The Wumpus

Damon Vander Lind
Mark Tobenkin

November 4th, 2005

Abstract

Team Wumpus will develop a sampling synthesizer keyboard. This keyboard will allow a user to record, edit, and playback a sound. The synthesizer will provide a set of filters, applied on playback, and two different playback modes. In the first of these modes a recorded sample will be mapped to a MIDI keyboard, and replayed with frequency shifts. The alternate playback mode will allow the user to listen to the sample on loop, and adjust various filter parameters. To aid in editing an external screen will be used to display a number of visualizations.

1 Introduction

We, team Wumpus, intend to build the Wumpus. The Wumpus is a sampling synthesizer keyboard. A user records a sample which is analyzed to determine it's frequency. The user may then use the keyboard to play back notes and chords composed of the sample, shifted to correspond to the keyboard note frequencies. Alternately, they may select to have the sample play back on loop, so as to adjust trimming on the leading and lagging edges of the sample, and to adjust parameters on filters applied to the output. All filter values are displayed on the hex display or the video output.

Under keyboard mode, the user can play at least 10 concurrent notes. When a key is held, the sample will play up to the user-set vowel hold marker, and then hold that vowel until the key is let up, at which point the rest of the sample is played.

Visualization in the form of an oscilloscope and a VU meter are displayed on separate monitor, showing the current state of the output. There are also displays of the unfiltered and filtered sound sample time series.

2 Design

The Wumpus consists of 8 top level modules: Control, Sample, Record, Notes, Echo, MIDI, AC97 and Visualization. Figure 2 displays the topology of these modules. The following sections describe each module and its essential submodules.

Figure 1: Definitions

sample The sample refers to the entirety of a recording being played by the sampling synthesizer.

frame A frame literally encodes a single value of the sample at a given time.

window A window consists of a small subset of the sample, consisting of many sequential frames.

2.1 Control

The Control module handles the primary timing of the sampling synthesizer. This includes signaling changes in playback and record states, coordinating the output of the *notes* modules with user input and the timing between the *AC97* and other modules.

2.1.1 Note Control

The *note_control* module organizes the control signals to the *notes* module depending on playback mode and user input. In "keyboard" mode, *note_control* maps the input from the *MIDI* to the appropriate values for the control signals. In "loop" mode, the *note_control* module signals a single *start* upon receiving an *eos* for one *note* module.

2.2 Sample

The *sample* module manages access to the sample. Pulling the *wr/ re* line high enables writing to memory and disables reading. The *w_addr* and *wr* signals control writing to the memory. Words are written into memory from the *data_in* signal. When read enabled, the module presents data addressed by *r_addr* to the *data_out* bus. The module exports the number of frames stored of the sample via the *length* signal.

