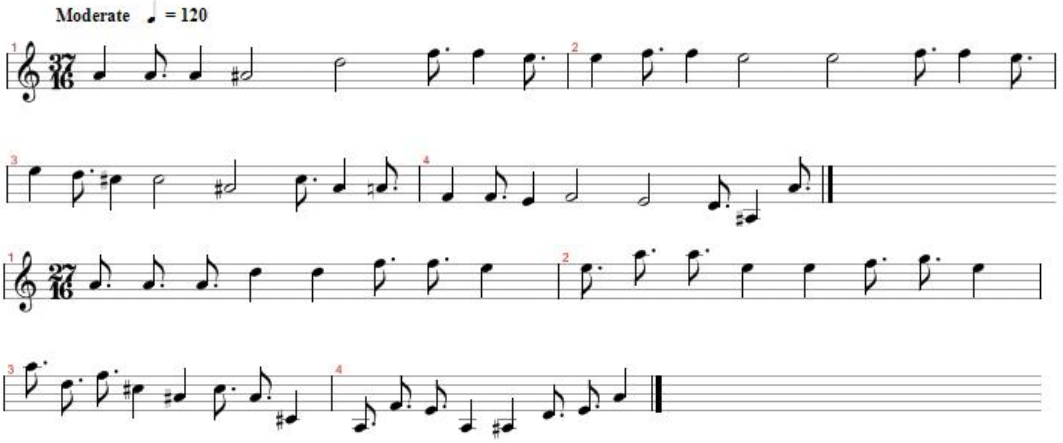
CHALMERS



Algorithmic Composition with Virtual Instrument in MATLAB and on FPGA

Master of Science Thesis in Integrated Electronic System Design

REMZI YAGIZ MUNGAN

Chalmers University of Technology
University of Gothenburg
Department of Computer Science and Engineering
Göteborg, Sweden, June 2010

Algorithmic Composition with Virtual Instrument in MATLAB and on FPGA

Remzi Yagiz Mungan

Examiner: Lena Peterson

Chalmers University of Technology
University of Gothenburg
Department of Computer Science and Engineering
SE-412 96 Göteborg
Sweden
Telephone + 46 (0)31-772 1000

[Cover: Music generated via simulating the HDL code in vsim. Figure 17.a and Figure 17.b in page 39.]

Department of Computer Science and Engineering
Göteborg, Sweden June 2010

Abstract

Music is an important part of most people's lives; it is dominantly created by humans. In this thesis, an alternate source of music, which is hardware in the form of a field programmable gate array (FPGA), is presented. The designed FPGA can create and play its own music. The music is composed by an algorithm, which was developed in MATLAB by combining genetic programming concepts with music theory in order to increase performance of the composer. Initially, a population made of composition pieces created with respect to harmony, is generated. Later, the population goes through genetic processes resulting with the final composition. The fitness function that determines the individual's ability to survive was optimized through listening tests. The virtual instrument was also developed in MATLAB as a waveform and an attack-decay-sustain-release (ADSR) envelope by recording and analysing an acoustic-electric guitar. The synthesizer like approach allows use of less circuit elements then does a physical model.

Hardware implementation is done in a Xilinx University Program Virtex II-Pro board. The audio coder/decoder, AC97 is used as the interface between the digital-to-analog converter (DAC) and the circuit block that handles the composition. The sampling frequency in the FPGA implementation is kept at 9.765 KHz due to problems with the design suite and a smaller population is employed in order to fit into the FPGA. The compositional performance of the implementation in FPGA is comparable to the performance of the algorithm in MATLAB.

This work shows that a portable music player that composes its own music seems a possibility in the future with improvements in compositional capability or the sound quality of the instrument. Introduction of a more complex generation conditions and a more complex elimination conditions could be an improvement. Another improvement would be to enable the creation of polyphonic compositions rather than monophonic compositions. On the instrument side, one possible improvement can be using industrially used sampled instruments.

Acknowledgement

First, I want to thank my soon-to-be wife Elif Selin Baytok, and my parents Günnur Mungan and Mehmet Emin Mungan. Since, with their support I was able to attend and complete my Master's Degree at Chalmers University of Technology.

Second, I thank my supervisor and my examiner Lena Peterson for allowing me and helping to work in this topic, which is very interdisciplinary. Her guidance helped me find my way when I was lost or headed into endless paths.

In addition, I must thank the people whom I have discussed the topic, including Prof. Torbjörn Lundh from Department of Mathematics for introducing the Hurst exponent, Gökhan Berberoğlu, a Master's Student from University of Gothenburg, Alen Bardizbanyan a fellow IESD student and people, who have participated in the listening tests.

Last but not least, I thank the Chalmers CSE Faculty, specially Lars Svensson, Per Larsson-Edefors and Lars Kollberg and my partners in IESD Programme and friends in Sweden for making my adventure in Chalmers University of Technology worthy, improving, exciting and beautiful both academically and socially.

Table of Contents

List of Abbreviations

ADC	Analog-to-Digital Converter
ADSR	Attack Decay Sustain Release
BRAM	Block Random-Access Memory
CF	Compact Flash
CODEC	Coder/Decoder
DAC	Digital-to-Analog Converter
DCM	Digital Clock Manager
FFT	Fast Fourier Transform
FIR	Finite Impulse Response
FPGA	Field Programmable Gate Array
GCLK	Global Clock Buffer
HDL	Hardware Description Language
IO	Input/Output
IOB	Input/Output Block
LFSR	Linear Feedback Shift Register
LUT	Look-up Table
MIDI	Musical Instrument Digital Interface
MEMS	MicroElectroMechanical Systems
MSB	Most Significant Bit
RAM	Random-Access Memory
ROM	Read-Only Memory
SNR	Signal-to-Noise Ratio
VHDL	Very-High-Speed Integrated Circuit Hardware Design Language
Xor	Exclusive Or

List of Figures

List of Tables

1. Preface

Since pre-historic ages, music has been a part of most of the cultures [1, 2]; and for most of the people, music is an important aspect of their lives. Researchers show that, on average a person listens to music around from 1.72 [3] to 2.5 [4] hours intentionally excluding the unintentional music that we hear such as music coming from stores, cars, cafes, cell phones, television shoes, street performers. Listening to music changes our emotions [1, 5, 6], our moods [7], our efficiency [3], our shopping behaviour [8] and even our eating behaviour [9, 10].

The music that we listen to comes from many sources such as radios, televisions, computers, internet, personal audio players, concerts and shows; and mainly it is recordings of compositions, performances of compositions or improvisations made by humans. The algorithmic composition approach tries to change the main source of music from humans to computers or machines.

Algorithmic composition, never a mainstream field of study, has long been a topic that has captured attention of a number of composers. However, the earlier approach was mostly to imitate the composer while later approaches include machine learning to develop an intelligence that can compose music. Examples of this kind of works exist where the algorithm is in software and is operated in a computer either aiding the composer [11] or composing itself [12].

1.1. Aim of the Thesis

In this thesis, the aim is to explore and understand the possibilities of implementing algorithmic composition as hardware in the form of an integrated circuit, which composes music for listeners or consumers. An ideal result of this work is to come up with a device like an mp3 player, which will compose and play its own music at the press of a button. By adding another option to the vast number of sources of music, I hope that new and different compositions will be achieved, introducing an improved understanding for music and different emotions caused by music.

1.2. Method of the Thesis

For the thesis work, initially previous works on algorithmic composition and virtual instruments are studied. After the literature search, a composition algorithm and an accompanying virtual instrument is developed in MATLAB. The algorithm is fine-tuned with listening tests, where the tests are developed by considering psychoacoustics and psychology. Finally, after all the development has been finished the design is

implemented on FPGA with Verilog and Very High Speed Integrated Circuit Hardware Design Language (VHDL).

1.3. Limitations of the Thesis

The main limitation of the thesis work is the time limit. The time of a master's thesis work in Chalmers University of Technology is 20 weeks. During the development and research phase of the project a number of ideas have been discarded in order to reach the designated end, which is to see whether it is possible to have a portable audio player with the ability to compose and play satisfactory songs.

2. Introduction to Algorithmic Composition

In this chapter, I will introduce algorithmic musical composition and talk about its history, starting from the early approaches to current works and trends. The conclusion at the end of this chapter will lead to the development of my own algorithm.

2.1. Background

An algorithm is a step-by-step solution to a problem where the number of steps is finite, thus algorithmic composition is using a systematic method for composing music. Even though, algorithmic composition is sometimes taken as electronic music, not every algorithmic composition is electronic. The algorithm can be done with pen and paper. Clarence Barlow argues that he can reach the same music without a computer while using the same algorithms [13].

Algorithmic composition, even though is a relatively new name, have long been a point of interest. Ancient Greek philosophers such as Pythagoras, Ptolemy and Plato have mentioned the formalism and the mathematical rules that lie beneath the music [14]. Even, Plato calls the music created by the movements of the planets "the music of the spheres". Later, in Middle Ages, algorithmic composition could be seen in canonical songs [15] where the composer was only creating a piece of melody and the rest of the song were derived from the core. In the 18^{th} century, some composers including Mozart and Haydn used algorithmic techniques to create a game where the music is composed by dice or just saying random numbers [16]. Works of John Cage include algorithmic composition where movements of a chess play are used for composing. However, it seemed inevitable to use computers for composing**.** The earliest example of computers used for composing, dates back to 1957 with the Illiac Suite [17].

2.2. Types of Algorithms

Today, different types of algorithms are used for composing. Mainstream algorithms include rule-based algorithms, grammar based algorithms, stochastic processes, Markov chains, chaotic algorithms, genetic algorithms and algorithms based on artificial intelligence. Some composition systems also include sound synthesis; thus, they can play the output composition. Examples for all the types is given in following sections.

2.2.1. Rule-Based Algorithms

With rule-based compositions, certain rules are used for forming up a composition. These rules are generally generated by investigating important works, and they are constructed such that the rules also define what will come next. Examples include

Ebcioglu's CHORAL, which composes chorals, where the more than 350 rules are derived from works of J. S. Bach [18].

One criticism [19] of rule-based algorithm is that, since the composer, either a human or a computer, is limited by well-defined rules, the resulting music is not something that is exceedingly new. Thus, even though rule-based algorithms produce correctly sounding music, they do not introduce compositions that have not been heard before. On the other hand, the tedious work of extracting rules increases the understanding of the dynamics of composition.

In this method, the composer's main job is to collect or to extract rules for the program, while the output music carries resemblance to the pieces, which the rules are extracted from.

2.2.2. Grammar-Based Algorithms

Formal grammars are one of the main topics in computer science and linguistics, where the initial aim is to formalize the spoken languages and the programming languages. Grammar-based composition takes music as a language; later it analyses and formalizes music like a language. L-systems are also used as grammar based composition [13, 20]

2.2.3. Stochastic Algorithms

Stochastic processes include probabilistic methods, as purely random choosing making up the simplest form. The introduction of weights and selection that is more intelligent increases the quality of the compositions.

The weighing functions can be generated by investigating compositions; later these statistics can be used for building a new composition. Markov chains are also used widely for composing. A Markov chain holds the probabilistic information in which a certain state can be followed by other states. Xenakis is among the pioneers in the use of stochastic processes for composition [21]. On the hand, even though randomness can be achieved many ways, random functions in software are pseudorandom, meaning that they are generated from deterministic processes such as the clock of the processor. Yet, completely random values can also be achieved using various noise sources in electronics such as radioactive decay, flicker noise, 1/f noise [22] or analog-to-digital converter (ADC) offset. Those techniques are also preferred in cryptology and security for their higher randomness.

2.2.4. Chaotic Algorithms

The research in irregularity, non-linearity in nature has given birth to chaos theory. Use of chaos theory in music has two main applications; small-scale applications such as sound synthesis and large-scale applications such as composition. Fractals, which are equations that can output equations that are similar to the parent functions, are used as the main tools for computation. In addition, one must not forget that chaos is not random. Instead, it is non-linear. Thus, the output composition is unexpected, but has a general pattern. One interesting point is that since chaotic systems are highly sensitive to initial point, the result of an algorithm or a program might change due to any circumstances such as the computer used for running the program [23].

2.2.5. Artificial Intelligence Based Algorithms

The main advantage of systems with artificial intelligence is their ability to learn. The system, Experiments in Musical Intelligence (EMI) has a database of rules like the rule-based algorithms, yet the system also has the capability to create a database by itself [24]. Methods based on artificial intelligence have been applied on composition, improvisation and performance-based systems in jazz [25].

2.2.6. Genetic Algorithms

With genetic algorithms, systems with evolution capability have been designed. Evolutionary algorithms are inspired after biology and biological concepts such as mutation, evolution and natural selection [26] and the algorithms search for optimum solutions. However, in the area of art in addition to optimum solutions other interesting solutions that result in satisfying aesthetics are also searched [27].

Composing with genetic algorithms requires also an entity in which selection methods are defined. The composed music parts either survive or die with respect to the selection following a Darwinist fashion. In some cases, three entities are developed for algorithmic composition where one entity composes, another gives feedback and the last one evolves the composition [10, 11, 28, 29, 30].

2.3. Conclusions

Algorithmic composition is an interesting interdisciplinary field, which mixes music, mathematics, physics, electronics, computer science and psychology. The aim of algorithmic composition is not only to recreate the creativity of a composer but also to understand and exceed it in order to reach new genres of music or unimagined types of compositions.

On the other hand, the future trend in algorithmic composition seems to lie in hybrid solutions with an ability to listen and understand where the listening or understanding is

mainly implemented by evolving algorithms such as genetic algorithms or artificial intelligence [27, 29, 31, 32].

3. Introduction to Virtual Instruments

In this chapter, virtual musical instruments are introduced. The introduction is followed by the literature search about the types of instruments with respect to implementation and realization issues. Finally, the chapter ends with a discussion and decision on implementing a virtual instrument by digital circuitry.

3.1. Background

With the advances in technology, a new set of musical instruments called virtual instruments have spawned. These instruments can be placed in two categories. The first type comprises the instruments that offer a new way of interaction between the player and the instrument. Examples include a pen, which creates sounds according to the writing or drawing [33], an enhanced saxophone that adds sound effects based on the players gestures [34] or video tracking based instruments where the instrument creates sounds based on the movements of hands or dance figures [35]. These instruments all have a new kind of feedback from the musician to the instrument via use of sensors, accelerometers, microelectromechanical systems (MEMS) devices and optics.

The second type of virtual instruments comprises instruments that create sound in a computer using either specific software or a hardware controller. Musical instrument digital interface (MIDI) is a standard and well-known example of implementation. These types of instruments employ either recorded samples or functions that create the sound dynamically. The functional types can be implemented in software by specific design kits in the form of certain formats.

3.2. Methods for Realizing Virtual Instruments

In this thesis, according to the above definitions, a virtual musical instrument of the second type will be implemented. It can be said that there are three methods for designing such an instrument. These are physical models, synthesizer method and wavetable method.

