ECE532: Vivaldo

Final Report

Joshua Calafato
Rohan Pavone
Ian Tramble

Table of Contents

1 Overview

1.1 Motivation

For novice musicians with an untrained ear, it is difficult to gauge performance on a particular piece or section of music. Without clear feedback on musical performance, practice can be inefficient and unnecessarily long. Mistakes might go unnoticed for long periods of time and incorrect techniques might be reinforced, making them more difficult to resolve in the future. Music teachers provide real-time feedback for students in order to improve their efficiencies when learning. It is, however, prohibitively expensive to have highly-skilled teachers present for all practice sessions.

A technology to provide automated real-time feedback on musical performance would resolve many of the issues described above. Incorrect techniques would be noticed and corrected immediately, allowing musicians to improve more quickly than through traditional practice methods. It would also do so cost-effectively given the number of hours that musicians typically spend practicing. With certain additions, this technology could also gamify the learning process, making practice more enjoyable and less arduous (for children learning to play music, for example).

1.2 Goals

Vivaldo is a self-contained product that requires only an internet connection and audio input from the user. It consists of a Nexys DDR 4 board and a server application. The major goals for this project were the following:

- Store a repository of songs on the cloud
- Allow users to download (or stream) songs from the cloud to their local device
- Prompt users to play the chosen song at their desired tempo
- Provide real-time feedback on their performance through an VGA display containing an image of the sheet music and a colored cursor tracking their progress across the sheet
- Store performance history on the server application to allow them to track progress and analyze trends

Due to time constraints and difficulty using the tools for this course, not all of these goals were achieved. The current iteration of Vivaldo contains:

- A server containing a centralized repository of songs (in .wav format)
- The ability to download a song from the server to a Nexys DDR 4 board
- An LED prompt indicating when to start playing the song and an LED metronome indicating the tempo at which to play the song (tempo fixed by .wav file on server)
- Real time LED feedback indicating correlation between song stored on server and actual audio inputted by user
- Plot of correlation values (vs. time) that is sent back to server and saved

1.3 System level block diagram

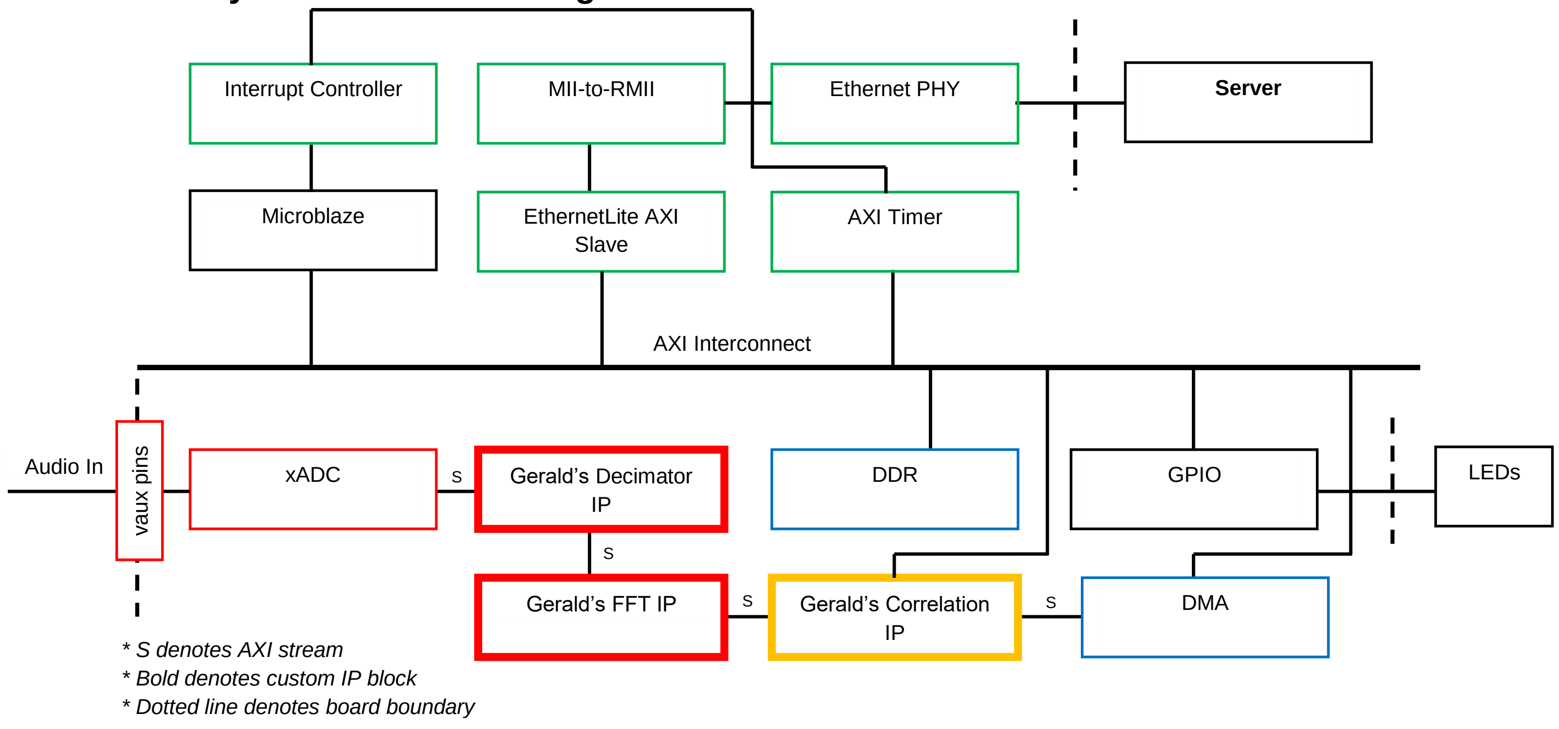

Figure 1: The block design of the top-level module of Vivaldo.

