

POLITECNICO DI TORINO

Corso di Laurea Magistrale in Ingegneria Elettronica

Tesi di Laurea Magistrale

A Look Up Table-free Gaussian Mixture Model-based Speaker Classifier



Relatori:

Prof. Mariagrazia GRAZIANO

Prof. Amit TRIVEDI

Candidato:

Alberto GIANELLI

Dicembre 2018

Acknowledgments

This thesis is the result of six months of research and study at the University of Illinois At Chicago and the culmination of my student career. I must express my very profound gratitude to my family, whose untiring and passionate support lead me through this beautiful path of fatigue and satisfactions and who gave me the opportunity to start my studies at Politecnico di Torino and conclude them on the other side of the world.

I would also like to thank my brilliant and stimulating advisor Prof. Amit Trivedi for his constant advice, my Italian advisor and Professor Prof. Mariagrazia Graziano, my laboratory and house mate Andrea for the inspiring brainstorming sessions, my colleague Nick and all the friends that I had the opportunity to share this amazing experience with.

Thank you.

Summary

In this thesis an ASIC design of an hardware GMM-Based Speaker Classifier is presented. The Classifier is a fundamental component of a Speaker Identification system, that is able to associate an unknown incoming speech signal to its unknown speaker, which is part of a previously modeled group of speakers. The design flow follows a software-to-hardware approach, since the whole system is firstly implemented in Matlab, then increasingly transformed from high-level to machine-level until its hardware description. Innovative techniques that avoid any memory accesses to perform hardware exponentials and logarithms are presented. All the computations are executed on-chip and this gives extra performances and extra security to the system. Thanks to its low power demand, it is suitable to be integrated in many IoT devices to personalize the user experience without compromising the power budget of the device.

Table of contents

Acknowledgments	I
1 Introduction	1
1.1 Text-Independent Speaker Identification	1
1.2 Purpose & Motivations: a low power-oriented design	3
1.2.1 Power in Digital systems	3
1.2.2 Applications and target features	4
1.2.3 FPGA V.S. ASIC	6
1.3 Document structure	7
2 Backround theory	8
2.1 <i>Mel-Frequency Cepstrum Coefficients (MFCCs)</i>	8
2.2 The <i>Gaussian Mixture Model (GMM)</i>	10
2.2.1 Model description	10
2.2.2 GMM Parameters estimation (<i>Training</i>)	11
2.2.3 Maximum Likelihood Classifier (<i>Identification</i>)	12
2.3 TIMIT Acoustic-Phonetic Continuous Speech Corpus	13
3 GMM-based Classification Algorithm	15
3.1 Code overview and structure	15
3.2 Frame partitioning and logarithmic domain	17
3.3 Training	19
3.3.1 Identification	22

3.4	Performance considerations	22
4	From Software to Hardware	24
4.1	Introduction	24
4.2	Hardware's computational requirements	24
4.2.1	$K_{s,m}$ computation	25
4.2.2	$z_{f,s,m}$ computation	26
4.3	Computational Challenges: exponentials and logarithms	27
4.3.1	The Log-Add algorithm	29
4.4	Linear Piecewise Function-based (LPF) approximation	30
4.4.1	Parameters estimation and data localization	31
4.4.2	LPF complexity and results	35
4.5	Precision analysis	36
4.6	Numeric Hardware Optimizations	39
5	Hardware GMM-based Classifier	41
5.1	Architecture	41
5.1.1	Top view	41
5.1.2	Control Unit	42
5.1.3	Datapath	47
5.1.4	Testbench	59
5.2	Results	62
5.2.1	Simulation	62
5.2.2	Synthesis	64
6	Conclusion	72

6.1	Achievements	72
6.2	Future goals	72
	Bibliography	74

List of figures

1.1	Speaker Recognition Classification	2
1.2	Application example	4
1.3	Power dissipation of commercial FPGAs	6
2.1	Physical classification of speech features	9
3.1	High-level representation of the Speaker Identification process	16
3.2	Block diagram of the Maximum likelihood classifier	19
3.3	Overlapping frame explanation	19
3.4	% of success: SA2 only train samples (up); SA1+SA2 (down)	23
4.1	Four segment-LPF approximation of the exponential	30
4.2	two segment-LPF	31
4.3	DR4, 10 Speakers' training data percentage distribution	34
4.4	Overview of the LPF parameter extraction	35
4.5	% of success: two segment-LPF and different speech lengths (10 Sp.)	36
4.6	% of success: two segment-LPF and different speech lengths (20 Sp.)	37
4.7	% success comparison: 10 Sp. VS 20 Sp. VS speech length	37
4.8	Decision boundaries illustration	38
4.9	“Fixed Point Converter” tool showing the range of z values	39
5.1	Top view of the Hardware GMM-Classifer	41
5.2	Control Unit states diagram	42
5.3	Datapath block diagram	48
5.4	Shifting Register File architecture	51
5.5	Timing snapshot of the load and the shifting phases	51

5.6	Arithmetic Logic Unit architecture	52
5.7	ELPF Unit architecture	54
5.8	Timing snapshot of the ELPF Unit in the case $a \leq z < b$	54
5.9	Timing snapshot of the ELPF Unit in the case $z < a$	55
5.10	Timing snapshot of the ELPF Unit in the case $z \geq b$	55
5.11	Programmable Shifter architecture	56
5.12	Arithmetic and Logic right shift example	56
5.13	LLPF Unit architecture	58
5.14	Speakers Shifting-Register File	59
5.15	Winner Takes All Circuit	60
5.16	Testbench structure	60
5.17	Matlab snapshot of the first two frames speakers' probabilities	62
5.18	Hardware probability values (first frame)	63
5.19	Hardware probability values (second frame)	63
5.20	Speakers' probabilities over 400 frames (expected wonned: Speaker 2)	64
5.21	Comparison between Software and Hardware-generated probabilities .	65
5.22	Relative error of each speakers (top), average relative error (bottom)	65
5.23	Snapshot of the top view of the system	67
5.24	System power consumption for $f_{clk} = 10$ Mhz	69

List of tables

1.1	KEY IOT EDGE PLATFORMS' POWER BUDGETS	5
2.1	DIALECT DISTRIBUTION OF SPEAKERS IN TIMIT	14
2.2	TIMIT SPEECH MATERIAL	14
3.1	SYSTEM CONSTANTS	20
4.1	% OF SUCCESS WITH DIFFERENT EXP. APPROXIMATIONS .	35
5.1	DATA ANALYSIS	61
5.2	BUSES DATA DISTRIBUTION	61
5.3	DETAILED POWER REPORT AT $F_{CLK} = 10MHz$	71

Chapter 1

Introduction

1.1 Text-Independent Speaker Identification

Speaker *identification* is the process of determining an unknown speaker's identity by performing a comparison 1:N between an unknown voice sample and previously modelled speakers. This practice is different from speaker *verification* or *authentication* which, instead, operates a 1:1 check between one speaker's voice and one model. These two areas are the major applications of the more general field that goes by the name of *Speaker recognition*, that generally refers to recognizing people from their voice. As explained in [1], Speaker recognition can also be classified as *text-dependent* or *text-independent* (1.1). In text-dependent systems the users are allowed to pronounce only a pre-fixed set of phrases in order to be recognized, for example determined password-phrases such as "Hey Siri " or "Ok Google ". In this work, a more challenging and flexible task is tackled: text-independent speaker recognition, which assumes that the identification operation happens regardless of which word is spoken by the user. This also implies that the *train* utterances, used for building the speakers models, may have to be completely different from the *test* ones. More precisely, it is convenient to choose not to use the same phrases for the training and the testing phase in order to avoid introducing unwanted text-dependencies, thus to give robustness to the system.

The utility of identifying a person from his/her voice is increasing with the growing use of speech interaction with computers. The ability of implementing an automatic identification of a speaker has multiple applications in *forensic sciences* since a lot of information is exchanged via telephone between two parties, including criminals for instance. Not only forensic specialists but also ordinary people can benefit from speaker recognition technology: telephone-based services already implement speech and speaker recognition algorithms and the developing of these technologies will

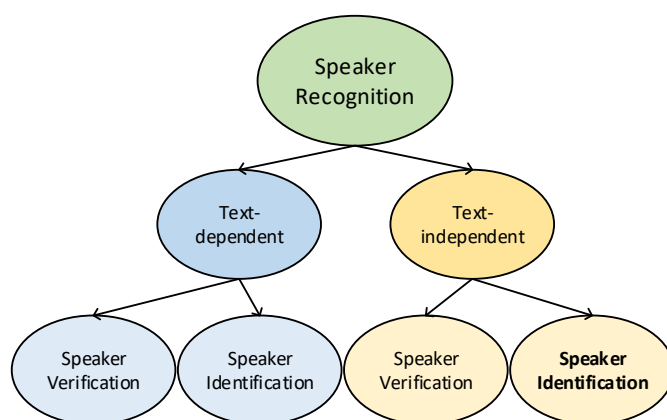


Figure 1.1: Speaker Recognition Classification

supplement or even replace human-operated telephone services in the near future. Recently Pradeep K. Bansal and his team [2] invented a *Speaker-verification digital signature system* that allows a party to provide a signature by speaking a phrase. This provides a greater confidence in communications because the signing party can speak a different phrase for each document, since the technology is text-independent, making very hard any falsification, that are still possible with the digital signature.

1.2 Purpose & Motivations: a low power-oriented design

1.2.1 Power in Digital systems

In the contemporary era of portable digital electronics, where many devices are battery powered, the energy demand of the systems has become a crucial aspect of a digital design.

The total power of a digital system can be divided in *Leakage power* and *Dynamic power*. The leakage power refers to the “stationary” consumption of a device when it is powered. This type of consumption mainly depends on the amount of resources employed and the area of the device, hence a way to reduce it is to apply techniques that are finalized to the reduction of the hardware, like *resource sharing*, and *parallelism minimization*. The leakage power also depends on the transistor threshold voltage V_{th} : with a low V_{th} the MOS requires less voltage to be activated, hence the transistor is faster, but it is also closer to an active stage from an electrical point of view and this cause it to have more current “leaks”, which are equivalent to an higher leakage power; on the other hand, a transistor with an high threshold voltage is less sensitive to current leaks (lower leakage power), but requires more energy to be activated (slower).

The Dynamic (or Active) power is related to the activity of the system, and it can be express as:

$$P_{dyn} = E_{sw} C_L V_{DD}^2 f_{clk} \quad (1.1)$$

where C_L is the capacitance load and E_{sw} is the switching probability of the load,

commonly known as *Switching Activity*. We can note the quadratic dependency of the power from the voltage supply and the linear dependency from the clock frequency, that is actually a design parameter. Later in the document, we will see how the the system allows us to reduce of the clock frequency, and so the dynamic power. Every switch from 0 to 1 in a digital system corresponds to a cycle of charge-discharge of a load capacitance, that is modeled with C_L and depends of the amount of resources that are electrically connected to the wire that is switching: this is why the more the system switches the more it consumes, therefore the dynamic power is related to the activity of the system. The reduction of the dynamic power is a more wide task, and it can be done reducing the power supply of the system, reducing the frequency and reducing the switching activity applying algorithm optimization techniques.

1.2.2 Applications and target features

The purpose of this work is to design the hardware for a Maximum Likelihood Classifier, essential component of an Speaker Identification System targeted for applications related to domestic Internet Of Things (IoT) environments.

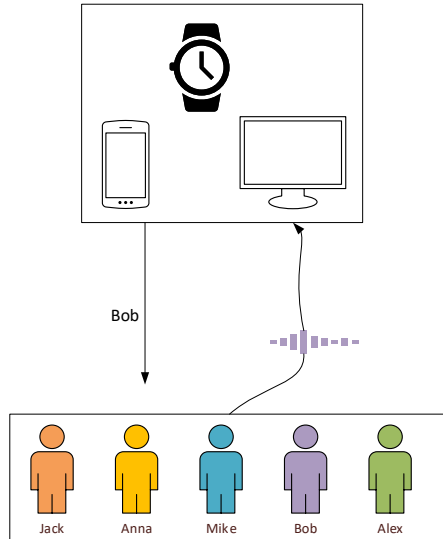


Figure 1.2: Application example

1.2 shows a general application example of the system: in the case of a domestic scenario, the integration of a speaker identification system inside every day-electronics like smart-phones, smart-watches, smart-TVs, and smart-hubs, would allow a much higher control, for example personalizing certain actions depending on who is interacting with the device. With an integrated speaker identification system, the ability to validate the identity of a speaker via a pre-determined set of words (i.e. “Ok Google ” or “Hey, Siri ”) would be extended to the recognition of a group of speakers and without any constraints on the phrase to be pronounced. An other strength of the system is that it operates completely on-chip, without the need to communicate with a remote server: this gives high reliability and security to the system, since it would not need an internet connection to work and the speech data would be kept locally safe. Other possible application areas where this kind of system could be employed are reported in 1.1 ([3]).

This application scenarios requires the system to be real-time, low-power and portable. Looking at 1.1, we can understand the order of magnitude of the power

IoT Edge Platforms	Active Mode Power	Potential new capabilities with speech processing
Wearable activity tracking	$\sim 9.5mW$	Environment tracking based on its sound. Log activity by speech.
Security, Smart locks	$\sim 400\mu W$	Voice enabled unlocking. Speaker verification.
Surveillance	$\sim 20mW$	Intruder detection based on sound.
Environment control	$\sim 5mW$	Voice enabled HVAC control using remote and distributed devices.
Healthcare, Hearing aids	$\sim 100 - 2000 \mu W$	Record conversation as text.
Body-area monitoring	$\sim 140\mu W$	Voice-enabled control.
Smart meeting/classroom	$\sim 10mW$	Record conversation as text using distributed and remote devices.

Table 1.1: KEY IOT EDGE PLATFORMS’ POWER BUDGETS

consumption of some IoT devices. Most of them requires around tens of milli-Watts of power, therefore a good achievement for this work would be to have total power consumption for the Classifier that does not go beyond 1 mW. In this way the

integration of a Speaker Identification system in a pre-existent IoT Device would not compromise the energy requirements.

Later we will discuss how the speed of the system is not a critical parameter in this kind of application, since the time available to process one single frame is 10 ms, which is not a restrictive constrain.

1.2.3 FPGA V.S. ASIC

Traditionally, SI Systems where implemented via software, whose sequential compiling causes the system to be slow. Recent FPGAs have a very high logic capacity and contain embedded Arithmetic Logic Units (ALUs) to optimize signal processing performance, and they are usually employed for high throughput real-time signal processing applications. However, the fine grain programmability of the FPGA is paid for by poor energy performances.

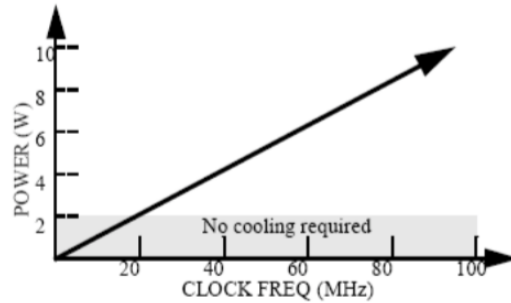


Figure 1.3: Power dissipation of commercial FPGAs

1.3 shows the power dissipation of commercial FPGAs. In a portable environment with a power budget of the order of mWs, the present FPGA architectures will dominate the power budget allocation. For this reason an Application Specific Integrated Circuit (ASIC) is chosen. The reprogrammability cost of the FPGA is cut by implementing ad-hoc circuit to execute a specific purpose. In fact, the circuit can be synthesized with the wanted technology scale and low-power techniques can be easily applied at algorithm, circuit and device level. Moreover, the ASIC unit size is significantly lower than that of the FPGAs.

1.3 Document structure

This document is structured following the logic process that has been used in order to obtain the final hardware system. Three main logic steps were followed:

- **High level coding of the Classification algorithm.** This first step is deepened in Chapter 3 and consists in studying and understanding how the algorithm and the different phases of the Speaker identification process, writing the system by means of high level functions in Matlab, testing its logical functioning and analyzing constraints and performances.
- **Conversion of code from high level to low level.** Once the functioning of the system is validated, it is necessary to re-write and re-structure the code with the final purpose of easily go through the fixed-point conversion. This requires the study of every single function of the high level code, localize the bottle-neck operations that require high computational effort, high storage demand and high decimal precision and re-model them in a low power hardware-oriented design style. This crucial phase of the design with all the innovative techniques, such as the *Linear Piecewise Approximation Unit* or the *Shifting Register Files*, are discussed in Chapter 4.
- **Hardware Design, Simulation and Synthesis.** The final stage of the process is dealt in Chapter 5. It consists in designing the actual hardware blocks that execute the classification starting from the previously obtained low-level Matlab code. In this section the Data Path and the Control Unit are designed on paper, then described in VHDL language in the *Libero SoC* Environment, simulated using *Altera Modelsim* and finally synthesized with the *RC Compiler* tool by *Candence*.
- **Reports and final results.** At the end of Chapter 5, all the synthesis results (power, area and timing) are reported and commented. A comparisons between the performances of the software and the hardware version of the speaker classifier is done.

Chapter 2

Background theory

2.1 *Mel-Frequency Cepstrum Coefficients (MFCCs)*

A speech signal, from a spectrum analysis point of view, is an audio signal with a frequency band that ranges approximately from 100 Hz to 17 kHz, including both male and female voice capabilities. However, since this type of sound is generated by a very specific apparatus of the human body, it has very unique and particular acoustic characteristics, generally called *features* that can be very meaningful in certain fields as linguistics, speech recognition or, such in the case of this work, speaker recognition. Speech signals contains an high number of features whose nature can be relevant or not depending on the application. Only specific types of features are important in order to recognize a speaker. As explained in [1] ideal features should

- have a large variability between different speakers and a small variability within the same speaker
- be robust against noise and distortion
- occur naturally and frequently in speech
- be reasonably easy to extract from the speech signal
- be difficult to mimic
- not be affected by long-term voice variation due to speaker's health or other human conditions.

Moreover, the number of feature should not be too high, since is proved that the GMM cannot handle high-dimensional data. In this work, for instance, 12 features

for every 20 ms speech frames are extracted. From a physical point of view the features can be classified as shown in 2.1.

