Keytar Hero Project Proposal

Hubert Hwang
Hui Tang

November 3, 2006

Introduction

The "Keytar Hero" project is intended to create a working keytar – a keyboard worn around the neck like a guitar – using the labkit as its sound generator. The system will involve several modules working in tandem to produce a variety of tones, with each module controlling some aspect of the sound production as a whole. The modules will consist of an instrument bank, a sine wave generator, a series of digital signal processing modules, and a mixer module that sends data to the AC97 sound chip. These will all be linked together by a main module that takes input from the physical keytar and controls all of the other modules to produce the proper output.

Modules

Main Module

The main module is the controller of all of the other modules. It takes input from the keytar as an array of bits, retrieves the appropriate frequencies from the sine wave generator, multiplies them by the appropriate amplitudes from the instrument bank, and then passes them off to the DSP chain. It also handles initial volume adjustment.

At each clock cycle, the `amp` array will be updated with the appropriate amplitudes for each possible key frequency. If the corresponding note signal is high, then that amplitude and its harmonics will be added to a running total, which will in the end be divided by the number of notes played, so as not to cause overflow. Then the running total will be sent to the `signal` output.

The main module will be designed and implemented by Hubert.

Inputs:

- `notes[35:0]` are the signals from each of the keys on the keytar.

- `h0[7:0], h1[7:0], h2[7:0]` are the amplitude ratios for the different harmonics. These are unsigned values, and a value of x represents an amplitude ratio of $x/255$.
- `amp[287:0]` contains the amplitudes for all of the requested frequencies, eight bits each.
- `volume[7:0]` is the volume to scale all notes to.

Outputs:

- `signal [7:0]` is the composite signal combining all of the waves corresponding to the keys pressed, plus all of the the appropriate harmonics.
- `freq [539:0]` are the frequencies for the 36 different notes. This is connected directly to the sine wave generator. These are normally hardcoded, but if pitch bending is implemented, these may be changed dynamically.
- `cycle` is a cycle counter, also passed to the sine wave generator, so that it knows the current time.

Instrument Bank

The instrument bank allows the keytar player to select an instrument to simulate at any given time. Different instruments are distinguished only by their harmonic content, so the instrument bank is a hardcoded table to constants indicating what the amplitudes of higher harmonic waves should be. Initially, if the base frequency is f, then the harmonics considered will have frequencies of $1.5f$, $2f$, and $3f$, corresponding to a fifth, an octave, and an octave plus a fifth above the base note. More may be added if it turns out that the sound quality is poor.

The instrument bank will require a hardcoded table of constants. In its operation, it will simply take in an instrument selection and provide the amplitude ratios for the higher harmonics.

The instrument bank will be designed and implemented by Hubert.

Inputs:

- `inst_sel[3:0]` will select one of several possible instruments to synthesize.

Outputs:

- `h0[7:0], h1[7:0], h2[7:0]` will all give amplitude ratios for the higher harmonics.

Sine Wave Generator

The sine wave generator will produce tones at various frequencies by sampling a pregenerated sine wave at different intervals. For example, if the sine table were constructed such that sampling each value would produce a tone at 440Hz, then sampling every other value would produce a tone at 880Hz. The frequency of the stored sine wave will be dependent on the clock speed and will have to be tailored to match the system clock. Additionally, the stored sine wave will be at a low frequency but at a high resolution, so that higher frequencies can be produced by down-sampling without too much loss of quality.

The table will need to be pre-generated. It can be easily precalculated and stored in a ROM.

The sine wave generator will be designed and implemented by Hui.

Inputs:

- `freq[539:0]` are the desired frequencies for each of the thirty-six notes, fifteen bits each.
- `cycle` is the current clock cycle, used to determine what the output should be.

Outputs:

- `amp[287:0]` are the amplitudes corresponding to those frequencies.

Digital Signal Processing Chain

This will actually be a series of modules that are connected in a chain, that will process the base tones to make them more interesting. Each module will have an `enable` input to determine whether to do any processing. If it is low, then the input signal will be sent directly to the output. Otherwise, some sort of processing will be done to alter the signal.

These will likely be fairly simple modules. The reverb module, for example, would need to take a signal from some previous time step and add it to the current signal at a diminished amplitude, which requires only a multiplication and enough storage to hold the signals from all previous steps.

The digital signal processing modules will be designed and implemented by Hubert.

