

Hardware Acceleration of Active Noise Control Algorithms

Von der Fakultät Elektrotechnik
der Helmut-Schmidt-Universität/
Universität der Bundeswehr Hamburg
zur Erlangung des akademischen Grades
eines Doktor-Ingenieurs
genehmigte

DISSERTATION

vorgelegt von

Alexander Georg Klemd, M.Sc. MBA
aus Köln

Hamburg 2022

Gutachter

Prof. Dr. Bernd Klauer

Prof. Dr. Delf Sachau

Vorsitzender der Prüfungskommission:

Prof. Dr. Udo Zölzer

Tag der mündlichen Prüfung:

24.05.2023

Abstract

Since the principle of active noise control was proposed for the first time in the year 1936, the implementation of these ideas always lagged behind in time due to a lack of sufficiently performant computers. With the introduction of digital computers and their rapid advancement until today, applied research projects and commercial products became well-established. However, active noise control applied to complex and large-scale structures is still limited by suitable implementation technologies to this date. The suitability is foremost characterized by raw processing power, but also by the resulting processing delay and energy efficiency. Besides, some research suggests that the noise reduction performance of well-established applications in active noise control can also benefit from additional computational power.

It is the goal of this thesis to narrow this technology-based gap for a wide range of active noise control applications by optimizing existing control algorithms down to the hardware level or rather the register-transfer level. The custom hardware architectures are realized using field-programmable gate arrays (FPGAs) and provide generic parameters to adjust the algorithm and the internal processing architecture to favor various applications. In case of the feedback FxLMS architecture, a short processing delay is achieved that is independent of any filter length. The performance of the proposed hardware architectures are validated against established software reference systems on experimental setups using identical configurations. The resources of different FPGA platforms are then fully utilized to indicate the corresponding benefits in terms of noise reduction performance for these experimental setups. For the feedback FxLMS algorithm, the most computational demanding, synthesized configuration uses 11 error sensors, 11 secondary sources and filter orders of 2049 at a sampling rate of 48 kHz. For this configuration, this translates to delay times well below 1 μ s. On multiple practical applications, the results show an additional noise reduction performance of up to 12 dB compared to the maximum possible configurations of the software reference.

Acknowledgments

This thesis is the result of my work at the Institute of Computer Engineering at the Helmut-Schmidt-University / University of Federal Armed Forces Hamburg.

Firstly, I am very grateful to Prof. Dr. Bernd Klauer, my Chair, for his guidance to successfully complete my dissertation.

Secondly, I would like to thank the remaining members of my dissertation committee Prof. Dr. Sachau and Prof. Dr. Zölzer.

Thirdly, I would like to thank all my current and former colleagues at Helmut-Schmidt-University with whom I worked closely for countless hours in the laboratory and discussed various topics concerning this thesis: Jonas Hanselka, Johannes Timmermann, Michael Sandner, Patrick Nowak, Piero Rivera Benois, Marcel Eckert, Dominik Meyer and Christina Sander.

Finally I would like to thank my wife Natalie and both of my sons Lian and Niklas for giving me the support and the balance I needed.

Contents

Abstract	i
1. Introduction	1
1.1. Thesis Objectives	3
1.2. Thesis Structure	3
2. Active Noise Control	5
2.1. Algorithms to Perform Active Noise Control	6
2.1.1. Single-Channel Feedback FxLMS	7
2.1.2. Multi-Channel Feedback FxLMS	9
2.2. Offline-Estimation of the Secondary-Path	11
2.2.1. Adaptive System Identification	12
2.2.2. System Identification Using Exponential Sine Sweeps	13
2.3. Resource Requirements for Presented Algorithms	17
3. Potential Hardware Platforms	21
3.1. Digital Signal Processor	22
3.2. Graphics Processing Unit	23
3.3. Field-Programmable Gate Arrays	24
3.3.1. Configurable Logic Blocks	26
3.3.2. DSP-Slices	27
3.3.3. Block-RAM Tiles	28
3.4. Application-Specific Integrated Circuits	29
3.5. Discussion	30
4. Hardware Architectures for FPGA platforms	33
4.1. Related Works and Design Decisions	33
4.1.1. Related Works	33
4.1.2. Design Decisions	38
4.2. Single-Channel Feedback FxLMS	44
4.3. LMS Modifications	54
4.4. Multi-Channel Feedback FxLMS	59
4.5. System Identification via LMS	64
4.6. System Identification using Exponential Sine Sweeps	65
4.6.1. Sine Sweep Generation	66
4.6.2. Amplitude Fade	69

4.7.	Transfer of the Secondary Path from System Identification to Controller	70
4.7.1.	Solution I - Serial Communication	71
4.7.2.	Solution II - Dynamic and Partial Reconfiguration	72
4.7.3.	Solution III - Bit-file Manipulation	74
4.8.	Used Verification Methodology and Toolchain	75
5.	Experimental Results and Evaluation	77
5.1.	Used Platforms	77
5.2.	Feedback FxLMS Controllers	79
5.2.1.	Configuration Strategies	79
5.2.2.	Configurations and Synthesis Results	79
5.2.3.	Interpretation of Utilized Hardware Resources	84
5.2.4.	Experimental Setup	86
5.2.5.	Validation and Noise Reduction Performance	86
5.3.	System Identification Using Sine Sweeps	92
5.3.1.	Configurations and Synthesis Results	92
5.3.2.	Validation and Noise Reduction Performance	92
5.4.	Evaluation of the Transfer Solutions of the Secondary Path Data	95
6.	Conclusion	99
6.1.	Summary	99
6.2.	Future Prospects	100
	Bibliography	101
	A. Appendix	109
	Acronyms	113
	List of Latin Symbols	115
	List of Greek Symbols	117
	List of Figures	119
	List of Tables	121

1. Introduction

The idea to perform active noise control (ANC) is not new and the principle was first described in a patent in 1936. There were no relevant publications in the next two decades, but in the 1950s different approaches to attenuate unwanted noise were proposed. The first demonstrators were built for research purposes using analog electronic circuits. However, using an analog circuit is challenging as the calculation of the antinoise signal requires sufficient accuracy, which these circuits lacked. Their calculations were impacted by unwanted effects, such as the components temperature, -aging and -tolerance. For the next 30 years the research interest in ANC suspended again, not due to a lack of interest or knowledge, but due to a lack of technology, as Elliott infers in [EN93]. With the advancements in digital technologies, research in ANC made a comeback with the introduction of digital signal processing, which enabled more robust systems with higher accuracy. [EN93] In the 1980s first commercial systems appeared whose patents and ideas often originated back to the 1950s. Those systems included active headsets for aviation purposes or were applied into industrial ducts. Headsets and ducts are structures that are characterized by either their small size or their simple structure where the attenuation of unwanted, propagating sound waves can often be simplified to a one dimensional problem. In addition, small-scale applications imply short acoustic paths. Both characterizations result in a lower computational complexity for the applied ANC-system. [KH98]

Today, the computational performance of digital signal processings (DSPs) developed immensely and suitable platforms for ANC-systems became more energy efficient and cheaper. Yet the application of ANC systems to larger and more complex structures remains a great challenge even for high-end, specialized digital platforms. Using the example of ANC applied to windows with the goal to attenuate noise in rooms of buildings, a review from Lee et al. in [Lee+21] shows that researchers today have to apply a variety of physical modifications to the window structure in order to lower their acoustic complexity and achieve acceptable noise reduction results. It is thus reasonable to argue that for large-scale, high performance ANC

applications, a lack of suitable technology is still present.

Beyond that, some studies suggest that increasing the sampling rate above the well-known 48 kHz brings additional benefits in terms of noise reduction performance and stability to an ANC-system [LHS18; CT06]. This means that traditional ANC applications can also benefit from additional computational performance. The simulative study of Lorenzen et al. shows an increasing noise reduction performance up to a sampling rate of 300 kHz in table 1.1 for feedback ANC with band-limited Gaussian white noise. The table also shows that with an increasing sampling rate it is sensible

Table 1.1.: Simulation results that show the normalized error signal power over a series of simulations with varying sampling frequencies and filter lengths from Lorenzen in [LHS18].

		Adaptive filter order					
		100	200	400	800	1000	1600
Sample frequency in kHz	10	0.6835	0.6639	0.6746	0.6926	0.6990	0.7248
	20	0.5112	0.5139	0.5229	0.5394	0.5465	0.5697
	40	0.4737	0.4486	0.4397	0.4432	0.4555	0.4739
	80	0.4907	0.4704	0.4330	0.4008	0.4022	0.3986
	100	0.5201	0.5006	0.4596	0.4219	0.4052	0.3977
	300	0.5616	0.5761	0.5708	0.5639	0.5645	0.5636

to increase the filter lengths proportionally, so that the ANC controller can monitor the same time window. Each of these modifications raise the computational load even further.

Despite raw computational power, there are more requirements to the underlying hardware platform of an ANC-system. Firstly, the processing hardware is often integrated as an embedded system into the physical setup with respective limitations to power consumption and cooling potential. Secondly, the noise reduction performance varies with the exact configuration of the algorithm and the given application. This configuration can not be found purely analytically and it is thus necessary to tune the parameters and the sampling rate of the ANC-system to find an optimum configuration empirically for each application. Without any experience about a certain ANC application, some adjustability of the ANC-system is important. Lastly, for transparent structures like windows, it is often not feasible to install feedforward-based ANC-systems due to the placement of additional reference sensors that can obstruct the view. A feedback approach avoids this problem, but is more complex and sensitive to path delays of the secondary paths, which includes

processing delays.

The knowledge of the required algorithms is there and so is the demand to reduce the increasing noise pollution in densely populated areas, as a survey from Hanselka in [Han20] using the example of the German city Hamburg suggests. Passive noise control measures do not work well at lower frequencies and requires thicker, more expensive absorber material. Hence, ANC does make sense as a complementary technology to passive silencing, as it is most effective in the lower frequency range.

1.1. Thesis Objectives

The goal of this thesis is to narrow the described technology-gap for high-performance ANC with a wide variety of applications in mind. This thesis focuses on available technologies that are suited to apply feedback ANC-systems to large-scale structures using increased sampling rates where possible. The ideal technology should provide a high level of computing power that can be utilized to tune different characteristics of the ANC-system. At the same time, the processing delay should be kept as low as possible to favor the feedback approach. A secondary goal is high efficiency to keep the systems power consumption and cooling requirements moderate. The exact configuration of its parameters should stay variable to be suitable for a variety of applications. Using the most promising technology, corresponding architectures are to be developed, implemented and evaluated against related works.

1.2. Thesis Structure

This thesis is structured as follows:

Chapter 2 gives a brief introduction into the principles of digital signal processing for ANC and the definition of the used algorithms for the here presented ANC systems. Some considerations regarding the scaling of the processing requirements with increased parameters are presented.

Chapter 3 introduces and discusses suitable technologies as the underlying hardware platform for the presented algorithms. In particular the technology of an field programmable gate array (FPGA) is explained in the necessary detail to understand important design choices and the resulting synthesis results later on.

Chapter 4 discusses related works, presents the hardware architecture and their functionality. The used verification toolchain including the assessment methodology is briefly explained.

Chapter 5 introduces the used hardware platforms, the experimental setups and shows the conducted experiments for the validation of the hardware designs against established software reference models. The chapter also recommends some configuration strategies for an efficient hardware utilization and presents a selection of configurations, their synthesis results and the achieved noise reduction performance. Chapter 6 concludes this thesis and discusses future prospects of this work.

2. Active Noise Control

ANC of sound uses actuators, such as loudspeakers as secondary sources to cancel undesired noise coming from a primary source. At the secondary sources, sound with an equal amplitude and an inverted phase, also called antinoise, is created and sent in the path of the primary noise. Destructive interference can happen, where the sound waves of the primary and secondary sources superpose each other.

When placing a microphone as an error sensor in the sound field, a control system can be introduced to minimize the residual error locally around the microphone position. It is the task of the control system to generate the antinoise signal that drives the secondary sources. The quality of the noise reduction performance depends on the accuracy of the amplitude and phase of the generated antinoise. For the control system, two main control strategies exist. The feedforward-based controller requires a reference signal of the primary noise $x(t)$ at a far enough distance to do the processing of the secondary signal before the primary signal reaches the secondary sources. Feedback-based systems reconstruct the reference signal from the residual error signal via internal model control and thus do not need additional sensors.

Figure 2.1 illustrates the basic principles of ANC using a digital feedback-based controller. Modern control systems are implemented digitally, as the underlying technology provides increased accuracy and is insensitive to changes in component tolerances or temperature. The controller periodically receives time-discrete samples of the error signal $e(n)$ from the analog-to-digital converter (ADC) and calculates a corresponding antinoise sample $y(n)$. The time between the processing of two samples is determined by the reciprocal of the sampling frequency f_s . The digital-to-analog converter (DAC) converts the sample to an analogue signal $y(t)$ that is emitted by the secondary sources. When the output $y(t)$ arrives at the error sensor, it was transformed by the secondary path S to the predicted disturbance signal $d'(t)$. The secondary path can be described by the electrical and acoustic transfer function $S(z)$ from the secondary speaker input to the microphone output. The local disturbance signal $d(t)$ at the error sensor represents primary noise signal $x(t)$

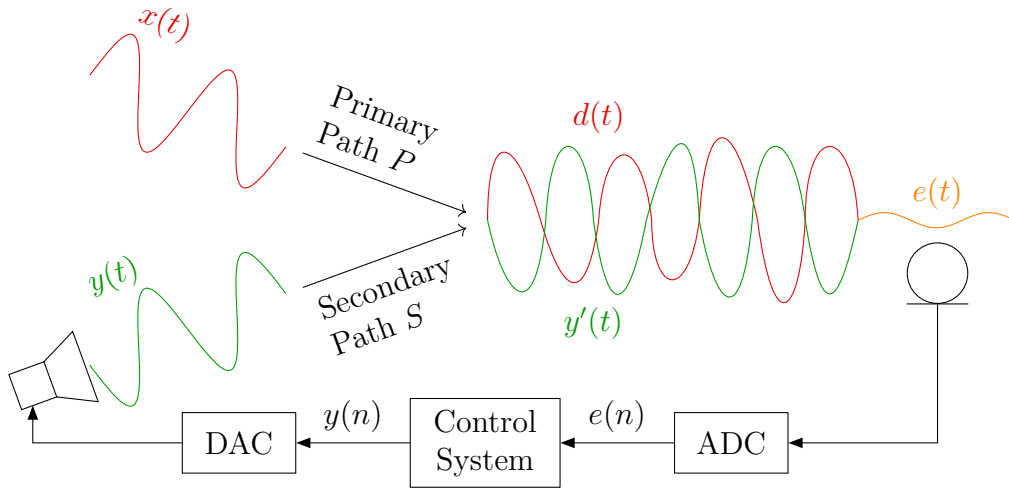


Figure 2.1.: Feedback-based active noise control using a digital controller.

after being transformed along the primary path P .

In order to satisfy the Nyquist-Shannon sampling theorem, anti-aliasing and reconstruction filters are required on the analogue sides of the ADC and DAC. The following block diagrams will be simplified by using only time-discrete signals and omitting the D/A converters. [KM95] [Eli00]

2.1. Algorithms to Perform Active Noise Control

Since this thesis focuses purely on feedback-based ANC-systems, this section introduces the feedback variants of the respective ANC algorithms.

To attenuate time-variant noise sources, the algorithm of the control system needs to use an adaptive filter. It adapts to the statistics of non-stationary signals provided the changes occur slowly compared to the convergence time of the adaptive filter. The two most widely used adaptation algorithms are called least mean squares (LMS) and recursive least Squares squares (RLS). Compared to the LMS algorithm, the RLS algorithm converges faster and leaves less error in the steady state [MH00]. The LMS algorithm is more stable in the steady state and "fairly robust against quantization errors" [Din97]. The latter part will be important for the implementation in section 4.1.2. Choosing the number of multiplications required per iteration as a rating for computational complexity, the RLS has a quadratic order $O(n^2)$ and the LMS has a linear order $O(n)$ of complexity [Mey14]. As a result, the LMS is chosen to adapt a transversal filter. In addition to modeling the primary path, the

control algorithm also has to compensate for the secondary path, since error sensors, secondary sources and the implementation of the control algorithm itself do not behave ideally and without delay. The filtered reference least mean squares (FxLMS) algorithm extends the adaptive filter for this functionality and is introduced in the following section.

2.1.1. Single-Channel Feedback FxLMS

This section focuses on the FxLMS algorithm using a single input and output (SISO) channel. Figure 2.2 shows the adaptive finite impulse response (FIR) filter $W(z)$, updated by the LMS algorithm, and both static FIR filters $\hat{S}(z)$ that are an estimate of the secondary path $S(z)$. The secondary-path transfer function $S(z)$ from $y(n)$ to $e(n)$ includes the D/A converters, the microphones, anti-aliasing and reconstruction filters, a power amplifier, loudspeaker, and the acoustic transfer function from loudspeaker to the microphone. The adaptive and static FIR filters each per-

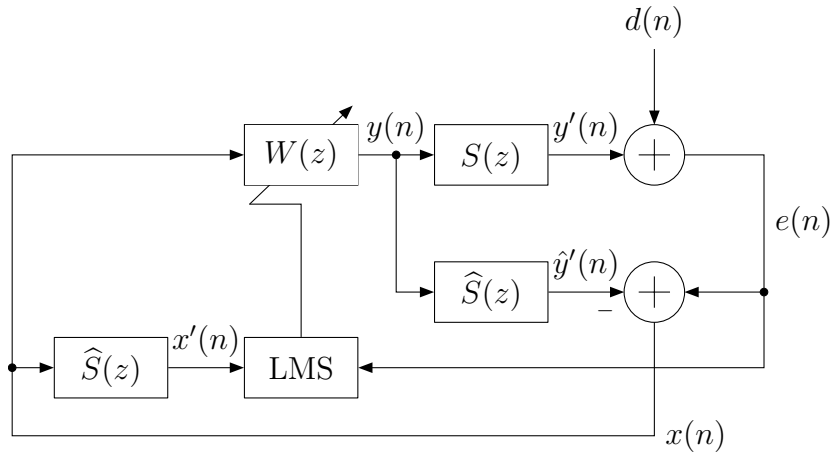


Figure 2.2.: Block diagram of the feedback FxLMS SISO algorithm [KM95, p. 64].

form a discrete-time convolution with an L -sample buffer. This operation is also often referred to as sum-of-products or multiply-accumulate (MAC). Accordingly the adaptive FIR filter computes the antinoise signal at the n -th iteration

$$y(n) = \sum_{l=0}^{L_{\text{adapt}}-1} w_l(n) x(n-l), \quad (2.1)$$

where L_{adapt} is the filter length of the adaptive filter. The filter length must be of sufficient order to accurately model the response of the primary path. Each iteration

the adaptive filter coefficients are then updated through the LMS algorithm to

$$w_l(n+1) = w_l(n) - \mu x'(n-l) e(n), \quad l = 0 \dots L_{\text{adapt}} - 1. \quad (2.2)$$

To obtain the necessary reference signal

$$x(n) = e(n) - \sum_{l=0}^{L_{\text{static}}-1} \hat{s}_l y(n-l-1) \quad (2.3)$$

and the filtered reference signal

$$x'(n) = \sum_{l=0}^{L_{\text{static}}-1} \hat{s}_l x(n-l), \quad (2.4)$$

two static FIR filters are used. The parameter L_{static} is the filter length of the static filters. Both use the transfer function or impulse response of the secondary path as their static coefficients s .

LMS Modifications

To optimize the convergence time or stability of the adaptive filter, there are various variants of the LMS algorithms. The introduction of a leakage factor $\nu(n)$ and a normalized step size $\mu(n)$ modifies the LMS algorithm from equation (2.2) to become

$$w_l(n+1) = \nu(n)w_l(n) - \mu(n) x'(n-l) e(n), \quad l = 0 \dots L_{\text{adapt}} - 1. \quad (2.5)$$

The normalization of the step size $\mu(n)$ according to the reference signal power is done to change between larger step sizes to improve the tracking of non-stationary signals and lower step sizes to minimize errors on stationary signals. This normalized step size is defined as

$$\mu(n) = \frac{\alpha}{L_{\text{adapt}} \max[\hat{P}_x(n), P_{\text{min}}]}. \quad (2.6)$$

In this equation appears the power estimation of the reference signal

$$\hat{P}_x(n) = (1 - \beta)\hat{P}_x(n-1) + \beta x^2(n). \quad (2.7)$$

In practice the estimation of the power signal should be averaged over time to prevent rapid changes of the adaptive step size. For this reason a smoothing parameter β is introduced in equation (2.7). Kuo et al. call the parameter α a "normalized step size" that should be chosen between

$$0 < \alpha < 2. \quad (2.8)$$

In equation (2.6) a parameter P_{\min} is specified that poses a lower limit for the power estimation $P_x(n)$. This prevents very large step sizes for low power reference signals that lead to instability. The initial power estimate $P_x(0)$ can only be guessed and should be chosen rather large to adjust to the optimal step size value from the bottom up.

Introducing a leakage factor $\nu(n)$ increases the stability of the LMS algorithm by steadily "leaking" a small fraction of the estimated adaptive coefficients. This can prevent a problem called "unbounded parameter estimates" where the coefficients start to drift and at some point overflow. This leads to non-linear distortion due to secondary sources being overdriven and causes instability. Another problem that can be reduced is "stalling" where the gradient estimate is too small to adjust the coefficients. According to Kuo et al., the leakage factor is defined as

$$\nu(n) = 1 - \mu(n)\gamma \quad (2.9)$$

where γ is a weighting factor that tunes the intensity of this leaking mechanism. [KM95; MA95]

2.1.2. Multi-Channel Feedback FxLMS

As mentioned before, traditional ANC-systems are only able to attenuate primary noise locally near the error sensors. To attenuate primary noise in larger volumes or in ducts with larger dimensions, usually multiple channels for the error sensors and secondary sources are required. Feedforward-based systems can also use multiple reference sensors. Those systems can be described using J reference sensors, K secondary sources and M error sensors. Since this work focuses on the feedback variants of the given algorithms where the reference signals are reconstructed from the error signals, we can use the special case $J \stackrel{!}{=} M$ for the upcoming equations. Figure 2.3 gives a general overview of the signal flow with M error microphone and

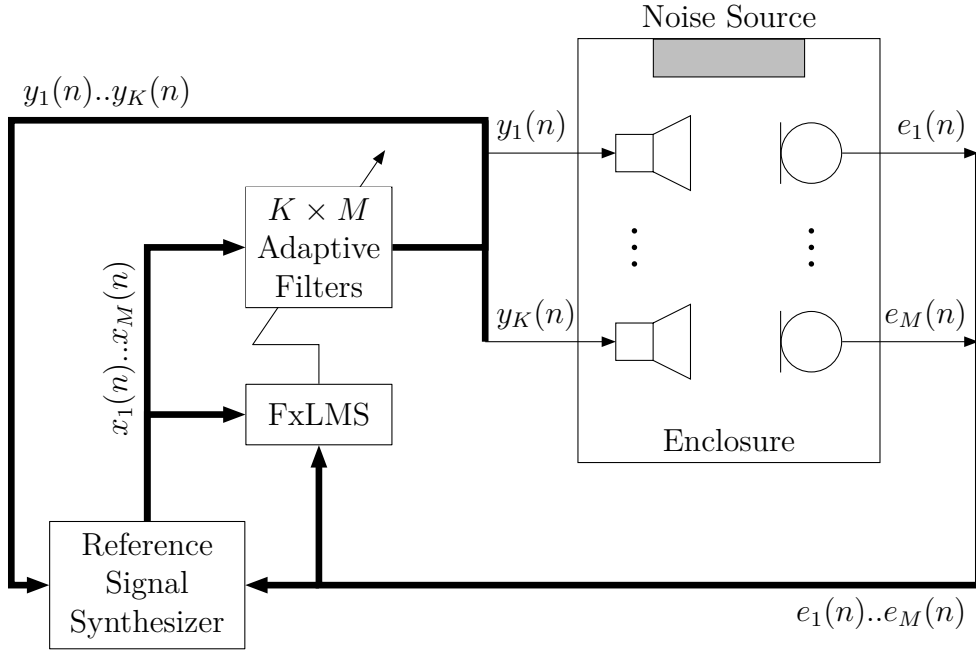


Figure 2.3.: Block diagram of a feedback multi-channel ANC-system using the FxLMS algorithm. The reference signal synthesizer implements equation (2.11) for each of the M error channels. [KM95]

K secondary sources.

The driving signals $y_k(n)$ for each channel are calculated from the reconstructed reference signals $x_j(n)$ and the adaptive filter coefficients $w_{km}(n)$ as shown in equation (2.10)

$$y_k(n) = \sum_{m=1}^M \sum_{l=0}^{L_{\text{adapt}}-1} w_{l,km}(n) x_j(n-l), \quad k = 1 \dots K. \quad (2.10)$$

The reference signals can be reconstructed using the equation

$$x_j(n) = e_m(n) - \sum_{k=1}^K \sum_{l=0}^{L_{\text{static}}-1} \hat{s}_{l,mk} y_k(n-l-1), \quad j = m = 1 \dots M \quad (2.11)$$

where \hat{s}_{mk} are the respective impulse responses of the secondary-path estimates $\hat{S}_{mk}(z)$. The filter coefficients for the MK adaptive filters can be updated with

equation

$$w_{l,kj}(n+1) = \nu_j(n)w_{l,kj}(n) - \mu_j(n) \sum_{m=1}^M x'_{jkm}(n)e_m(n), \quad l = 0 \dots L_{\text{adapt}} - 1,$$

$$j = 1 \dots M, k = 1 \dots K, \quad (2.12)$$

that includes the leakage factor and the step size normalization. In an analogous manner to the single-input and single-output (SISO) variant, the filtered reference signals equal to

$$x'_{jkm}(n) = \sum_{l=0}^{L_{\text{static}}-1} \hat{s}_{l,mk} x_j(n-l), \quad j = 1 \dots M, k = 1 \dots K, m = 1 \dots M. \quad (2.13)$$

The in section 2.1.1 mentioned modifications can also be used for multi-channel variants. Since the adaptive step size, leakage factor and power estimate all depend on the reference signals $x_j(n)$, they have to be calculated for each of the M channels individually.

2.2. Offline-Estimation of the Secondary-Path

It is an essential ability in acoustics to measure the transfer function or impulse response of an unknown system. Most control algorithms, including the FxLMS, require the impulse responses of their secondary paths. The identification of the secondary path impulse responses is easiest and also does not use additional resources of the control algorithm when the identification process is done offline before runtime. In discrete-time, the first K elements of the impulse response of the secondary path or any other causal system $h(n)$ can be described by

$$\hat{h}(n) = \sum_{k=0}^{K-1} h(k)\delta(n-k), \quad (2.14)$$

where $\delta(n)$ denotes the unit impulse. Provided the physical circumstances of the ANC system do not change, the length K can be set to fit the desired length for the model of the secondary path and these values can afterwards be used as coefficients in the static FIR-filters of the FxLMS algorithm. Driving an acoustic system with a unit impulse or rather a very short strong pulse results in non-linear distortion and thus should not be considered in practice. Instead two different implementations to

perform the system identification are presented in the following. Each result in individual advantages and disadvantages when implemented in hardware architectures, which will be referred to in chapter 4.

2.2.1. Adaptive System Identification

The first approach uses an adaptive filter to model the unknown system. This system is getting excited by a generated signal $x(n)$. Usually white noise or band-limited noise in the operating frequency range of the control algorithm is used. The adaption algorithm, in this case again the well-known LMS algorithm, compares the corresponding output of the unknown system $y(n)$ with the filter output $d(n)$ and adapts the filter to any remaining error $e(n)$. The adaptive filter's functionality

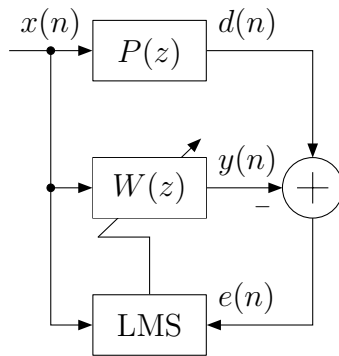


Figure 2.4.: Block diagram of the system identification process using an adaptive filter [KM95].

$$y(n) = \sum_{l=0}^{L-1} w_l(n) x(n-l) \quad (2.15)$$

is identical to the one in the FxLMS component with two exceptions. Firstly, when identifying the secondary path for the FxLMS algorithm, the filter length L corresponds to the filter length of the static filters L_{static} in the control algorithm. Secondly, the incremental steps with which the adaptive coefficients

$$w_l(n+1) = w_l(n) + \mu x(n-l)e(n), l = 0, 1 \dots L-1 \quad (2.16)$$

are being updated, are added to coefficients and not subtracted, as opposed to equation (2.2). The modifications of the LMS algorithm that mainly enhance the filter robustness are not needed here. Since the output of the adaptive filter is

only used internally and does not drive a loudspeaker, the design can not become unstable. As long as the magnitude of the fixed step size is chosen correctly, the adaptive filter will converge. After the residual error is low enough, the adaptive coefficients then represent a model of the unknown system and can be used further.

