Real Time Sound Analysis / Synthesis

Germain Martinez, Gerzain Mata, Michelle Qiu
6.111 Introductory Digital Systems Laboratory
Final Project Proposal
Fall 2015

Overview

There's been a proliferation of Digital Audio Workstation programs throughout the last couple of decades. Many of them have had varying degrees of success. Unfortunately, a bunch of these systems lack the intuitiveness to get them into the hands of the masses. The goal of our project is to produce a Human Interface Device that takes audio samples from a microphone, saves them into memory, and overlay sound effects on those samples for instant results. The user will be able to record multiple audio samples, choose which effects to perform (i.e. reverb, echo, vocoder), and reproduce the audio with the effects applied. At the same time we will be able to provide statistics on the sound sample, like maximum amplitude, frequency domain content, etc, on the screen during playback.

Design

The project is expected to work in the following fashion: the user can select from two modes: record and playback.

In the record mode, the FPGA picks up audio samples from the microphone input and sends those samples to the memory handler. The memory handler stores these samples to memory.

In the playback mode, the FPGA takes the samples stored in the memory and plays them back through the speakers. The user can add sound effects to the samples playing through the speakers by flipping switches on the labkit. Each switch will correspond to a different effect; switch 1 could correspond to adding compression, switch 2 could correspond to adding an echo, and so on. These samples will also be shown on the screen output as a real-time representation of the audio playback output. As these samples are being played back, a Fast Fourier Transform will be done on the samples and the results (the frequency content of the audio output) will be shown on the display.

We will be using the ac97 module, which outputs 18 bit samples at 48kHz. However, to save space in our memory, we will be only recording the 12 highest bits of the sample. In addition, we will only be storing every 2nd sample, so the rate of audio data transfer is 12 bits at 24kHz. Since we have 2 blocks of 512k x 36 bits of ZBT SRAM, if we allocate 2 480k x 36

blocks for songs and 2 32 x 36 blocks for graphics, we can store a total of 120 seconds of audio clips.

