



Final Year Research Project 2
Semester 2, 2018

Project Title: **SILO: The Hardware Granular Synthesizer**

Group No.: **29**

Assessment: **Research Report**

Supervisor: **Jagdish Patra**

Date of Submission: **29 October 2018**

Student Name	ID	Unit Code	Course
Timothy Opie	101046047	EEE40012	BH-EEE

EEE40012: Final Year Research Project 2

SILO

The Hardware Granular Synthesizer

Semester 2 Research Report

October 29, 2018

Author: Timothy Opie

Student Number: 101046047

Supervisor: Jagdish Patra

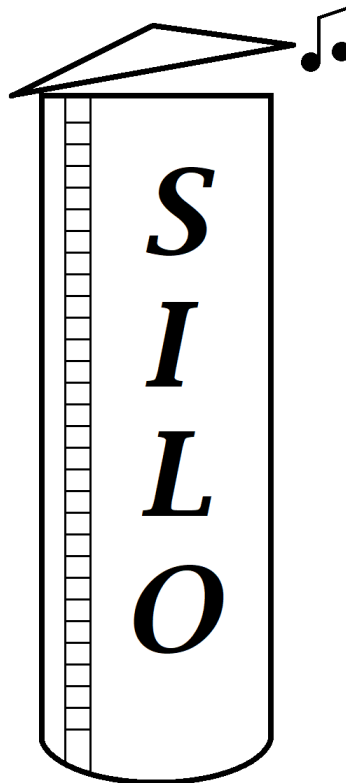


TABLE OF CONTENTS

1. Abstract	5
1.1 Problem Statement	5
1.2 Research Question	6
2. Introduction and Literature Review	7
2.1 Granular Synthesis	7
2.2 Software Granular Synthesis	8
2.3 Hardware Granular Synthesis	10
2.4 Hardware Options	12
2.5 Algorithms	13
2.6 Conclusion of the Literary Review	13
3. Methodology	14
3.1 Requirements with Checklist	14
3.2 Functionality with Checklist	15
3.3 Testing and Quality Assurance Plan	16
3.4 Hardware for Project	16
3.5 Budget	18
3.6 Schedule	19
4. Implementation of Design	20
4.1 Overall Design	20
4.2 Terminology	22
4.3 Detailed Design	24
4.4 Storing on QuadSPI	37
5. Results	38
6. Further Discussion	42
6.1 Timing Issues	42
6.2 Analog to Digital Conversion Issues:	43
6.3 Discussion of Results	43
6.4 Future Prospects	44
7. Conclusion	45

8. Source Code and Reports	46
silo.vhd	46
pitchshift.vhd	53
clock_generator.vhd	55
envelope.vhd	57
density.vhd	60
rnd.vhd	62
constraints.xdc	64
Xilinx Vivado Utilisation Report	66
Xilinx Vivado Synthesis Report	70
9. References	83

1. Abstract

This report outlines the completion of the project: SILO The Hardware Granular Synthesizer. Within this report, I will examine the thesis proposed and discuss the current literature. I will then discuss the methodology. Next, I will discuss the design of this project, examining how the design was implemented. I will then present the verification of the implementation and discuss the outcome. Finally, I will discuss future plans and conclude.

Please note that the problem and the literature review have not changed significantly since last semester, although I feel they are much clearer now, and I felt it was important to include them in my report.

1.1 Problem Statement

Element	Description
New sounds, new interfaces	As with many disciplines, the artist is always seeking creative inspiration, something that will spark a new creative idea. New sounds and new ways to control sound are two ways of accomplishing this.
Synthesis difficulty	This synthesis method has always been difficult to access, because the strain it puts on computer systems meant that for most users it was not even an option until the 2000s, despite being a concept from the 1940s.
Lack of hardware options	Currently there is not a hardware granular synthesizer available on the market. All implementations are software based. Two hardware modules were available for a very limited time, but were both discontinued due to design problems.
Reliability in Live performance	Software based synthesis is acceptable for composition purposes and in studio sound design, but in a live performance setting, stability and reliability are significantly more important.
Benefits of a solution	A complete hardware solution would mitigate most of the problems inherent in software solutions.
Hardware capability	In software the solution is already extremely good, so why then is it not available in hardware, when all other synthesis methods have many hardware implementations? Is it too hard to solve with hardware, or is it just that very few people have both the skills and musical knowledge to implement it?

1.2 Research Question

Can a hardware granular synthesizer be built that matches the functionality of a software implementation?

2. Introduction and Literature Review

This chapter covers the literature and research that already exists in this field. This was already covered in the proposal, however some new additions have been made that are quite pertinent to the outcome of the project.

2.1 Granular Synthesis

Granular synthesis is a sound synthesis technique used in musical performance and musical composition by which sonic textures are created by distributing thousands of tiny “grains” of sound over time. These sonic grain can either be created or derived from another sound source. Each grain is windowed into a 21ms slice of audio that is encapsulated by an amplitude varying envelope that ensures each grain starts and ends at 0.0 amplitude. Each grain is created separately with individual properties and then layered across time with thousands of other sonic grains to create the sonic textures. The textures are influenced by the density and synchronicity of grains, the spread of the grains, and the pitch, amplitude, and length of each individual grain. On their own, they are barely audible, but as a large group, they afford atomic building blocks for creating new sounds, and new experiences with audio.

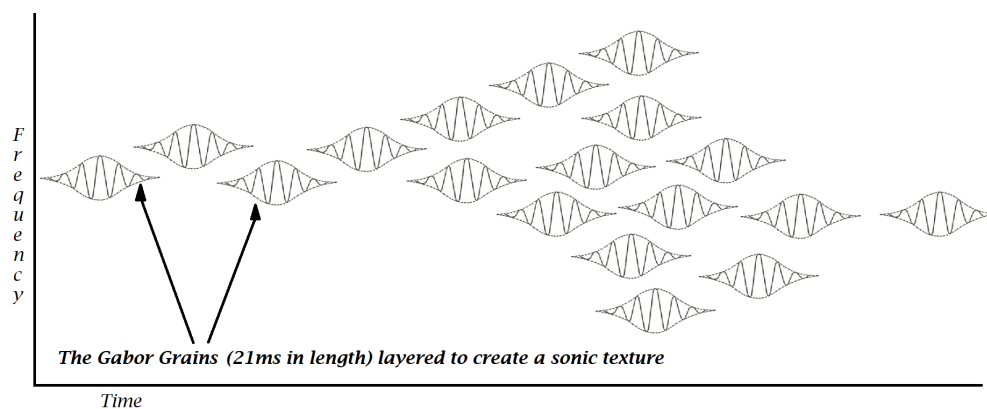


Figure 1: Diagram of how granular synthesis creates sonic textures

Granular synthesis was derived from Dennis Gabor’s theory of communication in 1947 [1] where he proposed reducing communication signals to a series elementary acoustical quanta. Each quantum a windowed sonic event no longer than 21 milliseconds he referred to as a logon. The logons were to be transmitted and later reassembled, in an effort to reduce telephone bandwidth. Unfortunately, Gabor failed to create a machine with enough fidelity to reproduce the original signal with any clarity, despite multiple attempts.

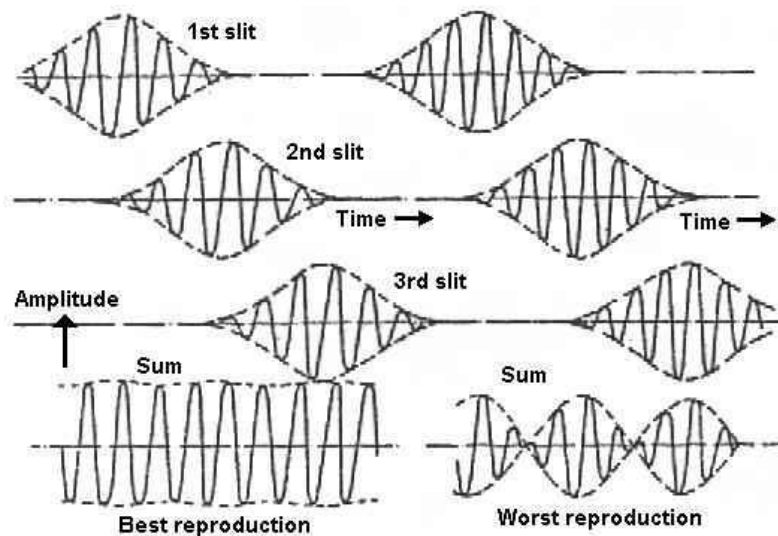


Figure 2: A diagram by Dennis Gabor of the theorised logons both separate, and summed[1].

In the late 1950s, the architect and composer Iannis Xenakis experimented with Gabor's theory with musical intention. Xenakis referred to the logons, with the term Gabor grain, which was later shortened to grain. Xenakis expanded on the theory, putting emphasis on the musical properties of the process. In 1958 he created a musical work entitled "Concret PH". In 1959 he created the musical work entitled "Analogique A et B". He created no more musical works using this method as they had been constructed using tape-splicing techniques and took many months to complete. He published an expansion and discussion on this theory in his 1971 book entitled "Formalized Music". [2]

2.2 Software Granular Synthesis

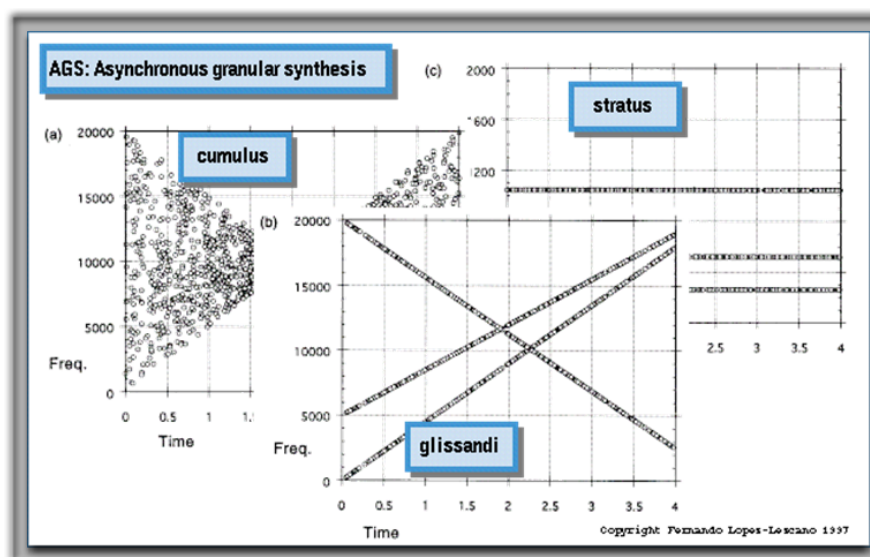
In 1972 Xenakis gave a guest lecture discussing the Gabor Grain, inspiring electronic composer and researcher Curtis Roads to implement this concept digitally. Using the program MUSIC V written by Max Matthews, running on a Burroughs B6700 mainframe computer at his university, he designed a composition technique in which each punch card represented a single audio grain. He created large stacks of cards which the computer would render into audio each weekend, initially conducting small sonic experiments, and eventually creating the musical work NSCOR which Roads slowly modified and perfected over a 10 year period. [3]

In the late 1970s, Dr. Barry Truax stated this method was far too cumbersome and set about creating a real-time granular synthesis process. He created a system called PodX which utilised a DMX-1000 Digital Signal Processor controlled by a DEC LSI-11/23 mini-computer [4]. This system still is still operable by appointment. It involves using the keyboard as a

controller for adjusting the output of the DMX-1000. Hotkeys have been assigned that control each element of granular synthesis. It is not portable and not used in live performance as it can only handle small amounts of data at a time. Truax overcomes this by recording and layering multiple channels in his studio to achieve the rich sound associated with his compositions. [23]

The CSound language developed by Barry Vercoe in 1985 enabled microcomputers to create rudimentary grains using a method similar to Curtis Roads. In 1994 Paris Smaragdís and John ffitch added the first complete grain OPCODE to CSound. It gave composers much more control and fidelity, however required them to set 23 parameters per grain. This is the version I first used in 1999. CSound is still an active project, and was included in the one laptop per child project.

In 1995 Curtis Roads created a free granular synthesis program for the Macintosh Classic OS called Cloud Generator that allowed the user to record short sound samples, granulate them, and disperse the grains using a number of different patterns. This was a non-real-time implementation that relied on visualisation to create sound. [24]



*Figure 3: Screenshots from Cloud Generator.
It displays each grain as a circle, on a frequency vs time graph [24].*

In the early 2000s as home computers became powerful enough to handle this technique, audio plugins and software implementations appeared, although in real-time mode they were unstable, and would crash regularly [10]. Now many software environments, and digital audio workstations, such as Ableton Live, can implement this synthesis technique much more reliably. It has however always been software based, which carries certain inherent risks in live performance.

2.3 Hardware Granular Synthesis

Two companies have attempted hardware implementations of granular synthesis, or at least granular synthesis adjacent devices, with poor results. Another company recently started a campaign on kickstarter, promoting a granular synthesis hardware device, however when I investigated it I found it was a Raspberry Pi hidden in a box, running software on Raspbian [31]. Having experimented with granular synthesis on a Raspberry Pi, I know this device will be quite limited, despite having a nice interface, plus it is not hardware based. The two hardware devices will be discussed now.

The Phonogene (retired) by the Make Noise Company. Designed as a digital tape manipulation tool for Musique Concrete composition¹. It can handle micromontage² which is similar, although much simpler to implement than granular synthesis, and sounds quite different [21]. The actual sonic results from this device are very noisy and of low fidelity. It can only handle a couple of seconds of audio, and it clips every sample it plays, resulting in a lot of ticking sounds, due to a lack of amplitude enveloping.



Figure 4: The physical interface for the Phonogene.

¹ Musique Concrete is a composition technique developed by Pierre Schaeffer in 1948 that uses only found sound (recorded sound) as the source material, and relies on audio tape manipulation techniques to create musical works.

² Micromontage involves splicing audio tape into 100ms to 1sec segments and rearranging the segments.

Clouds (retired) by Mutable Instruments. The name is a nod to Curtis Roads' early graphically controlled program. Only 250 were manufactured, as the designer, Olivier Gillet found it too problematic. The source code is online under the BSD license, written for an F4xx 32-bit ARM-based microcontroller. It only contains some functionality of granular synthesis [22]. Being ARM-based is most likely the reason it does not run well, as the device is much less powerful than most computers used for music production. I have not heard or seen this device in action as it is so rare, but having experimented with granular synthesis on low powered devices in the past I can deduce why Gillet found it too problematic.



Figure 5: The physical interface for the Clouds texture synthesizer.

2.4 Hardware Options

Although granular synthesis hardware designs are non-existent, other synthesis methods are plentiful in hardware. Numerous outlets sell FM synthesizers, subtractive and additive synthesizers, formant synthesizers, and digital samplers. It is not due to unpopularity with granular synthesis that appropriate devices do not exist, because it is popular. Famous artists such Daft Punk, Nine Inch Nails, Aphex Twin, Autechre, and Fat Boy Slim use granular synthesis in their music [10][26].

There are two paths that could provide a solution to hardware granular synthesis. The system could be built with a microcontroller (MCU) or a Field Programmable Gate Array (FPGA).

An MCU is much easier to work with, because it can be programmed in C, and is almost a purpose built computer. However The Mutable Instruments Clouds hardware interface was created on an MCU and they abandoned it because it could not perform very well. This is most likely because an MCU runs all operations sequentially, loading and running one line of code at a time, plus MCUs are low powered computers, designed to provide low power solutions and portability, but not provide grunt. The MCU option is basically a low powered dedicated software implementation.

An FPGA is significantly harder to program, but the throughput can be optimised to allow tremendous amounts of data to flow simultaneously. This is because an FPGA supports parallelism. A sequence that might require 10 sequential operations on an MCU, might be accomplished in a single clock cycle on an FPGA. A process like granular synthesis that requires 23 parameters to be applied to a tiny audio sample could potentially be optimised to perform all operation in just a few clock cycles [20]. Potentially you could generate very dense granular structures in real time, that are not possible on a sequential system.

Although there are no FPGA based granular synthesis projects, there are numerous DSP projects and other synthesizer implementations running on FPGA. Some recent articles on FPGA programming include FFT implementations [14] and wavetable synthesis [15]. Students from MIT created a basic synthesizer on an FPGA. It was square wave based, but included some rudimentary DSP effects to help shape the sound. One particularly insightful article on wavetable synthesis³ for FPGA discusses how to store waveforms in memory and play them back at different speeds to create a multiple pitched device [15].

There are also numerous books and manuals on FPGA programming that cover many facets related to behavioural design, tips, and examples for effective electronic design [17][18][19].

³ Wavetable synthesis was one of the earliest forms of digital synthesis, used before FM synthesis was invented. It stores wave data in an array that can be quickly called and applied.

2.5 Algorithms

Pitch Shifting is an important attribute of granular synthesis. Pitch shifting can be achieved by performing an FFT on the audio, manipulating the bins, and then performing an inverse FFT [32]. This process however requires collecting a large buffer of audio to analyse and then produce high quality audio, resulting in significant lag, which is problematic for musicians performing in real-time. There are other forms of pitch shifting using filters and offsets, however the application is for low fidelity analog implementations [27]. Another form of pitch shifting can be performed by manipulating the sample rate [33]. This generally involves adjusting out outgoing sample rate, although it could be accomplished by adjusting the incoming sample rate if you have good control of the analog to digital converter.

2.6 Conclusion of the Literary Review

Many decisions were made based on the literature, as will be discussed in the following chapters.

3. Methodology

This section will discuss the requirements, functionality, testing/quality assurance procedures, hardware that was used, and the schedule.

In summary, I required a proper hardware implementation, so it needed to be implemented on an FPGA using VHDL.

I have some experience working with VHDL, so despite being difficult, it is something I already understood to some degree. Ultimately I would like to create a granular synthesis integrated circuit design on an ASIC, so although that is well beyond the scope of this project, working with VHDL to create the FPGA prototype will get me most of the way there.

3.1 Requirements with Checklist

Requirement	Checklist
The system will be built using VHDL.	Accomplished
The system will be prototyped on a Xilinx Arty S7-50 FPGA.	Accomplished
The system will match the common functionality of software implementations.	Accomplished
A basic implementation will be ready for presentation by week 12.	Presented a system that could read and play back sound. Although that entire section was rewritten during semester 2 as the system improved
The remaining functionality will be implemented, tested, and presented by week 24.	A functional system was presented at the Capstone Expo
The system will be used to perform live music as the final test in week 24.	A performing system was presented at the Capstone Expo

Table 1: Requirements with a checklist of outcomes

3.2 Functionality with Checklist

In order to properly match the functionality of software based granular synthesis, I have defined the functionality that needs to be implement via hardware. Six functions are identified in the table below [12].

Functionality	Description	Checklist
Density [#]	The number of grains to be generated per second. This should be from 0 to at least 2000 grains per second.	2000 grains per second is about the default speed, and this can be easily reduced to 0. (Note that they are not overlapping - More than 4000 per second requires overlapping)
ASynchronicity	Determine whether the grains are produced synchronously, and if not, the degree of synchronicity allowed	Async enable and a multi level randomiser for degree of Asynchronicity has been built
Envelope Length [#]	The length of the audio segment, with the envelope applied. The envelope itself will be a gaussian curve. Range 15-40ms. Default will be 21ms	Variable length envelope created. It however uses an amplitude varying trapezoidal envelope, instead of Gaussian. This give more control over attack and release times of the grain.
Frequency [#]	Shift the grain frequency within the audio hearing range	Can shift frequency up or down an octave with a high degree of clarity.
Amplitude [#]	The peak amplitude of each grain, from 0.0 to 1.0	Complete amplitude control, created and utilised in envelope creation
Input Data control	Stream continuous audio or hold audio in one position - allows time stretching and elongation of sounds	Can stream or hold on a single grain.

[#]: These values can also have an offset value, which allows them to have a +/- random offset from the defined value.

Table 2: Functionality with a checklist of outcomes

3.3 Testing and Quality Assurance Plan

Pitch and timing are both critical in music and sound production. Not mission critical, but if the pitch is off, or if slow timing or lag can be perceived aurally, then it means the system is not effective or efficient enough. These were two key issues I tracked most vigilantly throughout the process.

For pitch testing I used sine wave input. Using sine waves allows me to easily hear discrepancies in the pitch. This is due to the fact that I have perfect pitch recognition skills. Using sine waves also allows me to measure the actual period of the waveform to ensure the pitch is what I expect it to be.

The correct value can be easily calculated with sine wave testing. For example a test sine wave of 432 Hz has a period $1/432$. If I shift the pitch up one full octave, the output frequency should be 864 Hz, with a period of $1/864$. This is exactly half the period of the input. If I shift the pitch down one octave the output pitch should be 216 Hz, with a period of $1/216$, which is exactly double that of the input. These values will be tested and verified in the results and verification chapter.

Testing the timing was done by monitoring the timing reports in Xilinx's Vivado software, as timing is also critical in VHDL designs. The timing procedures for the sampling and pitch shifting were modified and optimised numerous times during the implementation process in order to ensure that all data was ready by the next clock cycle, ensuring the system ran like clockwork. Making the code more modular also helped resolve some timing issues.

