PADAWAN
Parallel Accelerator for Digitising Audio With Attenuation of Noise
Final Report

EEE4084F Class of 2017
University of Cape Town
South Africa

James Cushway[*], George de Kock[†], and Johan Jansen van Vuuren[§]

[*]CSHJAM001
[†]DKCGEO002
[§]JNSJOH014

***Abstract*—This paper details the design and methodology for building a hardware accelerator for parallel digital audio processing. The Nexys 4 DDR evaluation board with a Xilinx Artix 7 FPGA is used for the implementation of the accelerator. The hardware accelerator is used to digitise an analog audio signal using a 12-bit ADC, and perform digital filtering thereon to create numerous audio effects. A PWM output, obtained by combining the results of the parallel filtering operations, is sent to a mono audio output port. Effects include echo, chorus, reverberation and distortion, specifically overdrive. The intensity of effects are designed to be dynamically adjustable using switches found on the Nexys 4 DDR board. The filters designed for the board are additionally implemented in Octave as a golden measure, and were tested on various audio signals to quantify performance, with which the FPGA performance is compared and evaluated. The paper also details the effects of bit truncation on audio, and the use of dithering and noise shaping to rectify said effects.**

I. Introduction

The PADAWAN (Parallel Digital Accelerator With Attenuation of Noise) is a digital accelerator which filtering and processing on an input audio signal, streamed to the device from a host PC. The system is be implemented on a Nexys 4 development board, which hosts a Xilinx Artix 7 FPGA.

The hardware accelerator implements parallel digital filtering, with each parallel filter implementing a different sound effect on the input audio stream. The different audio effects that are implemented are echo, chorus, reverberation and overdrive. [1] The effects are implemented by performing various combinations of the audio signal and delayed, weighted versions of itself, and various forms of modulation. These filters are adjustable during operation by various user inputs found on the Nexys 4 DDR.

Thereafter, the output of filters are modulated using PWM. To satisfy the clock speed requirements for PWM, bit truncation needs to occur which causes a some distortion and unwanted harmonics within the output analog waveform. Dithering [2] can be used to reduce the unwanted effects of the bit truncation, with the slight trade-off of having a slightly higher noise level on the output, and noise shaping can be used to reduce the perceived noise level on the output introduced from dithering.

Finally the system was tested on live audio streams, to verify the implemented effects caused the desired changes to the signal. The changes to effects via user input were also be tested in this manner. The system requirement that the device performs in real-time, as far as is perceivable to the human ear, was confirmed by playing the filtered output in parallel with the original audio signal and observing that the audio was synchronised. Thereafter, the filters were implemented on Octave and performance was compared by means of wall clock time, that is, how long it took for various combinations of filters to yield output samples. The Octave implementation was used as a golden measure to compare to the performance achieved on the hardware accelerator.

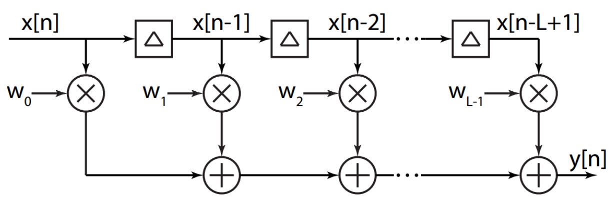

Fig. 1. Block diagram of basic FIR filter [3]

II. Background

A. Digital audio filtering

Digital signal processing is a very common task due to its wide application in a variety of fields. Digital filtering can be performed using two different implementations, i.e. finite impulse response (FIR) filters and infinite impulse response (IIR) filters. The most important distinction between these two implementations is that FIR filters utilise present and past input values only, while IIR filters use additional previously calculated output samples. IIR filters are inherently dependent on negative feedback, presenting possible instability issues. For the purposes of this proof of concept of hardware acceleration of real-time audio filtering, FIR filters are implemented to achieve specified audio effects. Achieving real-time filtering of audio signals using FIR filters, involves discretization of an analogue audio input, and discrete-time convolution, a process analogous to the multiplication of the current input sample and *(L-1)* previous input samples with the tap weights of the filter transfer function, where L is the number of nonzero weights in the transfer function *h[n]*, followed by the summation of all the products. This process can be represented by Equation 1.

$$y[n] = \sum_{k=-\infty}^{\infty} w[k]x[n-k] \quad (1)$$

To better understand the operation being performed, the system is visualised in Figure 1, which shows a block diagram a basic FIR filter. The input sample *x[n]* and its previous inputs, $\{x[n-c], c>0\}$are multiplied by constant tap weights corresponding to the delay of each sample. A one-sample delay is indicated by the Δ-operation.

