Circuits and Systems
Mekelweg 4,
2628 CD Delft
The Netherlands
http://ens.ewi.tudelft.nl/

CAS-2021-08

M.Sc. Thesis

Efficient implementation of an audio preprocessing algorithm for SNN keyword spotting

S.A. Hijlkema

Abstract

Mobile devices are getting increasingly powerful, becoming compatible for an ever increasing set of functionality. Applications based around neural networks however still have to offload parts of their computations to the cloud since current Artificial Neural Networks (ANNs) are still too computationally expensive for any practical standalone use in energy constrained mobile devices. Developments in the next generation of ANN: Spiking Neural Network (SNN), are expected to bring neural networks directly to the edge. Even though SNNs are becoming a reality, they can not (yet) effectively operate on raw sensory input data. For this, a preprocessing algorithm can be used to extract low-level features in an efficient way to boost the neural network efficiency. A parallel can be found in biology with the cochlea that, for audio, provides preprocessing for the brain. Recent research has shown that an SNN is capable of reaching high classification accuracy when combined with an biologically plausible audio preprocessing stage. To be of interest for edge-computing it however also needs to be area and energy efficient. This thesis will provide the first steps in researching the optimal configuration of a specific audio preprocessing algorithm by mapping its current software simulation to embedded hardware. For this purpose the software simulation is analyzed and an efficient hardware implementation is designed. For evaluation a prototype, and its hardware constrained simulation, is developed and optimized.

Faculty of Electrical Engineering, Mathematics and Computer Science
Delft University of Technology

Efficient implementation of an audio preprocessing algorithm for SNN keyword spotting

Thesis

submitted in partial fulfillment of the
requirements for the degree of

Master of Science

in

Computer Engineering

and

Embedded Systems

by

S.A. Hijlkema
born in It Hearrenfean, The Netherlands

This work was performed in:

Circuits and Systems Group
Department of Microelectronics & Computer Engineering
Faculty of Electrical Engineering, Mathematics and Computer Science
Delft University of Technology

Delft University of Technology
Department of
Microelectronics & Computer Engineering

The undersigned hereby certify that they have read and recommend to the Faculty of Electrical Engineering, Mathematics and Computer Science for acceptance a thesis entitled **"Efficient implementation of an audio preprocessing algorithm for SNN keyword spotting"** by **S.A. Hijlkema** in partial fulfillment of the requirements for the degree of **Master of Science**.

Dated: 25th of August 2021

Chairman:

prof.dr.ir. T.G.R.M van Leuken

Advisor:

prof.dr.ir. T.G.R.M van Leuken

Committee Members:

prof.dr.ir. J.S.S.M. Wong

dr. A. Zjajo

Abstract

Mobile devices are getting increasingly powerful, becoming compatible for an ever increasing set of functionality. Applications based around neural networks however still have to offload parts of their computations to the cloud since current Artificial Neural Networks (ANNs) are still too computationally expensive for any practical standalone use in energy constrained mobile devices. Developments in the next generation of ANN: Spiking Neural Network (SNN), are expected to bring neural networks directly to the edge. Even though SNNs are becoming a reality, they can not (yet) effectively operate on raw sensory input data. For this, a preprocessing algorithm can be used to extract low-level features in an efficient way to boost the neural network efficiency. A parallel can be found in biology with the cochlea that, for audio, provides preprocessing for the brain. Recent research has shown that an SNN is capable of reaching high classification accuracy when combined with an biologically plausible audio preprocessing stage. To be of interest for edge-computing it however also needs to be area and energy efficient. This thesis will provide the first steps in researching the optimal configuration of a specific audio preprocessing algorithm by mapping its current software simulation to embedded hardware. For this purpose the software simulation is analyzed and an efficient hardware implementation is designed. For evaluation a prototype, and its hardware constrained simulation, is developed and optimized.

Acknowledgments

First I would like to thank dr.ir. T.G.R.M. van Leuken in directing the general project outline. Secondly, I would like to thank dr.ir. C. Lo for guiding me during the first stage of this thesis project, related to research and design. Also, I would like to thank MSc. A. Viteri for providing assistance through the second stage of this thesis project, implementing and validating the prototype.

Moreover, I want to thank my colleagues for their open support each time I was struggling again to solve an issue.

Lastly, I want to thank my fellow master students for the encouraging working environment.

S.A. Hijlkema
Delft, The Netherlands
25th of August 2021

Contents

List of Acronyms

ASR	Automatic Speech Recognition	1
AXI	Advanced eXtensible Interface	60
ASIC	Application Specific Integrated Circuit	67
AI	Artificial Intelligence	1
ANN	Artificial Neural Network	1
BSSP	Bit-Stream Signal Processing	46
CLA	Carry-Look Ahead	72
CIC	Cascaded Integrator-Comb	xvii
CPU	Central Processing Unit	xix
DSE	Design Space Exploration	70
DMA	Direct Memory Access	xxi
DFT	Discrete Fourier Transform	xxi
FFT	Fast Fourier Transform	44
FPGA	Field Programmable Gate Array	xix
FIFO	First-In First-Out	60
FIR	Finite Impulse Response	xviii

FSDD	Free Spoken Digit Dataset	9
HDL	Hardware Description Language	35
IIR	Infinite Impulse Response	xviii
IP	Intellectual property	60
JSON	JavaScript Object Notation	xvii
LIF	Leaky Integrate Fire	21
LTD	long-term depression	xviii
LTP	long-term potentiation	xviii
MEMS	Micro Electro-Mechanical System	14
p-SFE	population Step Forward Encoding	xviii
p-TE	population Threshold Encoding	xviii
PSP	postsynaptic potential	56
PDM	Pulse Density Modulation	xvii
PCM	Pule Code Modulation	xvii
RTOS	Real-Time Operating System	63
SOM	Self-Organizing Map	9
SOS	Second-Order Sections	41

List of Figures

List of Tables

Introduction 1

Humans are skilled in communicating by speech. Most interactions with computers nowadays are however mostly based on textual input. There are several areas of computer interaction that can benefit from Automatic Speech Recognition (ASR) as a method of input. Examples can already be found in the so called Voice-Activated Personal Assitant (VAPA) devices, e.g. "Siri" from Apple or "Alexa" from Amazon. In which the VAPA can detect simple commands and act upon them. Since humans are skilled in speech recognition, it is often opted to use Artificial Intelligence (AI) for ASR. In particular the Artificial Neural Network (ANN) is a popular choice to attempt to mimic the general computational processing concepts of the brain. In order to solve ever increasing complex problems with ANNs, techniques like deep learning are becoming popular. With deep learning, multiple layers of ANNs are used in succession to extract higher-level features progressively. To perform ASR with ANN, a large network with many layers will be required. Moreover, these ANNs will require vast amounts of sample data to train and are computationally expensive to execute. Current ASR systems like VAPA are offloading the computationally expensive parts, i.e. the ANN, to cloud-computing. Essentially sending most of the voice data over the internet to servers, whereby the VAPA device is an interface to those cloud-computing servers.

Since in recent years the ASR devices are getting more common and popular there is a desire to reduce the network bandwidth requirements for the edge devices, e.g. smartphones. A proposed solution is to let the edge device perform all the computations on its own with dedicated hardware, without relying on cloud-computing.
To achieve this, while staying within strict device constraints, a more efficient implementation of AI will be required. A candidate for this is the so called Spiking Neural Network (SNN). With SNN the concept of time is introduced to neural networks, whereby the input is encoded in temporal information and nodes in the network are event-driven.

1.1 Motivation

Although the SNN can reduce compute complexity for the neural network, providing raw sensor input data to the network will not result in an efficient ASR. Looking at biological speech recognition it is seen that sound is first encoded by the cochlea and then send to the brain for further processing. To mimic the highly effective and efficient biological speech recognition, a preprocessing step is deemed beneficial where the extraction of the lowest-level features should be performed by a dedicated preprocessing module.

1.2 Problem definition

A software simulation for a specific audio preprocessing stage, aiming for SNN compatibility with keyword spotting, has already shown promising results [1], further evaluation is however still required to check for hardware compatibility. The aim is to perform all computations on an edge device, e.g. a smartphone, therefore all components including the preprocessing need to be energy and area efficient.

To determine suitable hardware and investigate trade-offs between energy, area and classification accuracy it is required to simulation and build a prototype.
The design of this prototype raises several research questions:

1.2.1 Research questions

- How to map the audio pipeline software to hardware in a structured approach?
- How to utilize the predetermined audio pipeline efficiently?
- What kind of hardware should be utilized for the audio preprocessing prototype?
- How does compute optimization affect SNN classification accuracy?

1.3 Objectives

The thesis objectives are the following:

- Create a structured approach: from software to hardware.
- Build a framework that can execute programs independently and interconnect their inputs and outputs.
- Separate the audio preprocessing pipeline into independent modules.
- Analyze and optimize each processing module separately.
- Choose appropriate hardware for the audio preprocessing pipeline prototype
- Implement and connect each module to create a prototype.
- Explore the design space of the prototype.

1.4 Contributions

This thesis makes several contributions:

- A framework that supports the chaining of several programs, written in different programming languages, together into one pipeline.
- Efficiency analysis of a PDM to PCM converter for audio signals in the 0 to $8kHz$ frequency range.

- Efficiency analysis between time and frequency filtering for a specific audio preprocessing pipeline.
- Functional hardware prototype of an audio preprocessing pipeline compatible with a SNN.
- Exploration of parameter influence on classification accuracy in an audio preprocessing pipeline.

1.5 Thesis outline

This work is organized in the following chapters:

Chapter 2 provides background information about neural networks and the work this thesis is based upon.
Chapter 3 illustrates the structured approach taken from hardware to software. Separating the design in modules and creating a program that can run them independently.
Chapters 4, 5, 6 and 7 contain the individual modules and their analysis.
Chapter 8 provides insight in how the modules can be interconnected and implemented on a Zybo device.
Chapter 9 evaluates the influence of several parameters on the classification accuracy and efficiency of the design.

Background 2

In this chapter a brief introduction to the network paradigms ANN and SNN is given. Their differences will influence the preprocessing which is mentioned afterwards.

2.1 Artificial Neural Networks

The Artificial Neural Network (ANN) is a popular tool nowadays. Tasks otherwise performed using hand-crafted feature extraction and decision rules can now be performed by a general purpose network. Moreover, it turned out that such a network is capable of obtaining higher performance than traditional handcrafted solutions. An example ANN is illustrated in Figure 2.1.

Three distinct layers are identified:

- An input layer that accepts inputs.
- One or multiple hidden layers, providing a connecting between nodes of different layers.
- Lastly the output layer which provides the output of the network.

Since the complete behavior of an ANN is not pre-programmed, but will depend on its training set, the same ANN structure can have different purposes.

A computational model was first proposed by Warren McCulloch and Walter Pitts in 1943 [22]. The model is loosely based upon biological neural networks, where the neurons and synapses are represented by nodes and edges respectively.
Although a practical training method for the network, backpropagation, was only developed in 1975 by Werbos in [23].

The networks can mathematically be described in matrices [24]. A simplified version is given below.

Neuron accumulation:

$$\vec{a}^{(l)} = W^{(l)} \times \vec{y}^{(l-1)} + \vec{b}^{(l)} \tag{2.1}$$

Where $a^{(l)}$ is the accumulation of all the inputs from the previous to the next layer.
Where $W^{(l)}$ is the matrix with weights of this layer.
Where $\vec{y}^{(l-1)}$ is the output vector of the previous layer.
Where $\vec{b}^{(l)}$ is the bias vector of this layer.

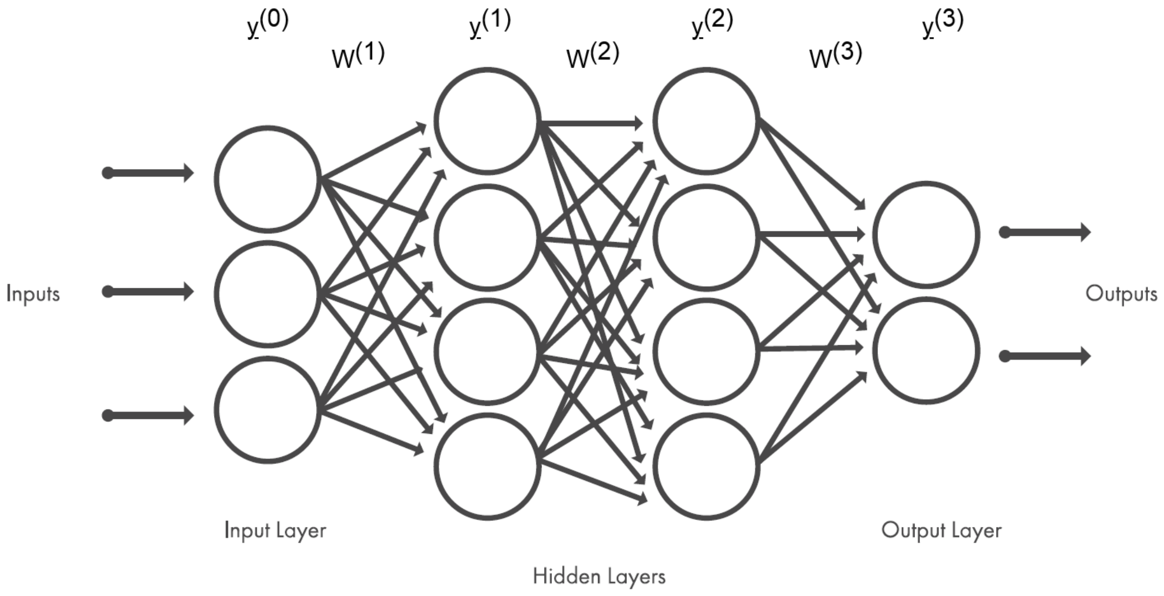


Figure 2.1: Illustration of an Artificial Neural Network, adapted from [1]

Neuron output:

$$\vec{y}^{(l)} = f(\vec{a}^{(l)}) \tag{2.2}$$

Where $\vec{y}^{(l)}$ is the output vector of the current layer.
Where f is the activation function, which is often a non-linear function, e.g. a Sigmoid.

Cost function example:

$$C = MSE = (\vec{y}^{(L)} - \vec{t})^2 \tag{2.3}$$

where $\vec{y}^{(l)}$ represents the output vector from the last layer
Where $\vec{t}$ is the target vector (label).

The cost function is a measure of the performance of the network. To improve performance, the derivative of the cost function is used to perform gradient descent. The cost of each previous layer can be found from its subsequent layer. As such error values, used for adapting weights, are propagated from its last to first layer.

After training the input-output relation is iteration-based and not continuous: the input-output relation is fixed regardless of the frequency or ordering of the input vectors. The ANN determines the classification probability of patterns depending on the output vector after the multi-bit input values are finished propagation through the network. Note that even though raw data can be directly applied to the input layer, preprocessing is often utilized to offload trivial feature extraction operations. This can reduce the complexity of the problem for the network and in turn enable it to obtain a higher classification accuracy.

2.2 Spiking Neural Networks

As mentioned before, the ANN is only loosely based upon biological neural networks. Most importantly, there is no concept of temporal information. Recent developments are aimed at Spiking Neural Networks (SNNs) which are more closely related to biological neural networks and their potential, also claimed to be ”the third generation artificial neural network” [25].

An illustration of a biological neuron and its components is provided in Figure 2.2. A

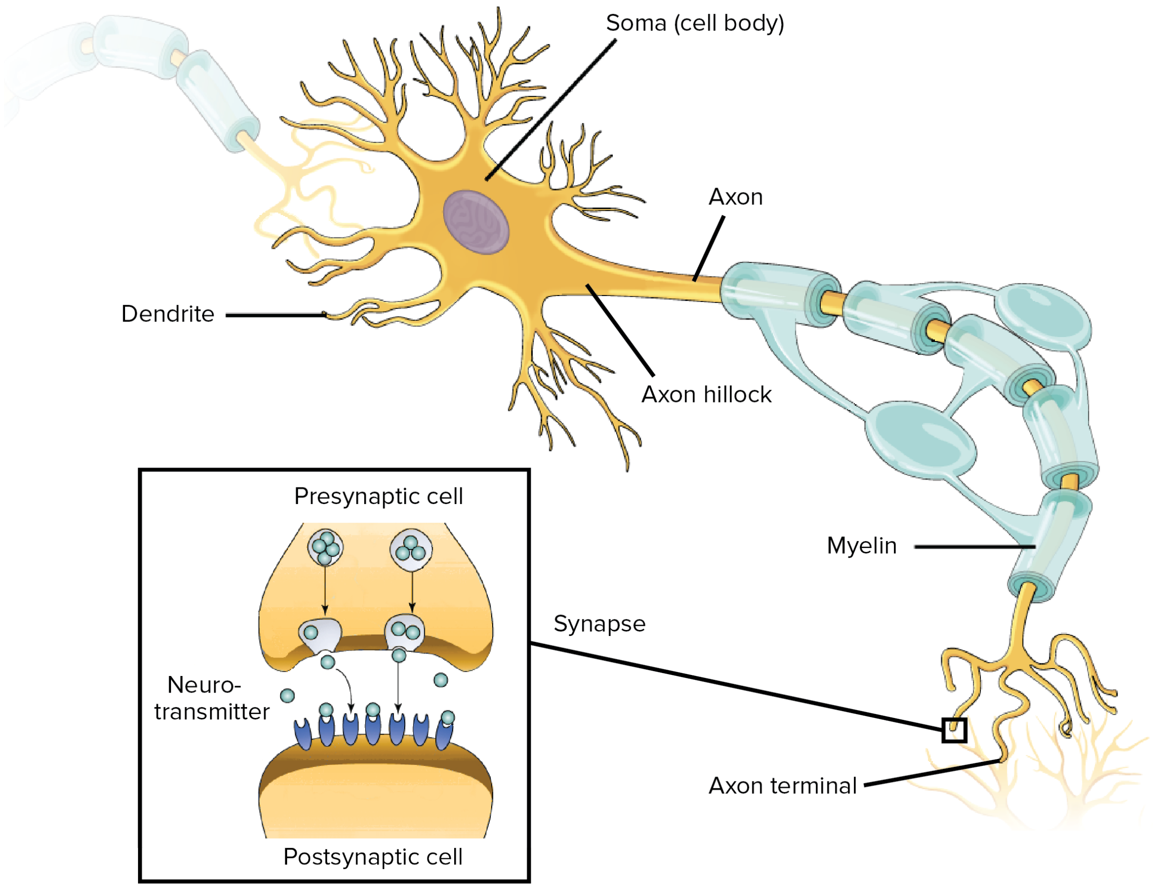


Figure 2.2: Overview of a neuron. [2]

human brain is composed of billions of these neurons, organized into circuits. Each neuron can be subdivided in multiple parts, some of which are highlighted in Figure 2.2. [2]

The neuron has a cell body, or soma, which has dendrites that can receive signals. Signals received from dendrites are processed within the dendrites and afterwards in the soma, whereby signals can be exhibitory or inhibitory. The signals from all dendrites are integrated in the neuron and, depending on if a threshold is crossed, a spike will be generated.
When a spike is generated: an action potential will be sent along the axon, the

integrated potential of the neuron will be reset, and a refractory period is initiated. [2] When no spike is generated: the integrated value is kept but is slowly, depending on the time constants, leaking away due to the analog nature of the neuron.
Biological neuron signals are spikes. These spikes can encode temporal and spatial information. Since spikes are linked to specific times they have a strong temporal aspect. Using this temporal aspect spikes have the potential to encode information efficiently making them essential for the SNN.

Spikes are however not directly differentiable since they do not have a continuous function, therefore the backpropagation method as used in ANN can not be used for training an SNN. Although surrogate gradients, to act as an approximation of a gradient, are shown to work: it has the exploding or vanishing gradient problem which hinders their training. [26] Moreover, multiple time-steps can contribute to the state of the network which makes the error assignment difficult.

A biological process called Hebbian learning, was discovered by Hebb [27] in 1949, this process is regarded as a basic model that allows learning by relating network activity to weights. On itself, Hebbian learning is however unstable. Several techniques were developed for stabilization, most noticeably: synaptic scaling, STDP, triple Spike Time Dependent Plasticity (tSTDP) and synaptic redistribution. [18]
Simplified models like the Tempotron learning rule [19] are available for simple networks that only have an input and output layer with one set of weights in between.

Even though SNN networks are still difficult to train and in general do not yet match the state-of-the-art performance of ANNs, taking the human brain as a reference, the SNN has the potential to be highly efficient and accurate.
Improvements in SNN models and their computational requirements are still on going to e.g. increase efficiency [25]

Since the SNN operates on spikes, the input and output layer node values are also restricted to single bit values. Moreover, the model incorporates memory and thus the order and input frequency can influence the output results. For the preprocessing it is important to preserve the temporal information and to convert multi-bit values to single-bit, spikes.

2.3 Audio preprocessing

In the work from Prozée [1], a commonly used audio preprocessing method based on [3] was used. The details can be found in [1] but since the remaining of this thesis is based upon this preprocessing method it is also summarized below.

2.3.1 Literature

The audio preprocessing pipeline as described in [3] is illustrated in Figure 2.3.
The information flow can be described as the following.

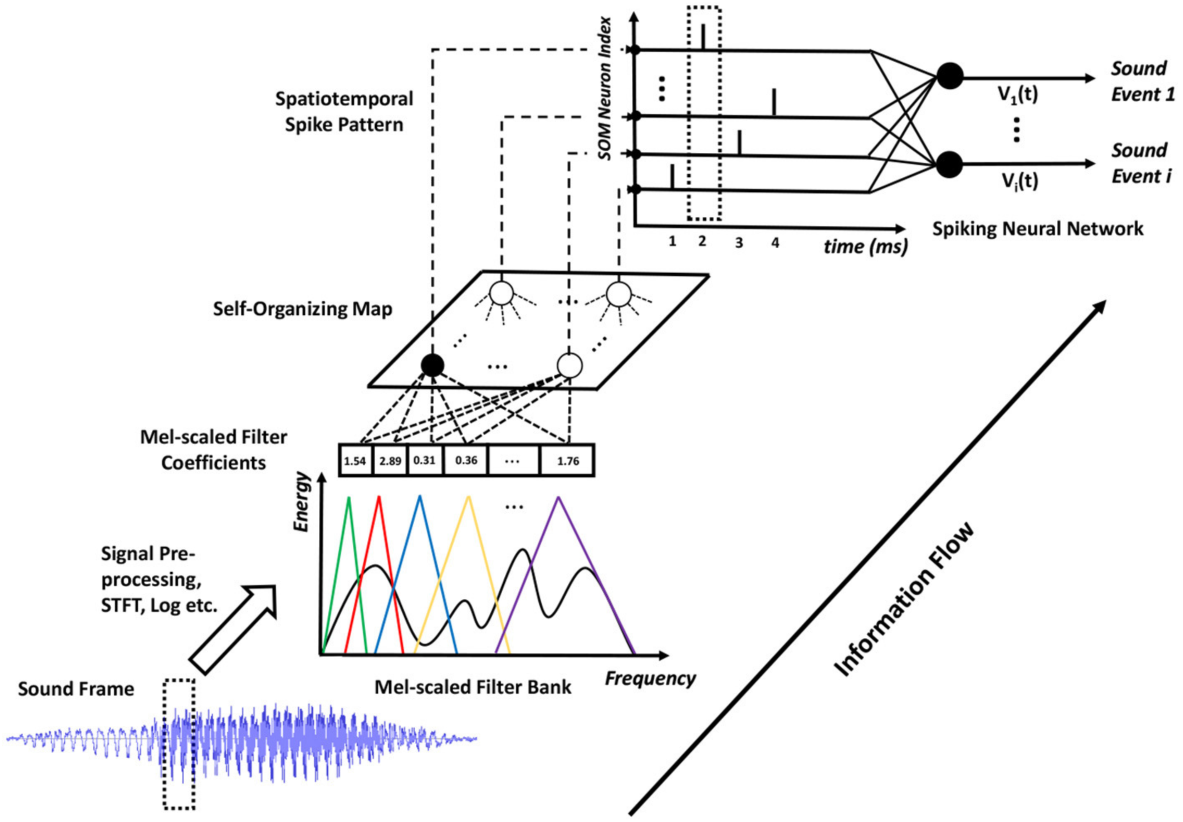


Figure 2.3: Audio preprocessing concept from literature [3]

Starting from a sound frame, first a time window is extracted. The samples in this time window are then filtered by a filterbank using mel-scaled filters. From the filtered signal, mel-scaled filter coefficients are obtained. These coefficients are an input to the Self-Organizing Map (SOM) which is used as an encoder. Eventually the SOM outputs spikes which are used as input for the SNN.

2.3.2 Overview

The schematic overview of [1] is illustrated in Figure 2.4. It should be noted that the figure proposes two separate architectures. The bottom architecture resembles the stages in the literature as described in [3], the top path is a simplified architecture. Both architectural paths have the same preprocessing step, the auditory front-end, but the method of encoding spikes is different.

Several blocks of interest in Figure 2.4 are highlighted below.

2.3.3 Audio Data

The used data set in [1] is Free Spoken Digit Dataset (FSDD) [28]. It contains 2,000 clean recordings of spoken digits ('0'-'9') from four speakers. The recordings are sampled at 8 kHz and stored in the Pule Code Modulation (PCM) format. This data set was

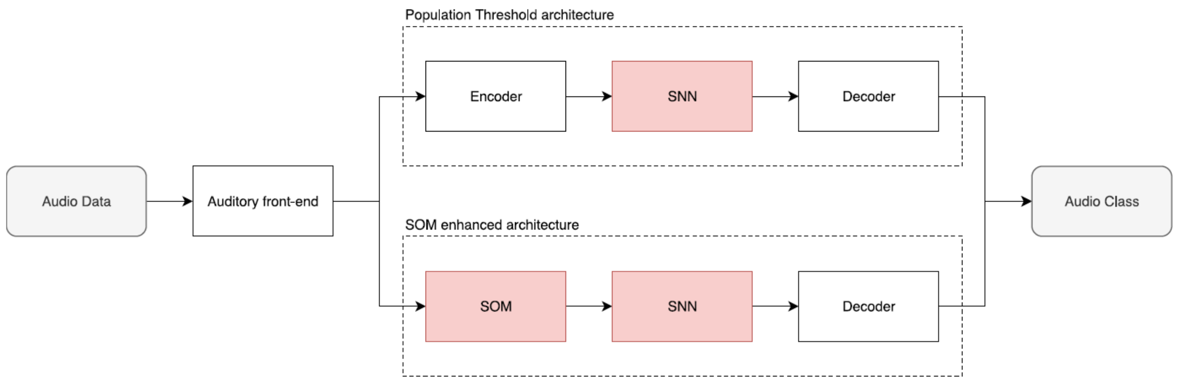


Figure 2.4: Audio processing overview as determined in the previous work [1]

chosen because it contains low noise which makes comparison under noisy conditions, by adding noise, easier.

2.3.4 Auditory front-end

The purpose of the auditory front-end block is to transform an audio data frame to a spectrogram. Examples for this are shown in Figure 2.5.

We identified two main steps within this module. First the audio signal is decomposed into frequency sub-bands, similar as to what happens in the human ear with the cochlear filterbank. To model the cochlear filterbank the Mel-Scale is used to represent the non-linear perception of the human ear. In state-of-the-art ASR systems the Mel-Scale parameters are handcrafted [29]. To translate between the Mel-Scale and frequency the following formulas can be used [30]

$$m = 2595 \log_{10}(1 + \frac{1}{700}) \tag{2.4}$$

$$f = 700(10^{m/2595} - 1) \tag{2.5}$$

where f stands for the frequency and m stands for the Mel-Scaled frequency.

Secondly the spectral energy within each sub-band is estimated, using

$$\log E = \ln \sum_{i=0}^{N} s_i^2 \tag{2.6}$$

Where s_i is the i-th sample in the frequency band.
The frequency spectrum of the mel-scaled filterbank is illustrated in Figure 2.6. Note that the filterbank is non-linear and it consists of triangles that overlap half of the frequency range of their closest neighbors.