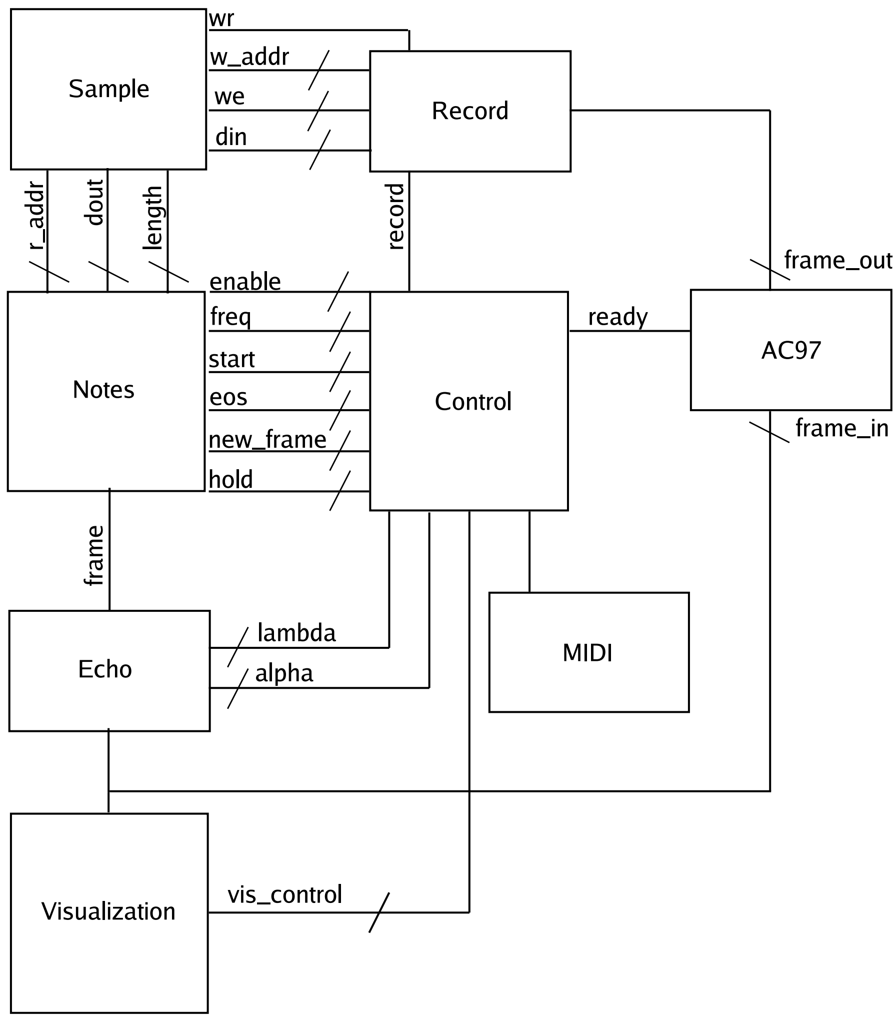
2.3 Record

The *record* module implements the audio recording logic which controls delivering new samples to the *sample* module. The module uses the control lines of *sample* to place the sample module in write/not read mode. It then clocks the data output of the *ac97* soundcard module into the *sample* module.

2.4 Notes

The *notes* module synthesizes a polyphonic output from multiple parallel transformations of the original sample. Each "transformation" consists of a pitch shift, delay and time elongation. The *notes* module consists of multiple *note* modules which each implement one of these parallel transformations. The *freq_select* bus specifies the frequency output of each individual *note*. The *start* bus causes *note* modules to begin their

Figure 2: Main Block Diagram

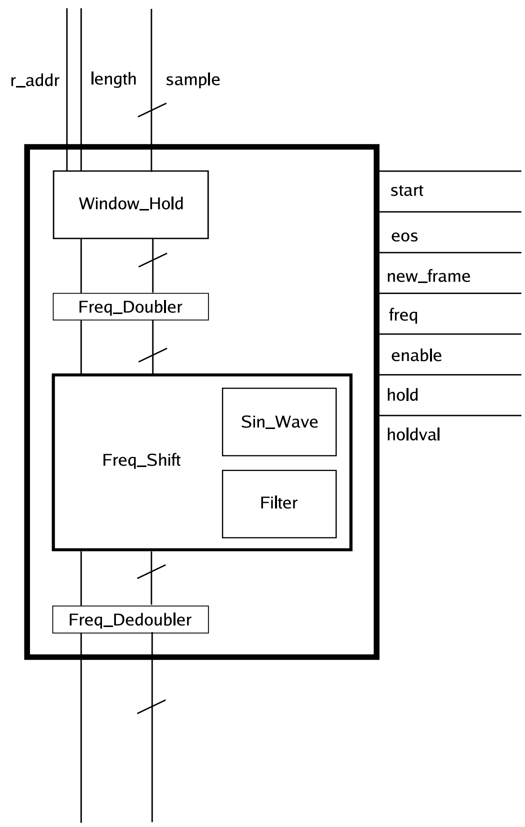


playback.. The *eos* module signals that a *note* has completed playing back the sample. Finally, the various *note* module outputs synthesized into a single output at *frame*. The *new_frame* signal acts as a clock to pipeline the *notes* module.

2.4.1 Note

Each *note* encompasses *window_hold*, *mem_note*, and *freq_shift*, *sin_wave*, and *filter*. Figure 3 details the topology of these modules.

Figure 3: Note Block Diagram



2.4.2 Frequency Shift

The *freq_shift* uses multiplication with sine waves to shift the frequency of a stored sample to correspond to the desired frequency of output. This is done by a series of two multiplications and two bandpass filters.

2.4.3 Sine Wave

The *sin_wave* module outputs a sin wave of a specified frequency. This is used to implement *freq_shift*.

2.4.4 Filter

The *filter* module takes in a signal and two filter parameters, and outputs the signal through a bandpass filter, with bounds set by two filter parameters.

