Digital Signal Processing of Audible Signals using an FPGA

31015 - Group 9

Jonas Kazem Andersen s203825
Dept. of Electrical Engineering
Technical University of Denmark
31015 Introductory project
Elctrotechnology

Johan Wilhelm Bartling s203824
Dept. of Electrical Engineering
Technical University of Denmark
31015 Introductory project
Elctrotechnology

René Antonio Hjort s203880
Dept. of Electrical Engineering
Technical University of Denmark
31015 Introductory project
Elctrotechnology

Karl Herman Krause s203852
Dept. of Electrical Engineering
Technical University of Denmark
31015 Introductory project
Elctrotechnology

Johannes Nørskov Toke s203871
Dept. of Electrical Engineering
Technical University of Denmark
31015 Introductory project
Elctrotechnology

Abstract* - *This paper proposes and tries to implement digital signal processing architecture and audio algorithms that enables users to program Linear and non-linear audio processors on FPGA technology using application spcific hardware accelerators and $\Sigma\Delta$ AD- and DA conversion following the Eurorack specifications.

***Index Terms*—FPGA, DSP, digital signal processing, audio, Sigma-Delta modulation**

I. Introduction

In the wake of the COVID-19 crisis the audio industry has seen growth and new unforeseen feature demands such as smart assistant integration, noise cancellation and more. These rapidly changing requirements force the audio products to push hardware revisions because of their static design and limited computing power which, due to Moore's law slowing down, forces the design to use highly specialized and expensive hardware and thus, adding less value to the customers. A solution to this problem would be to use an easily re-configurable digital signal processing accelerated architecture which could be implemented within the Eurorack specification for ease of use. Because of this we propose CAPNA, which stands for Complementary Audio Processing Network Architecture. CAPNA enables the user to accelerate algorithms within digital signal processing (DSP) such as FIR-filtering, machine learning (ML) such as Wavenet and arithmetic calculations such as summation or multiplication.

A. Problem Definition

To realize the CAPNA architecture and its Eurorack implementation, the main areas of interest in this paper will be:

- What architecture would be advantageous for implementing a digital signal processing system through FPGA technology.
- What are the capabilities of the DSP slices, and how can we optimize the processing within the limited resources
- How are the Eurorack and line level standards specified , and how can the system be made compliant with Line level signals and/or the Eurorack interfacing specifications.
- What tools should we use to create an open-source platform, as well as a compiler, which allows for easy reprogramming of the desired signal processing

II. Preliminary research

The construction of a digital signal processor consists of designing an analog system, designing an digital system and researching digital signal processing algorithms. The analog system handles the conversion of data between the digital and analog realms. Meanwhile the digital system as well as the algorithms handles the modulation of the signal. The designs for the systems in the different realms will be virtually independent of each other, and are therefore treated as separate problems during problem solving.
Much of our early research was into the hardware resources available on Artix-7 35T FPGA available on our Basys3 boards, since this would be a significant factor in the later construction of our digital and analog hardware. Besides that, another significant part of our early research was into the various DSP-algorithms that we wanted to compute, seeing that we would have to tailor our hardware to those specifications, and thus, it is also considered as its own problem during problem solving.

A. Solutions for Data conversion

The first step of the research regarding the analog design revolved around the Eurorack standard. The specifications required for interfacing with other audio equipment are rather relaxed. We specifically chose the Eurorack system

specifications as they can be used for a wide variety of applications within audio processing. Eurorack requires the system to be able to receive and send audible voltage signals in the range of $\pm 5\,\mathrm{V}$. The supply voltage has a positive value of $12\,\mathrm{V}$ and the negative voltage is $-12\,\mathrm{V}$. Lastly, the connection ports have to be $3.5\,\mathrm{mm}$ jack ports [1].

As a consequence of our choice of specification, we are not able to use the on-chip ADC, as it only supports an input range of 0 - 1 V

We will instead need an external analog to digital converter for the input, and an external digital to analog converter for the output. For this reason, the second phase of research was concerned with the pros, cons and topologies of different ADDAC types. Many different architectures exist for both converters. For the ADC, we decided to design a Sigma-Delta converter, as it offers the best flexibility and versatility of any method, as well as having potentially good noise rejection through noise shaping. In order to convert the signal from digital to analog, we looked into multiple solutions, those being R/2R ladder DAC (with and without a buffering op-amp) and use of Sigma-Delta modulated signals. We again chose Sigma-Delta modulation [2], because it provides the best versatility compared to its complexity, and requires little analog circuitry, having the benefit of being easier to debug. Both designs were simulated using digital design tools in order to verify them before implementing them on a circuit.