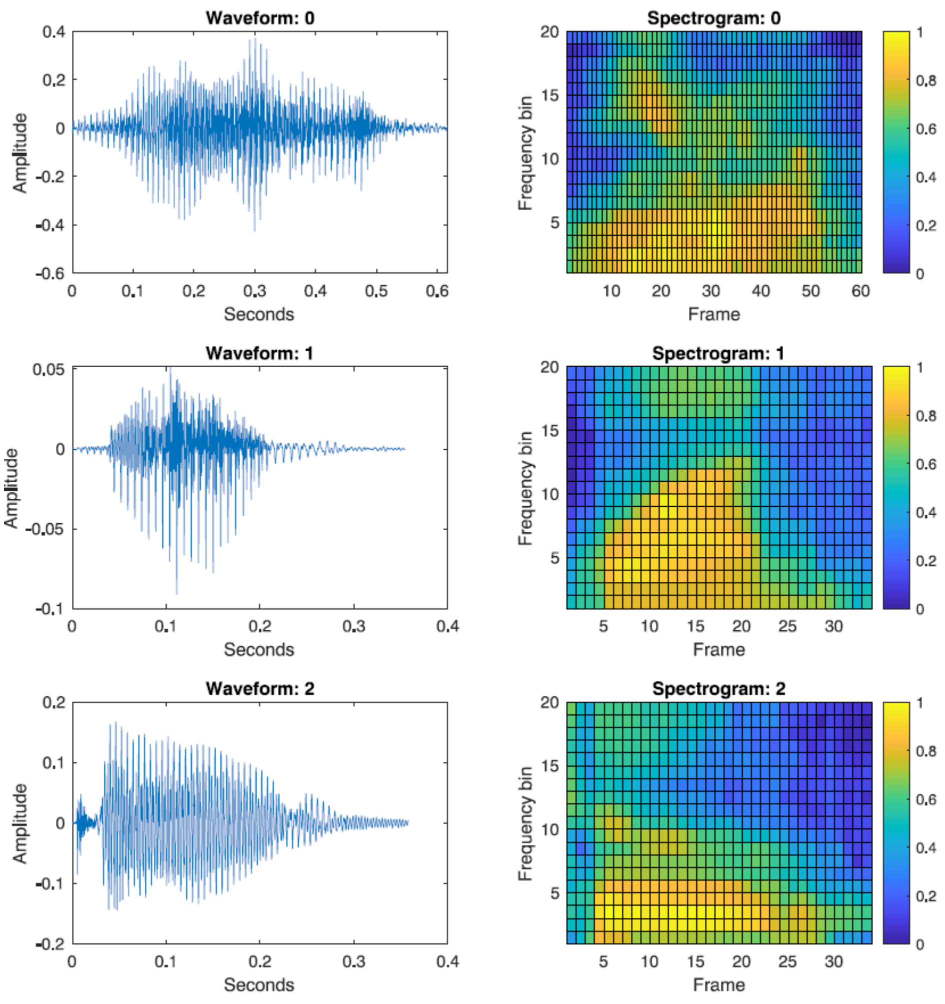


Figure 2.5: Examples for the audio to spectrogram stage [1]

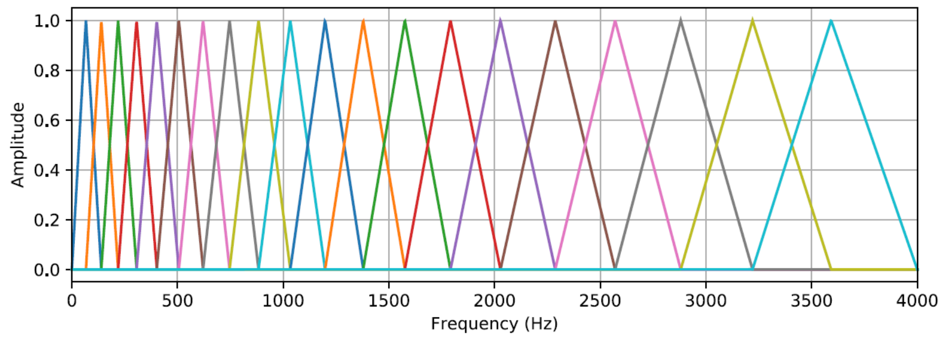


Figure 2.6: Frequency spectrum of the mel-scaled filterbank [1]

2.3.5 Encoder

Different encoding schemes are available of which two of them were compared: population Threshold Encoding (p-TE) and Self-Organizing Map (SOM). The input/output

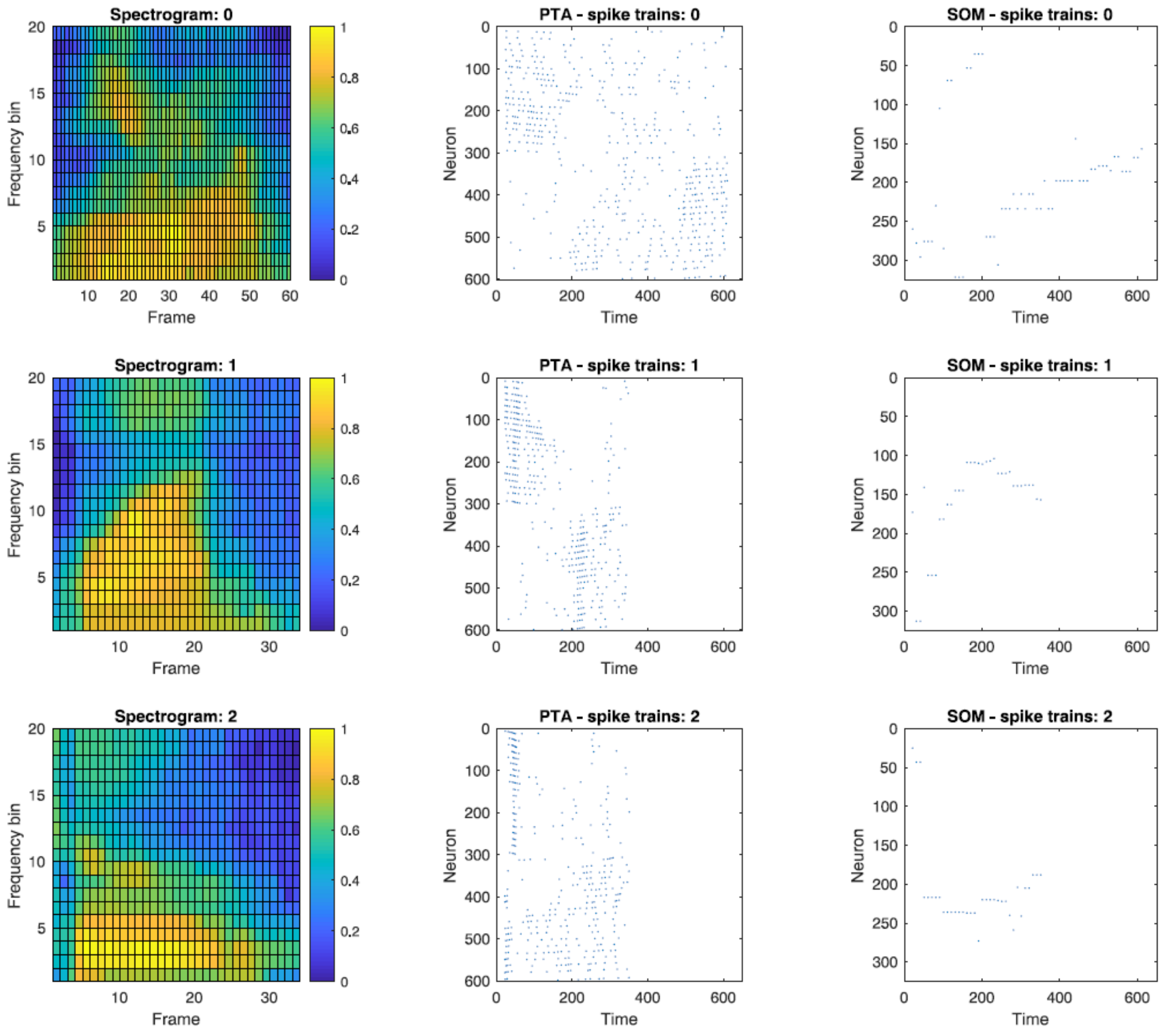


Figure 2.7: Examples for the spectrogram to spikes stage [1]

relation is illustrated in Figure 2.7 for both encoding schemes. The first column illustrates the input, the spectrograms, the second column indicates the output when using p-TE (PTA) while the third column indicates the output when using SOM as an encoder.

Both encoder schemes are mentioned in more detail below.

2.3.5.1 Population Threshold Encoding

When using p-TE [16] the temporal information is encoded by using absolute thresholds. Each threshold is linked to two nodes (neurons). When a parameter changes value, from one time instance to another, it can cross a threshold. Depending on the direction of the threshold crossing, from below or from above, one of the linked neurons to that threshold will fire. This is illustrated in Figure 2.8, where the temporal change of a parameter is shown in (a) and its corresponding spike pattern in (b). Note that there is twice the amount of neurons as thresholds and that the red spikes indicate crossings from below while blue spikes indicate crossings from above. As stated in [1], p-TE is a

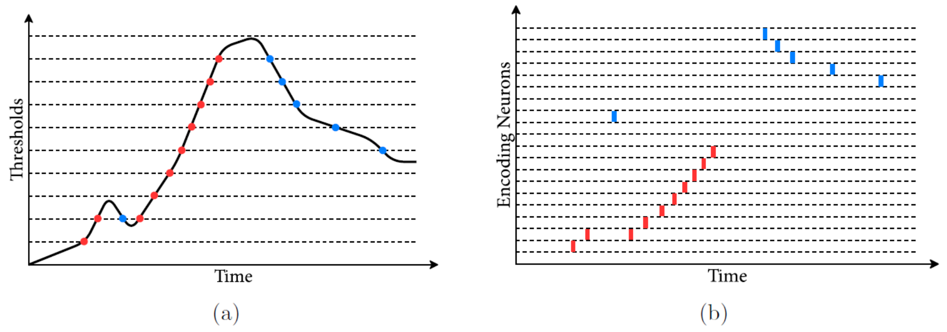


Figure 2.8: Population Threshold Encoding [1]

computationally simple algorithm but has non-optimal encoding performance.

2.3.5.2 Self-Organizing Map

The SOM, as first described in [31], is used for feature reduction. It is designed on the concept that the network groups together inputs which are similar. In [3] the SOM is, besides feature reduction, used to encode multi-bit values to single-bit, spikes. The node that best matches the input will win, inhibiting the surrounding nodes. The node that wins can be viewed as having generated a spike. As stated in [32], the SOM is a

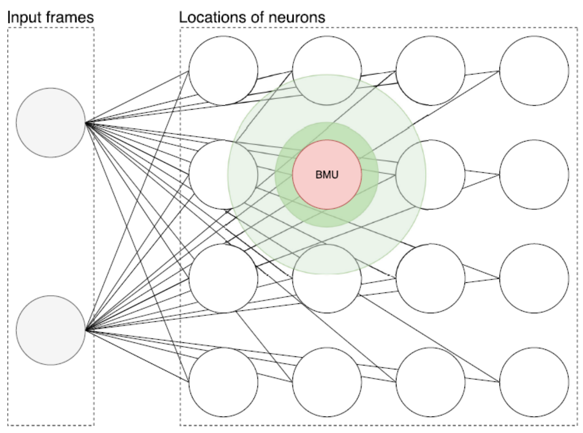


Figure 2.9: Concept of a SOM network [1]

computationally expensive algorithm that scales with $O((N * M)^2)$ where N is the size of the input vector and M the size of neurons in the grid. In [1] it however did enable the SNN to obtain accuracies above 95%.

2.3.6 SNN

The SNN module stands for the neural network that processes the spike input data and generates spike output data, using a Spiking Neural Network. In order for the network to produce useful output, training is necessary. Since an SNN is a temporal network, a temporal learning rule is required. In [1] an implementation of Tempotron [19] was used.

2.3.6.1 Tempotron

Tempotron [19] is a supervised temporal learning method that, depending on if a spike is desired will adjust the weights of the links according to an update rule. A limitation of Tempotron is that only a single layer is allowed, making the network shallow and thereby limiting the distinguishability of the SNN, the Tempotron learning rule is elaborated further in section 7.2.

2.3.7 Decoder and Audio Class

The SNN's output spikes need to be related to a certain class for classification. Different methods are available, e.g. first time to spike, highest spike count or highest rate. In [1], with Tempotron, the first time to spike decoding was used.

2.3.8 Results

Both the p-TE and SOM encoders were compared by Prozée [1] based upon the classification of 10 digit classes and one optional noise class.

For the p-TE architecture it was determined that at least 15 (uniform) thresholds are required for optimal performance, the obtained classification accuracy was about 86% for clean audio recordings. When noise is added the accuracy was found to drop to 30% with a $-5dB$ SNR.

For the SOM architecture at least an 18×18 network is recommended by Prozée [1] for his specific audio preprocessing. the classification accuracy was found to be around 96% for clean recordings while with $-5dB$ Signal-to-Noise-Ratio (SNR) the accuracy dropped to 32%.

2.4 Implementation

Several hardware implementation descriptions related to audio preprocessing for speech recognition are given in literature, two of which are highlighted here: First, Vocell which follows the overall audio preprocessing pipeline, as given in section 2.3, and secondly Blackfin which illustrates how to interface with Micro Electro-Mechanical System (MEMS) microphone.

2.4.1 Vocell

Vocell [4] is intended to be used as a Speech-Triggered wake up system. The focus is to reduce power by separating a pipeline into multiple modules. Only when a module detects something of interest the subsequent modules are activated. The modules are illustrated in Figure 2.10

Where its Frontend Extraction block, in purple, consists of [4]:

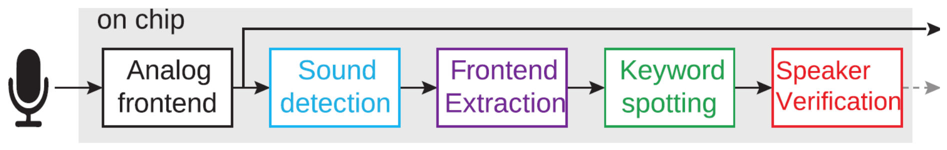


Figure 2.10: Vocell pipeline, adapted from [4]

1. FFT for spectral analysis
2. Mel filtering with triangular filters
3. Logarithmic compression
4. DFT over the intermediate values

Which, except for the DFT, is similar to the pipeline as described in section 2.3.
It is noted from this paper [4] that the DFT accounts for approximately 72% of all computations. Reduced energy consumption is obtained by creating optimized dedicated hardware and by placing a sound detector module, that acts as a gateway, at the beginning of the pipeline.
With this specific design they managed to reduce the power from $18.3\mu W$ to $10.6\mu W$, a reduction of about 45%.

2.4.2 Blackfin

To interface with a MEMS microphone, a microprocessor like the Blackfin [5] can be used. In [5] a design is given to obtain a multi-bit audio signal from an oversampled single-bit signal. Even though their design uses two microphones while we only consider one, as shown in Figure 2.11, the principle is the same.

To obtain multi-bit samples from single-bit samples a multi stage approach was used with the following modules [5]:

1. CIC Decimator
2. 2:1 HB filter
3. 2:1 HB filter
4. FIR low pass filter

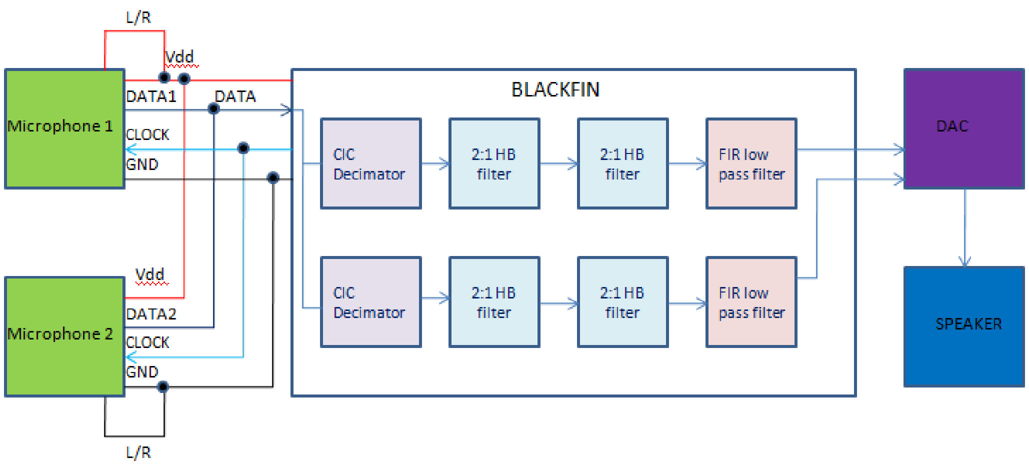


Figure 2.11: Interfacing MEMS microphone with Blackfin [5]

Methods 3

The aim of this thesis project is to perform a mapping from software to hardware and to evaluate the hardware mapping afterwards. In this chapter our approach and its subsequent requirements are highlighted to show how the mapping can be performed in a structured manner.

3.1 Approach

The software to hardware mapping was performed in incremental steps where for each step the influence on the whole system can be evaluated. This is illustrated in Figure 3.1

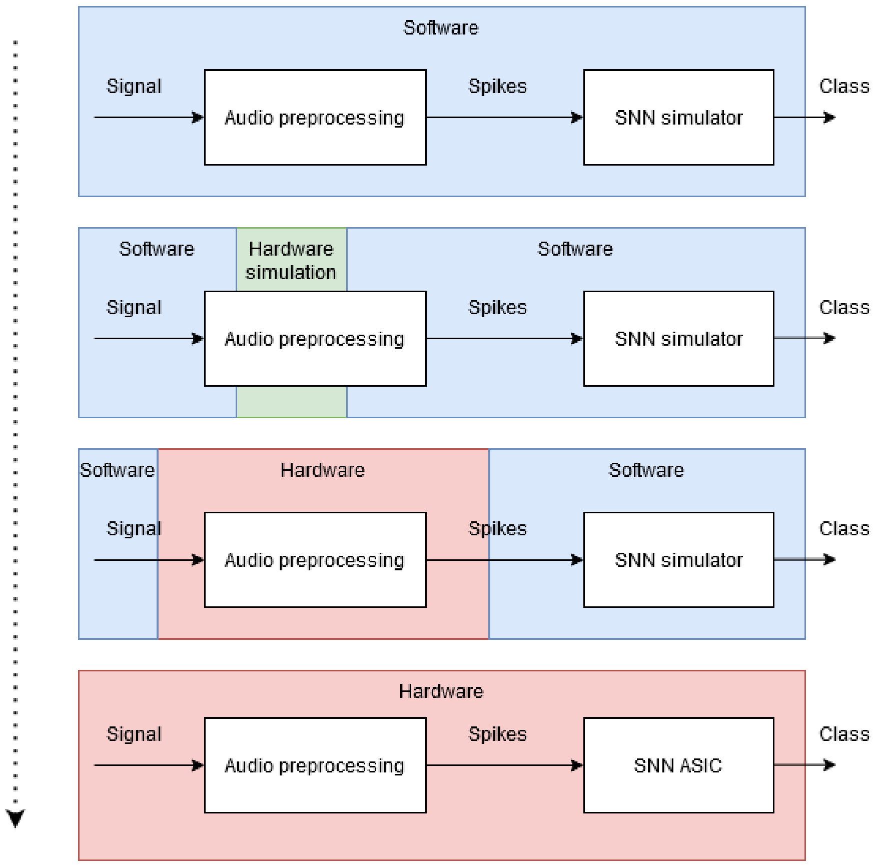


Figure 3.1: Use incremental steps for software to hardware mapping

The intent of this approach is to organize the project in a structured way using incremental steps and independent modules.

3.2 The Controller

To facilitate the approach, a framework called The Controller was built. The schematic of The Controller is given in Figure 3.2

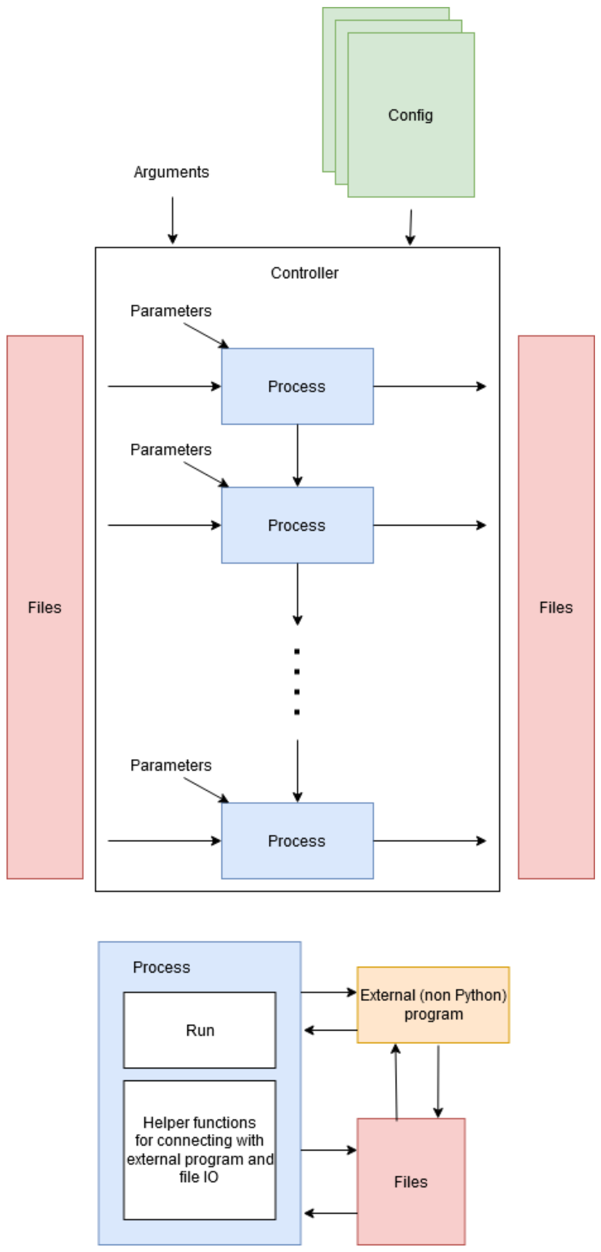
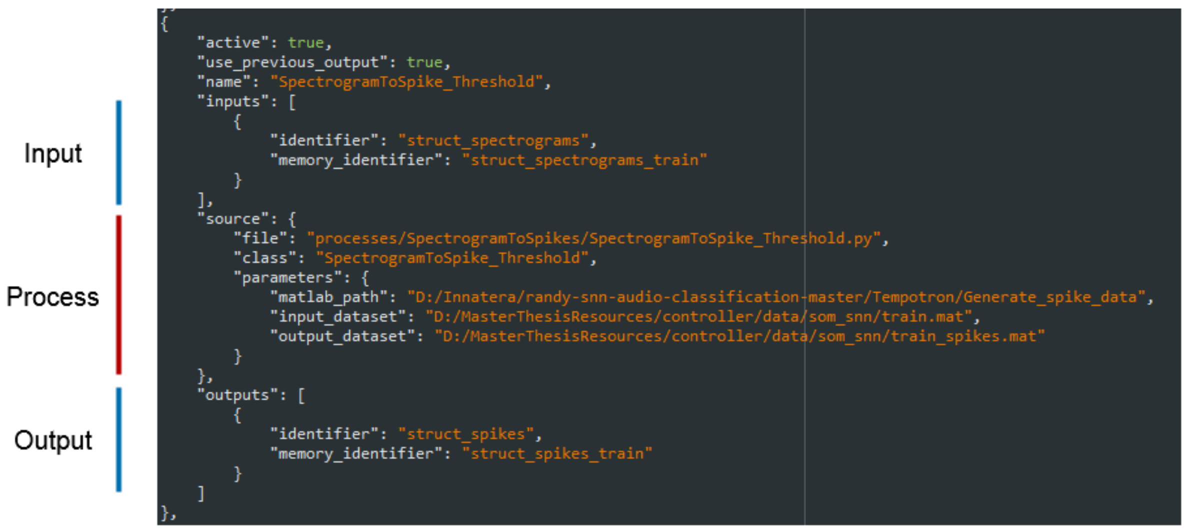


Figure 3.2: The controller can execute modules in a sequence, stitching together the input/output of each.

The Controller is a framework in which processes (modules) can be defined. Each module is written in the same language as the controller, Python. The module itself can however be used for anything, e.g. run executables, as long as the module adheres to the input/output relation of the controller. Configuration files, written in JavaScript

Object Notation (JSON) [33], are used to choose which modules should execute and how their inputs and outputs are connected. An example is shown in Figure 3.3.



```
{
    "active": true,
    "use_previous_output": true,
    "name": "SpectrogramToSpike_Threshold",
    "inputs": [
        {
            "identifier": "struct_spectrograms",
            "memory_identifier": "struct_spectrograms_train"
        }
    ],
    "source": {
        "file": "processes/SpectrogramToSpikes/SpectrogramToSpike_Threshold.py",
        "class": "SpectrogramToSpike_Threshold",
        "parameters": {
            "matlab_path": "D:/Innatera/randy-snn-audio-classification-master/Tempotron/Generate_spike_data",
            "input_dataset": "D:/MasterThesisResources/controller/data/som_snn/train.mat",
            "output_dataset": "D:/MasterThesisResources/controller/data/som_snn/train_spikes.mat"
        }
    },
    "outputs": [
        {
            "identifier": "struct_spikes",
            "memory_identifier": "struct_spikes_train"
        }
    ]
},
```

Figure 3.3: Example of a JSON configuration file

The hierarchy of the controller and its modules is illustrated in Figure 3.4

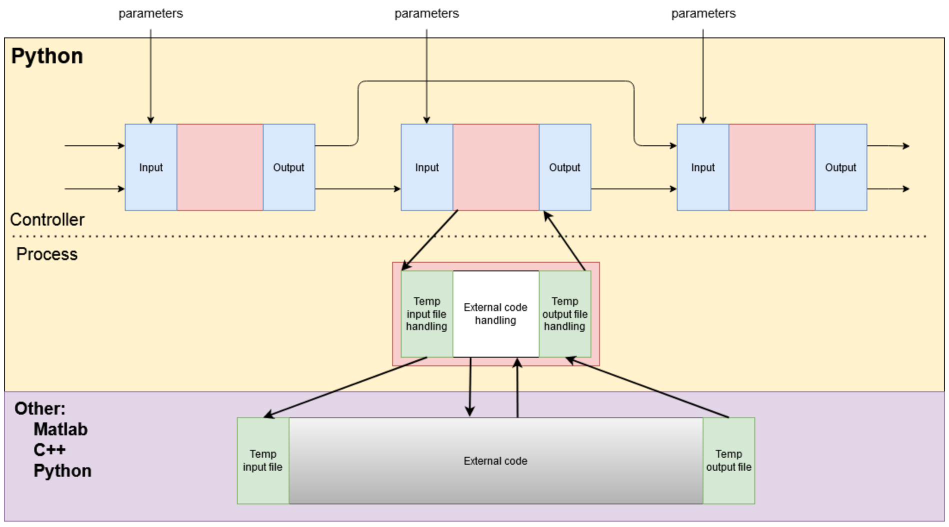


Figure 3.4: Each process handles its own interfacing with the controller and its external program

Using this hierarchy, the controller is a uniform runner that can run a pipeline irrespective of the module details. Making it possible to swap modules that have the same input/output relation but differ in implementation. Because the modules for both software and hardware are digital and should have the same input/output relation it means that software and hardware coded modules can be easily swapped.

Thus with the controller it is possible to test the performance of the entire system even though some parts are using different environments, e.g. Matlab, Python, C++, Verilog.

The Controller is the realization of the contribution “A framework that supports the chaining of several programs, written in different programming languages, together into one pipeline.”

3.3 Modules

As is clear from the approach and the corresponding controller, a subdivision in modules (processes) is required. The overview of the pipeline and the interaction between modules is provided in Figure 3.5.