2.2.2. System Identification Using Exponential Sine Sweeps

The following method of performing a system identification yields advantages regarding the ease of implementation as an standalone system in hardware compared the previous method. Those will be discussed in section 4.6. The use of exponential sine sweeps also provides acoustical advantages in the form of an improved and more homogeneous signal-to-noise ratio (SNR), when compared to pseudo-noise or linear sine sweeps, and is also able to identify weakly non-linear systems that are not perfectly time-invariant. The improved SNR originates from the possibility to detect and separate harmonic distortions from the measurement in post-processing. Among the available techniques for audio and acoustics measurements, it can be considered as a state-of-the-art method. [Far00]

The following mathematical derivation of this algorithm is an excerpt from [Kle+21b]. The excitation signal using exponential sine sweeps can be described as

$$x(n) = A \sin \left(\frac{\Omega_{\text{start}}(L_x - 1)}{\ln(\frac{\Omega_{\text{end}}}{\Omega_{\text{start}}})} \left(e^{\frac{n}{L_x-1} \ln(\frac{\Omega_{\text{end}}}{\Omega_{\text{start}}})} - 1 \right) \right), \quad (2.17)$$

where A is the chosen amplitude, L_x is the desired signal's length in samples, $\Omega_{\text{start}} = 2\pi f_{\text{start}}/f_s$ and $\Omega_{\text{end}} = 2\pi f_{\text{end}}/f_s$ are the starting and ending angular frequencies. The instantaneous angular frequency is the derivative of the argument of the sine function over time, which is given by

$$\Omega(n) = \Omega_{\text{start}} e^{\frac{n}{L_x-1} \ln(\frac{\Omega_{\text{end}}}{\Omega_{\text{start}}})}. \quad (2.18)$$

This equation can be reorganized as

$$\Omega(n) = \Omega_{\text{start}} \theta^n, \quad (2.19)$$

where

$$\theta = \left(\frac{\Omega_{\text{end}}}{\Omega_{\text{start}}} \right)^{\frac{1}{L_x-1}} \quad (2.20)$$

is the factor by which the instantaneous angular frequency increases with each sample step. A companion time-series $x_{\text{inv}}(n)$ is generated, such that the convolution between both results in a scaled unit impulse

$$\sum_{k=0}^{L_x-1} x(k)x_{\text{inv}}(n-k) = C\hat{\delta}(n-L_x-1), \quad (2.21)$$

which is time-shifted by $L_x - 1$ samples and band-limited by f_{start} and f_{end} . The scalar C can be calculated as

$$C = \frac{A^2\pi L_x \left(\frac{\Omega_{\text{start}}}{\Omega_{\text{end}}} - 1 \right)}{2(\Omega_{\text{end}} - \Omega_{\text{start}}) \ln \left(\frac{\Omega_{\text{start}}}{\Omega_{\text{end}}} \right)}, \quad (2.22)$$

following the parameters used for generating $x(n)$ [HCZ09]. The companion time-series $x_{\text{inv}}(n)$ is achieved by applying a time-dependent exponentially-decaying amplitude correction factor given by

$$A_{\text{inv}}(n) = \theta^{-n}, \quad (2.23)$$

to the time-mirrored sine sweep $x(n)$ as

$$x_{\text{inv}}(n) = A_{\text{inv}}(n)x(L_x - 1 - n). \quad (2.24)$$

All in all, the procedure using this measurement technique complies to the following steps: Firstly, the generation and storage of the signals $x(n)$ and $x_{\text{inv}}(n)$ by equation (2.17), equation (2.23) and equation (2.24). Secondly, the excitation of the system by means of $x(n)$ and the recording of the system's response

$$y(n) = \sum_{k=0}^{L_x-1} h(k)x(n-k), \quad (2.25)$$

where $n \in \{0, \dots, L_x + L_h - 2\}$ and L_h is the time required by the system of $h(n)$ to settle down, which can be chosen as the estimated time in samples. Thirdly, the calculation of the convolution between $y(n)$ and $x_{\text{inv}}(n)$ as

$$\tilde{h}(n) = \sum_{k=0}^{L_x+L_h-2} y(k)x_{\text{inv}}(n-k) \quad (2.26)$$

where $n \in \{0, \dots, 2L_x + L_h - 3\}$ and the storage of the result. Fourthly and final,

the extraction of the desired impulse response by

$$\hat{h}(n) = \frac{1}{C} \tilde{h}(n + L_x - 1) \quad (2.27)$$

where $n \in \{0, \dots, L_h - 1\}$, which discards the first $L_x - 1$ samples, keeps the desired length of the impulse response L_h , and divides these samples by the correlation factor in equation (2.22). This process is visualized in figure 2.5.

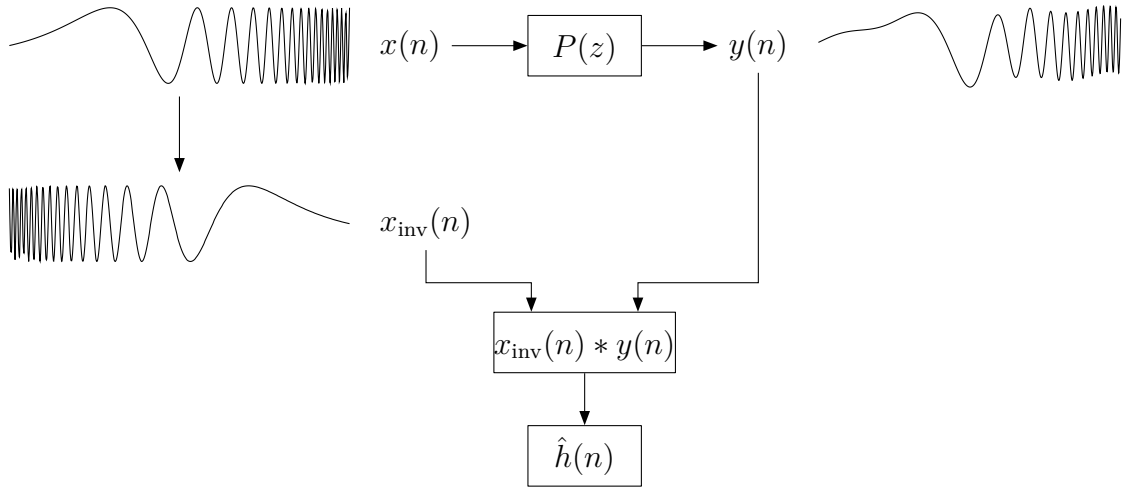


Figure 2.5.: Block diagram of the system identification process using exponential sine sweeps. Note that the use of the amplitude correction factor $A_{\text{inv}}(n)$ and the scalar C is not visualized here.

Generation of Exponential Sine Sweeps

Even before a deeper analysis, one can already notice that the system identification algorithm using exponential sine sweeps is primarily memory intensive given a sufficient processing time. To not having to store all the samples of the regular and time-mirrored sine sweeps in memory, those signals should instead be generated when they are needed. A simple and efficient way to generate a frequency-varying sinusoid is the Gold and Rader filter that can be used as a recursive oscillator [RG67; Zöl89]. It is defined by the difference equations

$$s_1(n) = rbs_1(n - 1) + ras_2(n - 1), \quad (2.28)$$

$$s_2(n) = rbs_2(n - 1) - ras_1(n - 1) + u(n - 1), \quad (2.29)$$

where

$$a(n) = \sin(\Omega(n)), \quad (2.30)$$

$$b(n) = \cos(\Omega(n)) \quad (2.31)$$

are filter coefficients and where $\Omega(n)$ is the angular velocity and should range between $[0, \pi]$ [Gol+69]. In quasi-stable mode, the parameter r can be set to $r = 1$ and an input signal $u(n)$ is not required and thus can be removed, which simplifies the difference equations to

$$s_1(n) = bs_1(n-1) + as_2(n-1), \quad (2.32)$$

$$s_2(n) = bs_2(n-1) - as_1(n-1). \quad (2.33)$$

The initial states of the oscillator

$$s_1(0) = \sin(\phi_0), \quad (2.34)$$

$$s_2(0) = \cos(\phi_0). \quad (2.35)$$

where the initial phase can be set to $\phi_0 = 0$, which in return results in $s_1(0) = 0$ and $s_2(0) = 1$. An oscillating sinusoid can then be obtained by selecting $s_1(n)$ as the output signal. For each iteration n , the angular velocity can then be exponentially increased according to equation (2.19) to obtain Lx exponential sine sweep samples. To generate the time-mirrored or inverted series of sine sweep samples from only the last oscillator state $s_1(L_x - 1)$ and $s_2(L_x - 1)$ and $\Omega(L_x - 1)$, the oscillator can perform an extra iteration and then flip the sign of $s_2(L_x)$ to construct the first state of the inverted sine sweep

$$s_{1,\text{inv}}(0) = s_1(L_x), \quad (2.36)$$

$$s_{2,\text{inv}}(0) = -s_2(L_x). \quad (2.37)$$

This sign flip effectively mirrors the oscillator state at the x-axis of the unit circle, so that it looks from $s_1(n)$ like the oscillator changed direction. For the inverted sine sweep, the angular velocity is now divided at each iteration by θ instead of multiplied, see equation

$$\Omega_{\text{inv}}(n+1) = \frac{\Omega_{\text{inv}}(n)}{\theta}. \quad (2.38)$$

2.3. Resource Requirements for Presented Algorithms

To help choosing the best hardware platform, this section gives the estimated processing power and memory requirements for the given ANC algorithms.

Feedback FxLMS Algorithm

The requirement of the FxLMS algorithm regarding processing power is characterized by the large number of MAC-operations per sample time. The static FIR-filters have to perform $KM(M + 1)L_{\text{static}}$ MAC-operations. The adaptive FIR-filters have to perform KML_{adapt} MAC-operations for the convolutions and $KM(M + 2)L_{\text{adapt}}$ MAC-operations for the LMS adaption. When the modifications of the LMS from section 2.1.1 are enabled an additional $6M$ MAC-operations are required. Taking the available sample time into consideration, the required processing power can be characterized as

$$N_{\text{MAC}}/s = (KM(L_{\text{stat}}(M + 1) + L_{\text{adapt}}(M + 3)) + 6M) f_{\text{clk}}. \quad (2.39)$$

Figure 2.6 shows this metric for the single-channel and the first nine multi-channel feedback FxLMS configurations all using filter lengths of $L = 2049$ and a sampling rate $f_s = 48$ kHz.

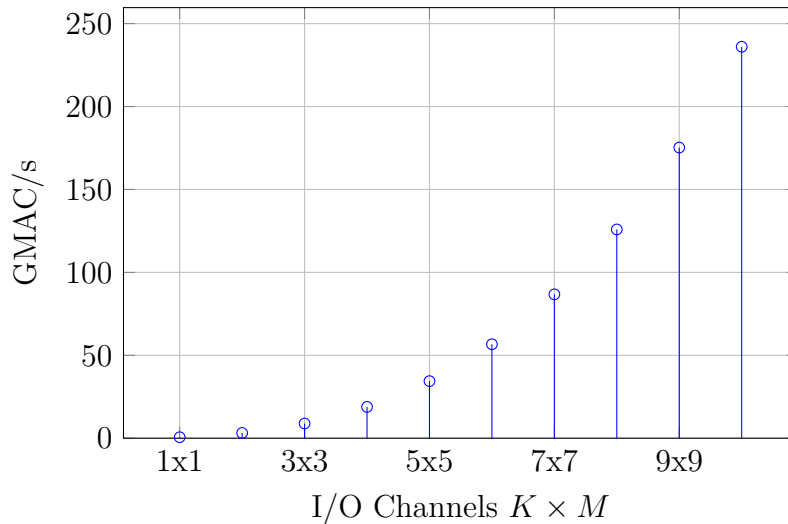


Figure 2.6.: Giga-MAC-operations per second to be performed by the feedback FxLMS algorithm with increasing I/O channels. A filter lengths of $L = 2049$ and a sampling rate of $f_s = 48$ kHz is used.

The memory requirement includes the storage of $M\max[L_{\text{static}}, L_{\text{adapt}}]$ samples of the reference signal and KL_{static} samples of the anti-noise signal, KML_{static} static coefficients and KML_{adapt} adaptive coefficients. This calculation includes only the data that is stored in bulk and does not include data that can be stored in one or a few registers. Assuming a bit width of $\text{bw}[x(n)] = 16$ for the reference signal, $\text{bw}[y(n)] = 16$ for the anti-noise signal, $\text{bw}[s] = 32$ for the static coefficients and $\text{bw}[w(n)] = 32$ for the adaptive coefficients, a total number of

$$\begin{aligned} & KM(L_{\text{static}}\text{bw}[s] + L_{\text{adapt}}\text{bw}[w(n)]) + \\ & M\max[L_{\text{static}}, L_{\text{adapt}}]\text{bw}[x(n)] + \\ & KM^2L_{\text{static}}\text{bw}[x(n)] + \\ & KL_{\text{static}}\text{bw}[y(n)] \end{aligned} \quad (2.40)$$

bits are stored in block-memory. For a configuration using 10×10 channels and filter lengths of 2049 this would accumulate to about 5.8 MB of data, which is not much compared to the capacity of main memory found in desktop computers. However, many embedded systems can not provide that amount of storage. In addition, this is the minimum capacity for the underlying memory architecture assuming that the memory interface is fast enough. Depending on the architecture, concurrent access to the memory is required, which results in fragmentation and can decrease effective memory utilization. In fact, concurrent memory access is likely mandatory, as MAC-operations will have to be performed to some degree in parallel to meet the required sample time.

System Identification Using Exponential Sine Sweeps

The system identification using sine sweeps has no real to requirement to perform the convolution in a specific time. Hence, the minimum processing power depends on how fast the impulse response of the system shall be ready. The only fixed requirement for the system is to be able to produce and send the samples of the excitation signal and receive the corresponding system response sample at the rate of the sampling frequency. In terms of required block-memory resources,

$$(T_x f_s + L_h)\text{bw}[x(n)] + L_h\text{bw}[\hat{h}(n)] \quad (2.41)$$

for a sine sweep of $T_x = 10$ s, a bit width of $\text{bw}[x(n)] = 16$ bit for the excitation and system response samples and a bit width of $\text{bw}[h(n)] = 32$ bit for the impulse

response values, the required memory capacity accumulates to about 1.0 MB of data.

3. Potential Hardware Platforms

A variety of integrated circuits (ICs) exist to perform digital signal processing. The underlying hardware platform is chosen based on the following criteria: computational power, processing delay, power consumption, development time and price. Computational power is the most important criteria to enable high-order filters combined with high sampling rates. A reasonable processing delay is required, as there is only limited time to actively reduce noise before the noise has already passed the area to be attenuated. The power consumption has a subordinate relevance during basic research, but is important afterwards. The hardware platform will presumably be deployed as an embedded system and integrated in the physical application. A high power consumption over extended periods is harder to provide and systems often requires active cooling which does result in additional unwanted noise. The development time and costs are less important during the development of a proof of concept and before any commercial use. Finding the optimal platform is always a trade-off between these criteria, as no technology is absolutely superior to the other.

ICs can be categorized by their fabrication technology, see figure 3.1. From these categories, fixed standard-IC can be ruled out, as they are not applicable to different ANC scenarios. From memory-programmable ICs, general purpose processors may provide the necessary computational power. Although their processing power is steadily increasing and more parallel processing cores are added, they lack efficiency compared to specialized processors and their versatility is mostly wasted with the here required tasks. With the rise of general purpose processors (GPPs) in special purpose devices, such as phones, the line between GPPs and specialized processors gets softened and more single instruction-multiple data (SIMD)-instructions that specialize on parallel signal processing are included. As an example, the instruction set for *ARMv8* processors get frequently extended with SIMD-arithmetic instructions aimed at signal processing, such as specialized variants of multiply accumulate instructions [Arm22]. In general, specialized processors should still be preferred for pure ANC applications due to their superior efficiency.

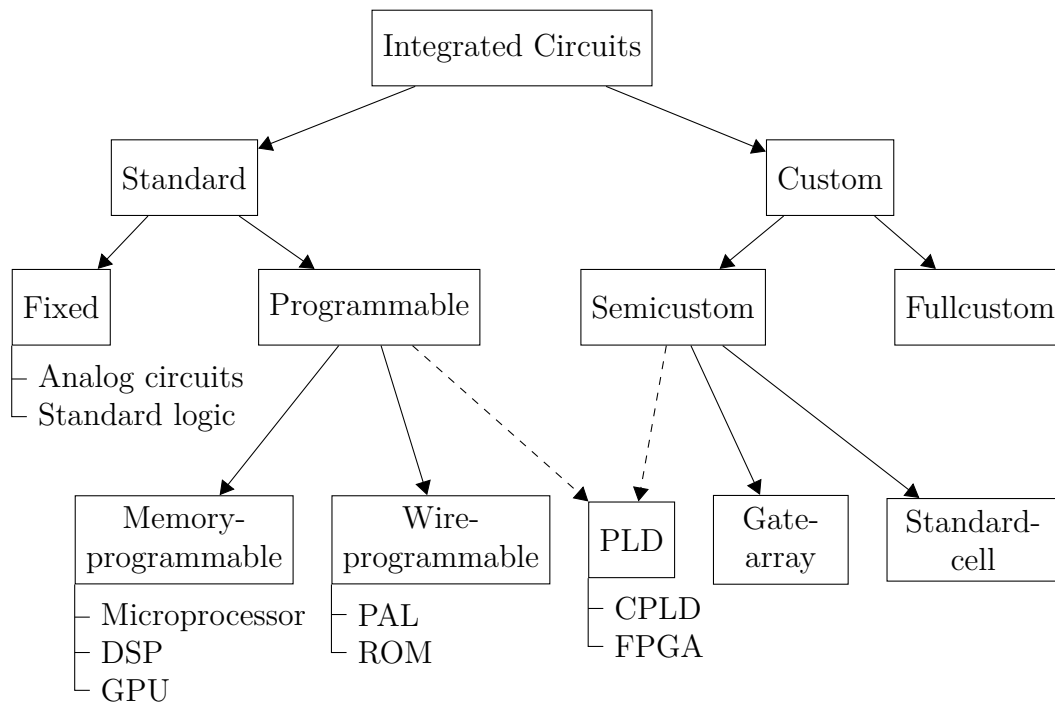


Figure 3.1.: Classification of integrated circuits based on fabrication technology. [Mey14] [KB13]

3.1. Digital Signal Processor

Specialized processors like a DSP feature improved efficiency by providing smaller processing cores with a reduced instruction set to serve a single or few related purposes. This way DSPs can offer more of these cores at similar levels of power consumption or a similar core count at a lower power consumption. Since many application in digital signal processing require similar operations, such as multiplying large value arrays and accumulating them, these cores typically provide fast array multipliers and an accumulator that supports an enlarged bit width. The specializations of DSPs vary a lot and can be divided into their application fields, such as audio- or video-processing. Some DSPs focus on a low power consumption to be used in embedded systems and others are optimized towards high performance. Usually DSPs are limited to process data either in the fixed-point or floating-point number format only. The processors can also be called hardware accelerators, when they are used supplementary in combination with a GPP. Similar to some GPP inheriting more DSP-like instructions, some DSP manufacturers began to integrate GPPs with dedicated hardware accelerators on a single chip, as demonstrated with the *OMAP3* from *Analog Digital* or the *C6000*-series from Texas Instruments. Both

manufacturers are the largest suppliers of DSPs. A common metric to measure the performance in digital signal processing is the number of MAC-operations a processor can perform per second. As a benchmark for DSPs, the high-end *TMS320C674X* from *Texas Instruments* can achieve up to 2.7 GMAC s^{-1} and the *Tigersharc ADSP-TS101S* from *Analog Devices* up to 2.4 GMAC s^{-1} [Mey14] [Ana21].

The development time is comparatively short, as modern DSPs are programmed mostly using high-level languages, such as *C* or *C++*.

3.2. Graphics Processing Unit

Graphics processing units (GPUs) function as a hardware accelerator for video processing and are traditionally used in desktop computers and servers. Real-time video processing requires massively data-parallel processing power to perform arithmetic operations on high-order 2D matrices. Thus GPUs are characterized by a high number of uniform processing cores attached to DRAM-memory in the order of gigabytes. Development time can be considered short, as the major GPU vendors provide libraries like *CUDA* that implement common signal processing algorithms for programming languages, such as *C*, *C++*, and even programming interfaces to *Matlab* to develop software for these custom processors. In general, modern GPUs provide the highest computational performance, as they are able to perform 10^{13} and more floating-point operations per second (FLOPS) [DKK21]. This metric can be roughly compared to MAC operation per second, when considering that it generally takes two floating-point operations to perform one multiply-accumulate operation. GPUs do consume significantly more power per floating-point operation than DSPs and thus lack efficiency. This is not problematic for prototypes but can be problematic for production systems that are energy sensitive or have restrictions regarding active cooling, such as many embedded systems. However, using a GPU for real-time audio applications does also bring obstacles to experimental setups. Firstly, the in- and outgoing audio data can not be fed directly to and from the GPU's memory, but has to be transferred between an audio card, the host central processor and the GPU. This negatively affects the processing delay and can be further impacted by the use of a non-real-time host operating system. Secondly, the architecture of GPUs is designed to process data in larger batches, such as 2D matrices containing image data. Algorithms for audio processing in the time domain generally work on a sample per sample base and thus can not utilize all the processing power. In

[Lor+14b] Lorente et al. present a multi-channel FxLMS controller using a GPU, but conclude that in order to improve his approach a switch to the frequency domain was necessary. This approach is further analyzed in section 4.1.1.

3.3. Field-Programmable Gate Arrays

Compared to DSPs and GPUs, FPGAs do not traditionally include processors that execute instructions of programs, but rather contain arrays of logic blocks and additional hardware resources that can be wired together by configurable interconnections. Thereby FPGAs can mimic arbitrary digital hardware circuits. The term "field-programmable" means that the device can be configured by the customer after leaving the production factory of the manufacturer. The programming can be done by writing code in a hardware description language (HDL), which is then compiled to binary file that defines the state of all programmable elements of the FPGA. Because each FPGA type and vendor uses different architectures, compiling the chip-specific binary file is a proprietary process called "synthesis".

FPGAs belong to the group of programmable logic devices (PLDs), which can be hard to categorize in terms of the fabrication process. In some way PLDs are application-specific ICs and thus can be assigned to the group of application specific integrated circuits (ASICs). However, the term ASIC usually is associated with ICs where the application-specific configuration happens during the fabrication process. At the same time PLDs are programmable devices. They can be either one-time programmable by burning (anti-)fuses or by using read only memory (ROM) or can be programmed many times by using random-access memory (RAM) or flash-memory. The dominant programming technology for FPGAs uses static random-access memory (SRAM)-cells, as they are more durable and area-efficient compared to flash-memory. Other PLDs like programmable logic arrays (PLA) and complex programmable logic devices (CPLDs) can be ignored, as they contain not enough hardware resources for the goal of this work and are aimed at simpler tasks.

In the following sections, the architecture and the available resources in modern FPGAs will be briefly explained to help with the discussion of the hardware utilization of the proposed architectures later on. Taking the architecture of the *7-series* from *Xilinx Inc.* as an example, there are several types of resources arranged on a grid. Each resource type is arranged in columns across the chip. Figure 3.2 shows the prin-

ciple arrangement represented by a few instances of each resource type. The basic

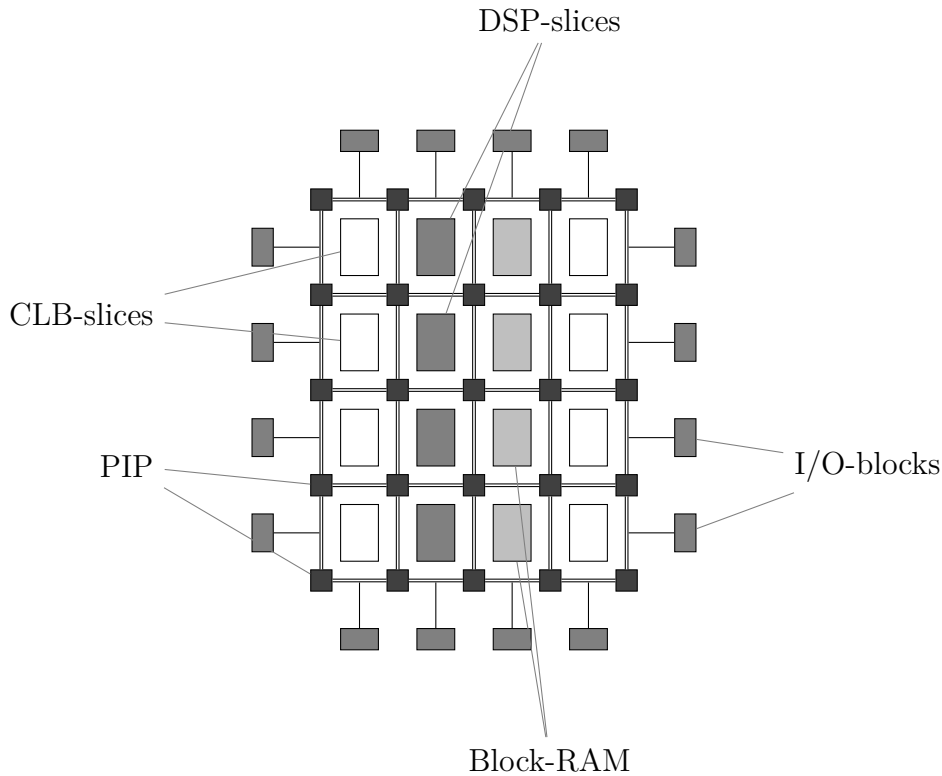


Figure 3.2.: A simplified schematic of the hardware resource arrangement on an FPGA chip.

configurable resources are the configurable logic block (CLB)-slices, the input/output (I/O)-blocks and the programmable interconnect points (PIPs), sometimes also referred to as a switch-matrix. The I/O-blocks can be configured either as inputs or outputs and are the interface from the FPGA-chip to peripheral hardware components. The PIPs build the programmable network between the inputs and outputs of the other hardware resources. In addition, a lot more optional components can be found on an FPGA-chip. Components like phase-locked loops (PLLs) can be used to modify the internal clock frequency, A/D converters, additional flash- or DDR-memory, physical intellectual property (IP)-cores for video tasks or ethernet and even full GPP processors can be found on FPGA-chips. The following section will focus on CLB-slices, the DSP-slices and block-RAM (BRAM)-tiles found in *7-series* FPGAs, as they help in the discussion in chapter 5.

3.3.1. Configurable Logic Blocks

CLBs, sometimes also referred to as logic block elements by other vendors, are elements that can serve to implement boolean logic, distributed memory and flip-flops. The available resources per CLB can be found in table 3.1. Each CLB is

Table 3.1.: Summary of the available logic resources in one CLB. Note that the distributed RAM and the shift-registers can be implemented by using LUTs from the *SLICEM*. [Xil16]

Slices	LUTs	Flip-Flops	Arithmetic and Carry Chains	Distributed RAM	Shift-Registers
2	8	16	2	256 bit	128 bit

organized into two subcomponents that *Xilinx Inc.* calls slices. Figure 3.3 shows the internal organization of the CLB. According to [Xil16] all slices in a CLB contain

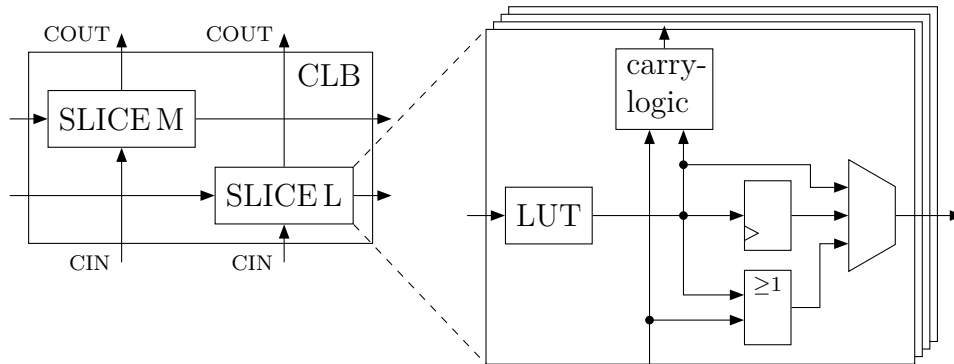


Figure 3.3.: A simplified schematic of a CLB containing two slices of different types. Each slice contains an array of 4 homogeneous elements. The datapath of one element for the *SLICEL* is illustrated on the right. A detailed schematic of CLB can be found in [Xil16].

four 6-input look-up tables, eight storage elements, wide-function multiplexers and carry-logic. The *SLICEL* contains these resources exactly and the *SLICEM* variant contains look-up-tables (LUTs) with additional features. While the *SLICEL*-LUTs can only be used for logic, the *SLICEM*-LUTs can additionally be configured as distributed RAM or as 32 bit shift registers. The input and output ports of the slices within a CLB are not directly connected, but are connected vertically by an independent carry chain with the neighboring CLBs.

3.3.2. DSP-Slices

FPGAs of the *7-series* do include DSP-slices that serve as fine-grained hardware accelerators across the FPGA to perform operations that would be resource-intensive and slow using CLB-slices. *Xilinx Inc.* highlights in [Xil18] the following functionality for the used *DSP48E1*:

- 25 bit \times 18 bit two's complement multiplier
- 48 bit accumulator
- power saving pre-adder
- SIMD arithmetic unit: dual 24 bit or quad 12 bit add/subtract/accumulate
- optional logic unit
- pattern detector
- optional pipelining and dedicated buses for cascading.

Note that the DSP-type of the high-end, *Ultrascale* FPGA use a different type, called *DSP48E2* that enables the same functionality with slightly higher bit widths. The architecture of a DSP-slice can be found in figure 3.4. In the context of FIR

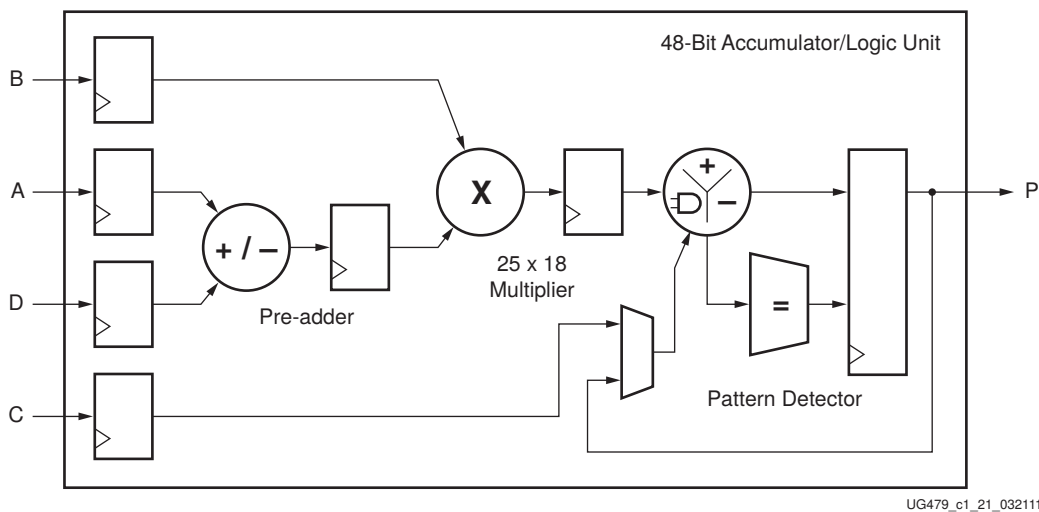


Figure 3.4.: The architecture of the DSP-slice *DSP48E1* used in the *7 series* FPGA from *Xilinx* [Xil18].

filters, the majority of functions is unused. The DSP-slices will be almost exclusively used for the multiplications and to accumulate signals. The multiplier together with the accumulator is able to perform a MAC-operation within a single 100 MHz clock cycle and higher depending on the exact configuration. Note that if the bit width of one of the operands exceeds 18 bit or both operands exceed 25 bit, then multiple

DSPs are cascaded to allow fast multiplications with larger bit widths.

3.3.3. Block-RAM Tiles

The BRAM in *Xilinx 7 series* FPGAs has two ports and can store up to 36 kbit of data. *Xilinx* calls this block *RAMB36E1* for the *7-series*. One *RAMB36E1* block can also be configured as two independent 18 kbit blocks, called *RAMB18E1*. Each of the two asynchronous ports supports to read from and write to arbitrary memory addresses synchronously. The ports of both blocks can be further configured in various combinations of memory depths and bit widths, called aspect ratios. The possible aspect ratios for both types of available BRAM in true dual-port mode (TDP) are presented in the tables 3.2 and 3.3. For these tables it is clear that with

Table 3.2.: Maximum port aspect ratios for *RAMB18E1* in TDP Mode. [Xil19]

Port Data Width	Port Address Width	Depth	ADDR Bus	DI Bus DO Bus	DIP Bus DOP Bus
1	14	16,384	[13:0]	[0]	NA
2	13	8,192	[13:1]	[1:0]	NA
4	12	4,096	[13:2]	[3:0]	NA
9	11	2,048	[13:3]	[7:0]	[0]
18	10	1,024	[13:4]	[15:0]	[1:0]

Table 3.3.: Maximum port aspect ratios for *RAMB36E1* in TDP Mode. [Xil19]

Port Data Width	Port Address Width	Depth	ADDR Bus	DI Bus DO Bus	DIP Bus DOP Bus
1	15	32,768	[14:0]	[0]	NA
2	14	16,384	[14:1]	[1:0]	NA
4	13	8,192	[14:2]	[3:0]	NA
9	12	4,096	[14:3]	[7:0]	[0]
18	11	2,048	[14:4]	[15:0]	[1:0]
36	10	1,024	[14:5]	[31:0]	[3:0]
1 (Cascade)	16	65536	[15:0]	[0]	NA

growing memory requirements, the utilized number of BRAM-tiles does increase

in discrete steps. If a an aspect ratio for a *RAMB36E1* block of $16 \text{ bit} \times 1024$ is supposed, we can increase the bit precision to 18 bit and double the depth of the memory instance without any additional cost. However any further increase in width or depth of the memory instance will result in additional utilization of BRAM-tiles. Hence, to optimize the efficient utilization of block-memory, these maximum aspect ratios need to be respected.