B. The issue of computational complexity

Audio filtering is a computationally expensive operation, as an L-length digital filter requires L^2 multiplications and L additions, leading to high computational complexity. When the filters are of high order, i.e. are nonzero for relatively long intervals, the cost of computing output samples becomes a significant limiting factor on the throughput. The problem also manifests when multiple filters are applied to an input in real time. This presents the opportunity for a parallelised system computing different outputs for different filters, and then combining the results to obtain an output corresponding to the combination of parallel filters used.

Two different methods exist for implementing FIR filters, i.e. multiple multipliers (MM) and distributed arithmetic (DA) FIR filtering. MM-FIR filtering is a typical method, where the number of multiplier elements is determined by the required throughput [3]. One accumulator is used to store the sum obtained in the aforementioned process.

For the purposes of this project, the number of multipliers is set to the length of the filter coefficient vector. The multipliers operate in parallel, and each element requires access to memory holding filter coefficients and past and present input samples. The throughput of an MM-FIR filtering system is proportional to the number of times the multipliers are used. As a result, higher sampling rates are obtained by using many multipliers.

The number of coefficient memories is equal to the number of filters being implemented, as all filters operate in parallel. The coefficients are therefore accessed simultaneously with a circular pointer system for incrementing the address during the filtering operation. The same pointer system is used for the memories holding the input samples. Each multiplier's sample and coefficient memories are combined, illustrated in Figure 2. A memory pointer iterates over the L addresses during each sampling period. Adders arranged in a tree structure are used to sum the products generated by the multipliers. The output of the adder tree is fed into the accumulator, with the output available after L clock cycles. The width of data buses (in bits) are indicated with *B* in Figure 2.

C. Open-source libraries and the development challenge

Existing HDL modules exist in online open-source code repositories, was used as a starting point for the PADAWAN project. An XADC and PWM output module written for the Nexys 4 DDR in Verilog is available in an open-source licensed repository [4].

Many digital signal processing techniques are independent of the data type. The interface and communication protocol between different elements

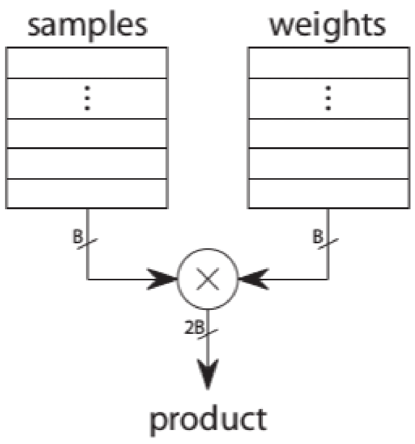

Fig. 2. Combination of multiplier input and coefficient memories [3]

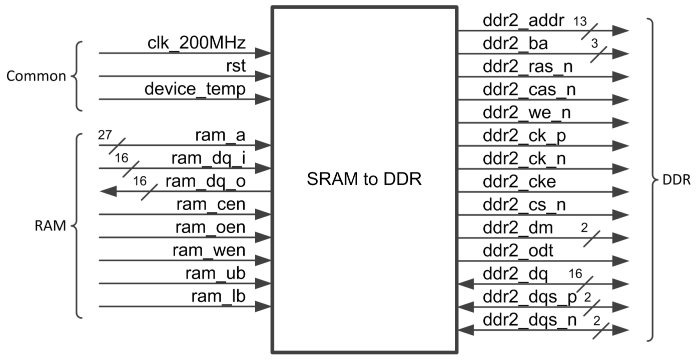

Fig. 3. DDR to RAM control signal interface block

on the Nexys 4 board remain the same regardless of the signal type (e.g. audio or visual). Naturally, no two projects are the same and many features and modules will have to be built from scratch. The Nexys 4 reference manual and Verilog programming guide will be of use during the system architecture design and software implementation. Initially, Xilinx ISE design suite was used for development, but many online resources rely on Vivado which uses a different file format for specifying constraints as well as a redesigned IP core generator. The IP core wizard on Vivado 2017.2 was of great help to simplify interfacing with the XACD and DDR2 RAM. The project mentioned above also included a VHDL file that translates the more complex DDR control signals to RAM control signals. A block diagram [5] shown below (figure 3) illustrates the conversion.

III. Design

In the design of the PADAWAN, FPGA level requirements were developed by analysing the system level requirements:

1) Near real-time audio filtering and processing
2) Dynamic adjustment of filters and effects
3) Digital audio stream as an input
4) Analogue output using pulse-width modulation
5) Noise shaping to increase apparent resolution

. Multiple audio effects are implemented on the audio signal and filters were implemented on the Verilog HDL platform. Implementation of pulse-width modulation is required to produce a PWM signal from discrete sample values.