The Nexys DDR 4 board is primarily responsible for computing the correlation between the audio input and .wav files in real-time. In order to do this, the block design can be grouped into functional components:

- Audio input (red): Computes FFTs of analog audio input received by vaux pins. Sends these FFTs to Gerald's Correlation IP via an AXI stream interface.
- Song storage (green): Downloads requested song (song represented by sequence of 16-bit, 1024 sample FFTs) from central server to on-chip DDR memory.
- Golden FFT fetching (blue): DMA responsible for fetching FFTs from DDR memory and sending these FFTs to Gerald's Correlation IP via an AXI stream interface.
- Correlation (gold): Accepts FFTs from "audio input" and "golden FFT fetching" blocks. Buffers FFTs so that they are properly synchronized, computes correlation between FFTs, and writes correlation value to register that is then read by the Microblaze processor.

1.4 Brief description of IP

The major IPs used by Vivaldo are listed below. If IP is custom, it can be found in the main repository of the project. If it is Xilinx IP, documentation can be found in the links provided in Appendix A.

Table 1: Major IP blocks used in Vivaldo's design, with a description of their function, their origin, and the block in which they can be found.

IP Name*	Function	Origin	Block Parent
MicroBlaze	32-bit soft processor. Manages Ethernet, executes DMA transactions, reads correlation values stored in geralds_correlation_wrapper.	Xilinx	Main
AXI Direct Memory Access	M_AXI_MM2S reads data from DDR and sends it as an AXI stream to geralds_correlation_wrapper.	Xilinx	Main
AXI Interconnect	Manages bus.	Xilinx	Main
AXI Timer	Timer interrupts for Ethernet packets.	Xilinx	Main
Memory Interface Generator (MIG 7 series)	Creates AXI interface for DDR on Nexy DDR 4 board.	Xilinx	Main
AXI EthernetLite	Ethernet media access controller (MAC).	Xilinx	Main
Ethernet PHY MII to Reduce MII	Provides interface between RMII PHY and AXI EthernetLite core	Xilinx	Main
AXI Interrupt Controller	Mediates interrupts between MicroBlaze and AXI timer.	Xilinx	Main
geralds_decimator	Downsamples signal from XADC to user specified value (in our case, 44 kHz).	Custom	Main
geralds_fft	Calculates FFT magnitude. Accepts AXI stream from geralds_decimator.	Custom	Main
geralds_correlation_wrapper	Accepts AXI stream from geralds_fft and AXI DMA. Computes correlation internally and stores result in register.	Custom	Main
XADC Wizard	Reads analog value from temp sensor (for MIG management) and vaux pins (audio input). Equal duty cycle per channel. Samples outputted at 1MSPS.	Xilinx	Main
AXI GPIO	Constrained to Nexys DDR 4 LEDs.	Xilinx	Main
AXI Uartlite	For debugging.	Xilinx	Main
Divider Generator	Pipelined AXI divide required to compute cosine correlation between 2 vectors.	Xilinx	geralds_correlation_wrapper
CORDIC	AXI pipelined square root required to compute cosine correlation between 2 vectors.	Xilinx	geralds_correlation_wrapper
RAM-based Shift Register	Required in pipeline cosine	Xilinx	geralds_correlation_wrapper

	correlation computation. Valid signals from geralds_correlator need to be delayed while square roots, multiplies and divides are performed.		
geralds_correlator	Accepts AXI stream from geralds_fft and DMA. Buffers inputs to ensure synchronization and computes dot product between each vector and magnitude of each vector.	Xilinx	geralds_correlation_wrapper
Fast Fourier Transform	Accepts input from geralds_decimator. Computes 32 bit, 1024 sample FFT.	Xilinx	geralds_fft

* IP name as it appears in Vivado block diagram

1.5 Brief description of software components

The following table provides a description of the software components used for this project. Source code for these can be found on the GitHub repository.

Table 2: Software blocks used in the project. All can be found in the project's GitHub.

Component	Description	Runs on	Language	Origin
WAV to FFT conversion	Converts .wav files to 'x' bit, n sample FFTs to be sent to Nexys DDR 4 board. 'x' determined by the wav file's encoding.	Server	Python	Custom
WAV Generation	Generates .wav files for debugging (ex. 440Hz sine wave for an A).	Server	C	Open-source
Server	Listens for packet on specified port, parses requested song from chip, sends song (i.e. FFTs) in TCP packets of length N.	Server	Python	Custom
DMA	Contains utility functions for executing DMA transactions. AXI master port reads from DDR and sends to slave AXI stream port.	MicroBlaze	C	Custom
Streaming	Establishes connection with server, requests song name, accepts packets from server and stores in custom structs on the heap.	MicroBlaze	C	Custom
Lwip	Utility functions for TCP callbacks.	MicroBlaze	C	Xilinx

Correlation monitoring	Reads correlation from geralds_correlation_wrapper (AXI slave).	MicroBlaze	C	Custom

2 Outcome

As discussed in section 1.2, several of the user interface and feedback goals were scrapped due to hardware and time constraints, and unexpected delays which occurred during the project. The goals scrapped, however, did little to affect the base functionality of Vivaldo, and can be seen as stretch features that simply did not have time to be implemented.

2.1 Scrapped Features

At its core, Vivaldo was created to improve the efficiency of musicians when practicing by highlighting areas of a piece that were played poorly when compared to a golden standard. This goal was accomplished within the limits of this course, as a user can load any .wav file they desire from the server application onto the board and test how well they can play it. In terms of its effectiveness as a music learning tool, there are several features that we wished to implement but were unable to. The first, and most notable, is the ability for the user to dynamically select the tempo and section of the piece they would like to practice in order to assist with learning. The original method of accomplishing this was to store a database of MIDI files (a standard format in the music industry) which could be converted to .wav files and sent to Vivaldo. MIDI files are beneficial in that they can be edited to change tempo, sections, and individual track volumes without affecting pitch or other tracks. Further, MIDI files provide information on bars, which .wav files do not. The issue with MIDI files is that they are intended to be read by MIDI controllers, which may be hardware or software base, and are typically proprietary. These controllers are what give the MIDI files tonal characteristics and make them sound like instruments. Unfortunately, these controllers are also often expensive, and the cost of purchasing them for this project was not justified. As such, the team decided to stick with .wav files that could be found online or recorded with common desktop computers and sound cards that were already owned. Thus, we were unable to allow the user to specify sections of a song to practice, or the practice tempo. Another feature that we were unable to implement due to time constraints was video feedback to the user to display the sheet music of the piece being played, a cursor identifying where in the piece they currently were, and some form of feedback identifying how well they played each note or chord. This feature would have made it easier for users to note their place in a piece as well as to identify the exact areas where they played well and those they played poorly.