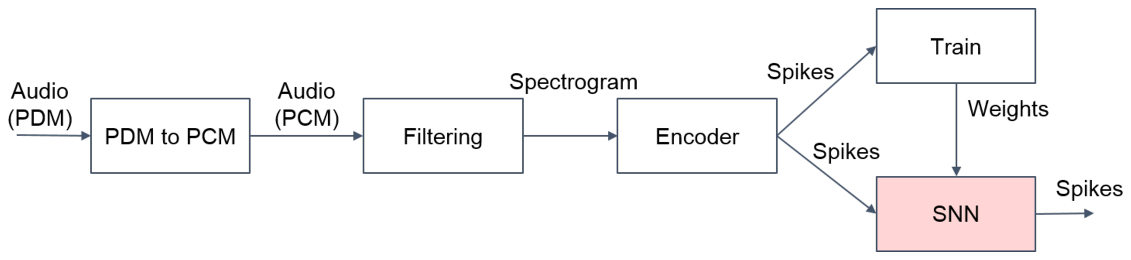


Figure 3.5: Interaction of modules

A short definition for each module is given below, the separation is partly based upon Figure 2.4. Each module, except SNN, is elaborated extensively in their own chapter.

3.3.1 PDM to PCM

This first module is responsible for audio encoding conversion. The input is an oversampled PDM signal while the output is a PCM signal at sampling frequency.

3.3.2 Filtering: The Filterbank

Similar as in Figure 2.4, its purpose is to convert a PCM audio signal to a spectrogram.

3.3.3 Encoding

The encoder is responsible for translating the varying time slice spectrogram values to temporal spikes, which are compatible with the SNN.

3.3.4 Training

Currently training only takes place in the simulator, there is no training on chip. Training is required to obtain the weights for the SNN. Once this is done, the weights are a part of the configuration file for the SNN. It has a separate module since different training methods are available.

3.3.5 SNN

This module is not elaborated since it is used as a black box, it includes the SNN itself and the decoder. General information about the SNN is provided in chapter 2. Its purpose in the pipeline is to indicate the bigger picture.
The network is based upon Leaky Integrate Fire (LIF) [1] and its size, with only an input and output layer, is bound to the input vector and output classes which are approximately 500 and 11 neurons respectively

PDM to PCM

4

The first stage in the pipeline is the Pulse Density Modulation (PDM) to Pule Code Modulation (PCM) module. This module is responsible for translating the oversampled PDM microphone data to a PCM signal that the subsequent modules can use.
First the definitions for PDM and PCM encoding schemes are given. Secondly, the PCM to PDM conversion is highlighted. Finally, the PDM to PCM conversion is evaluated.

Note that a conversion from PCM to PDM is often required in order to use existing datasets that are encoded using the PCM format since the audio preprocessing is expecting PDM samples duo to the MEMS microphones. In order to have reliable simulations, the PCM to PDM conversion is based upon the same modulator as the microphone under consideration, a sigma-delta modulator. Investigation into the microphone modulator provides insight in its noise shaping.

4.1 Definitions

Both PDM and PCM are methods to encode data in a digital form. In Figure 4.1 both encodings, PDM in red and PCM in blue, are shown superimposed on top of each other.

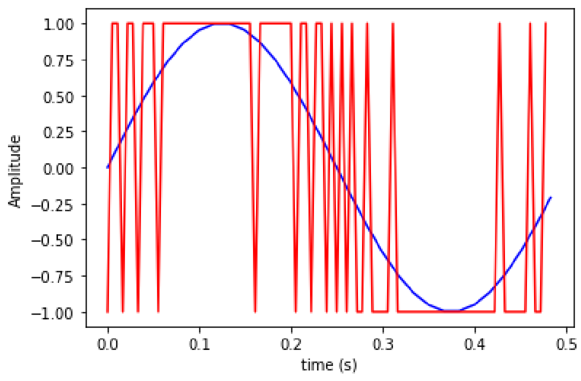


Figure 4.1: PDM (binary in red) and PCM (sine wave in blue) encoding

4.1.1 PDM

Pulse Density Modulation (PDM) encodes information using a single bit value, encoding data in densities. PDM is widely used as data format output of MEMS microphones since it has the potential to digitally represent high quality audio while being inexpensive. [34].

4.1.2 PCM

Pule Code Modulation (PCM) encodes information using a multi-bit value to represent a fraction of a range. If the range is know, the value at each data point is also defined. The PCM format is often used in signal processing [35], it has an abundant amount of theory and examples from literature and consequently simplifies the implementation and design choices for the signal processing operations.

4.2 Input Sources

4.2.1 Recording: Microphone

The supplied microphone, according to its description, has low self-noise, a large dynamic range, low distortions and a high acoustic overload point [6], making it a high performance microphone. Its frequency response can be found in Figure 4.2 and its noise floor in Figure 4.3.
Since the sample rate is $8kHz$, a flat frequency response is expected for most frequencies. The noise floor is quite low and therefore suggests that a high order $\Sigma\Delta$ modulator is used. According to literature a 5th order with $64x$ oversampling is usually used [36].

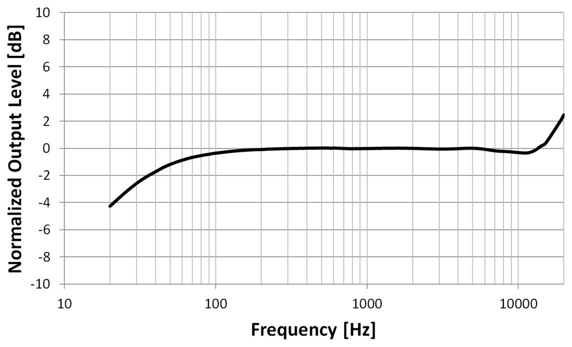


Figure 4.2: Microphone frequency transfer [6]

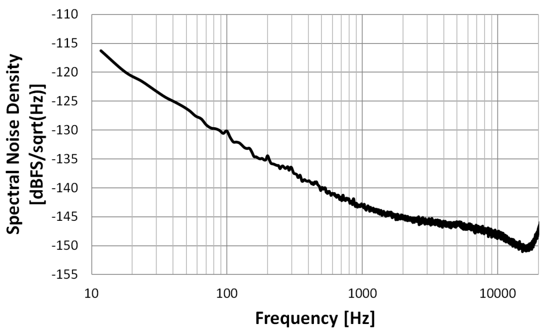


Figure 4.3: Microphone noise floor [6]

4.2.2 Datasets: PCM to PDM

The data set used for testing is stored in PCM encoding. To perform tests for the PDM to PCM module, the data should first be translated to PDM encoding.

To translate PCM coded data to PDM an $\Sigma\Delta$ modulator is used. The $\Sigma\Delta$ modulator can be understood as in that it stores a single PCM value as the average value over a window T of sequential binary values with a time step t. Written in an equation [37], it means that:

$$\mu(T) = \frac{1}{T}\int_0^T f(t)dt \tag{4.1}$$

Or written as a sequence N of fixed width pulses τ [0,1]:

$$\mu(N\tau) = \frac{1}{N\tau}\int_0^{N\tau} f(t)dt = \frac{1}{N}\sum_{t=0}^{N-1} l(i) = \frac{S(N)}{N} \tag{4.2}$$

Where $S(N)$ indicates the sum over N samples.
To store the average in a window of binary pulses, a running quantization error is fed back to the input.
In a block diagram this can be represented as shown in Figure 4.4 for a first order $\Sigma\Delta$ modulator.

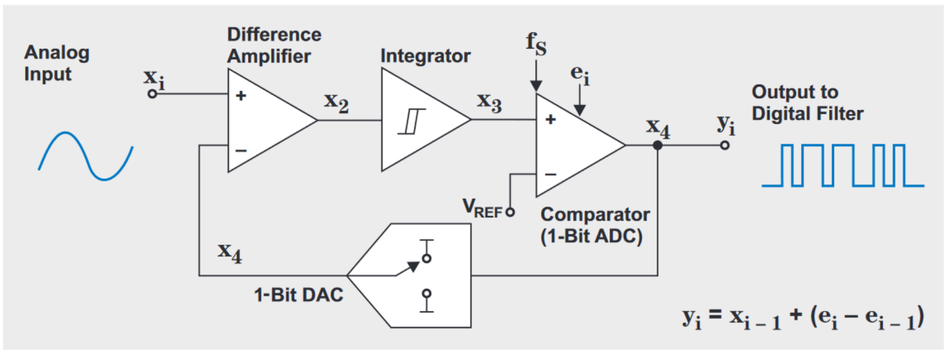


Figure 4.4: $\Sigma\Delta$, first order [7]

A $\Sigma\Delta$ modulator performs noise shaping, which depends on the order. The quantization noise over the entire frequency range can be modeled with the following equations for N-order, given in [38].

$$n_0 = e_{rms}\frac{\pi^N}{\sqrt{2N+1}}(2f_0\tau)^{\frac{2N+1}{2}} \tag{4.3}$$

with

$$e_{rms} \approx \frac{\Delta}{2\sqrt{3}} \tag{4.4}$$

where Δ is the step size to keep the waveform within range. Further elaboration on the noise shaping mathematics can be found in [39]

The concept of noise shaping for first until third order is illustrated in Figure 4.5. An enlarged version with values in the region of interest can be found in Figure A.1. It should be noted that the $\Sigma\Delta$ modulators inherently contain quantization noise but higher order modulators shape a larger fraction of this noise outside of the band of interest.

According to the noise shaping, a high order $\Sigma\Delta$ modulator is desired to lower the noise in the band of interest. In literature it is however noted that a modulator with

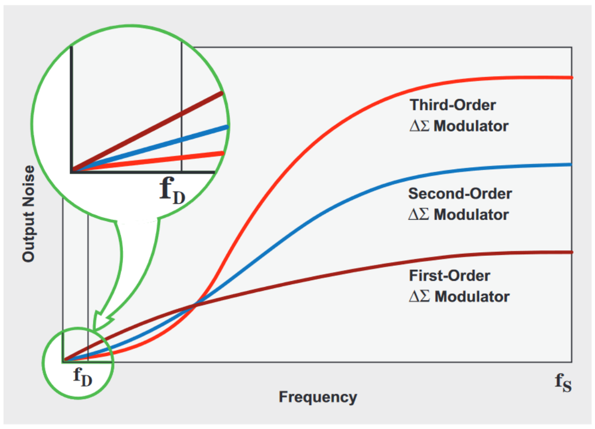


Figure 4.5: ΣΔ, noise shaping [7]

an order higher than 2 is not simple to construct because the phase turn, by more than two integrators, will make the system unstable [36]. An example of a second order modulator is illustrated in Figure 4.6.

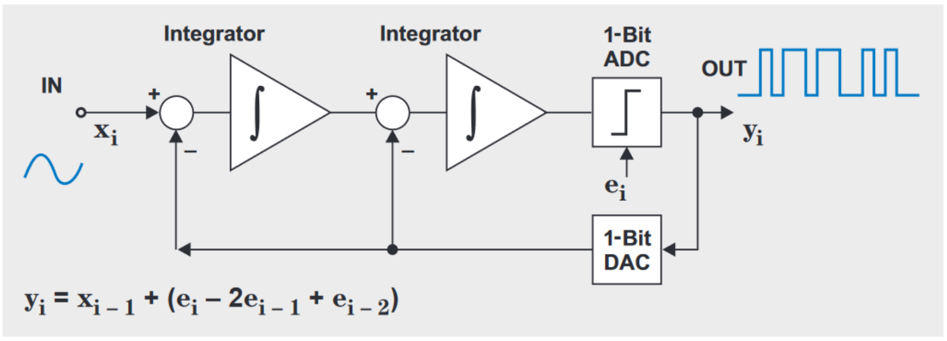


Figure 4.6: ΣΔ, second order [7]

Taking into account that the microphone is using a high order ΣΔ modulator to generate PDM encoded signals from the data set, a second order modulator was chosen. Being a compromise between noise suppression, stability and simplicity. According to Figure A.1, the second order modulator, with 64 times oversampling already suppresses the noise an additional 20 dB.

The Python code for PDM to PCM conversion, used only for simulation purposes, is provided in Listing B.

4.3 PDM to PCM

For PDM to PCM conversion a (simple) low-pass filter can be used, illustrated in Figure 4.7. In literature, multiple variants and even combinations of them are used, e.g. the Blackfin [5]. In this section several popular variants are explored and compared.

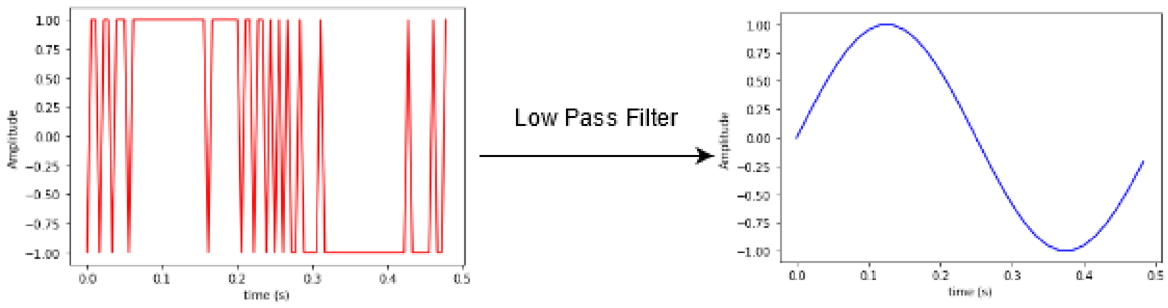


Figure 4.7: PDM to PCM encoding

4.3.1 IIR or FIR

A standard choice for a filter is to use an Infinite Impulse Response (IIR) or Finite Impulse Response (FIR). The advantage of an IIR is that, compared to an FIR, a lower order filter can obtain a comparable frequency response, making the IIR in general more computationally efficient. The disadvantage is that the IIR is potentially unstable, especially when fixed point with reduced precision is used.

The FIR has the advantage of always being stable and having zero phase shift although it requires a higher order than the IIR and is thus more computationally expensive.

The IIR and FIR act upon the oversampled PDM signal while their output is sampled at the sampling frequency. With the direct FIR implementation many intermediate values, before down sampling, will turn out to be unused. In the upcoming sections an optimized FIR implementation is elaborated and two different filtering methods are highlighted.

4.3.2 Polyphase FIR

Since both a downsampler and a (sparse) transfer function are present, the multirate noble identities [40][41] can be of interest. As illustrated in Figure 4.8, the downsampler and transfer function can be commuted if the transfer function can be modified.

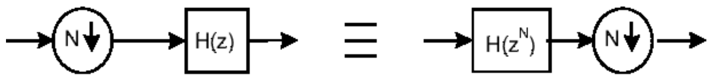


Figure 4.8: Multirate noble identity for downsampler [8]

Based upon the multirate noble identities a concept called polyphase filtering is known in literature. The idea is to prevent the computation of unused intermediate results, since the signal is downsampled after filtering. Note that the IIR has a feedback loop, not satisfying the noble identity, thus only the FIR can be optimized using polyphase filtering.

The concept of polyphase filtering is illustrated with Figure 4.9 where the direct form FIR is shown while the polyphase FIR is shown in Figure 4.10 The FIR tap coefficients are marked with h, input samples with x and (decimated) output samples with y,

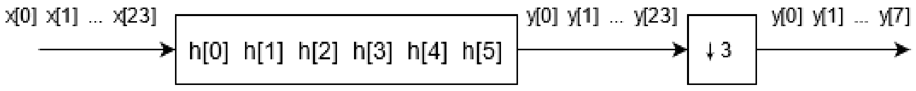


Figure 4.9: Direct Form FIR decimation

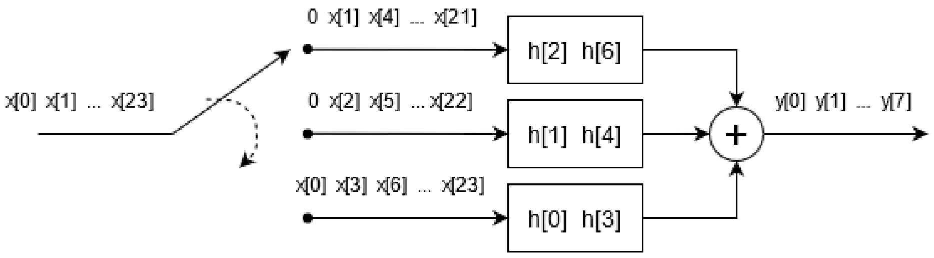


Figure 4.10: Polyphase form FIR decimation

In the direct form the FIR utilizes all of its hardware structure on each input sample, after which only several output samples are chosen by the downsampler. Based upon [42] and [43] this can be mathematically expressed for a 3 times downsampler as:

$$[s_0\ s_1\ ...s_8\ s_9] = [h_0\ h_1\ ...h_4\ h_5] * [x_0\ x_1\ ...\ x_8\ x_9]$$

$$[y_0\ y_1\ y_2\ y_3] = [s_0\ s_3\ s_6\ s_9]$$

In the polyphase form an input sample is directed to only one of the polyphase structures, where each structure is a small FIR. In this case only one structure will be executed each time step. After each polyphase structure has obtained a single input sample, the intermediate outputs are summed and the final output is obtained.
This is mathematically expressed as:

$$\begin{aligned}[0\ s_1\ s_4\ s_7] &= [h_2\ h_5] * [0\ x_1\ x_4\ x_7]\\ [0\ s_2\ s_5\ s_8] &= [h_1\ h_4] * [0\ x_2\ x_5\ x_8]\\ [s_0\ s_3\ s_6\ s_9] &= [h_0\ h_3] * [x_0\ x_3\ x_6\ x_9]\end{aligned}$$

$$[y_0\ y_1\ y_2\ y_3] = [(s_0)\ (s_1 + s_2 + s_3)\ (s_4 + s_5 + s_6)\ (s_7 + s_8 + s_9)]$$

The input / output relation is identical for both the direct and polyphase forms [42].

Inspection of these equations makes it clear that the coefficients of the polyphase FIR structures can be derived from the FIR coefficients by:

$$\text{coefficient\_polyphase}[i][j] = \text{coefficient}[\text{NUM\_POLY} - i + j * \text{NUM\_POLY} - 1] \quad (4.5)$$

where coefficient_polyphase is a 2-dimensional array with index i indicating the polyphase structure and index j the coefficient for it. NUM_POLY is a constant value, equal to the number of polyphase structures and the decimation factor.

Since the input / output relation of the direct FIR implementation with down sampling is equal to the polyphase FIR implementation it is sufficient to determine the amount of taps, and their coefficients, required for a desired polyphase filter with the frequency analysis of a direct FIR implementation.

To design the filter, the assumption is made that the PDM signal is sampled at 512kHz, which is an oversampling of 64 times with respect to the desired PCM signal at 8kHz. Different filter window types, e.g. Hamming or Chebyshev, are available. In this case it is chosen to use Hamming for its relatively steep transition from passband to stopband and sufficient (around 80 dB) stopband attenuation, assuming that most of the noise is expected at high frequencies because of noise shaping.

The frequency response of order 64, 128 and 192 filters are illustrated in Figure 4.11, Figure 4.12 and Figure 4.13 respectively. The passband for the different orders is given in Figure 4.14. Note that the values under consideration are a multiple of the oversampling rate, 64 times, which is required for efficiency reasons.
The 64 order filter only has a limited suppression of about 50 dB for out of passband frequencies while the 192 order has a significant passband droop. The 128 order filter has sufficient, 80 dB, suppression and no significant passband droop, making it the desired option.

4.3.3 CIC

Another often used decimation method [5] is called Cascaded Integrator-Comb (CIC). A simple block diagram is shown in Figure 4.15 and a possible frequency transfer response in Figure 4.16.

From Figure 4.15 it is clear that the CIC consists of two parts. The first is the Combinational part, which keeps track of a running average. The second is the Integrator that accumulates the running average values. In summary it is just a very efficient recursive implementation of a moving-average filter [44]. Note that Z^D implies that D amount of delays are used in this structure, where D linearly depends on the oversampling rate.

The operations can be rearranged to Figure 4.17. Here a decimation factor R is applied between the integrator and comb sections. Now only $N = D/R$ amount of delays should be used for the same frequency response since the comb section is running at a speed reduced by R. An usual value for N is 1 or 2, according to[9]. Therefore, it is estimated that 2 adders and 3 multi-bit delays are required per order. A higher order CIC can be obtained by increasing K.

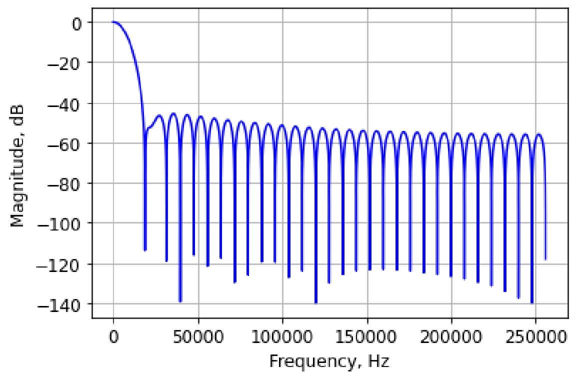


Figure 4.11: Polyphase frequency response, 64 order.

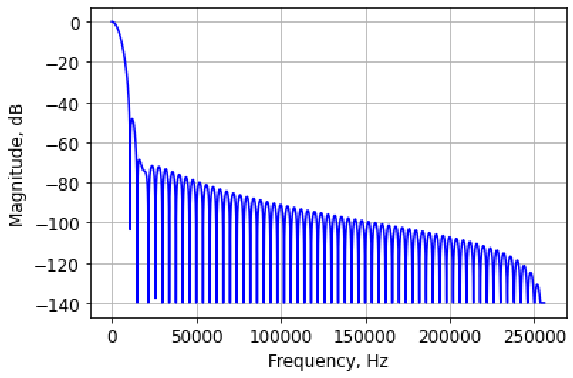


Figure 4.12: Polyphase frequency response, 128 order.

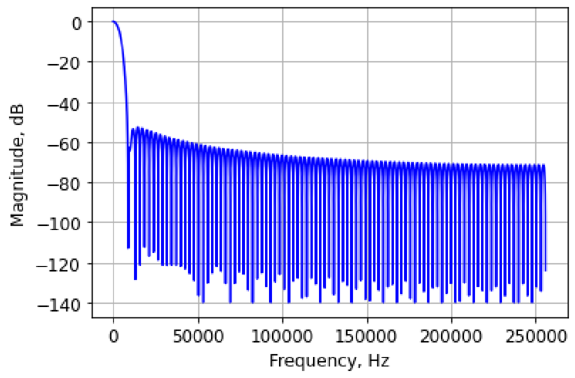


Figure 4.13: Polyphase frequency response, 192 order.

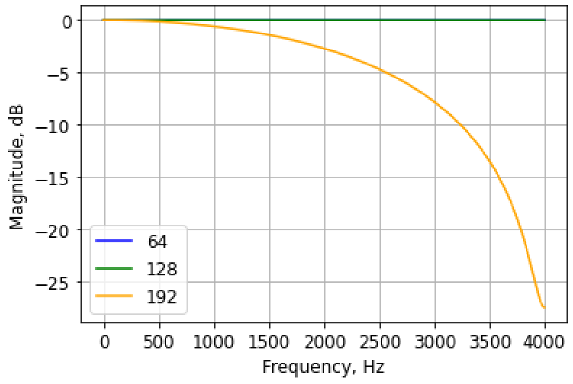


Figure 4.14: Polyphase frequency response at the passband for different filter orders

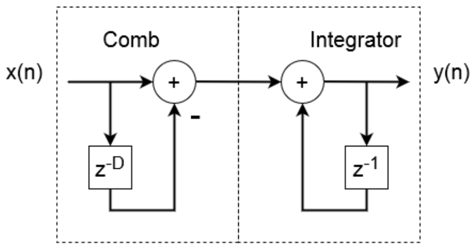


Figure 4.15: CIC first order

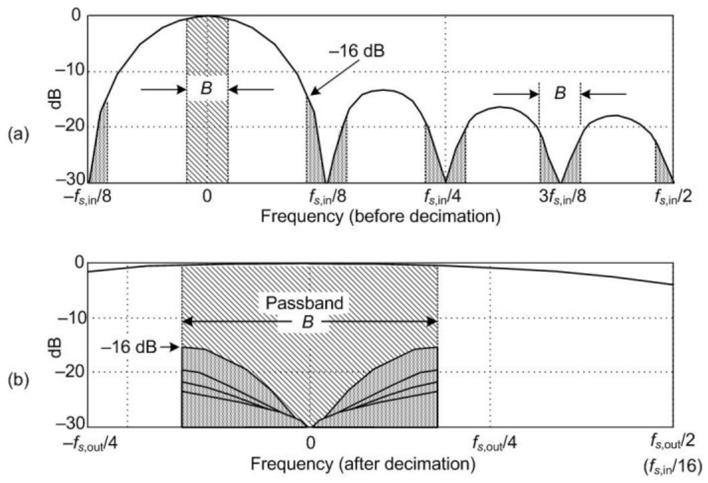


Figure 4.16: CIC Frequency response, the shaded areas will alias into the band of interest [9]

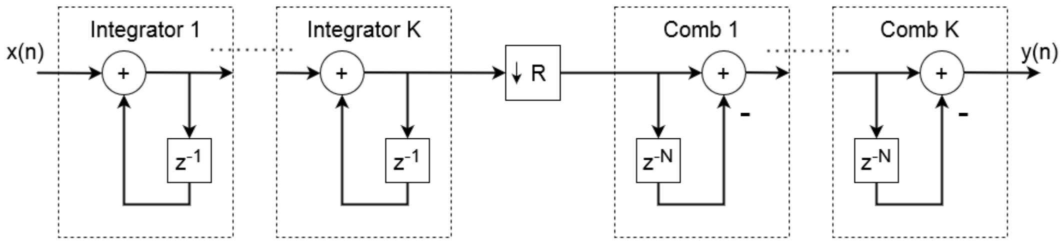


Figure 4.17: CIC multi-order rearranged and decimation

Mathematically, the CIC has a transfer function described by [5]

$$H(z) = H_I^K(z)H_C^K(z) = (1 - z^{-1})^K \frac{1}{(1 - z^{-D})^K} = \left[\sum_{i=0}^{RK-1} z^{-i}\right]^K \tag{4.6}$$

Where H_I and H_C are the transfer functions of the integrator and comb parts respectively. N is the number of delays, R the decimation factor and K the order.

The CIC frequency response of the entire range, assuming a 64 times oversampled PDM signal with respect to a 8 KHz sampled PCM signal, is shown in Figures 4.18 until 4.21 for different orders. The passband frequency response for different orders is given in Figure 4.22.
Note that with an increase of filter order the stopband attenuation is increased but the passband will have a larger gain droop.

Increasing N increases the stopband attenuation but worsens the CIC passband droop significantly (not shown here). From experiments it is seen that a CIC with $N = 1$ and $K = 3$ is deemed optimal for this use case, i.e. to have sufficient noise suppression at higher frequencies.

Because of the positive feedback the CIC can potentially have infinite gain. Fortunately as long as the stages use 2's complement and have at least the following amount of bits, overflow does not matter. [9]

$$\text{register bit width} = \text{input bits} + \lceil K \log_2(NR) \rceil \tag{4.7}$$

Since the input is single valued, this results in 19 bits required for each component, i.e. adders and registers, to prevent overflow consequences.

To compensate the drooping passband gain of the CIC, compensating filters can be applied after the CIC stage. The frequency responses for a first order and a third order CIC are illustrated in Figure 4.23. Note that the complexity of the compensating filter increases with the order of the CIC.