B. Solution for the Algorithms

Audio signals are continuous in nature but the information in them can be processed discretely using algorithms to be reproduced as continuous signals afterwards. Because of this notion we wanted to research and implement different ways of processing signals discretely.

To accomplish discrete signal processing we have had to obtain an understanding of discretization of continuous signals, discrete linear and non-linear signal processing and common current optimal implementations of signal processing algorithms. To leverage the fact that signal processing can be classified as either Linear or non-linear, we decided to accelerate the linear audio processing using the FIR-Engine, and compute the non-linear audio processing using general purpose computing processor. As for linear effects we choose to mainly use the research of Frank Werfers on discrete linear filtering. [3]

We decided to propose optimized implementations for FIR-filtering [4] and frequency domain based filtering using the overlay save method [3] using FHT [5] for the linear signal processing. For the non-linear signal processing, the general purpose processors is able to run custom algorithms suited to non-linear operations.

C. Solutions for the digital system

Initially, we reviewed our options regarding the design of the digital system. The central idea of the architecture were to have separate smaller processors, which could be configured individually, and interconnected in any number of ways.

Digital signal processing requires a lot of memory for storing coefficients used in the filters. Unfortunately, the memory resources of the DSP slices are limited, so we needed to know how much memory each processor would need. After having decided on how to compute the filtering it became apparent that we would have no problem regarding computation time, but the amount of memory we would need exceeded the amount of block ram available on-chip. We have therefore used a memory module that one of our group members has worked on in another project [6]. It is used to store the necessary coefficients and samples when they are not being used. Furthermore the FIR-filter filtering the input and output signals takes advantage of the symmetrical nature of FIR-filter impulse responses and stores only the first half thus, only using half the memory resources.

III. Design and Considerations

A. Data Conversion

Converting analog to digital is done through a series of steps consisting of subtraction, summation and finally comparison. The subtraction and integration is performed by an op-amp in a summing integrator configuration. The comparator sums the present analog voltage with an impulse based on the previously processed input. The output of the summing integrator is then compared to $0\,\mathrm{V}$, which is the middle point of the voltage swing. This will generate the PDM signal used as the digital input read by the FPGA. Inside the FPGA, the PDM signal is loaded into a serial to parallel buffer and low pass filtered to remove artifacts. The sampling is done at a rate of $N{\cdot}44.1\,\mathrm{kHz}$, where N is the target bit depth of the signal. This signal is then decimated by a factor N, the same as the bit-depth of the processor. The schematic used was inspired by an example shown in an online book [7], and is supplied in Appendix A. Choice of components for comparator and op-amp were respectively the *LM339* and *TL072ACDT*, both supplied by Texas Instruments. These were chosen due to their switching speeds and relatively low cost. Regarding the randomness of an incoming signal, Figure 1 shows a sine wave after random sampling delays, inherent to the process of S-D modulation. This shows that the process should be feasible.

As the FPGA PMOD IO ports have a logic one at $3.3\,\mathrm{V}$ and logic zero as $0\,\mathrm{V}$, it is not directly compatible with the Eurorack specifications. We solve this by having the open collector output of the comparator, that generates the PDM input, connected to VCC of the FPGA output ports thus, generating a signal below the absolute maximum ratings.

Regarding the digital to analog conversion, the following algorithm [2] is used:

$$PDM = MSB(Z_{-1}) \tag{1}$$

$$Z_{-1} = Z_{-1} + (Z - DDC) \tag{2}$$

$$DDC = -PDM \tag{3}$$

This essentially does the same thing as the analog part of the ADC, but in the digital domain. A sample Z is first interpolated by a factor N, then filtered through a LP

configured FIR-filter. The sample Z is then inserted into the algorithm shown in (1) - (3). In the analog domain, the PDM signal is filtered through a 2nd order LP Sallen-Key filter, before being sent to the output jack port.