3.4 Hardware for Project

The granular synthesis design was implemented on a Xilinx Arty S7-50 FPGA

The specification of the Arty S7-50 [30]:

- Spartan 7 FPGA
- 256 MB DDR3L with a 16-bit bus @ 650 MHz
- Internal clock speeds exceeding 450MHz;
- DSP Slices: 120
- Clock Management Tiles: 5
- Flip-flops: 65,200
- Slices: 8,150
- Logic Cells: 52,160
- 1 MSPS On-chip ADC

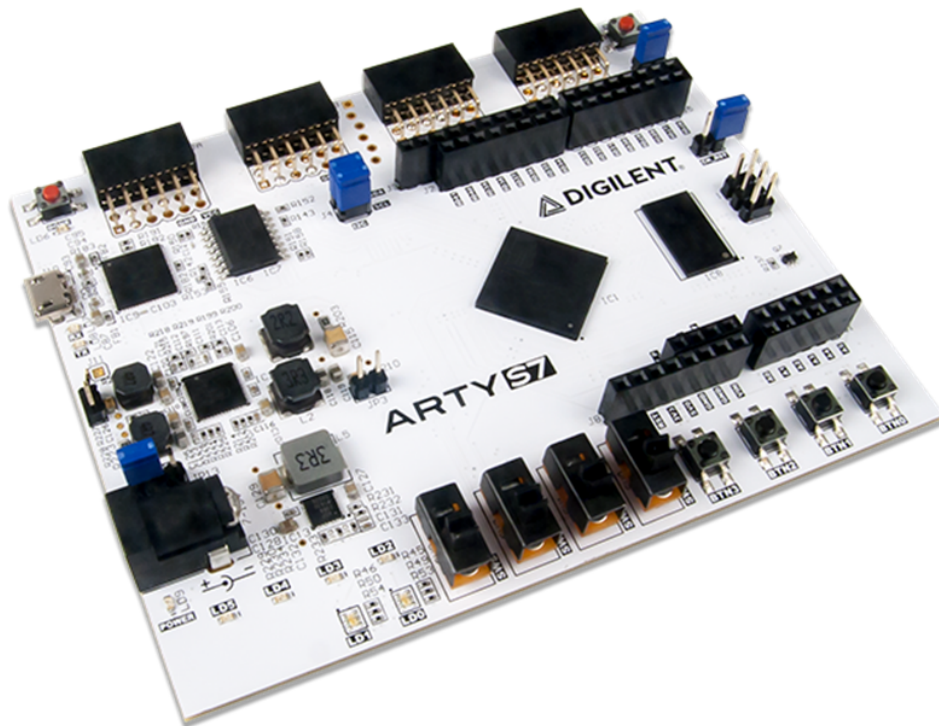


Figure 6: The Xilinx Artix S7-50 FPGA

In retrospect I wish I had purchased an older board. Because this board was only a couple of months old when I purchased it, there was very little documentation or example code available, other than the manufacturer's official documentation. This meant that there were many times where I felt I was fumbling around and having to experiment just to achieve some basic action that was well covered on many websites regarding older FPGA boards. Unfortunately I was drawn to this board because of the specification, instead of the resources. Despite this I eventually found my way though, and am now a lot more comfortable with this board, plus now it is almost 1 year old, so documentation and examples are starting to become more common.

Other Hardware required:

- Digital to Analogue converter - obtained - PMOD-AMP3
 - This took some time to adapt for the Artix S7, but has been an extremely useful module!
- Accelerometer - obtained - PMOD-ACL2
 - This was obtained but did not function. I spent days trying to get their inhouse demo running, but in the end I concluded the hardware was faulty, as I could never get any input from the device. I will return the device.
- MIDI input port - **Ordered in May but never received!!** - PMOD-MIDI-IN
 - I was really looking forward to this module. It would have been perfect for performance. I have contacted the distributor multiple times, over the past 5

months, but they are just ignoring my communication. This was going to be used to control the input sliders. At the moment I am stuck using the Arty S7 onboard switches and buttons - which I have optimised for performance, but it is not ideal, and has narrowed how I can interact considerably. I will probably create my own input device in the future, now that I feel a lot more confident with this technology.

3.5 Budget

There was no strict budget for this project. I decided from the onset to propose and fund the project myself, as this is a personal hobby as much as a research project for me. I purchased the parts with the intention of being able to keep using them for the next decade.

With my own personal budget in mind though, I wanted to keep the project under \$250AUD. Which I have accomplished, although I will need to make a few more purchases in the future if I want to go beyond a prototype stage.

I spent \$218.98AUD at digikey.com.au - including postage and GST:

All prices are in AUD

#	Product Details	Quantity	Availability	Unit Price	Extended Price
1	1286-1172-ND 410-352 ARTY S7-50 SPARTAN-7	1	1 - Immediate	182.79000	AU\$182.79
2	1286-1134-ND 410-270 PMDAMP3 STEREO POWER AMPLIFIER	1	1 - Immediate	13.40000	AU\$13.40
3	1286-1072-ND 410-255 PMDACL2 3AXIS ADXL362	1	1 - Immediate	22.79000	AU\$22.79

Table 3: Digikey.com.au expenses

The PMOD-MIDI was \$30AUD

So I am currently down \$50, which I could use to purchase much better controllers, which would bring me back to the \$250 range

3.6 Schedule

I kept to my schedule quite tightly, and am happy with the progress I made throughout the semester. The following chart from last semester, has been updated it with my current progress.

Note: Blue = completed, Purple = to do

Weeks	3	4	5	6	7	8	9	10	11	12	B	13	14	15	16	17	18	19	20	21	22	23	24	E	
Confirm project	Blue																								
Research Literature	Blue	Blue	Blue	Blue	Blue	Blue																			
Modules	Blue	Blue	Blue	Blue																					
Project Proposal			Blue	Blue																					
Flow charts			Blue	Blue	Blue																				
Acquire hardware				Blue	Blue	Blue																			
Test hardware					Blue	Blue																			
Basic Entity design					Blue	Blue	Blue	Blue																	
B. Behav. design						Blue	Blue	Blue	Blue	Blue															
Synthesis							Blue	Blue	Blue	Blue															
Compile								Blue	Blue	Blue															
Test							Blue	Blue	Blue	Blue															
First Presentation									Blue	Blue															
First Report								Blue	Blue	Blue															
Advanced Entity											Blue	Blue	Blue	Blue											
Adv, Behaviour											Blue	Blue	Blue	Blue	Blue	Blue	Blue	Blue	Blue						
Adv. Synthesis												Blue	Blue	Blue	Blue	Blue	Blue	Blue	Blue						
Test																Blue	Blue	Blue	Blue	Blue	Blue				
Upgrade																	Blue	Blue	Blue	Blue	Blue				
Journal Article																		Blue	Blue	Blue					
Final Report																			Blue	Blue	Blue	Blue	Blue		
Performance																							Blue	Purple	
Final Presentation																							Blue	Purple	

Table 4: Semester 2 progress

4. Implementation of Design

In this chapter I will reiterate the designs that were developed, and then compare them to what was actually implemented. I will use the schematic diagrams within this chapter to make the comparisons, as they are easier to visually compare and follow, and I will reference the VHDL code which will be included at the end of this report, to verify the schematics.

4.1 Overall Design

This granular synthesis system is designed to receive a constant audio stream via an Analog to Digital Converter. The converted audio data is then processed and manipulated, based on the functionality controllers defined in the previous chapter. The processed data is then converted back to audio by a Digital to Analog Converter.

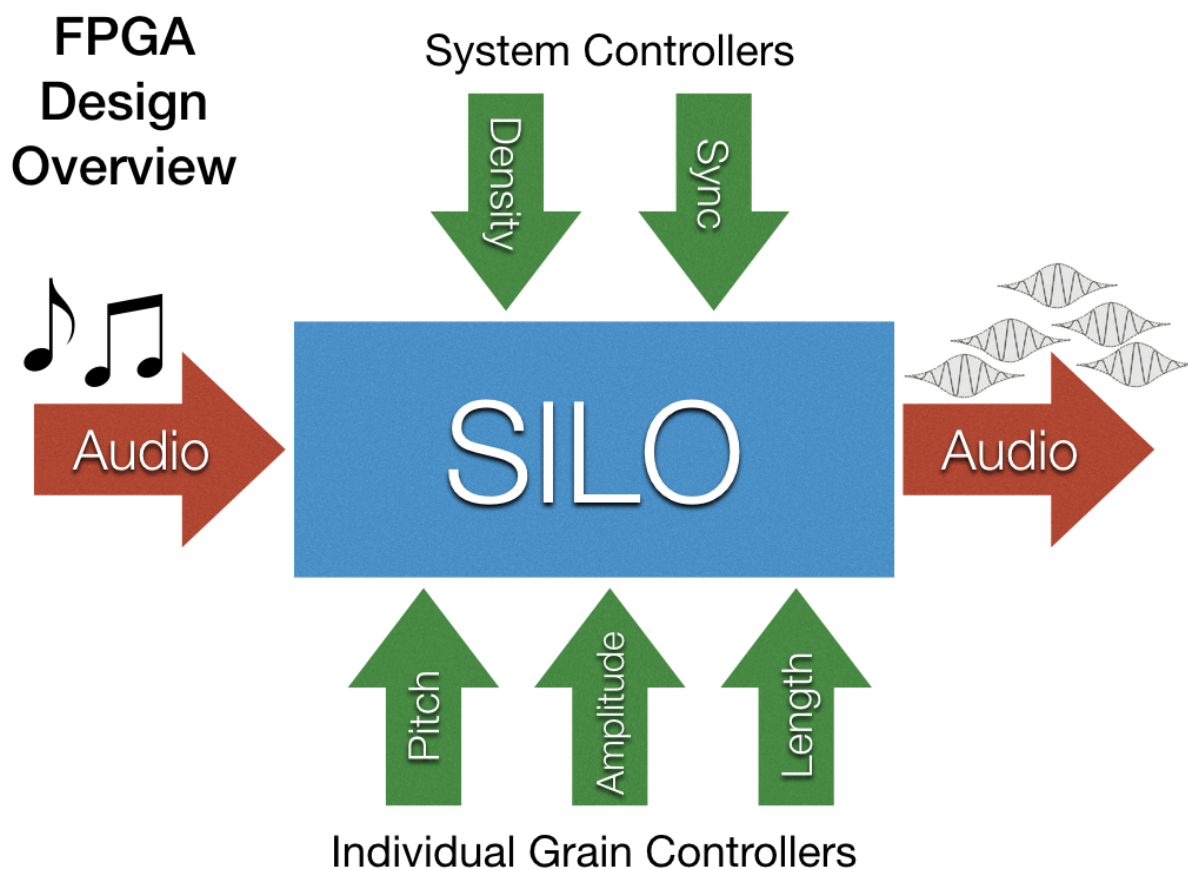


Figure 7: The overall design of the system

The system has been designed as a monophonic device, with a mono input, and mono output, which is common in audio hardware devices, as many instruments are mono. The audio input includes ¼ inch audio socket to plug in most audio device. This could be a recording, a microphone input, or an instrument, such as a keyboard, electric guitar or even another synthesizer. The PMOD-AMP3 actually contains two ⅛ inch mono jacks that could be used for stereo, using an adaptor, but per my specification, I have just created a mono output, and both outputs are identical. In the future I might consider spatialising the grains across a stereo spectrum. The audio output can then be plugged into a speaker, a mixing desk, or another synthesizer, if you want to create a modular chain of synthesizers.

Originally the controllers were going to be analog controllers, such as sliders or knobs, however after much thought I have decided the controllers will be digital inputs. This will of course make them easier to read, but the main reason they will be digital is because I have decided to implement them as MIDI input controllers.

MIDI is a communication protocol that stands for Musical Instrument Digital Interface. It was developed in 1983 by Dave Smith and uses an 8 bit communication protocol. MIDI can support up to 128 controllers simultaneously. The main reason I want to use MIDI however, is to allow a musician to easily bypass the built in controls and use their own controllers.

There are many forms of MIDI controllers, such as the traditional musical keyboards, and mixing board with sliders, knobs, but also more unique controllers such as breath controllers. Many musicians own at least one MIDI device. This will add a layer of usability that makes the device more versatile and multifaceted.

Unfortunately as I still do not have the MIDI interface device this cannot be implemented. I will however implement this feature at some point in the future, even if I have to create my own MIDI input device. There may be a market for this, considering how hard it was to find one in the first place, one that did not even deliver!

I acquired a 5000mAh battery with two usb outputs, so I could plug in both the FPGA and a speaker and make the system completely portable. With some retractable cords, this entire system easily and neatly packs into a small box. I have been using my mobile phone, generating a sine wave tone for testing.

The system currently looks like this when it is in use:

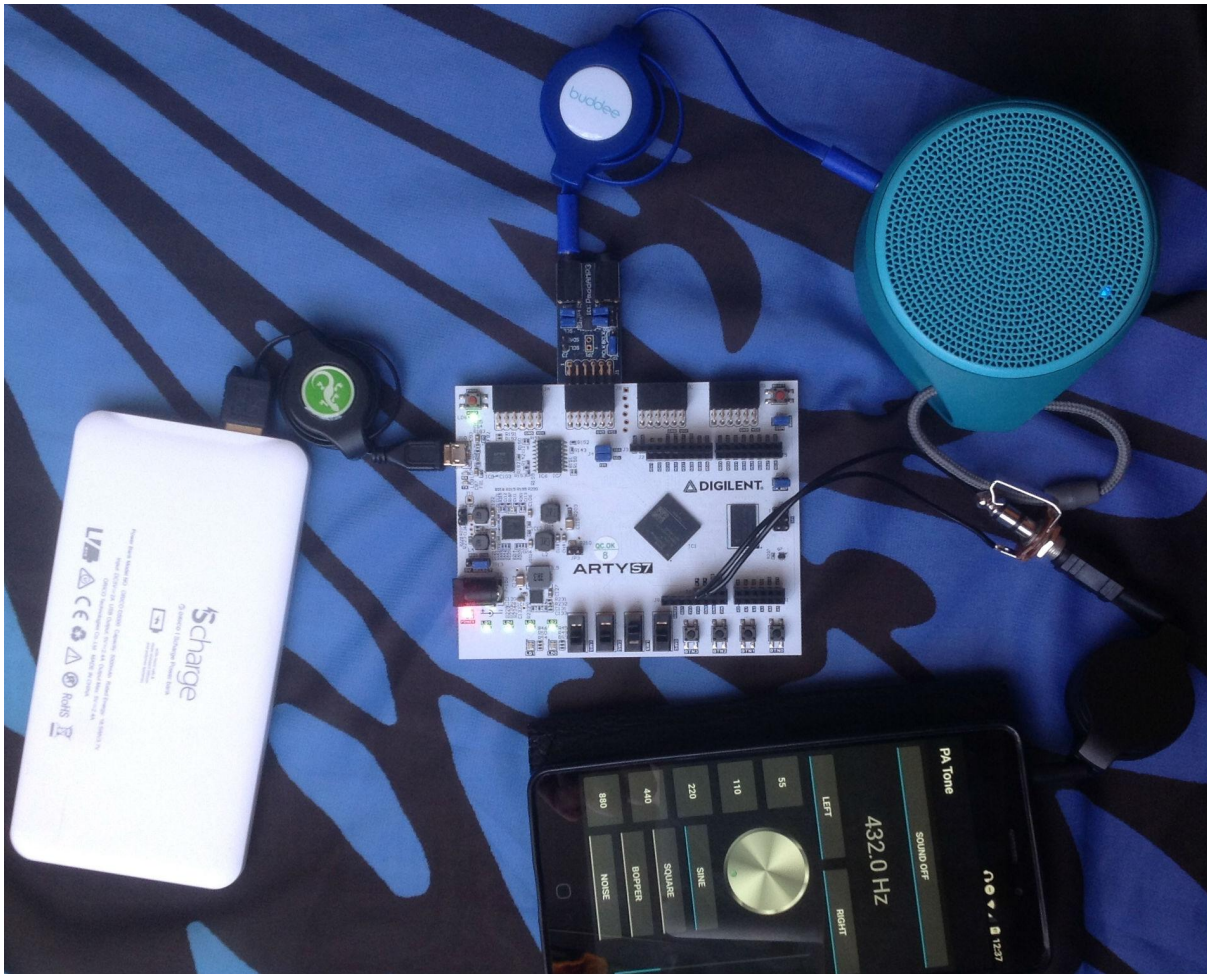


Figure 8: Arty S7 plugged into a battery, mobile phone, and speaker.

4.2 Terminology

Whilst designing the system, I defined terminology that is used to describe the various functions and features of the system. The following chart describes the terms used, plus it also lists the symbol used to define the term, and the default range and value. These terms are used to describe the design in section 4.3.

Term (Symbol)	Definition	Default Max Range
Amplitude (a)	The maximum allowable amplitude level of an individual grain.	Default: 1.0 $\sigma[0.0,1.0]$
Asynchronicity (s)	Each grain can be initiated synchronously, or allow a certain amount of random offset from the original initiation time. This offset renders the stream of grains asynchronous.	Default: 0.0 $\sigma[-t_i,t_i]$

Audio	In this system audio will refer to the analog or digital stream of audio	n/a
Counter (c)	To countdown event onsets	Default: i $\sigma[0,i]$
Density (ρ) (lowercase rho)	The number of grains initiated per second.	Default: 50 Per Sec $\sigma[0,2000]$
Envelope Shape (E)	The shape of the envelope applied to the grain, commonly a gaussian curve, although a simple linear ASR trapezoidal amplitude varying envelope is just as effective, and in some cases more useful, if you want to adjust the shape of the grain quickly.	n/a
Envelope Length (w)	The length of the envelope enclosing the grain, in other fields this may be referred to as a window. The length was derived from experiments in minimum sonic perception by Gabor in 1947 [1].	Default: 21ms $\sigma[10,50]$
Frequency (f)	The oscillation rate of sound measured over a second. By adjusting the oscillation rate it is possible to adjust the frequency of each grain. This can be accomplished with an FFT, shift, then inverse FFT process, or by controlling the input and output sample rates.	Default: 440Hz $\sigma[20,10000]$
Grain (G)	The sonic grain, originally defined as a Logon by Dennis Gabor in his 1947 publication the “Theory of Communication”, from the Journal of the Institution of Electrical Engineers.	n/a
Interonset (ti)	An acoustics term that defines the length of an interval between the onset of two successive sonic events. The onset, may be offset by a random value based on the asynchronous control.	Default: $1/\rho$ Per Sec $\sigma[n/a]$
Index (i)	For iterative processes	n/a
Poll Timer (t)	May be redundant - might use interonset to set polling event	n/a
Range ($\sigma[a,b]$)	σ denotes spread or range for each attribute of the grain where $[a,b] \Rightarrow \{x \in \mathbb{R}: a \leq x \leq b\}$. The input for this is a single controller so any change in this value will spread outward, with the base value in the centre.	n/a

Table 5: Terminology used in the design

4.3 Detailed Design

The next section contains detailed designs for the granular synthesis system, and the implemented schematic. It includes data flow, logic, and algorithms that will be implemented, and what has been implemented.

The next two pages show the overall system, in much more detail than figure 7.

The implementation diagram is broken into sections, each section defining a separate module that is given in more detail later on. For each separate module, there is also an associated implementation schematic so that the detail can also be observed in the implementation.

The implementation did not vary from the design, in any major way, although a few things seemed more beneficial to implement in different places, during the building stage. These discrepancies will be noted.

Essentially the system works by receiving audio data from the xadc, performing an optional pitch shift and storing in memory where the system is constantly reading from at the DVD audio sample rate (48KHz). That audio is then converted into a grain conforming to the size, envelope shape and density as supplied by the user. The grain is then played. When thousands of these grains play per second, each with their own individual settings, a sonic texture can emerge that is quite detailed and complex, providing a richer sonic experience than that offered by the original tone.

