A Parameterizable Feedback FxLMS Architecture for FPGA Platforms

ABSTRACT

In active noise control (ANC) systems the filtered-x least mean squares (FxLMS) algorithm is the most widely used reduction algorithm. The feedback FxLMS can operate without an external reference signal but has the disadvantage of being very sensible to delays in the secondary path. As the processing latency of the FxLMS itself adds to the secondary path delay, it is critical to minimize the processing latency. Large-scale ANC systems benefit from high order filters and high sample rates. This combination is very computational demanding. Specific hardware implementations can achieve more complex filter configurations compared to software implementations on general purpose processors, but are less flexible to changes.

This paper presents a parameterizable hardware design implemented in very-high-speed integrated circuit hardware description language (VHDL) that can operate at different bit widths and can be configured to optimize for high sample rates, long filter lengths or low hardware resource usage. The design is able to provide a constant processing latency of only 3 clock cycles that is independent from the filter length. Various configurations of this filter design are synthesized to a Xilinx Kintex7 XC7K325T field-programmable gate array (FPGA) with filter lengths up to 131072 adaptive coefficients or sample rates up to 4.5 MHz. The respective hardware utilization is evaluated.

Keywords

Feedback FxLMS, Hardware Design, VHDL, FPGA

1. INTRODUCTION

Digitally implemented ANC applications mainly use the FxLMS algorithm to calculate the driving signal for the anti-noise sources due to its simplicity and robustness. The feedforward FxLMS algorithm needs a reference signal from the noise source to be executed. In ANC applications where it is impossible to get a reference signal from the noise source a feedback ANC system can be used which estimates the reference signal via internal model control [10]. The stability of the feedback algorithm is mainly affected by the correctness of these models $\widehat{S}(z)$ and the timespan between the change of the noise at the error sensor and the respective change at the anti-noise source. As a result short processing delays are required. At the same time large-scale ANC applications aiming for offices or sleeping rooms require long filter lengths for the respective internal model of the secondary path.

Simulations in [13] showed that sample rates well above 48 kHz are beneficial to noise reduction in the audible spectrum and that with increasing sample rates it is also essential to further increase the static and adaptive filter length accordingly. Since long filter lengths contradict with high sample rates or low processing delays, a compromise needs to be found. The optimal compromise for each ANC scenario varies, thus the filter design also needs to be easily adjustable. In time domain the filtering of a signal with a finite impulse response (FIR) filter is a convolution, which means a lot of multiply-accumulate (MAC) operations which can be done in parallel. Thus FPGA are the ideal platforms to implement low delay and high order feedback FxLMS filters enabling high sample rates.

The hardware design for the FPGA was implemented purely using VHDL, although for the very common static FIR filter a variety of intellectual property (IP)-cores exist. The choice to use VHDL was made to have a more fine-grained control over the hardware design and to have a filter design that is independent from proprietary software or a specific hardware platform. The two largest FPGA producers Xilinx and Intel-Altera offer parameterizable IP-cores for static FIR filters. Xilinx limits the data and coefficient bit widths to 49 bit and the filter length to 2048 coefficients. Intel-Altera limits the data and coefficient bit widths to 32 bit. An IP-core for an adaptive filter using the least mean squares (LMS) algorithm by Xilinx or Intel-Altera was not found [21] [8].

A study of publications from the years 2005-2018 shows other researches presented hardware implementations for ANC applications using LMS or FxLMS filters. Many filter architectures were found using either a fully sequential or parallel approach for the arithmetic operations and thus are quickly limited in their sample rate or filter lengths.

Most LMS filters show filter lengths between 4-256 and sample rates between 1-100 kHz [2, 4, 7, 9, 14, 19]. Bahoura et al. presents a sequential architecture with a filter length of 1024 sampling at 50/60 Hz [3]. Parallel architectures achieve high sample rates in the 10-100 MHz range, but the respective filter length of 4-64 is rather short [5, 6, 11, 17, 19]. Feedworward- and feedback-FxLMS architectures are found in [12, 20] using 4-24 static and adaptive coefficients with sample rates between 24-192 kHz. The architecture of Shi et al. [18] is the only architecture of a FxLMS filter found

This work was presented in part at the international symposium on Highly-Efficient Accelerators and Reconfigurable Technologies (HEART2019) Nagasaki, JP, June 6–7, 2019.

