YODA: PADAWAN

Parallel Accelerator for Digitising Audio With Attenuation of Noise

Lloyd Hughes, Stephen Jermy, Ross Engers
Group 18A
EEE4084F Final Report • UCT • 17 May 2012

Table of Contents

Introduction 1

Problem specification 1

Report Structure 1

Methodology 2

Skill Identification 2

Hardware: 2

Software: 2

Design methodology 2

Design Objectives 3

Plan A 3

Plan B 4

Issues with Design Criteria 4

Redefining Design Criteria 4

Testing 6

System breakdown 6

Input Buffer 6

Filter Stage 7

Output Stage 7

Evaluation Procedure 8

Results 9

Conclusion 12

Appendix 13

Bibliography 15

Introduction

PROBLEM SPECIFICATION

The YODA project is designed to have a final outcome of a working digital accelerator, which is used to accelerate a common task by passing the work load off to a specialised piece of hardware.

This accelerator will be achieved by exploiting the parallel nature of certain tasks within the chosen topic. The solution will be implemented on dedicated hardware, in this case using an FPGA.

The PADAWAN project will solve the problem of slow sequential audio filtering that exists when equalising and adding effects to large audio signals. These filtering operations are usually run sequentially which leads to dead time in the output signal, which is undesirable for live audio streaming/performances.

REPORT STRUCTURE

This report will focus on an investigation into the tools and techniques required to successfully implement our audio accelerator.

An analysis of the required skills will be performed and our design methodology will be laid out. This methodology will specify our project's objectives; this involves laying out a complete plan of action that would represent our ideal outcome, and a scaled-down plan with reduced risks and less costly in terms of time.

Potential problems that we anticipated, as well as encountered, are discussed and possible solutions are described.

A PC-based golden measure is created to simulate likely results from a standard, "unaccelerated", system. The golden measure is used as a baseline for the performance of the accelerated system.

Results of the PADAWAN system are shown, discussed and compared to the golden measure.

Methodology

SKILL IDENTIFICATION

Several skills are necessary to correctly implement our audio accelerator in hardware, as well as to allow communication between our hardware and PC platforms. Furthermore, we will require knowledge of PC software programming to create a gold standard.

Hardware:

The skills we will require to implement the hardware design are:

- HDL Programming (VHDL and Verilog)
- Schematic Design
- Communication Protocol Design
- FPGA board proficiency

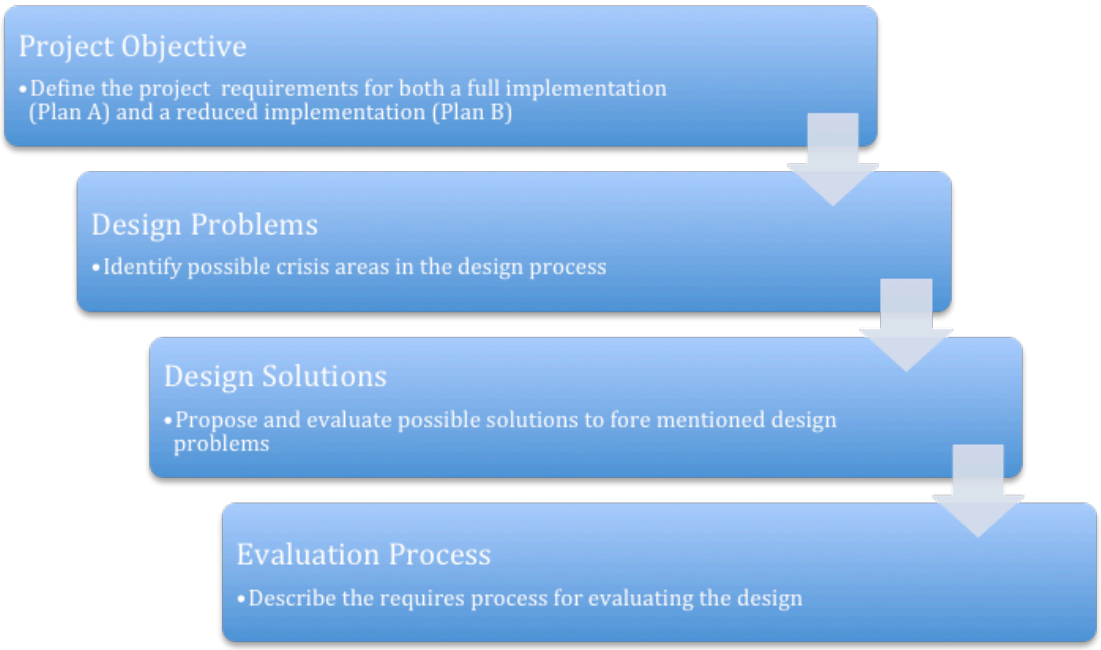
Software:

The skills we will require to implement the software design are:

- C programming
- Matlab programming
- DSP algorithm design
- Communication Protocol Design
- Linux/Unix

DESIGN METHODOLOGY

To develop the PADAWAN project from a theoretical basis into a fully working system, we will need to explore the exact design requirements and design processes to be used during system development. The process diagram below, illustrates the design methodology we will follow.