Although a compensating filter can be used, it is complex to design it. Moreover it will have an additional computational cost, making it undesirable.
Due that the CIC with N = 1 and K = 4 only has a slight passband droop while still providing some suppression, especially for high frequencies, it is deemed to be the

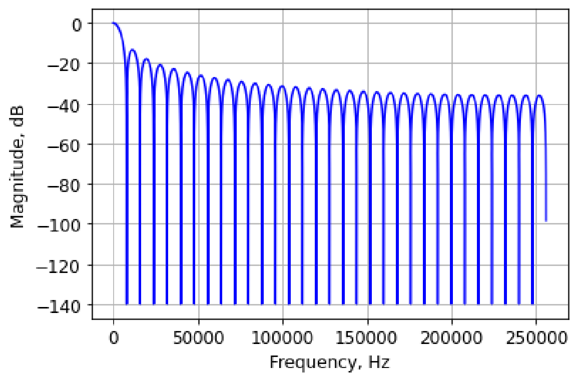


Figure 4.18: CIC frequency response: N = 1, K = 1

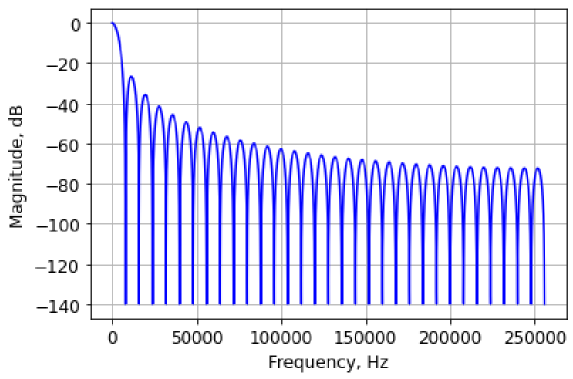


Figure 4.19: CIC frequency response: N = 1, K = 2

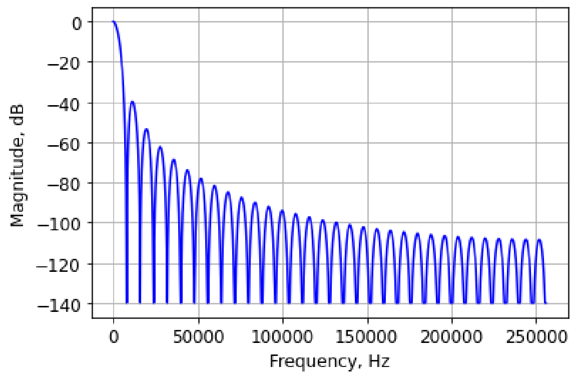


Figure 4.20: CIC frequency response: N = 1, K = 3

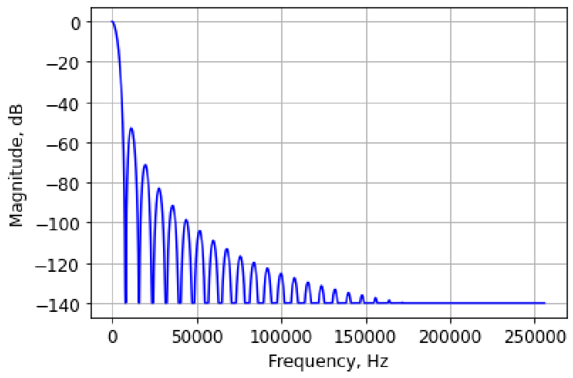


Figure 4.21: CIC frequency response: N = 1, K = 4

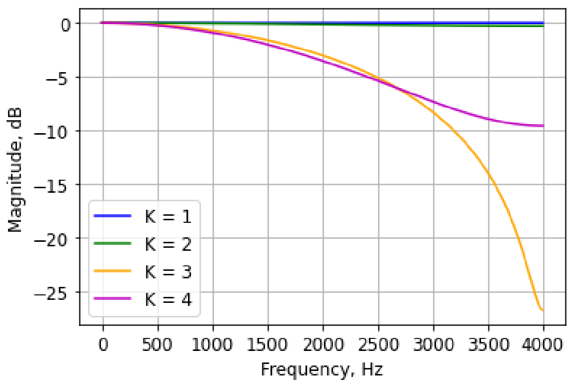


Figure 4.22: CIC passband frequency response for N = 1

most desired configuration.

There are versions of the CIC that have a higher quality frequency transfer response, e.g. Sharpened CIC [45] or Improved CIC [46], but they have higher computational cost. Another option includes using a combination of CIC and a

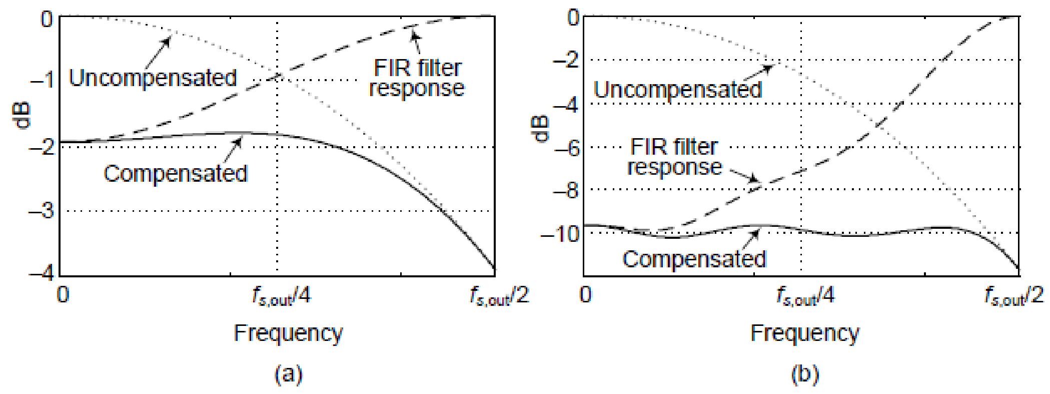


Figure 4.23: Frequency response of a CIC and FIR in a cascaded structure [9]

polyphase FIR filter, as used in [5], which is discussed next.

4.3.4 CIC and Polyphase FIR

Since the frequency-magnitude-response envelopes of a CIC filter are like $sin(x)/x$, a CIC filter is typically followed by a high performance low-pass FIR filter [9].
The concept is illustrated in Figure 4.24. Here the frequency response from the CIC filter and FIR filter are shown separately and combined.
Do note that the response of the FIR filter is up sampled, for visualization purposes, and thus it has images at the higher frequencies.

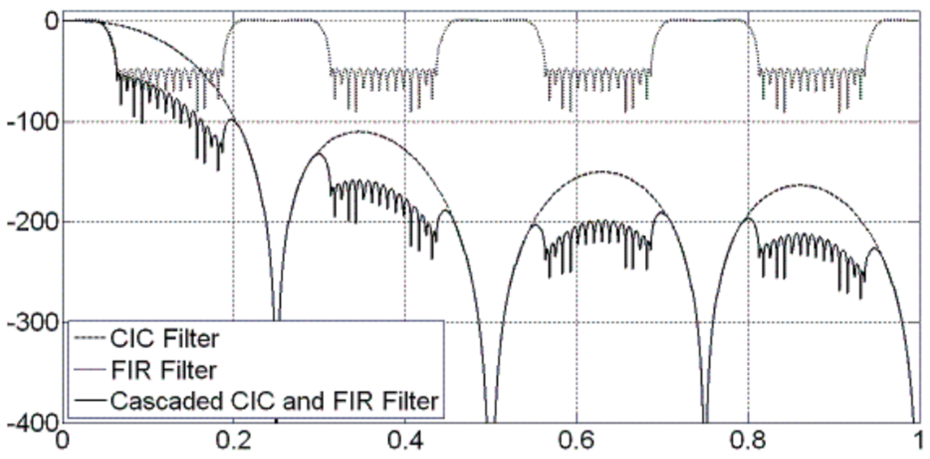


Figure 4.24: Frequency response of CIC, FIR and a cascaded Structure [10]

In such a way a large decimation step, and suppression of high frequencies, is performed with the CIC, which is low-quality but computationally efficient. The FIR

is used to perform the final filter step with high quality.
This approach reduces the frequency and order on which the FIR needs to operate and thus it can be more area and energy efficient compared to if it would be used without a CIC in front.
From preliminary experiments (not shown here) it was found that a configuration of N = 1 and K = 2 combined with a 8 taps polyphase FIR would be desired with respect to its frequency transfer.

The CIC however produces multi-bit values at its output, therefore the polyphase FIR will require multipliers instead of switches, making the polyphase FIR less efficient. This trade-off is elaborated further in the comparison section below.

4.3.5 Comparison

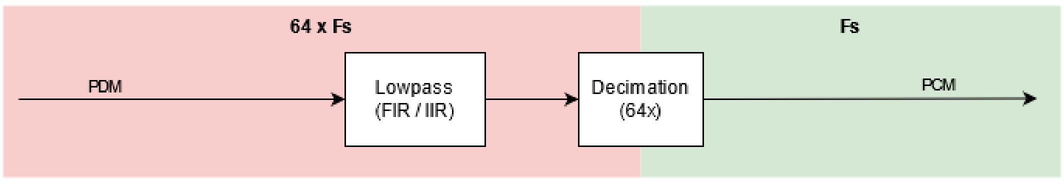


Figure 4.25: Decimation using FIR

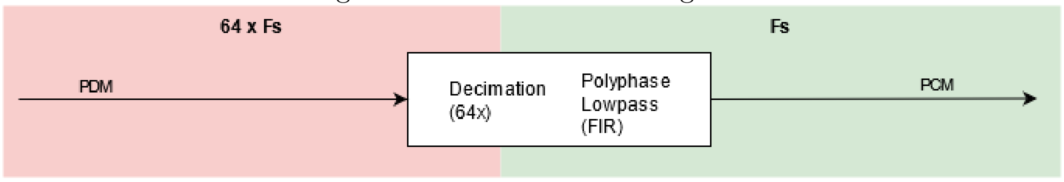


Figure 4.26: Decimation using Polyphase FIR

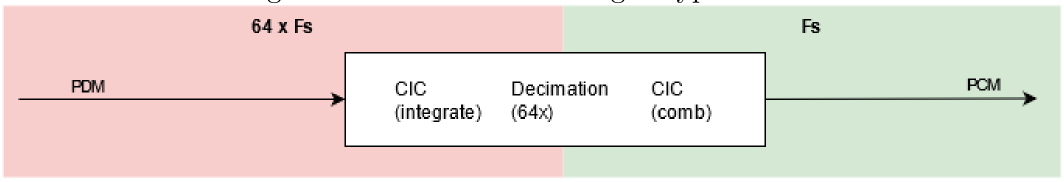


Figure 4.27: Decimation using CIC

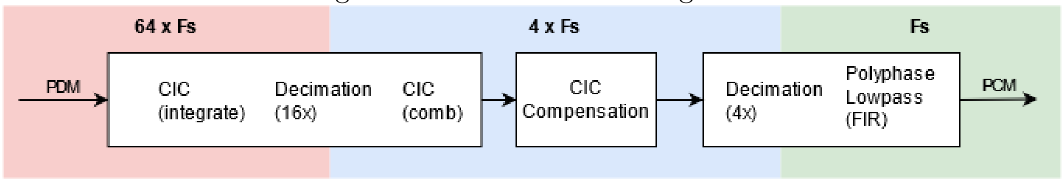


Figure 4.28: Decimation using CIC + Polyphase FIR

All the filters under consideration: FIR, polyphase FIR and CIC have linear phase and are stable thus only the magnitude frequency responses are provided and used in the comparison. Moreover, the frequency response of each of these filters, with their specific configuration, are deemed to be sufficiently functional accurate as mentioned

Decimation 64x	**Adders***	**Multipliers**	**Storage (bits)**	**Add/s**	**Mult/s**
FIR, 128 taps	127	0	127	65.02 M	0
Polyphase FIR, 128 taps	2	0	78	1.02 M	0
CIC, N=1, K=3	6	0	114	1.56 M	0
CIC, N=1, K=2 + p-FIR, 8 taps **	6	2	66	1.10 M	16 k

Table 4.1: Cost comparison, corresponding explanation and equations are listed in section C.1, *size depends on the word size, **CIC at 16 times decimation and p-FIR at 4 times decimation, where p-FIR stands for polyphase FIR

before. Note that there is no strict known SNR requirement since the output data is eventually interpreted by an neural network that still has to be defined. This comparison only takes into account the efficiency of the implementation from this point on.

The base case, a direct FIR implementation is illustrated in Figure 4.25. The improved decimation methods under consideration are schematically listed in Figures 4.26 until 4.28 for polyphase FIR, CIC and CIC + polyphase FIR respectively.

The computational complexity and area cost for each method is given in Table 4.1. The input is a 64 times oversampled 8 kHz signal.

Note that the size of the adders can vary widely between solutions and even for different places in the design. Moreover it is assumed here that the coefficients for the (polyphase) FIR are hard-coded in the design, so no registers are used for those. Taking into account the area and computational cost, in Table 4.1, it is clear that the polyphase FIR is the most efficient.
In the next section the hardware design for the polyphase FIR is elaborated.

4.4 Hardware design

In this section the hardware design of the polyphase FIR is given. Inspiration for this design originated from the Matlab implementation diagram, shown in Figure 4.29.

As seen in Figure 4.10 and schematically in Figure 4.30, the input samples are separated in different arrays. From Figure 4.29 it is noted that they can be rearranged as a tapped delay line in order to reuse similar hardware.

The hardware design, as implemented in Hardware Description Language (HDL) is shown in Figure 4.31. Note that the content of the dotted box closely resembles a direct form FIR but has additional delays. Moreover, the final sum is changed to an accumulator with a reset since it was recognized that the large final sum can be spread over multiple cycles.

Note that in the general design, Figure 4.31, there are multipliers in the design. But in the case of 1 bit input samples, as is the case with PDM, the multipliers should be replaced with switches.

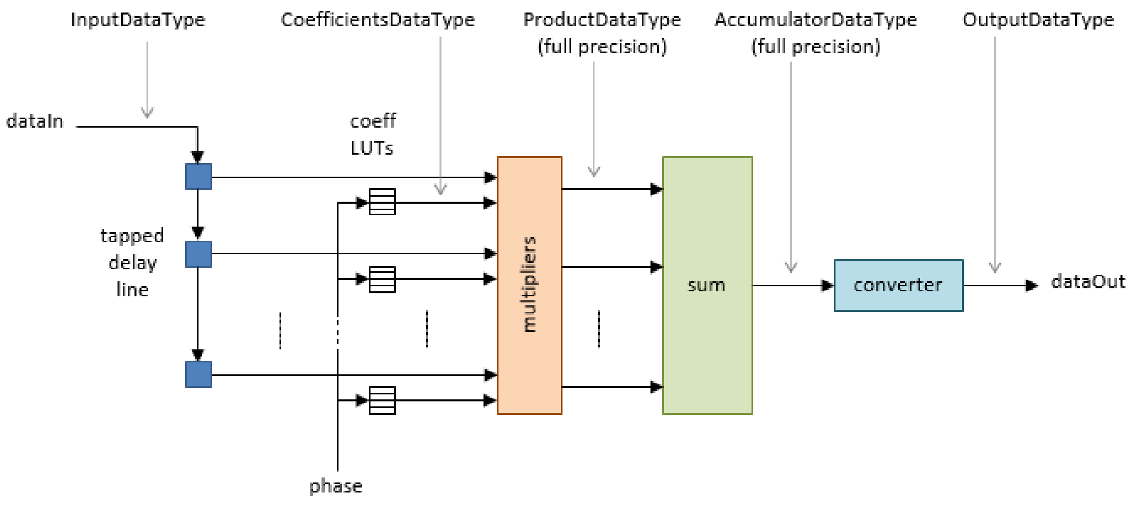


Figure 4.29: Matlab polyphase FIR implemenation diagram [11]

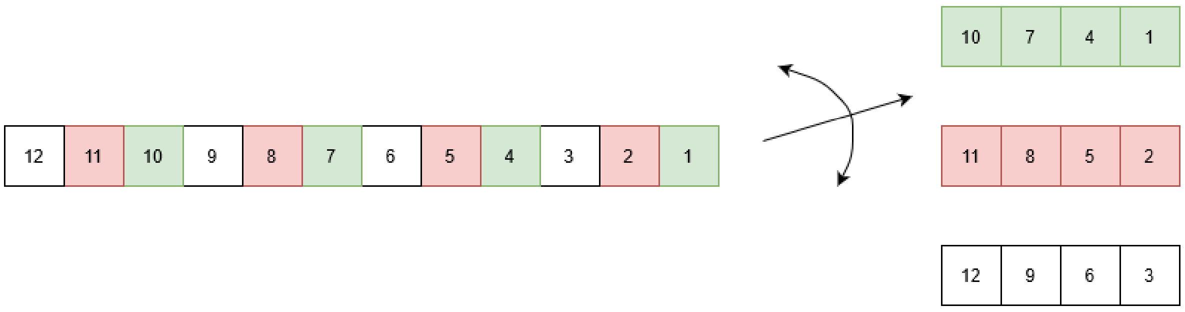


Figure 4.30: Separate the incoming PDM samples into different buffers

4.5 Conclusion

Multiple methods for decimation were investigated but the polyphase FIR turned out to be the most efficient for our use case. For the polyphase FIR a hardware design is designed, implemented and verified in HDL.

There are two unknowns left. First, it is not yet known how high the oversampling rate should be. Secondly, the amount of taps are still to be determined. Both are a trade-off between power and classification accuracy, which requires additional information and will be investigated in chapter 9.

This chapters contributes with an efficiency analysis of a PDM to PCM converter for audio.

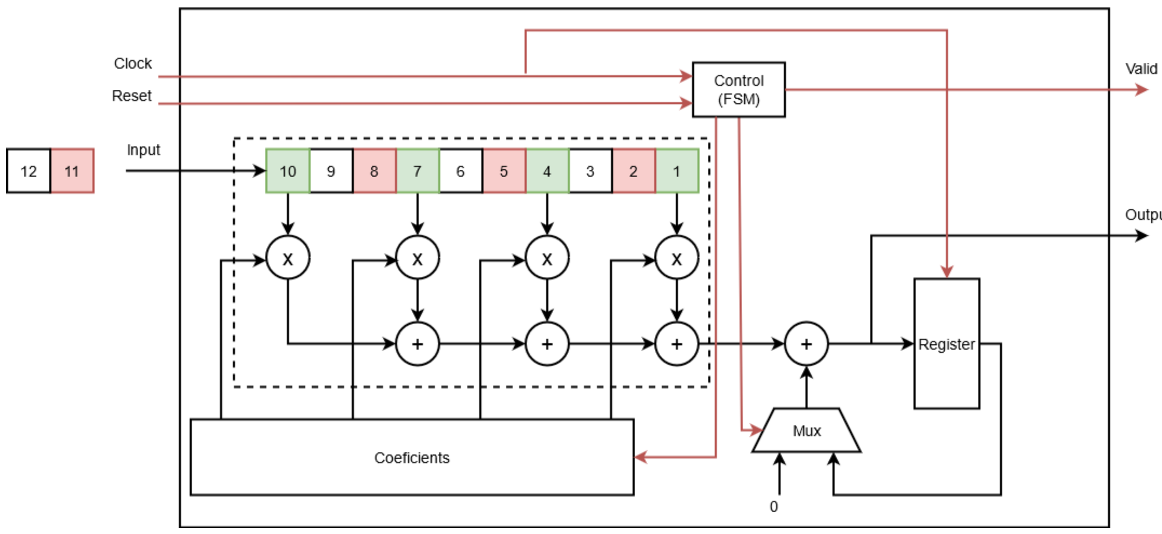


Figure 4.31: Hardware design of the polyphase filter, 12 taps and 3x decimation, where the red lines indicate control wires.

Filtering: The Filterbank 5

The second stage in the pipeline is the Filtering module. In this module a frame of PCM samples is converted to a single spectrogram column by utilizing a filterbank. The filtering can be performed in either time- or frequency-domain. As a starting point the time-domain filtering as described in [1] is analyzed, afterwards a translation to frequency-domain and further optimizations are proposed. Finally, a comparison is given and the most efficient pipeline is chosen.

5.1 Mel-scaling

The filter cut-off frequencies for the filterbank are determined by using the mel-scaling. This method is suggested in literature [3] in order to approximate the frequency response behaviour of a cochlea.

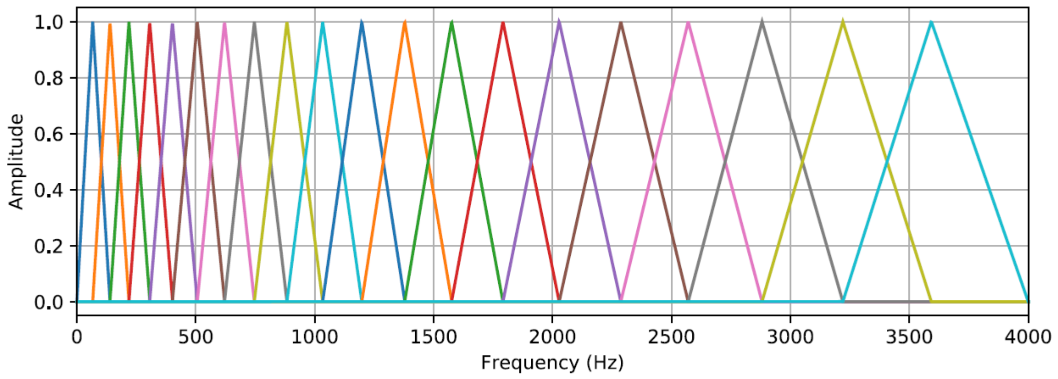


Figure 5.1: Frequency spectrum of the mel-scaled filterbank, repeated for convenience [1]

5.2 Time-domain

Time-domain filtering is used in [1] for implementing the mel-scaled filterbank. The implementation from [1] however requires an entire frame of data to be buffered before computation is executed. To optimize the filtering, a streaming implementation was derived in which the computational load is spread over the entire frame, removing the need for a large input buffer.

An illustration of the computation steps for the revised time-domain filtering is provided in Figure 5.2. The sub-components are highlighted below. Note that the computations, illustrated in Figure 5.2, need to be performed in parallel for each

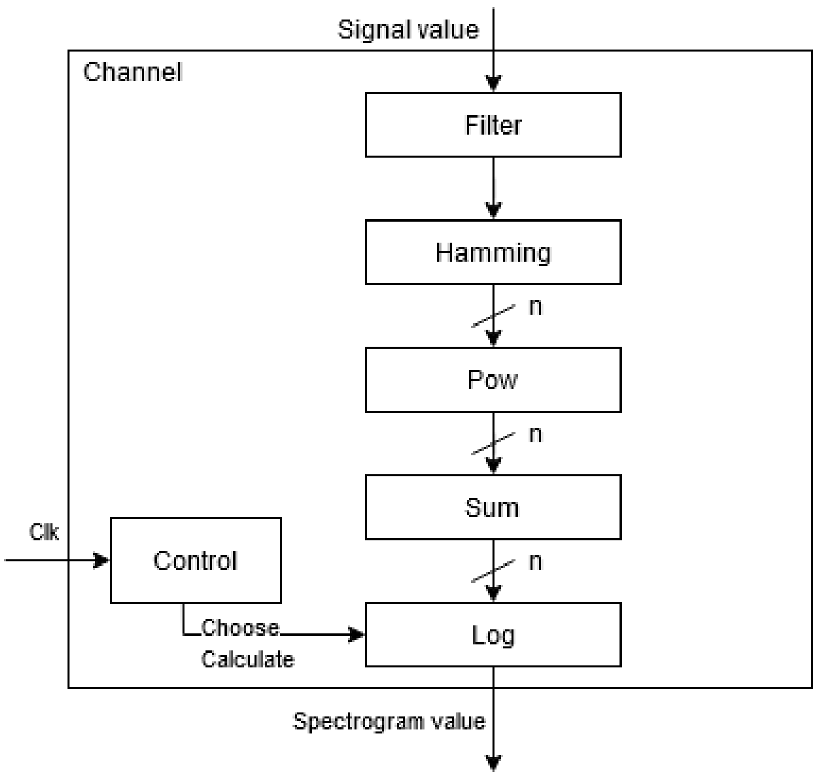


Figure 5.2: Time domain filtering for a single frequency channel, for a multi-channel filterbank most modules have to be duplicated

channel, in this case 20 times.

5.2.1 Filter

Note should be taken that even though the filter bank is described as a triangle filter filterbank in [1], the actual frequency response of the filters are designed with butterworth. An example of the actual frequency response for channel 19 is given in Figure 5.3. To keep the design similar to [1], for comparisons, the filters for the time-domain discussed here are also butterworth filters with their cut-off frequencies determined by the mel-scaled triangles.

5.2.1.1 IIR

A standard filter design can be obtained by using an Infinite Impulse Response (IIR). The corresponding discrete frequency response is described in Equation 5.1

$$H(z) = \text{Gain} * \frac{\sum_{n=0}^{M} b[n]z^{-n}}{\sum_{n=0}^{M} a[n]z^{-n}} \tag{5.1}$$

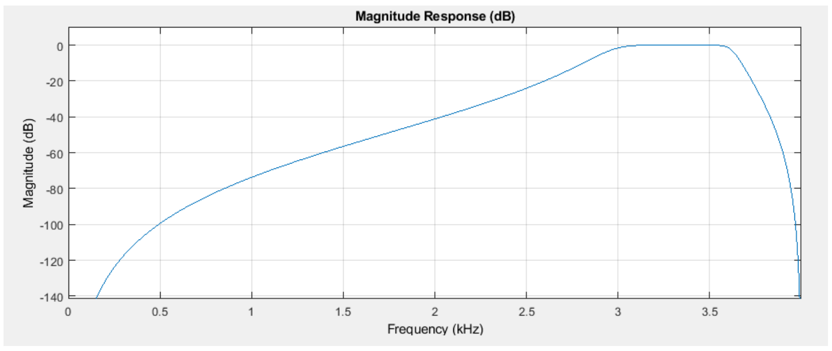


Figure 5.3: Actual frequency response for channel 19

To increase stability, Equation 5.1 can be rewritten to Equation 5.2 [12]. Which is a cascade of second order structures, as illustrated in Figure 5.4. In literature this is called a Second-Order Sections (SOS) filter.

$$H(z) = Gain * \prod_{n=0}^{N-1} \frac{b_n[0] + b_n[1]z^{-1} + b_n[2]z^{-2}}{1 + a_n[1]z^{-1} + a_n[2]z^{-2}} \tag{5.2}$$

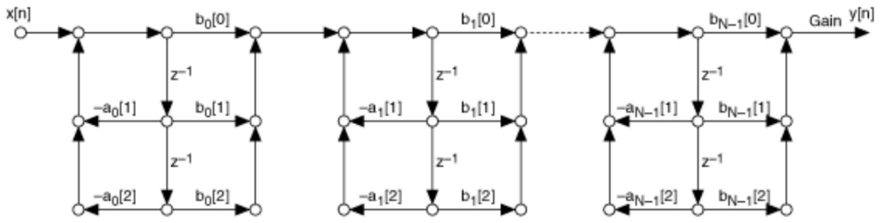


Figure 5.4: Cascade of IIR direct form 2 implementations [12]

From experiments (not shown here) it was seen that this implementation still suffers from finite-word-length effects, causing instability, when less than 32 bits are used for data representation and computation.

5.2.1.2 FIR

To prevent stability issues the Finite Impulse Response (FIR), which is always stable, is investigated.
The FIR, Equation 5.3 is similar to the IIR Equation 5.1, in the sense that the IIR has an additional denominator term, which is the feedback.

$$H(z) = \text{Gain} * \sum_{n=0}^{M} b[n]z^{-n} \tag{5.3}$$

It is widely known though that additional coefficients (taps) are required to obtain a similar frequency response with an FIR compared to an IIR. These additional taps will increase the implementation cost.

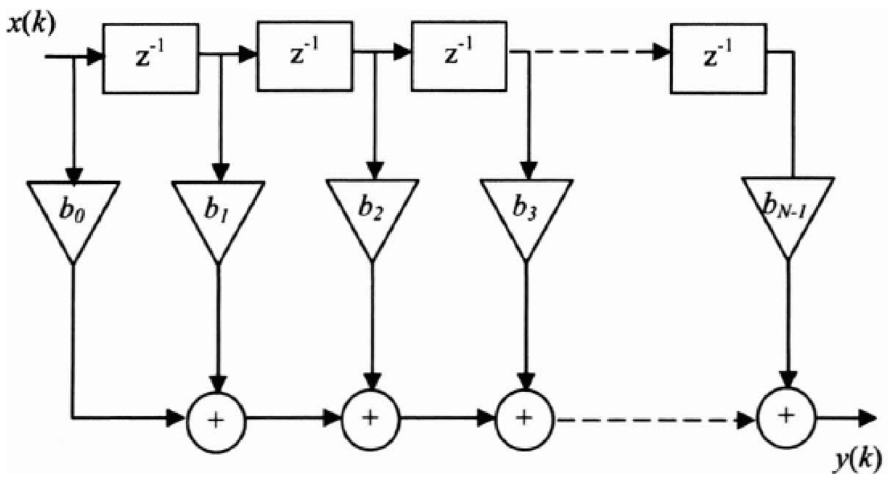
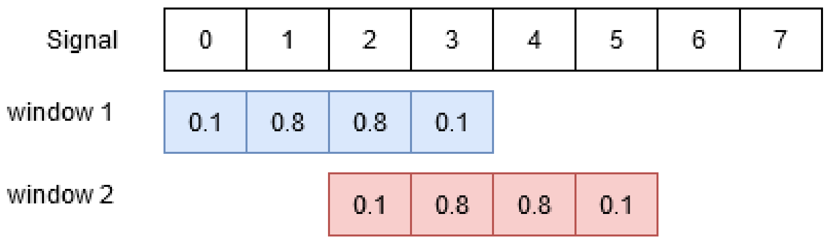


Figure 5.5: FIR direct structure [13]

The FIR has besides stability also a desirable linear phase characteristic.