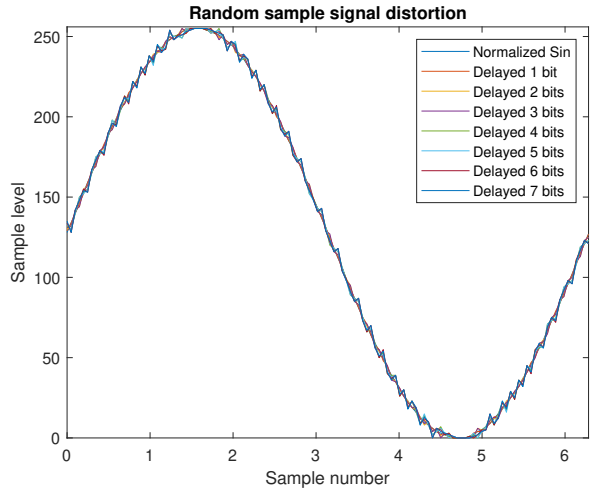

Fig. 1. Simulated distortion of an 8-bit Sine wave caused by time displacement of the samples using a Sigma-Delta converter.

B. Processor Architecture

CAPNA (Complementary Audio Processor Network Architecture), is a network configuration of individual CAPs (Complementary Audio Processor), each running different DSP applications. This network is synthesized from a configuration XML file, which allows the user to specify the interconnections between multible CAPs and the program each CAP runs. The intention with this architecture is uncomplicated scalable and application tailored digital signal processing.

1) CAP: CAP consists of 3 elements. The DSP core, FIR-engine and shared data-memory. This architecture is inspired by the motto: "Make the common case fast". Since we determined that FIR- and IIR filtration is an extremely common operation within DSP, we decided to build dedicated hardware specifically for this application, and named it the FIR-engine. To allow for a diverse array of possible operations, we additionally chose to add the DSP Core for non-linear operations. These modules are interconnected and therefore allows for effective generation of diverse effects.

2) The DSP core: The DSP core is an 18 bit, 16 register RISC processor with a 4 stage pipeline. The processor runs a custom ISA, built specifically for DSP applications. The ISA is specified on our github repository. The ISA includes instructions present in a typical RISC architecture, in addition to DSP specific instructions, such as *multiply accumulate* and fixed point multiplication. The core also supports external QSPI DRAM memory, with sizes up to one Megabyte.

3) The FIR-Engine: The FIR-engine operates as a hardware accelerator configured by the DSP core, and uses registers controlling a state machine, that determines the configuration of the multiply-accumulate hardware, in regards to the desired effect. This DSP core can access the registers by writing to certain addresses in the address-space and through them control the FIR-Engine. The DSP Core can enable the FIR-Engine as well as configure filter length and other parameters. The FIR-Engine is called FIR-Engine because it by default is set to accelerate FIR-filtering. This is our proposed design based on the Xilinx DSP48E1 slice (DSP slice) [8]. The state machine allows for the other proposed algorithms based on multiplying and accumulating such as IIR, FHT and dot products to be implemented and accessed by setting its state register differently.
A FIR-filter essentially performs a cyclic convolution of a signal, by calculating a sum of multiplications. Because of this, the FIR Engine is designed around the DSP slice. The DSP slice consists of an 18bit pre-adder, an 25x18bit pre-multiplier, an ALU and a pattern detector with registers between modules for pipelining purposes. Because of this, the DSP slice can perform both a multiplication and accumulation in only 2 clock cycles, and perform a cyclic convolution in $filter - length + 3$ clock cycles using separate memory for the coefficients and samples, same as in the IO-filter. If only using one separate memory, the cyclic convolution can be done in $2 \cdot filter - length + 1$. This is much faster than the DSP core, which uses 4 clock cycles per multiplication and accumulation and perform a cyclic convolution in $4 \cdot filter - length + 1$ clock cycles. This means that the FIR- Engine powered by the DSP slice potentially can perform convolutions four times faster than a general purpose computer and thus, employ a 4 times longer filter.
The transfer function of the filter depends on the filters coefficients since they are discrete samples of the filters impulse response. The filter impulse of an optimal FIR-filter is symmetric and thus, only half of the filter coefficients is required to do the convolution, and thereby have to be stored. To further minimize memory usage we singled out filters with the shortest number of coefficients which meet specifications to be implemented. This was done via the Parks–McClellan filter design algorithm which calculates the shortest filter response - which has a certain stopband, passpand, ripple gain and gain. To do this we used a site called TFilter [9] which has a javascript implementation of the Parks–McClellan algorithm.