Design Objectives

Two sets of design objectives will be laid out below. The reason for this is due to the limited time span of the project a complete implementation might not be possible. It is thus important to layout design objectives for a reduced, proof of concept, design.

The main idea of PADAWAN is to produce a hardware accelerator that can allow mass parallel audio/signal processing and conditioning in realtime. This is important as many industries rely on the ability to remove noise from their signals before further processing is done.

This signal filtering and conditioning can be done entirely in software, however, for large amount of signals and for many filter this becomes slow and can introduce a lot of dead-time into a system. Dead-time is bad as it means a signal is no longer being processed in realtime.

The PADAWAN project will focus on the implementation of signal filtering and conditioning on audio signals only. This however, can easily be expanded to other signals.

Plan A

We aim to create a system with the following specifications:

- Bandwidth 22kHz Sampled @ 44.1kHz mono
- 30 channels
  7 present effects per channel
- Fully customisable effect and filter parameters
- 1 gain stage per channel
- Cascading and stacking of effects in channels
- Interfacing via Ethernet with a PC
  - Receiving real-time digital audio stream
  - Returning digital audio to PC
  - Optional speaker output

Plan B

We aim to create a system with the following specifications:

- Bandwidth 10kHz Sampled @ 20kHz mono
- 4 channels
- 1 gain stage per channel
- Gain customisation for each channel
- Full equaliser capabilities
- Interfacing through PMod A/D and DAC

Issues with Design Criteria

From our initial Plan A design we can locate a number of possible implementation issues. These issues are listed below:

- A fast sampling time is required to be able to handle the larger bandwidth signals.
- Non-linearities of some effects can cause parallel processing issues due to the need for past samples.
- Filter and effect customisation will require a lot of storage space for all the parameters.
- Cascading effects requires advanced routing hardware, which could have a big overhead.
- Ethernet interface could be hard to configure and get working correctly.
- Ethernet protocols could cause packet delays, thus effectively removing the ability to perform realtime processing.

These possible problem areas have been removed from the Plan B design requirements, to ensure that our Plan B design can be implemented with a minimal number of issues.

Redefining Design Criteria

In order to reduce the number of issues and the time of development, the following solutions, to the design issues, have been proposed:

The sampling time was seen as an issue; however, if a normal PC sound card is used to sample the audio there will be no issues with sampling time as these cards are designed to handle audio sampling up to 48kHz. The only problem will be the fact that the communication between the PC and the PADAWAN hardware will cause too much of a delay.

An alternative solution is to use a PMod for the Nexys3 boards, which will contain a 100MSps ADC, thus allowing us to obtain samples directly from the Nexys board.

In order to solve the problems with non-linear effects, we can either limit the effects to only allow linear, time invariant operations or we can approximate the non-linear filters by linear operations. The latter is a better option, as it will preserve the flexibility of the PADAWAN platform.

The issues regarding customisable filters can be addressed by limiting the number of filter taps in the FIR filters to a reasonable amount such as 10 taps per filter. Each of these taps can also be scaled into a relative range instead of absolute thus meaning that simple 16bit integers can be used to describe them.

The same can be done for effects parameters. This will minimise the amount of storage space required.

These filter and effect parameters can be altered via the Ethernet connection, which can be used to update an area of memory. All the filter parameters will be memory mapped to simplify the implementation of this feature.

If all filters are linearized, then the properties of linearity can be exploited to simplify the routing protocols, as a chain of filters can be implemented in any order. This means that all the filters can be chained together in a set order, and should one not be required a simple multiplexor with a bypass flag can be used to bypass the filter action.

The final problem that needs to be dealt with is that of implementing Ethernet successfully. The Ethernet standard is a complex protocol and will require a lot of effort to design and implement correctly. To get around these issues we will use a prebuilt Ethernet core from openCores.org. These cores are released under the GNU GPL license agreement and are thus royalty free. Using a tried and tested implementation should hopefully reduce the amount of debugging needed to implement the standard.

An alternative approach is to make the PADAWAN device, a stand-alone device. This will completely remove the need for Ethernet and thus simplify the design process. However, the customisability will also be affected. This approach has been taken in our Plan B implementation, as it simplifies the design requirements to ensure the deadline can be met.