5.2.1.3 Hamming

In the case of overlapping frames the next module, the Hamming module, generates multiple output values for each input. The same signal value needs to be multiplied by a different Hamming window constant for different frames.
The situation is illustrated in Figure 5.6 using complex numbers. The top row, signal, indicates the incoming samples and the window rows indicate the Hamming constants that are used for one particular signal value. From Figure 5.6 it is clear that signal value 2 needs to be multiplied by 0.8 because of window 1 and with 0.1 because of window 2.



Signal	0	1	2	3	4	5	6	7
window 1	0.1	0.8	0.8	0.1				
window 2			0.1	0.8	0.8	0.1		

Figure 5.6: Hamming window

The amount of output values, and multiplications, from the Hamming window module depend how many windows are overlapping in the worst case.

5.2.1.4 Pow

With the pow module each sample is getting squared, which is the first step of obtaining the power in the frequency range.

5.2.1.5 Sum

The second step of obtaining the power is summing all the squared values. The sum module contains multiple registers to store the intermediate sum until all values for a window have passed. After all values for a particular window passed, it resets the registers linked to that window to prepare for the next window.

5.2.1.6 Log

To obtain the power, the log_{10} should be used on the sum of squares. In the work of Prozée [1] however the natural logarithm is utilized, Equation 2.6. To stay close to the functional implementation of [1], it is chosen to also use the natural logarithm here. A quick comparison revealed only a slight difference in spectrogram output between using the log_{10} and the natural logarithm.
The log is only used every x amount of samples, where x is the stride of the window. To optimize the implementation it should be noted that the computations for the logarithm can be spread over multiple clock cycles, reducing area and power.

5.2.2 Control

The control is used to select a range, the window, over which a summation and eventually the logarithm should be performed. An example for different window lengths and their repetition is illustrated in Figure 5.7.

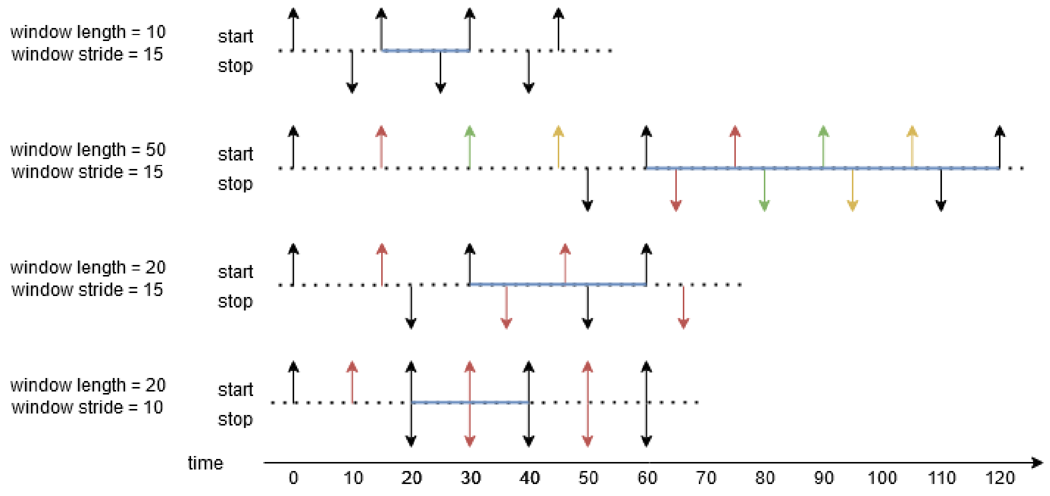


Figure 5.7: Time window control

In order to be able to handle an infinite amount of signal values as input, the periodicity of starting and completing a window calculation is used. This is illustrated in Figure 5.7. Here the upward pointing arrows indicate the start of a window calculation while the downward pointing arrows indicate the window completion. The color of the arrow indicates to which window it belongs. A window is reused when its previous iteration is finished (no overlap with itself) The blue horizontal bar indicates the part which is periodic.

It should be noted that the periodic part only starts after a setup phase because before time zero no window calculations were started and thus no calculations can finish.

5.3 Frequency-domain

Another popular filtering method uses the frequency- instead of time-domain representation of the data. Which uses the fact that convolution in time-domain is the same as point-wise multiplication in frequency-domain [47], mathematically represented as:

$$x * h = F^{-1}(X * H) \tag{5.4}$$

where x and h are two time-domain signals while X and H are two frequency-domain signals and F^{-1} the inverse Fourier transform.

For this filtering approach the following steps are taken:
First, time-domain signals are transformed to Frequency-domain by the Fourier transform, efficiently implemented by using the Fast Fourier Transform (FFT) or DFT for digital domain. [47].
Secondly, the frequency-domain signal is filtered by point-wise multiplicative (mel-scaled) windows.
Lastly, the content of each window is summed and the energy content is estimated in log scale.
These steps are illustrated in Figure 5.8.

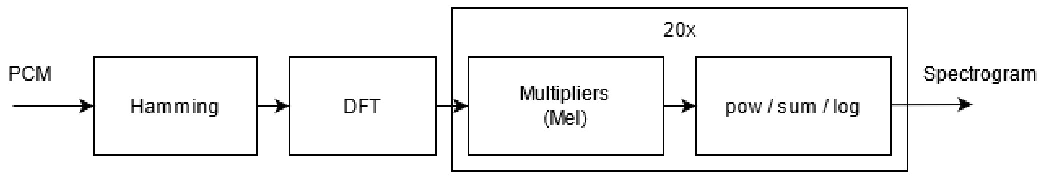


Figure 5.8: Frequency domain filtering

Note that there is a Hamming module in front of the DFT. Its purpose is to reduce the spectral leakage when cutting out a frame from a larger signal.

5.3.1 DFT

The transformation to frequency domain is performed by using a Short Time Fourier Transform (STFT). Which is the short time-domain version of an FFT that transforms the entire signal.

The FFT consists of basic computational elements which can be reused to compose larger DFT lengths. An example of such a basic computational element, the butterfly, for a length-8 DFT is illustrated in Figure 5.3.1. Since the basic elements are independent of each other, a trade-off between area and speed can be made by using less or more of these basic computational elements in parallel.

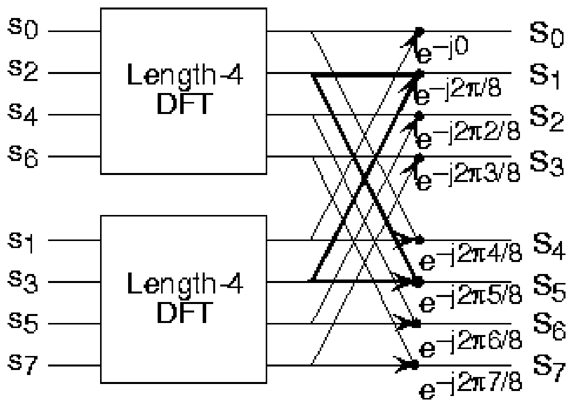


Figure 5.9: DFT of length 8 [14]

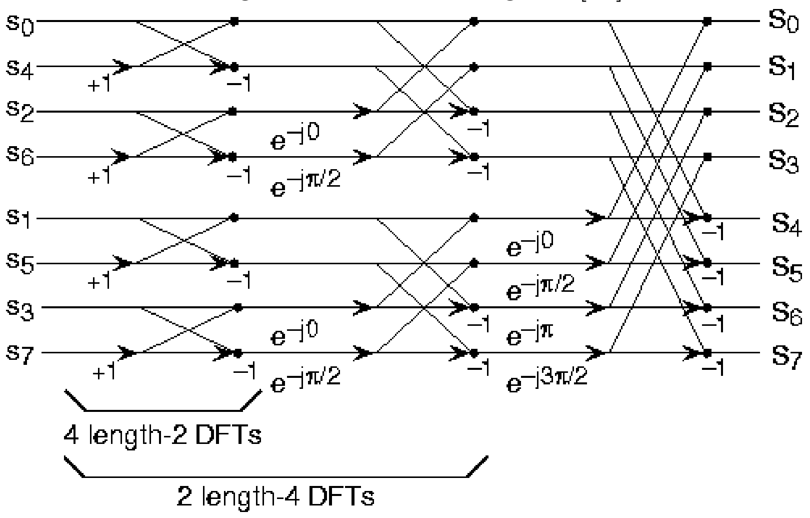


Figure 5.10: DFT of length 8 decomposed [14]

Further implementation details of the FFT algorithm are omitted here since it is out of scope, details can be found in [14]. From [14] it is found that the, frequently used, Cooley-Turkey FFT algorithm requires:
$\frac{N}{4}$ complex multipliers and $2N$ additions for each stage, while $\log_2(N)$ stages are required. Making the computationally complexity $O(N * log_2(N))$ for a real-valued input signal.

In the prototype the KISS FFT library [48] is utilized, which is a mixed-radix FFT library written in C with a BSD license. This library can compute the FFT with a floating-point and a customizable fixed-point implementation.

5.3.2 Triangle windows

Filtering in frequency domain is straight forward since frequencies bands can be obtained by simple multiplications and additions.
Because the filtering is based upon mel-scaling, triangle windows are used to accumulate the intensity over a range of frequencies. The power is calculated similar as in the time-domain filtering, by applying the log of the sum of squares of frequency intensities.

5.4 Others

Other methods that were investigated but will not be elaborated in detail are highlighted here.

5.4.1 BSSP

With Bit-Stream Signal Processing (BSSP) processing is performed directly on a binary signal. This means that a conversion step to PCM is not required. A primary issue with this method is that there is a limited amount of literature about this topic. Especially no literature about band-pass filtering with BSSP was found. Moreover, when topics like low-pass filtering are discussed it is unclear how to straightforwardly obtain coefficients.

5.4.2 Sliding DFT

To improve the efficiency of the DFT calculations a technique called sliding DFT is available. The idea of the sliding DFT is similar to a sliding window for moving averages. The last sample is subtracted while a new sample is added to a certain set of numbers. In such a way a recursive algorithm is used to compute the next DFT (shifted 1 sample) from the previous DFT.

In [49], it is mentioned that:
“if a transform is only needed every M input samples, and all output bins are computed, the computation order would be $O(N * M)$ rather than $O(N * log_2(N))$ for the FFT. The Sliding DFT will still provide an advantage in latency in all cases, however, since after the M^{th} input each output bin can be computed with only two complex-valued operations once $x_k + N$ - x_k is computed.”
Since a DFT calculation is only required every 160 input samples it is seen that a sliding DFT would however be less efficient in our case.

5.5 Comparison

Since both time- and frequency-domain filtering have the same input / output relation, according to the convolution theorem, the frequency responses are identical and thus are disregarded for the comparison.

Filtering	**Add/s**	**Mult/s**
Time-domain, FIR, 20 taps	3.21 M	3.21 M
Frequency-domain, DFT	0.14 M	32.0 k

Table 5.1: Computational cost comparison, corresponding explanation and equations are listed in section C.2.

From the computational cost comparison in Table 5.1 it is clear that the frequency-domain filtering requires significantly less computations. The improvement originates from reusing the results of a single (expensive) DFT operation for each channel. Note that the amount of adders and multipliers are not considered since this depends strongly on the trade-off between area and energy consumption.

5.6 Conclusion

Filtering in both time-domain and frequency-domain is discussed. The computationally complexity of both are compared and it is clear that frequency-domain filtering is preferred. Even though several optimization techniques were investigated, they do not reduce the computationally complexity in this case.

This chapter contributes with an efficiency analysis between time and frequency filtering for audio preprocessing.

Encoding 6

In Spiking Neural Networks information is encoded and processed using spikes. Therefore, to communicate with an SNN, spikes are required. There are several methods to obtain spikes from data. Two main categories are Rate encoding or Temporal encoding. For various reasons, highlighted below, only temporal encoding is considered in this thesis.

The encoders are primarily judged on the following requirements:

- Distinguishability
- Power
- Area

Distinguishability represents the expected usefulness of the data for the SNN to distinguish inputs from different classes.

The input for the encoding stage is the spectrogram, of which each frequency bin is encoded independently over time. The varying input signals shown in subsequent sections are the values of a single frequency bin changing over time.

6.1 Rate versus Temporal encoding

Rate encoding works on the principle that information is encoded in the firing rate of the neurons. Each signal value is translated to a distinct frequency within a window. This is illustrated in the left diagram in Figure 6.1. The benefit of rate encoding is that there is a close match to ANN, making direct conversions from ANN to SNN possible [51].

Temporal encoding works on the principle of encoding information in the time at which the spike is generated. A downside of temporal encoding is that there is no direct correspondence with ANNs. Making the popular back-propagation technique unavailable for temporal encoded signals [51].

From an efficiency standpoint the Temporal Encoding is likely favored in most applications. This is apparent from the density of spikes in Temporal Encoding compared to Rate Encoding. Do note that the sparsity of the spikes depend on the specific encoding algorithm used.

Since low power consumption is a primary requirement for the SNN, only Temporal Encoding is considered in this thesis. There are several temporal encoding algorithms available, some of them that were considered are elaborated below.

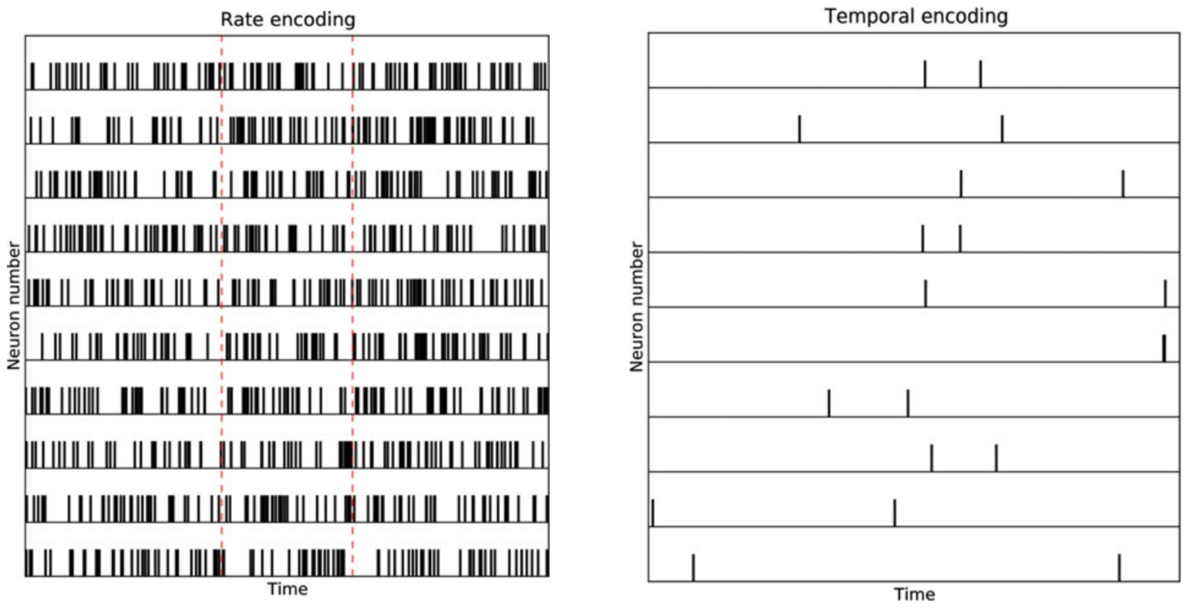


Figure 6.1: Rate- versus Temporal encoding example [15]

6.2 Population Threshold Encoding

In Threshold Encoding (TE) a temporal signal is encoded to spikes according to a threshold. When the temporal signal crosses the threshold, a spike is generated. The spike is positive when crossing from below to above and negative the other way around. From these spikes, the signal can be approximated. To increase the resolution, additional thresholds can be added. Each threshold is represented by an individual node (neuron), when a node is part of a population of multiple nodes. The encoding with multiple thresholds is called population Threshold Encoding (p-TE), which is a variant of TE. [16]

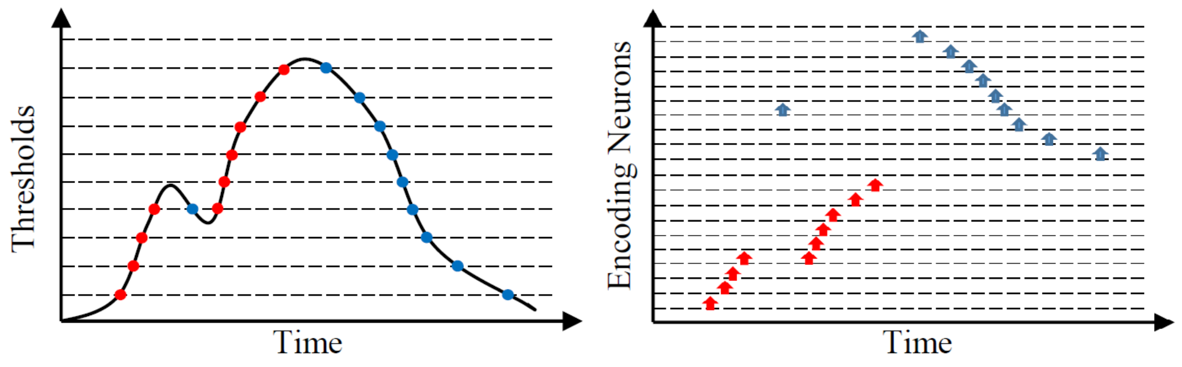


Figure 6.2: population Threshold Encoding (p-TE), adapted from [16]

Note that in an SNN it is not always possible to represent negative spikes directly. A workaround is to separate positive and negative spikes to individual nodes as shown in Figure 6.2.

The p-TE algorithm can be simply implemented by some memories and comparators. Making the algorithm efficient in power and area. Since the p-TE is a direct approximation of the signal it is expected that the distinguishability is not significantly influenced as long as a sufficient amount of thresholds are used.

6.3 Population Step Forward Encoding

The encoding algorithm, p-SFE, is a newly proposed algorithm from the work of de Gelder [17]. Similar as p-TE, thresholds are used to determine when to spike. However with p-SFE the thresholds are not directly linked to a value within the value range of the input signal. Instead, thresholds are used to compare to the changes in the signal. There are multiple variants available, they primarily differ in the complexity of the algorithm. In the most complex variant all thresholds can be freely chosen for each signal separately. In the simplest version, only 2 parameters need to be chosen.

An example of the algorithm is provided in Figure 6.3. Three time steps are shown, each time step is illustrated at a new row.
At step 1, the change (0 to 5) is larger than the threshold (4), thus a spike is generated and the boundary value is set as the next value. Since the next step starts from the boundary of the previous step (4), now the change (1) is within the range (2) and nothing happens. The algorithm repeats itself for the following time steps.

In [17] the algorithm was tested and compared according to decoding accuracy. Looking at its performance against other encoding algorithms it appears to hold a lot of potential. Moreover the implementation of the algorithm appears to be efficient. The problem with the algorithm is however that it is difficult to train the parameters. Our experiments did not manage to obtain any significant distinguishability for the SNN.

6.4 Self Organizing Map

In a Self-Organizing Map (SOM) nodes are organized in a sheet-like ANN, where the nodes become specifically tuned to various input signals. Each node acts like a separate decoder for the same input, it is the presence or absence of a generated spike that provides an interpretation of the input. The spatial segregation for different responses into related subsets results in a high degree of efficiency. Which is similar to how a brain functions, since according to modern imaging techniques the brain functions are also localized. [31] An example of a SOM network is illustrated with Figure 6.4 The SOM is, as the name suggests, self-organizing and therefore tuning of the map does not require labeled data. Training of the SOM is unsupervised and done by presenting input vectors. The best matching neuron, based on a distance metric, will fire and moreover inhibit the neurons in its surrounding region. The inhibition region is getting progressively smaller during training.

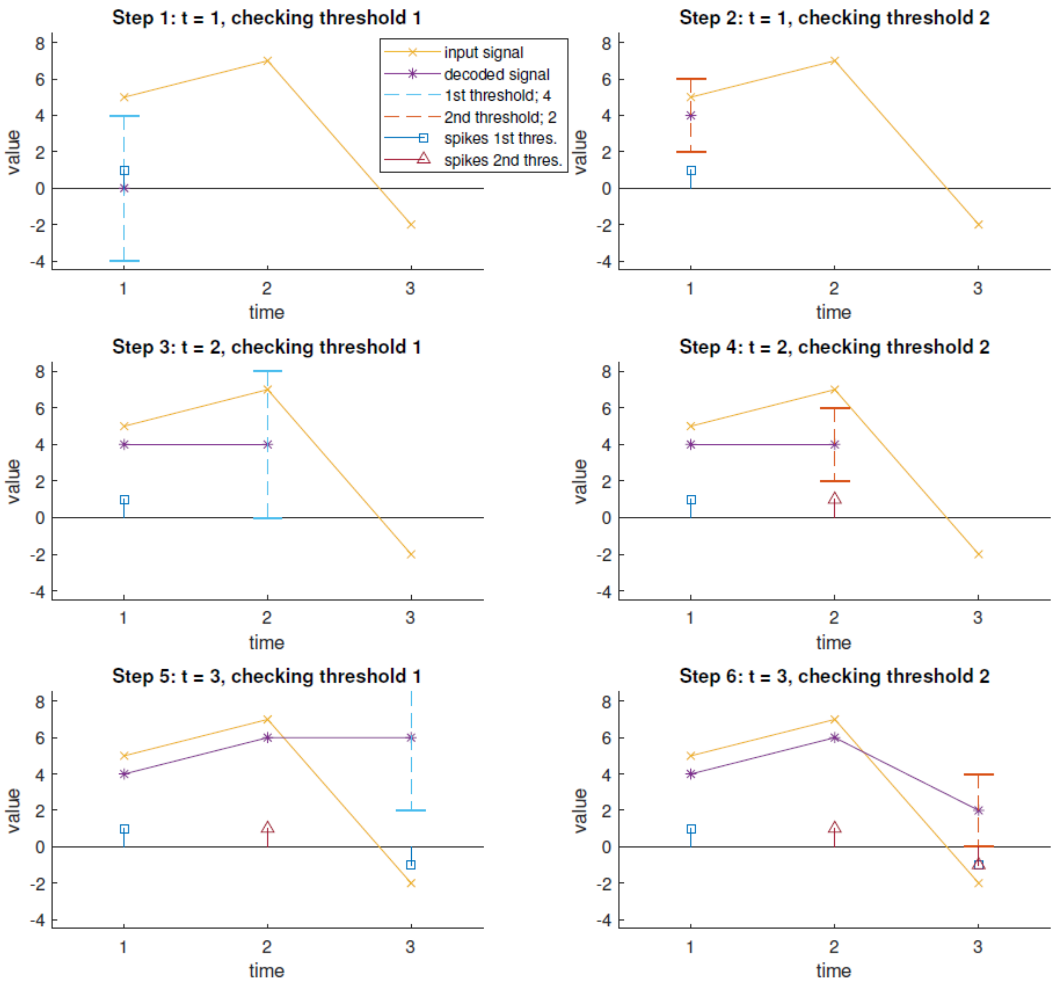


Figure 6.3: population Step Forward Encoding (p-SFE) [17]

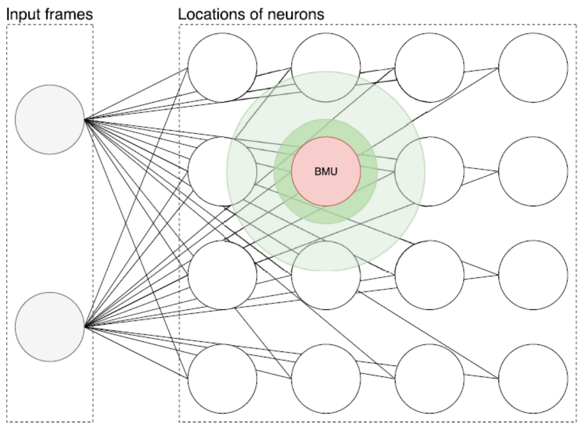


Figure 6.4: Concept of a SOM network [1], repeated for convenience

Even though the SOM can group related subsets, it should be noted that it can not be used as a classifier by itself since convergence is slow. [31]

The SOM is a fully connected network, as evident from Figure 6.4. Therefore, the computationally complexity of the algorithm is $O((N * M)^2)$ [32]. Making it potential unfeasible to be implemented as an encoder because of power and area considerations. From [1] it is however noted that the SOM has a significant positive impact on the classification accuracy of the SNN.

In literature [52][53] there also exists an spiking-SOM variant, perhaps allowing the feature reduction charcteristics of the SOM to be integrated in the SNN fabric, making it more efficient.

6.5 Conclusion

For power and area efficiency p-TE is preferred although for distinguishability the SOM is preferred. At this point in time, a spiking-SOM implementation, one that exists in the SNN fabric, is not yet available. Currently colleagues are researching how to implement a spiking-SOM for the SNN fabric. When utilizing the spiking-SOM implementation an efficient and accurate system might be obtained by using a p-TE as an encoder while applying the SOM functionality in the SNN fabric. The SOM is assumed to be necessary since the Tempotron training rule, highlighted in chapter 7, can only utilize a single layer. For this purpose the spiking-SOM can act as an additional layer.

Training 7

Patterns in the audio signal have to be detected. Traditionally, pattern recognition is performed by handcrafted programs or ANNs. Handcrafting a program is non-trivial while an ANN is inefficient. The SNN, based upon biology, promises to be an efficient pattern recognizer. Training an SNN is however not trivial, multiple algorithms are proposed in literature. Two popular techniques, Spike Time Dependent Plasticity (STDP) [54] and Tempotron [19] are mentioned here.

7.1 STDP

Most biological models of learning and memory in neural networks are based upon Hebbian plasticity [27]. Which is captured in an often paraphrased sentence as “Neurons that fire together wire together.” But synapse-specific learning with Hebbian plasticity, i.e. long-term potentiation (LTP) and long-term depression (LTD), require global regulatory processes in order to generate realistic results. [18]

One such biological regulatory model is STDP, ”This learning mechanism is not sensitive to overall input firing rates or variabilities, but selectively strengthens groups of synapses whose activities are correlated over short time periods” [54].
The STDP learning rule is assumed to depend on the interplay between NMDA receptor activations and the backpropagation of action potentials from postsynaptic neurons.
Depending on if the pre-synaptic spike precedes or follows the post synaptic spike, LTP or LTD is induced respectively, illustrated in Figure 7.1 and provided mathematically in Equation 7.1 [54]. Note that the STDP weakens exponentially with the distance from the postsynaptic spike. [18]

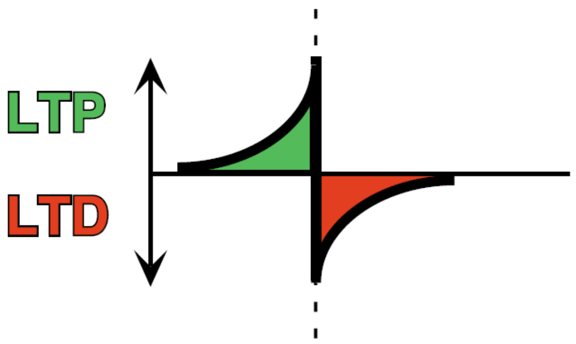


Figure 7.1: Depending on the order of the pre-synaptic spike with respect to the postsynaptic spike, at the vertical dashed location, LTP or LTD is induced. Adapted from [18]

$$F(\Delta t) = \begin{cases} A_+ \ exp(\frac{\Delta t}{\tau_+}), & \text{if } \Delta t < 0 \\ -A_- \ exp(\frac{-\Delta t}{\tau_-}), & \text{if } \Delta t > 0 \end{cases} \tag{7.1}$$

A simple learning rule is to define two functions $P_{pre}(t)$ and $P_{post}(t)$ that satisfy:

$$\tau_+ \frac{dP_{pre}(t)}{dt} = -P_{pre}(t) \tag{7.2}$$

$$\tau_- \frac{dP_{post}(t)}{dt} = -P_{post}(t) \tag{7.3}$$

When the pre-synaptic terminal receives an action potential $P_{pre}(t)$ is incremented. While when the postsynaptic neurons fires $P_{post}(t)$ is decremented. [18] Now $P_{pre}(t)$ determines how much the synapse is strengthened if the postsynaptic neuron fires at time t. While $P_{pre}(t)$ indicates the synapse weakening if the pre-synaptic terminal is fired at time t.
Note that STDP only requires local knowledge for updating its synapse strength while it still appears to solve the problem of competition between neurons, which the Hebbian learning has.