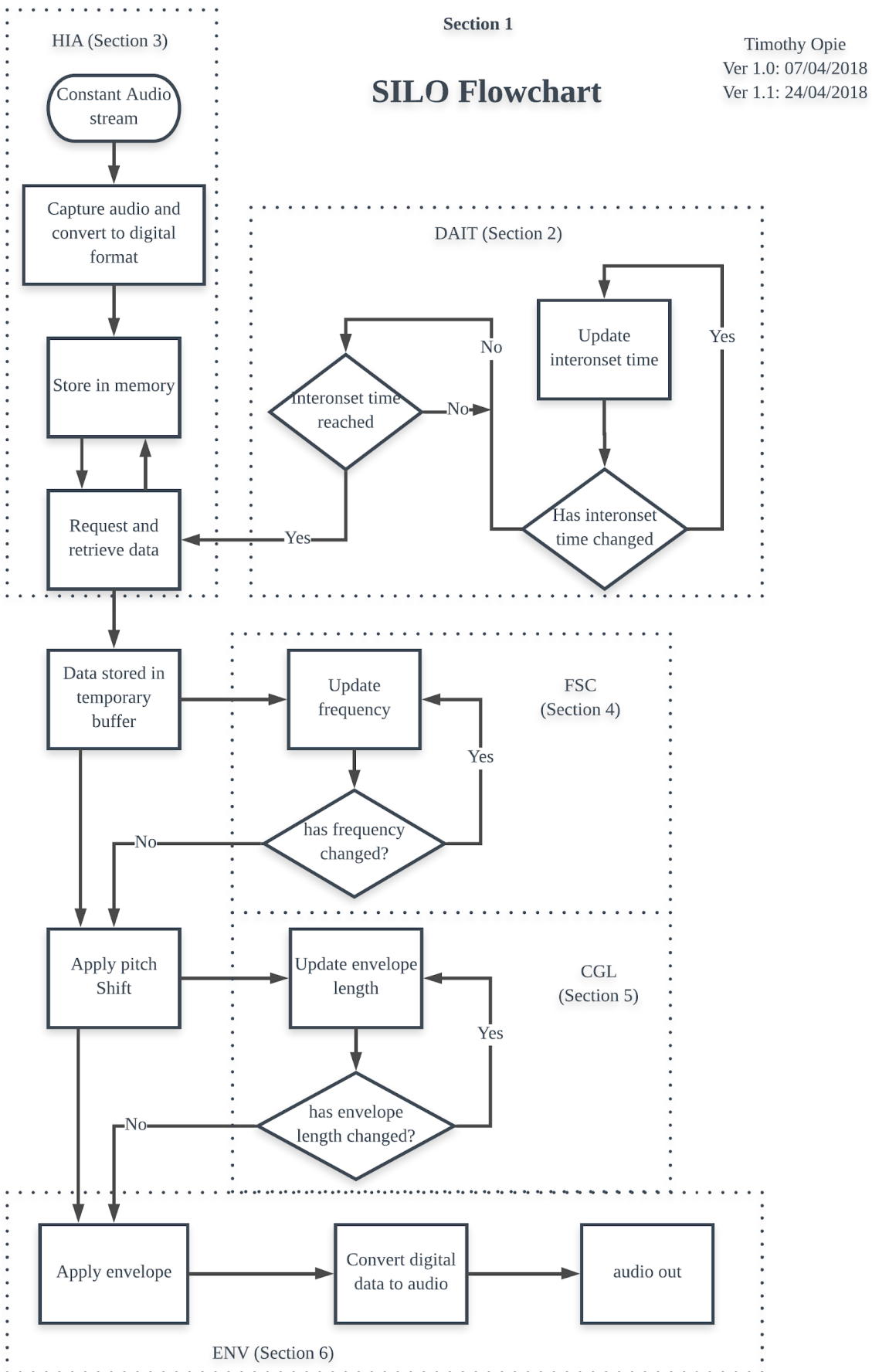


Figure 9: The overall design of the system

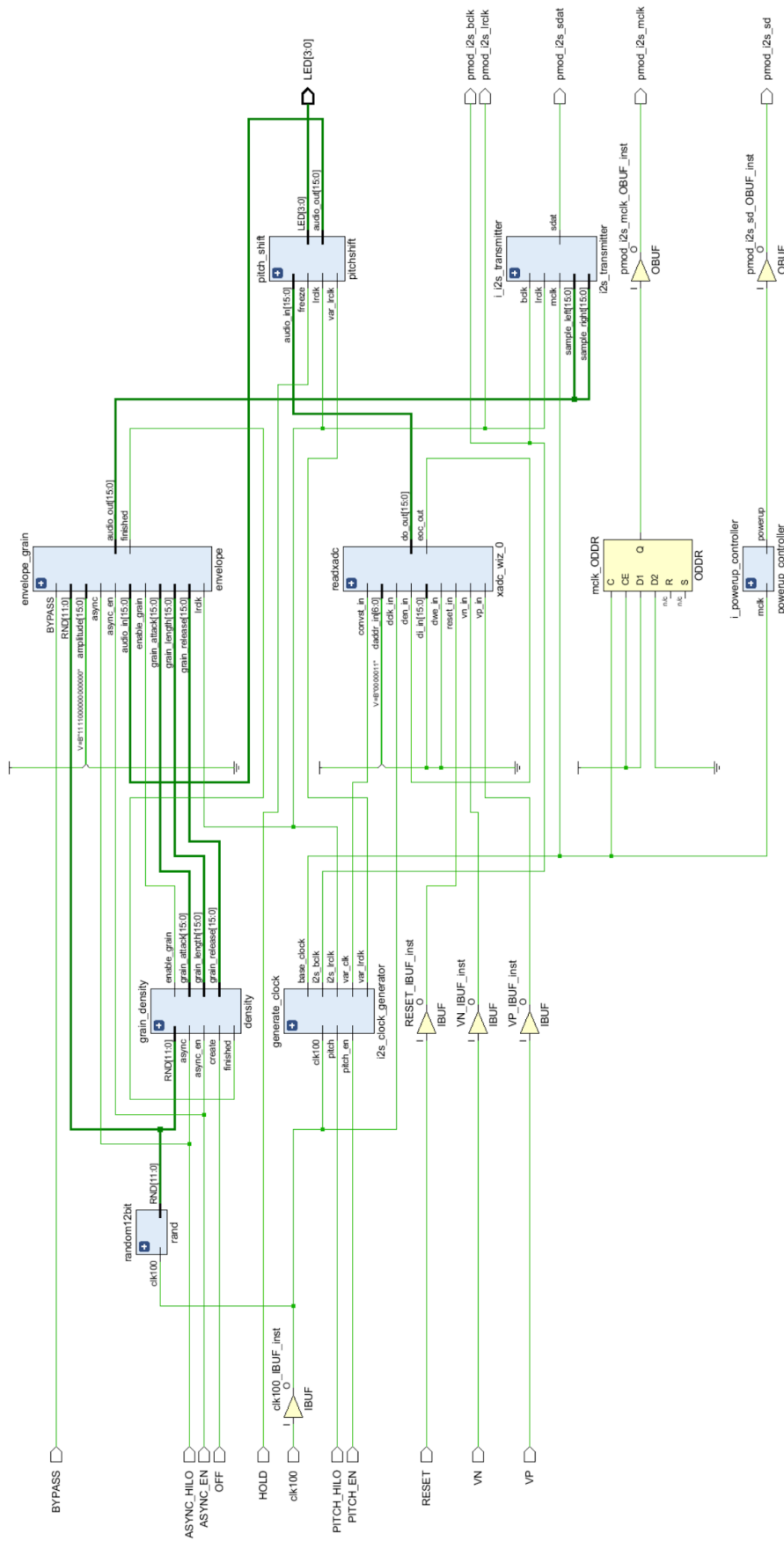


Figure 10: The overall implementation of the system

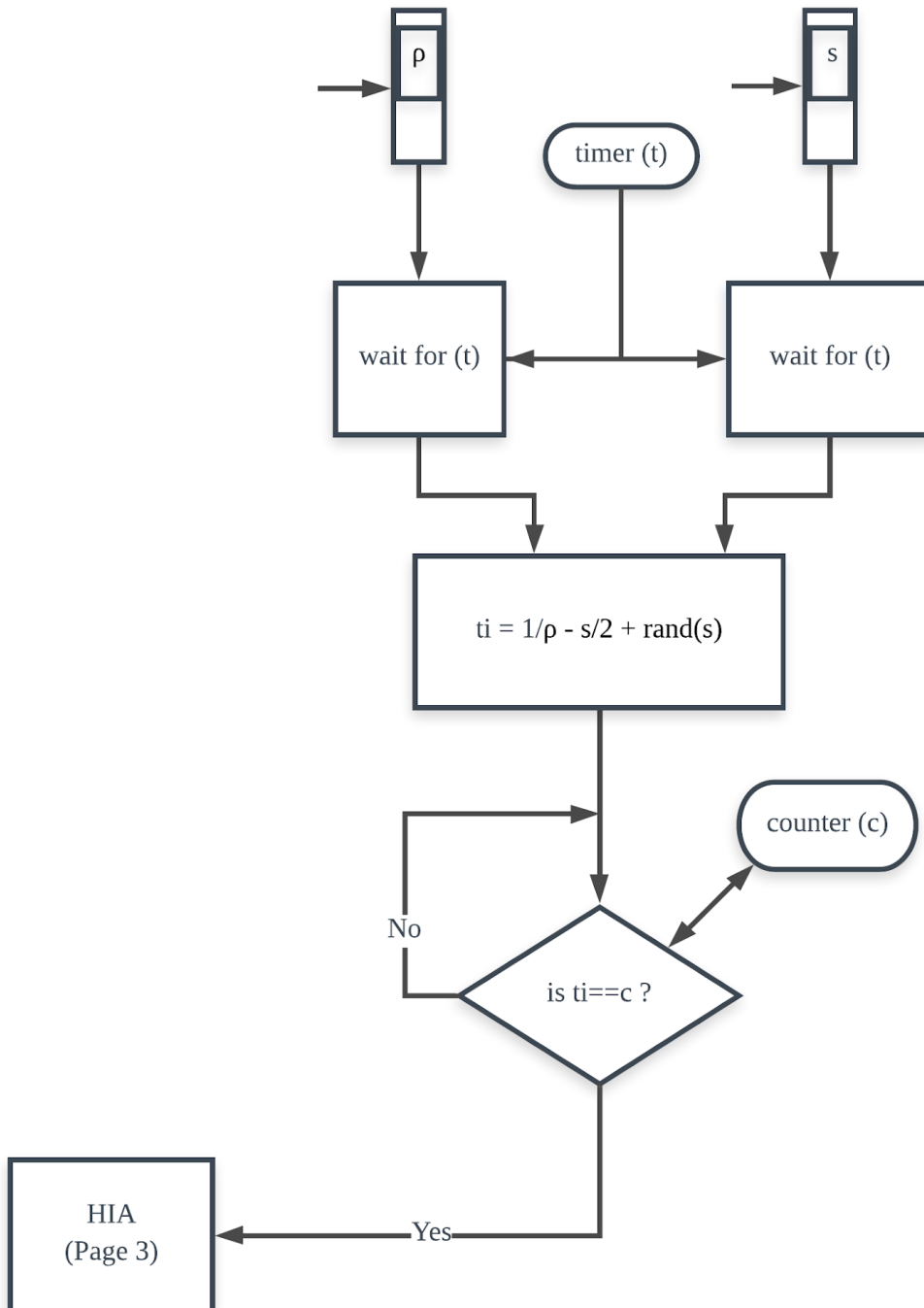
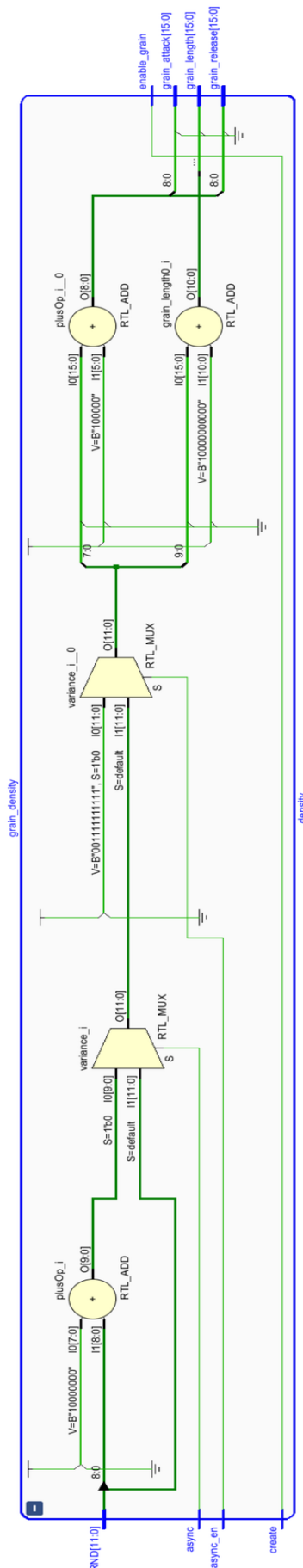


Figure 11: The design for grain density and grain timing, including asynchronous offset



< Figure 12: The design for grain density

This calculates when to enable a new grain. It also calculated the envelope shape and envelope length, as these attributes also contribute to the density.

The 12 bit random value passed in was calculated using the following polynomial:

$$\text{RND} [1 - 12] = f(x) [x3 - x14]$$

Where Polynomial: $f(x) = x35 + x33 + x30 + x29 + x24 + x22 + x21 + x17 + x14 + x9 + x8 + x7 + x4 + x2 + 1$

The code for the polynomial can be seen in the source code at the end of this document. It is just a standard LFSR.

The grain timing was carried out in a separate module that was responsible for the timing of all components, called generate clock. It calculates all the clocks required by the system, including the variable clocks for the XADC, the bitrate byteclock, and the sample rate lrclock.

The incoming system clock runs at 100MHz. I divide this by 8 to get a base rate of 12.5MHz, which is the rate at which the system sends audio bits. 100Mhz divided by 64 calculates the byte rate of 1.5625MHz. Dividing the byte rate by 32 gives me a sample rate of 48.828KHz which is just a tad over DVD sample rate quality. The variable clock rate uses the 100MHz system clock to adjust itself above and below the base rate to provide exact variable rate sample changes to enact pitch shifting.

The clock generator code was completely rewritten 5 times, this current system being extremely versatile and accurate despite seeming very sparse. The lack of code and idea to use various bits from the counter vector was a breakthrough in adding stability and keeping the logic well under a single 100MHz clock tick. I am very proud of it. (see figure 13)

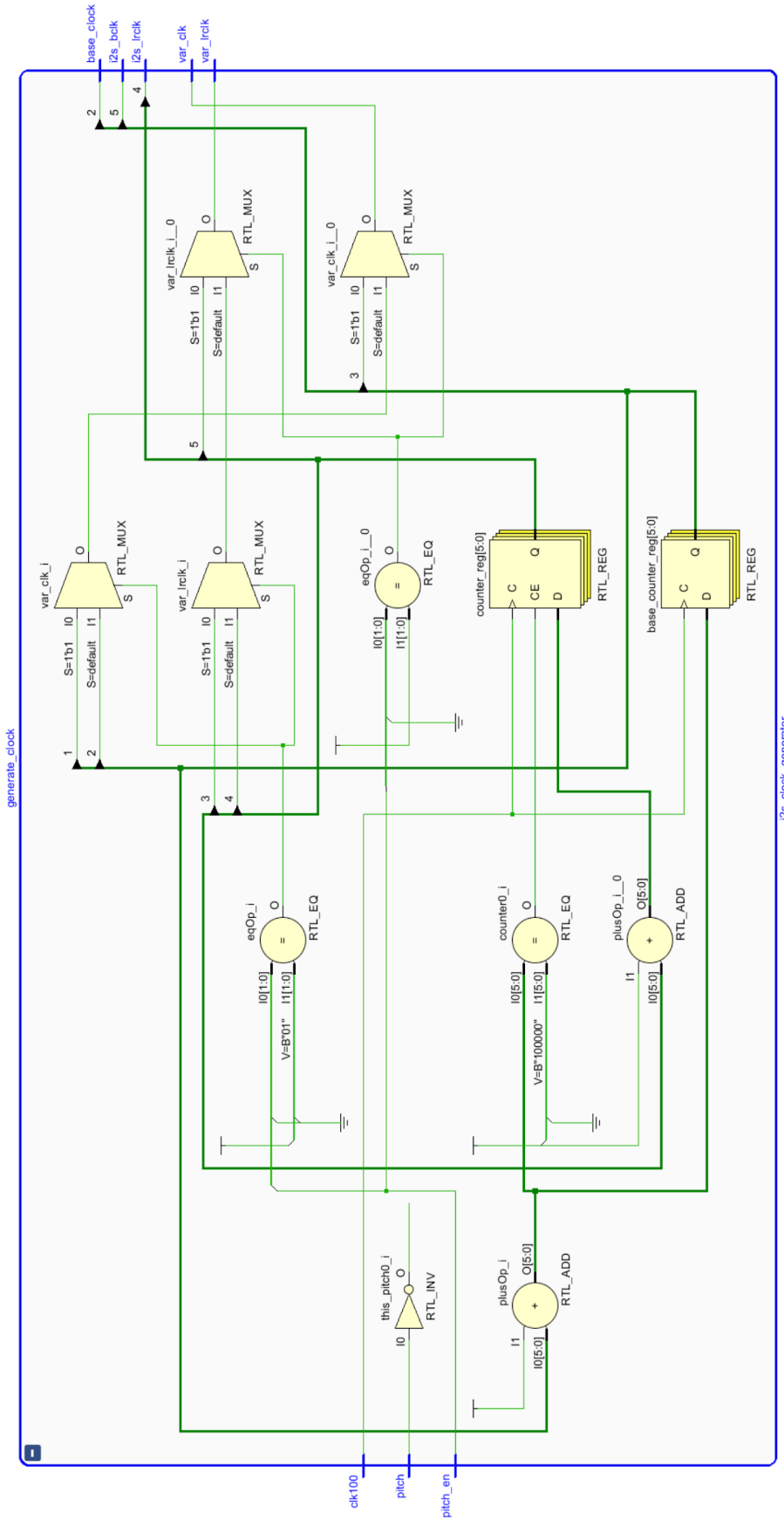


Figure 13: The timing module for the system.

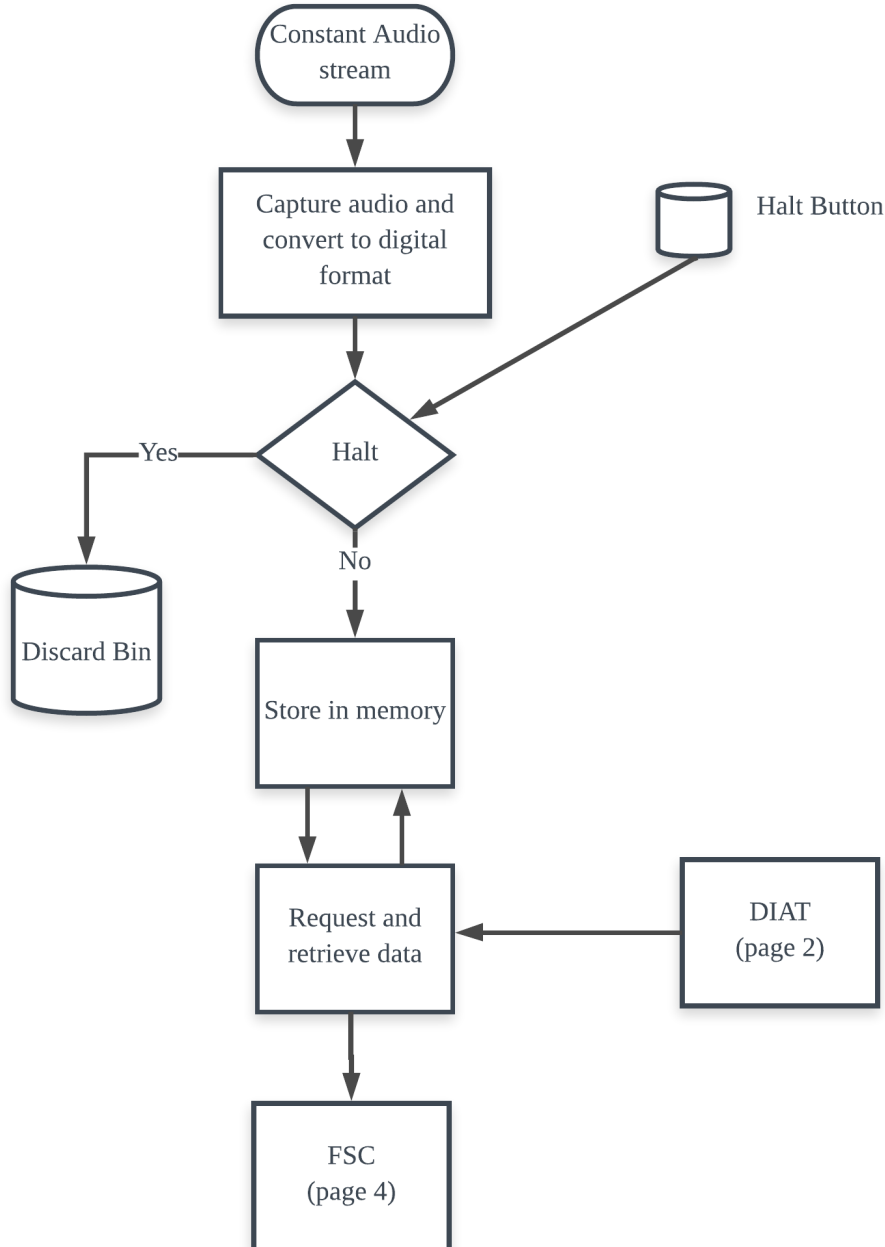
Halting Incoming Audio (HIA)

Figure 14: Halt incoming audio design

The audio halt algorithm was included in the pitch shift algorithm because it was decided to use the same memory space to store the current grain and perform the pitch shift. This means the system is always pulling data from the same location, whether or not it is updated or shifted in pitch. I think this was a fantastic idea for the implementation, and it works very well. (See figure 16 for implementation)

Section 4

Timothy Opie
Ver 1.0: 18/04/2018
Ver 1.1: 23/04/2018
Ver 1.2: 01/05/2018

Frequency Shifting Controller (FSC)