3.2.1. Physical Models

Physical models, as indicated by their name, try to model physically. However, what they model is not the sound, rather it is the instrument or any sound generating object itself. Using a physical model of an object is useful when doing an audio-video synthesis. On the instrument side, the physical models offer accuracy for some instruments that other methods cannot give. The reason is that physical models create

dynamic sounds that can duplicate the style of playing or the detailed and unique interactions between each note and the instrument.

Examples include the modelling of a piano [36] as a combination of the hammer, the string and the board. The hammer introduces non-linear differential system based on the speed of the keys. The string is modelled as a sum of two travelling waves and the soundboard is modelled as a 2000-tap finite impulse response (FIR) filter. Hardware wise, the implementation requires a number of filters and digital waveguides and the stability conditions must be met. In addition, there exist piano models without the board [37], violin models focusing on vibrato effect [38], guitar model [39, 40, 41]. More information on physical modelling of instruments can be found in "Virtual musical instruments – natural sound using physical models" [42].

3.2.2. Synthesizer Method

This method is used in the synthesizers. In this method, the aim is to produce a sound that resembles the sound of the target instrument. Simply, a periodic waveform is combined with an envelope to form the sound signal. The periodic waveform can be a simple sinusoidal, triangle, square, noise or more complex waveforms, which can be seen as the sound of the string, while the envelope adds the sound characteristic of the body or the room.

3.2.3. Wavetable Method

In fact, the wavetable method does not include any kind of sound generation. Instead, it is look-up-table method where recorded data is used. The sound quality of these instruments is based on the recorded instrument and recording gear and techniques, thus the quality can be as good as that of the instrument itself. One limitation of this method is that it cannot be used to realize new instruments.

3.3. Conclusions

In this project, a real-time virtual instrument will be designed in software and it will be implemented in an FPGA. The physical models, even though they are attractive due to the possibilities of sounds it offers, will not be used. The reasons for this decision are the design complexity of the mathematical and physical parts, and the implementation problems in hardware. The possible hardware problems can be argued as the necessity of a large circuitry for accuracy and stability reasons. Thus, in order to have a dynamic design the synthesizer method will be used. Yet, the single most important design criterion should be the performance of cognised sound, rather than the sound itself.

4. Software Implementation

In this chapter, I explain the algorithm that operates as the composer. The algorithm is developed by using genetic programming and tonal harmony in MATLAB. Thus, there will also be related introductory information about MATLAB, genetic programming and tonal harmony. The pseudo codes can be found in Appendix A.

4.1. MATLAB

MATLAB is chosen as the algorithm development platform in software rather than any of languages including more generic languages like C or more specific languages like MusicXML. The main reasons behind choosing MATLAB can be summarized as follows.

- Technical computing language: It allows easy implementation of mathematical functions such as signal creation, vector multiplications, statistical calculations, signal processing and filtering, which will be necessary for the thesis.
- Toolboxes for various specific topics exist or they can be designed. Toolboxes like MIDI, genetic programming, signal processing might be necessary.

4.2. Algorithm

After the literature study described in Chapter 2, I decided to create the algorithm based on music theory and genetic programming. The initial study shows that the composition process can benefit greatly from genetic programming. With the introduction of initially creating a number of compositions and later choosing the best, the performance is expected to be better than a process, where no kind of iteration is allowed.

4.2.1. Genetics and Evolution

As briefly mentioned in Chapter 1, genetic programming is inspired by biology. It introduces concepts like evolution, survival and generation to create an artificial intelligence. The aim is to give the computer the ability to start from a number of partial solutions and improve them in order to reach the final solution or the final code.

In order to provide a better understanding of the thesis, definitions of the genetic programming terms that are used in this work are given below [43].

Individual: In genetic programming, an individual is the unit partial solution that will form the final solution. The definition of the individual is important as it defines the structure of the algorithm.

Population: Population defines the number of individuals in the society. The number of individuals in a population increases the possibility of existence of fitter individuals with the cost of resources. In biology, these resources are food, water, shelter etc; while in genetic programming, resources mean complexity, power, memory and area.

Fitness: In biology, fitness of an individual determines its ability survive conditions such as climate changes, food shortage and existence of a predator. In genetic programming, by using a fitness function one can eliminate the weaker individuals, thus the next generation of individuals can be created with fitter individuals thus reaching the fittest individual. The fitness is defined via a fitness function, which employs the survival of the fittest in this project.

Tournament: With the employment of a tournament, the elimination process is not done on the whole population rather the elimination is done via direct comparison of two individuals. Thus, due to match-ups an individual may end up in a position that is much higher than the position it would be if a regular sorting were used. This may result with more variety if not fitter. It is also possible to employ tournaments where the participants are a randomly chosen subset of the whole population.

Crossover: In biology, during meiosis reproduction crossover provides the variability through the exchange of genes in the chromosomes. In genetic programming, this can be achieved by exchanging information between individuals.

Reproduction: In biology, reproduction means the production of an offspring and the offspring might have the same genetic information depending on the life form. However, in genetic programming, reproduction is the pass of an individual from one generation to another without any change.

4.2.2. Music Theory

Music theory is the study of music that investigates and analyses the working structure of music. The field has scholars focusing on music theory, psychology, acoustics, physics, physiology and psychoacoustics. Below are the explanations of the musical terms used in this thesis.

Temperament: Temperament is a term related to the tuning of the instruments; thus, it determines the actual frequencies of the notes. There are two types of temperament; they are just temperament and equal temperament. In just temperament, each 12 notes in an octave have the correct frequency. In just temperament, different notes are realized by dividing the length of a string into certain ratios. However, this is not implementable in most of the instruments as once the notes are realized in a certain key, notes in other

keys will be off since the string is divided with respect to the initial key. In those cases, equal temperament is used. Equal temperament is built by keeping the frequency of an octave as in the just temperament, while adjusting the other notes so that they are all slightly off in a way to compromise the over all accuracy [44]. The relative frequencies in the equal temperament can be seen in the following Table 1.

Place in a Major Scale	Interval	Relative Frequency
1	Unison	1.0000
#1 or ♭2	Minor Second	1.05946
2	Major Second	1.2246
#2 or ♭3	Minor Third	1.8921
3	Major Third	1.25992
4	Fourth	1.33483
#4 or ♭5	Diminished Fifth	1.41421
5	Fifth	1.49831
#5 or ♭6	Minor Sixth	1.58740
6	Major Sixth	1.68179
#6 or ♭7	Minor Seventh	1.78180
7	Major Seventh	1.88775
8	Octave	2.0000

Table 1: Relative Frequencies of Tones in an Octave

Scale: A scale is a set of notes from the 12 notes listed in Table 1. A scale has a reference, starting note, which is called the tonic, and the other notes in the scale are selected with respect to the tonic. Scales can include different number of notes, while most known scales like major and the minor scale have seven notes in them. Chromatic scale is the only scale that has 12 notes, where all the steps between the notes are chromatic and hence the name of the scale.

4.2.3. Scales

To limit the dissonancy and increase the tonality of the songs composed, libraries were created by using different scales. Even though, this ensures that the compositions will not wonder off-key notes, this also limits the sources that the algorithm can use. Thus, it can be seen as Rawls' maximin [45] approach.

The structure of MATLAB allows easy use of any scales or any signal at any frequency. All the scales have the tonal at A and in the A key. In addition, all the libraries are constructed around the note A4, which has a frequency of 440 Hz in equal temperament

and feature 16 notes for memory concerns. The number of notes is chosen as 16 since, it at worst gives two octaves of a scale and 16 can be index with four bits while using all the numbers generated efficiently. The scales that have been used in MATLAB implementation are as follows:

- Byzantine Scale
  - A – A# – C# –D – E – F – G#
- Be-Bop Scale
  - A – B – C# – D – E –F# – G – G#
- Chinese Scale
  - A – B – C# – E – F#
- Hungarian Gypsy Scale
  - A – B – C- D# – E – F – G
- Major Scale
  - A – B – C# – D – E – F# – G#
- Minor Blues Scale
  - A – C – D – D# – E – G
- Minor Scale
  - A – B – C – D – E – F – G
- Spanish Scale
  - A – A# – C# – D – E –F – G

4.2.4. Fitness Function

The fitness function concept, as introduced in Section 4.2.1 is a very important feature in genetic programming since it decides which individuals will be used and which will be discarded. However, one problem of using genetic programming in music is that even though the genetic algorithms evolve and search for the fittest or the optimal solution, there is no such solution in music since good music is a matter of personal preferences.

In this work, a number of fitness functions, which are in fact functions of mean and variance, are developed. The final fitness function, which is implemented in the hardware, is determined via a set of listening tests.

The inclusion of mean in a fitness function aims for the compositions that are not stuck either in the flat end or in the sharp end of the frequency spectra. This is done by calculating the mean of each individual and subtracting it from the index of the tonal, which is A4. The increase in the absolute difference between the mean and the tonal decreases the fitness of that individual.

On the other hand, variance is added to the fitness function in order to balance the effect of mean. If only mean is used as the fitness function, an individual that only has the tonal will be the fittest. Thus, in order to favour individuals that span a larger portion of frequency spectra, variance is included in the fitness function. I have designed the fitness function to be such that if the difference between the maximum variance in the population and the variance of the individual increases, the fitness of the individual decreases.

In addition, the Hurst exponent is also considered as another dimension for the fitness function. However, for hardware concerns, it has not been implemented. The Hurst exponent is a number between 0 and 1. If the Hurst exponent is 0.5, it means that the distribution is random and the number of increases is equal to number of decreases. On the other hand, if it is larger than 0.5, it means the distribution has an increasing trend and if it is smaller than 0.5, it means the distribution has a decreasing trend. Finally, the fitness function can be summarized as the following formula:

(tonic of the scale)-(mean of an individual) + (max variance in the population)-(population of an individual)

4.2.5. Structure of the Algorithm

The flow of the algorithm can be seen in Figure 1. Each individual is a set of pitches and they are initially created using harmony knowledge and a random number generator. The first pitch of the sequence, which is the first gene of the individual, is selected as the tonic of the key. The following pitches are decided via the random number generator.

True random number generators require physical actions such noise from the circuits, atmospheric noise, a uniform dice or a coin. On the other hand, pseudo-random events are algorithmically computer generated deterministic number, which can satisfy some of the statistical tests for randomness [46]. The “randi( )” function of MATLAB is used as the pseudo-random number generator. It gives uniformly distributed integers from 1 to the defined limit. The random generator is not suitable for security reasons; also, the randomly generated numbers can be recreated by initializing the function with the same seeds.

The range of the random generator is ∓ 2. It means that a pitch can at most be 3 notes flatter or 2 notes sharper than the previous one. The reason behind such a limitation is that it has been shown that humans [47] prefer smaller changes in frequency in sequential notes and preliminary sound tests have also proven so. The most preferred movement is chromatic while certain leaps like perfect fifth or an octave are also

accepted generally. The algorithm is allowed to wander in both directions with equal probability except for at the boundaries, where the current pitch is either the sharpest or the flattest in the library. Thus, at the boundaries, the algorithm is encouraged to wander away from the boundaries.

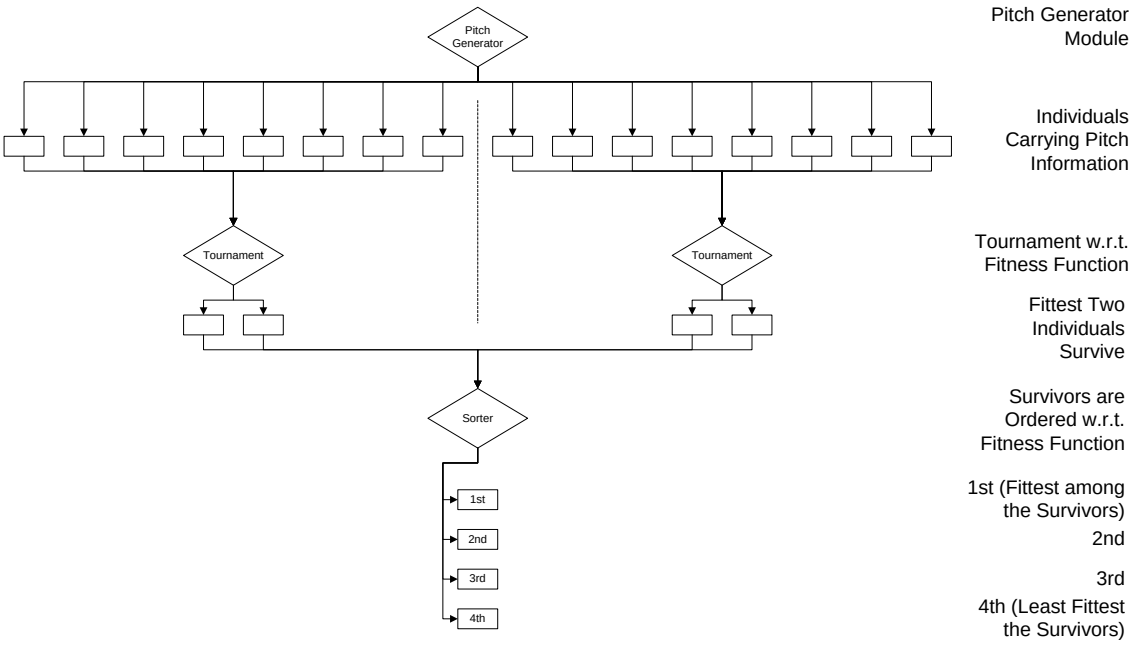


Figure 1: Flow of the Algorithm

Once, the desired number of individuals are generated, the tournament starts. Here the size of the pitch sequences pool or the population is a trade-off between the computational power and performance of the algorithm. As the size of the population increases, the possibility of having a fitter and different individual increases. Simulations in the software implementation have shown that populations of at least 64 individuals are necessary in this setup to have sufficient variety. The sufficiency is determined by the population size that can provide the above-mentioned different fitness functions with different fitter individuals.

Since there are two populations, two tournaments occur. One design question here is whether to have a tournament or a fitness proportional selection, which is the type of competition where all the individuals are sorted with respect to their fitness value. Even though, a tournament is computationally more complex and slow it might result in a better variety. In the regular competition, the fitter individuals will always end up as the survivors, in a tournament due to match-ups it is possible for an individual to end up with a rank that is higher than its fitness value suggests. This also explains why there are two different populations and tournaments, combining both populations and having a single tournament increases the chance that fitter individuals will be eliminated. Thus, I decided to use two independent tournaments where the finalists of the tournaments survive and the rest is destroyed.

Thus, at the end of the tournament section the algorithm is left with four sequences of pitches. These individuals are ordered with respect to their fitness values, and the final sets of pitches that will be used in the song are created as their children with crossovers between the individuals. This process can be seen in Figure 2.