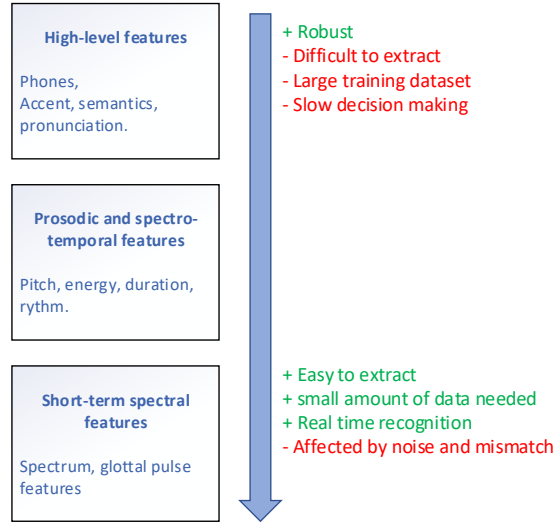


Figure 2.1: Physical classification of speech features

The choice of the type of features depends on the application: generally high-level features are used for capturing conversational-level data, while short-term ones are suitable for describing the *short-term envelope* which can be used to discriminate a voice *timbre* from another. For this reason short-term spectral features are the most suitable feature for a classic Speaker Recognition task, also due to their simple nature and low storage demand. Since the speech signal is continuously changing because of articulation movements, it is convenient to slice them in time *frames* of 20-30 ms duration. Within these intervals the signal is assumed to remain stationary and a spectral features vector is extracted for each frame. The Mel-Frequency Cepstrum Coefficient (MFCCs) are the result of a dimension reduction applied to the actual spectral feature extracted, and they are well-known to be effective in speaker recognition task. The extraction process of the MFCCs is dealt later in the document.

2.2 The *Gaussian Mixture Model* (GMM)

2.2.1 Model description

In order to perform a text-independent speaker identification, it is necessary to create an *acoustic model* that describes the features of the speaker's voice. The model chosen for this work is the Gaussian Mixture Model, a well-known method first applied to speaker recognition by Reynolds, D. A in 1992 [1]. It consists in a statistical process that models the distribution of each speaker's acoustic features, called *MEL-Frequency Spectrum Features (MFCC)*, in a multidimensional space. The Gaussian Mixture Model is the result of the fitting of a *Gaussian Mixture Density* over a series of observations, that in this applications corresponds to the phrases pronounced by the speakers.

A Gaussian mixture density is a weighted sum of M components. Its equation, given the GMM, can be expressed as follow:

$$p(\vec{x}|\lambda) = \sum_{m=1}^M p_m b_m(\vec{x}) \quad (2.1)$$

where \vec{x} is a D-dimensional vector, $b_m(\vec{x})$, $m = 1, \dots, M$, are the component densities and p_m are the mixture weights. Each component density is a D-variate Gaussian Function of the form,

$$b_m(\vec{x}) = \frac{1}{(2\pi)^{D/2} |\Sigma|^{1/2}} e^{-\frac{1}{2}(\vec{x} - \vec{\mu}_m)' \Sigma_m^{-1} (\vec{x} - \vec{\mu}_m)}. \quad (2.2)$$

The complete GMM Model λ is described by three parameters: the mean vector μ , the covariance matrix Σ and the mixture weights p :

$$\lambda = \{p_m, \mu_m, \Sigma_m\} \quad \text{where} \quad m = 1, \dots, M. \quad (2.3)$$

In order the mixture to be a true probability function, the weights must satisfy the constraint that $\sum_{m=1}^M p_m = 1$. In this application, every component of the GMM represent an *acoustic class*, namely a distinct group of phonetic events, such as vowels, nasals or fricatives. Thus, the mean vector μ_m represents the average

features for an acoustic class and the covariance matrix Σ_m indicates the variability of features within the acoustic class.

2.2.2 GMM Parameters estimation (*Training*)

Building the GMM Model is equivalent to find the optimal values of its three parameters with respect the input observation (2.3). This stage is commonly known as the *Training* phase of the model. It is based on the *Maximum Likelihood Estimation (MLE)* of the parameters, a general method for estimating the parameters of a stochastic process starting from a set of observations. More precisely is a *maximization* problem, since its purpose is to find the parameters that maximize the model's probability function over a set of observed samples. The probability function for a model λ is defined as the joint probability density of a set of observations $X = \{\vec{x}_1, \vec{x}_2, \dots, \vec{x}_F\}$ considered as a function of λ . Indeed $p(X|\lambda)$ (2.1) is called probability function when is treated as a function of λ . In order to be able to apply the maximum likelihood parameters estimation, the *likelihood equation* ([4]) has to be satisfied:

$$\frac{\partial p(X|\lambda)}{\partial \lambda} = 0 \quad (2.4)$$

Finding the GMM parameters

$$\lambda = \{p_m, \mu_m, \Sigma_m\} \quad \text{where} \quad m = 1, \dots, M$$

that solve 2.4 from an analytic approach resulted in having non-closed form solutions. This is why the maximum likelihood parameters can be found via an iterative estimation procedure based on the *Expectation Maximization (EM)*. The basic idea behind the EM algorithm is summarized as follows:

1. Initialize λ , set convergence threshold th_c
2. Estimate new model $\bar{\lambda}$ such as $p(X|\bar{\lambda}) > p(X|\lambda)$
3. Use estimate $\bar{\lambda}$ as new model
4. If th_c is reached, go to 5, otherwise go to 2
5. End

2.2.3 Maximum Likelihood Classifier (*Identification*)

In the identification phase, the parameters found in the previous Training phase are used to discriminate the speakers. The objective of this stage is to associate an input test utterance to a previously modeled speaker. The structure dedicated to this purpose is called *Maximum Likelihood Classifier*, and its design and implementation is the focus of this thesis. The main action of the unit is fairly simple:

1. Compute the posterior probability (2.1) of the test utterance over each speaker modeled in the training phase.
2. Declare the winner speaker selecting the maximum probability value.

In this segment the assumptions behind these two operations are shown.

Let's consider the case of identifying a speaker from a unique feature vector \vec{x} . Recalling the *Bayes' Theorem*

$$P(A|B) = \frac{P(B|A) \cdot P(A)}{P(B)} \quad (2.5)$$

the probability that \vec{x} was pronounced by the speaker s is given can be expressed as

$$P(\lambda_s|\vec{x}) = \frac{P(\vec{x}|\lambda_s) \cdot P(\lambda_s)}{P(\vec{x})} \quad (2.6)$$

where $P(\lambda_s)$ is the *a priori* probability of speaker s to be the unknown speaker and $p(\vec{x})$ is the probability density function for the observation \vec{x} . As introduced earlier, the winner speaker is the one that has the highest probability. The classification rule is expressed as follows:

$$\frac{P(\vec{x}|\lambda_s) \cdot P(\lambda_s)}{P(\vec{x})} > \frac{P(\vec{x}|\lambda_t) \cdot P(\lambda_t)}{P(\vec{x})} \quad \text{where } t = 1, \dots, S \ (t \neq s) \quad (2.7)$$

It is possible to simplify the terms $P(\vec{x})$ and assume that $P(\lambda_s) = P(\lambda_t)$ meaning that all the speakers have equal probabilities to be the unknown speaker. These two assumptions yield

$$P(\vec{x}|\lambda_s) > P(\vec{x}|\lambda_t) \quad \text{where } t = 1, \dots, S \ (t \neq s). \quad (2.8)$$

The classification is now based only on evaluating each speaker's probability density function for the observation vector \vec{x} and choosing the maximum value. Extending the equation for many subsequent statistically independent observations $\{\vec{x}_r\}_{r=1}^R$ we obtain:

$$\prod_{r=1}^R P(\vec{x}_r|\lambda_s) > \prod_{r=1}^R P(\vec{x}|\lambda_t) \quad \text{where } t = 1, \dots, S \ (t \neq s). \quad (2.9)$$

The left term is known as the *probability function* and in the following chapter it will be explained its relationship with the logarithmic domain.

2.3 TIMIT Acoustic-Phonetic Continuous Speech Corpus

All the speech samples used in this work comes from the *TIMIT Acoustic-Phonetic Continuous Speech Corpus* ([5]), a collection of phonemically and lexically transcribed speech of American English speaker of different sexes and dialects. The database was design to provide speech data to acoustic-related studies. TIMIT was built under sponsorship from the *Defense Advanced Research Projects Agency- Information Science and Technology Office (DARPA-ISTO)* and it is the result of the joint effort of Institute of Technology (MIT), Stanford Research Institute (SRI), and Texas Instruments (TI).

The corpus contains a total of 6300 sentences, 10 sentences spoken by each of 630 speakers from 8 major dialect regions of the United States. 2.1 shows details about the male/female and dialects distribution in the corpus. As illustrated in 2.2, three type of sentences are present for each speaker:

- **SA:** *dialect* sentences meant to expose the dialectal variants of the speakers. They are read by all 630 speakers.
- **SX:** *phonetically-compact* sentences, designed to provide a good coverage of pairs of phones, with extra occurrences of phonetic contexts thought to be either difficult or of particular interest. Each speaker read 5 of these sentences and each text was spoken by 7 different speakers.
- **SI:** *phonetically-diverse* sentences, selected from existing text sources - the

Table 2.1: DIALECT DISTRIBUTION OF SPEAKERS IN TIMIT

Dialect Region	Directory #	#Male	#Female	Total
New England	1	31 (63%)	18 (27%)	49 (8%)
Northern	2	71 (70%)	31 (30%)	102 (16%)
North Midland	3	79 (67%)	23 (23%)	102 (16%)
South Midland	4	69 (69%)	31 (31%)	100 (16%)
Southern	5	62 (63%)	36 (37%)	98 (16%)
New York City	6	30 (65%)	16 (35%)	46 (7%)
Western	7	74 (74%)	26 (26%)	100 (16%)
Army Brat (moved around)	8	22 (67%)	11 (33%)	33 (5%)

Brown Corpus (Kuchera and Francis, 1967) and the Playwrights Dialog (Hultzen, et al., 1964) - so as to add diversity in sentence types and phonetic contexts. Each speaker read 3 of these sentences, with each sentence being read only by a single speaker.

Table 2.2: TIMIT SPEECH MATERIAL

Sentence Type	#Sentences	#Speakers	Total	#Sentences/Speaker
Dialect (SA)	2	630	1260	2
Compact (SX)	450	7	3150	5
Diverse (SI)	1890	1	1890	3
Total	2342	/	6300	10

Chapter 3

GMM-based Classification Algorithm

3.1 Code overview and structure

At the beginning of the work a software version of the system was implemented in order to test the correctness of the operations from a logical point of view, without dealing with precision, resources or power consumption. To do so, it is necessary to first design and build the system in a developing environment where is easy to experiment and test and, possibly, with built in high level functions. For this purpose, the Matlab environment was chosen. Since probability calculations involve complex computations like exponentials and logarithms, having a fully working software code is crucial in order to have a solid base where to start designing the hardware from, and be aware of the problems that are related to the functional point of view. The choice to work in Matlab has many motivations:

- A scripting language, so it gives the possibility to either run commands through scripts automatically or to alternatively execute them one-by-one by a human operator, which is very helpful in the debug test and for testing quick commands on the go.
- A wide set of built-in functions available and fully documented that fits the first need to set up a code that is working without dealing too much about the efficiency or the implementation of that particular function.
- A straightforward Input/Output. It is relatively easy both to import and process external data and export results and draws plots.
- Useful toolboxes.

As shown in 3.1 The speaker identification process is made up of two consecutive phases: the *Training* phase and the *Identification* phase.

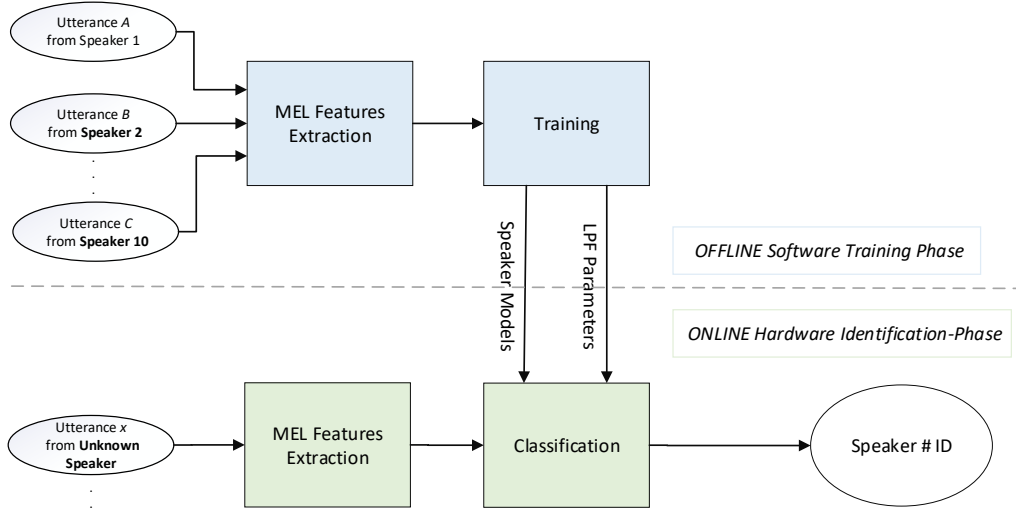


Figure 3.1: High-level representation of the Speaker Identification process

As the image shows, the identification phase is designated to be implemented in hardware. This work, in particular, is focused on the design of hardware classifier, hence it will be assumed to have the online MFCC features ready. However, in this preliminary design stage, both of the phases are executed by software functions in the Matlab environment. In the training phase the Gaussian Mixture Models of a specific set of speakers are built by extracting the MFCCs features from sentences pronounced by each individual user. This preliminary stage is executed offline and it is necessary to provide the system the tools to perform the classification in the identification phase. In the identification phase the actual classification happens: an unknown input phrase is analyzed, the MFCCs are extracted and, using the pre-computed speakers model, the probability that the phrase belongs to the Speaker s ($s = 1, \dots, S$) is computed. The winner speaker is the one that produced an higher probability among the other. In order to give a clearer understanding of the flow, a simplified pseudo-code of the algorithm is presented below:

```
1 for i = 1: n_speakers{
2     train_speech = read_audio_from_speaker(i);
3     train_mfcc   = extract_MFCC(train_speech)
4     GMM(i)       = create_GMM(train_mfcc);
5 }
6 test_speech      = read_input_audio();
7 test_mfcc        = extract_MFCC(test_speech)
8 for i = 1: n_speakers{
9     log_prob(i)  = compute_prob(test_mfcc, GMM(i))
10 }
11 winner          = find_max(log_prob)
```

3.2 Frame partitioning and logarithmic domain

All the speech data used in this project come from the *TIMIT database*. As previously introduced, the database is organized in different folders, that corresponds to different American English dialects. Since the phrases pronounced by the different speakers have different durations, it is convenient to fix the length of both the test and the train phrase. The *train_speech_length* is usually set to 5 seconds, and the *test_speech_length* can be set from 2 to 5 seconds. The variation of this two parameters influences the accuracy and the speed of the system. Since computing 5 seconds of speech at the same time would require an enormous storage demand and a huge latency, the audio sample is sliced in $T_w = 20$ ms duration frames (3.3). In order to give more robustness to the frame partition and be sure not to lose any voice articulation, the frames are overlapped by a time T_s that is usually set at 10 ms. In this way if an important feature occurs right at the end of the first frame, for instance, it would be captured during the processing of the second. Doing so the posterior probability is not calculated over the entire voice sample, but over every single frame. Indeed the theory of the Maximum Likelihood Classifier treated in the previous Chapter applies, since each frame can be considered statistically independent observations of the unknown speaker. Hence, being X the set of subsequent observations $X = \{\vec{x}_1, \vec{x}_2, \dots, \vec{x}_F\}$,

$$P(X|\lambda_s) = \prod_{f=1}^F P(\vec{x}_f|\lambda_s) \quad f = 1, \dots, F. \quad (3.1)$$

Where F is the number of frame, X is the entire unknown speech sample, x_f is the current frame and λ is the GMM of speaker s calculated in the training phase. Now it is clear how 2.8 applies to the current scenario. However, since the probability values of each frame is a number very close to zero, the multiplication of the values of many frames would lead to an extremely small number that would be very difficult to represent. Moreover, thinking in an hardware perspective, multiplying such small numbers, for hundreds of times would be computationally very expensive. For these reasons it is more convenient to shift into a logarithmic domain. Thanks to the logarithm property :

$$\log(A \cdot B) = \log(A) + \log(B) \quad (3.2)$$

Hence, taking the logarithm on both terms on 3.1, we obtain:

$$L(\lambda) = \log(P(X|\lambda_s)) = \log\left(\prod_{f=1}^F P(\vec{x}_f|\lambda_s)\right) = \sum_{f=1}^F \log(P(\vec{x}_f|\lambda_s)) \quad (3.3)$$

The right term is known as the *log probability function*. In this way it is possible to adopting a standard accumulation method, after having computed the logarithm of every frame probability. It is interesting to notice that, although it would be necessary to return in the linear domain in order to obtain the correct probability value, for our purpose this operation is not necessary, since it only matters the relative comparison between the probability values and not the single value itself. Hence, the identification is performed in a *logarithmic probability domain*. as shown in 1.1

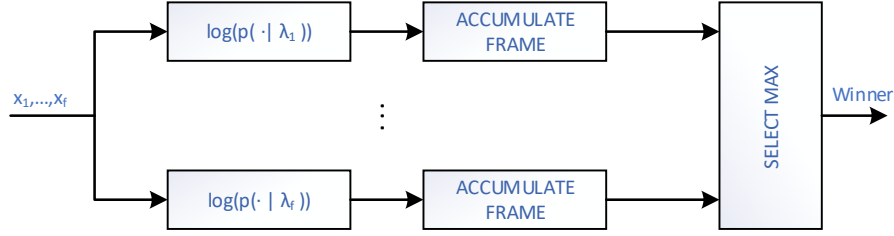


Figure 3.2: Block diagram of the Maximum likelihood classifier

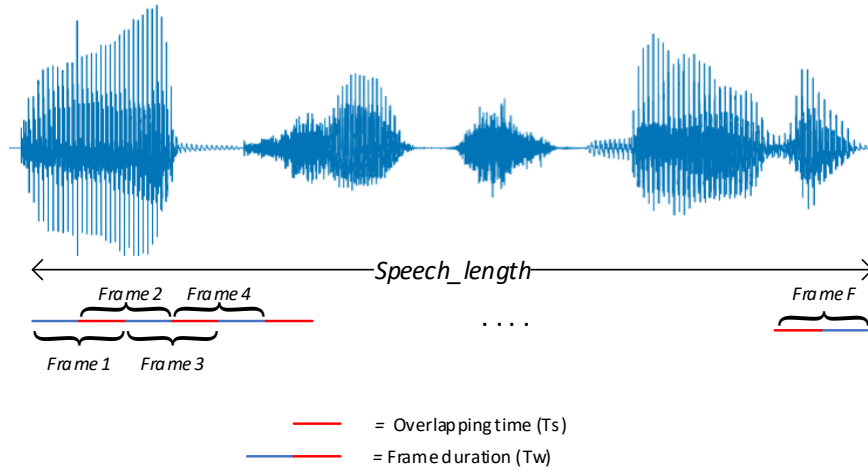


Figure 3.3: Overlapping frame explanation

3.3 Training

In the first section of the code, the parameters are defined. Analyzing them before going through the code, is useful and helps understanding some of the important parameters of the code whose modification can cause variations in performances. The most relevant parameters are reported in 3.1. As mentioned in section 4.2, the frames are 20 ms long and they are overlapped by 10 ms. In this way a 5 s speech sample would be divided in 500 frames.