Figure 4 shows a block diagram of the PADAWAN. An audio signal is fed into the ADC of the Nexys 4. The digitised signal is then sent to a parallel bank of filters where each filter is performing a different effect on the audio input.

A. *Pulse width modulation of output signal*

Pulse-width modulation (PWM) of a signal places a high load on the system clocking requirements. Increasing the resolution of the PWM output requires a higher system clock. To determine a suitable audio resolution, the Nexys 4 clock frequency and the PADAWAN sampling rate were considered. Since the desired output sampling rate is 44.1kHz, the PWM carrier will have this frequency. The XADC on the Nexys 4 has a maximum resolution of 12 bits, i.e. 4096 distinct output values. The required PWM granularity is therefore 4096. The pulse width modulator will be driven by the 100MHz system clock. The required PWM frequency however, is obtained by using Equation 2 [6].

$$F_{PWM} = F_{range} \times Resolution = 180.634MHz \tag{2}$$

It is clear that the device is not fast enough to produce this resolution of audio output, as it is clocked at 100MHz. Thus, bit truncation is necessary to meet the clock requirements for the PWM signal. Using the above calculations, it was discovered that an 11-bit sample is the maximum resolution that the 44.1kHz PWM signal can represent given the 100MHz clock.

B. *Noise shaping*

As described above, bit truncation was performed on the digital 12-bit samples. This process introduces quantization distortion, manifesting as unwanted harmonics in the output. The design initially accounted for this by the proposed implementation of noise shaping with dithering. Before sampling, additive white noise can be added to the input signal which greatly reduces the unwanted effects caused from bit truncation, with the penalty of a slightly higher noise level in the output. Noise shaping can be performed, a process that manipulates the power

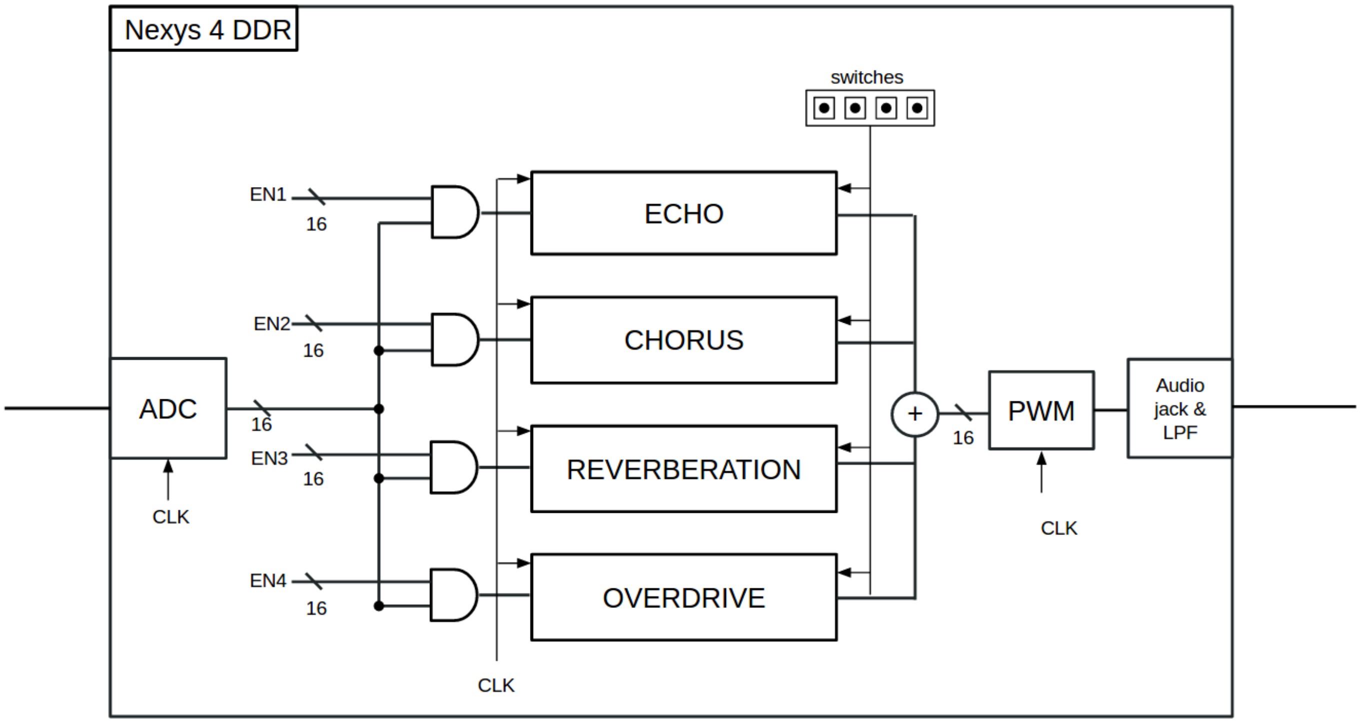

Fig. 4. Block diagram of PADAWAN System

spectral density (PSD) of the noise. The shaping has the objective of minimizing noise power in the audible frequency range by reducing the noise power in these ranges. A relevant trade-off is the need to increase noise PSD in some other frequency band - in this application, the noise needs to be increased in the inaudible, higher frequency range.