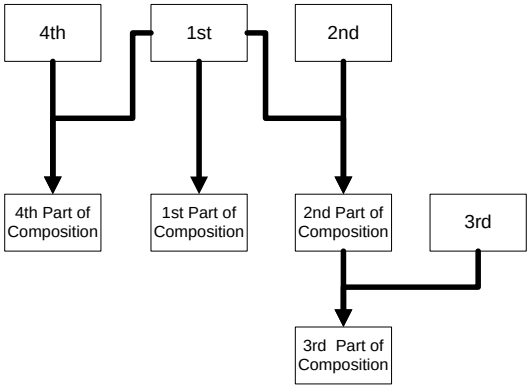


Figure 2: Crossover Map

Since there is no deterministic optimum solution that the program is trying to reach, no iteration is done. A single run of the algorithm ends with the second generation.

The rhythm generator is independent of the pitch generation process. The rhythm generator determines the durations of each pitch and the occurrences of rests. Again, four different rhythm sequences of eight notes are produced, each are looped a number of times such that the first part of the song is composed of the first set of final pitches and the first rhythm sequence looped enough times. The four different rhythm sequences are generated with different aims, such that some have longer notes, while some are faster or some cannot include rests. The aim of creating a limited number of rhythm elements such as eight and looping them is to employ the musical memory of humans, which is reported to be between 2 to 10 seconds [47], so that the mind has a repetitive pattern to follow.

Finally, a composition is achieved where initially a theme, which is in fact the fittest set of pitches, is introduced. Later, the theme is evaluated as the song deviates both rhythmically and melodically away from the main theme. At the last part, song makes a subtle return to theme by using the first-order child of the original theme.

4.3. Virtual Instrument

As described in Chapter 3, there are a number of ways to implement a virtual instrument. In this project, due to time limit, numerical or physical modelling of an instrument is not done. Instead, for creating a virtual instrument, synthesizer approach is selected.

In synthesizers, a periodic waveform and an envelope define the sound. The envelope is composed of four elements: attack, decay, sustain and release and hence called ADSR. Attack represents the time where the amplitude of an instrument starts from zero to its maximum value. Decay represents the time where the amplitude decreases to the stable level. Sustain represents the main sequence of the sound and release represents the time where the sound level returns to zero.

4.3.1. The Method

Once the synthesizer approach has been selected, the methods for extracting the coefficients of ADSR and the periodic waveform are searched. One must remember that the analysis methods in this works do not have to be implemented in real time, however the amount of information necessary to reconstruct the sounds is important as the reconstruction will be done in real time.

For the envelope, two different methods are found. Once the envelope is extracted from the signal, the corner points for the ADSR can be found either by using a differentiation method or a threshold method. The differentiation method is used in this work since it has been reported to detect the points more precisely [48, 49].

For the analysis of the periodical waveform, even though additive synthesis is seen as the most flexible and useful method, other methods are also investigated [50]. The linear time/frequency analysis method is an alternative to the fast Fourier transform (FFT) method; it increases the accuracy, which is decreased in FFT due to the windowing. The LTF method gives better frequency response in smaller times, and is useful for real time analysis [49]. However, that is not a problem in this case. Methods that define a sound with less information than the synthesizer method exists in [49], however the sound quality is reported to decrease.

4.3.2. Analysis of the Guitar

The target instrument is an Ovation 2771LX acoustic-electric guitar. That instrument is chosen due to its availability and due to is being equipped with electronics that allow direct recording through a sound card allowing a good signal-to-noise ratio (SNR).

The below recordings are done with a 24-bit ADC with a sampling frequency of 44100 Hz from the above mentioned guitar through a M-Audio Fast Track. The recorded signals in Figure 3 have a fundamental frequency of 440 Hz and each signal shows the production of the same note from different positions of the guitar. 440 Hz is chosen as the frequency to analyse since it is the tonal. In Figure 4**,** the periodic waveform is shown and in Figure 5, the extracted envelope is shown.

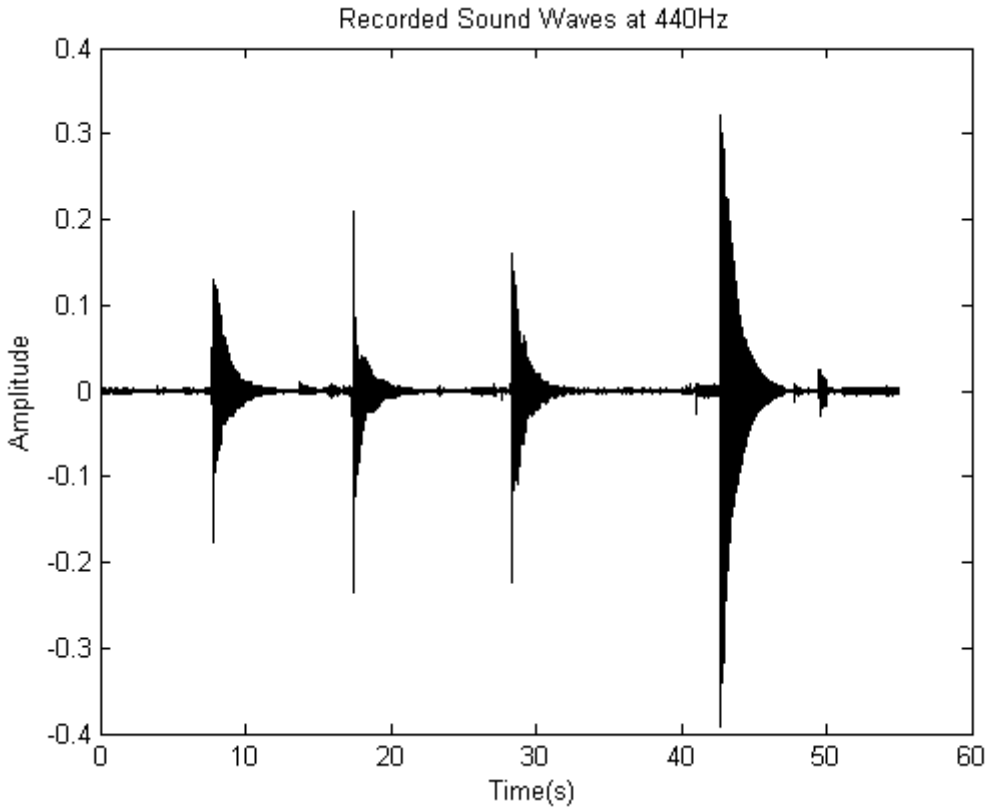


Figure 3: Recorded Sound Waves at 440 Hz

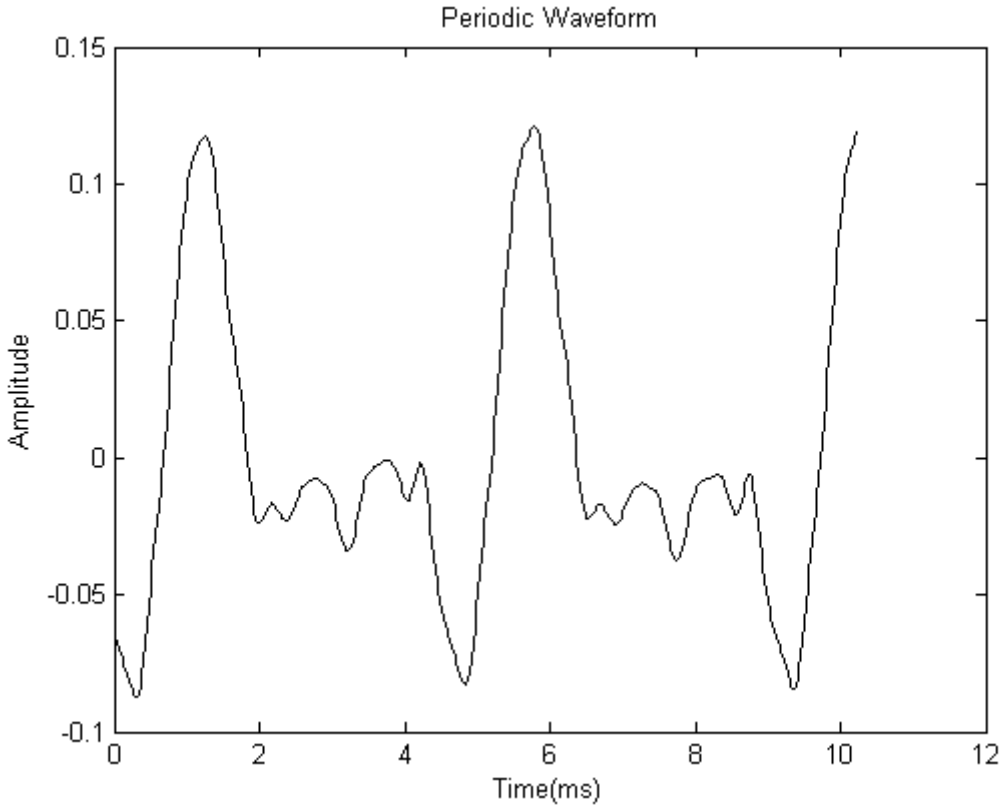


Figure 4: Periodic Recorded Waveform

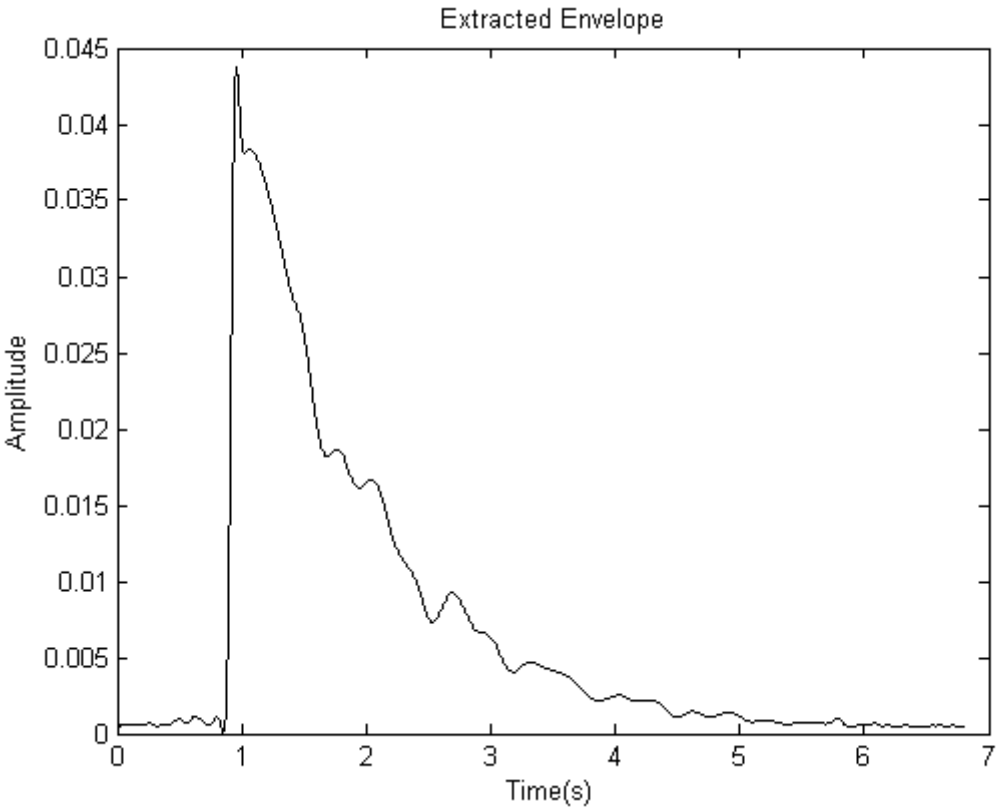


Figure 5: Extracted Envelope

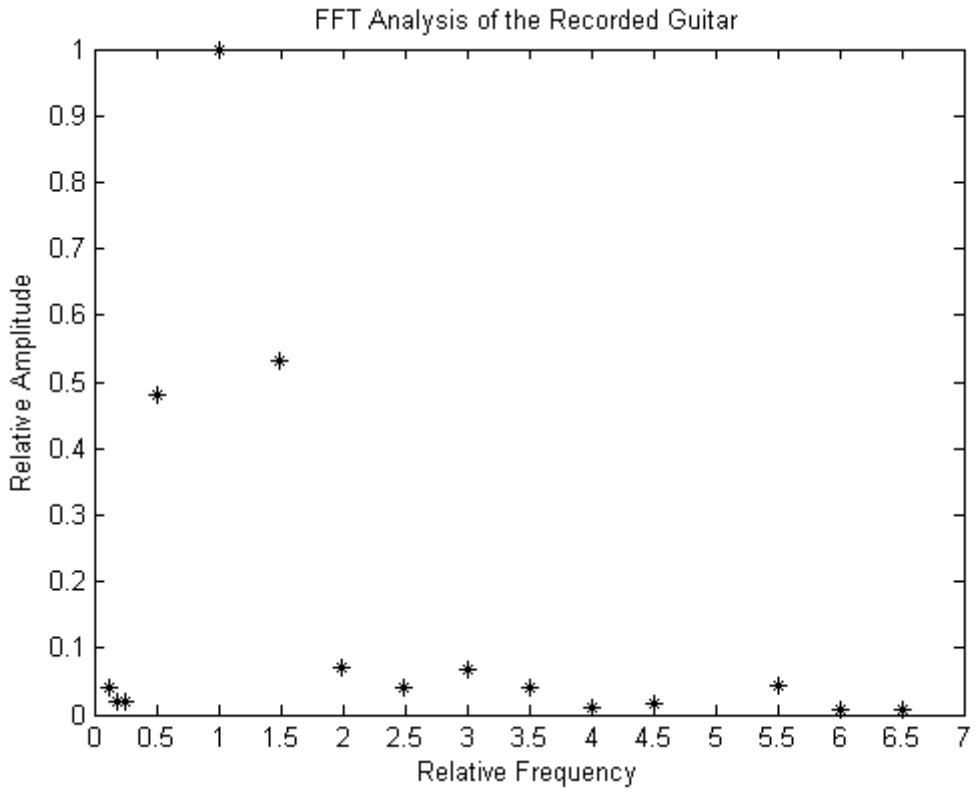


Figure 6: Results of the FFT Analysis

The recorded sound is analyzed by calculating the FFT of the wave with necessary amount of samples in MATLAB. The resulting harmonics and their amplitudes are shown in Figure 6**.** By comparing the results in Figure 6 with Table 1 in Chapter 4, one can have a better understanding of the sounds produced by a single stroke.

In addition, by looking at Figure 6, one can see that after the fundamental the most powerful tones are the first sharper fifth and the first flatter octave. These are followed by the second and the third sharper octaves.

4.3.3. Reconstruction of the Guitar Sound

Two different waveforms and one envelope are created in this work. In Figure 7 and Figure 8, the reconstructed waveforms can be seen. The first signal in Figure 7 is reconstructed by using 15 of the harmonics with the largest amplitudes, which are in Figure 6. Sine waves constructed with the amplitudes and frequencies from Figure 6 are added to create the signal. Since the phase information is not important for most cases, it is not used for the synthesis of the sounds [51, 49]. The mathematical formula including the phase for the additive synthesis is:

$$f(t) = \sum_{k=1}^{N} \mathrm{a_k} * \sin\left(w_k * t + \theta_k\right)$$

Here a_k is the amplitude coefficient of an harmonic, w_k is the angular frequency of the harmonic, θ_k is the phase and t is the time.