to provide a constant processing delay independent of the filter length. They propose a systolic architecture of a feedforward FxLMS simulated in Matlab Simulink with different filter parameters. This architecture has a constant low processing delay and allows for very high sample rates. A MAC-component is needed for each additional filter tap.
Rivera Benois et al. [15] present a feedforward and feedback FxLMS in one design with 2048 adaptive and static filter coefficients processing at a sample rate of 48 kHz. The combination of filter length and sample rate suggests some sort of mixed architecture, but cannot be further investigated as the architecture of the feedback part was not provided. The design is graphically programmed using Matlab Simulink and Xilinx System Generator. Since this design provides the highest found combination of filter taps and maximum sample rate, a comparison regarding filter parameters and sample rate is made in the results and evaluation.
All here mentioned designs are aimed at one specific ANC problem and are not parameterizable.

2. ARCHITECTURE

The proposed architecture implements the feedback FxLMS as it is described in [10] without any leakage factor or step size normalization. It is implemented as a single input single output system as shown in Fig.1. The area without gray

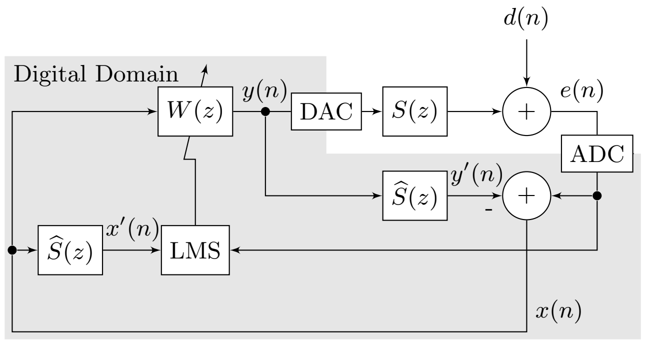

Figure 1: Block diagram of the feedback single input single output ANC-system.

background shows the analog domain where the error sensor measures the error signal $e(n)$. The system $S(z)$ represents the signal transformation between the input of the digital-to-analog converter (DAC) and the output of the analog-to-digital converter (ADC). The error signal is composed of the external disturbance signal $d(n)$ and the signal of the anti-noise source driven by the FxLMS output $y(n)$. The digital components are embedded in the gray area and represent the VHDL design. The VHDL design can be divided into three parts. The LMS filter including the coefficient adaption does the calculation of $y(n)$. Both FIR filters $\widehat{S}(z)$ are merged into one component calculating $x'(n)$ and $y'(n)$. The top-level component handles the communication with the ADC, DAC and filters. Also it performs the calculation of $x(n) = e(n) - y'(n-1)$. The architecture presented here is an extended development of the design presented in [1].

2.1 The LMS Component

The LMS algorithm described in [10] with a filter length of L_{LMS} first does the convolution of the samples x and the adaptive coefficients w

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

and then updates the filter coefficients

$$w_l(n+1) = w_l(n) - \mu x'(n-l)e(n), \quad l = 0, 1, ..., L_{\text{LMS}}-1. \tag{2}$$

To enable a constant processing delay for the filter output, the proposed implementation of the LMS does the same calculations in a different order. When a new sample $x(n)$ arrives, the LMS component calculates

$$y(n) = w_0(n)\, x(n) + \widetilde{y}(n). \tag{3}$$

in one clock cycle. Concurrently the part of the convolution of $y(n)$ that is independent of the follow-up sample $x(n+1)$

$$\widetilde{y}(n+1) = \sum_{l=1}^{L_{\text{LMS}}-1} w_l(n+1)\, x(n+1-l) \tag{4}$$

is pre-calculated. Note the adaptive filter coefficients need to be updated before as stated in eq. (2). This strategy for a constant processing delay independent of the filter length is proposed in [16] for a single static FIR filter.
The architecture of the LMS component is shown in Fig.2. Please note that the LMS component as described here merges