For filter orders higher than $36 \text{ bit} \times 1024$, two *RAMB36E1* blocks can be cascaded to form a deeper memory block without additional clock delay. One distinctive feature of the FPGA technology is that during the configuration sequence of the device, the memory content of the BRAM-tiles can be initialized to arbitrary values and does not have to be accessed sequentially. This is especially useful for the block-memory containing the static coefficients. [Xil19]

3.4. Application-Specific Integrated Circuits

ASICs are ICs fabricated for a specific application or customer, which means that they are usually not freely available like standard-ICs. ASICs can use the same process nodes and thus reach the same level performance as standard-ICs. It is commonly estimated that an ASIC can provide about ten times the performance in terms of clock rates, power consumption and area efficiency compared to FPGAs produced with the equivalent process node [KB13; KR10; Nur+16]. Moreover the amount and mixture of hardware resources can be perfectly adapted to the specific application increasing the efficiency further and enabling even the largest architectures. The downsides are the increased development time due to additional verification and fabrication steps and the non-recurring engineering (NRE)-costs are much higher. While development times for FPGA architectures around three to six months are considered average, for an ASICs the development time on transistor level increases to between one and multiple years [Ok17]. During this development cycle the use of FPGAs is common for prototyping hardware designs. The development costs before the final production can start is estimated at four million US-dollars on average for a 40 nm process node [Mey14].

3.5. Discussion

Analyzing the characteristics of the presented possible hardware platforms, GPUs show the greatest performance in terms of the MAC-performance. However, the processing delay when being used for audio applications is rather high, which impacts the feedback FxLMS algorithm negatively [Haj+18]. Also GPUs are designed for image and video processing, which can limit their full utilization of the processing power for real-time audio applications, as described by Lorente et al. in [Lor+14a]. DSPs designed for audio-applications are efficient and provide tools to minimize the development costs for ANC algorithms. Compared to GPUs and FPGAs, they do lack processing power for massively parallel ANC algorithms. In [Bai+13] Bai et al. performed a study on the throughput and power efficiency of a high-end DSP compared with an FPGA. They conclude that "[...] the parallelized FPGA implementation at 100 MHz outperformed DSP at 1.5 GHz by providing 12.3x speedup while consuming 2.4x less power at same process node (40 nm)". Other researchers do agree that FPGAs can provide more MAC-performance than DSPs [SN06], [Len+18]. Hence Meyer-Baese concludes that DSPs will dominate algorithms with large decision trees, while FPGAs will be preferred for simple arithmetic algorithms that are suitable for parallel execution. [Mey14]

HDLs can configure individual logic blocks and registers down to the register-transfer level (RTL) by adapting the datapath of the processing unit to the respective ANC algorithm. Compared to DSPs and GPUs which can only adapt their order of execution of software instructions, this lower level of abstraction allows for more optimizations, but also increases the complexity and development time, see figure 3.5. Firstly, those optimizations can be used to minimize the processing delay, as the audio samples from the A/D interface can immediately be processed by gates and registers without any management overhead introduced by a generalized processor architecture or even an operating system. Secondly, the utilization of the individual hardware resources per clock can be greatly increased by constructing application-specific data-paths. One example is the presented architecture in section 4.2 that uses a pipelined approach to perform the LMS update of the adaptive coefficients and the convolution. As a comparison, most processing cores of DSP have a fast, multi-stage pipeline architecture intended for multiply-accumulate operations available that can perform the convolution in L instruction cycles, where L is the filter length. However, when the update of the adaptive coefficients is included into consideration, the DSPs architecture can not adapt to this change and consequently the

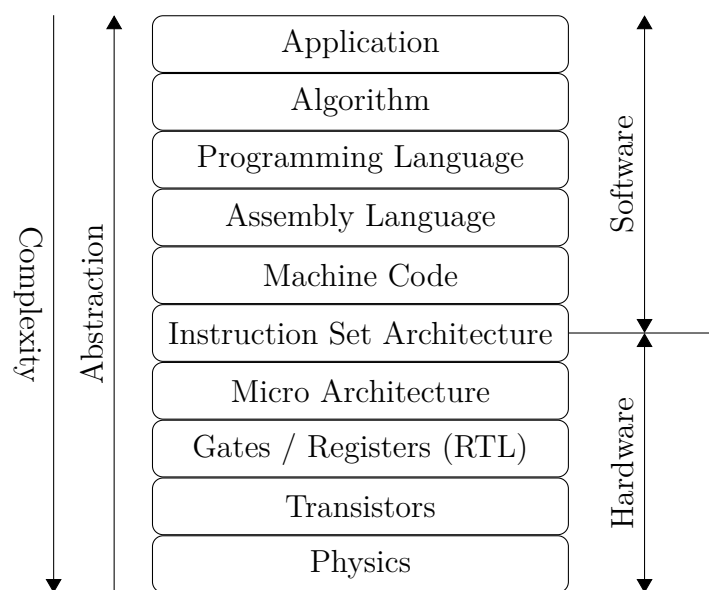


Figure 3.5.: Layers of abstraction in computer systems. The level of abstraction does increase from bottom to top. [Hoz18]

DSP can not use the MAC-pipeline as intended. Instead, it has to load, adapt and store each coefficient before it can process the next coefficient and be accessed by the same or another thread performing the convolution. Those additional instructions per filter tap are for memory-management only and do not contribute to the algorithm directly. They can thus decrease the clock-wise utilization of the arithmetic processing hardware significantly.

Despite all the advantages ASICs provide, due to their prohibitive NRE-costs these chips are not feasible for low volume production. Thus, the following implementations are developed as hardware architectures aimed at FPGAs platforms.

4. Hardware Architectures for FPGA platforms

4.1. Related Works and Design Decisions

4.1.1. Related Works

Single-Channel ANC

Hanselka did an extensive study in [Han20] on hardware implementations of the LMS or FxLMS algorithm for FPGA platforms with single input and output channels published until 2018. He summarizes the architectures as being either fully sequential, parallel or systolic filter architectures of fixed lengths between 2 to 1024 adaptive parameters and one architecture standing out with up to 4096 adaptive parameters. The architectures are either implemented using high level synthesis (HLS), graphically using IP-blocks in software such as Xilinx System Generator or a HDL. None of the architectures make use of generic parameters and the architectures seem to be fixed for a single application only.

The work from Shi et al. in [SSG16] stands out as it is the only other architecture found that features a constant short processing delay. The reason for the short delay lies in the use of transposed filters for the parallel systolic architecture. Systolic architectures that use direct form filters, add processing delay for each filter tap due to the pipelined registers. The systolic architecture of transposed form filters, provides the short processing delay by design, but results in other limitations. Due to the high fanout of the input signal leading to all filter taps, long routes and therefore timing issues resulting in low clock rates in the FPGA-chip are very likely. Also, as Hanselka mentions, the throughput is very high but the respective filter lengths for both filter forms are restricted as each filter tap requires its own hardware

component.

Alongside the study from Hanselka, Santhi published in [SUG19] a technical survey over 14, partly overlapping, publications and comes to a similar conclusion: “In this study, it has been observed that the performance of various adaptive filter structures varies considerably in terms of rate of convergence, throughput rate, improvement in signal-to-noise ratio (SNR), maximum clock speed, hardware complexity, power consumption and cost.” Santhi supports the claim of Hanselka that architectures focus on either fully parallel or fully sequential designs.

Multi-Channel ANC

As can be seen in section 2.3, the processing requirements for ANC-systems using multiple channels quickly challenge even modern platforms. To avoid the limits from these requirements, there are many approaches to reduce this complexity by tweaking the FxLMS algorithm itself [BN03; KP98] or by other techniques, such as multi-rate systems or delay-less subband filtering [BLL02; CD17; KGK12]. These methods usually trade savings of the computational effort for a decrease in noise reduction performance or make assumptions about the ANC-system that limit the application to a few specific usecases. Further discussion will disregard work that tweaks the algorithm and instead will focus on research that optimizes the implementation of the unmodified algorithm. Three multi-channel FxLMS implementations for FPGA platforms were found [Men+19; Shi+19; Ush16].

Mendez et al. propose a design optimized towards using a single reference sensor, 18 error sensors and one output. Static and adaptive filter lengths of 400 are used at a sampling rate of 1 kHz. The authors suggest to use one FPGA-based platform for each secondary source. This results in multiple asymmetric, multi-channel ANC systems that operate in parallel. The proposed architecture scales well, but the design is not fully interconnected and thus the crosstalk between the multiple outputs is not considered when the algorithm calculates the antinoise signal. Hence, using the architecture on multiple platforms in parallel poses a modification of the algorithm.

Ushenina presents a systolic architecture that uses a matrix of modules, see figure 4.1. One module is used for each adaptive filter and contains two DSP-slices as multipliers, one addition and several block-RAM tiles. In the first operation mode both multipliers are used to update the adaptive coefficients. The second operation

mode uses only the first multipliers to calculate the output signal. To perform the

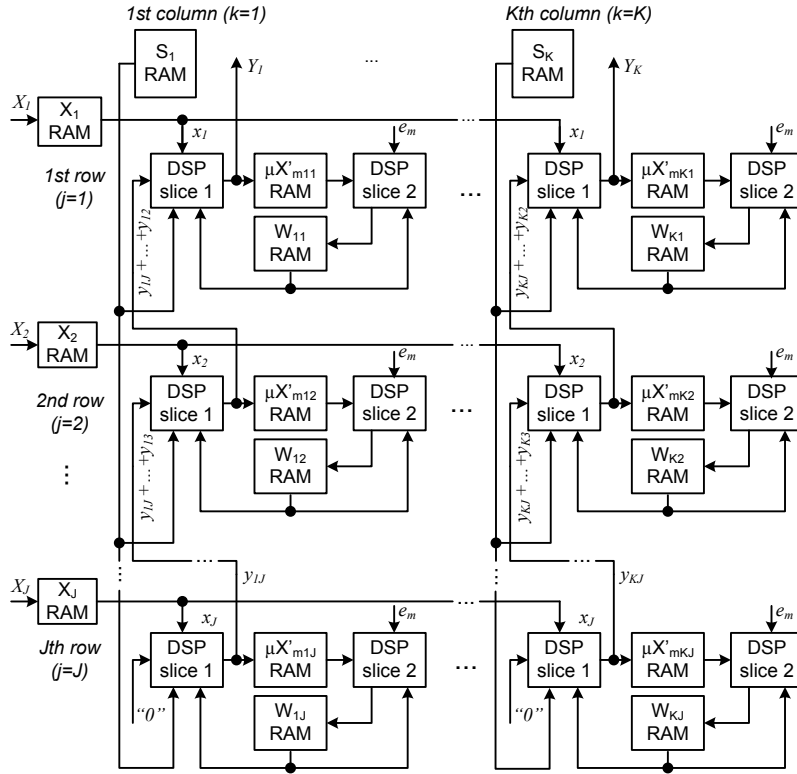


Figure 4.1.: A systolic architecture with $K \times J$ modules presented by Ushenina [Ush16]. Each modules performs the $M \times L$ MAC-operations sequentially.

multiplications with one DSP-slice, the data and coefficient bit widths are limited to fixed-point formats of 18 bit and 25 bit respectively. As the author states, the designs advantage is based on the assumption that the number of error microphones M is larger compared to the number of reference microphones J and secondary sources K and therefore also favours asymmetric ANC-systems. Another notable restriction Ushenina makes is that the static filter lengths has to be equal or smaller to the adaptive filter lengths. Unfortunately the author provides only a timing-analysis for the selected configurations and no synthesis results are given. One can speculate that the given platform for each configuration is fully utilized. As a reference, one configuration is using a *Xilinx XC7K325T-3* FPGA with $32 \times 8 \times 8$ channels and filter lengths of 1024. Ushenina gives a maximum operating for this configuration of $f_{\text{op_max}} 142 \text{ MHz}$ and states that the inequation

$$f_{\text{op}} > f_s N (M + 6) \quad (4.1)$$

must be satisfied. This results in a maximum sampling rate of $f_s \approx 3.65$ kHz for the given configuration. Another configuration using the same platform with $12 \times 10 \times 10$ channels and a filter length of 1024 achieves a maximum sampling rate of $f_s \approx 15.6$ kHz.

Shi et al. present a feedforward multiple-input and multiple-output (MIMO) FxLMS architecture with a $24 \times 24 \times 24$ dimension using filter lengths of 200 and a sampling rate of 25 kHz in [Shi+17; Shi+19]. The system is applied to a modified window for houses. The design uses floating-point arithmetic and thus requires more hardware resources per operation compared to fixed-point arithmetic. In [Shi+17], Shi et al. state to use 4 DSP-slices of the type *DSP48E1* for a single MAC-unit. For comparison, a fixed-point MAC-unit usually uses 1-2 DSP-slices depending on the operand's bit widths. The implementation of the algorithm uses a folding technique to time-multiplex the resources used for the multiply-accumulate operations. As can be seen in figure 4.2, the architecture is parallelized over J reference channels with one MAC-unit each. Each branch is sequentially performing the MAC-operations for $M + 1$ adaptive and static filters. Hence, higher filter dimensions are traded in exchange for proportionally decreased sampling rates. A configuration with $24 \times 24 \times 24$ channels is presented sampling at $f_s = 24$ kHz. The authors also make the important simplification to only allow systems where the number of secondary sources equals the number of reference sensors $K \stackrel{!}{=} J$. Based on this assumption, they reduce the JKM filtered reference signals from equation (2.13) to only JM , which is a significant reduction of processing complexity.

The architecture is applied to a sliding window with dimensions of $0.5 \text{ m} \times 1 \text{ m}$ that functions as a opening into a closed room. 24 identical ANC-units are equally placed on a security grill across the aperture. Each ANC-unit is constructed as a small tube with the reference microphone placed at the outside end of the cylinder and the secondary loudspeaker placed on the inner end. These ANC units allow to obtain a plausible reference signal at the front of each tube. This enables the use of the feedforward FxLMS algorithm in the first place.

Lorente et al. present in [Lor+14a] a GPU-based feedforward FxLMS implementation for noise attenuation in rooms. This design is hard to compare against the other architectures, as it operates in the frequency domain, which requires multiple regular and inverse fast Fourier transforms (FFTs) and differs from the other algorithms despite its name. Also their algorithm processes samples in batches or blocks between 256 and 2048 samples, because this way the GPU can be fully utilized.

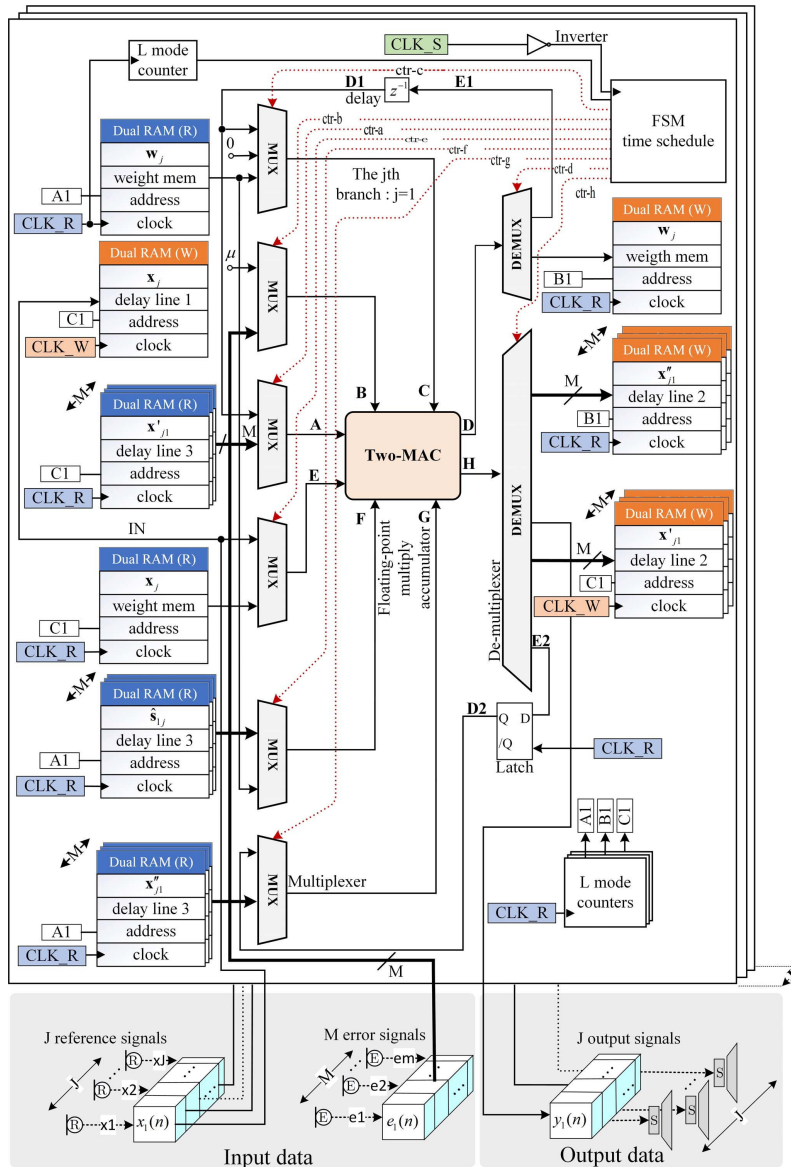


Figure 4.2.: Proposed hardware architecture of a multi-channel feedforward FxLMS by [Shi+19].

Due to architectural characteristics GPUs do not work well on a sample-per-sample processing algorithm. The authors present different configurations of which all use a single reference sensor. One configuration using $1 \times 13 \times 13$ channels and a block size of $B = 256$ samples can sample up to a maximum sampling rate of $f_s = 44$ kHz. A second configuration even uses $1 \times 36 \times 36$ channels and a block size of $B = 2048$ samples which lead to the same sampling rate. The system delay due to the block-input buffer and the processing delay is quite large with around $t = 50$ ms, which makes this approach unsuitable for delay sensitive algorithms.

The presented designs can be summarized as follows: All implement a feedforward FxLMS without step size normalization or leakage factor. The authors had applications in mind that allow to record a causal reference signal. Shi et al. even modified their physical setup to make their ANC-application suitable for the feedforward FxLMS. The goal of the authors seemed to focus on low computational complexity by omitting any expensive LMS modifications due to the inherently increased stability of the feedforward-based FxLMS algorithm. For the same reason, the designs are not optimized towards a low processing delay. The computational load is further decreased compared to the feedback FxLMS, because the feedforward-variant requires $K \times M$ fewer static FIR filters for the same configuration. All of the authors seemed to have specific applications in mind, thus the channel dimensions of the architectures have notable restrictions. Based on these optimizations each implementation presents impressive channel dimensions, although some are severely limited in terms of the maximum sampling rate or filter lengths.

4.1.2. Design Decisions

Implementation Technology

When starting to implement architectures for the FPGA platform, the first decision to be made is the preferred implementation technology. Suitable choices are HDLs like very high speed integrated circuit HDL (VHDL) or (System-)Verilog, HLS or graphical toolboxes like Xilinx System Generator [Xil20]. The following characteristics of these technologies are decisive here. HDLs like VHDL are a standardized language that is independent of the vendor or a family of FPGAs. Although the large vendors of FPGAs keep all the synthesis tools proprietary, one can generally be sure that VHDL code is portable among the synthesis tools and thus FPGA plat-

forms. Hardware implemented in a HDL is mostly described on the RTL, again see figure 3.5, which offers very good precision and optimization possibilities but at the same time requires more development time and effort. HDLs also provide the use of parameters that allow to design generic designs without changes to the underlying code.

A newer technology from the larger FPGA vendors like Xilinx and Intel (formerly Altera) is HLS. HLS synthesis tools are able to take code from languages that are normally developed to be executed on a processor as compiled software like *C* or *C++*. The HLS synthesis tools then try to interpret the intention of the programmer from algorithmic abstraction level down to RTL. This can lower development time significantly and it also allows for generic design using parameters. In [MFR20] Millón et al. compared HLS and HDLs for the development of an image processing application in a study case. The authors suggest that the development in HDLs takes more than three times as much development time (384 h vs. 121 h) compared to HLS. They also found that the resource usage and response time is lower for the HDL implementation.

The HLS synthesis tools require additional commands in the programming language, because the tool can not know, for example, if the programmer wants to have his software-loop executed in parallel or sequentially. Unfortunately this additional commands and some required libraries or other functionality are proprietary, which makes HLS-code developed for Intel FPGAs not portable to be used with Xilinx FPGAs and vice versa. Because the programming language only supports discrete number types with a fixed bit precision by default, a fixed-point library that supports arbitrary bit widths is one example for an important proprietary library that is vendor-specific.

Graphical tools like the Xilinx System Generator provide the highest layer of abstraction from the underlying hardware. They provide an easy and intuitive way to arrange proprietary Xilinx IP-cores to implement the desired architecture. The IP-cores range from simple registers and basic arithmetic operations to static FIR filters and FFT-cores. On the downside the designs can for obvious reasons not be used with FPGAs from other companies. Also there is no possibility to build generic designs that allow to easily change parameters of the algorithm. The IP-cores themselves can be tuned with the available parameters, but one can not put an arbitrary number of these cores in a design depending on a custom parameter. The high level of abstraction also results in low possibilities to optimize a design. For example the

IP-core of the FIR filter can not be modified to provide custom features, such as a constant processing delay.

Number Format

Another important design decision is the preferred number format. The IEEE-754 floating-point format [IEE85] is the most widely used format for processors or GPUs. It provides four binary subtypes with varying precision, called single, single-extended, double and double-extended. The binary number formats consist of a sign bit, a normalized fixed-point number, called the mantissa and a respective exponent. The subtypes differ in their bit widths. The exponent extends the number range of the mantissa exponentially, so for most applications in digital signal processing the number range can be considered as unlimited. In case a number overflow does occur, for example due to recursive parts in the algorithm, the IEEE-754 standard does have conditions to catch these exceptions and processors can propagate them through the algorithm. Floating-point numbers do have a limited relative number precision to represent rational numbers depending on the bit width of the mantissa. An additional effect for floating-point numbers is that the absolute number precision is dynamic with changing exponent values. One example where this can be an issue is when very large and small numbers are being added. For developers of DSP algorithms using floating-point numbers is trivial, as for most applications the above effects can be disregarded. Hardened floating-point arithmetic units in processors FPGAs or GPUs, are designed so that the incoming and outgoing number formats do not change. Since algorithms are mostly implemented using purely the single or double precision subtype, performing arithmetic operations require no additional thought about number formats. Naturally this increased comfort comes at the cost of increased complexity when floating-point compared to fixed-point arithmetic is performed. Historically, the FPGA platforms provide DSP-slices for integer arithmetic as they are more efficient to implement. Modern FPGAs including floating-point DSP-slices, like the *Xilinx 7 series* and the *Intel Arria 10*, increasingly emerge. Hettiarachchi et al. present a comparison in [HDB20] of various arithmetic test units targeting FPGA platforms that are implemented using single-precision floating-point and integer numbers. Both number formats use a bit width of 32 bit. Even though the comparison was performed on FPGA platforms that feature hardened floating-point units, the floating-point test units showed a 20 % to 27 % performance decrease in terms of timings. Additionally the fixed-point

units were able to save about 60% of DSP-slices and 10% to 20% of the other hardware resources.

In contrast, the fixed-point number format uses a plain integer number in two's complement representation and introduces an imaginary decimal point. All bits left of the decimal point contribute to the integer value and determine the available number range while all bits on the right form the fractional parts that determine the precision of a rational number. The location of the decimal point is not stored in the fixed-point number, so the format of each number has to be known to interpret the values correctly. To do this, the integer value has to be implicitly divided by a fixed scaling factor that corresponds to the decimal point location. Due to the simplicity of the number format there is no standard specifying any discrete bit widths, so usually a number format is specified for each application or each signal individually depending on the requirements. The Q-notation is used from here on as a convention to specify the fixed point format of a signal. This notation and its implications on the represented number format can be found in table 4.1. If a signed

Table 4.1.: Properties of signed und unsigned fixed-point numbers with m integer bits and n fractional bits in Q-notation.

Format	bit width	Number range	Precision
Q $m.n$	$m + n + 1$	$[-2^m, 2^m - 2^{-n}]$	2^{-n}
UQ $m.n$	$m + n$	$[0, 2^m - 2^{-n}]$	2^{-n}

or unsigned fixed-point number format uses no integer bits and only fractional bits as in (U)Q0. n , then the Q-notation allows to omit the index m and simply note the format as: (U)Q n .

As mentioned before, the difference between a fixed-point number and an integer number is only implicit, so arithmetic operations with fixed-point numbers can be performed by using the same hardware or software arithmetic units. As with integer arithmetic, to avoid overflows, the result's bit width and also the location of the decimal point can change depending on the respective operation. If arithmetic operations on fixed-point numbers with a different number of fractional bits are performed, it is essential to first align the decimal points of both numbers by adding padding bits with the value of the sign bit. The number format after the basic arithmetic operations can be found in table 4.2. Of course combining these operations result in a steadily increase of the format's bit width. So depending on the required precision and number range of the application, at some point quan-

Table 4.2.: Fixed-point formats to fit the resulting number range of the corresponding arithmetic operation.

Operation	Number format of result
$Q_{m_1.n_1} \pm Q_{m_2.n_2}$	$Q(\max[m_1, m_2] + 1).(\max[n_1, n_2])$
$UQ_{m_1.n_1} \pm UQ_{m_2.n_2}$	$UQ(\max[m_1, m_2] + 1).(\max[n_1, n_2])$
$Q_{m_1.n_1} \cdot Q_{m_2.n_2}$	$Q(m_1 + m_2 + 1).(n_1 + n_2)$
$UQ_{m_1.n_1} \cdot UQ_{m_2.n_2}$	$UQ(m_1 + m_2 + 1).(n_1 + n_2)$
$Q_{m_1.n_1} / Q_{m_2.n_2}$	$Q(m_1 + n_2 + 1).(m_2 + n_1)$
$UQ_{m_1.n_1} / UQ_{m_2.n_2}$	$UQ(m_1 + n_2).(m_2 + n_1 + 1)$
$1 / Q_{m.n}$	$Q(n + 1).m$
$1 / UQ_{m.n}$	$UQn.(m + 1)$

tization techniques are required. When an algorithm uses some form of recursion, the feedback loops of these data signal must use quantizers to resize the fixed-point number. For external hardware components like a DAC or components that transfer the data digitally, adapting the number format after the desired operations is also mandatory. If fractional bits are present, the quantizers can implement a truncation or rounding of an arbitrary length of fractional bits. The *Matlab* fixed-point library for *Simulink*-blocks [Mat] defines 7 different rounding modes and their behavior, but do not offer an implementation strategy. It includes truncation as the rounding mode *Floor*, as truncating of two's complement numbers is equivalent to "always round in direction of negative infinity". Implementing this mode requires no additional hardware, but it introduces a large negative biased rounding error. This can lead to a non-negligible cumulative output offset when being used on adaptive filters due to its recursive nature. The mode *Nearest* is the rounding mode from this list that is able round to the nearest quantized number at minimal computational complexity. It is not perfect, as it comes with a small positive bias. Though this small positive bias does only occur when the number to be rounded is tied right in the middle between next upper and lower quantization step and is then rounded in the direction of positive infinity. The rounding modes *Round* and *Convergent* introduce more logic to mitigate this constant positive bias for tied numbers, but come with an significantly increased computational cost according to [Mat]. Thus the rounding mode *Nearest* seem to provide a good trade-off between accuracy and hardware costs and should prevent measurable offsets caused by biased rounding errors. The implementation algorithm is briefly explained as follows. Let us consider a Q15 format with the numeric value 0.333251953 is supposed to be rounded to a Q7 format. The here chosen "rounding to nearest" algorithm then adds the most

significant bit (MSB) of the bits that are to be removed, in this case bit 7, to the least significant bit of the extracted format. The result in Q7 format is 0.3359375

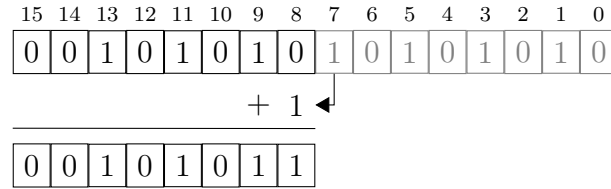


Figure 4.3.: Algorithm of rounding mode *Nearest* to convert a Q15 number to a Q7 format.

as can be seen in figure 4.3.

From table 4.2 we can see that also the number of bits in the integer region increases with most operations. Each integer bit doubles the available number range and is guarding the number from a number overflow, thus also called guard bit. In two's complement, the process of adding guard bits is also called *sign extension* as the value of these bits is the same as the sign bit value to preserve the original number value. When data with an increased number format is transferred to an DAC or other interfaces, it may be necessary to reduce the number of integer bits. Obviously removing integer bits from a fixed-point number without destroying the represented number value is only possible if the number value is within the newly reduced number range. Let us consider another example using a Q15.0 format that is supposed to be converted to a Q7.0 format. We have to check if one of the bits with index 7 to 14 contain a different value than the sign bit. If that is the case, than we can not represent the value in the new format and instead saturate the new number format to the closest representable number value, see figure 4.4. Else, the evaluated integer

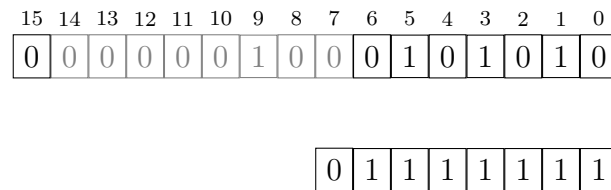


Figure 4.4.: Example for resizing a Q15.0 format to Q7.0 format. The value of bit 9 does not equal the sign bit's value, so the resulting number format is saturated to the closed possible number value.

bits can simply be truncated from the new format without altering the represented number value. When resizing the number format, it is generally necessary to reduce the fractional and integer bits to achieve the desired format. In the following implementations, this process will be referred to as *resizing*.

4.2. Single-Channel Feedback FxLMS

Static FIR Filter

The central part of all the FxLMS variations is the static FIR filter that performs the necessary convolutions. Figure 4.5 shows the implemented architecture for the one static FIR filter that does calculate the filtered reference signal $x'(n)$.