Block Diagram

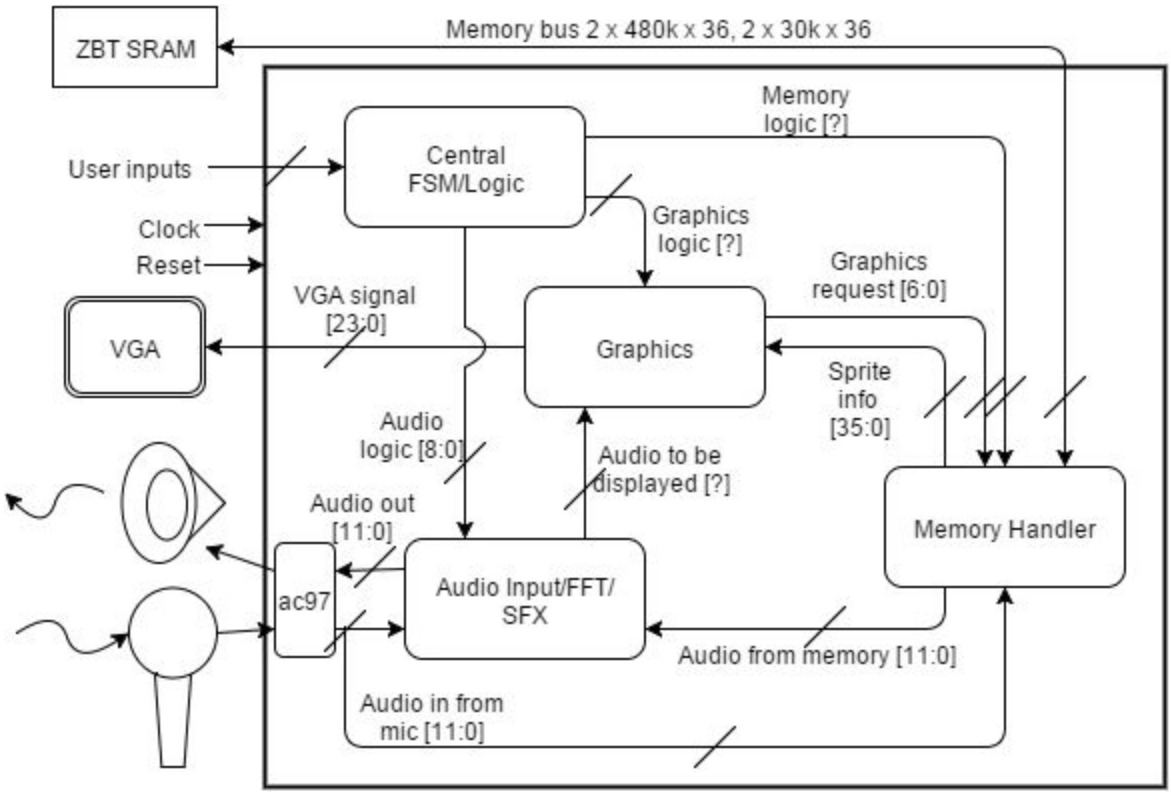

Figure 1: Overall block diagram.

Module Implementation

The design will be split into multiple modules to facilitate the distribution of work and debugging processes. The central FSM takes inputs from the user and the other modules in order to control the functions of the other modules. There are three main modules that will interact with the central FSM. These modules are the Audio / FFT / SFX Module, the Graphics Module, and the Memory Handler Module. The Audio / FFT / SFX module processes and outputs audio with the labkit's ac97 function and performs FFT and other sample transformations and calculations on the audio. The Graphics Modules displays the state of the program and statistics to the user. The Memory Handler Module stores audio and sprite information so that it can be used in playback and display.

Germain will be responsible for the Audio / FFT /SFX Module, Michelle will work on interfacing with the ZBT memory, and Gerzain will be tasked with handling the graphics output.

All group members will cooperate on handling the central FSM logic so that their individual modules communicate correctly with each other and with the central FSM.

Central FSM Module

The central FSM will be in charge of sending the correct states (Playback / Record) to the Audio / FFT module. At the same time, the central FSM will manage what statistics are displayed to the graphics and which bank of memory to access on playback.

This module controls the three other main modules based on inputs from the user and the other modules. This includes determining the state of the program (record, playback, filtered audio, other menus) by transitioning based on inputs from the user and determining the state of the other modules. The central FSM module controls the graphics module by telling which state that the graphics module should be in and what info from the audio, and to some extent, the memory, it should display. The central FSM controls the memory handler module by telling it whether it should be reading or writing, which memory block it should be reading/writing audio to, and when to start reading/writing to a memory block. The central FSM controls the audio module by telling it which effects to apply to the audio and when to send audio to the speaker.

In addition, the Central FSM will keep track of how many address locations in memory a recorded song takes up and translates the number of addresses to seconds.

Audio Input/ FFT /SFX Module (Audio Module)

Audio Module Block Diagram:

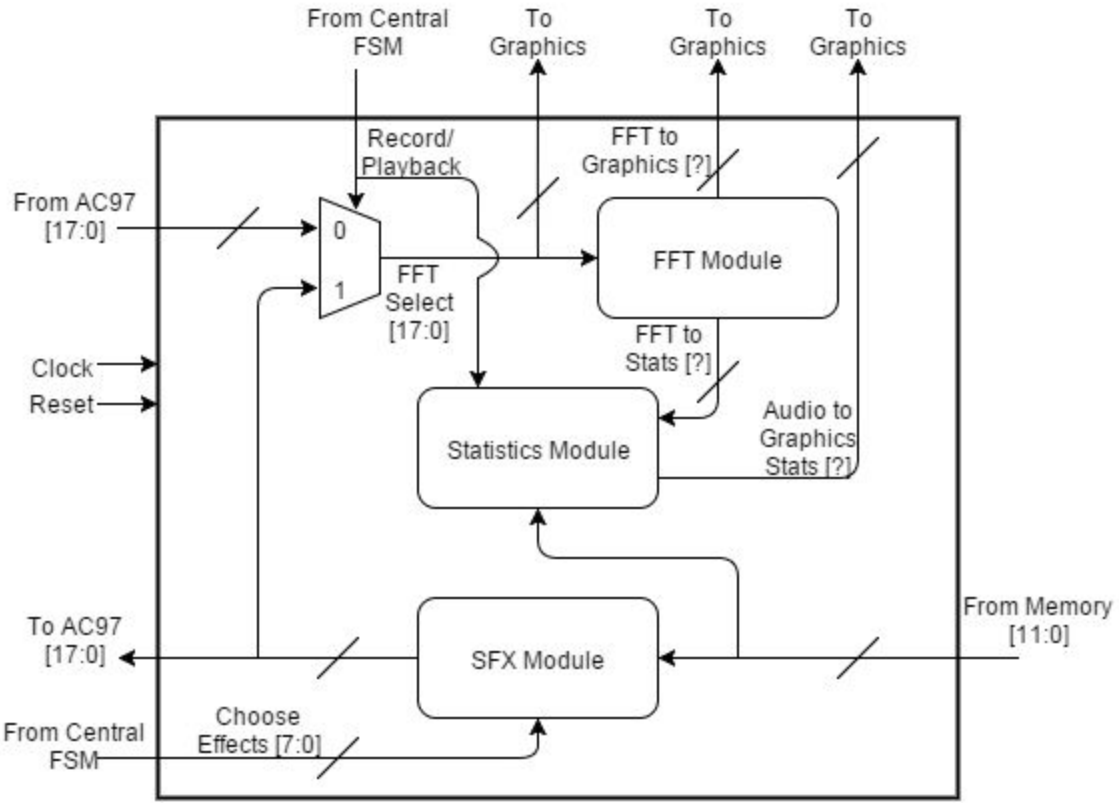

Figure 2: Audio Block Diagram.

The audio module records and plays the audio to ac97, calculates the statistics and information for a sound clip, and performs FFT and synthesizer transformations on the audio.

In the Record state, this module will take in raw audio samples from the AC97, run these samples through the FFT hardware core, and sending the decomposed frequency spectrum to the Graphics Modulefor visualization. At the same time, these audio samples will be written to the ZBT memory through the Memory Module.

In the Playback state, this module will take in audio samples from the chosen memory bank. Depending on what effects the user selects, like compression, delay, phasing, and limiting, this module applies these effects to the audio samples. These modified samples are then output to the AC97 headphone output. At the same time, the module takes an FFT on the audio output and sends this frequency information to the Graphics module for visualization. Optionally, the module can record certain statistics, like maximum output, and send these statistics to the graphics module for output.

While in previous labs we employed a 6 kHz sampling frequency, we obtained significant improvements in audio quality at a sampling rate of 24kHz. We plan to employ a 24 kHz sampling frequency with 12 bits per sample. This should give us a 288 kilobits per second data

rate. Ideally, the user should be able to record multiple audio samples; each song can be recorded on a bank of the ZBT memory.

Audio Module submodules and their functions:

FFT: performs the FFT transformation on the audio signal and sends it to graphics.
Statistics: calculates the statistics of the audio, including max amplitude, frequency, effects, and status.
SFX: applies special effects on the audio before it is sent to the speaker based on input from the central FSM.

Memory Module block diagram:

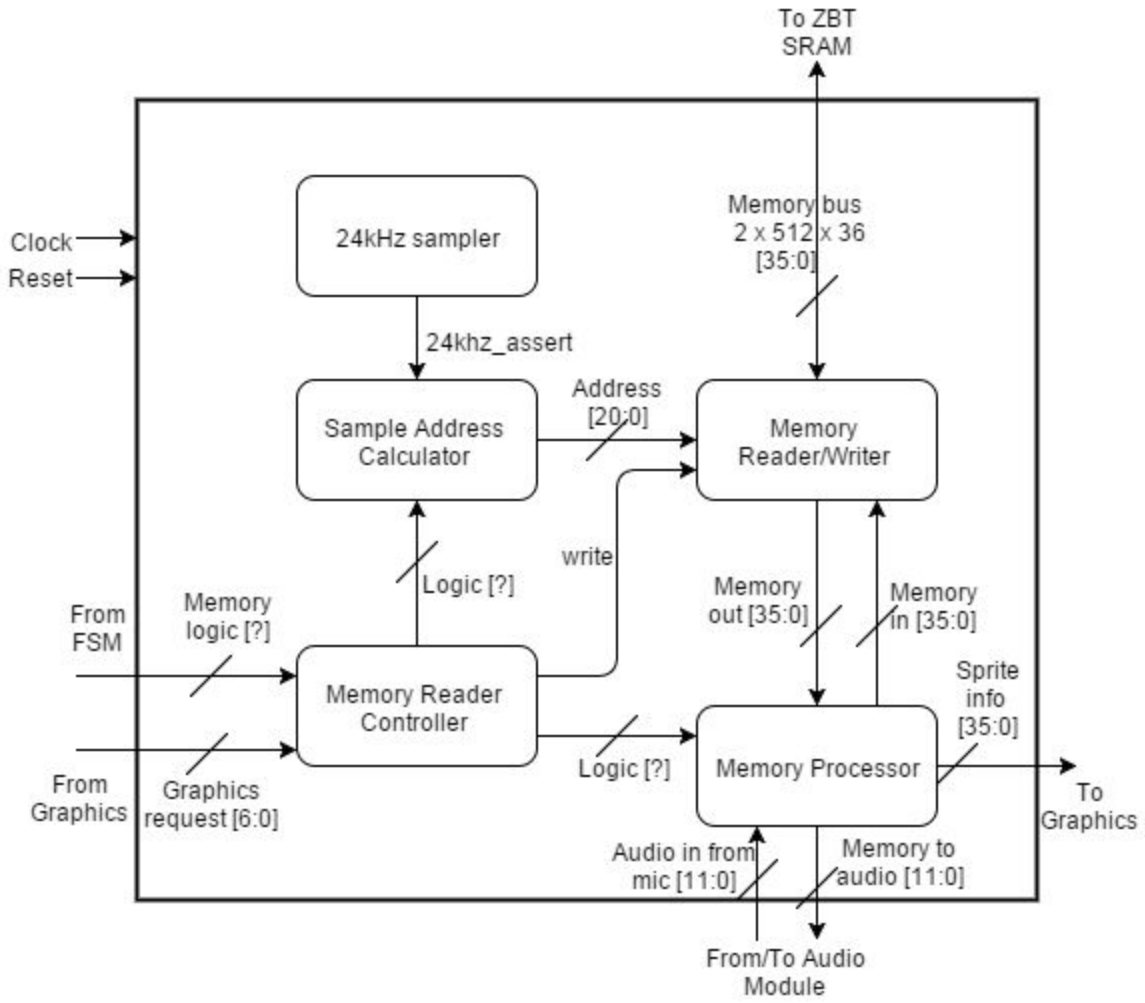

Figure 3: Memory Block Diagram.

The Memory Module is in charge of writing to and reading from memory. The module will be tasked with writing the input microphone audio samples into ZBT memory during Record mode and reading the audio samples once Playback mode is enabled.

At the same time this same memory module will be in charge of reading encoded image data so that the Graphics module can output images and sprites indicating numbers, letters, or other characters as needed. Since both video and audio data will be read there will have to be some memory bus and clock sharing. Since the ZBT SRAM has two banks of 512k by 36-bit memory this will allow the memory encoder to allocate between 480k locations of memory on each bank for a song sequence. Our plan is to leave at least 30k locations on each bank for image data.

Memory Module submodules and their functions:

24kHz sampler: tells the module when to increment to the next address for audio samples
Address Calculator: calculates which and what address should be read at that clock cycle (audio, graphics, etc), increments and resets address
Memory Reader Controller: Transforms requests from the FSM and Graphics into a form that the Address calculator and memory reader can use
Memory Reader/Writer: Reads and writes to memory given address information, data, and a write signal. Outputs
Memory Processor: Transforms the read memory signal into an audio signal [35:0] to [11:0] or a graphics signal [35:0] or from an audio signal into a memory signal [11:0] to [35:0]

Graphics Module

Graphics Module block diagram:

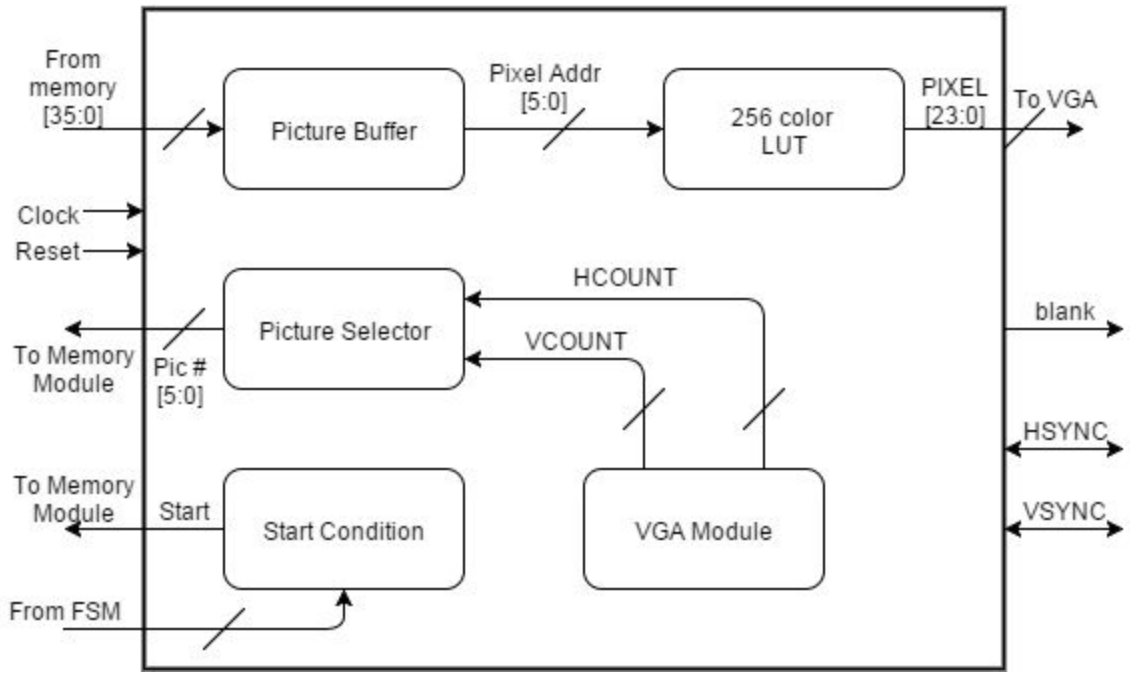

Figure 4: Graphics Block Diagram.

The Graphics Module will be tasked with displaying the frequency spectrum information and current audio sample information delivered by the Audio / FFT Module. The frequency spectrum will be displayed on the screen alongside the current running statistics of the sample, which are max amplitude, frequency, effects, and status.

The intended resolution is 1024 x 768 pixels. Image sprites will be used throughout this module, which means that it will have to fetch the images from the Memory Module. Since the ZBT memory has a 36-bit wide bus this allows one to fetch six pixels for every memory read. We will make each image 64 by 64 pixels using 256 colors. This means that each image will take up approximately 24.5 kilobits of memory, which entails 683 ZBT memory locations. A look-up table will be used to decode the 6 bits of pixel information into the 24 bits of information required for the VGA protocol.

Graphics Module submodules and their functions:

Picture Buffer: processes, times, and holds information from memory to be used by the 256 color LUT.
256 color LUT: transforms information from memory into a 24 bit pixel encoding that can be used by the VGA.
Picture Selector: chooses which picture needs to be extracted from memory..
VGA Module: decides what pixel from memory should be placed where in the display.
Start Condition: determines when a sprite needs to be pulled from memory.

Project Timeline

A summary of the proposed timeline is shown below. Unfortunately we initially planned to do a different project that ended up being unfeasible. As such we had to reorganize a new project and we were set back a week. Nevertheless we foresee having enough time to accomplish our objectives. The lines are colored to differentiate between each work stage. The timeline reflects the parallel workflow for each group member. It should take every team member approximately equal amounts of work, nevertheless it is expected that if a team member has accomplished their objectives with time left over then they should help out on other necessary areas.