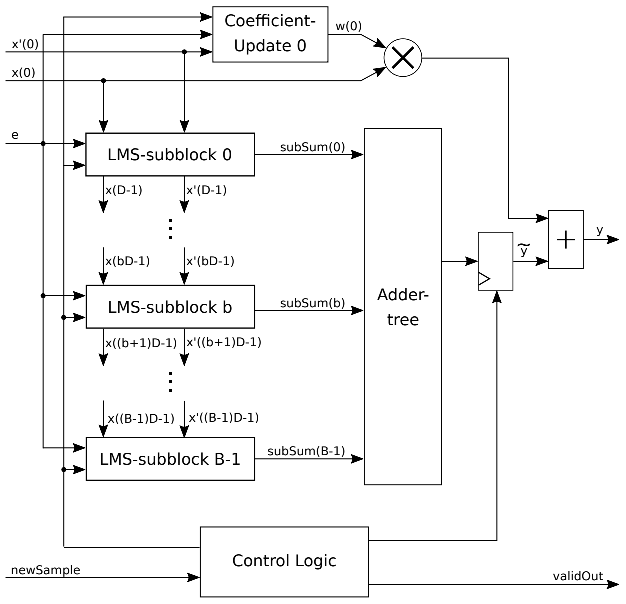

Figure 2: The architecture of the LMS component.

the LMS update-algorithm and the filter $W(z)$ into one component. To achieve a mixed sequential and parallel architecture, the LMS filter includes an array of components named LMS-subblock. The LMS-subblocks split the filter length L_{LMS} into individual LMS units doing the calculations concurrently. Assuming a LMS-subblock count of B means that the respective length of each LMS-subblock is $D = (L_{\text{LMS}} - 1)/B$. Hence each LMS-subblock with index $b = 0, 1, ..., B-1$ updates

$$w_{bD+d+1}(n+1) = w_{bD+d+1}(n) - \mu\, e(n)\, x'(n-bD-d), \quad d = 0, 1, ..., D-1 \tag{5}$$

for its internal coefficients and calculates a sub-sum of

$$\text{subsum}_b = \sum_{d=0}^{D-1} x(n - bD - d + 1)\, w_{bD+d+1}(n). \qquad (6)$$

Fig.3 shows the architecture of a LMS-subblock. The array

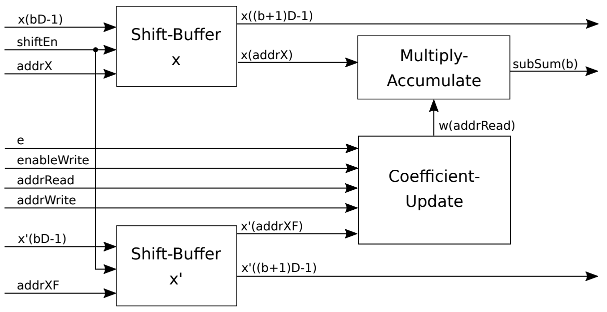

Figure 3: The architecture of a LMS-subblock.

of data samples $x(n)$ and the filtered data samples $x'(n)$ are stored in block RAM based shift buffers. When new samples arrive they are shifted into the shift buffers of the first LMS-subblock. The oldest sample in the shift buffer is then transferred into the shift buffer of the following LMS-subblock. The shift buffers need two clock cycles to perform the shifting of the samples and to output the first sample. Note that the shift buffer of x' is one sample behind the shift buffer of x, because the coefficient-update term in eq. (5) requires the filtered sample with index $x'(n - bD - d)$ while the sub-sum term in eq. (6) requires the unfiltered sample with index $x(n - bD - d + 1)$.
Next the control logic initiates the sequential update of the coefficients. At each clock cycle, a coefficient and the filtered sample of the same index is loaded into the coefficient-update component, see eq. (5). The coefficient update has a two-stage pipeline calculating the filter coefficient step in the first stage and subtract the step from the old coefficient value in the second stage, see Fig.4. The coefficients are loaded and stored in dual-port block RAM. The coefficient-update component needs $D + 2$ clock cycles to update its filter coefficients. To avoid an additional multiplication and

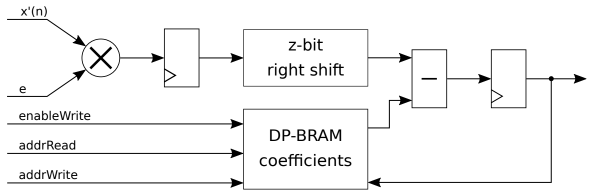

Figure 4: The coefficient-update component.

to save resources the step size μ is implemented in VHDL as a right bit shift of z-bits. Hence the step size value $\mu = 2^{-z}$ in the presented design can only be set to a power of 2. The MAC component also has a two-stage pipelined architecture, as shown in Fig.5, that first does the multiplication of the sample and the coefficient. This product is added to the register for the accumulating sum in the second stage, see eq. (6).
The MAC component is processing the subsum concurrently to the coefficient-update component, but starts delayed to it once the first filter coefficient is updated. In