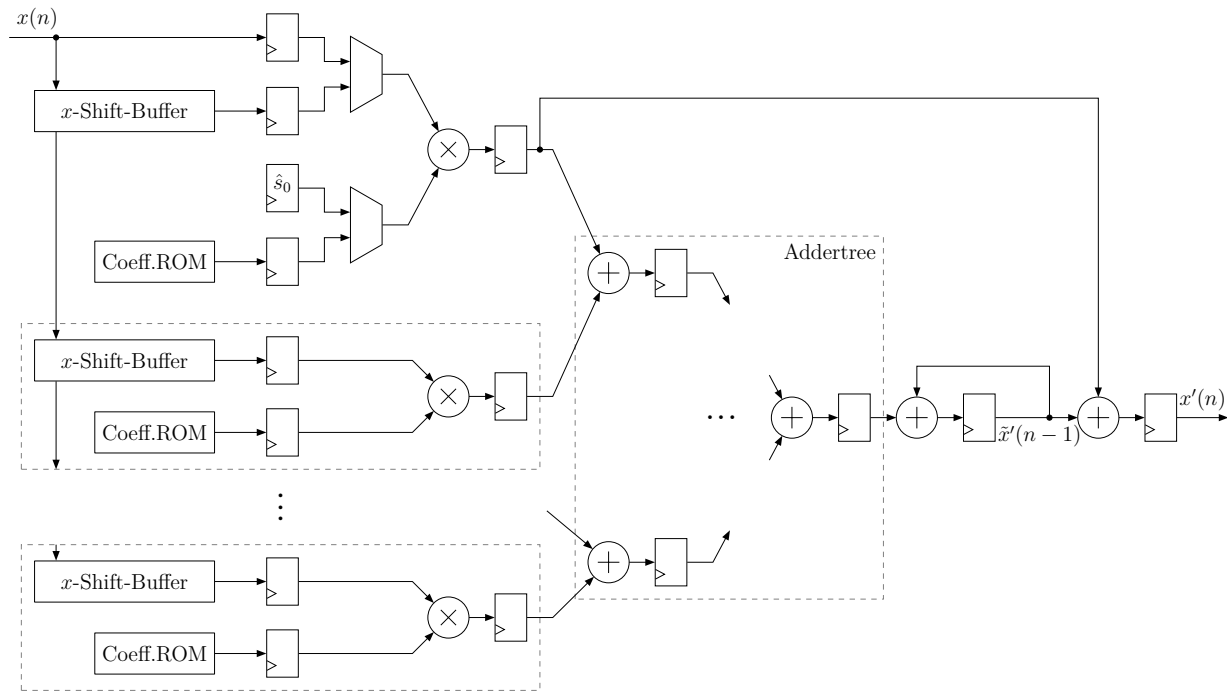


Figure 4.5.: Hardware architecture of the static FIR filter using a generic number of multiplier lanes.

The last $L_{\text{static}} - 1$ samples of the reference signal $x(n)$ are stored in shift-buffers that are based on dual-port block-RAM. The static coefficients are also stored in block-memory. To achieve a flexible architecture that can be optimized towards either a high filter order or sampling rate, the convolution in equation (2.4) is split into a generic array of multiplier lanes that are attached to an addertree. The hardware design of the static FIR filters provide generic parameters for the filter length L_{static} and the number of multiplier lanes per filter N_{static} . The generic number of multipliers is depicted with three dots in the figure 4.5. Each of the N_{static} multiplier lanes sequentially performs an equal part of the convolution while the multiplier lanes operate in parallel. The sequentially generated products of the multiplier lanes are summed up by the addertree and the succeeding recursive accumulator. To enable the short processing delay, the convolution from equation (2.4) is processed out of order. The static FIR filter does precalculate the part of the convolution

$$\tilde{x}'(n-1) = \sum_{l=1}^{L_{\text{static}}-1} \hat{s}_l x(n-l) \quad (4.2)$$

that is independent of the upcoming sample $x(n)$ one time step in advance. When this sample arrives, the filter has to perform only one multiplication and one addition to obtain

$$x'(n) = \hat{s}_0 x(n) + \tilde{x}'(n-1). \quad (4.3)$$

The principle originates from a hardware implementation proposed by Eckert et al. for a static FIR filter in [Eck+18]. The here presented architecture adopts this design and provides two additional optimizations to increase the hardware resource efficiency. Firstly, the number of used recursive accumulators is reduced to one by moving it behind the addertree. Secondly, one first multiplier lane is saved by having it process a part of precalculation of the convolution and equation (4.3) at the cost of additional multiplexers. This is possible, because both operations execute at different points in time.

A restriction of the currently implemented design is that the number of the filter coefficients in BRAM $L_{\text{static}} - 1$ and the number of multiplier lanes per filter N_{static} have to be a power of two. The number of inputs of the symmetrical addertree is a power of two and is one cause for the requirement of N_{static} . The restriction regarding the filter length ensures that the number multiplications can be equally distributed among the multiplier lanes and thus severely reduces the complexity of the corresponding control logic. Further developments avoiding these restrictions are possible, but the granularity of $L_{\text{static}} - 1$ and N_{static} as a power of two was suffi-

cient for the conducted experiments and the restrictions avoid additional hardware resources for managing any asymmetry. For the static FIR-filters one workaround is possible to achieve arbitrary filter lengths. By setting the value of undesired filter coefficients to zero, these filter taps do not contribute to the output value and the synthesis tool should be able to optimize some of the corresponding hardware resources away.

Adaptive FIR Filter

The adaptive FIR filter uses the same design principles as the static FIR filter. To incorporate the adaptive coefficients, the ROM-tiles for the static coefficients are each replaced by RAM-tiles and a component to perform the coefficient update, as can be seen in figure 4.6. An additional array of shift buffers is required to provide the filtered reference signal $x'(n)$ from equation (2.4).

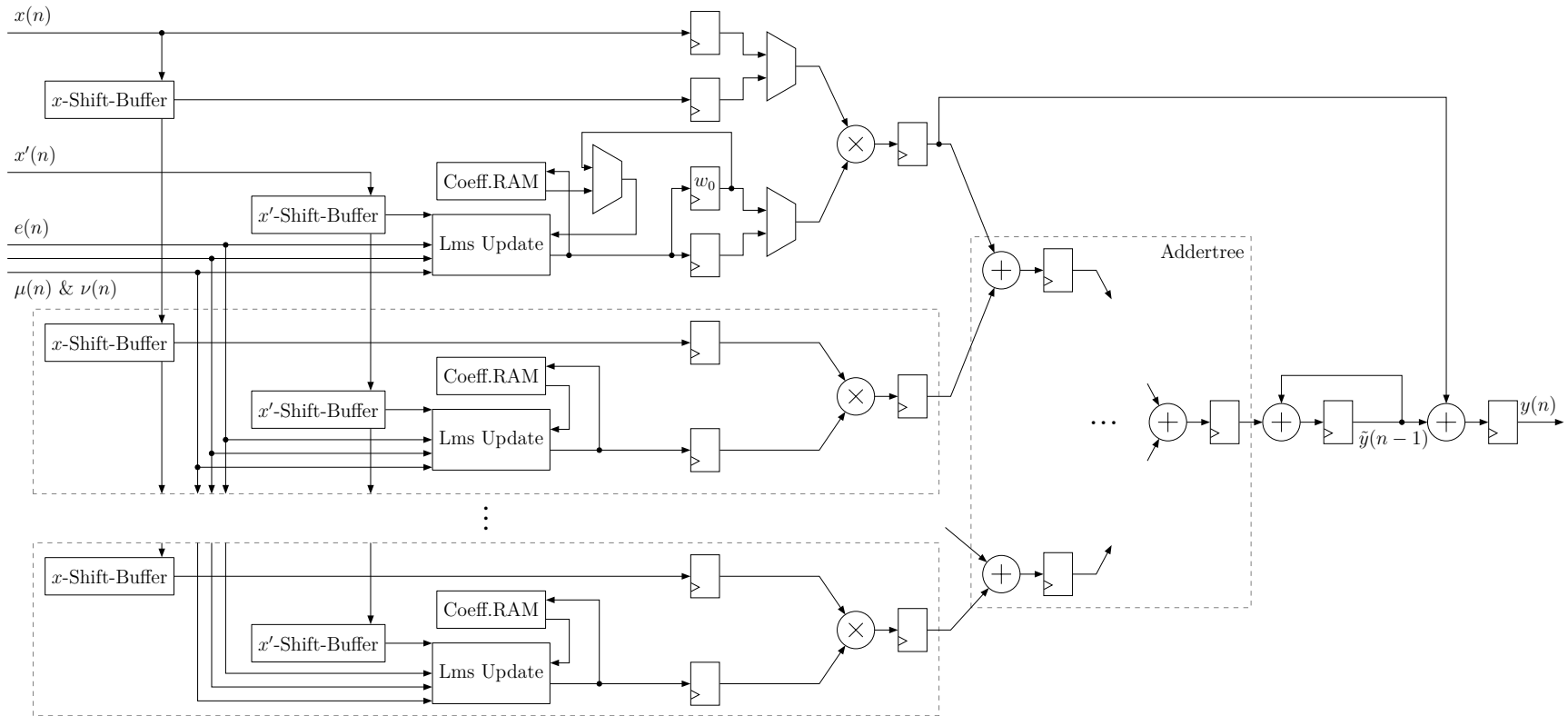


Figure 4.6.: Hardware architecture of the adaptive FIR filter using a generic number of multiplier lanes.

Like the static FIR filter, the convolution from equation (2.1) is processed out of order. First the output is calculated by

$$y(n) = w_0(n) x(n) + \tilde{y}(n - 1). \quad (4.4)$$

Before the precalculation of the convolution can be done in the known manner, the adaptive coefficients have to be updated by performing

$$w_l(n + 1) = w_l(n) - x'(n - l) e(n), \quad l = 1 \dots L_{\text{adapt}} - 1 \quad (4.5)$$

and subsequently updating the first coefficient

$$w_0(n + 1) = w_0(n) - x'(n) e(n) \quad (4.6)$$

by using first LMS update component. This component is time-shared using multiplexers like the first multiplier lane in the convolution area. Calculating the precalculated part of the convolution

$$\tilde{y}(n) = \sum_{l=1}^{L_{\text{static}}-1} w_l(n + 1) x(n + 1 - l) \quad (4.7)$$

and the update of the coefficients does not happen sequentially, but rather work in parallel with a minimal offset. The LMS update component produces one updated coefficient per clock cycle, which are fed into the multipliers just in time. To make this possible, the LMS update component is constructed as a five-stage pipeline, see figure 4.7.

At the same time the updated coefficients are fed into multipliers, they are also

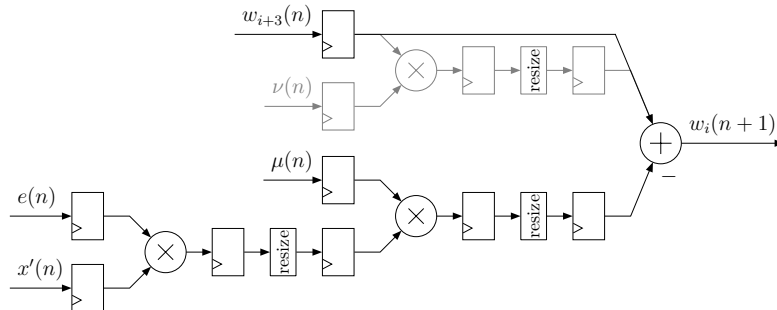


Figure 4.7.: The hardware architecture of the LMS update component. The hardware resources for the implementation of the leakage factor $\nu(n)$ are marked in gray, because they are optional, see section 4.3.

being stored back in BRAM. This is possible, as modern FPGAs have dual-port BRAM-tiles included. This way one port sequentially transfers the old coefficients into the LMS update component while the second port writes back the updated coefficients.

Top-level architecture

The top-level design uses two instances of the static FIR filter and one adaptive FIR filter. Unlike the figures 4.5 and 4.6 imply, figure 4.8 shows that the filters share memory-resources when implemented together. This way any redundant storage of

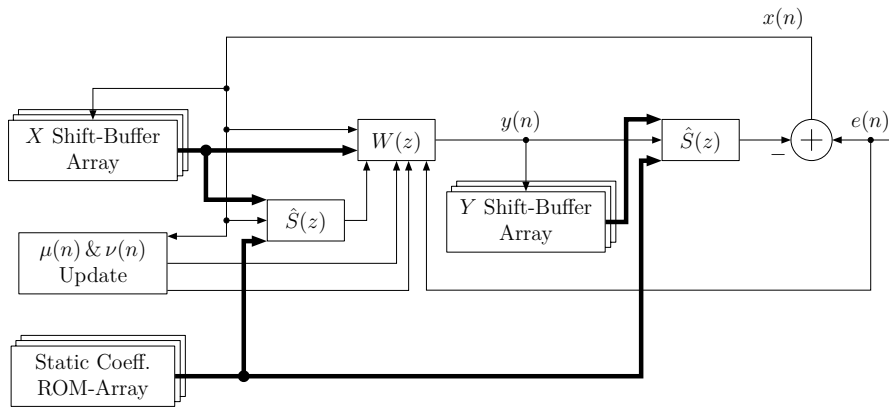


Figure 4.8.: The datapath of the top-level feedback FxLMS implementation. The components for the static and adaptive FIR filters are stylized as block diagrams. The adaptive FIR filter block includes the LMS algorithm, as it is also tightly integrated in the hardware implementation.

data samples or coefficients in block-memory can be avoided. The adaptive FIR filter and the static FIR filter that is responsible for the filtered reference signal $x'(n)$ share the sample-buffer array for the reference signal $x(n)$. Both static FIR filters share the ROM array for the static coefficients. Sharing those resources means that all three filters have to perform the convolution in sync, even though their results depend on each other. This is possible due to the fact that the convolution operations are executed out of order and only the precalculation of the three convolutions requires the block-memory. With the precalculated part of the convolutions ready, each filter can successively calculate its updated result and then perform the precalculated convolution part for the upcoming sample together. Besides saving block-memory and enabling the short processing delay, this method also reduces the complexity of the control path, as the same set of control signals can be used for all three filters.

Sharing the memory-resources introduces another requirement for the architecture. The sample length of the block-memory per multiplier-lane has to be the same for each filter

$$\frac{L_{\text{static}} - 1}{N_{\text{static}}} \stackrel{!}{=} \frac{L_{\text{adapt}} - 1}{N_{\text{adapt}}}. \quad (4.8)$$

Note that the adaptive and static FIR filters still can have different filter orders by using a different amount of multiplier lanes. In that case block-memory for the higher filter order will be instantiated and only the common subset of the memory will be shared.

Also note that for the figures that show the implemented hardware architectures only the datapath is shown. The control path that is omitted for clarity reasons. Instead the Gantt chart in figure 4.9 shows the order of execution that the control path follows.

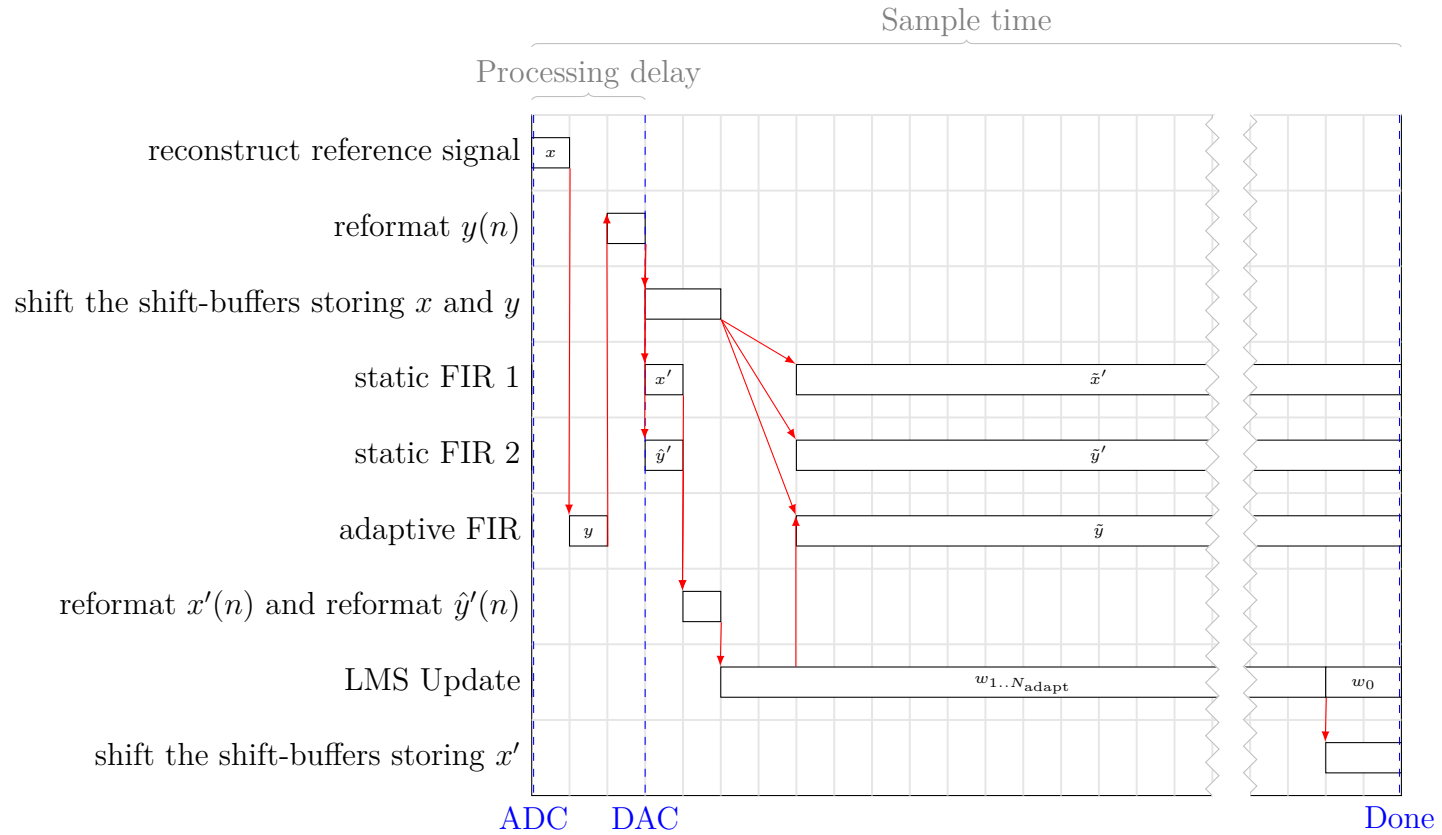


Figure 4.9.: Control flow diagram showing the dependencies and order of how operations of the FxLMS SISO implementation for a single sample cycle. The cycle starts when the ADC signals that a new sample is available. When the output signal is calculated and resized, the DAC is triggered. After the cycle is finished, the architecture is ready to receive a new sample from the ADC and idles until then.

Note that figure 4.9 is not clock cycle accurate, as it does not show clock delays between the different components. The figure shows that the reference signal, the output from the adaptive FIR filter $y(n)$ and its resized value is calculated first after a new sample arrives from the ADC. These operations take a constant of 4 clock cycles and represent the processing delay of this architecture, as the DAC is immediately triggered at this point. Another 4 clock cycles are necessary for the shift buffers that store the reference signal $x(n)$ and the output $y(n)$ to consume their new sample, for the static FIR filters to calculate their output and to resize the number format and for the LMS update component to update the first adaptive coefficient $w_1(n)$ in BRAM. The remaining clock cycles are used to precalculate the part of the convolutions and to update the remaining adaptive coefficients in parallel. Since the update component finishes some clock cycles in advance, it has time to update the coefficient $w_0(n+1)$ stored in a register at the end. The shift buffer for the filtered reference signal is triggered after the coefficients in BRAM are updated, because the LMS update component requires the n -th time step of $x'(n)$ for the coefficients $w_l(n+1)$.

It is essential to know the maximum sampling rate f_s for an arbitrary configuration of the given architecture. If the inequation

$$\frac{f_{clk}}{f_s} \geq \frac{L-1}{N} + \log_2(\max[N_{static}, N_{adapt}]) + 8 \quad (4.9)$$

is satisfied, the architecture returns to an idle state before or just in time to process the next sample. The right side shows the clock cycles consumed by the multiplier lanes, the addertree and the 8 constant cycles that were explained before. The left side represents the desired sampling rate f_s and the clock frequency f_{clk} that the FPGA is able to execute the architecture at. When isolating f_s in this equation, the maximum sampling frequency for a given configuration and clock frequency f_{clk} of the used FPGA platform can be found. [Kle+19]

Fixed-Point Number Formats

The bit-precision of the given architecture can be set via parameters in VHDL for the bit widths of the incoming and outgoing sample size, the static and adaptive coefficients, the step size, the leakage factor and the power estimation of the reference signal. The number format of all other internal signals are derived from these parameters accordingly to the respective operations that are being performed. The

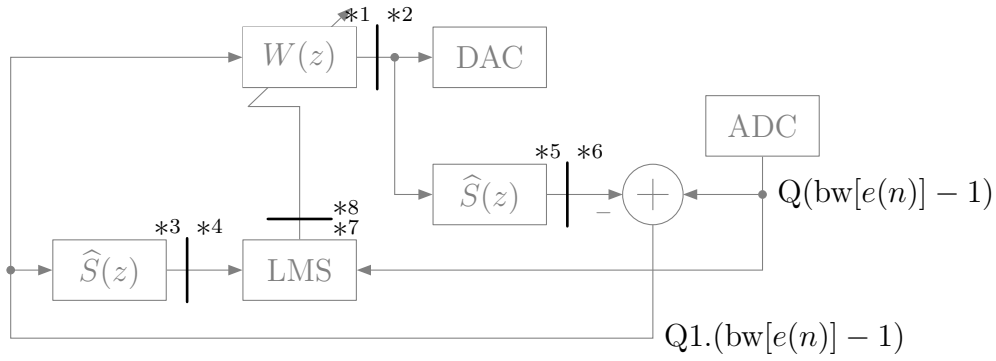
bit-precision or bit width of the incoming samples $\text{bw}[e(n)]$ and the outgoing samples $\text{bw}[y(n)]$ are commonly dependant on the precision of the used ADC and DAC components. The bit width for the adaptive and static coefficients is commonly doubled and Kuo et al. also emphasize that "when updating the weights [...], it is necessary to provide ample precision in the accumulator." in [KM95, p. 334]. As mentioned before, a regular resize of the number format is required to avoid growing unreasonable large number formats that increase resource utilization drastically. Hence, after the convolution results of each filter and the LMS update component a resizing takes place. The adaptive coefficients are the only data values in the FxLMS algorithm that can grow to infinity if the system is in permanent operation. Hence, the addition that updates the adaptive coefficients includes additional saturation logic to ensure that the adaptive coefficients can not overflow. Figure 4.10 gives an overview over the used number formats for this architecture. The number formats are consistent with table 4.2, except that only an additional $\log_2(L - 1)$, instead of $L - 1$ guard bits were added to accumulate the products of the multiplications during the convolution of each filter. This is due to the fact that data n guard bits are sufficient to accumulate 2^n values in contrast to additions with arbitrary operands.

4.3. LMS Modifications

By providing two additional boolean parameters in the VHDL implementation, the modifications of the LMS algorithm discussed in section 2.1.1 can each be optionally included to the hardware architecture before synthesis.

The normalization of the step size $\mu(n)$ and the update of the leakage factor $\nu(n)$ have to be performed only once for each sample cycle, so a pipelined architecture is not worthwhile. To keep the hardware resource utilization as low as possible, an iterative approach is chosen here to update both signals.

The following text passage is mainly an excerpt of [Tim+20]. For parameters like α , β , γ , \hat{P}_{\min} or the initial value for the power estimation $\hat{P}_x(0)$ that are constant at runtime, the values can be also set as generic parameters in VHDL, which are then evaluated at synthesis time and implemented as constants. Except for equation (2.6), the hardware implementation of the normalization of the step size and the update of the leakage factor is straightforward. Equation (2.6) requires a division by a non-constant, which is the most complex of the basic arithmetic operations. While



- 1: $Q(\log_2(L_{\text{adapt}} - 1) + 2) \cdot (\text{bw}[w(n)] + \text{bw}[e(n)] - 2)$
- 2: $Q(\text{bw}[e(n)] - 1)$
- 3: $Q(\log_2(L_{\text{static}} - 1) + 2) \cdot (\text{bw}[w(n)] + \text{bw}[e(n)] - 2)$
- 4: $Q1 \cdot (\text{bw}[e(n)] - 1)$
- 5: $Q(\log_2(L_{\text{static}} - 1) + 1) \cdot (\text{bw}[w(n)] + \text{bw}[e(n)] - 2)$
- 6: $Q(\text{bw}[e(n)] - 1)$
- 7: $Q3 \cdot (\text{bw}[\mu(n)] + 2\text{bw}[e(n)] - 2)$
- 8: $Q(\text{bw}[w(n)] - 1)$

Figure 4.10.: Used number formats for the feedback FxLMS implementation. The resizing of fixed-point formats is marked with a black bar. The footnotes refer to the original and resized number formats that are noted below.

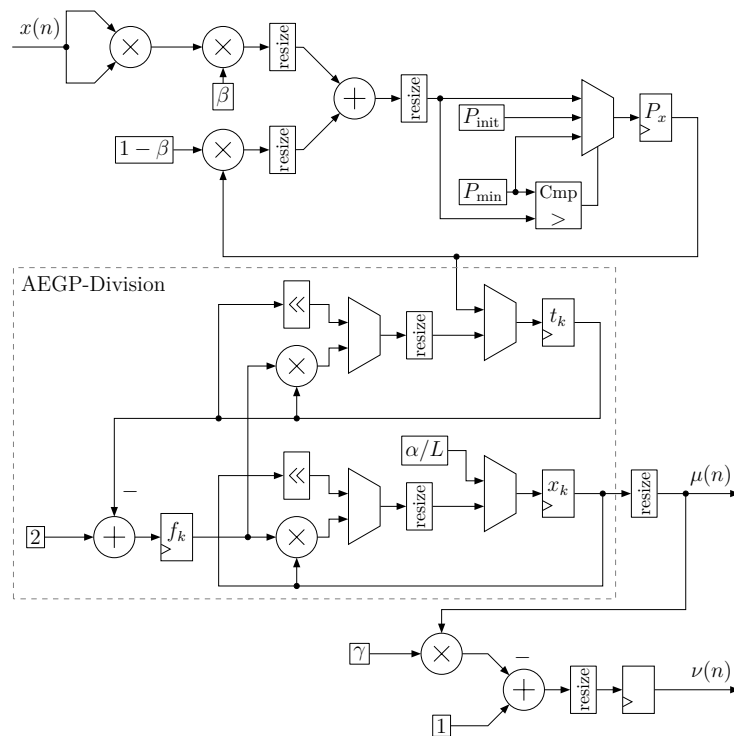


Figure 4.11.: The datapath for the normalization of the step size and the update of the leakage factor. Run-time constants are depicted with a small box around them. The components that implement the AEGP algorithm are located in the area outlined with dashes.

the operations *addition*, *subtraction* and *multiplication* are implemented in VHDL, the language's standard does not offer an implementation for a division.

Implementation of the Division Operation

The optimal algorithm depends on the scenario. The most intuitive method is what Meyer-Baese et al. in [Mey14] call the "pencil-and-paper" method, which is how students learn to divide decimal numbers in school. It can also be applied to binary numbers. The main disadvantage is that two steps are required to obtain one additional bit of the quotient per iteration, which makes this algorithm converge linearly. The fastest solution is to store all possible reciprocal values of the denominator in a table and multiply with the corresponding reciprocal value. On modern FPGAs this is only practical if bit widths up to 13 bit are used, as the table quickly becomes too memory-intensive. Since block-memory is a valuable resource for the proposed architectures, this approach is not feasible.

The Anderson-Earle-Goldschmidt-Powers (AEGP) algorithm is an iterative, numeric algorithm that does not require extra logic to handle signed numbers. The quotient's bit precision with this algorithm increases quadratically, which makes this algorithm converge fast, especially for higher bit widths. Its data path requires two multiplications, one subtraction, shift operations and resizing operations of the fixed point formats. Thus only a low to moderate amount of hardware resources and no tables in memory are required. Each iteration requires two clock cycles, because the multiplications can be computed in parallel. The quadratic convergence makes this algorithm much faster compared to the "pencil-and-paper" method in the given application, because the power estimate from equation (2.6) requires a rather high precision. Thus this algorithm was chosen to normalize the step size. The AEGP algorithm as described by Meyer-Baese et al. [Mey14] works as follows:

1. Normalize numerator N and denominator D such that D is close to 1.
2. Initialize $x_0 = N$ and $t_0 = D$.
3. Repeat the following set of instructions k -times until x_k shows the desired precision:

$$f_k = 2 - t_k \quad (4.10)$$

$$x_{k+1} = x_k \cdot f_k \quad (4.11)$$

$$t_{k+1} = t_k \cdot f_k \quad (4.12)$$

For this implementation one additional step was included as a first step: If the denominator is zero, then end the algorithm and return the maximum value for quotient that the associated number format allows. The numerator here is a constant $N = \alpha/L$ and can be calculated before synthesis. The denominator is $D = \max[\hat{P}_x(n), \hat{P}_{\min}]$. The implementation of the algorithm does the initialization of x_0 and t_0 first and then uses bit-shift components to normalize the numerator and denominator. The iterative normalization of the numerator and denominator uses incremental steps until the denominator is close enough to 1 or the number range of the numerator is depleted. Next the control logic does execute the loop until the number of iterations K_{AEGP} set in a generic parameter is reached. The result of x_k is then resized to become $\mu(n)$.

Leakage Factor

When the updated step size is ready, the control logic performs the update of the leakage factor $\nu(n)$ and subsequently begins the update of the adaptive coefficients. Enabling the leakage factor does alter the datapath of the LMS update component to include the part marked in gray in figure 4.7. This means that the control logic has to start iterating through the dual-port BRAM with the reading port two clock cycles further in advance, to accommodate the enlarged delay in the datapath.

Impact of LMS Modifications on the Sample Time

The update of $\mu(n)$ and $\nu(n)$ lies in the critical path of the algorithm, as the update of update of the adaptive coefficients can not start until $\mu(n)$ and $\nu(n)$ are ready. The normalization of the step size can start when the reference signal is ready and runs in parallel with the processing of the output signals $x'(n)$ and $\hat{y}'(n)$ of the static FIR filters and the shift-buffers for $x(n)$ and $y(n)$, see figure 4.9 for reference. The total number of clock cycles the component takes to update $\mu(n)$ and $\nu(n)$ is dependant on a variety of parameters. Firstly, the update of $\hat{P}_x(n)$ according to equation (2.7) takes a constant 4 clock cycles. Secondly, the normalization process of the denominator for the AEGP algorithm is active. This process iteratively bit-

shifts the nominator and denominator to the left, if the denominator has leading zeroes. The algorithm distinguishes if the denominator has 8, 4, 2 or 1 leading zero and performs corresponding the bit-shift accordingly. Hence the number of shift iterations is not constant and depends on the number value of $\hat{P}_x(n)$ at run-time. For the goal of this section we should consider the worst case with the maximum number of shift iterations possible. Assuming the minimal value for the denominator, $\left\lfloor \frac{\text{bw}[\hat{P}_x(n)]}{8} \right\rfloor + \left\lfloor \frac{\text{bw}[\hat{P}_x(n)] \bmod 8}{4} \right\rfloor + \left\lfloor \frac{\text{bw}[\hat{P}_x(n)] \bmod 4}{2} \right\rfloor + \text{bw}[\hat{P}_x(n)] \bmod 2$ iterations are possible with a denominator bit width of $\text{bw}[\hat{P}_x(n)]$. The delay from the third step of the AEGP algorithm also depends on the number of iterations. Each iteration adds 3 clock cycles to the total delay of the component: One clock cycle for equation (4.10), one for equation (4.11) and (4.12) and one clock cycle to resize the number format. The number of iterations of the AEGP algorithm can be also set manually for the implementation, but is by default set to $\lceil \log_2(\text{bw}[\hat{P}_x(n)]) \rceil$ with regard to the quadratic convergence of the quotient. Considering the delay of the three processes of the AEGP implementation and the mandatory FxLMS components, the total delay for the worst-case run-time scenario adds up to

$$\begin{aligned}
\frac{f_{clk}}{f_s} &\geq \frac{L-1}{N} + \log_2(\max[N_{\text{static}}, N_{\text{adapt}}]) + 5 \\
&+ \left\lfloor \frac{\text{bw}[\hat{P}_x(n)]}{8} \right\rfloor + \left\lfloor \frac{\text{bw}[\hat{P}_x(n)] \bmod 8}{4} \right\rfloor \\
&+ \left\lfloor \frac{\text{bw}[\hat{P}_x(n)] \bmod 4}{2} \right\rfloor + \text{bw}[\hat{P}_x(n)] \bmod 2 \\
&+ 3 \lceil \log_2(\text{bw}[\hat{P}_x(n)]) \rceil
\end{aligned} \tag{4.13}$$

clock cycles.