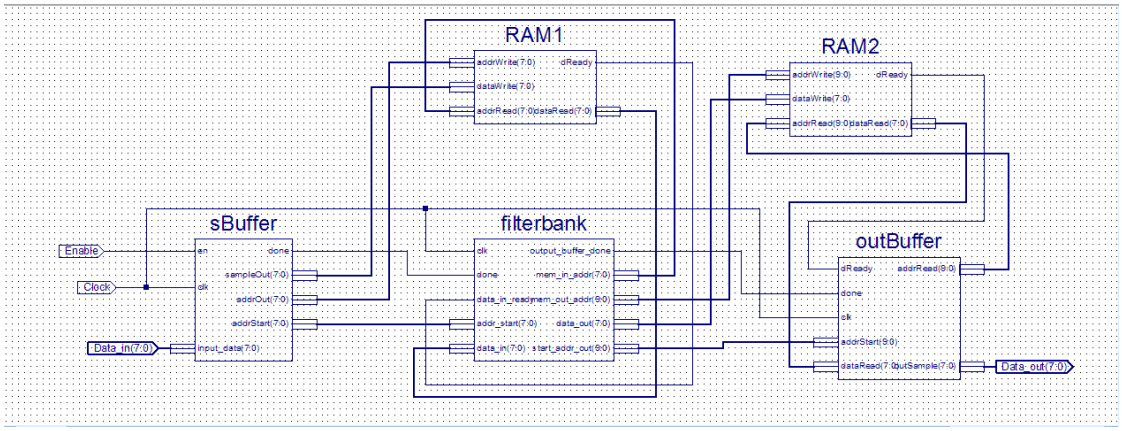
Testing

SYSTEM BREAKDOWN

In order to be able to design and implement a successful testing scheme we decided to break our system down into smaller, manageable chunks. Each of these chucks will be implemented and tested individually to ensure they perform as expected, once their operation is confirmed as correct we will combine the various elements into the completed system.

The final system will then be subjected to tests and evaluated in accordance with our gold standard.

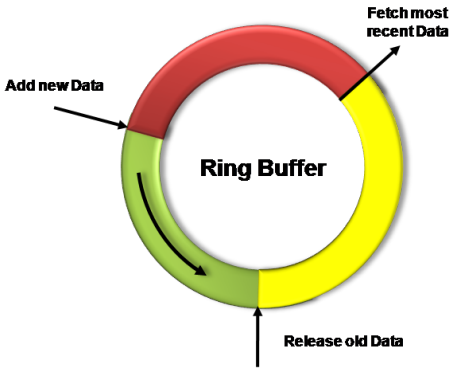
The system was broken down into 3 stages as shown in the schematic of the system, each of these stages is described in more detail below:



Schematic of fully implemented PADAWAN prototype

Input Buffer

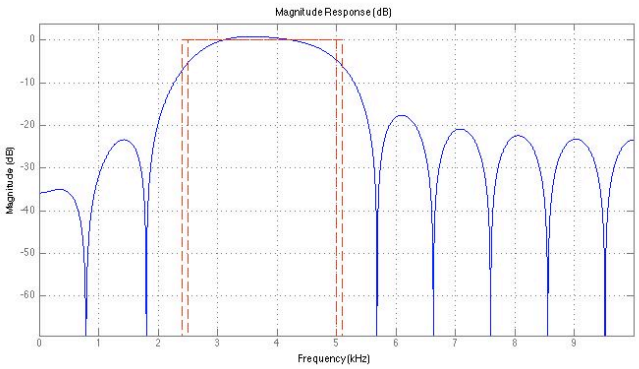
The input buffer stage consists of the section of hardware that takes in the input signal, sample by sample, at a rate of 20kHz and stores each of these samples into a cyclic buffer. A cyclic buffer is essentially a special case of a FIFO queue.



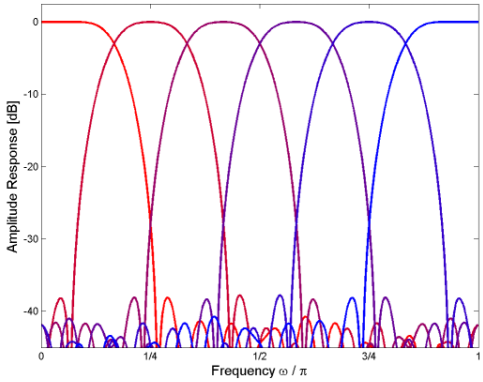
In the case of our input buffer stage, the incoming samples were stored into a background buffer, while the filter stage was working on the previous sample buffer. When the input stage read in 128 samples into the buffer it would notify the filter stage that a new sample set is ready, swap buffers, release the already processed data and then read the next 128 samples into that buffer. Thus the filter stage and input stage are never accessing the same segment of memory simultaneously.