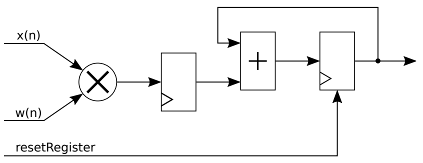

Figure 5: The multiply-accumulate component.

total the LMS-subblock uses two clock cycles for the shift buffers, two clock cycles to begin the coefficient update, one clock cycle to do the multiplication in the MAC component and D clock cycles to accumulate the susbum. Hence the LMS-subblock takes $D+5$ clock cycles to do the calculations in (5) and (6). The addertree needs $log_2(B)$ clock cycles to add the subsums to one sum. Therefore the LMS component needs

$$N_{\text{LMS}} = \frac{L_{\text{LMS}} - 1}{B} + log_2(B) + 5. \qquad (7)$$

clock cycles to obtain the sum $\widetilde{y}(n+1)$ and thus be ready to process a new sample $x(n+1)$. Note that the pre-calculation of $\widetilde{y}(n + 1)$ provides a constant low processing delay, but does not decrease the minimal processing time between new samples.

2.2 The FIR Component

The architecture of the static FIR filters ($\widehat{S}(z)$) uses a very similar architecture compared to the LMS. Like the proposed LMS the implementation of the FIR also uses a configurable array of parallel subblocks that sequentially accumulates subsums of the filter output and a following pipelined addertree. Each FIR-subblock contains one shift buffer for the samples and a MAC component. The coefficient-update component is replaced by read-only memory (ROM) for the static filter coefficients s built in block RAM. Using the same strategy as described for the LMS, the first FIR filter calculates the filter output

$$x'(n) = s_0\, x(n) + \widetilde{x}'(n) \qquad (8)$$

in one clock cycle and the pre-calculation for the upcoming sample

$$\widetilde{x}'(n + 1) = \sum_{m=1}^{L_{\text{FIR}}-1} s_m\, x(n + 1 - m). \qquad (9)$$

The second FIR filter calculates $y'(n)$ respectively. The two instances of the FIR filter are further optimized to save hardware resources and enable higher filter configurations. Both FIR filters share one control logic component and the static coefficient ROMs. The static coefficient ROM is possible to share as both FIR filters emulate the same secondary path and only the samples in the buffers vary. Both filters also have the same filter length L_{FIR}. As a consequence of using the same control logic, the FIR filters have to do their calculations simultaneously.

2.3 The Top-Level Component

The internal signals of the FxLMS use fixed-point number representations of different precision depending on the input/output (I/O) data word width. Every multiplication and addition causes additional bits in the result. Therefore the filter outputs of the underlying filters have higher bit widths compared to their input signals. Within the top-level component the signals $x'(n)$, $y(n)$ and $y'(n)$ are saturated and rounded to the original data bit width to save hardware resources and to have constant bit widths for the signals in the internal feedback path. If an overflow of guardbits is detected, then the result is saturated to the minimum or maximum value of the new, trimmed bit width. Else the guardbits are truncated and the fractional part is rounded to the new bit width.

When a new sample $e(n)$ arrives at the ADC, the top-level component first calculates $x(n)$ and next signals the LMS component to start. The raw signal $y(n)$ is ready after one clock cycle. A second clock cycle is needed to have the trimmed signal $y(n)$ ready and be transferred to the DAC. Both FIR now start to calculate $x'(n)$ and $y'(n)$ and have their results ready and trimmed after another two clock cycles. The LMS and FIR filters now signal the top-level component their busy state and do the respective pre-calculations for a future sample. When the first static FIR and the LMS are not busy, the top-level component is ready to accept a new sample from the ADC for processing.

2.4 Parametrization of the Architecture

By using VHDL generics the I/O data word width, the word width of the filter coefficients, the stepsize of the LMS, the length of the adaptive filter and the level of calculations done concurrently can be configured without modifying the underlying VHDL code. For the FxLMS design the I/O data word width can be set. The I/O data word width is usually determined by the bit width of the ADC and DAC. The word widths for the adaptive LMS coefficients and the static FIR coefficients are also parameterizable. The filter lengths L_{LMS} and L_{FIR} and the number of concurrent subblocks B_{LMS} and B_{FIR} of the LMS and FIR can be set individually. Yet there are limitations when configuring the filter design. To make sure the concurrent subblocks are of equal length and the addertrees are symmetrical, the number of subblocks of both filter components must be set to a power of 2. Thus the length of the filters can only be set to $L = 2^n + 1$ where n is a natural number. The generic addertree is only synthesized where the LMS or static FIR components have at least two subblocks. The stepsize μ for the LMS also must be a power of 2.