2.2 Additional Features

On top of the features described above, there are a few technical features of Vivaldo which could be implemented to improve it as a musical teaching tool. The first would be to have some way to identify fixed offsets in both pitch and timing of the user's input. Currently, playing a piece perfectly but starting half a beat off would result in Vivaldo reporting that the entire piece (or at least parts which are not similar for more than half a bar) was played incorrectly. Similarly, if an out of tune instrument was used to play a piece, Vivaldo would report very low scores of performance. Though this is technically correct (starting half a beat off or playing on an out of tune string is not the right way to play a piece of music), it does not effectively help the user figure out what to improve. A useful, though nontrivial, feature to implement would be more intelligent post processing of the received user input, so that the above two cases could be identified as a known mistake. For example, if the FFT frames extracted from the user's input were found to align with the golden FFTs with a delay of, say, 10 frames, then the correlation results could be calculated based on this fixed delay, and a warning could be given to the user saying they started late. Similarly, if there was a fixed frequency offset on every FFT calculated from the user relative to the golden values, a warning can be given to check instrument tuning.

2.3 Current Features and Methodology for Improvement

The current implementation of Vivaldo allows users to select a piece formatted in a .wav file and provides visual feedback via LEDs to identify how the user is playing real time. It also sends the correlation data back to the server, which is used to display a graph of the user's correlation to the golden standard with respect to time. A sample graph, which identifies areas of high correlation (accurate playing) and low correlation (poor playing) is shown in the figure below.

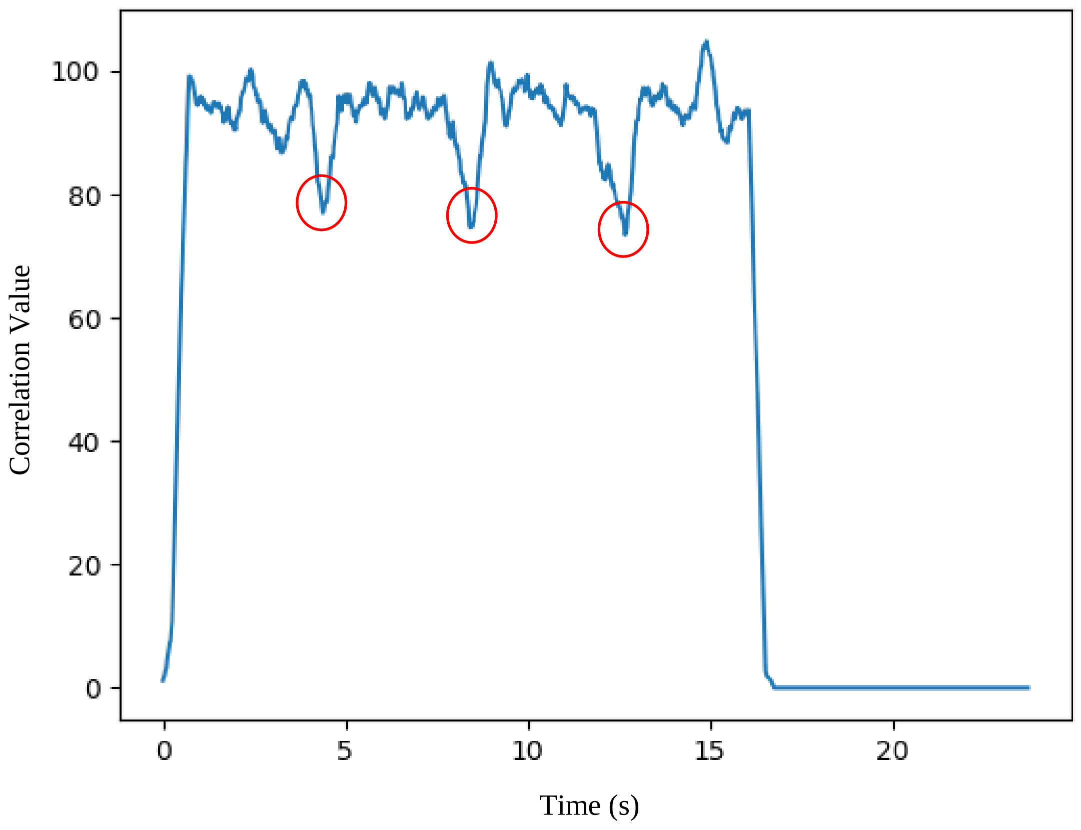

Figure 2: Sample correlation plot. Red circles identify note transitions, where the user was not exact with their tempo. After 16 seconds, the user stopped playing, showing low correlation for the remaining duration of the song.

Moving forward, the features in Section 2.1 should be completed first, as there is already infrastructure built in to assist with their completion. A VGA controller with a sweeping cursor was implemented, but went unused. The server program should be modified to transmit an image file representing the piece along with the golden FFT, and an additional block should be created which transfers the image file from memory, adds timing information for the cursor, and sends it to the input of the VGA block to be transmitted to a screen. Similarly, MIDI processing scripts have been developed, and simply need options for tempo setting or track selection to be inputted by the user. On top of this, MIDI controllers need to be purchased in order to convert MIDI files to .wav files, and a script which performs this conversion should be created.

The features in section 2.2 are more difficult to implement, as they have not been examined in this course. It is expected that these features would be implemented in software, as they require searching through significantly sized datasets and FFT-matching algorithm development. Perhaps hardware implementations of the software can be created after development is done, but the benefits of implementing this post processing in hardware seem minor.