C. Advanced design

In the ideal, advanced solution to the problem of real-time digital audio processing, an analogue voltage signal can be prefiltered to condition the signal for sampling, ensuring that unwanted high frequencies do not interfere with the audible frequency range. The signal is attenuated to fall inside [0:1]V, the range of the on-board ADC. The desired sampling rate is 44.1KHz. This results in a stream of amplitude values that is be concurrently filtered. The effects that are implemented include of echo, chorus, overdrive (a type of distortion). Had time permitted so, more complex effects such as phaser and flange would have been implemented.

Each filter has its own digital transfer function which can be translated to a difference equation. This results in the output of the current audio sample being the weighted sum of some number of past samples and the current sample. Noise present in the filtered digital signal can reduced using dithering. The outputs of each of the parallel filters are then added and normalised (to prevent overflow) to give a single output value.

Input signals from the user interface (i.e. buttons and switches) are be used to select and dynamically change effect and filter parameters. Two options were considered to allow real-time changes in parameters. The first is storing a table of pre-calculated filter coefficients with different cutoffs or other features, and dynamically selecting the desired filter tap weights based on a user input. An alternative is to calculate filter coefficients dynamically by using linear interpolation of existing coefficients at known cutoffs. This method is more efficient, but filter performance suffered. The chosen method was the storage of different filter coefficients, the values changing on user input, i.e. button presses.

After all audio effects are applied, the digital audio stream is converted to a PWM waveform. The mono audio output jack on the Nexys 4 has a built-in 4th order Butterworth low pass filter. Figure 5 shows the conversion performed by the on-board filter.

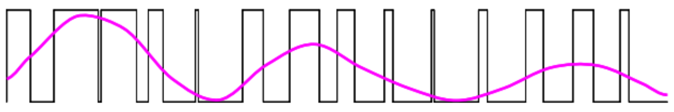

Fig. 5. Effect of integrating a PWM signal to obtain a continuous analogue signal [6]

The PWM waveform is fed to the onboard low pass filter input which effectively integrates it to form an analogue voltage waveform that is played on an external speaker connected to the mono audio jack socket of the Nexys 4.

D. Simplified design

In the case that the filtering fails to meet real-time requirements, i.e. the system latency is perceptible to the human ear, or the project falls behind schedule, some requirements could be relaxed. The options are:

1) Reducing the order of filters, and limit numbers of filters that can be active at a time
2) Reducing the sampling resolution to 8 bits.
3) Deprioritising noise shaping, abandoning the attempt to increase the apparent output resolution.
4) Reducing the sampling frequency to 30 kHz, as a last resort.

Of these options, only one requirement was relaxed, i.e. noise shaping.

IV. Methodology

A. Audio input

The audio input was of the form of an analog waveform, fed using an auxiliary cable from a host device streaming the audio file. Before being digitised by the Nexys 4's on-board ADC, the signal had to be conditioned for sampling. To meet the Nyquist criteria, a low pass filter was needed to ensure that all frequencies above 20kHz (maximum audible frequency) were attenuated. Furthermore, the ADC of the device has an operating range of [0:1]V in unipolar mode, and thus attenuation of the input signal was necessary. It was discovered that the output voltage from a host aux port is within the range of 1-2V_{pp}, ranging from -1V to 1V. Thus the input signal was first given a DC offset of 1V, and then fed into a voltage divider to attenuate the voltage by a factor of 2.

Thereafter, the conditioned input signal was fed into the ADC of the device. The ADC was set to sample at 44.1 kHz. The ADC produces 12-bit samples from the analogue waveform, however, the Nexys 4 makes use of 16-bit words, thus, zeros were appended to the 4 least significant bits of the sampled value. The sampled values were now ready for for storage in the DDR RAM as well as digital filtering.

The DDR RAM on the Nexys 4 DDR has a very fast read and write period compared to the sampling frequency of 44.1kHz. This enabled the storage of the current audio sample as well as the retrieval of 15 past audio samples to use in the digital filtering stage. The 128MiB storage space is also more than enough for the 1 second, or 44100 samples of audio stored in the prototype design.