The purpose of the training phase is to build the models of the selected set of speakers. This operation is executed in the code by the function:

Parameters name	Value	Description
M	20	Number of Gaussian Mixture Components
D	12	Number of MFCCs
Tw	20	analysis frame duration (ms)
Ts	10	analysis frame shift (ms)
Fs	16000	sampling frequency
$Train_speech_length$	2000	train speech length (ms)
$Test_speech_length$	5000	Test speech length (ms)

Table 3.1: SYSTEM CONSTANTS

```
1 Create_GMMModel(speech ,M, D, fs , Tw, Ts)
```

that takes as inputs: the entire speech sample, the number of GMM components of the wanted model, the number of feature to be extracted, the sampling frequency of the speech sample, the duration of the single frame and the duration of the frame overlapping. Inside the function two main operation are executed:

1. MFCCs Extraction
2. GMM Training (EM algorithm)

The features extraction is executed by the following Matlab built in function:

```
1 [CC, FBE, frames] = mfcc(...)
```

The process of the coefficients extraction involves filtering, short-term Fourier transform, magnitude spectrum computations and compression. For the purpose of this work is not necessary all the steps of the MFCCs extraction process, however an in-depth analysis is reported in the appendix A. The *mfcc()* function returns three parameters: *CC* is the matrix of Mel-Frequency Cepstral Coefficients (MFCCs) with feature vectors as columns, *FBE* is a matrix of filterbank energies with feature vectors as columns, *FRAMES* is a matrix of windowed frames. We are only interested in the *CC* matrix.

After having obtained the coefficients, only $D=12$ of those are selected:

```
1 MFCC = CC((1:D) , :)
```

The resulting MFCC matrix has as many columns as the number of frames (F) and D rows. In this way every column of D coefficients correspond to the set of observations for each frame. The GMMModel is then trained with the built-in function $fitgmdist(X,k)$

```
1 G=fitgmdist(MFCC,M, 'CovarianceType', 'diagonal', 'RegularizationValue',  
    ,0.01);
```

The function $fitgmdist(X,K)$ fits a k -components Gaussian mixture distribution to the data X using the Expectation Maximization (EM) algorithm. The matrix X in this application is the MFCCs features matrix of dimensions $[D \times F]$. Two directives are added to the inputs of the function:

- '*CovarianceType*': 'the covariance matrix is restricted to be diagonal. This reduces the complexity of the system and allows simplifications to the hardware.
- '*RegularizationValue*' A non-negative regularization value is added to the diagonal of each covariance matrix to ensure that the estimates are positive-definite.

The output of the function is a Matlab structure G that contains the parameters of the GMM Model.

Finally the parameters are extracted from the structure and saved in the proper directory:

```
1 Save_GMM_Model( GMMModel.mu, GMMModel.Sigma, GMMModel.  
    ComponentProportion, file_name, model_dir)
```

3.3.1 Identification

The section dedicated to the identification executes the following operations

1. Extract the MFCCs of the unknown input utterance $X = \{\vec{x}_1, \dots, \vec{x}_F\}$, with the same function presented in the last section.
2. For every speaker s compute the posterior probabilities of every frames (observations of the unknown speakers) $P(\lambda_s | \vec{x}_f)$ (2.3). Compute the logarithm of the probabilities of each frame (3.3) and accumulate them (3.2).
3. Select the maximum probability to declare the winning speaker.

3.4 Performance considerations

In order to understand if the system is actually working, the classification is applied in multiples directories of the TIMIT database. As illustrated in 2.1, every directory contains three different type utterances for each speaker, labeled as “SA ”, “SX ” and “SI ”. In [1] is explained that the SI and SX-type utterances are more suitable for the training, while the SA-type are more suitable for the testing. 3.4 shows the performance difference between two system: both of them are trained with the SI and SX files, but the first system is only tested with the SA2 utterances, while the second is tested with both SA1 and SA2 samples concatenated. The better performances of the second system demonstrates that, not only the length of the training sample is relevant, but also the amount of input data. Is intuitive to think that if the unknown user talks for a longer time is more likely to be recognized.

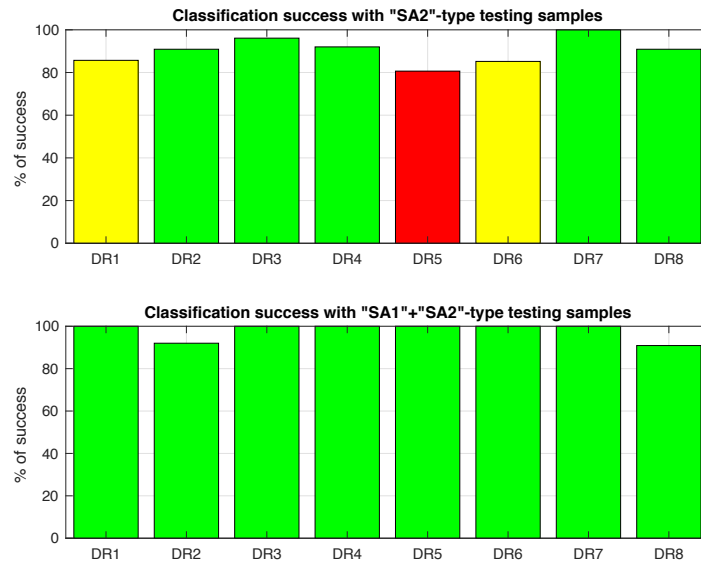


Figure 3.4: % of success: SA2 only train samples (up); SA1+SA2 (down)

Chapter 4

From Software to Hardware

4.1 Introduction

Now that the functioning of the algorithm is tested using Matlab high-level functions, it is necessary to unroll all the computations and write a low level code that only executes a series of simple operations in order to step towards the hardware implementation. From now on the focus of the work is shifted on designing the Maximum Likelihood classifier only, since is the purpose of this thesis. Recalling 1.1, the training phase executed offline by the software that has just been designed.

The following chapter is going through the steps taken for the transformation of the code from high-level to low-level code.

4.2 Hardware's computational requirements

Putting together 2.1, 2.2 and 3.3 it is possible to write a single equation that shows all the computations behind the *posterior log-probability* calculation.

$$p(X|\lambda_s) = \sum_{f=1}^F \log \left\{ \sum_{m=1}^M \left[\frac{p_{s,m}}{2\pi^{\frac{D}{2}} \sqrt{|\Sigma_{s,m}|}} \exp\left(-\frac{1}{2}(\vec{x}_f - \vec{\mu}_{s,m})' \Sigma_{s,m}^{-1} (\vec{x}_f - \vec{\mu}_{s,m})\right) \right] \right\} \quad (4.1)$$

After having applied 4.1 to each speaker $s = 1, \dots, S$ it is necessary to select the maximum probability, that indicates the unknown speaker. The calculation involves a lot of complex operations such as exponentials, logarithms, multiplications, divisions, matrix determinant, vector-matrix multiplications and square roots. Although implementing all these operations in hardware can seem intimidating and computationally too expensive, it is possible to reduce the computational load by

making some considerations about the nature of the variables, in order to understand which operation must be executed real-time by the classifier and which one can be pre-computed offline. Referring to 4.1, all the parameters labeled with the subscripts s are derived from the speakers’ models λ_s that are computed in the training phase. Let’s re-write 4.1 as follows:

$$p(X|\lambda_s) = \sum_{f=1}^F \log \left\{ \sum_{m=1}^M \left[\mathbf{K}_{s,m} \cdot \exp(-\mathbf{z}_{f,s,m}) \right] \right\} \quad (4.2)$$

where

$$K_{s,m} = \frac{p_{s,m}}{2\pi^{\frac{D}{2}} \sqrt{|\Sigma_{s,m}|}}, \quad (4.3)$$

$$z_{f,s,m} = \frac{1}{2}(\vec{x}_f - \vec{\mu}_{s,m})' \Sigma_{s,m}^{-1} (\vec{x}_f - \vec{\mu}_{s,m}) \quad (4.4)$$

4.2.1 $K_{s,m}$ computation

The factor $K_{s,m}$ only depends on numeric constants and factor derived from the speakers’ models: D is the number of the selected MFCC’s features, set to 12; $p_{s,m}$ is the proportion factor from the GMM Model; $\Sigma_{s,m}$ is the diagonal covariance matrix also derived from the GMM Model:

$$\Sigma_{s,m} = \begin{bmatrix} \sigma_{s,m}(1) & 0 & 0 & \dots & 0 \\ 0 & \sigma_{s,m}(2) & 0 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & \sigma_{s,m}(12) \end{bmatrix}$$

For this reason $K_{s,m}$ can be precomputed in the offline training phase and stored on-chip. In this way the multiplications, the division, the determinant calculation and the square root necessary to calculate the parameter don’t have to be executed real-time by the actual hardware. This consistently reduce the hardware complexity. However, the value of $K_{s,m}$ changes for every GMM components and for every speakers so actually a vector of numeric values has to be stored. This storage overhead will be discussed in Chapter 6.

4.2.2 $z_{f,s,m}$ computation

Unlike $K_{s,m}$, $z_{f,s,m}$ does depend on the input MFCC vector extracted from the unknown speech \vec{x}_f , therefore its value cannot entirely be pre-computed offline. Since a matrix-vector multiplication is involved, it is useful to further lower the complexity by expanding the calculation. Since every frame of the unknown input is described by $D = 12$ MFCC components we have:

$$\begin{pmatrix} x_f(1) \\ x_f(2) \\ \vdots \\ x_f(12) \end{pmatrix} - \begin{pmatrix} \mu_{s,m}(1) \\ \mu_{s,m}(2) \\ \vdots \\ \mu_{s,m}(12) \end{pmatrix} = \begin{pmatrix} x_f(1) - \mu_{s,m}(1) \\ x_f(1) - \mu_{s,m}(2) \\ \vdots \\ x_f(1) - \mu_{s,m}(12) \end{pmatrix} = \begin{pmatrix} y_{f,s,m}(1) \\ y_{f,s,m}(2) \\ \vdots \\ y_{f,s,m}(12) \end{pmatrix} \quad (4.5)$$

re-writing the parameter $z_{f,s,m}$ expliciting the matrix-vector multiplication we have:

$$z_{f,s,m} = \frac{1}{2} \cdot \begin{pmatrix} y_{f,s,m}(1) \\ y_{f,s,m}(2) \\ \vdots \\ y_{f,s,m}(12) \end{pmatrix}^T X \begin{pmatrix} \frac{1}{\sigma_{s,m}(1)} & 0 & 0 & \dots & 0 \\ 0 & \frac{1}{\sigma_{s,m}(2)} & 0 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & \frac{1}{\sigma_{s,m}(12)} \end{pmatrix} X \begin{pmatrix} y_{f,s,m}(1) \\ y_{f,s,m}(2) \\ \vdots \\ y_{f,s,m}(12) \end{pmatrix} \quad (4.6)$$

Expanding the computation we obtain:

$$\begin{aligned} z_{f,s,m} &= \frac{1}{2} \left(y_{f,s,m}(1) \cdot \frac{1}{\sigma_{s,m}(1)} \cdot y_{f,s,m}(1) \right) + \dots + \left(y_{f,s,m}(12) \cdot \frac{1}{\sigma_{s,m}(12)} \cdot y_{f,s,m}(12) \right) = \\ &= \frac{1}{2} \left((y_{f,s,m}(1))^2 \cdot \frac{1}{\sigma_{s,m}(1)} \right) + \dots + \left(y_{f,s,m}(12)^2 \cdot \frac{1}{\sigma_{s,m}(12)} \right) = \\ &= \frac{1}{2} \sum_{d=1}^{D=12} y_{f,s,m}(d)^2 \cdot \frac{1}{\sigma_{s,m}(d)} \end{aligned}$$

Developing the vector-matrix multiplication it is possible to get rid of multi-dimentional arithmetics (that would required dedicated hardware) by adding an other stage of

accumulation. We can re-write $z_{f,s}$, in the following way.

$$z_{f,s,m} = \frac{1}{2} \sum_{d=1}^D (x_f(d) - \mu_{s,m}(d))^2 \cdot \frac{1}{\sigma_{s,m}(d)} \quad (4.7)$$

In this way, by simply re-arranging the equation, we decided to execute the vector-matrix multiplication using a serial approach. This results in an additional simplification of the hardware and considerably lower power consumption. An alternative high-speed solution is proposed in [6], consisting in a serial-parallel multiplier that allows to execute the vector-matrix product in parallel. However, is important to remark that our timing constraint is very relaxed (10ms) hence every step of the design is thought to prioritise the power consumption over the performance.

From 4.7, it is possible to notice that the factor $\frac{1}{\sigma_{s,m}(d)}$ can be pre-computed offline. We can now express the re-arranged equation for the posterior probability calculation, which highlight the low-level operations that the hardware would execute:

$$p(X|\lambda_s) = \sum_{f=1}^F \log \left\{ \sum_{m=1}^M \left[\mathbf{K}_{s,m} \cdot \exp(-\mathbf{z}_{f,s,m}) \right] \right\} \quad (4.8)$$

where

$$K_{s,m} = \frac{p_{s,m}}{2\pi^{\frac{D}{2}} \sqrt{|\Sigma_{s,m}|}} \quad (4.9)$$

is going to be entirely precomputed offline and stored on-chip and

$$z_{f,s,m} = \frac{1}{2} \sum_{d=1}^D (x_f(d) - \mu_{s,m}(d))^2 \cdot \frac{1}{\sigma_{s,m}(d)} \quad (4.10)$$

is going to be calculated online by the hardware units.

4.3 Computational Challenges: exponentials and logarithms

4.8 explicits the main arithmetics of the classifier, which involves

- 1 Subtraction
- 1 Square root
- 2 Multiplications
- 3 Accumulations
- 1 **Exponential**
- 1 **Logarithm**

Note that the list above does not include the multiplication for $\frac{1}{2}$ that can be executed by only shifting the wires to the left by one position, without any need of hardware resources. This is valid for every multiplication or division for a factor that is a power of two, and this consideration will be the core of further optimizations that will be introduced later.

One of the harder challenge of this thesis, is to find a clever solution to implement the exponential and the logarithm in hardware, without impacting the power too much. A classic and well-known method for implementing complex function in hardware is using *Look Up Tables (LUT)*. Unfortunately LUT- based approximations suffers from large on-chip storage demand([6], [7]). Moreover, since in a digital design they are usually implemented by means of ROM or FLASH memories, the high number of accesses would hardly affect the power demand of the classifier.

In addition, the input values of the logarithm, are probability values that goes from 0 to 1. This means the slice of logarithmic domain that the data occupies is very non-linear and steep. This means that, in order to have a sufficiently precise grid of discrete values that would be stored in the LUT, a very high precision is required in order to avoid saturation phenomena. This is due to the intrinsic nature of the logarithm. This also excludes the possibilities to use a series of well-known approximation techniques ([?]) based on the *Mitchell Approximation*, that approximates the logarithm with a reasonably low error only in the part of the domain where x is greater than one, in fact the formulation is usually expressed as the approximation of $y = \log(1 + m)$.

4.3.1 The Log-Add algorithm

In this section the log-Add algorithm is presented and the reasons why was not chosen for this application are discussed. In [7] an other technique that simplifies the problem of the logarithm and the exponential computation is presented: the *log-add* technique exploits a logarithm property to avoid the computation of the exponential. 3.3 shows the basic principle behind the algorithm.

$$\log(A + B) = \log A \left(1 + \frac{B}{A}\right) = \log A + \log \left(1 + \frac{B}{A}\right), \quad (4.11)$$

where $A > B$. If $A < B$ A and B has to be switched in the formula. In the solution proposed in [7], the system obtains the term $\log(1 + \frac{A}{B})$ by calculating

$$\log\left(\frac{B}{A}\right) = \log B - \log A, \quad (4.12)$$

and using a LUT to convert $\log(\frac{B}{A})$ to $\log(1 + \frac{B}{A})$. In the paper is explained that the dimension of the lookup table used for this mapping is much smaller than the one required for computing the exponential, that in this case doesn't need to be computed, since $\log A$ and $\log B$ in this application corresponds to the logarithm of the posterior probabilities, whose exponential is cancelled by the logarithm:

$$\log A \rightarrow \log(p_m b_m(\vec{x}_f)) = \log(p_m) - \frac{D}{2} \log(2\pi) - \frac{1}{2} \log(|\Sigma_m|) - \frac{1}{2} (\vec{x}_f - \mu_m)' \Sigma_m^{-1} (\vec{x}_f - \mu_m) \quad (4.13)$$

In this way the problem of the exponential is bypassed in a smart way, but the following resource overhead is introduced

- Log-add look up table
- Hardware logic to verify $A > B$ made with a comparator, an adder, a subtractor and a multiplexer (see Figure 7 in [7])

Although this is a straightforward and very used in the speaker recognition architectures, the need of accessing a LUT many times can result in a higher power consumption, for this reason the focus is shifted to an other technique called *Linear Piecewise Function(LPF)-based approximation*.