2.4 Lessons Learned

Upon completion of this project, there has been time for reflection on how we implemented our functionality. Luckily, we do not think that any major changes to our flows would have been made if we could start over. Instead, the approach we took to implementing this project would most likely be modified. The largest time sinks came from members of the team trying to integrate or modify blocks which other members of the team created. Issues with misunderstandings and miscommunications meant that blocks operated subtly differently than expected, and made moving forward difficult at times. As such, it would have made more sense to thoroughly discuss the operation of each block as a group before and after it was implemented individually, so any difficulty, nuance, or problem that had to be circumvented was known to all members, and no surprises occurred when integrating parts.

3 Project Schedule

The following chart contains the original milestones and their planned completion alongside the actual weekly accomplishments according to our milestone reports. The original project sub-milestone tasks are included, even if they were not started or completed. Additional tasks that were completed through the duration of the project were added as unscheduled items (where plan start is set to 0) since they were unplanned for, yet necessary for the project's success.

Note that the purple blocks mark expected completion time, shaded regions indicate expected start date, and orange blocks indicate when the task was actually completed (with a completion percentage indicated for each task).

Table 3: Gantt chart of project tasks with planned and actual start and completion times. The periods are marking each week in the semester.

ACTIVITY	PLAN START	PLAN DURATION	ACTUAL START	ACTUAL DURATION	COMPLETION	PERIODS						
						1	2	3	4	5	6	7
MIDI converter	1	1	1	1	100%	🟪	▨		▨		▨	
Web Server	1	1	1	5	100%	🟪	🟧	🟧	🟧	🟧	▨	
Stream results to FPGA	1	1	1	5	100%	🟪	🟧	🟧	🟧	🟧	▨	

Audio Sensor	2	1	2	1	100%		🟪					
Integrate xADC IP block	2	1	2	4	100%		🟪	🟧	🟧	🟧		
Add Audio output (DAC)	2	1	2	1	100%		🟪					
FFT block with xADC	2	1	2	5	100%		🟪	🟧	🟧	🟧	🟧	
Custom IP Algorithm	3	1	3	1	100%			🟪				
DMA	0	0	3	2	100%			🟧	🟧			
Custom IP Verilog	3	1	3	5	100%			🟪	🟧	🟧	🟧	🟧
Custom IP testbenches	3	1	3	5	100%			🟪	🟧	🟧	🟧	🟧
Custom IP in hardware test	3	1	4	4	100%			▨	🟧	🟧	🟧	🟧
Integration of M1/M2/M3	3	2	4	4	100%			▨	🟪	🟧	🟧	🟧
Feedback LEDs	0	0	5	2	100%					🟧	🟧	
Decimation IP	0	0	5	1	100%					🟧		
Final Testing	7	1	6	2	100%						🟧	🟪
Verification	7	1	6	2	100%						🟧	🟪
Performance Measurement	7	1	6	2	100%						🟧	🟪
Feedback from Server	0	0	7	1	100%							🟧
VGA block IP Integration	5	1	5	2	33%				▨	🟪		
Add cursor to VGA	6	1	5	2	33%				▨	🟧	▨	
MIDI-to-sheet music	5	1	0	7	0%					▨		
Sheet music endpoint	5	1	0	1	0%					▨		
Display Sheet music	6	1	0	3	0%						▨	
Color cursor	6	1	0	0	0%						▨	

Note that none of the VGA milestones were completed due to the delays in other tasks. This is because the team found errors coming up during integration that needed to be resolved to move forward, and adding VGA to the project would have provided minimal returns to what the project was meant to accomplish. In its stead, LEDs were used to provide real-time feedback, and a new server endpoint was added that allowed the FPGA to send correlation data back to the server to be plotted over time there.

This still accomplished the task of providing feedback to the user without incurring much more work on the team.

The original milestones did not include some details that we found were necessary as the team dove into the project. For instance, the DMA milestone was never incorporated in the "golden" FFT stream of the correlation IP because moving the frames to and from memory was something we reasoned could be handled by the MicroBlaze (or a detail that we did not consider, since we assumed that AXIS protocols would have some simple block that would allow us to do this easily). Additionally, we had forgotten about the need for the sampling rates to match between the two FFT streams, and did not realize how cumbersome it was to use the xADC to modify the sampling rate to what was desired (especially when it had to multiplex between the temperature sensor and the audio in) – thus no time was budgeted for a decimator IP to provide the conversion for us.

Another glaring difference between the two schedules is that the original schedule assumed that all pieces in the project would be functional within the week they are deemed completed, thus leaving much more time for stretch goals (like VGA). However, a mixture between non-comprehensive test suites that allowed many bugs to fly under the radar, and unforeseen issues lead to long project delays. Some unforeseen issues included:

- MIG7/xADC issue – where the MIG7 requires a temperature sensor reading from the xADC and instantiates its own instance
- Integration project restart – the integrated project would not function as small parts were being added week-by-week, and had to be restarted at one point will all working components
- Version Control – did not realize the size of the projects and needed a more intelligent way of restructuring the project, leading to lost hours and delays
- Timing requirements and overflow issues – failure to meet timing requirements on complicated operations lead to some delays when trying to test in hardware. Additionally, overflow issues caused several bugs and needed to be addressed by non-trivially increasing the size of operations (which some IP blocks would not allow, requiring us to run bit-accurate simulations to examine which bits were significant).

Failing to write comprehensive tests or read documentation thoroughly lead to the following issues and delays:

- FFT modes – using real-time mode caused the block to ignore whether or not a sample was valid after the first valid signal of a packet, which would not have worked for an xADC being downsampled to 44.1KHz (with some incoming signals being invalid due to the MIG issue)
- Synchronization between streams – the processor reset on the correlation IP and the board reset on the FFT caused partial FFT frames to be fed into one pipe of the correlation IP, leading to frame synchronization issues and seemingly random correlation values.

