operations that are performed.

The communications method needs futher work before it can be considered realistically usable. A better system would need to be far faster, and possibly include features such as error-checking, handshaking, and control sequences. Designing a better communications method between two of the Micro Arcana boards would be an interesting project, due to the size and speed contraints of the family.

This project only implemented the first stages of a speech recognition, and did not at all focus on tasks such as decoding or language modelling. However, it provides an environment where these systems may be implemented and used. There is huge potential for futher work that could focus on any one of these areas.

7.3 Personal Gains

As the author is still exploring the vast world of electronic engineering, one of the attractions of this project was its diversity and complexity. The project involved several different areas of electronics, including digital systems design, complex algorithms and optimisation, embedded Linux programming, and hardware testing. Completing this project, with all the different activities involved, was personally satisfying, especially as it involved working with hardware that had never been used in this manner before.

A sense of accomplishment followed completion of the system implementation, as it represented the union of many different technologies and techniques, useful from a educatory point of view, as well as being an interesting and challenging project.

Bibliography

[1] Personal interview, srinandan dasmahapatra, Nov 2012.

[2] Voxforge, 2012. URL http://www.voxforge.org/.

[3] Tor Aamodt. A simple speech recognition algorithm for ece341, 04 2003. URL http://www.eecg.toronto.edu/~aamodt/ece341/speech-recognition/.

[4] J.A. Bilmes. What hmms can do. *IEICE TRANSACTIONS on Information and Systems*, 89(3):869–891, 2006.

[5] SJ Cox and G. Britain. *Hidden Markov models for automatic speech recognition: theory and application*. Royal Signals & Radar Establishment, 1988.

[6] Sadaoki Furui. *Digital Speech Processing, Synthesis, and Recognition*. Marcel Dekker, 1989.

[7] S.K. Gaikwad, B.W. Gawali, and P. Yannawar. A review on speech recognition technique. *International Journal of Computer Applications IJCA*, 10(3): 24–28, 2010.

[8] S.J. Melnikoff. *Speech recognition in programmable logic*. PhD thesis, University of Birmingham, 2003.

[9] S. Nedevschi, R.K. Patra, and E.A. Brewer. Hardware speech recognition for user interfaces in low cost, low power devices. In *Design Automation Conference, 2005. Proceedings. 42nd*, pages 684–689. IEEE, 2005.

[10] L.R. Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286, 1989.

[11] J. Schuster, K. Gupta, R. Hoare, and A.K. Jones. Speech silicon: an fpga architecture for real-time hidden markov-model-based speech recognition. *EURASIP Journal on Embedded Systems*, 2006(1):10–10, 2006.

[12] CMU Sphinx. *Sphinx 3 System Design Documentation*. CMU Sphinx.

[13] Saeed V. Vaseghi. *Advanced Digital Signal Processing*. John Wiley and Sons, fourth edition, 2008.

[14] Steve Young, Gunnar Evermann, Mark Gales, Thomas Hain, Dan Kershaw, Xunying Liu, Gareth Moore, Julian Odell, Dave Ollason, Dan Povey, Valtcho Valtchev, and Phil Woodland. *The HTK Book*. Cambridge University Engineering Department, 2009.