4.4. Multi-Channel Feedback FxLMS

The goal of the feedback FxLMS implementation using multiple channels is to feature a similar low processing latency as the single-channel variant. However, the design has a stronger focus on the optimization of DSP-slices and block-memory, because of the steep scaling when the number of channels is increased at the cost of some additional delay. The optimizations are based on the assumptions that the values of K and M are small compared to the filter lengths L_{adapt} and L_{static} .

The convolutions are executed out of order, as before with the single-channel implementation to allow the same optimizations minimizing the block-memory usage and reducing the complexity of the control logic. Moreover some components of the single-channel implementation can be reused and instantiated in the correct amount to fit this multi-channel implementation. This is the case for the $KM(M+1)$ static filters from section 4.2. Also the component that updates the step size $\mu(n)$ and the leakage factor $\nu(n)$ can be reused and is required M times for each reference signal that is reconstructed from the error sources. The KMN_{static} BRAM-tiles for the static coefficients use the same architecture as well. So can the VHDL component for the $M\max[N_{\text{static}}, N_{\text{adapt}}]$ BRAM-tiles for the reference signal and KN_{adapt} BRAM-tiles for the output signal.

The KM adaptive filters require a modified architecture. The multi-channel variant of the adaptive filter switches to a two dimensional array of x' -shift-buffers, which increases the computational complexity for the coefficient update greatly, see figure 4.12.

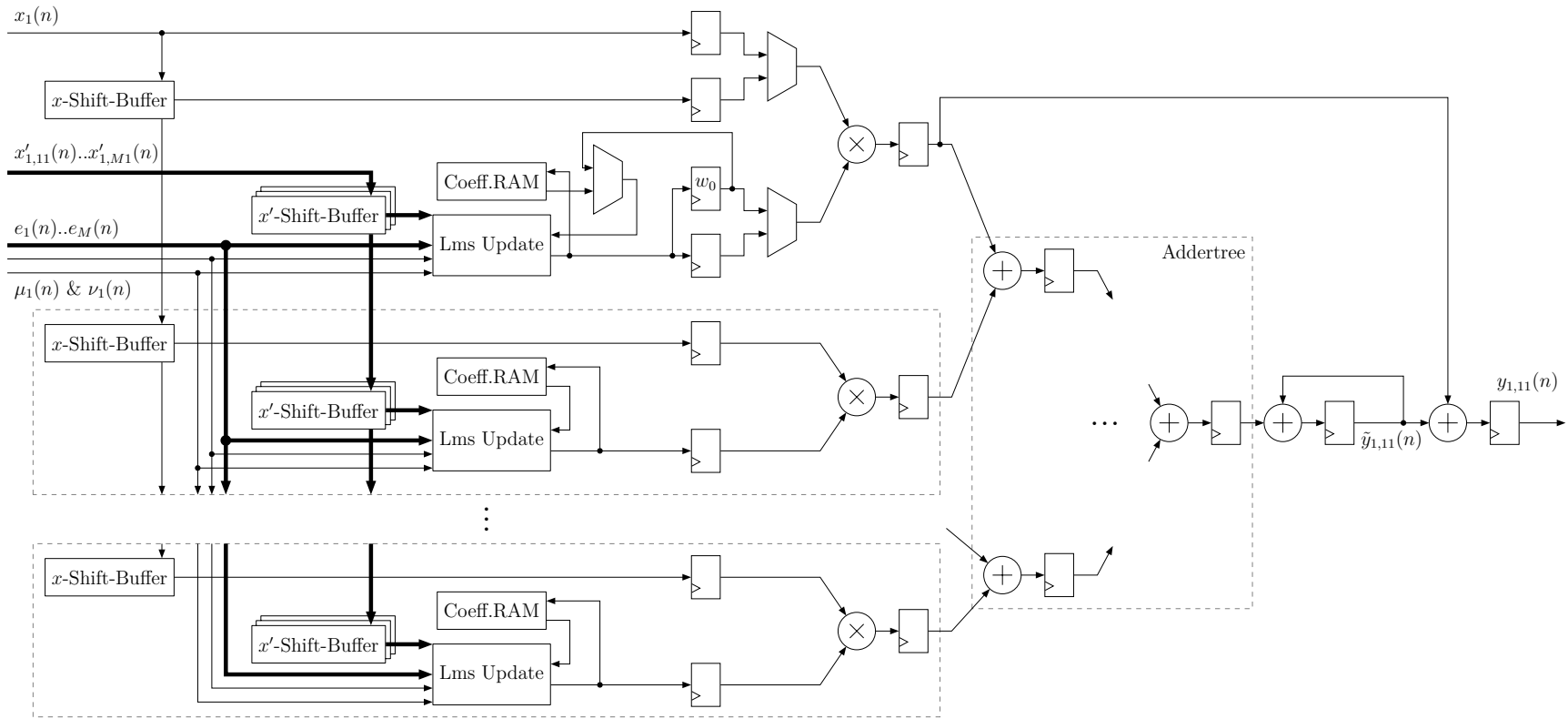


Figure 4.12.: Hardware architecture of the multi-channel adaptive FIR filter using a generic number of multiplier lanes.

To keep the pipelined architecture of the coefficient update and the convolution, the LMS update component for the MIMO variant was modified to still produce one updated coefficient per clock cycle. For this task the LMS update component performs the necessary M multiplications per clock cycle of $e_m(n)x'_{m,mk}(n)$ in parallel.

Figure 4.13 shows the corresponding architecture, which looks similar to the single-channel variant. Upon a closer look, the single multiplication is replaced by an

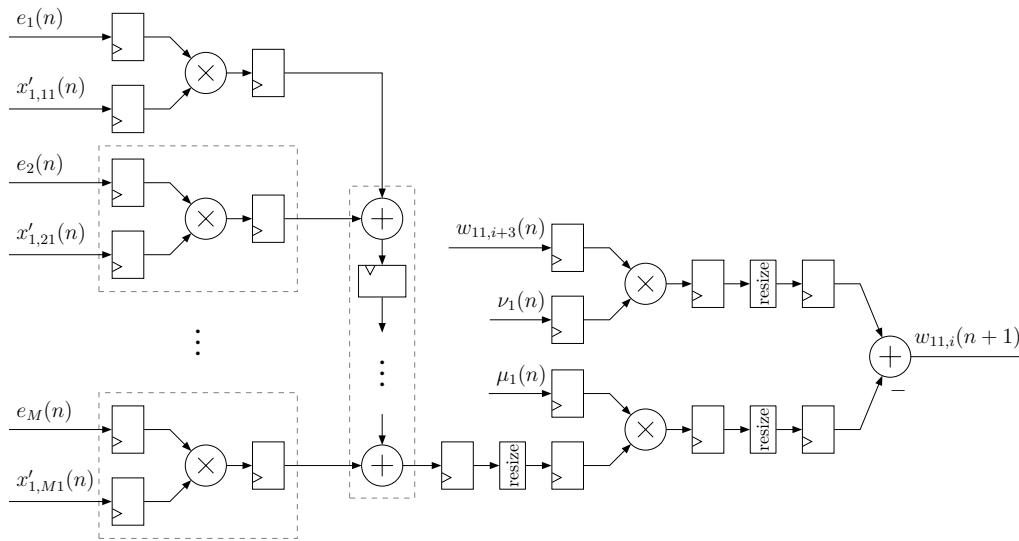


Figure 4.13.: The datapath of the LMS update component for the multi-channel FxLMS architecture.

array of parallel multiplications. To accumulate the products an adder-chain is used instead of the familiar addertree. The reason for this is that in section 5.2.2 it was found that the addertree structure does consume valuable DSP-slices. The update chain causes a clock delay of M instead of the addertree's $\log_2 M$ clock delay. However, the additional delay is rather minor, since M will range between 2 and 15 for most applications.

The following section is in parts an excerpt of [Kle+21a].

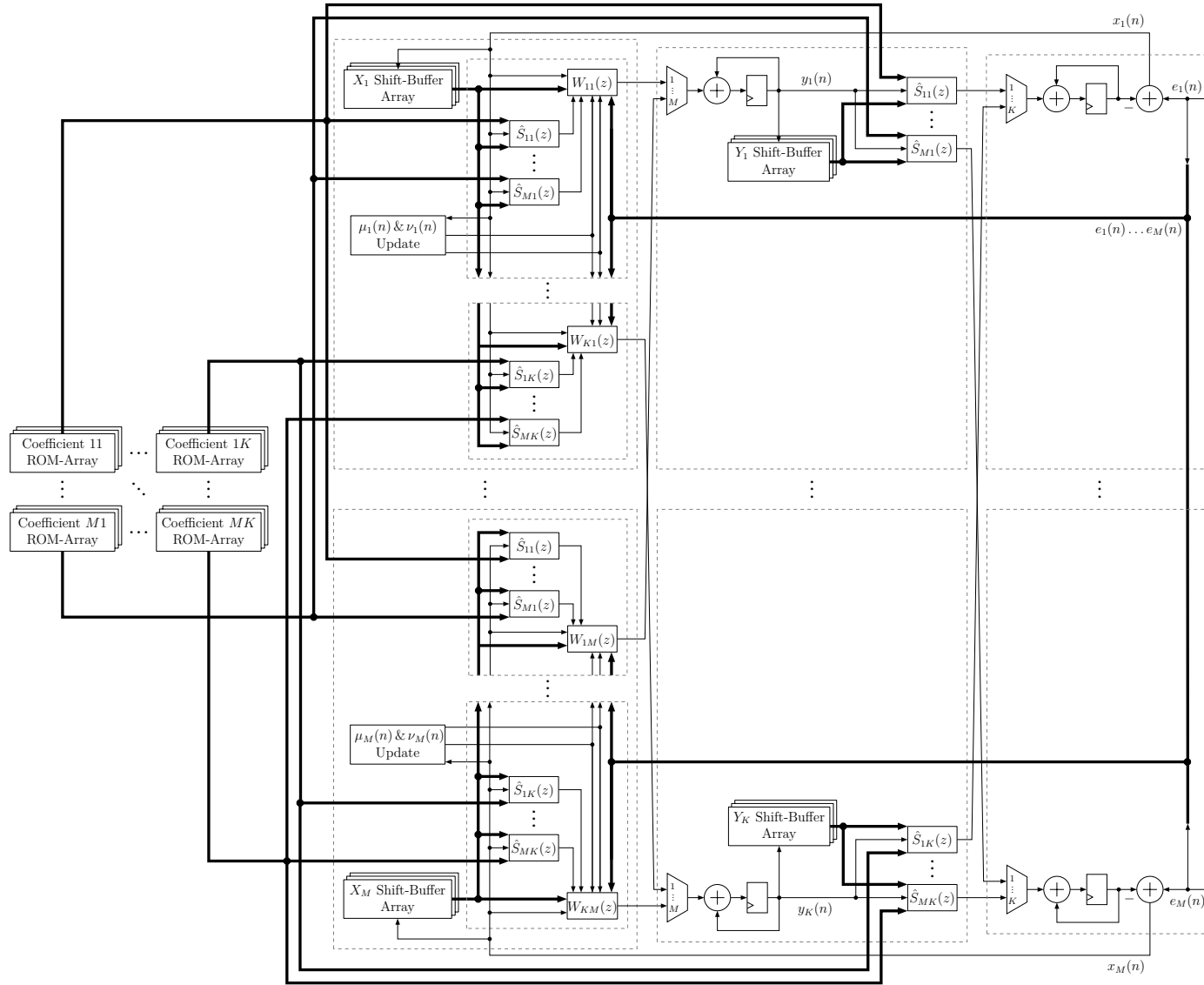


Figure 4.14.: The datapath of the top-level architecture for the multi-channel feedback FxLMS. The components for the static and adaptive FIR filters are stylized as block diagrams. The adaptive FIR filter block includes the LMS algorithm, as it is also tightly integrated in the hardware implementation.

The top-level datapath for the feedback FxLMS using multiple channels for the error sensors and secondary sources is shown in figure 4.14 and gives an overview over the top-level datapath. The figure shows that the BRAM-tiles for the static coefficients are arranged in a 3-dimensional ($M \times K \times N_{\text{static}}$) matrix. This way all the coefficients have to be stored only once, while all the static FIR filters can be executed in sync. The same is true for the y -shift-buffer outputs that are shared between the static FIR filters and the x -shift-buffers that are shared between the adaptive and static FIR filters. Otherwise the implementation functions in the same manner as explained in figure 4.9 with a minor exception. The output data of the adaptive filters and the static filters used for equation (2.11) each have to be accumulated to obtain $y_k(n)$ and $\hat{y}'_m(n)$. To save resources this accumulation is done sequentially, since the delay for sequentially accumulating M and K numbers respectively is minor compared to the total delay. The delay of the adaptive filters does not only contribute to the total delay but adds also to the processing delay. Thus the processing delay is not independent from the filter parameters anymore as with the single-channel variant, but is still independent of any filter lengths. The processing delay for the multi-channel implementation is $M + 5$. The maximum sampling rate can be found by using the inequation

$$\frac{f_{clk}}{f_s} \geq \frac{L-1}{N} + \log_2(\max[N_{\text{static}}, N_{\text{adapt}}]) + 2M + K + 16. \quad (4.14)$$

When step size normalization and the leakage factor is used the corresponding delays from section 4.3 must be added.

4.5. System Identification via LMS

The LMS-based method for the system identification of the secondary path uses the same implementation of the adaptive FIR-filter, as the FxLMS implementation. One difference is the addition of the step size instead of an subtraction, as mentioned in section 2.2.1. As the adaptive filter converges towards the unknown system, the hardware architecture outputs the remaining error to give feedback on the adaption progress. When a transfer of the currently identified impulse response is triggered, it is important that the adaptive coefficients are not updated further during the extraction of the data. In order to achieve this, there are two possibilities. Firstly, the adaptive filter stops to process incoming data samples until the transfer is complete or secondly, the content of the block-memory containing the adaptive coefficients is

copied to a second memory location in between two incoming samples. The content of the block-memory containing the copied data can then be accessed from outside at any time. Here, the secondary approach is implemented, as the block-memory is not a critical resource for this task and it enables multiple, fast data transfers, because the identification system stays adapted to the unknown system.

4.6. System Identification using Exponential Sine Sweeps

The following section is in parts an excerpt of [Kle+21b]. As before, this implementation makes heavy use of generic parameters to allow the hardware design to be optimized for different applications. Among other parameters the sampling rate, the sweep duration, the starting and ending frequency of the sweep, the desired length of the system to be identified, the amplitude fade length, the number of coordinate rotation digital computer (CORDIC)-iterations and the bit widths of the signals $\Omega(n)$, z , C , A , $x(n)$ and $y(n)$ can be set without code modifications via VHDL generics. Compared to the LMS-based method, the main motivation to implement this technique is the improved quality of the obtained impulse response and the suitability of this approach for stand-alone operation using only the FPGA platform. The algorithm finishes at a fixed moment, so no monitoring of the residual error is required to dynamically end the system identification phase, as would be necessary with the LMS-based technique. Moreover no external or even internal band-limited signal noise generation with variable parameters for the frequency range are required. Both characteristics make the approach based on sine sweeps much easier to execute stand-alone on the FPGA platform.

The datapath of the top-level architecture is shown in figure 4.15 and can be divided into three main components which perform the sine sweep generation, the convolution and the amplitude fade of the sine sweep signal. To identify a system, the control logic first triggers the periodic generation of the sine sweep samples from f_{start} to f_{end} with the given sampling rate. That signal is processed by the component that performs the amplitude fade and sent to the DAC. Synchronously to the DAC, the ADC is triggered to sample the system response. After L_x samples, the control logic stops the excitation but records L_h more samples. L_h is the time for the system to settle. After the completion of this sweep, the control logic sets the

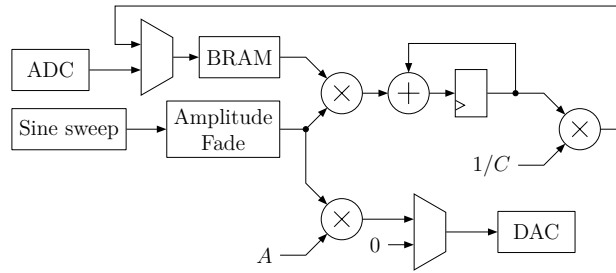


Figure 4.15.: The top-level datapath of the system identification component using exponential sine sweeps.

state of the sine sweep generator to reverse the sweep with the maximum sampling rate possible. For the convolution of the reversed sweep with the system response, the BRAM is traversed in the opposite direction by the control logic in sync with the sine sweep generator. A MAC-unit sequentially calculates the partial results of the convolution. According to equation (2.27), each result of the convolution has to be divided with the scalar C . This scalar is constant during runtime, so the division is implemented as a multiplication with the reciprocal value. When the first partial result of the convolution is accumulated in the MAC and divided by C , the control logic stores the result in the BRAM and thus overwrites the first sample of the system response. As the reversed system response is shifted to the right for each partial result of the convolution, more samples stop to overlap with the sine sweep and thus do not contribute to the result. When the partial result of the convolution reaches the desired length of the system to be identified, the architecture signals its readiness and returns to the idle state. The second port from the BRAM can be used by external entities to read the result.

4.6.1. Sine Sweep Generation

The hardware implementation of the Gold and Rader oscillator requires the use of trigonometric functions. Among hardware-efficient algorithms for trigonometric functions, the CORDIC algorithm is a simple and fast solution. A good survey about the CORDIC equations and the processor architecture with a focus on FPGA implementations can be found in [And98]. To calculate the sine and cosine function,

the CORDIC processor in rotational mode uses the the coordinate transform

$$x_R = A_R(x_0 \cos z_0 - y_0 \sin z_0) \quad (4.15)$$

$$y_R = A_R(y_0 \cos z_0 + x_0 \sin z_0) \quad (4.16)$$

$$z_R \approx 0 \quad (4.17)$$

$$A_R = 1 / \prod_{r=1}^R \sqrt{1 + 2^{-2r}} \quad (4.18)$$

to rotate the coordinate (x_0, y_0) in R iterative steps, where z_0 is the desired initial rotation angle. The main idea of the CORDIC algorithm that makes it efficient for hardware implementations is to recreate a rotation around an arbitrary angle z_0 by using multiple iterations around angles $\theta_r = \tan^{-1}(2^{-r})$, where r is the index of the iteration step. These R angles can be calculated before synthesis and stored in a table. Under this condition, the coordinate transform can be simplified to

$$x_{r+1} = K_r(x_r - y_r d_r 2^{-r}) \quad (4.19)$$

$$y_{r+1} = K_r(y_r + x_r d_r 2^{-r}), \quad (4.20)$$

where

$$K_r = \cos(\tan^{-1}(2^{-r})) = 1/\sqrt{1 + 2^{-2r}} \quad (4.21)$$

$$d_r = +1 \text{ if } z_r < 0, \text{ else } -1. \quad (4.22)$$

The update of the coordinate is now simplified to an addition or subtraction and the term $d_r 2^{-r}$ can be implemented as a simple bit-shift. The recursive factor K_r can be compensated by multiplying the initial coordinates (x_0, y_0) with $1/A_R$. With the parameters of the oscillator put into the CORDIC processor

$$x_0 = s_2(n)/A_R, \quad (4.23)$$

$$y_0 = s_1(n)/A_R, \quad (4.24)$$

$$z_0 = \omega(n), \quad (4.25)$$

the processor output (y_R, x_R) returns the new states of the oscillator $s_1(n+1)$ and $s_2(n+1)$ and thus can be used to implement the Gold and Rader oscillator from the equations (2.33) itself.

However, there are some important limitations when using the CORDIC algorithm. The coordinate transform of the CORDIC processor is limited to rotation angles z_0 between $-\frac{\pi}{2}$ and $\frac{\pi}{2}$ [And98]. Since $\Omega(n)$ can range between 0 and π , the CORDIC algorithm requires additional logic to shift the number range for the initial rotation angle of the CORDIC-processor z_0 accordingly. Both parameters use a modulo- 2π representation to save further hardware resources, but the fixed-point number format differs slightly. $\Omega(n)$ uses an unsigned fixed-point format ranging from $[0, \pi[$ and z uses a signed fixed-point format ranging from $[-\frac{\pi}{2}, \frac{\pi}{2}[$. A value from the Ω -register is fed into z_0 -register without any transformation. Thus the CORDIC-processor interprets the MSB of the incoming value differently. This effectively interprets the angle z_0 as the same value as $\Omega(n)$, if $\Omega(n) < \pi/2$. However, if $\Omega(n) \geq \pi/2$ which means the MSB of $\Omega(n)$ is zero, then the angle z_0 is interpreted as $z_0 = \Omega(n) - \pi$. To compensate the 180° rotated angle, the initial coordinates in equations (4.23) and (4.24) are then being multiplied with $-A_R$ to obtain the correct result for (x_R, y_R) . The result is a shift of the allow input angle range z_0 from $[-\frac{\pi}{2}, \frac{\pi}{2}[$ to $[0, \pi[$ at the cost of an additional register that stores the constant $-A_R$.

The datapath of the sine sweep architecture is shown in figure 4.16. To begin the

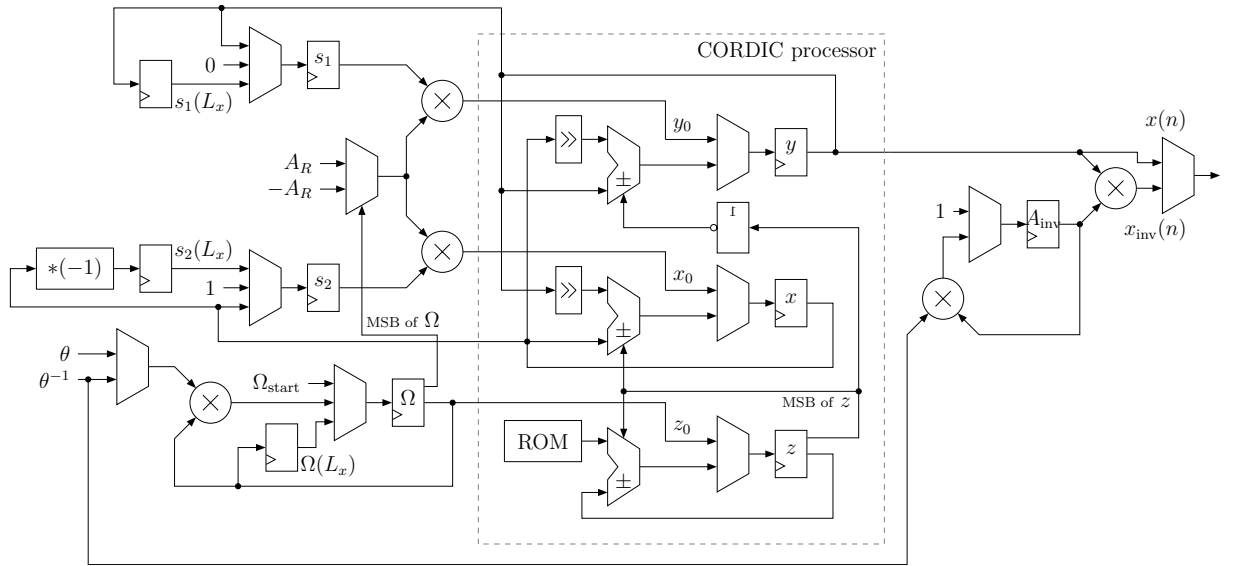


Figure 4.16.: The datapath of the component that generates the forward and reverse exponential sine sweeps including the CORDIC processor in rotational mode.

initial sweep with increasing frequency, the registers of the CORDIC-processor need to be initialized with the initial state of the oscillator according to equation (2.34) and (2.35). After R cycles of the CORDIC-processor, the state of the oscillator

$s_1(n)$ and $s_2(n)$ is updated and the succeeding angular frequency $\Omega(n)$ is calculated. After L_x samples are generated, one additional CORDIC-iteration is started and the state is stored in $s_{1/2}(L_x)$. The initial rotation frequency is stored in the register $\Omega(L_x)$.

The generation of the repeated, reversed sweep runs in an analogous manner. One difference is the concluding multiplication of the CORDIC output with the amplitude correction factor $A_{\text{inv}}(n)$ to obtain the reversed excitation signal $x_{\text{inv}}(n)$.

4.6.2. Amplitude Fade

An amplitude fade of the excitation signal is necessary to eliminate unwanted ripple effects in the frequency response near the starting and ending frequencies of the sweep [HCZ09]. Thus, the first and the last N_{fade} samples of the sweep with the length L_x get attenuated. The following algorithm was created, because it is more hardware efficient compared to common algorithms that use a quadratic equation for example.

The amplitude fade component multiplies the excitation signal with

$$A_{\text{fade}}(n) = 1 - \psi(n) \quad (4.26)$$

where $\psi(n)$ is a recursive factor defined as

$$\psi(n) = \begin{cases} 1 & \text{if } n = 0 \\ \psi(n-1) 2^{-12/N_{\text{fade}}} & \text{if } 0 < n \leq N_{\text{fade}} \\ \psi(n-1) 2^{12/N_{\text{fade}}} & \text{if } L_x - N_{\text{fade}} \leq n < L_x \\ \psi(n-1) & \text{otherwise.} \end{cases} \quad (4.27)$$

The architecture is plain and consists of two sequential multiplications, one subtraction and a register for $\psi(n)$. An exemplary progression of $A_{\text{fade}}(n)$ over 24 001 samples with different N_{fade} values can be seen in figure 4.17.

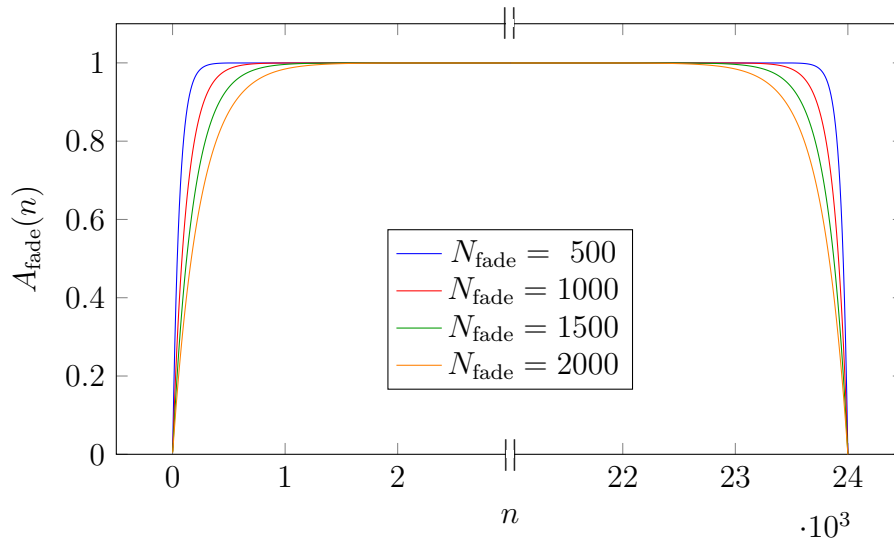


Figure 4.17.: The factor $A_{\text{fade}}(n)$ for the amplitude fade is plotted for a selection of N_{fade} over the first and last 3000 samples. The total sweep length in samples is $L_x = 24\,001$.

4.7. Transfer of the Secondary Path from System Identification to Controller

This section is in parts an excerpt of [Eck+22]. A problem arises when the system identification architecture needs to be applied to a physical system. After performing the system identification of a secondary path, the involved data has to be transferred from the running system identification component to the FxLMS controller. The extraction of this run-time data from block-memory and flip-flops can happen in different ways that each have advantages and disadvantages over each other.

The obvious choice is to configure the component for the FxLMS algorithm and the component for the system identification side by side to the FPGA and switch between them at run-time as needed. Both components then need to share the BRAM-tiles that contains the transfer function of the secondary path. Although for many ANC applications this solution is not ideal, as the ANC-system requires as much hardware resources as possible to maximize noise reduction performance. The additional hardware resources can be used to increase the number of I/O channels, the filter length, the bit precision or the sampling rate. Increasing the latter shortens the available processing time between each iteration. Particular when ANC systems with relatively large physical dimensions are concerned, research in [Shi+19] and

[Kle+21a] shows that even the resource capacity of modern, high-end FPGAs is easily exceeded. Thus all the available hardware resources need to be available to the FxLMS component when it is active. Consequently, the FPGA platform should be configured twice using both designs successively. Firstly, the identification algorithm needs to be run to acquire the required secondary path data. Afterwards this secondary path data needs to be included into the controller algorithm itself, before it is started. Since commercial ANC systems are typically implemented as embedded hardware, a transition of the FPGA configuration using as few processing power from outside of the FPGA-chip as possible is desirable.

A total of three approaches on how to switch between two FPGA configurations without the modification of specific memory locations are provided in the following. The first approach is certainly the most widely used principle. It adds a transfer protocol based on some transmission interface, in this case the serial communication protocol universal asynchronous receiver-transmitter (UART), and transfers the secondary path values outside of the FPGA chip. The data can afterwards be inserted into the FxLMS component before synthesis or at runtime using another data transport interface with the controller algorithm. The second approach uses dynamic partial reconfiguration (DPR) to switch between components while leaving the BRAM containing the secondary path values as part of the static design. Note that in the year 2019 *Xilinx Inc.* changed the method name from (dynamic) and partial reconfiguration to "dynamic function exchange". However, here the old term "(dynamic) partial reconfiguration" is used. The third approach uses a technique called context save restore (CSR) of partial reconfiguration regions, which essentially manipulates bit-files to include the data after synthesis.

4.7.1. Solution I - Serial Communication

This solution adds a transfer interface based on UART to the component for the system identification to extract the secondary path at runtime. The data for the coefficients of the secondary path can then be implemented as a constant and ROMs within the FxLMS component. This design flow results in no additional hardware resource allocation for the FxLMS component, which is the critical component resource-wise. One obvious disadvantage is that a new synthesis of the FxLMS component has to be performed each time a secondary path is updated. This a relatively resource intensive task, which is a high requirement for embedded systems.

One alternative to avoid this disadvantage is to add the transfer interface also to the memory elements of the secondary path in the FxLMS component. The secondary path values can then be inserted to the control component at runtime without the need of a new synthesis. Of course the interface itself requires hardware resources that are not available to the control component anymore. It is the goal of this section to switch configurations, while the control component has the maximum of hardware resources available. Thus, further investigations of this alternative approach will be discarded.