Filter Stage

The filter stage was responsible for channelising the sample buffer. In the prototype the filter stage would split the incoming audio stream into 4 individual channels by means of a bank of bandpass filters, as shown below. Note: the image below represents 6 channels, however, the same concept was used to implement 4 channels



Frequency responses of filter banks



Frequency response of one of our filters, 2.5kHz to 5kHz band

Each of these output channels was then written to a secondary memory block. This memory block worked in the same manner as the memory block used by the input stage.

Output Stage

The output stage was responsible for taking the output of the filter banks and recombining them into an audio stream. The purpose of this is to allow the next level prototype to be able to equalise, amplify or add effects to each channel

individually, and then output the altered audio stream out to a speaker or recording device. All alterations will happen by implementing further stages between the filter banks and the output stage.

The output stage reads the samples in from the buffer in memory that the filter stage is not using. These samples are scaled and added together in order to form a new 128 sample buffer. The output stage then clocks this buffer out at a rate of 20kHz.

EVALUATION PROCEDURE

In order to assess the performance of the full PADAWAN device, we implemented a gold standard in Matlab, using the same filter that we implemented in VHDL. This gold standard took in a waveform which consisted of 4 sinusoidal components, one centred in each filter.
The sinusoidal components were placed at frequencies, 200Hz, 3kHz, 6kHz and 8kHz.

Processing 4 of the previously mentioned filters in serial would require 128 add operations and 129 multiply operations for a grand total of 16 512 operations per frequency division. Therefore there are 66 048 sequential operations per sample. This means that the golden measure requires $O(N^2)$ operations and will execute in $4 \times O(N^2)$ / Average number of operations per second per sample.

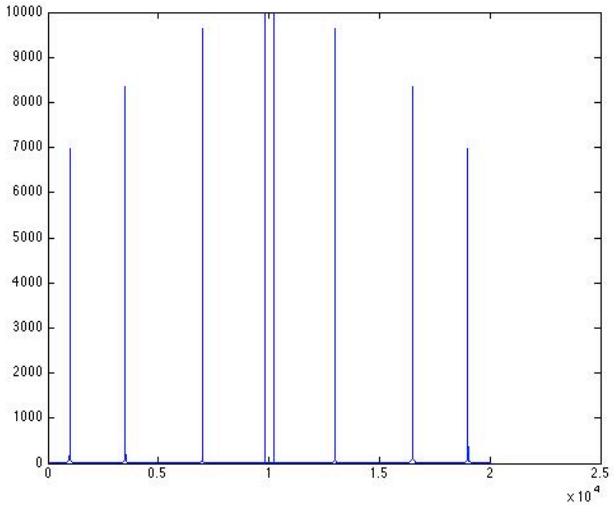
The PADAWAN system reduces the complexity of each filter to O(N), this is because the multiplications are independent of each other and can be parallelised. The array of filters can also be processed in parallel allowing all 4 of the above mentioned filters to be processed concurrently. Thus reducing the total execution time to O(N) / Average number of operations per second per sample. The speedup factor will be influenced by the operation speed of the FPGA chip used, which is influenced by its clock speed.

Assuming a 2GHz processor on the CPU, we approximated that our 50Mhz FPGA will perform operations 2000 / 50 = 40 times slower than the CPU. Using this rough approximation and the CPU time as the reference, its can be shows that for a $4 \times O(N^2)$ complexity golden standard solution, the equivalent complexity (compensating for the clock speed difference) would be 40 O(N). Therefore an N of 128 would yield a speed up factor of 12.804 per sample which is a total speed up of 12.804 x 128 = 1639.

Now that we have calculated our expected speedup, we then obtained our actual golden standard time by executing our golden measure in matlab. We then simulated our hardware design and compared the execution time of the simulation to that of our golden standard to obtain our actual speedup factor.