4.4 Linear Piecewise Function-based (LPF) approximation

In the article “An Efficient Digital VLSI Implementation of Gaussian Mixture Models-Based Classifier” from *Minghua Shi* and *Amine Bermak* ([6]) the LPF-based implementation is proved to be much more hardware-friendly. This technique is not based on a mathematical re-structuring of the posterior probability equation, but is based on the straight piecewise approximation of the exponential function. It is impor-

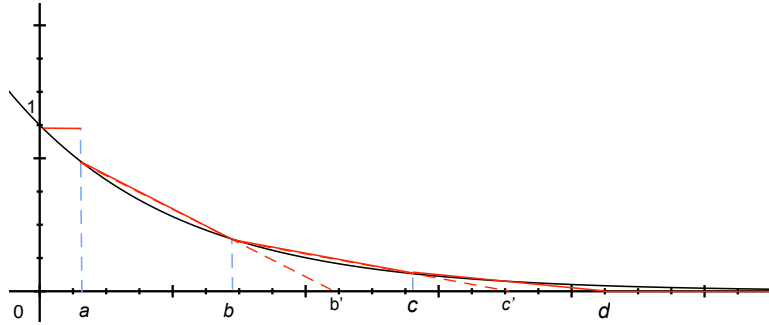


Figure 4.1: Four segment-LPF approximation of the exponential

tant to notice that even if we have to compute an exponential and a logarithm, we are interested in approximating the exact same function since the logarithm is the inverse function of the exponential. 4.1 shows an example of LPF approximation. In this case four segment are used to approximate the function and, clearly the greater is the number of segment used, the more accurate the approximation would be. Performing this type of operation, effects the whole Gaussian Mixture Model: in fact, the gaussian’s shape would be effected as well however, the decision boundaries mismatch is minimum, hence the classification capability of the system would not be compromised(??) . This statement can be false depending on the variability of the input data, in fact the complexity of the LPF, hence the number of segments used for the interpolation may vary depending on the application. In [6], the authors presented an hardware LPF unit that can perform the approximation using three different level of complexity: one, two or three segments. In this way the system is reusable for different type of data that requires different precision. Since we are dealing with only speech data, we are not interested in providing this flexibilty. For

this reason the complexity is set to two segments, and it would be eventually increased if a lack of accuracy is observed. The value of the approximated exponential would have a value depending in which segment the input value is located. Referring to 4.2 that shows a two segment-lpf, the output is calculated as follows:

$$LPF(z) = \begin{cases} 1, & \text{for } z < a \\ m_1(b - z), & \text{for } a \leq z < b \\ 0, & \text{for } z \geq b \end{cases} \quad (4.14)$$

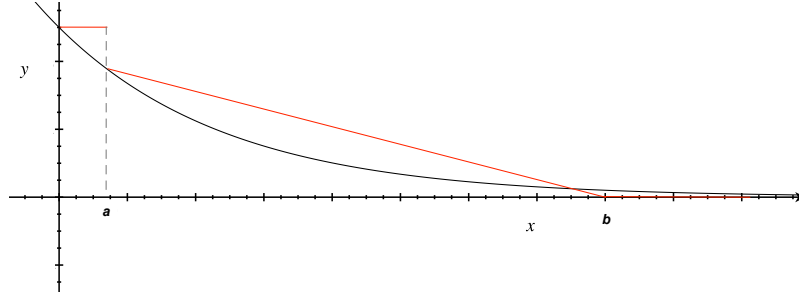


Figure 4.2: two segment-LPF

From 4.14 it's clear that the computation of the approximated exponential corresponds to executing basic operations such as comparisons, subtractions and multiplications, that does not require any memory access. It is interesting to notice that, even if we are heavily distorting the exponential function, hence the GMM gaussians, we are not compromising the functioning of the system, since we are not interested to the actual values posterior probabilities for the input samples over each individual speaker, but only on finding the maximum one. This means that the values of the actual likelihoods are wrong and meaningless but their classification is still valid (if the decision boundaries are not compromised).

4.4.1 Parameters estimation and data localization

In order to be able to implement the LPF, it is first necessary to find its parameters that are the junction points between the segments ($a, b, c \dots$) and the angular coefficients of the segments (m_1, m_2, \dots). The number of parameters to find depends on the level of complexity of the LPF. As previously said, the complexity

is initially set to two segments to avoid possible useless complications, hence the parameters to be found are: a, b and $m1$. Since this operation is not supposed to be executed by the hardware, it is possible to rely to the computational power of Matlab. In [6], the parameters a and b are obtained by performing a grid-search that selects the two parameters that give the lowest performance mismatch with respect the original GMM. However, since we also need to approximate the logarithm, that would require a different set of LPF parameters, the operation of minimizing a performance cost-function resulted too complex to implement. Moreover, in this phase of the project we are not interested in finding the optimal parameters combination, but we just need a combination that provides a reasonably low performance loss with respect the original GMM classifier. Hence, an alternative solution based on *data localization* is performed as follows: the same data used for building the GMM during the training phase (SI and SX audio samples from TIMIT database) are used to find the optimal parameters. More precisely all the values of z in every training-classification process are concatenated and ordered in a vector $z_{training}$. This operation has a double purpose: firstly, allowed us to understand the order of magnitude of the LPF Unit input z ; secondly, it defines the slice of exponential domain that actually needs to be approximated: since we are using a very rough approximation of the exponential in order to minimize the hardware required, it is not possible to approximate the exponential function over a wide portion of domain, since the approximation would be very poor and all the data would be saturated to the same probability. In 4.3 the location of the training input samples is shown. Note that they are not the actual values that are going to be used as LPF inputs in the actual identification phase, but, since they come from the same speakers, they give a trustable domain localization for the test data.

Once the vector $z_{training}$ is sorted, the domain of interest is established. The parameters a and b are chosen by setting the wanted percentage of data corresponding to their position. In order to better explain this concept an example is shown below:

Let's consider the case of a four segments-LPF (4.1), the function would be

computed as follows:

$$LPF(z) = \begin{cases} 1, & \text{for } z < a \\ m_1(b' - z), & \text{for } a \leq z < b \\ m_2(c' - z), & \text{for } b \leq z < c \\ m_3(d - z), & \text{for } c \leq z < d \\ 0, & \text{for } z \geq d \end{cases} \quad (4.15)$$

The critical areas of the approximation are where the inputs are saturated to the same output value, that is to say where $z < a$ ($LPF(z) = 1$) and where $z \geq d$ ($LPF(z) = 0$). This would result in the same posterior probability value, so the decision boundary mismatch would be mainly determined from the two saturation areas. For this reason is reasonable to chose to saturate a low percentage of the data, in order to spread most of the values between a and d , where they are weighted through the segments in a monotonic decreasing way. The following choice of percentages resulted efficient :

- $a \rightarrow 2\%$
- $b \rightarrow 30\%$
- $c \rightarrow 75\%$
- $d \rightarrow 95\%$

In this way we can control the amount of saturated data and or fitted data, being sure that our approximation is actually acting on the data in an efficient way.

It is interesting to analyze a numerical example, to have a quantitative idea of what explained above: The $z_{training}$ vector for 10 speakers in the directory DR4 has 59001 components. The range of values is

$$0 \leq z \leq 5533$$

, hence if we don't consider any type of percentage we would start approximating from 0 till 5333. However, 4.3 shows that the 95% of the data are included in the

interval (0 : 190), which is a 30 times smaller domain than the one containing the 100% of the data (0:5533). This means that we are compromising 5% in order to concentrate the domain where most of the data are, obtaining a more accurate fit.

4.3 also illustrate an other challenge of the hardware implementation: most of the data are in an area of the domain where the value of the exponential is very close to zero. This introduce the problem of dealing with very small numbers and very small slope coefficients. This problem will be tackled in the last section of this chapter.

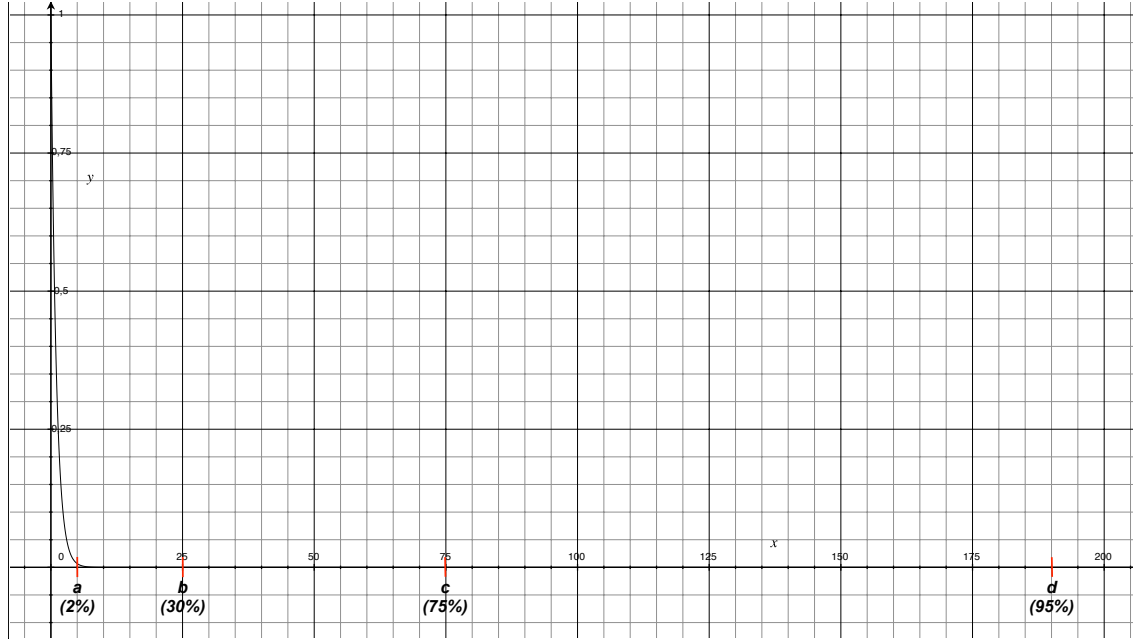


Figure 4.3: DR4, 10 Speakers' training data percentage distribution

In 4.4 a simplified overview of the functions used to implement the parameters' selection is illustrated.

The process for the logarithm is not explicitly described because is exactly the same procedure and the same considerations applies, with the difference that the domain and the co-domain are switched with respect to the exponential.

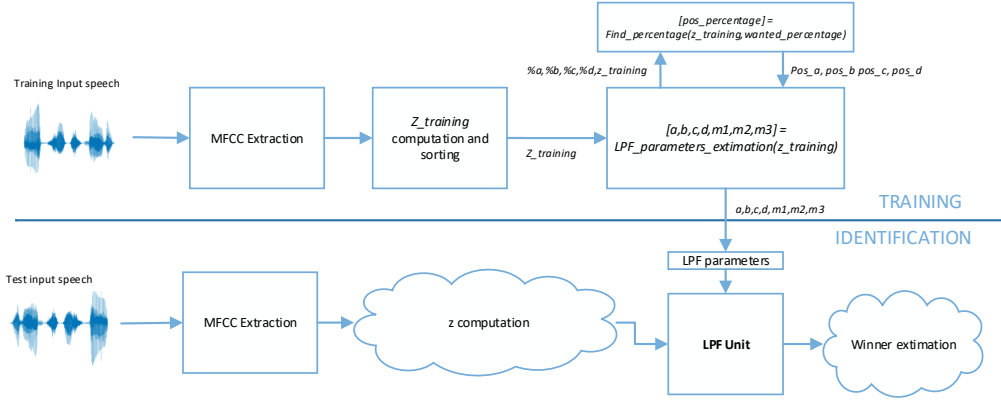


Figure 4.4: Overview of the LPF parameter extraction

4.4.2 LPF complexity and results

After many tests it turned out that a two-segments approximation is enough to have a performance very close to the one of system implemented with the original exponential function. For the logarithm instead, at least a three segments-LPF is necessary to make the system work, under three segments the logarithm is not evaluated properly. Moreover, as shown in 4.1, the complexity of the exponential

System	% of success (10 speakers)
Unbounded exponential	97.5
Two-segment LPF()	97.1
Three-segment LPF()	97.1

Table 4.1: % OF SUCCESS WITH DIFFERENT EXP. APPROXIMATIONS

function was not crucial to the classification percentage. A comparison between the classification success varying the length of the test speech length is reported in 4.5, for ten speakers tested and in 4.6 for twenty speakers. We can note that longer is the test speech, higher is the success percentage. It is interesting to note that a shorter test speech length the lower the number of frames to be computed is, hence the system is faster, but also the amount of data processed is less and this is the reason of the lack of precision when short voice samples are used. We can note from 4.7 that the success when analyzing 20 speakers is significantly lower. This happens because the more are the speakers and the smallest are the decision boundaries, that

might be violated from a weak LPF approximation; moreover, the LPF parameters are estimated using only the train data from the first ten speakers, so the parameters are related only to half of the speakers' data. It is possible to include all the speaker in the parameter estimation, but this required a considerable amount of time due to the need to compute a concatenation of very large vectors. since the hardware is going to be designed for the classification of 10 speakers. In the following section the precision needed for all the variables are studied and the code is finished so that the transition to the hardware is possible and clear.

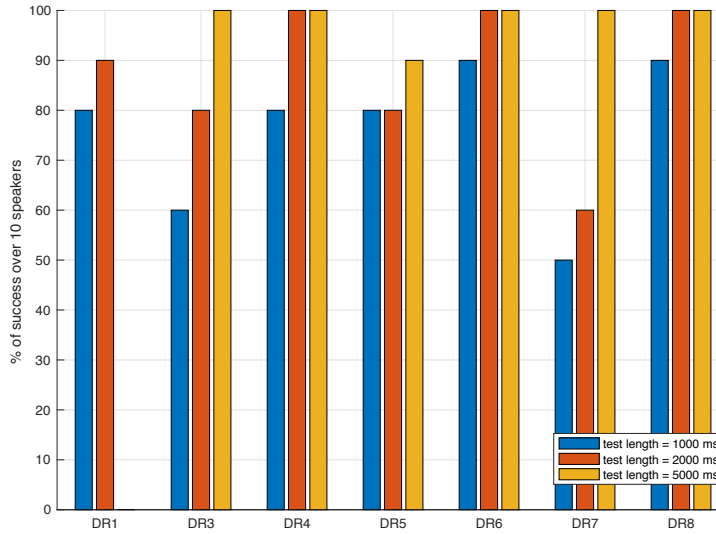


Figure 4.5: % of success: two segment-LPF and different speech lengths (10 Sp.)

4.5 Precision analysis

For the whole design process until now the number of digits or the precision of the computations employed, were not taken into account. Now that we have an hardware-like low level code, the last step is to adapt the code to a numeric format compatible with an hardware implementation. Working in Matlab a floating-point double precision was used; the purpose of this design stage is to cut all the unnecessary digits of every parameter of the classifier so that a conversion to fixed point is feasible without significant performance loss. Even if in the system are involved numbers that differs by many orders of magnitude, the idea of implementing

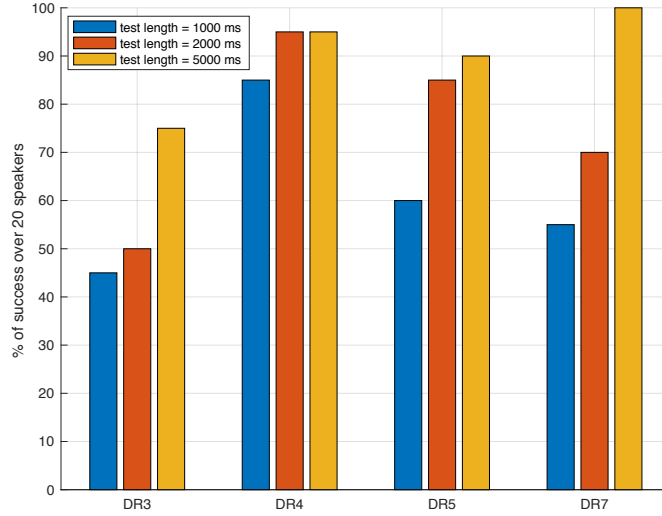


Figure 4.6: % of success: two segment-LPF and different speech lengths (20 Sp.)

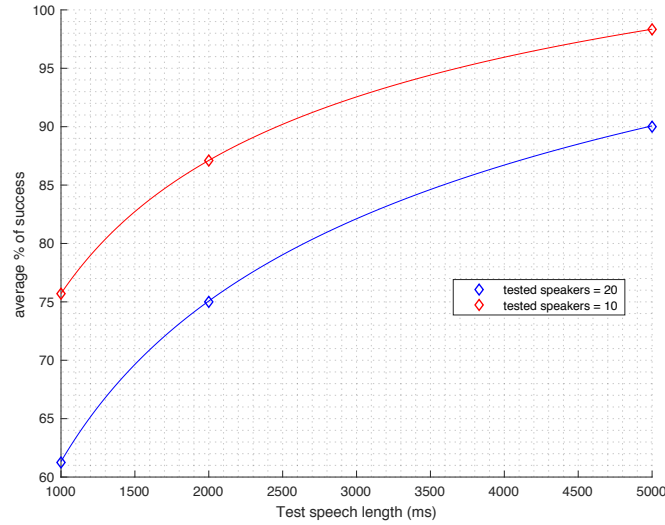


Figure 4.7: % success comparison: 10 Sp. VS 20 Sp. VS speech length

a floating point data-path is excluded due to the complexity overhead that would be introduced, that is possible to overcome by adapting some precision technique that will be shown later. Since we are dealing with fractional, positive and negative numbers, a *signed* format will be necessary in the hardware implementation. Moreover, the signed binary number will need an *integer* part and a *fractional* part, the following notation is used to represent the number of integer and fractional digits in

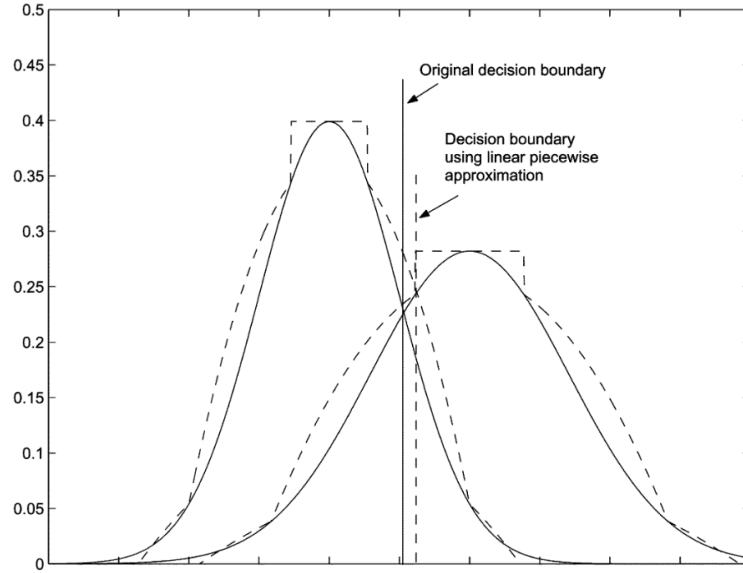


Figure 4.8: Decision boundaries illustration

a binary number:

$$[\text{integer_digits}].[\text{fractional_digits}] \rightarrow \text{IIII.FF} \rightarrow 4.2$$

The arithmetic rules behind a fractional binary number will be discussed in the next chapter.

