Final Project: Enhanced Music Synthesizer and Display

Introduction

The final project will provide you with a complete digital systems design experience. It encompasses all of the elements you have encountered in EE108A:

- combinational and sequential logic
- high level module design
- timing considerations
- user, memory and device I/O
- employing the right tools and strategies for testing
- debugging, debugging and debugging

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

Due Dates:

Checkpoint 1:	Monday, March 5th, 2007
Checkpoint 2:	Monday, March 12th, 2007
Submission:	Saturday, March 17th, 2007 files due at noon, report due at midnight
Demonstrations:	Sunday, March 18th, 2007, Packard 127 (time to be decided)

Project Overview

By the end of lab 5, you should have a working system on the Xilinx Virtex II Pro FPGA board that plays a song stored in memory using pure tones (sine waves), and displays the waveform being played on a VGA 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.

Your grade will be determined by 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. Your final project should be implemented with good design and coding style, it should work as defined and summarized in a well organized report. Extra credit will be awarded for more than 15 points worth of features.

To maximize your probability of success, adopt a rigorous design methodology (no shortcuts) and a strict timeline. These 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. check points, code submission, project report, demonstration, and presentation contribute to the final project grade, it is imperative that you manage your time wisely and plan to 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.

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%

Project Schedule

Checkpoint 1

The first checkpoint for the project is one week after the project is assigned. For this checkpoint you should have completed the following:

- Complete block diagrams of your system, including detailed interface specifications listing all signals and describing their timing.
- State diagrams for all FSMs.
- Timing table(s) showing all non-trivial system timing.

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.

Groups will sign up for 20 minute meetings with the course staff to review their progress. Sign up for the earliest time you can make so you'll get feedback sooner!

Checkpoint 2

The second checkpoint is one week after checkpoint 1. At this point you should have achieved the following:

- Successful simulation of all parts that can be simulated.
- At least 2/3 of your features must be working in the lab

In order to fulfill the checkpoint requirements, it is advisable to work incrementally and save a copy of your design files in a repository each time you add a feature or a significant part thereof. This way, if you break your system just before the check point, 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

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.

Project Deliverables

Project Files

Your project submission should include:

- All synthesizable Verilog files defining your system.
- All files pertaining to any memory modules used, i.e. ROMs, RAMs, and any spreadsheets you used to generate them.
- The Xilinx project file along with a *.bit file that encapsulates the final version of your system.
- All testbenches you wrote and used to verify your design along with any ModelSim *.do files.

It is recommended that you organize your files in the same manner as the lab files, i.e. files of synthesizable Verilog code in the top-level folder, Xilinx ISE project files in the `ise` subfolder (remember to clean up project files in Xilinx before submitting), ModelSim *.do files and Verilog testbenches in the `sim` subfolder.

Project Demonstration

The demonstrations will take place in the lab. During this session you will have exactly 8 minutes to demonstrate each of your features to the staff. 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.

Project Report

Along with the Verilog files submitted, you are also required to electronically submit a report with a **maximum length of 5 pages *including* 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.
- 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.

Audio Implementation and Additional Feature List

Audio Data Specification

Unlike labs 4 and 5, the format of the data inside the note memory for the project employs a more elaborate specification, analogous to a MIDI file. This is done in order to make the realization of a wider range of acoustic effects possible. More specifically, 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 by instantiating larger RAMs). Each entry will have one of two formats:

- 0 xxxxxx yyyyyy zzz – schedule note xxxxxx (6 bits) to be played for duration yyyyyy (6 bits) 48^{th} notes with an amplitude of zzz (3 bits) starting at the current time. We use 48^{th} notes as our time basis to allow both 16^{th} notes and triplets to be encoded.
- 1 yyyyyy – advance time by yyyyyy (6 bits) 48^{th} notes.

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 cord 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 48^{th} notes, ¼ note
- 0 E 12 – Play E for 12 48^{th} notes, ¼ note
- 0 G 12 – Play G for 12 48^{th} notes, ¼ note
- 1 12 – Advance time for 12 48^{th} 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
- 0 E 12 – Play E for ¼ bar

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 provided spreadsheet 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.)

Design a la Carte

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. Each feature has an associated maximum number of points it can garner. **To get full credit you need to successfully implement 15 points worth of features**.

You are welcome to propose your own enhancements to the music generator or the VGA display not listed in this handout. For these please consult the staff to get approval and point value.

Audio:

Reverberation (2-4 points) – This addition should create an "echo" to the sound being played by simultaneously playing a delayed version of the sound (note the note) 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 you can get 2 additional points.

REQUIRED: Chords (5 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.

Harmonics (5 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. (This is by far the coolest addition.)

Dynamics (5 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 from zero to the specified value (zzz) during the attack interval. The amplitude should then decay exponentially with a time constant set by the decay interval. Since the attack interval is usually quite short, you will receive full credit if you choose to implement just 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.

Multiple Voices (5 points) – This involves having he ability to play multiple instruments (notes or chords) simultaneously, possibly with different harmonics and 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.

Display:

Adjustable waveform display (2 points) – By using the available push buttons on the Xilinx FPGA board, allow the horizontal and vertical scale of the waveform display from lab 5 to be adjusted in real time.

Note display (2 points) – This additional feature should display the note or chord currently being played as text on the screen.

Enhanced note display (2-4 points) – For this item your system should display the few notes or chords most recently played *and* the few notes or chords soon to be played along with the current note. The sequence of notes can be displayed using characters or a graph with time along the horizontal direction and pitch along the vertical direction. You can get up to 2 points if you just display the previous notes or the next notes, and up to 4 points if you do both.

Enhanced waveform display (1 points) – Draw each note in a different color on the display and show the combined waveform in white. This is really easy and really cool.

Interface:

Playback control (2 points) – Add play, pause, fast-forward, and rewind functionality to your player. When rewinding you do not have to obey delays perfectly, but if you do we will give extra points.

Composition editor (5 points) – This feature entails a user interface that allows the composition currently stored in the note memory to be edited by the user.

Interactive Instrument Editing (5points) – 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.

MIDI reader (8 points) – Create a module that can read a standard MIDI file stored in the FPGA's RAM. You'll have to look up the MIDI file spec, but this will allow you to play any MIDI file you can find online.

Friendly Suggestions

Make sure you fully understand the requirements of each feature you plan to implement before committing to it. List its I/O requirements, points of change in 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 your case statement doesn't fit on a single screen page (width and length) it is probably too complicated. Break it down with intermediate values and give them meaningful names. 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, and save a copy of it.

When you are combining audio signals you are allowed to lose amplitude, but make sure you understand what you're doing. (You'll understand this by the time you get to checkpoint 1.)

You will need to pipeline your design. Make sure you are checking your timing reports when you synthesize in Xilinx. 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. (Hint: multipliers are slow...)

Good luck.