STDP is assumed to stabilize the Hebbian modifications by two opposing forces. First, LTD dominates over LTP when the average neuron membrane voltage exceeds the threshold. Secondly LTP is more likely since pre-synaptic spikes are more likely compared to postsynaptic spikes. [18] Thereby converging the neuron to a state where the neuron is only temporarily sensitive, for certain patterns.

Since only local information is required, it can learn even in a complex, multi-layer, neural network. However, since in practical implementations (not shown here) many hyper-parameters are involved, learning can be challenging.

7.2 Tempotron

From experiments in literature it is found that, instead of the rate, the timing of action potentials carries information. This introduces two important questions: First, how do neurons learn to read information from temporal code. Secondly, what kind of spike patterns are used to compute a response. [19] To bridge the gap between the notion of spatio-temporal spikes and supervised learning, Tempotron learning was introduced by Gütig and Sompolinsky [19]. Tempotron is a supervised biologically plausible model that can learn to categorize a broad range of input classes.

To obtain the sub-threshold membrane voltage each postsynaptic potential (PSP) can be summed as the following [19]

$$V(t) = \sum_i \omega_i \sum_{t_i} K(t - t_i) + V_{rest} \tag{7.4}$$

where t_i indicates the spike time of the ith synapse and K is defined as

$$K(t - t_i) = V_0(exp[-t + \frac{t_i}{\tau}] - exp[-t + \frac{t_i}{\tau_s}]) \tag{7.5}$$

to model the PSP of each synapse. The decay constants of membrane integration and synaptic currents are represented as τ and τ_s respectively. When $V(t)$ is crossed, the neuron will fire a spike and reset to V_{rest} by temporarily shunting all subsequent spikes.

To use gradient-descent, a cost function is used. When a pattern did not elicit a spike while it should or when it did spike while it should not the cost is given as $V_{thr} - V(t_{max})$ and $V(t_{max}) - V_{thr})$ respectively. Where the change of ω_i is proportional to the negative derivative. Thus to learn a pattern, the synaptic efficacy ω_i of a neuron is updated by

$$\Delta\omega_i = \lambda \sum_{t_i < t_{max}} K(t_{max} - t_i) \tag{7.6}$$

where t_{max} is the maximum value of the postsynaptic potential $V(t)$. If a neuron did not spike while it should have, ω_i is increased by $\Delta\omega_i$. When a neuron spiked while it should not have, the change is subtracted. [19]
This learning rule is illustrated in Figure 7.2. Note that there are 2 patterns superimposed on top each other. Where the $\oplus$ pattern should elicit a spike while $\ominus$ should not.

From experiments [19] it is seen that the Tempotron can read both time-based and rate-based codes. Where time-based have a higher capacity of learning multiple patterns, $\sim$ 3 versus 2. Moreover, a tolerance of roughly 1 until 10% is observed, depending on the amount of learned patterns.

The Tempotron learning rule can be simulated with an exact integration of the voltage dynamics of a leaky integrate-and-fire neuron, with decaying synaptic currents [19]. It is a learning rule that can converge to a solution quickly, however a supervisory signal which can not be back propagated is required, and thus Tempotron is limited to a single neural layer.

7.3 Conclusion

From discussions with the author of [1] and his results, it is opted to use the Tempotron learning rule in favor of simplicity and quick convergence. The Tempotron learning rule, when preceded with an SOM appears to provide significant classification accuracy, more than 90%, for simple low-noise scenarios.

It should be noted however that the Tempotron learning is limited to only a single layer of weights, which can restrict its ability to distinguish between patterns. When more complicated input data is presented, e.g. multi-keywords or noisy samples, another learning method, e.g. STDP, will likely be necessary.

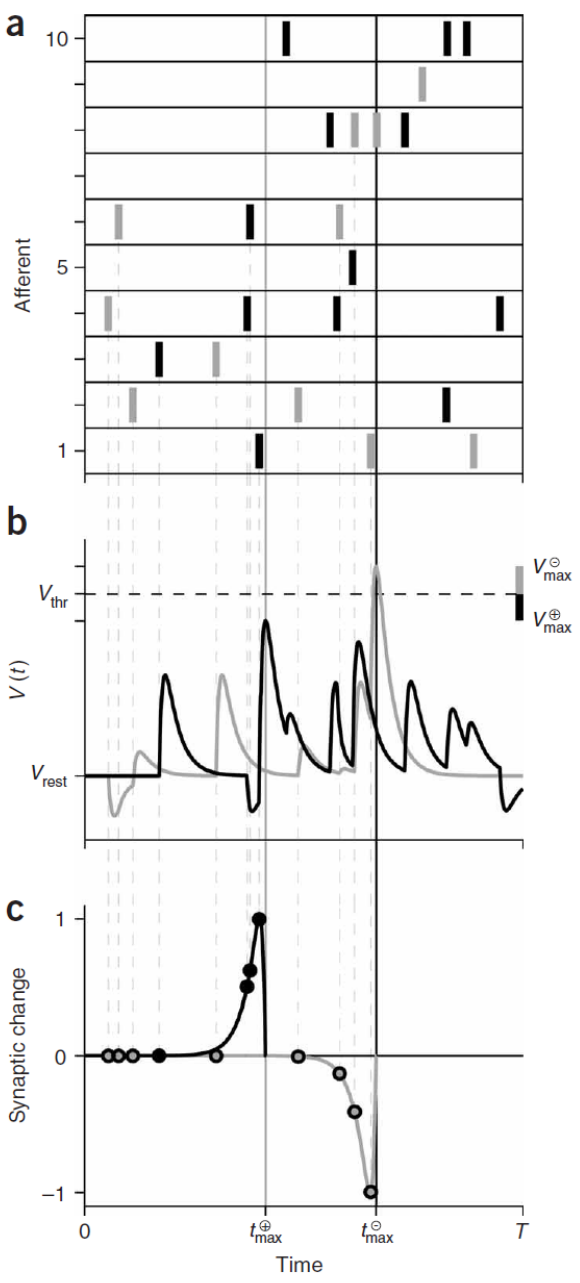


Figure 7.2: The Tempotron learning rule. (a) defines the spikes for a pattern $\oplus$ that should elicit a spike in the neuron and $\ominus$ a pattern that should not. (b) indicates the membrane potentials over time and (c) the change of ω for each afferent (synapse). As provided in [19].

Prototype 8

Each module, as defined in chapter 3, will need a physical device upon which the computations are performed. In this chapter the prototype design is presented which lists the necessary hardware, how they are interconnected and where the previously mentioned modules are used. A summary of the hardware and its high-level interconnections is provided in Figure 8.1 while the experimental setup is shown in Figure A.2.
The following sections will elaborate for each component in Figure 8.1 its function. Afterwards, several possible data flows are highlighted.

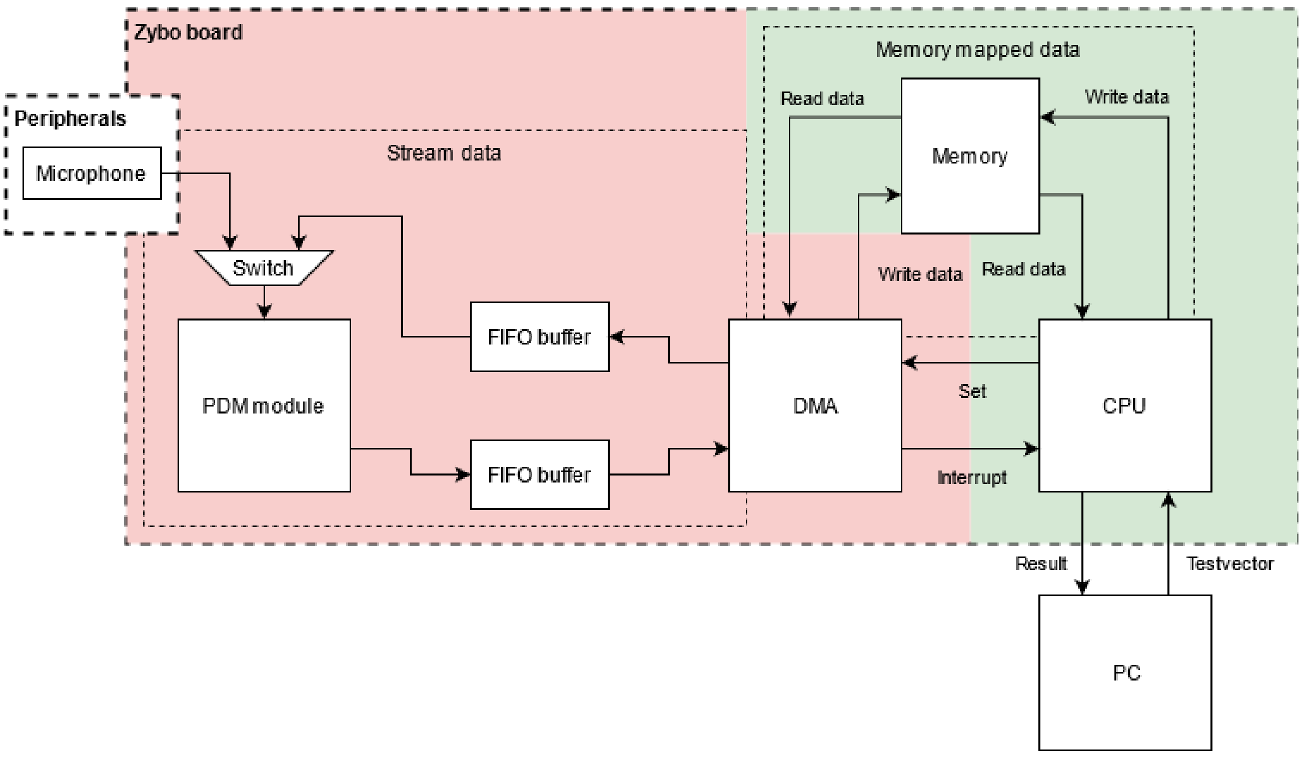


Figure 8.1: Design overview

Note that for this prototype a Zybo board, PC and microphone are utilized as actual hardware devices. Some of the hardware devices are broken down in smaller components in order to illustrate their usage.

8.1 Zybo board

The Zybo board [55] contains two relevant sub-components, the Field Programmable Gate Array (FPGA) and the Central Processing Unit (CPU). They are independent of each other and can run at different frequencies, inter communication is available via Direct Memory Access (DMA).

The Zybo board design for the audio preprocessing, illustrated in Figure 8.1, is separated in two regions: the components in red are operated at a high oversampled frequency while the components in green are run at the sampling frequency. Moreover two dotted regions, stream and memory mapped data, indicate if the component executes upon single sample data or frames respectively.

8.1.1 FPGA

The Field Programmable Gate Array (FPGA) is a device which can provide cycle accurate and fast simulation of dedicated hardware by programming gate arrays.

8.1.1.1 Switch

The switch is configurable from the CPU and provides the ability to choose between live data or recorded data.

8.1.1.2 PDM module

This module represents the implemented module described in chapter 4.

8.1.1.3 FIFO buffer

After the DMA completed a transfer, the CPU has to instruct its next task. Since the CPU can be busy, a short interruption in the data stream can occur. To mitigate this, First-In First-Out (FIFO) buffers are added to allow the CPU to catch up to the stream when it is not busy with other tasks.

8.1.1.4 DMA

This module is an Intellectual property (IP) block that uses the Advanced eXtensible Interface (AXI) and AXI stream to handle data transfer between FPGA and CPU

8.1.1.5 Peripheral: Microphone

The actual physical device that produces a PDM data stream.

8.1.2 CPU

The CPU is a dedicated processor which has a tight integration with the FPGA and can handle high-level communication like Universal Asynchronous Receiver-Transmitter (UART). Moreover it can be programmed bare-metal in standard C, allowing general processing with real-time constraints.

This component is used to execute the filtering module and transfer the data to subsequent systems, e.g. PC or other hardware device.

	Time (us)
Bytes to Floats	28
DFT-160	112
DFT-256	135

Table 8.1: Zybo board CPU timing per stride: Bytes from the DMA are processed to floats and thereafter processed by a 160 or 256 DFT.

8.1.2.1 Memory

Storage space for (temporary) data used both by the CPU and the FPGA via the DMA.

8.1.3 Timing

The filtering module should be run at low frequency and preferable with floating-point support while the pdm-pcm module only requires a high frequency. Using the Zybo board both requirements can be met by utilizing both the FPGA and CPU at different operating frequencies.

When using a STFT stride of 10 ms, the real-time constraint for a real time system requires that all computations for a frame are performed within this 10 ms bound. From timing measurements, provided in Table 8.1, it is seen that the contribution of the filtering module is well within this bound.

8.2 PC

The PC is used for storing, evaluating and providing data, i.e. test vectors to the device. Since the data can be send to the PC in real time, as shown in Table 8.1, real time processing is available. Depending on the program this can be as a visualization of the spectrogram columns or even as classification accuracy estimations from the SNN simulator, running in a custom made streaming mode.

Note that the PC is also used for simulations and to train the networks.

8.3 Interconnections

8.3.1 AXIS

A common protocol for interconnection is the Advanced eXtensible Interface (AXI). The AXI is a "parallel high-performance, synchronous, high-frequency, multi-master, multi-slave communication interface, mainly designed for on-chip communication" [56]. The AXI protocol has a streaming variant called AXI-Stream, which is optimized for streaming data and has a simpler handshake protocol. [20]. An example of the AXI-stream protocol is illustrated in Figure 8.2.

Assuming that data is sent from another device to this particular interface, all lines are input except the TREADY. With the TREADY the data rate can be con-

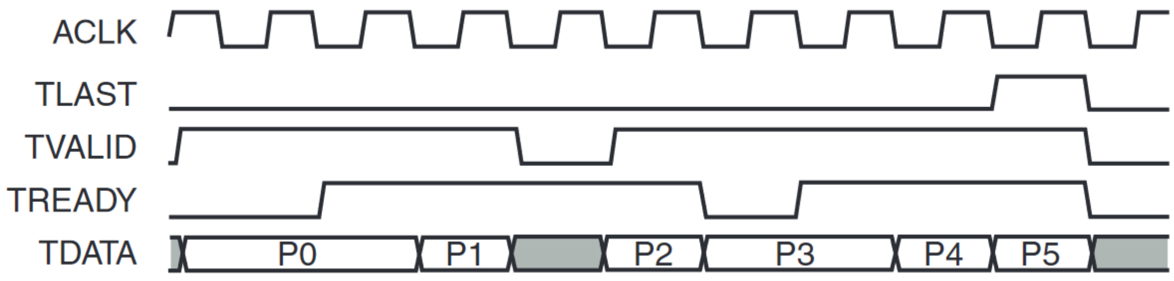


Figure 8.2: AXI-Stream example [20]

trolled. Only when TREADY is asserted there will be a new data packet. The stream is ended by the TLAST from the sending device, indicating that the stream is complete.

To communicate with other devices, this protocol had to be implemented in the PDM module.

8.3.2 DMA

Usually when data is moved, the CPU needs to explicitly move all the bytes and wait until the operation is finished before continuing. Since the CPU is used for a long processing task and also small data movements, the CPU will need to frequently switch between tasks in order to meet all constraints, as illustrated in Figure 8.3.

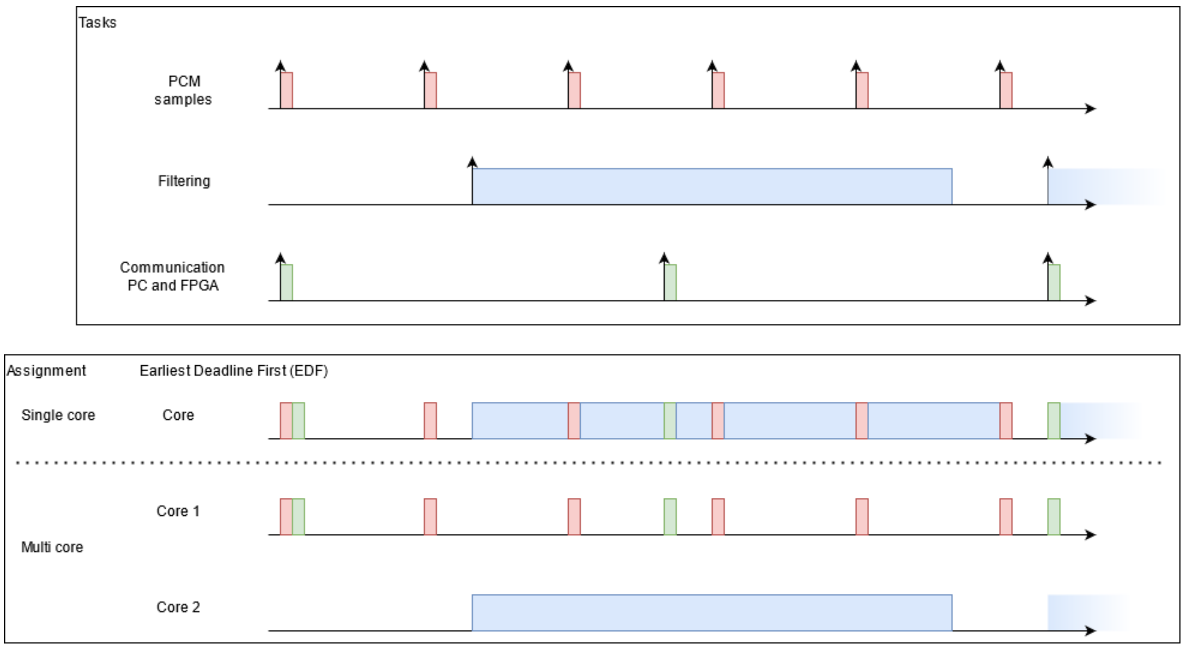


Figure 8.3: Scheduling example

To offload data transfers to and from the memory, a component called Direct Memory Access (DMA) can be used. There are multiple variants of the DMA but only

a simple version is considered here.
To send data from the CPU to the PDM module, the DMA is given an instruction to copy a range of bytes starting from an address to another location, in this case a FIFO buffer.
To receive data at the CPU, the DMA is given an instruction to copy values from a location, in this case a FIFO buffer, to a memory location.
When the DMA finished its instruction it interrupts the CPU to notify it has completed its assignment. The CPU can perform other tasks while the DMA performs the data transfer. After each interrupt the CPU needs to reassign the DMA for the next amount of bytes to copy.

Since the DMA is not always assigned a task, such as between a finished interrupt and the next assigned task. The values that are produced or need to be provided to the PDM module need to be buffered in order to not lose samples.

When using the DMA no Real-Time Operating System (RTOS), e.g. FreeRTOS [57], is required in this case.

8.3.3 UART

Communication between PC and Zybo board is supported through the UART protocol. The connection is established through the USB protocol. This means that the PC can connect to a COM port while the Zybo board can use its USB-UART bridge [55].

The Zybo manual [55] defines the UART protocol with the following parameters: 115200 baud rate, 1 stop bit, no parity, 8-bit character length. Which should be matched by the PC.

The specified parameters limits the data rate to:

$$\text{rate} = \text{baudrate} * \frac{8}{9} \text{ bits/s} \tag{8.1}$$

thus the theoretical transfer rate limit is 102400 bits/s or 3200 32 bit values per second. Taking into account a stride of 10 ms, this means that a maximum of 32 values (32 bits each) can be send each stride. From experiments it was determined that the actual transfer rate is around 92000 bits/s, thus only around 28.75 values (32 bits each) can be send per stride.

It is known that the sampling frequency is 8000 Hz, oversampling is 64 and stride is 0.01 s. Thus in total 5120 PDM samples (bits) are produced each stride, according to

$$\text{samples} = \text{sampling frequency} * \text{oversampling} * \text{stride} \tag{8.2}$$

Which is larger than even the theoretical transfer limit of 1024 bits per stride, making real time streaming of the PDM values infeasible.

The low-pass filtered PCM values can also not be transferred in real time, preliminary experiments indicated that around 20 bits are required for each PCM sample and each stride has 80 samples thus it would requires a transfer of 1600 bits per stride.

The spectrogram columns can be transferred in real time. Looking at new values for each stride, 20 new values are generated. This is within the bounds of the 28.75 and thus is feasible in real time.

8.4 Data flow

In this section different desired data flows are highlighted and their path through the prototype is described. Restating the importance of each part.

8.5 Forward

The forward data flow starts from the microphone, as selected by the switch.

8.5.1 Pass

This flow is required to obtain PDM values from the actual microphone. To be used for future training and testing.

In the forward pass, PDM data is unfiltered passed to the FIFO buffer. Where 32 subsequent PDM samples are packed in a single 4 byte word. The DMA reads the word from the FIFO and copies it to the memory. After sufficient words are stored by the DMA, the CPU is interrupted and the words are send to the PC via UART. The PC can now store these PDM values and use it later, e.g. as test vector or as simulation input.

8.5.2 Spectrogram

This flow is required to perform evaluations and optimizations of the prototype. In particular for the timing of the CPU on the Zybo board. Moreover, it turns out that the spectrogram columns can be sent to the PC in real time via UART, enabling a real time prototype.

Starting from the microphone, PDM samples are passed to the PDM module, the PDM samples are low-pass filtered by the PDM module and passed as PCM values to the FIFO buffer. The DMA copies the PCM values to memory and after each 80 copies it will interrupt the CPU. The CPU will combine the previous 80 samples with the new 80 samples in an array of 160 samples, because of overlapping frames, upon which the filter module is used. After the completion of the filter module, the spectrogram column is send to the PC via UART.

8.5.3 Encode

This flow is intended for future usage, in particular when a hardware version of the SNN is available.

The forward encode data flow is similar as the forward spectrogram flow. However, instead of sending a spectrogram column to the PC via UART. It should encode the spectrogram column to spikes, using an encoder module, and buffer these spikes in memory. After sufficient spike time slices are gathered, the spikes are sent in a burst fashion to the SNN.

8.6 Loop

This flow is required for functionality testing. Test vectors can be supplied to the system after which the results can be compared to golden references. Ensuring the correct implementation of the prototype.

The loop data flow start from the PC. Test vectors are supplied to the CPU on the Zybo board via UART, where the test vector is an array of PDM bits that are packed in words of 4 bytes. The CPU buffers a complete set of the test vectors and instructs the DMA to provide them to the PDM module via the FIFO buffer, still packed as 4 byte words. The PDM_module unpacks the PDM values and supplies them to the low-pass filter. From this point on the samples will follow the forward spectrogram data flow.

8.7 Optimization

From the prototype, several areas for optimization are identified and mentioned below.

8.7.1 Filter order

The filter order of the low-pass filter, used in the PDM_module, was estimated by observation of the spectrogram quality. The spectrogram quality however depends on the input PDM values and moreover is assumed to not directly relate to the SNN classification accuracy. Using samples from the microphone, a sweep of different order values need to be performed to perform a trade-off between area and power versus quality.

8.7.2 Bits-width

Due to simplicity of the hardware and reduced area and power, a fixed-point format is used to represent fractional values in the PDM module. A minimal amount of integer bits is necessary to prevent overflow, the amount of fractional bits does not have a strict bound. For the fractional bits a trade-off between amount of bits and classification accuracy should be made.

8.7.3 Signal Power Detector

During experiments it was noted that most of the input should be classified as silence. When there is silence, the signal power in the input data is substantially lower compared to when a number is pronounced.
As mentioned in subsection 2.4.1 for [4], adding a module to detect signal power and only pass its data to subsequent modules if a threshold is reached can substantially increase energy efficiency of the entire design.
Care should however be taken in order to find a balance between energy consumption of the power detection module and its miss detects and false alarms. In [4] the power of the signal is estimated by

$$P = \sum_{i=1}^{N} |x_i| \tag{8.3}$$

with N the amount of samples taken into account and x_i the actual sample, assuming a FIFO buffer.

Since the prototype design, as described in this thesis, expects complete frames of samples and moreover signal power is expected to be localized in small regions of a frame, a custom signal power detection algorithm is developed.

The design is based around a single FIFO buffer that can store an entire frame, as simplified in Figure 8.4. Where the numbers indicate the ordering of the input

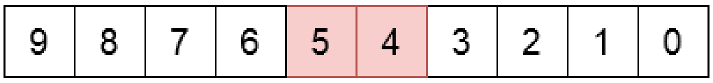


Figure 8.4: Signal power detection FIFO buffer, simplified

samples and the colored region in the middle indicates the samples considered for power estimation. If there is power detected in the designated region, the entire frame of samples currently in the buffer will be passed. If there is no power detected, nothing will be passed. If there are multiple short power regions in a frame, one or multiple complete frames are passed subsequently.
In such a way, the keywords will be mostly centered in a frame and only frames with energy are passed.

For energy efficiency the FIFO buffer is implemented as a ring buffer, using pointers to keep track of the state. Moreover the power estimation region is using a running average, only taking into account the newest and oldest in the region for each sample update.

8.8 Conclusion

The prototype illustrates a working real time example in hardware of the software simulation. The idea is that for the eventual target hardware, everything implemented in the FPGA will eventually be implemented on an Application Specific Integrated Circuit (ASIC) while the modules on the CPU of the Zybo board will run on a general embedded processor. Although the prototype is functionally working, further optimizations to reduce area and energy consumption are still required. These optimizations are performed and evaluated in the next chapter, chapter 9.

This chapter contributes with a functional hardware prototype of an audio pre-processing pipeline compatible with a SNN.

Evaluation 9

9.1 Dataset

Several datasets are available for simple command recognition, those that were considered are highlighted below.

9.1.1 TIDIGITS

This is a dataset that was collected at Texas Instruments. Its purpose was to design and evaluate algorithms for speaker-independent recognition of connected digit sequences [58]. A variety of speakers, consisting of 111 men, 114 women, 50 boys and 51 girls, pronounced 77 digit sequences in a quiet acoustic enclosure.

This dataset is available at [58] but is licensed and not freely available for download.

9.1.2 Free Spoken Digit Dataset (FFSD)

The purpose of this dataset is to be a free spoken digit data set [28], similar as what MNIST [59] is for the written digits. Where MNIST is a low complexity dataset used to experiment with different machine learning algorithms. Currently there are contributions from 5 speakers, whereby each speaker repeated a digit 50 times. The samples were recorded at 8kHz with minimal background noise.

This dataset is freely available at [28].

9.1.3 Speech Commands

This dataset contains spoken words, designed to help train and evaluate keyword spotting systems [60]. Besides the digits, other (short) words are also present. The quality of the recordings is not controlled, mimicing real-world conditions. The dataset contains 64,727 utterances from 1,881 speakers at 16kHz.

This dataset is freely available at [60]. Inspection of this dataset however revealed that a significant amount of samples are silent or wrongly labeled. These errors in the dataset can destabilize the learning [61]. An attempt to clean this dataset is made by [62], where the bad samples are separated from the proper ones. The presented results in this report are based upon this cleaned dataset.

9.1.4 Chosen

The chosen dataset is “Speech Commands” since it contains more realistic samples that better match the recordings from the microphone of the prototype setup.

9.2 Design Space Exploration

The prototype, as described in chapter 8, is implemented with parameters that were deemed appropriate based upon literature and intermediate results. However since the parameters of different module are correlated, it is unclear if the chosen parameters are still optimal and what the contribution of each parameter is to the entire system.

In this section a Design Space Exploration (DSE) is performed to provide insight in the contribution of each parameter and which trade-off between area, power and classification accuracy can be made.

9.2.1 Parameters

9.2.1.1 PDM oversampling ratio

The microphone can be utilized in the range of 0.4 until 3.3 MHz [6]. Assuming a desired PCM sampling frequency of 8 kHz, the oversampling ratio can be chosen. For example an oversampling ratio of 64 will result in a PDM frequency of 512 kHz.

It is expected that an higher oversampling ratio increases the SNR, because of noise shaping, but utilizes more energy since the device has to run at higher clock speeds.