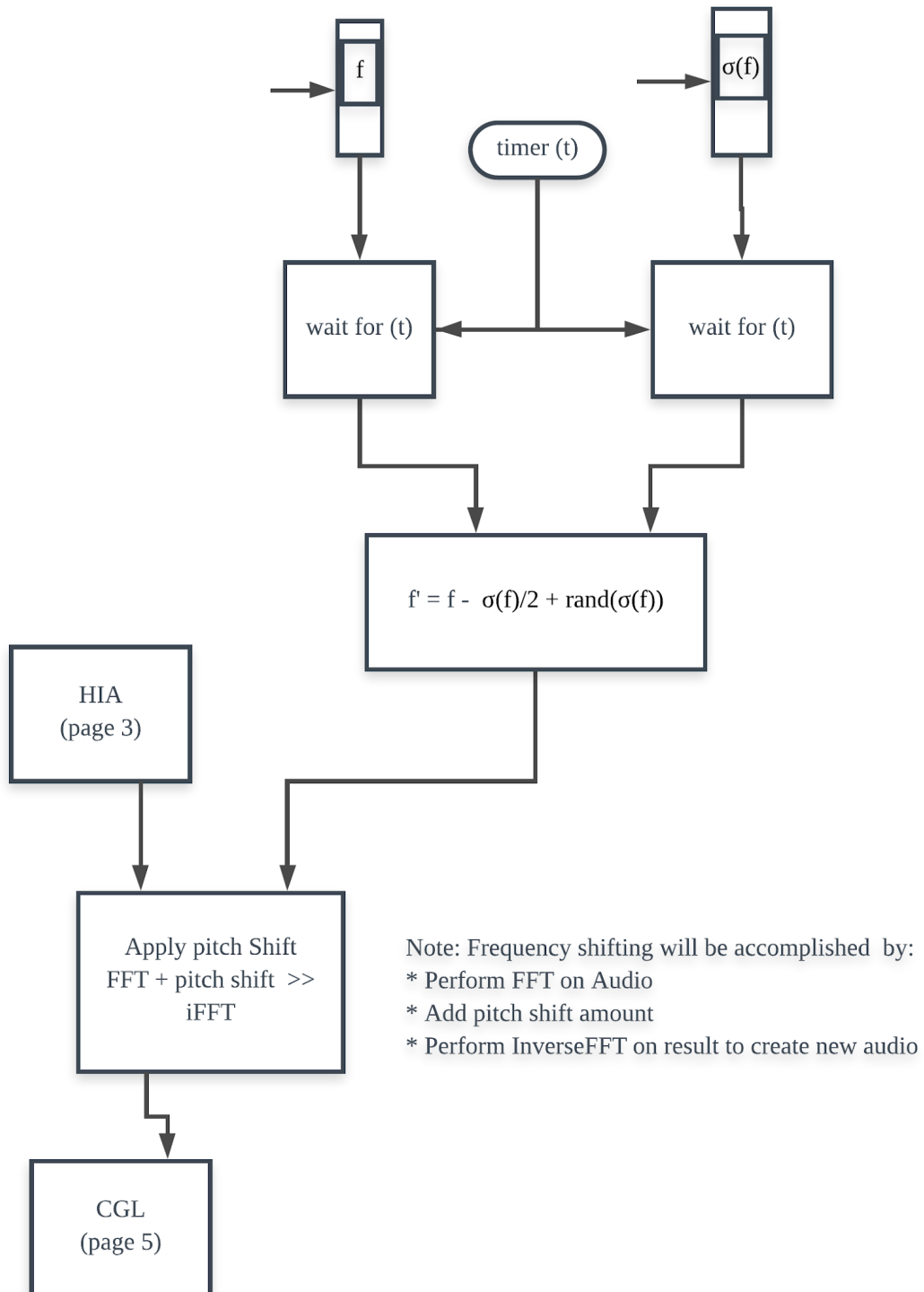


Figure 15: Pitch shift design

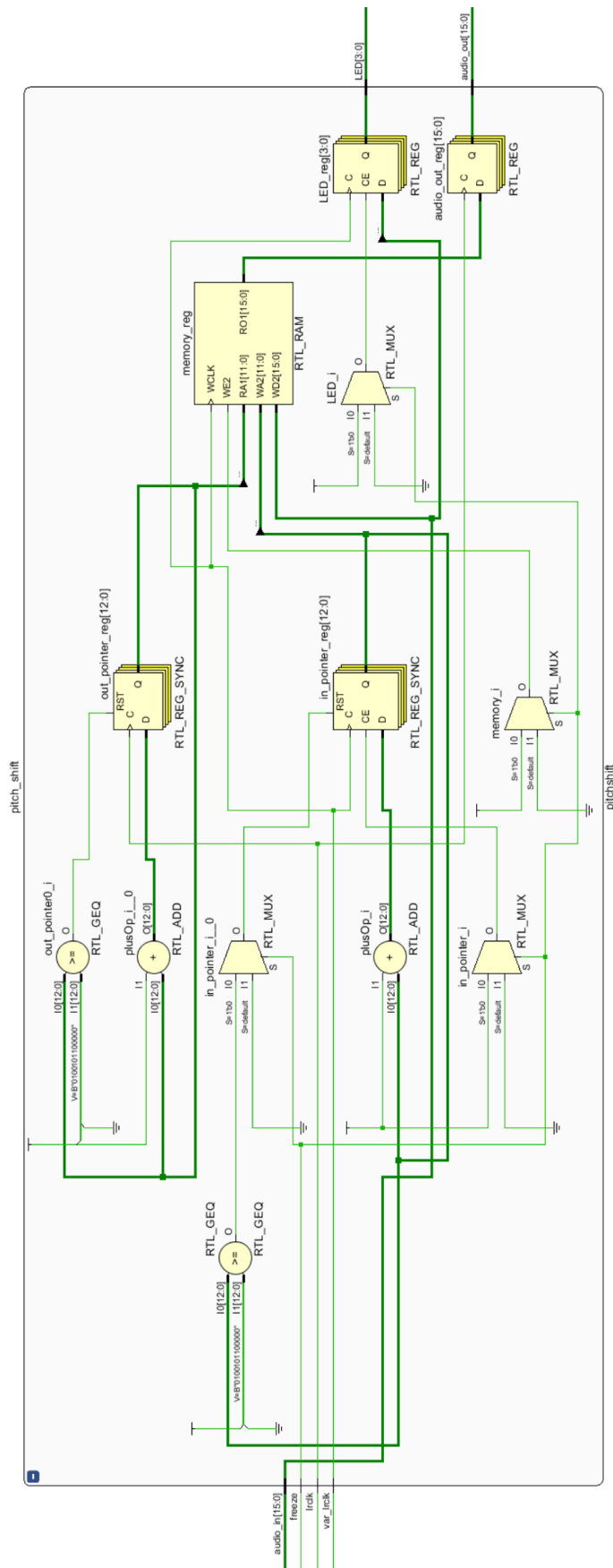


Figure 16: Pitch shift implementation

The original plan was to create an FFT and inverse FFT to control the pitch shift, however when this was implemented the shift always seemed inharmonic, and was also delayed by about 100ms, which is difficult to work with in live performance. The performer has to play slightly ahead of the beat to keep on the beat, and this becomes even more difficult when improvising with other musicians. You can't follow a musician if you have to play ahead of them. To put the delay into context, a modern song usually has a tempo of 140 beats per minute, which is about 430ms per beat. Having a delay of about 100ms already puts you out by quarter of a beat.

This process instead relies on adjusting the sample rate to achieve pitch shifting. I chose to change the sample rate at the input, as the inbuilt XADC is very fast, and I can control it very accurately. The outgoing audio is always running at 48Khz, without exception, to ensure a constant and smooth audio experience. The pitch shift module acts as a differential between the two rates, allowing the incoming rate to exceed the outgoing rate for shifting the pitch up, and vice versa. When it creates a sample it is copied into the memory array here. When the user halts input it just stops updating the memory array. The system tracks an input and output pointer to the memory to maintain coherency.

Calculate Grain Length (CGL)

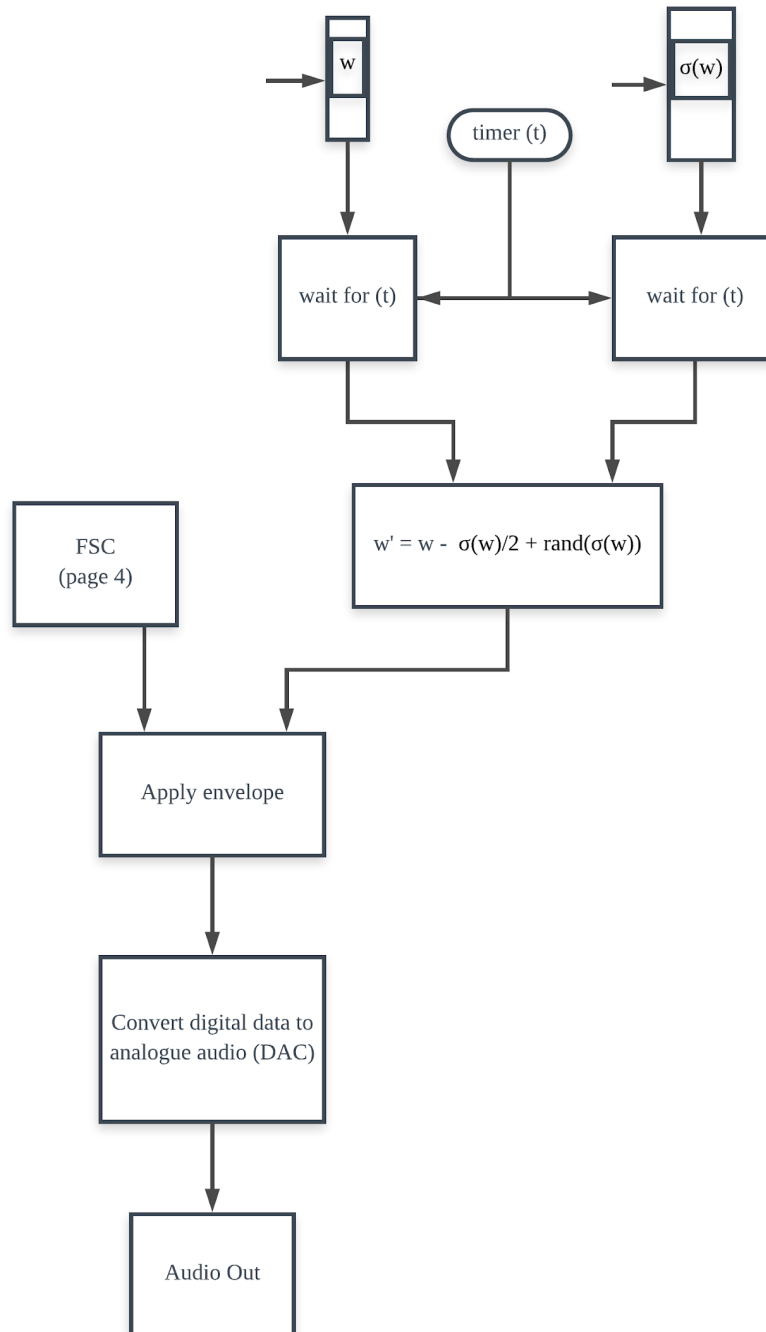


Figure 17: Grain length Calculation

Grain length is already calculated at the time the density is calculated in figure 12. This is because the length, density and grain overlap are all part of the same equation, and I deemed it more efficient to calculate these simultaneously

Section 6

Timothy Opie
Ver 1.0: 19/04/2018
Ver 1.1: 24/04/2018
Ver 1.2: 01/05/2018

Apply Envelope (ENV)

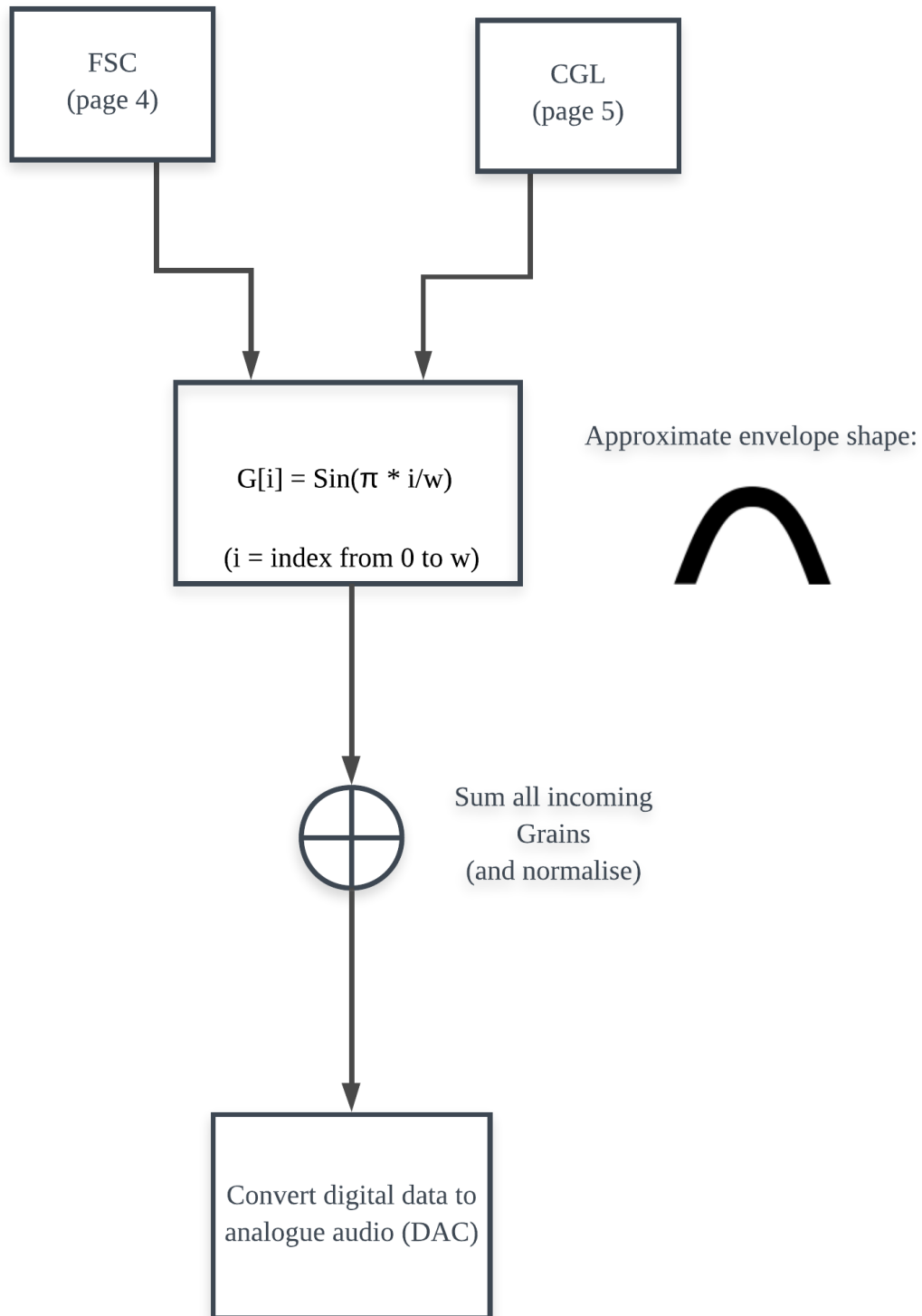


Figure 18: Apply the envelope design

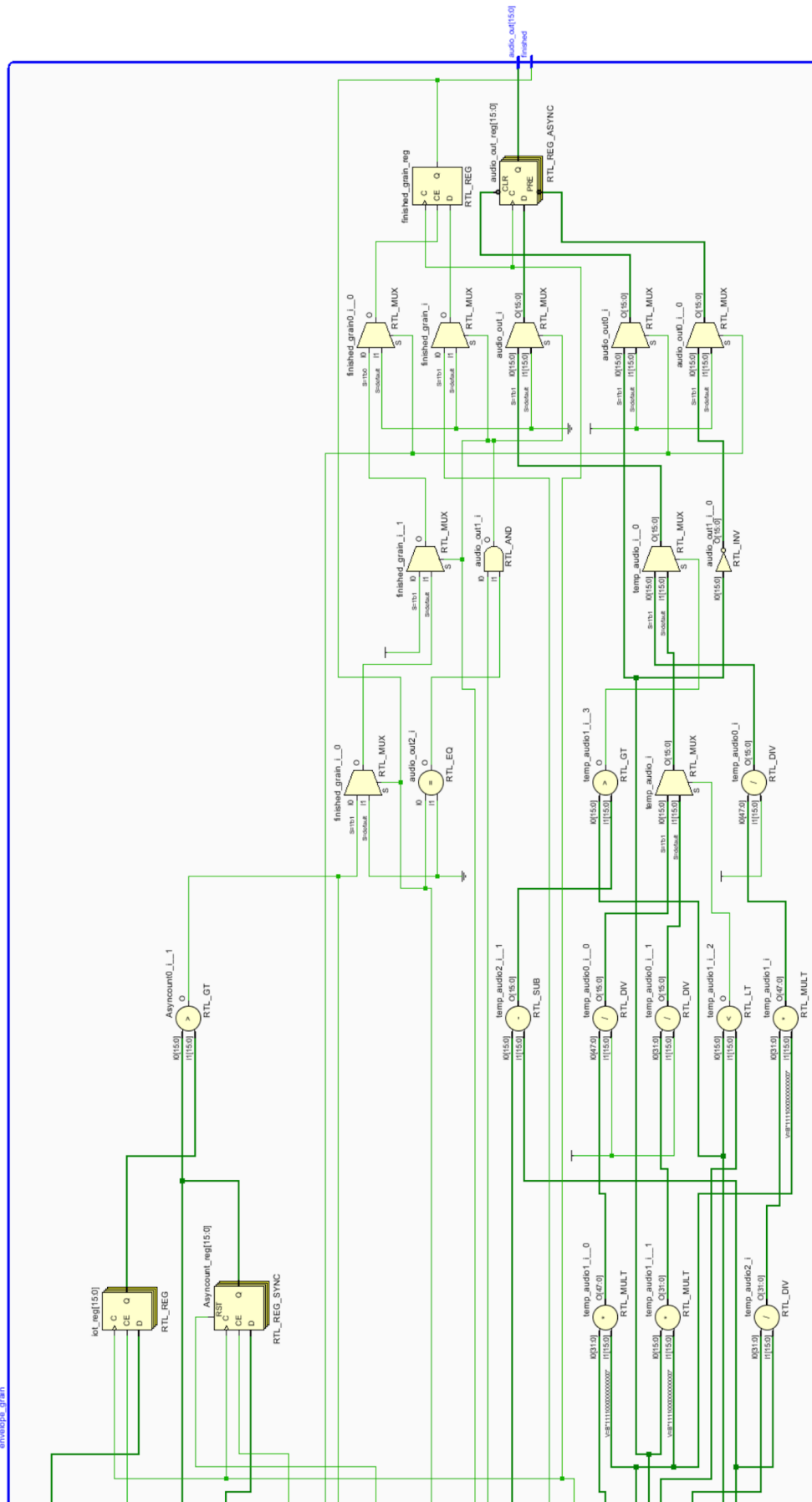


Figure 19 - Part 1: Apply the envelope implementation

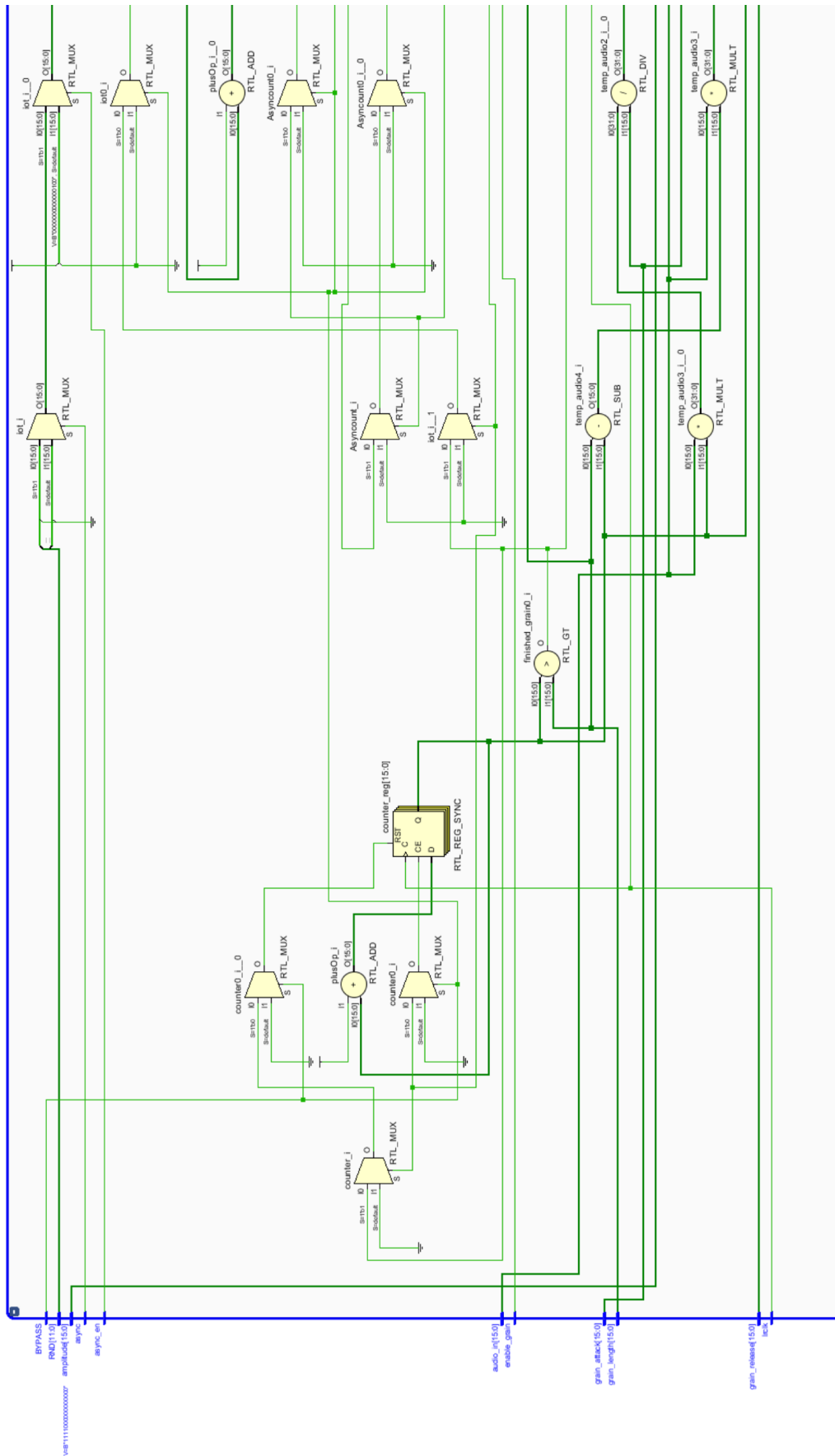


Figure 19 - Part 2: Apply the envelope implementation

The amplitude envelope implementation is quite detailed because this is where many of the components come together. All that is passed out of this module is the outgoing grain, and a finished flag, when each grain is completed.

The inputs for this module include when to begin creating a grain, with an optional random offset adjustment, the grain length, the attack and decay ramp of the trapezoidal amplitude envelope, bypass, and the incoming audio. At this point of the process the grain is complete, and can be sent to the output DAC.

4.4 Storing on QuadSPI

Storing the SILO program onto the QuadSPI allows for semi-permanent usage. That is, it will remain programmed through power cycles and resets, until the QSPI is reprogrammed.

As this was a new FPGA board, Xilinx had not yet published the procedures for storing via QSPI. Fortunately I was able to discover this myself through experimentation:

In Tools→Settings→Bitstream

- Tick .bin file

Synthesize

In Tools→Edit Device Properties

- set Enable Bitstream Compression to **TRUE**
- Under Configuration, set Configuration Rate (Mhz) to **50**
- Under Configuration Modes, select **Master SPI x4**

Generate Bitstream

Put Jumper on JP1

In the hardware manager select Add Configuration Memory Device

Add the memory device: **S25fl128s**

Select the SILO.bin file

After you click OK the process will begin. It takes about 1 minute.

5. Results

The device works, and can perform the functions as originally envisioned. I demonstrated the audio capabilities at the capstone expo to quite a large number of people who came to listen. I will also demonstrate it at the final presentation in November.

As stated in the methodology I would test the results aurally first, and then view the audio visually, to determine whether it suited the requirement.

Aurally the synthesis method works reasonably well. The XADC tends to normalise the signal - which is undesirable because I want to have complete control of the amplitude. Despite turning this feature off on the XADC it still remains on. I have notified Xilinx and am awaiting a response. It basically means that the signal to noise ratio is much lower than I would prefer, especially when the signal is very quiet. Also without the sliders, I cannot perform with nuance. I can however demonstrate all of the functions working, which was the goal of the project.

