# EE108 Final Project:

*Enhanced Music Synthesizer and Display*

## Introduction

| | |
|---|---|
| **Checkpoint 0:** | **Friday, November 8, 2019 (5 PM)** submitted digitally |
| **Checkpoint 1:** | **Wednesday, November 13, 2019** during class |
| **Checkpoint 2:** | **Wednesday, December 4, 2019** during class |
| **File Submission:** | **Monday, December 9 (noon)** |
| **Report Submission:** | **Monday, December 9 (3:30 pm)** |
| **Final Presentations:** | **Monday, December 9, 2019** in Lab from **5:30 - 8:30 p.m.** |

## So, what are we doing here?

The final project will provide you with a complete digital systems design experience. It encompasses all of the elements you have encountered in EE108:

Combinational and sequential logic
High level module design
Timing analysis and pipelining
User, memory, and device I/O
Developing a verification plan
Debugging, debugging, and debugging

Most importantly, this project gives you the chance to build a system of your own design.

## Overview

By the end of lab 5, you should have a working system on the Zedboard that plays a song stored in memory using pure tones (sine waves), and displays the waveform being played on a VGA/HDMI monitor. For the project you are to extend this basic music player, implementing additional sound effects, display functionality, and/or interface modes. A list of features and the points associated with each is included in this handout.

The number of extensions (number of points) you implement, the quality of your implementation, a demonstration of your system, an oral presentation and a written report will determine your grade for this project. Your final project should be implemented with good design and Verilog style; it should work as defined and be explained in a well-organized report.

There may or may not be a prize for the team with the best final project. In determining the "best" project, we will take into account **all** of the above criteria, along with the creativity and general polish of your final product.

To maximize your probability of success, be sure to adopt a strict specification, a rigorous design methodology (no shortcuts), and a strict timeline including extensive time spent verifying the behavior of your system in testbenches. This will ensure a timely completion, with minimal chance of needing a complete redesign near the due date.

To make sure you are on track, **there will be two graded checkpoints** in which the course staff will check on your progress and give you feedback.

Since all the components of the project (i.e. checkpoints, code submission, project report, demonstration, and presentation) contribute to the final project grade, it is imperative that you manage your time wisely and give each component adequate attention. In the remainder of this handout, further information is provided on the project's grade breakdown, schedule, deliverables, and implementation details.

## Possible Extensions

**You are only *__required__* to implement one additional feature for this project: chords**. The rest of this section presents the additional features you may choose from to implement for your final project. As has been mentioned, to get full credit you need to successfully implement at least 15 (20 for groups of 4) points of features. Note that the converse is not true: if you implement 15 points of features, you are not guaranteed full credit--that is contingent on the other parts of the project mentioned earlier. The checkpoints, report, and polish are worth 40% of your grade; features are not the be all and end all of this project.

**You are free to propose your own features. Consult the staff for approval and point value.**

### *Audio*

**REQUIRED: Chords (4 points)** – Allow your music player to play multiple (at least 3) notes simultaneously. Your design will need to generate multiple waveforms (one for each note in the chord) and combine them into a single audio sample to feed to the codec. See the "Audio Data Specification" section to get started. **This is the only required extension.**

**Echo (2-3 points)** – Add an "echo" to the sound being played by simultaneously playing a delayed version of the sound being fed to the output. The delay should be in the range of 100ms to 500ms and the delayed sound should be attenuated. Basic reverb is worth 2 points; if you add user controls for adjusting the length and attenuation or multiple voices with different echo lengths and attenuations, you can get an additional point.

**Harmonics (3 points)** – Add weighted harmonics (multiples of the fundamental frequency) to the note being played. Different weights will give different instrument sounds to the music. You can make a flute sound, a violin sound, a trumpet sound, or an electric guitar with distortion with the proper use of harmonics (and dynamics – see the next item). This is similar to playing chords, except the frequencies of the additional notes are multiples of the base note and the amplitudes of the higher harmonics are necessarily lower. (The addition of this feature enables a lot of variety.) This site may be useful in choosing harmonics: http://www.falstad.com/fourier/

**Dynamics (3-4 points)** – Modulate the signal amplitude during the time the note is played. The amplitude should be controlled by two time constants for *attack* and *decay*. The signal amplitude should increase linearly during the attack interval. The amplitude should then decay exponentially with a time constant set by the decay interval. You will receive 3 points for plain exponential decay, i.e. signal starts at full amplitude and decays exponentially. You will need to decide how to generate this amplitude “envelope” and how to use it to modulate the waveform. You can get more points if you implement more complex envelopes, such as the Attack-Decay-Sustain-Release (ADSR) envelope; consult the TAs for more information.