9.2.1.2 Polyphase filter order

The polyphase filter can have any amount of functional taps but taking into account the structure, a filter order that is a multiple of the oversampling ratio is recommended.

A higher filter order is assumed to increase the quality and will require additional computations.

9.2.1.3 Polyphase fractional bits

In fixed-point the smallest representable value is determined by the amount of fractional bits used. Since the PDM to PCM conversion is in fixed-point, the accuracy of the coefficients is determined by the amount of fractional bits.

Increasing the amount of fractional bits can increase the quality but it will also require larger adders and registers, consuming area and power.

9.2.1.4 Frame size

To obtain an STFT, multiple subsequent samples are grouped together into a single frame. The quality of the FFT will depend on the rate of change of frequencies. A smaller frame is expected to provide a higher resolution but noisy spectrogram.

Following the previous work [1], a frame size of 160 is used as base. From literature it is know that a multiple of 2 is in general more efficient in hardware. Therefore, frame sizes of 128 and 256 are potential candidates to increase efficiency.

9.2.1.5 Pow simplification

An intermediate step for power estimation involves the sum of squares. A multiplication operation is expensive and moreover is used many times for the sum of square. To simplify the computations it is considered to instead take the sum of absolutes.

9.2.1.6 Log simplification

The final step for power estimation involves a logarithm operation. To simplify the computations it is considered to not use this operation.

9.2.2 Quality Metrics

In order to estimate the influence of the parameters, several quality metrics are utilized. Note however that the metrics are rough estimates that only take into account a subset of the complete design. This is to reduce the complexity, because some parts cannot be reliably modeled, are expected to be modified later or are confidential. Thus only the change of the metric should be considered and not their absolute values. Thereby providing an indication of the design space for future reference, i.e. when the Tempotron training, SOM and SNN are combined into one model as proposed before.

The metrics under consideration are elaborated below

9.2.2.1 Spectrogram

The metric is calculated as a point wise comparison between a reference spectrogram obtained from running the simulation with highest classification accuracy settings and an spectrogram obtained with reduced accuracy settings. It provides an indication of the error introduced into the spectrogram.

Since large errors are expected to be more significant, a sum of squares is used to quantify the difference.

$$\text{difference} = \sum_{i=0}^{N} x_i^2 \tag{9.1}$$

Where N is the total amount of data points and x_i the i-th data point.

A smaller spectrogram value indicates a higher spectrogram quality, which is deemed to be positive.

9.2.2.2 Tempotron

The training algorithm to obtain weights, Tempotron, inherently computes classification errors for its weight updates. The Tempotron classification accuracy provides an indication of the weight update progress and the maximum attainable accuracy.

A higher value is deemed, in general, positive. Although care should be taken since Tempotron is likely to over fit when its classification accuracy is high.

9.2.2.3 SNN

The SNN is simulated using both the train set and the test set. Classification accuracies obtained from this simulation provide an indication of separability of the data set by a SNN.

Higher classification accuracy values are deemed to be positive, especially for the test accuracy.

9.2.2.4 Transistors

Since the target platform is likely area constrained a measure is made to estimate the influence of the parameters on the area requirement.

The basic unit of area is defined as the amount of transistors required, which depends on the amount of bits involved and the computations. Note that the following calculations only take into account a subset of the logic, which is expected to be significantly influenced by the parameters. Their purpose is to compare configurations between each other, therefore the reported values are normalized in the tables.

In [63] several adders are compared to each other. When looking at the Carry-Look Ahead (CLA) adder it is found that for a 4-bit adder it requires 188 transistors. Assuming a linear relation, as rough approximation, this would mean 47 transistors per bit. Making the transistor count for an addition:

$$\mathbf{Add} = 47 * \#\text{bits} \tag{9.2}$$

Taking into account an array multiplier, it is seen that approximately 3150 gates are required for this 16-bit multiplier [64]. Taking an average of 5 transistors per gate, this would translate to 15750 transistors. Assuming a linear relation, as rough approximation, this would mean 984 transistors per bit. Making the transistor count for a multiplication:

$$\mathbf{Mult} = 984 * \#\text{bits} \tag{9.3}$$

From [65] it is found that the logarithmic computation can be implemented by 4 adders and a multiplier. Making the transistor count for a logarithm:

$$\textbf{Log} = 4 * \text{Add} * \text{Mult} \tag{9.4}$$

The area estimation from the pdm-pcm and filterbank preprocessing modules is

$$\textbf{pdm-pcm} = \left\lceil \frac{\text{taps}}{\text{oversampling}} \right\rceil * \text{Add} \tag{9.5}$$

$$\textbf{filterbank DFT} = 2N * \text{Add} + \frac{N}{4} * \text{Mult} \tag{9.6}$$

$$\textbf{filterbank mel-scaled triangles} = \text{Add} + \text{Mult} + \text{Log} \tag{9.7}$$

Where N stands for the frame size. The frequency (freq) is fixed at $8kHz$.
For the *DFT* it is assumed that the butterfly can be reused for all rounds and for the *mel-scaled triangles* a single adder and multiplier are reused for all computations. The sum of these estimates will illustrate the global influence of the parameters under consideration.

$$\textbf{transistor estimation} = \text{pdm-pcm} + \text{filterbank DFT} + \text{filterbank mel-scaled triangles} \tag{9.8}$$

The unit is transistor count.

9.2.2.5 Power

Since the target platform is energy constrained a measure is made to estimate the influence of the parameters on the power consumption.

The basic unit of power is defined as the cost of using a single transistor, which depends on the amount of bits involved and the computations. Note that the following calculations only take into account a subset of the logic, which is expected to be significantly influenced by the parameters. Their purpose is to compare configurations between each other, therefore the reported values are normalized in the tables and figures.

According to [63] a 1-bit CLA adder consumes about $28\mu W$ at 4 Volts. Assuming a linear relation, as rough approximation, this would mean $28\mu W$ per bit. Making the power consumption for an addition:

$$\textbf{Add} = 28\mu W * \#\text{bits} \tag{9.9}$$

Assuming that the multiplier requires 984 transistors per bit while the adder requires 47 transistors per bit. A rough estimation indicates that the multiplier requires $586\mu W$ per bit.

$$\textbf{Mult} = 586\mu W * \#\text{bits} \tag{9.10}$$

From [65] it is found that the logarithmic computation can be implemented by 4 adders and a multiplier. Making the power consumption for a logarithm:

$$\textbf{Log} = 4 * \text{Add} + \text{Mult} \tag{9.11}$$

The power estimation from the pdm-pcm and filterbank preprocessing module is

$$\textbf{pdm-pcm} = \left\lceil \frac{\text{taps}}{\text{oversampling}} \right\rceil * \text{oversampling} * \text{Add} * \text{freq} \tag{9.12}$$

$$\textbf{filterbank DFT} = (2N * \text{Add} + \frac{N}{4} * \text{Mult}) * \lceil \log_2(N) \rceil * \frac{\text{freq}}{\text{N}} \tag{9.13}$$

$$\textbf{filterbank mel-scaled triangles} = (2N * (\text{Add} + \text{Mult}) + \text{Log}) * \frac{\text{freq}}{\text{N}} \tag{9.14}$$

Where N stands for the frame size. The frequency (freq) is fixed at $8kHz$.
Since each filter overlaps half of the frequency range of its closest neighbours all the frequencies in the frame are used twice, The sum of these estimates will illustrate the global influence of the parameters under consideration.

$$\textbf{power estimation} = \text{pdm-pcm} + \text{filterbank DFT} + \text{filterbank mel-scaled triangles} \tag{9.15}$$

The unit is Watt.

9.2.3 Results

The consequence of changing a parameter can also depend on other parameters. To keep the DSE manageable, the parameters that are expected to significantly influence each other: oversampling ratio, filter order and amount of bits are considered together. While those that are expected to be mostly independent: frame size, pow approximation and log approximation are considered separately.

The baseline for normalization is, except for the frequency domain filtering, equal to the audio preprocessing pipeline as in the work of Prozée [1]. Note that this means that the PDM to PCM module is also skipped in the normalization base and the frame size was set to 160. and The results in this chapter should thus be interpreted as an approximation of the change in transistor count and power relative to the implementation from [1].

9.2.3.1 Fixed hyper parameters

Based upon the work from Prozée [1] and preliminary experiments (not shown here) the SOM was fixed on a dimension of 22×22 together with 100 training epochs since higher values did not result in improved results. The DFT frame size is fixed at 160 and no simplification for the log and pow are made.

From Table 9.1 it is clear that additional training epochs result in higher Tempotron training accuracies. The SNN accuracies are approaching a limit after 200 Tempotron epochs.
The gap between Tempotron train classification accuracy and SNN train accuracy is assumed to originate from a too simple Tempotron model. To mitigate this gap, the Tempotron learning should use the same model as the SNN.
The gap between SNN train and test classification accuracy is assumed to originate from Tempotron over fitting on the train set. To mitigate this gap techniques to reduce over fitting can be used, e.g. more data, dropout or early stopping [66]

Applying additional Tempotron training epochs after 200 does not significantly

Tempotron epochs	**Tempotron (%)**	**SNN train (%)**	**SNN test (%)**
50	67.52 ± 1.6	60.93 ± 2.0	31.00 ± 2.9
100	78.50 ± 1.1	68.76 ± 1.2	33.32 ± 1.3
200	87.57 ± 1.2	73.61 ± 1.0	34.91 ± 2.1
500	96.43 ± 0.4	74.31 ± 2.2	34.73 ± 2.7
1000	97.97 ± 1.1	74.09 ± 1.1	35.41 ± 2.7

Table 9.1: Influence of Tempotron training epochs on classification accuracy, standard deviation included.

benefit the SNN, rather it increases the Tempotron over fitting. To mitigate further over fitting: early stopping is applied by limiting Tempotron learning to 200 epochs.

9.2.3.2 Polyphase parameters

The data for polyphase oversampling (s), polyphase order (r) and amount of polyphase bits (b) is presented in Table 9.2
A scatter plot for the trade-off between classification accuracy and cost, depending on polyphase oversampling, order and bits is provided in Figure 9.1. In this plot the Tempotron train accuracy is used as a measure of separability of the data set.

9.2.3.3 Frame size

Several DFT frame configurations and their influence are reported in Table 9.3. During these experiments the not mentioned parameters are equal to the parameters of the normalization base.

9.2.3.4 Pow simplification

The impact of taking the sum of absolutes instead of the sum of squares for frequency power estimation after the DFT stage appears to be significant according to Table 9.4

During these experiments the not mentioned parameters are equal to the parameters of the normalization base.

Index	Configuration	Spectrogram	Tempotron (%)	SNN Train (%)	SNN Test (%)	Transistors	Power
1	b: 12, s: 32, r: 32	0.116 ± 0.055	83.28 ± 1.0	70.68 ± 0.9	30.23 ± 0.7	1.00047	1.19
2	b: 12, s: 32, r: 64	0.115 ± 0.045	82.92 ± 0.5	70.23 ± 1.7	36.74 ± 0.6	1.00095	1.38
3	b: 12, s: 32, r: 128	0.185 ± 0.047	65.76 ± 0.2	59.92 ± 1.4	22.80 ± 2.0	1.00190	1.75
4	b: 12, s: 64, r: 32	0.115 ± 0.056	84.22 ± 0.8	69.39 ± 1.6	30.91 ± 2.2	1.00047	1.38
5	b: 12, s: 64, r: 64	0.083 ± 0.052	86.04 ± 0.4	72.40 ± 1.1	32.12 ± 1.5	1.00047	1.38
6	b: 12, s: 64, r: 128	0.097 ± 0.047	87.14 ± 0.4	71.95 ± 0.9	33.26 ± 1.2	1.00095	1.75
7	b: 12, s: 128, r: 32	0.116 ± 0.055	82.80 ± 1.5	69.59 ± 1.8	30.98 ± 1.9	1.00047	1.75
8	b: 12, s: 128, r: 64	0.084 ± 0.053	87.61 ± 1.2	69.79 ± 2.0	34.55 ± 2.5	1.00047	1.75
9	b: 12, s: 128, r: 128	0.066 ± 0.050	87.75 ± 1.0	72.65 ± 0.5	37.12 ± 3.0	1.00047	1.75
10	b: 14, s: 32, r: 32	0.117 ± 0.055	82.41 ± 1.0	71.74 ± 1.6	35.76 ± 1.5	1.00055	1.22
11	b: 14, s: 32, r: 64	0.114 ± 0.045	83.54 ± 0.6	71.78 ± 0.5	35.38 ± 0.7	1.00111	1.44
12	b: 14, s: 32, r: 128	0.182 ± 0.041	64.53 ± 0.8	59.53 ± 0.6	23.49 ± 1.2	1.00221	1.88
13	b: 14, s: 64, r: 32	0.116 ± 0.056	84.70 ± 0.8	69.15 ± 2.0	30.45 ± 1.3	1.00055	1.44
14	b: 14, s: 64, r: 64	0.084 ± 0.052	84.47 ± 0.7	71.44 ± 1.5	33.26 ± 0.6	1.00055	1.44
15	b: 14, s: 64, r: 128	0.093 ± 0.047	85.40 ± 1.0	70.68 ± 1.1	32.65 ± 0.6	1.00111	1.88
16	b: 14, s: 128, r: 32	0.116 ± 0.055	86.19 ± 1.1	69.41 ± 1.4	30.76 ± 0.9	1.00055	1.88
17	b: 14, s: 128, r: 64	0.085 ± 0.053	86.55 ± 1.0	73.41 ± 0.5	35.91 ± 1.0	1.00055	1.88
18	b: 14, s: 128, r: 128	0.060 ± 0.050	88.50 ± 0.8	74.90 ± 1.1	33.26 ± 1.9	1.00055	1.88

Table 9.2: Parameter influence of amount of polyphase bits (b), polyphase oversampling (s) and polyphase order (r). The transistor count and power is normalized.

Frame size	Tempotron (%)	SNN Train (%)	SNN Test (%)	Transistors	Power
128	80.93 ± 0.8	67.99 ± 0.9	29.70 ± 0.6	0.81688	0.93
160	87.25 ± 0.3	73.30 ± 1.0	36.74 ± 2.8	1.00000	1.00
256	74.41 ± 0.7	66.48 ± 0.3	34.55 ± 0.4	1.54937	1.00

Table 9.3: Influence of DFT frame size

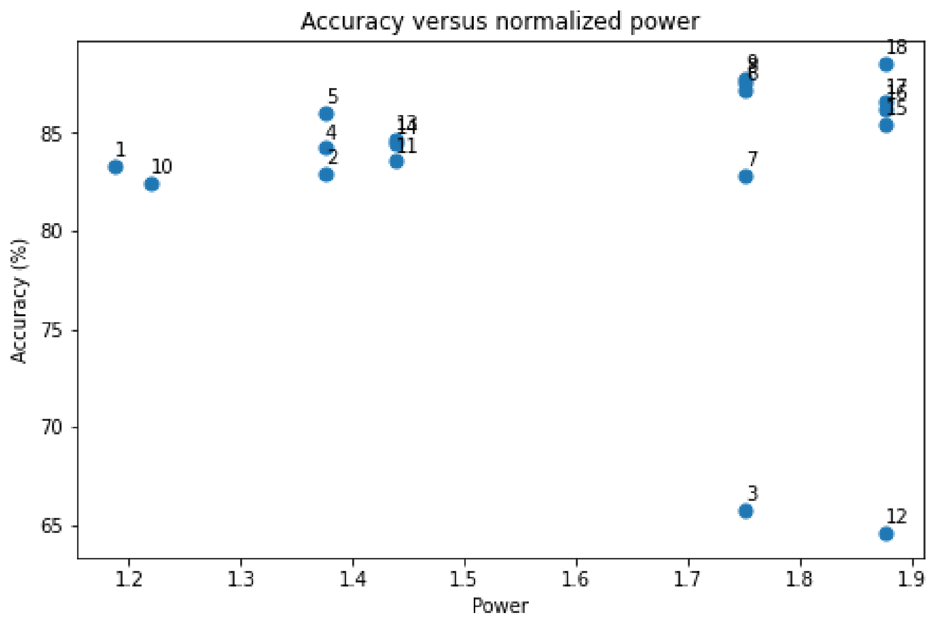
Configuration	Spectrogram	Tempotron (%)	SNN Train (%)	SNN Test (%)	Transistors	Power
square	0.000 ± 0.000	87.57 ± 1.2	73.61 ± 1.0	34.91 ± 2.1	1.00000	1.00
absolute	0.043 ± 0.015	86.15 ± 0.3	74.41 ± 1.3	36.73 ± 1.9	0.98344	0.59

Table 9.4: Influence of power simplification by taking the sum of absolutes instead of sum of squares

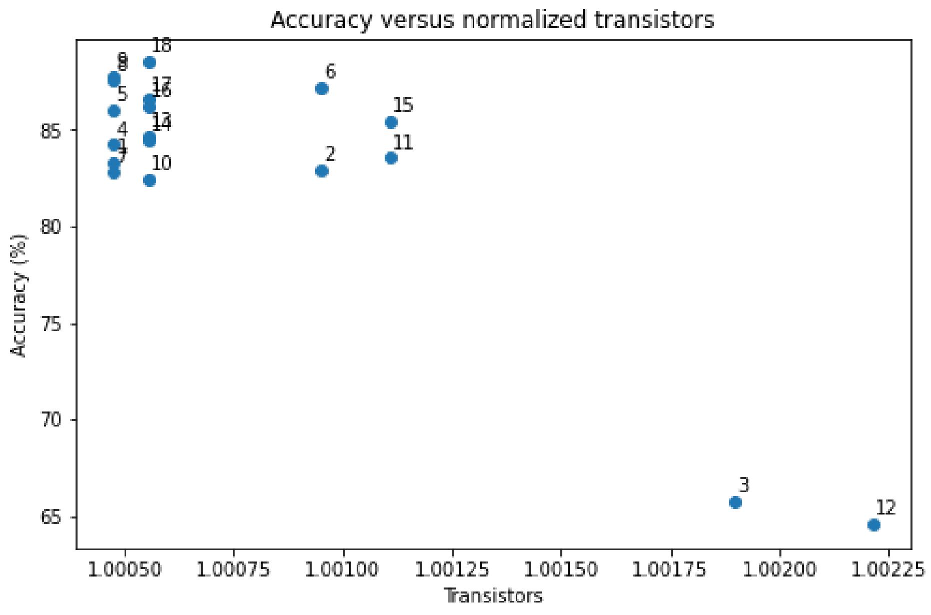
9.2.3.5 Log simplification

The impact of skipping the logarithm after the DFT stage is shown visually in Figure 9.3 and the consequences are shown in Table 9.5

During these experiments the not mentioned parameters are equal to the parameters of the normalization base.

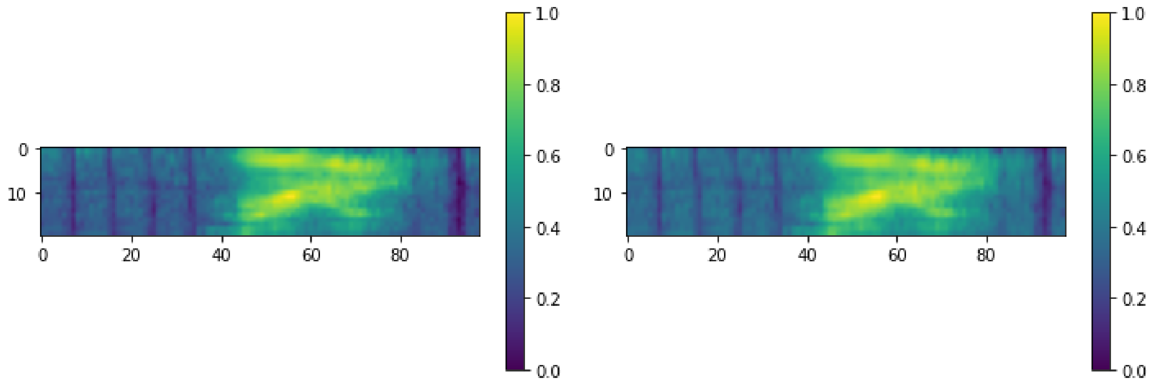


(a)



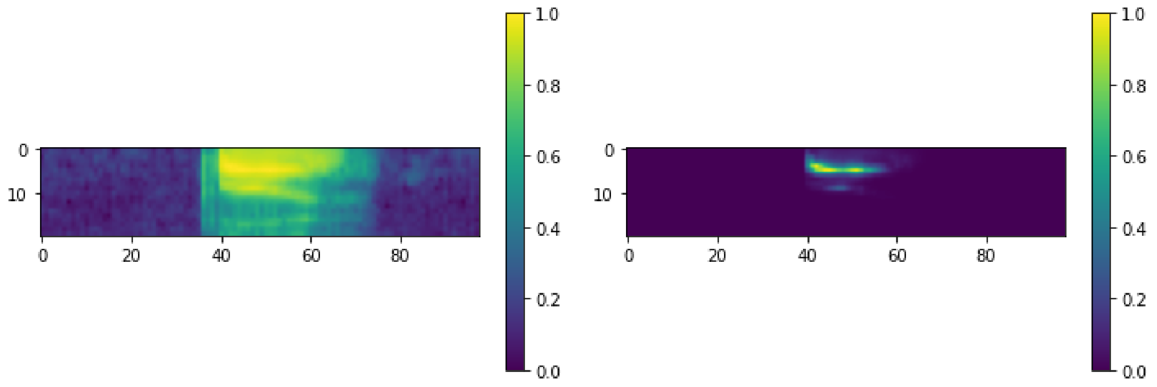
(b)

Figure 9.1: Scatter plot for Tempotron train classification accuracy versus normalized power and transistors in (a) and (b) respectively



(a) (b)

Figure 9.2: Spectrograms with the square and with absolute as (a) and (b) respectively, note that the area and energy estimation disregards the pdm-pcm module here



(a) (b)

Figure 9.3: Spectrograms with the logarithm (a) and without (b).

Configuration	**Spectrogram**	**Tempotron (%)**	**SNN**		**Transistors**	**Power**
			Train (%)	**Test (%)**		
log	0.000 ± 0.000	87.57 ± 1.2	73.61 ± 1.0	34.91 ± 2.1	1.00000	1.00
pass	0.440 ± 0.069	57.98 ± 1.6	51.15 ± 1.8	27.14 ± 1.9	0.93296	0.99

Table 9.5: Influence of not computing the logarithm

9.3 Evaluation

9.3.1 DSE

Based on the polyphase configurations, shown in Table 9.2, it is found that configuration 5 is be desired since it has the highest classification accuracy among the configurations that are at the lower power end in Figure 9.1. From the pow simplification in Table 9.4 it is found that taking the absolute appears to still provide sufficient contrast in the spectrogram. The frame size, Table 9.3, and logarithm, Table 9.5, should however not be modified. The performance of the proposed configuration is presented in Table 9.6.

Spectrogram	**Tempotron (%)**	**SNN**		**Transistors**	**Power**
		Train (%)	**Test (%)**		
0.110 ± 0.045	87.68 ± 1.3	74.15 ± 2.1	37.18 ± 0.8	0.98391	0.97

Table 9.6: Final configuration

9.3.2 Additional effects on SNN

The audio preprocessing, described in this thesis, generates samples in the order of X while the SNN, due to hardware constraints, needs to consume samples in the order of Y. To bridge this gap we propose to collect spectrogram values or spikes from multiple time steps in a single frame. The frame time-step can freely be scaled, in this case a compression of Z is sufficient. This frame will then be send to the SNN in a burst at a fixed time interval.

Using a frame as base unit, an indication of the SNN active time and input spike count can be provided.

9.3.2.1 SNN active time

The fixed time interval for bursts and the SNN active time are directly related to the frame size with Equation 9.16 and Equation 9.18 respectively.

$$\text{frame interval} = \text{frame size} * \text{spectrogram time step} \tag{9.16}$$

$$\text{frame duration} = \text{frame interval} * \text{frame compression} \tag{9.17}$$

where compression is a value in the order of $\frac{X}{Y}$.

$$\text{SNN active time fraction} = \frac{\text{frame duration}}{\text{frame interval}} = \frac{1}{\text{frame compression}} \tag{9.18}$$

In this case, with a spectrogram time step of X, frame size of 160 and compression of $\frac{X}{Y}$, the SNN active time fraction is $\frac{Y}{X}$. Depending on the optimal SNN input frequency, the compression can be adapted. Note that no exact values are given due to confidentiality.

9.3.2.2 Input spike count

When using the SOM as an encoder the total input spikes are constant since the SOM, per definition, provides a single output spikes for each input vector.

$$\text{input spike count} = \text{frame size} \tag{9.19}$$

For the threshold encoding the maximum amount of spikes is bounded by

$$\text{input spike count} = \text{frame size} * \#\ \text{thresholds} \tag{9.20}$$

while the average amount of spikes depend on the frequency content of the frame.

In this case, with the SOM, there will be 160 input spikes per frame.

9.3.3 Dataset expansion

Tests with own voice recordings, using the prototype, have reported (not shown here) significantly lower accuracies (20%) compared to the test set as reported in this chapter (35%). After expanding the train set with several own voice recordings the accuracies were found to be similar to the test set.
It is assumed that there is a discrepancy between the recordings from the physical prototype compared to the data set. It is however unclear if the discrepancy arises from the simple sigma-delta model, physical microphone characterization, fundamental difference between speakers or environmental influences.

Due time constraints it is deemed unfeasible to create a new sufficiently large data set using the prototype. A compromise is made by expanding the data set by own recordings. Adding own voice recordings to the data set (unique ones for train and test sets) has shown to significantly mitigate the discrepancy. In this way the original data set is meant to generalize between speakers while the own recordings should inject the microphone and environment characteristic information.

9.4 Comparison to literature

To compare the audio preprocessing pipeline to state-of-the-art, accurate power and classification accuracies are required.

SNN test classification accuracies reported in this thesis are significantly lower compared to the 97% from the work of Prozée [1]. Since the Tempotron train classification accuracies are capable of approaching those mentioned in [1] it is assumed that this gap is caused by modification of parameters to suit hardware. Since this gap will likely be mitigated when Tempotron utilizes the same model as the SNN the 97% classification accuracy from [1] is assumed to better reflect the capability of the system, since the audio preprocessing has near identical functionality. In the work of Prozée [1] it is seen that the classification accuracies obtained after proper training of the SNN are competitive with Convolutional Neural Networks, Long-Short Term Memory and Recurrent Neural Networks.

Rough area and energy estimates for comparison are given for a subset of the design in subsubsection 9.2.2.4 and subsubsection 9.2.2.5 in the form of transistor count and power. Due to the scale and the complexity of the design it is however not yet feasible in this stage of the project to obtain accurate energy and area estimates for the entire design. Problems include the estimation of cost for the DMA, SNN, UART communication and CPU.

9.5 Discussion

From Table 9.2, visually shown in Figure 9.1, it is clear that the parameters under consideration in this chapter do not have a significant influence on the amount of transistors, representing area. Instead, a large influence of the parameters on the power consumption is observed.

A clear discrepancy can be seen when comparing the accuracies obtained in section 9.2, around 35%, to those reported in the work from Prozée [1], at 97%. The main contribution of this discrepancy is assumed to originate from the fact that in the work from Prozée the SNN simulation is using an unrealistic small time step, in the order of 100 times too small, and inaccurate hardware values for time constants. While in section 9.2 only feasible hardware parameters are considered for the SNN. This is supported by the fact that when using an preprocessing pipeline with identical input and output relation as in the work of Prozée, shown in Table 9.1, the SNN test classification accuracy is already capped at around 35%. Additional tests (not shown here) confirmed that with invalid hardware parameters accuracies in the range of 90% can be obtained with the proposed design.

In this section a DSE is shown to estimate the suitability of certain configurations. Keep in mind however that depending on the implementation the power and transistor count estimations can still vary. The current estimations are only intended to provide a rough indication of the influence of a parameter and should only be used to compare between configurations. After specific hardware and implementations are chosen it is recommended to adjust and extend the energy and area estimation formulas in order to align the estimates closer to the hardware.
Nevertheless, a configuration is chosen that is expected to provide the best trade-off between distinguishability and power cost.

By expanding the dataset with own recordings, for a live demo, it can be argued that the dataset is polluted by making the network over fit on a specific speaker, in this case the author. Though for practical applications, to obtain significant performance, it is likely that a small training phase from a user's voice will be required before use anyhow.