In order to have an idea of how many bit the system requires in general, we used the Matlab tool “*Fixed Point Converter*” that runs the code and calculates the ranges of all the variables involved in the program. In order to do so the code had to be restructured so that the *main* work as a test-bench for the invoked function. The data analysis performed from the tools applies to all the function called from the main. Hence the GMM-based classifier that we derived was re-arrange so that it could have been written as a function, in order to be processed from the tool. ?? shows a screen-shot where it is possible to observe the range of values that the previously mentioned variable z can assume.

With a trial and error process, all the input parameters are truncated to the minimum number of decimals that keep unchanged the performance.

The number of bit required using a signed representation is computed with the

variable	Type	Sim Min	Sim Max	Whole Number	Proposed Type
▼ Input					
TOTP	double		0	149700 Yes ↕	numerictype(0, 18, 0)
log0	double		0	729 Yes ↕	numerictype(0, 10, 0)
prob_matrix	10 x :499 double		-22624	0 Yes ↕	numerictype(1, 16, 0)
z	1 x 30 double		0	10697.01 No ↕	numerictype(0, 16, 2)

Figure 4.9: “Fixed Point Converter” tool showing the range of z values

following equations:

$$n \text{ bit} = \lceil \log_2(|\text{range}|) \rceil + 1 \quad (4.16)$$

4.6 Numeric Hardware Optimizations

With the LPF-based approximation of the exponential and logarithm we avoided the use of any look up table, and the only parameters that have to be stored are found for every set of speaker, hence the storage required is one register for each LPF parameter. However, a potentially expensive operation is introduced: the multiplication for the slope coefficients of the approximation segments. In addition to this problem, the coefficient are very small ($m1 = -1.4 \cdot 10^{-4}$) in the case of the exponential since, as previously shown, the samples are located in a domain area where the exponential function is very close to zero. Viceversa, for the logarithm, the slope coefficient would be very high ($m1 = 6.7 \cdot 10^{14}$, $m2 = 1.2 \cdot 10^{16}$). This would require a multiplication with very different order of magnitude, so with a large number of bits to cover the all range.

For this reason, since we can manipulate those parameters in the training phase, the problem of dealing with too large or too small numbers was solved by introducing multiplications constants so that all parameters were in a reasonable range. Recalling the main log-probability equation

$$L(\lambda_s) = \sum_{f=1}^F \log \left\{ \sum_{m=1}^M K_{s,m} \exp(-z_{f,s,m}) \right\} \quad (4.17)$$

and substituting the exponential and the logarithm function with the LPF-based approximation (in a case where both the slope coefficients are involved), we obtain:

$$L(\lambda_s) = \sum_{f=1}^F m1 \left[\left(\sum_{m=1}^M K_{s,m} m1 (z_{f,s,m} - b) \right) - bl \right] \quad (4.18)$$

At this point, a multiplication constant 2^{40} is used to compensate too big or too small coefficients:

$$L(\lambda_s) = \sum_{f=1}^F \frac{ml1}{2^{40}} \left[\left(\sum_{m=1}^M 2^{40} \cdot K_{s,m} ml(z_{f,s,m} - b) \right) - 2^{40} \cdot bl \right] \quad (4.19)$$

In this way all the coefficient are compensated and the global result is not changed.

We still have to deal with all the multiplication introduced. In [6] the slope coefficient are approximated to the closest power of two, so that the multiplication does not requires an actual hardware multiplier, but only a wires shift. This idea was tested, and the system didn't show any precision loss, therefore the same technique is applied to all the multiplication constant introduced: $ml1, ml2, K_{s,m}, ml$.

Doing is is not necessary to save the actual values on-chip, but only the number of shifts that corresponds to the multiplication for the closest power of two of the parameter. This resulted feasible in terms of precision, and gives a double optimization: small numbers to store (the shifts can go from -16 to 16) and no multipliers needed. This requires the implementation of a programmable shifter, that will be studied in the next chapter.

Chapter 5

Hardware GMM-based Classifier

5.1 Architecture

The final stage of this thesis is the actual hardware implementation of the classifier. In this chapter a top-down approach is used to analyze the system:

5.1.1 Top view

5.1 shows the top-view of the classifier.

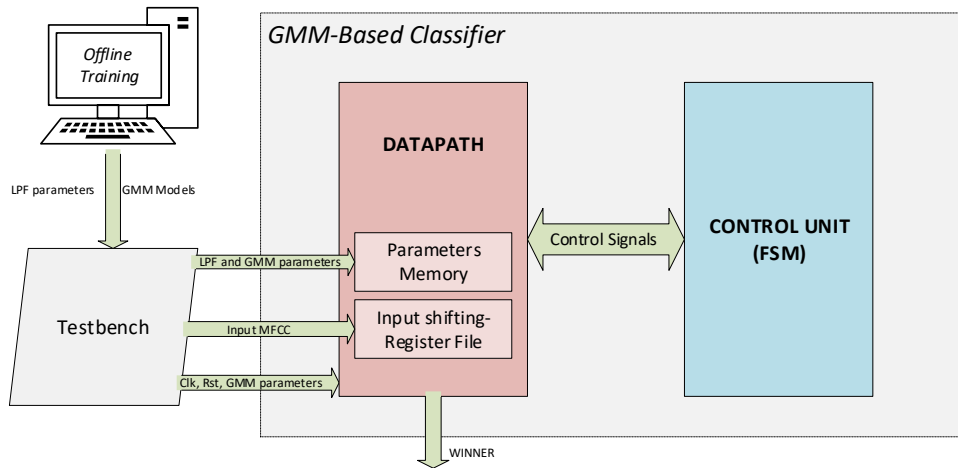


Figure 5.1: Top view of the Hardware GMM-Classifer

It consists of an *execution unit* or *Datapath* and a *Control unit*. This last is implemented as a *Finite State Machine (FSM)* and it communicates with the datapath exchanging control signals. A *testbench* reads all the LPF and GMM parameters and provides them to the datapath. Moreover it replaces the Online MFCC extractor that is necessary to complete the identification system providing, with the proper

5.1.2 Control Unit

5.2 illustrates the entire control unit:

Figure 5.2: Control Unit states diagram

The rectangular blocks represent *states*, while trapezoidal blocks represent conditional branches derived from control signals coming from the execution unit. The elevated number of states makes the representation of the diagram tricky to follow, hence a color code is used. In the following section of the chapter the flow of the operation is described, and an explanation of every state is given. The reader shall also refer to 5.3 in order to understand the main hardware involved along the states. All the units mentioned in this section will be treated in depth in the next Datapath section.

The control unit can be divided in five consecutive stages, that are color coded in 5.2: the first big portion of states (grey) is dedicated *loading of the LPF parameters*: these are fixed for every set of speakers so the loading only happens in the beginning of the classification process. This phase strictly depends on the type of memory where the parameters are stored, but this is not a problem treated in this project, and all the values are stored in files that are read from the testbench during this stage.

The following group of states (orange) are committed to the calculation of $z_{f,s,m}$ (4.2). This is done processing the $d = 12$ components singularly, executing arithmetical operations by means of an *Arithmetic Logic Unit* and a *Programmable Shifter*. At the end of this stage the components are accumulated from the accumulator *Acc_d*.

The next stage is colored in yellow, and it represents the *computation of the exponential*. The main character of this stage is the *Exponential Linear Piecewise Function (ELPF) Unit*. An algorithmic optimization is done in this stage: when $z_{f,s,m}$ is greater than b , the output of the exponential is zero, meaning that there is no need of any probability calculation in the current accumulation cycle, so the control unit skips the accumulation cycle, saving energy and time. On the other hand, when $z_{f,s,m}$ is lower or equal to a the output is one, and the value is directly loaded in the register *Acc_M IN* by means of the multiplexer *mux6* (5.3). At the end of this stage the probability of a single GMM component is ready and is accumulated over the M components.

The green set of states is dedicated to the logarithm computation of the frame probability of the current speaker, just calculated over all the M GMM components. It is possible to notice an higher number of states in comparison with the ELPF

computation, and this is because the hardware of the LLPF uses half of the resources, hence, a greater control is needed. In fact, the LLPF unit checks the position of the input by comparing it with its parameters once at time, and this is why the signal *log_lower* is checked two times in two different moments. This can be more efficient since the if $z_{f,s,m}$ is lower than b , then the second comparison is not performed, saving dynamic power. A similar “bypass ” optimization is implemented here: if the output of the LLPF is zero the value is directly loaded in the *Acc_f In* register trough *mux5*.

This whole cycle is performed for every GMM component with every Speaker models for every frame.

Finally, when the last frame is processed, the *Winner Takes All (WTA)* stage (blue) comes into play. This stage selects the maximum speaker’s probability over all the probabilities, that were saved in separate registers thanks to the *shifting register-file* architecture that will be presented later.

States description

A brief explanation of the main function of the states is already written inside the box of each state. However, in order to have a full understanding, in this section all the thirty-four states of the control unit are described. The reader can use the following list as a reference when the datapath is studied. The names of the states are reported on the upper side of the state blocks 5.2.

- **RESET**: wait until the *start* signal is asserted.
- **LD_AB**: load the ELPF parameters a and b into the ELPF registers.
- **LD_M1_CL**: load the ELPF slope coefficient shifts $m1$ and the LLPF parameter cl .
- **LD_ML2_BL**: load the LLPF slope coefficient shifts $ml2$ and the LLPF parameter bl .
- **LD_ML1_BLS**: load the LLPF slope coefficient shifts $ml1$ and the LLPF parameter bl' .

- **LOAD_FRAME**: parallel load of the current frame MFCC vector x in the *Input shifting-Register File*
- **LOAD_MU_EPS_X**: load the current speaker's GMM parameters $\mu_{s,m}(d)$ and $\frac{1}{\sigma_{s,m}}(d)$
- **X_MU2**: calculate $(x_f(d) - \mu_{s,m}(d))^2$
- **SIGMA_X_MU2**: calculate $(x_f(d) - \mu_{s,m}(d))^2 \cdot \frac{1}{\sigma_{s,m}(d)}$
- **WAIT_D**: wait the *acc_D_In* to sample the data.
- **ACC_D**: Accumulate over d and check if the *d-counter* has asserted its terminal count signal *D_TC*
- **INCR_D**: Enable the *d-counter*, enable the *Input shifting-register File* so that the next component of x_f is available on the next clock cycle.
- **LAST_ACC**: Wait the last accumulation addition to be stored.
- **LOAD_EXP**: Enable the *Exp_in* register.
- **EXP_LPF**: load $K_{s,m}(d)$; compare $z_{f,s,m}$, a and b and establish ELFP Unit output. Check if the signals *exp_zero* or *exp_one* are asserted.
- **M1_SH**: perform a shift of the ELFP Unit output by $m1$.
- **K_SH_PREP**: enable the *one* and the *K_FBK* registers.
- **K_SH**: compute the probability of the current GMM component by shifting the partial result by $K_{s,m}(m)$.
- **ACC_M**: Accumulate over M and check if the *m-counter* has asserted its terminal count signal *M_TC*.
- **INCR_M**: Enable the *M-counter*; Enable the *d-counter*; clear d ; enable the *Input shifting-register File* so that the next component of x_f is available on the next clock cycle.
- **LOAD_LOG**: enable the register *Log_In*.

- **LOG_LPF**: clear M ; compare the probability data with the LLPF parameter bl and check if the signal *log_lower* is asserted.
- **BLS**: subtract the LLPF parameter b' to the input of the LLPF unit.
- **ML2_SH**: shift the output of the LLPF unit by the value $ml2$; enable the *Acc_Fr_In* register.
- **SUB_C**: compare the probability data with the LLPF parameter cl and check if the signal *log_lower* is asserted.
- **CLS**: subtract the LLPF parameter c to the input of the LLPF unit.
- **ML1_SH**: shift the output of the LLPF unit by the value $ml1$; enable the *Acc_Fr_In* register.
- **LOAD_ZERO**: enable the registers *zero* and *Acc_Fr_In*.
- **ACC_FR**: Accumulate over F and check if the *S-counter* and the *F-counter* have asserted their terminal count signals S_TC and F_TC .
- **INCR_SP**: enable the *S-counter*; enable the *M-counter*; enable the *d-counter*; clear d ; clear M ; enable the *Input shifting-register File* so that the next component of x_f is available on the next clock cycle; enable the *Speaker Shifting-Register File* to shift the active speaker for the next cycle.
- **INCR_FR**: enable the *F-counter*; enable the *M-counter*; enable the *d-counter*; clear d ; clear M ; clear S ; enable the parallel load signal of the *Input shifting-register File* so that the next frame can be loaded in the next clock cycle; enable the *Speaker Shifting-Register File* to shift the active speaker for the next cycle.
- **WTA_PREP**: enable the *Speaker Shifting-Register File* to shift the active speaker so that its output is loaded in the WTA Unit.
- **WTA**: compare the new probability with the old one and keep the higher.
- **WINNER**: load the winner register.

5.1.3 Datapath

Overview

The datapath is nothing more than the implementation of 4.2 that was derived before. It is crucial to start with the following premise: the purpose of this design is not to provide an ultra-low power and highly optimized architecture, but to create a solid and functioning base that shows the possibilities that this type of implementation approach can have. The system was highly optimized from the algorithm point of view, in order to provide a minimal and low power data-path, however we will still see an optimization range that was not fully exploited in this work; for instance, some of the blocks, like the multiplier or the shifter, were built using a behavioural VHDL description that may not be the most efficient.

5.3 illustrates the Datapath block diagram. The blocks are color-coded:

- **Green:** Main units. They will be explained one by one in this chapter.
- **Orange:** Accumulation elements.
- **Blue:** Counters.
- **Yellow:** Multiplexers.
- **Grey:** Storage unit and registers. In the block diagram the sequential blocks are marked with a triangle on the bottom left of the box, to indicate the clock signal. The flow of the operations and the parallelism arrangements are highlighted, and for clarity reasons most of the names of the signals are omitted.

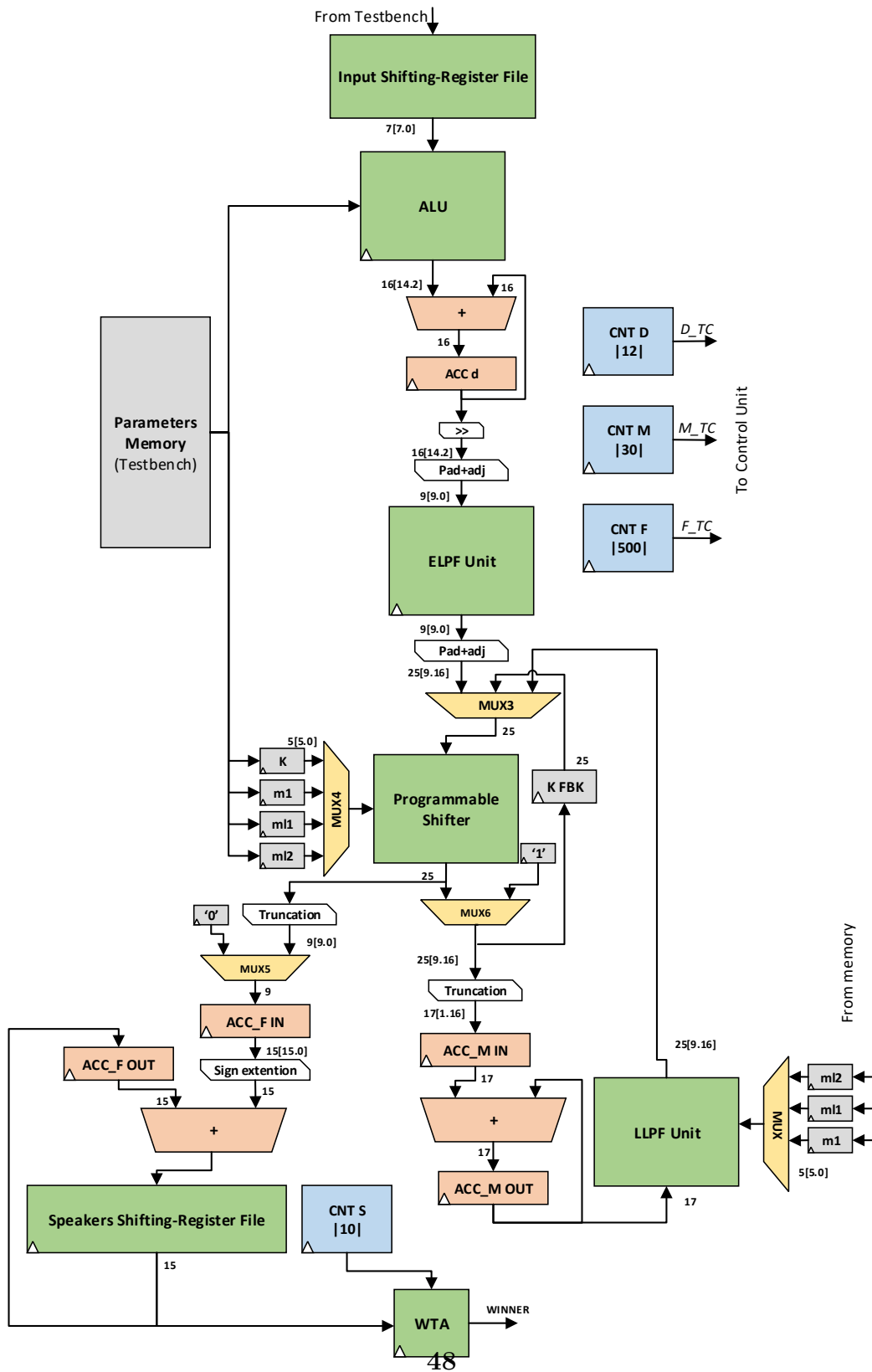


Figure 5.3: Datapath block diagram

The initial part of the datapath, before the *ELPF Unit*, is dedicated to the computation of $z_{f,s,m}$ (refer to 4.1). We shall recall that the computation of $z_{f,s,m}$ is accomplished by calculating the vector-matrix product components-by-components and then accumulating the partial results over d . Then, a shift by one position to the right is performed (multiplication by $\frac{1}{2}$) and only nine bits of the integer part of the number is kept. It is important to notice that the nine bits-number that we are keeping is the result of the 16 bits-arithmetic happening in the *ALU*, therefore the fact that we are only keeping 9 bits does not mean that the *ALU* can work over nine bits as well, because the final integer would be different. This is a crucial concept that also resulted from the precision analysis previously carried out in the software. Once $z_{f,s,m}$ is ready, it is processed by the *Exponential Linear Piecewise Function (ELPF) Unit* that executes the exponential following the technique explained in the previous chapter. The central part of the architecture is based on the *Programmable Shifter* that takes as inputs the value to be shifted and the number of shifts to be executed, that are saved in registers and multiplexed. The output of the shifter goes both to the *frame accumulation* section and to the *Mixture component section*, but clearly, by means of the registers *Acc_F_In* and *Acc_M_In*, it is possible to discriminate the signal path by enabling only one register from the control unit; in fact, the signals goes in different paths depending on the current state, even if the actual bus is routed to both the paths; this is an efficient way to de-multiplex a signal without using an actual de-multiplexer, that would be useless.