Another key difference between the planned schedule and the executed one was that tasks were divided among team members distinctly in the original schedule. However, as the project unfolded, we found that working together on each task was more effective in producing working products.

4 Description of Blocks

As described in Section 1.4, various custom IP blocks and Xilinx designed IP blocks were used in this design. The following sections provide a detailed description of their block origins, how they were modified, or a description of custom blocks.

4.1 Custom IP blocks

The following subsection contains descriptions of custom IP blocks that were designed for this project: the Decimator IP, the Correlation IP, and a PWM debugging module. IP wrappers are included in section 4.2.

4.1.1 Decimator IP

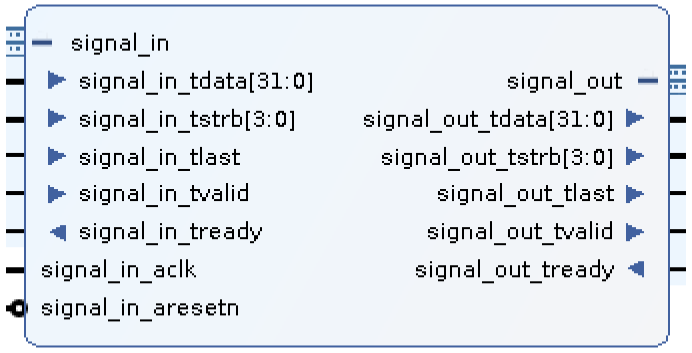

Figure 3: Decimator IP interfaces. signal_in and signal_out are AXIS interfaces (slave and master, respectively).

The decimator's goal is to output a decimated digital signal from an incoming digitally streamed input. This input is assumed to be oversampled and is downsampled based on a counter parameter given to the block an internal counter increments every clock cycle until the counter reaches a limit given to the block. When this is reached, a single valid sample (polled while the counter was incrementing) from the stream is exported – this effectively outputs a single sample at a reduced sampling rate.

4.1.2 Correlation IP

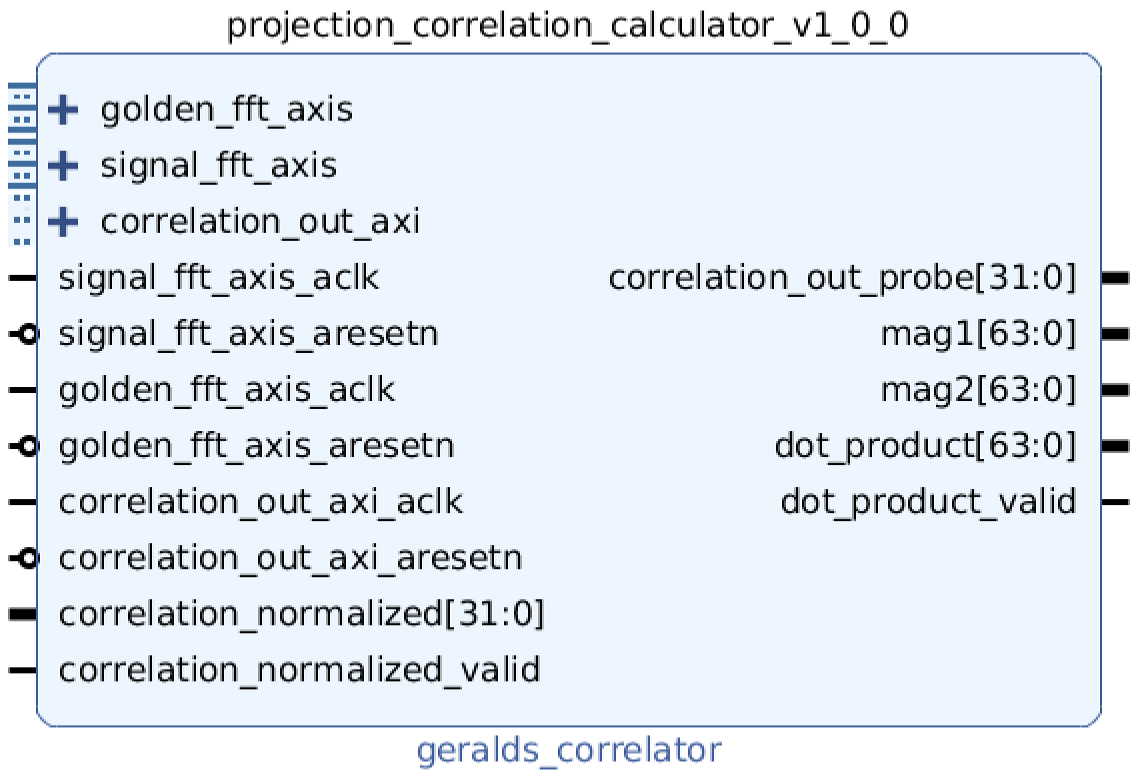

Figure 4: The correlation block contains several major interfaces: golden_fft_axis, signal_fft_axis, and correlation_out_axi. These standard interfaces receive two FFTs as AXIS signals for comparison. The correlation values can be read by a MicroBlaze via the correlation_out_axi port (AXI4-lite). The other interfaces help simplify the dot product calculation.

The correlation block receives two AXIS streams of FFTs (one from the Decimator IP and one from a DMA MM2S port) and calculates three important values related to the dot-product correlation:

1. Magnitude squared of signal 1
2. Magnitude squared of signal 2
3. Dot-product of signals 1 and 2, assuming synchronization between the frames

These are output via the `mag1`, `mag2` and `dot_product` ports, respectively, which is then fed to the correlation wrapper's remaining blocks (see section 4.2.1) to calculate the cosine of the angle between the two vectors (this is done to meet timing requirements). When this value is calculated, it is received via the `correlation_normalized` port, which then allows the block to store the result in the AXI4 port's registers (accessed via `correlation_out_axi`).

In order to ensure synchronization, frames are synchronized based on the `*_tlast` signal on the input of the FFT signal – if a last signal is asserted while the dot-product is accumulated only part-way, the products accumulated so far are discarded, and a new frame is collected from the FFT channel.