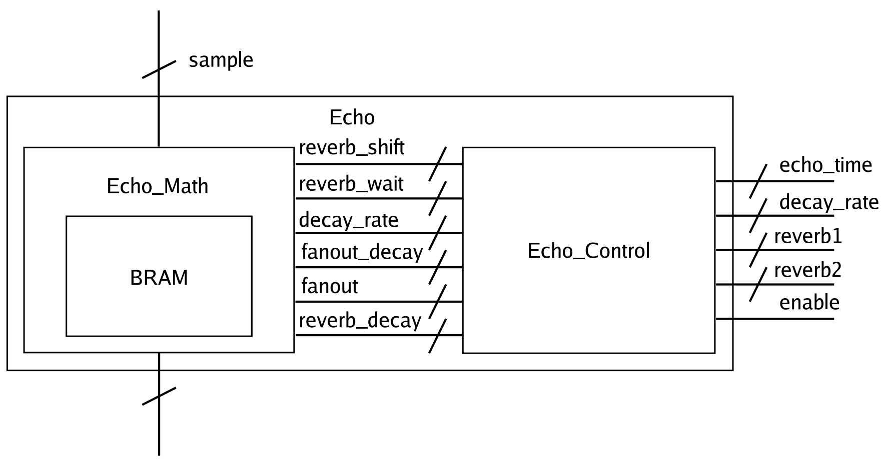
2.4.5 Window Hold

The *window_hold* module is intended to allow a piano key hold to hold a vowel in the sample for a long time. This can be accomplished by repeating a short segment of the word (order of magnitude 1/10 of a second) during a vowel. The function instructs memory to repeat a short segment following the address indicated by the user, while hold is asserted. The output of *window_hold* goes to *freq_shift*.

2.4.6 Bandpass

The *bandpass* module multiplies by two sinc functions of the right frequencies to block frequencies specified to it by *freq_low* and *freq_high*.

Figure 4: Echo Block Diagram



2.5 Echo

The *echo* module consists of two modules as detailed in Figure 4. One is a math module performing sound repeat operations based on a number of parameters. The other is a control module that takes in *enable*, *enable_reverb*, the *echo_time*, and two reverb parameters. Note that echo outputs sound at a sample width of 19 bits, to account for the ability of the maximum volume to double with the addition of *echo*'s at or below 50% quieting per *echo*. The output of *echo* goes to the ac97 input.

2.5.1 Echo Math

The *echo_math* module takes in *decay_rate*, *echo_length*, *fanout*, and *reverb_wait*, and *reverb_delay*. In normal echo mode, *fanout*, *fanout_decay*, *reverb_shift*, and *reverb_delay* are set to 0. In reverb mode, *fanout* controls the number of echos to write N**reverb_shift* locations earlier in the memory, with N indicating the Nth fanout. The *echo_math* uses a BRAM to perform these operations.

2.5.2 Echo Control

The *echo_control* sets the parameters for *echo_math* to correspond to a smaller number of more humanly intuitive parameters for echo and reverb.

2.6 Visualization

The *visualization* module displays 3 basic graphs, each generated by a module. Both the sample and the filtered sample are displayed as time series. The output will be displayed by a continuos ”oscilloscope”. Lastly, a 16 or 32 bar VU meter implemented with a small fast fourier transform.

2.6.1 VU meter

The *vu_meter* will use a fast fourier transform to build a bar graph of the power spectrum of the sample as it plays.

2.6.2 Time Series

The *time_series* module outputs an unfiltered time series of the output upon recording, and a filtered version of the output on playback.

2.6.3 Oscilloscope

Oscilloscope displays a running waveform of the output.

3 Timeline

The Wumpus will be implemented in six one week milestones. Mark will implement the modules for *sample*, *MIDI*, *record*, *note_control*, and *notes*. This places Mark's focus on memory management and use of MIDI to make sample notes play. Damon will implement *echo*, *note*, and *visualization*. This means he will focus on signal processing. There will certainly be overlaps in work in the implementation, as we work to resolve issues and fix bugs. Both of us will work on the *control* module, as we suspect it will be the hardest to debug.

Table 1: Milestone Timeline

Week	Author	Tasks
1st	mmt	Implement *sample* and *record*.
	damonv	Implement *frequency_shift*.
2nd	mmt	Implement *MIDI* interface.
	damonv	Implement *window_hold* and minimal *echo*.
3rd	mmt	Implement *notes* and *note_control*.
	damonv	Complete *echo* module.
4th	mmt	Implement Fast Fourier Transform.
	damonv	Implement waveform visualization.
5th	mmt	Implement ”loop” playback mode and ”chorus” sound effect
	damonv	Implement VU-meter visualization.
6th		Debug, test and refactor code base.