**Multiple Voices (1-3 points)** – This involves having the ability to play multiple instruments (notes or chords) simultaneously, possibly with different harmonics weightings, reverb delays, and/or attack and decay envelopes. You should extend the note format to specify the voice for each note, and you should provide a song that makes the multiple voices clear.  More points are awarded for tuning the multiple instruments to actually sound like real instruments.

**Stereo Effects (1-3 points)** – The codec is not monaural; it supports separate left and right audio channels. Generating a stereo signal can be as simple as choosing which channel a voice plays on (1 point), to more complicated effects such as detailed panning (2 points), stereo versions of the above features (1-2 points), or even a phaser effect (time-varying phase shifting) (3 points). You may need to use headphones to get the full effect, though.

## *Visual*

**Adjustable waveform display (1-3 points)** – Using user input to the FPGA board, allow the horizontal and vertical scale of the waveform display from lab 5 to be adjusted in real time. Simplistic adjustments such as “use full width of screen” and “use full height of screen” will earn 1 point, while being able to increment the scale over a variety of ranges will earn up to 3 points.

**Note display (1-4 points)** – For 1 point, display currently playing notes or chords as text on the screen.. For 2 points, also display the past few notes that have finished playing. For 4 points, also display notes in the future yet to be played.

**Enhanced waveform display (1 points)** – Draw each note in a different color on the display and show the combined waveform in white.

**Smooth waveform display (3 points)** – Use antialiasing techniques to render the waveform lines in a much higher-quality, smooth way. This involves treating the area between two samples not as boxes but as actual lines, and coloring pixels based on the pixel's sub-pixel distance from that line. Try experimenting with different techniques; some “technically incorrect” methods may be much simpler to implement and look good despite not necessarily being mathematically rigorous, and that's perfectly fine for this project. Depending on the visual quality and elegance of your solution you can earn anywhere from 1 to 3 points.

**Context-sensitive icons (1 points)** – Chances are, you'll end up with pretty complicated controls if you implement user interface features. Include some visual indicator on the screen that explains what the important controls do, and if you change 'modes' such that those controls do different things reflect this change in your visual indicator. As an example, start by showing play/pause and next-song icons arranged in the pattern of the board's push buttons, and either display the play or pause icon depending on the current state of the song.

### *Interface*

**Playback control (1-4 points)** – Add fast-forward and rewind functionality to your player for 1 point. When rewinding, you don't have to perfectly play your song in reverse, but you'll get up to 4 points if you do.

**Composition editor (4 points)** – This feature entails a user interface, complete with visual component, that allows the composition currently stored in the note memory to be edited by the user.

**Interactive Instrument Editing (4 points)** – This item should provide a means to interactively allow the user to edit:

The weighting of harmonics and/or

The attack and decay values used in dynamic amplitude modulation

To get all 4 points, you must implement some sort of graphical interface to display the values being changed. Points for interactive instrument editing are in addition to the one point for adjustable delay, if you choose to implement adjustable delay.

**Pulse width modulated LEDs (1 point)** – Use pulse width modulation to control the brightness of the LEDs on the board, and use this brightness control to display some kind of information such as (depending on what other features you've implemented) the current volume of the song, dynamics envelopes, or stereo panning.

**Human Interfaces (1-5 points)** – Our FPGA boards have a lot more human interfaces than just buttons and switches. Checkout the Pmods that can connect to the Zedboard for a variety of options. Let the course staff know ASAP if you would like to experiment with specific Pmods. We will order several, but the earlier you let us know you want to use a certain Pmod, the earlier you can start working it into your project. Points will be given based on the complexity of integration.

## Audio Data Specification (or, getting started with chords)

Unlike labs 4 and 5, the format of the data inside the note memory for the project employs a more elaborate specification, similar to a MIDI file. This is done in order to make the realization of a wider range of acoustic effects possible. The following is the format we recommend; you can, of course, extend the format or use your own format entirely, but **the entire song must be stored in a single ROM** (that means you can't just instantiate different song ROMs for each voice or note meant to play in parallel).

The note memory will be 16 bits wide by 512 entries long, providing 4 songs of 128 notes each (you can make it longer if needed). Each entry will have one of two formats:

0 xxxxxx yyyyyy zzz – schedule note xxxxxx (6 bits) to immediately start playing for a duration of yyyyyy (6 bits) 48th notes. The 3 zzz bits are metadata which can encode

volume, voice, or any other parameters you choose. We use 48th notes as our time basis to allow both 16th notes and triplets to be encoded. After scheduling this note, move to the next item in the song rom.
1 yyyyyy – wait yyyyyy (6 bits) 48th notes before reading the next entry in the song rom (or put another way, advance time in the song).