4.1.3 PWM

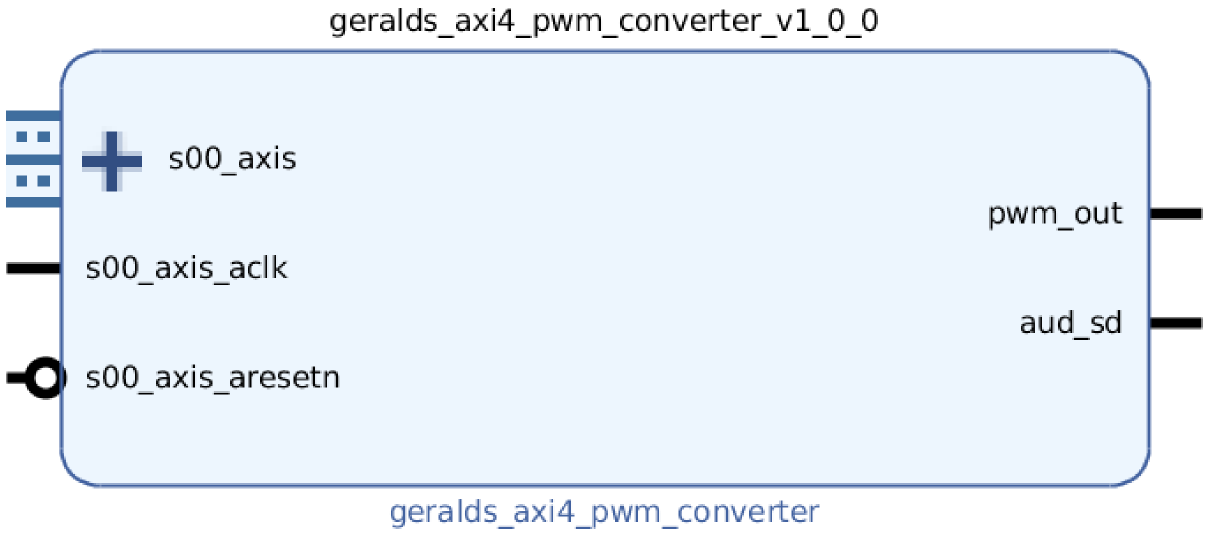

Figure 5: PWM controller block to output audio for debugging purposes. It receives a decimated signal in and outputs PWM control signals.

The PWM controller, while not officially necessary to Vivaldo, was used for debugging purposes. This block allows users to debug their audio signal-in by outputting a PWM signal to the board-mounted audio jack.

Upon receiving a valid signal from the input stream, it is stored in a register which is compared against a counter. The counter increments every clock cycle, and wraps around after the maximum possible input is reached (2^{12} for 12-bit signals in). While the counter is below the currently registered value, `pwm_out` outputs a high signal, and when it is above the registered value, it outputs a low signal. This produces an average voltage that is approximately proportional to the input voltage, emulating a DAC.

4.2 IP Wrappers

The following subsection contains descriptions of custom wrappers packaged as IP for partitioning of the project and abstracting complexities away from the higher-level block design. This applies to the FFT wrapper and the correlation wrapper.

4.2.1 Correlation Wrapper

The correlation wrapper takes the dot product and magnitudes of the 2 FFTs outputted from the custom correlation block (see section 4.1.2) and uses them to compute the cosine correlation between the 2 FFTs. Cosine correlation is given by the following formula:

$$\cos(\theta) = \frac{x \cdot y}{\|x\| \, \|y\|}$$

This value measures the linear dependence between the vectors x and y. Normally cos(θ) takes on values between 0 and 1. However, since we are using an integer divide block, we bitshift the numerator (i.e. the dot product) by 8 bits before performing the division. As a result, Vivaldo's correlation takes on values between 0 and 255. 255 is the maximum possible correlation value and it occurs when one FFT is a linearly scaled version of the other. The following IP block computes the correlation value:

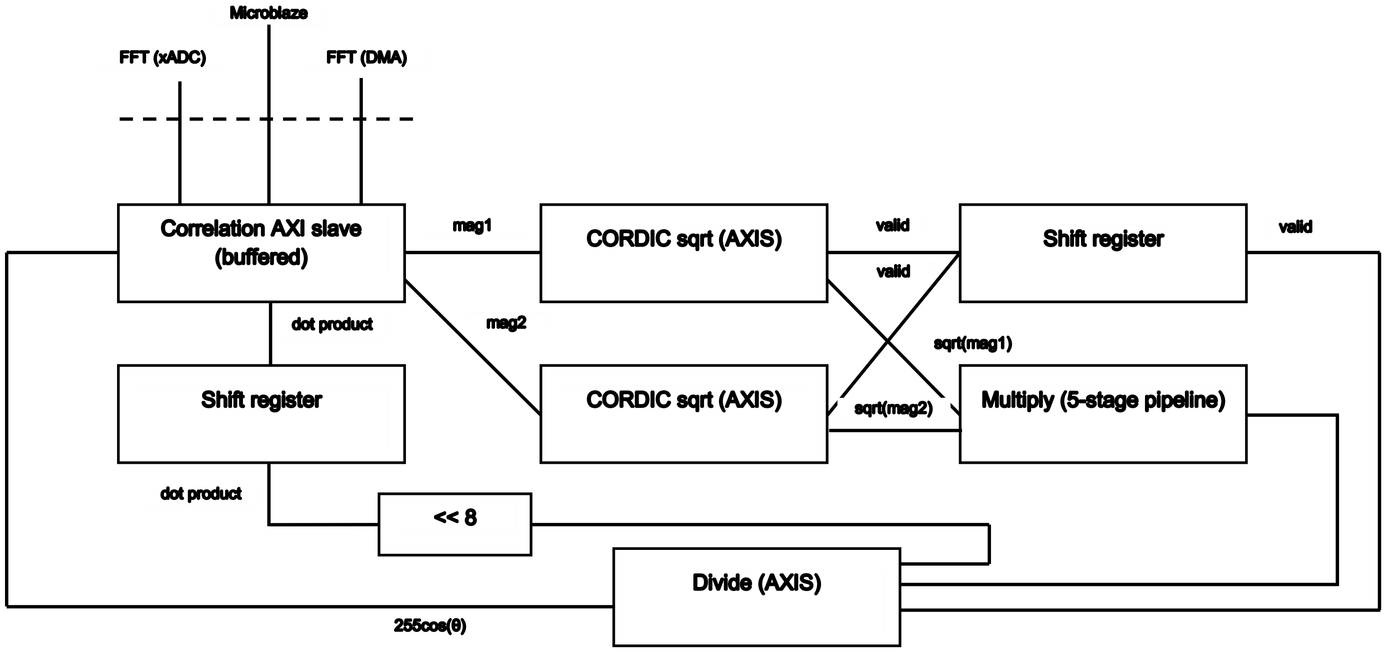

Figure 6: The correlation wrapper is used to produce the cosine dot product to meet timing requirements. The three outputs of the correlation IP (magnitude squared of the two input signals, and the dot product) are manipulated with Xilinx IP to produce the cosine dot product in a pipelined fashion.

The Xilinx IP blocks used are:

- Divider Generators (v5.1) – fitting the size of the outputs of the correlation IPs output
- Multiplier block (v12.0) – set to optimal pipeline depth (5 stages)
- RAM-based shift register (v12.0) – for pipelining work
- CORDIC blocks (v6.0) – set to perform square-root calculations with AXIS ports

4.2.2 FFT Wrapper

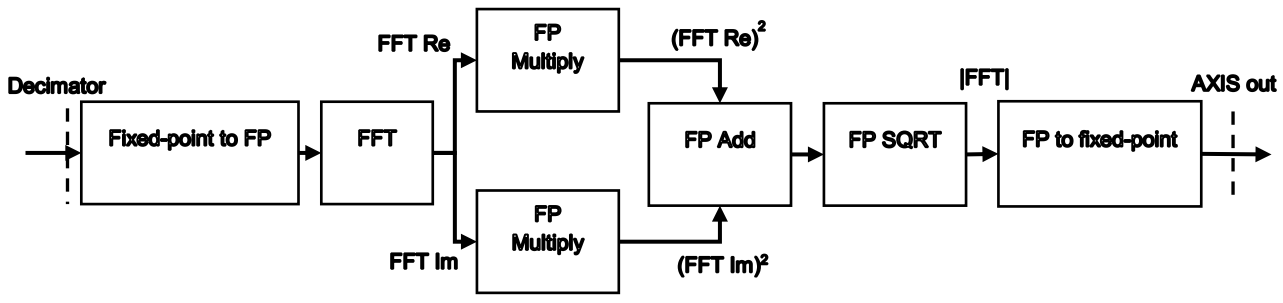

Figure 7: FFT wrapper block diagram. Decimated samples are input, converted into floating-point values, and fed into an FFT. The FFT's output (complex numbers) is converted to its magnitude (in floating-point), which is then fed back to the rest of the design as an integer. All arrows indicate AXIS connections.

The FFT wrapper wraps the Xilinx Fast Fourier Transform (v9.0) block to allow it to interface with its inputs and outputs. The input is assumed to be a 12-bit number (any signal from the xADC will suffice), which is then converted from an integer to a floating-point number.

The FFT is set to run in:

- Non real-time mode (so that it respects the valid signal of the incoming stream)
- 2048 samples per FFT
- Floating-point mode

The output is the real and imaginary parts of the complex number, but in floating-point form. These are converted into the magnitude of the complex number using floating-point blocks (v7.1). These blocks are set to:

- Multiply
- Add
- SQRT
- FP to fixed-point conversion
- Fixed-point to FP conversion

These are all also designed to forward control signals of the AXIS packet from the FFT - `_tlast` and `_tvalid`. This are important for the correlation IP's synchronization logic. Additionally, the FFT block was set to always be in "forward" FFT mode via a constant block.

4.3 Top-level design blocks

Figure 1 gives a full view of the top-level module of Vivaldo. As described earlier, a song is downloaded into memory as FFTs. When the entire song is downloaded, the MicroBlaze begins the correlation procedure: it incorporates two streams of FFTs into a correlation block (one via the xADC to Decimator to FFT Wrapper to Correlation Wrapper, and another via a DMA's MM2S port to Correlation Wrapper), whose correlation is then read in real-time via a MicroBlaze (and output to LEDs). After a song stream is completed, the results of the correlation are transmitted via Ethernet to a webserver which displays the overall result. The previous sections (Sections 1 and 2) describe its functionality.

Aside from the custom IP blocks described in the previous subsections, the following blocks are used in the design, with their versions and description on how they are set:

Table 4: Xilinx IPs used in design of this project with their versions and the settings used for each block. Some are set according to tutorials, which are listed.

IP Name*	Version	Settings
MicroBlaze	10.0	Instantiate with interrupt controller, and run block automation
AXI Direct Memory Access	7.1	Instantiate without Scatter-Gather mode. Needs MM2S to function, but useful to have S2MM for debugging.
AXI Interconnect	(Auto-generated via MicroBlaze block automation)	
AXI Timer	2.0	See ethernet tutorial.
Memory Interface Generator (MIG 7 series)	4.0	After block automation, must have xADC instantiation disabled. The temperature port must be connected to the temperature out port of the xADC. See ethernet tutorial.
AXI EthernetLite	3.0	See ethernet tutorial.
Ethernet PHY MII to Reduce MII	2.0	See ethernet tutorial.
AXI Interrupt Controller	(Auto-generated via MicroBlaze block automation)	
xADC Wizard	3.3	Has channel sequencer, continuous sequencer mode, and temperature bus enabled. Averaging is disabled, and the temperature and vauxp3/vauxn3 pins are enabled. In order to get audio data, digital logic should be added to filter based on the channel of the data out. Filtering for channel 19 will select the audio signal.
AXI GPIO	2.0	Connects to LEDs
AXI Uartlite	2.0	Should have tx and rx pins exported externally.
Clock Wizard	5.4	Has three clocks (100MHz, 200MHz, and 50MHz), a single-ended input clock that connects to sys_clk, and is set to Active Low Reset.