On the other hand, the second signal in Figure 8 is constructed via simple visual data fitting where the aim is to keep the main characteristic of the signal in Figure 4 in as simple as possible. The simpler signal is composed of two triangles, where the values of the triangle are the same except for the sign, thus the difference of the peaks and the wavy pattern in between the peaks are omitted. Preliminary sound tests have shown that the second signal is more pleasant. The reason is the existence of one of the high-end harmonics that causes obvious dislike for the first sound.

In addition to two periodic waveforms, two ADSR envelopes are also developed. In Figure 9, one can see the regular 3-pointed waveform. In order to increase the resemblance with the envelope in Figure 5, the sustain part in Figure 9 is divided into two parts with different slopes. However, informal listening tests showed that the resulting envelope in Figure 10 did not cause any change in the perceived sound.

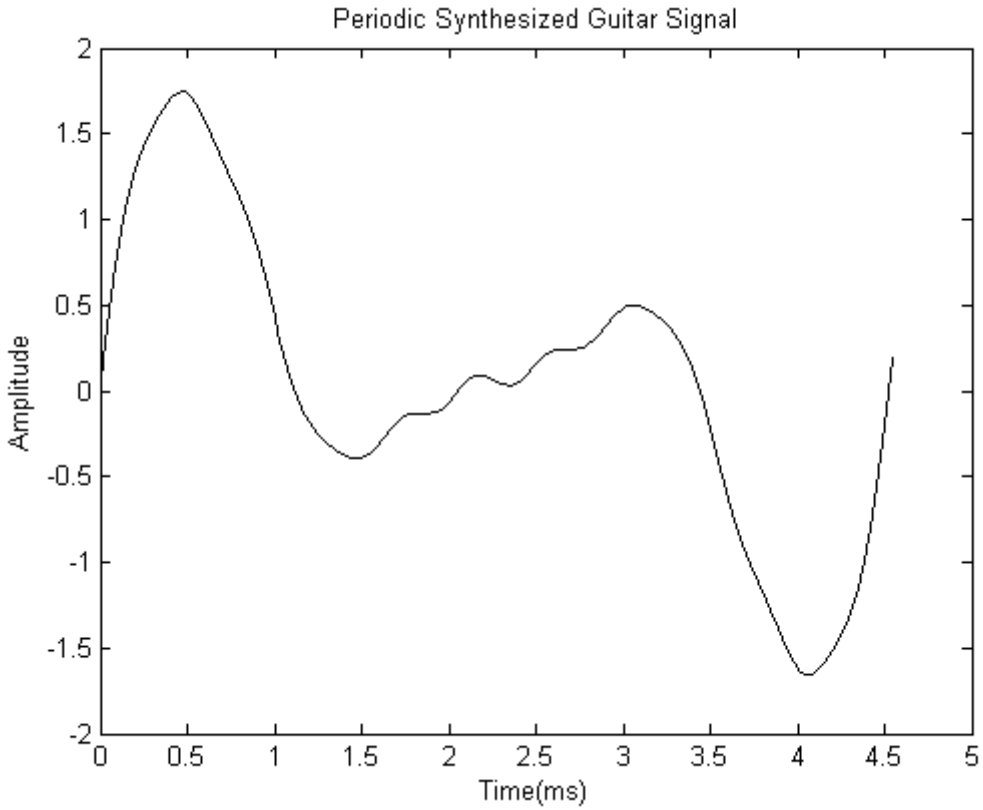


Figure 7: Periodic Waveform Reconstructed from 15 Harmonics

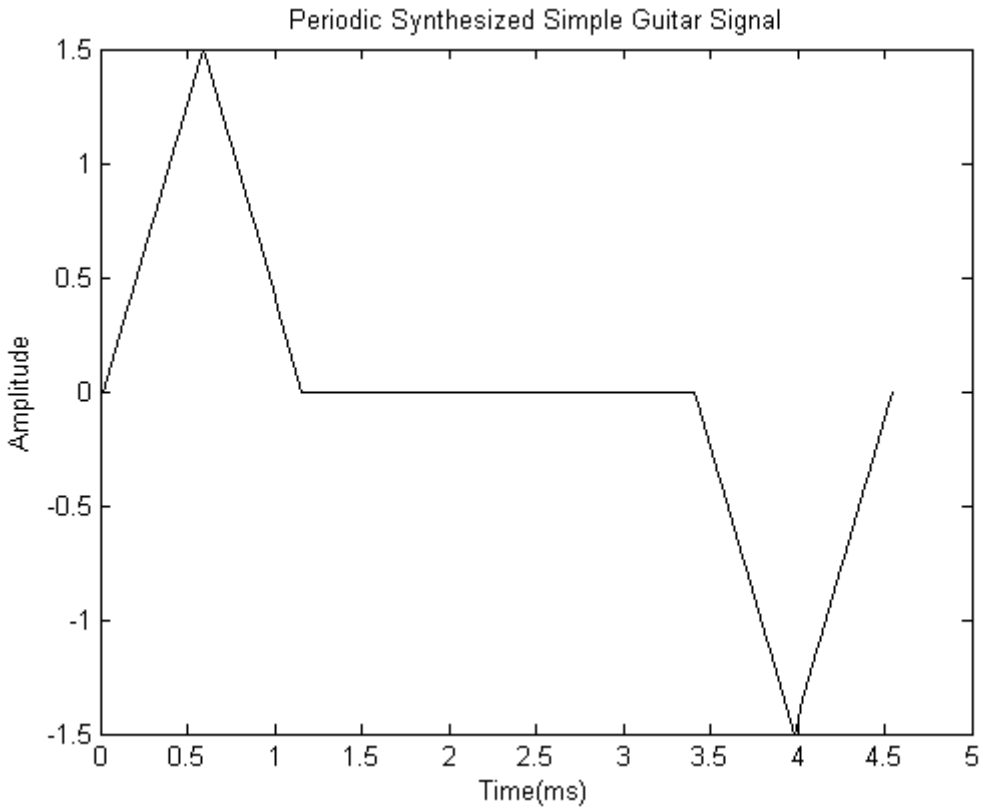


Figure 8: Periodic Waveform Reconstructed by Visual Data Fitting

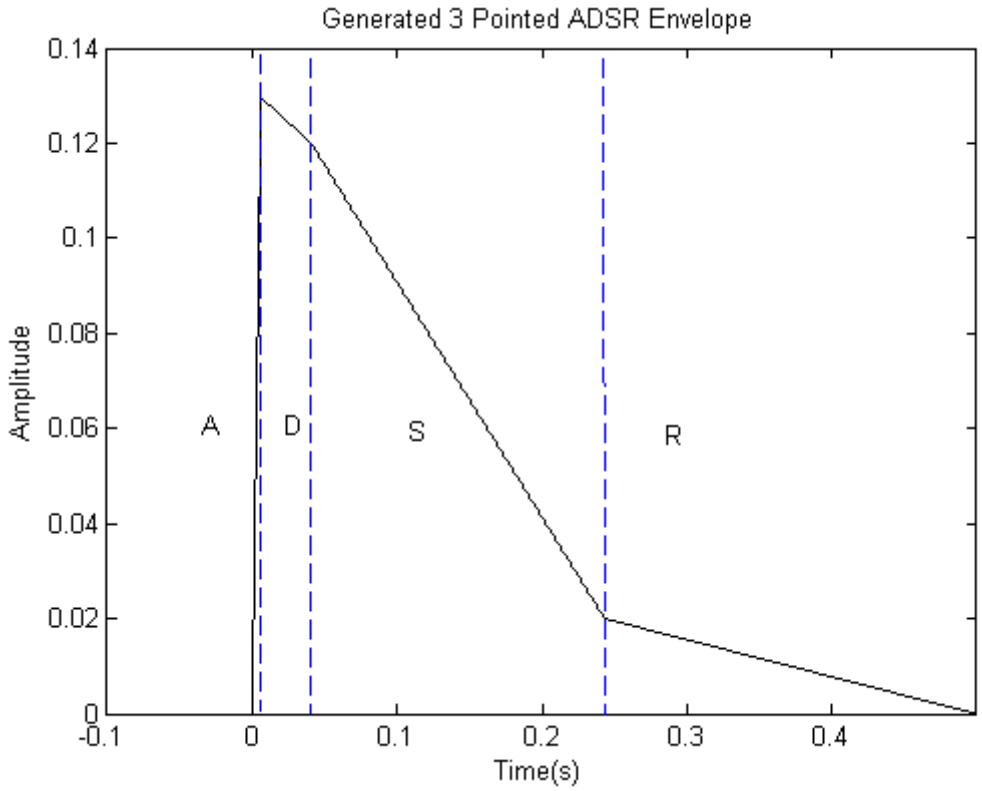


Figure 9: 3-Pointed ADSR Envelope

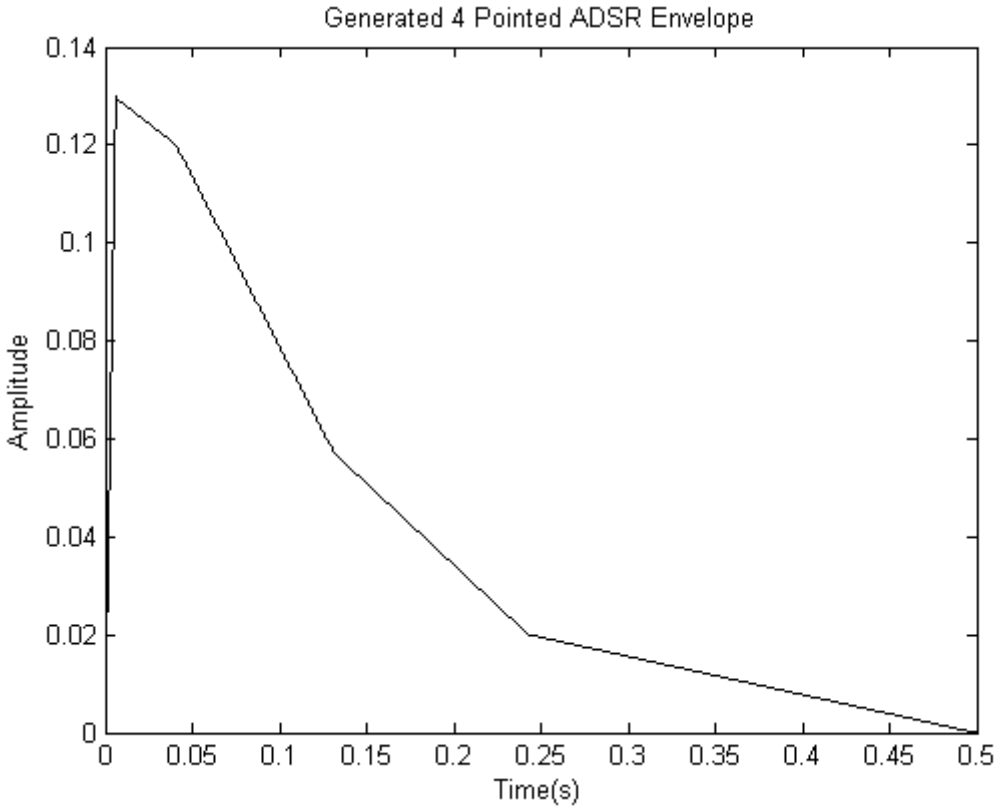


Figure 10: 4-Pointed ADSR Envelope

4.4. Conclusions

In this chapter, the MATLAB implementation of the composer algorithm and the virtual instrument was described. MATLAB has proven to be a good choice for development, the easy implementation of mathematical functions was useful in investigating different fitness functions, while the signal processing capabilities helped with the virtual instrument.

When humans compose music, they have the chance to edit, modify, revise or even dismiss the pieces they have created. The genetic programming gives that ability to an otherwise randomly working composition system. With the fitness function, the system is able to understand what is good or bad, and modify the results with the operations like crossover. Yet, this only shows how important the role of the fitness function in the algorithm, thus the software can only benefit from the improvements in the fitness function. The combination of mean and variance serves as a good beginning for the fitness function; however, after listening to the results for extended amount of times, one can easily understand the style of the composer. Even though this is the case also with most human composers, the initial aim was to create a more creative system. The fitness function can be modified by adding other parameters or one could use an evolving fitness function that would change over time by either inputs from the user or internal inputs.

The limitation of the available notes by using scales works both ways. Certain scales, like the major scale the produced music seemed uncorrelated, while using more characteristic scales result in better sounding pieces with the feel of the scale. Preliminary tests have shown that among the scales that are implemented the Hungarian gypsy, minor blues and the Spanish scales gave the better results.

The built-in random generator successfully serves the purposes of this work, thus there were no reasons for implementing a real or more secure random generator.

The construction of the virtual instrument via an ADSR envelope and a periodic waveform proved to be a successful method while maintaining the necessary information to create the signal as small as possible. The ADSR envelop requires the percentages of the three corner points in time and in amplitude, while for the periodic waveform the relative values of the frequency and the amplitude of the harmonics are sufficient. For the simple symmetric periodic wave, the total number of variables necessary for creation, including the envelope, is nine.

Finally, there is always room for improvements. The possible abilities that were investigated but not implemented in this work are:

- **Polyphonic Composition:** Modifying the algorithm for polyphonic music will basically involve more music theory. Multi-instrument music such as one that has lead, bass, an accompanying chords or arpeggios and percussions is though to increase the performance dramatically. When creating such music, in addition to the harmony of the sequential notes of each instrument, the algorithm should also consider the harmony between the instruments at a timestamp and the rhythmical components. In addition, with the bass, the chords, or the arpeggios a progression, which that will give the song a more structured movement, can be added to the music. On the instrument side, it will mean designing more instruments with different frequency ranges.
- **Interaction with the User:** As seen in Chapter 2, software has been reported where genetic programming is employed for composition, where due to the lack of an optimal solution, human response is introduced as the fitness function. In this project, during the development of the algorithm, a feedback from the user is also designed but not implemented. The feedback would be such that, upon the response from the user, the feedback function transfers the surviving individuals to the next run. Even though, the performance might benefit from the iteration, it is also possible that the previous survivors can dominate the next runs.
- **More Psychoacoustics:** The pleasantness [51] of the sounds can be increased with further study. Yet one must addition should be the use of Mel scales [50, 52] or the masking effects [53], which adds the perception of the human ear to the spectral analysis of the signal.
- **More Listening Tests:** Due to the time limit, formal listening tests on the sound performance of the instrument could not be done. Thus, listening tests would greatly help, since the term timbre does not provide much insight [49, 54].