B. Digital filtering

Upon receiving samples from the ADC, the Nexys 4 then fed current and previous sample values to each of the implemented digital effects. All past samples had to be fetched from the DDR2 RAM. The following effects were implemented:

1) Reverberation: Reverberation is essentially a dense series of decaying echoes. Reverberation was achieved with the following configuration:

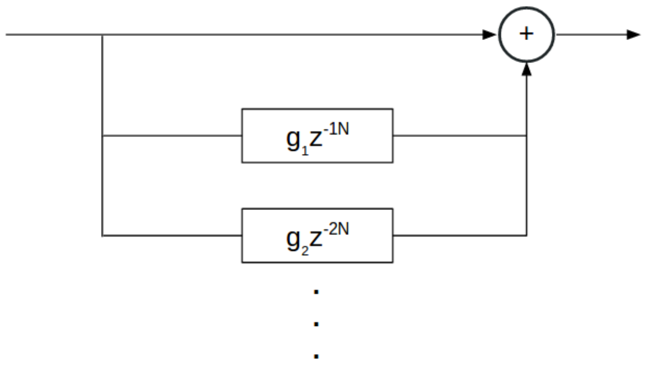

Fig. 6. Filter configuration for reverberation [7]

The configuration represents a comb filter (feed-forward loop). This was essentially the current sample multiplied by past samples, represented as z^{-N}, modulated with a decaying exponential, in the form of filter coefficients, g_N. The decay speed of the exponential determines the strength of reverberation present in the output.

The code shown in listing 1 was used for reverberation.

Since the Nexys 4 does not use floating point precision, multiplication by a floating point was done by first multiplying by an integer and then dividing by another integer, the quotient of which produce the desired floating point coefficient. Normalisation of the output of the addition of samples was performed to assure the result would not cause an overflow in the 16-bit output register.

```
module reverb(
    input clk,
    input [15:0] sample0,
    input [15:0] sample1,
    input [15:0] sample2,
    input [15:0] sample3,
    input [15:0] sample4,
    input [2:0] decay_fact,
    input [15:0] current,
    output reg [15:0] reverb_out
);
integer out=0;
always @ (posedge(clk)) begin

      out=current+(95*sample0)/100+(50*sample1)/100+(20*sample2)/100+(5*sample3)
       /100+(2*sample2)/100;
      reverb_out = (100*out)/272;


end
endmodule
```

Listing 1. Verilog code for reverberation

2) Overdrive: Overdrive is a distortion effect. Overdrive was implemented by means of hard clipping, or thresholding of the signal, as displayed below:

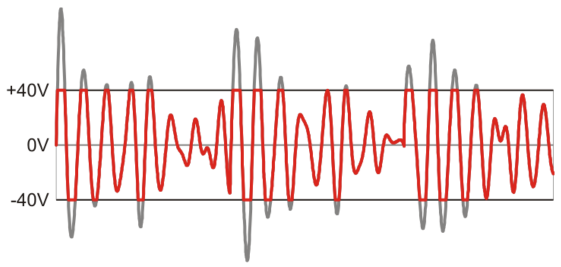

Fig. 7. Signal clipping

For a unique overdrive effect, it was decided that instead of implementing the standard hard-clipping, any value above the given threshold would be set to the maximum value, rather than the threshold value, creating a louder and rougher distortion effect for maximum overdrive. The code for overdrive is shown below:

```
module overdrive(
    input clk,
    input [15:0] current,
    output reg [15:0] overdrive
    );
always @(posedge(clk)) begin
    integer out=0;
    if(current>45000) out=65000;
    else out=current;
    overdrive=out;
end
endmodule
```

Listing 2. Verilog code for overdrive

3) Echo: Echo is a simple audio effect, in that it is just a delayed, reduced amplitude version of the signal mixed with itself. The echo delay calculation is as follows:

$$d[n] = \frac{1}{f_s} \times D_s \tag{3}$$

where d[n] is the delay in seconds, f_s is the sampling frequency and D_s is the delay in samples. Finally, echo was created as shown in the figure below.