4) Network generation: One of the main goals of the project was to generate networks of multiple parallel processors each running different effects. To allow for such a network we needed to build parametrizable chisel code that would be able to generate an arbitrary number of CAPs with different loaded programs, as well as interconnect the processors in a correct fashion. We decided to use xml files for configurations. These files allow us to easily define multiple CAPs, their programs, memory sizes and interconnections in an ordered and easily configurable manner. This would also allow for easier implementation of a possible GUI system, which would generate these files.

5) Assembler: The use of a custom 18 ISA necessitates the use of a custom compiler and assembler. Seeing that a custom compiler or compiler back-end was somewhat out of the scope of the project, we instead chose to focus out

efforts on the assembler. The assembler is a Java program, which is called by the top Scala program during compilation and testing. Since we don't have a compiler we additionally chose to add a number of pseudo-instructions to make it easier to read and write the assembly code. This also makes it easier for us to interact with the FIR-engine. The assembler goes through through 3 stages before producing the final binary output. First pseudo-instructions are replaced with their respective real instructions, secondly tags are removed and finally the assembler converts the individual instructions into hex machine-code.

C. Algorithms

1) Mathematical basis of Digital signal processing algorithms: To understand the working principles in the following algorithms, we have had to understand the basics of digital signal processing. In signal processing, the signals processed are sampled [10], which is the process of recording the signal amplitude at regular intervals. When sampling a signal images of the positive and negative signal frequency components appear at centered around the +/- the sampling frequency. This unwanted noise created when the image and the signal becomes indistinguishable is called aliasing, which can be avoided using the sampling theorem [11], which states that by only introducing frequency components less than half the sampling speed, this noise is avoided.
Sampling together with the Z-transform [10], the cyclic convolution [10], the discrete Fourier transform (DFT) [10] and difference equations [10] form the mathematical basis of most digital signal processing algorithms.
The Z-transform is a mapping from a discrete series of real or complex number sequence to the complex frequency plane often used for analyzing the transfer functions of discrete LTIC systems. The Z-transform is analogous to the Laplace Transform, but for discrete signals. The bilateral and unilateral Z-transform is used for LTI and LTIC systems respectively, and they are defined to be

$$X(z) \triangleq \sum_{n=-\infty}^{\infty} x(n) z^{-n} \tag{4}$$

$$(z) \triangleq \sum_{n=0}^{\infty} x(n) z^{-n} \tag{5}$$

To invert the Z-transform, one can use partial fraction decomposition [10] and transform the individual fractions (as one would with the inverse Laplace Transform). The discrete time Fourier transform or DTFT [10] is equivalent to the continuous Fourier transform FT but like the FT it too produces an infinite series of frequency components. To obtain a finite number of frequency components n from a time series of a signal with length n the discrete Fourier transform (DFT [11]) can be used. The DTFT,inverse DTFT, DFT and inverse DFT respectively are defined to be

$$X(\tilde{\omega}) \triangleq \sum_{\infty}^{-\infty} x(n) \exp(-j\tilde{\omega}n) \tag{6}$$

$$x(n) = \frac{1}{2\pi} \int_{-\pi}^{\pi} X(\tilde{\omega}) \exp(j\tilde{\omega}n) d\tilde{\omega} \tag{7}$$

$$X(\omega_k) \triangleq \sum_{n=0}^{N-1} x(t_n) \exp(-j\omega_k t_n), \qquad k = 1, 2..., N-1 \tag{8}$$

$$x(t_n) = \frac{1}{N} \sum_{k=0}^{N-1} X(\omega_k) \exp(j\omega_k t_n), \qquad n = 1, 2..., N-1 \tag{9}$$

The cyclic convolution is equivalent to the continuous convolution but for the DFT. Like the continuous convolution, cyclic convolution has a theorem which states that convolving two signals using the cyclic convolution in the time domain is equal to multiplying the two signals frequency components in the frequency domain. Because the DFT of a signal consisting of n samples has a length n, the two signals have to both have the same length, which can be achieved by padding, should one be shorter than the other. The discrete convolution is defined to be

$$X(n) \cdot H(n) \xrightarrow{DFT^{-1}} \sum_{m=0}^{n} x(m) h(n-m) \triangleq (x \circledast h)_n \tag{10}$$

A difference equation [10] is a formula for computing an output sample based on past and present output and input samples in the time domain. Difference equations are essential to DSP, since they are used to obtain the transfer function of discrete time LTIC filters and due to being easily implemented.