5. Hardware Implementation

In this chapter, I describe the hardware implementation, which includes converting the MATLAB algorithm into a hardware-like algorithm, implementation issues and results. The hardware implementation is done into Xilinx University Program Virtex II pro, which is the most suitable of the available boards due to the existences of an audio coder/decoder (CODEC) and a 2.5" output jack. I have used both Verilog and VHDL while designing the hardware. The pseudo codes can be found in Appendix B.

5.1. Algorithm

The composer algorithm is implemented as a number of circuit blocks, where I have defined each block with their functionality in the algorithm. These blocks are:

- **Population Generation**
- **Fitness Function**
- **Tournament**
- **Crossover**

5.1.1. Population Generation

This block generates a population of 64 individuals, where 64 is the minimum number of individuals the population should have in order to have a sufficient variety in the MATLAB simulations. The nested for loops in the MATLAB code are implemented as nested ifs. The block has three control signals. One of them is an input to the block and activates or resets the block; the other one is an output that signals that the population generation block is sending data to the top module and the final control signal indicates the completion of sending of data, thus triggering the next block. This block, initiated twice in order to create the two sub-sets of population, is seen in Figure 1 of Section 4.2.1.

The random number generator is implemented as a 32-bit long linear-feedback shift register (LFSR). The registers create a series of periodic pseudo- random bits, where the period is $2^{32} - 1$. Even though the basic shift register cannot be used in fields such as cryptology or security, for this application it provides the sufficient pseudo-random number. The advantages of the LFSR topology are that the period is independent of any word length, fast and that it is easy to implement as both software and hardware [55]. One simple improvement of the randomization will be to use LFSR with known values to initialize the second LFSR, which will give the random bits. The structure of the shift register can be seen in Figure 11.

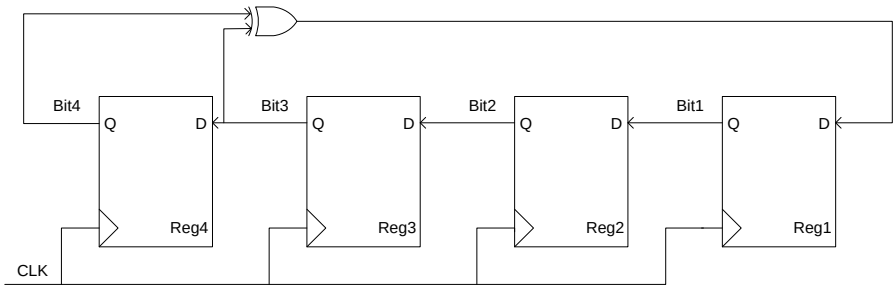


Figure 11: 4-Bit Linear-Feedback Shift-Register

If we assume that the registers are initially charged to "0001", the sequence of pseudo-random bits is as in Table 2.

Index	Bit4	Bit3	Bit2	Bit1
1	0	0	0	1
2	0	0	1	0
3	0	1	0	0
4	1	0	0	1
5	0	0	1	1
6	0	1	1	0
7	1	1	0	1
8	1	0	1	0
9	0	1	0	1
10	1	0	1	1
11	0	1	1	1
12	1	1	1	1
13	1	1	1	0
14	1	1	0	0
15	1	0	0	0
16	0	0	0	1

Table 2: Bits of One Cycle 4-Bit Shift Register

5.1.2. Fitness Function

The fitness function block takes the genes of the individuals, which are in fact the indices of the pitches, as input and after a certain clock period gives out the fitness function of that individual. The processing time is defined by the number of genes an individual has, which is kept at 32 in the hardware implementation.

The fitness function of mean and variance has been adjusted for more accurate hardware implementation. In the initial MATLAB algorithm, the actual values of mean and variance were not important rather their ratios to each other are important. The

calculation of mean and variance of a discrete distribution as seen below requires division.

$$mean = E[x] = \frac{1}{N}\sum_{n=1}^{N} x_n \text{ and } variance = E[x^2] - E[x]^2$$

However, when doing division the accuracy is directly related to the number of bits that will be used for representing the decimal part. Use of eight extra bits will result in accuracy of 0.5^8 which is 0.00390625.Thus instead of using the real values of mean and variance, in the hardware adjusted mean and adjusted variance are used in the fitness function, which are as follows:

$$mean_{adj} = N * \sum_{n=1}^{N} x_n \text{ and } variance_{adj} = N * (\sum_{n=1}^{N} x_n * x_n) - (\sum_{n=1}^{N} x_n)^2$$

Here, again, the sacrifice is the number of bits used however; we can have a better accuracy with less number of bits in this implementation.

This block is run for each individual, which means in a single run it is initiated 128 times. Thus, when the block sends a confirmation signal to the top module it sends out the fitness value of one individual. The block takes a single gene of an individual in each clock cycle. Once the fitness function block has been run 64 times, it is kept off and tournament is triggered.

5.1.3. Tournament

During the hardware implementation of the tournament function, the block did not go through any important changes. Thus, it was implemented as in the MATLAB code, except for the control signals and the finite state machine. A population of 64 can be reduced to two winners in 128 clock cycles. This block is run twice during the creation of a song; with the done signal, the block sends the address of the winner individuals to the top module. The block takes a single fitness value in each clock. The first time the tournament-done signal goes high, it re-initialises the population generation block. While the second time it goes, it high starts the crossover block.

5.1.4. Crossover

The crossover block also includes a designed sorter, which was a simple built-in function in MATLAB. The pseudo-random number generator in Section 5.1.1 is used for the occurrences of the crossover event, where in a clock cycle only a single pitch pair or a gene pair is evaluated for crossover. The crossover rate, which was 33.3% in the MATLAB code, has been changed to 37.5% in the hardware in order to keep the circuitry small. This block is run once, and takes in and sends out a single gene in each

clock. It has two control signals that are sent to the top module. The first one tells that data is being sent while the second one indicates that data send is done triggering the next block.

5.1.5. Rhythm Generator

The rhythm generator reflects the same structure and the functionality as in the MATLAB code, except again the built-in random integer random generator in software is replaced with the LFSR-based pseudo-random number generator. Like population generation, it initially sends out a control signal indicating data transfer. Then, the done signal goes high, indicating the start of the virtual instrument block.

5.2. Virtual Instrument

The virtual instrument block could be implemented in a number of ways. These options can be divided into as static and dynamic. The dynamic way would be to implement the instrument as a function as it is done in the software and the static way would be to load the pre-calculated signal samples into a memory and calling them.

The first option allows more flexibility at the cost of design complexity and more circuitry like multipliers and state machines for controlling the frequency. On the other hand, the second option provides an easier design at the cost of memory. For a sampling frequency equal to 44000 Hz, the required memory is around 2 MB.

Finally, I have decided to pre-calculate the signal and load it into memory, as this allows us to use less logic, keep the sound quality constant and use other virtual instruments that created by recording real instruments. Yet, in this thesis the virtual instruments that has been designed in Chapter 4 is used.

Only one of the scales is implemented in the hardware. The Spanish scale is chosen for demonstration and the deciding factor was the preliminary listening tests.

5.2.1. Audio CODEC

The XUP Virtex-II Pro board is equipped with a National Semiconductor LM4550 Audio CODEC [56]. The compatibility with AC97 allows easy interaction with PCs and MACs, however for this project; the LM4550 is a chip that controls the data sent to the DACs and thus needs to be employed properly. The schematic for the chip can be found as the Appendix D in Section 10.4.

In order to communicate with the LM4550, a controller circuit has been designed. The controller circuit sends the necessary start signals, clock and the data bit in the correct

fashion. Data is sent to the CODEC in packages of 256 bits. The 256 bits comprises control bits, data bits for the four channels of the CODEC and the reserved bits [57]. The CODEC handles 20 bits of data for each output, however; the DACs have 18 bits resolution so only the 18 most significant bits (MSB) of the sent 20 bits are sent to the DACs.

Thus, for each 256 bits sent to the CODEC, only one sample is sent. This gives the maximum sampling frequency possible with the system:

$$fs\_\max = \frac{System\ Clock}{256} = \frac{100\ MHz}{256} = 390625\ Hz$$

Even though, the value is more than the necessary, even the sampling frequency of 44000 Hz could not be implemented due to a software related limitation, where the computer that runs the synthesis tool goes stale due to the size of the read-only memory (ROM). This obstacle is passed by limiting the sampling frequency to 9765.625 Hz, thus limiting the size of the necessary ROM for loading the samples of the instrument.

5.3. Top Module

The top module is the circuitry that provides the interaction between the above blocks while also providing the input from the user to the FPGA. The structure of the whole system can be summarized with Figure 12.

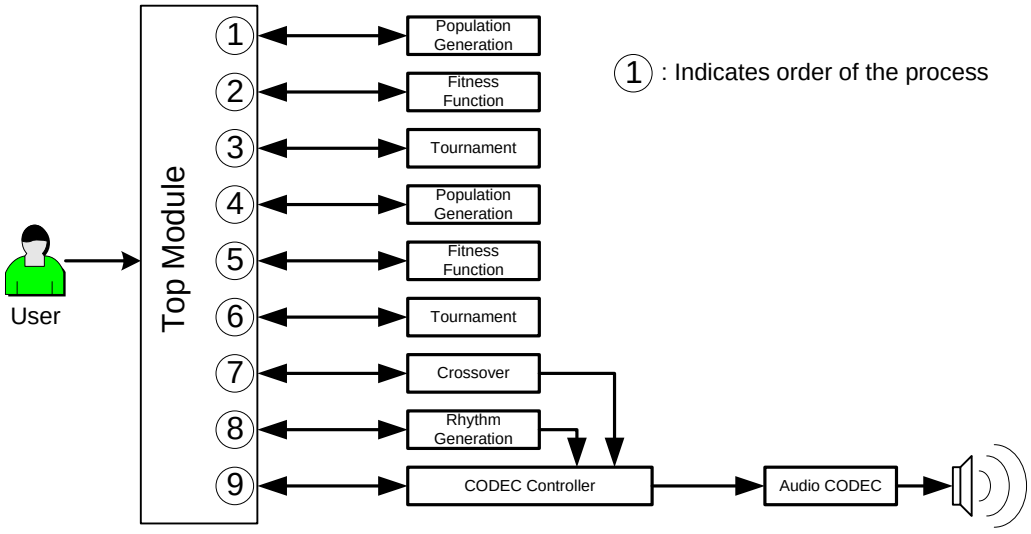


Figure 12: Hardware Top Module

The device is triggered with an input from the user. The top module sends the necessary control signals to the population generation module. Once the population generation sends back the done signal to the top module, fitness function block is triggered. After

receiving the done signal from the fitness function, the top module starts the tournament and waits for the done signal. This process is done twice for each population. Once the winners are decided, the top module enables the crossover and rhythm generator. Finally, upon the completion of their work, CODEC controller is triggered in order to latch the data to the LM4550 chip and the song is heard through the speakers. Finally, the top module not only enables a communication between the blocks but it also stores the necessary information so that they can be recalled later.

5.4. Conclusions

The effect of the population size on the circuitry can be guessed from the software side, still the resulting circuitry due to the population size is larger than the size initially guessed. The LFSR method is generally working sufficiently well, however the pseudo-random number generator can be improved, since there were some individuals whose genes were all 0 in the vsim simulation in ModelSim.

During the transfer of data between the block a semi-serial approach is used. The transferred data package is chosen as a gene or a pitch. When compared to using individual or song pieces as the data package, the used method decreases the wiring size but increases the number of clock periods required for a block to start its function.

To realize the virtual instrument as a ROM loaded with samples had some drawbacks. Due to the size of the ROM being large, the synthesis tool caused the computer to go stale. Thus, in order to reduce the size of the ROM, sampling frequency was reduced to 9.765 KHz. This caused a decrease in the quality of the sound, as the quantization noise became audible. In addition, since the audio codec was using unsigned data, the created signals were shifted to move the negative part to positive. This also caused a change in timbre, further decreasing the quality of the sound. However, all these problems could be solved by using better equipment and employing recorded virtual instruments.

6. Listening Tests

As mentioned before music composition is not a problem where an optimum solution exists, instead the musical taste varies from person to person. In order to fine-tune the algorithm, listening tests are formed where the aim is to finalize the fitness function, mentioned in Section 4.2.4.

6.1. Aim and Properties

The listening test is being applied for fine-tuning the fitness function by applying different coefficients to the mean and variance in the fitness function.

C1((tonic of the scale)-(mean of an individual)) +C2((max variance in the population)-(variance of an individual))

Since humans are the subjects of the test, it should be prepared such that the results of the test answer the right questions. For this reason, literature search in psychoacoustics, which is the science that relates the sound cognition, acoustics, psychology and physiology, were made. The final remarks, deducted from the literature [51, 58] for creating successful listening tests for this work, can be summarized as follows:

- Minimize the errors caused by the test
- Minimize the cultural effects
- Provide proper instructions
- Aim for sensory effects
- Make it reproducible

6.2. Listening Test I

The aim of the first test was to test the test before finalizing it, so that the final test is free from cultural influences**,** wrong expectations, improper questions and biases that may occur due to the test. For this reason, the draft version of the test is applied on ten subjects, where after completing the test they have been questioned to get feedback on the test.

The questions asked were as follows:

- Did the questions in the test ask what the introduction tells? If not, why?
- Was the test too long or too short? Why?
- Was it easy or hard to keep focus on the test? If hard, why?

According to the answers, the test was modified as follows:

- Increase the size of the initial population to provide different fittest elements for all fitness functions.
- Shorten the test by reducing the number of fitness functions tested.

6.3. Listening Test II

The test starts with some background information that describes the aim of the test, how to answer the questions and what the questions include. In addition, a short description of the type of the songs is given so that the subjects can have an idea of what to expect. The introduction is followed by three questions that collect the information regarding the testers. The collected information is the age, familiarity with western music and favourite musical genres. Following the first three data-collecting questions are the real questions, which asks the subjects to test their liking of the song they have listened on a scale of one to ten. An absolute unipolar scale of ten is chosen among other most used methods such as psychophysics, relative methods or sound maps since it gives more freedom for the tester and larger range for analysis and it is a standard [58]. However, the non-existence of a reference might cause a scaling problem depending on the tester. The last question is an optional one that asks the differences between the liked and disliked songs. The songs are presented in random order in order to avoid any biasing that might occur due to the song order. A sample of the test can be found in Appendix C in Section 10.3.

The pieces are created in major scale since it is a widely known and used scale. The aim is to avoid testers' having any special familiarity with certain local scales like Spanish scale or Byzantine scale. The length of the songs was 26 seconds and they had 128 notes, where each part had 32 notes.

Table 3 shows the tested fitness-function coefficients in the final test. The coefficients are chosen in order to provide a variety of fittest individuals and their efficiency is simulated in MATLAB.

Weigh of Mean (C1)	Weigh of Variance (C2)
1	1
4	1
1	4
1	-1
2	-1
4	-1
1	0
0	1

Table 3: Tested Fitness Function Coefficients

At the process of testing, the virtual instruments were not complete, thus a simpler instrument consisting of sinus signal as the waveform and half-period sinus signal as the envelope is used. An example waveform can be seen in Figure 13.