When the exponential is calculated, it is multiplied by K , using the shifter and then accumulated over M . At the end of the accumulation the *Logarithm Linear Piecewise Function (LLPF) Unit* computes the logarithm of the probability of the current frame against the current speaker, using a slightly different hardware compared to the *ELPF Unit* that will be discussed later. Once the log-probability of the frame is computed, it is accumulated in the *frame accumulation section* (bottom-right of 5.3). Notice that the accumulation it is not performed over a single register, but over a *Speaker Shifting-Register File* that updates the current speaker's probability; this is necessary since, as mentioned earlier, we are calculating the probability of the current frame over each speaker every time, so a proper storing mechanism is required to keep saved all the speakers' probabilities while the test speech is being processed. This means that, for example, the first frame is compared with all the 10

speakers, the probabilities related to that frame are stored, then the cycle repeats for the second frame and so on till the wanted frame is reached. This workflow allows the speech frame to be computed real-time, without the need to store it and it is versatile, since it allows to stop the classification process at the wanted frame just by changing the module of the *f-counter*. This is a freedom that allows *frame-related optimizations*, such as *Frame Skipping*. Finally when all the frames are evaluated for all the speakers, the *Speakers Shifting-Register File* sequentially load the *WTA* circuit with the final probabilities. The number coming from *CNT S* corresponding to the maximum speaker's probability is kept to indicate the winner speaker.

Input Shifting-Register File

Since we are comparing the same 12-components MFCC input vector x with ten speakers, the same twelve values has to be compared with ten different set of GMM parameters. The component of x are involved in the calculation one by one, hence it is necessary to find a structure that allows to load of the MFCC vector every frame cycle and then provide every component to the *ALU* for every speaker's GMM parameters. A shift register is what we would need if we wanted bit-by-bit shift, but since we want to shift an entire number at a time (7 bits), we need a *Shifting-Register File*. As we can see from 5.4 the architecture is an extended shift registers in a feedback configuration; by controlling the *select signal* of the multiplexers it is possible to choose to perform a parallel load (when a new frame has to be processed) or to loop back the coefficients to expose them to the next speaker.

From 5.5 it is possible to observe how, when a new frame is counted, the signal *ld_sh_n* is set to asserted and twelve new values are loaded in parallel in the unit. In the following clock cycle it is possible to notice the shift operation: the signal *load shift* is now set to zero and the numbers shifts from one position.

ALU

5.6 illustrates the *Arithmetic Logic Unit of the system*: The purpose of the unit is to compute the components of $z_{f,s,m}(d)$ so that they can be accumulated in the *d-accumulation section* of the datapath. A seven bits integer subtractor is used to compute $(x_f(d) - \mu_{s,m}(d))$. The result is squared by means of a multiplier, that is also

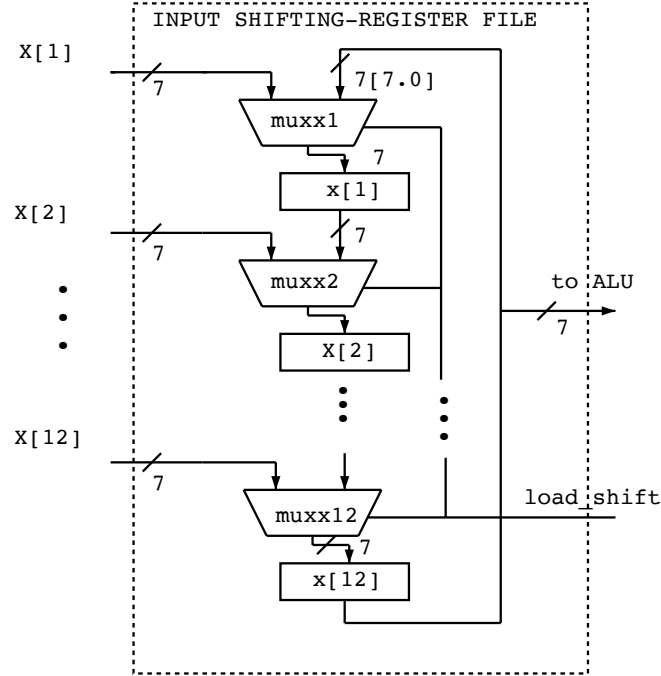


Figure 5.4: Shifting Register File architecture

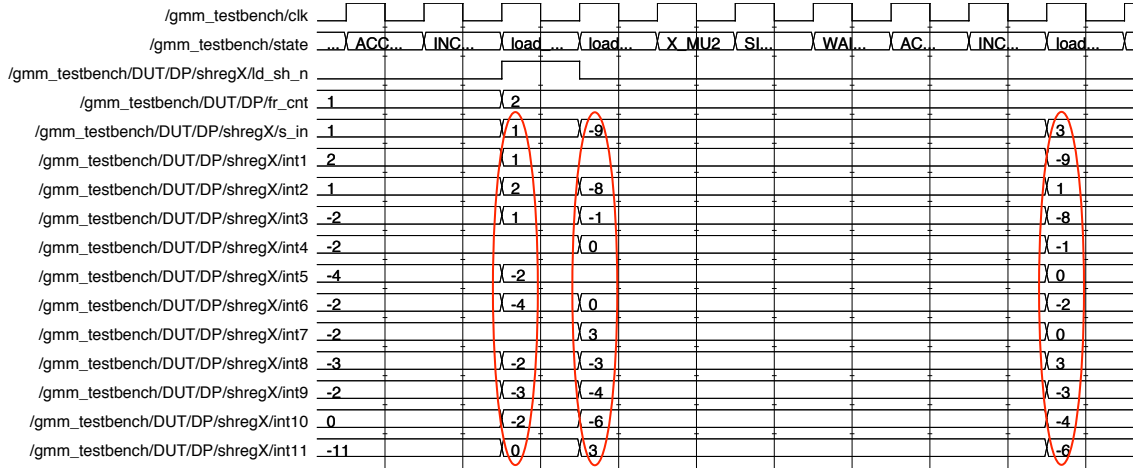


Figure 5.5: Timing snapshot of the load and the shifting phases

used to perform the multiplication by the inverse of the diagonal covariance matrix components $\frac{1}{\sigma_{s,m}(d)}$, that cannot be performed with shift technique since its decimal

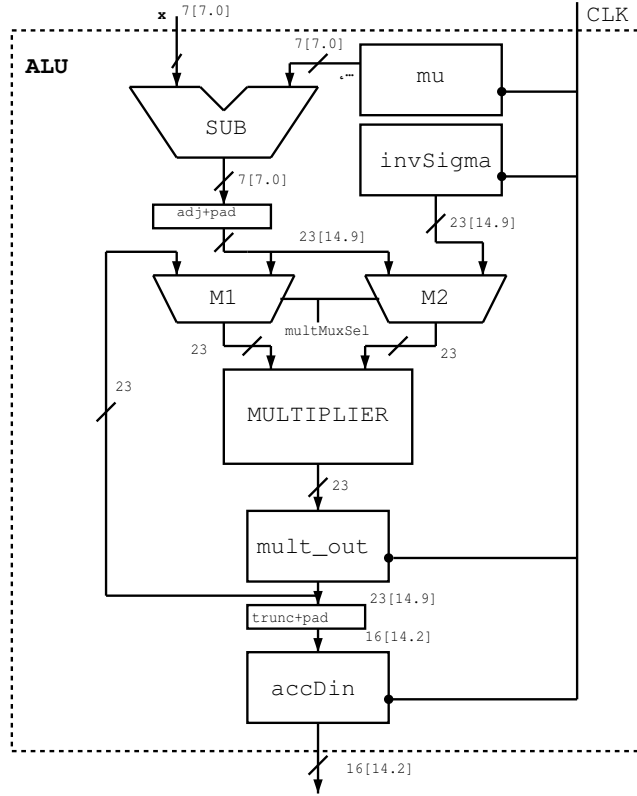


Figure 5.6: Arithmetic Logic Unit architecture

part resulted crucial; the system required a precision of four decimal digits to keep the classification success unchanged, that can be represented with nine fractional bits. Furthermore, it is known that multiplying two N-Bits number the result would be represented with 2N bits in order to avoid overflow. In this case the parallelism of the multiplier input is set equal to the output and this is due to the following fact:

The input of the multiplier is a 23 bits-fractional binary number with 14 integer bits and 9 fractional bits, hence, their multiplication yields a 46 bits-output:

$$23[14.9] \cdot 23[14.9] = 46[28.18]$$

Since it is sure that the numbers can be represented in 23 bits without any overflow, it is possible to keep the firsts 14 integer bits and the firsts 9 fractional

bits from the decimal point. It is possible to look at the VHDL description below:

```

entity multiplier is
2 generic(N      : integer := 16);
  port (a        : in signed(N-1 downto 0);
4         b        : in signed(N-1 downto 0);
        mult_out: out signed(N-1 downto 0));
6 end multiplier;
  architecture behaviour of multiplier is
8 signal mult_out_full: signed(2*N-1 downto 0);
  begin
10 mult_out_full <= (a * b);
    multOut      <= multOut_full(31 downto 9);
12 end behaviour;

```

ELPF Unit

The next block in the system is the *Exponential Linear Piecewise Function Unit*. As deeply described in the previous chapter, its purpose is to approximate the exponential using segments (5.7).

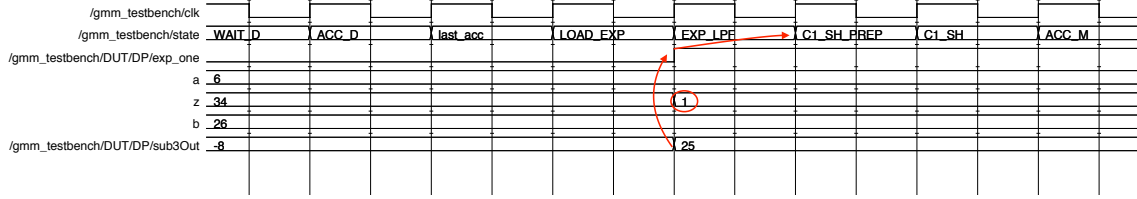
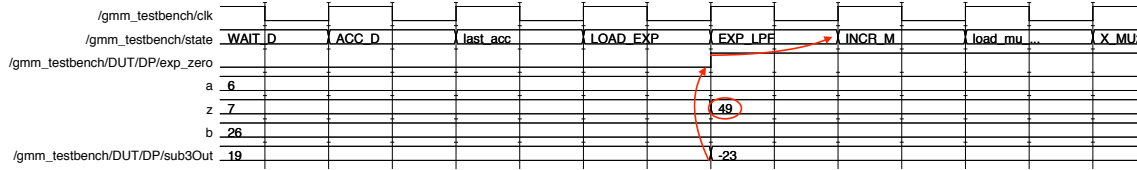
The two subtractors are used in parallel in order to assign the proper output depending on where the input $z_{f,s,m}$ located, as according the following equation.

$$ELPF(z_{f,s,m}) = \begin{cases} 1, & \text{when } z < a \\ m1(z - b), & \text{when } a \leq z < b \\ 0, & \text{when } z \geq b \end{cases} \quad (5.1)$$

It is possible to show that a comparison between two numbers A and B can be implemented by performing the subtraction $S = A - B$ and by looking at the produced *carry out* C_{out} and at the *difference* S :

- if $A \geq B$ then $C_{out} = 1$
- if $A < B$ then $C_{out} = 0$
- if $A = B$ then $C_{out} = 1$ and all the bits of S are 0.

Having this knowledge, it is possible to build the logic shown in 5.7 that discriminates the three cases. Note that the signal exp_m1 that is high when $a \leq z < b$, is the enable signal of the *ExpOut* register, that in this case would store the subtraction

Figure 5.9: Timing snapshot of the ELPF Unit in the case $z < a$ Figure 5.10: Timing snapshot of the ELPF Unit in the case $z \geq b$

Programmable Shifter

As previously mentioned, some of the multiplication parameters in 4.2 can be approximated to the closest power of two without having substantial changes in the classification performances of the system. In this way a complex and power consuming structure as the multiplier is, can be replaced with a programmable shifter. The shift operation itself is a resources-free operation since it corresponds to shift to the right or to the left the binary number. In our case the number of shifts to perform is not fixed, but varies depending on the different multiplication factors. For this reason, a multiplexing structure is needed to obtain the “programmability” that we need.

5.11 shows the programmable shifter structure. Since it was built with a behavioural description, its structure is established by the synthesizer, but we are only interested in its behaviour. It is a combinatorial structure that takes as external input the number of shifts to be performed. From the precision analysis treated in the software, it turned out that a range of 16 right and left shift is enough to cover every shift requirements. There is a difference between *logical* and *arithmetical* shifts: in the case of a right shift, both logical and arithmetical are executed in the same way, adding a zero on the as new LSB; in the case of a left shift instead, in the case of a logical shift a zero is added as new MSB, while in the arithmetical shift, the new MSB is a copy of the previous MSB, so that the sign (and so the arithmetical value)

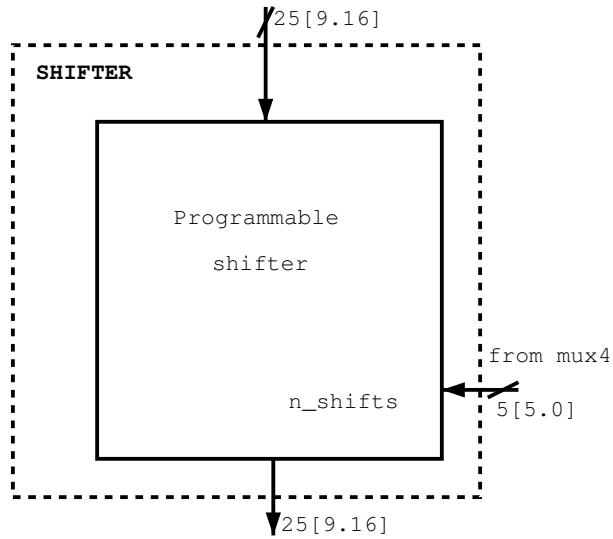


Figure 5.11: Programmable Shifter architecture

of the number is preserved. An example is shown in 5.12.

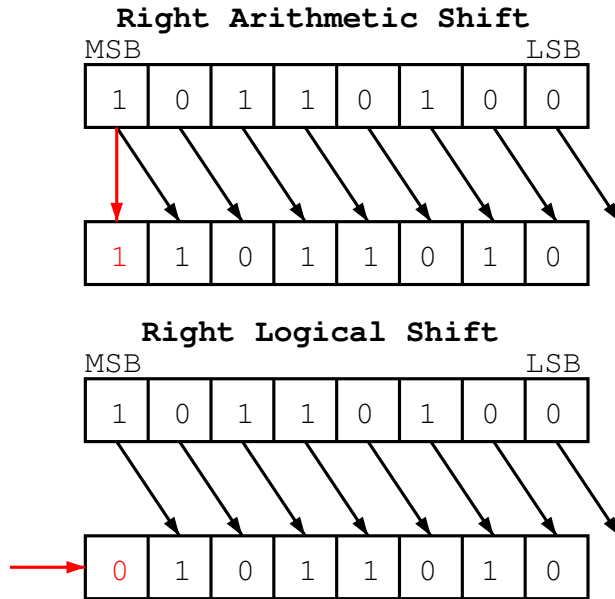


Figure 5.12: Arithmetic and Logic right shift example

Clearly, the shifter performs arithmetic shifts since we are interested in preserving the arithmetical value of the number.

LLPF Unit

The *Logarithmic Linear Piecewise Function Unit* is dedicated to the execution of the logarithm of the posterior probability. It is interesting to point out that making two separate units for the computation of the exponential and the logarithm is a trade off between the complexity of control unit and datapath: since the number of states is already high, merging the ELPF and LLPF unit would substantially increase the control over the execution of the two operations. For this reason we decided to implement the ELPF unit with two subtractors and the lpf unit with one subtractor, so that it is possible to understand on the ratio resources-complexity of the control. Moreover, the logarithm and the exponentials work on two different order of magnitude, therefore two separate units resulted a more natural implementation. From 5.13 we can observe the structure of the LLPF Unit. Since only one subtractor was used, the comparison with the LLPF parameters bl and cl cannot be executed in parallel, as in the case of the ELPF Unit, but has to happen in two different states, since we only have one control signal available (*log_lower*). This is possible thanks to the multiplexer *mux7* that provides different input to the subtractor. The comparison principle is exactly the same of the ELPF Unit with the only difference that, referring to 5.2, the second comparison is only done if necessary (refer to 5.2).