Inputs:

- `enable` is a one-bit signal determining whether to actually process the incoming signal or just pass it through unchanged.
- `in_tone[7:0]` is an incoming signal which is processed in some way.

Outputs:

- `out_tone[7:0]` is the processed tone if `enable` is high, or the input tone if `enable` is low.

Mixer

The mixer is the module at the end of the DSP chain. This will be the module that actually takes the sound data and sends it to the AC97 chip in the appropriate format.

Initially this will only take in the output from the DSP chain, convert it to the AC97 format, and send it to the chip. However, if we decide to implement drums as well, then this will have the additional task of taking in the sound wave from the drum modules and sending the average of the keytar and the drum amplitudes to the AC97.

The mixer will be designed and implemented by Hubert.

All inputs and outputs will be exactly the same as the `ac97` module from lab 4.

External Hardware

This project is really only meaningful if there is a nontrivial user interface. Thus a rudimentary keytar will be built as the user interface for the project.

A keytar has approximately thirty-six keys, or three octaves' worth. Each of these is a switch hooked up to a power supply, attached to a pull-up or pull-down resistor, depending on the kind of switch obtained. When the user presses a key, the switch will toggle and send a different signal to its output. These outputs will simply plug into the many user input ports.

In addition to the keys on the keyboard, there is some sort of mechanism for instrument switching. Preliminary testing will be done using the built-in switches on the labkit, although it would be nice to be able to switch instruments from the keytar by adding switches to it. Volume adjustments would also be handy, requiring an additional two buttons for volume up and volume down. Finally, DSP controls for each implemented DSP module would be useful, although it is possible for these to be placed separately.

The total is about four switches and forty push switches, all of which need to be appropriately connected to power and to the labkit, though the user input ports.

The hardware will be designed and implemented by Hui.

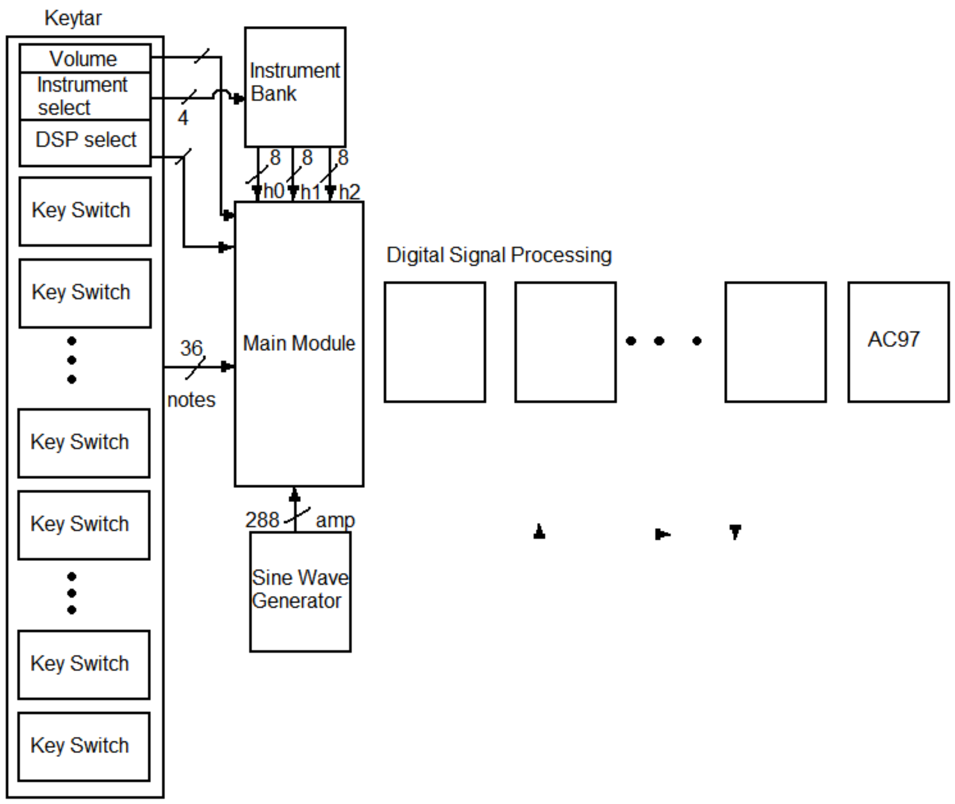

Figure 1: Block diagram for the keytar handler