Usually the optimal filter length and sample rate f_s is given by the physical circumstances in the ANC scenario. With a given clock frequency f_{clk} of the underlying hardware, the number of concurrent subblocks needs to be chosen so that eq. (10) and eq. (11) both are satisfied to provide enough clock cycles for the pre-calculations of $\widetilde{y}(n+1)$, $\widetilde{x}'(n+1)$ and $\widetilde{y}'(n+1)$ between samples.

$$\frac{f_{clk}}{f_s} \geq \frac{L_{\text{LMS}} - 1}{B_{\text{LMS}}} + log_2(B_{\text{LMS}}) + 7 \tag{10}$$

$$\frac{f_{clk}}{f_s} \geq \frac{L_{\text{FIR}} - 1}{B_{\text{FIR}}} + log_2(B_{\text{FIR}}) + 3 \tag{11}$$

The designs required clock cycles in eq. (10) result from eq. (7) plus two additional clock cycles for the subtraction $x(n) = e(n) - y'(n-1)$ and the trimming of $y(n)$. To also make sure the pre-sum of the FIR is calculated in time, eq. (11) needs to be true as well. If the top-level component is parameterized to ignore one equation, then samples will be skipped.

Since the LMS and static FIR filters have a constant processing delay it is recommended to use as few concurrent subblocks as possible to satisfy eq. (11) and (10). Configuring as few subblocks as possible saves hardware resources.

3. RESULTS AND EVALUATION

This section shows the hardware resources on the FPGA used by this design in different configurations. Most ANC systems found in the publication study from the introduction use a I/O data word width of 16 bit and static and adaptive filter coefficients of 32 bit. To make the here presented results easily comparable, we decided to use these bit widths in the following configurations. Also in [1] a practical application using these bit widths showed no loss in noise reduction performance compared to double precision floating point data types of the software implementation. The LMS stepsize is chosen as $\mu = 2^{-17}$. The hardware platform is a Xilinx Kintex7 XC7K325T FPGA clocked at $f_{\text{clk}} = 100\,\text{MHz}$.

There are too many configuration parameters of the filter design to present a complete evaluation of their impact on hardware resources. Instead only a small set of six configurations is selected as shown in table 1. The processing time given in number of clocks of the FxLMS N_{FxLMS} is calculated from eq. (10). The processing time of both static FIR filters also given in number of clocks N_{FIR} is calculated from eq. (11). The maximum sample rate $f_{s_{\max}}$ is the clock frequency divided by the higher processing time and is then rounded off to one kilohertz.

In [1] the performance increase of the hardware implementation on this platform compared to the software implementation was shown. The filter parameters in configuration C1 are similar with the one from the ANC-experiment M3 in [1]. The filter lengths differentiate by 1 due to the here included strategy for the constant processing delay. The experiment M3 was conducted at a sample rate of 40 kHz. With one subblock set per filter component, the FxLMS configuration of the VHDL design needs 1031 clocks to process a new sample. This results in a maximum sample rate of about 96 kHz. Table 2 shows the used FPGA resources in the synthesis result for C1 and the resources available. The Xilinx Kintex7 XC7K325T has a total of 50950 slices within an array of configurable logic blocks (CLB), further referred to as CLB slices. Also available are 445 block RAM tiles and 840 digital signal processing (DSP) slices. Each block RAM tile can be used as a block RAM with 36 kbit or can also be split into two 18 kbit block RAMs. For further information about the hardware, see [22].

As mentioned before the block RAM is used to store the data samples and filter coefficients. The DSP slices are

Table 1: FxLMS configurations with processing delays and maximum sample rate for $f_{\text{clk}} = 100\,\text{MHz}$.

	C1	C2	C3	C4	C5	C6
L_{LMS}	1025	1025	1025	131073	2049	65537
B_{LMS}	1	128	8	256	128	32
L_{FIR}	1025	1025	16385	1025	2049	131073
B_{FIR}	1	128	128	2	128	64
N_{LMS}	1031	22	138	527	30	2061
N_{FIR}	1027	18	138	523	26	2057
$f_{s_{\max}}$[kHz]	96	4545	724	189	3333	48

Table 2: Synthesis results for the given configurations.