Referring to the functionality list I can now expand on the output in table 5:

Functionality	Description	Results
Density	The number of grains to be generated per second. This should be from 0 to at least 2000 grains per second.	It easily manages 0 to 2000 grains per second - however it currently does not allow for overlapping grains. This can be addressed by creating parallel streams on the FPGA. According to the utilisation chart [See figure 20]. there are enough resources on the FPGA to create 9 more streams. It would just require duplicating most of the modules, and adding one small module that kept track of which streams were activate.
ASynchronicity	Determine whether the grains are produced synchronously, and if not, the degree of synchronicity allowed	Asynchronicity works well, it is just a pity I don't have a slider to change the variance of the asynchronicity, currently using buttons I only have 3 levels of asynchronicity, So it is either synchronous, somewhat asynchronous, and very asynchronous. [See figure 23].
Envelope Length	The length of the audio segment, with the envelope applied. The	The envelope length works as expected. Again a slider would provide more variation and different levels of variance, instead of relying on a switch and 2 levels of randomness to change length. The

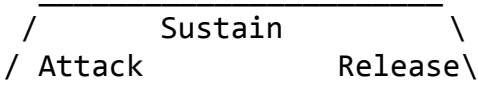
	envelope itself will be a gaussian curve. Range 15-40ms. Default will be 21ms	attack and release of the trapezoid are also both independent, which is a bonus feature not originally proposed. 
Frequency	Shift the grain frequency within the audio hearing range	Frequency can shift up or down an octave with a high degree of clarity, using button switch combination. [See figure 21 & 22] for more details.
Amplitude	The peak amplitude of each grain, from 0.0 to 1.0	Complete amplitude control, created and utilised in envelope creation. It sets the maximum level of the envelope allowed.
Input Data control	Stream continuous audio or hold audio in one position - allows time stretching and elongation of sounds	Can stream or hold on a single grain, at the press of a button. Works beautifully.

Table 5: Functionality with results

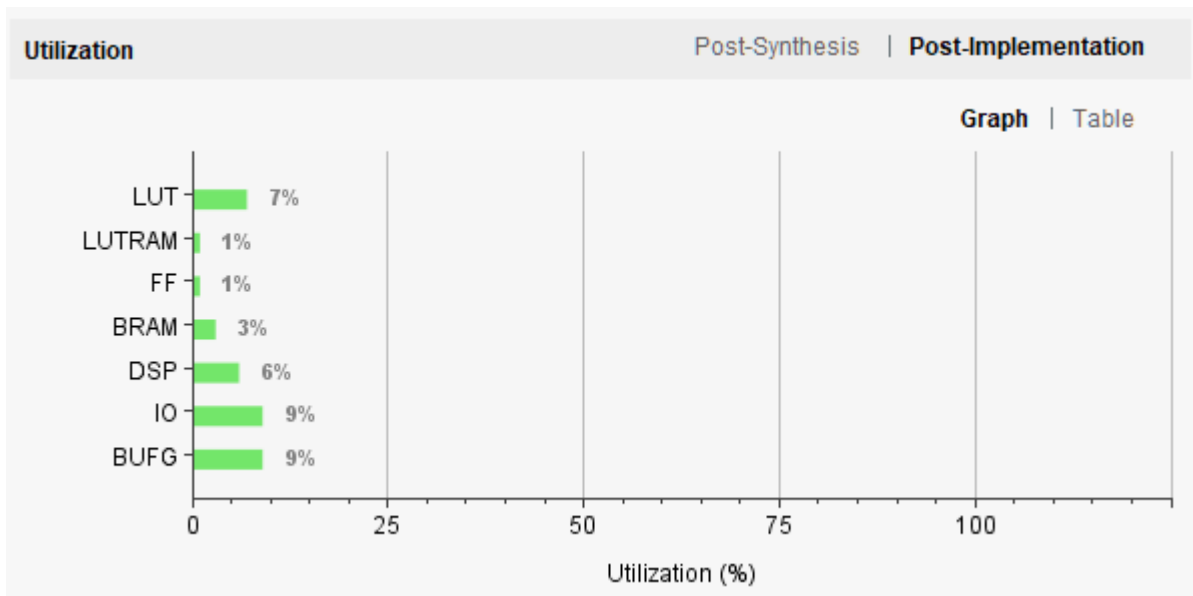


Figure 20: Xilinx utilisation summary of the project - I still have over 90% resources free

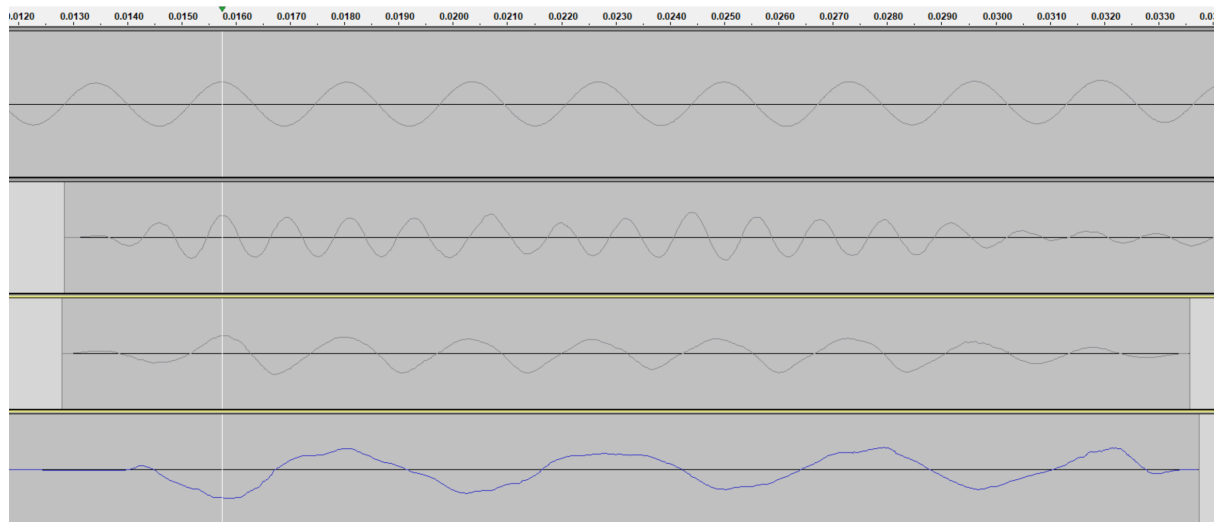


Figure 21: 3 grains lined up under the source audio - line on source peak

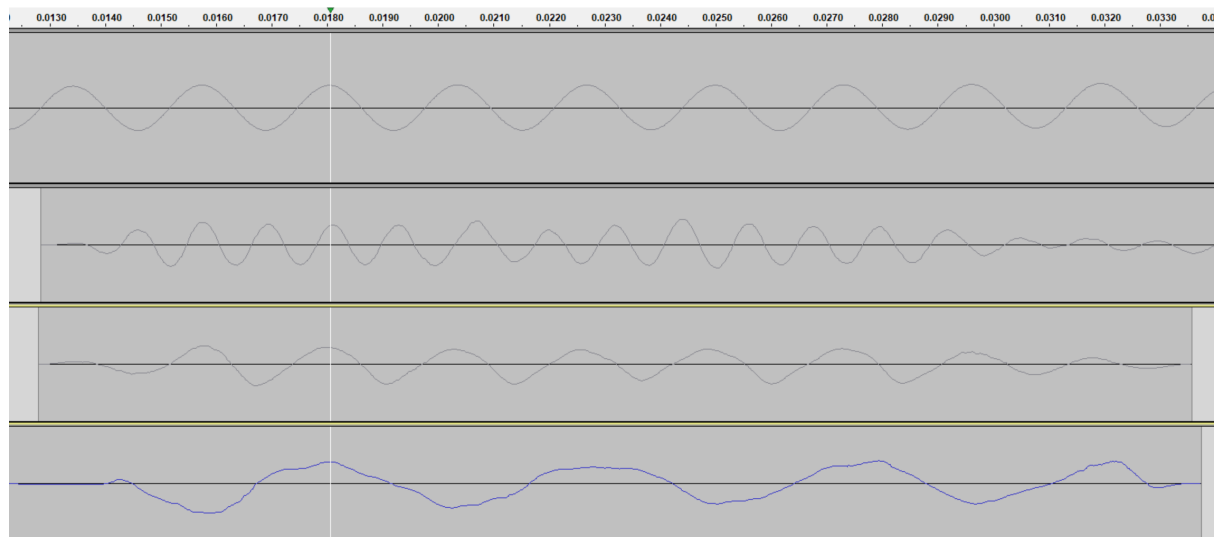


Figure 22: 3 grains lined up under the source audio - line on next source peak

When we line up three grains of varying frequency we can measure the effectiveness of the pitch shift.

The source is 432Hz

Grain 1: should be 864 Hz

Grain 2: should be 432 Hz

Grain 3: should be 216 Hz

As we can see in one cycle of the source:

Grain 1 has just completed 2 cycles - which is correct.

Grain 2 has just completed 1 cycle - which is correct.

Grain 3 has just completed $\frac{1}{2}$ a cycle - which is correct.



Figure 23: Example of Asynchronicity.

You can see the individual grains with variable start times and lengths

The other thing I mentioned explicitly to monitor in the methodology was the timing. The reports below report good news:

Timing	
Worst Negative Slack (WNS):	6.518 ns
Total Negative Slack (TNS):	0 ns
Number of Failing Endpoints:	0
Total Number of Endpoints:	50
Implemented Timing Report	

Figure 24: Xilinx timing summary

Max Delay Paths

```

-----
Slack (MET) :          6.518ns (required time - arrival time)
  Source:
    random12bit/x9_reg/C
    (rising edge-triggered cell FDRE clocked by sys_clk_pin
{rise@0.000ns fall@5.000ns period=10.000ns})
  Destination:
    random12bit/x10_reg/D
    (rising edge-triggered cell FDRE clocked by sys_clk_pin
{rise@0.000ns fall@5.000ns period=10.000ns})
  Path Group:
    sys_clk_pin
  Path Type:
    Setup (Max at Slow Process Corner)
  Requirement:
    10.000ns (sys_clk_pin rise@10.000ns - sys_clk_pin
rise@0.000ns)
  Data Path Delay:      3.357ns (logic 0.456ns (13.583%) route 2.901ns (86.417%))
  Logic Levels:         0
  Clock Path Skew:      0.015ns (DCD - SCD + CPR)
    Destination Clock Delay (DCD):      4.270ns = ( 14.270 - 10.000 )
    Source Clock Delay (SCD): 4.484ns
    Clock Pessimism Removal (CPR):      0.230ns
  Clock Uncertainty:    0.035ns ((TSJ^2 + TIJ^2)^1/2 + DJ) / 2 + PE
    Total System Jitter (TSJ): 0.071ns
    Total Input Jitter (TIJ): 0.000ns
    Discrete Jitter (DJ): 0.000ns
    Phase Error (PE): 0.000ns
  
```

Table 6: Snippet from Xilinx max delay report

Looking at the timing report in table 6 we can see the worst negative slack in the implementation is 6.518 NS. Looking more closely at this in figure 25 we can see it occurs generating random numbers. The timing requirement is 10 NS at 100 MHz, so this easily falls within the range.

6. Further Discussion

In this chapter I will discuss some development issues, the results, and also look at what I would like to do next with this project.

6.1 Timing Issues

This project was initially plagued by timing issues. Working with audio requires precise timing. As I mentioned around figure 12, I rewrote the timing code 5 times, looking for a more precise method. Originally it was very complex, and resulted in poor quality audio. I then tried using the system 12 MHz clock for the timing which worked well, unless I wanted to perform pitch shifting. So I went back to the 100MHz clock and looked at better ways to divide it. I needed 5 clocks and it occurred to me that a counter would give me different clocks based on which bit I was using. This concept totally changed the way I performed timing and made the process very simple, yet very accurate. I had to use two counters because I could not divide one counter into the required bits, but by creating a second counter that incremented every 32nd clock (100000b) allowed me to create a subset to achieve the remaining required clock values..

```
clk_process: process(clk100)
  begin
    if rising_edge(clk100) then
      if (base_counter + 1 = "100000") then
        counter <= counter + 1;
      end if;
      base_counter <= base_counter + 1;
    end if;
  end process;

  this_pitch <= not pitch & pitch_en;
  base_clock <= std_logic(base_counter(2));

  i2s_bclk <= std_logic(base_counter(5));

  var_clk <= std_logic(base_counter(3)) when this_pitch = "11" else
    std_logic(base_counter(1)) when this_pitch = "01" else
    std_logic(base_counter(2));

  i2s_lrclk <= std_logic(counter(4)); -- about 48KHz
```

```
var_lrcclk <= std_logic(counter(5)) when this_pitch = "11" else
    std_logic(counter(3)) when this_pitch = "01" else -- Pitch Lower
    std_logic(counter(4)); -- Pitch normal
end Behavioral;
```

6.2 Analog to Digital Conversion Issues:

The XADC analog to digital converter can sample at a rate up to 100 MHz. It is very configurable, but so complex it needs to be initiated through a Xilinx IP. I was able to figure out the IP, but when I first began work on this project there was just one demo and it was implemented as a block design, not in VHDL or even Verilog, so it took me quite a long time experimenting to discover where and how to trigger it correctly, using the appropriate pins on the board, and the correct byte code to tell the XADC which pins to use.

Even when that was finally working the audio signal was still low quality. At first I thought it was the timing causing the issue, but asking people on the Xilinx forums I was informed that it required AC coupling to move the signal into the positive range, because it was only reading 0v to 1v. This meant building some extra circuitry. I later found out however that the XADC has a bi-polar option that will change the sampling mechanism to read -0.5v to 0.5v. This was a lot simpler and almost immediately fixed the issue. There were also settings to adjust the gain and offset, which I turned off, however, it is still adding gain and I have not resolved that yet. This means that the signal to noise ratio is low, and as the signal gets softer the noise gets louder, because of the auto gain. I am still hoping to resolve this. It doesn't affect the functionality of the system, but it does reduce the quality.

6.3 Discussion of Results

Despite a few quality flaws, I believe I have created a prototype that fulfills the functionality requested. As already discussed I was unable to obtain the slider component I wanted to use, despite ordering 5 months ago. I have come to the realisation that I will never get the part I ordered, and the distributors will no longer help or communicate with me. I have however designed the system to function without it. I have less control over the functionality, but I still have enough control to demonstrate that all functional requirements were met.

The implementation was quite similar to the design proposed. I optimised it in a few places whilst building where I realised the design was more cumbersome than it needed to be. I think the implementation is now better than the original design.

I am very happy with the timing being well below the required clock rate. It is something I was vigilant about, and it paid off.

The FPGA utilisation is less than 10%. One of my earlier pitch shift modules used up 80% of the board, and took over half an hour to synthesize. I realised that this was not a good method, so I changed the pitch shift method, after discussing the issue with some audio engineers. The new method just relies on an accurate clock and a good ADC. As you could see in figure 21 and 22, the pitch shifting is quite accurate, although there is a little noise in the signal, which i hope to improve. I am still keen to run parallel streams and use more of this space to create a denser texture, however I don't want to do this until all quality issues are resolved, in case it involves a major rewrite of some component. I doubt it will, but I just want to be sure.

6.4 Future Prospects

I plan on working on this project beyond my degree, because I believe it is useful, it shows a lot of promise, and I believe it can be improved well beyond what other hardware developers have done with this synthesis technique. I would even say that my current prototype is already close to the two commercial hardware implementations I discussed in the literature review. I would eventually like to have a full implementation on an ASIC that can be included in a small synthesizer module that can be added to a synthesizer rig.

I also plan on making my own MIDI to FPGA controller because I think it will be useful to me, and also a wider public. I actually think that component alone would make a nice capstone project, and I might have considered it as an alternative to this project because the scope is much more defined, it is easier to test, and it would have been a better sized project for the time given. This project on the other hand has taken many many more hours than the 12 hours per week allotted to it. I don't mind because it is a passion project.

I am planning on presenting the SILO hardware granular synthesizer at the next international NIME (New Instruments for Musical Expression) conference, and the International Computer Music Conference. I believe it will be very popular at both of those conferences.

7. Conclusion

Originally I set out to create a hardware granular synthesizer that could match the functionality of the software granular synthesis implementations. After defining these functions, designing a solution, and building an implementation of the solution, I believe I can answer quite confidently that yes, you can build a hardware granular synthesizer to match the functionality.

Even with my limited but rapidly growing VHDL knowledge I have been able to create a usable prototype, that still has room for improvement, but very clearly addresses the thesis made in this project. I have enjoyed working on this project, and look forward to improving my VHDL skills well beyond my degree. I hope to use the SILO on a large stage one day.

8. Source Code and Reports

All VHDL code I created is listed below:

silovhd

```
-----  
-----  
--      silo.vhd  
-----  
-- SILO, The Solid State Granular Synthesizer  
-----  
-- Author: Timothy Opie  
-- Copyright 2018  
-----  
-----  
-- Libraries  
-----  
library IEEE;  
use IEEE.STD_LOGIC_1164.ALL;  
use IEEE.std_logic_unsigned.all;  
USE IEEE.NUMERIC_STD.ALL;  
use IEEE.math_real.all;  
  
Library UNISIM;  
use UNISIM.VComponents.all;  
  
-----  
-- Entity  
-----  
entity SILO is  
  Port (  
    clk100      : in  STD_LOGIC;  
    --## xadc  
    VP          : in  STD_LOGIC;  -- pin: j10  
    VN          : in  STD_LOGIC;  -- pin: k9  
    --## Buttons  
    RESET       : in  STD_LOGIC;  -- btn[0]  
    HOLD        : in  STD_LOGIC;  -- btn[1]  
    ASYNC_HILO  : in  STD_LOGIC;  -- btn[2]  
    PITCH_HILO  : in  STD_LOGIC;  -- btn[3]  
    --# Switches  
    OFF         : in  STD_LOGIC;  -- sw[0]  
    BYPASS      : in  STD_LOGIC;  -- sw[1]  
    ASYNC_EN    : in  STD_LOGIC;  -- sw[2]  
    PITCH_EN    : in  STD_LOGIC;  -- sw[3]
```

```

        LED                : out STD_LOGIC_VECTOR (3 downto 0);
        --## pmod amp3 - JB
        pmod_i2s_sd        : out STD_LOGIC := '0';-- pin: jb[7]
        pmod_i2s_mclk      : out STD_LOGIC; -- pin: jb[6]
        pmod_i2s_bclk      : out STD_LOGIC; -- pin: jb[3]
        pmod_i2s_lrclk     : out STD_LOGIC; -- pin: jb[0]
        pmod_i2s_sdat      : out STD_LOGIC -- pin: jb[1]
    );
end SILO;

-----
-- Components
-----\
architecture Behavioral of SILO is

    component xadc_wiz_0 is
    port (
        daddr_in          : in  STD_LOGIC_VECTOR (6 downto 0);    -- Address bus for
the dynamic reconfiguration port
        den_in            : in  STD_LOGIC;                        -- Enable Signal for
the dynamic reconfiguration port
        di_in             : in  STD_LOGIC_VECTOR (15 downto 0);   -- Input data bus for
the dynamic reconfiguration port
        dwe_in            : in  STD_LOGIC;                        -- Write Enable for
the dynamic reconfiguration port
        do_out            : out STD_LOGIC_VECTOR (15 downto 0);   -- Output data bus
for dynamic reconfiguration port
        drdy_out          : out STD_LOGIC;                        -- Data ready signal
for the dynamic reconfiguration port
        dclk_in           : in  STD_LOGIC;                        -- Clock input for
the dynamic reconfiguration port
        reset_in          : in  STD_LOGIC;                        -- Reset signal for
the System Monitor control Logic
        convst_in         : in  STD_LOGIC;                        -- Convert Start
Input
        busy_out          : out STD_LOGIC;                        -- ADC Busy signal
        channel_out       : out STD_LOGIC_VECTOR (4 downto 0);   -- Channel Selection
Outputs
        eoc_out           : out STD_LOGIC;                        -- End of Conversion
Signal
        eos_out           : out STD_LOGIC;                        -- End of Sequence
Signal
        alarm_out         : out STD_LOGIC;                        -- OR'ed output of
all the Alarms
        vp_in             : in  STD_LOGIC;                        -- Dedicated Analog
Input Pair
        vn_in             : in  STD_LOGIC
    );
    end component;

```

```

component pitchshift is
port (
  lrclk      : in STD_LOGIC;
  freeze     : in STD_LOGIC;
  var_lrclk  : in STD_LOGIC;
  audio_in   : IN  STD_LOGIC_VECTOR(15 DOWNTO 0);
  audio_out  : OUT STD_LOGIC_VECTOR(15 DOWNTO 0);
  LED        : out STD_LOGIC_VECTOR (3 downto 0)
);
END component;

component i2s_clock_generator is
Port (
  clk100     : in STD_LOGIC;
  pitch_en   : in STD_LOGIC;
  pitch      : in STD_LOGIC;
  var_clk    : out STD_LOGIC;
  var_lrclk  : out STD_LOGIC;
  base_clock : out STD_LOGIC;
  i2s_bclk   : out STD_LOGIC;
  i2s_lrclk  : out STD_LOGIC
);
end component;

component powerup_controller is
Port ( mclk      : in  STD_LOGIC;
        powerup   : out STD_LOGIC);
end component;

component i2s_transmitter is
Port ( mclk      : in  STD_LOGIC;
        bclk      : in  STD_LOGIC;
        lrclk     : in  STD_LOGIC;
        sample_left : in  STD_LOGIC_VECTOR (15 downto 0);
        sample_right : in  STD_LOGIC_VECTOR (15 downto 0);
        sdat      : out STD_LOGIC);
end component;

component envelope is
Port (
  lrclk      : in  STD_LOGIC;
  grain_length : in  unsigned(15 downto 0); -- samples from 480 to
2400 (10-50ms)
  grain_attack : in  unsigned(15 downto 0); -- expects samples from
16 to 1200 (1-50% of trapezoid)
  grain_release : in  unsigned(15 downto 0); -- expects samples
from 16 to 1200 (1-50% of trapezoid)
  audio_in    : in  STD_LOGIC_VECTOR (15 downto 0);
  enable_grain : in  STD_LOGIC; -- when disabled output is