4.7.2. Solution II - Dynamic and Partial Reconfiguration

DPR is a technique that allows to split the FPGA's hardware resources into a static and dynamic domain. The static area can be used as usual and the dynamic area can be used to change between multiple configuration at run-time that share the same entity ports. The idea for the here proposed application is to assign the necessary memory resources for the secondary path values to the static domain and leave as much hardware resources as possible for the dynamic domain. Figure 4.18 shows both desired states of the FPGA configuration, where 4.18a represents the running controller algorithm and 4.18b represents the running system identification algorithm.

DPR allows to omit the transfer logic, as the secondary path data does not have to be transported at all. The data can stay on the chip, which also saves time in between the switch of both configurations. On the negative side, at least for FPGAs from *Xilinx Inc.*, there is no possibility to tell the synthesis tool to set a small static area and use the rest of the hardware area as the dynamic domain. Instead a dynamic area has to be explicitly set. This area has to be rectangular in shape and can be set graphically in the synthesis tool. This means that if certain resources like BRAM-tiles for example have to be in the dynamic domain, then the synthesis tool effectively creates a bounding box around those resources and all other resources in this area have to a part of the dynamic domain as well. This shows that the hardware resources can not be freely assigned to the either domain and can result in a drastic amount of resources that are present, but not useable in either the static or dynamic domain depending on the respective architecture. Additionally, the strict separation in static and dynamic design areas prohibits the synthesis tools from performing optimizations across both areas. For completeness,

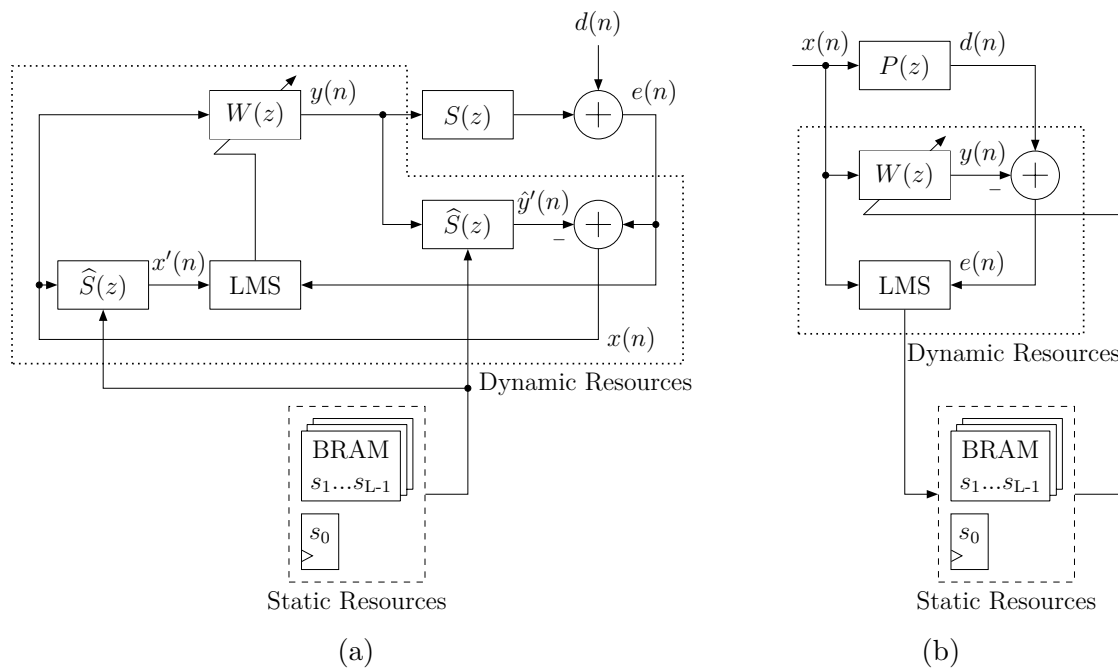


Figure 4.18.: (a) shows the feedback FxLMS controller divided by static and dynamic resources. The FxLMS controller does read-only from the static memory resources. Fig. (b) shows the system identification component. It does update the static memory resources to adapt to an arbitrary system $P(z)$ which in this case is the secondary path $S(z)$.

some overhead in form of decoupling logic is required to decouple the static and dynamic regions during a configuration switch of the dynamic area. This is due to spikes and hazards that can occur and accidentally distort the data values of the secondary path. The size of this decoupling logic is close to negligible and should not take critical resources away from the design. Similar to solution I, there is some sort of processor required to perform the reconfiguration of the dynamic area from outside of the FPGA-chip. Compared to performing a new synthesis, this task is much less computational intensive and is possible using a simpler processor.

4.7.3. Solution III - Bit-file Manipulation

This here used technique of saving and restoring a context was first introduced in [EMK19] by Eckert et al.. It is important to note that this technique is not officially supported by *Xilinx Inc.*. Though, the technique was proven to work with 7-series and *Ultrascale* FPGAs by *Xilinx Inc.*. The idea of this approach is to use the configuration interfaces of the FPGA to extract the secondary path coefficients from the FPGA while running the system identification design. The data is then inserted into an already fully synthesized Bit-file of the ANC-controller design. By default the Bit-file contains dummy coefficients at fixed, predefined locations in memory. By knowing the locations and having the necessary knowledge about how the Bit-file is formatted, the respective memory locations in this binary file can be edited as desired.

This solution combines some advantages of the previous two solutions. It avoids a new synthesis after a new system identification and also avoids any hardware resource overhead for the FxLMS controller. Some processing power from outside is still necessary to read the FPGA configuration after synthesis, to manipulate the controller Bit-files accordingly and to reconfigure the FPGA using the manipulated Bit-file. On the other hand, this workflow is not support by the FPGA vendor and it thus could break at any future point in time. The read-back feature is somewhat suggested by *Xilinx Inc.* in [Xil15], but the format of the Bit-file is specific to different FPGA models and was reverse-engineered only for specific models. Hence, the approach requires more work in the future and requires the respective vendor to refrain from any countermeasures.

For validation purposes, all three design-flows were tested on a simple experimental setup with the same single-channel feedback FxLMS implementation and all

achieved the same noise reduction performance. Section 5.4 discusses differences that occurred during the implementation.

4.8. Used Verification Methodology and Toolchain

Due to the many generic parameters and the high amount of different fixed-point number formats, the implementation of the given architectures can be error-prone and errors are hard to detect. This is especially true for components that use a lot of internal recursion, such as the adaptive FIR-filter or the sine sweep generator. Let us take the FxLMS implementations as an example. A miscalculation of the position of the coefficients's implicit decimal point might result in an unintentional multiplication or division by 2 and is hard to find. The adaptive FIR-filter might even be able to adapt to that calculation error and compensate it, so that a validation in a lab looks successful. Moreover the implementation might work as expected with a static step size, but can be flawed when step size normalization is used. A complete suite of unit tests that verify the design and its components with all the combination of generic parameters combined with the input signals is obviously not feasible. Even all combination of values for the generic parameters tested with a static input stream of data is out of scope. To keep the amount of test cases manageable, the combination of boolean generic parameters that enable different features and only a selection of limit values of the numeric parameters are tested. As input data, a pseudo-random data stream in the corresponding fixed-point number format is generated for each input. For this, an automated unit testing framework for VHDL has been used, called *VUnit* [Asp22]. As an example, a total of 123 test cases were created for the single- and multi-channel FxLMS implementations and its subcomponents that check different combinations of filter orders, multiplier lanes, channel dimensions and optional features enabled. The tests are included in the source code of the respective implementation.

The tests check for deviation against two reference models per unit. The reference models are implemented in VHDL, but use non-synthesizable code that implements the algorithms on a high abstraction level above the hardware level. Other reference implementations in programming languages like C or Matlab can be taken and adapted to the VHDL syntax. The first of the reference implementation uses a floating-point number format. This avoids most mistakes relating to a limited number range or bit precision. Also the here required arithmetic operations like the

division and trigonometric functions are implemented for floating-point numbers in non-synthesizable VHDL-code. Thus it is easy to check the CORDIC-processor or the AEGP component against equivalent functions that are part of the VHDL-standard. The second reference model is then also implemented on the high abstraction level, but uses the desired fixed-point number formats respectively. The difference in number formats for these reference models means that their number values are not identical and their behavior can deviate, especially when the algorithm makes heavy use of recursion where deviations add up. When an acceptable error threshold between the models is found, the conducted tests did trigger only when an implicit bit-shift or other error concerning the fixed-point number formats happened.

As a third model, the synthesizable implementation is developed that brings the design from the algorithmic abstraction level down to the register-transfer level. This synthesizable model is then checked against the fixed-point reference model. The incoming, outgoing and selected internal signals between both models must be identical to the last bit to pass the test, as they use the same number formats. This check is aimed primarily at finding differences in the correct execution of the algorithm. As an example, let us assume that within the synthesizable model in the convolution of the static FIR-filter an off-by-one-error occurs. If one product of a coefficient with a data sample is missing in the accumulation and the filter order is high enough, then the deviation of the filter output might be below the error threshold of the check with the floating-point reference implementation and thus the test would pass. However, the test fails with the second check against the fixed-point reference model, where the check does not tolerate any deviation.

5. Experimental Results and Evaluation

5.1. Used Platforms

The most widely used platform for the conducted experiments is the *DS1202*, also promoted as *MicroLabBox* by *dSpace GmbH*. This platform was mainly chosen for one reason: It houses an FPGA-chip and real-time capable processor that both have access to the same peripherals. Thus, it is excellent to compare equivalent hardware and software ANC implementations, where the platform itself is part of the secondary path. The processor and the FPGA can be used each individually or combined. The processor is a dual-core *NXP (Freescale) QorIQ P5020* processor clocked up to 2 GHz. The used FPGA is a *Xilinx Kintex-7 XC7K325T*. The system provides 1 GB of main memory. [dSP20]

Real-time software applications for this platform can be programmed graphically and most of the challenges of a real-time system are abstracted away from the developer to enable rapid-prototyping. Using the *DS1202* also brings some limitations. The platform is proprietary and any hardware modifications like upgrading the central processing unit (CPU) or FPGA are not possible. The development for this platform requires an increased software stack that requires *Matlab / Simulink* from *The MathWorks, Inc.* and *RTI, Control Desk* from *dSpace GmbH* on top of the mandatory synthesis tools. At present, the software tools on top of the synthesis tool *Vivado* from *Xilinx Inc.* prevent the synthesis tool from using the newer VHDL-2008 standard. This important changes of this standard allow for a much leaner and elegant code base. As a result, two code-bases supporting the VHDL-1993 and VHDL-2008 standards were developed and maintained.

For prototyping the evaluation board *Nexys DDR4* from *Digilent* was used. It uses a *Xilinx Kintex-7 XC7A100T* FPGA. The board is a simple and cheap platform with

a lot of interface options and can be configured by using *Vivado* only. Compared to the *DS1202*, this enables the use of the modern VHDL-2008 standard and DPR. The board does supply an on-chip dual-channel ADC with 12 bit precision and a maximum sampling rate of 1 MS s^{-1} . [Dig16a] Unfortunately no DAC components are present.

Due to the missing DAC interface, to make implementations better portable to other platforms and to make the audio interface expandable, two types of external modules as the ADC/DAC interface are used: The *PMOD AD1* and the *PMOD DA2* from *Digilent*. Both modules provide a dual-channel interface with 12 bit precision and a sampling rate up to 1 MS s^{-1} . The input/output signals are unipolar, as the analog voltages range from 0 V to 3.3 V [Dig16b; Dig16c]. This makes additional analogue circuitry necessary, as the attached loudspeakers and microphones expect or return bipolar signals. A circuit using operational amplifiers was developed that converts the unipolar output signals from the DAC to bipolar signals ranging from -3.3 V to 3.3 V and vice-versa for the input signal to the ADC. The circuit diagram can be found in the appendix as figure A.1. The included operational amplifiers introduce a negligible latency of $4 \mu\text{s}$ [Tex21]. A VHDL component was developed that handles the SPI communication protocol with both modules internally. It allows to trigger the DAC channels asynchronously from the ADC.

As a feasibility study, the *Virtex Ultrascale+ XCVU9P* from *Xilinx Inc.* is used for the multi-channel feedback FxLMS architecture to demonstrate possible configurations and to validate the scaling of the utilized hardware resources on high-end FPGAs. This board was not yet applied to any experimental setups, but could be used with multiple *PMOD AD1* and *PMOD DA2* modules in the future.

The available hardware resources of the mentioned FPGA-chips are shown in table 5.1.

Table 5.1.: FPGA-types that are used in the following experiments and their available resources.

FPGA-type	XC7A100T	XC7K325T	VU9P
CLB-Slices	15 850	50 950	147 780
BRAM-Tiles	135	445	2160
DSP-Slices	240	840	6840

5.2. Feedback FxLMS Controllers

5.2.1. Configuration Strategies

The provided parameters in the hardware design can be divided into three categories: Those that affect only the ANC performance, those that affect only the required hardware resources and those that affect both. The parameters that affect only the ANC performance are disregarded in this section and can be set purely after acoustic reasoning. For the the sampling rate, the static and adaptive filter lengths and the bit precisions that also affect hardware utilization, usually a trade-off has to be found for multi-channel ANC-applications due to limited hardware resources. In the conducted experiments the bit precision of the incoming and outgoing data from the A/D conversion was set to the precision that the converters provide. The filter coefficients bit precision is set to double the I/O data bit length. Regarding to why this precision was chosen, Kuo et al. write "When the convolution sum in Eqs. (B.10) and (B.11) is calculated using a multiplier with an internal double-precision accumulator, the round-off noise is far below the level of the input quantization noise" [KM95, p. 334]. The bit widths for the other parameters were found empirically during validation and are presented in section 5.2.2. The bit widths of these parameters were individually decreased until the ANC performance or stability declined and then increased back a bit as a safety measure. When the desired values for the parameters and the priorities between them is determined, we can use the equations (4.9) or (4.14) to check if the relation between the sampling rate and the number of required clock cycles due to the other parameters is met. The parameters for the number of multiplier lanes for the static and adaptive filters N_{static} and N_{adapt} do not affect the ANC performance, but are fundamental make the configuration satisfy the equations above. Both parameters should be set as low as possible to have the design use just less clock cycles than f_{clk}/f_s provides. This way the design uses fewer hardware resources without any losses to ANC performance.

5.2.2. Configurations and Synthesis Results

The configurations in table 5.2 were chosen for validation purposes and to demonstrate the required hardware resources. All configurations make use of step size normalization and the leakage factor due to the improved performance and stability.

Single-Channel Feedback FxLMS

The configuration SS1 is performed on the processor of the *DS1202*. This software implementation is the reference implementation to check that the hardware design behaves the same under equal conditions. Hence, the hardware configuration SH1 uses the same parameters. The configurations SH2 to SH4 increase the sampling rate above of what the software system is capable to process in real-time. Those configurations are used to measure how much the noise reduction performance can benefit from the additional computational power. Since the adaptive filter does now update its coefficients multiple times compared to SH1, the step size has to be decreased to keep the filter from overcompensating. This is achieved by reducing the value for α gradually.

Table 5.2.: Synthesized configurations for the single-channel feedback FxLMS architecture. [Tim+20]

Configuration	SS1	SH1	SH2	SH3	SH4
f_s in kHz	20	20	40	80	160
L_{static}	513	513	1025	2049	4097
L_{adapt}	513	513	1025	2049	4097
N_{static}	-	1	1	2	4
N_{adapt}	-	1	1	2	4
α	0.0003	0.0003	0.0001	0.00009	0.00003
β	0.001	0.001	0.001	0.001	0.001
γ	0.005	0.005	0.005	0.005	0.005
P_{min}	0.0001	0.0001	0.0001	0.0001	0.0001
$\hat{P}_x(0)$	0.5	0.5	0.5	0.5	0.5
bw[$e(n)$], bw[$y(n)$]	double	16	16	16	16
bw[\hat{s}], bw[$w(n)$]	double	32	32	32	32
bw[$\mu(n)$]	double	24	24	24	24
bw[$\hat{P}_x(n)$]	double	32	32	32	32

For the following synthesis results and evaluation, the *DS1202* was used together with *Vivado 2016.2*. Table 5.3 shows the synthesis results of the presented configurations. None of the configurations are close to fully utilize the available hardware resources. Comparing SS1 to SH4, the available hardware resources of the used FPGA allow an increase of the sampling rate and filter lengths each by a factor of 8, while the relative hardware utilization is still below 12% in the worst case. From the three categories of hardware resources, the DSP-slices is the most used resource

Table 5.3.: Synthesis results of the presented configurations performed on the *DS1202* platform.

Config.	CLB-Slices		BRAM-Tiles		DSP-Slices	
	Used	(%)	Used	(%)	Used	(%)
SH1	4365	(8.57)	3.5	(0.79)	59	(7.02)
SH2	4354	(8.55)	4.5	(1.01)	59	(7.02)
SH3	4507	(8.85)	8	(1.80)	72	(8.57)
SH4	4815	(9.45)	15	(3.37)	98	(11.67)

in relative terms. From configuration SH1 to SH2 the filter lengths are increased. Since the number of multiplier lanes stay the same, the number of utilized DSP-slices stay constant. The number of utilized BRAM-tiles stays also unchanged. This is due to the fact that the block-memory can not be arbitrarily fragmented and thus one BRAM-tiles is needed per multiplier lane per FIR filter. More block-memory is utilized when $\frac{L-1}{N}$ exceeds the depth of the BRAM-tiles or more multiplier lanes are being used. The latter is the case from configuration SH2 to SH3 and from SH3 to SH4. The utilization of CLB-slices is not reasonable to analyze, because they are used as flip-flops and general logic in all components. Also from the three categories it is the least critical resource.

Multi-Channel Feedback FxLMS

The configurations of the multi-channel FxLMS are presented in table 5.4. Again configuration MS1 is performed on the processor of the *DS1202* to be later compared against the equivalent hardware configuration MH1. The processor operates at full capacity for this configuration.

Table 5.4.: A parameter overview of the multi-channel FxLMS configurations. [Kle+21a]

Configuration	MS1	MH1	MH2	MH3	MH4	MH5	MH6	MH7
f_s in kHz	20	20	40	80	160	40	40	40
$K \times M$	2 x 2	2 x 2	2 x 2	2 x 2	2 x 2	5 x 5	6 x 6	11 x 11
L_{static}	129	129	513	1025	2049	2049	2049	2049
L_{adapt}	129	129	513	1025	2049	2049	2049	2049
N_{static}	-	1	1	2	4	1	1	1
N_{adapt}	-	1	1	2	4	1	1	1
α	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001
β	0.001	0.001	0.001	0.001	0.001	0.001	0.001	0.001
γ	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001
P_{min}	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001
$\hat{P}_x(0)$	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5
bw[$e(n)$], bw[$y(n)$]	double	16	16	16	16	16	12	16
bw[\hat{s}], bw[$w(n)$]	double	32	32	32	32	32	24	32
bw[$\mu(n)$]	double	24	24	24	24	24	24	24
bw[$\hat{P}_x(n)$]	double	32	32	32	32	32	32	32

The configurations MH2 to MH4 are used to measure additional noise reduction performance by increasing the sampling rate. The chosen experimental setup can not be extended to more than 2 error sensors and 2 secondary sources, so configuration MH5 and MH6 are synthesized to show feasible configurations for applications that benefit from higher dimensions.

The corresponding synthesis results are shown in table 5.5. As before, starting from

Table 5.5.: Synthesis results of the presented multi-channel configurations performed on two different platforms. [Kle+21a]

Config.	CLB-Slices		BRAM-Tiles		DSP-Slices	
	Used	(%)	Used	(%)	Used	(%)
MH1	1983	(3.89)	10	(2.25)	102	(12.14)
MH2	2053	(4.03)	10	(2.25)	102	(12.14)
MH3	2993	(5.87)	20	(4.49)	166	(19.76)
MH4	4957	(9.73)	38.5	(8.65)	294	(35.00)
MH5	16 075	(31.55)	235	(52.81)	720	(85.71)
MH6	14 468	(28.40)	336	(75.51)	756	(90.00)
MH7	71 686	(48.51)	1835	(84.95)	5412	(79.12)

MH1 the configurations to MH4 do scale similar to their single-channel variants. The Configurations MH5 and MH7 show that symmetrically increasing the I/O channels, does result in a massive increase of utilized hardware resources. Configuration MH5 demonstrates the maximum number of I/O channels for the given platform. The limiting resource is the number of available DSP-slices. Since addertrees use DSP-slices, this is why the LMS update component opted for a chained architecture instead. The change resulted in a reduction up to 7% of utilized DSP-slices compared to an older architecture used in [Kle+21a].

The individual filters are already parametrized to work fully sequentially, because $N_{\text{adapt}} = 1$, $N_{\text{static}} = 1$ and thus without any addertree structures. Further optimizations towards a lower DSP-slices utilization can not be done without affecting ANC performance. However, lowering the data bit widths of $\text{bw}[e(n)] = 12$, $\text{bw}[y(n)] = 12$ and the coefficients bit widths to $\text{bw}[\hat{s}] = 24$, $\text{bw}[w(n)] = 24$ results in a 25% reduction of utilized DSP-slices for configuration MH5 and allows for a 6x6 configuration, see MH6. When the same modifications are applied to MH7 the synthesis tools reports that the *VU9P* FPGA is missing 21 additional BRAM-tiles for this configuration. If this reduction in bit precision is feasible for most ANC-applications is debatable. A first comparison of noise reduction performance between two config-

urations of the multi-channel architecture, where only the bit precision is modified, shows some potential. If required, the corresponding plot [A.2](#) can be found in the appendix. While the noise reduction is comparable, the bit-reduced result shows more roughness in the frequency spectrum and might be less stable.

5.2.3. Interpretation of Utilized Hardware Resources

For future applications it might be of interest to estimate the BRAM and DSP-slice utilization of the presented architecture. With the knowledge about the implemented architecture, equation (2.39) can be tweaked to return the necessary amount of DSP-slices used for multiplications for a given configuration. When the sampling rate is removed from equation (2.39), it returns the total number multiplications required. The multiplications of each FIR-filter are performed by N_{static} or respectively N_{static} multiplier lanes sequentially, so

$$\begin{aligned}
 N_{\text{MUL}} = & KM((M + 1)N_{\text{static}} + N_{\text{adapt}}) + \\
 & KM(M + 2)N_{\text{adapt}} + \\
 & 6M
 \end{aligned} \tag{5.1}$$

returns the total number of multipliers for the FxLMS architecture. The equation is split into three lines, where the first line are the multipliers needed for the convolution of the static and adaptive FIR filters, the second line represents the multipliers required for the update of the adaptive coefficients and the third line shows the multipliers for the update of $\mu_j(n)$ and $\nu_j(n)$. Section 3.3.2 shows that one multiplication does not directly translate to one DSP-slice. Due to the bit widths of the coefficients, error signal and anti-noise signal, the configurations MH1 to MH5 and MH7 require two DSP-slices per multiplier lane. The configuration MH6 suffices one DSP-slice for the same task. For all configurations the multiplications in the second line of equation (5.1) use one DSP-slice per multiplication of $e(n)x'(n)$ and two DSP-slices the remaining two multiplications in the LMS update component. The multiplications of the third line each use 2 DSP-slices while both multiplications in the AEGP algorithm use three. Three DSP multiplications are required here, because the quotient needs have the bit width of the numerator and denominator combined. The resulting estimation of utilized DSP-slices can be found in table 5.6.. There is a deviation from the estimation and the synthesis results from table 5.5 of 5-15%. It seems that this number of DSP-slices is used for other functionality than

Table 5.6.: Estimated resource utilization of the presented configurations including the relative deviation from the actual synthesis results.

Config.	MH1-2	MH3	MH4	MH5	MH7
DSP-Slices	88	140	252	645	5115
Deviation in %	-14	-15	-14	-10	-5

implementing multipliers.

In a similar way the memory requirements of section 2.3 can be applied to the presented architecture and further specified. The number of required BRAM instances can be found in table 5.7 together with the required depth and bit width of these instances. Again the number of BRAM instances can not be directly mapped to

Table 5.7.: Estimated resource utilization of the presented configurations.

Content	RAM Instances	RAM-Depth	RAM-Width
Adaptive filter coefficients	KMN_{adapt}	$(L-1)/N$	$\text{bw}[w(n)]$
Static filter coefficients	KMN_{static}	$(L-1)/N$	$\text{bw}[s]$
Reference signal	$M\max[N_{\text{static}}, N_{\text{adapt}}]$	$(L-1)/N$	$\text{bw}[x(n)]$
Filtered reference signal	KM^2N_{static}	$(L-1)/N$	$\text{bw}[x'(n)]$
Antinoise signal	KN_{static}	$(L-1)/N$	$\text{bw}[y(n)]$

the number utilized BRAM-tiles. Section 3.3.3 mentions the possible aspect ratios a BRAM-tile can be used in. The filter lengths for all configurations are constant, so the depth of BRAM-tiles changes with the number of multiplier-lanes N_{static} and N_{adapt} . When the parameters for the individual configurations are included in table 5.7 and the resulting RAM depth and width is then compared with the tables 3.2 and 3.3, the number of utilized BRAM-tiles can now be estimated. The result is shown in table 5.8. For the configurations MH1-4 manages to utilize a *RAMB18E1* to store the filter coefficients and the I/O samples. A *RAMB36E1* BRAM-tile can be configured as two independent *RAMB18E1* blocks, which is why this block shows up as a 0.5 BRAM-tile in the synthesis report. The configurations MH5 and MH6

Table 5.8.: Estimated resource utilization of the presented configurations.

Config.	MH1-2	MH3	MH4	MH5	MH7
BRAM-Tiles	10	20	40	235	1837
Deviation in %	0	0	3.9	0	0.1

use only on multiplier lane and thus use a BRAM-depth of 2048. The corresponding tables for the aspect ratios of the available BRAM types show that the memory to store the coefficients has to utilize two BRAM-tiles per filter and one BRAM-tile per I/O signal. Table 5.8 shows that in two cases, for MH4 and MH7, the synthesis tool was able to optimize some BRAM-tiles away by using other resources, such as LUTs.

5.2.4. Experimental Setup

For validation of the single-channel FxLMS design, a duct with a length of 1440 mm and an inner square cross section of 95 mm x 95 mm is used. A schematic diagram showing the used acoustic equipment and a photograph is presented in figure 5.1. More details on the experimental setup are presented in [Tim+20]. Important to note is that due to the Nyquist-Shannon sampling theorem, anti-aliasing and reconstruction filters are used at the A/D interface. However, for configurations SH3 and SH4 these analogue filters were removed to reduce the processing latency of the secondary path. The reason this can be done while still satisfying the Nyquist-Shannon theorem is that the microphones and loudspeakers all have a natural cut-off frequency f_c at 20 kHz or below.

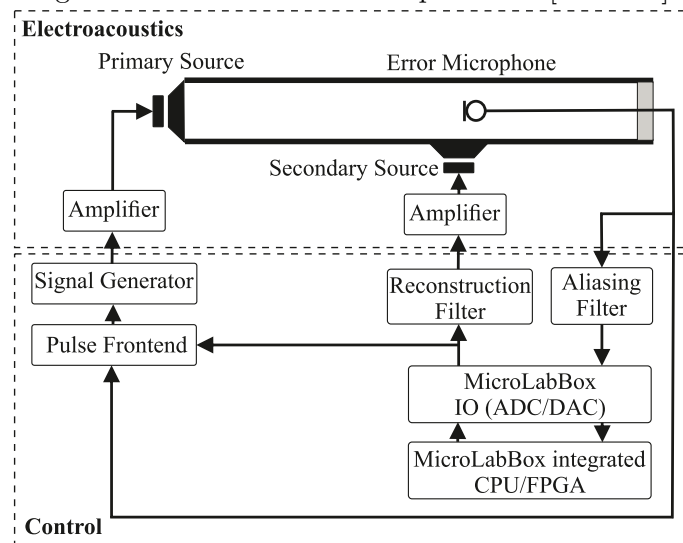
To validate the multi-channel implementation an active headrest is simulated. One pair of loudspeakers and microphones is placed next to each ear canal of a dummy head. The dummy head is located in front of a headrest to simulate a passenger. This setup is placed in an anechoic room. Figure 5.2 shows the photograph combined with a schematic of the connected components. Also for this setup, the anti-aliasing and reconstruction filters are omitted for sampling rates above 40 kHz. More details on the experimental setup are presented in [Kle+21a].

5.2.5. Validation and Noise Reduction Performance

For validation a primary source is producing limited Gaussian white noise between 200 Hz and 600 Hz in all experiments. The first measurements compare the software reference implementation with the configuration SS1 against the equivalent hardware configuration SH1. Figure 5.3 shows the three frequency spectrums of the recorded sound at the error microphone. The first one shows the remaining error when the ANC system is turned off, the second and the third one shows the remaining error



(a) A Photograph of the experimental setup with the duct placed behind the used filter, amplifier, signal generator and the hardware platform. [Han+18]



(b) A schematic of the experimental setup showing the signal flow between the involved components. [Han20]

Figure 5.1.: The experimental setup using a duct.

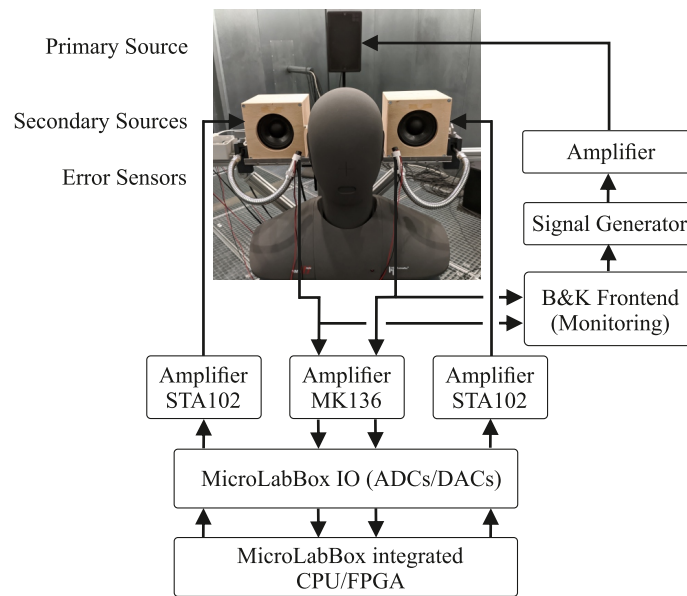


Figure 5.2.: The experimental setup for the multi-channel FxLMS variant using an active headrest. [Kle+21a]

when the corresponding ANC is active and has already converged to a static state. The ANC systems both cause noise reduction in the 200 Hz to 600 Hz band of about 5.9 dB compared to the systems being turned-off. Also both implementations show the same system behavior over the measured frequency range up to 2.5 kHz. The second measurement series goes beyond the purpose of validation and has the goal to allow the assumption that larger-scale single-channel applications can benefit from the increased computational capabilities. Figure 5.4 shows the noise reduction performance of configurations SH1 to SH4. With increasing sampling rate and filter lengths, the noise reduction performance does also increase in this experiment. This is true within the excited frequency range of 200 Hz to 600 Hz. Outside of this range the measured sound pressure level (SPL) does actually increase, but this is plausible and due to a known acoustic effect, called the waterbed-effect [Eli00, p. 286].