This format allows us to play multiple notes at the same time and to have some notes held for longer than others. For example, to play a C-major chord for one quarter note and then play an E (single note) for a half note we would encode:

`0 C 12` – Play C for 12 48th notes, ¼ note
`0 E 12` – Play E for 12 48th notes, ¼ note
`0 G 12` – Play G for 12 48th notes, ¼ note
`1   12` – Advance time for 12 48th notes
`0 E 24` – Play E for ½ note
`1   24` – Advance time ½ bar

The first three entries schedule three notes to be played starting at the current time. The fourth entry advances time. While time advances, the scheduled notes are played. All notes don't have to start and end at the same time. For example, suppose we want to hold a C through a bar and play E-G-E for ¼ note each on the 2-3-4 count. We would encode this as:

`0 C 48` – Play C for a whole bar
`1   12` – Advance ¼ bar
`0 E 12` – Play E for ¼ bar
`1   12` – Advance ¼  bar
`0 G 12` – Play G for ¼ bar
`1   12` – Advance ¼ bar
`0 E 12` – Play E for ¼ bar
`1   12` – Advance ¼ bar (C note and E note both end after this time)

Note that in addition to your enhancements you will need to modify the music player to handle this new note file format. You are advised to modify the spreadsheet provided in the lab 4 starter files to make it easier for you to generate songs in this new format. (This is easy and will save you a huge amount of time.)

**Chords amplitude FAQs & Verilog signed shift**

You may assume that the single notes from lab4 are operating at 100% amplitude, i.e. if you increase in amplitude, the audio will overflow, clip and distort. Our requirements for chords are simply (1) the music is audible, (2) the audio does not clip. If you're using bit shifters to scale amplitudes, please know that the signed bit shift operator is something available to you in Verilog. The way you accomplish a signed bit shift is by declaring the wire you want to shift as "signed", and using a triple carrot ">>>" instead of the traditional bit shift symbol.
E.g. dividing the signed signal "a" by 2 and assigning it to "b":

```
wire signed [15:0] a;
wire [15:0] b = a >>> 1;
```

## Project Schedule

### *Checkpoint 0*

For this checkpoint you should submit a brief proposal detailing the features you'd like to implement along with a short description of the implementation details for each feature.

### *Checkpoint 1 – 10%*

For this checkpoint you should have completed the following:

Complete specification for all elements of your system including block diagrams, and detailed specifications for the input/output interface.
State diagrams for all FSMs.
Timing table(s) describing all non-trivial system timing and communication protocols between different elements of the system.

The block diagrams should indicate how you plan to partition the overall system into more manageable sub-systems by function, as well as how they relate to and interact with one another. Timing tables detail how actions are sequenced in time. They will be especially helpful when dealing with audio, video, and memory events, which occur at well-defined time intervals. They are mandatory for communication between blocks that are managed by different group members.

Groups will sign up for short meetings with the course staff to review their progress.

### *Checkpoint 2 – 10%*

At this point you should demonstrate the following:

Successful simulation of all modules.
At least 2/3 of your features must be working in the lab (individual features demonstrated separately is acceptable, but integrated features are more desirable)

In order to fulfill the checkpoint requirements, it is advisable to work incrementally and to back up a copy of your design files in a repository each time you add a feature or a significant part thereof (see the Code Repository section below). This way, if you break your system just before the checkpoint, you would still be able to demonstrate the features you previously had working. Groups will need to demonstrate their working features and show their simulations in lab to the TAs for credit.

### *Project Submission – 60%*

You must submit all your project files, your final .bit file, and your project report electronically. Note that you will use this exact .bit file when you demo. This means the due date is absolute. You will not be able to make last minute changes after submission.

### *Project Report - 10%*

This report should give a high level overview of the project and its functionality. This includes a description of all extensions your group implemented, as well as FSMs and block diagrams for each. This should also include a description of any problems you encountered, how you dealt with them, and anything you are particularly proud of. Please keep this to five pages or fewer (excluding figures), points will be deducted if the report is too long.

### *Project Demonstration – 10%*

The demonstrations will take place in the lab. During this session you will have 10 minutes to demonstrate all of your features to the staff, and to discuss unique implementation challenges you faced. The performance and behavior of your system will be checked against the specifications and requirements in this document. You will use the same .bit file you submitted on the project due date. A polished, well-prepared demonstration can help a lot in the competition.