```



```

000000000000000000
    BYPASS      : in  STD_LOGIC;
    finished    : out STD_LOGIC;
    async_en    : in  STD_LOGIC;
    async       : in  STD_LOGIC;
    RND         : in  STD_LOGIC_VECTOR (11 downto 0);
    audio_out    : out STD_LOGIC_VECTOR (15 downto 0);
    amplitude    : in  STD_LOGIC_VECTOR (15 downto 0));
                  -- "1111111111111111" = full volume

end component;

component density is
Port (
    grain_length : out unsigned(15 downto 0);
                  -- samples from 480 to 2400 (10-50ms)
    enable_grain : out STD_LOGIC; -- when disabled output is
000000000000000000
    grain_attack : out unsigned(15 downto 0); -- expects samples from
16 to 1200 (1-50% of trapezoid)
    grain_release : out unsigned(15 downto 0); -- expects samples
from 16 to 1200 (1-50% of trapezoid)
    RND         : in  STD_LOGIC_VECTOR (11 downto 0);
    async_en    : in  STD_LOGIC;
    async       : in  STD_LOGIC;
    finished    : in  STD_LOGIC;
    create      : in  STD_LOGIC);
end component;

component rand is
Port (
    clk100 : in  STD_LOGIC;
    RND    : out STD_LOGIC_VECTOR (11 downto 0));
end component;

-----
-- Port Maps
-----

signal var_clk      : STD_LOGIC;
signal var_lrclk    : STD_LOGIC;
signal base_clock   : STD_LOGIC;
signal i2s_bclk     : STD_LOGIC;
signal i2s_lrclk    : STD_LOGIC;
signal i2s_sdat     : STD_LOGIC;
signal i2s_powerup  : STD_LOGIC;
signal enable       : STD_LOGIC;
signal ready        : STD_LOGIC;
signal grain_finished : STD_LOGIC;
signal audio_to_pitch : STD_LOGIC_VECTOR(15 downto 0);
signal audio_to_env  : STD_LOGIC_VECTOR(15 downto 0);

```

```

signal audio_out          : STD_LOGIC_VECTOR(15 downto 0);
signal enable_grain       : STD_LOGIC;
signal audio_to_buffer    : STD_LOGIC_VECTOR(15 downto 0);
signal random             : STD_LOGIC_VECTOR(11 downto 0);
signal grain_length       : unsigned(15 downto 0);
signal grain_attack       : unsigned(15 downto 0);
signal grain_release      : unsigned(15 downto 0);

begin
-- xadc instantiation connect the eoc_out to den_in to get continuous conversion
readxadc: xadc_wiz_0 port map (
  daddr_in => "0000011",
  den_in => enable,
  di_in => (others => '0'),
  dwe_in => '0',
  do_out => audio_to_pitch,
  drdy_out => ready,
  dclk_in => clk100,
  reset_in => RESET,
  convst_in => var_clk, --sample on rising edge
  busy_out => open,
  channel_out => open,
  eoc_out => enable,
  eos_out => open,
  alarm_out => open,
  vp_in => VP,
  vn_in => VN
);

pitch_shift: pitchshift port map (
  var_lrclk => var_lrclk,
  lrclk => i2s_lrclk,
  audio_in => audio_to_pitch,
  audio_out => audio_to_env,
  freeze => HOLD,
  LED => LED
);

generate_clock: i2s_clock_generator PORT MAP (
  clk100 => clk100,
  pitch_en => PITCH_EN,
  pitch => PITCH_HILO,
  var_clk => var_clk,
  var_lrclk => var_lrclk,
  base_clock => base_clock,
  i2s_bclk => i2s_bclk,
  i2s_lrclk => i2s_lrclk);

```

```

envelope_grain: envelope PORT MAP (
    lrclk => i2s_lrclk,
    grain_length => grain_length, -- 2400 in binary -- samples from 480 to
2400 (10-50ms)
    grain_attack => grain_attack, -- 1024b from 8 to 1200 (10-50ms)
    grain_release => grain_release, -- 256b from 8 to 1200 (10-50ms)
    BYPASS => BYPASS,
    async_en => ASYNC_EN,
    async => ASYNC_HILO,
    RND => random,
    audio_in => audio_to_env,
    enable_grain => enable_grain, -- when disabled audio output is
000000000000000000
    finished => grain_finished,
    audio_out => audio_out,
    amplitude => "1111000000000000" --"1111111111111111" -- full volume
);

grain_density: density PORT MAP (
    finished => grain_finished,
    grain_length => grain_length, -- samples from 480 to 2400 (10-50ms)
    enable_grain => enable_grain, -- when disabled audio output is
000000000000000000
    grain_attack => grain_attack, -- 1024b from 8 to 1200 (10-50ms)
    grain_release => grain_release, -- 256b from 8 to 1200 (10-50ms)
    RND => random,
    async_en => ASYNC_EN,
    async => ASYNC_HILO,
    create => OFF
);

random12bit: rand port map(
    clk100 => clk100,
    RND => random
);

i_i2s_transmitter: i2s_transmitter port map (
    mclk => base_clock,
    bclk => i2s_bclk,
    lrclk => i2s_lrclk,
    sample_left => audio_out,
    sample_right => audio_out,
    sdat => i2s_sdat);

i_powerup_controller: powerup_controller port map (
    mclk => base_clock,
    powerup => i2s_powerup);

-----

```

```

-- Send it to the PMOD's interface
-----
-- This small portion was from Mike Field:
-- Use a DDR output register to send out the I2S master clock
mclk_ODDR : ODDR generic map(
  DDR_CLK_EDGE => "OPPOSITE_EDGE", -- "OPPOSITE_EDGE" or "SAME_EDGE"
  INIT => '0', -- Initial value for Q port ('1' or '0')
  SRTYPE => "SYNC") -- Reset Type ("ASYNC" or "SYNC")
  port map (
    Q => pmod_i2s_mclk, -- 1-bit DDR output
    C => base_clock, -- 1-bit clock input
    CE => '1', -- 1-bit clock enable input
    D1 => '1', -- 1-bit data input (positive edge)
    D2 => '0', -- 1-bit data input (negative edge)
    R => '0', -- 1-bit reset input
    S => '0' -- 1-bit set input
  );

  pmod_i2s_sd <= i2s_powerup; -- Active low shutdown signal
  pmod_i2s_bclk <= i2s_bclk;
  pmod_i2s_lrclk <= i2s_lrclk;
  pmod_i2s_sdat <= i2s_sdat;
end Behavioral;

```

pitchshift.vhd

```
-----  
-----  
--   pitchshift.vhd  
-----  
-- SILO, The Solid State Granular Synthesizer  
-----  
-- Author: Timothy Opie  
-- Copyright 2018  
-----  
-----  
  
library IEEE;  
use IEEE.STD_LOGIC_1164.ALL;  
use IEEE.NUMERIC_STD.ALL;  
  
entity pitchshift is  
  port (  
    var_lrclk : in STD_LOGIC;  
    lrclk     : in  STD_LOGIC;  
    audio_in  : IN  STD_LOGIC_VECTOR(15 DOWNT0 0);  
    audio_out : OUT STD_LOGIC_VECTOR(15 DOWNT0 0);  
    freeze    : in  STD_LOGIC;  
    LED       : out STD_LOGIC_VECTOR (3 downto 0));  
end pitchshift;  
  
architecture Behavioral of pitchshift is  
  type a_memory is array(0 to 2400) of std_logic_vector(15 downto 0);  
  signal memory : a_memory := (others => (others => '0'));  
  signal in_pointer : unsigned (12 downto 0) := (others => '0');  
  signal out_pointer : unsigned (12 downto 0) := (others => '0');  
  
begin  
  pitch_shift: process (var_lrclk)  
  
    begin  
      if rising_edge(var_lrclk) then  
        if (freeze = '0') then  
          -- pitch shift down or hold  
          memory(to_integer(in_pointer)) <= audio_in;  
          in_pointer <= in_pointer + 1;  
          if (in_pointer >= "100101100000") then --2400b  
            in_pointer <= (others => '0');  
          end if;  
          LED <= audio_in(15 downto 12);  
        end if;  
      end if;  
    end process;  
end;
```

```
audio_outbound: process (lrclk)
begin
    if rising_edge(lrclk) then
        audio_out <= memory(to_integer(out_pointer));
        out_pointer <= out_pointer +1;
        if (out_pointer >= "100101100000") then --2400b
            out_pointer <= (others => '0');
        end if;
    end if;
end process;
end Behavioral;
```

clock_generator.vhd

```
-----  
-----  
--      clock_generator.vhd  
-----  
--      SILO, The Solid State Granular Synthesizer  
-----  
--      Author: Timothy Opie  
--      Copyright 2018  
-----  
-----  
--  
--      Module Name: clock_generator - Behavioral  
--  
--      Description: Generate all clocks from the 100MHz clock:  
--  
--      Base Counter [1] runs at 25Mhz - clock rate for sampling at about 96Khz  
--          sampling at 96KHz and playing back at 48KHz = 1 octave Lower  
--      Base Counter [2] runs at 12.5Mhz - clock rate for sampling at about 48Khz  
--          sampling at 48KHz and playing back at 48KHz = no change  
--      Base Counter [3] runs at 6.25Mhz - clock rate for sampling at about 24Khz  
--          sampling at 24KHz and playing back at 48KHz = 1 octave higher  
--  
--      Counter [4] generates a clock at 48.828Khz about DVD audio quality  
--      Counter [5] generates a clock at 97.657Khz  
--      Counter [3] generates a clock at 24.414Khz  
-----  
--  
library IEEE;  
use IEEE.STD_LOGIC_1164.ALL;  
use IEEE.NUMERIC_STD.ALL;  
Use Ieee.std_logic_unsigned.all;  
  
library UNISIM;  
use UNISIM.VComponents.all;  
  
entity i2s_clock_generator is  
    Port (  
        clk100      : in STD_LOGIC;  
        pitch_en    : in STD_LOGIC;  
        pitch       : in STD_LOGIC;  
        var_clk     : out STD_LOGIC;  
        base_clock  : out STD_LOGIC;  
        i2s_bclk    : out STD_LOGIC;  
        i2s_lrc1k   : out STD_LOGIC;  
        var_lrc1k   : out STD_LOGIC);  
end i2s_clock_generator;
```

```

architecture Behavioral of i2s_clock_generator is
    signal base_counter      : unsigned(5 downto 0) := (others => '0');
    signal counter           : unsigned(5 downto 0) := (others => '0');
    signal this_pitch        : STD_LOGIC_VECTOR(1 downto 0) := (others =>
'0');

begin

clk_process: process(clk100)
    begin
        if rising_edge(clk100) then
            if (base_counter + 1 = "100000") then
                counter <= counter + 1;
            end if;
            base_counter <= base_counter + 1;
        end if;
    end process;

    this_pitch <= not pitch & pitch_en;
    base_clock <= std_logic(base_counter(2));

    i2s_bclk <= std_logic(base_counter(5));

    var_clk <= std_logic(base_counter(3)) when this_pitch = "11" else
std_logic(base_counter(1)) when this_pitch = "01" else
std_logic(base_counter(2));

    i2s_lrcclk <= std_logic(counter(4)); -- about 48KHz

    var_lrcclk <= std_logic(counter(5)) when this_pitch = "11" else -- Pitch
higher
std_logic(counter(3)) when this_pitch = "01" else -- Pitch Lower
std_logic(counter(4)); -- Pitch normal
end Behavioral;

```

envelope.vhd

```
-----
-----
--   envelope.vhd
-----
--   SILO, The Solid State Granular Synthesizer
-----
--   Author: Timothy Opie
--   Copyright 2018
-----
-----
--
--   Description: Trapezoidal Time varying Gain Amplifier
--
--   This part of the code does not check the validity of the data
--   it expects this to be validated already
--
--   _____
--   /      Sustain      \
-- / Attack              Release\
--
--
--   Revision:
--   Revision 0.01 - File Created
--   Additional Comments:
--
-----
--

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity envelope is
  Port (
    lrclk      : in STD_LOGIC;
    grain_length : in unsigned(15 downto 0); -- expects samples from
480 to 2400 (10-50ms)
    grain_attack : in unsigned(15 downto 0); -- expects samples from 8
to 1200 (1-50% of trapezoid)
    grain_release  : in unsigned(15 downto 0); -- expects samples from
8 to 1200 (1-50% of trapezoid)
    audio_in      : in STD_LOGIC_VECTOR (15 downto 0);
    enable_grain  : in STD_LOGIC; -- when disabled output is
0000000000000000
    BYPASS       : in STD_LOGIC;
    RND          : in STD_LOGIC_VECTOR (11 downto 0);
    async_en     : in STD_LOGIC; -- allow asynchronicity
```

```

        async      : in  STD_LOGIC; -- basic asynchronicity Levels
        audio_out   : out STD_LOGIC_VECTOR (15 downto 0);
        amplitude   : in  STD_LOGIC_VECTOR (15 downto 0); --
        "111111111111111" = full volume
        finished    : out STD_LOGIC
    );
end envelope;

architecture Behavioral of envelope is
    signal counter      : unsigned (15 downto 0) := (others => '0');
    signal iot          : unsigned(15 downto 0) := "0000000000001000";
    signal Asynccount   : unsigned (15 downto 0) := (others => '0');
    signal finished_grain : STD_LOGIC := '0';

begin

    process(lrclk)
        variable temp_audio: unsigned(47 downto 0);

    begin
        if (BYPASS = '1') then
            audio_out <= audio_in;
        else
            if rising_edge(lrclk) then
                audio_out <= (others => '0');

                if finished_grain = '1' then
                    Asynccount <= Asynccount +1;
                    if (Asynccount > iot) then
                        Asynccount <= (others => '0');
                        finished_grain <= '0';
                    end if;
                end if;

                if ((enable_grain = '1') and (finished_grain = '0')) then
                    finished_grain <='0';
                    temp_audio := "0000000000000000" & unsigned(audio_in) *
unsigned(amplitude) / "111111111111111";

                    if (counter < grain_attack) then
                        temp_audio := unsigned(audio_in) * counter / grain_attack *
unsigned(amplitude) / "111111111111111";
                    end if;

                    if (counter > (grain_length - grain_release)) then
                        temp_audio := unsigned(audio_in) * (grain_length - counter)
/ grain_release * unsigned(amplitude) / "111111111111111";
                    end if;

                    audio_out <= std_logic_vector(temp_audio(15 downto 0));
                end if;
            end if;
        end process;
    end architecture;

```

```
counter <= counter + 1;

if (counter > grain_length) then
    counter <= (others => '0');
    finished_grain <= '1';
    iot <= "0000000000001000";
    if (async_en = '1') then
        if (async = '1') then
            iot <= "000" & unsigned(RND) & '0';
        else
            iot <= "00000" & unsigned(RND(9 downto 0)) & '0';
        end if;
    else
        iot <= "0000000000000100";
    end if;
end if;
end if;
end if;
end process;
finished <= finished_grain;

end Behavioral;
```

density.vhd

```
-----  
-----  
--      density.vhd  
-----  
-- SILO, The Solid State Granular Synthesizer  
-----  
-- Author: Timothy Opie  
-- Copyright 2018  
-----  
-----  
  
library IEEE;  
use IEEE.STD_LOGIC_1164.ALL;  
use IEEE.NUMERIC_STD.ALL;  
use IEEE.STD_LOGIC_UNSIGNED.ALL;  
  
entity density is  
  Port (  
    grain_length : out unsigned(15 downto 0); -- samples from 480 to  
2400 (10-50ms)  
    enable_grain : out STD_LOGIC; -- when disabled output is  
0000000000000000  
    grain_attack : out unsigned(15 downto 0); -- expects samples from  
16 to 1200 (1-50% of trapezoid)  
    grain_release  : out unsigned(15 downto 0); -- expects samples  
from 16 to 1200 (1-50% of trapezoid)  
    finished      : in STD_LOGIC;  
    RND           : in STD_LOGIC_VECTOR (11 downto 0);  
    async_en     : in STD_LOGIC; -- allow asynchronicity  
    async        : in STD_LOGIC; -- basic asynchronicity levels  
    create       : in STD_LOGIC  
  );  
end density;  
  
architecture Behavioral of density is  
  signal variance : unsigned (11 downto 0);  
begin  
  
control: process(finished)  
begin  
  variance <= unsigned(RND);  
  if (async_en = '0') then  
    variance <= "001111111111";  
  else  
    if (async = '0') then  
      variance <= unsigned("000010000000" + RND(8 downto 0));  
    end if;  
  
end process;
```

```
    end if;
    enable_grain <= create;
    grain_length <= "000000" & (variance(9 downto 0)) + "10000000000";
--"0000100101100000";
    grain_attack <= "00000000" & (variance(7 downto 0)) + 32;
--"0000100101100000";
    grain_release <= "00000000" & (variance(7 downto 0)) + 32;
--"0000100101100000";
end process;
end Behavioral;
```

rnd.vhd

```
-----  
-----  
--      rnd.vhd  
-----  
-- SILO, The Solid State Granular Synthesizer  
-----  
-- Author: Timothy Opie  
-- Copyright 2018  
-----  
-----  
  
library IEEE;  
use IEEE.STD_LOGIC_1164.ALL;  
  
entity rand is  
    Port (  
        clk100 : in  STD_LOGIC;  
        RND : out  STD_LOGIC_VECTOR (11 downto 0)  
    );  
end rand;  
  
architecture Behavioral of rand is  
    signal x0, x1, x4, x6, x7, x8, x11, x12, x13, x14, x16, x17, x30, x31,  
x34: STD_LOGIC := '1';  
    signal x2, x3, x5, x9, x10, x15, x18, x19, x20, x21, x22, x23, x24, x25,  
x26, x27, x28, x29, x32, x33 : STD_LOGIC := '0';  
begin  
  
seq: process(clk100)  
begin  
    if rising_edge(clk100) then  
        x0 <= x34;  
        x1 <= x0;  
        x2 <= x1 xor x34;  
        x3 <= x2;  
        x4 <= x3 xor x34;  
        x5 <= x4;  
        x6 <= x5;  
        x7 <= x6 xor x34;  
        x8 <= x7 xor x34;  
        x9 <= x8 xor x18;  
        x10 <= x9;  
        x11 <= x10;  
        x12 <= x11;  
        x13 <= x12;  
        x14 <= x13 xor x34;  
        x15 <= x14;
```

```

x16 <= x15;
x17 <= x16 xor x34;
x18 <= x17;
x19 <= x18;
x20 <= x19;
x21 <= x20 xor x34;
x22 <= x21 xor x34;
x23 <= x22;
x24 <= x23 xor x34;
x25 <= x24;
x26 <= x25;
x27 <= x26;
x28 <= x27;
x29 <= x28 xor x34;
x30 <= x29 xor x34;
x31 <= x30;
x32 <= x31;
x33 <= x32 xor x34;
x34 <= x33;
end if;
end process seq;

-- Polynomial: f(x) = x35 + x33 + + x30 + x29 + x24 + x22 + x21 + x17 + x14 + x9
+ x8 + x7 + x4 + x2 +1
-- RND = f(x) [x3 - x14]

RND <= (x3 & x4 & x5 & x6 & x7& x8 & x9 & x10 & x11 & x12 & x13 & x14);

end Behavioral;

```

constraints.xdc

```
## This file is a general .xdc for the Arty S7-50 Rev. B

set_property -dict {PACKAGE_PIN R2 IOSTANDARD SSTL135} [get_ports clk100]
create_clock -period 10.000 -name sys_clk_pin -waveform {0.000 5.000} -add
[get_ports clk100]

# Switches
set_property -dict { PACKAGE_PIN H14   IOSTANDARD LVCMOS33 } [get_ports { OFF
}]; #IO_L20N_T3_A19_15 Sch=sw[0] SW[0]
set_property -dict { PACKAGE_PIN H18   IOSTANDARD LVCMOS33 } [get_ports { BYPASS
}]; #IO_L21P_T3_DQS_15 Sch=sw[1] SW[1]
set_property -dict { PACKAGE_PIN G18   IOSTANDARD LVCMOS33 } [get_ports {
ASYNC_EN }]; #IO_L21N_T3_DQS_A18_15 Sch=sw[2] SW[2]
set_property -dict { PACKAGE_PIN M5    IOSTANDARD SSTL135 } [get_ports {
PITCH_EN}]; #IO_L6N_T0_VREF_34 Sch=sw[3] SW[3]

## LEDs
set_property -dict {PACKAGE_PIN E18 IOSTANDARD LVCMOS33} [get_ports {LED[0]}]
set_property -dict {PACKAGE_PIN F13 IOSTANDARD LVCMOS33} [get_ports {LED[1]}]
set_property -dict {PACKAGE_PIN E13 IOSTANDARD LVCMOS33} [get_ports {LED[2]}]
set_property -dict {PACKAGE_PIN H15 IOSTANDARD LVCMOS33} [get_ports {LED[3]}]

## Buttons
set_property -dict { PACKAGE_PIN G15   IOSTANDARD LVCMOS33 } [get_ports { RESET
}]; #IO_L18N_T2_A23_15 Sch=btn[0]
set_property -dict { PACKAGE_PIN K16   IOSTANDARD LVCMOS33 } [get_ports { HOLD
}]; #IO_L19P_T3_A22_15 Sch=btn[1]
set_property -dict { PACKAGE_PIN J16   IOSTANDARD LVCMOS33 } [get_ports {
ASYNC_HILO }]; #IO_L19N_T3_A21_VREF_15 Sch=btn[2]
set_property -dict { PACKAGE_PIN H13   IOSTANDARD LVCMOS33 } [get_ports {
PITCH_HILO }]; #IO_L20P_T3_A20_15 Sch=btn[3]

## PMOD Header JB
#1
set_property -dict {PACKAGE_PIN P17 IOSTANDARD LVCMOS33} [get_ports
pmod_i2s_lrclk]
#2
set_property -dict {PACKAGE_PIN P18 IOSTANDARD LVCMOS33} [get_ports
pmod_i2s_sdat]
#3
set_property -dict { PACKAGE_PIN R18   IOSTANDARD LVCMOS33 } [get_ports { jb[2]
}]; #IO_L10P_T1_D14_14 Sch=jb_p[2]
#4
set_property -dict {PACKAGE_PIN T18 IOSTANDARD LVCMOS33} [get_ports
pmod_i2s_bclk]
```

```
#7
#set_property -dict { PACKAGE_PIN P14    IOSTANDARD LVCMOS33 } [get_ports { jb[4]
}]; #IO_L11P_T1_SRCC_14 Sch=jb_p[3]
#8
#set_property -dict { PACKAGE_PIN P15    IOSTANDARD LVCMOS33 } [get_ports { jb[5]
}]; #IO_L11N_T1_SRCC_14 Sch=jb_n[3]
#9
set_property -dict {PACKAGE_PIN N15 IOSTANDARD LVCMOS33} [get_ports
pmod_i2s_mclk]
#10
set_property -dict {PACKAGE_PIN P16 IOSTANDARD LVCMOS33} [get_ports pmod_i2s_sd]

## Dedicated Analog Inputs
set_property -dict {PACKAGE_PIN J10} [get_ports VP]
set_property -dict {PACKAGE_PIN K9} [get_ports VN]

## Configuration options, can be used for all designs
set_property CONFIG_VOLTAGE 3.3 [current_design]
set_property CFGBVS VCC0 [current_design]
set_property BITSTREAM.CONFIG.SPI_BUSWIDTH 4 [current_design]

## SW3 is assigned to a pin M5 in the 1.35v bank. This pin can also be used as
## the VREF for BANK 34. To ensure that SW3 does not define the reference
voltage
## and to be able to use this pin as an ordinary I/O the following property must
## be set to enable an internal VREF for BANK 34. Since a 1.35v supply is being
## used the internal reference is set to half that value (i.e. 0.675v). Note
that
## this property must be set even if SW3 is not used in the design.
set_property INTERNAL_VREF 0.675 [get_iobanks 34]

set_property CONFIG_MODE SPIx4 [current_design]
set_property BITSTREAM.CONFIG.CONFIGRATE 50 [current_design]
```

Xilinx Vivado Utilisation Report

Copyright 1986-2018 Xilinx, Inc. All Rights Reserved.

| Tool Version : Vivado v.2018.2 (win64) Build 2258646 Thu Jun 14 20:03:12 MDT
2018
| Date : Sun Oct 28 18:07:32 2018
| Host : evo running 64-bit major release (build 9200)
| Command : report_utilization -file SILO_utilization_synth.rpt -pb
SILO_utilization_synth.pb
| Design : SILO
| Device : 7s50csga324-1
Design State : Synthesized

Utilization Design Information

Table of Contents

- 1. Slice Logic
1.1 Summary of Registers by Type
2. Memory
3. DSP
4. IO and GT Specific
5. Clocking
6. Specific Feature
7. Primitives
8. Black Boxes
9. Instantiated Netlists

1. Slice Logic

Site Type	Used	Fixed	Available	Util%
Slice LUTs*	2378	0	32600	7.29
LUT as Logic	2376	0	32600	7.29
LUT as Memory	2	0	9600	0.02
LUT as Distributed RAM	0	0		
LUT as Shift Register	2	0		
Slice Registers	221	0	65200	0.34
Register as Flip Flop	205	0	65200	0.31
Register as Latch	16	0	65200	0.02
F7 Muxes	0	0	16300	0.00
F8 Muxes	0	0	8150	0.00

* Warning! The Final LUT count, after physical optimizations and full implementation, is typically lower. Run opt_design after synthesis, if not already completed, for a more realistic count.