The validation of the multi-channel variant uses primary noise with the same characteristics as before. Figure 5.5 shows very similar system behavior between the software configuration MS1 and the equivalent hardware configuration MH1. Also both sides of the active headrest yield very similar results. When the additional configurations are applied to the experimental setup, remarkably similar gains in noise reduction performance compared to the single-channel FxLMS measurements can be found in figure 5.6.

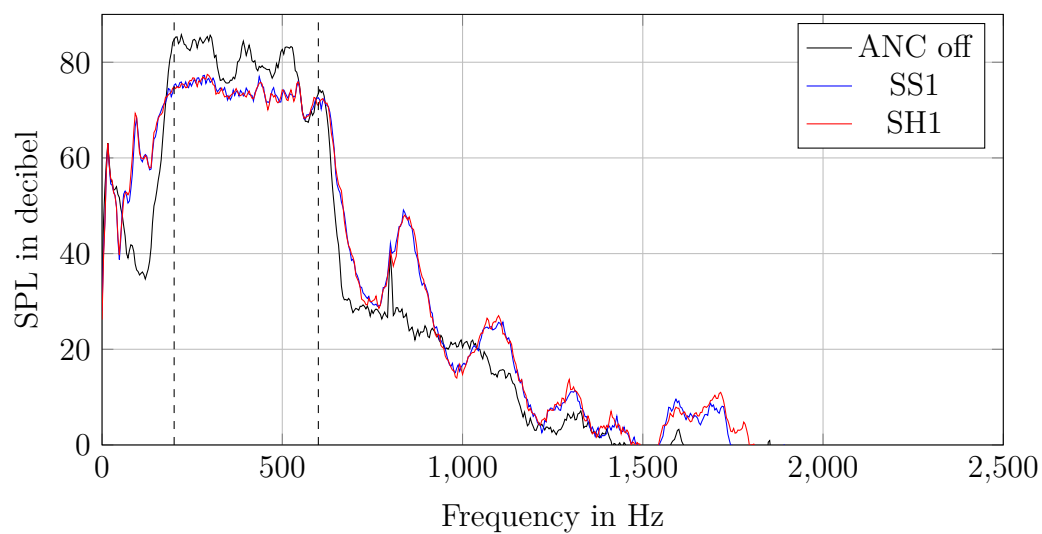


Figure 5.3.: Comparison of the software-based configuration SS1 against the equivalent hardware configuration SH1. [Tim+20]

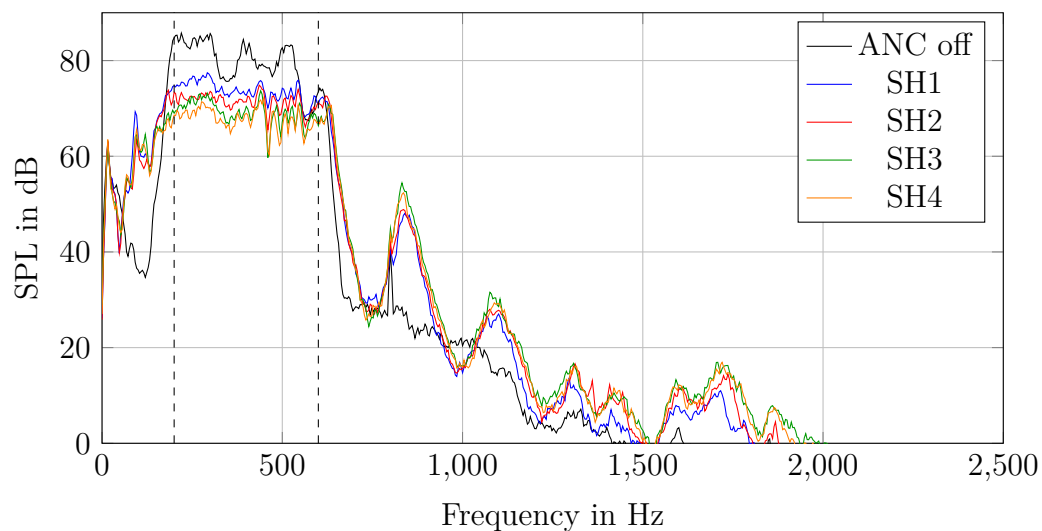
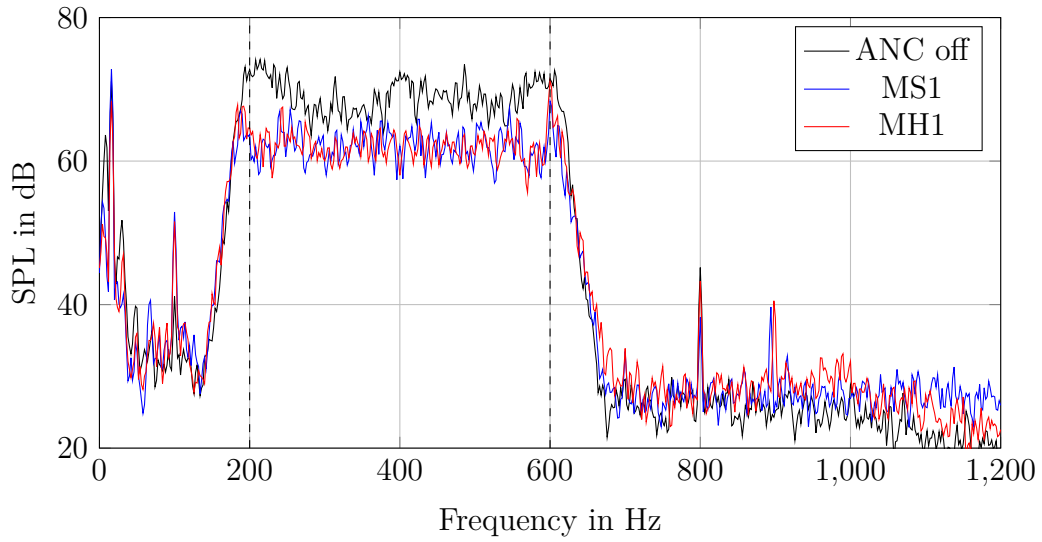
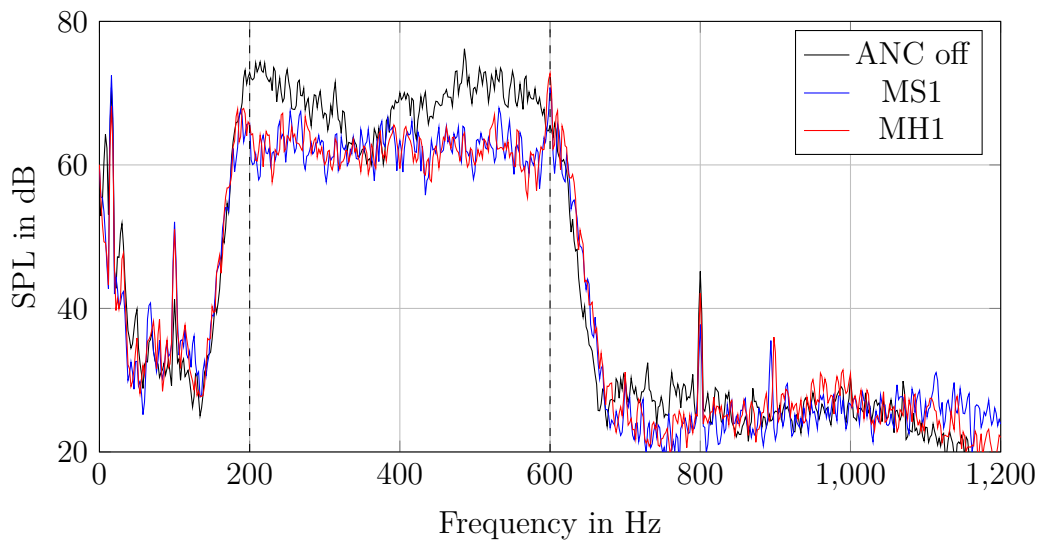


Figure 5.4.: Comparison of configuration SH1 to SH4 with achieved noise reduction between 200 Hz to 600 Hz: SH1: 5.9 dB, SH2: 8.3 dB, SH3: 10.5 dB, SH4: 11.9 dB. [Tim+20]

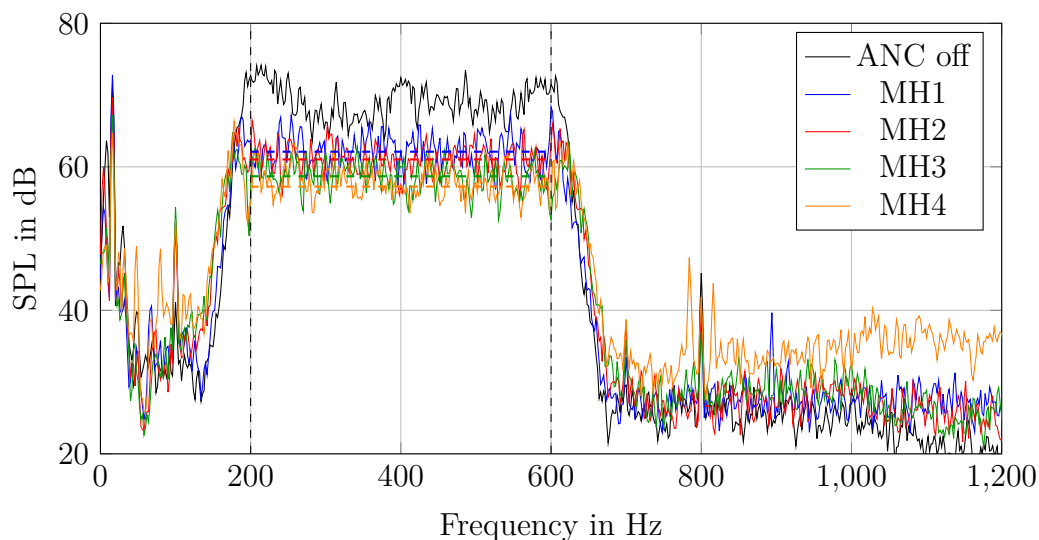


(a) This channel is located on the left side of the dummy head.

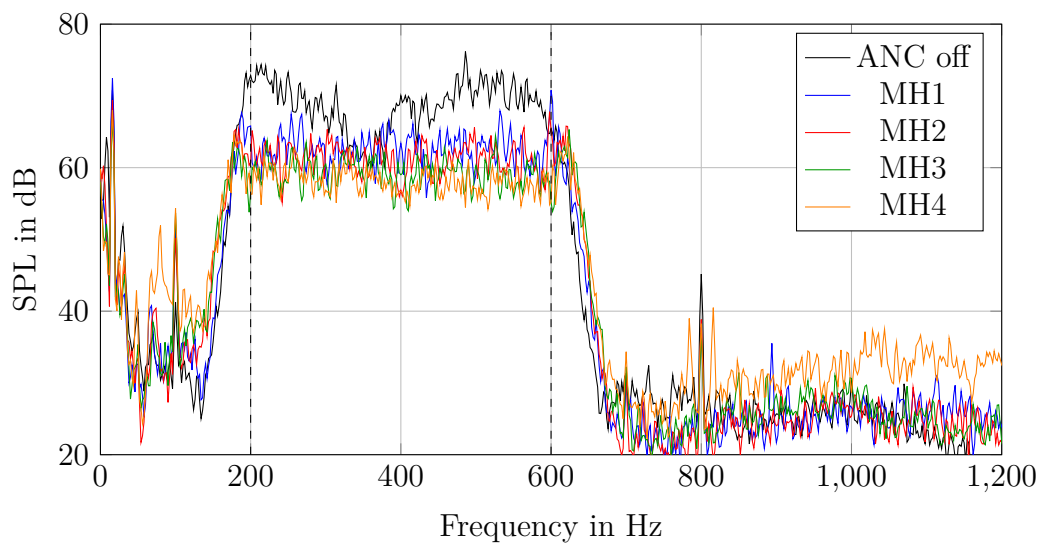


(b) This channel is located on the right side of the dummy head.

Figure 5.5.: Comparison of the software-based configuration MS1 against the equivalent hardware configuration MH1.



(a) The left-side channel of the dummy head.



(b) The right-side channel of the dummy head.

Figure 5.6.: Comparison of configuration MH1 to MH4 with achieved noise reduction between 200 Hz to 600 Hz: MH1: 7.0 dB, MH2: 8.1 dB, MH3: 10.4 dB, MH4: 11.9 dB. [Kle+21a]

5.3. System Identification Using Sine Sweeps

5.3.1. Configurations and Synthesis Results

Like before configuration C1 was chosen as it represents a maximum configuration for the software system, with $T_x = 0.5$ s, $f_s = 48$ kHz, $f_{\text{start}} = 20$ Hz, $f_{\text{end}} = 20$ kHz, and $N_{\text{fade}} = 500$ samples. The short sweep time of $T_x = 0.5$ s results from the limitations of the processor executing the software model in real-time. This is not intuitive at first, as the processor should primarily be challenged by the sampling rate, but the issue is related to a limited buffer- or cache-size. Configuration C2 uses the same parameters except that the sweep time is increased to $T_x = 10$ s. This second configuration is closer to a full utilization of the available block-memory according to equation (2.41) in section 2.3.

Table 5.9.: The utilized hardware resources for the presented configurations. [Kle+21b]

Config.	CLB-Slices	BRAM Tiles	DSP-Slices
	Used (%)	Used (%)	Used (%)
C1	4530 (8.89)	120 (26.96)	36 (4.28)
C2	5007 (9.82)	368 (82.69)	36 (4.28)

5.3.2. Validation and Noise Reduction Performance

The LMS-based hardware identification can be already be considered validated by producing secondary path estimates in section 5.2.5 that perform identical to the software LMS-based identification. This section is validating if the proposed hardware architecture of the system identification using sine sweeps behaves like the software reference. Firstly, figure 5.7 compares a section of the excitation signal of the reference model and the hardware implementation with the given configuration. Both sine sweeps show a slight offset at the first two turning points of the sine wave. With increasing frequency this offset does not drift and becomes indistinguishable visually.

For a validation of the whole hardware design, the impulse responses of two different systems are measured and compared with the measurements of the software reference

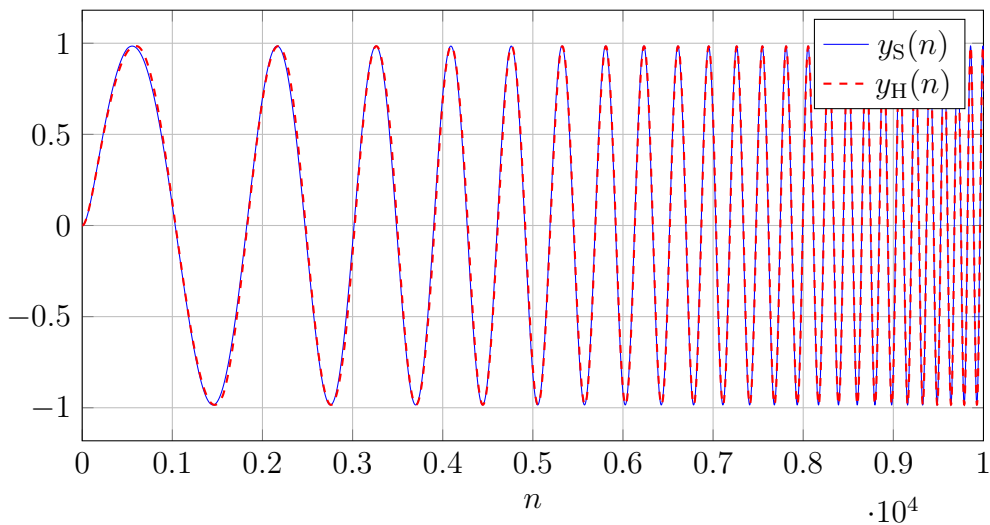
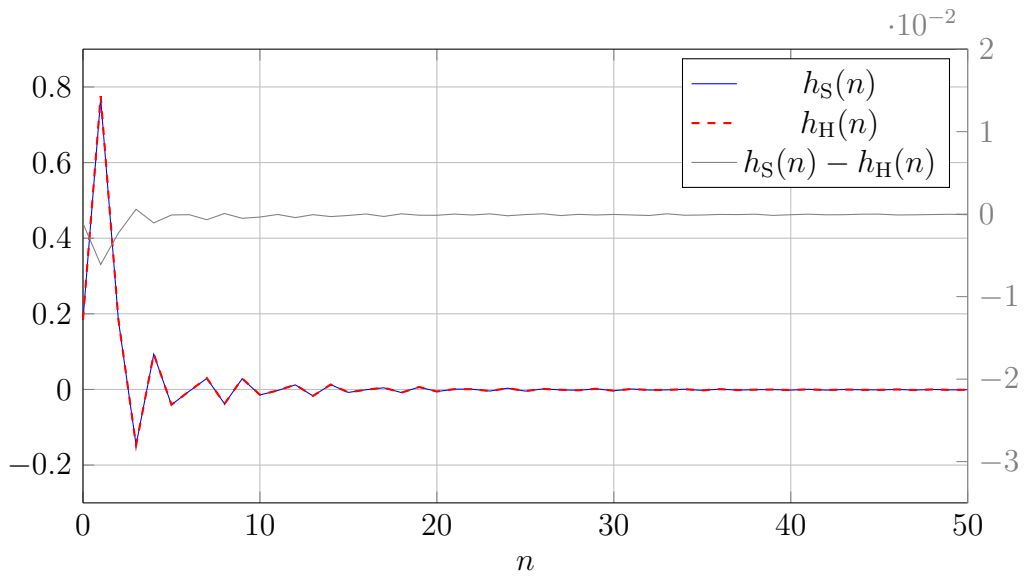


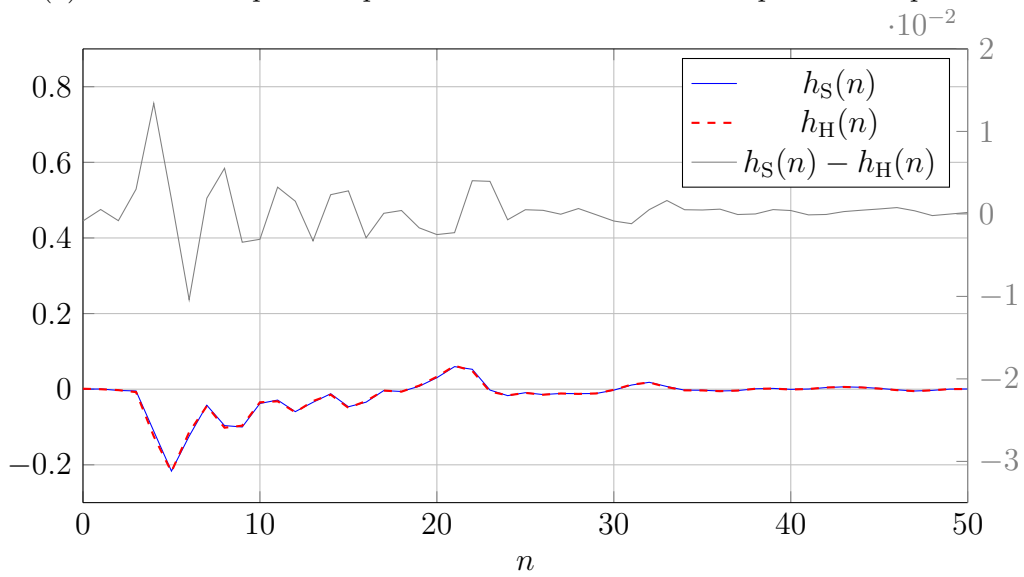
Figure 5.7.: Comparison of the generated exponential sine sweeps on the processor and the FPGA for the first 0.2 seconds with configuration C1. [Kle+21b]

model. The first system is created by simply connecting the output port of the FPGA with its input port. Ideally the impulse response should look like a Dirac delta. The second system is a non-trivial system whose acoustic path contains a headphone with a loudspeaker and microphone in the earcup placed on an artificial head that is similar to the one used in figure 5.2. The used parameters of the algorithm are the same as in figure 5.7. The measurement results from figure 5.8 look to be very similar. To also obtain a quantitative impression, the absolute error between the software and hardware models is plotted in gray using the secondary y-axis. Some differences between the software and hardware models were expected, as they use different number formats and numeric algorithms to implement the main algorithm. Further causes could be the different processing and sampling delays between the processor and the FPGA platform. More details about the experimental setup and an in-depth acoustic analysis, including a comparison of the magnitude responses, can be found in [Kle+21b].

Another measurement was conducted that compares both hardware implementations for the system identification as a brief sanity check and for interested parties. This measurement uses the duct from section 5.2.4 as an experimental setup using a single error microphone and secondary loudspeaker pair. Both implementations operate at a sampling rate of 48 kHz. The sine sweep-based component uses sine sweeps starting from $f_{\text{start}} = 20$ Hz to $f_{\text{end}} = 20$ kHz over a sweep time of $T_x = 0.5$ s. The LMS-based component uses an adaptive filter length of $L_{\text{adapt}} = 2049$ and a static stepsize of $\mu = 0.01$. The excitation signal used with this component is a limited Gaussian



(a) Measured impulse responses of the short-circuited input and output.



(b) Measured impulse responses of the headphone channel.

Figure 5.8.: Comparison of the measured impulse responses of the software and hardware implementation for both acoustic systems. [Kle+21b]

white noise between up to 25.6 kHz. The measured acoustic systems can be found in figure 5.9. Naturally, due to the approaches to measure the acoustic systems, the

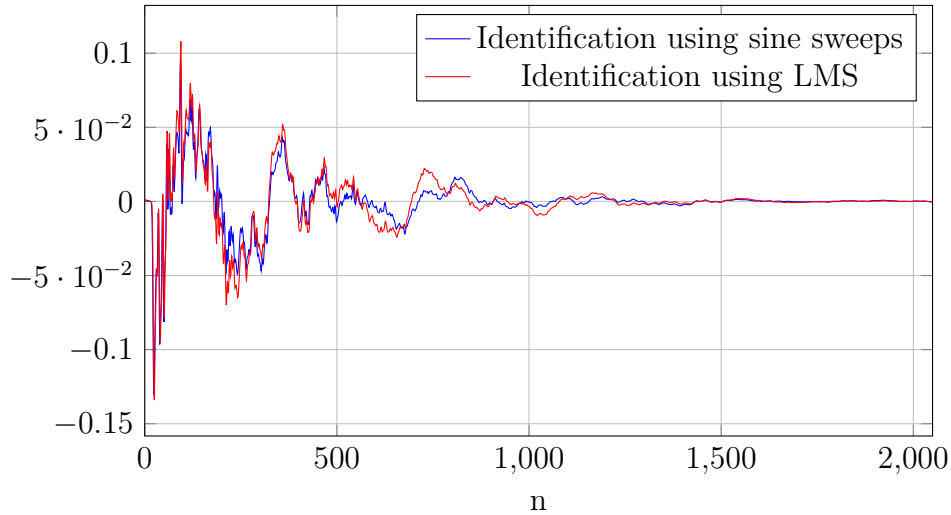


Figure 5.9.: Comparison of the LMS-based system identification hardware implementation against the system identification component using sine sweeps.

results are prone to differ. The measured coefficients show some deviations, but are overall quite comparable. Especially the coefficients with high absolute values show a good correlation, which are the most important ones to characterize the system.

5.4. Evaluation of the Transfer Solutions of the Secondary Path Data

The here used FPGA is a *Xilinx Artix-7 XC7A100T-1CSG324*. The synthesis result were performed using *Vivado 2018.3*. The three approaches use the same single-channel FxLMS implementation and in the same configuration that can be found in table 5.10. Note that the parameters are increased beyond what is beneficial for the experimental setup with the goal to magnify possible resource overheads or limitations of assignments of dynamic or static domains. The LMS-based system identification component uses the same parameters where they are applicable. The analysis of the synthesis results focusses on the FxLMS controller, as this component is the critical resource-wise. Table 5.11 shows the corresponding synthesis results for the three proposed solutions. Firstly, the synthesis results of solution I and III are identical, as they have no communication interface and are of free of any resource

Table 5.10.: Chosen configuration of the FxLMS component that maximizes the resource utilization of the underlying hardware platform. [Eck+22]

Parameter	Value
f_s in kHz	48
L_{static}	32 769
L_{adapt}	32 769
N_{static}	32
N_{adapt}	32
α	0.00005
β	0.001
γ	0.0001
P_{min}	0.0001
$\hat{P}_x(0)$	0.1
bw[$e(n)$], bw[$y(n)$]	12
bw[\hat{s}], bw[$w(n)$]	24
bw[$\mu(n)$]	24
bw[$\hat{P}_x(n)$]	24

Table 5.11.: The synthesis result for the chosen configurations. [Eck+22]

Resource Type	Solution I		Solution II		Solution III		Avail. FPGA
	Used	Util.%	Used	Util.%	Used	Util.%	
CLB-Slices	5900	37.22	6161	38.87	5900	37.22	15850
BRAM	112	82.96	112	82.96	112	82.96	135
DSP	230	95.83	230	95.83	230	95.83	240

overhead. The only difference is that solution III has fixed resource locations for the memory resources that store the secondary path. However, since these locations are best determined after synthesizing the FxLMS controller without any locations restrictions, they should be the same. Secondly, the usage of BRAM and DSP-tiles is unchanged over all three solutions, as these resources used only for specific tasks in the FxLMS controller. And thirdly, solution II results in a resource overhead using more CLB-slices, as can be found in table 5.12. These additional resources result

Table 5.12.: The resource utilization divided for the dynamic and static domains of the FxLMS controller with solution II.[Eck+22]

	CLB-Slices		BRAM		DSP	
	(Used	/ Avail.)	(Used	/ Avail.)	(Used	/ Avail.)
static	265	/ 2000	32	/ 40	0	/ 0
dynamic	5896	/ 13840	80	/ 95	230	/ 240
total	6161	/ 15850	112	/ 135	230	/ 240

from additional routing and glue-, or decoupling-logic due to placement constraints.

Together with the considerations regarding the required processing power of each solution from section 4.7, it is clear that there is no dominant solution over the others. From a technical standpoint, solution III looks promising, as it does not bring any hardware overhead or placement constraints and also has low processing requirements to perform the context switch. However, the method is not officially supported and could not be feasible with future other FPGA-families.

6. Conclusion

6.1. Summary

In order to narrow the technology-gap for ANC-systems applied to complex structures with a focus on increased sampling rates, this thesis selects the FPGA-technology as the platform. For this technology, hardware architectures for the given ANC algorithms are optimized down to the RTL while providing parameters to tune the algorithm and its architectural implementation. The goal is to maximize the available hardware resource utilization, while keeping the configuration of the algorithm flexible for a range of applications.

For the feedback FxLMS architecture, the increased computational performance can be prioritized towards high filter orders, high channel dimensions or the sampling rate. A constant short processing delay of a single or low double-digit clock delay is achieved that is independent of the respective filter orders. The most computational demanding, synthesized configuration uses 11 error sensors, 11 secondary sources and filter orders of 2049 at a sampling rate of 48 kHz. For this configuration, this translates to delay times well below 1 μ s for the given system clock frequency. The corresponding synthesis results show that the architecture utilizes the different, available hardware resource types uniformly for the used platforms with block-memory and DSP-slices being the critical resources depending on the respective configuration. All of the presented architectures are validated against established software reference models. The available parameters of the here presented hardware implementations are then tweaked to further utilize the available processing performance. On multiple practical applications, first results show an additional noise reduction performance of up to 12 dB compared to the maximum possible configuration of the software reference.

Besides the classical method of LMS-based system identification, a state-of-the-art

approach is implemented that uses exponential sine sweeps. This technique should obtain the impulse response of a system with an improved quality, but its main advantage is the suitability for stand-alone operation. The feasibility of a stand-alone FPGA-based platform is studied that is able to perform the system identification and automatically switch to the ANC controller using the freshly identified impulse response. For the transfer of this data from the system identification component to the ANC controller, three solutions are presented and discussed that make use of dynamic, partial reconfiguration techniques to efficiently time multiplex the available hardware resources.

6.2. Future Prospects

In general more acoustic tests to quantize the benefits of the additional processing power and the short delay in terms of noise reduction, but also stability need to be conducted. The proposed system could be applied to larger structures where additional channels are required or medium structures with increased sampling rates. Alternatively applications that focus on under-water structures could also be of interest, where ANC-systems benefit even more from shorter processing delay due to the higher sound propagation speed compared to air. The implementations for the ANC-systems provide unusual degrees of freedom, such as variable bit precisions, to shift the focus of the available processing power. Only first trends as a part of the validation process are provided, to demonstrate what combination of sampling rate, bit precisions, filter lengths and other parameters is most beneficial in terms of noise reduction performance and stability. More research in this area is required to validate the influence and limits of these parameters on ANC.

The here presented architectures are implemented independently of a specific vendor, platform or hardware accelerator. In case present FPGA platforms do not supply the desired performance for a specific ANC application, the implementations will very likely also be compatible with future generations of FPGAs that may provide more and faster hardware resources, as long as they provide synthesis tools that can read VHDL code.

Bibliography

- [Ana21] Analog Devices, Inc. *TigerSHARC Embedded Processor*. Rev. D. Apr. 2021. URL: <https://www.analog.com/media/en/technical-documentation/data-sheets/ADSP-TS101S.pdf>.
- [And98] Ray Andraka. “A survey of CORDIC algorithms for FPGA based computers”. In: *Proceedings of the 1998 ACM/SIGDA sixth international symposium on Field programmable gate arrays*. 1998, pp. 191–200.
- [Arm22] Arm Ltd. *Arm Architecture Reference Manual for A-profile architecture*. Feb. 2022. URL: <https://developer.arm.com/documentation/ddi0487/ha/>.
- [Asp22] Lars Asplund. *VUnit Homepage*. Feb. 2022. URL: <https://vunit.github.io/>.
- [Bai+13] Yuhui Bai et al. “FPGA vs DSP: A throughput and power efficiency comparison for Hierarchical Enumerative Coding”. In: *2013 IFIP/IEEE 21st International Conference on Very Large Scale Integration (VLSI-SoC)*. IEEE. 2013, pp. 318–321.
- [BLL02] Mingsian R. Bai, Yuanpei Lin, and Jienwen Lai. “Reduction of electronic delay in active noise control systems— A multirate signal processing approach”. In: *The Journal of the Acoustical Society of America* 111.2 (2002). DOI: [10.1121/1.1432980](https://doi.org/10.1121/1.1432980).
- [BN03] Martin Bouchard and Scott Norcross. “Computational load reduction of fast convergence algorithms for multichannel active noise control”. In: *Signal Processing* 83.1 (2003). DOI: [10.1016/S0165-1684\(02\)00382-1](https://doi.org/10.1016/S0165-1684(02)00382-1).
- [CD17] Jordan Cheer and Stephen Daley. “An Investigation of Delayless Sub-band Adaptive Filtering for Multi-Input Multi-Output Active Noise Control Applications”. In: *IEEE/ACM Transactions on Audio Speech and Language Processing* 25.2 (2017). DOI: [10.1109/TASLP.2016.2637298](https://doi.org/10.1109/TASLP.2016.2637298).