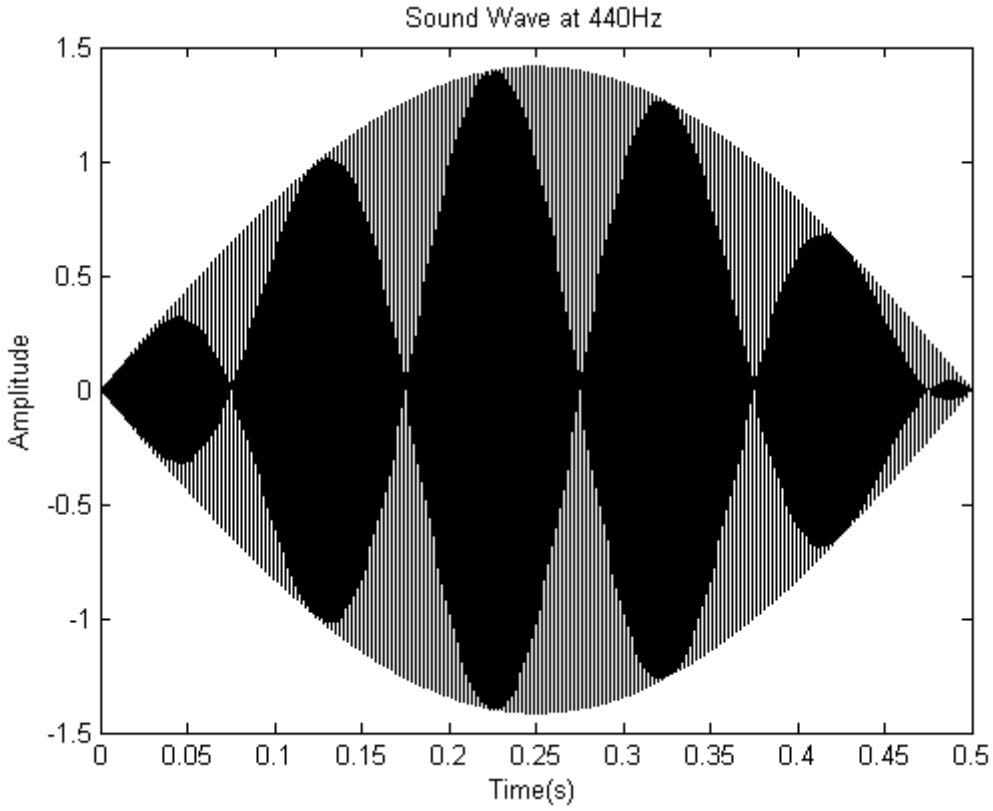


Figure 13: Sound Wave of the Testing Instrument

6.4. Results and Conclusions of Listening Test II

The results of listening test II can be seen in the Figures 14 and 15. Figure 14 shows the results as they are while in Figure 15 shows the scaled results. Both figures show that the 50 participants of the listening test have chosen the fitness function coefficients to be C1=1 and C2=0, which means the function will only be including the mean.

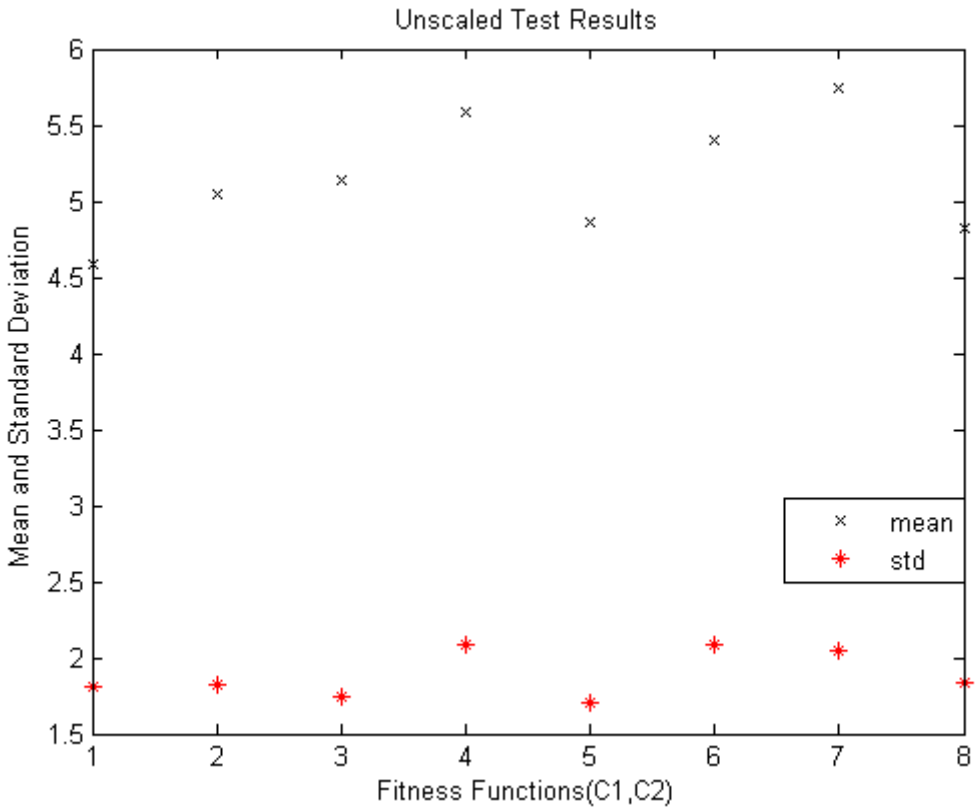


Figure 14: Original Listening Test Results

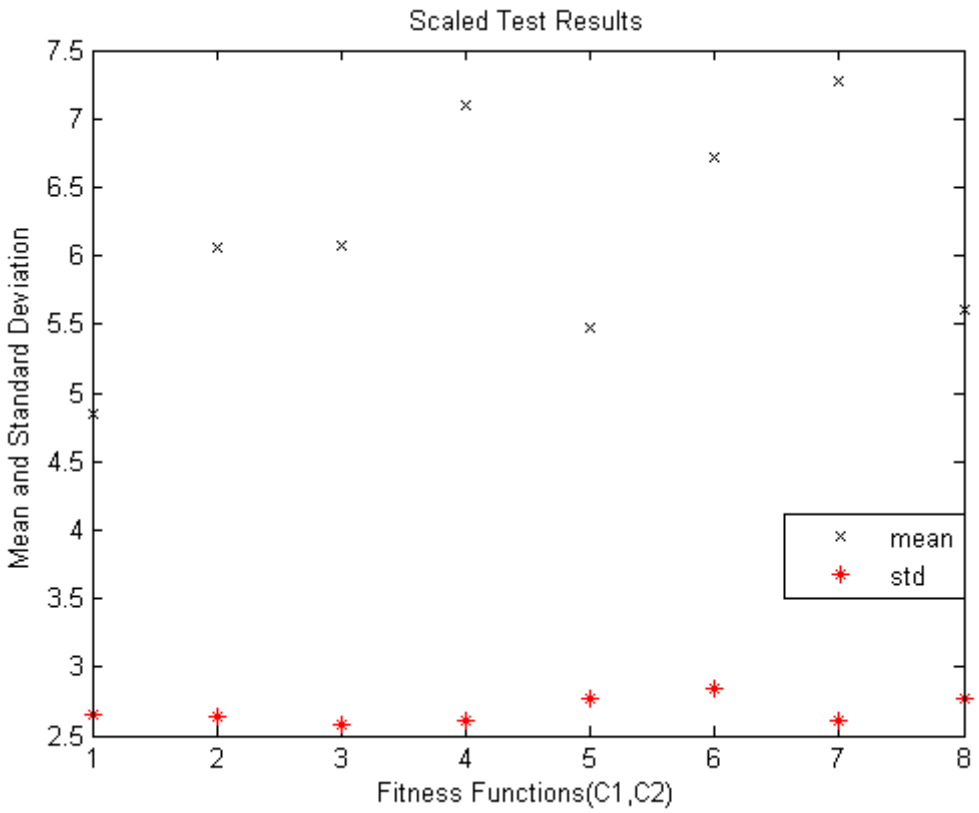


Figure 15: Scaled Listening Test Results

The participants had an age distribution with mean equal to 28.84 and standard deviation equal to 17.2. Finally, answers to the Question 12 of the test shows that many listeners did not like the songs when the rhythm gets faster. In addition, one other important complaint was about the large leaps caused by the crossover operation; this can also explain the choice for the mean only as removing variance eliminates the individuals that wander a lot. This means statistically smaller leaps even after crossover. Moreover, the same comments also show that the parts with the fast rhythm should have the smaller movement, thus at least the fast rhythm should not be used with the parts that are created via crossover, and instead an original individual should be used. The full set of answers can be found in Appendix C.

7. Results

In this chapter, the results of the MATLAB and hardware description language (HDL) implementation is evaluated. The evaluation is done on the compositional performance of the implementations and the synthesis results of the hardware.

7.1. Compositional Performance

In the figures below the results of a random MATLAB run and a random vsim run can be seen. In Figure 16.a the fittest individual, which is the theme of the song is seen. The song part in Figure 16.b is the generated via crossover between the theme and the second fittest individual. The melody in Figure 16.c is the child of the song part in Figure 16.b and the third fittest individual. Finally, the child of the fittest and the fourth fittest individual is seen in Figure 16.d as the last part of the song.

Figure 16.a: MATLAB Run Part 1

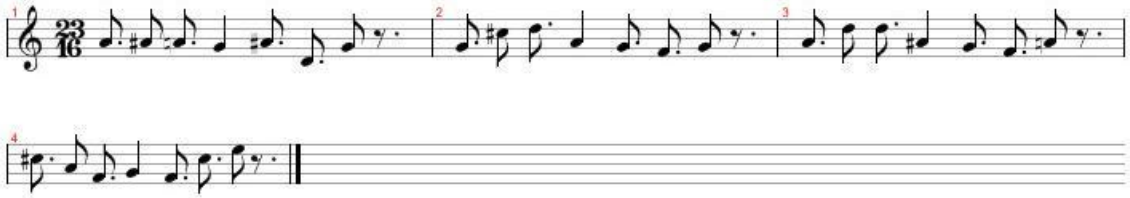


Figure 16.b: MATLAB Run Part 2

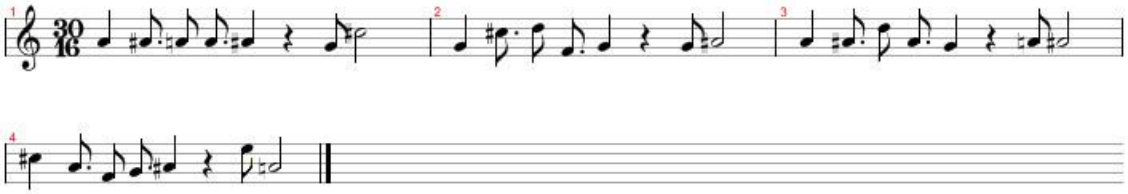


Figure 16.c: MATLAB Run Part 3

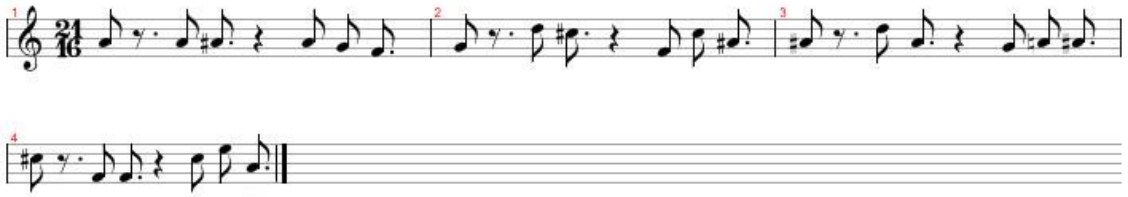


Figure 16.d: MATLAB Run Part 4

The song in Figure 17 is created in vsim of ModelSim with the code implemented in VHDL.

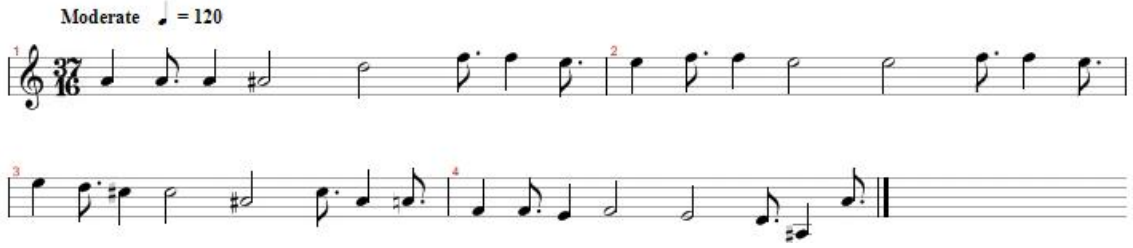


Figure 17.a: vsim Run Part 1

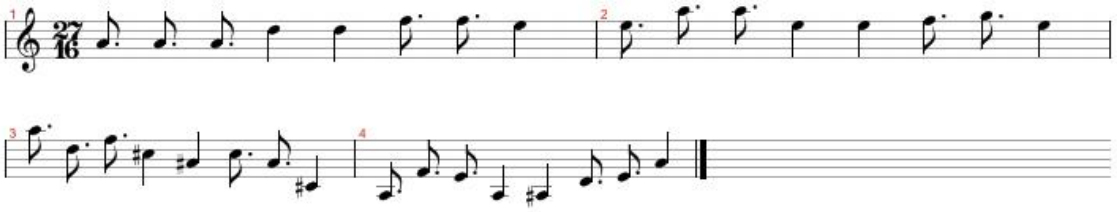


Figure 17.b: vsim Run Part 2

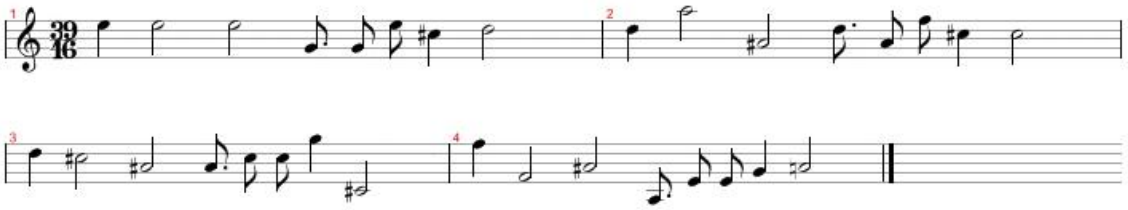


Figure 17.c: vsim Run Part 2

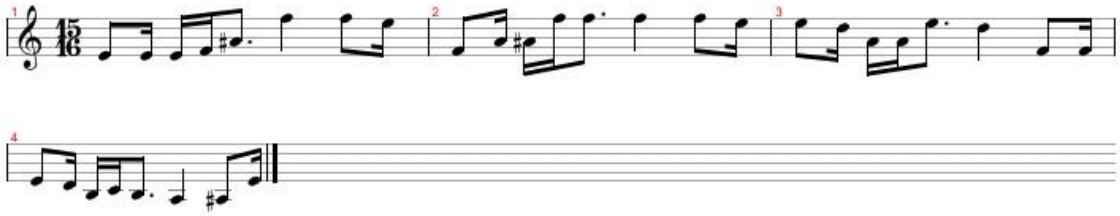


Figure 17.d: vsim Run Part 3

7.2. Circuit Based Results

The FPGA with its 100-MHz clock is able to compose a song in 97.29 µs. The generation of the song requires 9729 clock cycles after the device is activated. The population generation block needs 2049 clock cycles and it is summoned twice. The fitness function block consumes 36 clock cycles but it is used 128 times for the "64 × 2" implementation. The tournament block takes 128 clock cycles and it is used twice. The crossover block consumes 358 clock cycles, and finally the rhythm generator block uses 32 clock cycles.