1.1 Summary of Registers by Type

```

-----
+-----+-----+-----+-----+
| Total | Clock Enable | Synchronous | Asynchronous |
+-----+-----+-----+-----+
| 0     | -           | -           | -           |
| 0     | -           | -           | Set        |
| 0     | -           | -           | Reset      |
| 0     | -           | Set        | -          |
| 0     | -           | Reset      | -          |
| 0     | Yes        | -           | -          |
| 16    | Yes        | -           | Set        |
| 32    | Yes        | -           | Reset      |
| 0     | Yes        | Set        | -          |
| 173   | Yes        | Reset      | -          |
+-----+-----+-----+-----+

```

2. Memory

```

-----
+-----+-----+-----+-----+-----+
| Site Type | Used | Fixed | Available | Util% |
+-----+-----+-----+-----+-----+
| Block RAM Tile | 2 | 0 | 75 | 2.67 |
| RAMB36/FIFO* | 2 | 0 | 75 | 2.67 |
| RAMB36E1 only | 2 | | | |
| RAMB18 | 0 | 0 | 150 | 0.00 |
+-----+-----+-----+-----+-----+

```

* Note: Each Block RAM Tile only has one FIFO logic available and therefore can accommodate only one FIFO36E1 or one FIFO18E1. However, if a FIFO18E1 occupies a Block RAM Tile, that tile can still accommodate a RAMB18E1

3. DSP

```

-----
+-----+-----+-----+-----+-----+
| Site Type | Used | Fixed | Available | Util% |
+-----+-----+-----+-----+-----+
| DSPs | 7 | 0 | 120 | 5.83 |
| DSP48E1 only | 7 | | | |
+-----+-----+-----+-----+-----+

```

4. IO and GT Specific

Site Type	Used	Fixed	Available	Util%
Bonded IOB	20	0	210	9.52
Bonded IPADs	0	0	2	0.00
PHY_CONTROL	0	0	5	0.00
PHASER_REF	0	0	5	0.00
OUT_FIFO	0	0	20	0.00
IN_FIFO	0	0	20	0.00
IDELAYCTRL	0	0	5	0.00
IBUFDS	0	0	202	0.00
PHASER_OUT/PHASER_OUT_PHY	0	0	20	0.00
PHASER_IN/PHASER_IN_PHY	0	0	20	0.00
IDELAYE2/IDELAYE2_FINEDELAY	0	0	250	0.00
ILOGIC	0	0	210	0.00
OLOGIC	1	0	210	0.48
ODDR	1			

5. Clocking

Site Type	Used	Fixed	Available	Util%
BUFGCTRL	3	0	32	9.38
BUFIO	0	0	20	0.00
MMCME2_ADV	0	0	5	0.00
PLLE2_ADV	0	0	5	0.00
BUFMRCE	0	0	10	0.00
BUFHCE	0	0	72	0.00
BUFR	0	0	20	0.00

6. Specific Feature

Site Type	Used	Fixed	Available	Util%
BSCANE2	0	0	4	0.00
CAPTUREE2	0	0	1	0.00
DNA_PORT	0	0	1	0.00
EFUSE_USR	0	0	1	0.00
FRAME_ECCE2	0	0	1	0.00
ICAPE2	0	0	2	0.00

STARTUPE2	0	0	1	0.00
XADC	0	0	1	0.00

7. Primitives

Ref Name	Used	Functional Category
LUT2	1157	LUT
LUT4	663	LUT
CARRY4	616	CarryLogic
LUT3	368	LUT
LUT5	252	LUT
FDRE	173	Flop & Latch
LUT6	166	LUT
LUT1	94	LUT
LDCE	16	Flop & Latch
FDPE	16	Flop & Latch
FDCE	16	Flop & Latch
IBUF	11	IO
OBUF	9	IO
DSP48E1	7	Block Arithmetic
BUFG	3	Clock
SRL16E	2	Distributed Memory
RAMB36E1	2	Block Memory
ODDR	1	IO

8. Black Boxes

Ref Name	Used
xadc_wiz_0	1

9. Instantiated Netlists

Ref Name	Used
----------	------

Xilinx Vivado Synthesis Report

```
#-----
# Vivado v2018.2 (64-bit)
# SW Build 2258646 on Thu Jun 14 20:03:12 MDT 2018
# IP Build 2256618 on Thu Jun 14 22:10:49 MDT 2018
# Start of session at: Sun Oct 28 18:06:18 2018
# Process ID: 4496
# Current directory: Q:/Xilinx/work/grain_silo/grain_silo.runs/synth_1
# Command line: vivado.exe -log SILO.vds -product Vivado -mode batch -messageDb vivado.pb
-notrace -source SILO.tcl
# Log file: Q:/Xilinx/work/grain_silo/grain_silo.runs/synth_1/SILO.vds
# Journal file: Q:/Xilinx/work/grain_silo/grain_silo.runs/synth_1\vivado.jou
#-----
source SILO.tcl -notrace
Command: synth_design -top SILO -part xc7s50csga324-1
Starting synth_design
Attempting to get a license for feature 'Synthesis' and/or device 'xc7s50'
INFO: [Common 17-349] Got license for feature 'Synthesis' and/or device 'xc7s50'
INFO: Launching helper process for spawning children vivado processes
INFO: Helper process launched with PID 10356
-----
Starting RTL Elaboration : Time (s): cpu = 00:00:04 ; elapsed = 00:00:05 . Memory (MB):
peak = 383.973 ; gain = 100.621
-----
INFO: [Synth 8-638] synthesizing module 'SILO'
[Q:/Xilinx/work/grain_silo/grain_silo.srcs/sources_1/imports/new/silo.vhd:74]
INFO: [Synth 8-3491] module 'xadc_wiz_0' declared at
'Q:/Xilinx/work/grain_silo/grain_silo.runs/synth_1/.Xil/Vivado-4496-evo/realtime/xadc_wiz_0
_stub.vhdl:5' bound to instance 'readxadc' of component 'xadc_wiz_0'
[Q:/Xilinx/work/grain_silo/grain_silo.srcs/sources_1/imports/new/silo.vhd:207]
INFO: [Synth 8-638] synthesizing module 'xadc_wiz_0'
[Q:/Xilinx/work/grain_silo/grain_silo.runs/synth_1/.Xil/Vivado-4496-evo/realtime/xadc_wiz_0
_stub.vhdl:27]
INFO: [Synth 8-3491] module 'pitchshift' declared at
'Q:/Xilinx/work/grain_silo/grain_silo.srcs/sources_1/imports/new/pitchshift.vhd:27' bound
to instance 'pitch_shift' of component 'pitchshift'
[Q:/Xilinx/work/grain_silo/grain_silo.srcs/sources_1/imports/new/silo.vhd:226]
INFO: [Synth 8-638] synthesizing module 'pitchshift'
[Q:/Xilinx/work/grain_silo/grain_silo.srcs/sources_1/imports/new/pitchshift.vhd:38]
INFO: [Synth 8-256] done synthesizing module 'pitchshift' (1#1)
[Q:/Xilinx/work/grain_silo/grain_silo.srcs/sources_1/imports/new/pitchshift.vhd:38]
INFO: [Synth 8-3491] module 'i2s_clock_generator' declared at
'Q:/Xilinx/work/grain_silo/grain_silo.srcs/sources_1/imports/new/i2s_clock_generator.vhd:47
' bound to instance 'generate_clock' of component 'i2s_clock_generator'
[Q:/Xilinx/work/grain_silo/grain_silo.srcs/sources_1/imports/new/silo.vhd:239]
INFO: [Synth 8-638] synthesizing module 'i2s_clock_generator'
[Q:/Xilinx/work/grain_silo/grain_silo.srcs/sources_1/imports/new/i2s_clock_generator.vhd:59
]
INFO: [Synth 8-256] done synthesizing module 'i2s_clock_generator' (2#1)
[Q:/Xilinx/work/grain_silo/grain_silo.srcs/sources_1/imports/new/i2s_clock_generator.vhd:59
]
```

INFO: [Synth 8-3491] module 'envelope' declared at
'Q:/Xilinx/work/grain_silo/grain_silo.srcs/sources_1/imports/new/envelope.vhd:32' bound to
instance 'envelope_grain' of component 'envelope'
[Q:/Xilinx/work/grain_silo/grain_silo.srcs/sources_1/imports/new/silo.vhd:254]
INFO: [Synth 8-638] synthesizing module 'envelope'
[Q:/Xilinx/work/grain_silo/grain_silo.srcs/sources_1/imports/new/envelope.vhd:50]
WARNING: [Synth 8-614] signal 'BYPASS' is read in the process but is not in the sensitivity
list [Q:/Xilinx/work/grain_silo/grain_silo.srcs/sources_1/imports/new/envelope.vhd:58]
WARNING: [Synth 8-614] signal 'audio_in' is read in the process but is not in the
sensitivity list
[Q:/Xilinx/work/grain_silo/grain_silo.srcs/sources_1/imports/new/envelope.vhd:58]
WARNING: [Synth 8-6014] Unused sequential element temp_audio_reg was removed.
[Q:/Xilinx/work/grain_silo/grain_silo.srcs/sources_1/imports/new/envelope.vhd:78]
INFO: [Synth 8-256] done synthesizing module 'envelope' (3#1)
[Q:/Xilinx/work/grain_silo/grain_silo.srcs/sources_1/imports/new/envelope.vhd:50]
INFO: [Synth 8-3491] module 'density' declared at
'Q:/Xilinx/work/grain_silo/grain_silo.srcs/sources_1/imports/new/density.vhd:28' bound to
instance 'grain_density' of component 'density'
[Q:/Xilinx/work/grain_silo/grain_silo.srcs/sources_1/imports/new/silo.vhd:275]
INFO: [Synth 8-638] synthesizing module 'density'
[Q:/Xilinx/work/grain_silo/grain_silo.srcs/sources_1/imports/new/density.vhd:43]
WARNING: [Synth 8-614] signal 'RND' is read in the process but is not in the sensitivity
list [Q:/Xilinx/work/grain_silo/grain_silo.srcs/sources_1/imports/new/density.vhd:47]
WARNING: [Synth 8-614] signal 'async_en' is read in the process but is not in the
sensitivity list
[Q:/Xilinx/work/grain_silo/grain_silo.srcs/sources_1/imports/new/density.vhd:47]
WARNING: [Synth 8-614] signal 'async' is read in the process but is not in the sensitivity
list [Q:/Xilinx/work/grain_silo/grain_silo.srcs/sources_1/imports/new/density.vhd:47]
WARNING: [Synth 8-614] signal 'create' is read in the process but is not in the sensitivity
list [Q:/Xilinx/work/grain_silo/grain_silo.srcs/sources_1/imports/new/density.vhd:47]
WARNING: [Synth 8-614] signal 'variance' is read in the process but is not in the
sensitivity list
[Q:/Xilinx/work/grain_silo/grain_silo.srcs/sources_1/imports/new/density.vhd:47]
INFO: [Synth 8-256] done synthesizing module 'density' (4#1)
[Q:/Xilinx/work/grain_silo/grain_silo.srcs/sources_1/imports/new/density.vhd:43]
INFO: [Synth 8-3491] module 'rand' declared at
'Q:/Xilinx/work/grain_silo/grain_silo.srcs/sources_1/imports/new/rnd.vhd:25' bound to
instance 'random12bit' of component 'rand'
[Q:/Xilinx/work/grain_silo/grain_silo.srcs/sources_1/imports/new/silo.vhd:287]
INFO: [Synth 8-638] synthesizing module 'rand'
[Q:/Xilinx/work/grain_silo/grain_silo.srcs/sources_1/imports/new/rnd.vhd:32]
INFO: [Synth 8-256] done synthesizing module 'rand' (5#1)
[Q:/Xilinx/work/grain_silo/grain_silo.srcs/sources_1/imports/new/rnd.vhd:32]
INFO: [Synth 8-3491] module 'i2s_transmitter' declared at
'Q:/Xilinx/work/grain_silo/grain_silo.srcs/sources_1/imports/new/i2s_transmitter.vhd:33'
bound to instance 'i_i2s_transmitter' of component 'i2s_transmitter'
[Q:/Xilinx/work/grain_silo/grain_silo.srcs/sources_1/imports/new/silo.vhd:295]
INFO: [Synth 8-638] synthesizing module 'i2s_transmitter'
[Q:/Xilinx/work/grain_silo/grain_silo.srcs/sources_1/imports/new/i2s_transmitter.vhd:42]
INFO: [Synth 8-256] done synthesizing module 'i2s_transmitter' (6#1)
[Q:/Xilinx/work/grain_silo/grain_silo.srcs/sources_1/imports/new/i2s_transmitter.vhd:42]
INFO: [Synth 8-3491] module 'powerup_controller' declared at
'Q:/Xilinx/work/grain_silo/grain_silo.srcs/sources_1/imports/new/powerup_controller.vhd:31'
bound to instance 'i_powerup_controller' of component 'powerup_controller'

```

[Q:/Xilinx/work/grain_silo/grain_silo.srcs/sources_1/imports/new/silo.vhd:303]
INFO: [Synth 8-638] synthesizing module 'powerup_controller'
[Q:/Xilinx/work/grain_silo/grain_silo.srcs/sources_1/imports/new/powerup_controller.vhd:36]
INFO: [Synth 8-256] done synthesizing module 'powerup_controller' (7#1)
[Q:/Xilinx/work/grain_silo/grain_silo.srcs/sources_1/imports/new/powerup_controller.vhd:36]
  Parameter DDR_CLK_EDGE bound to: OPPOSITE_EDGE - type: string
  Parameter INIT bound to: 1'b0
  Parameter IS_C_INVERTED bound to: 1'b0
  Parameter IS_D1_INVERTED bound to: 1'b0
  Parameter IS_D2_INVERTED bound to: 1'b0
  Parameter SRTYPE bound to: SYNC - type: string
INFO: [Synth 8-113] binding component instance 'mclk_ODDR' to cell 'ODDR'
[Q:/Xilinx/work/grain_silo/grain_silo.srcs/sources_1/imports/new/silo.vhd:313]
INFO: [Synth 8-256] done synthesizing module 'SILO' (8#1)
[Q:/Xilinx/work/grain_silo/grain_silo.srcs/sources_1/imports/new/silo.vhd:74]
WARNING: [Synth 8-3331] design density has unconnected port finished

```

```

-----
Finished RTL Elaboration : Time (s): cpu = 00:00:06 ; elapsed = 00:00:06 . Memory (MB):
peak = 439.270 ; gain = 155.918
-----

```

Report Check Netlist:

```

+-----+-----+-----+-----+-----+-----+
|      |Item                |Errors |Warnings |Status |Description      |
+-----+-----+-----+-----+-----+-----+
|1     |multi_driven_nets  |      0|      0|Passed |Multi driven nets |
+-----+-----+-----+-----+-----+-----+

```

Start Handling Custom Attributes

```

-----
Finished Handling Custom Attributes : Time (s): cpu = 00:00:06 ; elapsed = 00:00:07 .
Memory (MB): peak = 439.270 ; gain = 155.918
-----

```

```

-----
Finished RTL Optimization Phase 1 : Time (s): cpu = 00:00:06 ; elapsed = 00:00:07 . Memory
(MB): peak = 439.270 ; gain = 155.918
-----

```

```

INFO: [Netlist 29-17] Analyzing 1 Unisim elements for replacement
INFO: [Netlist 29-28] Unisim Transformation completed in 0 CPU seconds
INFO: [Device 21-403] Loading part xc7s50csga324-1
INFO: [Project 1-570] Preparing netlist for logic optimization

```

Processing XDC Constraints

Initializing timing engine

Parsing XDC File

```

[q:/Xilinx/work/grain_silo/grain_silo.srcs/sources_1/ip/xadc_wiz_0/xadc_wiz_0/xadc_wiz_0_in
_context.xdc] for cell 'readxadc'

```

Finished Parsing XDC File

```

[q:/Xilinx/work/grain_silo/grain_silo.srcs/sources_1/ip/xadc_wiz_0/xadc_wiz_0/xadc_wiz_0_in
_context.xdc] for cell 'readxadc'

```

Parsing XDC File

```

[q:/Xilinx/work/grain_silo/grain_silo.srcs/constrs_1/imports/constraints/Arty-S7-50-Master.
xdc]

```

Finished Parsing XDC File

[Q:/Xilinx/work/grain_silo/grain_silo.srcs/constrs_1/imports/constraints/Arty-S7-50-Master.xdc]

INFO: [Project 1-236] Implementation specific constraints were found while reading constraint file

[Q:/Xilinx/work/grain_silo/grain_silo.srcs/constrs_1/imports/constraints/Arty-S7-50-Master.xdc]. These constraints will be ignored for synthesis but will be used in implementation. Impacted constraints are listed in the file [.Xil/SILO_propImpl.xdc].

Resolution: To avoid this warning, move constraints listed in [.Xil/SILO_propImpl.xdc] to another XDC file and exclude this new file from synthesis with the used_in_synthesis property (File Properties dialog in GUI) and re-run elaboration/synthesis.

Completed Processing XDC Constraints

INFO: [Project 1-111] Unisim Transformation Summary:

No Unisim elements were transformed.

Constraint Validation Runtime : Time (s): cpu = 00:00:00 ; elapsed = 00:00:00.013 . Memory (MB): peak = 762.789 ; gain = 0.000

Finished Constraint Validation : Time (s): cpu = 00:00:17 ; elapsed = 00:00:21 . Memory (MB): peak = 762.789 ; gain = 479.438

Start Loading Part and Timing Information

Loading part: xc7s50csga324-1

Finished Loading Part and Timing Information : Time (s): cpu = 00:00:17 ; elapsed = 00:00:21 . Memory (MB): peak = 762.789 ; gain = 479.438

Start Applying 'set_property' XDC Constraints

Applied set_property DONT_TOUCH = true for readxadc. (constraint file auto generated constraint, line).