The ethernet tutorial can be found here: https://reference.digilentinc.com/learn/programmable-logic/tutorials/nexys-4-ddr-getting-started-with-microblaze-servers/start. This assumes you are using a Nexys 4 DDR board.

4.4 Software blocks

All software blocks can be found in the project GitHub. The components were written based on a combination of modified tutorials read online and personal experience with writing drivers.

5 Description of Design Tree

The complete design tree can be found on github at https://github.com/tramblei/vivaldo (as well as a short video demonstrating functionality). The main project is vivaldo_fresh_eth_xadc_fft_dma_corr.xpr. The custom IP (discussed above) is located inside the ip_repo/ directory. The IP projects can be edited through the projects located in the ip_repo/ip_projects directory. The server code is tracked as a git submodule (https://github.com/tramblei/vivaldo-server).

To run Vivaldo:

- Launch the Xilinx SDK from within the Vivaldo project in Vivado
- Program the FPGA with the most recently generated bitstream
- Connect the board to a server containing the vivaldo-server code with an Ethernet cable (be sure to disable firewalls that might interfere with transfer)
- Run echo_server.py (vivaldo-server/echo_server.py) on the server
- Modify the requested song name in https://github.com/tramblei/vivaldo/blob/master/vivaldo_fresh_eth_xadc_fft_dma_corr.sdk/eth_with_dma_fft_xadc/src/main.c (ensuring that it is also located in the vivaldo-server/sample_wav_files directory)
- Run the eth_plain test from the Xilinx SDK
- Wait for the LED prompt before starting to play the requested song (song start denoted by LED flashing 3 times at correct tempo)

6 Tips and Tricks

Implementing this project was equal parts frustrating and rewarding. Most complications resulted from the designers (i.e. us) misunderstanding how certain Xilinx blocks work. As such, we found that it was paramount to read the documentation of any black box you decide to use in your design. At the very least, the relevant sections of the documentation should be thoroughly understood (including operation latencies, expectations on inputs, resolution restrictions, etc.) before testing. It was also found that project management was vital for keeping such a large and complicated project manageable and modifiable. One decision the team made that was useful for future development was to separate individual IP components into a packaged directory and a project directory. This way, each IP block could be accessed and modified in its own project, and the data necessary to place these IP blocks into other projects were held elsewhere. As such, any development on the IP project only affected the packaged IP after it was repackaged, and we could be sure that changes to the IP packages were made by cleaning the package directory and repackaging the projects. Other tips and tricks come from dealing with the design tools. We found that the SDK window had to be closed before recreating a bit stream (else new BSPs were made), and that often times errors in the SDK could be resolved by closing it and opening it again. Similar things happened when dealing with multiple Vivado projects at the same time. We also found it necessary to clean packaged IP directories when making changes to internal block diagrams of IP projects and before repackaging. Finally, a general understanding of clock cycles needed for data to propagate through blocks is necessary when developing new IP, and it is useful to have test modules ready to be thrown into a block diagram. As an example, we had LED, HEX, and PWM audio drivers that could be thrown into any block diagram in order to verify the operation of our changes.

Appendix A – Links to Documentation

The following links are to documentation pertaining to IP blocks used during this project. Links that were particularly useful and marked in bold and are italicized.

IP Name	Documentation
MicroBlaze	https://www.xilinx.com/products/design-tools/microblaze.html
AXI Direct Memory Access	***https://www.xilinx.com/support/documentation/ip_documentation/axi_dma/v7_1/pg021_axi_dma.pdf***
AXI Interconnect	https://www.xilinx.com/support/documentation/ip_documentation/axi_interconnect/v2_1/pg059-axi-interconnect.pdf
AXI Timer	https://www.xilinx.com/support/documentation/ip_documentation/axi_timer/v2_0/pg079-axi-timer.pdf
Memory Interface Generator (MIG 7 series)	https://www.xilinx.com/support/documentation/ip_documentation/mig_7series/v1_4/ug586_7Series_MIS.pdf
AXI EthernetLite	https://www.xilinx.com/support/documentation/ip_documentation/axi_ethernetlite/v3_0/pg135-axi-ethernetlite.pdf ***https://reference.digilentinc.com/learn/programmable-logic/tutorials/nexys-4-ddr-getting-started-with-microblaze-servers/start***
Ethernet PHY MII to Reduce MII	https://www.xilinx.com/support/documentation/ip_documentation/mii_to_rmii/v2_0/pg146-mii-to-rmii.pdf
AXI Interrupt Controller	https://www.xilinx.com/support/documentation/ip_documentation/axi_intc/v4_1/pg099-axi-intc.pdf
XADC Wizard	***https://www.xilinx.com/support/documentation/ip_documentation/xadc_wiz/v3_0/pg091-xadc-wiz.pdf***
AXI GPIO	https://www.xilinx.com/support/documentation/ip_documentation/axi_gpio/v2_0/pg144-axi-gpio.pdf
AXI Uartlite	https://www.xilinx.com/support/documentation/ip_documentation/axi_uartlite/v2_0/pg142-axi-uartlite.pdf
Divider Generator	https://www.xilinx.com/support/documentation/ip_documentation/div_gen/v5_1/pg151-div-gen.pdf
CORDIC	https://www.xilinx.com/support/documentation/ip_documentation/cordic/v6_0/pg105-cordic.pdf
RAM-based Shift Register	https://www.xilinx.com/support/documentation/ip_documentation/shift_ram/v12_0/pg122-c-shift-ram.pdf
Fast Fourier Transform	***https://www.xilinx.com/support/documentation/ip_documentation/xfft/v9_0/pg109-xfft.pdf***