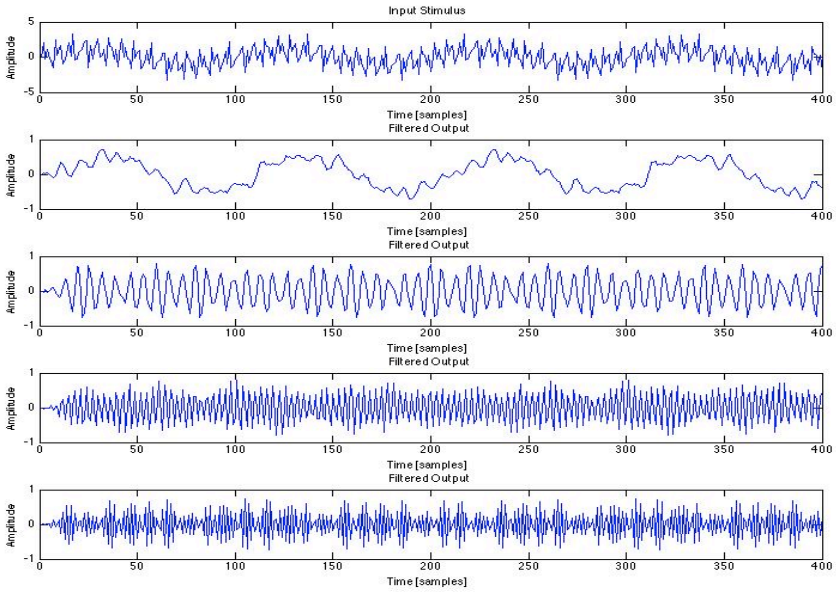
Results

The input stimulus, shown in figure below, was constructed out of 4 sinusoidal waveforms. Each frequency was specifically chosen to fall within the pass band of each of the filters. This waveform was used as our test case, which was used to test our golden standard as well as our design.



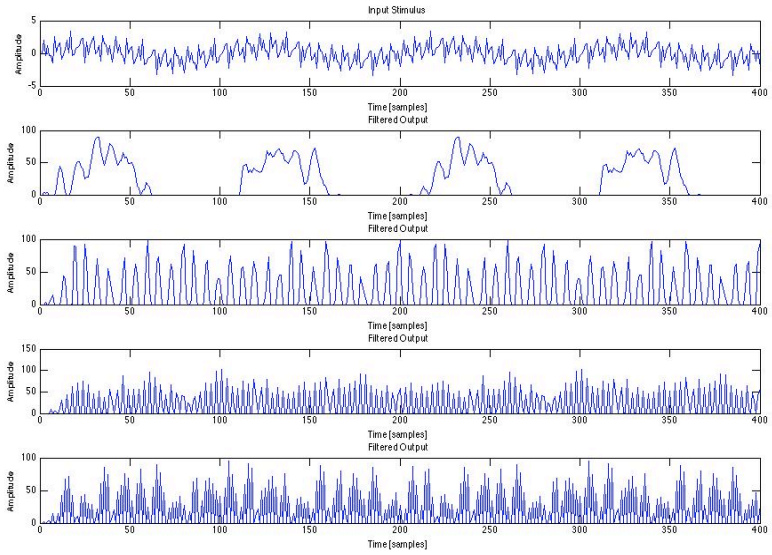
FFT of the original input waveform

The actual waveform, which can be seen in the figure below, was passed through the golden standard's filter bank; the results of which can be seen in the figure below. Each of these waveforms correspond to the output of each of the filters. Each filter was able to correctly extract the frequency in its passband, with a small amount of noise.



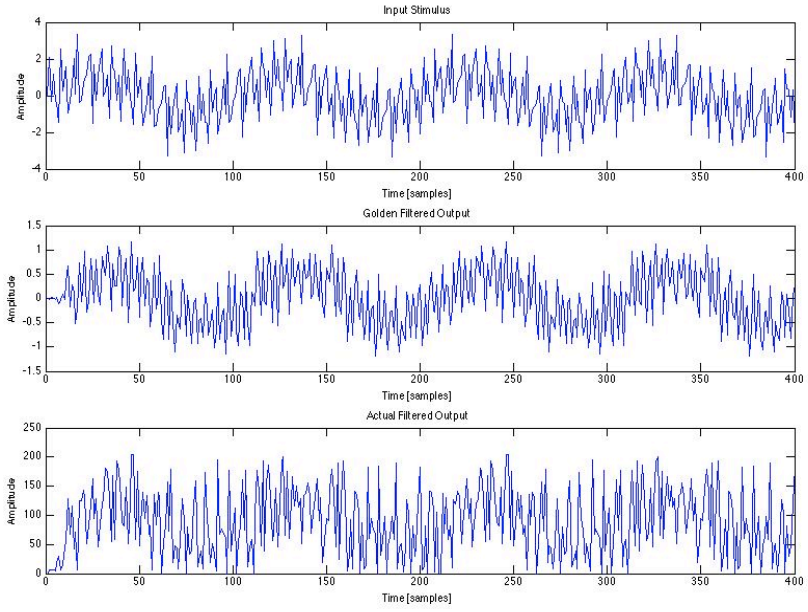
Golden standard filter outputs

With the results of the golden standard available, the test waveform was then passed through our design's filter bank; with the results shown in the figure below. Again each filter bank was able to correctly output the waveform in its passband. However with our design we used unsigned integers for our filters. This explains the discrepancy between the golden standard's outputs and our design's outputs, which shows only the positive values of the waveforms. In future development we would be able to change the filters to signed values which would remove this discrepancy between the two outputs.