However, even with the reduction of the size of the ROM mentioned in Section 5.4. , the designed hardware did not fit into the target FPGA due to routing problems and

memory size. In order to remove circuitry, the overall population size has been decreased from "64 × 2" to "32 × 2", allowing a successful place and routing.

The circuit inventory of the both designs can be seen in Table 4. In Table 5, the utilization ratio of the FPGA can be observed. Tables 4 and 5 do not only provide an insight in the size of the circuitry but they also show the change in circuit size with respect to the population size

Item Type	HDL Synthesis (32x2)	HDL Synthesis (64x2)	Advanced HDL Synthesis (32x2)	Advanced HDL Synthesis (64x2)
Random-Access Memory (RAM)	5	4	10	10
ROM	6	5	1	1
Adder/Subtractor	35	36	35	36
Counter	20	20	20	20
Register	1896	2983	7871	12684
Comparator	115	116	115	116
Multiplexer	79	81	85	87
Exclusive Or (Xor)	3	3	3	3

Table 4: Circuit Inventory

Item Type	Amount/Available (32x2)	Amount/Available (64x2)	Ratio (32x2)	Ratio (64x2)
Slices	85885 out of 13696	14335 out of 13696	62%	104%
Slice Flip Flops	7795 out of 27392	12586 out of 27392	28%	45%
4-Input Look-up Table (LUT)	14874 out of 27392	23700 out of 27392	54%	86%
Input/Output (IO)	11	11	-	-
Input/Output Block (IOB)	10 out of 556	10 out of 556	1%	1%
Block Random-Access Memory (BRAM)	134 out of 136	135 out of 136	98%	99%
Global Clock Buffer (GCLK)	4 out of 16	4 out of 16	25%	25%
Digital Clock Manager (DCM)	1 out of 8	1 out of 8	12%	12%

Table 5: FPGA Utilization

8. Conclusions

In this thesis, a FPGA implementation of a music composing and playing circuit is presented. The current performance of the FPGA implementation is comparable to the MATLAB implementation composition wise. A successful FPGA implementation shows that it is possible to build a personal audio player that can compose and play its own music. However, in order to come up with a product that can be used daily, the design needs some improvements.

On the composition side:

- Today's music is hardly monophonic, even when the performance includes only a single instrument, generally the music has more than one sound at the same time. Thus, the algorithm should be able to compose and arrange polyphonic music.
- The methods for creation and survival of the individuals can be improved. Currently, individuals are created from a limited gene pool, which are the scales. Black notes, scale changes can be added. The survival method or the artificial intelligence of the system can be improved in many ways. Mathematical concepts like the Hurst exponent or musical concepts like tonality can be added to the current fitness function. Other possible changes can be the employing an interaction between the user and the composer, allowing the user to be a part of the fitness function or giving the composer a short-term memory so that a number of individuals from the previous composition can be added to the population of the next composition.
- Currently, the songs have four individuals, where each individual has a different rhythm where the rhythm is generated as a progression. However, the rhythms are constant throughout the respective individuals; this causes a sense repetition rather than auditory resemblance and can be improved. In addition, the rhythm should be further adjusted with respect to the listening tests results. Large leaps due to crossover combined with fast notes result in a dislike for most of the testers.
- The listening test results show that the usage of the fast rhythm and child individuals cause disliked songs, where the sequential change in frequency is high while the duration of a note is short. Separating them may give better results; one other option could be using different movement limitations for different speed of rhythms.
- Finally, as stated initially, the composer only composes the pitch and the duration of the notes. Thus, an improved version should have the ability to adjust dynamics, playing style and other concepts that are included in a musical composition

On the instrument side:

- As depicted in Chapter 3, the virtual instruments that are created from sampling real life instruments are better than the instruments that are created functionally with the cost of memory. This also can be used in this work, with sufficient memory, the composer can be provided with a number of professionally used virtual instruments.

With more time, the MATLAB implementation can only get better and the improvements' reflection on the hardware side can be summarized as follows. The initial problem might be the required amount of memory, as the composer has more abilities and freedom such more pitches and dynamics, it will need to utilize more detailed virtual instruments that need to include every combination of pitch and dynamics. Today, sampled virtual instruments can be found to consume 10 GB of memory [59] and this could be a problem. For the FPGA implementation, the memory problem can be solved by using the external onboard RAM or the compact flash (CF) card slot. The external RAM can be used to store internal signals while the non-volatile CF card can be used to provide virtual instrument samples.

The increase in the complexity of the composer will result in more hardware, which causes an increase in the area of the device, price of the device, computational time and the power dissipation. Here the power consumption is the most critical, since the ideal aim is to create a mobile device. Yet, it has been seen that the size of the population has a great effect on the size of the circuitry due to the amount of required storage elements.

However, as a final note, it must be said that even if an ideal device that can replace human musicians is created, it might not be successful since many people might prefer music that humans create.

9. References

[1] V. Menon and D.J. Levitin, "The rewards of music listening: response and physiological connectivity of the mesolimbic system." *NeuroImage*, vol. 28, pp. 175-84, 2005.
[2] I. Cross, "Music, cognition, culture, and evolution," *Ann. N. Y. Acad. Sci.*, vol. 930, pp. 28-42, 2001.
[3] T. Lesiuk, "The effect of music listening on work performance," *Psychology of Music*, vol. 33, pp. 173-191, 2005.
[4] A. Wells and H. Tokinoya, "The genre preferences of western popular music by Japanese adolescents," *Popular Music and* Society, vol. 22, pp. 41-53, 1998.
[5] P.N. Juslin, S. Liljeström, D. Västfjäll, G. Barradas, and A. Silva, "An experience sampling study of emotional reactions to music: listener, music, and situation," *Emotion*, vol. 8, pp. 668-83, 2008.
[6] P.N. Juslin and D. Västfjäll, "Emotional responses to music: the need to consider underlying mechanisms," *The Behavioral and Brain Sciences*, vol. 31, pp. 559-621, 2008.
[7] S. Saarikallio, "Music as mood regulation in adolescence," Ph.D. dissertation, University of Jyväskylä, Jyväskylä, Finland, 2007.
[8] R. E. Milliman, "Using background music to affect the behaviour of supermarket shoppers," *The Journal of Marketing*, vol. 46, No. 3, pp. 86-91, 1982.
[9] R.E. Milliman, "The influence of background music on the behavior of restaurant patrons," *Russell The Journal Of The Bertrand Russell Archives*, vol. 13, pp. 286-289, 1986.
[10] H. Ragneskog, G. Bråne, I. Karlsson and M. Kihlgren, "Influence of dinner music on food intake and symptoms common in dementia," *Scandinavian Kournal of Caring Sciences*, vol. 10, pp. 11-47, 2010.
[11] A Moroni, J. Manzolli, F. Von Zuben and R. Gudwin, "Vox Populi: an interactive evolutionary system for algorithmic music composition," *Leonardo Music Journal*, vol. 10, pp. 49-54, 2000.
[12] J. A Biles, "GenJam: a genetic algorithm for generating jazz solos," in *International Computer Music Conference*, 1994.
[13] M. Supper, "A few remarks on algorithmic composition," *Computer Music Journal*, vol. 25, no. 1, pp. 48-53, Spring 2001.
[14] M. Simoni, *Algorithmic Composition: a Gentle Introduction to Music Composition Using Common LISP and Common Music*, The Scholarly Publishing Office, The University of Michigan, 2003.
[15] J. A. Maurer, "History of Algorithmic Composition," March 1999. [Online]. Available: https://ccrma.stanford.edu/~blackrse/algorithm.html [Accessed: Jan. 2, 2010].

[16] J. Chuang, “Mozart’s Msikalisches Würfelspiel”, *SunSite Austria at the University of Vienna,* 1995. [Online]. Available: http://sunsite .univie.ac.at/Mozart/dice/#options. [Accessed: Jan. 5, 2010].
[17] L. A. Hiller and L. M. Isaacson, “Music composition with a high-speed digital computer,” *Journal of the Audio Engineering Society*, vol. 6 Issue 3 pp. 154-160; July 1958.
[18] K. Ebcioglu, “An expert system for chorale harmonization,” *Association for the Advancement of the Artificial Intelligence*, 1986.
[19] P.M. Todd and G. M. Werner, “Frankensteinian methods for evolutionary music composition,” in *Musical Networks: Parallel Distributed Perception and Performance*, 1998.
[20] J. McCormack, “Grammar based nusic composition,” in *Complex Systems 96: From Local Interactions to Global Phenomena*, 1996 pp. 320-336.
[21] A. Alpern, “Techniques for algorithmic composition of music,” 1995 [Online]. Available: Computer and Information Science CiteSeer Publications, http://citeseer.ist.psu.edu. [Accessed: Jan. 2, 2010].
[22] Voss, R.F. and J. Clarke, “1/f Noise in Music and Speech,” *Nature*. Vol. 258 pp. 317-318, 1975.
[23] J. A. Maurer, “The Influence of Chaos on Computer-Generated Music,” March 1999. [Online]. Available: https://ccrma.stanford.edu/ ~blackrse/chaos.html [Accessed: Jan. 2, 2010].
[24] D. Cope, “Computer modelling of musical intelligence in EMI,” *Computer Music Journal*, vol. 16, no. 2, pp. 69-83, 1992.
[25] R. L. de Mantaras and J. L. Arcos, “AI and music from composition to expressive performance,” *AI Magazine*, Volume 23, No. 3, pp. 43-57, 2002]
[26] A.R. Brown, “Opportunities for evolutionary music composition,” In *Australasian Computer Music Conference*, 2002, pp. 27-34.
[27] D. Tzimeas and E. Mangina, “Dynamic techniques for genetic algorithm-based music systems,” *Computer Music* Journal, vol. 33, no. 3, pp. 45-60, 2009
[28] Y.-P. Chen, “Interactive music composition with evolutionary computation,” *NCLab Report*, January 2007.
[29] N. Tokui and H. Iba, “Music composition with interactive evolutionary computation,” in *GA2000, third International Conference on Generative Art*, 2000.
[30] Z. W. Geem and J.-Y. Choi, “Music composition using harmony search algorithm,” in *EvoWorkshops* 2007, pp. 593-600.
[31] A. O. de la Puente, R. S. Alfonso and M. A. Moreno, “Automatic composition of music by means of grammatical evolution,” in *APL’2002* Madrid,2002 pp. 148-155.
[32] B. Manaris, P. Roos, P. Machado, D. Krehbiel, L. Pellicoro and J. Romero, “A corpus-based hybrid approach to music analysis and composition,” in *22nd National Conference on Artificial Intelligence*,2007,pp. 839-845.

[33] N.-W. Gong, M. Laibowitz and J. A. Paradiso, "MusicGrip: A writing instrument for music control," in *Proceedings of the 9th International Conference New Interfaces for Musical Expression*, 2009.
[34] N. Böttcher and S. Dimitrov, "An early prototype of the augmented PsychoPhone," in *Proceedings of the 9th International Conference New Interfaces for Musical Expression*, 2009.
[35] F. Bevilacqua, L. Naugle and I. Valverde. "Virtual dance and music environment using motion capture," *Proceeding of the IEE Multimedia Technology and Applications Conference*, Irvine, 2001.
[36] F. Avanzini, G. Borin, G.D. Poli, F. Fontana, and D. Rocchesso, "For piano sound synthesis: a research overview," *EURASIP Journal on Applied Signal Processing*, pp. 941-952, 2003.
[37] S. Bilbao and J. Fitch, "Prepared piano sound synthesis," *Proceedings of the 9th International Conference on Digital Audio Effects*, Montreal, Canada, September 2006.
[38] C.E. Gough, "Measurement, modelling and synthesis of violin vibrato sounds," *Acta Acustica United With Acustica*, vol. 91, pp. 229 - 240, 2005.
[39] M. Karjalainen, V. Välimäki, and Z. Jánosy, "Towards high-quality sound synthesis of the guitar and string instruments," *International Computer Music Conference*, Tokyo, Japan, 1993.
[40] M. Laurson, C. Erkut, V. Välimäki, and M. Kuuskankare, "Methods for modeling realistic playing in acoustic guitar synthesis," *Computer Music Journal*, vol. 23, no. 8, pp 38-49, 2001.
[41] M. Karjalainen and U. K. Laine. "A model for real-time sound synthesis of guitar on a floating-point signal processor," in *Proc. IEEE Int. Conf. on Acoustics, Speech, and Signal Processing (ICASSP'91)*, 1991, pp. 3653-3656.
[42] V. Välimäki and T. Takala. "Virtual musical instruments – natural sound using physical models," *Organised Sound*, vol. 1, no.2, pp. 75-86, 1996.
[43] W. Banzhaf, P. Nordin, R. E. Keller and F. D. Francone, *Genetic Programming ~ An Introduction: On the Automatic Evolution of Computer Programs and Its Applications*, San Francisco: Morgan Kaufmann, 1998.
[44] B. H. Suits, "Scales: Just vs Equal Temperament," 1998. [Online] Available: http://www.phy.mtu.edu/~suits/scales.html. [Accessed: Feb. 13, 2010].
[45] B. Gieben and S. Hall, Ed., *Formations of Modernity: Introduction to Sociology*. Oxford: Blackwell Publishers, 2001.
[46] H. Niederreiter, "Quasi-Monte Carlo methods and pseudo-random number," *Bulletin of the American Mathematical Society*, Vol. 85, No. 6, November 1978.
[47] R. Parncutt, *Harmony: A Psychoacoustical Approach*, New York: Springer, 1989.
[48] K. Jensen, "Envelope model of isolated musical sounds," in *Proceedings of the 2nd COST G-6 Workshop on Digital Audio Effects*, 1999.
[49] K. Jensen, "Timbre models of musical sounds," Ph.D. dissertation, University of Copenhagen, Copenhagen, Denmark, 1999.