## Grading

The grade breakdown for the project is as follows:

Checkpoint 1 – 10%
Checkpoint 2 – 10%
Project submission (Verilog files, simulations, and functionality) – 60%
Report – 10%
Presentation and polish – 10%

The overall goal is to implement at least 15 points worth of extensions, as elaborated on earlier. Extra credit is awarded according to the following breakdown: 5 additional points of extensions will garner 10% more on the project submission portion; an additional 5 points will garner 5% more. Thus, 10 extra points (25 points total) will grant 15% more credit for the project submission portion. Any extra points beyond this will grant no additional credit. Keep in mind all features must be well-designed and correctly implemented in order to receive full credit. For groups of 4, add 5 points to each of these values (20 for full credit, 25 for 10%, 30 for 15%).

Your final design must be free of latches, combinational loops, and timing constraint violations. Recall that the .syr report generated during synthesis will report any inferred latches or accidental asynchronous feedback loops, and the .twr report will report your design's critical paths and whether they meet timing constraints. Synthesize parts of your project as early as possible to find these mistakes before it's too late to debug them. For serious.

## Suggestions!

Make sure you fully understand the requirements of each feature you plan to implement before committing to it. List the I/O requirements, changes required to the current music player and high level event and data flow.

Don’t bite off more than you can chew. It is better to have a fully functional set of features worth 15 points, than to have a partial implementation with a potential of 18.

Visit office hours and ask questions early in the development stage. Don’t start coding before you have a complete module breakdown, including control and data interfaces. The most important thing is to understand the data flow and timings.

Keep your code simple. If you can't read an always block without scrolling the document (vertically OR horizontally) it is probably too complicated. Break it down into intermediate logical blocks with meaningful functionality. This will also assist in testing and debugging.

Add features incrementally. Before working on the next feature, make sure the current design runs smoothly on the FPGA. Back up your work frequently, or better yet, use a code repository, as described in the section Code Repositories.

When you are mixing audio signals, you are allowed to lose amplitude, but make sure you understand what you’re doing. Trees of logic are better than long chains of logic.

You will need to pipeline your design. Make sure you are checking your timing reports when you synthesize in ISE. If you do not meet the timing requirements for our 100MHz clock you will have to figure out what your critical path is and insert flip flops to break it up.

For features that render text on the screen, use the provided tcgrom.v module which implements a ROM containing images for alphanumeric characters and various punctuation symbols.

## Physical Constraints

The BRAMs (i.e. memories) on the FPGA can be configured to accept up to two read/write ports. While memories with one write and two read ports can easily be implemented, memories with two write ports are complicated to implement. Please contact the course staff should you desire to use a memory with two write ports.

The FPGA contains 220 multipliers, each of which supports multiplications of up to 25 by 18 bits. Larger multiplications should not be implemented and will not be supported by the course staff. You should additionally avoid using multipliers whenever possible, as they will make it more difficult for your project to satisfy timing constraints.

Please resist the urge to change the frequency of the FPGA clock. If for any reason you need to divide the FPGA clock, please talk to the course staff immediately.

## Source Control

Source control is an invaluable tool for keeping track of all of the changes you and your partners make. It allows you to work on your own personal copy of the files, changing them however you wish, and then periodically sync up with your partners and resolve any conflicts that may arise. Whenever you make some changes, you can “commit” them to the repository along with a message. If your design breaks, you can see the changes between your current version and any version you've committed in the past, and even revert your code.

You can run `vcs-setup` in the terminal to create a new repository. Repositories are hosted on your own personal AFS space, so be sure to check to see how much free space you have by running `lelandquota`. You can choose between Mercurial, Git, and SVN as your version control system; if you don't know the difference, we recommend you start with Mercurial. Follow the prompts in `vcs-setup` to set up your repository (only you and the teammates you specify will have access to it), and you will get a list of hints, commands, and links specific to the version control system you chose. If you have any difficulties, please contact the TAs.

## Project Deliverables

### *Project Files*

Your project submission should include:

All synthesizable Verilog files used in your system.

All testbenches you wrote and used to verify your design, and annotated versions of their output.

All files pertaining to any memory modules used, i.e. ROMs, RAMs, and any spreadsheets you used to generate them.

Timing and synthesis reports (.twr and .syr files) for your final bit file.

The ISE project along with the *.bit file for the final version of your system.

### *Project Report*

Along with the Verilog files submitted, you are also required to electronically submit a report with a **maximum length of 5 pages, excluding figures**. Please adhere to this length restriction. Points will be deducted for reports that are too long. Topics in the report should include the following:

Specification: a description of each extension you implemented. Include block diagrams and FSM state diagrams where relevant.

High-level design: describe your block diagram and overall functionality.

Key implementation details: describe aspects of your design that you think are noteworthy. Include design techniques and timing that is particularly clever or subtle.

Brief descriptions of any problems encountered during design and how you resolved them.

### *Project Presentation (described above)*

### *Final Evaluation*

At the end of your project presentation, an anonymous survey will be given to each team member in order to provide an assessment of the project and to bring forth any concerns.