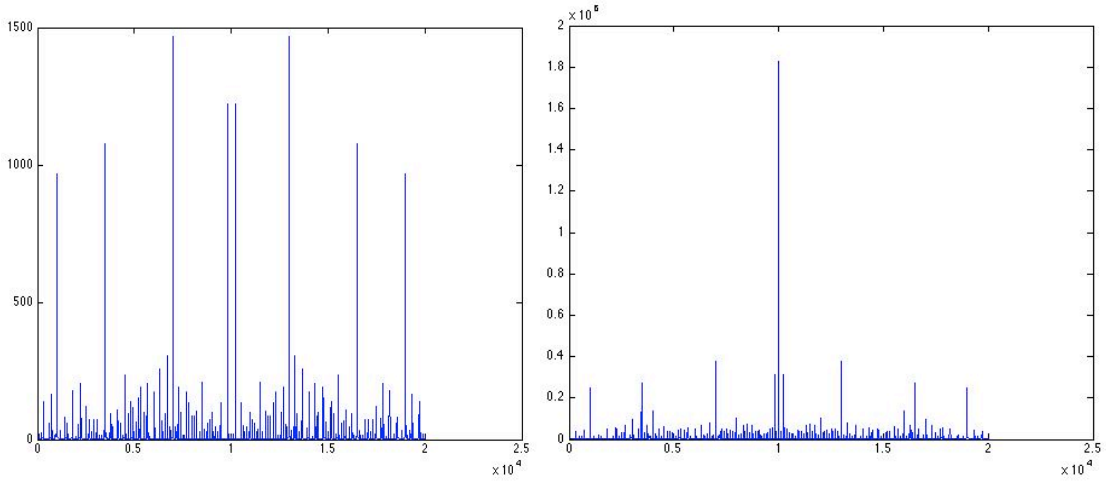
Filter output of actual implemented system

The outputs of the golden standard's filters, as well as our own, where added back together and the results of this addition can be seen in the firgure below. The golden standard closely matches the original waveform. Although our design's final output has some noise, due to the use of unsigned integers, it still matches the golden standard, and thus the original waveform.



Original input and reconstructed outputs of the gold standard and the actual system

The fast Fourier transforms, shown in the figures below, show the frequency spectrums of the golden standard's output and our design's output respectively. Both match the original waveforms FFT closely, with a small amount of noise, and a large dc signal in our output.



FFT of reconstructed outputs, gold standard (left) and actual (right)

The golden standard took approximately 11ms to process 128 samples while our design took only 12us per 128 samples. This gives us a speed factor of approximately 900.

Conclusion

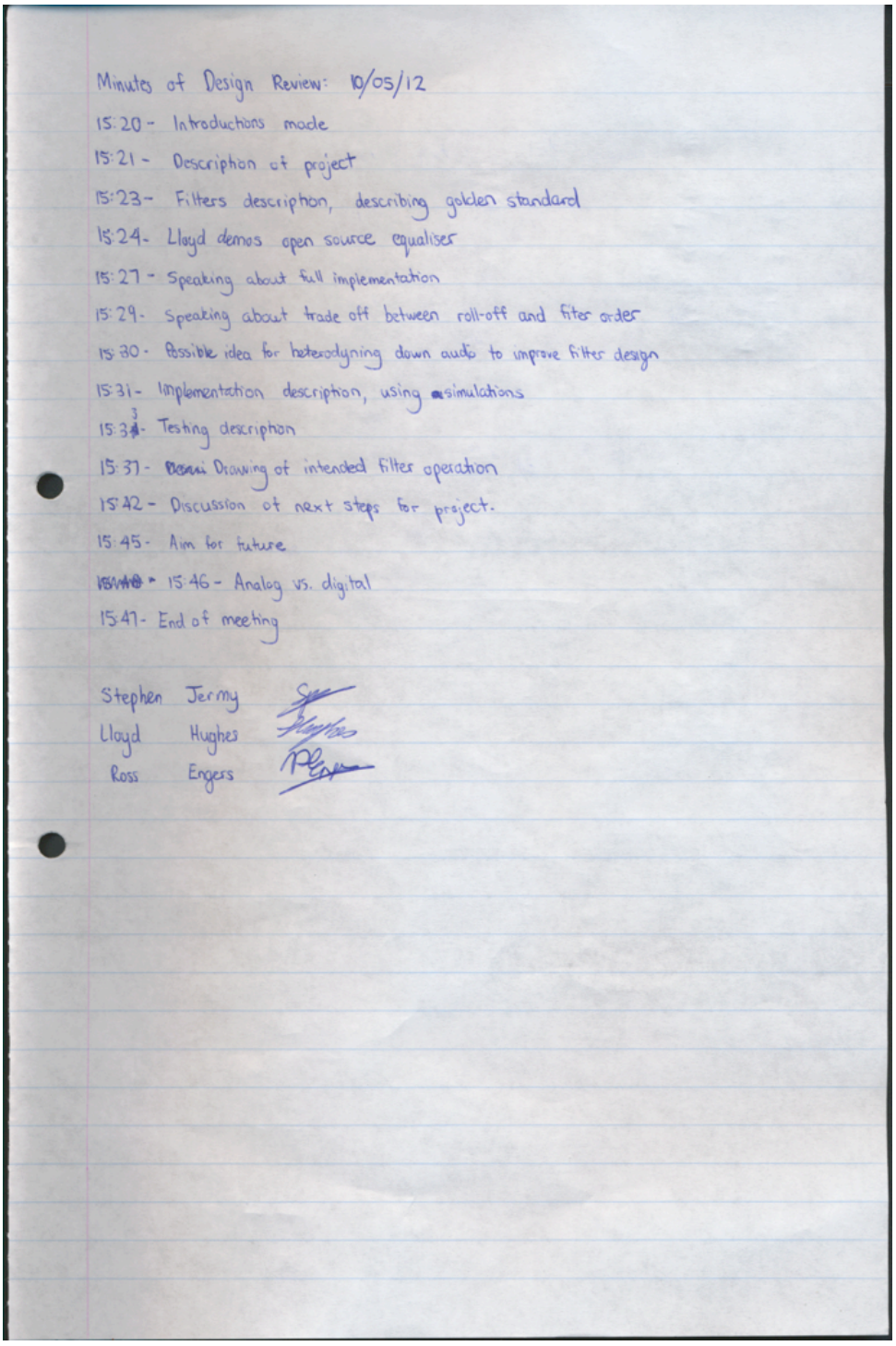
The output of our design, although there were issues with noise due to our use of unsigned integers instead of signed integers, matched the output of the golden standard quite closely and by extension, matched the original waveform.