This chapter contributes with an exploration of parameter influence on classification accuracy in an audio preprocessing pipeline.

Conclusion 10

In the work of Prozée [1] an audio preprocessing pipeline is described that can support a SNN to obtain a competitive classification accuracy of 97%. The described preprocessing algorithm has however disregarded hardware constraints. To obtain an audio preprocessing algorithm that can be used with hardware constrained devices, the algorithm is analyzed and optimized in this thesis.
To ensure a structured approach a Controller framework has first been developed that can support the automatic execution of a pipeline of modules written in different languages. Using this controller, each module was considered and optimized individually.

The algorithm pipeline consists of five modules that can be freely interchanged with other implementations. They are: PDM to PCM, filtering, encoding, training and SNN. The PDM to PCM module is used to reformat the microphone PDM data to PCM for algorithm simplification. It was seen that the polyphase FIR filter is the most computational effective from several options.
The filtering module estimates the energy in Mel-scaled frequency bands. Due to the amount of frequency bands an improved implementation, compared to [1], was obtained by utilizing a DFT.
The encoding module is aimed at translating multi-bit values to single bit spikes. There were several available methods but, as described in the work of Prozée, only when utilizing the computationally complex SOM significant accuracies, above 90%, can be obtained.
The training module is used to obtain weights for the SNN. Though there are training methods that support multi-layered networks they are difficult to utilize because of their hyper-parameters. Because of the time constraints of this thesis the simple single layer Tempotron learning method was chosen.
The SNN module models the (future) functionality of the SNN in hardware for which the audio preprocessing algorithm should be compatible.

After individual modules were optimized a DSE was used on the entire design to determine suitable values for several module interdependent parameters. A clear discrepancy between the work from Prozée [1] was observed when the parameters were modified to reflect hardware feasible values. Moreover, from preliminary tests with the prototype it was observed that it will likely be required to include recordings of the user in the train set to effectively utilize the prototype.

By implementing a real time prototype it is found that additional components, mainly for communication, are required, e.g. DMA. Moreover, it is seen that a combination of both dedicated and general hardware can cover the varying requirements of the pipeline, e.g. clock frequencies. Even though a functional prototype has been build, it did not progress far enough to asses the performance of the entire design with it just yet.

A significant issue that remains unresolved is that the SOM is too computationally expensive to be used as a standalone component, e.g. encoder. Moreover, the Tempotron learning is performed based on a simple model which is not related close enough to the SNN simulator model. From chapter 9 a clear reduction in train classification accuracy can be seen from Tempotron to SNN. Work is currently being performed by a colleague on integrating the Tempotron learning, SOM and SNN in a single simulator model. Making the Tempotron and SOM utilize the same model as the SNN is likely going to significantly improve the overall classification accuracy compared to the accuracies reported in this thesis. Moreover, it would enable the encoder to use a computationally simpler algorithm, e.g. p-TE, reducing energy and area consumption.

Future Work 11

In chapter 9 it is seen that the classification accuracies are lacking compared to those from the work of Prozée [1]. Even though the discrepancy originates from modifying the SNN model to hardware feasible parameters, further fine tuning of fixed parameters, e.g. SOM size or training epochs, and improving the Tempotron model can possibly improve the performance.

Since the SOM is too computationally expensive for embedded devices it is recommended in future work to circumvent the need of the non-spiking SOM. Suggestions include using a multi-layered learning method, e.g. STDP, or creating a spiking-SOM implementation. The spiking-SOM implementation can then be moved into the SNN, but note however that this will require another encoder

Further optimization of the audio preprocessing pipeline might be obtained by converting all computations to Bit-Stream Signal Processing (BSSP). With BSSP computations are performed using single-bits but the computations are performed at a higher frequency compared to similar computations for multi-bit values. Using such a scheme it would circumvent the need to transform the (microphone) input from single- to multi-bit. Moreover, filtering operations can possibly be computationally simplified since no multipliers will be required. For now it remains unclear if BSSP is applicable for audio preprocessing.

After the classification accuracy is improved, additional experiments with the prototype should be performed to determine the effective accuracy in a realistic setup. Lastly, the PDM to PCM module should be implemented on an ASIC to increase efficiency.

Bibliography

[1] R. Prozée, “Sound-recognition using spiking neural networks,” Master’s thesis, Delft University of Technology, 2021.

[2] K. Academy, “Overview of neuron structure and function.” https://www.khanacademy.org/science/biology/human-biology/neuron-nervous-system/a/overview-of-neuron-structure-and-function, 2016. [Online; accessed 3-June-2021].

[3] J. Wu, Y. Chua, M. Zhang, H. Li, and K. Tan, “A spiking neural network framework for robust sound classification,” *Frontiers in Neuroscience*, vol. 12, 11 2018.

[4] J. S. P. Giraldo, S. Lauwereins, K. Badami, and M. Verhelst, “Vocell: A 65-nm speech-triggered wake-up soc for 10- μ w keyword spotting and speaker verification,” *IEEE Journal of Solid-State Circuits*, vol. 55, no. 4, pp. 868–878, 2020.

[5] N. Hegde and A. DEVICES, “Seamlessly interfacing mems microphones with blackfin processors,” 08 2010.

[6] * Ommited due to confidentiality *.

[7] B. Baker, “How delta-sigma adcs work, part 1.” https://www.ti.com/lit/an/slyt423a/slyt423a.pdf. Accessed: 2021-5-12.

[8] J. Smith, “Multirate noble identities.” https://ccrma.stanford.edu/~jos/sasp/Multirate_Noble_Identities.html, 7 2020. Accessed: 2021-6-22.

[9] R. Lyons, “A beginner’s guide to cascaded integrator-comb (cic) filters.” https://www.dsprelated.com/showarticle/1337.php, 26 2020. Accessed: 2021-6-16.

[10] V. Jayaprakasan and M. Madheswara, “Cascading sharpened cic and polyphase fir filter for decimation filter,” *2nd International Conference on Advances in Electrical and Electornics Engineering*, 03 2013.

[11] MathWorks, “Fir rate conversion hdl optimized.” https://fr.mathworks.com/help/dsp/ref/firrateconversionhdloptimized.html, 2021. Accessed: 2021-5-12.

[12] NI, “Labview 2011 digital filter design toolkit help.” https://zone.ni.com/reference/en-XX/help/371325F-01/lvdfdtconcepts/iir_sos_specs/, 2021. Accessed: 2021-6-17.

[13] O’REILLY, “Structure of fir filters.” https://www.oreilly.com/library/view/digital-filters-design/9781905209453/ch007-sec002.html, 2021. Accessed: 2021-6-17.

[14] R. Baraniuk, *Signals and Systems*. Orange Grove Texts Plus, 2009.

[15] N. K. Kasabov, *Time-Space, Spiking Neural Networks and Brain-Inspired Artificial Intelligence*. Springer Berlin Heidelberg, 2019.

[16] Z. Pan, J. Wu, M. Zhang, and H. Li, "Neural population coding for effective temporal classification," 09 2019.

[17] L. de Gelder, "Population step forward encoding algorithm," Master's thesis, Delft University of Technology, 2021.

[18] L. Abbott and S. Nelson, "Synaptic plasticity: Taming the beast," *Nature neuroscience*, vol. 3 Suppl, pp. 1178–83, 12 2000.

[19] R. Gütig and H. Sompolinsky, "The tempotron: a neuron that learns spike-timing based decisions," *Reviews in the neurosciences*, vol. 16, pp. S27–S27, 01 2005.

[20] Xilinx, "Axi reference guide." https://www.xilinx.com/support/documentation/ip_documentation/ug761_axi_reference_guide.pdf, 2021. [Online; accessed 19-May-2021].

[21] T. Verbeure, "Pdm microphones and sigma-delta a/d conversion." https://tomverbeure.github.io/2020/10/04/PDM-Microphones-and-Sigma-Delta-Conversion.html, 10 2020. Accessed: 2021-5-12.

[22] W. S. McCulloch and W. Pitts, "A logical calculus of the ideas immanent in nervous activity," *Bulletin of Mathematical Biology*, vol. 52, no. 1, pp. 99–115, 1990.

[23] P. J. Werbos, *Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences*. PhD thesis, Harvard University, 1974.

[24] M. Nielsen, "Neural networks and deep learning." http://neuralnetworksanddeeplearning.com/, 2019. [Online; accessed 3-June-2021].

[25] Y. Wang, K. Shahbazi, H. Zhang, K. OH, J.-J. Lee, and S.-B. Ko, "Efficient spiking neural network training and inference with reduced precision memory and computing," *IET Computers and Digital Techniques*, vol. 13, 06 2019.

[26] E. Ledinauskas, J. Ruseckas, A. Jursenas, and G. Burachas, "Training deep spiking neural networks," *CoRR*, vol. abs/2006.04436, 2020.

[27] D. Hebb, "The organization of behavior: A neuropsychological theory," 1949.

[28] Z. Jackson, C. Souza, J. Flaks, Y. Pan, H. Nicolas, and A. Thite, "Jakobovski/free-spoken-digit-dataset: v1.0.8," 08 2018.

[29] Z. Pan, Y. Chua, J. Wu, M. Zhang, H. Li, and E. Ambikairajah, "An efficient and perceptually motivated auditory neural encoding and decoding algorithm for spiking neural networks," *Frontiers in Neuroscience*, vol. 13, 01 2020.

[30] A. das, M. R. Jena, and K. K. Barik, “Mel-frequency cepstral coefficient (mfcc) - a novel method for speaker recognition,” *Digital Technologies*, vol. 1, no. 1, pp. 1–3, 2014.

[31] T. Kohonen, “The self-organizing map,” *Proceedings of the IEEE*, vol. 78, no. 9, pp. 1464–1480, 1990.

[32] D. Roussinov and H.-c. Chen, “A scalable self-organizing map algorithm for textual classification: A neural network approach to thesaurus generation,” *CC-AI*, vol. 15, 10 1999.

[33] ecma TC39, “JSON.” https://www.json.org, 12 1999.

[34] “More about pdm.” https://www.ap.com/technical-library/more-about-pdm/. Accessed: 2021-5-12.

[35] T. Memon, P. Beckett, and A. Sadik, “Sigma-delta modulation based digital filter design techniques in fpga,” *ISRN Electronics*, vol. 2012, 11 2012.

[36] U. Beis, “An introduction to delta sigma converters.” https://www.beis.de/Elektronik/DeltaSigma/DeltaSigma.html, 2 2020. Accessed: 2021-5-12.

[37] M. Perkins, “Delta sigma converters: Modulation.” https://www.cardinalpeak.com/blog/delta-sigma-converters-modulation, 2 2010. Accessed: 2021-5-12.

[38] Wikipedia contributors, “Delta-sigma modulation.” https://en.wikipedia.org/wiki/Delta-sigma_modulation, 2021. [Online; accessed 12-May-2021].

[39] A. Loloee, “Understanding delta-sigma modulators.” https://www.electronicdesign.com/technologies/analog/article/21798185/understanding-deltasigma-modulators, 7 2013. Accessed: 2021-7-30.

[40] F. Harris, *Multirate Signal Processing for Communication Systems*. Prentice Hall, 03 2004.

[41] N. Fliege, *Multirate Digital Signal Processing: Multirate Systems - Filter Banks - Wavelets*. Wiley, 01 2000.

[42] K. Kastner, “Polyphase filters and filterbanks.” https://www.dsprelated.com/showarticle/191.php, 3 2013. Accessed: 2021-5-12.

[43] M. Fowler, “Eece 521: Digital signal processing ii.” Accessed: 2021-5-12.

[44] R. Lyons, “Understanding cascaded integrator-comb filters.” https://www.embedded.com/understanding-cascaded-integrator-comb-filters/, 3 2005. Accessed: 2021-5-12.

[45] C. Candan, “Optimal sharpening of cic filters and an efficientimplementation through saramäki-ritoniemi decimation filter structure (extended version),” *Department of Electrical Engineering, METU*.

[46] D. Pichandi and V. Jayaprakasan, “Design and implementation of efficient cic filter structure for decimation,” *International Journal of computer applications*, vol. 65, pp. 975–8887, 03 2013.

[47] J. Cooley, P. Lewis, and P. Welch, “The finite fourier transform,” *IEEE Transactions on Audio and Electroacoustics*, vol. 17, no. 2, pp. 77–85, 1969.

[48] M. Borgerding, “Keep it simple, stupid fft library.” https://github.com/mborgerding/kissfft, 2012.

[49] Jacobsen, Eric, “Understanding and implementing the sliding dft.” https://www.dsprelated.com/showarticle/776.php, 4 2015. Accessed: 2021-6-17.

[50] Elder, Robert, “Overlap add, overlap save visual explanation.” https://blog.robertelder.org/overlap-add-overlap-save/, 2 2018. Accessed: 2021-6-17.

[51] A. Kugele, T. Pfeil, M. Pfeiffer, and E. Chicca, “Efficient processing of spatio-temporal data streams with spiking neural networks,” *Frontiers in Neuroscience*, vol. 14, p. 439, 05 2020.

[52] T. Rumbell, S. Denham, and T. Wennekers, “A spiking self-organizing map combining stdp, oscillations, and continuous learning,” *Neural Networks and Learning Systems, IEEE Transactions on*, vol. 25, pp. 894–907, 05 2014.

[53] H. Hazan, D. Saunders, D. Sanghavi, H. Siegelmann, and R. Kozma, “Unsupervised learning with self-organizing spiking neural networks,” 07 2018.

[54] S. Song, K. Miller, and L. Abbott, “Competitive hebbian learning through spike timing-dependent plasticity,” *Nature neuroscience*, vol. 3, pp. 919–26, 10 2000.

[55] Digilent, *Zybo Z7Board Reference Manual.*

[56] Wikipedia contributors, “Advanced extensible interface.” https://en.wikipedia.org/wiki/Advanced_eXtensible_Interface, 2021. [Online; accessed 19-May-2021].

[57] Developers, “FreeRTOS.” https://freertos.org/, 8 2021.

[58] R. Gary Leonard and G. Doddington, “Tidigits ldc93s10,” 1993. Philadelphia: Linguistic Data Consortium.

[59] Y. LeCun, C. Cortes, and C. Burges, “The mnist database of handwritten digits.” http://yann.lecun.com/exdb/mnist/, 1998.

[60] P. Warden, “Speech commands: A dataset for limited-vocabulary speech recognition,” *https://arxiv.org/abs/1804.03209*, 04 2018.

[61] C. G. Northcutt, A. Athalye, and J. Mueller, “Pervasive label errors in test sets destabilize machine learning benchmarks,” 2021.

[62] N. Günther, “Cleaned up version of the speechcommand_0.2 dataset.” https://www.kaggle.com/guntherneumair1/speechcommandv02-cleaned, 08 2018.

[63] P. Prashanth and P. Swamy, "Architecture of adders based on speed, area and power dissipation," in *2011 World Congress on Information and Communication Technologies*, pp. 240–244, 2011.

[64] R. Jain and D. Chand Gupta, "Design and analysis of generic architecture of multipliers," vol. 3, 2014. ISSN 2278-0181.

[65] D. Bariamis, D. Maroulis, and D. Iakovidis, "Adaptable, fast, area-efficient architecture for logarithm approximation with arbitrary accuracy on fpga," *Signal Processing Systems*, vol. 58, pp. 301–310, 03 2010.

[66] C.-e. D. Lin, "8 simple techniques to prevent overfitting." https://towardsdatascience.com/8-simple-techniques-to-prevent-overfitting-4d443da2ef7d?gi=953a9931aa07, 6 2020. Accessed: 2021-8-4.

Additional Figures

A

A.1 PDM to PCM

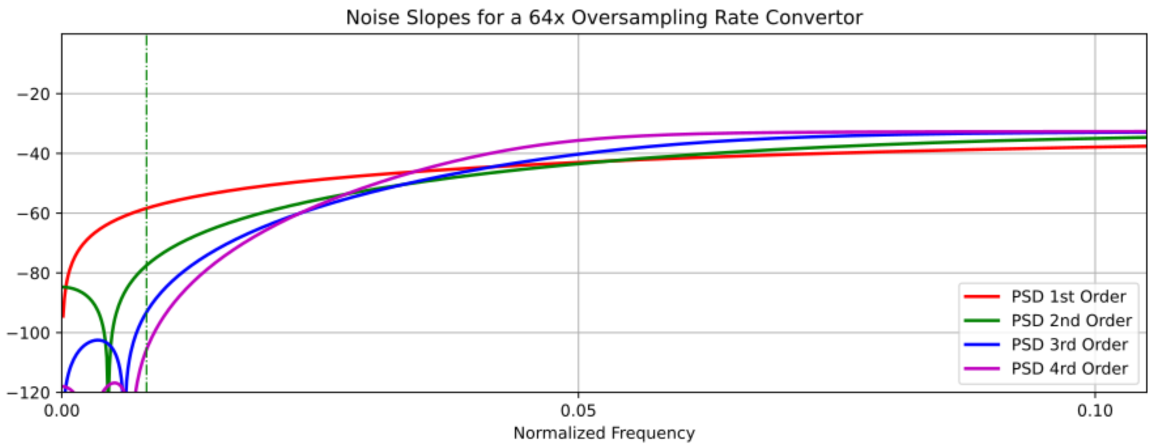


Figure A.1: Noise slopes for different order ΣΔ modulators [21]

A.2 Prototype

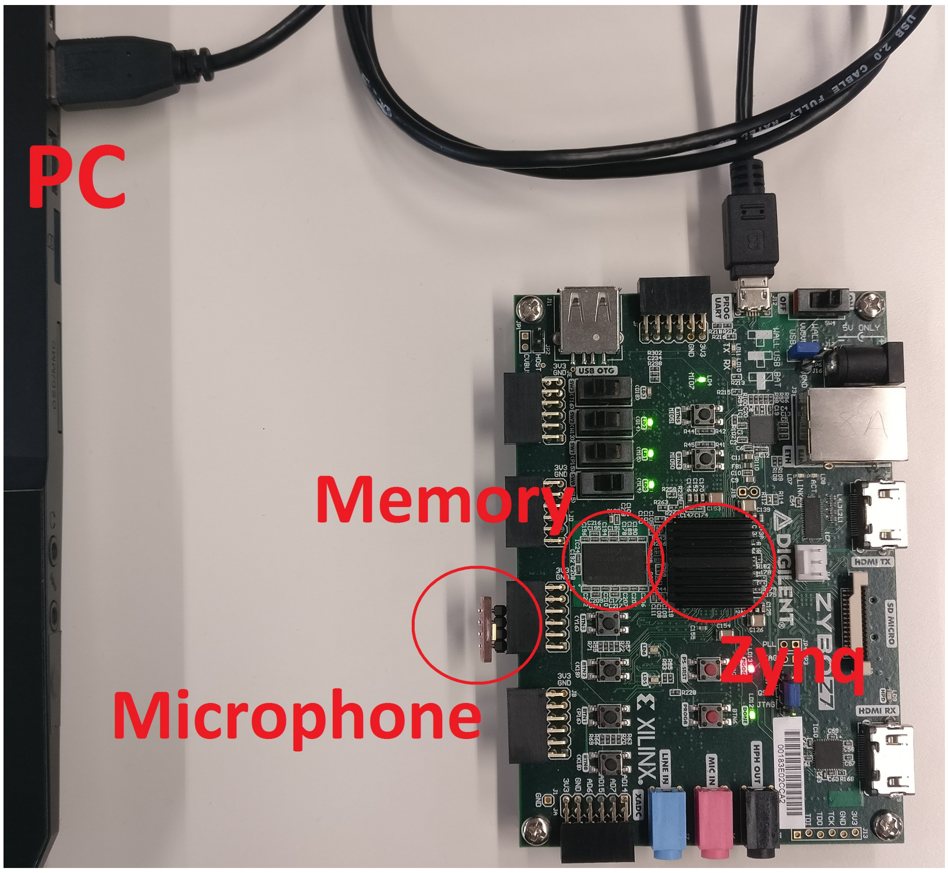


Figure A.2: Annotated prototype setup, where the Zynq includes both the FPGA and CPU components

B

Sigma-Delta Python code

```python
import numpy as np

def pcm_to_pdm(data, oversampling):
    data, max_arr, min_arr = normalize_array(data)

    out = [0] * ((len(data)-1) * oversampling)

    qe = 0
    for i in range(len(data)-1):
        start = data[i]
        stop = data[i+1]

        values = np.linspace(start, stop, oversampling, endpoint=False)

        for j in range(oversampling):
            temp_in = values[j]

        if temp_in > qe:
                temp_out = 1
        else:
            temp_out = -1
        qe = temp_out - temp_in + qe

        value_out = temp_out

        out[i*oversampling + j] = value_out
        return out, max_arr, min_arr
```

C Comparison Calculations

C.1 PDM to PCM computational cost

C.1.1 direct FIR

C.1.1.1 Adders

In the direct FIR implementation each tap except the first requires one addition per input. When assuming a fully parallel implementation we will need 127 adders.

C.1.1.2 Multipliers

No multipliers are required.

C.1.1.3 Storage

Each tap except the first requires an unique 1-bit register, requiring 127 1-bit registers in total.

It is assumed that the coefficients are no stored in memory but that individual bits are fixed to VDD or ground.

C.1.1.4 Add/s

The FIR filter runs at 512 kHz, for each clock cycle all 127 adders need to be executed once. Resulting in 65024000 Add/s required.

$$\text{Add/s} = \text{clock frequency} * \text{Additions per clock cycle} \tag{C.1}$$

C.1.1.5 Mul/s

No multipliers present.

C.1.2 Polyphase FIR

C.1.2.1 Adders

The polyphase FIR runs at 512 kHz but at each clock cycle only a single polyphase structure is used. Since the example, with 128 taps and 64 times oversampling, has 2 taps per polyphase, only 2 adders are executed each clock cycle. Thereby requiring 2 adders in total since they can be reused in subsequent clock cycles.
Moreover, note that the adders for the taps can be significantly smaller compared to

the accumulating adder, approximately 10 versus 14 bits respectively.

$$\text{addders} = \left\lceil \frac{\text{taps}}{\text{oversampling}} \right\rceil \tag{C.2}$$

C.1.2.2 Multipliers

No multipliers required.

C.1.2.3 Storage

For the polyphase FIR 64 1-bit input signals need to be stored, one for each tap, except the first in the polyphase structures. Besides this, there is an accumulating register of about 14 bits. Thus in total 78 1-bit registers are required.

The assumption here is that the coefficients do not require a register since they can be directly wired to VDD or ground. If this is not true, an additional of $64 * 9 = 567$ 1-bit registers will be needed. This is estimated from the fact that this FIR is a casual filter and thus only half of the coefficients are unique. From inspection of the coefficients it is moreover noticed that only the last 9 bits contain information.

The measure for storage is thus defined as:

$$\text{storage} = \text{oversampling} + \#\text{bits} \tag{C.3}$$

C.1.2.4 Add/s

Since the polyphase FIR runs at 512kHz, and each adder runs every clock cycle, 1024000 Add/s are required.

$$\text{Add/s} = \text{clock frequency} * \text{Additions per clock cycle} \tag{C.4}$$

C.1.2.5 Mul/s

No multipliers present.

C.1.3 CIC

C.1.3.1 Adders

Assuming a 3th order CIC we will need six adders, three for the integrate section and another three for the comb section. Requiring six adders in total.

C.1.3.2 Multipliers

No multipliers required.

C.1.3.3 Storage

To accommodate the growth of values, each delay should have a width of 19 bits. In this example with $N = 1$ it is found that there are 6 delays in total thus 114 1-bit registers are required.

C.1.3.4 Add/s

The integrate section runs at 512kHz, executing 3 adders at each clock cycle, thus 1536000 Add/s. The comb section runs at 8kHz thus 24000 Add/s. In total 1560000 Add/s are required.

$$\text{Add/s} = \text{clock frequency integrate} * \text{Additions per clock cycle integrate} + \text{clock frequency comb} * \text{Addit} \tag{C.5}$$

C.1.3.5 Mul/s

No multipliers present.

C.1.4 CIC + polyphase FIR

C.1.4.1 Adders

Each adder for the CIC is a 19-bit adder. where 2 adders are required for the integrate section and 2 for the comb section.

The polyphase FIR requires, according to Equation C.2, 2 additional adders.
In total requiring 6 adders for the CIC and polyphase FIR combined.

C.1.4.2 Multipliers

The CIC does not contain multipliers.

The polyphase FIR will require multipliers for multiplication with its coefficients since the input is now a multi-bit value. But since each polyphase structure contains two coefficients there are also two multipliers required.

C.1.4.3 Storage

Using Equation 4.7 it is found that the CIC requires four 9-bit register, in total thus 36 1-bit registers.

From Equation C.3 it is calculated that 30 1-bit registers are needed for the polyphase FIR.

Summing these, indicates a storage of 66 1-bit registers.

C.1.4.4 Add/s

The integrate section runs at 512kHz with 2 adders, thus 1024000 Add/s. The comb section runs at 32kHz with 2 addders, thus 64000 Add/s. Requiring 1088000 Add/s in total for the CIC.

The polyphase FIR uses 2 adders at 8kHz thus 16000 add/s.

In total requiring 1104000 add/s

C.1.4.5 Mul/s

The polyphase FIR has 2 multipliers thus 16000 mul/s.

C.2 Filterbank filtering computational cost

C.2.1 Time-domain

C.2.1.1 Add/s

There are 20 filters and each filter has 20 taps. Since a frame consists of 1600 samples, 64000 additions are required per frame.

$$\textbf{Add/s} = F * N * T * \frac{\text{freq}}{\text{N}} \tag{C.6}$$

Where F, N and T stand for the amount of filters, the frame size and the amount of taps per filter respectively. The frequency is fixed at $8kHz$.

A frame has a length of 160 samples, taking the sum requires 160 additions per frame.

$$\textbf{Add/s} = N * \frac{\text{freq}}{\text{N}} \tag{C.7}$$

Where N stands for the frame size.

Thus a total of 3208000 additions per second are required.

C.2.1.2 Mul/s

There are 20 filters and each filter has 20 taps. Since a frame consists of 1600 samples, 64000 multiplications are required per frame.

$$\textbf{Mul/s} = F * N * T * \frac{\text{freq}}{\text{N}} \tag{C.8}$$

Where F, N and T stand for the amount of filters, the frame size and the amount of taps per filter respectively. The frequency is fixed at $8kHz$.

A frame has a length of 160 samples, taking the power requires 160 multiplications per frame.

$$\textbf{Mul/s} = N * \frac{\text{freq}}{\text{N}} \tag{C.9}$$

Where N stands for the frame size.

Thus a total of 3208000 multiplications per second are required.

C.2.2 Frequency-domain

C.2.2.1 Add/s

According to [14], $2N$ additions per stage and $\lceil \log_2(N) \rceil$ stages are required for a N point DFT. Assuming a 160-point DFT, this results in 2560 additions per frame.

$$\textbf{Add/s} = 2N * \lceil \log_2(N) \rceil * \frac{\text{freq}}{\text{N}} \tag{C.10}$$

The Mel-scaled filters approximately cover the frame twice, where each filter overlaps half of the neighboring frequency range, so each frame sample is approximately added in two unique frames.

$$\textbf{Add/s} = 2N * \frac{\text{freq}}{\text{N}} \tag{C.11}$$

Where N stands for the frame size.

Thus a total of 144000 additions per second are required.

C.2.2.2 Mul/s

A time frame of 160 samples is taken from the input and a hamming window is applied. The hamming window has to multiply each sample with an unique constant, requiring 160 multiplications per frame.

$$\textbf{Mul/s} = N * \frac{\text{freq}}{\text{N}} \tag{C.12}$$

According to [14], $\frac{N}{4}$ complex multiplies per stage and $\lceil \log_2(N) \rceil$ stages are required for a N point DFT. Assuming a 160-point DFT, 320 multiplies per frame are required.

$$\textbf{Mul/s} = \frac{N}{4} * \lceil \log_2(N) \rceil * \frac{\text{freq}}{\text{N}} \tag{C.13}$$

The Mel-scaled filters approximately cover the frame twice, where each filter overlaps half of the neighboring frequency range, so each frame sample is approximately multiplied by two unique constants.

$$\textbf{Mul/s} = 2N * \frac{\text{freq}}{\text{N}} \tag{C.14}$$

Where N stands for the frame size.

Thus a total of 32000 multiplications per second are required.