2) Finite impulse response filtering (FIR-filtering): As mentioned earlier in the Section "The design of the FIR-Engine", FIR-filtering is filtration using cyclic convolution of an input signal with finite impulse response, described in the previous section. The finite impulse response is created from a shifted and truncated the impulse of an ideal filter. This response is then time-shifted to create a causal impulse, and truncated to make it finite and avoid infinite input terms. The truncation can be done using different methods such as Mean Square Error (MSE) method but the simplest is the window method were the impulse is multiplied with a window function(a discrete finite signal). The truncation adds a discontinuity which translates to ripples in the frequency domain. The amount of ripples is different depending on window functions. As mentioned in the prior section "The design of the FIR-Engine", an optimal finite impulse response in regards to the ripple gain attenuation, gain in the pass- and stopband can be found using the Parks Mclellan algorithm [9]. This is especially in relation to the memory usage. .
The impulse responses used for FIR-filtering can be both symmetric or anti-symmetric, but symmetric FIR-filters both have linear phase and only half of the coefficients have to be stored because of symmetry. Since The Parks Mclellan algorithm always calculates the minimum symmetric filter, it is the most advantageous FIR-filter design regarding memory utilization.
FIR-filtering is implemented as a difference equation. It can be calculated using multiplication and accumulation. Because of this, the FIR-Engine can accelerate calculation of these filters.

3) Infinite impulse response filtering (IIR-filtering): IIR-filtering is done using a difference equation obtained using

the Z-transform on a desired discrete transfer function. Designing the discrete transfer-function can either be done directly in the Z-domain or by transforming a continuous transfer-function from the s-domain to the Z-domain. The most common method for this is frequency warping the continuous transfer-function and using the bi-linear transform [10] and the Matched Z-transform method [10]. IIR-filters are often shorter in length compared to FIR-filters, but have non-linear phase. This makes them advantageous in situations where linear phase is not important. The difference equation for IIR-filter is similar to that of the FIR-filter thus, the FIR-Engine can also be used for IIR-filters.

D. Algorithms combining FIR- and IIR-Filtration

Many combinations of FIR- and IIR filtration can be found in digital signal processing. One is the CIC- filter, which combines FIR- and IIR filtering to efficiently decimate a signal, when configured as an IIR-filter followed by a FIR-filter or interpolate when configured as a FIR-FILTER followed by IIR-filter.
Another design type of filters, is one which relies on comb-based filtering. These are either feed forward or feed back. Using feed-back comb filters acoustic reverberation and Phase-based filtering can be achieved because of destructive and constructive interference between samples [10]. Note that feed-forward comb filters are computationally more efficient than IIR filters. Using feed-forward comb filters, delays can be achieved using tapped delay line structures [10]. Since all these effects use multiplications and accumulations, the FIR-Engine structure can be used to accelerate these.

1) Non-linear algorithms: Non-linear algorithms are effects containing non-linear operators. Examples of these are dynamic effects like distortion and compression or modulation effects like tremolo, phaser and chorus or non-linear filtering like amplifier-modelling. Also machine learning algorithms such as auto encoders, which use neural networks, are nonlinear as they use an non-linear activation functions like the RELU or the sigmoid function. All these non-linear effects can be computed using a general purpose computer, and some algorithms like the neural network might benefit from having multiple general purpose computers, since they run in parallel. Thus, by having a network of CAPs each with a DSP Core, the CAPNA architecture is able to run these algorithms

IV. Measurements and Analysis

As there were a couple of mistakes regarding the AD-DAC board, a lot of time was invested in debugging and resoldering wrong connections.At the same time, it proved more difficult than anticipated to make the FPGA process the input and output signals in the correct way. This caused a setback resulting in us not having a finished product at the time of writing.