We found that our speed factor was 900 however we expected a speed up of 1639. This speed up factor is acceptable as it is nearly 1000 times faster than the normal system.

The reason our speed up factor does not match the expected speed up was due to overhead caused by loading data into buffers at the beginning of each cycle as well as between each stage of the system. The expected speed up value was also quite a crude calculation as it assumed that all the operations had the same complexity, for instance that an add operation would take exactly the same amount of time on an FPGA as it does on a PC. Matlab also optimizes most of its function which means that its timing might be inaccurate.

We designed our system to be very modular meaning that it can be quite easily expanded to include more filters. As it stands our system would also need to have some of its variables changed to be fully compatible with the new filters however a future expansion could be to remove this necessity by increasing the size of the RAM module and changing the memory mapping system to match RAM's actual mapping system more closely.

Appendix



Minutes of Design Review: 10/05/12

15:20 - Introductions made

15:21 - Description of project

15:23 - Filters description, describing golden standard

15:24 - Lloyd demos open source equaliser

15:27 - Speaking about full implementation

15:29 - Speaking about trade off between roll-off and filter order

15:30 - Possible idea for heterodyning down audio to improve filter design

15:31 - Implementation description, using simulations

15:34 - Testing description

15:37 - Drawing of intended filter operation

15:42 - Discussion of next steps for project.