Finished applying 'set_property' XDC Constraints : Time (s): cpu = 00:00:17 ; elapsed = 00:00:21 . Memory (MB): peak = 762.789 ; gain = 479.438

WARNING: [Synth 8-6014] Unused sequential element out_pointer_reg_rep was removed. [Q:/Xilinx/work/grain_silo/grain_silo.srcs/sources_1/imports/new/pitchshift.vhd:65]

Finished RTL Optimization Phase 2 : Time (s): cpu = 00:00:18 ; elapsed = 00:00:22 . Memory (MB): peak = 762.789 ; gain = 479.438

Report RTL Partitions:

RTL Partition	Replication	Instances

Start RTL Component Statistics

Detailed RTL Component Info :

+---Adders :
 2 Input 16 Bit Adders := 1
 3 Input 16 Bit Adders := 2
 2 Input 13 Bit Adders := 1
 2 Input 10 Bit Adders := 2
 2 Input 9 Bit Adders := 1
 2 Input 6 Bit Adders := 3
+---XORs :
 2 Input 1 Bit XORs := 13
+---Registers :
 32 Bit Registers := 1
 16 Bit Registers := 4
 10 Bit Registers := 1
 6 Bit Registers := 2
 4 Bit Registers := 1
 1 Bit Registers := 39
+---Multipliers :
 16x32 Multipliers := 2
+---RAMs :
 37K Bit RAMs := 1
+---Muxes :
 2 Input 32 Bit Muxes := 1
 2 Input 16 Bit Muxes := 7
 2 Input 10 Bit Muxes := 2
 2 Input 1 Bit Muxes := 6

Finished RTL Component Statistics

Start RTL Hierarchical Component Statistics

Hierarchical RTL Component report

Module pitchshift

Detailed RTL Component Info :

+---Adders :
 2 Input 13 Bit Adders := 1
+---Registers :
 16 Bit Registers := 1
 4 Bit Registers := 1
+---RAMs :
 37K Bit RAMs := 1

Module i2s_clock_generator

Detailed RTL Component Info :

+---Adders :
 2 Input 6 Bit Adders := 3
+---Registers :
 6 Bit Registers := 2
+---Muxes :
 2 Input 1 Bit Muxes := 4

Module envelope

Detailed RTL Component Info :

+---Adders :
 2 Input 16 Bit Adders := 1

```

      3 Input      16 Bit      Adders := 2
+---Registers :
      16 Bit Registers := 3
      1 Bit  Registers := 1
+---Multipliers :
      16x32 Multipliers := 2
+---Muxes :
      2 Input      16 Bit      Muxes := 7
      2 Input      1 Bit       Muxes := 1
Module density
Detailed RTL Component Info :
+---Adders :
      2 Input      10 Bit      Adders := 1
      2 Input      9 Bit       Adders := 1
+---Muxes :
      2 Input      10 Bit      Muxes := 2
Module rand
Detailed RTL Component Info :
+---XORs :
      2 Input      1 Bit       XORs := 13
+---Registers :
      1 Bit  Registers := 35
Module i2s_transmitter
Detailed RTL Component Info :
+---Registers :
      32 Bit Registers := 1
      1 Bit  Registers := 3
+---Muxes :
      2 Input      32 Bit      Muxes := 1
      2 Input      1 Bit       Muxes := 1
Module powerup_controller
Detailed RTL Component Info :
+---Adders :
      2 Input      10 Bit      Adders := 1
+---Registers :
      10 Bit Registers := 1

```

Finished RTL Hierarchical Component Statistics

Start Part Resource Summary

Part Resources:

```

DSPs: 120 (col length:60)
BRAMs: 150 (col length: RAMB18 60 RAMB36 30)

```

Finished Part Resource Summary

Start Cross Boundary and Area Optimization

Warning: Parallel synthesis criteria is not met

```

INFO: [Synth 8-4471] merging register 'counter_reg[15:0]' into 'counter_reg[15:0]'
[Q:/Xilinx/work/grain_silo/grain_silo.srcs/sources_1/imports/new/envelope.vhd:80]

```

WARNING: [Synth 8-6014] Unused sequential element counter_reg was removed.
[Q:/Xilinx/work/grain_silo/grain_silo.srcs/sources_1/imports/new/envelope.vhd:80]
INFO: [Synth 8-5845] Not enough pipeline registers after wide multiplier. Recommended levels of pipeline registers is 2
[Q:/Xilinx/work/grain_silo/grain_silo.srcs/sources_1/imports/new/envelope.vhd:81]
INFO: [Synth 8-5845] Not enough pipeline registers after wide multiplier. Recommended levels of pipeline registers is 2
[Q:/Xilinx/work/grain_silo/grain_silo.srcs/sources_1/imports/new/envelope.vhd:85]
DSP Report: Generating DSP temp_audio1, operation Mode is: A*B.
DSP Report: operator temp_audio1 is absorbed into DSP temp_audio1.
DSP Report: Generating DSP temp_audio3, operation Mode is: A*B2.
DSP Report: register counter_reg is absorbed into DSP temp_audio3.
DSP Report: operator temp_audio3 is absorbed into DSP temp_audio3.
DSP Report: Generating DSP temp_audio1, operation Mode is: A*B.
DSP Report: operator temp_audio1 is absorbed into DSP temp_audio1.
DSP Report: operator temp_audio1 is absorbed into DSP temp_audio1.
DSP Report: Generating DSP temp_audio1, operation Mode is: (PCIN>>17)+A*B.
DSP Report: operator temp_audio1 is absorbed into DSP temp_audio1.
DSP Report: operator temp_audio1 is absorbed into DSP temp_audio1.
DSP Report: Generating DSP temp_audio3, operation Mode is: A*B.
DSP Report: operator temp_audio3 is absorbed into DSP temp_audio3.
DSP Report: Generating DSP temp_audio1, operation Mode is: A*B.
DSP Report: operator temp_audio1 is absorbed into DSP temp_audio1.
DSP Report: operator temp_audio1 is absorbed into DSP temp_audio1.
DSP Report: Generating DSP temp_audio1, operation Mode is: (PCIN>>17)+A*B.
DSP Report: operator temp_audio1 is absorbed into DSP temp_audio1.
DSP Report: operator temp_audio1 is absorbed into DSP temp_audio1.
INFO: [Synth 8-4652] Swapped enable and write-enable on 2 RAM instances of RAM pitch_shift/memory_reg to conserve power
INFO: [Synth 8-3886] merging instance 'envelope_grain/iot_reg[0]' (FDE) to 'envelope_grain/iot_reg[15]'
INFO: [Synth 8-3886] merging instance 'envelope_grain/iot_reg[13]' (FDE) to 'envelope_grain/iot_reg[15]'
INFO: [Synth 8-3886] merging instance 'envelope_grain/iot_reg[14]' (FDE) to 'envelope_grain/iot_reg[15]'
INFO: [Synth 8-3333] propagating constant 0 across sequential element (envelope_grain/\iot_reg[15])
WARNING: [Synth 8-3332] Sequential element (iot_reg[15]) is unused and will be removed from module envelope.

Finished Cross Boundary and Area Optimization : Time (s): cpu = 00:00:25 ; elapsed = 00:00:29 . Memory (MB): peak = 762.789 ; gain = 479.438

Start ROM, RAM, DSP and Shift Register Reporting

Block RAM: Preliminary Mapping Report (see note below)

```

+-----+-----+-----+-----+-----+-----+-----+-----+-----+
+-----+-----+-----+
|Module Name | RTL Object | PORT A (Depth x Width) | W | R | PORT B (Depth x Width) | W | R |
| Ports driving FF | RAMB18 | RAMB36 |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+
+-----+-----+-----+

```

```

|pitchshift: | memory_reg | 4 K x 16(NO_CHANGE) | W | | 4 K x 16(WRITE_FIRST) | | R
| Port A and B | 0 | 2 |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+
+-----+-----+-----+

```

Note: The tale above is a preliminary report that shows the Block RAMs at the current stage of the synthesis flow. Some Block RAMs may be reimplemented as non Block RAM primitives later in the synthesis flow. Multiple instantiated Block RAMs are reported only once.

DSP: Preliminary Mapping Report (see note below)

```

+-----+-----+-----+-----+-----+-----+-----+-----+-----+
+-----+-----+-----+-----+
|Module Name | DSP Mapping | A Size | B Size | C Size | D Size | P Size | AREG | BREG |
CREG | DREG | ADREG | MREG | PREG |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+
+-----+-----+-----+-----+
|envelope | A*B | 16 | 16 | - | - | 32 | 0 | 0 | - | -
| - | 0 | 0 |
|envelope | A*B2 | 16 | 16 | - | - | 32 | 0 | 1 | - | -
| - | 0 | 0 |
|envelope | A*B | 18 | 17 | - | - | 48 | 0 | 0 | - | -
| - | 0 | 0 |
|envelope | (PCIN>>17)+A*B | 17 | 16 | - | - | 48 | 0 | 0 | - | -
| - | - | 0 | 0 |
|envelope | A*B | 16 | 16 | - | - | 32 | 0 | 0 | - | -
| - | 0 | 0 |
|envelope | A*B | 18 | 17 | - | - | 48 | 0 | 0 | - | -
| - | 0 | 0 |
|envelope | (PCIN>>17)+A*B | 17 | 16 | - | - | 48 | 0 | 0 | - | -
| - | - | 0 | 0 |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+
+-----+-----+-----+-----+

```

Note: The table above is a preliminary report that shows the DSPs inferred at the current stage of the synthesis flow. Some DSP may be reimplemented as non DSP primitives later in the synthesis flow. Multiple instantiated DSPs are reported only once.

Finished ROM, RAM, DSP and Shift Register Reporting

```

-----
INFO: [Synth 8-4480] The timing for the instance i_0/pitch_shift/memory_reg_0 (implemented
as a block RAM) might be sub-optimal as no optional output register could be merged into
the block ram. Providing additional output register may help in improving timing.
INFO: [Synth 8-4480] The timing for the instance i_0/pitch_shift/memory_reg_1 (implemented
as a block RAM) might be sub-optimal as no optional output register could be merged into
the block ram. Providing additional output register may help in improving timing.

```

Report RTL Partitions:

```

+-----+-----+-----+
| |RTL Partition |Replication |Instances |
+-----+-----+-----+
+-----+-----+-----+

```

Start Applying XDC Timing Constraints

Finished Applying XDC Timing Constraints : Time (s): cpu = 00:00:45 ; elapsed = 00:00:51 .
Memory (MB): peak = 812.496 ; gain = 529.145

Start Timing Optimization

Finished Timing Optimization : Time (s): cpu = 00:00:46 ; elapsed = 00:00:52 . Memory (MB):
peak = 825.695 ; gain = 542.344

Start ROM, RAM, DSP and Shift Register Reporting

Block RAM: Final Mapping Report

```
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
+-----+-----+-----+
|Module Name | RTL Object | PORT A (Depth x Width) | W | R | PORT B (Depth x Width) | W | R |
| Ports driving FF | RAMB18 | RAMB36 |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
+-----+-----+-----+
|pitchshift: | memory_reg | 4 K x 16(NO_CHANGE) | W | | 4 K x 16(WRITE_FIRST) | | R |
| Port A and B | 0 | 2 |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
+-----+-----+-----+
```

Finished ROM, RAM, DSP and Shift Register Reporting

Report RTL Partitions:

```
+-----+-----+-----+
| |RTL Partition |Replication |Instances |
+-----+-----+-----+
+-----+-----+-----+
```

Start Technology Mapping

INFO: [Synth 8-4480] The timing for the instance pitch_shift/memory_reg_0 (implemented as a
block RAM) might be sub-optimal as no optional output register could be merged into the
block ram. Providing additional output register may help in improving timing.
INFO: [Synth 8-4480] The timing for the instance pitch_shift/memory_reg_1 (implemented as a
block RAM) might be sub-optimal as no optional output register could be merged into the
block ram. Providing additional output register may help in improving timing.

Finished Technology Mapping : Time (s): cpu = 00:00:48 ; elapsed = 00:00:55 . Memory (MB):
peak = 853.461 ; gain = 570.109

Report RTL Partitions:

```
+-----+-----+-----+
| |RTL Partition |Replication |Instances |
+-----+-----+-----+
```

```

++-----+-----+-----+
-----
Start IO Insertion
-----
-----
Start Flattening Before IO Insertion
-----
-----
Finished Flattening Before IO Insertion
-----
-----
Start Final Netlist Cleanup
-----
-----
Finished Final Netlist Cleanup
-----
-----
Finished IO Insertion : Time (s): cpu = 00:00:50 ; elapsed = 00:00:56 . Memory (MB): peak =
853.461 ; gain = 570.109
-----

```

Report Check Netlist:

```

+-----+-----+-----+-----+-----+
|      |Item                |Errors |Warnings |Status |Description      |
+-----+-----+-----+-----+-----+
|1     |multi_driven_nets |      0|      0|Passed |Multi driven nets |
+-----+-----+-----+-----+-----+

```

```

-----
Start Renaming Generated Instances
-----

```

```

-----
Finished Renaming Generated Instances : Time (s): cpu = 00:00:50 ; elapsed = 00:00:57 .
Memory (MB): peak = 853.461 ; gain = 570.109
-----

```

Report RTL Partitions:

```

++-----+-----+-----+
| |RTL Partition |Replication |Instances |
+-----+-----+-----+
+-----+-----+-----+

```

```

-----
Start Rebuilding User Hierarchy
-----

```

```

-----
Finished Rebuilding User Hierarchy : Time (s): cpu = 00:00:51 ; elapsed = 00:00:57 . Memory
(MB): peak = 853.461 ; gain = 570.109
-----

```

```

-----
Start Renaming Generated Ports
-----

```

```

-----
Finished Renaming Generated Ports : Time (s): cpu = 00:00:51 ; elapsed = 00:00:57 . Memory
(MB): peak = 853.461 ; gain = 570.109
-----

```

Start Handling Custom Attributes

Finished Handling Custom Attributes : Time (s): cpu = 00:00:51 ; elapsed = 00:00:57 .
Memory (MB): peak = 853.461 ; gain = 570.109

Start Renaming Generated Nets

Finished Renaming Generated Nets : Time (s): cpu = 00:00:51 ; elapsed = 00:00:57 . Memory
(MB): peak = 853.461 ; gain = 570.109

Start ROM, RAM, DSP and Shift Register Reporting

Static Shift Register Report:

Module Name	RTL Name	Length	Width	Reset Signal	Pull out first Reg	Pull out last Reg	SRL16E	SRLC32E
SILO YES	random12bit/x28_reg 1 0	5	1	NO	NO			
SILO YES	random12bit/x32_reg 1 0	3	1	NO	NO			

Finished ROM, RAM, DSP and Shift Register Reporting

Start Writing Synthesis Report

Report BlackBoxes:

	BlackBox name	Instances
1	xadc_wiz_0	1

Report Cell Usage:

	Cell	Count
1	xadc_wiz_0_bbox_0	1
2	BUFG	3
3	CARRY4	616
4	DSP48E1	6

5	DSP48E1_1		1
6	LUT1		94
7	LUT2		1157
8	LUT3		368
9	LUT4		663
10	LUT5		252
11	LUT6		166
12	ODDR		1
13	RAMB36E1		2
14	SRL16E		2
15	FDCE		16
16	FDPE		16
17	FDRE		173
18	LDC		16
19	IBUF		11
20	OBUF		9

+-----+-----+-----+

Report Instance Areas:

	Instance	Module	Cells
1	top		3598
2	grain_density	density	20
3	envelope_grain	envelope	3141
4	generate_clock	i2s_clock_generator	28
5	i_i2s_transmitter	i2s_transmitter	36
6	i_powerup_controller	powerup_controller	22
7	pitch_shift	pitchshift	95
8	random12bit	rand	78

Finished Writing Synthesis Report : Time (s): cpu = 00:00:51 ; elapsed = 00:00:57 . Memory (MB): peak = 853.461 ; gain = 570.109

Synthesis finished with 0 errors, 0 critical warnings and 3 warnings.

Synthesis Optimization Runtime : Time (s): cpu = 00:00:38 ; elapsed = 00:00:47 . Memory (MB): peak = 853.461 ; gain = 246.590

Synthesis Optimization Complete : Time (s): cpu = 00:00:51 ; elapsed = 00:00:57 . Memory (MB): peak = 853.461 ; gain = 570.109

INFO: [Project 1-571] Translating synthesized netlist

INFO: [Netlist 29-17] Analyzing 653 Unisim elements for replacement

INFO: [Netlist 29-28] Unisim Transformation completed in 0 CPU seconds

WARNING: [Netlist 29-101] Netlist 'SILO' is not ideal for floorplanning, since the cellview 'envelope' contains a large number of primitives. Please consider enabling hierarchy in synthesis if you want to do floorplanning.

INFO: [Project 1-570] Preparing netlist for logic optimization

INFO: [Opt 31-138] Pushed 0 inverter(s) to 0 load pin(s).

INFO: [Project 1-111] Unisim Transformation Summary:

A total of 16 instances were transformed.

LDC => LDCE: 16 instances

INFO: [Common 17-83] Releasing license: Synthesis

52 Infos, 13 Warnings, 0 Critical Warnings and 0 Errors encountered.

synth_design completed successfully
synth_design: Time (s): cpu = 00:00:54 ; elapsed = 00:01:01 . Memory (MB): peak = 853.461 ;
gain = 581.582
WARNING: [Constraints 18-5210] No constraint will be written out.
INFO: [Common 17-1381] The checkpoint
'Q:/Xilinx/work/grain_silo/grain_silo.runs/synth_1/SILO.dcp' has been generated.
INFO: [runtcl-4] Executing : report_utilization -file SILO_utilization_synth.rpt -pb
SILO_utilization_synth.pb
report_utilization: Time (s): cpu = 00:00:00 ; elapsed = 00:00:00.107 . Memory (MB): peak =
853.461 ; gain = 0.000
INFO: [Common 17-206] Exiting Vivado at Sun Oct 28 18:07:32 2018...

9. References

- [1] D. Gabor, "Theory of communication," *Journal of the Institution of Electrical Engineers - Part I: General*, vol. 94, no. 73, pp. 58–58, 1947.
- [2] I. Xenakis, *Formalized music: thought and mathematics in composition*. Bloomington: Indiana University Press, 1971.
- [3] C. Roads, "Automated Granular Synthesis of Sound," *Computer Music Journal*, vol. 2, no. 2, p. 61, 1978.
- [4] B. Truax, "Discovering inner complexity: Time shifting and transposition with a real-time granulation technique". *Computer Music Journal*, 18(2), 38-48, 1994.
- [5] J.-C. Risset, "Musical sound models for digital synthesis," in *ICASSP '86. IEEE International Conference on Acoustics, Speech, and Signal Processing*, 1986.
- [6] M. Goodwin and M. Vetterli, "Atomic decompositions of audio signals," in *Proceedings of 1997 Workshop on Applications of Signal Processing to Audio and Acoustics*.
- [7] Min-Ho Hyun, M.-H. Hyun, J.-H. Na, and S.-Y. Hwang, "Design of a pipelined music synthesizer based on the wavetable method," *IEEE Trans. Consum. Electron.*, vol. 43, no. 3, pp. 605–613, 1997.
- [8] R. Fischman, "Microstructure and macrostructure," *Organised Sound*, vol. 3, no. 1, pp. 1–2, 1998.
- [9] S. Dubnov, Z. Bar-Joseph, R. El-Yaniv, D. Lischinski, and M. Werman, "Synthesizing sound textures through wavelet tree learning," *IEEE Comput. Graph. Appl.*, vol. 22, no. 4, pp. 38–48, 2002.
- [10] T. Opie, "Creation of a Real-Time Granular Synthesis Instrument for Live Performance". Masters Thesis. Brisbane, Australia. QUT", 2003.
- [11] R. Fischman, "Clouds, Pyramids, and Diamonds: Applying Schrödinger's Equation to Granular Synthesis and Compositional Structure," *Computer Music Journal*, vol. 27, no. 2, pp. 47–69, 2003.
- [12] C. Roads, *Microsound*. MIT Press, 2004.
- [13] I. Song, G. Governatori, and L. Diederich, "Automatic synthesis of reactive agents," in *2010 11th International Conference on Control Automation Robotics & Vision*, 2010.
- [14] A. Haveliya, "Design and Simulation of 32-Point FFT Using Radix-2 Algorithm for FPGA Implementation," in *2012 Second International Conference on Advanced Computing & Communication Technologies*, 2012.
- [15] S. Aisyah, "FPGA-based sound synthesis by digital waveguide," in *2015 6th International Conference on Modeling, Simulation, and Applied Optimization (ICMSAO)*, 2015.
- [16] S. O'Leary and A. Robel, "A Montage Approach to Sound Texture Synthesis," *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, vol. 24, no. 6, pp. 1094–1105, 2016.

-
- [17] Doulos, “The VHDL Golden Reference Guide”,
<http://www.ics.uci.edu/~jmoorkan/vhdlref>, Doulos; 2nd edition edition, 1997.
- [18] P. J. Ashenden, “The VHDL Cookbook, First Edition”, <http://esd.cs.ucr.edu/vhdlcook>,
Ashenden, 1998.
- [19] Yalamanchili, “VHDL: A Starter's Guide, 2nd Edition”,
<http://sudha-curr.ece.gatech.edu/vhdl-starters-guide>, Prentice Hall; 2 edition, 2005.
- [20] B. Mealy and F Tappero, “Free Range VHDL: The No-frills Guide to Writing Powerful
Code for Your Digital Implementations”, https://archive.org/details/free_range_vhdl,
Free Range Factory, 2013.
- [21] Make Noise Company, “The Phonogene”,
<http://www.makenoisemusic.com/modules/phonogene>
- [22] Mutable Instruments, “Clouds”, <https://mutable-instruments.net/modules/clouds>,
Accessed 07/04/2018, published 2014.
- [23] B. Truax, *Personal communication via email*, 2004.
- [24] C. Roads, *CloudGenerator*, <http://clang.mat.ucsb.edu/software.html>, accessed
19/05/2018. Published in 1995.
- [25] P. Smaragdis and J. ffitich, *grain.c*,
<https://github.com/csound/csound/blob/develop/Opcodes/grain.c>, accessed 19/05/2018,
published in 1994.
- [26] N. Collins, “Review of Radiohead Kid A/Amnesiac/Hail to the Thief.” in *Computer
Music Journal*, Vol 28(1): 73-77, 2004.
- [27] J. Groh, “An efficient, Precise Frequency Shifter”, in *Csound Magazine*, 2006.
- [28] H.S. Carslaw, *Introduction to the Theory of Fourier's Series and Integrals*, 3rd ed., rev.
and enl. New York: Dover, 1921.
- [29] J. Arndt, *Matters Computational*. <https://archive.org/details/ftxbook>, accessed
19/05/2018, published in 2010.
- [30] Xilinx, *Digilent Arty S7-50: Spartan-7 FPGA*.
<https://www.xilinx.com/products/boards-and-kits/1-pnziih.html>, accessed 10/03/2018,
published in 2017.
- [31] Tasty Chips Electronics
<http://cdm.link/2017/08/granular-lovers-get-kickstarter-funded-hardware-synth>,
accessed 24/10/2018
- [32] S. Bernsee, *Pitch Shifting Using the Fourier Transform*, 1999.
<http://blogs.zynaptiq.com/bernsee/pitch-shifting-using-the-ft>
- [33] J. Clark and R. Hordijk, “Chapter 13. Frequency and Pitch Shifting”, from *A Primer on
Advanced Synthesis Techniques*.
https://www.cim.mcgill.ca/~clark/nordmodularbook/nm_spectrum_shift.html