A. Data conversion & Signal to noise ratio

The signal to noise ratio has not been quantitatively measured, as a qualitative evaluation by simply listening to the signal proved that the system would not yet produce a satisfactory SNR. The test setup consisted of an analog signal generator, such as a computer or another device with a mini jack port that can play audio, the ADDAC, FPGA, and a speaker. The input of the ADDAC is connected to the signal, and the output to the speaker. The input and output of the FPGA is connected to the ADC and DAC parts of the ADDAC board respectively.
Before testing the interface between the FPGA and ADDAC, the ADDAC board was tested on an Arduino Nano running a similar algorithm. The resulting output was audible, however with a lot of noise and low bitdepth (10 bits).This yielded far better results than when the arduino is replaced with the FPGA, suggesting that the internal data conversion or the digital to analog Sigma-Delta modulation on the output is not working properly.

B. Verification and debugging of digital design

For analyzing the digital systems, besides generating physical implementations to test, the primary method of testing has been generating waveform files through Verilator and Scala test. Waveform files has the file extension .VCD, and can be viewed through VCD viewers such as GTKWave. Additionally we included a sample tester which can simulate the audio output of the processors when a certain sound is input. The tester is a based on code created by Martin Schoeberl and Clemens Pircher [12].

These virtual tests have several advantages over physical tests, as they are faster and provide the ability to examine every signal, both internal and external. Whereas physical tests only allow for examining external signals. Various different test programs have been written for verifying the proper functionality of each subsystem.

C. Parameters of the digital implementation

To evaluate the parameters of the digital system, we used the implementation report tool provided in Vivado. This tells us about the speed, power and area the chip takes up. Current implementations take up 437 LUT's, 348 FlipFlops, 2 Block Ram, 2 DSP slices and 0.084W with a max clock frequency of 104.8MHz. These implementations are without the DSP cores. That is, the input controller is connected directly to the output controller.

D. Algorithms

For analyzing the implementation of a FIR filter using the FIR-Engine, we ran a digital test and compared the cyclic convolution results to digital waveforms created and analyzed using chiseltest and GTKwave. For analysis of non-linear algorithms such as a bitcrusher or distortion, we also used chiseltest and GTKwave.

V. Evaluation of product

Sadly we have not been able to produce a finished product. Which means there are a lot of tests that we were not able to conduct to truly measure the performance of the system.
We have yet to test the actual chip area, clock speed, or any other parameter of the complete design. The same

could be said for the delay from input to output. And as the ADDAC is partly digital, we cannot get a clear image of what the signal to noise ratio would be. It would also have been convenient to know the performance limitations of the processor implementation. e.g. how much filtering or how many effects we could add to the signal.
Although the system does not yet work as intended, we can already list a number of changes that could improve the system.

As the ADDAC system is a prototype, a future version would be improved if it allowed for more connections, ie. more parallel inputs to the board, and integrating the FPGA directly onto the board, either through CMOD A7 evaluation board or directly placing a FPGA on the board. Another addition would be faster comparators, as the *LM339DR* just manages the switching speeds for 18-bit signals. The S-D converters could also be designed as higher order converters compared to the first order converters implemented in the current design, as signal noise is pushed into higher frequencies whilst keeping the desired signal somewhat unchanged. [13]

Another improvement for the ADDAC-circuit could be a digitally controlled resistor or capacitor network connected to the ADC-integrator, in order to support higher oversampling ratios, thereby higher sampling frequencies whilst keeping the bit-depth unchanged. A third improvement could be an implementation of an AGC-amplifier, as the Sigma-Delta ADC modulation is highly prone to the volume level of the incoming sound. This could be implemented with a network of digitally controlled resistors connected to an OTA like the LM13700, as opposed to the currently implemented op-amp with manually variable gain on the input of the ADC. The DAC could benefit from having a comparator or a set of open-drain comparators to switch an outgoing PDM signal, as this could generate a desired voltage level of $\pm 5\,\mathrm{V}$.

Area-efficiency is key for operation and since the Input-Output filters are hardware that is in constant use, making them as cheap as possible is desirable. A CIC-filter would be an alternative. Regarding the common-case, the FIR/IIR-filters would be the hardware accelerators in use to create the desired effect through eg. cascaded all pass filters.