- [CT06] S. R. Chen and Gee-Pinn James Too. “Effects of sampling rate on ANC systems using LMS algorithm”. English. In: Institute of Noise Control Engineering of the USA - 35th International Congress and Exposition on Noise Control Engineering, INTER-NOISE 2006. Dec. 2006, pp. 2554–2561.
- [Dig16a] Digilent. *Nexys4 DDR FPGA Board Reference Manual*. Apr. 2016. URL: https://digilent.com/reference/_media/nexys4-ddr:nexys4ddr_rm.pdf.
- [Dig16b] Digilent. *PmodAD1 Reference Manual*. Apr. 2016. URL: <https://digilent.com/reference/pmod/pmodad1/reference-manual>.
- [Dig16c] Digilent. *PmodDA2 Reference Manual*. Apr. 2016. URL: <https://digilent.com/reference/pmod/pmodda2/reference-manual>.
- [Din97] Paulo SR Diniz. *Adaptive filtering*. Springer, 1997.
- [DKK21] William J. Dally, Stephen W. Keckler, and David B. Kirk. “Evolution of the Graphics Processing Unit (GPU)”. In: *IEEE Micro* 41.6 (2021), pp. 42–51. DOI: [10.1109/MM.2021.3113475](https://doi.org/10.1109/MM.2021.3113475).
- [dSP20] dSPACE GmbH. *MicroLabBox - Compact Prototyping Unit for the Laboratory*. Jan. 2020. URL: <https://www.dspace.com/de/gmb/home/products/hw/microlabbox.cfm>.
- [Eck+18] Marcel Eckert et al. “Low-Latency FIR Filter Structures Targeting FPGA Platforms”. In: *Proceedings of the 9th International Symposium on Highly-Efficient Accelerators and Reconfigurable Technologies*. University of Toronto, 2018.
- [Eck+22] Marcel Eckert et al. “Transferring Run-Time-Data Between Distinct FPGA Designs - Solutions in the Context of an ANC-Application”. In: *IECON 2022-48th Annual Conference of the IEEE Industrial Technology Society*. 2022.
- [Eli00] S. Elliott. *Signal Processing for Active Control*. Signal Processing and its Applications. Elsevier Science, 2000.
- [EMK19] Marcel Eckert, Dominik Meyer, and Bernd Klauer. “Context save and restore of partial reconfiguration regions for xilinx fpgas”. In: *2019 14th International Symposium on Reconfigurable Communication-centric Systems-on-Chip (ReCoSoC)*. IEEE. 2019, pp. 5–12.

-
- [EN93] S.J. Elliott and P.A. Nelson. “Active noise control”. In: *IEEE Signal Processing Magazine* 10.4 (1993), pp. 12–35. DOI: [10.1109/79.248551](https://doi.org/10.1109/79.248551).
- [Far00] Angelo Farina. “Simultaneous Measurement of Impulse Response and Distortion with a Swept-Sine Technique”. In: *Audio Engineering Society Convention 108*. Feb. 2000.
- [Gol+69] Bernard Gold et al. *Digital Processing of Signals*. Lincoln Laboratory publications. McGraw-Hill, 1969.
- [Haj+18] Amir HajiRassouliha et al. “Suitability of recent hardware accelerators (DSPs, FPGAs, and GPUs) for computer vision and image processing algorithms”. In: *Signal Processing: Image Communication* 68 (2018), pp. 101–119. DOI: <https://doi.org/10.1016/j.image.2018.07.007>. URL: <https://www.sciencedirect.com/science/article/pii/S0923596518303606>.
- [Han+18] Jonas Hanselka et al. “Experimental Results of the Effect of Increased Filter Length and Sample Rate of a Feedback Active Noise Control System with the FxLMS-Algorithm implemented in VHDL”. In: *INTER-NOISE and NOISE-CON Congress and Conference Proceedings*. 2018.
- [Han20] Jonas Hanselka. “Adaptive Lärminderung an einem teilgeöffneten Fenster mittels mehrdimensionaler Regelung-Konstruktive und algorithmische Optimierung und Umsetzung als Hardware-Schaltung”. PhD thesis. Helmut-Schmidt-Universität/Universität der Bundeswehr Hamburg, 2020. URL: <https://openhsu.ub.hsu-hh.de/handle/10.24405/9924>.
- [HCZ09] Martin Holters, Tobias Corbach, and Udo Zölzer. “Impulse Response Measurement Techniques and their Applicability in the Real World”. In: *Proc. of the 12th Int. Conference on Digital Audio Effects (DAFx-09)*. Sept. 2009. URL: https://www.dafx.de/paper-archive/2009/papers/paper_26.pdf.
- [HDB20] Don Lahiru Nirmal Hettiarachchi, Venkata Salini Priyamvada Davuluru, and Eric J. Balster. “Integer vs. Floating-Point Processing on Modern FPGA Technology”. In: *2020 10th Annual Computing and Communication Workshop and Conference (CCWC)*. 2020, pp. 0606–0612. DOI: [10.1109/CCWC47524.2020.9031118](https://doi.org/10.1109/CCWC47524.2020.9031118).

- [Hoz18] Emil Hozan. *Understanding the Layers of a Computer System*. Sept. 2018. URL: <https://www.secplicity.org/2018/09/19/understanding-the-layers-of-a-computer-system/>.
- [IEE85] IEEE. “IEEE Standard for Binary Floating-Point Arithmetic”. In: *ANSI/IEEE Std 754-1985* (1985), pp. 1–20. DOI: [10.1109/IEEESTD.1985.82928](https://doi.org/10.1109/IEEESTD.1985.82928).
- [KB13] F. Kesel and R. Bartholomä. *Entwurf von digitalen Schaltungen und Systemen mit HDLs und FPGAs: Einführung mit VHDL und SystemC*. Grundlagen der Elektro- und Informationstechnik. De Gruyter, 2013.
- [K GK12] Yoshinobu Kajikawa, Woon Seng Gan, and Sen M. Kuo. “Recent advances on active noise control: Open issues and innovative applications”. In: *Journal of Institutional Economics* 1.2 (2012). DOI: [10.1017/ATSIP.2012.4](https://doi.org/10.1017/ATSIP.2012.4).
- [KH98] Colin D Kestell and Colin H Hansen. “An overview of active noise control”. In: *Safety in Action* (1998).
- [Kle+19] Alexander Klemd et al. “A Parameterizable Feedback FxLMS Architecture for FPGA Platforms”. In: *Proceedings of the 10th International Symposium on Highly-Efficient Accelerators and Reconfigurable Technologies*. 2019.
- [Kle+21a] Alexander Klemd et al. “A Flexible Multi-Channel Feedback FxLMS Architecture for FPGA Platforms”. In: *International Conference on Field-Programmable Logic and Applications (FPL)*. 2021.
- [Kle+21b] Alexander Klemd et al. “Exponential Sine Sweep Measurement Implementation Targeting FPGA Platforms”. In: *International Conference on Field-Programmable Technology (FPT)*. 2021.
- [KM95] Sen M. Kuo and Dennis Morgan. *Active noise control systems: algorithms and DSP implementations*. John Wiley & Sons, Inc., 1995.
- [KP98] H. S. Kim and Y. Park. “Delayed-X LMS algorithm: An efficient ANC algorithm utilizing robustness of cancellation path model”. In: *Journal of Sound and Vibration* 212.5 (1998). DOI: [10.1006/jsvi.1997.1484](https://doi.org/10.1006/jsvi.1997.1484).
- [KR10] Ian Kuon and Jonathan Rose. *Quantifying and exploring the gap between FPGAs and ASICs*. Springer Science & Business Media, 2010.

-
- [Lee+21] Hsiao Mun Lee et al. “A review of the application of active noise control technologies on windows: Challenges and limitations”. In: *Applied Acoustics* 174 (2021), p. 107753. DOI: <https://doi.org/10.1016/j.apacoust.2020.107753>. URL: <https://www.sciencedirect.com/science/article/pii/S0003682X20308574>.
- [Len+18] George Lentaris et al. “High-performance embedded computing in space: Evaluation of platforms for vision-based navigation”. In: *Journal of Aerospace Information Systems* 15.4 (2018), pp. 178–192.
- [LHS18] Max Lorenzen, Jonas Hanselka, and Delf Sachau. “Simulative study on the effect of the increase of the sample rate of a feedback active noise control system”. In: *INTER-NOISE and NOISE-CON Congress and Conference Proceedings*. 2018.
- [Lor+14a] Jorge Lorente et al. “GPU based implementation of multichannel adaptive room equalization”. In: *2014 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE. 2014, pp. 7535–7539.
- [Lor+14b] Jorge Lorente et al. “GPU Implementation of Multichannel Adaptive Algorithms for Local Active Noise Control”. In: *IEEE/ACM Transactions on Audio, Speech, and Language Processing* 22.11 (2014), pp. 1624–1635. DOI: [10.1109/TASLP.2014.2344852](https://doi.org/10.1109/TASLP.2014.2344852).
- [MA95] K.A. Mayyas and T. Aboulnasr. “Leaky LMS: a detailed analysis”. In: *Proceedings of ISCAS’95 - International Symposium on Circuits and Systems*. Vol. 2. 1995, 1255–1258 vol.2. DOI: [10.1109/ISCAS.1995.520373](https://doi.org/10.1109/ISCAS.1995.520373).
- [Mat] Mathworks, Inc. *Rounding Modes for Fixed-Point Simulink Blocks*. URL: <https://www.mathworks.com/help/fixedpoint/ug/rounding-modes-for-fixed-point-simulink-blocks.html>.
- [Men+19] Diego Mendez et al. “Parallel Architecture of Reconfigurable Hardware for Massive Output Active Noise Control”. In: *Parallel Processing Letters* 29.3 (2019). DOI: [10.1142/S0129626419500142](https://doi.org/10.1142/S0129626419500142).
- [Mey14] Uwe Meyer-Bäse. *Digital signal processing with field programmable gate arrays*. 4th ed. Springer, 2014.

- [MFR20] Roberto Millón, Emmanuel Frati, and Enzo Rucci. “A Comparative Study between HLS and HDL on SoC for Image Processing Applications”. In: *Elektron* 4.2 (Dec. 2020), pp. 100–106. DOI: [10.37537/rev_elektron.4.2.117.2020](https://doi.org/10.37537/rev_elektron.4.2.117.2020). URL: http://dx.doi.org/10.37537/rev_elektron.4.2.117.2020.
- [MH00] George Moschytz and Markus Hofbauer. *Adaptive Filter*. Springer, 2000.
- [Nur+16] Eriko Nurvitadhi et al. “Accelerating recurrent neural networks in analytics servers: Comparison of FPGA, CPU, GPU, and ASIC”. In: *2016 26th International Conference on Field Programmable Logic and Applications (FPL)*. 2016, pp. 1–4. DOI: [10.1109/FPL.2016.7577314](https://doi.org/10.1109/FPL.2016.7577314).
- [Ok117] Vojin G Oklobdzija. *Digital design and fabrication*. CRC press, 2017.
- [RG67] C.M. Rader and B. Gold. “Effects of parameter quantization on the poles of a digital filter”. In: *Proceedings of the IEEE* 55.5 (1967), pp. 688–689. DOI: [10.1109/PROC.1967.5634](https://doi.org/10.1109/PROC.1967.5634).
- [Shi+17] Dongyuan Shi et al. “Multiple parallel branch with folding architecture for multichannel filtered-x least mean square algorithm”. In: *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE. 2017, pp. 1188–1192.
- [Shi+19] Dongyuan Shi et al. “Practical implementation of multichannel filtered-x least mean square algorithm based on the multiple-parallel-branch with folding architecture for large-scale active noise control”. In: *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 28.4 (2019), pp. 940–953.
- [SN06] MS Safadi and DL Ndzi. “Digital hardware choices for software radio (SDR) baseband implementation”. In: *2006 2nd International Conference on Information & Communication Technologies*. Vol. 2. IEEE. 2006, pp. 2623–2628.
- [SSG16] Dongyuan Shi, Chuang Shi, and Woon-Seng Gan. “A systolic FxLMS structure for implementation of feedforward active noise control on FPGA”. In: *2016 Asia-Pacific Signal and Information Processing Association Annual Summit and Conference (APSIPA)*. IEEE. 2016, pp. 1–6.

-
- [SUG19] S Santhi, E Udayakumar, and T Gowthaman. “Design and Implementation of an Area-and Delay-Efficient FxLMS Filter for Active Noise Cancellation”. In: *Computational Intelligence and Sustainable Systems*. Springer, 2019, pp. 115–129.
- [Tex21] Texas Instruments. *Industry-Standard Dual Operational Amplifiers*. July 2021. URL: https://www.ti.com/lit/ds/symlink/lm358.pdf?ts=1644864904082&ref_url=https%253A%252F%252Fwww.ti.com%252Fproduct%252FLM358.
- [Tim+20] Johannes Timmermann et al. “Validation and performance analysis of a parameterizable normalized feedback FxLMS architecture for FPGA platforms”. In: *INTER-NOISE and NOISE-CON Congress and Conference Proceedings*. 2020.
- [Ush16] I V Ushenina. “FPGA-based Multi-Channel Adaptive FXLMS Filter Implemented as an Array of Processing Blocks”. In: *Technical Acoustics* 1 (2016).
- [Xil15] Xilinx Inc. *Configuration Readback Capture in UltraScale FPGAs*. XAPP1230. Rev. v1.1. Nov. 2015. URL: https://www.xilinx.com/support/documentation/application_notes/xapp1230-configuration-readback-capture.pdf.
- [Xil16] Xilinx, Inc. *7 Series FPGAs Configurable Logic Block - User Guide*. Rev. 1.8. Sept. 2016. URL: https://www.xilinx.com/support/user_guides/ug474_7Series_CLB.pdf.
- [Xil18] Xilinx, Inc. *7 Series DSP48E1 Slice - User Guide*. Rev. 1.10. Mar. 2018. URL: https://www.xilinx.com/support/user_guides/ug479_7Series_DSP48E1.pdf.
- [Xil19] Xilinx, Inc. *7 Series FPGAs Memory Resources - User Guide*. Rev. 1.14. July 2019. URL: https://www.xilinx.com/support/documentation/user_guides/ug473_7Series_Memory_Resources.pdf.
- [Xil20] Xilinx, Inc. *Model-Based DSP Design Using System Generator*. Rev. v2020.1. June 2020. URL: https://www.xilinx.com/content/dam/xilinx/support/documentation/sw_manuals/xilinx2020_1/ug948-vivado-sysgen-tutorial.pdf.
- [Zöl89] Udo Zölzer. “Entwurf Digitaler Filter Für die Anwendung im Tonstudiobereich”. PhD thesis. Hamburg, Germany: Hamburg University of Technology, June 1989.

A. Appendix

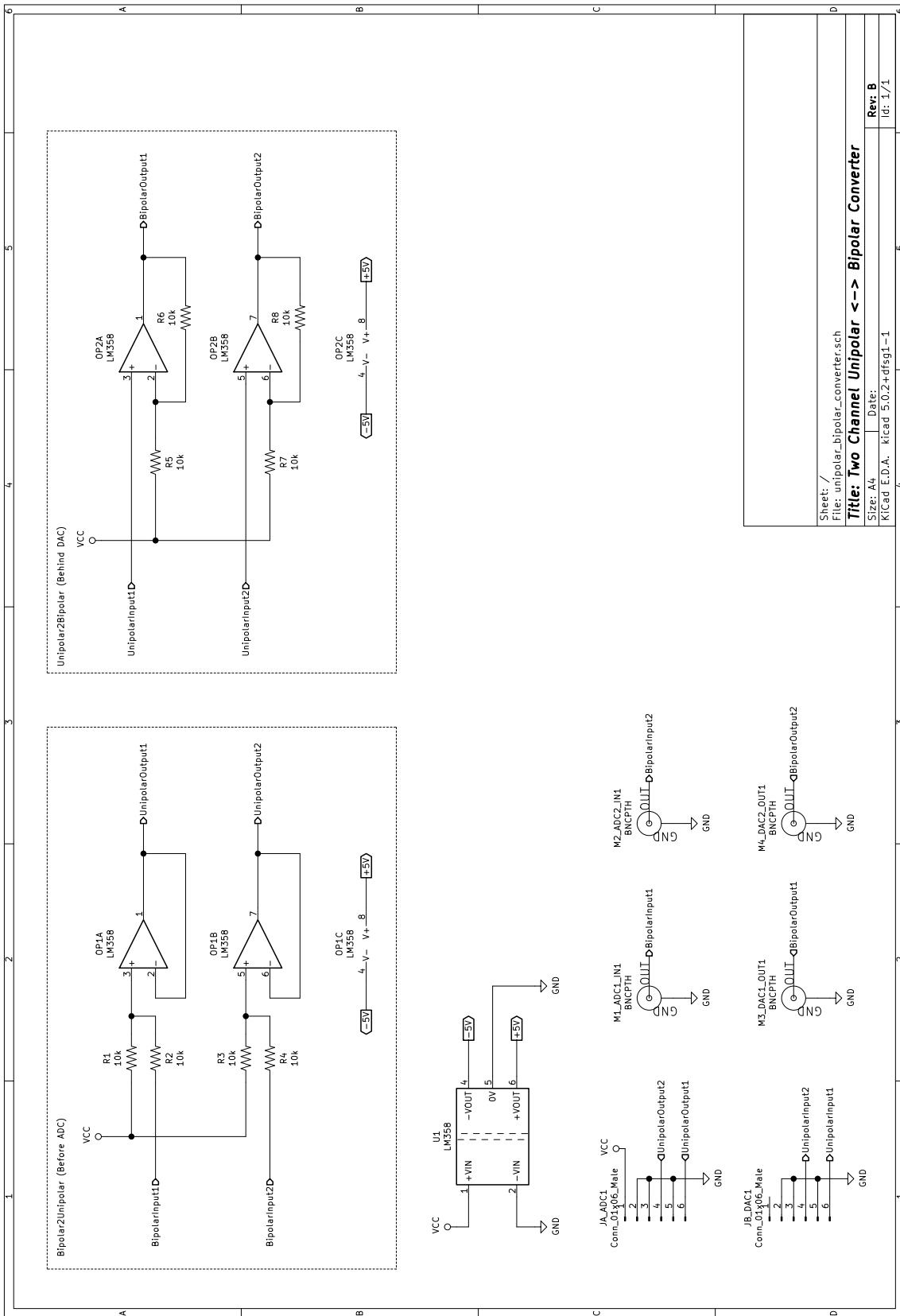
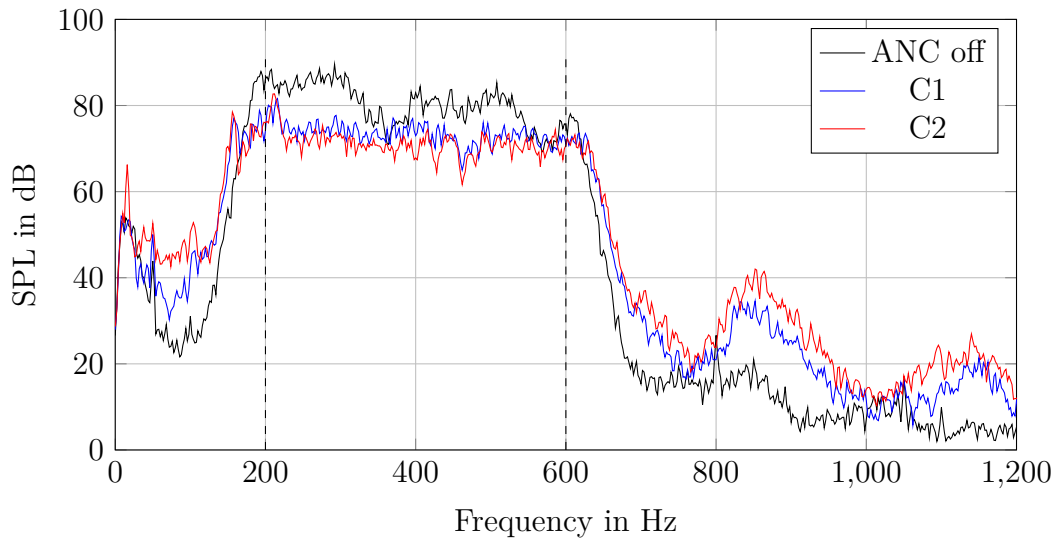
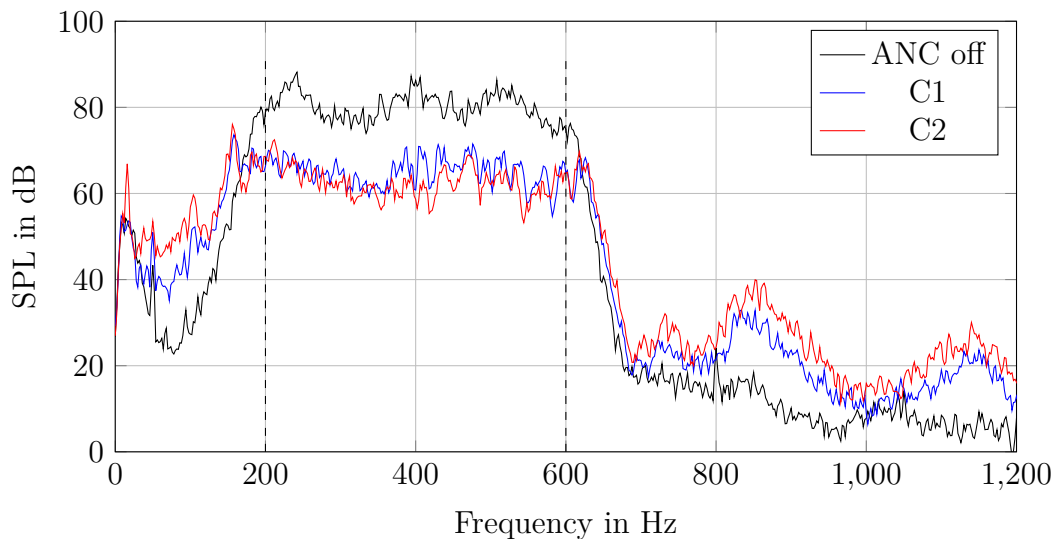


Figure A.1.: Schematic of the board used to convert between unipolar and bipolar signals.



(a) shows the first stage of the 2×2 feedback FxLMS.



(b) shows the second stage of the 2×2 feedback FxLMS.

Figure A.2.: The noise reduction performance of the multi-channel feedback FxLMS applied to a duct. The two stages are placed in a row, thus the second stage yields a better noise reduction performance. The second configuration uses a reduced number of bit precision for the I/O signals and filter coefficients. Both configurations use filter lengths of 4097 and a sampling rate of 160 kHz. Configuration C1 uses $\text{bw}[e(n)] = 16$ bit, $\text{bw}[y(n)] = 16$ bit, $\text{bw}[\hat{s}] = 32$ bit, $\text{bw}[w(n)] = 32$ bit. Configuration C2 uses $\text{bw}[e(n)] = 13$ bit, $\text{bw}[y(n)] = 13$ bit, $\text{bw}[\hat{s}] = 16$ bit, $\text{bw}[w(n)] = 28$ bit. The remaining parameters are the same as the configurations in table 5.4.

Acronyms

ADC	Analog-to-Digital Converter
AEGP	Anderson-Earle-Goldschmidt-Powers
ANC	Active Noise Control
ASIC	Application Specific Integrated Circuit
BRAM	Block-RAM
CLB	Configurable Logic Block
CORDIC	Coordinate Rotation Digital Computer
CPLD	Complex Programmable Logic Device
CPU	Central Processing Unit
CSR	Context Save Restore
DAC	Digital-to-Analog Converter
DPR	Dynamic Partial Reconfiguration
DSP	Digital Signal Processing
FFT	Fast Fourier Transform
FIR	Finite Impulse Response
FLOPS	Floating-Point Operations per Second
FPGA	Field Programmable Gate Array
FxLMS	Filtered Reference Least Mean Square
GPP	General Purpose Processor
GPU	Graphics Processing Unit
HDL	Hardware Description Language
HLS	High Level Synthesis
I/O	Input/Output
IC	Integrated Circuit
IP	Intellectual Property
LMS	Least Mean Squares
LUT	Look-Up-Table

MAC	Multiply-Accumulate
MIMO	Multiple-Input and Multiple-Output
MSB	Most Significant Bit
NRE	Non-Recurring Engineering
PIP	Programmable Interconnect Point
PLA	Programmable Logic Arrays
PLD	Programmable Logic Device
PLL	Phase-Locked Loop
RAM	Random-Access Memory
RLS	Recursive Least Squares
ROM	Read Only Memory
RTL	Register-Transfer Level
SIMD	Single Instruction-Multiple Data
SISO	Single-Input and Single-Output
SNR	Signal-to-Noise Ratio
SPL	Sound Pressure Level
SRAM	Static Random-Access Memory
TDP	True Dual-Port (Mode)
UART	Universal Asynchronous Receiver-Transmitter
VHDL	Very High Speed Integrated Circuit HDL

List of Latin Symbols

A	Amplitude of sine sweeps
$A_{\text{inv}}(n)$	Time-mirrored amplitude correction factor
A_R	Gain caused by R cordic iterations
$d(n)$	Desired signal
$e(n)$	Error signal measured at microphone
f_{clk}	Clock frequency
f_s	Sampling rate
J	Number of reference sensors
j	Index of the reference sensor
K	Number of secondary sources
k	Index of the secondary source
K_{AEGP}	Number of iterations for the AEGP algorithm
K_r	Recursive gain per CORDIC iteration
L_{adapt}	Filter order of the adaptive FIR-filters
L_{static}	Filter order of the static FIR-filters
L_x	Desired sweep length in samples
M	Number of error sensors
N_{fade}	Number of samples that will be attenuated by the amplitude fade function
m	Index of the error sensor
P_{min}	Minimal allowed power estimation of the reference signal
$\hat{P}_x(n)$	Power estimation of the reference signal
\hat{s}	Coefficients of the static FIR-filters
$S(z)$	Secondary-path transfer function
$\hat{S}(z)$	Estimated secondary-path transfer function
$w(n)$	Coefficients / weights of the adaptive FIR-filters
$W(z)$	Primary-path transfer function
$x(n)$	Reference signal
$x'(n)$	Filtered reference signal
x_r	Transformed x-coordinate of CORDIC
$\tilde{x}(n)$	Precalculated part of the filtered reference signal
$y(n)$	Output signal
$y'(n)$	Output signal measured at the error sensor
$\hat{y}'(n)$	Filtered output signal

y_r	Transformed y-coordinate of CORDIC
$\tilde{y}(n)$	Precalculated part of the output signal
z_r	Remaining desired rotation angle for the CORDIC

List of Greek Symbols

α	Normalized step size of LMS
β	Smoothing parameter of normalized LMS
δ	Dirac unit impulse
γ	Weighting factor of normalized LMS
μ	Step size of LMS
Ω	Angular frequency of sine sweep
ν	Leakage factor of LMS
ϕ_0	Initial phase of oscillator
$\psi(n)$	Recursive factor for the amplitude fade function
θ	Factor of the oscillators angular frequency increase

List of Figures

2.1.	Feedback-based active noise control using a digital controller.	6
2.2.	Block diagram of the feedback FxLMS SISO algorithm	7
2.3.	Block diagram of a feedback multi-channel ANC-system using the FxLMS algorithm	10
2.4.	Block diagram of the system identification process using an adaptive filter	12
2.5.	Block diagram of the system identification process using exponential sine sweeps	15
2.6.	Giga-MAC-operations per second to be performed by the feedback FxLMS algorithm with increasing I/O channels	17
3.1.	Classification of integrated circuits based on fabrication technology	22
3.2.	A simplified schematic of the hardware resource arrangement on an FPGA chip.	25
3.3.	A simplified schematic of a CLB containing two slices of different types	26
3.4.	The architecture of the DSP-slice <i>DSP48E1</i> used in the <i>7 series</i> FPGA from <i>Xilinx</i>	27
3.5.	Layers of abstraction in computer systems	31
4.1.	A systolic architecture with $K \times J$ modules presented by Ushenina	35
4.2.	Proposed hardware architecture of a multi-channel feedforward FxLMS	37
4.3.	Algorithm of rounding mode <i>Nearest</i> to convert a Q15 number to a Q7 format.	43
4.4.	Example for resizing a Q15.0 format to Q7.0 format	43
4.5.	Hardware architecture of a static FIR filter	45
4.6.	Hardware architecture of an adaptive FIR filter	48
4.7.	The hardware architecture of the LMS update component's datapath	49
4.8.	The datapath of the top-level feedback FxLMS implementation	50
4.9.	Control flow diagram of the FxLMS SISO architecture	52
4.10.	Used number formats for the feedback FxLMS implementation	55
4.11.	The datapath for the normalization of the step size and the update of the leakage factor	56
4.12.	Hardware architecture of the multi-channel adaptive FIR filter using a generic number of multiplier lanes	61
4.13.	The datapath of the LMS update component for the multi-channel FxLMS architecture	62

4.14. The datapath of the top-level architecture for the multi-channel feedback FxLMS	63
4.15. The top-level datapath of the system identification component using exponential sine sweeps.	66
4.16. The datapath of the component that generates the forward and reverse exponential sine sweeps including the CORDIC processor in rotational mode.	68
4.18. The feedback FxLMS and system identification divided by static and dynamic resources	73
5.1. The experimental setup using a duct.	87
5.2. The experimental setup for the multi-channel FxLMS variant using an active headrest	88
5.3. Comparison of the software-based configuration SS1 against the equivalent hardware configuration SH1	89
5.4. Comparison of configuration SH1 to SH4 with the achieved noise reduction	89
5.5. Comparison of the software-based configuration MS1 against the equivalent hardware configuration MH1	90
5.6. Comparison of configuration MH1 to MH4 with the achieved noise reduction	91
5.7. Comparison of the generated exponential sine sweeps on the processor and the FPGA	93
5.8. Comparison of the measured impulse responses of the software and hardware implementation	94
5.9. Comparison of the LMS-based system identification hardware implementation against the system identification component using sine sweeps	95
A.1. Schematic of the board used to convert between unipolar and bipolar signals.	110
A.2. The noise reduction performance of the multi-channel feedback FxLMS applied to a duct with reduced bit-precisions	111

List of Tables

1.1. Simulation results that show the normalized error signal power over a series of simulations with varying sampling frequencies and filter lengths	2
3.1. Summary of the logic resources in one CLB	26
3.2. Maximum port aspect ratios for <i>RAMB18E1</i> in TDP-mode	28
3.3. Maximum port aspect ratios for <i>RAMB36E1</i> in TDP-mode	28
4.1. Properties of signed und unsigned fixed-point numbers in Q-notation	41
4.2. Fixed-point formats of the corresponding arithmetic operation	42
5.1. FPGA-types that are used in the following experiments and their available resources.	78
5.2. Synthesized configurations for the single-channel feedback FxLMS architecture	80
5.3. Synthesis results of the presented configurations performed on the DS1202 platform	81
5.4. A parameter overview of the multi-channel FxLMS configurations . . .	82
5.5. Synthesis results of the presented multi-channel configurations performed on two different platforms	83
5.9. The utilized hardware resources for the presented configurations . . .	92
5.10. Chosen configuration of the FxLMS component that maximizes the resource utilization of the underlying hardware platform	96
5.11. The synthesis result for the chosen configurations	96
5.12. The resource utilization divided for the dynamic and static domains of the FxLMS controller with solution II	97

Curriculum Vitae

Due to privacy reasons, this CV is redacted in the published version.

Der Lebenslauf wird aus Gründen des Datenschutzes in der elektronischen Fassung der Arbeit nicht veröffentlicht.