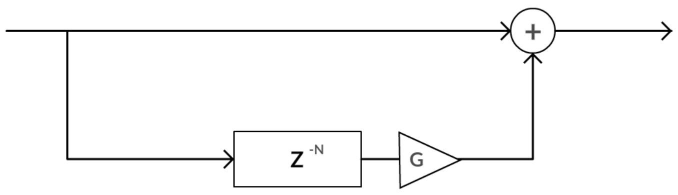

Fig. 8. Filter configuration for echo

The code used for echo can be seen in listing 3 below.
A right shift by one bit, equivalent to division by 2, was performed on the output variable, out, before sending it into the echo output register. This was to normalise the output to 16-bit to ensure it did not overflow the output register. A shift was used instead of division as shifting is a lot faster, and far more easily implementable on an FPGA than division.

```
module echo(
    input clock,
    input [15:0] delay_samp,
    input [15:0] curr_samp,
    output reg [15:0] echo
    );
    integer out = 0;
    always @ (posedge(clock)) begin
        out = curr_samp + delay_samp;
        echo = out>>1;
    end
endmodule
```

Listing 3. Verilog code for echo

4) Chorus: Chorus is an effect which takes one sound and makes it sound like many sounds. This was implemented by adding several delayed versions of the sample with itself to give a chorus like effect, with the constraint that the delay must fall within a delay interval of 10-25ms.

Such an effect can be implemented as shown in the figure below:

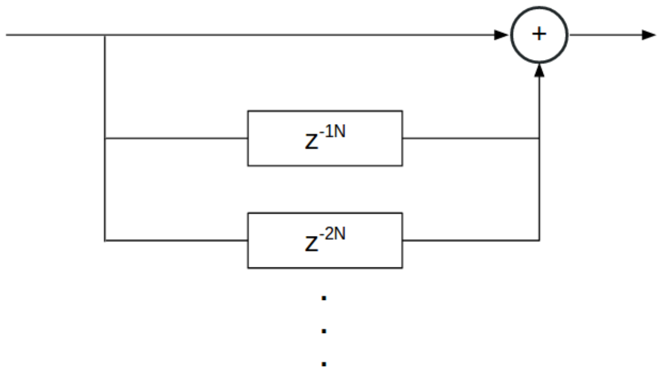

Fig. 9. Filter configuration for chorus

Chorus is similar to reverb in that it uses a summation of equally spaced apart delayed samples, however, chorus does not make use of a decay which prevents the 'bouncing' effect caused from reverberation. The code used for chorus is shown in the listing below.

```
module chorus(
    input clk,
    input [15:0] sample0,
    input [15:0] sample1,
    input [15:0] sample2,
    input [15:0] sample3,
    input [15:0] sample4,
    input [15:0] current,
    output reg [15:0] chorus_out
    );
    integer out=0;
    always @ (posedge(clk)) begin
        out=sample0+sample1+sample2+sample3+sample4+current;
        chorus_out=out/6;
    end
endmodule
```

Listing 4. Verilog code for chorus

C. User input

All effects, save chorus, were made adjustable by means of buttons found on the Nexys 4. Each effect was given 5 'states' that it could be in at any given time. The state represents a different strength in the effect, with state 1 being the lowest strength state, and state 5 being the highest strength state. Each effect was allocated a button where, when pushed, incremented the state register for the effect, which altered the effect strength. The state of each effect was also displayed on the seven segment display on the Nexys 4, which allowed the user to observe which state the effect was in, and hence, allowed the user to gauge the current strength of the effect. The code for the state control is shown below.

```
if (btns[`LEFT] == 1) begin  //LEFT pressed
  echo_state = echo_state+1; //increment echo state
  if (echo_state > 4) echo_state = 0; //cycle state
end

else if (btns[`RIGHT] == 1) begin //RIGHT pressed
  reverb_state=reverb_state+1;
  if(reverb_state>4) reverb_state=0;
end

else if (btns[`UP] == 1) begin //UP pressed
  overdive_state=overdive_state+1;
  if(overdive_state>4) overdive_state=0;
end
```

Listing 5. Verilog code for effect control

D. Noise shaping

Due to time constraints, it was not possible to getting a working implementation of noise shaping. Noise shaping, and how it could have been implemented, will be discussed in the conclusions and recommendations section of the report.

E. PWM output

Once the audio signal had passed through the filters, it was modulated using PWM. The PWM waveform had a frequency equal to that of the sampling frequency, 44100Hz. As stated above, the clock speed requirement for a 12-bit, 44.1kHz PWM is almost two times higher than that of the FPGA's clock. Thus, the final bit of the signal will be truncated, converting the samples into 11-bit samples. This reduces the clock-speed requirement by a factor of 2.