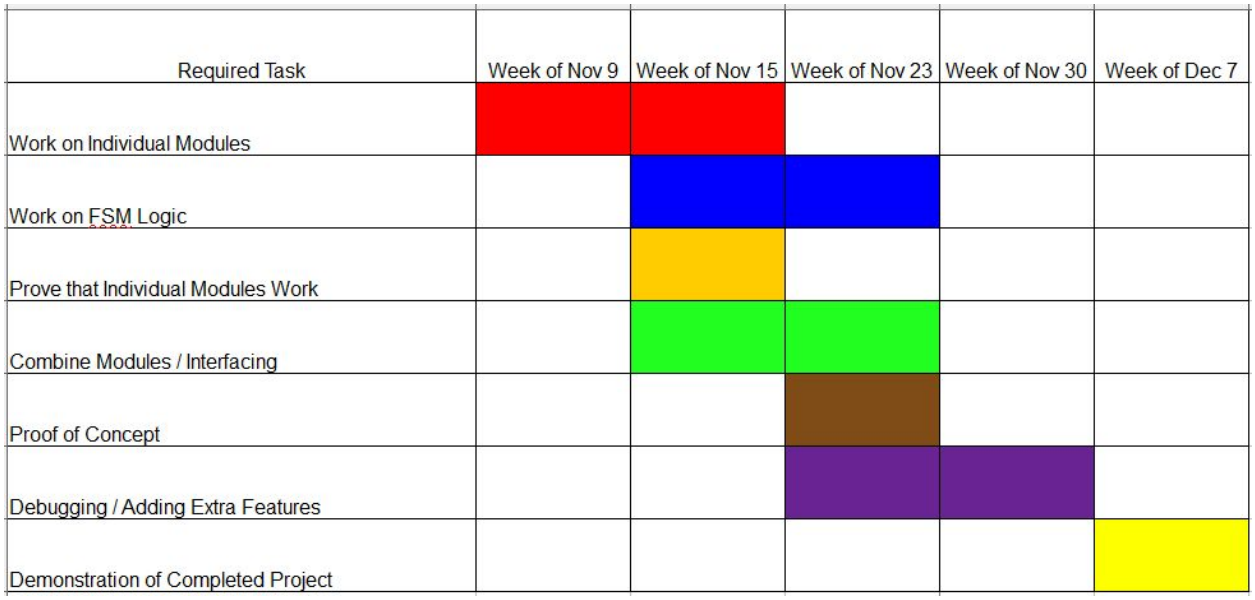

Figure 5: Timeline.

Week	Tasks to Complete
Nov 9th	Basic graphics output; reading/writing to ZBT memory; sampling from microphone, playback from memory
Nov 15th	Incorporating images in graphics; memory timings; saving samples to memory; incorporating different sound effects; incorporate central FSM logic
Nov 23rd	Graphics, sound and memory modules synchronized to central FSM logic
Nov 30th	Final debugging, meeting stretch goals (time permitting)
Dec 7th	Demonstration of Completed Project

Testing

For testing, we will create testbench modules to verify that a particular piece of the project will work. Once the sound module has been completed, we will make a test project that will check on whether or not the sound modules can perform the desired transformations on the audio signal before outputting it to the AC97. On the FPGA, the memory handler module can be tested in real-time by using the switches as the inputs and manually checking to see if the output is as desired. The graphics module can be tested by running the code on the FPGA and seeing the output on the screen.

Upon further progression, more testbenches will be implemented where communication between specific modules will be tested (i.e. graphics - FFT communication, graphics - memory communication, audio - memory communication). As for the audio effects, our plan is to determine which effects are easier to perform. We hope to successively implement more challenging effects with the hope that we can implement a rudimentary vocoder if time allows.

Resources Needed

In our approach we will use the 6.111 Labkit, which comes with a plethora of integrated components, amongst which we will employ the AC97 codec, the ADV7185KST video decoder, along with the two banks of CY7C1470V33 ZBT SRAM. For playback and recording, we will use a headset provided by the course. Everything else can be synthesized through the FPGA.

Stretch Goals

Accomplishing anything extra beyond our description of the project will entail a fairly substantial commitment. Nevertheless if time allows we plan to introduce additional functionality to the project, including but not limited to implementing complex effects like an equalizer or flanger, improved graphics, pause/ stop/ start functionality, and having variable sound clips to manipulate. If time and constraints permit we would ideally like to have vocoder functionality.

Conclusion

Our project seeks to implement an introductory Digital Audio Workstation (DAW). Using what our team has learned throughout the semester we plan to make use of the FPGA to implement audio synthesis and processing to help interested audio enthusiasts gain a better intuition of the inherent properties of audio signals and how effects influence those sounds in the frequency spectrum.