[50] X. Rodet, “Musical sound signal analysis/synthesis: sinusoidal + residual and elementary waveform models,” in *Proceedings of the IEEE Time-Frequency and Time-Scale* Workshop, 1997.
[51] H. Fastl and E. Zwicker, *Psychoacoustics: Facts and Models*, 3rd Edition, New York: Springer, 2007.
[52] M. Helen and F. Tampere, “Perceptually motivated parametric representation for harmonic sounds for data compression,” *Proceedings of the International Conference on Digital Audio Effects*, 2003.
[53] B. C. J. Moore, *an Introduction to the Psychology of Hearing*, 4th Edition, San Diego: Academic Press, 2000.
[54] American Standard Association, *Acoustical Terminology*, New York, 1960.
[55] T. G. Lewis and W. H. Payne, “Generalized feedback shift register pseudorandom number algorithm,” *Journal of the Association for Computing* Machinery, Vol. 20, No. 3, pp 456-468, 1973.
[56] Xilinx, *Xilinx University Program Virtex-II Pro Development System: Hardware Reference Manual*, 2005.
[57] National Semiconductor, *LM4550: AC’97 Rev 2.1 Multi-Channel Audio Codec with Stereo Headphone Amplifier, Sample Rate Conversion and National 3D Sound*, 2004.
[58] F.E. Toole, “Listening tests – turning opinion into fact,” *Journal of Audio Engineering Society*, Vol. 30, No. 6, pp. 431- 445, 1982.
[59] Steinberg Media Technologies, *The Grand 3 Product Data Sheet EN*, 2009.

Appendix A – MATLAB Pseudo Codes

Main algorithm

```
constant declerations


%%pitch generation

for all sub-populations
    for for all individuals
        for for all genes except 1st and last
            if first or last
                        gene=9
                    elseif previous gene is at the ends
                        fix out of bounds conditions
            elseif
                        gene=previous_gene+movement
                    end;
        end;
    end;
end;


%%rhythm generation
for all song parts
    for 1 to 8
            if (song part)
               rhythm = random (song part)
       if (verse==1)
           rhytm(verse,n) = randi(4)
       end
    end
end


%%fitness function
for all sub-populations
    for all individuals
        fitness=mean(individual)
    end
end


%%tournament
for all sub-populations
    if(comparison of fitness function)
            last2_1= individual
            last2_2= individual
    end
end

%%crossovers
sort(survivors)
1st=individual
2nd=individual
```

```
3rd=individual
4th=individual

composition_1st_part=1st
if(crossover)
   composition_2nd_part=2nd
else
   composition_2nd_part=1st
end
if(crossover)
   composition_3rd_part=3rd
else
   composition_3rd_part=composition_2nd_part
end
if(crossover)
   composition_4th_part=4th
else
   composition_4th_part=1st
end


composition=[composition_1st_part,composition_2nd_part,composition_3rd
_part,composition_4th_part]

for all genes in composition
    song=cat(song,virtualinstrument(song(),rhythm(i)))
end
```

Appendix B – HDL Pseudo Codes

VHDL Entity Definitions of Blocks

```
ENTITY top IS
port (Clk : in std_logic;
      start : in std_logic;
      ac97_bit_clock : in std_logic;
      led1 : out std_logic;
      led2 : out std_logic;
      led3 : out std_logic;
      led4 : out std_logic;
      ac97_synch : out std_logic;
      ac97_sdata_out : out std_logic;
      audio_reset_top : out std_logic);
END top;

ENTITY populationgeneration is
port ( popgen_start : in Std_Logic;
       Clk : in Std_Logic;
       pitch : out Std_Logic_Vector (3 downto 0);
       pitch_out : out std_logic;
       popgendone : out std_logic);
END populationgeneration;

ENTITY fitnessfunction is
port ( Reset : in Std_Logic;
       clk : in Std_Logic;
       Start : in Std_Logic;
       pitch_in : in Std_Logic_Vector (3 downto 0);
       done :out Std_Logic;
       fitness : out Std_Logic_Vector (15 downto 0));
END fitnessfunction;

ENTITY tournament is
port ( Reset : in Std_Logic;
       clk : in Std_Logic;
       Start : in Std_Logic;
       Fitness_in : in Std_Logic_Vector (15 downto 0);
       address1 : out Std_Logic_Vector (5 downto 0);
       address2 : out Std_Logic_Vector (5 downto 0);
       done :out Std_Logic);
END tournament;

ENTITY crossover is
port ( Reset : in Std_Logic;
       clk : in Std_Logic;
       Start : in Std_Logic;
       pitch_in : in std_logic_vector (3 downto 0);
       ff_in : in Std_Logic_Vector (15 downto 0);
       pitch_out: out std_logic_vector (3 downto 0);
       crossover_out: out std_logic;
       done:out Std_Logic);
END crossover;
```

```
ENTITY rhythmer is
port ( reset : in Std_Logic;
       start : in Std_Logic;
       Clk : in Std_Logic;
       rhythm : out Std_Logic_Vector (3 downto 0);
       rhythm_out,rhythm_done : out std_logic);
END rhythmer;


ENTITY audio_top
PORT ( clk : in std_logic;
       pitch_in : in std_logic_vector(3 downto 0);
       rhythm_in : in std_logic_vector(3 downto 0);
       audiostart : in std_logic;
       ac97_bit_clock : in std_logic;
       ac97_sdata_out: out std_logic;
       ac97_synch: out std_logic;
       audio_reset_b: out std_logic;
       done: out std_logic;
       slow_clock: out std_logic);
END audio_top;
```

Appendix C – Listening Test and Complete Results

Listening Test

Algorithmic Composition

In the test, you will listen a number of songs generated by an algorithm. You will be asked to rate your likings of the songs. The answers will be used for fine-tuning the algorithm.

The songs you will listen are monophonic. When they are composed, only rhythm, theme and exploration of the theme are taken into account. Certain parts of the songs can be similar as different algorithms may end up with similar song parts so there are not any trick songs or questions.

The data collected in questions 1, 2 and 3 will be used to understand the results of the following questions.

For questions 4 to 11, click on the above play button to listen a song and rate your liking on a scale of 10, where 10 means most liked and 1 means least liked. There is no right or wrong answer.

Question 12 is an optional free association question, where you can enter your thoughts freely in the text box.

*** 1.** Are you familiar with western music (any song that is created with chromatic scale: notes (C-D-E-F-G-A-B or do-re-mi-fa-sol-la)?

○ Yes ○ No

*** 2.** Enter your age. _____

*** 3.** Write your favorite music genre(s). _____

*** 4.** Rate your liking of the song ?

○ 1 ○ 2 ○ 3 ○ 4 ○ 5 ○ 6 ○ 7 ○ 8 ○ 9 ○ 10

*** 5.** Rate your liking of the song ?

○ 1 ○ 2 ○ 3 ○ 4 ○ 5 ○ 6 ○ 7 ○ 8 ○ 9 ○ 10

*** 6.** Rate your liking of the song ?

○ 1 ○ 2 ○ 3 ○ 4 ○ 5 ○ 6 ○ 7 ○ 8 ○ 9 ○ 10

*** 7.** Rate your liking of the song ?

○ 1 ○ 2 ○ 3 ○ 4 ○ 5 ○ 6 ○ 7 ○ 8 ○ 9 ○ 10

*** 8.** Rate your liking of the song ?

1 2 3 4 5 6 7 8 9 10

*** 9.** Rate your liking of the song ?

1 2 3 4 5 6 7 8 9 10

*** 10.** Rate your liking of the song ?

1 2 3 4 5 6 7 8 9 10

*** 11.** Rate your liking of the song ?

1 2 3 4 5 6 7 8 9 10

12. (Optional) Shortly describe the differences between the most and least liked songs?

***** Indicates Response Required

Full Answers for Question 3

- Rock N Roll, metal
- Tango, latin
- metal rock
- rock
- hip-hop
- rock
- Americana, Roots
- All
- Excellence
- classical
- indie/progressive rock, progressive metal, baroque, romantic, folk
- Rock n'Roll
- orijinal eastern music
- Pop music
- New age
- Pop
- jazz
- classical
- folk, new age, world music
- Rock
- Rock
- r & b

- jazz
- metal, classic, folk
- country
- alternative, British, progressive
- rock
- Metal
- classical
- funk, indie rock, blues, jazz, pop
- classical
- Rock, Heavy Metal
- Rock
- New age
- Pop., soft rock, jazz, classical
- dance, blues, ethnics
- Smooth jazz
- metal
- Classic, Alternative Rock, Grunge, Power Metal
- Nirvana
- Classical, Ethnic
- metal
- rock, blues, country
- Popular Music, R&B, Soul, Classical Music
- Jazz
- Classical
- Metal, techno
- pop

Full Answers for Question 12

- Clarity
- I disliked all songs, mainly because of the lack of context needed to support the largely shapeless melodies. Those I liked better seemed to have a bit more structure and direction.
- They all have too many notes
- They sound like sad lullabies
- Actually, they are basically the same for me.
- No. 9 is more like song. No 5 I feel a lot noise in the song.
- I feel that way, and least liked songs seemed to be more chaotic, which is trying to imply that I am chaotic, I am different, but when something strongly implies something, it is generally from reason, that it does not work that way.
- I like when you play all of them at the same time! Why do they start with the same note? I enjoy the slower tunes more.

- in number four, the melody is not discrete, on the other hand in number 5 after lowering the volume the fast part at the end does not match with the start
- there are some annoying intense transitions in my least liked song, the music is very smooth in my most liked song
- they lack tension and resolution
- fro bad ones the rhythm was repeating itself too much in the beginning while th end of the song was too fast leasding tio sounds that sounded unrealistic.I liked the 9 th the most due to its variety in the beginning but ending is still problematic.
- seems not only some parts but the complete theme is the same for each song. nevertheless I believe 6 contains the scale which best fits the melody.
- Songs should be softer, also they can be happier but that may be caused by the flute like sound
- The sense of rhythm/beat - is felt in those songs where it is clear. And then it gets very confusing towards the end of the track. If it's not really meant to have any set beat, then I have no comment.
- In my view, for song number 11, the transitions between the notes are smoother. I did not hear sudden changes. For the songs I rated with 5, I heard sudden transitions that made the songs hard (or less interesting) to listen to.
- Most liked: less random, Least liked: like R2-D2 singing!
- non-random progressing makes it sound more like real music, instead of random noise
- not cool

Full Answers for the Fitness Functions

	FF1	FF2	FF3	FF4	FF5	FF6	FF7	FF8
Tester 1	*3*	*4*	*3*	*5*	*3*	*4*	*3*	*4*
Tester 2	*7*	*3*	*6*	*5*	*7*	*5*	*7*	*3*
Tester 3	*5*	*3*	*6*	*2*	*8*	*5*	*4*	*5*
Tester 4	*4*	*5*	*5*	*6*	*6*	*6*	*3*	*7*
Tester 5	*3*	*4*	*4*	*5*	*4*	*4*	*4*	*5*
Tester 6	*3*	*4*	*4*	*5*	*5*	*5*	*4*	*4*
Tester 7	*4*	*5*	*4*	*6*	*4*	*4*	*6*	*5*
Tester 8	*3*	*3*	*4*	*3*	*4*	*3*	*4*	*4*
Tester 9	*2*	*3*	*3*	*4*	*5*	*5*	*6*	*4*
Tester 10	*4*	*6*	*7*	*5*	*6*	*7*	*5*	*6*
Tester 11	*2*	*2*	*3*	*1*	*3*	*2*	*2*	*2*
Tester 12	*8*	*7*	*7*	*8*	*7*	*7*	*7*	*9*
Tester 13	*5*	*6*	*6*	*5*	*6*	*6*	*6*	*6*
Tester 14	*6*	*5*	*6*	*8*	*6*	*7*	*8*	*7*
Tester 15	*3*	*4*	*5*	*4*	*4*	*6*	*5*	*4*
Tester 16	*4*	*7*	*9*	*6*	*6*	*7*	*7*	*6*

Tester 17	*4*	*6*	*6*	*5*	*7*	*7*	*8*	*6*
Tester 18	*3*	*5*	*4*	*6*	*4*	*6*	*5*	*5*
Tester 19	*7*	*4*	*4*	*7*	*5*	*5*	*7*	*5*
Tester 20	*7*	*5*	*5*	*8*	*6*	*4*	*8*	*3*
Tester 21	*6*	*7*	*7*	*6*	*4*	*7*	*7*	*4*
Tester 22	*5*	*5*	*3*	*5*	*4*	*7*	*5*	*5*
Tester 23	*4*	*6*	*5*	*5*	*3*	*2*	*8*	*6*
Tester 24	*7*	*7*	*4*	*7*	*1*	*3*	*9*	*5*
Tester 25	*6*	*7*	*7*	*6*	*4*	*7*	*6*	*8*
Tester 26	*5*	*5*	*4*	*7*	*3*	*7*	*5*	*3*
Tester 27	*3*	*6*	*4*	*4*	*4*	*5*	*4*	*2*
Tester 28	*3*	*3*	*2*	*3*	*2*	*2*	*1*	*1*
Tester 29	*4*	*3*	*3*	*3*	*3*	*3*	*3*	*3*
Tester 30	*1*	*1*	*8*	*1*	*5*	*3*	*4*	*6*
Tester 31	*5*	*2*	*7*	*3*	*8*	*5*	*6*	*3*
Tester 32	*4*	*3*	*6*	*8*	*4*	*3*	*4*	*5*
Tester 33	*7*	*6*	*5*	*6*	*4*	*8*	*8*	*4*
Tester 34	*7*	*7*	*9*	*8*	*7*	*7*	*7*	*7*
Tester 35	*4*	*7*	*4*	*8*	*4*	*4*	*6*	*8*
Tester 36	*1*	*1*	*1*	*1*	*1*	*1*	*1*	*1*
Tester 37	*8*	*7*	*6*	*9*	*6*	*10*	*9*	*6*
Tester 38	*6*	*8*	*7*	*8*	*6*	*7*	*7*	*6*
Tester 39	*4*	*3*	*2*	*3*	*2*	*3*	*5*	*3*
Tester 40	*3*	*6*	*4*	*5*	*3*	*3*	*4*	*3*
Tester 41	*7*	*5*	*5*	*7*	*5*	*9*	*8*	*7*
Tester 42	*4*	*7*	*7*	*5*	*6*	*5*	*4*	*6*
Tester 43	*4*	*5*	*5*	*6*	*4*	*7*	*5*	*5*
Tester 44	*5*	*5*	*6*	*5*	*6*	*7*	*7*	*5*
Tester 45	*6*	*8*	*7*	*8*	*6*	*6*	*8*	*7*
Tester 46	*2*	*6*	*6*	*9*	*5*	*9*	*5*	*2*
Tester 47	*5*	*7*	*7*	*7*	*8*	*9*	*8*	*7*
Tester 48	*5*	*7*	*7*	*7*	*8*	*9*	*8*	*7*
Tester 49	*7*	*8*	*7*	*8*	*7*	*7*	*9*	*6*
Tester 50	*7*	*5*	*5*	*8*	*6*	*4*	*8*	*3*

Appendix D – Schematics

Schematic of LM4550 [57]