Speakers Shifting-Register File

The *Speakers Shifting-Register File* architecture comes from the need of having a structure that allows to access different registers periodically. As previously explained, the system processes the speech in real time, hence the probabilities of all the speakers are updated frame by frame separately. Using the structure presented in 5.14 it is possible to accumulate the probabilities of all the speakers separately, but using only one active register. The speakers' probabilities are not fixed to a register, but they shift so that they reach the active speaker register periodically. From the outside the system is not aware about this shifting mechanism, since the

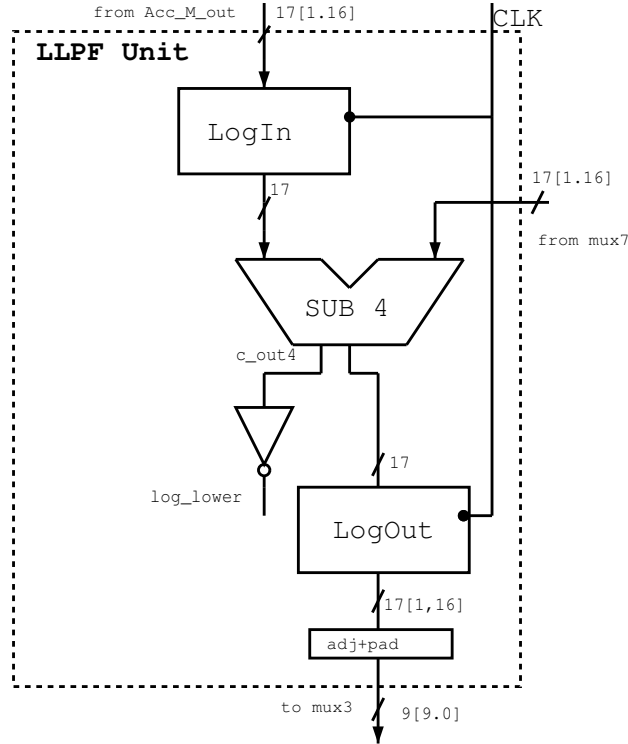


Figure 5.13: LLPF Unit architecture

accumulation is always done on the active speaker register. The shift is performed by switching the multiplexer to the right position and by enabling both the idle registers (*idleSP_EN*) and the *activeSP_EN* and it happens in the states *INCR_SP* and *INCR_FRAME*, so whenever a new speakers comes in the cycle or whenever a new frame has to be loaded and so the configuration has to come back to the initial one (Speaker 1 on the active speaker register).

WTA

The *Winner Takes All* circuit architecture (derived from [6]) sequentially compares the probabilities with each other and saves the speaker index corresponding to the maximum probability. 5.15 shows the WTA architecture. Using the same comparison technique employed in the LPF units, the content of *reg_A* is compared with the content of *reg_B*, and if *B* is greater than *A*, meaning that the current probability

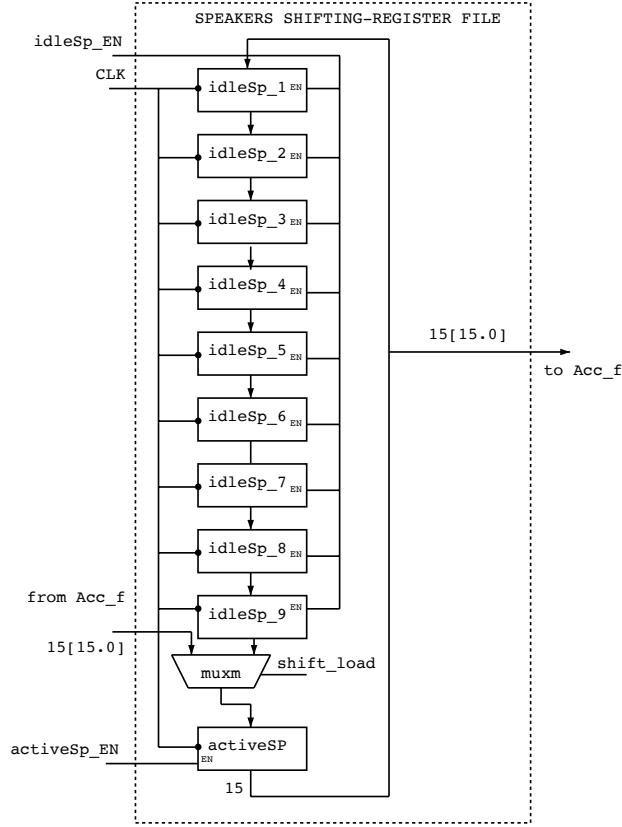


Figure 5.14: Speakers Shifting-Register File

is larger than the previous one, then *reg_A* is loaded with the new probability value and the winner register is loaded with the speaker number corresponding to that probability.

5.1.4 Testbench

A testbench was created in order to replace a I/O structure that reads the parameters and the MFCC data and provide them to the system. Referring to the Control Unit chapter, the testbench has its peak of activity in the loading stage of the workflow. As we can see from 5.16, three buses were used to send the data from the testbench to the datapath. The .txt files were generated by the Matlab code as a result of the training phase. We can notice that the file *x.txt* in an complete identification

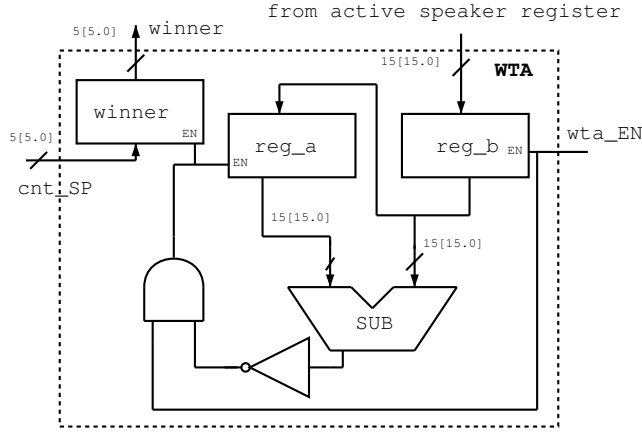


Figure 5.15: Winner Takes All Circuit

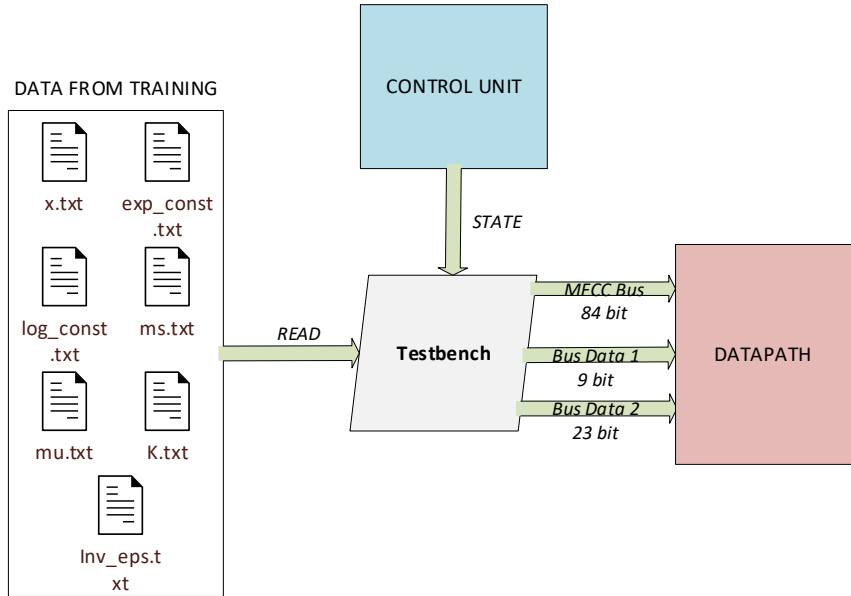


Figure 5.16: Testbench structure

system should be the real-time output of the MFCC extractor, that is not designed in this work. The files are structured as data-vector of a fixed parallelism. From table 5.1, we can understand the content of the files and also the dimensions demand

that their storage would require. It is important to understand the very low storage demand overhead that we are introducing with the LPF units, since the storage of the extra parameters only requires 10.5 Byte in addition to what we already needed to store, which is approximately 7KB.

File	Data	# elements	Parallelism (bit)	Storage (Byte)
exp_const.txt	a,b	2	9	2.25
log_const.txt	al,bl,bls	3	16	6
ms.txt	m1,ml2,ml2	3	6	2.25
mu.txt	$\mu_{s,m}$	3600(30 x 12 x 10)	7	3150
inv_eps.txt	$\Sigma_{s,m}^{-1}$	3600(30 x 12 x 10)	9	4050
K.txt	$K_{s,m}$	300 (30 x 10)	6	225
x.txt	x_f	6000 (12 x 500)	7	/
				7435.5

Table 5.1: DATA ANALYSIS

In the following table we can understand the data distribution over the different buses:

State	Bus data 1	Bus data 2	MFCC Bus
LD_M1LC	m1	cl	/
LD_ML2BL	ml2	bl	/
LD_ML1BLS	ml1	bls	/
LOAD_FRAME	/	/	$x_{f,d=1}, \dots, x_{f,d=12}$
LD_MU_EPS_X	mu	inv_eps	/
EXP_LPF	K	/	/

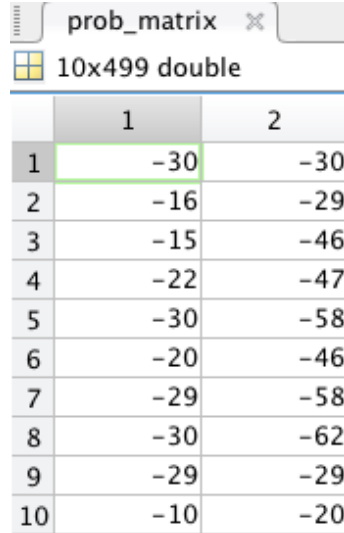
Table 5.2: BUSES DATA DISTRIBUTION

The Bus Data 2 parallelism is 23 bits because the variable inv_eps is stored with a 16 bit zero-padding that is needed in the datapath, the reader may refer to 5.3. The MFCC Bus is 84 bits wide so that a parallel load of twelve 7 bits-mfcc components is possible.

5.2 Results

5.2.1 Simulation

The software *Altera Modelsim* was used for the testing phase. The purpose of the testing process is to debug the system and bring it to its correct functioning. After a debugging stage, where parallelism and logic errors were adjusted, the correct working principle of the system was verified. As a debug technique, the Matlab code was used as a reference for the expected results of every partial operations. A good starting point to test most of the hardware, is to analyze the computation of the probabilities of the first two frames, so that the simulation time is not too large and a complete control unit cycle is performed. In 5.17 the expected output for both frames is reported, while in 5.18 and 5.19 the postscripts of the actual outputs can be observed (the first speaker is marked with the signal *int10*).



A screenshot of a Matlab variable viewer window titled 'prob_matrix'. It shows a 10x499 double matrix. The first two columns are labeled '1' and '2'. The first 10 rows are displayed, with values ranging from -30 to -20.

	1	2
1	-30	-30
2	-16	-29
3	-15	-46
4	-22	-47
5	-30	-58
6	-20	-46
7	-29	-58
8	-30	-62
9	-29	-29
10	-10	-20

Figure 5.17: Matlab snapshot of the first two frames speakers' probabilities

We can see that the values of the Matlab-implemented system (5.17) are very close to the result that the designed hardware provides (5.18 and 5.19). They are not identical since the Matlab code works in floating points and with a similar precision but not exactly the same as the hardware. Now that we know that the first two frames give a reasonable result, the system is tested for 400 frames. In 5.20 the speakers' probabilities at each frame are plotted: the input phrase comes from

/gmm_testbench/DUT/DP/RF/clock							
/gmm_testbench/DUT/DP/M_cnt	29						0
/gmm_testbench/DUT/DP/d_cnt	11						0
/gmm_testbench/DUT/DP/fr_cnt	0						1
/gmm_testbench/state	INCR_FR						load_frame
/gmm_testbench/DUT/DP/RF/int1	-28						-9
/gmm_testbench/DUT/DP/RF/int2	-31						-28
/gmm_testbench/DUT/DP/RF/int3	-30						-31
/gmm_testbench/DUT/DP/RF/int4	-20						-30
/gmm_testbench/DUT/DP/RF/int5	-30						-20
/gmm_testbench/DUT/DP/RF/int6	-22						-30
/gmm_testbench/DUT/DP/RF/int7	-16						-22
/gmm_testbench/DUT/DP/RF/int8	-16						
/gmm_testbench/DUT/DP/RF/int9	-30						-16
/gmm_testbench/DUT/DP/RF/int10	-30						-16

Figure 5.18: Hardware probability values (first frame)

/gmm_testbench/DUT/DP/M_cnt	29						0
/gmm_testbench/DUT/DP/d_cnt	11						0
/gmm_testbench/DUT/DP/fr_cnt	1						2
/gmm_testbench/state	INCR_FR						load_frame
/gmm_testbench/DUT/DP/RF/int1	-28						-18
/gmm_testbench/DUT/DP/RF/int2	-62						-28
/gmm_testbench/DUT/DP/RF/int3	-60						-62
/gmm_testbench/DUT/DP/RF/int4	-46						-60
/gmm_testbench/DUT/DP/RF/int5	-58						-46
/gmm_testbench/DUT/DP/RF/int6	-43						-58
/gmm_testbench/DUT/DP/RF/int7	-47						-43
/gmm_testbench/DUT/DP/RF/int8	-30						-47
/gmm_testbench/DUT/DP/RF/int9	-30						
/gmm_testbench/DUT/DP/RF/int10	-30						

Figure 5.19: Hardware probability values (second frame)

SPEAKER 2 from *DR4* in the *TEST* directory of TIMIT database. It is interesting to observe the relationship between the different speakers probabilities: in this case we can see that the speaker number two is the winner from, approximately the 300th frame, which means that in this case the identification process could have been stopped before the end of the entire speech sample. This is one example of a possible *frames-level* optimization that this type of design allows. Moreover, the probability values are negative, since they are actually the logarithm of the

probabilities, so the logarithm of a value between zero and one.

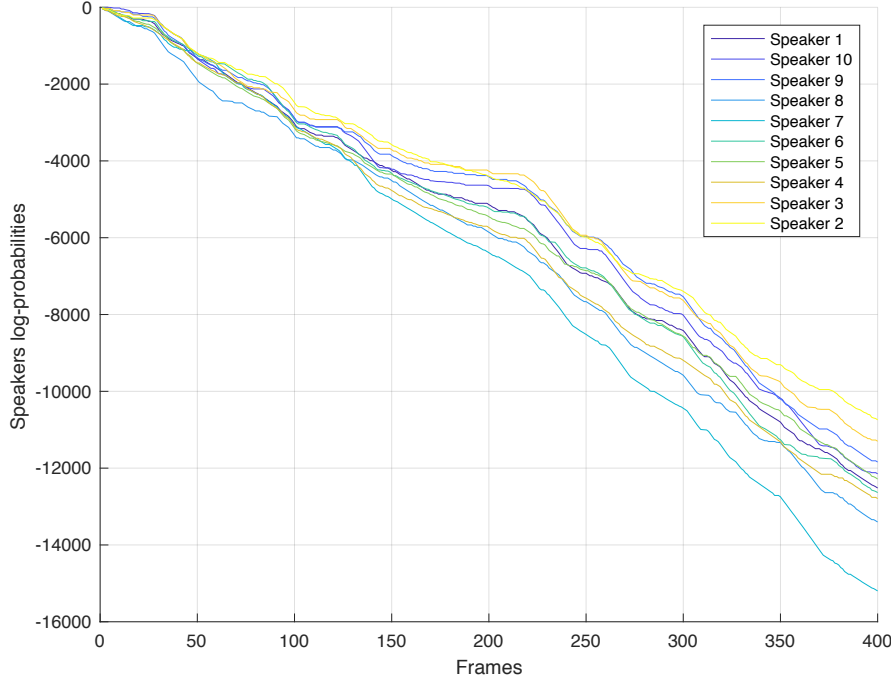


Figure 5.20: Speakers' probabilities over 400 frames (expected winned: Speaker 2)

5.21 a comparison between the software and the hardware results of the same samples against the same set of speakers is illustrated. We can note how, in this case, the hardware probabilities are more widely spaced between each other, probably due to the bit quantizations. The relative error between the two outputs with respect to the software values is illustrated in 5.22. In the bottom figure we can appreciate how the average error between the hardware and the software tends to zero.