Config.	Type	Used	Util. %
C1	CLB Slices	286	0.56
	Block RAM Tiles	4	0.90
	DSPs	12	1.43
C2	CLB Slices	25290	49.64
	Block RAM Tiles	320	71.91
	DSPs	840	100.00
C3	CLB Slices	12931	25.38
	Block RAM Tiles	200	44.94
	DSPs	541	64.40
C4	CLB Slices	22816	44.78
	Block RAM Tiles	387	86.97
	DSPs	781	92.98
C5	CLB Slices	26687	52.38
	Block RAM Tiles	320	71.91
	DSPs	760	90.48
C6	CLB Slices	15377	30.18
	Block RAM Tiles	322	72.36
	DSPs	357	42.50

Available	
CLB Slices	50950
Block RAM Tiles	445
DSPs	840

used for the multiplications. The CLB slices are used for all other components such as summations, registers or state machines. For C1 only a small fraction of the available hardware is used.
The configuration C2 optimizes the filter dimensions of C1 for high sample rates. It implements the same filter lengths, but uses as many concurrent subblocks as possible for this FPGA. This configuration needs 22 clocks to process a sample and to be ready for a new one. This allows for sample rates up to 4545 kHz. Table 2 shows that for C2 the number of available DSP slices sets a limit for the number of concurrent subblocks used. Also the utilization of the other resource types increases. Configuration C2 has the same filter lengths as C1. Thus the amount of data samples and filter coefficients stored in block RAM does not change. The number of used block RAM tiles increases, because each subblock has to instantiate its own physical block RAM. Since the subblock length is short in this configuration only a fraction of each block RAM memory is used.
The static FIR filters model the secondary path $S(z)$. Increasing the sample rate means that longer filters are necessary to model the same secondary path with equal precision. Configuration C3 is also optimized for high sample rates, but enlarges the static FIR filter lengths proportionally compared to C1. A static FIR filter length of 16385 and sample rate of 640 kHz means a 16 times increase. With a maximum of 128 concurrent subblocks the static FIR filters require 138 clocks between each sample to process. This limits the executable sample rate to 724 kHz. The next possible step would mean an increase by 32 times compared to configuration C1. Since we can not increase the number of concurrent FIR-subblocks anymore, this would decrease the executable sample rate to 362 kHz. Thus in a practical application a sample rate up to 640 kHz must be chosen. The number of concurrent LMS subblocks is set to satisfy eq. (10) for this sample rate. Table 2 shows the hardware utilization of C3.
Configuration C4 uses static FIR filter lengths of 1024 while increasing the filter length of the adaptive LMS as far as possible. As much as 131073 adaptive filter coefficients and 256 concurrent subblocks are possible to synthesize for the LMS component. The FxLMS needs 527 clocks to process a sample which results in a maximum sample rate of 189 kHz. The LMS filter length cannot be further increased due to limited memory. The number of concurrent subblocks cannot be increased on the given platform due to the limited number of DSP slices.
Configuration C5 and C6 are conducted to have a comparison to the design presented by Rivera Benois et al. in [15]. Since we do not know the architecture and its limitations, we use their feedback part with 2048 static and adaptive coefficients sampling with 48 kHz as a reference. Configuration C5 uses the same filter lengths which results in a maximum sample rate of 3333 kHz. About 52 % of the CLB slices and 72 % of the block RAM tiles are used. Configuration 6 tries to maximize the filter lengths while enabling a sample rate of at least 48 kHz. Table 1 shows 65537 adaptive and 131073 static coefficients can be used at this sample rate. Any further duplication of the filter lengths is prevented by the limited number of Block RAM tiles of which 72 % are utilized.