The PWM wave created from the audio stream was then fed to the mono audio jack of the Nexys 4, which contains a 4th order Butterworth low pass filter. The low pass filter acted as an integrator for the PWM signal, yielding the average of each cycle. This ultimately reconstructed the audio signal, which was then outputted to external speakers. The code for the PWM output is shown below.

```
module pwm_module(
input clk,
input [10:0] PWM_in,
output reg PWM_out
);
reg [10:0]new_pwm=0;
reg [10:0] PWM_ramp=0;
always @(posedge clk)
begin
    if (PWM_ramp==0)new_pwm<=PWM_in;
      PWM_ramp <= PWM_ramp + 1'b1;
      PWM_out<=(new_pwm>PWM_ramp);
end
endmodule
```

Listing 6. Verilog code for PWM output

F. Golden Measure

For the golden measure, all filters and effects mentioned above were coded in octave, and performed in sequential fashion. A random array of 44100 random values between 1 and 65535 values was created, to simulate the one second of samples stored in RAM. Effects were then performed using the different delayed samples from the created array. The total time taken for an output to be generated after all filters were completed was measured. The experiment was repeated 1000 times in a loop and the average time was taken. The code for the golden measure is shown below.

```
function [Time] = filters ()

  wave=rand(44100,1)*65535;     #"16-bit audio file"
  current=wave(1);
  out=0;
  Time=0;
  for i = 1:1000
    tic;

    out+=(current+0.95*wave(100)+0.7*wave(200)+0.5*wave(300)+0.38*wave(300)+0.28*
       wave(400))/(1+0.95+0.7+0.5+0.38+0.28); #reverb
    out+=(current+wave(441)+wave(621)+wave(800)+wave(980)+wave(1160))/6;#chorus
    if(current>45000)                     #distortion
      out+=65000;                     #distortion
    else                          #distortion
      out+=current;                   #distortion
    endif;
    out+=current+wave(22050);               #echo
    out=out/4;                        #normalisation
    Time+=toc();                      #measure time
  endfor
  Time=Time/1000;
endfunction
```

Listing 7. Octave code for golden measure

The Verilog code for the Nexys 4 was then simulated in Vivado to observe the timing of signals, and amount of time taken for the filters to produce an output. The golden measure and simulation performances were then compared to obtain the speed-up of the PADAWAN system.

V. Proposed development strategy

The PADAWAN frees up the processor on the host system by handling audio filtering tasks. These filters can be made configurable and can be expanded to operate on multiple channels simultaneously. If the PADAWAN were to be developed into a commercial product, it would need a female audio jack input, as well as an Ethernet port to send and receive digital data. An application programmer interface (API) will need to be developed to enable reconfiguration of the PADAWAN by the end user. This API will accept user level input parameters and use these to modify or select preconfigured Verilog code using a specialised compiler or look-up table. The altered code is then compiled on the host machine and sent as a bit stream to the PADAWAN. A GUI on the host computer will be a visual wrapper of the custom API allowing standard users to reconfigure the PADAWAN.

A. Commercial Uses for the PADAWAN

The PADAWAN can either be used as a standalone audio processor for use with microphones, instruments and other devices that output analogue audio via a standard 3.5mm auxiliary jack. The PADAWAN also accepts 5.1 channel audio using 3 stereo 3.5mm inputs connected to GPIO ports on the FPGA. The volume of the output can be adjusted using a knob connected to a potentiometer. Additionally, sliders will be used to allow the selection of different tap weights for each of the implemented digital filters.
The PADAWAN can also be reconfigured in more detail using a GUI application running on a host computer such as a PC. A user can implement and choose custom effects to port onto the device and the software will generate the required code and filters to implement the effects. In addition to changing the number of audio channels needed, the sampling rate and bit depth of the audio can also be specified. If analogue output is needed, the effective bit rate will be limited, due to the 100MHz clock speed of the Artix-7 FPGA.

B. Proposed commercial design

A custom board similar to the Nexys 4 DDR with an Artix-7 FPGA, an Ethernet port, 3 input and and 3 output 3.5mm audio jacks together with knobs and sliders is proposed. The custom board will also need memory, in the form of DDRAM. A controller interface is needed to interpret digital data received via the Ethernet connection. This data can either be audio or configuration commands. The configuration commands will be in the form of a bit stream that will be sent via JTAG to the Artix-7.
The PADAWAN will have three modes of operation the first of which is reconfiguration mode. Depending on user specified variables, at the end of the reconfiguration process, new digital filter tap weights are computed and synthesised on the Artix-7. The I/O mode is also chosen as either digital or analogue. In digital input output mode, input audio is sent from a host computer via Ethernet. The digital audio is then filtered and sent back over Ethernet. In digital mode, a PWM signal is not generated and no ADC is necessary, allowing for the filtering of higher fidelity sound e.g. 16 or 24-bit audio.
In analogue mode, mono, stereo or 5.1 channel audio is fed to the system, the audio is sampled, filtered and output on the separate output auxiliary jacks.