15:45 - Aim for future

15:46 - Analog vs. digital

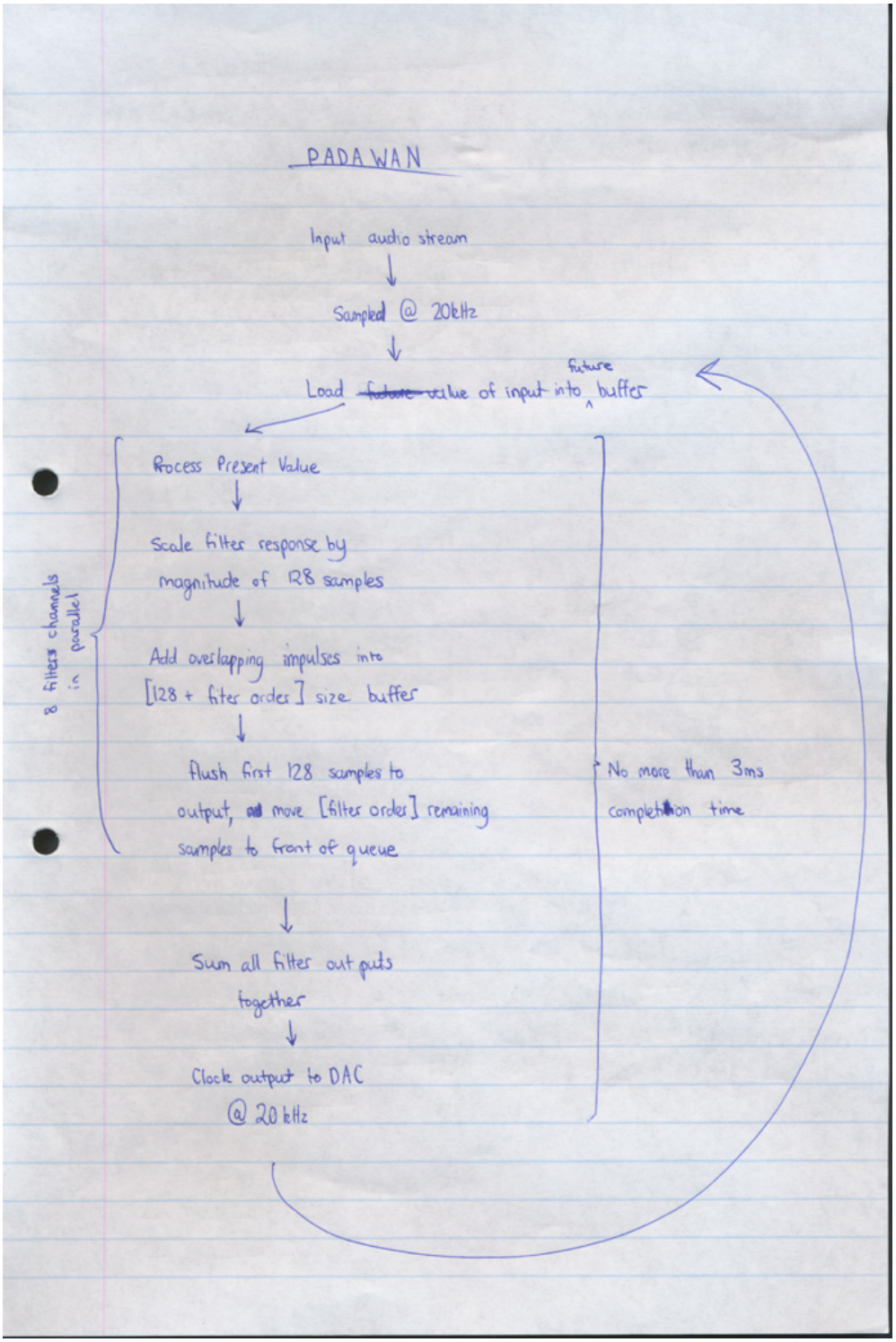
15:47 - End of meeting

Stephen Jermy

Lloyd Hughes

Ross Engers

Minutes of design review meeting



Flow diagram of system

Bibliography

1. **McClelland, Chris.** FPGALink: Easy USB to FPGA Communication. *MakeStuff.* [Online] 26 03 2011. [Cited: 28 04 2012.] http://www.makestuff.eu/wordpress/?page_id=1400.

2. *Implementing Digital Audio Effects Using A Hardware/Software Co-Design Approach.* **Pfaff, Markus et al.** Bordeaux : s.n., 2007. 10th International Conference on Digital Audio Effects.

3. **Schultz, Richard.** *FPGA Implementation of Audio Effects.* Electrical and Computer Engineering, University of Alberta. Alberta : University of Alberta, 2003.

4. **Shapiro, Andrew and Resnick, Marc.** *Field Programmable Digital Audio Effects Rack.* Massachusetts Institute of Technology. Cambridge : MIT Press, 2010.

5. **Micea, Mihai, et al.** *Implementing Professional Audio Effects with DSPs.* University of Timisoara. Romania : University of Timisoara, 2001.

6. **Smith, Steven.** *The Scientist and Engineer's Guide to Digital Signal Processing.* 2nd Edition. San Diego : California Technical Publishing, 1998.

7. **Tredennick, Nick.** The Death of DSP. *TTI Vanguard.* [Online] 08 2000. [Cited: 29 04 2012.] http://www.ttivanguard.com/dublin/dspdealth.pdf.

8. **Chou, Chi-Jui, Mohanakrishnan, Satish and Evans, Joseph B.** *FPGA Implementation of Digital Filters.* Lawrence, KS : University of Kansas, 1993.

9. **Jagatpal, Navarun, et al.** *Guitar Effects.* Computer Science, Columbia University. New York : Columbia University, 2007.

10. *A lecture course series: From concept engineering to implementation of signal processing algorithms with FPGAs.* **Huemer, Mario, Lunglmayr, Michael and Pfaff, Markus.** Antalya : Istanbul Technical University, 2005. 13th European Signal Processing Conference EUSIPCO.