4. CONCLUSION

Large-scale ANC applications require long filter lengths and benefit from high sample rates. As these requirements contradict, a compromise needs to be found. This compromise varies for each specific ANC application and therefore demands for a fast and flexible design.
This paper presents a VHDL hardware implementation of a feedback FxLMS aimed for these use-cases. This architecture is parameterizable regarding filter length, sample rate, data word width, stepsize and the degree of parallelism to fully utilize the FPGA. Different configurations are synthesized to a Xilinx XC7K325T FPGA to show possible filter designs. The highest-order filter configuration uses 131073 static and 65537 adaptive coefficients and is able to sample at 48 kHz. The fastest, presented configuration uses 1024 static and adaptive coefficients and is able to process samples at 4.5 MHz. All configurations provide a constant short processing delay of three clock cycles, which is beneficial to the stability of the feedback FxLMS.
Although the performance improvement over the software implementation is good, the FxLMS algorithm gets unstable for high filter lengths and sample rates. Using the experimental setup described in [1], practical experiments with the here proposed architecture show that the design is stable using filter lengths of 1024 at 40 kHz and below. Higher order filters or higher sample rate let the filter become unstable after a few minutes, before the maximum noise reduction performance can be determined. Future work will integrate the leaky LMS-algorithm and normalized LMS-algorithm to improve stability.

5. REFERENCES

[1] **Removed for blind review**.

[2] C. Anghel, C. Paleologu, J. Benesty, and S. Ciochina. Fpga implementation of an acoustic echo canceller using a vss-nlms algorithm. In *International Symposium on Signals, Circuits and Systems, 2009*, pages 1–4. IEEE.

[3] M. Bahoura and H. Ezzaidi. FPGA-implementation of a sequential adaptive noise canceller using xilinx system generator. In *Proceedings of the International Conference on Microelectronics (ICM)*, pages 213–216.

[4] S. G. Boroujeny and M. Eshghi. FPGA implementation of a modular active noise control system. In *2010 18th Iranian Conference on Electrical Engineering*. IEEE.

[5] A. Di Stefano, A. Scaglione, and C. Giaconia. Efficient fpga implementation of an adaptive noise canceller. In *Proceedings of Seventh International Workshop on Computer Architecture for Machine Perception*.

[6] P. Goel and M. Chandra. Vlsi implementations of retimed high speed adaptive filter structures for speech enhancement. 24(12):4799–4806. cited By 0.

[7] I. Homana, I. Muresan, M. Topa, and C. Contan. Fpga implementation of an acoustic echo canceller.

[8] Intel Corp. *FIR II IP Core - User Guide*, 5 2016.

[9] A. Jalali, S. Gholami Boroujeny, and M. Eshghi. Design and implementation of a fast active noise control system on fpga. In *2007 Mediterranean Conference on Control and Automation (MED 2007)*.

[10] S. M. Kuo and D. Morgan. *Active noise control systems: algorithms and DSP implementations*. John Wiley & Sons, Inc., 1995.

[11] T. Lan and J. Zhang. Fpga implementation of an adaptive noise canceller. In *2008 International Symposiums on Information Processing*, pages 553–558. IEEE.

[12] A. Leva and L. Piroddi. FPGA-based implementation of high-speed active noise and vibration controllers. 19(8):798–808.

[13] M. Lorenzen, J. Hanselka, and D. 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.

[14] M. Pavuluri and B. Prasanthi. Low latency area efficient adaptive lms filter using fpga. 2018-January:114–116. cited By 0.

[15] P. Rivera Benois, P. Nowak, and U. Zölzer. Evaluation of a decoupled feedforward-feedback hybrid structure for active noise control headphones in a multi-source environment. In *INTER-NOISE and NOISE-CON Congress and Conference Proceedings*, 2017.

[16] P. Rivera Benois, P. Nowak, U. Zölzer, M. Eckert, and B. Klauer. Low-latency fir filter structures targeting fpga platforms. In *Proc. of HEART 2018 - International Symposium on Highly-Efficient-Accelerators and Reconfigurable Technologies*, 2018.

[17] C. Safarian, T. Ogunfunmi, and W. J. Kozacky. Fpga implementation of lms-based fir adaptive filter for real time digital signal processing applications. In *2015 IEEE International Conference on Digital Signal Processing (DSP)*, pages 1251–1255. IEEE.

[18] D. Shi, C. Shi, and W. 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)*, pages 1–6, Dec 2016.

[19] J. Shobba, Y. Murali, and M. Tech. Efficient fixed-point dlms adaptive filter implementation on fpga. (3.6):359–367.

[20] H.-S. Vu and K.-H. Chen. A high-performance feedback fxlms active noise cancellation vlsi circuit design for in-ear headphones. *Circuits, Systems, and Signal Processing*, pages 1–19, 2017.

[21] Xilinx, Inc. *FIR Compiler v7.2 - LogiCORE IP Product Guide*, 11 2015.

[22] Xilinx, Inc. *7 Series FPGAs Data Sheet: Overview*, 2 2018. Rev. 2.6.