VI. Experiments and results

Octave source code that performs the filtering was implemented and used as a golden measure for performance, by executing the processing on a desktop computer. After implementation, the performance of this golden measure was compared to the wall clock time of our FPGA implementation. The golden measure implemented all filters in sequential fashion, timing the amount of time it took to perform all filter calculations on a single sample. The experiment was repeated 100 times and an average of 54μs. This was actually found to be too slow for real-time audio processing, where filtering calculations need to be performed within the space of 1 sampling period, which is 22μs for 44100Hz. On the other hand, through timing simulations of the Verilog code, it was found that to perform all filtering operations, the simulation took 729ns. This results in a speed-up of 74 over the golden measure. Moreover, this is more than adequate for the real-time constraint of audio processing. This is mainly due to the massively parallel operation of the filters, where not only were the filters parallelized, but the operations within filters as well.
To ensure that the PADAWAN outputs an appropriate analogue waveform to the mono audio output, we used an oscilloscope to plot the output signal and compare it to the input signal with all filters disabled. Because the input and output waveforms matched closely, the modulation and integration of the PWM signal was

verified as correct. Thereafter, the dithering and noise shaping of the input signal would have been tested by means of a comparison. The dithering would be initially tested by comparing the output signal with dithering to that without, and all filters disabled. If the harmonics and distortion found within the non-dithering output was found to be eliminated in the output that used dithering, then the dithering would be proved to work correctly. The noise shaping of the dithering can then be checked in the same manner, where outputs are compared. If the noise level is lower at specific frequency bands in the noise shaped output, compared to the unshaped output, then the noise shaping will have been implemented correctly.

Furthermore, specific filters used to create audio effects such as echo, reverberation, chorus and overdrive were verified by means of a comparison to an already existing effects generator on a computer. The same audio effects implemented on the PADAWAN system were created on a computer and compared to the effects generated by the PADAWAN system. The audible similarity of the computer and PADAWAN effects determined that the filters were successful in their ability to create the desired effects. The dynamic adjustment of the filters was tested by simply adjusting them, and verifying that clearly audible changes to the audio effects occurred.

VII. Conclusions and Recommendations

The final step in the development of the PADAWAN was the implementation of a Verilog prototype synthesised on the Nexys 4 DDR. A main Verilog module initiating the XADC with the correct parameters as well as specifying the pin assignments and module interconnects was developed. The audio effects were simulated in Octave and implemented as individual Verilog modules. The Octave simulations also served as a golden measure for performance comparison. A PWM Verilog module taking the sampling rate and bit depth into account was also developed.

User input peripherals were used to enable the dynamic selection and adjustment of audio effects. Once all the Verilog modules had been written and the signals routed between them correctly, the design was be synthesised and tested. After ensuring that the output was as expected from simulations using an oscilloscope and that the switch and button interface affected the output, the performance of the design was evaluated. There was a significant speedup in the filtering tasks, implying that the PADAWAN can be accurately referred to as a parallel hardware accelerator.

The PADAWAN device can be improved in many different facets. Firstly, stereo or surround sound would make the device much more marketable. This would involve parallel filtering of the different audio channels, and implementation of custom low-pass filters, either high-order analog filters or digital filters implemented in Verilog. The low-pass filtering is necessary for the audio output, since the device generates a PWM output, which needs to be converted to an anologue voltage. Secondly, the functionality of the device can be expanded by offering the feature of a sound equaliser, so that the user can use sliders to boost different audio frequency ranges. This can be enhanced further by the addition of the ability to apply effects on isolated bands of interest, making the device suitable for applications such as denoising a wide range of analog frequencies by isolating noisy bands. Finally, the initial plan was to implement noise shaping to change the power spectral density of the output signal such that the noise power was reduced in audible frequency ranges, and correspondingly increased in higher frequencies. This can be a starting point if an effort is made to improve the PADAWAN.

References

[1] "Audio effects, http://www.mediacollege.com/audio/effects/,"

[2] "Dithering and Noise Shaping, http://downloads.izotope.com/guides/izotope-dithering-with-ozone.pdf,"

[3] "Design of a Reusable Distributed Arithmetic Filter and its Application to the Affine Projection Algorithm, https://smartech.gatech.edu/bitstream/handle/1853/28199/lo_hawjing_200905_phd.pdf,"

[4] "Nexys-4-DDR-Music-Looper development, https://github.com/Digilent/Nexys-4-DDR-Music-Looper, date accessed 2017-02-16,"

[5] "SRAM to DDR Component [Reference.Digilentinc], https://reference.digilentinc.com/learn/programmable-logic/tutorials/nexys-4-ddr-sram-to-ddr-component/start urldate:2017-06-24."

[6] "PWM Limitation, http://www.ti.com/lit/an/slaa405a/slaa405a.pdf,"

[7] "Reverberation, https://christianfloisand.wordpress.com/2012/09/04/digital-reverberation/,"