Also, since memory is such a limited resource algorithms such as a CORDIC (Coordinate Rotation Digital Computer) would be an efficient way to approximate tedious values instead of storing them. Furthermore, it would allow for an expansion of a complementary Fast Transform, that would be used for longer filter lengths for more effective operation. We would also be happy to see a compiler and a user friendly GUI created in the future. In regards to correcting the errors making the design non-functional, more testing is needed before we can identify the error that is causing us problems.

VI. Conclusion

We propose an architecture design made of a basic computer processor complemented by a network of hardware accelerators. Specifically using the DSP slices proved rather simple, as the Vivado software automatically interprets when to use the slices by inferrences in the code. We propose that the most optimal architecture for a scalable DSP application uses the DSP slices as complementary hardware accelerators.
The Eurorack specifications are designed for easy compatibility. Making our design compatible only meant designing the electronic circuit to have an input range of $\pm 5\,\mathrm{V}$ and a positive supply voltage of $12\,\mathrm{V}$ and a negative supply voltage of $-12\,\mathrm{V}$. To implement analog to digital and digital to analog conversion, we designed a Sigma-Delta ADDAC. The project is uploaded to git [1] and has been made public. Furthermore the project structure, including an open source license has been copied from a chisel project template. [14] In the future we propose to create a docker [15] environment, which includes verilator, for easier developer access, and we intend to make the user experience better by creating a compiler and GUI.

References

[1] D. Doepfer. A-100 construction details. Doepfer Musikelektronik GmbH. [Online]. Available: https://doepfer.de/a100_man/a100m_e.html

[2] D. Boschen. (2020, Jul) How does the world's simplest sigma-delta dac work? [Online]. Available: https://dsp.stackexchange.com/questions/69004/how-does-the-worlds-simplest-sigma-delta-dac-work

[3] F. Wefers, "Partitioned convolution algorithms for real-time auralization," Ph.D. dissertation, Rhenish-Westphalian Technical University of Aachen, 2014.

[4] P. Horowitz and H. Winfield, "Digital signal proccessing," in *The Art Of Electronics*, 3rd ed., 2016, ch. 6.3.7.

[5] L. Pygras, P. Kitsos, and A. Skodras, "Compact fpga architectures for the two-band fast discrete hartley transform," *Microprocessors and Microsystems*, 2018.

[6] M. Schoeberl et al. Patmos. [Online]. Available: https://github.com/os-chip-design/patmos.git

[7] T. R. Kuphaldt. Delta-sigma adc. [Online]. Available: https://www.allaboutcircuits.com/textbook/digital/chpt-13/delta-sigma-adc/

[8] (2018, Mar) 7 series dsp48e1 slice user guide (ug479). Xilinx Inc. [Online]. Available: https://docs.xilinx.com/v/u/en-US/ug479_7Series_DSP48E1

[9] P. Isza. (2011) Tfilter. [Online]. Available: http://t-filter.engineerjs.com

[10] J. O. Smith, *Introduction to Digital Filters with Audio Applications*. http://ccrma.stanford.edu/ jos/filters/, accessed 17.06.2022, ch. 3,6,7,14, online book, 2007 edition.

[11] ——, *Mathematics of the Discrete Fourier Transform (DFT*. http://ccrma.stanford.edu/ jos/mdft/, accessed 17.06.2022, ch. 2,11,13, online book, 2007 edition.

[12] M. Schoeberl and C. Pircher. soundbytes. [Online]. Available: https://github.com/schoeberl/soundbytes/blob/main/src/test/scala/soundbytes/SoundTest.scala

[13] B. Baker, *How delta-sigma ADCs work, Part 1*, Texas Instruments Inc., September 2016. [Online]. Available: https://www.ti.com/lit/an/slyt423a/slyt423a.pdf?ts=1655568597532&ref_url=https%253A%252F%252Fwww.google.com%252F

[14] Chisel template. Free Chips Project. [Online]. Available: https://github.com/freechipsproject/chisel-template.git

[15] (2013) Docker. Docker Inc. [Online]. Available: https://www.docker.com/

[1] https://github.com/mohanjan/Fagprojekt2022_Grp9