5.2.2 Synthesis

The final section of this work consists in the actual synthesis of the circuit. Since this is a ASIC implementation, all the components are mapped to a standard library. In this case the design is synthesized using the open-source 45 nm library *gscl45nm* using the *Encounter RTL Compiler* by *Cadence*. In order to reduce the power, a very convenient technique that we can use is to reduce as much as we can the clock frequency, since the dynamic power linearly depends with the clock frequency (1.1),

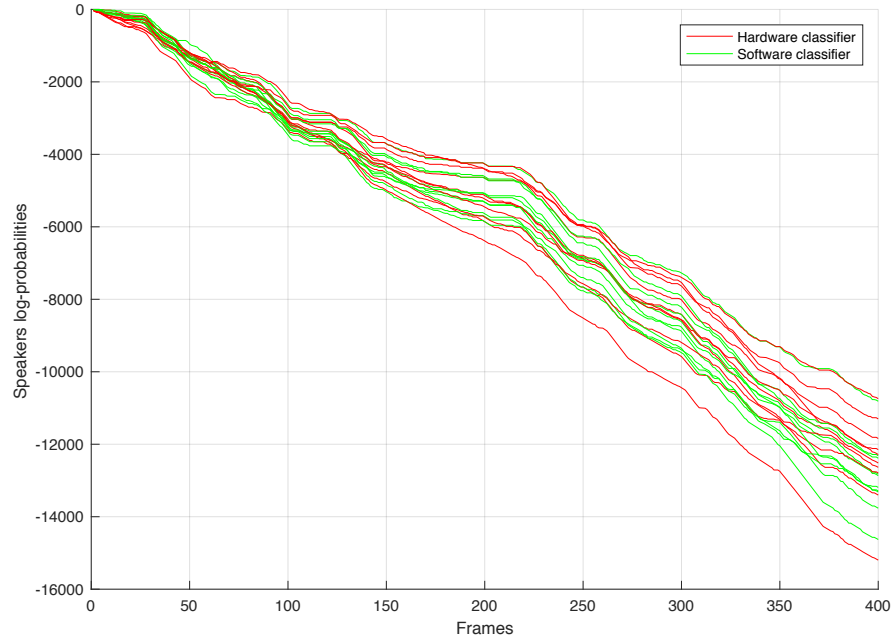


Figure 5.21: Comparison between Software and Hardware-generated probabilities

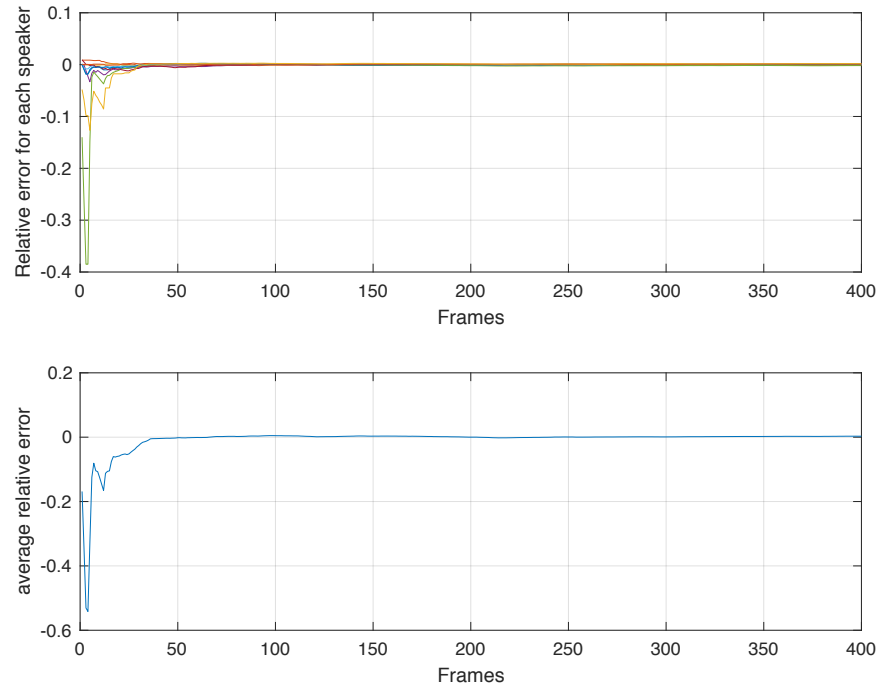


Figure 5.22: Relative error of each speakers (top), average relative error (bottom)

as explained in the introductory chapter.

For this reason, it is useful to make some considerations about the timing constraints that the operations imposes in order to understand how low the clock frequency can be: the system processes the input speech real-time, breaking it in 20 ms frames with a 10 ms overlapping (3.3). Hence, we can suppose to have a new frame every 10 ms (without considering the delay that the MFCC extractor would introduce). Therefore each frame has to be processed in approximately less than 10 ms. Referring to the control unit, for each frame five clock cycles are needed for the loading part, six for the $z_{f,s,m}$ computation, six for the logarithm computation and finally ten clock cycles for the WTA stage. Therefore we have a total of

$$5 + (((6 \cdot 12) + 6) \cdot 30) + 6 \cdot 10 + 11 = 23476$$

clock cycles for each frames, to be run in 10 ms hence the minimum frequency required is

$$f_{min} = \frac{23476}{0.01} = 2.35 \text{ MHz}$$

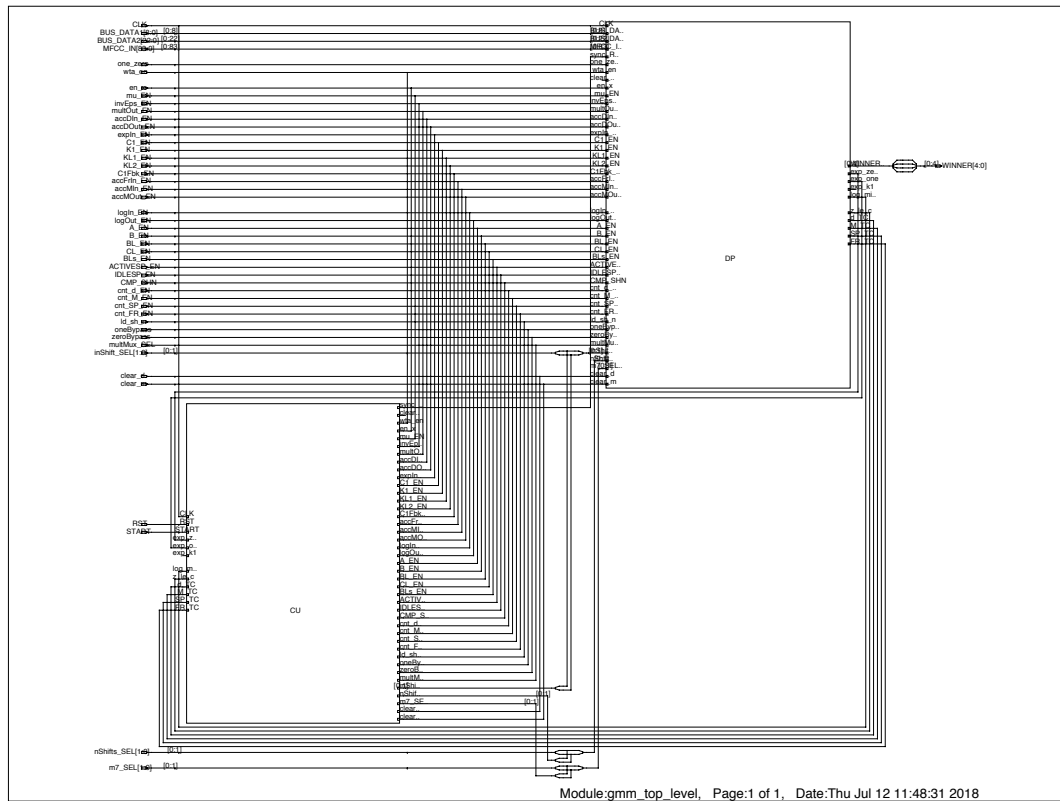
Doing this analysis we know that approximately we cannot go slower that 2.5 MHz.

The VHDL code was adjusted to make it compatible with the synthesizer. We shall remember that the Testbench was only created for testing the system, but it does not represent an actual digital device that the synthesizer can build. A clock frequency of 10MHz was chosen since the timing evaluation that has just been done only refers to a portion of the entire system, so the actual minimum frequency of the system might result higher than 2.5 MHz. Choosing 10 Mhz, the dynamic power would be sacrificed to give more timing robustness. The design was synthesized and the results are presented below:

The total power of the system at $f_{clk} = 10 \text{ Mhz}$ is:

Cells:	5121
Leakage Power:	124.68 μW
Dynamic Power:	477408.896 μW
Total Power:	602.1 μW

We can see that, even if the clock frequency is much higher than the estimated minimum frequency, the power is already substantially lower than one mW, thanks to the design choices that were employed. This means that without applying any synthesis-level low power techniques such as power gating or clock gating, and providing a clock frequency that can be potentially lower, the system dissipates around 0.6 mW of total power. It is necessary, however, to make the following considerations: since we are not providing any information about the input data, the synthesizer is not aware about the switching activities of the inputs. This causes the dynamic power estimation not to be precise at all, since the synthesizer, by default, considers that all the primary inputs have a 50% switching activity, that for our system is clearly a worst case scenario. For this reason all the control signals that the Control Unit generates, are set as primary inputs in the top-level design, so that the system can attribute a 50% switching activity to all the signals (5.23). This corresponds in



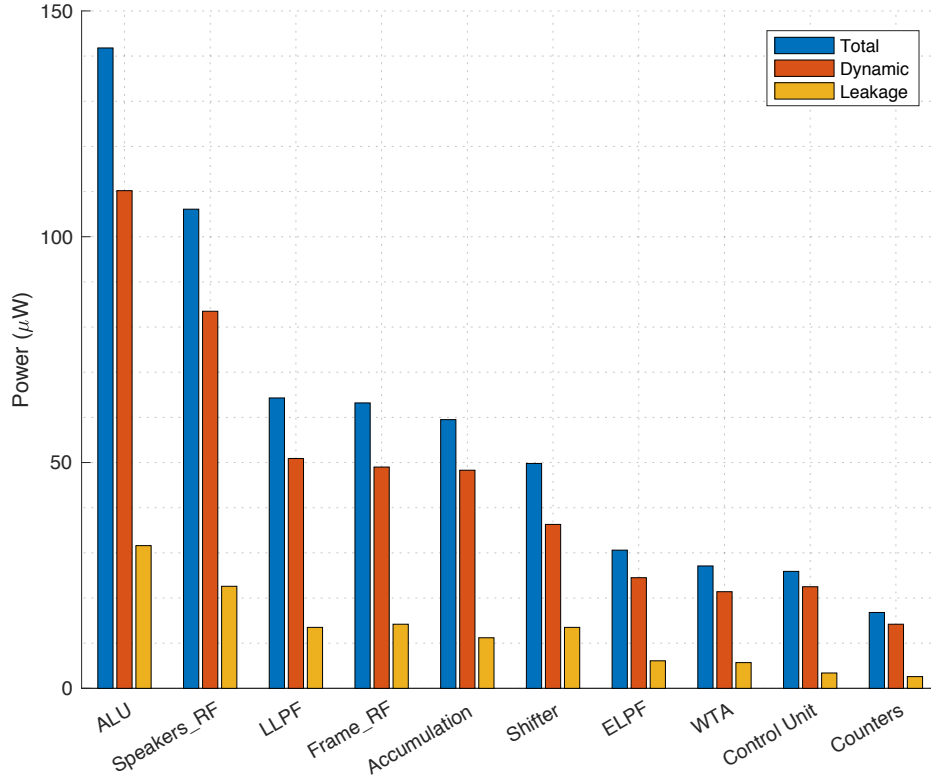
Cadence Schematics, Copyright 1997-2006

Figure 5.23: Snapshot of the top view of the system

having a datapath where all the components are active in every state, but clearly, each state only uses the portion of the datapath usefull to accomplish the task of the state. Moreover, some of the states might be not executed at all thanks to conditional branching optimizations introduced in the control unit, hence we have to be aware that the dynamic power estimation is not realistic, but represents an upper bound. This is enough for the purpose of this work, since the constraint of one mW is met also working in a worst case scenario. However, in order to have a proper and significant power estimation, the switching activity of the inputs should be stored in a SAIF file in the simulation stage, that should be given as input to the synthesizer. In this way, the synthesizer is aware of the switching activities of the inputs, and the resulting power estimation would be accurate. However, this process requires time and good knowledge of the synthesizer, and it is not crucial at this stage of the design, since the memory and the MFCC feature extractor are yet to be designed and the classifier still have optimization dynamics. In 5.24 the power consumption of all the units is reported. The system is divided in groups that does not necessary corresponds to the units presented in 5.3, but their composition is illustrated in 5.3.

It is possible to observe that the ALU has the higher power demand, which is reasonable since the multiplier is known to be a very expensive structure. In addition, we can see in 5.3 that the consumption of the multiplier is more that seven times higher than the shifter, and this is a substantial proof that validate the choice of replacing the multiplier with a programmable shifter. It is also interesting to note the power difference between the *LLPF* and the *ELPF* Units: the computation of the logarithm uses an higher set of complexity but one subtractor unit instead of two, and it is still more consuming than the exponential unit. This is also mainly because the *LLPF* operates on an higher parallelism with respect the *ELPF* unit, and this is why the its leakage power is almost the double of the *ELPF* one.

An other important result of the synthesis, is the *Timing Slack*. The slack time can be defined as the amount of time an operation can be delayed without causing the next operation to be delayed. In other words, having a negative Timing Slack means that the clock frequency imposed is too high for the system to operate without timing violations. In an optimized system, the slack time should be a positive number close to zero, meaning that the operations are properly distributed

Figure 5.24: System power consumption for $f_{clk} = 10$ Mhz

along the clock cycle. Having a positive slack means that the system is running faster than what required, so the operation is ended in advance of the clock edge. From an design point of view the positive slack represent an optimization room: if the design is performace-oriented, the system can run on an higher clock frequency, and so the troughput of the system would be higher; if, as in this case, the design is low power-oriented, the system can be optimized from a power perspective, for example operating at a lower voltage supply or with higher threshold voltages that, as explained in the introduction chapter, slows the system and reduces the leakage power.

With a frequency of 10 Mhz, the system resulted in a positive slack,

$$\text{Timing slack} : 92420 \text{ ps}$$

This result is not surprising, since 10 Mhz is still a fairly low frequency for a 45 nm

technology and it gives even more optimization dynamics and space for low power techniques that can decrease even more the power consumption of the classifier.

Finally, the cell area of the circuit is

$$\text{Total Area: } 18926 \mu m^2$$

But this data does not take into account the net area, that usually depends on the type of the interconnections and the wire-load.

5.2 – Results

Group	Component	Cells	Leakage (nW)	Dynamic (nW)	Total (nW)
ALU	mult	1364	22664	55311	77975
	sub1	15	406	2192	2598
	regMU	28	1041	3698	4738
	m1	40	334,4	6425,704	6760,104
	m2	40	334,4	3503,921	3838,32
	regMult_out	92	3419	27162	30.581
	regINV_SIGMA	92	3419,146	11950,501	15369,647
ELPF	regExpIn	36	1337,927	5033,64	6371,566
	reg_exp_out	36	1337,927	4625,235	5963,162
	rega	36	1337,927	5679,711	7017,638
	sub2	21	317,633	1535,53	1853,163
	regb	36	1337,927	5851,206	7189,133
	sub3	20	389,992	1803,226	2193,218
LLPF	reg_log_in	68	2527,195	9503,441	12030,636
	reg_log_out	68	2527,195	8160,325	10687,52
	reg_bls	64	2378,536	9604,838	11983,374
	reg_bl	64	2378,536	9997,618	12376,154
	reg_cl	64	2378,536	8795,111	11173,647
	sub4	36	720,38	3474,29	4194,67
	m7	69	567,722	1320,568	1888,289
SHIFT.	sh	414	4471,912	5807,938	10279,851
	reg_K_fbk	100	3716,463	13666,34	17382,804
	reg_K	24	891,951	3082,912	3974,863
	reg_m1	24	891,951	4066,014	4957,965
	reg_ml1	24	891,951	3644,103	4536,054
	reg_ml2	24	891,951	4311,796	5203,747
	m4	47	493,891	544,807	1038,698
	m6	26	390,027	334,011	724,038
	m3	90	897,316	807,004	1704,32
ACC.	reg_Acc_m_out	68	2527,195	9203,379	11730,574
	reg_Acc_m_In	68	2527,195	9217,854	11745,049
	regAcc_d_in	40	1486,585	10099,204	11585,789
	regAcc_d_out	40	1486,585	7332,638	8819,223
	acc_m	17	674,931	2303,888	2978,819
	reg_accFr_In	36	1337,927	5480,957	6818,884
	m5	10	142,195	314,021	456,215
	acc_fr	15	596,51	1961,615	2558,126
	acc_d	10	400,458	2370,673	2771,132
COUNT.	fr_counter	74	1052,47	5907,655	6960,125
	M_counter	45	631,089	2934,437	3565,526
	sp_counter	35	488,682	2741,477	3230,16
	d_counter	29	436,995	2612,965	3049,96
SP RF	RF	630	22603,158	83484,701	106087,859
FR RF	shregX	504	14191,839	48980,307	63172,146
WTA	wta_block	168	5660,551	21387,957	27048,508
CU	CU	239	3427,045	22486,682	25913,727

Table 5.3: DETAILED POWER REPORT AT $F_{CLK} = 10MHz$

Chapter 6

Conclusion

6.1 Achievements

In this work a complete software-to-hardware design workflow of a low power LUT free GMM-based speaker classifier was presented. The main achievements of this work are:

- Elimination of any memory access for the computation of the exponential and the logarithm in the hardware system.
- Application of algorithm-level power optimization techniques.
- Frame-level optimization “friendly ” design.
- Fulfillment of power constraint: the system requires 0.6 mW at $f_{clk} = 10\text{ MHz}$ with further optimization margin due to a positive slack time.
- Solid and working hardware foundation for future works.

6.2 Future goals

As discussed in the document, this design was though to be a solid starting point to obtain a fully optimized speaker identification system. The following works can be realized starting from this design:

- Design of the storage unit for the GMM parameters.
- Design of the hardware MFCC features Extractor.

- Datapath restructuring and optimization: Include the parameters memory and study of possible interconnection-level power optimization such as bus-encoding, between the memory and the Datapath.
- Accurate power estimation with SAIF back-annotation technique.
- Application of frame-level optimization, such as Frame-skipping.
- Application of low power technique such as power gating and clock gating to further reduce the power exploiting the positive slack.

Bibliography

- [1] Haizhou Li Tomi Kinnunen. An overview of text-independent speaker recognition: From features to supervectors. *Speech Communications*, 52(2010):12–40, August 2009.
- [2] Jay Wilpon Pradeep K. Bansal Lee Begeja Carroll W. Creswell Jeffrey Farah Benjamin J. Stern. ”digital signatures for communications using text-independent speaker verification”, Jun 2014.
- [3] Amit Ranjan Trivedi and Ahish Shylendra. Ultralow power acoustic feature-scoring using gaussian i-v transistors.
- [4] Douglas A. Reynolds. *A Gaussian Mixture Modeling Approach to Text-Independent Speaker Identification*. PhD thesis, Georgia Institute of Technology, Aug 1992.
- [5] Lori F. Lamel John S. Garofolo et al. Timit documentation. <https://catalog.ldc.upenn.edu/docs/LDC93S1/>, 1990. Online; accessed May 2018.
- [6] Minghua Shi and Amine Bermak. An efficient digital vlsi implementation of gaussian mixture models-based classifier. *IEEE TRANSACTIONS ON VERY LARGE SCALE INTEGRATION (VLSI) SYSTEMS*, 14(9):962–974, September 2006.
- [7] Timothy Allen Phaklen EhKan and Steven F. Quigley. Fpga implementation for gmm-based speaker identification. *International Journal of Reconfigurable Computing*, 2011(420369):1–8, November 2010.